模板

基础算法

高精度计算

```
bool cmp(vector<int> &A, vector<int> &B) { // 比较函数
  if (A.size() != B.size()) return A.size() > B.size();
  for (int i = A.size() - 1; i >= 0; --i) {
    if (A[i] < B[i]) return false;
  }
  return true;
}
vector<int> add(vector<int> &A, vector<int> &B) { // 高精度加法
  if (A.size() < B.size()) return add(B, A);
  vector<int> C;
  int t = 0;
  for (int i = 0; i < A.size(); ++i) {
    t += A[i];
    if (i < B.size()) t += B[i];
    C.push_back(t % 10);
    t /= 10;
  }
  if (t) C.push_back(1);
  return C;
}
vector<int> sub(vector<int> &A, vector<int> &B) { // 高精度减法
  if(!cmp(A, B)) return sub(B, A);
  vector<int> C;
  int t = 0;
  for (int i = 0; i < A.size(); ++i) {
    t = A[i] - t;
    if (i < B.size()) t -= B[i];
    C.push_back((t + 10) % 10);
    if (t < 0) t = 1;
    else t = 0;
  }
  while (C.size() > 1 && !C.back() == 0) C.pop_back();
  return C;
}
vector<int> mul(vector<int> &A, int b) { // 高精度乘低精度
  vector<int> C;
  int t = 0;
  for (int i = 0; i < A.size() || t; ++i) {
    if (i < A.size()) t = A[i] * b;
    C.push_back(t % 10);
    t /= 10;
  }
  return C;
}
```

```
vector<int> div(vector<int> &A, int b) { // 高精度除以低精度
  vector<int> C;
  int t = 0;
  for (int i= A.size(); i >= 0; --i) {
    t =  t * 10 + A[i] / b;
    C.push_back(t);
    t %= b;
  }
  reverse(C.begin(), C.end());
  while (C.size() > 1 && !C.back()) C.pop_back();
  return C;
}
```

二分浮点数

```
bool check(double x) { /* ... */ }  // 检查x是否满足某种性质

double bsearch_3(double l, double r) {
  const double eps = 1e-6;  // eps 表示精度，取决于题目对精度的要求
  while (r - l > eps) {
    double mid = (l + r) / 2;
    if (check(mid))
      r = mid;
    else
      l = mid;
  }
  return l;
}
```

数据结构

单链表

```
int head, // head存储链表头，e[]存储节点的值，ne[]存储节点的next指针，idx表示当前用到了哪个节点
e[N], ne[N], idx;
void init() { head = -1, idx = 0; } // 初始化
void insert(int a) { e[idx] = a, ne[idx] = head, head = idx++; } // 在链表头插入一个数a
void remove() { head = ne[head]; } // 将头结点删除，需要保证头结点存在
```

双链表

```
int e[N], // e[]表示节点的值，l[]表示节点的左指针，r[]表示节点的右指针，idx表示当前用到了哪个节点
```

```
l[N], r[N], idx;
void init() { // 初始化 0是左端点，1是右端点
  r[0] = 1, l[1] = 0;
  idx = 2;
}
void insert(int a, int x) { // 在节点a的右边插入一个数x
  e[idx] = x;
  l[idx] = a, r[idx] = r[a];
  l[r[a]] = idx, r[a] = idx++;
}
void remove(int a) { l[r[a]] = l[a], r[l[a]] = r[a]; } // 删除节点a
```

滑动数组

```
void huadongmin(int a[], int n, int k) {
  deque<int> q;
  for (int i = 1; i <= n; i++) {
    while (!q.empty() && a[i] < q.back()) q.pop_back();
    q.push_back(a[i]);
    if (i >= k + 1)
      if (q.front() == a[i - k]) q.pop_front();
    if (i >= k) cout << q.front() << " ";
  }
  q.clear();
}
void huadongmax(int a[], int n, int k) {
  deque<int> q;
  for (int i = 1; i <= n; i++) {
    while (!q.empty() && a[i] > q.back()) q.pop_back();
    q.push_back(a[i]);
    if (i >= k + 1)
      if (q.front() == a[i - k]) q.pop_front();
    if (i >= k) cout << q.front() << " ";
  }
  q.clear();
}
```

滚动前缀和 某一节点的加减

```
int lowbit(int x) { return x & -x; }

void init() {
  for (int i = 1; i <= n; i++) {
    c[i] = pre[i] - pre[i - lowbit(i)];
  }
}
void add(int x, int y) {
  for (; x <= N; x += lowbit(x)) {
```

```
    c[x] += y;
  }
}
int ask(int x) {  // 查询a序列的 [1,x] 中所有数的和
  int ans = 0;
  for (; x; x -= lowbit(x)) {
    ans += c[x];
  }
  return ans;
}
```

