TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

---------- ℘ 🕮 ℘ ----------

BÀI TẬP LỚN

MÔN: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Giảng viên hướng dẫn : TS. Hoàng Văn Thông

Sinh viên thực hiện: Trịnh Mạnh Quang

Lớp: CNTT4 – K63

Hà Nội, tháng 11 năm 2023

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

---------- ℘ 🕮 ℘ ----------

BÀI TẬP LỚN

MÔN: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Giảng viên hướng dẫn : TS. Hoàng Văn Thông

Sinh viên thực hiện: Trịnh Mạnh Quang

Lớp: CNTT4 – K63

Hà Nội, tháng 11 năm 2023

Mục lục

LỜI CẢM ƠN

Lời đầu tiên em xin gửi lời chào trân trọng và lòng biết ơn sâu sắc đến thầy Hoàng Văn Thông về sự hướng dẫn và cung cấp kiến thức quý báu trong môn học Cấu Trúc Dữ Liệu và Giải Thuật. Bài báo cáo này là kết quả của sự nỗ lực không ngừng nghỉ của em trong quá trình thực hiện bài tập lớn, và em xin được trình bày nó với lòng tôn trọng đối với sự giúp đỡ và những tri thức mà thầy đã cung cấp cho chúng em.

Bài tập lớn này đã giúp em hiểu rõ hơn về quy trình thiết kế cấu trúc dữ liệu và giải thuật, cùng với khả năng áp dụng kiến thức vào thực tế. Nó cũng giúp em phát triển kỹ năng, tư duy logic và khả năng giải quyết vấn đề.

Một lần nữa em xin gửi lời cảm ơn đặc biệt đến thầy vì sự hỗ trợ, sự hướng dẫn và những lời khuyên quý báu trong suốt thời gian qua. Không có được những sự chỉ dẫn của thầy, em không thể hoàn thành bài tập này một cách hiệu quả.

Bài báo cáo này sẽ tập trung trình bày về nhiệm vụ của em, quá trình thiết kế và triển khai cấu trúc dữ liệu và giải thuật, cũng như kết quả đạt được. Em rất mong nhận được sự đánh giá và phản hồi từ thầy Hoàng Văn Thông để cải thiện kiến thức và kỹ năng của bản thân mình.

Bài 1: (Bài số 23 trong danh sách bài tập)

1.1. Đề bài:

Tạo một cây từ điển để giải bài toán xây dựng danh bạ sau:

http://laptrinhonline.club/problem/Tèodanhba

Tèo muốn xây dựng danh bạ liên lạc để lưu vào trong máy tính. Trong quá trình xây dựng danh bạ Tèo để ý và thấy rằng rất nhiều từ nọ là tiền tố của từ kia. Nếu từ x được ghép bởi hai từ y và z tức là x = yz thì y được gọi là tiền tố và z gọi là hậu tố của x (Ví dụ từ tichpx có các tiền tố t, ti, tic, tich, tichp, tichpx). Trong khi cập nhật danh bạ với mỗi từ Tèo có 1 trong hai thao tác

- add là bổ sung một liên lạc vào danh bạ
- find là tìm xem một từ là tiền tố của bao nhiêu liên lạc đã có

Bạn hãy lập trình để xác định giúp Tèo nhé

Input:

- Dòng đầu tiên chứa số nguyên dương n là số thao tác (1<= n <= 105)
- n dòng tiếp theo mỗi dòng thuộc một trong hai thao tác hoặc là add hoặc là find một liên lạc là một xâu gồm toàn chữ thường tiếng anh có độ dài không quá 30 kí tự

Output:

Với mỗi thao tác find xuất ra màn hình số liên lạc đã có trong danh bạ có tiền tố muốn tìm

