|  |  |
| --- | --- |
| **TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN** | **ĐỀ THI CUỐI HỌC KỲ II (2023-2024)** |
| **KHOA KHOA HỌC MÁY TÍNH** | **MÔN: Cấu trúc dữ liệu & Giải thuật**  **MÃ LỚP: IT003** |
| **Đề 1** | *Thời gian: 90 phút* |

(***Sinh viên không được sử dụng tài liệu***)

|  |  |
| --- | --- |
| **HỌ VÀ TÊN SV**: ……………………………………  **MSSV**: ……………………………………………….  **STT**: ………………………………………………….  **PHÒNG THI:…..**…………………………………… | **CÁN BỘ COI THI** |

***Đề 1 và Đề 2 có đáp án giống nhau, chỉ có thứ tự câu hỏi trong phần CÂU HỎI ĐIỀN KHUYẾT là khác nhau. Nên xin quý thầy, cô xem đáp án ở đây và chấm trên hai đề giúp.***

**CÂU HỎI ĐIỀN KHUYẾT**

1. (0.5 điểm)

Xét độ phức tạp theo BigO, thuật toán tìm nhị phân tốt hơn thuật toán tìm tuyến tính, vì sao? (0.25đ)

Đáp án: Vì thuật toán tìm nhị phân có độ phức tạp O(logn), thấp hơn độ phức tạp của thuật toán tìm tuyến tính O(n).

Chỉ cần sinh viên ghi độ phức tạp của thuật toán tìm nhị phân là O(logn) là đủ.

Thuật toán tìm nhị phân với thứ tự tăng dần sẽ có kết quả tìm khóa 4 trong mảng gồm các khóa 1, 2, 3, 4, 5, 6, 7, 8, 9 sau bao nhiêu lần **so sánh giá trị khóa**? (0.25đ)

Đáp án: 4 **hoặc 7**

**Do một số bạn hiểu một lần so sánh giá trị khóa là một câu lệnh so sánh giá trị khóa. Trong khi một lần so sánh giá trị khóa tương ứng với một giá trị m, mình sẽ kiẻm tra khóa có bằng, lớn hơn hay không, Tuy nhiên, mình vẫn chấp nhận cả 2 đáp án vì cả hai đều cho thấy sinh viên biết thuật toán.**

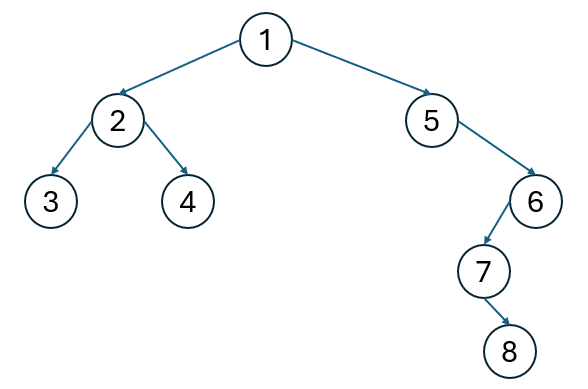
1. (0.5 điểm)

Nếu dùng thuật toán Insertion Sort (Chèn trực tiếp) để sắp xếp một dãy số tăng dần, thì khi hoàn thành việc sắp xếp mảng A={3,2,4,5,1,5}, số lần dịch chuyển phần tử sang phải một vị trí là bao nhiêu?

Đáp án: 5

1. (0.5 điểm)

Cho cây nhị phân T1 như Hình 1.



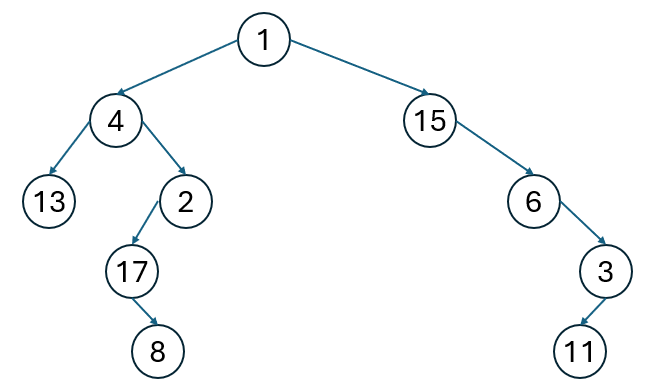
*Hình 1. Cây nhị phân T1*

Hỏi khóa 7 xuất hiện ở vị trí thứ mấy (vị trí đầu tiên là 0) trong kết quả duyệt LNR?

