

实验报告 (week-10)
2020111235 马靖淳

一. 实验任务

1. 拓扑排序
2. 关键路径
3. Dijkstra 算法
4. Floyd 算法

二. 实验上机时间

三. 知识点

1. 拓扑排序
2. 关键路径
3. Dijkstra 算法
4. Floyd 算法

四. 源代码及结果截屏

1. 拓扑排序

a. TopologicalSort.h

```
#pragma once

#ifndef TOPOLOGICALSORT_H
#define TOPOLOGICALSORT_H

#include <stdlib.h>
#include "Status.h"
#include "AdjacencyList.h"

Status TopologicalSort(ALGraph G);
void FindInDegree(ALGraph G, int indegree[]);

#endif
```

b. TopologicalSort.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "AdjacencyList.h"
#include "TopologicalSort.h"
#include "Stack.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

Status TopologicalSort(ALGraph G)
{
    //有向图采用邻接表存储结构
    SqStack S;
    int count, k, i;
    ArcNode* p;
```

```

int indegree[MAX_VERTEX_NUM];
FindInDegree(G, indegree); //对各顶点求入度indegree[0..vernum-1]
InitStack(&S);
for (i = 1; i < G.vexnum; ++i) //零入度顶点入栈S
    if (!indegree[i]) //入度为0者进栈
        Push(&S, i);
count = 0; //对输出顶点计数
while (!StackEmpty(S))
{
    Pop(&S, &i);
    printf("%c\n", G.vertices[i].data);
    ++count; //输出i号顶点并计数
    for (p = G.vertices[i].firstarc; p; p = p->nextarc)
    {
        k = p->adjvex; //对i号顶点的每个邻接点的入度减1
        if (!(--indegree[k]))
            Push(&S, k); //入度减为0入栈
    } //for
} //while
if (count < G.vexnum)
    return ERROR; //该有向图有回路
else
    return OK;
} //TopologicalSort

void FindInDegree(ALGraph G, int indegree[])
{
    //求顶点的入度
    int i;
    ArcNode* p;
    for (i = 1; i <= G.vexnum; i++)
        indegree[i] = 0; //赋初值
    for (i = 1; i <= G.vexnum; i++)
    {
        p = G.vertices[i].firstarc;
        while (p)
        {
            indegree[p->adjvex]++;
            p = p->nextarc;
        }
    }
}

```

c. Test.c

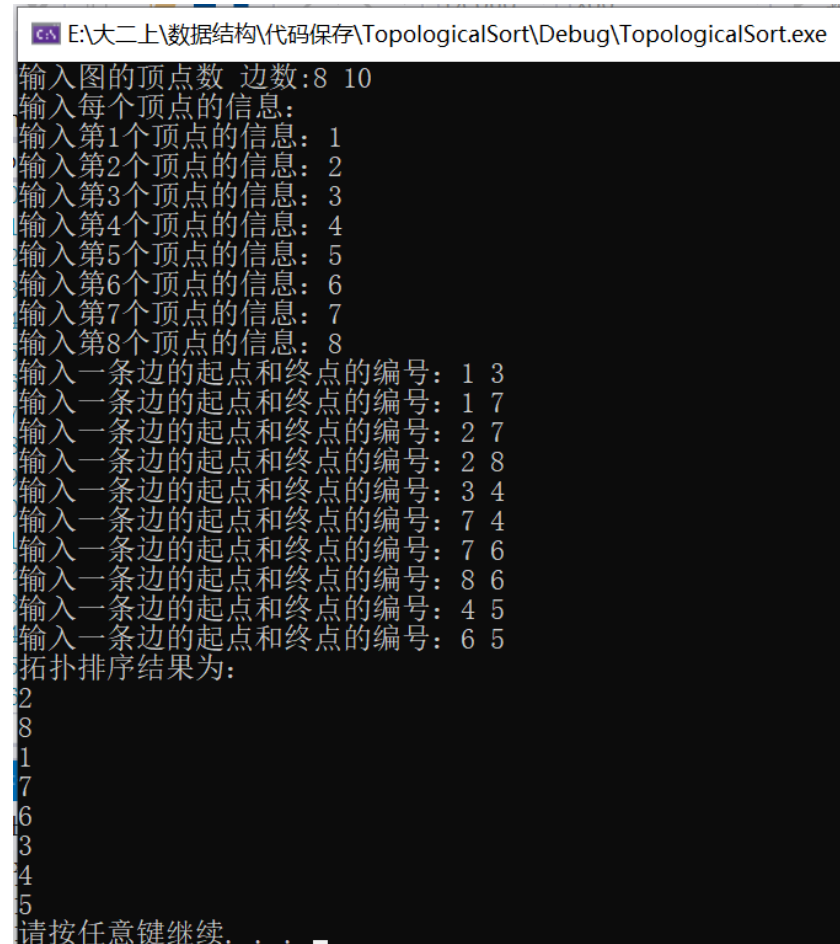
```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "TopologicalSort.h"

