《数据结构》课程设计报告

题　　目：　　关键路径应用

班　　级：　　计算机[redacted]

姓　　名：　　[redacted]

学　　号：　　[redacted]

指导老师：　　[redacted]

完成日期：　　2018 年 12 月 25 日

1. 设计题目与要求

关键路径的应用：利用关键路径算法，规划项目活动。已知每个活动的持续时间和活动间的依赖关系。利用关键路径算法确定每个活动的最早发生时间和最迟发生时间，影响工程进度的关键活动，计算完成整个工程至少需要多少时间。

2. 设计软硬件环境

计算机：Surface pro 6

处理器：I5-8250U 1.8GHZ

内存容量：8.00GB

操作系统：Windows 10 企业版

IDE：Dev c++

3. 功能设计与描述

本代码设计实现了通过构建AOE网并使用关键路径算法确定每个活动的最早最迟发生时间，最后得出整个活动需要的最少时间。

1.功能设计

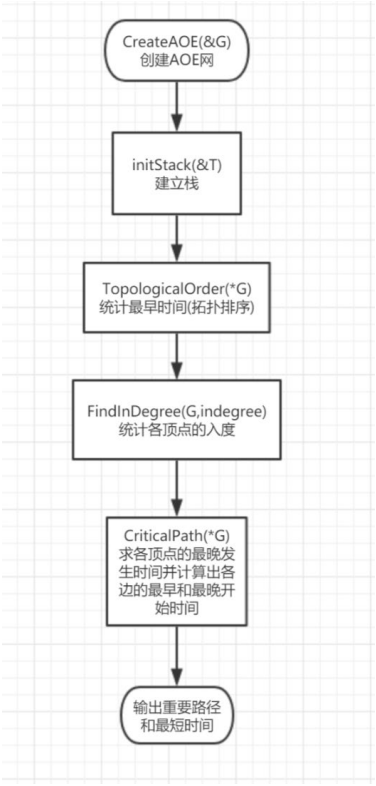


图 1 功能设计

2.算法设计

统计最早时间的算法设计：采用拓扑排序并进行最早时间计算

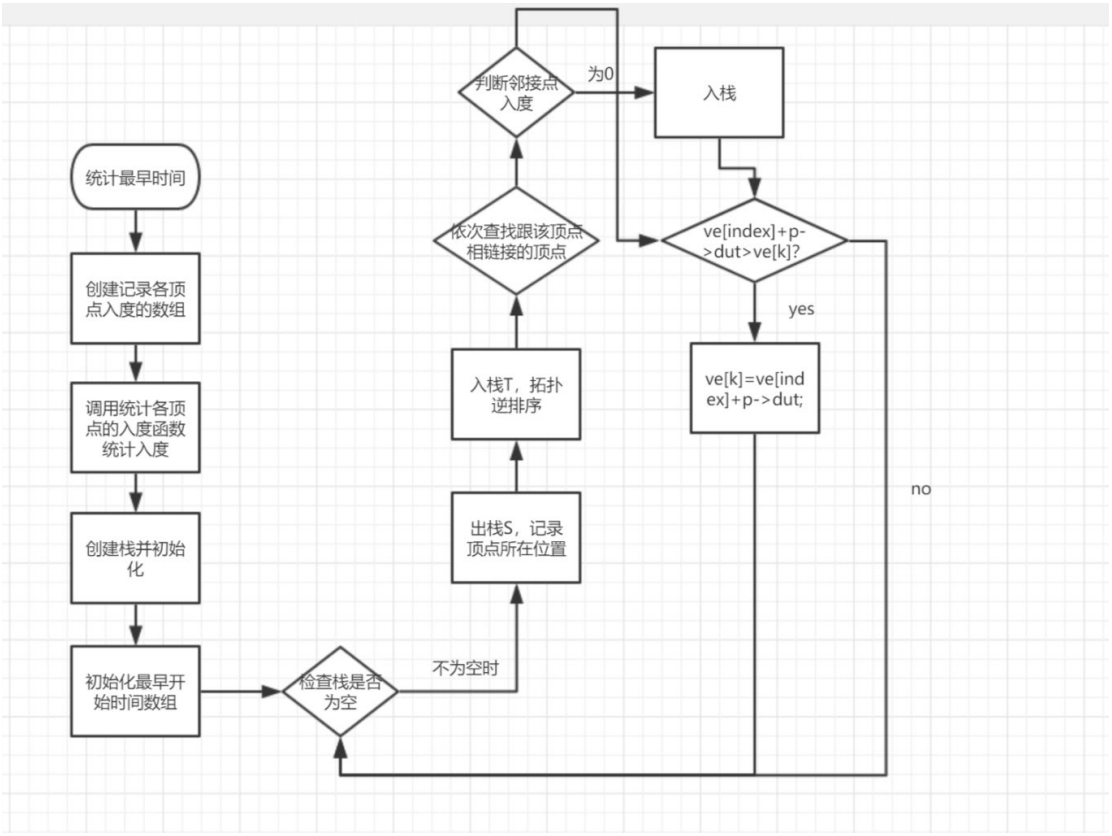


图 1：判断回路的算法设计

计算最迟发生时间与关键路径的算法设计：

