**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：**图的关键路径算法

**算法思想：**利用邻接表建立图，利用STL函数，借用队列实现顺序拓扑排序，借用栈实现逆序拓扑排序。最早开始时计算过程间与顺序拓扑排序一致，最晚开始时间计算过程是逆序拓扑排序，最后输出关键路径。

**运行结果：**输入图的信息，输出关键路径

**结果分析：**正确

**附源程序。**

#include<iostream>

using namespace std;

#include<stdlib.h>

#include<queue>

#include<stack>

typedef struct ArcNode

{

int adjvex;

int weight;

struct ArcNode \*nextarc;

}ArcNode;

typedef struct

{

int data;

ArcNode \*firstarc;

}VNode,\*AdjList;

typedef struct

{

int vexnum,arcnum;

AdjList arcs;//邻接表，动态申请空间

}ALGraph;

void CreateALGraph(ALGraph &G)

{

cout<<"请输入顶点数和边数：";

cin >> G.vexnum >> G.arcnum;

G.arcs = (AdjList)malloc((G.vexnum + 1) \* sizeof(VNode));//从1开始存

for(int i = 0; i < G.vexnum + 1; i++)

{

(G.arcs + i)->firstarc = NULL;

}

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

{

int v0, v1, w;

ArcNode\* p, \*pcur;

cout<<"请输入与边关联的两个顶点及其权值：";

cin >> v0 >> v1 >> w;

p = (ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex = v1;

p->weight = w;

p->nextarc = NULL;

if((G.arcs + v0)->firstarc != NULL)

{

pcur = (G.arcs + v0)->firstarc;

while(pcur->nextarc)

{

pcur = pcur->nextarc;

}

pcur->nextarc = p;

}

else

{

(G.arcs + v0)->firstarc = p;

}

}

}

void TopologicalSort(ALGraph G, int \*ve, int \*vl, stack<int> &S)

{

int indegree[G.vexnum + 1];//入度表

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

{

indegree[i] = 0;

\*(ve + i) = \*(vl + i) = 0;

}

for(int i = 0; i <= G.vexnum; i++)//计算相应点入度

{

ArcNode \*pcur = (G.arcs + i)->firstarc;

while(pcur)

{

indegree[pcur->adjvex] ++;

pcur = pcur->nextarc;

}

}

queue<int> Q;

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

{

if(indegree[i] == 0)

{

Q.push(i);

indegree[i] = -1;//标记已入队

}

}

while(!Q.empty())

{

int index = Q.front();

Q.pop();

S.push(index);

ArcNode \*pcur = (G.arcs + index)->firstarc;

while(pcur)

{

indegree[pcur->adjvex]--;

if(\*(ve + pcur->adjvex) < \*(ve + index) + pcur->weight)

{

\*(ve + pcur->adjvex) = \*(ve + index) + pcur->weight;

}

pcur = pcur->nextarc;

}

for(int i = 1; i <= G.vexnum; i++)//将入度为0点入队

{

if(indegree[i] == 0)

{

Q.push(i);

indegree[i] = -1;

}

}

}

cout<<endl;

}

//关键路径

void CriticalPath(ALGraph G)

{

int \*ve, \*vl;

int count=0;

stack<int> S;

queue<int> Print;

ve = (int\*)malloc((G.vexnum + 1)\*sizeof(int));

vl = (int\*)malloc((G.vexnum + 1)\*sizeof(int));

TopologicalSort(G,ve,vl,S);//顺序拓扑

//逆序拓扑，借用栈S

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

{

\*(vl + i) = \*(ve + S.top());//初始化最迟开始时间数组

}

while(!S.empty())

{

int index = S.top();

S.pop();

ArcNode \*pcur = (G.arcs + index)->firstarc;

while(pcur)//判断出度是否为0

{

if(\*(vl + pcur->adjvex) < \*(vl + index) + pcur->weight)

{

\*(vl + index) = \*(vl + pcur->adjvex) - pcur->weight;

}

pcur = pcur->nextarc;

}

}

cout<<endl;

Print.push(1);

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

{

ArcNode \*p = (G.arcs + i)->firstarc;

while(p)

{

if(\*(ve + i) == \*(vl + p->adjvex) - p->weight)

{

cout << "起点： "<<i<<" 终点： "<<p->adjvex<<" 权值： "<<p->weight<<endl;

if(Print.back() ==i)

{

count+=p->weight;

Print.push(p->adjvex );

}

}

p = p->nextarc;

}

}

cout<<endl<<"一条关键路径为：";

while(!Print.empty() )

{

cout<<Print.front();

Print.pop() ;

if(!Print.empty() )

cout<<"-->";

}

cout<<endl<<"权值为：";

cout<<count;

}

int main()

{

ALGraph G;

CreateALGraph(G);

CriticalPath(G);

return 0;

}

**2、**

**题目：**编程实现Dijkstra算法

**算法思想：**利用邻接矩阵建立图，借助书上所给的Dijkstra算法，借助一个判断函数visited，实现Dijkstra算法求出最短路径

**运行结果：**输入图的信息，输出最短路径

**结果分析：**正确

**附源程序。**

#include <stdio.h>

#include <stdlib.h>

#define LIMITLESS 9999

#define MAX\_V 20

#define INFINITY INT\_MAX

typedef struct

{

int code;

char info;

} VertexType;

typedef struct

{

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

int vexnum;

int arcnum;

