

stl

zhou888

雅礼中学

November 1, 2018



Preface

stl在noip中的应用很多，有时一些题用stl会使代码简洁很多。



algorithm

max, min, swap, max_element, min_element, reverse

algorithm

max, min, swap, max_element, min_element, reverse

*sort*将区间按升序排序,*partial_sort*将区间内较小的N个元素排序,*stable_sort*稳定排序,*unique*合并排序后相同的元素

algorithm

max, min, swap, max_element, min_element, reverse

*sort*将区间按升序排序, *partial_sort*将区间内较小的N个元素排序, *stable_sort*稳定排序, *unique*合并排序后相同的元素

*nth_element(begin, nth, end)*将第nth小的元素放在第nth个位置上, 然后将比它小的放在它之前, 比它大的放在它之后, $O(n)$

algorithm

max, min, swap, max_element, min_element, reverse

*sort*将区间按升序排序,*partial_sort*将区间内较小的N个元素排序,*stable_sort*稳定排序,*unique*合并排序后相同的元素

*nth_element(begin, nth, end)*将第nth小的元素放在第nth个位置上,然后将比它小的放在它之前,比它大的放在它之后, $O(n)$

排序后二分查找:*lower_bound*第一个大于等于某个数的位置,*upper_bound*第一个严格大于某个数的位置,这里的等于定义为 x 不小于 y 且 y 不小于 x

algorithm

max, min, swap, max_element, min_element, reverse

*sort*将区间按升序排序, *partial_sort*将区间内较小的 N 个元素排序, *stable_sort*稳定排序, *unique*合并排序后相同的元素

*nth_element(begin, nth, end)*将第 n th小的元素放在第 n th个位置上, 然后将比它小的放在它之前, 比它大的放在它之后, $O(n)$

排序后二分查找: *lower_bound*第一个大于等于某个数的位置, *upper_bound*第一个严格大于某个数的位置, 这里的等于定义为 x 不小于 y 且 y 不小于 x

*next/prev_permutation*按照字典序排列的下一个/上一个排列



iterator

迭代器，一个指针。

iterator

迭代器，一个指针。

```
set < int >:: iterator it;
```



iterator

迭代器，一个指针。

```
set < int >:: iterator it;
```

可以用 $++it$ 或 $--it$ 改变位置

iterator

迭代器，一个指针。

```
set < int >:: iterator it;
```

可以用 $++it$ 或 $--it$ 改变位置

用 $it \rightarrow$ 或 $*it$ 访问指针代表元素的数据

iterator

迭代器，一个指针。

```
set < int >:: iterator it;
```

可以用 $++it$ 或 $--it$ 改变位置

用 $it \rightarrow$ 或 $*it$ 访问指针代表元素的数据

几个常用的指针 $begin, end, rbegin, rend$

iterator

迭代器，一个指针。

```
set < int >:: iterator it;
```

可以用 $++it$ 或 $--it$ 改变位置

用 $it \rightarrow$ 或 $*it$ 访问指针代表元素的数据

几个常用的指针 $begin, end, rbegin, rend$

以下为了方便迭代器都用 it 代替



basic

基本操作：

`.clear()`清空某个容器..`.empty()`测试某个容器是否为空

basic

基本操作：

`.clear()`清空某个容器..`.empty()`测试某个容器是否为空

`.resize(n)`将某个容器长度定义为 n 的



basic

基本操作：

`.clear()`清空某个容器..`.empty()`测试某个容器是否为空

`.resize(n)`将某个容器长度定义为 n 的

`.size()`返回容器大小

basic

基本操作：

`.clear()`清空某个容器..`.empty()`测试某个容器是否为空

`.resize(n)`将某个容器长度定义为 n 的

`.size()`返回容器大小

`.insert(it, x)`在 it 处加一个 x 元素(stack,queue,bitset没有)

basic

基本操作：

`.clear()`清空某个容器..`.empty()`测试某个容器是否为空

`.resize(n)`将某个容器长度定义为 n 的

`.size()`返回容器大小

`.insert(it, x)`在 it 处加一个 x 元素(stack,queue,bitset没有)