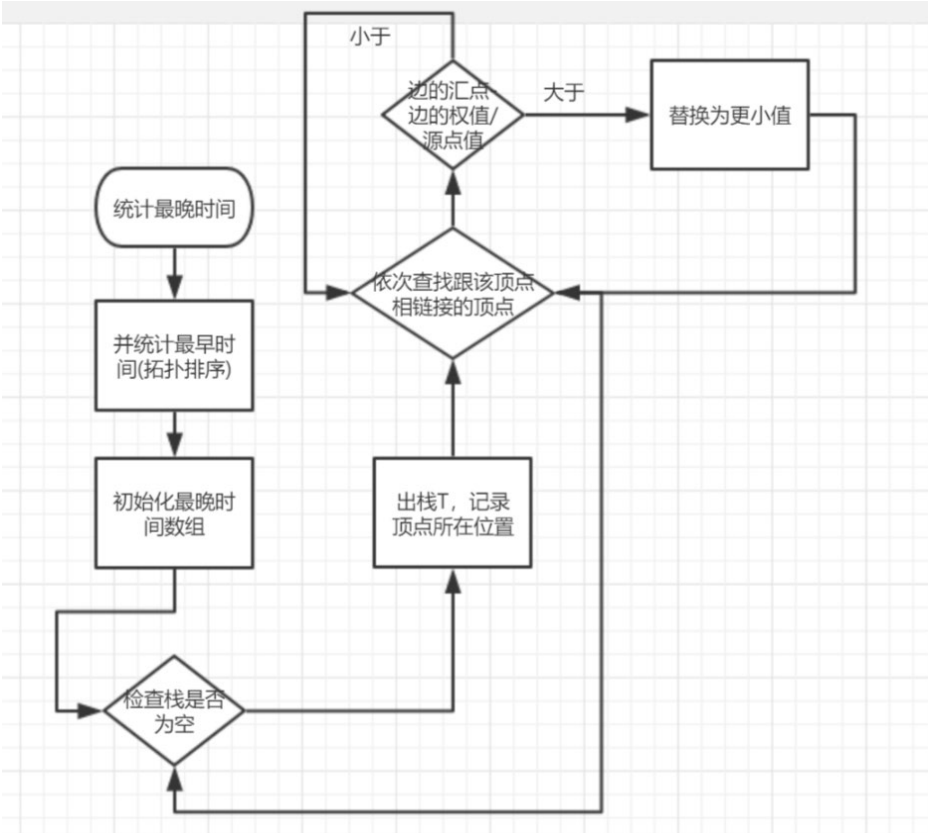


图 2：计算最迟发生时间与关键路径的算法设计

3.主要函数及结构体

主要函数：

```
void CreateAOE(ALGraph **G);                    //创建 AOE 网，构建邻接表
int LocateVex(ALGraph G,VertexType u);          //返回顶点对应邻接表数组中位置下标
void FindInDegree(ALGraph G,int indegree[]);    //统计各顶点的入度
bool TopologicalOrder(ALGraph G);
void CriticalPath(ALGraph G);                   //求各顶点的最晚发生时间并计算出各
边的最早和最晚开始时间
void initStack(stack* *S);                      //初始化栈结构
bool StackEmpty(stack S);                       //判断栈是否为空
void push(stack *S,VertexType u);               //进栈操作
void pop(stack *S,VertexType *i);               //弹出栈顶元素
```

