**要求代码和实验报告规范，在算法思想中：对实验涉及的数据结构进行有效设计和分析；对算法进行分析并给出时间、空间复杂度的结论；清晰表达实验思路、出现的问题及解决方法。**

**一、调试成功程序及说明**

**1、**

**题目：**编程实现书P96 ADT Graph 基本操作11个，用邻接矩阵存储结构实现

**算法思想：**利用矩阵存储结构实现基本操作，结合书本与之前顺序表的知识，关键在于矩阵数组的使用，最后由switch菜单一一运行。

**运行结果：**运行出现包含11个基本操作的菜单，初始化和创建之后便可以实现各个命令操作的要求。

**结果分析：**正确

**附源程序。**

#include<iostream>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

#define OVERFLOW -2

#define OK 1

#define ERROR 0

#define TRUE 1

#define FALSE 0

#define INFINITY INT\_MAX

#define MAX\_V 20

#define MAXSIZE 256

typedef int Status;

typedef int ElemType;

typedef int GraphKind;

typedef struct

{

int code;

ElemType info;

} VertexType;

typedef struct

{

int arcs[MAX\_V][MAX\_V];

int vexnum,arcnum;

VertexType vexs[MAX\_V];

GraphKind type;

}MGraph;

Status CreateGraph(MGraph &G)

{

printf("请输入图的类型：1.无向图 2.有向图 3.无向网 4.有向网:");

scanf("%d",&G.type);

printf("请输入图的顶点数和边数:");

scanf("%d %d",&G.vexnum,&G.arcnum);

printf("请输入顶点的信息:");

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

{

scanf("%d",&G.vexs[i].code);

getchar();

scanf("%c",&G.vexs[i].info);

}

if(G.type==1 || G.type==2)

{

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

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

G.arcs[i][j] = 0;

}

else if(G.type==3 || G.type==4)

{

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

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

G.arcs[i][j] = INT\_MAX;

}

for(int k=1; k<=G.arcnum; k++)

{

int i,j;

printf("请输入边的两个顶点编号：");

scanf("%d",&i);

getchar();

scanf("%d",&j);

if(G.type == 1)

{

G.arcs[i][j] = 1;

G.arcs[j][i] = 1;

}

else if(G.type == 2)

G.arcs[i][j] = 1;

else if(G.type == 3)

{

printf("请输入该边的权值：");

scanf("%d",&G.arcs[i][j]);

G.arcs[j][i] = G.arcs[i][j];

}

else if(G.type == 4)

{

printf("请输入该边的权值：");

scanf("%d",&G.arcs[i][j]);

}

}

return OK;

}

Status DestroyGraph(MGraph &G)

{

if(G.type==3 || G.type==4)

{

for(int i = 0; i < G.vexnum; i++)

{

for(int j = 0; j < G.vexnum; j++)

{

G.arcs[i][j]= INFINITY;

}

}

}

else

{

for(int i = 0; i < G.vexnum; i++)

{

for(int j = 0; j < G.vexnum; j++)

{

G.arcs[i][j]= 0;

}

}

}

G.vexnum = 0;

G.arcnum = 0;

}

Status GetVex(MGraph &G, int index)

{

if(index <= G.vexnum)

return G.vexs[index].info;

}

Status LocateVex(MGraph G,char v)

{

for(int i=0; i<G.vexnum; i++)

{

if(G.vexs[i].info==v)

return i;

}

return -1;

}

Status FirstAdjVex(MGraph &G, char v)

{

int index = LocateVex(G,v);

if(G.type==1 || G.type==2)

{

for(int i=0; i<G.vexnum; i++)

{

if(G.arcs[index][i]!=0)

return i;

}

}

else if(G.type==3 || G.type==4)

{

for(int i=0; i<G.vexnum; i++)

{

if(G.arcs[index][i]!=INFINITY)

return i;

}

}

return -1;

}

Status NextAdjVex(MGraph &G, char v, char w)

