CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

LAB 5 – ĐỒ THỊ VÀ BIỂU DIỄN ĐỒ THỊ (4 TIẾT)

I. Mục tiêu

Sau khi thực hành, sinh viên cần:

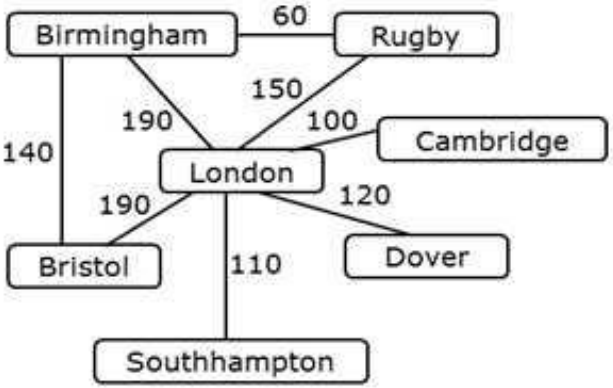
- Nắm vững định nghĩa đồ thị, các khái niệm liên quan đến đồ thị.
- Cài đặt được kiểu dữ liệu đồ thị và các thao tác, phép toán trên đồ thị được biểu diễn bởi ma trận kề.
- Vận dụng kiến thức đã học để giải một số bài toán thực tế.

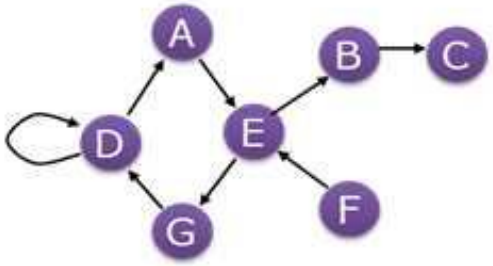
II. Yêu cầu

- Sinh viên phải hoàn thành ***4 bài*** thuộc mục V. Mỗi bài tập tạo một project, xóa các thư mục debug của project này. Sau đó chép các project vào thư mục: Lab5_CTK39_HoTen_MSSV_Nhom#. Nén thư mục , đặt tên tập tin nén theo dạng sau: Lab5_CTK39_HoTen_MSSV_Nhom#.rar.
  Ví dụ: Lab5_CTK39_NguyenVanA_141111_Nhom4.rar.
- Sinh viên sẽ nộp bài Lab qua mạng tại phòng lab theo hướng dẫn của giáo viên.

III. Ôn tập lý thuyết

1. Đồ thị và các định nghĩa





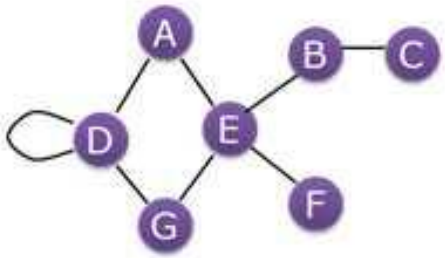
Một đồ thị G được định nghĩa là G = (V, E). Trong đó:

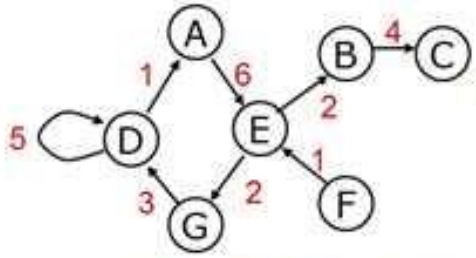
- V(G) là tập hữu hạn các đỉnh và khác rỗng. E(G) là tập các cung (cạnh) = Tập các cặp đỉnh (u, v) mà u, v ϵ V.
- Các đỉnh còn gọi là các nút (node) hay điểm (point)
- Mỗi cung nối giữa hai đỉnh v, w có thể ký hiệu là cặp (v,w). Hai đỉnh có thể trùng nhau. Nếu cặp (v,w) có thứ tự thì gọi là cung có thứ tự hay cạnh có hướng. Ngược lại ta nói là cung không có thứ tự hay cạnh vô hướng.

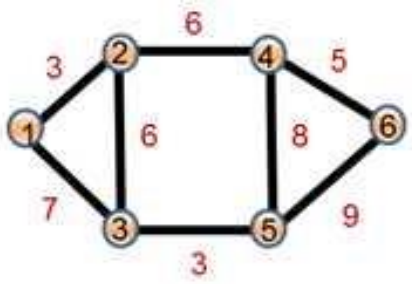
- Đỉnh kề: Hai đỉnh được gọi là kề nhau nếu chúng được nối với nhau bởi một cạnh. Trong trường hợp này, hai nút đỉnh được gọi là hàng xóm (neighbors) của nhau.
- Đường đi là một dãy tuần tự các đỉnh v_1, v_2, …, v_n sao cho (v_i, v_{i+1}) là một cung trên đồ thị. Đỉnh v_1 được gọi là đỉnh đầu, đỉnh v_n được gọi là đỉnh cuối. Độ dài đường đi là số cạnh trên đường đi.
- Đỉnh X gọi là có thể đi đến được từ đỉnh Y nếu tồn tại một đường đi từ đỉnh Y đến đỉnh X.
- Đường đi đơn là đường đi mà mọi đỉnh trên đó đều khác nhau, ngoại trừ đỉnh đầu và cuối có thể trùng nhau.
- Chu trình là đường đi có đỉnh đầu trùng với đỉnh cuối. Chu trình đơn là một đường đi đơn có đỉnh đầu và đỉnh cuối trùng nhau và có độ dài ít nhất là 1.
- Cấp của đồ thị là số đỉnh của đồ thị. Kích thước của đồ thị là số cạnh của đồ thị. Đồ thị rỗng là đồ thị có kích thước bằng 0
- Bậc của một đỉnh là số cạnh nối với đỉnh đó
- Trong nhiều ứng dụng, ta thường kết hợp các giá trị (value) hoặc nhãn (label) cho các cạnh hoặc đỉnh. Khi đó, ta gọi là đồ thị có trọng số.
- Nhãn có thể có kiểu tùy ý và dùng để biểu diễn tên, chi phí, khoảng cách, …
- Đồ thị có hướng là đồ thị mà các cạnh của nó là có hướng. Nghĩa là các cặp đỉnh (v,w) có phân biệt thứ tự.
- Đồ thị vô hướng là đồ thị mà các cặp đỉnh tương ứng với các cạnh của nó không phân biệt thứ tự.