结构体：

```
VertexType ve[MAX_VERTEX_NUM];//全局变量，保存边的最早开始时间
VertexType vl[MAX_VERTEX_NUM];//全局变量，保存边的最晚开始时间
typedef struct ArcNode{
    int adjvex;                 //邻接点在数组中的位置下标
    struct ArcNode *nextarc;//指向下一个邻接点的指针
    VertexType dut;         //信息域
}ArcNode;
typedef struct VNode{
    VertexType data;        //顶点的数据域
    ArcNode * firstarc;     //指向邻接点的指针
}VNode,AdjList[MAX_VERTEX_NUM];
typedef struct {
    AdjList vertices;       //图中顶点及各邻接点数组
    int vexnum,arcnum;          //记录图中顶点数和边或弧数
    int kind;               //记录图的种类
}ALGraph;
typedef struct stack{       //结构体定义栈结构
    VertexType data;
    struct stack * next;
}stack;
stack *T;
```

4．源程序代码清单

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_VERTEX_NUM 20     //最大顶点个数
#define VertexType int        //顶点数据类型
#define InfoType int          //图中弧或边包含信息类型
VertexType ve[MAX_VERTEX_NUM];//全局变量，保存边的最早开始时间
VertexType vl[MAX_VERTEX_NUM];//全局变量，保存边的最晚开始时间
```

```
typedef struct ArcNode{
    int adjvex;                 //邻接点在数组中的位置下标
    struct ArcNode *nextarc;//指向下一个邻接点的指针
    VertexType dut;         //信息域
}ArcNode;
typedef struct VNode{
    VertexType data;        //顶点的数据域
    ArcNode * firstarc;     //指向邻接点的指针
}VNode,AdjList[MAX_VERTEX_NUM];
typedef struct {
    AdjList vertices;       //图中顶点及各邻接点数组
    int vexnum,arcnum;          //记录图中顶点数和边或弧数
    int kind;               //记录图的种类
}ALGraph;
typedef struct stack{           //结构体定义栈结构
    VertexType data;
    struct stack * next;
}stack;
stack *T;
void CreateAOE(ALGraph **G);                    //创建 AOE 网，构建邻接表
int LocateVex(ALGraph G,VertexType u);          //返回顶点对应邻接表数组中位置下标
void FindInDegree(ALGraph G,int indegree[]);    //统计各顶点的入度
bool TopologicalOrder(ALGraph G);
void CriticalPath(ALGraph G);                   //求各顶点的最晚发生时间并计算出各
边的最早和最晚开始时间
void initStack(stack* *S);                      //初始化栈结构
bool StackEmpty(stack S);                       //判断栈是否为空
void push(stack *S,VertexType u);               //进栈操作
void pop(stack *S,VertexType *i);               //弹出栈顶元素
////////////////////////////////////////////////////////////////////////////////
int main(){
    ALGraph *G;
    CreateAOE(&G);//创建 AOE 网
    initStack(&T);
    TopologicalOrder(*G);
    CriticalPath(*G);
    return 0;
}
////////////////////////////////////////////////////////////////////////////////
void CreateAOE(ALGraph **G){                    //创建 AOE 网，构建邻接表
    *G=(ALGraph*)malloc(sizeof(ALGraph));
    printf("输入活动数和活动间路径个数<x,y>\n");
    scanf("%d,%d",&((*G)->vexnum),&((*G)->arcnum));//记录定顶点数和边数
    int i=0,k=(*G)->arcnum;
```

```
    for (i=0; i<(*G)->vexnum; i++) {            //创建邻接表表头结点
        (*G)->vertices[i].data=i+1;             //顶点数据域存储顶点编号
        (*G)->vertices[i].firstarc=NULL;        //将指向邻接点的指针设置为空
    }
    VertexType initial,end,dut;
    for (int j=0; j<(*G)->arcnum; j++) {        //创建顶点对应邻接表
        printf("输入结点依赖关系以及路径权值<x,y,z>(剩余%d 对)\n",k--);
        scanf("%d,%d,%d",&initial,&end,&dut);
        ArcNode *p=(ArcNode*)malloc(sizeof(ArcNode));
        p->adjvex=LocateVex(*(*G), end);        //找到路径结束点对应在邻接表数组中的位置下标
        p->nextarc=NULL;                        //将指向下一个邻接点的指针设置为空
        p->dut=dut;