Đáp án: 5

1. (0.5 điểm)

Cho cây nhị phân T2 như Hình 2.



*Hình 2. Cây nhị phân T2*

Hỏi mức nào trên cây T2 có tổng giá trị khóa cao nhất?

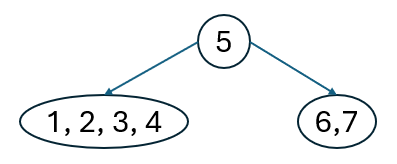
Đáp án: 2

***Bổ sung: Có một số bạn ghi đáp án là 3, có ghi thêm giá trị là 21. Theo slide bài giảng chung thì mức tính từ 0. Nhưng không rõ có trường hợp nào sinh viên được học là mức tính từ 1 hay không.***

***Vì vậy, nếu sinh viên ghi là 3 thì chấm 0.25 điểm***

1. (0.5 điểm)

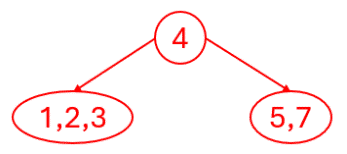
Cho B-Tree bậc 5, T3, như Hình 3.



*Hình 3. B-Tree bậc 5 T3*

Hãy vẽ cây T3 sau khi xóa khóa 6 (0.25đ)? Và cho biết tên thao tác đã điều chỉnh các nút trên T3 (0.25đ)?

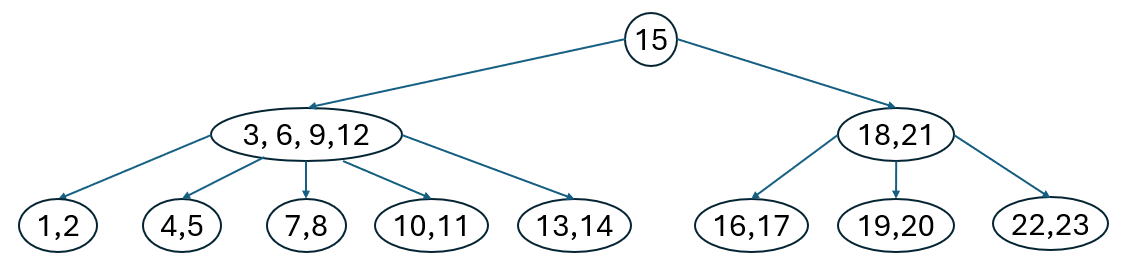
Đáp án



Tên thao tác: nhường khóa, hoặc underflow.

1. (0.5 điểm)

Cho B-Tree bậc 5, T4, như Hình 4. Biết thao tác tìm khóa trên một nút sử dụng thuật toán tìm tuyến tính.



*Hình 4. B-Tree bậc 5 T4*

Hỏi số lần **so sánh giá trị khóa** khi có kết quả tìm khóa 14 trên T4?

Đáp án: 7 **hoặc 13**

**Lý do tương tự như số bước so sánh khóa ở câu tìm tuyến tính.**

***Đề bài bên dưới được sử dụng cho Câu 7, Câu 8 và Câu 9***.

Cho bảng băm địa chỉ mở HT1, có kích thước M=11, hàm băm f và hàm băm lại f’ lần lượt là:

Trong đó, phép toán % là phép toán lấy phần dư của phép chia nguyên.

*Bảng băm HT1*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chỉ số | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Khóa | 11 | 1 |  |  | 15 | 5 |  |  |  | 9 |  |

1. (0.5 điểm)

Hỏi số lần **so sánh giá trị khóa** khi có kết quả tìm khóa 4 trên HT1?

