|  |
| --- |
| **TRƯỜNG ĐẠI HỌC ĐÀ LẠT**  **KHOA CÔNG NGHỆ THÔNG TIN**    **BÁO CÁO MÔN HỌC**  **CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2**  **Nhóm 10**  **Tên đề tài:**  **ĐỒ THỊ BIỂU DIỄN BẰNG DANH SÁCH CẠNH**          **Giảng viên hướng dẫn: Đinh Viết Tuấn**  **Sinh viên thực hiện: 1610207 – La Quốc Thắng**  **1610229 – Cao Quốc Bảo Toàn**  **1610235 – Nguyễn Hiếu Trung**  ***Đà Lạt – 04/2018*** |

**BÁO CÁO MÔN HỌC**

**CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2**

**Nhóm 10**

**Tên đề tài:**

**ĐỒ THỊ BIỂU DIỄN BẰNG DANH SÁCH CẠNH**

**Mục lục**

[I. Tổng quan về phương pháp 4](#_Toc509251726)

[II. Mục tiêu 5](#_Toc509251727)

[III. Cài đặt 5](#_Toc509251728)

[IV. Các thao tác trên cấu trúc dữ liệu 11](#_Toc509251729)

[1. Thêm một đỉnh có nhãn label vào đồ thị 11](#_Toc509251730)

[2. Xuất thông tin của một đỉnh trong đồ thị 11](#_Toc509251731)

[3. Kiểm tra hai đỉnh u, v có kề nhau (có cạnh nối giữa chúng) hay không? 12](#_Toc509251732)

[4. Thêm một cạnh có trọng số w nối 2 đỉnh u, v vào đồ thị 12](#_Toc509251733)

[5. Lưu thông tin đồ thị xuống file 13](#_Toc509251734)

[6. Tạo đồ thị dữ liệu được lấy từ file 13](#_Toc509251735)

[7. Duyệt đồ thị theo chiều rộng 14](#_Toc509251736)

[8. Duyệt đồ thị theo chiều sâu 14](#_Toc509251737)

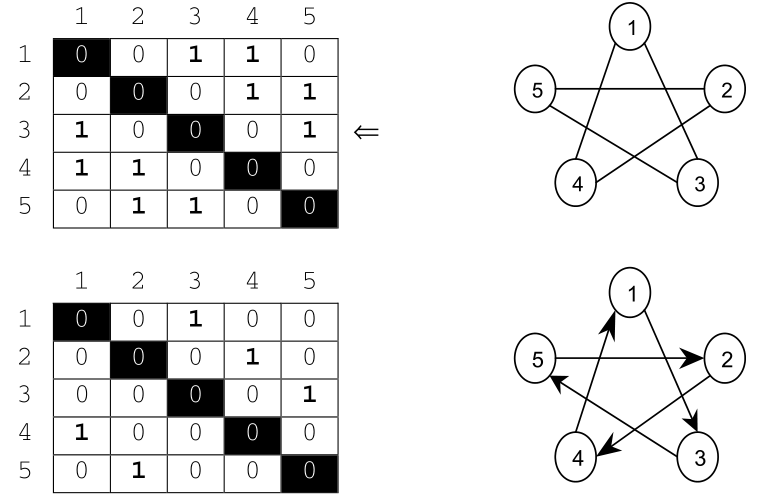
[V. Kết luận 15](#_Toc509251738)

[1. Nhận xét 15](#_Toc509251739)

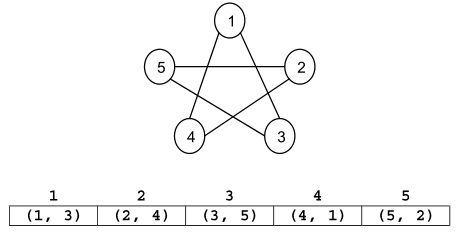
[2. Hướng phát triển 15](#_Toc509251740)

[3. Hạn chế - Khó khăn 15](#_Toc509251741)

