**数据结构课程设计**

**——小组设计报告**

**组 号:\_\_\_\_\_\_\_\_\_\_\_\_\_\_01\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**指导教师: \_\_\_\_\_\_\_\_ \_冯向阳\_\_\_\_\_\_\_\_\_\_\_\_\_**

**组成员及贡献度**

|  |  |  |
| --- | --- | --- |
| 姓名 | 学号 | 贡献度 |
| 周立成 | 221310332 | 1 |
| 吕嘉乐 | 221310403 | 1 |
| 刘子睿 | 221310304 | 1 |
| 刘璇 | 221310402 | 1 |
| 汪淇 | 221310405 | 1 |

日期: 2023年6月20日至2022年7月07日

目录：

[1.课程设计目的 5](#_Toc139528756)

[2.任务完成情况 5](#_Toc139528757)

[3. 设计报告 6](#_Toc139528758)

[**3.1操作系统任务调度问题 6**](#_Toc139528759)

[**3.1.1 题目以及要求 6**](#_Toc139528760)

[**3.1.2概要设计 6**](#_Toc139528761)

[**3.1.3算法分析 6**](#_Toc139528762)

[**3.1.4使用说明 7**](#_Toc139528763)

[**3.1.5 测试结果以及分析 8**](#_Toc139528764)

[**3.2 找出从指定结点出发且长度为m的所有简单路径 11**](#_Toc139528765)

[**3.2.1 题目及要求 11**](#_Toc139528766)

[**3.2.2 概要设计 11**](#_Toc139528767)

[**3.2.3 算法分析 12**](#_Toc139528768)

[**3.2.4 使用说明 12**](#_Toc139528769)

[**3.2.5 测试结果与分析 13**](#_Toc139528770)

[**3.3 Prim算法的设计 19**](#_Toc139528771)

[**3.3.1 题目以及要求 19**](#_Toc139528772)

[**3.3.2 概要设计 20**](#_Toc139528773)

[**3.3.3 算法分析 20**](#_Toc139528774)

[**3.3.4 使用说明 21**](#_Toc139528775)

[**3.3.5测试结果以及分析 22**](#_Toc139528776)

[**3.4 Kruskal算法的设计 33**](#_Toc139528777)

[**3.4.1 题目以及要求 33**](#_Toc139528778)

[**3.4.2概要设计 33**](#_Toc139528779)

[**3.4.3算法分析 34**](#_Toc139528780)

[**3.4.4使用说明 35**](#_Toc139528781)

[**3.4.5 测试结果以及分析 36**](#_Toc139528782)

[**3.5 Dijkstra算法的设计 43**](#_Toc139528783)

[**3.5.1 题目以及要求 43**](#_Toc139528784)

[**3.5.2概要设计 44**](#_Toc139528785)

[**3.5.3算法分析 44**](#_Toc139528786)

[**3.5.4使用说明 46**](#_Toc139528787)

[**3.5.5 测试结果以及分析 47**](#_Toc139528788)

[**3.6获取AOE网的关键路径 58**](#_Toc139528789)

[**3.6.1题目以及要求 58**](#_Toc139528790)

[**3.6.2概要设计 58**](#_Toc139528791)

[**3.6.3算法分析 58**](#_Toc139528792)

[**3.6.4使用说明 62**](#_Toc139528793)

[**3.6.5测试结果以及分析 62**](#_Toc139528794)

[**3.7 判断给定的二叉树是否是二叉排序树 79**](#_Toc139528795)

[**3.7.1 题目以及要求 79**](#_Toc139528796)

[**3.7.2概要设计 79**](#_Toc139528797)

[**3.7.3算法分析 80**](#_Toc139528798)

[**3.7.4使用说明 80**](#_Toc139528799)

[**3.7.5 测试结果以及分析 80**](#_Toc139528800)

[**3.8 AVL树的判断 82**](#_Toc139528801)

[**3.8.1 题目以及要求 82**](#_Toc139528802)

[**3.8.2概要设计 82**](#_Toc139528803)

[**3.8.3算法分析 82**](#_Toc139528804)

[**3.8.4使用说明 83**](#_Toc139528805)

[**3.8.5 测试结果以及分析 84**](#_Toc139528806)

[**3.9二叉树的重建 88**](#_Toc139528807)

[**3.9.1 题目以及要求 88**](#_Toc139528808)

[**3.9.2 概要设计 89**](#_Toc139528809)

[**3.9.3 算法分析 89**](#_Toc139528810)

[**3.9.4 使用说明 90**](#_Toc139528811)

[**3.9.5测试结果以及分析 90**](#_Toc139528812)

[**3.10第 K 个最小的素数分数 93**](#_Toc139528813)

[**3.10.1题目及描述 93**](#_Toc139528814)

[**3.10.2概要设计 93**](#_Toc139528815)

[**3.10.3算法分析 93**](#_Toc139528816)

[**3.10.4使用说明 93**](#_Toc139528817)

[**3.10.5测试结果及分析 94**](#_Toc139528818)

[**3.11 LRU缓存 96**](#_Toc139528819)

[**3.11.1 题目以及要求 96**](#_Toc139528820)

[**3.11.2 概要设计 96**](#_Toc139528821)

[**3.11.3算法分析 96**](#_Toc139528822)

[**3.11.4使用说明 97**](#_Toc139528823)

[**3.11.5测试结果及分析 98**](#_Toc139528824)

[**3.12 设计推特 104**](#_Toc139528825)

[**3.12.1 题目及其要求 104**](#_Toc139528826)

[**3.12.2 概要设计 104**](#_Toc139528827)

[**3.12.3 算法分析 105**](#_Toc139528828)

[**3.12.4 使用说明 106**](#_Toc139528829)

[**3.12.5 测试结果及分析 107**](#_Toc139528830)

[**3.13 单线程 CPU 114**](#_Toc139528831)

[**3.13.1题目及其要求 114**](#_Toc139528832)

[**3.13.2概要设计 114**](#_Toc139528833)

[**3.13.3算法分析 114**](#_Toc139528834)

[**3.13.4使用说明 116**](#_Toc139528835)

[**3.13.5测试结果及分析 116**](#_Toc139528836)

[**3.14我的日程安排表 I 167**](#_Toc139528837)

[**3.14.1题目及其要求 167**](#_Toc139528838)

[**3.14.2概要设计 167**](#_Toc139528839)

[**3.14.3算法分析 167**](#_Toc139528840)

[**3.14.4使用说明 168**](#_Toc139528841)

[**3.14.5测试结果及分析 168**](#_Toc139528842)

[**3.15设计电影租借系统 175**](#_Toc139528843)

[**3.15.1题目及要求 175**](#_Toc139528844)

[**3.15.2概要设计 175**](#_Toc139528845)

[**3.15.3算法分析 176**](#_Toc139528846)

[**3.15.4使用说明 177**](#_Toc139528847)

[**3.15.5测试结果及分析 178**](#_Toc139528848)

## 1.课程设计目的

1. 学习获取知识的方法；
2. 提高发现问题、分析问题和解决实际问题的能力；
3. 加强创新意识和创新精神；
4. 加强团队的分工与合作；
5. 掌握面向实际背景思考问题的方法。

## 2.任务完成情况

|  |  |  |
| --- | --- | --- |
| 任务序号 | 任务名称 | 完成情况 |
| 1 | 操作系统任务调度问题 | AC |
| 2 | 找出从指定结点出发且长度为*m*的所有简单路径 | AC |
| 3 | Prim算法的设计 | AC |
| 4 | Kruskal算法的设计 | AC |
| 5 | Dijkstra算法的设计 | AC |
| 6 | 获取AOE网的关键路径 | AC |
| 7 | 判断给定的二叉树是否是二叉排序树 | AC |
| 8 | AVL树的判断 | AC |
| 9 | 二叉树的重建 | AC |
| 10 | 第 K 个最小的素数分数 | AC |
| 11 | LRU缓存 | AC |
| 12 | 设计推特 | AC |
| 13 | 单线程 CPU | AC |
| 14 | 我的日程安排表 I | AC |
| 15 | 设计电影租借系统 | AC |

## 3. 设计报告

### 3.1操作系统任务调度问题

#### 3.1.1 题目以及要求

**题目描述:**

目的：设计并实现一个算法*Schedule*，完成简单的操作系统任务调度问题。

描述：操作系统任务分为系统

任务和用户任务两种。其中，系统任务的优先级，用户任务的优先级且。优先级大于的为非法任务，应予以剔除。现有一任务序列*task*（顺序存储），任务数为*n*。每个*task*的元素值表示任务的优先级，数值越小，优先级越高。算法*scheduler*实现如下功能，将*task*中的任务按照系统任务、用户任务依次存放到队列 *system\_task*和 *user\_task*中（队列中元素的值是任务在顺序表*task*中对应的下标），并且优先级高的任务排在前面。优先级相同的任务按照在*task*中的相对顺序入队排列（即*task*序列中在先的任务排在前面）。

要求：顺序表使用*vector*或数组，队列使用本学期设计好的顺序队列*ADT*

参考函数原形：

*template<class ElemType>*

*void Schedule(vector<ElemType> &A, SqQueue<int> &system\_task, SqQueue<int> &user\_task);*

**输出说明:**

第一行：任务序列*task*的遍历结果，以","分隔

格式： (序号,优先级) ...

空行

第二行：系统任务队列遍历结果，以空格分隔。如系统任务队列为空，输出 *none*

第三行：用户任务队列遍历结果，以空格分隔。如用户任务队列为空，输出 *none*

#### 3.1.2概要设计

设计结构体*node*用来存放下标和数值,使用*bool operator <* 重载其小于运算符以数值为第一比较参数,编号为第二比较参数,将输入数据按条件放入两个*set*容器中使其自动排序.最后输出.

#### 3.1.3算法分析

*算法说明：Set*容器会自动为放入的元素排序(由于每个*node*的*index*一定不同,故可以不使用*multiset*)

以样例[0 30 155 1 80 300 155 170 40 30 99]进行分析

在输出第一行内容后,要先将非法的任务(优先级>255)删除:

图示

描述已自动生成

其次按照任务的优先级分类,并以数值为第一比较参数,序号为第二比较参数分别放入不同的集合中;

图表, 气泡图

描述已自动生成

*算法时间复杂度:*仅插入操作占用时间,*set*内部使用红黑树实现,单次插入时间复杂度为在这里实现了*n*次插入操作故总体时间复杂度为.

#### 3.1.4使用说明

**输入说明:**

第一行：任务序列*task*中各个任务的优先级（任务优先级之间以空格分隔）

程序名称为*dhu\_teams\_1.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图4.1

文本

描述已自动生成

图4-1:运行截图

在输入完数据后,需要使用键盘的*Ctrl+D*(程序内)/*Ctrl+Z*(外部*exe*)即可显示结果.

#### 3.1.5 测试结果以及分析

* 第一组数据:所有数据顺序排列,且有系统任务队列,用户任务队列,无非法值

IN:

0 1 2 3 4 5 51 52 53 54 55 56

OUT:

(0,0),(1,1),(2,2),(3,3),(4,4),(5,5),(6,51),(7,52),(8,53),(9,54),(10,55),(11,56)

0 1 2 3 4 5

6 7 8 9 10 11

* 第二组数据:数据随机排列,但保证系统任务队列和用户任务队列均有,但有非法值

IN:

0 30 155 1 80 300 155 170 40 30 99

OUT:

(0,0),(1,30),(2,155),(3,1),(4,80),(5,300),(6,155),(7,170),(8,40),(9,30),(10,99)

0 3 1 9 8

4 10 2 6 7

* 第三组数据,数据随机排列,保证有系统任务队列,但无用户任务队列,无非法值

IN:

0 1 2 3 4 7 8 10 9 41 20 23

OUT:

(0,0),(1,1),(2,2),(3,3),(4,4),(5,7),(6,8),(7,10),(8,9),(9,41),(10,20),(11,23)

0 1 2 3 4 5 6 8 7 10 11 9

none

* 第四组数据,数据随机排列,保证有用户任务队列,但无系统任务队列,有非法值

IN:

52 53 56 58 99 100 68 70 300 400 89 88 87

OUT:

(0,52),(1,53),(2,56),(3,58),(4,99),(5,100),(6,68),(7,70),(8,300),(9,400),(10,89),(11,88),(12,87)

none

0 1 2 3 6 7 12 11 10 4 5

* 第五组数据,全是非法值,检测鲁棒性

IN:

500 600 700 800 900 1000 400

OUT:

(0,500),(1,600),(2,700),(3,800),(4,900),(5,1000),(6,400)

none

none

* 第六组数据,大随机数检验,测试时间复杂度

IN:

884 221 993 658 59 513 941 31 974 159 499 493 120 719 277 65 121 112 91 950 193 81 178 53 729 932 350 521 145 39 506 379 709 168 741 81 254 728 76 12 597 55 541 864 268 551 23 352 538 336 683 111 514 949 936 660 419 64 80 846 309 960 323 374 774 721 633 439 344 700 481 819 485 497 188 354 84 409 575 77 766 808 383 339 19 554 35 610 608 1 194 642 463 802 316 91 129 100 661 752 668 398 355 99 623 184 676 954 913 40 773 997 935 804 395 234 188 986 602 313 184 775 75 203 193 359 219 486 291 292

OUT:

(0,884),(1,221),(2,993),(3,658),(4,59),(5,513),(6,941),(7,31),(8,974),(9,159),(10,499),(11,493),(12,120),(13,719),(14,277),(15,65),(16,121),(17,112),(18,91),(19,950),(20,193),(21,81),(22,178),(23,53),(24,729),(25,932),(26,350),(27,521),(28,145),(29,39),(30,506),(31,379),(32,709),(33,168),(34,741),(35,81),(36,254),(37,728),(38,76),(39,12),(40,597),(41,55),(42,541),(43,864),(44,268),(45,551),(46,23),(47,352),(48,538),(49,336),(50,683),(51,111),(52,514),(53,949),(54,936),(55,660),(56,419),(57,64),(58,80),(59,846),(60,309),(61,960),(62,323),(63,374),(64,774),(65,721),(66,633),(67,439),(68,344),(69,700),(70,481),(71,819),(72,485),(73,497),(74,188),(75,354),(76,84),(77,409),(78,575),(79,77),(80,766),(81,808),(82,383),(83,339),(84,19),(85,554),(86,35),(87,610),(88,608),(89,1),(90,194),(91,642),(92,463),(93,802),(94,316),(95,91),(96,129),(97,100),(98,661),(99,752),(100,668),(101,398),(102,355),(103,99),(104,623),(105,184),(106,676),(107,954),(108,913),(109,40),(110,773),(111,997),(112,935),(113,804),(114,395),(115,234),(116,188),(117,986),(118,602),(119,313),(120,184),(121,775),(122,75),(123,203),(124,193),(125,359),(126,219),(127,486),(128,291),(129,292)

89 39 84 46 7 86 29 109

23 41 4 57 15 122 38 79 58 21 35 76 18 95 103 97 51 17 12 16 96 28 9 33 22 105 120 74 116 20 124 90 123 126 1 115 36

**3.2 找出从指定结点出发且长度为m的所有简单路径**

**3.2.1 题目及要求**

题目要求：

在使用图的邻接表*ADT*的基础上，设计一个算法，按照深度优先搜索的思想找出从指定结点出发且长度为*m*的所有简单路径。并将此算法加入到邻接表*ADT*中，在邻接表*ADT*中提供一个公有的成员函数*FindPath(start, m)*。

提示：

（1）这个问题相当于从指定结点开始深度优先遍历，而且遍历的深度正好为*m*。为此，在遍历时需要记住遍历的深度，当深度达到*m*时，就不需要递归了。此时需要输出这条路径，因此在遍历的过程中还需要记住整条路径。

（2）由于深度优先遍历是用递归实现的，所以*FindPath*函数最好也设计两个。一个是共有的*FindPath*函数，供用户使用（外壳）；另一个是私有的*FindPath*函数，实现递归的遍历。公有的*FingPath*函数调用私有的*FindPath*函数找出这些路径。

参考函数原型：

（1）找出从指定结点出发且长度为*m*的所有简单路径（外壳部分）

*template<class TypeOfVer, class TypeOfEdge>*

*void adjlist\_graph<TypeOfVer, TypeOfEdge>::FindPath(int start, int m);*

（2）找出从指定结点出发且长度为*m*的所有简单路径（递归部分）

*template<class TypeOfVer, class TypeOfEdge>*

*void adjlist\_graph<TypeOfVer, TypeOfEdge>::FindPath(int start, int m, int &count, int top, int visited[], int stack[]);*

输出说明：

第一行：顶点集

第二行：邻接表

空行

如无符合要求的路径：输出 0

否则：路径输出（一条路径占一行）

路径数目

**3.2.2 概要设计**

设计一个*adjlist\_graph*类，其中抽象数据类型包括：

数据对象：V为顶点数据类型，E为边的数据类型。

基本操作：

*Adjlist\_graph（）*构造函数，对图进行初始化。

基于*adjlist\_graph*类，设计*make*\_*none\_weight\_adjlist\_graph（）*函数对图进行构造。

*print\_vers( )*输出顶点集。

*printList\_no\_weight( )*输出邻接表。*find\_m\_path(vector<****int****>vis,vector<V>paths,vector<vector<V>>&result,****int*** *n,****int*** *m,****int*** *s)* 找出从指定结点出发且长度为*m*的所有简单路径（递归部分）。

*find\_path（****int*** *start,****int*** *m）*找出从指定结点出发且长度为*m*的所有简单路径（外壳部分）及数目并输出。

**3.2.3 算法分析**

算法说明：递归的*find\_m\_path*函数首先将起点放入这条路径，并标记这个结点已被访问，然后判断路径长度是否是*m*。如果长度已达到*m*，则输出这条路径，并将最后一个结点从路径上删除，返回上一层调用，检查是否还有其它的途径；否则逐个检查起点的后继结点。如果该后继结点没有被访问过，则对该结点调用递归的*find\_m\_path*函数继续寻找。在所有的后继都检查后，表示这条路径处理完毕。将起始结点从这条路径上删除，返回上一层调用。直到所有结点都被访问过。

需要注意：

1）要找的是长度为*m*的简单路径，因此路径上不能有相同的结点，于是定义了一个数组*visited*记录结点是否在路径上。

2）当路径长度等于m时要输出这条路径，于是定义了一个数组stack保存这条路径。每访问一个结点，都要把结点记录在stack中。

3）递归的*find\_m\_path*函数有6个参数。第1个参数是遍历的起点的序号；第2个参数是要求的路径长度；第3个参数是符合要求的路径数目；第4个参数是当前路径中的结点数，当前路径的长度是结点数减1；第5个参数是*visited*数组，记录结点是否在路径上；第6个参数是一个用于记录路径上结点序号的数组，作用和栈类似。

时间复杂度分析：因为从指定顶点出发，最坏的情况就是把每条边都访问一遍，故算法的时间复杂度为*O(E)*。

**3.2.4 使用说明**

输入说明：

第一行：图的类型

第二行：结点数

第三行：结点集

第四行：边数

第五行：边集

第六行：起点start

第七行：路径长度m

|  |
| --- |
|  |
|  |  |

程序名称为*FindPath.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图

图：运行截图

在输入完数据后,需要使用键盘的*Ctrl+D*(程序内)/*Ctrl+Z*(外部*exe*)即可显示结果。

**3.2.5 测试结果与分析**

|  |
| --- |
|  |
|  |  |

1）没有边的

输入：

DG

6

1 2 3 4 5 6

0

2

4

输出：

1 2 3 4 5 6

1

2

3

4

5

6

0

|  |
| --- |
|  |
|  |  |

2）成环的

输入：

UDG

5

A B C D E

5

0 1

1 2

2 3

3 4

4 0

1

3

输出：

A B C D E

A->4->1

B->2->0

C->3->1

D->4->2

E->0->3

B->C->D->E

B->A->E->D

2

|  |
| --- |
|  |
|  |  |

3）完全图

输入：

DG

5

V0 V1 V2 V3 V4

10

0 1

0 2

0 3

0 4

1 2

1 3

1 4

2 3

2 4

3 4

0

2

输出：

V0 V1 V2 V3 V4

V0->4->3->2->1

V1->4->3->2

V2->4->3

V3->4

V4

V0->V3->V4

V0->V2->V4

V0->V2->V3

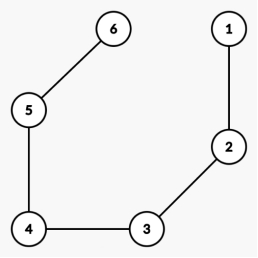
V0->V1->V4

V0->V1->V3

V0->V1->V2

6

4）一条链



输入：

UDG

6

1 2 3 4 5 6

5

0 1

1 2

2 3

3 4

4 5

2

1

输出：

1 2 3 4 5 6

1->1

2->2->0

3->3->1

4->4->2

5->5->3

6->4

3->4

3->2

2

|  |
| --- |
|  |
|  |  |

5）不存在的路径

输入：

DG

4

1 2 3 4

6

0 1

0 3

2 3

1 2

3 1

2 0

1

7

输出：

1 2 3 4

1->3->1

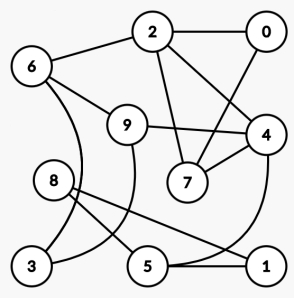
2->2

3->0->3

4->1

0

6）比较大的数据



输入：

DG

10

1 2 3 4 5 6 7 8 9 10

15

0 2

3 6

4 2

0 7

9 6

8 1

7 4

6 3

4 5

5 8

7 2

2 6

4 9

9 3

5 1

9

2

输出:

1 2 3 4 5 6 7 8 9 10

1->7->2

2

3->6

4->6

5->9->5->2

6->1->8

7->3

8->2->4

9->1

10->3->6

10->4->7

10->7->4

2

### 3.3 Prim算法的设计

#### 3.3.1 题目以及要求

**题目描述：**

在使用图的邻接矩阵*ADT*的基础上，设计能指定起始结点*u*的*Prim*算法，用以求无向网的最小生成树，并以文本形式输出生成树中各条边以及它们的权值。将此算法加入到邻接矩阵*ADT*中，在邻接矩阵*ADT*中提供一个公有的成员函数*Prim(u, adjvex, lowcost)*。

提示：

（1）*Prim*算法的基本思想是从结点的角度出发。初始时，生成树中一个结点也没有，然后将一个个结点加入生成树，直到所有的结点都被加入，生成树就形成了。具体来讲，*Prim*算法由以下几个步骤组成：

1）初始时，生成树的结点集*U*为空；

2）随意选择一个结点，加入结点集*V*，然后重复下列工作，直到*U=V*；

3）把*(u,v)*加入生成树的边集，*v*加入到*U*。

（2）在*Prim*算法的执行过程中，需要保存哪些结点在*U*中，哪些结点不在*U*中的信息。这可以用一个布尔型的一维数组*flag*来保存。如果结点i已在生成树中，则*flag[i]*的值为*true*，否则为*false*。

（3）在*Prim*算法的执行过程中，还需要保存*U*中的结点到*V-U*中结点的权值最小的边，这可以用2个一维数组*lowcost*和*adjvex*来记录。*lowcost[i]*表示*U*中的结点到结点*i*的所有边中的最小权值。*adjvex[i]*表示从*U*中的哪一个结点出发到结点*i*的权值是最小的。

参考函数原型：

*template<class TypeOfVer, class TypeOfEdge>*

*bool adjmatrix\_graph<TypeOfVer, TypeOfEdge>::Prim(int u, int adjvex[], TypeOfEdge lowcost[]);*

**输出说明：**

第一行：图的类型

第二行：顶点集

空行

第三行：邻接矩阵（列与列之间用格式控制符'\t'分隔）

第四行：各条边以及它们的权值，输出格式为：

(c,b,5),(d,c,3),(e,d,8),(a,e,14),(d,f,21),(e,g,16)

其中，每个括号内的信息为一条边，格式为（邻接点, 自己, 权值），边的输出顺序为按照（自己这个）点的编号顺序输出。“自己这个”不包括起始点，比如范例中输出的边不包括起始点a，也就是没有(\*,a,\*)。

#### 3.3.2 概要设计

设计一个*adjmartix\_graph*类，其中抽象数据类型包括：

数据对象：*V*为顶点数据类型，E为边的数据类型。

基本操作：

*adjmartix\_graph（）*构造函数，对图进行初始化。

*Prim（u）*普里姆算法，从输入的u结点开始寻找权值最小边并纳入最小生成树。

*print\_edge()*输出图的邻接矩阵*。*

*print\_vers()*输出结点集*。*

*get\_graph\_type()*得到图的类型*。*

基于*adjmartix\_graph*类，设计*make\_adjmartix\_graph（）*函数对图进行构造。

#### 3.3.3 算法分析

*算法说明：*基于图类设计抽象数据类型*Prim（）*函数，从输入的一个节点出发，依次访问没有访问过的节点，通过比较权值将权值最小的边纳入最小生成树，直到所有结点都被访问过。

以下图为例:

图示

描述已自动生成

图片包含 图示

描述已自动生成

该算法需要注意几个难点:

1.如何区分没访问过的节点与访问过的节点:设置*vis*数组,用来标记该结点是否被访问过——用于遍历邻接矩阵时区别是否是访问过的节点,设置*has\_visited*数组存放已经访问顶点在节点集合中的位序——用来判断循环终止条件,即当*has\_visited*数组的大小等于顶点集数组大小时候,Prim算法的循环将终止;

2.如何判断每次要加入的顶点:只需要遍历*has\_visited*数组中的每一个已访问顶点的位序,用他们取探索未访问的顶点即可;

3.如何存放结果——小组使用**map<V,pair<V,E>>*res***存放结果,其三个参数分别为自身节点,邻接顶点,权值,最后使用迭代器输出

4.如何控制输出格式——有时候会遇到比如*map*中已经存在一个键值d对应(e,d,8),但是之后又有同样的键值*d*对应着(f,d,21),尽管*fd*也是我们要找的边,但是我们需要将键值从d变为*f*,利用*map*中的*count*函数可以判断是否已经存在该键值;

5.*map*模板类的输出——在使用模板类*stl*迭代器输出时候如本题应该在前加入*typename*关键字,如本题中是用**for(typename std::map<V,pair<V,E> >::iterator it =res.begin();it!=res.end();it++)**进行输出的.

