《数据结构》上机报告

2018 年 12 月 1 日

姓名：刘思源　学号：1651390　班级：电子三班　得分：

<table>
<tr><td>实验题目</td><td colspan="2">图</td></tr>
<tr><td>问题描述</td><td colspan="2">有向无环图（directed acyclic graph,简称 DAG 图），是描述工程或系统的进行过程的有效工具。如果有向图中从顶点 v 到 w 有一条有向路径，则 v 一定排在 w 之前，这样构成的一个顶点序列就称为拓扑序，构造拓扑序的过程就是拓扑排序。若拓扑排序不能输出所有顶点，说明 AOV 网络中存在有向环，此 AOV 网络所代表的工程是不可行的。 拓扑排序的算法有 2 种：1 种是删边法，另一种是采用 DFS 深度优先搜索的方法。 本题给出一组有向图，请采用一种拓扑排序方法判断该图是否有环。</td></tr>
<tr><td rowspan="2">基本要求</td><td colspan="2">1. (p1) 本题给出一组有向图，请采用一种拓扑排序方法判断该图是否有环。(
2. （p2）一个工程项目由一组活动（或称子任务）构成，活动之间有的可以并行执行，有的必须在完成了其它一些活动后才能执行，并且每个活动完成需要一定的时间。对于一个工程，需要研究的问题是：(1) 由这样一组活动描述的工程是否可行？ (2) 若可行，计算完成整个工程需要的最短时间。(3) 这些活动中，哪些活动是关键活动（也就是必须按时完成的任务，否则整个项目就要延迟）。 现给定一个 AOE 网，有向边（弧）表示活动，弧上权值表示活动完成需要的时间。请你编写程序，回答上述三个问题。</td></tr>
<tr><td>已完成基本内容（序号）：</td><td>1，2</td></tr>
<tr><td rowspan="2">选做要求</td><td colspan="2"></td></tr>
<tr><td>已完成选做内容（序号）</td><td></td></tr>
<tr><td>数据结构设计</td><td colspan="2"><pre>typedef int Status;
typedef int ElemType;

typedef struct ArcNode {
    int adjvex;
    int weight;
    struct ArcNode *next;
}ArcNode;

typedef struct Node {
    int date;
    int info;</pre></td></tr>
</table>

<table>
<tr><td></td><td>

```
    ArcNode *first;
}Node, AdjList[MAXSIZE];

typedef struct {
    AdjList vertices;
    int vexnum;
    int arcnum;
}MGraph;
int arr[MAXSIZE];
```

</td></tr>
<tr><td>功能(函数)说明</td><td>

1、 判断图是否成环

假设图以邻接矩阵表示，一条深度遍历路线中如果有结点被第二次访问到，那么有环。我们用一个变量来标记某结点的访问状态（未访问，访问过，其后结点都被访问过），然后判断每一个结点的深度遍历路线即可。

因为采用邻接矩阵存储，一般至少需要将矩阵中元素的一半给过一下，由于矩阵元素个数为 n^2， 因此时间复杂度就是 O(n^2)。如果采用邻接表存储，则只存储了边结点(e 条边，无向图是 2e 条边)，加上表头结点为 n（也就是顶点个数），因此时间复杂度为 O(n+e)。

当然也可以使用拓扑排序的方法：

方法是重复寻找一个入度为 0 的顶点，将该顶点从图中删除（即放进一个队列里存着，这个队列的顺序就是最后的拓扑排序，具体见程序），并将该结点及其所有的出边从图中删除（即该结点指向的结点的入度减 1），最终若图中全为入度为 1 的点，则这些点至少组成一个回路。

采用邻接矩阵存储时，遍历二维数组，求各顶点入度的时间复杂度是 O(n^2)。遍历所有结点，找出入度为 0 的结点的时间复杂度是 O(n)。对于 n 个入度为 0 的结点，删除他们的出边的复杂度为 O(n^2)。 所以总的复杂度为 O(n^2)。

对于邻接表，遍历所有边，求各顶点入度的时间复杂度是 O(e)，即边的个数。遍历所有结点，找出入度为 0 的结点的时间复杂度是 O(n)，即顶点的个数。遍历所有边，删除入度为 0 的结点的出边的复杂度为 O(e)，即边的个数。所以总的时间复杂度是 O(n+e)。

```
void dfs(MGraph *G, int i)
{
    color[i] = -1;
    pre[i] = ++point;
    for (int j = 1; j <= G->vexNum; j++){
        if (G->arcs[i][j] != 0){
            if (color[j] == -1){
                is_DAG = false;
            }
            else if (color[j] == 0)
                dfs(G, j);
        }
```

