Contents

1 Data Structure

1.1 Splay

```
struct node_t {
  node_t();
  void update();
  int dir() { return (this == p->ch[1]); }
  void setc(node_t *c, int d) { ch[d] = c, c->p = this; }
  node_t *p, *ch[2];
  int size, cnt; // maintain tag from top to bottom (via find).
} s[maxn], *nil = s, *root;
node_t::node_t() { p = ch[0] = ch[1] = nil; }

void node_t::update() {
  if (this == nil) return;
  size = ch[0]->size + ch[1]->size + cnt;
}

node_t *newNode(int cnt) {
  ++pt;
  s[pt].cnt = cnt; s[pt].p = s[pt].ch[0] = s[pt].ch[1] = nil;
  s[pt].update();
  return &s[pt];
}

void rotate(node_t *t) {
  node_t *p = t->p;
  p->p->update();
  p->update();
  t->update();
  int d = t->dir();
  p->p->setc(t, p->dir());
  p->setc(t->ch[!d], d);
  t->setc(p, !d);
  if (p == root) root = t;
  p->update(), t->update();
}

node_t *splay(node_t *t, node_t *dst = nil) {
  while (t->p != dst) {
    if (t->p->p == dst) rotate(t);
    else if (t->dir() == t->p->dir()) rotate(t->p), rotate(t);
    else rotate(t), rotate(t);
  }
  t->update();
  return t;
}

node_t *prev(node_t *p) {
  splay(p);
  p = p->ch[0], p->update();
  while (p->ch[1] != nil) p = p->ch[1], p->update();
  return p;
}

node_t *succ(node_t *p) {
  splay(p);
  p = p->ch[1], p->update();
  while (p->ch[0] != nil) p = p->ch[0], p->update();
  return p;
}

void insert(node_t *y, node_t *x) { // Insert node x after y
  splay(y);
  if (y->ch[1] == nil) {
    y->ch[1] = x;
    x->p = y;
    y->update();
  } else {
    y = y->ch[1], y->update();
    while (y->ch[0] != nil) y = y->ch[0], y->update();
    y->ch[0] = x;
    x->p = y;
    y->update();
  }
  splay(x);
}

void removeAll(node_t *x) { // Remove all the whole subtree of x
  x->p->update();
  x->p->ch[x->dir()] = nil;
  x->p->update();
  splay(x->p);
  x->p = nil;
}

void remove(node_t *x) { // Remove the single node x
  node_t *p = prev(x); node_t *s = succ(x);
  splay(p);
  splay(s, p);
  removeAll(s->ch[0]);
}

node_t *find(node_t *t, int k) {
  t->update();
  if (t->ch[0]->size < k && t->ch[0]->size + t->cnt >= k) return t;
  if (t->ch[0]->size >= k) return find(t->ch[0], k);
  return find(t->ch[1], k - t->ch[0]->size - t->cnt);
}

node_t *findAndSplit(node_t *t, int k) {
  t->update();
  if (t->ch[0]->size < k && t->ch[0]->size + t->cnt >= k) {
    int cnt = t->cnt;
    k -= t->ch[0]->size;
    t->cnt = 1;
    splay(t);
    node_t *p = prev(t);
    if (k - 1) insert(p, newNode(k - 1));
    if (cnt - k) insert(t, newNode(cnt - k));
    return t;
  }
  if (t->ch[0]->size >= k) return findAndSplit(t->ch[0], k);
  return findAndSplit(t->ch[1], k - t->ch[0]->size - t->cnt);
}

void init() {
  pt = 0, nil->p = nil->ch[0] = nil->ch[1] = nil;
}

node_t *expose(node_t *x, node_t *y) {
  x = prev(x), y = succ(y);
  return splay(y, splay(x))->ch[0];
}
```

1.2 Dynamic Tree

```
struct node_t {
  node_t();
  node_t *ch[2], *p;
  int size, root;
  int dir() { return this == p->ch[1]; }
  void setc(node_t *c, int d) { ch[d] = c, c->p = this; }
  void update() { size = ch[0]->size + ch[1]->size + 1; }
} s[maxn], *nil = s;

node_t::node_t() {
  size = 1, root = true;
  ch[0] = ch[1] = p = nil;
}

void rotate(node_t *t) {
  node_t *p = t->p;
  int d = t->dir();
  if (!p->root) {
    p->p->setc(t, p->dir());
  } else {
    p->root = false, t->root = true;
    t->p = p->p; // Path Parent
  }
  p->setc(t->ch[!d], d);
  t->setc(p, !d);
  p->update(), t->update();
}

void splay(node_t *t) {
  // t->update(); // tag!
  while (!t->root) {
    // if (!t->p->root) t->p->p->update(); t->p->update(), t->update(); // !
    if (!t->p->root) rotate(t->dir() == t->p->dir() ? t->p : t);
    rotate(t);
  }
}

void access(node_t *x) { // Ask u, v: access(u), access(v, true), x = LCA
  node_t *y = nil;
  while (x != nil) {
    splay(x);
    // if (x->p == nil) at second call, x->ch[1](rev) + (x)_single + y
    x->ch[1]->root = true;
    x->ch[1] = y, y->root = false;
    x->update();
    y = x, x = x->p;
  }
}

void cut(node_t *x) {
  access(x);
  splay(x);
  x->ch[0]->root = true;
  x->ch[0]->p = nil;
  x->ch[0] = nil;
}

void link(node_t *x, node_t *y) {
  access(y);
  splay(y);
  y->p = x;
  access(y);
}

void init() { nil->size = 0; }
```

1.3 KD Tree

```cpp
// scnt: the number of target points
// sp: the point to search the closest target points
const int DIM = 5;
int n, d, scnt;
struct point_t {
  int x[DIM];
} points[maxn], sp;
struct node_t {
  int sd;
  node_t *l, *r;
} node[maxn], *root;
struct cmp_t {
  int dd;
  bool operator()(const point_t &a, const point_t &b) {
    return a.x[dd] < b.x[dd];
  }
} cmp[DIM];

int dist2(const point_t &a, const point_t &b) {
  int res = 0;
  for (int i = 0; i < d; ++i) {
    res += (b.x[i] - a.x[i]) * (b.x[i] - a.x[i]);
  }
  return res;
}

struct cmp2_t {
  bool operator()(const point_t &a, const point_t &b) {
    return dist2(a, sp) < dist2(b, sp);
  }
};
priority_queue<point_t, vector<point_t>, cmp2_t> pq;

node_t *build(int dd, int l, int r) {
  if (l > r) return NULL;
  sort(points + l, points + 1 + r, cmp[dd]);
  int mid = (l + r) / 2;
  node[mid].l = build((dd + 1) % d, l, mid - 1);
  node[mid].r = build((dd + 1) % d, mid + 1, r);
  node[mid].sd = dd;
  return &node[mid];
}

void search(node_t *nd) {
  int sd = nd->sd, p = nd - node;
  pq.push(points[p]);
  while (pq.size() > scnt) pq.pop();
  int d2 = (sp.x[sd] - points[p].x[sd]) * (sp.x[sd] - points[p].x[sd]);
  if (sp.x[sd] < points[p].x[sd]) {
    if (nd->l) search(nd->l);
    if (nd->r && (pq.size() < scnt || dist2(sp, pq.top()) >= d2)) search(nd->r);
  } else {
    if (nd->r) search(nd->r);
    if (nd->l && (pq.size() < scnt || dist2(sp, pq.top()) >= d2)) search(nd->l);
  }
}

void init() {
  for (int i = 0; i < DIM; ++i) cmp[i].dd = i;
  root = build(0, 1, n);
}
```

1.4 Treap

```cpp
struct node {
  int v, key, size;
  node *c[2];
  void resize() { size = c[0]->size + c[1]->size + 1; }
};
node *newNode(int _v, node *n) {
  ++ref;
  pool[ref].v = _v, pool[ref].c[0] = pool[ref].c[1] = n, pool[ref].size = 1, pool[
    ref].key = rand();
  return &pool[ref];
}
struct Treap {
  node *root, *nil;
  void rotate(node *&t, int d) {
    node *c = t->c[d];
    t->c[d] = c->c[!d];
    c->c[!d] = t;
    t->resize(); c->resize();
    t = c;
  }
  void insert(node *&t, int x) {
    if (t == nil) t = newNode(x, nil);
    else {
      if (x == t->v) return;
      int d = x > t->v;
      insert(t->c[d], x);
      if (t->c[d]->key < t->key) rotate(t, d);
      else t->resize();
    }
  }
  void remove(node *&t, int x) {
    if (t == nil) return;
    if (t->v == x) {
      int d = t->c[1]->key < t->c[0]->key;
      if (t->c[d] == nil) {
        t = nil;
        return;
      }
      rotate(t, d);
      remove(t->c[!d], x);
    } else {
      int d = x > t->v;
      remove(t->c[d], x);
    }
    t->resize();
  }
  int rank(node *t, int x) {
    if (t == nil) return 0;
    int r = t->c[0]->size;
    if (x == t->v) return r + 1;
    if (x < t->v) return rank(t->c[0], x);
    return r + 1 + rank(t->c[1], x);
  }
  int select(node *t, int k) {
    int r = t->c[0]->size;
    if (k == r + 1) return t->v;
    if (k <= r) return select(t->c[0], k);
    return select(t->c[1], k - r - 1);
  }
  int size() {
    return root->size;
  }
  void init(int *a, int n) {
    nil = newNode(0, 0);
    nil->size = 0, nil->key = ~0U >> 1;
    root = nil;
  }
};
```

1.5 President Treap

```
struct node_t {
  int key, cnt, size;
  string v;
  node_t *c[2];
  void resize() {
    size = (c[0] ? c[0]->size : 0) + cnt + (c[1] ? c[1]->size : 0);
  }
} *nil;

node_t *newNode(string v, node_t *l = nil, node_t *r = nil, int key = rand()) {
  node_t *ret = new node_t();
  ret->key = key;
  ret->cnt = v.length(), ret->v = v;
  ret->c[0] = l, ret->c[1] = r;
  ret->resize();
  return ret;
}

void init() {
  nil = newNode("", 0, 0);
  nil->size = 0, nil->key = ~0U >> 1;
}

struct PresidentTreap {
  node_t *root;
  node_t *splitL(node_t *a, int size) {
    if (a == nil || size == 0) return nil;
    if (a->c[0]->size >= size) return splitL(a->c[0], size);
    if (a->c[0]->size + a->cnt >= size) return newNode(a->v.substr(0, size - a->c
      [0]->size), a->c[0], nil, a->key);
    return newNode(a->v, a->c[0], splitL(a->c[1], size - a->c[0]->size - a->cnt), a
      ->key);
  }
  node_t *splitR(node_t *a, int size) {
    if (a == nil || size == 0) return nil;
    if (a->c[1]->size >= size) return splitR(a->c[1], size);
    if (a->c[1]->size + a->cnt >= size) return newNode(a->v.substr(a->v.length() - (
      size - a->c[1]->size), size - a->c[1]->size), nil, a->c[1], a->key);
    return newNode(a->v, splitR(a->c[0], size - a->c[1]->size - a->cnt), a->c[1], a
      ->key);
  }
  node_t *merge(node_t *a, node_t *b) {
    if (a == nil) return b;
    if (b == nil) return a;
    if (a->key > b->key) return newNode(a->v, a->c[0], merge(a->c[1], b), a->key);
    return newNode(b->v, merge(a, b->c[0]), b->c[1], b->key);
  }
  node_t *insert(string v, int p) { // insert after p
    int l = root->size;
    return merge(merge(splitL(root, p), newNode(v, nil, nil)), splitR(root, l - p));
  }
  node_t *remove(int x, int y) { // remove [x, y]
    int l = root->size;
    return merge(splitL(root, x - 1), splitR(root, l - y));
  }
};
```

1.6 President Segment Tree

```cpp
struct Node {
  int s, d;
  Node *left, *right;
} pool[maxm], *nil, *root[maxn];
int pt, a[maxn];

Node *newNode(int _d, int _s, Node *_left, Node *_right) {
  ++pt;
  pool[pt].d = _d, pool[pt].s = _s, pool[pt].left = _left, pool[pt].right = _right;
  return pool + pt;
}

Node *build(int l, int r) {
  if (l == r) return newNode(0, a[l], nil, nil);
  int mid = (l + r) / 2;
  Node *nl = build(l, mid), *nr = build(mid + 1, r);
  return newNode(0, nl->s + nr->s, nl, nr);
}

void init(int n) {
  pt = 0; nil = newNode(0, 0, NULL, NULL);
  root[0] = build(1, n);
}

void push(Node *node, int l, int r) {
  if (l == r) {
    node->d = 0;
  } else {
    if (node->d == 0) return;
    int mid = (l + r) / 2;
    Node *nl = newNode(node->left->d + node->d, node->left->s + node->d * int(mid -
      l + 1), node->left->left, node->left->right);
    Node *rl = newNode(node->right->d + node->d, node->right->s + node->d * int(r -
      mid), node->right->left, node->right->right);
    node->d = 0;
    node->left = nl;
    node->right = rl;
  }
}

int ask(Node *node, int l, int r, int ll, int rr) {
  push(node, l, r);
  if (l == ll && r == rr) return node->s;
  int mid = (l + r) / 2;
  if (rr <= mid) return ask(node->left, l, mid, ll, rr);
  else if (ll > mid) return ask(node->right, mid + 1, r, ll, rr);
  else return ask(node->left, l, mid, ll, mid) + ask(node->right, mid + 1, r, mid +
    1, rr);
}

Node *add(Node *node, int l, int r, int ll, int rr, int d) {
  push(node, l, r);
  if (l == ll && r == rr) return newNode(node->d + d, node->s + d * int(r - l + 1),
    node->left, node->right);
  int mid = (l + r) / 2;
  if (rr <= mid) {
    Node *nl = add(node->left, l, mid, ll, rr, d);
    return newNode(0, nl->s + node->right->s, nl, node->right);
  } else if (ll > mid) {
    Node *nr = add(node->right, mid + 1, r, ll, rr, d);
    return newNode(0, node->left->s + nr->s, node->left, nr);
  } else {
    Node *nl = add(node->left, l, mid, ll, mid, d);
    Node *nr = add(node->right, mid + 1, r, mid + 1, rr, d);
    return newNode(0, nl->s + nr->s, nl, nr);
  }
}
```

2 String Algorithms

2.1 Common

```cpp
// please note that all strings are indexed from 0
void kmp(const char *s, int *next) {
  --s, --next;
  next[1] = 0;
  int j = 0, n = strlen(s + 1);
  for (int i = 2; i <= n; ++i) {
    while (j > 0 && s[j + 1] != s[i]) j = next[j];
    if (s[j + 1] == s[i]) j = j + 1;
    next[i] = j;
  }
}

// s: text, t: text being searched, ex[i]: maximum l satisfying s[i...i+l-1] = t
  [0...l-1]
void exkmp(const char *s, const char *t, int *next, int *ex) {
  int n = strlen(t), m = strlen(s), k, c;
  next[0] = n;
  k = 0, c = 1;
  while (k + 1 < n && t[k] == t[k + 1]) ++k;
  next[1] = k;
  for (int i = 2; i < n; ++i) {
    int p = next[c] + c - 1;
    int l = next[i - c];
    if (i + l < p + 1) next[i] = l;
    else {
      k = max(0, p - i + 1);
      while (i + k < n && t[i + k] == t[k]) ++k;
      next[i] = k;
      c = i;
    }
  }
  k = c = 0;
  while (k < m && k < n && s[k] == t[k]) ++k;
  ex[0] = k;
  for (int i = 1; i < m; ++i) {
    int p = ex[c] + c - 1;
    int l = next[i - c];
    if (l + i < p + 1) ex[i] = next[i - c];
    else {
      k = max(0, p - i + 1);
      while (i + k < m && k < n && s[i + k] == t[k]) ++k;
      ex[i] = k;
      c = i;
    }
  }
}

// minimum representation of a string
int minimum_representation(string s) {
  s += s;
  int l = s.length(), i = 0, j = 1, k = 0;
  while (i + k < l && j + k < l) {
    if (s[i + k] == s[j + k]) {
      ++k;
    } else {
      if (s[j + k] > s[i + k]) j += k + 1;
      else i += k + 1;
      k = 0;
      if (i == j) ++j;
    }
  }
  return min(i, j);
}

// l[i], the length of palindrome at the centre of i
int manacher(const char *s, int *l) {
  int n = strlen(s);
  for (int i = 0, j = 0, k; i < n * 2; i += k, j = max(j - k, 0)) {
    while (i >= j && i + j + 1 < n * 2 && s[(i - j) / 2] == s[(i + j + 1) / 2]) ++j;
    l[i] = j;
    for (k = 1; i >= k && j >= k && l[i - k] != j - k; ++k) {
      l[i + k] = min(l[i - k], j - k);
    }
  }
  return *max_element(l, l + n + n);
}
```

