基于 Dijkstra 算法过必经点的最短路径设计

王小会 , 薛延刚, 李晓青

( 兰州工业学院 电气工程学院,甘肃 兰州 730000)

先是输入图的问题，论文中采用了通过微软基础类库( Microsoft Foundation Classes, MFC)中已有的CTypedPtrArray数组类模板实现路网的存储，动态创建新元素，统计元素个数等功能的方法，这不是本文讨论的算法问题，予以忽略。

下面算法采取经典的邻接矩阵的存储方式，简化问题，直达算法本身。

本文讨论的问题可简化为：在给定图中，有起点和终点，给出K个必经点以及N的大小，求出起点到终点必须经过那K个节点的N条最短路径（K, N < 5).

原图有回路是没有问题的，问题在于求出的最短路径中可不可以存在回路

如下图，若从A到C必过B，那么从A到C到B再到C是应该的吧，所以。。。

B

100 2

A

C

1

基本原理：

对于过 K 个必经点的 N 条最短路径算法, 必须先求得过第一条 K 个必经点的最短路径 , 然后再通

过断开第一最短路径所形成的子图集对应的最短路径集合求第二最短路径 , 如此循环以致求得第 N 条

最短路径。

1. 找到经过k个节点的最短路径

文章中只简单提及了一句，经网上查询后，方法如下：

1. 先判断这K个中间节点之间的连通情况，若有任意俩节点不连通，则满足条件的路径不存在; 反之，下一步。 判断是否连通的方法:在这K个节点中，任选一个为根节点，深度优先遍历该图，若其他的n-1个所有节点都遍历到了，则是连通的;反之，未连通

