



同濟大學
TONGJI UNIVERSITY

数据结构课程设计

项目说明文档

牧场修理

姓 名： 马小龙

学 号： 2353814

学 院： 计算机科学与技术学院（软件学院）

专 业： 软件工程

指导教师： 张颖

二〇二三年十一月二十三日

目录

第 1 章 项目分析.....	1
1.1 项目背景分析.....	1
1.2 项目需求分析.....	1
1.3 项目功能分析.....	1
1.3.1 最低费用计算功能.....	1
1.3.2 异常处理功能.....	2
第 2 章 项目设计.....	3
2.1 数据结构设计.....	3
2.2 算法.....	3
2.3 最小堆类设计.....	3
2.3.1 概述.....	3
2.3.2 MinHeap 类定义.....	4
2.3.3 构造函数与析构函数.....	4
2.3.4 私有数据成员.....	5
2.3.5 私有成员函数.....	5
2.3.6 公有成员函数.....	5
2.4 项目框架设计.....	5
2.4.1 项目框架流程图.....	6
2.4.2 项目框架流程.....	6
第 3 章 项目功能实现.....	7
3.1 项目主体功能实现.....	7
3.1.1 实现思路.....	7
3.1.2 流程图.....	7
3.1.3 核心代码.....	8
3.1.4 示例.....	8
3.2 异常处理功能.....	9
3.2.1 输入非法的异常处理.....	9
3.2.2 动态内存申请失败的异常处理.....	9
3.2.3 最小堆已满或为空的异常处理.....	10
第 4 章 项目测试.....	11
4.1 项目输入功能测试.....	11
4.1.1 木头需要被锯成的块数输入功能测试.....	11
4.1.2 每一段木头的长度输入功能测试.....	11
4.2 最低费用计算功能测试.....	12
第 5 章 相关说明.....	13

5.1 编程语言	13
5.2 Windows 环境	13
5.3 Linux 环境	13

第 1 章 项目分析

1.1 项目背景分析

在乡村生活中，农夫常常需要对牧场进行维护和修缮。栅栏作为牧场的重要组成部分，其稳固性直接关系到牧场的安全与动物的管理。修理栅栏时，木材是必不可少的材料。然而，由于农夫自己没有锯木工具，他需要雇佣他人对木材进行加工。锯木的酬金是根据木材的总长度来计算的，因此如何减少锯木过程中的费用成为一个需要解决的问题。通过分析发现，锯木的总成本不仅取决于木材的初始长度，还与锯木的顺序密切相关。如果将木头分成两部分后继续锯小部分的木材，费用可能会低于直接从大木头分割到目标长度。因此，优化锯木顺序以最小化总费用成为农夫降低成本的关键。

这一问题的本质属于贪心算法和优先队列（最小堆）的应用场景，通过优先处理短木材的合并，可以有效地降低每次锯木的成本。这类问题在实际生活中具有广泛的应用，例如数据压缩的霍夫曼编码或任务调度等场景。解决这一问题不仅能帮助农夫节省修缮成本，还能够为类似的资源分配优化问题提供思路。

1.2 项目需求分析

锯木问题中，农夫希望在满足长度要求的前提下，尽可能降低总花费。因此，系统需要实现一个高效的解决方案，确保可以处理大规模数据并得出最低费用的锯木方案。具体来说，系统需要能够接受输入数据，包括目标木材分段的数量及每段的长度，在此之后，系统需要基于输入数据计算出将木材锯成指定长度所需的最小费用，最终输出最小花费值。

技术需求包括支持 Windows 和 Linux 等操作系统，采用 C++ 等主流编程语言，采用合适的数据结构。

1.3 项目功能分析

本项目旨在开发一套高效的解决牧场修理问题系统，以此来帮助农夫降低修理成本。项目核心功能为计算最低费用，同时需要有基本的异常处理功能。

1.3.1 最低费用计算功能

该功能需要实现根据输入的木头段数以及各个木头长度,计算出由一根木头锯成这些木头所需要花费的最低费用。实际计算中,需要借助贪心算法以及霍夫曼树的相关思想,计算出最低总费用。

