TRƯỜNG ĐẠI HỌC GIAO THÔNG VÂN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

---------------o0o---------------



**BÀI TẬP LỚN MÔN HỌC**

**CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

Giảng viên hướng dẫn: Phạm Xuân Tích

Sinh viên thực hiện: Trần Huy Hiệp - Lớp CNTT 1 - K61

**Hà Nội, tháng 12 năm 2021**

Mục lục

[I. Đề bài 3](#_Toc89186318)

[II. Phân tích bài toán 3](#_Toc89186319)

[1. Xác định các lớp, các thuộc tính, các phương thức của lớp và mô tả chức năng của từng lớp, từng phương thức. 3](#_Toc89186320)

[a. Lớp STACK 3](#_Toc89186321)

[b. Lớp Grap 3](#_Toc89186322)

2[. Vẽ sơ đồ khối của những phương thức có chứa các cấu trúc if, cấu trúc lặp và đánh giá độ phức tạp của bài toán 4](#_Toc89186325)

[a. Lớp STACK 4](#_Toc89186321)

* void push(T x) 4

[b. Lớp Grap 4](#_Toc89186322)

* void dfs() 6
* void dijkstra(int start, int finish) 7

3[. Phân tích thời gian chạy của từng phương thức 10](#_Toc89186325)

[III. Cài đặt các lớp và hàm main bằng C++ 10](#_Toc89186339)

[IV. Danh sách tài liệu tham khảo 17](#_Toc89186340)

## I. 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:
   1. Nhập đồ thị từ file
   2. Ghi đồ thị ra file
   3. Duyệt đồ thị theo chiều sâu (DFS)
   4. Tìm đường đi ngắn nhất giữa 2 đỉnh bất kỳ
3. Viết hàm main thực hiện các công việc trên

## II. Phân tích bài toán

### 1, Xác định các lớp, các thuộc tính, các phương thức của lớp và mô tả chức năng của từng lớp, từng phương thức

**a, Lớp ngăn xếp (STACK):** Lớp dùng để xây dựng cấu trúc ngăn xếp.

* *Các thuộc tính:*

+, element: Giá trị kiểu mẫu, dùng để lưu trữ dữ liệu của STACK.

+, n: Giá trị kiểu số nguyên (int), dùng để lưu số phần tử của STACK..

+, capacity: Giá trị kiểu số nguyên (int), dùng để lưu số phần tử tối đa có thể chứa của STACK.

* *Các phương thức:*

+, Hàm tạo không đối STACK(): Dùng để khởi tạo ngăn xếp (STACK) mới và không có phần tử nào.

+, Hàm hủy ~STACK(): Dùng để hủy hàm ngăn xếp (STACK).

+, size(): Trả về kích thước hiện tại của ngăn xếp (STACK).

+, empty(): Kiểm tra ngăn xếp (STACK) có rỗng hay không.

+, &top(): Trả về phần tử được bổ sung vào cuối cùng của ngăn xếp (STACK) và phẩn tử này có thể thay đổi giá trị được được.

+, pop(): Xóa bỏ phần tử được bổ sung vào cuối cùng của ngăn xếp (STACK).

+, push(element): Thêm phần tử element vào ngăn xếp (STACK).

**b, Lớp đồ thị (Graph):** Lớp dùng để khai báo đồ thị vô hướng và giải quyết bài toán duyệt đồ thị theo chiều sâu, và tìm đường đi ngắn nhất giữa 2 đỉnh bất kỳ.

* *Các thuộc tính:*

+, n: Giá trị kiểu số nguyên (int), dùng để lưu số đỉnh của đồ thị hay còn lưu số hàng số cột của ma trận.

+, \*\*a: Con trỏ kiểu số nguyên (int), dùng để lưu trữ giá trị của mảng 2 chiều, lưu trữ trọng số của các đỉnh của đồ thị.

* *Các phương thức:*

+, Hàm tạo không đối Grap(): Dùng để khởi tạo đồ thị (Graph) mới và không có đỉnh nào.

+, readfile(namefile): Dùng để đọc dữ liệu từ file có tên “namefile”.

+, write(): Hàm dùng để in ma trận vừa nhập từ file ra màn hình.