*算法时间复杂度：*外层循环需要寻找条边，故外层循环时间复杂度为**，循环内，选择权值最短的边，故算法的时间复杂度为*。*

#### 3.3.4 使用说明

**输入说明：**

第一行：图的类型

第二行：结点数

第三行：结点集

第四行：无边标记

第五行：边数

第六行：边集

第七行：权集

第八行：起始结点序号u

程序名称为*Prim.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图4.3

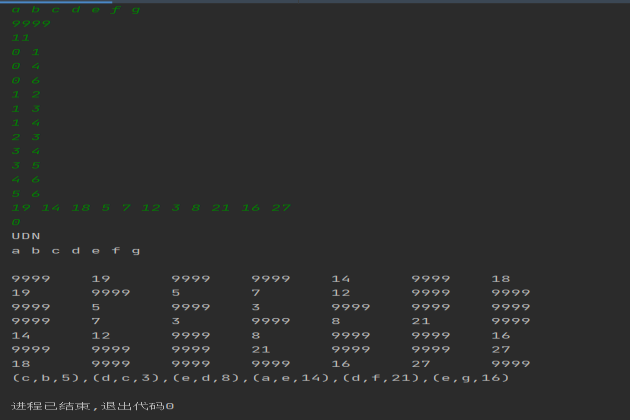


图4.3：运行截图

在输入完数据后,需要使用键盘的*Ctrl+D*(程序内)/*Ctrl+Z*(外部*exe*)即可显示结果。

#### 3.3.5测试结果以及分析

* 第一组数据:已经自成一棵树,起始点序号为原点,简图如下:

图表

描述已自动生成

IN:

UDN

7

V1 V2 V3 V4 V5 V6 V7

9999

6

0 1

0 2

1 3

2 4

3 5

4 6

1 3 2 4 5 6

0

OUT:

UDN

V1 V2 V3 V4 V5 V6 V7

9999 1 3 9999 9999 9999 9999

1 9999 9999 2 9999 9999 9999

3 9999 9999 9999 4 9999 9999

9999 2 9999 9999 9999 5 9999

9999 9999 4 9999 9999 9999 6

9999 9999 9999 5 9999 9999 9999

9999 9999 9999 9999 6 9999 9999

(V3,V1,3),(V1,V2,1),(V5,V3,4),(V2,V4,2),(V7,V5,6),(V4,V6,5)

* 第二组数据:成一个环,起始点不为序号原点,简图如下

图表, 形状, 雷达图, 正方形

描述已自动生成

IN:

UDN

7

V1 V2 V3 V4 V5 V6 V7

9999

7

0 1

1 2

2 3

3 4

4 5

5 6

6 0

1 3 2 4 5 6 8

2

OUT:

UDN

V1 V2 V3 V4 V5 V6 V7

9999 1 9999 9999 9999 9999 8

1 9999 3 9999 9999 9999 9999

9999 3 9999 2 9999 9999 9999

9999 9999 2 9999 4 9999 9999

9999 9999 9999 4 9999 5 9999

9999 9999 9999 9999 5 9999 6

8 9999 9999 9999 9999 6 9999

(V1,V2,1),(V2,V3,3),(V3,V4,2),(V4,V5,4),(V5,V6,5),(V6,V7,6)

* 第三组数据,普通图,输入数据为*int*,输入量适中,起始点为源点,简图如下:

形状, 多边形

描述已自动生成

IN:

UDN

7

V1 V2 V3 V4 V5 V6 V7

9999

13

0 1

0 2

0 4

0 6

1 2

1 5

2 3

2 6

3 4

3 5

4 5

4 6

5 6

1 4 2 3 6 7 8 10 2 4 5 23 7

0

OUT:

UDN

V1 V2 V3 V4 V5 V6 V7

9999 1 4 9999 2 9999 3

1 9999 6 9999 9999 7 9999

4 6 9999 8 9999 9999 10

9999 9999 8 9999 2 4 9999

2 9999 9999 2 9999 5 23

9999 7 9999 4 5 9999 7

3 9999 10 9999 23 7 9999

(V5,V1,2),(V1,V2,1),(V1,V3,4),(V6,V4,4),(V4,V5,2),(V1,V7,3)

* 第四组数据:普通图,输入数据类型为*double*,为了测试程序的广泛性,起始点不为源点,简图如下:

形状, 多边形

描述已自动生成

IN:

UDN

7

V1 V2 V3 V4 V5 V6 V7

9999

13

0 1

0 2

0 4

0 6

1 2

1 5

2 3

2 6

3 4

3 5

4 5

4 6

5 6

1.1 4.2 2 3.5 6.9 7 8.1 10.2 2.1 4 5 23 7

0

OUT:

UDN

V1 V2 V3 V4 V5 V6 V7

9999 1.1 4.2 9999 2 9999 3.5

1.1 9999 6.9 9999 9999 7 9999

4.2 6.9 9999 8.1 9999 9999 10.2

9999 9999 8.1 9999 2.1 4 9999

2 9999 9999 2.1 9999 5 23

9999 7 9999 4 5 9999 7

3.5 9999 10.2 9999 23 7 9999

(V5,V1,2),(V1,V2,1.1),(V1,V3,4.2),(V6,V4,4),(V4,V5,2.1),(V1,V7,3.5)

* 第五组数据:图的类型不为”*UDN*”,直接不执行*prim*算法部分,测试鲁棒性

IN:

DN

7

V1 V2 V3 V4 V5 V6 V7

9999

13

0 1

0 2

0 4

0 6

1 2

1 5

2 3

2 6

3 4

3 5

4 5

4 6

5 6

1.1 4.2 2 3.5 6.9 7 8.1 10.2 2.1 4 5 23 7

0

OUT:

UDN

V1 V2 V3 V4 V5 V6 V7

9999 1.1 4.2 9999 2 9999 3.5

1.1 9999 6.9 9999 9999 7 9999

4.2 6.9 9999 8.1 9999 9999 10.2

9999 9999 8.1 9999 2.1 4 9999

2 9999 9999 2.1 9999 5 23

9999 7 9999 4 5 9999 7

3.5 9999 10.2 9999 23 7 9999

* 第六组数据,全是孤立的点,则也不应该执行*prim*算法,或者*prim*算法得到的结果为空,测试鲁棒性,简图如下:

图表, 散点图

描述已自动生成

IN:

UDN

7

V1 V2 V3 V4 V5 V6 V7

9999

0

0

OUT:

UDN

V1 V2 V3 V4 V5 V6 V7

9999 9999 9999 9999 9999 9999 9999

9999 9999 9999 9999 9999 9999 9999

9999 9999 9999 9999 9999 9999 9999

9999 9999 9999 9999 9999 9999 9999

9999 9999 9999 9999 9999 9999 9999

9999 9999 9999 9999 9999 9999 9999

9999 9999 9999 9999 9999 9999 9999

第七组数据:大数据压力测试680*KB*读入23*ms*运行成功,本样例考察时间复杂度是否良好,故数据类型全为*int*,且从源点开始,本题简图如下:

图示

描述已自动生成

IN:

UDN

20

V1 V2 V3 V4 V5 V6 V7 V8 V9 APPLE BANANA PEACH GRAPE PEOPLE FATHER ZLC LZR LX WQ LJL

9999

130

0 1

0 2

0 3

0 4

0 5

0 9

0 10

0 12

0 13

0 14

0 19

1 2

1 4

1 5

1 6

1 7

1 9

1 10

1 11

1 12

1 13

1 14

1 17

1 18

1 19

2 3

2 4

2 5

2 7

2 8

2 9

2 10

2 11

2 13

2 15

2 17

2 19

3 4

3 6

3 7

3 8

3 9

3 10

3 14

3 13

3 18

3 19

4 5

4 6

4 7

4 8

4 9

4 10

4 11

4 12

4 13

4 14

4 15

4 16

4 17

4 19

5 6

5 7

5 8

5 9

5 11

5 12

5 13

5 14

5 15

5 16

5 19

6 7

6 8

6 9

6 10

6 13

6 15

6 19

7 9

7 10

7 11

7 13

7 15

7 17

7 19

8 9

8 10

8 11

8 12

8 13

8 14

8 15

8 16

8 19

9 10

9 11

9 12

9 13

9 14

9 15

9 16

9 17

9 18

9 19

10 11

10 14

10 19

11 12

11 13

11 15

11 16

11 17

11 18

12 16

12 17

12 19

13 14

13 15

13 16

13 19

14 15

14 16

15 16

15 17

16 17

16 18

17 18

17 19

18 19

291 723 425 111 676 788 134 203 848 909 373 402 178 841 530 467 315 670 866 870 289 960 621 560 607 345 122 419 792 518 203 987 341 548 415 785 850 434 975 328 318 851 768 186 1 39 261 605 211 209 719 65 99 697 267 514 773 336 150 199 641 604 945 951 959 264 31 549 95 339 769 457 624 739 717 754 691 332 81 730 195 895 490 792 798 758 977 643 481 265 660 822 327 413 668 619 772 295 579 277 808 874 270 303 379 679 389 925 406 101 266 972 759 56 689 915 806 942 445 425 858 338 853 686 149 137 96 83 651 850

0

OUT:

UDN

V1 V2 V3 V4 V5 V6 V7 V8 V9 APPLE BANANA PEACH GRAPE PEOPLE FATHER ZLC LZR LX WQ LJL

9999 291 723 425 111 676 9999 9999 9999 788 134 9999 203 848 909 9999 9999 9999 9999 373

291 9999 402 9999 178 841 530 467 9999 315 670 866 870 289 960 9999 9999 621 560 607

723 402 9999 345 122 419 9999 792 518 203 987 341 9999 548 9999 415 9999 785 9999 850

425 9999 345 9999 434 9999 975 328 318 851 768 9999 9999 1 186 9999 9999 9999 39 261

111 178 122 434 9999 605 211 209 719 65 99 697 267 514 773 336 150 199 9999 641

676 841 419 9999 605 9999 604 945 951 959 9999 264 31 549 95 339 769 9999 9999 457

9999 530 9999 975 211 604 9999 624 739 717 754 9999 9999 691 9999 332 9999 9999 9999 81

9999 467 792 328 209 945 624 9999 9999 730 195 895 9999 490 9999 792 9999 798 9999 758

9999 9999 518 318 719 951 739 9999 9999 977 643 481 265 660 822 327 413 9999 9999 668

788 315 203 851 65 959 717 730 977 9999 619 772 295 579 277 808 874 270 303 379

134 670 987 768 99 9999 754 195 643 619 9999 679 9999 9999 389 9999 9999 9999 9999 925

9999 866 341 9999 697 264 9999 895 481 772 679 9999 406 101 9999 266 972 759 56 9999

203 870 9999 9999 267 31 9999 9999 265 295 9999 406 9999 9999 9999 9999 689 915 9999 806

848 289 548 1 514 549 691 490 660 579 9999 101 9999 9999 942 445 425 9999 9999 858

909 960 9999 186 773 95 9999 9999 822 277 389 9999 9999 942 9999 338 853 9999 9999 9999

9999 9999 415 9999 336 339 332 792 327 808 9999 266 9999 445 338 9999 686 149 9999 9999

9999 9999 9999 9999 150 769 9999 9999 413 874 9999 972 689 425 853 686 9999 137 96 9999

9999 621 785 9999 199 9999 9999 798 9999 270 9999 759 915 9999 9999 149 137 9999 83 651

9999 560 9999 39 9999 9999 9999 9999 9999 303 9999 56 9999 9999 9999 9999 96 83 9999 850

373 607 850 261 641 457 81 758 668 379 925 9999 806 858 9999 9999 9999 651 850 9999

(V5,BANANA,99),(V4,FATHER,186),(V6,GRAPE,31),(V7,LJL,81),(WQ,LX,83),(V5,LZR,150),(WQ,PEACH,56),(V4,PEOPLE,1),(V5,V1,111),(V5,V2,178),(V5,V3,122),(WQ,V4,39),(APPLE,V5,65),(FATHER,V6,95),(V5,V7,211),(BANANA,V8,195),(GRAPE,V9,265),(LZR,WQ,96),(LX,ZLC,149)

### 3.4 Kruskal算法的设计

#### 3.4.1 题目以及要求

**题目描述:**

目的：在使用*vector*（数组）、图的邻接表*ADT*以及快速排序的基础上，设计Kruskal算法，用以生成**无向网的最小生成树，并以文本形式输出生成树中各条边以及它们的权值。**将此算法加入到邻接表*ADT*中，在邻接表*ADT*中提供一个公有的成员函数*Kruskal*。

要求：在使用顺序表*ADT*、图的邻接表*ADT*以及快速排序的基础上，设计*Kruskal*算法，用以生成无向网的最小生成树，并以文本形式输出生成树中各条边以及它们的权值。

参考函数原形：

//Kruskal算法  
*template<class TypeOfVer, class TypeOfEdge>  
bool adjlist\_graph<TypeOfVer, TypeOfEdge>::Kruskal( int esize,  int \*\*edge, TypeOfEdge \*weight,  TypeOfEdge &cost);*

//*esize*：边的数量\*\**edge*：输入的边集二维数组\**weight*：初始的权值数组*cost*：树的代价

输出说明：

第一行：图的类型

第二行：顶点集

     空行

第三行：排序前的边的权值结果（按照输入顺序）

第四行：排序前对应的边的序号关系（按照输入顺序）

     空行

第三行：排序后的边的权值结果（按照权值递增排序顺序）

第四行：排序后的对应的边的序号关系（对应权值递增排序顺序）

     空行

第五行：最小生成树的各条边以及它们的权值（格式如测试数据所示）

   空行

第六行：最小生成树的代价

#### 3.4.2概要设计

* 设计*edgeNode* 结构体、*Kruskal\_struct*结构体，增加*operator<*运算符。
* 设计一个*adjlist\_graph*类，基于其上设计*make\_weight\_adjlist\_graph\_map*函数、*make\_weight\_adjlist\_graph*和*insert\_node*函数。
* 对于合并连通分量和检查连通分量,需要设计类内*find\_set*函数以及*Union*递归函数
* 设计类内*public: pair<vector<Kruskal\_struct<V,E>>,E> Kruskal()*函数
* 最终设计*solve\_oj\_tems\_kruskal*函数解决问题

#### 3.4.3算法分析

**算法说明：**

（1）*Kruskal*算法的基本思想是为使生成树上边的权值之和达到最小，则应使生成树中每一条边的权值尽可能地小。初始时，先构造一个只含*n*个顶点的子图 *T*，然后从权值最小的边开始，若它的添加不使*T*中产生回路，则在*T*上加上这条边，如此重复，直至加上*n-1*条边为止。具体来讲，*Kruskal*算法由以下几个步骤组成：

1）设母图*G=（V，E）*为一个具有*n*个顶点的带权的连通网络，其最小生成树的初始状态为有*n*个顶点但无边的非连通图 *T=(V, Φ)*。

2）将*E*中的边按权值的递增顺序排序。

3）选择权值最小的边，若不构成环，则将其加入*T*中，否则，将其弃舍。

4）循环至有*N-1*条边。

（1）在排序的过程中，由于根据边的权值大小进行排序，需要同步保持边的权值和边之间的对应关系。

（2）定义*vector<pair<TypeOfEdge, int>>，pair->first*对应边的权值，*pair->second*对应边，使用*sort*进行排序。

（3）实现*Kruskal*算法的关键是如何判断所选取的边是否与生成树中已保留的边形成回路，这可通过判断边的两个顶点所在的连通分量的方法来解决。为此设计类内成员*map<V,V>fa*来记录每个节点对应的父节点,在初始化时,父节点均为自己,通过设计*find\_set*函数和*union*函数来统一父节点具体实现如下:

**V find\_set(V *x*) {  
 if (*x* != fa[*x*]) {  
 fa[*x*] = find\_set(fa[*x*]);}  
 return fa[*x*];}  
void Union(V *u*, V *v*) {  
 if (fa[*u*] == *u*) {  
 fa[*u*] = fa[*v*];  
 return;} else if (fa[*u*] == fa[*v*]) {  
 return;} else {  
 Union(fa[*u*], *v*);  
 fa[*u*] = fa[*v*];*//父节点统一;* return;}  
}**

**算法时间复杂度:**

*V*代表节点数量，*E*代表边的数量

1、初始化生成树的边集*A*为空集：

2、对集合中的每一个顶点，都将它的集合初始化为自身：

4、将边按权值进行排序：

5、对排序好后的边从小到大进行判断，如果这条边所连的2个顶点不在同一个集合中，则将这条边加入到生成树的边集*A*中，并将此边所连的两个顶点*u*和*v*的集合做一个*Union*操作，如此循环加到生成树中的边集数量为*n-1*时停止。最坏的情况可能要枚举完所有的边，此时要循环|*E*|次，所以这一步的时间复杂度为O(|E|α(V))，其中α为*Ackermann*函数，其增长非常慢，我们可以视为常数。

综上，*Kruskal*算法的时间复杂度为O(|Elog|E|)。

#### 3.4.4使用说明

输入说明：

第一行：权值数据类型（0：int；1：double）//暂不考虑容错处理

第二行：图的类型

第三行：顶点数

第四行：顶点集

第五行：边数

第六行：边集

第七行：权集

程序名称为*Kruskal.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

在输入完数据后,需要使用键盘的*Ctrl+D*(程序内)/*Ctrl+Z*(外部*exe*)即可显示结果.

#### 3.4.5 测试结果以及分析

IN1:int型测试用例

0

UDN

6

1 2 3 4 5 6

10

0 1

0 2

0 3

1 2

1 4

2 3

2 4

2 5

3 5

4 5

6 1 5 5 3 5 6 4 2 6

OUT:

UDN

6 1 5 5 3 5 6 4 2 6

0 1 2 3 4 5 6 7 8 9

1 2 3 4 5 5 5 6 6 6

1 8 4 7 2 3 5 0 6 9

(1,3),1

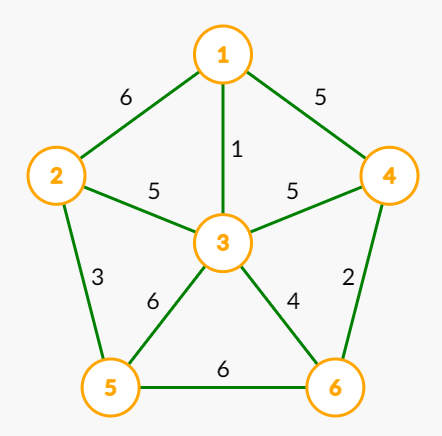
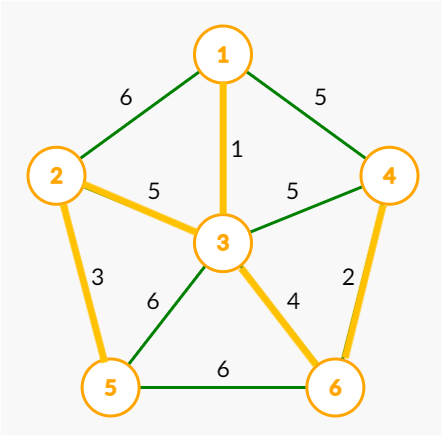
(4,6),2

(2,5),3

(3,6),4

(2,3),5

15

IN2：double型测试用例

1

UDN

6

1 2 3 4 5 6

10

0 1

0 2

0 3

1 2

1 4

2 3

2 4

2 5

3 5

4 5

5.3 1.6 6.2 5.1 5.1 3.9 6.9 4.2 2.2 6.5

OUT：

UDN

5.3 1.6 6.2 5.1 5.1 3.9 6.9 4.2 2.2 6.5

0 1 2 3 4 5 6 7 8 9

1.6 2.2 3.9 4.2 5.1 5.1 5.3 6.2 6.5 6.9

1 8 5 7 3 4 0 2 9 6

(1,3),1.6

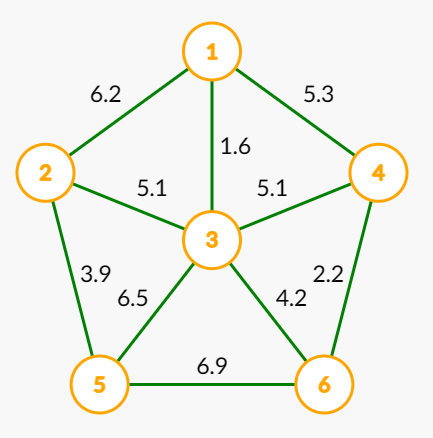
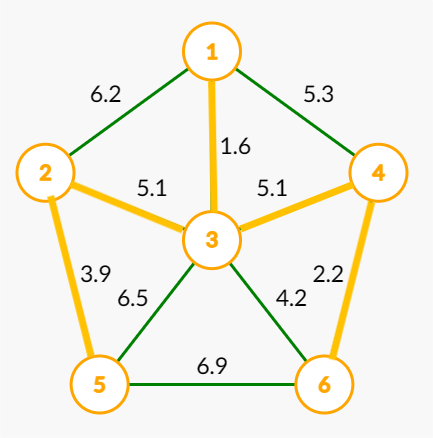
(4,6),2.2

(3,4),3.9

(2,3),5.1

(2,5),5.1

17.9

IN3:特殊用例，权值相同

0

UDN

6

1 2 3 4 5 6

10

0 1

0 2

0 3

1 2

1 4

2 3

2 4

2 5

3 5

4 5

1 1 1 1 1 1 1 1 1 1

OUT:

UDN

1 1 1 1 1 1 1 1 1 1

0 1 2 3 4 5 6 7 8 9

1 1 1 1 1 1 1 1 1 1

0 1 2 3 4 5 6 7 8 9

(1,2),1

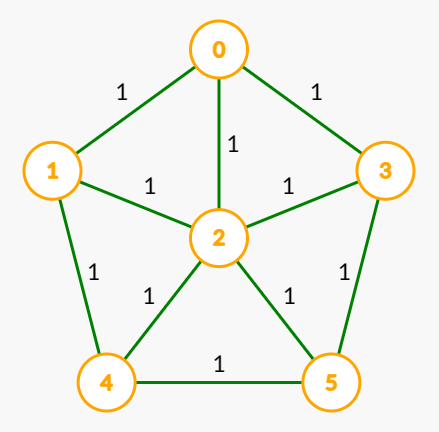
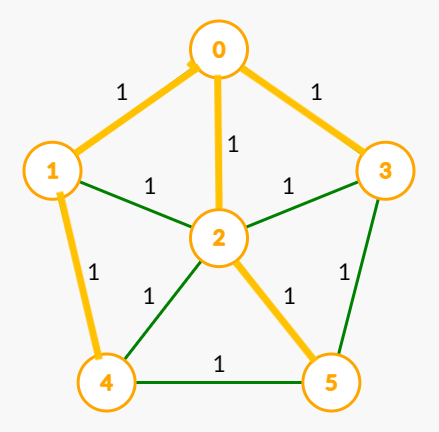
(1,3),1

