

---

# ACM TEMPLATE



UESTC\_Jungle

Last build at October 25, 2018

# Contents

<b>1</b>	<b>Datastructure</b>	<b>3</b>
1.1	Fenwick . . . . .	3
1.2	BST in pb_ds . . . . .	4
1.3	Segment Tree . . . . .	4
1.4	Sparse Table . . . . .	7
1.5	Treap . . . . .	7
1.6	Leftist Heap . . . . .	10
1.7	Splay . . . . .	11
1.8	Persistent Segment Tree . . . . .	15
<b>2</b>	<b>Dynamic Programming</b>	<b>15</b>
2.1	LIS $O(n \log n)$ . . . . .	15
2.2	LCS $O(n \log n)$ . . . . .	15
2.3	Improved by quadrilateral inequality . . . . .	16
2.4	Improved by Slope . . . . .	17
2.5	Steiner Tree . . . . .	18
<b>3</b>	<b>Geometry</b>	<b>19</b>
3.1	2D . . . . .	19
3.1.1	Point . . . . .	19
3.1.2	Circle . . . . .	20
3.1.3	Convex hull . . . . .	26
3.1.4	Intersect Area . . . . .	27
3.1.5	Universe . . . . .	29
<b>4</b>	<b>Graph</b>	<b>36</b>
4.1	Tree . . . . .	36
4.1.1	Universe . . . . .	36
4.1.2	Point Divide and Conquer . . . . .	37
4.1.3	Heavy chain decomposition . . . . .	43
4.2	2-SAT . . . . .	46
4.3	Cut Edge and Point . . . . .	47
4.4	Euler Path . . . . .	48
4.5	Shortest Path . . . . .	49
4.5.1	Dijkstra . . . . .	49
4.5.2	Shortest Path Fast Algorithm . . . . .	50
4.5.3	$K$ -th shortest path . . . . .	51
4.6	Maxflow . . . . .	54
4.7	Strongly Connected Component . . . . .	58
4.8	Perfect elimination ordering . . . . .	61
<b>5</b>	<b>Math</b>	<b>63</b>
5.1	Euler Function . . . . .	63
5.2	Möbius Function . . . . .	63
5.3	Number Theory Inverse . . . . .	63
5.4	Chinese Remainder Theorem . . . . .	64
5.5	Linear congruences . . . . .	64
5.6	FFT . . . . .	65
5.7	NTT . . . . .	69
5.8	Fast Walsh—Hadamard transform . . . . .	71
5.9	Lucas . . . . .	72

5.10	Linear Programming . . . . .	73
5.11	Big Prime Test . . . . .	75
5.11.1	Miller Rabin . . . . .	77
5.11.2	Pollard's rho . . . . .	78
5.12	Montgomery modular multiplication . . . . .	79
5.13	Berlekamp Massey . . . . .	80
5.14	Inclusion—exclusion principle . . . . .	82
5.14.1	General form . . . . .	82
5.14.2	A generalization . . . . .	82
5.14.3	Applications . . . . .	83
5.15	Lindström—Gessel—Viennot lemma . . . . .	83
<b>6</b>	<b>String</b>	<b>83</b>
6.1	Hash . . . . .	83
6.2	KMP . . . . .	83
6.3	exKMP . . . . .	84
6.4	Suffix Array . . . . .	85
6.5	Aho-Corasick Automata . . . . .	88
6.6	Manacher . . . . .	89
<b>7</b>	<b>Utility</b>	<b>90</b>
7.1	IO plug-in . . . . .	90
7.2	Random Numbers . . . . .	94
<b>8</b>	<b>Appendix</b>	<b>95</b>
8.1	C++ Reference . . . . .	95
8.1.1	STL . . . . .	95
8.2	Java Reference . . . . .	95
8.2.1	Basic . . . . .	95
8.2.2	BigInteger . . . . .	95
8.2.3	BigDecimal . . . . .	97
8.2.4	Sorting . . . . .	97
8.3	Environment test items . . . . .	98

## I Datastructure

### I.I Fenwick

```

1  /* Fenwick Tree (Binary Indexed Tree), by Abreto <m@abreto.net>. */
2  #include <cstring>
3
4  using namespace std;
5
6  template <class T = int, int MAXN = 100001>
7  struct fenwick {
8      static inline int lowbit(int x) {
9          return (x&(-x));
10     }
11     int N;
12     T f[MAXN]; /* 1-based. */
13     fenwick(void):N(MAXN) {
14         init();
15     }
16     fenwick(int n):N(n) {
17         init();
18     }
19     void init(void) {
20         memset(f,0,sizeof(f));
21     }
22     void upd(int i, T dx) {
23         while(i <= N) {
24             f[i] += dx;
25             i += lowbit(i);
26         }
27     }
28     T sum(int i) {
29         T ret = 0;
30         while(i) {
31             ret += f[i];
32             i -= lowbit(i);
33         }
34         return ret;
35     }
36 };

```

```

1  /* Fenwick Tree (Binary Indexed Tree), by Abreto <m@abreto.net>. */
2
3  #define MAXN 100001
4  #define LOWBIT(x) ((x)&(-(x)))
5
6  int N;
7  int fen[MAXN];
8
9  void update(int i, int dx) {
10     while(i <= N) {
11         fen[i] += dx;
12         i += LOWBIT(i);
13     }
14 }
15
16 int sum(int i) {
17     int s = 0;
18     while(i > 0) {
19         s += fen[i];
20         i -= LOWBIT(i);
21     }
22     return s;
23 }

```

## 1.2 BST in pb\_ds

```

1  /* Red-Black tree via pb_ds. */
2  #include<bits/stdc++.h>
3  #include<ext/pb_ds/assoc_container.hpp>
4  #include<ext/pb_ds/tree_policy.hpp>
5  using namespace __gnu_pbds;
6  using namespace std;
7  template <typename T>
8  using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
9
10 int main() {
11     ordered_set<int> s;
12     s.insert(1);
13     s.insert(3);
14     cout << s.order_of_key(2) << endl; // the number of elements in the s less than 2
15     cout << *s.find_by_order(0) << endl; // print the 0-th smallest number in s(0-based)
16 }

```

## 1.3 Segment Tree

```

1  /* Segment tree (Interval tree, range tree), by Abreto <m@abreto.net>. */
2
3  template <int STMAX = 1000000>
4  struct segment_tree {
5      struct node_t {
6          static inline node_t merge(node_t n1, node_t n2) {
7              node_t ans;
8              ans.l = n1.l;
9              ans.r = n2.r;
10             /* merge n1 and n2 to ans. */
11             return ans;
12         }
13
14         /* Data field */
15         int l,r;
16     } nodes[(STMAX+1)<<2];
17
18     struct lazy_t {
19         int marked; /* Optional */
20         /* lazy mark. */
21
22         lazy_t(void) {
23             clear();
24         }
25         void clear(void) {
26             marked=0;
27         }
28     } marks[(STMAX+1)<<2];
29
30     inline void maintain_leaf(int o, int idx) {
31         nodes[o].l = nodes[o].r = idx;
32         /* Operations to single elements ... */
33     }
34     inline void maintain(int o) {
35         nodes[o] = node_t::merge(nodes[o<<1], nodes[o<<1|1]);
36     }
37
38     /* Usage: build(1,1,n); */
39     void build(int o, int l, int r) { /* [l,r] */

```

```

40     if( r <= l ) {
41         maintain_leaf(o, l);
42     } else {
43         int mid = l+r>>1;
44         build(o<<1, l, mid);
45         build(o<<1|1, mid+1, r);
46         maintain(o);
47     }
48 }
49
50 /* Modify all elements in [l,r] */
51 void mark(lazy_t act, int o) {
52     /* do something .. */
53     marks[o].marked = 1;
54 }
55
56 /* Pass cached updates. */
57 void pushdown(int o) {
58     if( marks[o].marked ) {
59         mark(marks[o], o<<1);
60         mark(marks[o], o<<1|1);
61         marks[o].clear();
62     }
63 }
64
65 /* Do act on all elements in [L,R] */
66 void upd(int L, int R, lazy_t act, int o, int l, int r) {
67     if( L <= l && r <= R ) {
68         mark(act, o);
69     } else if (L <= R) {
70         int mid = (l+r)>>1;
71         pushdown(o);
72         if( L <= mid ) upd(L, R, act, o<<1, l, mid);
73         if( R > mid ) upd(L, R, act, o<<1|1, mid+1, r);
74         maintain(o);
75     }
76 }
77
78 node_t qry(int L, int R, int o, int l, int r) {
79     if(L <= l && r <= R)
80         return nodes[o];
81     else if (L <= R) {
82         int mid = (l+r)>>1;
83         pushdown(o);
84         if(R <= mid) return qry(L,R,o<<1,l,mid);
85         if(L > mid) return qry(L,R,o<<1|1,mid+1,r);
86         return node_t::merge(qry(L,R,o<<1,l,mid),qry(L,R,o<<1|1,mid+1,r));
87     }
88 }
89
90 int N;
91
92 segment_tree(void):N(STMAX) {}
93 segment_tree(int n):N(n) {}
94 void build(int n) {
95     N = n;
96     build(1,1,N);
97 }
98 void update(int L, int R, lazy_t act) {
99     upd(L,R,act,1,1,N);
100 }
101 node_t query(int L, int R) {
102     return qry(L,R,1,1,N);
103 }

```

```

104 |};

1  /* Segment tree (Interval tree, range tree), by Abreto <m@abreto.net>. */
2
3  #define MAXN    1000001
4
5  typedef struct {
6      int l,r;
7      /* Data field */
8  } node_t;
9
10 node_t merge(node_t n1, node_t n2) {
11     node_t ans;
12     ans.l = n1.l;
13     ans.r = n2.r;
14     /* merge n1 and n2 to ans. */
15     return ans;
16 }
17
18 typedef struct {
19     int marked; /* Optional */
20     /* lazy mark. */
21 } lazy_t;
22
23 int A[MAXN];
24 node_t nodes[MAXN<<2];
25 lazy_t marks[MAXN<<2];
26
27 void maintain_leaf(int o, int idx) {
28     nodes[o].l = nodes[o].r = idx;
29     /* Operations to single elements ... */
30 }
31 void maintain(int o) {
32     nodes[o] = merge(nodes[o<<1], nodes[o<<1|1]);
33 }
34
35 /* Usage: build(1,1,n); */
36 void build(int o, int l, int r) { /* [l,r] */
37     if( r <= l ) {
38         maintain_leaf(o, l);
39     } else {
40         int mid = l+r>>1;
41         build(o<<1, l, mid);
42         build(o<<1|1, mid+1, r);
43         maintain(o);
44     }
45     marks[o].marked = 0;
46 }
47
48 /* Modify all elements in [l,r] */
49 void mark(lazy_t act, int o) {
50     /* do something .. */
51     marks[o].marked = 1;
52 }
53
54 /* Pass cached updates. */
55 void pushdown(int o) {
56     if( marks[o].marked ) {
57         mark(marks[o], o<<1);
58         mark(marks[o], o<<1|1);
59         marks[o].marked = 0;
60     }
61 }
62

```

```

63 /* **DISCARDED** */
64 /* Set A[p]=v. Usage: modify(p, v, 1, 1, n);
65 void modify(int p, int v, int o, int l, int r)
66 {
67     if( r - l < 2 )
68     {
69         maintain_leaf(o, v);
70     } else {
71         int mid = (l+r)/2;
72         pushdown(o);
73         if( p <= mid ) modify(p, v, o*2, l, mid);
74         else modify(p, v, o*2+1, mid, r);
75         maintain(o);
76     }
77 }*/
78
79 /* Do act on all elements in [L,R] */
80 void update(int L, int R, lazy_t act, int o, int l, int r) {
81     if( L <= l && r <= R ) {
82         mark(act, o);
83     } else if (L <= R) {
84         int mid = (l+r)>>1;
85         pushdown(o);
86         if( L <= mid ) update(L, R, act, o<<1, l, mid);
87         if( R > mid ) update(L, R, act, o<<1|1, mid+1, r);
88         maintain(o);
89     }
90 }

```

#### 1.4 Sparse Table

```

1 /* RMQ with Sparse Table, by Abreto <m@abreto.net>. */
2
3 int min(int a, int b) {
4     return (a<b)?a:b;
5 }
6
7 #define MAXN    100001
8 #define MAXLOG  32
9
10 int N;
11 int A[MAXN];    /* indexed from 0. */
12 int st[MAXN][MAXLOG];
13
14 void st_init() {
15     int i = 0, j = 0, t = 0;
16     for(i = 0; i < N; ++i) st[i][0] = A[i];
17     for(j = 1; (t=(1<<j)) <= N; ++j)
18         for(i = 0; (i+t-1) < N; ++i)
19             st[i][j] = min(st[i][j-1], st[i+(t>>1)][j-1]);
20     /* st(i,j) = min(st(i,j-1), st(i+2^(j-1),j-1)). */
21 }
22
23 int st_query(int l, int r) {
24     int k = 0;
25     while((1<<(k+1)) <= (r-l+1)) k++;
26     return min(st[l][k], st[r-(1<<k)+1][k]);
27 }