        int locate=LocateVex(*(*G), initial);   //找到路径起始结点在邻接表数组中的位置下标
        p->nextarc=(*G)->vertices[locate].firstarc;//将指向下一个邻接点的指针指向路径起始点
        (*G)->vertices[locate].firstarc=p;      //将路径起始点的下一个邻接点指针指向路径终止点
    }
}
////////////////////////////////////////////////////////////////////////////////////////
int LocateVex(ALGraph G,VertexType u){          //找到顶点对应在邻接表数组中的位置下标
    for (int i=0; i<G.vexnum; i++) {
        if (G.vertices[i].data==u) {
            return i;
        }
    }
    return -1;
}
////////////////////////////////////////////////////////////////////////////////////////
void FindInDegree(ALGraph G,int indegree[]){  //统计各顶点的入度
    for (int i=0; i<G.vexnum; i++) {            //初始化数组，默认初始值全部为 0
        indegree[i]=0;
    }
    for (int i=0; i<G.vexnum; i++) {            //遍历邻接表，根据各链表中结点的数据域存储的各顶点位置下标，在 indegree 数组相应位置+1
        ArcNode *p=G.vertices[i].firstarc;
        while (p) {
            indegree[p->adjvex]++;
            p=p->nextarc;
        }
```

```
    }
}
////////////////////////////////////////////////////////////////////////////////
bool TopologicalOrder(ALGraph G){                    //检查图中是否存在回路
    int indegree[G.vexnum];                          //创建记录各顶点入度的数组
    FindInDegree(G,indegree);                        //统计各顶点的入度
    stack *S;
    initStack(&S);
    for (int i=0; i<G.vexnum; i++) {                 //数组初始化
        ve[i]=0;
    }
    for (int i=0; i<G.vexnum; i++) {                 //将入度不为零的结点入栈
        if (!indegree[i]) {
            push(S, i);
        }
    }
    int count=0;
     while (!StackEmpty(*S)) {                       //如果栈不为空
        int index;
        pop(S,&index);                               //弹栈，并记录栈中保存的顶点所在
邻接表数组中的位置
        push(T, index);                              //入栈 T，为求各边的最晚开始时间
做准备
        ++count;
        for (ArcNode *p=G.vertices[index].firstarc; p ; p=p->nextarc) {
            VertexType k=p->adjvex;                  //依次查找跟该顶点相链接的顶点，如果
初始入度为 1，当删除前一个顶点后，该顶点入度为 0
            if (!(--indegree[k])) {
                push(S, k);
             }
            if (ve[index]+p->dut>ve[k]) {            //如果边的源点的最长路径长度加上边的权
值比汇点的最长路径长度还长，就覆盖 ve 数组中对应位置的值，最终结束时，ve 数组中存
储的就是各顶点的最长路径长度
                ve[k]=ve[index]+p->dut;
             }
        }
    }
    if (count<G.vexnum) {
        printf("该图有回路");
        return false;
    }
    return true;
}
////////////////////////////////////////////////////////////////////////////////
```

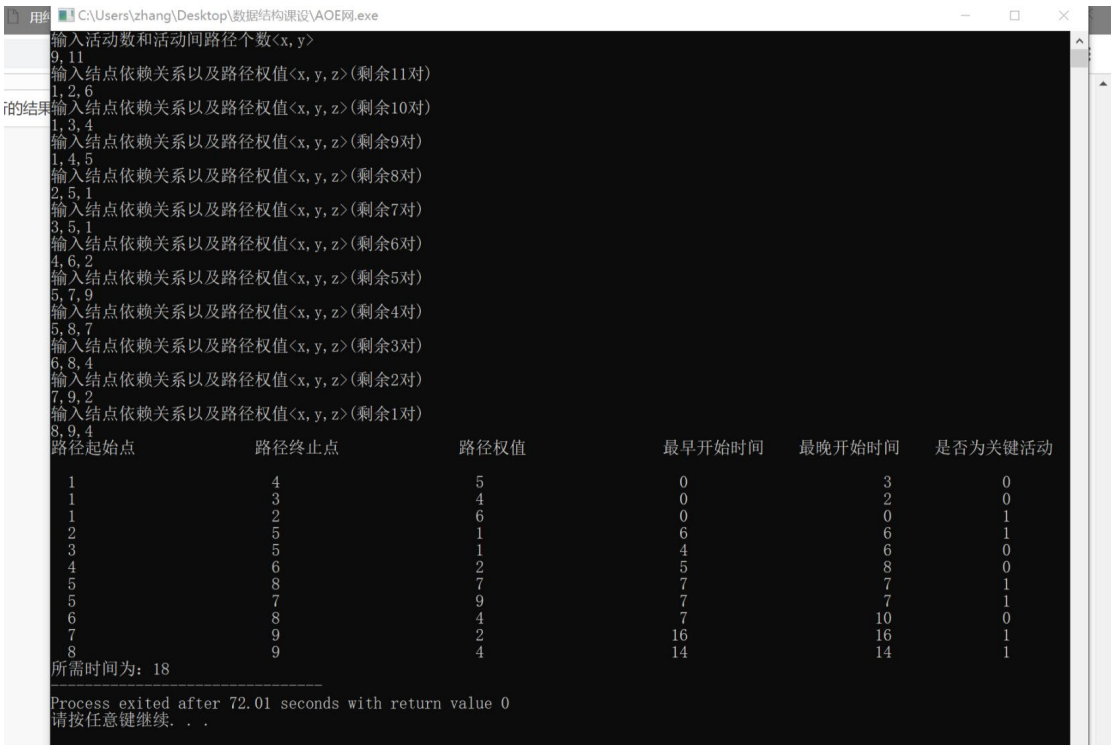
```
void CriticalPath(ALGraph G){                    //求各顶点的最晚发生时间并计算出各边的最早和最晚开始时间
    if (!TopologicalOrder(G)) {                  //检查图中是否存在回路
        return ;
    }
    for (int i=0 ; i<G.vexnum ; i++) {
        vl[i]=ve[G.vexnum-1];
    }
    int j,k,flag=0,a;
    while (!StackEmpty(*T)) {
        pop(T, &j);                                  //出栈，返回头结点位置
        for (ArcNode* p=G.vertices[j].firstarc ; p ; p=p->nextarc) {
            k=p->adjvex;
            if (vl[k]-p->dut<vl[j]) {            //构建 Vl 数组，在初始化时，Vl 数组中每个单元都是 18，如果每个边的汇点-边的权值比源点值小，就保存更小的
                vl[j] = vl[k]-p->dut;
            }
        }
    }
    printf("路径起始点\t\t 路径终止点\t\t 路径权值\t\t 最早开始时间\t 最晚开始时间\t 是否为关键活动\t\n");
    for (j = 0; j < G.vexnum; j++) {
        for (ArcNode*p = G.vertices[j].firstarc; p ;p = p->nextarc) {
            flag=0;
            k = p->adjvex;
            int ee = ve[j];                      //求各边的最早开始时间 e[i],等于 ve 数组中相应源点存储的值
            int el = vl[k]-p->dut;               //求各边的最晚开始时间 l[i]，等于汇点在 vl 数组中存储的值减边的权值
            if(ee==el){
                flag=1;

