TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HCM
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

KỲ THI CUỐI KỲ – HỌC KỲ 1 / 2016-2017
Môn thi: Cấu trúc dữ liệu & giải thuật [CO2003]
Đáp án

1 Câu hỏi trắc nghiệm

1.1 LO 7.4

Cho đồ thị có hướng với tập đỉnh $V = \{V_0, V_1, V_2, V_3, V_4, V_5\}$ cùng tập cạnh $E = \{<V_0, V_1>, <V_0, V_3>, <V_1, V_5>, <V_2, V_0>, <V_2, V_3>, <V_3, V_4>, <V_4, V_1>, <V_4, V_5>\}$. Hãy cho biết thứ tự topo (topological order) của đồ thị trên (dùng phương pháp Depth-First với các đỉnh được chọn theo thứ tự tăng dần).

$V_2, V_0, V_3, V_4, V_1, V_5$

1.2 LO 7

Nhận định nào dưới đây SAI.

Phương pháp duyệt theo chiều sâu không áp dụng cho đồ thị có hướng.

1.3 LO 3.1

Xây dựng cây nhị phân tìm kiếm với mỗi dãy số sau, trình tự chèn các giá trị vào cây theo đúng thứ tự của dãy. Dãy nào dưới đây tạo ra cây nhị phân khác với 3 dãy số còn lại.

(100, 60, 80, 90, 120, 110, 130)

1.4 LO 4.2

Dùng dãy (mảng) để biểu diễn heap, trong những dãy số dưới đây, dãy nào là biểu diễn của heap.

Cả A, B, C đều sai.

1.5 LO 5.1

Không gian địa chỉ bảng hash là [0...17], hàm hash $H(K) = K\%17$, sử dụng phương pháp thăm dò tuyến tính để giải quyết xung đột, giả sử bảng hash đã có các phần tử 26 , 25, 72, 38, 8, 18. Chèn thêm phần tử 59 vào bảng hash, vậy địa chỉ của phần tử này là:

11

1.6 LO 2.1

Dùng mảng V[1...m] để hiện thực 2 ngăn xếp có chung không gian lưu trữ, top[i] là đỉnh của ngăn xếp i (i = 1, 2), đáy của ngăn xếp 1 là V[1], đáy của ngăn xếp 2 là V[m]. Ngăn xếp sẽ đầy khi:

$top[1] + 1 = top[2]$

1.7 LO 1

Hãy cho biết thân vòng lặp sau thực thi bao nhiêu lần

```
int i = 0, s = 0, n = 100;
do {i = i + 1; s = s + 10 * i;}
while((i < n) && (s < n));
```

4

1.8 LO 2.1

Nếu một danh sách nào đó có thao tác sử dụng nhiều nhất là truy xuất phần tử bất kỳ, ngoài ra thao tác chèn và xóa được thực hiện ở phần tử cuối cùng của danh sách, thì cấu trúc dữ liệu nào sau đây tiết kiệm thời gian nhất.

Danh sách liên tục.

1.9 LO 1.2

Cho $f_1(n) = 100 + 2n$ và $f_2(n) = 1 + 2 + 3 + \ldots + n$, big-O của hai biểu thức này lần lượt là:

$O_1(n)$ **và** $O_2(n^2)$

1.10 LO 2.1

Dùng mảng A[0...5] để hiện thực hàng đợi vòng (circular queue), giả sử rear = 0 và front = 3. Giá trị của rear và front sau khi xóa 1 phần tử và thêm 2 phần tử mới vào hàng đợi.

2 và 4

1.11 LO 6.1

Phương pháp sắp xếp nào mà ở mỗi bước, giải thuật sẽ lựa chọn giá trị nhỏ nhất trong phần chưa được sắp xếp, sau đó chèn vào cuối phần đã được sắp xếp:

Lựa chọn

1.12 LO 6.1

Sử dụng phương pháp Quick Sort để sắp xếp dãy số (28, 16, 32, 12, 60, 2, 5, 72) theo thứ tự từ nhỏ đến lớn. Hãy cho biết kết quả sau khi tiến hành thao tác Partition lần đầu tiên (lấy phần tử đầu tiên làm pivot, hoán pivot với phần tử cuối trong partition đầu).

