项目说明文档

数据结构课程设计

——修理牧场

作 者 姓 名： 郑柯凡

学 号： 1950072

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目录

[1 分析 2](#_Toc15300)

[1.1背景分析 3](#_Toc9915)

[1.2功能分析 3](#_Toc6647)

[2 设计 3](#_Toc1248)

[2.1思路设计 3](#_Toc10170)

[2.2数据结构的选择 3](#_Toc16407)

[2.3类的设计 4](#_Toc12593)

[3 功能实现 5](#_Toc14465)

[3.1 堆调整功能的实现 5](#_Toc14117)

[3.1.1调整功能流程图 5](#_Toc20523)

[3.1.2调整功能核心代码 5](#_Toc3637)

[3.1.3调整功能截屏示例 6](#_Toc8435)

[3.2 堆插入功能的实现 6](#_Toc2457)

[3.2.1插入功能流程图 6](#_Toc5818)

[3.2.2插入功能核心代码 7](#_Toc23819)

[3.2.3插入功能截屏示例 7](#_Toc14379)

[3.3 构造霍夫曼树计算功能的实现 7](#_Toc30464)

[3.3.1计算功能流程图 7](#_Toc31380)

[3.3.2计算功能核心代码 7](#_Toc23953)

[3.3.3计算功能截屏示例 8](#_Toc8610)

[4测试 8](#_Toc11204)

[4.1计算功能测试 8](#_Toc29196)

**1 分析**

**1.1背景分析**

农夫要修理牧场的一段栅栏，他测量了栅栏，发现需要N块木头，每块木头长度为整数Li个长度单位，于是他购买了一个很长的，能锯成N块的木头，即该木头的长度是Li的总和。

但是农夫自己没有锯子，请人锯木的酬金跟这段木头的长度成正比。假设酬金等于所锯木头的长度，试问将木头锯成N块最少需要花费多少酬金。例如，要将长度为20的木头锯成长度为8，7和5的三段，第一次锯木头将木头锯成12和8，花费20；第二次锯木头将长度为12的木头锯成7和5花费12，总花费32元。如果第一次将木头锯成15和5，则第二次将木头锯成7和8，那么总的花费是35（大于32）.

**1.2功能分析**

第一行输入给出正整数N（N<10000），表示要将木头锯成N块。第二行给出N个正整数，表示每块木头的长度。计算木头锯成N块的最小花费，并输出。

**2 设计**

**2.1思路设计**

将要锯成的N块木头看作N个结点，每块木头的长度（酬金）即结点的权值，只需要将这N个结点构造成一棵霍夫曼树，就可以使树的带权路径长度最短，也就意味着农夫所需支付的酬金最少。

一开始，将N个结点视作独立的子树，每次从这所有的子树中挑选两个权值最小的结点进行合并，生成一棵新树，其权值为两个合并结点的权值之和。经过N-1次重复的操作，这N个结点就生成了一棵完整的树，这棵树就是霍夫曼树，其根结点的权值即最短的路径长度，也就是这道题目中要求的最小花费。

**2.2数据结构的选择**

每一轮操作都需要找出所有结点中权值最小的两个，可以采用排序时间复杂度较小的堆来实现。而本道题目中只要求计算出最小花费，因此不需要真正地生成一棵霍夫曼树，只要借用霍夫曼树的思想即可。

**2.3类的设计**

小顶堆类：其私有成员包括动态数组的首元地址，以及两个int型变量maxsize和cursixe记录堆的最大大小和当前大小。其公有成员包括插入，上滑调整，下滑调整，获取最小元等功能函数。

1. **class** MinHeap {
2. **friend** **class** Field;
3. **private**:
4. **int**\* heap;
5. //堆的最大大小
6. **int** maxsize;
7. //当前堆的大小
8. **int** cursize;
9. **public**:
10. //构造函数
11. MinHeap()
12. {
13. heap = **new** **int**[1];
14. maxsize = cursize = 0;
15. };
16. MinHeap(**int** sz)
17. {
18. heap = **new** **int**[sz];
19. maxsize = cursize = sz;
20. }
21. //析构函数
22. ~MinHeap() {};
23. //将x插入到堆中
24. **bool** Insert(**const** **int**& x);
25. //从start到0上滑调整为最小堆(用于插入）
26. **void** Siftup();
27. //把整个序列调整成最小堆
28. **void** Adjust();
29. //从start开始自顶向下调整为最小堆(用于调整成最小堆）
30. **void** Siftdown(**int** start);
31. //获取堆顶的元素,并将其删除
32. **int** GetMin();
33. };

