|  |
| --- |
| **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](#_Toc510630756)

[II. Mục tiêu 5](#_Toc510630757)

[III. Cài đặt 5](#_Toc510630758)

[IV. Các thao tác trên cấu trúc dữ liệu 13](#_Toc510630759)

[1. Xuất danh sách cạnh của đồ thị 13](#_Toc510630760)

[2. Thêm một đỉnh có nhãn label vào đồ thị 14](#_Toc510630761)

[3. Xuất thông tin của một đỉnh trong đồ thị 14](#_Toc510630762)

[4. Kiểm tra hai đỉnh u, v có kề nhau (có cạnh nối giữa chúng) hay không? 14](#_Toc510630763)

[5. Thêm một cạnh có trọng số w nối 2 đỉnh u, v vào đồ thị 15](#_Toc510630764)

[6. Lưu thông tin đồ thị xuống file 16](#_Toc510630765)

[7. Tạo đồ thị dữ liệu được lấy từ file 16](#_Toc510630766)

[8. Duyệt đồ thị theo chiều rộng 17](#_Toc510630767)

[9. Duyệt đồ thị theo chiều sâu dùng vòng lặp 18](#_Toc510630768)

[10. Duyệt đồ thị theo chiều sâu dùng đệ quy 18](#_Toc510630769)

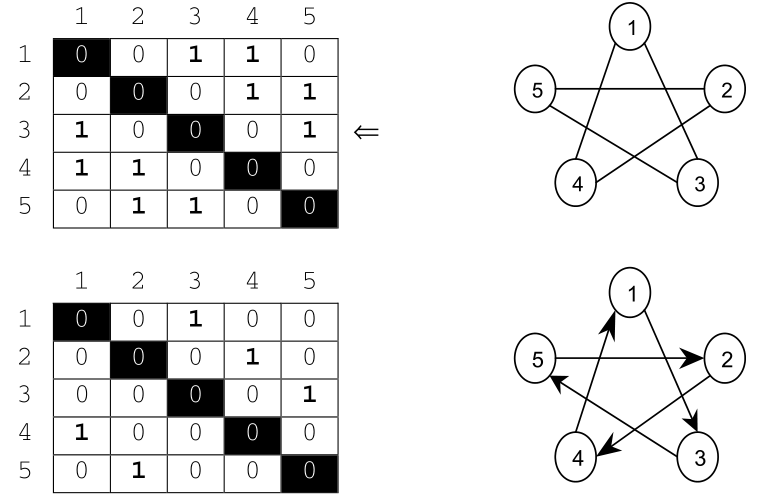
[V. Kết luận 18](#_Toc510630770)

[1. Nhận xét 19](#_Toc510630771)

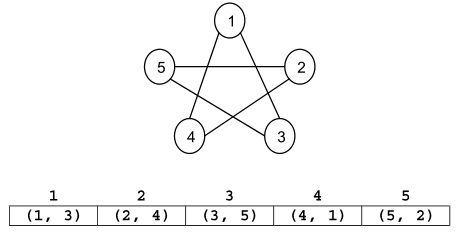
[2. Hướng phát triển 19](#_Toc510630772)

[3. Hạn chế - Khó khăn 19](#_Toc510630773)

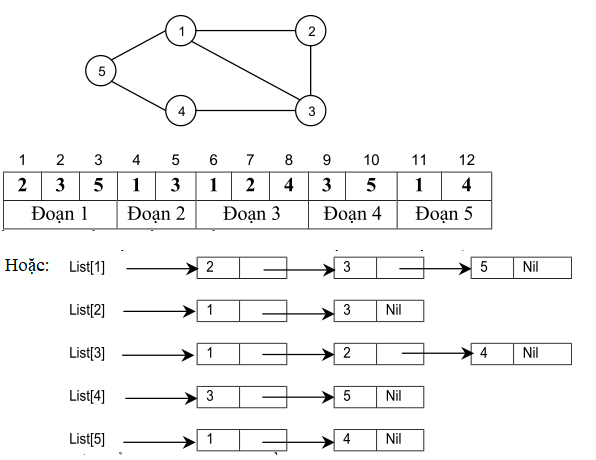
# Tổng quan về phương pháp

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)