```

#### 1.5 Treap



```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define MAXN (2560000)
6
7 int __treap_mem[MAXN];
8 void init_treap_mem(void) {
9     for( int i = 1 ; i < MAXN ; i++ )
10         __treap_mem[i-1] = i;
11 }
12 int alloc_address(void) {
13     int ret = __treap_mem[0];
14     __treap_mem[0] = __treap_mem[ret];
15     return ret;
16 }
17 void free_address(int p) {
18     __treap_mem[p] = __treap_mem[0];
19     __treap_mem[0] = p;
20 }
21
22 typedef int key_t;
23 typedef int val_t;
24 struct treap {
25     key_t x;
26     val_t v;
27     int r; /* random priority */
28     int eq, s; /* number of equal ones, size of subtree (include root itself) */
29     treap *fa; /* point to its father */
30     treap *ch[2]; /* 0 for left child, 1 for right child. */
31
32     treap(void);
33     inline void maintain(void); /* update s */
34     inline void set_child(int d, treap *child);
35     inline int which(void); /* determine which child this is of its father */
36     inline int cmp(key_t ox); /* determine which child to insert ox */
37     treap *rotate(void); /* rotate this to its father, return this */
38 } treap_nodes[MAXN];
39
40 treap *new_treap(key_t x, val_t v, treap *f) {
41     treap *ret = treap_nodes + alloc_address();
42     ret->x = x;
43     ret->v = v;
44     ret->eq = ret->s = 1;
45     ret->fa = f;
46     ret->ch[0] = ret->ch[1] = NULL;
47 }
48 void free_treap(treap *p) {
49     free_address( p - treap_nodes );
50 }
51 void treap_clear(void) {
52     init_treap_mem();
53 }
54
55 treap::treap(void) {
56     r = rand();
57     eq = s = 0;
58     fa = ch[0] = ch[1] = NULL;
59 }
60 inline void treap::maintain(void) {
61     s = eq;
62     for( int i = 0 ; i < 2 ; i++ )
63         if( ch[i] )
64             s += ch[i]->s;

```

```

65 }
66 inline void treap::set_child(int d, treap *child) {
67     ch[d] = child;
68     maintain();
69     if( child ) child->fa = this;
70 }
71 inline int treap::which(void) {
72     if( NULL == fa ) return -1; /* this is not a child */
73     else return ( fa->ch[1] == this );
74 }
75 inline int treap::cmp(key_t ox) {
76     if( ox == x ) return -1; /* equal */
77     else return ( ox > x ); /* left less, right more */
78 }
79 treap *treap::rotate(void) {
80     if ( NULL == fa ) return this; /* no father, already global root. */
81     int d = which();
82     fa->set_child(d, ch[d^1]);
83     set_child(d^1, fa);
84     return this;
85 }
86
87 // — deprecated |
88 typedef int T;
89
90 struct node {
91     T v; /* value of this node */
92     int r; /* random priority */
93     int eq; /* the number of equal things */
94     int s; /* the size of subtree rooted at this */
95     node *ch[2]; /* 0 for left child, 1 for right child. */
96     node(void) {
97         r = rand();
98         ch[0] = ch[1] = NULL;
99     }
100     /* return where to insert x */
101     int cmp(T x) {
102         if(v == x) return -1;
103         else return (x < v) ? 0 : 1;
104     }
105     /* return 1 if this node is prior to other */
106     int pri(node *o) {
107         return (r > (o->r));
108     }
109     /* maintain the s field */
110     void maintain(void) {
111         s = eq;
112         if(NULL != ch[0]) s += ch[0]->s;
113         if(NULL != ch[1]) s += ch[1]->s;
114     }
115 };
116
117 /* move o to ch[d] of o->ch[d^1] */
118 void rotate(node *&o, int d) {
119     node *k = o->ch[d^1];
120     o->ch[d^1] = k->ch[d];
121     o->maintain();
122     k->ch[d] = o;
123     k->maintain();
124     o = k;
125 }

```

## 1.6 Leftist Heap

```

1  /* HDU 1512 Monkey King (左偏树模板题) */
2  #include<iostream>
3  #include<cstdio>
4  using namespace std;
5  const int maxn = 100000+5;
6
7  int n, m;
8
9  struct Heap {
10     int l,r,fa,val,dis;
11 } t[maxn];
12
13
14 int finds(int x) {
15     return t[x].fa == -1? x:t[x].fa = finds(t[x].fa);
16 }
17
18 int merge(int x, int y) {
19     if(x == 0) return y;    //如果为0的话，就说明是空子树，根节点当然就是另一节点了
20     if(y == 0) return x;
21     if(t[y].val>t[x].val) swap(x,y);    //始终往右子树进行插入
22     t[x].r = merge(t[x].r,y);
23     t[t[x].r].fa = x;
24     if(t[t[x].l].dis < t[t[x].r].dis) swap(t[x].l,t[x].r);    //是否需要左右子树的对换，
        这样是为了右子树尽量短
25     if(t[x].r == 0) t[x].dis = 0;    //距离的重新分配
26     else t[x].dis = t[t[x].r].dis + 1;
27     return x;
28 }
29
30 int pop(int &root) {
31     int l = t[root].l;
32     int r = t[root].r;
33     t[root].l = t[root].r = t[root].dis = 0;
34     t[root].fa = -1;
35     t[l].fa = t[r].fa = -1;    //删除root根节点
36     return merge(l,r);    //这样一来相当于分裂成了两棵子树，重新进行合并，最后返回值
        为合并后的根节点
37 }
38
39 int push(int x, int y) {
40     return merge(x,y);
41 }
42
43 int main() {
44     //freopen("in.txt","r",stdin);
45     while(~scanf("%d",&n)) {
46         for(int i=1; i<=n; i++) {
47             t[i].l=t[i].r=t[i].dis=0;
48             t[i].fa=-1;
49             scanf("%d",&t[i].val);
50         }
51         scanf("%d",&m);
52         while(m--) {
53             int a,b;
54             scanf("%d%d",&a,&b);
55             int x=finds(a);
56             int y=finds(b);
57             if(x!=y) {
58                 t[x].val/=2;
59                 int xx = push( pop(x),x);

```

```

60         t[y].val/=2;
61         int yy = push( pop(y),y);
62         printf("%d\n",t[merge(xx,yy)].val);
63     } else puts("-1");
64 }
65 }
66 return 0;
67 }

```

## 1.7 Splay

```

1  /* splay, by Abreto<m@abreto.net>. */
2
3  #ifndef NULL
4  #define NULL 0
5  #endif
6
7  struct node {
8      node *f, *ch[2];
9      int sz;
10     node(node *fa = NULL, node *lc = NULL, node *rc = NULL) {
11         f = fa;
12         ch[0] = lc;
13         ch[1] = rc;
14         maintain();
15     }
16     inline int szof(const int d) const {
17         return ch[d] ? ch[d]->sz : 0;
18     }
19     inline void maintain(void) {
20         sz = szof(0) + szof(1) + 1;
21     }
22     inline int which(void) {
23         if (NULL == f) return 0;
24         return (f->ch[1] == this); /* f[which()] == this */
25     }
26     inline node *setf(node *fa, int d = 0) {
27         f = fa;
28         if (f) {
29             f->ch[d] = this;
30             f->maintain();
31         }
32         return f;
33     }
34     inline node *setc(node *son, int d = 0) {
35         ch[d] = son;
36         if (son) son->f = this;
37         maintain();
38         return this;
39     }
40     /* rotate this to its fater, return this. */
41     inline node *rotate(void) {
42         if (f != NULL) {
43             node *ff = f->f;
44             int d = which(), fd = f->which();
45             setc(f->setc(ch[d ^ 1], d), d ^ 1);
46             setf(ff, fd);
47         }
48         return this;
49     }
50     /* splay this to child of target */
51     inline node *splay(node * const target = NULL) {

```

```

52     while (f != target) {
53         if (target != f->f) {
54             ( (which() == f->which()) ? f : this )->rotate();
55         }
56         rotate();
57     }
58     return this;
59 }
60 /* 0-based rank */
61 inline node *get_k_th(unsigned k) {
62     node *p = this;
63     int rank;
64     while (k != (rank = (p->szof(0)))) {
65         if (k < rank) {
66             p = p->ch[0];
67         } else {
68             k -= (rank + 1);
69             p = p->ch[1];
70         }
71     }
72     return p->splay(f);
73 }
74 };

```

```

1  /* HDU 3487 – Play with Chain, by Abreto<m@abreto.net>. */
2  #include <bits/stdc++.h>
3
4  using namespace std;
5
6  #define MAXN    300300
7
8  int n, m;
9
10 #define LC(p)    ch[p][0]
11 #define RC(p)    ch[p][1]
12 #define TARGET(p) LC(RC(p))
13
14 int nodes;
15 int val[MAXN], ch[MAXN][2], fa[MAXN], sz[MAXN];
16 int rev[MAXN];
17
18 inline int new_node(int v, int f) {
19     int p = (++nodes);
20     val[p] = v;
21     fa[p] = f;
22     ch[p][0] = ch[p][1] = rev[p] = 0;
23     sz[p] = 1;
24     return p;
25 }
26 inline void maintain(int p) {
27     if (p) {
28         sz[p] = sz[LC(p)] + sz[RC(p)] + 1;
29     }
30 }
31 inline void make_child(int f, int d, int p) { /* make p the d-th ch of f */
32     ch[f][d] = p;
33     if (p) fa[p] = f;
34 }
35 inline void myrev(int p) {
36     if (p) {
37         rev[p] ^= 1;
38         swap(LC(p), RC(p));
39     }
40 }

```

```

41 inline void pushdown(int p) {
42     if(p && rev[p]) {
43         if(LC(p)) myrev(LC(p));
44         if(RC(p)) myrev(RC(p));
45         rev[p] = 0;
46     }
47 }
48 int build(int f = 0, int l = 0, int r = n+1) {
49     if(r < l) return 0;
50     if(l == r) return new_node(l, f);
51     int mid = l+r>>1;
52     int p = new_node(mid, f);
53     LC(p) = build(p, l, mid-1);
54     RC(p) = build(p, mid+1, r);
55     maintain(p);
56     return p;
57 }
58 inline int which(int p) { /* return 1 if p is a right child or 0 if p is a left
59     child. */
60     return (RC(fa[p]) == p);
61 }
62 inline int rotate(int p) { /* rotate p to its father. [!] make sure p is not global
63     root. */
64     int f = fa[p], ff = fa[f];
65     if(0 == f) return p; /* p is global root */
66     pushdown(f);
67     pushdown(p);
68     int d = which(p), df = which(f);
69     make_child(f, d, ch[p][d^1]);
70     make_child(p, d^1, f);
71     maintain(f);
72     maintain(p);
73     fa[p] = ff;
74     if(ff) ch[ff][df] = p;
75     return p;
76 }
77 inline int splay(int p, int fr) { /* splay p to the son of fr, return p. */
78     pushdown(p);
79     while(fa[p] != fr) {
80         int f = fa[p], dp = which(p);
81         if(fa[f] == fr) {
82             return rotate(p);
83         } else {
84             int df = which(f);
85             if(dp == df) {
86                 rotate(f);
87             } else {
88                 rotate(p);
89             }
90         }
91     }
92     return p;
93 }
94 inline int get_k_th(int root, int k) {
95     int p = root;
96     int rank;
97     while(k != (rank = (sz[LC(p)] + 1))) {
98         pushdown(p);
99         if(k < rank) p = LC(p);
100         else {
101             k -= rank;
102             p = RC(p);
103         }
104     }

```

```

103     }
104     return splay(p, fa[root]);
105 }
106 inline int merge(int left, int right) {
107     pushdown(left);
108     if(RC(left)) left = get_k_th(left, sz[left]);
109     RC(left) = right;
110     maintain(left);
111     fa[right] = left;
112     return left;
113 }
114 inline int split(int root, int d) { /* split ch[root][d], return the root of splitted
    out. */
115     pushdown(root);
116     int child = ch[root][d];
117     ch[root][d] = 0;
118     maintain(root);
119     fa[child] = 0;
120     return child;
121 }
122 inline int concat(int root, int d, int p) { /* make p be ch[root][d], return root */
123     pushdown(root);
124     ch[root][d] = p;
125     fa[p] = root;
126     maintain(root);
127     return root;
128 }
129
130 void myclear(void) {
131     nodes = 0;
132 }
133
134 int ans[MAXN];
135 void inorder(int p, int &pos) {
136     if(0 == p) return;
137     pushdown(p);
138     inorder(LC(p), pos);
139     if( (0 < val[p]) && (val[p] < n+1) ) ans[pos++] = val[p];
140     inorder(RC(p), pos);
141 }
142
143 void handle() {
144     int i;
145     int root;
146     myclear();
147     root = build(0);
148     while(m--) {
149         char command[8];
150         int a, b, c;
151         int tar;
152         scanf("%s%d%d", command, &a, &b);
153         if('C' == command[0]) {
154             scanf("%d", &c);
155             root = get_k_th(root, a);
156             RC(root) = get_k_th(RC(root), b-a+2);
157             tar = split(RC(root), 0);
158             maintain(root);
159             root = get_k_th(root, c+1);
160             RC(root) = get_k_th(RC(root), 1);
161             RC(root) = concat(RC(root), 0, tar);
162             maintain(root);
163         } else {
164             root = get_k_th(root, a);
165             RC(root) = get_k_th(RC(root), b-a+2);

```

```

166     myrev(TARGET(root));
167 }
168 }
169 int pos = 0;
170 inorder(root, pos);
171 for(i = 0; i < n; i++) printf("%s%d", i ? " ": "", ans[i]);
172 puts("");
173 }
174
175 int main(void) {
176     while( scanf("%d%d", &n, &m) && (n > 0) && (m > 0) )
177         handle();
178     return 0;
179 }

```

## 1.8 Persistent Segment Tree

1. 首先, 给你一颗值为横坐标的线段树, 每个节点上存着该值出现了多少次, 这样的一颗线段树你会求区间  $k$  大值吧. 二分即可.
2. 然后, 假设区间是数组  $arr[n]$ , 区间长度是  $n$ , 那么给你  $n$  颗线段树, 第  $i$  颗线段树是第  $i - 1$  颗线段树插入  $arr[i]$  得到.
3. 如果你有了这  $n$  颗线段树, 想求区间  $[l, r]$  中的第  $k$  大值, 那么你需要在第  $r$  颗和第  $l - 1$  颗线段树的差线段树上作二分, 就可以求得区间第  $k$  大值.
4. 差线段树很好理解, 比如你有一个部分和数组  $sum$ ,  $sum[r] - sum[l - 1]$  就是部分和的差, 代表区间  $[l, r]$  的和, 差线段树同理.
5. 现在, 可持久化线段树出现为你解决最后一个问题, 空间问题. 内存很小, 不能够存下  $n$  颗线段树. 但是, 第 2 条中提到, 由于第  $i$  颗线段是第  $i - 1$  颗线段是插入仅一个值得到的, 两颗线段树的区别不大, 仅有  $\log(n)$  个节点发生了改变, 我们仅仅需要记录这  $\log(n)$  的数据就可以记录这个增量, 这就是可持久化线段树.

