数据结构课程设计

项目说明文档

修理牧场

软件工程

张靖凯

2151396

目录

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 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）.

项目功能要求

（1） 输入格式：输入第一行给出正整数 N（N《10^4），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。

项目示例

项目设计

数据结构设计

仿照 STL，自行实现了 priority_queue，是一个容器适配器。

```
template<typename T, typename Container = Vector<T>, typename compare = std::less<T>>
class Priority_queue
```

T 为值类型，Container 是容器类型，默认为自行实现的 Vector，compare 是 function object，默认为 std 中的 less 类型。

Priority_queue 类

Public Member Functions

	Priority_queue ()
	~Priority_queue ()
void	**push** (const T &key)
void	**pop** ()
T	**top** ()
void	**clear** ()
size_t	**size** ()
bool	**empty** ()

Private Attributes

size_t	**_size** = 0 容量大小 More...
Container	**_pq** 以vector实现 More...
compare	**_cmp** function object More...

Vector 类

void	**push_back** (**value_type** val)
void	**pop_back** ()
size_t	**Size** () const
size_t	**capacity** () const
bool	**empty** ()
void	**clear** ()
value_type	**front** () const
value_type	**back** () const
void	**insert** (**iterator** it, **value_type** val)
void	**erase** (**iterator** it)

Private Attributes

value_type *	**_data** 动态分配实现的数组 More...
size_t	**_size** 已有元素数量 More...
size_t	**_capacity** 容器容量 More...

数据结构关键代码

Pop 函数：同堆操作，自顶向下循环调整顺序。

```
//将末尾的元素换到头部，末尾的 pop 出去
_pq[0] = _pq[_size - 1];
_pq.pop_back();
//然后下沉新换到头部的元素
_size--;
int idx = 0;
T tmp = _pq[0];
int leftChild = 2 * (idx + 1) - 1;
int rightChild = 2 * (idx + 1);
while (leftChild < _size)
{
//child,记录下沉到左子节点还是右子节点，还是不动
int child = idx;
if (_cmp(tmp, _pq[leftChild]))
    child = leftChild;
if (rightChild < _size && _cmp(_pq[child], _pq[rightChild]))
    child = rightChild;
    if (idx == child)
       break;
    _pq[idx] = _pq[child];//将 child 浮上去，child 的位置暂时给 tmp
    idx = child;
    leftChild = 2 * (idx + 1) - 1;//如果 idx 还存在左右子节点，继续；
否则退出
    rightChild = 2 * (idx + 1);
}
_pq[idx] = tmp;
```

Push 函数同上，是从下到上调整顺序：

```
//上浮，key 大于其父节点，即上浮（将父节点换到下面），idx 的父节点是(idx
- 1) / 2
while (idx > 0 && _cmp(_pq[(idx - 1) / 2], key))
{
   _pq[idx] = _pq[(idx - 1) / 2];//将父节点换到下面
   idx = (idx - 1) / 2;//更新 idx，直到 idx 的父节点大于 key，停止
}
_pq[idx] = key;//将 key 放到正确的位置
```

解题思路

利用**贪心算法**思想，将所有数据入队，当优先队列不为空的时候，每次弹出两个元素，因为是优先队列，所以弹出的元素是队列中最小的两个，然后求和加入答案，直到队列为空。最终 sum 的值即为答案。

解题关键代码

```
    while (q.size() > 1)
    {
        a = q.top();
        q.pop();
        a += q.top();
        q.pop();
        q.push(a);
        sum += a;
    }
    cout << sum;
```

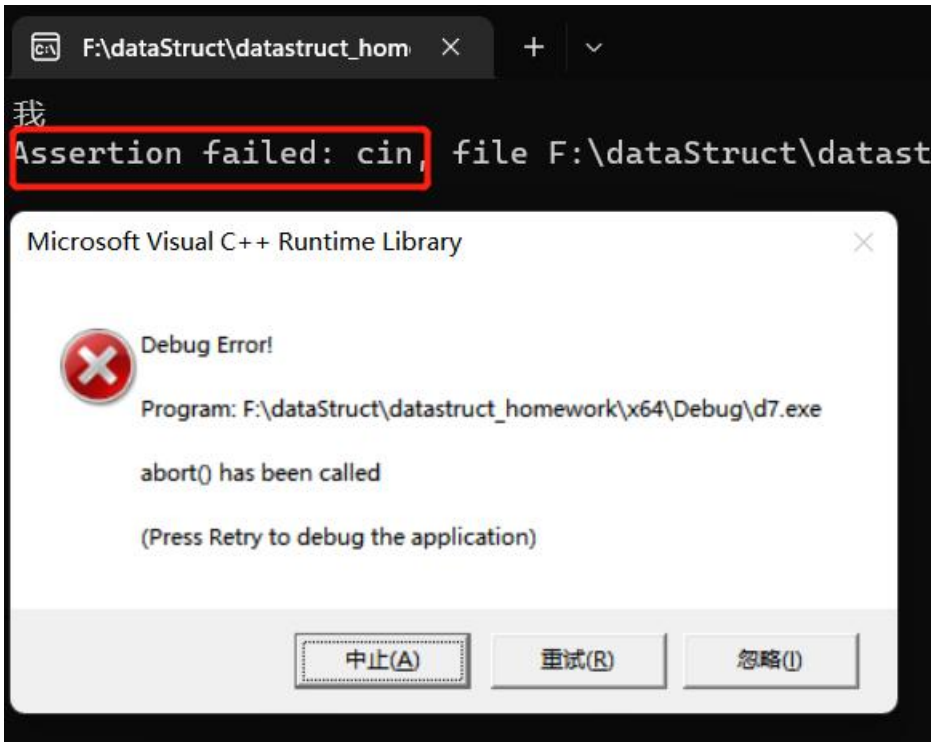
设计亮点

代码注释规范

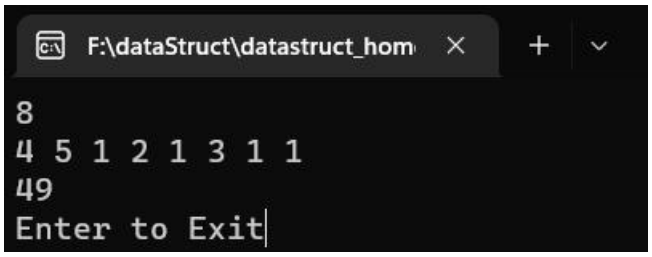
采用了 doxygen 注释规范，对类、函数等有简要说明，命名采用驼峰命名法，类内的各个声明规范，方便本人回顾之前写过的代码和 code reviewer 查看，使 API 规范，帮助这个开发流程高效、规范地进行。

```
/// @brief 仿照STL用堆思想实现优先队列
template<typename T, typename Container = Vector<T>, typename compare = std::less<T>>
class Priority_queue
{
private:
    size_t _size = 0;   ///< 容量大小
    Container _pq;      ///< 以vector实现
    compare _cmp;       ///< function object
```

输入错误处理



项目测试



Linux 测试运行（Ubuntu）

```
kk@LAPTOP-UJDPHKT8:~/dataStruct$ ls
homework1  homework2  homework3  homework4  homework5  homework6  homework7  test
kk@LAPTOP-UJDPHKT8:~/dataStruct$ cd homework7
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework7$ ls
Priority_queue.hpp  Vector.hpp  h7.cpp  run
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework7$ ./run
8
4 5 1 2 1 3 1 1
49
Enter to Exit
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework7$
```