Cây nhị phân và ứng dụng~

Bởi:

Khoa CNTT ĐHSP KT Hưng Yên

CÂY BIỂU THỨC

Cách dựng cây biểu thức

TTTH2TTTH1TH2 Đối với phép toán hai ngôi (chẳng hạn +, -, *, /) được biểu diễn bởi cây nhị phân mà gốc của nó chứa toán tử, cây con trái biểu diễn toán hạng bên trái, còn cây con bên phải biểu diễn toán hạng bên phải

(1)

Hình 5.11

a+ba + b!nn!xexpexp(x) Đối với phép toán một ngôi như "phủ định" hoặc "lấy giá trị đối", hoặc các hàm chuẩn như exp() hoặc cos()...thì cây con bên trái rỗng. Còn với các phép toán một toán hạng như phép "lấy đạo hàm" ()' hoặc hàm "giai thừa" ()! thì cây con bên phải rỗng.

<>=orabcd(a < b) or (c >= d)/a+bca/(b + c)

Hình 15.1.Một số cây biểu thức

Nhận xét

Nếu ta đuyệt cây biểu thức theo thứ tự trước thì ta được biểu thức Balan dạng tiền tố (prefix). Nếu duyệt cây nhị phân theo thứ tự sau thì ta có biểu thức Balan dạng hậu tố (postfix); còn theo thứ giữa thì ta nhận được cách viết thông thường của biểu thức (dạng trung tố).

Ví dụ

Để minh cho nhận xét này ta lấy ví dụ sau:

Cho biểu thức P = a*(b - c) + d/e

a) Hãy dựng cây biểu thức biểu diễn biểu thức trên

TH1+TH2b) Đưa ra biểu thức ở dạng tiền tố và hậu tố

Giải

a) Ta có TH1 = a*(b - c) suy ra câybiểu thức có dạng

TH2 = d/e

Xét TH1 = a*(b - c), toán hạng chưa ở dạng cơ bản ta phải phân tích để được như ở

TH3*TH4(1)

TH3 = a cây biểu thức của toán hạng này là

TH4 = b - c

b-c

Toán hạng TH4 đã ở dạng cơ bản

nên ta có ngay cây biểu thức

d/e

Cũng tương tự như vậy đối với

toán hạng TH2, cây biểu thức

tương ứng với toán hạng này là

Hình 5.13

Tổng hợp cây biểu thức của các toán hạng ta được cây biểu thức sau

*/+a-debc

Hình 15.2. Cây biểu thức

b) Bây giờ ta duyệt cây biểu thức ở hình 5.14

Duyệt theo thứ tự trước : + * a - b c / d e

Duyệt theo thứ sau: a b c - * d e / +

CÂY NHỊ PHÂN TÌM KIẾM

Cây nhị phân được sử dụng vào nhiều mục đích khác nhau. Tuy nhiên việc sử dụng cây nhị phân để lưu giữ và tìm kiếm thông tin vẫn là một trong những áp dụng quan trọng nhất của cây nhị phân. Trong phần này chúng ta sẽ nghiên cứu một lớp cây nhị phân đặcbiệt phục vụ cho việc tìm kiếm thông tin, đó là cây nhị phân tìm kiếm.

Trong thực tế, một lớp đối tượng nào đó có thể được mô tả bởi một kiểu bản ghi, các trường của bản ghi biểu diễn các thuộc tính của đối tượng. Trong bài toán tìm kiếm thông tin, ta thường quan tâm đến một nhóm các thuộc tính nào đó của đối tượng mà thuộc tính này hoàn toàn xác định được đối tượng. Chúng ta gọi các thuộc tính này là khoá. Như vậy, khoá là một nhóm các thuộc tính của một lớp đối tượng sao cho hai đối tượng khác nhau cần phải có giá trị khác nhau trên nhóm thuộc tính đó.

Định nghĩa

Cây nhị phân tìm kiếm (CNPTK) là cây nhị phân hoặc rỗng hoặc thoả mãn đồng thời các điều kiện sau:

- Khoá của các đỉnh thuộc cây con trái nhỏ hơn khoá nút gốc
- Khoá của nút gốc nhỏ hơn khoá của các đỉnh thuộc cây con phải của của gốc
- Cây con trái và cây con phải của gốc cũng là cây nhị phân tìm kiếm

Hình 5.19 biểu diễn một cây nhị phân tìm kiếm, trong đó khoá của các đỉnh là các số nguyên.