2. *Biểu diễn đồ thị*

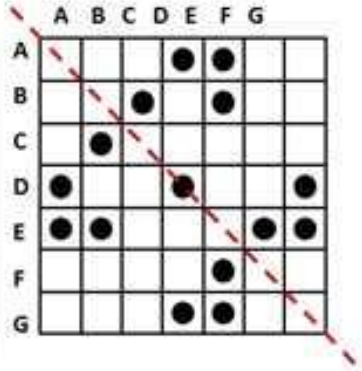
Biểu diễn đồ thị bằng ma trận kề







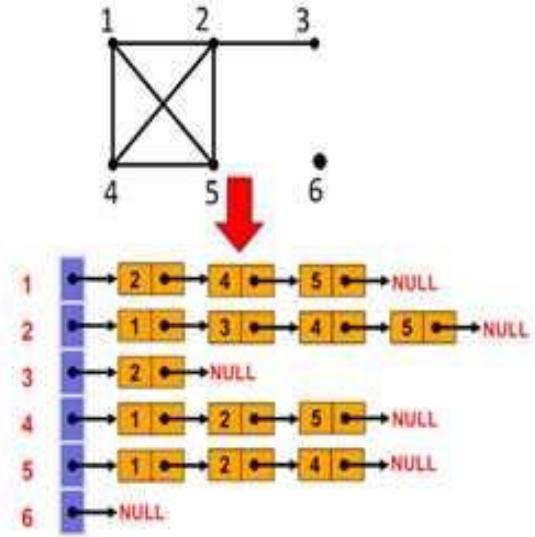
	A	B	C	D	E	F	G
A				1	1		
B			1		1		
C		1					
D	1			1			1
E	1	1				1	1
F					1		
G				1	1		

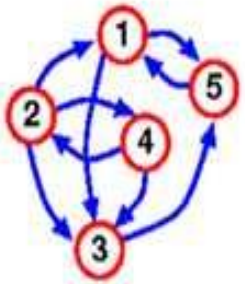


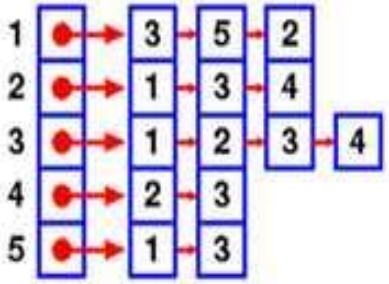
	A	B	C	D	E	F	G
A	0	0	0	0	6	0	0
B	0	0	4	0	0	0	0
C	0	0	0	0	0	0	0
D	1	0	0	5	0	0	0
E	0	2	0	0	0	0	2
F	0	0	0	0	1	0	0
G	0	0	0	3	0	0	0

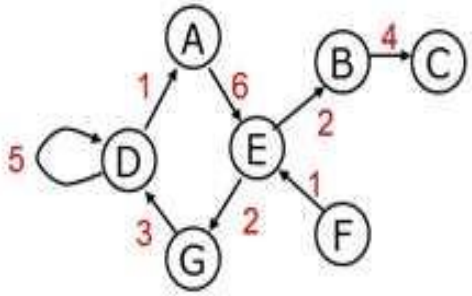
	1	2	3	4	5	6
1	0	3	7	0	0	0
2	3	0	6	6	0	0
3	7	6	0	0	3	0
4	0	6	0	0	8	5
5	0	0	3	8	0	9
6	0	0	0	5	9	0

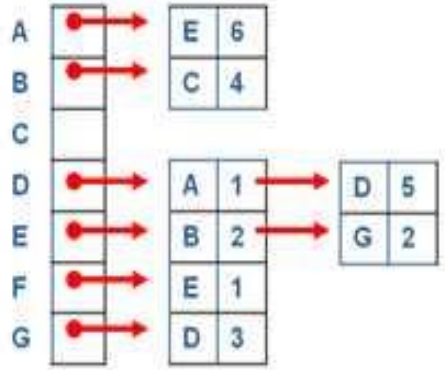
Biểu diễn bằng danh sách kề

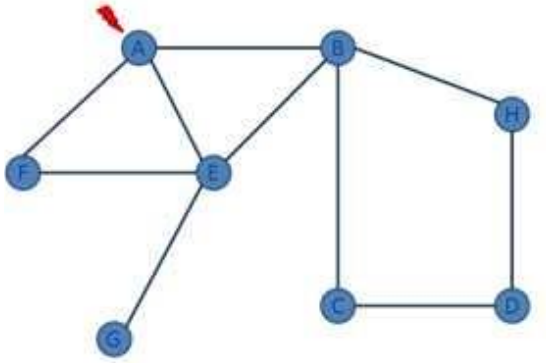




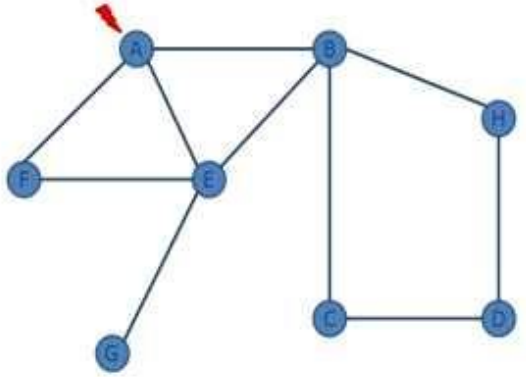








Kết quả: A B C D H E F G



Kết quả: A B E F C H G D

3. *Các phương pháp duyệt đồ thị*

Duyệt đồ thị là một thủ tục có hệ thống để khám phá đồ thị bằng cách kiểm tra tất cả các đỉnh và các cạnh của nó. Có hai cách duyệt đồ thị phổ biến

- Duyệt đồ thị theo chiều sâu (Depth First Search - DFS)
- Duyệt đồ thị theo chiều rộng (Breadth First Search -BFS)

Cách 1: Duyệt đồ thị theo chiều sâu

- B1. Khởi gán tất cả các đỉnh đều chưa được duyệt
- B2. Xuất phát từ một đỉnh **v** bất kỳ của đồ thị, đánh dấu đỉnh v đã được duyệt
- B3. Xử lý đỉnh v
- B4. Với mỗi đỉnh w chưa được duyệt và kề với v, ta thực hiện đệ quy quá trình trên cho w.