(1,4),1

(2,5),1

(3,6),1

5

IN4:

0

UDN

6

1 2 3 4 5 6

5

0 1

1 2

2 3

3 4

4 5

1 2 3 4 5

OUT:

UDN

1 2 3 4 5

0 1 2 3 4

1 2 3 4 5

0 1 2 3 4

(1,2),1

(2,3),2

(3,4),3

(4,5),4

(5,6),5

15

形状, 箭头

描述已自动生成 形状, 箭头

描述已自动生成

IN5：大样例测试

0

UDN

20

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

41

0 1

1 2

2 3

3 4

4 5

10 18

18 14

11 14

14 19

19 16

16 5

5 8

8 12

12 5

5 9

9 6

15 7

7 6

6 15

15 2

2 17

17 13

13 0

0 17

10 17

11 10

10 14

14 4

4 16

16 12

12 8

8 9

9 6

6 4

4 2

7 2

7 0

10 18

18 19

19 11

11 13

1 2 3 4 5 5 2 6 7 9 4 1 5 6 6 4 3 2 5 4 3 2 5 7 6 8 9 10 4 3 8 6 5 7 9 3 5 3 7 4 7

OUT：

UDN

1 2 3 4 5 5 2 6 7 9 4 1 5 6 6 4 3 2 5 4 3 2 5 7 6 8 9 10 4 3 8 6 5 7 9 3 5 3 7 4 7

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

1 1 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 5 5 5 5 5 5 5 6 6 6 6 6 7 7 7 7 7 8 8 9 9 9 10

0 11 1 6 17 21 2 16 20 29 35 37 3 10 15 19 28 39 4 5 12 18 22 32 36 7 13 14 24 31 8 23 33 38 40 25 30 9 26 34 27

(1,2),1

(6,9),1

(2,3),2

(19,15),2

(8,7),2

(18,14),2

(3,4),3

(16,8),3

(3,18),3

(17,13),3

(8,3),3

(11,19),3

(4,5),4

(17,6),4

(10,7),4

(5,17),4

(20,12),4

(12,15),6

(11,18),6

60

图片包含 图表

描述已自动生成 图表, 雷达图

描述已自动生成

### 3.5 Dijkstra算法的设计

#### 3.5.1 题目以及要求

**题目描述:**

在使用图的邻接矩阵*ADT*的基础上，设计*Dijkstra*算法，用以解决单源最短路径问题，并以文本形式输出从源点到其余各个顶点的路径以及路径长度。将此算法加入到邻接矩阵*ADT*中，在邻接矩阵*ADT*中提供一个公有的成员函数*Dijkstra*。

提示：

（1）单源最短路径问题：已知有向带权图(简称有向网)*G=(V，E)*，找出从某个源点*s∈V*到*V*中其余各顶点的最短路径。限定权值均为正整数。

（2）目的： 设一有向图*G=（V, E）*，已知各边的权值，以某指定点*v0*为源点，求从*v0*到图的其余各点的最短路径。限定各边上的权值大于或等于0。应按路径“长度”递增的次序，逐步产生最短路径。

（3）*Dijkstra*算法的基本步骤：设*V0*是起始源点，*U* = 已求得最短路径终点集合。*V-U* = 未确定最短路径的顶点的集合，初始时 *U* ={*V0*}。

1）“长度”最短的最短路径是边数为1的长度最小的路径。

2）下一条“长度”最短的路径：

① *Vi ∈ V - U* ，先求出*V0* 到*Vi* 中间只经 *U* 中结点的最短路径；

② 上述最短路径中长度最小者即为下一条长度最短的路径；

③ 将所求最短路径的终点加入*U* 中；

3）重复2）直到求出所有的最短路径。

（4）实现方法：

1）图用带权邻接矩阵存储*ad*[][]；

2）数组*dist*[]存放当前找到的从源点*V0*到每个终点的最短路径长度，其初态为图中直接路径权值；

3）数组*pre*[]表示从*V0*到各终点的最短路径上，此顶点的前一顶点的序号；若从*V0*到某终点无路径，则用-1作为其前一顶点的序号。

参考函数原形：

（1）//Dijkstra算法（成员函数）

*template<class TypeOfVer, class TypeOfEdge>*

*bool adjmatrix\_graph<TypeOfVer, TypeOfEdge>::Dijkstra( int u, TypeOfEdge \*dist, int \*pre);* // u：源点的位序

（2）辅助函数

//最短路径输出（用户函数）

*template<class TypeOfVer, class TypeOfEdge>*

*void searchPath(TypeOfVer \*ver, int \*prev, TypeOfEdge \*dist, int v, int u);* // ver：输入的顶点集  v：源点的位序  u：终点的位序

**输出说明:**

第一行：顶点集

     空行

第二行：图的邻接矩阵

     空行

第三行：dist数组的初值

第四行：pre数组的初值

     空行

第五行：dist数组的值

第六行：pre数组的值

   空行

第七行：源点到其余各顶点的最短路径及最短路径长度（输出格式参见测试数据）

#### 3.5.2概要设计

设计一个*adjmartix\_graph*类，其中抽象数据类型包括：

数据对象：V为顶点数据类型，E为边的数据类型。

基本操作：

*adjmartix\_graph（）*构造函数，对图进行初始化。

*void Dijkstra(int u)* 执行*dji*算法

*print\_edge()*输出图的邻接矩阵*。*

*print\_vers()*输出结点集*。*

*get\_graph\_type()*得到图的类型*。*

基于*adjmartix\_graph*类，设计*make\_adjmartix\_graph（）*函数对图进行构造。

#### 3.5.3算法分析

***算法说明：***

假设路网中每一个顶点都有标号*（dt , pt）*，*dt*是从出发点*s*到点*t*的最短路径长度；*pt*表示从出发点*s*到点*t*的最短路径中*t*前的一个顶点。基本过程如下：

1. 初始化：出发点*ds=0，ps*为空；其它所有点，*di=*无边标记，*pi*未定义。起点设为已标记，记*k*=*s*,其它顶点均未被标记。
2. 计算相连路径的距离：检验从所有已标记的点k到其他直接连接的未标记的点j的距离，并设置:*dj = min[ dj , dk + len (k , j) ]*,其中*len (k , j)*表示从*k*到*j*的路径长度。
3. 选取下一个点：从所有未标记的顶点中选取最小的点 *i*，点*i*被选作最短路径中的一点，并设为已标记。
4. 回溯*i*前一个顶点：从已标记的顶点集合中找到与*i*直接相连的顶点x，并记作*pi=x*。
5. 判断算法是否结束:若所有点被标记，则算法结束；否则记*k=i*，执行步骤2）。

赋权有向图的顶点数和边数分别设为n、m，算法总共进行（n-1）步，每一步选出具有最小dist[s,pi]值的顶点放在集合中，需要的时间。

以图4-1所示测试用例进行分析

赋权有向图如图5-1所示，顶点集为*V={V1,V2,V3,V4,V5,V6}*,选定V1作为源点S，则从*V1*出发到*V*中所有顶点的最短路径如图5-2所示（途径顶点由灰色标记，起点与终点顶点由绿色标记，最短路径由绿色标记）。

V1

V4

V5

V6

V3

V2

3

7

7

4

1

6

3

2

5

10

V1

V4

V5

V6

V3

V2

3

7

7

4

1

6

3

2

5

10

图5-1:赋权有向图

V1

V4

V5

V6

V3

V2

3

7

7

4

1

6

3

2

5

10

V1

V4

V5

V6

V3

V2

3

7

7

4

1

6

3

2

5

10

V1

V4

V5

V6

V3

V2

3

7

7

4

1

6

3

2

5

10

V1

V4

V5

V6

V3

V2

3

7

7

4

1

6

3

2

5

10

图5-2:顶点V2-V6的最短路径

下面对图5-2进行具体说明：

首先我们还是选择 *V1* 为原点*S*，那么在算法的开始*S={V1}*。之后我们计算除了 *V1*以外的其余与*V1*直接相连的顶点到 *V1*的*dist[V2,V1]~dist[V6,V1]*。

由图可知：*dist[V1,V2]=10；dist[V1,V6]=3*。

此时检验*V-S*是否为空集，发现*V-S={V2,V3,V4,V5,V6}*。故在集合V-S中查找相对于集合*S*的最短路径中距离最短的那个顶点*Vj*，此时显然是*V6*。故将*V6*放入集合*S*，说明从源点*V1*到*V6*的最短路径已经找到。

此时集合*S*更新为*S={V1,V6}*，后续在计算相对于*S*的最短路径时，就有两个顶点可以被经过。后续算法同上，不再进行赘述。

算法执行至*V-S={}=*，算法结束。

***算法时间复杂度:***根据分析该算法的时间复杂度为。

#### 3.5.4使用说明

**输入说明:**

第一行：图的类型

第二行：顶点数

第三行：顶点集

第四行：无边标记

第五行：边数

第六行：边集

第七行：权集

第八行：源点位序

程序名称为*test\_04\_graph2.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图4-1、4-2

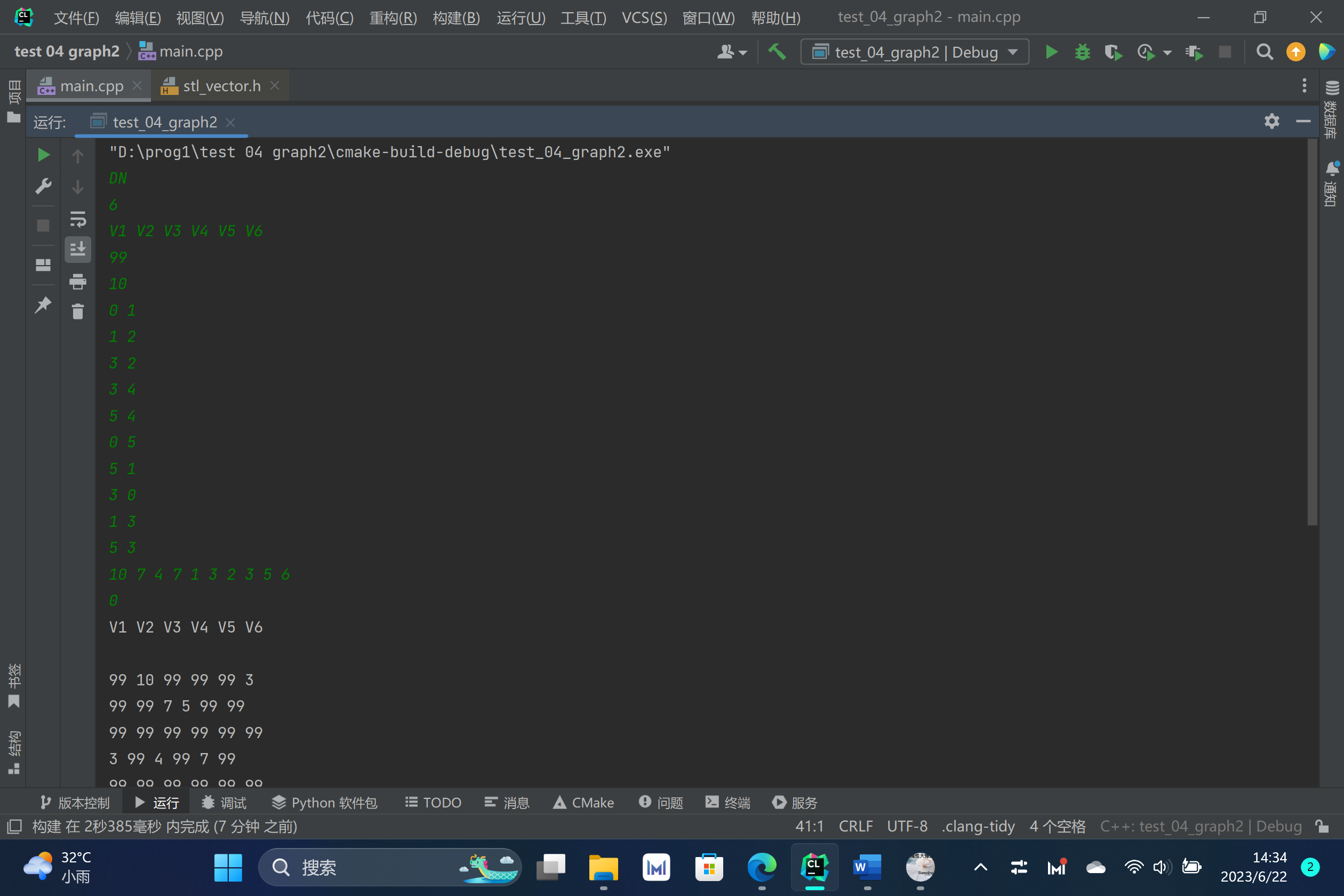


图4-1:运行截图（测试用例输入）

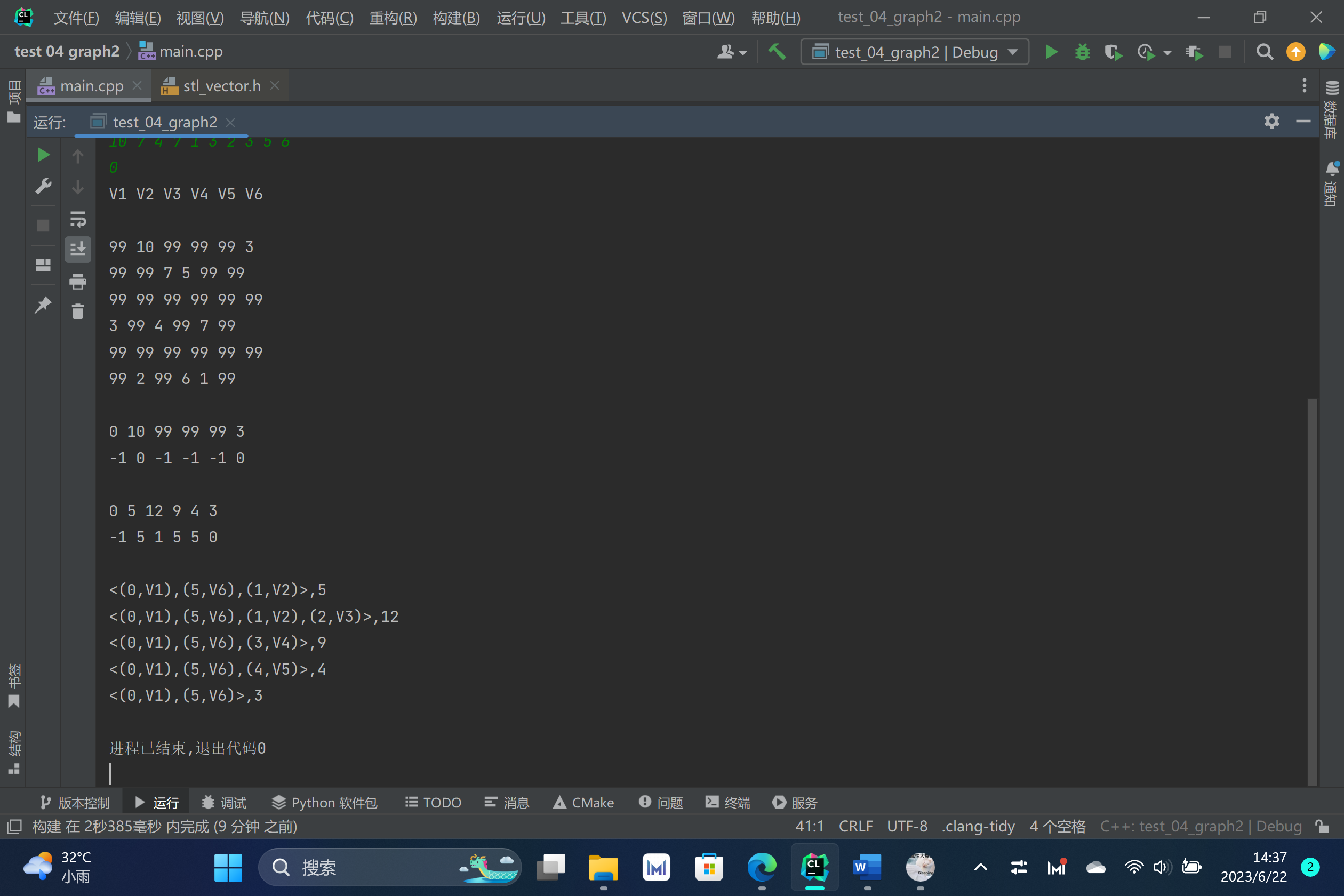


图4-2:运行截图（测试用例输出）

在输入完数据后,需要使用键盘的*Ctrl+D*(程序内)/*Ctrl+Z*(外部*exe*)即可显示结果.

#### 3.5.5 测试结果以及分析

IN1:普通测试用例

DN

7

V1 V2 V3 V4 V5 V6 V7

99

10

0 1

0 2

0 4

0 6

1 5

1 6

2 3

3 4

4 5

5 6

13 8 30 32 9 7 5 6 2 17

0

OUT:

V1 V2 V3 V4 V5 V6 V7

99 13 8 99 30 99 32

99 99 99 99 99 9 7

99 99 99 5 99 99 99

99 99 99 99 6 99 99

99 99 99 99 99 2 99

99 99 99 99 99 99 17

99 99 99 99 99 99 99

0 13 8 99 30 99 32

-1 0 0 -1 0 -1 0

0 13 8 13 19 21 20

-1 0 0 2 3 4 1

<(0,V1),(1,V2)>,13

<(0,V1),(2,V3)>,8

<(0,V1),(2,V3),(3,V4)>,13

<(0,V1),(2,V3),(3,V4),(4,V5)>,19

<(0,V1),(2,V3),(3,V4),(4,V5),(5,V6)>,21

<(0,V1),(1,V2),(6,V7)>,20

IN2:特殊用例,无向图,测试鲁棒性,不会进入dji算法模块

IN:

UDN

7

V1 V2 V3 V4 V5 V6 V7

99

10

0 1

0 2

0 4

0 6

1 5

1 6

2 3

3 4

4 5

5 6

13 8 30 32 9 7 5 6 2 17

0

OUT:

V1 V2 V3 V4 V5 V6 V7

99 13 8 99 30 99 32

99 99 99 99 99 9 7

99 99 99 5 99 99 99

99 99 99 99 6 99 99

99 99 99 99 99 2 99

99 99 99 99 99 99 17

99 99 99 99 99 99 99

0 13 8 99 30 99 32

-1 0 0 -1 0 -1 0

0 13 8 13 19 21 20

-1 0 0 2 3 4 1

IN 3:特殊用例,权值一样

IN:

DN

7

V1 V2 V3 V4 V5 V6 V7

99

10

0 1

0 2

0 4

0 6

1 5

1 6

2 3

3 4

4 5

5 6

1 1 1 1 1 1 1 1 1 1

0

V1 V2 V3 V4 V5 V6 V7

99 1 1 99 1 99 1

99 99 99 99 99 1 1

99 99 99 1 99 99 99

99 99 99 99 1 99 99

99 99 99 99 99 1 99

99 99 99 99 99 99 1

99 99 99 99 99 99 99

OUT:

0 1 1 99 1 99 1

-1 0 0 -1 0 -1 0

0 1 1 2 1 2 1

-1 0 0 2 0 1 0

<(0,V1),(1,V2)>,1

<(0,V1),(2,V3)>,1

<(0,V1),(2,V3),(3,V4)>,2

<(0,V1),(4,V5)>,1

<(0,V1),(1,V2),(5,V6)>,2

<(0,V1),(6,V7)>,1

IN4:特殊用例,连成一条线,且原点位序不为0



IN5:

DN

7

V1 V2 V3 V4 V5 V6 V7

99

6

0 1

1 2

2 3

3 4

4 5

5 6

1 2 3 4 5 6

2

OUT:

V1 V2 V3 V4 V5 V6 V7

99 1 99 99 99 99 99

99 99 2 99 99 99 99

99 99 99 3 99 99 99

99 99 99 99 4 99 99

99 99 99 99 99 5 99

99 99 99 99 99 99 6

99 99 99 99 99 99 99

99 99 0 3 99 99 99

-1 -1 -1 2 -1 -1 -1

99 99 0 3 7 12 18

-1 -1 -1 2 3 4 5

<(0,V1)>,99

<(1,V2)>,99

<(2,V3),(3,V4)>,3

<(2,V3),(3,V4),(4,V5)>,7

<(2,V3),(3,V4),(4,V5),(5,V6)>,12

<(2,V3),(3,V4),(4,V5),(5,V6),(6,V7)>,18

IN5:普通数据,但是是double 类型,测试成功

图示

描述已自动生成

IN:

DN

5

V1 V2 V3 V4 V5

9999

8

0 1

0 3

0 4

1 2

1 4

2 3

2 4

3 4

1.1 1.2 1.3 1.4 1.1 1.2 1.3 0.6

0

OUT:

V1 V2 V3 V4 V5

9999 1.1 9999 1.2 1.3

9999 9999 1.4 9999 1.1

9999 9999 9999 1.2 1.3

9999 9999 9999 9999 0.6

9999 9999 9999 9999 9999

0 1.1 9999 1.2 1.3

-1 0 -1 0 0

0 1.1 2.4 1.2 1.3

-1 0 1 0 0

<(0,V1),(1,V2)>,1.1

<(0,V1),(1,V2),(2,V3)>,2.4

<(0,V1),(3,V4)>,1.2

<(0,V1),(4,V5)>,1.3

进程已结束,退出代码0

IN6:大样例测试,数据集大小为1MB,23ms运行成功,与参考代码输出结果一致,如下图:

图示

描述已自动生成

IN:

DN

20

V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 LX LJL WQ LZR ZLC HQB FXY LT ZJF

9999

152

0 1

0 2

0 3

0 4

0 5

0 6

0 7

0 10

0 11

0 15

0 16

0 17

0 18

0 19

1 2

1 3

1 4

1 5

1 6

1 7

1 8

1 9

1 10

1 11

1 12

1 13

1 14

1 15

1 16

1 17

1 18

1 19

2 3

2 4

2 5

2 7

2 8

2 9

2 10

2 15

2 19

3 4

3 5

3 6

3 7

3 8

3 9

3 10

3 11

3 12

3 15

3 16

3 17

3 18

3 19

4 5

4 6

4 7

4 8

4 9

4 10

4 11

4 12

4 13

4 15

4 19

5 6

5 7

5 8

5 9

5 10

5 11

5 12

5 13

5 14

5 15

5 16

5 17

5 18

5 19

6 7

6 8

6 9

6 10

6 11

6 12

6 13

6 14

6 15

6 19

7 8

7 9

7 10

7 11

7 12

7 13

7 14

7 15

7 16

7 19

8 9

8 10

8 11

8 12

8 13

8 14

8 15

8 19

9 10

9 11

9 12

9 13

9 14

9 15

9 16

9 17

9 19

10 11

10 12

10 13

10 14

10 15

10 16

10 17

10 18

10 19

11 12

11 14

11 18

11 19

12 13

12 14

12 16

12 18

12 19

13 14

13 15

13 17

13 18

13 19

14 15

14 18

14 19

15 16

15 17

15 19

16 17

16 18

16 19

17 18

17 19

18 19

38 71 66 24 1 75 59 19 25 17 88 63 75 63 80 79 4 77 48 100 28 82 7 41 95 79 45 87 84 78 80 15 73 28 96 98 89 87 44 100 73 19 61 48 47 72 79 27 88 41 75 27 27 98 12 37 89 29 33 81 2 53 29 46 35 50 34 74 23 69 24 50 72 77 59 26 78 6 36 51 54 19 42 23 90 66 60 12 67 31 6 6 23 70 29 19 23 51 66 37 23 71 80 27 28 31 18 45 66 1 59 9 26 91 41 73 13 58 80 74 62 94 16 15 82 9 67 50 97 36 88 5 37 28 1 4 98 12 34 85 20 98 36 17 81 63 75 91 81 37 34 11

0

OUT:

V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 LX LJL WQ LZR ZLC HQB FXY LT ZJF

9999 38 71 66 24 1 75 59 9999 9999 19 25 9999 9999 9999 17 88 63 75 63

9999 9999 80 79 4 77 48 100 28 82 7 41 95 79 45 87 84 78 80 15

9999 9999 9999 73 28 96 9999 98 89 87 44 9999 9999 9999 9999 100 9999 9999 9999 73

