数据结构课程设计

项目说明文档

关键活动

软件工程

张靖凯

2151396

目录

项目内容

本实验项目是要求在任务调度问题中，如果还给出了完成每个字任务需要的时间，则可以算出完成整个工程项目需要的最短时间。在这些子任务中，有些任务即使推迟几天完成，也不会影响全局的工期；但是有些任务必须准时完成，否则整个项目的工期就要因此而延误，这些任务叫做“关键活动”。

请编写程序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。

项目功能要求

1 输入说明：输入第 1 行给出两个正整数 N（N《=100）和 M，其中 N 是任务交接点（即衔接两个项目依赖的两个子任务的结点，例如：若任务 2 要在任务 1 完成后才开始，则两个任务之间必有一个交接点）的数量，交接点按 1～N 编号，M 是字任务的数量，依次编号为 1～M。随后 M 行，每行给出 3 个正整数，分别是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。

2 输出说明：如果任务调度不可行，则输出 0；否则第一行输出完成整个项目所需要的时间，第 2 行开始输出所有关键活动，每个关键活动占一行，按照格式“v->W”输出，其中 V 和 W 为该任务开始和完成涉及的交接点编号。关键活动输出的顺序规则是：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。如下面测试用例 2 中，任务<5，7>先于任务<5，8>输入，而作为关键活动输出时则次序相反。

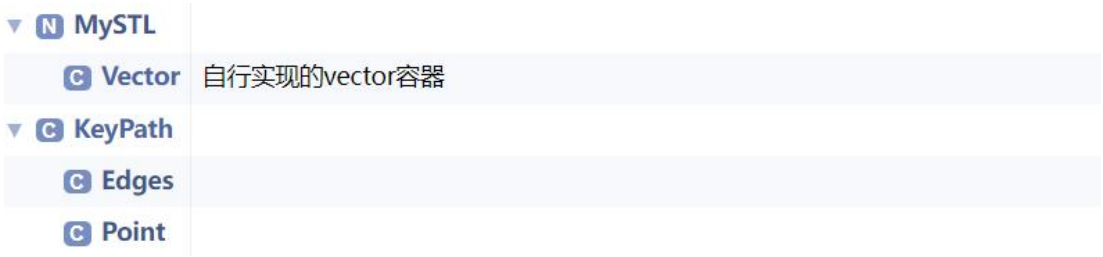
测试用例

序号	输入	输出	说明
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	17 1 –>2 2 –>4 4 –>6 6 –>7	简单情况测试

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	18 1 –>2 2 –>5 5 –>8 5 –>7 7 –>9 8 –>9	一般情况测试，单个起点和单个终点
3	4 5 1 2 4 2 3 5 3 4 6 4 2 3 4 1 2	0	不可行的方案测试

项目设计

数据结构设计



在命名空间 MySTL 中有自行实现的配有迭代器的 Vector 类；在 KeyPath 类中定义了 Edges 和 Point 两个嵌套类，仅用于类内使用。

KeyPath 类

在此类中**仅有一个 public 函数**供用户使用。其余均为类内实现。

```cpp
int main()
{
    KeyPath solution;
    solution.carry();
    system("pause");
    return 0;
}
```

类内储存边的结构为邻接矩阵，使用 static const 成员变量来限定邻接矩阵的大小。Private data member 中的 Vector<Edges>是按照输入顺序储存边。

Classes

struct **Edges**

struct **Point**

Public Member Functions

void **carry** ()

Public Attributes

struct **KeyPath::Point** **Points** [**MAXN**]

Static Public Attributes

static const int **MAXN** = 100

static const int **INF** = 0x3f3f3f3f

Private Member Functions

void **InitGraph** ()

void **InitPoints** ()

void **InsertActivities** ()

int **EarlytimeCheck** ()

void **LasttimeCheck** ()

Private Attributes

int **checkpoints**
顶点数 More...

int **activities**
边数 More...

