图结构及其应用

1. 需求分析

图是一种比树和表要复杂得多的数据结构。在图形结构中，结点之间的关系可以是任意的。图中的任意结点之间的两个数据元素都可以相关。

给出一张某公园的导游图,游客通过终端询问可知:

1）从某一景点到另一个景点的最短路径;

2）游客从公园大门进入,选一条最佳路线,使游客可以不重复的游览各景点,最后回到出口。

3）将导游图看作一张带权无向图,顶点表示公园的各个景点,边表示各景点之间的道路,边上

的权值表示距离,选择适当的数据结构。

4）为游客提供图中任意景点相关信息的查询。

5）为游客提供任意两个景点之间最短的简单路径。

6）为游客选择最佳游览路径。

(1) 输入的形式和输入值的范围：int 整型

(2) 输出的形式：int 整型

(3) 程序所能达到的功能：实现图的两种储存方式（邻接矩阵

和邻接表),利用Prim算法或Kruskal算法求图的最小生成树，求图的最小生成树（MST），实现导游图功能。

(4) 测试数据：见下面测试环节中示例图片

2. 概要设计

(1)临接矩阵图的基本操作：

void InitGrap(Grap &Grap,int vex_num,int arc_num)

操作结果:构造了一个含有vex_num个顶点和arc_num条边的无向图。

void DFStraverse(Grap &G,int num)

初始条件：图已存在。

操作结果：按深度优先遍历该图。

void BFStarverse(Grap &G,int num)

初始条件：图已存在。

操作结果：按深度优先遍历该图。

(2)临接表图的基本操作：

void InitGrap(Grap &Grap,int vex_num,int arc_num)

操作结果:构造了一个含有vex_num个顶点和arc_num条边的无向图。

void DFStraverse(Grap &G,int num)

初始条件：图已存在。

操作结果：按深度优先遍历该图。

void BFStarverse(Grap &G,int num)

初始条件：图已存在。

操作结果：按深度优先遍历该图。

(3)DFS 和 BFS 简单测试（两种存储结构测试方法相似）

main()

操作结果：调用相应函数，执行相应操作，输出所有信息。

(4) Prim 算法求图的最小生成树

void MST(Grap G,int max)

初始条件：图已存在。

操作结果：在控制台直接打印最小生成树。

int minium(closedge closedge[],int n)

初始条件：图已存在。

操作结果：求出所有待求路径中最短的一条。

(5) 旅游导航图

void ShowPath(Grap G,int v0,int vn)

初始条件：图已存在。

操作结果：求出顶点 v0 到 vn 的最短路径，并在控制台打印最短路径的路径和长度。

3. 详细设计

（1）辅助链式队列———头文件

```
#include <iostream>

using namespace std;



typedef struct QNode{


    int data;
    QNode *next;
}QNode,*Queeue;

QNode *head,*rear;

void InitQueeue(Queeue &Q){
    head = (QNode *)malloc(sizeof(QNode));
    Q = head;
    rear = head;
    head->next = head;

}

int GetlenthQueeue(Queeue &q)
{
```

```
    QNode *p = head;
    int num=0;
    while(p->next!=head){
        p=p->next;
        num++;
    }
    return num;
}
int IsEmptyQueeue(Queeue &Q)
{
    if(head->next == head)return 1;
    else return 0;
}

void EnQueeue(Queeue &Q,int data){
    QNode *newnode = (QNode *)malloc(sizeof(QNode));


    rear->next = newnode;
    newnode->next = head;
    rear = newnode;//采用尾插法入队

    newnode->data = data;

}

int DeQueeue(Queeue &Q,int &data){


    if(head->next==head)return 0;//队列为空


    QNode *oldnode = head->next;

    if(oldnode == rear)rear=head;

    head->next = oldnode->next;

    data = oldnode->data;
    free(oldnode);
    return 1;

}
```