`.erase(it)`删除 it 处的元素,..`.erase(itl, itr)`,删除 $[itl, ltr)$ 区间中的元素

string

顾名思义，就是一个字符串,可以自带+,=

string

顾名思义，就是一个字符串,可以自带+,=

定义:*string* s;

string

顾名思义，就是一个字符串,可以自带+,=

定义:*string* s;

s+=可以加一个字符串，也可以加一个字符

string

顾名思义，就是一个字符串,可以自带+,=

定义:*string* s;

s+=可以加一个字符串，也可以加一个字符

调用第*i*个元素:s[i]

string

顾名思义，就是一个字符串,可以自带+,=

定义:*string* s;

s+=可以加一个字符串，也可以加一个字符

调用第*i*个元素:s[i]

一般stl容器都默认从0下标开始

function

s.length() 返回长度



function

s.length() 返回长度

s.c_str() 从 *string* 转为 *char[]*

function

s.length() 返回长度

s.c_str() 从 *string* 转为 *char[]*

s.substr(l, r) 返回 $s[l \rightarrow r]$ 代表的子串

function

s.length() 返回长度

s.c_str() 从 *string* 转为 *char[]*

s.substr(l, r) 返回 $s[l \rightarrow r]$ 代表的子串

getline(cin, str) 读入一行字符串

vector

0下标的动态数组



vector

0下标的动态数组

*ve.push/pop_back()*在最后加/删一个元素

vector

0下标的动态数组

*ve.push/pop_back()*在最后加/删一个元素

*ve.insert()*特别快(1e5次vector的insert()和7e5次set的insert()用时相当)

vector

0下标的动态数组

*ve.push/pop_back()*在最后加/删一个元素

*ve.insert()*特别快(1e5次vector的insert()和7e5次set的insert()用时相当)

*vector*申请的内存为连续的

bitset

bitset 可以看作一个 bool 数组，也可以看作一个二进制数，自带除 32/64 常数，支持左/右移，与，或，异或等二进制操作



bitset

bitset 可以看作一个 bool 数组，也可以看作一个二进制数，自带除 32/64 常数，支持左/右移，与，或，异或等二进制操作

bit.count(), *bit* 中 1 的个数 *bit.any()*, *bit* 中是否有 1

bitset

bitset 可以看作一个 bool 数组，也可以看作一个二进制数，自带除 32/64 常数，支持左/右移，与，或，异或等二进制操作

bit.count(), *bit* 中 1 的个数 *bit.any()*, *bit* 中是否有 1

bit.reset(), *bit* 清零

bitset

bitset 可以看作一个 bool 数组，也可以看作一个二进制数，自带除 32/64 常数，支持左/右移，与，或，异或等二进制操作

bit.count(), *bit* 中 1 的个数 *bit.any()*, *bit* 中是否有 1

bit.reset(), *bit* 清零

bit.set(x) 将第 x 位设为 1, *bit.flip(x)* 将第 x 位异或 1

bitset

bitset 可以看作一个 bool 数组，也可以看作一个二进制数，自带除 32/64 常数，支持左/右移，与，或，异或等二进制操作

bit.count(), *bit* 中 1 的个数 *bit.any()*, *bit* 中是否有 1

bit.reset(), *bit* 清零

bit.set(x) 将第 *x* 位设为 1, *bit.flip(x)* 将第 *x* 位异或 1

bit._Find_first(), *bit._Find_next(x)* 找到第一个 1 的位置，找到 *x* 之后下一个 1 的位置

stack

一个栈



stack

一个栈

$S.top()$ 栈顶

stack

一个栈

$S.top()$ 栈顶

$S.push()$ 加入一个元素在栈顶

stack

一个栈

$S.top()$ 栈顶

$S.push()$ 加入一个元素在栈顶

$S.pop()$ 弹出栈顶元素

stack

一个栈

$S.top()$ 栈顶

$S.push()$ 加入一个元素在栈顶

$S.pop()$ 弹出栈顶元素

(其实还不如手写...)