9999 9999 9999 9999 19 61 48 47 72 79 27 88 41 9999 9999 75 27 27 98 12

9999 9999 9999 9999 9999 37 89 29 33 81 2 53 29 46 9999 35 9999 9999 9999 50

9999 9999 9999 9999 9999 9999 34 74 23 69 24 50 72 77 59 26 78 6 36 51

9999 9999 9999 9999 9999 9999 9999 54 19 42 23 90 66 60 12 67 9999 9999 9999 31

9999 9999 9999 9999 9999 9999 9999 9999 6 6 23 70 29 19 23 51 66 9999 9999 37

9999 9999 9999 9999 9999 9999 9999 9999 9999 23 71 80 27 28 31 18 9999 9999 9999 45

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 66 1 59 9 26 91 41 73 9999 13

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 58 80 74 62 94 16 15 82 9

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 67 9999 50 9999 9999 9999 97 36

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 88 5 9999 37 9999 28 1

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 4 98 9999 12 34 85

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 20 9999 9999 98 36

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 17 81 9999 63

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 75 91 81

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 37 34

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 11

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999

0 38 71 66 24 1 75 59 9999 9999 19 25 9999 9999 9999 17 88 63 75 63

-1 0 0 0 0 0 0 0 -1 -1 0 0 -1 -1 -1 0 0 0 0 0

0 38 71 66 24 1 35 53 24 47 19 25 51 52 47 17 34 7 37 28

-1 0 0 0 0 0 5 4 5 8 0 0 8 8 6 0 15 5 5 10

<(0,V1),(1,V2)>,38

<(0,V1),(2,V3)>,71

<(0,V1),(3,V4)>,66

<(0,V1),(4,V5)>,24

<(0,V1),(5,V6)>,1

<(0,V1),(5,V6),(6,V7)>,35

<(0,V1),(4,V5),(7,V8)>,53

<(0,V1),(5,V6),(8,V9)>,24

<(0,V1),(5,V6),(8,V9),(9,V10)>,47

<(0,V1),(10,V11)>,19

<(0,V1),(11,LX)>,25

<(0,V1),(5,V6),(8,V9),(12,LJL)>,51

<(0,V1),(5,V6),(8,V9),(13,WQ)>,52

<(0,V1),(5,V6),(6,V7),(14,LZR)>,47

<(0,V1),(15,ZLC)>,17

<(0,V1),(15,ZLC),(16,HQB)>,34

<(0,V1),(5,V6),(17,FXY)>,7

<(0,V1),(5,V6),(18,LT)>,37

<(0,V1),(10,V11),(19,ZJF)>,28

### 3.6获取AOE网的关键路径

#### 3.6.1题目以及要求

建立一个有向网*AOE*网，设计并完成一算法*Get\_CriticalPath()*，获取关键路径。该路径仅输出，不须保存。

提示：在AOE网中，

（1）关键活动：开始时间余量为0的活动。即活动的最早开始时间等于它的最迟开始时间。

（2）根据各个顶点的*Ve*和*Vl*值，在求得每条弧*s*的最早开始时间*e*[*s*]和最迟开始时间*l*[*s*]后，若某条弧满足条件*e*[*s*]=*l*[*s*]，该弧所对应的活动即为关键活动。

（3）关键路径：由关键活动所形成的从源点到汇点的每一条路径（注意：关键路径可能有多条）。

参考函数原型：

//获取AOE网各顶点事件的最早发生时间*ve*和最迟发生时间*vl*、活动*ak*的最早开始时间*e*和最迟开始时间*l*、一条关键路径

*template<class TypeOfVer, class TypeOfEdge>*

*bool adjlist\_graph<TypeOfVer, TypeOfEdge>::Get\_CriticalPath(int ve[], int vl[]);*

#### 3.6.2概要设计

1. 由于是使用邻接表储存,故要设计邻接表储存介质: ***struct edgeNode*** ,以及邻接表类***class adjlist\_graph***
2. 设计用户函数template<class V, class E>adjlist\_graph<V, E> make\_weight\_adjlist\_graph()用来构造一张有权图
3. 设计类内成员函数*void AOE*()用来寻找关键路径
4. 设计类外用户函数*template <class V> void solve\_team\_6*()用来启动这个项目

#### 3.6.3算法分析

**算法说明:**

首先要理解*AOE*网所具有的性质:①只有在某顶点所代表的事件发生后,从该顶点出发的各有向边所代表的活动才能开始;②只有在进入某顶点的各有向边所代表的活动都已经结束时,该顶点所代表的事件才能发生.

故需要定义以下几个数组:

1. 事件的最早发生时间

其指代从原点到顶点的最长路径长度.事件的最早发生时间决定了所有从开始的活动能开工的最早时间,可以使用下面的递推公式计算:

其中为的后继

计算该部分的核心代码如下:

**for(int *i*=0;*i*<verList.size();*i*++)  
{  
 edgeNode<int>\**p*=verList[*i*]->next;  
 while(*p*)  
 {  
 *ve*[*p*->data]=max(*ve*[*p*->data],*ve*[*i*]+*p*->weight);  
 *p*=*p*->next;  
 }  
}**

1. 事件的最迟发生时间

它是指在不推迟整个工程完成的前提下,即保证它的后继事件在其最迟发生时间能够发生时,该事件最迟必须发生的时间,可以用下面的递推公式计算:

其中为的任意前驱

计算该部分的核心代码如下:

***vl*[Vers-1]=*ve*[Vers-1];  
for(int *i*=verList.size()-1;*i*>=0;*i*--)  
{  
 edgeNode<int>\**p*=verList[*i*]->next;  
 while(*p*)  
 {  
 *vl*[*i*]=min(*vl*[*i*],*vl*[*p*->data]-*p*->weight);  
 *p*=*p*->next;  
 }  
}**

1. 活动的最早开始时间

它是指该活动弧的起点所表示的事件的最早发生时间.若边表示活动则有:

计算该部分的核心代码如下:

***//活动ai开始的最早时间*map<pair<int, int>, int> *e*;  
vector<pair<int, int>> *tmp*;  
for (int *i* = 0; *i* < verList.size(); *i*++) {  
 edgeNode<int> \**p* = verList[*i*]->next;  
 while (*p*) {  
 *e*[make\_pair(*i*, *p*->data)] = *ve*[*i*];  
 *tmp*.push\_back({*i*, *p*->data});  
 *p* = *p*->next;  
 }  
}**

1. 活动的最迟发生时间

它是指该活动弧的终点所表示的事件的最迟发生时间与该活动的时间之差.若边表示活动,则有:

计算该部分的核心代码如下:

**for (int *i* = 0; *i* < verList.size(); *i*++) {  
 edgeNode<int> \**p* = verList[*i*]->next;  
 while (*p*) {  
 *l*[make\_pair(*i*, *p*->data)] = *vl*[*p*->data] - *p*->weight;  
 *p* = *p*->next;  
 }  
}**

1. 判断关键活动

只需要检查即可

1. 判断是否存在环

我们认为一个有效的AOE网是不应该存在环的,所以本小组判定规则为:如果发现存在一个或以上的环,将直接return,退出AOE函数,具体代码如下:

**bool check\_circle() {  
 vector<int> *has\_visited*;  
 queue<int> *que*;  
 vector<int> *vis*(ver.size(), 0);  
 vector<int> *degree*(ver.size(), 0);  
 for (int *i* = 0; *i* < ver.size(); *i*++) {  
 int *k* = get\_in\_degree(*i*);  
 *degree*[*i*] = *k*;  
 if (!*k*) {  
 *que*.push(*i*);  
 *has\_visited*.push\_back(*i*);  
 *vis*[*i*] = 1;  
 }  
 }  
 while (!*que*.empty()) {  
 edgeNode<E> \**p* = verList[*que*.front()]->next;  
 while (*p*) {  
 *degree*[*p*->data]--;  
 if (*degree*[*p*->data] == 0 && !*vis*[*p*->data])*//康康有没有被访问过* {  
 *que*.push(*p*->data);  
 *has\_visited*.push\_back(*p*->data);  
 *vis*[*p*->data] = 1;  
 }  
 *p* = *p*->next;  
 }  
 *que*.pop();  
 }  
 if (*has\_visited*.size() == ver.size()) {  
 return false;  
 } else {  
 return true;  
 }  
}**

**时间复杂度分析**

计算数组时,共访问了条边故耗时,在计算数组时,由于用到了,内部红黑树结构,单次插入,故总共是,算上后期需要遍历打印出来的时间复杂度,以及初始化数组的时间复杂度,整个算法的时间复杂度是:



#### 3.6.4使用说明

输入说明 :

第一行：图的类型

第二行：结点数

第三行：结点集

第四行：边数

第五行：边集

第六行：权集

**运行界面如下:在Clion2023.1,MVSC 17.0编译器的环境下可以正常运行**

图形用户界面

低可信度描述已自动生成

#### 3.6.5测试结果以及分析

* 第一组数据:特殊数据,仅含有一个顶点:



IN:

DN

1

A

0

OUT:

A

A

A 0 0

* 第二组数据:仅有一条边



In:

DN

2

A B

1

0 1

2

OUT:

A B

A->1(2)

B

A 0 0

B 2 2

<A,B> 0 0

(A,B)

* 第三组数据,测试鲁棒性,图的类型不为有向网,应当直接退出

IN:

UDN

6

A B C D E F

8

0 1

0 2

1 3

1 4

2 3

2 5

3 5

4 5

3 2 2 3 4 3 2 1

OUT

A B C D E F

A->2(2)->1(3)

B->4(3)->3(2)->0(3)

C->5(3)->3(4)->0(2)

D->5(2)->2(4)->1(2)

E->5(1)->1(3)

F->4(1)->3(2)->2(3)

* 第四组数据:全部连成一个环



IN:

DN

5

V1 V2 V3 V4 V5

5

0 1

1 2

2 3

3 4

4 0

4 3 6 4 6

OUT:

V1 V2 V3 V4 V5

V1->1(4)

V2->2(3)

V3->3(6)

V4->4(4)

V5->0(6)

* 第四组数据:有两个环



IN:

DN

7

V1 V2 V3 V4 V5 V6 V7

9

0 1

0 2

1 4

2 1

3 2

3 5

4 3

5 6

6 3

4 4 5 5 5 5 1 1 1

OUT:

V1 V2 V3 V4 V5 V6 V7

V1->2(4)->1(4)

V2->4(5)

V3->1(5)

V4->5(5)->2(5)

V5->3(1)

V6->6(1)

V7->3(1)

* 第五组数据:所有边权值一样,保证只有一个入度为0的点



IN:

DN

8

V1 V2 V3 V4 V5 V6 V7 V8

10

0 1

0 2

0 3

1 4

2 5

3 6

4 7

5 7

6 5

6 7

1 1 1 1 1 1 1 1 1 1

OUT:

V1 V2 V3 V4 V5 V6 V7 V8

V1->3(1)->2(1)->1(1)

V2->4(1)

V3->5(1)

V4->6(1)

V5->7(1)

V6->7(1)

V7->7(1)->5(1)

V8

V1 0 0

V2 1 1

V3 1 1

V4 1 1

V5 2 2

V6 3 2

V7 2 2

V8 3 3

<V1,V4> 0 0

<V1,V3> 0 0

<V1,V2> 0 0

<V2,V5> 1 1

<V3,V6> 1 1

<V4,V7> 1 1

<V5,V8> 2 2

<V6,V8> 3 2

<V7,V8> 2 2

<V7,V6> 2 1

(V1,V4)->(V1,V3)->(V1,V2)->(V2,V5)->(V3,V6)->(V4,V7)->(V5,V8)->(V7,V8)

* 第七组数据:有多个入度为0的点

图示, 示意图

描述已自动生成

IN:

DN

10

V1 V2 V3 V4 V5 V6 V7 V8 V9 V10

11

0 3

1 3

2 4

3 5

4 5

4 8

5 6

5 7

6 8

7 8

8 9

10 5 8 10 5 4 1 2 4 3 7

OUT:

V1 V2 V3 V4 V5 V6 V7 V8 V9 V10

V1->3(10)

V2->3(5)

V3->4(8)

V4->5(10)

V5->8(4)->5(5)

V6->7(2)->6(1)

V7->8(4)

V8->8(3)

V9->9(7)

V10

V1 0 0

V2 0 5

V3 0 7

V4 10 10

V5 8 15

V6 20 20

V7 21 21

V8 22 22

V9 25 25

V10 32 32

<V1,V4> 0 0

<V2,V4> 0 5

<V3,V5> 0 7

<V4,V6> 10 10

<V5,V9> 8 21

<V5,V6> 8 15

<V6,V8> 20 20

<V6,V7> 20 20

<V7,V9> 21 21

<V8,V9> 22 22

<V9,V10> 25 25

(V1,V4)->(V4,V6)->(V6,V8)->(V6,V7)->(V7,V9)->(V8,V9)->(V9,V10)

* 第八组数据:有权值为0的点

图示

描述已自动生成

IN:

DN

6

V1 V2 V3 V4 V5 V6

5

0 1

0 2

0 3

2 4

4 5

0 1 4 6 0

OUT:

V1 V2 V3 V4 V5 V6

V1->3(4)->2(1)->1(0)

V2

V3->4(6)

V4

V5->5(0)

V6

V1 0 0

V2 0 7

V3 1 1

V4 4 7

V5 7 7

V6 7 7

<V1,V4> 0 3

<V1,V3> 0 0

<V1,V2> 0 7

<V3,V5> 1 1

<V5,V6> 7 7

(V1,V3)->(V3,V5)->(V5,V6)

* 第九组数据:普通数据

图示

描述已自动生成

IN:

DN

11

V0 V1 V2 V3 V4 V5 V6 V7 V8 V9 V10

15

0 1

0 2

0 3

1 2

1 3

2 3

2 4

3 5

4 6

5 7

6 7

7 8

7 9

8 9

9 10

2 1 2 3 6 8 10 12 9 11 17 19 20 1 2

OUT:

V0 V1 V2 V3 V4 V5 V6 V7 V8 V9 V10

V0->3(2)->2(1)->1(2)

V1->3(6)->2(3)

V2->4(10)->3(8)

V3->5(12)

V4->6(9)

V5->7(11)

V6->7(17)

V7->9(20)->8(19)

V8->9(1)

V9->10(2)

V10

V0 0 0

V1 2 2

V2 5 5

V3 13 18

V4 15 15

V5 25 30

V6 24 24

V7 41 41

V8 60 60

V9 61 61

V10 63 63

<V0,V3> 0 16

<V0,V2> 0 4

<V0,V1> 0 0

<V1,V3> 2 12

<V1,V2> 2 2

<V2,V4> 5 5

<V2,V3> 5 10

<V3,V5> 13 18

<V4,V6> 15 15

<V5,V7> 25 30

<V6,V7> 24 24

<V7,V9> 41 41

<V7,V8> 41 41

<V8,V9> 60 60

<V9,V10> 61 61

(V0,V1)->(V1,V2)->(V2,V4)->(V4,V6)->(V6,V7)->(V7,V9)->(V7,V8)->(V8,V9)->(V9,V10)

* 第十组数据:大型数据,负责检查时间复杂度,输入数据不保证只有一个汇点

图示

描述已自动生成

IN:

DN

20

V0 V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 ZLC LZR WQ LX LJL

100

0 1

0 2

0 3

0 4

0 5

0 6

0 7

0 9

1 2

1 4

1 5

1 7

1 9

1 10

1 11

2 4

2 5

2 6

2 7

2 8

2 9

2 10

2 14

2 17

2 18

3 5

3 7

3 9

3 10

3 13

3 15

3 18

4 5

4 6

4 8

4 9

4 10

4 11

4 12

4 13

4 14

4 15

4 16

4 18

5 8

5 9

5 10

5 14

6 10

6 11

6 14

6 15

6 18

7 10

7 11

7 12

7 14

8 9

8 10

8 11

8 12

8 13

8 14

8 15

8 16

9 10

9 11

9 13

9 14

10 11

10 12

10 13

10 14

10 15

10 16

10 17

11 12

11 14

11 15

11 16

11 18

12 13

12 14

12 17

12 18

13 15

13 16

13 17

13 18

14 15

14 16

14 18

15 16

13 18

15 19

16 17

16 18

16 19

17 19

18 19

135 800 502 9 668 134 890 483 868 6 902 920 649 66 773 105 700 980 606 996 360 128 368 596 367 219 705 647 296 773 450 0 355 701 865 586 256 373 981 221 476 118 808 741 113 546 336 662 991 164 940 833 337 914 141 619 652 988 625 382 553 458 278 329 67 726 973 680 14 129 677 993 912 307 879 551 473 406 66 46 582 185 708 383 441 672 183 899 947 404 696 498 855 217 121 2 951 106 575 752

OUT:

V0 V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 ZLC LZR WQ LX LJL

V0->9(483)->7(890)->6(134)->5(668)->4(9)->3(502)->2(800)->1(135)

V1->11(773)->10(66)->9(649)->7(920)->5(902)->4(6)->2(868)

V2->18(367)->17(596)->14(368)->10(128)->9(360)->8(996)->7(606)->6(980)->5(700)->4(105)

V3->18(0)->15(450)->13(773)->10(296)->9(647)->7(705)->5(219)

V4->18(741)->16(808)->15(118)->14(476)->13(221)->12(981)->11(373)->10(256)->9(586)->8(865)->6(701)->5(355)

V5->14(662)->10(336)->9(546)->8(113)

V6->18(337)->15(833)->14(940)->11(164)->10(991)

V7->14(652)->12(619)->11(141)->10(914)

V8->16(67)->15(329)->14(278)->13(458)->12(553)->11(382)->10(625)->9(988)

V9->14(14)->13(680)->11(973)->10(726)

V10->17(551)->16(879)->15(307)->14(912)->13(993)->12(677)->11(129)

V11->18(582)->16(46)->15(66)->14(406)->12(473)

V12->18(441)->17(383)->14(708)->13(185)

V13->18(217)->18(947)->17(899)->16(183)->15(672)

V14->18(498)->16(696)->15(404)

ZLC->19(121)->16(855)

LZR->19(106)->18(951)->17(2)

WQ->19(575)

LX->19(752)

LJL

V0 0 0

V1 135 135

V2 1003 1003

V3 502 1667

V4 1108 1134

V5 1703 1886

V6 1983 2765

V7 1609 2842

V8 1999 1999

V9 2987 2987

V10 3713 3756

V11 3960 3960

V12 4433 4433

V13 4706 4873

V14 5141 5141

ZLC 5545 5545

LZR 6400 6400

WQ 6402 7528

LX 7351 7351

LJL 8103 8103

<V0,V9> 0 2504

<V0,V7> 0 1952

<V0,V6> 0 2631

<V0,V5> 0 1218

<V0,V4> 0 1125

<V0,V3> 0 1165

<V0,V2> 0 203

<V0,V1> 0 0

<V1,V11> 135 3187

<V1,V10> 135 3690

<V1,V9> 135 2338

<V1,V7> 135 1922

<V1,V5> 135 984

<V1,V4> 135 1128

<V1,V2> 135 135

<V2,LX> 1003 6984

<V2,WQ> 1003 6932

<V2,V14> 1003 4773

<V2,V10> 1003 3628

<V2,V9> 1003 2627

<V2,V8> 1003 1003

<V2,V7> 1003 2236

<V2,V6> 1003 1785

<V2,V5> 1003 1186

<V2,V4> 1003 1029

<V3,LX> 502 7351

<V3,ZLC> 502 5095

<V3,V13> 502 4100

<V3,V10> 502 3460

<V3,V9> 502 2340

<V3,V7> 502 2137

<V3,V5> 502 1667

<V4,LX> 1108 6610

<V4,LZR> 1108 5592

<V4,ZLC> 1108 5427

<V4,V14> 1108 4665

<V4,V13> 1108 4652

<V4,V12> 1108 3452

<V4,V11> 1108 3587

<V4,V10> 1108 3500

<V4,V9> 1108 2401

<V4,V8> 1108 1134

<V4,V6> 1108 2064

<V4,V5> 1108 1531

<V5,V14> 1703 4479

<V5,V10> 1703 3420

<V5,V9> 1703 2441

<V5,V8> 1703 1886

<V6,LX> 1983 7014

<V6,ZLC> 1983 4712

<V6,V14> 1983 4201

<V6,V11> 1983 3796

<V6,V10> 1983 2765

<V7,V14> 1609 4489

<V7,V12> 1609 3814

<V7,V11> 1609 3819

<V7,V10> 1609 2842

<V8,LZR> 1999 6333

<V8,ZLC> 1999 5216

<V8,V14> 1999 4863

<V8,V13> 1999 4415

<V8,V12> 1999 3880

<V8,V11> 1999 3578

<V8,V10> 1999 3131

<V8,V9> 1999 1999

<V9,V14> 2987 5127

<V9,V13> 2987 4193

<V9,V11> 2987 2987

<V9,V10> 2987 3030

<V10,WQ> 3713 6977

<V10,LZR> 3713 5521

<V10,ZLC> 3713 5238

<V10,V14> 3713 4229

<V10,V13> 3713 3880

<V10,V12> 3713 3756

<V10,V11> 3713 3831

<V11,LX> 3960 6769

<V11,LZR> 3960 6354

<V11,ZLC> 3960 5479

<V11,V14> 3960 4735

<V11,V12> 3960 3960

<V12,LX> 4433 6910

<V12,WQ> 4433 7145

<V12,V14> 4433 4433

<V12,V13> 4433 4688

<V13,LX> 4706 6404

<V13,LX> 4706 6404

<V13,WQ> 4706 6629

<V13,LZR> 4706 6217

<V13,ZLC> 4706 4873

<V14,LX> 5141 6853

<V14,LZR> 5141 5704

<V14,ZLC> 5141 5141

<ZLC,LJL> 5545 7982

<ZLC,LZR> 5545 5545

<LZR,LJL> 6400 7997

<LZR,LX> 6400 6400

<LZR,WQ> 6400 7526

<WQ,LJL> 6402 7528

(V0,V1)->(V1,V2)->(V2,V8)->(V8,V9)->(V9,V11)->(V11,V12)->(V12,V14)->(V14,ZLC)->(ZLC,LZR)->(LZR,LX)

**3.7** **判断给定的二叉树是否是二叉排序树**

**3.7.1 题目以及要求**

**题目描述:**

在二叉树的二叉链表存储形式建立的基础上，使用递归的程序设计方法，设计并完成判断一棵给定的二叉树是否是二叉排序树的算法。

初始条件：二叉树*T*。

操作结果：若*T*是二叉排序树，则返回*true*，否则返回*false*。

提示：

（1）首先对二叉树*T*进行一次中序遍历，遍历结果存放在顺序表*A*中

（2）针对顺序表*A*进行一趟冒泡排序。若交换标志不改变，则说明是一棵二叉排序树。

参考函数原形：

（1）主函数：针对顺序表*A*进行一趟冒泡排序，判断是否是二叉排序树

*template<class ElemType>  
bool IsBST(vector<ElemType> &A);* //*A*为存放中序遍历结果的顺序表

（2）辅助函数1

//重载二叉树的中序遍历（用户函数）

*template<class ElemType>*

*bool InOrderTraverse(BinaryTree<ElemType> &T, vector<ElemType> &A);*

（3）辅助函数2

//重载二叉树的中序遍历（成员函数）

*bool InOrderTraverse( BinaryTreeNode<ElemType> \*root, vector<ElemType> &A, bool (\*visit1)(BinaryTreeNode<ElemType> \*root, vector<ElemType> &A) ) const;*

（4）辅助函数3

//重写遍历的*visit*函数，将遍历结果依次存放在顺序表*A*中（为避免连锁改动，将函数名由*visit*改为*visit1*）

*template<class ElemType>*

*bool visit1(BinaryTreeNode<ElemType> \* root, vector<ElemType> &A);*

**输出说明:**

第一行：二叉树的中序遍历结果

第二行：*true*（是二叉排序树）/*false*（不是二叉排序树）

第六行：*pre*数组的值

空行

第七行：源点到其余各顶点的最短路径及最短路径长度（输出格式参见测试数据）

**3.7.2概要设计**

设计一个*BinaryTree*类，其中抽象数据类型包括：

基本操作：

*vector<ElemType> Create\_tree\_with\_level(ElemType &str,ElemType&empty)* 根据层次遍历结果构造二叉树，并实现把输入数据中那些非终止符的字符保存到顺序表*A*中。

*vector<ElemType> InOrderTraverse(BinaryTreeNode<ElemType>\*root\_)print\_edge()* 中序遍历二叉树，同时将遍历结果保存在顺序表B中。

*sort()* 对顺序表*A*进行排序

最后比较*B*和排序后的顺序表*A*

**3.7.3算法分析**

***算法说明****：*首先根据层次遍历的结果构建一棵二叉树，同时将不含分隔符的结点元素储存在顺序表A中。接着对构建好的二叉树进行中序遍历，并将遍历结果储存在顺序表B中。需要的时间对二叉树进行中序遍历。利用*sort*()函数对顺序表A中的元素进行从小到大的排序。需要的时间实现排序。若二叉树为二叉排序树，则中序遍历结果中元素应按从小到大的顺序排列，即条件顺序表A与B相同成立。故判断两顺序表是否成立，以此得到二叉树是否为二叉排序树的结论，判断是否一致的过程需要。

*算法时间复杂度:*根据分析该算法的时间复杂度为

**3.7.4使用说明**

**输入说明:**

第一行：表示无孩子或指针为空的特殊分隔符

第二行：二叉树的层次次序序列（结点元素之间以空格分隔）（参见二叉树*ADT*的建立 20题）

程序名称为*test\_04\_graph2.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图7-1

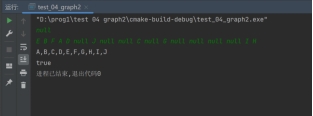


图7-1:运行截图（测试用例输入）

在输入完数据后,需要使用键盘的*Ctrl+D*(程序内)/*Ctrl+Z*(外部*exe*)即可显示结果.

