Mã đề
**CD 2011 - 01**

# TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
# VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
# BỘ MÔN KHOA HỌC MÁY TÍNH

***

| | | |
|---|---|---|
| **Họ tên**: ……<br>**Lớp**: ……<br>**SHSV**: …… | **ĐỀ THI MÔN: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**<br>**Ngày thi**: ……/……/….<br>**Thời gian 90'**<br>(**Sinh viên được sử dụng tài liệu**) | Hà nội, ….. /….. / ……<br>***Trưởng bộ môn*** |

**Bài 1.**

a) Phân biệt giữa mảng cấp phát động và mảng cấp phát tĩnh. Khi nào nên dùng mảng cấp phát động, hoặc mảng cấp phát tĩnh? Cho ví dụ minh họa. **(1 Điểm)**

| | **Mảng cấp phát tĩnh** | **Mảng cấp phát động** |
|---|---|---|
| Bộ nhớ | Bộ nhớ lấy từ phần DATA | Bộ nhớ lấy từ HEAP |
| Kích thước | Bộ nhớ giới hạn, chỉ có thể cấp phát cho các biến có kích thước nhỏ | Dung lượng bộ nhớ lớn, có thể cấp phát được cho các biến kích thước lớn |
| Thời điểm cấp phát | Bộ nhớ được cấp phát tại thời điểm biên dịch chương trình | Bộ nhớ được cấp phát tại thời điểm chạy |
| Thu hồi bộ nhớ | Hệ điều hành sẽ tự thu hồi bộ nhớ khi không dùng đến | Người lập trình phải tự thu hồi bộ nhớ xin cấp phát |

**Cấp phát tĩnh**: cho các biến kích thước nhỏ, các biến đơn

```
float a,b, Ar[100];
```

**Cấp phát động**: Dùng cho các biến kích thước lớn, các mảng lớn,…

```
double *A;
A = (double*)malloc(10000*sizeof(double));
```

b) Đánh giá thời gian thực hiện tồi nhất của hàm sau theo O-lớn **(1 Điểm)**

```
double fastPower(double x, int n)
{
   double fract;
   if(n==0) return 1;

   if(n%2==0) return fastPower(x,n/2)*fastPower(x,n/2);
   else return fastPower(x,n/2)*fastPower(x,n/2)*x;
}
```

Hàm trên được cài đặt đệ quy, lời gọi đệ quy là `fract = fastPower(x,n/2);`
Được gọi 2 lần trong hàm (ứng với if hoặc else), ta có công thức đệ quy tổng quát là

$$T(n) = \begin{cases} 1 \text{ nếu } n = 0 \\ 2T(n/2) + 1 \text{ nếu } n > 0 \end{cases}$$

Ta có thể viết gọn lại là $T(n) = 2T\left(\frac{n}{2}\right) + 1$
Áp dụng định lý thợ với $a = 2, b = 2$ và $f(n) = 1$
$n^{\log_b a} = n^{\log_2 2} = n$ → trường hợp 1 của định lý thợ
Vậy kết luận $T(n) = \theta(n)$

c) So sánh ưu nhược điểm của phương pháp tổ chức tìm kiếm dùng mảng và áp dụng thuật toán tìm kiếm nhị phân, cây nhị phân tìm kiếm và dùng bảng băm theo các tiêu chí sau **(1 Điểm)**

| **Tiêu chí** | **Tìm kiếm nhị phân** | **Cây nhị phân tìm kiếm** | **Bảng băm** |
|---|---|---|---|
| Bộ nhớ dùng lưu trữ các phần tử | $O(n)$<br>Tỉ lệ với số phần tử, tuy nhiên mỗi phần tử không phải lưu trữ thêm dữ liệu thừa | $O(n)$<br>Tỉ lệ với số phần tử, tuy nhiên mỗi phần tử phải lưu trữ thêm dữ liệu thừa (2 con trỏ) | Số lượng ô nhớ xác định trước, là kích thước bảng băm (thường lớn hơn số lượng phần tử cần lưu nhiều lần) |
| Thời gian tìm kiếm | $O(\log n)$ | $O(\log n)$ | $O(1)$ |
| Thêm phần tử | $O(n)$ | $O(\log n)$ | $O(1)$ |
| Xoá phần tử | $O(n)$ | $O(\log n)$ | $O(1)$ |
| In ra danh sách các phần tử hiện có | $O(n)$ | $O(n)$ | Không hỗ trợ thao tác này, nếu muốn in ta phải duyệt toàn bộ bảng băm |

