中南大学

数据结构试验报告

题 目 实验五

学生姓名 王云鹏

学 号 8213180228

指导老师 郑 瑾

学 院 计算机学院

专业班级 物联网1802

完成时间 2020.6

指导老师评定 签名

实验

1. 需求分析

的任意结点之间的两个数据元素都可以相关。本实验希望读者理解图的数据结构，掌握图的邻接表存

储结构，建立邻接表的算法，以及如何应用图解决具体问题（即原理与应用的结合）等。

1．从键盘输入的数据建立图，并进行深度优先搜索和广度优先搜索（验证性实验）

问题描述

很多涉及图上操作的算法都是以图的遍历操作为基础的。试编写一个程序，演示无向图的遍历

操作。

在主程序中提供下列菜单：

(1)“1”代表图的建立；

(2)“2”代表深度优先遍历图；

(3)“3”代表广度优先遍历图；

(4)“0”代表结束。

基本要求

以邻接表为存储结构，实现连通无向图的深度优先和广度优先遍历。以用户指定的结点为起点，

分别输出每种遍历下的结点访问序列和相应生成树的边集。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如单个结点。

实现提示

设图的结点不超过30个，每个结点用一个编号表示（如果一个图有*n*个结点，则它们的编号分别

为1, 2, …, *n*）。通过输入图的所有边输入一个图，每个边为一个数对，可以对边的输入顺序作出某种

限制。注意，生成树的边是有向边，端点顺序不能颠倒。

2．利用最小生成树算法解决通信网的总造价最低问题（设计性实验）

问题描述

若在*n*个城市之间建通信网络，只需架设*n*−1条线路即可。如何以最低的经济代价建设这个通信网

是一个网的最小生成树问题。

基本要求

以邻接表为存储结构，利用Prim算法或Kruskal算法求网的最小生成树。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如单个结点。