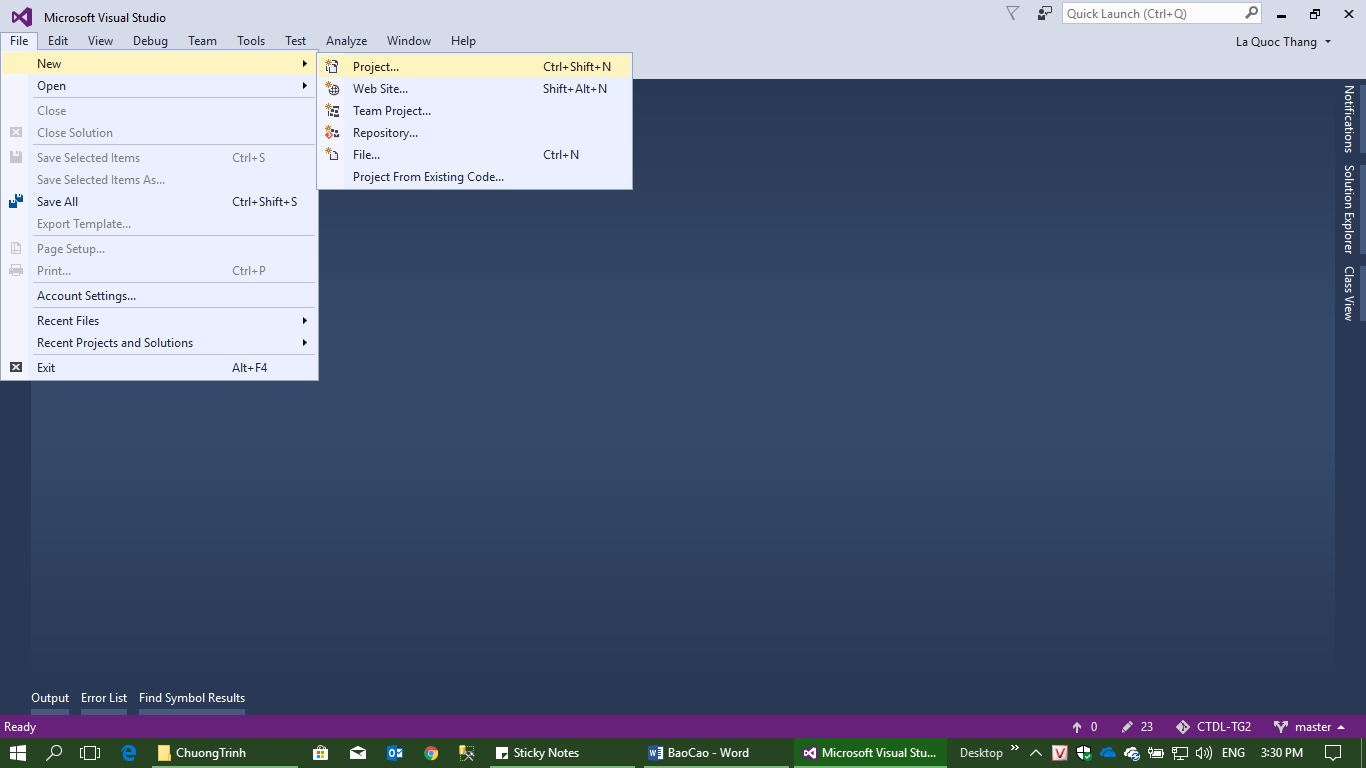
# Mục tiêu

Lưu trữ và biểu diễn đồ thị trên máy tính đồng thời thực hiện được 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.