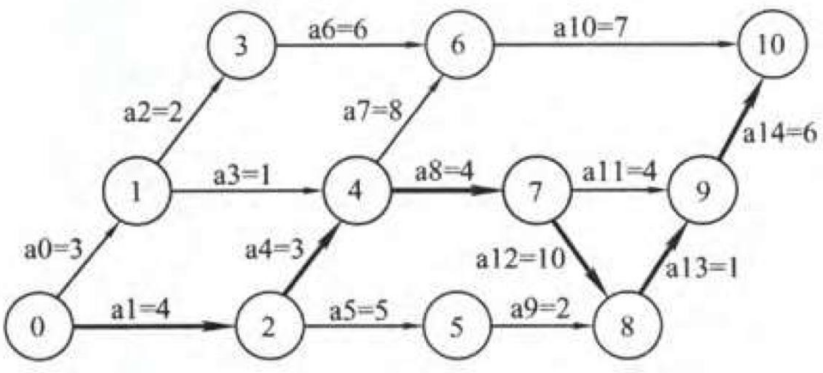
int **G** [**MAXN**][**MAXN**]
邻接矩阵 More...

int **totaltime** = 0
总任务时间 More...

Vector< **Edges** > **inputEdges**
存按输入顺序的边 More...

解题算法解题思路及关键代码

本题本质为 AOE 网。用例子来解释：



概念	说明	举例
边	边表示活动，且有权值，权值代表活动的持续时间	【边a4】代表活动，权重3，表示该活动要持续3天
顶点	顶点代表事件，事件是图中新活动开始、旧活动结束的标志	【顶点1】代表事件，是活动a2、a3的开始，是活动a0的结束
源点	入度为0的顶点，即工程的开始	顶点0
汇点	出度为0的顶点，即工程的结束	顶点10

事件k的最早发生时间ve(k)	ve即vertex earliest 【图中表现为】源点到顶点k的最长路径	若事件4要发生，事件4的上游0、1、2都要先发生 【即】事件4的最早发生时间ve(4)=0到4的最长路径=7
活动z的最早发生时间ee(z)	ee即edges earliest ve是说顶点的最早发生时间，ee是说边的最早发生时间	【解释】a7、a8的最早发生时间，即是事件4的最早发生时间，如ee(a7)=ee(a8)=ve(4)=7 【算法】`ee(z)=ve(k)` 顶点k是边z的头顶点 【例】边a3的头顶点是1：`ee(a3)=ve(1)=3`
关键路径	从源点到汇点，图中**最长的路径**称为关键路径 【现实意义】整个工期所完成的最短时间	【即0->10的最长路径】`a1->a4->a8->a12->a13->a14` 【关键路径的时间即为此工程完成的最短时间】`a1+a4+a8+a12+a13+a14=28`
关键活动	关键路径上的活动 【现实意义】若想要最短时间完成这项工程，这些活动的上游完成后，不能休息，要立刻开始！	`a1、a4、a8、a12、a13、a14`
事件k的最迟发生时间vl(k)	vl即vertex latest 在不推迟工期的前提下，事件k最迟发生时间	1. 根据关键路径得到工程结束的最短时间为28天 2. 事件7距离完成需要17天（7-10有两条路，取时间大的：①7->9->10的时间要10天，②7->8->9->10要17天） 3. 那么，事件7最迟要在第11天开始！不然28天就完不成了！ 【即】`vl(7)=28-17=11`
活动的最迟发生时间el(z)	el即edges latest 在不推迟工期的前提下，活动z最迟发生的时间	【解释】a8的最迟发生时间=vl(7)-a8的权重=vl(7)-4=7 【算法】`el(z)=vl(k)-z` 顶点k是边z的尾顶点 【例】边a14的尾顶点是10：`el(a14)=vl(10)-a14=28-6=22`

求关键路径，关键活动

	关键活动	关键路径	整个工程完成的最短时间
说明	ee=el的活动即为关键活动	关键活动构成的路径	关键活动的权重之和
解释	ee=el代表活动不能够推迟的，若该活动推迟了，工程就不能在工期内完成	关键路径即图中的最长路径	图中的所有活动都要完成，中间不停歇，完成整个工程的最短时间就是图中最长的路径，即关键路径
例子	`a1 a4 a8 a12 a13 a14`	`0->2->4->7->8->9->10`	整个工程完成的最短时间：`a1+a4+a8+a12+a13+a14=28`