实现提示

设通信线路一旦建立，必然是双向的。因此，构造最小生成树的网一定是无向网。为简单起见，

图的顶点数不超过10，网中边的权值设置成小于100。

4．导游问题（综合性实验）

问题描述

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。

基本要求

(1) 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。

(2) 为游客提供图中任意景点相关信息的查询。

(3) 为游客提供任意两个景点之间最短的简单路径。

(4) 为游客选择最佳游览路径。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如单个结点。

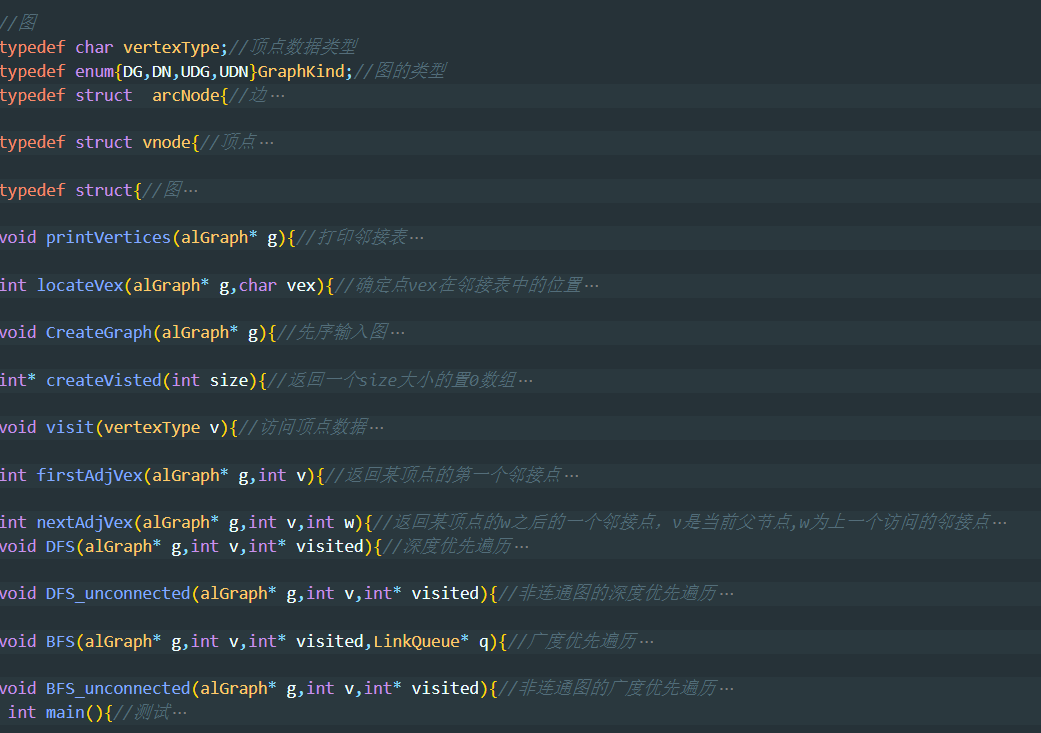
实现提示

以邻接表为存储结构，利用Dijkstra算法或Floyd算法求最短路径，利用搜索求最佳路径

1. 概要设计

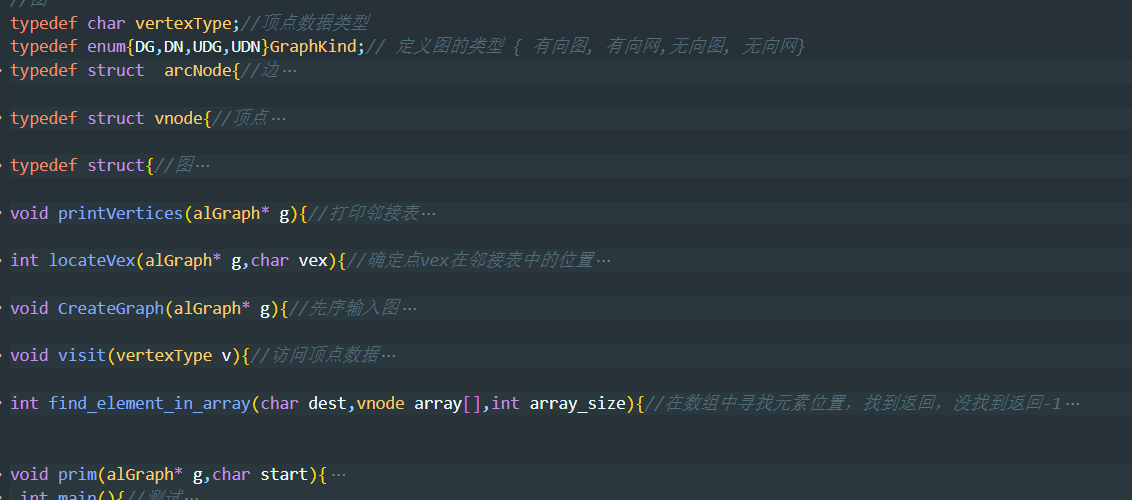
* 验证性实验

函数：



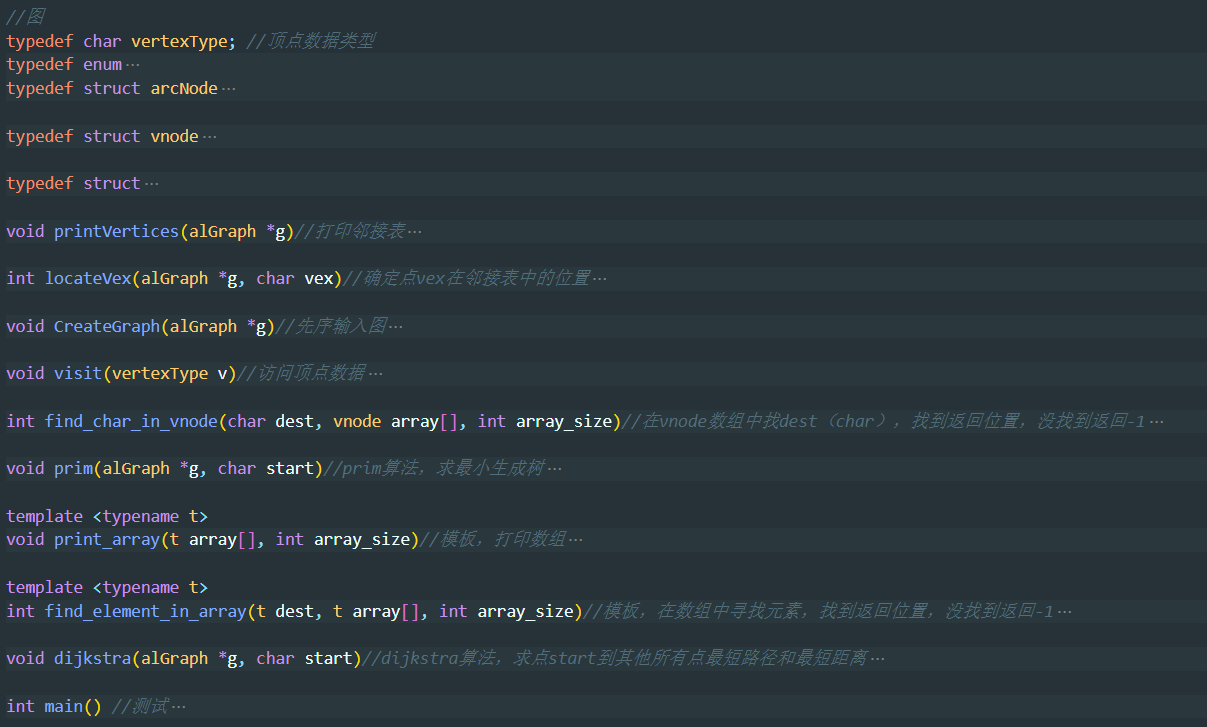
* 设计性实验

函数：



* 综合性实验

函数：



1. 详细设计

* 验证性实验

|  |
| --- |
| *#include*<stdio.h>  *#include*<stdlib.h>  *#include*<string.h>  *#define* MAX\_VERTEX\_NUM 30*//图的最大顶点数目*  *//队列*  *#define* ok 0  *#define* fail 1  *#define* underflow -1  typedef int QElemType;  typedef struct QNode  { *//节点*      QElemType data;      struct QNode \*next; *//链队列*  } QNode;  typedef struct  {      QNode \*front; *//头指针，带空头节点*      QNode \*rear;  *//尾节点*  } LinkQueue;  int InitQueue(LinkQueue \*q)  {                                                        *//初始化*      q->front = q->rear = (QNode \*)malloc(sizeof(QNode)); *//申请空间，front指向头节点（空的）*  *if* (q->front == NULL)      { *//若是申请失败*          printf("malloc fail in InitQueue");  *return* fail;      }      q->front->next = NULL; *//空队的头节点下一个自然为空*  *return* ok;  }  int EnQueue(LinkQueue \*q, QElemType e)  {                                              *//插入*      QNode \*p = (QNode \*)malloc(sizeof(QNode)); *//将要插入的节点*  *if* (p == NULL)      {          printf("malloc fail in EnQueue");  *return* fail;      }      p->data = e;       *//p的值为e*      p->next = NULL;    *//在尾部插入，其下一个为空*      q->rear->next = p; *//接到尾节点的下一个*      q->rear = p;       *//p为新的尾节点*  *return* ok;  }  int DeQueue(LinkQueue \*q, QElemType \*e)  { *//删除*  *if* (q->front == q->rear)      { *//空队无法删除*          printf("underflow\n");  *return* underflow;      }      QNode \*p = q->front->next; *//需要删除的节点*      \*e = p->data;              *//用e带出删掉的值*      q->front->next = p->next;  *//将要删除的节点从链中断开*  *if* (q->rear = p)      {                       *//加入原来队中只有一个节点*          q->rear = q->front; *//删完后为空队，头节点与尾节点指到一起*      }      free(p);  *return* ok;  }  *#define* empty 0  *#define* notempty 1  int isempty(LinkQueue q){  *if*(q.front==q.rear){  *return* empty;      }*else*{  *return* notempty;      }  }  *//图*  typedef char vertexType;*//顶点数据类型*  typedef enum{DG,DN,UDG,UDN}GraphKind;*//图的类型*  typedef struct  arcNode{*//边*      