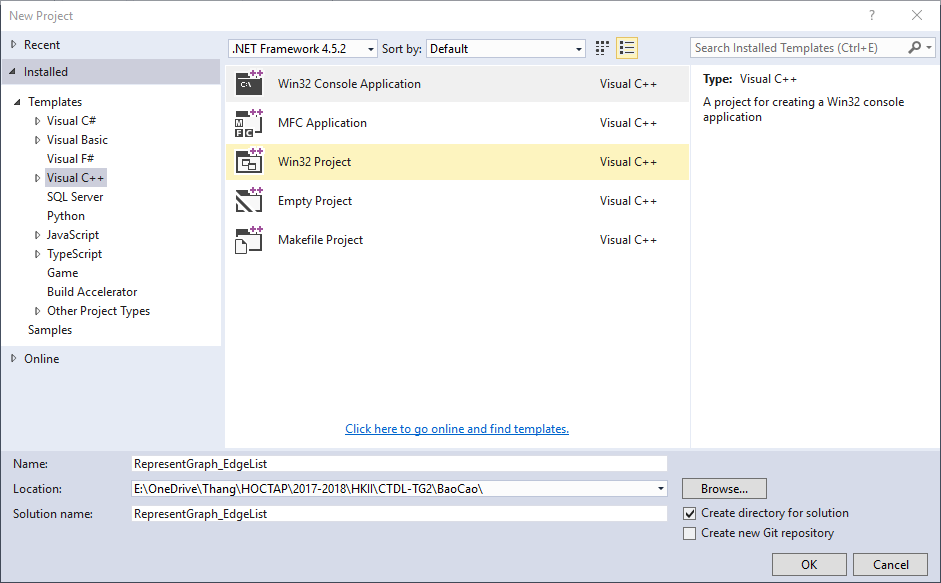
# Cài đặt

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



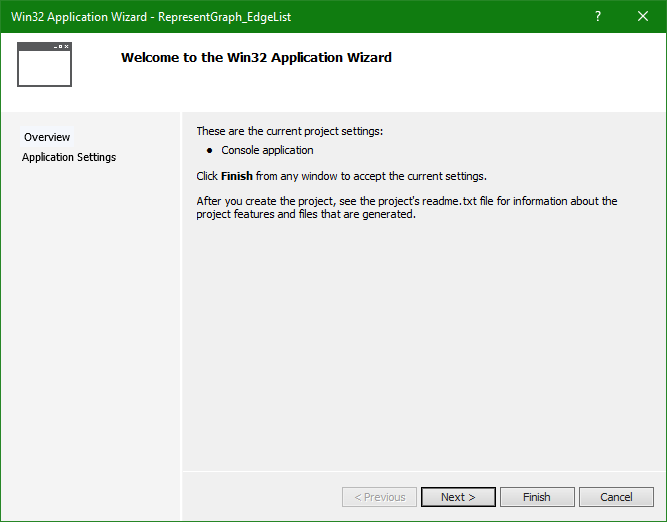
*Hình 4. Các bước tạo một project trong Visual Studio*

Tiếp theo chọn ngôn ngữ Visual C++ 🡪 chọn Win32 Console Application, đặt tên dự án là RepresentGraph\_EdgeList và chỉ định nơi lưu trữ 🡪 OK

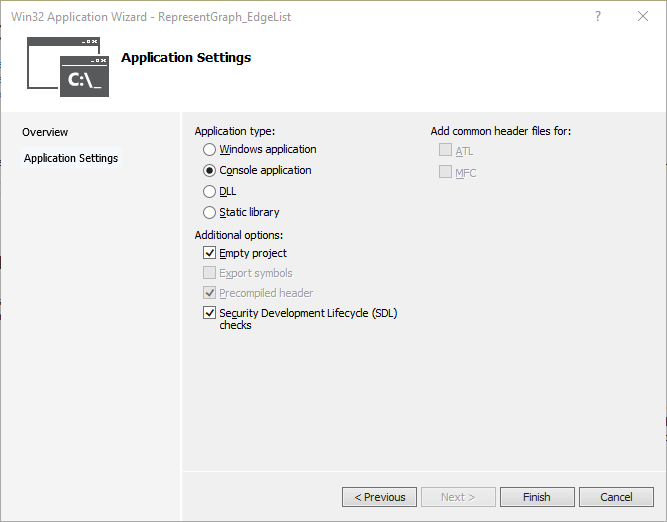


*Hình 5. Lựa chọn ngôn ngữ C++ và đặt tên cho project*

Thực hiện các thao tác khác để hoàn tất quá trình tạo một dự án Win32 Console Application