2.2 Aho-Crosick Automaton

```
struct trie_t {
  bool flag;
  trie_t *child[C], *fail;
} trie[maxn], *root;
trie_t *new_trie() { return &trie[++pt]; }

void add(char *str) {
  int l = strlen(str);
  trie_t *p = root;
  for (int i = 0; i < l; ++i) {
    int ch = str[i]; // fixed to [0, C - 1], C = |SIGMA|
    if (!p->child[ch]) p->child[ch] = new_trie();
    p = p->child[ch];
  }
  p->flag = true;
}

void build() {
  queue<trie_t *> q;
  q.push(root);
  while (!q.empty()) {
    trie_t *p = q.front(), *t;
    q.pop();
    for (int i = 0; i < C; ++i) {
      t = p->fail;
      while (t && !t->child[i]) t = t->child[i];
      t = !t ? root : t->child[i];
      if (p->child[i]) {
        p->child[i]->fail = t;
        p->child[i]->flag |= t->flag;
        q.push(p->child[i]);
      } else p->child[i] = t;
    }
  }
}
```

2.3 Suffix Array

```
const int maxn = 100002, logn = 21, maxint = 0x7f7f7f7f;
int n, sa[maxn], r[maxn + maxn], h[maxn], mv[maxn][logn];
// r is `doubled`
void initlg() {
  _lg[1] = 0;
  for (int i = 2; i < maxn; ++i) _lg[i] = _lg[i - 1] + ((i & (i - 1)) == 0 ? 1 : 0);
}
// remember to clear r & call initlg() before calling
void construct(int n, int *s) { // initlg();
  static pair<int/* Type */, int> ord[maxn];
  for (int i = 1; i <= n; ++i) ord[i] = make_pair(s[i], i);
  sort(ord + 1, ord + 1 + n);
  for (int i = 1; i <= n; ++i) {
    sa[i] = ord[i].second;
    r[sa[i]] = (i == 1 ? 1 : r[sa[i - 1]] + (ord[i - 1].first != ord[i].first));
  }
  static int tr[maxn], tsa[maxn], c[maxn];
  for (int l = 1; l < n; l <<= 1) {
    int cnt = 0;
    for (int i = 1; i <= n; ++i) if (sa[i] + l > n) tsa[++cnt] = sa[i];
    for (int i = 1; i <= n; ++i) if (sa[i] > l) tsa[++cnt] = sa[i] - l;
    memset(c, 0, sizeof(c));
    for (int i = 1; i <= n; ++i) ++c[r[i]];
    for (int i = 1; i <= n; ++i) c[i] += c[i - 1];
    for (int i = n; i >= 1; --i) sa[c[r[tsa[i]]]--] = tsa[i];
    tr[sa[1]] = 1;
    for (int i = 2; i <= n; ++i) {
      tr[sa[i]] = tr[sa[i - 1]] + (r[sa[i]] != r[sa[i - 1]] || r[sa[i] + l] != r[sa[
        i - 1] + l]);
    }
    for (int i = 1; i <= n; ++i) r[i] = tr[i];
    if (r[sa[n]] == n) break;
  }
  int k = 0; /* Height && RMQ */
  for (int i = 1; i <= n; ++i) {
    if (--k < 0) k = 0;
    for (int j = sa[r[i] - 1]; r[i] != 1 && s[i + k] == s[j + k]; ++k);
    h[r[i]] = k;
  }
  for (int i = 1; i <= n; ++i) mv[i][0] = h[i];
  for (int k = 1; k < logn; ++k) {
    for (int i = 1, len = 1 << (k - 1); i + len <= n; ++i) {
      mv[i][k] = min(mv[i][k - 1], mv[i + len][k - 1]);
    }
  }
}

//faster algorithm
void da(int *r,int *sa,int n,int m) //r[n] = 0!!
{
  int i,j,p,*x=wa,*y=wb,*t;
  for(i=0;i<m;i++) ws[i]=0;
  for(i=0;i<n;i++) ws[x[i]=r[i]]++;
  for(i=1;i<m;i++) ws[i]+=ws[i-1];
  for(i=n-1;i>=0;i--) sa[--ws[x[i]]]=i;
  for(j=1,p=1;p<n;j*=2,m=p)
  {
    for(p=0,i=n-j;i<n;i++) y[p++]=i;
    for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
    for(i=0;i<n;i++) wv[i]=x[y[i]];
    for(i=0;i<m;i++) ws[i]=0;
    for(i=0;i<n;i++) ws[wv[i]]++;
    for(i=1;i<m;i++) ws[i]+=ws[i-1];
    for(i=n-1;i>=0;i--) sa[--ws[wv[i]]]=y[i];
    t=x; x=y; y=t;
    p = 1;
    x[sa[0]] = 0;
    for (i = 1; i < n; i++)
      x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + j] == y[sa[i - 1] + j]) ? p
        -1 : p++;
  }
  return;
}

int askRMQ(int l, int r) {
  int len = r - l + 1, log = _lg[r - l + 1];
  return min(mv[l][log], mv[r - (1 << log) + 1][log]);
}

int LCP(int i, int j) {
  i = r[i], j = r[j];
  if (i > j) swap(i, j);
  return askRMQ(++i, j);
}
```

3 Math

3.1 Matrix Multiplication

```
struct matrix_t {
  int x[N + 1][N + 1];
  matrix_t(int v) {
    memset(x, 0, sizeof(x));
    for (int i = 1; i <= N; ++i) x[i][i] = v;
  }
  matrix_t operator*(const matrix_t &r) {
    matrix_t p = 0;
    for (int k = 1; k <= N; ++k) {
      for (int i = 1; i <= N; ++i) {
        if (x[i][k] == 0) continue;
        for (int j = 1; j <= N; ++j) {
          p.x[i][j] += x[i][k] * r.x[k][j];
          p.x[i][j] %= MOD;
        }
      }
    }
    return p;
  }
  matrix_t power(LL p) {
    matrix_t r = 1, a = *this;
    for (; p; p >>= 1) {
      if (p & 1) r = r * a;
      a = a * a;
    }
    return r;
  }
};
```

Optimization of recursion matrix:
$h_n = a_1 h_{n-1} + a_2 h_{n-2} + a_3 h_{n-3} + \ldots + a_k h_{n-k}$, Construct matrix of $k * k$:

$$\mathbf{M} = \begin{bmatrix} a_1 & a_2 & a_3 & \cdots & a_{k-2} & a_{k-1} & a_k \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix}$$

Then the characteristic polynomial of $\mathbf{M}$ is

$$f(\lambda) = |\lambda\mathbf{E} - \mathbf{M}| = \begin{bmatrix} \lambda - a_1 & -a_2 & -a_3 & \cdots & -a_{k-2} & -a_{k-1} & -a_k \\ -1 & \lambda & 0 & \cdots & 0 & 0 & 0 \\ 0 & -1 & \lambda & \cdots & 0 & 0 & 0 \\ 0 & 0 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & \lambda & 0 \\ 0 & 0 & 0 & \cdots & 0 & -1 & \lambda \end{bmatrix}$$
$$= \lambda^k - a_1\lambda^{k-1} - a_2\lambda^{k-2} - \ldots - a_k.$$

Apply Hamilton-Cayley theorem, we have $f(\mathbf{M}) = \mathbf{0}$.
And, $\forall i$, $\mathbf{M}^i$ can be written as a linear combination of $\mathbf{E}$,$\mathbf{M}$,$\mathbf{M}^2$,$\ldots$,$\mathbf{M}^{k-1}$.

So the matrix multiplication is reduced to polynomial multiplication, which can be computed in $O(n^2)$.

3.2 Gauss Elimiation

```
void gauss(int n, double g[maxn][maxn]) { // input: N * (N + 1) Matrix
  for (int i = 1; i <= n; ++i) {
    double temp = 0;
    int pos = -1;
    for (int j = i; j <= n; ++j) {
      if (fabs(g[j][i]) > temp) temp = fabs(g[j][i]), pos = j;
    }
    if (pos == -1) continue;
    for (int k = 1; k <= n + 1; ++k) swap(g[pos][k], g[i][k]);
    temp = g[i][i];
    for (int k = 1; k <= n + 1; ++k) g[i][k] /= temp;
    for (int j = i + 1; j <= n; ++j) {
      temp = g[j][i];
      for (int k = 1; k <= n + 1; ++k) g[j][k] -= temp * g[i][k];
    }
  }
  for (int i = n; i >= 1; --i) {
    for (int j = 1; j < i; ++j) {
      g[j][n + 1] -= g[i][n + 1] * g[j][i];
      g[j][i] = 0;
    }
  }
}
```

3.3 Determinant

```
LL determinant() {
  LL result = 1;
  for (int i = 1; i <= n; ++i) {
    for (int j = i + 1; j <= n; ++j) {
      while (det[j][i]) {
        LL ratio = det[i][i] / det[j][i];
        for (int k = i; k <= n; ++k) {
          det[i][k] -= ratio * det[j][k];
          swap(det[i][k], det[j][k]);
        }
        result = -result;
      }
    }
    result = result * det[i][i];
  }
  return result;
}
```

Laplacian matrix $L = (\ell_{i,j})_{n*n}$ is defined as: $L = D - A$, that is, it is the difference of the degree matrix D and the adjacency matrix A of the graph.
From the definition it follows that:

$$\ell_{i,j} = \begin{cases} deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Then the number of spanning trees of a graph on n vertices is the determinant of any $n-1$ submatrix of L.

3.4 Polynomial Root

```
double cal(const vector<double> &coef, double x) {
  double e = 1, s = 0;
  for (int i = 0; i < coef.size(); ++i) s += coef[i] * e, e *= x;
  return s;
}

double find(const vector<double> &coef, double l, double r) {
  int sl = dblcmp(cal(coef, l)), sr = dblcmp(cal(coef, r));
  if (sl == 0) return l;
  if (sr == 0) return r;
  if (sl * sr > 0) return maxdbl;
  for (int tt = 0; tt < 100 && r - l > eps; ++tt) {
    double mid = (l + r) / 2;
    int smid = dblcmp(cal(coef, mid));
    if (smid == 0) return mid;
    if (sl * smid < 0) r = mid;
    else l = mid;
  }
  return (l + r) / 2;
}

vector<double> solve(vector<double> coef, int n) {
  vector<double> ret; // c[0]+c[1]*x+c[2]*x^2+...+c[n]*x^n
  if (n == 1) {
    if (dblcmp(coef[1]) != 0) ret.push_back(-coef[0] / coef[1]);
    return ret;
  }
  vector<double> dcoef(n);
  for (int i = 0; i < n; ++i) dcoef[i] = coef[i + 1] * (i + 1);
  vector<double> droot = solve(dcoef, n - 1);
  droot.insert(droot.begin(), -maxdbl);
  droot.push_back(maxdbl);
  for (int i = 0; i + 1 < droot.size(); ++i) {
    double tmp = find(coef, droot[i], droot[i + 1]);
    if (tmp < maxdbl) ret.push_back(tmp);
  }
  return ret;
}
```

3.5 Number Theory Library

```
LL mult64(LL a, LL b, LL m) { // 64bit multiply 64bit
  a %= m, b %= m;
  LL ret = 0;
  for (; b; b >>= 1) {
    if (b & 1) ret = (ret + a) % m;
    a = (a + a) % m;
  }
  return ret;
}

LL fpow(LL a, LL p, int mod) { // fast power-modulo algorithm
  LL res = 1;
  for (; p; p >>= 1) {
    if (p & 1) res = (res * a) % mod; // using mult64 when mod is 64-bit
    a = (a * a) % mod;
  }
  return res;
}

int exgcd(int x, int y, int &a, int &b) { // extended gcd, ax + by = g.
  int a0 = 1, a1 = 0, b0 = 0, b1 = 1;
  while (y != 0) {
    a0 -= x / y * a1; swap(a0, a1);
    b0 -= x / y * b1; swap(b0, b1);
    x %= y; swap(x, y);
  }
  if (x < 0) a0 = -a0, b0 = -b0, x = -x;
  a = a0, b = b0;
  return x;
}

int inverse(int x, int mod) { // multiplicative inverse.
  int a = 0, b = 0;
  if (exgcd(x, mod, a, b) != 1) return -1;
  return (a % mod + mod) % mod; // C1: x & mod are co-prime
  return fpow(x, mod - 2, mod); // C2: mod is prime
}

void init_inverse(int mod) { // O(n), all multiplicative inverse, mod is prime
  inv[0] = inv[1] = 1;
  for (int i = 2; i < n; ++i) {
    inv[i] = (LL)inv[mod % i] * (mod - mod / i) % mod; // overflows?
  }
}

LL CRT(int cnt, int *p, int *b) { // chinese remainder theorem
  LL N = 1, ans = 0;
  for (int i = 0; i < k; ++i) N *= p[i];
  for (int i = 0; i < k; ++i) {
    LL mult = (inverse(N / p[i], p[i]) * (N / p[i])) % N;
    mult = (mult * b[i]) % N;
    ans += mult; ans %= N;
  }
  if (ans < 0) ans += N;
  return ans;
}

void sieve(int n) { // generating primes using euler's sieve
  notP[1] = 1;
  for (int i = 2; i <= n; ++i) {
    if (!notP[i]) P[++Pt] = i;
    for (int j = 1; j <= Pt && P[j] * i <= n; ++j) {
      notP[P[j] * i] = 1;
      if (i % P[j] == 0) break;
    }
  }
}

bool miller_rabin(LL n, LL b) { // miller-rabin prime test
  LL m = n - 1, cnt = 0;
  while (m % 2 == 0) m >>= 1, ++cnt;
  LL ret = fpow(b, m, n);
  if (ret == 1 || ret == n - 1) return true;
  --cnt;
  while (cnt >= 0) {
    ret = mult(ret, ret, n);
    if (ret == n - 1) return true;
    --cnt;
  }
  return false;
}

bool prime_test(LL n) {
  if (n < 2) return false;
  if (n < 4) return true;
  if (n == 3215031751LL) return false;
  const int BASIC[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
  for (int i = 0; i < 12 && BASIC[i] < n; ++ i) {
    if (!miller_rabin(n, BASIC[i])) return false;
  }
  return true;
}

LL pollard_rho(LL n, LL seed) { // pollard-rho divisors factorization
  LL x, y;
  x = y = rand() % (n - 1) + 1;
  LL head = 1, tail = 2;
  while (true) {
    x = mult(x, x, n);
    x = (x + seed) % n;
    if (x == y) return n;
    LL d = gcd(max(x - y, y - x), n);
    if (1 < d && d < n) return d;
    if (++head == tail) y = x, tail <<= 1;
  }
}

void factorize(LL n, vector<LL> &divisor) {
  if (n == 1) return;
  if (prime_test(n)) divisor.push_back(n);
  else {
    LL d = n;
    while (d >= n) d = pollard_rho(n, rand() % (n - 1) + 1);
    factorize(n / d, divisor);
    factorize(d, divisor);
  }
}

// discrete-logarithm, finding y for equation k = x^y % mod
int discrete_logarithm(int x, int mod, int k) {
  if (mod == 1) return 0;
  int s = 1, g;
  for (int i = 0; i < 64; ++i) {
    if (s == k) return i;
    s = ((LL)s * x) % mod;
  }
  while ((g = gcd(x, mod)) != 1) {
    if (k % g) return -1;
    mod /= g;
  }
  static map<int, int> M; M.clear();
  int q = int(sqrt(double(euler(mod)))) + 1;
  for (int i = 0, b = 1; i < q; ++i) {
    if (M.find(b) == M.end()) M[b] = i;
    b = ((LL)b * x) % mod;
  }
  int p = fpow(x, q, mod);
```

```cpp
  for (int i = 0, b = 1; i <= q; ++i) {
    int v = ((LL)k * inverse(b, mod)) % mod;
    if (M.find(v) != M.end()) {
      int y = i * q + M[v];
      if (y >= 64) return y;
    }
    b = ((LL)b * p) % mod;
  }
  return -1;
}

// primtive root, finding the number with order p-1
int primtive_root(int p) {
  vector<int> factor;
  int tmp = p - 1;
  for (int i = 2; i * i <= tmp; ++i) {
    if (tmp % i == 0) {
      factor.push_back(i);
      while (tmp % i == 0) tmp /= i;
    }
  }
  if (tmp != 1) factor.push_back(tmp);
  for (int root = 1; ; ++root) {
    bool flag = true;
    for (int i = 0; i < factor.size(); ++i) {
      if (fpow(root, (p - 1) / factor[i], p) == 1) {
        flag = false;
        break;
      }
    }
    if (flag) return root;
  }
}
```