代码如下：

求最早发生时间（同时使用队列求好了拓扑序列）

```
int i, Queue[MAXN], top = 0, rear = 0, point, maxtime = 0;
        for (i = 1; i <= checkpoints; i++)
        {
            if (!Points[i].indegree)
            {
                Points[i].indegree--;
                Queue[++top] = i;
            }
        }
        while (top != rear)
        {
            point = Queue[++rear];
            for (i = 1; i <= checkpoints; i++)
            {
                if (G[point][i])
                {
                    Points[i].indegree--;
                    if (!Points[i].indegree) Queue[++top] = i;
                    if (Points[i].earlytime < Points[point].earlytime + G[point][i])
                    {
                        Points[i].earlytime = Points[point].earlytime + G[point][i];
                        if (Points[i].earlytime > maxtime)
                            maxtime = Points[i].earlytime;
                    }
                }
            }
        }
```

求最迟时间：

```
        while (top != rear)
        {
            point = Queue[++rear];
            for (i = 1; i <= checkpoints; i++)
            {
                if (G[i][point])
                {
                    Points[i].outdegree--;
                    if (!Points[i].outdegree)  Queue[++top] = i;

                    if (Points[point].lasttime - G[i][point] < Points[i].lasttime)
                        Points[i].lasttime = Points[point].lasttime - G[i][point];
                }
            }
        }
```

最后要求当起点相同的时候输出顺序是逆序，处理方法即为用 Vector<Edges> 先储存按照输入顺序的所有边，然后输出时，先将起点从小打大输出，当有相同起点时，使用 vector 的类似 reverse iterator 来逆序输出终点。

```
for (auto it = inputEdges.end(); it >= inputEdges.begin(); it--)
                for (auto it2 = temp.begin(); it2 != temp.end(); it2++)
                    if (it->a == it2->a && it->b == it2->b)
                        printf("%d->%d\n", it->a, it->b);
```

设计亮点

代码注释规范

采用了 doxygen 注释规范，对类、函数等有简要说明，命名采用驼峰命名法，类内的各个声明规范，方便本人回顾之前写过的代码和 code reviewer 查看，使 API 规范，帮助这个开发流程高效、规范地进行。

```
    struct Point
    {
        int earlytime;  ///< 最早时间
        int lasttime;   ///< 最晚时间
        int indegree;   ///< 入度
        int outdegree;  ///< 出度
    } Points[MAXN];

private:
    int checkpoints;  ///< 顶点数
    int activities;   ///< 边数
    int G[MAXN][MAXN];///< 邻接矩阵
    int totaltime = 0;///< 总任务时间

    Vector<Edges> inputEdges; ///< 存按输入顺序的边
```

项目测试

```
为类似OJ便于测试数据，本题无输入提示，输入方式如下：
第一行输入任务交接点n和任务数量m
第2行至第m + 1行每行为3个正整数，分别是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
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

17
1->2
2->4
4->6
6->7
请按任意键继续. . .
```

```
为类似OJ便于测试数据，本题无输入提示，输入方式如下：
第一行输入任务交接点n和任务数量m
第2行至第m + 1行每行为3个正整数，分别是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
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

18
1->2
2->5
5->8
5->7
7->9
8->9
请按任意键继续. . .
```

```
为类似OJ便于测试数据，本题无输入提示，输入方式如下：
第一行输入任务交接点n和任务数量m
第2行至第m + 1行每行为3个正整数，分别是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
4 5
1 2 4
2 3 5
3 4 6
4 2 3
4 1 2
0
请按任意键继续. . .
```

Linux 下测试运行（Ubuntu 系统）

```
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework9$ vim makefile
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework9$ ls
h9.cpp  makefile
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework9$ vim Vector.hpp
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework9$ ls
Vector.hpp  h9.cpp  makefile
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework9$ make
g++    -c -o h9.o h9.cpp
g++ -o run h9.o
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework9$ ./run
为类似OJ便于测试数据，本题无输入提示，输入方式如下：
第一行输入任务交接点n和任务数量m
第2行至第m + 1行每行为3个正整数，分别是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
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

18
1->2
2->5
5->8
5->7
7->9
8->9
```