1.3.2 异常处理功能

程序对各种异常进行了基本处理,以提升系统稳定性。

第2章 项目设计

2.1 数据结构设计

本项目设计并实现了最小堆，作为主要数据结构，采用最小堆的主要原因是：

1.最小堆是一种特殊的完全二叉树结构，满足以下性质：堆中每个节点的值小于或等于其子节点的值；堆顶（根节点）始终是堆中最小的元素。在农夫锯木问题中，我们需要频繁地获取当前最短的两段木材进行合并操作，而最小堆能够直接提供最小值，非常高效。

2.锯木问题需要反复执行以下操作：移除最小的两段木材；将合并后的木材长度重新加入队列。最小堆在插入和删除操作上均能以 $O(\log_2 N)$ 的时间复杂度完成，这在需要频繁动态更新的场景中非常高效。

2.2 算法

贪心算法是一种在求解问题时采取逐步构建解决方案的策略，每一步都做出当前状态下的局部最优选择，希望通过这些局部最优解最终得到全局最优解。贪心算法简单高效，常用于优化类问题。

贪心算法的特点是：选择性最优性，即算法每次的选择都是基于当前状态的最优解；无后效性，即当前的选择不会影响未来的选择，即之前的决策不需要重新调整；适用性，即问题需要满足贪心选择性和最优子结构两个条件，才能使用贪心算法。

在农夫锯木问题中，我们希望最小化锯木的总花费。该问题可以采用的贪心策略为：优先合并最短的两段木材，以最小化每次锯木的费用。因为较短的木材在后续合并中对总花费的影响较小，所以合并两段最短的木材会减少未来锯木的总花费。

2.3 最小堆类设计

2.3.1 概述

该通用模板类 `MinHeap` 用于表示最小堆。该最小堆提供了一系列基本操作函数，包括数据的插入、删除，最小堆的向上构建的向下构建的实现以及最小堆的构造和析构，满足了常见的最小堆操作需求。

2.3.2 MinHeap 类定义

```
template<typename T>
class MinHeap {
public:
    MinHeap(int maxSize);
    MinHeap(T arr[],int n);
    ~MinHeap(){delete[] heap;};
    bool Insert(const T& x);
    bool Remove(T& x);
    bool isEmpty()const{return currentSize==0;}
    bool isFull()const{return currentSize==maxHeapSize;}
    void makeEmpty(){delete[] heap; currentSize=0;}
    int Size(){return currentSize;}
    MinHeap<T>& operator=(const MinHeap<T>& R);
private:
    enum{DefaultSize = 10};
    T* heap;
    int currentSize;
    int maxHeapSize;
    void FilterDown(int start,int end_of_heap);
    void FilterUp(int start);
};
```

2.3.3 构造函数与析构函数

`MinHeap(int maxSize);`

构造函数，指定堆的最大容量，初始化一个空堆

`MinHeap(T arr[],int n);`

构造函数，使用一的大小为 `n` 的数组来构建最小堆；

`~MinHeap();`

析构函数，清除分配的内存，防止内存泄漏。

2.3.4 私有数据成员

enum{DefaultSize = 10};定义了 DefaultSize 常量，用于确定指出堆的最小大小（容量）。

T* heap; 指向模板类型 T 的动态数组的指针，用于实现堆中元素的存储

int currentSize; 堆的当前大小

int maxHeapSize; 堆的最大大小（即容量）

2.3.5 私有成员函数

void FilterDown(int start,int end_of_heap);

从堆底向上调整堆，恢复最小堆性质。

void FilterUp(int start);

从堆底向上调整堆，恢复最小堆性质。

2.3.6 公有成员函数

bool Insert(const T& x);

向堆中插入一个新元素；

bool Remove(T& x);

删除堆顶元素；

bool isEmpty()const;

检查堆是否为空；

bool isFull()const;