3.6 Number Partition

```cpp
// number of ways to divide n to integers(unordered), O(n^(3/2))
int partition(int n) {
  int dp[n + 1];
  dp[0] = 1;
  for (int i = 1; i <= n; i++) {
    dp[i] = 0;
    for (int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; ++j, r *= -1) {
      dp[i] += dp[i - (3 * j * j - j) / 2] * r;
      if (i - (3 * j * j + j) / 2 >= 0) dp[i] += dp[i - (3 * j * j + j) / 2] * r;
    }
  }
  return dp[n];
}
```

4 Computational Geometry

4.1 Common 2D

```cpp
// implementation of (dblcmp,dist,cross,dot) is trivial

// count-clock wise is positive direction
double angle(point_t p1, point_t p2) {
  double x1 = p1.x, y1 = p1.y, x2 = p2.x, y2 = p2.y;
  double a1 = atan2(y1, x1), a2 = atan2(y2, x2);
  double a = a2 - a1;
  while (a < -pi) a += 2 * pi;
  while (a >= pi) a -= 2 * pi;
  return a;
}

bool onSeg(point_t p, point_t a, point_t b) {
  return dblcmp(cross(a - p, b - p)) == 0 && dblcmp(dot(a - p, b - p)) <= 0;
}

// 1 normal intersected, -1 denormal intersected, 0 not intersected
int testSS(point_t a, point_t b, point_t c, point_t d) {
  if (dblcmp(max(a.x, b.x) - min(c.x, d.x)) < 0) return 0;
  if (dblcmp(max(c.x, d.x) - min(a.x, b.x)) < 0) return 0;
  if (dblcmp(max(a.y, b.y) - min(c.y, d.y)) < 0) return 0;
  if (dblcmp(max(c.y, d.y) - min(a.y, b.y)) < 0) return 0;
  int d1 = dblcmp(cross(c - a, b - a));
  int d2 = dblcmp(cross(d - a, b - a));
  int d3 = dblcmp(cross(a - c, d - c));
  int d4 = dblcmp(cross(b - c, d - c));
  if ((d1 * d2 < 0) && (d3 * d4 < 0)) return 1;
  if ((d1 * d2 <= 0 && d3 * d4 == 0) || (d1 * d2 == 0 && d3 * d4 <= 0)) return -1;
  return 0;
}

vector<point_t> isLL(point_t a, point_t b, point_t c, point_t d) {
  point_t p1 = b - a, p2 = d - c;
  vector<point_t> ret;
  double a1 = p1.y, b1 = -p1.x, c1;
  double a2 = p2.y, b2 = -p2.x, c2;
  if (dblcmp(a1 * b2 - a2 * b1) == 0) return ret; // colined <=> a1*c2-a2*c1=0 && b1
    *c2-b2*c1=0
  else {
    c1 = a1 * a.x + b1 * a.y;
    c2 = a2 * c.x + b2 * c.y;
    ret.push_back(point_t((c1 * b2 - c2 * b1) / (a1 * b2 - a2 * b1), (c1 * a2 - c2 *
       a1) / (b1 * a2 - b2 * a1)));
    return ret;
  }
}

point_t angle_bisector(point_t p0, point_t p1, point_t p2) {
  point_t v1 = p1 - p0, v2 = p2 - p0;
  v1 = v1 / dist(v1) * dist(v2);
  return v1 + v2 + p0;
}

point_t perpendicular_bisector(point_t p1, point_t p2) {
  point_t v = p2 - p1;
  swap(v.x, v.y);
  v.x = -v.x;
  return v + (p1 + p2) / 2;
}

point_t circumcenter(point_t p0, point_t p1, point_t p2) {
  point_t v1 = perpendicular_bisector(p0, p1);
  point_t v2 = perpendicular_bisector(p1, p2);
  return isLL((p0 + p1) / 2, v1, (p1 + p2) / 2, v2);
}

point_t incenter(point_t p0, point_t p1, point_t p2) {
  point_t v1 = angle_bisector(p0, p1, p2);
  point_t v2 = angle_bisector(p1, p2, p0);
  return isLL(p0, v1, p1, v2);
}

point_t orthocenter(point_t p0, point_t p1, point_t p2) {
  return p0 + p1 + p2 - circumcenter(p0, p1, p2) * 2;
}

// count-clock wise is positive direction
point_t rotate(point_t p, double a) {
  double s = sin(a), c = cos(a);
  return point_t(p.x * c - p.y * s, p.y * c + p.x * s);
}

bool insidePoly(point_t *p, int n, point_t t) {
  p[0] = p[n];
  for (int i = 0; i < n; ++i) if (onSeg(t, p[i], p[i + 1])) return true;
  point_t r = point_t(2353456.663, 5326546.243); // random point
  int cnt = 0;
  for (int i = 0; i < n; ++i) {
    if (testSS(t, r, p[i], p[i + 1]) != 0) ++cnt;
  }
  return cnt & 1;
}

bool insideConvex(point_t *convex, int n, point_t t) { // O(logN), convex polygen,
   cross(p[2] - p[1], p[3] - p[1]) > 0
  if (n == 2) return onSeg(t, convex[1], convex[2]);
  int l = 2, r = n;
  while (l < r) {
    int mid = (l + r) / 2 + 1;
    int side = dblcmp(cross(convex[mid] - convex[1], t - convex[1]));
    if (side == 1) l = mid;
    else r = mid - 1;
  }
  int s = dblcmp(cross(convex[l] - convex[1], t - convex[1]));
  if (s == -1 || l == n) return false;
  point_t v = convex[l + 1] - convex[l];
  if (dblcmp(cross(v, t - convex[l])) >= 0) return true;
  return false;
}
```

4.2 Graham Convex Hull

```
bool cmp(const point_t p1, const point_t p2) {
  return dblcmp(p1.y - p2.y) == 0 ? p1.x < p2.x : p1.y < p2.y;
}

int graham(point_t *p) { // Points co-lined are ignored.
  int top = 2; static point_t sk[maxn];
  sort(p + 1, p + 1 + n, cmp);
  sk[1] = p[1], sk[2] = p[2];
  for (int i = 3; i <= n; ++i) {
    while (top >= 2 && dblcmp(cross(p[i] - sk[top - 1], sk[top] - sk[top - 1])) >=
       0) --top;
    sk[++top] = p[i];
  }
  int ttop = top;
  for (int i = n - 1; i >= 1; --i) {
    while (top > ttop && dblcmp(cross(p[i] - sk[top - 1], sk[top] - sk[top - 1])) >=
       0) --top;
    sk[++top] = p[i];
  }
  for (int i = 1; i < top; ++i) p[i] = sk[i];
  return --top;
}
```

4.3 Minkowski Sum of Convex Hull

Wiki:
The Minkowski sum of two sets of position vectors A and B in Euclidean space is formed by adding each vector in A to each vector in B, i.e. the set

$$A + B = \{\vec{a} + \vec{b} \,|\, \vec{a} \in A, \vec{b} \in B\}.$$

For all subsets S_1 and S_2 of a real vector-space, the convex hull of their Minkowski sum is the Minkowski sum of their convex hulls $\mathrm{Conv}(S_1 + S_2) = \mathrm{Conv}(S_1) + \mathrm{Conv}(S_2)$. Minkowski sums are used in motion planning of an object among obstacles. They are used for the computation of the configuration space, which is the set of all admissible positions of the object. In the simple model of translational motion of an object in the plane, where the position of an object may be uniquely specified by the position of a fixed point of this object, the configuration space are the Minkowski sum of the set of obstacles and the movable object placed at the origin and rotated 180 degrees.

```cpp
int minkowski(point_t *h, point_t *h1, point_t *h2, int n, int m) {
  point_t c = point_t(0, 0);
  for (int i = 1; i <= m; ++i) c = c + h2[i];
  c = c / m;
  for (int i = 1; i <= m; ++i) h2[i] = h2[i] - c;
  int cur = -1;
  for (int i = 1; i <= m; ++i) {
    if (dblcmp(cross(h2[i], h1[1] - h1[n])) >= 0) {
      if (cur == -1 || cross(h2[i], h1[1] - h1[n]) > cross(h2[cur], h1[1] - h1[n]))
        cur = i;
    }
  }
  int cnt = 0;
  h1[n + 1] = h1[1];
  for (int i = 1; i <= n; ++i) {
    while (true) {
      h[++cnt] = h1[i] + h2[cur];
      int next = (cur == m ? 1 : cur + 1);
      if (dblcmp(cross(h2[cur], h1[i + 1] - h1[i])) < 0) cur = next;
      else {
        if (cross(h2[next], h1[i + 1] - h1[i]) > cross(h2[cur], h1[i + 1] - h1[i]))
          cur = next;
        else break;
      }
    }
  }
  for (int i = 1; i <= cnt; ++i) h[i] = h[i] + c;
  for (int i = 1; i <= m; ++i) h2[i] = h2[i] + c;
  return graham(h, cnt);
}
```

4.4 Rotating Calipers

```cpp
// Calculate the maximum distance of a point set.
double rotate_caliper() {
  p[0] = p[n];
  int to = 0;
  double ans = 0;
  for (int i = 0; i < n; ++i) {
    while ((to + 1) % n != i) {
      if (dblcmp(cross(p[i + 1] - p[i], p[to + 1] - p[i]) - cross(p[i + 1] - p[i], p
        [to] - p[i])) >= 0) to = (to + 1) % n;
      else break;
    }
    ans = max(ans, dist(p[i], p[to]));
    ans = max(ans, dist(p[i + 1], p[to]));
  }
  return ans;
}
```

4.5 Closest Pair Points

```cpp
double dac(point_t *p, int l, int r) {
  double d = 10e100;
  if (r - l <= 3) {
    for (int i = l; i <= r; ++i) {
      for (int j = i + 1; j <= r; ++j) {
        d = min(d, dist2(p[i], p[j]));
      }
    }
    sort(p + l, p + r + 1, cmpY);
  } else {
    int mid = (l + r) / 2;
    d = min(dac(p, l, mid), dac(p, mid + 1, r));
    inplace_merge(p + l, p + mid + 1, p + r + 1, cmpY);
    static point_t tmp[maxn]; int cnt = 0;
    for (int i = l; i <= r; ++i) {
      if ((p[i].x - p[mid].x) * (p[i].x - p[mid].x) <= d) tmp[++cnt] = p[i];
    }
    for (int i = 1; i <= cnt; ++i) {
      for (int j = 1; j <= 8 && j + i <= cnt; ++j) {
        d = min(d, dist2(tmp[i], tmp[j + i]));
      }
    }
  }
  return d;
}

double cal(point_t *p, int n) {
  sort(p + 1, p + 1 + n, cmpX);
  return sqrt(dac(p, 1, n));
}
```

4.6 Halfplane Intersection

```
// O(N^2) sol, polygon counterclockwise order
// i.e., left side of vector v1->v2 is the valid half plane
const double maxd = 1e5;
int n, cnt;
point_t p[maxn];

void init() { // order reversed if right side
  cnt = 4;
  p[1] = point_t(-maxd, -maxd);
  p[2] = point_t(maxd, -maxd);
  p[3] = point_t(maxd, maxd);
  p[4] = point_t(-maxd, maxd);
}

void cut(point_t p1, point_t p2) {
  int tcnt = 0;
  static point_t tp[maxn];
  p[cnt + 1] = p[1];
  for (int i = 1; i <= cnt; ++i) {
    double v1 = cross(p2 - p1, p[i] - p1);
    double v2 = cross(p2 - p1, p[i + 1] - p1);
    if (dblcmp(v1) >= 0) tp[++tcnt] = p[i]; // <= if right side
    if (dblcmp(v1) * dblcmp(v2) < 0) tp[++tcnt] = isLL(p1, p2, p[i], p[i + 1]);
  }
  cnt = tcnt;
  for (int i = 1; i <= cnt; ++i) p[i] = tp[i];
}
```

```
// O(NlogN) sol, Left is valid half plane. Note that the edge of hull may degenerate
   to a point.
struct hp_t {
  point_t p1, p2;
  double a;
  hp_t() { }
  hp_t(point_t tp1, point_t tp2) : p1(tp1), p2(tp2) {
    tp2 = tp2 - tp1;
    a = atan2(tp2.y, tp2.x);
  }
  bool operator==(const hp_t &r) const {
    return dblcmp(a - r.a) == 0;
  }
  bool operator<(const hp_t &r) const {
    if (dblcmp(a - r.a) == 0) return dblcmp(cross(r.p2 - r.p1, p2 - r.p1)) >= 0;
    else return a < r.a;
  }
} hp[maxn];

void addhp(point_t p1, point_t p2) {
  hp[++cnt] = hp_t(p1, p2);
}

void init() {
  cnt = 0;
  addhp(point_t(-maxd, -maxd), point_t(maxd, -maxd));
  addhp(point_t(maxd, -maxd), point_t(maxd, maxd));
  addhp(point_t(maxd, maxd), point_t(-maxd, maxd));
  addhp(point_t(-maxd, maxd), point_t(-maxd, -maxd));
}

bool checkhp(hp_t h1, hp_t h2, hp_t h3) {
  point_t p = isLL(h1.p1, h1.p2, h2.p1, h2.p2);
  return dblcmp(cross(p - h3.p1, h3.p2 - h3.p1)) > 0;
}

vector<point_t> hp_inter() {
  sort(hp + 1, hp + 1 + cnt);
  cnt = unique(hp + 1, hp + 1 + cnt) - hp - 1;
  deque<hp_t> DQ;
  DQ.push_back(hp[1]);
  DQ.push_back(hp[2]);
  for (int i = 3; i <= cnt; ++i) {
    while (DQ.size() > 1 && checkhp(*----DQ.end(), *--DQ.end(), hp[i])) DQ.pop_back
      ();
    while (DQ.size() > 1 && checkhp(*++DQ.begin(), *DQ.begin(), hp[i])) DQ.pop_front
      ();
    DQ.push_back(hp[i]);
  }
  while (DQ.size() > 1 && checkhp(*----DQ.end(), *--DQ.end(), DQ.front())) DQ.
    pop_back();
  while (DQ.size() > 1 && checkhp(*++DQ.begin(), *DQ.begin(), DQ.back())) DQ.
    pop_front();
  DQ.push_front(DQ.back());
  vector<point_t> res;
  while (DQ.size() > 1) {
    hp_t tmp = DQ.front();
    DQ.pop_front();
    res.push_back(isLL(tmp.p1, tmp.p2, DQ.front().p1, DQ.front().p2));
  }
  return res;
}
```

