Contents

1 Data Structure

1.1 Splay

```cpp
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;
}
```

```
46 node_t *prev(node_t *p) {
47   splay(p);
48   p = p->ch[0], p->update();
49   while (p->ch[1] != nil) p = p->ch[1], p->update();
50   return p;
51 }
52
53 node_t *succ(node_t *p) {
54   splay(p);
55   p = p->ch[1], p->update();
56   while (p->ch[0] != nil) p = p->ch[0], p->update();
57   return p;
58 }
59
60 void insert(node_t *y, node_t *x) { // Insert node x after y
61   splay(y);
62   if (y->ch[1] == nil) {
63     y->ch[1] = x;
64     x->p = y;
65     y->update();
66   } else {
67     y = y->ch[1], y->update();
68     while (y->ch[0] != nil) y = y->ch[0], y->update();
69     y->ch[0] = x;
70     x->p = y;
71     y->update();
72   }
73   splay(x);
74 }
75
76 void removeAll(node_t *x) { // Remove all the whole subtree of x
77   x->p->update();
78   x->p->ch[x->dir()] = nil;
79   x->p->update();
80   splay(x->p);
81   x->p = nil;
82 }
83
84 void remove(node_t *x) { // Remove the single node x
85   node_t *p = prev(x); node_t *s = succ(x);
86   splay(p);
87   splay(s, p);
88   removeAll(s->ch[0]);
89 }
90
91 node_t *find(node_t *t, int k) {
92   t->update();
93   if (t->ch[0]->size < k && t->ch[0]->size + t->cnt >= k) return t;
```

```
94    if (t->ch[0]->size >= k) return find(t->ch[0], k);
95    return find(t->ch[1], k - t->ch[0]->size - t->cnt);
96 }
97
98 node_t *findAndSplit(node_t *t, int k) {
99    t->update();
100   if (t->ch[0]->size < k && t->ch[0]->size + t->cnt >= k) {
101     int cnt = t->cnt;
102     k -= t->ch[0]->size;
103     t->cnt = 1;
104     splay(t);
105     node_t *p = prev(t);
106     if (k - 1) insert(p, newNode(k - 1));
107     if (cnt - k) insert(t, newNode(cnt - k));
108     return t;
109   }
110   if (t->ch[0]->size >= k) return findAndSplit(t->ch[0], k);
111   return findAndSplit(t->ch[1], k - t->ch[0]->size - t->cnt);
112 }
113
114 void init() {
115   pt = 0, nil->p = nil->ch[0] = nil->ch[1] = nil;
116 }
117
118 node_t *expose(node_t *x, node_t *y) {
119   x = prev(x), y = succ(y);
120   return splay(y, splay(x))->ch[0];
121 }
```

1.2   Dynamic Tree

```
1  struct node_t {
2    node_t();
3    node_t *ch[2], *p;
4    int size, root;
5    int dir() { return this == p->ch[1]; }
6    void setc(node_t *c, int d) { ch[d] = c, c->p = this; }
7    void update() { size = ch[0]->size + ch[1]->size + 1; }
8  } s[maxn], *nil = s;
9
10 node_t::node_t() {
11   size = 1, root = true;
12   ch[0] = ch[1] = p = nil;
13 }
14
15 void rotate(node_t *t) {
16   node_t *p = t->p;
17   int d = t->dir();
```

```
18   if (!p->root) {
19     p->p->setc(t, p->dir());
20   } else {
21     p->root = false, t->root = true;
22     t->p = p->p; // Path Parent
23   }
24   p->setc(t->ch[!d], d);
25   t->setc(p, !d);
26   p->update(), t->update();
27 }
28
29 void splay(node_t *t) {
30   // t->update(); // tag!
31   while (!t->root) {
32     // if (!t->p->root) t->p->p->update(); t->p->update(), t->update(); // !
33     if (!t->p->root) rotate(t->dir() == t->p->dir() ? t->p : t);
34     rotate(t);
35   }
36 }
37
38 void access(node_t *x) { // Ask u, v: access(u), access(v, true), x = LCA
39   node_t *y = nil;
40   while (x != nil) {
41     splay(x);
42     // if (x->p == nil) at second call, x->ch[1](rev) + (x)_single + y
43     x->ch[1]->root = true;
44     x->ch[1] = y, y->root = false;
45     x->update();
46     y = x, x = x->p;
47   }
48 }
49
50 void cut(node_t *x) {
51   access(x);
52   splay(x);
53   x->ch[0]->root = true;
54   x->ch[0]->p = nil;
55   x->ch[0] = nil;
56 }
57
58 void link(node_t *x, node_t *y) {
59   access(y);
60   splay(y);
61   y->p = x;
62   access(y);
63 }
64
65 void init() { nil->size = 0; }
```

1.3   KD Tree

```
1  //如果被卡可以考虑写上 minx,maxx,miny,maxy 维护矩形，修改 KDTree_Build
      加上对应的维护。
2  struct POINT { int x,y,id; };
3  inline bool cmp_x(const POINT& a,const POINT& b) { return a.x == b.x ? a.y < b.y : a
      .x < b.x; }
4  inline bool cmp_y(const POINT& a,const POINT& b) { return a.y == b.y ? a.x < b.x : a
      .y < b.y; }
5
6  struct KDNODE {
7    POINT p;
8  //   int minx,maxx,miny,maxy;
9
10   KDNODE* Child[2], *fa;
11 };
12 KDNODE NPool[111111];
13 KDNODE* NPTop = NPool;
14 KDNODE* Root;
15
16 inline KDNODE* AllocNode() { memset(NPTop,0,sizeof(KDNODE)); return NPTop++; }
17 inline ll PDist(const POINT& a,const POINT& b) { return sqr((ll)(a.x-b.x))+sqr((ll)(
      a.y-b.y)); }
18
19 POINT pnt[111111];
20 KDNODE* KDTree_Build(int l,int r,int depth=0) {
21   if(l >= r) return NULL;
22
23   if(depth&1) sort(pnt+l,pnt+r,cmp_y);
24   else sort(pnt+l,pnt+r,cmp_x);
25
26   int mid = (l+r)/2;
27   KDNODE* t = AllocNode();
28
29   t->Child[0] = KDTree_Build(l,mid,depth+1);
30   t->Child[1] = KDTree_Build(mid+1,r,depth+1);
31   for(int i = 0;i < 2;i++) if(t->Child[i]) t->Child[i]->fa = t;
32   return t;
33 }
34
35 void KDTree_Insert(KDNODE* cur,POINT& P,int depth=0) {
36   KDNODE* node = AllocNode(); node->p = P;
37   while(cur) {
38     if(cur->p.x == P.x && cur->p.y == P.y && cur->p.id == P.id) break;
39     int dir = 0;
40     if(depth&1) dir = cmp_y(x->p,P);
41     else dir = cmp_x(x->p,P);
42     if(!cur->Child[dir]) {
43       cur->Child[dir] = node;
```

```
44        node->fa = cur;
45        break;
46      } else {
47        cur = cur->Child[dir];
48        depth++;
49      }
50    }
51  }
52
53  ll KDTree_Nearest(KDNODE* x,const POINT& q,int depth=0) {
54    KDNODE* troot = x->fa;
55    int dir = 0;
56    while(x) {
57      if(depth&1) dir = cmp_y(x->p,q);
58      else dir = cmp_x(x->p,q);
59
60      if(!x->Child[dir]) break;
61      x = x->Child[dir];
62      depth++;
63    }
64    ll ans = ~0ULL>>1;
65    while(x != troot) {
66      ll tans = PDist(q,x->p);
67      if(tans < ans) ans = tans;
68      KDNODE* oside = x->Child[dir^1];
69      if(oside) {
70        ll ldis = 0;
71        /*if(depth&1) ldis = min(sqr((ll)q.y-oside->miny),sqr((ll)q.y-oside->maxy));
72        else ldis = min(sqr((ll)q.x-oside->minx),sqr((ll)q.x-oside->maxx));*/
73        if(depth & 1) ldis = sqr<ll>(x->p.y-q.y);
74        else ldis = sqr<ll>(x->p.x-q.x);
75        if(ldis < ans) {
76          tans = KDTree_Nearest(oside,q,depth+1);
77          if(tans && tans < ans) ans = tans;
78        }
79      }
80
81      if(x->fa && x == x->fa->Child[0]) dir = 0;
82      else dir = 1;
83      x = x->fa;
84      depth--;
85    }
86    return ans;
87  }
```

1.4 Treap

```
1  struct node {
2    int v, key, size;
3    node *c[2];
4    void resize() { size = c[0]->size + c[1]->size + 1; }
5  };
6  node *newNode(int _v, node *n) {
7    ++ref;
8    pool[ref].v = _v, pool[ref].c[0] = pool[ref].c[1] = n, pool[ref].size = 1, pool[
       ref].key = rand();
9    return &pool[ref];
10  }
11  struct Treap {
12    node *root, *nil;
13    void rotate(node *&t, int d) {
14      node *c = t->c[d];
15      t->c[d] = c->c[!d];
16      c->c[!d] = t;
17      t->resize(); c->resize();
18      t = c;
19    }
20    void insert(node *&t, int x) {
21      if (t == nil) t = newNode(x, nil);
22      else {
23        if (x == t->v) return;
24        int d = x > t->v;
25        insert(t->c[d], x);
26        if (t->c[d]->key < t->key) rotate(t, d);
27        else t->resize();
28      }
29    }
30    void remove(node *&t, int x) {
31      if (t == nil) return;
32      if (t->v == x) {
33        int d = t->c[1]->key < t->c[0]->key;
34        if (t->c[d] == nil) {
35          t = nil;
36          return;
37        }
38        rotate(t, d);
39        remove(t->c[!d], x);
40      } else {
41        int d = x > t->v;
42        remove(t->c[d], x);
43      }
44      t->resize();
45    }
46    int rank(node *t, int x) {
47      if (t == nil) return 0;
48      int r = t->c[0]->size;
```

```
49      if (x == t->v) return r + 1;
50      if (x < t->v) return rank(t->c[0], x);
51      return r + 1 + rank(t->c[1], x);
52    }
53    int select(node *t, int k) {
54      int r = t->c[0]->size;
55      if (k == r + 1) return t->v;
56      if (k <= r) return select(t->c[0], k);
57      return select(t->c[1], k - r - 1);
58    }
59    int size() {
60      return root->size;
61    }
62    void init(int *a, int n) {
63      nil = newNode(0, 0);
64      nil->size = 0, nil->key = ~0U >> 1;
65      root = nil;
66    }
67 };
```

1.5 President Treap

```
1  struct node_t {
2    int key, cnt, size;
3    string v;
4    node_t *c[2];
5    void resize() {
6      size = (c[0] ? c[0]->size : 0) + cnt + (c[1] ? c[1]->size : 0);
7    }
8  } *nil;
9
10 node_t *newNode(string v, node_t *l = nil, node_t *r = nil, int key = rand()) {
11   node_t *ret = new node_t();
12   ret->key = key;
13   ret->cnt = v.length(), ret->v = v;
14   ret->c[0] = l, ret->c[1] = r;
15   ret->resize();
16   return ret;
17 }
18
19 void init() {
20   nil = newNode("", 0, 0);
21   nil->size = 0, nil->key = ~0U >> 1;
22 }
23
24 struct PresidentTreap {
25   node_t *root;
26   node_t *splitL(node_t *a, int size) {
```

```
27     if (a == nil || size == 0) return nil;
28     if (a->c[0]->size >= size) return splitL(a->c[0], size);
29     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);
30     return newNode(a->v, a->c[0], splitL(a->c[1], size - a->c[0]->size - a->cnt), a
           ->key);
31   }
32   node_t *splitR(node_t *a, int size) {
33     if (a == nil || size == 0) return nil;
34     if (a->c[1]->size >= size) return splitR(a->c[1], size);
35     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);
36     return newNode(a->v, splitR(a->c[0], size - a->c[1]->size - a->cnt), a->c[1], a
           ->key);
37   }
38   node_t *merge(node_t *a, node_t *b) {
39     if (a == nil) return b;
40     if (b == nil) return a;
41     if (a->key > b->key) return newNode(a->v, a->c[0], merge(a->c[1], b), a->key);
42     return newNode(b->v, merge(a, b->c[0]), b->c[1], b->key);
43   }
44   node_t *insert(string v, int p) { // insert after p
45     int l = root->size;
46     return merge(merge(splitL(root, p), newNode(v, nil, nil)), splitR(root, l - p));
47   }
48   node_t *remove(int x, int y) { // remove [x, y]
49     int l = root->size;
50     return merge(splitL(root, x - 1), splitR(root, l - y));
51   }
52 };
```

1.6 President Segment Tree

```
1  struct Node {
2    int s, d;
3    Node *left, *right;
4  } pool[maxm], *nil, *root[maxn];
5  int pt, a[maxn];
6
7  Node *newNode(int _d, int _s, Node *_left, Node *_right) {
8    ++pt;
9    pool[pt].d = _d, pool[pt].s = _s, pool[pt].left = _left, pool[pt].right = _right;
10   return pool + pt;
11 }
12
13 Node *build(int l, int r) {
14   if (l == r) return newNode(0, a[l], nil, nil);
15   int mid = (l + r) / 2;
```

```
16    Node *nl = build(l, mid), *nr = build(mid + 1, r);
17    return newNode(0, nl->s + nr->s, nl, nr);
18  }
19
20  void init(int n) {
21    pt = 0; nil = newNode(0, 0, NULL, NULL);
22    root[0] = build(1, n);
23  }
24
25  void push(Node *node, int l, int r) {
26    if (l == r) {
27      node->d = 0;
28    } else {
29      if (node->d == 0) return;
30      int mid = (l + r) / 2;
31      Node *nl = newNode(node->left->d + node->d, node->left->s + node->d * int(mid -
          l + 1), node->left->left, node->left->right);
32      Node *rl = newNode(node->right->d + node->d, node->right->s + node->d * int(r -
          mid), node->right->left, node->right->right);
33      node->d = 0;
34      node->left = nl;
35      node->right = rl;
36    }
37  }
38
39  int ask(Node *node, int l, int r, int ll, int rr) {
40    push(node, l, r);
41    if (l == ll && r == rr) return node->s;
42    int mid = (l + r) / 2;
43    if (rr <= mid) return ask(node->left, l, mid, ll, rr);
44    else if (ll > mid) return ask(node->right, mid + 1, r, ll, rr);
45    else return ask(node->left, l, mid, ll, mid) + ask(node->right, mid +
        1, rr);
46  }
47
48  Node *add(Node *node, int l, int r, int ll, int rr, int d) {
49    push(node, l, r);
50    if (l == ll && r == rr) return newNode(node->d + d, node->s + d * int(r - l + 1),
        node->left, node->right);
51    int mid = (l + r) / 2;
52    if (rr <= mid) {
53      Node *nl = add(node->left, l, mid, ll, rr, d);
54      return newNode(0, nl->s + node->right->s, nl, node->right);
55    } else if (ll > mid) {
56      Node *nr = add(node->right, mid + 1, r, ll, rr, d);
57      return newNode(0, node->left->s + nr->s, node->left, nr);
58    } else {
59      Node *nl = add(node->left, l, mid, ll, mid, d);
60      Node *nr = add(node->right, mid + 1, r, mid + 1, rr, d);
61      return newNode(0, nl->s + nr->s, nl, nr);
62    }
63  }
```

1.7   Merge-Split Treap

```
1   struct TNODE {
2     int val,rd,size;
3     TNODE* left,*right,*fa;
4     inline int update() {
5       size = 1;
6       if(left) { size += left->size; left->fa = this; }
7       if(right) { size += right->size; right->fa = this; }
8       fa = NULL;
9       return 0;
10    }
11  };
12  typedef pair<TNODE*,TNODE*> ptt;
13  TNODE TPool[233333];
14  TNODE* TPTop = TPool;
15
16  inline int real_rand() { return ((rand()&32767)<<15)^rand(); }
17  TNODE* newNode(int val,TNODE* left=NULL,TNODE* right=NULL) {
18    TNODE* result = TPTop++;
19    result->val = val; result->rd = real_rand();
20    result->left = left; result->right = right; result->fa = NULL;
21    result->update();
22    return result;
23  }
24
25  TNODE* Merge(TNODE* t1,TNODE* t2) {
26    if(!t1) return t2;
27    if(!t2) return t1;
28    if(t1->rd <= t2->rd) { t1->right = Merge(t1->right,t2); t1->update(); return t1; }
29    else { t2->left = Merge(t1,t2->left); t2->update(); return t2; }
30  }
31
32  ptt Split(TNODE* x,int pos) {
33    if(pos == 0) return ptt(NULL,x);
34    if(pos == x->size) return ptt(x,NULL);
35
36    int lsize = x->left ? x->left->size : 0;
37    int rsize = x->right ? x->right->size : 0;
38    if(lsize == pos) {
39      TNODE* oleft = x->left;
40      if(x->left) x->left->update();
41      x->left = NULL;
```

```
42     x->update();
43     return ptt(oleft,x);
44   }
45   if(pos < lsize) {
46     ptt st = Split(x->left,pos);
47     x->left = st.second; x->update(); if(st.first) st.first->update();
48     return ptt(st.first,x);
49   } else {
50     ptt st = Split(x->right,pos-lsize-1);
51     x->right = st.first; x->update(); if(st.second) st.second->update();
52     return ptt(x,st.second);
53   }
54 }
55
56 inline int Rank(TNODE* x) {
57   int ans = x->left ? x->left->size : 0;
58   for(;x->fa;x = x->fa)
59     if(x == x->fa->right) ans += (x->fa->left ? x->fa->left->size : 0) + 1;
60   return ans;
61 }
```

2   String Algorithms

2.1   Common

```
1 // please note that all strings are indexed from 0
2 void kmp(const char *s, int *next)
3 {
4   --s, --next;
5   next[1] = 0;
6   int j = 0, n = strlen(s + 1);
7   for (int i = 2; i <= n; ++i)
8   {
9     while (j > 0 && s[j + 1] != s[i]) j = next[j];
10     if (s[j + 1] == s[i]) j = j + 1;
11     next[i] = j;
12   }
13 }
14
15 // s: text, t: text being searched, ex[i]: maximum l satisfying s[i...i+l-1] = t
      [0...l-1]
16 void exkmp(const char *s, const char *t, int *next, int *ex)
17 {
18   int n = strlen(t), m = strlen(s), k, c;
19   next[0] = n;
20   k = 0, c = 1;
21   while (k + 1 < n && t[k] == t[k + 1]) ++k;
22   next[1] = k;
23   for (int i = 2; i < n; ++i)
24   {
25     int p = next[c] + c - 1;
26     int l = next[i - c];
27     if (i + l < p + 1) next[i] = l;
28     else
29     {
30       k = max(0, p - i + 1);
31       while (i + k < n && t[i + k] == t[k]) ++k;
32       next[i] = k;
33       c = i;
34     }
35   }
36   k = c = 0;
37   while (k < m && k < n && s[k] == t[k]) ++k;
38   ex[0] = k;
39   for (int i = 1; i < m; ++i)
40   {
41     int p = ex[c] + c - 1;
42     int l = next[i - c];
43     if (l + i < p + 1) ex[i] = next[i - c];
44     else
45     {
46       k = max(0, p - i + 1);
47       while (i + k < m && k < n && s[i + k] == t[k]) ++k;
48       ex[i] = k;
49       c = i;
50     }
51   }
52 }
53
54 // minimum representation of a string
55 int minimum_representation(string s)
56 {
57   s += s;
58   int l = s.length(), i = 0, j = 1, k = 0;
59   while (i + k < l && j + k < l)
60   {
61     if (s[i + k] == s[j + k])
62       ++k;
63     else
64     {
65       if (s[j + k] > s[i + k])
66         j += k + 1;
67       else
68         i += k + 1;
69       k = 0;
```

```
70        if (i == j)
71          ++j;
72      }
73    }
74    return min(i, j);
75 }
76
77 // l[i], the length of palindrome at the centre of i
78 int manacher(const char *s, int *l)
79 {
80   int n = strlen(s);
81   for (int i = 0, j = 0, k; i < n * 2; i += k, j = max(j - k, 0))
82   {
83     while (i >= j && i + j + 1 < n * 2 && s[(i - j) / 2] == s[(i + j + 1) / 2])
84       ++j;
85     l[i] = j;
86     for (k = 1; i >= k && j >= k && l[i - k] != j - k; ++k)
87       l[i + k] = min(l[i - k], j - k);
88   }
89   return *max_element(l, l + n + n);
90 }
```