2. 对这K个中间节点做全排列，生成一个中间节点序列 全排列的起点记为 V1 ,终点记为 Vn，计算 V1和 Vn之间的最短距离和路径。

具体方法：对于一个中间节点序列,若任意俩相邻节点有边直接相连,则它们之间的距离为1(在带权图中,该距离为对应边的权重值);反之,使用Dijkstra

这？？？感觉它那个后面的样例都错了，中间最短路径咋在求啊（见参考文献），

应该是不管怎样，都用Dijkstra吧，

3. 求目标源点到 V1的局部最短路径,即为单源点最短路径,直接使用迪杰斯特拉算法即可。

4. 求 V1到 Vn的经过所有中间节点的局部最短路径,(2)中已求出。

1. 求 Vn 到目标终点的局部最短路径,即为单源点最短路径,直接使用迪杰斯特拉算法即可。

6. 将以上3条路径依序连接起来即得经过Smid中所有节点的(待选)全程最短路径。将以上3条路径的距离相加即得该条待选全程最短路径的距离。

7. 搜索(6)中求出的路径，其中距离最小的即为待求的全程最短距离，对应的路径就是待求的最短路径。

算法分析

记n, m分别为总节点和必须经过的中间节点的个 数。

由于Dijkstra 算法的时间复杂度为 O(n^2) (n 为网络 节点个数),与边数无关,特别适用于稠密图。

本算法以 Dijkstra 算法为基础,算法的时间复杂度与顶点个数和 中间节点个数有关,与边数无关,也适用于稠密图。

理论上,该算法的时间复杂度为 O(n^2 × m !) 。即,可以适用于大规模网络,不太适用于中间节点个数太多的情况。

- 参考文献：

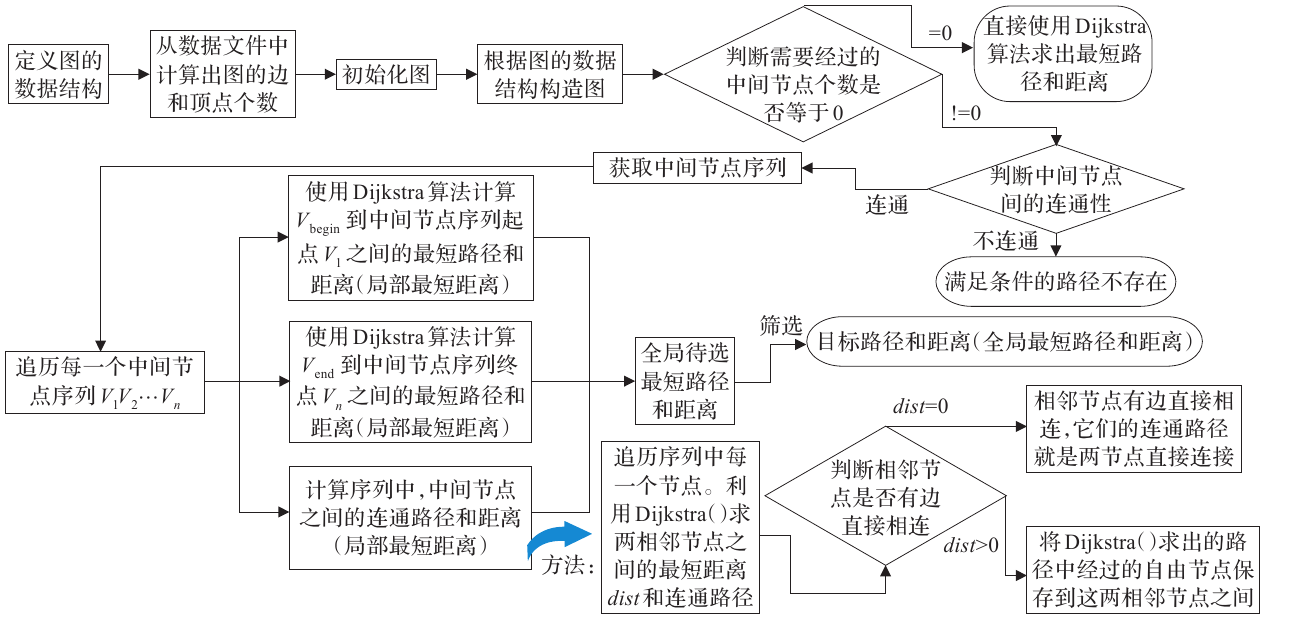
- 经过指定的中间节点集的最短路径算法

黄书力,胡大裟,蒋玉明

HUANG Shuli, HU Dasha, JIANG Yuming

四川大学 计算机(软件)学院,成都 610065

College of Computer Science, Sichuan University, Chengdu 610065, China



注：那个判断相邻节点是否有边相连，应该只适用于权值一样。

2.求n条最短径

针对文章中的算法描述，我大致提出以下问题：

- 断开第一条路径，找子图，怎么断开，断几条边

- 在所有子图中找到所有最短路径，最短的一条即为原图第几条最短路径（why，算了）

- 再次断开，找路径至N条，（是上一次的最短路径断开吗）

- 若断开不同的边可产生相同的最短路径，取值是一样的，但对于下一个子图来说，明显不一样，这又如何解决呢？

查了资料后，还是有问题解决不了，真不知道这最终结果是咋弄出来的，算法如下，

算法：

1. 将第 1 最短路径分离出来，分别将第1最短路径上的 一条边 从图G的边集中删除，就可得到一批子图Gk, k = 1,... , n ,

2. 这些子图上路径集合的并集就等于非第1最短路径集合k= 1,... ,n{ r2 ,... , rM}. 同样 ,可将路径集合 Rk, k= 1,... , n 分离为第1和非第1最短路径 .

3. 如此递归进行可将全部路径转化为某一子图上的第 1最短路径 . 将所有路径全部求出,然后按路径长度值大小对其进行排序即可求解前 N 条最短路径问题 .

完整算法：

1. 定义最短路径边集与切割边集（此中要放相对于原图的切割边。），有第一步算出的第一条路径，其他路径赋无穷大，切割集全赋空，有路径长度与切割边集长度

2. 将图中切割边集的边全删除（赋1000），并用dist存值，得子图

3. 在子图中依次删除上一最短路径中的边，并求相应的新的最短路径（在此过程中，注意若删除了同一条边，可以跳过求最短路径的步骤），求下一个时，需要恢复上一次删除的边

4. 求得一个最短路径，与最短路径集中比较，若小，则将之赋入路径集中（这一步又有一点注意，若新求得的最短路径小于最短路径集当前的路径，那么所有的最短路径集后移一格，这又需要N循环。。。这个过程可能导致最终求得的路径大与N条，不过不重要）