(2,16,5,12) 28 (60,32,72)

1.13 LO 5.2

Giả sử bảng hash có không gian địa chỉ là [0...9999]. Biết rằng khóa $K = 442205883$ có địa chỉ là 837, hãy cho biết hàm hash nào dưới đây được sử dụng.

Folding.

1.14 LO 6.1

Sắp xếp dãy số sau {15, 9, 7, 8, 20, -1, 4}, giả sử sau một bước chạy, ta nhận được kết quả là {4, 9, -1, 8, 20, 7, 15}. Hãy cho biết phương pháp sắp xếp được sử dụng là:

Shell.

1.15 LO 3.1

Cho biết nhận định nào dưới đây về cây B bậc m là KHÔNG đúng

Nút trong có ít nhất $m/2$ cây con (nếu m chẵn) hoặc $m/2+1$ cây con (nếu m lẻ).

2 Điền kết quả

2.1 LO 3.1

Cho cây nhị phân với 7 nút là A, B, C, D, E, F và G. Biết kết quả duyệt tiền thứ tự (preorder) là AFBCDEG, kết quả duyệt trung thứ tự (inorder) là CEDBGFA. Cho biết kết quả duyệt theo chiều rộng:

AFBCGDE

2.2 LO 3.1

Tính giá trị của biểu thức tiền tố sau: * * 7 6 - / 8 2 1

126

2.3 LO 1

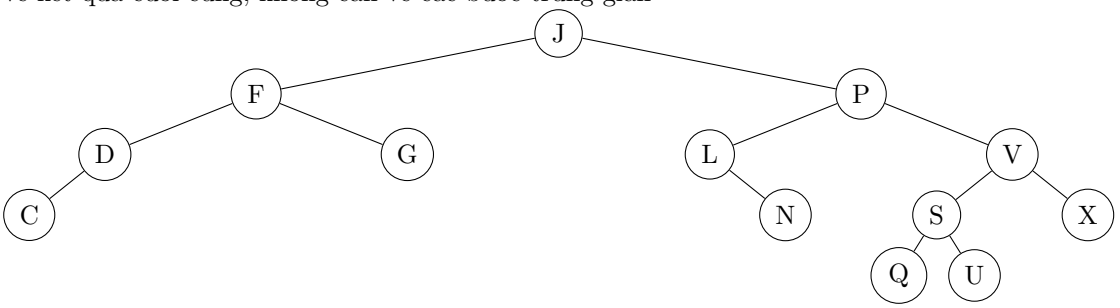
Cho biết sau khi đoạn chương trình sau thực thi xong, giá trị m bằng bao nhiêu:

```
int m = 0, n = 1000;
for(int i =1; i <= n ; i++)
for(int j =2*i ; j <= n ; j++)
m = m+1;
```

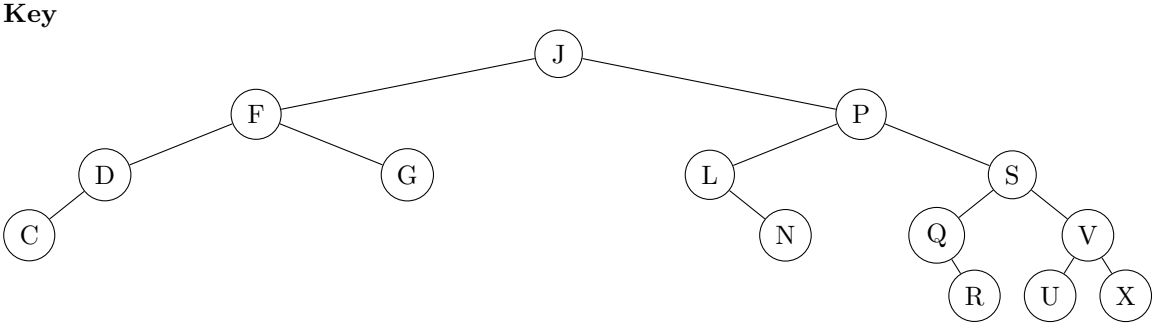
1000*1000/4 = 250000

2.4 LO 3.5

Cho cây AVL giống như hình vẽ. Hãy vẽ lại cây AVL này sau khi chèn giá trị R vào cây. Chỉ cần vẽ kết quả cuối cùng, không cần vẽ các bước trung gian

