**Còn 17; 18; 23; 24**

Câu 1: trình bày mối quan hệ giữa cấu trúc dữ liệu và giải thuật. cho ví dụ:

* Giải thuật là một hệ thống chặt chẽ và rõ ràng các qui tắc nhằm xác định các thao tác trên đối tượng, sao cho thu được kết quả mong muốn. <người không học code cũng có thể hiểu>
* Cấu trúc dữ liệu là cách tổ chức dữ liệu trong hệ thống có một cách thứ tự nhằm sử dụng 1 cách hiệu quả
* Có mỗi quan hệ chắt chẽ, luôn đi kèm với nhau. Chương trình = cấu trúc dữ liệu + giải thuật
* Với 1 cấu trúc dữ liệu ta sẽ có giải thuật xử lí tương ứng. cấu trúc dữ liệu thay đổi thì giải thuật cũng thay đổi
* Ví dụ: có danh sách các trường đại học: tên, địa chỉ, sđt. Muốn chỉ ra tên trường là in ra địa chỉ, sđt. Nếu làm theo cách truy xuất lần lượt các tên trường thì gặp phải danh sách dài thì sẽ mất thời gian. Để tối ưu người ta có cách tổ chức thêm bảng mục lục chỉ dẫn theo chữ cái và so sánh với chữ cái đầu vào
* Như vậy có mối quan hệ đó là mối quan hệ mật thiết như hình với bóng

Câu 2: trình bày mối quan hệ giữa cấu trú dữ liệu và các phép toán trên cấu trúc dữ liệu

* Đối với các bài toán phi số học (khối lượng dữ liệu lớn, đa dạng, biến động, phép xử lí không còn đơn giản là phép số học), đi đôi với các cấu trúc dữ liệu mới thì các phép toán mới cũng xuất hiện. các phép toán sẽ có những tác dụng khác nhau đối với từng cấu trúc (phép toán ở đây là tác động lên cấu trúc dữ liệu có thể ví dụ tạo lập, hủy bỏ một cấu trúc, truy nhập, … không chỉ đơn giản là phép toán số học), có phép toán hữu hiệu với cấu trúc này nhưng không hữu hiệu với cấu trúc khác. Vì vậy khi chọn một cấu trúc dữ liệu ta phải nghĩ ngay đến các phép toán tác động đến cấu trúc đó và ngược lại, khi nói tới phép toán thì chú ý phép toán đó được tác động lên cấu trúc dữ liệu nào. Vì vậy người ta thường nói tới cấu trúc dữ liệu là bao gồm cả phép toán tác động lên cấu trúc ấy

Câu 3: trình bày sự khác nhau của cấu trúc dữ liệu và cấu trúc lưu trữ, cho ví dụ

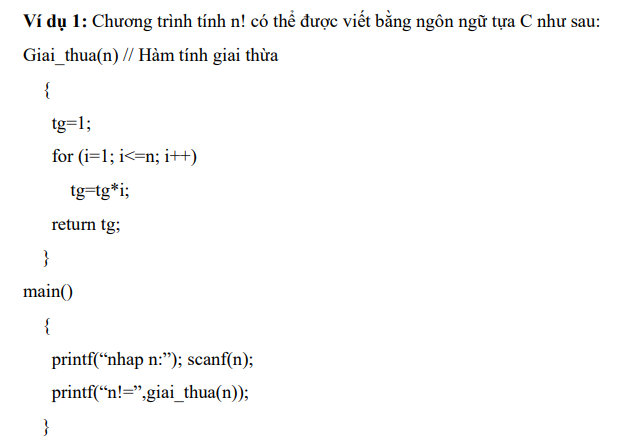
* Các biểu diễn một cấu trúc dữ liệu trên bộ nhớ máy tính được gọi là cấu trúc lưu trữ. Đó chính là cài đặt cấu trúc dữ liệu trên máy tính và trên cơ sở cấu trúc lưu trữ để xử lí. Có thể có nhiều cấu trúc lưu trữ khác nhau trên cùng một cấu trúc dữ liệu và ngược lại
* Ví dụ: cấu trúc dữ liệu cây thì để cho máy hiểu thì ta sẽ dung cấu trúc lưu trữ là mảng hoặc móc nối trên bộ nhớ máy tính. Mặt khác cấu trúc dữ liệu như: danh sách, ngăn xếp đều có thể cài đặt trong qua lưu trữ mảng và móc nối

Câu 4: trình bày những đặc điểm về cấu trúc dữ liệu trong các ngôn ngữ lập trình bậc cao, có liên hệ đến ngôn ngữ C

* Trong các ngôn ngữ lập trình bậc cao, các dữ liệu được phân nhánh thành các kiểu dữ liệu. kiểu dữ liệu nhận của một biến được xác định với tập giá trị mà biến đó có thể nhận và các phép toán có thể thực hiện trên các giá trị đó
* Ví dụ trong C có int có thể nhận giá trị tring miền từ -32768 bđến 32767, các phép toán có thể thực hiện là: bit, logic, so sánh và số học
* Mỗi ngôn ngữ lập trình cho ta các kiểu dữ liệu căn bản, trong các ngôn ngữ khác hau thì các kiểu dữ liệu có thể khác nhau.
* Các kiểu dữ liệu được tạo thành từ nhiều kiểu dữ liệu khác nhau được gọi là kiểu dữ liệu có cấu trúc. Các dữ liệu thuộc kiểu dữ kiệu cấu trúc liên kết với nhau tạo thành cấu trúc dữ liệu
* Trong ngôn ngữ C để liên kết dữ liệu:
* Liên kết dữ liệu cùng kiểu tạo thành mảng
* Liên kết các dữ liệu (không cùng kiểu) thành cấu trúc (struct)
* Sử dụng con trỏ để liên kết dữ liệu

