

数据结构课程设计

——关键活动

项目说明文档

作 者 姓 名： 韦世贸

学 号： 2351131

指 导 老 师： 软件学院

专 业： 软件工程

同济大学

Tongji University

1 介绍

1.1 背景介绍

迷宫指的是充满复杂通道，很难找到从其内部到达入口或从入口到达中心的道路，道路复杂难辨，人进去不容易出来的建筑物。通常比喻复杂艰深的问题或难以捉摸的局面。

求解“关键活动”问题对现实生活的工作也有很大的帮助，所以此次的项目就“关键活动”问题展开探究。

1.2 功能介绍

这个程序能够判定一个给定的工程项目的任务调度是否可行；如果该调度方案可行，则计算完成整个项目需要的最短时间，并且输出所有的关键活动。

2 设计

2.1 数据结构设计

该本项目的实质是对关键路径的寻找，所以本项目数据结构采用结构体图，并且将图的度的转化到数组中解决，可以更加简化，同时还使用了链式队列来进行拓扑排序。

2.2 类结构设计

在该项目中创建一个图结构体和一个链式队列，用栈来进行拓扑排序，大大简化了代码的复杂度。

2.3 成员与操作设计

图结构体的代码如下

// 定义边的结构体，包含起点u、终点v、权重w和输入顺序input\_order

struct Edge

{

int u, v, w, input\_order;

};

链式队列的实现都是很常规的写法，这里用的是模板类栈，代码省略。

3 实现

3.1 创建图功能的实现

3.1.1 创建图功能流程简介

创建图是为了建立各种任务之间的关系已经存储消耗的时间，这里用动态分配数组内存的方式来存储图的各种信息。

3.1.2 创建图功能核心代码（动态分配数组内存）

LinkedQueue <int>topo\_queque; // 用于拓扑排序的队列

int N, M; // 节点数和边数

cout << "给出两个正整数N（N《=100）和M，其中N是任务交接点的数量，交接点按1～N编号，M是子任务的数量" << endl;

cin >> N >> M; // 读取节点数和边数

// 动态分配数组内存

struct Edge \*edges=new Edge[M]; // 存储所有边

int\* indegree = new int[N + 1]();// 存储每个节点的入度

int\* outdegree = new int[N + 1](); // 存储每个节点的出度

int \*earliest = new int[N + 1](); // 存储每个节点的最早完成时间

int \*latest = new int[N + 1](); // 存储每个节点的最晚完成时间

int \*topo\_order = new int[N + 1](); // 存储拓扑排序顺序

// 读取所有边的信息，并统计入度和出度

cout << "输入该任务开始和完成设计的交接点编号以及完成该任务所需要的时间，整数间用空格分隔: " << endl;

for (int i = 0; i < M; i++) {

cin >> edges[i].u >> edges[i].v >> edges[i].w;

edges[i].input\_order = i; // 记录边的输入顺序

indegree[edges[i].v]++; // 终点的入度增加

outdegree[edges[i].u]++; // 起点的出度增加

}

3.2拓扑排序和寻找关键活动功能的实现

3.2.1拓扑排序功能简介

拓扑排序是为了检验任务调度方案是否可行，如果不可行，直接结束程序。

若拓扑有序，则更新最晚完成时间，根据公式 latest[v] - earliest[u] - w == 0 找出关键活动

3.2.2拓扑排序核心代码

// 初始化拓扑排序队列，将所有入度为0的节点加入队列

for (int u = 1; u <= N; u++) {

if (indegree[u] == 0) {

topo\_queque.EnQueue(u);

}

}

int topo\_len = 0; // 拓扑排序的长度

// 执行拓扑排序

while (!topo\_queque.isEmpty()) {

int u; // 取出队列头部的节点

topo\_queque.DeQueue(u);

topo\_order[topo\_len++] = u; // 将该节点加入拓扑排序顺序

// 遍历所有边，更新入度

for (int j = 0; j < M; j++) {

if (edges[j].u == u) {

indegree[edges[j].v]--; // 终点的入度减1

if (indegree[edges[j].v] == 0) {

// 如果入度为0，加入队列

topo\_queque.EnQueue(edges[j].v);

}

}

}

}

// 检查是否有环

if (topo\_len != N) {

cout << "任务调度不可行！" << endl;

cout << 0 << endl; // 有环，输出0

}

else {

// 初始化最早完成时间为0

for (int u = 1; u <= N; u++) {

earliest[u] = 0;

