……..o0o……..

Giáo trình
Cấu trúc dữ liệu 1

Chương 1:

TỔNG QUAN

1. VAI TRÒ CỦA CẤU TRÚC DỮ LIỆU TRONG MỘT ĐỀ ÁN TIN HỌC

Thực hiện một đề án tin học là chuyển bài toán thực tế thành bài toán có thể giải quyết trên máy tính. Một bài toán thực tế bất kỳ đều bao gồm các đối tượng dữ liệu và các yêu cầu xử lý trên những đối tượng đó. Vì thế, để xây dựng một mô hình tin học phản ánh được bài toán thực tế cần chú trọng đến hai vấn đề :

- Tổ chức biểu diễn các đối tượng thực tế : Các thành phần dữ liệu thực tế đa dạng, phong phú và thường chứa đựng những quan hệ nào đó với nhau, do đó trong mô hình tin học của bài toán, cần phải tổ chức , xây dựng các cấu trúc thích hợp nhất sao cho vừa có thể phản ánh chính xác các dữ liệu thực tế này, vừa có thể dễ dàng dùng máy tính để xử lý. Công việc này được gọi là xây dựng *cấu trúc dữ liệu* cho bài toán.
- Xây dựng các thao tác xử lý dữ liệu: Từ những yêu cầu xử lý thực tế, cần tìm ra các giải thuật tương ứng để xác định trình tự các thao tác máy tính phải thi hành để cho ra kết quả mong muốn, đây là bước xây dựng *giải thuật* cho bài toán.

Tuy nhiên khi giải quyết một bài toán trên máy tính, chúng ta thường có khuynh hướng chỉ chú trọng đến việc xây dựng giải thuật mà quên đi tầm quan trọng của việc tổ chức dữ liệu trong bài toán. Giải thuật phản ánh các phép xử lý , còn đối tượng xử lý của giải thuật lại là dữ liệu, chính dữ liệu chứa đựng các thông tin cần thiết để thực hiện giải thuật. Để xác định được giải thuật phù hợp cần phải biết nó tác động đến loại dữ liệu nào và khi chọn lựa cấu trúc dữ liệu cũng cần phải hiểu rõ những thao tác nào sẽ tác động đến nó. Như vậy trong một đề án tin học, giải thuật và cấu trúc dữ liệu có mối quan hệ chặt chẽ với nhau, được thể hiện qua công thức:

Cấu trúc dữ liệu + Giải thuật = Chương trình

Với một cấu trúc dữ liệu đã chọn, sẽ có những giải thuật tương ứng, phù hợp. Khi cấu trúc dữ liệu thay đổi thường giải thuật cũng phải thay đổi theo để tránh việc xử lý gượng ép, thiếu tự nhiên trên một cấu trúc không phù hợp. Hơn nữa, một cấu trúc dữ liệu tốt sẽ giúp giải thuật xử lý trên đó có thể phát huy tác dụng tốt hơn, vừa đáp ứng nhanh vừa tiết kiệm vật tư, giải thuật cũng dễ hiểu và đơn giản hơn.

2. CÁC TIÊU CHUẨN ĐÁNH GIÁ CẤU TRÚC DỮ LIỆU

Một cấu trúc dữ liệu tốt phải thỏa mãn các tiêu chuẩn sau:

- Phản ánh đúng thực tế: Đây là tiêu chuẩn quan trọng nhất, quyết định tính đúng đắn của toàn bộ bài toán. Cần xem xét kỹ lưỡng cũng như dự trù các trạng thái biến đổi của dữ liệu trong chu trình sống để có thể chọn cấu trúc dữ liệu lưu trữ thể hiện chính xác đối tượng thực tế.
- Phù hợp với các thao tác trên đó: Tiêu chuẩn này giúp tăng tính hiệu quả của đề án: việc phát triển các thuật toán đơn giản, tự nhiên hơn; chương trình đạt hiệu quả cao hơn về tốc độ xử lý.
- Tiết kiệm tài nguyên hệ thống: Cấu trúc dữ liệu chỉ nên sử dụng tài nguyên hệ thống vừa đủ để đảm nhiệm được chức năng của nó.Thông thường có 2 loại tài nguyên cần lưu tâm nhất : CPU và bộ nhớ. Tiêu chuẩn này nên cân nhắc tùy vào tình huống cụ thể khi thực hiện đề án . Nếu tổ chức sử dụng đề án cần có những

xử lý nhanh thì khi chọn cấu trúc dữ liệu yếu tố tiết kiệm thời gian xử lý phải đặt nặng hơn tiêu chuẩn sử dụng tối ưu bộ nhớ, và ngược lại.

3. TRỪU TƯỢNG HOÁ DỮ LIỆU

Trừu tượng hoá là ý niệm về sự vật hay hiện tượng sau khi thu thập chắt lọc những thông tin có ý nghĩa; và loại bỏ đi những thông tin không cần thiết hoặc những chi tiết không quan trọng. Thông tin bao gồm các trạng thái tĩnh(data) và các tác vụ(operation) lên dữ liệu đó.

Sự trừu tượng hoá bao hàm trừu tượng hoá dữ liệu để thu thập thông tin về dữ liệu và trừu tượng hoá tác vụ để thu tập các tác vụ liên quan. Kết quả của quá trình trừu tượng hoá giúp chúng ta xây dựng một mô hình cho một kiểu dữ liệu mới gọi là kiểu dữ liệu trừu tượng(Abstract Data Type - ADT), mỗi kiểu dữ liệu trừu tượng có mô tả dữ liệu và các tác vụ liên quan.

Ví dụ: mô tả kiểu dữ liệu trừu tượng về số hữu tỉ a/b với các tác vụ cộng hai số hữu tỉ, nhân hai số hữu tỉ, chia hai số hữu tỉ.

KIỂU DỮ LIỆU TRỪU TƯỢNG VỀ SỐ HỮU TỈ

Mô tả dữ liệu:

- Tử số.
- Mẫu số (mẫu số phải khác 0).

Mô tả tác vụ:

- Tác vụ cộng: add(sốhữutỉ1, sốhữutỉ2)
  Nhập:
  a,b là tử và mẫu của sốhữutỉ1
  c,d là tử và mẫu của sốhữutỉ2
  Xuất:
  ad+bc là tử của số hữu tỉ kết quả
  bd là mẫu của số hữu tỉ kết quả.
- Tác vụ nhân: mul(sốhữutỉ1,sốhữutỉ2)
  Nhập: ….
  Xuất: ….

….

4. KIỂU DỮ LIỆU CƠ BẢN

Các loại dữ liệu cơ bản thường là các loại dữ liệu đơn giản, không có cấu trúc. Chúng thường là các giá trị vô hướng như các số nguyên, số thực, các ký tự, các giá trị logic ... Các loại dữ liệu này, do tính thông dụng và đơn giản của mình, thường được các ngôn ngữ lập trình (NNLT) cấp cao xây dựng sẵn như một thành phần của ngôn ngữ để giảm nhẹ công việc cho người lập trình. Chính vì vậy đôi khi người ta còn gọi chúng là các kiểu dữ liệu định sẵn.

Thông thường, các kiểu dữ liệu cơ bản bao gồm :

Kiểu có thứ tự rời rạc: số nguyên, ký tự, logic , liệt kê, miền con …

Kiểu không rời rạc: số thực

Các kiểu dữ liệu định sẵn trong C gồm các kiểu sau:

Tên kiểu	Kthước	Miền giá trị	Ghi chú
char	01 byte	-128 đến 127	Có thể dùng như số nguyên 1

			byte có dấu hoặc kiểu ký tự
unsign char	01 byte	0 đến 255	Số nguyên 1 byte không dấu
int	02 byte	-32738 đến 32767	
unsign int	02 byte	0 đến 65335	Có thể gọi tắt là unsign
long	04 byte	-2^{32} đến 2^{31} -1	
unsign long	04 byte	0 đến 2^{32}-1	
float	04 byte	3.4E-38 … 3.4E38	Giới hạn chỉ trị tuyệt đối.Các giá trị <3.4E-38 được coi = 0. Tuy nhiên kiểu float chỉ có 7 chữ số có nghĩa.
double	08 byte	1.7E-308 … 1.7E308	
long double	10 byte	3.4E-4932… 1.1E4932	

5. KIỂU DỮ LIỆU CÓ CẤU TRÚC

Tuy nhiên trong nhiều trường hợp, chỉ với các kiểu dữ liệu cơ sở không đủ để phản ánh tự nhiên và đầy đủ bản chất của sự vật thực tế, dẫn đến nhu cầu phải xây dựng các kiểu dữ liệu mới dựa trên việc tổ chức, liên kết các thành phần dữ liệu có kiểu dữ liệu đã được định nghĩa. Những kiểu dữ liệu được xây dựng như thế gọi là kiểu dữ liệu có cấu trúc. Đa số các ngôn ngữ lập trình đều cài đặt sẵn một số kiểu có cấu trúc cơ bản như mảng, chuỗi, tập tin, bản ghi...và cung cấp cơ chế cho lập trình viên tự định nghĩa kiểu dữ liệu mới.

Ví dụ : Để mô tả một đối tượng sinh viên, cần quan tâm đến các thông tin sau:
- Mã sinh viên: chuỗi ký tự
- Tên sinh viên: chuỗi ký tự
- Ngày sinh: kiểu ngày tháng
- Nơi sinh: chuỗi ký tự
- Điểm thi: số nguyên

Các kiểu dữ liệu cơ sở cho phép mô tả một số thông tin như :

```
int Diemthi;
```

Các thông tin khác đòi hỏi phải sử dụng các kiểu có cấu trúc như :

```
char masv[15];
char tensv[15];
char noisinh[15];
```

Để thể hiện thông tin về ngày tháng năm sinh cần phải xây dựng một kiểu bản ghi,

```
typedef struct tagDate{
    char ngay;
    char thang;
    char thang;
}Date;
```

Cuối cùng, ta có thể xây dựng kiểu dữ liệu thể hiện thông tin về một sinh viên :

```
typedef struct tagSinhVien{
    char masv[15];
```

```
    char tensv[15];
    char noisinh[15];
    int Diem thi;
}SinhVien;
```

Giả sử đã có cấu trúc phù hợp để lưu trữ một sinh viên, nhưng thực tế lại cần quản lý nhiều sinh viên, lúc đó nảy sinh nhu cầu xây dựng kiểu dữ liệu mới...Mục tiêu của việc nghiên cứu cấu trúc dữ liệu chính là tìm những phương cách thích hợp để tổ chức, liên kết dữ liệu, hình thành các kiểu dữ liệu có cấu trúc từ những kiểu dữ liệu đã được định nghĩa.

6. BÀI TẬP

1. Viết chương trình C khai báo kiểu dữ liệu là mảng một chiều, chương trình có các chức năng như sau:
   - Nhập giá trị vào mảng.
   - Sắp xếp mảng theo thứ tự từ nhỏ đến lớn.
   - Xem nội dung các phần tử trong mảng.
2. Viết chương trình C có khai báo kiểu dữ liệu là mảng hai chiều, chương trình có các chức năng sau:
   - Nhập giá trị vào ma trận.
   - Nhân hai ma trận thành ma trận tích
   - Xem nội dung của các phần tử trong ma trận.
3. Hãy xây dựng và hiện thực kiểu dữ liệu trừu tượng của số hữu tỉ a/b với các tác vụ cộng hai số hữu tỉ, nhân hai số hữu tỉ, chia hai số hữu tỉ.
4. Hãy xây dựng và hiện thực kiểu dữ liệu trừu tượng cho số phức với các tác vụ cộng, trừ, nhân, chia hai số phức.

Chương 2:

DANH SÁCH

Danh sách(list) là một trong những cấu trúc cơ bản nhất được cài đặt trong hầu hết các chương trình ứng dụng. Danh sách là một kiểu dữ liệu trừu tượng có nhiều nút cùng kiểu dữ liệu, các nút trong danh sách có thứ tự.
Có hai cách cài đặt danh sách là cài đặt theo kiểu kế tiếp và cài đặt theo kiểu liên kết. Với cách cài đặt thứ nhất chúng ta có danh sách kề hay còn gọi là danh sách đặc, với cách cài đặt thứ hai chúng ta được danh sách liên kết.

1. MÔ TẢ CẤU TRÚC DANH SÁCH

Mô tả dữ liệu:

Danh sách là một tập hợp các nút cùng kiểu dữ liệu, các nút trong danh sách có thứ tự.

Mô tả các tác vụ:

- Tác vụ initialize:
  Chức năng: khởi động danh sách.
  Dữ liệu nhập: không.
- Tác vụ empty:
  Chức năng: kiểm tra danh sách có bị rỗng không.
  Dữ liệu nhập: không.
  Dữ liệu xuất: TRUE|FALSE
- Tác vụ full:
  Chức năng: kiểm tra danh sách có bị đầy không.
  Dữ liệu nhập: không.
  Dữ liệu xuất: TRUE|FALSE.
- Tác vụ listsize:
  Chức năng: kiểm tra số nút có trong danh sách.
  Dữ liệu nhập: không.
  Dữ liệu xuất: số nút trong danh sách.
- Tác vụ retrieve:
  Chức năng: truy xuất nút tại vị trí position trong danh sách.
  Dữ liệu nhập: pos là vị trí của nút cần truy xuất trong danh sách.
  Điều kiện: 0=<pos<=numnodes - 1 (numnodes là số nút của danh sách)
- Tác vụ insert:
  Chức năng: thêm nút vào vị trí pos của danh sách.
  Dữ liệu nhập: nút mới và vị trí pos (vị trí thêm nút mới).
  Điều kiện: 0=<pos<=numnodes.
  Dữ liệu xuất: không.
- Tác vụ remove:
  Chức năng: Xóa nút tại vị trí pos của danh sách.
  Dữ liệu nhập: pos (vị trí của nút xóa).
  Điều kiện: 0=<pos<=numnodes – 1
  Dữ liệu xuất: nút bị xóa.
- Tác vụ replace:
  Chức năng: thay thế nút tại vị trí pos của danh sách bằng nút khác.

Dữ liệu nhập: nút khác và vị trí thay thế pos.
Điều kiện: 0=<pos<=numnodes-1
Dữ liệu xuất: không

- Tác vụ traverse:
  Chức năng: duyệt tấc cả các nút của danh sách.
  Dữ liệu nhập: không.
  Dữ liệu xuất: không.
- Tác vụ sort:
  Chức năng: sắp xếp lại danh sách theo một khoá sắp xếp.
  Dữ liệu nhập: key (khóa sắp xếp)
  Dữ liệu xuất: không.
- Tác vụ search:
  Chức năng: tìm kiếm một nút trong danh sách theo một khoá tìm kiếm.
  Dữ liệu nhập: key là khóa cần tìm.
  Dữ liệu xuất: TRUE|FALSE và pos. TRUE là tìm thấy key trong danh sách và pos chỉ vị trí tìm thấy.
- Tác vụ copylist:
  Chức năng: copy một danh sách thành 1 danh sách mới giống danh sách cũ.
  Dữ liệu nhập: không.
  Dữ liệu xuất: danh sách mới.
- Tác vụ clearlist:
  Chức năng: xoá danh sách.
  Dữ liệu nhập: không
  Dữ liệu xuất: không.

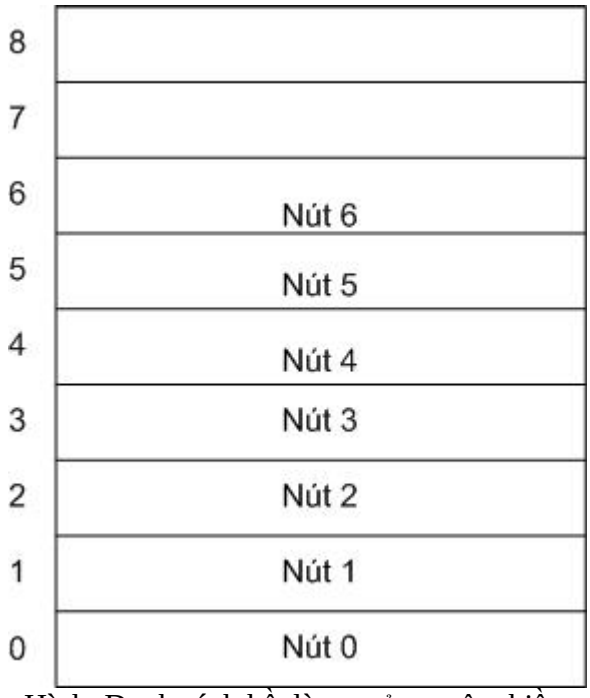
2. PHƯƠNG PHÁP CÀI ĐẶT DANH SÁCH

Có hai cách cài đặt danh sách: cài đặt theo kiểu danh sách kế tiếp và cài đặt theo kiểu danh sách liên kết.

2.1 Cài đặt theo kiểu kế tiếp:

Cài đặt theo kiểu kế tiếp sẽ bố trí các nút trong danh sách liên kết kế cận nhau trong bộ nhớ, cài đặt kiểu này tạo nên danh sách kề. Mảng, chuỗi ký tự, stack hay hàng đợi cài đặt theo kiểu kế tiếp, … là những dạng khác nhau của danh sách kề.

Hình sau đây minh họa danh sách kề dùng mảng 1 chiều, mỗi phần tử trên mạng là một nút của danh sách, danh sách hiện có 7 nút trải dài từ nút 0 đến nút 6 của mảng.

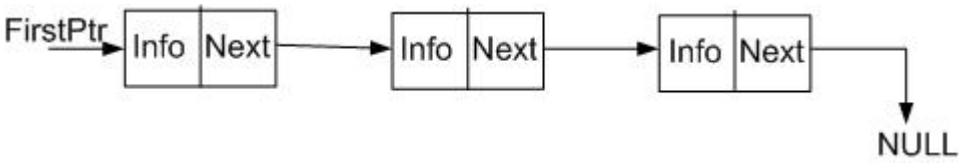


Hình: Danh sách kề dùng mảng một chiều.

2.2 Cài đặt theo kiểu liên kết:

Danh sách được cài đặt theo kiểu liên kết gọi là danh sách liên kết. Mỗi nút trong danh sách có trường info là nội dung của nút và trường next là con trỏ chỉ nút kế tiếp trong danh sách. Con trỏ đầu của danh sách (FirstPtr) chỉ nút đầu tiên, nút cuối cùng của danh sách có trường next trỏ đến vị trí null.

Hình vẽ sau minh hoạ cách cài đặt bằng danh sách liên kết:



Hình: Danh sách liên kết.

2.3 So sánh hai kiểu cài đặt:

Danh sách kề nếu khai báo kích thước danh sách phù hợp thì danh sách kề tối ưu về bộ nhớ vì tại mỗi nút sẽ không cần chứa trường next. Và tốc độ truy xuất phần tử thứ i trong danh sách kề rất nhanh.

Tuy nhiên, về số nút cấp phát cho danh sách kề là cố định nên số nút cần dùng lúc thừa, lúc thiếu. Hơn nữa, danh sách kề bị hạn chế khi thực hiện các tác vụ insert, remove vì mỗi khi thực hiện các tác vụ này chúng ta phải dời chỗ rất nhiều nút. Số nút của danh sách càng lớn thì số lần dời chỗ càng nhiều nên càng chậm.

Số nút cấp phát cho danh sách liên kết thay đổi khi chương trình đang chạy nên việc cấp phát nút cho danh sách liên kết rất linh hoạt: khi nào cần thì cấp phát nút, khi không cần thì giải phóng nút. Danh sách liên kết rất thích hợp khi hiện thực các tác vụ remove, insert vì lúc này chúng ta không phải dời nút mà chỉ sửa lại các vùng liên kết cho phù hợp. Thời gian thực hiện các tác vụ này không phụ thuộc vào số lượng các nút có trong danh sách liên kết.

Tuy nhiên, vì mỗi nút của danh sách liên kết phải chứa thêm trường next nên không sử dụng tối ưu bộ nhớ, việc truy xuất nút thứ i trên danh sách liên kết chập vì phải truy xuất từ đầu danh sách, các tác vụ tìm kiếm trên danh sách liên kết cũng không tối ưu vì thường phải dùng phương pháp tìm kiếm tuyến tính.

3. HIỆN THỰC DANH SÁCH KỀ

3.1 Khai báo cấu trúc của danh sách kề:

Là một mẩu tin có hai trường:

- Trường numnodes: lưu số nút hiện có trong danh sách.
- Trường nodes: là mảng một chiều, mỗi phần tử của mảng là một nút của danh sách.

```
#define MAXLIST 100
typedef struct list{
        int numnodes;
        int nodes[MAXLIST];
};
```

Lưu ý:

- Khi khai báo kích thước mảng (MAXLIST) đủ lớn để có thể chứa hết các nút của danh sách kề.
- Ta có thể khai báo danh sách kề bằng biến cấu trúc ở tầm vực cục bộ hoặc toàn cục.
- Khi danh sách bị rỗng thì không thể hiện thực tác vụ xóa một phần tử ra khỏi danh sách.
- Khi danh sách bị đầy thì không thể thực hiện tác vụ thêm vào.

3.2 Các tác vụ trên danh sách kề

Tác vụ khởi động danh sách:

```
void initialize(struct list *plist){
        plist->numnodes=0;
}
```

Tác vụ xác định số nút của danh sách

```
int listsize(struct list *plist){
        return plist->numnodes;
}
```

Tác vụ kiểm tra danh sách rỗng

```
int empty(struct list *plist){
        if(plist->numnodes==0)
                return TRUE;
        else
                return FALSE;
}
```

Tác vụ kiểm tra danh sách đầy.

```
int full(struct list *plist){
        if(plist->numnodes==MAXLIST)
                return TRUE;
        else
                return FALSE;
}
```

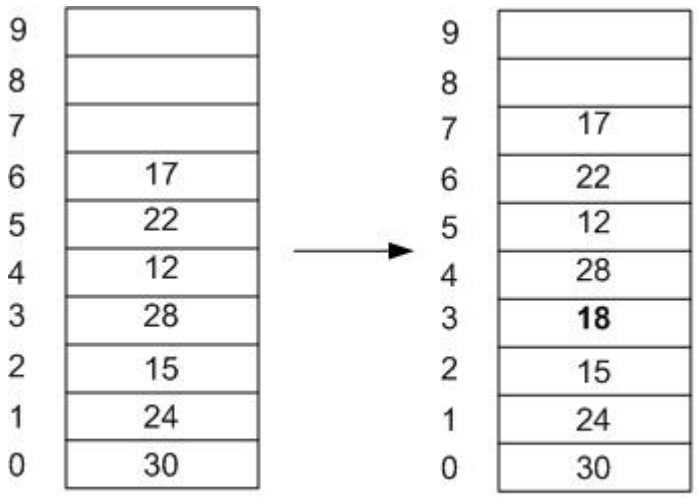
Tác vụ truy xuất một phần tử của danh sách

```
int retrieve(struct list *plist, int pos){
        if(pos<0||pos>=listsize(plist))
                printf(“Vi tri %d khong hop le”,pos);
        else
                if(empty(list))
                        printf(“danh sach bi rong”);
                else
                        return plist->nodes[pos];
}
```

Tác vụ thêm một phần tử mới vào danh sách

```
void insert(struct list *plist, int pos, int x){
        int i;
        if(pos <0 || pos> listsize(plist))
                printf("\n Vi tri chen khong phu hop");
        else{
                if(full(plist)){
                        printf("Danh Sach bi day");
        return;
                }else{
                        for(i=listsize(plist)-1;i>=pos;i--){
                                plist->nodes[i+1]=plist->nodes[i];

                        }
                        plist->nodes[pos]=x;
                        plist->numnodes++;
                }
        }
}
```

Hình vẽ sau miêu tả quá trình thêm một phần tử vào danh sách kề:



Hình: Thêm nút 18 vào vị trí 3 trong danh sách

Tác vụ xoá một phần tử ra khỏi danh sách

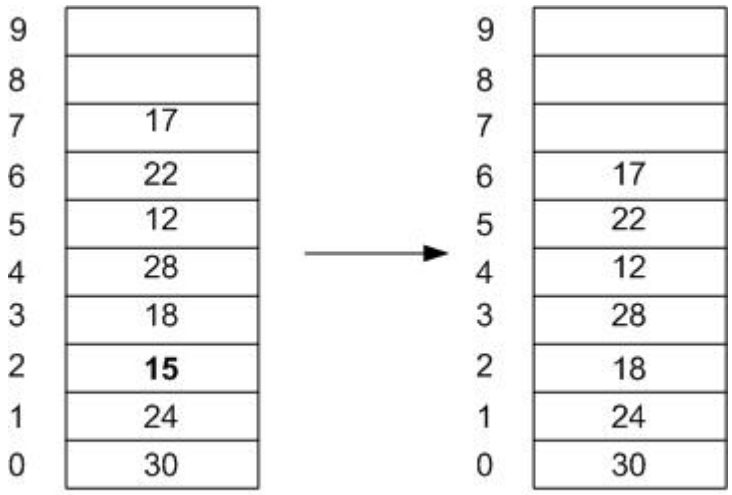
```
int remove(struct list *plist, int pos){
        int i;
        int x;
        if(pos <0||pos>=listsize(plist))
                printf("\n Vi tri xoa khong phu hop");
        else{
                if(empty(plist)){
                        printf("\n Danh sach khong co sinh vien");
                }
                else{
                        x=plist->nodes[pos];
                        for(i=pos;i<listsize(plist)-1;i++){
                                plist->nodes[i]=plist->nodes[i+1];
                        }
                        plist->numnodes--;
                        return x;
                }
        }
        return x;
}
```

Tác vụ thay thế một phần tử của danh sách.

```
void replace(struct list *plist, int pos, int x){
        if(pos<0||pos>=listsize(plist)){
                printf("\n Vi tri hieu chinh khong dung");
                return;
        }else{
                if(empty(plist)){
                        printf("\n Danh sach khong co sinh vien");
                         return;
```

```
            }else
                    plist->nodes[pos]=x;
        }
}
```

Hình vẽ sau miêu tả tác vụ xoá một khoá trong danh sách kề:



Hình: Xóa nút 15 ở vị trí 2 trong danh sách

Tác vụ duyệt danh sách

```
void traverse(struct list *plist){
        int i;
        if(plist->numnodes==0){
                printf("\n Danh sach khong co sinh vien");
                return;
        }
        for(i=0;i<plist->numnodes;i++){
                printf("\n%d", plist->nodes[i]);
        }
}
```

Tác vụ tìm kiếm một phần tử trong danh sách

```
int linearsearch(struct list *plist, int x){
        int vitri=0;
        while(plist->nodes[vitri]!=x && vitri<plist->numnodes)
                vitri++;
        if(vitri==plist->numnodes)
                return -1;
        else
                return vitri;
}
```

Tác vụ sắp xếp các phần tử bên trong danh sách.

```
void selectionsort(struct list *plist){
```

```
        int i,j,vitrimin,min;
        for(i=0;i<plist->numnodes-1;i++){
                min=plist->nodes[i];
                vitrimin=i;
                for(j=i+1;j<plist->numnodes;j++){
                        if(min >plist->nodes[j]){
                                min=plist->nodes[j];
                                vitrimin=j;
                        }
                }
                plist->nodes[vitrimin]=plist->nodes[i];
                plist->nodes[i]=min;
        }
}
```

4. CHƯƠNG TRÌNH MINH HOA

Chương trình sau để quản lý danh sách sinh viên. Chương trình cung cấp các chức năng: xem danh sách sinh viên, thêm một sinh viên vào danh sách, xoá một sinh viên trong danh sách, hiệu chỉnh thông tin về một sinh viên, sắp xếp danh sách sinh viên theo thứ tự, tìm kiếm một sinh viên khi biết mã số sinh viên.

```
//HIEN THUC DANH SACH LIEN KET BANG DANH SACH KE
#include <stdio.h>
#include <conio.h>
#define MAXLIST 100
#define TRUE 1
#define FALSE 0
typedef struct sinhvien{
        int mssv;
        char hoten[20];
};
typedef struct list{
        int numnodes;
        sinhvien nodes[MAXLIST];
};
void initialize(struct list *plist){
        plist->numnodes=0;
}
int listsize(struct list *plist){
        return plist->numnodes;
}
int empty(struct list *plist){
        if(plist->numnodes==0)
                return TRUE;
        else
        return FALSE;
```

```
}
int full(struct list *plist){
	if(plist->numnodes==MAXLIST)
		return TRUE;
	else
	return FALSE;
}
void insert(struct list *plist, int pos, sinhvien x){
	int i;
	if(pos <0 || pos> listsize(plist))
		printf("\n Vi tri chen khong phu hop");
	else{
		if(full(plist)){
			printf("Danh Sach bi day");
	return;
		}else{
			for(i=listsize(plist)-1;i>=pos;i--){
				plist->nodes[i+1]=plist->nodes[i];

			}
			plist->nodes[pos]=x;
			plist->numnodes++;
		}
	}
}

sinhvien remove(struct list *plist, int pos){
	int i;
	sinhvien x;
	if(pos <0||pos>=listsize(plist))
		printf("\n Vi tri xoa khong phu hop");
	else{
		if(empty(plist)){
			printf("\n Danh sach khong co sinh vien");
		}
		else{
			x=plist->nodes[pos];
			for(i=pos;i<listsize(plist)-1;i++){
				plist->nodes[i]=plist->nodes[i+1];
			}
			plist->numnodes--;
			return x;
		}
	}
	return x;
}
```

```
void clearlist(struct list *plist){
	plist->numnodes=0;
}
void replace(struct list *plist, int pos, sinhvien x){
	if(pos<0||pos>=listsize(plist)){
		printf("\n Vi tri hieu chinh khong dung");
		return;
	}else{
		if(empty(plist)){
			printf("\n Danh sach khong co sinh vien");
			 return;
		}else
			plist->nodes[pos]=x;
	}
}
void traverse(struct list *plist){
	int i;
	if(plist->numnodes==0){
		printf("\n Danh sach khong co sinh vien");
		return;
	}
	for(i=0;i<plist->numnodes;i++){
		printf("\n%7d%7d%16s",i, plist->nodes[i].mssv,plist->nodes[i].hoten);
	}
}
void selectionsort(struct list *plist){
	int i,j,vitrimin;
	sinhvien svmin;
	for(i=0;i<plist->numnodes-1;i++){
		svmin=plist->nodes[i];
		vitrimin=i;
		for(j=i+1;j<plist->numnodes;j++){
			if(svmin.mssv >plist->nodes[j].mssv){
				svmin=plist->nodes[j];
				vitrimin=j;
			}
		}
		plist->nodes[vitrimin]=plist->nodes[i];
		plist->nodes[i]=svmin;
	}
}
int linearsearch(struct list *plist, int mssv){
	int vitri=0;
	while(plist->nodes[vitri].mssv !=mssv && vitri<plist->numnodes)
		vitri++;
	if(vitri==plist->numnodes)
```

```
                return -1;
        else
                return vitri;
}

void main(){
        struct list ds;
        sinhvien sv;
        int chucnang,vitri;
        char c;
        initialize(&ds);
        do{
                printf("\n\n CHUONG TRINH QUAN LY DANH SACH HOC SINH:
\n");
                printf("\n Cac chuc nang chinh cua chuong trinh: ");
                printf("\n 1: Xem danh sach sinh vien");
                printf("\n 2: Them mot sinh vien vao danh sach");
                printf("\n 3: Xoa mot sinh vien trong danh sach");
                printf("\n 4: Hieu chinh sinh vien");
                printf("\n 5: Sap xep danh sach theo MSSV");
                printf("\n 6: Tim kiem sinh vien theo MSSV");
                printf("\n 7: Xoa toan bo danh sach");
                printf("\n 0: ket thuc chuong trinh");
                printf("\n Chuc nang ban chon: ");
                scanf("%d",&chucnang);
                switch(chucnang){
                        case 1:{
                                printf("\n Xem danh sach sinh vien");
                                printf("\n       STT   MSSV      HOTEN");
                                traverse(&ds);
                                break;
                        }
                        case 2: {
                                printf("\n Nhap vao vi tri them");
                                scanf("%d",&vitri);
                                printf("Ma so sinh vien: ");
                                scanf("%d",&sv.mssv);
                                printf("Ho va ten SV: ");
                                scanf("%s", &sv.hoten);
                                insert(&ds,vitri,sv);
                                break;
                        }
                        case 3: {
                                printf("\n Vi tri xoa: ");
                                scanf("%d",&vitri);
                                remove(&ds,vitri);
```

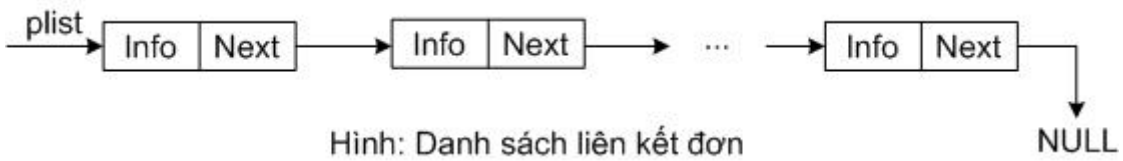
```
                    break;
                }
                case 4:{
                    printf("\n Vi tri hieu chinh: ");
                    scanf("%d",&vitri);
                    printf("Ma so sinh vien moi: ");
                    scanf("%d",&sv.mssv);
                    printf("Ho ten sinh vien moi: ");
                    scanf("%s",&sv.hoten);
                    replace(&ds,vitri,sv);
                    break;
                }
                case 5: {
                    selectionsort(&ds);
                    break;
                }
                case 6:{
                    printf("\n Nhap vao ma so sinh vien can tim: ");
                    scanf("%d",&sv.mssv);
                    vitri=linearsearch(&ds,sv.mssv);
                    if(vitri==-1){
                        printf("\n Khong tim thay sinh vien trong danh sach");
                    }else{
                            printf("\n Tim thay sinh vien trong danh sach");
                    }
                    break;

                }
                case 7:{
                    clearlist(&ds);
                    break;
                }
            }
        }while(chucnang!=0);

}
```

5. HIỆN THỰC DANH SÁCH LIÊN KẾT ĐƠN

5.1 Giới thiệu danh sách liên kết đơn:

Danh sách liên kết đơn là một danh sách có nhiều nút và các nút của nó có thứ tự. Mỗi nút là một cấu trúc có trường info - chứa nội dung thật sự của nút và trường next là con trỏ chỉ nút tiếp theo trong danh sách liên kết. Thứ tự của các nút được thể hiện qua trường next: con trỏ đầu (plist) chỉ nút đầu tiên trong danh sách liên kết, nút đầu chỉ nút thứ hai, …, nút cuối cùng của danh sách liên kết đơn là nút có trường next có giá trị NULL.

