**C++ STL:**

**pair**

Pair<T> a (T) 直接声明一个pair<T> 的a;

Pair<T> a ; a = make\_pair(first,second);

**set**

s.begin() // 返回指向第一个元素的迭代器

s.clear() // 清除所有元素

s.count() // 返回某个值元素的个数

s.empty() // 如果集合为空，返回true(真）

s.end() // 返回指向最后一个元素之后的迭代器，不是最后一个元素

s.equal\_range() // 返回集合中与给定值相等的上下限的两个迭代器

s.erase() // 删除集合中的元素

s.find() // 返回一个指向被查找到元素的迭代器

s.get\_allocator() // 返回集合的分配器

s.insert() // 在集合中插入元素

s.lower\_bound() // 返回指向大于（或等于）某值的第一个元素的迭代器

s.key\_comp() // 返回一个用于元素间值比较的函数

s.max\_size() // 返回集合能容纳的元素的最大限值

s.rbegin() // 返回指向集合中最后一个元素的反向迭代器

s.rend() // 返回指向集合中第一个元素的反向迭代器

s.size() // 集合中元素的数目

s.swap() // 交换两个集合变量

s.upper\_bound() // 返回大于某个值元素的迭代器

s.value\_comp() // 返回一个用于比较元素间的值的函数

**multiset**

在<set>头文件中，还定义了另一个非常实用的模版类multiset（多重集合）。多重集合与集合的区别在于集合中不能存在相同元素，而多重集合中可以存在。

## vector

vector<int> s;

// 定义一个空的vector对象，存储的是int类型的元素

vector<int> s(n);

// 定义一个含有n个int元素的vector对象

vector<int> s(first, last);

// 定义一个vector对象，并从由迭代器first和last定义的序列[first, last)中复制初值

vector的基本操作：

s[i] // 直接以下标方式访问容器中的元素

s.front() // 返回首元素

s.back() // 返回尾元素

s.push\_back(x) // 向表尾插入元素x

s.size() // 返回表长

s.empty() // 表为空时，返回真，否则返回假

s.pop\_back() // 删除表尾元素

s.begin() // 返回指向首元素的随机存取迭代器

s.end() // 返回指向尾元素的下一个位置的随机存取迭代器

s.insert(it, val) // 向迭代器it指向的元素前插入新元素val

s.insert(it, n, val)// 向迭代器it指向的元素前插入n个新元素val

s.insert(it, first, last)

// 将由迭代器first和last所指定的序列[first, last)插入到迭代器it指向的元素前面

s.erase(it) // 删除由迭代器it所指向的元素

s.erase(first, last)// 删除由迭代器first和last所指定的序列[first, last)

s.reserve(n) // 预分配缓冲空间，使存储空间至少可容纳n个元素

s.resize(n) // 改变序列长度，超出的元素将会全部被删除，如果序列需要扩展（原空间小于n），元素默认值将填满扩展出的空间

s.resize(n, val) // 改变序列长度，超出的元素将会全部被删除，如果序列需要扩展（原空间小于n），val将填满扩展出的空间

s.clear() // 删除容器中的所有元素

s.swap(v) // 将s与另一个vector对象进行交换

s.assign(first, last) // 将序列替换成由迭代器first和last所指定的序列[first, last)，[first, last)不能是原序列中的一部分

## stack

stack<int> s;

stack<string> ss;

stack的基本操作有：

s.push(x); // 入栈

s.pop(); // 出栈

s.top(); // 访问栈顶

s.empty(); // 当栈空时，返回true

s.size(); // 访问栈中元素个数

## queue

queue<int> q;

queue<double> qq;

queue的基本操作：

q.push(x); // 入队列

q.pop(); // 出队列

q.front(); // 访问队首元素

q.back(); // 访问队尾元素

q.empty(); // 判断队列是否为空

q.size(); // 访问队列中的元素个数

## map

map<string, int> m;

map的基本操作：

/\* 向map中插入元素 \*/

m[key] = value; // [key]操作是map很有特色的操作,如果在map中存在键值为key的元素对, 则返回该元素对的值域部分,否则将会创建一个键值为key的元素对,值域为默认值。所以可以用该操作向map中插入元素对或修改已经存在的元素对的值域部分。

m.insert(make\_pair(key, value)); // 也可以直接调用insert方法插入元素对,insert操作会返回一个pair,当map中没有与key相匹配的键值时,其first是指向插入元素对的迭代器,其second为true;若map中已经存在与key相等的键值时,其first是指向该元素对的迭代器,second为false。