**Bài 2.**

a) Biểu thức dạng hậu tố là gì? Ưu điểm của biểu thức dạng hậu tố? **(1 Điểm)**

**Biểu thức dạng hậu tố**: Là cách biểu diễn biểu thức trong đó toàn tử đứng sau các toán hạng mà nó tác động
Ví dụ: 3 5 + a –
**Ưu điểm** của biểu thức dạng hậu tố là chỉ có một cách định giá (cách tính) duy nhất. Không như biểu thức dạng trung tố cần quy định thêm về độ ưu tiên của toán tử, và dấu ngoặc.
Biểu thức dạng hậu tố được dùng để biểu diễn biểu thức trong máy tính

b) Chuyển biểu thức dạng trung tố sau sang dạng hậu tố **(1 Điểm)**

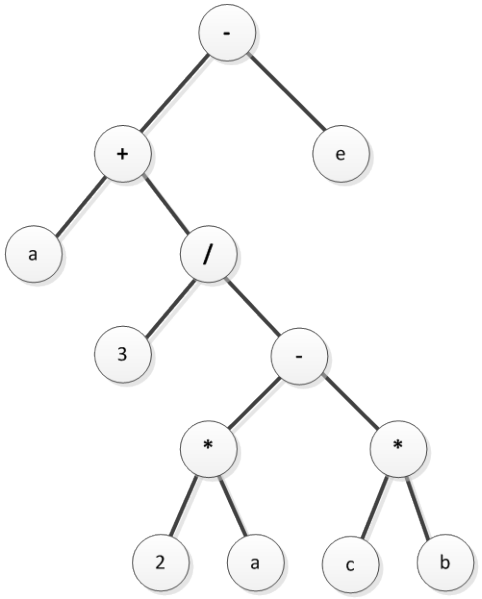
$$a + 3/(2 * a - c * b) - e$$

| Gặp | STACK |
|---|---|
| a | ∅ |
| + | + |
| 3 | + |
| / | +,/ |
| ( | +,/,( |
| 2 | +,/,( |
| * | +,/,(,* |
| a | +,/,(,* |

| | |
|---|---|
| - | $+, /, (, -$ |
| c | $+, /, (, -$ |
| * | $+, /, (, -, *$ |
| b | $+, /, (, -, *$ |
| ) | $+, /$ |
| - | $-$ |
| e | $-$ |

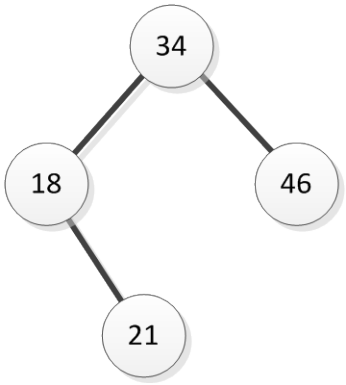
**Biểu thức kết quả**: $a\ 3\ 2\ a\ *\ c\ b\ *\ -\ /\ +\ e\ -$

c) Vẽ cây biểu thức biểu diễn cho biểu thức ở phần b (không cần phải trình bày các bước trung gian) **(1 Điểm)**