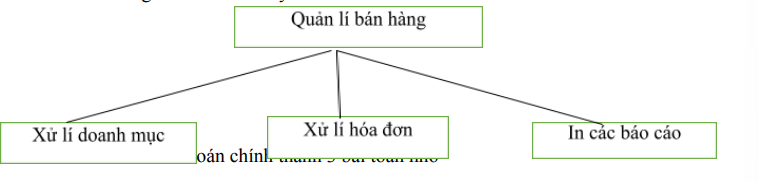
Chú ý khi code giấy sẽ dùng ngôn ngữ tựa C:

* Không có chỉ thị tiền bên dịch: #include
* Sử dụng// để đặt chú thích
* Không cần khai báo biến, mảng trước khi sử dụng
* Không gò bó trong việc mô tả các cấu trúc dữ liệu, mà chỉ cần diễn đạt được ý tưởng
* Không cần khải báo kiểu dữ liệu của hàm và các đối số của hàm
* Hàm scanf và printf không cần đặc tả
* Ngoài những ý trên thì toàn bộ viết với cú pháp ngôn ngữ C
* Ví dụ:

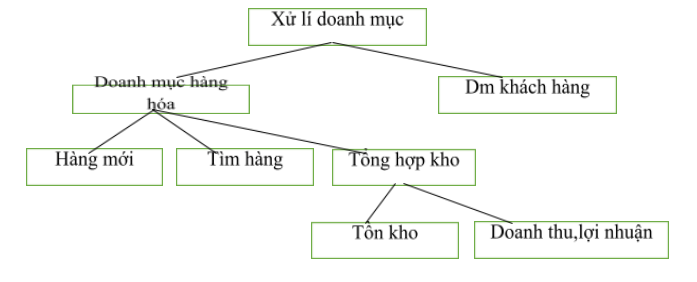


Câu 5: phương pháp thiết kế top down. Cho ví dụ minh họa

* Các bài toán ngày nay rất đa dạng và phức tạp, vì vậy các giải thuật và chương trình để giải quyết chúng cũng có quy mô ngày càng lớn. mọi việc sẽ đơn giản khi ta chia thành các phần nhỏ hơn nghĩa là chia thành các modun nhỏ. Cách giải quyết đó gọi là chiến thuật chia để trị để thực hiện chiến thuật đó, người ta dùng cách thiết kế top\_down. Đó là cách phân tích tổng quát toàn bộ vấn đề đề cập ra những công việc chủ yếu sau đó mới đi dần vào giải quyết các phần việc một cách chi tiết
* Ví dụ: để viết chương trình quản lí bán hang với các yêu cầu là: hang ngày phải nhập các hóa đơn bán hang và nhập hang, tìm kiếm các hóa đơn đã nhập để xem hoặc sửa lại, in các hóa đơn cho khách hang, tính doanh thu, lợi nhuận trong khoảng thời gian bất kì, tính tổng hợp kho, tính doanh số của từng mặt hàng.
* Xuất phát từ bài ta không thể có giải thuật ta có chia ra làm 3 modun theo sơ đồ:



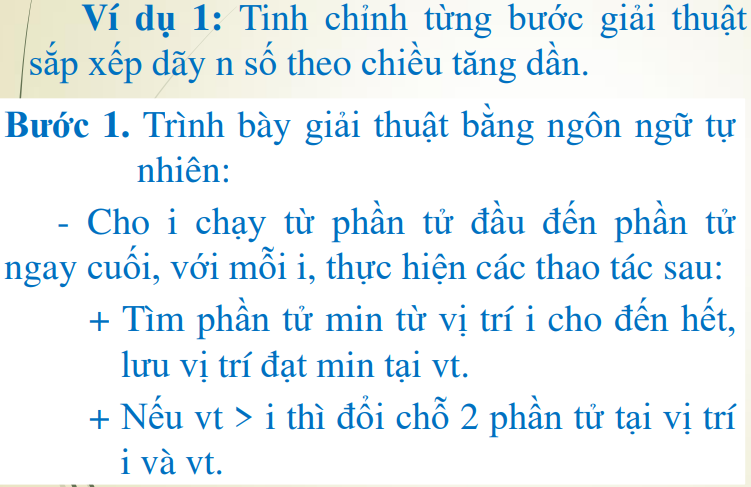
* Các nhiệm vụ ở mức đầu này thường vẫn còn tương đối phức tạp, nên cần phải chia tiếp ví dụ xử lí doanh mục