/\* 查找元素 \*/

int i = m[key]; // 要注意的是,当与该键值相匹配的元素对不存在时,会创建键值为key（当另一个元素是整形时，m[key]=0）的元素对。

map<string, int>::iterator it = m.find(key); // 如果map中存在与key相匹配的键值时,find操作将返回指向该元素对的迭代器,否则,返回的迭代器等于map的end()(参见vector中提到的begin()和end()操作)。

/\* 删除元素 \*/

m.erase(key); // 删除与指定key键值相匹配的元素对,并返回被删除的元素的个数。

m.erase(it); // 删除由迭代器it所指定的元素对,并返回指向下一个元素对的迭代器。

/\* 其他操作 \*/

m.size(); // 返回元素个数

m.empty(); // 判断是否为空

m.clear(); // 清空所有元素

## bitset

const int MAXN = 32;

bitset<MAXN> bt; // bt 包括 MAXN 位，下标 0 ~ MAXN - 1，默认初始化为 0

bitset<MAXN> bt1(0xf); // 0xf 表示十六进制数 f，对应二进制 1111，将 bt1 低 4 位初始化为 1

bitset<MAXN> bt2(012); // 012 表示八进制数 12，对应二进制 1010，即将 bt2 低 4 位初始化为 1010

bitset<MAXN> bt3("1010"); // 将 bt3 低 4 位初始化为 1010

bitset<MAXN> bt4(s, pos, n);// 将 01 字符串 s 的 pos 位开始的 n 位初始化 bt4

bitsetbitset 基本操作：

bt.any() // bt 中是否存在置为 1 的二进制位？

bt.none() // bt 中不存在置为 1 的二进制位吗？

bt.count() // bt 中置为 1 的二进制位的个数

bt.size() // bt 中二进制位的个数

bt[pos] // 访问 bt 中在 pos 处的二进制位

bt.test(pos) // bt 中在 pos 处的二进制位是否为 1

bt.set() // 把 bt 中所有二进制位都置为 1

bt.set(pos) // 把 bt 中在 pos 处的二进制位置为 1

bt.reset() // 把 bt 中所有二进制位都置为 0

bt.reset(pos) // 把 bt 中在pos处的二进制位置为0

bt.flip() // 把 bt 中所有二进制位逐位取反

bt.flip(pos) // 把 bt 中在 pos 处的二进制位取反

bt[pos].flip() // 同上

bt.to\_ulong() // 用 bt 中同样的二进制位返回一个 unsigned long 值

os << bt // 把 bt 中的位集输出到 os 流

## [algorithm](http://baike.baidu.com/view/87622.htm" \t "_blank)

A

adjacent\_find

B

binary\_search (int  binary\_search (T arr[],  int size,  T target) ;)

C

copy / copy\_backward / count / count\_if

E

equal / equal\_range

F

fill / fill\_n / find / find\_end / find\_first\_of / find\_if / for\_each

G

generate / generate\_n

I

includes / inplace\_merge / iter\_swap

L

lexicographical\_compare / lower\_bound

M

make\_heap / max / max\_element / merge / min / min\_element / mismatch

N

next\_permutation / nth\_element

P

partial\_sort / partial\_sort\_copy / partition / pop\_heap / prev\_permutation / push\_heap

R

random\_shuffle / remove / remove\_copy / remove\_copy\_if / remove\_if / replace / replace\_copy / replace\_copy\_if / replace\_if / reverse / reverse\_copy / rotate / rotate\_copy

S

search / search\_n / set\_difference / set\_intersection / set\_symmetric\_difference / set\_union / sort / sort\_heap / stable\_partition / stable\_sort / swap / swap\_ranges