</td></tr>
</table>

```
    }
    post[i] = ++point;
    color[i] = 1;
}

void DFS(MGraph *G)
{
    int i;
    for (i = 1; i <= G->vexNum; i++){
        color[i] = 0;
        pre[i] = 0;
        post[i] = 0;
    }
    for (i = 1; i <= G->vexNum; i++)
    {
        if (color[i] == 0){
            dfs(G, i);

        }
    }
}
```

2、 关键路径

首先使用拓扑排序确定该有向图有没有环，如果存在，证明该工程无解，返回 0；否则，再求关键路径。

要准备两个数组：最早开始时间数组和最迟开始时间数组。从源点 V0 出发，令 etv[0]（源点）=0，按拓扑有序求其余各顶点的最早发生时间 etv[i]（1 ⩽ i ⩽ n-1）。从汇点 Vn 出发，令 ltv[n-1] = etv[n-1]，按拓扑排序求各个其余各顶点的最迟发生时间 ltv[i]（n-2 ⩾ i ⩾ 2）;根据各顶点的 etv 和 ltv 数组的值，求出弧（活动）的最早开工时间和最迟开工时间，求每条弧的最早开工时间和最迟开工时间是否相等，若相等，则是关键活动。应当注意的是，1，2 完成点（事件）的最早和最迟。3 根据事件来计算活动最早和最迟，从而求的该弧（活动）是否为关键活动。

```
bool topo(MGraph G)
{
    SqQueue Q;
    InitQueue(Q);
    int i,sum = 0, flag[MAXSIZE];
    for (i = 0; i < MAXSIZE; i++) {
        flag[i] = false;
    }
    for (i = 1; i <= G.vexnum; i++) {
        if (G.vertices[i].info == 0) {
            EnQueue(Q, i);
            flag[i] = 1;
```

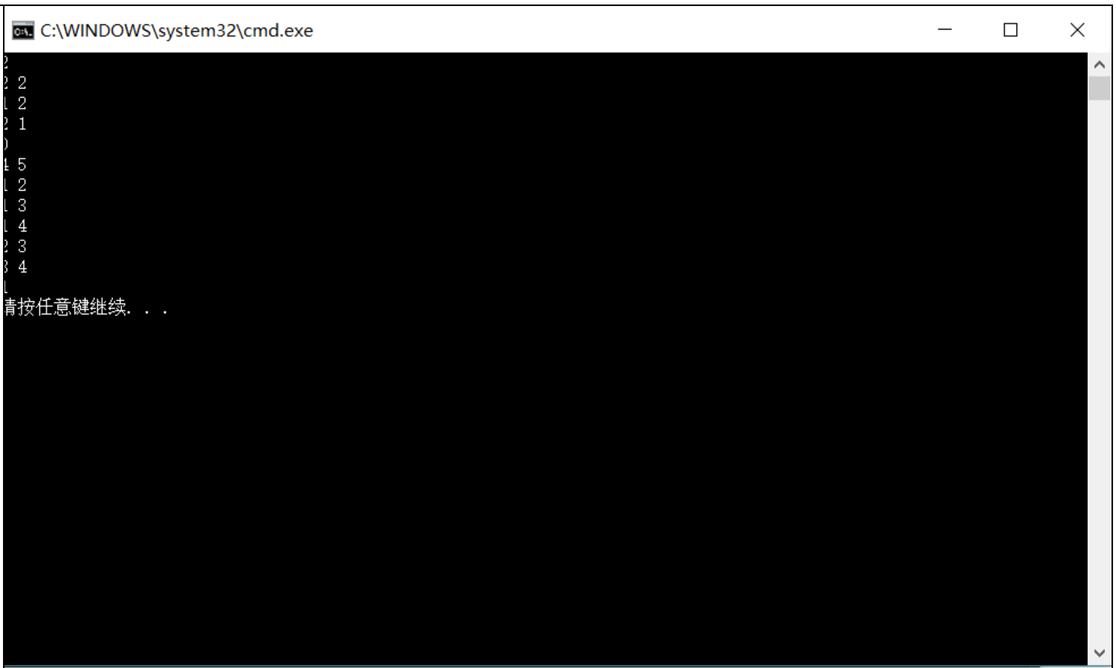
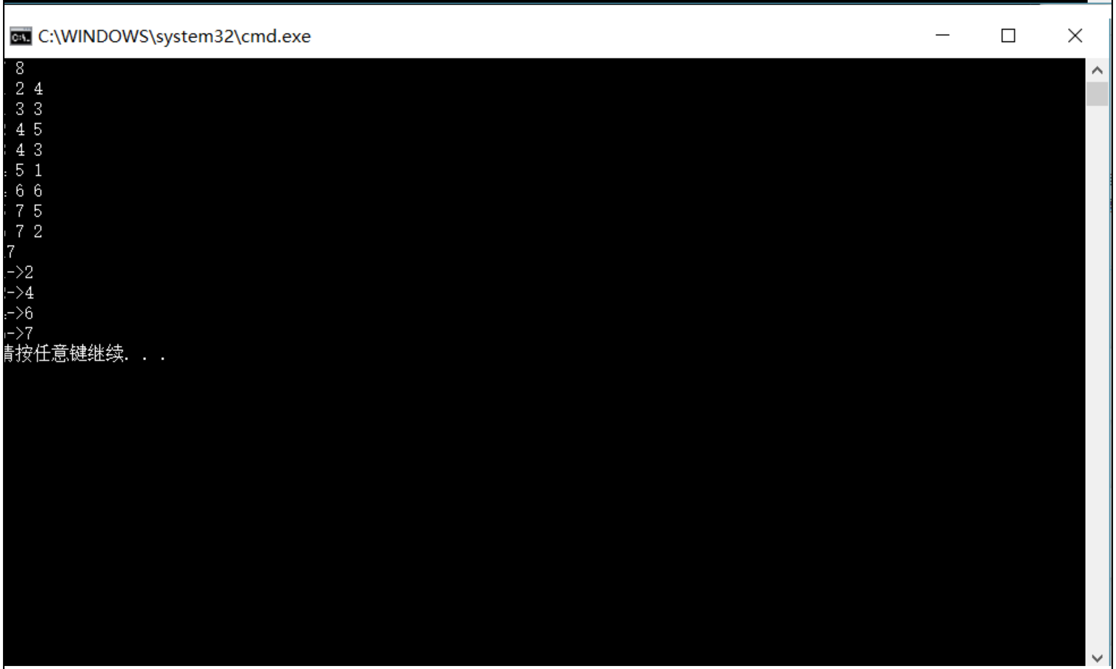
```
        }
    }

    while(!QueueEmpty(Q)){
        int t = Q.base[Q.front];
        sum++;
        DeQueue(Q);
        arr[sum] = t;
        ArcNode *pos = G.vertices[t].first;
        while (pos != NULL) {
            G.vertices[pos->adjvex].info--;
            pos = pos->next;
        }

        for (i = 1; i <= G.vexnum; i++) {
            if (G.vertices[i].info == 0 && !flag[i]) {
                EnQueue(Q, i);
                flag[i] = 1;
            }
        }
    }
    return sum == G.vexnum ? true : false;
}
void CriticalPath(MGraph G)
{
    int ve[MAXSIZE], vl[MAXSIZE];
    int i;
    for (i = 1; i <= G.vexnum; i++) {
        ve[i] = 0;
    }
    for (i = 1; i <= G.vexnum; i++){
        ArcNode *pos = G.vertices[arr[i]].first;
        while (pos != NULL) {
            int t = pos->adjvex;
            if (ve[arr[i]] + pos->weight > ve[t]) {
                ve[t] = ve[arr[i]] + pos->weight;
            }
            pos = pos->next;
        }
    }

    for (i = 1; i <= G.vexnum; i++) {
        vl[i] = ve[arr[G.vexnum]];
    }
```