Cách 2: Duyệt đồ thị theo chiều rộng

- B1. Khởi gán tất cả các đỉnh đều chưa được duyệt
- B2. Xuất phát từ một đỉnh **v** bất kỳ của đồ thị, đánh dấu đỉnh v đã được duyệt. Đưa v vào hàng đợi.

- B3. Lấy x ra khỏi hàng đợi. Xử lý đỉnh x.
- B4. Với mỗi đỉnh w chưa được duyệt và kề với x, ta đánh dấu w đã được duyệt. Đưa w vào hàng đợi.
- B5. Quay lại B3.

IV. Hướng dẫn thực hành

1. Tạo dự án

Sinh viên có thể chọn cài đặt kiểu dữ liệu đồ thị theo ngôn ngữ C# hoặc C++. Các phần sau minh họa cách cài đặt, hướng dẫn bằng mã giả trên cả ngôn ngữ C# (bên trái) và C++ (bên phải).

- Tạo dự án mới, đặt tên là Lab3_GraphAsMatrix
- Tạo các lớp sau:
  - Matrix.cs: Định nghĩa kiểu dữ liệu ma trận
  - Vertex.cs: Định nghĩa một đỉnh của đồ thị
  - Edge.cs: Định nghĩa một cạnh
  - Path.cs: Định nghĩa một đoạn đường đi
  - Graph.cs: Định nghĩa kiểu dữ liệu đồ thị

- Tạo dự án mới, đặt tên là Lab3_GraphAsMatrix
- Trong thư mục Header Files, tạo ra 4 files sau:
  - common.h: Định nghĩa các hằng số và kiểu dữ liệu đồ thị
  - graph.h: Định nghĩa các thao tác trên đồ thị
  - stack.h: Định nghĩa kiểu dữ liệu Stack và các thao tác
  - queue.h: Định nghĩa kiểu dữ liệu Queue và các thao tác
- Trong thư mục Source Files, tạo tập tin: program.cpp

2. Định nghĩa kiểu dữ liệu đồ thị (Biểu diễn bằng ma trận kề)

- Nhấp đôi chuột vào file Matrix.cs, định nghĩa kiểu dữ liệu ma trận như sau:

```
class Matrix
{
        private int[,] _data;

        public int NumRows { get; set; }
        public int NumColumns { get; set; }

        public int this[int row, int co] {
                get { return _data[row, co]; }
                set { _data[row, co] = value; }
        }
        public Matrix(int size)
                : this(size, size) {
        }
        public Matrix(int numRows, int numCos) {
                this.NumRows = numRows;
                this.NumColumns = numCos;

                _data = new int[numRows, numCos];
                for (int i = 0; i < numRows; i++)
```

- Nhấp đôi chuột vào file common.h, định nghĩa các hằng số và kiểu dữ liệu như sau:

```
#ifndef _GRAPH_
#define _GRAPH_

// Định nghĩa hằng số
#define  UPPER          100     // Số ptử 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      // Giả giá trị rỗng

// ================================================
// Dùng cho kiểu dữ liệu đồ thị
// ================================================

// Định nghĩa các kiểu dữ liệu
typedef  char           LabelType;
typedef  int            CostType;
typedef  CostType       MaTrix[MAX][MAX]; // Ma trận
```

```
                for (int j = 0; j < numCos; j++)
                    _data[i, j] = 0;
        }
}
```

- Nhấp đôi chuột vào file Vertex.cs, định nghĩa kiểu dữ liệu Vertex (thể hiện một đỉnh của đồ thị) như sau:

```
class Vertex
{
        public bool Visited { get; set; }
        public string Label { get; set; }

        public Vertex()
                : this (string.Empty)
        { }
        public Vertex(string lab) {
                this.Label = lab;
                this.Visited = false;
        }
        public override string ToString() {
                return this.Label + "\t";
        }
}
```

- Nhấp đôi chuột vào file Edge.cs, định nghĩa kiểu dữ liệu Edge (thể hiện một cạnh của đồ thị) như sau:

```
class Edge
{
        public int Source { get; set; }
        public int Target { get; set; }
        public int Weight { get; set; }
        public bool Marked { get; set; }

        public Edge()
                : this(0, 0, 0)
        { }
        public Edge(int s, int t)
                : this(s, t, 1)
        { }
        public Edge(int s, int t, int w) {
                this.Source = s;
                this.Target = t;
                this.Weight = w;
                this.Marked = false;
        }
}
```

```
// Định nghĩa cấu trúc của một đỉnh
struct Vertex
{
        LabelType       Label;    // Nhãn của đỉnh
        int             Visited;  // Trạng thái
};

// Định nghĩa cấu trúc một cạnh
struct Edge
{
        int             Source;  // Đỉnh đầu
        int             Target;  // Đỉnh cuối
        CostType        Weight;  // Trọng số
        int             Marked; // Trạng thái
};

// Định nghĩa cấu trúc một đoạn đường đi
struct Path
{
        CostType        Length;  // Độ dài đđi
        int             Parent;  // Đỉnh trước
};

// Định nghĩa kiểu dữ liệu đồ thị
struct  Graph
{
        bool            Directed;               // ĐT có hướng?
        int             NumVertices;    // Số đỉnh
        int             NumEdges;               // Số cạnh
        MaTrix  Cost;                   // MTrận kề
        Vertex  Vertices[MAX];  // DS đỉnh
};

// ================================================
// Dùng cho Stack và Queue
// ================================================

// Kiểu dữ liệu phần tử của Queue, Stack
struct Entry
{
        // Dữ liệu chứa trong một nút của Queue
        int             Data;           // hoặc Stack
        Entry*  Next;           // Con trỏ Next
};

// Định nghĩa kiểu con trỏ tới một Entry
```

- Nhấp đôi chuột vào file Path.cs, định nghĩa kiểu dữ liệu Path (thể hiện một đoạn đường đi) như sau:

```
class Path
{
        public int Length { get; set; }
        public int Parent { get; set; }

        public Path() {     }
        public Path(int length, int parent)  {
                this.Length = length;
                this.Parent = parent;
        }
}
```

- Nhấp đôi chuột vào file Graph.cs, định nghĩa kiểu dữ liệu đồ thị (biểu diễn bằng ma trận kề) như sau:

```
class Graph
{
        public static readonly int MAX = 30;
        public static readonly int INF = 1000000;
        public static readonly int UPPER = 100;
        public static readonly int NOTFOUND = -1;

        public int NumVertices { get; set; }
        public int NumEdges { get; set; }

        public Matrix Cost { get; set; }
        public List<Vertex> Vertices { get; set; }

        public Graph()     {
        }
}
```