int adjvex;*//边上数据，指向其父节点下一个邻接点在顶点数组中的位置*      struct arcNode \*nextarc;*//指向下一条边*  }arcNode;  typedef struct vnode{*//顶点*      vertexType vexData;*//顶点数据*      arcNode \*firstarc;*//指向第一个邻接点的边*  }vnode;  typedef struct{*//图*      vnode\* vertices;*//顶点数组*      int vexnum,arcnum;*//顶点数目，边的数目*      GraphKind kind;*//图类型*  }alGraph;  void printVertices(alGraph\* g){*//打印邻接表*  *for*(int i=0;i<g->vexnum;i++){*//遍历顶点数组*          printf("%c ",g->vertices[i].vexData);  *//遍历每个顶点的所有边*          arcNode\* a=(arcNode\*)malloc(sizeof(arcNode));          a=g->vertices[i].firstarc;  *while*(a!=NULL){              printf("%d ",a->adjvex);              a=a->nextarc;          }          printf("\n");      }  }  int locateVex(alGraph\* g,char vex){*//确定点vex在邻接表中的位置*  *for*(int i=0;i<g->vexnum;i++){*//遍历比较*  *if*(g->vertices[i].vexData==vex){  *return* i;          }      }      printf("vex not exist in locateVex:%c\n",vex);  *return* -1;*//没找到返回-1*  }  void CreateGraph(alGraph\* g){*//先序输入图*      printf("输入顶点数，边数和图类\n");      scanf("%d%d%d",&(g->vexnum),&(g->arcnum),&(g->kind));      g->vertices=(vnode\*)malloc(g->vexnum\*sizeof(vnode));  *// printf("%d %d %d",g->vexnum,g->arcnum,g->kind);*      printf("输入顶点\n");  *for*(int i=0;i<g->vexnum;i++){          printf("第%d个顶点：\n",i+1);          getchar();*//清掉回车*          g->vertices[i].vexData=getchar();  *// putchar(g->vertices[i].vexData);*          g->vertices[i].firstarc=NULL;*//尾部置空*      }  *// printVertices(g);*      printf("输入边\n");  *for*(int i=0;i<g->arcnum;i++){          char sv;          char tv;          getchar();*//清掉回车*          printf("第%d条边：\n",i+1);          scanf("%c%c",&sv,&tv);  *// printf("%c,%c\n",sv,tv);*          int s=locateVex(g,sv);          int t=locateVex(g,tv);          arcNode\* pi=(arcNode\*)malloc(sizeof(arcNode));          pi->adjvex=t;          pi->nextarc=g->vertices[s].firstarc;*//头插法*          g->vertices[s].firstarc=pi;  *if*(g->kind==UDG || g->kind==UDN){*//若是无向图或无向网，在边的另一端也要加入边*              arcNode\* pj=(arcNode\*)malloc(sizeof(arcNode));              pj->adjvex=s;              pj->nextarc=g->vertices[t].firstarc;*//头插法*              g->vertices[t].firstarc=pj;          }      }  *// printVertices(g);*  }  int\* createVisted(int size){*//返回一个size大小的置0数组*      int\* visited=(int\*)malloc(size\*sizeof(int));      memset(visited,0,size\*sizeof(int));  *return* visited;  }  void visit(vertexType v){*//访问顶点数据*      printf("%c",v);  }  int firstAdjVex(alGraph\* g,int v){*//返回某顶点的第一个邻接点*  *if*(g->vertices[v].firstarc==NULL)*return* -1;*//无邻接点时，返回-1*  *return* g->vertices[v].firstarc->adjvex;  }  int nextAdjVex(alGraph\* g,int v,int w){*//返回某顶点的w之后的一个邻接点，v是当前父节点,w为上一个访问的邻接点*      arcNode\* a=(arcNode\*)malloc(sizeof(arcNode));      a=g->vertices[v].firstarc;  *if*(a==NULL)*return* -1;  *while*(a->adjvex!