int main()
{
    ALGraph G;
    CreateUDG(&G);
    printf("拓扑排序结果为: \n");
    TopologicalSort(G);

    system("pause");
    return 0;
}

```

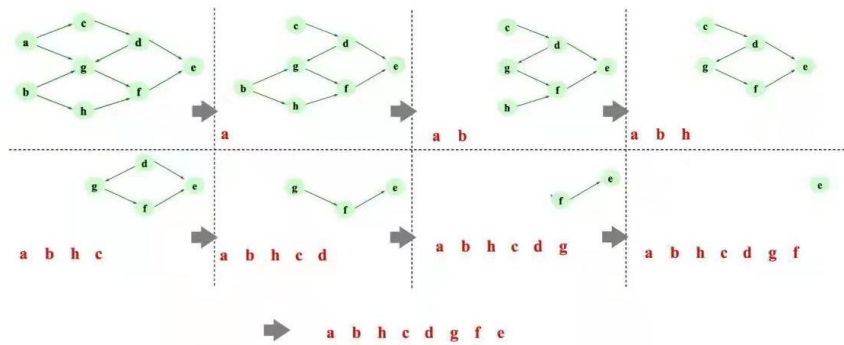


```

E:\大二上\数据结构\代码保存\TopologicalSort\Debug\TopologicalSort.exe
输入图的顶点数 边数:8 10
输入每个顶点的信息:
输入第1个顶点的信息: 1
输入第2个顶点的信息: 2
输入第3个顶点的信息: 3
输入第4个顶点的信息: 4
输入第5个顶点的信息: 5
输入第6个顶点的信息: 6
输入第7个顶点的信息: 7
输入第8个顶点的信息: 8
输入一条边的起点和终点的编号: 1 3
输入一条边的起点和终点的编号: 1 7
输入一条边的起点和终点的编号: 2 7
输入一条边的起点和终点的编号: 2 8
输入一条边的起点和终点的编号: 3 4
输入一条边的起点和终点的编号: 7 4
输入一条边的起点和终点的编号: 7 6
输入一条边的起点和终点的编号: 8 6
输入一条边的起点和终点的编号: 4 5
输入一条边的起点和终点的编号: 6 5
拓扑排序结果为:
2
8
1
7
6
3
4
5
请按任意键继续. . .

```

- 拓扑排序



- 一个有向图的拓扑序列不一定唯一：b a h c d g f e

2. 关键路径

1) TopologicalOrder.h

```
#pragma once
#ifndef TOPOLOGICALORDER_H
#define TOPOLOGICALORDER_H

#include <stdlib.h>
#include "Status.h"
#include "AdjacencyList.h"
#include "Stack.h"

int ve[MAX_VERTEX_NUM];

Status TopologicalOrder(ALGraph G, SqStack* T);
void FindInDegree(ALGraph G, int indegree[]);

#endif
```

2) CriticalPath.h

```
#ifndef CRITICALPATH_H
#define CRITICALPATH_H

#include <stdlib.h>
#include "Status.h"
#include "AdjacencyList.h"
#include "Stack.h"
#include "TopologicalOrder.h"