7 24 15 2 10 20 34 55 9 12

Cài đặt cây nhị phân tìm kiếm

Mỗi nút trên cây nhị phân tìm kiếm có dạng

left	info	right

Trong đó trường Left :con trỏ chỉ tới cây con trái

Right :con trỏ chỉ tới cây con phải

Info : chứa thông tin của nút

Type

```
keytype =...; {kiểu dữ liệu của mỗi nút}

Tree = ^ node;

Node = record

Info : keytype;

Left, Right : Tree;

end;

Var T : Tree;
```

Các thao tác cơ bản trên cây nhị phân tìm kiếm

1. Tìm kiếm

Tìm kiếm trên cây là một trong các phép toán quan trọng nhất đối với cây nhị phân tìm kiếm . Ta xét bài toán sau

Bài toán: Giả sử mỗi đỉnh trên cây nhị phân tìm kiếm là một bản ghi, biến con trỏ Root chỉ tới gốc của cây và x là khoá cho trước. Vấn đề đặt ra là, tìm xem trên cây có chứa đỉnh với khoá x hay không.

Giải thuật đệ qui

```
Procedure search (Root : Tree; x : keytype; var p : tree);

{ Nếu tìm thấy thì p chỉ tới nút có trường khoá bằng x, ngược lại p=nil}

Begin

p:=root;

if p <> nil then

if x < p^.info then search (p^.left, x, p)

else

if x > p^.info then search (p^.Right, x, p)

End;
```

Giải thuật lặp

Trong thủ tục này, ta sẽ sử dụng biến địa phương found có kiểu boolean để điều khiển vòng lặp, nó có giá trị ban đầu là false. Nếu tìm kiếm thành công thì found nhận giá trị true, vòng lặp kết thúc và đồng thời p trỏ đến nút có trường khoá bằng x. Còn nếu không tìm thấy thì giá trị của found vẫn là false và giá trị của p lànil

```
Procedure search (Root : Tree; x:keytype; var p : Tree);

Var Found : boolean;

Begin

Found:=False;

P:=Root;

while (p<>nil) and( not Found ) do

if x > p^.info then p := p^.right

else

if x < p^.info then p := p^.left

else Found = True;

End;
```

1. Đi qua CNPTK

Như ta đã biết CNPTK cũng là cây nhị phân nên các phép duyệt trên cây nhị phân cũng vẫn đúng trên CNPTK. Chỉ có lưu ý nhỏ là khi duyệt theo thứ tự giữa thì được dãy khoá theo thứ tự tăng dần.

c) Chèn một nút vào CNPTK

Việc thêm một một nút có trường khoá bằng x vào cây phải đảm bảo điều kiện ràng buộc của CNPTK. Ta có thể thêm vào nhiều chỗ khác nhau trên cây, nhưng nếu thêm vào một nút lá sẽ là tiện lợi nhất do ta có thể thực hiện quá trình tương tự như thao tác tìm kiếm. Khi kết thúc việc tìm kiếm cũng chính là lúc tìm được chỗ cần chèn.

Giải thuật đệ quy

```
Procedure Insert (var Root :Tree; x: kkeytype);

Var p : tree;

Begin

New(p);{Tạo ra nút mới}

P^.info := x;

if root=nil then

begin

Root :=p;

P^.left:= nil;

P^.right:= nil;

End

Else

with Root^ do

if x> info then insert (Right, x)

else if x < info then insert (left, x);

End;
```

Giải thuật lặp

Trong thủ tục này ta sử dụng biến con trỏ địa phương q chạy trên các đỉnh của cây bắt đầu từ gốc. Khi đang ở một đỉnh nào đó, q sẽ xuống đỉnh con trái (phải) tuỳ theo khoá ở đỉnh lớn hơn (nhỏ hơn) khoá x.

Ở tại một đỉnh nào đó khi p muốn xuống đỉnh con trái (phải) thì phải kiểm tra xem đỉnh này có đỉnh con trái (phải) không. Nếu có thì tiếp tục xuống, ngược lại thì treo đỉnh mới vào bên trái (phải) đỉnh đó. Điều kiện q = nil sẽ kết thúc vòng lặp. Quá trình này được lại lặp khi có đỉnh mới được chèn vào.

```
procedure Insert (var Root : Tree; x: keytype)
```

```
var p, q : tree;
begin
New(p);
P^.info :=x;
if Root = nil then
Begin
Root:=p;
P^.left:= nil;
P^.right:= nil;
End
Else
Begin
q:=Root;
while q <> nil do
if x < q^.info then
if q^.left <> nil then q := q^.left
else begin
q^.left :=p;
p := nil;
end
else if x > q^.info then
if q^.right <> nil then q:=q^.right
```

```
else begin

q^.right :=p;

q =nil;

end;

end;

end;
```

Nhận xét: Để dựng được CNPTK ứng với một dãy khoá đưa vào bằng cách liên tục bổ các nút ứng với từng khoá, bắt đầu từ cây rỗng. Ban đầu phải dựng lên cây với nút gốc là khoá đầu tiên sau đó đối với các khoá tiếp theo, tìm trên cây không có thì bổ sung vào.