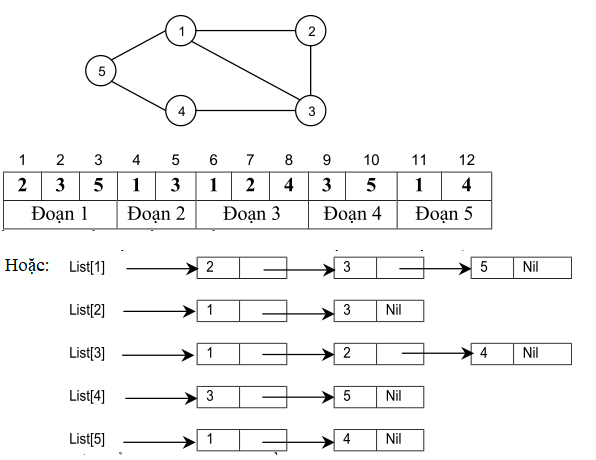
# Tổng quan về phương pháp

Trong lĩnh vực tin học và toán học, để lưu trữ đồ thị và thực hiện các thuật toán khác nhau với đồ thị trên máy tính, người ta thường dùng những cấu trúc để biểu diễn đồ thị như ma trận liền kề (ma trận kề), danh sách cạnh, danh sách kề,…

*Hình 1. Minh họa bằng ma trận liền kề (ma trận kề)*



*Hình 2. Minh họa bằng danh sách cạnh*

*Hình 3. Minh họa bằng danh sách kề*

Trong trường hợp đồ thị có n đỉnh, m cạnh, ta có thể biểu diễn đồ thị dưới dạng danh sách cạnh, trong cách biểu diễn này, người ta liệt kê tất cả các cạnh của đồ thị trong một danh sách, mỗi phần tử của danh sách là một cặp (u, v) tương ứng với một cạnh của đồ thị. (Trong trường hợp đồ thị có hướng thì mỗi cặp (u, v) tương ứng với một cung, u là đỉnh đầu và v là đỉnh cuối của cung)