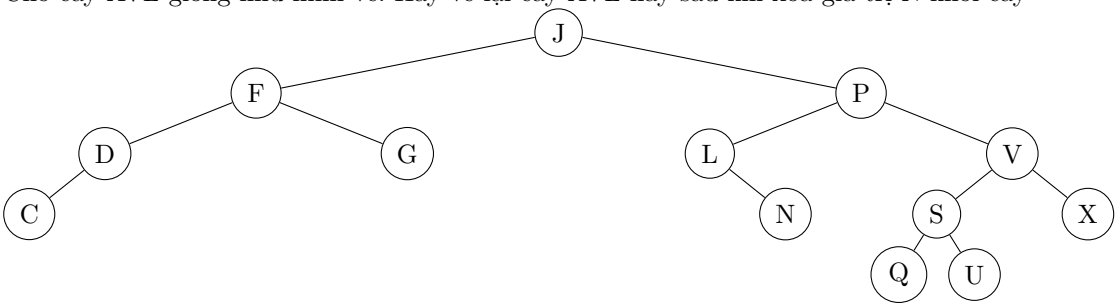


Key

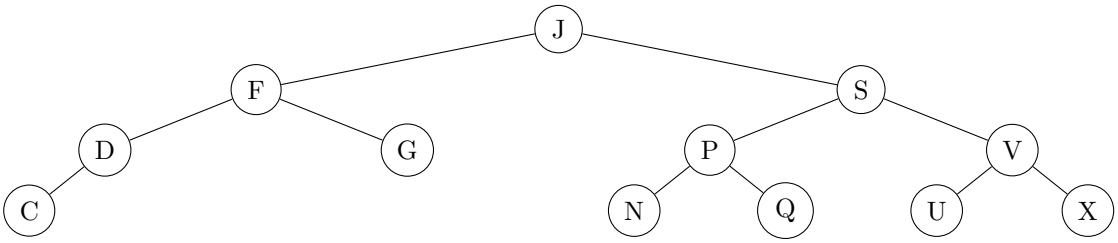


2.5 LO 3.5

Cho cây AVL giống như hình vẽ. Hãy vẽ lại cây AVL này sau khi xóa giá trị N khỏi cây



Key



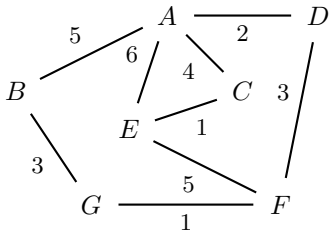
2.6 LO 7.4

Cho đồ thị vô hướng G = (V, E), trong đó V={a, b, c, d, e, f}, E={(a,b), (a,e), (a,c), (b,e), (c,f), (f,d), (e,d)}. Hãy cho biết kết quả duyệt đồ thị theo chiều sâu, biết rằng bắt đầu duyệt từ đỉnh A. Lưu ý, trong trường hợp nhiều đỉnh có thể được chọn, thì các đỉnh sẽ được đưa vào stack theo thứ tự trong bảng chữ cái.

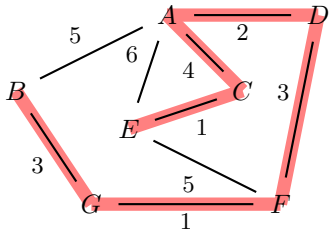
a e d f c b (lời giải đúng)
a b e d f c (lời giải chấp nhận, vì một số lớp tôi không nói nhiều về sử dụng stack)

2.7 LO 7.7

Cho đồ thị như hình vẽ dưới, sử dụng giải thuật Prim để tìm cây phủ tối thiểu. Giả sử bắt đầu từ đỉnh A, hãy vẽ cây phủ tối thiểu.

