

C++常用语法

1. 常用数据结构

1. `std::pair`

初始化：可以用构造函数来初始化。

例题

987. 二叉树的垂序遍历

困难



256



相关企业

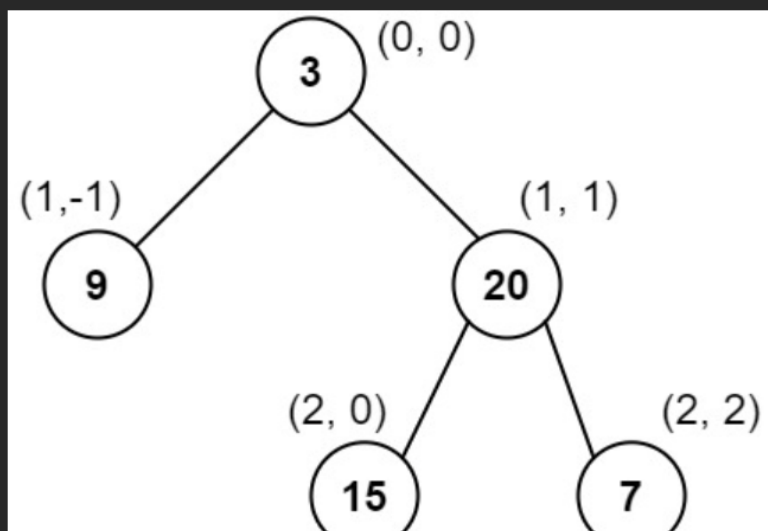
给你二叉树的根结点 `root`，请你设计算法计算二叉树的 **垂序遍历** 序列。

对位于 (row, col) 的每个结点而言，其左右子结点分别位于 $(row + 1, col - 1)$ 和 $(row + 1, col + 1)$ 。树的根结点位于 $(0, 0)$ 。

二叉树的 **垂序遍历** 从最左边的列开始直到最右边的列结束，按列索引每一列上的所有结点，形成一个按出现位置从上到下排序的有序列表。如果同行同列上有多个结点，则按结点的值从小到大进行排序。

返回二叉树的 **垂序遍历** 序列。

示例 1:



```
1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8  *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9  *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
    right(right) {}
10  * };
11  */
12 class Solution {
13 public:
14     vector<vector<int>> verticalTraversal(TreeNode* root) {
15         deque<std::pair<TreeNode*, vector<int>>> q;    // vector:{row, col}
```

```

16         unordered_map<int, vector<vector<int>>> col_map;    // key:col pair:
    {val, row}
17         priority_queue<int, vector<int>, greater<int>> id_col;
18         vector<vector<int>> ans;
19         // bfs, construct the hash map while traveling
20         q.push_back(std::make_pair(root, vector<int>(2,0)));
21
22         while(!q.empty())
23         {
24             auto cur_node = q.back().first;
25             auto position = q.back().second;
26             q.pop_back();
27             if(col_map.find(position[1]) != col_map.end())
28             {
29                 // already exist
30                 col_map[position[1]].push_back({cur_node->val, position[0]});
31             }
32             else
33             {
34                 // does not exist
35                 col_map[position[1]] = {{cur_node->val, position[0]}};
36                 // add into the heap
37                 id_col.push(position[1]);
38             }
39
40             if(cur_node->left)
41             {
42                 /** 看这里!! 这里用std::make_pair初始化会报错 **/
43                 q.push_back(std::pair<TreeNode*, vector<int>>(cur_node->left,
    {position[0]+1, position[1]-1}));
44             }
45             if(cur_node->right)
46             {
47                 q.push_back(std::pair<TreeNode*, vector<int>>(cur_node->right,
    {position[0]+1, position[1]+1}));
48             }
49         }
50
51         // sort each kv pair in the hash map
52         while(!id_col.empty())
53         {
54             auto cur_list = col_map[id_col.top()];
55             id_col.pop();
56
57             std::sort(cur_list.begin(), cur_list.end(), [](const vector<int>
    &a, const vector<int> &b) {
58                 if (a[1] != b[1]) {

```

```
59         return a[1] < b[1];
60     } else {
61         return a[0] < b[0];
62     }
63 });
64
65     vector<int> s;
66     for(auto each:cur_list)
67     {
68         s.push_back(each[0]);
69     }
70     ans.push_back(s);
71 }
72 return ans;
73 }
74 };
```


2. deque: 双端队列

例题

1696. 跳跃游戏 VI

提示

中等   200  

 相关企业

给你一个下标从 0 开始的整数数组 `nums` 和一个整数 `k`。

一开始你在下标 0 处。每一步，你最多可以往前跳 `k` 步，但你不能跳出数组的边界。也就是说，你可以从下标 `i` 跳到 `[i + 1, min(n - 1, i + k)]` 包含两个端点的任意位置。

你的目标是到达数组最后一个位置（下标为 `n - 1`），你的 **得分** 为经过的所有数字之和。

请你返回你能得到的 **最大得分**。

示例 1:

输入: `nums = [1,-1,-2,4,-7,3]`, `k = 2`

输出: 7

解释: 你可以选择子序列 `[1,-1,4,3]`（上面加粗的数字），和为 7。

示例 2:

输入: `nums = [10,-5,-2,4,0,3]`, `k = 3`

输出: 17

解释: 你可以选择子序列 `[10,4,3]`（上面加粗数字），和为 17。

示例 3:

输入: `nums = [1,-5,-20,4,-1,3,-6,-3]`, `k = 2`

输出: 0

解答:

```
1 // 这里实际上是用双端队列对动态规划做出了优化
2 // 其实这里就是所谓的sliding windows那个东西
3 class Solution {
4 public:
5     int maxResult(vector<int>& nums, int k) {
6         int n = nums.size();
7         vector<int> dp(n,0);
8         deque<int> the_queue;
9         dp[0] = nums[0];
10        the_queue.push_back(0);
11
12        for(int i = 1; i < n; i++)
13        {
14            if(the_queue.front() < i - k)
15            {
16                the_queue.pop_front();
```

```

17         }
18         dp[i] = nums[i] + dp[the_queue.front()];
19         while(!the_queue.empty() && dp[the_queue.back()] < dp[i])
20         {
21             // 注意pop_back()本身返回void
22             the_queue.pop_back();
23         }
24         the_queue.push_back(i);
25     }
26
27     // dp[0] = nums[0];
28     // for(int i = 1; i < n; i++)
29     // {
30     //     int maxVal = -1000000000;
31     //     for(int j = -k; j <= -1; j++)
32     //     {
33     //         if(i + j >= 0 && dp[i+j] + nums[i] > maxVal)
34     //         {
35     //             maxVal = dp[i+j] + nums[i];
36     //         }
37     //     }
38     //     dp[i] = maxVal;
39     // }
40     return dp[n-1];
41 }
42 };

```

语法

<https://www.cainiaojc.com/cpp/cpp-deque.html>

```

1 #include <iostream>
2 #include <algorithm>
3 #include <deque>
4
5 using namespace std;
6
7
8 int main()
9 {
10     std::deque<int> i;
11     i.push_back(1);
12     i.push_back(2);
13     i.push_back(3);
14     i.push_back(4);

```

```

15     i.push_back(5);
16     // print i
17     for(auto it = i.begin(); it != i.end(); it++)
18     {
19         std::cout << *it;
20     }
21     std::cout << std::endl;
22     i.pop_front();
23     for(auto it = i.begin(); it != i.end(); it++)
24     {
25         std::cout << *it;
26     }
27     std::cout << std::endl;
28     i.pop_back();
29     for(auto it = i.begin(); it != i.end(); it++)
30     {
31         std::cout << *it;
32     }
33     std::cout << std::endl;
34     // just read the head and tail
35     std::cout << i.front() << std::endl;
36     std::cout << i.back() << std::endl;
37     return 0;
38 }

```

输出:

```

1 "D:\coding\leetcode-brush questions\leetcode-2023autumn\testing\cmake-build-
  debug\testing.exe"
2 12345
3 2345
4 234
5 2
6 4
7
8 进程已结束,退出代码0

```

3. std::vector

语法：如何操作某些特定位置？

Insert

66. 加一



简单



👍 1.4K



🔒 相关企业

给定一个由 **整数** 组成的 **非空** 数组所表示的非负整数，在该数的基础上加一。

最高位数字存放在数组的首位， 数组中每个元素只存储**单个**数字。

你可以假设除了整数 0 之外，这个整数不会以零开头。

示例 1:

输入: digits = [1,2,3]

输出: [1,2,4]

解释: 输入数组表示数字 123。

示例 2:

输入: digits = [4,3,2,1]

输出: [4,3,2,2]

解释: 输入数组表示数字 4321。

```
1 class Solution {
2 public:
3     vector<int> plusOne(vector<int>& digits) {
4         int counter = 1;
5         int n = digits.size();
6         for(int i = n - 1; i >= 0; i--)
7         {
8             int new_this = (digits[i] + counter) % 10;
9             counter = (digits[i] + counter) / 10;
10            digits[i] = new_this;
11            if(!counter)
12                break;
13        }
14
15        if(counter)
16            // here
17            digits.insert(digits.begin(),1);
18        return digits;
19    }
20 };
```


Erase

```
1 #include <vector>
2 #include <iostream>
3
4
5 void print_container(const std::vector<int>& c)
6 {
7     for (int i : c)
8         std::cout << i << ' ';
9     std::cout << '\n';
10 }
11
12 int main()
13 {
14     std::vector<int> c{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
15     print_container(c);
16
17     c.erase(c.begin());
18     print_container(c);
19
20     c.erase(c.begin() + 2, c.begin() + 5);
21     print_container(c);
22
23     // Erase all even numbers
24     for (std::vector<int>::iterator it = c.begin(); it != c.end(); )
25     {
26         if (*it % 2 == 0)
27             it = c.erase(it);
28         else
29             ++it;
30     }
31     print_container(c);
32 }
```

输出:

```
1 0 1 2 3 4 5 6 7 8 9
2 1 2 3 4 5 6 7 8 9
3 1 2 6 7 8 9
4 1 7 9
```

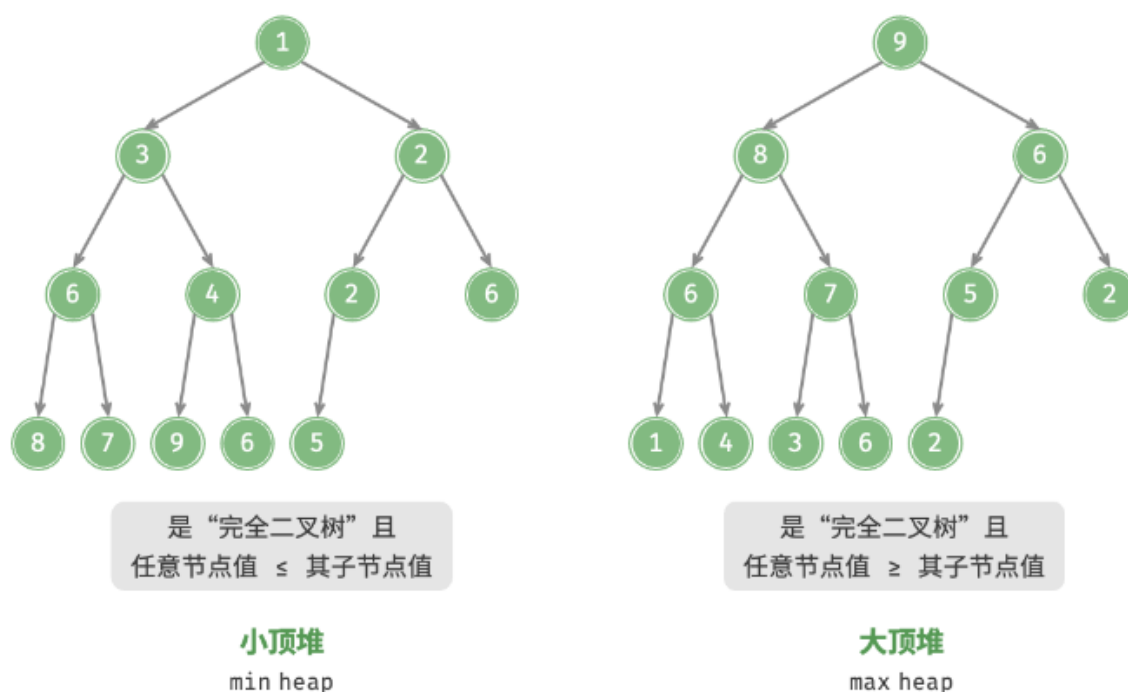
4. C++堆（优先队列）

基础知识

1. <https://www.w3cschool.cn/hellocpp/hellocpp-aibg3tit.html>
2. <https://marian5211.github.io/2017/11/04/c-STL%E4%B8%AD%E5%A0%86%E7%9A%84%E4%BD%BF%E7%94%A8/>

「堆 heap」是一种满足特定条件的完全二叉树，主要可分为图 8-1 所示的两种类型。

- 「大顶堆 max heap」：任意节点的值 \geq 其子节点的值。
- 「小顶堆 min heap」：任意节点的值 \leq 其子节点的值。



堆作为完全二叉树的一个特例，具有以下特性。

- 最底层节点靠左填充，其他层的节点都被填满。
- 我们将二叉树的根节点称为“堆顶”，将底层最靠右的节点称为“堆底”。
- 对于大顶堆（小顶堆），堆顶元素（即根节点）的值分别是最大（最小）的。

需要指出的是，许多编程语言提供的是「优先队列 priority queue」，这是一种抽象数据结构，定义为具有优先级排序的队列。

实际上，堆通常用作实现优先队列，大顶堆相当于元素按从大到小顺序出队的优先队列。从使用角度来看，我们可以将“优先队列”和“堆”看作等价的数据结构。因此，本书对两者不做特别区分，统一使用“堆”来命名。

堆的常用操作见表 8-1，方法名需要根据编程语言来确定。

表 8-1 堆的操作效率

方法名	描述	时间复杂度
push()	元素入堆	$O(\log n)$
pop()	堆顶元素出堆	$O(\log n)$
peek()	访问堆顶元素（大 / 小顶堆分别为最大 / 小值）	$O(1)$
size()	获取堆的元素数量	$O(1)$
isEmpty()	判断堆是否为空	$O(1)$

std::priority_queue 语法

```
1  #include <iostream>
2  #include <algorithm>
3  #include <queue>
4
5  using namespace std;
6
7
8  int main()
9  {
10     priority_queue<int> q;
11     for(int i = 0; i < 10; i++)
12     {
13         q.push(i);
14     }
15     while(!q.empty())
16     {
17         cout << q.top() << endl;
18         q.pop();
19     }
20     std::cout << "Hello, World!" << std::endl;
21     q.push(1);
22     q.push(1);
23     q.push(1);
24     while(!q.empty())
25     {
26         cout << q.top() << endl;
27         q.pop();
28     }
29     return 0;
30 }
```

输出：

```
1 "D:\coding\leetcode-brush questions\leetcode-2023autumn\testing\cmake-build-  
debug\testing.exe"  
2 9  
3 8  
4 7  
5 6  
6 5  
7 4  
8 3  
9 2  
10 1  
11 0  
12 Hello, World!  
13 1  
14 1  
15 1  
16  
17 进程已结束,退出代码0
```

小根堆：

```
1 #include <iostream>  
2 #include <algorithm>  
3 #include <queue>  
4  
5 using namespace std;  
6  
7  
8 int main()  
9 {  
10     // 默认是大根堆，现在构造小根堆  
11     // 三个参数各自的含义：  
12     // 1. 数据类型: int  
13     // 2. 容器类型: 也即是底层容器，这里是vector  
14     // 3. 比较函数: 这里是greater<int>，也即是小根堆  
15     priority_queue<int, vector<int>, greater<int>> pq;  
16     pq.push(3);  
17     pq.push(1);  
18     pq.push(4);  
19  
20     while(!pq.empty())
```

```

21     {
22         // **注意! **这里top是获得堆顶, 但是pop()本身返回void, 只是把堆顶丢出去而已
23         cout << pq.top() << endl;
24         pq.pop();
25     }
26
27     return 0;
28 }

```

输出:

```

1 "D:\coding\leetcode-brush questions\leetcode-2023autumn\testing\cmake-build-
  debug\testing.exe"
2 1
3 3
4 4
5
6 进程已结束,退出代码0

```

自定义存放的数据类型和比较函数

347. 前 K 个高频元素

已解答 

中等  相关标签  相关企业  Aa

给你一个整数数组 `nums` 和一个整数 `k`，请你返回其中出现频率前 `k` 高的元素。你可以按 **任意顺序** 返回答案。

示例 1:

输入: `nums = [1,1,1,2,2,3]`, `k = 2`
 输出: `[1,2]`

示例 2:

输入: `nums = [1]`, `k = 1`
 输出: `[1]`

提示:

- `1 <= nums.length <= 105`
- `k` 的取值范围是 `[1, 数组中不相同的元素的个数]`
- 题目数据保证答案唯一，换句话说，数组中前 `k` 个高频元素的集合是唯一的

进阶：你所设计算法的时间复杂度 **必须** 优于 `O(n log n)`，其中 `n` 是数组大小。

知识点：仿函数

在C++中，仿函数（functor）是一个类或者结构体，它重载了函数调用操作符 `()`。这使得它可以被使用在像函数一样的上下文中，例如作为参数传递给算法或者容器的成员函数。通过使用仿函数，你可以将函数对象打包到一个类中，从而使其具有状态并且更加灵活。

再次强调一下：

1. 仿函数不是函数，是类；
2. 仿函数重载了 `()` 运算符，拥有类的行为。

一个简单的示例是实现一个加法仿函数：

```
1 class AddFunctor {
2 public:
3     int operator()(int a, int b) const {
4         return a + b;
5     }
6 };
7
8 int main() {
9     AddFunctor add;
10    int result = add(3, 4); // 调用仿函数的operator()来执行加法运算
11    return 0;
12 }
```

在这个例子中，`AddFunctor` 类重载了 `operator()`，允许它在 `main()` 函数中使用起来就像一个函数一样。仿函数在STL中被广泛应用，用于自定义排序、查找等算法的行为。

算法时间复杂度分析

$O(n)$ 。因为我们维护的堆的大小是 k ，因此，每次插入的时间复杂度是 $\log K$ ，是常数。总共操作了 n 次。

大顶堆还是小顶堆？

```
1 class Solution {
2 public:
3     struct compare_func{
4         bool operator()(const pair<int,int>& p1, const pair<int,int>& p2)
5         {
6             return p1.second >= p2.second;
7         }
8     };
9 }
```

```

7         }
8     };
9     vector<int> topKFrequent(vector<int>& nums, int k) {
10         unordered_map<int,int> mapping;
11         for(auto each:nums)
12         {
13             mapping[each]++;
14         }
15
16         /*
17         然后定义优先级队列。其实这里有个很tricky的地方是，用小顶堆还是大顶堆？
18         如果用大顶堆，我们维持堆里面有k个元素，每次push一个，pop一个，其实留下的最后是整
        个list中
19         出现频率最小的k个元素。
20         */
21         priority_queue<std::pair<int,int>,vector<pair<int,int>>,compare_func>
        q;
22         int counter = 0;
23
24         for(auto each_pair:mapping)
25         {
26             q.push(each_pair);
27             counter++;
28             if(counter > k)
29                 q.pop();
30         }
31
32         vector<int> ans;
33         // for(auto each_pair:q)
34         //     ans.push_back(each_pair.first);
35         while(!q.empty())
36         {
37             ans.push_back(q.top().first);
38             q.pop();
39         }
40
41         return ans;
42     }
43 };
44
45 // 或者:
46 class Solution {
47 public:
48     class compare_func{
49     public:
50         bool operator()(const pair<int,int>& p1, const pair<int,int>& p2)
51         {

```

```

52         return p1.second >= p2.second;
53     }
54 };
55 vector<int> topKFrequent(vector<int>& nums, int k) {
56     unordered_map<int,int> mapping;
57     for(auto each:nums)
58     {
59         mapping[each]++;
60     }
61
62     /*
63     然后定义优先级队列。其实这里有个很tricky的地方是，用小顶堆还是大顶堆？
64     如果用大顶堆，我们维持堆里面有k个元素，每次push一个，pop一个，其实留下的最后是整
    个list中
65     出现频率最小的k个元素。
66     */
67     priority_queue<std::pair<int,int>,vector<pair<int,int>>,compare_func>
    q;
68     int counter = 0;
69
70     for(auto each_pair:mapping)
71     {
72         q.push(each_pair);
73         counter++;
74         if(counter > k)
75             q.pop();
76     }
77
78     vector<int> ans;
79     // for(auto each_pair:q)
80     //     ans.push_back(each_pair.first);
81     while(!q.empty())
82     {
83         ans.push_back(q.top().first);
84         q.pop();
85     }
86
87     return ans;
88 }
89 };

```

5. std::unordered_map

遍历

在C++中，可以使用`auto`关键字和范围循环来遍历`unordered_map`。以下是一种方法：


```
1 std::unordered_map<std::string, int> myMap = {{ "a", 1}, {"b", 2}, {"c", 3}};
2
3 for (const auto &pair : myMap) {
4     std::cout << "Key: " << pair.first << ", Value: " << pair.second <<
    std::endl;
5 }
```

在这个例子中，我们使用`auto`关键字来自动推断`pair`的类型，然后使用范围循环来遍历`unordered_map`中的键值对。

不存在的key?

2024/2/9 14:40:06

unordered_map中，如果访问一个不存在的key，这个键值对会自动创建吗？

2024/2/9 14:40:13

在C++的 `std::unordered_map` 中，如果您尝试访问一个不存在的键（key），则该键会被自动创建并关联到一个默认构造的值。这种行为与 `std::map` 相似。

例如，考虑以下代码：

```
std::unordered_map<std::string, int> myMap;

// 尝试访问一个不存在的键
myMap["newKey"] = 5;

// 现在 "newKey" 已经被自动创建，并且与值 5 相关联
```

在上面的示例中，当 `myMap["newKey"]` 被赋值为 5 时，如果 "newKey" 之前不存在于 `myMap` 中，那么会自动创建一个新的键值对。

需要注意的是，这种行为可能不适用于所有情况。在某些情况下，您可能希望通过检查键是否存在来避免意外创建新的键值对。可以使用 `find` 函数或者 `count` 函数来检查键是否存在，然后再做相应的处理。

```
if (myMap.find("newKey") == myMap.end()) {
    // "newKey" 不存在
    // 执行相关操作...
}
```

因此，在使用 `std::unordered_map` 时，要特别注意如何处理不存在的键的情况，以免出现意外的行为。

基本函数一览

Operations	Invalidated
All read only operations, <code>swap</code> , <code>std::swap</code>	Never
<code>clear</code> , <code>rehash</code> , <code>reserve</code> , <code>operator=</code>	Always
<code>insert</code> , <code>emplace</code> , <code>emplace_hint</code> , <code>operator[]</code>	Only if causes rehash
<code>erase</code>	Only to the element erased

```

1 #include <iostream>
2 #include <string>
3 #include <unordered_map>
4
5 int main()
6 {
7     // Create an unordered_map of three strings (that map to strings)
8     std::unordered_map<std::string, std::string> u =
9     {
10         {"RED", "#FF0000"},
11         {"GREEN", "#00FF00"},
12         {"BLUE", "#0000FF"}
13     };
14
15     // Helper lambda function to print key-value pairs
16     auto print_key_value = [](const auto& key, const auto& value)
17     {
18         std::cout << "Key:[" << key << "]" Value:[" << value << "]\n";
19     };
20
21     std::cout << "Iterate and print key-value pairs of unordered_map, being\n"
22         "explicit with their types:\n";
23     for (const std::pair<const std::string, std::string>& n : u)
24         print_key_value(n.first, n.second);
25
26     std::cout << "\nIterate and print key-value pairs using C++17 structured
27 binding:\n";
28     for (const auto& [key, value] : u)
29         print_key_value(key, value);
30
31     // Add two new entries to the unordered_map
32     u["BLACK"] = "#000000";
33     u["WHITE"] = "#FFFFFF";
34
35     std::cout << "\nOutput values by key:\n"
36         "The HEX of color RED is:[" << u["RED"] << "]\n"
37         "The HEX of color BLACK is:[" << u["BLACK"] << "]\n\n";
38
39     std::cout << "Use operator[] with non-existent key to insert a new key-
40 value pair:\n";
41     print_key_value("new_key", u["new_key"]);
42
43     std::cout << "\nIterate and print key-value pairs, using `auto`;\n"
44         "new_key is now one of the keys in the map:\n";
45     for (const auto& n : u)
46         print_key_value(n.first, n.second);
47 }

```

输出:

```
1 Iterate and print key-value pairs of unordered_map, being
2 explicit with their types:
3 Key:[BLUE] Value:[#0000FF]
4 Key:[GREEN] Value:[#00FF00]
5 Key:[RED] Value:[#FF0000]
6
7 Iterate and print key-value pairs using C++17 structured binding:
8 Key:[BLUE] Value:[#0000FF]
9 Key:[GREEN] Value:[#00FF00]
10 Key:[RED] Value:[#FF0000]
11
12 Output values by key:
13 The HEX of color RED is:[#FF0000]
14 The HEX of color BLACK is:[#000000]
15
16 Use operator[] with non-existent key to insert a new key-value pair:
17 Key:[new_key] Value:[]
18
19 Iterate and print key-value pairs, using `auto`;
20 new_key is now one of the keys in the map:
21 Key:[new_key] Value:[]
22 Key:[WHITE] Value:[#FFFFFF]
23 Key:[BLACK] Value:[#000000]
24 Key:[BLUE] Value:[#0000FF]
25 Key:[GREEN] Value:[#00FF00]
26 Key:[RED] Value:[#FF0000]
```