牧场类：其私有成员包括一个小顶堆，用于储存所有木块以及计算最小花费，还有一个int型变量记录木块的树量。其公有成员包括输入函数，以及计算函数。

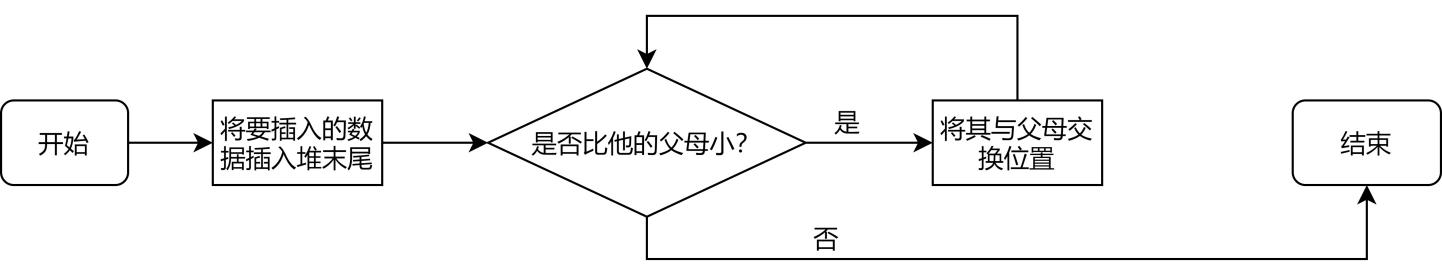
1. **class** Field {
2. **private**:
3. MinHeap wood;
4. **int** woodnum;
5. **public**:
6. //构造函数
7. Field() { woodnum = 0; };
8. //析构函数
9. ~Field() {};
10. //输入牧场的具体信息
11. **void** Input();
12. //构造霍夫曼树，计算最小花费
13. **void** Huffmantree();
14. };