Hình vẽ sau đây mô tả một danh sách liên kết đơn:



Hình: Danh sách liên kết đơn

Lưu ý:

- Khi khởi động danh sách liên kết đơn, con trỏ đầu plist được gán giá trị NULL: plist=NULL; danh sách liên kết lúc này bị rỗng.
- Khi danh sách bị rỗng thì không thể thực hiện tác vụ xoá một phần tử, vì vậy trước khi thực hiện tác vụ xoá, chúng ta nên kiểm tra danh sách có bị rỗng hay không.
- Khi duyệt danh sách liên kết đơn nhờ con trỏ đầu plist chúng ta truy xuất được nút đầu và cứ lần theo liên kết có ở từng nút để tuần tự truy xuất đến nút cuối cùng của danh sách liên kết.

5.2 Khai báo cấu trúc danh sách liên kết đơn

Khai báo mỗi cấu trúc là một mẫu tin có hai trường info và trường next:

- Trường info: chứa nội dung của nút.
- Trường next: là con trỏ chỉ nút, dùng để chỉ đến nút kế tiếp trong danh sách liên kết.

```
struct node{
        int info;
        struct node *next;
}
Typedef struct node *NODEPTR;
```

5.3 Các tác vụ trên danh sách liên kết đơn

Tác vụ getnode:

Tác vụ này dùng để cung cấp một biến động làm một nút cho danh sách liên kết.

```
NODEPTR getnode(){
        NODEPTR p;
        p=(NODEPTR)new node;
        return p;
}
```

Tác vụ freenode:

Tác vụ này dùng để giải phóng biến động đã cấp trước đó:

```
void freenode(NODEPTR p){
        delete p;
}
```

Tác vụ initialize:

Tác vụ này dùng để khởi động danh sách liên kết.

```
void initialize(NODEPTR *plist){
        *plist=NULL;
}
```

Tác vụ empty:

Tác vụ này dùng để kiểm tra danh sách có rỗng hay không.

```
int empty(NODEPTR *plist){
```

```
	if(*plist==NULL)
		return TRUE;
	else
		return FALSE;
}
```

Tác vụ nodepointer:

Tác vụ này dùng để xác định con trỏ của nút thứ i trong danh sách liên kết.

```
NODEPTR nodepointer(NODEPTR *plist, int i){
	NODEPTR p;
	int vitri;
	p=*plist;
	vitri=0;
	while(p!=NULL && vitri<i){
		p=p->next;
		vitri++;
	}
	return p;
}
```

Tác vụ position:

Tác vụ này dùng để xác định vị trí của nút p trong danh sách liên kết.

```
int position(NODEPTR *plist, NODEPTR p){
	int vitri;
	NODEPTR q;
	q=*plist;
	vitri=0;
	while(q!=NULL && q!=p){
		q=q->next;
		vitri++;
	}
	if(q==NULL)
		return -1;
	return vitri;
}
```

Tác vụ prenode:

Tác vụ này dùng để xác định nút trước của nút p trong danh sách liên kết.

```
NODEPTR prenode(NODEPTR *plist, NODEPTR p){
	NODEPTR q;
	if(p==*plist)
		return NULL;
	q=*plist;
	while(q!=NULL && q->next !=p)
		q=q->next;
	return q;
}
```

Tác vụ push:

Tác vụ này dùng để thêm một nút có nội dung x vào đầu danh sách liên kết.

```
void push(NODEPTR *plist, int x){
        NODEPTR p;
        p=getnode();
        p->info=x;
        p->next=*plist;
        *plist=p;
}
```

Tác vụ isafter:

Tác vụ này dùng để thêm một nút có nội dung x ngay sau nút p.

```
void insafter (NODEPTR p, int x){
        NODEPTR q;
        if(p!=NULL){
                q=getnode();
                q->info=x;
                q->next=p->next;
                p->next=q;
        }
}
```

Tác vụ pop:

Tác vụ này dùng để xoá nút ở đầu danh sách liên kết, trả về nội dung của nút bị xoá.

```
int pop(NODEPTR *plist){
        NODEPTR p;
        int x;
        if(empty(plist))
                printf("\n danh sach bi rong");
        else{
                p=*plist;
                x=p->info;
                *plist=p->next;
                freenode(p);
                return x;
        }
}
```

Tác vụ delafter:

Tác vụ này dùng để xoá nút ngay sau nút p, trả về nội dung của nút bị xoá

```
int deafter(NODEPTR p){
        NODEPTR q;
        int x;
        if(p==NULL ||p->next==NULL)
                printf("\n Nut khong ton tai");
        else{
                q=p->next;
                x=q->info;
                p->next=q->next;
                freenode(q);
                return x;
```

```
        }
}
```

Tác vụ place:

Tác vụ này dùng để thêm một nút có nội dung x trên danh sách liên kết có thứ tự. Giả sử trường info của các nút có thứ tự tăng dần từ nhỏ đến lớn.

```
void place(NODEPTR *plist, int x){
        NODEPTR p,q;
        q=NULL;
        for(p=*plist;p!=NULL&&x>p->info;p=p->next){
                q=p;
        }
        if(q==NULL)
                push(plist,x);
        else
                insafter(q,x);
}
```

Tác vụ traverse:

Tác vụ này dùng để duyệt danh sách liên kết.

```
void traverse(NODEPTR *plist){
        NODEPTR p;
        p=*plist;
        if(p==NULL)
                printf("\n Danh sach bi rong");
        while(p!=NULL){
                printf("%d",p->info);
                p=p->next;
        }
}
```

Tác vụ search:

Tác vụ này dùng để tìm kiếm nút có nội dung x trên danh sách liên kết bằng phương pháp tìm tuyến tính.

```
NODEPTR search(NODEPTR *plist, int x){
        NODEPTR p;
        p=*plist;
        while(p->info !=x&&p!=NULL)
                p=p->next;
        return p;
}
```

Tác vụ sort:

Tác vụ này dùng để sắp xếp danh sách liên kết theo giá trị tăng dần.

```
void sort(NODEPTR *plist){
        NODEPTR p,q,pmin;
        int min;
        for(p=*plist;p->next!=NULL;p=p->next){
                min=p->info;
                pmin=p;
```

```
                for(q=p->next;q!=NULL;q=q->next){
                        if(min>q->info){
                                min=q->info;
                                pmin=q;
                        }
                }
                pmin->info=p->info;
                p->info=min;
        }
}
```

Tác vụ clearlist:

Tác vụ này dùng để xoá danh sách liên kết bằng cách giải phóng tấc cả các nút có trên danh sách.

```
void clearlist(NODEPTR *plist){
        NODEPTR p,q;
        q=NULL;
        p=*plist;
        while(p!=NULL){
                q=p;
                p=p->next;
                freenode(q);
        }
}
```

6. CHƯƠNG TRÌNH MINH HOẠ

Chương trình sau để quản lý danh sách sinh viên, chương trình được cài đặt bằng biến động.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#define TRUE 1
#define FALSE 0
typedef struct sinhvien{
        int mssv;
        char ten[20];
};
struct node{
        sinhvien info;
        struct node *next;
};
typedef node* NODEPTR;
//tac vu khoi tao mot node moi
NODEPTR getnode(){
        NODEPTR p;
```

```
	p=(NODEPTR) malloc(sizeof(struct node));
	return p;
}
//delete node ra khoi bo nho
void freenode(NODEPTR p){
	free(p);
}
//khoi dau mot danh sach lien ket
void initialize(NODEPTR *plist){
	*plist=NULL;
}
//Kiem tra xem danh sach co empty hay khong
int empty(NODEPTR *plist){
	if(*plist==NULL)
		return TRUE;
	else
		return FALSE;
}
//Them mot node vao dau danh sach
void push(NODEPTR *plist, sinhvien x){
	NODEPTR p;
	p=getnode();
	p->info=x;
	p->next=*plist;
	*plist=p;
}
//Duyet danh sach va in ra noi dung
void traverse(NODEPTR *plist){
	NODEPTR p;
	int stt=0;
	p=*plist;
	if(p==NULL)
		printf("\n Khong co sinh vien trong danh sach");
	while(p!=NULL){
		printf("\n %5d%8d%20s",stt++,p->info.mssv,p->info.ten);
		p=p->next;
	}
}

//Tra ve contro node thu i
NODEPTR nodepointer(NODEPTR *plist, int i){
	NODEPTR p;
	int vitri;
	p=*plist;
	vitri=0;
	while(p!=NULL &&vitri<i) {
```

```
                p=p->next;
                vitri++;
        }
        return p;
}

//Them mot node moi sau node p
void insafter(NODEPTR p, sinhvien x){
        NODEPTR q;
        if(p==NULL)
                printf("Khong them sinh vien vao danh sach duoc");
        else{
                q=getnode();
                q->info=x;
                q->next=p->next;
                p->next=q;
        }
}
//Xoa mot node ngay sau node p
sinhvien delafter(NODEPTR p){
        NODEPTR q;
        sinhvien x;
        if((p==NULL) ||(p->next==NULL))
                printf("\n Khong xoa sinh vien nay duoc");
        else{
                q=p->next;
                x=q->info;
                p->next=q->next;
                freenode(q);
        }
        return x;
}
//Xoa mot node o dau danh sach lien ket
sinhvien pop(NODEPTR *plist){
        NODEPTR p;
        sinhvien x;
        if(empty(plist))
                printf("Khong co sinh vien nao trong danh sach");
        else{
                p=*plist;
                x=p->info;
                *plist=p->next;
                freenode(p);
        }
        return x;
}
```

```
//Xoa toan bo danh sach
void clearlist(NODEPTR *plist){
	NODEPTR p,q;
	q=NULL;
	p=*plist;
	while(p!=NULL){
		q=p;
		p=p->next;
		freenode(q);
	}
	*plist=NULL;
}
//sap xep danh sach theo thu tu tang dan cua ma ssv.
void selectionsort(NODEPTR *plist){
	NODEPTR p,q,pmin;
	sinhvien min;
	for(p=*plist;p->next!=NULL;p=p->next){
		min=p->info;
		pmin=p;
		for(q=p->next;q !=NULL;q=q->next){
			if(min.mssv>q->info.mssv){
				min=q->info;
				pmin=q;
			}
		}
		pmin->info=p->info;
		p->info=min;

	}
}
//tra ve vi tri cua mot node trong danh sach lien ket
int position(NODEPTR *plist, NODEPTR p){
	int vitri;
	NODEPTR q;
	q=*plist;
	vitri=0;
	while(q!=NULL && q!=p){
		q=q->next;
		vitri++;
	}
	if(q==NULL){
		return -1;
	}else{
		return vitri;
	}
}
```

```
//Them sinh vien vao mot danh sach da co thu tu
void place(NODEPTR *plist, sinhvien x){
        NODEPTR p,q;
        q=NULL;
        for(p=*plist;p!=NULL&&x.mssv>p->info.mssv;p=p->next)
                q=p;
        if(q==NULL)
                push(plist,x);
        else
                insafter(q,x);
}
//tim kiem sinh vien dua vao ma so
NODEPTR search(NODEPTR *plist, int x){
        NODEPTR p;
        p=*plist;
        while(p->info.mssv!=x&&p !=NULL)
                p=p->next;
        return p;
}
void main(){
        NODEPTR plist,p;
        sinhvien sv;
        int vitri,chucnang;
        char c;
        initialize(&plist);
        do{
                printf("\n\n CHUONG TRINH QUAN LY DANH SACH SINH VIEN");
                printf("\n\n Cac chuc nang cua chuong trinh");
                printf("\n 1: Xem danh sach sinh vien");
                printf("\n 2: Them sinh vien vao danh sach");
                printf("\n 3: Xoa sinh vien trong danh sach");
                printf("\n 4: Hieu chinh sinh vien");
                printf("\n 5: Sap xep danh sach theo MSSV");
                printf("\n 6: Tim kiem sinh vien theo MSSV");
                printf("\n 7: Them sinh vien vao danh sach da co thu tu");
                printf("\n 8: Xoa toan bo danh sach");
                printf("\n 0: thoat khoi chuong trinh");
                printf("\n\n Chuc nang ban chon: ");
                scanf("%d",&chucnang);
                switch(chucnang){
                        case 1:{
                                printf("\n Danh sach sinh vien: ");
                                printf("\n   STT    MSSV          HOTEN");
                                traverse(&plist);
                                break;
                        }
```

```
		case 2:{
			printf("\n Nhap vao vi tri can them: ");
			scanf("%d",&vitri);
			printf(" Ma so sinh vien: ");
			scanf("%d",&sv.mssv);
			printf(" Ten sinh vien: ");
			//scanf("%s",&sv.ten);
			fflush(stdin);
			gets(sv.ten);
			if(vitri==0)
				push(&plist,sv);
			else{
				p=nodepointer(&plist,vitri-1);
				if(p==NULL)
					printf("Vi tri khong hop le");
				else{
					insafter(p,sv);
				}
			}
			break;
		}
		case 3:{
			printf("\n Nhap vao vi tri can xoa; ");
			scanf("%d",&vitri);
			if(vitri==0){
				pop(&plist);

			}else{
				p=nodepointer(&plist,vitri-1);
				delafter(p);
			}
			break;
		}
		case 4:{
			printf("\n Nhap vao vi tri can hieu chinh: ");
			scanf("%d",&vitri);
			p=nodepointer(&plist,vitri);
			if(p==NULL)
				printf("\n Vi tri khong phu hop");
			else{
				printf("\n STT: %d  MSSV:  %d   TEN: %s",vitri,p-
>info.mssv,p->info.ten);
				printf("\n Ma so sinh vien moi: ");
				scanf("%d",&sv.mssv);
				printf("\n Ten sinh vien moi: ");
				scanf("%s",&sv.ten);
```

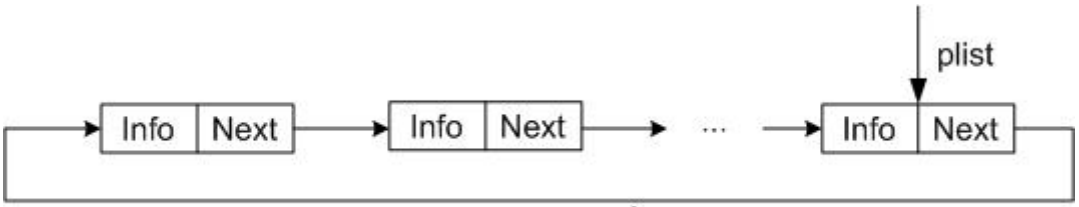
```
                                        p->info=sv;
                                }
                                break;
                        }
                        case 5:{
                                selectionsort(&plist);
                                break;
                        }
                        case 6:{
                                printf("\n Nhap vao ma so sinh vien can tim: ");
                                scanf("%d",&sv.mssv);
                                p=search(&plist,sv.mssv);
                                if(p==NULL)
                                        printf("\n Khong co sinh vien trong danh sach");
                                else
                                        printf("\n Tim thay sinh vien o vi tri %d trong danh
sach",position(&plist,p));
                                break;
                        }
                        case 7:{
                                //Them sinh vien vao mot danh sach da co thu tu
                                printf("\n Ma so sinh vien: ");
                                scanf("%d",&sv.mssv);
                                printf("\n Ten sinh vien: ");
                                scanf("%s",&sv.ten);
                                place(&plist,sv);
                                break;
                        }
                        case 8:{
                                clearlist(&plist);
                                break;
                        }
                }

        }while(chucnang !=0);

}
```

7. CÁC LOẠI DANH SÁCH LIÊN KẾT KHÁC

7.1 Danh sách liên kết vòng

Danh sách liên kết vòng là danh sách liên kết nhưng trường next của nút cuối chỉ nút đầu tiên của danh sách.

Hình vẽ sau đây mô tả danh sách liên kết vòng.



Hình: Danh sách liên kết vòng

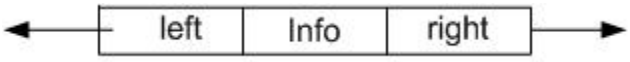
Lưu ý:

- Chúng ta quy ước plist trỏ đến nút cuối của danh sách liên kết vòng.
- Khi khởi động danh sách plist được gán bằng NULL.
- Với danh sách liên kết vòng khi biết con trỏ p của một nút chúng ta có thể truy xuất bất kỳ nút nào trong danh sách bằng cách lần theo vòng liên kết.

7.2 Danh sách liên kết kép

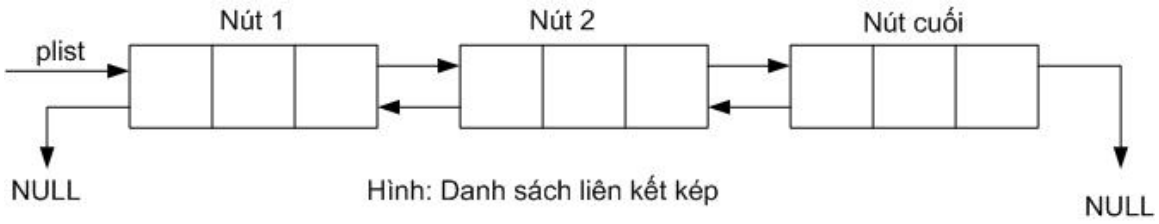
Danh sách liên kết kép là danh sách liên kết mà mỗi nút có hai trường liên kết: một trường liên kết chỉ nút trước (trường left) và một trường liên kết chỉ nút sau (trường right).

Hình ảnh sau mô tả một nút của danh sách liên kết kép.



Hình: Một nút của danh sách liên kết kép

Hình ảnh sau mô tả một danh sách liên kết kép với plist là con trỏ chỉ nút đầu tiên của danh sách liên kết kép.



Hình: Danh sách liên kết kép

Với danh sách liên kết kép chúng ta có thể duyệt danh sách liên kết theo thứ tự xuôi danh sách (lần theo liên kết right) hoặc duyệt ngược danh sách (lần theo liên kết left). Nút cuối của danh sách liên kết có trường right chỉ NULL, nút đầu của danh sách liên kết có trường left chỉ NULL.

8. BÀI TẬP

1. Viết chương trình hiện thực danh sách liên kết kép.
2. Viết 1 hàm giúp xoá nút cuối của danh sách liên kết đơn.
3. Viết 1 hàm nối 2 danh sách liên kết đơn thành 1 danh sách liên kết đơn.
4. Viết 1 hàm để copy một danh sách liên kết thành 1 danh sách liên kết khác giống với nó.
5. So sánh ưu khuyết điểm của danh sách liên kết đơn với danh sách kề.
6. Cài đặt tác vụ copylist để tạo một danh sách mới giống như danh sách cũ.

7. Viết chương trình nhập vào một danh sách liên kết N số nguyên. Xác định có bao nhiêu nút có giá trị x?
8. Viết chương trình nhập vào danh sách liên kết có N số nguyên. Hãy lọc các nút giống nhau ra khỏi danh sách.
9. Viết chương trình hiện thực danh sách liên kết vòng.

Chương 4

CÂY NHỊ PHÂN

Stack, hàng đợi, danh sách là các cấu trúc tuyến tính - các nút trong các cấu trúc này có thứ tự, khi duyệt các cấu trúc này chúng ta duyệt tuần tự từ nút 1, nút 2, … đến nút cuối.

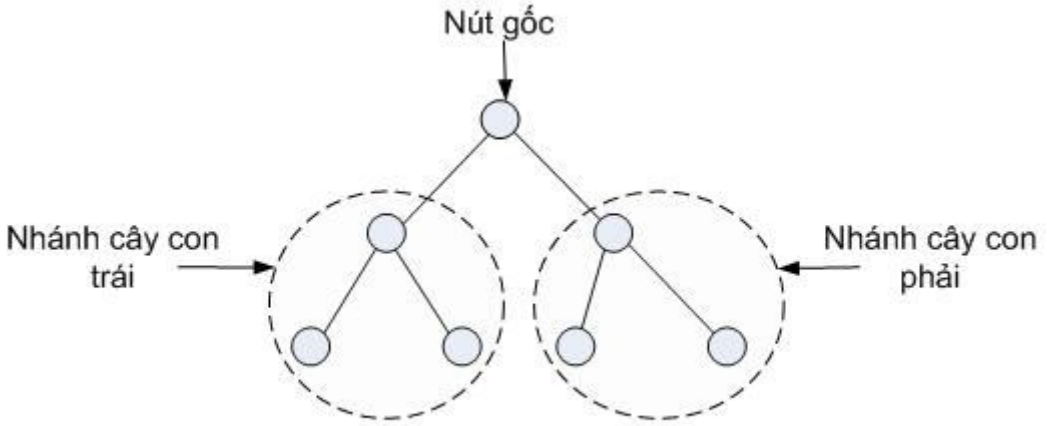
Chương này chúng ta sẽ nghiên cứu một cấu trúc không tuyến tính được sử dụng rất phổ biến là cây nhị phân. Các nút trên cây nhị phân không có thứ tự, mỗi cây nhị phân có một nút gốc, có nhánh cây con bên trái và nhánh cây con bên phải; mỗi nhánh cây con lại tự thân hình thành một cây nhị phân cũng có nút gốc và hai nhánh cây con riêng. Người ta gọi cây nhị phân là cây bậc 2, vì mỗi nút trên cây có tối đa hai nhánh cây con. Cây nhiều nhánh là cây có bậc lớn hơn 2, mỗi nút trên cây nhiều nhánh có thể có nhiều hơn 2 nhánh cây con. Cây nhiều nhánh sẽ được xem xét ở chương sau.

1. CÂY NHỊ PHÂN TỔNG QUÁT

1.1 Định nghĩa

Cây nhị phân là một cấu trúc gồm một tập hữu hạn các nút cùng kiểu dữ liệu (tập nút này có thể rỗng) và được phân thành 3 tập con:

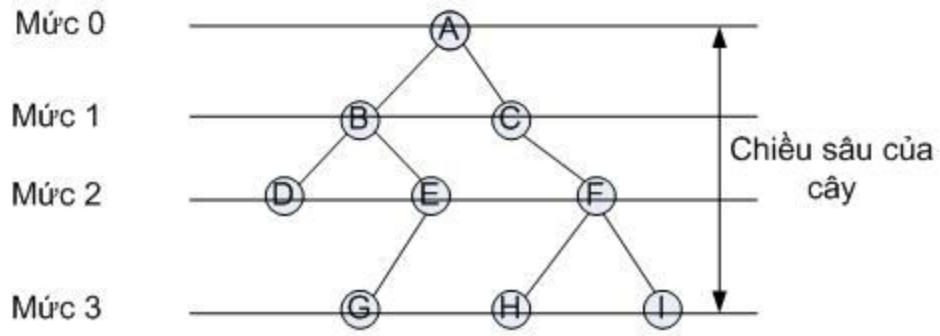
- Tập con thứ nhất có một nút gọi là nút gốc (root)
- Hai tập con còn lại tự thân hình thành hai cây nhị phân là nhánh cây con bên trái (left subtree) và nhánh cây con bên phải (right subtree) của nút gốc. Nhánh cây con bên trái hoặc bên phải cũng có thể là cây rỗng.



Hình vẽ mô tả cây nhị phân

1.2 Các khái niệm cơ bản về cây nhị phân

Chúng ta dùng cây nhị phân như hình sau để mô tả các khái niệm trên cây nhị phân:



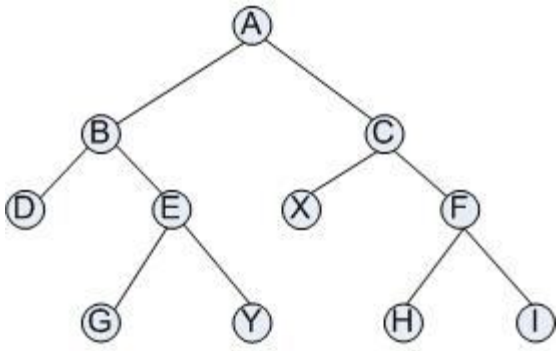
Hình: Khái niệm trên cây nhị phân

- Nút gốc (root): là nút đầu tiên của cây, hình vẽ trên có A là nút gốc.
- Nút cha (father), nút con bên trái (left son), nút con bên phải (right son): nút A là cha của nút B và C. Nút B là nút con bên trái của A, nút C là nút con bên phải của A.
- Nút lá (leaf): là nút không có con, ví dụ các nút D, G, H và I là các nút lá.
- Nút trung gian (internal node): Nút trung gian là nút ở giữa cây nhị phân, nó không là nút lá cũng không phải là nút gốc. Ví dụ các nút B, C, E và F là những nút trung gian.
- Nút trước (ancestor): Nút x gọi là nút trước của nút y nếu cây con nút gốc x có chứa nút y. Ví dụ C là nút trước của nút H.
- Nút sau bên trái (left descendant), nút sau bên phải (right descendant): nút y được gọi là nút sau bên trái của x nếu như cây con bên trái của x có chứa nút y. Tương tự nút y được gọi là nút sau bên phải của nút x nếu cây con bên phải của nút x chứa y.
- Nút anh em (brothers): Hai nút gọi là anh em với nhau nếu chúng là nút con bên trái và nút con bên phải của cùng một nút cha.
- Bậc của cây (degree of tree): Bậc của cây là số cây con tối đa của một nút trên cây. Cây nhị phân là cây có bậc là 2, cây nhiều nhánh là cây có bậc lớn hơn 2.
- Bậc của nút (degree of node): Bậc của nút là số nút con của nút đó. Với cây nhị phân bậc của nút có 1 trong ba giá trị: 0, 1, 2. Ví dụ nút A có bậc của nút là 2, nút E có bậc của nút là 1, nút D có bậc của nút là 0.
- Mức của nút (level of node): Mức của một nút trên cây được định nghĩa như sau:
  Mức của nút gốc là 0.
  Mức của nút khác trong cây nhị phân bằng mức của nút cha + 1.
- Chiều sâu của cây nhị phân (depth of tree): là mức lớn nhất của nút lá trên cây. Chiều sâu chính là đường đi dài nhất từ nút gốc đến nút lá.
- Đường đi, chiều dài của đường đi: đường đi là đoạn đường đi từ nút trước đến nút sau. Chiều dài của đường đi = mức của nút sau - mức của nút trước.

1.3 Các cây nhị phân đặc biệt

1.3.1 Cây nhị phân đúng (strictly binary tree)

Một cây nhị phân gọi là cây nhị phân đúng nếu nút gốc và tất cả các nút trung gian đều có hai nút con. Nếu cây nhị phân đúng có n nút lá thì cây này sẽ có tất cả 2n - 1 nút.
Hình vẽ sau đây miêu tả cây nhị phân đúng:



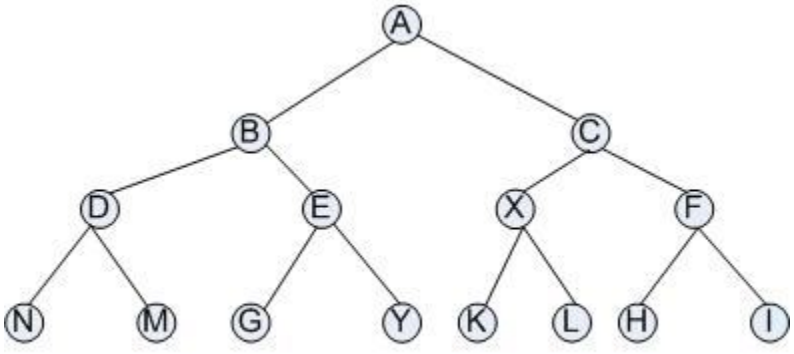
Hình: Cây nhị phân đúng

1.3.2 Cây nhị phân đầy (complete binary tree)

Một cây nhị phân được gọi là cây nhị phân đầy với chiều sâu d thì:

- Trước tiên nó phải là cây nhị phân đúng.
- Tất cả các nút lá đều có mức là d.

Cây nhị phân đầy là cây nhị phân có số nút tối đa ở mỗi mức.



Hình: Cây nhị phân đầy

1.4 Mô tả cây nhị phân

1.4.1 Mô tả dữ liệu

Cây nhị phân là một cấu trúc gồm một tập hữu hạn các nút cùng kiểu dữ liệu và các nút này được phân thành 3 tập con như sau:

- Tập con thứ nhất chỉ có một nút gọi là nút gốc.
- Hai tập con còn lại tự thân hình thành hai cây nhị phân là nhánh cây con bên trái và nhánh của cây con bên phải của nút gốc. Nhánh cây con bên trái hoặc bên phải có thể rỗng.

1.4.2 Mô tả tác vụ

- Tác vụ initialize
  Chức năng: khởi động cây nhị phân.
  Dữ liệu nhập: không.
- Tác vụ empty
  Chức năng: Kiểm tra cây có rỗng hay không.
  Dữ liệu nhập: Không
  Dữ liệu xuất: TRUE|FALSE.
- Tác vụ makenode
  Chức năng: Cung cấp một nút mới cho cây nhị phân.

Dữ liệu nhập: nội dung của nút mới x.
Dữ liệu xuất: Con trỏ chỉ đến nút vừa mới cấp phát.

- Tác vụ setleft
  Chức năng: tạo một nút con bên trái (nút lá) của nút p.
  Dữ liệu nhập: Con trỏ chỉ nút p và nội dung của nút x.
  Điều kiện: nút p chưa có nút con bên trái.
  Dữ liệu xuất: không.

- Tác vụ setright
  Chức năng: tạo nút con bên phải (nút lá) của nút p.
  Dữ liệu nhập: Con trỏ chỉ nút p và nội dung của nút x.
  Điều kiện: Nút p chưa có nút con bên phải.
  Dữ liệu xuất: không.

- Tác vụ delleft
  Chức năng: xoá nút con bên trái (nút lá) của nút p.
  Dữ liệu nhập: con trỏ chỉ nút p.
  Điều kiện: nút con trái của nút p là nút lá.
  Dữ liệu xuất: nút bị xoá.
- Tác vụ delright
  Chức năng: xoá nút con bên phải (nút lá) của nút p.
  Dữ liệu nhập: con trỏ chỉ nút p.
  Điều kiện: nút con phải của nút p là nút lá.
  Dữ liệu xuất: nút bị xoá.

- Tác vụ pretrav
  Chức năng: duyệt cây theo thứ tự trước (NLR).
  Dữ liệu vào: không.
  Dữ liệu ra: Không.

- Tác vụ intrav
  Chức năng: duyệt cây theo thứ tự giữa (LNR)
  Dữ liệu vào: Không.
  Dữ liệu ra: Không.

- Tác vụ posttrav
  Chức năng: duyệt cây theo thứ tự sau (LRN)
  Dữ liệu vào: Không.
  Dữ liệu ra: Không.

- Tác vụ search
  Chức năng: tìm kiếm nút trong cây nhị phân theo một khoá tìm kiếm.
  Dữ liệu nhập: khoá tìm kiếm.
  Dữ liệu xuất: con trỏ chỉ nút tìm thấy.

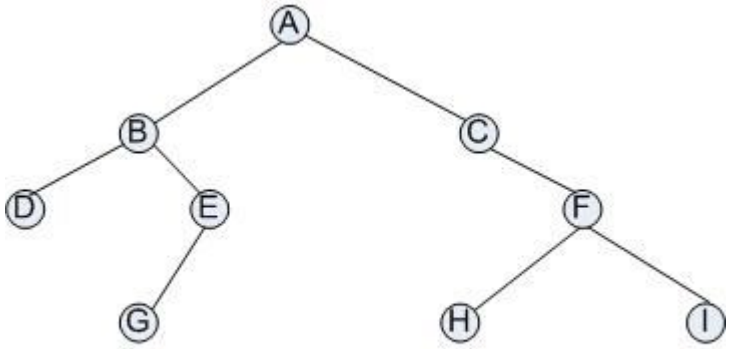
- Tác vụ cleartree
  Chức năng: dùng để xoá cây nhị phân.

1.5 Ba phép duyệt cây nhị phân

Có ba phéo duyệt cây nhị phân:

- Pretrav: duyệt cây nhị phân theo thứ tự trước (NLR- Node Left Right). Đầu tiên thăm nút gốc, sau đó đến duyệt cây con bên trái, sau đó duyệt cây con bên phải.
- Intrav: duyệt cây theo thứ tự giữa (LNR): Đầu tiên duyệt qua nhánh cây con bên trái, sau đó thăm nút gốc, cuối cùng duyệt cây con bên phải.
- Posttrav: Duyệt cây theo thứ tự sau (LRN): Đầu tiên, duyệt nhánh cây con bên trái, sau đó duyệt nhánh cây con bên phải, cuối cùng thăm nút gốc.

Hình vẽ sau đây mô tả ví dụ của ba phép duyệt cây nhị phân:



Hình: Ba phép duyệt cây nhị phân

Nếu duyệt cây trên theo thứ tự NLR thì thứ tự các nút sẽ là: A B D E G C F H I
Nếu duyệt cây trên theo thứ tự LNR thì thứ tự các nút là: D B G E A C H F I
Nếu duyệt cây trên theo thứ tự LRN thì thứ tự các nút là: D G E B H I F C A

1.6 Hiện thực cây nhị phân tổng quát

1.6.1 Khai báo cấu trúc của một nút

Mỗi nút trên cây nhị phân tổng quát là một mẩu tin có các trường như sau:

- Trường info: chứa nội dung của nút.
- Trường left là con trỏ chỉ nút, dùng để chỉ nút con bên trái.
- Trường right là con trỏ chỉ nút, dùng để chỉ nút con bên phải.