+, writefile(filename): Hàm dùng để in ma trận vừa nhập từ file ra file có tên “filename”.

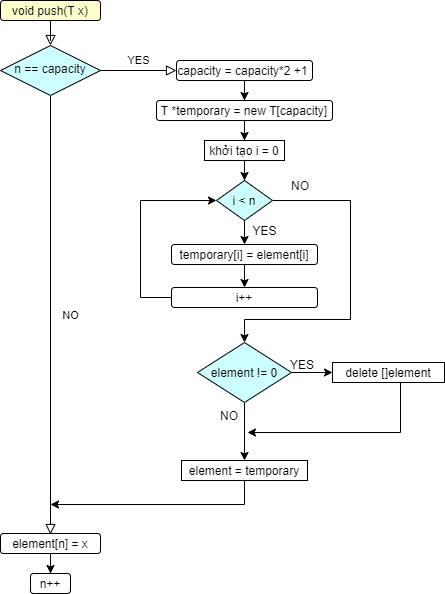
+, dfs(): Duyệt đồ thị vô hướng theo chiều sâu (DFS)

+, dijkstra(start, finish): Tìm đường đi ngắn nhất giữa 2 đỉnh, bắt đầu từ đỉnh start và kết thúc ở đỉnh finish. Tìm và in ra giá trị nhỏ nhất và in ra đường đi giữa 2 đỉnh.

### 2, Vẽ sơ đồ khối của những phương thức có chứa các cấu trúc if, cấu trúc lặp và đánh giá độ phức tạp bài toán

**a, Lớp STACK**

* **push(T x)**

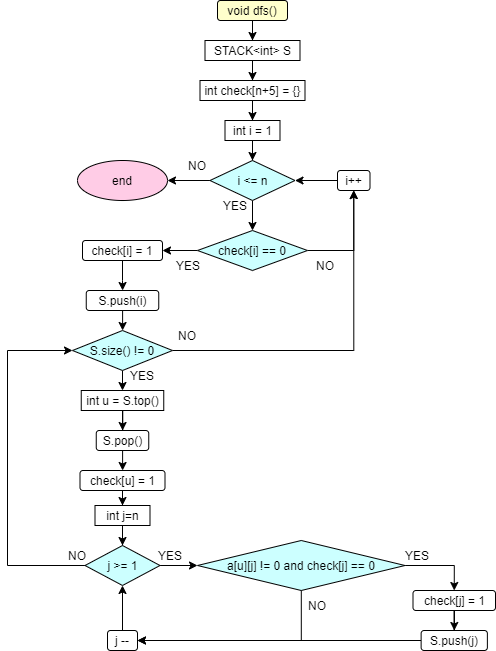
****

**Nhận xét:** Khi thực hiện phép toán push, trong khi mảng đầy sẽ dẫn đến ngoại lệ. Ta có thể thay thế mảng bằng mảng có kích thước lớn hơn. Với chiến lược gia tăng ta thay thế mảng cũ bằng một mảng mới có kích thước bằng kích thước mảng cũng cộng với hằng số c (cụ thể ở đây là 1). Ta thực hiện thay thế mảng k = log2n lần. Với chiến lược gấp đôi này với T(n) của việc push chuỗi n phần tử vào vào STACK tương ứng với: n+1+2+4+…+2k = n+2k+1-1 = 2n-1. Vậy T(n) là O(n) => mỗi lần push là O(1)

* **Độ phức tạp thời gian của thuật toán push là:** O(1).

**b, Lớp Graph**

* **void dfs()**

****

**Các biến sử dụng trong thuật toán:**

+, check[i]: dùng để đánh dấu điểm i đã được duyệt hay chưa (0 là chưa duyệt và 1 là đã duyệt)

+, S: là một ngăn xếp STACK dùng để lưu các đỉnh sẽ được duyệt.