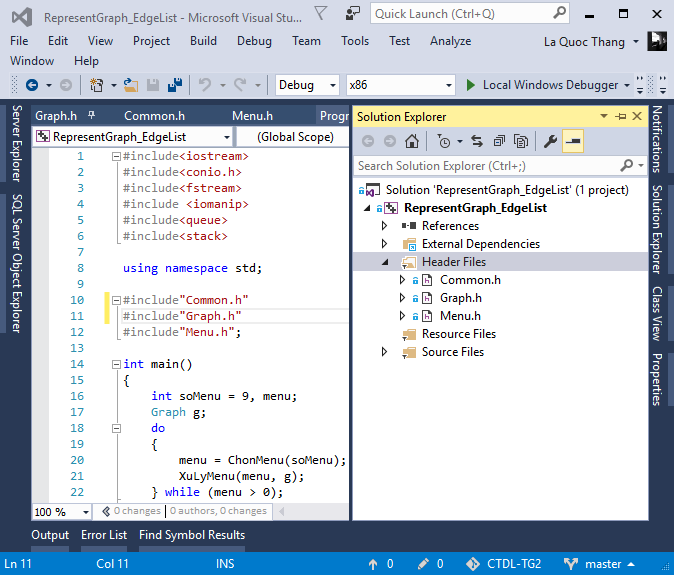
*Hình 6. Trình thuật sĩ tạo Win32 Application*



*Hình 7. Tùy chỉnh cài đặt cho project (tích vào Empty projects)*

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

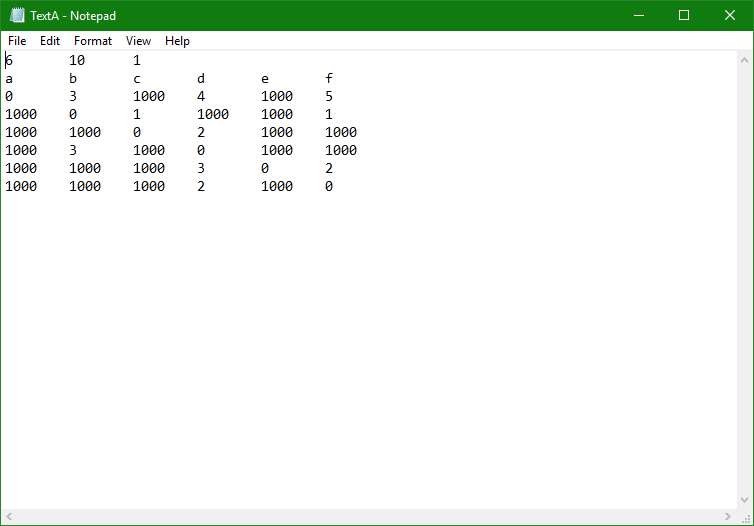
* Menu.h: Định nghĩa menu vận hành các chức năng của chương trình.
* Common.h: Định nghĩa các hằng số và kiểu dữ liệu của đồ thị.
* Graph.h: Định nghĩa các thao tác trên đồ thị.



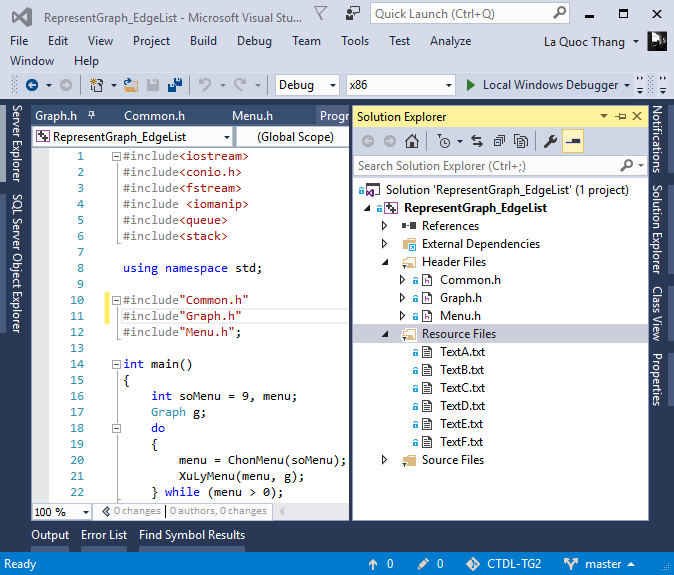
*Hình 6. Nội dung thư mục Header Files của project*