检查堆是否已满；

void makeEmpty(){delete[] heap; currentSize=0;}

清空堆中元素，释放堆数组所占内存资源；

int Size();

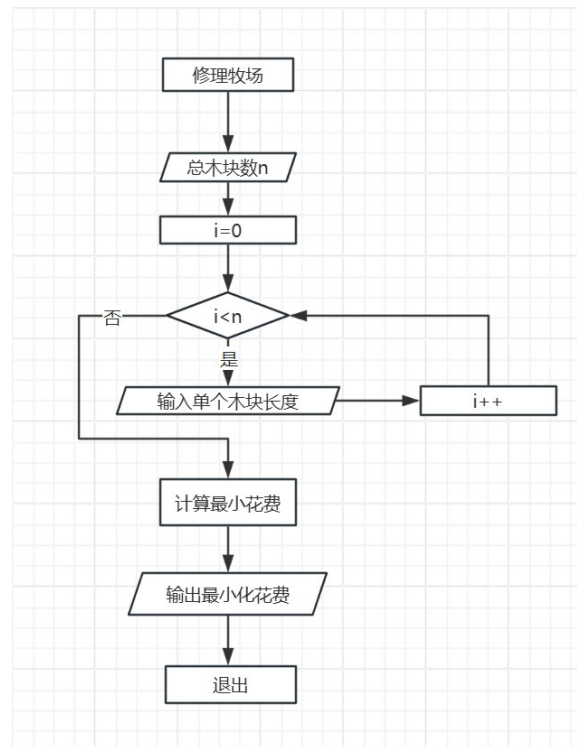
获取当前堆的大小；

MinHeap<T>& operator=(const MinHeap<T>& R);

重载=操作符，实现堆对象的深拷贝

2.4 项目框架设计

2.4.1 项目框架流程图



2.4.2 项目框架流程

1. 进入牧场修理。
2. 输入木材分段的数量及每段的长度。
3. 借助贪心算法和最小堆计算最小花费；
4. 输出最小化非并退出程序。

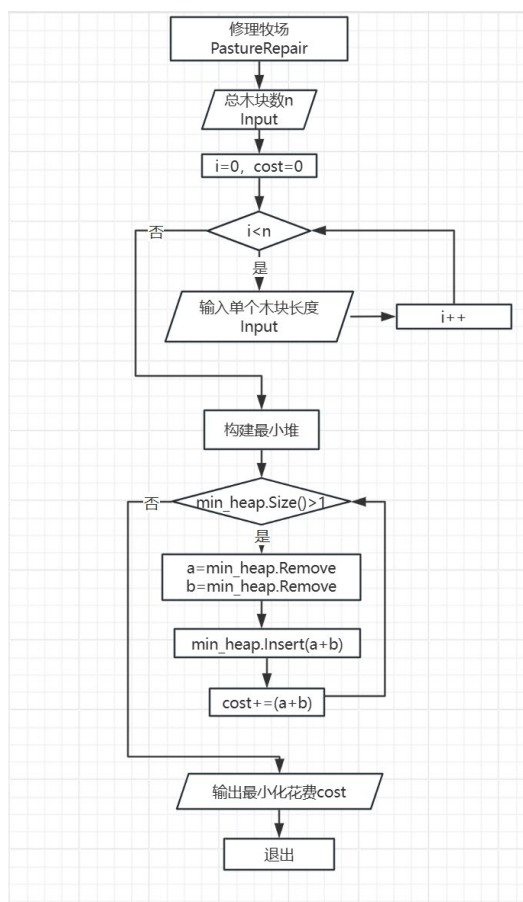
第3章 项目功能实现

3.1 项目主体功能实现

3.1.1 实现思路

1. 进入 PastureRepair 函数以进入考试报名系统。
2. 调用 Input 函数输入木头需要被锯成的块数；
3. 根据快熟，多次调用 Input 函数输入每一段木头的长度。
4. 进入循环 while，开始处理最小堆，依次从最小堆中取出两个最小元素，将其相加，结果加入总费用中，并插入最小堆中；当最小堆中元素为一个时，退出循环，此时总费用即为最小费用；
5. 输出最小费用；
5. 退出系统。