Đáp án: 2 / **3**

***Bổ sung: Một số bạn ghi là 3. Theo thuật toán, lần 3 thoát là do ô chỉ số 8 là ô rỗng chứ không so sánh giá trị khóa. Tuy nhiên trên code kiểm tra là, chẳng hạn***

***while (ht[i] != EMPTY && i < m)***

***Được các bạn hiểu là so sánh giá trị khóa và được tính là lần thứ 3***

1. (0.5 điểm)

Hỏi hệ số tải của HT1 (lấy 2 số lẻ sau phần thập phân)?

Đáp án: 0.45

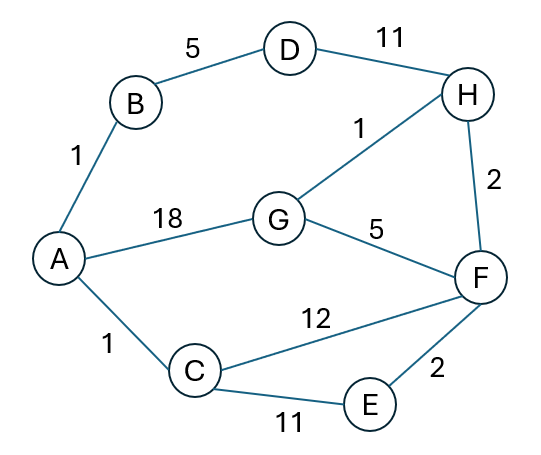
1. (0.5 điểm)

Hỏi vị trí khi thêm khóa 22?

Đáp án: 3

1. (0.5 điểm)

Cho đồ thị vô hướng G như Hình 5.



*Hình 5. Đồ thị vô hướng G*

Hỏi độ dài đường đi ngắn nhất từ A đến các đỉnh còn lại?

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | B | C | D | E | F | G | H |
| Độ dài đường đi ngắn nhất từ A | 1 | 1 | 6 | 12 | 13 | 16 | 15 |

Đáp án: Nếu sai 1 trong các độ dài đường đi trên bảng thì được 0.25, sai nhiều hơn 1 giá trị không tính điểm.

**CÂU HỎI TỰ LUẬN**

*Giả sử các đoạn chương trình trong phần này đều có các chỉ thị #include, đảm bảo các kiểu dữ liệu và các hàm được C++ hỗ trợ đều sẵn sàng để sử dụng.*

1. (2 điểm)

Cho các bản đồ thể hiện không quá 10000 giao lộ (ngã ba, ngã tư, ..). Mỗi giao lộ được đánh số từ 0 đến 9999. Các bản đồ này có thể biểu diễn theo đồ thị đơn, vô hướng và được biểu diễn trên máy tính bằng ma trận, với kiểu dữ liệu ***Bando*** như sau:

struct Bando {

int N;

bool Duong[10000][10000];

};

Trong đó, ***N*** là số giao lộ; ***Duong***[***i***][***j***]=true cho biết giao lộ ***i*** có đường đi trực tiếp đến giao lộ ***j***; ***Duong***[***i***][***j***]=false cho biết giao lộ ***i*** không có đường đi trực tiếp đến giao lộ ***j***.

1. Cài đặt hàm ***DemNgaNam*** để đếm số ngã năm trên bản đồ ***m*** (1đ):

Đáp án: đếm số dòng có đúng 5 giá trị true hoặc đếm số cột có đúng 5 giá trị true.

**Nếu đúng đoạn chương trình xác định số đỉnh kề của một đỉnh i (đoạn code có nền vàng) thì được 0.5đ.**

Code tham khảo:

int DemNgaNam(Bando m) {

int kq = 0;

for (int i = 0; i < m.N; i++) {

int ndinh = 0;

for (int j = 0; j < m.N; j++)

if (m.Duong[i][j]) ndinh ++; // đổi thành cột cũng được

if (ndinh == 5) kq++;

}

return kq;