=w){*//遍历寻找w*  *if*(a==NULL)*return* -1;*//到底返回-1*          a=a->nextarc;      }  *if*(a->nextarc==NULL)*return* -1;*//若是最后一个点，返回-1*  *return* a->nextarc->adjvex;*//找到就返回下一个邻接点*  }  void DFS(alGraph\* g,int v,int\* visited){*//深度优先遍历*      visited[v]=1;*//已访问，置1*      visit(g->vertices[v].vexData);  *for*(int w=firstAdjVex(g,v);w!=-1;w=nextAdjVex(g,v,w)){*//遍历所有邻接点，深度优先访问它们*  *// printf("#%d#",w);*  *if*(visited[w]==0){              DFS(g,w,visited);          }      }  }  void DFS\_unconnected(alGraph\* g,int v,int\* visited){*//非连通图的深度优先遍历*  *for*(int i=0;i<g->vexnum;i++){*//遍历该图所有点，对未访问的顶点，深度优先遍历*  *// printf("%d",i);*  *if*(visited[i]==0){              DFS(g,i,visited);          }  *// printf("hhh");*      }      printf("\n");  }  void BFS(alGraph\* g,int v,int\* visited,LinkQueue\* q){*//广度优先遍历*      visited[v]=1;      visit(g->vertices[v].vexData);      EnQueue(q,v);*//入队要访问的顶点*  *while*(isempty(\*q)==notempty){*//当队列不为空时*          int u;          DeQueue(q,&u);*//从队列取出要访问的点，置为u*  *for*(int w=firstAdjVex(g,u);w!=-1;w=nextAdjVex(g,u,w)){*//遍历u的所有邻接点*  *if*(visited[w]==0){*//若未访问就访问*                  visited[w]=1;                  visit(g->vertices[w].vexData);                  EnQueue(q,w);*//将u的邻接点入队*              }          }      }  }  void BFS\_unconnected(alGraph\* g,int v,int\* visited){*//非连通图的广度优先遍历*      LinkQueue\* q=(LinkQueue\*)malloc(sizeof(LinkQueue));*//设置一个队列，用来存放将要访问的顶点*      InitQueue(q);  *for*(int v=0;v<g->vexnum;v++){*//遍历该图所有点，广度优先访问*  *if*(visited[v]==0){              BFS(g,v,visited,q);          }      }  }   int main(){*//测试*       alGraph\* g=(alGraph\*)malloc(sizeof(alGraph));  *if*(g==NULL){           printf("malloc fail in main for g\n");  *return* -1;       }       CreateGraph(g);  *//  printVertices(g);//查看邻接表*       printf("输入遍历起点：\n");       getchar();*//清掉回车*       char start=getchar();       printf("深度优先遍历结果：\n");  *//  DFS(g,locateVex(g,start),createVisted(g->vexnum));*       DFS\_unconnected(g,locateVex(g,start),createVisted(g->vexnum));       printf("广度优先遍历结果：\n");  *//  BFS(g,locateVex(g,start),createVisted(g->vexnum));*      BFS\_unconnected(g,locateVex(g,start),createVisted(g->vexnum));  *return* 0;   } |