- Trong tập tin Program.cs, nhập đoạn mã sau:

```
class Program
{
        static void Main(string[] args) {
                Graph g = new Graph();
                Console.ReadLine();
        }
}
```

```
typedef Entry*    EntryPtr;

// Kiểu dữ liệu ngăn xếp (Stack)
typedef EntryPtr Stack;

// Tạo một phần tử của Stack chứa
// dữ liệu là data
EntryPtr CreateEntry(int data)
{
        EntryPtr item = new Entry;
        if (item)
        {
                item->Data = data;
                item->Next = NULL;
        }
        return item;
}

#endif
```

- Trong tập tin program.cpp, nhập đoạn mã sau:

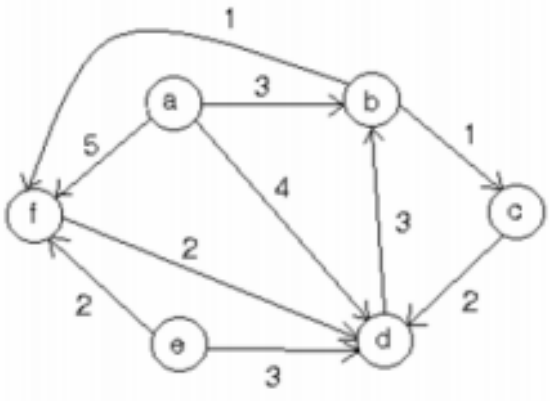
```
#include <iostream>
#include <conio.h>
#include <fstream>
using namespace std;

#include "common.h"
#include "queue.h"
#include "stack.h"
#include "graph.h"

void main()
{
        _getch();
}
```

Vào menu Build > chọn Build Solution để biên dịch chương trình, kiểm tra lỗi. Nếu có lỗi, kiểm tra lại mã nguồn bạn đã viết. Nếu chương trình không có lỗi, ta sẽ cài đặt các thao tác, phép toán trên đồ thị.

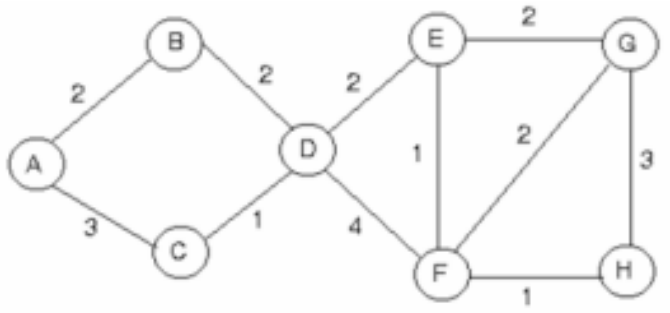
V. Bài tập thực hành (có HD)

Bài 1. Với mỗi đồ thị sau, hãy tạo các tập tin (sử dụng chương trình notepad, wordpad, …) lưu trữ chúng theo định dạng:

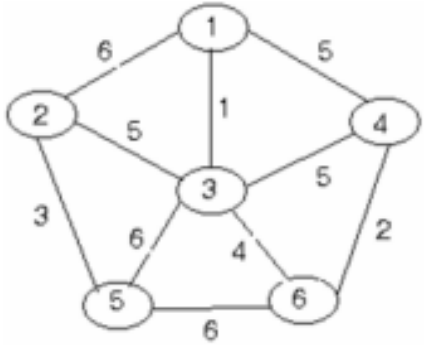
- Dòng đầu: Ghi số N (số đỉnh của đồ thị)
- Dòng thứ 2: Ghi số M (số cạnh của đồ thị)
- Dòng thứ 3: Ghi số 0 (nếu đồ thị vô hướng) hoặc 1 (nếu đồ thị có hướng)
- N dòng kế tiếp, mỗi dòng ghi nhãn của 1 đỉnh
- N dòng tiếp theo, mỗi dòng ghi N giá trị biểu diễn cho ma trận kề hay ma trận trọng số (gồm N dòng, N cột). Quy ước, nếu giữa hai đỉnh v và w có cạnh nối thì ghi giá trị trọng số của cạnh đó. Nếu không có cạnh nối thì ghi 0 (nếu v = w) hoặc 1000 (=INF nếu v khác w).



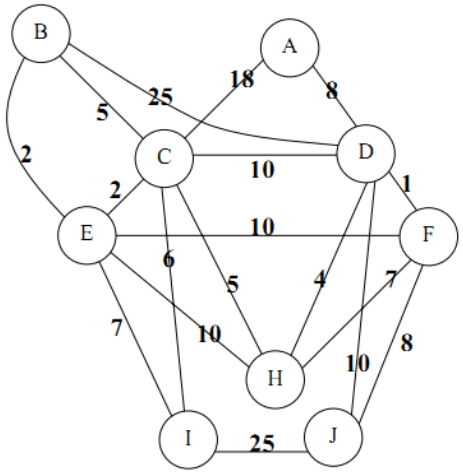
a)



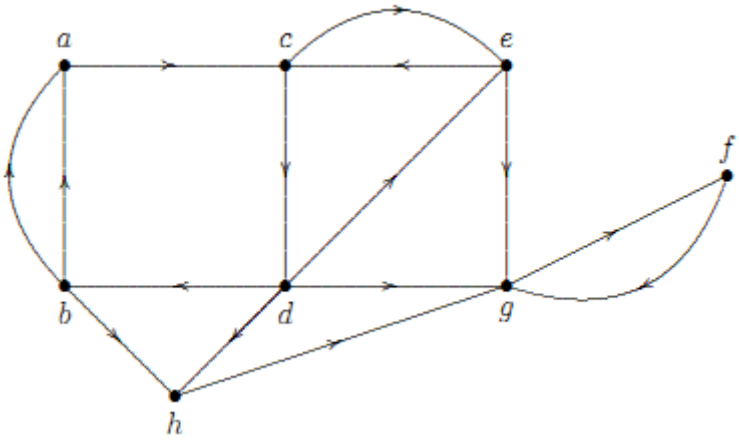
b)



