***Cấu trúc dữ liệu và giải thuật***

***Mục lục***

Phần 1: Thuật toán

Chương: Thuật toán

Bài 1: Thuật toán và các vấn đề liên quan

Thuật toán được hiểu là sự đặc tả chính xác một dãy các bước có

thể thực hiện được một cách máy móc để giải quyết một vấn đề. Cần nhấn

mạnh rằng, mỗi thuật toán có một dữ liệu vào (Input) và một dữ liệu ra

(Output); khi thực hiện thuật toán (thực hiện các bước đã mô tả), thuật toán

cần cho ra các dữ liệu ra tương ứng với các dữ liệu vào.

Tính hiệu quả của thuật toán:

* Thuật toán đơn giản, dễ hiểu
* Thuật toán dễ cài đặt
* Thuật toán cần ít bộ nhớ
* Thuật toán chạy nhanh

Thời gian chạy mà ta thu được sẽ phụ thuộc vào nhiều nhân tố:

• Kỹ năng của người lập trình • Chương trình dịch

• Tốc độ thực hiện các phép toán của máy tính • Dữ liệu vào Thời gian chạy trong trường hợp xấu nhất (worst-case running

time) của một thuật toán là thời gian chạy lớn nhất của thuật toán đó trên tất

cả các dữ liệu vào cùng cỡ . Chúng ta sẽ ký hiệu thời gian chạy trong trường

hợp xấu nhất là T(n), trong đó n là cỡ của dữ liệu vào. Sau này khi nói tới

thời gian chạy của thuật toán chúng ta cần hiểu đó là thời gian chạy trong

trường hợp xấu nhất. Sử dụng thời gian chạy trong trường hợp xấu nhất để

biểu thị thời gian chạy của thuật toán có nhiều ưu điểm. Trước hết, nó đảm

bảo rằng, thuật toán không khi nào tiêu tốn nhiều thời gian hơn thời gian

chạy đó. Hơn nữa, trong các áp dụng, trường hợp xấu nhất cũng thường

xuyên xảy ra.

Chúng ta xác định thời gian chạy trung bình (average running time)

của thuật toán là số trung bình cộng của thời gian chạy của thuật toán đó trên

tất cả các dữ liệu vào cùng cỡ n. Thời gian chạy trung bình của thuật toán sẽ

được ký hiệu là Ttb(n). Đánh giá thời gian chạy trung bình của thuật toán là

công việc rất khó khăn, cần phải sử dụng các công cụ của xác suất, thống kê

và cần phải biết được phân phối xác suất của các dữ liệu vào. Rất khó biết

được phân phối xác suất của các dữ liệu vào. Các phân tích thường phải dựa

trên giả thiết các dữ liệu vào có phân phối xác suất đều. Do đó, sau này ít khi

ta đánh giá thời gian chạy trung bình.

Bài 2: Ký hiệu Ô lớn và biểu diễn thời gian chạy bỏi ký hiệu ô lớn

1. Định nghĩa về ký hiệu Ô lớn

Định nghĩa: Giả sử f(n) và g(n) là các hàm thực không âm của đối số nguyên khong âm n . Ta nói “f(n) là ô lớn của g(n)” và được viết là:

f(n) = O(g(n))

Nếu tồn tại các hằng số dương c và n0 sao cho f(n) <= cg(n) với mọi n >= n0  Như vậy, f(n)= O(g(n)) có nghĩa là hàm f(n) bị chặn trên bởi hàm g(n) với một nhân tử hằng nào đó khi n đủ lớn. Muốn chứng minh được f(n) = O(g(n)), chúng ta cần chỉ ra nhân tử hằng c , số nguyên dương n0 và chứng minh được f(n) <= cg(n) với mọi n >= no .

Ví dụ: Giả sử f(n) = 5n3

+ 2n2 + 13n + 6 , ta có:

f(n) = 5n3

+ 2n2 + 13n + 6 <= 5n3

+ 2n3 + 13n3 + 6n3 = 26n3

Bất đẳng thức trên đúng với mọi n >= 1, và ta có n0 = 1, c = 26. Do đó, ta có

thể nói f(n) = O(n3).