5. 在赋入最短路径的同时，也求得相应的切割集

6. 恢复图，再在基于所求的新的最短路径来求。。。进入2步骤

7. 若求得N次，或如上一最短路径不存在，则完

- 复杂度

从算法描述可以看出 ,对于 G(V, E) , N是所要求的最短路径数目, 与图的大小无关, 而且通常较小, 可以为常数量级;

EdgeNum是Paths[i-1]边的数目,其上限是边的总数目(实际应用中,远不能达到),

由于 Dijkstra 算法的时间复杂度为 O(n^2), 因此总的时间复杂度为 O ( e \* n2) ,其中e和n分别为图G(V,E)的边和顶点的数目,

- 参考文献：

- 前 N 条最短路径问题的算法及应用

柴登峰 , 张登荣

(浙江大学 空间信息技术研究所 ,杭州 浙江 310027)

- Dijkstra及基于 Dijkstra的前 N 条最短路径算法

在智能交通系统中的应用

王 峰 , 游志胜 , 曼丽春, 高 燕 , 汤丽萍

四川大学 计算机学院 , 四川 成都 610065;2. 成都信息工程学院 , 四川 成都 610225

待解决问题：

1. 求N条路径时，断开不同的边产生了相同的最短路径，在算法中却只保存了其中的一条，断开的边也只是那一条，所以在求下一条最短路径时产生了错误，

2. 若是求必过K点的N条最短路径，则问题更为严重，因为在求出第一条过K点的最短路径后，若将之断开，大概率会断开必过的那K个点，原文中就压根没提到这个问题（当然，它还有好多问题都没有提到。。），要想在断开路径时，不断开必经的K个点，这玩意。。。

目前应该就这两个大问题吧，时间有限来不及解决，还有这难度确实太大了（至少对于我而言），还有为什么相关文献写的都那样，是觉得细节太简单了吗，到底是在解决一般问题，还是特定问题啊。。。

完整代码如下：

为方便图的输入，以将样例图置入其中，26个节点，65条无向边，我输入都快吐了，若需输入其他图，可能需要轻微修改，如数组大小等，不过这问题太小。

还有就是原图中的顶点竟然是从0开始的

#include<stdio.h>

#include<stdlib.h>

#define VERTEX\_NUM 26

#define Maxsize 1000

int K, N; // K, N < 5

int startnode, endnode;

int perm[Maxsize][Maxsize], permN = 0;

void swap(int\* a, int\* b)

{

int c = \*a;

\*a = \*b;

\*b = c;

}

void Permutations(int str[], int index, int str\_size)

{

int i = 0,j = 0;

if(index == str\_size)

{

for(i = 0; i < str\_size; i++){

perm[permN][i] = str[i];

}

permN++;

}

else

{

for(j = index;j < str\_size; j++)

{

swap(&str[j],&str[index]);

Permutations(str,index+1,str\_size);

swap(&str[j],&str[index]);

}

}

}

typedef struct

{

int edges[VERTEX\_NUM][VERTEX\_NUM]; // The value of this two dimensional array is the weight of the edge.

int vertextNum; // The number of vertex.

int edgeNum; // The number of edge.

int vex[VERTEX\_NUM]; // To store vertex.

}MGraph;

int dijkstra(MGraph\* G, int startVertexIndex, int endVertexIndex, int p1[], int \*Npath)

{

int d[VERTEX\_NUM]; // 记录v0到各定点的最小路径

int use[VERTEX\_NUM]; // 记录各节点是否已求得最短路径，0表示未求得，1表示已求得

int p[VERTEX\_NUM]; // 记录v0到个顶点最小路径的中间节点

for(int i = 0; i < VERTEX\_NUM; i++){

use[i] = 0;

p[i] = startVertexIndex;

d[i] = G->edges[startVertexIndex][i];

}

use[startVertexIndex] = 1; //由于v0-v0不存在，因此当v0已求得最短路径

for(int i = 1; i < VERTEX\_NUM; i++){

if(use[endVertexIndex])

break;

int s, min = 1000;// s为中间点,min为最小路径

for(int j = 0; j < VERTEX\_NUM; j++){ // 找到最短的路径

if(!use[j] && d[j] < min){

min = d[j];

s = j;