Status CriticalPath(ALGraph G);

#endif
```

3) TopologicalOrder.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "TopologicalOrder.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

Status TopologicalOrder(ALGraph G, SqStack* T)
{
    //有向图采用邻接表存储结构, 求各事件的最早发生时间ve (全局变量)。
    //T为拓扑序列顶点栈, S为零入度顶点栈
    SqStack S;
    int count = 0;
    int k, j;
    int indegree[40];
    ArcNode* p;
    InitStack(&S);
    FindInDegree(G, indegree); //对各顶点求入度indegree
    for (j = 1; j <= G.vexnum; ++j) //零入度顶点入栈S
        if (!indegree[j]) //入度为0者进栈
            Push(&S, j);
    InitStack(&(*T)); //建拓扑序列顶点栈T
    count = 0; //对输出顶点计数
    for (int i = 1; i <= G.vexnum; i++)
        ve[i] = 0; //初始化
    while (!StackEmpty(S))
    {
        Pop(&S, &j);
        Push(&(*T), j);
        ++count; //顶点入栈并计数
        for (p = G.vertices[j].firstarc; p; p = p->nextarc)
        {
            k = p->adjvex; //对j号顶点的每个邻接点的入度减1
            if (--indegree[k] == 0)
                Push(&S, k); //入度为0, 入栈
            if (ve[j] + *(p->info) > ve[k])
                ve[k] = ve[j] + *(p->info);
        } //*(p->info)=dut(<j, k>)
    } //while
    if (count < G.vexnum)
        return ERROR; //该有向图有回路
    else
        return OK;
}
```

```

} // TopologicalOrder

void FindInDegree(ALGraph G, int indegree[])
{
    // 求顶点的入度
    int i;
    ArcNode* p;
    for (i = 1; i <= G.vexnum; i++)
        indegree[i] = 0; // 赋初值
    for (i = 1; i <= G.vexnum; i++)
    {
        p = G.vertices[i].firstarc;
        while (p)
        {
            indegree[p->adjvex]++;
            p = p->nextarc;
        }
    }
}

```

4) CriticalPath.c

```

#define _CRT_SECURE_NO_WARNINGS
#include "CriticalPath.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

Status CriticalPath(ALGraph G)
{
    // G为有向网，输出G的关键活动
    SqStack T;
    int a, j, k, el, ee, dut;
    char tag;
    ArcNode* p;
    int vl[MAX_VERTEX_NUM];
    if (!TopologicalOrder(G, &T)) // 产生有向环
        return ERROR;
    // j=ve[0];
    for (a = 1; a <= G.vexnum; a++)
        vl[a] = ve[G.vexnum];
    while (!StackEmpty(T)) // 按拓扑逆序求各顶点的vl值
        for (Pop(&T, &j), p = G.vertices[j].firstarc; p; p = p->nextarc)
        {

```

```

        k = p->adjvex;
        dut = *(p->info); //dut<j,k>
        if (vl[k] - dut < vl[j])
            vl[j] = vl[k] - dut;
    } //for
    for (j = 1; j <= G.vexnum; ++j) //求ee,el的关键活动
        for (p = G.vertices[j].firstarc; p; p = p->nextarc)
        {
            k = p->adjvex;
            dut = *(p->info);
            ee = ve[j];
            el = vl[k] - dut;
            tag = (ee == el) ? '*' : ' ';
            printf("%d %d %d %d %d %c\n", j, k, dut, ee, el, tag); //输出关键活动
        } //for
    return OK;
}

