/\*\*

\* floyd.c 弗洛伊德算法

\* @author Darbuly 2018-2019

\* @copyright Darbuly

\*/

#include <stdio.h>

#include <stdlib.h>

#define VRType int

#define VertexType char

#define InfoType int

#define MAX\_VERTEX\_NUM 20

#define MAX\_VALUE 65535

typedef enum{DG,DN,AG,AN}GraphKind;

typedef int Pathmatirx[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];

typedef int ShortPathTable[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];

typedef struct ArcCell{

VRType adj;

InfoType \*info;

}AreCell,AdjMatrix[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];

typedef struct

{

VertexType vexs[MAX\_VERTEX\_NUM];

AdjMatrix arc;

int vexnum,arcnum;

GraphKind kind;

}MGraph;

void g\_create(MGraph \*graph)

{

int num;

int i,j,k;

char c;

printf("Please enter the vexnum of vertex:\n");

scanf("%d",&graph->vexnum);

getchar();

printf("Please enter vertex infomations:\n");

for(i=0;i<graph->vexnum;i++)

{

scanf("%c",&graph->vexs[i]);

getchar();

}

for(i=0;i<graph->vexnum;i++)

for(j=0;j<graph->vexnum;j++)

graph->arc[i][j].adj = MAX\_VALUE;

graph->arcnum = 0;

for(i=0;i<graph->vexnum;i++)

{

printf("Please enter vertex nextto the %c ,and end by #\n",graph->vexs[i]);

for(j=0;j<graph->vexnum;j++)

{

scanf("%c",&c);

if('#'==c)

{

getchar();

break;

}

scanf("%d",&num);

for(k=0;k<graph->vexnum;k++)

{

if(graph->vexs[k]!=c)

continue;

graph->arc[i][k].adj = num;

graph->arcnum++;

}

getchar();

}

}

graph->arcnum /=2;

printf("\n");

for(i=0;i<graph->vexnum;i++)

{

for(j=0;j<graph->vexnum;j++)

{

printf("%5d\t",graph->arc[i][j].adj);

}

printf("\n");

}

}

void ShortestPath\_Floyd(MGraph G,Pathmatirx \*P,ShortPathTable \*D)

{

int v,w,k;

for(v=0;v<G.vexnum;v++)

{

for(w=0;w<G.vexnum;w++)

{

(\*D)[v][w] = G.arc[v][w].adj;

(\*P)[v][w] = w;

}

}

for(k=0;k<G.vexnum;k++)

{

for(v=0;v<G.vexnum;v++)

{

for(w=0;w<G.vexnum;w++)

{

if( (\*D)[v][w] > (\*D)[v][k] + (\*D)[k][w])

{

(\*D)[v][w] = (\*D)[v][k] + (\*D)[k][w];

(\*P)[v][w] = (\*P)[v][k];

}

}

}

}

for(k=0;k<G.vexnum;k++)

{

for(v=0;v<G.vexnum;v++)

{

printf("%d\t",(\*D)[k][v]);

}

printf("\n");

}

printf("\n");

for(k=0;k<G.vexnum;k++)

{

for(v=0;v<G.vexnum;v++)

{

printf("%d\t",(\*P)[k][v]);

}

printf("\n");

}

}

int main()

{

MGraph graph;

Pathmatirx P;

ShortPathTable D;

g\_create(&graph);

ShortestPath\_Floyd(graph,&P,&D);

printf("FLOYD.C TEST\n");

return 0;

}

for(k=0;k<G.vexnum;k++)

{

for(v=0;v<G.vexnum;v++)

{

for(w=0;w<G.vexnum;w++)

{

if( (\*D)[v][w] > (\*D)[v][k] + (\*D)[k][w])

{

(\*D)[v][w] = (\*D)[v][k] + (\*D)[k][w];

(\*P)[v][w] = (\*P)[v][k];