}

// 按照拓扑顺序更新最早完成时间

for (int i = 0; i < topo\_len; i++) {

int u = topo\_order[i];

// 遍历所有从u出发的边，更新终点的最早完成时间

for (int j = 0; j < M; j++) {

if (edges[j].u == u) {

if (earliest[edges[j].v] < earliest[u] + edges[j].w) {

earliest[edges[j].v] = earliest[u] + edges[j].w;

}

}

}

}

int total\_time = 0; // 总时间

// 找到所有终点节点的最早完成时间的最大值

for (int u = 1; u <= N; u++) {

if (outdegree[u] == 0) { // 如果是终点节点

if (earliest[u] > total\_time) {

total\_time = earliest[u];

}

}

}

// 初始化最晚完成时间

for (int u = 1; u <= N; u++) {

if (outdegree[u] == 0) { // 如果是终点节点

latest[u] = earliest[u];

}

else {

latest[u] = 999999; // 其他节点初始化为一个大数

}

}

// 从后往前更新最晚完成时间

for (int i = topo\_len - 1; i >= 0; i--) {

int u = topo\_order[i];

if (outdegree[u] == 0) {

latest[u] = earliest[u]; // 终点节点的最晚完成时间等于最早完成时间

}

else {

int min\_latest = 9999999; // 初始化为一个大数

// 遍历所有从u出发的边，找到最小的latest[v] - w

for (int j = 0; j < M; j++) {

if (edges[j].u == u) {

if (latest[edges[j].v] - edges[j].w < min\_latest) {

min\_latest = latest[edges[j].v] - edges[j].w;

}

}

}

latest[u] = min\_latest; // 更新u的最晚完成时间

}

}

3.3关键活动的排序

3.3.1关键活动排序简介

按照要求，任务开始的交接点编号小者优先，起点编号相同时，与输入时任务的顺序相反。所以在定义图的时候增加了一个记录输入顺序的变量。这里排序关键活动使用的是冒泡排序，按照u升序排序，如果u相同，则按输入顺序降序排序

3.3.2关键活动排序核心代码

// 按照u升序排序，如果u相同，则按input\_order降序排序

void sort\_critical\_activities(struct Edge\* activities, int count)

{

for (int i = 0; i < count - 1; i++) {

for (int j = 0; j < count - 1 - i; j++) {

// 如果当前边的u大于下一条边的u，或者u相同但当前边的input\_order小于下一条边的input\_order

if (activities[j].u > activities[j + 1].u ||

(activities[j].u == activities[j + 1].u && activities[j].input\_order < activities[j + 1].input\_order))

{

// 交换两条边

struct Edge temp = activities[j];

activities[j] = activities[j + 1];

activities[j + 1] = temp;