## 2 Dynamic Programming

### 2.1 LIS $O(n \log n)$

```

1
2 int top = 0;
3 for( int i=1; i<=n; i++ ) {
4     if( ap[i] > dp[top] ) { // 如果大于 "模拟栈" 的栈顶元素直接 入栈 长度加 1
5         top++;
6         dp[top] = ap[i];
7         continue;
8     }
9     int m = ap[i];
10    // lower_bound 前闭后开 返回不小于 m 的最小值的位置
11    pos = lower_bound(dp, dp+top, m)-dp; // 注意减去 dp
12    if( dp[pos] > ap[i] )
13        dp[pos] = ap[i];
14 }

```

### 2.2 LCS $O(n \log n)$

总的来说, 就是把 LCS 转化成 LIS, 然后用 LIS 的  $O(N \log N)$  算法来求解。

实现如下: (引用)

假设有两个序列  $s_1[1 \dots 6] = abcadac$ ,  $s_2[1 \dots 7] = cabedab$ .



记录  $s_1$  中每个元素在  $s_2$  中出现的位置, 再将位置按降序排列, 则上面的例子可表示为:  
 $loc(a) = \{6, 2\}, loc(b) = \{7, 3\}, loc(c) = \{1\}, loc(d) = \{5\}$ . (倒着扫一遍  $s_2$  即可把位置扔进 vector).  
 将  $s_1$  中每个元素的位置按  $s_1$  中元素的顺序排列成一个序列  $s_3 = \{6, 2, 7, 3, 1, 6, 2, 5, 1\}$ .  
 在对  $s_3$  求 LIS 得到的值即为求 LCS 的答案。

### 2.3 Improved by quadrilateral inequality

```

1  /*
2  * 四边形不等式
3  *
4  * 如果 dp(i,j) 满足 dp(i,j) <= dp(i,j+1) <= dp(i+1,j+1)
5  * 那么决策 s(i,j) 满足 s(i,j) <= s(i,j+1) <= s(i+1,j+1)
6  * 可以变形为:
7  *      s(i-1,j) <= s(i,j) <= s(i,j+1)  // dp方向: i增j减
8  * 或
9  *      s(i,j-1) <= s(i,j) <= s(i+1,j)  // dp方向: 区间长度L增
10 */
11 #include <bits/stdc++.h>
12
13 using namespace std;
14
15 #define MAXN    1024
16 #define inf     (0x3fffffff)
17
18 int n, m;
19 int v[MAXN];
20 int s[MAXN];
21 int w[MAXN][MAXN];
22 int dp[MAXN][MAXN];
23 int c[MAXN][MAXN];
24
25 int wa(void) {
26     int i, j, k;
27     for(i = 1; i <= n; ++i) {
28         scanf("%d", v+i);
29         s[i] = v[i] + s[i-1];
30     }
31     for(i = 1; i <= n; ++i) {
32         w[i][i] = 0;
33         for(j = i+1; j <= n; ++j)
34             w[i][j] = w[i][j-1] + v[j] * (s[j-1] - s[i-1]);
35     }
36     /* doing dp */
37     for(i = 1; i <= n; ++i) {
38         dp[i][0] = w[1][i];
39         c[i][0] = 1;
40         c[i][i] = i-1;
41         for(j = i-1; j > 0; j--) {
42             dp[i][j] = inf;
43             for(k = c[i-1][j]; k <= c[i][j+1]; ++k)
44                 if(dp[k][j-1] + w[k+1][i] <= dp[i][j]) {
45                     dp[i][j] = dp[k][j-1] + w[k+1][i];
46                     c[i][j] = k;
47                 }
48         }
49     }
50     /* dp done */
51     return dp[n][m];
52 }
53
54 int main(void) {
55     while(EOF != scanf("%d%d", &n, &m) && n && m) {

```

```

56     printf("%d\n", wa());
57 }
58 return 0;
59 }

```

## 2.4 Improved by Slope

```

1  /* type 1: */
2  /* bzoj 1010 */
3  #include <bits/stdc++.h>
4
5  using namespace std;
6  typedef long double ll;
7  #define MAXN    50050
8  #define eps     (1e-8)
9
10 int N;
11 ll L;
12 ll S[MAXN];
13 ll f[MAXN];
14 ll dp[MAXN];
15
16 inline ll k(int j) {
17     return (-2.0) * (f[j] + L);
18 }
19 inline ll b(int j) {
20     return dp[j] + f[j]*f[j] + 2ll*f[j]*L;
21 }
22 inline ll g(int j, int i) {
23     return k(j) * f[i] + b(j);
24 }
25
26 /* check if l1 & l3 <= l2 */
27 inline int check(int l1, int l2, int l3) {
28     /*ll left = b(l3)*k(l1)+b(l1)*k(l2)+b(l2)*k(l3);
29     ll right = b(l1)*k(l3)+b(l3)*k(l2)+b(l2)*k(l1);*/
30     ll left = b(l3)*k(l1)-b(l1)*k(l3);
31     ll right = k(l2)*(b(l3)-b(l1))+b(l2)*(k(l1)-k(l3));
32     return (left <= right);
33 }
34
35 int Q[MAXN], ql, qr;
36
37 int main(void) {
38     int i;
39     scanf("%d%Lf", &N, &L);
40     L += 1.0;
41     for(i = 1; i <= N; ++i) {
42         scanf("%Lf", S+i);
43         S[i] += S[i-1];
44         f[i] = S[i] + (double)i;
45     }
46     Q[qr++] = 0;
47     for(i = 1; i <= N; ++i) {
48         /* <!-- STARED */
49         for(; ql+1 < qr && g(Q[ql],i) >= g(Q[ql+1],i); ql++);
50         dp[i] = g(Q[ql], i) + f[i]*f[i] + L*L; //printf("%d: %lld,%lld\n", i, dp[i], dp[i]
           -f[i]*f[i]);
51         for(; ql+1 < qr && check(Q[qr-2], Q[qr-1], i); qr--);
52         Q[qr++] = i;
53         /* --> */
54     }

```

```

55 | printf("%lld\n", (long long int)round(dp[N]));
56 | return 0;
57 | }

```

## 2.5 Steiner Tree

令  $f[i][sta]$  表示  $i$  号节点，与其他节点的连通性为  $sta$  时的最小代价，这里  $sta$  是一个二进制数，在它二进制下的每一位中，0 表示不连通，1 表示联通  
状态转移：

- 由子集转移而来

$$f[i][sta] = \min_{s \subseteq sta} \{f[i][s] + f[i][sta \setminus s] - val[i]\}$$

- 由不含该节点的状态转移而来

$$f[i][j] = \min \{f[k][j] + val[i]\}$$

流程：

```

1 | 枚举状态集S
2 | {
3 |     枚举S的子集s
4 |     {
5 |         更新f[S][1~n]
6 |     }
7 |     将 f[S][x]<inf 的x入队
8 |     spfa(S)
9 | }

```

代码：

```

1 | int f[1<<M][N];
2 | queue<int> q;
3 | bool in[N];
4 |
5 | void spfa(int S) {
6 |     while (!q.empty()) {
7 |         int now=q.front();
8 |         q.pop();
9 |         in[now]=0;
10 |         for (int i=st[now]; i; i=way[i].nxt) {
11 |             int y=way[i].y;
12 |             if (f[S][y]>f[S][now]+val[y]) {
13 |                 f[S][y]=f[S][now]+val[y];
14 |                 if (!in[y]) q.push(y),in[y]=1;
15 |             }
16 |         }
17 |     }
18 | }
19 |
20 | void work() {
21 |     int cnt=0;
22 |     memset(f,0x7f,sizeof(f));
23 |
24 |     for (int i=1; i<=n; i++)
25 |         if (!val[i]) f[1<<cnt][i]=0,cnt++;
26 |     for (int S=1; S<(1<<cnt); S++) {
27 |         for (s=(S-1)&S; s; s=(s-1)&S)
28 |             for (int i=1; i<=n; i++)
29 |                 f[S][i]=min(f[S][i],f[s][i]+f[S^s][i]-val[i]);
30 |         for (int i=1; i<=n; i++)
31 |             if (f[S][i]<INF&&!in[i])

```

```

32     q.push(i),in[i]=1;
33     spfa(S);
34 }
35
36 int ans=INF;
37 for (int i=1; i<=n; i++) ans=min(ans,f[(1<<cnt)-1][i]);
38 printf("%d\n",ans);
39 }
40 // -----
41 // 作者: Coco_T_
42 // 来源: CSDN
43 // 原文: https://blog.csdn.net/wu\_tongtong/article/details/78992913
44 // 版权声明: 本文为博主原创文章, 转载请附上博文链接!

```

### 3 Geometry

#### 3.1 2D

##### 3.1.1 Point

```

1  /* 2D Point Class, by Abreto<m@abreto.net> */
2  #include <cmath>
3
4  /**
5   * Define ABG2d_USE_LL if you want to use long long int for cordnates.
6   */
7
8  namespace ab_geometry_2d {
9
10 using namespace std;
11
12 typedef double ab_float;
13
14 const ab_float pi = acos(-1.);
15
16 #ifdef ABG2d_USE_LL
17 typedef long long int T;
18 #else
19 typedef ab_float T;
20 const ab_float eps = 1e-8;
21 #endif
22
23 inline T myabs(T x) {
24     if(x < 0) return (-x);
25     return x;
26 }
27
28 inline int sgn(T x) {
29     /* no difference'' in fact */
30 #ifdef ABG2d_USE_LL
31     if (0 == x) return 0;
32 #else
33     if (myabs(x) < eps) return 0;
34 #endif
35     return (x > 0) ? 1 : -1;
36 }
37
38 inline T sqr(T x) {
39     return (x * x);
40 }
41
42 struct point {

```

```

43 | T x,y;
44 | point(void):x(T()),y(T()) {}
45 | point(T xx, T yy):x(xx),y(yy) {}
46 | inline T norm2(void) {
47 |     return sqr(x) + sqr(y);
48 | }
49 | inline ab_float norm(void) {
50 |     return sqrt((ab_float)(norm2()));
51 | }
52 | inline point rotate(const ab_float &cost, const ab_float &sint) {} // TODO:
53 | inline point operator-(void) const {
54 |     return point(-x,-y);
55 | }
56 | inline point operator+(const point& b) const {
57 |     return point(x+b.x,y+b.y);
58 | }
59 | inline point operator-(const point& b) const {
60 |     return point(x-b.x,y-b.y);
61 | }
62 | inline point operator->*(const point &b) const {
63 |     return (b-(*this));
64 | }
65 | inline T operator*(const point& b) const {
66 |     return ((x)*(b.x))+((y)*(b.y)); /* inner product */
67 | }
68 | inline T operator^(const point& b) const {
69 |     return ((x)*(b.y))-((b.x)*(y)); /* outter product */
70 | }
71 | inline point& operator+=(const point& b) {
72 |     point tmp=(*this)+b;
73 |     (*this)=tmp;
74 |     return (*this);
75 | }
76 | inline point& operator-=(const point& b) {
77 |     point tmp=(*this)-b;
78 |     (*this)=tmp;
79 |     return (*this);
80 | }
81 | inline bool operator==(const point& b) const {
82 |     return (0==sgn(x-b.x))&&(0==sgn(y-b.y));
83 | }
84 | inline bool operator!=(const point& b) const {
85 |     return !((*this)==b);
86 | }
87 | inline point operator<<(const ab_float& theta) const {
88 |     ab_float ct = cos(theta), st = sin(theta); /* rotate counter-clockwise in radian
89 |         */
90 |     return point(ct*x - st*y, st*x + ct*y);
91 | };
92 |
93 | typedef point vec;
94 |
95 |
96 | } // namespace ab_geometry_2d