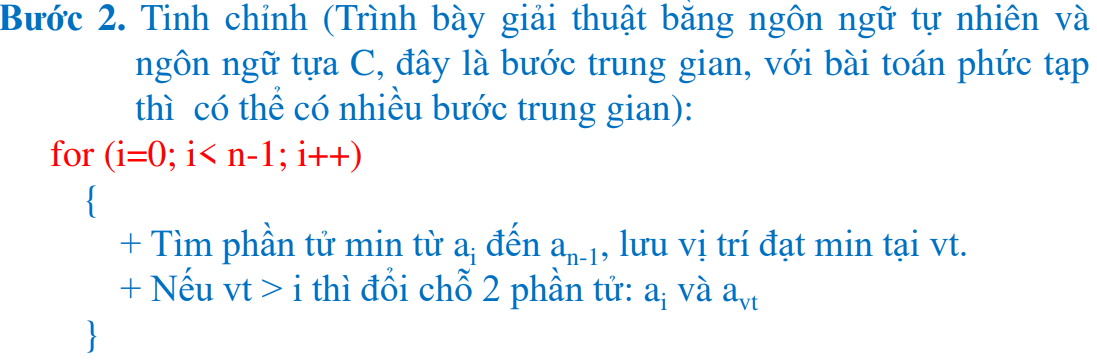


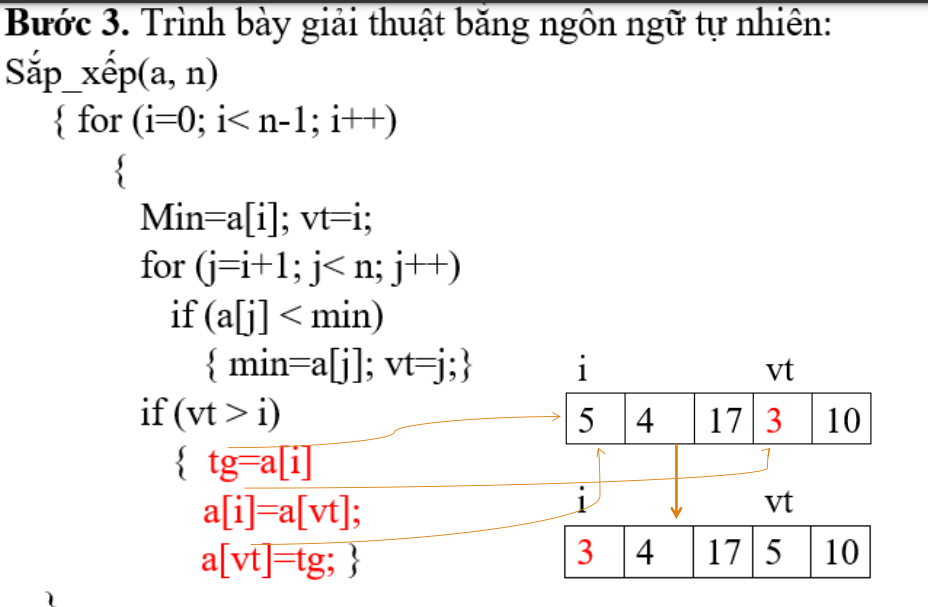
* Cách thiết kế kiểu top-down như trên giúp cho việc giải quyết bài toán được định hướng rõ ràng, thuận tiện cho nhiều người có thể tham gia thiết kế. các modun sẽ độc lập vì vậy khi giải quyết sẽ không ảnh hưởng đến nhau.
* Trong thực tế, việc phân tích bài toán thành các bài toán con không phải việc dễ dàng. Chinh vì vậy mà có những bài toán sẽ mất nhiều thời gian và công sức hơn cả nhiệm vụ lập trình

Câu 6: phương pháp tinh chỉnh từng bước

* Tinh chỉnh là bước phương pháp thiết kế giải gắn liền với lập trình
* Ban đầu chương trình thể hiện giải thuật được trình bày bằng ngôn ngữ tự nhiên, phản ánh ý chính của công việc cần làm. Quá trình thiết kế giải thuật sẽ được thể hiện dần từ dạng ngôn ngữ tự nhiên qua giả ngôn ngữ rồi đến ngôn ngữ lập trình và đi từ từ mức làm cái gì đến làm như thế nào ngày càng sát với các chức năng ứng với các câu lệnh của ngôn ngữ lập trình đã chọn



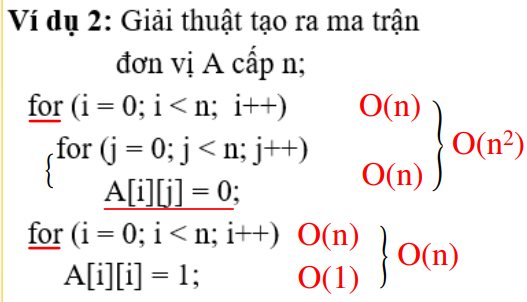




Câu 7: trình bày cách phân tích thời gian thực hiện giải thuật. định nghĩa O lớn

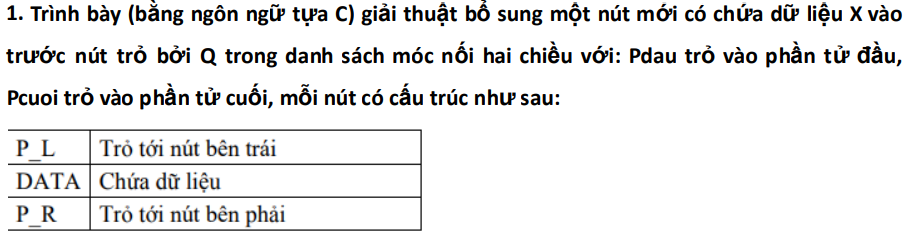
* thời gian thực hiện một giải thuật phụ thuộc và rất nhiều yếu tố. yếu tố cần chú ý trước tiên là kích thước dữ liệu đưa vào. Chẳng hạn thời gian sắp xếp 1 dãy sẽ chịu ảnh hưởng của độ dài dãy. Nếu gọi n là số lượng thì thời gian thực hiện T của một giải thuật được biểu diễn như 1 hàm của n: T(n)
* các kiểu lệnh và tốc độ xử lí của máy tính ngôn ngữ viết chương trình và chương trình dịch ngôn ngữ đó đều ảnh hưởng tới thời gian thực hiện, nhưng những yếu tố đó không đồng đều trên mỗi loại máy vì vậy không thể dựa vào chúng khi xác lập T(n). điều đó cũng có nghĩa là T(n) không thể biểu diễn thành đơn vị thời gian giờ, phút, giây,… (đơn vị mới sẽ được gọi là số phép tính cơ sở)
* tuy nhiên không phải vì thế mà không thể so sánh được các giải thuật về mặt tốc độ. Ví dụ nếu thời gian thực hiện (được biểu diễn dạng hàm) của giải thuật 1 là cn2 và thời gian của giải thuật 2 là kn với c và k là 1 hằng số. khi với n lớn thì giải thuật 2 sẽ ít hơn so với giải thuật 1. Và như vậy nếu nói thời gian thực hiện giải thuật T(n) tỉ lệ với n2 hay tỉ lệ với n cũng cho ta ý niệm về tốc độ thực hiện giải thuật đó khi n khá lớn (với n nhỏ thì việc xét T(n) không có ý nghĩa)
* ta nói f(n) là ô lớn của g(n) và viết f(n) = O(g(n)) nếu tồn tại hằng số dương c và n0 sao cho f(n)<=cg(n) khi n>=n0 nghĩa là f(n) bị chặn trên bởi một hằng số nhân với g(n) với mọi n từ một thời điểm nào đó

