Mảng và danh sách

Bởi:
Khoa CNTT ĐHSP KT Hưng Yên

MẢNG

Mảng một chiều, mảng nhiều chiều

Khái niệm

Mảng là một tập hợp có thứ tự gồm một số cố định các phần tử. Không có phép bổ sung phần tử hoặc loại bỏ phần tử được thực hiện.

Các phép toán thao tác trên mảng bao gồm : phép tạo lập (create) mảng, phép tìm kiếm (retrieve) một phần tử của mảng, phép lưu trữ (store) một phần tử của mảng.

Các phần tử của mảng được đặc trưng bởi chỉ số (index) thể hiện thứ tự của các phần tử đó trong mảng.

Mảng bao gồm các loại:

+ Mảng một chiều: Mảng mà mỗi phần tử a_i của nó ứng với một chỉ số i.

Ví dụ : Véc tơ a[i] trong đó 0 = 1 . . n cho biết véc tơ là mảng một chiều gồm có n phần tử.

Khai báo : kiểu phần tử A[0...n]

A: Tên biến mảng; Kiểu phần tử: Chỉ kiểu của các phần tử mảng (integer, real, . . .)

+ Mảng hai chiều: Là mảng mà mỗi phần tử a_{ij} của nó ứng với hai chỉ số i và j

Ví dụ : Ma trận A[i],[j] là mảng 2 chiều có i là chỉ số hàng của ma trận và j là chỉ số cột của ma trận.

i = 0 . . n; j = 0 . . m

n: Số hàng của ma trận; m : số cột của ma trận.

Khai báo : kiểu phần tử A[n][m];

+ Mảng n chiều : Tương tự như mảng 2 chiều.

Cấu trúc lưu trữ của mảng.

Cấu trúc dữ liệu đơn giản nhất dùng địa chỉ tính được để thực hiện lưu trữ và tìm kiếm phần tử, là mảng một chiều hay véc tơ.

Thông thường thì một số từ máy sẽ được dành ra để lưu trữ các phần tử của mảng. Cách lưu trữ này được gọi là cách *lưu trữ kế tiếp* (sequential storage allocation).

Trường hợp một mảng một chiều hay véc tơ có n phần tử của nó có thể lưu trữ được trong một từ máy thì cần phải dành cho nó n từ máy kế tiếp nhau. Do kích thước của véc tơ đã được xác định nên không gian nhớ dành ra cũng được ấn định trước.

Véc tơ A có n phần tử, nếu mỗi phần tử a_i ($0 \le i \le n$) chiếm c từ máy thì nó sẽ được lưu trữ trong cn từ máy kế tiếp như hình vẽ:

a_0	a_1	. . .		a_i	. . .		a_n

cn từ máy kế tiếp nhau

L_0 – Địa chỉ của phần tử a_0

Địa chỉ của a_i được tính bởi công thức:

$Loc(a_i) = L_0 + c * i$

trong đó :

L_0 được gọi là địa chỉ gốc - đó là địa chỉ từ máy đầu tiên trong miền nhớ kế tiếp dành để lưu trữ véc tơ (gọi là véc tơ lưu trữ).

$f(i) = c * i$ gọi là hàm địa chỉ (address function)

Đối với mảng nhiều chiều việc lưu trữ cũng tương tự như vậy nghĩa là vẫn sử dụng một véc tơ lưu trữ kế tiếp như trên.

a_{01}	a_{11}	. . .		a_{ij}	. . .		a_{nm}

Giả sử mỗi phần tử trong ma trận n hàng m cột (mảng nhiều chiều) chiếm một từ máy thì địa chỉ của a_{ij} sẽ được tính bởi công thức tổng quát như sau:

$Loc(a_{ij}) = L_0 + j * n + i$ { theo thứ tự ưu tiên cột (column major order }

Cũng với ma trận n hàng, m cột cách lưu trữ theo thứ tự ưu tiên hàng (row major order) thì công thức tính địa chỉ sẽ là:

$Loc(a_{ij}) = L_0 + i * m + j$

+ Trường hợp cận dưới của chỉ số không phải là 1, nghĩa là ứng với a_{ij} thì $b_1 \le i \le u_1$, $b_2 \le j \le u_2$ thì ta sẽ có công thức tính địa chỉ như sau:

$Loc(a_{ij}) = L_0 + (i - b_1) * (u_2 - b_2 + 1) + (j - b_2)$

vì mỗi hàng có $(u_2 - b_2 + 1)$ phần tử.