**Cách triển khai:** Thuật toán DFS duyệt đồ thị theo chiều sâu.

* Tư tưởng của thuật toán tìm theo chiều sâu là bắt đầu tại tại 1 đỉnh u và đánh đấu điểm đó là đã duyệt. Tiếp đến sẽ duyệt đến tất cả các đỉnh kề với đỉnh u gọi tập các đỉnh kề là j đánh dấu điểm đó dã duyệt và in ra đỉnh kề nhỏ nhất. Tiếp tục lấy đỉnh kề nhỏ nhất để đi tìm các đỉnh kề của nó và đánh đấu. Cứ như thế cho đến tất cả các đỉnh. Và ta muốn duyệt tất cả các đỉnh (kể cả đỉnh không có cạnh nối) nên ta dùng vòng for duyệt i từ 1 tới n (duyệt tất cả các đỉnh). Cụ thể

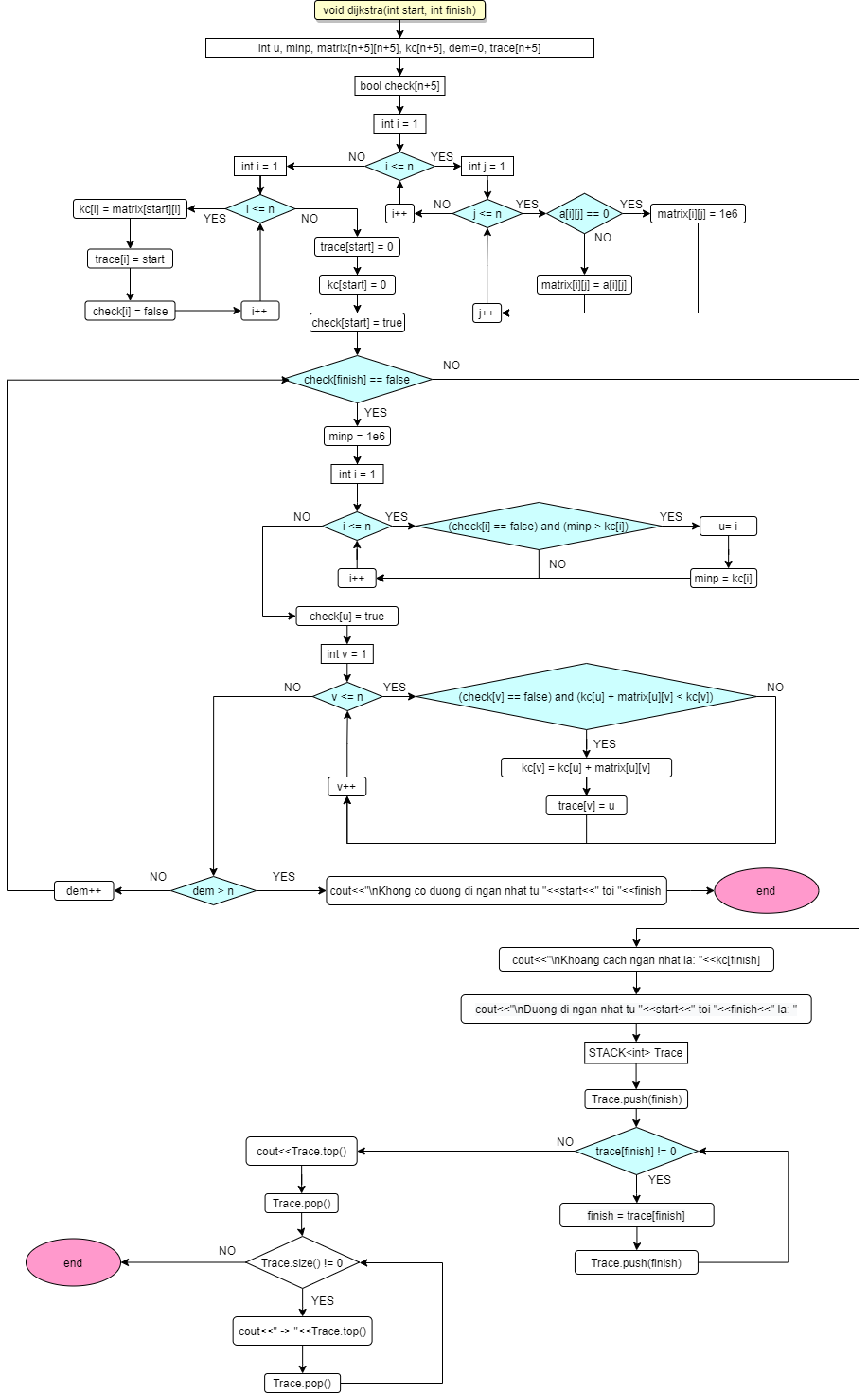
**Bước 1:** Chọn điểm i là gốc, nếu đỉnh i chưa được duyệt thì đánh dấu đỉnh i là đã duyệt (check[i] = 1) cho vào ngăn xếp và in ra đỉnh I. Nếu đã duyệt thì đi đến đỉnh tiếp theo và quay lại bước 1.

