上海大学实验报告

专业：计算机科学
姓名：颜乐春
学号：15124542
日期：2017-10-09
地点：704

课程名称：算法设计於分析　指导老师：神人　成绩：59
实验名称：棋盘覆盖问题　实验类型：Null　同组学生姓名：None

一、 问题描述於实验目的

目前网络上电子地图的使用很普遍，如百度地图、高德地图等。利用电子地图可以很方便地确定从一个地点到另一个地点的最短路径。

电子地图可以看成是一个图，而公交线路图可看成是带权有向图 $G = (V, E)$，其中每条边的权是非负实数。

你的任务：对给定的一个（无向）图 G，及 G 中的两点 s t，确定一条从 s 到 t 的最短路径。

1. 输入

输入有若干组测试数据（组数不超过 20）。

每组测试数据的第 1 行是一个正整数 n，表示地图 G 的顶点数，$n < 50$。

接下来的 n 行每行 n 个数，采用矩阵方式描述这一个地图。第 i 行的 n 个数，依次表示第 i 个顶点与第 1 2 3 … n 个顶点的路径长。假如两个顶点间无边相连，用一个 -1 表示。相邻的两个整数之间用空格隔开。注意，图中每个顶点 i 处都没有自己到自己的边。

再在下面的一行上给出两个整数 s t，表示上述地图上的两个顶点。

每组测试数据输入结束后空一行。直到输入结束。

2. 输出

对输入中的每个城市地图，先在一行上输出 “Case Num”，其中 “Num” 是测试数据的组号（从 1 开始），再在下一行输出从顶点 s 到顶点 t 的最短距离，格式如 **“The least distance from 1->5 is 60”** ，参见输出样例。

在接下来的一行上输出从顶点 s 到顶点 t 的最短路径上的顶点序列，格式如 **“the path is 1->4->3->5”** 。

3. 输入样例

```
5
-1 10 -1 30 100
-1 -1 50 -1 -1
-1 -1 -1 -1 10
-1 -1 20 -1 60
-1 -1 -1 -1 -1
```

```
1 5

6
-1 1 12 -1 -1 -1
-1 -1 9 3 -1 -1
-1 -1 -1 -1 5 -1
-1 -1 4 -1 13 13
-1 -1 -1 -1 -1 4
-1 -1 -1 -1 -1 -1
1 6
```

4. 输出样列

```
Case 1
The least distance from 1->5 is 60
the path is 1->4->3->5
Case 2
The least distance from 1->6 is 17
the path is 1->2->4->6
```

二、 实验环境

Ubuntu 17.04 + gcc 6.3

三、 实验内容和步骤

1. 设计思路

这个算法是通过为每个顶点 v 保留目前为止所找到的从 s 到 v 的最短路径来工作的。初始时，原点 s 的路径权重被赋为 0 $d[s] = 0$ 。若对于顶点 s 存在能直接到达的边 s, m ，则把 $d[m]$ 设为 $w\ s, m$ ，同时把所有其他（s 不能直接到达的）顶点的路径长度设为无穷大，即表示我们不知道任何通向这些顶点的路径（对于所有顶点的集合 V 中的任意顶点 v，若 v 不为 s 和上述 m 之一，$d[v] = \inf$）。当算法结束时，$d[v]$ 中存储的便是从 s 到 v 的最短路径，或者如果路径不存在的话是无穷大。

2. 算法描述

Algorithm 1 Dijkstra's Algorithm

```
function DIJKSTRA(Graph, source)
    create vertex set Q
    for vertex v in Graph do
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        addvtoQ
     dist[source] ← 0
    while Q is not empty do
        u ← vertexinQwithmindist[u]
        removeufromQ
        for neighbor v of u do alt ← dist[u] + length(u, v)
            if alt < dist[v] then dist[v] ← alt prev[v] ← u
    return dist prev
```

四、 实现程序

```
// Program to find Dijkstra's shortest path using
// priority_queue in STL
#include<bits/stdc++.h>
#include<cstdio>
using namespace std;
# define INF 0x3f3f3f3f

// iPair ==> Integer Pair
typedef pair<int, int> iPair;

// This class represents a directed graph using
// adjacency list representation
class Graph
{
    int V;   // No. of vertices

    // In a weighted graph, we need to store vertex
    // and weight pair for every edge
    list< pair<int, int> > *adj;

public:
    Graph(int V); // Constructor

    // function to add an edge to graph
    void addEdge(int u, int v, int w);
```

```
    // prints shortest path from s
    int shortestPath(int s, int d);
};

// Allocates memory for adjacency list
Graph::Graph(int V)
{
    this->V = V;
    adj = new list<iPair> [V];
}

void Graph::addEdge(int u, int v, int w)
{
    adj[u].push_back(make_pair(v, w));
    adj[v].push_back(make_pair(u, w));
}

// Prints shortest paths from src to all other vertices
int Graph::shortestPath(int src, int des)
{
    // Create a priority queue to store vertices that
    // are being preprocessed. This is weird syntax in C++.
    // Refer below link for details of this syntax
    // http://geeksquiz.com/implement-min-heap-using-stl/
    priority_queue< iPair, vector <iPair> , greater<iPair> > pq;

    // Create a vector for distances and initialize all
    // distances as infinite (INF)
    vector<int> dist(V, INF);

    // Insert source itself in priority queue and initialize
    // its distance as 0.
    pq.push(make_pair(0, src));
    dist[src] = 0;

    /* Looping till priority queue becomes empty (or all
      distances are not finalized) */
    while (!pq.empty())
    {
        // The first vertex in pair is the minimum distance
        // vertex, extract it from priority queue.
        // vertex label is stored in second of pair (it
        // has to be done this way to keep the vertices
        // sorted distance (distance must be first item
        // in pair)
        int u = pq.top().second;
        pq.pop();

        // 'i' is used to get all adjacent vertices of a vertex
        list< pair<int, int> >::iterator i;
```

```
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
        {
            // Get vertex label and weight of current adjacent
            // of u.
            int v = (*i).first;
            int weight = (*i).second;

            //  If there is shorted path to v through u.
            if (dist[v] > dist[u] + weight)
            {
                // Updating distance of v
                dist[v] = dist[u] + weight;
                pq.push(make_pair(dist[v], v));
            }
        }
    }

    // Print shortest distances stored in dist[]
    //printf("Vertex Distance from Source\n");
    //for (int i = 0; i < V; ++i)
        //printf("%d \t\t %d\n", i, dist[i]);
  return dist[des];
}


// Driver program to test methods of graph class
int main()
{
    // create the graph given in above fugure

  int N;
  int tmp;
  int src, des;
  int counter = 0;
  while(scanf("%d", &N)!=EOF)
  {
    counter += 1;
    Graph g(N);
    for(int i=0; i < N; i++)
      for(int j=0; j < N; j++)
      {
        scanf("%d", &tmp);
        if (tmp!=-1)
        {
          g.addEdge(i, j, tmp);
          g.addEdge(j, i, tmp);
        }

      }
    scanf("%d %d", &src, &des);
```

```
    printf("Case %d\n", counter);
    printf("%d\n", g.shortestPath(des-1, src-1));
  }

    return 0;
}
```