Ví dụ : Xét mảng ba chiều B có các phần tử b_{ijk} với $1 \le i \le 2$;

$1 \le j \le 3$; $1 \le k \le 4$; được lưu trữ theo thứ tự ưu tiên hàng thì các phần tử của nó sẽ được sắp đặt kế tiếp như sau:

$b_{111}, b_{112}, b_{113}, b_{114}, b_{121}, b_{122}, b_{123}, b_{124}, b_{131}, b_{132}, b_{133}, b_{134}, b_{211}, b_{212}, b_{213}, b_{214}, b_{221}, b_{222}, b_{223}, b_{224}, b_{231}, b_{232}, b_{233}, b_{234}$.

Công thức tính địa chỉ sẽ là :

$Loc(a_{ijk}) = L_0 + (i - 1) *12 + (j - 1) * 4 + (k - 1)$

VD $Loc(b_{223}) = L_0 + 22$.

Xét trường hợp tổng quát với mảng A n chiều mà các phần tử là :

$A[s_1, s_2, . . ., s_n]$ trong đó $b_i \le s_i \le u_i$ (i = 1, 2, . . ., n), ứng với thứ tự ưu tiên hàng ta có:

$nLoc(A[s_1, s_2, . . ., s_n]) = L_0 + ? p_i(s_i - b_i)$

$i = 1n$

$k = i + 1$với $p_i = \Pi (u_k - b_k + 1)$

đặc biệt $p_n = 1$.

<u>Chú ý</u> :

1. Khi mảng được lưu trữ kế tiếp thì việc truy nhập vào phần tử của mảng được thực hiện trực tiếp dựa vào địa chỉ tính được nên tốc độ nhanh và đồng đều đối với mọi phần tử.
2. Mặc dầu có rất nhiều ứng dụng ở đó mảng có thể được sử dụng để thể hiện mối quan hệ về cấu trúc giữa các phần tử dữ liệu, nhưng không phải không có những trường hợp mà mảng cũng lộ rõ những nhược điểm của nó.

Ví dụ : Xét bài toán tính đa thức của x,y chẳng hạn cộng hai đa thức sau:

$(3x^2 - xy + y^2 + 2y - x)$

$+ (x^2 + 4xy - y^2 + 2x)$

$= (4x^2 + 3xy + 2y + x)$

Ta biết khi thực hiện cộng 2 đa thức ta phải phân biệt được từng số hạng, phân biệt được các biến, hệ số và số mũ.

Để biểu diễn được một đa thức với 2 biến x,y ta có thể dùng ma trận: hệ số của số hạng x^iy^j sẽ được lưu trữ ở phần tử có hàng i cột j của ma trận. Nếu ta hạn chế kích thước của ma trận là n × n thì số mũ cao nhất của x,y chỉ xử lý được với đa thức bậc n-1 thôi.

0 1 2 3 4 0 0 -1 0 02 4 0 0 01 0 0 0 00 0 0 0 00 0 0 0 00 1 2 3 4VD : Với $x^2 + 4xy - y^2 + 2x$ thì ta sẽ sử dụng ma trận 5 × 5 biểu diễn nó sẽ có dạng:

Với cách biểu diễn kiểu này thì việc thực hiện phép cộng hai đa thức chỉ là cộng ma trận mà thôi. Nhưng nó có một số hạn chế : số mũ của đa thức bị hạn chế bởi kích thước của ma trận do đó lớp các đa thức được xử lý bị giới hạn trong một phạm vi hẹp. Mặt khác ma trận biểu diễn có nhiều phần tử bằng 0, dẫn đến sự lãng phí bộ nhớ.