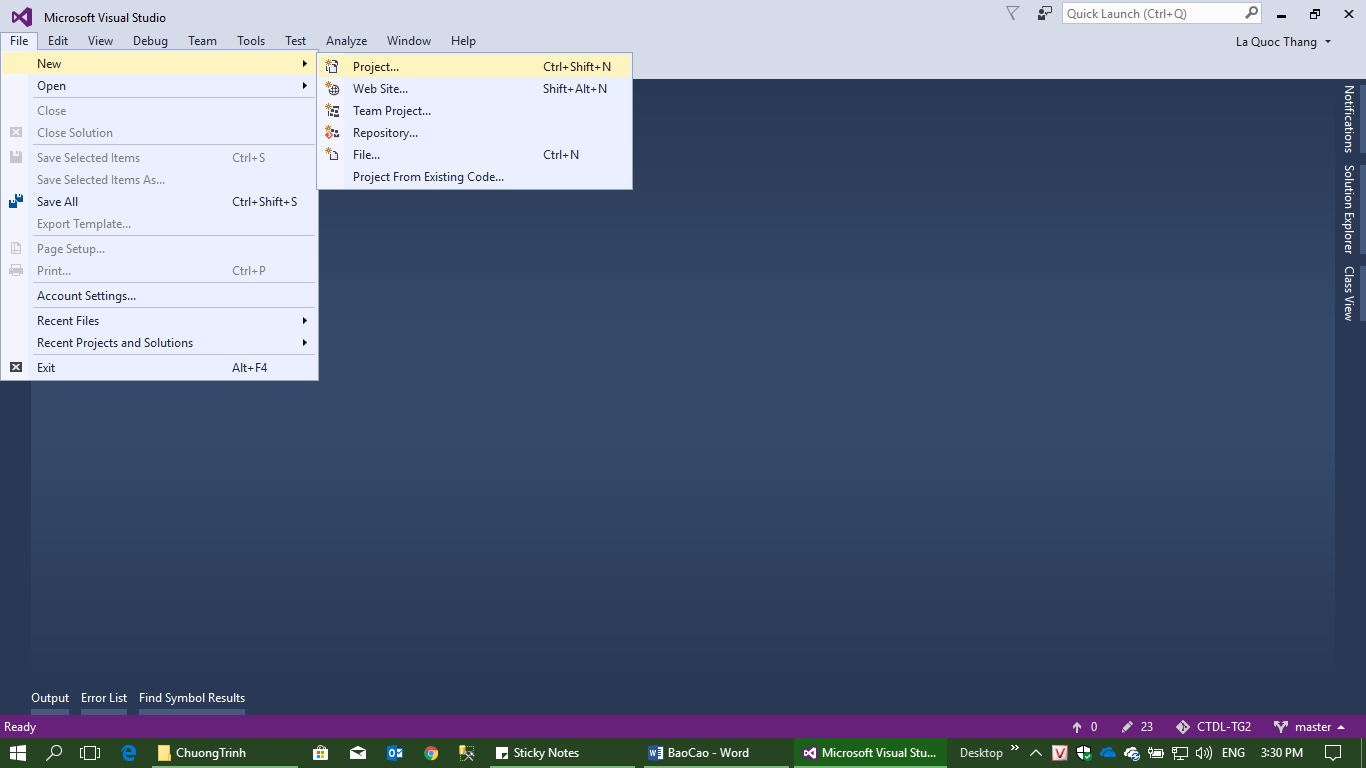
# Mục tiêu

Lưu trữ và biểu diễn đồ thị trên máy tính và thực hiện các thao tác cơ bản trên đồ thị: tạo một đỉnh có nhãn lab, hiển thị thông tin của một đỉnh, khởi tạo 1 đồ thị, thiết lập lại trạng thái của các đỉnh, kiểm tra 2 đỉnh có kề nhau không, thêm một cạnh nối giữa 2 đỉnh, đọc dữ liệu đồ thị từ file (có 6 file text đi kèm), xuất dữ liệu đồ thị vào file và các thao tác duyệt đồ thị (DFS, BFS).

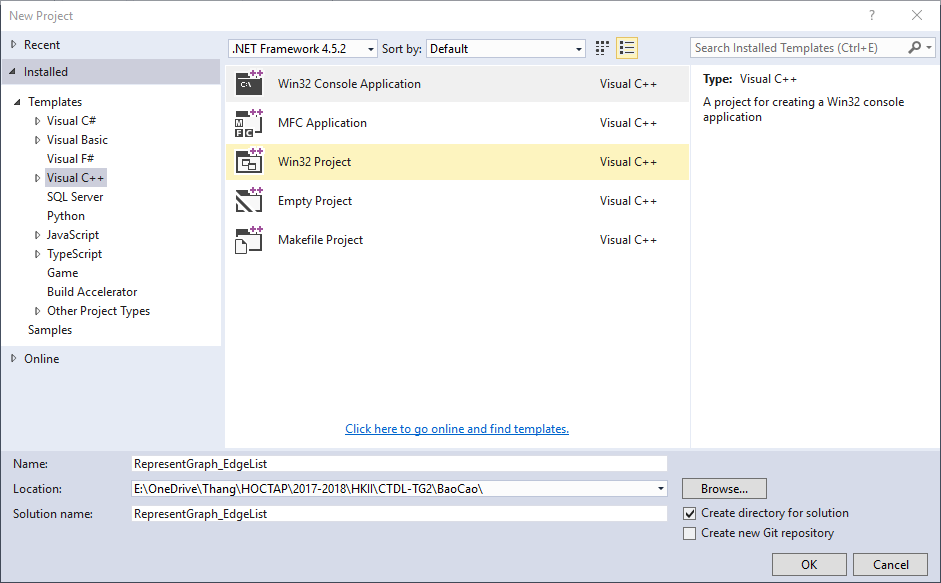
Qua các thao tác trên rút ra được những thuận lợi, khó khăn của phương pháp biểu diễn đồ thị bằng danh sách cạnh để từ đó đề xuất những phương hướng phát triển tốt hơn.

# Cài đặt

Khởi động Microsoft Visual Studio lên, vào File 🡪 New 🡪Project…

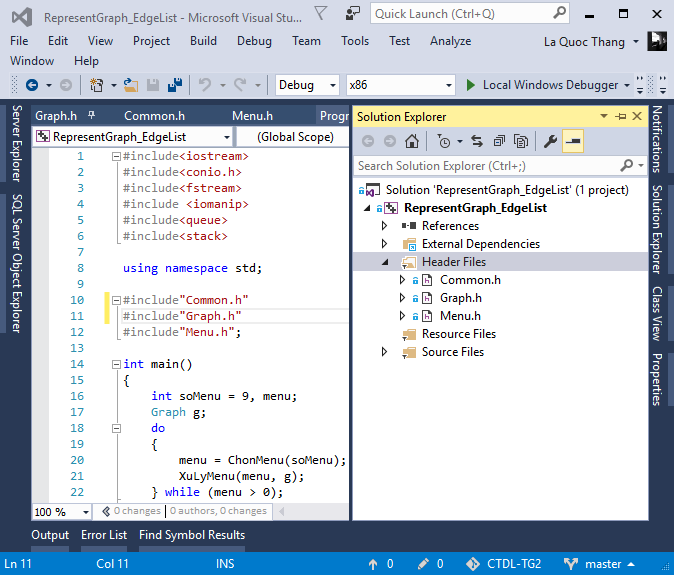


Tiếp theo chọn ngôn ngữ Visual C++ 🡪 chọn Win32 Console Application và đặt tên dự án là RepresentGraph\_EdgeList