（2）临接矩阵图的基本内容———头文件

```
#include "Queeue.h"
#include <iostream>
using namespace std;


typedef struct {
    //顶点信息
    int no;


}Node1;

typedef struct {

    double weight;//边的权重


}Arc,ArcMatrix[20][20];

typedef struct{

    Node1 vexs[20];//顶点信息
    ArcMatrix arc;//边的权重
    int vex_num,arc_num;//顶点数、边数
}Grap;



void InitGrap(Grap &Grap,int vex_num,int arc_num)
{

    Grap.vex_num=vex_num;


    Grap.arc_num=arc_num;
```

```
cout<<"请依次输入每个顶点的信息："<<endl;

for(int i=0;i<vex_num;i++)
{
    int num;
    cout<<"请输入第"<<i+1<<"个点的编号";
    cin>>num;
    Grap.vexs[i].no = num;
    cout<<endl;
}
cout<<endl;


for(int i=0;i<vex_num;i++)
    for(int j = 0;j<vex_num;j++)
        Grap.arc[i][j].weight = -1;//每条边初始化为无穷



cout<<"请依次输入每条边的权重："<<endl;

for(int i=0;i<arc_num;i++)
{
    int vnum1,vnum2;
    double anum;
    cout<<"请输入第"<<i+1<<"条边的信息";
     cout<<endl;

    cout<<"请输入第 1 个点编号：";
    cin>>vnum1;
    int search1;
    for(search1=0;search1<vex_num;search1++)
        if(Grap.vexs[search1].no == vnum1)break;
    cout<<endl;

     cout<<"请输入第 2 个点编号：";
    cin>>vnum2;
    int search2;
    for(search2=0;search2<vex_num;search2++)
        if(Grap.vexs[search2].no == vnum2)break;
    cout<<endl;
```

```
          cout<<"请输入边的权重：";
         cin>>anum;
         Grap.arc[search1][search2].weight = anum;
          Grap.arc[search2][search1].weight = anum;
        cout<<endl;cout<<endl;
    }


}

 int visit0[20];

void DFS(Grap &G,int num,int vex)
{
    if(visit0[vex]==0)
    {
    visit0[vex]=1;
    cout<<G.vexs[vex].no<<" ";
    for(int i=0;i<num;i++)
        if(G.arc[vex][i].weight!=-1)DFS(G,num,i);

    }
}

void DFStraverse(Grap &G,int num)
{

    for(int i=0;i<20;i++)
        visit0[i]=0;//全部设置为未被访问

    for(int i=0;i<num;i++)
    {
        if(visit0[i]==0)
        {
          DFS(G,num,i);
            cout<<endl;
        }

    }
}
```

```
void BFStarverse(Grap &G,int num)
{
    int visit1[20];
    for(int i=0;i<20;i++)
        visit1[i]=0;//全部设置为未被访问
    Queeue Q;
    InitQueeue(Q);

    for(int i=0;i<num;i++)
        {
            if(visit1[i]==0)
            {
                visit1[i]=1;
                cout<<G.vexs[i].no<<" ";
                EnQueeue(Q, i);
                int vex;
                while(!IsEmptyQueeue(Q))
                {


                    DeQueeue(Q, vex);

                    for(int j=0;j<num;j++)
                    {
                        if(G.arc[vex][j].weight!=-1 && visit1[j]==0)
                        {
                            EnQueeue(Q, j);
                            visit1[j]=1;
                            cout<<G.vexs[j].no<<" ";

                        }


                    }

                }
                cout<<endl;
            }
```

```
        }

}
```