{

int k1 = LocateVex(G,v);

int k2 = LocateVex(G,w);

if(G.type==1 || G.type==2)

{

for(int i=k2; i<G.vexnum; i++)

{

if(G.arcs[k1][i]!=0)

return i;

}

}

else if(G.type==3 || G.type==4)

{

for(int i=k2; i<G.vexnum; i++)

{

if(G.arcs[k1][i]!=INFINITY)

return i;

}

}

return -1;

}

bool Visit[MAXSIZE];

Status DFS(MGraph &G,int index)

{

Visit[index] = true;

printf("%c ",G.vexs[index].info);

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

if(G.arcs[index][i] && !Visit[i])

DFS(G,i);

}

Status DFSTraverse(MGraph &G)

{

int i;

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

Visit[i] = false;

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

{

if(Visit[i]==false)

DFS(G,i);

}

printf("\n");

return OK;

}

Status BFSTraverse(MGraph &G)

{

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

Visit[i] = false;

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

{

if(!Visit[i])

{

Visit[i] = true;

printf("%c ",G.vexs[i].info);

char qu[MAXSIZE];

int temp;

int index;

int front,rear;

front = rear = 0;

qu[rear] = i;

rear++;

while(front!=rear)

{

temp = qu[front];

front = (front+1)%MAXSIZE;

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

{

if(G.arcs[i][j] && !Visit[i] )

{

Visit[j] = true;

printf("%c ",G.vexs[j].info);

qu[rear] = j;

rear = (rear+1)%MAXSIZE;

}

}

}

}

}

printf("\n");

}

Status InsertVex(MGraph &G, char v)

{

G.vexnum += 1;

G.vexs[G.vexnum].info = v;

if(G.type==1 || G.type==2)

{

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

G.arcs[i][G.vexnum] = G.arcs[G.vexnum][i] = 0;

}

else if(G.type==3 || G.type==4)

{

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

G.arcs[i][G.vexnum] = G.arcs[G.vexnum][i] = INFINITY;

}

}

Status InsertArc(MGraph &G, char v,char w)

{

int k1 = LocateVex(G,v);

int k2 = LocateVex(G,w);

if(k1==-1 || k2==-1)

return 0;

if(G.type == 1)

G.arcs[k1][k2] = G.arcs[k2][k1] = 1;

else if(G.type == 2)

G.arcs[k1][k2] = 1;

else if(G.type == 3)

{

int weight;

printf("请输入该边的权值：");

scanf("%d",&weight);

G.arcs[k1][k2] = G.arcs[k2][k1] = weight;

}

G.arcnum++;

return OK;

}

Status DeleteVex(MGraph &G, char v)

{

int i;

int k = LocateVex(G,v);

if(k==-1)

return 0;

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

{

if(G.arcs[k][i])

G.arcnum--;

}

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

{

G.vexs[i].code = G.vexs[i+1].code;

G.vexs[i].info = G.vexs[i+1].info;

}

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

for(int j=k; j<G.vexnum; j++)

G.arcs[i][j]=G.arcs[i][j+1];

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

{

for(int j=k; j<=G.vexnum; j++)

G.arcs[i][j] = G.arcs[i+1][j];

}

G.vexnum--;

return OK;

}

Status DeleteArc(MGraph &G, char v,char w)

{

int k1 = LocateVex(G,v);

int k2 = LocateVex(G,w);

if(k1==-1 || k2==-1)

return 0;

if(G.type == 1)

G.arcs[k1][k2] = G.arcs[k2][k1] = 0;

else if(G.type == 2)

G.arcs[k1][k2] = 0;

else if(G.type == 3)

G.arcs[k1][k2] = G.arcs[k2][k1] = INFINITY;

else if(G.type == 4)

G.arcs[k1][k2] = INFINITY;

G.arcnum--;

}

void menu()