queue

动态开内存的队列，bfs时最好用这个(防止爆数组)



queue

动态开内存的队列，bfs时最好用这个(防止爆数组)

$q.front()$ 返回队首的数

queue

动态开内存的队列，bfs时最好用这个(防止爆数组)

$q.front()$ 返回队首的数

其他的操作都和 $stack$ 差不多

deque

双端队列，可以当一个可以队首队尾加/删的数组，可以直接调用下标

deque

双端队列，可以当一个可以队首队尾加/删的数组，可以直接调用下标

deq.push_front()/push_back(),deq.pop_front()/deq.pop_back()

deque

双端队列，可以当一个可以队首队尾加/删的数组，可以直接调用下标

deq.push_front()/push_back(),deq.pop_front()/deq.pop_back()

可以直接*insert*和*erase*

deque

双端队列，可以当一个可以队首队尾加/删的数组，可以直接调用下标

`deq.push_front()/push_back(), deq.pop_front()/deq.pop_back()`

可以直接`insert`和`erase`

`deq.front()/back()` 队首和队尾

priority_queue

优先队列，即堆

priority_queue

优先队列，即堆

默认为大根堆,小根堆

为 `priority_queue < int, vector < int >, greater < int > >`

priority_queue

优先队列，即堆

默认为大根堆,小根堆

为 $\text{priority_queue} < \text{int}, \text{vector} < \text{int} >, \text{greater} < \text{int} > >$

$q.\text{top}()$ 堆顶, $q.\text{pop}()$ 弹出



priority_queue

优先队列，即堆

默认为大根堆,小根堆

为 $\text{priority_queue} < \text{int}, \text{vector} < \text{int} >, \text{greater} < \text{int} > >$