（3）临接表图的基本内容———头文件

```
#include <iostream>
#include "Queeue.h"
using namespace std;



typedef struct arc{
    //弧的信息
    double weight;
    int no;//指向的下一个顶点编号
    arc *next;

}arc;


typedef struct {
    //顶点信息
    int no;
    arc *head;//顶点引出的弧

}node[20];



typedef struct{
    //
    int vex_num,arc_num;
    node Node_List;
```

```
}Grap;


void InitGrap(Grap &G,int vex_num,int arc_num)
{

    G.vex_num = vex_num;
    G.arc_num = arc_num;
    cout<<"请依次输入每个顶点的信息："<<endl;

    for(int i=0;i<vex_num;i++)
    {
        int num;
        cout<<"请输入第"<<i+1<<"个点的编号";
        cin>>num;
        G.Node_List[i].no = num;
        cout<<endl;
    }
    cout<<endl;









    cout<<"请依次输入每条边的权重："<<endl;

    for(int i=0;i<arc_num;i++)
    {
        int vnum1,vnum2;
        double anum;
        cout<<"请输入第"<<i+1<<"条边的信息";
        cout<<endl;

        cout<<"请输入第 1 个点编号：";
        cin>>vnum1;
        int search1;
        for(search1=0;search1<vex_num;search1++)
            if(G.Node_List[search1].no == vnum1)break;
        cout<<endl;

        cout<<"请输入第 2 个点编号：";
```

```
        cin>>vnum2;
        int search2;
        for(search2=0;search2<vex_num;search2++)
            if(G.Node_List[search2].no == vnum2)break;
        cout<<endl;

        cout<<"请输入边的权重：";
        cin>>anum;
        arc *newnode1 = (arc*)malloc(sizeof(arc));
        if(G.Node_List[search1].head==NULL)
        {
            G.Node_List[search1].head = newnode1;
            newnode1->next = NULL;
        }
        else{
            newnode1->next = G.Node_List[search1].head;
            G.Node_List[search1].head = newnode1;
            }
        newnode1->weight = anum;
        newnode1->no = search2;

        arc *newnode2 = (arc*)malloc(sizeof(arc));
        if(G.Node_List[search2].head==NULL)
        {
            G.Node_List[search2].head = newnode2;
            newnode2->next = NULL;
        }
        else{
            newnode2->next = G.Node_List[search2].head;
            G.Node_List[search2].head = newnode2;
        }
        newnode2->weight = anum;
        newnode2->no = search1;
        cout<<endl;
    }




}
```

```
int visit[20];
void DFS(Grap &G,int num)
{
    if(visit[num]==0)
    {
        visit[num]=1;
    cout<<G.Node_List[num].no<<" ";
    for(arc *find=G.Node_List[num].head;find;find=find->next)
        DFS(G,find->no);
    }
}

void DFStraverse(Grap &G,int num)
{
    for(int i=0;i<num;i++)
        visit[i] = 0;
    for(int i=0;i<num;i++)
        if(visit[i]==0)
        {

                DFS(G,i);
            cout<<endl;
        }
}


void BFStarverse(Grap &G,int num)
{
    int visit0[20];
    for(int i=0;i<num;i++)
        visit0[i]=0;


    Queeue Q;
    InitQueeue(Q);

    for(int i=0;i<num;i++)
    {
        if(visit0[i]==0)
        {
            visit0[i]=1;
            cout<<G.Node_List[i].no<<" ";
            EnQueeue(Q, i);
            while(IsEmptyQueeue(Q)==0)
```

```
            {
                int vex;
                DeQueeue(Q, vex);
    for(arc *find = G.Node_List[vex].head;find!=NULL;find=find->next)
                    if(visit0[find->no]==0)
                        {
                            visit0[find->no]=1;
                            cout<<G.Node_List[find->no].no<<" ";
                            EnQueeue(Q, find->no);

                        }
            }
            cout<<endl;
        }
    }


}
```

（4）两种图 BFS 和 DFS 遍历测试文件

```
#include <iostream>
#include "Grap.h"
using namespace std;

int main() {

    int vex_num,arc_num;
    cout<<"请输入总的顶点数：";
    cin>>vex_num;
    cout<<endl;
    cout<<"请输入总的边数：";
    cin>>arc_num;
    cout<<endl;
    Grap G;
    InitGrap(G, vex_num, arc_num);
    cout<<endl<<"深度优先便利："<<endl;
```

```
    DFStraverse(G, vex_num);
        cout<<endl<<"广度优先便利："<<endl;
    BFStarverse(G, vex_num);
    return 0;
}
```