**3.7.5 测试结果以及分析**

* 第一组数据:二叉树为空

IN：null

null

OUT：true

* 第二组数据:二叉树仅有一个结点

IN：null

A null null

OUT：true

* 第三组数据:二叉树左子树为空

IN: null

1 null 5 null null null null

OUT：1,5

true

* 第四组数据:二叉树右子树为空

IN: null

5 1 null null null null null

OUT：1,5

true

* 第五组数据:二叉树左右子树均存在

IN: null

4 2 6 1 3 5 7 null null null null null null null null

OUT：1,2,3,4,5,6,7

true

**3.8 AVL树的判断**

**3.8.1 题目以及要求**

**题目描述:**

目的：设计并实现一个算法*Judge\_AVL*，用来判断1棵给定的二叉排序树是否是平衡二叉树。

提示：

（1）若一棵二叉排序树中每个结点的左、右子树的深度之差（平衡因子）的绝对值不超过1，则称这样的二叉树为平衡二叉树（*AVL*树）。

（2）在小组任务12的基础上，首先判断该二叉树是否是一棵二叉排序树。如不是，则可直接给出*false*（不是*AVL*树）；如是，进入（3）

（3）使用任意的遍历方式对该二叉树进行遍历（测试数据使用非递归的后序遍历得到，参考个人任务12），在*visit*函数中计算访问的结点的平衡因子。参照个人任务13，但要稍作修改（不要通过数据元素的值，而是直接通过遍历到的结点地址来定位）。

（4）如有结点的平衡因子的绝对值超过1，则可给出*false*；如所有结点的平衡因子的绝对值都不超过1，则可给出*true*。

注意：结点的数据域的类型为*string*

**3.8.2概要设计**

* 设计*BinaryTreeNode*结构体，基于此设计*BinaryTree*类建立二叉树的存储结构（外壳）
* 构造*CreateTree*函数，建立二叉树的存储结构（递归部分，成员函数）
* *BinaryTree*类中包含*InorderTraverse*函数判断是否为二叉排序树。
* *BinaryTree*类中包含*visit\_AVL*函数、*check\_AVL*函数等用来验证是否为AVL二叉排序树。

**3.8.3算法分析**

**算法说明：**

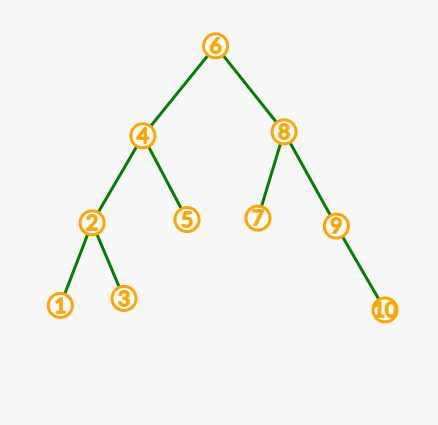
首先中序遍历二叉树，通过对结点域的值*sort*排序来判断是否为二叉排序树。

平衡二叉树（*AVL*树）是满足下面条件的二叉树：空树；左右子树均为*AVL*树，且左右子树的深度之差的绝对值不大于1。由此可知，要判断一棵树是否是*AVL*树，只需判断其左右子树的深度之差。求子树深度采用递归算法即可。

另外根据题目要求,要在输出*false*时候及时终止程序,故需要加入如下语句:

**if(!visit\_AVL(*tmp*))  
 {  
 cout<<"false";  
 exit(0);  
 }  
}**

*如下图所示*



算法时间复杂度分析：

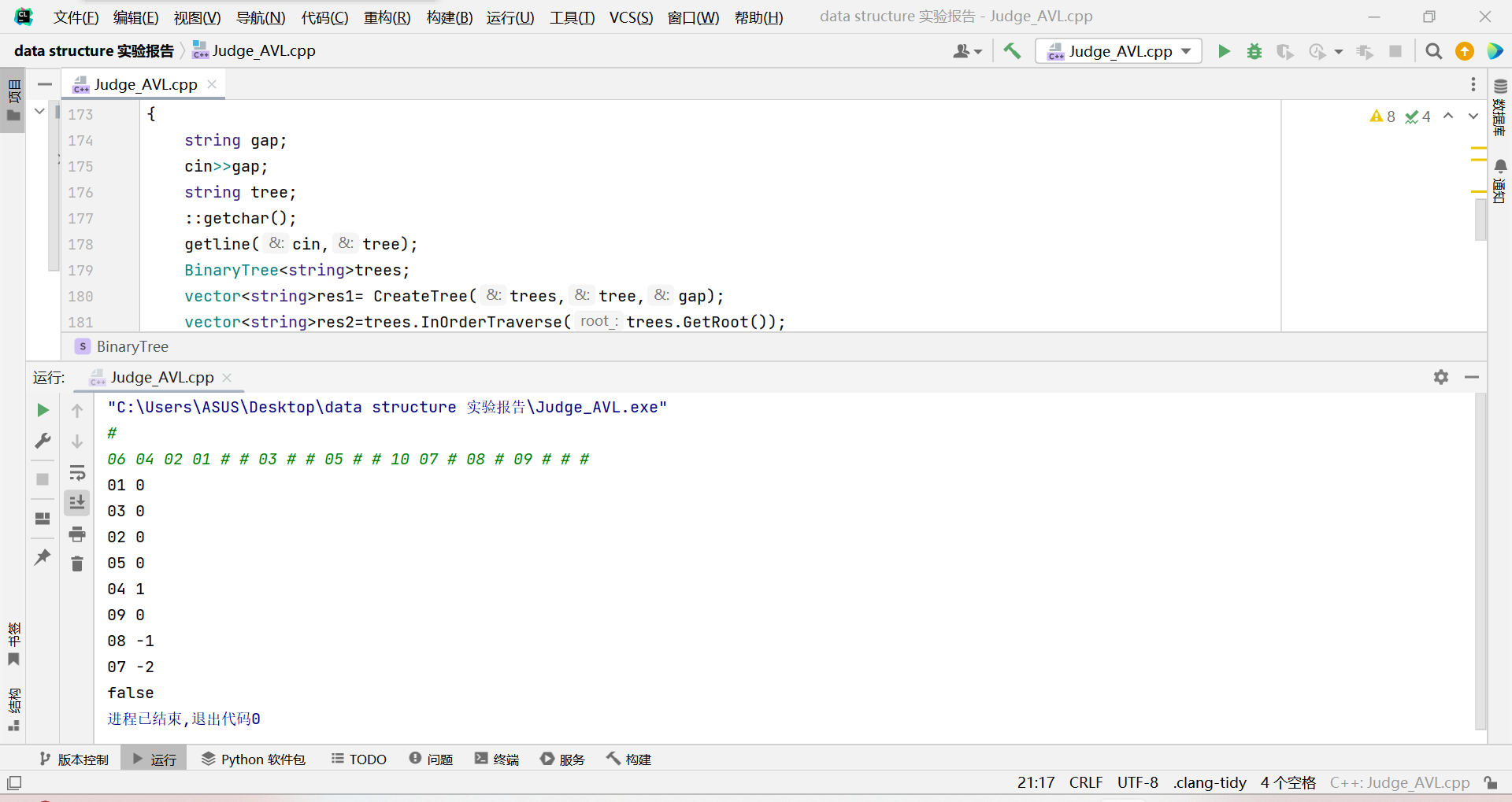
首先，判断当前节点是否是平衡二叉树，则需要开始遍历整棵树，求其左右子树的高度。递归判断其左右子树是否为平衡二叉树，又一次需要遍历其子树求其高度。多次求其高度，越接近叶子节点的节点高度被求的次数越多。时间复杂度是

**3.8.4使用说明**

第一行：表示无孩子或指针为空的特殊分隔符

第二行：二叉树的先序序列（结点元素之间以空格分隔）

程序名称为Judge\_AVL*.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图



**3.8.5 测试结果以及分析**

IN1:普通测试用例，非AVL树

#

06 04 02 01 # # 03 # # 05 # # 10 07 # 08 # 09 # # #

OUT:

01 0

03 0

02 0

05 0

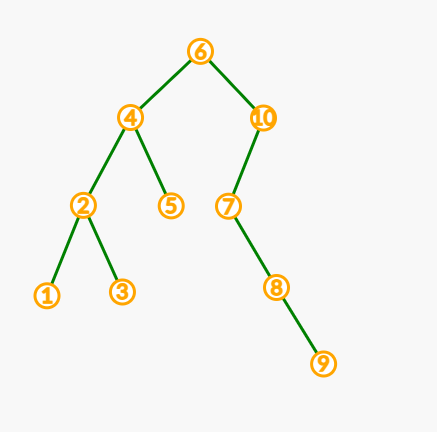
04 1

09 0

08 -1

07 -2

False



INT2:普通测试用例，是AVL树

#

06 04 02 01 # # 03 # # 05 # # 08 07 # # 09 # 10 # #

OUT:

01 0

03 0

02 0

05 0

04 1

07 0

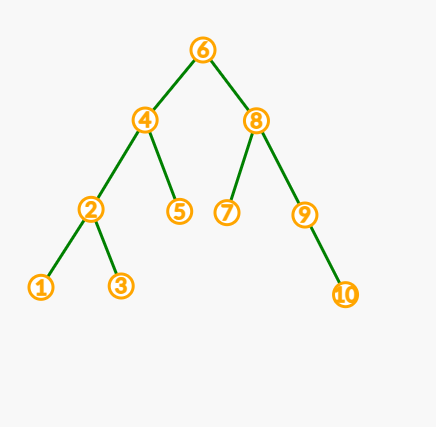
10 0

09 -1

08 -1

06 0

true



IN3:退化成一条线的特殊用例

#

06 05 04 03 02 01 # # # # # # #

OUT:

01 0

02 1

03 2

False



IN4:特殊用例空树

#

#

OUT:

True

IN5:

#

14 08 04 02 01 # # 03 # # 06 05 # # 07 # # 11 09 # 10 # # 12 # 13 # # 20 17 15 14 # # 16 # # 19 18 # 24 22 21 # # 23 # # 26 25 # # #

OUT:

01 0

03 0

02 0

05 0

07 0

06 0

04 0

10 0

09 -1

13 0

12 -1

11 0

08 0

14 0

16 0

15 0

18 0

19 1

17 0

21 0

23 0

22 0

25 0

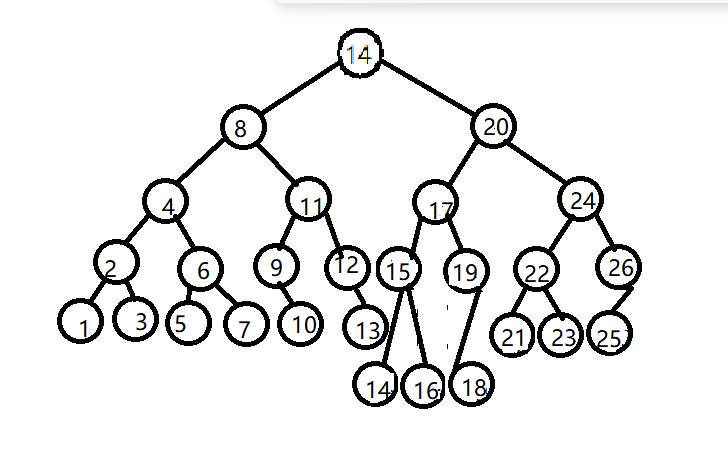
26 1

24 0

20 0

14 0

True



### 3.9二叉树的重建

#### 3.9.1 题目以及要求

**题目描述：**

目的：现有两个结点序列，分别是对同一个二叉树进行前序遍历和中序遍历（或中序和后序）的结果。要求设计一个算法，重构该二叉树，并输出该二叉树按照后序（前序）遍历时的结点序列。假定二叉树所有的结点的数据域的值都不相同。二叉树的存储结构的建立参见二叉树应用1。

一、已知前序和中序的结果，进行二叉树的重构：

提示：*preOrder*按照先根再左再右的顺序递归遍历，*inOrder*按照先左再根再右的顺序递归遍历。

举例说明：*preOrder*的输入*pre={A,B,D,G,H,C,E,I,F}*，*inOrder*的输入*in={G,D,H,B,A,E,I,C,F}*。首先按*preOrder*遍历的顺序依次访问各结点。访问过程中，通过in得知各子树内*inOrder*遍历的顺序，从而重建以当前访问结点*c*为根的左子树与右子树。即：设*preOrder*遍历的当前结点为*c*，*c*在*in*中的位置为*m，m*左侧就是*c*的左子树，右侧就是*c*的右子树。同理递归。

参考函数原型：

（1）//重建二叉树的存储结构 (外壳）  
*template<class ElemType>  
BinaryTreeNode<ElemType> \*ReBuildTree\_Pre\_In(vector[ElemType] &pre, vector[ElemType] &in);*

（2）//重建二叉树的存储结构 (递归）  
*template<class ElemType>  
BinaryTreeNode<ElemType> \*reConstructCore\_Pre\_In(vector[ElemType] &pre, vector[ElemType] &in, int preStart, int preEnd, int inStart, int inEnd );*

二、已知中序和后序的结果，进行二叉树的重构

参考函数原型：

（1）//重建二叉树的存储结构 (外壳）  
*template<class ElemType>  
BinaryTreeNode<ElemType> \*ReBuildTree\_In\_Post(vector[ElemType] &in, vector[ElemType] &post);*

（2）//重建二叉树的存储结构 (递归）  
*template<class ElemType>  
BinaryTreeNode<ElemType> \*reConstructCore\_In\_Post(vector[ElemType] &in, vector[ElemType] &post, int inStart, int inEnd, int postStart, int postEnd );*

**输出说明：**

0：

第一行：按后序遍历的结点序列，相邻结点用","隔开

1：

第一行：按前序遍历的结点序列，相邻结点用","隔开

#### 3.9.2 概要设计

*void dfs\_zq(vector<string> s1,vector<string> s2,int &num)*

函数功能：*s1*储存二叉树的中序遍历序列，*s2*储存二叉树的前续遍历序列，*num*用于标记是否已经过依次输出（控制输出格式），本函数可根据所给的中序和前序遍历序列，得到并输出该二叉树的后续遍历序列。

*void dfs\_zh(vector<string>s1,vector<string> s2,int &num)*

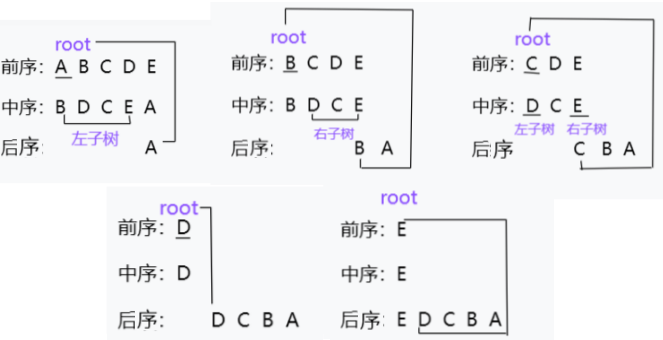
函数功能：*s1*储存二叉树的中序遍历序列，*s2*储存二叉树的后续遍历序列，*num*用于标记是否已经过依次输出（控制输出格式），本函数可根据所给的中序和后序遍历序列，得到并输出该二叉树的前续遍历序列。

#### 3.9.3 算法分析

*算法说明：*首先根据所给的前序或后序遍历序列得到二叉树的根节点，再由中序遍历序列得到根节点的左子树与右子树，将得到的左右子树分别看做一个二叉树进行递归，每次输出一个根节点，当左右子树为空时退出循环，即可得到二叉树的后续或前序遍历序列。

以下图为例:

给出前序和中序时：



给出中序和后续时：

图片包含 图示

描述已自动生成

该算法需要注意的难点:

找到每次从先序和中序或中序和后序序列中抽取出左子树和右子树的通解，并应用于递归算法。

*算法时间复杂度：*每一个元素均作为根节点被返回一次，花费，查询根节点的时间复杂度为，因此总的时间复杂度为*。*

#### 3.9.4 使用说明

**输入说明：**

第一行：重构选择（0：前序中序重构  1：中序后序重构）

0：

第二行：按前序遍历的结点序列，相邻结点用空格隔开

第三行：按中序遍历的结点序列，相邻结点用空格隔开

1：

第二行：按中序遍历的结点序列，相邻结点用空格隔开

第三行：按后序遍历的结点序列，相邻结点用空格隔开

程序名称为*Rebuild\_BinaryTree.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图4.3

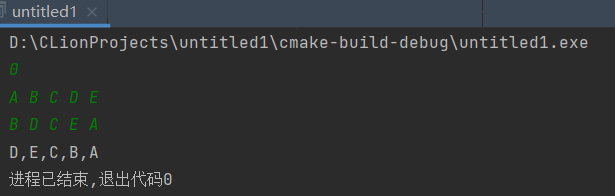


图4.3：运行截图

在输入完数据后,需要使用键盘的*Ctrl+D*(程序内)/*Ctrl+Z*(外部*exe*)即可显示结果。

#### 3.9.5测试结果以及分析

* 第一组数据:树只有一个节点,简图如下:
* 

IN

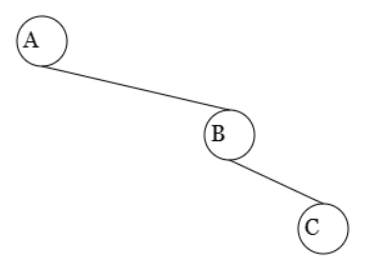
0

A

A

OUT

A

* 第二组数据：树的前序和中序序列相同，简图如下：
* 

IN

0

A B C

A B C

OUT

C B A

* 第三组数据：树为空

IN

1

OUT

* 第四组数据：树的前序和中序序列相同，简图如下：
* 图片包含 游戏机

  描述已自动生成

IN

1

A B C D

A B C D

OUT

D C B A

* 第五组数据：普通树，给出前序和中序遍历序列，简图如下：

图片包含 衣架, 游戏机, 画

描述已自动生成

IN

0

A B C D E F G H

B D C E A F G H

OUT

D E C B H G F A

* 第六组数据：普通树，给出中序和后序遍历序列，简图如下：
* 图示, 形状

  描述已自动生成

IN

1

D B G E H A C I J F

D G H E B J I F C A

OUT

A B D E G H C F I J

### 3.10第 K 个最小的素数分数

#### 3.10.1题目及描述

给你一个按递增顺序排序的数组 *arr* 和一个整数 *k* 。数组 *arr* 由 1 和若干素数组成，且其中所有整数互不相同。

对于每对满足的 *i* 和 *j* ，可以得到分数。

那么第 *k* 个最小的分数是多少呢?

请输出相应的*arr*[*i*]和*arr*[*j*] 。

#### 3.10.2概要设计

设计函数**vector<int>solve(vector<int>&*arr*,int *k*)**负责处理问题及返回结果



#### 3.10.3算法分析

**算法说明:**

个不同的素数可以得到个不同的分数,将其升序排列后,即可得到结果,但是比较分数会损失精度,对此可以将用来判断,储存方面可以使用*pair*来储存分数.

**时间复杂度分析:**

素数分数一共有个,对其进行快速排序,故时间复杂度是.

#### 3.10.4使用说明

输入说明 :

输入三行：

第一行输入一个整数*n*表示数组*arr*的长度。

第二行输入*n*个整数表示数组的元素。

第三行输入一个整数表示*k*.

提示：



*arr*[*i*] 是一个素数 ，*i* > 0

*arr* 中的所有数字互不相同 ，且按严格递增排序



#### 3.10.5测试结果及分析

本题无特殊数据,故可以直接使用随机素数生成器生成,具体代码见下:



* 第一组数据:

IN:

40

1 5 11 13 19 31 43 53 61 73 89 97 101 103 107 113 137 139 157 167 181 197 211 227 239 241 257 269 277 281 307 317 337 353 359 367 373 379 383 389

186

OUT:

53 373

* 第二组数据:

IN:

30

1 23 31 43 89 137 149 163 223 257 283 317 347 349 353 383 401 443 487 563 617 619 653 719 757 809 829 863 937 1009

309

OUT:

347 563

* 第三组数据:

IN:

36

1 5 41 71 101 149 179 233 257 293 331 359 419 443 479 547 601 607 613 673 691 761 829 863 911 977 1009 1031 1033 1039 1087 1151 1153 1201 1249 1301

273

OUT:

359 1009

* 第四组数据:

36

1 131 269 571 773 911 1019 1091 1117 1171 1289 1571 1627 1861 2081 2267 2609 2861 3083 3167 3169 3469 3727 3889 4133 4201 4339 4639 4877 5003 5197 5501 5503 5849 6079 6151

290

OUT:

1171 3083

* 第五组数据:

50

1 43 59 127 347 379 563 709 1049 1151 1279 1451 1753 2087 2267 2273 2293 2617 2713 2879 3137 3229 3499 3617 3671 3701 3797 3907 4261 4481 4561 4621 4813 5261 5471 5717 5807 5879 6257 6343 6521 6581 6961 7307 7589 8039 8167 8179 8423 8501

23

OUT:

1 3907

* 第六组数据:

IN:

70

1 11 197 487 683 727 887 1223 1583 1657 1823 2129 2207 2251 2399 2753 3089 3301 3623 3691 4007 4079 4253 4283 4597 4871 5227 5483 5527 5743 5927 6007 6359 6421 6607 6673 6737 6841 6869 7247 7549 7841 8101 8179 8243 8329 8363 8539 8741 9127 9391 9643 9719 10007 10273 10723 10733 11093 11251 11551 11807 12007 12011 12281 12289 12539 12739 12791 13127 13183

1645

OUT:

6421 9643

* 第七组数据:

IN:

100

1 211 419 739 937 1213 1447 1453 1481 1753 1783 2081 2389 2711 2731 2837 2909 3343 3371 3719 3803 3931 4271 4507 4721 4999 5323 5519 5569 5591 5881 5927 6173 6529 6947 7127 7457 7549 7793 8009 8243 8663 8867 9011 9377 9629 10039 10457 10781 10937 10949 11411 11497 11833 11933 12157 12541 12641 12781 13171 13451 13627 13799 14107 14431 14831 15077 15161 15619 15859 16253 16301 16411 16747 16759 17033 17327 17341 17539 17581 17977 18269 18341 18553 18701 19267 19709 19843 20147 20287 20759 20807 21001 21227 21701 21977 22073 22541 22543 22937

2445

OUT:

5591 12541

### 3.11 LRU缓存

#### 3.11.1 题目以及要求

**题目描述:**

请你设计并实现一个满足 *LRU* (最近最少使用) 缓存 约束的数据结构。

实现 *LRUCache* 类：

*LRUCache*(*int* *capacity*) 以 正整数 作为容量 *capacity* 初始化 *LRU* 缓存

*int* *get*(*int* *key*) 如果关键字 *key* 存在于缓存中，则返回关键字的值，否则返回 -1 。

*void* *put*(*int* *key*, *int* *value*) 如果关键字 *key* 已经存在，则变更其数据值 *value* ；如果不存在，则向缓存中插入该组 *key*-*value* 。如果插入操作导致关键字数量超过 *capacity* ，则应该逐出 最久未使用的关键字。

#### 3.11.2 概要设计

设计图如下:



#### 3.11.3算法分析

**算法说明:**

定义*start*变量,初始时是0,不论进行什么操作,其都会自增一次,以使得*map*可以根据时间排序

* 在设计*put*函数时,需要考虑三种情况:
  + 如果*key*已经存在在表中(使用*map*的成员函数*count*来进行),先要从表中移出(调用*erase*即可),再将*start++*,之后重新建立映射即可
  + 如果*key*不存在于表中但是表的容量已经到达*capacity*则需要移出头部元素(因为*map*的排序机制使得头部元素一定是时间最早的),之后重新建立映射即可
  + 普通的情况,直接插入建立映射即可,无需多余的操作
* 对于*get*函数,有两种情况,第一种是查询不到这个元素,直接输出-1即可,否则先输出其值,再将其拿出来(存到变量后从表中*erase*),更新时间后重新插回
* 由于*map*中需要用到两个*pair*的比较,那么需要重载*pair*的小于运算符,如下:

**bool operator <(pair<int,int>*a1*,pair<int,int>*a2*)  
{  
 return *a1*.first<*a2*.first;  
}**

**时间复杂度分析:**

*Map*的单次查询,插入,删除均为,故总时间复杂度为,如果想要达到的时间复杂度可以尝试使用,但其要求重载*pair*的哈希函数,较为麻烦,故本题小组不使用这种方法,对于*pair*的重载哈希,小组将在最后一题进行尝试.