}

}

}

}

3.4总体功能的实现

在实现总体功能时，我增加了输入和输出提示，让整个程序结果更加直观。

3.4.1总体功能实现截屏

测试用例1：

7 8

1 2 4

1 3 3

2 4 5

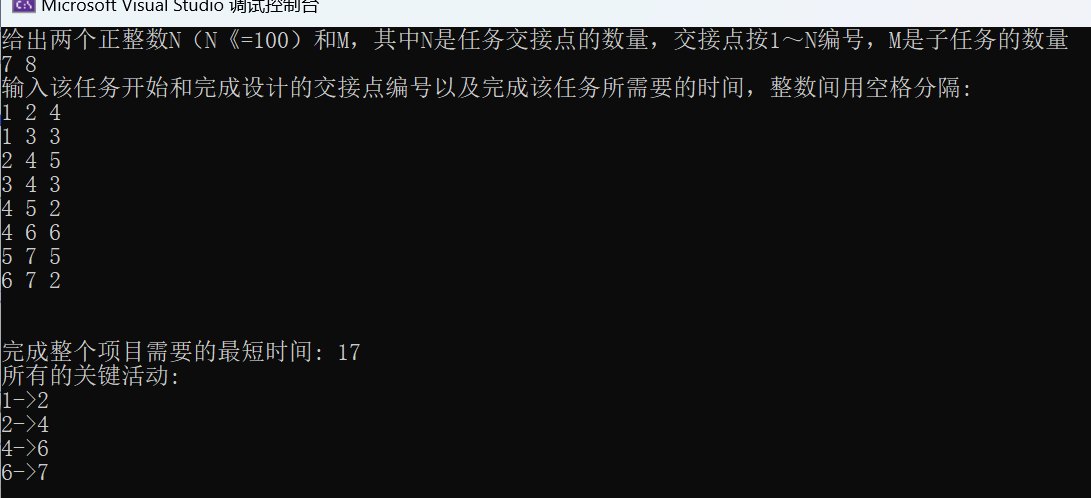
3 4 3

4 5 2

4 6 6

5 7 5

6 7 2



测试用例2：

9 11

1 2 6

1 3 4

1 4 5

2 5 1

3 5 1

4 6 2

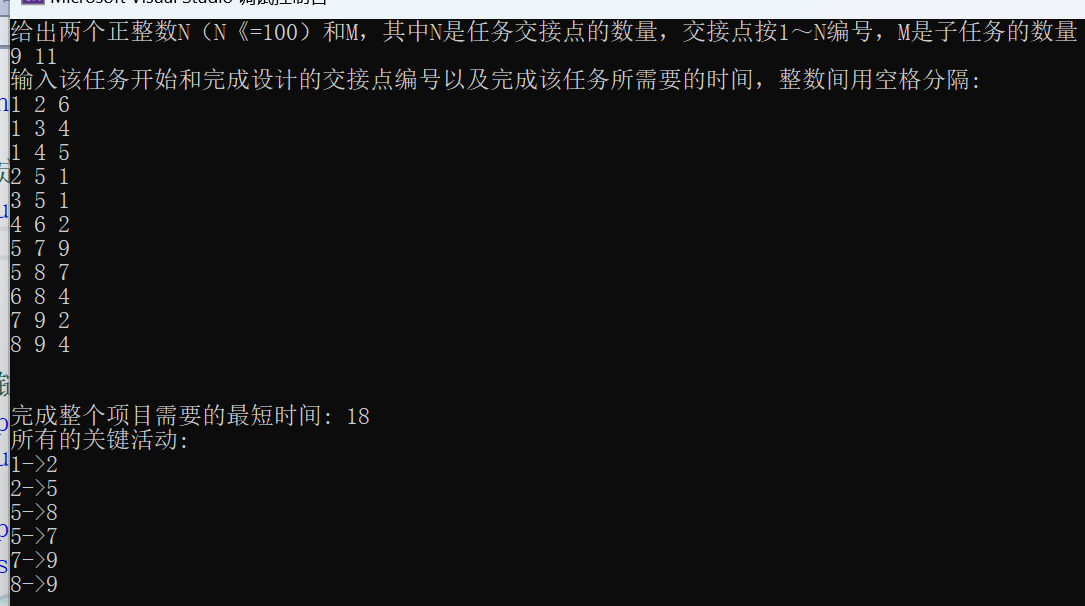
5 7 9

5 8 7

6 8 4

7 9 2

8 9 4



测试用例3：（不可行方案）

4 5

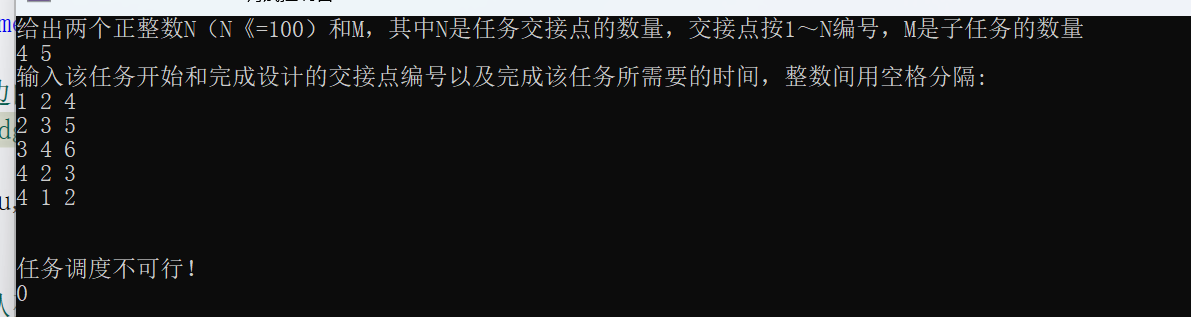
1 2 4

2 3 5

3 4 6

4 2 3

4 1 2



4.设计小结

通过设计这个程序，我复习了拓扑排序和寻找关键活动并求解完成整个项目的最短时间的知识。同时也学会了用队列进行拓扑排序，用结构体实现的图来完成这个项目，我还复习了冒泡排序法。

但是这个程序的错误处理设置得比较少，希望在以后的编程可以更加注重这个方面，同时边集也可以使用模板类进行定义，适合不同的情况。