（4）最小生成树———头文件

```
#include"Graph.h"

typedef struct{

    int no;
    double lowcost;

}closedge;


int minium(closedge closedge[],int n)
{
    int min = 1000;
    int no ;
    for(int i=0;i<n;i++)
    {
    if(closedge[i].lowcost!=0&&closedge[i].lowcost<=min )
        {
            min = closedge[i].lowcost;
           no = i;
        }

    }
     return no;
}



void MST(Grap G,int max)
{
    closedge  closedge[max];
    for(int j=1;j<G.vex_num;j++)
    {

            closedge[j].no = G.vexs[0].no;
        closedge[j].lowcost = G.arc[j][0].weight;
```

```
    }
    closedge[0].lowcost=0;


    for(int i=1;i<G.vex_num;i++)
    {
        int k=minium(closedge,max);

        cout<<closedge[k].no<<"---"<<G.vexs[k].no<<"
"<<closedge[k].lowcost<<endl;
        closedge[k].lowcost=0;
        for(int j=0;j<G.vex_num;j++)
        {
            if(G.arc[k][j].weight<closedge[j].lowcost)
            {closedge[j].no = G.vexs[k].no;
                closedge[j].lowcost = G.arc[k][j].weight;}
        }

    }


}
```

（4）旅游图头文件

```
#include"Graph.h"
#include <iostream>

using namespace std;




void ShowPath(Grap G,int v0,int vn)
{
    int P[G.vex_num][G.vex_num];
    int D[G.vex_num];
    int finally[G.vex_num];
```

```
    for(int i=0;i<G.vex_num;i++)//初始化
    {
        finally[i]=0;
        D[i]=G.arc[v0][i].weight;

        for(int j=0;j<G.vex_num;j++)
            P[i][j]=0;

        if(D[i]<1000)//判断 v0 到 i 有没有路径
        {
            P[i][v0]=1;
            P[i][i]=1;


        }
    }
    D[v0]=0;
    finally[v0]=1;
    int v;//v 保存到 v0 路径最短的点
    for(int i=1;i<G.vex_num;i++)
    {
        int min = 1000;

        for(int w=0;w<G.vex_num;w++)
        { if(!finally[w])
            if(D[w]<min)
            {
                min = D[w];
                v = w;
            }

        }
        finally[v]=1;

        for(int w=0;w<G.vex_num;w++)
        {
            if(!finally[w]&&(D[w]>min+G.arc[v][w].weight))
                //如果 w 没有被并入最短路径集合，并且通过 v 到 w 的路径更
短
            {
                D[w]=min+G.arc[v][w].weight;
                for(int j=0;j<G.vex_num;j++)
```

```
                {//将 v 的路径赋值给 P[w]
                    if(P[v][j]==1)P[w][j]=1;
                }
                P[w][w]=1;
            }

        }



    }


    cout<<"最短游览路径为一次经过下面景点的路径："<<endl;
    for(int i=0;i<G.vex_num;i++)
    {
        if(P[vn][i]==1)cout<<" "<<i+1<<" ";
    }

    cout<<endl<<"最短游览路径长度为："<<D[vn];


}
```