}

}

use[s] = 1; // 标记此点已过

for(int w = 0; w < VERTEX\_NUM; w++){ // 重新标记路径

if(!use[w] && (min + G->edges[s][w] < d[w])){

d[w] = min + G->edges[s][w];

p[w] = s;

}

}

}

if(!use[endVertexIndex]){

\*Npath = 0;

return 1000;

}

// 将Path赋值

int w, p2[26], i = 0;

p2[0] = endVertexIndex;

int z = endVertexIndex;

for(w = 1; w < VERTEX\_NUM; w++){ // 此处隐含所经路径节点不超过总节点数。

p2[w] = p[z];

z = p[z];

if(z == startVertexIndex)

break;

}

\*Npath = w+1;

while(w >= 0){

p1[i++] = p2[w--];

}

return d[endVertexIndex];

}

/\*

//深度优先递归算法

void DFS(MGraph \*G, int i, int visited[])

{

if(!visited[i])

return ;

visited[i] = 1; //被访问的标记

//printf("%c->",G.vexs[i]);

for(int j = 0; j < G->vertextNum; j++) {

if(G->edges[i][j] < 1000 && !visited[j]) //边(i,j)存在且j顶点未被访问，递归

DFS(G, j, visited);

}

}

int isConnected(int p[], MGraph \*G)

{

int root = p[0];

int visited[26];

for(int i = 0; i < G->vertextNum; i++)

visited[i] = 0;

DFS(G, root, visited);

for(int i = 0; i < K; i++){

if(!visited[p[i]])

return 0;

}

return 1;

}

\*/

void FindKpaths(MGraph \*G, int Paths[], int Dist[], int Npaths[])

{

int Path1[VERTEX\_NUM], Path2[VERTEX\_NUM], Path22[VERTEX\_NUM], Path3[VERTEX\_NUM];

int Npath1, Npath2 = 0, Npath22, Npath3;

int \*a = &Npath1, \*b = &Npath22, \*c = &Npath3;

int dist1, dist2 = 0, dist22, dist3, dist;

//if(!(isConnected(perm[0], G)))

// return ;

if(!K){

Dist[0] = dijkstra(G, startnode, endnode, Paths, Npaths);

return;

}

for(int i = 0; i < permN; i++){

if(K == 1){

dist2 = 0;

Npath2 = 0;

}

dist2 = 0;

Npath2 = 0;

for(int j = 0; j < K-1; j++){

dist22 = dijkstra(G, perm[i][j], perm[i][j+1], Path22, b);

dist2 += dist22;

if(!j)

for(int m = 0; m < Npath22;){

Path2[Npath2++] = Path22[m++];

}

else

for(int m = 1; m < Npath22;){

Path2[Npath2++] = Path22[m++];

}

}

dist1 = dijkstra(G, startnode, perm[i][0], Path1, a);

dist3 = dijkstra(G, perm[i][K-1], endnode, Path3, c);

dist = dist1 + dist2 + dist3;

if(dist < Dist[0]){

Dist[0] = dist;

int s = 0;

for(int k = 0; k < Npath1; ){

Paths[s++] = Path1[k++];

}

for(int k = 1; k < Npath2; ){

Paths[s++] = Path2[k++];

}

for(int k = 1; k < Npath3; ){

Paths[s++] = Path3[k++];

}

Npaths[0] = s;

}

}

}

void FindNpaths(MGraph \*G, int Paths[][VERTEX\_NUM], int Dist[], int Npaths[])

{

int P[26], Np, \*np = &Np, dist[26];

int CutEdgeSet[26][26], CutEdgeNum[26];

int EdgeNum = 0;

int i, j;

int edge, edgedist, flagContinue;

for(i = 1; i < N; i++){ // 初始化

for(j = 0; j < VERTEX\_NUM; j++){

CutEdgeSet[0][j] = -1;

CutEdgeSet[i][j] = -1;

Paths[i][j] = -1;

}

Npaths[i] = 0;

dist[i] = -1;

CutEdgeNum[i] = 0;

}

CutEdgeNum[0] = 0;

for(i = 1; i < N ; i++){

if(Npaths[i-1] == 0) break; //上一条最短路径不存在 , 结束运行

for(j = 0; j < CutEdgeNum[i-1]; j++){ // 在G 中将 CutEdgeSet中各边删除 , 得到子图 G′

edge = CutEdgeSet[i-1][j];

dist[j] = G->edges[edge/100][edge%100];