4.7 Tri-Cir Intersection & Tangent

```cpp
vector<point_t> tanCP(point_t c, double r, point_t p) {
  double x = dot(p - c, p - c);
  double d = x - r * r;
  vector<point_t> res;
  if (d < -eps) return res;
  if (d < 0) d = 0;
  point_t q1 = (p - c) * (r * r / x);
  point_t q2 = ((p - c) * (-r * sqrt(d) / x)).rot90(); // rot90: (-y, x)
  res.push_back(c + q1 - q2);
  res.push_back(c + q1 + q2);
  return res;
}

vector<seg_t> tanCC(point_t c1, double r1, point_t c2, double r2) {
  vector<seg_t> res;
  if (abs(r1 - r2) < eps) {
    point_t dir = c2 - c1;
    dir = (dir * (r1 / dir.l())).rot90();
    res.push_back(seg_t(c1 + dir, c2 + dir));
    res.push_back(seg_t(c1 - dir, c2 - dir));
  } else {
    point_t p = ((c1 * -r2) + (c2 * r1)) / (r1 - r2);
    vector<point_t> ps = tanCP(c1, r1, p), qs = tanCP(c2, r2, p);
    for (int i = 0; i < ps.size() && i < qs.size(); ++i) {
      res.push_back(seg_t(ps[i], qs[i]));
    }
  }
  point_t p = ((c1 * r2) + (c2 * r1)) / (r1 + r2);
  vector<point_t> ps = tanCP(c1, r1, p), qs = tanCP(c2, r2, p);
  // point_t tmp = (c2 - c1).rot90().rot90().rot90();
  for (int i = 0; i < ps.size() && i < qs.size(); ++i) {
    /* if (dblcmp(dist(ps[i], qs[i])) == 0) {
      qs[i] = qs[i] + tmp;
      tmp = tmp.rot90().rot90();
    }*/
    res.push_back(seg_t(ps[i], qs[i]));
  }
  return res;
}
```

```cpp
// Assume d <= r1 + r2 && d >= |r1 - r2|
pair<point_t, point_t> isCC(point_t c1, point_t c2, double r1, double r2) {
  if (r1 < r2) swap(c1, c2), swap(r1, r2);
  double d = dist(c1, c2);
  double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
  double mid = atan2(y2 - y1, x2 - x1);
  double a = r1, c = r2;
  double t = acos(max(0.0, a * a + d * d - c * c) / (2 * a * d));
  point_t p1 = point_t(cos(mid - t) * r1, sin(mid - t) * r1) + c1;
  point_t p2 = point_t(cos(mid + t) * r1, sin(mid + t) * r1) + c1;
  return make_pair(p1, p2);
}

int testCC(point_t c1, point_t c2, double r1, double r2) {
  double d = dist(c1, c2);
  if (dblcmp(r1 + r2 - d) <= 0) return 1; // not intersected or tged
  if (dblcmp(r1 + d - r2) <= 0) return 2; // C1 inside C2
  if (dblcmp(r2 + d - r1) <= 0) return 3; // C2 inside C1
  return 0; // intersected
}

point_t isCL(point_t a, point_t b, point_t o, double r) {
  double x0 = o.x, y0 = o.y;
  double x1 = a.x, y1 = a.y;
  double x2 = b.x, y2 = b.y;
  double dx = x2 - x1, dy = y2 - y1;
  double A = dx * dx + dy * dy;
  double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
  double C = (x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 - y0) - r * r;
  double delta = B * B - 4 * A * C;
  if (delta >= 0) {
    delta = sqrt(delta);
    double t1 = (-B - delta) / 2 / A;
    double t2 = (-B + delta) / 2 / A;
    if (dblcmp(t1) >= 0) return point_t(x1 + t1 * dx, y1 + t1 * dy); // Ray
    if (dblcmp(t2) >= 0) return point_t(x1 + t2 * dx, y1 + t2 * dy);
  }
  return point_t();
}
```

```cpp
double areaTC(point_t ct, double r, point_t p1, point_t p2) { // intersected area
  double a, b, c, x, y, s = cross(p1 - ct, p2 - ct) / 2;
  a = dist(ct, p2), b = dist(ct, p1), c = dist(p1, p2);
  if (a <= r && b <= r) {
    return s;
  } else if (a < r && b >= r) {
    x = (dot(p1 - p2, ct - p2) + sqrt(c * c * r * r - sqr(cross(p1 - p2, ct - p2))))
      / c;
    return asin(s * (c - x) * 2 / c / b / r) * r * r / 2 + s * x / c;
  } else if (a >= r && b < r) {
    y = (dot(p2 - p1, ct - p1) + sqrt(c * c * r * r - sqr(cross(p2 - p1, ct - p1))))
      / c;
    return asin(s * (c - y) * 2 / c / a / r) * r * r / 2 + s * y / c;
  } else {
    if (fabs(2 * s) >= r * c || dot(p2 - p1, ct - p1) <= 0 || dot(p1 - p2, ct - p2)
      <= 0) {
```

```
      if (dot(p1 - ct, p2 - ct) < 0) {
        if (cross(p1 - ct, p2 - ct) < 0) {
          return (-pi - asin(s * 2 / a / b)) * r * r / 2;
        } else {
          return (pi - asin(s * 2 / a / b)) * r * r / 2;
        }
      } else {
        return asin(s * 2 / a / b) * r * r / 2;
      }
    } else {
      x = (dot(p1 - p2, ct - p2) + sqrt(c * c * r * r - sqr(cross(p1 - p2, ct - p2))
        )) / c;
      y = (dot(p2 - p1, ct - p1) + sqrt(c * c * r * r - sqr(cross(p2 - p1, ct - p1))
        )) / c;
      return (asin(s * (1 - x / c) * 2 / r / b) + asin(s * (1 - y / c) * 2 / r / a))
         * r * r / 2 + s * ((y + x) / c - 1);
    }
  }
}

double areaTC(point_t ct, double r, point_t p1, point_t p2, point_t p3) {
  return areaTC(ct, r, p1, p2) + areaTC(ct, r, p2, p3) + areaTC(ct, r, p3, p1);
}
```

4.8 Circle Area Union

```
/* O(n^2logn), please remove coincided circles first. */
point_t center[maxn];
double radius[maxn], cntarea[maxn];

pair<double, double> isCC(point_t c1, point_t c2, double r1, double r2) {
  double d = dist(c1, c2);
  double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
  double mid = atan2(y2 - y1, x2 - x1);
  double a = r1, c = r2;
  double t = acos((a * a + d * d - c * c) / (2 * a * d));
  return make_pair(mid - t, mid + t);
}

struct event_t {
  double theta;
  int delta;
  event_t(double t, int d) : theta(t), delta(d) { }
  bool operator<(const event_t &r) const {
    if (fabs(theta - r.theta) < eps) return delta > r.delta;
    return theta < r.theta;
  }
};
vector<event_t> e;

void add(double begin, double end) {
  if (begin <= -pi) begin += 2 * pi, end += 2 * pi;
  if (end > pi) {
    e.push_back(event_t(begin, 1));
    e.push_back(event_t(pi, -1));
    e.push_back(event_t(-pi, 1));
    e.push_back(event_t(end - 2 * pi, -1));
  } else {
    e.push_back(event_t(begin, 1));
    e.push_back(event_t(end, -1));
  }
}

double calc(point_t c, double r, double a1, double a2) {
  double da = a2 - a1;
  double aa = r * r * (da - sin(da)) / 2;
  point_t p1 = point_t(cos(a1), sin(a1)) * r + c;
  point_t p2 = point_t(cos(a2), sin(a2)) * r + c;
  return cross(p1, p2) / 2 + aa;
}

void circle_union() {
  for (int c = 1; c <= n; ++c) {
    int cvrcnt = 0;
    e.clear();
    for (int i = 1; i <= n; ++i) {
      if (i != c) {
        int r = testCC(center[c], center[i], radius[c], radius[i]);
        if (r == 2) ++cvrcnt;
        else if (r == 0) {
          pair<double, double> paa = isCC(center[c], center[i], radius[c], radius[i
            ]);
          add(paa.first, paa.second);
        }
      }
    }
    if (e.size() == 0) {
      double a = pi * radius[c] * radius[c];
      cntarea[cvrcnt] -= a;
      cntarea[cvrcnt + 1] += a;
    } else {
      e.push_back(event_t(-pi, 1));
      e.push_back(event_t(pi, -2));
      sort(e.begin(), e.end());
      for (int i = 0; i < int(e.size()) - 1; ++i) {
        cvrcnt += e[i].delta;
        double a = calc(center[c], radius[c], e[i].theta, e[i + 1].theta);
        cntarea[cvrcnt - 1] -= a;
        cntarea[cvrcnt] += a;
      }
    }
  }
}
```

4.9 Minimum Covering Circle

```
void set_circle(point_t &p, double &r, point_t a, point_t b) {
  r = dist(a, b) / 2;
  p = (a + b) / 2;
}

void set_circle(point_t &p, double &r, point_t a, point_t b, point_t c) {
  if (dblcmp(cross(b - a, c - a)) == 0) {
    if (dist(a, c) > dist(b, c)) {
      r = dist(a, c) / 2;
      p = (a + c) / 2;
    } else {
      r = dist(b, c) / 2;
      p = (b + c) / 2;
    }
  } else {
    p = circumcenter(a, b, c);
    r = dist(p, a);
  }
}

bool in_circle(point_t &p, double &r, point_t x) {
  return dblcmp(dist(x, p) - r) <= 0;
}

pair<point_t, double> minimum_circle(int n, point_t *p) {
  point_t c = point_t(0, 0);
  double r = 0;
  random_shuffle(p + 1, p + 1 + n);
  set_circle(c, r, p[1], p[2]);
  for (int i = 3; i <= n; ++i) {
    if (in_circle(c, r, p[i])) continue;
    set_circle(c, r, p[i], p[1]);
    for (int j = 2; j < i; ++j) {
      if (in_circle(c, r, p[j])) continue;
      set_circle(c, r, p[i], p[j]);
      for (int k = 1; k < j; ++k) {
        if (in_circle(c, r, p[k])) continue;
        set_circle(c, r, p[i], p[j], p[k]);
      }
    }
  }
  return make_pair(c, r);
}
```

4.10 Convex Polygon Area Union

```
// modified from syntax_error's code
bool operator<(const point_t &a, const point_t &b) {
  if (dblcmp(a.x - b.x) == 0) return a.y < b.y;
  return a.x < b.x;
}

bool operator==(const point_t &a, const point_t &b) {
  return dblcmp(a.x - b.x) == 0 && dblcmp(a.y - b.y) == 0;
}

struct segment_t {
  point_t a, b;
  segment_t() { a = b = point_t(); }
  segment_t(point_t ta, point_t tb) : a(ta), b(tb) { }
  double len() const { return dist(a, b); }
  double k() const { return (a.y - b.y) / (a.x - b.x); }
  double l() const { return a.y - k() * a.x; }
};

struct line_t {
  double a, b, c;
  line_t(point_t p) { a = p.x, b = -1.0, c = -p.y; }
  line_t(point_t p, point_t q) {
    a = p.y - q.y;
    b = q.x - p.x;
    c = a * p.x + b * p.y;
  }
};

bool ccutl(line_t p, line_t q) {
  if (dblcmp(p.a * q.b - q.a * p.b) == 0) return false;
  return true;
}

point_t cutl(line_t p, line_t q) {
  double x = (p.c * q.b - q.c * p.b) / (p.a * q.b - q.a * p.b);
  double y = (p.c * q.a - q.c * p.a) / (p.b * q.a - q.b * p.a);
  return point_t(x, y);
}

```

```
bool onseg(point_t p, segment_t s) {
  if (dblcmp(p.x - min(s.a.x, s.b.x)) < 0 || dblcmp(p.x - max(s.a.x, s.b.x)) > 0)
    return false;
  if (dblcmp(p.y - min(s.a.y, s.b.y)) < 0 || dblcmp(p.y - max(s.a.y, s.b.y)) > 0)
    return false;
  return true;
}

bool ccut(segment_t p, segment_t q) {
  if (!ccutl(line_t(p.a, p.b), line_t(q.a, q.b))) return false;
  point_t r = cutl(line_t(p.a, p.b), line_t(q.a, q.b));
  if (!onseg(r, p) || !onseg(r, q)) return false;
  return true;
}

point_t cut(segment_t p, segment_t q) {
  return cutl(line_t(p.a, p.b), line_t(q.a, q.b));
}

struct event_t {
  double x;
  int type;
  event_t() { x = 0, type = 0; }
  event_t(double _x, int _t) : x(_x), type(_t) { }
  bool operator<(const event_t &r) const {
    return x < r.x;
  }
};

vector<segment_t> s;

double solve(const vector<segment_t> &v, const vector<int> &sl) {
  double ret = 0;
  vector<point_t> lines;
  for (int i = 0; i < v.size(); ++i) lines.push_back(point_t(v[i].k(), v[i].l()));
  sort(lines.begin(), lines.end());
  lines.erase(unique(lines.begin(), lines.end()), lines.end());
  for(int i = 0; i < lines.size(); ++i) {
    vector<event_t> e;
    vector<int>::const_iterator it = sl.begin();
    for(int j = 0; j < s.size(); j += *it++) {
      bool touch = false;
      for (int k = 0; k < *it; ++k) if (lines[i] == point_t(s[j + k].k(), s[j + k].l
        ())) touch = true;
      if (touch) continue;
      vector<point_t> cuts;
      for (int k = 0; k < *it; ++k) {
        if (!ccutl(line_t(lines[i]), line_t(s[j + k].a, s[j + k].b))) continue;
        point_t r = cutl(line_t(lines[i]), line_t(s[j + k].a, s[j + k].b));
        if (onseg(r, s[j + k])) cuts.push_back(r);
      }
      sort(cuts.begin(), cuts.end());
      cuts.erase(unique(cuts.begin(), cuts.end()), cuts.end());
      if (cuts.size() == 2) {
        e.push_back(event_t(cuts[0].x, 0));
        e.push_back(event_t(cuts[1].x, 1));
      }
    }
    for (int j = 0; j < v.size(); ++j) {
      if (lines[i] == point_t(v[j].k(), v[j].l())) {
        e.push_back(event_t(min(v[j].a.x, v[j].b.x), 2));
        e.push_back(event_t(max(v[j].a.x, v[j].b.x), 3));
      }
    }
    sort(e.begin(), e.end());
    double last = e[0].x;
    int cntg = 0, cntb = 0;
    for (int j = 0; j < e.size(); ++j) {
      double y0 = lines[i].x * last + lines[i].y;
      double y1 = lines[i].x * e[j].x + lines[i].y;
      if (cntb == 0 && cntg) ret += (y0 + y1) * (e[j].x - last) / 2;
      last = e[j].x;
      if (e[j].type == 0) ++cntb;
      if (e[j].type == 1) --cntb;
      if (e[j].type == 2) ++cntg;
      if (e[j].type == 3) --cntg;
    }
  }
  return ret;
}

double polyUnion(vector<vector<point_t> > polys) {
  s.clear();
  vector<segment_t> A, B;
  vector<int> sl;
  for (int i = 0; i < polys.size(); ++i) {
    double area = 0;
    int tot = polys[i].size();
    for (int j = 0; j < tot; ++j) {
      area += cross(polys[i][j], polys[i][(j + 1) % tot]);
    }
    if (dblcmp(area) > 0) reverse(polys[i].begin(), polys[i].end());
    if (dblcmp(area) != 0) {
      sl.push_back(tot);
      for (int j = 0; j < tot; ++j) s.push_back(segment_t(polys[i][j], polys[i][(j +
        1) % tot]));
    }
  }
  for (int i = 0; i < s.size(); ++i) {
    int sgn = dblcmp(s[i].a.x - s[i].b.x);
    if (sgn == 0) continue;
    else if (sgn < 0) A.push_back(s[i]);
    else B.push_back(s[i]);
  }
  return solve(A, sl) - solve(B, sl);
}
```

4.11 3D Common

```
double dot(point_t p1, point_t p2) {
  return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
}

point_t cross(point_t p1, point_t p2) {
  return point_t(p1.y * p2.z - p1.z * p2.y, p1.z * p2.x - p1.x * p2.z, p1.x * p2.y -
      p1.y * p2.x);
}

double volume(point_t p1, point_t p2, point_t p3, point_t p4) {
  point_t v1 = cross(p2 - p1, p3 - p1);
  p4 = p4 - p1;
  return dot(v1, p4) / 6;
}

double area(point_t p1, point_t p2, point_t p3) {
  return cross(p2 - p1, p3 - p1).length() / 2;
}

pair<point_t, point_t> isFF(point_t p1, point_t o1, point_t p2, point_t o2) {
  point_t e = cross(o1, o2), v = cross(o1, e);
  double d = dot(o2, v);
  if (fabs(d) < eps) throw -1;
  point_t q = p1 + (v * (dot(o2, p2 - p1) / d));
  return make_pair(q, q + e);
}

double distLL(point_t p1, point_t u, point_t p2, point_t v) {
  double s = dot(u, v) * dot(v, p1 - p2) - dot(v, v) * dot(u, p1 - p2);
  double t = dot(u, u) * dot(v, p1 - p2) - dot(u, v) * dot(u, p1 - p2);
  double deno = dot(u, u) * dot(v, v) - dot(u, v) * dot(u, v);
  if (dblcmp(deno) == 0) return dist(p1, p2 + v * (dot(p1 - p2, u) / dot(u, v)));
  s /= deno; t /= deno;
  point_t a = p1 + u * s, b = p2 + v * t;
  return dist(a, b);
}
```