Tổng quát nếu f(n) là 1 đa thức bậc k của n:

f(n) = an nk + ak-1n k-1 + ... + a1n + a0 thì f(n) = O(nk )

Sau đây chúng ta đưa ra 1 số hệ quả từ định nghĩa trên:

* Nếu f(n) = g(n)+g1n +...+ gk(n), , trong đó các hàm gi(n) (i=1,...,k) tăng chậm hơn hàm g(n) (tức là gi(n)/g(n) --> 0, khi n-->0) thì f(n) = O(g(n))

• Nếu f(n) = O(g(n)) thì f(n) = O(d.g(n)), trong đó d là hằng số

dương bất kỳ

• Nếu f(n) = O(g(n)) và g(n) = O(h(n)) thì f(n) = O(h(n)) (tính bắc

cầu)

Bài 4: Phân tích các hàm đệ quy

Các hàm đệ quy là các hàm có chứa lời gọi hàm đến chính nó. Trong

mục này, chúng ta sẽ trình bầy phương pháp chung để phân tích các hàm đệ

quy, sau đó sẽ đưa ra một số kỹ thuật phân tích một số lớp hàm đệ quy hay

gặp.

Giả sử ta có hàm đệ quy F, thời gian chạy của hàm này là T(n), với n

là cỡ dữ liệu vào. Khi đó thời gian chạy của các lời gọi hàm ở trong hàm F

sẽ là T(m) với m < n. Trước hết ta cần đánh giá thời gian chạy của hàm F

trên dữ liệu cỡ nhỏ nhất n = 1, giả sử T(1) = a với a là một hằng số nào đó.

Sau đó bằng cách đánh giá thời gian chạy của các câu lệnh trong thân của

hàm F, chúng ta sẽ tìm ra quan hệ đệ quy biểu diễn thời gian chạy của hàm F

thông qua lời gọi hàm, tức là biểu diễn T(n) thông qua các T(m), với m < n.

Chẳng hạn, giả sử hàm đệ quy F chứa hai lời gọi hàm với thời gian chạy

tương ứng là T(m1) và T(m2), trong đó m1, m2 <n, khi đó ta thu được quan

hệ đệ quy có dạng như sau:

T(1) = 1

T(n) = f(T(m1),T(m2))

Trong đó, f là một biểu thức nào đó của T(m1) và T(m2). Giải quan hệ đệ

quy trên, chúng ta sẽ đánh giá được thời gian chạy T(n). Nhưng cần lưu ý

rằng, giải các quan hệ đệ quy là rất khó khăn, chúng ta sẽ đưa ra kỹ thuật

giải cho một số trường hợp đặc biệt.

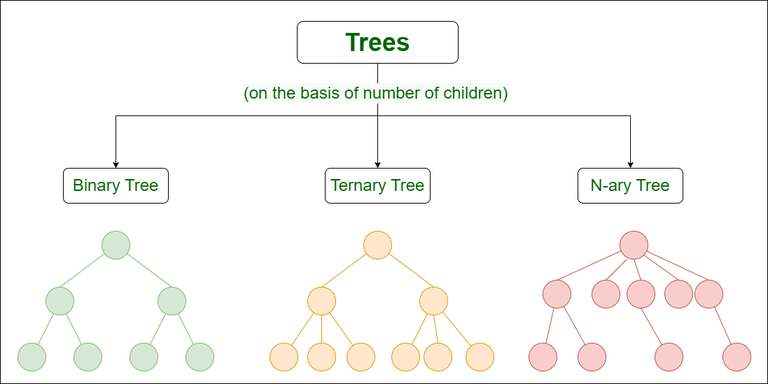
Chương 2: Sắp xếp

Sắp xếp đóng vai trò quan trọng trong tìm kiếm dữ liệu. Chẳng hạn, nếu danh sách đã được sắp xếp theo thứ tự tăng dần (hoặc giảm dần), ta có thể sử dụng kỹ thuật tìm kiếm nhị phân hiệu quả hơn nhiều tìm kiếm tuần tự… Trong thiết kế thuật toán, ta cũng thường xuyên cần đến sắp xếp, nhiều thuật toán được thiết kế dựa trên ý tưởng xử lý các đối tượng theo một thứ tự xác định

Có hai loại thuật toán:

1. Sắp xếp trong (sắp xếp mảng)
2. Sắp xếp ngoài (độ lưu trữ lớn)
3. Các thuật toán sắp xếp đơn giản
4. Sắp xếp lựa chọn

Chương 4: Cây



1. Các khái niệm cơ bản

Chúng ta có thể xác định khái niệm cây bằng hai cách: đệ quy và không đệ quy.

