哈尔滨工业大学计算机科学与技术学院  
实验报告

课程名称：数据结构与算法  
课程类型：必修  
实验项目名称：数据结构实验三   
实验题目：图

实验时间： 2021.11.08

班级： 2003009班

姓名： 孙莹

1. 实验目的  
   1.熟悉图的建立，递归算法的深度优先搜索，非递归算法的深度优先搜索。  
   2.学习图的关键AOV网的拓扑排序，学习求最短路径的原理，练习其求法。

二、实验要求及实验环境

1. 搜索算法：设计算法完成至少8个顶点10条边的图的深度优先搜索算法（递归和非递归）
2. 一项工程由多道工序组成， 按照施工过程的要求，这些工序之间，客观上有一个必须遵守的先后关系。 对那些紧接在已知工序前的工序叫紧前工序，把在已知工序后边紧接的工序叫后项工序， 只有已知工序的所有紧前工序都完成，已知工序才能开始施工。一天中可以同时进行若干道工序。编程实现：工程最少在几天内完成，并找出一种工程施工安排方案。
3. 设计思想（用到的主要函数、数据类型的定义，主要功能的流程图（1-2个）及各程序模块之间的调用关系，自己扩展内容的等）

1.

主函数：

void CreateGraph();//建图

void DFS1(int i); //递归深度优先遍历

void DFS2(int i); //非递归深度优先遍历

数据类型定义：

typedef struct node {// 边表 结点

int ad; // 邻接点域（下标）

struct node\* next; // 下一边链接指针

} EdgeNode;

typedef struct { // 顶点表 结点

int id; // 顶点数据域

EdgeNode\* first;// 边链表头指针

} VertexNode;

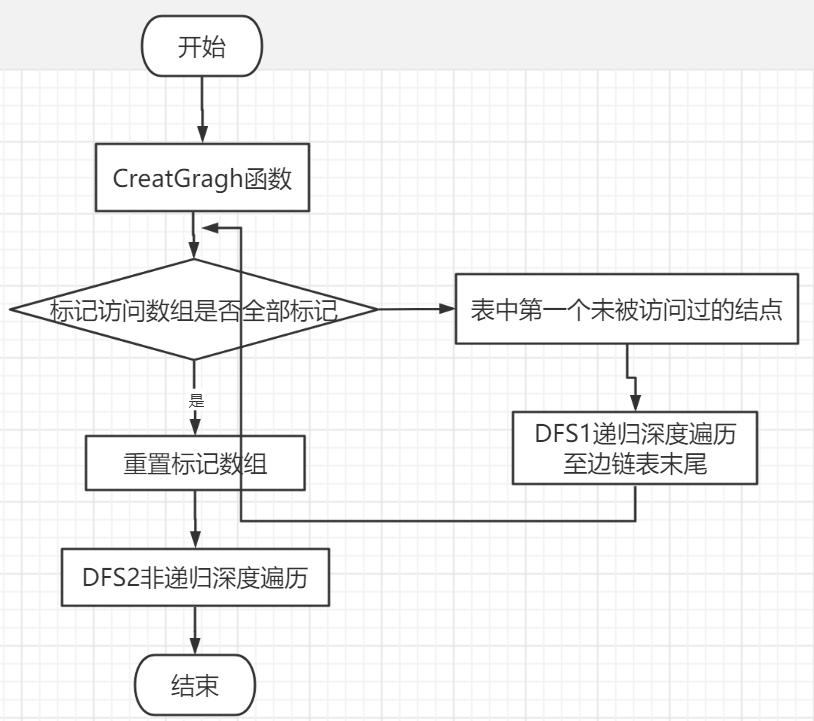
typedef struct { // 图的 邻接表

VertexNode vlist[NumVertices];

int n, e; // 顶点个数与边数

} AdjGraph;

各模块调用：

  
设计思想：图的深度优先搜索非递归实现算法步骤：

①第一个结点入栈；

②栈中退出一个顶点，先访问，再将其所有未被访问过的邻接结点入栈；

③重复②至栈空；

2.