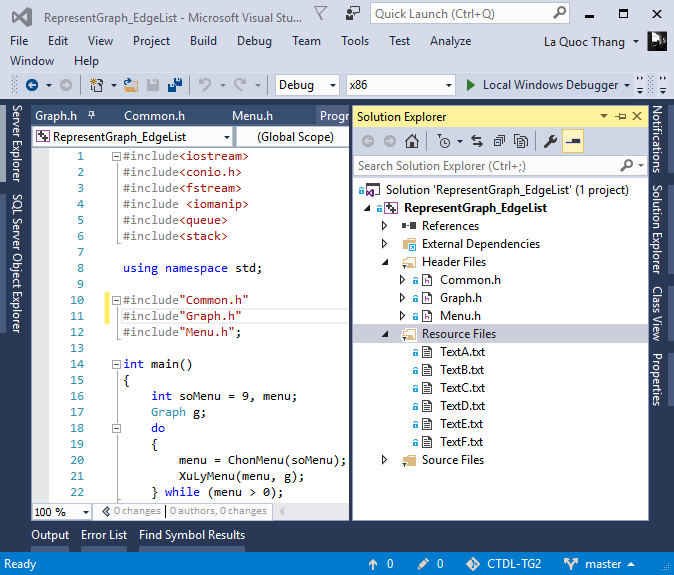


Trong dự án tạo ra 3 tập tin nằm trong thư mục Header Files như sau:

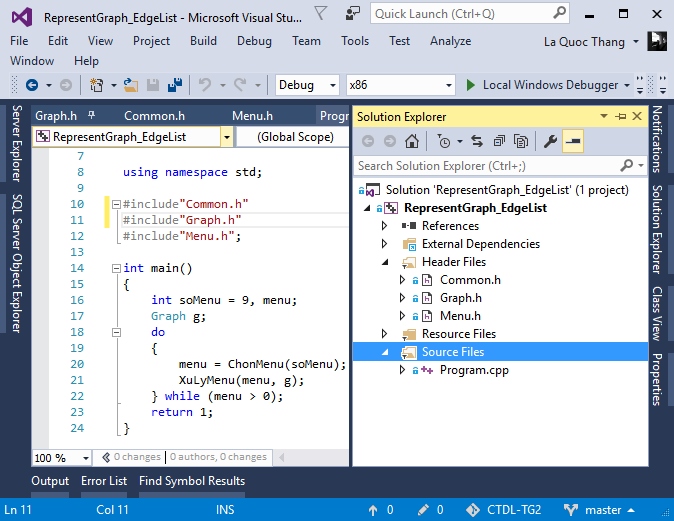
* Menu.h: Định nghĩa menu vận hành các chức năng của chương trình.
* Common.h: Định nghĩa các hằng số và kiểu dữ liệu của đồ thị.
* Graph.h: Định nghĩa các thao tác trên đồ thị.



Thư mục Resource Files ta tạo 6 tập tin Text.txt chứa dữ liệu về đồ thị có cấu trúc xác định để việc đọc dữ liệu thành công.



Thư mục cuối cùng là Source Files, tạo 1 tập tin là Program.cpp chứa hàm main để gọi hàm XuLyMenu ở file Menu.h thực thi các chức năng.



Các bước tiến hành cài đặt:

1. Trong tập tin Common.h:

* Định nghĩa các hằng số và kiểu dữ liệu:

#define UPPER 100 //Số phần tử tối đa

#define ZERO 0 //Giá trị 0

#define MAX 20 //Số đỉnh tối đa

#define INF 1000 //Vô cùng

#define YES 1 //Đã xét

#define NO 0 //Chưa xét

#define NULLDATA -1 //Giá trị rỗng

* Định nghĩa các kiểu dữ liệu:

typedef char LabelType; //Kiểu tên nhãn là một ký tự

typedef int CostType; //Trọng số

* Định nghĩa cấu trúc một cạnh:

struct Data

{

int Marked; //Trạng thái

char source; //Đỉnh đầu

char target; //Đỉnh cuối

CostType Weight; //Trọng số

};

* Định nghĩa cấu trúc danh sách cạnh:

struct Edge

{

Data info;

Edge \*Next;

};