（4）旅游图菜单及测试文件

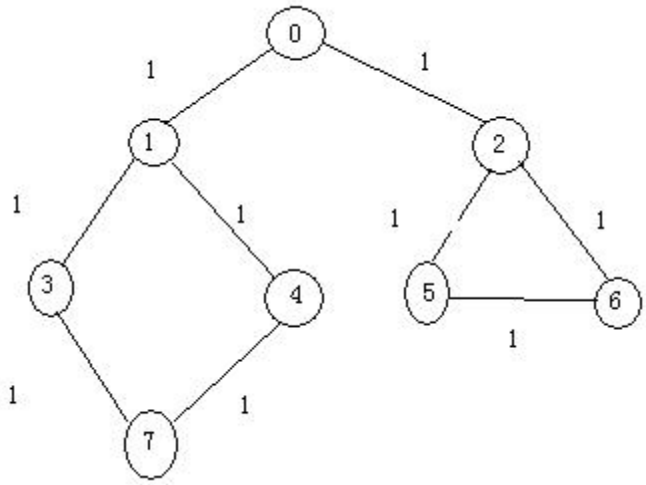
```
#include "Guide.h"
#include <iostream>
using namespace std;

int main() {

    int vex_num,arc_num;
    cout<<"请输入总的顶点数：";
    cin>>vex_num;
    cout<<endl;
    cout<<"请输入总的边数：";
    cin>>arc_num;
    cout<<endl;
    Grap G;
    InitGrap(G, vex_num, arc_num);
```

```
int vo,vn;


while(true)
{
    cout<<endl;

    cout<<"=======">>"旅游导航图"<<"=======">>endl;
    cout<<"   1.最佳游览路径"<<endl<<endl;
    cout<<"   2.景点最短路径"<<endl<<endl;
    cout<<"   3.退出"<<endl<<endl;
    cout<<"=======">>"=======">>"============">>endl;
    int i;
    cin>>i;
    switch (i) {
        case 1:
            cout<<"最佳游览路径如下："<<endl;
            DFStraverse(G, vex_num);
            break;
        case 2:
            cout<<"请输入起点和终点：";
            cin>>vo>>vn;
            int search1,search2;
            for(search1=0;search1<vex_num;search1++)
                if(G.vexs[search1].no == vo)break;
            for(search2=0;search2<vex_num;search2++)
                if(G.vexs[search2].no == vn)break;

            ShowPath(G,search1, search2);
            break;

        case 3:exit(1);break;
        default:continue;

    }



}
```

```
    return 0;
}
```