3.1.2 流程图



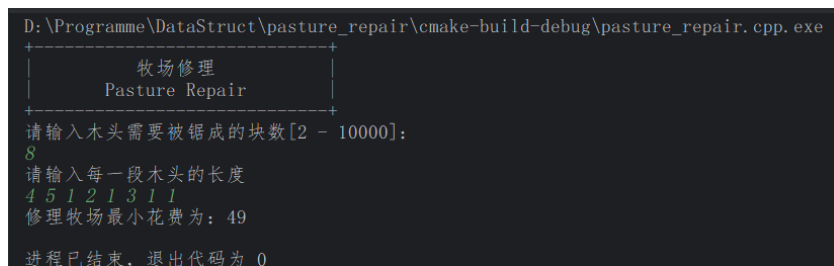
3.1.3 核心代码

```

void PastureRepair()
{
    std::cout << "+-----+\n";
    std::cout << "|      牧场修理      |\n";
    std::cout << "|      Pasture Repair  |\n";
    std::cout << "+-----+\n";
    const int N=Input(false,"请输入木头需要被锯成的块数[2 - 10000]: ");
    auto nums = new(std::nothrow) int[N];
    assert(nums!=nullptr);
    std::cout << "请输入每一段木头的长度\n";
    for (int i = 0; i < N; i++)
        nums[i] = Input(true);
    MinHeap<int> min_heap(nums, N);
    int cost=0;
    while(min_heap.Size()>1) {
        int x,y;
        min_heap.Remove(x);
        min_heap.Remove(y);
        int total=x+y;
        cost+=total;
        min_heap.Insert(total);
    }
    std::cout<<"修理牧场最小花费为: "<<cost<<"\n";
    delete [] nums;
}

```

3.1.4 示例



```

D:\Programme\DataStruct\pasture_repair\cmake-build-debug\pasture_repair.cpp.exe
+-----+
|      牧场修理      |
|      Pasture Repair  |
+-----+
请输入木头需要被锯成的块数[2 - 10000]:
8
请输入每一段木头的长度
4 5 1 2 1 3 1 1
修理牧场最小花费为: 49
进程已结束，退出代码为 0

```

3.2 异常处理功能

3.2.1 输入非法的异常处理

在木头需要被锯成的块数以及每一段木头的长度时，需要保证输入的数据为正整数，同时，木头需要被锯成的块数还应当大于等于 2 并小于等于 1000；在输入时，统一借助 Input 函数，通过传入参数的不同来确定错误判断逻辑。

具体实现时，通过一个 while 无限循环来确保输入的数据无误；当输入错误时，会清除缓冲区，继续执行循环函数；当输入正确时，则会通过 break 退出循环，返回当前输入的数。

实现代码如下：

```
int Input(const bool ret,const char* str=nullptr)
{
    double x;
    while(true) {
        if(!ret)
            std::cout<<str<<"\n";
        std::cin >> x;
        if (std::cin.fail() || x!=static_cast<int>(x)|| x<=0 || (!ret && (x <= 1 || x >
MAX))) {
            std::cout << "输入非法，请重新输入！ \n";
            std::cin.clear();// 清除错误标志
            std::cin.ignore(2147483647, '\n');
            continue;
        }
        break;
    }
    return static_cast<int>(x);
}
```

3.2.2 动态内存申请失败的异常处理

在进行 MinHeap 类与主函数中申请动态内存时，程序使用 new(std::nothrow) 来尝试分配内存。new(std::nothrow)在分配内存失败时不会引发异常，而是返回一个空指针（NULL 或 nullptr），代码检查指针是否为空指针，