Ví dụ

Input

6

add daihocgiaothongvantai

add daihocthuyloi

find daihoc

add daihocxaydung

find hocvien

find dai

Output

2

0

3

1.2. Phân tích bài toán

- Yêu cầu của bài toán là tạo một cây từ điển để giải bài toán xây dựng danh bạ. Ý tưởng của báo cáo là tạo một cây tiền tố Trie để giải quyết bài toán.
- Các lớp cần xác định là:
    - Lớp cho các node trong cây tiền tố được khai báo dưới tên "TrieNode"
        - Các thuộc tính gồm:
            - TrieNode *child[26]: 26 con trỏ tương ứng với 26 kí tự tiếng anh thường trong bảng mã ASCII
            - Int exist: lưu số xâu là xâu được thể hiện bởi đỉnh u (đỉnh u là đỉnh ta xét đến)
            - Int count: lưu số xâu có tiền tố được thể hiện bởi đỉnh u.
        - Các phương thức cơ bản gồm:
            - TrieNode(): Phương thức khởi tạo của trienode sau khi được khai báo
            - ~TrieNode(): Phương thức hủy dữ liệu con trỏ của TrieNode
    - Lớp Trie là lớp chứa logic và thao tác trên cấu trúc Trie
        - Các thuộc tính bao gồm:
            - Int cur: lưu số node hiện tại có trong cây Trie
            - TrieNode* root: tạo nút gốc cho Trie
        - Các phương thức cơ bản gồm:
            - Trie() : cur(0) : tạo một nút mới làm nút gốc của cây Trie
            - ~Tride(): Phương thức hủy dữ liệu con trỏ của TrieNode

        - Void addString: Phương thức kích hoạt chức năng thêm chuỗi vào cây Trie có gốc root
        - Int getCount: Trả về số xâu có tiền tố giống với xâu mình tìm
        - Void run: Phương thức để chạy chương trình

- Cách thức hoạt động:
    - Lớp TrieNode:
        - TrieNode() : Khởi tạo một nút của Trie. Mỗi nút có một mảng con trỏ child có độ dài 26 và biến exist và count được khởi tạo với giá trị 0
    - Lớp Trie:
        - Void addString(string s):
            - B1: Tạo một con trỏ p và khởi tạo nó trỏ đi đến nút gốc root của cây Trie. Con trỏ p sẽ di chuyển theo đường đi của xâu s
            - B2: Sử dụng vòng lặp for duyệt qua từng kí tự của xâu s. Tại mỗi bước tính toán giá trị tmp bằng cách chuyển đổi ký tự c sang một chỉ số trong khoảng từ 0 đến 25 tương ứng với các ký tự ‘a’ đến ‘z’
            - B3: Kiểm tra xem nút con tương ứng với ký tự ‘c’ có tồn tại hay không (tức p->child[temp] == NULL). Nếu chưa tồn tại, tạo một nút con mới và cập nhật con trỏ p để trỏ đến nút con mới được tạo
            - B4: Tăng giá trị “count” tại nút p lên 1. “count” đại diện cho số lượng từ đi qua nút đó
            - B5: Cập nhật con trỏ p để trỏ đến nút con tương ứng với kí tự ‘c’. Tiếp tục vòng lặp để di chuyển con trỏ đến nút cuối cùng của xâu s
            - B6: Tăng giá trị “exist” tại nút cuối cùng lên 1. “exist” đại diện cho số lượng từ kết thúc tại nút đó
        - Void getCount(string s):

- B1: Tạo một con trỏ p và khởi tạo nó trỏ đến nút gốc root của cây Trie. Con trỏ p sẽ di chuyển theo đường đi của xâu s
- B2: Sử dụng vòng lặp for duyệt qua từng ký tự của xâu s. Tại mỗi bước, tính toán giá trị temp bằng cách chuyển đổi ký tự c sang một chỉ số trong khoảng từ 0 đến 25, tương tự với các kí tự ‘a’ đến ‘z’
- B3: Kiểm tra xem nút con tương tứng với ký tự c có tồn tại hay không. Nếu không tồn tại trả về 0 vì không có từ nào có đường đi giống với xâu s
- B4: Cập nhật con trỏ p để trỏ đến nút con tương ứng với ký tự c. Tiếp tuc vòng lặp để di chuyển con trỏ đến nút cuối cùng của xâu s
- B5: Sau khi duyệt qua xâu s, trả về giá trị “count” tại nút cuối cùng mà con trỏ p đang trỏ đến. “count” là số lượng từ đi qua nút đó tại cây Trie

- Void run():
  - B1: Nhập n là số thao tác thực hiện
  - B2: Khai báo 1 vector để lưu giá trị của phương thức getCount
  - B3: Chạy vòng lặp while để thực hiện n thao tác. Trong mỗi vòng lặp thực hiện khai báo hai biến q,x theo kiểu string rồi nhập. Kiểm tra xem xâu q có phải là “add” không. Nếu đúng thì gọi phương thức addString để nhập xâu x vào cây, nếu không thì tiến hành gọi giá trị của phương thức “getCount” rồi đẩy vào trong vector a. Tiếp tục vòng lặp cho đến khi n bằng 0
  - B4: Chạy vòng for duyệt vector a để in ra theo yêu cầu đề bài

