


项目说明文档

数据结构课程设计

——修理牧场

作者姓名： 陈垲昕

学 号： 

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

可通过按住 **Ctrl** 并单击访问说明文档各个模块：

1. 分析

1.1 背景分析

1.2 功能分析

2. 设计

2.1 主要数据结构设计

1. 最小堆实现的优先队列

2.2 系统类设计

流程图

3. 核心功能实现

3.1 数据的上浮（优先队列插入数据）

3.2 数据的下沉（优先队列删除数据）

3.3 最小带权路径长度计算

4. 测试

4.1 常规结果测试

Task 1 正常测试

Task 2 全相同段数测试

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 L_i 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 L_i 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。为简单起见，不妨就设酬金等于所锯木头的长度。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

（1）输入格式：输入第一行给出正整数 N （ $N < 104$ ），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。

（2）输出格式：输出一个整数，即将木头锯成 N 块的最小花费。

2.设计

2.1 主要数据结构设计

修理牧场的问题，可以使用 Huffman 树数据结构解决问题。Huffman 树是一类加权路径长度最短的二叉树，在编码设计、决策和算法设计等领域有着广泛的应用。

由于本题涉及到的是计算 Huffman 树的 WPL(带权路径长度)，故可以仅建立优先队列（最小堆）的数据结构，通过每次从堆顶取出两个最小的元素，将它们相加后重新放入堆中，循环以上操作最终得到 Huffman 树的带权路径长度

本程序设计了以最小堆形式构建的优先队列数据结构，以及用于程序执行的系统类

1.最小堆实现的优先队列

描述，模板类优先队列，数据成员主要包括模板类类型的动态数组_dataVec，记录堆中元素的记录变量_size，实现了对元素的下沉，上浮等操作，基本功能与最小堆的功能一致

```
template<class T>
class PriorityQueue {
public:
    //构造与析构函数
    PriorityQueue() :_size(0) {}
    PriorityQueue(int capacity) :_dataVec(new T[capacity]) {}
    ~PriorityQueue() { delete[] _dataVec; }

    //元素操作
    void push(T& data);    //将元素推入优先队列
    void pop();            //退出栈顶元素
    inline bool empty()const { return _size == 0; } //查空
    inline T top()const { assert(_size > 0); return _dataVec[0]; }
    //获取栈顶元素

private:
    T* _dataVec;    //动态数组
    int64 _size;    //当前堆中元素数量

    //交换_dataVec[i]与_daaVec[j]的值
    inline void _swap(Index i, Index j);
    //下沉
    void _sink(Number startNumber);
    //上浮
    void _floating(Number startNumber);
};
```

2.2 系统类设计

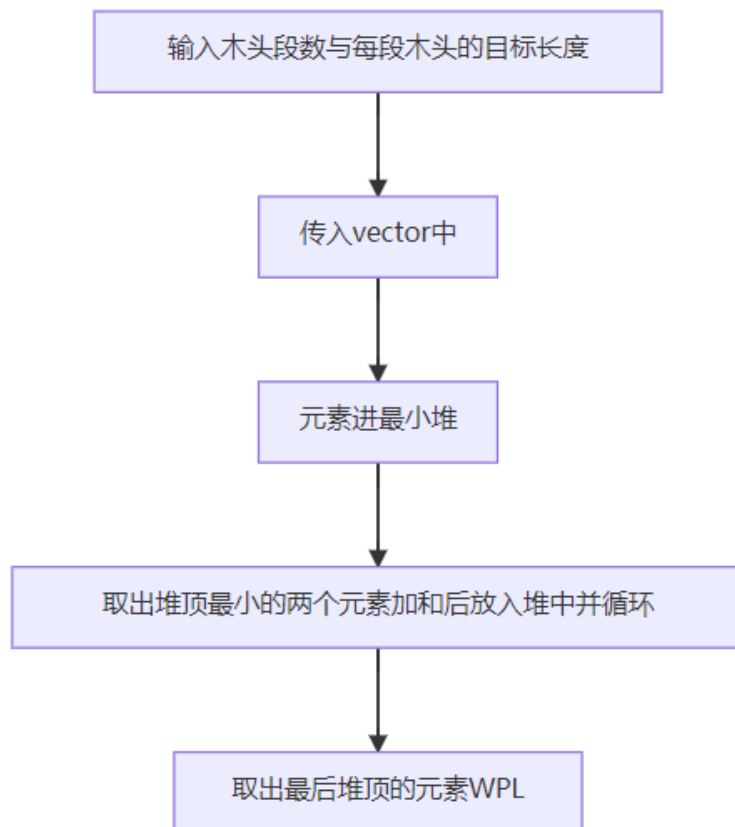
通过 **HuffmanTreeWPLCalculator** 类实现对霍夫曼树带权路径长度的计算，最终输出 WPL

```
class HuffmanTreeWPLCalculator {
public:
    // 构造
    HuffmanTreeWPLCalculator(vector<double> woodVec);

    // 获得WPL
    double getWeightedPathLength()const { return _WPL; }

private:
    double _WPL;
};
```

