### OOP第九周作业文档

2019010175 孔瑞阳 土木92

### 项目信息

**1、功能说明**

(1)

输入第一行一个正整数n，表示排序的数的个数。

输入第二行n个整数，表示需要排序的数。

输出第一行不去重排序的结果，用sort()实现。

输出第二行 去重排序的结果，用set 实现。

(2)

模仿vector实现了一个MyVector,具有以下主要功能：

|  |  |
| --- | --- |
| MyVector() {} | 默认构造 |
| MyVector(size\_t n, const T& value = T()) | 赋值构造 |
| MyVector(MyVector<int>& vec) | 拷贝构造 |
| MyVector(iterator begin, iterator end) | 复制构造 |
| ~MyVector() | 析构 |
| MyVector<T>& operator=(const MyVector<T>& vec) | 赋值 |
| void swap(MyVector<T>& vec) | 交换 |
| iterator begin() | 最前面的元素的迭代器 |
| iterator end() | 最后面的元素的后面一个迭代器 |
| size\_t capacity() | 容量 |
| size\_t size() | 长度 |
| void reserve(size\_t n) | 修改容量 |
| void resize(size\_t n, const T& value = T()) | 修改长度 |
| void clear() | 清空 |
| bool empty() | 判断是否为空 |
| T& at(size\_t pos) | 访问某个位置的引用（成员函数形式） |
| T& operator [](size\_t pos) | 访问某个位置的引用（数组形式） |
| T& front() | 第一个位置的引用 |
| T& back() | 最后一个位置的引用 |
| void insert(iterator pos, const T& x) | 在某一位置插入 |
| void push\_back(const T& x) | 在最后一个位置插入 |
| iterator erase(iterator pos) | 删除某个位置 |
| void pop\_back() | 删除最后一个位置 |

### **2、测试环境**

|  |  |
| --- | --- |
| CPU | Intel(R) Core(TM)i7-9750H CPU @ 2.6Ghz 6核12线程 |
| GPU | NVIDIA GeForce RTX2070 |
| RAM | DDR4 16G+16G |
| Operating System | Microsoft Windows 版本1909 |
| Compiler | MSVC++ 14.24 |

### 验证

##### 排序的验证

1. 手动验证

分为没有重复数据和有重复数进行验证。

|  |  |  |
| --- | --- | --- |
| **数据** | **不去重排序（sort）** | **去重排序（set）** |
| 9 6 6 4 7 5 8 7 1 2 | 1 2 4 5 6 6 7 7 8 9 | 1 2 4 5 6 7 8 9 |
| 10 10 10 10 10 10 10 10 | 10 10 10 10 10 10 10 10 | 10 |
| 0 9 7 3 1 4 6 8 2 5 | 0 1 2 3 4 5 6 7 8 9 | 0 1 2 3 4 5 6 7 8 9 |

1. 对拍验证

用unique对用sort进行不去重排序之后的数组进行去重，并与用set进行去重排序之后的结果进行比较，如果元素个数相同、结果相同的话，那么就可以认为两种排序算法都是正确的。

进行了10分钟的对拍，大概进行了600次验证，都没有出现错误，可以基本认为两种程序都是正确的。

##### MyVector的验证

|  |  |
| --- | --- |
| **操作** | **MyVector中的元素** |
| push\_back(0)...push\_back(9) | 0 1 2 3 4 5 6 7 8 9 |
| 5次pop\_back | 0 1 2 3 4 |
| 拷贝构造 V(v) | 0 1 2 3 4 |
| erase(v.begin()) | 1 2 3 4 |
| insert(v.end() - 2, 5) | 1 2 5 3 4 |
| V.front() = 5, V.back() = 3, V[1] = 4, V.at(3) = 4 | 5 4 5 4 3 |
| empty(), clear(), empty() | (FALSE TRUE)(0 1) |
| 复制构造 w(v.begin() + 1, v.end() - 1) | 1 2 3 |
| resize(5,4) | 1 2 3 4 4 |
| resize(4) | 1 2 3 4 |
| V.swap(w) | 1 2 3 4 |
| v = V | 1 2 3 4 |
| 默认构造 W |  |
| 赋值构造 WW(6, 6) | 6 6 6 6 6 6 |

另外，析构、size()、capacity()、reerve()，也都在上述操作和输出的过程中涉及了。

至此，所有操作的初步正确性验证完毕。

##### MyVector和vector的效率比较

选取了push\_back, pop\_back, insert, erase, 赋值构造这几个操作进行了比较。

分 不开编译优化，和O2编译优化进行比较。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **操作** | **MyVector** | **MyVector (O2)** | **vector** | **Vector (O2)** |
| 10^7次push\_back | **1.403s** | **1.229s** | 3.529s | 3.326s |
| 10^7次pop\_back | **0.404s** | **0.349s** | 1.58s | 1.34s |
| 100次插入10^4个数 | 5.799s | 6.871s | **2.336s** | **3.115s** |
| 100次删除10^4个数 | 5.505s | 6.838s | **1.676s** | **2.277s** |
| 100次构造10^6个数 | **0.763s** | **0.733s** | 13.058s | 13.595s |

可以发现，最基础的push\_back和pop\_back操作，MyVector的时间运行效率是vector的大概2.5倍，并且，O2编译优化有一定的优化效果。

但是对于insert和erase操作，MyVector的效率大概只有vector的80%，并且发现O2编译优化似乎出现了比较严重的负优化……

对于赋值构造MyVector(size\_t n, const T& value = T())操作，MyVector的速度大概是vector的20倍，O2优化的效果不明显，或者有较小的负优化。

总的来说，对于构造、push\_back和pop\_back，手写的MyVector的运行效率较好，并且O2编译优化有一定的效果。对于insert和erase操作，虽然vector比MyVector快速，但是在实际运用当中，因为insert和erase都是线性复杂度，如果操作数多的话就会选择平衡树类的数据结构。所以，总体来说，MyVector的效率要比vector更高。