$q.\text{top}()$ 堆顶, $q.\text{pop}()$ 弹出

各种题里应用很多

set

set为一个集合，内置一个红黑树。

set

set为一个集合，内置一个红黑树。

如果要用可重集要用 $multiset$

set

set为一个集合，内置一个红黑树。

如果要用可重集要用 $multiset$

操作和之前差不多

set

set为一个集合，内置一个红黑树。

如果要用可重集要用`multiset`

操作和之前差不多

(multi)set/map如果要二分查找的话最好用`st.lower_bound()`

set

set为一个集合，内置一个红黑树。

如果要用可重集要用 $multiset$

操作和之前差不多

(multi)set/map如果要二分查找的话最好用 $st.lower_bound()$

(multi)set.erase一个指针为 $O(1)$,删除一个数为 $O(\log)$

set

set为一个集合，内置一个红黑树。

如果要用可重集要用`multiset`

操作和之前差不多

(multi)set/map如果要二分查找的话最好用`st.lower_bound()`

(multi)set.erase一个指针为 $O(1)$,删除一个数为 $O(\log)$

multiset里erase一个数会删掉所有这个数

map

一个平衡树，可以当可以二分查找的 $hash$ 数组用

map

一个平衡树，可以当可以二分查找的 $hash$ 数组用

定义: $map < ty1, ty2 > \dots$

map

一个平衡树，可以当可以二分查找的 $hash$ 数组用

定义: $map < ty1, ty2 > \dots$

可以作为一个下标为任何东西的数组

map

一个平衡树，可以当可以二分查找的 $hash$ 数组用

定义: $map < ty1, ty2 > \dots$

可以作为一个下标为任何东西的数组

但是由于本质上是一个平衡树，会带一个 \log

unordered_map

需要使用 `include < tr1/unordered_map >` 头文件（如果c++11则不需要）

unordered_map

需要使用`include < tr1/unordered_map >`头文件（如果c++11则不需要）

直接当一个哈希表就可以了，只有`int, long long`少数几种类型可以用

unordered_map

需要使用`include < tr1/unordered_map >`头文件（如果c++11则不需要）

直接当一个哈希表就可以了，只有`int, long long`少数几种类型可以用

`unordered_set`和`unordered_map`差不多，一个没有排序查找的`set`

unordered_map

需要使用`include < tr1/unordered_map >`头文件（如果c++11则不需要）

直接当一个哈希表就可以了，只有`int, long long`少数几种类型可以用

`unordered_set`和`unordered_map`差不多，一个没有排序查找的`set`

需要使用`include < tr1/unordered_set >`头文件（如果c++11则不需要）

unordered_map

需要使用`include < tr1/unordered_map >`头文件（如果c++11则不需要）

直接当一个哈希表就可以了，只有`int, long long`少数几种类型可以用

`unordered_set`和`unordered_map`差不多，一个没有排序查找的`set`

需要使用`include < tr1/unordered_set >`头文件（如果c++11则不需要）

复杂度 $O(1)$ (`unordered_set`比`unordered_map`快很多)

bit

`__builtin_popcount(x)` 的 x 的二进制中有多少个 1

bit

`__builtin_popcount(x)` x 的二进制中有多少个1

`__builtin_ffs(x)` x 的二进制末尾最后一个1的位置，从1开始

bit

`__builtin_popcount(x)` x 的二进制中有多少个 1

`__builtin_ffs(x)` x 的二进制末尾最后一个 1 的位置，从 1 开始

`__builtin_ctz(x)` x 的二进制末尾后面 0 的个数

bit

`__builtin_popcount(x)` x 的二进制中有多少个1

`__builtin_ffs(x)` x 的二进制末尾最后一个1的位置，从1开始

`__builtin_ctz(x)` x 的二进制末尾后面0的个数

`__builtin_clz(x)` x 的二进制前导0的个数

bit

`__builtin_popcount(x)` x 的二进制中有多少个1

`__builtin_ffs(x)` x 的二进制末尾最后一个1的位置，从1开始

`__builtin_ctz(x)` x 的二进制末尾后面0的个数

`__builtin_clz(x)` x 的二进制前导0的个数

pbds

见pbds.pdf

CF 443 Div1 C

Berland要举行 n 次锦标赛,第一次只有一个人,之后每一次会新加入一个人。锦标赛中有 k 种运动项目,每个人在这 k 种项目上都有一个能力值,每次会选择任意两个还未被淘汰的人进行某个项目的比赛,能力值高的人胜出,输的人被淘汰,直至只剩下一个人成为冠军。

给出每个人每个项目的能力值,保证它们两两不同,求每次锦标赛有多少人可能成为冠军。

$$n \leq 5 \times 10^4, k \leq 10$$

Solution by dy

只要选手 a 在某个项目上比选手 b 强, a 就可以淘汰 b ,我们可以连一条 a 到 b 的边。

对整个图求强连通分量,缩点后一定会形成一个竞赛图,拓扑序最靠前的分量中的所有点都可能成为冠军。

每加入一个点时,我们可能需要合并拓扑序在一段区间内强连通分量。用 set 按拓扑序维护每个强连通分量,对每个分量记录它的大小,以及在每个项目上的最大和最小能力值,就可以直接在 set 上二分找到需要合并的区间。

最多只会合并 $n - 1$ 次, $O(nk \log n)$

代码实现

```
bool operator < (const point &rhs) const{  
    REP(i, 1, m) if(Max[i] > rhs.Min[i]) return 0;  
    return 1;  
}
```

这样通过`lower_bound`可以找出第一个有项目上能力大于当前人的位置，通过`upper_bound`可以找出所有项目均大于当前人的位置。

给定一个序列,支持以下操作:

修改一个元素

求第 l 个到第 r 个数中最早的一个 w 的位置

给定一个序列,支持以下操作:

修改一个元素

求第 l 个到第 r 个数中最早的一个 w 的位置

$map < int, set < int > >$

Thanks