

实验五

1. 问题描述与实验目的:

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。。

2. 要点分析和算法描述:

要计算从 s 到 t 的最短路径，用 $dist[N]$ ，记录从 s 到所有点的最短路径， $dist[t]$ 即为答案。使用结构体 $edge\{v,w\}$ 保存边的终点和权值。借助 $vector$ ，采用邻接表存储图。

初始时将 $dist[s]$ 设为 0，从 s 点出发，遍历 s 点的邻接点 i ，如果 $dist[i]>dist[s]+w$ ，则将 $dist[i]$ 更新为较小值。之后从未被访问的点中，找出 $dist$ 值最小的值，用该值进行下一轮的邻接点的更新。

对于多个最短路径，输出字典序最小的。容器 $path$ 中存储了从 s 到 t 的最短路径，可以使用一个 cmp 比较函数，遍历两个 $path$ 对象 a,b 中的各项，如果 a 字典序大于 b ，返回 $true$ 。

该算法更新 n 次 $dist$ 数组值，每次遍历一个点中所有邻接点，时间复杂度在 $O(n^2)$

3. 代码实现:

```
#include <vector>
#include <iostream>
using namespace std;
#define inf 99999
const int N = 1e6 + 5;
// Dijkstra 算法求无向图任意两点最短路径

// 边结构体
struct edge
{
    int v, w;
};

int dist[N];           // dist[i]表示从源点s到i的最短距离
bool st[N];            // st[i]表示i是否已经被访问
vector<int> path[N];    // path[i]存放从源点s到i的最优路径
vector<edge> g[N];      // 图存储, g[i] = {v,w}, 存储i的所有邻接点v和权值w
int n, s, t;           // n表示点数, s表示源点, t表示终点

// 初始化, 将dist全部置为inf, st全部置为0
void init()
{
```

```

    for (int i = 0; i <= n; i++)
    {
        dist[i] = inf; // 初始化为无穷大
        st[i] = 0;
        path[i].clear();
        path[i].push_back(i); // 路径初始化为自己, 即 i->i
        g[i].clear();
    }
}

// 添加边, u->v 的权值为w
void add(int u, int v, int w)
{
    g[u].push_back({v, w});
}

// 比较两个路径的字典序, 返回true 表示a 字典序小于b
bool cmp(const vector<int> &a, const vector<int> &b)
{
    for (size_t i = 1; i < a.size() && i < b.size(); i++)
        if (a[i] > b[i])
            return true;
        else
            return false;
    return a.size() > b.size();
}

// Dijkstra 算法, 求最短路径, 路径存储在path 数组中, dist 数组存储最短距离
void dijkstra()
{
    // 初始化 u == s
    int u = s;
    dist[u] = 0;
    st[u] = true;

    // 求n 次最短路径
    for (int i = 1; i <= n; i++)
    {
        // 更新u 的所有邻接点, 初始时u 为源点
        for (auto x : g[u])
        {
            int v = x.v, w = x.w; // 取出邻接点和权值

            // 如果v 已经被访问过, 跳过

```

```

    if (st[v])
        continue;
    // u->v
    if (dist[v] > dist[u] + w)
    {
        dist[v] = dist[u] + w;
        path[v] = path[u];    // 将更短的路径更新到path[v]中
        path[v].push_back(v); // 将v加入到路径中
    }
    // 路径权值和相等情况
    else if (dist[v] == dist[u] + w)
    {
        // 如果当前路段数较长, 更新为较短路径
        if (path[v].size() > path[u].size() + 1)
        {
            path[v] = path[u];
            path[v].push_back(v);
        }
        // 如果当前路径长度相等, 比较字典序
        else if (path[v].size() == path[u].size() + 1)
        {
            path[v].pop_back();

            // 如果true, 表示path[v]字典序大于path[u], 还是更新
            if (cmp(path[v], path[u]))
                path[v] = path[u];
            path[v].push_back(v);
        }
    }
}

// 更新完u的邻接点后, 找未访问的点中, dist最小的点
int li = inf, pos = 0;
for (int j = 1; j <= n; j++)
{
    // 如果已经被访问, 跳过
    if (st[j])
        continue;

    // 找到最短的dist[j]
    if (dist[j] < li)
    {
        li = dist[j];
        pos = j;
    }
}

```

```

    }
    // 如果找到两个dist[j]相等的点
    else if (dist[j] == li)
    {
        // 路段数更新为较少的
        if (path[j].size() < path[pos].size())
        {
            li = dist[j];
            pos = j;
        }
        // 如果路段数相同, 比较字典序
        else if (path[j].size() == path[pos].size())
        {
            // 更新为字典序较小的
            if (cmp(path[pos], path[j]))
            {
                li = dist[j];
                pos = j;
            }
        }
    }
}

st[pos] = 1; // 标记pos 已经被访问
u = pos;    // 下一个最短路径
}
}

int main()
{
    int k = 1;
    // 输入点数
    while (cin >> n)
    {
        init();
        // 输入边
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                int w;
                cin >> w;
                if (w != -1)
                    add(i, j, w);
            }
        }
    }
}

```

```

    }

    // 输入源点和终点
    cin >> s >> t;
    // 求最短路径
    dijkstra();

    cout << "Case " << k++ << endl;
    cout << "The least distance from " << s << "->" << t << " is " <<
dist[t] << endl;
    cout << "The path is ";

    // 输出最短路径
    for (size_t i = 0; i < path[t].size(); i++)
    {
        // 第一个不输出->
        if (i)
            cout << "->";
        cout << path[t][i];
    }
    cout << endl;
}
return 0;
}

```

4. 运行结果

```

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

```

5. 实验体会

Dijkstra 算法是一个经典的求单源最短路径算法，主要思想是初始化距离表和逐步更新节点距离，这是基于贪心的策略，同时将最短路径存入容器，可观察到具体的最短路径过程。本次实验，巩固了我的图论知识，提升了我的编程能力受益匪浅。