```

### 3.1.2 Circle

#### Base

```

1 | /* 2D Circle Base Class, by Abreto<m@abreto.net>. */
2 |
3 | /* requirement: point.cc */

```

```

4 #include "point.cc"
5
6 #include <utility>
7
8 namespace ab_geometry_2d {
9
10 using namespace std;
11
12 struct circle {
13     point o;
14     T r;
15     circle(void) : r(T()) {}
16     circle(point center, T radius) : o(center), r(radius) {}
17
18     inline ab_float arclen(ab_float theta) {
19         return theta * r;
20     }
21     inline ab_float circumference(void) {
22         return 2. * pi * r;
23     }
24     inline ab_float area(void) {
25         return pi * r * r;
26     }
27
28     /* bool contain(const circle &C, const bool including_touch = false) const
29     {
30         T dis2 = (o->*(C.o)).norm2();
31         T raw_diff = r - C.r;
32         if ( -1 == sgn(raw_diff) ) return false;
33         T dr2 = sqr(raw_diff);
34         return (dis2 < dr2) || (including_touch && (dis2 == dr2));
35     }
36     inline bool in(const circle &C, const bool including_touch = false) const
37     {
38         return C.contain(*this, including_touch);
39     } */
40     enum relation_t {
41         same = 0x000000,
42         contain = 0x000001,
43         intouch = 0x000010,
44         intersect = 0x000100,
45         outtouch = 0x001000,
46         separate = 0x010000,
47         unknow_relation = 0xffff
48     };
49     relation_t with(const circle &C) const {
50         T dis2 = (o->*(C.o)).norm2();
51         T dr2 = sqr(r - C.r), rs2 = sqr(r + C.r);
52         if ( 0 == sgn(dis2) && 0 == sgn(dr2) ) return same;
53         if ( -1 == sgn(dis2 - dr2) ) return contain;
54         if ( 0 == sgn(dis2 - dr2) ) return intouch;
55         if ( -1 == sgn(dr2 - dis2) && -1 == sgn(dis2 - rs2) ) return intersect;
56         if ( 0 == sgn(dis2 - rs2) ) return outtouch;
57         if ( -1 == sgn(rs2 - dis2) ) return separate;
58         return unknow_relation;
59     }
60
61     enum point_relation_t {
62         in = 0x001,
63         on = 0x010,
64         out = 0x100,
65         unknow_point_relation = 0xffff
66     };
67     point_relation_t with(const point &P) const {

```

```

68     T dis2 = (o->*P).norm2();
69     T r2 = sqr(r);
70     int type = sgn(dis2 - r2);
71     if (-1 == type) return in;
72     if ( 0 == type) return on;
73     if (+1 == type) return out;
74     return unknow_point_relation;
75 }
76
77 ab_float central_angle(const point &A, const point &B, const bool reflex = false)
78     const {
79     T dot = (A * B);
80     if (0 == sgn(dot)) return 1. * (A != B) * pi;
81     ab_float angle = ((ab_float)(dot)) / r / r;
82     if ( reflex ) angle = 2. * pi - angle;
83     return angle;
84 }
85 /* be sure (*this) intersect with C */
86 pair<point,point> crosspoint(const circle &C) const {
87     ab_float d = (o ->* (C.o)).norm();
88     // TODO:
89 }
90 };
91
92 }

```

### k 次圆交

```

1 //china no.1
2 #pragma comment(linker, "/STACK:1024000000,1024000000")
3 #include <vector>
4 #include <iostream>
5 #include <string>
6 #include <map>
7 #include <stack>
8 #include <cstring>
9 #include <queue>
10 #include <list>
11 #include <stdio.h>
12 #include <set>
13 #include <algorithm>
14 #include <cstdlib>
15 #include <cmath>
16 #include <iomanip>
17 #include <cctype>
18 #include <sstream>
19 #include <functional>
20 #include <stdlib.h>
21 #include <time.h>
22 #include <bitset>
23 using namespace std;
24
25 #define pi acos(-1)
26 #define PI acos(-1)
27 #define endl '\n'
28 #define srand() srand(time(0));
29 #define me(x,y) memset(x,y,sizeof(x));
30 #define foreach(it,a) for(__typeof((a).begin()) it=(a).begin();it!=(a).end();it++)
31 #define close() ios::sync_with_stdio(0); cin.tie(0);
32 #define FOR(x,n,i) for(int i=x;i<=n;i++)
33 #define FOr(x,n,i) for(int i=x;i<n;i++)
34 #define W while
35 #define sgn(x) ((x) < 0 ? -1 : (x) > 0)
36 #define bug printf("*****\n");

```

```

37 #define db double
38 typedef long long LL;
39 const int INF=0x3f3f3f3f;
40 const LL LINF=0x3f3f3f3f3f3f3f3fLL;
41 const int dx[]={-1,0,1,0,1,-1,-1,1};
42 const int dy[]={0,1,0,-1,-1,1,-1,1};
43 const int maxn=1e3+10;
44 const int maxx=1e6+100;
45 const double EPS=1e-8;
46 const double eps=1e-8;
47 const int mod=100000007;
48 template<class T>inline T min(T a,T b,T c) {
49     return min(min(a,b),c);
50 }
51 template<class T>inline T max(T a,T b,T c) {
52     return max(max(a,b),c);
53 }
54 template<class T>inline T min(T a,T b,T c,T d) {
55     return min(min(a,b),min(c,d));
56 }
57 template<class T>inline T max(T a,T b,T c,T d) {
58     return max(max(a,b),max(c,d));
59 }
60 template <class T>
61 inline bool scan_d(T &ret) {
62     char c;
63     int sgn;
64     if (c = getchar(), c == EOF) {
65         return 0;
66     }
67     while (c != '-' && (c < '0' || c > '9')) {
68         c = getchar();
69     }
70     sgn = (c == '-') ? -1 : 1;
71     ret = (c == '-') ? 0 : (c - '0');
72     while (c = getchar(), c >= '0' && c <= '9') {
73         ret = ret * 10 + (c - '0');
74     }
75     ret *= sgn;
76     return 1;
77 }
78
79 inline bool scan_lf(double &num) {
80     char in;
81     double Dec=0.1;
82     bool IsN=false,IsD=false;
83     in=getchar();
84     if(in==EOF) return false;
85     while(in!='-'&&in!='.'&&(in<'0' || in>'9'))in=getchar();
86     if(in=='-') {
87         IsN=true;
88         num=0;
89     } else if(in=='.') {
90         IsD=true;
91         num=0;
92     } else num=in-'0';
93     if(!IsD) {
94         while(in=getchar(),in>='0'&&in<='9') {
95             num*=10;
96             num+=in-'0';
97         }
98     }
99     if(in!='.') {
100         if(IsN) num=-num;

```



```

101     return true;
102 } else {
103     while(in=getchar(),in>='0'&&in<='9') {
104         num+=Dec*(in-'0');
105         Dec*=0.1;
106     }
107 }
108 if(IsN) num=-num;
109 return true;
110 }
111
112 void Out(LL a) {
113     if(a < 0) {
114         putchar('-');
115         a = -a;
116     }
117     if(a >= 10) Out(a / 10);
118     putchar(a % 10 + '0');
119 }
120 void print(LL a) {
121     Out(a),puts("");
122 }
123 //freopen( "in.txt" , "r" , stdin );
124 //freopen( "data.txt" , "w" , stdout );
125 //cerr << "run time is " << clock() << endl;
126 /*struct Point
127 {
128     double x, y;
129     Point(const Point& rhs): x(rhs.x), y(rhs.y) { } //拷贝构造函数
130     Point(double x = 0, double y = 0) : x(x), y(y) { }
131     inline void input()
132     {
133         scanf("%lf%lf",&x,&y);
134     }
135     inline void print()
136     {
137         printf("%.6lf %.6lf\n",x,y);
138     }
139 };*/
140 db sqr(db x) {
141     return x*x;
142 }
143 int dcmp(double x) {
144     if(fabs(x) < EPS) return 0;
145     else return x < 0 ? -1 : 1;
146 }
147 struct Circle {
148     double x, y, r, angle;
149     int d;
150     Circle() {}
151     Circle(double xx, double yy, double ang = 0, int t = 0) {
152         x = xx;
153         y = yy;
154         angle = ang;
155         d = t;
156     }
157     void get() {
158         scanf("%lf%lf%lf", &x, &y, &r);
159         d = 1;
160     }
161 };
162 Circle cir[maxn],tp[maxn*2];
163 double area[maxn];
164 double dis(Circle a,Circle b) {

```

```

165     return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
166 }
167 double cross(Circle p0, Circle p1, Circle p2) {
168     return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y) * (p2.x - p0.x);
169 }
170 //圆相交
171 int CirCrossCir(Circle p1, double r1, Circle p2, double r2, Circle &cp1, Circle &cp2) {
172     double mx = p2.x - p1.x, sx = p2.x + p1.x, mx2 = mx * mx;
173     double my = p2.y - p1.y, sy = p2.y + p1.y, my2 = my * my;
174     double sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) * (sq - sqr(r1 + r2));
175     if (d + eps < 0) return 0;
176     if (d < eps) d = 0;
177     else d = sqrt(d);
178     double x = mx * ((r1 + r2) * (r1 - r2) + mx * sx) + sx * my2;
179     double y = my * ((r1 + r2) * (r1 - r2) + my * sy) + sy * mx2;
180     double dx = mx * d, dy = my * d;
181     sq *= 2;
182     cp1.x = (x - dy) / sq;
183     cp1.y = (y + dx) / sq;
184     cp2.x = (x + dy) / sq;
185     cp2.y = (y - dx) / sq;
186     if (d > eps) return 2;
187     else return 1;
188 }
189 bool circmp(const Circle& u, const Circle& v) {
190     return dcmp(u.r - v.r) < 0;
191 }
192 bool cmp(const Circle& u, const Circle& v) {
193     if (dcmp(u.angle - v.angle)) return u.angle < v.angle;
194     return u.d > v.d;
195 }
196 //0.5*r*r*(K-sin(K))
197 double calc(Circle cir, Circle cp1, Circle cp2) {
198     double ans = (cp2.angle - cp1.angle) * sqr(cir.r)
199                 - cross(cir, cp1, cp2) + cross(Circle(0, 0), cp1, cp2);
200     return ans / 2;
201 }
202
203 void CirUnion(Circle cir[], int n) {
204     Circle cp1, cp2;
205     sort(cir, cir + n, circmp);
206     for (int i = 0; i < n; ++i)
207         for (int j = i + 1; j < n; ++j)
208             if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r) <= 0)
209                 cir[i].d++;
210     for (int i = 0; i < n; ++i) {
211         int tn = 0, cnt = 0;
212         for (int j = 0; j < n; ++j) {
213             if (i == j) continue;
214             if (CirCrossCir(cir[i], cir[i].r, cir[j], cir[j].r,
215                             cp2, cp1) < 2) continue;
216             cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);
217             cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);
218             cp1.d = 1;
219             tp[tn++] = cp1;
220             cp2.d = -1;
221             tp[tn++] = cp2;
222             if (dcmp(cp1.angle - cp2.angle) > 0) cnt++;
223         }
224         tp[tn++] = Circle(cir[i].x - cir[i].r, cir[i].y, pi, -cnt);
225         tp[tn++] = Circle(cir[i].x + cir[i].r, cir[i].y, -pi, cnt);
226         sort(tp, tp + tn, cmp);
227         int p, s = cir[i].d + tp[0].d;
228         for (int j = 1; j < tn; ++j) {

```

```

229     p = s;
230     s += tp[j].d;
231     area[p] += calc(cir[i], tp[j - 1], tp[j]);
232 }
233 }
234 }
235 int n;
236 void solve() {
237     for(int i=0; i<n; i++)
238         cir[i].get();
239     me(area,0);
240     CirUnion(cir,n);
241     for(int i=1; i<=n; i++) {
242         area[i]-=area[i+1];
243         printf("[%d]_=%.3f\n", i, area[i]);
244     }
245 }
246 int main() {
247     while(scanf("%d",&n)!=EOF)
248         solve();
249 }

```

### universe

```

1
2 Point CircumCenter(Point a,Point b,Point c) { //三角形的外心
3     Point cp;
4     double a1 = b.x-a.x,b1 = b.y-a.y,c1 = (a1*a1 + b1*b1)/2;
5     double a2 = c.x-a.x,b2 = c.y-a.y,c2 = (a2*a2 + b2*b2)/2;
6     double d = a1*b2 - a2*b1;
7     cp.x = a.x + (c1*b2-c2*b1)/d;
8     cp.y = a.y + (a1*c2-a2*c1)/d;
9     return cp;
10 }

```

### 3.1.3 Convex hull

```

1 /* 2D Convex Hull, by Abreto <m@abreto.net>. */
2 #include "2d_base.hh"
3 #include <cmath>
4 #include <algorithm>
5
6 using namespace std;
7
8 point O;
9
10 bool comp_angle(point_t a, point_t b) {
11     double t = (a-O).X(b-O);
12     if(fe(t,0.0)) return fl((b-O).mag2(),(a-O).mag2());
13     else return fl(0.0,t);
14 }
15
16 void convex_hull_graham(vp& convex, vp src) {
17     int i = 0, top = 0;
18     O = src[0];
19     for(auto pt : src)
20         if( pt.x < O.x || (pt.x == O.x && pt.y < O.y))
21             O = pt;
22     sort(src.begin(), src.end(), comp_angle);
23     convex.push_back(src[0]);
24     convex.push_back(src[1]);
25     top = 1;
26     for(i = 2; i < src.size(); ++i) {

```

```

27     while(top>1 && fle((convex[top]-convex[top-1]).X(src[i]-convex[top]),0.0)) {
28         convex.pop_back();
29         --top;
30     }
31     convex.push_back(src[i]);
32     ++top;
33 }
34 }

```

### 3.1.4 Intersect Area

```

1  #include <cstdio>
2  #include <cmath>
3  #include <algorithm>
4
5  using namespace std;
6
7  // #define inf 1000000000000000
8  #define M 8
9  #define LL long long
10 #define eps 1e-12
11 #define PI acos(-1.0)
12 using namespace std;
13 struct node {
14     double x,y;
15     node() {}
16     node(double xx,double yy) {
17         x=xx;
18         y=yy;
19     }
20     node operator -(node s) {
21         return node(x-s.x,y-s.y);
22     }
23     node operator +(node s) {
24         return node(x+s.x,y+s.y);
25     }
26     double operator *(node s) {
27         return x*s.x+y*s.y;
28     }
29     double operator ^(node s) {
30         return x*s.y-y*s.x;
31     }
32 };
33 double max(double a,double b) {
34     return a>b?a:b;
35 }
36 double min(double a,double b) {
37     return a<b?a:b;
38 }
39 double len(node a) {
40     return sqrt(a*a);
41 }
42 double dis(node a,node b) { //两点之间的距离
43     return len(b-a);
44 }
45 double cross(node a,node b,node c) { //叉乘
46     return (b-a)^(c-a);
47 }
48 double dot(node a,node b,node c) { //点积
49     return (b-a)*(c-a);
50 }
51 int judge(node a,node b,node c) { //判断c是否在ab线段上 (前提是c在直线ab上)
52     if(c.x>=min(a.x,b.x)