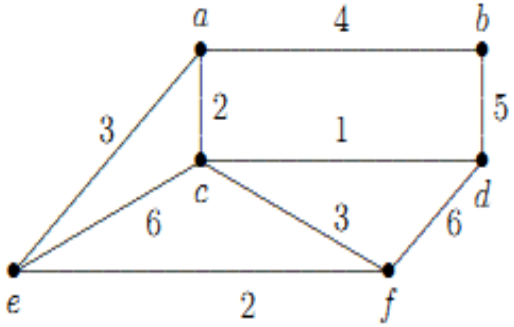
c)



d)



e)



f)

Bài 2. Cài đặt các thao tác cơ bản trên kiểu dữ liệu đồ thị – Viết hàm trong tập tin graph.h (C++) hoặc các lớp tương ứng (C#)

a. Tạo một đỉnh có nhãn lab (C++)

```
// Viết hàm này trong lớp Vertex
public Vertex(string lab)
{
        this.Label = lab;
        this.Visited = false;
}
// Khi cần, sử dụng lệnh sau để tạo 1 đỉnh
Vertex v = new Vertex("nhãn");
```

```
// Tạo một đỉnh có nhãn label
Vertex CreateVertex(LabelType lab)
{
        Vertex v;                       // Khai báo 1 biến kiểu Vertex
        v.Label = lab;      // Gán nhãn cho đỉnh
        v.Visited = NO;   // Cho biết đỉnh chưa xét
        return v;
}
```

b. Hiển thị thông tin của một đỉnh

Tạo ra một interface, đặt tên là Ivisitor như sau:

```
interface IVisitor
{
        void Visit(Vertex v);
}
// Ta sẽ tạo thêm các lớp thực thi interface này khi cần
```

```
// Hiển thị thông tin đỉnh thứ pos trong đồ thị
void DisplayVertex(Graph g, int pos)
{
        cout << g.Vertices[pos].Label << "\t";
}
```

c. Hàm khởi tạo một đồ thị

```
// Viết hàm này trong lớp Graph. Đây chính là phương thức tạo lập của lớp Graph
public Graph()
{
        this.NumEdges = 0;
        this.NumVertices = 0;
        this.Cost = new Matrix(MAX);
        this.Vertices = new List<Vertex>();

        for (int i = 1; i < MAX; i++)
        {
                for (int j = 0; j < i; j++)
                {
                        this.Cost[i, j] = INF;
                        this.Cost[j, i] = INF;
                }
        }
}
```

```
// Khởi tạo một đồ thị
// directed = true: Đồ thị có hướng
Graph InitGraph(bool directed)
{
        Khai báo và khởi tạo một biến g kiểu Graph;
        Khởi tạo số cạnh của g = 0;
        Khởi tạo số đỉnh của g = 0;
        Gán loại đồ thị = directed; // Có hướng hay ko

        // Khởi tạo ma trận kề (ma trận trọng số, chi phí)
        for (int i=0; i<MAX; i++)
                for (int j=0; j<MAX; j++)
                        if (i == j)                     // ĐĐi từ một đỉnh tới
                                g.Cost[i][j] = 0;   //chính nó thì =0
                        else
                                g.Cost[i][j] = INF;
        return g;
}
```

d. Hàm thiết lập lại trạng thái của các đỉnh trong đồ thị

```
public void ResetFlags()
{
        foreach (Vertex v in this.Vertices) {
```

```
// Thiết lập lại trạng thái của các đỉnh
void ResetFlags(Graph &g)
{
```

```
                v.Visited = false;
        }
}
```

```
        for (int i=0; i < Số đỉnh của g; i++)
                Thiết lập trường Visited của đỉnh i = NO;
}
```

e. Hàm kiểm tra 2 đỉnh có kề nhau hay không (2 đỉnh được nối với nhau bởi ít nhất một cạnh)

```
// Viết hàm trong lớp Graph
public bool IsConnected(int start, int end)
{
        if (this.Cost[start, end] == 0 ||
                this.Cost[start, end] == INF)
                return false;
        else
                return true;
}
```

```
// Kiểm tra hai đỉnh start và end có được nối với nhau bởi 1 cạnh hay không
int IsConnected(Graph g, int start, int end)
{
        if (Chi phí đi từ start đến end = 0 hoặc INF)
                Trả về 0;
        else
                Trả về 1;
}
```

f. Hàm thêm một đỉnh có nhãn lab vào đồ thị

```
// Viết hàm trong lớp Graph
public void AddVertex(string label)
{
        Vertex v = new Vertex(label);
        this.Vertices.Add(v);
        this.NumVertices++;
}
```

```
// Thêm một đỉnh có nhãn lab vào đồ thị g
void AddVertex(Graph &g, LabelType lab)
{
        Gọi hàm tạo một đỉnh v với nhãn lab;
        Đưa đỉnh v vào đồ thị g;
        Tăng số đỉnh của g lên 1;
}
```

g. Hàm thêm một cạnh nối giữa hai đỉnh của đồ thị

```
// Viết hàm trong lớp Graph
public void AddEdge(int start, int end, int weight)
{
        if (!IsConnected(start, end))
                this.NumEdges += 2;

        this.Cost[start, end] = weight;
        this.Cost[end, start] = weight;
}
public void AddEdge(int start, int end)
{
        AddEdge(start, end, 1);
}
private int FindVertex(string label)
{
        for (int i = 0; i < NumVertices; i++)
                if (Vertices[i].Label == label)
                        return i;
        return NOTFOUND;
```

```
// Thêm một cạnh có trọng số là weight bắt đầu
// từ đỉnh start và kết thúc tại đỉnh end
// directed=false: Thêm cạnh vô hướng
void AddEdge(Graph &g, int start, int end,
                        CostType weight, bool directed)
{
        if (Hai đỉnh start, end không kề nhau)
                Tăng số cạnh lên 1;

        Gán chi phí đi từ start đến end = weight;
        if (Nếu đồ thị là vô hướng)
                Gán chi phí đi từ end đến start = weight;
}

// Thêm một cạnh có trọng số là weight bắt đầu
// từ đỉnh start và kết thúc tại đỉnh end
void AddEdge(Graph &g, int start, int end, CostType weight)
{
        AddEdge(g, start, end, weight, g.Directed);
```

```
}
public void AddEdge(string start, string end)
{
	int si = FindVertex(start);
	int ei = FindVertex(end);
	if (si == NOTFOUND || ei == NOTFOUND)
		throw new VertexNotFoundException();
	AddEdge(si, ei);
}
```