Cấu trúc lưu trữ mảng trên một số ngôn ngữ lập trình

Lưu trữ mảng trong ngôn ngữ lập trình C

Hay như để lưu trữ các từ khóa của ngôn ngữ lập trình C, ta cũng dùng đến một mảng để lưu trữ chúng.

Ví dụ 1: Viết chương trình cho phép nhập 2 ma trận a, b có m dòng n cột, thực hiện phép toán cộng hai ma trận a,b và in ma trận kết quả lên màn hình.

Trong ví dụ này, ta sẽ sử dụng hàm để làm ngắn gọn hơn chương trình của ta. Ta sẽ viết các hàm: nhập 1 ma trận từ bàn phím, hiển thị ma trận lên màn hình, cộng 2 ma trận.

```
#include<conio.h>
#include<stdio.h>
void Nhap(int a[][10],int M,int N)
{
int i,j;
for(i=0;i<M;i++)
for(j=0; j<N; j++){
printf("Phan tu o dong %d cot %d: ",i,j);
scanf("%d",&a[i][j]);
}
}
void InMaTran(int a[][10], int M, int N)
{
int i,j;
for(i=0;i<M;i++){
for(j=0; j< N; j++)
printf("%d ",a[i][j]);
printf("\n");
}
}
/* Cong 2 ma tran A & B ket qua la ma tran C*/
void CongMaTran(int a[][10],int b[][10],int M,int N,int c[][10]){
```

```
int i,j;
for(i=0;i<M;i++)
for(j=0; j<N; j++)
c[i][j]=a[i][j]+b[i][j];
}
int main()
{
int a[10][10], b[10][10], M, N;
int c[10][10];/* Ma tran tong*/
printf("So dong M= "); scanf("%d",&M);
printf("So cot M= "); scanf("%d",&N);
printf("Nhap ma tran A\n");
Nhap(a,M,N);
printf("Nhap ma tran B\n");
Nhap(b,M,N);
printf("Ma tran A: \n");
InMaTran(a,M,N);
printf("Ma tran B: \n");
InMaTran(b,M,N);
CongMaTran(a,b,M,N,c);
printf("Ma tran tong C:\n");
InMaTran(c,M,N);
```

getch();

return 0;

}

Lưu trữ mảng trong ngôn ngữ lập trình C#

Array là một cấu trúc dữ liệu cấu tạo bởi một số biến được gọi là những phần tử mảng. Tất cả các phần tử này đều thuộc một kiểu dữ liệu. Bạn có thể truy xuất phần tử thông qua chỉ số (index). Chỉ số bắt đầu bằng zero.

Có nhiều loại mảng (array): mảng một chiều, mảng nhiều chiều.

Cú pháp :

type[ ] array-name;

thí dụ: int[] myIntegers; // mảng kiểu số nguyên string[] myString ; // mảng kiểu chuỗi chữ Bạn khai báo mảng có chiều dài xác định với từ khoá new như sau: // Create a new array of 32 ints int[] myIntegers = new int[32]; integers[0] = 35;// phần tử đầu tiên có giá trị 35 integers[31] = 432;// phần tử 32 có giá trị 432 Bạn cũng có thể khai báo như sau:

int[] integers; integers = new int[32]; string[] myArray = {"first element", "second element", "third element"};

Làm việc với mảng (Working with Arrays)

Ta có thể tìm được chiều dài của mảng sau nhờ vào thuộc tính Length thí dụ sau : int arrayLength = integers.Length

Nếu các thành phần của mảng là kiểu định nghĩa trước (predefined types), ta có thể sắp xếp tăng dần vào phương thức gọi là static Array.Sort() method: Array.Sort(myArray);

