2019/05/10 位运算及标准模版库 - AcWing

> 位运算及标准模版库

位运算

- 1、位运算符

  > 这些位运算只能用于整型操作数，即只能用于带符号或无符号的 char、short、int、与 long 类型。

```
☆按位与（&）
☆按位或（|）
☆按位异或（^）
☆取反（~）
☆左移（<<）n<<1=2*n,1<<n=2^n
☆右移（>>）n>>1=n/2
```

```
优先级：
加减    移位    比较大小    位与    异或    位或
+,-    <<,>>    >,<,==,!=    &    ^    |
```

- 2、位运算的应用举例

```
☆高效代替布尔型数组
☆表示集合、搜索算法中的状态判重（hash）
☆动态规划算法中的状态压缩
```

vector

> 使用 vector，需添加 `#include<vector>`，同时要有 using namespace std。

- 1、定义

```
vector<typename> name;
```

```
vector<int> a;
vector<double> b;
vector<node> student;
vector<int> c[100];
vector<vector<int> > a;
```

- 2、访问

```
☆下标访问：对于vector<int> v,使用v[idx]访问它的第idx个元素
0<=idx<=v.size()-1,v.size()表示vector中元素个数

☆迭代器访问：定义：
vector<typename>::iterator it;
可以通过*it来访问vector里的元素。迭代器也可以进行自加、自减操作。
v.begin()为取v的首元素地址，v.end()为取尾元素地址的下一个地址。
```

- 3、常用函数

```
☆push_back()
push_back(X)在vector后添加一个元素x。

☆size()
一维：vector中元素个数
二维：vector中第二维元素个数
用resize(n)重设数组大小

☆pop_back()
删除vector的尾元素

☆clear(n)
清空vector中的所有元素

☆insert()
insert(it,x)用来向vector任意迭代器it处插入一个元素x

☆erase()
erase(it):删除it处元素
earse(left right):删除[left,right)内所有元素
```

stack

> stack 翻译为栈，是一个” 后进先出 “的容器。使用时需添加 `include<stack>`，同时要有 using namespace std。

- 1、定义：

```
stack<typename> name;
```

- 2、常用函数

```
☆push()
push(x)用来将x压栈
```

```
stop()
获得栈顶元素

spop()
弹出栈顶元素

sempty()
检测stack是否为空，空返回true，否则返回false

ssize()
返回stack内元素个数
```

- 只能通过函数 `top()` 和 `pop()` 来访问栈顶元素。

queue 和 priority_queue

queue

> queue 翻译为队列，是一个“ 先进先出 ”的容器。使用时需添加 `include<queue>`，同时要有 using namespace std。

- 1、定义

```
queue<typename> name;
```

- 2、常用函数

```
spush()
push(x)用来将x入队

sfront()
获得队首元素

sback()
获得队尾元素

spop()
让队首元素出队

sempty()
检测queue是否为空

ssize()
返回queue内元素的个数
```

- queue 只能通过函数 `front()` 来访问队首元素，或通过函数 `back()` 来访问队尾元素。一般应用在宽搜中。在使用 `front()` 和 `pop()` 前，必须用 `empty()` 判断队列是否为空，否则可能因为队空而出现错误。

priority_queue

> priority_queue 翻译为优先队列，一般用来解决一些贪心问题，其底层是用 “堆” 来实现的。使用时需添加 `include<queue>`，同时要有 using namespace std。

- 1、定义

```
priority_queue<typename> name;
```

- 和 queue 不一样的是，priority_queue 没有 `front()` 和 `back()`，而只能通过 `top()` 或 `pop()` 访问队首元素（也称堆顶元素），也就是优先级最高的元素。

- 2、常用函数

```
spush()
push(x)是将x加入优先队列

stop()
获得队首元素（堆顶元素）

spop()
让队首元素（堆顶元素）出队
```

- 使用 top() 必须用 empty() 判断优先队列是否为空，否则有可能出错。

- 2、元素优先级的设置
  优先队列对它们的优先级设置一般是数字越大的优先级越高（对于 char，则是字典序最大的）。

```
以下两种优先队列的定义是等价的：
priority_queue<int> q;
priority_queue<int,vector<int>,less<int>> q;
```

map 和 pair

map

> map 翻译为映射。使用时需添加 `include<map>`，同时要有 using namespace std。

```
map的用途有以下情形：
s需要建立字符（串）与整数之间的映射，使用map可以减少的代码量。
s判断大整数（比如几千位）或者其他类型数据是否存在，可以把map当布尔型数组使用（哈希表）。
s字符串与字符串之间的映射。
```

定义一个 map 的方法如下：

```
map<typename1,typename2> name;
```

```
map<int,int> a;
map<string,int> a;
map<set<int>,string>mp;
```