h. *Hàm lưu đồ thị xuống file*

```
// Viết hàm trong lớp Graph
public void SaveGraph(string path)
{
	FileStream fs = new FileStream(path,
		FileMode.OpenOrCreate, FileAccess.Write);
	StreamWriter writer = new StreamWriter(fs);

	writer.WriteLine(this.NumVertices);
	writer.WriteLine(this.NumEdges);

	foreach (Vertex v in this.Vertices)
	{
		writer.WriteLine(v.Label);
	}
	for (int i = 0; i < this.NumVertices; i++)
	{
		for (int j = 0; j < this.NumVertices; j++)
			writer.Write("{0}\t", this.Cost[i, j]);
		writer.WriteLine();
	}
	writer.Close();
	fs.Close();
}
```

i. *Hàm tạo đồ thị từ dữ liệu được lưu trong file*

```
// Viết hàm trong lớp Graph
}

// Thêm một cạnh bắt đầu từ đỉnh start và kết thúc tại
// đỉnh end. Dùng cho đồ thị không có trọng số.
void AddEdge(Graph &g, int start, int end)
{
	AddEdge(g, start, end, 1);
}
```

```
// Lưu đồ thị xuống file
void SaveGraph(Graph g, char* fileName)
{
	// Khai báo biến và mở tập tin để ghi
	ofstream os(fileName);

	// Lưu số đỉnh
	os << g.NumVertices << '\n';

	// Lưu số cạnh
	os << g.NumEdges << '\n';

	// Lưu loại đồ thị
	os << g.Directed << '\n';

	// Lưu tên các đỉnh
	for (int i=0; i<g.NumVertices; i++)
		os << g.Vertices[i].Label << '\n';

	// Lưu ma trận kề
	for (int i=0; i<g.NumVertices; i++)
	{
		for (int j=0; j<g.NumVertices; j++)
		{
			os << g.Cost[i][j] << '\t';
		}
		os << '\n';
	}
	// Đóng tập tin
	os.close();
}
```

```
// Đọc dữ liệu từ file để tạo đồ thị
```

```
public void OpenGraph(string path)
{
        FileStream fs = new FileStream(path,
                FileMode.Open, FileAccess.Write);
        StreamReader reader = new StreamReader(fs);

        this.NumVertices = int.Parse(reader.ReadLine());
        this.NumEdges = int.Parse(reader.ReadLine());

        for (int i=0; i<this.NumVertices; i++)
        {
                AddVertex(reader.ReadLine());
        }

        for (int i = 0; i < this.NumVertices; i++)
        {
                string[] parts = reader.ReadLine().Split(
                                new string[] { "\t" },
                StringSplitOptions.RemoveEmptyEntries);

                for (int j = 0; j < this.NumVertices; j++)
                        this.Cost[i, j] = int.Parse(parts[j]);
        }
        reader.Close();
        fs.Close();
}
```

```
void OpenGraph(Graph &g, char* fileName)
{
        // Khai báo biến và mở file để đọc
        ifstream is(fileName);
        // kiểm tra đã mở được file chưa?
        if (is.is_open())
        {
                int n = 0, m = 0;
                bool d = false;
                LabelType lab;

                is >> n;  // Đọc số đỉnh của đồ thị
                is >> m; // Đọc số cạnh
                is >> d;  // Đọc loại đồ thị

                g = InitGraph(d);               // Khởi tạo đồ thị
                g.NumEdges = m;                 // Gán số cạnh

                // Khởi tạo nhãn của các đỉnh
                for (int i=0; i<n; i++)
                {
                        is >> lab;                              // Đọc nhãn
                        AddVertex(g, lab);          // Thêm đỉnh
                }

                // Đọc ma trận kề từ file
                for (int i=0; i<n; i++)
                {
                        for (int j=0; j<n; j++)
                        {
                                // Và lưu vào đồ thị
                                is >> g.Cost[i][j];
                        }
                }
                is.close();                             // Đóng file
        }
}
```

j. Hàm tìm đỉnh đầu tiên chưa được xét và kề với đỉnh hiện tại (curr)

```
// Viết hàm trong lớp Graph
private int FindFirstAdjacencyVertex(int curr)
{
        for (int i = 0; i < NumVertices; i++)
        {
          if (!Vertices[i].Visited && IsConnected(curr, i))
```

```
// Tìm đỉnh đầu tiên kề với curr mà chưa xét
int FindFirstAdjacentVertex(Graph g, int curr)
{
        // Duyệt qua mọi đỉnh
        for (int i=0; i < Số đỉnh của g; i++)
        {
                // Kiểm tra đỉnh đã xét chưa và kề với curr ko?
```

```
                return i;
        }
        return NOTFOUND;
}
```

Bài 3. Các phương pháp duyệt đồ thị.

a. Hàm duyệt đồ thị theo chiều sâu (dạng đệ quy)

```
// Viết hàm trong lớp Graph
public void DFSRecursion(int start, IVisitor visitor)
{
        Vertices[start].Visited = true;
        visitor.Visit(Vertices[start]);

        for (int adj = 0; adj < NumVertices; adj++)
                if (IsConnected(start, adj) &&
                        !Vertices[adj].Visited)
                {
                        DFSRecursion(adj, visitor);
                }
}
```

b. Hàm duyệt đồ thị theo chiều sâu (dạng lặp)

```
// Viết hàm trong lớp Graph
public void DFSLoop(int start, IVisitor visitor)
{
        Vertices[start].Visited = true;
        visitor.Visit(Vertices[start]);

        Stack<int> s = new Stack<int>();
        s.Push(start);

        int curr, adj;
        while (s.Count > 0)
        {
                curr = s.Peek();
                adj = FindFirstAdjacencyVertex(curr);
                if (adj == NOTFOUND)
                        s.Pop();
                else
                {
                        Vertices[adj].Visited = true;
```

```
                if (Đỉnh i chưa được xét và
                        Có cạnh nối từ curr đến i))
                        return i;           // Thỏa đk -> tìm thấy
        }
        return NULLDATA;                    // Không tìm thấy
}
```

```
// Duyệt đồ thị theo chiều sâu dạng đệ quy
void DFS_Recursion(Graph &g, int start)
{
        Đánh dấu đỉnh start đã xét : gán Visited = YES;
        Xuất thông tin đỉnh start;

        while (true)
        {
                Tìm đỉnh adj đầu tiên kề với start và chưa xét;
                if (Không tìm thấy đỉnh adj như vậy)
                        break;              // thì dừng, quay lui
                else
                        Gọi đệ quy, duyệt từ đỉnh adj;
        }
}
```

```
// Duyệt đồ thị theo chiều sâu (Depth First Search)
// Dạng lặp, sử dụng Stack
void DFS_Loop(Graph g, int start)
{
        Đánh dấu đỉnh start đã xét;
        Xuất thông tin đỉnh start;

        Khởi tạo Stack s;
        Đưa đỉnh start vào Stack s, start;

        int curr, adj;                  // curr : Đỉnh đang xét
                                        // adj  : Đỉnh kề với curr
        while (Stack s khác rỗng)
        {
                Xem phần tử đầu tiên curr từ Stack s;

                Tìm đỉnh adj đầu tiên kề với curr và chưa xét;
                if (Không tìm thấy đỉnh adj như vậy)
                        Pop(s);   // Loại bỏ 1 đỉnh trong Stack
```

```
                        visitor.Visit(Vertices[adj]);
                        s.Push(adj);
                }
        }
}
```