* 设计性实验

|  |
| --- |
| *#include*<iostream>  *#include*<string>  *#include*<cstring>  using namespace std;  *#define* MAX\_VERTEX\_NUM 30*//图的最大顶点数目*  *//图*  typedef char vertexType;*//顶点数据类型*  typedef enum{DG,DN,UDG,UDN}GraphKind;*// 定义图的类型 { 有向图, 有向网,无向图, 无向网}*  typedef struct  arcNode{*//边*      int adjvex;*//边上数据，指向其父节点下一个邻接点在顶点数组中的位置*      int weight;*//边权重*      struct arcNode \*nextarc;*//指向下一条边*  }arcNode;  typedef struct vnode{*//顶点*      vertexType vexData;*//顶点数据*      arcNode \*firstarc;*//指向第一个邻接点的边*  }vnode;  typedef struct{*//图*      vnode\* vertices;*//顶点数组*      int vexnum,arcnum;*//顶点数目，边的数目*      GraphKind kind;*//图类型*  }alGraph;  void printVertices(alGraph\* g){*//打印邻接表*  *for*(int i=0;i<g->vexnum;i++){*//遍历顶点数组*          printf("%c ",g->vertices[i].vexData);  *//遍历每个顶点的所有边*          arcNode\* a=(arcNode\*)malloc(sizeof(arcNode));          a=g->vertices[i].firstarc;  *while*(a!=NULL){              printf("%d&%d ",a->adjvex,a->weight);              a=a->nextarc;          }          printf("\n");      }  }  int locateVex(alGraph\* g,char vex){*//确定点vex在邻接表中的位置*  *for*(int i=0;i<g->vexnum;i++){*//遍历比较*  *if*(g->vertices[i].vexData==vex){  *return* i;          }      }      printf("vex not exist in locateVex:%c\n",vex);  *return* -1;*//没找到返回-1*  }  void CreateGraph(alGraph\* g){*//先序输入图*      printf("输入顶点数，边数和图类\n");      scanf("%d%d%d",&(g->vexnum),&(g->arcnum),&(g->kind));      g->vertices=(vnode\*)malloc(g->vexnum\*sizeof(vnode));      printf("输入顶点\n");  *for*(int i=0;i<g->vexnum;i++){          printf("第%d个顶点：\n",i+1);          getchar();*//清掉回车*          g->vertices[i].vexData=getchar();          g->vertices[i].firstarc=NULL;*//尾部置空*      }      printf("输入边和权重\n");  *for*(int i=0;i<g->arcnum;i++){          char sv;          char tv;          int w;          getchar();*//清掉回车*          printf("第%d条边和权重：\n",i+1);          scanf("%c%c",&sv,&tv);          scanf("%d",&w);          int s=locateVex(g,sv);          int t=locateVex(g,tv);          arcNode\* pi=(arcNode\*)malloc(sizeof(arcNode));          pi->adjvex=t;          pi->weight=w;          pi->nextarc=g->vertices[s].firstarc;*//头插法*          g->vertices[s].firstarc=pi;  *if*(g->kind==UDG || g->kind==UDN){*//若是无向图或无向网，在边的另一端也要加入边*              arcNode\* pj=(arcNode\*)malloc(sizeof(arcNode));              pj->adjvex=s;              pj->weight=w;              pj->nextarc=g->vertices[t].firstarc;*//头插法*              g->vertices[t].firstarc=pj;          }      }  }  void visit(vertexType v){*//访问顶点数据*      printf("%c",v);  }  int find\_element\_in\_array(char dest,vnode array[],int array\_size){*//在数组中寻找元素位置，找到返回，没找到返回-1*  *for*(int i=0;i<array\_size;i++){  *if*(dest==array[i].vexData){  *return* i;          }      }  *// cout<<"fail to find element"<<endl;*  *return* -1;  }  void prim(alGraph\* g,char start){      alGraph p;*//用邻接表存最小生成树*      p.vertices=(vnode\*)malloc(g->vexnum\*sizeof(vnode));      p.vertices[0].vexData=start;      p.vertices[0].firstarc=NULL;      p.vexnum=1;      p.arcnum=0;      p.kind=UDN;  *while*(true){  *if*(p.vexnum==g->vexnum)*break*;*//Vnew更新完则退出循环*          int min\_weight=100;*//假设所有边权重不超过100*          int min\_index=0;          int min\_index\_source=0;  *for*(int i=0;i<p.vexnum;i++){*//遍历Vnew所有边*              int index\_find=find\_element\_in\_array(p.vertices[i].vexData,g->vertices,g->vexnum);  *// cout<<index\_find<<endl;*  *if*(index\_find!=-1){*//若在g中找到该点,遍历该点所有邻接边，查找权重最小的边*                  arcNode\* a=(arcNode\*)malloc(sizeof(arcNode));                  a=g->vertices[index\_find].firstarc;  *if*(a==NULL)*continue*;  *while*(true){  *if*(a==NULL)*break*;  *if*(a->weight<min\_weight && find\_element\_in\_array(g->vertices[a->adjvex].vexData,p.vertices,p.vexnum)==-1){*//若找到更小权值的边（该边另一端点不在Vnew中）*                          min\_weight=a->weight;                          min\_index=a->adjvex;                          min\_index\_source=i;                      }                      a=a->nextarc;                  }              }          }  *//找到权重最小的之后，更新Vnew和Enew*          p.vertices[p.vexnum].vexData=g->vertices[min\_index].vexData;          p.vertices[p.vexnum].firstarc=NULL;          arcNode\* pi=(arcNode\*)malloc(sizeof(arcNode));          pi->adjvex=p.vexnum;          pi->weight=min\_weight;          pi->nextarc=p.vertices[min\_index\_source].firstarc;*//头插法*          p.vertices[min\_index\_source].firstarc=pi;  *if*(p.kind==UDG || p.kind==UDN){*//若是无向图或无向网，在边的另一端也要加入边*              arcNode\* pj=(arcNode\*)malloc(sizeof(arcNode));              pj->adjvex=min\_index\_source;              pj->weight=min\_weight;              pj->nextarc=p.vertices[p.vexnum].firstarc;*//头插法*              p.vertices[p.vexnum].firstarc=pj;          }          p.vexnum++;  *// printVertices(&p);*      }      printVertices(&p);  }   int main(){*//测试*       alGraph\* g=(alGraph\*)malloc(sizeof(alGraph));  *if*(g==NULL){           printf("malloc fail in main for g\n");  *return* -1;       }       CreateGraph(g);       printVertices(g);*//查看邻接表*       printf("输入起点：\n");       getchar();*//清掉回车*       char start=getchar();       cout<<"该图最小生成树为："<<endl;       prim(g,start);  *return* 0;   } |

|  |
| --- |
| *#include* <iostream>  *#include* <string>  *#include* <cstring>  using namespace std;  *#define* MAX\_VERTEX\_NUM 30*//图的最大顶点数目*  *//图*  typedef char vertexType;*//顶点数据类型*  typedef enum  {      DG,      DN,      UDG,      UDN  } GraphKind;*// 定义图的类型 { 有向图, 有向网,无向图, 无向网}*  typedef struct arcNode  {*//边*      int adjvex;*//边上数据，指向其父节点下一个邻接点在顶点数组中的位置*      int weight;*//边权重*      struct arcNode \*nextarc;*//指向下一条边*  } arcNode;  typedef struct vnode  {*//顶点*      vertexType vexData;*//顶点数据*      string info;*//顶点信息*      arcNode \*firstarc;*//指向第一个邻接点的边*  } vnode;  typedef struct  {*//图*      vnode \*vertices;*//顶点数组*      int vexnum, arcnum;*//顶点数目，边的数目*      GraphKind kind;*//图类型*  } alGraph;  void printVertices(alGraph \*g)*//打印邻接表*  {  *for* (int i = 0; i < g->vexnum; i++)*//遍历顶点数组*      {          printf("%c ", g->vertices[i].vexData);  *//遍历每个顶点的所有边*          arcNode \*a = (arcNode \*)malloc(sizeof(arcNode));          a = g->vertices[i].firstarc;  *while* (a != NULL)          {              printf("%d&%d ", a->adjvex, a->weight);              a = a->nextarc;          }          printf("\n");      }  }  int locateVex(alGraph \*g, char vex)*//确定点vex在邻接表中的位置*  {  *for* (int i = 0; i < g->vexnum; i++)      {*//遍历比较*  *if* (g->vertices[i].vexData == vex)          {  *return* i;          }      }      printf("vex not exist in locateVex:%c\n", vex);  *return* -1;*//没找到返回-1*  }  void CreateGraph(alGraph \*g)*//先序输入图*  {      printf("输入顶点数，边数和图类\n");      scanf("%d%d%d", &(g->vexnum), &(g->arcnum), &(g->kind));      g->vertices = (vnode \*)malloc(g->vexnum \* sizeof(vnode));      printf("输入顶点\n");  *for* (int i = 0; i < g->vexnum; i++)      {          printf("第%d个顶点：\n", i + 1);          getchar();*//清掉回车*          g->vertices[i].vexData = getchar();          g->vertices[i].firstarc = NULL;*//尾部置空*      }      printf("输入边和权重\n");  *for* (int i = 0; i < g->arcnum; i++)      {          char sv;          char tv;          int w;          getchar();*//清掉回车*          printf("第%d条边和权重：\n", i + 1);          scanf("%c%c", &sv, &tv);          scanf("%d", &w);          int s = locateVex(g, sv);          int t = locateVex(g, tv);          arcNode \*pi = (arcNode \*)malloc(sizeof(arcNode));          pi->adjvex = t;          pi->weight = w;          pi->nextarc = g->vertices[s].firstarc;*//头插法*          g->vertices[s].firstarc = pi;  *if* (g->kind == UDG || g->kind == UDN)*//若是无向图或无向网，在边的另一端也要加入边*          {              arcNode \*pj = (arcNode \*)malloc(sizeof(arcNode));              pj->adjvex = s;              pj->weight = w;              pj->nextarc = g->vertices[t].firstarc;*//头插法*              g->vertices[t].firstarc = pj;          }      }  }  void visit(vertexType v)*//访问顶点数据*  {      printf("%c", v);  }  int find\_char\_in\_vnode(char dest, vnode array[], int array\_size)*//在vnode数组中找dest（char），找到返回位置，没找到返回-1*  {  *for* (int i = 0; i < array\_size; i++)*//遍历寻找*      {  *if* (dest == array[i].vexData)          {  *return* i;          }      }  *// cout<<"fail to find element"<<endl;*  *return* -1;  }  void prim(alGraph \*g, char start)*//prim算法，求最小生成树*  {      alGraph p;*//用邻接表存最小生成树*      p.vertices = (vnode \*)malloc(g->vexnum \* sizeof(vnode));      p.vertices[0].vexData = start;      p.vertices[0].firstarc = NULL;      p.vexnum = 1;      p.arcnum = 0;      p.kind = UDN;  *while* (true)      {  *if* (p.vexnum == g->vexnum)*//Vnew更新完则退出循环*  *break*;          int min\_weight = 100;*//假设所有边权重不超过100*          int min\_index = 0;          int min\_index\_source = 0;  *for* (int i = 0; i < p.vexnum; i++)*//遍历Vnew所有边*          {              int index\_find = find\_char\_in\_vnode(p.vertices[i].vexData, g->vertices, g->vexnum);  *// cout<<index\_find<<endl;*  *if* (index\_find != -1)*//若在g中找到该点,遍历该点所有邻接边，查找权重最小的边*              {                  arcNode \*a = (arcNode \*)malloc(sizeof(arcNode));                  a = g->vertices[index\_find].firstarc;  *if* (a == NULL)  *continue*;  *while* (true)                  {  *if* (a == NULL)  *break*;  *if* (a->weight < min\_weight && find\_char\_in\_vnode(g->vertices[a->adjvex].vexData, p.vertices, p.vexnum) == -1)*//若找到更小权值的边（该边另一端点不在Vnew中）*                      {                          min\_weight = a->weight;                          min\_index = a->adjvex;                          min\_index\_source = i;                      }                      a = a->nextarc;                  }              }          }  *//找到权重最小的之后，更新Vnew和Enew*          p.vertices[p.vexnum].vexData = g->vertices[min\_index].vexData;          p.vertices[p.vexnum].firstarc = NULL;          arcNode \*pi = (arcNode \*)malloc(sizeof(arcNode));          pi->adjvex = p.vexnum;          pi->weight = min\_weight;          pi->nextarc = p.vertices[min\_index\_source].firstarc;*//头插法*          p.vertices[min\_index\_source].firstarc = pi;  *if* (p.kind == UDG || p.kind == UDN)*//若是无向图或无向网，在边的另一端也要加入边*          {              arcNode \*pj = (arcNode \*)malloc(sizeof(arcNode));              pj->adjvex = min\_index\_source;              pj->weight = min\_weight;              pj->nextarc = p.vertices[p.vexnum].firstarc;*//头插法*              p.vertices[p.vexnum].firstarc = pj;          }          p.vexnum++;  *// printVertices(&p);*      }      printVertices(&p);  }  template <typename t>  void print\_array(t array[], int array\_size)*//模板，打印数组*  {  *for* (int i = 0; i < array\_size; i++)      {          cout << array[i] << " ";      }  }  template <typename t>  int find\_element\_in\_array(t dest, t array[], int array\_size)*//模板，在数组中寻找元素，找到返回位置，没找到返回-1*  {  *for* (int i = 0; i < array\_size; i++)      {  *if* (array[i] == dest)          {  *return* i;          }      }  *return* -1;  }  void dijkstra(alGraph \*g, char start)*//dijkstra算法，求点start到其他所有点最短路径和最短距离*  {      int dis[g->vexnum];      int infinity = 100;*//假设所有边权重不超过100*  *for* (int i = 0; i < g->vexnum; i++)      {  *if* (i == find\_char\_in\_vnode(start, g->vertices, g->vexnum))          {              dis[i] = 0;*//start位赋值为0*          }  *else*          {              dis[i] = infinity;*//其他位先赋为无穷大*          }      }      arcNode \*a = (arcNode \*)malloc(sizeof(arcNode));      a = g->vertices[find\_char\_in\_vnode(start, g->vertices, g->vexnum)].firstarc;  *while* (true)*//初始化dis*      {  *if* (a == NULL)  *break*;          dis[a->adjvex] = a->weight;          a = a->nextarc;      }  *// print\_array(dis,g->vexnum);*      char T[g->vexnum];*//已经找到最短路径的点的集合*      int T\_index = 0;      T[T\_index++] = start;*//T初始化*      char temp = start;*//当前点*      string toTemp="";*//当前路径*      toTemp=toTemp+temp;  *while* (true)*//寻找start到所有顶点路径和距离*      {  *if* (T\_index == g->vexnum)  *break*;          int min\_dis = infinity;          int min\_index = -1;  *for* (int i = 0; i < g->vexnum; i++)*//从V-T中找出min(dis)*          {  *if* (find\_element\_in\_array(g->vertices[i].vexData, T, g->vexnum) == -1 && dis[i] < min\_dis)              {                  min\_dis = dis[i];                  min\_index = i;              }          }          T[T\_index++] = g->vertices[min\_index].vexData;          cout << toTemp << "->" << g->vertices[min\_index].vexData << "=" << dis[min\_index] << endl;*//输出路径和距离*          arcNode \*a2 = (arcNode \*)malloc(sizeof(arcNode));          a2 = g->vertices[min\_index].firstarc;  *while* (true)*//查看与当前最小所连接的点，是否可发现 到其他顶点的更短路径*          {  *if* (a2 == NULL)  *break*;  *if* (dis[min\_index] + a2->weight < dis[a2->adjvex])*//若找到更短的路径到达*              {                  dis[a2->adjvex] = dis[min\_index] + a2->weight;*//更新最短距离*  *if* (temp != g->vertices[min\_index].vexData)*//若当前点变化，就更新*                  {                      temp = g->vertices[min\_index].vexData;                      toTemp = toTemp + "->" + temp;*//当前路径更新延伸到当前点*                  }              }              a2 = a2->nextarc;          }  *// print\_array(dis,g->vexnum);*      }  *// print\_array(dis, g->vexnum);*  }  int main()*//测试*  {      alGraph \*g = (alGraph \*)malloc(sizeof(alGraph));  *if* (g == NULL)      {          printf("malloc fail in main for g\n");  *return* -1;      }      CreateGraph(g);      printVertices(g);*//查看邻接表*      printf("输入起点：\n");      getchar();*//清掉回车*      char start = getchar();      cout << "该图最小生成树为：" << endl;      prim(g, start);      cout << "从" << start << "到各点最短路径及最短距离为：" << endl;      dijkstra(g, start);  *return* 0;  } |

* 综合性实验

4.调试分析

1. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
2. 对设计和编码的讨论和分析。该程序实现了顺序栈的操作。分析程序代码的质量，主要从以下几个方面考虑。

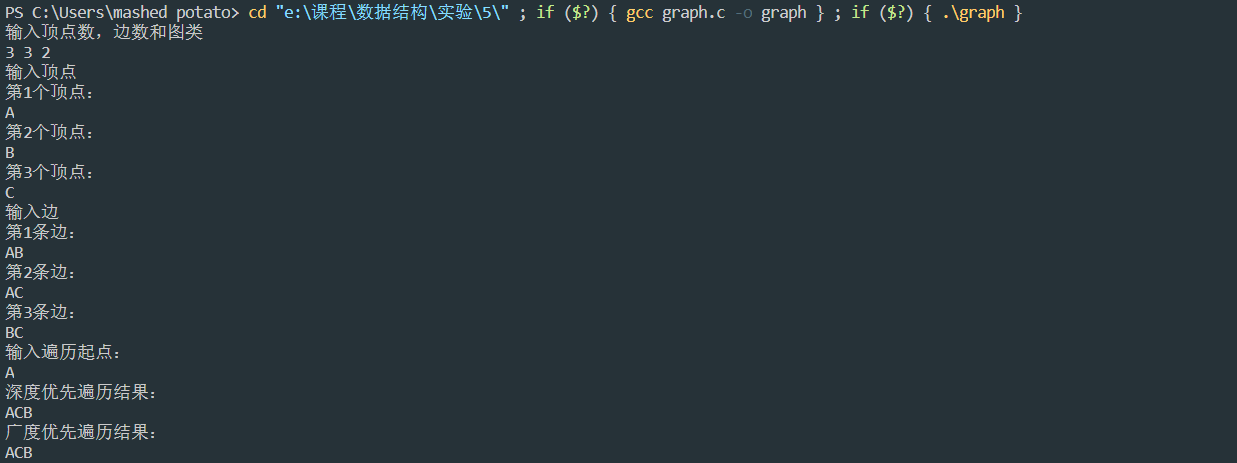
* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

5.使用说明

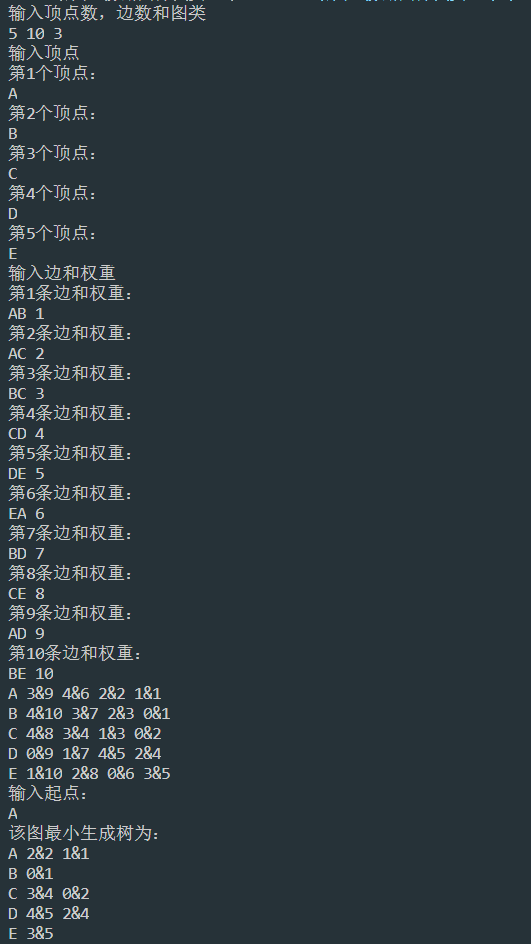
按照屏幕提示，选择想要的功能并输入对应数字，按下ENTER键后，根据屏幕提示进行输入，即可得到想要的结果。

6.测试程序运行结果

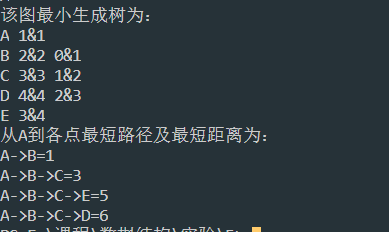
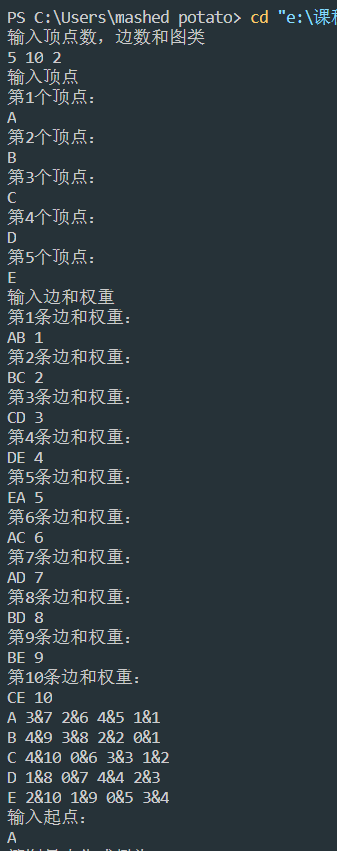
* 验证性实验



* 设计性实验



* 综合性实验



1. **心得体会**

通过本次实验，使我对数据结构有了更深的理解，对指针的运用更加熟练，熟悉了对函数的定义和操作。