Cuối cùng chúng ta có thể đảo ngược mảng đã có nhờ vào the static Reverse() method: Array.Reverse(myArray); string[] artists = {"Leonardo", "Monet", "Van Gogh",

"Klee"}; Array.Sort(artists); Array.Reverse(artists); foreach (string name in artists) { Console.WriteLine(name); }

Mảng nhiều chiều (Multidimensional Arrays in C#)

Cú pháp :

type[,] array-name;

Thí dụ muốn khai báo một mảng hai chiều gồm hai hàng ba cột với phần tử kiểu nguyên :

int[,] myRectArray = new int[2,3];

Bạn có thể khởi gán mảng xem các ví dụ sau về mảng nhiều chiều:

int[,] myRectArray = new int[,]{ {1,2},{3,4},{5,6},{7,8}}; // mảng 4 hàng 2 cột string[,] beatleName = { {"Lennon","John"}, {"McCartney","Paul"}, {"Harrison","George"}, {"Starkey","Richard"} }; chúng ta có thể sử dụng : string[,,] my3DArray; double [,] matrix = new double[10, 10]; for (int i = 0; i < 10; i++) { for (int j=0; j < 10; j++) matrix[i, j] = 4; }

Mảng jagged

Một loại thứ 2 của mảng nhiều chiều trong C# là Jagged array. Jagged là một mảng mà mỗi phần tử là một mảng với kích thước khác nhau. Những mảng con này phải được khai báo từng mảng con một. Thí dụ sau đây khai báo một mảng jagged hai chiều nghĩa là hai cặp [], gồm 3 hàng mỗi hàng là một mảng một chiều: int[][] a = new int[3][]; a[0] = new int[4]; a[1] = new int[3]; a[2] = new int[1]; Khi dùng mảng jagged ta nên sử dụng phương thức GetLength() để xác định số lượng cột của mảng. Thí dụ sau nói lên điều này: using System; namespace Wrox.ProCSharp.Basics { class MainEntryPoint { static void Main() { // Declare a two-dimension jagged array of authors' names string[][] novelists = new string[3][]; novelists[0] = new string[] { "Fyodor", "Mikhailovich", "Dostoyevsky"}; novelists[1] = new string[] { "James", "Augustine", "Aloysius", "Joyce"}; novelists[2] = new string[] { "Miguel", "de Cervantes", "Saavedra"}; // Loop through each novelist in the array int i; for (i = 0; i < novelists.GetLength(0); i++) { // Loop through each name for the novelist int j; for (j = 0; j < novelists[i].GetLength(0); j++) { // Display current part of name Console.Write(novelists[i][j] + " "); } // Start a new line

for the next novelist Console.Write("\n"); } }
} }

Kết quả chương trình sau khi chạy: csc AuthorNames.cs Microsoft (R) Visual C# .NET Compiler version 7.00.9466 for Microsoft (R) .NET Framework version 1.0.3705 Copyright (C) Microsoft Corporation 2001. All rights reserved. AuthorNames Fyodor Mikhailovich Dostoyevsky James Augustine Aloysius Joyce Miguel de Cervantes Saavedra

DANH SÁCH

Khái niệm danh sách tuyến tính

Danh sách là một tập hợp có thứ tự nhưng bao gồm một số biến động các phần tử (x_1, x_2, . . ., x_n)

nếu n = 0 ta có một danh sách rỗng.

Một danh sách mà quan hệ lân cận được hiển thị gọi là *danh sách tuyến tính* (linear list).

VD: Véc tơ chính là một trường hợp đặc biệt của danh sách tuyến tính xét tại một thời điểm nào đấy.