G->edges[edge/100][edge%100] = 1000;

}

// 在G′中依次删除Paths[i-1]中的各边,求G′的最短路径上一条最短路径的边数

EdgeNum = Npaths[i-1] - 1; // 边数为顶点数-1

for(j = 0; j < EdgeNum; j++){

// 找到要断开的边, 如是1022，表示是10顶点与22，而120，表示1与20顶点。。。小顶点一定在前，例如：121 是边1与21 还是 12与1.

if(Paths[i-1][j] <= Paths[i-1][j+1])

edge = Paths[i-1][j] \* 100 + Paths[i-1][j+1];

else

edge = Paths[i-1][j+1] \* 100 + Paths[i-1][j];

/\*

//新删除边集合与已有删除边集合是否有相同的标志 , 若相同则为 false

flagContinue = 1;

// 若 edge ∪ CutEdgeSet与已有删除边集合相同, 则没有必要继续进行下面的运算, 因为得到的最短路径是一样的

//if(CutEdgeSetCompare(edge ∪ CutEdgeSet, jj) == 1){

for(int m = 1; m < 26; m++){

flagContinue = 0;

for(int n = 0; n < CutEdgeNum[m]; n++){

if(CutEdgeSet[i-1][n] != CutEdgeSet[m][n]){

flagContinue = 1;

break;

}

}

}

if(!flagContinue)

break;

\*/

for(int m = 0; m < Np; m++){

P[m] = -1;

}

edgedist = G->edges[edge/100][edge%100]; // 存储边值

G->edges[edge/100][edge%100] = 1000; // 断开边

int dist1; // 暂存路径的值

dist1 = dijkstra(G, startnode, endnode, P, np);

G->edges[edge/100][edge%100] = edgedist; // 还原边

for(int k = i; k < N; k++){

if(dist1 <= Dist[k] && dist1 != 1000){ // 找所有子图最短路径中的最短的

//路径P尚未出现过

int flag = 1;

for(int m = 0; m < N; m++){

flag = 1;

for(int n = 0; n < Np; n++){

if(P[n] != Paths[m][n]){

flag = 0;

break;

}

}

if(flag == 1)

break;

}

if(Npaths[i] && flag)

break;

for(int m = N - 1; m > k; m--){ //将路径P插入适当位置 , 使 Path s[i] 中各条路径按升序存放

for(int n = 0; n < Npaths[m-1]; n++){

Paths[m][n] = Paths[m-1][n]; // 将在上一个子图中找到的路径合理利用，减少时间

CutEdgeSet[m][n] = CutEdgeSet[m-1][n];

}

CutEdgeNum[m] = CutEdgeNum[m-1];

Npaths[m] = Npaths[m-1];

Dist[m] = Dist[m-1];

}

for(int n = 0; n < CutEdgeNum[k-1]; n++)

CutEdgeSet[k][n] = CutEdgeSet[k-1][n];

Npaths[k] = Np;

for(int m = 0; m < Npaths[k]; m++)

Paths[k][m] = P[m];

CutEdgeNum[k] = CutEdgeNum[k-1] + 1;

CutEdgeSet[k][CutEdgeNum[k-1]] = edge;

Dist[k] = dist1;

break;

}

}

}

for(j = 0; j < CutEdgeNum[i-1]; j++){ // 恢复图

edge = CutEdgeSet[i-1][j];

G->edges[edge/100][edge%100] = dist[j]; // 还原边

//edge.dist = 1000;// 原算法的edge.dist = ∞ 是不正确的

}

}

}

int main(void)

{

MGraph \*G = (MGraph \*)malloc(sizeof(MGraph));

//初始化图

/\*

printf("Please input the number of the vertexs:");

scanf("%d", &G->vertextNum);

printf("Please input the number of the edges:");

scanf("%d", &G->edgeNum);

for(int i = 0; i < VERTEX\_NUM; i++){

G->vex[i] = i;

}B

for(int i = 0; i < VERTEX\_NUM; i++){

for(int j = 0; j < VERTEX\_NUM; j++){

scanf("%d", G->edges[i][j]);

