# 实验要求

以邻接表的形式存储图，给出图的深度和广度搜索算法（要求图的顶点不少于10个，边不少于20个，深度优先搜索要给出递归/非递归两种算法）。

# 图的储存

采用邻接表的形式储存一个图，每一个节点相当于一个链表的表头，这个链表储存了与当前点连接的所有点。

**储存：**

我们采用stl中的vector 来模拟一个链表

vector <int> G[maxn]; //邻接表储存边

其中G[i] 就是和i节点相邻的所有节点组成的链表。

**插入边**

void addedge(int u, int v)

{

G[u].push\_back(v);

}

这里我们插入的是一条单向的边，如果是双向的边我们需要执行addedge(u, v), addedge(v, u)

# 广度优先搜索

**思路：**

广度优先搜索我们通过队列来实现，每一次我们访问一个点时，将它所有的没有被访问过的点加入队列中，并设置访问标记，根据队列的先进先出的性质可以实现广度优先搜索。

**核心代码：**

void bfs(int s)

{

//memset(dist,-1,sizeof(dist));

deque<int> deq;

deq.clear();

dist[s] = 0;

deq.push\_back(s);

int now;

while(!deq.empty())

{

now = deq.front();

deq.pop\_front();

cout << now << endl;

for(int i = 0, len = G[now].size(); i < len; i++)if(dist[G[now][i]] == -1)

{

dist[G[now][i]] = dist[now]+1;

deq.push\_back(G[now][i]);

}

}

}

# 深度优先搜索

递归

**思路：**

深度优先搜索的思想就是不停地向当前结点的相邻的节点中没有被访问的节点访问，直到所有的节点都被访问完为止就实现了深度优先搜索

**核心代码：**

void dfs(int s)

{

dist[s] = 1;

cout << s << endl;

for(int i = 0, len = G[s].size(); i < len; i++)if(dist[G[s][i]] == -1)

{

dist[G[s][i]] = 1;

dfs(G[s][i]);

}

}

非递归

**思路：**

类似与广度优先搜索，但这次我们不适用队列，而是用栈，这样就可以保证后进去的点优先被访问到，所以也就实现了深度优先搜索的非递归算法。

**核心代码：**

void dfs\_none\_reverse(int s)

{

memset(dist, -1, sizeof(dist));

deque<int > deq; //栈来储存节点

deq.clear();

deq.push\_front(s);

int now, to;

while(!deq.empty())

{

now = deq.front();

deq.pop\_front();

if(dist[now] == 1) continue;

cout << now << endl;

dist[now] = 1;

for(int i = 0, len = G[now].size(); i < len; i++)if(dist[G[now][i]] == -1)

{

deq.push\_front(G[now][i]);

}

}

}

# 搜索算法的应用

Dijstra 单元双关键字最短路

/\*

给定 n 个点以及 m 条无向边，每条边的长度和花费是已知的，求两个点的最短距离，输出最短的距离并且最短距离下最小的费用。

输入样例：

3 2 //三个点， 两条边。

1 2 5 6 //1 到 2 的边 长度5 花费6

2 3 4 5 //2 到 3 的边 长度4 花费5

1 3 //求 1 到 3 的最短距离和最小消费

\*/

#include <bits/stdc++.h> // 包含所有 c++ 标准库

#define INF 0x3f3f3f3f

using namespace std;

typedef pair<int, int> PII; // pair->first 存储第一关键字， pair->second存储第二关键字

const int maxn = 10000;

struct Edge

{

int u, v, l, c;

Edge();

Edge(int u, int v, int l, int c):u(u),v(v),l(l),c(c){}

};

struct Node

{

int v, l, c;

Node(){}

Node(int v, int l, int c):v(v),l(l),c(c){}

bool operator < (const Node &a) const

{

if(l == a.l) return c > a.c;

return l > a.l;

}

};

vector<Edge> G[maxn];

priority\_queue<Node>pq;

int dist[maxn],cost[maxn],vis[maxn],n,m,tot;

void add\_edge(int u, int v, int l , int c)

{

G[u].push\_back(Edge(u,v,l,c));

}

// 返回 pair <int , int > 类型， first 为长度， second 为花费。

PII dijstra(int s, int d)

{

memset(dist, INF, sizeof(dist));

memset(cost, INF, sizeof(cost));

memset(vis, 0, sizeof(vis));

while(!pq.empty()) pq.pop();

pq.push(Node(s, 0, 0));

while(!pq.empty())

{

const Node nd = pq.top();

pq.pop();

if(vis[nd.v]) continue;

vis[nd.v] = true;

dist[nd.v] = nd.l;

cost[nd.v] = nd.c;

if(nd.v == d) return make\_pair(dist[d], cost[d]);

for(int i = 0, len = G[nd.v].size(); i < len; i++)

{

Edge& e = G[nd.v][i];

if(!vis[e.v])

{

pq.push(Node(e.v, nd.l + e.l, nd.c + e.c));

}

}

}

return make\_pair(dist[d], cost[d]);

}

int main()

{

while(scanf("%d%d", &n, &m) != EOF && n && m)

{

for(int i = 0; i <= n; i++) G[i].clear();

tot = 0;

int a, b, c, d;

for(int i = 0; i < m; i++)

{

scanf("%d%d%d%d", &a, &b, &c, &d);

add\_edge(a, b, c, d);

add\_edge(b, a, c, d);

}

int s, t;

scanf("%d%d", &s, &t);

PII p = dijstra(s, t);

printf("%d %d\n", p.first, p.second);

}

return 0;

}