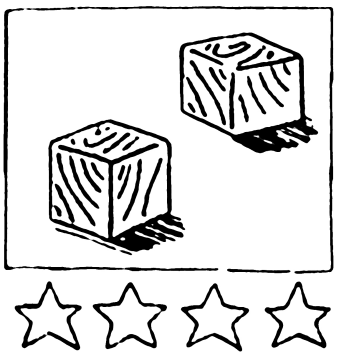
Wood Cube

For Manual/Intelligence

August 6, 2018

ACM-ICPC World Finals 2017

Reference Document



Fudan University

Wood Cube

Coach	教练
Weiwei Sun	孙未未

Contestant	队员
Zhihao Ma	马志豪
Dongjian Tang	汤栋坚
Xi Gao	高　熙

April 14, 2017

スタンダードコードライブラリー

庁界大辞典

Fudan U Wood Cube

1 Data Structure

1.1 Conclutions

1.1.1 经常忘记的思路

分块、压位、整体二分、前 8 位后 8 位。

1.1.2 多重集 $O(n)$ 哈希

随机一个 r，计算 $(r+a_1)(r+a_2)...(r+a_n)\%m$。注意 r 的取值范围，避免 $r+a_i=0$。

1.2 BIT Kth

```
int kth(int k) { // 返回第k个1的下标
  int re = 0, val = 0;
  for (int i = hbit; i; i >>= 1) // hbit取n的最高位或1<<19等
    if (re + i <= n && val + d[re + i] < k)
      val += d[re += i];
  return re + 1;
}
```

1.3 Weighted DSU

```
int root(int x) {
  return p[x] == x ? x
  : root(p[x]), s[x] = s[x] + s[p[x]], p[x] = p[p[x]]; }
bool merge(int x, int y, Val val) { // x -> y 连val的边
  if (root(x) == root(y)) return s[x] + -s[y] == val;
  s[p[x]] = -s[x] + val + s[y]; p[p[x]] = p[y];
  return true; } // 返回值表示连边是否成功(不与之前矛盾)
```

1.4 Splay

0. 请仔细思考需求是否能用 set、pbds、BIT 或各种魔改线段树搭配实现!!
1. 弃用记录 root 写法，维护单序列时请使用尾哨兵 ed。
2. 无尾哨兵时请用 insert/split 确保索引到 0(右 end) 后操作的正确性。
3. 请勿写出执行顺序不明确的复杂嵌套语句，易导致未定义行为。
4. 手动操作树时注意 $x=0, y=0, x=y$ 等情况，并及时 down/splay。

```
struct Node {
  int p, son[2]; int &operator[](int s) { return son[s]; }
  // Num cnt, cnts; Val val, vals; Key key; Ope laz; bool rev;
} node[mxv]; int sz = 0, ed = NEW(1, MAX_KEY);
#define X node[x], Y node[y], L node[X[0]], R node[X[1]]
int NEW(Num cnt,Val val){int x=++sz;X={0,0,0,};return sz;}
// 二叉树操作
int sid(int x) { return x == node[X.p][1]; }
void link(int x, int s, int y) {
  if (x) X[s] = y; if (y) Y.p = x; }
void cut(int x, int s, int y) { X[s] = Y.p = 0; }
void zig(int x) {
  int y = X.p, s = sid(x); link(y, s, X[!s]);
  link(Y.p, sid(y), x); link(x, !s, y); renew(y); }
int splay(int x, int top = 0) {
  down(x);
  while (X.p && X.p != top) {
    int y = X.p;
    // down(Y.p); down(y); down(x); // 自底向上
    if (Y.p && Y.p != top) zig(sid(x) == sid(y) ? y : x);
    zig(x);
  }
  return renew(x), x;
}
// 值、标记操作: 注意判!x, 修改操作后记得splay
void renew(int x) { if(x){
  X.cnts = L.cnts + X.cnt + R.cnts;
  X.vals = L.vals + X.val + R.vals; }}
void down(int x) {
  if(X.rev){swap(X[0],X[1]);rev(X[0]);rev(X[1]);X.rev=0;}
  if(X.laz){set(X[0],X.laz);set(X[1],X.laz);X.laz=0;}}
int set(int x, Ope ope){
  if(x){X.vals+=ope;X.val+=ope;X.laz+=ope;} return x;}
int rev(int x){if(x){X.rev ^= 1;} return x;}
// 序列操作: s为方向, 默认参数与函数名所示方向一致; 0均视为右end
int next(int x, int s = 1) { // 序列s方向x的下一个节点
  for (splay(x),down(x = X[s]); X[!s]; down(x = X[!s])) {}
  return splay(x); }
int ins(int x, int y, int s = 0) { // x的s边插入y子树
  next(x, s); link(x, s, y); return splay(y); }
int pick(int t, int x, int y) { // 左开右开, 返回区间不含x和y
  splay(x,splay(y)); return !x ? !y ? splay(t):Y[0]:X[1];}
int cutpick(int x) { // 切掉x子树后的剩余部分
  int y = X.p; cut(y, sid(x), x); return splay(y); }
int erase(int x) { // 切掉x节点后的剩余部分
  return cutpick(pick(x, next(x, 0), next(x, 1))); }
int cutleft(int x, int s = 0) { // 切掉x的s边(不含x)的序列
  splay(x); int y = X[s]; cut(x,s,y); return splay(x), y;}
pii split(int t, int x) { // 序列t按x(允许0)切成两半:{左, x+右}
  if (!x) return {splay(t), 0}; return {cutleft(x), x}; }
int last(int x, int s = 1) { // x的s方向的最后一个节点
  for (splay(x); X[s]; down(x = X[s])){} return splay(x);}
int insert(int t, int x, int y) { //序列t的x(允许0)左边插y子树
  return x ? ins(x, y) : ins(last(t), y, 1); }
int merge(int x, int y, int s = 1) { // 在序列x的s边接上序列y
  return x ? ins(last(y, !s), splay(x), !s) : splay(y); }
int rotfirst(int x, int s = 0) { // 将x旋转至序列的最s端
  int y = cutleft(x, s); return merge(x, y, !s); }
bool samelist(int x, int y) { // 判断x、y是否在同一个序列中
  splay(x); splay(y); return x == y || X.p; }
bool isleft(int x, int y, int s = 0) { // 判断x是否在y的s边
  splay(x, splay(y)); return Y[s] == x; }
// 索引操作: lbound/kth的x、pick等的t均用于指示其所在序列
int lbound(int x, Key key) {
  for (splay(x);;) {
    down(x); int s = !(key <= X.key); // ubound : "<"
    if (!X[s]) return s ? next(x) : splay(x); x = X[s]; }}
int spt(int x, Num ind) { // 拆点, 返回含ind的右半节点
  if (!x || !--ind) return x; // ind先减1
  int y = NEW(ind);
  tie(Y.val, X.val) = X.val.spt(ind); // | ind | X.cnt-ind |
  X.cnt -= ind; return ins(x, y), splay(x); }
int kth(int x, Num ind) { //不拆点"spt(x,ind)"改成"splay(x)"
  splay(x); if (!(1 <= ind && ind <= X.cnts)) return 0;
  for (;;) {
    down(x); if (ind <= L.cnts) x = X[0];
    else if ((ind -= L.cnts) <= X.cnt) return spt(x, ind);
    else ind -= X.cnt, x = X[1]; }}
int pickir(int x, Num l, Num r) { // 单序列改成int x = ed;
  return pick(x, next(kth(x, l), 0), kth(x, r + 1)); }
// 常见应用(单序列)
void set(Num l, Num r, Ope ope) {
  splay(set(pickir(l, r), ope)); }
Val get(Num l, Num r) { return node[pickir(l, r)].vals; }
void reverse(Num l, Num r) { splay(rev(pickir(l, r))); }
void remove(Num l, Num r) { cutpick(pickir(l, r)); }
void move(Num l, Num r, Num t) {
  int x = pickir(l, r); ins(kth(cutpick(x), t), x); }
void insert(Num t, int x) { ins(kth(ed, t), x); }
void pb(int x) { ins(ed, x); } // Q.pb(Q.NEW(1, val));
#undef X Y L R
```

1.5 Link-Cut Tree

```
struct Node {
  int p, son[2]; bool rev;
  int &operator[](int s) { return son[s]; }
} node[mxv];
#define X Y P L R = node[x, y, X.p, X[0], X[1]]
bool isr(int x) { return P[0] != x && P[1] != x; }
int sid(int x) { return P[1] == x; }
void link(int x, int s, int y) {
  if (x) X[s] = y; if (y) Y.p = x; }
void zig(int x) {
  int y = X.p, s = sid(x);
  link(y, s, X[!s]);
  if (!isr(y)) link(Y.p, sid(y), x); else X.p = Y.p;
  link(x, !s, y); renew(y);
}
void splay(int x) {
  if (!x) return; down(x);
  while (!isr(x)) {
    int y = X.p; down(Y.p); down(y); down(x);
    if (!isr(y)) zig(sid(x) == sid(y) ? y : x);
    zig(x);
  }
  renew(x);
}
void access(int x) {
  for (int t = 0; x; t = x, x = X.p) {
    splay(x); X[1] = t; renew(x);
}}
#define chk ({ if (!x || !y) return; })
// 有根树操作: 注意link/cut/relink 中x为y的父亲
void link(int x, int y) { chk; splay(y); Y.p = x; }
void cut(int x, int y){chk; access(x); splay(y); Y.p = 0;}
void pick(int x) { access(x); splay(x); } // x到根的路径
```

```
void relink(int x, int y) { cut(pof(y), y); link(x, y); }
// 无根树操作
void root(int x) { access(x); splay(x); rev(x); down(x); }
void link(int x, int y) { chk; root(y); Y.p = x; }
void cut(int x, int y){chk; pick(x,x); splay(y); Y.p = 0;}
void pick(int x, int y) { root(x); access(y); splay(x); }
void down(int x)
{ if(X.rev){swap(X[0],X[1]);rev(X[0]);rev(X[1]);X.rev=0;}}
void rev(int x) { if (x) X.rev ^= 1; }
// 通用操作: next/last见Splay模板,其返回改成return splay(x),x;
int pof(int x) { access(x); return next(x, 0); } // parent
int rof(int x) { access(x); return last(x, 0); } // root
bool sametree(int x, int y) { // 有根树把root()换成pick()
  root(x); root(y); return x == y || X.p; }
int lca(int x, int y) {
  access(x); access(y); splay(y); splay(x);
  return X.p ? X.p : Y.p ? x : x == y ? x : 0;
}
#undef X Y P L R
```

1.6 KD Tree

k 维: 区间操作 $O(k \cdot N^{1-\frac{1}{k}})$, 随机点最近点查询 $O(2^k + \log N)$, 用于区间操作建议每 $O(\sqrt{N \cdot \log N})$ 次插入重构一次, 用于随机点最近点查询建议不重构。

```
// clear => ((add => rebuild) | insert)
using poi = array<int,2>; struct rect { poi l, r; };
bool incl(rect a, rect b) {
  return a.l[0] <= b.l[0] && b.r[0] <= a.r[0]
      && a.l[1] <= b.l[1] && b.r[1] <= a.r[1]; }
bool itsc(rect a, rect b) {
  return a.l[0] <= b.r[0] && b.l[0] <= a.r[0]
      && a.l[1] <= b.r[1] && b.l[1] <= a.r[1]; }
#define nil mem
struct Node { poi d; rect f; Node *s[2]; // Node *p;
  Val val, vals; Ope laz; } mem[mxn]; using ptr = Node*;
int sz; ptr p[mxn], root;
void clear() { sz = 0; root = nil;
  *nil = {{}, {{MXL, MXL}, {-MXL, -MXL}}}; }
void add(poi d, Val val) {
  ++sz; *(p[sz] = mem+sz) = {d,{d,d},{nil,nil},val,val};}
void setsub(ptr x, Ope ope) {
  if (x == nil) return;
  x->val += ope; x->laz += ope; x->vals += ope; }
void down(ptr x) {
  if (x->laz) { setsub(x->s[0], x->laz);
    setsub(x->s[1], x->laz); x->laz = Ope(); }}
void renew(ptr x) {
  x->vals = x->s[0]->vals + x->s[1]->vals + x->val; }
void renewFrame(ptr x) {
  renew(x); x->f = {x->d, x->d}; // x->s[0]->p=x->s[1]->p=x;
  for (int i=0; i<=1; ++i) for (int j=0; j<=1; ++j) {
    x->f.l[j] = min(x->f.l[j], x->s[i]->f.l[j]);
    x->f.r[j] = max(x->f.r[j], x->s[i]->f.r[j]); }}
ptr build(ptr *l, ptr *r, int h = 0) {
  if (l >= r) return nil; auto m = l + (r - l) / 2;
  nth_element(l, m, r, [&](auto a, auto b) {
    return a->d[h] < b->d[h]; });
  (*m)->s[0] = build(l, m, h ^ 1);
  (*m)->s[1] = build(m + 1, r, h ^ 1);
  renewFrame(*m); return *m; }
void insert(ptr x, ptr y, int h = 0) {
  down(x); int t = x->d[h] < y->d[h];
  if (x->s[t] != nil) insert(x->s[t], y);
  else x->s[t] = y; renewFrame(x); }
void rdown(ptr x) {
  if (x == nil) return;
  down(x); rdown(x->s[0]); rdown(x->s[1]); }
void rebuild() { rdown(root); root = build(p+1, p+sz+1); }
void insert(poi d, Val val) { // 每插LIM个点重构一次
  add(d, val); if (sz % LIM == 0) rebuild();
  else if (root != nil) insert(root, mem + sz);
  else root = mem + sz; }
void set(ptr x, rect f, Ope ope) {
  if (x == nil || !itsc(f, x->f)) return;
  if (incl(f, x->f)) { setsub(x, ope); return; }
  down(x); if (incl(f, {x->d, x->d})) x->val += ope;
  set(x->s[0], f, ope); set(x->s[1], f, ope); renew(x); }
Val get(ptr x, rect f) {
  if (x == nil || !itsc(f, x->f)) return Val();
  if (incl(f, x->f)) return x->vals;
  return down(x), (incl(f, {x->d, x->d}) ? x->val : Val())
    + get(x->s[0], f) + get(x->s[1], f); }
Val getv(int id) { // 单点val, 需要维护父亲指针p, 见注释代码
  ptr x = mem + id; Val re = x->val; // 注意root->p是垃圾值
  while (x != root) x = x->p, re += x->laz; return re; }
poi d; ll ans; ptr sol;
void nearest(ptr x, int h = 0) { // 注意 "ll sqr(ll x)"
  if (x == nil) return;
  if (sqr(max({0, d[0] - x->f.r[0], x->f.l[0] - d[0]}))
    + sqr(max({0, d[1] - x->f.r[1], x->f.l[1] - d[1]}))
    >= ans) return;
  ll dist = sqr(d[0] - x->d[0]) + sqr(d[1] - x->d[1]);
  if (ans > dist) ans = dist, sol = x; // 重点:&& dist > 0
  int t = x->d[h] <= d[h];
  nearest(x->s[t], h ^ 1); nearest(x->s[1 - t], h ^ 1); }
pair<int,ll> nearest(poi d) { // {id, sqr(dist)}
  this->d = d; ans = sqr(MXL * 2) * 2; sol = nil;
  nearest(root); return {sol - mem, ans}; }
```

1.7 Persistent Segment Tree

注意: 操作可持久化数据结构时请勿在同一语句中使用多个 NEW/COPY 宏。
多个可持久化数据结构同时使用: 套 namespace 并将 NEW/COPY 改成函数。

```
#define nil mem
struct Node { Node *l, *r; Val val; }
  mem[mxv] = {{nil, nil}}; int sz; using ptr = Node*;
#define NEW(arg...) new(mem + ++sz)Node{arg}
#define COPY(x) new(mem + ++sz)Node(*(x))
#define m (l + r >> 1)
ptr renew(ptr x) { x->val=x->l->val+x->r->val; return x; }
ptr build(int l, int r) {
  if (l == r) return NEW(nil, nil, Val());
  return renew(NEW(build(l, m), build(m + 1, r)));
}
Val Get(ptr x, int l, int r, int sl, int sr) {
  if (sl <= l && r <= sr) return x->val;
  if (sr <= m) return Get(x->l, l,m, sl,sr);
  if (m < sl) return Get(x->r, m+1,r, sl,sr);
  return Get(x->l, l,m, sl,sr) + Get(x->r, m+1,r, sl,sr);
}
ptr Set(ptr x, int l, int r, int s, Ope ope) {
  x = COPY(x); // 值域线段树: if (x == nil) x = NEW(nil,nil);
  if (l == r) { x->val += ope; return x; }
  if (s <= m) x->l = Set(x->l, l, m, s, ope);
  else x->r = Set(x->r, m + 1, r, s, ope);
  return renew(x);
}
#undef m
```

1.8 Merge-Split Treap

合并两个历史版本在构造数据下深度会不断退化, 可达 log 的几十倍。

```
#define nil mem
struct Node {int fit; Node *l, *r; Key key; Val val, vals;
} mem[mxv] = {{0, nil, nil}}; int sz; using ptr = Node*;
#define NEW(arg...) new(mem+ ++sz)Node{rand(),nil,nil,arg}
ptr down(ptr x) {x = COPY(x); if (x->laz) {...} return x;}
pair<ptr,ptr> split(ptr x, Key key) {
  ptr t; if (x == nil) return {nil, nil};
  x = down(x);
  return x->key >= key // 节点割在右半边的条件
    ? (tie(t, x->l) = split(x->l, key), mp(t, renew(x)))
    : (tie(x->r, t) = split(x->r, key), mp(renew(x), t));
}
ptr merge(ptr x, ptr y) {
  if (x == nil) return y;
  if (y == nil) return x;
  return x->fit < y->fit // rand() % (X.cnts + Y.cnts) < X.cnts
    ? (x = down(x), x->r = merge(x->r, y), renew(x))
    : (y = down(y), y->l = merge(x, y->l), renew(y));
}
```

1.9 Leftist Tree

```
#define nil mem
struct Node { Node *l, *r; int dist; Val val; }
  mem[mxv] = {{nil, nil, -1}}; int sz = 0;
#define NEW(arg...) (new(mem + ++sz)Node{nil,nil,0,arg})
//add(x,ope){if(x!=nil){x->val+=ope,x->laz+=ope;}}
//down(x){if(x->laz){add(x->l,x->laz);add(x->r,x->laz);x->laz=0;}}
Node *merge(Node *x, Node *y) {
  if (x == nil) return y;
  if (y == nil) return x;
  if (y->val < x->val) swap(x, y); // 默认小根堆
  // down(x); // 朱刘算法下传标记预留位置
```

```
  x->r = merge(x->r, y);
  if (x->l->dist < x->r->dist) swap(x->l, x->r);
  x->dist = x->r->dist + 1;
  return x;
}
Node *pop(Node *x) { /*down(x);*/ return merge(x->l, x->r);}
```

1.10 Confluently Persistent Heap

```
#define nil mem
struct Node { Node *l, *r, *s; int dist; Val val, laz;
} mem[mxv]={{nil,nil,nil,-1}}; int sz; using ptr = Node*;
#define NEW(arg...) new(mem + ++sz)Node{nil,nil,nil,0,arg}
#define COPY(x) new(mem + ++sz)Node(*(x))
ptr add(ptr x, Val ope) {
  if (x == nil) return nil;
  x = COPY(x); x->val += ope; x->laz += ope; return x; }
ptr down(ptr x) {
  x = COPY(x); if (x->laz)
  { x->l = add(x->l, x->laz); x->r = add(x->r, x->laz);
    x->s = add(x->s, x->laz); x->laz = 0; } return x; }
ptr sub_merge(ptr x, ptr y) {
  if (x == nil) return y; if (y == nil) return x;
  if (cmp(y->val, x->val)) swap(x, y);
  x = down(x); x->r = sub_merge(x->r, y);
  if (x->l->dist < x->r->dist) swap(x->l, x->r);
  x->dist = x->r->dist + 1; return x; }
ptr merge(ptr x, ptr y) {
  if (x == nil) return y; if (y == nil) return x;
  if (cmp(y->val, x->val)) swap(x, y); // 默认小根堆(less)
  x = down(x); x->s = sub_merge(x->s, y); return x; }
ptr pop(ptr x) { // pop操作注意仔细计算复杂度
  x = down(x); x = x->s; x = down(x);
  x->s = sub_merge(x->s, sub_merge(x->l, x->r));
  x->l = x->r = nil; return x; }
```

1.11 Confluently Persistent Trie

考虑 Trie 的 Σ^N 暴力做法，然后倒着用可持久化 Trie 构造状态 DAG。大字符集可以把字符拆成二进制串。请考虑清楚汇点被 COPY 时的意义。

```
ptr T = NEW(nil, nil), S = T;
for (int i = n; i >= 1; --i) {
  S = NEW(S, S);
  for (auto &s : str[i])
    S = rem(S, s, 0); // (可持久化)删除字符串s对应的trie子树
}
```