4.12 3D Convex Hull

```
int n, bf[maxn][maxn], fcnt;
point_t pt[maxn];
struct face_t {
  int a, b, c;
  bool vis;
} fc[maxn << 5]; /* Number of Faces(Unknown) */

bool remove(int p, int b, int a) {
  int f = bf[b][a];
  face_t ff;
  if (fc[f].vis) {
    if (dblcmp(volume(pt[p], pt[fc[f].a], pt[fc[f].b], pt[fc[f].c])) >= 0) {
      return true;
    } else {
      ff.a = a, ff.b = b, ff.c = p;
      bf[ff.a][ff.b] = bf[ff.b][ff.c] = bf[ff.c][ff.a] = ++fcnt;
      ff.vis = true;
      fc[fcnt] = ff;
    }
  }
  return false;
}

void dfs(int p, int f) {
  fc[f].vis = false;
  if (remove(p, fc[f].b, fc[f].a)) dfs(p, bf[fc[f].b][fc[f].a]);
  if (remove(p, fc[f].c, fc[f].b)) dfs(p, bf[fc[f].c][fc[f].b]);
  if (remove(p, fc[f].a, fc[f].c)) dfs(p, bf[fc[f].a][fc[f].c]);
}

void hull3d() {
  for (int i = 2; i <= n; ++i) {
    if (dblcmp((pt[i] - pt[1]).length()) > 0) swap(pt[i], pt[2]);
  }
  for (int i = 3; i <= n; ++i) {
    if (dblcmp(fabs(area(pt[1], pt[2], pt[i]))) > 0) swap(pt[i], pt[3]);
  }
  for (int i = 4; i <= n; ++i) {
    if (dblcmp(fabs(volume(pt[1], pt[2], pt[3], pt[i]))) > 0) swap(pt[i], pt[4]);
  }
  zm(fc), fcnt = 0, zm(bf);
  for (int i = 1; i <= 4; ++i) {
    face_t f;
    f.a = i + 1, f.b = i + 2, f.c = i + 3;
    if (f.a > 4) f.a -= 4;
    if (f.b > 4) f.b -= 4;
    if (f.c > 4) f.c -= 4;
    if (dblcmp(volume(pt[i], pt[f.a], pt[f.b], pt[f.c])) > 0) swap(f.a, f.b);
    f.vis = true;
    bf[f.a][f.b] = bf[f.b][f.c] = bf[f.c][f.a] = ++fcnt;
    fc[fcnt] = f;
  }
  random_shuffle(pt + 5, pt + 1 + n);
  for (int i = 5; i <= n; ++i) {
    for (int j = 1; j <= fcnt; ++j) {
      if (!fc[j].vis) continue;
      if (dblcmp(volume(pt[i], pt[fc[j].a], pt[fc[j].b], pt[fc[j].c])) >= 0) {
        dfs(i, j);
        break;
      }
    }
  }
  for (int i = 1; i <= fcnt; ++i) if (!fc[i].vis) swap(fc[i--], fc[fcnt--]);
}
```

5 Graph

5.1 Tarjan

```
void tarjan(int u) {
  int v;
  dfn[u] = low[u] = ++ind;
  sk.push(u), instack[u] = true;
  for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
    v = *it; // Check if v is the parent of u(BCC)
    if (!dfn[v]) {
      tarjan(v);
      low[u] = min(low[u], low[v]);
      // Check BCC here
      // Cut vertex: Pop stack until v is poped, then add u.
    } else if (instack[v]) { // ! instack[v] -> else
      low[u] = min(low[u], dfn[v]);
    }
  }
  /* Check SCC here
  if (dfn[u] == low[u]) {
    ++color;
    do {
      v = sk.top(), instack[v] = false;
      sk.pop();
      block[v] = color;
    } while (v != u);
  } */
}
```

For bidirectional graph(cut vertex & bridge):
root: u has 2 or more children.
others: exist a child v satifsying $dfn[u] \le low[v]$.
(u, v) is bridge only if $dfn[u] < low[v]$ (trick: multiple edges).

5.2 Maximum Flow

```
// assuming the sink has the maximum vertex ID => t = n
int n, m, s, t, ec, d[maxn], vd[maxn];
struct edge_link {
  int v, r;
  edge_link *next, *pair;
} edge[maxm], *header[maxn], *current[maxn];
void add(int u, int v, int r) // (u, v, r), (v, u, 0)

int augment(int u, int flow) {
  if (u == t) return flow;
  int temp, res = 0;
  for (edge_link *&e = current[u]; e != NULL; e = e->next) {
    if (e->r && d[u] == d[e->v] + 1) {
      temp = augment(e->v, min(e->r, flow - res));
      e->r -= temp, e->pair->r += temp, res += temp;
      if (d[s] == t || res == flow) return res;
    }
  }
  if (--vd[d[u]] == 0) d[s] = t;
  else current[u] = header[u], ++vd[++d[u]];
  return res;
}

int sap() {
  int flow = 0;
  memset(d, 0, sizeof(d)), memset(vd, 0, sizeof(vd));
  vd[0] = t;
  for (int i = 1; i <= t; ++i) current[i] = header[i];
  while (d[s] < t) flow += augment(s, maxint);
  return flow;
}
```

Notes on vertex covering and independent set on bipartite graph:
Minimum Vertex Covering Set V': $\forall (u, v) \in E$, $u \in V'$ or $v \in V'$ holds.
Maximum Vertex Independent Set V': $\forall u, v \in V'$, $(u, v) \notin E$ holds.
Construct a flow graph G, run DFS from s on reduction graph, the vertices not visited

in left side and visited in right side form the minimum vertex covering set. Maximum vertex independent set vice versa.

5.3 Minimum Cost, Maximum Flow

```
int n, m, s, t, ec, d[maxn]; // Minimum cost, maximum flow(ZKW version), t=n
struct edge_link {
  int v, r, w;
  edge_link *next, *pair;
} edge[maxm], *header[maxn];
bool vis[maxn];
void add(int u, int v, int r, int w) // (u, v, r, w), (v, u, 0, -w)

void spfa() {
  queue<int> q;
  for (int i = 1; i <= t; ++i) d[i] = maxint, vis[i] = false;
  d[s] = 0, q.push(s), vis[s] = true;
  while (!q.empty()) {
    int u = q.front();
    q.pop(), vis[u] = false;
    for (edge_link *e = header[u]; e != NULL; e = e->next) {
      if (e->r && d[u] + e->w < d[e->v]) {
        d[e->v] = d[u] + e->w;
        if (!vis[e->v]) q.push(e->v), vis[e->v] = true;
      }
    }
  }
  for (int i = 1; i <= t; ++i) d[i] = d[t] - d[i];
}

int augment(int u, int flow) {
  if (u == t) return flow;
  vis[u] = true;
  for (edge_link *e = header[u]; e != NULL; e = e->next) {
    if (e->r && !vis[e->v] && d[e->v] + e->w == d[u]) {
      int temp = augment(e->v, min(flow, e->r));
      if (temp) {
        e->r -= temp, e->pair->r += temp;
        return temp;
      }
    }
  }
  return 0;
}

bool adjust() {
  int delta = maxint;
  for (int u = 1; u <= t; ++u) {
    if (!vis[u]) continue;
    for (edge_link *e = header[u]; e != NULL; e = e->next) {
      if (e->r && !vis[e->v] && d[e->v] + e->w > d[u]) {
        delta = min(delta, d[e->v] + e->w - d[u]);
      }
    }
  }
  if (delta == maxint) return false;
  for (int i = 1; i <= t; ++i) {
    if (vis[i]) d[i] += delta;
  }
  memset(vis, 0, sizeof(vis));
  return true;
}

pair<int, int> cost_flow() {
  int temp, flow = 0, cost = 0;
  spfa();
  do {
    while (temp = augment(s, maxint)) {
      flow += temp;
      memset(vis, 0, sizeof(vis));
    }
  } while (adjust());
  for (int i = 2; i <= ec; i += 2) cost += edge[i].r * edge[i - 1].w;
  return make_pair(flow, cost);
}
```

5.4 Kuhn Munkras

```
// Kuhn-Munkras's algorithm, maxn: Left Size; maxm: Right Size.
int n, m, g[maxn][maxm], lx[maxn], ly[maxm], slack[maxm], match[maxm];
bool vx[maxn], vy[maxm];

bool find(int x) {
  vx[x] = true;
  for (int y = 1; y <= m; ++y) {
    if (!vy[y]) {
      int delta = lx[x] + ly[y] - g[x][y];
      if (delta == 0) {
        vy[y] = true;
        if (match[y] == 0 || find(match[y])) {
          match[y] = x;
          return true;
        }
      } else slack[y] = min(slack[y], delta);
    }
  }
  return false;
}

int km() { // #define sm(p, f) memset((p), f, sizeof(p))
  // maximum weight, if minimum, negate all g then restore at the end.
  sm(lx, 0x80), sm(ly, 0), sm(match, 0);
  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) lx[i] = max(lx[i], g[i][j]);
  }
  for (int k = 1; k <= n; ++k) {
    sm(slack, 0x7f);
    while (true) {
      sm(vx, 0), sm(vy, 0);
      if (find(k)) break;
      else {
        int delta = maxint;
        for (int i = 1; i <= m; ++i) {
          if (!vy[i]) delta = min(delta, slack[i]);
        }
        for (int i = 1; i <= n; ++i) {
          if (vx[i]) lx[i] -= delta;
        }
        for (int i = 1; i <= m; ++i) {
          if (vy[i]) ly[i] += delta;
          if (!vy[i]) slack[i] -= delta;
        }
      }
    }
  }
  int result = 0;
  for (int i = 1; i <= n; ++i) result += lx[i];
  for (int i = 1; i <= m; ++i) result += ly[i];
  return result;
}
```

5.5 Hopcroft Karp

```
int n, m, vis[maxn], level[maxn], pr[maxn], pr2[maxn];
vector<int> edge[maxn]; // for Left

bool dfs(int u) {
  vis[u] = true;
  for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
    int v = pr2[*it];
    if (v == -1 || (!vis[v] && level[u] < level[v] && dfs(v))) {
      pr[u] = *it, pr2[*it] = u;
      return true;
    }
  }
  return false;
}

int hopcroftKarp() {
  memset(pr, -1, sizeof(pr)); memset(pr2, -1, sizeof(pr2));
  for (int match = 0; ;) {
    queue<int> Q;
    for (int i = 1; i <= n; ++i) {
      if (pr[i] == -1) {
        level[i] = 0;
        Q.push(i);
      } else level[i] = -1;
    }
    while (!Q.empty()) {
      int u = Q.front(); Q.pop();
      for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
        int v = pr2[*it];
        if (v != -1 && level[v] < 0) {
          level[v] = level[u] + 1;
          Q.push(v);
        }
      }
    }
    for (int i = 1; i <= n; ++i) vis[i] = false;
    int d = 0;
    for (int i = 1; i <= n; ++i) if (pr[i] == -1 && dfs(i)) ++d;
    if (d == 0) return match;
    match += d;
  }
}
```

5.6 Blossom Matching

```
int n, match[maxn], pre[maxn], base[maxn]; // maximum matching on graphs
vector<int> edge[maxn];
bool inQ[maxn], inB[maxn], inP[maxn];
queue<int> Q;

int LCA(int u, int v) {
  for (int i = 1; i <= n; ++i) inP[i] = false;
  while (true) {
    u = base[u];
    inP[u] = true;
    if (match[u] == -1) break;
    u = pre[match[u]];
  }
  while (true) {
    v = base[v];
    if (inP[v]) return v;
    v = pre[match[v]];
  }
}

void reset(int u, int a) {
  while (u != a) {
    int v = match[u];
    inB[base[u]] = inB[base[v]] = true;
    v = pre[v];
    if (base[v] != a) pre[v] = match[u];
    u = v;
  }
}

void contract(int u, int v) {
  int a = LCA(u, v);
  for (int i = 1; i <= n; ++i) inB[i] = false;
  reset(u, a), reset(v, a);
  if (base[u] != a) pre[u] = v;
  if (base[v] != a) pre[v] = u;
  for (int i = 1; i <= n; ++i) {
    if (!inB[base[i]]) continue;
    base[i] = a;
    if (!inQ[i]) Q.push(i), inQ[i] = true;
  }
}

bool dfs(int s) {
  for (int i = 1; i <= n; ++i) pre[i] = -1, inQ[i] = false, base[i] = i;
  while (!Q.empty()) Q.pop();
  Q.push(s), inQ[s] = true;
  while (!Q.empty()) {
    int u = Q.front();
    Q.pop();
    for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
      int v = *it;
      if (base[u] == base[v] || match[u] == v) continue;
      if (v == s || (match[v] != -1 && pre[match[v]] != -1)) contract(u, v);
      else if (pre[v] == -1) {
        pre[v] = u;
        if (match[v] != -1) {
          Q.push(match[v]), inQ[match[v]] = true;
        } else {
          u = v;
          while (u != -1) {
            v = pre[u];
            int w = match[v];
            match[u] = v, match[v] = u;
            u = w;
          }
          return true;
        }
      }
    }
  }
  return false;
}

int blossom() {
  int ans = 0;
  for (int i = 1; i <= n; ++i) match[i] = -1;
  for (int i = 1; i <= n; ++i) {
    if (match[i] == -1 && dfs(i)) ++ans;
  }
  return ans;
}
```

5.7 Stoer Wagner

```
// stoer-wagner algorithm, complexity: O(n^3)
// used to compute the global minimum cut, and self-loop is ignored.
int n, g[maxn][maxn], v[maxn], d[maxn], vis[maxn];
int stoer_wagner(int n) {
  int res = maxint;
  for (int i = 1; i <= n; i++) v[i] = i, vis[i] = 0;
  while (n > 1) {
    int p = 2, prev = 1;
    for (int i = 2; i <= n; ++i) {
      d[v[i]] = g[v[1]][v[i]];
      if (d[v[i]] > d[v[p]]) p = i;
    }
    vis[v[1]] = n;
    for (int i = 2; i <= n; ++i) {
      if (i == n) {
        res = min(res, d[v[p]]); // if d[v[p]] < res, then s = v[p] & t = v[prev]
        for (int j = 1; j <= n; ++j) {
          g[v[prev]][v[j]] += g[v[p]][v[j]];
          g[v[j]][v[prev]] = g[v[prev]][v[j]];
        }
        v[p] = v[n--];
        break;
      }
      vis[v[p]] = n;
      prev = p;
      p = -1;
      for (int j = 2; j <= n; ++j) {
        if (vis[v[j]] != n) {
          d[v[j]] += g[v[prev]][v[j]];
          if (p == -1 || d[v[p]] < d[v[j]]) p = j;
        }
      }
    }
  }
  return res;
}
```

5.8 Arborescence

```
int n, ec, ID[maxn], pre[maxn], in[maxn], vis[maxn];
struct edge_t {
  int u, v, w;
} edge[maxm];
void add(int u, int v, int w) {
  edge[++ec].u = u, edge[ec].v = v, edge[ec].w = w;
}

int arborescence(int n, int root) {
  int res = 0, index;
  while (true) {
    for (int i = 1; i <= n; ++i) {
      in[i] = maxint, vis[i] = -1, ID[i] = -1;
    }
    for (int i = 1; i <= ec; ++i) {
      int u = edge[i].u, v = edge[i].v;
      if (u == v || in[v] <= edge[i].w) continue;
      in[v] = edge[i].w, pre[v] = u;
    }
    pre[root] = root, in[root] = 0;
    for (int i = 1; i <= n; ++i) {
      res += in[i];
      if (in[i] == maxint) return -1;
    }
    index = 0;
    for (int i = 1; i <= n; ++i) {
      if (vis[i] != -1) continue;
      int u = i, v;
      while (vis[u] == -1) {
        vis[u] = i;
        u = pre[u];
      }
      if (vis[u] != i || u == root) continue;
      for (v = u, u = pre[u], ++index; u != v; u = pre[u]) ID[u] = index;
      ID[v] = index;
    }
    if (index == 0) return res;
    for (int i = 1; i <= n; ++i) if (ID[i] == -1) ID[i] = ++index;
    for (int i = 1; i <= ec; ++i) {
      int u = edge[i].u, v = edge[i].v;
      edge[i].u = ID[u], edge[i].v = ID[v];
      edge[i].w -= in[v];
    }
    n = index, root = ID[root];
  }
  return res;
}
```