2.2   Aho-Crosick Automaton

```
1 struct trie_t {
2   bool flag;
3   trie_t *child[C], *fail;
4 } trie[maxn], *root;
5 trie_t *new_trie() { return &trie[++pt]; }
6
7 void add(char *str) {
8   int l = strlen(str);
9   trie_t *p = root;
10   for (int i = 0; i < l; ++i) {
11     int ch = str[i]; // fixed to [0, C - 1], C = |SIGMA|
12     if (!p->child[ch]) p->child[ch] = new_trie();
13     p = p->child[ch];
14   }
15   p->flag = true;
16 }
17
18 void build() {
19   queue<trie_t *> q;
20   q.push(root);
21   while (!q.empty()) {
22     trie_t *p = q.front(), *t;
23     q.pop();
24     for (int i = 0; i < C; ++i) {
```

```
25       t = p->fail;
26       while (t && !t->child[i]) t = t->child[i];
27       t = !t ? root : t->child[i];
28       if (p->child[i]) {
29         p->child[i]->fail = t;
30         p->child[i]->flag |= t->flag;
31         q.push(p->child[i]);
32       } else p->child[i] = t;
33     }
34   }
35 }
```

2.3   Suffix Array

```
1 const int maxn = 100002, logn = 21, maxint = 0x7f7f7f7f;
2 int n, sa[maxn], r[maxn + maxn], h[maxn], mv[maxn][logn];
3 void initlg() {
4   _lg[1] = 0;
5   for (int i = 2; i < maxn; ++i)
6     _lg[i] = _lg[i - 1] + ((i & (i - 1)) == 0 ? 1 : 0);
7 }
8 void da(int *r, int *sa, int n, int m) //r[n] = 0!!
9 {
10   int i, j, p, *x = wa, *y = wb, *t;
11   for(i = 0; i < m; i++) ws[i] = 0;
12   for(i = 0; i < n; i++) ws[x[i] = r[i]]++;
13   for(i = 1; i < m; i++) ws[i] += ws[i-1];
14   for(i = n-1; i >= 0; i--) sa[--ws[x[i]]] = i;
15   for(j = 1, p = 1; p < n; j *= 2, m = p)
16   {
17     for(p = 0, i = n - j; i < n; i++) y[p++] = i;
18     for(i = 0; i < n; i++) if(sa[i] >= j) y[p++] = sa[i] - j;
19     for(i = 0; i < n; i++) wv[i] = x[y[i]];
20     for(i = 0; i < m; i++) ws[i] = 0;
21     for(i = 0; i < n; i++) ws[wv[i]]++;
22     for(i = 1; i < m; i++) ws[i] += ws[i-1];
23     for(i = n - 1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
24     t = x; x = y; y = t;
25     p = 1;
26     x[sa[0]] = 0;
27     for (i = 1; i < n; i++)
28       x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + j] == y[sa[i - 1] + j]) ? p
              -1 : p++;
29   }
30   return;
31 }
32 int height[maxn], rank[maxn];     //height[2..n]
33 void calheight(int *r, int *sa)
```

```
34 {
35   for (int i = 1; i <= n; i++)
36     rank[sa[i]] = i;
37   int j, k = 0;
38   for (int i = 0; i < n; height[rank[i++]] = k)
39     for (k ? k-- : 0, j = sa[rank[i] - 1]; r[i+k] == r[j+k]; k++);
40 }
41 int askRMQ(int l, int r) {
42   int len = r - l + 1, log = _lg[r - l + 1];
43   return min(mv[l][log], mv[r - (1 << log) + 1][log]);
44 }
45
46 int LCP(int i, int j) {
47   i = r[i], j = r[j];
48   if (i > j) swap(i, j);
49   return askRMQ(++i, j);
50 }
```

2.4 Suffix Automation

```
1  SAMNODE* Root,*Last; // take care, init them
2  int append_char(int ch) {
3    SAMNODE* x = Last, t = SPTop++;
4    t->len = x->len+1;
5    for(;x && !x->child[ch];x = x->fa) x->child[ch] = t;
6    if(!x) t->fa = Root;
7    else {
8      SAMNODE* bro = x->child[ch];
9      if(x->len+1 == bro->len) t->fa = bro; // actually it's fa.
10     else {
11       SAMNODE* nfa = SPTop++;
12       nfa[0] = bro[0];
13       nfa->len = x->len+1;
14       bro->fa = t->fa = nfa;
15
16       for(;x && x->child[ch] == bro;x = x->fa) x->child[ch] = nfa;
17     }
18   }
19   Last = t;
20   return 0;
21 }
22
23 // SAM::Match //
24 SAMNODE* x = Root;
25 int mlen = 0;
26 for(int j = 0;j < len;j++) {
27   int ch = Text[j];
28   /*// 强制后撤一个字符，部分情况下可能有用
```

```
29   if(mlen == qlen) {
30     mlen--;
31     while(mlen <= x->fa->len) x = x->fa;
32   } */
33   if(x->child[ch]) { mlen++; x = x->child[ch]; }
34   else {
35     while(x && !x->child[ch]) x = x->fa;
36     if(!x) {
37       mlen = 0;
38       x = Root;
39     } else {
40       mlen = x->len+1;
41       x = x->child[ch];
42     }
43   }
44   Match[j] = mlen;
45 } // End of SAM::Match //
46
47 // 基排方便上推一些东西，比如出现次数 //
48 SAMNODE* order[2222222];
49 int lencnt[1111111];
50 int post_build(int len) {
51   for(SAMNODE* cur = SPool;cur < SPTop;cur++) lencnt[cur->len]++;
52   for(int i = 1;i <= len;i++) lencnt[i] += lencnt[i-1];
53   int ndcnt = lencnt[len];
54   for(SAMNODE* cur = SPTop-1;cur >= SPool;cur--) order[--lencnt[cur->len]] = cur;
55   for(int i = ndcnt-1;i >= 0;i--) {
56     // 此处上推
57     if(order[i]->fa) order[i]->fa->cnt += order[i]->cnt;
58   }
59   return 0;
60 }
```

2.5 Palindromic Tree

所谓的 Palindrome Tree 其实是每个点表示了一个回文子串，而边则是表示在两侧同时添加上这个字母可以得到的新回文子串。从点 u 到点 w 的 suffix link 表示 w 是 u 的所有不是 u 本身的后缀中最长的回文子串。这个所谓的 "Tree" 实际上有两个根。一个表示 −1 长度的串，用于表示只有一个字母的新回文串的产生，一个表示空串。两个根的 suffix link 都指向 -1 根。有编号大的点就是拓扑序小的点这个性质。

　一些应用：

1. 统计加入一个字母的时候增加了多少个新的（不同的）回文串：看看加入字母的时候多出来几个点就行了。答案只可能是 0 或 1。

2. 计算回文子串个数：注意到 Suffix Link 关系是棵树（两个根两棵树），对每个点维护它到根的连接数，然后对于新加入的点加上它的连接数即可（考虑 Suffix Link 关系的意义，显然）。

3. 计算每个不同的子回文串的出现次数: 基本同上，注意到对于新加入的点它是对本身和 Suffix Link 上的所有点贡献了 1 的答案，于是上推一遍即可。

```
1  struct Palindromic_Tree {
2    int next[maxn][N], fail[maxn], cnt[maxn], num[maxn], len[maxn];
3    int S[maxn];
4    int last, n, p;
5
6    int newnode(int l) {
7      for (int i = 0; i < N; ++i) next[p][i] = 0;
8      cnt[p] = 0;
9      num[p] = 0;
10     len[p] = l;
11     return p++;
12   }
13
14   void init() {
15     p = 0;
16     newnode(0);
17     newnode(-1);
18     n = last = 0;
19     S[n] = -1;
20     fail[0] = 1;
21   }
22
23   void add(int c) {
24     c -= 'a';
25     S[++n] = c;
26     int cur = last;
27     while (S[n - len[cur] - 1] != S[n])
28       cur = fail[cur];
29     if (!next[cur][c]) {
30       int now = newnode(len[cur] + 2);
31       int x = fail[cur];
32       while (S[n - len[x] - 1] != S[n])
33         x = fail[x];
34       fail[now] = next[x][c];
35       next[cur][c] = now;
36       num[now] = num[fail[now]] + 1;
37     }
38     last = next[cur][c];
39     cnt[last]++;
40   }
41
42   void count() {
43     for (int i = p - 1; i >= 0; --i) cnt[fail[i]] += cnt[i];
44   }
45 };
```

2.6   Cyclic LCS

```
1  const int maxn = 3001;
2  int dp[maxn][maxn], pa[maxn][maxn];
3
4  int trace(int sx, int sy, int ex, int ey) {
5    int l = 0;
6    while (ex != sx || ey != sy) {
7      if (pa[ex][ey] == 1) --ey;
8      else if (pa[ex][ey] == 2) --ex, --ey, ++l;
9      else --ex;
10   }
11   return l;
12 }
13
14 void reroot(int root, int m, int n) {
15   int i = root, j = 1;
16   while (j <= n && pa[i][j] != 2) ++j;
17   if (j > n) return;
18   pa[i][j] = 1;
19   while (i < 2 * m && j < n) {
20     if (pa[i + 1][j] == 3) pa[++i][j] = 1;
21     else if (pa[i + 1][j + 1] == 2) pa[++i][++j] = 1;
22     else ++j;
23   }
24   while (i < 2 * m && pa[i + 1][j] == 3) pa[++i][j] = 1;
25 }
26
27 void lcs(char *a, char *b) {
28   int m = strlen(a + 1), n = strlen(b + 1);
29   for (int i = 0; i <= m; ++i) {
30     for (int j = 0; j <= n; ++j) {
31       if (i != 0 || j != 0) dp[i][j] = -1;
32       if (j >= 1 && dp[i][j] < dp[i][j - 1]) dp[i][j] = dp[i][j - 1], pa[i][j] = 1;
33       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;
34       if (i >= 1 && dp[i][j] < dp[i - 1][j]) dp[i][j] = dp[i - 1][j], pa[i][j] = 3;
35     }
36   }
37 }
38
39 int clcs(char *a, char *b) {
40   int m = strlen(a + 1), n = strlen(b + 1), ans = 0;
41   for (int i = m + 1; i <= m + m; ++i) a[i] = a[i - m];
42   a[m + m + 1] = 0;
43   lcs(a, b);
44   ans = trace(0, 0, m, n);
45   for (int i = 1; i < m; ++i) {
46     reroot(i, m, n);
```

```
47      ans = max(ans, trace(i, 0, m + i, n));
48    }
49    a[m + 1] = 0;
50    return ans;
51 }
```

3 Math

3.1 Matrix Multiplication

```
1  struct matrix_t {
2    int x[N + 1][N + 1];
3    matrix_t(int v) {
4      memset(x, 0, sizeof(x));
5      for (int i = 1; i <= N; ++i) x[i][i] = v;
6    }
7    matrix_t operator*(const matrix_t &r) {
8      matrix_t p = 0;
9      for (int k = 1; k <= N; ++k) {
10       for (int i = 1; i <= N; ++i) {
11         if (x[i][k] == 0) continue;
12         for (int j = 1; j <= N; ++j) {
13           p.x[i][j] += x[i][k] * r.x[k][j];
14           p.x[i][j] %= MOD;
15         }
16       }
17     }
18     return p;
19   }
20   matrix_t power(LL p) {
21     matrix_t r = 1, a = *this;
22     for (; p; p >>= 1) {
23       if (p & 1) r = r * a;
24       a = a * a;
25     }
26     return r;
27   }
28 };
```

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

```
1  void gauss(int n, double g[maxn][maxn]) { // input: N * (N + 1) Matrix
2    for (int i = 1; i <= n; ++i) {
3      double temp = 0;
4      int pos = -1;
5      for (int j = i; j <= n; ++j) {
6        if (fabs(g[j][i]) > temp) temp = fabs(g[j][i]), pos = j;
7      }
8      if (pos == -1) continue;
9      for (int k = 1; k <= n + 1; ++k) swap(g[pos][k], g[i][k]);
10     temp = g[i][i];
11     for (int k = 1; k <= n + 1; ++k) g[i][k] /= temp;
12     for (int j = i + 1; j <= n; ++j) {
13       temp = g[j][i];
14       for (int k = 1; k <= n + 1; ++k) g[j][k] -= temp * g[i][k];
15     }
16   }
17   for (int i = n; i >= 1; --i) {
18     for (int j = 1; j < i; ++j) {
```

```
19        g[j][n + 1] -= g[i][n + 1] * g[j][i];
20        g[j][i] = 0;
21      }
22    }
23 }
24 // n is the number of variables, t is the number of equations
25 // index from 1
26 // return the number of free variables
27 int gauss()
28 {
29   int pos;
30   int i = 1, j = 1;
31   while (i <= t && j <= n)
32   {
33     pos = i;
34     for (int k = i; k <= t; k++)
35       if (a[k][j] != 0)
36       {
37         pos = k;
38         break;
39       }
40     if (a[pos][j] > 0)
41     {
42       if (pos != i)
43         for (int k = 1; k <= n; k++)
44           swap(a[i][k], a[pos][k]);
45       for (int p = i+1; p <= t; p++)
46         if (a[p][j] > 0)
47         {
48           for (int k = j; k <= n; k++)
49             a[p][k] ^= a[i][k];
50         }
51       i++;
52     }
53     j++;
54   }
55   return n - i + 1;
56 }
```

3.3 Linear Dependency

求线性无关方程组，本质是个消元，不过按照常用的形式进行了压位（这里是 31 位）。
可以顺便维护出一组基。

```
1 for(int i = 0;i < n;i++) {
2   for(int j = 31;j >= 0;j--) {
3     if(xx[i] & (1LL<<j)) {
4       if(!ind[j]) { ind[j] = xx[i]; break; }
```

```
5       else xx[i] ^= ind[j];
6     }
7   }
8 }
```

3.4 Determinant

```
1 LL determinant() {
2   LL result = 1;
3   for (int i = 1; i <= n; ++i) {
4     for (int j = i + 1; j <= n; ++j) {
5       while (det[j][i]) {
6         LL ratio = det[i][i] / det[j][i];
7         for (int k = i; k <= n; ++k) {
8           det[i][k] -= ratio * det[j][k];
9           swap(det[i][k], det[j][k]);
10        }
11        result = -result;
12      }
13    }
14    result = result * det[i][i];
15  }
16  return result;
17 }
```

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.

3.5 Polynomial Root

```
1 double cal(const vector<double> &coef, double x) {
2   double e = 1, s = 0;
3   for (int i = 0; i < coef.size(); ++i) s += coef[i] * e, e *= x;
4   return s;
5 }
6
7 double find(const vector<double> &coef, double l, double r) {
8   int sl = dblcmp(cal(coef, l)), sr = dblcmp(cal(coef, r));
9   if (sl == 0) return l;
```

```
10    if (sr == 0) return r;
11    if (sl * sr > 0) return maxdbl;
12    for (int tt = 0; tt < 100 && r - l > eps; ++tt) {
13      double mid = (l + r) / 2;
14      int smid = dblcmp(cal(coef, mid));
15      if (smid == 0) return mid;
16      if (sl * smid < 0) r = mid;
17      else l = mid;
18    }
19    return (l + r) / 2;
20  }
21
22  vector<double> solve(vector<double> coef, int n) {
23    vector<double> ret; // c[0]+c[1]*x+c[2]*x^2+...+c[n]*x^n
24    if (n == 1) {
25      if (dblcmp(coef[1]) != 0) ret.push_back(-coef[0] / coef[1]);
26      return ret;
27    }
28    vector<double> dcoef(n);
29    for (int i = 0; i < n; ++i) dcoef[i] = coef[i + 1] * (i + 1);
30    vector<double> droot = solve(dcoef, n - 1);
31    droot.insert(droot.begin(), -maxdbl);
32    droot.push_back(maxdbl);
33    for (int i = 0; i + 1 < droot.size(); ++i) {
34      double tmp = find(coef, droot[i], droot[i + 1]);
35      if (tmp < maxdbl) ret.push_back(tmp);
36    }
37    return ret;
38  }
```