主函数：

void CreatGraph(FILE\* fp, VertexNode\* G, int vertexnum, int edgenum) //生成图的邻接表

void LeastDay(VertexNode\* G, int vertexnum, int edgenum) //求关键路径

数据类型定义：

typedef struct node //边表结点

{

int adjvex; //下标

int weight; //权值

char letter; //项目序号

struct node\* next;

}EdgeNode;

typedef struct //顶点表结点

{

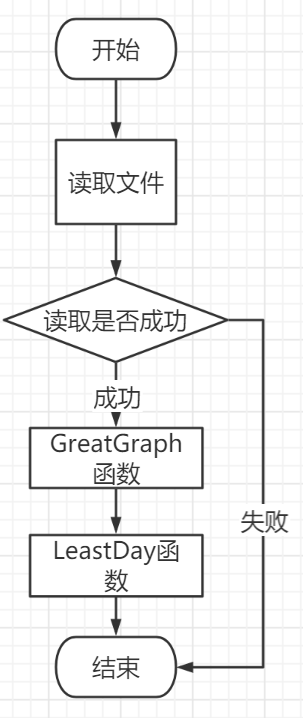
int vertexdata; //顶点数据域

int in; //顶点入度

EdgeNode\* firstedge;//边链表头指针

}VertexNode;