Câu 8: trình bày cách xác độ phức tạp tính toán của giải thuật, với những nội dung: qui tắc tổng, phép toán tích cực, thời gian chạy của các câu lặp lệnh, cho ví dụ

* nếu thời gian thực hiện một thuật toán là T(n) = cn2 thì độ phức tạp của giải thuật này có cấp là n2 kí hiệu là: T(n) = O(n2). Một cách tổng quát ta nói f(n) là ô lớn của g(n) và viết f(n) = O(g(n)) nếu tồn tại hằng số dương c và n0 sao cho f(n)<=cg(n) khi n>=n0 nghĩa là f(n) bị chặn trên bởi một hằng số nhân với g(n) với mọi n từ một thời điểm nào đó
* quy tắc tổng: giả sử T1(n) và T2(n) là thời gian thực hiện của chương trình 1 và 2 có T1(n) = O(f(n)); T2(n) = O(g(n)) thì thời gian thực hiện chương trình 1 và chương trình 2 sẽ là T1(n)+T2(n) = O(max(f(n), g(n))
* ví dụ: trong 1 chương trình có 3 bước thực hiện mà độ phức tạp tính toán từng bước lần lượt là O(n 2 ), O(n 3 ) và O(n log2n) thì độ phức tạp tính toán của 2 bước đầu là O(max(n 2 ,n 3 ))=O(n 3 ) độ phức tạp tính toán của chương trình sẽ là: O(max(n 3 , n log2n))=O(n 3 ).
* Vòng lặp: For: Nếu một vòng lặp for duyệt qua một mảng có kích thước n, thì độ phức tạp là O(n); While: Độ phức tạp của một vòng lặp while phụ thuộc vào điều kiện dừng của nó. Trong trường hợp tốt nhất, vòng lặp sẽ không chạy (O(1)), trong trường hợp xấu nhất, nó có thể chạy O(n).
* Ví dụ:
* phép toán tích cực: Đó là phép toán thuộc giải thuật mà độ phức tạp tính toán không ít hơn độ phức tạp tính toán của phép khác hay nói cách khác: số lần thực hiện nó không kém các phép khác. Tại các chương trình chia làm các giải thuật nơi nào có phép toán tích cực thì sẽ tính độ phức tạp nếu không phải sẽ loại
* Ví dụ: có phép toán tích cự là A[i][j] = 0 

**Left: trái; right: phải**

Câu 9:



* Them\_nut(Pdau, Pcuoi, Q, X) //X: dữ liệu cần them

{

Node MOI = malloc(); //khởi tạo nút mới

MOI ->DATA = X;

MOI ->P\_L = NULL;

MOI->P\_R = NULL;

if (Pcuoi == NULL)

{

//list rỗng

Pdau =MOI;

Pcuoi=MOI;

{

else if (Q ==Pdau)

{

//Q là list đầu

Pdau=MOI;

MOI -> P\_R = Q;

Q -> P\_L = MOI;

}

else

{

// Q ở giữa hoặc cuối

MOI ->P\_L = Q ->P\_L;

MOI->P\_R = Q;

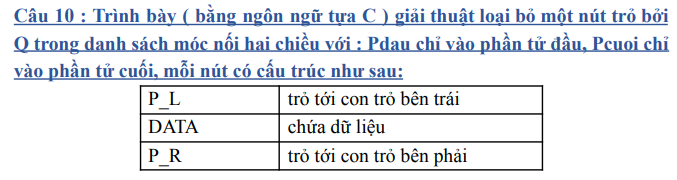
Q->P\_L = MOI;

MOI -> P\_L -> P\_R = MOI;

}

}

Câu 10:



* Xoa\_nut(Pdau, Pcuoi, Q)

{

if (P\_cuoi ==NULL)

print(“List empty”);

else if (Pdau == Pcuoi)

{

//có 1 phần từ

Pdau = Pcuoi = NULL;

}

else if (Pdau == Q && Pcuoi != Q)

{

//Q ở đầu

Pdau = Pdau ->P\_R; //chuyển Pdau thành chứa địa chỉ của thành phần tiếp theo

Pdau -> P\_L = NULL;// ngắt liên kết của thành phần kế tiếp

}

else if (Pcuoi == Q && Pdau != Q)

{

//Q ở cuối, ngược lại

Pcuoi = Pcuoi ->P\_R;

Pdau -> P\_R = NULL;

}

else

{

// Q ở giữa

Q->P\_L->P\_R = Q->P\_R;

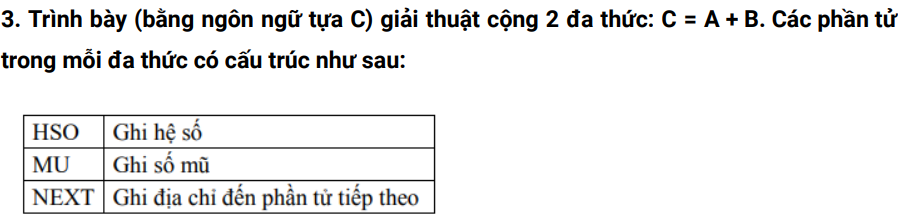
Q->P\_R->P\_L = Q->P\_L;