**Bước 2:** Lấy điểm đầu (top) của ngăn xếp ra và gọi là đỉnh u, đánh dấu là đã duyệt (check[u] = 1). In ra màn hình đỉnh u.

**Bước 3:** Duyệt tất cả đỉnh kề với u và cho vào ngăn xếp và đánh dấu là đã duyệt.

**Bước 4:** Kiểm ra ngăn xếp có còn phần tử nào không, nếu có thì quay lại bước 2. Nếu ngăn xếp rỗng thì quay lại bước 1.

* **Độ phức tạp thời gian của thuật toán Dfs là:** O(V + E) với V là số đỉnh và E là số cạnh.
* **void dijkstra(int start, int finish)**

****

**Các biến sử dụng trong thuật toán:**

+, u: Biến kiểu số nguyên, dùng để lưu giá trị của đỉnh tạm thời.

+, minp: Biến kiểu số nguyên, dùng để lưu giá trị nhỏ nhất của đỉnh tạm thời.

+, matrix[][]: Ma trận kiểu số nguyên, dùng để lưu khoảng cách giữa các đỉnh nếu không có đường đi sẽ lưu một giá trị lớn (khoảng 1000000)

+, kc[]: Mảng kiểu số nguyên, lưu quãng đường đi nhỏ nhất từ đỉnh start đến các đỉnh khác.

+, start: Biến kiểu số nguyên, lưu đỉnh bắt đầu.

+, finish: Biến kiểu số nguyên, lưu đỉnh kết thúc.

+, dem: lưu số lần đi của thuật toán.

+, check[]: Mảng kiểu bool, dùng để kiểm tra đỉnh đó đã được duyệt hay chưa với giá trị false là chưa được duyệt và true là đã được duyệt.

+, trace[]: Mảng kiểu số nguyên, dùng để lưu quãng đường đi của thuật toán.

**Cách triển khai:** thuật toán Dijkstra tìm đường đi ngắn nhất giữa 2 đỉnh.

* Xét đồ thị G=<V,E>: Trong đó |V| là số đỉnh, và E là số đường đi. Với mỗi cạnh (u, v) thuộc E, ta đặt tương ứng với nó một số nguyên maxtrix<u,v> được gọi là trọng số của cạnh. Ta sẽ đặt matrix[u,v] = 1e6, Nếu (u, v) không thuộc E. Nếu dãy v1, v2, .. , vk là một đường đi trên G thì tổng của tất cả các cạnh (A[vi-1, vi]) được gọi là độ dài của dường đi.
* Từ ma trận trọng số matrix[u][v], u, v thuộc V, ta tìm cận trên kc[v] của khoảng cách từ s đến tất cả các đỉnh v thuộc V. Mỗi khi phát hiện thấy kc[u] + matrix[u][v] < kc[v] thì cận trên kc[v] sẽ được làm tốt lên bằng cách gán kc[v] = kc[u] + matrix[u][v]. Quá trình sẽ kết thúc khi nào ta không thể làm tốt hơn lên được bất kỳ cận trên nào, khi đó kc[v] sẽ cho ta giá trị ngắn nhất từ đỉnh s đến v. Giá trị kc[v] được gọi là nhãn của đỉnh v.
* Các bước triển khai cụ thể như sau:

**Bước 1**: Gán tạm thời khoảng cách từ điểm bắt đầu đến các điểm khác với giá trị là khoảng cách từ điểm đầu đến điểm khác tương ứng và gán cho nhãn đỉnh bắt đầu(start) là 0 (hay kc[start] = 0) .