1.3. Cài đặt các lớp và hàm main

1. Class TrieNode{};

```
class TrieNode
{
private:
   TrieNode *child[26];
   int exist, count;

protected:
   friend class Trie;
   TrieNode()
   {
      for (int i = 0; i < 26; i++)
         child[i] = NULL;
      exist = count = 0;
   }
   ~TrieNode(){
      for(int i = 0; i  < 26; i++){
         delete[] child[i];
      }
   }
};
```

2. Class Trie{};

```
class Trie

{

private:
```

```
    int cur;
    TrieNode *root;


public:
    Trie() : cur(0)
    {
        root = new TrieNode();
    }
    void addString(string s)
    {
        TrieNode *p = root;
        for (auto c : s)
        {
            int temp = c - 'a';
            if (p->child[temp] == NULL)
                p->child[temp] = new TrieNode();
            p = p->child[temp];
            p->count++;
        }
```

```
    p->exist++;
}


int getCount(string s)
{
    TrieNode *p = root;
    for (auto c : s)
    {
        int temp = c - 'a';
        if (p->child[temp] == NULL)
            return 0;
        p = p->child[temp];
    }
    return p->count;
}
void run()
{
    int n;
    cin >> n;
```

```
vector<int> a;
while (n--)
{
    string q, x;
    cin >> q >> x;
    if (q == "add")
    {
        this->addString(x);
    }
    else
    {
        a.push_back(this->getCount(x));
    }
}
for (auto x : a)
{
    cout << x << endl;
}
```

```
    }
    ~Trie(){
        if(root){
            delete[]root;
        }
    }
};
```

3. Int main(){};

```
int main()
{
    ios_base::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    Trie *trie = new Trie();
    trie->run();
    return 0;
}
```

1.4. Phân tích thời gian chạy của từng phương thức có trong các lớp

- Class TrieNode
    - Phương thức khởi tạo, hủy: O(1).

- Class Trie
    - Phương thức khởi tạo: O(1)
    - addString: O(n)
    - getCount: O(n)
    - run: O(1)

Bài 2: (Bài số 18 trong danh sách bài tập)

2.1. Đề bài:

1. Xây dựng cấu trúc dữ liệu ngăn xếp
2. Xây dựng lớp biểu diễn đồ thị vô hướng có trọng số bằng ma trận kề có các phương thức:
    a. Nhập đồ thị từ file
    b. Ghi đồ thị ra file
    c. Duyệt đồ thị theo chiều sâu (DFS)
    d. Tìm đường đi ngắn nhất giữa 2 đỉnh bất kỳ
3. Viết hàm main thực hiện công việc trên

2.2. Phân tích bài toán:

- Yêu cầu của bài toán là xây dựng cấu trúc dữ liệu ngăn xếp và lớp biểu diễn đồ thị có trọng số bằng ma trận kề có các phương thức thực hiện được các yêu cầu như nhập đồ thị từ file, ghi đồ thị ra file, duyệt đồ thị theo chiều sâu, tìm đường đi ngắn nhất giữa hai đồ thị
- Các lớp cần xác định là:
    - Lớp mystack để xây dựng cấu trúc dữ liệu ngăn xếp
        - Các thuộc tính của lớp:
            - T *buf: con trỏ buf để quản lý mảng động chứa các phần tử của stack

        - Int n: để lưu số phần tử hiện tại có trong stack
        - Int cap: để lưu số phần tử tối đa mà stack có thể chứa được
    - Các phương thức của lớp:
        - mystack(){} : Phương thức khởi tạo của mystack sau khi được khai báo
        - ~mystack() {}: Phương thức hủy của mystack, sử dụng để hủy biến con trỏ buf
        - Int size() : Phương thức sẽ trả về số lượng phần tử có trong mystack hiện tại
        - Bool empty() : Phương thức sẽ trả về xem trong mystack đã trống phần tử hay chưa
        - Void pop() : Phương thức loại bỏ phần tử trên cùng của mystack
        - T &top() : Phương thức trả về giá trị trên cùng của mystack
        - Void push(T x) : Phương thức thêm vào mystack 1 phần tử của giá trị x kiểu T
- Lớp Graph để thực hiện chức năng như đề bài yêu cầu
    - Các thuộc tính của lớp:
        - Int size: dùng để lưu số nốt của đồ thị
        - Vector<vector<int>> matrixQ: dùng để lưu ma trận có kích thước là size x size và biểu diễn trọng số của đồ thị.
    - Các phương thưc của lớp:
        - Graph() : Phương thức khởi tạo của Graph sau khi được khai báo
        - Graph(int vertices) : size(vertices), matrixQ(vertices, vector<int>(vertices, 0)) {}: Phương thức dùng để khởi tạo đồ thị với biến vertices