3.6   Number Theory Library

```
1   LL mult64(LL a, LL b, LL m) { // 64bit multiply 64bit
2     a %= m, b %= m;
3     LL ret = 0;
4     for (; b; b >>= 1) {
5       if (b & 1) ret = (ret + a) % m;
6       a = (a + a) % m;
7     }
8     return ret;
9   }
10  /* return x*y%mod. no overflow if x,y < mod
11   * remove 'i' in "idiv"/"imul" -> unsigned */
12  inline long mulmod(long x,long y,long mod)
13  {
14    long ans = 0;
15    __asm__ (
16      "movq %1,%%rax\n imulq %2\n idivq %3\n"
```

```
17      :"=d"(ans):"m"(x),"m"(y),"m"(mod):"%rax"
18    );
19    return ans;
20  }
21
22  LL fpow(LL a, LL p, int mod) { // fast power-modulo algorithm
23    LL res = 1;
24    for (; p; p >>= 1) {
25      if (p & 1) res = (res * a) % mod; // using mult64 when mod is 64-bit
26      a = (a * a) % mod;
27    }
28    return res;
29  }
30
31  int exgcd(int x, int y, int &a, int &b) { // extended gcd, ax + by = g.
32    int a0 = 1, a1 = 0, b0 = 0, b1 = 1;
33    while (y != 0) {
34      a0 -= x / y * a1; swap(a0, a1);
35      b0 -= x / y * b1; swap(b0, b1);
36      x %= y; swap(x, y);
37    }
38    if (x < 0) a0 = -a0, b0 = -b0, x = -x;
39    a = a0, b = b0;
40    return x;
41  }
42
43  int inverse(int x, int mod) { // multiplicative inverse.
44    int a = 0, b = 0;
45    if (exgcd(x, mod, a, b) != 1) return -1;
46    return (a % mod + mod) % mod; // C1: x & mod are co-prime
47    return fpow(x, mod - 2, mod); // C2: mod is prime
48  }
49
50  void init_inverse(int mod) { // O(n), all multiplicative inverse, mod is prime
51    inv[0] = inv[1] = 1;
52    for (int i = 2; i < n; ++i) {
53      inv[i] = (LL)inv[mod % i] * (mod - mod / i) % mod; // overflows?
54    }
55  }
56
57  LL CRT(int cnt, int *p, int *b) { // chinese remainder theorem
58    LL N = 1, ans = 0;
59    for (int i = 0; i < k; ++i) N *= p[i];
60    for (int i = 0; i < k; ++i) {
61      LL mult = (inverse(N / p[i], p[i]) * (N / p[i])) % N;
62      mult = (mult * b[i]) % N;
63      ans += mult; ans %= N;
64    }
```

```
 65      if (ans < 0) ans += N;
 66      return ans;
 67  }
 68
 69  void sieve(int n) { // generating primes using euler's sieve
 70      notP[1] = 1;
 71      for (int i = 2; i <= n; ++i) {
 72          if (!notP[i]) P[++Pt] = i;
 73          for (int j = 1; j <= Pt && P[j] * i <= n; ++j) {
 74              notP[P[j] * i] = 1;
 75              if (i % P[j] == 0) break;
 76          }
 77      }
 78  }
 79  void sieve(int n)
 80  {
 81      memset(isP,0,sizeof(isP));
 82      mu[1] = 1;
 83      phi[1] = 0;
 84      for (int i = 2; i <= n; i++)
 85      {
 86          if (isP[i] == 0)
 87          {
 88              isP[i] = 1;
 89              mu[i] = -1;
 90              phi[i] = i-1;
 91              prime[np++] = i;
 92          }
 93          for (int j = 0; j < np && i * prime[j]<=n ; j++)
 94              if (i % prime[j])
 95              {
 96                  int k = i * prime[j];
 97                  isP[k] = -1;
 98                  mu[k] = -mu[i];
 99                  phi[k] = phi[i] * (prime[j]-1);
100              }
101              else
102              {
103                  int k = i * prime[j];
104                  isP[k] = -1;
105                  mu[k] = 0;
106                  phi[k] = phi[i] * prime[j];
107                  break;
108              }
109          summu[i] = summu[i-1] + mu[i];
110          sumphi[i] = sumphi[i-1] + phi[i];
111      }
112  }
113
114  int p[1000010], prime[100010], psize = 1000000;
115  LL a[1000100];
116  void sieve(){
117      int i,j,tot,t1;
118      for (i=1;i<=psize;i++) p[i]=i;
119      for (i=2,tot=0;i<=psize;i++){
120          if (p[i]==i) prime[++tot]=i;
121          for (j=1;j<=tot && (t1=prime[j]*i)<=psize;j++){
122              p[t1]=prime[j];
123              if (i%prime[j]==0) break;
124          }
125      }
126  }
127  inline LL mul(LL a, LL b, LL p)
128  {
129      if (p <= 1000000000)
130          return a * b % p;
131      else
132          if (p<=1000000000000LL)
133              return (((a * (b >> 20) % p) << 20) + (a * (b & ((1 << 20) - 1)))) % p;
134          else
135          {
136              LL d = (LL)floor(a * (long double)b / p + 0.5);
137              LL ret = (a * b - d * p) % p;
138              if (ret < 0) ret += p;
139              return ret;
140          }
141  }
142  LL fpow(LL a,LL n,LL p)
143  {
144      LL ans=1;
145      for (; n; n >>= 1)
146      {
147          if (n & 1) ans = mul(ans, a, p);
148          a = mul(a, a, p);
149      }
150      return ans;
151  }
152  bool witness(LL a, LL n)  //二次探查
153  {
154      int t = 0;
155      LL u = n - 1;
156      for (; ~u & 1; u >>= 1) t++;
157      LL x = fpow(a, u, n), _x = 0;
158      for (; t; t--)
159      {
160          _x = mul(x, x, n);
```

```
161      if (_x == 1 && x != 1 && x != n-1) return 1;
162      x = _x;
163    }
164    return _x != 1;
165  }
166  bool miller(LL n)
167  {
168    if (n < 2) return 0;
169    if (n < psize) return p[n] == n;
170    if (~n & 1) return 0;
171    for (int j = 0; j <= 7; j++)
172      if (witness(rand() % (n - 1) + 1, n))
173        return 0;
174    return 1;
175  }
176  LL gcd(LL a,LL b)
177  {
178    LL ret = 1;
179    while (a != 0)
180    {
181      if ((~a & 1) && (~b & 1))
182        ret <<= 1,a >>= 1,b >>= 1;
183      else
184        if (~a & 1)
185          a >>= 1;
186        else
187          if (~b & 1)
188            b >>= 1;
189          else
190          {
191            if (a < b)
192              swap(a, b);
193            a -= b;
194          }
195    }
196    return ret * b;
197  }
198  LL rho(LL n)
199  {
200    for (;;)
201    {
202      LL X = rand() % n, Y, Z, T = 1, *lY = a, *lX = lY;
203      int tmp = 20;
204      LL C = rand() % 10 + 3;
205      X = mul(X, X, n) + C;
206      *(lY++) = X; lX++;
207      Y = mul(X, X, n) + C;
208      *(lY++) = Y;
209      for(; X != Y;)
210      {
211        LL t = X - Y + n;
212        Z = mul(T, t, n);
213        if(Z == 0)
214          return gcd(T, n);
215        tmp--;
216        if (tmp == 0)
217        {
218          tmp = 20;
219          Z = gcd(Z, n);
220          if (Z != 1 && Z != n)
221            return Z;
222        }
223        T = Z;
224        Y = *(lY++) = mul(Y, Y, n) + C;
225        Y = *(lY++) = mul(Y, Y, n) + C;
226        X = *(lX++);
227      }
228    }
229  }
230  void find(LL n, int c)
231  {
232    for (int i = 0; i < ct; i++)
233      if (n % fac[i] == 0)
234        n /= fac[i], fac[ct++] = fac[i];
235    if(n == 1) return;
236    if (n <= psize)
237    {
238      for (; n != 1; n /= p[n])
239        fac[ct++] = p[n];
240      return;
241    }
242    if(miller(n))
243    {
244        fac[ct++] = n;
245        return ;
246    }
247    LL p = n;
248    LL k = c;
249    while(p >= n) p = rho(p);
250    find(p, k);
251    find(n / p, k);
252  }
253  void factorize(LL n, vector<pair<LL, LL> > &result)
254  {
255    result.clear();
256    if (n == 1)
```

```
257      return;
258    ct = 0;
259    find(n, 120);
260    sort(fac, fac + ct);
261    num[0] = 1;
262    int k = 1;
263    for(int i=1; i<ct; i++)
264    {
265      if(fac[i] == fac[i-1])
266        ++num[k-1];
267      else
268      {
269        num[k] = 1;
270        fac[k++] = fac[i];
271      }
272    }
273    cnt = k;
274    for (int i = 0; i < cnt; i++)
275      result.push_back(make_pair(fac[i], num[i]));
276 }
277
278 // discrete-logarithm, finding y for equation b = g^y % p
279 //p is prime
280 int M;  //M = (int)sqrt(phi(p));
281 void discrete_log_init(LL g, LL p)
282 {
283    hash_init();
284    int i;
285    LL tmp;
286    for(i = 0, tmp = 1; i < M; i++, tmp = tmp * g % p)
287      insert(tmp % p, i * 1LL);
288 }
289 LL discrete_log(LL g, LL p, LL b)
290 {
291    LL res, am = fpow(g, M, p), inv = fpow(b, p - 2, p), x = 1;
292    for(LL i = M; ; i += M)
293    {
294      if((res = find((x = x * am % p) * inv % p)) != -1)
295      {
296        return i - res;
297      }
298      if(i > p)break;
299    }
300    return -1;
301 }
302
303 //A^x=B mod C
304 //hash add(); find();
```

```
305 int Inval(int a,int b,int n){
306    int x,y,e;
307    ext_gcd(a, n, x, y);
308    e=(LL)x * b % n;
309    return e < 0 ? e + n : e;
310 }
311 int BabyStep(int A, int B, int C)
312 {
313    top = maxn; ++ idx;
314    LL buf = 1 % C, D = buf, K;
315    int i, d = 0, tmp;
316    for(i = 0; i <= 100; buf = buf * A % C, ++i)
317      if (buf == B)
318        return i;
319    while((tmp = gcd(A, C)) != 1)
320    {
321      if(B % tmp) return -1;
322      ++d;
323      C /= tmp;
324      B /= tmp;
325      D = D * A / tmp % C;
326    }
327    int M = (int)ceil(sqrt((double)C));
328    for(buf = 1 % C, i = 0; i <= M; buf = buf * A % C, ++i)
329      add(i, buf); //hash
330    for(i = 0, K = fpow((LL)A, M, C); i <= M; D = D * K % C, ++i)
331    {
332      tmp = Inval((int)D, B, C);
333      int w;
334      if(tmp >= 0 && (w = find(tmp)) != -1)  //hash
335        return i * M + w + d;
336    }
337    return -1;
338 }
339
340 // primtive root, finding the number with order p-1
341 int primtive_root(int p) {
342    vector<int> factor;
343    int tmp = p - 1;
344    for (int i = 2; i * i <= tmp; ++i) {
345      if (tmp % i == 0) {
346        factor.push_back(i);
347        while (tmp % i == 0) tmp /= i;
348      }
349    }
350    if (tmp != 1) factor.push_back(tmp);
351    for (int root = 1; ; ++root) {
352      bool flag = true;
```

```
353    for (int i = 0; i < factor.size(); ++i) {
354      if (fpow(root, (p - 1) / factor[i], p) == 1) {
355        flag = false;
356        break;
357      }
358    }
359    if (flag) return root;
360  }
361 }
```

3.7   Number Partition

```
1  // number of ways to divide n to integers(unordered), O(n^(3/2))
2  int partition(int n) {
3    int dp[n + 1];
4    dp[0] = 1;
5    for (int i = 1; i <= n; i++) {
6      dp[i] = 0;
7      for (int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; ++j, r *= -1) {
8        dp[i] += dp[i - (3 * j * j - j) / 2] * r;
9        if (i - (3 * j * j + j) / 2 >= 0) dp[i] += dp[i - (3 * j * j + j) / 2] * r;
10     }
11   }
12   return dp[n];
13 }
```

3.8   Lucas

```
1  LL C(LL n, LL m)
2  {
3    if (m > n) return 0;
4    LL ans = 1;
5    for (int i = 1; i <= m; i++)
6    {
7      LL a = (n + i - m) % p;
8      LL b = i % p;
9      ans = ans * (a * fpow(b, p-2, p) % p) % p;
10   }
11   return ans;
12 }
13 LL lucas(LL n, LL m)
14 {
15   if (m == 0) return 1;
16   return C(n % p, m % p) * lucas(n / p, m / p) % p;
17 }
```

3.9   Bonulli Number

```
1  //O(n^2)
2  LL fac[maxn], C[maxn][maxn], B[maxn], Inv[maxn], n, k;
3  void init()
4  {
5    for (int i = 0; i < maxn; i++)
6    {
7      C[i][0] = C[i][i] = 1;
8      if (i == 0) continue;
9      for (int j = 1; j < i; j++)
10       C[i][j] = (C[i-1][j] + C[i-1][j-1]) % Mod;
11   }
12   Inv[1] = 1;
13   for (int i = 2; i < maxn; i++)
14     Inv[i] = (Mod - Mod / i) * Inv[Mod % i] % Mod;
15   B[0] = 1;
16   for (int i = 1; i < maxn; i++)
17   {
18     LL ans = 0;
19     if (i == maxn - 1)
20       break;
21     for (int j = 0; j < i; j++)
22     {
23       ans += B[j]*C[i+1][j];
24       ans %= Mod;
25     }
26     ans *= -Inv[i+1];
27     ans = (ans % Mod + Mod) % Mod;
28     B[i] = ans;
29   }
30 }
31 LL Cal(int n, int k)
32 {
33   LL ans = Inv[k+1];
34   LL sum = 0;
35   for (int i = 1; i <= k+1; i++)
36   {
37     sum += C[k+1][i]*fac[i] % Mod * B[k+1-i] % Mod;
38     sum %= Mod;
39   }
40   ans *= sum;
41   ans %= Mod;
42   return ans;
43 }
```

$\sum_{i=1}^{n} i^k$

3.10   Fast Walsh Transform

```
1  //n is power of 2
2  void FWT_And(int x[], int l, int r, int v) //FWT v = 1, DFWT v = -1
3  {
4    if (l == r)
5      return;
6    int mid = (l + r) >> 1;
7    FWT_And(x, l, mid, v);
8    FWT_And(x, mid + 1, r, v);
9    for (int i = 0; i <= mid - l; i++)
10     x[i + l] += x[mid + i + 1] * v;
11 }
12 void FWT_Or(int x[], int l, int r, int v) // FWT v = 1 DFWT v = -1
13 {
14   if (l == r)
15     return;
16   int mid = (l + r) >> 1;
17   FWT_Or(x, l, mid, v);
18   FWT_Or(x, mid + 1, r, v);
19   for (int i = 0; i <= mid - l; i++)
20     x[mid + i + 1] += x[l + i] * v;
21 }
22 void FWT_Xor(int x[], int l, int r)
23 {
24   if (l == r)
25     return;
26   int mid = (l + r) >> 1;
27   FWT_Xor(x, l, mid);
28   FWT_Xor(x, mid + 1, r);
29   for (int i = 0; i <= mid - l; i++)
30     x[l + i] +=  x[mid + i + 1], x[mid + i + 1] = x[l + i] - 2 * x[mid + i + 1];
31 }
32 void DFWT_Xor(int x[], int l, int r)
33 {
34   if (l == r)
35     return;
36   int mid = (l + r) >> 1;
37   DFWT_Xor(x, l, mid);
38   DFWT_Xor(x, mid + 1, r);
39   for (int i = 0; i <= mid - l; i++)
40     x[l + i] = (x[l + i] + x[mid + i + 1]) / 2, x[mid + i + 1] = x[l + i] - x[mid +
         i + 1];
41 }
```

3.11   Fast Fourier Transform

```
1  void fft(int sign, int n, double *real, double *imag) {
```

```
2    double theta = sign * 2 * pi / n;
3    for (int m = n; m >= 2; m >>= 1, theta *= 2) {
4      double wr = 1, wi = 0, c = cos(theta), s = sin(theta);
5      for (int i = 0, mh = m >> 1; i < mh; ++i) {
6        for (int j = i; j < n; j += m) {
7          int k = j + mh;
8          double xr = real[j] - real[k], xi = imag[j] - imag[k];
9          real[j] += real[k], imag[j] += imag[k];
10         real[k] = wr * xr - wi * xi, imag[k] = wr * xi + wi * xr;
11       }
12       double _wr = wr * c - wi * s, _wi = wr * s + wi * c;
13       wr = _wr, wi = _wi;
14     }
15   }
16   for (int i = 1, j = 0; i < n; ++i) {
17     for (int k = n >> 1; k > (j ^= k); k >>= 1);
18     if (j < i) swap(real[i], real[j]), swap(imag[i], imag[j]);
19   }
20 }
21 // Compute Poly(a)*Poly(b), write to r; Indexed from 0
22 int mult(int *a, int n, int *b, int m, int *r) {
23   static double ra[maxn], rb[maxn], ia[maxn], ib[maxn];
24   int fn = 1;
25   while (fn < n + m) fn <<= 1; // n + m: interested length
26   for (int i = 0; i < n; ++i) ra[i] = a[i], ia[i] = 0;
27   for (int i = n; i < fn; ++i) ra[i] = ia[i] = 0;
28   for (int i = 0; i < m; ++i) rb[i] = b[i], ib[i] = 0;
29   for (int i = m; i < fn; ++i) rb[i] = ib[i] = 0;
30   fft(1, fn, ra, ia);
31   fft(1, fn, rb, ib);
32   for (int i = 0; i < fn; ++i) {
33     double real = ra[i] * rb[i] - ia[i] * ib[i];
34     double imag = ra[i] * ib[i] + rb[i] * ia[i];
35     ra[i] = real, ia[i] = imag;
36   }
37   fft(-1, fn, ra, ia);
38   for (int i = 0; i < fn; ++i) r[i] = (int)floor(ra[i] / fn + 0.5);
39   return fn;
40 }
```

3.12   Number Theoretic Transform

```
1  int n, K, inv_K;
2  int P = 1998585857, g = 3;
3  int w[2][100000];
4  int fpm(int a, int b)
5  {
6    int ret = 1;
```

```
 7    for (; b; b >>= 1)
 8    {
 9      if (b & 1)
10        ret = (LL)ret * a % P;
11      a = (LL)a * a % P;
12    }
13    return ret;
14 }
15 void FFT_Init() {
16    for ( K = 1; K < n << 1; K <<= 1 ); inv_K = fpm(K, P - 2);
17    w[ 0 ][ 0 ] = w[ 0 ][ K ] = w[ 1 ][ 0 ] = w[ 1 ][ K ] = 1;
18    int G = fpm( g, ( P - 1 ) / K );
19    FOR( i, 1, K - 1 ) {
20      w[ 0 ][ i ] = (ll)w[ 0 ][ i - 1 ] * G % P;
21    }
22    FOR( i, 0, K ) {
23      w[ 1 ][ i ] = w[ 0 ][ K - i ];
24    }
25 }
26
27 void FFT( int X[], int k, int v ) {
28    int i, j, l;
29    for ( i = j = 0; i < k; i++ ) {
30      if ( i > j ) swap( X[ i ], X[ j ] );
31      for ( l = k >> 1; ( j ^= l ) < l; l >>= 1 );
32    }
33    for ( i = 2; i <= k; i <<= 1 )
34      for ( j = 0; j < k; j += i )
35        for ( l = 0; l < i >> 1; l++ ) {
36          int t = (ll)X[ j + l + ( i >> 1 ) ] * w[ v ][ ( K / i ) * l ] % P;
37          X[ j + l + ( i >> 1 ) ] = ( (ll)X[ j + l ] - t + P ) % P;
38          X[ j + l ] = ( (ll)X[ j + l ] + t ) % P;
39        }
40    if (v)
41      for (i = 0; i < k; i++)
42        X[i] = (ll)X[i] * inv_K % P;
43 }
44 int tmp[100000];
45 void GetInv( int A[], int A0[], int t ) {
46    if ( t == 1 ) { A0[ 0 ] = fpm( A[ 0 ], P - 2); return; }
47    GetInv( A, A0, ( t + 1 ) >> 1 );
48    K = 1; for ( ; K <= ( t << 1 ) + 3; K <<= 1 ); inv_K = fpm( K, P - 2);
49    w[ 0 ][ 0 ] = w[ 0 ][ K ] = w[ 1 ][ 0 ] = w[ 1 ][ K ] = 1;
50    int G = fpm( g, ( P - 1 ) / K );
51    FOR( i, 1, K - 1 ) {
52      w[ 0 ][ i ] = (ll)w[ 0 ][ i - 1 ] * G % P;
53    }
54    FOR( i, 0, K ) {
55      w[ 1 ][ i ] = w[ 0 ][ K - i ];
56    }
57    FOR ( i, 0, t - 1 ) { tmp[ i ] = A[ i ]; } FOR ( i, t, K - 1 ) { tmp[ i ] = 0; }
58    FFT( tmp, K, 0 ); FFT( A0, K, 0 );
59    FOR ( i, 0, K - 1 ) { tmp[ i ] = 2 - (ll)tmp[ i ] * A0[ i ] % P + P; tmp[i] %= P;
       }
60    FOR ( i, 0, K - 1 ) { A0[ i ] = (ll)A0[ i ] * tmp[ i ] % P; }
61    FFT( A0, K, 1 );
62    FOR ( i, t, K - 1 ) { A0[ i ] = 0; }
63 }
64 int fac[100010], inv_fac[100010], B[100010];
65 void GetBernoulli(int n)
66 {
67    fac[0] = inv_fac[0] = 1;
68    For(i, 1, n - 1)
69      fac[i] = (ll)fac[i - 1] * i % P;
70    For(i, 1, n)
71      inv_fac[i] = (ll)inv_fac[i - 1] * fpm(i + 1, P - 2) % P;
72    GetInv(inv_fac, B, n);
73    rep(i, n)
74      B[i] = (ll)B[i] * fac[i] % P;
75    rep(i, n)
76      cout << B[i] << endl;
77 }
78
79
80 //CRT version
81 const int P = 1000003; // Approximate 10^6
82 const int P1 = 998244353, P2 = 995622913;
83 const LL M1 = 397550359381069386LL, M2 = 5963245912385909004LL;
84 const LL MM = 9938749506119660289LL;
85
86 int CRT(int x1, int x2) {
87    return (mult(M1, x1, MM) + mult(M2, x2, MM)) % MM % P; // 64bit multiplication
88 }
89
90 void NTT(int *A, int PM, int PW, int n) {
91    for (int m = n, h; h = m / 2, m >= 2; PW = (LL)PW * PW % PM, m = h) {
92      for (int i = 0, w = 1; i < h; ++i, w = (LL)w * PW % PM) {
93        for (int j = i; j < n; j += m) {
94          int k = j + h, x = (A[j] - A[k] + PM) % PM;
95          A[j] += A[k]; A[j] %= PM;
96          A[k] = (LL)w * x % PM;
97        }
98      }
99    }
100   for (int i = 0, j = 1; j < n - 1; ++j) {
101     for (int k = n / 2; k > (i ^= k); k /= 2);
```

```
102      if (j < i) swap(A[i], A[j]);
103    }
104 }
105
106 int E1, E2, F1, F2, I1, I2;
107 int init(int n) { // assert(k <= 19);
108    int k = 1, N = 2, p;
109    while (N < n) N <<= 1, ++k;
110    p = 7 * 17; for (int i = 1; i <= 23 - k; ++i) p *= 2;
111    E1 = fpow(3, p, P1); F1 = fpow(E1, P1 - 2, P1); I1 = fpow(N, P1 - 2, P1);
112    p = 9 * 211; for (int i = 1; i <= 19 - k; ++i) p *= 2;
113    E2 = fpow(5, p, P2); F2 = fpow(E2, P2 - 2, P2); I2 = fpow(N, P2 - 2, P2);
114    return N;
115 }
116
117 void mul(int *A, int *B, int *C, int n) {
118    static int A1[maxn], B1[maxn], C1[maxn];
119    int N = init(n);
120    memset(A1, 0, sizeof(*A1) * N); memset(B1, 0, sizeof(*B1) * N); memset(C1, 0,
           sizeof(*C1) * N);
121    memset(C, 0, sizeof(*C) * N);
122    memcpy(A1, A, sizeof(*A) * n); memcpy(B1, B, sizeof(*B) * n);
123    NTT(A1, P1, E1, N); NTT(B1, P1, E1, N);
124    for (int i = 0; i < N; ++i) C1[i] = (LL)A1[i] * B1[i] % P1;
125    NTT(C1, P1, F1, N);
126    for (int i = 0; i < N; ++i) C1[i] = (LL)C1[i] * I1 % P1;
127    NTT(A, P2, E2, N); NTT(B, P2, E2, N);
128    for (int i = 0; i < N; ++i) C[i] = (LL)A[i] * B[i] % P2;
129    NTT(C, P2, F2, N);
130    for (int i = 0; i < N; ++i) C[i] = (LL)C[i] * I2 % P2;
131    for (int i = 0; i < N; ++i) C[i] = CRT(C1[i], C[i]);
132    for (int i = n; i < N; ++i) C[i] = 0;
133 }
```