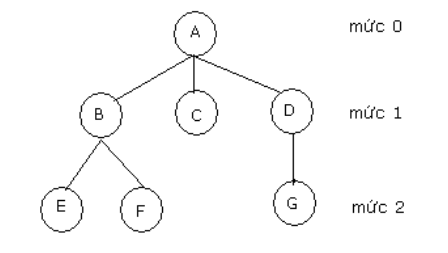
Trước hết chúng ta đưa ra định nghĩa cây thông qua các khái niệm trong đồ thị định hướng. Một ví dụ điển hình về cây là tập hợp các thành viên trong một dòng họ với quan hệ cha – con. Trừ ông tổ của dòng họ này, mỗi một người trong dòng họ là con của một người cha nào đó trong dòng họ. Biểu diễn dòng họ dưới dạng đồ thị hướng: quan hệ cha – con được biểu diễn bởi các cung của đồ thị, nếu A là cha của B, thì trong đồ thị có cung đi từ đỉnh A tới đỉnh B. Xem xét các đặc điểm của đồ thị định hướng này, chúng ta đưa ra định nghĩa cây như sau:

Cây là một đồ thị định hướng thỏa mãn các tính chất sau:

• Có một đỉnh đặc biệt được gọi là gốc cây

• Mỗi đỉnh C bất kỳ không phải là gốc, tồn tại duy nhất một đỉnh P có cung đi từ P đến C. Đỉnh P được gọi là cha của đỉnh C, và C là con của P

• Có đường đi duy nhất từ gốc tới mỗi đỉnh của cây.



Hình 4.1 Biểu diễn hình học của cây

Sau đây chúng ta sẽ đưa ra một số thuật ngữ hay được dùng đến sau

này.

Mở rộng của quan hệ cha – con, là quan hệ tổ tiên – con cháu. Trong cây nếu có đường đi từ đỉnh A tới đỉnh B thì A được gọi là tổ tiên của B, hay B là con cháu của A. Chẳng hạn, gốc cây là tổ tiên của các đỉnh còn lại trong cây.

Các đỉnh cùng cha được xem là anh em. Chẳng hạn, trong cây ở hình

4.1 các đỉnh B, C, D là anh em.

Các đỉnh không có con được gọi là lá. Trong hình 4.1, các đỉnh lá là

E, F, C, G. Một đỉnh không phải là lá được gọi là đỉnh trong.

Một đỉnh bất kỳ A cùng với tất cả các con cháu của nó lập thành một cây gốc là A. Cây này được gọi là cây con của cây đã cho. Nếu đỉnh A là con của gốc, thì cây con gốc A được gọi là cây con của gốc.

Độ cao của cây là số đỉnh nằm trên đường đi dài nhất từ gốc tới một

lá. Chẳng hạn, cây trong hình 4.1 có độ cao là 3. Dễ dàng thấy rằng, độ cao

của cây là độ cao lớn nhất của cây con của gốc cộng thêm 1.

Độ sâu của một đỉnh là độ dài đường đi từ gốc tới đỉnh đó. Chẳng

hạn, trong hình 4.1, đỉnh G có độ sâu là 2.

Cây là CTDL phân cấp : Các đỉnh của cây được phân thành các

mức. Mức của mỗi đỉnh được xác định đệ quy như sau:

• Gốc ở mức 1

• Mức của một đỉnh = mức của đỉnh cha + 1

Như vậy, các đỉnh trong cùng một mức là đỉnh con của một đỉnh nào

đó ở mức trên. Độ cao của cây chính là mức lớn nhất của cây. Ví dụ, cây

trong hình 8.1 được phân thành 3 mức: mức 1 chỉ gồm có gốc, mức 2 gồm

các đỉnh A, B, C, D, mức 3 gồm các đỉnh E, F, G.

Sau này chúng ta chỉ quan tâm đến các cây được sắp. Cây được sắp

là cây mà các đỉnh con của mối đỉnh được sắp sếp theo một thứ thứ tự xác

định. Giả sử a là một đỉnh và các con của nó được sắp xếp theo thứ tự b1, b2,

..., bk (k ≥ 1), khi đó đỉnh b1 được gọi là con cả của a, còn đỉnh bi+1 (i=1, 2,

...k-1) được gọi là em liền kề của đỉnh bi.

Trong biểu diễn hình học, đỉnh con cả là đỉnh ngoài cùng bên trái,

đỉnh bk là đỉnh ngoài cùng bên phải.

Trên đây chúng ta đã định nghĩa cây như một đồ thị định hướng có

một số tính chất đặc biệt. Khái niệm cây còn có thể định nghĩa một cách

khác: định nghĩa đệ quy.

Định nghĩa đệ quy. Cây là một tập hợp không rỗng T các phần tử

(được gọi là các đỉnh) được xác định đệ quy như sau:

• Tập chỉ có một đỉnh a là cây, cây này có gốc là a.