	`` cout << ve[arr[G.vexnum]] << endl; for (i = G.vexnum; i >= 1; i--){ ArcNode *pos = G.vertices[arr[i]].first; while (pos != NULL) { int t = pos->adjvex; if (vl[t] - pos->weight < vl[arr[i]]) vl[arr[i]] = vl[t] - pos->weight; pos = pos->next; } } for (i = 1; i <= G.vexnum; i++){ ArcNode *pos = G.vertices[i].first; while (pos != NULL) { int t = pos->adjvex; int e = ve[i]; int l = vl[t] - pos->weight; if (e == l) { cout << i << "->" << t << endl; } pos = pos->next; } } } ```
开发环境	Visual studio 2017

调试分析	 
心得体会	1、 关于关键路径的思考：我们寻找关键路径——关键路径就是关键活动（顶点与顶点之间的边组成），就是我们怎么判断该顶点是否为关键活动（边）的顶点，即判断边是否为关键活动。前面定义过，关键路径就是图中从源点到汇点最长（权值最大）的路径。这条路径就决定了整个工程的工期，这说明一个什么问题？ 关键路径上的顶点与顶点之间的活动的应该最早开始和最迟开始时间是相等的，如果不等那么说明活动还有余额时间（在最早开始时间和最迟开始时间之间可以任选一个时间点开始），这说明还有其他活动是决定这个工程时间的，那就不是关键路径了。 2、 判断无向图是否有环： 求出图中所有顶点的度， 删除图中所有度<=1 的顶点以及与该顶点相关的边，把与这些边相关的顶点

	的度减一 如果还有度<=1 的顶点重复步骤 2 最后如果还存在未被删除的顶点，则表示有环；否则没有环 时间复杂度为 O（E+V），其中 E、V 分别为图中边和顶点的数目。