```
typedef struct nodetype{
        int info;
        nodetype *left;
        nodetype *right;
};
typedef nodetype *NODEPTR;
```

1.6.2 Hiện thực các tác vụ

- Tác vụ makenode
  Tác vụ này dùng để cấp phát một nút mới.

```
NODEPTR makenode(int x){
        NODEPTR p;
        p=getnode();
        p->info=x;
        p->left=NULL;
        p->right=NULL;
        return p;
}
```

- Tác vụ pretrav

```
void pretrav(NODEPTR proot,int level){
        if(proot !=NULL){
                for(int i=0;i<level;i++)
        printf("-");
                printf("%d\n",proot->info);
                pretrav(proot->left,level+1);
                pretrav(proot->right,level+1);
        }
}
```

- Tác vụ Intrav

```
void intrav(NODEPTR proot){
        if(proot!=NULL){
                intrav(proot->left);
                printf("%4d",proot->info);
      intrav(proot->right);
        }
}
```

- Tác vụ posttrav

```
void posttrav(NODEPTR proot){
        if(proot!=NULL){
                posttrav(proot->left);
                posttrav(proot->right);
                printf("%4d",proot->info);
        }
}
```

- Tác vụ search

```
NODEPTR search(NODEPTR proot,int x){
        NODEPTR p;
        if(proot->info==x)
                return proot;
        if(proot==NULL)
                return NULL;
        p=search(proot->left,x);
```

```
        if(p==NULL)
                p=search(proot->right,x);
    return p;
}
```

- Tác vụ cleartree

```
void cleartree(NODEPTR proot){
        if(proot!=NULL){
                cleartree(proot->left);
                cleartree(proot->right);
                freenode(proot);
        }
}
```

- Tác vụ setleft

```
void setleft(NODEPTR p, int x){
        if(p==NULL)
                printf("\n Nut khong ton tai");
        else
                if(p->left !=NULL)
                        printf("\n Nut p da co con ben trai");
                else
                        p->left=makenode(x);
}
```

- Tác vụ setright

```
void setright(NODEPTR p, int x){
        if(p==NULL)
                printf("\n Nut khong ton tai");
        else
                if(p->right!=NULL)
                        printf("\n Nut da co con ben phai");
                else
        p->right=makenode(x);
}
```

- Tác vụ delleft

```
int delleft(NODEPTR p){
        NODEPTR q;
        int x;
        if(p==NULL){
        printf("\n Nut khong ton tai");
        }
        else{
                q=p->left;
                x=q->info;
                if(q==NULL)
                        printf("\n Nut khong co con ben trai");
```

```
                else{
                        if(q->left!=NULL ||q->right !=NULL)
                                printf("\n Nut khong phai la la");
                        else{
                                p->left=NULL;
                                freenode(q);
                        }
                }
        }
    return x;
}
```

- Tác vụ delright

```
int delright(NODEPTR p){
        NODEPTR q;
        int x;
        if(p==NULL){
        printf("\n Nut khong ton tai");
        }
        else{
                q=p->right;
                x=q->info;
                if(q==NULL)
                        printf("\n Nut khong co con ben phai");
                else{
                        if(q->left!=NULL ||q->right !=NULL)
                                printf("\n Nut khong phai la la");
                        else{
                                p->right=NULL;
                                freenode(q);
                        }
                }
        }
    return x;
}
```

1.7 Chương trình minh hoạ hiện thực cây nhị phân tổng quát

```
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0

//dinh nghia cau truc cho cay nhi phan
typedef struct nodetype{
        int info;
```

```
        nodetype *left;
        nodetype *right;
};

typedef nodetype *NODEPTR;

NODEPTR ptree;
int nodecount;
int chieucao;

NODEPTR getnode(){
        NODEPTR p=(NODEPTR)malloc(sizeof(nodetype));
   return p;
}


void freenode(NODEPTR p){
        free(p);
}

void initialize(){
        nodecount=0;
        ptree=NULL;
}

int empty(){
        if(ptree==NULL)
        return TRUE;
        else
        return FALSE;
}

NODEPTR makenode(int x){
        NODEPTR p;
        p=getnode();
        p->info=x;
        p->left=NULL;
        p->right=NULL;
   return p;
}

//them mot nut moi co noi dung x vao ben trai node p
void setleft(NODEPTR p, int x){
        if(p==NULL)
                printf("\n Nut khong ton tai");
        else
```

```
                if(p->left !=NULL)
                        printf("\n Nut p da co con ben trai");
                else
        p->left=makenode(x);
}

//them mot nut moi co noi dung x vao ben phai node p
void setright(NODEPTR p, int x){
        if(p==NULL)
                printf("\n Nut khong ton tai");
        else
                if(p->right!=NULL)
                        printf("\n Nut da co con ben phai");
                else
        p->right=makenode(x);
}

//Xoa nut la ben trai node p
int delleft(NODEPTR p){
        NODEPTR q;
        int x;
        if(p==NULL){
        printf("\n Nut khong ton tai");
        }
        else{
                q=p->left;
                x=q->info;
                if(q==NULL)
                        printf("\n Nut khong co con ben trai");
                else{
                        if(q->left!=NULL ||q->right !=NULL)
                                printf("\n Nut khong phai la la");
                        else{
                                p->left=NULL;
                                freenode(q);
                        }
                }
        }
   return x;
}

//Xoa node la ben phai node p
int delright(NODEPTR p){
        NODEPTR q;
        int x;
        if(p==NULL){
```

```
        printf("\n Nut khong ton tai");
        }
        else{
                q=p->right;
                x=q->info;
                if(q==NULL)
                        printf("\n Nut khong co con ben phai");
                else{
                        if(q->left!=NULL ||q->right !=NULL)
                                printf("\n Nut khong phai la la");
                        else{
                                p->right=NULL;
                                freenode(q);
                        }
                }
        }
   return x;
}

//Duyet cay theo thu thu NLR
void pretrav(NODEPTR proot,int level){
        if(proot !=NULL){
                for(int i=0;i<level;i++)
        printf("-");
                printf("%d\n",proot->info);
                pretrav(proot->left,level+1);
                pretrav(proot->right,level+1);
        }
}
//Duyet cay theo thu thu LNR
void intrav(NODEPTR proot){
        if(proot!=NULL){
                intrav(proot->left);
                printf("%4d",proot->info);
      intrav(proot->right);
        }
}
//Duyet cay theo thu thu LRN
void posttrav(NODEPTR proot){
        if(proot!=NULL){
                posttrav(proot->left);
                posttrav(proot->right);
                printf("%4d",proot->info);
        }
}
//dem so nut tren cay
```

```
void demnut(NODEPTR proot){
        if(proot!=NULL){
                nodecount++;
                demnut(proot->left);
      demnut(proot->right);
        }
}
//dem so nut la tren cay
void demnutla(NODEPTR proot){
        if(proot!=NULL){
        if(proot->left==NULL &&proot->right==NULL)
                        nodecount++;

                demnutla(proot->left);
      demnutla(proot->right);
        }
}

void tinhchieucao(NODEPTR proot, int level){
        if(proot!=NULL){
                if(level>chieucao)
                        chieucao=level;
                tinhchieucao(proot->left,level+1);
      tinhchieucao(proot->right,level+1);
        }
}

//tim kiem phan tu x tren cay
NODEPTR search(NODEPTR proot,int x){
        NODEPTR p;
        if(proot->info==x)
                return proot;
        if(proot==NULL)
                return NULL;
        p=search(proot->left,x);
        if(p==NULL)
                p=search(proot->right,x);
   return p;
}

//Xoa cay, tra bo nho ve cho he thong
void cleartree(NODEPTR proot){
        if(proot!=NULL){
                cleartree(proot->left);
                cleartree(proot->right);
                freenode(proot);
```

```
        }
}

void main(){
        NODEPTR p;
        int chucnang,noidung,noidung1;

        initialize();
        do{
                printf("\n CAY NHI PHAN");
                printf("\n Cac chuc nang cua chuong trinh: ");
                printf("\n1. Tao nut goc cho cay nhi phan");
                printf("\n2. Them mot nut la ben trai nut cho truoc");
                printf("\n3. Them mot nut la ben phai nut cho truoc");
                printf("\n4. Xoa mot nut la ben trai");
                printf("\n5. Xoa mot nut la ben phai");
                printf("\n6. Duyet cay theo thu tu NLR");
                printf("\n7. Duyet cay theo thu tu LNR");
                printf("\n8. Duyet cay theo thu tu LRN");
                printf("\n9. Tim kiem");
                printf("\n10. Xoa toan bo cay");
                printf("\n11.Dem so nut tren cay");
      //printf("\n12.So nut la tren cay: ");
                //printf("\n0. Ket thuc chuong trinh");

                printf("\n\nChuc nang ban chon: ");
                scanf("%d",&chucnang);

                switch(chucnang){
                        case 1:
                                if(empty()){
                                        printf("\n Nhap vao noi dung cua nut goc: ");
                                        scanf("%d",&noidung);
              ptree=makenode(noidung);
                                }
                                break;
                        case 2:
                                printf("\nNhap vao noi dung cua nut can them: ");
                                scanf("%d",&noidung);
                                p=search(ptree,noidung);
                                if(p!=NULL)
                                        printf("\n Bi trung khoa");
            else{
                                        printf("\n Nhap vao noi dung cua node cha: ");
                                        scanf("%d",&noidung1);
                                        p=search(ptree,noidung1);
```

```
                        if(p==NULL)
                                printf("\n Khong tim thay node cha");
                        else
        setleft(p,noidung);
}
                        break;
                case 3:
                        printf("\n Nhap vao noi dung nut la can them:");
                        scanf("%d",&noidung);
                        p=search(ptree,noidung);
                        if(p!=NULL)
                                printf("\n Bi trung khoa khong them vao duoc");
                        else{
                                printf("\n Nhap vao noi dung nut cha");
                                scanf("%d",&noidung1);
                                p=search(ptree,noidung1);
                                if(p==NULL)
                                        printf("\n Khong tim thay node cha");
                                else
        setright(p,noidung);
                        }
                        break;
                case 4:
                        printf("\nNhap vao noi dung cua node cha: ");
                        scanf("%d",&noidung);
                        p=search(ptree,noidung);
                        if(p==NULL)
                                printf("\n Khong tim thay node cha");
                        else
                                delleft(p);
                        break;
                case 5:
                        printf("\nNhap vao noi dung cua node cha: ");
                        scanf("%d",&noidung);
                        p=search(ptree,noidung);
                        if(p==NULL)
                                printf("\n Khong tim thay node cha");
                        else
                                delright(p);

                        break;
                case 6:
                        printf("\n Duyet cay theo thu tu NLR: \n");
                        pretrav(ptree,0);
                        break;
                case 7:
```

```
                printf("\n Duyet cay theo thu tu LNR");
        intrav(ptree);
                break;
            case 8:
                printf("\n Duyet cay theo thu tu LRN");
        posttrav(ptree);
                break;
            case 9:
                printf("\n Nhap vao noi dung can tim: ");
                scanf("%d",&noidung);
                if(search(ptree,noidung)!=NULL)
                    printf("\n tim thay phan tu %d tren cay",noidung);
                else
            printf("\n Khong tim thay phan tu %d tren cay",noidung);
                break;
            case 10:
                printf("\n Xoa cay");
                cleartree(ptree);
        ptree=NULL;
                break;
            case 11:
                nodecount=0;
                demnut(ptree);
                printf("\nSo nut co tren cay: %d",nodecount);
                break;
            case 12:
            nodecount=0;
                demnutla(ptree);
                printf("\n so nut la tren cay: %d",nodecount);
                break;
            case 13:
                chieucao=-1;
                tinhchieucao(ptree,0);
                printf("\nChieu cao cua cay la: %d",chieucao);
                break;
            }

        }while(chucnang !=0);
        if(ptree!=NULL){
            cleartree(ptree);
    ptree=NULL;
        }
}
```

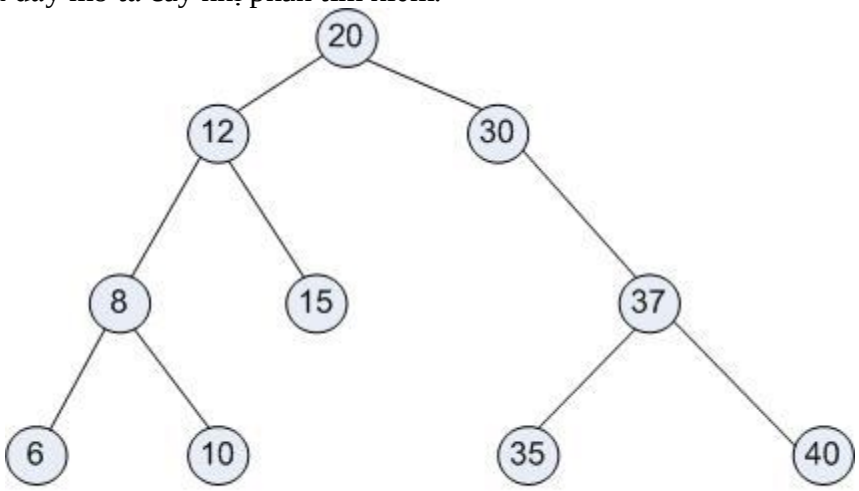
2. CÂY NHỊ PHÂN TÌM KIẾM BST (Binary Search Tree)

2.1 Định nghĩa cây nhị phân tìm kiếm

Cây nhị phân tìm kiếm là cây nhị phân hoặc bị rỗng, hoặc tất cả các nút trên cây có nội dung thoả mãn các điều kiện sau:

- Nội dung của tất cả các nút thuộc nhánh cây con bên trái đều nhỏ hơn nội dung của nút gốc.
- Nội dung của tất cả các nút thuộc nhánh cây con bên phải đều lớn hơn nội dung của nút gốc.
- Cây con bên trái và cây con bên phải tự thân cũng hình thành hai cây nhị phân tìm kiếm.

Hình vẽ sau đây mô tả cây nhị phân tìm kiếm.



Hình: Cây nhị phân tìm kiếm

2.2 Ưu điểm của cây nhị phân tìm kiếm

Trong phần này, ta sẽ so sánh các đặt điểm của cây nhị phân tìm kiếm với danh sách liên kết và danh sách kề dựa trên 2 tiêu chí là việc tìm kiếm dữ liệu và việc cập nhật dữ liệu.

- Với danh sách kề
  Tác vụ thêm nút, xoá nút trên danh sách kề không hiệu quả vì chúng phải dời chổ nhiều lần các nút trong danh sách. Tuy nhiên, nếu danh sách kề là có thứ tự thì tác vụ tìm kiếm trên danh sách thực hiện rất nhanh bằng phương pháp tìm kiếm nhị phân, tốc độ tìm kiếm tỉ lệ với O(log n).
- Với danh sách liên kết
  Tác vụ thêm nút, xoá nút trên danh sách liên kết rất hiệu quả, lúc này chúng ta không phải dời chỗ các nút mà chỉ hiệu chỉnh một vài liên kết cho phù hợp. Nhưng tác vụ tìm kiếm trên danh sách liên kết không hiệu quả vì thường dùng phương pháp tìm kiếm tuyến tính dò từ đầu danh sách. Tốc độ tìm kiếm tỉ lệ với O(n).
- Cây nhị phân tìm kiếm
  Là cấu trúc dung hoà được 2 yếu tố trên: việc thêm nút hay xoá nút trên cây khá thuận lợi và thời gian tìm kiếm khá nhanh. Nếu cây nhị phân tìm kiếm là cân bằng thì thời gian tìm kiếm là O(log n), với n là số phần tử trên cây.

2.3 Cài đặt cây nhị phân tìm kiếm

Cây nhị phân tìm kiếm là một dạng đặc biệt của cây nhị phân nên chúng ta vẫn dùng các tác vụ trong phần trên hiện thực cho cây nhị phân tìm kiếm. Ở phần này chúng ta chỉ xét

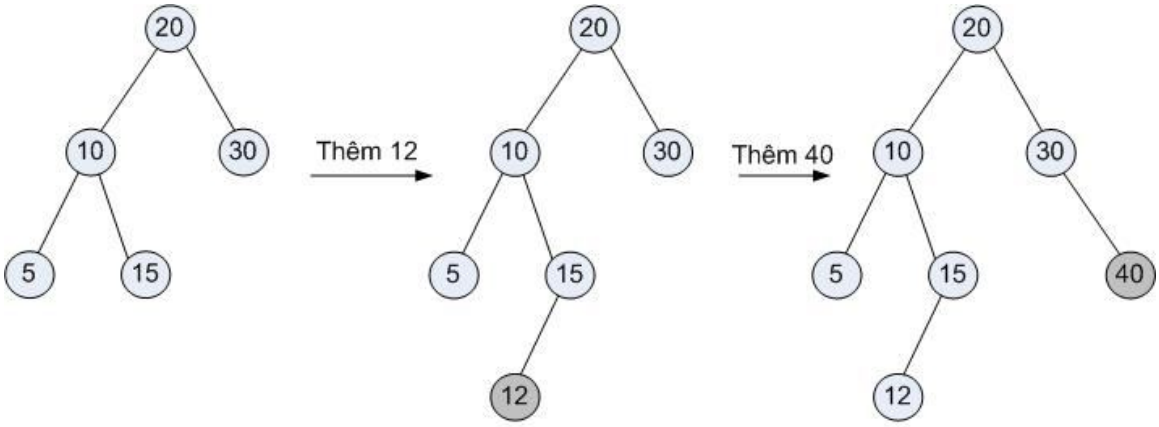
các tác vụ tìm kiếm, thêm vào cây một phần tử và xoá một phần tử ra khỏi cây nhị phân tìm kiếm.

2.3.1 Tác vụ tìm kiếm (Search) trên cây BST

```
NODEPTR search(NODEPTR proot,int x){
        NODEPTR p;
        p=proot;
        if(p!=NULL)
                if(x<proot->info)
                        p=search(proot->left,x);
                else
                        if(x>proot->info)
                                p=search(proot->right,x);
        return p;
}
```

2.3.2 Tác vụ thêm một phần tử vào cây BST

Hình vẽ sau đây mô tả việc thêm 2 nút có nội dung là 12 và 40 vào cây nhị phân tìm kiếm.



Hình minh hoạ tác vụ thêm vào trên cây nhị phân tìm kiếm

Sau đây là hiện thực tác vụ thêm một phần tử vào cây nhị phân tìm kiếm dùng phương pháp đệ qui.

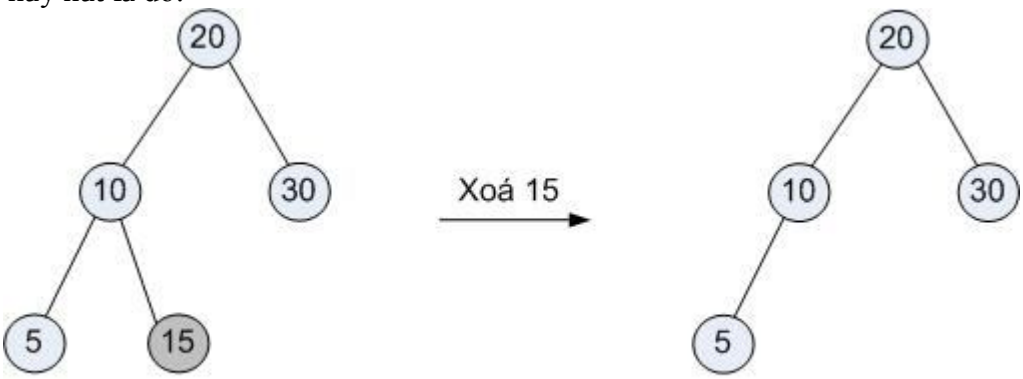
```
void insert(NODEPTR proot, int x){
        if(x==proot->info){
                printf("\n Bi trung noi dung, khong them vao node nay duoc");
                return;
        }
        if(x<proot->info&&proot->left==NULL){
                setleft(proot,x);
                return;
        }
        if(x>proot->info&&proot->right==NULL){
                setright(proot,x);
                return;
        }
```

```
        if(x<proot->info)
                insert(proot->left,x);
        else
                insert(proot->right,x);
}
```

2.3.3 Tác vụ xoá một phần tử ra khỏi cây BST

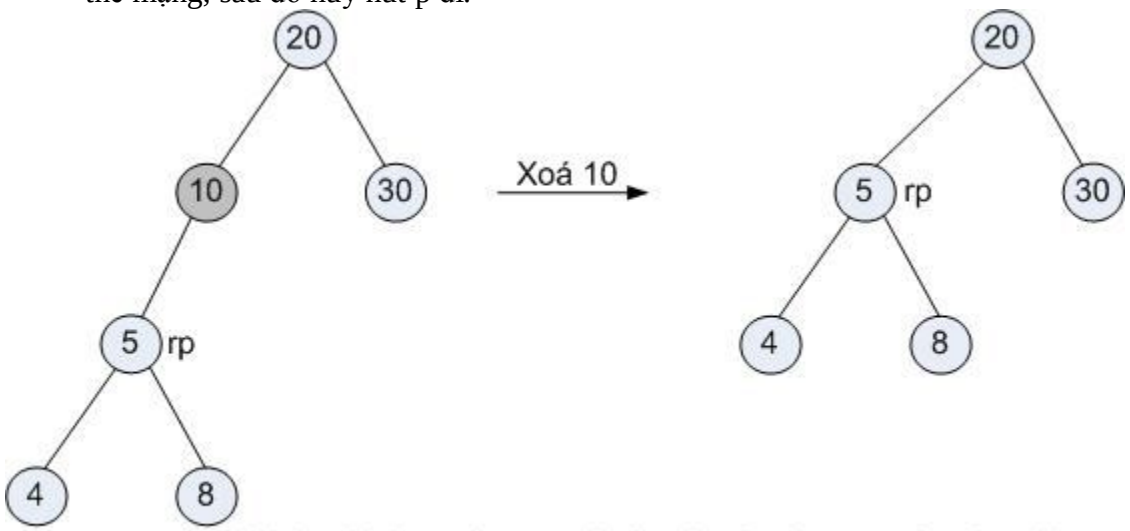
Việc xoá một nút p trong cây nhị phân tìm kiếm khá phức tạp, vì khi đó chúng ta phải điều chỉnh lại cây sao cho nó vẫn là cây nhị phân tìm kiếm. Có 3 trường hợp cần chú ý khi xoá một phần tử trên cây nhị phân tìm kiếm.

- *Trường hợp 1:* Nếu nút p cần xoá là nút lá, việc xoá nút lá chỉ đơn giản là việc huỷ nút lá đó.



Hình minh hoạ tác vụ xoá nút lá trên cây nhị phân tìm kiếm

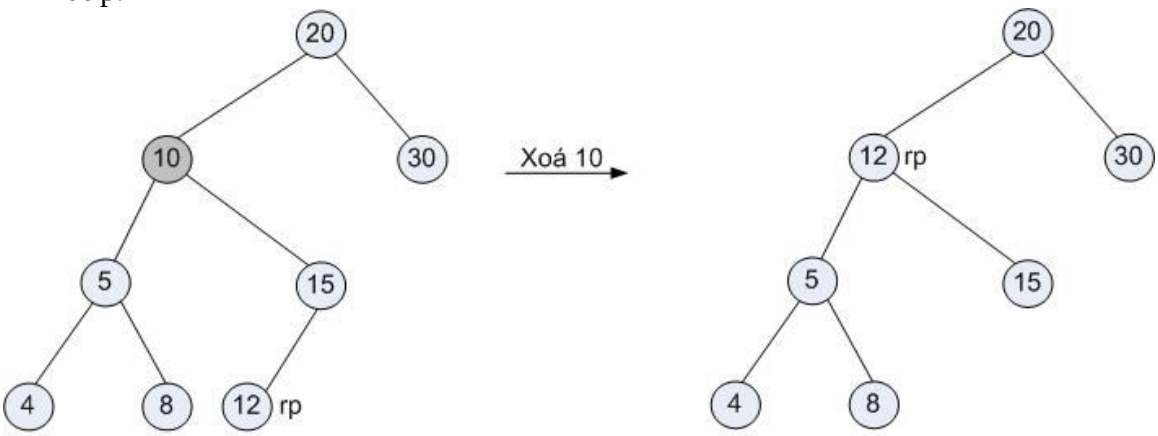
- *Trường hợp 2:* Nếu nút p cần xoá có một cây con, chúng ta chọn nút con của p làm nút thế mạng cho nút p. Chúng ta phải tạo liên kết từ nút cha của p đến nút thế mạng, sau đó huỷ nút p đi.



Hình minh hoạ tác vụ xoá nút có một cây con trên cây nhị phân tìm kiếm

- *Trường hợp 3:* Nếu nút p cần xoá có hai cây con, chúng ta chọn nút có nội dung gần p nhất làm nút thế mạng cho nút p. Nút thế mạng cho nút p có thể là nút trái

nhất của nhánh cây con bên phải nút p hoặc là nút phải nhất của cây con bên trái nút p.



Hình minh hoạ tác vụ xoá nút có hai cây con trên cây nhị phân tìm kiếm