• Giả sử T1, T2, ....., Tk (k≥ 1) là các cây có gốc tương ứng là r1,

r2, ...., rk , trong đó hai cây bất kỳ không có đỉnh chung. Giả sử r

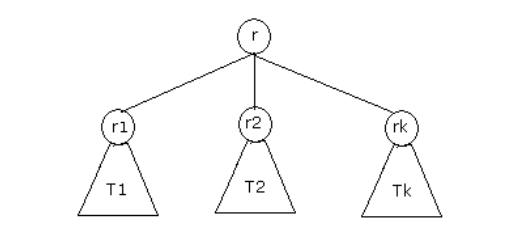
là một đỉnh mới không có trong các cây đó. Khi đó tập T gồm

đỉnh r và tất cả các đỉnh trong các cây Ti (i=1, ...., k) lập thành

một cây có gốc là đỉnh r, và r là đỉnh cha của đỉnh ri hay ri là

đỉnh con của r (i=1, ..., k). Các cây Ti (i=1, ..., k) được gọi là

các cây con của gốc r. Cây T được biểu diễn hình học như sau:



***Cài đặt cây***

Cây có thể cài đặt bởi các CTDL khác nhau. Chúng ta có thể sử dụng mảng để cài đặt cây. Song cách này không thuận tiện, ít được sử dụng. Sau đây, chúng ta trình bày hai phương pháp cài đặt cây thông dụng nhất.

***Phương pháp 1:*** (Chỉ ra dsach các đỉnh con của mỗi đỉnh). Với mỗi đỉnh của cây, ta sử dụng một con trỏ trỏ tới một đỉnh con của nó. Và như vậy, mỗi đỉnh cảu cây được biểu diễn bởi cấu trúc gồm hai thành phần: một biến Data lưu dữ liệu chứa trong đỉnh đó và một mảng Child các con trỏ trỏ tới các đỉnh con. Giả sử, mỗi đỉnh chỉ có nhiều nhất K đỉnh con, khi đó ta có thể mô tả mỗi đỉnh bởi cấu trúc sau:

Const int K = 10;

Template<class Item>

{

Item data;

Node\*child[k];

};

Chúng ta có thể truy cập tới một đỉnh bất kỳ trong cây bằng cách đi

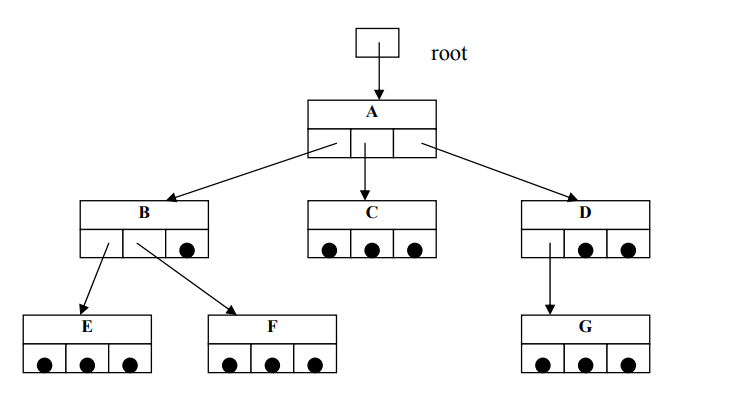
theo các con trỏ bắt đầu từ gốc cây. Vì vậy, ta cần có một con trỏ ngoài trỏ

tới gốc cây, con trỏ root:

Node <Item>\* root;

với cách cài đặt này, cây trong hình 8.1 được cài đặt bởi CTDT được biểu

diễn hình học trong hình 4.2.



***Hình 4.2 Cài đặt con trỏ bằng mảng***

2.

3. Cây nhị phân

Trong mục này, chúng ta xét một lớp cây đặc biệt: Cây nhị phân

(Binary tree). Cây nhị phân có thể rỗng (không có đỉnh nào), nếu không

rỗng thì mỗi đỉnh của cây nhị phân chỉ có nhiều nhất hai con được phân biệt

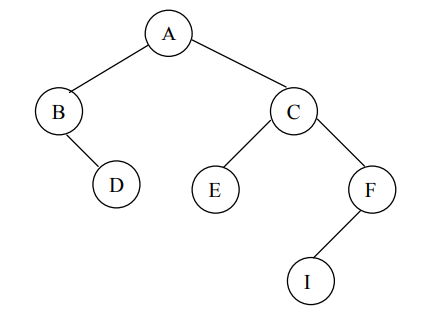
là con trái và con phải. Điều đó có nghĩa rằng, trong cây nhị phân không

rỗng, một đỉnh có thể không có con, có thể có đầy đủ cả con trái và con phải,

có thể có con trái không có con phải hoặc có con phải nhưng không có con

trái. Hình 4.5 biểu diễn một cây nhị phân: các đỉnh A, C có hai con, đỉnh B

có con phải, đỉnh F có con trái.