}

1. Cài đặt hàm ***CoDuongdi*** trả về ***1*** nếu bản đồ ***m*** có đường đi từ giao lộ ***s*** đến giao lộ ***e***, trả về ***0*** trong các trường hợp còn lại. Yêu cầu phải dùng queue thay cho hàm đệ quy (1đ):

Code tham khảo:

**Nếu đúng đoạn đưa các đỉnh kề chưa duyệt vào queue (có nền vàng) được 0.5đ**

int CoDuongdi(Bando m, int s, int e) {

queue<int> q;

vector<bool> closed(m.N, false);

q.push(s);

while (!q.empty()) {

int c = q.front(); q.pop();

closed[c] = true;

if (c == e) return 1;

for (int j = 0; j < m.N; j++) {

if (closed[j]) continue;

if (m.Duong[c][j]) q.push(j);

}

}

return 0;

}

1. (1 điểm)

Cho kiểu dữ liệu ***DblStack***, là một stack chứa các giá trị double, và một hàm ***CreateDblItem*** được khai báo như bên dưới.

struct DblStack {

double data;

DblStack \*next;

};

DblStack \* CreateDblItem(double x) {

DblStack r = new DblStack;

if (r == NULL) exit(1);

r->data = x;

r->next = NULL;

return r;

};

Hãy cài đặt hàm ***push*** (0.5đ) và hàm ***pop*** (0.5đ) ở bên dưới, lần lượt để đưa một số double vào stack và xóa bỏ phần tử ở đỉnh stack.

Đáp án tham khảo: Thao tác push là chèn vào đầu DSLK đơn, thao tác pop là xóa đầu DSLK đơn

Hàm push đúng đoạn code tao phần tử (có nền vàng) được 0.25đ. Đúng toàn bộ được 0.5đ

void push(DblStack &s, double x) {

DblStack \* p = CreateDblItem(x);

if (s != NULL)

p->next = s;

s = p;

}

***Bổ sung: một số bạn thêm vào cuối thay vì thêm vào đầu. Trường hợp này mình sẽ chấm 0.25 điểm***

Hàm pop đúng đoạn code kiểm tra stack rỗng (có nền vàng) được 0.25đ. Đúng toàn bộ được 0.5đ

void pop(DblStack &s) {

if (s == NULL) return;

DblStack \*p = s;

s = s->next;

delete p;

}

***Bổ sung: một số bạn xóa ở cuối thay vì xóa ở đầu. Trường hợp này mình sẽ chấm 0.25 điểm***

1. (1 điểm)

Cho kiểu dữ liệu ***TREE***, để biểu diễn cây nhị phân tìm kiếm, được khai báo như bên dưới. Hãy cài đặt hàm ***ChonKhoa*** để trả về một danh sách các khóa ***k*** trên cây nhị phân tìm kiếm sao cho ***a*** < ***k*** < ***b***, với ***a***, ***b*** là các số cho trước. Yêu cầu dùng queue thay cho hàm đệ quy.

struct TNode {

int key;

TNode \*left, \*right;

};

typedef TNode \*TREE;

Đáp án tham khảo: **Nếu đúng được khung thuật toán BFS** (có nền vàng và **có push cây con vào queue**) được 0.5đ.

vector<int> ChonKhoa(TREE t, int a, int b) {

queue<TREE> q;

vector<int> keys;

q.push(t);

while (!q.empty()) {

TREE c = q.front(); q.pop();

if (c->key > a && c->key < b) {

keys.push\_back(c->key);

q.push(c->left);

q.push(c->right);

}

else {

if (c->key < a) {

q.push(c->right);

}

else {

q.push(c->left);

}

}

}

return keys;

}

1. (1 điểm)

Quỳnh được giao công việc cài đặt một hàm nhận một danh sách số nguyên và trả về một tập hợp số nguyên (các phần tử không trùng nhau). Các phần tử trong tập hợp được sắp thứ tự tăng dần.

Ví dụ: Input là 1, 5, 2, 7, 4, 1, 3, 2, 3, 5 thì output là 1, 2, 3, 4, 5, 7.