        - Void add_edge(int u, int v, int weight) : Phương thức dùng để nhập trọng số weight giữa hai đỉnh u và v
        - Void print_graph(): Phương thức dùng để in ra ma trận kề có trọng số
        - Void readfile(const string &filename): Phương thức dùng để đọc đồ thị từ file
        - Void writefile(const string &filename): Phương thức dùng để ghi đồ thị ra file
        - Void DFS(int u): Phương thức dùng để duyệt đồ thị theo chiều sâu bắt đầu từ đỉnh u
        - Void dijkstra(int start, int end) : Phương thức dùng để tìm đường đi ngắn nhất giữa hai đỉnh start và end được nhập

- Cách hoạt động
    - Lớp mystack
        - Int size() : Hàm trả về số phần tử có trong stack
        - Bool empty: Hàm trả về xem stack có rỗng hay là không
        - Void pop(): Hàm trả xóa đi mất 1 phần tử ở trên cùng của stack
        - T &top() : Hàm truy cập đến phần tử ở đỉnh của stack mà không thay đổi cấu trúc của stack
        - Void push(T x):
            - B1: Kiểm tra xem stack đã đầy hay chưa. Nếu chưa thì cập nhật thêm giá trị x vào ô nhớ tiếp theo trong stack. Nếu đầy rồi thì thực hiện các lệnh trong if
            - B2: Nếu mà đầy rồi thì nhân đôi giá trị của biến cap nếu cap không phải là 0 hoặc gán cho cap bằng 1 nếu cap bằng 0
            - B3: Tạo một con trỏ có kiểu dữ liệu là T sau đó cấp phát bộ nhớ cho nó bằng cap

- B4: Chạy vòng for từ i bằng 0 tới n, trong vòng for lần lượt gán các giá trị của tmp[i] = buf[i]
- B5: Kiểm tra xem buf có khác NULL hay không nếu có bằng NULL thì thực hiện lệnh delete để giải phóng bộ nhớ đã cấp phát cho mảng động
- B6: Cập nhật con trỏ buf sao cho nó trỏ đến vùng nhớ mà con trỏ tmp đang trỏ đến

- Lớp Graph
  - Void add_edge(int u, int v, int weight):
    - Xét cho tất cả các phần tử trong ma trận có chỉ số [u-1][v-1] và ngược lại có hệ số bằng weight
  - Void readfile(const string &filename):
    - Kiểm tra file nhập vào, nếu file mở được thì nhập vào giá trị của size tức số đỉnh trong đồ thị.
    - Cấp phát cho ma trận matrixQ với kích thước size x size matrixQ = vector<vector<int>(size, vector<int>(size,0));
    - Nhập vào số cạnh của đồ thị sau đó duyệt đủ số cạnh, hình thành ma trận kề ban đầu gồm các đỉnh từ 1 đến đỉnh có chỉ số lớn nhất
    - Nhập đủ số cạnh, với mỗi cạnh được duyệt sẽ xét 2 đỉnh đầu và cuối cùng trọng số sau đó đối với mỗi phần tử trong ma trận, phần tử nào có chỉ số [đỉnh đầu][đỉnh cuối] và ngược lại thì có chỉ số bằng trọng số, các phần tử khác có chỉ số bằng 0.
  - Void writefile(const string &filename):
    - Kiểm tra xem file có mở được không.
    - In ra số đỉnh sau đó duyệt ma trận xem có bao nhiêu số có chỉ số lớn hơn 0 thì công thêm cho biến cạnh. Kết thúc thì in ra số cạnh của đồ thị.

- Duyệt ma trận kề sau đó in ra theo cú pháp quy định sẵn " Đỉnh A kết nối với đỉnh B có trọng số C ".