3.13 Modular Factorial

$n! \mod mod$ where $mod = p^k$. $O(p \log n)$

```
1 LL get(int n, int mod, int p) {
2    LL ans = 1;
3    for (int i = 1; i <= n; ++i) if (i % p != 0) {
4       ans = ans * i % mod;
5    }
6    return ans;
7 }
8 pii solve(LL n, int mod, int p) {
9    LL init = get(mod, mod, p);
10    pii ans = pii(1, 0);
11    for (LL now = p; now <= n; now *= p) {
```

```
12      ans.second += n / now;
13      if (now > n / p) break;
14    }
15    while (n > 0) {
16      ans.first = (LL) ans.first * fpow(init, n / mod, mod) % mod;
17      ans.first = ans.first * get(n % mod, mod, p) % mod;
18      n /= p;
19    }
20    return ans;
21 }
```

3.14 Linar Programming

```
1 double a[maxn][maxm], b[maxn], c[maxm], d[maxn][maxm];
2 int ix[maxn + maxm]; // !!! array all indexed from 0
3 // max{cx|Ax<=b,x>=0}, n: constraints, m: vars
4 double simplex(double a[maxn][maxm], double b[maxn], double c[maxm], int n, int m) {
5    ++m;
6    int r = n, s = m - 1;
7    memset(d, 0, sizeof(d));
8    for (int i = 0; i < n + m; ++i) ix[i] = i;
9    for (int i = 0; i < n; ++i) {
10      for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
11      d[i][m - 1] = 1;
12      d[i][m] = b[i];
13      if (d[r][m] > d[i][m]) r = i;
14    }
15    for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
16    d[n + 1][m - 1] = -1;
17    for (double dd;; ) {
18      if (r < n) {
19        int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
20        d[r][s] = 1.0 / d[r][s];
21        for (int j = 0; j <= m; ++j) if (j != s) d[r][j] *= -d[r][s];
22        for (int i = 0; i <= n + 1; ++i) if (i != r) {
23          for (int j = 0; j <= m; ++j) if (j != s) d[i][j] += d[r][j] * d[i][s];
24          d[i][s] *= d[r][s];
25        }
26      }
27      r = -1; s = -1;
28      for (int j = 0; j < m; ++j) if (s < 0 || ix[s] > ix[j]) {
29        if (d[n + 1][j] > eps || (d[n + 1][j] > -eps && d[n][j] > eps)) s = j;
30      }
31      if (s < 0) break;
32      for (int i = 0; i < n; ++i) if (d[i][s] < -eps) {
33        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;
34      }
```

```
35     if (r < 0) return -1; // not bounded
36   }
37   if (d[n + 1][m] < -eps) return -1; // not executable
38   double ans = 0;
39   for (int i = m; i < n + m; ++i) { // the missing enumerated x[i] = 0.  x[i] = c[ix
       [i]]
40     if (ix[i] < m - 1) ans += d[i - m][m] * c[ix[i]];
41   }
42   return ans;
43 }
```

3.15   Simpson Integration

```
1  const double eps = 1e-15;
2  double f(double x) { return 0.0; }
3  double sim(double l, double r, double lv, double rv, double mv) {
4    return (r - l) * (lv + rv + 4 * mv) / 6;
5  }
6  double rsim(double l, double r, double lv, double rv, double mv, double m1v, double
     m2v) {
7    double mid = (l + r) / 2;
8    if (fabs(sim(l, r, lv, rv, mv) - sim(l, mid, lv, mv, m1v) - sim(mid, r, mv, rv,
       m2v)) / 15 < eps) {
9      return sim(l, r, lv, rv, mv);
10   } else {
11     double mid = (l + r) / 2, m1 = (l + (l + r) / 2) / 2, m2 = ((l + r) / 2 + r) /
         2;
12     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));
13   }
14 }
15 double simpson(double l, double r) {
16   double mid = (l + r) / 2;
17   return rsim(l, r, f(l), f(r), f(mid), f((l + mid) / 2), f((mid + r) / 2));
18 }
```

4   Computational Geometry

4.1   Common 2D

```
1  // implementation of (dblcmp,dist,cross,dot) is trivial
2  const double eps = 1e-8;
3  int dblcmp(double x)
4  {
5    if (fabs(x) < eps)
6      return 0;
7    return x > 0 ? 1 : -1;
8  }
9  struct point_t
10 {
11   double x, y;
12   point_t(): x(0), y(0) {}
13   point_t(double x, double y): x(x), y(y) {}
14   bool operator <(const point_t &b) const
15   {
16     return dblcmp(x - b.x) < 0 || (dblcmp(x - b.x) == 0 && dblcmp(y - b.y) < 0);
17   }
18   bool operator ==(const point_t &b) const
19   {
20     return dblcmp(x - b.x) == 0 && dblcmp(y - b.y) == 0;
21   }
22   point_t operator +(const point_t &b)
23   {
24     return point_t(x + b.x, y + b.y);
25   }
26   point_t operator -(const point_t &b)
27   {
28     return point_t(x - b.x, y - b.y);
29   }
30   point_t operator /(double k)
31   {
32     return point_t(x / k, y / k);
33   }
34   double operator *(const point_t b)
35   {
36     return x * b.x + y * b.y;
37   }
38 };
39 double dist(point_t p1, point_t p2)
40 {
41   return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
42 }
43 double cross(point_t p1, point_t p2)
44 {
45   return p1.x * p2.y - p1.y * p2.x;
46 }
47 // count-clock wise is positive direction
48 double angle(point_t p1, point_t p2) {
49   double x1 = p1.x, y1 = p1.y, x2 = p2.x, y2 = p2.y;
50   double a1 = atan2(y1, x1), a2 = atan2(y2, x2);
51   double a = a2 - a1;
52   while (a < -pi) a += 2 * pi;
53   while (a >= pi) a -= 2 * pi;
54   return a;
```

```cpp
 55 }
 56
 57 bool onSeg(point_t p, point_t a, point_t b) {
 58   return dblcmp(cross(a - p, b - p)) == 0 && dblcmp(dot(a - p, b - p)) <= 0;
 59 }
 60
 61 // 1 normal intersected, -1 denormal intersected, 0 not intersected
 62 int testSS(point_t a, point_t b, point_t c, point_t d) {
 63   if (dblcmp(max(a.x, b.x) - min(c.x, d.x)) < 0) return 0;
 64   if (dblcmp(max(c.x, d.x) - min(a.x, b.x)) < 0) return 0;
 65   if (dblcmp(max(a.y, b.y) - min(c.y, d.y)) < 0) return 0;
 66   if (dblcmp(max(c.y, d.y) - min(a.y, b.y)) < 0) return 0;
 67   int d1 = dblcmp(cross(c - a, b - a));
 68   int d2 = dblcmp(cross(d - a, b - a));
 69   int d3 = dblcmp(cross(a - c, d - c));
 70   int d4 = dblcmp(cross(b - c, d - c));
 71   if ((d1 * d2 < 0) && (d3 * d4 < 0)) return 1;
 72   if ((d1 * d2 <= 0 && d3 * d4 == 0) || (d1 * d2 == 0 && d3 * d4 <= 0)) return -1;
 73   return 0;
 74 }
 75
 76 vector<point_t> isLL(point_t a, point_t b, point_t c, point_t d) {
 77   point_t p1 = b - a, p2 = d - c;
 78   vector<point_t> ret;
 79   double a1 = p1.y, b1 = -p1.x, c1;
 80   double a2 = p2.y, b2 = -p2.x, c2;
 81   if (dblcmp(a1 * b2 - a2 * b1) == 0) return ret; // colined <=> a1*c2-a2*c1=0 && b1
        *c2-b2*c1=0
 82   else {
 83     c1 = a1 * a.x + b1 * a.y;
 84     c2 = a2 * c.x + b2 * c.y;
 85     ret.push_back(point_t((c1 * b2 - c2 * b1) / (a1 * b2 - a2 * b1), (c1 * a2 - c2 *
        a1) / (b1 * a2 - b2 * a1)));
 86     return ret;
 87   }
 88 }
 89
 90 point_t angle_bisector(point_t p0, point_t p1, point_t p2) {
 91   point_t v1 = p1 - p0, v2 = p2 - p0;
 92   v1 = v1 / dist(v1) * dist(v2);
 93   return v1 + v2 + p0;
 94 }
 95
 96 point_t perpendicular_bisector(point_t p1, point_t p2) {
 97   point_t v = p2 - p1;
 98   swap(v.x, v.y);
 99   v.x = -v.x;
100   return v + (p1 + p2) / 2;
101 }
102
103 point_t circumcenter(point_t p0, point_t p1, point_t p2) {
104   point_t v1 = perpendicular_bisector(p0, p1);
105   point_t v2 = perpendicular_bisector(p1, p2);
106   return isLL((p0 + p1) / 2, v1, (p1 + p2) / 2, v2);
107 }
108
109 point_t incenter(point_t p0, point_t p1, point_t p2) {
110   point_t v1 = angle_bisector(p0, p1, p2);
111   point_t v2 = angle_bisector(p1, p2, p0);
112   return isLL(p0, v1, p1, v2);
113 }
114
115 point_t orthocenter(point_t p0, point_t p1, point_t p2) {
116   return p0 + p1 + p2 - circumcenter(p0, p1, p2) * 2;
117 }
118
119 // count-clock wise is positive direction
120 point_t rotate(point_t p, double a) {
121   double s = sin(a), c = cos(a);
122   return point_t(p.x * c - p.y * s, p.y * c + p.x * s);
123 }
124
125 bool insidePoly(point_t *p, int n, point_t t) {
126   p[0] = p[n];
127   for (int i = 0; i < n; ++i) if (onSeg(t, p[i], p[i + 1])) return true;
128   point_t r = point_t(2353456.663, 5326546.243); // random point
129   int cnt = 0;
130   for (int i = 0; i < n; ++i) {
131     if (testSS(t, r, p[i], p[i + 1]) != 0) ++cnt;
132   }
133   return cnt & 1;
134 }
135
136 bool insideConvex(point_t *convex, int n, point_t t) { // O(logN), convex polygen,
        cross(p[2] - p[1], p[3] - p[1]) > 0
137   if (n == 2) return onSeg(t, convex[1], convex[2]);
138   int l = 2, r = n;
139   while (l < r) {
140     int mid = (l + r) / 2 + 1;
141     int side = dblcmp(cross(convex[mid] - convex[1], t - convex[1]));
142     if (side == 1) l = mid;
143     else r = mid - 1;
144   }
145   int s = dblcmp(cross(convex[l] - convex[1], t - convex[1]));
146   if (s == -1 || l == n) return false;
147   point_t v = convex[l + 1] - convex[l];
```

```
148    if (dblcmp(cross(v, t - convex[l])) >= 0) return true;
149    return false;
150  }
```

4.2 Graham Convex Hull

```
1  bool cmp(const point_t p1, const point_t p2) {
2    return dblcmp(p1.y - p2.y) == 0 ? p1.x < p2.x : p1.y < p2.y;
3  }
4
5  int graham(point_t *p) { // Points co-lined are ignored.
6    int top = 2; static point_t sk[maxn];
7    sort(p + 1, p + 1 + n, cmp);
8    sk[1] = p[1], sk[2] = p[2];
9    for (int i = 3; i <= n; ++i) {
10     while (top >= 2 && dblcmp(cross(p[i] - sk[top - 1], sk[top] - sk[top - 1])) >=
          0) --top;
11     sk[++top] = p[i];
12   }
13   int ttop = top;
14   for (int i = n - 1; i >= 1; --i) {
15     while (top > ttop && dblcmp(cross(p[i] - sk[top - 1], sk[top] - sk[top - 1])) >=
          0) --top;
16     sk[++top] = p[i];
17   }
18   for (int i = 1; i < top; ++i) p[i] = sk[i];
19   return --top;
20 }
```

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.

```
1  int minkowski(point_t *h, point_t *h1, point_t *h2, int n, int m) {
2    point_t c = point_t(0, 0);
3    for (int i = 1; i <= m; ++i) c = c + h2[i];
4    c = c / m;
5    for (int i = 1; i <= m; ++i) h2[i] = h2[i] - c;
6    int cur = -1;
7    for (int i = 1; i <= m; ++i) {
8      if (dblcmp(cross(h2[i], h1[1] - h1[n])) >= 0) {
9        if (cur == -1 || cross(h2[i], h1[1] - h1[n]) > cross(h2[cur], h1[1] - h1[n]))
10         cur = i;
11     }
12   }
13   int cnt = 0;
14   h1[n + 1] = h1[1];
15   for (int i = 1; i <= n; ++i) {
16     while (true) {
17       h[++cnt] = h1[i] + h2[cur];
18       int next = (cur == m ? 1 : cur + 1);
19       if (dblcmp(cross(h2[cur], h1[i + 1] - h1[i])) < 0) cur = next;
20       else {
21         if (cross(h2[next], h1[i + 1] - h1[i]) > cross(h2[cur], h1[i + 1] - h1[i]))
22           cur = next;
23         else break;
24       }
25     }
26   }
27   for (int i = 1; i <= cnt; ++i) h[i] = h[i] + c;
28   for (int i = 1; i <= m; ++i) h2[i] = h2[i] + c;
29   return graham(h, cnt);
30 }
```

4.4 Rotating Calipers

```
1  // Calculate the maximum distance of a point set.
2  double rotate_caliper() {
3    p[0] = p[n];
4    int to = 0;
5    double ans = 0;
6    for (int i = 0; i < n; ++i) {
7      while ((to + 1) % n != i) {
8        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;
9        else break;
10     }
11     ans = max(ans, dist(p[i], p[to]));
12     ans = max(ans, dist(p[i + 1], p[to]));
13   }
```

```
14│   return ans;
15│}
```

4.5   Closest Pair Points

```
 1│double dac(point_t *p, int l, int r) {
 2│   double d = 10e100;
 3│   if (r - l <= 3) {
 4│     for (int i = l; i <= r; ++i) {
 5│       for (int j = i + 1; j <= r; ++j) {
 6│         d = min(d, dist2(p[i], p[j]));
 7│       }
 8│     }
 9│     sort(p + l, p + r + 1, cmpY);
10│   } else {
11│     int mid = (l + r) / 2;
12│     d = min(dac(p, l, mid), dac(p, mid + 1, r));
13│     inplace_merge(p + l, p + mid + 1, p + r + 1, cmpY);
14│     static point_t tmp[maxn]; int cnt = 0;
15│     for (int i = l; i <= r; ++i) {
16│       if ((p[i].x - p[mid].x) * (p[i].x - p[mid].x) <= d) tmp[++cnt] = p[i];
17│     }
18│     for (int i = 1; i <= cnt; ++i) {
19│       for (int j = 1; j <= 8 && j + i <= cnt; ++j) {
20│         d = min(d, dist2(tmp[i], tmp[j + i]));
21│       }
22│     }
23│   }
24│   return d;
25│}
26│
27│double cal(point_t *p, int n) {
28│   sort(p + 1, p + 1 + n, cmpX);
29│   return sqrt(dac(p, 1, n));
30│}
```

4.6   Halfplane Intersection

```
 1│// O(N^2) sol, polygon counterclockwise order
 2│// i.e., left side of vector v1->v2 is the valid half plane
 3│const double maxd = 1e5;
 4│int n, cnt;
 5│point_t p[maxn];
 6│
 7│void init() { // order reversed if right side
 8│   cnt = 4;
 9│   p[1] = point_t(-maxd, -maxd);
```

```
10│   p[2] = point_t(maxd, -maxd);
11│   p[3] = point_t(maxd, maxd);
12│   p[4] = point_t(-maxd, maxd);
13│}
14│
15│void cut(point_t p1, point_t p2) {
16│   int tcnt = 0;
17│   static point_t tp[maxn];
18│   p[cnt + 1] = p[1];
19│   for (int i = 1; i <= cnt; ++i) {
20│     double v1 = cross(p2 - p1, p[i] - p1);
21│     double v2 = cross(p2 - p1, p[i + 1] - p1);
22│     if (dblcmp(v1) >= 0) tp[++tcnt] = p[i]; // <= if right side
23│     if (dblcmp(v1) * dblcmp(v2) < 0) tp[++tcnt] = isLL(p1, p2, p[i], p[i + 1]);
24│   }
25│   cnt = tcnt;
26│   for (int i = 1; i <= cnt; ++i) p[i] = tp[i];
27│}
```

```
 1│// O(NlogN) sol, Left is valid half plane. Note that the edge of hull may degenerate
   │    to a point.
 2│struct hp_t {
 3│   point_t p1, p2;
 4│   double a;
 5│   hp_t() { }
 6│   hp_t(point_t tp1, point_t tp2) : p1(tp1), p2(tp2) {
 7│     tp2 = tp2 - tp1;
 8│     a = atan2(tp2.y, tp2.x);
 9│   }
10│   bool operator==(const hp_t &r) const {
11│     return dblcmp(a - r.a) == 0;
12│   }
13│   bool operator<(const hp_t &r) const {
14│     if (dblcmp(a - r.a) == 0) return dblcmp(cross(r.p2 - r.p1, p2 - r.p1)) >= 0;
15│     else return a < r.a;
16│   }
17│} hp[maxn];
18│
19│void addhp(point_t p1, point_t p2) {
20│   hp[++cnt] = hp_t(p1, p2);
21│}
22│
23│void init() {
24│   cnt = 0;
25│   addhp(point_t(-maxd, -maxd), point_t(maxd, -maxd));
26│   addhp(point_t(maxd, -maxd), point_t(maxd, maxd));
27│   addhp(point_t(maxd, maxd), point_t(-maxd, maxd));
28│   addhp(point_t(-maxd, maxd), point_t(-maxd, -maxd));
29│}
```

```
30
31 bool checkhp(hp_t h1, hp_t h2, hp_t h3) {
32   point_t p = isLL(h1.p1, h1.p2, h2.p1, h2.p2);
33   return dblcmp(cross(p - h3.p1, h3.p2 - h3.p1)) > 0;
34 }
35
36 vector<point_t> hp_inter() {
37   sort(hp + 1, hp + 1 + cnt);
38   cnt = unique(hp + 1, hp + 1 + cnt) - hp - 1;
39   deque<hp_t> DQ;
40   DQ.push_back(hp[1]);
41   DQ.push_back(hp[2]);
42   for (int i = 3; i <= cnt; ++i) {
43     while (DQ.size() > 1 && checkhp(*----DQ.end(), *--DQ.end(), hp[i])) DQ.pop_back
        ();
44     while (DQ.size() > 1 && checkhp(*++DQ.begin(), *DQ.begin(), hp[i])) DQ.pop_front
        ();
45     DQ.push_back(hp[i]);
46   }
47   while (DQ.size() > 1 && checkhp(*----DQ.end(), *--DQ.end(), DQ.front())) DQ.
      pop_back();
48   while (DQ.size() > 1 && checkhp(*++DQ.begin(), *DQ.begin(), DQ.back())) DQ.
      pop_front();
49   DQ.push_front(DQ.back());
50   vector<point_t> res;
51   while (DQ.size() > 1) {
52     hp_t tmp = DQ.front();
53     DQ.pop_front();
54     res.push_back(isLL(tmp.p1, tmp.p2, DQ.front().p1, DQ.front().p2));
55   }
56   return res;
57 }
```