Quỳnh sử dụng hàm ***sort***, có **độ phức tạp O(nlogn)**, và cài đặt hàm ***ListToSet*** như sau.

vector<int> ListToSet(vector<int> A) {

vector<int> r;

long long unsigned int i, j;

for (i = 0; i < A.size(); i++) {

for (j = 0; j < i; j++) {

if (A[j] == A[i]) break;

}

if (j >= i) r.push\_back(A[i]);

}

sort(r.begin(), r.end());

return r;

}

Mentor của Quỳnh nói rằng, cách này chưa tốt, phải sắp xếp trước rồi xử lý. Mentor đưa cho Quỳnh hàm ***List2Set*** như bên dưới, và yêu cầu Quỳnh hoàn thành nó sao cho thời gian thực hiện tốt hơn hàm ***ListToSet***.

Bạn cũng phải hoàn thành hàm ***List2Set*** này.

Đáp án tham khảo: Dùng tìm kiếm nhị phân trên vector r để kiểm tra trùng lặp, hoặc kiểm tra hai phần tử liên tiếp khác nhau, hoặc kiểm tra phần tử cuối của r khác với phần tử đang xét, sẽ được 1đ.

Nếu tìm tuyến tính từ phần tử đầu tiên trên vector r để kiểm tra trùng lặp, được 0.5đ.

**1. Các đoạn code 1 điểm có thể tham khảo như sau:**

Code tham khảo theo cách Tìm nhị phân: (Mặc dù phức tạp hơn 2 đoạn code còn lại nhưng đoạn code này không làm tăng độ phức tạp của đoạn chương trình List2Set)

vector<int> List2Set(vector<int> A) {

vector<int> r;

sort(A.begin(), A.end());

int i, j;

for (i = 0; i < A.size(); i++) {

int lf=0, rt=r.size()-1, m;

while (lf <= rt) {

m = (lf+rt) /2;

if (r[m] == A[i]) break;

if (r[m] < A[i]) lf = m+1;

else rt = m-1;

}

if (lf > rt) {

r.push\_back(A[i]);

}

}

return r;

}

Code tham khảo theo cách kiểm tra 2 phần tử liên tiếp: (Mặc dù đơn giản hơn đoạn code tìm nhị phân nhưng đoạn code này không làm giảm độ phức tạp của đoạn chương trình List2Set)

vector<int> List2Set(vector<int> A) {

vector<int> r;

sort(A.begin(), A.end());

r.push\_back(A[0]);

for (int i = 1; i < A.size(); i++) {

if (A[i] != A[i-1])

r.push\_back(A[i]);

}

return r;

}

Code tham khảo theo cách kiểm tra phần tử cuối của r: (Mặc dù đơn giản hơn đoạn code tìm nhị phân nhưng đoạn code này không làm giảm độ phức tạp của đoạn chương trình List2Set)

vector<int> List2Set(vector<int> A) {

vector<int> r;

sort(A.begin(), A.end());

r.push\_back(A[0]);

for (int i = 1; i < A.size(); i++) {

if (A[i] != r[r.size()-1])

r.push\_back(A[i]);

}

return r;

}

***Bổ sung: Một số bạn sử dụng bảng băm thay vì tìm nhị phân cũng được chấm 1 điểm***

**2. Đoạn code 0.5 điểm có thể tham khảo như sau:** (Mặc dù có giảm chi phí tính toán so với hàm ListToSet nhưng độ phức tạp vẫn là O(n2) không giảm so với hàm ListToSet)

vector<int> List2Set(vector<int> A) {

vector<int> r;

sort(A.begin(), A.end());

for (int i = 1; i < A.size(); i++) {

bool notfound = true;

for (int j = 0; j < r.size(); j++) {}\

if (r[j] == A[i]) {

found = false;

break;

}

}

if (notfound) r.push\_back(A[i]);

}

return r;

}

***Bổ sung: Một số bạn sử dụng kiểu set cũng được chấm 0.5 điểm. Vì đề bài đã yêu cầu viết lại hàm tạo tập hợp từ một danh sách. Nên việc dùng hàm cài đặt sẵn không hợp lý.***