Chúng ta có thể định nghĩa cây nhị phân một cách đệ quy như sau:

• Một tập rỗng là cây nhị phân. Ta gọi là cây nhị phân rỗng

• Giả sử TL và TR là hai cây nhị phân không có đỉnh chung và r là

một đỉnh không có trong các cây TL và TR. Khi đó một cây nhị

phân T được tạo thành với gốc là r, có TL là cây con trái của

gốc và TR là cây con phải của gốc. Cây nhị phân T được biểu

diễn hình học như sau:

1. Cây nhị phân tìm kiếm

Cây tìm kiếm nhị phân (binary search tree) là cây nhị phân thỏa mãn

tính chất sau: đối với mỗi đỉnh x trong cây, nếu y là đỉnh bất kỳ ở cây con

trái của x thì khóa của x lớn hơn khóa của y, còn nếu y là đỉnh bất kỳ ở cây

con phải của x thì khóa của x nhỏ hơn khóa của y.

VD: Xây dựng cây nptk từ dãy khóa

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 68 | 89 | 67 | 65 | 66 | 82 | 84 | 91 | 83 | 64 |

***Duyệt trước :*** 68; 67; 65;

66; 89;82;84;83;91.

***Duyệt giữa:*** 64;65;66;67;68;82;83;84;89;91

***Duyệt sau:*** 64 66 65 67 83 84 82 91 89 68

Duyệt cây là việc đi thăm tất cả các nút, mỗi nút thăm đúng 1 lần

***Duyệt cây théo thứ tự trước (Duyệt trước - preorder)***

Gốc - Cây con trái – cây con phải

***Duyệt giữa (- in \_order)***

Cây con trái – Gốc – Cây con phải

***Duyệt sau (pos\_order)***

Cây con trái – cây con phải – Gốc

Khi duyệt trến con trái, cây con phải ta lại áp dựng đệ quy thứ tự trên

VD: Duyệt cây vừa mới xây dựng trên

Code:

68

89

67

NULL NULL NULL NULL

Struct Node

{

Int data;

Node\*left;

Node\*right;

};

Node\*root;

//các hàm duyệt cây

Void duyettruoc (Node\*root)

{

If(root!=NULL)

{

Cout<<root->data<<””;

Duyettruoc (root-> left);

Duyettruoc(root->right);

}

}

Void duyetgiua(Node\*root)

{

If(root!=NULL)

{

Duyetgiua(root->left);

Cout<<root->data<<””;

Duyetgiua(root-> right);

}

}

Void duyetsau(Node\*root)

{

If(root!=NULL)

{

Duyetsau(root->left)

Duyetsau(root->right)

Cout<<root->data<<””;

}

} temp

//Hàm tạo nút => tạo ra một nút có data là x

X

Node\*create\_node(int x)

NULL NULL

{

Node\*temp;

temp=new Node;

temp->left = NULL; temp->right=NULL;

return temp; root f;p x=65

68

} f;p

89

67

*//Tìm vị trí để thêm nút có data là x vào cây p*

NULL NULL NULL NUL

return f ;

Node\*timvitri(int x,Node\*root)

{ Node\*f,p;

If(root==NULL) return NULL;

else

{

P=roof; f=p;

while (p!=NULL)

{

f=p;

if(x<p->data) p=p->left;

else p=p->right;

}

Return f;

}

}

//Hàm thêm nút vào cây

void themnut(int x, Node\*&root)

{

Node\*temp,f;

Temp = create\_node(x );

If(root==NULL) root=temp;

Else {f=timvitri(x,root);

If(x<f->data) f-> left = temp;

Else f->right = temp;

}

}

***Bài tập:*** *Viết chương trình đọc dãy số từ tệp, đưa vào cây nhị phân tìm kiếm duyệt cây theo thứ tự trước, giữa, sau. Đưa dãy kết quả ra màn hình*

***Viết chương trình xắp xếp 1 mảng sử dụng cây nhị phân tìm kiếm***

* ***Đọc mảng từ tệp***
* ***Xây dựng cây nhị phân***
* ***Duyệt cây theo thứ tự giữa, duyệt đến nút nào đẩy nút đó vào mảng ghi mảng vào tệp***