**3.11.4使用说明**

输入若干行：

第一行输入一个整数*capacity*表示LRU缓存的容量。

后面每行输入为*put*或*get*：

如果指令为*put*类型，后面需要输入两个整数表示*key*和*value*。

如果指令为*get*类型，后面需要输入一个整数表示*key*。

提示：



最多调用次 *get* 和 *put*

程序名称为11*.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图

图形用户界面, 应用程序

描述已自动生成

#### 3.11.5测试结果及分析

* 第一组数据:特殊数据,*capacity*为1,且保证输入的*key*不重复

IN:

1

put 1 2

get 1

put 3 4

put 5 6

get 2

get 3

get 4

put 7 8

OUT：

2

-1

-1

-1

* 第二组数据:拥有重复的*key*,测试更新是否正常

IN:

5

get 2

put 1 2

get 1

put 1 4

get 1

put 2 3

get 2

get 1

put 1 8

put 2 6

get 2

get 1

put 3 7

get 3

put 3 8

put 3 9

get 3

put 4 5

put 5 6

put 7 7

put 7 9

get 7

get 6

get 5

get 4

get 3

get 2

get 1

OUT:

-1

2

4

3

4

6

8

7

9

9

-1

6

5

9

-1

8

* 第三组数据,只有*get*

IN:

5

get 2

get 3

get 4

get 5

OUT:

-1

-1

-1

-1

* 第四组数据:普通数据,*capacity*比较小,会多次修改其中的某个*key*,这里被选中的*key*是8695以便后期检查

20

get 1004

put 8695 91383

put 5705 22739

get 2946

put 3329 53040

get 4701

put 6066 99957

get 7482

get 104

get 740

put 6262 56122

put 7247 13114

get 8695

put 120 33528

put 6265 35065

put 9692 63098

put 5682 6807

get 6262

get 6066

put 7823 28233

get 6553

put 4432 73993

put 8695 25356

put 1056 95212

put 7673 26524

get 4432

put 8695 94585

get 5068

get 5697

get 8695

get 9759

get 7521

put 146 70332

get 5757

put 2976 61751

put 8362 47634

get 5427

get 6262

put 4277 59208

put 2976 23445

get 4091

get 1276

put 7355 32041

put 2874 33853

put 8695 55337

get 7355

put 4866 94247

get 5632

put 6958 72048

get 9129

put 7040 10619

put 5946 23277

get 8695

put 1687 81940

get 2874

get 2495

get 2885

get 1687

put 1425 75639

get 4484

put 1029 47735

put 367 9858

get 8511

put 1039 33494

put 7223 3638

get 5314

put 9980 70534

get 4617

get 8695

put 7713 61187

put 6392 80409

put 3905 61794

get 7713

get 3968

put 9456 13888

get 9456

OUT:

-1

-1

-1

-1

-1

-1

91383

56122

99957

-1

73993

-1

-1

94585

-1

-1

-1

-1

56122

-1

-1

32041

-1

-1

55337

33853

-1

-1

81940

-1

-1

-1

-1

55337

61187

-1

13888

* 第五组数据,7500次操作大数据测试,测试时间复杂度,输入数据保证有重复的*key*

IN:由于输入量较大,这里在word中嵌入了txt文档保持美观(双击即可查看)



OUT:



**说明:小组使用了如下代码生成随机的数据,相比纯随机的数据,这串代码会使得产生的数据更有意义,仅有极小的概率会产生全是*put*,*get*,或者*keys*全不相同的情况**

**std::random\_device *rd*;  
std::uniform\_int\_distribution<int>*putting\_key*(0,10000);  
std::uniform\_int\_distribution<int>*putting\_value*(0,100000);  
vector<int>*keys*;  
std::mt19937\_64 *gen*(*rd*()+time(0));  
int *capacity*=20;  
cout<<*capacity*<<endl;  
int *k*=0;  
for(int *i*=0;*i*<7500;*i*++)  
{  
 if(*k*%2==0)  
 {  
 int *my\_key*=*putting\_key*(*gen*);  
 cout<<"put"<<" "<<*my\_key*<<" "<<*putting\_value*(*gen*)<<endl;  
 *keys*.push\_back(*my\_key*);  
 }  
 else  
 {  
 if(*k*<=3000)  
 cout<<"get"<<" "<<*putting\_key*(*gen*)<<endl;  
 else  
 {  
 cout<<"get"<<" "<<*keys*[*putting\_key*(*gen*)%*keys*.size()]<<endl;*//保证随机样例中有之前插入进去的key* }  
 }  
 *k*=*putting\_key*(*gen*);  
}**

### 3.12 设计推特

#### 3.12.1 题目及其要求

设计一个简化版的推特(Twitter)，可以让用户实现发送推文，关注/取消关注其他用户，能够看见关注人（包括自己）的最近 10 条推文。

实现 Twitter 类：

Twitter() ：初始化简易版推特对象

*void postTweet(int userId, int tweetId)* ：根据给定的 *tweetId* 和 *userId* 创建一条新推文。每次调用此函数都会使用一个不同的 *tweetId* 。

*vector<int> getNewsFeed(int userId)* ：检索当前用户新闻推送中最近 10 条推文的 ID 。新闻推送中的每一项都必须是由用户关注的人或者是用户自己发布的推文。推文必须 按照时间顺序由最近到最远排序 。

*void follow(int followerId, int followeeId)*:ID为 *followerId* 的用户开始关注 ID 为 *followeeId* 的用户。

*void unfollow(int followerId, int followeeId)*:ID为*followerId*用户不再关注 ID 为 *followeeId* 的用户。

#### 3.12.2 概要设计

**设计图如下**:



#### 3.12.3 算法分析

**算法说明**:

* 对于*postTweet*方法只需要调用*set<news>*的*insert*方法即可完成自动插入,自动排序
* 对于*getNewsFeed*方法,需要遍历整个*pos* 集合,对于其中的每一个结构体元素的*name*,需要判断其名称是否为自身,或者在*followers* *map*中查找是否为关注的人(调用*find*方法),另外需要判断如果超过10个需要退出循环
* 对于*follow*方法只需要将*followeeId*其push\_back到队列末端即可
* 对于*unfollow*方法,需要调用*find*在队列中找到被取关者,随后删除
* 需要用到排序,需要重载结构体的小于运算符使其根据时间降序排列,如下:

bool operator < (news *x* ,news *y*)  
{  
 return *x*.time>*y*.time;  
}

**时间复杂度分析:**

postTweet方法调用set的insert函数,单次插入

**void postTweet(int *userId*,int *tweetId*)  
{  
 pos.insert({*userId*,++Time,*tweetId*});  
}**

getNewsFeed方法外层循环遍历整个set用时,内层循环需要在中查找遍历其所在的队列确认是否为其关注者用时,故总时间复杂度为

**for(auto *it*=pos.begin();*it*!=pos.end();*it*++)  
{ if(*k*>=10){return *res*;}  
else{ if(*it*->name==*userId*||find(followers[*userId*].begin(),followers[*userId*].end(),*it*->name)!=followers[*userId*].end())  
 { *k*++;*res*.push\_back(*it*->post);}}}**

Follow方法,单次尾插deque用时

Unfollow方法,map中查找followerId用时,deque中查找followeeId用时,deque中调用erase用时,故单次取关操作为

**void unfollow(int *followerId*,int *followeeId*)  
{followers[*followerId*].erase(find(followers[*followerId*].begin(),followers[*followerId*].end(),*followeeId*));}**

#### 3.12.4 使用说明

输入若干行

每行输入为*postTweet*、*getNewsFeed*、*follow*或者*unfollow*：

如果指令为*postTweet*类型，后面需要输入两个整数表示*tweetId* 和 *userId*

如果指令为*getNewsFeed*类型，后面需要输入一个整数表示*userId*

如果指令为*follow*类型，后面需要输入两个整数*followerId*和*followeeId*

如果指令为*unfollow*类型，后面需要输入两个整数*followerId*和*followeeId*

提示：



所有推特的 ID 都互不相同

*postTweet、getNewsFeed、follow* 和 *unfollow* 方法最多调用次

程序名称为12*.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图



#### 3.12.5 测试结果及分析

* 第一组数据:保证不会有unfollow操作

IN:

postTweet 3 2

postTweet 3 3

postTweet 3 1

postTweet 3 4

getNewsFeed 3

follow 3 4

postTweet 4 5

getNewsFeed 3

OUT:

4 1 3 2

5 4 1 3 2

* 第二组数据:不会有*postTweet*操作,测试*getNewsFeed* 返回空对象是否正常.

IN:

follow 4 5

follow 4 7

follow 4 8

follow 4 9

unfollow 4 9

getNewsFeed

OUT:

\

* 第三组数据,会*unfollow*不在关注列表里的人,测试鲁棒性

IN：

postTweet 1 2

postTweet 2 3

postTweet 3 4

follow 1 2

follow 1 3

unfollow 1 4

getNewsFeed 1

OUT:

3 2

* 第四组数据:普通数据,包含一个用户拥有超过10条的新闻推送

IN：

postTweet 5 6

postTweet 5 111

postTweet 5 1231

postTweet 5 21312

postTweet 5 1211

postTweet 9 11231221

postTweet 9 12

postTweet 9 13

postTweet 9 1

postTweet 9 2

postTweet 8 3

follow 5 9

follow 5 8

getNewsFeed 5

postTweet 5 66666

follow 9 8

follow 9 5

getNewsFeed 9

unfollow 9 5

getNewsFeed 9

OUT:

3 2 1 13 12 11231221 1211 21312 1231 111

66666 3 2 1 13 12 11231221 1211 21312 1231

3 2 1 13 12 11231221

* 第五组数据,普通数据,随机测试,且会*unfollow*没有关注的人,为了结果美观,保证前10次调用都是*getNewsFeed*

IN:

postTweet 8 119128

postTweet 3 47252

postTweet 3 67441

postTweet 6 17887

postTweet 2 79465

postTweet 5 40334

postTweet 7 129982

postTweet 1 171731

postTweet 2 163559

postTweet 4 32573

getNewsFeed 8

unfollow 5 4

getNewsFeed 2

unfollow 5 1

postTweet 5 9183

follow 8 2

follow 7 3

follow 8 3

getNewsFeed 2

follow 9 6

unfollow 4 8

unfollow 8 6

getNewsFeed 5

unfollow 6 7

postTweet 4 49636

getNewsFeed 6

follow 8 1

unfollow 6 5

unfollow 5 6

follow 6 5

follow 1 8

postTweet 1 154873

postTweet 5 72328

unfollow 6 4

postTweet 6 82327

postTweet 1 73847

postTweet 5 29679

getNewsFeed 2

unfollow 5 6

postTweet 9 101832

follow 1 3

postTweet 8 77093

getNewsFeed 9

postTweet 1 98235

follow 2 9

postTweet 4 186638

follow 7 8

postTweet 3 38588

follow 4 9

postTweet 6 34894

postTweet 8 169601

getNewsFeed 3

follow 5 4

unfollow 3 4

getNewsFeed 5

getNewsFeed 3

getNewsFeed 1

unfollow 1 6

postTweet 7 161612

postTweet 3 28649

getNewsFeed 4

getNewsFeed 8

getNewsFeed 4

unfollow 1 7

getNewsFeed 5

getNewsFeed 1

postTweet 3 62590

follow 4 7

unfollow 5 6

follow 7 5

getNewsFeed 3

getNewsFeed 1

postTweet 4 149463

getNewsFeed 3

postTweet 7 74373

getNewsFeed 1

unfollow 3 2

postTweet 4 46038

postTweet 4 40218

postTweet 1 61741

unfollow 3 7

postTweet 6 95215

getNewsFeed 8

postTweet 8 102729

follow 5 9

follow 3 8

follow 8 7

postTweet 3 144165

postTweet 6 27762

unfollow 5 2

follow 4 8

getNewsFeed 8

unfollow 1 7

postTweet 1 101409

follow 4 1

getNewsFeed 6

postTweet 5 61324

getNewsFeed 6

postTweet 2 178148

postTweet 7 190079

OUT:

119128

163559 79465

163559 79465

9183 40334

17887

163559 79465

101832 82327 17887

38588 67441 47252

186638 29679 72328 49636 9183 32573 40334

38588 67441 47252

169601 38588 98235 77093 73847 154873 171731 67441 47252 119128

186638 101832 49636 32573

28649 169601 38588 98235 77093 73847 154873 163559 171731 79465

186638 101832 49636 32573

186638 29679 72328 49636 9183 32573 40334

28649 169601 38588 98235 77093 73847 154873 171731 67441 47252

62590 28649 38588 67441 47252

62590 28649 169601 38588 98235 77093 73847 154873 171731 67441

62590 28649 38588 67441 47252

62590 28649 169601 38588 98235 77093 73847 154873 171731 67441

61741 62590 28649 169601 38588 98235 77093 73847 154873 163559

144165 102729 61741 74373 62590 28649 161612 169601 38588 98235

27762 95215 34894 29679 82327 72328 9183 40334 17887

61324 27762 95215 34894 29679 82327 72328 9183 40334 17887

* 第六组数据:大数据测试5000次调用,确保随机性,输入量较大,在Word中以txt嵌入,**双击即可打开**

IN:



OUT:



**注:小组使用以下代码生成本题样例,相比纯随机数的生成,该代码生成的每一次样例定能符合题目条件限制,且确保四个函数调用平均,也可以灵活的通过修改参数的方式得到符合预期的样例;**

**#include <bits/stdc++.h>  
#include<random>  
using namespace std;  
int main() {  
 std::random\_device *rd*;map<int, deque<int>> *follows*;  
 std::uniform\_int\_distribution<int> *USER\_FOLLOWER\_FOLLOWEE\_ID\_GEN*(1, 500);  
 std::uniform\_int\_distribution<int> *TWEETID*(1, 200000);  
 vector<long long> *check*;  
 std::uniform\_int\_distribution<int> *CHOOSE*(1, 2000); std::mt19937\_64 *gen*(*rd*() + time(0));  
 int *choose* = *CHOOSE*(*gen*);  
 for (int *i* = 0; *i* < 5000; *i*++) {  
 if (*choose* % 4 == 0){int *tweetID* = *TWEETID*(*gen*);int *userID* = *USER\_FOLLOWER\_FOLLOWEE\_ID\_GEN*(*gen*);  
 while (find(*check*.begin(), *check*.end(), *tweetID*) != *check*.end()) {*tweetID* = *TWEETID*(*gen*);*//确保tweetID不重复*} *check*.push\_back(*tweetID*);  
 cout << "postTweet" << " " << *userID* << " " << *tweetID* << " " << endl;}else if (*choose* % 4 == 1) {cout << "getNewsFeed " << *USER\_FOLLOWER\_FOLLOWEE\_ID\_GEN*(*gen*) << endl;} else if (*choose* % 4 == 2) {  
 int *followerId* = *USER\_FOLLOWER\_FOLLOWEE\_ID\_GEN*(*gen*);  
 int *followeeId* = *USER\_FOLLOWER\_FOLLOWEE\_ID\_GEN*(*gen*);  
 while (*followerId* == *followeeId* ||find(*follows*[*followerId*].begin(), *follows*[*followerId*].end(), *followeeId*) !=*follows*[*followerId*].end()) {*followeeId* = *USER\_FOLLOWER\_FOLLOWEE\_ID\_GEN*(*gen*);*//确保id不同,且不重复添加*}  
 *follows*[*followerId*].push\_back(*followeeId*);  
 cout << "follow" << " " << *followerId* << " " << *followeeId* << endl;  
 } else if (*choose* % 4 == 3) { int *followerId* = *USER\_FOLLOWER\_FOLLOWEE\_ID\_GEN*(*gen*);  
 int *followeeId* = *USER\_FOLLOWER\_FOLLOWEE\_ID\_GEN*(*gen*);while (*followerId* == *followeeId*) { *followeeId* = *USER\_FOLLOWER\_FOLLOWEE\_ID\_GEN*(*gen*);*//确保id不同* } cout << "unfollow" << " " << *followerId* << " " << *followeeId* << endl; }*choose* = *CHOOSE*(*gen*);  
 }  
 return 0;  
}**

### 3.13 单线程 CPU

#### 3.13.1题目及其要求

给你一个二维数组*tasks* ,用于表示 *n* 项从 0 到 *n* - 1 编号的任务.其中意味着第 *i* 项任务将会于 *enqueueTimei* 时进入任务队列，需要 *processingTimei*的时长完成执行。

现有一个单线程 CPU ,同一时间只能执行 最多一项 任务，该 CPU 将会按照下述方式运行：

如果 CPU 空闲，且任务队列中没有需要执行的任务，则 CPU 保持空闲状态。

如果 CPU 空闲，但任务队列中有需要执行的任务，则 CPU 将会选择 执行时间最短的任务开始执行。如果多个任务具有同样的最短执行时间，则选择下标最小的任务开始执行。

一旦某项任务开始执行，CPU 在 执行完整个任务 前都不会停止。

CPU 可以在完成一项任务后，立即开始执行一项新任务。

返回 CPU 处理任务的顺序。

#### 3.13.2概要设计

**设计图如下:**

****

#### 3.13.3算法分析

**算法说明:**

很容易理解,在读入数据时,我们需要按照等待时间,执行时间,序号的优先级对数据进行排序,但是在处理数据的时候,可能会因为每个任务的执行时间长短,导致原先的顺序会再次发生变化,故我们需要设计两个集合为我们处理不同阶段的工作,其中集合会保证当前的时间大于等于其放入任务的等待时间,故只需要按照进行排序即可.我们的目标是将集合中的每一个元素搬运到集合中并输出,为了体现时间的流失,我们需要额外设计一个游标,初始时可以直接快进到中的首元素的等待时间,将其放入集合,对于之后的每一步操作,输出的永远是集合中的头部元素,并将游标增加其的,在每一遍循环开始前,将已经过了等待时间的集合中的元素取出放入集合,周而复始,即可完成全部的输出,由于我们一次循环只输出一个集合中的元素,故当集合为空时,集合可能会有剩余,故最后将集合中的剩余元素输出即可.

重载运算符的方法与前几次实验报告雷同,这里省略.

int *Time* = *CPU*.begin()->queue\_time;*//开始时间直接快进到第一个事件的等待时间*std::set<c>::iterator *it* = *CPU*.begin();  
while (*it* != *CPU*.end()) { while (*it* != *CPU*.end() && *Time* >= *it*->queue\_time) {*go*.insert({*it*->index\_, *it*->wait\_time}); *it*++;}  
 if (!*go*.empty()) {cout << *go*.begin()->index\_ << " ";*//输出第一个  
 Time* += *go*.begin()->wait\_time; *go*.erase(*go*.begin());} else {  
 if (++*it* != *CPU*.end())*//判断一下是不是越界了* {*Time* = *it*->queue\_time;} else {break;}} }  
while (!*go*.empty()) {  
 cout << *go*.begin()->index\_ << " ";*//输出第一个  
 Time* += *go*.begin()->wait\_time;  
 *go*.erase(*go*.begin());  
}

**时间复杂度分析:**

遍历整个集合,将元素搬运至集合每次调用删除与插入操作,单次操作,故总时间复杂度为

#### 3.13.4使用说明

输入若干行：

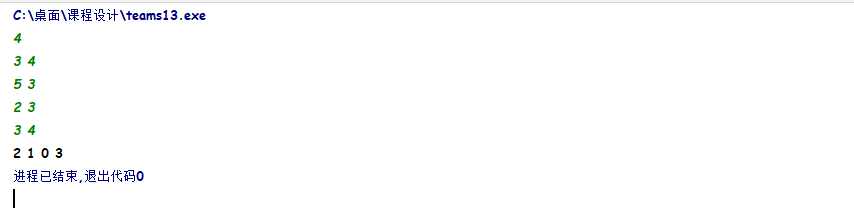
第一行为一个整数*n*代表二维数组*tasks*的行数。

后面*n*行，每行输入两个整数代表*enqueueTimei*和*processingTime*i

提示：



程序名称为13*.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图



#### 3.13.5测试结果及分析

* 第一组数据:输入数据保证是有序的,CPU按照输入的顺序执行,特殊用例

IN:

10

1 1

2 1

3 1

4 1

5 1

6 1

7 1

8 1

9 1

10 1

OUT:

0 1 2 3 4 5 6 7 8 9

* 第二组数据:输入的*queueTime*是无序的,但是其*waitTime*会保证其输出的结果是有序的

IN:

10

1 10

2 20

3 30

6 40

7 50

10 60

9 70

8 80

5 90

4 100

OUT:

0 1 2 3 4 5 6 7 8 9

* 第三组数据:有多个任务具有相同的*queueTime*

IN:

10

4 10

4 2

3 1

4 8

2 9

7 77

2 11

10 2

1 1

2 1

OUT:

8 9 2 1 3 7 4 0 6 5

* 第四组数据:多个任务在*queueTime*相同的情况下会有相同的*waitTime*

IN:

15

4 10

4 2

3 1

4 10

2 9

7 77

2 11

10 2

1 1

2 11

7 6

1 2

3 9

4 2

4 5

OUT:

8 11 2 1 13 14 7 10 4 12 0 3 6 9 5

* 第五组数据:普通数据,数据随机

IN:

34

77 19

100 56

24 72

42 33

11 62

63 34

39 92

69 89

41 1

25 71

98 77

82 25

16 56

91 98

53 17

77 57

25 28

37 40

82 4

11 62

72 35

8 76

62 22

88 58

84 73

98 12

34 45

77 86

14 78

16 79

55 47

35 26

88 93

85 68

OUT:

21 8 18 14 25 0 22 11 31 16 3 5 20 17 26 30 1 12 15 23 4 19 33 9 2 24 10 28 29 27 7 6 32 13

* 第六组数据:大数据检验

IN:

C:\桌面\课程设计\teams13.exe

2000

675 394

56 870

709 828

768 601

830 175

616 362

906 559

39 266

888 658

361 617

983 564

519 499

939 368

140 990

85 292

901 253

672 868

930 440

10 126

124 255

696 439

815 193

613 827

190 787

266 143

752 219

558 788

404 21

438 48

394 149

93 873

569 418

544 238

919 175

157 708

57 355

798 468

749 797

290 297

884 845

553 432

935 41

224 615

262 908

671 635

190 142

998 100

413 910

189 95

221 880

574 750

956 682

472 492

395 975

139 477

499 152

472 468

105 922

100 962

634 19

1 775

445 818

342 652

679 870

95 42

788 841

207 421

828 216

850 652

624 733

687 747

766 956

781 406

645 837

327 956

197 937

440 101

394 718

346 626

11 617

143 423

920 503

913 363

595 227

453 277

646 463

557 272

164 756

964 401

293 498

486 131

834 222

395 518

292 278

32 957

272 533

850 609

183 146

12 47

46 933

205 945

980 308

134 378

641 590

869 145

226 19

238 126

586 992

590 470

939 282

431 776

768 95

188 624

24 339

986 193

54 986

436 292

213 555

604 109

889 301

257 493

595 552

962 931

35 457

645 85

437 405

694 633

770 851

863 854

567 537

21 644

679 13

806 947

47 163

352 353

726 962

550 311

512 90

784 247

937 773

496 270

918 358

125 352

332 371

932 653

173 531

621 183

27 321

272 372

41 859

585 816

210 572

242 12

479 670

885 81

687 711

318 618

358 841

670 1

761 447

797 672

465 686

78 244

795 679

510 791

787 342

630 721

575 269

69 317

922 639

726 935

352 189

251 976

763 171

12 798

916 453

421 265

205 751

256 508

225 613

697 722

610 107

578 839

553 246

614 635

155 162

264 437

88 797

332 749

299 58

237 556

497 310

936 724

506 538

793 946

443 346

818 19

478 693

739 340

320 202

74 506

649 543

802 96

828 883

46 569

410 763

938 942

179 829

903 975

416 849

460 396

980 394

654 39

714 741

114 518

425 223

335 317

546 624

714 910

43 262

352 277

23 146

683 934

770 378

653 615

929 163

729 240

724 265

192 293

163 260

779 49

516 456

237 537

84 54

256 316

227 455

191 313

196 547

851 310

266 677

89 588

272 718

56 889

286 176

298 841

955 895

58 451

412 371

215 187

883 90

258 482

334 794

205 561

134 506

190 487

186 128

864 369

511 138

423 628

521 100

179 272

554 29

923 644

397 885

922 948

637 533

100 751

152 536

552 115

857 341

488 971

148 325

929 133

67 717

841 597

49 93

495 263

257 76

140 51

572 489

976 340

75 995

737 959

952 806

655 565

369 715

937 943

713 181

339 756

106 54

605 507

891 955

330 98

405 265

985 86

60 737

91 310

710 725

543 575

555 793

529 942

990 594

980 435

164 148

132 982

219 223

284 433

271 338

176 74

951 490

704 621

113 375

150 743

418 243

94 375

605 798

225 456

752 382

212 569

888 607

461 585

994 239

575 880

291 926

249 36

283 983

57 468

70 936

189 306

479 505

822 802

411 349

409 797

795 114

790 29

535 346

971 360

498 914

35 637

684 37

117 504

737 543

162 222

809 307

235 548

537 692

488 908

345 469

345 493

617 357

192 898

52 908

192 421

324 278

683 908

546 388

390 538

281 596

924 290

945 802

531 115

438 114

762 685

209 74

346 718

772 401

236 45

722 256

14 187

9 357

839 759

43 548

553 469

944 259

815 169

605 2

221 793

340 451

402 335

55 666

560 444

912 186

676 666

176 846

342 805

712 62

937 216

30 430

988 138

301 216

507 584

829 809

42 139

761 729

286 569

813 50

675 622

874 435

120 256

596 559

359 315

667 413

591 701

211 46

161 692

920 699

179 891

364 690

764 340

957 847

684 193

991 696

523 471

46 181

379 592

231 824

226 158

814 637

180 902

177 223

373 554

88 51

985 348

252 56

312 699

549 482

295 134

181 200

319 76

200 785

629 665

51 54

198 934

750 276

347 33

384 285

231 656

576 839

586 264

120 12

704 833

682 30

728 462

28 94

812 804

737 925

211 810

16 136

754 14

654 556

415 332

583 273

651 559

493 18

859 194

77 441

153 465

499 619

958 306

163 773

413 41

704 744

287 310

339 502

580 195

754 498

512 605

578 379

459 844

430 919

262 825

617 82

177 966

946 323

778 754

474 609

504 281

867 362

329 19

514 157

958 230

937 409

318 421

295 418

233 38

426 807

456 26

859 890

91 556

730 344

851 896

240 157

644 230

76 450

619 880

717 781

395 101

165 928

700 586

186 581

423 85

430 242

321 294

117 920

295 98

516 586

919 330

761 402

263 914

493 87

450 675

3 302

199 306

688 416

881 695

508 821

460 571

101 382

646 263

266 313

471 976

209 22

264 12

905 229

951 76

937 749

546 586

287 118

922 750

456 797

926 521

611 324

914 800

61 908

375 190

689 415

876 180

666 529

165 423

200 971

396 643

271 577

309 385

85 482

151 791

537 320

308 340

487 157

875 767

256 291

5 557

232 241

457 910

787 71

654 401

586 978

794 213

8 991

809 828

898 91

855 540

160 666

110 47

462 153

916 295

799 304

548 68

788 246

192 658

207 230

571 103

213 492

487 357

636 126

436 442

882 293

453 322

752 256

425 170

377 641

797 986

595 461

587 725

188 178

421 722

141 24

475 337

600 613

103 928

632 33

196 479

504 788

723 437

2 598

442 811

354 8

344 345

716 79

736 823

791 903

734 490

247 41

918 766

811 242

788 722

221 502

872 746

185 257

759 291

622 280

116 551

12 105

797 372

484 267

635 5

82 468

348 929

557 477

219 902

213 508

450 872

921 723

918 16

136 774

321 279

820 197

735 585

474 558

816 532

590 781

979 324

1000 820

505 410

689 628

921 451

4 328

766 82

485 830

808 533

287 633

453 359

67 51

464 70

184 388

883 578

230 509

341 430

159 650

329 580

286 917

498 319

482 490

227 949

732 261

885 322

14 495

974 344

299 756

492 189

565 855

847 485

928 259

880 240

809 122

383 957

419 900

164 495

375 477

825 780

325 991

587 494

303 632

886 639

947 617

670 865

800 699

361 859

217 860

305 803

33 183

188 206

170 297

982 203

343 826

846 254

773 888

692 539

823 877

539 264

423 648

64 259

877 804

979 364

164 519

376 169

240 817

33 556

693 908

791 797

554 944

401 559

466 133

658 199

192 938

692 53

527 576

508 525

713 371

7 175

617 140

378 1

128 67

766 284

517 257

287 495

194 233

760 950

785 17

68 140

775 826

475 530

750 57

870 233

681 369

823 129

13 398

651 223

686 775

276 417

658 154

926 462

631 376

324 37

169 302

433 603

461 61

847 217

602 65

737 817

917 29

338 963

51 451

366 244

341 492

488 3

582 103

252 971

348 823

177 320

928 186

794 13

860 948

297 549

809 937

50 657

746 557

433 242

678 42

607 998

654 308

145 215

967 746

42 803

559 964

876 196

229 122

564 780

923 529

669 937

889 803

978 737

843 856

392 5

254 522

292 991

249 491

739 434

468 17

836 267

518 761

622 103

881 44

502 570

899 52

95 465

529 96

342 143

986 397

439 685

446 953

399 481

605 21

531 69

182 565

693 298

381 302

289 401

355 356

366 765

136 121

957 646

602 33

85 960

706 860

401 470

82 675

744 960

378 906

733 545

78 899

595 473

15 595

339 339

73 193

981 964

992 348

181 174

84 377

263 873

798 733

725 600

671 97

68 818

131 938

135 164

656 118

288 598

292 671

460 707

521 700

9 360

271 353

81 553

517 209

539 241

405 119

659 970

320 118

861 1

318 705

946 244

394 812

958 538

288 832

820 720

247 190

632 933

435 581

381 771

195 756

589 519

687 821

229 31

920 918

953 705

175 657

8 314

607 723

610 705

347 332

309 441

15 337

614 318

743 407

515 958

19 761

533 442

403 1000

660 438

526 148

377 839

973 424

147 42

684 592

858 819

558 882

301 433

8 36

878 962

746 513

511 16

594 405

105 560

614 221

795 758

331 24

553 768

949 257

406 639

166 132

992 120

375 441

255 24

105 707

388 429

676 629

782 16

677 642

241 516

224 997

231 631

891 311

347 667

164 809

649 677

986 913

783 768

512 954

985 103

586 413

783 483

960 200

922 279

413 297

26 947

626 333

181 12

247 6

403 21

849 931

239 683

585 786

889 610

7 499

317 498

223 556

784 255

586 615

537 894

833 674

405 112

194 803

932 845

837 785

602 939

259 191

345 646

880 495

883 753

18 371

857 31

139 601

931 667

776 768

285 762

309 111

911 316

162 507

624 936

390 209

639 2

148 469

759 185

398 548

980 262

425 490

94 100

160 307

826 504

706 460

480 876

349 848

932 584

624 749

198 636

774 957

825 857

972 857

816 269

404 311

59 145

136 40

106 201

682 465

6 768

975 614

202 498

14 524

732 889

470 555

913 337

559 295

34 425

834 234

227 789

221 984

223 315

812 346

520 609

288 242

534 507

521 303

157 173

311 186

42 572

556 651

590 746

634 827

47 963

538 740

519 797

493 501

173 50

116 339

704 576

357 832

611 299

216 958

888 441

152 766

298 801

504 639

440 205

680 889

834 459

894 112

148 280

194 369

640 430

516 91

191 780

344 495

596 310

508 597

283 847

54 21

551 165

884 508

567 800

987 195

916 647

698 206

885 993

982 44

283 431

566 436

271 111

295 327

189 170

210 534

462 24

812 854

372 923

895 660

934 687

956 783

620 113

325 359

87 106

649 332

723 855

814 586

626 818

912 39

526 507

993 342

509 445

758 577

999 965

376 986

215 655

208 234

624 231

413 433

936 972

353 435

314 969

659 95

967 492

537 878

236 916

274 664

19 16

46 88

938 769

461 580

433 528

393 48

291 15

37 248

595 334

768 617

132 395

513 409

336 784

709 412

392 694

822 211

395 325

99 68

686 759

478 97

924 184

577 913

809 41

586 958

493 446

1 468

805 656

740 121

173 403

266 178

1000 42

262 759

350 610

327 57

64 888

866 60

14 806

821 904

437 640

768 973

902 234

92 574

407 846

425 266

221 467

360 933

703 426

47 205

798 783

374 947

857 485

676 320

708 771

295 494

903 12

233 610

356 295

313 677

759 625

136 586

980 282

898 309

167 501

789 126

882 370

913 123

782 344

939 781

270 119

541 497

70 112

873 717

917 122

853 300

844 554

976 824

126 650

586 644

463 595

362 582

519 611

817 420

288 42

938 691

176 64

328 270

202 901

969 260

785 693

914 596

804 204

922 16

639 289

175 116

916 696

433 807

887 6

338 501

268 69

971 192

578 798

405 237

421 318

104 647

979 423

295 703

226 222

421 783

796 429

239 64

848 188

749 324

887 658

340 868

808 606

931 314

205 690

839 685

923 916

220 774

202 758

776 993

202 318

428 666

707 691

766 338

169 345

572 262

533 612

20 759

642 86

6 200

435 789

391 135

545 83

283 334

221 668

554 427

932 189

134 586

118 289

270 44

610 634

407 441

784 371

451 344

831 847

321 41

647 5

474 9

114 227

783 953

397 327

333 96

401 840

565 946

315 99

980 793

631 190

941 12

685 586

749 98

567 633

610 810

880 207

768 254

733 866

63 202

632 76

923 410

250 716

30 318

346 963

522 834

36 707

925 317

262 411

616 915

381 886

938 281

873 977

102 441

439 605

607 558

877 911

893 230

62 63

640 218

619 620

918 460

768 105

793 29

488 419

489 605

520 243

992 611

711 878

306 656

526 862

88 383

917 460

861 461

497 954

22 293

411 427

562 369

463 524

333 642

581 281

16 34

374 352

136 650

859 296

636 157

182 225

394 561

670 335

921 975

87 542

813 481

757 486

105 308

267 772

722 227

883 180

115 742

747 75

852 691

699 162

378 994

288 568

593 388

984 315

386 282

643 794

313 54

545 328

376 220

535 553

653 610

438 616

292 493

911 326

484 169

148 616

12 993

366 172

22 541

648 81

950 577

847 907

575 193

768 189

822 595

11 608

305 730

421 273

815 534

978 203

513 351

632 25

761 417

327 762

410 516

602 383

191 410

309 287

493 604

865 818

43 701

211 878

37 922

496 611

911 591

785 171

434 241

305 509

336 402

541 475

629 391

441 874

826 686

209 498

272 485

859 856

946 454

56 747

487 314

385 502

89 379

625 860

880 167

696 170

861 356

366 420

713 934

585 243

885 386

266 374

804 455

678 567

169 926

915 312

140 657

496 438

823 555

979 811

861 428

134 101

669 975

786 471

757 401

752 711

424 267

226 754

48 809

154 30

675 335

719 939

989 940

90 217

356 966

465 100

504 546

745 468

597 658

996 432

224 687

266 422

15 273

913 792

275 163

436 107

174 249

198 204

232 42

169 491

47 933

240 263

965 901

251 105

221 717

658 333

665 995

29 679

107 85

748 517

329 499

1 869

577 726

391 507

473 258

609 242

363 947

397 944

448 348

476 389

98 247

293 953

635 428

626 261

566 834

589 871

900 506

245 742

337 306

700 45

894 420

731 104

239 559

725 44

881 618

783 956

68 712

65 391

652 839

87 464

888 454

618 128

717 593

955 995

796 745

901 507

291 457

794 34

919 718

555 76

836 987

625 120

110 273

885 943

759 457

855 27

980 847

961 18

968 510

576 459

83 420

590 570

746 66

747 996

76 712

111 363

33 269

429 191

493 523

431 595

687 145

689 798

758 180

485 178

405 935

143 698

252 205

590 641

919 168

411 695

40 903

766 873

966 120

434 455

206 694

540 518

320 702

652 421

118 614

369 417

329 520

112 714

107 292

967 275

760 931

878 387

782 313

984 366

539 250

871 648

402 263

851 685

578 153

634 606

35 695

63 717

283 35

931 815

597 592

752 668

764 926

218 756

549 806

102 511

422 957

320 373

717 902

567 740

824 788

999 79

324 69

267 971

448 252

861 376

597 326

724 431

740 137

466 560

770 490

680 130

892 556

443 742

975 887

556 658

680 439

119 570

575 418

174 604

800 671

840 166

52 107

967 597

882 771

260 717

210 673

571 797

934 1

160 933

497 761

553 156

921 109

952 477

637 328

599 851

261 234

599 777

478 656

955 808

954 524

59 342

443 911

468 101

848 491

775 999

692 708

909 328

795 16

106 499

965 530

960 663

648 319

978 311

787 419

35 174

937 588

914 726

780 552

65 633

387 218

951 377

351 698

85 46

37 545

608 146

795 690

798 959

847 485

13 290

978 958

395 626

847 382

433 242

668 184

327 988

20 208

610 465

589 94

450 739

84 734

894 887

791 586

814 122

802 789

288 813

907 727

492 645

814 121

295 192

390 635

409 315

146 667

930 17

392 296

844 260

297 113

165 919

534 785

114 682

930 159

743 84

406 200

938 126

751 172

583 998

874 211

955 256

799 226

955 496

185 685

584 374

880 347

287 24

383 255

621 597

202 653

528 367

916 64

470 73

126 530

466 410

829 857

808 449

569 661

603 820

933 155

700 900

810 390

976 543

484 9

832 674

725 643

309 870

74 490

731 184

648 165

973 459

262 73

350 431

803 442

826 934

789 102

789 103

369 380

509 250

331 652

701 491

661 410

510 391

585 429

851 189

557 210

800 693

445 812

686 254

475 365

286 111

934 415

926 197

399 422

206 120

423 335

71 14

917 130

360 619

600 217

291 655

209 401

936 340

108 389

709 464

587 411

2 343

257 881

402 31

63 978

333 149

471 999

132 520

655 85

792 959

727 292

478 175

293 316

400 105

318 973

855 429

300 743

883 160

214 328

503 140

476 955

124 864

694 516

863 213

121 802

525 674

531 929

705 142

431 854

913 64

941 716

669 124

64 430

148 945

72 499

742 811

439 886

680 480

484 46

539 211

10 597

45 572

715 811

67 920

111 140

424 453

161 767

466 950

320 775

365 212

536 211

640 230

585 146

103 381

324 898

249 107

24 82

56 649

839 429

959 507

482 561

24 341

423 657

937 468

974 318

63 408

1000 602

987 938

930 591

471 686

850 862

490 917

14 217

529 502

180 108

179 78

924 302

888 73

173 26

235 428

486 433

578 702

331 98

522 590

593 29

844 357

247 728

185 860

412 705

508 699

515 642

449 655

855 222

169 415

904 495

358 470

817 782

10 133

372 827

132 64

804 778

565 982

818 176

699 223

129 650

942 944

595 907

471 657

414 66

822 770

389 179

885 285

9 869

145 266

320 59

505 353

92 336

293 879

20 805

312 929

169 539

424 374

944 789

120 199

276 185

874 706

423 517

794 907

528 281

450 614

542 356

633 24

45 318

702 707

239 58

777 458

299 81

272 142

95 791

427 453

239 133

436 964

745 215

395 970

964 721

836 77

615 77

119 892

573 656

893 291

321 796

636 116

864 890

118 567

616 55

805 690

831 729

636 250

39 906

999 651

100 739

141 587

196 679

303 966

637 674

78 420

773 302

327 720

692 137

669 267

166 909

735 776

923 655

460 123

504 822

779 695

110 441

488 227

475 205

953 27

749 840

640 416

763 217

585 767

693 609

362 129

122 841

745 853

646 634

762 546

19 159

103 798

265 417

762 565

949 709

195 233

694 625

828 694

576 694

75 705

80 869

786 577

488 677

772 164

612 489

653 903

760 871

205 870

295 278

394 988

467 63

224 166

119 490

827 479

868 684

800 600

735 489

299 7

189 312

861 764

246 945

773 894

443 847

837 894

41 159

80 539

128 233

272 606

26 463

597 836

16 192

499 615

463 836

540 641

23 853

927 851

41 237

307 428

752 584

462 504

601 954

382 597

736 624

945 764

703 726

745 454

459 324

303 607

862 442

30 895

538 90

274 477

974 175

821 712

213 455

921 399

532 457

859 85

654 437

135 453

830 201

351 709

571 91

938 858

769 37

711 245

101 581

607 599

599 971

579 727

286 963

179 816

664 102

669 724

745 171

68 218

412 868

179 567

151 128

269 719

111 390

475 782

757 871

570 831

223 974

950 302

149 204

631 887

722 733

261 785

452 393

156 284

827 212

655 468

539 448

32 461

606 147

116 325

834 433

39 942

81 586

OUT:

1097 710 772 917 1916 744 595 1221 1657 152 439 522 916 1690 1078 880 375 1072 777 950 614 1220 453 158 59 131 105 478 27 448 791 750 896 1575 717 838 196 918 1168 1126 622 1555 1163 1231 1620 1466 1025 521 585 885 892 1040 1640 1830 1332 486 1777 1464 1878 261 334 739 1263 1783 441 1388 852 940 1702 434 589 801 1281 1456 1515 324 877 339 732 1963 484 212 1053 971 41 460 601 1094 1219 64 757 872 1102 1154 1407 781 1033 1213 1442 366 1438 403 1590 1737 98 562 28 1077 230 395 1002 278 421 641 783 704 233 289 431 1307 1853 423 721 1105 189 1833 1813 1107 735 385 1258 1909 1156 1181 1645 1728 1798 737 1471 1807 711 566 1089 792 1170 1529 642 553 1646 1665 1776 308 363 1298 277 428 524 1240 1458 1844 1845 1774 597 1528 154 1320 1835 471 636 1755 1206 1628 124 500 1417 1707 1956 294 1202 509 1073 137 249 1949 559 1019 1961 275 443 1605 48 111 1067 202 785 1225 821 1091 292 504 1233 1781 1228 46 259 956 1394 76 496 1380 1570 1669 1971 570 745 780 908 1670 1440 611 1262 1412 1712 1048 181 1404 1549 1754 1773 118 1559 945 1036 1684 930 1015 1142 1046 1623 333 361 268 360 1165 1850 527 825 837 835 1140 890 1460 1491 1688 799 1099 1615 663 765 1144 1610 1137 1872 1730 18 106 573 1135 1630 255 1450 1977 724 1884 1538 1691 90 889 272 701 1796 1839 426 1205 447 1535 1867 257 388 392 709 718 1718 1743 45 1726 1836 24 786 104 970 1479 97 221 1592 1751 1995 303 869 29 1704 55 563 1511 729 1653 1558 479 491 547 1285 416 1627 1889 1923 1716 185 1300 133 225 1403 824 1902 1026 1663 1548 1910 1363 1487 374 694 1315 578 1038 1364 173 1346 1973 1318 1631 992 816 1582 4 33 708 1710 1951 243 1801 583 1101 1482 1809 536 1296 1481 287 413 146 679 1092 1601 1662 952 1823 381 749 993 248 368 1182 171 658 1210 1324 1678 534 845 1230 935 1476 1171 1616 1929 21 114 410 813 1323 454 464 1029 764 625 1686 702 1822 427 911 1203 1629 972 1959 199 1239 682 1330 1162 1406 1985 1012 1119 1485 1877 680 1031 1236 1603 833 949 1679 1087 1633 1738 1749 1748 1991 556 1722 760 1841 67 386 389 736 1392 1693 1771 1881 1259 1587 1974 25 1309 883 91 342 1178 1791 215 305 419 726 1802 1286 1635 83 1222 1295 1876 523 480 492 569 1257 1750 1062 715 722 1894 1925 983 1061 1112 1563 1173 1935 32 321 226 662 551 834 1347 501 603 756 989 1424 1600 313 1266 1368 162 742 840 1964 183 567 138 1429 1079 1405 1507 1672 1856 1531 15 684 1237 1682 19 926 1641 367 398 577 1634 607 713 887 1423 373 661 690 229 1159 1622 653 1432 219 954 1199 276 518 1410 1509 438 688 176 227 293 7 1115 1812 613 778 1385 1868 167 968 1475 140 1157 86 260 451 1328 1401 1461 1502 433 84 220 93 353 1907 624 912 609 1016 476 1251 1280 1827 109 1132 1305 712 1990 435 1810 1338 1164 1212 358 1596 549 608 1848 14 116 1501 1709 228 575 1275 502 564 981 1128 1284 1621 38 681 913 794 1006 1145 119 511 733 795 1775 1865 1984 991 565 328 458 512 1437 343 957 101 759 1293 1133 191 238 296 462 1022 136 901 969 1580 1374 1917 236 519 1505 856 1187 1359 400 986 1304 1618 234 946 1711 168 216 1247 862 1174 1194 1243 1763 1831 650 1579 545 748 1123 147 576 654 473 531 630 1183 1945 271 1088 1996 1314 1533 1037 1224 635 1308 1561 1574 1717 506 450 859 1049 915 1414 1080 1207 378 1288 1389 1689 1815 586 861 980 307 1197 113 812 1003 198 280 408 546 1696 269 1760 165 1055 1568 1700 489 656 1138 1217 596 1198 195 335 987 1639 422 815 1427 331 1331 142 1282 134 831 1814 35 797 1365 1829 349 369 572 1784 141 640 1047 336 830 5 477 82 1474 692 1683 1506 1644 12 256 723 1017 1277 1136 143 247 707 939 1216 148 612 1524 1370 1638 1820 311 314 731 1532 817 1588 102 223 467 1361 1671 1752 317 517 1599 1271 1336 542 1369 1504 355 643 1303 1428 1697 1655 1979 1351 1446 1676 1989 0 211 1082 210 787 725 1954 88 365 554 796 1383 1695 507 1349 1100 125 881 72 863 1764 481 1083 632 1241 1337 1648 1675 1248 1699 1085 401 909 535 1685 1792 513 1880 728 1333 1498 1891 31 483 1545 1264 1581 1153 1366 1439 1469 1864 66 352 482 1496 1400 1687 80 538 1176 871 982 1118 1209 1276 1379 1431 1778 1936 894 1180 1677 1714 1757 387 646 1018 1731 1034 1534 1666 40 1398 306 876 1063 1779 1997 776 302 397 1065 1035 186 592 1957 868 1376 20 1543 17 455 860 891 1008 1215 1253 1875 574 866 1667 1947 380 1056 1096 159 1993 1650 493 246 377 634 741 175 1744 1838 1958 1357 1449 1944 235 1371 1492 1953 231 316 123 1455 1463 1955 1834 1014 1468 1664 959 1261 1272 581 1273 1994 442 730 85 1927 1448 1698 456 784 973 1604 1116 36 56 326 615 1396 1762 1992 347 372 951 108 804 1794 412 1382 810 1350 54 617 667 1560 1950 590 1912 1736 790 1291 250 425 543 910 660 1122 1355 1595 1292 254 279 1903 1915 309 600 651 955 1537 1661 1911 775 1408 1571 1674 52 571 743 1068 120 348 1313 670 1125 655 666 714 937 1021 1793 1636 1141 89 465 924 976 1354 11 923 1419 1576 1733 1001 1134 1169 463 605 1360 1772 81 340 958 1938 329 200 253 1435 290 947 990 1054 1422 1454 1758 178 619 1027 645 1348 1467 1522 879 898 1335 1721 1418 1825 92 214 1494 693 850 1499 1706 530 773 1477 977 1278 1567 706 1076 537 767 720 1577 1647 145 628 95 265 638 1039 1329 267 129 232 193 356 842 686 1819 1924 560 1319 1290 201 341 1656 808 1591 1395 1888 237 344 371 953 752 610 121 1585 832 1310 420 1146 117 979 1377 190 449 488 696 925 1539 550 755 627 1255 6 399 452 700 1441 882 1536 252 1287 1759 10 284 793 1892 1372 1852 1976 1302 204 318 394 782 1470 1544 516 151 994 1740 1113 298 705 1004 541 1057 1321 1900 644 648 1075 499 847 1965 1151 390 962 1937 320 626 498 505 526 1051 1131 1211 1232 1609 1999 1860 240 1583 103 1782 1345 1767 414 873 1517 1451 301 811 1150 1325 1478 357 1161 274 1023 1550 1642 1739 1940 593 826 1966 820 1914 3 941 1765 734 1339 1546 466 1254 1265 1186 1512 1926 319 1946 1326 96 475 988 1883 922 1104 1127 1311 1152 1267 1344 1200 179 587 975 1497 1828 42 224 927 1930 1312 1316 9 79 673 1081 156 1443 457 1692 1260 310 396 112 217 1941 1130 1895 78 1598 258 633 895 900 671 126 639 1234 1586 1214 1887 44 184 1617 964 338 417 169 672 888 1011 1110 579 1486 1932 897 1279 1789 540 1659 130 262 1149 1614 800 936 1030 1175 689 1508 1756 647 1148 1283 1803 995 1858 62 68 1673 144 1643 1060 1694 1790 1871 436 1098 1269 1565 1847 754 855 1375 1761 1806 8 568 1184 1397 1542 1043 1651 1578 1071 430 379 382 561 1195 902 942 1619 1208 1518 153 827 1547 160 1553 929 1658 1724 1863 510 805 239 904 1129 1901 163 1416 1861 51 1626 920 1913 362 788 1189 1510 1637 161 1353 1768 1044 1399 407 1188 1593 1854 1155 1196 1299 345 404 197 1160 1680 1086 1493 1896 1897 514 1488 1513 1874 411 1166 1484 1589 405 424 675 1788 829 402 1341 1495 1780 1177 839 854 858 1787 1898 1824 828 893 1246 1832 34 1573 1893 1960 155 1384 1445 1473 1952 1500 285 1242 1729 273 1143 1413 1514 1552 77 241 364 1457 1978 844 1866 166 1843 180 584 604 621 857 192 1972 297 582 1421 1584 1943 1613 1968 1785 393 1855 1327 69 819 1987 1607 295 770 1606 1859 999 1526 213 1297 1436 1540 312 1715 461 1453 606 761 996 70 1358 188 525 963 50 528 177 266 938 474 1386 87 288 657 849 1520 884 1192 370 1090 1103 1201 779 865 1557 944 1334 205 1918 1942 798 602 1009 548 1745 1882 886 906 943 974 1074 1808 848 1124 1551 1294 139 459 623 1191 60 727 1747 110 1870 1564 1799 668 766 1020 495 629 1139 1795 1980 1045 1120 1179 1084 429 933 1625 1988 921 23 26 591 1527 984 1204 1611 1821 164 544 1837 1402 299 376 1229 251 1306 1849 37 187 332 529 698 1000 1554 174 315 1172 1480 1890 532 1028 1010 330 359 1723 678 762 769 931 444 691 384 1817 283 1108 1521 485 1167 1566 391 903 1387 446 1235 594 1378 1734 1741 841 1681 1612 1516 150 1970 695 738 61 822 1052 1340 874 631 1652 515 851 1873 598 747 415 1147 470 683 719 22 997 1797 2 558 207 637 1982 843 1005 440 1245 1433 1928 1931 73 182 437 870 1447 1226 1879 65 157 244 1885 468 39 932 383 1114 409 1024 1218 1465 1921 961 209 127 1562 1934 1886 1933 128 1041 1727 659 1050 771 1356 966 967 1649 1962 149 676 677 803 1362 1786 1270 1769 1720 674 1238 16 1185 1975 1420 1811 1899 1 63 1660 1906 1434 1905 1981 620 30 818 1490 1352 960 687 1069 1268 1342 1816 49 322 494 1701 875 203 263 1250 1735 1541 1608 1986 685 1106 242 978 1013 487 1851 406 1846 928 1920 1922 245 1948 490 350 1753 809 665 1654 1158 1411 418 618 1525 599 1489 1904 1109 807 1857 1322 1805 1826 43 346 351 354 533 697 1869 47 218 552 1256 1569 905 1093 337 508 1249 1070 1190 649 1770 853 469 1624 503 1742 57 1343 1042 445 323 1373 1519 497 588 616 1725 1818 122 919 1503 99 846 1117 1409 1556 222 432 1367 1668 170 1483 327 948 75 753 768 703 823 1766 934 1390 1391 206 300 1998 286 1462 699 1426 1804 100 1732 1919 194 1227 132 914 1121 1425 264 751 652 716 1746 789 1223 1430 907 1274 1939 291 1719 71 74 1444 94 664 965 1523 864 1007 1095 1597 282 1594 1708 802 806 58 135 878 740 998 1244 1969 763 814 1840 1058 472 1393 1862 1066 836 1842 270 539 746 1530 1967 1064 1111 1713 1983 53 208 1289 1381 172 520 1252 555 1703 304 1800 325 985 115 580 1059 1459 1602 1908 13 557 669 774 107 1032 1193 1317 1301 281 1415 1452 1472 899 758 1632 1572 1705 867