5.9 Manhattan MST

```
struct point {
  int x, y, index;
  bool operator<(const point &p) const { return x == p.x ? y < p.y : x < p.x; }
} p[maxn];
struct node {
  int value, p;
} T[maxn];

int query(int x) {
  int r = maxint, p = -1;
  for (; x <= n; x += (x & -x)) if (T[x].value < r) r = T[x].value, p = T[x].p;
  return p;
}

void modify(int x, int w, int p) {
  for (; x > 0; x -= (x & -x)) if (T[x].value > w) T[x].value = w, T[x].p = p;
}

int manhattan() {
  for (int i = 1; i <= n; ++i) p[i].index = i;
  for (int dir = 1; dir <= 4; ++dir) {
    if (dir == 2 || dir == 4) {
      for (int i = 1; i <= n; ++i) swap(p[i].x, p[i].y);
    } else if (dir == 3) {
      for (int i = 1; i <= n; ++i) p[i].x = -p[i].x;
    }
    sort(p + 1, p + 1 + n);
    vector<int> v; static int a[maxn];
    for (int i = 1; i <= n; ++i) a[i] = p[i].y - p[i].x, v.push_back(a[i]);
    sort(v.begin(), v.end()); v.erase(unique(v.begin(), v.end()), v.end());
  for (int i = 1; i <= n; ++i) a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.
    begin() + 1;
  for (int i = 1; i <= n; ++i) T[i].value = maxint, T[i].p = -1;
  for (int i = n; i >= 1; --i) {
    int pos = query(a[i]);
    if (pos != -1) add(p[i].index, p[pos].index, dist(p[i], p[pos]));
    modify(a[i], p[i].x + p[i].y, i);
  }
 }
 return kruskal();
}
```

5.10 Minimum Mean Cycle

```
int dp[maxn][maxn]; // minimum mean cycle(allow negative weight)
double mmc(int n) {
  for (int i = 0; i < n; ++i) {
    memset(dp[i + 1], 0x7f, sizeof(dp[i + 1]));
    for (int j = 1; j <= ec; ++j) {
      int u = edge[j].u, v = edge[j].v, w = edge[j].w;
      if (dp[i][u] != maxint) dp[i + 1][v] = min(dp[i + 1][v], dp[i][u] + w);
    }
  }
  double res = maxdbl;
  for (int i = 1; i <= n; ++i) {
    if (dp[n][i] == maxint) continue;
    double value = -maxdbl;
    for (int j = 0; j < n; ++j) {
      value = max(value, (double)(dp[n][i] - dp[j][i]) / (n - j));
    }
    res = min(res, value);
  }
  return res;
}
```

5.11 Divide and Conquer on Tree

```
int bk, size[maxn], parent[maxn], ver[maxn];
bool cut[maxn];

void bfs(int r) { // bfs in each sub-tree
  parent[r] = 0, bk = 0; // maintain root extra information
  static queue<int> Q; static stack<int> U;
  Q.push(r);
  while (!Q.empty()) {
    int u = Q.front();
    Q.pop(); U.push(u);
    size[u] = 1, ver[++bk] = u; // find a node in sub-tree
    for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
      int v = *it;
      if (v == parent[u] || cut[v]) continue;
      parent[v] = u; // maintain v from u
      Q.push(v);
    }
  }
  while (!U.empty()) {
    int u = U.top(); U.pop();
    if (parent[u]) size[parent[u]] += size[u];
  }
}

int findCentre(int r) {
  static queue<int> Q;
  int result = 0, rsize = maxint;
  bfs(r);
  Q.push(r);
  while (!Q.empty()) {
    int u = Q.front();
    Q.pop();
    int temp = size[r] - size[u];
    for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
      int v = *it;
      if (cut[v] || v == parent[u]) continue;
      temp = max(temp, size[v]);
      Q.push(v);
    }
    if (temp < rsize) rsize = temp, result = u;
  }
  return result;
}

int work(int u) {
  int result = 0;
  u = findCentre(u);
  cut[u] = true;
  for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
    int v = *it;
    if (!cut[v]) /*result += */work(v); // process each sub-tree
  }
  for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
    int v = *it;
    if (cut[v]) continue;
    bfs(v); // then combine sub-trees
  }
  cut[u] = false;
```

```cpp
  return result;
}
```

5.12 Dominator Tree

A dominator tree is a tree where each node's children are those nodes it immediately dominates. Because the immediate dominator is unique, it is a tree. The start node is the root of the tree.

```cpp
int parent[maxn], label[maxn], cnt, real[maxn];
vector<int> edge[maxn], succ[maxn], pred[maxn];
int semi[maxn], idom[maxn], ancestor[maxn], best[maxn];
deque<int> bucket[maxn];

void dfs(int u) {
  label[u] = ++cnt; real[cnt] = u;
  for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
    int v = *it;
    if (v == parent[u] || label[v] != -1) continue;
    parent[v] = u;
    dfs(v);
  }
}

void link(int v, int w) {
  ancestor[w] = v;
}

void compress(int v) {
  int a = ancestor[v];
  if (ancestor[a] == 0) return;
  compress(a);
  if (semi[best[v]] > semi[best[a]]) best[v] = best[a];
  ancestor[v] = ancestor[a];
}

int eval(int v) {
  if (ancestor[v] == 0) return v;
  compress(v);
  return best[v];
}

void dominator() { // clear succ & pred, let cnt = 0 first
  for (int i = 1; i <= n; ++i) label[i] = -1;
  dfs(n); // n is root
  for (int u = 1; u <= n; ++u) {
    for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
      int v = *it;
      if (label[u] != -1 && label[v] != -1) {
        succ[label[u]].push_back(label[v]);
        pred[label[v]].push_back(label[u]);
      }
    }
  }
  for (int i = 1; i <= n; ++i) {
    semi[i] = best[i] = i;
    idom[i] = ancestor[i] = 0;
    bucket[i].clear();
  }
  for (int w = cnt; w >= 2; --w) {
    int p = label[parent[real[w]]];
    for (vector<int>::iterator it = pred[w].begin(); it != pred[w].end(); ++it) {
      int v = *it;
      int u = eval(v);
      if (semi[w] > semi[u]) semi[w] = semi[u];
    }
    bucket[semi[w]].push_back(w);
    link(p, w);
    while (!bucket[p].empty()) {
      int v = bucket[p].front();
      bucket[p].pop_front();
      int u = eval(v);
      idom[v] = (semi[u] < p ? u : p);
    }
  }
  for (int w = 2; w <= cnt; ++w) {
    if (idom[w] != semi[w]) idom[w] = idom[idom[w]];
  }
  idom[1] = 0;
  for (int i = 1; i <= cnt; ++i) {
    int u = real[idom[i]], v = real[i];
    // u is immediate dominator of v (i == 1?)
  }
}
```

6 Miscellaneous

6.1 High Precision Calculation

```cpp
const int B = 10000, D = 4;
struct bnum_t {
  int l;
  vector<int> b;
  bnum_t(int n) {
    b.push_back(0);
    if (n == 0) l = 1, b.push_back(0);
    else {
      l = 0;
      for (int i = 1; n; ++i, n /= B) ++l, b.push_back(n % B);
    }
  }
  bnum_t() { *this = bnum_t(0); }
  bnum_t(char *str) {
    l = 0, b.push_back(0);
    int temp = strlen(str);
    for (int i = temp - 1; i >= 0; i -= D) {
      int v = 0;
      for (int j = 0, p10 = 1; j < D && i - j >= 0; ++j, p10 *= 10) v += (str[i - j]
          - '0') * p10;
      ++l, b.push_back(v);
    }
  }
  friend ostream &operator<<(ostream &os, const bnum_t &r) {
    os << r.b[r.l];
    for (int i = r.l - 1; i >= 1; --i) os << setfill('0') << setw(D) << r.b[i];
    return os;
  }
  bnum_t operator+(const bnum_t &r) {
    bnum_t f(*this);
    f.l = max(l, r.l);
    f.b.resize(f.l + 5);
    for (int i = 1; i <= min(f.l, r.l); ++i) {
      f.b[i] += r.b[i];
      if (f.b[i] / B) f.b[i + 1] += f.b[i] / B, f.b[i] %= B;
    }
    while (f.b[f.l + 1]) f.l += 1;
    f.b.resize(f.l + 1);
    return f;
  }
  bnum_t operator-(const bnum_t &r) { // assume a > b
    bnum_t f = *this;
    f.l = max(l, r.l);
    for (int i = 1; i <= min(f.l, r.l); ++i) {
      f.b[i] -= r.b[i];
      while (f.b[i] < 0) f.b[i] += B, --f.b[i + 1];
    }
    while (f.b[f.l] == 0) f.l -= 1;
    f.b.resize(f.l + 1);
    return f;
  }
  bnum_t operator*(const bnum_t &r) {
    bnum_t f = 0;
    f.l = l + r.l - 1;
    f.b.resize(f.l + 5);
    for (int i = 1; i <= l; ++i) {
      for (int j = 1; j <= r.l; ++j) {
        f.b[i + j - 1] += b[i] * r.b[j];
        if (f.b[i + j - 1] / B) f.b[i + j] += f.b[i + j - 1] / B, f.b[i + j - 1] %=
          B;
      }
    }
    while (f.b[f.l + 1]) f.l += 1;
    f.b.resize(f.l + 1);
    return f;
  }
  bnum_t operator*(const int &r) {
    return (*this) * bnum_t(r);
  }
  bnum_t operator/(const int &r) { // assume a > b
    bnum_t f = *this;
    for (int i = f.l; i >= 1; --i) f.b[i - 1] += (f.b[i] % r) * B, f.b[i] /= r;
    while (f.b[f.l] == 0) f.l -= 1;
    f.b.resize(f.l + 1);
    return f;
  }
};
```

6.2 Expression Parsing

```
void deal(stack<int> &num, stack<char> &oper) {
  int x, y;
  y = num.top(); num.pop();
  char op = oper.top(); oper.pop();
  if (op == '?') num.push(-y);
  else {
    x = num.top(); num.pop();
    num.push(cal(x, y, op));
  }
}

int parse(char *s) {
  static int priv[256];
  stack<int> num;
  stack<char> oper;
  priv['+'] = priv['-'] = 3;
  priv['*'] = priv['/'] = 2;
  priv['('] = 10;
  int len = strlen(s);
  char last = 0;
  for (int i = 0; i < len; ++i)  {
    if (isdigit(s[i])) {
```

```cpp
      int tmp = 0;
      while (isdigit(s[i])) tmp = tmp * 10 + s[i++] - '0';
      i -= 1;
      num.push(tmp);
    } else if (s[i] == '(') {
      oper.push(s[i]);
    } else if (s[i] == ')') {
      while (oper.top() != '(') deal(num, oper);
      oper.pop();
    } else if (s[i] == '-' && (last == 0 || last == '(')) {
      oper.push('?'); // unary operator
    } else if (priv[s[i]] > 0) {
      while (!oper.empty() && priv[s[i]] >= priv[oper.top()]) deal(num, oper); // >= used for operator of left associative law
      oper.push(s[i]);
    } else continue;
    last = s[i];
  }
  while (!oper.empty()) deal(num, oper);
  return num.top();
}
```

6.3 Steiner's Problem

```cpp
// Steiner's Problem: ts[m], list of vertices to be united, indexed from 0.
int steiner(int *ts, int m) { // O(3^m*n+2^m*n^2+n^3)
  floyd();
  memset(dp, 0, sizeof(dp));
  for (int i = 0; i < m; ++i) {
    for (int j = 1; 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 = 1; j <= n; ++j) {
        dp[i][j] = maxint;
        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 = 1; j <= n; ++j) {
        for (int k = 1; k <= n; ++k) {
          dp[i][j] = min(dp[i][j], dp[i][k] + g[k][j]);
        }
      }
    }
  }
  return dp[(1 << m) - 1][ts[0]];
}
```

6.4 AlphaBeta

```cpp
int alphabeta(state s, int alpha, int beta) {
  if (s.finished()) return s.score();
  for (state t : s.next()) {
    alpha = max(alpha, -alphabeta(t, -beta, -alpha));
    if (alpha >= beta) break;
  }
  return alpha;
}
```

6.5 LCA

```cpp
const int logn = 20;
int parent[maxn], lca[logn][maxn], depth[maxn];

void initLCA() {
  for (int i = 1; i <= n; ++i) lca[0][i] = parent[i];
  for (int j = 1; j < logn; ++j) {
    for (int i = 1; i <= n; ++i) {
      lca[j][i] = lca[j - 1][lca[j - 1][i]];
    }
  }
}

int LCA(int x, int y) {
  if (depth[x] < depth[y]) swap(x, y);
  for (int i = logn - 1; i >= 0; --i) {
    if (depth[x] - (1 << i) >= depth[y]) x = lca[i][x];
  }
  if (x == y) return x;
  for (int i = logn - 1; i >= 0; --i) {
    if (lca[i][x] != lca[i][y]) {
      x = lca[i][x], y = lca[i][y];
    }
  }
  return lca[0][x];
}
```

6.6 Dancing Links X

```
const int maxm = 325, maxk = 5001;
// maxk = number of 1 + number of columns, be carefully when memset
int pt, L[maxk], R[maxk], U[maxk], D[maxk], C[maxk], A[maxk], O[maxm], S[maxm], res[
  maxk];

void init(int m) { // m: number of columns, #define ms memset
  ms(L), ms(R), ms(U), ms(D), ms(C), ms(A), ms(O), ms(S), pt = m;
  for (int i = 0; i <= m; ++i) L[i] = i - 1, R[i] = i + 1, C[i] = D[i] = U[i] = i;
  L[0] = m, R[m] = 0;
}

void insert(int row, int *t, int total) { // t: positions of 1
  for (int i = 1; i <= total; ++i) {
    int col = t[i];
    ++S[col], ++pt;
    C[pt] = col, A[pt] = row, U[pt] = col, D[pt] = D[col];
    U[D[col]] = pt, D[col] = pt;
    if (i != 1) {
      L[pt] = pt - 1, R[pt] = pt - i + 1, R[pt - 1] = L[pt - i + 1] = pt;
    } else {
      L[pt] = R[pt] = pt;
    }
  }
}

void remove(int x) {
  L[R[x]] = L[x], R[L[x]] = R[x];
  for (int i = D[x]; i != x; i = D[i]) {
    for (int j = R[i]; j != i; j = R[j]) {
      U[D[j]] = U[j], D[U[j]] = D[j], --S[C[j]];
    }
  }
}

void resume(int x) {
  for (int i = U[x]; i != x; i = U[i]) {
    for (int j = L[i]; j != i; j = L[j]) {
      U[D[j]] = j, D[U[j]] = j, ++S[C[j]];
    }
  }
  L[R[x]] = x, R[L[x]] = x;
}

bool dlx(int k) {
  if (R[0] == 0) {
    for (int i = 0; i < k; ++i) res[i] = O[i];
    return true;
  } else {
    int val = maxint, col = 0;
    for (int i = R[0]; i != 0; i = R[i]) {
      if (S[i] <= val) col = i, val = S[i];
    }
    if (val == 0) return false;
    remove(col);
    for (int i = D[col]; i != col; i = D[i]) {
      O[k] = A[i];
      for (int j = R[i]; j != i; j = R[j]) remove(C[j]);
      if (dlx(k + 1)) return true;
      for (int j = L[i]; j != i; j = L[j]) resume(C[j]);
    }
    resume(col);
    return false;
  }
}
// call dlx(0)
```

Find the minimum row set, satisfying each column has exactly one 1.
Let the row representing each choice, if some choices are mutually exclusive, add a column with these rows associated.
Use the sudoku problem as an instance. There are 729 choices(81 squares can be filled in with 9 numbers), 4 constraints:
(1). Each box has exactly one number;
(2). Number 1 to 9 appears exactly once in each row, column, sub square.
Thus we can construct a 729∗324 matrix, each 81 columns representing each constraint.

