TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN **ĐÁP ÁN** **ĐỀ THI THỬ CUỐI KỲ**

**BHT KHOA HTTT x BHT KHOA KH&KTTT** **HỌC KỲ 2 – NĂM HỌC 2021 – 2022**

Môn thi: CTDL&GT

Thời gian làm bài: 90 phút

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

**Câu 1:**

a.

- Ý tưởng của thuật toán sắp xếp chèn:

+Input: Một chuỗi chưa được sắp xếp

+Output: Một chuỗi đã được sắp xếp giảm dần theo thuật toán Insertion sort

Thuật toán sắp xếp chèn thực hiện sắp xếp dãy số theo cách duyệt từng phần tử và chèn từng phần tử đó vào đúng vị trí trong mảng con đã sắp xếp sao cho dãy số trong mảng đó vẫn đảm bảo tính chất của một dãy số giảm dần

1. Khởi tạo mảng với dãy con đã sắp xếp có k = 1 phần tử

2. Duyệt từng phần tử từ phần tử thứ 2, tại mỗi lần duyệt phần tử ở chỉ số I thì đặt phần tử đó vào một vị trí nào đó trong đoạn từ [0..i] sao cho dãy số từ [0..1] vẫn đảm bảo tính chất dãy số tăng dần. Sau mỗi lần duyệt, số phần tử đã được sắp xếp K trong mảng tăng thêm 1 phần tử

3. Lặp cho tới khi duyệt hết tất cả các phần tử trong mảng

- Độ phức tạp:

+ Xấu nhất : O(n2)

+ Tốt nhất : O(n)

+ Trung bình: O(n2)

b. Cho dãy số 8, 4, 3, 0, 1, 2, 5, 1. Trình bày các bước áp dụng thuật toán sắp xếp chèn (insertion sort) để sắp xếp mảng giảm dần

Input : 8,4,3,0,1,2,5,1

Output: 8,5,4,3,2,1,1,0

1. Chọn i = 1, j = i -1,so sánh a[i] với a[j], ta thấy a[i] < a[j](Thõa mãn), i++ và tiếp tục vòng lặp

2. i = 2, j = i-1, so sánh a[i] với a[j], ta thấy a[i] < a[j] (Thõa mãn), i++ và tiếp tục vòng lặp

3. I = 3, j = i -1, so sánh a[i] với a[j], ta thấy a[i] < a[j] (Thõa mãn), i++ và tiếp túc vòng lặp

4. I = 4, j = i- 1, so sánh a[i] với a[j], ta thấy a[i] > a[j]( Không thõa mãn), ta gán phần tử tại a[j+1] = a[j] (a[4] = a[3]), j = j - 1

Tiếp tục so sánh a[j] với a[i], ta thấy a[i] < a[j](Thõa mãn), ta gán a[j+1] = a[i], i++ và tiếp tục vòng lặp

Mảng hiện tại sẽ trở thành: 8, 4, 3, 1, 0, 2, 5, 1

5. I = 5, j = I – 1, so sánh a[i] với a[j], ta thấy a[i] > a[j](Không thõa mãn), ta gán phần tử tại a[j+1] = a[j](a[5] = a[4]), j = j – 1

Tiếp tục so sánh a[j] với a[i], ta thấy a[i] > a[j] (Không thõa mãn), ta gán phần tử tại a[j+1] = a[j] (a[4] = a[3]), j = j -1

Tiếp tục so sánh a[j] với a[j], ta thấy a[i] < a[j] (Thỏa mãn), ta gán a[j+1] = a[i], i++ và tiếp tục vòng lặp

Mảng hiện tại sẽ trở thành: 8, 4, 3, 2, 1, 0, 5, 1

6. I = 6, j = i- 1, so sánh a[i] với a[j], ta thấy a[i] > a[j](Không thỏa mãn), ta gán phần tử tại a[j+1] = a[j] (a[6] = a[5]),j = j – 1

Tiếp tục so sánh a[j] với a[i], ta thấy a[i] > a[j](Không thỏa mãn), ta gán phần tử tại a[j+1]= a[j] (a[5] = a[4]) , j = j -1

Tiếp tục so sánh a[j] với a[i], ta thấy a[i] > a[j](Không thõa mãn), ta gán phần tử tại a[j+1]= a[j] (a[4] = a[3]), j = j -1

Tiếp tục so sánh a[j] với a[i], ta thấy a[i] > a[j](Không thõa mãn), ta gán phần tử tại a[j+1]= a[j] (a[3] = a[2]) ,j = j -1

Tiếp tục so sánh a[j] với a[i], ta thấy a[i] > a[j](Không thõa mãn), ta gán phần tử tại a[j+1]= a[j] (a[2] = a[1]), j = j -1

Tiếp tục so sánh a[j] với a[i], ta thấy a[i] < a[j](Thõa mãn), ta gán a[j+1] = a[i], i++ và tiếp tục vòng lặp