4. 调试分析

1、设置图的基本结构如下：



2、输入相应顶点数和边数：

```
请输入总的顶点数：8

请输入总的边数：9
```

3、按图输入顶点信息：

请依次输入每个顶点的信息：
请输入第1个点的编号0

请输入第2个点的编号1

请输入第3个点的编号2

请输入第4个点的编号3

请输入第5个点的编号4

请输入第6个点的编号5

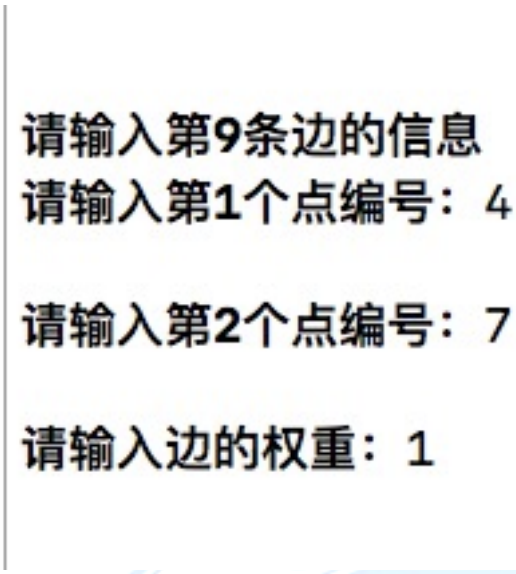
请输入第7个点的编号6

请输入第8个点的编号7

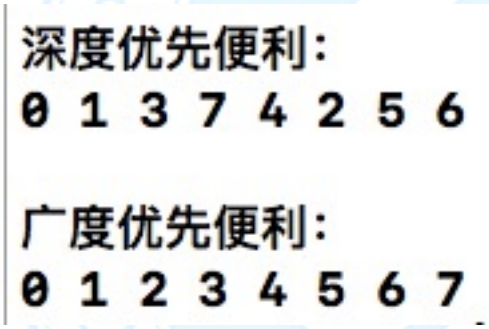
4、按图输入边的信息：

```
请依次输入每条边的权重:
请输入第1条边的信息
请输入第1个点编号: 0

请输入第2个点编号: 1

请输入边的权重: 1


请输入第2条边的信息
请输入第1个点编号: 0

请输入第2个点编号: 2

请输入边的权重: 1


请输入第3条边的信息
请输入第1个点编号: 2

请输入第2个点编号: 5

请输入边的权重: 1


请输入第4条边的信息
请输入第1个点编号: 2

请输入第2个点编号: 6

请输入边的权重: 1
```

```
请输入第5条边的信息
请输入第1个点编号：5

请输入第2个点编号：6

请输入边的权重：1


请输入第6条边的信息
请输入第1个点编号：1

请输入第2个点编号：3

请输入边的权重：1


请输入第7条边的信息
请输入第1个点编号：1

请输入第2个点编号：4

请输入边的权重：1


请输入第8条边的信息
请输入第1个点编号：3

请输入第2个点编号：7

请输入边的权重：1
```



5、输出 DFS 和 BFS 遍历结果：



测试完毕，两种存储方式的图均可正常使用。

<1>调试过程中遇到的问题：

在最早调试的时候忘记图是无向图这一情况，导致出了很多莫名其妙的问题，最后才想起来要同时修改两条边的信息；改正后图的矩阵存储基本没什么问题，可正常运行，但临接表就显示错误，后来发现是创建头指针的时候忘了一些处理，改正后可以正常使用。

<2>时间和空间分析；

邻接矩阵：矩阵包含 n^2 个元素，在算法中，共 n 个顶点，对每个顶点都要遍历 n 次，所以时间复杂度为 O(n^2)

邻接表：包含 n 个头结点和 e 个表结点，算法中对所有结点都要遍历一次，所以时间复杂度为 O（n+e）

<3>改进设想：

在有些情况下可以把矩阵中长度无限大改写为－1。

<4>经验、心得和体会：

图是学到的最为复杂的存储结构，存储方式也是多种多样，相比而言更喜欢使用矩阵来存储，虽然占用空间比较大，但使用起来简单方便，代码看起来也很直观；最小生成树的算法都很巧妙，同时在图的学习中也了解了关键路径、最短路径等等复杂的算法和新颖的问题，对以后的发展很有帮助。