4.7 Tri-Cir Intersection & Tangent

```
1 vector<point_t> tanCP(point_t c, double r, point_t p) {
2   double x = dot(p - c, p - c);
3   double d = x - r * r;
4   vector<point_t> res;
5   if (d < -eps) return res;
6   if (d < 0) d = 0;
7   point_t q1 = (p - c) * (r * r / x);
8   point_t q2 = ((p - c) * (-r * sqrt(d) / x)).rot90(); // rot90: (-y, x)
9   res.push_back(c + q1 - q2);
10  res.push_back(c + q1 + q2);
11  return res;
12 }
13
```

```
14 vector<seg_t> tanCC(point_t c1, double r1, point_t c2, double r2) {
15   vector<seg_t> res;
16   if (abs(r1 - r2) < eps) {
17     point_t dir = c2 - c1;
18     dir = (dir * (r1 / dir.l())).rot90();
19     res.push_back(seg_t(c1 + dir, c2 + dir));
20     res.push_back(seg_t(c1 - dir, c2 - dir));
21   } else {
22     point_t p = ((c1 * -r2) + (c2 * r1)) / (r1 - r2);
23     vector<point_t> ps = tanCP(c1, r1, p), qs = tanCP(c2, r2, p);
24     for (int i = 0; i < ps.size() && i < qs.size(); ++i) {
25       res.push_back(seg_t(ps[i], qs[i]));
26     }
27   }
28   point_t p = ((c1 * r2) + (c2 * r1)) / (r1 + r2);
29   vector<point_t> ps = tanCP(c1, r1, p), qs = tanCP(c2, r2, p);
30   // point_t tmp = (c2 - c1).rot90().rot90().rot90();
31   for (int i = 0; i < ps.size() && i < qs.size(); ++i) {
32     /* if (dblcmp(dist(ps[i], qs[i])) == 0) {
33       qs[i] = qs[i] + tmp;
34       tmp = tmp.rot90().rot90();
35     }*/
36     res.push_back(seg_t(ps[i], qs[i]));
37   }
38   return res;
39 }
```

```
1 // Assume d <= r1 + r2 && d >= |r1 - r2|
2 pair<point_t, point_t> isCC(point_t c1, point_t c2, double r1, double r2) {
3   if (r1 < r2) swap(c1, c2), swap(r1, r2);
4   double d = dist(c1, c2);
5   double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
6   double mid = atan2(y2 - y1, x2 - x1);
7   double a = r1, c = r2;
8   double t = acos(max(0.0, a * a + d * d - c * c) / (2 * a * d));
9   point_t p1 = point_t(cos(mid - t) * r1, sin(mid - t) * r1) + c1;
10  point_t p2 = point_t(cos(mid + t) * r1, sin(mid + t) * r1) + c1;
11  return make_pair(p1, p2);
12 }
13
14 int testCC(point_t c1, point_t c2, double r1, double r2) {
15   double d = dist(c1, c2);
16   if (dblcmp(r1 + r2 - d) <= 0) return 1; // not intersected or tged
17   if (dblcmp(r1 + d - r2) <= 0) return 2; // C1 inside C2
18   if (dblcmp(r2 + d - r1) <= 0) return 3; // C2 inside C1
19   return 0; // intersected
20 }
21
22 point_t isCL(point_t a, point_t b, point_t o, double r) {
```

```
23  double x0 = o.x, y0 = o.y;
24  double x1 = a.x, y1 = a.y;
25  double x2 = b.x, y2 = b.y;
26  double dx = x2 - x1, dy = y2 - y1;
27  double A = dx * dx + dy * dy;
28  double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
29  double C = (x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 - y0) - r * r;
30  double delta = B * B - 4 * A * C;
31  if (delta >= 0) {
32    delta = sqrt(delta);
33    double t1 = (-B - delta) / 2 / A;
34    double t2 = (-B + delta) / 2 / A;
35    if (dblcmp(t1) >= 0) return point_t(x1 + t1 * dx, y1 + t1 * dy); // Ray
36    if (dblcmp(t2) >= 0) return point_t(x1 + t2 * dx, y1 + t2 * dy);
37  }
38  return point_t();
39 }
```

```
1  double areaTC(point_t ct, double r, point_t p1, point_t p2) { // intersected area
2    double a, b, c, x, y, s = cross(p1 - ct, p2 - ct) / 2;
3    a = dist(ct, p2), b = dist(ct, p1), c = dist(p1, p2);
4    if (a <= r && b <= r) {
5      return s;
6    } else if (a < r && b >= r) {
7      x = (dot(p1 - p2, ct - p2) + sqrt(c * c * r * r - sqr(cross(p1 - p2, ct - p2))))
            / c;
8      return asin(s * (c - x) * 2 / c / b / r) * r * r / 2 + s * x / c;
9    } else if (a >= r && b < r) {
10     y = (dot(p2 - p1, ct - p1) + sqrt(c * c * r * r - sqr(cross(p2 - p1, ct - p1))))
            / c;
11     return asin(s * (c - y) * 2 / c / a / r) * r * r / 2 + s * y / c;
12   } else {
13     if (fabs(2 * s) >= r * c || dot(p2 - p1, ct - p1) <= 0 || dot(p1 - p2, ct - p2)
          <= 0) {
14       if (dot(p1 - ct, p2 - ct) < 0) {
15         if (cross(p1 - ct, p2 - ct) < 0) {
16           return (-pi - asin(s * 2 / a / b)) * r * r / 2;
17         } else {
18           return (pi - asin(s * 2 / a / b)) * r * r / 2;
19         }
20       } else {
21         return asin(s * 2 / a / b) * r * r / 2;
22       }
23     } else {
24       x = (dot(p1 - p2, ct - p2) + sqrt(c * c * r * r - sqr(cross(p1 - p2, ct - p2))
            )) / c;
25       y = (dot(p2 - p1, ct - p1) + sqrt(c * c * r * r - sqr(cross(p2 - p1, ct - p1))
            )) / c;
```

```
26    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);
27  }
28 }
29 }
30
31 double areaTC(point_t ct, double r, point_t p1, point_t p2, point_t p3) {
32   return areaTC(ct, r, p1, p2) + areaTC(ct, r, p2, p3) + areaTC(ct, r, p3, p1);
33 }
```

4.8   Circle Area Union

```
1  /* O(n^2logn), please remove coincided circles first. */
2  point_t center[maxn];
3  double radius[maxn], cntarea[maxn];
4
5  pair<double, double> isCC(point_t c1, point_t c2, double r1, double r2) {
6    double d = dist(c1, c2);
7    double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
8    double mid = atan2(y2 - y1, x2 - x1);
9    double a = r1, c = r2;
10   double t = acos((a * a + d * d - c * c) / (2 * a * d));
11   return make_pair(mid - t, mid + t);
12 }
13
14 struct event_t {
15   double theta;
16   int delta;
17   event_t(double t, int d) : theta(t), delta(d) { }
18   bool operator<(const event_t &r) const {
19     if (fabs(theta - r.theta) < eps) return delta > r.delta;
20     return theta < r.theta;
21   }
22 };
23 vector<event_t> e;
24
25 void add(double begin, double end) {
26   if (begin <= -pi) begin += 2 * pi, end += 2 * pi;
27   if (end > pi) {
28     e.push_back(event_t(begin, 1));
29     e.push_back(event_t(pi, -1));
30     e.push_back(event_t(-pi, 1));
31     e.push_back(event_t(end - 2 * pi, -1));
32   } else {
33     e.push_back(event_t(begin, 1));
34     e.push_back(event_t(end, -1));
35   }
36 }
```

```
37
38 double calc(point_t c, double r, double a1, double a2) {
39   double da = a2 - a1;
40   double aa = r * r * (da - sin(da)) / 2;
41   point_t p1 = point_t(cos(a1), sin(a1)) * r + c;
42   point_t p2 = point_t(cos(a2), sin(a2)) * r + c;
43   return cross(p1, p2) / 2 + aa;
44 }
45
46 void circle_union() {
47   for (int c = 1; c <= n; ++c) {
48     int cvrcnt = 0;
49     e.clear();
50     for (int i = 1; i <= n; ++i) {
51       if (i != c) {
52         int r = testCC(center[c], center[i], radius[c], radius[i]);
53         if (r == 2) ++cvrcnt;
54         else if (r == 0) {
55           pair<double, double> paa = isCC(center[c], center[i], radius[c], radius[i
                 ]);
56           add(paa.first, paa.second);
57         }
58       }
59     }
60     if (e.size() == 0) {
61       double a = pi * radius[c] * radius[c];
62       cntarea[cvrcnt] -= a;
63       cntarea[cvrcnt + 1] += a;
64     } else {
65       e.push_back(event_t(-pi, 1));
66       e.push_back(event_t(pi, -2));
67       sort(e.begin(), e.end());
68       for (int i = 0; i < int(e.size()) - 1; ++i) {
69         cvrcnt += e[i].delta;
70         double a = calc(center[c], radius[c], e[i].theta, e[i + 1].theta);
71         cntarea[cvrcnt - 1] -= a;
72         cntarea[cvrcnt] += a;
73       }
74     }
75   }
76 }
```

4.9 Minimum Covering Circle

```
1 void set_circle(point_t &p, double &r, point_t a, point_t b) {
2   r = dist(a, b) / 2;
3   p = (a + b) / 2;
4 }
```

```
5
6 void set_circle(point_t &p, double &r, point_t a, point_t b, point_t c) {
7   if (dblcmp(cross(b - a, c - a)) == 0) {
8     if (dist(a, c) > dist(b, c)) {
9       r = dist(a, c) / 2;
10      p = (a + c) / 2;
11    } else {
12      r = dist(b, c) / 2;
13      p = (b + c) / 2;
14    }
15  } else {
16    p = circumcenter(a, b, c);
17    r = dist(p, a);
18  }
19 }
20
21 bool in_circle(point_t &p, double &r, point_t x) {
22   return dblcmp(dist(x, p) - r) <= 0;
23 }
24
25 pair<point_t, double> minimum_circle(int n, point_t *p) {
26   point_t c = point_t(0, 0);
27   double r = 0;
28   random_shuffle(p + 1, p + 1 + n);
29   set_circle(c, r, p[1], p[2]);
30   for (int i = 3; i <= n; ++i) {
31     if (in_circle(c, r, p[i])) continue;
32     set_circle(c, r, p[i], p[1]);
33     for (int j = 2; j < i; ++j) {
34       if (in_circle(c, r, p[j])) continue;
35       set_circle(c, r, p[i], p[j]);
36       for (int k = 1; k < j; ++k) {
37         if (in_circle(c, r, p[k])) continue;
38         set_circle(c, r, p[i], p[j], p[k]);
39       }
40     }
41   }
42   return make_pair(c, r);
43 }
```

4.10 Convex Polygon Area Union

```
1 // modified from syntax_error's code
2 bool operator<(const point_t &a, const point_t &b) {
3   if (dblcmp(a.x - b.x) == 0) return a.y < b.y;
4   return a.x < b.x;
5 }
6
```

```
 7 bool operator==(const point_t &a, const point_t &b) {
 8   return dblcmp(a.x - b.x) == 0 && dblcmp(a.y - b.y) == 0;
 9 }
10
11 struct segment_t {
12   point_t a, b;
13   segment_t() { a = b = point_t(); }
14   segment_t(point_t ta, point_t tb) : a(ta), b(tb) { }
15   double len() const { return dist(a, b); }
16   double k() const { return (a.y - b.y) / (a.x - b.x); }
17   double l() const { return a.y - k() * a.x; }
18 };
19
20 struct line_t {
21   double a, b, c;
22   line_t(point_t p) { a = p.x, b = -1.0, c = -p.y; }
23   line_t(point_t p, point_t q) {
24     a = p.y - q.y;
25     b = q.x - p.x;
26     c = a * p.x + b * p.y;
27   }
28 };
29
30 bool ccutl(line_t p, line_t q) {
31   if (dblcmp(p.a * q.b - q.a * p.b) == 0) return false;
32   return true;
33 }
34
35 point_t cutl(line_t p, line_t q) {
36   double x = (p.c * q.b - q.c * p.b) / (p.a * q.b - q.a * p.b);
37   double y = (p.c * q.a - q.c * p.a) / (p.b * q.a - q.b * p.a);
38   return point_t(x, y);
39 }
40
41 bool onseg(point_t p, segment_t s) {
42   if (dblcmp(p.x - min(s.a.x, s.b.x)) < 0 || dblcmp(p.x - max(s.a.x, s.b.x)) > 0)
43       return false;
43   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;
44   return true;
45 }
46
47 bool ccut(segment_t p, segment_t q) {
48   if (!ccutl(line_t(p.a, p.b), line_t(q.a, q.b))) return false;
49   point_t r = cutl(line_t(p.a, p.b), line_t(q.a, q.b));
50   if (!onseg(r, p) || !onseg(r, q)) return false;
51   return true;
52 }
53
54 point_t cut(segment_t p, segment_t q) {
55   return cutl(line_t(p.a, p.b), line_t(q.a, q.b));
56 }
57
58 struct event_t {
59   double x;
60   int type;
61   event_t() { x = 0, type = 0; }
62   event_t(double _x, int _t) : x(_x), type(_t) { }
63   bool operator<(const event_t &r) const {
64     return x < r.x;
65   }
66 };
67
68 vector<segment_t> s;
69
70 double solve(const vector<segment_t> &v, const vector<int> &sl) {
71   double ret = 0;
72   vector<point_t> lines;
73   for (int i = 0; i < v.size(); ++i) lines.push_back(point_t(v[i].k(), v[i].l()));
74   sort(lines.begin(), lines.end());
75   lines.erase(unique(lines.begin(), lines.end()), lines.end());
76   for(int i = 0; i < lines.size(); ++i) {
77     vector<event_t> e;
78     vector<int>::const_iterator it = sl.begin();
79     for(int j = 0; j < s.size(); j += *it++) {
80       bool touch = false;
81       for (int k = 0; k < *it; ++k) if (lines[i] == point_t(s[j + k].k(), s[j + k].l
            ())) touch = true;
82       if (touch) continue;
83       vector<point_t> cuts;
84       for (int k = 0; k < *it; ++k) {
85         if (!ccutl(line_t(lines[i]), line_t(s[j + k].a, s[j + k].b))) continue;
86         point_t r = cutl(line_t(lines[i]), line_t(s[j + k].a, s[j + k].b));
87         if (onseg(r, s[j + k])) cuts.push_back(r);
88       }
89       sort(cuts.begin(), cuts.end());
90       cuts.erase(unique(cuts.begin(), cuts.end()), cuts.end());
91       if (cuts.size() == 2) {
92         e.push_back(event_t(cuts[0].x, 0));
93         e.push_back(event_t(cuts[1].x, 1));
94       }
95     }
96     for (int j = 0; j < v.size(); ++j) {
97       if (lines[i] == point_t(v[j].k(), v[j].l())) {
98         e.push_back(event_t(min(v[j].a.x, v[j].b.x), 2));
99         e.push_back(event_t(max(v[j].a.x, v[j].b.x), 3));
```

```
100        }
101      }
102      sort(e.begin(), e.end());
103      double last = e[0].x;
104      int cntg = 0, cntb = 0;
105      for (int j = 0; j < e.size(); ++j) {
106        double y0 = lines[i].x * last + lines[i].y;
107        double y1 = lines[i].x * e[j].x + lines[i].y;
108        if (cntb == 0 && cntg) ret += (y0 + y1) * (e[j].x - last) / 2;
109        last = e[j].x;
110        if (e[j].type == 0) ++cntb;
111        if (e[j].type == 1) --cntb;
112        if (e[j].type == 2) ++cntg;
113        if (e[j].type == 3) --cntg;
114      }
115    }
116    return ret;
117 }
118
119 double polyUnion(vector<vector<point_t> > polys) {
120    s.clear();
121    vector<segment_t> A, B;
122    vector<int> sl;
123    for (int i = 0; i < polys.size(); ++i) {
124      double area = 0;
125      int tot = polys[i].size();
126      for (int j = 0; j < tot; ++j) {
127        area += cross(polys[i][j], polys[i][(j + 1) % tot]);
128      }
129      if (dblcmp(area) > 0) reverse(polys[i].begin(), polys[i].end());
130      if (dblcmp(area) != 0) {
131        sl.push_back(tot);
132        for (int j = 0; j < tot; ++j) s.push_back(segment_t(polys[i][j], polys[i][(j +
               1) % tot]));
133      }
134    }
135    for (int i = 0; i < s.size(); ++i) {
136      int sgn = dblcmp(s[i].a.x - s[i].b.x);
137      if (sgn == 0) continue;
138      else if (sgn < 0) A.push_back(s[i]);
139      else B.push_back(s[i]);
140    }
141    return solve(A, sl) - solve(B, sl);
142 }
```

4.11   3D Common

```
1 double dot(point_t p1, point_t p2) {
```

```
2    return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
3 }
4
5 point_t cross(point_t p1, point_t p2) {
6    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);
7 }
8
9 double volume(point_t p1, point_t p2, point_t p3, point_t p4) {
10    point_t v1 = cross(p2 - p1, p3 - p1);
11    p4 = p4 - p1;
12    return dot(v1, p4) / 6;
13 }
14
15 double area(point_t p1, point_t p2, point_t p3) {
16    return cross(p2 - p1, p3 - p1).length() / 2;
17 }
18
19 pair<point_t, point_t> isFF(point_t p1, point_t o1, point_t p2, point_t o2) {
20    point_t e = cross(o1, o2), v = cross(o1, e);
21    double d = dot(o2, v);
22    if (fabs(d) < eps) throw -1;
23    point_t q = p1 + (v * (dot(o2, p2 - p1) / d));
24    return make_pair(q, q + e);
25 }
26
27 double distLL(point_t p1, point_t u, point_t p2, point_t v) {
28    double s = dot(u, v) * dot(v, p1 - p2) - dot(v, v) * dot(u, p1 - p2);
29    double t = dot(u, u) * dot(v, p1 - p2) - dot(u, v) * dot(u, p1 - p2);
30    double deno = dot(u, u) * dot(v, v) - dot(u, v) * dot(u, v);
31    if (dblcmp(deno) == 0) return dist(p1, p2 + v * (dot(p1 - p2, u) / dot(u, v)));
32    s /= deno; t /= deno;
33    point_t a = p1 + u * s, b = p2 + v * t;
34    return dist(a, b);
35 }
```

4.12   3D Convex Hull

```
1 int n, bf[maxn][maxn], fcnt;
2 point_t pt[maxn];
3 struct face_t {
4    int a, b, c;
5    bool vis;
6 } fc[maxn << 5]; /* Number of Faces(Unknown) */
7
8 bool remove(int p, int b, int a) {
9    int f = bf[b][a];
10    face_t ff;
```

```
11    if (fc[f].vis) {
12      if (dblcmp(volume(pt[p], pt[fc[f].a], pt[fc[f].b], pt[fc[f].c])) >= 0) {
13        return true;
14      } else {
15        ff.a = a, ff.b = b, ff.c = p;
16        bf[ff.a][ff.b] = bf[ff.b][ff.c] = bf[ff.c][ff.a] = ++fcnt;
17        ff.vis = true;
18        fc[fcnt] = ff;
19      }
20    }
21    return false;
22 }
23
24 void dfs(int p, int f) {
25    fc[f].vis = false;
26    if (remove(p, fc[f].b, fc[f].a)) dfs(p, bf[fc[f].b][fc[f].a]);
27    if (remove(p, fc[f].c, fc[f].b)) dfs(p, bf[fc[f].c][fc[f].b]);
28    if (remove(p, fc[f].a, fc[f].c)) dfs(p, bf[fc[f].a][fc[f].c]);
29 }
30
31 void hull3d() {
32    for (int i = 2; i <= n; ++i) {
33      if (dblcmp((pt[i] - pt[1]).length()) > 0) swap(pt[i], pt[2]);
34    }
35    for (int i = 3; i <= n; ++i) {
36      if (dblcmp(fabs(area(pt[1], pt[2], pt[i]))) > 0) swap(pt[i], pt[3]);
37    }
38    for (int i = 4; i <= n; ++i) {
39      if (dblcmp(fabs(volume(pt[1], pt[2], pt[3], pt[i]))) > 0) swap(pt[i], pt[4]);
40    }
41    zm(fc), fcnt = 0, zm(bf);
42    for (int i = 1; i <= 4; ++i) {
43      face_t f;
44      f.a = i + 1, f.b = i + 2, f.c = i + 3;
45      if (f.a > 4) f.a -= 4;
46      if (f.b > 4) f.b -= 4;
47      if (f.c > 4) f.c -= 4;
48      if (dblcmp(volume(pt[i], pt[f.a], pt[f.b], pt[f.c])) > 0) swap(f.a, f.b);
49      f.vis = true;
50      bf[f.a][f.b] = bf[f.b][f.c] = bf[f.c][f.a] = ++fcnt;
51      fc[fcnt] = f;
52    }
53    random_shuffle(pt + 5, pt + 1 + n);
54    for (int i = 5; i <= n; ++i) {
55      for (int j = 1; j <= fcnt; ++j) {
56        if (!fc[j].vis) continue;
57        if (dblcmp(volume(pt[i], pt[fc[j].a], pt[fc[j].b], pt[fc[j].c])) >= 0) {
58          dfs(i, j);
```

```
59          break;
60        }
61      }
62    }
63    for (int i = 1; i <= fcnt; ++i) if (!fc[i].vis) swap(fc[i--], fc[fcnt--]);
64 }
```

