Bài thực hành số 12. ĐƯỜNG ĐI VÀ TÍNH LIÊN THÔNG

Mục tiêu: hiểu và sử dụng được thuật toán tìm đường đi, kiểm tra tính liên thông, thành phần liên thông với đồ thị tổ chức bằng danh sách kề.

Nội dung thực hành:

Bài 1. Đường đi và Liên thông

Tìm đường đi trên đồ thị biểu diễn bằng danh sách kề sử dụng thuật toán duyệt theo chiều rộng.

Kiểm tra đồ thị vô hướng liên thông. Xác định các thành phần liên thông của đồ thị vô hướng.

Các thư viện:

```
#include <iostream>
#include <fstream>
#include <queue>
using namespace std;
```

Tổ chức dữ liệu:

```
#define MAX_VERTICES 100
#define INF 1000   //infinity

struct AdjList
{
  int adjVertex;
  float weight;
  AdjList* next;
};
struct AdjListGraph
{
  int numVertices;
  AdjList* adjList[MAX_VERTICES];
};
```

a) Đọc đồ thị từ tệp

```
void readGraphFromFile(AdjListGraph& graph, string fileName)
```

```
{
    ifstream file(fileName);
    file >> graph.numVertices;

    for (int i = 0; i < graph.numVertices; i++)
    {
    graph.adjList[i] = nullptr;
        for (int j = 0; j < graph.numVertices; j++)
        {
            float weight;
            file >> weight;
            if (weight != INF)
            {
                AdjList* newNode = new AdjList;
                newNode->adjVertex = j;
                newNode->weight = weight;
                newNode->next = graph.adjList[i];
                graph.adjList[i] = newNode;
            }
        }
    }

    file.close();
}
```

```
Graph3.txt
4
1000 1 1000 1
1 1000 1 1000
1000 1 1000 1
1 1000 1 1000
```

b) In đồ thị

```
void printGraph(AdjListGraph graph)
{
    cout << "Num of vertices: " << graph.numVertices << endl;
  for (int i = 0; i < graph.numVertices; i++)
    {
        cout << "adjList[" << i << "]: ";
        for (AdjList* p = graph.adjList[i]; p != nullptr; p = p->next)
        {
            cout << p->adjVertex << "(" << p->weight << ") ";
        }
```

```
        cout << endl;
    }
    cout << endl;
}
```

c) Tìm đường đi xuất phát từ một đỉnh bằng thuật toán duyệt đồ thị theo chiều rộng

```
/*  finds the path from a source vertex 's' to all other vertices in an
AdjListGraph using Breadth-First Search (BFS) algorithm */
void PathBFS(AdjListGraph graph, int s, int parent[])
{
    bool visited[MAX_VERTICES] = { false };

    for(int i = 0; i<graph.numVertices; i++) parent[i] = -1;
    queue<int> q;
    visited[s] = true;
    parent[s] = s;
    q.push(s);
    while (!q.empty())
    {
        int curr = q.front();
        q.pop();
        for (AdjList* p = graph.adjList[curr]; p!= nullptr; p=p->next)
        {
            if (!visited[p->adjVertex])
            {
                visited[p->adjVertex] = true;
                parent[p->adjVertex] = curr;
                q.push(p->adjVertex);
            }
        }
    }
}
```

d) In đường đi:

```
//Print a path from s to v
void printPath(int parent[],int s, int v)
{
```

```
    if (v == s)
    {
    cout << s << " ";
    return;
  }
    printPath(parent, s, parent[v]);
    cout << v << " ";
}
```

e) Kiểm tra đồ thị vô hướng liên thông

```
//Traverse a graph using Depth-First Search (DFS) algorithm
void DFS(AdjListGraph graph, int v, bool visited[])
{
    visited[v] = true;
    for (AdjList* p = graph.adjList[v]; p != NULL; p = p->next)
    {
        if (!visited[p->adjVertex])
            DFS(graph, p->adjVertex, visited);
    }
}
// checks an undirected graph is connected
bool isConnected(AdjListGraph graph)
{
    bool visited[MAX_VERTICES] = { false };
    DFS(graph, 0, visited);
    for(int i = 0; i < graph.numVertices; i++)
    if (!visited[i])
        return false;
    return true;
}
```

f) Xác định các thành phần liên thông của đồ thị vô hướng

```
// DFS for identify connected components
void DFSConnectedComponents(AdjListGraph graph, int v, bool visited[],
int component[], int k)
{
    visited[v] = true; component[v] = k;
```

```
    for (AdjList* p = graph.adjList[v]; p != NULL; p = p->next)
    {
        if (!visited[p->adjVertex])
            DFSConnectedComponents(graph,    p->adjVertex,    visited,
component, k);
    }
}

//identify connected components of a graph
int connectedComponents(AdjListGraph graph, int component[])
{
    bool visited[MAX_VERTICES] = { false };

    int k = 0; //num of connected components
    for (int i = 0; i < graph.numVertices; i++)
    {
        if (!visited[i])
        {
            k++;
            DFSConnectedComponents(graph, i, visited, component, k);
        }
    }
    return k;
}
```

g) In các thành phần liên thông:

```
// print connected components
void printConnectedComponents(int component[], int n, int k)
{
  cout << "Num of connected components is: " << k << endl;;
  for(int i = 1; i <= k; i++)
  {
    cout << "Component[" << i << "]: ";
    for(int j = 0; j < n; j++)
      if (component[j]==i)
        cout << j << " ";
```

```
    cout << endl;
  }
  cout << endl;
}
```

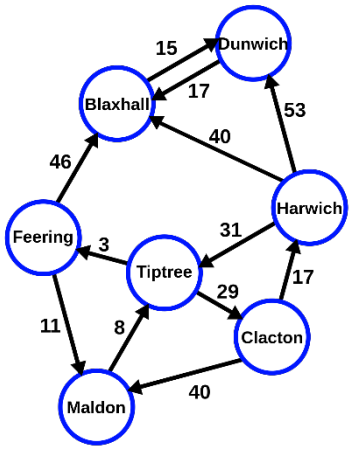
Hàm main

```
int main()
{
  int main()
{
  AdjListGraph g;
  readGraphFromFile(g, "Graph3.txt");
  printGraph(g);

  int parent[MAX_VERTICES];
  int s, d;
  cout << "Input start vertex: "; cin >> s;
  cout << "Input destination vertex: "; cin >> d;
  PathBFS(g, s, parent);
  cout << "A path from " << s << " to " << d << " is: ";
  printPath(parent, s, d);
  cout << endl;
  bool connected = isConnected(g);
  if (connected)
    cout << "Graph is connected." << endl;
  else
    cout << "Graph is unconnected." << endl;

  int comp[MAX_VERTICES] = {-1};
  int k = connectedComponents(g, comp);
  printConnectedComponents(comp, g.numVertices, k);
  return 0;
}
```

Thực hành với chương trình:

Tạo tệp Graph4.txt cho đồ thị như hình 1. Tìm đường đi từ Blaxhall đến các đỉnh còn lại.

Tạo tệp Graph5.txt cho đồ thị biểu diễn bằng ma trận kề như hình 2. Kiểm tra đồ thị có liên thông không? Cho biết số thành phần liên thông và từng thành phần liên thông của đồ thị.



Hình 1

```
10
0 1 0 0 1 0 0 0 0 0
1 0 1 0 0 0 0 0 0 0
0 1 0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
1 0 0 0 0 1 0 0 0 0
0 0 0 0 1 0 1 1 0 0
0 0 0 0 0 1 0 0 1 0
0 0 0 0 0 1 0 0 0 1
0 0 0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 1 1 0
```

Hình 2

Viết các hàm thực hiện:

h) Sửa lại mã lệnh sao cho kiểm tra được có hay không có đường đi giữa hai đỉnh. Sử dụng trong hàm main thông báo khi không có đường đi.

i) Cài đặt thuật toán tìm đường đi bằng duyệt theo chiều sâu. So sánh kết quả với đường đi bằng duyệt theo chiều rộng trên đồ thị Graph4.txt.

```
void pathDFS(AdjListGraph graph, int s, int visited[], int parent[])
{



}
```

j) Tìm đường đi xuất phát từ đỉnh s đến đỉnh d không qua đỉnh v.

Hàm trả về vector chứa đường đi từ s đến d không qua v. Nếu vector rỗng thì không có đường đi thỏa điều kiện.

```
vector<int> pathNoVertex(AdjListGraph graph, int s, int d, int v)
{



```

```
}
```

k) Tìm đường đi xuất phát từ đỉnh s và chỉ qua các cạnh có trọng số >=M.

```
vector<int> pathGreatThan(AdjListGraph graph, int s, int d, int M)
{




}
```

l) Kiểm tra đồ thị có hướng liên thông mạnh không?

```
bool strongConnected(AdjListGraph graph)
{




}
```

m) Tìm tất cả các đường đi đơn từ đỉnh s đến đỉnh d.

```
void allPaths(AdjListGraph graph, int s, int d, vector<vector<int>>
paths)
{









}
```

Bài 2. Sử dụng vector cho danh sách kề.

Ngoài việc sử dụng danh sách kề như cách khai báo ở trên, có thể sử dụng vector để biểu diễn đồ thị danh sách kề như sau:

Khai báo các phần tử trong danh sách kề như các cạnh:

```
struct Edge{
  int adjVertex;
  float weight;
};
```

Khi đó một đồ thị là vector có số phần tử là số đỉnh của đồ thị, mỗi phần tử là một danh sách kề được biểu diễn như một vector các Edge:

```
vector<vector<Edge>> graph;
```

Hãy cài đặt các thao tác của Bài 1 với cách tổ chức dữ liệu này.

Bài 3. Đồ thị biết số điện thoại

Trong đồ thị biết số điện thoại, hãy viết các hàm thực hiện:

a) Cho biết sinh viên a có thể biết gián tiếp số điện thoại sinh viên b không? Nếu có thì sinh viên a phải hỏi số điện thoại ít nhất bao nhiêu sinh viên và theo thứ tự nào?
b) Cho biết N sinh viên có thỏa tính chất hai sinh viên bất kỳ có thể biết số điện thoại của nhau qua trực tiếp hay gián tiếp không?