            }

    printf("%3d\t\t\t%3d\t\t\t%3d\t\t\t%3d\t\t\t%3d\t\t\t%d\n",j+1,k+1,p->dut,ee,el,flag);
            if(ee==el&&k+1==G.vexnum){
                a=el+p->dut;
              }
        }
    }
    printf("所需时间为：%d",a);
}
////////////////////////////////////////////////////////////////////////////////////////
void initStack(stack* *S){
```

```
    (*S)=(stack*)malloc(sizeof(stack));
    (*S)->next=NULL;
}
bool StackEmpty(stack S){
    if (S.next==NULL) {
        return true;
    }
    return false;
}
void push(stack *S,VertexType u){
    stack *p=(stack*)malloc(sizeof(stack));
    p->data=u;
    p->next=NULL;
    p->next=S->next;
    S->next=p;
}
void pop(stack *S,VertexType *i){
    stack *p=S->next;
    *i=p->data;
    S->next=S->next->next;
    free(p);
}
```

5．程序运行结果



6. 设计总结

本代码设计为实现关键路径算法在实际问题上的应用。运用邻接表，链栈构建了关键路径的寻找以及活动所需最小时间的计算。为实现本设计前后翻阅多本图书与资料并加以整理理解，最后体现为此模拟应用。通过本次代码设计加深了对 AOE 网的理解并了解其在实际中的使用价值。锻炼了语言编程能力，数据结构使用能力与对代码的调试能力。首次尝试为所有关键代码书写注释，便于后续对代码的持续性维护与理解。

参考文献

[1]维基百科 AOE 网 https://zh.wikipedia.org/wiki/AOE%E7%BD%91
[2]李春葆. 数据结构教程. 第五版. 北京：清华大学出版社, 2017：
[3]菜鸟教程 c 标准库参考手册 www.runoob.com
[4]Stephan14 判断图中是否有回路 blog.csdn.net/stephan14/article/details/41945555