```

```

53     && c.x <= max(a.x, b.x)
54     && c.y >= min(a.y, b.y)
55     && c.y <= max(a.y, b.y))
56     return 1;
57     return 0;
58 }
59 double area(node b, node c, double r) {
60     node a(0.0, 0.0);
61     if(dis(b, c) < eps)
62         return 0.0;
63     double h = fabs(cross(a, b, c)) / dis(b, c);
64     if(dis(a, b) > r - eps && dis(a, c) > r - eps) { // 两个端点都在圆的外面则分为两种情况
65         double angle = acos(dot(a, b, c) / dis(a, b) / dis(a, c));
66         if(h > r - eps) {
67             return 0.5 * r * r * angle;
68         } else if(dot(b, a, c) > 0 && dot(c, a, b) > 0) {
69             double angle1 = 2 * acos(h / r);
70             return 0.5 * r * r * fabs(angle - angle1) + 0.5 * r * r * sin(angle1);
71         } else {
72             return 0.5 * r * r * angle;
73         }
74     } else if(dis(a, b) < r + eps && dis(a, c) < r + eps) { // 两个端点都在圆内的情况
75         return 0.5 * fabs(cross(a, b, c));
76     } else { // 一个端点在圆上一个端点在圆内的情况
77         if(dis(a, b) > dis(a, c)) { // 默认b在圆内
78             swap(b, c);
79         }
80         if(fabs(dis(a, b)) < eps) { // ab距离为0直接返回0
81             return 0.0;
82         }
83         if(dot(b, a, c) < eps) {
84             double angle1 = acos(h / dis(a, b));
85             double angle2 = acos(h / r) - angle1;
86             double angle3 = acos(h / dis(a, c)) - acos(h / r);
87             return 0.5 * dis(a, b) * r * sin(angle2) + 0.5 * r * r * angle3;
88         } else {
89             double angle1 = acos(h / dis(a, b));
90             double angle2 = acos(h / r);
91             double angle3 = acos(h / dis(a, c)) - angle2;
92             return 0.5 * r * dis(a, b) * sin(angle1 + angle2) + 0.5 * r * r * angle3;
93         }
94     }
95 }
96 }
97
98 node A, B, C;
99 int R;
100
101 bool compar(node &p1, node &p2) {
102     return (p1 ^ p2) > eps;
103 }
104
105 double f(double x, double y) {
106     node O(x, y);
107     node p[8];
108     p[0] = A - O;
109     p[1] = B - O;
110     p[2] = C - O;
111     sort(p, p + 3, compar);
112     p[3] = p[0];
113     O = node(0, 0);
114     double sum = 0;
115     /* <!-- 求面积交部分 */
116     for(int i = 0; i < 3; i++) { /* 按顺或逆时针顺序最后取绝对值就好 */

```

```

117     int j=i+1;
118     double s=area(p[i],p[j],(double)R);
119     if(cross(0,p[i],p[j])>0)
120         sum+=s;
121     else
122         sum-=s;
123 }
124 if(sum < -eps) sum = -sum;
125 /* --> */
126 return sum;
127 }
128
129 double trfind(double x, double y1, double y2) {
130     double l = y1, r = y2;
131     while(r-l>eps) {
132         double mid = (l+r)/2.0;
133         double mmid = (mid+r)/2.0;
134         if( f(x,mmid) > f(x,mid)+eps )
135             l = mid;
136         else
137             r = mmid;
138     }
139     return f(x,l);
140 }
141
142 double findmin(double x1, double x2, double y1, double y2) {
143     double l = x1, r = x2;
144     while(r-l>eps) {
145         double mid = (l+r)/2.0;
146         double mmid = (mid+r)/2.0;
147         if( trfind(mmid,y1,y2) > trfind(mid,y1,y2)+eps )
148             l = mid;
149         else
150             r = mmid;
151     }
152     return trfind(l,y1,y2);
153 }
154
155 double ans(int a, int b, int c, int r) {
156     A = node(0,0);
157     B = node((double)c,0);
158     R = r;
159     double da = a, db = b, dc = c;
160     double cosa = (db*db+dc*dc-da*da)/(2.0*db*dc);
161     double alpha = acos(cosa);
162     C = node(db*cosa, db*sin(alpha));
163     return findmin(0.0, c, 0.0, db*sin(alpha));
164 }
165
166 int main(void) {
167     int a = 0, b = 0, c = 0, r = 0;
168     while(EOF != scanf("%d%d%d%d",&a,&b,&c,&r) && (a||b||c||r))
169         printf("%.8lf\n", ans(a,b,c,r));
170     return 0;
171 }

```

### 3.1.5 Universe

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct Point {
5     double x, y;

```

```

6   Point(double x = 0, double y = 0) : x(x), y(y) {}
7   };
8
9   typedef Point Vector;
10
11  Vector operator + (Vector A, Vector B) {
12      return Vector(A.x + B.x, A.y + B.y);
13  }
14  Vector operator - (Vector A, Vector B) {
15      return Vector(A.x - B.x, A.y - B.y);
16  }
17  Vector operator * (Vector A, double p) {
18      return Vector(A.x*p, A.y*p);
19  }
20  Vector operator / (Vector A, double p) {
21      return Vector(A.x/p, A.y/p);
22  }
23
24  bool operator < (const Point& a, const Point b) {
25      return a.x < b.x || (a.x == b.x && a.y < b.y);
26  }
27
28  const double EPS = 1e-10;
29
30  int dcmp(double x) {
31      if(fabs(x) < EPS) return 0;
32      else return x < 0 ? -1 : 1;
33  }
34
35  bool operator == (const Point& a, const Point& b) {
36      return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y);
37  }
38
39  //向量a的极角
40  double Angle(const Vector& v) {
41      return atan2(v.y, v.x); // \share\CodeBlocks\templates\wizard\console\cpp
42  }
43
44  //向量点积
45  double Dot(Vector A, Vector B) {
46      return A.x*B.x + A.y*B.y;
47  }
48
49  //向量长度 \share\CodeBlocks\templates\wizard\console\cpp
50  double Length(Vector A) {
51      return sqrt(Dot(A, A));
52  }
53
54  //向量夹角
55  double Angle(Vector A, Vector B) {
56      return acos(Dot(A, B) / Length(A) / Length(B));
57  }
58
59  //向量叉积
60  double Cross(Vector A, Vector B) {
61      return A.x*B.y - A.y*B.x;
62  }
63
64  //三角形有向面积的二倍
65  double Area2(Point A, Point B, Point C) {
66      return Cross(B-A, C-A);
67  }
68
69  //向量逆时针旋转rad度(弧度)

```

```

70 Vector Rotate(Vector A, double rad) {
71     return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
72 }
73
74 //计算向量A的单位法向量。左转90°, 把长度归一。调用前确保A不是零向量。
75 Vector Normal(Vector A) {
76     double L = Length(A);
77     return Vector(-A.y/L, A.x/L);
78 }
79
80 /*****
81 使用复数类实现点及向量的简单操作
82
83 #include <complex>
84 typedef complex<double> Point;
85 typedef Point Vector;
86
87 double Dot(Vector A, Vector B) { return real(conj(A)*B)}
88 double Cross(Vector A, Vector B) { return imag(conj(A)*B);}
89 Vector Rotate(Vector A, double rad) { return A*exp(Point(0, rad)); }
90
91 *****/
92
93 /*****
94 * 用直线上的一点p0和方向向量v表示一条指向。直线上的所有点P满足P = P0+t*v;
95 * 如果知道直线上的两个点则方向向量为B-A, 所以参数方程为A+(B-A)*t;
96 * 当t 无限制时, 该参数方程表示直线。
97 * 当t > 0时, 该参数方程表示射线。
98 * 当 0 < t < 1时, 该参数方程表示线段。
99 *****/
100
101 //直线交点, 须确保两直线有唯一交点。
102 Point GetLineIntersection(Point P, Vector v, Point Q, Vector w) {
103     Vector u = P - Q;
104     double t = Cross(w, u)/Cross(v, w);
105     return P+v*t;
106 }
107
108 //点到直线距离
109 double DistanceToLine(Point P, Point A, Point B) {
110     Vector v1 = B - A, v2 = P - A;
111     return fabs(Cross(v1, v2) / Length(v1)); //不取绝对值, 得到的是有向距离
112 }
113
114 //点到线段的距离
115 double DistanceToSegmentS(Point P, Point A, Point B) {
116     if(A == B) return Length(P-A);
117     Vector v1 = B-A, v2 = P-A, v3 = P-B;
118     if(dcmp(Dot(v1, v2)) < 0) return Length(v2);
119     else if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
120     else return fabs(Cross(v1, v2)) / Length(v1);
121 }
122
123 //点在直线上的投影
124 Point GetLineProjection(Point P, Point A, Point B) {
125     Vector v = B - A;
126     return A+v*(Dot(v, P-A)/Dot(v, v));
127 }
128
129 //线段相交判定, 交点不在一条线段的端点
130 bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2) {
131     double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);
132     double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);
133     return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) < 0;

```



```

134 }
135
136 //判断点是否在点段上, 不包含端点
137 bool OnSegment(Point P, Point a1, Point a2) {
138     return dcmp(Cross(a1-P, a2-P) == 0 && dcmp((Dot(a1-P, a2-P)) < 0));
139 }
140
141 //计算凸多边形面积
142 double ConvexPolygonArea(Point *p, int n) {
143     double area = 0;
144     for(int i = 1; i < n-1; i++)
145         area += Cross(p[i] - p[0], p[i+1] - p[0]);
146     return area/2;
147 }
148
149 //计算多边形的有向面积
150 double PolygonArea(Point *p, int n) {
151     double area = 0;
152     for(int i = 1; i < n-1; i++)
153         area += Cross(p[i] - p[0], p[i+1] - p[0]);
154     return area/2;
155 }
156
157 /*****
158 * Morley定理: 三角形每个内角的三等分线, 相交成的三角形是等边三角形。
159 * 欧拉定理: 设平面图的定点数, 边数和面数分别为V,E,F。则V+F-E = 2;
160 *****/
161
162 struct Circle {
163     Point c;
164     double r;
165
166     Circle(Point c, double r) : c(c), r(r) {}
167     //通过圆心角确定圆上坐标
168     Point point(double a) {
169         return Point(c.x + cos(a)*r, c.y + sin(a)*r);
170     }
171 };
172
173 struct Line {
174     Point p;
175     Vector v;
176     double ang;
177     Line() {}
178     Line(Point p, Vector v) : p(p), v(v) {}
179     bool operator < (const Line& L) const {
180         return ang < L.ang;
181     }
182 };
183
184 //直线和圆的交点, 返回交点个数, 结果存在sol中。
185 //该代码没有清空sol。
186 int getLineCircleInterseccion(Line L, Circle C, double& t1, double& t2, vector<Point>& sol) {
187     double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
188     double e = a*a + c*c, f = 2*(a*b + c*d), g = b*b + d*d - C.r*C.r;
189     double delta = f*f - 4*e*g;
190     if(dcmp(delta) < 0) return 0; //相离
191     if(dcmp(delta) == 0) { //相切
192         t1 = t2 = -f / (2*e);
193         sol.push_back(C.point(t1));
194         return 1;
195     }
196     //相交

```

```

197     t1 = (-f - sqrt(delta)) / (2*e);
198     sol.push_back(C.point(t1));
199     t2 = (-f + sqrt(delta)) / (2*e);
200     sol.push_back(C.point(t2));
201     return 2;
202 }
203
204 //两圆相交
205 int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol) {
206     double d = Length(C1.c - C2.c);
207     if(dcmp(d) == 0) {
208         if(dcmp(C1.r - C2.r == 0)) return -1;    //两圆完全重合
209         return 0;                               //同心圆, 半径不一样
210     }
211     if(dcmp(C1.r + C2.r - d) < 0) return 0;
212     if(dcmp(fabs(C1.r - C2.r) == 0)) return -1;
213
214     double a = Angle(C2.c - C1.c);            //向量C1C2的极角
215     double da = acos((C1.r*C1.r + d*d - C2.r*C2.r) / (2*C1.r*d));
216     //C1C2到C1P1的角
217     Point p1 = C1.point(a-da), p2 = C1.point(a+da);
218     sol.push_back(p1);
219     if(p1 == p2) return 1;
220     sol.push_back(p2);
221     return 2;
222 }
223
224 const double PI = acos(-1);
225 //过定点做圆的切线
226 //过点p做圆C的切线, 返回切线个数。v[i]表示第i条切线
227 int getTangents(Point p, Circle C, Vector* v) {
228     Vector u = C.c - p;
229     double dist = Length(u);
230     if(dist < C.r) return 0;
231     else if(dcmp(dist - C.r) == 0) {
232         v[0] = Rotate(u, PI/2);
233         return 1;
234     } else {
235         double ang = asin(C.r / dist);
236         v[0] = Rotate(u, -ang);
237         v[1] = Rotate(u, +ang);
238         return 2;
239     }
240 }
241
242 //两圆的公切线
243 //返回切线的个数, -1表示有无数条公切线。
244 //a[i], b[i] 表示第i条切线在圆A, 圆B上的切点
245 int getTangents(Circle A, Circle B, Point *a, Point *b) {
246     int cnt = 0;
247     if(A.r < B.r) {
248         swap(A, B);
249         swap(a, b);
250     }
251     int d2 = (A.c.x - B.c.x)*(A.c.x - B.c.x) + (A.c.y - B.c.y)*(A.c.y - B.c.y);
252     int rdiff = A.r - B.r;
253     int rsum = A.r + B.r;
254     if(d2 < rdiff*rdiff) return 0;    //内含
255     double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
256     if(d2 == 0 && A.r == B.r) return -1;    //无限多条切线
257     if(d2 == rdiff*rdiff) {           //内切一条切线
258         a[cnt] = A.point(base);
259         b[cnt] = B.point(base);
260         cnt++;

```