* Định nghĩa cấu trúc của một đỉnh:

struct Vertex

{

LabelType Label; //Nhãn của đỉnh

int Visited; //Trạng thái

};

* Định nghĩa danh sách cạnh: typedef Edge\* LIST;
* Định nghĩa kiểu dữ liệu đồ thị:

struct Graph

{

bool Directed; //ĐT có hướng hay không

int NumVertices; //Số đỉnh

int NumEdges; //Số cạnh

Vertex Vertices[MAX]; //Danh sách đỉnh

LIST ListEdges; //Danh sach cạnh

};

1. Trong tập tin Program.cpp:

#include <iostream>

#include <conio.h>

#include <fstream>

#include <stack>

#include <queue>

using namespace std;

#include "Common.h"

#include "Graph.h"

#include "Menu.h"

int main()

{

int soMenu = 9, menu;

Graph g;

do

{

menu = ChonMenu(soMenu);

XuLyMenu(menu, g);

} while (menu > 0);

return 1;

}

1. Trong tập tin Graph.h:

//Tạo và trả về đỉnh có nhãn là lab

Vertex CreateVertex(LaberType lab)

{

Vertex v;

v.Label = lab;

v.Visited = NO;

return v;

}

//Tạo 1 cạnh từ 2 đỉnh và trả về data của cạnh đó

Data CreateEdge(char start, char end, int w)

{

Data x;

x.Source = start;

x.Target = end;

x.Weight = w;

x.Marked = NO;

return x;

}

//Chuyển một cạnh thành một nút trong danh sách

Edge\* GetEdge(Data v)

{

Edge\* kq = new Edge;

if (kq != NULL)

{

kq->info = v;

kq->Next = NULL;

}

return kq;

}

//Khởi tạo đồ thị

Graph InitGraph(bool directed)

{

Graph g;

g.NumEdges = 0;

g.NumVertices = 0;

g.Directed = directed;

g.ListEdges = NULL;

return g;

}

//Trả về vị trí của một đỉnh trong danh sách đỉnh

int FindIndexVertex(Graph g, char v)

{

for (int i = 0; i < g.NumVertices; i++)

if (g.Vertices[i].Label == v)

return i;

return -1;

}

# Các thao tác trên cấu trúc dữ liệu

## Thêm một đỉnh có nhãn label vào đồ thị

void AddVertex(Graph &g, LaberType lab)

{

//Nếu chưa có đỉnh lab trong đồ thị

if (FindIndexVertex(g, lab) == -1)

{

//Thêm đỉnh vào đồ thị

g.Vertices[g.NumVertices] = CreateVertex(lab);

//Tăng số lượng đỉnh

g.NumVertices++;

}

}

## Xuất thông tin của một đỉnh trong đồ thị

//Xuất tiêu đề

void DisplayTitle()

{

cout << endl;

for (size\_t i = 0; i < 34; i++)

cout << '=';

cout << endl

<< '|'

<< setw(10) << "Dinh dau"

<< '|'

<< setw(10) << "Dinh cuoi"

<< '|'

<< setw(10) << "Trong so"

<< '|';

cout << endl;

for (size\_t i = 0; i < 34; i++)

cout << '=';

}

//Hiển thị thông tin của một cạnh có data là e

void Display1Edge(Data e)

{

cout << endl

<< '|'

<< setw(10) << e.Source

<< '|'

<< setw(10) << e.Target

<< '|'

<< setw(10) << e.Weight

<< '|';

}

//Hiển thị thông tin 1 đỉnh

void DisplayInfoVertex(Graph g, char d)

{

Edge \*e = g.ListEdges;

DisplayTitle();

while (e != NULL)

{

if (e->info.Source == d || e->info.Target == d)

Display1Edge(e->info);

e = e->Next;

}

cout << endl;

for (size\_t i = 0; i < 34; i++)

cout << '=';

}

## Kiểm tra hai đỉnh u, v có kề nhau (có cạnh nối giữa chúng) hay không?

int IsConnected(Graph g, char u, char v)

{

Edge \*e = g.ListEdges;

while (e != NULL)

{

if ((e->info.Source == u) && (e->info.Target == v))

return 1;

e = e->Next;

}

return 0;

}

## Thêm một cạnh có trọng số w nối 2 đỉnh u, v vào đồ thị

//Thêm một cạnh có data v vào danh sách cạnh

void AddEdge(LIST &l, Data v)