Ví dụ với dãy khoá: 42 23 74 11 65 58 94 36 99 87

thì cây nhị phân tìm kiếm dựng được sẽ có dạng ở hình 5.20

23744211366594589987

Hình 5.20. Một cây nhị phân tìm kiếm

d)Loại bỏ nút trên cây nhị phân tìm kiếm

Đối lập với phép toán chèn vào là phép toán loại bỏ. Chúng ta cần phải loại bỏ khỏi CNPTK một đỉnh có khoá x (ta gọi tắt là nút x) cho trước, sao cho việc huỷ một nút ra khỏi cây cũng phải bảo đảm điều kiện ràng buộc của CNPTK.

Có ba trường hợp khi huỷ một nút x có thể xảy ra:

- X là nút lá
- X là nút nửa lá (chỉ có một con trái hoặc con phải)
- X có đủ hai con (trường hợp tổng quát)

Trường hợp thứ nhất: chỉ đơn giản huỷ nút x vì nó không liên quan đến phần tử nào khác.

320TCây trước khi xoá20TCây sau khi xoá

Hình 5.21

Trường hợp thứ hai: Trước khi xoá nút x cần móc nối cha của x với nút con (nút con trái hoặc nút con phải) của nó

$T_2$2010318T_1

$T_2$201025318T_1

a) Cây trước khi xoáb) Cây sau khi xoá đỉnh (25)

Trường hợp tổng quát: khi nút bị loại bỏ có cả cây con trái và cây con phải, thì nút thay thế nó hoặc là nút ứng với khoá nhỏ hơn ngay sát trước nó (nút cực phải của cây con trái nó) hoặc nút ứng với khoá lớn hơn ngay sát sau nó (nút cực trái của cây con phải nó). Như vậy ta sẽ phải thay đổi một số mối nối ở các nút:

- Nút cha của nút bị loại bỏ
- Nút được chọn làm nút thay thế
- Nút cha của nút được chọn làm nút thay thế

T_2b) Cây sau khi xoá đỉnh 201810253T_1T_2a) Cây trước khi xoá đỉnh 20201025318T_1

Trong ví dụ này ta chọn nút thay thế nút bị xoá là nút cực phải của cây con trái (nút 18).

T_5ABT_4C$ET_2T_3T_1$RQTSa) Cây trước khi xoá nút trỏ bởi QT_5AEB$T_4C T_2T_1$RQTST_3b) Cây sau khi xoá nút trỏ bởi Q Sau đây là giải thuật thực hiện việc loại bỏ một nút trỏ bởi Q. Ban đầu Q chính là nối trái hoặc nối phải của một nút R trên cây nhị phân tìm kiếm, mà ta giả sử đã biết rồi.

```
procedure Del (var Q: Tree); {xoá nút trỏ bởi Q}

var T, S : Tree;

begin

P := Q; {Xử lý trường hợp nút lá và nút nửa lá}

if P^.left = nil then

Begin

Q:=P^.right ;{R^.left := P^.right}

Dispose(P);
```

```
end
else
if P^.right = nil then
begin
Q :=P^.left;
Dispore (P);
end
else { Xử lý trường hợp tổng quát}
begin
T := P^.left;
if T^.right = nil then
begin
T^.right := P^.right;
Q := T;
Dispose (P);
end
else
begin
S := T^.right; {Tìm nút thay thế, là nút cực phải của cây }
while S^.right <> nil do
begin
T := S;
```

```
S := T^.right;

end;

S^.right := P^.right;

T^.right := S^.left;

S^.left := P^.left;

Q:=S;

Dispose(p);

end;

end;

end;
```

Thủ tục xoá trường dữ liệu bằng X

- Tìm đến nút có trường dữ liệu bằng X
- Áp dụng thủ tục Del để xoá

Sau đây chúng ta sẽ viết thủ tục loại khỏi cây gốc Root đỉnh có khoá x cho trước. Đó là thủ tục đệ qui, nó tìm ra đỉnh có khoá x, sau đó áp dụng thủ tục Del để loại đỉnh đó ra khỏi cây.

```
procedure Delete (var Root :Tree ; x : keytype);

begin

if Root <> nil then

if x < Root^.info then Delete (Root^.left, x)

else if x > Root^.info then Delete (Root^.right, x)

else Del(Root);

end;
```

Nhận xét: Việc huỷ toàn bộ cây có thể thực hiện thông qua thao tác duyệt cây theo thứ sau. Nghĩa là ta sẽ huỷ cây con trái, cây con phải rồi mới huỷ nút gốc.

```
procedure RemoveTree (var Root: Tree);

begin

if Root <> nil then

begin

RemoveTree(Root^.left);

RemoveTree(Root^.right);

Dispose (Root);

end;

end;
```