```

261     return 1;
262 }
263 //有外共切线
264 double ang = acos((A.r-B.r) / sqrt(d2));
265 a[cnt] = A.point(base+ang);
266 b[cnt] = B.point(base+ang);
267 cnt++;
268 a[cnt] = A.point(base-ang);
269 b[cnt] = B.point(base-ang);
270 cnt++;
271 if(d2 == rsum*rsum) { //一条公切线
272     a[cnt] = A.point(base);
273     b[cnt] = B.point(PI+base);
274     cnt++;
275 } else if(d2 > rsum*rsum) { //两条公切线
276     double ang = acos((A.r + B.r) / sqrt(d2));
277     a[cnt] = A.point(base+ang);
278     b[cnt] = B.point(PI+base+ang);
279     cnt++;
280     a[cnt] = A.point(base-ang);
281     b[cnt] = B.point(PI+base-ang);
282     cnt++;
283 }
284 return cnt;
285 }
286
287 typedef vector<Point> Polygon;
288
289 //点 在多边形内的判定
290 int isPointInPolygon(Point p, Polygon poly) {
291     int wn = 0;
292     int n = poly.size();
293     for(int i = 0; i < n; i++) {
294         if(OnSegment(p, poly[i], poly[(i+1)%n])) return -1; //在边界上
295         int k = dcmp(Cross(poly[(i+1)%n]-poly[i], p-poly[i]));
296         int d1 = dcmp(poly[i].y - p.y);
297         int d2 = dcmp(poly[(i+1)%n].y - p.y);
298         if(k > 0 && d1 <= 0 && d2 > 0) wn++;
299         if(k < 0 && d2 <= 0 && d1 > 0) wn++;
300     }
301     if(wn != 0) return 1; //内部
302     return 0; //外部
303 }
304
305 //凸包
306 /*****
307 * 输入点数组p, 个数为p, 输出点数组ch。返回凸包顶点数
308 * 不希望凸包的边上有输入点, 把两个<= 改成 <
309 * 高精度要求时建议用dcmp比较
310 * 输入点不能有重复点。函数执行完以后输入点的顺序被破坏
311 *****/
312 int ConvexHull(Point *p, int n, Point* ch) {
313     sort(p, p+n); //先比较x坐标, 再比较y坐标
314     int m = 0;
315     for(int i = 0; i < n; i++) {
316         while(m > 1 && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
317         ch[m++] = p[i];
318     }
319     int k = m;
320     for(int i = n-2; i >= 0; i++) {
321         while(m > k && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
322         ch[m++] = p[i];
323     }
324     if(n > 1) m--;

```

```

325     return m;
326 }
327
328 //用有向直线A→B切割多边形poly, 返回“左侧”。 如果退化, 可能会返回一个单点或者线段
329 //复杂度O(n2);
330 Polygon CutPolygon(Polygon poly, Point A, Point B) {
331     Polygon newpoly;
332     int n = poly.size();
333     for(int i = 0; i < n; i++) {
334         Point C = poly[i];
335         Point D = poly[(i+1)%n];
336         if(dcmp(Cross(B-A, C-A)) >= 0) newpoly.push_back(C);
337         if(dcmp(Cross(B-A, C-D)) != 0) {
338             Point ip = GetLineIntersection(A, B-A, C, D-C);
339             if(OnSegment(ip, C, D)) newpoly.push_back(ip);
340         }
341     }
342     return newpoly;
343 }
344
345 //半平面交
346
347 //点p再有向直线L的左边。(线上不算)
348 bool Onleft(Line L, Point p) {
349     return Cross(L.v, p-L.p) > 0;
350 }
351
352 //两直线交点, 假定交点唯一存在
353 Point GetIntersection(Line a, Line b) {
354     Vector u = a.p - b.p;
355     double t = Cross(b.v, u) / Cross(a.v, b.v);
356     return a.p+a.v*t;
357 }
358
359 int HalfplaneIntersection(Line* L, int n, Point* poly) {
360     sort(L, L+n);           //按极角排序
361
362     int first, last;         //双端队列的第一个元素和最后一个元素
363     Point *p = new Point[n]; //p[i]为q[i]和q[i+1]的交点
364     Line *q = new Line[n];   //双端队列
365     q[first = last = 0] = L[0]; //队列初始化为只有一个半平面L[0]
366     for(int i = 0; i < n; i++) {
367         while(first < last && !Onleft(L[i], p[last-1])) last--;
368         while(first < last && !Onleft(L[i], p[first])) first++;
369         q[++last] = L[i];
370         if(fabs(Cross(q[last].v, q[last-1].v)) < EPS) {
371             last--;
372             if(Onleft(q[last], L[i].p)) q[last] = L[i];
373         }
374         if(first < last) p[last-1] = GetIntersection(q[last-1], q[last]);
375     }
376     while(first < last && !Onleft(q[first], p[last-1])) last--;
377     //删除无用平面
378     if(last-first <= 1) return 0; //空集
379     p[last] = GetIntersection(q[last], q[first]);
380
381     //从deque复制到输出中
382     int m = 0;
383     for(int i = first; i <= last; i++) poly[m++] = p[i];
384     return m;
385 }

```

## 4 Graph

### 4.1 Tree

#### 4.1.1 Universe

```

1  /* find root(重心) */
2
3
4  void findroot(int u, int fa) {
5      int i;
6      size[u] = 1;
7      f[u] = 0;
8      for (i = last[u]; i; i = e[i][2]) {
9          if (!vis[e[i][0]] && e[i][0] != fa) {
10             findroot(e[i][0], u);
11             size[u] += size[e[i][0]];
12             if (f[u] < size[e[i][0]])
13                 f[u] = size[e[i][0]];
14         }
15     }
16     if (f[u] < ALL - size[u])
17         f[u] = ALL - size[u];
18     if (f[u] < f[root]) root = u;
19 }
20
21 /* —— da —— */
22
23 int dep[MAXN+1];
24 int ancestor[MAXN+1][MAXLGN];
25 int minw[MAXN+1][MAXLGN];
26
27 void dfs(int u, int fa) {
28     ancestor[u][0] = fa;
29     dep[u] = dep[fa] + 1;
30     for(int e = u[front]; e; e = E[e].n) {
31         int v = E[e].v, w = E[e].w;
32         if(v != fa) {
33             minw[v][0] = w;
34             dfs(v, u);
35         }
36     }
37 }
38
39 void init_system(void) {
40     int i = 0, w = 0;
41     int t = 0;
42     dep[0] = -1;
43     dfs(1,0);
44     for(w = 1; (t=(1<<w)) < N; ++w)
45         for(i = 1; i <= N; ++i) if( dep[i] >= t ) {
46             ancestor[i][w] = ancestor[ancestor[i][w-1]][w-1];
47             minw[i][w] = min(minw[i][w-1], minw[ancestor[i][w-1]][w-1]);
48         }
49 }
50
51 int query(int a, int b) {
52     if(dep[a] < dep[b]) return query(b,a);
53     else { /* now dep[s] > dep[t] */
54         int i = 0;
55         int maxbit = MAXLGN-1;
56         int ret = INF;
57         //while((1<<maxbit) <= dep[a]) maxbit++;

```

```

58     /* first up a to same dep with b. */
59     for(i = maxbit; i >= 0; i--)
60         if(dep[a] - (1<<i) >= dep[b]) {
61             ret = min(ret, minw[a][i]);
62             a = ancestor[a][i];
63         }
64     if(a == b) return ret;
65     for(i = maxbit; i >= 0; i--)
66         if(dep[a] - (1<<i) >= 0 && ancestor[a][i] != ancestor[b][i]) {
67             ret = min(ret, min(minw[a][i], minw[b][i]));
68             a = ancestor[a][i];
69             b = ancestor[b][i];
70         }
71     ret = min(ret, min(minw[a][0], minw[b][0]));
72     return ret;
73 }
74 }

```

#### 4.1.2 Point Divide and Conquer

##### Version I

```

1  /* Tree::Point divide and conquer, by Abreto<m@abreto.net>. */
2  #include <bits/stdc++.h>
3
4  using namespace std;
5  typedef long long int ll;
6
7  #define MAXN    (100001)
8  #define MAXV    (MAXN+1)
9  #define MAXE    (MAXN<<1)
10 struct edge {
11     int v;
12     edge *n;
13     edge(void):v(0),n(NULL) {}
14     edge(int vv,edge *nn):v(vv),n(nn) {}
15 };
16 int nE;
17 edge E[MAXE];
18 edge *front[MAXV];
19 int label[MAXV]; /* 0 for '(', 1 for ')' */
20 void add_edge(int u, int v) {
21     int ne = ++nE;
22     E[ne] = edge(v, u[front]);
23     u[front] = &(E[ne]);
24 }
25
26 int n;
27 ll ans;
28
29 char del[MAXV];
30 namespace findroot {
31     int ALL;
32     int nfind;
33     int vis[MAXV];
34     int size[MAXV];
35     int f[MAXV];
36     int root;
37     void __find(int u, int fa) {
38         vis[u] = nfind;
39         size[u] = 1;
40         f[u] = 0;
41         for(edge *e=u[front]; e; e = e->n) {

```

```

42     int v = e->v;
43     if((!del[v]) && (vis[v] != nfind) && (v != fa)) {
44         __find(v, u);
45         size[u] += size[v];
46         if(f[u] < size[v]) f[u] = size[v];
47     }
48 }
49 if(f[u] < ALL - size[u]) f[u] = ALL - size[u];
50 if(f[u] < f[root]) root = u;
51 }
52 int find(int u, int all) {
53     ++nfind;
54     ALL = all;
55     f[root = 0] = MAXV;
56     __find(u, 0);
57     return root;
58 }
59 }
60
61 namespace workspaces {
62 int maxdep;
63 int dep[MAXV];
64 ll cntin[MAXV], cntout[MAXV];
65 int in[2][MAXV]; /* 0 for '(', 1 for ')' */
66 int out[2][MAXV];
67 void getdeep(int u, int fa) {
68     dep[u] = dep[fa] + 1;
69     if(dep[u] > maxdep) maxdep = dep[u];
70     for(edge *e = u[front]; e; e = e->n)
71         if((!del[e->v]) && (fa != e->v))
72             getdeep(e->v, u);
73 }
74 void dfs(int u, int fa) {
75     {
76         /* out from root */
77         out[0][u] = out[0][fa];
78         out[1][u] = out[1][fa];
79         if(0 == label[u]) { /* meet '(' */
80             out[0][u]++;
81         } else { /* meet ')' */
82             if(out[0][u]) out[0][u]--;
83             else out[1][u]++;
84         }
85         if(out[0][u] == 0)
86             cntout[out[1][u]]++;
87     }
88     {
89         /* in to root */
90         in[0][u] = in[0][fa];
91         in[1][u] = in[1][fa];
92         if(0 == label[u]) { /* meet '(' */
93             if(in[1][u]) in[1][u]--;
94             else in[0][u]++;
95         } else { /* meet ')' */
96             in[1][u]++;
97         }
98         if(0 == in[1][u])
99             cntin[in[0][u]]++;
100     }
101     /* do something */
102     for(edge *e = u[front]; e; e = e->n) {
103         int v = e->v;
104         if((!del[v]) && (v != fa)) {
105             dfs(v, u);

```

```

106     }
107 }
108 }
109 inline void init_maxdep(void) {
110     maxdep = 0;
111 }
112 inline void update_maxdep(int u) {
113     dep[u] = 1;
114     if(dep[u] > maxdep) maxdep = dep[u];
115     for(edge *e = u[front]; e; e = e->n)
116         if((!del[e->v]))
117             getdeep(e->v, u);
118 }
119 inline void clear(void) {
120     for(int i = 0; i <= maxdep+1; ++i)
121         cntin[i] = cntout[i] = 0;
122 }
123 inline void work(int u) {
124     in[0][u] = in[1][u] = out[0][u] = out[1][u] = 0;
125     in[label[u]][u] = out[label[u]][u] = 1;
126     if(out[0][u] == 0) cntout[out[1][u]]++;
127     if(0 == in[1][u]) cntin[in[0][u]]++;
128     /* update in and out if neccessary */
129     for(edge *e = u[front]; e; e = e->n)
130         if(!del[e->v])
131             dfs(e->v, u);
132 }
133 };
134
135 ll count(int u, int p) {
136     ll ret = 0;
137     workspace::init_maxdep();
138     workspace::update_maxdep(u);
139     workspace::clear();
140     if(-1 == p) {
141         for(edge *e = u[front]; e; e = e->n)
142             if((!del[e->v]))
143                 workspace::work(e->v);
144         p = label[u];
145         /* single end */
146         if(0 == p) ret = workspace::cntout[1];
147         else ret = workspace::cntin[1];
148     } else {
149         workspace::work(u);
150     }
151     if(0 == p) { /* p is '(' */
152         for(int i = 0; i < workspace::maxdep; ++i) /* concatenation */
153             ret += workspace::cntin[i] * workspace::cntout[i+1];
154     } else { /* p is ')' */
155         for(int i = 0; i < workspace::maxdep; ++i) /* concatenation */
156             ret += workspace::cntin[i+1] * workspace::cntout[i];
157     }
158     return ret;
159 }
160
161 void handle(int u) {
162     del[u] = 1; /* delete current root. */
163     ans += count(u, -1);
164     /* do something */
165     for(edge *e = u[front]; e; e = e->n) {
166         int v = e->v;
167         if(!del[v]) {
168             ans -= count(v, label[u]);
169             /* do something */

```



