数据结构课程设计



班级： 1618403

学号： 161840227

姓名： 韦 鑫

指导教师：孙 涵

目录

1.采用的数据结构 ………………………………………… 3

2.算法设计思想 …………………………………………… 3

3.关键代码 ………………………………………………… 3

4.测试数据和结果 ………………………………………… 7

5.算法的时间复杂度及其改进方法 ……………………… 7

6.结束语 …………………………………………………… 8

一、采用的数据结构

链式邻接表

//链式邻接表结构

struct node

{

int flag; //1表示小道，0表示大道

int v; //到达的点

long long dis; //两点间的距离

node \*next; //指向下一个

}Adj[maxv];

二、算法设计思想

①使用链式邻接表存储顶点信息、通过next指针连接可达的顶点

②通过Dijkstra算法求出1号路口到n号路口所需最小疲劳值

③利用to[]数组存储之前连续走过小道的总路长；Dijkstra算法中d[]数组的更新策略：若当前点u到下一点v走的是小道，权值为dis，先以d[u]减去to[u]的平方，在加上(to[u]+dis)的平方，所得值若小于d[v]，则说明以u为中介点更新v可使得1号口至v更优，更新d[v],并且更新to[v]；若为大道，则d[u]加上dis，所得值若小于d[v]，则更新d[v]、并令to[v] = 0

④通过pre[]数组存储顶点前驱，通过DFS输出路径

三、关键代码

const int maxv = 505;

const int inf = 1 << 30 - 1;

//链式邻接表结构

struct node

{

int flag; //1表示小道，0表示大道

int v; //到达的点

long long dis; //两点间的距离

node \*next; //指向下一个

}Adj[maxv];

//加入邻接点

void add(int flag, int u, int v, long long dis)

{

node \*p = &Adj[u];

while (p->next != NULL)p = p->next;

node \*s = (node \*)malloc(sizeof(node));

s->v = v;

s->dis = dis;

s->flag = flag;

p->next = s;

s->next = NULL;

}

bool vis[maxv]; //0表示该点未被访问

long long to[maxv]; //to[i]表示到达i时前面总共走过的路长

long long d[maxv]; //d[i]表示起点s到点i的最短距离

int pre[maxv]; //pre[i]表示点i的前驱,使用递归便可生成路径

int n, m;

//关键算法

void dijkstra(int s)

{

for (int i = 1; i <= n; i++)

{

vis[i] = 0;

d[i] = inf;

pre[i] = i;

}

d[s] = 0;

for (int i = 1; i <= n; i++)

{

int u = -1;

long long Min = inf;

for (int j = 1; j <= n; j++)

{

if (!vis[j] && d[j] < Min)

{

Min = d[j];

u = j;

}

}

if (u == -1)return;//说明剩下未被访问的点不连通

vis[u] = 1;

for (node \*p = &Adj[u]; p != NULL; p = p->next)

{

int v = p->v, flag = p->flag;

long long dis = p->dis;

if (!vis[v])

{

if (flag)//若为小道

{

long long tempd = (d[u] - to[u] \* to[u]) + (to[u] + dis) \* (to[u] + dis); //为了得到连续小道路程的平方，需要先减去to[u]\*to[u](即之前连续小道的平方),

if (tempd < d[v]) //再让之间的连续小道+下一段小道距离,取平方即得到接下来一步的总路程

{

d[v] = tempd;

to[v] = to[u] + dis;

pre[v] = u;//记录v的前驱u

}

}

else//若为大道

{

long long tempd = d[u] + dis;

if (tempd < d[v])

{

d[v] = tempd;

to[v] = 0;

pre[v] = u;//记录v的前驱u

}

}

}

}

}

}

//递归显示路径

void path(int s)

{

if (pre[s] == s)

{

printf("%d", s);

return;

}

else

{

path(pre[s]);

printf("->%d", s);

}

}

int main()

{

FILE \*fp;

if ((fp = fopen("data.txt", "r")) == NULL)

{

printf("打开文件失败！\n");

exit(0);

}

fscanf(fp, "%d %d", &n, &m);

for (int i = 1; i <= n; i++)Adj[i].next = NULL;

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

{

int flag, u, v;

long long dis;

fscanf(fp, "%d %d %d %lld", &flag, &u, &v, &dis);

add(flag, u, v, dis);

add(flag, v, u, dis);

}

fclose(fp);

dijkstra(1);

printf("最小疲劳度为：%lld\n", d[n]);

printf("路径为：\n");

path(n);

printf("\n从出发点至各个点的最小疲劳值：\n");

for (int i = 2; i <= n; i++)

{

printf("点%d：%lld\n", i, d[i]);

}

system("pause");

return 0;

}

四、测试数据和结果

data.txt:

6 7

1 1 2 3

1 2 3 2

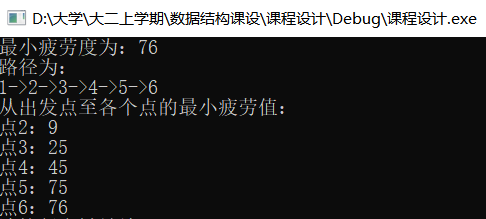
0 1 3 30

0 3 4 20

0 4 5 30

1 3 5 6

1 5 6 1



五、算法的时间复杂度即改进方法

Dijkstra算法时间复杂度为O(n\*n)

可以借助STL库中的优先级队列priority\_queue对Dijkstra算法进行堆优化，使得时间复杂度降至O(n\*logn)

六、结束语

代码共145行

该实验让我更加熟悉了Dijkstra算法的优化过程，并且学会对该算法的更新策略部分进行适当修改以解决相关问题。

另外通过查找相关资料，我了解到了Dijkstra算法的堆优化，只是通过调用库函数中的优先级队列便可将O(n\*n)的时间复杂度降至O(n\*logn)。

除了本题使用的邻接表、课内教的邻接矩阵存储图的结构外，我还搜集资料知道了链式前向星这种在竞赛常用的数据结构。