数学知识

线性筛法求素数

```
int primes[N], cnt;  // primes[]存储所有素数
bool st[N];          // st[x]存储x是否被筛掉
void get_primes(int n) {
  for (int i = 2; i <= n; i++) {
    if (!st[i]) primes[cnt++] = i;
    for (int j = 0; primes[j] <= n / i; j++) {
      st[primes[j] * i] = true;
      if (i % primes[j] == 0) break;
    }
  }
}
```

试除法求所有约数

```
vector<int> get_divisors(int x) {
  vector<int> res;
  for (int i = 1; i <= x / i; i++)
    if (x % i == 0) {
      res.push_back(i);
      if (i != x / i) res.push_back(x / i);
    }
  sort(res.begin(), res.end());
  return res;
}
```

欧几里得算法

```
int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }
```

快速幂

```
int qmi(int m, int k, int p) { // 求 m ^ k mod p，时间复杂度 O(logk)。
  int res = 1 % p, t = m;
  while (k) {
    if (k & 1) res = res * t %
    t = t * t % p;
    k >>= 1;
  }
  return res;
}
```

点积αβ=|α||β|cosθ

```
struct Point {
  double x, y;
  Point(double x = 0, double y = 0) : x(x), y(y) {}  // 构造函数
} Vector[100];
double Dot(Vector A, Vector B) { return A.x * B.x + A.y * B.y; }
/*
α×β
若β在α的逆时针方向，则为正值、顺时针则为负值、两向量共线则为0
*/
double Cross(Vector A, Vector B) { return A.x * B.y - B.x * A.y; }
// 取模（求长度）(Length)
double Length(Vector A) { return sqrt(Dot(A, A)); }
// 计算两向量夹角(Angle)
double Angle(Vector A, Vector B) {
  return acos(Dot(A, B) / Length(A) / Length(B));
}
// 计算两向量构成的平行四边形有向面积(Area2)
double Area2(Point A, Point B, Point C) { return Cross(B - A, C - A); }
// 计算向量逆时针旋转后的向量(Rotate)
Vector Rotate(Vector A, double rad) {
  return Vector(A.x * cos(rad) - A.y * sin(rad),
                A.x * sin(rad) + A.y * cos(rad));
}
```

海伦公式

```
p = (a + b + c) / 2;
s = sqrt(p * (p - a) * (p - b) * (p - c));
```

C++ STL

vector

```
size()  返回元素个数
empty()  返回是否为空
clear()  清空
front()/back()
push_back()/pop_back()
begin()/end()

FAQ: 支持比较运算，按字典序
```

pair<int, int>

```
first，第一个元素
second，第二个元素
支持比较运算，以first为第一关键字，以second为第二关键字（字典序）
```

string

```
size()/length()  返回字符串长度
empty()
clear()
substr(起始下标，(子串长度))  返回子串
c_str()  返回字符串所在字符数组的起始地址
push_back() 尾插
pop_back() 尾删
```

queue

```
size()
empty()
push()  向队尾插入一个元素
front()  返回队头元素
back()  返回队尾元素
pop()  弹出队头元素
```

priority_queue, 优先队列，默认是大根堆

```
size()
empty()
push()  插入一个元素
```

```
top()  返回堆顶元素
pop()  弹出堆顶元素

FAQ: 默认是大根堆
     定义成小根堆的方式: priority_queue<int, vector<int>, greater<int>> q;
```

stack

```
size()
empty()
push()  向栈顶插入一个元素
top()  返回栈顶元素
pop()  弹出栈顶元素
```

deque

```
size()
empty()
clear()
front()/back()
push_back()/pop_back()
push_front()/pop_front()
begin()/end()
```

set, map, multiset, multimap, 基于平衡二叉树（红黑树），动态维护有序序列

```
size()
empty()
clear()
begin()/end()
FAQ: ++, -- 返回前驱和后继，时间复杂度 O(logn)
```

set/multiset

```
    insert()  插入一个数
    find()  查找一个数
    count()  返回某一个数的个数
    erase()
        (1) 输入是一个数x，删除所有x   O(k + logn) !
        (2) 输入一个迭代器，删除这个迭代器
    lower_bound()/upper_bound()
```

```
lower_bound(x)  返回大于等于x的最小的数的迭代器
upper_bound(x)  返回大于x的最小的数的迭代器
```

map/multimap

```
insert()  插入的数是一个pair
erase()  输入的参数是pair或者迭代器
find()

FAQ:  注意multimap不支持此操作。 时间复杂度是 O(logn)
      lower_bound()/upper_bound()
```

unordered ~

```
不支持 lower_bound()/upper_bound(),  迭代器的++, --
```

bitset, 压位

```
bitset<10000> s;
~, &, |, ^
>>, <<
==, !=

count()  返回有多少个1
any()  判断是否至少有一个1
none()  判断是否全为0
set()  把所有位置成1
set(k, v)  将第k位变成v
reset()  把所有位变成0
flip()  等价于~
flip(k) 把第k位取反
```