```

170     int r = findroot::find(v, findroot::size[v]);
171     handle(r);
172 }
173 }
174 }
175
176 void proc(void) {
177     int r = findroot::find(1,n);
178     handle(r);
179 }
180
181 char ls[MAXV+1];
182 int main(void) {
183     int i = 0;
184     scanf("%d", &n);
185     scanf("%s", ls);
186     for(i = 0; i < n; ++i)
187         label[i+1] = ls[i] - '(';
188     for(i = 1; i < n; ++i) {
189         int ai, bi;
190         scanf("%d_%d", &ai, &bi);
191         add_edge(ai, bi);
192         add_edge(bi, ai);
193     }
194     proc();
195     printf("%lld\n", ans);
196     return 0;
197 }

```

## Version 2

```

1  /* 2016 ACM/ICPC Asia Regional Dalian. Problem , by Abreto<m@abreto.net>. */
2  #include <bits/stdc++.h>
3
4  using namespace std;
5  typedef long long int ll;
6
7  /* offset in [1,k] */
8  #define GET(i,offset) (((i)>>((offset)-1))&1)
9  #define SET(i,offset) ((i)|(1<<((offset)-1)))
10 #define REV(i,offset) ((i)^(1<<((offset)-1)))
11
12 #define MAXN    (50005)
13 #define MAXV    (MAXN+1)
14 #define MAXE    (MAXN<<1)
15 struct edge {
16     int v;
17     edge *n;
18     edge(void):v(0),n(NULL) {}
19     edge(int vv,edge *nn):v(vv),n(nn) {}
20 };
21 int nE;
22 edge E[MAXE];
23 edge *front[MAXV];
24 int label[MAXV];    /* each kind */
25 void add_edge(int u, int v) {
26     int ne = ++nE;
27     E[ne] = edge(v, u[front]);
28     u[front] = &(E[ne]);
29 }
30
31 int n, k;
32 ll ans;
33 int all_kind;
34

```

```

35 int ndel;
36 int del[MAXV];
37 namespace findroot {
38 int ALL;
39 ll nfind;
40 ll vis[MAXV];
41 int size[MAXV];
42 int f[MAXV];
43 int root;
44 void __find(int u, int fa) {
45     vis[u] = nfind;
46     size[u] = 1;
47     f[u] = 0;
48     for(edge *e=u[front]; e; e = e->n) {
49         int v = e->v;
50         if((del[v] != ndel) && (vis[v] != nfind) && (v != fa)) {
51             __find(v, u);
52             size[u] += size[v];
53             if(f[u] < size[v]) f[u] = size[v];
54         }
55     }
56     if(f[u] < ALL - size[u]) f[u] = ALL - size[u];
57     if(f[u] < f[root]) root = u;
58 }
59 int find(int u, int all) {
60     ++nfind;
61     ALL = all;
62     f[root = 0] = MAXV;
63     __find(u, 0);
64     return root;
65 }
66 }
67
68 namespace workspace {
69 ll cnt[1024];
70 int dp[MAXV];
71 void dfs(int u, int fa) {
72     dp[u] = dp[fa] | label[u];
73     cnt[dp[u]] ++;
74     /* dig into children */
75     for(edge *e = u[front]; e; e = e->n) {
76         int v = e->v;
77         if((del[v] != ndel) && (v != fa)) {
78             dfs(v, u);
79         }
80     }
81 }
82 inline void clear(void) {
83     for(int i = 1; i <= all_kind; ++i)
84         cnt[i] = 0;
85 }
86 inline void work(int u) {
87     dp[u] = label[u];
88     cnt[dp[u]] ++;
89     for(edge *e = u[front]; e; e = e->n)
90         if((del[e->v] != ndel))
91             dfs(e->v, u);
92 }
93 inline void show(void) {
94     for(int i = 0; i <= all_kind; ++i)
95         printf("cnt[%d]_u=%lld\n", i, cnt[i]);
96     for(int i = 1; i <= n; ++i)
97         printf("dp[%d]_u=%d\n", i, dp[i]);
98 }

```

```

99 };
100
101
102 ll count(int u, int p) {
103     ll ret = 0;
104     workspace::clear();
105     //printf("%d,%d : \n", u, p);
106     if(-1 == p) {
107         for(edge *e = u[front]; e; e = e->n)
108             if(((del[e->v]) != ndel))
109                 workspace::work(e->v);
110         p = label[u];
111         /* single end */
112         for(int i = 1; i <= all_kind; i++)
113             if(all_kind == (i|p))
114                 ret += (workspace::cnt[i]<<1);
115     } else {
116         workspace::work(u);
117     }
118     //workspace::show();
119     for(int i = 1; i <= all_kind; ++i)
120         if( workspace::cnt[i] > 0 )
121             for(int j = 1; j <= all_kind; ++j)
122                 if(all_kind == (i|j))
123                     ret += workspace::cnt[i] * workspace::cnt[j];
124     //printf("%lld\n", ret);
125     return ret;
126 }
127
128 void handle(int u) {
129     //printf("proprocessing %d\n", u);
130     del[u] = ndel; /* delete current root. */
131     ans += count(u, -1);
132     /* do something */
133     for(edge *e = u[front]; e; e = e->n) {
134         int v = e->v;
135         if(del[v] != ndel) {
136             ans -= count(v, label[u]);
137             /* do something */
138             int r = findroot::find(v, findroot::size[v]);
139             handle(r);
140         }
141     }
142 }
143
144 void proc(void) {
145     int r = findroot::find(1,n);
146     handle(r);
147 }
148
149 void clear(void) {
150     int i;
151     ans = 0;
152     nE = 0;
153     for(i = 0; i <= n; ++i) {
154         front[i] = NULL;
155     }
156     //findroot::nfind = 0;
157     ndel++;
158 }
159
160 void mozhu(void) {
161     int i = 0;
162     int li;

```

```

163     for(i = 1; i <= n; ++i) {
164         scanf("%d", &li);
165         label[i] = 1<<(li-1);
166     }
167     for(i = 1; i < n; ++i) {
168         int ai, bi;
169         scanf("%d%d", &ai, &bi);
170         add_edge(ai, bi);
171         add_edge(bi, ai);
172     }
173     all_kind = (1<<k)-1;
174     proc();
175     if(1 == k) ans += n;
176     printf("%lld\n", ans);
177 }
178
179 int main(void) {
180     while( EOF != scanf("%d%d", &n, &k) ) {
181         clear();
182         mozhu();
183     }
184     return 0;
185 }

```

#### 4.1.3 Heavy chain decomposition

```

1  /* bzoj 1036 */
2  /* 树链剖分 */
3  #include <bits/stdc++.h>
4
5  using namespace std;
6
7  #define MAXN    30030
8  #define MAXM    (MAXN<<1)
9  struct edge {
10     int v;
11     edge *n;
12     edge(void) {}
13     edge(int vv, edge *nn):v(vv),n(nn) {}
14 };
15 typedef edge *ep;
16 int nE;
17 edge E[MAXM];
18 ep front[MAXN];
19 void add_edge(int u, int v) {
20     int ne = ++nE;
21     E[ne] = edge(v, u[front]);
22     u[front] = &(E[ne]);
23 }
24
25 int n;
26 int fa[MAXN], son[MAXN], sz[MAXN], dep[MAXN];
27 int top[MAXN];
28 int id[MAXN];
29 int tot;
30
31 void calc(int u, int uf) {
32     dep[u] = dep[uf] + 1;
33     fa[u] = uf;
34     sz[u] = 1;
35     son[u] = -1;
36     for(ep e = u[front]; e; e = e->n) {
37         if(e->v != uf) {

```

```

38     calc(e->v, u);
39     sz[u] += sz[e->v];
40     if( -1 == son[u] || sz[son[u]] < sz[e->v] )
41         son[u] = e->v;
42 }
43 }
44 }
45 void link(int u, int f) {
46     id[u] = (++tot);
47     top[u] = f;
48     if(son[u] > 0) {
49         link(son[u], f);
50     }
51     for(ep e = u[front]; e; e = e->n) {
52         if(e->v != fa[u] && e->v != son[u]) {
53             link(e->v, e->v);
54         }
55     }
56 }
57
58 /* 其实是树链剖分 */
59 void make_link_cut_tree(void) {
60     calc(1, 0);
61     link(1, 1);
62 }
63
64 int w[MAXN];
65 int sum[MAXN<<2], mx[MAXN<<2];
66
67 void maintain(int o, int l, int r) {
68     sum[o] = sum[o<<1] + sum[o<<1|1];
69     mx[o] = max(mx[o<<1], mx[o<<1|1]);
70 }
71 void build(int o = 1, int l = 1, int r = n) {
72     if(r == l) {
73         sum[o] = w[l];
74         mx[o] = w[l];
75     } else {
76         int mid = l+r>>1;
77         build(o<<1, l, mid);
78         build(o<<1|1, mid+1, r);
79         maintain(o, l, r);
80     }
81 }
82 void update(int p, int x, int o = 1, int l = 1, int r = n) {
83     if(p <= l && r <= p) {
84         sum[o] = x;
85         mx[o] = x;
86     } else {
87         int mid = l+r>>1;
88         if(p <= mid) update(p,x,o<<1,l,mid);
89         else update(p,x,o<<1|1,mid+1,r);
90         maintain(o,l,r);
91     }
92 }
93 int qs(int L, int R, int o = 1, int l = 1, int r = n) {
94     if(R < l || r < L) return 0;
95     else if (L <= l && r <= R) {
96         return sum[o];
97     } else {
98         int mid = l+r>>1;
99         return qs(L,R,o<<1,l,mid)+qs(L,R,o<<1|1,mid+1,r);
100     }
101 }

```

```

102 int qm(int L, int R, int o = 1, int l = 1, int r = n) {
103     if(L <= l && r <= R) {
104         return mx[o];
105     } else {
106         int mid = l+r>>1;
107         if(R <= mid) return qm(L, R, o<<1, l, mid);
108         else if ( L > mid ) return qm(L, R, o<<1|1, mid+1, r);
109         else return max(qm(L, R, o<<1, l, mid),qm(L, R, o<<1|1, mid+1, r));
110     }
111 }
112
113 void change(int u, int t) {
114     update(id[u], t);
115 }
116 int qmax(int u, int v) {
117     int ret = -1000000000;
118     while(top[u] != top[v]) {
119         if( dep[top[u]] > dep[top[v]] ) {
120             /* jump u */
121             ret = max(ret, qm(id[top[u]], id[u]));
122             u = fa[top[u]];
123         } else {
124             ret = max(ret, qm(id[top[v]], id[v]));
125             v = fa[top[v]];
126         }
127     }
128     ret = max(ret, qm(min(id[u],id[v]),max(id[u],id[v])));
129     return ret;
130 }
131 int qsum(int u, int v) {
132     int ret = 0;
133     while(top[u] != top[v]) {
134         if( dep[top[u]] > dep[top[v]] ) {
135             /* jump u */
136             ret += qs(id[top[u]], id[u]);
137             u = fa[top[u]];
138         } else {
139             ret += qs(id[top[v]], id[v]);
140             v = fa[top[v]];
141         }
142     }
143     ret += qs(min(id[u],id[v]),max(id[u],id[v]));
144     return ret;
145 }
146
147 int main(void) {
148     int i;
149     scanf("%d", &n);
150     for(i = 1; i < n; ++i) {
151         int a, b;
152         scanf("%d%d", &a, &b);
153         add_edge(a, b);
154         add_edge(b, a);
155     }
156     make_link_cut_tree();
157     for(i = 1; i <= n; ++i) {
158         scanf("%d", &(w[id[i]]));
159     }
160     build();
161     scanf("%d", &i);
162     while(i--) {
163         char command[8];
164         int a, b;
165         scanf("%s%d%d", command, &a, &b);

```

```

166     if('C' == command[0]) change(a, b);
167     else if ('M' == command[1]) printf("%d\n", qmax(a, b));
168     else if ('S' == command[1]) printf("%d\n", qsum(a, b));
169 }
170 return 0;
171 }

```

## 4.2 2-SAT

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  namespace two_sat {
6  const int maxn = 100000;
7  const int maxm = 1000000;
8  struct edge {
9      int v;
10     edge *n;
11     edge(void):v(0),n(NULL) {}
12     edge(int vv, edge *nn):v(vv),n(nn) {}
13 };
14 typedef edge *ep;
15 int n;
16 int nE;
17 edge E[maxm];
18 ep front[maxn];
19 void add_edge(int u, int v) {
20     int ne = ++nE;
21     E[ne] = edge(v, u[front]);
22     u[front] = &(E[ne]);
23 }
24 /* (x = xval or y = yval), indexed from 0 */
25 void add_clause(int x, int xv, int y, int yv) {
26     x = x*2 + xv;
27     y = y*2 + yv;
28     add_edge(x^1, y);
29     add_edge(y^1, x);
30 }
31
32 char mark[maxn<<1];
33 int S[maxn<<1], c;
34 void init(int N) {
35     n = N;
36     for(int i = 0; i < n*2; ++i) {
37         i[front] = NULL;
38         i[mark] = 0;
39     }
40     nE = 0;
41 }
42
43 int dfs(int x) {
44     if(mark[x^1]) return 0;
45     if(mark[x]) return 1;
46     mark[x] = 1;
47     S[c++] = x;
48     for(ep e = x[front]; e; e = e->n)
49         if(!dfs(e->v)) return 0;
50     return 1;
51 }
52
53 int solve(void) {