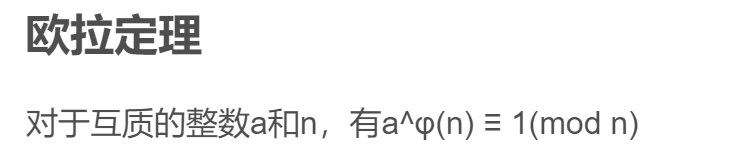
T

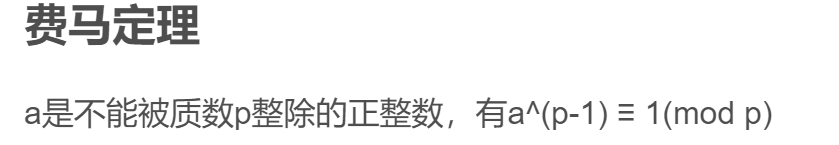
transform

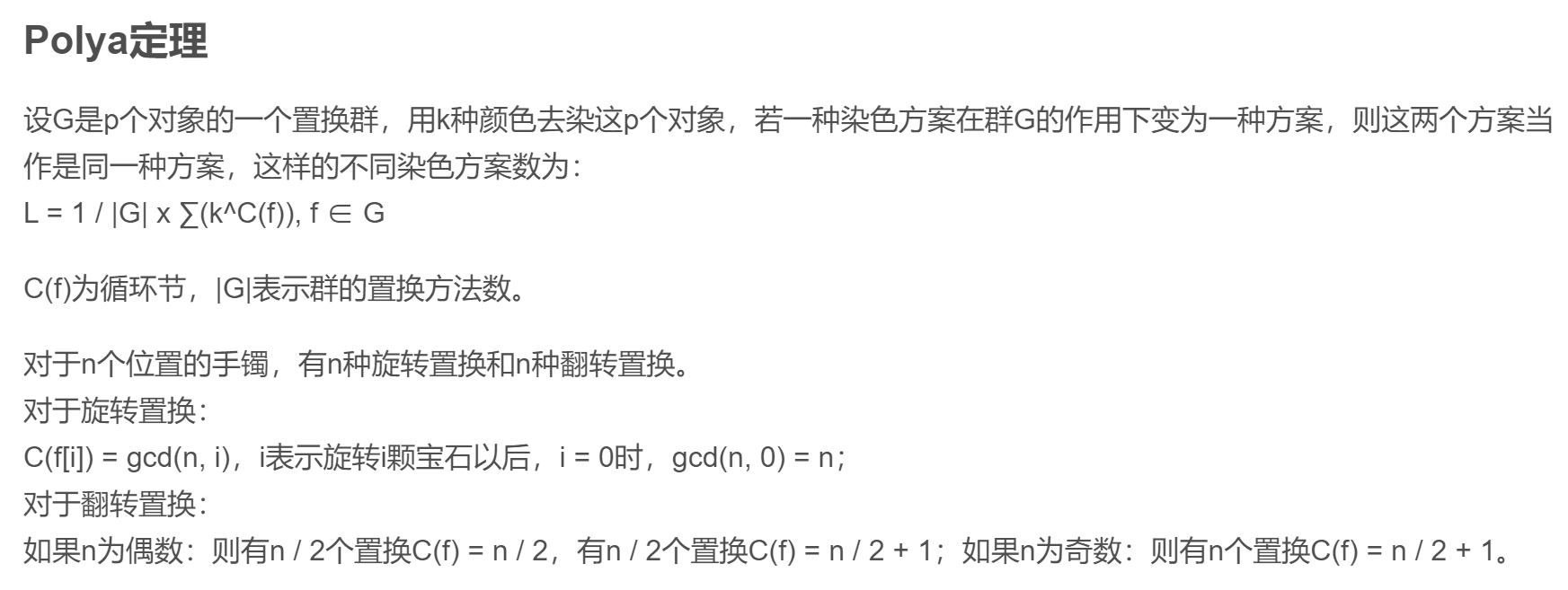
U

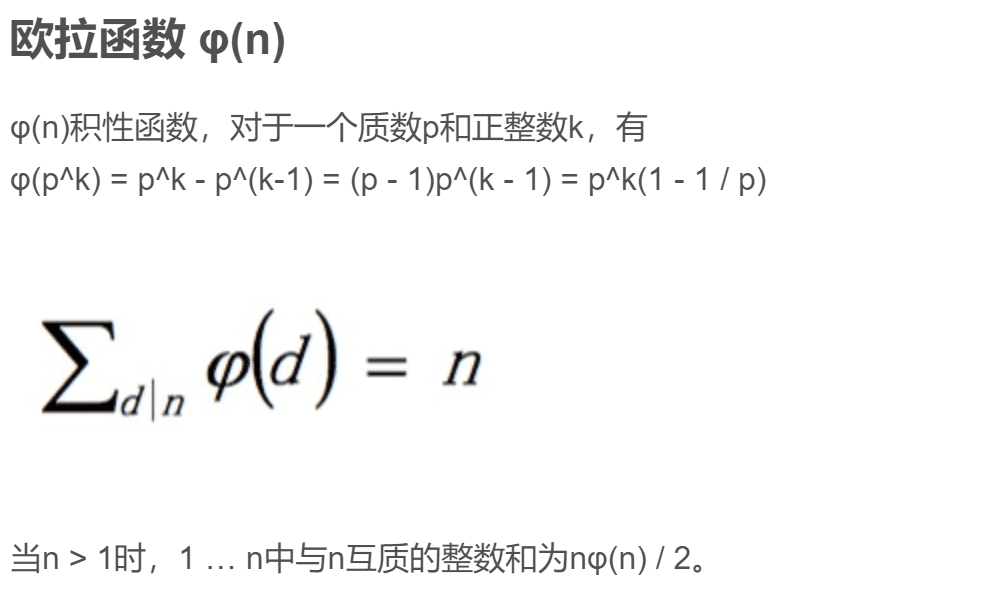