各模块调用：

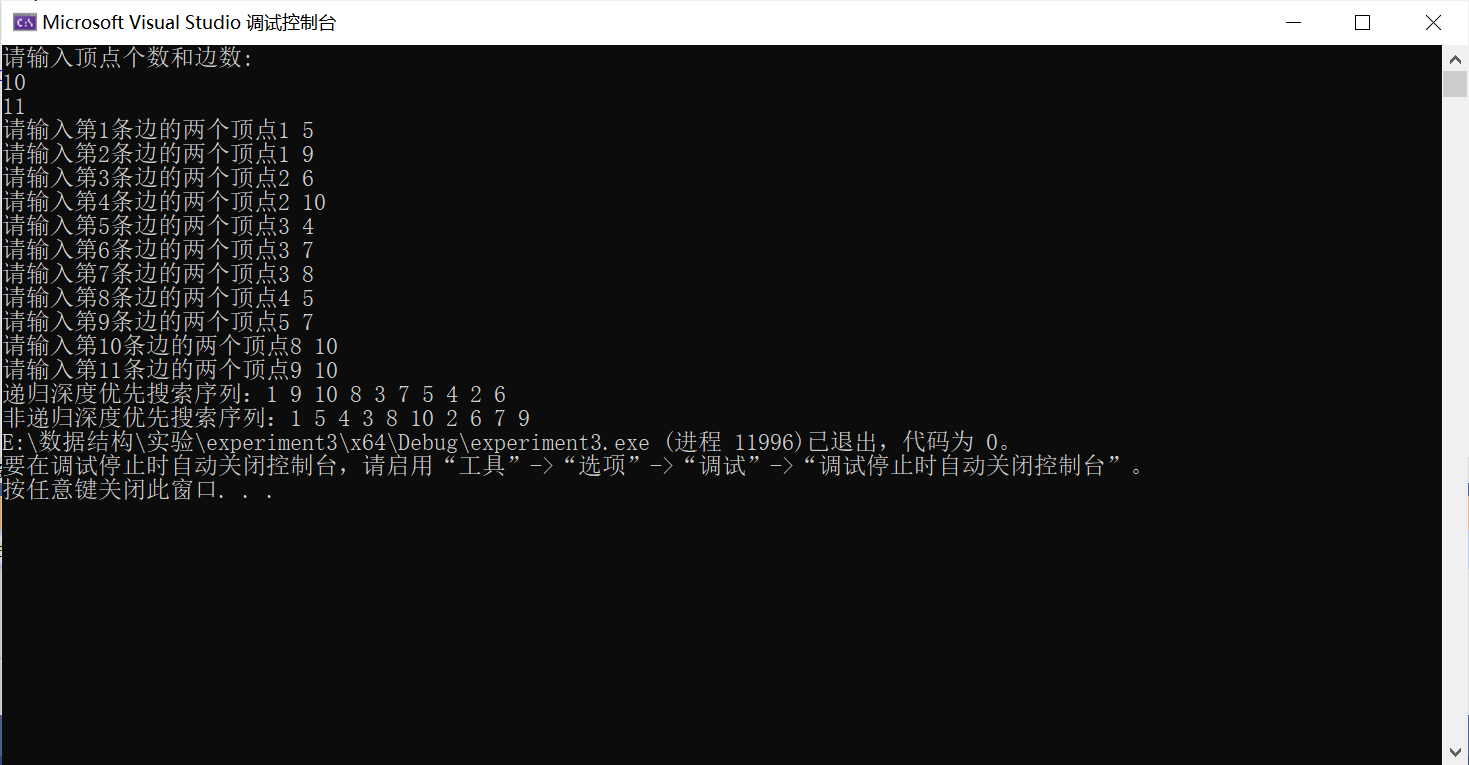
  
设计思想：求关键路径的算法步骤如下：  
①从源点出发，另ve[源点]为0，按拓扑排序求其余顶点的最早发生时间ve[]；

②从汇点出发，令vl[汇点]=ve[汇点]，按逆拓扑有序求其余顶点的最迟发生时间vl[]；

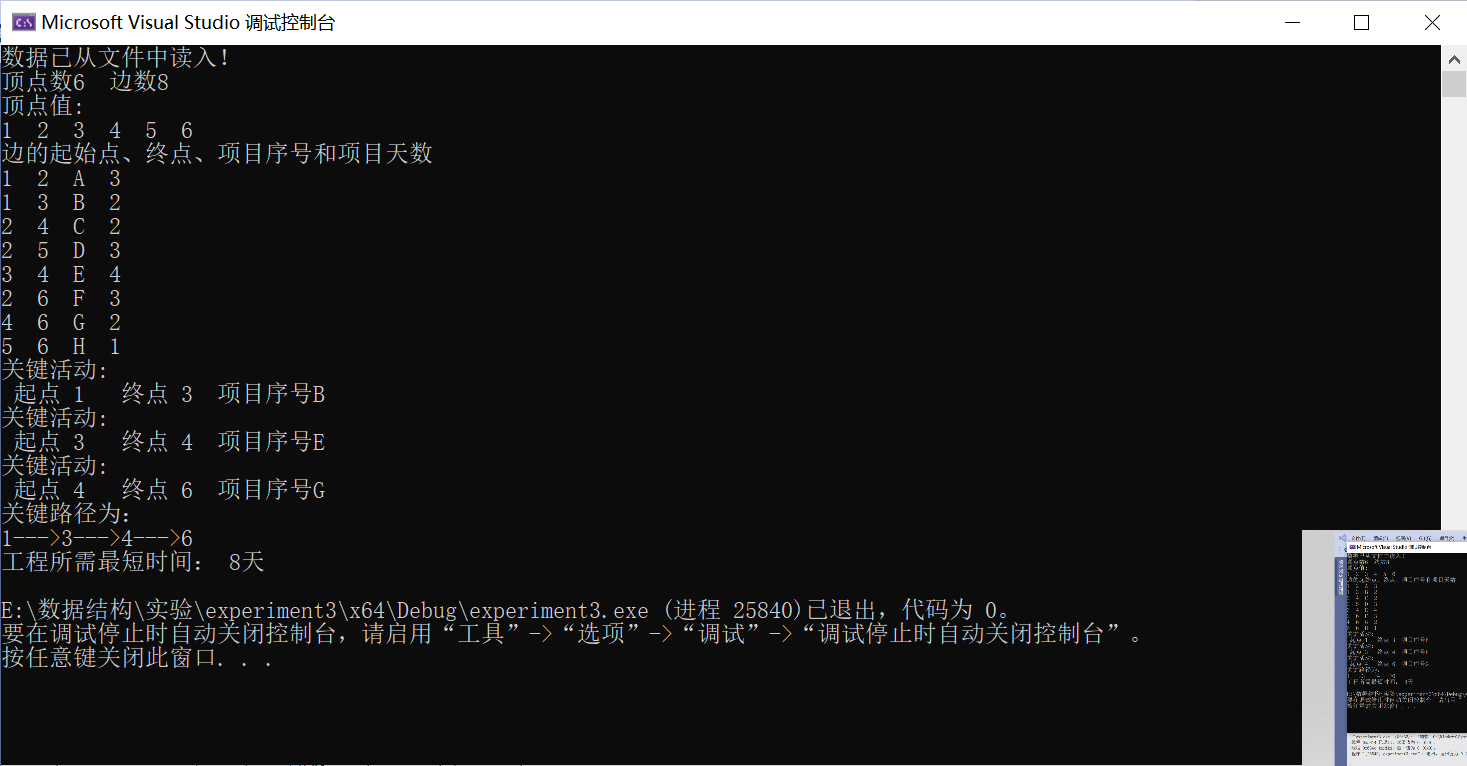
③根据各顶点的ve[]值求所有弧最早开始时间e();

④根据个顶点的vl[]值求所有弧最晚开始时间l()；

⑤求AOE网中所有活动的差额d()，找出所有d()=0的活动构成关键路径。

1. 测试结果   
   1.  
   

