常用算法之最短路径算法

（摘自csdn，加上本人解释，帮助理解）

最短路径算法常用的有两种

Dijkstra算法和floyd算法

1.定义概览

Dijkstra(迪杰斯特拉)算法是典型的单源最短路径算法，用于计算一个节点到其他所有节点的最短路径。主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止。Dijkstra算法是很有代表性的最短路径算法，在很多专业课程中都作为基本内容有详细的介绍，如数据结构，图论，运筹学等等。注意该算法要求图中不存在负权边（注意适用范围）。

问题描述：在无向图 G=(V,E) 中，假设每条边 E[i] 的长度为 w[i]，找到由顶点 V0 到其余各点的最短路径。（单源最短路径）

2.算法描述

1)算法思想：设G=(V,E)是一个带权有向图，把图中顶点集合V分成两组，第一组为已求出最短路径的顶点集合（用S表示，初始时S中只有一个源点（这个点就是起点，起点到本身的最短距离，这个不用说了吧），以后每求得一条最短路径 , 就将加入到集合S中，直到全部顶点都加入到S中，算法就结束了），第二组为其余未确定最短路径的顶点集合（用U表示），按最短路径长度的递增次序依次把第二组的顶点加入S中。在加入的过程中，总保持从源点v到S中各顶点的最短路径长度不大于从源点v到U中任何顶点的最短路径长度。此外，每个顶点对应一个距离，S中的顶点的距离就是从v到此顶点的最短路径长度，U中的顶点的距离，是从v到此顶点只包括S中的顶点为中间顶点的当前最短路径长度。

2)算法步骤：

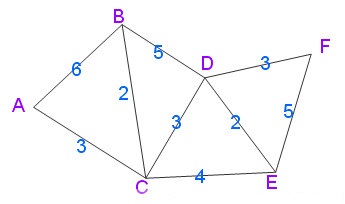
a.初始时，S只包含源点，即S＝{v}，v的距离为0。U包含除v外的其他顶点，即:U={其余顶点}，若v与U中顶点u有边，则<u,v>正常有权值，若u不是v的出边邻接点，则<u,v>权值为∞。（这句话可以理解为map[i][j]代表从i到j的距离，如果有边连接就是边的长度，没有就设置为一个非常大的数）

b.从U中选取一个距离v最小的顶点k，把k，加入S中（该选定的距离就是v到k的最短路径长度）。

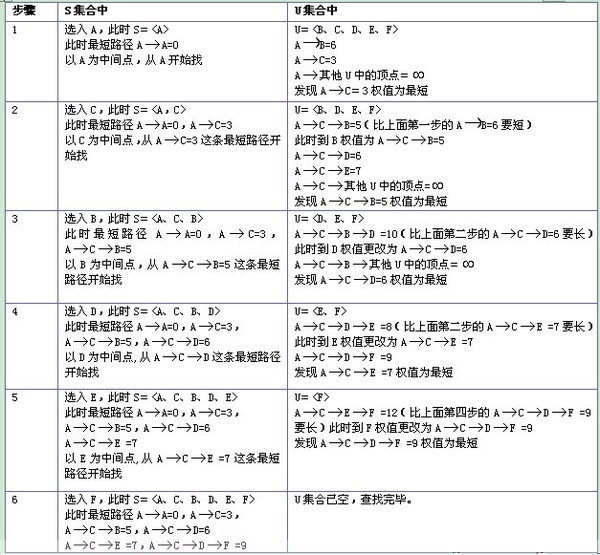
c.以k为新考虑的中间点，修改U中各顶点的距离；若从源点v到顶点u的距离（经过顶点k）比原来距离（不经过顶点k）短，则修改顶点u的距离值，修改后的距离值的顶点k的距离加上边上的权。

d.重复步骤b和c直到所有顶点都包含在S中。

（看了以下例子之后应该就会更加明白了）



以A为起点



（以下代码是我自己实现的，可能会有错误，大家可以自己实现）

#include<stdio.h>//dijiskar 最短路径算法

#include<string.h>

#define MAX 0x0fffff// 设置一个很大的数

// 表示节点数

int main()

{

int map[20][20];

int pre[20]= {0}; // 表示当前点的前一个节点是谁

int dist[20]; // 表示距离

int visited[20]= {0};

int m,n;// m表示图的大小，n表示有多少种连接

scanf("%d%d",&m,&n);//

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

{

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

{

map[i][j]=MAX;

}

}// 初始化，权值先全部设置成比较大的数

// memset(map,8,sizeof(map));// 批量赋值，因为memset是按照字节赋值所以其实这里 给的是很大的数

for(int j=0; j<n; j++)

{

int a,b,len;

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

map[a][b]=len;

map[b][a]=len; // 因为是无向图，所以要这样赋值

}

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

memset(dist,8,sizeof(dist));//现在还没有选择起点，所以把他这是

pre[0]=0;

dist[0]=0;// 先把点0选了

int u;

visited[0]=1;

u=0;

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

{

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

{

if(visited[j]!=1&&map[u][j]<MAX) //更新dist

{

if(dist[j]>map[u][j]+dist[u])

{

pre[j]=u;// 更新前驱节点

dist[j]=map[u][j]+dist[u];

//printf("this u= %d this j = %d\n",u,j);

}

}

}

int min = MAX;

for(int j=0; j<m; j++) // 寻找下一个目标点,在剩下的点中寻找最小的

{

if(min>dist[j]&&visited[j]!=1)

{

min=dist[j];

u=j;

}

}

visited[u]=1;

}

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

// {

// printf("%d ",dist[i]);

// }

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

// {

// printf("%d ",pre[i]);

// }

return 0;

}

思考： 怎么计算出每个顶点的最短路径到底有多少条呢？

**Floyd算法**

1.定义概览

**Floyd-Warshall算法**（Floyd-Warshall algorithm）是解决任意两点间的最短路径的一种算法，可以正确处理有向图或负权（适用范围）的最短路径问题，同时也被用于计算有向图的传递闭包。Floyd-Warshall算法的时间复杂度为O(N3)，空间复杂度为O(N2)。

2.算法描述

1)算法思想原理：

     Floyd算法是一个经典的动态规划算法。用通俗的语言来描述的话，首先我们的目标是寻找从点i到点j的最短路径。从动态规划的角度看问题，我们需要为这个目标重新做一个诠释（这个诠释正是动态规划最富创造力的精华所在）

      从任意节点i到任意节点j的最短路径不外乎2种可能，1是直接从i到j，2是从i经过若干个节点k到j。所以，我们假设Dis(i,j)为节点u到节点v的最短路径的距离，对于每一个节点k，我们检查Dis(i,k) + Dis(k,j) < Dis(i,j)是否成立，如果成立，证明从i到k再到j的路径比i直接到j的路径短，我们便设置Dis(i,j) = Dis(i,k) + Dis(k,j)，(其实这就是它的动态转移方程)这样一来，当我们遍历完所有节点k，Dis(i,j)中记录的便是i到j的最短路径的距离。

2).算法描述：

a.从任意一条单边路径开始。所有两点之间的距离是边的权，如果两点之间没有边相连，则权为无穷大。

b.对于每一对顶点 u 和 v，看看是否存在一个顶点 w 使得从 u 到 w 再到 v 比己知的路径更短。如果是更新它。

// 这个算法看起来好像很容易理解，那么怎么实现呢

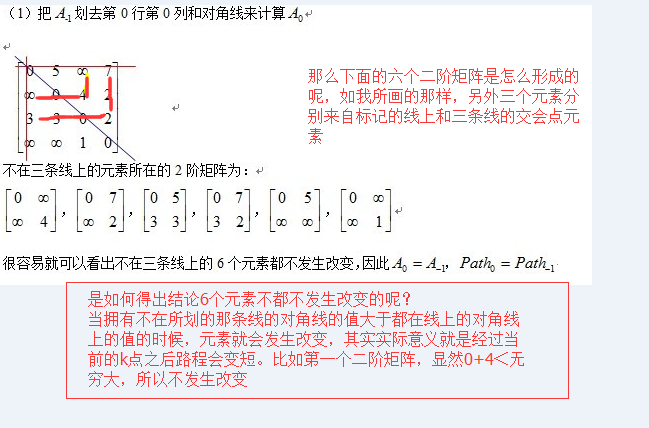
3).Floyd算法过程矩阵的计算----十字交叉法

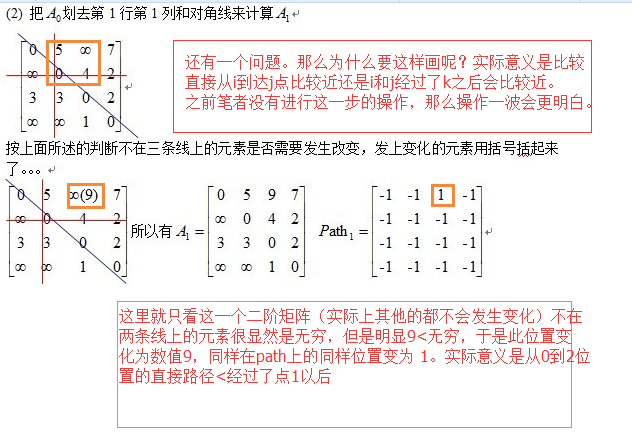
方法：两条线，从左上角开始计算一直到右下角 如下所示

给出矩阵，其中矩阵A是邻接矩阵，而矩阵Path记录u,v两点之间最短路径所必须经过的点

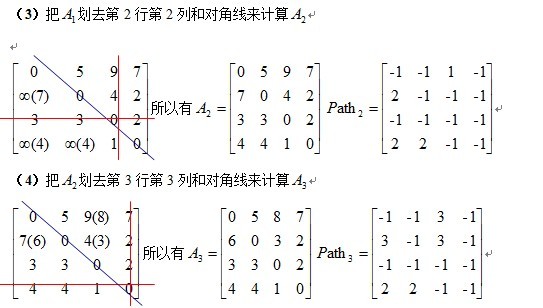
计算方法如下

//以下方法来自网上，但是讲的并不详细，然后翻了资料之后终于理解了。。





接下来的内容就不实际继续操作了。。。。



// 当然这里就不去证明这个算法为什么是正确的（一开始我对这个也是抱有怀疑的态度，总觉得有点毛病。。。） 不过的确是正确的，有兴趣可以去搜索以下。

那么接下来就是代码实现

#include<stdio.h>

#include<string.h>

#define N 4

#define MAX 0x8fffff

void Floyd()

{

int map[N][N]={0,5,MAX,7,

MAX,0,4,2,

3,3,0,2,

MAX,MAX,1,0};

int path[N][N];

memset(path,-1,sizeof(path));// 要注意的是memset是按字节赋值

for(int k=0;k<4;k++) // 在这里明显看出来效率是O(n^3)

{

for(int i=0;i<4;i++)

{

for(int j=0;j<4;j++)

{

if(map[i][j]>map[i][k]+map[k][j])

{

map[i][j]=map[i][k]+map[k][j];

path[i][j]=k;

}

}

}

}

for(int i=0;i<4;i++)

{

for(int j=0;j<4;j++)

{

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

}

printf("\n");

}

printf("---------path------------\n");

for(int i=0;i<4;i++)

{

for(int j=0;j<4;j++)

{

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

}

printf("\n");

}

}

int main()

{

Floyd();

return 0;

}