}

free(Q); // giải phóng Q

}

Câu 11:



* Tong = NULL; // biến toàn cục
* taoNUT (HSO, MU)

{

NUT = malloc();

NUT ->HSO = H;

NUT->MU = M;

NUT->NEXT = NULL;

return NUT;

}

* themNUT (p)

{

if (TONG == NULL)

{

TONG = p;

}

else

{

k = TONG;

while (k -> next != NULL)

{

k = k -> next;

}

k->next = p;

{

}

* CongDaThuc (A, B, Tong)

{

p = NULL;

while (A != NULL && B != NULL)

{

if (A->HSO > B->HSO)

{

p = taoNUT(A -> HSO, A -> MU);

themNUT(p);

A = A -> NEXT;

}

else if (A -> HSO < B -> HSO)

{

p = taoNUT(B->HSO, B->MU);

themNUT(p);

B = B -> NEXT;

}

else if (A->HSO == B->HSO)

{

p = taoNUT(A->HSO + B->HSO, A->MU);

themNUT(p)

A = A -> NEXT;

B = B -> NEXT;

}

}

if (A == NULL)

{

While (B != NULL)

{

p = taoNUT(B->HSO, B->MU);

themNUT(p);

B = B -> NEXT;

}

}

else

{

While (A != NULL)

{

p = taoNUT(A->HSO, A->MU);

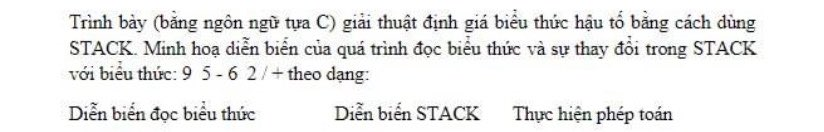
themNUT(p);

A = A -> NEXT;

}

}

Câu 12:



* DINH\_GIA\_BIEU\_THUC ()

{

// giải thuật này sử dụng một ngăn xếp S, được trỏ bởi con trỏ T

Do

{

//đọc phần tử X tiếp theo trong biểu thức

If (X là toán hạng)

PUSH (S,T,X) // đẩy phần tử vào S

Else

{

// lấy phần tử từ S

Y = POP (S,T);

Z = POP (S,T);

W = Z X Y; // thực hiện phép toán

PUSH (S,T,W)

}

}

While (gặp dấu kết thúc);

R = POP (S,T)

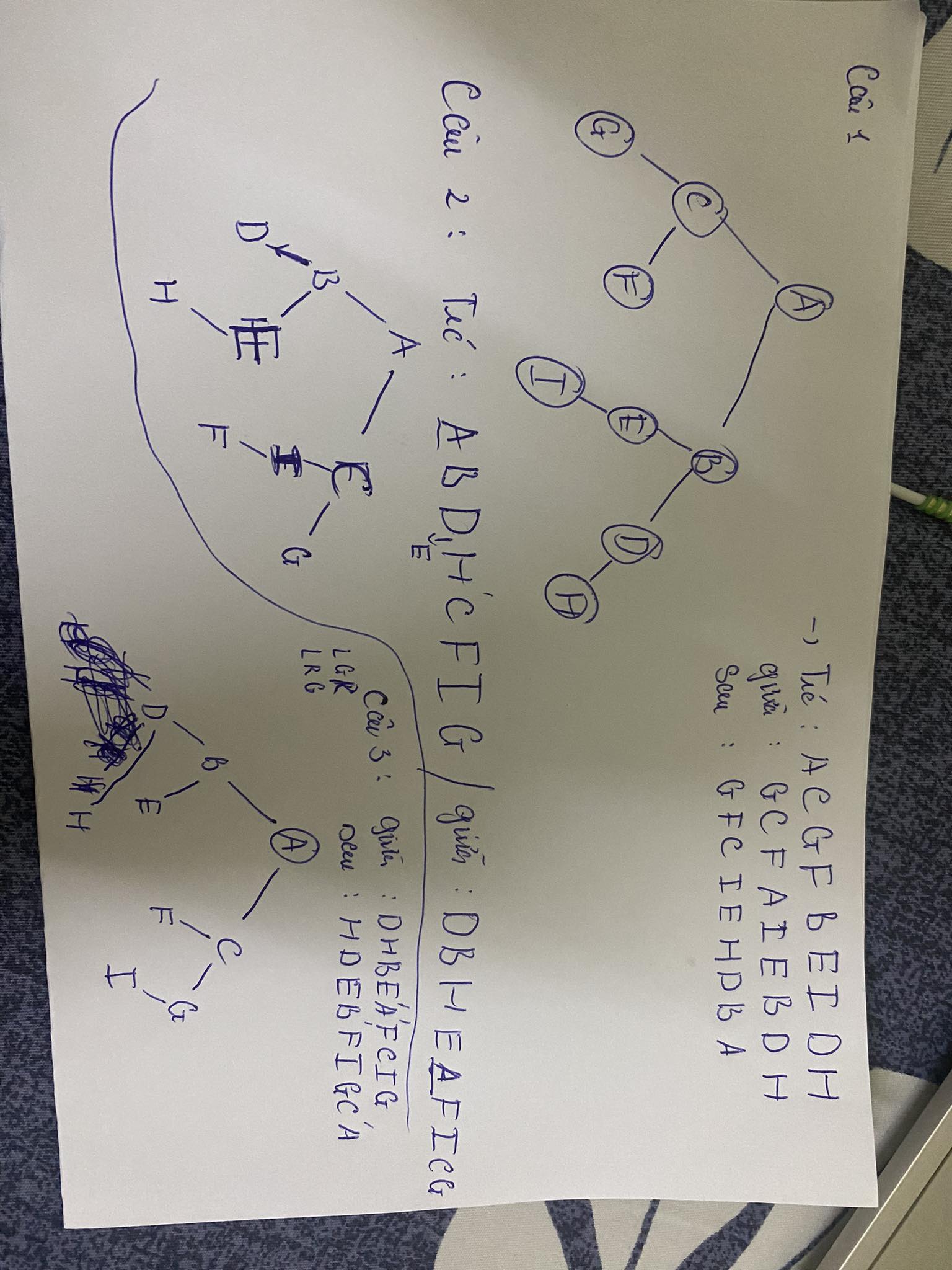
Printf (R);