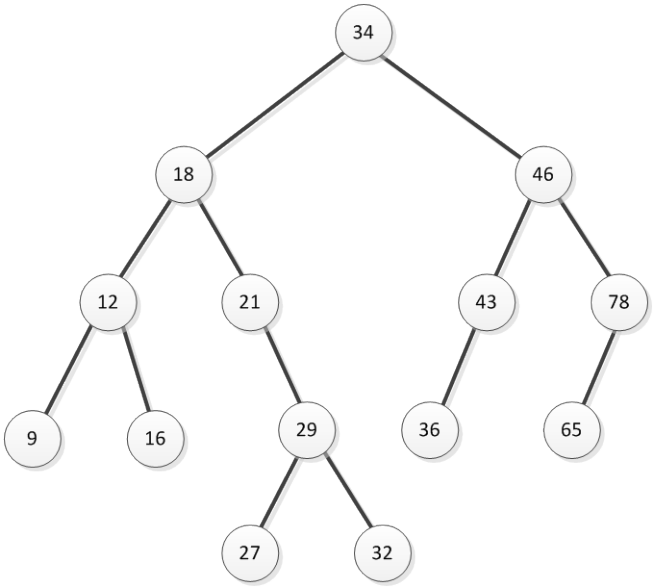



**Bài 3.**

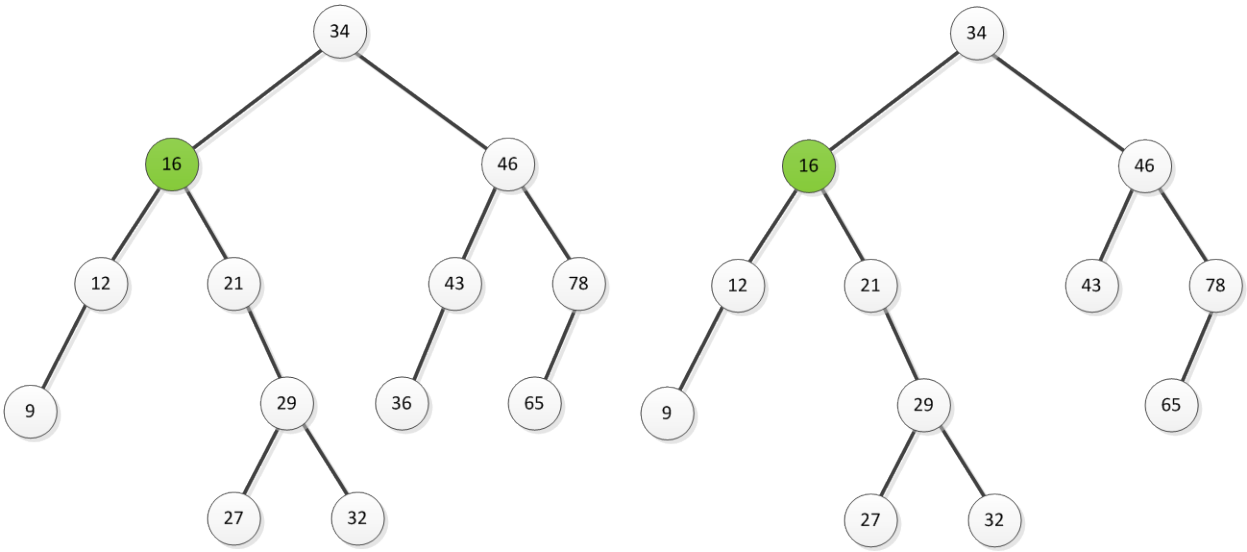
a) Cho cây nhị phân tìm kiếm ban đầu như hình




thêm lần lượt dãy khóa 43, 12, 36, 78, 29, 16, 9, 65, 27, 32. Hãy vẽ cây nhị phân kết quả thu được cuối cùng (không cần trình bày các bước trung gian). **(1 Điểm)**




b) Với cây nhị phân tìm kiếm thu được ở phần a, thực hiện xóa lần lượt khóa 18 và 36. Hãy vẽ cây kết quả thu được sau mỗi lần xóa
Chú ý: chọn nút thay thế là nút phải nhất trên cây con trái
**(1 Điểm)**