流程图

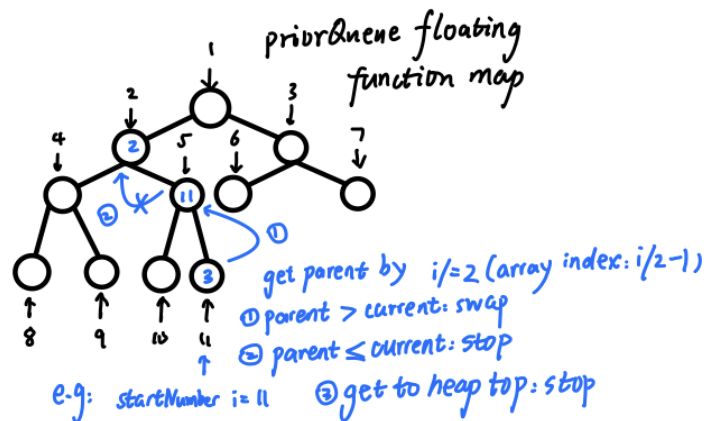


3.核心功能实现

3.1 数据的上浮（优先队列插入数据)

描述：本程序的优先队列从 **1----size** 对每个单元进行编号，输入一个参数 **startNumber**，表示对 **dataVec[startNumber-1]**处的数据进行上浮并用 **i** 进行记录。数据上浮时，对每个父节点处(以 **_dataVec[i/2-1]**)的数据，如果大于当前数据，便进行交换，并以 **i/=2** 的方式重新定位需要处理的数据，这样满足了最小堆数据将较小的数据置于堆的顶部的要求

插入数据时，对当前数组第一个未存储数据的单元进行写入，并对该单元进行数据上浮

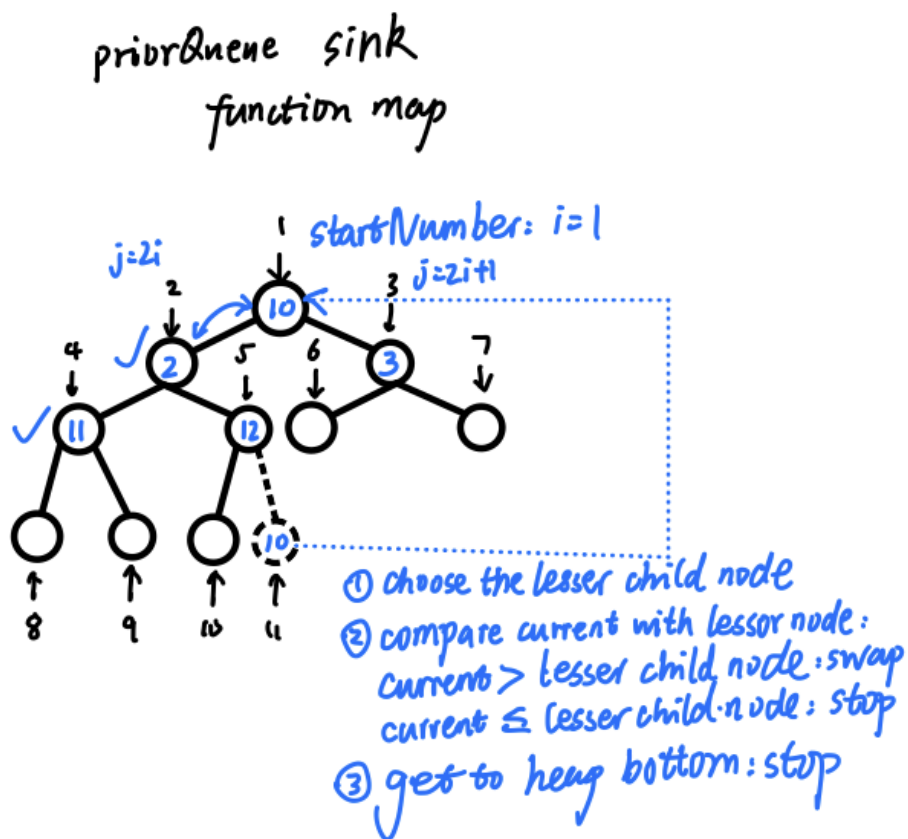


```
template<class T>
void PriorityQueue<T>::_floating(Number startNumber)
{
    if (_size <= 1)
    {
        return;
    }
    for (int i = startNumber; i > 0; i /= 2)
    {
        if (_dataVec[i - 1] < _dataVec[i / 2 - 1])
        {
            _swap(i - 1, i / 2 - 1);
        }
        else
        {
            break;
        }
    }
    return;
}
```