Thư mục Resource Files ta tạo 6 tập tin TextX.txt chứa dữ liệu về đồ thị có cấu trúc xác định như sau: (Mỗi thông tin phải phân biệt rõ ràng bằng khoảng trắng hoặc xuống dòng)

* Số đỉnh
* Số cạnh
* Loại đồ thị (0 : vô hướng/ 1 : có hướng)
* Ma trận kề

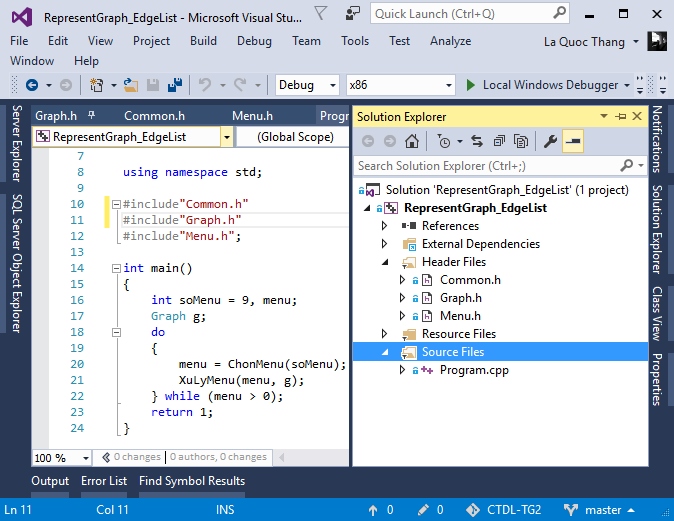


*Hình 7. Ví dụ về dữ liệu cho 1 đồ thị*



*Hình 8. Mỗi file text tương ứng với một đồ thị*

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.



*Hình 9. Thư mục Source Files của project*

Các bước tiến hành cài đặt:

1. Trong tập tin Common.h viết như sau:

* 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 = 10, 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

## Xuất danh sách cạnh của 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 danh sách cạnh

void DisplayEdges(Graph g)

{

Edge \*e = g.ListEdges;

if (e == NULL)

{

cout << "\nDanh sach rong!";

return;

}

DisplayTitle();

while (e != NULL)

{

Display1Edge(e->info);

e = e->Next;

}

cout << endl;

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

cout << '=';

cout << endl;

}

## 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)

{

//Thì thêm đỉnh vào đồ thị

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

//Và tăng số lượng đỉnh

g.NumVertices++;

}

}

## Xuất thông tin của một đỉnh trong đồ thị

void DisplayInfoVertex(Graph g, char d)

{

Edge \*e = g.ListEdges;

DisplayTitle();

while (e != NULL) //Duyệt trong danh sách cạnh

{

//Tìm những cạnh có chứa đỉnh d và hiển thị nó

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)

{

//Nếu trong danh sách tồn tại 1 cạnh như thế

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

return 1; //Thì trả về 1

e = e->Next;

}

return 0; //Ngược lại thì trả về 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

{

//Thì thêm cạnh đó vào

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

g.NumEdges++;

if (!directed) //Nếu nó là đồ thị vô hướng

{

//Thì thêm cạnh ngược của n

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

g.NumEdges++;

}

}

else //Còn nếu tồn tại trong danh sách rồi

{

Edge \*e = g.ListEdges;

while (e != NULL)

{

if (e->info.Source == start &&

e->info.Target == end)

//Thì thay trọng số mới vào

e->info.Weight = w;

e = e->Next;

}

}

}

//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 << '\n'; //Ghi số đỉnh

out << g.NumEdges << '\n'; //Ghi số cạnh

out << g.Directed << '\n'; //Ghi kiểu đồ thị

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

out << g.Vertices[i].Label << '\t';

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

{

out << endl;

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

{

int t = FindEdge(g, g.Vertices[i].Label, g.Vertices[j].Label);

if (t == NULLDATA)

if (i == j)

out << 0 << '\t';

else out << 1000 << '\t';

else out << t << '\t';

}

}

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, t;

char x, y;

char l;

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 (size\_t i = 0; i < a; i++) //Đọc tên đỉnh