如果为空指针，意味着内存分配失败，对于内存分配失败，可以通过 `assert` 断言抛出异常：

```
template<typename T>
MinHeap<T>::MinHeap(int maxSize)
{
    maxHeapSize =  maxSize>DefaultSize?maxSize:DefaultSize;
    heap = new(std::nothrow) T[maxHeapSize];
    assert(heap!=nullptr);
    currentSize = 0;
}
```

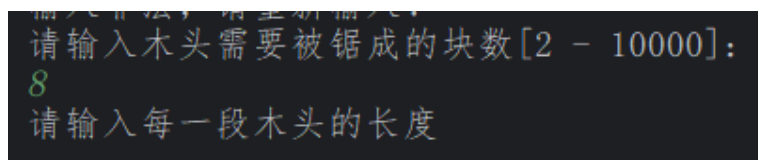
3.2.3 最小堆已满或为空的异常处理

在 `MinHeap` 类中，`Insert` 和 `Remove` 函数在对最小堆数组进行插和删除工作时，会分别遇到最小堆已满或最小堆为空的情况，会使程序陷入异常。

为此，在插入时和删除时，分别使用 `assert` 断言，判断堆是否已满或为空，及时抛出异常信息，便于程序修改。

第4章 项目测试

4.1.1 木头需要被锯成的块数输入功能测试



分别输入超过类型上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理。

```
请输入木头需要被锯成的块数[2 - 10000]:
8
请输入每一段木头的长度
-999999999999
输入非法, 请重新输入!
999999999999
输入非法, 请重新输入!
-5
输入非法, 请重新输入!
2.2
输入非法, 请重新输入!
a
输入非法, 请重新输入!
casc
输入非法, 请重新输入!
```

4.2 最低费用计算功能测试

当输入正确数据时, 可以计算出正确结果。

```
+-----+
|           |
|  牧场修理  |
| Pasture Repair  |
|           |
+-----+
请输入木头需要被锯成的块数[2 - 10000]:
2
请输入每一段木头的长度
1 6
修理牧场最小花费为: 7
```

```
+-----+
|           |
|  牧场修理  |
| Pasture Repair  |
|           |
+-----+
请输入木头需要被锯成的块数[2 - 10000]:
9
请输入每一段木头的长度
1 5 3 8 4 5 6 2 7
修理牧场最小花费为: 124
```

第 5 章 相关说明

5.1 编程语言

本项目全部.cpp 文件以及.h 文件均使用 C++编译完成，使用 UTF-8 编码。

5.2 Windows 环境

Windows 系统：Windows 11 x64

Windows 集成开发环境：CLion 2024

工具集：MinGW 11.0 w64

5.3 Linux 环境

基于 Linux 内核的操作系统发行版：Ubuntu 24.04.1

Linux 命令编译过程为：

1. 定位包含项目所在文件夹，包括.pp 与.h 文件；具体命令为：cd /home/bruce/programe/ pasture_repair

- 2.编译项目，生成可执行文件；具体命令为：g++ -static -o pasture_repair_linux pasture_repair.cpp my_min_heap.h;

其中指令含义分别为：

g++: 调用 GNU 的 C++编译器

-static: 使用静态链接而非动态链接，将所有依赖库直接嵌入到可执行文件，文件存储空间变大，但可以单独运行

-o pasture_repair_linux: -o 表示输出文件选项，pasture_repair_linux 为可执行文件名

pasture_repair.cpp my_min_heap.h:编译所需要的文件。

- 3.运行可执行文件；具体命令为 ./ pasture_repair_linux


```
bruce@Jarvis: ~/programe/pasture_repair
bruce@Jarvis:~/programe/pasture_repair$ cd /home/bruce/programe/pasture_repair
bruce@Jarvis:~/programe/pasture_repair$ g++ -static -o pasture_repair_linux pasture_repair.cpp my_min_heap.h
bruce@Jarvis:~/programe/pasture_repair$ ./pasture_repair_linux
+-----+
|      牧场修理      |
|      Pasture Repair  |
+-----+
请输入木头需要被锯成的块数 [2 - 10000]:
█
```