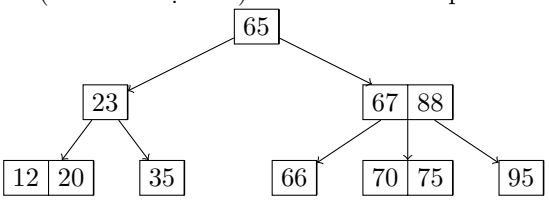


Key

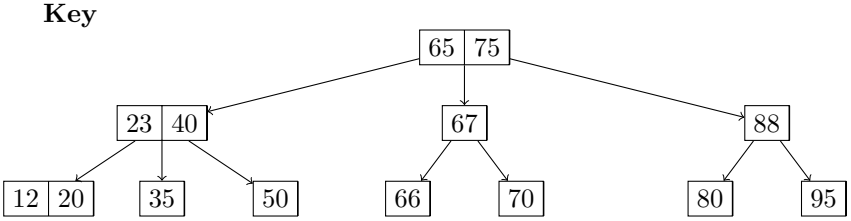


2.8 LO 3.5

Cho cây B-tree bậc 3 như hình vẽ sau. Hãy vẽ cây này sau khi đã chèn vào cây 3 giá trị 80, 40, 50 (theo thứ tự trên). Chỉ cần vẽ kết quả cuối cùng, không cần vẽ các bước trung gian.

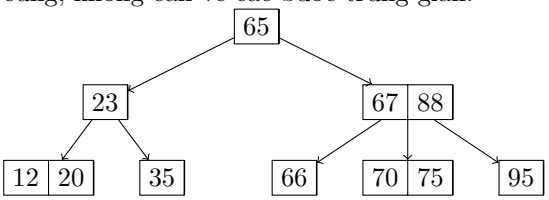


Key

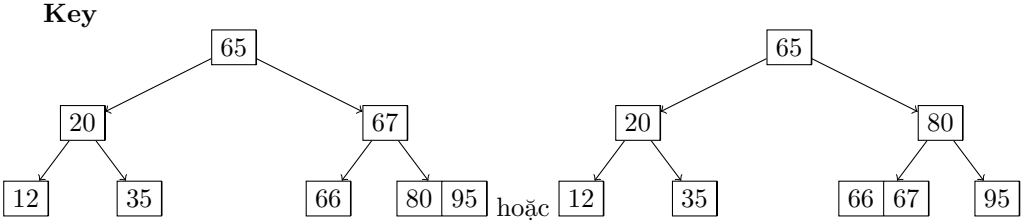


2.9 LO 3.5

Cho cây B-tree bậc 3 như hình vẽ sau. Hãy vẽ cây này sau khi lần lượt xóa khỏi cây 3 giá trị 23, 70, 75 (khi xoá thì thay thế bằng entry phải cùng của cây con bên trái). Chỉ cần vẽ kết quả cuối cùng, không cần vẽ các bước trung gian.



Key



2.10 LO 4.4

Giả sử ta biểu diễn min-heap dưới dạng dãy, hiện tại ta có min-heap (10, 21, 35, 32, 37, 56, 64, 48, 85, 63, 71, 92). Chèn thêm vào min-heap này phần tử có giá trị 50, sau đó xóa phần tử nhỏ nhất khỏi min-heap. Hãy cho biết min-heap này (dưới dạng dãy) sau khi thực hiện 2 thao tác trên:

21, 32, 35, 48, 37, 50, 64, 56, 85, 63, 71, 92

3 Tự luận

3.1 B+ tree:

3.1.1 LO 3.2

```
struct BpEntry<K, T>
    key       <type K>
    rightPtr  <pointer>// void pointer is the best choice
end struct BpEntry
struct BpNode<N, K, T>
    leftPtr       <pointer>
    nEntries      <integer>
    entries[N-1]  <array of BpEntries<K, T>>
end struct BpNode
class BpTree<degree, K, D>
    root      <pointer to BpNode<degree, K, D>>
    // methods declarations
    <bool> insert(key <K>, data <D>);
    <bool> remove(key <D>);
    <pointer to D> find(key <K>);
    traverse(op <function>);
end class BpTree
```

3.1.2 LO 3.3

Các phương thức cơ bản được mô tả như mã của BpTree bên trên.