2 Tree

2.1 Conclusions

2.1.1 虚树的直径

点集并的直径端点 $\subset$ 每个点集直径端点的并；可以用 dfs 序的 ST 表维护子树直径，建议使用 RMQLCA。

2.2 Heavy Path Decomposition

```
#define fors(i) for (auto i : e[x]) if (i != p)
int cnt; ai s, h, top, pa, dfn /*,hea*/;
int size(int x, int p)
{ s[x] = 1; fors(i) s[x] += size(i, x); return s[x]; }
void dfs(int x, int p, int t) {
  pa[x] = p, top[x] = t, h[x] = h[p] + 1, dfn[x] = ++cnt;
  int y = 0; // int &y = hea[x] = 0;
  fors(i) if (s[y] < s[i]) y = i;
  if (y) dfs(y, x, t);
  fors(i) if (i != y) dfs(i, x, i);
}
void build() { size(1, 0); cnt = 0; dfs(1, 0, 1); }
void path(int x, int y) {
  while (top[x] != top[y])
    if (h[top[x]] >= h[top[y]]) {
      foo(dfn[top[x]], dfn[x]); x = pa[top[x]];
    } else { // swap(x, y); 边权无向时可改用这句
      foo(dfn[top[y]], dfn[y]); y = pa[top[y]];
    }
  if (dfn[x] < dfn[y]) foo(dfn[x], dfn[y]);
  else foo(dfn[y], dfn[x]);
}
void subtree(int x) { foo(dfn[x], dfn[x] + s[x] - 1); }
```

2.3 树点分治

```
bool ena[mxn]; int s[mxn];
#define fore(i) for (auto i : e[x]) if (!ena[i])
#define fors(i) fore(i) if (i != p)
int size(int x, int p)
{ s[x] = 1; fors(i) s[x] += size(i, x); return s[x]; }
int core(int x, int p) {
  fors(i) if (s[x] - s[i] < s[i])
    return s[x] -= s[i], s[i] += s[x], core(i, x);
  return x;
}
void divide(int x) {
  x = core(x, 0); // s[x]为子树大小
  ena[x] = true;
  fore(i) divide(i);
  fore(i) foo(i, 0, ...);
  ena[x] = false;
}
size(1, 0); divide(1); // "1"可换成任意点
```

2.4 Virtual Tree

```
vi V; int p;
void link(int x, int y) { if (y && x != y) E[x].pb(y); }
int dtree(int l, int x) {
  if (p == V.size() || h[l] > h[V[p]]) return x;
  int y = V[p++]; link(y, x); link(y, dtree(y, 0));
  return dtree(l, y);
}
void make(vi &v) {
  // sort(..., [&](int a, int b) { return dfn[a] < dfn[b]; });
  int t = v.back(); V = {};
  for (auto i : v) { V.pb(lca(t, i)); V.pb(t = i); }
  for (auto i : V) E[i] = {};
  p = 0; dtree(0, 0);
}
```

2.5 Tree Isomorphism

```
#define fors(i) for (auto i : e[x]) if (i != p)
struct Val {
  ll x, p; static const ll MOD = 1E18L+3;
  Val() : x(0), p(1) {}
  Val(char c) : x(c), p(128) {}
  Val(ll x, ll p) : x(x), p(p) {}
  Val operator+(Val b) const {
    return {((__int128)x * b.p + b.x) % MOD,
      (__int128)p * b.p % MOD}; }
  bool operator<(Val b) const {
    return tie(x, p) < tie(b.x, b.p); }
  Val wrap() { return Val('(') + *this + ')'; }
};
struct Sub {
  multimap<Val,Val> s;
  void operator+=(Val b) { s.insert({b, {}}); }
  Val hash() { Val re; for (auto i : s) re = re + i.fir;
    return re.wrap(); }
  void build() {
    Val l, r; for (auto &i : s) i.sec = l, l = l + i.fir;
    for (auto i = s.rbegin(); i != s.rend(); ++i)
      i->sec = i->sec + r, r = i->fir + r; }
  Val operator-(Val b) {
    return s.lower_bound(b)->sec.wrap(); }
};
Val D[mxn], U[mxn], A[mxn]; Sub tree[mxn];
void dfsD(int x, int p) {
  tree[x] = {};
  fors(i) dfsD(i, x), tree[x] += D[i];
  D[x] = tree[x].hash();
}
void dfsU(int x, int p) {
  if (p) tree[x] += U[x];
  A[x] = tree[x].hash(), tree[x].build();
  fors(i) U[i] = tree[x] - D[i], dfsU(i, x);
}
```

2.6 Pseudotree, Cactus

```
// 基环无向树
vi °ring, °edge[mxn]; int °vis[mxn]; // int类型!!
int ptree(int x, int p) {
```

```
  vis[x] = 1; int a = 0;
  for (auto i : e[x]) if (i != p)
    if (!vis[i]) {
      int t = ptree(i, x);
      if (t) a = (t == x) ? -1 : t;
    } else if (vis[i] == 1) {
      a = i, ring = {i};
    }
  vis[x] = 2;
  if (a > 0) ring.pb(x);
  if (!a && p) edge[p].pb(x);
  if (!~a && p) edge[x].pb(p);
  return a;
}
// 仙人掌：无视自环，有重边则需改边表及p
vi e[mxn]; int *dfn[mxn], *cnt;
vi *edge[mxn]; vector<vi> *ring[mxn]; // 仙人掌边与环
int cactus(int x, int p) { // 返回返祖边祖先编号或0
  dfn[x] = ++cnt; int a = 0;
  for (auto i : e[x]) if (i != p)
    if (!dfn[i]) {
      int t = cactus(i, x);
      if (t && t != x) assert(!a), a = t;
    } else if (dfn[i] < dfn[x]) {
      assert(!a), a = i;
      ring[i].pb({i}); // ring[x][][0] == x, 即环顶
    }
  if (a) ring[a].back().pb(x);
  if (!a && p) edge[p].pb(x);
  return a;
}
int ans, f[mxn]; // 示例：仙人掌直径DP
void dfs(int x) {
  multiset<int> s = {0, 0}; // 堆(或O(1))维护最长、次长链
  for (auto i : edge[x]) { dfs(i); s.insert(f[i] + 1); }
  for (auto &r : ring[x]) {
    // f[c[i]], f[r[i]] 不要忘记套 c[i], r[i] !!
    for (auto i : r) if (i != x) dfs(i);
    int m = r.size(); vi c = r; for (auto i : r) c.pb(i);
    int fr = 0; // 经过环到x的(不绕远)最长链
    for (int i = 1; i < m; ++i)
      fr = max(fr, f[r[i]] + min(i, m - i));
    s.insert(fr);
    deque<int> q = {0}; // 单调队列维护环DP
    for (int i = 1; i < m * 2; ++i) { // 下面两句一般无需判空
      if ((i - q[0]) * 2 > m) q.pop_front();
      ans = max(ans, f[c[q[0]]] - q[0] + f[c[i]] + i);
      while (!q.empty()
        && f[c[q.back()]] - q.back() <= f[c[i]] - i)
          q.pop_back();
      q.pb(i); }}
  f[x] = *s.rbegin();
  ans = max(ans, *s.rbegin() + *next(s.rbegin())); }
```

3 Graph Theory

3.1 Conclusions

3.1.1 图在删边后连通性判定

随机一个生成树，给非树边随机权值，树边为经过它的非树边的权值的 xor；一个边的集合在删去后使得图不连通 <=> 非空子集边权异或和 $= 0$。

3.1.2 三元环、四元环

无重边、自环的稀疏图的三元环个数为 $O(N\sqrt{N})$：$deg \geq \sqrt{N}$ 的点三方枚举，$deg < \sqrt{N}$ 的点枚举两个相邻点。

稀疏二分图的四元环计数 (复杂度 $O(N\sqrt{N})$)：左排点中 $deg \geq \sqrt{N}$ 的点 $O(M)$ 做两遍计数，然后忽略这些大度数点，枚举右排点后遍历从这个点出发的所有 L 形，同时维护 L 形终点的计数器。

3.1.3 欧拉回路

无向图建议使用 multiset 边表每步 erase，有向图可以使用 vector 边表每步 pop_back。

3.1.4 图同构

$$F_t(i) = (F_{t-1}(i) \times A + \sum_{i \to j} F_{t-1}(j) \times B + \sum_{j \to i} F_{t-1}(j) \times C + D \times (i = a)) \bmod P$$

枚举点 a，迭代 K 次后求得的就是 a 点所对应的 hash 值。其中 K, A, B, C, D, P 为 hash 参数，可自选。

3.1.5 Prufer Code

【根据树构造】我们通过不断地删除顶点编过号的树上的叶子节点直到还剩下 2 个点为止的方法来构造这棵树的 Prüfer sequence。特别的，考虑一个顶点编过号的树 T，点集为 $1, 2, 3, \ldots, n$。在第 i 步中，删除树中编号值最小的叶子节点，设置 Prüfer sequence 的第 i 个元素为与这个叶子节点相连的点的编号。

【还原】设 a_i 是一个 Prüfer sequence。这棵树将有 $n+2$ 个节点，编号从 1 到 $n+2$，对于每个节点，计它在 Prüfer sequence 中出现的次数 +1 为其度数。然后，对于 a 中的每个数 a_i，找编号最小的度数值为 1 节点 j，加入边 (j, a_i)，然后将 j 和 a_i 的度数值减少 1。最后剩下两个点的度数值为 1，连起来即可。

【一些结论】完全图 K_n 的生成树，顶点的度数必须为 $d_1, d_2, \ldots, d_n$，这样的生成树棵数为：

$$\frac{(n-2)!}{[(d_1-1)!(d_2-1)!(d_3-1)!\ldots(d_n-1)!]}$$

一个顶点编号过的树，实际上是编号的完全图的一棵生成树。通过修改枚举 Prüfer sequence 的方法，可以用类似的方法计算完全二分图的生成树棵数。如果 G 是完全二分图，一边有 n_1 个点，另一边有 n_2 个点，则其生成树棵数为 $n_1^{n_2-1} * n_2^{n_1-1}$。

3.1.6 生成树计数

对于 n 个点的无向图的生成树计数，令矩阵 D 为图 G 的度数矩阵，即 $D = diag(deg_1, deg_2, \ldots, deg_n)$，$A$ 为 G 的邻接矩阵表示，则 $D - A$ 的任意一个 $n-1$ 阶主子式的行列式的值即为答案。

3.1.7 平面图

【Euler Characteristic】$\chi = V - E + F$。其中，V 为点数，E 为边数，F 为面数，对于平面图即为划分成的平面数（包含外平面），χ 为对应的欧拉示性数，对于平面图有 $\chi = C + 1$，C 为连通块个数。

【Dual Graph】将原图中所有平面区域作为点，每条边若与两个面相邻则在这两个面之间连一条边，只与一个面相邻连个自环，若有权值 (容量) 保留。

【Maxflow on Planar Graph】连接 s 和 t，显然不影响图的平面性，转对偶图，令原图中 s 和 t 连接产生的新平面在对偶图中对应的节点为 s'，外平面对应的顶点为 t'，删除 s' 和 t' 之间直接相连的边。此时 s' 到 t' 的一条最短路就对应了原图上 s 到 t 的一个最大流。

3.2 Tarjan SCC, Bridge, Cut Point

```
// 2-SAT: A-B矛盾 A->B', B->A'; 注意A<->B', B<->A'的伪2-SAT题
ai *dfn, low, scc; int *cnt, *scnt; // SCC (Tarjan)
stack<int,vi> sta; bool ins[mxn];
int tarjan(int x) {
  dfn[x] = low[x] = ++cnt;
  sta.push(x); ins[x] = true;
  for (auto i : e[x]) low[x] = min(low[x],
    dfn[i] ? dfn[ins[i] ? i : x] : tarjan(i));
  if (low[x] == dfn[x]) {
    ++scnt; int t;
    do { t = sta.top(); sta.pop(); ins[t] = false;
      scc[t] = scnt; } while (t != x);
  }
  return low[x];
}
ai *dfn, low; int *cnt; bool *bri[mxm]; // Bridge
int tarjan(int x, int pe) { // 无重边时只需vi边表并传入父节点p
  dfn[x] = low[x] = ++cnt;
  fore(i) if (e != pe) low[x] = min(low[x],
    (dfn[i] ? dfn[i] : tarjan(i, e ^ 1)));
  if (low[x] == dfn[x] && pe)
    bri[pe] = bri[pe ^ 1] = true; // 建树时注意判掉反向边
  return low[x];
}
ai *dfn, low; int *cnt; int *cut[mxn]; // Cut Point
int tarjan(int x, int p) { // vi边表、不记录点双版本
  dfn[x] = low[x] = ++cnt; int ecnt = 0;
  for (auto i : e[x]) if (i != p)
    if (!dfn[i]) { // 搜索树子树,dfn[x]<=low[i]时亦为BCCV树子树
      low[x] = min(low[x], tarjan(i, x)); ++ecnt;
      if (dfn[x] <= low[i] && p) cut[x] = true;
    } else low[x] = min(low[x], dfn[i]);
  if (!p && ecnt > 1) cut[x] = true;
  return low[x];
}
ai *dfn, low; int *cnt; bool *cut[mxn]; // BCCV, Cut Point
stack<int,vi> sta; vector<vi> *bcc;
int tarjan(int x, int pe) {
  dfn[x] = low[x] = ++cnt; int ecnt = 0;
  fore(i) if (e != pe) {
    if (dfn[i] < dfn[x]) sta.push(e); // 自环大概会被无视
    if (!dfn[i]) {
      low[x] = min(low[x], tarjan(i, e ^ 1));
      ++ecnt;
      if (low[i] >= dfn[x]) {
        bcc.pb({});
        do { bcc.back().pb(sta.top()); sta.pop();
          } while (bcc.back().back() != e);
        if (pe) cut[x] = true;
      }
    } else low[x] = min(low[x], dfn[i]);
  }
  if (!pe && ecnt > 1) cut[x] = true;
  return low[x];
}
```

```
for (int i = 1; i <= n; ++i) if (cut[i]) MAP[i] = ADDV();
for (auto &es : bcc) {
  set<int> s; // 改用hashset或bool数组可以减掉log
  for (auto e : es) s.insert({ev[e], ev[e ^ 1]});
  int x = ADDV(s.size() - 1);
  for (auto i : s)
    if (cut[i]) adde2(x, MAP[i]); else MAP[i] = x;
}
```

3.3 Hungarian, Hopcroft, Vertex Cover

```
vector<int> e[mxn]; int mat[mxn]; #define forl ...
bool v[mxn]; // hungarian
bool extend(int x) {
  for (auto i : e[x]) if (!v[i]) {
    v[i] = true;
    if (!mat[i] || extend(mat[i]))
    { mat[i] = x; mat[x] = i; return true; }
  }
  return false;
}
int hungarian() {
  int re = 0; fill(mat, mat + n + 1, 0);
  forl(i) { fill(v, v + n + 1, 0); re += extend(i); }
  return re;
}
int lev[mxn]; // hopcroft O(M*sqrt(N)), 随机数据O(M*log(N))
bool extend(int x) {
  for (auto i : e[x]) if (lev[mat[i]] == lev[x] + 1)
    if (!mat[i] || extend(mat[i]))
    { mat[i] = x; mat[x] = i; return true; }
  lev[x] = 0;
  return false;
}
int q[mxn], *qb, *qe;
int hopcroft() {
  int re = 0; fill(mat, mat + n + 1, 0);
  for (;;) {
    qb = qe = q; fill(lev, lev + n + 1, 0);
    forl(i) if (!mat[i]) lev[*qe++ = i] = 1;
    while (qb != qe) {
      int x = *qb++;
      if (lev[0]) break;
      for (auto j : e[x]) if (!lev[mat[j]])
        lev[*qe++ = mat[j]] = lev[x] + 1;
    }
    if (!lev[0]) break;
    forl(i) if (!mat[i]) re += extend(i);
  }
  return re;
}
bool cov[mxn]; // vertex cover
void mark(int x) {
  cov[x] = true;
  for (auto i : e[x]) if (!cov[i]) {
    cov[i] = true;
    if (mat[i]) mark(mat[i]);
}}
void vertexCover() { // 最小点覆盖集，最大点独立集与之互补
  fill(cov, cov + n + 1, false);
  forl(i) if (!mat[i]) mark(i);
  forl(i) cov[i] = !cov[i];
}
```

3.4 KM

最大权匹配, 过程保证 $l_i + r_j \geq e_{ij}$; 复杂度 $O(N^3)$, 随机数据约 $O(N^{2.5})$;
输入需满足 $n \leq m; max\{|e_{ij}|\} * 2n < MXL < maxint/2$;

```
Val e[mxn][mxn], l[mxn], r[mxn], s[mxn]; // e[][]初始-MXL
int p[mxn], f[mxn]; bool v[mxn];
Val km(int n, int m) { // 匹配方案: p[R] = L
  fill(p, p + m + 1, 0);
  fill(r, r + m + 1, 0);
  for (int i = 1; i <= n; ++i)
    r[0] -= l[i] = *max_element(e[i] + 1, e[i] + m + 1);
  for (int i = 1; i <= n; ++i) {
    fill(s, s + m + 1, MXL * 2);
    fill(v, v + m + 1, false);
    int y = 0, t; p[0] = i;
    do {
      v[y] = true; int x = p[y]; Val d = MXL * 2;
      for (int j = 1; j <= m; ++j) if (!v[j]) {
        Val c = l[x] + r[j] - e[x][j];
        if (c < s[j]) s[j] = c, f[j] = y;
        if (s[j] < d) d = s[j], t = j;
      }
      if (e[p[f[t]]][t] == -MXL) return -MXL; // 无完备匹配
      for (int j = 0; j <= m; ++j)
        if (v[j]) l[p[j]] -= d, r[j] += d; else s[j] -= d;
    } while (p[y = t]);
    while (y) p[y] = p[f[y]], y = f[y];
  }
  return -r[0]; // double边权可以使用精度更高的sum{l} + sum{r}
}
```

3.5 Dinic

1. Float_Dinic 外部用于判断满流时允许的误差应设为 eps 的边数倍。
2. 令 G' 为残量网络 G 在缩强连通分量之后的图:
一定在最小割方案中的边: uv 满流，且在 G' 中 $u = S$，$v = T$;
可能在最小割方案中的边: uv 满流，且在 G' 中 $u \neq v$;
3. 对二分图而言:
在最大流方案中一定满流的边: uv 满流，且在 G' 中 $u \neq v$;
在最大流方案中可能满流的边: uv 满流，或 (uv 流量为 0, 且在 G' 中 $u = v$);

```
int n, m, fe[mxn], ne[mxm], ev[mxm]; Flow f[mxm];
void clear(int n)
{ this->n = n; fill(fe, fe + n + 1, 0); m = 1; }
void add_(int x, int y, Flow F)
{ ne[++m] = fe[x]; fe[x] = m; ev[m] = y; f[m] = F; }
void add1(int x, int y, Flow F)
{ add_(x, y, F); add_(y, x, 0); }
int T, arc[mxn], lev[mxn];
Flow extend(int x, Flow lim) {
  if (x == T) return lim;
  Flow re = 0;
  for (int &e = arc[x]; e; e = ne[e])
    if (f[e] && lev[ev[e]] == lev[x] + 1) { // sgn(f[e]) &&
      Flow t = extend(ev[e], min(lim - re, f[e]));
      if (t) { // sgn(t)
        f[e] -= t, f[e ^ 1] += t;
        if ((re += t) == lim) break; // !sgn((re+=t), lim)
    }}
  if (!re) lev[x] = 0; // !sgn(re)
  return re;
}
int q[mxn], *qb, *qe;
Flow dinic(int S, int T) {
  Flow re = 0; this->T = T;
  for (;;) {
    fill(lev, lev + n + 1, 0);
    copy(fe, fe + n + 1, arc);
    for (qb = qe = q, lev[*qe++ = S] = 1; qb != qe;) {
      int x = *qb++;
      if (lev[T]) break;
      for (int e = fe[x]; e; e = ne[e])
        if (f[e] && !lev[ev[e]]) // sgn(f[e]) &&
          lev[*qe++ = ev[e]] = lev[x] + 1;
    }
    if (!lev[T]) break;
    re += extend(S, MXL);
  }
  return re;
}
```