3.2 数据的下沉（优先队列删除数据）

描述：输入一个参数 **startNumber**，表示对 **dataVec[startNumber-1]**处的数据进行下沉并用 **i** 进行记录。数据下沉时，定位子节点中数据较小的结点，并与当前，如果大于当前数据，便进行交换，并定位子节点的位置，这样满足了最小堆数据将较大的数据置于堆的底部的要求

在删除数据时，将数组第一位（堆顶）与堆底数据交换，并对堆顶数据进行数据下沉



```
template<class T>
void PriorityQueue<T>::_sink(Number startNumber)
{
    int i = startNumber;

    while (i * 2 <= _size)
    {
        int j = 2 * i;
        if (j <= _size && _dataVec[j - 1] > _dataVec[j])
        {
            j++;
        }
        if (_dataVec[i - 1] <= _dataVec[j - 1])
        {
            break;
        }
        else
        {
            _swap(i - 1, j - 1);
        }
        i = j;
    }
}
```


3.3 最小带权路径长度计算

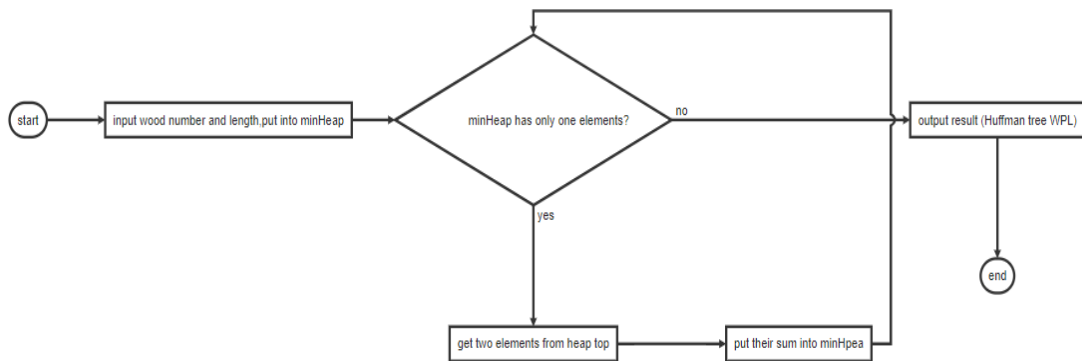
将木头长度数据存入优先队列，此后每次循环，取出堆顶的数据两次（由于输入时的 **floating** 机制与取出堆顶数据时 **sinking** 机制的维护，每次取出堆顶的数据都是优先队列中最小的元素），将数据相加后放入堆中，循环到直至堆中只剩最后一个元素时，即最终求得 **Huffman** 树的带权路径长度

```
for (int i = 0; i < operateTimes; i++)
{
    double first = priorQueue.top();
    priorQueue.pop();
    double second = priorQueue.top();
    priorQueue.pop();

    double parent = first + second;
    priorQueue.push(parent);

    _WPL += parent;
}
```

流程图



4.测试

4.1 常规结果测试

Task 1 正常测试

测试样例 1:

8
4 5 1 2 1 3 1 1

预期结果 1:

49

(此处贴出计算过程, 后续测试样例的计算过程原理同此)

1---原始序列:4 5 1 2 1 3 1 1

取出最小两个元素:1 1 放入后序列:4 5 1 2 1 3 2 WPL:2

2---原始序列:4 5 1 2 1 3 2

取出最小两个元素:1 1 放入后序列:4 5 2 2 3 2 WPL:4

3---原始序列:4 5 2 2 3 2

取出最小两个元素:2 2 放入后序列:4 5 4 3 2 WPL:8

4---原始序列:4 5 4 3 2

取出最小两个元素:2 3 放入后序列:4 5 4 5 WPL:13

5---原始序列:4 5 4 5

取出最小两个元素:4 4 放入后序列:8 5 5 WPL:21

6---原始序列:8 5 5

取出最小两个元素:5 5 放入后序列:8 10 WPL:31

7---原始序列:8 10

取出最小两个元素:8 10 放入后序列:18 WPL:49

实际结果 1:

```
[1]+ Stopped
ckx@VM-0-10-ubuntu:
8
4 5 1 2 1 3 1 1
49
```

测试样例 2:

```
10
7 9 10 23 1 3 5 9 41 29
```

预期结果 2:

388

实际结果 2:

```
ckx@VM-0-10-ubuntu:
10
7 9 10 23 1 3 5 9 41 29
388
```

Task 2 全相同段数测试

测试样例:

```
12
2 2 2 2 2 2 2 2 2 2 2 2
```

预期结果:

88

实际结果:

```
ckx@VM-0-10-ubuntu:
12
2 2 2 2 2 2 2 2 2 2 2 2
88
```