```
NODEPTR remove(NODEPTR p){
        NODEPTR rp,f;
        if(p==NULL){
                printf("\n Nut p khong hien huu, khong xoa nut duoc");
                return NULL;
        }
        else{
                if(p->right==NULL)
                        rp=p->left;
                else{
                        if(p->left==NULL)
                                rp=p->right;
                        else{
                                f=p;
                                rp=p->right;
                                while(rp->left !=NULL){
                                        f=rp;
                                        rp=rp->left;
                                }
                                if(f!=p){
                                        f->left=rp->right;
                                        rp->right=p->right;
                                }
                                 rp->left=p->left;
                        }
                }
                free(p);
                return rp;
        }
}
```

2.4 Chương trình hiện thực cây nhị phân tìm kiếm

```
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0

//dinh nghia cau truc cho cay nhi phan
typedef struct nodetype{
        int info;
        nodetype *left;
   nodetype *right;
};

typedef nodetype *NODEPTR;

NODEPTR ptree;

NODEPTR getnode(){
        NODEPTR p=(NODEPTR)malloc(sizeof(nodetype));
   return p;
}


void freenode(NODEPTR p){
        free(p);
}

void initialize(){
        ptree=NULL;
}

int empty(){
        if(ptree==NULL)
        return TRUE;
        else
        return FALSE;
}

NODEPTR makenode(int x){
        NODEPTR p;
        p=getnode();
        p->info=x;
        p->left=NULL;
        p->right=NULL;
   return p;
```

```
}

//them mot nut moi co noi dung x vao ben trai node p
void setleft(NODEPTR p, int x){
	if(p==NULL)
		printf("\n Nut khong ton tai");
	else
		if(p->left !=NULL)
			printf("\n Nut p da co con ben trai");
		else
	p->left=makenode(x);
}

//them mot nut moi co noi dung x vao ben phai node p
void setright(NODEPTR p, int x){
	if(p==NULL)
		printf("\n Nut khong ton tai");
	else
		if(p->right!=NULL)
			printf("\n Nut da co con ben phai");
		else
	p->right=makenode(x);
}

//Duyet cay theo thu thu NLR
void pretrav(NODEPTR proot,int level){
	if(proot !=NULL){
		for(int i=0;i<level;i++)
	printf("-");
		printf("%d\n",proot->info);
		pretrav(proot->left,level+1);
		pretrav(proot->right,level+1);
	}
}
//Duyet cay theo thu thu LNR
void intrav(NODEPTR proot){
	if(proot!=NULL){
		intrav(proot->left);
		printf("%4d\n",proot->info);
       intrav(proot->right);
	}
}
//Duyet cay theo thu thu LRN
void posttrav(NODEPTR proot){
	if(proot!=NULL){
		posttrav(proot->left);
```

```
                posttrav(proot->right);
                printf("%4d",proot->info);
        }
}

//tim kiem phan tu x tren cay
NODEPTR search(NODEPTR proot,int x){
        NODEPTR p;
        p=proot;
        if(p!=NULL)
                if(x<proot->info)
                        p=search(proot->left,x);
                else
                        if(x>proot->info)
                                p=search(proot->right,x);
   return p;
}

//them mot nut x vao cay BST
void insert(NODEPTR proot, int x){
        if(x==proot->info){
                printf("\n Bi trung noi dung, khong them vao node nay duoc");
      return;
        }
        if(x<proot->info&&proot->left==NULL){
                setleft(proot,x);
      return;
        }
        if(x>proot->info&&proot->right==NULL){
                setright(proot,x);
      return;
        }
        if(x<proot->info)
                insert(proot->left,x);
        else
        insert(proot->right,x);
}

NODEPTR remove(NODEPTR p){
        NODEPTR rp,f;
        if(p==NULL){
                printf("\n Nut p khong hien huu, khong xoa nut duoc");
      return NULL;
   }
        else{
                if(p->right==NULL)
```

```
                rp=p->left;
            else{
                if(p->left==NULL)
                    rp=p->right;
                else{
                    f=p;
                    rp=p->right;
                    while(rp->left !=NULL){
                        f=rp;
        rp=rp->left;
                    }
                    if(f!=p){
                        f->left=rp->right;
        rp->right=p->right;
                    }
      rp->left=p->left;
                }
            }
            free(p);
    return rp;
        }
}

//Xoa cay, tra bo nho ve cho he thong
void cleartree(NODEPTR proot){
    if(proot!=NULL){
        cleartree(proot->left);
        cleartree(proot->right);
        freenode(proot);
    }
}

void main(){
    NODEPTR p;
    int chucnang,noidung,noidung1;

    initialize();
    do{
        printf("\n\n CAY NHI PHAN TIM KIEM");
        printf("\n\n Cac chuc nang cua chuong trinh: ");
        printf("\n1. Them nut cho cay nhi phan");
        printf("\n2. Xoa nut goc");
        printf("\n3. Xoa nut con ben trai");
        printf("\n4. Xoa nut con ben phai");
        printf("\n5. Xoa toan bo cay");
        printf("\n6. Duyet cay theo thu tu NLR");
```

```
printf("\n7. Duyet cay theo thu tu LNR");
printf("\n8. Duyet cay theo thu tu LRN");
printf("\n9. Tim kiem");
printf("\n0. Ket thuc chuong trinh");

printf("\n\nChuc nang ban chon: ");
scanf("%d",&chucnang);

switch(chucnang){
	case 1:
		printf("\n Nhap vao noi dung nut moi: ");
		scanf("%d",&noidung);
		if(ptree==NULL)
			ptree=makenode(noidung);
		else
insert(ptree,noidung);
		break;
	case 2:
		if(ptree ==NULL)
			printf("\nCay bi rong");
		else
ptree=remove(ptree);
		break;
	case 3:
		printf("\n Cho biet noi dung nut cha: ");
		scanf("%d",&noidung);
		p=search(ptree,noidung);
		if(p!=NULL)
			p->left=remove(p->left);
		else
printf("\n Khong tim thay nut cha ");
		break;
	case 4:
		printf("\n Cho biet noi dung nut cha: ");
		scanf("%d",&noidung);
		p=search(ptree,noidung);
		if(p!=NULL){
p->right=remove(p->right);
		}
		else
			printf("\n Khong thay nut cha.");
		break;
	case 5:
		while(ptree)
ptree=remove(ptree);
		break;
```

```
                case 6:
                        printf("\n Duyet cay theo thu tu NLR:\n");
                        pretrav(ptree,0);
                        break;
                case 7:
                        printf("\n Duyet cay theo thu tu LNR");
        intrav(ptree);
                        break;
                case 8:
                        printf("\n Duyet cay theo thu tu LRN");
        posttrav(ptree);
                        break;
                case 9:
                        printf("\n Nhap vao noi dung can tim: ");
                        scanf("%d",&noidung);
                        if(search(ptree,noidung))
                                printf("\n Tim thay");
                        else
            printf("\n Khong tim thay");
                        break;
                }
        }while(chucnang !=0);

        if(ptree!=NULL){
                cleartree(ptree);
      ptree=NULL;
        }
}
```

3. CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG

3.1 Định nghĩa cây nhị phân tìm kiếm cân bằng (AVL Tree)

Người ta dùng cây nhị phân tìm kiếm với mục đích thực hiện tác vụ tìm kiếm cho nhanh, tuy nhiên để tìm kiếm trên cây nhanh thì cây cần phải cân đối. Trường hợp tối ưu nhất là cây nhị phân tìm kiếm hoàn toàn cân bằng (là cây có sự khác biệt của tổng số nút cây con bên trái và tổng số nút của cây con bên phải không quá 1). Tuy nhiên khi thêm vào hoặc xoá nút trên cây rất dễ làm cây mất cân bằng, và chi phí để cân bằng lại cây rất lớn vì phải thao tác trên toàn bộ cây.

Do vậy, người ta tìm cách tổ chức một cây nhị phân tìm kiếm đạt trạng thái cân bằng yếu hơn nhằm làm giảm thiểu chi phí cân bằng khi thêm nút hay xoá nút. Một dạng cây cân bằng là cây nhị phân tìm kiếm cân bằng do hai nhà toán học người Nga là Adelson Velski và Landis xây dựng vào năm 1962 nên còn được gọi là cây AVL. Cây AVL là cây nhị phân tìm kiếm mà tại tất cả các nút của nó chiều sâu của cây con bên phải và chiều sâu của cây con bên trái chênh nhau không quá 1.

Gọi lh(p) và rh(p) là chiều sâu của cây con bên trái và chiều sâu của cây con bên phải của nút p. Có 3 trường hợp có thể xảy ra đối với cây AVL:

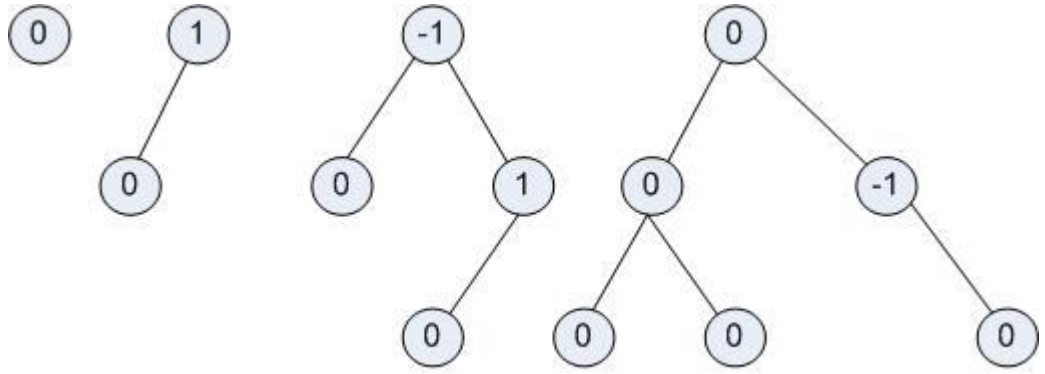
- lh(p)=rh(p): nút p cân bằng.
- lh(p)=rh(p) + 1: nút p bị lệch về bên trái.
- lh(p)=rh(p) – 1: nút p bị lệch về bên phải.

Với cây AVL, khi thêm nút hay xoá nút trên cây có thể làm tăng hay giảm chiều sâu của cây nên có thể làm cây AVL mất cân bằng, khi đó chúng ta phải cân bằng lại cây. Tuy nhiên việc cân bằng lại trên cây AVL chỉ xảy ra ở phạm vi cục bộ bằng cách xoay trái hay xoay phải ở một vài nhánh cây con nên giảm thiểu chi phí cân bằng.

Chỉ số cân bằng (balance factor) bf của một nút trên cây AVL là hiệu của chiều sâu cây con bên trái và chiều sâu cây con bên phải của nút đó.
Xét một nút p bất kỳ trên cây AVL, chỉ số cân bằng của nút p chỉ có thể là một trong 3 giá trị sau:

bf(p)=0:	lh(p)=rh(p)	nút p cân bằng
bf(p)=1:	lh(p)=rh(p) + 1	nút p bị lệch trái
bf(p)=-1:	lh(p)=rh(p) – 1	nút p bị lệch phải.

Hình vẽ sau đây minh hoạ các cây AVL, mỗi nút có một số là chỉ số cân bằng của nút đó.



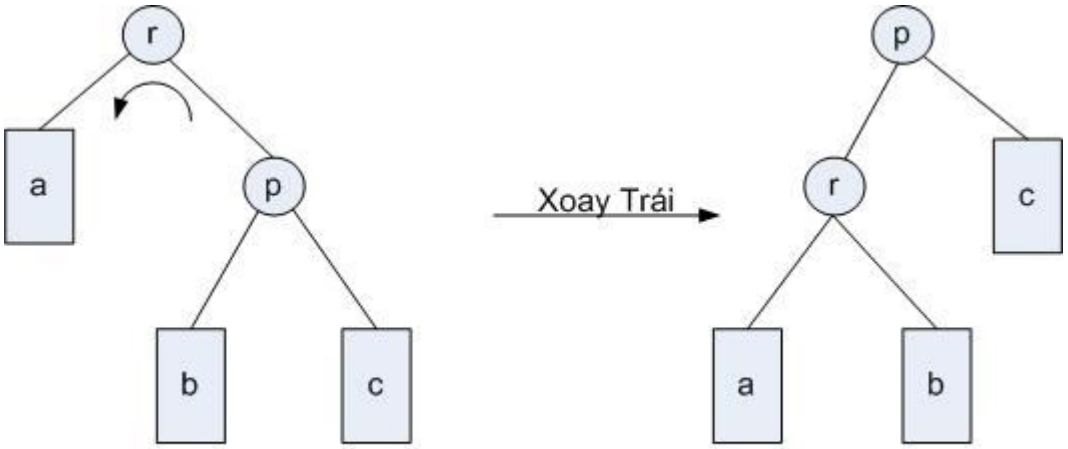
Hình các cây AVL và chỉ số cân bằng của mỗi nút

3.2 Các tác vụ xoay

Khi thêm vào hoặc xoá đi một nút trên cây AVL, cây có thể mất cân bằng. Để cân bằng lại cây, chúng ta sẽ dùng các tác vụ xoay trái và xoay phải. Trong phần này chúng ta sẽ nghiên cứu hai phép xoay trái và xoay phải.

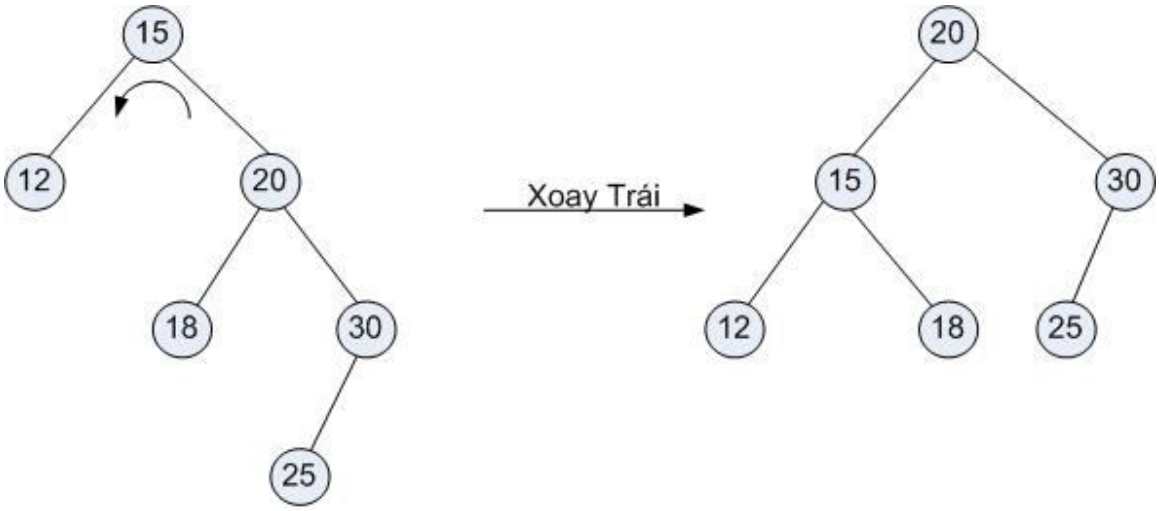
3.2.1 Tác vụ xoay trái (RotateLeft)

Hình vẽ sau mô tả tác vụ xoay trái cây nhị phân tìm kiếm quanh nút r , yêu cầu là nút r phải có nút con bên phải gọi là nút p. Sau khi xoay xong, nút p thành gốc của cây nhị phân tìm kiếm.



Hình xoay trái cây nhị phân quanh gốc là r

Hình vẽ sau minh hoạ cho việc xoay trái quanh cây nhị phân tìm kiếm có nút gốc là 15 như sau:



Hình xoay trái cây nhị phân quanh gốc là 15

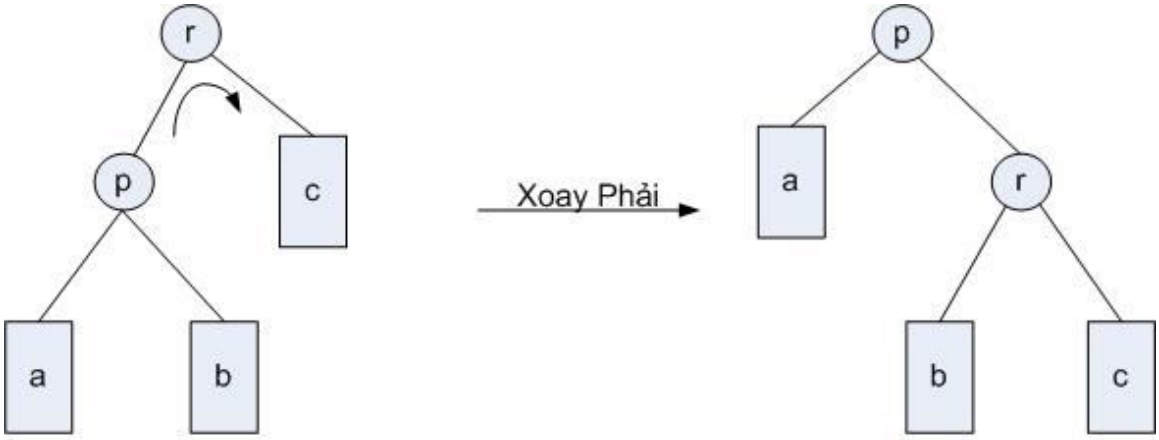
Đoạn code sau sẽ minh hoạ tác vụ xoay trái quanh nút gốc proot, tác vụ này trả về con trỏ chỉ nút gốc mới của nhánh.

```
NODEPTR rotateleft(NODEPTR proot){
        NODEPTR p;
        p=proot;
        if(proot==NULL){
                printf("\n Khong the xoay trai vi cay rong");
        }else{
                if(proot->right==NULL)
                        printf("\n Khong the xoay trai vi khong co node con ben phai");
                else{
                        p=proot->right;
                        proot->right=p->left;
                        p->left=proot;
                }
```

```
        }
        return p;
}
```

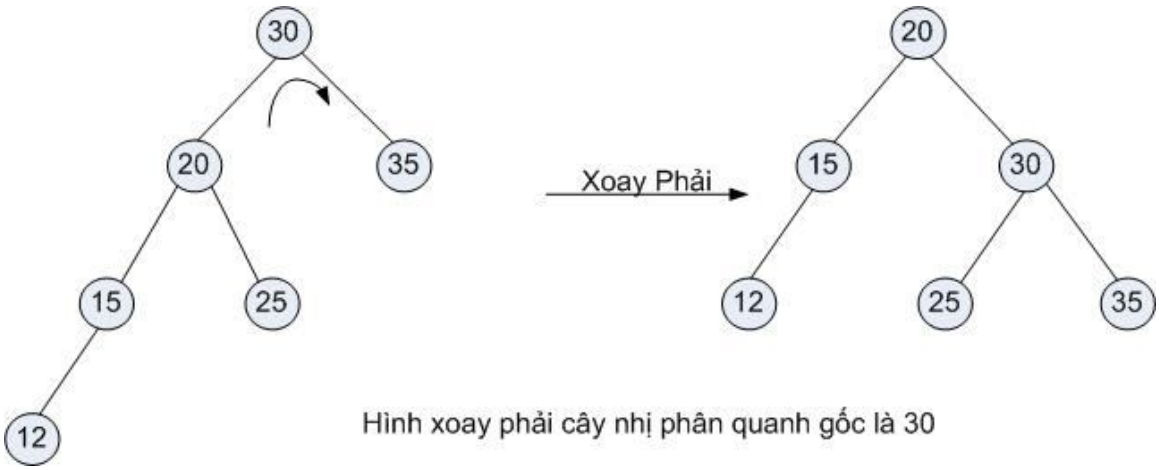
3.2.2 Tác vụ xoay phải (RotateRight)

Hình vẽ sau mô tả tác vụ xoay phải quanh nút gốc r.



Hình xoay phải cây nhị phân quanh gốc là r

Hình vẽ sau minh hoạ cho việc xoay phải quanh nút gốc của cây nhị phân tìm kiếm:



Hình xoay phải cây nhị phân quanh gốc là 30

Đoạn mã sau đây hiện thực tác vụ xoay phải quanh nút proot, tác vụ này trả về con trỏ chỉ nút gốc mới của nhánh.

```
NODEPTR rotateright(NODEPTR proot){
        NODEPTR p;
        p=proot;
        if(proot==NULL)
                printf("\n Khong the xoay phai vi cay bi rong");
        else
                if(proot->left==NULL)
                        printf("\n Khong the xoay phai vi khong co con ben trai");
```

```
        else{
                p=proot->left;
                proot->left=p->right;
    p->right=proot;
        }
    return p;
}
```

3.3 Thêm một nút vào cây AVL

Việc thêm một nút vào cây AVL khá phức tạp, được tiến hành qua các bước sau:

- Trước tiên chúng ta thêm nút vào cây AVL như thêm nút vào cây nhị phân tìm kiếm, nghĩa là nút mới thêm vào sẽ là nút lá ở vị trí thích hợp trên cây.
- Tiếp theo chúng ta tính lại chỉ số cân bằn của các nút có bị ảnh hưởng.
- Sau đó chúng ta xét cây có bị mất cân bằng không, nếu cây bị mất cân bằng thì phải cân bằng lại cây (bằng cách thực hiện các phép xoay phù hợp)

3.3.1 Trường hợp thêm vào làm cây mất cân bằng.

Có hai trường hợp khi thêm nút vào thì cây sẽ bị mất cân bằng.

- Cây sẽ bị mất cân bằng nếu vị trí thêm nút là vị trí sau bên trái của một nút trước gần nhất bị lệch trái.
- Cây sẽ bị mất cân bằng nếu vị trí thêm nút là vị trí sau bên phải của một nút trước gần nhất bị lệch phải.

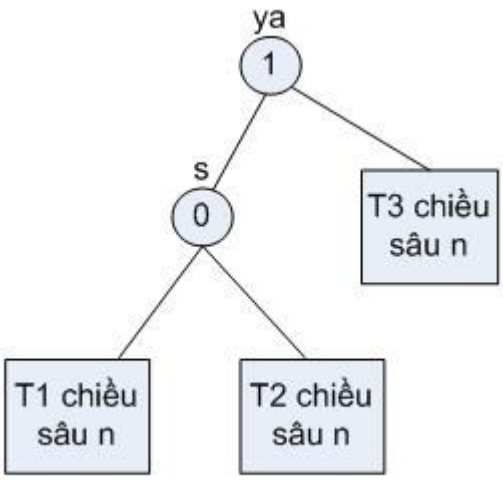
3.3.2 Thực hiện các phép xoay để cân bằng lại cây

Nếu khi thêm nút làm cây bị mất cân bằng, người ta sẽ thực hiện các phép xoay phù hợp để đưa cây trở về trạng thái cân bằng. Gọi ya là nút trước gần nhất bị mất cân bằng khi thêm nút x vào cây AVL, Vấn đề là phải xoay cây như thế nào trong 2 trường hợp sau:

- Trường hợp 1: bf(ya)=1, nút lá thêm vào cây là nút sau bên trái ya.
- Trường hợp 2: bf(ya)=-1, nút lá thêm vào cây là nút sau bên phải ya.

Vì hai trường hợp là tương tự nhau nên ở đây ta chỉ xét trường hợp 1, còn trường hợp 2 được suy ra dựa trên trường hợp 1.

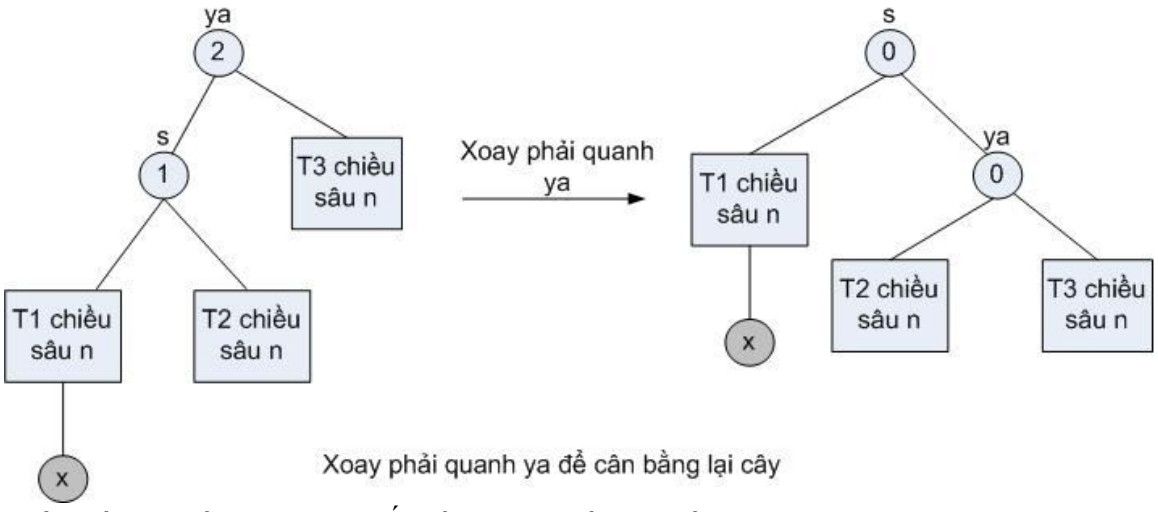
Trong trường hợp 1, xét nhánh cây có nút gốc ya, như hình vẽ:



Hình nhánh cây con nút gốc ya trước khi thêm nút

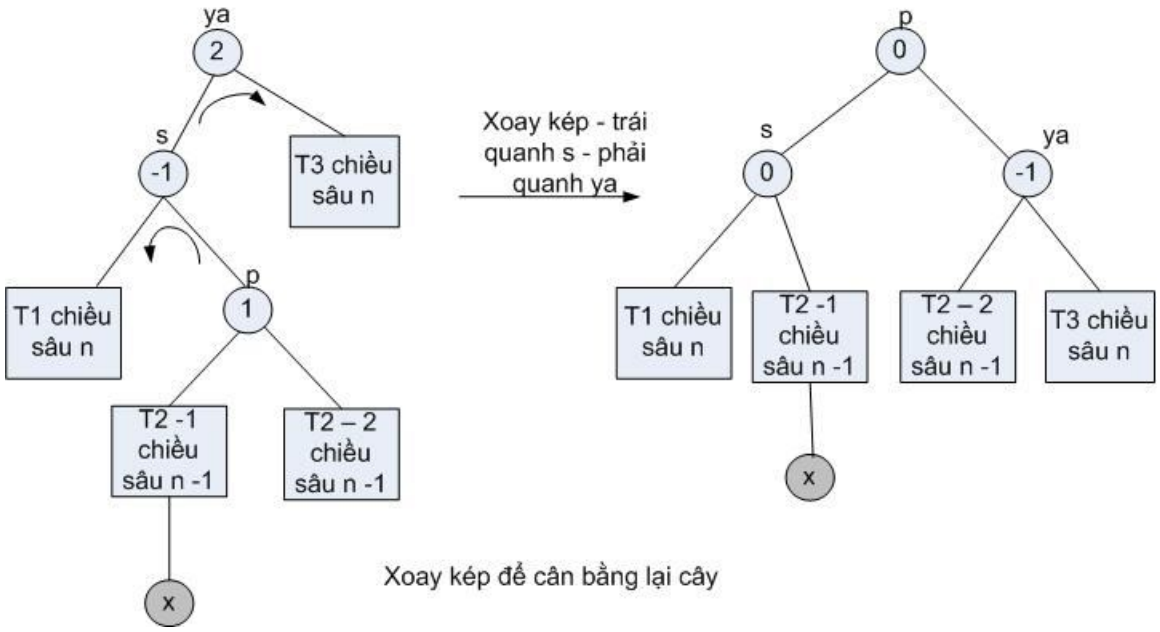
Khi thêm nút x vào nhánh cây con trên, có 2 vị trí thêm phải cân bằng lại là thêm vào nhánh T1 và thêm vào ở nhánh T2.

- Thêm vào ở nhánh T1



Xoay phải quanh ya để cân bằng lại cây

- Thêm vào ở nhánh T2: Phải tiến hành xoay kép như hình vẽ



Trường hợp cây bị lệch phải và thêm một nút vào nhánh cây con bên phải thì làm tương tự như trường hợp trên.

3.4 Cài đặt cây AVL

3.4.1 Khai báo cấu trúc cho cây AVL

Khi khai báo nút của cây AVL, ta cần khai báo thêm một trường bf (balance factor) cho biết chỉ số cân bằng của nút.

```
struct nodetype{
        int info;
        int bf; //balance factor
   struct nodetype *left, *right;
};
typedef struct nodetype *NODEPTR;
```

3.4.2 Các tác vụ của cây AVL

Phần lớn các tác vụ được dùng lại từ những phần trên trừ tác vụ thêm một nút vào cây AVL. Nên phần này ta chỉ xét tác vụ thêm một phần tử vào cây AVL.

```
void insert(NODEPTR *pavltree, int x)
{
        //fp la nut cha cua p, q la con cua p
        //ya la nut truoc gan nhat co the mat can bang, fya la cha cua ya
        //s la nut con cua ya theo huong mat can bang
        //imbal=1: lech trai; =-1 lech phai

        NODEPTR fp,p,q, fya,ya,s;
        int imbal;
```

```
//khoi dong cac gia tri
	fp=NULL;
	p=*pavltree;
	fya=NULL;
	ya=p;

	while(p!=NULL){
		if(x==p->info)
			return;
		if(x<p->info)
			q=p->left;
		if(x>p->info)
			q=p->right;

		if(q!=NULL)
			if(q->bf !=0){  //neu bi mat can bang
				fya=p;
	   ya=q;
			}
		fp=p;
		p=q;
	}

	q=makenode(x);
//them vao mot node la con cua fp
	if(x<fp->info)
		fp->left=q;
	else
		fp->right=q;

//hieu chinh lai chi so can bang cua tac  ca cac node giua ya va q
	if(x<ya->info)
		p=ya->left;
	else
		p=ya->right;

	s=p;
	while(p!=q){
		if(x<p->info){
			p->bf=1;
	p=p->left;
		}else{
			p->bf=-1;
	p=p->right;
		}
	}
```

```
    //xac dinh huong lech
    if(x<ya->info)
            imbal=1;
    else
            imbal=-1;

    if(ya->bf==0){
            ya->bf=imbal;
return;
    }
    if(ya->bf !=imbal){
            ya->bf=0;
return;
    }

    if(s->bf==imbal){
            if(imbal==1){
    p=rotateright(ya);
            }else{
    p=rotateleft(ya);
            }
            ya->bf=0;
s->bf=0;
    }else{
            if(imbal==1){
                    ya->left=rotateleft(s);
    p=rotateright(ya);
            }else{
                    ya->right=rotateright(s);
    p=rotateleft(ya);
            }
            if(p->bf==0){
                    ya->bf=0;
    s->bf=0;
            }else
                    if(p->bf==imbal){
                            ya->bf=-imbal;
                            s->bf=0;
                    }else{
                            ya->bf=0;
       s->bf=imbal;
                    }
            p->bf=0;

    }
```

```
        if(fya==NULL)
                *pavltree=p;
        else
                if(ya==fya->right)
                        fya->right=p;
                else
                        fya->left=p;
}
```

3.5 Chương trình hiện thực cây AVL

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

struct nodetype{
        int info;
        int bf; //balance factor
   struct nodetype *left, *right;
};
typedef struct nodetype *NODEPTR;

//lay ve mot node moi tu he thong
NODEPTR getnode(){
        NODEPTR p;
        p=(NODEPTR) malloc(sizeof (struct nodetype));
   return p;
}

//khoi dong gia tri ban dau cho cay can bang
void initialize(NODEPTR *pavltree){
        *pavltree=NULL;
}

//tao mot node moi trong bo nho
NODEPTR makenode(int x){
        NODEPTR p;
        p=getnode();
        p->info=x;
        p->bf=0;
        p->left=NULL;
        p->right=NULL;
   return p;
}

//duyet cay theo thu tu truoc
void pretrav(NODEPTR proot){
```

```
        if(proot!=NULL){
                printf("%4d",proot->info);
                pretrav(proot->left);
      pretrav(proot->right);
        }
}

//duyet cay theo thu tu giua
void intrav(NODEPTR proot){
        if(proot !=NULL){
                intrav(proot->left);
                printf("%4d",proot->info);
      intrav(proot->right);
        }
}

//duyet cay theo thu tu cuoi
void posttrav(NODEPTR proot){
        if(proot !=NULL){
                posttrav(proot->left);
                posttrav(proot->right);
      printf("%4d",proot->info);
        }
}

//tim kiem khoa x trong cay avl
NODEPTR search(NODEPTR proot, int x){
        NODEPTR p;
        p=proot;
        if(p!=NULL)
                if(x<proot->info)
                        p=search(proot->left,x);
                else
        if(x>proot->info)
                                p=search(proot->right,x);
   return p;
}

//Xoay trai
NODEPTR rotateleft(NODEPTR proot){
        NODEPTR p;
        p=proot;
        if(proot==NULL){
                printf("\n Khong the xoay trai vi cay rong");
        }else{
                if(proot->right==NULL)
```

```
                printf("\n Khong the xoay trai vi khong co node con ben phai");
            else{
                p=proot->right;
                proot->right=p->left;
        p->left=proot;
            }
        }
    return p;
}

//Xoay phai
NODEPTR rotateright(NODEPTR proot){
        NODEPTR p;
        p=proot;
        if(proot==NULL)
            printf("\n Khong the xoay phai vi cay bi rong");
        else
            if(proot->left==NULL)
                printf("\n Khong the xoay phai vi khong co con ben trai");
            else{
                p=proot->left;
                proot->left=p->right;
        p->right=proot;
            }
        return p;
}

//Them mot node moi vao cay AVL
void insert(NODEPTR *pavltree, int x)
{
        //fp la nut cha cua p, q la con cua p
        //ya la nut truoc gan nhat co the mat can bang, fya la cha cua ya
        //s la nut con cua ya theo huong mat can bang
        //imbal=1: lech trai; =-1 lech phai

        NODEPTR fp,p,q, fya,ya,s;
        int imbal;

    //khoi dong cac gia tri
        fp=NULL;
        p=*pavltree;
        fya=NULL;
        ya=p;

        while(p!=NULL){
            if(x==p->info)
```

```
                return;
        if(x<p->info)
                q=p->left;
        if(x>p->info)
                q=p->right;

        if(q!=NULL)
                if(q->bf !=0){  //neu bi mat can bang
                        fya=p;
   ya=q;
                }
        fp=p;
        p=q;
}

q=makenode(x);
//them vao mot node la con cua fp
if(x<fp->info)
        fp->left=q;
else
        fp->right=q;

//hieu chinh lai chi so can bang cua tac  ca cac node giua ya va q
if(x<ya->info)
        p=ya->left;
else
        p=ya->right;

s=p;
while(p!=q){
        if(x<p->info){
                p->bf=1;
p=p->left;
        }else{
                p->bf=-1;
p=p->right;
        }
}

//xac dinh huong lech
if(x<ya->info)
        imbal=1;
else
        imbal=-1;

if(ya->bf==0){
```

```
                ya->bf=imbal;
        return;
           }
           if(ya->bf !=imbal){
                ya->bf=0;
        return;
           }

           if(s->bf==imbal){
                if(imbal==1){
           p=rotateright(ya);
                }else{
           p=rotateleft(ya);
                }
                ya->bf=0;
        s->bf=0;
           }else{
                if(imbal==1){
                        ya->left=rotateleft(s);
           p=rotateright(ya);
                }else{
                        ya->right=rotateright(s);
           p=rotateleft(ya);
                }
                if(p->bf==0){
                        ya->bf=0;
           s->bf=0;
                }else
                        if(p->bf==imbal){
                                ya->bf=-imbal;
                                s->bf=0;
                        }else{
                                ya->bf=0;
              s->bf=imbal;
                        }
                p->bf=0;

           }
           if(fya==NULL)
                *pavltree=p;
           else
                if(ya==fya->right)
                        fya->right=p;
                else
                        fya->left=p;
}
```

```
void main(){
        int noidung,chucnang;
        NODEPTR pavltree,p;
   initialize(&pavltree);
        do{
                printf("\n\n \t CAY NHI PHAN TIM KIEM CAN BANG AVL");
                printf("\n Cac chuc nang cua chuong trinh: ");
                printf("\n 1. Them node");
                printf("\n 2. Duyet cay theo thu tu truoc NLR");
                printf("\n 3. Duyet cay theo thu tu giua LNR");
                printf("\n 4. Duyet cay theo thu tu sau LRN");
                printf("\n 5. Tim kiem");
                printf("\n 0. Ket thuc chuong trinh");
                printf("\n\n Chuc nang cua ban la:");
                scanf("%d",&chucnang);
                switch(chucnang){
                        case 1:
                                printf("Noi dung moi: ");
                                scanf("%d",&noidung);
                                if(pavltree==NULL)
                                        pavltree=makenode(noidung);
                                else
                                        insert(&pavltree,noidung);
                                break;
                        case 2:
                                printf("\n Duyet cay theo thu tu truoc NLR: ");
            pretrav(pavltree);
                                break;
                        case 3:
                                printf("\n Duyet cay theo thu tu giua LNR: ");
                                intrav(pavltree);
                                break;
                        case 4:
                                printf("\n Duyet cay theo thu tu sau LRN: ");
                                posttrav(pavltree);
                                break;
                        case 5:
                                printf("\n Noi dung can tim");
                                scanf("%d",&noidung);
                                if(search(pavltree,noidung))
                                        printf("\n Tim thay");
                                else
                printf("\n Khong tim thay");
                                break;
```

```
            }
        }while(chucnang !=0);

}
```

BÀI TẬP

1. Một cây nhị phân được gọi là cây nhị phân đúng nếu nút gốc của cây và các nút trung gian đều có hai nút con. Chứng minh rằng nếu cây nhị phân đúng có n nút lá thì cây có tấc cả 2n -1 nút. Hãy viết chương trình kiểm tra xem một cây nhị phân có phải là một cây nhị phân đúng hay không? nếu cây không phải là cây nhị phân đúng, tìm cách bổ sung một số nút để cây trở thành cây nhị phân đúng.
2. Một cây nhị phân được gọi là cây nhị phân đầy với chiều sâu d khi và chỉ khi ở mức i (0=<i<=d) cây có đúng 2i nút. Hãy viết chương trình kiểm tra xem một cây nhị phân có phải là cây nhị phân đầy hay không? Nếu chưa phải là cây nhị phân đầy, tìm cách bổ sung một số node vào cây nhị phân để nó trở nên đầy.
3. Hãy xây dựng các thao tác sau trên cây nhị phân:
    - Tạo lập cây nhị phân
    - Đếm số nút của cây.
    - Xác định chiều cao của cây nhị phân
    - Xác định số nút lá của cây.
    - Xác định số nút trung gian của cây.
    - Xác định số nút trong từng mức của cây nhị phân
4. Xét tác vụ insert và remove để thêm nút và xoá nút trên cây nhị phân tìm kiếm.
    - Vẽ lại hình ảnh của cây BST nếu thêm các nút vào cây theo thứ tự như sau: 8, 3, 5, 2, 20, 11, 30, 9, 18, 4.
    - Vẽ lại hình ảnh của cây trên nếu ta lần lược xoá 2 nút 5 và 20.
5. Làm lại bài 4 với giả thuyết là cây AVL?
6. Viết chương trình làm lịch công tác có các chức năng sau:
    - Nhập nội dung công việc cần làm theo ngày, theo giờ.
    - Xem lịch công tác theo ngày yêu cầu.
    - Xem lịch công tác từ ngày1 đến ngày2
    - Xoá, điều chỉnh lịch công tác. Nếu sau khi điều chỉnh, ngày nào không còn việc phải làm sẽ xoá khỏi lịch công tác.

Yêu cầu chương trình có cài đặt cây nhị phân tìm kiếm: mỗi nút trên cây là một ngày của lịch công tác, trong mỗi nút ngày lại có một danh sách liên kết lưu giữ các giờ có trong ngày.

Chương 5:

CÂY NHIỀU NHÁNH TÌM KIẾM

Cây nhị phân là cây bậc 2, mỗi nút của cây nhị phân có tối đa là hai nhánh cây con. Còn cây nhiều nhánh là cây có bậc lớn hơn 2, mỗi nút trên cây nhiều nhánh thường có nhiều khoá và có nhiều hơn hai nhánh cây con.

Cây nhiều nhánh có rất nhiều loại, trong chương này chúng ta chỉ nghiên cứu cây nhiều nhánh tìm kiếm thông qua hai loại cây như sau: Cây nhiều nhánh tìm kiếm trên xuống (top-down multiway search tree) và cây B-Tree.

1. GIỚI THIỆU CÂY NHIỀU NHÁNH

1.1 Định nghĩa cây nhiều nhánh

Cây nhiều nhánh là một cấu trúc gồm một tập hữu hạn các nút cùng kiểu dữ liệu (tập các nút này có thể là tập rỗng), tập nút này được phân thành các tập con như sau:

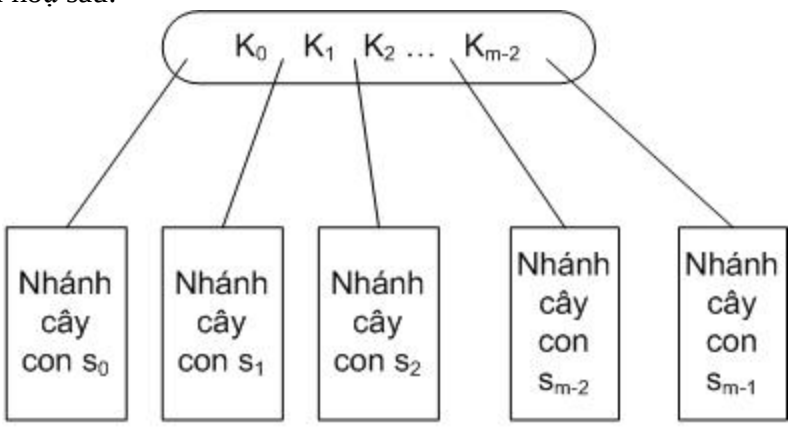
- Tập thứ nhất có một nút gọi là nút gốc.
- Các tập con còn lại tự thân hình thành các cây nhiều nhánh, gọi là các nhánh cây con của nút gốc, các nhánh cây con này cũng có thể là cây rỗng.

Người ta thường dùng đồ thị để biểu diễn các cây nhiều nhánh, mỗi nút của cây được minh hoạ bằng một vòng tròn, trong vòng tròn có ghi các khoá của nút.
Các khái niệm trên cây nhị phân trong chương trước cũng được áp dụng cho cây nhiều nhánh như: bậc của cây, bậc của nút, đường đi …

1.2 Định nghĩa cây nhiều nhánh tìm kiếm

Ở chương trước, chúng ta đã nghiên cứu và cài đặt cây nhị phân tìm kiếm (Binary Search Tree), cây nhiều nhánh tìm kiếm cũng giống như cây nhị phân tìm kiếm nhưng tổng quát hơn. Mỗi nút trên cây nhiều nhánh tìm kiếm có nhiều khoá và nhiều nhánh cây con, số khoá ít hơn số nhánh cây con là 1. Ví dụ nút có 3 khoá thì có 4 nhánh cây con, nút có 4 khoá thì có 5 nhánh cây con…

Xét hình minh hoạ sau:



Hình minh hoạ nút có m -1 khoá và m nhánh cây con

Hình trên minh hoạ một nút trên cây nhiều nhánh tìm kiếm. Giả sử nút này có bậc m: nút có m – 1 khoá và có m nhánh cây con. Gọi:

$k_0, k_1, k_2, \ldots, k_{m-2}$ là m – 1 khoá (theo thứ tự tăng dần của nút).

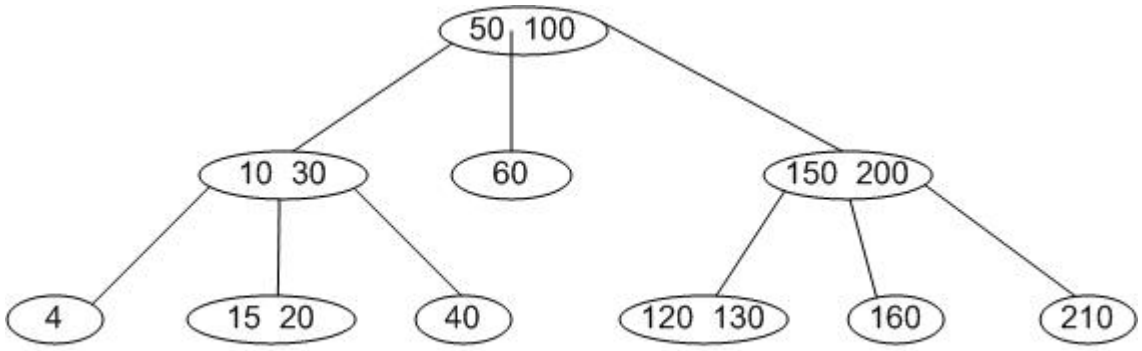
$s_0, s_1, s_2, \ldots, s_{m-1}$ là m nhánh cây con của nút

Nhánh cây con s_i gọi là nhánh cây con bên trái của khoá k_i và nút gốc của nhánh cây con s_i gọi là nút con bên trái của khoá k_i.
Tương tự, nhánh cây con s_i còn được gọi là nhánh cây con bên phải của khoá k_{i-1} và nút gốc của nhánh cây con s_i gọi là nút con bên phải của khoá k_{i-1}.

Định nghĩa cây nhiều nhánh tìm kiếm
Cây nhiều nhánh tìm kiếm là cây nhiều nhánh mà ở mỗi nút của cây thoả mãn các tính chất sau:

- Tất cả các khoá trên nhánh cây con s_0 đều nhỏ hơn hay bằng khoá k_0.
- Tất cả các khoá trên nhánh cây con s_i (1<=i<=m-2) đều lớn hơn khoá k_{i-1} và nhỏ hơn hay bằng khoá k_i.
- Tất cả các khoá trên nhánh cây con s_{m-1} đều lớn hơn khoá k_{m-2}.

Hình vẽ sau đây minh hoạ cây nhiều nhánh tìm kiếm bậc 3:



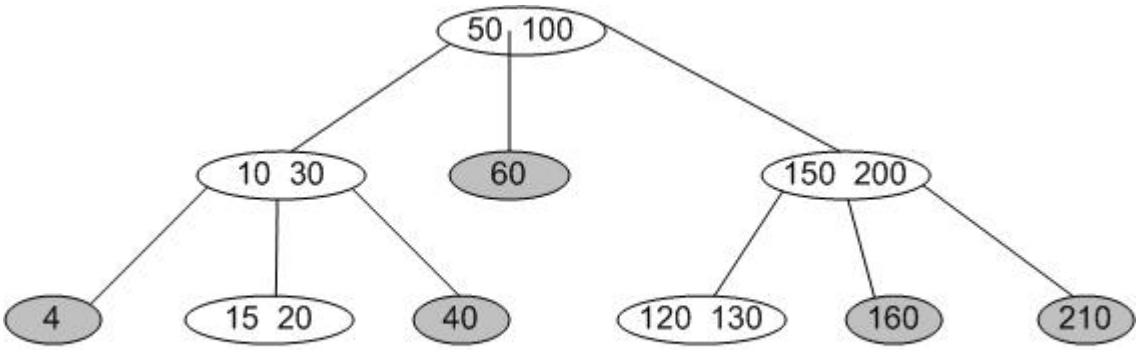
Hình minh hoạ cây nhiều nhánh tìm kiếm bậc 3

Sau đây chúng ta sẽ tiến hành xem xét hai cây tìm kiếm nhiều nhánh thông dụng là cây top-down và cây Btree.

2. CÂY TRÊN XUỐNG

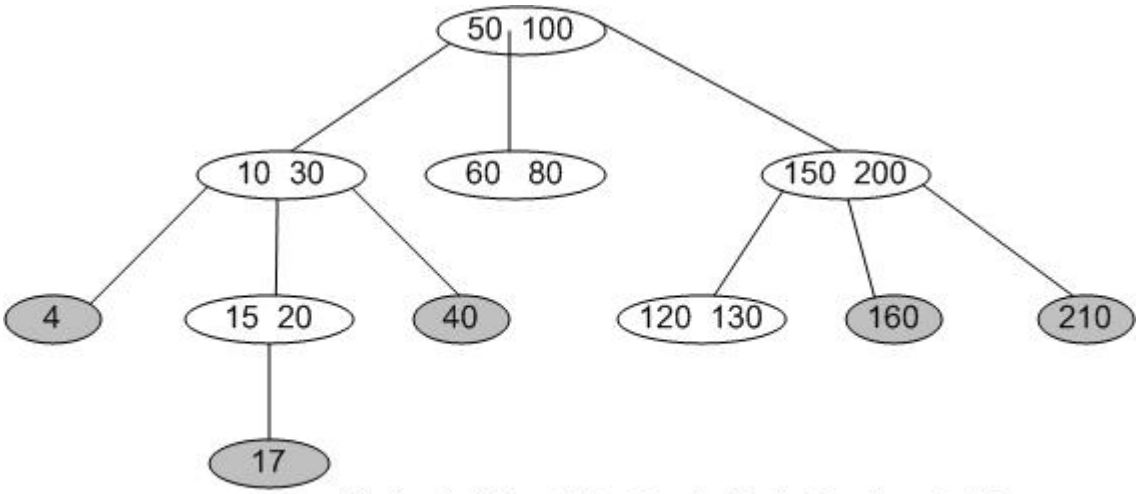
2.1 Giới thiệu cây trên-xuống

Cây trên xuống là cây nhiều nhánh tìm kiếm mà tất cả các nút không đầy đều là nút lá. Hình sau mô tả cây trên xuống bậc 3, với các nút không đầy được tô màu:



Hình minh hoạ cây trên - xuống bậc 3

Khi thêm một khoá vào cây trên-xuống chúng ta phải tìm nút lá phù hợp để chèn khoá mới vào nút lá này. Nếu nút lá chưa đầy thì ta chèn khoá vào, còn nếu nút lá này đã đầy thì chúng ta phải cấp phát một nút lá mới để chứa khoá, nút lá mới này là con của nút lá cũ.
Hình vẽ sau mô tả việc thêm 2 khoá 17 và 80 vào cây trên-xuống ở trên:



Hình minh hoạ thêm khoá 17 và 80 vào cây trên - xuống bậc 3

2.2 Cài đặt cây trên - xuống
2.2.1 Khai báo cấu trúc
Gọi ORDER là bậc của cây trên xuống.
Gọi numtrees là số nhánh của cây con của một nút (numtrees <= ORDER), nút này sẽ có numtrees -1 khoá.
Khai báo mỗi nút trên cây trên-xuống là một mẩu tin có các trường sau:

- Trường numtrees: số nhánh cây con của nút.
- Trường key: là mảng chứa các khoá của nút.
- Trường son: là mảng chứa các con trỏ chỉ đến các nút con của nút.

```
#define ORDER 4
```

```
struct node{
        int numtrees;//so cay con cua mot nut
        int key[ORDER -1];//cac khoa cua mot node
        struct node *son[ORDER];//cac con tro chi den cac nut con cua mot node cha
};
typedef struct node *NODEPTR;
NODEPTR ptree;
```

2.2.2 Các tác vụ

- *Tác vụ makenode*

  Tác vụ này dùng để tạo nút mới cho cây trên xuống. Nút mới tạo là nút có một khoá và hai nhánh cây con. Hàm makenode được gọi khi thêm khoá vào cây trên xuống trong các trường hợp: cây đang bị rỗng và chúng ta thêm khoá đầu tiên vào cây đó hoặc là nút lá để chèn khoá vào đã đầy, chúng ta phải cấp phát một nút lá mới để chứa khoá, nút lá mới là con của nút lá trước.

```
NODEPTR makenode(int k){
        int i;
        NODEPTR p;
        p=getnode();
        p->numtrees=2;
        p->key[0]=k;
        //nut moi chua co cac nut con
        for(i=0;i<ORDER;i++)
                p->son[i]=NULL;
        return p;
}
```

- *Tác vụ tìm kiếm một khoá trên nút*

  Trả về vị trí nhỏ nhất của khoá trong nút p bắt đầu lớn hơn hay bằng k. Trường hợp k lớn hơn tất cả các khoá trong nút p thì trả về vị trí p->numtrees – 1.

```
int  nodesearch(NODEPTR p, int k){
        int i;
        for(i=0;i<p->numtrees -1&&p->key[i]<k;i++);
        return i;
}
```

- *Tác vụ tìm kiếm 1 khoá trên cây*

  Tìm khoá k trên cây trên - xuống. Con trỏ p xuất phát từ nút gốc và len xuống các nhánh cây con phù hợp để kiếm khoá k có trong một nút nào trên cây hay không.

  Nếu có khoá k tại nút p thì:
  - Biến found trả về giá trị TRUE.
  - Hàm search() trả về con trỏ p chỉ nút có chứa khoá k.
  - Biến position trả về vị trí của khoá k có trong nút p.

Nếu không có khoá k trên cây, lúc này p=NULL và q (nút cha của p) chỉ nút lá có thể thêm khoá k vào.

- Biến found trả về giá trị FALSE
- Hàm search trat về con trỏ q chỉ vị trí của nút lá có thể thêm khoá k.
- Biến Position trả về vị trí có thể chèn khoá k vào nút lá q.

```
NODEPTR search(int k, int *pposition, int *pfound){
        int i;
        NODEPTR p,q;
        q=NULL;
        p=ptree;
        while(p!=NULL){
                i=nodesearch(p,k);
                if(i< p->numtrees-1 && k==p->key[i]){//found
                        *pfound=TRUE;
                        *pposition=i;
                        return p;
                }
                q=p;
                p=p->son[i];
        }
        *pfound=FALSE;
        *pposition=i;
   return q;
}
```

- *Tác vụ duyệt cây*

```
void traverse(NODEPTR proot){
        int i, nt;
        if(proot==NULL)
                return;
        else{
                nt=proot->numtrees;
                for(i=0;i<nt-1;i++){
                        traverse(proot->son[i]);
        printf("%8d",proot->key[i]);
                }
      traverse(proot->son[nt-1]);
        }
}

void viewnodes(NODEPTR proot, int level){
        int i;
        if(proot==NULL)
                return;
```

```
        else{
            printf("\n Nut %p (muc %4d): ",proot,level);
            for(i=0;i<proot->numtrees;i++){
                printf("%4d",proot->key[i]);
            }
            printf("\n");
            for(i=0;i<proot->numtrees;i++){
        viewnodes(proot->son[i],level+1);
            }
        }
    }
```

- *Tác vụ chèn 1 khoá vào nút lá*

```
    void insleaf(NODEPTR s, int k, int pos){
        int i,nt;
        nt=s->numtrees;
        s->numtrees=nt+1;
        for(i=nt-1;i>pos;i--){
        s->key[i]=s->key[i-1];
        }
      s->key[pos]=k;
    }
```

- *Tác vụ chèn 1 khoá vào cây top-down*

```
    NODEPTR insert(int k){
        NODEPTR s,p;
        int position,found;
        if(ptree==NULL){
            ptree=makenode(k);
      return ptree;
        }
        s=search(k,&position,&found);
        if(found==TRUE){
            printf("\n Bi trung khoa");
      return s;
        }
        if(s->numtrees<ORDER){
            insleaf(s,k,position);
      return s;
        }
        p=makenode(k);
        s->son[position]=p;
    return p;
    }
```

2.3 Chương trình minh hoạ cây top-down

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>

#define TRUE 1
#define FALSE 0
#define ORDER 4

struct node{
        int numtrees;//so cay con cua mot nut
        int key[ORDER -1];//cac khoa cua mot node
        struct node *son[ORDER];//cac con tro chi den cac nut con cua mot node cha
};
typedef struct node *NODEPTR;
NODEPTR ptree;

//tac vu khoi tao cho cay nhieu nhanh
void initialize(){
        ptree=NULL;
}

NODEPTR getnode(){
        NODEPTR p;
        p=(NODEPTR)malloc(sizeof(struct node));
        return p;
}

void freenode(NODEPTR p){
        free(p);
}

NODEPTR makenode(int k){
        int i;
        NODEPTR p;
        p=getnode();
        p->numtrees=2;
        p->key[0]=k;
   //nut moi chua co cac nut con
        for(i=0;i<ORDER;i++)
                p->son[i]=NULL;
   return p;
}

//search tim khoa k trong nut p
//neu tim thay tra ve index i, neu khong thay tra ve p->numtrees
```

```
int  nodesearch(NODEPTR p, int k){
        int i;
        for(i=0;i<p->numtrees -1&&p->key[i]<k;i++);
        return i;
}

NODEPTR search(int k, int *pposition, int *pfound){
        int i;
        NODEPTR p,q;
        q=NULL;
        p=ptree;
        while(p!=NULL){
                i=nodesearch(p,k);
                if(i< p->numtrees-1 && k==p->key[i]){//found
                        *pfound=TRUE;
                        *pposition=i;
                        return p;
                }
                q=p;
                p=p->son[i];
        }
        *pfound=FALSE;
        *pposition=i;
   return q;
}

void traverse(NODEPTR proot){
        int i, nt;
        if(proot==NULL)
                return;
        else{
                nt=proot->numtrees;
                for(i=0;i<nt-1;i++){
                        traverse(proot->son[i]);
        printf("%8d",proot->key[i]);
                }
      traverse(proot->son[nt-1]);
        }
}

void viewnodes(NODEPTR proot, int level){
        int i;
        if(proot==NULL)
                return;
        else{
                printf("\n Nut %p (muc %4d): ",proot,level);
```

```
                for(i=0;i<proot->numtrees;i++){
                        printf("%4d",proot->key[i]);
                }
                printf("\n");
                for(i=0;i<proot->numtrees;i++){
        viewnodes(proot->son[i],level+1);
                }
        }
}

void insleaf(NODEPTR s, int k, int pos){
        int i,nt;
        nt=s->numtrees;
        s->numtrees=nt+1;
        for(i=nt-1;i>pos;i--){
        s->key[i]=s->key[i-1];
        }
   s->key[pos]=k;
}

NODEPTR insert(int k){
        NODEPTR s,p;
        int position,found;
        if(ptree==NULL){
                ptree=makenode(k);
      return ptree;
        }
        s=search(k,&position,&found);
        if(found==TRUE){
                printf("\n Bi trung khoa");
      return s;
        }
        if(s->numtrees<ORDER){
                insleaf(s,k,position);
      return s;
        }
        p=makenode(k);
        s->son[position]=p;
   return p;
}



void main(){
        int i,n,k,pos,timthay,chucnang;
        NODEPTR p;
```

```
	initialize();
	do{
		printf("\n\n CHUONG TRINH HIEN THUC CAY NHIEU NHANH
TREN XUONG");
		printf("\n\n Cac chuc nang chinh cua chuong trinh");
		printf("\n 1. Them mot khoa");
		printf("\n 2. Them ngau nhien nhieu khoa");
		printf("\n 3. Duyet cay theo thu tu nho den lon");
		printf("\n 4. Xem noi dung tung nut cua cay tren xuong");
		printf("\n 5. Tim kiem");
		printf("\n 0. Ket thuc chuong trinh");
		printf("\n\n Chuc nang ban chon: ");
		scanf("%d",&chucnang);

		switch(chucnang){
			case 1:
				printf("\n Noi dung khoa moi: ");
				scanf("%d",&k);
	insert(k);
				break;
			case 2:
				printf("\n So node muon chen vao: ");
				scanf("%d",&n);
				for(i=0;i<n;i++){
					insert(random(1000));
				}
				printf("\n Da them vao cay %d node ngau nhien",n);
				break;
			case 3:
				printf("\n Duyet cay theo thu tu tu nho den lon");
				if(!ptree)
					printf("\n Cay bi rong");
				else
		traverse(ptree);
				break;
			case 4:
				printf("\n Xem noi dung tung node cua cay tu tren xuong:
");
				if(!ptree)
					printf("\n Cay rong");
				else{
					viewnodes(ptree,0);
	getch();
				}
				break;
			case 5:
```