unique / unique\_copy / upper\_bound

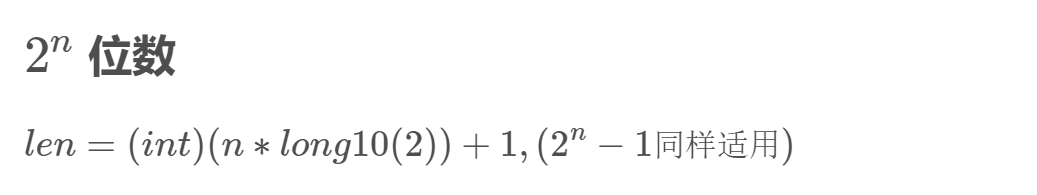
**定理**

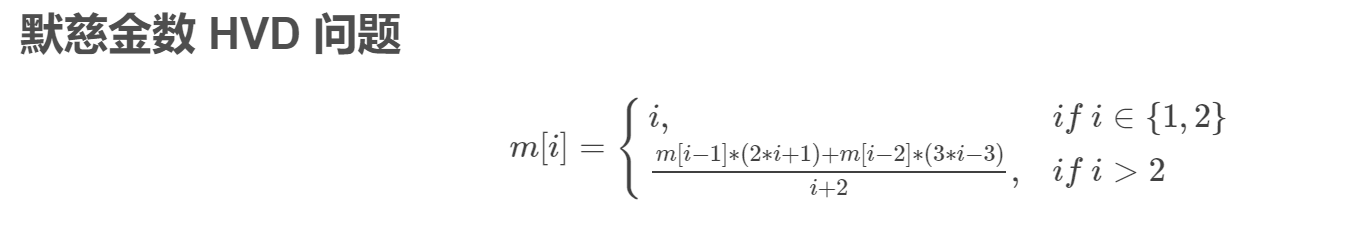




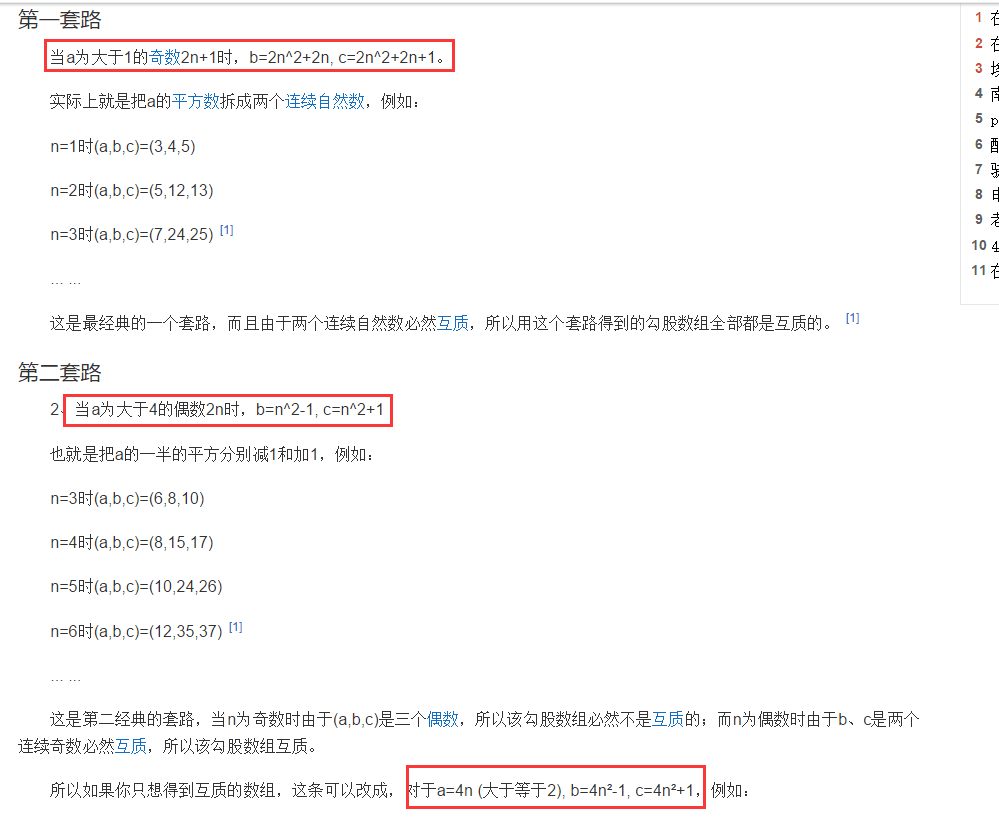








**费马大定理**

****