3.6 MCMF

```
#define fore(i,e,x) for(int i,e=fe[x]; i=ev[e],e; e=ne[e])
int n,m,fe[mxn],ne[mxm],ev[mxm]; Flow f[mxm]; Cost c[mxm];
void clear(int n)
{ this->n = n; fill(fe, fe + n + 1, 0); m = 1; }
void add_(int x, int y, Flow F, Cost C)
{ ne[++m] = fe[x]; fe[x] = m; ev[m] = y; f[m]=F; c[m]=C; }
void add1(int x, int y, Flow F, Cost C)
{ add_(x, y, F, C); add_(y, x, 0, -C); }
Cost d[mxn]; bool inq[mxn]; int p[mxn]; // 普通spfa费用流
void spfa(int S) {
  fill(d, d + n + 1, MXL); d[S] = 0; //省略fill(inq);inq[S]...
  for (deque<int> q = {S}; !q.empty(); ) {
    int x = q.front(); q.pop_front(), inq[x] = false;
    fore(i,e,x) if (f[e] && d[i] > d[x] + c[e]) {
      d[i] = d[x] + c[e], p[i] = e;
      if (!inq[i]) q.push_back(i), inq[i] = true;
}}}
Cost mcmf(int S, int T) {
```

```
  Cost cost = 0;
  while (spfa(S), d[T] < MXL) { // 可行流: < 0;
    Flow t = MXL; // MX_FLOW
    for (int i = T; i != S; i = ev[p[i] ^ 1])
      t = min(t, f[p[i]]);
    for (int i = T; i != S; i = ev[p[i] ^ 1])
      f[p[i]] -= t, f[p[i] ^ 1] += t;
    cost += d[T] * t; // 注意Cost可能会爆int
  }
  return cost;
}
// 原始对偶算法, 跑完会破坏f[]和c[]
int S, T; Cost cost, dist, d[mxn]; bool v[mxn];
Flow extend(int x, Flow lim) { // 从T开始反向增广
  if (x == S) return cost += dist * lim, lim; // flow+=lim,
  Flow re = 0; v[x] = true;
  fore(i,e,x) if (f[e ^ 1] && !c[e] && !v[i]) {
    Flow t = extend(i, min(lim - re, f[e ^ 1]));
    f[e] += t, f[e ^ 1] -= t;
    if ((re += t) == lim) { v[x] = false; break; }
  }
  return re;
}
void dijk() { // 稠密图可改用O(N^2)dijkstra
  fill(d, d + n + 1, MXL); d[S] = 0;
  using P = pair<Cost,int>;
  priority_queue<P,vector<P>,greater<P>> h;
  for (h.push({0, S}); !h.empty(); ) {
    Cost t; int x; tie(t, x) = h.top(); h.pop();
    if (x == T) break; if (t > d[x]) continue;
    fore(i,e,x) if (f[e] && d[i] > d[x] + c[e])
      h.push({d[i] = d[x] + c[e], i});
  }
  for (int i = 1; i <= n; ++i) d[i] = min(d[i], d[T]);
}
void label() {
  for (int i = 2; i <= m; ++i)
    c[i] -= d[ev[i]] - d[ev[i ^ 1]];
  dist += d[T];
}
Cost mcmf(int S, int T) {
  this->S = S, this->T = T; cost = dist = 0;
  // spfa(), label(); // 有负权边时
  while (dijk(), label(), d[T] < MXL) // 可行流: && dist < 0
    do fill(v, v + n + 1, false); while (extend(T, MXL));
  return cost;
}
/* 消圈算法: 1.用spfa(初始所有点都要入队)找入队n+1次的点x 2.沿p[x]
走到第一个重复点(环上点)y 3.沿p[y]处理环, 方便起见请用!v[i]判环尾 */
```

3.7 Stoer-Wagner

无向图全局最小割。跑完会破坏边表。注释代码用于记录点集。复杂度 $O(V^3)$
堆优化后为 $O(VE\log V)$，斐波那契堆优化后为 $O(VE+V^2\log V)$

```
Val e[mxn][mxn]; Val s[mxn]; int p[mxn];
Val sw(int n) {
  Val ans = MXL;
  // list<int> sol, c[mxn]; for (i = 1..n) c[i] = {i};
  iota(p, p + n + 1, 0);
  for (; n > 1; --n) {
    for (int i = 1; i <= n; ++i) s[p[i]] = 0;
    for (int i = 1; i < n; ++i) {
      int t = i + 1;
      for (int j = i + 1; j <= n; ++j)
        if ((s[p[j]] += e[p[i]][p[j]]) > s[p[t]]) t = j;
      swap(p[i + 1], p[t]);
    }
    if (ans > s[p[n]]) { ans = s[p[n]]; /*sol = c[p[n]];*/}
    // c[p[n - 1]].splice(c[p[n - 1]].end(), c[p[n]]);
    for (int i = 1; i <= n - 2; ++i)
      e[p[n-1]][p[i]] = e[p[i]][p[n-1]] += e[p[i]][p[n]];
  }
  return ans;
}
vector<pair<int,Val>> e[mxn]; // 堆优化版本: O(VElogE)
int p[mxn]; bool v[mxn]; Val s[mxn]; DSU U;
Val sw(int n) {
  Val ans = MXL; iota(p, p + n + 1, 0); U.clear(n);
  for (; n > 1; --n) { priority_queue<pair<Val,int>> h;
    for (int i = 1; i <= n; ++i)
      v[p[i]] = false, h.push({s[p[i]] = 0, p[i]});
    for (int i = 1; i <= n; ) {
      Val t; int x, y; tie(t, x) = h.top(); h.pop();
      if (t < s[x]) continue; p[i++] = x, v[x] = true;
      for (auto j : e[x]) if (!v[y = U[j.fir]] && j.sec)
        h.push({s[y] += j.sec, y}); }
    if (ans > s[p[n]]) ans = s[p[n]];
    for (auto i : e[p[n]]) e[p[n - 1]].pb(i);
    e[p[n]] = {}; U.merge(p[n], p[n - 1]); }
  return ans; }
```

3.8 Gomory-Hu Tree

```
void build(int *l, int *r) { // 左闭右开
  auto t = r - 1; if (l >= t) return;
  random_shuffle(l, r);
  G.reset(); // 重置流量
  add2(*l, *t, G.dinic(*l, *t)); // 添加树边
  fill(G.v, G.v + G.n + 1, false); G.dfscut(*l); // 求割集
  auto m = partition(l, r, [](int x){return G.v[x];});
  build(l, m); build(m, r);
}
```

3.9 Dominator Tree

dom 为输出结果，即每个点在支配树中的儿子集合。复杂度 $O(E+V\log V)$
DAG 还可按拓扑序建树: 一个点的父亲是它的所有前驱在树中的 LCA。(不推荐)

```
ai dfn, id, pa, semi, idom, p, mn; avi be, dom; int cnt;
void dfs(int x) {
  dfn[x] = ++cnt; id[cnt] = x;
  for (auto i : e[x]) {
    if (!dfn[i]) { dfs(i); pa[dfn[i]] = dfn[x]; }
    be[dfn[i]].pb(dfn[x]);
}}
int get(int x) {
  if (p[x] != p[p[x]]) {
    if (semi[mn[x]] > semi[get(p[x])]) mn[x] = get(p[x]);
    p[x] = p[p[x]];
  }
  return mn[x];
}
void LT() {
  for (int i = cnt; i > 1; --i) {
    for (auto j : be[i]) minto(semi[i], semi[get(j)]);
    dom[semi[i]].pb(i);
    int x = p[i] = pa[i];
    for (auto j : dom[x])
      idom[j] = (semi[get(j)] < x ? get(j) : x);
    dom[x] = {};
  }
  for (int i = 2; i <= cnt; ++i) {
    if (idom[i] != semi[i]) idom[i] = idom[idom[i]];
    dom[id[idom[i]]].pb(id[i]);
}}
void build(int s) {
  for (int i = 1; i <= n; ++i) {
    dfn[i] = 0; dom[i] = be[i] = {};
    p[i] = mn[i] = semi[i] = i;
  }
  cnt = 0; dfs(s); LT();
}
```

3.10 Chu-Liu

```
// struct Node { Node *l, *r; int dist; int x, y; Val val, laz; };
DSU W, S; Node *H[mxn], *pe[mxn]; // 带lazy标记左偏树
Val chuliu(int s) { // O(ElogE), 加边: NEW(x, y, w);
  Val re = 0; W.clear(n); S.clear(n); // int rid = 0;
  fill(H, H + n + 1, (Node*)nil);
  for (auto i = mem + 1; i <= mem + sz; ++i)
    H[i->y] = merge(i, H[i->y]);
  for (int i = 1; i <= n; ++i) if (i != s)
    for (;;) {
      auto in = H[S[i]]; H[S[i]] = pop(H[S[i]]);
      if (in == nil) return MXL; // 无解
      if (S[in->x] == S[i]) continue;
      re += in->val; pe[S[i]] = in; //if(in->x==s) 实根=in->y;
      add(H[S[i]], -in->val); // add注意判nil和加laz
      if (W[in->x] != W[i]) { W.merge(in->x, i); break; }
      // G[in->x].pb({in->y, ++rid});
      for (int j = S[in->x]; j != S[i]; j = S[pe[j]->x]) {
        // G[pe[j]->x].pb({pe[j]->y, rid});
        H[j] = merge(H[S[i]], H[j]); // merge注意标记下传
        S.merge(i, j);
    }}
```

```
  // ++rid; for (int i = 1; i <= n; ++i) if (i != s && S[i] == i)
  //  G[pe[i]->x].pb({pe[i]->y, rid});
  return re;
}
DSU W, S; ai p; pair<Val,int> e[mxn][mxn]; // 初始{MXL,0}
Val chuliu(int s) { // O(V^2), add(x,y,w){minto(e[y][x], {w, y});}
  Val re = 0; W.clear(n); S.clear(n); // int rid = 0;
  for (int i = 1; i <= n; ++i) if (i != s)
    for (;;) {
      #define E(x) e[S[i]][x]
      #define Ew(x) E(x).first // 不求方案时 Val e[][]; 即可
      int x = s;
      for (int j = 1; j <= n; ++j) if (S[j] != S[i])
        if (E(j) < E(x)) x = j;
      if (Ew(x) == MXL) return MXL; // 无解
      re += Ew(x); p[S[i]] = x; // if (x == s) 实根 = E(x).sec;
      for (int j = 1; j <= n; ++j)
        if (j != x && Ew(j) != MXL) Ew(j) -= Ew(x);
      if (W[x] != W[i]) { W.merge(x, i); break; }
      // G[x].pb({E(x).sec, ++rid});
      for (int j = S[x]; j != S[i]; j = S[p[j]]) {
        // G[p[j]].pb({e[j][p[j]].sec, rid});
        for (int k = 1; k <= n; ++k) minto(e[j][k], E(k));
        S.merge(i, j);
    }}
  // ++rid; for (int i = 1; i <= n; ++i) if (i != s && S[i] == i)
  //  G[p[i]].pb({e[i][p[i]].sec, rid});
  return re;
}
vpii °G[mxn]; ai dist, pa; // pa为输出结果,即树形图中点的父亲
void makeSol(int s) { // 用堆优化Prim构造方案
  fill(dist, dist + n + 1, n + 1); pa[s] = 0;
  for (multiset<pii> h = {{0, s}}; !h.empty();) {
    int x = h.begin()->sec; h.erase(h.begin()); dist[x]=0;
    for (auto i : G[x]) if (i.sec < dist[i.fir]) {
      h.erase({dist[i.fir], i.fir});
      h.insert({dist[i.fir] = i.sec, i.fir});
      pa[i.fir] = x; }}}
```

3.11 Blossom Algorithm

U 中一个集合表示一朵压缩花, 树根表示花根; 本算法的 DSU 无需实现 merge.

```
avi e; DSU U; ai mat, c, p, q, v; int *qb, *qe, ts;
int lca(int x, int y) {
  ++ts; for (x = U[x]; x; x = U[p[mat[x]]]) v[x] = ts;
  for (y = U[y]; y; y = U[p[mat[y]]])if(v[y]==ts)return y;
}
void contract(int x, int y, int o) {
  for (; U[x] != o; y = mat[x], x = p[y]) {
    p[x] = y;
    if (c[mat[x]] == 1) c[*qe++ = mat[x]] = 2;
    U.p[x] = U.p[mat[x]] = o; // 不使用DSU::merge()
}}
bool extend(int s) { // O(V^2)
  U.clear(n); fill(c, c + n + 1, 0);
  for (qb = qe = q, c[*qe++ = s] = 2; qb != qe;) {
    int x = *qb++;
    for (auto y : e[x])
      if (!c[y]) {
        if (!mat[y]) {
          for (int t; x; y = t, x = p[y])
            t = mat[x], mat[x] = y, mat[y] = x;
          return true;
        }
        p[y] = x, c[y] = 1, c[*qe++ = mat[y]] = 2;
      } else if (c[y] == 2 && U[x] != U[y]) {
        int o = lca(x, y);
        contract(x, y, o); contract(y, x, o);
  }}
  return false;
}
int blossom() { // O(V^3)
  int re = 0; fill(mat, mat + n + 1, 0);
  for (int i = 1; i <= n; ++i) if (!mat[i]) re+=extend(i);
  return re;
}
```

3.12 Maximum and Maximal Clique

0 下标, 需删除自环 (即确保 $E_{ii} = false$, 补图要特别注意)

```
// 极大团计数, 最坏情况O(3^(n/3)), ull版莫名其妙比bitset版快20%
ll ans; ull E[64]; #define bit(i) (1ULL << (i))
void dfs(ull P, ull X, ull R) { // 不需要方案时可去掉R相关语句
  if (!P && !X) { ++ans; sol.pb(R); return; }
  ull Q = P & ~E[__builtin_ctzll(P | X)];
  for (int i; i = __builtin_ctzll(Q), Q; Q &= ~bit(i)) {
    dfs(P & E[i], X & E[i], R | bit(i));
    P &= ~bit(i), X |= bit(i); }}
ans = 0; dfs(n == 64 ? ~0ULL : bit(n) - 1, 0, 0);
// NTU最大团, n <= 100的稠密图能基本保证2s内出解, 但会被某些数据卡
using bs = bitset<N>;
int ans; bs E[N], sol, R; // 不需要方案时可去掉sol、R相关语句
void dfs(bs P, int cnt) {
  if (P.none()) return;
  if (ans <= cnt) { ans = cnt + 1; sol = R;
    sol.set(P._Find_first()); }
  int t = P.count() + cnt; bs Q = P;
  for (int i = Q._Find_first(); i < Q.size();
      i = Q._Find_next(i)) {
    if (--t < ans) break;
    if ((Q & E[i]).any())
      R.set(i), dfs(P & E[i], cnt + 1), R.reset(i);
    P.reset(i), Q.reset(i);
    if (P == Q) Q &= ~E[i]; }}
bs U; for (int i = 0; i < n; ++i) U.set(i);
ans = 0; sol = R = bs(); dfs(U, 0);
// 随机调整最大团, 对于构造数据实在不行可以试试纯随机不调整
bs E[N], sol; int p[N];
int solve() {
  int re = 0; iota(p, p + n, 0); random_shuffle(p, p + n);
  for (int last=0, tem=1, cnt=0, tm = 500000; --tm; ) {
    int x = rand() % n, y = rand() % n; swap(p[x], p[y]);
    bs v; for (int i = 0; i < n; ++i)
      if ((v & ~E[p[i]]).none()) v.set(p[i]);
    int now = v.count(); if (now > re) re = now, sol = v;
    if (now < last && rand() % (tem + 1) < tem)
      swap(p[x], p[y]), ++tem, cnt = 0;
    else if (last = now, tem > n * 5 || ++cnt > n * 5)
      tem = 1, cnt = 0, random_shuffle(p, p + n);
  } return re; }
// DN超级快最大团, 100个点的速度是NTU的39倍, 建议 n <= 150
typedef bool BB[N]; struct Maxclique {
const BB *e; int pk, level; const float Tlimit;
struct Vertex { int i, d; Vertex(int i) : i(i), d(0) {}};
typedef vector<Vertex> Vertices; Vertices V;
typedef vector<int> ColorClass; ColorClass QMAX, Q;
vector<ColorClass> C;
static bool desc_degree(const Vertex &vi,const Vertex &vj)
{ return vi.d > vj.d; }
void init_colors(Vertices &v) {
  const int max_degree = v[0].d;
  for (int i = 0; i < (int)v.size(); i++)
    v[i].d = min(i, max_degree) + 1; }
void set_degrees(Vertices &v) {
  for (int i = 0, j; i < (int)v.size(); i++)
    for (v[i].d = j = 0; j < (int)v.size(); j++)
      v[i].d += e[v[i].i][v[j].i]; }
struct StepCount{ int i1, i2; StepCount(): i1(0),i2(0){}};
vector<StepCount> S;
bool cut1(const int pi, const ColorClass &A) {
  for (int i = 0; i < (int)A.size(); i++)
    if (e[pi][A[i]]) return true; return false; }
void cut2(const Vertices &A, Vertices & B) {
  for (int i = 0; i < (int)A.size() - 1; i++)
    if (e[A.back().i][A[i].i]) B.push_back(A[i].i); }
void color_sort(Vertices & R) { int j=0, maxno=1;
  int min_k=max((int)QMAX.size()-(int)Q.size()+1,1);
  C[1].clear(), C[2].clear();
  for (int i = 0; i < (int)R.size(); i++) {
    int pi = R[i].i, k = 1; while (cut1(pi, C[k])) k++;
    if (k > maxno) maxno = k, C[maxno + 1].clear();
    C[k].push_back(pi); if (k < min_k) R[j++].i = pi; }
  if (j > 0) R[j - 1].d = 0;
  for (int k = min_k; k <= maxno; k++)
    for (int i = 0; i < (int)C[k].size(); i++)
      R[j].i = C[k][i], R[j++].d = k; }
void expand_dyn(Vertices &R) {
  S[level].i1 = S[level].i1 + S[level-1].i1 - S[level].i2;
  S[level].i2 = S[level - 1].i1;
  while ((int)R.size()) {
    if ((int)Q.size() + R.back().d > (int)QMAX.size()) {
      Q.push_back(R.back().i); Vertices Rp; cut2(R, Rp);
      if ((int)Rp.size()) {
        if((float)S[level].i1/++pk<Tlimit)degree_sort(Rp);
        color_sort(Rp); S[level].i1++, level++;
```

```
        expand_dyn(Rp); level--;
      } else if ((int)Q.size() > (int)QMAX.size()) QMAX=Q;
      Q.pop_back(); } else return; R.pop_back(); }}
void mcqdyn(int *maxclique, int &sz) {
  set_degrees(V); sort(V.begin(), V.end(), desc_degree);
  init_colors(V);
  for (int i=0; i<(int)V.size()+1; i++) S[i].i1=S[i].i2=0;
  expand_dyn(V); sz = (int)QMAX.size();
 for(int i=0;i<(int)QMAX.size();i++)maxclique[i]=QMAX[i];}
void degree_sort(Vertices & R) {
  set_degrees(R); sort(R.begin(), R.end(), desc_degree); }
Maxclique(const BB *conn,const int sz,const float tt=.025)
  : pk(0), level(1), Tlimit(tt){
  for(int i = 0; i < sz; i++) V.push_back(Vertex(i));
  e = conn, C.resize(sz + 1), S.resize(sz + 1); }};
BB e[N]; int ans, sol[N]; for (...) e[x][y]=e[y][x]=true;
Maxclique mc(e, n); mc.mcqdyn(sol, ans); // 全部0下标
for (int i = 0; i < ans; ++i) cout << sol[i] << endl;
```

3.13 Minimum Mean Cycle

点标号为 $1, 2, \cdots, n$, 0 为虚拟源点向其他点连权值为 0 的单向边. 求最大权值环时对边权取反.

```
ll f[N][N] = {Inf}; int u[M], v[M], w[M]; f[0][0] = 0;
for(int i = 1; i <= n + 1; i ++)
  for(int j = 0; j < m; j ++)
    f[i][v[j]] = min(f[i][v[j]], f[i - 1][u[j]] + w[j]);
double ans = Inf;
for(int i = 1; i <= n; i ++) {
  double t = -Inf;
  for(int j = 1; j <= n; j ++)
    t = max(t, (f[n][i] - f[j][i]) / (double)(n - j));
  ans = min(t, ans);
}
```

3.14 Stable Marriage

男性 (x) 向女性 (y) 求婚, 最后每一位男性得到的伴侣都是所有可能的稳定婚姻匹配方案中最理想的, 同时每一位女性得到的伴侣都是所有可能的稳定婚姻匹配方案中最差的. 复杂度 $O(N^2)$

```
// 输入: a[x][x心目中y的排名] = y, rk[y][x] = y心目中x的排名;
// 输出: mat[y] = x;
aai a, rk; ai p, mat; deque<int> q;
void smp() {
  for (int i = 1; i <= n; ++i) q.pb(i), p[i]=1, mat[i]=0;
  while (!q.empty()) {
    int x = q.front(), y; q.pop_front();
    while(mat[y = a[x][p[x]]] && rk[y][mat[y]] < rk[y][x])
      ++p[x];
    if (mat[y]) q.pb(mat[y]);
    mat[y] = x;
}}
```

3.15 Perfect Elimination Ordering

```
vi mcs(int n) {
  static int rank[N], deg[N]; set <pii> st;
  fill(deg, deg + n + 1, 0);
  fill(rank, rank + n + 1, 0);
  for(int i = 1; i <= n; i ++) st.insert({0, i});
  for(int i = n; i >= 1; i --) {
    auto p = *st.rbegin(); st.erase(p);
    rank[p.se] = i;
    for(int i : e[p.se]) if(!rank[i]) {
      st.erase({deg[i], i});
      st.insert({++ deg[i], i});
  }}
  vi ret(n);
  for(int i = 1; i <= n; i ++) ret[rank[i] - 1] = i;
  return ret;
} // 点从1开始标号
```

3.16 Steiner Tree (Saga)

求非负无向图上包含 ts 中所有点的最小 Steiner 树。G 是邻接矩阵。dp[S][v] 表示包含 S 中的点和 v 的最小 Steiner 树。$O(3^t n + 2^t n^2 + n^3)$

```
int dp[1 << MAX_M][MAX_N]; // no memset needed
int steiner(int n, vector<int> ts) {
  int m = ts.size(); if (m < 2) return 0;
  floyd(G);
  for(int i = 0;i < m;i++)
    for(int j = 0;j < n;j++)
      dp[1 << i][j] = G[ts[i]][j];
  for(int i = 1;i < (1 << m);i++) if( ((i-1) & i) != 0 ) {
    for(int j = 0;j < n;j++) {
      dp[i][j] = INF; // 0x3F3F3F3F or something like.
      for(int k = (i-1) & i;k > 0;k = (k-1) & i)
        dp[i][j] = min(dp[i][j], dp[k][j] + dp[i^k][j]);
    }
    for(int j = 0;j < n;j++) for(int k = 0;k < n;k++)
      dp[i][j] = min(dp[i][j], dp[i][k] + G[k][j]);
  }
  return dp[(1<<m) - 1][ts[0]];
}
```