2.





1. 系统不足与经验体会   
    1.本次图的深度优先搜索依然用到了栈作为辅助，结合上次前、中、后序遍历也利用了栈和队列，我意识到栈、队列这些数据结构不仅是一种数据存储方式，更是一种可以应用到算法设计里的思想，比如栈的“后进先出”，就满足深度搜索时我们需要的“遍历到最深再回退”。在以后编写程序的过程中，要进一步注意根据设计要求思考利用上符合要求、操作简便的数据结构。

2.本次工程排序问题，我用来读入的文件中存储的是图信息，而不是直接一张工程表，因为我没有想到一种简洁的的方法可以让计算机自动由工程表得到图信息。

1. 带注释的源代码

1.

#include <iostream>

using namespace std;

#define NumVertices 50

typedef struct node {// 边表 结点

int ad; // 邻接点域（下标）

struct node\* next; // 下一边链接指针

} EdgeNode;

typedef struct { // 顶点表 结点

int id; // 顶点数据域

EdgeNode\* first;// 边链表头指针

} VertexNode;

typedef struct { // 图的 邻接表

VertexNode vlist[NumVertices];

int n, e; // 顶点个数与边数

} AdjGraph;

AdjGraph G; //图

int visited[NumVertices]; //访问数组

int dfn[NumVertices]; // 顶点的深度优先搜索编号

void CreateGraph();//建图

void DFS1(int i); //递归深度优先遍历

void DFS2(int i); //非递归深度优先遍历

int main()

{

CreateGraph(); //建图

for (int i = 1; i <= G.n; i++) visited[i] = -1; //访问数组初始化

cout << "递归深度优先搜索序列：";

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

{

if (visited[i] == -1)

DFS1(i); // 从顶点 i 出发的一次搜索，DFSX(G, i )

}

cout << endl << "非递归深度优先搜索序列：";

for (int i = 1; i <= G.n; i++) visited[i] = -1; //访问数组再次初始化

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

{

if (visited[i] == -1)

DFS2(i); // 从顶点 i 出发的一次搜索，DFSX(G, i )

}

}

void CreateGraph()

{

cout << "请输入顶点个数和边数:" << endl;

cin >> G.n >> G.e; //1. 输入顶点个数和边数

for (int i = 1; i <= G.n; i++) { //2. 建立顶点表

G.vlist[i].id = i; //2.1 顶点信息

G.vlist[i].first = NULL;

} //2.2 边表置为空表

for (int i = 1; i <= G.e; i++) { //3. 逐条边输入, 建立边表

int tail, head;

cout << "请输入第" << i << "条边的两个顶点";

cin >> tail >> head; //3.1 输入

EdgeNode\* p = new EdgeNode; //3.2 建立边结点

p->ad = head; //3.3 设置边结点

p->next = G.vlist[tail].first; //3.4 链入第 tail 号链表的前端

G.vlist[tail].first = p;

p = new EdgeNode;

p->ad = tail;

p->next = G.vlist[head].first; // 链入第 head 号链表的前端

G.vlist[head].first = p;

}

}

void DFS1(int i)

// 以第i号结点为出发点时对邻接表表示的图G 进行深度优先搜索

{

EdgeNode\* p;

cout << G.vlist[i].id << " ";

visited[i] = 1;

p = G.vlist[i].first;

while (p != NULL) {

if (visited[p->ad] == -1)

DFS1(p->ad);

p = p->next;

}

} //DFS1 递归算法

void DFS2(int i) //非递归深度优先遍历

{

int s[NumVertices];

int top = -1;

s[++top] = i;

visited[i] = 1;

while (top >= 0)

{

int k = s[top--];

cout << G.vlist[k].id << " ";

for (EdgeNode\* j = G.vlist[k].first; j != NULL; j = j->next)

{

if (visited[j->ad] == -1)

{

s[++top] = j->ad;

visited[j->ad] = 1;

}

}

}

}

2.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LEN 30

typedef struct node //边表结点

{

int adjvex; //下标

int weight; //权值

char letter; //项目序号

struct node\* next;