6.7 Linar Programming

```
double a[maxn][maxm], b[maxn], c[maxm], d[maxn][maxm];
int ix[maxn + maxm]; // !!! array all indexed from 0
// max{cx|Ax<=b,x>=0}, n: constraints, m: vars
double simplex(double a[maxn][maxm], double b[maxn], double c[maxm], int n, int m) {
  ++m;
  int r = n, s = m - 1;
  memset(d, 0, sizeof(d));
  for (int i = 0; i < n + m; ++i) ix[i] = i;
  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;
  for (double dd;; ) {
    if (r < n) {
```

```cpp
      int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
      d[r][s] = 1.0 / d[r][s];
      for (int j = 0; j <= m; ++j) if (j != s) d[r][j] *= -d[r][s];
      for (int i = 0; i <= n + 1; ++i) if (i != r) {
        for (int j = 0; j <= m; ++j) if (j != s) d[i][j] += d[r][j] * d[i][s];
        d[i][s] *= d[r][s];
      }
    }
    r = -1; 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 || (dd = d[r][m] / d[r][s] - d[i][m] / d[i][s]) < -eps || (dd < eps
        && ix[r + m] > ix[i + m])) r = i;
    }
    if (r < 0) return -1; // not bounded
  }
  if (d[n + 1][m] < -eps) return -1; // not executable
  double ans = 0;
  for (int i = m; i < n + m; ++i) { // the missing enumerated x[i] = 0
    if (ix[i] < m - 1) ans += d[i - m][m] * c[ix[i]];
  }
  return ans;
}
```

6.8 Simpson Integration

```cpp
const double eps = 1e-15;
double f(double x) { return 0.0; }
double sim(double l, double r, double lv, double rv, double mv) {
  return (r - l) * (lv + rv + 4 * mv) / 6;
}
double rsim(double l, double r, double lv, double rv, double mv, double m1v, double
  m2v) {
  double mid = (l + r) / 2;
  if (fabs(sim(l, r, lv, rv, mv) - sim(l, mid, lv, mv, m1v) - sim(mid, r, mv, rv,
    m2v)) / 15 < eps) {
    return sim(l, r, lv, rv, mv);
  } else {
    double mid = (l + r) / 2, m1 = (l + (l + r) / 2) / 2, m2 = ((l + r) / 2 + r) /
      2;
    return rsim(l, mid, lv, mv, m1v, f((l + m1) / 2), f((m1 + mid) / 2)) + rsim(mid,
       r, mv, rv, m2v, f((mid + m2) / 2), f((m2 + r) / 2));
  }
}
double simpson(double l, double r) {
  double mid = (l + r) / 2;
  return rsim(l, r, f(l), f(r), f(mid), f((l + mid) / 2), f((mid + r) / 2));
}
```

6.9 Fast Fourier Transform

```cpp
void fft(int sign, int n, double *real, double *imag) {
  double theta = sign * 2 * pi / n;
  for (int m = n; m >= 2; m >>= 1, theta *= 2) {
    double wr = 1, wi = 0, c = cos(theta), s = sin(theta);
    for (int i = 0, mh = m >> 1; i < mh; ++i) {
      for (int j = i; j < n; j += m) {
        int k = j + mh;
        double xr = real[j] - real[k], xi = imag[j] - imag[k];
        real[j] += real[k], imag[j] += imag[k];
        real[k] = wr * xr - wi * xi, imag[k] = wr * xi + wi * xr;
      }
      double _wr = wr * c - wi * s, _wi = wr * s + wi * c;
      wr = _wr, wi = _wi;
    }
  }
  for (int i = 1, j = 0; i < n; ++i) {
    for (int k = n >> 1; k > (j ^= k); k >>= 1);
    if (j < i) swap(real[i], real[j]), swap(imag[i], imag[j]);
  }
}
// Compute Poly(a)*Poly(b), write to r; Indexed from 0
int mult(int *a, int n, int *b, int m, int *r) {
  static double ra[maxn], rb[maxn], ia[maxn], ib[maxn];
  int fn = 1;
  while (fn < n + m) fn <<= 1; // n + m: interested length
  for (int i = 0; i < n; ++i) ra[i] = a[i], ia[i] = 0;
  for (int i = n; i < fn; ++i) ra[i] = ia[i] = 0;
  for (int i = 0; i < m; ++i) rb[i] = b[i], ib[i] = 0;
  for (int i = m; i < fn; ++i) rb[i] = ib[i] = 0;
  fft(1, fn, ra, ia);
  fft(1, fn, rb, ib);
  for (int i = 0; i < fn; ++i) {
    double real = ra[i] * rb[i] - ia[i] * ib[i];
    double imag = ra[i] * ib[i] + rb[i] * ia[i];
    ra[i] = real, ia[i] = imag;
  }
  fft(-1, fn, ra, ia);
  for (int i = 0; i < fn; ++i) r[i] = (int)floor(ra[i] / fn + 0.5);
  return fn;
}
```

6.10 Number Theoretic Transform

```
const int P = 1000003; // Approximate 10^6
const int P1 = 998244353, P2 = 995622913;
const LL M1 = 397550359381069386LL, M2 = 596324591238590904LL;
const LL MM = 993874950619660289LL;

int CRT(int x1, int x2) {
  return (mult(M1, x1, MM) + mult(M2, x2, MM)) % MM % P; // 64bit multiplication
}

void NTT(int *A, int PM, int PW, int n) {
  for (int m = n, h; h = m / 2, m >= 2; PW = (LL)PW * PW % PM, m = h) {
    for (int i = 0, w = 1; i < h; ++i, w = (LL)w * PW % PM) {
      for (int j = i; j < n; j += m) {
        int k = j + h, x = (A[j] - A[k] + PM) % PM;
        A[j] += A[k]; A[j] %= PM;
        A[k] = (LL)w * x % PM;
      }
    }
  }
  for (int i = 0, j = 1; j < n - 1; ++j) {
    for (int k = n / 2; k > (i ^= k); k /= 2);
    if (j < i) swap(A[i], A[j]);
  }
}

int E1, E2, F1, F2, I1, I2;
int init(int n) { // assert(k <= 19);
  int k = 1, N = 2, p;
  while (N < n) N <<= 1, ++k;
  p = 7 * 17; for (int i = 1; i <= 23 - k; ++i) p *= 2;
  E1 = fpow(3, p, P1); F1 = fpow(E1, P1 - 2, P1); I1 = fpow(N, P1 - 2, P1);
  p = 9 * 211; for (int i = 1; i <= 19 - k; ++i) p *= 2;
  E2 = fpow(5, p, P2); F2 = fpow(E2, P2 - 2, P2); I2 = fpow(N, P2 - 2, P2);
  return N;
}

void mul(int *A, int *B, int *C, int n) {
  static int A1[maxn], B1[maxn], C1[maxn];
  int N = init(n);
  memset(A1, 0, sizeof(*A1) * N); memset(B1, 0, sizeof(*B1) * N); memset(C1, 0,
    sizeof(*C1) * N);
  memset(C, 0, sizeof(*C) * N);
  memcpy(A1, A, sizeof(*A) * n); memcpy(B1, B, sizeof(*B) * n);
  NTT(A1, P1, E1, N); NTT(B1, P1, E1, N);
  for (int i = 0; i < N; ++i) C1[i] = (LL)A1[i] * B1[i] % P1;
  NTT(C1, P1, F1, N);
  for (int i = 0; i < N; ++i) C1[i] = (LL)C1[i] * I1 % P1;
  NTT(A, P2, E2, N); NTT(B, P2, E2, N);
  for (int i = 0; i < N; ++i) C[i] = (LL)A[i] * B[i] % P2;
  NTT(C, P2, F2, N);
  for (int i = 0; i < N; ++i) C[i] = (LL)C[i] * I2 % P2;
  for (int i = 0; i < N; ++i) C[i] = CRT(C1[i], C[i]);
  for (int i = n; i < N; ++i) C[i] = 0;
}
```

6.11 Mo-Tao Algorithm

```
// Complexity: Q*N^0.5 * O(add)
int SQRTN = (int)sqrt((double)q);
sort(Q + 1, Q + 1 + q, cmpL);
for (int i = 1; i <= q; i += SQRTN) {
  clear();
  int begin = i, end = i + SQRTN - 1;
  if (end > q) end = q;
  sort(Q + begin, Q + end + 1, cmpR);
  Q[begin - 1].l = 1, Q[begin - 1].r = 0;
  for (int j = begin; j <= end; ++j) {
    for (int k = Q[j - 1].r + 1; k <= Q[j].r; ++k) add(k, 1);
    if (Q[j].l > Q[j - 1].l) {
      for (int k = Q[j - 1].l; k < Q[j].l; ++k) add(k, -1);
    } else if (Q[j].l < Q[j - 1].l) {
      for (int k = Q[j].l; k < Q[j - 1].l; ++k) add(k, 1);
    }
    ans[Q[j].ID] = res;
  }
}
```

6.12 Cyclic LCS

```
const int maxn = 3001;
int dp[maxn][maxn], pa[maxn][maxn];

int trace(int sx, int sy, int ex, int ey) {
  int l = 0;
  while (ex != sx || ey != sy) {
    if (pa[ex][ey] == 1) --ey;
    else if (pa[ex][ey] == 2) --ex, --ey, ++l;
    else --ex;
  }
  return l;
}

void reroot(int root, int m, int n) {
  int i = root, j = 1;
  while (j <= n && pa[i][j] != 2) ++j;
  if (j > n) return;
  pa[i][j] = 1;
  while (i < 2 * m && j < n) {
    if (pa[i + 1][j] == 3) pa[++i][j] = 1;
    else if (pa[i + 1][j + 1] == 2) pa[++i][++j] = 1;
    else ++j;
  }
  while (i < 2 * m && pa[i + 1][j] == 3) pa[++i][j] = 1;
}

void lcs(char *a, char *b) {
  int m = strlen(a + 1), n = strlen(b + 1);
  for (int i = 0; i <= m; ++i) {
    for (int j = 0; j <= n; ++j) {
      if (i != 0 || j != 0) dp[i][j] = -1;
      if (j >= 1 && dp[i][j] < dp[i][j - 1]) dp[i][j] = dp[i][j - 1], pa[i][j] = 1;
      if (i >= 1 && j >= 1 && dp[i][j] < dp[i - 1][j - 1] + 1 && a[i] == b[j]) dp[i
        ][j] = dp[i - 1][j - 1] + 1, pa[i][j] = 2;
      if (i >= 1 && dp[i][j] < dp[i - 1][j]) dp[i][j] = dp[i - 1][j], pa[i][j] = 3;
    }
  }
}

int clcs(char *a, char *b) {
  int m = strlen(a + 1), n = strlen(b + 1), ans = 0;
  for (int i = m + 1; i <= m + m; ++i) a[i] = a[i - m];
  a[m + m + 1] = 0;
  lcs(a, b);
  ans = trace(0, 0, m, n);
  for (int i = 1; i < m; ++i) {
    reroot(i, m, n);
    ans = max(ans, trace(i, 0, m + i, n));
  }
  a[m + 1] = 0;
  return ans;
}
```

7 Tips

7.1 Useful Codes

- Accelerated C++ stream IO

```
#include <iomanip>
ios_base::sync_with_stdio(false);
```

- Enumerate all non-empty subsets

```
for (int sub = mask; sub > 0; sub = (sub - 1) & mask)
```

- Enumerate C_n^k

```
for (int comb = (1 << k) - 1; comb < 1 << n; ) {
  // ...
  int x = comb & -comb, y = comb + x;
  comb = ((comb & ~y) / x >> 1) | y;
}
```

- Convert YY/MM/DD to date

```
int days(int y, int m, int d) {
  if (m < 3) y--, m += 12;
  return 365 * y + y / 4 - y / 100 + y / 400 + (153 * m + 2) / 5 + d;
}
```

7.2 Formulas

7.2.1 Geometry

■ 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.

■ Pick's Theorem

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

S is the area of lattice polygon, I is the number of lattice interior points, and B is the number of lattice boundary points.

■ Heron's Formula

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

■ Volumes

- 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
  For ellipsoid with the standard equation in a Cartesian coordinate system $\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} + \frac{(z-z_0)^2}{c^2} = 1$, $V = \frac{4}{3}\pi abc$.

- Ellipsoid $V = \frac{4}{3}\pi abc$.
- Tetrahedron
  For tetrahedron $O - ABC$, let $a = AB, b = BC, c = CA, d = OC, e = OA, f = OB$, $(12V)^2 = a^2d^2(b^2+c^2+e^2+f^2-a^2-d^2) + b^2e^2(c^2+a^2+f^2+d^2-b^2-e^2) + c^2f^2(a^2+b^2+d^2+e^2-c^2-f^2) - a^2b^2c^2 - a^2e^2f^2 - d^2b^2f^2 - d^2e^2c^2$.

■ **Radius of Inscribedcircle & Circumcircle**

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

■ **Hypersphere**

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

■ **Affine Transformation**

$$\begin{aligned}
\mathrm{Tr} = \mathrm{TrTra} * \mathrm{TrRot} * \mathrm{TrSca} &= \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}\\
&= \begin{bmatrix} S_x\cos\alpha & -S_y\sin\alpha & T_x \\ S_x\sin\alpha & S_y\cos\alpha & T_y \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

The fixed point is (x_0, y_0), where

$$\begin{aligned}
x_0 &= -\frac{T_x(S_y\cos\alpha - 1)}{S_xS_y - S_x\cos\alpha - S_y\cos\alpha + 1} - \frac{S_yT_y\sin\alpha}{S_xS_y - S_x\cos\alpha - S_y\cos\alpha + 1}\\
y_0 &= \frac{S_xT_x\sin\alpha}{S_xS_y - S_x\cos\alpha - S_y\cos\alpha + 1} - \frac{T_y(S_x\cos\alpha - 1)}{S_xS_y - S_x\cos\alpha - S_y\cos\alpha + 1}
\end{aligned}$$

■ **Matrix of rotating θ about arbitrary axis A**

$$\begin{bmatrix} c + (1-c)A_x^2 & (1-c)A_xA_y - sA_z & (1-c)A_xA_z + sA_y \\ (1-c)A_xA_y + sA_z & c + (1-c)A_y^2 & (1-c)A_yA_z - sA_x \\ (1-c)A_xA_z - sA_y & (1-c)A_yA_z + sA_x & c + (1-c)A_z^2 \end{bmatrix}$$

7.2.2 Math

■ **Sums**

$$\begin{aligned}
1 + 2 + \ldots + n &= \frac{n^2}{2} + \frac{n}{2}\\
1^2 + 2^2 + \ldots + n^2 &= \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}\\
1^3 + 2^3 + \ldots + n^3 &= \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}\\
1^4 + 2^4 + \ldots + n^4 &= \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}\\
1^5 + 2^5 + \ldots + n^5 &= \frac{n^6}{6} + \frac{n^5}{2} + \frac{5n^4}{12} - \frac{n^2}{12}\\
1^6 + 2^6 + \ldots + n^6 &= \frac{n^7}{7} + \frac{n^6}{2} + \frac{n^5}{2} - \frac{n^3}{6} + \frac{n}{42}\\
\mathrm{P}(k) &= \frac{(n+1)^{k+1} - \sum_{i=0}^{k-1}\binom{k+1}{i}\mathrm{P}(i)}{k+1}, \mathrm{P}(0) = n\mathbf{+1}
\end{aligned}$$

$$\begin{aligned}
\sum_{k=1}^{n} k(k+1) &= \frac{n(n+1)(n+2)}{3}\\
\sum_{k=1}^{n} k(k+1)(k+2) &= \frac{n(n+1)(n+2)(n+3)}{4}\\
\sum_{k=1}^{n} k(k+1)(k+2)(k+3) &= \frac{n(n+1)(n+2)(n+3)(n+4)}{5}
\end{aligned}$$

■ **Power Reduction** $a^b \% p = a^{(b\%\varphi(p))+\varphi(p)} \% p\ (b \geq \varphi(p))$

■ **Burnside's Lemma**

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g|$$

$$\text{Polya} : X^g = t^{c(g)}$$

Let X^g denote the set of elements in X fixed by g.
$c(g)$ is the number of cycles of the group element g as a permutation of X.