4 String

4.1 Conclusions

4.1.1 双回文串

如果 $s = x_1x_2 = y_1y_2 = z_1z_2, |x_1| < |y_1| < |z_1|, x_2, y_1, y_2, z_1$ 是回文串, 则 x_1 和 z_2 也是回文串。

4.1.2 Border 的结构

字符串 s 的所有不小于 $|s|/2$ 的 border 长度组成一个等差数列。字符串 s 的所有 border 按长度排序后可分成 $O(\log |s|)$ 段，每段是一个等差数列。回文串的回文后缀同时也是它的 border。

4.1.3 子串最小后缀

设 $s[p..n]$ 是 $s[i..n]$, $(l \le i \le r)$ 中最小者, 则 minsuf(l, r) 等于 $s[p..r]$ 的最短非空 border。minsuf(l, r) $= \min\{s[p..r]$, minsuf(r $- 2^k + 1$, r)$\}$, $(2^k < r-l+1 \le 2^{k+1})$。

4.1.4 子串最大后缀

从左往右扫，用 set 维护后缀的字典序递减的单调队列，并在对应时刻添加“小于事件”点以便在之后修改队列；查询直接在 set 里 lower_bound。

4.2 KMP

```
void kmp(char *s, int *a, int n) { // s[1..n] -> a[0..n]
  a[0] = -1;
  int t = -1; // 匹配时用 t = 0;
  for (int i = 1; i <= n; ++i) {
    while (t >= 0 && s[t + 1] != s[i]) t = a[t];
    a[i] = ++t;
  }
}
```

4.3 ExKMP

```
void exkmp(char *s, int *a, int n) {
  a[0] = n; int p = 0, r = 0;
  for (int i = 1; i < n; ++i) {
    a[i] = (r > i) ? min(r - i, a[i - p]) : 0;
    while (i + a[i] < n && s[i + a[i]] == s[a[i]]) ++a[i];
    if (r < i + a[i]) r = i + a[i], p = i;
}}
// 目标串 s: a[i] = lcp(s + i, t); 模式串 t: b[i] = lcp(t + i, t)
void mat(char *s, char *t, int *a, int *b, int n, int m) {
  exkmp(t, b, m); int p = 0, r = 0;
  for (int i = 0; i < n; ++i) {
    a[i] = (r > i) ? min(r - i, b[i - p]) : 0;
    while (i + a[i] < n && a[i] < m
      && s[i + a[i]] == t[a[i]]) ++a[i];
    if (r < i + a[i]) r = i + a[i], p = i;
}}
```

4.4 Manacher

```
// 偶回文扩展: s = "aabbcc", 奇回文扩展: s = "#a#b#c#"
void manacher(char *s, int *a, int n) { // a: 半径(取上整)
  int p = 0, r = 0; // 偶回文对称轴下标取其右边字符的下标
  for (int i = 0; i-1 < n; ++i) { // 奇: i-1 → i
    a[i] = (r > i) ? min(r - i, a[2 * p - i]) : 0;
    while (i + a[i] < n && i-1 - a[i] >= 0 // 奇: i-1 → i
      && s[i + a[i]] == s[i-1 - a[i]]) ++a[i];// 奇: i-1 → i
    if (r < i + a[i]) r = i + a[i], p = i;
}}
```

4.5 Minimum Representation

```
int minrep(char *s, int n) {
  int i = 0, j = 1;
  while (i < n && j < n) {
    int k = 0;
    while (k < n && s[(i + k) % n] == s[(j + k) % n]) ++k;
    if (k >= n) break;
    if (s[(j + k) % n] < s[(i + k) % n]) // 最大表示用 >
      i = max(i + k + 1, j + 1);
    else
      j = max(i + 1, j + k + 1);
  }
  return min(i, j); // 返回起始下标最小解；可通过循环节求所有解
}
```

4.6 Suffix Array

mxn 需大于字符集大小 m, 计算 height 数组时 s 需以 0 结尾

```
void radix(int *s, int *a, int *b, int n, int m) {
  static int c[mxn];
  fill(c, c + m + 1, 0);
  for (int i = n; i--;) ++c[s[a[i]]];
  partial_sum(c, c + m + 1, c);
  for (int i = n; i--;) b[--c[s[a[i]]]] = a[i];
}
void SA(int *s, int *sa, int *h, int n, int m) {
  static int a[mxn * 2], t[mxn];
  fill(a, a + n * 2, 0);
  copy(s, s + n, a);
  for (int len = 1; len / 2 < n; len *= 2) {
    auto b = a + len / 2;
    iota(sa, sa + n, 0);
    radix(b, sa, t, n, m);
    radix(a, t, sa, n, m);
    t[sa[0]] = 1;
    for (int i = 1; i < n; ++i)
      t[sa[i]] = t[sa[i - 1]] + (a[sa[i]] != a[sa[i - 1]]
        || b[sa[i]] != b[sa[i - 1]]);
    copy(t, t + n, a);
    // copy(t, t + n, a + n); // 循环串补丁
    m = t[sa[n - 1]];
    if (m == n) break; // 剪枝
  }
  for (int i = 0; i < n; ++i) --a[i];
  // copy(a, a + n, rk);
  int k = 0;
  for (int i = 0; i < n; ++i) {
    if (k) --k;
    if (a[i]) while (s[i + k] == s[sa[a[i] - 1] + k]) ++k;
    h[a[i]] = k;
  }
}
struct ST { int s[mxn][lgn];
  void build(int *a, int n) {
    for (int i=0;i<n;++i) for (int j = 0; i + 1 >> j; ++j)
      s[i][j] = j?min(s[i][j-1],s[i-(1<<j-1)][j-1]):a[i];}
  int get(int l, int r) {
    int k = 31 - __builtin_clz(r + 1 - l);
    return min(s[r][k], s[l + (1 << k) - 1][k]);
}};
struct LCP { int n, *r; ST Q; // 下标0..n-1, 合并取min的ST表
  void build(int *r, int *h, int n)
  { this->n = n; this->r = r; Q.build(h, n); }
  int get(int x, int y) {
    if (x == y) return n - x;
    return Q.get(min(r[x], r[y]) + 1, max(r[x], r[y])); }
  int get(int rx, int ry) { // 需将build()改为记录*sa指针
    if (rx == ry) return n - sa[rx];
    return Q.get(min(rx, ry) + 1, max(rx, ry)); }};
```

4.7 Aho-Corasick Automaton

```
struct AC { // 0: nil, 1: root
  int e[mxv][mxc], f[mxv], sz; vi que;
  int NEW() { // 请手动清空附加变量
    ++sz, f[sz] = 0; memset(e[sz], 0, sizeof e[sz]);
    return sz; }
  void clear()
  { sz = -1; NEW(); NEW(); fill(e[0], e[0] + mxc, 1); }
  int add(int x, char c)
  { return e[x][c] ? e[x][c] : e[x][c] = NEW(); }
  void build() { // 加边版
    que = {1};
    for (int i = 0; i < que.size(); ++i) {
      int x = que[i];
      for (int c = 0; c < mxc; ++c) if (e[x][c])
        f[e[x][c]] = e[f[x]][c], que.pb(e[x][c]);
        else e[x][c] = e[f[x]][c]; }}
  void build() { // 不加边版
    que = {1};
    for (int i = 0; i < que.size(); ++i) {
      int x = que[i];
      for (int c = 0; c < mxc; ++c) if (e[x][c])
        f[e[x][c]] = to(f[x], c), que.pb(e[x][c]); }}
  int to(int x, char c)
  { while (!e[x][c]) x = f[x]; return e[x][c]; }
};
```

4.8 Palindromic Automaton

```
struct PAM { // 回文串种数 = sz - 1; 0: 偶回文根, 1: 奇回文根
  int e[mxv][mxc], f[mxv], len[mxv], cnt[mxv], sz, n,last;
  char s[mxv]; // s[1..n]
  int NEW(int tlen, int tf) { // 请手动清空附加变量
    ++sz, len[sz] = tlen, f[sz] = tf, cnt[sz] = 0;
    memset(e[sz], 0, sizeof e[sz]);
    return sz;
  }
  void clear()
  { sz = -1; NEW(0, 1); last = NEW(-1, 0); s[n = 0] = -1;}
  int getf(int x) {
    while (s[n - len[x] - 1] != s[n]) x = f[x];
    return x;
  }
  void add(char c) {
    s[++n] = c;
    int x = getf(last), &y = e[x][c];
    if (!y) {
      y = NEW(len[x] + 2, e[getf(f[x])][c]);
      // num[y] = num[f[y]] + 1; // num: 当前串的回文后缀个数
      /* if (len[y] <= 2) hf[y] = f[y]; // hf: 右半串的最长回文后缀
       * else { int t = getf(hf[x]); while (len[t] * 2 + 4 > len[y])
       *   t = getf(f[t]); hf[y] = e[t][c]; } */
    }
    last = y; ++cnt[last];
  }
  void count() { // cnt: 每种串出现的次数
    for (int i = sz; i >= 2; --i) cnt[f[i]] += cnt[i];
    cnt[0] = cnt[1] = 0;
  }
};
```

4.9 Suffix Automaton

对 Trie 建 SAM 时请使用 BFS 序添加字符，边数及复杂度均为 $O(N * \Sigma)$。
大字符集 Trie 可将字符拆成 01 串，注意实状态、实点、实 fail 的求法。

```
struct SAM { // 注意mxv要开到mxn * 2; 0 : nil, 1: root
  int e[mxv][mxc], f[mxv], len[mxv], sz;
  int NEW(int tlen) { // 请手动清空附加变量
    ++sz, f[sz] = 0, len[sz] = tlen;
    memset(e[sz], 0, sizeof e[sz]);
    return sz;
  }
  void clear() { sz = -1; NEW(-1); NEW(0);
    fill(e[0], e[0] + mxc, 1); } // 这样可省略0->1的特判
  int add(int x, char c) { // 支持多串
    int y = e[x][c] ? 0 : NEW(len[x] + 1);
    while (x && !e[x][c]) e[x][c] = y, x = f[x];
    if (!x) f[y] = 1;
    else {
      int a = e[x][c];
      if (len[x] + 1 == len[a]) f[y] = a;
      else {
        int b = NEW(len[x] + 1);
        copy(e[a], e[a] + mxc, e[b]);
        f[b] = f[a], f[a] = f[y] = b;
        while (x && e[x][c] == a) e[x][c] = b, x = f[x];
    }}
    return y ? y : f[y]; // f[0]作为临时变量
  }
  vi que;
  void sort() { // 拓扑序: 按len[]基数排序
    static int c[mxv];
```

```
    fill(c, c + sz + 1, 0);
    for (int i = sz; i; --i) ++c[len[i]];
    partial_sum(c, c + sz + 1, c);
    que.resize(sz);
    for (int i = sz; i; --i) que[--c[len[i]]] = i;
  }
  pii to(int x, int l, char c) {
    while (!e[x][c]) l = len[x = f[x]];
    return {e[x][c], l + 1};
  }
};
```

4.10 Lyndon Word

满足 s 的最小后缀等于 s 本身的串 s 称为 Lyndon 串。等价于：s 是它自己的所有循环移位中唯一最小的一个。任意字符串 s 可以分解为 $s = s_1 s_2 \ldots s_k$, 其中 s_i 是 Lyndon 串, $s_i \geq s_{i+1}$。且这种分解方法是唯一的。

```
void mnsuf(char *s, int *mn, int n) { // 每个前缀的最小后缀
  for (int i = 0; i < n; ) {
    int j = i, k = i + 1; mn[i] = i;
    for (; k < n && s[j] <= s[k]; ++k)
      if (s[j] == s[k]) mn[k] = mn[j] + k - j, ++j;
      else mn[k] = j = i;
    for (; i <= j; i += k - j) {} // lyn+=s[i .. i + k - j - 1]
}}
void mxsuf(char *s, int *mx, int n) { // 每个前缀的最大后缀
  fill(mx, mx + n, -1);
  for (int i = 0; i < n; ) {
    int j = i, k = i + 1; if (mx[i] == -1) mx[i] = i;
    for (; k < n && s[j] >= s[k]; ++k) {
      j = s[j] == s[k] ? j + 1 : i;
      if (mx[k] == -1) mx[k] = i;
    }
    for (; i <= j; i += k - j) {}
}}
```

4.11 Palindomic Factorization

```
// pl[i][0] : s[0..i-1]的最小偶分解, pl[i][1] : 最小奇分解
namespace zimpha { // O(NlgN), s[0..n-1] -> pl[1..n]
const int MXL = 1E9; int pl[mxn][2], gpl[mxn][2];
inline void set(int *a, int x, int y, int z) {
  a[0] = x, a[1] = y, a[2] = z; }
inline void set(int *a, int *b) {
  a[0] = b[0], a[1] = b[1], a[2] = b[2]; }
inline void set(int pl[][2], int x, int val) {
  if (val <= 0) return; pl[x][val & 1] = val; }
inline void upd(int pl[][2], int x, int val) {
  if (val <= 0) return; int &r = pl[x][val & 1];
  if (r == -1 || r > val) r = val; }
void factor(char *s, int n) { // 无解: MXL 或 MXL + 1
  for (int i = 0; i <= n; ++i)
    gpl[i][0] = MXL, gpl[i][1] = MXL + 1;
  static int g[32][3], gp[32][3], gpp[32][3]; int pg = 0;
  for (int j = 0; j < n; ++j) { int pgp = 0;
    for (int u = 0; u < pg; ++u) {
      int i = g[u][0]; if (i - 1 >= 0 && s[i - 1] == s[j])
        g[u][0]--, set(gp[pgp++], g[u]); }
    int pgpp = 0, r = -j - 2;
    for (int u = 0; u < pgp; ++u) {
      int i = gp[u][0], d = gp[u][1], k = gp[u][2];
      if (i - r != d) { set(gpp[pgpp++], i, i - r, 1);
        if (k > 1) set(gpp[pgpp++], i + d, d, k - 1); }
      else set(gpp[pgpp++], i, d, k);
      r = i + (k - 1) * d; }
    if (j - 1 >= 0 && s[j - 1] == s[j])
      set(gpp[pgpp++], j - 1, j - 1 - r, 1), r = j - 1;
    set(gpp[pgpp++], j, j - r, 1);
    pg = 0; int *front = gpp[0];
    for (int u = 1; u < pgpp; ++u) { int *x = gpp[u];
      if (x[1] == front[1]) front[2] += x[2];
      else set(g[pg++], front), front = x; }
    set(g[pg++], front);
    if ((j + 1) % 2 == 0)
      pl[j + 1][0] = j + 1, pl[j + 1][1] = MXL + 1;
    else pl[j + 1][0] = MXL, pl[j + 1][1] = j + 1;
    for (int u = 0; u < pg; ++u) {
      int i = g[u][0], d = g[u][1], k = g[u][2];
      r = i + (k - 1) * d;
      upd(pl, j + 1, pl[r][0] + 1);
      upd(pl, j + 1, pl[r][1] + 1);
      if (k > 1) { upd(pl, j + 1, gpl[i + 1 - d][0]);
        upd(pl, j + 1, gpl[i + 1 - d][1]); }
    if (i + 1 >= d) {
      if (k > 1) { upd(gpl, i + 1 - d, pl[r][0] + 1);
        upd(gpl, i + 1 - d, pl[r][1] + 1); }
      else { set(gpl, i + 1 - d, pl[r][0] + 1);
        set(gpl, i + 1 - d, pl[r][1] + 1); }}}}}
```

5 Geometry

5.1 Formula (Saga)

5.1.1 三角形内心

$$\frac{a\vec{A} + b\vec{B} + c\vec{C}}{a + b + c}$$

5.1.2 三角形外心

$$\frac{\vec{A} + \vec{B} - \frac{\overrightarrow{BC}\cdot\overrightarrow{CA}}{\overrightarrow{AB}\times\overrightarrow{BC}}\overrightarrow{AB}^T}{2}$$

5.1.3 三角形垂心

$$\vec{H} = 3\vec{G} - 2\vec{O}$$

5.1.4 三角形偏心

$$\frac{-a\vec{A} + b\vec{B} + c\vec{C}}{-a + b + c}$$

剩余两点的同理。

5.1.5 三角形内接外接圆半径

$$r = \frac{2S}{a + b + c},\ R = \frac{abc}{4S}$$

5.1.6 Pick's Theorem

$$S = I + \frac{B}{2} - 1$$

5.1.7 Euler's Formula

For convex polyhedron: $V - E + F = 2$.
For planar graph: $|F| = |E| - |V| + n + 1$, n denotes the number of connected components. S is the area of lattice polygon, I is the number of lattice interior points, and B is the number of lattice boundary points.

5.1.8 Heron's Formula

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$
$$p = \frac{a + b + c}{2}$$

5.1.9 超球坐标系

$$\begin{aligned}
x_1 &= r\cos(\phi_1)\\
x_2 &= r\sin(\phi_1)\cos(\phi_2)\\
x_3 &= r\sin(\phi_1)\sin(\phi_2)\cos(\phi_3)\\
&\cdots\\
x_{n-1} &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\cos(\phi_{n-1})\\
x_n &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\sin(\phi_{n-1})\\
\phi_{n-1} &= 0..2*\pi\\
\forall i = 1..n-1 \phi_i &= 0..\pi
\end{aligned}$$

5.1.10 三维旋转公式

绕着 $(0,0,0) - (ux, uy, uz)$ 旋转 θ, (ux, uy, uz) 是单位向量

$$R = \begin{matrix}
\cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta\\
u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta\\
u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta)
\end{matrix}.$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

5.1.11 立体角公式

ϕ : 二面角

$\Omega = (\phi_{ab} + \phi_{bc} + \phi_{ac})\ \text{rad} - \pi\ \text{sr}$

$$\tan\left(\frac{1}{2}\Omega/\text{rad}\right) = \frac{\left|\vec{a}\ \vec{b}\ \vec{c}\right|}{abc + \left(\vec{a}\cdot\vec{b}\right)c + \left(\vec{a}\cdot\vec{c}\right)b + \left(\vec{b}\cdot\vec{c}\right)a}$$

$$\theta_s = \frac{\theta_a + \theta_b + \theta_c}{2}$$

5.1.12 常用体积公式

- Pyramid $V = \frac{1}{3}Sh$.
- Sphere $V = \frac{4}{3}\pi R^3$.
- Frustum $V = \frac{1}{3}h(S_1 + \sqrt{S_1 S_2} + S_2)$.
- Ellipsoid $V = \frac{4}{3}\pi abc$.