- Void DFS():
  - Nhập đỉnh bắt đầu duyệt.
  - Xây dựng một ngăn xếp có kiểu số nguyên để lưu đỉnh đã duyệt. Khai báo thêm một vector visited có kiểu bool với kích thước bằng size và giá trị các phần tử đều bằng false.
  - Xét đỉnh bắt đầu duyệt, nhập đỉnh đó vào stack và gán giá trị vector visited tại u bằng true
  - Chạy vòng lặp cho đến khi stack trống hết, khai báo một biến check và gán cho giá trị trên đỉnh của stack sau đó xóa phần tử đó khỏi stack. In phần tử đó ra và gán giá trị visited tại check bằng true.
  - Tiếp tục chạy thêm vòng lặp for để duyệt các phần tử tiếp theo, kiểm tra xem phần tử i đã được kiểm tra chưa nếu rồi thì push i vào stack và đánh dấu visited tại i = true.
- Void dijkstra():
  - Khai báo hai đỉnh muốn tính khoảng cách
  - Khởi tạo vector dist có size + 1 phần tử để lưu khoảng cách nhỏ nhất để đi từ đỉnh đầu đến mỗi đỉnh, vector res để lưu đỉnh có đường đi ngắn nhất từ đỉnh đầu
  - Khởi tạo hàng đợi ưu tiên priority_queue với cặp giá trị là khoảng cách ngắn nhất đi đến mỗi đỉnh và đỉnh đó. Ưu tiên từ bé đến lớn.
  - Sử dụng vòng lặp while đến khi hàng đợi rỗng. Trong mỗi vòng lặp sẽ lấy giá trị ở đầu hàng đợi, kiểm tra xem khoảng cách ngắn nhất ở các đỉnh kề hiện tại có ngắn

hơn khoảng cách từ nó đi đến hay không. Nếu có đường đi khác ngắn hơn sẽ cập nhật và push vào hàng đợi. Cập nhật lại giá trị trong vector res để lưu lại đỉnh kề có đường đi tới ngắn nhất.

- Khởi tạo vector _r để lưu đường đi ngắn nhất. Push đỉnh cuối cùng vào, dùng vòng lặp while để truy xuất đường đi đến mỗi đỉnh.

2.3. Cài đặt các lớp và hàm main bằng C++:

1. File class mystack

```
#include <bits/stdc++.h>

#ifndef _mystack_cpp

#define _mystack_cpp

#define ll long long

#define MOD 10e9+7

using namespace std;

template <class T>

class mystack

{

private:

   T *buf;

   int n, cap;
```

```
public:
    mystack()
    {
        n = cap = 0;
        buf == NULL;
    }
    ~mystack()
    {
        if (buf)
            delete[] buf;
    }
    int size() { return n; }
    bool empty() { return n == 0; }
    void pop() { n--; }
    T &top() { return buf[n - 1]; }
    void push(T x)
    {
        if (n == cap)
        {
```

```
            cap = cap ? cap * 2 : 1;
            T *tmp = new T[cap];
            for (int i = 0; i < n; i++)
                tmp[i] = buf[i];
            if (buf)
                delete[] buf;
            buf = tmp;
        }
        buf[n++] = x;
    }
};
#endif
```