**3 功能实现**

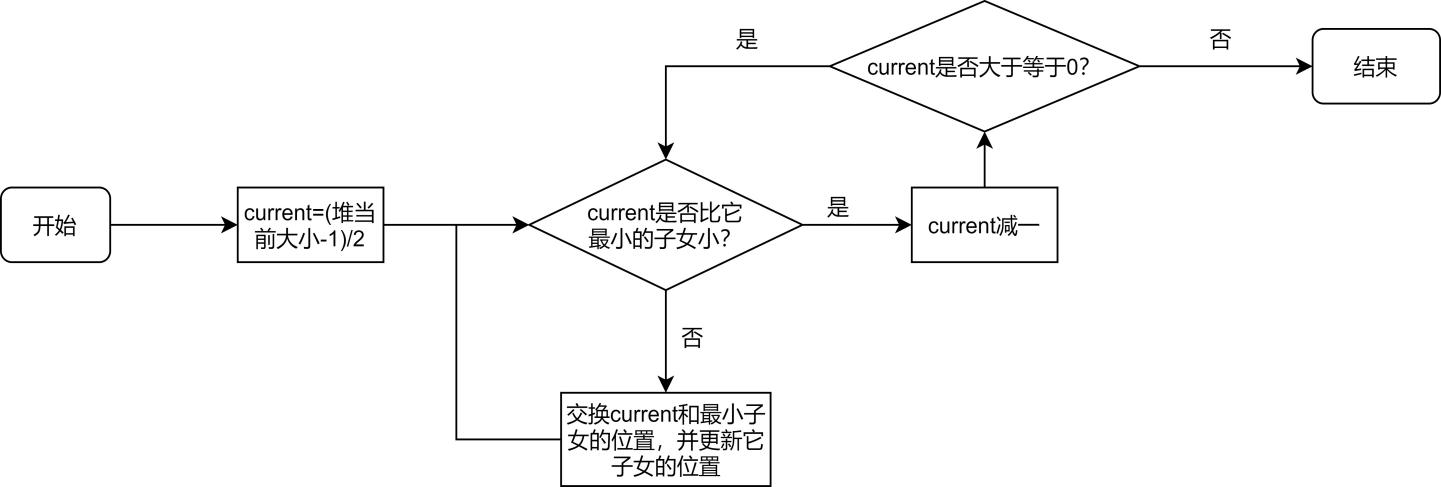
**3.1 堆调整功能的实现**

**3.1.1调整功能流程图**

**上滑调整（用于将数据插入已有的最小堆中）**

****

**下滑调整（用于将原本无序的堆调整成最小堆）**

****

**3.1.2调整功能核心代码**

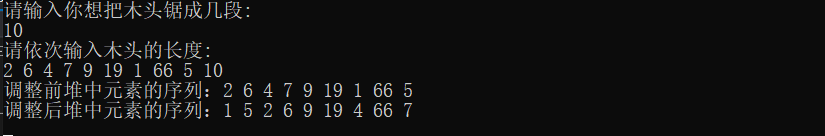
（上滑调整主要用于将数据插入已经成型的最小堆）将要插入的元素放在堆末尾，比较其和父母的大小，若其数据比父母小则交换数据，重复此操作直至其数据比父母大。由于堆本来就是一个最小堆，所以这种上滑操作不会影响最小堆的结构。

1. //从start到0上滑调整为最小堆(用于插入）
2. **void** MinHeap::Siftup()
3. {
4. **int** child = cursize, parent = (cursize - 1) / 2; **int** temp = heap[child];
5. **while** (child > 0)
6. {
7. **if** (heap[parent] <= temp)
8. {
9. **break**;
10. }
11. **else**
12. {
13. heap[child] = heap[parent];
14. child = parent;
15. parent = (parent - 1) / 2;
16. }
17. }
18. heap[child] = temp;
19. }

（下滑调整主要用于将原来无序的堆调整为最小堆）令当前堆的大小为cursize，则对编号为（cursize-1）/2到0的元素依次进行下滑调整，使以该元素为根结点的堆形成最小堆，最终可使整个堆变成最小堆。在下滑调整中，也是通过比较父母与子女的关系，使他们符合最小堆的定义来实现的。

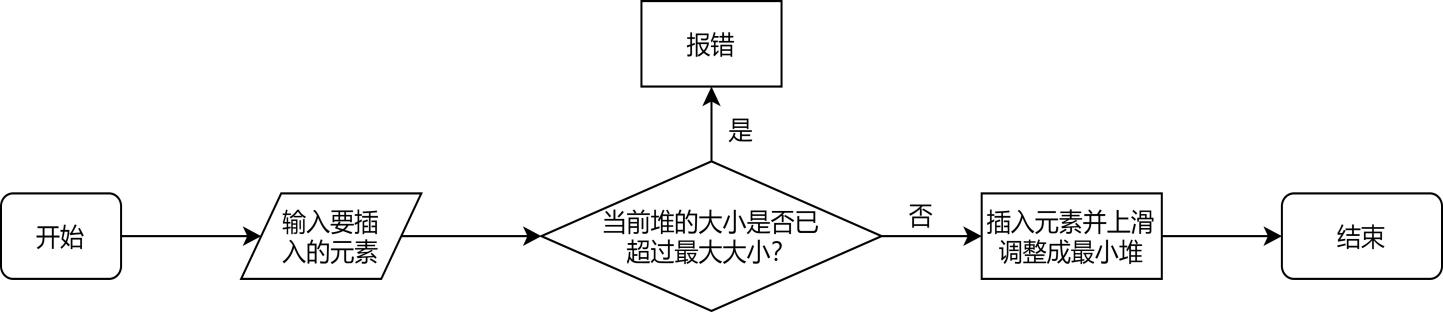
1. //从start开始自顶向下调整为最小堆(用于调整成最小堆）
2. **void** MinHeap::Siftdown(**int** start)
3. {
4. **int** parent = start, son = parent\*2+1,temp=heap[parent];
5. **while** (son < cursize)
6. {
7. **if** (son<cursize - 1 && heap[son]>heap[son + 1])
8. {
9. son++;
10. }
11. **if** (temp <= heap[son])
12. {
13. **break**;
14. }
15. **else**
16. {
17. heap[parent] = heap[son];
18. parent = son;
19. son = son \* 2 + 1;
20. }
21. }
22. heap[parent] = temp;
23. };

