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

序列 $Z =< B\ C\ D\ B >$ 是序列 $X =< A\ B\ C\ B\ D\ A\ B >$ 的子序列，相应的递增下标序列为 $< 2\ 3\ 5\ 7 >$。

一般地，给定一个序列 $X =< x_1, x_2, \cdots, x_m >$，则另一个序列 $Z =< z_1, z_2, \cdots, z_k >$ 是 X 的子序列，是指存在一个严格递增的下标序列 $i1, i2, ..., ik$ 使得对于所有 $j = 1, 2, ..., k$ 使 Z 中第 j 个元素 z_j 与 X 中第 i_j 个元素相同。给定 2 个序列 X 和 Y，当另一序列 Z 既是 X 的子序列又是 Y 的子序列时，称 Z 是序列 X 和 Y 的公共子序列。

你的任务是：给定 2 个序列 $X\ Y$，求 X 和 Y 的最长公共子序列 Z。

1. 输入

输入文件中的第 1 行是一个正整数 $T, (0 < T \leqslant 10)$，表示有 T 组测试数据。接下来是每组测试数据的描述，每组测试数据有 3 行。

测试数据的第 1 行有 2 个正整数 $m\ n$，中间用一个空格隔开，$(0 < m\ n < 50)$；第 2 3 行是长度分别为 $m\ n$ 的 2 个序列 X 和 Y，每个序列的元素间用一个空格隔开。序列中每个元素由字母、数字等构成。

输入直到文件结束。

2. 输出

对输入中的每组测试数据，输出 2 行。先在一行上输出 “Case Num”，其中 “Num” 是测试数据的组号（从 1 开始），再在第 2 行上输出这 2 个序列 $X\ Y$ 的最长公共子序列 Z 的长度及子序列 Z（至少一个）。

3. 输入样例

```
2
7 6
A B C B D A B
B D C A B A
8 9
b a a b a b a b
a b a b b a b b a
```

4. 输出样列

```
Case 1
4 LCS(X,Y):B C B A
Case 2
6 LCS(X,Y):a b a b a b
```

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;
}
```