5   Graph

5.1   Tarjan

```
1 //求强联通分量
2 void tarjan(int u)
3 {
4    low[u] = dfn[u] = ++curDfn;
5    sta[sta_n++] = u;
6    for (int i = head[u]; i; i = E[i].nxt)
7    {
8      if (!dfn[E[i].nxt])
9      {
10        tarjan(E[i].nxt);
11        low[u] = min(low[u], low[E[i].v]);
12      }
13      else
14        if (!sccNum[E[i].v])
15          low[u] = min(low[u], dfn[E[i].v]);
16    }
17    if (low[u] == dfn[u])
18    {
19      nScc++;
20      int v;
21      do
22        v = sta[--sta_n], sccNum[v] = nScc;
23      while (u != v);
24    }
25 }
26 // 求点双连通分量+割点
27 void tarjan(int u, int peid = -1)
28 {
29    low[u] = dfn[u] = ++curDfn;
30    int nSubtree = 0;
31    for (int i = head[u]; i = E[i].nxt; i = E[i].nxt)
32      if (i != peid)
33      {
34        if (!dfn[E[i].v])
35        {
36          sta[sta_n++] = i;
```

```
37        tarjan(E[i].v, i ^ 1);
38        nSubtree++;
39        if (low[E[i].v] >= dfn[v])
40        {
41          if (dfn[u] != 1)
42            isCutPoint[u] = true;
43          nBcc++;
44          int e;
45          do
46            e = sta[--sta_n], bccNum[e] = nBcc;
47          while (e != i);
48        }
49        low[u] = min(low[u], low[E[i].v]);
50      }
51      else
52      {
53        low[u] = min(low[u], dfn[E[i].v]);
54        if (dfn[E[i].v] < dfn[u])
55          sta[sta_n++] = i;
56      }
57    }
58  if (dfn[u] == 1 && nSubtree >= 2)
59    isCutPoint[u] = true;
60 }
61
62 // 求边双连通分量+桥
63 void tarjan(int u, int fa = -1)
64 {
65   low[u] = dfn[u] = ++curDfn;
66   sta[sta_n++] = u;
67   inSta[u] = 1;
68   for (int i = head[u]; i; i = E[i].nxt)
69     if (E[i].v != fa)
70     {
71       if (!dfn[E[i].v])
72       {
73         tarjan(E[i].v, u);
74         low[u] = min(low[u], low[E[i].v]);
75       }
76       else
77         if (inSta[E[i].v])
78           low[u] = min(low[u], dfn[E[i].v]);
79     }
80   if (low[u] == dfn[u])  //edge fa <-> u is a bridge
81   {
82     int v;
83     do
84       v = sta[--sta_n], ebccNum[v] = u, inSta[v] = 0;
85     while (u != v);
86   }
87 }
```

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(ISAP)

```
1  // assuming the sink has the maximum vertex ID => t = n
2  int n, m, s, t, ec, d[maxn], vd[maxn];
3  struct edge_link {
4    int v, r;
5    edge_link *next, *pair;
6  } edge[maxm], *header[maxn], *current[maxn];
7  void add(int u, int v, int r) // (u, v, r), (v, u, 0)
8  {
9    ec++;
10   edge[ec].v = v; edge[ec].r = r;
11   edge[ec].next = header[u]; header[u] = &edge[ec];
12   edge[ec].pair = &edge[ec+1];
13   ec++;
14   edge[ec].pair = &edge[ec-1];
15   edge[ec].next = header[v]; header[v] = &edge[ec];
16   edge[ec].v = u; edge[ec].r = 0;
17 }
18 int augment(int u, int flow) {
19   if (u == t) return flow;
20   int temp, res = 0;
21   for (edge_link *&e = current[u]; e != NULL; e = e->next) {
22     if (e->r && d[u] == d[e->v] + 1) {
23       temp = augment(e->v, min(e->r, flow - res));
24       e->r -= temp, e->pair->r += temp, res += temp;
25       if (d[s] == t || res == flow) return res;
26     }
27   }
28   if (--vd[d[u]] == 0) d[s] = t;
29   else current[u] = header[u], ++vd[++d[u]];
30   return res;
31 }
32
33 int sap() {
34   int flow = 0;
35   memset(d, 0, sizeof(d)), memset(vd, 0, sizeof(vd));
36   vd[0] = t;
```

```
37    for (int i = 1; i <= t; ++i) current[i] = header[i];
38    while (d[s] < t) flow += augment(s, maxint);
39    return flow;
40 }
```

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.

上下界网络流：

1：无源汇的可行流：新建源点，汇点，M[i] 为每个点进来的下界流减去出去的下界流，如果 M[i] 为正，由源点向改点建 M[i] 的边，反之，由该点向汇点建 M[i] 的边，原图中的边为每条边的上界建去下界。跑一遍最大流，每条边的流量加上下界流就是答案。

2：有源汇的最大流：从汇点向源点建一条容量为 INF 的边，用上面的方法判断是否有解，有解就再跑一遍从原图中源点到汇点的最大流

3：有源汇的最小流：先跑一遍最大流，然后连上从汇点到源点的边，再按照 1 的方法做就好了

5.3   Maximum Flow(HLPP)

```cpp
1  #include <cstdio>
2  #include <cstring>
3  #include <cstdlib>
4  #include <cmath>
5  #include <ctime>
6  #include <iostream>
7  #include <algorithm>
8  #include <set>
9  #include <map>
10 #include <vector>
11 #include <string>
12 #include <queue>
13 using namespace std;
14 typedef long long LL;
15 #define For(i,a,b) for (int i = (a); i <= (b); i++)
16 #define Cor(i,a,b) for (int i = (a); i >= (b); i--)
17 #define Fill(a,b) memset(a,b,sizeof(a))
18
19 const int MAX_SIDE = 100000, MAX_NODE = 100000;
20
21 class network_flow
22 {
23     private:
24
25     int label[MAX_NODE], GAP[MAX_NODE];
26     bool visited[MAX_NODE];
27     int in_flow[MAX_NODE];
28     int vS, vT;
29     int tot_node, label_max;
30     queue<int> active[MAX_NODE];
31
32 public:
33     struct EDGES
34     {
35         int pre[MAX_SIDE], val[MAX_SIDE], node[MAX_SIDE], last[MAX_NODE];
36         int tot;
37         EDGES()
38         {
39             tot = 1;
40         }
41         void add_edge(int s, int t, int v)
42         {
43             tot++;
44             pre[tot] = last[s];
45             node[tot] = t;
46             val[tot] = v;
47             last[s] = tot;
48         }
49     }edge;
50     void set_s(int s) { vS = s; }
51     void set_t(int t) { vT = t; }
52     void set_tot_node(int n) { tot_node = n; }
53     void HLPP()
54     {
55         get_length();
56         prepare();
57         max_flow();
58     }
59
60     void add_edge(int s, int t, int v)
61     {
62         edge.add_edge(s, t, v);
63         edge.add_edge(t, s, 0);
64     }
65
66     void max_flow()
67     {
68         while(label_max)
69         {
70             if(active[label_max].empty())
71             {
72                 label_max--;
```

```
 73                           continue;
 74                   }
 75
 76           int now = active[label_max].front(), ext, val;
 77           int label_min = tot_node + 1;
 78           int push_flow;
 79
 80           active[label_max].pop();
 81           for(int pos(edge.last[now]); pos; pos = edge.pre[pos])
 82           {
 83               ext = edge.node[pos];
 84               val = edge.val[pos];
 85               if(val > 0)
 86               {
 87                   if(label[ext] + 1 == label[now])
 88                   {
 89                       push_flow = min(val, in_flow[now]);
 90                       edge.val[pos] -= push_flow;
 91                       edge.val[pos ^ 1] += push_flow;
 92                       in_flow[now] -= push_flow;
 93                       in_flow[ext] += push_flow;
 94                       if(push_flow)
 95                           active[label[ext]].push(ext);
 96                   }
 97               }
 98               if(edge.val[pos])
 99                   label_min = min(label_min, label[ext]);
100               if(!in_flow[now])
101                   break;
102           }
103           if(in_flow[now] && now != vT && label_min < tot_node)
104           {
105               int cache = label[now];
106               GAP[label[now]] --;
107               label[now] = label_min + 1;
108               GAP[label[now]] ++;
109               if(GAP[cache] == 0)
110               {
111                   for(int i(1); i <= tot_node; ++ i)
112                       if(label[i] > cache && label[i] < tot_node + 1)
113                       {
114                           GAP[label[i]]--;
115                           GAP[tot_node + 1]++;
116                           label[i] = tot_node + 1;
117                       }
118               }
119               active[label[now]].push(now);
120               if(label[now] > label_max)
121                       label_max= label[now];
122               }
123           }
124   }
125
126   void prepare()
127   {
128       for(int pos(edge.last[vS]); pos; pos = edge.pre[pos])
129       {
130           int ext = edge.node[pos], val = edge.val[pos];
131           if(val > 0)
132           {
133               in_flow[ext] += val;
134               edge.val[pos] -= val;
135               edge.val[pos ^ 1] += val;
136               if(label[ext] > label_max)
137                   label_max = label[ext];
138               active[label[ext]].push(ext);
139           }
140       }
141   }
142
143   void get_length()
144   {
145       int queue[MAX_NODE];
146
147       fill(label + 1, label + 1 + tot_node, tot_node + 1);
148       memset(visited, 0, sizeof(visited));
149       queue[0]  = vT;
150       label[vT]= 0;
151       visited[vT] = 1;
152       GAP[0] = 1;
153       GAP[tot_node + 1] = tot_node - 1;
154
155       for(int p1(0), p2(0); p1 <= p2; ++ p1)
156       {
157           int now = queue[p1], ext;
158           for(int pos(edge.last[now]); pos; pos = edge.pre[pos])
159           {
160               ext = edge.node[pos];
161               if(!visited[ext])
162               {
163                   visited[ext] = 1;
164                   queue[++p2] = ext;
165                   label[ext] = label[now] + 1;
166                   GAP[tot_node + 1]--;
167                   GAP[label[ext]]++;
168               }
```

```
169                      }
170                   }
171              }
172          int get_ans()
173          {
174              return in_flow[vT];
175          }
176 }net;
```

5.4 Minimum Cost, Maximum Flow

```
1  using namespace std;
2  int n, m, s, t, ec, d[maxn]; // Minimum cost, maximum flow(ZKW version), t=n
3  struct edge_link {
4    int v, r, w;
5    edge_link *next, *pair;
6  } edge[maxm], *header[maxn];
7  bool vis[maxn];
8  //void add(int u, int v, int r, int w) // (u, v, r, w), (v, u, 0, -w)
9  void add(int u, int v, int r)
10 {
11   ec++;
12   edge[ec].v = v; edge[ec].r = r; edge[ec].w = w;
13   edge[ec].next = header[u]; header[u] = &edge[ec];
14   edge[ec].pair = &edge[ec+1];
15   ec++;
16   edge[ec].pair = &edge[ec-1];
17   edge[ec].next = header[v]; header[v] = &edge[ec];
18   edge[ec].v = u; edge[ec].r = 0; edge[ec].r = -w;
19 }
20 void spfa() {
21   queue<int> q;
22   for (int i = 1; i <= t; ++i) d[i] = maxint, vis[i] = false;
23   d[s] = 0, q.push(s), vis[s] = true;
24   while (!q.empty()) {
25     int u = q.front();
26     q.pop(), vis[u] = false;
27     for (edge_link *e = header[u]; e != NULL; e = e->next) {
28       if (e->r && d[u] + e->w < d[e->v]) {
29         d[e->v] = d[u] + e->w;
30         if (!vis[e->v]) q.push(e->v), vis[e->v] = true;
31       }
32     }
33   }
34   for (int i = 1; i <= t; ++i) d[i] = d[t] - d[i];
35 }
36
37 int augment(int u, int flow) {
38   if (u == t) return flow;
39   vis[u] = true;
40   for (edge_link *e = header[u]; e != NULL; e = e->next) {
41     if (e->r && !vis[e->v] && d[e->v] + e->w == d[u]) {
42       int temp = augment(e->v, min(flow, e->r));
43       if (temp) {
44         e->r -= temp, e->pair->r += temp;
45         return temp;
46       }
47     }
48   }
49   return 0;
50 }
51
52 bool adjust() {
53   int delta = maxint;
54   for (int u = 1; u <= t; ++u) {
55     if (!vis[u]) continue;
56     for (edge_link *e = header[u]; e != NULL; e = e->next) {
57       if (e->r && !vis[e->v] && d[e->v] + e->w > d[u]) {
58         delta = min(delta, d[e->v] + e->w - d[u]);
59       }
60     }
61   }
62   if (delta == maxint) return false;
63   for (int i = 1; i <= t; ++i) {
64     if (vis[i]) d[i] += delta;
65   }
66   memset(vis, 0, sizeof(vis));
67   return true;
68 }
69
70 pair<int, int> cost_flow() {
71   int temp, flow = 0, cost = 0;
72   spfa();
73   do {
74     while (temp = augment(s, maxint)) {
75       flow += temp;
76       memset(vis, 0, sizeof(vis));
77     }
78   } while (adjust());
79   for (int i = 2; i <= ec; i += 2) cost += edge[i].r * edge[i - 1].w;
80   return make_pair(flow, cost);
81 }
82 void init()
83 {
84
85 }
```

5.5   Kuhn Munkras

```
1  // Kuhn-Munkras's algorithm, maxn: Left Size; maxm: Right Size.
2  int n, m, g[maxn][maxm], lx[maxn], ly[maxm], slack[maxm], match[maxm];
3  bool vx[maxn], vy[maxm];
4
5  bool find(int x) {
6    vx[x] = true;
7    for (int y = 1; y <= m; ++y) {
8      if (!vy[y]) {
9        int delta = lx[x] + ly[y] - g[x][y];
10       if (delta == 0) {
11         vy[y] = true;
12         if (match[y] == 0 || find(match[y])) {
13           match[y] = x;
14           return true;
15         }
16       } else slack[y] = min(slack[y], delta);
17     }
18   }
19   return false;
20 }
21
22 int km() { // #define sm(p, f) memset((p), f, sizeof(p))
23   // maximum weight, if minimum, negate all g then restore at the end.
24   sm(lx, 0x80), sm(ly, 0), sm(match, 0);
25   for (int i = 1; i <= n; ++i) {
26     for (int j = 1; j <= m; ++j) lx[i] = max(lx[i], g[i][j]);
27   }
28   for (int k = 1; k <= n; ++k) {
29     sm(slack, 0x7f);
30     while (true) {
31       sm(vx, 0), sm(vy, 0);
32       if (find(k)) break;
33       else {
34         int delta = maxint;
35         for (int i = 1; i <= m; ++i) {
36           if (!vy[i]) delta = min(delta, slack[i]);
37         }
38         for (int i = 1; i <= n; ++i) {
39           if (vx[i]) lx[i] -= delta;
40         }
41         for (int i = 1; i <= m; ++i) {
42           if (vy[i]) ly[i] += delta;
43           if (!vy[i]) slack[i] -= delta;
44         }
45       }
46     }
47   }
48   int result = 0;
49   for (int i = 1; i <= n; ++i) result += lx[i];
50   for (int i = 1; i <= m; ++i) result += ly[i];
51   return result;
52 }
```

5.6   Hopcroft Karp

```
1  int n, m, vis[maxn], level[maxn], pr[maxn], pr2[maxn];
2  vector<int> edge[maxn]; // for Left
3
4  bool dfs(int u) {
5    vis[u] = true;
6    for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
7      int v = pr2[*it];
8      if (v == -1 || (!vis[v] && level[u] < level[v] && dfs(v))) {
9        pr[u] = *it, pr2[*it] = u;
10       return true;
11     }
12   }
13   return false;
14 }
15
16 int hopcroftKarp() {
17   memset(pr, -1, sizeof(pr)); memset(pr2, -1, sizeof(pr2));
18   for (int match = 0; ;) {
19     queue<int> Q;
20     for (int i = 1; i <= n; ++i) {
21       if (pr[i] == -1) {
22         level[i] = 0;
23         Q.push(i);
24       } else level[i] = -1;
25     }
26     while (!Q.empty()) {
27       int u = Q.front(); Q.pop();
28       for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
29         int v = pr2[*it];
30         if (v != -1 && level[v] < 0) {
31           level[v] = level[u] + 1;
32           Q.push(v);
33         }
34       }
35     }
36     for (int i = 1; i <= n; ++i) vis[i] = false;
37     int d = 0;
38     for (int i = 1; i <= n; ++i) if (pr[i] == -1 && dfs(i)) ++d;
39     if (d == 0) return match;
40     match += d;
```

```
41      }
42  }
```

5.7 Blossom Matching

```
1  int n, match[maxn], pre[maxn], base[maxn]; // maximum matching on graphs
2  vector<int> edge[maxn];
3  bool inQ[maxn], inB[maxn], inP[maxn];
4  queue<int> Q;
5
6  int LCA(int u, int v) {
7    for (int i = 1; i <= n; ++i) inP[i] = false;
8    while (true) {
9      u = base[u];
10     inP[u] = true;
11     if (match[u] == -1) break;
12     u = pre[match[u]];
13   }
14   while (true) {
15     v = base[v];
16     if (inP[v]) return v;
17     v = pre[match[v]];
18   }
19 }
20
21 void reset(int u, int a) {
22   while (u != a) {
23     int v = match[u];
24     inB[base[u]] = inB[base[v]] = true;
25     v = pre[v];
26     if (base[v] != a) pre[v] = match[u];
27     u = v;
28   }
29 }
30
31 void contract(int u, int v) {
32   int a = LCA(u, v);
33   for (int i = 1; i <= n; ++i) inB[i] = false;
34   reset(u, a), reset(v, a);
35   if (base[u] != a) pre[u] = v;
36   if (base[v] != a) pre[v] = u;
37   for (int i = 1; i <= n; ++i) {
38     if (!inB[base[i]]) continue;
39     base[i] = a;
40     if (!inQ[i]) Q.push(i), inQ[i] = true;
41   }
42 }
43
```

```
44  bool dfs(int s) {
45    for (int i = 1; i <= n; ++i) pre[i] = -1, inQ[i] = false, base[i] = i;
46    while (!Q.empty()) Q.pop();
47    Q.push(s), inQ[s] = true;
48    while (!Q.empty()) {
49      int u = Q.front();
50      Q.pop();
51      for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
52        int v = *it;
53        if (base[u] == base[v] || match[u] == v) continue;
54        if (v == s || (match[v] != -1 && pre[match[v]] != -1)) contract(u, v);
55        else if (pre[v] == -1) {
56          pre[v] = u;
57          if (match[v] != -1) {
58            Q.push(match[v]), inQ[match[v]] = true;
59          } else {
60            u = v;
61            while (u != -1) {
62              v = pre[u];
63              int w = match[v];
64              match[u] = v, match[v] = u;
65              u = w;
66            }
67            return true;
68          }
69        }
70      }
71    }
72    return false;
73  }
74
75  int blossom() {
76    int ans = 0;
77    for (int i = 1; i <= n; ++i) match[i] = -1;
78    for (int i = 1; i <= n; ++i) {
79      if (match[i] == -1 && dfs(i)) ++ans;
80    }
81    return ans;
82  }
```

5.8 Stoer Wagner

```
1  // stoer-wagner algorithm, complexity: O(n^3)
2  // used to compute the global minimum cut, and self-loop is ignored.
3  int n, g[maxn][maxn], v[maxn], d[maxn], vis[maxn];
4  int stoer_wagner(int n) {
5    int res = maxint;
6    for (int i = 1; i <= n; i++) v[i] = i, vis[i] = 0;
```

```
 7    while (n > 1) {
 8      int p = 2, prev = 1;
 9      for (int i = 2; i <= n; ++i) {
10        d[v[i]] = g[v[1]][v[i]];
11        if (d[v[i]] > d[v[p]]) p = i;
12      }
13      vis[v[1]] = n;
14      for (int i = 2; i <= n; ++i) {
15        if (i == n) {
16          res = min(res, d[v[p]]); // if d[v[p]] < res, then s = v[p] & t = v[prev]
17          for (int j = 1; j <= n; ++j) {
18            g[v[prev]][v[j]] += g[v[p]][v[j]];
19            g[v[j]][v[prev]] = g[v[prev]][v[j]];
20          }
21          v[p] = v[n--];
22          break;
23        }
24        vis[v[p]] = n;
25        prev = p;
26        p = -1;
27        for (int j = 2; j <= n; ++j) {
28          if (vis[v[j]] != n) {
29            d[v[j]] += g[v[prev]][v[j]];
30            if (p == -1 || d[v[p]] < d[v[j]]) p = j;
31          }
32        }
33      }
34    }
35    return res;
36  }
```