**3.1.3调整功能截屏示例**



**3.2 堆插入功能的实现**

**3.2.1插入功能流程图**

****

**3.2.2插入功能核心代码**

堆未满则插入，若堆满则输出错误提示

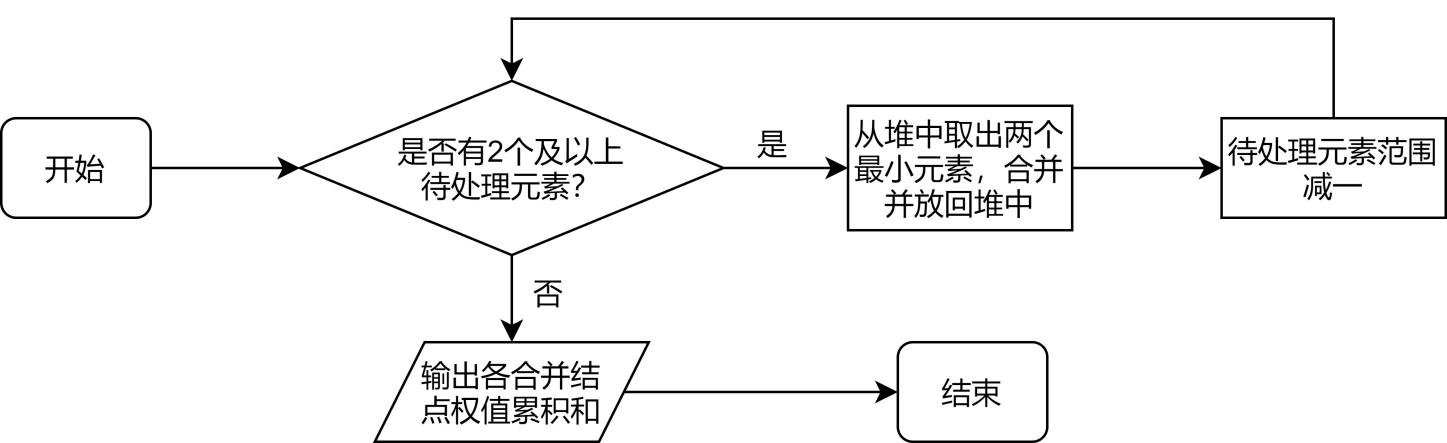
1. **bool** MinHeap::Insert(**const** **int**& x)
2. {
3. **if** (cursize < maxsize)
4. {
5. heap[cursize] = x;
6. Siftup();
7. cursize++;
8. **return** **true**;
9. }
10. **else**
11. {
12. cout << "堆已满，无法插入!" << endl;
13. **return** **false**;
14. }
15. }

**3.2.3插入功能截屏示例**



**3.3 构造霍夫曼树计算**功能的实现

**3.3.1**计算功能流程图



**3.3.2计算功能核心代码**

1. //计算最小花费
2. **void** Field::Huffmantree()
3. {
4. **int** ans = 0;
5. **while** (wood.cursize > 1)
6. {
7. **int** min1 = wood.GetMin();
8. **int** min2 = wood.GetMin();
9. **int** merge = min1 + min2;
10. ans = ans + merge;
11. wood.Insert(merge);
12. }
13. cout << "所需的最小花费为：" << ans << endl;
14. }

**3.3.3计算功能截屏示例**



**4测试**

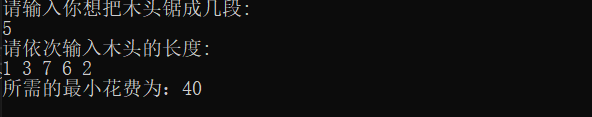
**4.1计算功能测试**

**测试用例****：**5

1 3 7 6 2

**预期结果：**40

**实验结果：**



**测试用例：**10

1 3 5 5 2 4 7 9 10 6

**预期结果：**164

**实验结果：**