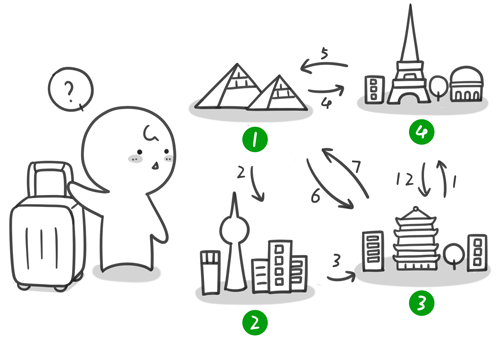
}

}

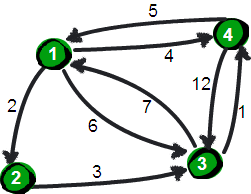
}

}

# [Floyd-傻子也能看懂的弗洛伊德算法（转）](https://www.cnblogs.com/wangyuliang/p/9216365.html)



暑假，小哼准备去一些城市旅游。有些城市之间有公路，有些城市之间则没有，如下图。为了节省经费以及方便计划旅程，小哼希望在出发之前知道任意两个城市之前的最短路程。



        上图中有4个城市8条公路，公路上的数字表示这条公路的长短。请注意这些公路是单向的。我们现在需要求任意两个城市之间的最短路程，也就是求任意两个点之间的最短路径。这个问题这也被称为“多源最短路径”问题。

        现在需要一个数据结构来存储图的信息，我们仍然可以用一个4\*4的矩阵（二维数组e）来存储。比如1号城市到2号城市的路程为2，则设e[1][2]的值为2。2号城市无法到达4号城市，则设置e[2][4]的值为∞。另外此处约定一个城市自己是到自己的也是0，例如e[1][1]为0，具体如下。



        现在回到问题：如何求任意两点之间最短路径呢？通过之前的学习我们知道通过深度或广度优先搜索可以求出两点之间的最短路径。所以进行n2遍深度或广度优先搜索，即对每两个点都进行一次深度或广度优先搜索，便可以求得任意两点之间的最短路径。可是还有没有别的方法呢？

        我们来想一想，根据我们以往的经验，如果要让任意两点（例如从顶点a点到顶点b）之间的路程变短，只能引入第三个点（顶点k），并通过这个顶点k中转即a->k->b，才可能缩短原来从顶点a点到顶点b的路程。那么这个中转的顶点k是1~n中的哪个点呢？甚至有时候不只通过一个点，而是经过两个点或者更多点中转会更短，即a->k1->k2b->或者a->k1->k2…->k->i…->b。比如上图中从4号城市到3号城市（4->3）的路程e[4][3]原本是12。如果只通过1号城市中转（4->1->3），路程将缩短为11（e[4][1]+e[1][3]=5+6=11）。其实1号城市到3号城市也可以通过2号城市中转，使得1号到3号城市的路程缩短为5（e[1][2]+e[2][3]=2+3=5）。所以如果同时经过1号和2号两个城市中转的话，从4号城市到3号城市的路程会进一步缩短为10。通过这个的例子，我们发现每个顶点都有可能使得另外两个顶点之间的路程变短。好，下面我们将这个问题一般化。

        当任意两点之间不允许经过第三个点时，这些城市之间最短路程就是初始路程，如下。



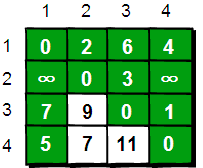
假如现在只允许经过1号顶点，求任意两点之间的最短路程，应该如何求呢？只需判断e[i][1]+e[1][j]是否比e[i][j]要小即可。e[i][j]表示的是从i号顶点到j号顶点之间的路程。e[i][1]+e[1][j]表示的是从i号顶点先到1号顶点，再从1号顶点到j号顶点的路程之和。其中i是1~n循环，j也是1~n循环，代码实现如下。

[IMG_260](http://www.cnblogs.com/wangyuliang/p/javascript:void(0);)

1 for (i = 1; i <= n; i++)2 {3 for (j = 1; j <= n; j++)4 {5 if (e[i][j] > e[i][1] + e[1][j])6 e[i][j] = e[i][1] + e[1][j];7 }8 }

[IMG_261](http://www.cnblogs.com/wangyuliang/p/javascript:void(0);)

在只允许经过1号顶点的情况下，任意两点之间的最短路程更新为：



        通过上图我们发现：在只通过1号顶点中转的情况下，3号顶点到2号顶点（e[3][2]）、4号顶点到2号顶点（e[4][2]）以及4号顶点到3号顶点（e[4][3]）的路程都变短了。

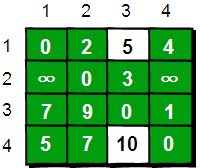
        接下来继续求在只允许经过1和2号两个顶点的情况下任意两点之间的最短路程。如何做呢？我们需要在只允许经过1号顶点时任意两点的最短路程的结果下，再判断如果经过2号顶点是否可以使得i号顶点到j号顶点之间的路程变得更短。即判断e[i][2]+e[2][j]是否比e[i][j]要小，代码实现为如下。

[IMG_263](http://www.cnblogs.com/wangyuliang/p/javascript:void(0);)

1 //经过1号顶点2 for(i=1;i<=n;i++) 3 for(j=1;j<=n;j++) 4 if (e[i][j] > e[i][1]+e[1][j]) e[i][j]=e[i][1]+e[1][j]; 5 //经过2号顶点6 for(i=1;i<=n;i++) 7 for(j=1;j<=n;j++) 8 if (e[i][j] > e[i][2]+e[2][j]) e[i][j]=e[i][2]+e[2][j];

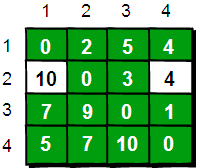
[IMG_264](http://www.cnblogs.com/wangyuliang/p/javascript:void(0);)

在只允许经过1和2号顶点的情况下，任意两点之间的最短路程更新为：

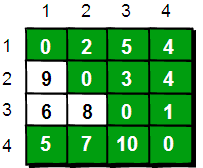


        通过上图得知，在相比只允许通过1号顶点进行中转的情况下，这里允许通过1和2号顶点进行中转，使得e[1][3]和e[4][3]的路程变得更短了。

        同理，继续在只允许经过1、2和3号顶点进行中转的情况下，求任意两点之间的最短路程。任意两点之间的最短路程更新为：



        最后允许通过所有顶点作为中转，任意两点之间最终的最短路程为：



        整个算法过程虽然说起来很麻烦，但是代码实现却非常简单，核心代码只有五行：

[IMG_268](http://www.cnblogs.com/wangyuliang/p/javascript:void(0);)

1 for(k=1;k<=n;k++) 2 for(i=1;i<=n;i++) 3 for(j=1;j<=n;j++) 4 if(e[i][j]>e[i][k]+e[k][j]) 5 e[i][j]=e[i][k]+e[k][j];

[IMG_269](http://www.cnblogs.com/wangyuliang/p/javascript:void(0);)

这段代码的基本思想就是：最开始只允许经过1号顶点进行中转，接下来只允许经过1和2号顶点进行中转……允许经过1~n号所有顶点进行中转，求任意两点之间的最短路程。用一句话概括就是：从i号顶点到j号顶点只经过前k号点的最短路程。

[IMG_270](http://www.cnblogs.com/wangyuliang/p/javascript:void(0);)

1 #include 2 int main() 3 { 4 int e[10][10],k,i,j,n,m,t1,t2,t3; 5 int inf=99999999; //用inf(infinity的缩写)存储一个我们认为的正无穷值 6 //读入n和m，n表示顶点个数，m表示边的条数 7 scanf("%d %d",&n,&m); 8 //初始化 9 for(i=1;i<=n;i++) 10 for(j=1;j<=n;j++) 11 if(i==j) e[i][j]=0; 12 else e[i][j]=inf; 13 //读入边14 for(i=1;i<=m;i++) 15 { 16 scanf("%d %d %d",&t1,&t2,&t3); 17 e[t1][t2]=t3; 18 } 19 //Floyd-Warshall算法核心语句20 for(k=1;k<=n;k++) 21 for(i=1;i<=n;i++) 22 for(j=1;j<=n;j++) 23 if(e[i][j]>e[i][k]+e[k][j] ) 24 e[i][j]=e[i][k]+e[k][j]; 25 //输出最终的结果26 for(i=1;i<=n;i++) 27 { 28 for(j=1;j<=n;j++) 29 { 30 printf("%10d",e[i][j]); 31 } 32 printf("\n"); 33 } 34 return 0; 35 }

[IMG_271](http://www.cnblogs.com/wangyuliang/p/javascript:void(0);)

        另外需要注意的是：Floyd-Warshall算法不能解决带有“负权回路”（或者叫“负权环”）的图，因为带有“负权回路”的图没有最短路。例如下面这个图就不存在1号顶点到3号顶点的最短路径。因为1->2->3->1->2->3->…->1->2->3这样路径中，每绕一次1->-2>3这样的环，最短路就会减少1，永远找不到最短路。其实如果一个图中带有“负权回路”那么这个图则没有最短路。