5. 使用说明

<1>最小生成树

1、按要求输入各顶点信息和各边的信息。

2、输入完成后即可自动生成最小生成树。

<2>导游图

1、按要求输入各顶点信息和各边的信息。

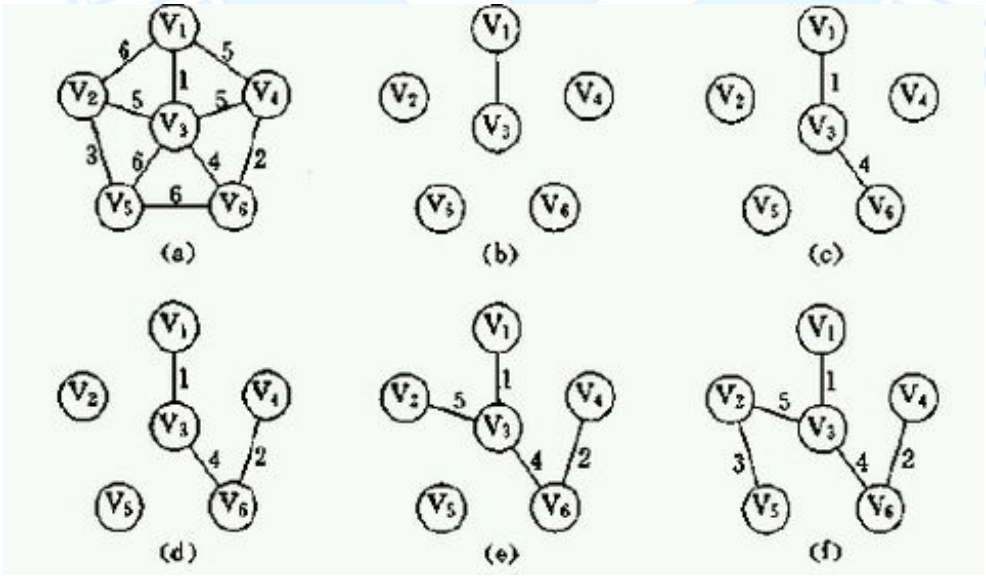
2、在菜单页输入 1 可显示最佳游览路径。

2、在菜单页输入 2 可显示任意两个景点之间最短路径。

6. 测试程序的运行结果

<1>最小生成树

（1）最小生成树图示如下：



（2）输入顶点数量和边的数量。

```
请输入总的顶点数：6

请输入总的边数：10
```

（3）输入顶点信息。

```
请依次输入每个顶点的信息：
请输入第1个点的编号1

请输入第2个点的编号2

请输入第3个点的编号3

请输入第4个点的编号4

请输入第5个点的编号5

请输入第6个点的编号6
```

（4）输入边的信息。

```
请依次输入每条边的权重:
请输入第1条边的信息
请输入第1个点编号: 1

请输入第2个点编号: 2

请输入边的权重: 6


请输入第2条边的信息
请输入第1个点编号: 1

请输入第2个点编号: 4

请输入边的权重: 5


请输入第3条边的信息
请输入第1个点编号: 1

请输入第2个点编号: 3

请输入边的权重: 1


请输入第4条边的信息
请输入第1个点编号: 2

请输入第2个点编号: 3

请输入边的权重: 5
```

请输入第5条边的信息
请输入第1个点编号：3

请输入第2个点编号：6

请输入边的权重：4

请输入第6条边的信息
请输入第1个点编号：3

请输入第2个点编号：5

请输入边的权重：6

请输入第7条边的信息
请输入第1个点编号：2

请输入第2个点编号：5

请输入边的权重：3

请输入第8条边的信息
请输入第1个点编号：5

请输入第2个点编号：6

请输入边的权重：6

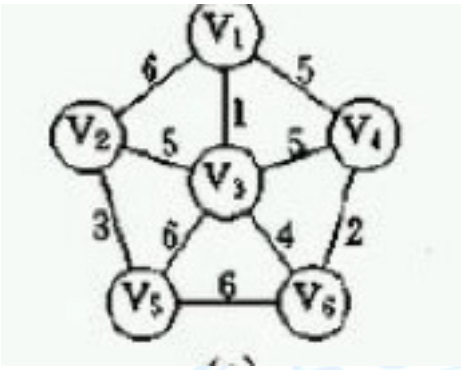
```
请输入第9条边的信息
请输入第1个点编号：4

请输入第2个点编号：6

请输入边的权重：2


请输入第10条边的信息
请输入第1个点编号：3

请输入第2个点编号：4

请输入边的权重：5
```

（2）得到的最小生成树如图。（V1———V2　权重）

```
1---3   1
3---6   4
6---4   2
3---2   5
2---5   3
```

<2>导游图

（1）景区大体结构如下：



（2）依次输入各顶点和边的信息：

```
请输入总的顶点数：6

请输入总的边数：10

请依次输入每个顶点的信息：
请输入第1个点的编号1

请输入第2个点的编号2

请输入第3个点的编号3

请输入第4个点的编号4

请输入第5个点的编号5

请输入第6个点的编号6
```