Mảng sẽ trở thành: 8, 5, 4, 3, 2, 1, 0, 1

7. I = 7, j = I – 1, so sánh a[i] với a[j], ta thấy a[i] > a[j](Không thõa mãn), ta gán phần tử tại a[j+1] = a[j](a[7] = a[6])

Tiếp tục so sánh a[j] với a[i], ta thấy a[i] = a[j](Thõa mãn), Ta gán a[j+1] = a[i], i++ và tiếp tục vòng lặp

8. Kết thúc vòng lặp

**Câu 2:**

a)

b) NLR: 10 7 6 4 9 12 11

LRN: 4 6 9 7 11 12 10

c)

void findvector(Node\* root, vector<int> &vt)

{

if (!root) return;

findvector(root->left,vt);

vt.push\_back(root->data);

findvector(root->right,vt);

}

bool checkBST(Node\* root) {

vector<int> vt;

findvector(root,vt);

for (int i = 0; i < vt.size()-1; i++) if (vt[i] >= vt[i+1]) return 0;

return 1;

}

**Câu 3:**

Bảng băm là một CTDL phục vụ cho việc tìm kiếm. và biểu diễn dữ liệu dưới dạng các cặp khóa và giá trị. Mỗi khóa được ánh xạ tới một giá trị trong bảng băm. Các khóa được sử dụng để lập chỉ số cho các giá trị hay dữ liệu.

Ý tưởng bảng băm:

* Dùng mảng lưu trữ các phần tử (bảng)
* Mỗi phần tử sẽ được lưu tại một ô duy nhất trong bảng căn cứ vào giá trị khóa của nó,
* Khi tìm kiếm thì căn cứ vào khóa cần tìm ta tìm đến ô tương ứng
* Nếu ô đó có chứa phần tử thì tìm thấy, ngược lại là không tìm thấy

Bảng băm có tốc độ tìm kiếm nhanh hơn so với tìm kiếm tuyến tính và tìm kiếm nhị phân.

Ví dụ: Với một mảng {1,4,6,7,9} để tìm kiếm một giá trị có thuộc mảng này không. Độ phức tạp trung bình cho việc tìm kiếm một phần tử trong mảng của từng thuật toán:

* Tìm kiếm tuyến tính: O(n).
* Tìm kiếm nhị phân: O(log(n)).
* Bảng băm: O(1).

1. .

| Key | Giá trị lưu | Xung đột |
| --- | --- | --- |
| 10 | 76 |  |
| 9 | 28 | 28%11 = 7 xung đột |
| 8 | 18 | 18%11 = 7 xung đột |
| 7 | 40 |  |
| 6 | 59 | 59%11 = 4 xung đột |
| 5 | 93 |  |
| 4 | 12 | 12%11 = 1 xung đột |
| 3 | 47 |  |
| 2 | 24 |  |
| 1 | 55 | 55%11 = 0 xung đột |
| 0 | 10 | 10%11 = 10 xung đột |

**Câu 4:**

1. Split node: 5,15,37,30,9,19,18
2. Diagram

   Description automatically generated with medium confidence
3. Nhập vào x bắt đầu truyền x vào hàm để duyệt hàm FindB-Tree

* So sánh x với node nếu x bé hơn giá trị bên trái cùng của nốt thì ta sẽ duyệt bên left của node. Nếu x lớn hơn giá trị bên phải cùng của nốt thì ta sẽ duyệt bên right của node. Nếu x có giá trị nằm trong khoảng của giá trị bên trái cùng của node và bên phải cùng của node thì ta sẽ duyệt các nhánh ở giữa(1)
* Và ta tiếp tục việc lặp của (1) cho đến khi tìm thấy phần tử == x lúc này ta sẽ trả về giá trị của x hoặc pLeft và pRight trỏ đến NULL thì ta sẽ xuất ra màn hình “Không tồn tại x trong B-Tree"

**Câu 5:**

Code:

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

vector<int> bfsOfGraph(int V, vector<int> a[])

{

    vector<bool> visit(V,false);

    vector<int> res;

    queue<int> trace;

    visit[0]=true;

    trace.push(0);

    while(!trace.empty())

    {

        int u = trace.front();

        trace.pop();

        res.push\_back(u);

        for(int v : a[u])

            if(visit[v]==false)

            {

                trace.push(v);

                visit[v]=true;

            }

    }

    return res;

}

int main() {

    int V, E;

    cin >> V >> E;

    vector<int> a[V];

    for (int i = 0; i < E; i++) {

        int u, v;

        cin >> u >> v;

        a[u].push\_back(v);

    }

    vector<int> ans = bfsOfGraph(V, a);

    for (int i = 0; i < ans.size(); i++) cout << ans[i] << " ";

    return 0;

}