5.1.13 高维球体积

$$V_2 = \pi R^2,\ S_2 = 2\pi R$$
$$V_3 = \frac{4}{3}\pi R^3,\ S_3 = 4\pi R^2$$
$$V_4 = \frac{1}{2}\pi^2 R^4,\ S_4 = 2\pi^2 R^3$$
$$V_5 = \frac{8}{15}\pi^2 R^5,\ S_5 = \frac{8}{3}\pi^2 R^4$$
$$V_6 = \frac{1}{6}\pi^3 R^6,\ S_6 = \pi^3 R^5$$
$$\text{Generally, } V_n = \frac{2\pi}{n}V_{n-2},\ S_{n-1} = \frac{2\pi}{n-2}S_{n-3}$$
$$\text{Where, } S_0 = 2,\ V_1 = 2,\ S_1 = 2\pi,\ V_2 = \pi$$

5.2 2D Geometry

```
struct Point {
  Point rot(double a) {
    return {x * cos(a) - y * sin(a),
      x * sin(a) + y * cos(a)};
  }
  Point proj(Point p, Point q) {
    double s = (*this - p).dot(q - p) / (q - p).len2();
    return p + (q - p) * s;
  }
  bool inner(poi a, poi b, poi c) {
    if((c - b).cross(a - b) > 0)
      return cross(c - b) <= 0 && cross(a - b) >= 0;
    return cross(c - b) <= 0 || cross(a - b) >= 0;
  } // 正方向多边形中从b出发的射线是否在多边形中
  bool collide(poi a, poi b, poi c) {
    return (b - a).cross(c - b) > 0 ||
      cross(b - a) * cross(c - b) > 0;
  } // 正方向多边形中过b点的直线是否需计算和b相交
  bool tan(Point o, double r, Point &p, Point &q) {
    double x = (*this - o).len2(), d = x - r * r;
    if(sgn(d) <= 0) return false;
    Point t = (*this - o) * (r * r / x);
    Point s = ((*this - o) * (-r * sqrt(d)/x)).rot90();
    p = o + t + s; q = o + t - s; return true;
}};
int ccw(poi a, poi b, poi c) {
  poi u = b - a, v = c - a;
  if(u.cross(v) > eps) return 1;
  if(u.cross(v) < -eps) return -1;
  if(u.dot(v) < -eps) return -2;
  if(u.len2() + eps < v.len2()) return 2;
  return 0;
}
bool si(poi a, poi b, poi c, poi d) {
  return ccw(a, b, c) * ccw(a, b, d) <= 0 &&
    ccw(c, d, a) * ccw(c, d, b) <= 0;
}
poi li(poi a, poi b, poi c, poi d) {
  db u = xmul(c, d, a), v = xmul(c, d, b);
  return (a * v - b * u) / (v - u);
}
db li(poi a, poi b, poi c, poi d) {
  if(xmul(a, b, c) * xmul(a, b, d) >= 0) return Inf;
  return (c - a).cross(d - c) / (b - a).cross(d - c);
} // cd是否严格在ab两侧，并返回(交点-a)相对ba的长度

vp cc(Point c1, double r1, Point c2, double r2) {
  vp ret; double x = (c1 - c2).len2();
  double y = ((r1 * r1 - r2 * r2) / x + 1) / 2;
  double d = r1 * r1 / x - y * y;
  if(sgn(d) == -1) return ret; d = max(d, 0.0);
  Point p1 = c1 + (c2 - c1) * y;
  Point p2 = ((c2 - c1) * sqrt(d)).rot(Pi / 2);
  ret.push_back(p1 - p2); ret.push_back(p1 + p2);
  return ret;
} // c1 != c2, 点按c1正方向返回，内切 / 外切交弧需额外判断
double cc_area(Point c1, double r1, Point c2, double r2) {
  double d = (c1 - c2).length();
  if(sgn(r1 + r2, d) <= 0) return 0;
  if(sgn(d, fabs(r1 - r2)) <= 0)
    return min(r1, r2) * min(r1, r2) * Pi;
  double ret = 0;
  for(int t = 0; t < 2; t ++) {
    double p = (r1 * r1 + d * d - r2 * r2) / (2 * r1 * d);
    double da = acos(p) * 2;
    ret += r1 * r1 * (da - sin(da)) / 2; swap(r1, r2);
  }
  return ret;
}
pdd cc(Point c1, double r1, Point c2, double r2, bool &f){
  f = false; double d = (c1 - c2).len();
  if(sgn(r1 + r2, d) < 0 || sgn(r2 + d, r1) < 0)
    return {0, 0}; f = true;
  if(sgn(r1 + d, r2) <= 0) return {0, 2 * Pi};
  vp p = ci(c1, r1, c2, r2);
  double a1 = (p[0] - c1).arg(), a2 = (p[1] - c1).arg();
  if(sgn(a1, a2) > 0) a2 += 2 * Pi; return {a1, a2};
} // 返回圆c1的角度区间
vp cl(Point c, double r, Point p, Point q) { vp ret;
  double x = (p - c).dot(q - p), y = (q - p).length2();
  double d = x * x - y * ((p - c).length2() - r * r);
  if(sgn(d) == -1) return ret; d = max(d, 0.0);
  Point p1 = p - ((q - p) * (x / y));
  Point p2 = (q - p) * (sqrt(d) / y);
  ret.push_back(p1 - p2); ret.push_back(p1 + p2);
  return ret;
}
vppp intan(Point o1, double r1, Point o2, double r2) {
  vppp ret; Point p1, p2, q1, q2;
  Point p = (o1 * r2 + o2 * r1) / (r1 + r2);
  if(p.tan(o1, r1, p1, p2) && p.tan(o2, r2, q1, q2)) {
    ret.pb({p2, q2}); ret.pb({p1, q1});
  }
  return ret;
} // 按c1正方向返回，两圆相切认为没有切线
vppp extan(Point o1, double r1, Point o2, double r2) {
  vppp ret; Point p1, p2, q1, q2;
  if(sgn(r1 - r2) == 0) {
    Point dir = o2 - o1;
    dir = (dir * (r1 / dir.len())).rot(Pi / 2);
    ret.pb({o1 - dir, o2 - dir});
    ret.pb({o1 + dir, o2 + dir});
  } else {
    Point p = (o1 * -r2 + o2 * r1) / (r1 - r2);
    if(p.tanCP(o1, r1, p1, p2) && p.tan(o2, r2, q1, q2)) {
      if(r1 < r2) { swap(p1, p2); swap(q1, q2); }
      ret.pb({p2, q2}); ret.pb({p1, q1});
  }}
  return ret;
} // 按c1正方向返回
bool contain(vp poly, Point p) {
  int ret = 0, n = poly.size();
  for(int i = 0; i < n; ++ i) {
    Point u = poly[i], v = poly[(i + 1) % n];
    if(p.on_seg(u, v)) return true;
    if(sgn(u.y, v.y) <= 0) swap(u, v);
    if(sgn(p.y, u.y) > 0 || sgn(p.y, v.y) <= 0) continue;
    ret += sgn(xmul(p, v, u)) > 0;
  }
  return ret & 1;
}
vp cp_cut(vp cp, Point p, Point q) {
  vp ret; int n = cp.size();
  for(int i = 0; i < n; ++i) {
    Point s = cp[i], t = cp[(i + 1) % n];
    int d1 = sgn(xmul(p, q, s)), d2 = sgn(xmul(p, q, t));
    if(d1 >= 0) ret.pb(s);
    if(d1 * d2 < 0) ret.pb(li(s, t, p, q));
  }
  return ret;
}
double ct_area(Point p, Point q, double r) {
  if(p.len() < q.len()) swap(p, q);
  if(sgn(q.len()) == 0) return 0;
  double a = q.len(), b = p.len(), c = (q - p).len();
  double sb = fabs(q.cross(q - p) / a / c);
  double cb = q.dot(q - p) / a / c;
  double sc = fabs(p.cross(q) / a / b);
  double cc = p.dot(q) / a / b;
  double B = atan2(sb, cb), C = atan2(sc, cc), S = 0;
```

```
  if(a > r) {
    S = C / 2 * r * r;
    double h = a * b * sc / c;
    if(h < r && B < Pi / 2)
      S -= (acos(h/r) * r * r - h * sqrt(r * r - h * h));
  } else if(b > r) {
    double theta = Pi - B - asin(sb / r * a);
    S = a * r * sin(theta) / 2 + (C - theta) / 2 * r * r;
  } else S = sc * a * b / 2; return S;
}
vp convex(vp v) {
  sort(v.begin(), v.end()); vp r;
  for(int i = 0; i < v.size(); i ++) {
    while(r.size() > 1 &&
      sgn(xmul(*++r.rbegin(), v[i], *r.rbegin())) >= 0)
      r.pop_back();
    r.pb(v[i]);
  }
  int d = r.size();
  for(int i = (int)v.size() - 2; i >= 0; i --) {
    while(r.size() > d &&
      sgn(xmul(*++r.rbegin(), v[i], *r.rbegin())) >= 0)
      r.pop_back();
    if(i != 0) r.pb(v[i]);
  }
  return r;
} // 保留共线点时>=改为>, 且需保证点集无重点
Poi in_center(Poi a, Poi b, Poi c) {
  double d = (b - c).len(),
    e = (c - a).len(), f = (a - b).len();
  return (a * d + b * e + c * f) / (d + e + f);
} // 需保证三角形不退化, 下同
Poi circum_center(Poi a, Poi b, Poi c) {
  Poi ba = b - a, ca = c - a;
  double lb = ba.len2(), lc = ca.len2(), s = ba.cross(ca);
  return a - Poi{ba.y * lc - ca.y * lb,
    ca.x * lb - ba.x * lc} / s / 2;
} // h = a + b + c - 2o
```

5.3 Nearest Pair

```
db solve(poi *p, int l, int r) {
  if(l + 1 == r) return Inf;
  int m = (l + r) / 2; db mx = p[m].x; vector <poi> v;
  db ret = min(solve(p, l, m), solve(p, m, r));
  for(int i = l; i < r; i ++)
    if(poi{p[i].x - mx, 0}.len2() < ret) v.pb(p[i]);
  sort(v.begin(), v.end(), by_y);
  for(int i = 0; i < v.size(); i ++)
    for(int j = i + 1; j < v.size(); j ++) {
      if(poi{0, v[i].y - v[j].y}.len2() > ret) break;
      ret = min(ret, (v[i] - v[j]).len2());
    }
  return ret;
} // 需先对p[]进行排序
```

5.4 Half-plane Intersection

```
struct Seg {
  Point s, e; double a; Seg() {}
  Seg(Point s, Point e) : s(s), e(e) {
    a = atan2(e.y - s.y, e.x - s.x);
}};
Point inter(Seg s1, Seg s2) {
  double u = xmul(s1.s, s1.e, s2.s);
  double v = xmul(s1.e, s1.s, s2.e);
  double tx = (s2.s.x * v + s2.e.x * u) / (u + v);
  double ty = (s2.s.y * v + s2.e.y * u) / (u + v);
  return {tx, ty};
}
bool cmp(Seg a, Seg b) {
  if(sgn(a.a - b.a) == 0)
    return sgn(xmul(a.s, a.e, b.s)) < 0;
  return sgn(a.a - b.a) < 0;
}
bool par(Seg p, Seg q) {
  return sgn((p.e - p.s).cross(q.e - q.s)) == 0;
}
Seg q[N], seg[N]; Point hull[N];
double hp_inter(int n) {
  sort(seg, seg + n, cmp); int tmp = 1;
  for(int i = 1; i < n; i ++)
    if(sgn(seg[i].a - seg[tmp - 1].a) != 0)
      seg[tmp ++] = seg[i];
  n = tmp; q[0] = seg[0]; q[1] = seg[1];
  int hd = 0, tl = 1;
  for(int i = 2; i < n; i ++) {
    if(par(q[tl], q[tl - 1]) || par(q[hd], q[hd + 1]))
      return 0;
    while(hd < tl && xmul(seg[i].s, seg[i].e,
      inter(q[tl], q[tl - 1])) < -Eps) -- tl;
    while(hd < tl && xmul(seg[i].s, seg[i].e,
      inter(q[hd], q[hd + 1])) < -Eps) ++ hd;
    q[++ tl] = seg[i];
  }
  while(hd < tl && xmul(q[hd].s, q[hd].e,
    inter(q[tl], q[tl - 1])) < -Eps) tl --;
  while(hd < tl && xmul(q[tl].s, q[tl].e,
    inter(q[hd], q[hd + 1])) < -Eps) hd ++;
  int cnt = 0; q[++ tl] = q[hd];
  for(int i = hd; i < tl; i ++)
    hull[cnt ++] = inter(q[i], q[i + 1]);
  double ans = 0;
  for(int i = 1; i < cnt - 1; i ++)
    ans += fabs((hull[i] - hull[0]).
      cross(hull[i + 1] - hull[0]));
  return ans / 2;
}
```

5.5 Circle Union

```
double area[N]; Cir c[N];
struct Eve { Poi p; double a; int d;
  bool operator <(Eve e)const{ return sgn(a, e.a) < 0; }};
void add(Cir a, Cir b, vector <Eve> &eves, int &cnt) {
  double d2 = (a.o - b.o).len2(),
    dr = ((sqr(a.r) - sqr(b.r)) / d2 + 1) / 2,
    pr = sqrt(-(d2 - sqr(a.r - b.r)) *
      (d2 - sqr(a.r + b.r)) / (d2 * d2 * 4));
  Poi d = b.o - a.o, p = d.rot90(),
    q0 = a.o + d * dr + p * pr,q1 = a.o + d * dr - p * pr;
  double a0 = (q0 - a.o).ang(), a1 = (q1 - a.o).ang();
  eves.pb({q1, a1, 1}); eves.pb({q0, a0, -1}); cnt+=a1>a0;
}
bool same(Cir a, Cir b) {
  return !sgn((a.o - b.o).len()) && !sgn(a.r - b.r); }
bool overlap(Cir a, Cir b) {
  return sgn(a.r - b.r - (a.o - b.o).len()) >= 0; }
bool intersect(Cir a, Cir b) {
  return sgn((a.o - b.o).len() - a.r - b.r) < 0; }
void solve(int n) {
  fill(area, area + n + 1, 0);
  for(int i = 0; i < n; i ++) {
    int cnt = 1; vector <Eve> eves;
    for(int j = 0; j < i; j ++) cnt += same(c[i], c[j]);
    for(int j = 0; j < n; j ++) cnt += (j != i &&
      !same(c[i], c[j]) && overlap(c[j], c[i]));
    for(int j = 0; j < n; j ++) if(j != i &&
      !overlap(c[j], c[i]) && !overlap(c[i], c[j]) &&
      intersect(c[i], c[j])) add(c[i], c[j], eves, cnt);
    if(!eves.size()) area[cnt] += Pi * c[i].r * c[i].r;
    else {sort(eves.begin(),eves.end()); eves.pb(eves[0]);
      for(int j = 0; j + 1 < eves.size(); j ++) {
        cnt += eves[j].d;
        area[cnt] += eves[j].p.cross(eves[j + 1].p) / 2;
        double a = eves[j + 1].a - eves[j].a;
        if(a < 0) a += Pi * 2;
        area[cnt] += (a - sin(a)) * c[i].r * c[i].r / 2;
  }}}
  for(int i = 1; i < n; i ++) area[i] -= area[i + 1];
}
```

5.6 Minkowski Sum

```
vp minkowski(vp u, vp v) {
  u = conv(u); v = conv(v); vp r = {u[0] + v[0]};
  int n = u.size(), m = v.size(); u.pb(u[0]); v.pb(v[0]);
  for(int i = 0, j = 0; i < n || j < m; ) if(j == m ||
      sgn((u[i + 1] - u[i]).cross(v[j + 1] - v[j])) > 0)
      r.pb(r.back() + u[i + 1] - u[i]), i ++;
    else r.pb(r.back() + v[j + 1] - v[j]), j ++;
  r.pop_back(); return r;
} // 返回凸包可能有共线点
```

5.7 3D Geometry

绕单位向量旋转

$$\begin{bmatrix} x^2+(1-x^2)\cos\theta & xy(1-\cos\theta)-z\sin\theta & xz(1-\cos\theta)+y\sin\theta \\ xy(1-\cos\theta)+z\sin\theta & y^2+(1-y^2)\cos\theta & yz(1-\cos\theta)-x\sin\theta \\ xz(1-\cos\theta)-y\sin\theta & yz(1-\cos\theta)+x\sin\theta & z^2+(1-z^2)\cos\theta \end{bmatrix}$$

```
poi inter(poi A, poi B, poi C, poi D) {
  poi p0 = (C - A).cross(D - C),p1 = (B - A).cross(D - C);
  return A + (B - A) * (p0.dot(p1) / p1.len2());
} // 异面时返回共垂线在AB上的垂足
```

5.8 Mininal Ball

```
double det(double m[3][3]) {
  double *t = &m[0][0];
  double a = t[0] * t[4] * t[8], b = t[0] * t[5] * t[7];
  double c = t[1] * t[5] * t[6], d = t[1] * t[3] * t[8];
  double e = t[2] * t[3] * t[7], f = t[2] * t[4] * t[6];
  return a - b + c - d + e - f;
}
void ball() {
  Point q[3]; double m[3][3], sol[3], l[3], d;
  res = {0, 0, 0}; r = 0;
  switch(cnt) {
    case 1: res = o[0]; break;
    case 2: res = (o[0] + o[1]) / 2;
      r = (res - o[0]).len2(); break;
    case 3: for(int i = 0; i < 2; i ++)
        q[i] = o[i + 1] - o[0];
      for(int i = 0; i < 2; i ++)
        for(int j = 0; j < 2; j ++)
          m[i][j] = q[i].dot(q[j]) * 2;
      for(int i = 0; i < 2; i ++) sol[i] = q[i].len2();
      d = m[0][0] * m[1][1] - m[0][1] * m[1][0];
      if(sgn(d) == 0) return;
      l[0] = (sol[0] * m[1][1] - sol[1] * m[0][1]) / d;
      l[1] = (sol[1] * m[0][0] - sol[0] * m[1][0]) / d;
      res = o[0] + q[0] * l[0] + q[1] * l[1];
      r = (res - o[0]).len2(); break;
    case 4: for(int i = 0; i < 3; i ++) {
        q[i] = o[i + 1] - o[0]; sol[i] = q[i].len2();
      }
      for(int i = 0; i < 3; i ++)
        for(int j = 0; j < 3; j ++)
          m[i][j] = q[i].dot(q[j]) * 2;
      d = det(m); if(sgn(d) == 0) return;
      for(int j = 0; j < 3; j ++) {
        for(int i = 0; i < 3; i ++) m[i][j] = sol[i];
        l[j] = det(m) / d;
        for(int i = 0; i < 3; i ++)
          m[i][j] = q[i].dot(q[j]) * 2;
      }
      res = o[0];
      for(int i = 0; i < 3; i ++)
        res = res + q[i] * l[i];
      r = (res - o[0]).len2();
}}
void mb(int m) { ball();
  if(cnt < 4) for(int i = 0; i < m; i ++)
    if(sgn((res - p[i]).len2(), r) > 0) {
      o[cnt ++] = p[i]; mb(i); cnt --;
      if(i > 0) { Point t = p[i];
        memmove(&p[1], &p[0], sizeof(Point) * i);
        p[0] = t;
}}}
double solve() { r = -1;
  for(int i = 0; i < n; i ++)
    if(sgn((res - p[i]).len2(), r) > 0) {
      cnt = 1; o[0] = p[i]; mb(i);
    }
  return sqrt(r);
}
```

5.9 3D Convex Hull

```
using i3 = array<int, 3>; using vi3 = vector<i3>;
namespace Convex { int n, ts, m[N][N]; vi3 v;
  void add(int x) { vi3 t; ts ++;
    for(auto f : v) if(sgn(vol(x, f[0], f[1], f[2])) < 0)
        for(int i : f) for(int j : f) m[i][j] = ts;
      else t.pb(f); v = t;
    for(auto f : t) {
      int a = f[0], b = f[1], c = f[2];
      if(m[a][b] == ts) v.pb({b, a, x});
      if(m[b][c] == ts) v.pb({c, b, x});
      if(m[c][a] == ts) v.pb({a, c, x});
  }}
  bool find() {
    for(int i = 2; i < n; i ++) {
      Point d = (p[0] - p[i]).cross(p[1] - p[i]);
      if(sgn(d.len()) == 0) continue; swap(p[i], p[2]);
      for(int j = i + 1; j < n; j ++)
        if(sgn(vol(0, 1, 2, j)) != 0) { swap(p[i], p[3]);
          v.pb({0, 1, 2}); v.pb({0, 2, 1}); return true;
    }}
    return false;
  }
  vi3 convex(int n) {
    sort(p, p + n); n = unique(p, p + n) - p;
    v.clear(); random_shuffle(p, p + n); Convex :: n = n;
    if(find()) { memset(m, 0, sizeof(m)); ts = 0;
      for(int i = 3; i < n; i ++) add(i);
    }
    return v;
}}
```

6 Math

6.1 Formula

6.1.1 Arithmetic Function

$$\sigma_k(n) = \sum_{d|n} d^k = \prod_{i=1}^{\omega(n)} \frac{p_i^{(a_i+1)k} - 1}{p_i^k - 1}$$

$$J_k(n) = n^k \prod_{p|n} (1 - \frac{1}{p^k})$$

$J_k(n)$ is the number of k-tuples of positive integers all less than or equal to n that form a coprime $(k+1)$-tuple together with n.

$$\sum_{\delta|n} J_k(\delta) = n^k$$

$$\sum_{\delta|n} \delta^s J_r(\delta) J_s(\frac{n}{\delta}) = J_{r+s}(n)$$

$$\sum_{\delta|n} \varphi(\delta) d(\frac{n}{\delta}) = \sigma(n)$$

$$\sum_{\delta|n} |\mu(\delta)| = 2^{\omega(n)}$$

$$\sum_{\delta|n} 2^{\omega(\delta)} = d(n^2)$$

$$\sum_{\delta|n} d(\delta^2) = d^2(n)$$

$$\sum_{\delta|n} d(\frac{n}{\delta}) 2^{\omega(\delta)} = d^2(n)$$

$$\sum_{\delta|n} \frac{\mu(\delta)}{\delta} = \frac{\varphi(n)}{n}$$

$$\sum_{\delta|n} \frac{\mu(\delta)}{\varphi(\delta)} = d(n)$$

$$\sum_{\delta|n} \frac{\mu^2(\delta)}{\varphi(\delta)} = \frac{n}{\varphi(n)}$$

$$n | \varphi(a^n - 1)$$

$$\sum_{\substack{1 \le k \le n \\ \gcd(k,n)=1}} f(\gcd(k-1, n)) = \varphi(n) \sum_{d|n} \frac{(\mu * f)(d)}{\varphi(d)}$$

$$\varphi(\mathrm{lcm}(m,n)) \varphi(\gcd(m,n)) = \varphi(m)\varphi(n)$$

$$\sum_{\delta|n} d^3(\delta) = (\sum_{\delta|n} d(\delta))^2$$

$$d(uv) = \sum_{\delta | \gcd(u,v)} \mu(\delta) d(\frac{u}{\delta}) d(\frac{v}{\delta})$$

$$\sigma_k(u)\sigma_k(v) = \sum_{\delta | \gcd(u,v)} \delta^k \sigma_k(\frac{uv}{\delta^2})$$

$$\mu(n) = \sum_{k=1}^{n} [\gcd(k,n) = 1] \cos 2\pi \frac{k}{n}$$

$$\varphi(n)=\sum_{k=1}^{n}[\gcd(k,n)=1]=\sum_{k=1}^{n}\gcd(k,n)\cos 2\pi\frac{k}{n}$$

$$\begin{cases}S(n)=\sum_{k=1}^{n}(f*g)(k)\\ \sum_{k=1}^{n}S(\lfloor\frac{n}{k}\rfloor)=\sum_{i=1}^{n}f(i)\sum_{j=1}^{\lfloor\frac{n}{i}\rfloor}(g*1)(j)\end{cases}$$

$$\begin{cases}S(n)=\sum_{k=1}^{n}(f\cdot g)(k),g\text{ completely multiplicative}\\ \sum_{k=1}^{n}S(\lfloor\frac{n}{k}\rfloor)g(k)=\sum_{k=1}^{n}(f*1)(k)g(k)\end{cases}$$

6.1.2 Binomial Coefficients

$$\binom{n}{k}=(-1)^k\binom{k-n-1}{k}$$

$$\sum_{k\le n}\binom{r+k}{k}=\binom{r+n+1}{n}$$

$$\sum_{k=0}^{n}\binom{k}{m}=\binom{n+1}{m+1}$$

$$\sqrt{1+z}=1+\sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{k\times 2^{2k-1}}\binom{2k-2}{k-1}z^k$$

$$\sum_{k=0}^{r}\binom{r-k}{m}\binom{s+k}{n}=\binom{r+s+1}{m+n+1}$$

$$C_{n,m}=\binom{n+m}{m}-\binom{n+m}{m-1},n\ge m$$

$$\binom{n}{k}\equiv[n\&k=k]\pmod 2$$

6.1.3 Fibonacci Numbers

$$F(z)=\frac{z}{1-z-z^2}$$

$$f_n=\frac{\phi^n-\hat{\phi}^n}{\sqrt{5}},\phi=\frac{1+\sqrt{5}}{2},\hat{\phi}=\frac{1-\sqrt{5}}{2}$$

$$\sum_{k=1}^{n}f_k=f_{n+2}-1$$

$$\sum_{k=1}^{n}f_k^2=f_nf_{n+1}$$

$$\sum_{k=0}^{n}f_kf_{n-k}=\frac{1}{5}(n-1)f_n+\frac{2}{5}nf_{n-1}$$

$$f_n^2+(-1)^n=f_{n+1}f_{n-1}$$

$$f_{n+k}=f_nf_{k+1}+f_{n-1}f_k$$

$$f_{2n+1}=f_n^2+f_{n+1}^2$$

$$(-1)^kf_{n-k}=f_nf_{k-1}-f_{n-1}f_k$$

$$\text{Modulo }f_n,f_{mn+r}\equiv\begin{cases}f_r, & m\bmod 4=0;\\ (-1)^{r+1}f_{n-r}, & m\bmod 4=1;\\ (-1)^nf_r, & m\bmod 4=2;\\ (-1)^{r+1+n}f_{n-r}, & m\bmod 4=3.\end{cases}$$

6.1.4 Stirling Cycle Numbers

$$\begin{bmatrix}n+1\\k\end{bmatrix}=n\begin{bmatrix}n\\k\end{bmatrix}+\begin{bmatrix}n\\k-1\end{bmatrix}$$

$$\begin{bmatrix}n+1\\2\end{bmatrix}=n!H_n$$

$$x^{\overline{n}}=\sum_k\begin{bmatrix}n\\k\end{bmatrix}x^k$$

$$x^{\underline{n}}=\sum_k\begin{bmatrix}n\\k\end{bmatrix}(-1)^{n-k}x^k$$

6.1.5 Stirling Subset Numbers

$$\begin{Bmatrix}n+1\\k\end{Bmatrix}=k\begin{Bmatrix}n\\k\end{Bmatrix}+\begin{Bmatrix}n\\k-1\end{Bmatrix}$$

$$x^n=\sum_k\begin{Bmatrix}n\\k\end{Bmatrix}x^{\underline{k}}=\sum_k\begin{Bmatrix}n\\k\end{Bmatrix}(-1)^{n-k}x^{\overline{k}}$$

$$m!\begin{Bmatrix}n\\m\end{Bmatrix}=\sum_k\binom{m}{k}k^n(-1)^{m-k}$$

6.1.6 Eulerian Numbers

$$\left\langle{n\atop k}\right\rangle=(k+1)\left\langle{n-1\atop k}\right\rangle+(n-k)\left\langle{n-1\atop k-1}\right\rangle$$

$$x^n=\sum_k\left\langle{n\atop k}\right\rangle\binom{x+k}{n}$$

$$\left\langle{n\atop m}\right\rangle=\sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$$

6.1.7 Harmonic Numbers

$$\sum_{k=1}^{n}H_k=(n+1)H_n-n$$

$$\sum_{k=1}^{n}kH_k=\frac{n(n+1)}{2}H_n-\frac{n(n-1)}{4}$$

$$\sum_{k=1}^{n}\binom{k}{m}H_k=\binom{n+1}{m+1}(H_{n+1}-\frac{1}{m+1})$$

6.1.8 Pentagonal Number Theorem

$$\prod_{n=1}^{\infty}(1-x^n)=\sum_{n=-\infty}^{\infty}(-1)^kx^{k(3k-1)/2}$$

$$p(n)=p(n-1)+p(n-2)-p(n-5)-p(n-7)+\cdots$$

$$f(n,k)=p(n)-p(n-k)-p(n-2k)+p(n-5k)+p(n-7k)-\cdots$$

6.1.9 Bell Numbers

$$B_{n+1}=\sum_{k=0}^{n}\binom{n}{k}B_k$$

$$B_{p^m+n}\equiv mB_n+B_{n+1}\pmod p$$

6.1.10 Bernoulli Numbers

$$B_n=1-\sum_{k=0}^{n-1}\binom{n}{k}\frac{B_k}{n-k+1}$$

$$G(x)=\sum_{k=0}^{\infty}\frac{B_k}{k!}x^k=\frac{1}{\sum_{k=0}^{\infty}\frac{x^k}{(k+1)!}}$$

$$S_m(n)=\frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_kn^{m-k+1}$$

6.1.11 Tetrahedron Volume

$$V=\frac{\sqrt{4u^2v^2w^2-\sum_{cyc}u^2(v^2+w^2-U^2)^2+\prod_{cyc}(v^2+w^2-U^2)}}{12}$$

6.1.12 BEST Thoerem

$$\mathrm{ec}(G)=t_w(G)\prod_{v\in V}(\deg(v)-1)!$$

When calculating $t_w(G)$ for directed multigraphs, the entry $q_{i,j}$ for distinct i and j equals $-m$, where m is the number of edges from i to j, and the entry $q_{i,i}$ equals the indegree of i minus the number of loops at i.

6.1.13 Others

$$S_j=\sum_{k=1}^{n}x_k^j$$

$$h_m=\sum_{1\le j_1<\cdots<j_m\le n}x_{j_1}\cdots x_{j_m}$$

$$H_m=\sum_{1\le j_1\le\cdots\le j_m\le n}x_{j_1}\cdots x_{j_m}$$

$$h_n=\frac{1}{n}\sum_{k=1}^{n}(-1)^{k+1}S_kh_{n-k}$$

$$H_n=\frac{1}{n}\sum_{k=1}^{n}S_kH_{n-k}$$

$$\sum_{k=0}^{n}kc^k=\frac{nc^{n+2}-(n+1)c^{n+1}+c}{(c-1)^2}$$

$$n!=\sqrt{2\pi n}(\frac{n}{e})^n(1+\frac{1}{12n}+\frac{1}{288n^2}+O(\frac{1}{n^3}))$$

$$max\{x_a-x_b,y_a-y_b,z_a-z_b\}-\min\{x_a-x_b,y_a-y_b,z_a-z_b\}$$
$$=\frac{1}{2}\sum_{cyc}|(x_a-y_a)-(x_b-y_b)|$$

$$(a+b)(b+c)(c+a)=\frac{(a+b+c)^3-a^3-b^3-c^3}{3}$$

6.2 FFT, NTT, FWT

```cpp
using comp = complex<ld>; //手写comp运行时间: d /= 1.5, ld /= 3
using vc = vector<comp>; const ld PI = atan(1.0L) * 4;
void dft(vc &a, int op = 1) {
  int n = a.size();
  for (int i = 1, j = 0; i < n - 1; ++i) {
    for (int s = n; j ^= s >>= 1, ~j & s;) {}
    if (i < j) swap(a[i], a[j]);
  }
  for (int i = 1; i < n; i *= 2) {
    comp u = {cos(PI / i), op * sin(PI / i)};
    for (int j = 0; j < n; j += i * 2) {
      comp w = 1;
      for (int k = 0; k < i; ++k, w = w * u) {
        comp x = a[j + k], y = w * a[j + k + i];
        a[j + k] = x + y, a[j + k + i] = x - y;
  }}}
  if (op == -1) for (int i = 0; i < n; ++i) a[i] /= n;
}
vl multi(vl a, vl b) { // 单项答案: d <= 1e12, ld <= 1e14
  int n = 1; while (n < a.size() + b.size() - 1) n *= 2;
  vc fa(n), fb(n), fc(n);
  copy(a.begin(), a.end(), fa.begin()); dft(fa);
  copy(b.begin(), b.end(), fb.begin()); dft(fb);
  for (int i = 0; i < n; ++i) fc[i] = fa[i] * fb[i];
  dft(fc, -1); vl c(n);
  for (int i = 0; i < n; ++i) c[i] = round(fc[i].real());
  return c;
}
vl multi(vl a, vl b) { // 单项答案: d <= 1e21, ld <= 1e23
  int n = 1; while (n < a.size() + b.size() - 1) n *= 2;
  a.resize(n); b.resize(n);
  ll q = round(sqrt(MAX_ELE)); // MAX_ELE = MOD 或 1e9 等
  vc A(n), B(n), C(n), D(n);
  for (int i = 0; i < n; ++i) {
    A[i] = comp(a[i] / q, a[i] % q);
    B[i] = comp(b[i] / q, b[i] % q);
  }
  dft(A); dft(B); // 模1e9的dft长度: d <= 16384, ld <= 524288
  for (int i = 0; i < n; ++i) {
    int j = (n - i) % n;
    C[i] = B[i] * (A[i] - conj(A[j])) * comp(0, -0.5);
    D[i] = B[i] * (A[i] + conj(A[j])) * comp(0.5);
  }
  dft(C, -1); dft(D, -1); vl c(n);
  for (int i = 0; i < n; ++i)
    c[i] = (llround(D[i].real()) % MOD * q * q
    + llround(D[i].imag()) % MOD * q
    + llround(C[i].real()) % MOD * q
    + llround(C[i].imag()) % MOD) % MOD; // 求精确值时不取模
  return c;
}
void ntt(vi &a, int op = 1) { // P: 模数, G: 原根
  /* dft前5行 */
  // 注意, multi: c[i] = (ll)a[i] * b[i] % P; 否则会爆int
  for (int i = 1; i < n; i *= 2) {
    ll u = POW(G, (P - 1) / (i * 2));
    if (op == -1) u = INV(u);
    for (int j = 0; j < n; j += i * 2) {
      ll w = 1;
      for (int k = 0; k < i; ++k, w = w * u % P) {
        int x = a[j + k], y = w * a[j + k + i] % P;
        #define modto(x) ({if ((x) >= P) (x) -= P;})
        a[j + k] = x + y; modto(a[j + k]);
        a[j + k + i] = x - y + P; modto(a[j + k + i]);
        #undef modto
  }}}
  if (op == -1) { ll inv = INV(n);
    for (int i = 0; i < n; ++i) a[i] = a[i] * inv % P; }
}
void fwt(ll *a, int n, int op = 1) {
  for (int i = 1; i < n; i *= 2)
    for (int j = 0; j < n; j += i * 2)
      for (int k = 0; k < i; ++k) {
        ll x = a[j + k], y = a[j + k + i];
        a[j + k] = x + y, a[j + k + i] = x - y;
        if (op == -1) a[j + k] /= 2, a[j + k + i] /= 2;
      }
  //if (op == -1) for (int i = 0; i < n; ++i) a[i] /= n;
}
// 带模的情况下建议注释掉76行的逆变换改为使用78行的, 参考NTT 67行
// 模任意数时先将模数乘以n, 并使用快速乘防止爆ll.
```

6.3 Polynomial Inversion (xxx)

```cpp
void FFT_Init() {
  for (K = 1; K < n << 1; K <<= 1); inv_K = inver(K);
  w[0][0] = w[0][K] = w[1][0] = w[1][K] = 1;
  int G = fpm(g, (P - 1) / K);
  FOR(i, 1, K - 1)
    w[0][i] = (ll)w[0][i - 1] * G % P;
  FOR(i, 0, K)
    w[1][i] = w[0][K - i];
}
void FFT(int X[], int k, int v) {
  int i, j, l;
  for (i = j = 0; i < k; i++) {
    if (i > j) swap(X[i], X[j]);
    for (l = k >> 1; (j ^= l) < l; l >>= 1);
  }
  for (i = 2; i <= k; i <<= 1)
    for (j = 0; j < k; j += i)
      for (l = 0; l < i >> 1; l++) {
        int t = (ll)X[j+l+(i >> 1)] * w[v][(K/i)*l] % P;
        X[j + l + (i >> 1)] = ((ll)X[j + l] - t + P) % P;
        X[j+l] = ((ll)X[j + l] + t) % P;
      }}
void GetInv(int A[], int A0[], int t) {
  if (t == 1) { A0[0] = fpm(A[0], P - 2, P); return; }
  GetInv(A, A0, (t + 1) >> 1);
  int k = 1; for (; k <= (t << 1) + 3; k <<= 1);
  int inv_k = fpm(k, P - 2, P);
  FOR (i, 0, t - 1) { tmp[i] = A[i]; }
  FOR (i, t, k - 1) { tmp[i] = 0; }
  FFT(tmp, k, 0); FFT(A0, k, 0);
  FOR (i, 0, k - 1) {
    tmp[i] = 2 - (ll)tmp[i] * A0[i] % P + P; mod(tmp[i]);
  }
  FOR (i, 0, k - 1) { A0[i] = (ll)A0[i] * tmp[i] % P; }
  FFT(A0, k, 1);
  FOR (i, 0, t - 1) { A0[i] = (ll)A0[i] * inv_k % P; }
  FOR (i, t, k - 1) { A0[i] = 0; }
}
```

6.4 Phi, Möbius

```cpp
phi[1] = mu[1] = 1;
for (int i = 2; i <= N; ++i) {
  int j = i / p[i];
  phi[i] = phi[j] * (p[j] == p[i] ? p[i] : p[i] - 1);
  mu[i] = (p[j] == p[i] ? 0 : -mu[j]);
}
```

6.5 ExGCD, CRT, Inverse

```cpp
ll exgcd(ll a, ll b, ll &x, ll &y) { // ax + by = gcd(a,b)
  if (b == 0) return x = 1, y = 0, a;
  ll t = exgcd(b, a % b, y, x);
  y -= a / b * x;
  return t;
}
bool merge(pll a, pll b, pll &re) { // pll : {div, rem}
  ll x, y, g = exgcd(a.div, b.div, x, y);
  if ((a.rem - b.rem) % g != 0) return false;
  ll m = a.div / g * b.div;
  x = -x % m * (__int128)((a.rem - b.rem) / g) % m;
  re = {m, (((__int128)x * a.div + a.rem) % m + m) % m};
  return true;
}
ll CRT(vpll s) { // 不互质CRT
  pll re = {1, 0};
  for (auto i : s) if (!merge(re, i, re)) return -1; //无解
  return re.rem;
}
ll CRT(vpll s) { // 互质CRT
  ll re = 0, m = 1; for (auto i : s) m *= i.div;
  for (auto i : s) {
    ll x, y, t = m / i.div; exgcd(t, i.div, x, y);
    re = (re + (__int128)t * x * i.rem) % m;
  }
  return (re + m) % m;
}
ll INV(ll a, ll n) { // 任意模数逆元
  ll x, y;
  if (exgcd(a, n, x, y) == 1) return (x % n + n) % n;
  return -1; // 无解: gcd > 1 或 a,n 均为 0
```

```
}
// 线性逆元(质模数)
inv[1] = fac[0] = fac[1] = ifac[0] = ifac[1] = 1;
for (int i = 2; i <= N; ++i) {
  inv[i] = (ll)MOD / i * (MOD - inv[MOD % i]) % MOD;
  fac[i] = (ll)fac[i - 1] * i % MOD;
  ifac[i] = (ll)ifac[i - 1] * inv[i] % MOD;
}
```

6.6 Miller Rabin, Pollard's Rho

```
ll qmul(ll a, ll b, ll m) { a %= m; b %= m;
  ll r = a * b, s = ld(a) * b / m;
  return ((r - m * s) % m + m) % m;
}
bool miller_rabin(ll n, ll base) {
  ll n2 = n - 1, s = 0;
  while(n2 % 2 == 0) n2 /= 2, s ++;
  ll t = qpow(base, n2, n);
  if(t == 1 || t == n - 1) return true;
  for(s --; s >= 0; s --)
    if((t = qmul(t, t, n)) == n - 1) return true;
  return false;
}
bool is_prime(ll n) {
  static ll base[] = {2,3,5,7,11,13,17,19,23,29,31,37};
/*static ll lim[]={4,2047,1373653,25326001,3215031751LL,21\
52302898747LL,3474749660383LL,341550071728321LL,0,0,0,0};*/
  if(n < 2) return false;
  for(int i = 0; i < 12 && base[i] < n; i ++) {
    // if(n < lim[i]) return true;
    if(!miller_rabin(n, base[i])) return false;
  }
  return true;
}
ll f(ll x, ll m) { return (qmul(x, x, m) + 1) % m; }
ll rho(ll n) {
  if(is_prime(n)) return n;
  if(n % 2 == 0) return 2;
  for(int i = 1; ; i ++) {
    ll x = i, y = f(x, n), p = __gcd(y - x, n);
    while(p == 1) { x = f(x, n); y = f(f(y, n), n);
      p = __gcd((y - x + n) % n, n) % n;
    }
    if(p != 0 && p != n) return p;
}} // 分解时需特判n = 1
```

6.7 Quadratic Residue

```
ll quadratic_residue(ll a, ll p) {
  if(p == 2) return 1;
  ll p2 = p / 2, tmp = qpow(a, p2, p);
  if(tmp == p - 1) return 0;
  if((p + 1) % 4 == 0) return qpow(a, (p + 1) / 4, p);
  ll t = 0, h = p - 1, b, pb;
  for(; h % 2 == 0; h /= 2) t ++;
  if(t >= 2) {
    do { b = rand() % (p - 2) + 1;
    } while(qpow(b, p / 2, p) != p - 1);
    pb = qpow(b, h, p);
  }
  ll s = qpow(a, h / 2, p);
  for(int i = 2; i <= t; i ++) {
    ll ss = (s * s) % p * a % p;
    for(int j = 0; i + j < t; j ++) ss = ss * ss % p;
    if(ss + 1 == p) s = s * pb % p; pb = pb * pb % p;
  } return s * a % p;
} // x^2 = a (mod p), 0 < a < p, 返回0时无解, 使用前需srand
```

6.8 Lattice Count

直线下整点计数: $\sum_{0\le i<n}\lfloor\frac{a+b\cdot i}{m}\rfloor$ $(n,m>0,a,b\ge 0)$

```
ll cal(ll n, ll a, ll b, ll m) {
  return b == 0 ? n * (a / m) :
  a >= m ? n * (a / m) + cal(n, a % m, b, m) :
  b >= m ? (n - 1) * n / 2 * (b / m) + cal(n, a, b % m, m)
         : cal((a + b * n) / m, (a + b * n) % m, m, b);
}
```

6.9 单变元模线性不等式

满足 $l \le d\cdot x \bmod m \le r$ 的最小非负整数解 x. $(0\le d,l,r<m; l\le r)$

```
ll cal(ll m, ll d, ll l, ll r) {
  if (l == 0) return 0;
  if (d == 0) return MXL; // 无解
  if (d * 2 > m) return cal(m, m - d, m - r, m - l);
  if ((l - 1) / d < r / d) return (l - 1) / d + 1;
  ll k = cal(d, (-m % d + d) % d, l % d, r % d);
  return k == MXL ? MXL : (k * m + l - 1) / d + 1; // 无解2
}
```

6.10 Pell's Equation

$x_{k+1} = x_0x_k + ny_0y_k,\ y_{k+1} = x_0y_k + y_0x_k$

```
pll pell(ll n) {
  static ll p[N], q[N], g[N], h[N], a[N];
  p[1] = q[0] = h[1] = 1; p[0] = q[1] = g[1] = 0;
  a[2] = (ll)(floor(sqrtl(n) + 1e-7L));
  for(int i = 2; ; i ++) {
    g[i] = -g[i - 1] + a[i] * h[i - 1];
    h[i] = (n - g[i] * g[i]) / h[i - 1];
    a[i + 1] = (g[i] + a[2]) / h[i];
    p[i] = a[i] * p[i - 1] + p[i - 2];
    q[i] = a[i] * q[i - 1] + q[i - 2];
    if(p[i] * p[i] - n * q[i] * q[i] == 1)
      return {p[i], q[i]};
}} // x^2 - n * y^2 = 1最小正整数根, n为完全平方数时无解.
```

6.11 $n!\bmod p^q$

```
ll get(int n, int mod, int p) {
  ll ans = 1;
  for(int i = 1; i <= n; i ++)
    if(i % p != 0) ans = ans * i % mod;
  return ans;
}
pll solve(ll n, int mod, int p) {
  ll init = get(mod, mod, p); pll ans = {1, 0};
  for(ll now = p; now <= n; now *= p) {
    ans.se += n / now;
    if(now > n / p) break;
  }
  while(n > 0) {
    ans.fi = ans.fi * qpow(init, n / mod, mod) % mod;
    ans.fi = ans.fi * get(n % mod, mod, p) % mod;
    n /= p;
  }
  return ans;
}
//mod = p ^ q, 返回{n!不含因子p的乘积, n!中因子p的指数}
//后续求逆元需使用exgcd
```

