小技巧笔记

STL: map and unordered_map

https://blog.csdn.net/u013550000/article/details/80521509

unordered_map 的使用方法和 map 没有太多差，他们的实现机理不同罢了。map 是通过红黑树组织的，而 unordered_map 是通过哈希表。后者的内存占用多但是查询速度快，前者内存占用少，但是查询速度较后者更慢一些。

1. Pair 简介

   pair 是将 2 个数据组合成一组数据，当需要这样的需求时就可以使用 pair，如 stl 中的 map 就是将 key 和 value 放在一起来保存。

   https://blog.csdn.net/sevenjoin/article/details/81937695
2. insert()成员函数，插入数据
3. size()成员函数，查看已经插入了几个数据
4. 遍历：

   map<int, string>::iterator iter; or map<int, string>::reverse_iterator iter;

   for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)

   iter->first……

   iter->second……

   ……

   or

   for(iter = mapStudent.rbegin(); iter != mapStudent.rend(); iter++)

   ……
5. 增删查看上面的连接，需要注意的是如果以结构体作为 key 需要在结构体内重载一下小于操作符

   a) 查，就是 find 或者 count

   dict.find(key); //返回一个 iterator

   dict.count(key); //有就返回 1，没有就返回 0

   b) 增，就是 insert：

   dict.insert(pair<int, string>(1, “Eric”));

   还可以用数组的方式插入啦

   还可以类似 Python 的 dict 一样插入啦，比如直接 m[“ass”] = 2;

   c) 删，就是 erase：

   先查找出要删除的迭代器 it

   dict.erase(it);

   dict.erase(left_it, right_it); //也可以

   dict.clear(); //全删

   也可以直接 dict.erase(KeyType key); //这样来删也可以

   d) 改：可以根据 Python dict 那样的方法来改：

   比如说已经有有了：m[“Eric”] = 99; m[“Sophia”] = 98;

   可以用 m[“Eric”] = 100; 来修改键对应的值。

   甚至可以 m[“Eric”] += 1; 这样来修改欸！！！

还有一个比较特别的性质，那就是重载的操作符[]。当你创建了一个空的map<keyType, ValueType> dict 的时候，你可以直接用下标[]去访问一个元素，若这个元素不存在，那么 STL 会自动插入一个此元素，并且调用 ValueType 的默认构造函数给它赋初值。比如 value 是 int 就为 0，value 为 string 就是""。这在一些情况下很好用，比如你想用 int 当做 value 来计数，那么可以直接用 dict[key] += 1; 这样来写，就不用先判断是否存在这个元素，若不存在赋 1 了。当然，如果你想判断某键值对应的元素是否存在，最好不要用 dict[key] == 0 这样，因为它会插入一个键值对进去，会降低后续的性能，最好还是用 count 或者 find 来判断。

STL: string

1. 构造函数：
   string a("Eric");
   string a("Eric"), b("Erika");
2. 判断相等时可以直接用:
   if(a == "Eric")
3. find 系列函数：
   s.find(string/char*/char substr, int start)
   从源字符串的下标为 start 的地方开始找，找到了就返回所在子串的起始位置下标，没有找到返回 string::npos
   s.rfind(string/char*/char substr, int start)
   从 start 位置开始向前找
   s.find_first_of(string/char*/char substr, int start)
   从 start 开始找到 s 中，与 substr 中任意一个字符相同的字符所在的位置
   s.find_last_of(string/char*/char substr, int start)
   从 start 开始反正找到 s 中，与 substr 中任意一个字符相同的字符所在的位置
4. 连接两个 string 类型直接用+（加号）
5. 取子字符串：
   s.substr(int start, int offset)
   从 s 中取出下标从 start 开始长度为 offset 的子串。
6. cin 一个 string 不能读空格，但是可以用 getline()函数来读取一行。

```
int main()
{
    string test;
    getline(cin, test);
    cout << test << endl;
    return 0;
}
```

STL：堆

STL 的堆不是一个库，而是结合了 vector（可迭代数据结构）和 algorithm 这个库。

1. 建立堆：

   void make_heap (RandomAccessIterator first, RandomAccessIterator last);

   void push_heap (RandomAccessIterator first, RandomAccessIterator last,

   Compare comp);

   默认生成最大化堆，comp 函数可以生成最小化堆。

   //建立小顶堆

   cout<<"(greater) make_heap:"<<endl;

   make_heap(nums.begin(),nums.end(),greater<int>());

2. 添加数据：

   void push_heap (RandomAccessIterator first, RandomAccessIterator last);

   void push_heap (RandomAccessIterator first, RandomAccessIterator last,

   Compare comp);

3. 删除数据：

   void pop_heap (RandomAccessIterator first, RandomAccessIterator last);

   void pop_heap (RandomAccessIterator first, RandomAccessIterator last,

   Compare comp);

   pop_heap()并没有删除元素，而是将堆顶元素和数组最后一个元素进行了替换，如果要删除这个元素，**还需要对数组进行 pop_back()操作**。

NMD，好像这个堆操作的效率很低啊？？？

STL：集合 set

集合可以理解为只有键值的 map，其底层的数据结构也是一棵平衡二叉树（红黑树），其支持高效的插入、删除操作，复杂度为 O(logN)。

1. 初始化：

   set<int> s;

2. 插入数据：

   s.insert(2);

3. 删除数据：

   s.delete(2);

   s.clear(); //清空

4. 查找操作（判断有没有）

   s.find(2) != s.end()如果为 true 表示该元素存在于集合中

   s.count(2) == 0 也一样

5. 遍历：

   set<double>::iterator it;

   for(it=tmpset.begin(); it!=tmpset.end(); it++)

   cout << *it << endl;

STL：优先级队列

1. 优先级队列有三个模板参数<type, container, cmp>

   第一个参数是元素的类型，可以是结构体、类等

第二个参数是容器，可以是 vector<type>等
第三个参数是 cmp 函数，更 cmp 的就排在后面（我也不知道为什么要是这种机制，反正就是这样！记住就好了！）。默认的第二个参数是 vector，第三个参数是小于，即小的排在后面，也就是最大化堆（大的具有优先级）

2. 当用自己写的类作为优先级队列的元素类型时，需要在类中重载一下比较函数。注意一下这个比较函数重载小于号就好了，**但是如果想用最小化堆，就在实现中写大于的逻辑即可**。
3. push、top、pop、empty 和 clear 等函数和普通的队列差不多