Insert

```
1 #include <iostream>
2 #include <string>
3 #include <unordered_map>
4
5 int main ()
6 {
7     std::unordered_map<int, std::string> dict = {{1, "one"}, {2, "two"}};
8     dict.insert({3, "three"});
9     dict.insert(std::make_pair(4, "four"));
10    dict.insert({{4, "another four"}, {5, "five"}});
11
12    bool ok = dict.insert({1, "another one"}).second;
```

```

13     std::cout << "inserting 1 => \"another one\" "
14         << (ok ? "succeeded" : "failed") << '\n';
15
16     std::cout << "contents:\n";
17     for (auto& p : dict)
18         std::cout << ' ' << p.first << " => " << p.second << '\n';
19 }

```

输出:

```

1 inserting 1 => "another one" failed
2 contents:
3 5 => five
4 1 => one
5 2 => two
6 3 => three
7 4 => four

```

Erase

```

1 #include <unordered_map>
2 #include <iostream>
3
4 int main()
5 {
6     std::unordered_map<int, std::string> c =
7     {
8         {1, "one"}, {2, "two"}, {3, "three"},
9         {4, "four"}, {5, "five"}, {6, "six"}
10    };
11
12    // erase all odd numbers from c
13    for (auto it = c.begin(); it != c.end(); )
14    {
15        if (it->first % 2 != 0)
16            it = c.erase(it);
17        else
18            ++it;
19    }
20
21    for (auto& p : c)
22        std::cout << p.second << ' ';
23    std::cout << '\n';

```

```
24 }
```

输出:

```
1 two four six
```

```
1 #include <iostream>
2 #include <unordered_map>
3
4 int main() {
5     std::unordered_map<std::string, int> myMap = {"apple", 5}, {"banana", 3},
6     {"cherry", 8};
7
8     // 删除特定key对应的键值对
9     myMap.erase("banana");
10
11    // 打印剩余的键值对
12    for (const auto& pair : myMap) {
13        std::cout << pair.first << ": " << pair.second << std::endl;
14    }
15
16    return 0;
17 }
```

Find

```
1 #include <cstdint>
2 #include <functional>
3 #include <iostream>
4 #include <string>
5 #include <string_view>
6 #include <unordered_map>
7
8 using namespace std::literals;
9
10 struct string_hash
11 {
12     using hash_type = std::hash<std::string_view>;
13     using is_transparent = void;
```

```

14
15     std::size_t operator()(const char* str) const      { return hash_type{}
    (str); }
16     std::size_t operator()(std::string_view str) const { return hash_type{}
    (str); }
17     std::size_t operator()(std::string const& str) const { return hash_type{}
    (str); }
18 };
19
20 int main()
21 {
22     // simple comparison demo
23     std::unordered_map<int, char> example = {{1, 'a'}, {2, 'b'}};
24
25     if (auto search = example.find(2); search != example.end())
26         std::cout << "Found " << search->first << ' ' << search->second <<
    '\n';
27     else
28         std::cout << "Not found\n";
29
30     // C++20 demo: Heterogeneous lookup for unordered containers (transparent
    hashing)
31     std::unordered_map<std::string, size_t, string_hash, std::equal_to<>>
    map{{"one"s, 1}};
32     std::cout << std::boolalpha
33         << (map.find("one") != map.end()) << '\n'
34         << (map.find("one"s) != map.end()) << '\n'
35         << (map.find("one"sv) != map.end()) << '\n';
36 }

```

输出:

```

1 Found 2 b
2 true
3 true
4 true

```

6. std::unordered_set

基本函数一览

Operations	Invalidated
All read only operations, <code>swap</code> , <code>std::swap</code>	Never
<code>clear</code> , <code>rehash</code> , <code>reserve</code> , <code>operator=</code>	Always
<code>insert</code> , <code>emplace</code> , <code>emplace_hint</code>	Only if causes rehash
<code>erase</code>	Only to the element erased

```

1 #include <iostream>
2 #include <unordered_set>
3
4 void print(const auto& set)
5 {
6     for (const auto& elem : set)
7         std::cout << elem << ' ';
8     std::cout << '\n';
9 }
10
11 int main()
12 {
13     std::unordered_set<int> mySet{2, 7, 1, 8, 2, 8}; // creates a set of ints
14     print(mySet);
15
16     mySet.insert(5); // puts an element 5 in the set
17     print(mySet);
18
19     if (auto iter = mySet.find(5); iter != mySet.end())
20         mySet.erase(iter); // removes an element pointed to by iter
21     print(mySet);
22
23     mySet.erase(7); // removes an element 7
24     print(mySet);
25 }

```

输出:

```

1 8 1 7 2
2 5 8 1 7 2
3 8 1 7 2

```

Erase

```

1 #include <unordered_set>
2 #include <iostream>
3
4 int main()
5 {
6     std::unordered_set<int> c = {1, 2, 3, 4, 1, 2, 3, 4};
7
8     auto print = [&c]
9     {
10         std::cout << "c = { ";
11         for (int n : c)
12             std::cout << n << ' ';
13         std::cout << "}\n";
14     };
15     print();
16
17     std::cout << "Erase all odd numbers:\n";
18     for (auto it = c.begin(); it != c.end(); )
19     {
20         if (*it % 2 != 0)
21             it = c.erase(it);
22         else
23             ++it;
24     }
25     print();
26
27     std::cout << "Erase 1, erased count: " << c.erase(1) << '\n';
28     std::cout << "Erase 2, erased count: " << c.erase(2) << '\n';
29     std::cout << "Erase 2, erased count: " << c.erase(2) << '\n';
30     print();
31 }
```

输出:

```

1 c = { 1 2 3 4 }
2 Erase all odd numbers:
3 c = { 2 4 }
4 Erase 1, erased count: 0
5 Erase 2, erased count: 1
```



```
6 Erase 2, erased count: 0
7 c = { 4 }
```

常用的操作流程

1. 插入一个元素之前，先去看set中这个元素是否存在：

```
1 std::unordered_set<std::string, string_hash, std::equal_to<>> set{"one"s,
  "two"s};
2
3     logger::enabled = true;
4     std::cout << "Heterogeneous lookup for unordered containers (transparent
  hashing):\n"
5     << (set.find("one") != set.end())
6     << '\n' << (set.find("one"s) != set.end())
7     << '\n' << (set.find("one"sv) != set.end())
8     << '\n';
```

2. 常用算法和库函数

1. 排序

这里涉及到的一个知识点其实就是lambda表达式。

lambda表达式

1. <https://changkun.de/modern-cpp/zh-cn/03-runtime/#%E5%9F%BA%E7%A1%80>

1686. 石子游戏 VI

提示 ...

中等



106



相关企业

Alice 和 Bob 轮流玩一个游戏，Alice 先手。

一堆石子中总共有 n 个石子，轮到某个玩家时，他可以 **移出** 一个石子并得到这个石子的价值。Alice 和 Bob 对石子价值有 **不一样的评判标准**。双方都知道对方的评判标准。

给你两个长度为 n 的整数数组 `aliceValues` 和 `bobValues`。`aliceValues[i]` 和 `bobValues[i]` 分别表示 Alice 和 Bob 认为第 i 个石子的价值。