}EdgeNode;

typedef struct //顶点表结点

{

int vertexdata; //顶点数据域

int in; //顶点入度

EdgeNode\* firstedge;//边链表头指针

}VertexNode;

void CreatGraph(FILE\* fp, VertexNode\* G, int vertexnum, int edgenum) //生成图的邻接表

{

int i;

int begin, end, edgevalue;

char pro;

EdgeNode\* p;

for (i = 0; i < vertexnum; i++) //初始化

{

G[i].in = 0;

G[i].firstedge = NULL;

}

printf("顶点值:\n");

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

{

fscanf\_s(fp, " %d", &G[i].vertexdata);

printf("%d ", G[i].vertexdata);

}

printf("\n边的起始点、终点、项目序号和项目天数\n");

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

{

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

fscanf\_s(fp, " %d %d %d %c", &begin, &end, &edgevalue, &pro);

printf("%d %d %c %d\n", begin, end, pro, edgevalue);

p->adjvex = end - 1;

p->weight = edgevalue;

p->letter = pro;

p->next = G[begin - 1].firstedge;

G[begin - 1].firstedge = p; //头插

G[end - 1].in++;

}

}

void LeastDay(VertexNode\* G, int vertexnum, int edgenum) //求关键路径

{

int i = 0, j = 0, k = 0, n = 0, days = 0, quantity = 0;

int key[50]; //关键路径的顶点

int front, rear, \* Queue;

front = rear = -1;

int ve[MAX\_LEN] = { 0 }; //顶点最早发生时间

int vl[MAX\_LEN] = { 0 }; //顶点最晚发生时间

int ee[MAX\_LEN] = { 0 }; //活动最早发生时间

int el[MAX\_LEN] = { 0 }; //活动最晚发生时间

EdgeNode\* p;

Queue = (int\*)malloc(vertexnum \* sizeof(int));

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

{

if (G[i].in == 0) //入度为0的顶点入队

Queue[++rear] = i;

quantity++;

}

while (front != rear) //先广遍历求ve

{

j = Queue[++front];

quantity++;

p = G[j].firstedge;

while (p)

{

k = p->adjvex;

G[k].in--;

if ((ve[j] + p->weight) > ve[k])

ve[k] = ve[j] + p->weight;

if (G[k].in == 0)

Queue[++rear] = k;

p = p->next;

}

}

if (quantity < vertexnum)

{

printf("此图有回路，无法计算关键路径！\n");

return;

}

days = ve[vertexnum - 1]; //总天数

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

vl[i] = days;

for (i = vertexnum - 2; i >= 0; i--) //回退阶段求vl

{

j = Queue[i];

p = G[j].firstedge;

while (p)

{

k = p->adjvex;

if ((vl[k] - p->weight) < vl[j])

vl[j] = vl[k] - p->weight;

p = p->next;

}

}

i = -1;

for (j = 0; j < vertexnum; j++)

{

p = G[j].firstedge;

while (p)

{

k = p->adjvex;

ee[++i] = ve[j]; //求ee ，E(i)=VE(j)

el[i] = vl[k] - p->weight; //求el ， L(i)=VL(k)-ACT(ai)

if (el[i] == ee[i])

{

printf("关键活动:\n ");

printf("起点 %d 终点 %d ", G[j].vertexdata, G[k].vertexdata);

if (G[j].firstedge->adjvex == G[k].vertexdata - 1)

printf("项目序号%c\n", G[j].firstedge->letter);

else

printf("项目序号%c\n", G[j].firstedge->next->letter);

key[n] = G[j].vertexdata;

n++;

}

p = p->next;

}

}

key[n] = G[vertexnum - 1].vertexdata;

printf("关键路径为：\n");

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

{

printf("%d", key[i]);

if (key[i] != G[vertexnum - 1].vertexdata)

printf("--->");

}

printf("\n");

printf("工程所需最短时间： %d天\n", days);

}

int main()

{

FILE\* fp;

int vertexnum, edgenum;

VertexNode\* G;

fopen\_s(&fp, "project.txt", "r");

if (fp == NULL)

{

printf("读取文件project.txt失败！\n");

exit(0);

}

printf("数据已从文件中读入！\n");

fscanf\_s(fp, " %d", &vertexnum);

fscanf\_s(fp, " %d", &edgenum);

printf("顶点数%d 边数%d\n", vertexnum, edgenum);

G = (VertexNode\*)malloc(vertexnum \* sizeof(VertexNode));

CreatGraph(fp, G, vertexnum, edgenum);

LeastDay(G, vertexnum, edgenum);

return 0;

}