- 1、访问
  ①通过下标访问：

```
先定义"map<char,int> mp",然后就可以通过mp['c']的方式来访问它对应的元素。
```

- 2、常用函数

```
①find()
find(key)是返回键为key的映射的迭代器。

②size()
size()获得map中映射的对数。

③erase()
删除单个元素：
erase(it):it为要删除的元素的迭代器
erase(key):key为要删除的映射的键
删除一个区间的所有元素：
erase(left,right):删除[left,right)内所有元素

④clear()
清空map
```

pair

> pair 是 “二元结构体” 的替代品。使用时需添加 `include<utility>`，同时要有 using namespace std。

- 定义

```
pair<typename1,typename2> name;
```

相当于：

```
struct pair{
    typename1 first;
    typename2 second;
}
```

```
pair<string,int> p;make_pair("haha",5);
pair<string,int> p("haha",5);
```

- pair 可以直接做比较运算，比较规则是先以 first 的大小作为标准，只有当 first 相等时才去判断 second 的大小。
- 所以，pair 可以作为 map 的键值对来插入（排序）。

set

> set 翻译为集合，是一个内部自动有序且不含重复元素的容器。使用时需添加 `include<set>`，同时要有 using namespace std。

- 1、定义

```
set<typename> name;
```

```
set<int> st;
set<int> st[100];
```

set 只能通过迭代器访问。

```
定义迭代器：
set<typename>::iterator it;
然后使用*it来访问set中的元素。
set不支持*(it+i),it<st.end()的访问方式。
```

- 2、常用函数

```
①insert()
insert(x)用来将x插入到set中，并自动递增排序和去重。

②size()
获得set中的元素个数

③find()
find(value)是返回set中对应值为value的迭代器

④clear()
清空set中的所有元素

⑤erase()
删除单个元素：
erase(it):it为要删除元素的迭代器
earse(value):value为要删除的元素的值
删除一个区间的所有元素：
erase(left,right):删除[left,right)内所有元素
```

string

> 使用时需添加 `include<string>`，同时要有 using namespace std。
> * 定义

```
string name;
```

- 1、访问

```
①像普通字符数组一样操作
②通过迭代器
定义迭代器：
string::iterator it;
然后就可以通过*it来访问string里的每一个元素。
```

- 2、运算

```
加法：把两个字符串直接拼接起来。
关系：按照字典序比较两个string类型的大小。
```

- 3、常用函数

```
①length()
返回string的长度（字符个数）

②size()
与length()一样

③clear()
用来清空string中的所有元素

④substr()
substr(pos,len)返回从pos号位置开始、长度为len的子串

⑤insert()
insert(pos,string):在pos号位置插入字符串string
insert(it,it2,it3):it为原字符串的欲插入位置；it2和it3为待插入字符串的首尾迭代器（左毕右开区间）

⑥erase()
删除单个元素：
erase(it):it为要删除元素的迭代器
删除一个区间内的所有元素：
erase(left,right):删除[left,right)中的所有元素
erase(pos,length):pos为需要删除的字符串起始位置；length为要删除的字符个数

⑦find()
str.find(str2):当str2是str的子串时，返回其在str中第一次出现的位置；否则，返回string::npos
string::npos是一个常数，其本身的值等于-1
str.find(str2,pos):从str的pos号位开始匹配str2，返回值同上

⑧replace()
str.replace(pos,len,str2):把str从pos号位开始、长度为len的子串替换为str2
str.replace(it1,it2,str2):把str的迭代器it1~it2范围内（左闭右开区间）的子串替换为str2
```

algorithm

> algoriyhm 翻译为算法提供了大量基于迭代器的非成员模版函数。要使用这些函数，需要添加 `#include<algorithm>`，必须要有 using namespace std。

- 1、 `max(),min(),abs(),swap()`
- 2、 `reverse()`：`reverse(it,it2)` 可以将数组指针在 it~it2（左闭右开区间）之间的元素，或容器的迭代器在 it~it2 范围内的所有元素进行反转。
- 3、 `next_permutation()`：求出一个序列在全排列中的下一个序列。
- 4、 `fill()`： 可以把数组或容器的某一段区间赋值为某个相同的值

```
fill(a,a+5,123);
```

- 5、 `sort()`：实现排序的函数

```
sort(首元素地址，尾元素地址的下一地址，比较函数);
```

如果要实现递减排序，或者对结构体（本身没有大小关系）等进行排序，就需要用到比较函数，一般写成 cmp() 函数。
递减排序：

```
bool cmp(int a,int b){
    return a>b;
}
```

结构体按照 x 从大到小排列：

```
bool cmp(node a,node b){
    return a.x>b.x;
}
```

- 在 STL 标准容器中，只有 vector、string、deque 是可以用 sort() 的。