Danh sách tuyến tính là một danh sách hoặc *rỗng* (không có phần tử nào) hoặc có dạng (a_1, a_2, . . ., a_n) với a_i ($1 \le i \le n$) là các dữ liệu nguyên tử. Trong danh sách tuyến tính luôn tồn tại một phần tử đầu a_1, phần tử cuối a_n. Đối với mỗi phần tử a_i bất kỳ với $1 \le i \le n - 1$ thì có một phần tử a_{i+1} gọi là *phần tử sau* a_i, và với $2 \le i \le n$ thì có một phần tử a_{i-1} gọi là *phần tử trước* a_i. a_i được gọi là phần tử thứ i của danh sách tuyến tính, n được gọi là độ dài hoặc kích thước của danh sách.

Mỗi phần tử trong danh sách thường là một bản ghi (gồm một hoặc nhiều trường (fields)) đó là phần thông tin nhỏ nhất có thể tham khảo. VD: Danh sách sinh viên trong một lớp là một danh sách tuyến tính mà mỗi phần tử ứng với một sinh viên, nó bao gồm các trường:

Mã SV (STT), Họ và tên, Ngày sinh, Quê quán, . . .

Các phép toán thao tác trên danh sách :

+ Phép bổ sung một phần tử vào trong danh sách (Insert) .

+ Phép loại bỏ một phần tử trong danh sách (Delete).

+ Phép ghép nối 2 hoặc nhiều danh sách.

+ Phép tách một danh sách thành nhiều danh sách.

+ Phép sao chép một danh sách.

+ Phép cập nhật (update) danh sách.

+ Phép sắp xếp các phần tử trong danh sách theo thứ tự ấn định.

+ Phép tìm kiếm một phần tử trong danh sách theo giá trị ấn định của một trường nào đó.

Trong đó phép bổ sung và phép loại bỏ là hai phép toán thường xuyên được sử dụng trong danh sách.

Tệp cũng là một trường hợp của danh sách nó có kích thước lớn và thường được lưu trữ ở bộ nhớ ngoài. Còn danh sách nói chung thường được xử lý ở bộ nhớ trong.

Bộ nhớ trong được hình dung như một dãy các từ máy(words) có thứ tự, mỗi từ máy ứng với một địa chỉ. Mỗi từ máy chứa từ 8 ? 64 bits, việc tham khảo đến nội dung của nó thông qua địa chỉ.

+ *Cách xác định địa chỉ của một phần tử trong danh sách*: Có 2 cách xác định địa chỉ:

Cách 1: Dựa vào đặc tả của dữ liệu cần tìm . Địa chỉ này gọi là địa chỉ tính được (hay địa chỉ trực tiếp).

VD: Xác định địa chỉ của các phần tử trong véc tơ, ma trận thông qua các chỉ số.

Cách 2: Lưu trữ các địa chỉ cần thiết ở trong bộ nhớ, khi cần xác định sẽ lấy ở đó ra. Địa chỉ này được gọi là con trỏ (pointer) hay mối nối (link).

Lưu trữ kế tiếp của danh sách tuyến tính

Lưu trữ kế tiếp là phương pháp lưu trữ sử dụng mảng một chiều làm cấu trúc lưu trữ của danh sách tuyến tính nghĩa là có thể dùng một véc tơ lưu trữ V_i với $1 \leq i \leq n$ để lưu trữ một danh sách tuyến tính $(a_1, a_2, \ldots, a_n)$ trong đó phần tử a_i được chứa ở V_i.

Ưu điểm : Tốc độ truy nhập nhanh, dễ thao tác trong việc bổ sung, loại bỏ và tìm kiếm phần tử trong danh sách.