}

}

\*/

G->vertextNum = 26;

G->edgeNum = 100; // 估计。。。

for(int i = 0; i < VERTEX\_NUM; i++){

G->vex[i] = i+1;

}

for(int i = 0; i < VERTEX\_NUM; i++){

for(int j = 0; j < VERTEX\_NUM; j++){

G->edges[i][j] = 1000;

}

}

G->edges[0][1] = 62; G->edges[0][4] = 37; G->edges[0][5] = 16; G->edges[0][6] = 23; G->edges[0][8] = 37;

G->edges[1][2] = 25; G->edges[1][3] = 18; G->edges[1][4] = 34;

G->edges[2][3] = 26; G->edges[2][13] = 32; G->edges[2][14] = 20;

G->edges[3][4] = 19; G->edges[3][11] = 32; G->edges[3][12] = 31; G->edges[3][14] = 17;

G->edges[4][8] = 13; G->edges[4][11] = 27; G->edges[4][12] = 35;

G->edges[5][6] = 10; G->edges[5][7] = 22;

G->edges[6][7] = 17; G->edges[6][8] = 27; G->edges[6][10] = 33;

G->edges[7][9] = 25; G->edges[7][10] = 40; G->edges[7][15] = 41;

G->edges[8][10] = 16; G->edges[8][11] = 22;

G->edges[9][10] = 28; G->edges[9][15] = 21; G->edges[9][16] = 15; G->edges[9][17] = 38;

G->edges[10][17] = 25;

G->edges[11][12] = 19; G->edges[11][16] = 45; G->edges[11][18] = 25; G->edges[11][22] = 42;

G->edges[12][13] = 25; G->edges[12][14] = 17; G->edges[12][18] = 20;

G->edges[13][14] = 22; G->edges[13][18] = 39; G->edges[13][19] = 18;

G->edges[14][19] = 38;

G->edges[15][16] = 26; G->edges[15][21] = 30;

G->edges[16][17] = 22; G->edges[16][20] = 17;

G->edges[17][18] = 23; G->edges[17][20] = 28; G->edges[17][22] = 20;

G->edges[18][19] = 29; G->edges[18][22] = 20; G->edges[18][24] = 31; G->edges[18][25] = 43;

G->edges[19][25] = 29;

G->edges[20][21] = 17; G->edges[20][23] = 30;

G->edges[21][22] = 55; G->edges[21][23] = 41;

G->edges[22][23] = 23; G->edges[22][24] = 23; G->edges[22][25] = 44;

G->edges[23][24] = 34;

G->edges[24][25] = 20;

for(int i = 0; i < VERTEX\_NUM; i++){

for(int j = 0; j < VERTEX\_NUM; j++){

if(i == j)

G->edges[i][j] = 0;

G->edges[j][i] = G->edges[i][j];

}

}

printf("Please input the startnode - 1 !!!:");

scanf("%d", &startnode);

printf("Please input the endnode - 1 !!!:");

scanf("%d", &endnode);

//while(1){

printf("Please input the number of N:");

scanf("%d", &N);

printf("Please input the number of K:");

scanf("%d", &K);

// 全排列

int k[K];

if(K){

printf("Please input the vertexs of K(attention: - 1 !!!):");

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

scanf("%d", k+i);

Permutations(k,0,K);

}

int Paths[N][VERTEX\_NUM];

int Npaths[VERTEX\_NUM];

int Dist[N];

for(int i = 0; i < N; i++){

A

Dist[i] = 1000;

}

FindKpaths(G, Paths[0], Dist, Npaths);

if(N > 1)

FindNpaths(G, Paths, Dist, Npaths);

for(int j = 0; j < N; j++){

printf("The paths: ");

for(int i = 0; i < Npaths[j]; i++)

printf("%d ", Paths[j][i]);

printf("The length: ");

printf("%d\n", Dist[j]);

}

/\*

printf("You need input more? (0 to stop, 1 to go on)");

int z = 1;

scanf("%d", &z);

if(z == 0)

break;

for(int j = 0; j < N; j++){

for(int i = 0; i < Npaths[j]; i++){

Paths[j][i] = -1;

}

Dist[j] = 0;

Npaths[j] = 0;

}

permN = 0;

}

\*/

return 0;

}