5.9   Arborescence

```
 1  /*最小树形图
 2  不定根的情况，造一个虚拟根， MAXINT 连上所有的点，最后答案减去 MAXINT 。
 3  求有向森林的同上，插0边即可。可以支持负边权求最大。*/
 4  int n, ec, ID[maxn], pre[maxn], in[maxn], vis[maxn];
 5  struct edge_t {
 6    int u, v, w;
 7  } edge[maxm];
 8  void add(int u, int v, int w) {
 9    edge[++ec].u = u, edge[ec].v = v, edge[ec].w = w;
10  }
11
12  int arborescence(int n, int root) {
13    int res = 0, index;
14    while (true) {
15      for (int i = 1; i <= n; ++i) {
```

```
16        in[i] = maxint, vis[i] = -1, ID[i] = -1;
17      }
18      for (int i = 1; i <= ec; ++i) {
19        int u = edge[i].u, v = edge[i].v;
20        if (u == v || in[v] <= edge[i].w) continue;
21        in[v] = edge[i].w, pre[v] = u;
22      }
23      pre[root] = root, in[root] = 0;
24      for (int i = 1; i <= n; ++i) {
25        res += in[i];
26        if (in[i] == maxint) return -1;
27      }
28      index = 0;
29      for (int i = 1; i <= n; ++i) {
30        if (vis[i] != -1) continue;
31        int u = i, v;
32        while (vis[u] == -1) {
33          vis[u] = i;
34          u = pre[u];
35        }
36        if (vis[u] != i || u == root) continue;
37        for (v = u, u = pre[u], ++index; u != v; u = pre[u]) ID[u] = index;
38        ID[v] = index;
39      }
40      if (index == 0) return res;
41      for (int i = 1; i <= n; ++i) if (ID[i] == -1) ID[i] = ++index;
42      for (int i = 1; i <= ec; ++i) {
43        int u = edge[i].u, v = edge[i].v;
44        edge[i].u = ID[u], edge[i].v = ID[v];
45        edge[i].w -= in[v];
46      }
47      n = index, root = ID[root];
48    }
49    return res;
50  }
```

5.10   Manhattan MST

```
 1  struct point {
 2    int x, y, index;
 3    bool operator<(const point &p) const { return x == p.x ? y < p.y : x < p.x; }
 4  } p[maxn];
 5  struct node {
 6    int value, p;
 7  } T[maxn];
 8
 9  int query(int x) {
10    int r = maxint, p = -1;
```

```
11    for (; x <= n; x += (x & -x)) if (T[x].value < r) r = T[x].value, p = T[x].p;
12    return p;
13  }
14
15  void modify(int x, int w, int p) {
16    for (; x > 0; x -= (x & -x)) if (T[x].value > w) T[x].value = w, T[x].p = p;
17  }
18
19  int manhattan() {
20    for (int i = 1; i <= n; ++i) p[i].index = i;
21    for (int dir = 1; dir <= 4; ++dir) {
22      if (dir == 2 || dir == 4) {
23        for (int i = 1; i <= n; ++i) swap(p[i].x, p[i].y);
24      } else if (dir == 3) {
25        for (int i = 1; i <= n; ++i) p[i].x = -p[i].x;
26      }
27      sort(p + 1, p + 1 + n);
28      vector<int> v; static int a[maxn];
29      for (int i = 1; i <= n; ++i) a[i] = p[i].y - p[i].x, v.push_back(a[i]);
30      sort(v.begin(), v.end()); v.erase(unique(v.begin(), v.end()), v.end());
31      for (int i = 1; i <= n; ++i) a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.
         begin() + 1;
32      for (int i = 1; i <= n; ++i) T[i].value = maxint, T[i].p = -1;
33      for (int i = n; i >= 1; --i) {
34        int pos = query(a[i]);
35        if (pos != -1) add(p[i].index, p[pos].index, dist(p[i], p[pos]));
36        modify(a[i], p[i].x + p[i].y, i);
37      }
38    }
39    return kruskal();
40  }
```

5.11 Minimum Mean Cycle

```
1  int dp[maxn][maxn]; // minimum mean cycle(allow negative weight)
2  double mmc(int n) {
3    for (int i = 0; i < n; ++i) {
4      memset(dp[i + 1], 0x7f, sizeof(dp[i + 1]));
5      for (int j = 1; j <= ec; ++j) {
6        int u = edge[j].u, v = edge[j].v, w = edge[j].w;
7        if (dp[i][u] != maxint) dp[i + 1][v] = min(dp[i + 1][v], dp[i][u] + w);
8      }
9    }
10   double res = maxdbl;
11   for (int i = 1; i <= n; ++i) {
12     if (dp[n][i] == maxint) continue;
13     double value = -maxdbl;
14     for (int j = 0; j < n; ++j) {
```

```
15       value = max(value, (double)(dp[n][i] - dp[j][i]) / (n - j));
16     }
17     res = min(res, value);
18   }
19   return res;
20  }
```

5.12 Divide and Conquer on Tree

```
1  int bk, size[maxn], parent[maxn], ver[maxn];
2  bool cut[maxn];
3
4  void bfs(int r) { // bfs in each sub-tree
5    parent[r] = 0, bk = 0; // maintain root extra information
6    static queue<int> Q; static stack<int> U;
7    Q.push(r);
8    while (!Q.empty()) {
9      int u = Q.front();
10     Q.pop(); U.push(u);
11     size[u] = 1, ver[++bk] = u; // find a node in sub-tree
12     for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
13       int v = *it;
14       if (v == parent[u] || cut[v]) continue;
15       parent[v] = u; // maintain v from u
16       Q.push(v);
17     }
18   }
19   while (!U.empty()) {
20     int u = U.top(); U.pop();
21     if (parent[u]) size[parent[u]] += size[u];
22   }
23  }
24
25  int findCentre(int r) {
26    static queue<int> Q;
27    int result = 0, rsize = maxint;
28    bfs(r);
29    Q.push(r);
30    while (!Q.empty()) {
31      int u = Q.front();
32      Q.pop();
33      int temp = size[r] - size[u];
34      for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
35        int v = *it;
36        if (cut[v] || v == parent[u]) continue;
37        temp = max(temp, size[v]);
38        Q.push(v);
39      }
```

```
40      if (temp < rsize) rsize = temp, result = u;
41    }
42    return result;
43  }
44
45  int work(int u) {
46    int result = 0;
47    u = findCentre(u);
48    cut[u] = true;
49    for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
50      int v = *it;
51      if (!cut[v]) /*result += */work(v); // process each sub-tree
52    }
53    for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
54      int v = *it;
55      if (cut[v]) continue;
56      bfs(v); // then combine sub-trees
57    }
58    cut[u] = false;
59    return result;
60  }
61
62  int fa[maxn], dep[maxn], son[maxn], size[maxn];
63  void dfs1(int x, int p, int depth)
64  {
65    fa[x] = p; dep[x] = depth;
66    size[x] = 1;
67    int maxsize = 0;
68    son[x] = 0;
69    for (int i = head[x]; i; i = E[i].nxt)
70    {
71      int v = E[i].v;
72      if (v != p)
73      {
74        dfs1(v, x, depth + 1);
75        size[x] += size[v];
76        if (size[v] > maxsize)
77        {
78          maxsize = size[v];
79          son[x] = v;
80        }
81      }
82    }
83  }
84  int top[maxn], p[maxn], fp[maxn], lable;
85  void dfs2(int x, int sp)
86  {
87    top[x] = sp;
```

```
88    p[x] = ++lable;
89    fp[p[x]] = x;
90    if (son[x])
91      dfs2(son[x], sp);
92    else
93      return;
94    for (int i = head[x]; i; i = E[i].nxt)
95    {
96      int v = E[i].v;
97      if (v != son[x] && v != fa[x])
98        dfs2(v, v);
99    }
100 }
101 int find(int u, int v)
102 {
103   int f1 = top[u], f2 = top[v];
104   int ret = 0;
105   while (f1 != f2)
106   {
107     if (dep[f1] < dep[f2])
108     {
109       swap(f1, f2);
110       swap(u, v);
111     }
112     ret = max(ret, seg.query(p[f1], p[u], 1, n, 1));
113     u = fa[f1]; f1 = top[u];
114   }
115   if (u == v)
116     return ret;
117   if (dep[u] > dep[v])
118     swap(u, v);
119   ret = max(ret, seg.query(p[son[u]], p[v], 1, n, 1));
120   return ret;
121 }
```

5.13   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.

```
1  int parent[maxn], label[maxn], cnt, real[maxn];
2  vector<int> edge[maxn], succ[maxn], pred[maxn];
3  int semi[maxn], idom[maxn], ancestor[maxn], best[maxn];
4  deque<int> bucket[maxn];
5
6  void dfs(int u) {
7    label[u] = ++cnt; real[cnt] = u;
```

```
 8     for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
 9       int v = *it;
10       if (v == parent[u] || label[v] != -1) continue;
11       parent[v] = u;
12       dfs(v);
13     }
14 }
15
16 void link(int v, int w) {
17   ancestor[w] = v;
18 }
19
20 void compress(int v) {
21   int a = ancestor[v];
22   if (ancestor[a] == 0) return;
23   compress(a);
24   if (semi[best[v]] > semi[best[a]]) best[v] = best[a];
25   ancestor[v] = ancestor[a];
26 }
27
28 int eval(int v) {
29   if (ancestor[v] == 0) return v;
30   compress(v);
31   return best[v];
32 }
33
34 void dominator() { // clear succ & pred, let cnt = 0 first
35   for (int i = 1; i <= n; ++i) label[i] = -1;
36   dfs(n); // n is root
37   for (int u = 1; u <= n; ++u) {
38     for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
39       int v = *it;
40       if (label[u] != -1 && label[v] != -1) {
41         succ[label[u]].push_back(label[v]);
42         pred[label[v]].push_back(label[u]);
43       }
44     }
45   }
46   for (int i = 1; i <= n; ++i) {
47     semi[i] = best[i] = i;
48     idom[i] = ancestor[i] = 0;
49     bucket[i].clear();
50   }
51   for (int w = cnt; w >= 2; --w) {
52     int p = label[parent[real[w]]];
53     for (vector<int>::iterator it = pred[w].begin(); it != pred[w].end(); ++it) {
54       int v = *it;
55       int u = eval(v);
56       if (semi[w] > semi[u]) semi[w] = semi[u];
57     }
58     bucket[semi[w]].push_back(w);
59     link(p, w);
60     while (!bucket[p].empty()) {
61       int v = bucket[p].front();
62       bucket[p].pop_front();
63       int u = eval(v);
64       idom[v] = (semi[u] < p ? u : p);
65     }
66   }
67   for (int w = 2; w <= cnt; ++w) {
68     if (idom[w] != semi[w]) idom[w] = idom[idom[w]];
69   }
70   idom[1] = 0;
71   for (int i = 1; i <= cnt; ++i) {
72     int u = real[idom[i]], v = real[i];
73     // u is immediate dominator of v (i == 1?)
74   }
75 }
```

5.14   Steiner's Problem

```
 1 // Steiner's Problem: ts[m], list of vertices to be united, indexed from 0.
 2 int steiner(int *ts, int m) { // O(3^m*n+2^m*n^2+n^3)
 3   floyd();
 4   memset(dp, 0, sizeof(dp));
 5   for (int i = 0; i < m; ++i) {
 6     for (int j = 1; j <= n; ++j) {
 7       dp[1 << i][j] = g[ts[i]][j];
 8     }
 9   }
10   for (int i = 1; i < (1 << m); ++i) {
11     if (((i - 1) & i) != 0) {
12       for (int j = 1; j <= n; ++j) {
13         dp[i][j] = maxint;
14         for (int k = (i - 1) & i; k > 0; k = (k - 1) & i) {
15           dp[i][j] = min(dp[i][j], dp[k][j] + dp[i ^ k][j]);
16         }
17       }
18       for (int j = 1; j <= n; ++j) {
19         for (int k = 1; k <= n; ++k) {
20           dp[i][j] = min(dp[i][j], dp[i][k] + g[k][j]);
21         }
22       }
23     }
24   }
25   return dp[(1 << m) - 1][ts[0]];
```

```
26 }
```

5.15   LCA

```
1  const int logn = 20;
2  int parent[maxn], lca[logn][maxn], depth[maxn];
3
4  void initLCA() {
5    for (int i = 1; i <= n; ++i) lca[0][i] = parent[i];
6    for (int j = 1; j < logn; ++j) {
7      for (int i = 1; i <= n; ++i) {
8        lca[j][i] = lca[j - 1][lca[j - 1][i]];
9      }
10   }
11 }
12
13 int LCA(int x, int y) {
14   if (depth[x] < depth[y]) swap(x, y);
15   for (int i = logn - 1; i >= 0; --i) {
16     if (depth[x] - (1 << i) >= depth[y]) x = lca[i][x];
17   }
18   if (x == y) return x;
19   for (int i = logn - 1; i >= 0; --i) {
20     if (lca[i][x] != lca[i][y]) {
21       x = lca[i][x], y = lca[i][y];
22     }
23   }
24   return lca[0][x];
25 }
```

5.16   Chordal Graph

一些结论:
    弦: 连接环中不相邻的两个点的边。
    弦图: 一个无向图称为弦图当且仅当图中任意长度大于 3 的环都至少有一个弦。
    单纯点: 设 N(v) 表示与点 v 相邻的点集。一个点称为单纯点当 v + N(v) 的诱导子图为一个团。
    完美消除序列: 这是一个序列 v[i], 它满足 v[i] 在 v[i..n] 的诱导子图中为单纯点。
    弦图的判定: 存在完美消除序列的图为弦图。可以用 MCS 最大势算法求出完美消除序列。
    最大势算法从 n 到 1 的顺序依次给点标号 (标号为 i 的点出现在完美消除序列的第 i 个)。设 label[i] 表示第 i 个点与多少个已标号的点相邻, 每次选择 label[i] 最大的未标号的点进行标号。
    判断一个序列是否为完美消除序列: 设 vi+1,⋯,vn 中所有与 vi 相邻的点依次为 vj1,⋯, vjk。只需判断 vj1 是否与 vj2,⋯, vjk 相邻即可。弦图的最大点独立集——完美消除序列从前往后能选就选。最小团覆盖数 = 最大点独立集数。

```
1  int label[10010], order[10010], seq[10010], color[10010], usable[10010];
2
3  int chordal() {
4    label[0] = -5555;
5    for(int i = N;i > 0;i--) {
6      int t = 0;
7      for(int j = 1;j <= N;j++) if(!order[j] && label[j] > label[t]) t = j;
8      order[t] = i; seq[i] = t;
9      for(auto y: edges[t]) label[y]++;
10   }
11
12   int ans = 0;
13   for(int i = N;i > 0;i--) {
14     for(auto y: edges[seq[i]]) usable[color[e->y]] = i;
15     int c = 1;
16     while(usable[c] == i) c++;
17     color[seq[i]] = c;
18     ans = max(ans, c)
19   }
20   return ans;
21 }
```

6   Planar Gragh

6.0.1   Euler Characteristic

$$\chi = V - E + F$$

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

6.0.2   Dual Graph

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

6.0.3   Maxflow on Planar Graph

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

7 Prufer Code

7.0.4 根据树构造

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

7.0.5 还原

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

7.0.6 一些结论

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

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

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

8 Miscellaneous

8.1 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])) {
      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();
}
```

8.2 AlphaBeta

```
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;
}
```

8.3 Dancing Links X

```
const int maxm = 2000, maxk = 500000;
struct dancingLinksX
{
    int pt, L[maxk], R[maxk], U[maxk], D[maxk];
    int C[maxk], A[maxk];
    int S[maxm], H[maxm];
```

```
7    int ans[maxm], totAns;
8    void init(int m) {
9        Fill(H, -1);
10       for (int i = 0; i <= m; ++i) S[i] = 0, L[i] = i - 1, R[i] = i + 1, D[i] = U[
             i] = i;
11       L[0] = m, R[m] = 0;
12       pt = m;
13   }
14   inline void insert(int row, int col) {
15           ++S[col], ++pt;
16           C[pt] = col, A[pt] = row, U[pt] = U[col], D[pt] = col;
17           D[U[col]] = pt, U[col] = pt;
18           if (~H[row]) {
19               L[pt] = L[H[row]], R[pt] = H[row], L[R[pt]] = R[L[pt]] = pt;
20           } else {
21               H[row] = L[pt] = R[pt] = pt;
22           }
23   }
24
25   inline void remove(int x) {
26       L[R[x]] = L[x], R[L[x]] = R[x];
27       for (int i = D[x]; i != x; i = D[i]) {
28           for (int j = R[i]; j != i; j = R[j]) {
29               U[D[j]] = U[j], D[U[j]] = D[j], --S[C[j]];
30           }
31       }
32   }
33
34   inline void resume(int x) {
35       for (int i = U[x]; i != x; i = U[i]) {
36           for (int j = L[i]; j != i; j = L[j]) {
37               U[D[j]] = j, D[U[j]] = j, ++S[C[j]];
38           }
39       }
40       L[R[x]] = x, R[L[x]] = x;
41   }
42
43   bool dlx(int k) {
44       if (R[0] == 0)
45       {
46           totAns = k;
47           return true;
48       }
49       int col = R[0];
50       for (int i = R[0]; i != 0; i = R[i]) {
51           if (S[col] > S[i]) col = i;
52       }
53       if (S[col] == 0) return false;
54       remove(col);
55       for (int i = D[col]; i != col; i = D[i]) {
56           ans[k] = A[i];
57           for (int j = R[i]; j != i; j = R[j]) remove(C[j]);
58           if (dlx(k + 1)) return true;
59           for (int j = L[i]; j != i; j = L[j]) resume(C[j]);
60       }
61       resume(col);
62       return false;
63   }
64   // call dlx(0)
65 }DLX;
```

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.

8.4 Mo-Tao Algorithm

```
1  // Complexity: Q*N^0.5 * O(add)
2  int SQRTN = (int)sqrt((double)q);
3  sort(Q + 1, Q + 1 + q, cmpL);
4  for (int i = 1; i <= q; i += SQRTN) {
5    clear();
6    int begin = i, end = i + SQRTN - 1;
7    if (end > q) end = q;
8    sort(Q + begin, Q + end + 1, cmpR);
9    Q[begin - 1].l = 1, Q[begin - 1].r = 0;
10   for (int j = begin; j <= end; ++j) {
11     for (int k = Q[j - 1].r + 1; k <= Q[j].r; ++k) add(k, 1);
12     if (Q[j].l > Q[j - 1].l) {
13       for (int k = Q[j - 1].l; k < Q[j].l; ++k) add(k, -1);
14     } else if (Q[j].l < Q[j - 1].l) {
15       for (int k = Q[j].l; k < Q[j - 1].l; ++k) add(k, 1);
16     }
17     ans[Q[j].ID] = res;
18   }
19 }
```

8.5 Digits Dp

```
 1 #include <cstdio>
 2 #include <cstring>
 3 #include <cmath>
 4 #include <cstdlib>
 5 #include <algorithm>
 6 #include <iostream>
 7 using namespace std;
 8 typedef long long LL;
 9 L1 getsum1(int n, int k)
10 {
11   LL B = 1;
12   for (int i = 0; i < n; i++) B *= k;
13   return B * n * (k - 1) / 2;
14 }
15 L2 getsum2(int prefixsum, int n, int k)
16 {
17   LL B = getsum1(n, k);
18   LL C = prefixsum;
19   for (int i = 0; i < n; i++)
20     C *= k;
21   return B + C;
22 }
23 LL getsum3(int prefixsum, long long n, int k)
24 {
25   if (n < k)
26   {
27     LL ret = 0;
28     for (int i = 0; i <= n; i++)
29       ret += prefixsum + i;
30     return ret;
31   }
32   LL t = 1, tn = n;
33   int d = 0;
34   while (tn >= k)
35   {
36     tn /= k;
37     t *= k;
38     d++;
39   }
40   LL ret = 0;
41   for (int i = 0; i < tn; i++)
42     ret += getsum2(prefixsum + i, d, k);
43   ret += getsum3(prefixsum + tn, n - tn * t, k);
44   return ret;
45 }
46 int main()
47 {
48   int k;
```

```
49   long long a, b;
50   scanf("%lld%lld%d",&a, &b, &k);
51   printf("%lld\n", getsum3(0,b,k) - getsum3(0,a-1,k));
52   return 0;
53 }
```

8.6 Plugin Dp

```
 1 //verison 1
 2 //hash
 3 const int maxn = 1000010;
 4 const int Hash_Mod = 30007;
 5 const int maxm = 15;
 6 int maze[maxm][maxm];
 7 int code[maxm];
 8 int n, m;
 9 struct HashMap
10 {
11   int head[Hash_Mod], nxt[maxn], size;
12   long long f[maxn],state[maxn];
13   void init()
14   {
15     memset(head,0,sizeof(head));
16     size = 0;
17   }
18   void push(long long st, long long ans)
19   {
20     int ht = st % Hash_Mod;
21     for (int i = head[ht]; i; i = nxt[i])
22     {
23       if (state[i] == st)
24       {
25         f[i] += ans;
26         return;
27       }
28     }
29     size++;
30     nxt[size] = head[ht]; head[ht] = size;
31     state[size] = st; f[size] = ans;
32   }
33 }hm[2];
34 void decode(int *code, int m, long long st)
35 {
36   for (int i = m; i >= 0; i--)
37   {
38     code[i] = st & 7;
39     st >>= 3;
40   }
```