**Bước 2**: Kiểm tra điểm kết thúc đã được duyệt hay chưa. Nếu chưa thì tiếp tục bước 3, nếu đã duyệt thì kết thúc chương trình.

**Bước 3:** Tìm đỉnh u là đỉnh có nhãn tạm thời sao cho khoảng cách từ điểm start đến đỉnh u là nhỏ nhất (hay tìm điểm u sao cho kc[u] là min). Sau đó đánh dấu đỉnh u là đã duyệt.

**Bước 4:** Gán lại nhãn cho các đỉnh, ta tìm điểm kề với đỉnh u gọi là v, và kiểm tra xem đỉnh v đã tối ưu khoảng cách chưa nếu chưa thì ta gán kc[v] = kc[u] + matrix[u][v] ( khoảng cách từ start tới v sẽ thay bằng khoảng cách từ start tới u cộng với khoảng u tới v) và lưu quãng đường đi vào mảng trace;

* **Độ phức tạp thời gian của thuật toán dsijkstra là:** O(V + ElogV) với V là số đỉnh và E là số cạnh. Và phép toán gán ma trận O(n^2)

### 3, Phân tích thời gian chạy của từng phương thức

**a, Lớp STACK**

* **STACK():** O(1)
* **~STACK():** O(1)
* **size():** O(1)
* **empty():** O(1)
* **&top():** O(1)
* **push():** O(1)

**b, Lớp Graph**

* **Graph():** O(1)
* **readfile(string fname):** O(n^2)
* **write():** O(n^2)
* **writefile(string fname):** O(n^2)
* **dfs():** O(V + E)
* **dijkstra(int start, int finish):** O(V + ElogV), phép gán ma trận O(n^2)

# III. Cài đặt các lớp và hàm main bằng C++

* Các lớp và hàm main được cài đặt bằng C++ được lưu trữ trong thư mục source\_code. Trong đó:

+, Lớp STACK được lưu trữ trong cùng một file tên là STACK.cpp

#ifndef \_\_STACK\_\_CPP\_\_

#define \_\_STACK\_\_CPP\_\_

template<class T>

class STACK

{

private:

int n, capacity;

T \*element;

public:

STACK() {

n = 0;

capacity = 0;

element = 0;

}

~STACK() {

if(element != 0) delete []element;

}

int size() {

return n;

}

bool empty() {

return n == 0;

}

T &top() {

return element[n-1];

}

void pop(){

n--;

}

void push(T x)

{

if(n == capacity)

{

capacity = capacity\*2 + 1;

T \*temporary = new T[capacity];

for(int i=0; i<n; i++)

{

temporary[i] = element[i];

}

if(element != 0) delete []element;

element = temporary;

}

element[n] = x;

n++;

}

};

#endif

+, Lớp Graph và hàm main được lưu trữ trong cùng một file tên là graph.cpp

#include<bits/stdc++.h>

#include <conio.h>

#include"STACK.cpp"

using namespace std;

struct Graph

{

private:

int n;

int \*\*a;

public:

Graph() {

n = 0;

a = 0;

}

void readfile(string fname)

{

ifstream fin(fname);

fin>>n;

a = new int\*[n+5];

for(int i=1; i<=n; i++) {

a[i] = new int[n+5];

for(int j=1; j<=n; j++) {

fin >> a[i][j];

}

}

fin.close();

}

void write()

{

cout<<"Do thi duoc nhap tu file la: \n";

for(int i=1; i<=n; i++) {

for(int j=1; j<=n; j++) {

cout <<setw(3) << fixed << a[i][j];

}

cout<<"\n";

}

}

void writefile(string fname)

{

ofstream fout(fname);

for(int i=1; i<=n; i++) {

for(int j=1; j<=n; j++) {

fout <<setw(3) << fixed << a[i][j];

}

fout<<"\n";

}

}

void dfs()

{

cout<<"\nDuyet do thi theo chieu sau (DFS): ";

STACK<int> S;

int check[n+5] = {};

for(int i=1; i<=n; i++)

{

if(check[i] == 0)

{

check[i] = 1;