2. File Graph

```
#include <bits/stdc++.h>
#include "mystack.cpp"
#ifndef _graph_cpp
#define _graph_cpp
#define ll long long
```

```
#define INF LLONG_MAX

#define MOD 10e9 + 7

using namespace std;

class Graph

{

private:

    int size;

    vector<vector<int>> matrixQ;

public:

    Graph() { size = 0; }

    Graph(int vertices) : size(vertices), matrixQ(vertices, vector<int>(vertices, 0)) {}

    void add_edge(int u, int v, int weight)

    {

        matrixQ[u - 1][v - 1] = weight;

        matrixQ[v - 1][u - 1] = weight;

    }

    void print_graph()

    {

        if (size == 0)
```

```
    {
        cout << "Do thi trong ! " << endl;
        return;
    }
    cout << "Ma tran ke: " << endl;
    for (int i = 0; i < size; ++i)
    {
        for (int j = 0; j < size; ++j)
        {
            cout << matrixQ[i][j] << ' ';
        }
        cout << endl;
    }
}
void readfile(const string &filename)
{
    ifstream file(filename);
    if (file.is_open())
    {
```

```
file >> size;
if (size <= 0)
{
    cerr << "Chi so khong hop !" << endl;
    file.close();
    return;
}
matrixQ = vector<vector<int>>(size, vector<int>(size, 0));
int edges;
file >> edges;
if (edges <= 0)
{
    cerr << "Chi so khong hop le !" << endl;
    file.close();
    return;
}
int u, v, weight;
while (edges--)
{
```

```
            file >> u >> v >> weight;
            if (u <= 0 || u > size || v <= 0 || v > size)
            {
                cerr << "Chi so khong hop le !" << endl;
                continue;
            }
            add_edge(u, v, weight);
        }
        file.close();
    }
    else
    {
        cerr << "Khong the tao hoac mo tep :" << filename << endl;
    }
}
void writefile(const string &filename)
{
    ofstream file(filename);
    if (file.is_open())
```

```
{
    file << "So dinh: " << size << endl;
    int edges = 0;
    for (int i = 0; i < size; i++)
    {
        for (int j = i + 1; j < size; j++)
        {
            if (matrixQ[i][j] > 0)
            {
                edges++;
            }
        }
    }
    file << "So canh: " << edges << endl;
    for (int i = 0; i < size; i++)
    {
        file << "Dinh " << i + 1 << " ket noi voi: " << endl;
        for (int j = 0; j < size; j++)
        {
```

```
            if (matrixQ[i][j] > 0)
            {
                file << "\tDinh " << j + 1 << " co trong so: " << matrixQ[i][j] << endl;
            }
        }
    }
    file.close();
  }
  else
  {
    cerr << "Khong the tao hoac mo tep: " << filename << endl;
  }
}
void DFS()
{
  int u;
  cout << endl << "Nhap dinh can duyet: "; cin >> u;
  mystack<int> s;
  vector<bool> visited(size, false);
```

```
s.push(u-1);
visited[u-1] = true;
cout << "DFS: ";
while (!s.empty())
{
   int check = s.top();
   s.pop();
   cout << check + 1 << ' ';
   visited[check] = true;
   for (int i = 0; i < matrixQ[check].size(); ++i)
   {
      if (!visited[i] && matrixQ[check][i])
      {
         s.push(i);
         visited[i] = true;
      }
   }
}
cout << endl;
```

```
}
void dijkstra()
{
    int start_node, end_node;
    cout << "Enter the starting node for Dijkstra: ";
    cin >> start_node ;
    cout << endl <<  "Enter the ending node for Dijkstra: ";
    cin >> end_node;
    vector<int> dist(size + 1, INT_MAX), res(size + 1);
    dist[start_node - 1] = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    pq.push({0, start_node - 1});
    while (!pq.empty())
    {
        int w = pq.top().first;
        int u = pq.top().second;
        pq.pop();
        for (int v = 0; v < matrixQ[u].size(); v++)
        {
```

```
        if (dist[u] + matrixQ[u][v] < dist[v] && matrixQ[u][v] != 0)
        {
            res[v] = u;
            dist[v] = dist[u] + matrixQ[u][v];
            pq.push({dist[v], v});
        }
    }
}
cout << "Shortest distances from node " << start_node - 1 << ":\n";
int i = res[end_node - 1];
vector<int> _r;
_r.push_back(end_node - 1 );
while (i != start_node - 1)
{
    _r.push_back(i);
    i = res[i];
}
_r.push_back(start_node - 1);
for (int i = _r.size() - 1; i >= 0; i--)
```

```
        cout << _r[i] + 1 << " ";
    cout << "\nKhoang cach: " << dist[end_node - 1] << endl;
  }
};
#endif
```

3. File main

```
#include<bits/stdc++.h>
#include "graph.cpp"
#define ll long long
#define INF LLONG_MAX
#define MOD 10e9+7
using namespace std;
int main(){
  Graph Q;
  Q.readfile("input.txt");
  Q.writefile("output.txt");
  Q.DFS();
  Q.print_graph();
```

```
    Q.dijkstra();

}
```

2.4. Phân tích thời gian chạy của từng phương thức có trong các lớp:

- Class mystack:
    - Phương thức khởi tạo, hủy, empty,size, pop, top, push: O(1)
- Class Graph:
    - Phương thức khởi tạo, add_edge: O(1)
    - Phương thức: print_graph, readfile, writefile, DFS, dijkstra: O(n^2)

Danh sách tài liệu tham khảo

[1] Slide bài giảng Cấu trúc dữ liệu và Giải thuật.

[2] Code mẫu trong bài giảng Cấu trúc dữ liệu và Giải thuật.

[3] Diễn đàn tin học, thuật toán VNOI: https://vnoi.info/