}

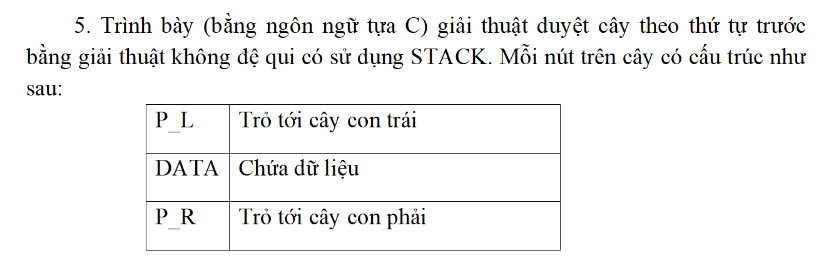
* Biểu diễn:

|  |  |  |
| --- | --- | --- |
| Đọc thức | STACK | Thực hiện phép toán |
| 9 5 – 6 2 / + | 9 | Đẩy 9 vào stack |
| 5 – 6 2 / + | 5  9 | Đẩy 5 và stack |
| - 6 2 / + | 4 | Lấy lần lượt 5 và 9 ra khỏi stack rồi thực hiện phép toán 9-5 |
| 6 2 / + | 6  4 | Đẩy 6 vào stack |
| 2 / + | 2  6  4 | Đẩy 2 vào stack |
| / + | 3  4 | Lấy lần lượt 2 và 6 ra khỏi stack rồi thực hiện phép toán 6/2 |
| + | 7 | Lấy lần lượt 3 và 4 ra khỏi stack rồi thực hiện phép toán 4+3 |

**Duyệt cây nhị phân**

* Duyệt cây: thứ tự trước: gốc -> duyệt cây con trái theo thứ tự trước -> duyệt cây con phải theo thứ tự trước; thứ tự giữa: duyệt cây con trái theo thứ tự giữa -> gốc -> duyệt cây con phải theo thứ tự giữa; thứ tự sau: duyệt cây con trái theo thứ tự sau -> duyệt cây con phải theo thứ tự sau -> gốc
* Thần chú trước: GLR; giữa: LGR; sau: LRG
* 

Câu 13:



TT\_TRUOC (T)

{

If (T == NULL)

{

Printf(“cây rỗng);

Return;

}

Else

{

TOP = -1;

PUSH (S, TOP, T);

While (TOP > -1) //trong stack còn dữ liệu hay không

{

P = POP (S,TOP);

While (P!=NULL)

{

Printf(P->Data);

If(P->P\_R != NULL)

{

PUSH (S,TOP,P->P\_R)

}

P = P ->P\_L;

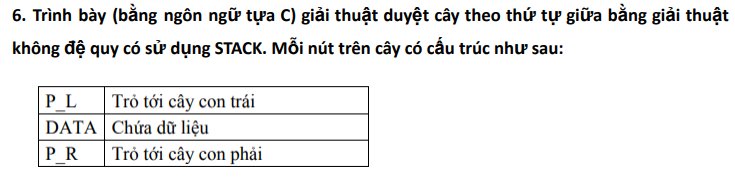
}

}

}

}

Câu 14:



TT\_GIUA (T)

{

If (T == NULL)

{

Printf(“cây rỗng);

Return;

}

Else

{

TOP = -1;

P = T; // gán P là nút gốc

While ((P != NULL) || (TOP > -1))

{

While (P!=NULL)

{

PUSH(S ,TOP, P);

P = P ->P\_L;

}

P = pop(S,TOP);

Printf(P->DATA);

P = P->P\_R;