所有石子都被取完后，得分较高的人为胜者。如果两个玩家得分相同，那么为平局。两位玩家都会采用 **最优策略** 进行游戏。

请你推断游戏的结果，用如下的方式表示：

- 如果 Alice 赢，返回 `1`。
- 如果 Bob 赢，返回 `-1`。
- 如果游戏平局，返回 `0`。

```
1 class Solution {
2 public:
3     int stoneGameVI(vector<int>& aliceValues, vector<int>& bobValues) {
4         /*
5         这个题目好小学啊，“游戏与对策”
6         贪心？
7         拿对自己利益最大的、拿对别人利益最大的
8         【整个游戏的必胜策略是什么？】
9         1. 可以理解为，拿走一块石头的时候，不仅给自己带来了财富，也剥夺了别人的财富
10        因为如果你不拿这块石头，另一个人是拥有这些财富的，你拿了这块石头，你相当于把
11        这些财富也
12        剥夺了。
13        2. 一开始的时候可以建立两个最大堆。所以可以看看C++里面怎么表示堆这个数据结构。
14        当然也可以不用堆。直接算就行了。
15        */
16        int n = aliceValues.size();
17        vector<vector<int>> value(n,vector<int>(3,0));
18        for(int i = 0; i < n; i++)
19        {
20            value[i] =
21            {aliceValues[i],bobValues[i],aliceValues[i]+bobValues[i]};
22        }
23        // sort
```

```

24     sort(value.begin(),value.end(),[](vector<int> &a, vector<int> &b){
25         return a[2] > b[2];
26     });
27
28     int alice = 0, bob = 0;
29     for(int i = 0; i < n; i++)
30     {
31         if(i % 2 == 0)
32         {
33             alice += value[i][0];
34         }
35         else
36         {
37             bob += value[i][1];
38         }
39     }
40     if(alice > bob)
41         return 1;
42     else if(alice == bob)
43         return 0;
44     return -1;
45 }
46 };

```

注意到lambda表达式：

```

1     sort(value.begin(),value.end(),[](vector<int> &a, vector<int> &b){
2         // 当a[2] > b[2]的时候返回true，所以是降序排序
3         return a[2] > b[2];
4     });

```

奇葩玩意大赏

如何将排序速度加快十倍？

452. 用最少数量的箭引爆气球

中等



928



相关企业

有一些球形气球贴在一堵用 XY 平面表示的墙面上。墙面上的气球记录在整数数组 `points`，其中 `points[i] = [x_start, x_end]` 表示水平直径在 `x_start` 和 `x_end` 之间的气球。你不知道气球的确切 y 坐标。

一支弓箭可以沿着 x 轴从不同点 **完全垂直** 地射出。在坐标 `x` 处射出一支箭，若有一个气球的直径的开始和结束坐标为 `x_start, x_end`，且满足 $x_{start} \leq x \leq x_{end}$ ，则该气球会被 **引爆**。可以射出的弓箭的数量 **没有限制**。弓箭一旦被射出之后，可以无限地前进。

给你一个数组 `points`，返回引爆所有气球所必须射出的 **最小** 弓箭数。

示例 1:

输入: `points = [[10,16],[2,8],[1,6],[7,12]]`

输出: 2

解释: 气球可以用2支箭来爆破:

-在 `x = 6` 处射出箭，击破气球 `[2,8]` 和 `[1,6]`。

-在 `x = 11` 处发射箭，击破气球 `[10,16]` 和 `[7,12]`。

示例 2:

输入: `points = [[1,2],[3,4],[5,6],[7,8]]`

输出: 4

解释: 每个气球需要射出一支箭，总共需要4支箭。

原始答案:

```
1 class Solution {
2 public:
3     int findMinArrowShots(vector<vector<int>>& points) {
4         /*
5         翻译人话:
6         1. 很多个闭区间集合, 比如[1,2]
7         2. 我要求最小的离散整数集, 保证这个整数集和上面所有的区间
8         集合都有交集
9
10        1. 先将所有区间按照第一个元素从小到大排序;
11        有一个很重要的原则, 就是所有区间都要被包括在内。
12        2. 贪心? 然后从前往后找, 直到有一个人的头大于前面所有人的交集为止,
13        然后把这一批都拉出来,
14        算作一组即可。
15        */
16        // sort
17        std::sort(points.begin(), points.end(), [](vector<int> a, vector<int> b){
18            return a[0] <= b[0];
19        });
```

```

20     // for(auto each:points)
21     // {
22     //     for(auto each_:each)
23     //     {
24     //         cout << each_ <<" ";
25     //     }
26     //     cout << endl;
27     // }
28     int ans = 0;
29     // vector<vector<int>> cur;
30     int cur = 0;
31     vector<int> inter(2,0);
32     for(auto &each:points)
33     {
34         // if(cur.empty())
35         if(!cur)
36         {
37             // cur.push_back(each);
38             cur++;
39             inter = each;
40             continue;
41         }
42         // check if can not be merged into the current group
43         if(each[0] > inter[1])
44         {
45             // cout << cur << endl;
46             // cannot be merged
47             ans++;
48             // cur.clear();
49             cur = 1;
50             inter = each;
51         }
52         else
53         {
54             // can merge, update the inter
55             inter[0] = max(inter[0], each[0]);
56             inter[1] = min(inter[1], each[1]);
57             cur++;
58         }
59     }
60     // TODO do not forget the remaining!(might exist)
61     if(cur)
62         ans++;
63     return ans;
64 }
65 };

```

升级版：

```
1 class Solution {
2 public:
3     int findMinArrowShots(vector<vector<int>>& points) {
4         /*
5         翻译人话：
6         1. 很多个闭区间集合，比如[1,2]
7         2. 我要求最小的离散整数集，保证这个整数集和上面所有的区间
8         集合都有交集
9
10        1. 先将所有区间按照第一个元素从小到大排序；
11        有一个很重要的原则，就是所有区间都要被包括在内。
12        2. 贪心？然后从前往后找，直到有一个人的头大于前面所有人的交集为止，
13        然后把这一批都拉出来，
14        算作一组即可。
15        */
16        // sort
17        std::sort(points.begin(),points.end(),[](const vector<int>& a, const
vector<int>&b){
18            return a[0] <= b[0];
19        });
20        // for(auto each:points)
21        // {
22        //     for(auto each_:each)
23        //     {
24        //         cout << each_ <<" ";
25        //     }
26        //     cout << endl;
27        // }
28        int ans = 0;
29        // vector<vector<int>> cur;
30        int cur = 0;
31        vector<int> inter(2,0);
32        for(auto &each:points)
33        {
34            // if(cur.empty())
35            if(!cur)
36            {
37                // cur.push_back(each);
38                cur++;
39                inter = each;
40                continue;
41            }
42            // check if can not be murged into the current group
43            if(each[0] > inter[1])
```

```
44         {
45             // cout << cur << endl;
46             // cannot be merged
47             ans++;
48             // cur.clear();
49             cur = 1;
50             inter = each;
51         }
52         else
53         {
54             // can merge, update the inter
55             inter[0] = max(inter[0], each[0]);
56             inter[1] = min(inter[1], each[1]);
57             cur++;
58         }
59     }
60     // TODO do not forget the remaining!(might exist)
61     if(cur)
62         ans++;
63     return ans;
64 }
65 };
```

可以看到，就是一个排序，内存和运行速度都有巨大的差距：

通过

官方题解

写题解

时间

250 ms

击败 99.57% 使用 C++ 的用户

内存

91.18 MB

击败 18.88% 使用 C++ 的用户

更多挑战

253. 会议室 II

435. 无重叠区间

所有状态	所有语言	执行用时	消耗内存	备注
通过 11 分钟前	C++	250 ms	91.2 MB	
通过 11 分钟前	C++	261 ms	91.2 MB	
通过 12 分钟前	C++	2332 ms	546 MB	
通过 13 分钟前	C++	2381 ms	546 MB	
通过 13 分钟前	C++	2350 ms	558.1 MB	
通过 13 分钟前	C++	2430 ms	558.2 MB	

3. 专题：字符串的处理

同时注意string/char...



在C++中，双引号 `"` 用于包裹字符串常量，而单引号 `'` 用于包裹字符常量。双引号内的内容被视为字符串，可以包含多个字符，而单引号内的内容只能包含一个字符。例如：


```
1 char ch = 'a'; // 单个字符常量
2 string str = "Hello"; // 字符串常量
```

另外，使用双引号包裹的字符串常量会在末尾自动添加一个空字符 `\0`，表示字符串的结束，而单引号包裹的字符常量不会自动添加。

题目：

14. 最长公共前缀

简单  3.1K  

 相关企业

编写一个函数来查找字符串数组中的最长公共前缀。

如果不存在公共前缀，返回空字符串 `""`。

示例 1:

输入: strs = ["flower","flow","flight"]
输出: "fl"

示例 2:

输入: strs = ["dog","racecar","car"]
输出: ""
解释: 输入不存在公共前缀。

提示:

- `1 <= strs.length <= 200`
- `0 <= strs[i].length <= 200`
- `strs[i]` 仅由小写英文字母组成

```
1 class Solution {
2 public:
3     string longestCommonPrefix(vector<string>& strs) {
4         int idx = 0;
5         int max_idx = 1000;
6         string prefix = "";
7         for(auto each:strs)
8         {
9             if(each.size() < max_idx)
10                 max_idx = each.size();
11         }
12         while(true)
13         {
14             if(idx >= max_idx)
15                 break;
16             // 这里single的类型是char
17             auto single = strs[0][idx];
18             for(auto each:strs)
```

```

19         {
20             if(each[idx] != single)
21             {
22                 // break
23                 return prefix;
24             }
25         }
26         prefix += single;
27         idx++;
28     }
29     return prefix;
30 }
31 };

```

错误写法：

题目描述 评论 (4.5K) 题解 (7.4K) 提交记录

示例 1:

输入: strs = ["flower","flow","flight"]
输出: "fl"

示例 2:

输入: strs = ["dog","racecar","car"]
输出: ""
解释: 输入不存在公共前缀。

提示:

- 1 ≤ strs.length ≤ 200
- 0 ≤ strs[i].length ≤ 200

编译出错

Line 16: Char 29: error: no matching conversion for C-style cast from 'value_type' (aka 'char') to 'string' (aka 'basic_string<char>')

```

16 |         string single = (string)strs[0][idx];
    |                        ~~~~~^~~~~~
/usr/bin/../lib/gcc/x86_64-linux-gnu/11/../../../../include/c++/11/bits/basic_string.h:533:7: note: candidate constructor template not viable: no known conversion from 'value_type' (aka 'char') to 'const char*' for 1st argument; take the address of the argument with &
524 |         basic_string(const _CharT* __s, const _Alloc& __a = _Alloc())
    |         ~~~~~^~~~~~
/usr/bin/../lib/gcc/x86_64-linux-gnu/11/../../../../include/c++/11/bits/basic_string.h:448:7: note: candidate constructor template not viable: no known conversion from 'value_type' (aka 'char') to 'const std::allocator'

```

```

1  class Solution {
2  public:
3      string longestCommonPrefix(vector<string>& strs) {
4          int idx = 0;
5          int max_idx = 1000;
6          string prefix = "";
7          for(auto each:strs)
8          {
9              if(each.size() < max_idx)
10                 max_idx = each.size();
11          }
12          while(true)
13          {
14              if(idx >= max_idx)
15                  break;
16              string single = (string)strs[0][idx];
17              for(auto each:strs)
18              {
19                  if(each[idx] != single)
20                  {
21                      // break
22                      return prefix;
23                  }
24              }
25              prefix += single;
26              idx++;
27          }
28          return prefix;
29      }
30  };

```

1. 字符串和整数转换

在C++中，你可以使用标准库中的函数来实现字符串和整数之间的相互转换。以下是一些常用的方法：

从整数到字符串的转换：

```

1 #include <iostream>
2 #include <string>
3 #include <sstream>
4

```

```

5 int main() {
6     int number = 123;
7     std::string strNumber = std::to_string(number);
8     std::cout << "The integer as a string: " << strNumber << std::endl;
9     return 0;
10 }

```

从字符串到整数的转换：

```

1 #include <iostream>
2 #include <string>
3 #include <sstream>
4
5 int main() {
6     std::string strNumber = "123";
7     int number = std::stoi(strNumber);
8     std::cout << "The string as an integer: " << number << std::endl;
9     return 0;
10 }

```

另外，你还可以使用stringstream进行字符串和整数之间的转换：

```

1 #include <iostream>
2 #include <string>
3 #include <sstream>
4
5 int main() {
6     int number = 123;
7     std::stringstream ss;
8     ss << number;
9     std::string strNumber = ss.str();
10    std::cout << "The integer as a string: " << strNumber << std::endl;
11
12    std::string strNumber2 = "456";
13    int number2;
14    ss.clear(); // 清空状态
15    ss.str(strNumber2); // 设置新的字符串
16    ss >> number2; // 从字符串中读取整数
17    std::cout << "The string as an integer: " << number2 << std::endl;
18    return 0;

```

使用这些方法，你可以在C++中方便地进行字符串和整数之间的转换。

2. 子串

```
s.substr(pos, n)
```

- 截取s中从pos开始（包括0）的n个字符的子串，并返回

```
s.substr(pos)
```

- 截取s中从pos开始（包括0）到末尾的所有字符的子串，并返回

4. Modern c++

5. 屑题大赏

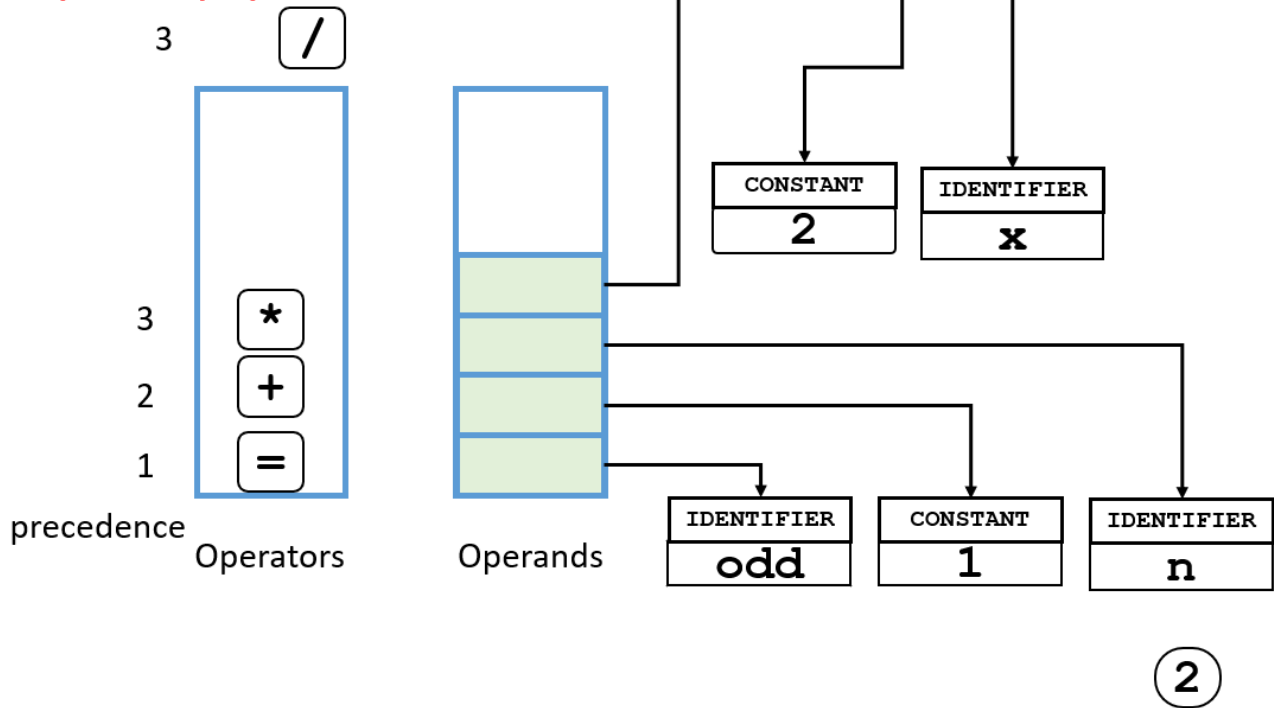
224 基本计算器

核心思路：

1. 两个栈，一个操作符，一个操作数
2. 为了处理-不是减号而是负号的问题，在tokenize这一步，多加一个0进去即可，这个时候一元运算符又变成了二元运算符
3. 还有一个核心就是和basic解释器一样，一个运算符入栈前，先要把优先级不低于它的运算符先出栈算完，然后它自己才能入栈。

precedence	operators
3	* /
2	+ -
1	=

Pop until top's precedence < 3



$$\text{Odd} = 1 + n * (2 - x) / 2$$

224. 基本计算器



困难 1K

相关企业

给你一个字符串表达式 `s`，请你实现一个基本计算器来计算并返回它的值。

注意:不允许使用任何将字符串作为数学表达式计算的内置函数，比如 `eval()`。

示例 1:

输入: `s = "1 + 1"`

输出: 2

示例 2:

输入: `s = " 2-1 + 2 "`

输出: 3

示例 3:

输入: `s = "(1+(4+5+2)-3)+(6+8)"`

输出: 23

```
1 class Solution {
2 public:
3     deque<string> tokenizer(string & s)
4     {
5         deque<string> ans;
6         string cur;
7         int n = s.size();
8
9         for(auto each:s)
10        {
11            if(each == ' ')
12                continue;
13
14            if(each >= '0' && each <= '9')
15            {
16                cur += each;
17            }
18            else
19            {
20                if(cur.size() > 0)
21                {
22                    ans.push_back(cur);
```

```

23         cur = "";
24     }
25     if(each == '-' && (ans.size() == 0 || ans.back() == "("))
26     {
27         // each = '!';
28         ans.push_back("0");
29     }
30     cur += each;
31     ans.push_back(cur);
32     cur = "";
33 }
34 }
35 if(cur.size() > 0)
36     ans.push_back(cur);
37 return ans;
38 }
39 int babyCalculator(string& operand1, string& operand2, string& operator_)
40 {
41     int integer_operand1 = std::stoi(operand1);
42     int integer_operand2 = std::stoi(operand2);
43     if(operator_ == "+")
44         return integer_operand1 + integer_operand2;
45     else if(operator_ == "-")
46         return integer_operand1 - integer_operand2;
47     else if(operator_ == "*")
48         return integer_operand1 * integer_operand2;
49
50     return integer_operand1 / integer_operand2 + 1;
51 }
52 int calculate(string s) {
53     /*
54     当年basic解释器的感觉又回来了.....
55     1. 两个栈，一个操作符，一个操作数
56     */
57     deque<string> operand_stack; // 操作数
58     deque<string> operator_stack; // 操作符
59     string cal1, cal2;
60     deque<string> tokens = tokenizer(s);
61
62     unordered_map<string, int> hierarchy;
63     hierarchy["("] = -1;
64     hierarchy["+"] = 0;
65     hierarchy["-"] = 0;
66     hierarchy["*"] = 1;
67     hierarchy["/"] = 1;
68
69     for(auto each:tokens)

```

```

70     {
71         if(each == "(")
72         {
73             // just push into stack
74             operator_stack.push_back(each);
75         }
76         else if(each == ")")
77         {
78             // pop until we meet the "("
79             while(operator_stack.back() != "(")
80             {
81                 string cur_operator = operator_stack.back();
82                 operator_stack.pop_back();
83                 cal2 = operand_stack.back();
84                 operand_stack.pop_back();
85                 cal1 = operand_stack.back();
86                 operand_stack.pop_back();
87
88                 operand_stack.push_back(std::to_string(babyCalculator(cal1,cal2,cur_operator)))
89                 ;
90             }
91             operator_stack.pop_back(); // "("
92         }
93         else if(each == "*" || each == "/" || each == "+" || each == "-")
94         {
95             while(operator_stack.size() &&
96                 hierarchy[operator_stack.back()] >= hierarchy[each])
97             {
98                 string cur_operator = operator_stack.back();
99                 operator_stack.pop_back();
100                cal2 = operand_stack.back();
101                operand_stack.pop_back();
102                cal1 = operand_stack.back();
103                operand_stack.pop_back();
104
105                operand_stack.push_back(std::to_string(babyCalculator(cal1,cal2,cur_operator)))
106                ;
107            }
108            operator_stack.push_back(each);
109        }
110        else
111        {
112            operand_stack.push_back(each);
113        }
114    }

```



```

111      // operate until there is only a single element left in the operand
      stack
112      while(operand_stack.size() > 1)
113      {
114          string cur_operator = operator_stack.back();
115          operator_stack.pop_back();
116          cal2 = operand_stack.back();
117          operand_stack.pop_back();
118          cal1 = operand_stack.back();
119          operand_stack.pop_back();
120
121          operand_stack.push_back(std::to_string(babyCalculator(cal1,cal2,cur_operator)))
122          ;
123      }
124      return std::stoi(operand_stack.back());
125  };

```

222 完全二叉树的节点个数（二分法）

这里主要写一下二分查找应该怎么做？

注意：

1. 下面第一条注释掉的写法会导致unbounded loop（当max仅比min大1的时候，出不去的）
2. 第二种写法是保证了左端点一定是闭的

```

1
2 // int man_in_the_middle = (min_nodes + max_nodes) / 2;
3 int man_in_the_middle = (max_nodes - min_nodes + 1) / 2 + min_nodes;

```

222. 完全二叉树的节点个数

简单



1.1K

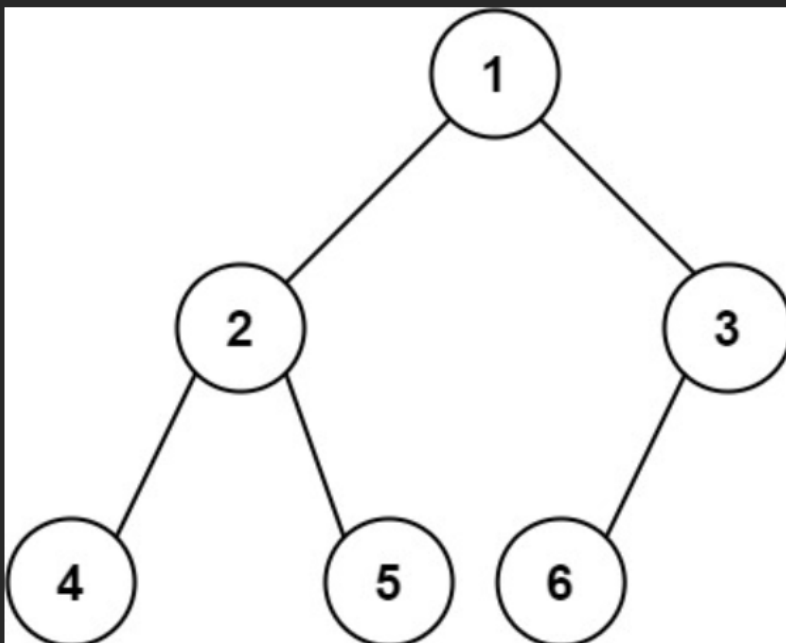


相关企业

给你一棵 **完全二叉树** 的根节点 `root`，求出该树的节点个数。

完全二叉树 的定义如下：在完全二叉树中，除了最底层节点可能没填满外，其余每层节点数都达到最大值，并且最下面一层的节点都集中在该层最左边的若干位置。若最底层为第 h 层，则该层包含 $1 \sim 2^h$ 个节点。

示例 1:



输入: `root = [1,2,3,4,5,6]`

输出: 6

```
1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8  *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9  *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
    right(right) {}
10  * };
11 */
```

```

12 class Solution {
13 public:
14     int countNodes(TreeNode* root) {
15         int level = 1, counter = 0;
16         if(!root)
17             return counter;
18         if(!root->left)
19             return 1;
20         TreeNode* shadow = root;
21         while(shadow->left)
22         {
23             shadow = shadow->left;
24             level++;
25         }
26
27         // then we can get the possible max and min value of nodes in the tree
28         int min_nodes = (1 << (level - 1));
29         int max_nodes = (1 << level) - 1;
30         // int man_in_the_middle = (min_nodes + max_nodes) / 2;
31         int man_in_the_middle = (max_nodes - min_nodes + 1) / 2 + min_nodes;
32         int mask = (1 << (level - 1));
33
34         // for each man_in_the_middle, check routine
35         while(min_nodes < max_nodes)
36         {
37             // check if man_in_the_middle exist
38             // create routine: bit operation 最高位不看, 然后后面所有bit往左挪动;
39             // 1代表向右, 0代表向左
40             shadow = root;
41             int save_man = man_in_the_middle;
42             int cur_choice;
43             cout << min_nodes << "," << max_nodes << "," << man_in_the_middle
44             << endl;
45             man_in_the_middle = man_in_the_middle << 1;
46             for(int i = 1; i < level; i++)
47             {
48                 cur_choice = man_in_the_middle & mask;
49                 // != 0 向右, == 0 向左
50                 if(cur_choice)
51                 {
52                     shadow = shadow->right;
53                     cout << "right" << endl;
54                 }
55                 else
56                 {
57                     shadow = shadow->left;
58                     cout << "left" << endl;
59                 }
60             }
61             min_nodes = shadow->left ? min_nodes : max_nodes;
62             max_nodes = shadow->right ? max_nodes : min_nodes;
63             man_in_the_middle = (min_nodes + max_nodes) / 2;
64         }
65         return counter;
66     }
67 };

```

```

58         }
59         man_in_the_middle = man_in_the_middle << 1;
60     }
61
62     // check
63     if (shadow)
64         min_nodes = save_man;
65     else
66         max_nodes = save_man - 1;
67
68     // update
69     // man_in_the_middle = (min_nodes + max_nodes) / 2;
70     man_in_the_middle = (max_nodes - min_nodes + 1) / 2 + min_nodes;
71
72     cout << ">>" << min_nodes << "," << max_nodes << "," <<
    man_in_the_middle << endl;
73     }
74
75     // return answer
76     return min_nodes;
77 }
78 };