{

cout<<"1.构造图 2.销毁"<<endl;

cout<<"3.顶点i（编号）的元素值 4.顶点的邻接点"<<endl;

cout<<"5.顶点相对于下一顶点的邻接点 6.深度优先遍历"<<endl;

cout<<"7.广度优先遍历 8.添加顶点"<<endl;

cout<<"9.添加边 10.删除顶点"<<endl;

cout<<"11.删除边 "<<endl;

}

int main()

{

MGraph G;

int i,j,k,choice;

char ch1,ch2,v;

while(1)

{

menu();

cout<<"请选择："<<endl;

cin>>choice;

switch(choice)

{

case(1):CreateGraph(G);break;

case(2):DestroyGraph(G);break;

case(3):

cout<<"请输入想查看值的顶点编号：";

cin>>i;

cout<<"图中编号为"<<i<<"的顶点值为：";

printf("%c\n",GetVex(G,i));

break;

case(4):

cout<<"请输入想查看邻接点的顶点：";

cin>>ch1;

i=FirstAdjVex(G,ch1);

if(i==-1)

cout<<"没有邻接点"<<endl;

else

{

k = G.vexs[i].code;

v = G.vexs[i].info;

printf("元素值为B的顶点第一个顶点编号及元素值为：%d %c\n",k,v);

}

break;

case(5):

cout<<"请输入两顶点（查看顶点1相对于顶点2的下一个邻接点）：";

cin>>ch1>>ch2;

i=NextAdjVex(G,ch1,ch2);

if(i==-1)

cout<<"没有邻接点"<<endl;

else

{

k = G.vexs[i].code;

v = G.vexs[i].info;

printf("%c相对于%c的下一个邻接点为：%d %c\n",ch1,ch2,k,v);

}

break;

case(6):DFSTraverse(G); break;

case(7):BFSTraverse(G);

break;

case(8):cout<<"请输入想添加的顶点值:";

cin>>ch1;

InsertVex(G,ch1);

break;

case(9):cout<<"请输入想添加边的两顶点值:";

cin>>ch1>>ch2;

InsertArc(G,ch1,ch2);

break;

case(10):cout<<"请输入想删除的顶点值:";

cin>>ch1;

DeleteVex(G,ch1);

break;

case(11):cout<<"请输入想删除边的两顶点值:";

cin>>ch1>>ch2;

DeleteArc(G,ch1,ch2);

break;

}

cout<<endl;

}

}

**2、**

**题目：**输入N个权值（1-100正整数），建立哈夫曼树。

**算法思想：**利用二叉树知识，比较结点大小，最小的两个相连，直到构成霍夫曼树。

**运行结果：**输入N个权值，输出哈夫曼树。（各结点括号表示）

**结果分析：**正确

**附源程序。**

#include <stdio.h>

#include <string.h>

#include<iostream>

#include<malloc.h>

using namespace std;

typedef struct

{

unsigned int weight;

unsigned int parent, lchild, rchild;

}HTNode,\*HufTree;

void select\_two\_small(HufTree HT, int n, int &s1, int &s2)

{

int min;

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

{

if(HT[i].parent == 0)

{

min = i;

break;

}

}

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

{

if(HT[i].parent == 0)

if(HT[i].weight < HT[min].weight)

min = i;

}

s1 = min;

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

{

if(HT[i].parent == 0 && i != s1)

{

min = i;

break;

}

}

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

{

if(HT[i].parent == 0 && i != s1)

if(HT[i].weight < HT[min].weight)

min = i;

}

s2 = min;

}

void HufCode(HufTree HT, int \*w, int n)

{

int m, s1, s2;

HT = (HufTree)malloc(2\*n\*sizeof(HTNode));

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

{

HT[i].weight = w[i];

HT[i].parent = 0;

HT[i].rchild = 0;

HT[i].lchild = 0;

}

for(int i=n+1; i<2\*n; i++)

{

HT[i].weight = 0;

HT[i].parent = 0;

HT[i].lchild = 0;

HT[i].rchild = 0;

}

printf("霍夫曼树的结构:\n");

for(int i = n+1; i<2\*n; i++)

{

select\_two\_small(HT, i-1, s1, s2);

HT[s1].parent = i;

HT[s2].parent = i;

HT[i].lchild = s1;

HT[i].rchild = s2;

HT[i].weight = HT[s1].weight + HT[s2].weight;

printf("%d (%d, %d)\n", HT[i].weight, HT[s1].weight, HT[s2].weight);

}

printf("\n");

}

int main()

{

HufTree HT;

int n;

printf("请输入霍夫曼树的叶结点个数：");

scanf("%d", &n);

int w[n+1];

printf("请输入权值:",n);

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

scanf("%d", &w[i]);

HufCode(HT,w,n);

return 0;

}

**3、**

**题目：**编写函数，对二叉链表结构的二叉树，求宽度。（书P94 4）

**算法思想：**结合二叉树知识，借助之前求深度的经验，利用辅助数组求出宽度。

**运行结果：**输入二叉树，得出宽度。

**结果分析：**正确

**附源程序。**

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

#include<iostream>

using namespace std;

#define OK 1

#define ERROR 0

#define TRUE 1

#define FALSE 0

#define OVERFLOW -1

#define MAX\_SIZE 256

typedef char ElemType;

typedef int Status;

typedef struct BiTNode

{

ElemType data;

struct BiTNode \*lchild,\*rchild;//左右孩子指针

}BiTNode,\*BiTree;

struct BiTreeQueue

{

BiTree t;

struct BiTreeQueue \*next;

};

Status InitBiTree(BiTree &T)

{

T=(BiTree)malloc(sizeof(BiTNode));

if(T==NULL) exit(OVERFLOW);

T->lchild=NULL;

T->rchild=NULL;

return OK;

}

Status CreateBiTree(BiTree &T)

{

char ch;

cin>>ch;

if(ch=='.')//.表示空

T = NULL;

else

{

T=(BiTNode\*)malloc(sizeof(BiTNode));

if(!T) exit(OVERFLOW);

T->data=ch;

CreateBiTree(T->lchild);

CreateBiTree(T->rchild);

}

return OK;

}

Status Width(BiTree T)

{

if(T==NULL)

return 0;

BiTree Q[100];

int front=1,rear=1,last=1,width=0,WidthMax=0;

Q[rear]=T;

while(front<=last)

{

width++;

BiTree P=Q[front++];

if(P->lchild)

Q[++rear]=P->lchild;

if(P->rchild)

Q[++rear]=P->rchild;

if(front>last)

{

last=rear;

WidthMax=WidthMax>width?WidthMax:width;

width=0;

}

}

return WidthMax;

}

int main()

{

BiTree T;

InitBiTree(T);

cout<<"请输入二叉树（“.”表示空）："<<endl;

CreateBiTree(T);

cout<<"二叉树宽度为："<<Width(T)<<endl;

}

**4、**

**题目：** 编写函数，对一棵以孩子-兄弟链表表示的树，输出第i层的所有元素。

**算法思想：**类似于二叉树，孩子-兄弟链表改变了指针，利用队列结构，存储结构产生了小的改变，建立之后输出第i层元素。

**运行结果：**依据提示建立孩子-兄弟链表树，输入i，得出第i层元素。

**结果分析：**正确

**附源程序。**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct BiTNode

{

char data;

struct BiTNode \*firstchild,\*nextbrother;

}BiTNode,\*BiTree,\*\*pBiTree;

typedef struct queue

{

struct queue \*next;

BiTree data;

}QueueNode,\*pQueueNode;

typedef struct QUEUE

{

pQueueNode head;

pQueueNode tail;

int length;

}Queue,\*pQueue;

int InitQueue(Queue &Q)

{

Q.head = Q.tail = (pQueueNode)malloc(sizeof(QueueNode));

if(!Q.head)

{

return 0;

}

Q.length = 0;

Q.head->next = NULL;

return 1;

}

int Enqueue(pQueue Q,BiTree T)

{

if(Q == NULL)

{

return 0;

}

pQueueNode pNew = (pQueueNode)malloc(sizeof(QueueNode));

if(pNew == NULL)

{

return 0;

}

pNew->data = T;

pNew->next = NULL;

Q->tail->next = pNew;

Q->tail = pNew;

Q->length++;

return 1;

}

int Dequeue(pQueue Q,pBiTree T)

{

if(Q == NULL)

{

return 0;

}

pQueueNode p = Q->head->next;

Q->head->next = p->next;

if(p == Q->tail)

{

Q->tail = Q->head;

}

\*T = p->data;

free(p);

Q->length--;

return 1;

}

bool EmptyQueue(Queue Q)

{

return Q.length == 0;

}

void DestroyQueue(pQueue Q)

{

if(Q != NULL)

{

pQueueNode pTravel = Q->head->next;

while(pTravel != NULL)

{

Q->head->next = pTravel->next;

free(pTravel);

pTravel = Q->head->next;

}

free(Q->head);

Q->length = 0;

}

}

int CreateTree(BiTree &T)

{

T= (BiTree)malloc(sizeof(BiTree));

Queue Q;

InitQueue(Q);

char buffChild[20];

printf("请输入树的根结点（字符,以#代表空）:\n");

scanf("%c", &buffChild[0]);

if (buffChild[0] != '#')

{

T->data = buffChild[0];

T->nextbrother = NULL;

Enqueue(&Q,T);

while (!EmptyQueue(Q))

{

BiTree e;

Dequeue(&Q,&e);

printf("请按长幼顺序输入结点%c的孩子(输入的字符串以#结束):\n", e->data);

scanf("%s", buffChild);

if (buffChild[0] != '#')

{

BiTree q,p;

q =p= (BiTree)malloc(sizeof(BiTNode));

if (!q)

return 0;

q->data = buffChild[0];

e->firstchild = q;

Enqueue(&Q,q);

p = q;

for (int i = 1; i < strlen(buffChild) ; ++i)

{

q = (BiTree)malloc(sizeof(BiTNode));

if (!q)

return 0;

q->data = buffChild[i];

p->nextbrother = q;

Enqueue(&Q,q);

p = q;

}

p->nextbrother = NULL;

}

else

{

e->firstchild = NULL;

}

}

}

else

{

T = NULL;

}

return 1;

}

int LevelOrderTraverse( BiTree T)

{

//层序遍历树

Queue Q;

InitQueue(Q);

if (T)

{

printf("%c ", T->data);

Enqueue(&Q,T);

while (!EmptyQueue(Q))

{

BiTree e, p;

e = p = (BiTree)malloc(sizeof(BiTree));

Dequeue(&Q,&e);

p = e->firstchild;

while (p)

{

printf("%c ", p->data);

Enqueue(&Q,p);

p = p->nextbrother;

}

}

return 1;

}

return 0;

}

void LevelTravel(BiTree T)

{

Queue Q;

InitQueue(Q);

if(T)

{

printf("%c ",T->data);

Enqueue(&Q,T);

BiTree temp;

while(!EmptyQueue(Q))

{

Dequeue(&Q,&temp);

if(temp->firstchild != NULL)

{

temp = temp->firstchild;

printf("%c ",temp->data);

Enqueue(&Q,temp);

while(temp->nextbrother != NULL)

{

temp = temp->nextbrother;

printf("%c ",temp->data);

Enqueue(&Q,temp);

}

}

}

}

}

void Level(BiTree T,int n)

{

BiTree temp;

if(n==1)

printf("%c ",T->data);

else if(n==2)

{

T = T->firstchild;

printf("%c ",T->data);

while(T->nextbrother)

{

T = T->nextbrother;

printf("%c ",T->data);

}

}

else

{

int i=1;

T = T->firstchild;

while(i<n-2)

{

if(T->firstchild)

{

T = T->firstchild;

i++;

}

else

T = T->nextbrother;

}

Queue Q;

InitQueue(Q);

Enqueue(&Q,T);

while(T->nextbrother)

{

T = T->nextbrother;

Enqueue(&Q,T);

}

while(!EmptyQueue(Q))

{

Dequeue(&Q,&temp);

if(temp->firstchild)

{

temp = temp->firstchild;

printf("%c ",temp->data);

while(temp->nextbrother)

{

temp = temp->nextbrother;

printf("%c ",temp->data);

}

}

}

}

}

int main()

{

BiTree T;

printf("请输入树的元素：");

CreateTree(T);

printf("层次遍历输出：");

LevelOrderTraverse(T);

int i;

printf("\n选择输出第几层元素:");

scanf("%d",&i);

printf("输出第%d层元素：",i);

Level(T,i);

return 0;

}

**二、未调试成功程序及说明**

**1、**

**题目：**CSP题目 俄罗斯方块

**算法思想：**简单数组问题，一个数组依据另一个数组进行改变。

**错误原因：**对方块是否触底的判定比较模糊。

**附源程序。**

#include<iostream>

using namespace std;

int flow(int a[15][10],int b[4][4],int hang,int col)

{

int x=0,y=0;

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

{

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

{

a[i][col+j-1]=b[x][y];

y++;

}

y=0;

x++;

}

if(hang>0)

{

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

{

a[hang-1][col+j-1]=0;

}

}

}

int main()

{

int a[15][10],b[4][4];

int col,hang=0;

for(int i=0;i<15;i++)

{

for(int j=0;j<10;j++)

{

cin>>a[i][j];

}

}

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

{

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

{

cin>>b[i][j];

}

}

cin>>col;

flow(a,b,hang,col);

while(1)

{

if(a[hang+4][col-1]==0&&a[hang+4][col]==0&&a[hang+4][col+1]==0&&a[hang+4][col+2]==0)

{

hang++;

flow(a,b,hang,col);

}

else if((a[hang+3][col-1]==1&&a[hang+4][col-1]==1)||(a[hang+3][col]==1&&a[hang+4][col]==1)||(a[hang+3][col+1]==1&&a[hang+4][col+1]==1)||(a[hang+3][col+2]==1&&a[hang+4][col+2]==1))

{

break;

}

else

{

hang++;

flow(a,b,hang,col);

}

}

for(int i=0;i<15;i++)

{

for(int j=0;j<10;j++)

{

cout<<a[i][j]<<" ";

}

cout<<endl;

}

return 0;

}

**正确代码：**

#include<iostream>

using namespace std;

int main()

{

int a[15][10],b[4][4];

int col,hang=0;

for(int i=0;i<15;i++)

{

for(int j=0;j<10;j++)

{

cin>>a[i][j];

}

}

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

{

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

{

cin>>b[i][j];

}

}

cin>>col;

int i,j,k;

for(i=1;i<=15;i++)

{//i:方块向下移动i次。

for(j=3;j>=0;j--)

{//i+j:方块当前位置的下一行

for(k=0;k<4;k++)

{//k:方块的列

if(b[j][k]==1 && (a[i+j][col+k-1]==1 || i+j==15))

{//如果下一行有1，或者到达底部

goto end;

}

}

}

}

end:

for(int j=3;j>=0;j--)

{

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

{

if(b[j][k]==1) a[j+i-1][col+k-1]=1;

}

}

for(int i=0;i<15;i++)

{

for(int j=0;j<10;j++)

{

cout<<a[i][j]<<" ";

}

cout<<endl;

}

return 0;

}

**三、代码行数及小结**

行数：合计900行左右

小结：此次上机重点在于图和二叉树的巩固，可借助上一次上机的成果，并与之前的顺序表、单链表有很多联系之处，并运用到了队列结构，有一定的书本给出的代码，要求我们实现。