6.12 ExBSGS (cx_love)

```
int bsgs(int A , int B , int C) { //a ^ x == b (mod c)
  int tmp , i;
  for(i = 0 , tmp = 1 % C ; i <= 50 ; i++ , tmp = (long
  long)tmp * A % C)
    if(tmp == B)return i;
  int d = 0 , D = 1 % C;
  while((tmp = __gcd(A , C)) != 1) {
    if(B % tmp > 0)return -1;
    B /= tmp;
    C /= tmp;
    D = (long long)D * (A / tmp) % C;
    d++;
  }
  int blk = (int)(ceil(sqrt(double(C))) + 1e-5);
  map<int,int> table;
  for(i = 0 , tmp = 1 % C ; i <= blk ; i++ , tmp = (long
  long)tmp * A % C)
    if(table.find(tmp) == table.end())
      table[tmp] = i;
  tmp = 1 % C;
  for(int i = 1 ; i <= blk ; i++)tmp = (long long)tmp * A
  % C;
  for(i = 0 ; i <= blk ; i++ , D = (long long)D * tmp % C)
   {
    int inv = (long long)inverse(D , C) * B % C;
    if(table.find(inv) != table.end())
      return i * blk + table[inv] + d;
```

```
  }
  return -1;
}
```

6.13 Simplex

```
vd simplex(vector <vd> a, vd b, vd c) {
  int n = a.size(), m = a[0].size() + 1, r = n, s = m - 1;
  vector <vd> d(n + 2, vd(m + 1, 0)); vd x(m - 1);
  vi ix(n + m); iota(ix.begin(), ix.end(), 0);
  for(int i = 0; i < n; i ++) {
    for(int j = 0; j < m - 1; j ++) d[i][j] = -a[i][j];
      d[i][m - 1] = 1; d[i][m] = b[i];
    if(d[r][m] > d[i][m]) r = i;
  }
  for(int j = 0; j < m - 1; j ++) d[n][j] = c[j];
  d[n + 1][m - 1] = -1;
  while(true) {
    if(r < n) { vd su;
      swap(ix[s], ix[r + m]); d[r][s] = 1 / d[r][s];
      for(int j = 0; j <= m; j ++) if(j != s) {
        d[r][j] *= -d[r][s]; if(d[r][j]) su.pb(j);
      }
      for(int i = 0; i <= n + 1; i ++) if(i != r) {
        for(int j = 0; j < su.size(); j ++)
          d[i][su[j]] += d[r][su[j]] * d[i][s];
        d[i][s] *= d[r][s];
    }} r = s = -1;
    for(int j = 0; j < m; j ++) if(s < 0 || ix[s] > ix[j])
      if(d[n + 1][j] > Eps || d[n + 1][j] > -Eps &&
        d[n][j] > Eps) s = j; if(s < 0) break;
    for(int i = 0; i < n; i ++) if(d[i][s] < -Eps) {
      if(r < 0) { r = i; continue; }
      double e = d[r][m] / d[r][s] - d[i][m] / d[i][s];
      if(e < -Eps || e < Eps &&
        ix[r + m] > ix[i + m]) r = i;
    } if(r < 0) return vd(); // 无界
  } if(d[n + 1][m] < -Eps) return vd(); // 无解
  for(int i = m; i < n + m; i ++)
    if(ix[i] < m - 1) x[ix[i]] = d[i - m][m];
  return x;
}
```

6.14 Simpson

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{8}\left[f(a)+3f\left(\frac{2a+b}{3}\right)+3f\left(\frac{a+2b}{3}\right)+f(b)\right]$$

```
d simpson(d fl,d fr,d fmid,d l,d r) {
  return (fl+fr+4.0*fmid)*(r-l)/6.0; }
d rsimpson(d slr,d fl,d fr,d fmid,d l,d r) {
  d mid = (l+r)/2,fml = f((l+mid)/2),fmr = f((mid+r)/2);
  d slm = simpson(fl,fmid,fml,l,mid);
  d smr = simpson(fmid,fr,fmr,mid,r);
  if(fabs(slr - smr - slm) / slr < eps)return slm + smr;
  return rsimpson(slm,fl,fmid,fml,l,mid)+
    rsimpson(smr,fmid,fr,fmr,mid,r);
}
```

6.15 Primitive Root

对 $p-1$ 的每个质因子 a, $g^{(p-1)/a} \bmod p$ 均不为 1.

```
ll primroot(ll p) {
  ll s = p - 1; vector<ll> f; // factor
  for (ll i = 2; i * i <= s; ++i) while (s % i == 0)
    f.pb(i), s /= i;
  if (s > 1) f.pb(s);
  for (ll i = 1; ; ++i)
    if (none_of(f.begin(), f.end(),
      [&](ll a) { return qpow(i, (p - 1) / a, p) == 1; }))
      return i;
}
```

7 Other

7.1 表达式计算

```
Val expr(char pre, Val a) { // eval: p = s, expr('(', next())
  if (!(pri(pre) < pri(*p))) return a;
  char op = *p++;
  return expr(pre, cal(op, a, expr(op, next())));
}
```

7.2 Date

```
// weekday = (id + 1) % 7; {Sun = 0, Mon = 1, ...}
int dateid(int y, int m, int d) { // id(公元1年1月1日) = 0
  if (m < 3) --y, m += 12;
  return 365 * y + y / 4 - y / 100 + y / 400
      + (153 * (m - 3) + 2) / 5 + d - 307; }
// y <= 0时将'/'改为向下取整后即可保证正确性(或统一加400的倍数年)
auto date(int id) {
  int x = id + 1789995, n, i, j, y, m, d;
  n = 4 * x / 146097; x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001; x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447; d = x - 2447 * j / 80; x = j / 11;
  m = j + 2 - 12 * x; y = 100 * (n - 49) + i + x;
  return make_tuple(y, m, d); }
```

7.3 Iterator

```
// 字典序枚举C(n,k), (k > 0, n <= 30)
for (int c = (1 << k) - 1; c < (1 << n); ) {
  // foo
  int x = c & -c, y = c + x;
  c = (((c & ~y) / x) >> 1) | y;
}
// O(n^0.5) 枚举[n / i]相同的i值中最大的一个
for (int i = 1; ; i = n / (n / (i + 1))) {
  // foo
  if (i == n) break;
}
```

7.4 DLX

```
struct Node { Node *u, *d, *l, *r, *c; int s, row; }
  *h, *col[mxm], mem[mxv]; int sz, n; // 行列均为0下标
#define forc(i,x,f) for (auto i = x->f; i != x; i = i->f)
#define link(h,t,l,r) (t->l=h->l, t->r=h, h->l=h->l->r=t)
Node *NEW(int ts = 0, int trow = 0) {
  auto t = mem + ++sz; *t = {t, t, t, t, t, ts, trow};
  return t; }
void cover(Node *x) {
  x->l->r = x->r, x->r->l = x->l;
  forc(i,x,d)forc(j,i,r)
    j->u->d = j->d, j->d->u = j->u, --j->c->s; }
void resume(Node *x) {
  forc(i,x,d)forc(j,i,l) j->u->d = j->d->u = j, ++j->c->s;
  x->l->r = x->r->l = x; }
vi st, sol;
bool dfs() { // 调用顺序: clear => add * n => dfs
  if (h->r == h) { sol = st; return true; } // found sol
  auto t = h;
  forc(i,h,r) if (i->s < t->s) t = i;
  cover(t);
  forc(i,t,u) {
    forc(j,i,r) cover(j->c); st.push_back(i->row);
    if (dfs()) return true;
    forc(j,i,l) resume(j->c); st.pop_back(); }
  resume(t); return false; }
void clear(int m) { // 请仔细思考列顺序的玄学因素
  sz = n = 0; st = {}; h = NEW(MXL);
  for (int i = 0; i < m; ++i) {
    auto t = col[i] = NEW(0);
    link(h, t, l, r); }}
void add(vi a) { // 添加一行
  Node *row = NULL;
  for (auto i : a) {
    auto t = NEW(0, n); t->c = col[i], ++t->c->s;
    link(col[i], t, u, d);
    if (row) link(row, t, l, r); else row = t; }
  ++n; }
```

7.5 Fast-Longest-Common-Subsequence (Saga)

Complexity $O(N^2/64)$

```
typedef unsigned long long LL;
LL cmask[26][1111];
LL f[2][1111], X[1111];
```

```
int solve(char *A, char *B, int n, int m) {
  memset(cmask, 0, sizeof cmask);
  for (int i = 0; i < n; ++i) cmask[A[i] - 'a'][i >> 6] |=
   1ULL << (i & 63);
  memset(f, 0, sizeof f);
  int L = (n - 1) / 64 + 1;
  for (int i = 0; i < m; ++i) {
    int id = B[i] - 'a';
    int cur = i & 1, nxt = cur ^ 1;
    for (int j = 0; j < L; ++j) X[j] = f[cur][j] | cmask[
    id][j];
    for (int j = 0, x = 1; j < L; ++j) {
      int y = f[cur][j] >> 63 & 1;
      f[cur][j] <<= 1; f[cur][j] |= x;
      x = y;
    }
    memcpy(f[nxt], X, sizeof X);
    for (int j = 0, x = 0; j < L; ++j) {
      LL t = f[cur][j] + x;
      x = (f[nxt][j] < t);
      f[nxt][j] -= t;
      f[nxt][j] ^= X[j];
      f[nxt][j] &= X[j];
    }
  }
  int ans = 0;
  for (int i = 0; i < L; ++i) ans += __builtin_popcountll(
  f[m & 1][i]);
  return ans;
}
```

7.6 Circular LCS (Saga)

a、b 是两个串，正常的 0 下标模式。n 是长度。

```
int n, a[N << 1], b[N << 1];
bool has(int i, int j)
{ return a[(i - 1) % n] == b[(j - 1) % n]; }
const int DELTA[3][2] = {{0, -1}, {-1, -1}, {-1, 0}};
int from[N][N];
int solve() {
  memset(from, 0, sizeof(from));
  int ret = 0;
  for (int i = 1; i <= 2 * n; ++ i) {
    from[i][0] = 2;
    int left = 0, up = 0;
    for (int j = 1; j <= n; ++ j) {
      int upleft = up + 1 + !!from[i - 1][j];
      if (!has(i, j)) upleft = INT_MIN;
      int max = std::max(left, std::max(upleft, up));
      if (left == max) from[i][j] = 0;
      else if (upleft == max) from[i][j] = 1;
      else from[i][j] = 2;
      left = max;
    }
    if (i >= n) {
      int count = 0;
      for (int x = i, y = n; y;) {
        int t = from[x][y];
        count += t == 1;
        x += DELTA[t][0]; y += DELTA[t][1];
      }
      ret = std::max(ret, count);
      int x = i - n + 1;
      from[x][0] = 0;
      int y = 0;
      while (y <= n && from[x][y] == 0) y++;
      for (; x <= i; ++ x) {
        from[x][y] = 0;
        if (x == i) break;
        for (; y <= n; ++ y) {
          if (from[x + 1][y] == 2) break;
          if (y + 1 <= n && from[x + 1][y + 1] == 1) {
            y ++; break;
  }}}}}
  return ret;
}
```

7.7 Others

```
// 格雷码
g[i] = i ^ (i >> 1);
```

8 Tips

8.1 Integration Table

8.1.1 含有 $ax+b$ 的积分 $(a \neq 0)$

1. $\int \frac{x}{ax+b}\mathrm{d}x = \frac{1}{a^2}(ax+b-b\ln|ax+b|)+C$
2. $\int \frac{x^2}{ax+b}\mathrm{d}x = \frac{1}{a^3}\left(\frac{1}{2}(ax+b)^2-2b(ax+b)+b^2\ln|ax+b|\right)+C$
3. $\int \frac{\mathrm{d}x}{x(ax+b)} = -\frac{1}{b}\ln\left|\frac{ax+b}{x}\right|+C$
4. $\int \frac{\mathrm{d}x}{x^2(ax+b)} = -\frac{1}{bx}+\frac{a}{b^2}\ln\left|\frac{ax+b}{x}\right|+C$
5. $\int \frac{x}{(ax+b)^2}\mathrm{d}x = \frac{1}{a^2}\left(\ln|ax+b|+\frac{b}{ax+b}\right)+C$
6. $\int \frac{x^2}{(ax+b)^2}\mathrm{d}x = \frac{1}{a^3}\left(ax+b-2b\ln|ax+b|-\frac{b^2}{ax+b}\right)+C$
7. $\int \frac{\mathrm{d}x}{x(ax+b)^2} = \frac{1}{b(ax+b)}-\frac{1}{b^2}\ln\left|\frac{ax+b}{x}\right|+C$

8.1.2 含有 $\sqrt{ax+b}$ 的积分

1. $\int \sqrt{ax+b}\mathrm{d}x = \frac{2}{3a}\sqrt{(ax+b)^3}+C$
2. $\int x\sqrt{ax+b}\mathrm{d}x = \frac{2}{15a^2}(3ax-2b)\sqrt{(ax+b)^3}+C$
3. $\int x^2\sqrt{ax+b}\mathrm{d}x = \frac{2}{105a^3}(15a^2x^2-12abx+8b^2)\sqrt{(ax+b)^3}+C$
4. $\int \frac{x}{\sqrt{ax+b}}\mathrm{d}x = \frac{2}{3a^2}(ax-2b)\sqrt{ax+b}+C$
5. $\int \frac{x^2}{\sqrt{ax+b}}\mathrm{d}x = \frac{2}{15a^3}(3a^2x^2-4abx+8b^2)\sqrt{ax+b}+C$
6. $\int \frac{\mathrm{d}x}{x\sqrt{ax+b}} = \begin{cases}\frac{1}{\sqrt{b}}\ln\left|\frac{\sqrt{ax+b}-\sqrt{b}}{\sqrt{ax+b}+\sqrt{b}}\right|+C & (b>0)\\ \frac{2}{\sqrt{-b}}\arctan\sqrt{\frac{ax+b}{-b}}+C & (b<0)\end{cases}$
7. $\int \frac{\mathrm{d}x}{x^2\sqrt{ax+b}} = -\frac{\sqrt{ax+b}}{bx}-\frac{a}{2b}\int\frac{\mathrm{d}x}{x\sqrt{ax+b}}$
8. $\int \frac{\sqrt{ax+b}}{x}\mathrm{d}x = 2\sqrt{ax+b}+b\int\frac{\mathrm{d}x}{x\sqrt{ax+b}}$
9. $\int \frac{\sqrt{ax+b}}{x^2}\mathrm{d}x = -\frac{\sqrt{ax+b}}{x}+\frac{a}{2}\int\frac{\mathrm{d}x}{x\sqrt{ax+b}}$

8.1.3 含有 $x^2 \pm a^2$ 的积分

1. $\int \frac{\mathrm{d}x}{x^2+a^2} = \frac{1}{a}\arctan\frac{x}{a}+C$
2. $\int \frac{\mathrm{d}x}{(x^2+a^2)^n} = \frac{x}{2(n-1)a^2(x^2+a^2)^{n-1}}+\frac{2n-3}{2(n-1)a^2}\int\frac{\mathrm{d}x}{(x^2+a^2)^{n-1}}$
3. $\int \frac{\mathrm{d}x}{x^2-a^2} = \frac{1}{2a}\ln\left|\frac{x-a}{x+a}\right|+C$

8.1.4 含有 $ax^2+b(a>0)$ 的积分

1. $\int \frac{\mathrm{d}x}{ax^2+b} = \begin{cases}\frac{1}{\sqrt{ab}}\arctan\sqrt{\frac{a}{b}}x+C & (b>0)\\ \frac{1}{2\sqrt{-ab}}\ln\left|\frac{\sqrt{a}x-\sqrt{-b}}{\sqrt{a}x+\sqrt{-b}}\right|+C & (b<0)\end{cases}$
2. $\int \frac{x}{ax^2+b}\mathrm{d}x = \frac{1}{2a}\ln|ax^2+b|+C$
3. $\int \frac{x^2}{ax^2+b}\mathrm{d}x = \frac{x}{a}-\frac{b}{a}\int\frac{\mathrm{d}x}{ax^2+b}$
4. $\int \frac{\mathrm{d}x}{x(ax^2+b)} = \frac{1}{2b}\ln\frac{x^2}{|ax^2+b|}+C$
5. $\int \frac{\mathrm{d}x}{x^2(ax^2+b)} = -\frac{1}{bx}-\frac{a}{b}\int\frac{\mathrm{d}x}{ax^2+b}$
6. $\int \frac{\mathrm{d}x}{x^3(ax^2+b)} = \frac{a}{2b^2}\ln\frac{|ax^2+b|}{x^2}-\frac{1}{2bx^2}+C$
7. $\int \frac{\mathrm{d}x}{(ax^2+b)^2} = \frac{x}{2b(ax^2+b)}+\frac{1}{2b}\int\frac{\mathrm{d}x}{ax^2+b}$

8.1.5 含有 $ax^2+bx+c(a>0)$ 的积分

1. $\frac{\mathrm{d}x}{ax^2+bx+c} = \begin{cases}\frac{2}{\sqrt{4ac-b^2}}\arctan\frac{2ax+b}{\sqrt{4ac-b^2}}+C & (b^2<4ac)\\ \frac{1}{\sqrt{b^2-4ac}}\ln\left|\frac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right|+C & (b^2>4ac)\end{cases}$
2. $\int \frac{x}{ax^2+bx+c}\mathrm{d}x = \frac{1}{2a}\ln|ax^2+bx+c|-\frac{b}{2a}\int\frac{\mathrm{d}x}{ax^2+bx+c}$

8.1.6 含有 $\sqrt{x^2+a^2}(a>0)$ 的积分

1. $\int \frac{\mathrm{d}x}{\sqrt{x^2+a^2}} = \operatorname{arsh}\frac{x}{a} + C_1 = \ln(x+\sqrt{x^2+a^2}) + C$
2. $\int \frac{\mathrm{d}x}{\sqrt{(x^2+a^2)^3}} = \frac{x}{a^2\sqrt{x^2+a^2}} + C$
3. $\int \frac{x}{\sqrt{x^2+a^2}}\mathrm{d}x = \sqrt{x^2+a^2} + C$
4. $\int \frac{x}{\sqrt{(x^2+a^2)^3}}\mathrm{d}x = -\frac{1}{\sqrt{x^2+a^2}} + C$
5. $\int \frac{x^2}{\sqrt{x^2+a^2}}\mathrm{d}x = \frac{x}{2}\sqrt{x^2+a^2} - \frac{a^2}{2}\ln(x+\sqrt{x^2+a^2}) + C$
6. $\int \frac{x^2}{\sqrt{(x^2+a^2)^3}}\mathrm{d}x = -\frac{x}{\sqrt{x^2+a^2}} + \ln(x+\sqrt{x^2+a^2}) + C$
7. $\int \frac{\mathrm{d}x}{x\sqrt{x^2+a^2}} = \frac{1}{a}\ln\frac{\sqrt{x^2+a^2}-a}{|x|} + C$
8. $\int \frac{\mathrm{d}x}{x^2\sqrt{x^2+a^2}} = -\frac{\sqrt{x^2+a^2}}{a^2 x} + C$
9. $\int \sqrt{x^2+a^2}\mathrm{d}x = \frac{x}{2}\sqrt{x^2+a^2} + \frac{a^2}{2}\ln(x+\sqrt{x^2+a^2}) + C$
10. $\int \sqrt{(x^2+a^2)^3}\mathrm{d}x = \frac{x}{8}(2x^2+5a^2)\sqrt{x^2+a^2} + \frac{3}{8}a^4\ln(x+\sqrt{x^2+a^2}) + C$
11. $\int x\sqrt{x^2+a^2}\mathrm{d}x = \frac{1}{3}\sqrt{(x^2+a^2)^3} + C$
12. $\int x^2\sqrt{x^2+a^2}\mathrm{d}x = \frac{x}{8}(2x^2+a^2)\sqrt{x^2+a^2} - \frac{a^4}{8}\ln(x+\sqrt{x^2+a^2}) + C$
13. $\int \frac{\sqrt{x^2+a^2}}{x}\mathrm{d}x = \sqrt{x^2+a^2} + a\ln\frac{\sqrt{x^2+a^2}-a}{|x|} + C$
14. $\int \frac{\sqrt{x^2+a^2}}{x^2}\mathrm{d}x = -\frac{\sqrt{x^2+a^2}}{x} + \ln(x+\sqrt{x^2+a^2}) + C$

8.1.7 含有 $\sqrt{x^2-a^2}(a>0)$ 的积分

1. $\int \frac{\mathrm{d}x}{\sqrt{x^2-a^2}} = \frac{x}{|x|}\operatorname{arch}\frac{|x|}{a} + C_1 = \ln\left|x+\sqrt{x^2-a^2}\right| + C$
2. $\int \frac{\mathrm{d}x}{\sqrt{(x^2-a^2)^3}} = -\frac{x}{a^2\sqrt{x^2-a^2}} + C$
3. $\int \frac{x}{\sqrt{x^2-a^2}}\mathrm{d}x = \sqrt{x^2-a^2} + C$
4. $\int \frac{x}{\sqrt{(x^2-a^2)^3}}\mathrm{d}x = -\frac{1}{\sqrt{x^2-a^2}} + C$
5. $\int \frac{x^2}{\sqrt{x^2-a^2}}\mathrm{d}x = \frac{x}{2}\sqrt{x^2-a^2} + \frac{a^2}{2}\ln|x+\sqrt{x^2-a^2}| + C$
6. $\int \frac{x^2}{\sqrt{(x^2-a^2)^3}}\mathrm{d}x = -\frac{x}{\sqrt{x^2-a^2}} + \ln|x+\sqrt{x^2-a^2}| + C$
7. $\int \frac{\mathrm{d}x}{x\sqrt{x^2-a^2}} = \frac{1}{a}\arccos\frac{a}{|x|} + C$
8. $\int \frac{\mathrm{d}x}{x^2\sqrt{x^2-a^2}} = \frac{\sqrt{x^2-a^2}}{a^2 x} + C$
9. $\int \sqrt{x^2-a^2}\mathrm{d}x = \frac{x}{2}\sqrt{x^2-a^2} - \frac{a^2}{2}\ln|x+\sqrt{x^2-a^2}| + C$
10. $\int \sqrt{(x^2-a^2)^3}\mathrm{d}x = \frac{x}{8}(2x^2-5a^2)\sqrt{x^2-a^2} + \frac{3}{8}a^4\ln|x+\sqrt{x^2-a^2}| + C$
11. $\int x\sqrt{x^2-a^2}\mathrm{d}x = \frac{1}{3}\sqrt{(x^2-a^2)^3} + C$
12. $\int x^2\sqrt{x^2-a^2}\mathrm{d}x = \frac{x}{8}(2x^2-a^2)\sqrt{x^2-a^2} - \frac{a^4}{8}\ln|x+\sqrt{x^2-a^2}| + C$
13. $\int \frac{\sqrt{x^2-a^2}}{x}\mathrm{d}x = \sqrt{x^2-a^2} - a\arccos\frac{a}{|x|} + C$
14. $\int \frac{\sqrt{x^2-a^2}}{x^2}\mathrm{d}x = -\frac{\sqrt{x^2-a^2}}{x} + \ln|x+\sqrt{x^2-a^2}| + C$