VertexType vexs[MAX\_V];

int type;

}MGraph;

int CreateGraph(MGraph &G)

{

printf("请输入有向网的顶点数和边数:");

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

printf("请输入顶点的信息(由0开始):");

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

{

getchar();

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

}

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

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

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

for(int k=0; k<G.arcnum; k++)

{

int i,j;

printf("请输入边的两个顶点编号及权值：");

scanf("%d",&i);

scanf("%d",&j);

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

}

return 1;

}

void Ppath(MGraph G,int path[], int i, int v)

{

int k;

k = path[i];

if (k == v)

return;

Ppath(G,path, k, v);

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

printf("->");

}

void print\_path(MGraph G,int dist[],int path[], int visited[],int n,int v)

{

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

{

if(visited[i] == 1)

{

if(dist[i]!=LIMITLESS)

{

printf("从%c到%c的最短路径长度为:%d\t路径为:",G.vexs[v].info,G.vexs[i].info,dist[i]);

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

printf("->");

Ppath(G,path,i,v);

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

}

}

}

}

void Dijkstra(MGraph G, int v)

{

int mindis, i, j, u;

int visited[MAX\_V];

int dist[MAX\_V];

int path[MAX\_V];

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

{

visited[i] = 0;

dist[i] = G.arcs[v][i];

if (G.arcs[v][i] < LIMITLESS)

path[i] = v;

else

path[i] = -1;

}

visited[v] = 1;

path[v] = 0;

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

{

mindis = INT\_MAX;

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

{

if (visited[j] == 0 && dist[j] <mindis)

{

mindis = dist[j];

u = j;

}

}

visited[u] = 1;

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

{

if (visited[j] == 0)

{

if (G.arcs[u][j] < LIMITLESS&&dist[u] + G.arcs[u][j] < dist[j])

{

dist[j] = dist[u] + G.arcs[u][j];

path[j] = u;

}

}

}

}

print\_path(G,dist,path,visited,G.vexnum,v);

}

int main()

{

MGraph G;

CreateGraph(G);

int s;

printf("请输入起始点编号(非末尾顶点):");

scanf("%d",&s);

Dijkstra(G,s);

return 0;

}

/\*5 8

0 1 2 3 4

0 1 2

0 2 6

0 3 9

1 2 3

2 3 1

1 4 7

2 4 2

3 4 2\*/

**3、**

**题目：**CSP题目 苹果掉落

**算法思想：**对输入数据的简单计算分析

**运行结果：**输入树和检查的信息，按题要求输出

**结果分析：**正确

**附源程序。**

#include <iostream>

#include <math.h>

using namespace std;

#define inf 9999

int main()

{

int n,m,s,i,j,a;

bool b[99]= {0}; //记录苹果掉落

int t=0,d=0,e=0;

cin>>n;

for(i=0; i<n; i++)

{

cin>>m;

s=0;

cin>>s;

for(j=1; j<m; j++)

{

cin>>a;

if(a>0)

{

if(s>a)

{

b[i]=1;

}

s=a;

}

else if(a<=0)

{

s+=a;

}

}

t+=s;

}

for(i=0; i<n; i++)

{

if(b[i]==1)

{

d++;

}

if(i==n-2&&b[i]==1&&b[i+1]==1&&b[0]==1)

{

e++;

}

else if(i==n-1&&b[i]==1&&b[0]==1&&b[1]==1)

{

e++;

}

else if(b[i]==1&&b[i+1]==1&&b[i+2]==1)

{

e++;

}

}

cout<<t<<" "<<d<<" "<<e<<endl;

}

**4、**

**题目：**CSP题目 麦田灌溉

**算法思想：**本质上是最小生成树的算法

**运行结果：**输入麦田和水渠的信息，输出最小费用和方案

**结果分析：**正确

**附源程序。**

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

using namespace std;

int cost[256][256];

typedef struct

{

int a;//可连麦田

int b;

int c;//费用

}Canal;

int main()

{

int n,m;

printf("请输入麦田的片数和小刘可以建立的水渠的数量：");

scanf("%d%d",&n,&m);

Canal canals[m];

int nodes[n];

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

scanf("%d%d%d",&canals[i].a,&canals[i].b,&canals[i].c);

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

{

Canal min = canals[i];

int temp = i;

for(int j=i+1; j<m; j++)

{

if(canals[j].c<min.c)

{

min = canals[j];

temp = j;

}

}

canals[temp] = canals[i];

canals[i] = min;

}

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

{

nodes[i] = 0;

}

nodes[0] = 1;

int cost = 0;

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

{

int min = -1;

int hi = -1;

int fi;

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

{

if((nodes[canals[i].a-1]==1)&&(nodes[canals[i].b-1]==0))

{

min = canals[i].c;

hi = canals[i].b;

fi=canals[i].a;

break;

}

else if((nodes[canals[i].a-1]==0)&&(nodes[canals[i].b-1]==1))

{

min = canals[i].c;

hi = canals[i].a;

fi=canals[i].b;

break;

}

}

printf("建立水渠：麦田%d与麦田%d\n",fi,hi);

nodes[hi-1] = 1;

cost += min;

}

printf("最小费用为：%d",cost);

return 0;

}

**二、代码行数及小结**

行数：合计450行左右

小结：本次上机复习图的基本操作，完成图的关键路径以及Dijkstra最短路径算法，练习在程序中的应用，需要我们对图的操作熟练掌握。