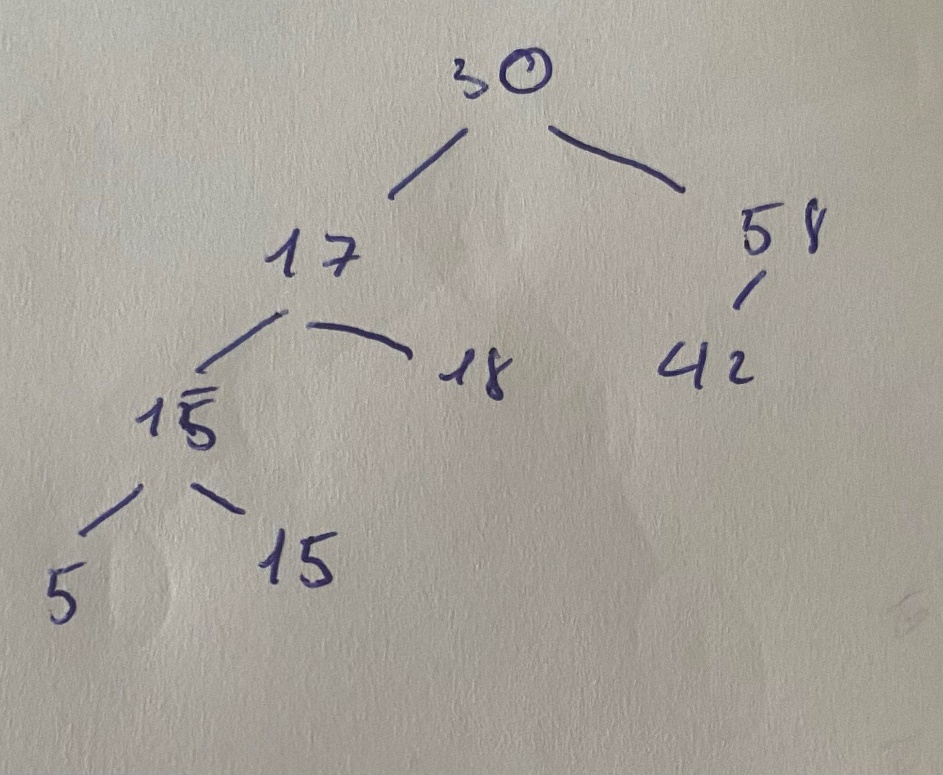
}

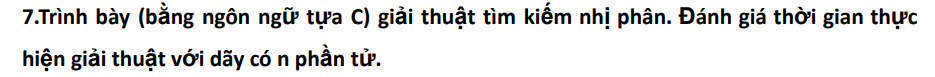
}

}

**cây nhị phân tìm kiếm**

* Cây nhị phân tìm kiếm là cây trong đó các phần tử của cây con bên trái đều nhỏ hơn phần tử hiện hành và các phần tử cây con bên phải đều lớn hơn giá trị hiện hành. Nếu bằng thường được viết bên phải
* 



Câu 15: 

thuật toán tìm kiếm nhị phân là một phương pháp tìm kiếm hiệu quả trong một mảng đã được sắp xếp.

Binary\_Search (K, size, x)

{

i = 0; //vị trí phần tử đầu tiên

p = size -1; //vị trí phần tử cuối

while (t <=p)

{

G = (p+t)/2; // vị trí giữa

If(K[g] == x)

{

Return g;

}

If (x < K[g])

{

P = g -1;

}

Else

{

P = g+1;

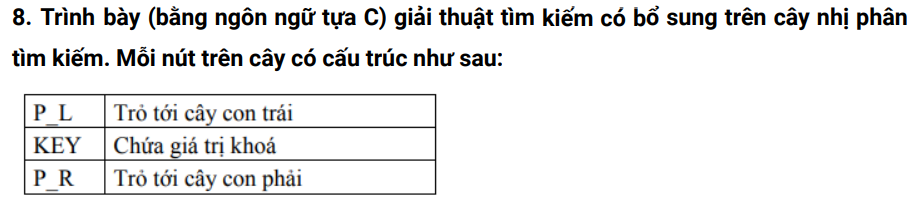
}

}

Return -1;

}

* Xét giải thuật trên ta thấy trường hợp tốt nhất là trường hợp vị trí cần tìm ở giữa mảng thì độ phức tạp là: O(1). Trường hợp xấu nhất là ở đầu hoặc cuối hoặc không có trong mảng thì độ phức tạp O(log2(n)).

Câu 16: 

BST (T,x,P)

{

// tìm giá trị x trên cây T nếu thấy thì trỏ P vào, nếu không thì khởi tạo thêm nút rồi trỏ P vào

P = T; // P = root

Q = NULL;

While (P != NULL)

{

If (P->key == x)

{

Return P;

}

Q = P;

Else If (x < P->key)

{

P = p -> P\_L;

}

Else if (x > P->key)

{

P = P->P\_R

}

}

// trường hợp không có

P = malloc();

P ->key = x;

P ->P\_L = NULL;

P ->P\_R = NULL;

If (x < Q->key)

{

Q -> P\_L = P;

}

Else

{

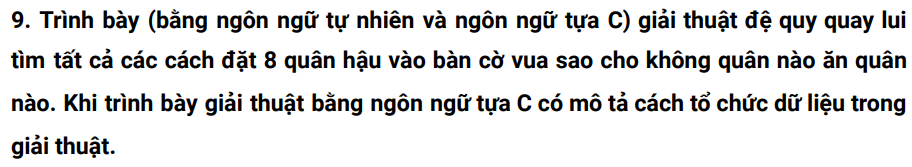
Q -> P\_R = P;

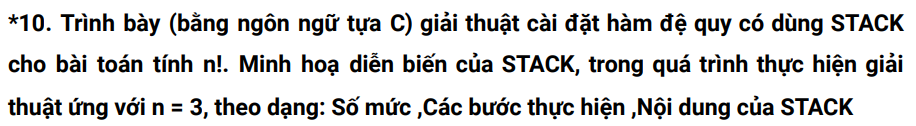
{

Printf(“khong tim thay da bo sung”);

Return P;

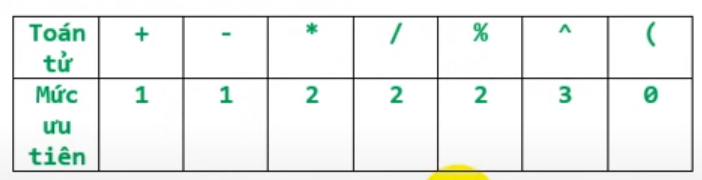
}

Câu 17: 

Câu 18: 

Câu 19:



* Mức độ ưu tiên: 
* Quy tắc:
* Nếu là toán hạng thì ghi.
* Nếu là dấu ( thì push vào
* Nếu là ) thì pop ra các giá trị trong ( ) từ stack và loại bỏ 2 dấu ( )
* Nếu là toán tử thì nếu mức độ ưu tiên mới và <= mức độ ưu tiên bên dưới thì pop ra giá trị ưu tiên bên dưới để ghi ưu tiên mới.
* Ngôn ngữ tự nhiên:
* Khởi tạo 1 stack rỗng
* Đọc lần lượt các thành phần trong biểu thức X nếu X là toán hạng sẽ được in ra. Nếu x là toán tử thì:
* Stack không rỗng thì nếu phần tử ở đỉnh là phép toán có độ ưu tiên cao hơn hoặc bằng phép toán ở X thì phép toán đó được kéo ra khỏi stack và in ra
* Nếu stack rỗng hoặc dấu mở ngoặc hoặc đỉnh stack có độ ưu tiên thấp hơn phép toán X thì đẩy vào ngăn xếp phép toán tại X
* Nếu X là dấu đóng ngoặc thì in ra các phần tử trong stack đến khi gặp dấu mở ngoặc và loại bỏ dấu mở ngoặc
* Sau khi thực hiện đến khi stack rỗng ta sẽ có biểu thức hậu tố
* Ngôn ngữ tựa C:

Chuyen\_TrungTo\_sang\_HauTo()

{

// sử dụng stack S và con trỏ Top để quản lí

Do

{

Đọc phần tử X trong biểu thức

If(X là toán hạng)

Printf(X)

Else if (X là phép toán)

{

Do

{

If (Top > -1 && Top là phép toán có độ ưu tiên cao hơn hay bằng X)

{

Printf(pop(S,Top));

}

If (Top==-1 || Top =’(‘ || Top là phép toán cho độ ưu tiên thấp hơn X)

{

Push(S, Top, X);

}

}

While (phép X chưa được đưa vào stack);

}

Else if (X == ’(‘ )

{

Push(S,Top,X);

}

Else if (X == ‘)’ )

{

Do

{

Printf(pop(S,Top));

}

While (Top != ‘)’ );

Pop(S, Top);

}

}

While (chưa gặp kết thúc của biểu thức);

If (Top != 1)

{

Do

{

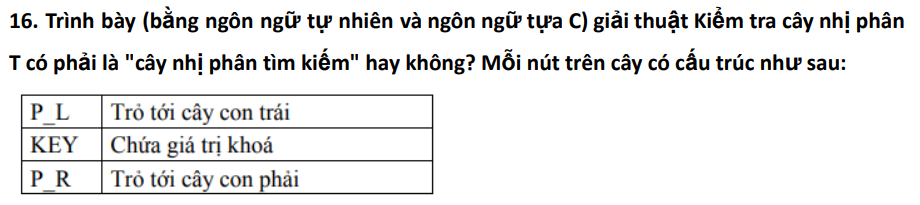
Printf(pop(S,Top));

}

While (T>-1);

}

}

Câu 20: 

Kiemtra(T)

{

if (T == NULL)

Return true;

Top = -1;

P = T;

NutTrc = NULL;

While ( Top >-1 || P != NULL);

{

While (P != NULL)

{

PUSH(S, Top, P); // lưu địa chỉ root

P = P -> P\_L; // chuyển đến con trái

}

P = pop(S, Top);

If (NutTrc != NULL && P -> Key < NutTrc ->Key)

{

Return false;

}

Else

{

NutTrc = P;

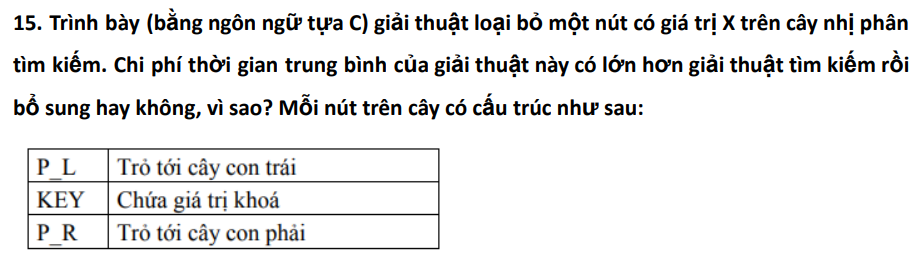
}

P = P -> P\_R;

}

Return true;

}

Câu 21: 

Delete(T,x)

{

P = T;

Q = NULL;

While (P!=NULL)

{

If (P->key =x)

{

Break;

}

Q = P;

If (x < Q->key)

{

P = P ->P\_L;

}

Else

{

P = P -> P\_R;

}

}

If (P ==NULL)

{

Printf (“không tìm thấy”);

Return;

}

// đã tìm ra vị trí cần xóa

If ( P->P\_L != NULL && P->P\_R!=NULL) //P có đầy đủ 2 cây con

{

Node = P; // ghi nhớ nút cần xóa

While (P -> P\_R!=NULL)

{

Q = P;

P = P -> P\_R;

}

Node -> key = P ->key; // gán lại giá trị cho nút cần xóa

}

If (P -> P\_L != NULL)

{

Child = p -> P\_L;

}

Else

{

Child = p->P\_R;

}

If (P==T)

{

T = Child;

}

Else if (Q->P\_L == P)

{

Q -> P\_L = Child;

}

Else

{

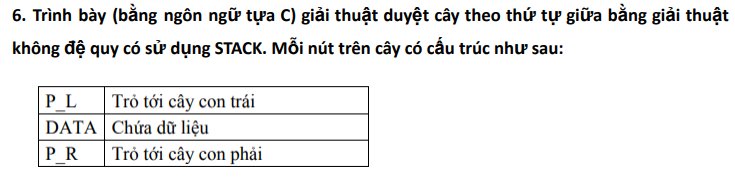
Q -> P\_R = Child;

}

Free (p);

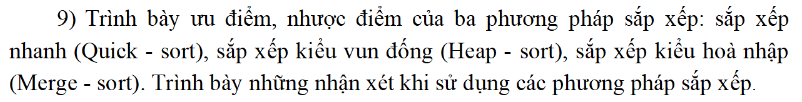
}

Câu 22: Duyệt cây thứ tự sau



=>>> Nó tương tự thuật toán thứ tự trước nhưng ngược lại (gốc – phải – trái và đọc cũng ngược lại)

Câu 23:



Câu 24:

