Bài thực hành số 11. DUYỆT ĐỒ THỊ

Mục tiêu: hiểu được cách lập trình với thao tác duyệt đồ thị trên đồ thị biểu diễn bằng ma trận kề. Một số thao tác đơn giản trên đồ thị để làm quen với mô hình và cấu trúc dữ liệu này.

Nội dung thực hành:

Bài 1. Duyệt đồ thị

Chương trình sau minh họa thao tác duyệt đồ thị (in các đỉnh được thăm) trên đồ thị biểu diễn bằng ma trận kề với dữ liệu được đọc từ tệp văn bản.

Các thư viện:

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

Khai báo kiểu Đồ thị biểu diễn bằng ma trận kề

```
#define MAX_VERTICES 100
struct AdjMatrixGraph
{
  int numVertices;
  int adjMatrix[MAX_VERTICES][MAX_VERTICES];
};
```

a) Hàm đọc tệp dữ liệu đồ thị:

Cấu trúc tệp văn bản có dạng:

Dòng đầu là số đỉnh

Các dòng tiếp theo là ma trận kề của đồ thị

Graph1.txt

```
4
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0
```

```
void readGraphFromFile(string fileName, AdjMatrixGraph &graph)
{
    // Open the file for reading
    ifstream file(fileName);

  // Read the number of vertices from the file
  file >> graph.numVertices;
```

```
    // Read adjacency matrix from file
    for (int i = 0; i < graph.numVertices; i++)
    {
        for (int j = 0; j < graph.numVertices; j++)
        {
            file >> graph.adjMatrix[i][j];
        }
    }

    // Close the file
    file.close();
}
```

b) Hàm in đồ thị biểu diễn bằng ma trận kề:

```
void printGraph(AdjMatrixGraph graph)
{
  cout << "Num of vertices: " << graph.numVertices << endl;
    for (int i = 0; i < graph.numVertices; i++)
    {
        for (int j = 0; j < graph.numVertices; j++)
        {
            cout << graph.adjMatrix[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}
```

c) Hàm duyệt đồ thị theo chiều sâu:

```
void DFS(const AdjMatrixGraph& graph, int vertex, bool visited[])
{
    // Mark the current vertex as visited
    visited[vertex] = true;
    cout << vertex << " ";

    // Recur for all the vertices adjacent to this vertex
    for (int i = 0; i < graph.numVertices; i++)
```

```
    {
        if (graph.adjMatrix[vertex][i] != 0 && !visited[i])
        {
            DFS(graph, i, visited);
        }
    }
}
```

Hàm main:

```
int main()
{
  AdjMatrixGraph g;
  bool visited[MAX_VERTICES];
  memset(visited, false, sizeof(visited));
  readGraphFromFile("Graph1.txt", g);
  printGraph(g);
  DFS(g, 0, visited);
  return 0;
}
```

Thực hành các yêu cầu sau:

d) Tạo tệp Graph2.txt là đồ thị có trọng số, hãy sửa chương trình để duyệt đồ thị này xuất phát từ 1 đỉnh nào đó.

```

```

e) Viết hàm duyệt hết tất cả các đỉnh của đồ thị với đồ thị không liên thông.

Gợi ý: sử dụng DFS duyệt xuất phát từ những đỉnh chưa được thăm.

```
void traverseGraph(AdjMatrixGraph g)
{

```

```
}
```

f) Cài đặt thuật toán duyệt đồ thị theo chiều rộng xuất phát từ đỉnh v.

Hướng dẫn:

Thuật toán:
+ Thăm đỉnh v
+ Thăm lần lượt những đỉnh kề của v mà chưa thăm
+ Với mỗi đỉnh vừa thăm, thăm lần lượt các đỉnh kề với nó mà chưa thăm
+ Lặp lại bước trên cho đến khi không thăm thêm được đỉnh nào

Cài đặt: dùng 1 hàng đợi lưu các đỉnh trong quá trình duyệt.

```
void BFS(const AdjMatrixGraph& graph, int vertex)
{
    // Create a queue for BFS
    queue<int> queue;

    // Create a boolean array to track visited vertices
    bool visited[MAX_VERTICES];
    memset(visited, false, sizeof(visited));

    // Mark the current vertex as visited and enqueue it
    visited[vertex] = true;
    queue.push(vertex);

    // Iterate while the queue is not empty
    while (!queue.empty())
    {
        // Dequeue a vertex from queue and print it
        int currVertex = queue.front();
        cout << currVertex << " ";
        queue.pop();

        /* Get all adjacent vertices of the dequeued vertex
           If an adjacent vertex has not been visited,
           mark it visited and enqueue it */
        for (int i = 0; i < graph.numVertices; i++)
```

```
        {
            if (graph.adjMatrix[currVertex][i] != 0 && !visited[i])
            {
                visited[i] = true;
                queue.push(i);
            }
        }
    }
}
```

g) Viết hàm đếm số cạnh của đồ thị vô hướng.

```
int countEdges(AdjMatrixGraph g)
{




}
```

h) Viết hàm tính số cạnh vào đỉnh x với đồ thị có hướng, không có trọng số.

```
int countIncomingEdges(AdjMatrixGraph g, int x)
{




}
```

i) Viết hàm liệt kê những đỉnh không kề với đỉnh x.

```
void notAdjacent(AdjMatrixGraph g, int x)
{




}
```

j) Viết hàm liệt kê những đỉnh không thăm được khi duyệt xuất phát từ đỉnh x.

```
void notVisited(AdjMatrixGraph g, int x)
{
```

```
}
```

k) Kiểm tra một dãy các đỉnh có là đường đi của đồ thị không?

```
bool isPath(AdjMatrixGraph g, int arr[], int k)
{



}
```

l) Liệt kê những đỉnh đến được đỉnh x (đồ thị có hướng).

```
void pathToVertex(AdjMatrixGraph g, int x)
{



}
```

Bài 2. Đồ thị biết số điện thoại

Trong một lớp học có N sinh viên được đánh số từ 0 đến N-1, trong đó mỗi sinh viên biết số điện thoại của một số sinh viên trong lớp (gọi là biết trực tiếp). Sinh viên a không biết trực tiếp số điện thoại của sinh viên c nhưng biết trực tiếp số điện thoại của sinh viên b và sinh viên b biết trực tiếp số điện thoại của sinh viên c thì sinh viên a gọi là biết gián tiếp số điện thoại của sinh viên c. Cho trước thông tin biết số điện thoại của N sinh viên. Hãy viết các hàm thực hiện:

a) Cho biết sinh viên x biết trực tiếp số điện thoại sinh viên y không?
b) Cho biết sinh viên x biết trực tiếp hoặc gián tiếp số điện thoại của sinh viên z không?
c) Cho biết sinh viên x có thể biết được số điện thoại hết cả lớp không?
d) Cho biết những sinh viên mà không ai biết số điện thoại.
e) Cho biết có sinh viên nào biết số điện thoại hết cả lớp không?
f) Tìm tập hợp ít sinh viên nhất mà những sinh viên này biết hết số điện thoại của cả lớp.

```

```