Nhược điểm: Do số phần tử trong danh sách tuyến tính thường biến động (kích thước n thay đổi) dẫn đến hiện tượng lãng phí bộ nhớ. Mặt khác nếu dự trữ đủ rồi thị việc bổ

sung hay loại bỏ một phần tử trong danh sách mà không phải là phần tử cuối sẽ đòi hỏi phải dồn hoặc dãn danh sách (nghĩa là phải dịch chuyển một số phần tử để lấy chỗ bổ sung hay tiến lên để lấp chỗ phần tử bị loại bỏ) sẽ tốn nhiều thời gian

Nhu cầu xây dựng cấu trúc dữ liệu động

Với các cấu trúc dữ liệu được xây dựng từ các kiểu cơ sở như: kiểu thực, kiểu nguyên, kiểu ký tự ... hoặc từ các cấu trúc đơn giản như mẩu tin, tập hợp, mảng ... lập trình viên có thể giải quyết hầu hết các bài toán đặt ra. Các đối tượng dữ liệu được xác định thuộc những kiểu dữ liệu này có đặc điểm chung là không thay đổi được kích thước, cấu trúc trong quá trình sống, do vậy thường cứng ngắt, gò bó khiến đôi khi khó diễn tả được thực tế vốn sinh động, phong phú. Các kiểu dữ liệu kể trên được gọi là các kiểu dữ liệu tĩnh.

Ví dụ :

1. Trong thực tế, một số đối tượng có thể được định nghĩa đệ qui, ví dụ để mô tả đối tượng "con người" cần thể hiện các thông tin tối thiểu như :

? Họ tên

? Số CMND

? Thông tin về cha, mẹ

Để biểu diễn một đối tượng có nhiều thành phần thông tin như trên có thể sử dụng kiểu bản ghi. Tuy nhiên, cần lưu ý cha, mẹ của một người cũng là các đối tượng kiểu NGƯỜI, do vậy về nguyên tắc cần phải có định nghĩa như sau:

```
typedef struct NGUOI{

char Hoten[30];

int So_CMND ;

NGUOI Cha,Me;

};
```

Nhưng với khai báo trên, các ngôn ngữ lập trình gặp khó khăn trong việc cài đặt không vượt qua được như xác định kích thước của đối tượng kiểu NGUOI.

2. Một số đối tượng dữ liệu trong chu kỳ sống của nó có thể thay đổi về cấu trúc, độ lớn, như danh sách các học viên trong một lớp học có thể tăng thêm, giảm đi ... Khi

đó nếu cố tình dùng những cấu trúc dữ liệu tĩnh đã biết như mảng để biểu diễn những đối tượng đó lập trình viên phải sử dụng những thao tác phức tạp, kém tự nhiên khiến chương trình trở nên khó đọc, do đó khó bảo trì và nhất là khó có thể sử dụng bộ nhớ một cách có hiệu quả.

3. Một lý do nữa làm cho các kiểu dữ liệu tĩnh không thể đáp ứng được nhu cầu của thực tế là tổng kích thước vùng nhớ dành cho tất cả các biến tĩnh có giới hạn. Khi có nhu cầu dùng nhiều bộ nhớ hơn ta phải sử dụng các *cấu trúc dữ liệu động.*

4. Cuối cùng, do bản chất của các dữ liệu tĩnh, chúng sẽ chiếm vùng nhớ đã dành cho chúng suốt quá trình hoạt động của chương trình. Tuy nhiên, trong thực tế, có thể xảy ra trường hợp một dữ liệu nào đó chỉ tồn tại nhất thời hay không thường xuyên trong quá trình hoạt động của chương trình. Vì vậy việc dùng các CTDL tĩnh sẽ không cho phép sử dụng hiệu quả bộ nhớ.

Do vậy, nhằm đáp ứng nhu cầu thể hiện sát thực bản chất của dữ liệu cũng như xây dựng các thao tác hiệu quả trên dữ liệu, cần phải tìm cách tổ chức kết hợp dữ liệu với những hình thức mới linh động hơn, có thể thay đổi kích thước, cấu trúc trong suốt thời gian sống. Các hình thức tổ chức dữ liệu như vậy được gọi là *cấu trúc dữ liệu động*. Bài sau sẽ giới thiệu về các cấu trúc dữ liệu động và tập trung khảo sát cấu trúc đơn giản nhất thuộc loại này là danh sách liên kết.