### 3.14我的日程安排表 I

#### 3.14.1题目及其要求

实现一个 MyCalendar 类来存放你的日程安排。如果要添加的日程安排不会造成 重复预订 ，则可以存储这个新的日程安排。

当两个日程安排有一些时间上的交叉时（例如两个日程安排都在同一时间内），就会产生 重复预订 。

日程可以用一对整数 *start* 和 *end* 表示，这里的时间是半开区间，即 [*start*, *end*), 实数 *x* 的范围为,  。

实现 MyCalendar 类：

MyCalendar() 初始化日历对象。

*boolean book(int start, int end)* 如果可以将日程安排成功添加到日历中而不会导致重复预订，返回 *true* 。否则，返回 *false* 并且不要将该日程安排添加到日历中。

#### 3.14.2概要设计

程序设计图如下:



#### 3.14.3算法分析

* 将题目要求想象成一辆公交车,在某个时间段内有人上车,对于本题来说,车上的乘客数量不能大于1
* 设置成员对象map<int,int> mp 将*start*处+1,*end*处-1,则遍历整个*map*使用变量*ans*去访问*mp*中每个元素,如果,则退出循环,返回*false*,并将*mp*复原核心代码如下:

bool book(int *start*, int *end*) {  
 mp[*start*]++;mp[*end*]--;int *ans* = 0;  
 *//想象一辆公交车，按时间有人上车，有人下车。对于这一题，车上的乘客不能多于一个人。* for (std::map<int, int>::iterator *it* = mp.begin(); *it* != mp.end(); *it*++) {  
 *ans* += *it*->second;  
 if (*ans* >= 2) {mp[*start*]--;mp[*end*]++;return false;}}  
 return true;}

**时间复杂度分析:**

单次插入为,遍历整个表一遍用时,故总时间复杂度为

#### 3.14.4使用说明

输入说明 :

输入若干行：

每行一个*book*指令，后面跟两个整数*start*和*end*。

提示：

*book*指令不超过 1000 次。



程序名称为14*.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图

图形用户界面

低可信度描述已自动生成

#### 3.14.5测试结果及分析

* 第一组数据:保证各个区间均是有效的

IN:

book 10 20

book 20 30

book 30 40

book 40 50

book 50 60

book 60 70

OUT:

true

true

true

true

true

true

* 第二组数据:测试边界条件

IN:

book 0 10000000000

book 1 4

book 2 3

book 6 9

book 5 10

OUT:

true

false

false

false

* 第三组数据:普通数据,随机样例

IN:

book 783 452

book 641 29

book 938 648

book 872 708

book 373 96

book 604 420

book 302 474

book 762 384

book 686 413

book 404 330

book 72 954

book 839 794

book 413 829

book 935 29

book 825 453

book 659 970

book 143 443

book 184 646

book 819 931

book 293 762

book 503 860

book 724 164

book 35 997

book 690 999

book 191 827

book 240 494

book 779 663

book 839 708

book 445 78

book 505 919

OUT:

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

false

true

true

true

false

true

true

false

false

true

false

true

true

true

true

* 第四组数据,普通数据,随机样例
* IN:

book 890 294

book 607 871

book 456 227

book 992 694

book 101 140

book 472 422

book 137 395

book 649 103

book 866 164

book 807 734

book 406 968

book 525 27

book 439 750

book 723 543

book 502 449

book 964 140

book 283 908

book 946 891

book 899 733

book 829 711

book 151 784

book 889 514

book 917 896

book 823 941

book 619 521

book 772 704

book 328 92

book 177 907

book 410 488

book 193 640

OUT:

true

true

true

true

true

true

false

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

true

* 第五组数据:普通数据,随机样例

IN:

book 157 405

book 250 773

book 629 488

book 333 323

book 383 380

book 415 426

book 427 995

book 563 277

book 884 991

book 504 639

book 430 852

book 839 737

book 188 149

book 340 621

book 227 425

book 969 422

book 221 652

book 885 372

book 344 90

book 553 717

book 445 518

book 898 109

book 853 987

book 149 177

book 733 800

book 439 531

book 744 83

book 879 950

book 136 201

book 171 320

OUT:

true

false

true

true

true

true

true

true

false

false

false

true

true

true

false

true

false

true

true

true

true

true

false

true

true

true

true

true

true

true

* 第六组数据:大样例测试

IN:



OUT:



### 3.15设计电影租借系统

#### 3.15.1题目及要求

你有一个电影租借公司和 *n* 个电影商店。你想要实现一个电影租借系统，它支持查询、预订和返还电影的操作。同时系统还能生成一份当前被借出电影的报告。

所有电影用二维整数数组 *entries* 表示，其中 *entries*[*i*] = [*shopi*, *moviei*, *pricei*] 表示商店 *shopi* 有一份电影 *moviei* 的拷贝，租借价格为 *pricei* 。每个商店有 至多一份 编号为 *moviei* 的电影拷贝。

系统需要支持以下操作：

*Search*：找到拥有指定电影且 未借出 的商店中 最便宜的 5 个 。商店需要按照 价格 升序排序，如果价格相同，则 *shopi* 较小 的商店排在前面。如果查询结果少于 5 个商店，则将它们全部返回。如果查询结果没有任何商店，则返回空列表。

*Rent*：从指定商店借出指定电影，题目保证指定电影在指定商店 未借出 。

*Drop*：在指定商店返还 之前已借出 的指定电影。

*Report*：返回 最便宜的 5 部已借出电影 （可能有重复的电影 *ID*），将结果用二维列表 *res* 返回，其中 *res*[*j*] = [*shopj*, *moviej*] 表示第 *j* 便宜的已借出电影是从商店 *shopj* 借出的电影 *moviej* 。*res* 中的电影需要按 价格 升序排序；如果价格相同，则 *shopj* 较小 的排在前面；如果仍然相同，则 *moviej* 较小 的排在前面。如果当前借出的电影小于 5 部，则将它们全部返回。如果当前没有借出电影，则返回一个空的列表。

请你实现 *MovieRentingSystem* 类：

*MovieRentingSystem*(*int* *n*, *vector*<*vector*<*int*>>& *entries*) 将 *MovieRentingSystem* 对象用 *n* 个商店和 *entries* 表示的电影列表初始化。

*vector*<*int*> *search*(*int* *movie*) 如上所述，返回 未借出 指定 *movie* 的商店列表。

*void* *rent*(*int* *shop*, *int* *movie*) 从指定商店 *shop* 借出指定电影 *movie* 。

*void* *drop*(*int* *shop*, *int* *movie*) 在指定商店 *shop* 返还之前借出的电影 *movie* 。

*vector*<*vector*<*int*>> *report*() 如上所述，返回最便宜的 5部已借出 电影列表。

注意：测试数据保证 *rent* 操作中指定商店拥有未借出的指定电影，且 *drop* 操作指定的商店之前已借出指定电影。

#### 3.15.2概要设计



#### 3.15.3算法分析

**算法说明**:

* 由于*price*在整个操作过程中的不变的,而且较多的函数需要知道一部电影在某个商店对应的价格,故可以设置哈希表来通过电影和商店查询价格,但是在*unordered\_map*中并没有重载*pair*的哈希函数,故需要手动实现,手动实现的方式有两种,一种是使用auto关键字如LeetCode题解一样,该实现方法较为简洁,一种是设置自己的*hash*结构体重载()运算符即可,这种方法的可扩展性和可移植性较强,小组选择了后者,如下:

**struct hash\_func {  
 hash\_func() {}  
 size\_t operator()(const std::pair<int, int> &*i*) const*//注意这里一定要const* {  
 return std::hash<int>()(*i*.first << 16) ^ std::hash<int>()(*i*.second);  
 *//hash键值对为第一pair左移12位再异或第二pair的hash,最后统一hash* }  
};**

* 对于*search*操作,我们需要找出拥有指定电影的商店,且需要按照一定的规则排序,故我们可以设计哈希表*t\_vaild*,以*movie*为索引,映射至一个*set*集合,该集合内部的每一个元素是一个结构体,其中存放两个元素*price*和*shop*按照*price*为第一关键字,*shop*为第二关键字对其重载<运算符,在每一次调用*search*时,只需要遍历其对应*movie*背后的集合,输出前5个即可
* 对于*rent*操作,由于需将借出去的电影生成*report*,只需要先通过*t\_price*得到价格,再从*t\_vaild*[*movie*]对应的*shop*中将其移出,再插入到*report*集合即可
* 对于*drop*操作,即*rent*的反向操作,从*report*集合中移出,并将对应的值插入*t\_vaild*集合即可.
* 对于*rep*操作,只需从*report*集合中输出前5个即可.

本题涉及到重载小于运算符,重载的方法和之前一致,这里略.

**时间复杂度分析**:

* *Init*操作,哈希表单次插入,set集合单词插入,一共执行*m*次,则总复杂度为
* *Search*操作,哈希表的查找是的,访问其集合中前5个元素也是,总复杂度为
* *Rent* 操作,哈希表单次查找,集合的单词插入与移出元素是
* *Drop*操作: 哈希表单次查找,集合的单词插入与移出元素是
* *Rep*操作:输出集合中的前5个元素,用时

#### 3.15.4使用说明

输入说明 :

输入若干行：

第一行两个整数*n*和*m*，*n*表示电影商店的个数，*m*表示用于初始化对象的*entries*数组的行数。

然后*m*行，每行输入三个整数*shopi*, *moviei*, *pricei*表示*entries*的每一行。

后面有若干行，每行输入为*search*、*rent*、*drop*或*report*其中之一：

    如果指令为*search*类型，后面需要输入一个整数表示*movie*。

    如果指令为*rent*类型，后面需要输入两个整数表示*shop*和*movie*。

    如果指令为*drop*类型，后面需要输入两个整数表示*shop*和*movie*。

    如果指令为*report*类型，后面不需要输入整数。

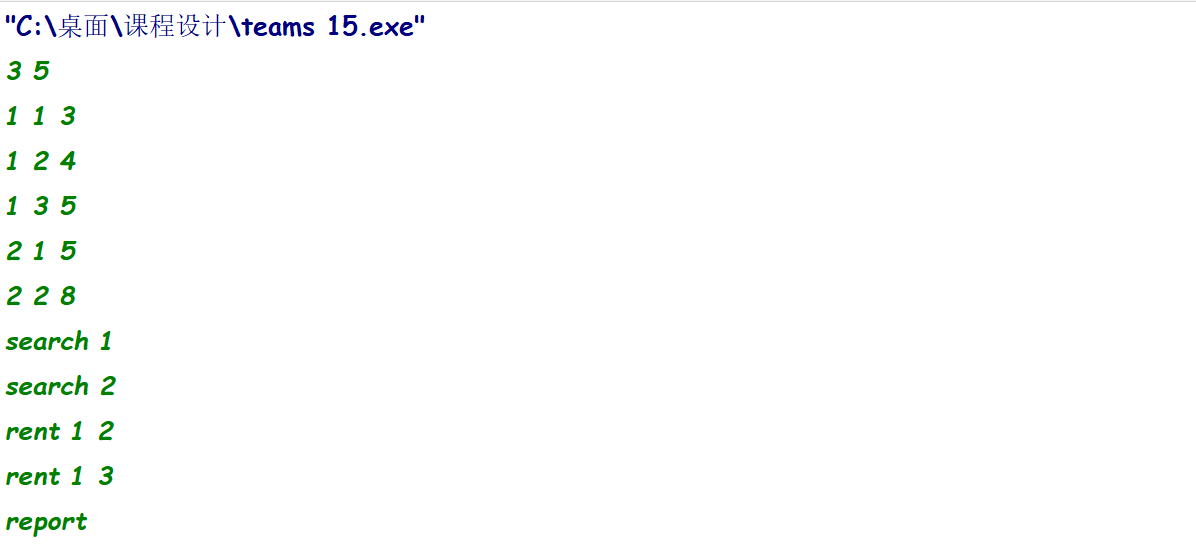
提示：



    每个商店 至多 有一份电影 *moviei* 的拷贝。

*search*，*rent*，*drop* 和 *report* 的调用 总共 不超过次。

程序名称为15*.cpp* , 程序在*Clion 2023.1,MVSC 17.0*编译器下能正常调试运行,程序运行如图



#### 3.15.5测试结果及分析

注:由于本题输入数据较为复杂,小组设计以下随机样例生成器,相比于纯随机生成器,该生成器除了保证生成的数据在约定范围内外,更保证了每个商店至多有一份电影 *movie*i 的拷贝; *rent* 操作中指定商店拥有未借出的指定电影，且 *drop* 操作指定的商店之前已借出指定电影,不会出现没有*rent*就*drop*的情况,且调用4个函数的概率随机,在产生特殊数据,如只借不还的操作时,也保留了灵活调整的功能.(双击可打开以下附件)



* 第一组数据:仅有search操作,测试该功能是否良好

IN:

267851 80

64 38 96

8 85 25

61 16 5

46 48 1

90 89 36

22 30 26

36 42 1

3 58 64

8 3 54

87 57 83

42 15 58

81 9 26

85 14 83

13 46 25

49 18 95

38 7 54

66 69 70

28 35 18

42 29 81

2 26 75

17 54 32

15 64 96

46 62 75

41 98 33

37 63 79

47 15 31

7 98 3

72 43 26

10 61 26

87 94 98

36 31 83

59 2 28

66 44 85

54 21 60

74 77 37

60 95 53

16 49 8

40 90 48

93 84 98

48 58 75

98 97 28

87 64 60

49 44 20

81 0 36

100 46 91

91 1 70

34 8 21

37 91 83

49 34 95

87 29 76

22 96 49

96 20 84

7 23 44

8 31 80

0 48 58

87 99 68

64 82 16

1 22 79

76 15 25

23 5 88

55 35 6

93 44 50

11 8 26

43 63 16

28 93 61

38 9 98

31 33 27

58 51 14

91 73 62

14 5 77

40 32 86

34 17 52

64 77 16

42 45 79

39 32 4

22 93 70

41 11 21

70 84 100

73 73 20

85 43 70

search 44

search 57

search 38

search 2

search 93

search 54

search 34

search 89

search 97

search 57

search 8

search 11

search 51

search 61

search 98

search 0

search 17

search 98

search 31

search 18

search 16

search 32

search 32

search 34

search 97

search 18

search 17

search 22

search 62

search 44

OUT:

49 93 66

87

64

59

28 22

17

49

90

98

87

34 11

41

58

10

7 41

81

34

7 41

8 36

49

61

39 40

39 40

49

98

49

34

1

46

49 93 66

* 第二组数据:仅包含rent和report

IN:

158568 28

97 47 84

100 28 100

70 16 23

46 36 64

73 96 21

24 29 65

38 82 53

74 56 53

38 16 14

35 60 28

70 38 91

24 53 92

39 25 79

56 15 38

36 64 3

88 29 4

7 59 67

48 82 3

88 81 88

70 21 3

23 91 83

80 54 25

64 63 84

22 2 95

47 75 87

4 38 19

41 31 92

2 34 2

rent 22 2

report

rent 41 31

report

rent 97 47

rent 56 15

report

rent 48 82

report

OUT:

22 2

41 31

22 2

56 15

97 47

41 31

22 2

48 82

56 15

97 47

41 31

22 2

* 第三组数据:不包含drop操作

IN:

75846 97

76 33 88

32 73 1

20 27 92

91 10 6

38 23 30

40 12 87

96 93 79

12 41 6

83 99 27

29 97 60

78 42 0

61 38 50

81 60 82

92 85 74

70 55 58

76 17 26

37 86 7

9 100 30

80 77 99

43 54 2

52 45 89

100 69 45

73 50 92

99 65 39

14 61 41

98 96 4

43 1 44

36 69 79

97 63 75

71 53 56

39 34 44

29 47 11

59 77 40

82 83 8

89 91 50

47 65 52

86 85 17

70 61 14

61 70 34

91 29 21

57 12 90

67 20 58

80 67 71

41 73 57

62 27 85

89 43 54

38 43 84

78 37 17

67 71 60

28 73 40

80 16 63

41 4 46

48 50 61

7 0 34

40 36 43

72 88 63

19 30 16

76 71 78

72 71 41

76 63 85

36 52 21

45 54 79

37 53 31

31 73 66

91 82 46

64 68 98

39 94 11

58 37 90

10 51 71

88 55 23

37 57 44

33 28 73

85 72 89

89 8 96

31 25 98

23 28 64

76 65 8

15 18 71

61 56 81

34 12 73

92 2 20

97 50 84

15 97 29

31 9 56

20 62 70

43 70 18

10 22 52

13 59 0

15 12 53

27 88 83

59 76 85

3 13 41

28 63 78

63 71 55

17 73 22

31 4 91

89 67 94

rent 34 12

search 47

rent 36 69

report

report

report

search 13

report

report

report

rent 99 65

report

report

search 99

report

search 41

search 28

rent 100 69

report

search 82

OUT:

29

34 12

36 69

34 12

36 69

34 12

36 69

3

34 12

36 69

34 12

36 69

34 12

36 69

99 65

34 12

36 69

99 65

34 12

36 69

83

99 65

34 12

36 69

12

23 33

99 65

100 69

34 12

36 69

91

* 第四组数据:普通数据,无任何限制

IN:

29 48

14 23 55

10 0 23

4 1 42

15 22 54

25 11 99

12 8 65

3 28 52

24 2 89

3 8 24

16 22 56

18 3 43

10 22 44

4 27 93

28 18 51

27 10 69

27 29 39

19 9 18

13 20 3

6 23 43

5 25 7

13 1 98

15 6 10

25 2 86

4 16 28

17 18 65

7 17 97

10 4 58

6 6 67

13 22 35

8 1 66

2 1 2

11 28 51

9 27 56

26 1 54

5 27 89

15 17 31

5 30 45

22 9 7

6 19 26

19 21 32

5 28 78

13 17 68

20 9 63

0 25 90

11 5 37

1 6 36

9 16 9

18 0 55

report

report

search 22

search 11

search 19

search 11

report

search 4

report

rent 6 19

rent 9 16

search 18

search 22

rent 17 18

drop 17 18

rent 14 23

report

rent 19 21

report

search 1

rent 1 6

report

rent 4 1

rent 27 29

rent 6 6

rent 17 18

rent 26 1

drop 17 18

report

drop 9 16

rent 12 8

report

search 4

search 20

search 4

search 27

report

rent 17 18

drop 17 18

report

rent 13 1

search 4

search 3

search 4

drop 19 21

report

rent 13 22

drop 13 1

OUT:

13 10 15 16

25

6

25

10

28 17

13 10 15 16

9 16

6 19

14 23

9 16

6 19

19 21

14 23

2 4 26 8 13

9 16

6 19

19 21

1 6

14 23

9 16

6 19

19 21

1 6

27 29

6 19

19 21

1 6

27 29

4 1

10

13

10

9 5 4

6 19

19 21

1 6

27 29

4 1

6 19

19 21

1 6

27 29

4 1

10

18

10

6 19

1 6

27 29

4 1

26 1

6 19

13 22

1 6

27 29

4 1

* 第六组数据:大数据检验,包含9万数据读入以及2万次的函数调用,运行情况良好

IN:



OUT:

