/\*\*

\* prim.c 普里姆算法(邻接矩阵实现)

\* @author Darbuly 2018-2019

\* @copyright 2018-2019 Darbuly

\*/

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define VRType int

#define InfoType int

#define VertexType char

#define MAX\_VERTEX\_NUM 20

#define MAX\_VALUE 65535

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

typedef struct ArcCell{

VRType adj;

InfoType \*info;

}ArcCell,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 number of vertex:\n");

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

getchar();

printf("Please enter the 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 MinSpanTree\_Prim(MGraph G)

{

int min,i,j,k;

int adjvex[MAX\_VERTEX\_NUM];

int lowcost[MAX\_VERTEX\_NUM];

printf("\nMinSpanTree\_Prim\n");

lowcost[0]=0;

adjvex[0]=0;

for(i=1;i<G.vexnum;i++)

{

lowcost[i] = G.arc[0][j].adj;

adjvex[i] = 0;

}

for(i=1;i<G.vexnum;i++)

{

min = MAX\_VALUE;

j=1;

k=0;

while(j<G.vexnum)

{

if(lowcost[j]!=0 && lowcost[j]<min)

{

min = lowcost[j];

k=j;

}

j++;

}

printf("(%d,%d)",adjvex[k],k);

lowcost[k]=0;

for(j=1;j<G.vexnum;j++)

{

if(lowcost[j]!=0 && G.arc[k][j].adj <lowcost[j])

{

lowcost[j] = G.arc[k][j].adj;

adjvex[j]=k;

}

}

}

printf("\n");

}

int main()

{

MGraph graph;

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

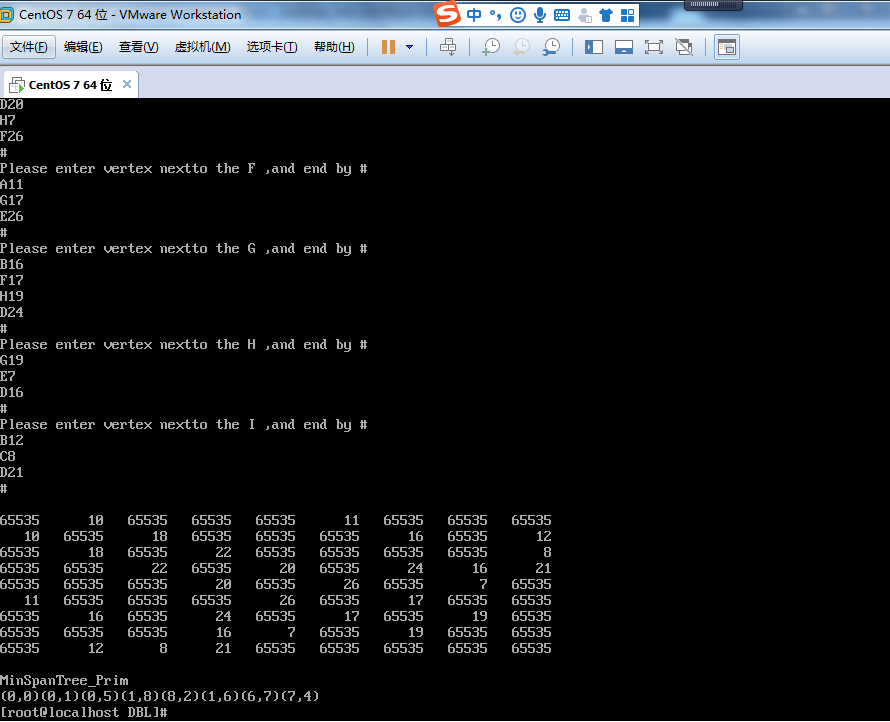
g\_create(&graph);

MinSpanTree\_Prim(graph);

return 0;

}





代码分析：

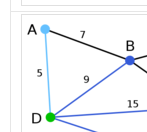
1. 时间复杂度：

有n个元素，每个元素都要循环2n次。，所以是O(n^2);\

1. 思路分析
   1. 三角形原则

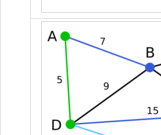
走过3个点，怎么才是最短距离？

固然是避免三角形中最长边



走DAB三个点，怎么走？

肯定是这样



故这段代码的好处

for(j=1;j<G.vexnum;j++)

{

if(lowcost[j]!=0 && G.arc[k][j].adj <lowcost[j])

{

lowcost[j] = G.arc[k][j].adj;

adjvex[j]=k;

}

}

1. 就是躲过最长边的选择。
2. 渗透式的层层切入，对比不同边，选出必须走的最优边，把必要走的路线优先划出来++++++必要性思维的优越性!!!!!!!
3. 特别适用于稠密网，因为它是渗透式的，如同灌水一般，而稠密网就像海绵一样，吸水。
4. 就像盲人摸象，类似于贪心算法，遇到问题又退出来，又体现了回溯法的思想