```
                                printf("\n Khoa can tim: ");
                                scanf("%d",&k);
                                p=search(k,&pos,&timthay);
                                if(timthay)
                                        printf("\n Tim thay tai vi tri %d cua node co con tro
%p",pos,p);
                                else
                    printf("\n Khong tim thay");
                break;
                    }
        }while(chucnang !=0);
}
```

3. CÂY BTREE

3.1 Định nghĩa cây Btree

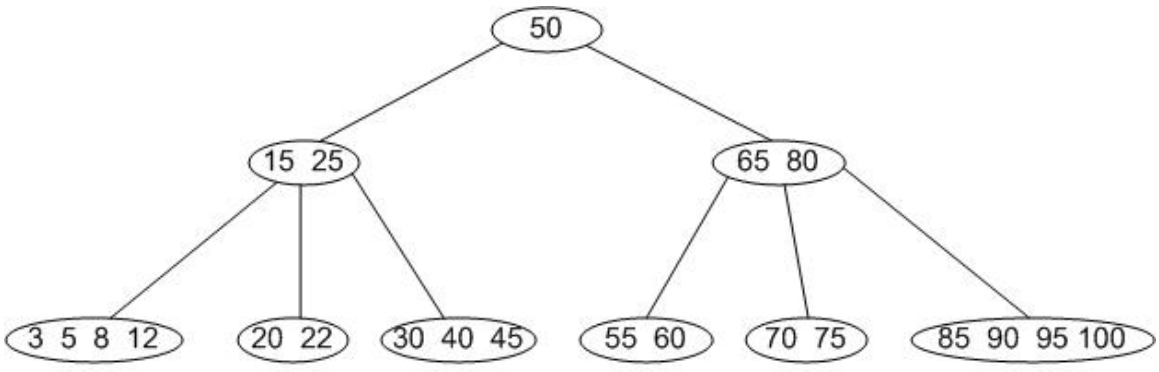
Cây Btree bậc ORDER là cây nhiều nhánh tìm kiếm bậc ORDER thoả hai điều kiện sau:

- Tất cả các nút lá trên cây có cùng một mức.
- Tất cả các nút trên cây (trừ nút gốc) có ít nhất (ORDER -1)/2 khoá.

Một số nhận xét về cây Btree:

- Btree là cây cân bằng và chiều sâu của cây Btree nhỏ nên việc tìm một khoá trên cây Btree được thực hiện nhanh do ít lần so sánh.
- Vì tất cả các nút đều đầy hơn một nửa nên cấu trúc B-Tree khá tối ưu về bộ nhớ.
- Người ta thường dùng cấu trúc Btree để truy xuất dữ liệu được tổ chức ở bộ nhớ ngoài.

Hình vẽ sau đây minh hoạ hình ảnh của cây Btree bậc 5:



Hình minh hoạ cây Btree bậc 5

3.2 Thêm khoá vào cây Btree

Khi thêm một khoá vào cây Btree chúng ta phải tìm nút lá phù hợp để chèn khoá mới vào nút lá này.

- Nếu nút lá này chưa đầy thì chúng ta chèn khoá mới vào nút lá.
- Nếu nút lá đã đầy (đã có ORDER -1 khoá và ORDER nhánh con), nếu tính luôn khoá mới ta sẽ có ORDER -1 khoá và ORDER + 1 nhánh cây con. Gọi midkey là

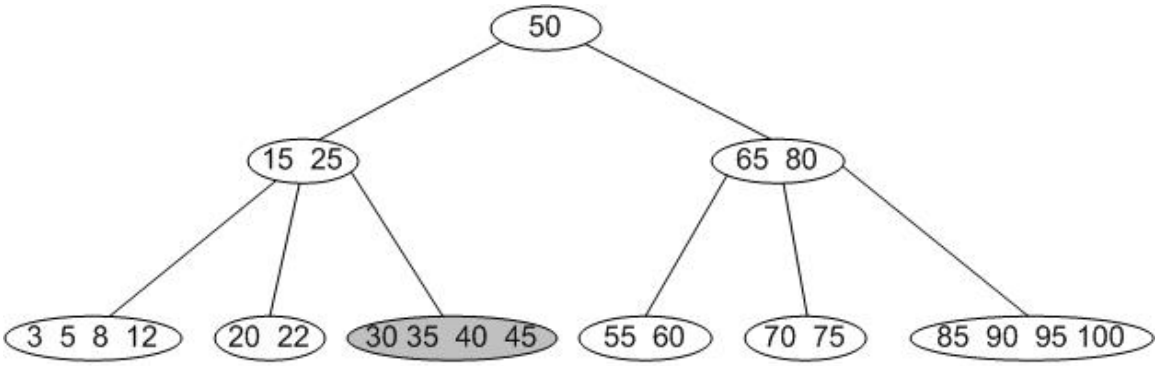
khoá nằm ngay chính giữa của ORDER khoá, chúng ta tách nút lá bị tràn này thành hai nút bằng nhau như sau:
Nút nữa trái (gọi là nút nd) gồm các khoá nhỏ từ vị trí 0 đến vị trí midkey -1.
Nút nữa phải (gọi là nút nd2) gồm các khoá lớn từ vị trí midkey + 1 đến ORDER.

- Và khoá chính giữa tại vị trí midkey và nút con nd2 được chèn vào nút cha. Vấn đề được sử lý tương tự khi chèn khoá midkey và nút con nd2 vào nút cha.

Hình vẽ sau minh hoạ việc chèn các khoá 35, 2, 42, 41, 44, 43 vào cây Btree bậc 5 ở trên:

- *Thêm khoá 35*

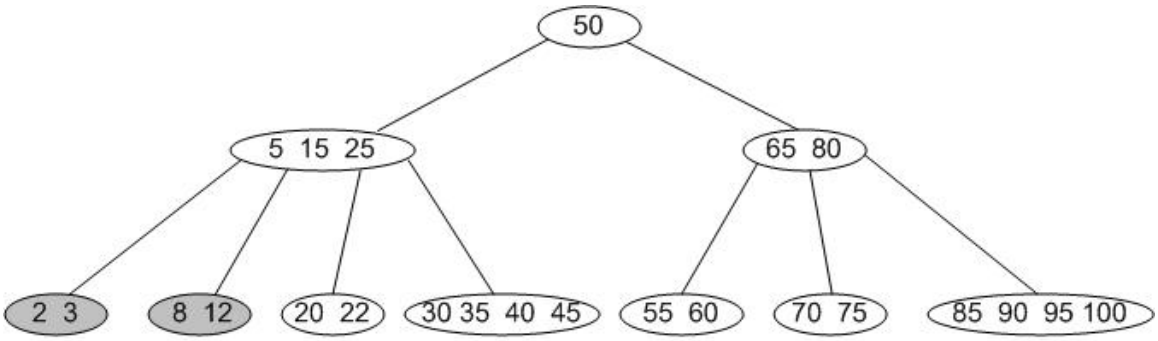
Thêm khoá 35 bằng cách chèn khoá 35 vào nút lá chưa đầy như hình:



Hình Chèn 35 vào nút lá chưa đầy

- *Thêm khoá 2*

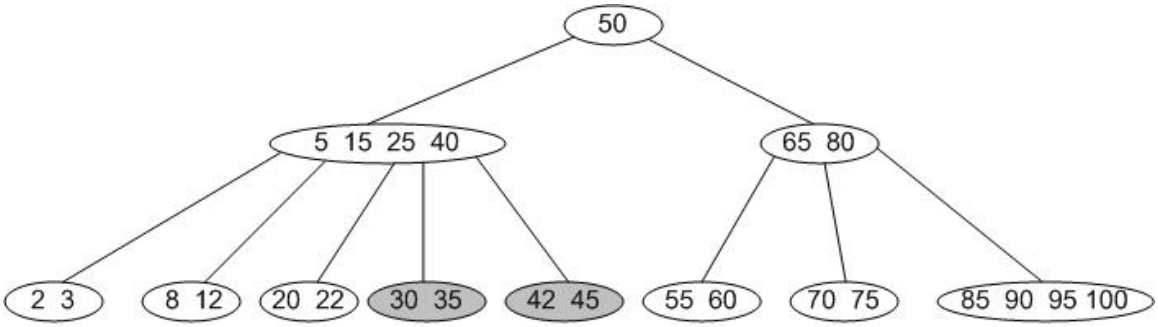
Thêm 2 vào nút lá đầy thì ta tách nút lá ra làm 2 nút, nửa trái có 2 khoá là 2 và 3, nửa phải có 2 khoá là 8 và 12, nút 5 được đưa lên nút cha.



Hình Chèn 2 vào nút lá đã đầy

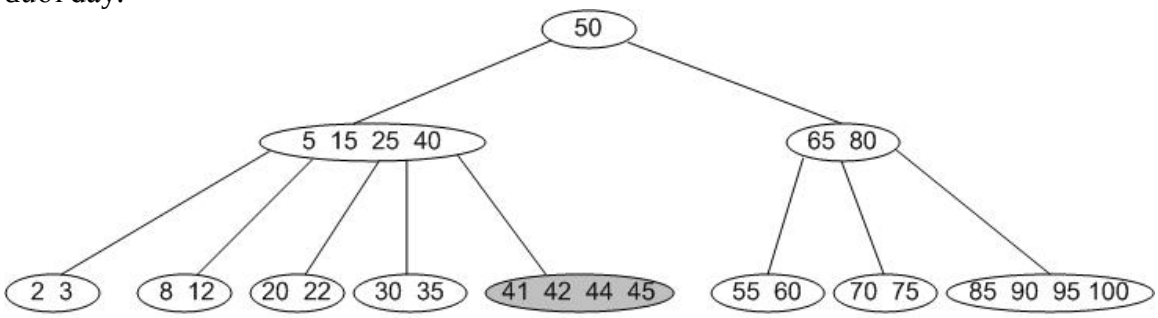
- *Thêm khoá 42*

Thêm 42 vào nút lá đã đầy, tách nút lá này ra làm 2 nút con giống như quá trình ở trên.



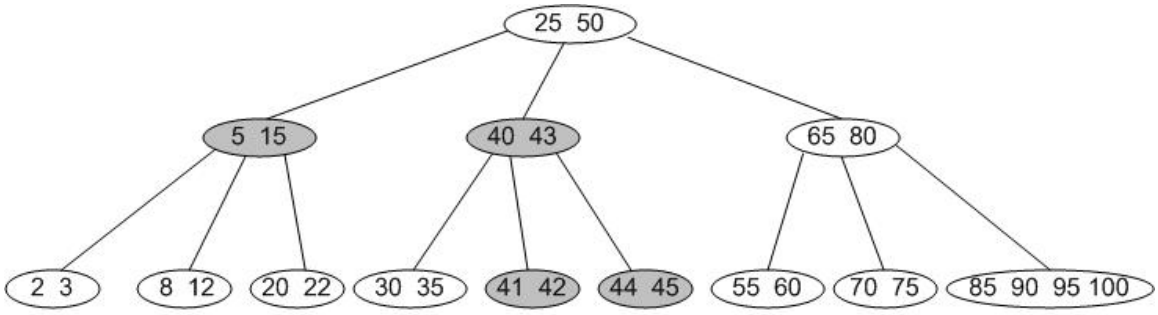
Hình Chèn 42 vào nút lá đã đầy

- *Thêm khoá 41 và 44*
  Thêm khoá 41 và 44 vào nút lá chưa đầy, ta cứ thêm vào 2 khoá này như hình vẽ dưới đây.



Hình chèn 41 và 44 vào nút lá chưa đầy

- *Thêm khoá 43*
  Khi thêm 43 vào nút lá đã đầy, nút này tách ra làm 2 nút như trên, nhưng khi chèn nút giữa là 43 vào nút cha, thì nút cha bị đầy và tiếp tục tách nút tại nút cha. Hình vẽ sau mô tả kết quả của quá trình chèn 43 vào cây Btree trên.



Hình chèn 43 vào nút lá đã đầy

3.3 Cài đặt cây Btree

3.3.1 Khai báo cấu trúc cho cây Btree

Gọi ORDER là bậc của cây Btree
Gọi Ndiv2 là ORDER/2
Gọi Numtrees là số nhánh cây con của một nút, nút này sẽ có numtrees – 1 khoá.
Khai báo mỗi nút của cây Btree là một mẩu tin có các trường như sau:

- Trường numtrees là số nhánh cây con của một nút.
- Trường key: là mảng chứa khoá của một nút.
- Trường son: là mảng chứa các con trỏ chỉ các nút con của nút.

```
struct node{
        int numtrees;
        int key[ORDER-1];
   struct node* son[ORDER];
};
typedef struct node *NODEPTR;

//khai bao goc cua cay BTree
NODEPTR ptree;
```

3.3.2 Các tác vụ

- *Tác vụ makeroot*

Tác vụ này được gọi khi thêm khoá vào cây Btree trong các trường hợp:

- Btree đang bị rỗng và chúng ta thêm khoá đầu tiên vào Btree.
- Nút lá thích hợp để chèn khoá và tất cả các nút cha đều đầy, lúc này chúng ta phải tách hàng loạt nút từ nút lá đến nút gốc sau đó gọi hàm makeroot để tạo nút gốc mới của cây Btree.
- Mỗi lần gọi hàm makeroot thì chiều sâu của cây Btree tăng 1.

```
NODEPTR makeroot(int k){
        int i;
        NODEPTR p;
        p=getnode();
        p->numtrees=2;
        p->key[0]=k;
        for(i=0;i<ORDER;i++){
        p->son[i]=NULL;
        }
   return p;
}
```

- *Tác vụ nodesearch*

Trả về vị trí nhỏ nhất của khoá trong nút p bắt đầu lớn hơn hay bằng khoá k. Trường hợp k lớn hơn tất cả các khoá trong nút p thì trả về vị trí p->numtrees -1.

```
int nodesearch(NODEPTR p, int k){
        int i;
        for(i=0;i<p->numtrees-1&&p->key[i]<k;i++);
   return i;
}
```

- *Tác vụ father*

Tìm nút cha của nút s trên cây Btree.

```
NODEPTR father(NODEPTR s){
```

```
        int i;
        NODEPTR p,q;
        if(s==ptree)
                return NULL;
        q=NULL;
        p=ptree;
        while(p!=s){
                i=nodesearch(p,s->key[0]);
                q=p;
                p=p->son[i];
        }
        return q;
}
```

- *Tác vụ search*

Tìm khoá k trên cây Btree. Con trỏ p xuất phát từ gốc và len xuống nhánh các cây con phù hợp để tìm khoá k có trong một nút p hay không.

Nếu có khoá k tại nút p trên cây:

- Biến found trả về giá trị TRUE.
- Hàm search trả về con trỏ chỉ nút p có chứa khoá k.
- Biến position trả về vị trí của khoá k có trong nút p này.

Nếu không có khoá k trên cây: lúc này p=NULL và q (nút cha của p) chỉ nút lá có thể thêm khoá k vào.

- Biến found trả về giá trị False
- Hàm search trả về con trỏ q là nút lá có thể thêm khoá k vào.
- Biến position trả về vị trí có thể chèn khoá k vào nút lá q này.

```
    NODEPTR search(int k, int *pposition, int *pfound){
        int i;
        NODEPTR q,p;
        q=NULL;
        p=ptree;
        while(p!=NULL){
                i=nodesearch(p,k);
                if(i<p->numtrees-1&&k==p->key[i]){
                        *pfound=TRUE;
                        *pposition=i;
                        return p;
                }
                q=p;
                p=p->son[i];
        }
        *pfound=FALSE;
        *pposition=i;
    return q;
```

```
}
```

- *Tác vụ duyệt cây*

```
void traverse(NODEPTR proot){
        int i;
        if(proot==NULL)
                return;
        else{
                for(i=0;i<proot->numtrees-1;i++){
                        traverse(proot->son[i]);
        printf("%8d",proot->key[i]);
                }
      traverse(proot->son[proot->numtrees-1]);
        }
}
```

- *Tác vụ chèn khoá vào cây Btree*

Thêm khoá k vào vị trí position của nút lá s.
Nếu nút lá s chưa đầy, gọi tác vụ insnode để chèn khoá k vào s.
Nếu nút lá s đã đầy, tách nút lá s này thành hai nút nửa trái và nửa phải.

```
void insert(NODEPTR s, int k, int position){

        NODEPTR nd,nd2,f,newnode;
        int pos,newkey,midkey;

        nd=s;
        newkey=k;
        newnode=NULL;
        pos=position;
        f=father(nd);

    //nut bi day
        while(f!=NULL&&nd->numtrees==ORDER){
                split(nd,newkey,newnode,pos,&nd2,&midkey);
                nd=f;
                newkey=midkey;
                newnode=nd2;
                pos=nodesearch(f,midkey);
      f=father(nd);
        }
    //nut chua day
        if(nd->numtrees<ORDER){
                insnode(nd,newkey,newnode,pos);
      return;
        }
```

```
            //nut goc bi day, chieu cao tang len 1
            split(nd,newkey,newnode,pos,&nd2,&midkey);
            ptree=makeroot(midkey);
            ptree->son[0]=nd;
      ptree->son[1]=nd2;
}
```

- *Tác vụ split*

Tách nút đầy nd, tác vụ này được gọi bởi hàm insert.

- nd là nút đầy bị tách, sau khi tách xong nút nd chỉ còn lại một nữa số khoá bên trái.
- Newkey, newnode và pos là khoá mới, nhánh cây con, và vị trí chèn vào nút nd.
- Nút nd2 là nút nữa phải có được sau lần tách, nút nd2 chiếm phân nữa số khoá bên phải.
- Midkey là khoá ngay chính giữa sẽ được chèn vào nút cha.

```
void split(NODEPTR nd, int newkey, NODEPTR newnode,int pos,
NODEPTR *pnd2,int *pmidkey ){

      NODEPTR p;
      p=getnode();

   //vi tri can chen o phia nua phai
      if(pos>Ndiv2){
      copy(nd,Ndiv2+1,ORDER-2,p);
            insnode(p,newkey,newnode,pos-Ndiv2-1);
            nd->numtrees=Ndiv2+1;//so nhanh con lai cua node nua
trai
            *pmidkey=nd->key[Ndiv2];
            *pnd2=p;
   return;
      }

   //vi tri can chen nam ngay giua
      if(pos==Ndiv2){
            copy(nd,Ndiv2,ORDER-2,p);
            nd->numtrees=Ndiv2+1;
            p->son[0]=newnode;
            *pmidkey=newkey;
            *pnd2=p;
   return;
      }

   //vi tri can chen vao ben nua trai
      if(pos<Ndiv2){
```

```
                        copy(nd,Ndiv2,ORDER-2,p);
                        nd->numtrees=Ndiv2;
                        *pmidkey=nd->key[Ndiv2-1];
                        insnode(nd,newkey,newnode,pos);
                        *pnd2=p;
                return;
                    }
            }
```

3.4 Chương trình minh hoạ cây Btree được tổ chức bằng bộ nhớ trong

```
#include<stdio.h>
#include<stdlib.h>

#define ORDER 5
#define Ndiv2 ORDER/2
#define TRUE 1
#define FALSE 0

struct node{
        int numtrees;
        int key[ORDER-1];
   struct node* son[ORDER];
};
typedef struct node *NODEPTR;

//khai bao goc cua cay BTree
NODEPTR ptree;

void initialize(){
        ptree=NULL;
}

NODEPTR getnode(){
        NODEPTR p;
        p=(NODEPTR)malloc(sizeof(struct node));
   return p;
}

NODEPTR makeroot(int k){
        int i;
        NODEPTR p;
        p=getnode();
        p->numtrees=2;
        p->key[0]=k;
        for(i=0;i<ORDER;i++){
        p->son[i]=NULL;
```

```
        }
    return p;
}

//search khoa k trong node p, tra ve vi tri cua khoa nho nhat bat dau lon hon hay bang k
int nodesearch(NODEPTR p, int k){
        int i;
        for(i=0;i<p->numtrees-1&&p->key[i]<k;i++);
    return i;
}

//Tim node cha cua node s tren btree
NODEPTR father(NODEPTR s){
        int i;
        NODEPTR p,q;
        if(s==ptree)
                return NULL;
        q=NULL;
        p=ptree;
        while(p!=s){
                i=nodesearch(p,s->key[0]);
                q=p;
      p=p->son[i];
        }
        return q;
}

//tim kiem mot gia tri trong Btree tra ve node thay hoac la vi tri ma le ra no phai co o do
NODEPTR search(int k, int *pposition, int *pfound){
        int i;
        NODEPTR q,p;
        q=NULL;
        p=ptree;
        while(p!=NULL){
                i=nodesearch(p,k);
                if(i<p->numtrees-1&&k==p->key[i]){
                        *pfound=TRUE;
                        *pposition=i;
        return p;
                }
                q=p;
      p=p->son[i];
        }
        *pfound=FALSE;
        *pposition=i;
    return q;
```

```
}

//duyet cay Btree theo thu tu tang dan
void traverse(NODEPTR proot){
        int i;
        if(proot==NULL)
                return;
        else{
                for(i=0;i<proot->numtrees-1;i++){
                        traverse(proot->son[i]);
        printf("%8d",proot->key[i]);
                }
      traverse(proot->son[proot->numtrees-1]);
        }
}

//chep cac khoa tu vi tri first den last tu node nd sang node nd2
void copy(NODEPTR nd, int first, int last, NODEPTR nd2){
        int i;
        for(i=first;i<=last;i++){
                nd2->key[i-first]=nd->key[i];
        }
        for(i=first;i<=last+1;i++){
                nd2->son[i-first]=nd->son[i];
        }
   nd2->numtrees=last-first+2;
}

//chen node newkey vao vi tri pos cua cay chua day p, newnode se la cay
//con ben phai cua khoa newkey
void insnode(NODEPTR p, int newkey, NODEPTR newnode, int pos){
        int i;
        for(i=p->numtrees-1;i>=pos+1;i--){
                p->son[i+1]=p->son[i];
                p->key[i]=p->key[i-1];
        }
        p->key[pos]=newkey;
        p->son[pos+1]=newnode;
   p->numtrees++;
}

//tach node day nd,
void split(NODEPTR nd, int newkey, NODEPTR newnode,int pos, NODEPTR *pnd2,int
*pmidkey ){

        NODEPTR p;
```

```
        p=getnode();

    //vi tri can chen o phia nua phai
        if(pos>Ndiv2){
        copy(nd,Ndiv2+1,ORDER-2,p);
                insnode(p,newkey,newnode,pos-Ndiv2-1);
                nd->numtrees=Ndiv2+1;//so nhanh con lai cua node nua trai
                *pmidkey=nd->key[Ndiv2];
                *pnd2=p;
      return;
        }

    //vi tri can chen nam ngay giua
        if(pos==Ndiv2){
                copy(nd,Ndiv2,ORDER-2,p);
                nd->numtrees=Ndiv2+1;
                p->son[0]=newnode;
                *pmidkey=newkey;
                *pnd2=p;
      return;
        }

    //vi tri can chen vao ben nua trai
        if(pos<Ndiv2){
                copy(nd,Ndiv2,ORDER-2,p);
                nd->numtrees=Ndiv2;
                *pmidkey=nd->key[Ndiv2-1];
                insnode(nd,newkey,newnode,pos);
                *pnd2=p;
      return;
        }
}

//chen khoa k vao nut s o vi tri position
void insert(NODEPTR s, int k, int position){

        NODEPTR nd,nd2,f,newnode;
        int pos,newkey,midkey;

        nd=s;
        newkey=k;
        newnode=NULL;
        pos=position;
        f=father(nd);

    //nut bi day
```

```
        while(f!=NULL&&nd->numtrees==ORDER){
                split(nd,newkey,newnode,pos,&nd2,&midkey);
                nd=f;
                newkey=midkey;
                newnode=nd2;
                pos=nodesearch(f,midkey);
      f=father(nd);
        }
   //nut chua day
        if(nd->numtrees<ORDER){
                insnode(nd,newkey,newnode,pos);
      return;
        }

        //nut goc bi day, chieu cao tang len 1
        split(nd,newkey,newnode,pos,&nd2,&midkey);
        ptree=makeroot(midkey);
        ptree->son[0]=nd;
   ptree->son[1]=nd2;
}

void main(){
        int chucnang,k,pos,timthay;
        NODEPTR p;
   initialize();
        do{
                printf("\n\n CHUONG TRINH HIEN THUC CAY BTREE");
                printf("\n Cac chuc nang cua chuong trinh");
                printf("\n 1.Them vao mot khoa ");
                printf("\n 2.Duyet cay theo thu tu tu nho den lon");
                printf("\n 3.Tim kiem");
                printf("\n 0.Ket thuc chuong trinh");
                printf("\n Chuc nang ban chon: ");
                scanf("%d",&chucnang);

                switch(chucnang){
                        case 1:
                                printf("\n Nhap vao noi dung cua khoa moi: ");
                                scanf("%d",&k);
                                if(ptree==NULL)
                                        ptree=makeroot(k);
                                else{
                                        p=search(k,&pos,&timthay);
                                        if(timthay==TRUE)
                                                printf("\n Bi trung khoa, khong them vao
duoc");
```

```
                                        else{
                insert(p,k,pos);
                                        }
                                }
                                break;
                        case 2:
                                printf("\n Duyet cay Btree theo thu tu tang dan: ");
                                if(!ptree){
                                        printf("\n BTREE RONG");
                                }
                                else{
                insert(ptree);
                                }
                                break;
                        case 3:
                                printf("\n Nhap vao mot khoa can tim: ");
                                scanf("%d",&k);
                                p=search(k,&pos,&timthay);
                                if(timthay)
                                        printf("\ntim thay vi tri cua %d tai con tro
%p",pos,p);
                                else
                printf("\nKhong tim thay");
                                break;
                }
        }while(chucnang!=0);
}
```

3.5 Btree cải tiến

Vì tất cả các nút trên cây Btree đều đầy hơn một nữa nên cấu trúc của cây Btree khá tối ưu bộ nhớ. Để dùng bộ nhớ hiệu quả hơn người ta cải tiến cây Btree thành những cấu trúc như sau:

3.5.1 B - Tree*

B*-Tree bậc ORDER cũng là cây Btree bậc ORDER nhưng tất cả các nút trên cây (trừ nút gốc) phải đầy hơn 2/3.

Chúng ta thấy cấu trúc B*-Tree tối ưu bộ nhớ hơn Btree, hiệu xuất dùng bộ nhớ là lớn hơn 67%.

3.5.2 Compact B-Tree

Là Btree mà tấc cả các nút đều đầy (có thể trừ một vài nút lá ở cuối).

Cây Compact Btree có ưu điểm là tìm kiếm một khoá trên cây nhanh và là cấu trúc đạt hiệu suất sử dụng bộ nhớ tối ưu: 100%.

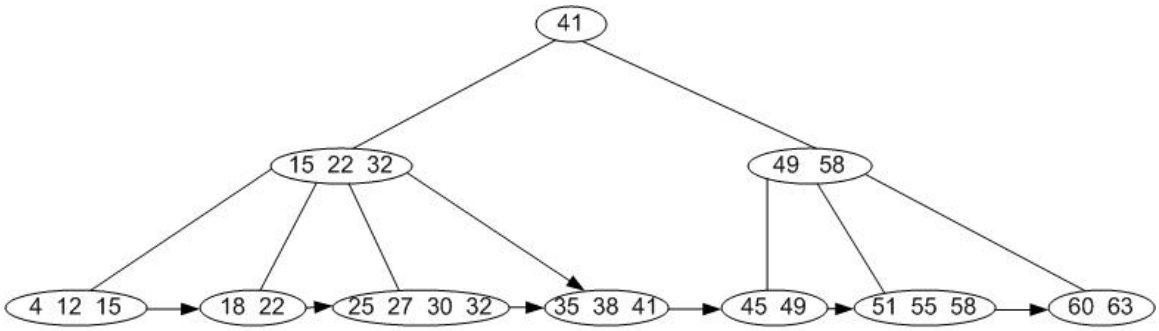
Tuy nhiên cây Compact Btree được ít dùng vì giải thuật để thêm một khoá vào cây rất phức tạp và chi phí để chuyển cây về dạng Compact Btree rất lớn.

3.5.3 B+-Tree

Vì Btree duyệt cây phức tạp nên người ta cải tiến cây Btree thành cây B+tree để quá trình duyệt cây hiệu quả hơn.

B+Tree là Btree mà tất cả các khoá trên cây đều có mặt ở các nút lá. Các nút lá được liên kết với nhau từ trái qua phải hình thành một danh sách liên kết giúp duyệt các khoá trên cây.

Hình vẽ mô tả cây B+Tree:



Hình vẽ mô tả cây B+Tree bậc 4

BÀI TẬP

1. Viết các tác vụ xác định các thông tin về cây Btree.
   - Xác định số nút có trên cây.
   - Xác định số nút lá
   - Xác định chiều sâu của cây.
   - Xác định số nút trên từng mức.
   - Xác định số nút có nội dung > x (x là số nhập vào).
   - Xác định số nút có nội dung < x (x là số nhập vào).
2. Vẽ cây Btree bậc 5 khi chèn vào các khoá sau: 1, 2, 3, 4, 5, 6, 7, 8, 9.
3. Viết giải thuật xoá một nút trên cây Btree.
4. Cài đặt cây B+Tree.

Chương 7

BẢNG BĂM

Các tác vụ trên các cấu trúc như danh sách, cây nhị phân,…phần lớn được hiện thực bằng cách so sánh các nút của cấu trúc, do vậy thời gian truy xuất không nhanh và phụ thuộc vào kích thước của cấu trúc. Chương này chúng ta sẽ xét một cấu trúc mới là bảng băm (hash table), các tác vụ trên bảng băm hạn chế số lần so sánh với mong muốn thời gian truy xuất là tức thời có bậc O(1) và không phụ thuộc vào kích thước của bảng băm.

Với mỗi bảng băm người ta xây dựng một phép băm (hash function) để chuyển đổi số học các khoá của nút thành các địa chỉ trên bảng băm. Bảng băm là cấu trúc dung hòa tốt giữa thời gian truy xuất và dung lượng bộ nhớ. Bảng băm được ứng dụng nhiều trong thực tế, rất thích hợp khi tổ chức dữ liệu có kích thước lớn và được lưu trữ ở bộ nhớ ngoài.

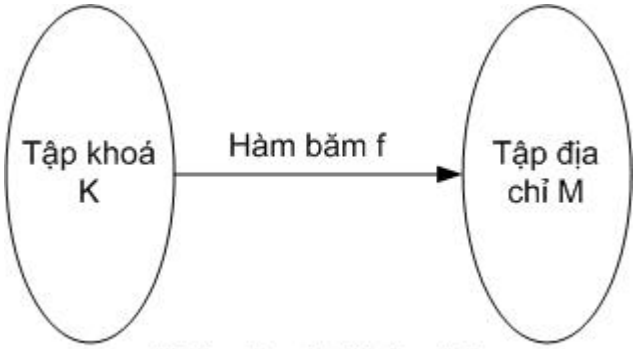
1. MÔ TẢ BẢNG BĂM

1.1 Mô tả dữ liệu

Bảng băm được mô tả bằng các thành phần sau:

- Có tập khoá của các nút trên bảng băm gọi là tập K.
- Có tập các địa chỉ của bảng băm được gọi là tập M.
- Có hàm băm để ánh xạ một khoá trong tập K thành 1 địa chỉ trong tập M.

Bảng băm được mô tả bằng hình vẽ như sau:



Hình vẽ mô tả bảng băm

1.2 Các tác vụ trên bảng băm

Bảng băm có thể có các tác vụ sau:

- Tác vụ khởi động: Cấp phát bộ nhớ và khởi động các giá trị ban đầu cho bảng băm.
- Tác vụ tìm kiếm: Đây là một trong những tác vụ thường được sử dụng nhất của bảng băm. Tác vụ này sẽ tìm kiếm các phần tử trong bảng băm dựa vào khoá của từng phần tử.
- Tác vụ thêm một phần tử:Tác vụ này thêm một phần tử mới vào bảng băm.
- Tác vụ xoá một phần tử: Tác vụ này được dùng để xoá một phần tử ra khỏi bảng băm.
- Tác vụ duyệt bảng băm: Tác vụ này dùng để duyệt qua tất cả các phần tử trên bảng băm.

1.3 Các bảng băm thông dụng

Với mỗi loại bảng băm, chúng ta phải xác định tập khoá K, xác định tập địa chỉ M và xây dựng hàm băm.

Khi xây dựng hàm băm chúng ta muốn các khoá khác nhau sẽ ánh xạ vào các địa chỉ khác nhau, nhưng thực tế thì thường xảy ra trường hợp các khoá khác nhau lại ánh xạ vào cùng một địa chỉ, chúng ta gọi là xung đột. Do đó khi xây dựng bảng băm chúng ta phải xây dựng phương án giải quyết sự xung đột trên bảng băm.

Trong chương này ta sẽ nghiên cứu các bảng băm thông dụng như sau với mỗi bảng băm có chiến lược giải quyết sự xung đột riêng.

- *Bảng băm với phương pháp nối kết trực tiếp:* mỗi địa chỉ của bảng băm tương ứng với một danh sách liên kết. Các nút bị xung đột được nối kết với nhau trên một danh sách liên kết.
- *Bảng băm với phương pháp nối kết hợp nhất:* Bảng băm loại này được cài đặt bằng danh sách kề, mỗi nút có hai trường: trường key chứa khoá của nút và trường next chỉ nút kế bị xung đột. Các nút bị xung đột được nối kết với nhau qua trường liên kết next.
- *Bảng băm với phương pháp dò tuyến tính:* ví dụ như khi thêm nút vào bảng băm loại này nếu băm lần đầu bị xung đột thì lần lược dò địa chỉ kế…cho đến khi gặp địa chỉ trống đầu tiên thì thêm nút vào địa chỉ này.

1.4 Hàm băm

Hàm băm là hàm biến đổi khoá của nút thành địa chỉ trên bảng băm.

- Khoá có thể là khoá ở dạng số hay dạng chuỗi.
- Địa chỉ được tính ra là các số nguyên trong khoảng 0 đến M – 1 với M là số địa chỉ trên bảng băm.
- Hàm băm thường được dùng ở dạng công thức: ví dụ như công thức f(key)=key % M với M là độ lớn cuả bảng băm.

Một hàm băm được coi là tốt thường phải thoả các yêu cầu sau:

- Phải giảm thiểu sự xung đột
- Phải phân bố đều các nút trên M địa chỉ khác nhau của bảng băm.

1.5 Ưu điểm của bảng băm

Bảng băm là một cấu trúc dung hoà tốt giữa thời gian truy xuất và dung lượng bộ nhớ:

- Nếu không có sự giới về bộ nhớ thì chúng ta có thể xây dựng bảng băm với mỗi khoá ứng với một địa chỉ với mong muốn thời gian truy xuất tức thời.
- Nếu dung lượng của bộ nhớ có giới hạn thì tổ chức một số khoá có cùng địa chỉ. Lúc này thời gian truy xuất có bị giảm đi.

Bảng băm được ứng dụng nhiều trong thực tế, rất thích hợp khi tổ chức dữ liệu có kích thước lớn và được lưu trữ ở bộ nhớ ngoài.

2. BẢNG BĂM VỚI PHƯƠNG PHÁP KẾT NỖI TRỰC TIẾP

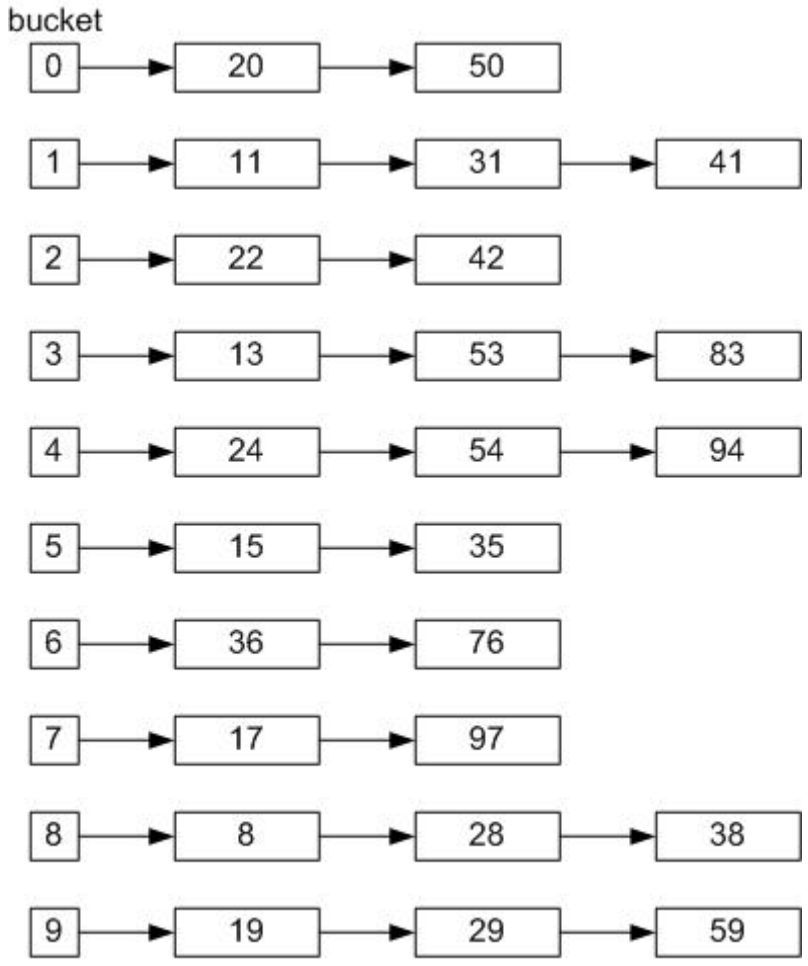
2.1 Mô tả

Bảng băm được cài đặt bằng danh sách liên kết, các nút trên bảng băm được băm thành M danh sách liên kết (từ danh sách 0 đến danh sách M-1). Các nút bị xung đột tại địa chỉ i được nối kết trực tiếp với nhau qua danh sách liên kết i.

Khi thêm một nút có khoá key vào bảng băm, hàm băm f(key) sẽ xác định địa chỉ i trong khoảng 0 đến M – 1 ứng với danh sách liên kết i mà nút này sẽ được thêm vào.

Khi tìm kiếm một nút có khoá key trên bảng băm, hàm băm f(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M – 1 ứng với danh sách liên kết i có thể chứa nút, việc tìm kiếm nút trên bảng băm quy về bài toán tìm kiếm trên danh sách liên kết.

Sau đây là minh hoạ cho bảng băm có tập khoá là tập số tự nhiên, tập địa chỉ có 10 địa chỉ và chọn hàm băm là f(key)=key % 10.



Bảng băm dùng phương pháp nối kết trực tiếp

2.2 Cài đặt

2.2.1Khai báo cấu trúc bảng băm