{

if (l == NULL) //Khi tới cuối danh sách

l = GetEdge(v); //Thì thêm cạnh đó vào

else

AddEdge(l->Next, v);

}

//Thêm một cạnh vào đồ thị

void AddEdge(Graph &g, char start, char end, CostType w, bool directed)

{

if (!IsConnected(g, start, end))//Nếu 2 đỉnh không kề

{

g.NumEdges++; //Thì thêm cạnh mới vào

AddEdge(g.ListEdges, CreateEdge(start, end, w));

}

}

//Thêm một cạnh vào đồ thị

void AddEdge(Graph &g, char n, char d, CostType w)

{

AddEdge(g, n, d, w, g.Directed);

}

## Lưu thông tin đồ thị xuống file

void SaveGraph(Graph g, char \*f)

{

ofstream out(f);

Edge \*e = g.ListEdges;

out << g.NumVertices << '\t'; //Ghi số đỉnh

out << g.NumEdges << '\t'; //Ghi số cạnh

out << g.Directed; //Ghi kiểu đồ thị

while (e != NULL) //Xuất danh sách cạnh

{

out << '\n' << e->info.Source

<< '\t' << e->info.Target

<< '\t' << e->info.Weight;

e = e->Next;

}

out.close();

}

## Tạo đồ thị dữ liệu được lấy từ file

int OpenFile(char \*f, Graph &g)

{

ifstream in(f);

if (in.is\_open())

{

int a, b, w;

char x, y;

bool directed;

in >> a; //Đọc số đỉnh

in >> b; //Đọc số cạnh

in >> directed; //Đọc hướng đồ thị

g = InitGraph(directed); //Khởi tạo đồ thị

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

{

in >> x; //Đỉnh đầu

in >> y; //Đỉnh cuối

in >> w; //Trọng số

//Thêm cạnh vào đồ thị

AddEdge(g, x, y, w, g.Directed);

//Thêm đỉnh đầu vào danh sách đỉnh

AddVertex(g, x);

//Thêm đỉnh cuối vào danh sách đỉnh

AddVertex(g, y);

}

in.close();

return 1;

}

else

return 0;

}

## Duyệt đồ thị theo chiều rộng

//Tìm đỉnh đầu tiên gằn với cur

char FindFirstAdjacentVertex(Graph g, char cur)

{

for (size\_t i = 0; i < g.NumVertices; i++)

if (g.Vertices[i].Label == cur)

for (size\_t j = 0; j < g.NumVertices; j++)

if (IsConnected(g, cur, g.Vertices[j].Label) && g.Vertices[j].Visited == NO)

return g.Vertices[j].Label;

return NULLDATA;

}

//Duyệt đồ thị theo chiều rộng

void BFS(Graph g, char start)

{

ResetFlags(g);

int index = FindIndexVertex(g, start);

g.Vertices[index].Visited = YES;

queue<char> q;

q.push(start);

cout << start << '\t';

char cur, adj;

while (!q.empty())

{

cur = q.front();

adj = FindFirstAdjacentVertex(g, cur);

if (adj == NULLDATA)

{

q.pop();

if (q.size() != 0)

cur = q.front();

else

break;

}

else

{

index = FindIndexVertex(g, adj);

cout << adj << '\t';

g.Vertices[index].Visited = YES;

q.push(adj);

}

}

}

## Duyệt đồ thị theo chiều sâu

void DFS(Graph g, char start)

{

ResetFlags(g);

int index = FindIndexVertex(g, start);

g.Vertices[index].Visited = YES;//Đánh dấu đỉnh đã xét

cout << start << '\t';

stack<char> s; //Tạo stack

s.push(start); //Thêm đỉnh start vào stack

char cur, adj;

while (!s.empty())

{

cur = s.top();

adj = FindFirstAdjacentVertex(g, cur);

if (adj == NULLDATA)

s.pop();

else

{

int index = FindIndexVertex(g, adj);

cout << adj << '\t';

g.Vertices[index].Visited = YES;

s.push(adj);

}

}

}

# Kết luận

## Nhận xét

## Hướng phát triển

## Hạn chế - Khó khăn

Nhược điểm cơ bản của danh sách cạnh là khi ta cần duyệt tất cả các đỉnh kề với đỉnh v nào đó của đồ thị, thì chẳng có cách nào khác là phải duyệt tất cả các cạnh, lọc ra những cạnh có chứa đỉnh v và xét đỉnh còn lại. Điều đó khá tốn thời gian trong trường hợp đồ thị dày (nhiều cạnh).