请依次输入每条边的权重：
请输入第1条边的信息
请输入第1个点编号：1

请输入第2个点编号：2

请输入边的权重：6

请输入第2条边的信息
请输入第1个点编号：1

请输入第2个点编号：4

请输入边的权重：5

请输入第3条边的信息
请输入第1个点编号：4

请输入第2个点编号：6

请输入边的权重：2

请输入第4条边的信息
请输入第1个点编号：5

请输入第2个点编号：6

请输入边的权重：6

请输入第5条边的信息
请输入第1个点编号：2

请输入第2个点编号：5

请输入边的权重：3

请输入第6条边的信息
请输入第1个点编号：1

请输入第2个点编号：3

请输入边的权重：1

请输入第7条边的信息
请输入第1个点编号：2

请输入第2个点编号：3

请输入边的权重：5

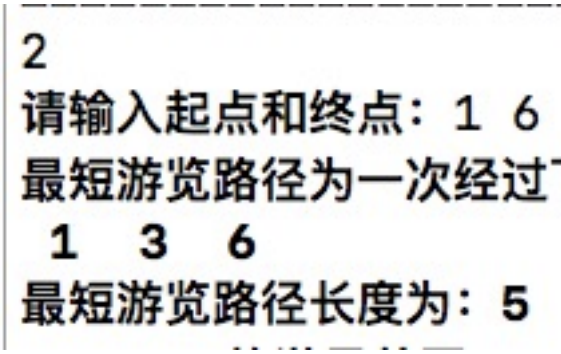
```
请输入第8条边的信息
请输入第1个点编号：3

请输入第2个点编号：4

请输入边的权重：5


请输入第9条边的信息
请输入第1个点编号：3

请输入第2个点编号：6

请输入边的权重：4


请输入第10条边的信息
请输入第1个点编号：3

请输入第2个点编号：5

请输入边的权重：6
```

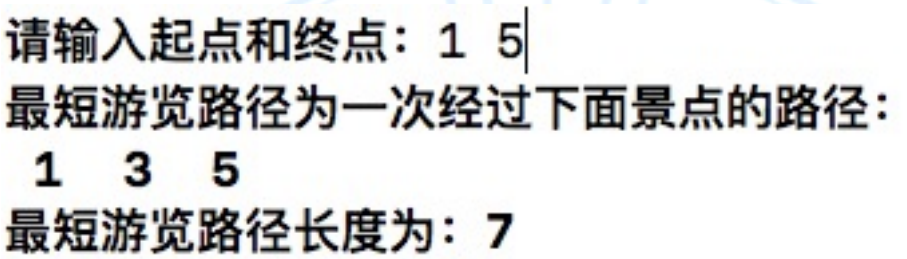
（3）在菜单页输入 1 可显示最佳游览路径。

```
=======旅游导航图=======
    1.最佳游览路径

    2.景点最短路径

    3.退出

============================
1
最佳游览路径如下:
1 2 3 4 6 5
```

（4）在菜单页输入 2 可显示任意两个景点之间最短路径。

```
=======旅游导航图=======
    1.最佳游览路径

    2.景点最短路径

    3.退出

============================
2
请输入起点和终点：1 4
最短游览路径为一次经过下面景点的路径:
 1  4
最短游览路径长度为：5
```





7. 心得体会

从前根本就没能想到怎么用计算机来存储图的结构，这次算是学会了，原来链表还能当作临接表来使用，而且矩阵在计算机中的作用竟然这么大，第一次接触图的代码也是很吃力，这四个代码写了好几周才完成，也算是对图知识的巩固和温习，以后生活生产中离不开图的操作，书上更多代码我也争取在课余时间都亲手实现试一试，这样动手操作对于数据结构的学习更有帮助。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在 bin 子目录。

文件清单如下：

文件夹 Map……临接矩阵的存储方式存储图，并进行简单

测试

文件夹 Map_adj……临接表的存储方式存储图，并进行简单测试

文件夹 MST……最小生成树的实现，并进行简单测试

文件夹 Guide_Map……导游图的实现，并进行简单测试