8.1.8 含有 $\sqrt{a^2-x^2}(a>0)$ 的积分

1. $\int \frac{\mathrm{d}x}{\sqrt{a^2-x^2}} = \arcsin\frac{x}{a} + C$
2. $\frac{\mathrm{d}x}{\sqrt{(a^2-x^2)^3}} = \frac{x}{a^2\sqrt{a^2-x^2}} + C$
3. $\int \frac{x}{\sqrt{a^2-x^2}}\mathrm{d}x = -\sqrt{a^2-x^2} + C$
4. $\int \frac{x}{\sqrt{(a^2-x^2)^3}}\mathrm{d}x = \frac{1}{\sqrt{a^2-x^2}} + C$
5. $\int \frac{x^2}{\sqrt{a^2-x^2}}\mathrm{d}x = -\frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a} + C$
6. $\int \frac{x^2}{\sqrt{(a^2-x^2)^3}}\mathrm{d}x = \frac{x}{\sqrt{a^2-x^2}} - \arcsin\frac{x}{a} + C$
7. $\int \frac{\mathrm{d}x}{x\sqrt{a^2-x^2}} = \frac{1}{a}\ln\frac{a-\sqrt{a^2-x^2}}{|x|} + C$
8. $\int \frac{\mathrm{d}x}{x^2\sqrt{a^2-x^2}} = -\frac{\sqrt{a^2-x^2}}{a^2 x} + C$
9. $\int \sqrt{a^2-x^2}\mathrm{d}x = \frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a} + C$
10. $\int \sqrt{(a^2-x^2)^3}\mathrm{d}x = \frac{x}{8}(5a^2-2x^2)\sqrt{a^2-x^2} + \frac{3}{8}a^4\arcsin\frac{x}{a} + C$
11. $\int x\sqrt{a^2-x^2}\mathrm{d}x = -\frac{1}{3}\sqrt{(a^2-x^2)^3} + C$
12. $\int x^2\sqrt{a^2-x^2}\mathrm{d}x = \frac{x}{8}(2x^2-a^2)\sqrt{a^2-x^2} + \frac{a^4}{8}\arcsin\frac{x}{a} + C$
13. $\int \frac{\sqrt{a^2-x^2}}{x}\mathrm{d}x = \sqrt{a^2-x^2} + a\ln\frac{a-\sqrt{a^2-x^2}}{|x|} + C$
14. $\int \frac{\sqrt{a^2-x^2}}{x^2}\mathrm{d}x = -\frac{\sqrt{a^2-x^2}}{x} - \arcsin\frac{x}{a} + C$

8.1.9 含有 $\sqrt{\pm ax^2+bx+c}(a>0)$ 的积分

1. $\int \frac{\mathrm{d}x}{\sqrt{ax^2+bx+c}} = \frac{1}{\sqrt{a}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
2. $\int \sqrt{ax^2+bx+c}\mathrm{d}x = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
3. $\int \frac{x}{\sqrt{ax^2+bx+c}}\mathrm{d}x = \frac{1}{a}\sqrt{ax^2+bx+c} - \frac{b}{2\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
4. $\int \frac{\mathrm{d}x}{\sqrt{c+bx-ax^2}} = -\frac{1}{\sqrt{a}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$
5. $\int \sqrt{c+bx-ax^2}\mathrm{d}x = \frac{2ax-b}{4a}\sqrt{c+bx-ax^2} + \frac{b^2+4ac}{8\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$
6. $\int \frac{x}{\sqrt{c+bx-ax^2}}\mathrm{d}x = -\frac{1}{a}\sqrt{c+bx-ax^2} + \frac{b}{2\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$

8.1.10 含有 $\sqrt{\pm\frac{x-a}{x-b}}$ 或 $\sqrt{(x-a)(x-b)}$ 的积分

1. $\int \sqrt{\frac{x-a}{x-b}}\mathrm{d}x = (x-b)\sqrt{\frac{x-a}{x-b}} + (b-a)\ln(\sqrt{|x-a|}+\sqrt{|x-b|}) + C$
2. $\int \sqrt{\frac{x-a}{b-x}}\mathrm{d}x = (x-b)\sqrt{\frac{x-a}{b-x}} + (b-a)\arcsin\sqrt{\frac{x-a}{b-x}} + C$
3. $\int \frac{\mathrm{d}x}{\sqrt{(x-a)(b-x)}} = 2\arcsin\sqrt{\frac{x-a}{b-x}} + C\ (a<b)$
4. $$\int \sqrt{(x-a)(b-x)}\mathrm{d}x = \frac{2x-a-b}{4}\sqrt{(x-a)(b-x)} + \frac{(b-a)^2}{4}\arcsin\sqrt{\frac{x-a}{b-x}} + C, (a<b) \quad (1)$$

8.1.11 含有三角函数的积分

1. $\int \tan x\mathrm{d}x = -\ln|\cos x| + C$
2. $\int \cot x\mathrm{d}x = \ln|\sin x| + C$
3. $\int \sec x\mathrm{d}x = \ln\left|\tan\left(\frac{\pi}{4}+\frac{x}{2}\right)\right| + C = \ln|\sec x+\tan x| + C$
4. $\int \csc x\mathrm{d}x = \ln\left|\tan\frac{x}{2}\right| + C = \ln|\csc x-\cot x| + C$
5. $\int \sec^2 x\mathrm{d}x = \tan x + C$
6. $\int \csc^2 x\mathrm{d}x = -\cot x + C$
7. $\int \sec x\tan x\mathrm{d}x = \sec x + C$
8. $\int \csc x\cot x\mathrm{d}x = -\csc x + C$
9. $\int \sin^2 x\mathrm{d}x = \frac{x}{2} - \frac{1}{4}\sin 2x + C$
10. $\int \cos^2 x\mathrm{d}x = \frac{x}{2} + \frac{1}{4}\sin 2x + C$
11. $\int \sin^n x\mathrm{d}x = -\frac{1}{n}\sin^{n-1}x\cos x + \frac{n-1}{n}\int \sin^{n-2}x\mathrm{d}x$
12. $\int \cos^n x\mathrm{d}x = \frac{1}{n}\cos^{n-1}x\sin x + \frac{n-1}{n}\int \cos^{n-2}x\mathrm{d}x$
13. $\frac{\mathrm{d}x}{\sin^n x} = -\frac{1}{n-1}\frac{\cos x}{\sin^{n-1}x} + \frac{n-2}{n-1}\int \frac{\mathrm{d}x}{\sin^{n-2}x}$
14. $\frac{\mathrm{d}x}{\cos^n x} = \frac{1}{n-1}\frac{\sin x}{\cos^{n-1}x} + \frac{n-2}{n-1}\int \frac{\mathrm{d}x}{\cos^{n-2}x}$
15. $$\begin{aligned}&\int \cos^m x\sin^n x\mathrm{d}x\\ =&\frac{1}{m+n}\cos^{m-1}x\sin^{n+1}x + \frac{m-1}{m+n}\int \cos^{m-2}x\sin^n x\mathrm{d}x\\ =&-\frac{1}{m+n}\cos^{m+1}x\sin^{n-1}x + \frac{n-1}{m+1}\int \cos^m x\sin^{n-2}x\mathrm{d}x\end{aligned}$$

16. $\int \sin ax\cos bx\mathrm{d}x = -\frac{1}{2(a+b)}\cos(a+b)x - \frac{1}{2(a-b)}\cos(a-b)x + C$
17. $\int \sin ax\sin bx\mathrm{d}x = -\frac{1}{2(a+b)}\sin(a+b)x + \frac{1}{2(a-b)}\sin(a-b)x + C$
18. $\int \cos ax\cos bx\mathrm{d}x = \frac{1}{2(a+b)}\sin(a+b)x + \frac{1}{2(a-b)}\sin(a-b)x + C$
19. $\int \frac{\mathrm{d}x}{a+b\sin x} = \begin{cases} \frac{2}{\sqrt{a^2-b^2}}\arctan\frac{a\tan\frac{x}{2}+b}{\sqrt{a^2-b^2}} + C & (a^2>b^2) \\ \frac{1}{\sqrt{b^2-a^2}}\ln\left|\frac{a\tan\frac{x}{2}+b-\sqrt{b^2-a^2}}{a\tan\frac{x}{2}+b+\sqrt{b^2-a^2}}\right| + C & (a^2<b^2) \end{cases}$
20. $\int \frac{\mathrm{d}x}{a+b\cos x} = \begin{cases} \frac{2}{a+b}\sqrt{\frac{a+b}{a-b}}\arctan\left(\sqrt{\frac{a-b}{a+b}}\tan\frac{x}{2}\right) + C & (a^2>b^2) \\ \frac{1}{a+b}\sqrt{\frac{a+b}{a-b}}\ln\left|\frac{\tan\frac{x}{2}+\sqrt{\frac{a+b}{b-a}}}{\tan\frac{x}{2}-\sqrt{\frac{a+b}{b-a}}}\right| + C & (a^2<b^2) \end{cases}$
21. $\int \frac{\mathrm{d}x}{a^2\cos^2 x+b^2\sin^2 x} = \frac{1}{ab}\arctan\left(\frac{b}{a}\tan x\right) + C$
22. $\int \frac{\mathrm{d}x}{a^2\cos^2 x-b^2\sin^2 x} = \frac{1}{2ab}\ln\left|\frac{b\tan x+a}{b\tan x-a}\right| + C$
23. $\int x\sin ax\mathrm{d}x = \frac{1}{a^2}\sin ax - \frac{1}{a}x\cos ax + C$
24. $\int x^2\sin ax\mathrm{d}x = -\frac{1}{a}x^2\cos ax + \frac{2}{a^2}x\sin ax + \frac{2}{a^3}\cos ax + C$
25. $\int x\cos ax\mathrm{d}x = \frac{1}{a^2}\cos ax + \frac{1}{a}x\sin ax + C$
26. $\int x^2\cos ax\mathrm{d}x = \frac{1}{a}x^2\sin ax + \frac{2}{a^2}x\cos ax - \frac{2}{a^3}\sin ax + C$

8.1.12 含有反三角函数的积分 (其中 $a>0$)

1. $\int \arcsin\frac{x}{a}\mathrm{d}x = x\arcsin\frac{x}{a} + \sqrt{a^2-x^2} + C$
2. $\int x\arcsin\frac{x}{a}\mathrm{d}x = (\frac{x^2}{2}-\frac{a^2}{4})\arcsin\frac{x}{a} + \frac{x}{4}\sqrt{x^2-x^2} + C$
3. $\int x^2\arcsin\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arcsin\frac{x}{a} + \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
4. $\int \arccos\frac{x}{a}\mathrm{d}x = x\ arccos\frac{x}{a} - \sqrt{a^2-x^2} + C$
5. $\int x\arccos\frac{x}{a}\mathrm{d}x = (\frac{x^2}{2}-\frac{a^2}{4})\arccos\frac{x}{a} - \frac{x}{4}\sqrt{a^2-x^2} + C$
6. $\int x^2\arccos\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arccos\frac{x}{a} - \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
7. $\int \arctan\frac{x}{a}\mathrm{d}x = x\arctan\frac{x}{a} - \frac{a}{2}\ln(a^2+x^2) + C$
8. $\int x\arctan\frac{x}{a}\mathrm{d}x = \frac{1}{2}(a^2+x^2)\arctan\frac{x}{a} - \frac{a}{2}x + C$
9. $\int x^2\arctan\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arctan\frac{x}{a} - \frac{a}{6}x^2 + \frac{a^3}{6}\ln(a^2+x^2) + C$

8.1.13 含有指数函数的积分

1. $\int a^x\mathrm{d}x = \frac{1}{\ln a}a^x + C$
2. $\int \mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a}a^{ax} + C$
3. $\int x\mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a^2}(ax-1)a^{ax} + C$
4. $\int x^n\mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a}x^n\mathrm{e}^{ax} - \frac{n}{a}\int x^{n-1}\mathrm{e}^{ax}\mathrm{d}x$
5. $\int xa^x\mathrm{d}x = \frac{x}{\ln a}a^x - \frac{1}{(\ln a)^2}a^x + C$
6. $\int x^na^x\mathrm{d}x = \frac{1}{\ln a}x^na^x - \frac{n}{\ln a}\int x^{n-1}a^x\mathrm{d}x$
7. $\int \mathrm{e}^{ax}\sin bx\mathrm{d}x = \frac{1}{a^2+b^2}\mathrm{e}^{ax}(a\sin bx - b\cos bx) + C$
8. $\int \mathrm{e}^{ax}\cos bx\mathrm{d}x = \frac{1}{a^2+b^2}\mathrm{e}^{ax}(b\sin bx + a\cos bx) + C$
9. $\int \mathrm{e}^{ax}\sin^n bx\mathrm{d}x = \frac{1}{a^2+b^2n^2}\mathrm{e}^{ax}\sin^{n-1}bx(a\sin bx - nb\cos bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int \mathrm{e}^{ax}\sin^{n-2}bx\mathrm{d}x$
10. $\int \mathrm{e}^{ax}\cos^n bx\mathrm{d}x = \frac{1}{a^2+b^2n^2}\mathrm{e}^{ax}\cos^{n-1}bx(a\cos bx + nb\sin bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int \mathrm{e}^{ax}\cos^{n-2}bx\mathrm{d}x$

8.1.14 含有对数函数的积分

1. $\int \ln x\mathrm{d}x = x\ln x - x + C$
2. $\int \frac{\mathrm{d}x}{x\ln x} = \ln|\ln x| + C$
3. $\int x^n\ln x\mathrm{d}x = \frac{1}{n+1}x^{n+1}(\ln x - \frac{1}{n+1}) + C$
4. $\int (\ln x)^n\mathrm{d}x = x(\ln x)^n - n\int (\ln x)^{n-1}\mathrm{d}x$
5. $\int x^m(\ln x)^n\mathrm{d}x = \frac{1}{m+1}x^{m+1}(\ln x)^n - \frac{n}{m+1}\int x^m(\ln x)^{n-1}\mathrm{d}x$

8.2 Primes

```
for ((i=100;;i++)); do if [[ `factor $i` = "$i: $i" ]];
  then echo $i; break; fi done
1E2+1 2E2+11 5E2+3 1E3+9 2E3+3 5E3+3   1E4+7 2E4+11 5E4+21
1E5+3 2E5+3 5E5+9  1E6+3 2E6+3 5E6+11  1E7+19 2E7+3 5E7+17
1E8+7,37,39,49     1E9+7,9,21,33       1E10+19,33,61,69
1E11+3,19,57,63    1E12+39,61,63,91    1E13+37,51,99,129
1E14+31,67,97,99   1E15+37,91,159,187  1E16+61,69,79,99
1E17+3,13,19,21    1E18+3,9,31,79
```

8.3 Utilities

```
ulimit -s unlimited; ulimit -a; g++ -pg ...; gprof;
gdb: bt, f; g++ -fsanitize=address -fsanitize=undefined
gdb: record, record stop, rn;
g++ -Wall -Wextra -Wpedantic
```

8.4 C++

```
// bitset
for (int i = a._Find_first(); i < a.size();
  i = a._Find_next(i))
// fgets
#define gets(s) fgets((s), INT_MAX, stdin)
// srand
#include <sys/time.h>
timeval tv; gettimeofday(&tv, 0);
srand(tv.tv_sec * 1000000 + tv.tv_usec);
// stack
register char *_sp __asm__("rsp"); // rsp/esp/sp : 64/32/16
int main() { const int size = 256 << 20;
  static char *sys, *mine(new char[size] + size - 4096);
  sys = _sp; _sp = mine; main_(); _sp = sys; return 0; }
// 整数读入优化
int readint() { // 注意-2147483648；Linux下只比scanf快20%
  char c, o = 0; while (!isdigit(c = getchar())) o = c;
  int re = c - '0'; while (isdigit(c = getchar()))
    (re *= 10) += c - '0'; return o != '-' ? re : -re; }
// pb_ds：带下标(0下标)set/map，可用pair<Val,id>来模拟多重集
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template<class T> using rset = tree<T, null_type, less<T>,
  rb_tree_tag, tree_order_statistics_node_update>;
template<class T,class U> using rmap = tree<T, U, less<T>,
  rb_tree_tag, tree_order_statistics_node_update>;
rset<char> s; s.order_of_key('a'); *s.find_by_order(0);
a.join(b); // 合并，需保证a中所有元素的key都严格小于b中的
a.split(key, b); // 拆分，b获得所有 > key的元素，O(NlgN)，奇慢!!
// pb_ds：可并堆，实测Dijkstra效率不如std::priority_queue
#include <ext/pb_ds/priority_queue.hpp>
__gnu_pbds::priority_queue<int, greater<int>,
  __gnu_pbds::pairing_heap_tag> h;
  // thin_heap_tag, binomal_heap_tag, rc_binomal_heap_tag
auto it = h.push(val); h.modify(it, val); // 也支持.join()
```

8.5 Java

```
// Evaluate
import javax.script.ScriptEngineManager;
import javax.script.ScriptEngine;
public class Main {
  public static void main(String[] args) throws Exception{
    ScriptEngineManager sem = new ScriptEngineManager();
    ScriptEngine se = sem.getEngineByName("JavaScript");
    String foo = "3+4";
    System.out.println(se.eval(foo));
}}
// Java References. DecimalFormat if you want to output
// double with specific precision.
PrintStream fout =
new PrintStream(new FileOutputStream("file.out"));
BufferedInputStream fin =
new BufferedInputStream(new FileInputStream("file.in"));
BufferedReader reader =
new BufferedReader(new InputStreamReader(System.in));
Scanner in = new Scanner(fin);
StringTokenizer tokenizer = null;
public String next() {
  while (tokenizer == null || !tokenizer.hasMoreTokens()){
    try {
      tokenizer = new StringTokenizer(reader.readLine());
```

```
    } catch(IOException e) {
      throw new RuntimeException(e);
  }}
  return tokenizer.nextToken();
}
public int nextInt() {
  return Integer.parseInt(next());
  // Double.parseDouble .....
}
```

8.6 Python

```
a, b = map(int, raw_input().split())
from datetime import *
  date(y,m,d).weekday()
  date(...) + timedelta(days = ...)
import sys
  sys.exit(0)
from __future__ import print_function
  print('...', end = '')
```

8.7 Geany, test.sh

```
# Cmp: g++ -o "%e" "%f" -std=gnu++14 -g
# Bld: g++ -o "%e" "%f" -std=gnu++14 -g -Wall
# Exe: g++ -o "%e" "%f" -std=gnu++14 -g -w -O2 && bash test.sh %e
# Pnt: printfile "%f" -T 4 --line-number=1
# Keybindings > Format > Toggle line commentation = [^/]
# test.sh
f=$@*.in; if [ "`echo $f`" = $@\*.in ]; then ./$@; else
for i in $f;do echo $i; echo -----; ./$@<$i; echo; done fi
```

8.8 Constants Table

n	$n!$	$\binom{n}{n/2}$	$lcm(1\ldots n)$	P_n	B_n
2	2	2	2	2	2
3	6	3	6	3	5
4	24	6	12	5	15
5	120	10	60	7	52
6	720	20	60	11	203
7	5040	35	420	15	877
8	40320	70	840	22	4140
9	362880	126	2520	30	21147
10	3628800	252	2520	42	115975
11	39916800	462	27720	56	678570
12	479001600	924	27720	77	4213597
13	6.227e+09	1716	360360	101	27644437
14	8.718e+10	3432	360360	135	190899322
15	1.308e+12	6435	360360	176	1.383e+09
16	2.092e+13	12870	720720	231	1.048e+10
17	3.557e+14	24310	12252240	297	8.286e+10
18	6.402e+15	48620	12252240	385	6.821e+11
19	1.216e+17	92378	232792560	490	5.833e+12
20	2.433e+18	184756	232792560	627	5.172e+13
25	1.551e+25	5200300	2.677e+10	1958	4.639e+18
30	2.653e+32	155117520	2.329e+12	5604	
35	1.033e+40	4.538e+09	1.444e+14	14883	
40	8.159e+47	1.378e+11	5.343e+15	37338	
45	1.196e+56	4.117e+12	9.420e+18	89134	
50	3.041e+64	1.264e+14	3.099e+21	204226	
70	1.20e+100	1.122e+20	7.921e+28	4087968	
90	1.49e+138	1.038e+26	7.188e+38	56634173	