■ **Lagrange Multiplier**
A strategy for finding the local extrema of a function subject to equality constraints. If we want to maximize $f(x_1, x_2, \ldots, x_n)$, which subject to $\varphi(x_1, x_2, \ldots, x_n) = 0$. We introduce a new variable λ called a Lagrange multiplier, and study the Lagrange function $L(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_n) + \lambda\varphi(x_1, x_2, \ldots, x_n)$.

Then we calculate x_i's first partial derivatives of $L(x_1, x_2, \ldots, x_n)$, and add the simultaneous equation $\varphi(x_1, x_2, \ldots, x_n) = 0$. We get these equations

$$\begin{cases} \ldots \\ f_{x_i}(x_1, x_2, \ldots, x_n) + \lambda\varphi_{x_i}(x_1, x_2, \ldots, x_n) = 0 \\ \ldots \\ \varphi(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

Solve it to get all **probable** extrema points.
Also it can extend to multiple constraints. Just set multiple Lagrange multipliers λ, ψ, etc.

■ Lucas' Theorem

For non-negative integers m and n and a prime p, holds the equation

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \mod p$$

where $m = m_k p^k + m_{k-1}p^{k-1} + \ldots + m_1 p + m_0$, and $n = n_k p^k + n_{k-1}p^{k-1} + \ldots + n_1 p + n_0$, are the base p expansions of m and n respectively.

■ Wilson's Theorem

p is a prime $\iff (p-1)! \equiv -1 \pmod p$.

■ Polynomial Congruence Equation

Solve the polynomial congruence equation $f(x) \equiv 0 \mod m$, $m = \prod_{i=1}^{k} p_i^{a_i}$.
We just simply consider the equation $f(x) \equiv 0 \mod p^a$, then use the Chinese Remainder theorem to merge the result.
If x is the root of the equation $f(x) \equiv 0 \mod p^a$, then x is also the root of $f(x) \equiv 0 \mod p^{a-1}$.
$f'(x') \equiv 0 \mod p$ **and** $f(x') \equiv 0 \mod p^a \Rightarrow x = x' + dp^{a-1} (d = 0, \ldots, p-1)$
$f'(x') \not\equiv 0 \mod p \Rightarrow x = x' - \frac{f(x')}{f'(x')}$

■ Binomial Coefficients

$$\mathrm{C}_r^k = \frac{r}{k}\mathrm{C}_{r-1}^{k-1} \qquad \mathrm{C}_r^k = \mathrm{C}_{r-1}^k + \mathrm{C}_{r-1}^{k-1}$$

$$\mathrm{C}_r^m \mathrm{C}_m^k = \mathrm{C}_r^k \mathrm{C}_{r-k}^{m-k} \qquad \sum_{k \le n} \mathrm{C}_{r+k}^k = \mathrm{C}_{r+n+1}^n$$

$$\sum_{0 \le k \le n} \mathrm{C}_k^m = \mathrm{C}_{n+1}^{m+1} \qquad \sum_k \mathrm{C}_r^k \mathrm{C}_s^{n-k} = \mathrm{C}_{r+s}^n$$

The number of non-negative solutions to equation $x_1 + x_2 + x_3 + \ldots + x_k = n$ is $\binom{n+k-1}{k-1}$.

■ Probability Distribution

- Binomial distribution: $\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, p + q = 1, \mathrm{E}[X] = np$.
- Poisson distribution: $\Pr[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}, \mathrm{E}[X] = \lambda$.
- Gaussian distribution: $p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}, \mathrm{E}[X] = \mu$.
- Geometric distribution: $\Pr[X = k] = pq^{k-1}, p + q = 1, \mathrm{E}[X] = \frac{1}{p}$.

■ Catalan Number

The number of sequences with 1 of m and -1 of n, and m$\ge n$, satifisying the constraint that any sum of the first k elements is always non-negative: $\mathrm{C}_{m+n}^m - \mathrm{C}_{m+n}^{m+1}$.
Specially, when $m = n$, it equals to $\mathrm{C}_n = \frac{\mathrm{C}_{2n}^n}{n+1}$, which is the Catalan number.
The first 10 Catalan numbers are 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, from $n = 1$, and $C_0 = 1$.
Besides, $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$, for $n \ge 0$.

■ Stirling Number of 2nd

The Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets, denoted by $S(n, k)$.
$S(n, k) = kS(n-1, k) + S(n-1, k-1)$, and $S(0, 0) = 1$, $S(n, 0) = 0$, $S(n, n) = 1$.

■ Bell Number

The Bell Number is the number of ways to partition a set of n objects into several subsets, denoted by B_n.
The first few Bell Numbers are 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975.
$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$, $B_n = \sum_{k=0}^{n} S(n, k)$, where $S(n, k)$ is Stirling Number of 2nd.
If p is a prime then, $B_{p+n} \equiv B_n + B_{n+1} (mod\, p)$, $B_{p^m+n} \equiv mB_n + B_{n+1} (mod\, p)$.

■ Derangement Number

The number of permutations of n elements with no fixed points, denotes as D_n.
$D_n = n!(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \ldots + (-1)^n \frac{1}{n!})$, or $D_n = n * D_{n-1} + (-1)^n, D_n = (n-1) * (D_{n-1} + D_{n-2})$, with $D_1 = 1, D_2 = 1$. The first 10 derangement numbers are 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 41334961, from $n = 1$.

7.2.3 Integration

■ $ax + b (a \ne 0)$

1. $\int \frac{\mathrm{d}x}{ax+b} = \frac{1}{a} \ln|ax + b| + C$
2. $\int (ax + b)^\mu \mathrm{d}x = \frac{1}{a(\mu+1)} (ax + b)^{\mu+1} + C (\mu \ne 1)$
3. $\int \frac{x}{ax+b} \mathrm{d}x = \frac{1}{a^2} (ax + b - b \ln|ax + b|) + C$

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

■ $\sqrt{ax+b}$

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. $\int \frac{\sqrt{ax+b}}{x^2}\mathrm{d}x = -\frac{\sqrt{ax+b}}{x} + \frac{a}{2}\int \frac{\mathrm{d}x}{x\sqrt{ax+b}}$

■ $x^2 \pm a^2$

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

■ $ax^2+b(a>0)$

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. $\int \frac{x}{ax^2+b}\mathrm{d}x = \frac{1}{2a}\ln\left|ax^2+b\right| + C$
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. $\int \frac{\mathrm{d}x}{(ax^2+b)^2} = \frac{x}{2b(ax^2+b)} + \frac{1}{2b}\int \frac{\mathrm{d}x}{ax^2+b}$

■ $ax^2+bx+c(a>0)$

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. $\int \frac{x}{ax^2+bx+c}\mathrm{d}x = \frac{1}{2a}\ln|ax^2+bx+c| - \frac{b}{2a}\int \frac{\mathrm{d}x}{ax^2+bx+c}$

■ $\sqrt{x^2+a^2}(a>0)$

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. $\int \frac{\mathrm{d}x}{x^2\sqrt{x^2+a^2}} = -\frac{\sqrt{x^2+a^2}}{a^2x} + C$
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. $\int \frac{\sqrt{x^2+a^2}}{x^2}\mathrm{d}x = -\frac{\sqrt{x^2+a^2}}{x} + \ln(x+\sqrt{x^2+a^2}) + C$

■ $\sqrt{x^2-a^2}(a>0)$

1. $\int \frac{\mathrm{d}x}{\sqrt{x^2-a^2}} = \frac{x}{|x|}\mathrm{arch}\frac{|x|}{a} + C_1 = \ln\left|x+\sqrt{x^2-a^2}\right| + C$
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. $\int \frac{\mathrm{d}x}{x^2\sqrt{x^2-a^2}} = \frac{\sqrt{x^2-a^2}}{a^2x} + C$
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. $\int \frac{\sqrt{x^2-a^2}}{x^2}\mathrm{d}x = -\frac{\sqrt{x^2-a^2}}{x} + \ln|x+\sqrt{x^2-a^2}| + C$

■ $\sqrt{a^2-x^2}(a>0)$

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. $\int \frac{\mathrm{d}x}{x^2\sqrt{a^2-x^2}} = -\frac{\sqrt{a^2-x^2}}{a^2x} + C$
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. $\int \frac{\sqrt{a^2-x^2}}{x^2}\mathrm{d}x = -\frac{\sqrt{a^2-x^2}}{x} - \arcsin\frac{x}{a} + C$

■ $\sqrt{\pm ax^2+bx+c}(a>0)$

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

■ $\sqrt{\pm\frac{x-a}{x-b}}$ **or** $\sqrt{(x-a)(x-b)}$

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

■ **Exponentials**

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. $\int \mathrm{e}^{ax}\cos^n bx\mathrm{d}x=\frac{1}{a^2+b^2n^2}\mathrm{e}^{ax}\cos^{n-1}bx(a\cos bx+nb\sin bx)+\frac{n(n-1)b^2}{a^2+b^2n^2}\int\mathrm{e}^{ax}\cos^{n-2}bx\mathrm{d}x$

■ **Logarithms**

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

■ **Trigonometric Functions**

1. $\int\sin x\mathrm{d}x=-\cos x+C$
2. $\int\cos x\mathrm{d}x=\sin x+C$
3. $\int\tan x\mathrm{d}x=-\ln|\cos x|+C$
4. $\int\cot x\mathrm{d}x=\ln|\sin x|+C$
5. $\int\sec x\mathrm{d}x=\ln\left|\tan\left(\frac{\pi}{4}+\frac{x}{2}\right)\right|+C=\ln|\sec x+\tan x|+C$
6. $\int\csc x\mathrm{d}x=\ln\left|\tan\frac{x}{2}\right|+C=\ln|\csc x-\cot x|+C$
7. $\int\sec^2x\mathrm{d}x=\tan x+C$
8. $\int\csc^2x\mathrm{d}x=-\cot x+C$
9. $\int\sec x\tan x\mathrm{d}x=\sec x+C$
10. $\int\csc x\cot x\mathrm{d}x=-\csc x+C$
11. $\int\sin^2x\mathrm{d}x=\frac{x}{2}-\frac{1}{4}\sin 2x+C$
12. $\int\cos^2x\mathrm{d}x=\frac{x}{2}+\frac{1}{4}\sin 2x+C$
13. $\int\sin^nx\mathrm{d}x=-\frac{1}{n}\sin^{n-1}x\cos x+\frac{n-1}{n}\int\sin^{n-2}x\mathrm{d}x$
14. $\int\cos^nx\mathrm{d}x=\frac{1}{n}\cos^{n-1}x\sin x+\frac{n-1}{n}\int\cos^{n-2}x\mathrm{d}x$
15. $\int\frac{\mathrm{d}x}{\sin^nx}=-\frac{1}{n-1}\frac{\cos x}{\sin^{n-1}x}+\frac{n-2}{n-1}\int\frac{\mathrm{d}x}{\sin^{n-2}x}$
16. $\int\frac{\mathrm{d}x}{\cos^nx}=\frac{1}{n-1}\frac{\sin x}{\cos^{n-1}x}+\frac{n-2}{n-1}\int\frac{\mathrm{d}x}{\cos^{n-2}x}$
17. $$\begin{aligned}&\int\cos^mx\sin^nx\mathrm{d}x\\=&\frac{1}{m+n}\cos^{m-1}x\sin^{n+1}x+\frac{m-1}{m+n}\int\cos^{m-2}x\sin^nx\mathrm{d}x\\=&-\frac{1}{m+n}\cos^{m+1}x\sin^{n-1}x+\frac{n-1}{m+1}\int\cos^mx\sin^{n-2}x\mathrm{d}x\end{aligned}$$
18. $\int\sin ax\cos bx\mathrm{d}x=-\frac{1}{2(a+b)}\cos(a+b)x-\frac{1}{2(a-b)}\cos(a-b)x+C$
19. $\int\sin ax\sin bx\mathrm{d}x=-\frac{1}{2(a+b)}\sin(a+b)x+\frac{1}{2(a-b)}\sin(a-b)x+C$
20. $\int\cos ax\cos bx\mathrm{d}x=\frac{1}{2(a+b)}\sin(a+b)x+\frac{1}{2(a-b)}\sin(a-b)x+C$
21. $\int\frac{\mathrm{d}x}{a+b\sin x}=\begin{cases}\frac{2}{\sqrt{a^2-b^2}}\arctan\frac{a\tan\frac{x}{2}+b}{\sqrt{a^2-b^2}}+C & (a^2>b^2)\\ \frac{1}{\sqrt{b^2-a^2}}\ln\left|\frac{a\tan\frac{x}{2}+b-\sqrt{b^2-a^2}}{a\tan\frac{x}{2}+b+\sqrt{b^2-a^2}}\right|+C & (a^2<b^2)\end{cases}$

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

23. $\int \frac{\mathrm{d}x}{a^2\cos^2 x+b^2\sin^2 x} = \frac{1}{ab}\arctan\left(\frac{b}{a}\tan x\right) + C$

24. $\int \frac{\mathrm{d}x}{a^2\cos^2 x-b^2\sin^2 x} = \frac{1}{2ab}\ln\left|\frac{b\tan x+a}{b\tan x-a}\right| + C$

25. $\int x\sin ax\mathrm{d}x = \frac{1}{a^2}\sin ax - \frac{1}{a}x\cos ax + C$

26. $\int x^2\sin ax\mathrm{d}x = -\frac{1}{a}x^2\cos ax + \frac{2}{a^2}x\sin ax + \frac{2}{a^3}\cos ax + C$

27. $\int x\cos ax\mathrm{d}x = \frac{1}{a^2}\cos ax + \frac{1}{a}x\sin ax + C$

28. $\int x^2\cos ax\mathrm{d}x = \frac{1}{a}x^2\sin ax + \frac{2}{a^2}x\cos ax - \frac{2}{a^3}\sin ax + C$

■ **Inverse Trigonometric Functions($a > 0$)**

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. $\int x^2\arctan\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arctan\frac{x}{a} - \frac{a}{6}x^2 + \frac{a^3}{6}\ln(a^2+x^2) + C$

7.3 Primes

100003, 200003, 300007, 400009, 500009, 600011, 700001, 800011, 900001,
1000003, 2000003, 3000017, 4100011, 5000011, 8000009, 9000011,
10000019, 20000003, 50000017, 50100007,
100000007, 100200011, 200100007, 250000019

7.4 Vimrc

```
 syntax on
 set guifont=consolas:h10
 set tabstop=4
 set softtabstop=4
 set shiftwidth=4
 set autoindent
 set smartindent
 set cindent
 set nu
 map <F5> :!g++ % -o %< -g -Wall<CR>:!gdb %<<CR>
 imap <F5> :!g++ % -o %< -g -Wall<CR>:!gdb %<<CR>
 :com W w
```

7.5 Makefile

```
 all : prog

 prog : prog.cpp
g++ -o prog -g prog.cpp -Wall
```

7.6 Java Fast IO

```java
import java.io.*;
import java.util.*;
import java.math.*;
public class Main {
  public static void main(String[] args) {
    InputStream inputStream = System.in;
    OutputStream outputStream = System.out;
    InputReader in = new InputReader(inputStream);
    PrintWriter out = new PrintWriter(outputStream);
    int tests = in.nextInt();
    for (int noT = 1; noT <= tests && in.hasNext(); ++noT) (new Task()).solve(noT, in, out);
    out.close();
  }
}
class Task {
  public void solve(int testNumber, InputReader in, PrintWriter out) {
    // Implementation here.
  }
}
class InputReader {
  BufferedReader reader;
  StringTokenizer tokenizer;
  public InputReader(InputStream stream) {
    reader = new BufferedReader(new InputStreamReader(stream));
    tokenizer = null;
  }
  public boolean hasNext() {
    while (tokenizer == null || !tokenizer.hasMoreTokens()) {
      try {
        tokenizer = new StringTokenizer(reader.readLine());
      } catch (Exception e) {
        return false;
      }
    }
    return tokenizer.hasMoreTokens();
  }
  public String next() {
    while (tokenizer == null || !tokenizer.hasMoreTokens()) {
      try {
        tokenizer = new StringTokenizer(reader.readLine());
      } catch (Exception e) {
        throw new RuntimeException(e);
      }
    }
    return tokenizer.nextToken();
  }
}
```