c. Hàm duyệt đồ thị theo chiều rộng

```
// Viết hàm trong lớp Graph
public void BFS(int start, IVisitor visitor)
{
        Vertices[start].Visited = true;
        Queue<int> q = new Queue<int>();
        q.Enqueue(start);

        int curr, adj;
        while (q.Count > 0)
        {
                curr = q.Dequeue();
                visitor.Visit(Vertices[curr]);
                for (adj = 0; adj < NumVertices; adj++)
                        if (IsConnected(curr, adj) &&
                                !Vertices[adj].Visited)
                        {
                                Vertices[adj].Visited = true;
                                q.Enqueue(adj);
                        }
        }
}
```

Bài 4. Kiểm tra chương trình và xem kết quả

Tạo thêm lớp NameVisitor, thực thi interface Visitor như sau:

```
class NameVisitor : IVisitor
{
        public void Visit(Vertex v)
        {
                Console.Write("{0}\t", v);
        }
}
```

```
                else                    // nhằm quay lại đỉnh trước
                {
                        Đánh dấu đỉnh adj đã xét,gán Visited=YES;
                        Hiển thị thông tin đỉnh adj;
                        Đưa đỉnh adj vào Stack s;
                }
        }
}
```

```
// Duyệt đồ thị theo chiều rộng
void BFS(Graph g, int start)
{
        Đánh dấu đỉnh start đã được xét;
        Khởi tạo hàng đợi q;
        Đưa đỉnh start vào hàng đợi q;

        int curr, adj;
        while (Hàng đợi q khác rỗng)        // Còn đỉnh chưa xét?
        {
                Lấy đỉnh đầu tiên curr từ hàng đợi;
                Xuất thông tin đỉnh curr;
                while (true)
                {
                        Tìm đỉnh adj kề với curr và chưa xét;
                        if (Không tim thấy adj như vậy) break;
                        else
                        {
                                Đánh dấu đỉnh adj đã được xét;
                                Đưa đỉnh adj vào hàng đợi;
                        }
                }
        }
}
```

```
#include <iostream>
#include <fstream>
#include <conio.h>

using namespace std;

#include "common.h"
#include "stack.h"
#include "queue.h"
```

Trong tập tin Program.cs, cập nhật lại đoạn mã sau:

```
class Program
{
        static void Main(string[] args)
        {
                Graph g = new Graph();
                g.AddVertex("A"); // 0
                g.AddVertex("B"); // 1
                g.AddVertex("C"); // 2
                g.AddVertex("D"); // 3
                g.AddVertex("E"); // 4
                g.AddVertex("F"); // 5
                g.AddVertex("G"); // 6
                g.AddVertex("H"); // 7

                // Nếu đồ thị không có trọng số
                //g.AddEdge(0, 1);          // AB
                //g.AddEdge(0, 4);          // AE
                //g.AddEdge(0, 5);          // AF
                //g.AddEdge(1, 2);          // BC
                //g.AddEdge(1, 4);          // BE
                //g.AddEdge(1, 7);          // BH
                //g.AddEdge(2, 3);          // CD
                //g.AddEdge(3, 7);          // DH
                //g.AddEdge(4, 5);          // EF

                // Sử dụng nhãn của đỉnh thay vì chỉ số
                //g.AddEdge("A", "B");
                //g.AddEdge("A", "E");
                //g.AddEdge("A", "F");
                //g.AddEdge("B", "C");
                //g.AddEdge("B", "E");
                //g.AddEdge("B", "H");
                //g.AddEdge("C", "D");
                //g.AddEdge("D", "H");
                //g.AddEdge("E", "F");


                g.AddEdge(0, 1, 4);         // AB 4
                g.AddEdge(0, 4, 6);         // AE 6
                g.AddEdge(0, 5, 2);         // AF 2
                g.AddEdge(1, 2, 1);         // BC 1
                g.AddEdge(1, 4, 1);         // BE 1
                g.AddEdge(1, 7, 5);         // BH 5
```

```
#include "graph.h"

void main()
{
        Graph g = InitGraph(false);

        //Graph g;
        //OpenGraph(g, "graph1.txt");

        /*
        // Nếu LabelType = int
        AddVertex(g, 0);
        AddVertex(g, 1);
        AddVertex(g, 2);
        AddVertex(g, 3);
        AddVertex(g, 4);
        AddVertex(g, 5);
        AddVertex(g, 6);
        AddVertex(g, 7);*/

        // Nếu LabelType = char
        AddVertex(g, 'A');          // 0
        AddVertex(g, 'B');          // 1
        AddVertex(g, 'C');          // 2
        AddVertex(g, 'D');          // 3
        AddVertex(g, 'E');          // 4
        AddVertex(g, 'F');          // 5
        AddVertex(g, 'G');          // 6
        AddVertex(g, 'H');          // 7

        // Nếu đồ thị không có trọng số
        //AddEdge(g, 0, 1);         // AB
        //AddEdge(g, 0, 4);         // AE
        //AddEdge(g, 0, 5);         // AF
        //AddEdge(g, 1, 2);         // BC
        //AddEdge(g, 1, 4);         // BE
        //AddEdge(g, 1, 7);         // BH
        //AddEdge(g, 2, 3);         // CD
        //AddEdge(g, 3, 7);         // DH
        //AddEdge(g, 4, 5);         // EF
        //AddEdge(g, 4, 6);         // EG

        // Nếu đồ thị có trọng số
        AddEdge(g, 0, 1, 4);        // AB 4
        AddEdge(g, 0, 4, 6);        // AE 6
        AddEdge(g, 0, 5, 2);        // AF 2
        AddEdge(g, 1, 2, 1);        // BC 1
```

```
            g.AddEdge(2, 3, 2);         // CD 2
            g.AddEdge(3, 7, 1);         // DH 1
            g.AddEdge(4, 5, 2);         // EF 2
            g.AddEdge(4, 6, 3);         // EG 3

            Console.WriteLine("So dinh : ", g.NumVertices);
            Console.WriteLine("So canh : ", g.NumVertices);

            IVisitor visitor = new NameVisitor();

            Console.WriteLine("Ket qua duyet chieu sau");
            g.ResetFlags();
            g.DFSLoop(0, visitor);
            //g.DFSRecursion(0, visitor);

            Console.WriteLine();
            Console.WriteLine("Ket qua duyet chieu rong");
            g.ResetFlags();
            g.BFS(0, visitor);

            Console.ReadLine();
        }
    }
```

```
    AddEdge(g, 1, 4, 1);        // BE 1
    AddEdge(g, 1, 7, 5);        // BH 5
    AddEdge(g, 2, 3, 2);        // CD 2
    AddEdge(g, 3, 7, 1);        // DH 1
    AddEdge(g, 4, 5, 2);        // EF 2
    AddEdge(g, 4, 6, 3);        // EG 3


    cout << endl << "So dinh cua do thi "
         << g.NumVertices << endl;
    cout << endl << "So canh cua do thi "
         << g.NumEdges << endl;
    //SaveGraph(g, "graph1.txt");

    cout << endl << endl << "======================";
    cout << endl << "Ket qua duyet theo chieu sau"
         << endl;
    ResetFlags(g);
    //DFS_Loop(g, 3);
    DFS_Recursion(g, 3);

    cout << endl << endl << "======================";
    cout << endl << "Ket qua duyet theo chieu rong"
         << endl;
    ResetFlags(g);
    BFS(g, 3);

    _getch();
}
```

VI. Bài tập

Bài 1. Tính liên thông của đồ thị

Cho một đồ thị G=(V,E). Viết chương trình:

- Kiểm tra xem đồ thị G có liên thông hay không?
- Nếu không, đếm số thành phần liên thông của đồ thị G.
- Liệt kê các thành phần (đồ thị con) liên thông của đồ thị G

Gợi ý:

- Dùng 1 trong các phương pháp duyệt DFS hoặc BFS: Mỗi khi không thể đi đến một đỉnh khác, ta tăng số thành phần liên thông lên 1. Tìm một đỉnh khác chưa được xét và duyệt tiếp.
- Hoặc sử dụng thuật toán tìm cây bao trùm (Xem Lab 4): Đồ thị G là liên thông khi và chỉ khi G có cây bao trùm.

Bài 2. Đồ thị biểu diễn bằng danh sách kề.

- Định nghĩa kiểu dữ liệu đồ thị (biểu diễn bằng danh sách kề)
- Viết hàm khởi tạo đồ thị một đồ thị
- Thêm một đỉnh có nhãn label vào đồ thị
- Xuất thông tin của một đỉnh trong đồ thị
- Kiểm tra hai đỉnh u, v có kề nhau (có cạnh nối giữa chúng) hay không?
- Thêm một cạnh có trọng số w nối 2 đỉnh u, v vào đồ thị
- Lưu thông tin đồ thị xuống file
- Tạo đồ thị dữ liệu được lấy từ file
- Duyệt đồ thị theo chiều rộng
- Duyệt đồ thị theo chiều sâu (dạng lặp và đệ quy).

Bài 3. Đồ thị biểu diễn bằng danh sách cạnh

- Tìm hiểu cách biểu diễn đồ thị bằng danh sách cạnh (Soạn slide thuyết trình)
- Cài đặt chương trình theo yêu cầu tương tự như bài tập 2 ở trên.

Bài 4. Đồ thị biểu diễn bằng ma trận liên thuộc

- Tìm hiểu cách biểu diễn đồ thị bằng ma trận liên thuộc (Soạn slide thuyết trình)
- Cài đặt chương trình theo yêu cầu tương tự như bài tập 2 ở trên.

VII. Phụ lục

Stack.h

```
// Tạo một Stack rỗng
Stack CreateStack()
{
	return NULL;
}

// Kiểm tra Stack có rỗng hay không?
bool IsEmpty(Stack s)
{
	return (s == NULL);
}
```

Queue.h

```
// Kiểu dữ liệu hàng đợi (Queue)
struct Queue
{
	EntryPtr Head;
	EntryPtr Tail;
};

Queue CreateQueue()
{
	Queue q;
	q.Head = q.Tail = NULL;
	return q;
```

```
// Đưa một phần tử vào stack
void Push(Stack &s, int data)
{
	EntryPtr node = CreateEntry(data);
	if (IsEmpty(s))
		s = node;
	else
	{
		node->Next = s;
		s = node;
	}
}

// Lấy phần tử ở đỉnh Stack
int Pop(Stack &s)
{
	if (IsEmpty(s))
		return NULLDATA;
	else
	{
		EntryPtr node = s;
		int item = node->Data;
		s = s->Next;

		delete node;
		return item;
	}
}

// Xem phần tử ở đỉnh Stack
int Top(Stack s)
{
	if (IsEmpty(s))
		return NULLDATA;
	else
		return s->Data;
}
```

```
}

bool IsEmpty(Queue q)
{
	return (q.Head == NULL);
}

void Enqueue(Queue &q, int data)
{
	EntryPtr node = CreateEntry(data);
	if (IsEmpty(q))
	{
		q.Head = q.Tail = node;
	}
	else
	{
		q.Tail->Next = node;
		q.Tail = node;
	}
}

int Dequeue(Queue &q)
{
	if (IsEmpty(q))
		return NULL;
	else
	{
		EntryPtr node = q.Head;
		q.Head = q.Head->Next;
		if (q.Head == NULL)
			q.Tail = NULL;
		int data = node->Data;
		delete node;
		return data;
	}
}

int First(Queue &q)
{
	if (IsEmpty(q))
		return NULL;
	else
		return q.Head->Data;
}
```

=== HẾT ===