```

6. ACM模式

leetcode刷题一般是核心代码模式。ACM模式需要自己处理输入输出。

知识点：读入一行字符串

7. 平均绩点

时间限制：1.000S 空间限制：32MB

题目描述

每门课的成绩分为A、B、C、D、F五个等级，为了计算平均绩点，规定A、B、C、D、F分别代表4分、3分、2分、1分、0分。

输入描述

有多组测试样例。每组输入数据占一行，由一个或多个大写字母组成，字母之间由空格分隔。

输出描述

每组输出结果占一行。如果输入的大写字母都在集合 {A,B,C,D,F} 中，则输出对应的平均绩点，结果保留两位小数。否则，输出“Unknown”。

输入示例

```
A B C D F
B F F C C A
D C E F
```

输出示例

```
2.00
1.83
Unknown
```

```
1 #include <iostream>
2 #include <unordered_map>
3
4 using namespace std;
5
6 int main()
7 {
8     unordered_map<char, int> mapping;
9     mapping['A'] = 4;
10    mapping['B'] = 3;
11    mapping['C'] = 2;
12    mapping['D'] = 1;
13    mapping['F'] = 0;
14    string line;
15
16    while(getline(cin, line))
17    {
18        int score = 0;
19        int counter = 0;
20        int n = line.size();
21        for(int i = 0; i < n; i++)
```

```
22     {
23         if(line[i] == ' ')
24             continue;
25         else if(mapping.find(line[i]) == mapping.end())
26         {
27             score = -1;
28             break;
29         }
30         score += mapping[line[i]];
31         counter++;
32     }
33     if(score == -1)
34         cout << "Unknown" << endl;
35     else
36         printf("%.2f\n",score/(double)counter);
37 }
38 return 0;
39 }
```

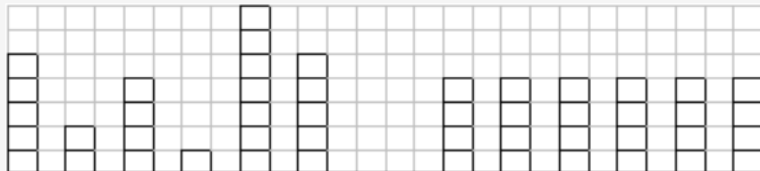
题目：摆平积木

8. 摆平积木

时间限制：1.000S 空间限制：32MB

题目描述

小明很喜欢玩积木。一天，他把许多积木块组成了好多高度不同的堆，每一堆都是一个摞一个的形式。然而此时，他又想把这些积木堆变成高度相同的。但是他很懒，他想移动最少的积木块来实现这一目标，你能帮助他吗？



输入描述

输入包含多组测试样例。每组测试样例包含一个正整数 n ，表示小明已经堆好的积木堆的个数。
接着下一行是 n 个正整数，表示每一个积木堆的高度 h ，每块积木高度为1。其中 $1 \leq n \leq 50, 1 \leq h \leq 100$ 。
测试数据保证积木总数能被积木堆数整除。
当 $n=0$ 时，输入结束。

输出描述

对于每一组数据，输出将积木堆变成相同高度需要移动的最少积木块的数量。
在每组输出结果的下面都输出一个空行。

输入示例

```
6
5 2 4 1 7 5
0
```

输出示例

```
5
```

```
1 # include <iostream>
2 # include <vector>
3
4 using namespace std;
5
6 int main()
7 {
8     // 注意，测试包含多组输入样例
9     int n;
10    while(cin >> n)
11    {
12        if(n == 0)
13            continue;
```

```
14     vector<int> v;
15     int counter = 0;
16     int a;
17     for(int i = 0; i < n; i++)
18     {
19         cin >> a;
20         v.emplace_back(a);
21         counter+=a;
22     }
23     counter /= n;
24     int ans = 0;
25     for(auto each:v)
26     {
27         ans += max(each-counter, counter-each);
28     }
29     cout << ans/2 << endl << endl;
30 }
31
32 return 0;
33 }
```

题目：奇怪的信

9. 奇怪的信

时间限制：1.000S 空间限制：32MB

题目描述

有一天, 小明收到一张奇怪的信, 信上要小明计算出给定数各个位上数字为偶数的和。
例如: 5548, 结果为12, 等于 $4 + 8$ 。
小明很苦恼, 想请你帮忙解决这个问题。

输入描述

输入数据有多组。每组占一行, 只有一个整数, 保证数字在32位整型范围内。

输出描述

对于每组输入数据, 输出一行, 每组数据下方有一个空行。

输入示例

```
415326
3262
```

输出示例

```
12

10
```

```
1 # include <iostream>
2 # include <string>
3
4 using namespace std;
5
6 int main()
7 {
8     string line;
9     while(getline(cin, line))
10    {
11        int n = line.size();
12        int counter = 0;
13        for(auto each:line)
14        {
15            if((each-'0') % 2 == 0)
16            {
17                counter += each - '0';
18            }
19        }
```

```
20         cout << counter << endl << endl;
21     }
22     return 0;
23 }
```

题目：运营商活动

10. 运营商活动

时间限制：1.000S 空间限制：32MB

题目描述

小明每天的话费是1元，运营商做活动，手机每充值K元就可以获赠1元话费，一开始小明充值M元，问最多可以用多少天？注意赠送的话费也可以参与到奖励规则中

输入描述

输入包括多个测试实例。每个测试实例包括2个整数M, K ($2 \leq k \leq M \leq 1000$)。M=0, K=0代表输入结束。

输出描述

对于每个测试实例输出一个整数，表示M元可以用的天数。

输入示例

```
2 2
4 3
13 3
0 0
```

输出示例

```
3
5
19
```

提示信息

注意第三组数据「13 3」结果为什么是19呢， $13/3=4$ ，获得4元奖励。 $13\%3=1$ ，还剩下1元， $4+1=5$ ，5元继续参加奖励规则。 $5/3=1$ ，获得1元奖励。 $5\%3=2$ ，剩下2元， $1+2=3$ ，3元继续参与奖励规则。 $3/3=1$ ，获得1元奖励。 $3\%3=0$ ，剩下0元， $1+0=1$ 。1元不能参与剩下奖励。所以一共可以使用天数是 $13+4+1+1=19$

```
1 # include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
```

```

7     int a,b;
8     while(cin >> a >> b)
9     {
10         int cur_val = a;
11         if(a == 0 && b == 0)
12             break;
13         while(cur_val >= b)
14         {
15             a += (cur_val) / b;
16             cur_val = cur_val % b + cur_val / b;
17         }
18         cout << a << endl;
19     }
20     return 0;
21 }

```

用stringstream读入一行用空白隔开的字符串

2129. 将标题首字母大写

已解答 

简单

 相关标签

 相关企业

 提示

Ax

给你一个字符串 `title`，它由单个空格连接一个或多个单词组成，每个单词都只包含英文字母。请你按以下规则将每个单词的首字母 **大写**：

- 如果单词的长度为 `1` 或者 `2`，所有字母变成小写。
- 否则，将单词首字母大写，剩余字母变成小写。

请你返回 **大写后** 的 `title`。

示例 1:

输入: `title = "capiTalIze tHe titLe"`

输出: `"Capitalize The Title"`

解释:

由于所有单词的长度都至少为 `3`，将每个单词首字母大写，剩余字母变为小写。

示例 2:

输入: `title = "First leTTeR of EACH Word"`

输出: `"First Letter of Each Word"`

解释:

单词 `"of"` 长度为 `2`，所以它保持完全小写。

其他单词长度都至少为 `3`，所以其他单词首字母大写，剩余字母小写。

```
1 class Solution {
2 public:
3     string capitalizeTitle(string title) {
4         // 注意这里的字符流，每次就可以读入一个单词了
5         stringstream ss(title);
6         string word;
7         string ans;
8         while(ss >> word)
9         {
10             int n = word.size();
11             if(n == 1 || n == 2)
12             {
13                 for(auto &c:word)
14                 {
15                     if(c >= 'A' && c <= 'Z')
16                         c = c - 'A' + 'a';
17                 }
18             }
19             else
20             {
21                 if(word[0] >= 'a' && word[0] <= 'z')
22                     word[0] = word[0] - 'a' + 'A';
23                 for(int i = 1; i < n; i++)
24                 {
25
26                     if(word[i] <= 'Z' && word[i] >= 'A')
27                         word[i] = word[i] - 'A' + 'a';
28                 }
29             }
30
31             ans += word + ' ';
32         }
33         ans = ans.substr(0,ans.size()-1);
34         return ans;
35     }
36 };
```