3.2 Sắp xếp

3.2.1 LO 4.5, 8.4

```
template <class T>
void sortHM(T* pD, int N) {
    if (N < 2) return;
    if (N < 1000) sortMinHeap(pD, N);// sort using minHeap
    else {
        sortHM(pD, N/2);
        sortHM(pD + N/2, N - N/2);
        mergeDesc(pD, N/2,  // first array + size
                  pD + N/2, N - N/2, // second array + size
                  pD // output array
                );// merge two arrays in decreasing order
    }
}
```

```
template <class T>
void sortMinHeap(T* pD, int N) {
    buildMinHeap(pD, N);
    for (int i = N - 1; i > 0; i--) {
        swap(pD[0], pD[i]);
        minHeapDown(pD, i);
    }
}
```

3.2.2 LO 4.6

Có thể thu giảm số lần hoán vị trong heap sort nếu chỉ sắp xếp index thay vì xấp xếp toàn bộ mảng dữ liệu gốc. Tradeoff của thuật toán là không gian bộ nhớ.

3.3 Thiết kế hệ thống: vận dụng cấu trúc dữ liệu

3.3.1 LO 2.5, 3.7, 4.1

```
struct CustomerInfo {
    int                 age;
    CustomerStatus      status;// CustomerStatus is a structure
                               // defined before
    int                 arriveTime;
    Queue<CustomerReq>  requests;// CustomerReq is a struct
                               // containting request info
};
struct CustomerAVLData {
    CustomerInfo*       pData;
    operator < (CustomerAVLData &b) {
        return pData->arriveTime < b.pData->arriveTime;
    }
    // other comparison methods should be defined
};
template <class T>
struct PriorityNode {
    int         priotity;
    T*          pData;
    operator < (PriorityNode<T>& b) {
        return priotity < b.priotity;
    }
    // other comparision methods should be defined
};

class MySystem{
```

```
// Priority queue for customer checkin, use Max-heap
vector<PriorityNode<CustomerInfo>>  prioQueue;
// AVL-tree to store customer information
AVLTree<CustomerAVLData>    cusTree;
// Request queue
Queue<CustomerReq>          reqQueue;
// Queue to store arrive time of customers
Queue<int>                  arriveQueue;
```

3.3.2 LO 4.1, 4.4

```
// Checkin customer
void checkinCustomer(int age, CustomerStatus status, int arriveTime) {
    CustomerInfo* pC = new CustomerInfo;
    pC->age = age;
    pC->status = status;
    pC->arriveTime = arriveTime;
    PriorityNode<CustomerInfo>  cNode;
    cNode.priotity = getPriority(age, status, arriveTime);
    cNode.pData = pc;
    heapInsert<PriorityNode<CustomerInfo>>(prioQueue, cNode);
}
```

3.3.3 LO 3.7, 4.4

```
// pick customer from priority queue
bool pickCustomer() {
    if (prioQueue.size() < 1) return;
    CustomerAVLData     cData;
    cData.pData = prioQueue[0].pData;
    cusTree.insert(cData);
    heapRemove<PriorityNode<CustomerInfo>>(prioQueue);
}
```

3.3.4 LO 2.5, 3.7

```
// traverse through customer list
void traverse() {
    _traverse(cusTree.root);
    CustomerAVLData cData;
    CustomerInfo    cInfo;
    cData.pData = &cInfo;
    while (!arriveQueue.isEmpty()) {
        cInfo.arriveTime = arriveQueue.first();
        cusTree.remove(cData);
```

```
        }
    }
    void _traverse(AVLNode<CustomerAVLData>* pR) {
        if (pR == NULL) return;
        _traverse(pR->pLeft);
        if (pR->pData->request.size() < 1) {
            // to remove this customer later
            arriveQueue.enqueue(pR->pData->arriveTime);
        }
        else {
            reqQueue.enqueue(pR->pData->requests.first());
            pR->pData->requests.dequeue();
            if (pR->pData->request.size() < 1) {
                // to remove this customer later
                arriveQueue.enqueue(pR->pData->arriveTime);
            }
        }
        _traverse(pR->pRight);
    }
```

3.4 LO 3.6: BST traverse

```
template <class T>
void traverseInsert(BSTNode<T>* pR, List<T>& outList) {
    if (pR == NULL) return;
    traverseInsert(pR->pRight, outList);
    outList.push_back(pR->data);
    traverseInsert(pR->pLeft, outList);
}
```