```
#define M 10
```

```
struct nodes{
        int key;
   struct nodes *next;
};
typedef struct nodes *NODEPTR;
NODEPTR bucket[M];
```

2.2.2 Hàm băm

```
int hashfunc(int key){
        return (key % M);
}
```

2.2.3 Tìm kiếm một phần tử trên bảng băm

```
int search(int k){
        NODEPTR p;
        int b;
        b=hashfunc(k);
        p=bucket[b];
        while(k>p->key&&p!=NULL)
                p=p->next;
        if(p==NULL || k!=p->key)
                return -1;
        else
                return b;
}
```

2.2.4 Thêm vào một phần tử

```
void insafter(NODEPTR p, int k){
        NODEPTR q;
        if(p==NULL)
                printf("Khong them vao node moi duoc");
        else{
                q=getnode();
                q->key=k;
                q->next=p->next;
      p->next=q;
        }
}

// tac vu nay chi su dung khi them vao mot bucket co thu tu
void place(int b, int k){
        NODEPTR p,q;
        q=NULL;
        for(p=bucket[b];p!=NULL && k>p->key;p=p->next)
                q=p;
        if(q==NULL){
```

```
            push(b,k);
            }else{
            insafter(q,k);
            }

}

//them mot node co khoa la k vao trong bang bam
void insert(int k){
            int b;
            b=hashfunc(k);
            place(b,k);
}
```

2.2.5 Xoá một phần tử

```
int delafter(NODEPTR p){
            NODEPTR q;
            int k;
            if(p==NULL||p->next==NULL){
                        printf("\n Khong xoa node duoc");
        return 0;
            }
            q=p->next;
            k=q->key;
            p->next=q->next;
            freenode(q);
    return k;
}

//Xoa mot phan tu co khoa k ra khoi bang bam
void remove(int k){
            int b;
            NODEPTR p,q;
            b=hashfunc(k);
            p=bucket[b];
            q=p;
            while(p!=NULL&&p->key!=k)
            {
                        q=p;
                        p=p->next;
            }
            if(p==NULL)
                        printf("\n Khong co node co khoa la: %d",b);
            else
                        if(p==bucket[b])
                                    pop(b);
```

```
                else
                        delafter(q);
}
```

2.3 Chương trình minh hoạ

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>

#define TRUE 1
#define FALSE 0
#define M 10
struct nodes{
        int key;
   struct nodes *next;
};
typedef struct nodes *NODEPTR;
NODEPTR bucket[M];

// khoi tao bang bam
void initbucket(){
        int b;
        for(b=0;b<M;b++){
                bucket[b]=NULL;
        }
}

//cap nhat mot node moi cho bang bam
NODEPTR getnode(){
        NODEPTR p;
        p=(NODEPTR)malloc(sizeof(struct nodes));
   return p;
}

//xoa mot node trong bo nho
void freenode(NODEPTR p){
        free(p);
}

//Kiem tra mot bucket co phai empty?
int emptybucket(int b){
        if(bucket[b]==NULL)
                return TRUE;
        else
```

```
        return FALSE;
}

//kiem tra toan bo bang bam co null hay khong
int empty(){
        int b;
        for(b=0;b<M;b++)
                if(bucket[b]!=NULL)
                        return FALSE;
   return TRUE;
}
//traverse tren tung bucket
void traversebucket(int b){
        NODEPTR p;
        p=bucket[b];
        while(p!=NULL){
                printf("%4d",p->key);
      p=p->next;
        }
}

//duyet qia bang bam
void traverse(){
        int b;
        for(b=0;b<M;b++){
                printf("\n Bucket[%d]=",b);
      traversebucket(b);
        }
}

//Xoa tren mot bucket
void clearbucket(int b){
        NODEPTR p,q;
        q=NULL;
        p=bucket[b];
        while(p!=NULL){
                q=p;
                p=p->next;
      freenode(q);
        }
   bucket[b]=NULL;
}

//Xoa toan bo bang bam
void clear(){
        int b;
```

```
        for(b=0;b<M;b++){
        clearbucket(b);
        }
}

//Ham hash function
int hashfunc(int key){
        return (key % M);
}

//them mot node vao dau bucket
void push(int b, int x){
        NODEPTR p;
        p=getnode();
        p->key=x;
        p->next=bucket[b];
   bucket[b]=p;
}

//Xoa mot node o dau bucket
int pop(int b){
        NODEPTR p;
        int k;
        if(emptybucket(b)){
                printf("Bucket %d bi rong, khong xoa duoc",b);
      return 0;
        }
        p=bucket[b];
        k=p->key;
        bucket[b]=p->next;
        freenode(p);
   return k;
}

//tac vu them vao bucket mot node moi sau node p
void insafter(NODEPTR p, int k){
        NODEPTR q;
        if(p==NULL)
                printf("Khong them vao node moi duoc");
        else{
                q=getnode();
                q->key=k;
                q->next=p->next;
      p->next=q;
        }
}
```

```
// tac vu nay chi su dung khi them vao mot bucket co thu tu
void place(int b, int k){
        NODEPTR p,q;
        q=NULL;
        for(p=bucket[b];p!=NULL && k>p->key;p=p->next)
                q=p;
        if(q==NULL){
        push(b,k);
        }else{
        insafter(q,k);
        }

}

//them mot node co khoa la k vao trong bang bam
void insert(int k){
        int b;
        b=hashfunc(k);
        place(b,k);
}

//Tac vu tim kiem mot khoa trong bang bam
int search(int k){
        NODEPTR p;
        int b;
        b=hashfunc(k);
        p=bucket[b];
        while(k>p->key&&p!=NULL)
                p=p->next;
        if(p==NULL || k!=p->key)
                return -1;
        else
        return b;
}

//Xoa mot node ngay sau node p
int delafter(NODEPTR p){
        NODEPTR q;
        int k;
        if(p==NULL||p->next==NULL){
                printf("\n Khong xoa node duoc");
      return 0;
        }
        q=p->next;
        k=q->key;
        p->next=q->next;
```

```
        freenode(q);
    return k;
}

//Xoa mot phan tu co khoa k ra khoi bang bam
void remove(int k){
        int b;
        NODEPTR p,q;
        b=hashfunc(k);
        p=bucket[b];
        q=p;
        while(p!=NULL&&p->key!=k)
        {
                q=p;
                p=p->next;
        }
        if(p==NULL)
                printf("\n Khong co node co khoa la: %d",b);
        else
                if(p==bucket[b])
                        pop(b);
                else
                        delafter(q);
}

void main(){
        int b,key,i,n,chucnang;
        char c;
        initbucket();
        do{
                printf("\n\n CAC CHUC NANG CUA CHUONG TRINH");
                printf("\n 1: Them mot node vao bang");
                printf("\n 2: Them ngau nhien nhieu node vao bang bam");
                printf("\n 3: Xoa mot node trong bang bam");
                printf("\n 4: Xoa toan bo bang bam");
                printf("\n 5: Duyet bang bam");
                printf("\n 6: Tim kiem tren bang bam");
                printf("\n 0: ket thuc chuong trinh");
                printf("\n chuc nang ban chon: ");
                scanf("%d",&chucnang);

                switch(chucnang){
                        case 1:{
                                printf("\n Them mot node vao trong bang bam");
                                printf("\n Nhap vao khoa cua node can them vao: ");
                                scanf("%d",&key);
```

```
                        insert(key);
                break;
                        }
                        case 2:{
                                printf("\n them mot bang ngau nhien nhieu node vao
bang");
                                printf("\n So node ban muon them: ");
                                scanf("%d",&n);
                                for(i=0;i<n;i++){
                                        key=random(100);
            insert(key);
                                }
                break;
                        }
                        case 3:{
                                printf("\n Xoa mot node tren bang bam");
                                printf("\n Nhap vao khoa cua node can xoa: ");
                                scanf("%d",&key);
                                remove(key);
                break;
                        }
                        case 4:{
                clear();
                break;
                        }
                        case 5:{
                                printf("\n Duyet Bang Bam");
          traverse();
                break;
                        }
                        case 6:{
                                printf("\n Tim kiem mot khoa tren bang bam");
                                printf("\n Khoa can tim: ");
                                scanf("%d",key);
                                b=search(key);
                                if(b==-1)
                                        printf(" Khong thay");
                                else
                printf(" Tim thay trong bucket %d",b);
                                break;
                        }
                }
        }while(chucnang !=0);
}
```

3. BẢNG BĂM VỚI PHƯƠNG PHÁP KẾT NỐI HỢP NHẤT

3.1 Mô tả

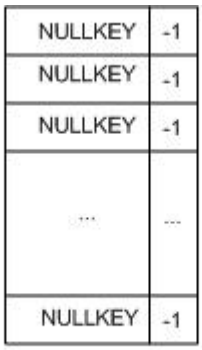
Bảng băm trong trường hợp này được cài đặt bằng danh sách liên kết dùng mảng, có M nút. Các nút bị xung đột địa chỉ được nối kết với nhau qua một danh sách liên kết.

Mỗi nút của bảng băm là một mẫu tin có hai trường:

- Trường key: chứa khoá của nút.
- Trường next: con trỏ chỉ nút kế tiếp nếu có xung đột.

Khi khởi động bảng băm thì tất cả trường key được gán giá trị là NULLKEY, tất cả các trường next được gán là -1.

Hình vẽ sau mô tả bảng băm ngay sau khi khởi động:



Hình minh hoạ bảng băm khi khởi động

Khi thêm một nút có khoá key vào bảng băm, hàm băm f(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M – 1.

- Nếu chưa bị xung đột thì thêm nút mới tại địa chỉ i này.
- Nếu bị xung đột thì nút mới được cấp phát là nút trống phía cuối mảng. Cập nhật liên kết next sao cho các nút bị xung đột hình thành một danh sách liên kết.

Khi tìm một nút có khoá key trong bảng băm, hàm băm f(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M – 1, tìm nút khoá key trong danh sách liên kết xuất phát từ địa chỉ i.

Minh hoạ

Sau đây là minh hoạ cho bảng băm có tập khoá là số tự nhiên, tập địa chỉ có 10 địa chỉ (M=10), chọn hàm băm f(key)=key % 10. Hình vẽ sau đây minh hoạ cho tiến trình thêm các nút 10, 15, 26, 30, 25, 35 vào bảng băm.

Hình (a): Sau khi thêm 3 nút 10, 15, 26 vào bảng băm – lúc này chưa bị xung đột.

Hình (b): Thêm nút 30 vào bảng băm - bị xung đột tại địa chỉ 0 – nút 30 được cấp phát tại địa chỉ 9, trường next của nút tại địa chỉ 0 được gán là 9.

Hình (c): Thêm nút 25 vào bảng băm - bị xung đột tại địa chỉ 5 – nút 25 được cấp phát tại địa chỉ 8, trường next của nút tại địa chỉ 5 được gán là 8.

Hình (d): Thêm nút 35 vào bảng băm - bị xung đột tại địa chỉ 5 và địa chỉ 8 – nút 35 được cấp phát tại địa chỉ 7, trường next của nút tại địa chỉ 8 được gán là 7.

(a)

0	10	-1
1		-1
2		-1
3		-1
4		-1
5	15	-1
6	26	-1
7		-1
8		-1
9		-1

(b)

0	10	9
1		-1
2		-1
3		-1
4		-1
5	15	-1
6	26	-1
7		-1
8		-1
9	30	-1

(c)

0	10	9
1		-1
2		-1
3		-1
4		-1
5	15	8
6	26	-1
7		-1
8	25	-1
9	30	-1

(d)

0	10	9
1		-1
2		-1
3		-1
4		-1
5	15	8
6	26	-1
7	35	-1
8	25	7
9	30	-1

Hình minh hoạ việc thêm các khoá vào bảng băm

3.2 Cài đặt

3.2.1 Khai báo cấu trúc bảng băm

```
#define M 10
struct node{
        int key;
   int next;
};
struct node hashtable[M];
int avail;
```

3.2.2 Tác vụ khởi động cho bảng băm

```
void initialize(){
        int i;
        for(i=0;i<M;i++){
                hashtable[i].key=NULLKEY;
                hashtable[i].next=-1;
        }
   avail=M-1;
}
```

3.2.3 Tác vụ tìm kiếm

```
int search(int k){
        int i;
        i=hashfunc(k);
        while(k!=hashtable[i].key && i !=-1)
                i=hashtable[i].next;
        if(k==hashtable[i].key)
                return i;
        else
        return M;
}
```

3.2.4 Tác vụ thêm một phần tử vào bảng băm

```
int insert(int k){
        int i,j;
        i=search(k);
        if(i!=M){
                printf("\n khoa %d bi trung, khong them vao duoc",k);
      return i;
        }
        i=hashfunc(k);
        while(hashtable[i].next>=0)
                i=hashtable[i].next;
        if(hashtable[i].key==NULLKEY)
                j=i;//khong co su dung do, first time
        else{
                j=getempty();
                if(j <0){
                        printf("\n Bang bam bi day khong them vao duoc");
                        return j;
                }else{
        hashtable[i].next=j;
                }
        }
        hashtable[j].key=k;
        return j;
}
```

3.2.5 Tác vụ duyệt bảng băm

```
void viewtable(){
        int i;
        for(i=0;i<M;i++){
                printf("\n table[%2d]: %4d %4d",i,hashtable[i].key,hashtable[i].next);
        }
}
```

3.3 Chương trình minh hoạ

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

#define TRUE 1
#define FALSE 0
#define NULLKEY -1
#define M 10

struct node{
        int key;
   int next;
```

```
};

struct node hashtable[M];
int avail;

//ham bam
int hashfunc(int key){
	return (key %M);
}

//khoi dong bang bam
void initialize(){
	int i;
	for(i=0;i<M;i++){
		hashtable[i].key=NULLKEY;
		hashtable[i].next=-1;
	}
   avail=M-1;
}

//kiem tra bang bam co rong hay khong?
int empty(){
	int i;
	for(i=0;i<M;i++){
		if (hashtable[i].key!=NULLKEY)
			return FALSE;
	}
   return TRUE;
}

//Xem chi tiet thong tin bang bam
void viewtable(){
	int i;
	for(i=0;i<M;i++){
		printf("\n table[%2d]: %4d %4d",i,hashtable[i].key,hashtable[i].next);
	}
}

//tim mot khoa co trong bang bam hay khong, tim thay tra ve dia chi
//khong thayy tra ve M
int search(int k){
	int i;
	i=hashfunc(k);
	while(k!=hashtable[i].key && i !=-1)
		i=hashtable[i].next;
	if(k==hashtable[i].key)
```

```
                return i;
        else
        return M;
}

//chon node con trong phia duoi bang hash de cap nhat khi xay ra xung dot
int getempty(){
        while(hashtable[avail].key !=NULLKEY)
                avail --;
   return avail;
}

//them mot node co khoa k vao bang bam
int insert(int k){
        int i,j;
        i=search(k);
        if(i!=M){
                printf("\n khoa %d bi trung, khong them vao duoc",k);
      return i;
        }
        i=hashfunc(k);
        while(hashtable[i].next>=0)
                i=hashtable[i].next;
        if(hashtable[i].key==NULLKEY)
                j=i;//khong co su dung do, first time
        else{
                j=getempty();
                if(j <0){
                        printf("\n Bang bam bi day khong them vao duoc");
                        return j;
                }else{
        hashtable[i].next=j;
                }
        }
        hashtable[j].key=k;
        return j;
}

void main(){
        int i,n;
        int key,chucnang;
        initialize();

        do{
                printf("\n\n CAC CHUC NANG CUA CHUONG TRINH");
                printf("\n 1: Them node moi vao bang bam");
```

```
		printf("\n 2: Them ngau nhien nhieu node vao bang bam");
		printf("\n 3: Xoa toan bo bang bam");
		printf("\n 4: Xem chi tiet bang bam");
		printf("\n 5: Tim kiem tren bang bam");
		printf("\n 0: Ket thuc chuong trinh");

		printf("\n\n Chuc nang ban chon: ");
		scanf("%d",&chucnang);

		switch(chucnang){
			case 1:{
				printf("\n THEM MOT NODE MOI VAO BANG BAM");
				printf("\n Khoa cua node moi: ");
				scanf("%d",&key);
				insert(key);
		break;
			}
			case 2:{
				printf("\n THEM NGAU NHIEN NHIEU NODE VAO
BANG BAM");
				printf("\n Ban muon them vao bao nhieu node: ");
				scanf("%d",&n);
				for(i=0;i<n;i++){
					key=random(1000);
	  insert(key);
				}
				break;
			}
			case 3:{
				printf("\n XOA TOAN BO BANG BAM");
	initialize();
		break;
			}
			case 4:{
				printf("\n CHI TIET THONG TIN BANG BAM");
				viewtable();
		break;
			}
			case 5:{
				printf("\n TIM KIEM KHOA TREN BANG BAM");
				printf("\n Khoa can tim: ");
				scanf("%d",&key);
	i=search(key);
				if(i==M){
		printf("\n KHONG THAY");
				}
```

```
                    else{
            printf("Tim thay tai dia chi %d cua bang bam ",i);
                    }
                    break;

                }
            }
        }while(chucnang!=0);
}
```

4. BẢNG BĂM VỚI PHƯƠNG PHÁP DÒ TUYẾN TÍNH

4.1 Mô tả

Bảng băm trong trường hợp này được cài đặt bằng một danh sách kề có M nút, mỗi nút của bảng băm là một mẩu tin có một trường key để chứa khoá của nút.

Khi thêm nút có khoá key vào bảng băm, hàm băm f(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1:

- Nếu chưa bị xung đột thì thêm nút mới tại địa chỉ i này.
- Nếu bị xung đột thì hàm băm lần 1 f_1 sẽ xét địa chỉ kế tiếp, nếu lại bị xung đột thì hàm băm lại lần 2 f_2 sẽ xét địa chỉ kế tiếp nữa… quá trình cứ thế cho đến khi nào tìm được địa chỉ trống và thêm nút vào địa chỉ này.

Khi tìm một nút có khoá key trong bảng băm, hàm băm f(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M – 1, tìm nút khoá key trong khối đặc chứa các nút xuất phát từ địa chỉ i.

Hàm băm lại của phương pháp dò tìm tuyến tính là truy xuất địa chỉ kế tiếp. Hàm băm lại được biểu diễn bằng công thức sau:

f(key)=(f(key)+i)%M

Minh hoạ:

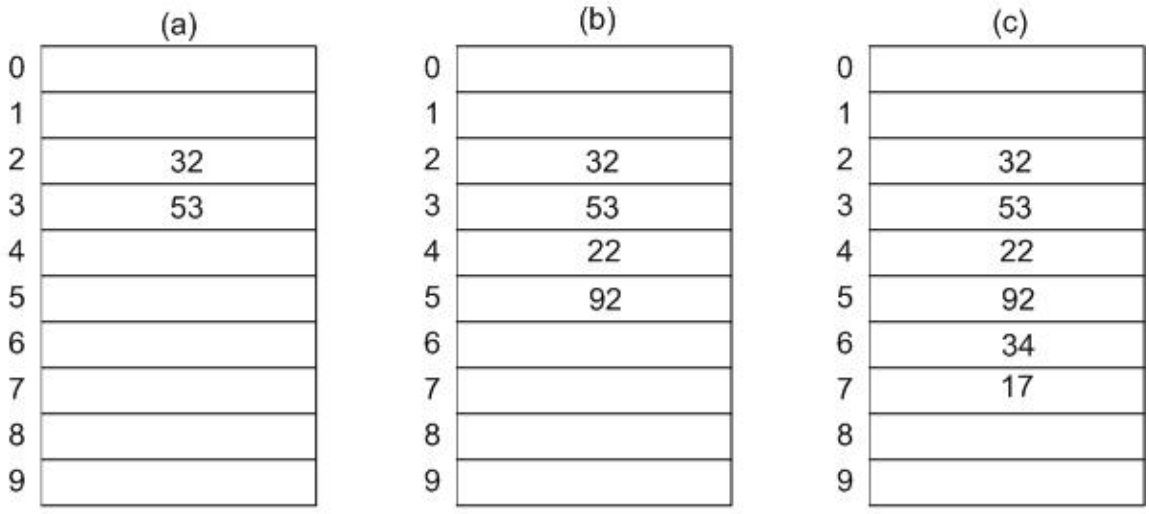
Sau đây là minh hoạ cho bảng băm có tập khoá là tập số tự nhiên, tập địa chỉ có 10 địa chỉ, chọn hàm băm f(key)=key %10.

Hình vẽ sau miêu tả tiến trình thêm các nút 32, 53, 22, 92, 17, 34 vào bảng băm.

Hình (a): Sau khi thêm 2 nút 32 và 53 vào bảng băm – lúc này chưa bị xung đột.

Hình (b): Thêm nút 22 và 92 vào bảng băm - bị xung đột tại địa chỉ 2, nút 22 được cấp phát tại địa chỉ 4, nút 92 được cấp phát tại địa chỉ 5.

Hình (c): thêm nút 17 và 34 vào bảng băm – nút 17 không bị xung đột cấp phát tại địa chỉ 7, nút 34 bị xung đột tại địa chỉ 4, được cấp tại địa chỉ 6.



Hình minh hoạ việc thêm các khoá vào bảng băm

4.2 Cài đặt

4.2.1 Mô tả cấu trúc

```
#define M 10

struct node{
        int key;
};

struct node hashtable[M];
int N;
```

4.2.2 Tác vụ khởi động

```
void initialize(){
        int i;
        for(i=0;i<M; i++){
                hashtable[i].key=NULLKEY;
        }
   N=0;
}
```

4.2.3 Tác vụ tìm kiếm

```
int search(int k){
        int i;
        i=hashfunc(k);
        while(hashtable[i].key!=k&&hashtable[i].key!=NULLKEY){
                i=i+1;
                if(i>=M)
                        i=i-M;
        }
        if(hashtable[i].key==k)
                return i;
        else
```

```
        return M;
}
```

4.2.4 Tác vụ thêm một phần tử vào bảng băm

```
int insert(int k){
        int i,j;
        if(full()){
                printf("\n Bang bam bi day, khong them vao duoc");
      return M;
        }
        i=hashfunc(k);
        while(hashtable[i].key !=NULLKEY){
                i++;
                if(i>=M)
        i=i-M;
        }
        hashtable[i].key=k;
        N++;
   return i;
}

void viewtable(){
        int i;
        for(i=0;i<M;i++){
        printf("\n table[%2d]: %4d",i,hashtable[i].key);
        }
}
```

4.3 Chương trình minh hoạ

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#define TRUE 1
#define FALSE 0
#define NULLKEY -1
#define M 10

struct node{
        int key;
};

struct node hashtable[M];
int N;
```

```
int hashfunc(int key){
	return (key %M);
}

void initialize(){
	int i;
	for(i=0;i<M; i++){
		hashtable[i].key=NULLKEY;
	}
   N=0;
}

//tac vu kiem tra bang bam co rong hay khong
int empty(){
	if(N==0)
		return TRUE;
	else
	return FALSE;
}

//tac vu kiem tra bang bam co day:
int full(){
	if(N==M-1)
		return TRUE;
	else
	return FALSE;
}

//tac vu tim 1 key, tim thay tra ve index, khong thay tra ve M
int search(int k){
	int i;
	i=hashfunc(k);
	while(hashtable[i].key!=k&&hashtable[i].key!=NULLKEY){
		i=i+1;
		if(i>=M)
			i=i-M;
	}
	if(hashtable[i].key==k)
		return i;
	else
	return M;
}

//them khoa k vao bang bam
int insert(int k){
	int i,j;
```

```
        if(full()){
                printf("\n Bang bam bi day, khong them vao duoc");
      return M;
        }
        i=hashfunc(k);
        while(hashtable[i].key !=NULLKEY){
                i++;
                if(i>=M)
        i=i-M;
        }
        hashtable[i].key=k;
        N++;
   return i;
}

void viewtable(){
        int i;
        for(i=0;i<M;i++){
        printf("\n table[%2d]: %4d",i,hashtable[i].key);
        }
}

void main(){
        int i,n;
        int b,key,chucnang;
        initialize();
        do{
                printf("\n\n BANG BAM THEO PHUONG PHAP DO TUYEN TINH");
                printf("\n 1: them mot node moi vao bang bam");
                printf("\n 2: them ngau nhien nhieu node vao bang bam");
                printf("\n 3: Xoa node tren bang bam");
                printf("\n 4: Xoa toan bo cac node tren bang bam");
                printf("\n 5: Xem chi tiet thong tin tren bang bam");
                printf("\n 6: Tim kiem tren bang bam");
                printf("\n 0: Ket thuc chuong trinh");
                printf("\n\n Chuc nang cua ban: ");
                scanf("%d",&chucnang);

                switch(chucnang){
                        case 1:{
                                printf("\n THEM MOT NODE MOI VAO BANG BAM");
                                printf("\n khoa cua node moi: ");
                                scanf("%d",&key);
            insert(key);
                        break;
                                }
```

```
                case 2:{
                        printf("\n THEM NGAU NHIEN NHIEU NODE VAO
BANG");
                        printf("\n Ban muon them vao bao nhieu node: ");
                        scanf("%d",&n);
                        for(i=0;i<n;i++){
                                key=random(1000);
                                insert(key);
                        }
        break;
                }
                case 3:{
        break;
                }
                case 4:{
                        printf("\n XOA TOAN BO CAY TREN BANG BAM");
    initialize();
                        break;
                }
                case 5:{
                        printf("\n THONG TIN CHI TIET TREN BANG BAM");
    viewtable();
        break;
                }
                case 6:{
                        printf("\n TIM KIEM TREN BANG BAM");
                        printf("\n khoa can tim: ");
                        scanf("%d",key);
                        if(search(key)==M)
                                printf("\n Khong thay");
                        else
        printf("\n Tim thay khoa trong bang bam tai dia chi: %d trong
bang",search(key));
        break;
                }
        }
    }while(chucnang!=0);
}
```

BÀI TẬP

1. Viết một chương trình hiện thực từ điển Anh - Việt, chương trình có cài đặt bảng băm với phương pháp nối kết trực tiếp. Mỗi nút của bảng băm có khai báo các trường sau:

- Trường word là khoá chứa một từ tiếng anh.
- Trường mean là nghĩa tiếng Việt.
- Trường next là con trỏ chỉ nút kế bị xung đột cùng địa chỉ.