{

in >> l;

AddVertex(g, l);

}

for (size\_t i = 0; i < a; i++) //Đọc ma trận

for (size\_t j = 0; j < a; j++)

{

in >> t;

if (t != 1000 && t != 0)

AddEdge(g, g.Vertices[i].Label, g.Vertices[j].Label, t);

}

in.close();

return 1;

}

else

return 0;

}

## Duyệt đồ thị theo chiều rộng

//Tìm đỉnh đầu tiên gằn với cur mà chưa xét

//Nếu thấy thì trả về tên đỉnh

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; //Còn không thấy thì trả về -1

}

//Duyệt đồ thị theo chiều rộng dùng vòng lặp

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 dùng vòng lặp

void DFS\_Loop(Graph g, char start)

{

ResetFlags(g);

int index = FindIndexVertex(g, start);

g.Vertices[index].Visited = YES;

cout << start << '\t';

stack<char> s;

s.push(start);

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);

}

}

}

## Duyệt đồ thị theo chiều sâu dùng đệ quy

void DFS\_Recursion(Graph &g, char start)

{

int vt = FindIndexVertex(g, start);

g.Vertices[vt].Visited = YES;

cout << start << '\t';

char adj;

while (true)

{

adj = FindFirstAdjacentVertex(g, start);

if (adj == NULLDATA)

break;

else

DFS\_Recursion(g, adj);

}

}

# Kết luận

## Nhận xét

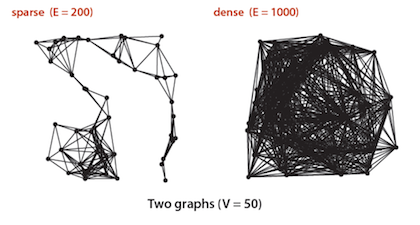
Phương pháp cài đặt và biểu diễn đồ thị bằng danh sách cạnh gặp không quá nhiều khó khăn bởi vì nó là một trong những phương pháp biểu diễn đơn giản nhất về đồ thị. Nhưng không tốt nếu biểu diễn cho một đồ thị có nhiều đỉnh |V| và nhiều cạnh |E|.

## Hướng phát triển

Dựa vào danh sách cạnh, ta có thể nâng lên thành một phương pháp biểu diễn đồ thị khác, đó là danh sách đỉnh kề. Trong phương pháp này, ta sẽ lưu trữ các đỉnh kề với đỉnh i bằng danh sách cạnh (danh sách cạnh này không có tên đỉnh đầu) và một mảng một chiều LIST lưu n đỉnh với LIST[i] là con trỏ trỏ tới danh sách các đỉnh kề với đỉnh i. Nếu như trong cách biểu diễn đồ thị bằng danh sách cạnh, để kiểm tra một cạnh có tồn tại hay không (tức là có cạnh nối giữa u và v) ta phải đi từ đầu danh sách xuống thì ở phương pháp này, ta có thể truy xuất nhanh các đỉnh kề của đỉnh i và duyệt danh sách cạnh với thao tác ít hơn.

## 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 (ví dụ như hàm FindFirstAdjacentVertex ). Điều đó khá tốn thời gian trong trường hợp đồ thị dày (nhiều cạnh).

*Hình 10. Đồ thị có 50 đỉnh nhưng có tới 200 cạnh và 1000 cạnh*

Hãy tưởng tượng phải duyệt qua một danh lớn để xem một cạnh có tồn tại hay không, vì danh sách các cạnh không tuân theo bất kỳ thứ tự cụ thể nào nên cạnh có thể ở cuối của danh sách! Hoặc có khi nó không hề tồn tại mà vẫn phải lặp lại toàn bộ danh sách để kiểm tra nó. Không chỉ công việc này sẽ làm mất thời gian tuyến tính với O(E) lần, trong đó E đại diện cho tất cả các cạnh của đồ thị - một danh sách cạnh cũng cần đòi hỏi không gian lưu trữ O(E), mà nó còn khá là hạn chế thao tác trên danh sách cạnh. Đây là một ví dụ minh họa cho phương pháp biểu diễn đơn giản nhất nhưng lại khá là hạn chế cho mục đích công việc.