S.push(i);

cout<<"\n"<<i;

while(S.size() != 0)

{

int u = S.top();

if(u != i) cout<<" -> "<<u;

S.pop();

check[u] = 1;

for(int j=n; j>=1; j--)

{

if(a[u][j] != 0 and check[j] == 0)

{

check[j] = 1;

S.push(j);

}

}

}

}

}

}

void dijkstra(int start, int finish)

{

int u, minp, matrix[n+5][n+5], kc[n+5], dem=0, trace[n+5];

bool check[n+5];

for(int i=1; i<=n; i++)

{

for(int j=1; j<=n; j++)

{

if(a[i][j] == 0) matrix[i][j] = 1e6;

else matrix[i][j] = a[i][j];

}

}

for(int i=1; i<=n; i++)

{

kc[i] = matrix[start][i];

trace[i] = start;

check[i] = false;

}

trace[start] = 0;

kc[start] = 0;

check[start] = true;

while(check[finish] == false)

{

minp = 1e6;

for(int i=1; i<=n; i++)

{

if((check[i] == false) and (minp > kc[i]))

{

u = i;

minp = kc[i];

}

}

check[u] = true;

for(int v=1; v<=n; v++)

{

if((check[v] == false) and (kc[u] + matrix[u][v] < kc[v]))

{

kc[v] = kc[u] + matrix[u][v];

trace[v] = u;

}

}

if(dem > n)

{

cout<<"\nKhong co duong di ngan nhat tu "<<start<<" toi "<<finish;

return;

}

dem++;

}

cout<<"\nKhoang cach ngan nhat la: "<<kc[finish];

cout<<"\nDuong di ngan nhat tu "<<start<<" toi "<<finish<<" la: ";

STACK<int> Trace;

Trace.push(finish);

while(trace[finish] != 0)

{

finish = trace[finish];

Trace.push(finish);

}

cout<<Trace.top();

Trace.pop();

while(Trace.size() != 0)

{

cout<<" -> "<<Trace.top();

Trace.pop();

}

}

};

int main() {

int key;

bool ktnhap = false;

Graph G;

do {

system("cls");

cout<<"1. Nhap ma tran tu file\n";

cout<<"2. In ma tran ra file va man hinh\n";

cout<<"3. Duyet do thi theo thuat toan dfs (tim kiem theo chieu sau)\n";

cout<<"4. Tim duong di ngan nhat giua 2 dinh\n";

cout<<"5. Ket thuc\n";

cout<<"Moi chon: "; cin>>key;

system("cls");

switch(key) {

case 1:

cout<<"1. Nhap ma tran tu file\n";

G.readfile("input.txt");

cout << "Da nhap file tu dong thanh cong !!\n";

ktnhap = true;

break;

case 2:

cout<<"2. In ma tran ra file va man hinh\n";

if(ktnhap == false) {

cout<<"Chua nhap file!!!";

break;

}

G.write();

G.writefile("output.txt");

break;

case 3:

cout<<"3. Duyet do thi theo thuat toan dfs (tim kiem theo chieu sau)\n";

if(ktnhap == false) {

cout<<"Chua nhap file!!!";

break;

}

G.dfs();

break;

case 4:

int start, finish;

cout<<"4. Tim duong di ngan nhat giua 2 dinh\n";

if(ktnhap == false) {

cout<<"Chua nhap file!!!";

break;

}

cout<<"Moi nhap dinh bat dau: "; cin>>start;

cout<<"Moi nhap dinh ket thuc: "; cin>>finish;

G.dijkstra(start, finish);

break;

case 5:

cout<<"KET THUC CHUONG TRINH!!!";

}

getch();

}while(key != 5);

}

# IV. Danh sách tài liệu tham khảo

1. Slide bài giảng môn học Cấu trúc dữ liệu và Giải thuật, *Lecture 8 - Stack.ppt*, Trường Đại Học Giao Thông Vận Tải
2. <https://expressmagazine.net/development/4009/thuat-toan-ve-tim-kiem-theo-chieu-sau-dfs-bang-ngon-ngu-cc>
3. https://expressmagazine.net/development/4024/tim-duong-di-ngan-nhat-dijkstra-cai-dat-bang-cc