Tập khoá là một chuỗi tiếng anh, tập địa chỉ có 26 chữ cái. Chọn hàm băm sau cho khoá bắt đầu bằng ký tự a được băm vào địa chỉ 0, b băm vào địa chỉ 1,…, z băm vào địa chỉ 25. Chương trình có những chức năng như sau:

- Nhập vào một từ
- Xem từ điển theo ký tự đầu.
- Xem toàn bộ từ điển.
- Tra từ điển.
- Xoá một từ.
- Xóa toàn bộ từ điển.
- Toát khỏi chương trình.

2. Viết chương trình xem thông tin về một sinh viên qua mã số sinh viên.
Thông tin về tất cả sinh viên chứa trong tập tin văn bản, mỗi sinh viên chiếm một dòng văn bản gồm MSSV, họ, tên và điểm các môn học. Chương trình sẽ đọc tập tin văn bản này để tạo ra bảng băm. Mỗi nút của bảng băm bao gồm trường MSSV dùng làm khoá, trường line cho biết vị trí dòng văn bản của sinh viên. Khi truy xuất thông tin về một sinh viên chúng ta nhập MSSV, qua bảng băm chúng ta xác định được vị trí dòng văn bản và đọc thông tin sinh viên qua dòng văn bản này.
Chương trình có các chức năng như sau:

- Xem tất cả MSSV
- Xem MSSV trong khoảng từ … đến …
- Xem thông tin về sinh viên qua MSSV.

Chương 7:

SẮP XẾP

1. GIỚI THIỆU VỀ BÀI TOÁN SẮP XẾP

Sắp xếp các nút của một cấu trúc theo thứ tự tăng dần (hay giảm dần) là một công việc được thực hiện thường xuyên. Với một cấu trúc đã được sắp xếp chúng ta rất thuận tiện khi thực hiện các tác vụ trên cấu trúc như tìm kiếm, trích lọc duyệt cấu trúc…

Có hai giải thuật sắp xếp được dùng phổ biến trong khoa học máy tính là sắp xếp dữ liệu trên bộ nhớ trong (internal sort) và sắp xếp dữ liệu trên bộ nhớ ngoài (external sort).

Với sắp xếp dữ liệu trên bộ nhớ trong thì toàn bộ dữ liệu cần sắp xếp được đưa vào bộ nhớ trong, do vậy kích thước dữ liệu cần sắp xếp không lớn, tuy nhiên thời gian sắp xếp được thực hiện rất nhanh.

Với sắp xếp dữ liệu trên bộ nhớ ngoài thì chỉ một phần nhỏ dữ liệu cần sắp xếp được đưa vào bộ nhớ trong, phần lớn dữ liệu được lưu trữ ở bộ nhớ ngoài như đĩa từ, băng từ, đĩa cứng… kích thước dữ liệu cần được sắp xếp lúc này rất lớn và thời gian sắp xếp rất chậm.

Để phân tích đánh giá giải thuật sắp xếp, chúng ta cần thẩm định giải thuật chiếm dụng bao nhiêu vùng nhớ, giải thuật chạy nhanh hay chạy chậm. Hai tiêu chí chính dùng để phân tích một giải thuật sắp xếp là:

- Sự chiếm dụng bộ nhớ của giải thuật.
- Thời gian thực hiện của giải thuật.

2. SẮP XẾP BỘ NHỚ TRONG

Có rất nhiều giải thuật để hiện thực việc sắp xếp dữ liệu trong bộ nhớ trong. Ở phần này ta xét các phương pháp: bubble sort, simple selection sort, simple insertion sort, quicksort và merge sort.

2.1 Giải thuật bubble sort

2.1.1 Mô tả phương pháp

Giải thuật này sẽ duyệt danh sách nhiều lần, trong mỗi lần duyệt sẽ lần lượt so sánh từng cập nút thứ i và thứ i + 1 và đổi chỗ hai nút này nếu chúng không đúng thứ tự.
Minh hoạ:
Dùng phương pháp bubble sort để sắp xếp lại danh sách dưới đây.
25 55 45 40 10 90 85 35
Bảng sau đây minh hoạ quá trình so sánh và đổi chỗ cho lần duyệt đầu tiên

i	Nodes[i] (trước)	Nodes[i+1] (trước)	Kiểm tra nodes[i]>nodes[i+1]?	Nodes[i] (sau)	Nodes[i+1] (sau)
0	25	55	Sai ->không đổi chỗ	25	55
1	55	45	Đúng ->đổi chỗ	45	55
2	55	40	Đúng ->đổi chỗ	40	55
3	55	10	Đúng ->đổi chỗ	10	55
4	55	90	Sai ->không đổi chỗ	55	90

5	90	85	Đúng ->đổi chỗ	85	90
6	90	35	Đúng ->đổi chỗ	35	90

Nếu dùng phương pháp bubble sort để sắp xếp danh sách có n nút:

- Sau lần duyệt thứ 1, nút lớn nhất được định vị đúng chỗ.
- Sau lần duyệt thứ 2, nút thứ 2 được định vị đúng chỗ.
- Sau lần duyệt thứ n-1 thì n nút trong danh sách sẽ được sắp xếp thứ tự.

Sự biến đổi của danh sách qua các lần duyệt được mô tả trong bảng dưới đây.

lần duyệt	Dữ liệu							
Ban đầu	25	55	45	40	10	90	85	35
1	25	45	40	10	55	85	35	90
2	25	40	10	45	55	35	85	90
3	25	10	40	45	35	55	85	90
4	10	25	40	35	45	55	85	90
5	10	25	35	40	45	55	85	90
6	10	25	35	40	45	55	85	90
7	10	25	35	40	45	55	85	90

2.1.2 Cài đặt giải thuật bubble sort

```
void bubblesort(int nodes[], int n){
        int temp, i,j;
        int doicho=TRUE;
        for(i=1; i<n&&doicho==TRUE;i++){
                doicho=FALSE;
                for(j=0;j<n-i;j++)
                        if(nodes[j]>nodes[j+1]){
                                doicho=TRUE;
                                temp=nodes[j];
                                nodes[j]=nodes[j+1];
                                nodes[j+1]=temp;
                        }
        }
}
```

2.2 Giải thuật simple selection sort

2.2.1 Mô tả phương pháp

Phương pháp này lần lượt chọn nút nhỏ nhất cho các vị trí 0, 1, 2,…,n-1. Cụ thể:

Lần chọn thứ 0:

Dò tìm trong khoảng vị trí từ 0 đến n-1 để xác định nút nhỏ nhất tại vị trí min_0.

Đổi chỗ hai nút tại vị trí min_0 và 0.

Lần chọn thứ 1:

Dò tìm trong khoảng vị trí từ 1 đến n-1 để xác định nút nhỏ nhất tại vị trí min_1.

Đổi chỗ hai nút tại vị trí min_1 và 1.

Lần chọn thứ i:
Dò tìm trong khoảng vị trí từ i đến n-i để xác định nút nhỏ nhất tại vị trí min_i.
Đổi chỗ hai nút tại vị trí min_i và i.

Lần chọn thứ n-2 (lần chọn cuối cùng):
Dò tìm trong khoảng từ vị trí n-2 đến n-1 để xác định nút nhỏ nhất tại vị trí min_{n-2}.
Đổi chỗ hai nút tại vị trí min_{n-2} và vị trí n-2.

Minh hoạ: dùng giải thuật simple selection sort để sắp xếp cho danh sách sau:
25 55 45 40 10 90 85 35

Lần chọn	Dữ liệu							
Ban đầu	25	55	45	40	10	90	85	35
0	10	55	45	40	25	90	85	35
1	10	25	45	40	55	90	85	35
2	10	25	35	40	55	90	85	45
3	10	25	35	40	55	90	85	45
4	10	25	35	40	45	90	85	55
5	10	25	35	40	45	55	85	90
6	10	25	35	40	45	55	85	90

2.2.2Cài đặt giải thuật simple selection sort

```
void selectionsort(int nodes[], int n){
        int i,j,min,vitrimin;
        for(i=0;i<n-1;i++){
                min=nodes[i];
                vitrimin=i;
                for(j=i+1;j<=n-1;j++)
                        if(min >nodes[j]){
                                min=nodes[j];
                                vitrimin=j;
                        }
                nodes[vitrimin]=nodes[i];
                nodes[i]=min;
        }
}
```

2.3 Giải thuật simple insertion sort
2.3.1 Mô tả phương pháp
Phương pháp này sẽ lần lược chèn các nút vào danh sách đã có thứ tự:

- Xem danh sách đầu tiên đã có thứ tự chỉ là 1 nút nodes[0].
- Lần chèn 1: chèn nodes[1] vào đúng vị trí chúng ta được danh sách đã có thứ tự có đúng hai nút là nodes[0] và nodes[1].
- Lần chèn 2: chèn nodes[2] vào đúng vị trí chúng ta được danh sách đã có thứ tự có đúng 3 nút là nodes[0], nodes[1] và nodes[2].
- …

- Lần chèn n-1: chèn nodes[n-1] vào đúng vị trí chúng ta được danh sách cuối cùng đã có thứ tự có n nút là nodes[0], nodes[1],…,nodes[n-1].

Minh hoạ: dùng phương pháp Simple Insertion Sort sắp xếp danh sách sau:
25 55 45 40 10 90 85 35

Lần chèn	Danh sách trước lần chèn	Nút chèn vào danh sách nodes[i]	Danh sách sau khi chèn
1	25	55	25,55
2	25,55	45	25,45,55
3	25,45,55	40	25,40,45,55
4	25,40,45,55	10	10, 25,40,45,55
5	10, 25,40,45,55	90	10, 25,40,45,55,90
6	10, 25,40,45,55,90	85	10, 25,40,45,55,85,90
7	10, 25,40,45,55,85,90	35	10, 25,35,40,45,55,85,90

2.3.2 Cài đặt giải thuật simple insertion sort

```
void simpleinsertionsort(int nodes[],int n){
        int x;
        for(i=1;i<n;i++){
                x=nodes[i];
                for(j=i-1;j>=0&&x<nodes[j];j--)
                        nodes[j+1]=nodes[j];
                nodes[j+1]=x;
        }
}
```

2.4 Giải thuật quick sort

2.4.1 Mô tả giải thuật

Quick Sort là giải thuật rất hiệu quả, rất thông dụng và thời gian chạy của giải thuật trong khoảng O(nlogn). Nội dung của giải thuật này như sau:

- Chọn một nút bất kỳ trong danh sách (giả sử là nút đầu tiên) gọi là nút làm trục (pivot node), xác định vị trí của nút này trong danh sách gọi là vị trí pivot.
- Tiếp theo chúng ta phân hoạch các nút còn lại trong danh sách cần sắp xếp sao cho các nút từ vị trí 0 đến vị trí pivot -1 đều có nội dung nhỏ hơn hoặc bằng nút làm trục, các nút từ vị trí pivot + 1 đến n-1 đều có nội dung lớn hơn hoặc bằng nút làm trục.
- Quá trình lại tiếp tục như thế với hai danh sách con từ vị trí 0 đến vị trí pivot -1 và từ vị trí pivot + 1 đến vị trí n-1. Sau cùng chúng ta sẽ được danh sách đã có thứ tự.

Minh hoạ:

Dùng giải thuật quick sort sắp xếp danh sách sau:
25 55 45 40 10 90 85 35

Nút làm trục	Dữ liệu							
	25	55	45	40	10	90	85	35
25	10	25	55	45	40	90	85	35
10	10	25	55	45	40	90	85	35
55	10	25	45	40	35	55	90	85
45	10	25	40	35	45	55	90	85
40	10	25	35	40	45	55	90	85
35	10	25	35	40	45	55	90	85
90	10	25	35	40	45	55	85	90
85	10	25	35	40	45	55	85	90

2.4.2 Cài đặt giải thuật

```
void quicksort(int nodes[],int low, int up) {
        int pivot;
        if(low>=up)
                return;
        if(low<up){
                partition(nodes,low,up,&pivot);
                quicksort(nodes,low,pivot-1);
                quicksort(nodes,pivot+1,up);
        }
}

void partition(nodes[], int low, int up, int* pivot){
        int nuttruc,l,u,temp;
        nuttruc=nodes[low];
        l=low;
        u=up;
        while(l<u){
                while(nodes[l]<=nuttruc && l<up)
                        l++;
                while(nodes[u] >nuttruc) u--;
                if(l<u){
                        temp=nodes[l];
                        nodes[l]=nodes[u];
                        nodes[u]=temp;
                }
        }
        nodes[low]=nodes[u];
        nodes[u]=nuttruc;
        *pivot=u;
}
```

2.5 Giải thuật merge sort

2.5.1 Mô tả giải thuật

Là phương pháp sắp xếp bằng cách trộn hai danh sách đã có thứ tự thành một danh sách đã có thứ tự. Phương pháp Merge Sort tiến hành qua nhiều bước trộn như sau:

Bước 1:

Xem danh sách cần sắp xếp như n danh sách con đã có thứ tự, mỗi danh sách con chỉ có 1 nút.

Trộn từng cặp hai danh sách con kế cận chúng ta được n/2 danh sách con đã có thứ tự, mỗi danh sách con có 2 nút.

Bước 2:

Xem danh sách cần sắp xếp như n/2 danh sách con đã có thứ tự, mỗi danh sách con có 2 nút.

Trộn từng cặp hai danh sach con kế cận chúng ta được n/4 danh sách con đã có thứ tự, mỗi danh sách con có 4 nút.

…

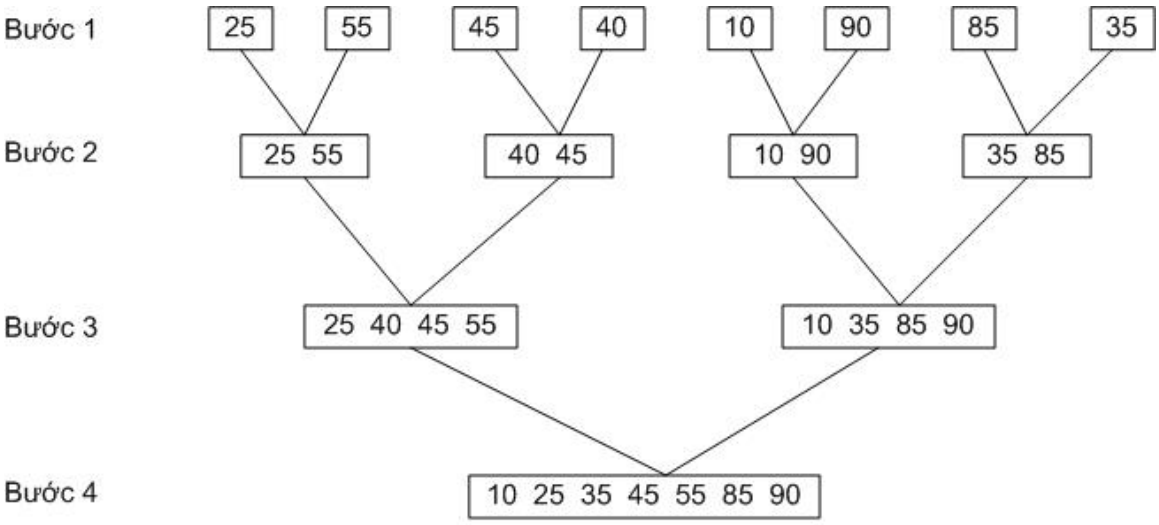
Quá trìnnh cứ tiếp tục diễn ra như thế cho đến khi được một danh sách có n nút.

Nếu danh sách có n nút thì ta phải tiến hành $\log_2 n$ bước trộn.

Minh hoạ:

Dùng phương pháp merge sort để sắp xếp danh sách như sau:

25 55 45 40 10 90 85 35



Hình minh hoạ giải thuật merge sort

2.5.2 Cài đăt giải thuật Merge Sort

```
void mergesort(int nodes[], int n){
        int i,j,k,low1, up1, low2, up2,size;
        int dstam[MAXLIST];
        size=1;
        while(size<n){
                low1=0;
                k=0;
                while(low1+size<n){
```

```
                low2=low1+size;
                up1=low2-1;
                up2=(low2+size-1<n)? low2+size-1:n-1;
                for(i=low1,j=low2;i<=up1 && j<=up2;k++)
                        if(nodes[i]<=nodes[j])
                                dstam[k]=nodes[i++];
                        else
                                dstam[k]=nodes[j++];
                for(;i<=up1;k++)
                        dstam[k]=nodes[i++];
                for(;j<=up2;k++)
                        dstam[k]=nodes[j++];
                low1=up2+1;
        }
        for(i=low1;k<n;i++)
                dstam[k++]=nodes[i];
        for(i=0;i<n;i++)
                nodes[i]=dstam[i];
        size *=2;
    }
}
```

3. SẮP XẾP BỘ NHỚ NGOÀI

3.1 Giải thuật trộn Run

Run là một dãy liên tiếp các phần tử được sắp thứ tự.

Ví dụ: *1 2 3 4 5 là một run gồm có 5 phần tử*

Chiều dài run chính là số phần tử trong Run. Chẳng hạn, run trong ví dụ trên có chiều dài là 5. Như vậy, mỗi phần tử của dãy có thể xem như là 1 run có chiều dài là 1. Hay nói khác đi, mỗi phần tử của dãy chính là một run có chiều dài bằng 1.Việc tạo ra một run mới từ 2 run ban đầu gọi là trộn run (merge). Hiển nhiên, run được tạo từ hai run ban đầu là một dãy các phần tử đã được sắp thứ tự.

Giải thuật sắp xếp tập tin bằng phương pháp trộn run có thể tóm lược như sau:

Input: f0 là tập tin cần sắp thứ tự.

Output: f0 là tập tin đã được sắp thứ tự.

Gọi f1, f2 là 2 tập tin trộn.

Các tập tin f0, f1, f2 có thể là các tập tin tuần tự (text file) hay có thể là các tập tin truy xuất ngẫu nhiên (File of <kiểu>)

Bước 1:

- Giả sử các phần tử trên f0 là:
  24 12 67 33 58 42 11 34 29 31
- f1 ban đầu rỗng, và f2 ban đầu cũng rỗng.
- Thực hiện phân bố m=1 phần tử lần lượt từ f0 vào f1 và f2:
  f1: 24 67 58 11 29
  f0: 24 12 67 33 58 42 11 34 29 31

f2: 12 33 42 34 31
- Trộn f1, f2 thành f0:
f0: 12 24 33 67 42 58 11 34 29 31

Bước 2:
-Phân bố m=2 phần tử lần lượt từ f0 vào f1 và f2:
f1: 12 24 42 58 29 31
f0: 12 24 33 67 42 58 11 34 29 31
f2: 33 67 11 34
- Trộn f1, f2 thành f0:
f1: 12 24 42 58 29 31
f0: 12 24 33 67 11 34 42 58 29 31
f2: 33 67 11 34

Bước 3:
- Tương tự bước 2, phân bố m=4 phần tử lần lượt từ f0 vào f1 và f2, kết quả thu được như sau:
f1: 12 24 33 67 29 31
f2: 11 34 42 58
- Trộn f1, f2 thành f0:
f0: 11 12 24 33 34 42 58 67 29 31

Bước 4:
- Phân bố m=8 phần tử lần lượt từ f0 vào f1 và f2:
f1: 11 12 24 33 34 42 58 67
f2: 29 31
- Trộn f1, f2 thành f0:
f0: 11 12 24 29 31 33 34 42 58 67 29
Bước 5:
Lặp lại tương tự các bước trên, cho đến khi chiều dài m của run cần phân bổ lớn hơn chiều dài n của f0 thì dừng. Lúc này f0 đã được sắp thứ tự xong.

3.2 Chương trình minh hoạ giải thuật trộn run

```
#include <stdio.h>
int p,n;
void tao_file()
{
        int i,x;
        FILE *fp;
        fp=fopen("c:\\bang.int","wb");
        printf("Cho biet so phan tu : ");
        scanf("%d",&n);
        for (i=0;i<n;i++)
        {
                scanf("%d",&x);
                fprintf(fp,"%3d",x);
```

```
	}
	fclose(fp);
}

void xuat_file()
{
	int x,i;
	FILE *fp;
	fp=fopen("c:\\bang.int","rb");
	i=0;
	while (i<n)
	{
		fscanf(fp,"%d",&x);
		printf("%3d",x);
		i++;
	}
	fclose(fp);
}

//chia cac phan tu cho 2 file b va c moi lan p phan tu
void chia(FILE *a,FILE *b,FILE *c,int p)
{
	int dem,x;
	a=fopen("c:\\bang.int","rb");
	b=fopen("c:\\bang1.int","wb");
	c=fopen("c:\\bang2.int","wb");
	while (!feof(a))
	{
		/*Chia p phan tu cho b*/
		dem=0;
		while ((dem<p) && (!feof(a)))
		{
			fscanf(a,"%3d",&x);
			fprintf(b,"%3d",x);
			dem++;
		}
		/*Chia p phan tu cho c*/
		dem=0;
		while ((dem<p) && (!feof(a)))
		{
			fscanf(a,"%3d",&x);
			fprintf(c,"%3d",x);
			dem++;
		}
	}
	fclose(a); fclose(b); fclose(c);
```

```
}

/*Tron p phan tu tren b voi p phan tu tren c thanh 2*p phan tu tren a
cho den khi file b hoac c het.  */
void tron(FILE *b,FILE *c,FILE *a,int p)
{
        int stop,x,y,l,r;
        a=fopen("c:\\bang.int","wb");
        b=fopen("c:\\bang1.int","rb");
        c=fopen("c:\\bang2.int","rb");
        while ((!feof(b)) && (!feof(c)))
        {
                l=0;/*so phan tu cua b da ghi len a*/
                r=0;/*so phan tu cua c da ghi len a*/
                fscanf(b,"%3d",&x);
                fscanf(c,"%3d",&y);
                stop=0;
                while ((l!=p) && (r!=p) && (!stop))
                {
                        if (x<y)
                        {
                                fprintf(a,"%3d",x);
                                l++;
                                if ((l<p) && (!feof(b)))
                                        /*chua du p phan tu va chua het file b*/
                                        fscanf(b,"%3d",&x);
                                else
                                {
                                        fprintf(a,"%3d",y);
                                        r++;
                                        if (feof(b))  stop=1;
                                }
                        }
                        else
                        {
                                fprintf(a,"%3d",y);
                                r++;
                                if ((r<p) && (!feof(c)))
                                        /*chua du p phan tu va chua het file c*/
                                        fscanf(c,"%3d",&y);
                                else
                                {
                                        fprintf(a,"%3d",x);
                                        l++;
                                        if (feof(c))stop=1;
                                }
```

```
                        }

                }
        }

        /*  Chep phan con lai cua p phan tu tren b len a*/
        while ((!feof(b)) && (l<p))
        {
                fscanf(b,"%3d",&x);
                fprintf(a,"%3d",x);
                l++;
        }
        /*  Chep phan con lai cua p phan tu tren c len a*/
        while ((!feof(c)) && (r<p))
        {
                fscanf(c,"%3d",&y);
                fprintf(a,"%3d",y);
                r++;
        }
//
        if (!feof(b))
        {
                /*chep phan con lai cua b len a*/
                while (!feof(b))
                {
                        fscanf(b,"%3d",&x);
                        fprintf(a,"%3d",x);
                }
        }
        if (!feof(c))
        {
                /*chep phan con lai cua c len a*/
                while (!feof(c))
                {
                        fscanf(c,"%3d",&x);
                        fprintf(a,"%3d",x);
                }
        }
        fclose(a); fclose(b); fclose(c);

}
void main ()
{
        FILE *a,*b,*c;
        tao_file();
        xuat_file();
```

```
        p = 1;
        while (p < n)
        {
                chia(a,b,c,p);
                tron(b,c,a,p);
                p=2*p;
        }
        printf("\n");
        xuat_file();
}
```

3.3 Giải thuật trộn tự nhiên

Trong phương pháp trộn đã trình bày ở trên, giải thuật không tận dụng được chiều dài cực đại của các run trước khi phân bổ; do vậy, việc tối ưu thuật toán chưa được tận dụng.Đặc điểm cơ bản của phương pháp trộn tự nhiên là tận dụng độ dài "tự nhiên" của các run ban đầu; nghĩa là, thực hiện việc trộn các run có độ dài cực đại vơi nhau cho đến khi dãy chỉ bao gồm một run: dãy đã được sắp thứ tự.

Input: f0 là tập tin cần sắp thứ tự.
Output: f0 là tập tin đã được sắp thứ tự.

Lặp Cho đến khi dãy cần sắp chỉ gồm duy nhất một run.

Phân bố:
- Chép một dây con có thứ tự vào tắp tin phụ fi (i>=1). Khi chấm dứt dây con này, biến eor (end of run) có giá trị True.
- Chép dây con có thứ tự kế tiếp vào tập tin phụ kế tiếp fi+1 (xoay vòng).
- Việc phân bố kết thúc khi kết thúc tập tin cần sắp f0.

Trộn:
- Trộn 1 run trong f1 và1 run trong f2 vào f0.
- Việc trộn kết thúc khi duyệt hết f1 và hết f2 (hay nói cách khác, việc trộn kết thúc khi đã có đủ n phần tử cần chép vào f0).

3.4 Chương trình minh hoạ giải thuật trộn tự nhiên

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>

FILE *F0,*F1,*F2;
int M,N,Eor;

/*Bien eor dung de kiem tra ket thuc Run hoac File*/
int X1,X2,X,Y;
```

```
void CreatFile(FILE *Ft,int Num)
/*Tao file co ngau nhien n phan tu* */
{
        randomize();
        Ft=fopen("c:\\bang.int","wb");
        for( int i = 0 ; i < Num ; i++)
   {
        X = random(30);
                fprintf(Ft,"%3d",X);
   }
   fclose(Ft);
}

void ListFile(FILE *Ft)
/*Hien thi noi dung cua file len man hinh */
{
        int X,I=0;
        Ft = fopen("c:\\bang.int","rb");
   while ( I < N )
   {
        fscanf(Ft,"%3d",&X);
      cout<<" "<<X;
      I++;
        }
        printf("\n\n");
   fclose(Ft);
}

/**/

void Copy(FILE *Fi,FILE *Fj)
{
//Doc phan tu X tu Tap tin Fi, ghi X vao Fj
//Eor==1, Neu het Run(tren Fi) hoac het File Fi
        fscanf(Fi,"%3d",&X);
        fprintf(Fj,"%3d",X);
   if( !feof(Fi) )
   {
        fscanf(Fi,"%3d",&Y);
      long curpos = ftell(Fi)-2;
      fseek(Fi, curpos, SEEK_SET);
   }
   if ( feof(Fi) ) Eor = 1;
   else Eor = (X > Y) ? 1 : 0 ;
}
```

```
void CopyRun(FILE *Fi,FILE *Fj)
/*Chep 1 Run tu Fi vao Fj */
{
        do
        Copy(Fi,Fj);
   while ( !Eor);
}
void Distribute()
/*Phan bo luan phien cac Run tu nhien tu F0 vao F1 va F2*/
{
        do
   {
        CopyRun(F0,F1);
      if( !feof(F0) ) CopyRun(F0,F2);
        }while( !feof(F0) );

   fclose(F0);
   fclose(F1);
   fclose(F2);
}

void MergeRun()
/*Tron 1 Run cua F1 va F2 vao F0*/
{
        do
   {
        fscanf(F1,"%3d",&X1);
      long curpos = ftell(F1)-2;
      fseek(F1, curpos, SEEK_SET);
      fscanf(F2,"%3d",&X2);
      curpos = ftell(F2)-2;
      fseek(F2, curpos, SEEK_SET);
      if( X1 <= X2 )
      {
        Copy(F1,F0);
        if (Eor) CopyRun(F2,F0);
      }
      else
      {
        Copy(F2,F0);
        if ( Eor ) CopyRun(F1,F0);
      }
   } while ( !Eor );
}

void Merge()
```

```
/*Tron cac run tu F1 va F2 vao F0*/
{
        while( (!feof(F1)) && (!feof(F2)) )
        {
        MergeRun();
      M++;
   }
   while( !feof(F1) )
   {
        CopyRun(F1,F0);
      M++;
   }
   while( !feof(F2) )
   {
        CopyRun(F2,F0);
      M++;
   }
   fclose(F0);
   fclose(F1);
   fclose(F2);
}

void main()
{
        randomize();
   cout<<" Nhap so phan tu: ";
   cin>>N;
   CreatFile(F0,N);
   ListFile(F0);
   do
   {
                F0=fopen("c:\\bang.int","rb");
      F1=fopen("c:\\bang1.int","wb");
      F2=fopen("c:\\bang2.int","wb");
      Distribute();
      F0=fopen("c:\\bang.int","wb");
      F1=fopen("c:\\bang1.int","rb");
      F2=fopen("c:\\bang2.int","rb");
      M=0;
      Merge();
   }while (M != 1);
   ListFile(F0);
}
```

BÀI TẬP

1. Hiện thực giải thuật quicksort không dùng đệ quy mà dùng stack.

2. Viết chương trình nhập vào danh sách sinh viên được tổ chức thành danh sách liên kết, thông tin của sinh viên bao gồm: mã số sinh viên, họ và tên, điểm trung bình. Sau đó tiến hành xếp hạng cho các sinh viên bằng cách sắp xếp theo thứ tự giảm dần của điểm.

3. Viết chương trình minh hoạ các phương pháp sắp xếp, chương trình có các chức năng chính như sau:

- Nhập ngẫu nhiên n số vào danh sách.
- Chọn phương pháp sắp xếp, sau khi chạy xong, chương trình có báo thời gian chạy.
- Xem danh sách sau khi đã sắp xếp.
- Kết thúc chương trình.

4. Nghiên cứu và hiện thực giải thuật heap sort và shell sort.