```
41 }
42 int ch[maxm];
43 long long encode(int *code, int m)
44 {
45   long long st = 0;
46   memset(ch,-1,sizeof(ch));
47   ch[0] = 0;
48   int cnt = 1;
49   for (int i = 0; i <= m; i++)
50   {
51     if (ch[code[i]] == -1)
52       ch[code[i]] = cnt++;
53     code[i] = ch[code[i]];
54     st <<= 3;
55     st |= code[i];
56   }
57   return st;
58 }
59 //换行原理: 上一行行尾是----|,下一行行首是|----,水平线上的状态完全相同
60 void shift(int *code, int m)
61 {
62   for (int i = m; i > 0; i--)
63     code[i] = code[i-1];
64   code[0] = 0;
65 }
66 int ex, ey;
67 void dpblank(int i, int j, int cur)
68 {
69   int k, up, left;
70   for (k = 1; k <= hm[cur].size; k++)
71   {
72     decode(code, m, hm[cur].state[k]);
73     left = code[j-1];
74     up = code[j];
75     if (up && left)
76     {
77       if (up == left)
78       {
79         if (i == ex && j == ey)
80         {
81           code[j-1] = code[j] = 0;
82           if (j == m)
83             shift(code,m);
84           hm[cur ^ 1].push(encode(code,m),hm[cur].f[k]);
85         }
86       }
87       else
88       {
89         code[j] = code[j-1] = 0;
90         for (int p = 0; p <= m; p++)
91           if (code[p] == left)
92             code[p] = up;
93         if (j == m)
94           shift(code,m);
95         hm[cur ^ 1].push(encode(code,m),hm[cur].f[k]);
96       }
97     }
98     else
99       if ((left != 0 && up == 0) || (up != 0 && left == 0))
100      {
101        int p = left + up;
102        if (maze[i+1][j])
103        {
104          code[j-1] = p; code[j] = 0;
105          if (j == m)
106            shift(code,m);
107          hm[cur ^ 1].push(encode(code,m),hm[cur].f[k]);
108        }
109        if (maze[i][j+1])
110        {
111          code[j-1] = 0; code[j] = p;
112          hm[cur ^ 1].push(encode(code,m),hm[cur].f[k]);
113        }
114      }
115      else
116      {
117        if (maze[i][j+1] && maze[i+1][j])
118        {
119          code[j-1] = code[j] = 13;
120          hm[cur ^ 1].push(encode(code,m),hm[cur].f[k]);
121        }
122      }
123  }
124
125 }
126 void dpblock(int i, int j, int cur)
127 {
128   for (int k = 1; k <= hm[cur].size; k++)
129   {
130     decode(code, m, hm[cur].state[k]);
131     code[j-1] = code[j] = 0;
132     if (j == m)
133       shift(code,m);
134     hm[cur ^ 1].push(encode(code,m), hm[cur].f[k]);
135   }
136 }
```

```
137 void solve()
138 {
139   int cur = 0;
140   hm[cur].init();
141   hm[cur].push(0,1);
142   for (int i = 1; i <= n; i++)
143     for (int j = 1; j <= m; j++)
144     {
145       hm[cur ^ 1].init();
146       if (maze[i][j])
147         dpblank(i,j,cur);
148       else
149         dpblock(i,j,cur);
150       cur ^= 1;
151     }
152   long long ans = 0;
153   for (int i = 1; i <= hm[cur].size; i++)
154     ans += hm[cur].f[i];
155   printf("%lld\n", ans);
156 }
157
158 //version 2
159 //matrix improved
160 const int maxm = 7, maxs = 2187, pow3[] = {1, 3, 9, 27, 81, 243, 729, 2187}; // m+1,
      3**(m+1)
161 int n, m, top, stack[maxm + 1], match[maxs][maxm + 1], cnt, valid[maxs], ID[maxs];
162 int mt[N + 1][N + 1], mt2[N + 1][N + 1], f[maxm][maxs], g[maxs]; // N: cnt
163 int gb3(int v, int bit) { return (v / pow3[bit - 1]) % 3; }
164 int mb3(int v, int bit, int value) { return v - pow3[bit - 1] * gb3(v, bit) + pow3[
      bit - 1] * value; }
165 int ub3(int v, int bit) { return v - pow3[bit - 1] * gb3(v, bit); }
166 void upd(int &x, int v) { x += v; x %= mod; }
167
168 void dfs(int p, int st, int lt) {
169   if (p > m + 1) {
170     if (lt == 0) {
171       valid[st] = true;
172       if (st % 3 == 0) ID[st] = ++cnt; // omitted
173       top = 0;
174       for (int j = 1; j <= m + 1; ++j) {
175         if (gb3(st, j) == 1) {
176           stack[++top] = j;
177         } else if (gb3(st, j) == 2) {
178           match[st][j] = stack[top];
179           match[st][stack[top--]] = j;
180         }
181       }
182     }
```

```
183     } else {
184       dfs(p + 1, st, lt); // #
185       dfs(p + 1, mb3(st, p, 1), lt + 1); // (
186       if (lt) dfs(p + 1, mb3(st, p, 2), lt - 1); // )
187   }
188 }
189
190 int plugDP(int n, int m) {
191   memset(valid, 0, sizeof(valid)), memset(ID, -1, sizeof(ID)), cnt = 0;
192   memset(mt, 0, sizeof(mt)), memset(mt2, 0, sizeof(mt2));
193   dfs(1, 0, 0);
194   for (int start = 0; start < pow3[m + 1]; ++start) {
195     if (ID[start] == -1) continue;
196     for (int fi = 0; fi < 2; ++fi) { // Last colume flag
197       memset(f, 0, sizeof(f)); memset(g, 0, sizeof(g));
198       f[0][start] = 1;
199       for (int j = 0; j <= m; ++j) {
200         for (int k = 0; k < pow3[m + 1]; ++k) {
201           if (!valid[k]) continue;
202           if (j == m) {
203             if (!fi && !gb3(k, j + 1)) { // i != n
204               upd(g[ub3(k, j + 1) * 3], f[j][k]); // Unmark last bit
205             }
206           } else {
207             // Consider mp[i][j + 1] valid ? (bit1 = bit2 = 0)
208             int bit1 = gb3(k, j + 1), bit2 = gb3(k, j + 2),
209               t = ub3(ub3(k, j + 1), j + 2), tt;
210             if (bit1 == 1 && bit2 == 2) { // Merge two brackets
211               if (fi && j + 1 == m) { // i == n
212                 upd(f[j + 1][t], f[j][k]);
213               }
214             } else if (bit1 == 2 && bit2 == 1) {
215               upd(f[j + 1][t], f[j][k]);
216             } else if (!bit1 && !bit2) { // bit1 == 0 && bit2 == 0
217               tt = mb3(mb3(t, j + 1, 1), j + 2, 2);
218               upd(f[j + 1][tt], f[j][k]);
219             } else if (bit1 == bit2) {
220               if (bit1 == 1) { // bit1 == 1 && bit2 == 1
221                 tt = mb3(t, match[k][j + 2], 1);
222                 upd(f[j + 1][tt], f[j][k]);
223               } else { // bit1 == 2 && bit2 == 2
224                 tt = mb3(t, match[k][j + 1], 2);
225                 upd(f[j + 1][tt], f[j][k]);
226               }
227             } else { // bit1 == 0 || bit2 == 0
228               upd(f[j + 1][k], f[j][k]);
229               swap(bit1, bit2);
230               tt = mb3(mb3(t, j + 1, bit1), j + 2, bit2);
```

```
231            upd(f[j + 1][tt], f[j][k]);
232          }
233        }
234      }
235    }
236    if (fi) {
237      for (int i = 0; i < pow3[m + 1]; ++i) if (ID[i] != -1) mt2[ID[start]][ID[i]]
            = f[m][i];
238    } else {
239      for (int i = 0; i < pow3[m + 1]; ++i) if (ID[i] != -1) mt[ID[start]][ID[i]]
          = g[i];
240    }
241  }
242 }
243 matrix_t mat = 0, mat2 = 0, mf = 0;
244 for (int i = 1; i <= cnt; ++i) for (int j = 1; j <= cnt; ++j) {
245   mat.x[j][i] = mt[i][j], mat2.x[j][i] = mt2[i][j];
246 }
247 mat = mat2 * mat.power(n - 1);
248 mf.x[ID[0]/* ?? */][1] = 1;
249 mf = mat * mf;
250 return mf.x[ID[0]][1];
251 }
```

9 Tips

9.1 Useful Codes

- Accelerated C++ stream IO
```
1 #include <iomanip>
2 ios_base::sync_with_stdio(false);
```

- Enumerate all non-empty subsets
```
1 for (int sub = mask; sub > 0; sub = (sub - 1) & mask)
```

- Enumerate C_n^k
```
1 for (int comb = (1 << k) - 1; comb < 1 << n; ) {
2   // ...
3   int x = comb & -comb, y = comb + x;
4   comb = ((comb & ~y) / x >> 1) | y;
5 }
```

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

- Increase Stack Size
```
1 #pragma comment(linker,"/STACK:102400000,102400000")
2
3 register char *_sp __asm__("rsp"); // esp / sp
4 int main(void) {
5   const int size = 64*1024*1024;
6   static char *sys, *mine(new char[size]+size-4096);
7   sys = _sp; _sp = mine; mmain(); _sp = sys;
8   return 0;
9 }
```

- Compare Code
```
1 #!/bin/bash
2 while true; do
3 ./make_data
4 ./prog1
5 ./prog2
6 diff ans1.out ans2.out
7 if [ $? -ne 0 ] ; then break; fi
8 done
```

9.2 Formulas

9.2.1 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 $a = AB, b = BC, c = CA, d = OC, e = OA, f = OB$, $(12V)^2 = a^2 d^2(b^2 + c^2 + e^2 + f^2 - a^2 - d^2) + b^2 e^2(c^2 + a^2 + f^2 + d^2 - b^2 - e^2) + c^2 f^2(a^2 + b^2 + d^2 + e^2 - c^2 - f^2) - a^2 b^2 c^2 - a^2 e^2 f^2 - d^2 b^2 f^2 - d^2 e^2 c^2$.

■ **Radius of Inscribedcircle & Circumcircle**

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

■ **Euler Point**   欧拉线上的四点中，九点圆圆心到垂心和外心的距离相等，而且重心到外心的距离是重心到垂心距离的一半。

■ **Hypersphere**

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

■ **Affine Transformation**

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

$$= \begin{bmatrix} S_x\cos\alpha & -S_y\sin\alpha & T_x \\ S_x\sin\alpha & S_y\cos\alpha & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

The fixed point is (x_0, y_0), where

$$x_0 = -\frac{T_x(S_y\cos\alpha - 1)}{S_x S_y - S_x\cos\alpha - S_y\cos\alpha + 1} - \frac{S_y T_y\sin\alpha}{S_x S_y - S_x\cos\alpha - S_y\cos\alpha + 1}$$
$$y_0 = \frac{S_x T_x\sin\alpha}{S_x S_y - S_x\cos\alpha - S_y\cos\alpha + 1} - \frac{T_y(S_x\cos\alpha - 1)}{S_x S_y - S_x\cos\alpha - S_y\cos\alpha + 1}$$

■ **Matrix of rotating θ about arbitrary axis A** $|A| = 1$

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

9.2.2 Math

■ **Sums**

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

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

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

■ **C(n,m) 奇偶**   如果 (n & m)=m，那么为奇数，否则为偶数

■ **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**

$$C_r^k = \frac{r}{k} C_{r-1}^{k-1} \qquad C_r^k = C_{r-1}^k + C_{r-1}^{k-1}$$

$$C_r^m C_m^k = C_r^k C_{r-k}^{m-k} \qquad \sum_{k \leq n} C_{r+k}^k = C_{r+n+1}^n$$

$$\sum_{0 \leq k \leq n} C_k^m = C_{n+1}^{m+1} \qquad \sum_k C_r^k C_s^{n-k} = C_{r+s}^n$$

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$\geq n$, satisfying the constraint that any sum of the first k elements is always non-negative: $C_{m+n}^m - C_{m+n}^{m+1}$.
Specially, when $m = n$, it equals to $C_n = \frac{C_{2n}^n}{n+1}$, 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 $n \geq 0$.

■ **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 $n = 1$.

9.2.3 Integration

■ $ax + b(a \neq 0)$

1. $\int \frac{\mathrm{d}x}{ax+b} = \frac{1}{a}\ln|ax+b| + C$

2. $\int (ax+b)^\mu \mathrm{d}x = \frac{1}{a(\mu+1)}(ax+b)^{\mu+1} + C(\mu \neq 1)$

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

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

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

5. $\int \frac{\mathrm{d}x}{x^2(ax^2+b)} = -\frac{1}{bx} - \frac{a}{b}\int \frac{\mathrm{d}x}{ax^2+b}$

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

7. $\int \frac{\mathrm{d}x}{(ax^2+b)^2} = \frac{x}{2b(ax^2+b)} + \frac{1}{2b}\int \frac{\mathrm{d}x}{ax^2+b}$

■ $ax^2 + bx + c(a > 0)$

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

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

10. $\int \sqrt{(x^2+a^2)^3}\mathrm{d}x = \frac{x}{8}(2x^2+5a^2)\sqrt{x^2+a^2} + \frac{3}{8}a^4\ln(x+\sqrt{x^2+a^2}) + C$

11. $\int x\sqrt{x^2+a^2}\mathrm{d}x = \frac{1}{3}\sqrt{(x^2+a^2)^3} + C$

12. $\int x^2\sqrt{x^2+a^2}\mathrm{d}x = \frac{x}{8}(2x^2+a^2)\sqrt{x^2+a^2} - \frac{a^4}{8}\ln(x+\sqrt{x^2+a^2}) + C$

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

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

■ $\sqrt{x^2-a^2}(a>0)$

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

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

7. $\int \frac{\mathrm{d}x}{x\sqrt{x^2-a^2}} = \frac{1}{a}\arccos\frac{a}{|x|} + C$

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

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

10. $\int \sqrt{(x^2-a^2)^3}\mathrm{d}x = \frac{x}{8}(2x^2-5a^2)\sqrt{x^2-a^2} + \frac{3}{8}a^4\ln\left|x+\sqrt{x^2-a^2}\right| + C$

11. $\int x\sqrt{x^2-a^2}\mathrm{d}x = \frac{1}{3}\sqrt{(x^2-a^2)^3} + C$

12. $\int x^2\sqrt{x^2-a^2}\mathrm{d}x = \frac{x}{8}(2x^2-a^2)\sqrt{x^2-a^2} - \frac{a^4}{8}\ln\left|x+\sqrt{x^2-a^2}\right| + C$

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

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

■ $\sqrt{a^2-x^2}(a>0)$

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

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

10. $\int \sqrt{(a^2-x^2)^3}\mathrm{d}x = \frac{x}{8}(5a^2-2x^2)\sqrt{a^2-x^2} + \frac{3}{8}a^4\arcsin\frac{x}{a} + C$

11. $\int x\sqrt{a^2-x^2}\mathrm{d}x = -\frac{1}{3}\sqrt{(a^2-x^2)^3} + C$

12. $\int x^2\sqrt{a^2-x^2}\mathrm{d}x = \frac{x}{8}(2x^2-a^2)\sqrt{a^2-x^2} + \frac{a^4}{8}\arcsin\frac{x}{a} + C$

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

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

■ $\sqrt{\pm ax^2+bx+c}(a>0)$

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

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

3. $\int \frac{x}{\sqrt{ax^2+bx+c}}\mathrm{d}x = \frac{1}{a}\sqrt{ax^2+bx+c} - \frac{b}{2\sqrt{a^3}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right| + C$

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. $\int x^n a^x\mathrm{d}x = \frac{1}{\ln a}x^n a^x - \frac{n}{\ln a}\int x^{n-1}a^x\mathrm{d}x$

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. $\int \frac{\mathrm{d}x}{x\ln x} = \ln|\ln x| + C$

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

4. $\int(\ln x)^n\mathrm{d}x = x(\ln x)^n - n\int(\ln x)^{n-1}\mathrm{d}x$

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

■ **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. $\int \sec^2 x\mathrm{d}x = \tan x + C$

8. $\int \csc^2 x\mathrm{d}x = -\cot x + C$

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. $\int \sin^2 x\mathrm{d}x = \frac{x}{2} - \frac{1}{4}\sin 2x + C$

12. $\int \cos^2 x\mathrm{d}x = \frac{x}{2} + \frac{1}{4}\sin 2x + C$

13. $\int \sin^n x\mathrm{d}x = -\frac{1}{n}\sin^{n-1}x\cos x + \frac{n-1}{n}\int \sin^{n-2}x\mathrm{d}x$

14. $\int \cos^n x\mathrm{d}x = \frac{1}{n}\cos^{n-1}x\sin x + \frac{n-1}{n}\int \cos^{n-2}x\mathrm{d}x$

15. $\int \frac{\mathrm{d}x}{\sin^n x} = -\frac{1}{n-1}\frac{\cos x}{\sin^{n-1}x} + \frac{n-2}{n-1}\int \frac{\mathrm{d}x}{\sin^{n-2}x}$

16. $\int \frac{\mathrm{d}x}{\cos^n x} = \frac{1}{n-1}\frac{\sin x}{\cos^{n-1}x} + \frac{n-2}{n-1}\int \frac{\mathrm{d}x}{\cos^{n-2}x}$

17.
$$\int \cos^m x\sin^n x\mathrm{d}x$$
$$= \frac{1}{m+n}\cos^{m-1}x\sin^{n+1}x + \frac{m-1}{m+n}\int \cos^{m-2}x\sin^n x\mathrm{d}x$$
$$= -\frac{1}{m+n}\cos^{m+1}x\sin^{n-1}x + \frac{n-1}{m+1}\int \cos^m x\sin^{n-2}x\mathrm{d}x$$

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

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. $\int x\arccos\frac{x}{a}\,\mathrm{d}x = \left(\frac{x^2}{2} - \frac{a^2}{4}\right)\arccos\frac{x}{a} - \frac{x}{4}\sqrt{a^2-x^2} + C$

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

9.3 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

9.4 Vimrc

```
set smartindent
set cindent
set number
set st=4
set ts=4
set sw=4
map <F9> :w<cr>:!g++ % -o %< -g -Wall<cr>
map <C-F9> :!time ./%<<cr>
map <F4> :w<cr>:!gedit %<cr>
set smarttab
set nowrap
```

9.5 Makefile

```
all : prog

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

9.6 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;
```

```
22  StringTokenizer tokenizer;
23  public InputReader(InputStream stream) {
24    reader = new BufferedReader(new InputStreamReader(stream));
25    tokenizer = null;
26  }
27  public boolean hasNext() {
28    while (tokenizer == null || !tokenizer.hasMoreTokens()) {
29      try {
30        tokenizer = new StringTokenizer(reader.readLine());
31      } catch (Exception e) {
32        return false;
33      }
34    }
35    return tokenizer.hasMoreTokens();
36  }
37  public String next() {
38    while (tokenizer == null || !tokenizer.hasMoreTokens()) {
39      try {
40        tokenizer = new StringTokenizer(reader.readLine());
41      } catch (Exception e) {
42        throw new RuntimeException(e);
43      }
44    }
45    return tokenizer.nextToken();
46  }
47 }
48 import java.*;
49 import java.math.*;
50 import java.util.*;
51 public class Main
52 {
53   public static Scanner cin = new Scanner(System.in);
54   public static void main(String[] args)
55   {
56     int n = cin.nextInt();
57     BigInteger a[] = new BigInteger[200];
58     for (int i = 1; i <= n; i++)
59     {
60       BigInteger k = cin.nextBigInteger();
61       System.out.print("Case "+i+": ");
62       a[0] = BigInteger.valueOf(1);
63       for (int j = 1; j < 75; j++)
64         a[j] = a[j-1].multiply(k);
65       BigInteger ans = a[74];
66       ans = ans.add(a[38].multiply(BigInteger.valueOf(9)));
67       ans = ans.add(a[20].multiply(BigInteger.valueOf(6)));
68       ans = ans.add(a[26].multiply(BigInteger.valueOf(8)));
69       ans = ans.divide(BigInteger.valueOf(24));
```

```
70       ans = ans.mod(BigInteger.valueOf(10007));
71       System.out.println(ans);
72     }
73   }
74 }
```

9.7 Java Evaluate

```
1  import javax.script.ScriptEngineManager;
2  import javax.script.ScriptEngine;
3  public class Main {
4    public static void main(String[] args) throws Exception {
5      ScriptEngineManager mgr = new ScriptEngineManager();
6      ScriptEngine engine = mgr.getEngineByName("JavaScript");
7      String foo = "3+4";
8      System.out.println(engine.eval(foo));
9    }
10 }
```