修理牧场

作 者 姓 名： 李翠琪

学 号： 1751022

指 导 教 师： 张颖 \_\_

学院、 专业： 软件工程 \_\_

同济大学

Tongji University

目录

[1 分析 1](#_Toc23490280)

[1.1 项目简介 1](#_Toc23490281)

[1.2 解题分析 1](#_Toc23490282)

[2 设计 2](#_Toc23490283)

[2.1 优先队列设计 2](#_Toc23490284)

[2.2 主程序设计 2](#_Toc23490285)

[3 实现 2](#_Toc23490286)

[3.1 优先队列的实现 2](#_Toc23490287)

[3.1.1 堆构造函数的实现 2](#_Toc23490288)

[3.1.2 添加元素入堆 3](#_Toc23490290)

[3.1.3 删除堆中最小元素 3](#_Toc23490291)

[3.1.4 sink函数的实现 3](#_Toc23490292)

[3.1.5 swim函数实现 4](#_Toc23490293)

[3.2 主函数的实现 4](#_Toc23490294)

[4 测试 5](#_Toc23490295)

# 1 分析

## 1.1 项目简介

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。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）.

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。

输出格式：输出一个整数，即将木头锯成N块的最小花费。

## 1.2 解题分析

给了要将木头锯成N块,与每块木头的长度。最后需要输出最小的花费。

刚开始时，我的思路是因为是从一块木头开始分割的，且每次都将木块分割为两块，无非是大小不同而已。所有这道题等价与用n个元素构建一个二叉树,n个元素在树的最底层，并且使得除了root外,其他节点相加最小。但是到这里就卡住了，就不懂怎么继续进行。于是上网查了文档，它提供了另外一种思路：通过反向思考,如果每次对半分的时候,最后结果最小，那么,在只有碎片的时候,每一次都拿出最小的两个进行拼凑，就是对半分的逆过程了， 所以只需要有一个每次都能取出最小元素的优先队列即可，所有就只要构建一个优先队列就能解决问题了。

# 2 设计

## 2.1 优先队列设计

设计一个每次都能取出最小元素的优先队列即可。总体思路是，每一次往优先队列加入元素时，把元素放在尾部，然后让这个元素“上浮”到合适的位置,构成一个堆,堆顶为最小元素。每次取出元素时，取出堆顶元素，然后让堆顶元素与最后一个元素交换，并且让堆的数量减一，堆顶元素“下沉”。这样，不管是取出元素还是加入元素，都构成了一个堆。

## 2.2 主程序设计

首先利用输入的N个正整数逐个放入堆中，然后不断循环直到堆只剩下一个元素。在循环中，每次都从堆中取出最小的两个元素，让总的花费sum+=他们两个的值。并且把他们两个加起来的值继续放入堆中。

直观上来说，就是在一堆木块碎片中，每次都把最小的两块碎片拼成一块，不断的拼，直到最后就只剩下最初的完整的木块了。

# 3 实现

## 3.1 优先队列的实现

### 3.1.1 堆构造函数的实现

### 当构造一个堆时，需要输入一个堆中应该存的元素数量最大值len，令堆中的成员变量数组的长度为len+1，（因为不用数组的vec[0]），堆的这个类还有一个N记录当前堆的元素数量。

MinPQ(int len):pq(len+1)  
{  
  
}

private:  
 vector<int> pq;  
 int N;//元素个数,从1开始

### 3.1.2 添加元素入堆

令N+1,并且在N处添加元素，然后将这个元素swim，使得还是一个堆。

void push(int i)  
{  
 pq[++N] = i;  
 swim(N);  
}

### 3.1.3 删除堆中最小元素

最小元素在堆顶，先用一个变量存储堆顶的值，将堆顶与堆中最后一个元素交换，令N-1，此时不是一个堆，需要将堆顶sink到合适的位置，最后返回之前存的堆顶元素的值。

int delMin()  
{  
 int max = pq[1];  
 exch(1, N--);  
 sink(1);  
 return max;  
}

### 3.1.4 sink函数的实现

Sink的思路是将这个元素与他的子节点做对比，并且将较小的放上面，一直对比到这个元素小于他的两个子节点为止。

具体实现是，假如要下沉k，则先找它的子节点2\*k与2\*k+1中最小的一个（当然得判断是否存在），如果小于他的子节点的值，那么就交换他们的值，并且让k=2\*k或2\*k+1（＝最小的那个），这是为了继续追踪原本的k，继续和他此时的两个子节点做对比。不断的循环，直到他没有子节点或者他小于了子节点。

void sink(int k)  
 {  
 while(2 \* k <=N)  
 {  
int j = 2 \* k;  
 if(j+1<=N && pq[j]>pq[j+1])  
 {  
 ++j;  
 }  
 if(pq[k]<=pq[j])  
 {  
 break;  
 }  
exch(k, j);  
k = j;  
 }  
 }

### 3.1.5 swim函数实现

直观看就是一个元素比较轻，然后它要不断上浮，直到它重与他的父节点。而代码的实现就是，查看他的父节点k/2的值是否小于它，如果不小于它，则交换它和它父节点的值，并且让k=k/2，即继续追踪这个元素，让它继续与它当前的父节点做对比。直到它本身为数组的第一个元素或者它不小于它的父节点。

void swim(int k)  
{  
 while(k>1 && pq[k] < pq[k/2])  
 {  
 exch(k, k / 2);  
 k /= 2;  
 }  
}

## 3.2 主函数的实现

基本和之前的设计描述差不多，就是构造堆，不断的删除两个最小元素，并且把他们的和继续加入堆中，不断循环，只剩下一个元素的时候终止。

int main()  
{  
 cout << "输入正整数N(分成N块,N<1024),及N个正整数(每块长度):";  
 int N;  
 cin >> N;  
 int sum = 0;  
 MinPQ minPQ(N);  
 for (int i = 0; i < N; ++i)  
 {  
 int len;  
 cin >> len;  
 minPQ.push(len);  
  
 }  
 while(minPQ.size()>=2)  
 {  
 int min1=minPQ.delMin();  
 int min2=minPQ.delMin();  
 sum += min1 + min2;  
 minPQ.push(min1 + min2);  
 }  
 cout << sum;  
 return 0;  
}

# 4 测试