```

5) test.c

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "CriticalPath.h"

int main()
{
    ALGraph G;
    CreateUDG(&G);
    CriticalPath(G);

    system("pause");
    return 0;
}

```

C:\选择E:\大二上\数据结构\代码保存\CriticalPath\Debug\Critic

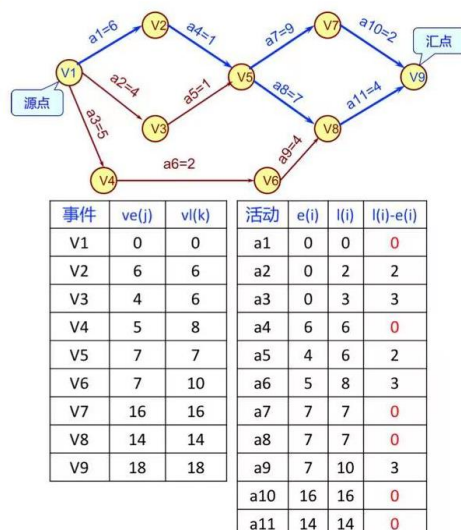
```

输入图的顶点数 边数:9 11
输入每个顶点的信息:
输入第1个顶点的信息: 1
输入第2个顶点的信息: 2
输入第3个顶点的信息: 3
输入第4个顶点的信息: 4
输入第5个顶点的信息: 5
输入第6个顶点的信息: 6
输入第7个顶点的信息: 7
输入第8个顶点的信息: 8
输入第9个顶点的信息: 9
输入一条边的起点和终点的编号以及权值: 1 2 6
输入一条边的起点和终点的编号以及权值: 1 3 4
输入一条边的起点和终点的编号以及权值: 1 4 5
输入一条边的起点和终点的编号以及权值: 2 5 1
输入一条边的起点和终点的编号以及权值: 3 5 1
输入一条边的起点和终点的编号以及权值: 4 6 2
输入一条边的起点和终点的编号以及权值: 5 7 9
输入一条边的起点和终点的编号以及权值: 5 8 7
输入一条边的起点和终点的编号以及权值: 6 8 4
输入一条边的起点和终点的编号以及权值: 7 9 2
输入一条边的起点和终点的编号以及权值: 8 9 4
1 2 6 0 0 *
1 3 4 0 2
1 4 5 0 3
2 5 1 6 6 *
3 5 1 4 6
4 6 2 5 8
5 7 9 7 7 *
5 8 7 7 7 *
6 8 4 7 10
7 9 2 16 16 *
8 9 4 14 14 *
请按任意键继续. . .

```

关键路径

- 关键路径



五. 实验总结

1. 注意拓扑排序等使用的是有向图，要对图的创建函数进行修改

```
Status CreateUDG(ALGraph* G)
{
    ArcNode* p, * q;
    int i, j, k;
    VertexType v1, v2;
    int w;
    ArcNode* r[MAX_VERTEX_NUM + 1];
    printf("输入图的顶点数 边数:");
    scanf("%d %d", &((*G).vexnum), &((*G).arcnum)); //顶点和边个数
    getchar();
    printf("输入每个顶点的信息: \n");
    for (i = 1; i <= (*G).vexnum; i++)
    {
        printf("输入第%d个顶点的信息: ", i);
        scanf("%c", &((*G).vertices[i].data)); //顶点信息
        getchar();
        (*G).vertices[i].firstarc = NULL;
        r[i] = NULL;
    }

    for (k = 1; k <= (*G).arcnum; k++)
    {
        //依次输入每个边信息
        printf("输入一条边的起点和终点的编号以及权值: ");
        scanf("%c %c %d", &v1, &v2, &w);
        getchar();
        i = LocateVex(*G, v1); //需查找顶点编号
        j = LocateVex(*G, v2);

        p = (ArcNode*)malloc(sizeof(ArcNode));
        if (!p)
            exit(OVERFLOW);
        p->adjvex = j;
        p->info = (int*)malloc(sizeof(int));
        *(p->info) = w;
        p->nextarc = NULL;
        if (r[i] == NULL) //邻接表中第一个结点
            (*G).vertices[i].firstarc = p; //加入到邻接表
        else
            r[i]->nextarc = p;
        r[i] = p;
    }
}
```

```
        return OK;  
    } //CreateUDG
```

删去无向图创建部分

2. 要注意书写循环时，i, j, k 的初始值和结束条件