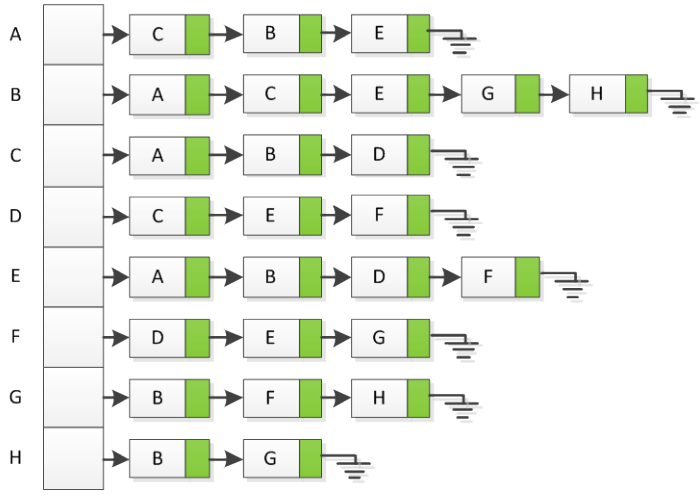



**Bài 4.** Cho một ***đơn đồ thị vô hướng*** $G(V, E)$ như sau

$V = \{A, B, C, D, E, F, G, H\}$

$E = \{(A, B), (A, C), (A, E), (B, E), (B, G), (C, D), (C, B), (D, E), (F, D), (F, E), (F, G), (H, B), (H, G)\}$

a) Hãy biểu diễn đồ thị trên dùng danh sách kề **(1 Điểm)**




b) Thực hiện DFS từ đỉnh D, hãy đưa ra thứ tự các đỉnh được thăm.

**(1 Điểm)**

**Chỉ cần vẽ được hình trạng hàng đợi hoặc cây khung DFS tại D là được đủ điểm**

| STT | STACK |
|---|---|
| 1 | D |
| 2 | D, C |
| 3 | D, C, A |
| 4 | D, C, A, B |
| 5 | D, C, A, B, E |
| 6 | D, C, A, B, E, F |
| 7 | D, C, A, B, E, F, G |
| 8 | D, C, A, B, E, F, G, H |
| 9 | D, C, A, B, E, F, G |
| 10 | D, C, A, B, E, F |
| 11 | D, C, A, B, E |
| 12 | D, C, A, B |
| 13 | D, C, A |
| 14 | D, C |
| 15 | D |
| 16 | Ø |

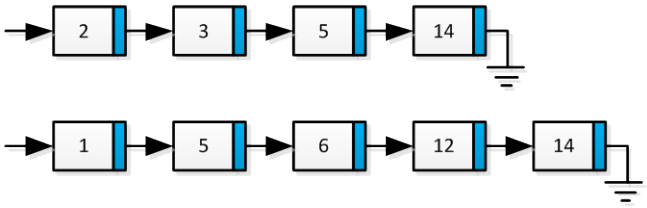

c) Hãy đưa ra các loại cạnh thu được khi DFS tại đỉnh D (BackEdge, CrossEdge, TreeEdge và ForwardEdge).

Lưu ý: Các đỉnh trên đồ thị được thăm theo thứ tự ABC **(1 Điểm)**

| | |
|---|---|
| Cạnh cây<br>Tree-Edge | DC, CA, AB, BE, EF, FG, GH |
| Cạnh ngược<br>Back-edge | BC, ED, FD, GB, HB, EA |
| Cạnh tới<br>Forward-Edge | |
| Cạnh vòng<br>Cross-Edge | |

**Bài 5.** Để biểu diễn các tập hợp số nguyên ta dùng danh sách liên kết đơn với cấu trúc một phần tử được khai báo như sau:

```
typedef struct Node
{
     int data;
     struct node *pNext;
} NODE;
```




a) Hãy xây dựng hàm tìm và trả về giá trị phần tử chẵn lớn nhất trong tập hợp trong trường hợp biết các phần tử của tập hợp được sắp xếp theo thứ tự tăng dần về giá trị. **(1 Điểm)**

```
int FindMax (NODE *pHead)
{

}
```

b) Hãy đánh giá thời gian thực hiện trong trường hợp tồi nhất của hàm bạn viết theo O-lớn **(0.5 Điểm)**

```
int FindMax (NODE *pHead)
{
    NODE *ptr = pHead;
    int i= 0; //biến để đếm vị trí các phần tử
    int pos=-1;
    while(ptr!=NULL)
    {
           if(ptr->data%2==0)
           {
                  pos = i;
           }
           ptr = ptr->pNext;
           i++;
    }
    return pos;
}
```

Dễ thấy thời gian thực hiện của thuật toán trong trường hợp tồi nhất cỡ $O(n)$

**Tổng điểm: 12.5 điểm**