```

```

54 for(int i = 0; i < n*2; i += 2)
55     if(!mark[i] && !mark[i+1]) {
56         c = 0;
57         if(!dfs(i)) {
58             while(c > 0) mark[S---c] = 0;
59             if(!dfs(i+1)) return 0;
60         }
61     }
62     return 1;
63 }
64 }

```

### 4.3 Cut Edge and Point

```

1 Finding cut edges
2 The code below works properly because the lemma above (first lemma):
3   h[root] = 0
4   par[v] = -1
5   dfs (v):
6       d[v] = h[v]
7       color[v] = gray
8       for u in adj[v]:
9           if color[u] == white
10              then par[u] = v and dfs(u) and d[v] = min(
11                  d[v], d[u])
12                  if d[u] > h[v]
13                      then the edge v-u is a cut edge
14                      else if u != par[v])
15              then d[v] = min(d[v], h[u])
16              color[v] = black
17
18 In this code, h[v] = height of vertex v in the DFS
19 tree and d[v] = min(h[w] where there is at least
20 vertex u in subtree of v in the DFS tree where
21 there is an edge between u and w).
22
23 Finding cut vertices
24 The code below works properly because the lemma
25 above (first lemma):
26   h[root] = 0
27   par[v] = -1
28   dfs (v):
29       d[v] = h[v]
30       color[v] = gray
31       for u in adj[v]:
32           if color[u] == white
33              then par[u] = v and dfs(
34                  u) and d[v] = min(d[v]
35                      , d[u])
36                  if d[u] >= h[v]
37                      and (v != root
38                          or
39                          number_of_children
40                              (v) > 1)
41                      then the edge v
42                          is a cut
43                          vertex
44                      else if u != par
45                          [v])
46              then d[v] = min(d[v], h[u])
47              color[v] = black
48
49 In this code, h[v] = height of vertex v in
50 the DFS tree and d[v] = min(h[w] where

```



there is at least vertex  $u$  in subtree of  $v$  in the DFS tree where there is an edge between  $u$  and  $w$ ).

#### 4.4 Euler Path

```

1  /* Euler path, by Abreto<m@abreto.net>. */
2  #define MAXV    (1024)
3  #define MAXE    (MAXV*MAXV)
4
5  typedef struct {
6      int id;
7      int nxt;
8      int del;
9  } egde_t;
10 int front[MAXV];
11 egde_t edg[MAXE];
12 int d[MAXV];
13 int ind[MAXV], outd[MAXV];
14 int nedges;
15 void add_edge(int u, int v) {
16     int newedge = ++nedges;
17     edg[newedge].id = v;
18     edg[newedge].nxt = u[front];
19     edg[newedge].del = 0;
20     u[front] = newedge;
21     outd[u]++;
22     ind[v]++;
23     d[u]++;
24     d[v]++;
25 }
26 void del_edge(int u, int v) {
27     int e = 0;
28     for(e=u[front]; e; e=edg[e].nxt)
29         if(edg[e].id==v) {
30             edg[e].del = 1;
31             outd[u]--;
32             ind[v]--;
33             d[u]--;
34             d[v]--;
35             return;
36         }
37 }
38
39 int path[MAXV];
40 int l;
41
42 void add2path(int u) {
43     path[l++] = u;
44 }
45
46 /* Directed graph */
47 void euler(int x) {
48     if(outd[x]) {
49         int e = 0;
50         for(e=x[front]; e; e=edg[e].nxt)
51             if(!edg[e].del) {
52                 int v = edg[e].id;
53                 del_edge(x,v);
54                 euler(v);
55             }
56     }

```

```

57     add2path(x);
58 }
59
60 /* Undirected graph */
61 void euler(int x) {
62     if(d[x]) {
63         int e = 0;
64         for(e=x[front]; e; e=edg[e].nxt)
65             if(!edg[e].del) {
66                 int v = edg[e].id;
67                 del_edge(x,v);
68                 del_edge(v,x);
69                 euler(v);
70             }
71     }
72     add2path(x);
73 }

```

## 4.5 Shortest Path

### 4.5.1 Dijkstra

```

1  /* Shortest Path Dijkstra, by Abreto<m@abreto.net>. */
2  #include <cstdio>
3  #include <set>
4  #include <utility>
5
6  using namespace std;
7  typedef set< pair<int,int> > spii;
8
9  #define MAXN    512
10 #define MAXV    (MAXN*MAXN)
11
12 struct egde_t {
13     int id;
14     int nxt;
15 };
16 int front[MAXV];
17 egde_t edg[MAXV<<3];
18 int nedges;
19 void add_edge(int u, int v) {
20     int newedge = ++nedges;
21     edg[newedge].id = v;
22     edg[newedge].nxt = u[front];
23     u[front] = newedge;
24 }
25
26 int d[MAXV];
27 int vis[MAXN];
28 int solid[MAXV];
29
30 int dijkstra(int s, int t) {
31     int v = s[front];
32     spii q;
33     q.insert(make_pair(0, s));
34     while(!q.empty()) {
35         auto it = q.begin();
36         int u = it->second;
37         int v = u[front];
38         q.erase(it);
39         solid[u] = 1;
40         if(u == t) break;
41         while(v) {

```

```

42     int w = edg[v].id;
43     if(!solid[w]) {
44         if( (0==d[w]) || (d[u] + 1 < d[w]) ) {
45             q.erase(make_pair(d[w],w));
46             d[w] = d[u] + 1;
47             q.insert(make_pair(d[w],w));
48         }
49     }
50     v = edg[v].nxt;
51 }
52 }
53 return d[t];
54 }

```

#### 4.5.2 Shortest Path Fast Algorithm

```

1  /* Shortest Path Fast Algorithm, by Abreto<m@abreto.net>. */
2  #include <cstdio>
3  #include <cstring>
4  #include <queue>
5  #include <utility>
6
7  using namespace std;
8
9  #define MAXN    128
10
11 struct edge {
12     int v;
13     int w;
14     int n;
15 };
16 edge edg[MAXN<<1];
17 int nedg;
18 int indegree[MAXN];
19 int front[MAXN];
20 int find_edge(int u, int v) {
21     int e = u[front];
22     while(e) {
23         if(edg[e].v == v) return e;
24         e = edg[e].n;
25     }
26     return 0;
27 }
28 void add_edge(int u, int v, int w) {
29     int e = find_edge(u,v);
30     if(0==e) {
31         int newnode = ++nedg;
32         edg[newnode].v = v;
33         edg[newnode].w = w;
34         edg[newnode].n = u[front];
35         u[front] = newnode;
36         indegree[v]++;
37     } else {
38         edg[e].w = (w < edg[e].w)?w:(edg[e].w);
39     }
40 }
41
42 int n;
43
44 char inq[MAXN];
45 int vis[MAXN];
46 int d[MAXN];
47 int spfa(int s) { /* return 1 if fuhuan exists. */

```

```

48 queue<int> q;
49 memset(inq, 0, sizeof(inq));
50 memset(d, -1, sizeof(d));
51 memset(vis, 0, sizeof(vis));
52 d[s] = 0;
53 inq[s] = 1;
54 q.push(s);
55 while(!q.empty()) {
56     int u = q.front();
57     q.pop();
58     printf("proc_%d..\n", u);
59     inq[u] = 0;
60     if(vis[u]++ > n)
61         return 1;
62     for(int e = front[u]; e; e = edg[e].n) {
63         int v = edg[e].v, w = edg[e].w;
64         if( -1==d[v] || d[u] + w < d[v] ) {
65             d[v] = d[u] + w;
66             if(!inq[v]) {
67                 inq[v] = 1;
68                 q.push(v);
69             }
70         }
71     }
72 }
73 return 0;
74 }

```

#### 4.5.3 *K*-th shortest path

```

1 /**
2  * poj
3  * Problem#2449
4  * Accepted
5  * Time: 250ms
6  * Memory: 9252k
7  */
8 #include <iostream>
9 #include <fstream>
10 #include <sstream>
11 #include <algorithm>
12 #include <cstdio>
13 #include <cstdlib>
14 #include <cstring>
15 #include <ctime>
16 #include <cctype>
17 #include <cmath>
18 #include <vector>
19 #include <queue>
20 #include <stack>
21 #include <map>
22 #include <set>
23 #include <bitset>
24 using namespace std;
25 typedef bool boolean;
26
27 typedef class Edge {
28 public:
29     int end;
30     int next;
31     int w;
32
33     Edge(int end = 0, int next = -1, int w = 0):end(end), next(next), w(w) {

```

```

34 } Edge;
35
36 const int N = 1e3, M = 1e5;
37
38 typedef class MapManager {
39 public:
40     int cnt;
41     int h[N + 5];
42     Edge edge[M + 5];
43
44     MapManager() { }
45     MapManager(int n):cnt(-1) {
46 //         h = new int[(n + 1)];
47 //         edge = new Edge[(m + 1)];
48         memset(h, -1, sizeof(int) * (n + 1));
49     }
50
51     inline void addEdge(int u, int v, int w) {
52         edge[++cnt] = (Edge(v, h[u], w));
53 //         h[u] = (signed)edge.size() - 1;
54         h[u] = cnt;
55     }
56
57     inline int start(int node) {
58         return h[node];
59     }
60
61     Edge& operator [] (int pos) {
62         return edge[pos];
63     }
64 } MapManager;
65 #define m_endpos -1
66
67 int n, m;
68 MapManager g;
69 MapManager rg;
70 int s, t, k;
71 int ds[N + 5];
72
73 inline void init() {
74     scanf("%d%d", &n, &m);
75     memset(g.h, -1, sizeof(int) * (n + 1));
76     memset(rg.h, -1, sizeof(int) * (n + 1));
77     for(int i = 1, u, v, w; i <= m; i++) {
78         scanf("%d%d%d", &u, &v, &w);
79         g.addEdge(u, v, w);
80         rg.addEdge(v, u, w);
81     }
82     scanf("%d%d%d", &s, &t, &k);
83 //     ds = new int[(n + 1)];
84 }
85
86 #define g rg
87 #define f ds
88 #define que que1
89 boolean vis[N + 5];
90 queue<int> que;
91 boolean spfa(int s, int t) {
92     memset(f, 0x7f, sizeof(int) * (n + 1));
93     memset(vis, false, sizeof(boolean) * (n + 1));
94     que.push(s);
95     f[s] = 0;
96     while(!que.empty()) {
97         int e = que.front();

```

```

98     que.pop();
99     vis[e] = false;
100     for(int i = g.start(e); i != m_endpos; i = g[i].next) {
101         int& eu = g[i].end;
102         // cout << e << " " << eu << " " << i << endl;
103         if(f[e] + g[i].w < f[eu]) {
104             f[eu] = f[e] + g[i].w;
105             if(!vis[eu]) {
106                 que.push(eu);
107                 vis[eu] = true;
108             }
109         }
110     }
111 }
112 return (f[t] != 0x7f7f7f7f);
113 }
114 #undef g
115 #undef f
116 #undef que
117
118 typedef class Status {
119 public:
120     int node;
121     int dis;
122     int priority;
123
124     Status(int node = 0, int dis = 0):node(node), dis(dis), priority(h()) {
125
126     }
127     int h() {
128         return dis + ds[node];
129     }
130     boolean operator < (Status b) const {
131         return priority > b.priority;
132     }
133 } Status;
134
135 int label[N + 5];
136 priority_queue<Status> que;
137 int bfs(int s, int t) {
138     if(s == t) k++;
139     // label = new int[(n + 1)];
140     memset(label, 0, sizeof(int) * (n + 1));
141     que.push(Status(s, 0));
142     while(!que.empty()) {
143         Status e = que.top();
144         que.pop();
145         label[e.node]++;
146         if(e.node == t && label[e.node] == k)
147             return e.dis;
148         for(int i = g.start(e.node); i != m_endpos; i = g[i].next) {
149             if(label[g[i].end] < k)
150                 que.push(Status(g[i].end, e.dis + g[i].w));
151         }
152     }
153     return -1;
154 }
155
156 inline void solve() {
157     if(!spfa(t, s)) {
158         puts("-1");
159         return;
160     }
161     printf("%d", bfs(s, t));

```

```

162 }
163
164 int main() {
165     init();
166     solve();
167     return 0;
168 }

```

## 4.6 Maxflow

```

1  /* Max Flow Problem, by Abreto<m@abreto.net> */
2
3  #include <bits/stdc++.h>
4  using namespace std;
5
6  #define MAXV    (100000)
7  #define MAXE    (1000000)
8  struct edge {
9      static int N;
10     int v, w;
11     edge *n;
12     edge(void):v(0),w(0),n(NULL) {}
13     edge(int vv, int ww, edge *nn):v(vv),w(ww),n(nn) {}
14 };
15 int nE;
16 edge E[MAXE];
17 edge *front[MAXV];
18 void add_edge(int u, int v, int w) {
19     int ne = ++nE;
20     E[ne] = edge(v, w, u[front]);
21     u[front] = &(E[ne]);
22 }
23 edge *find_edge(int u, int v) {
24     for(edge *e = u[front]; e != NULL; e = e->n)
25         if(e->v == v)
26             return e;
27     return NULL;
28 }
29 void grant_e(int u, int v, int w) {
30     edge *e = find_edge(u, v);
31     if(NULL == e) add_edge(u,v,w);
32     else e->w += w;
33 }
34
35 int vis[MAXV];
36 int path[MAXV];
37 int dfs(int u, int t) {
38     vis[u] = 1;
39     if(u == t) return 1;
40     for(edge *e = u[front]; e != NULL; e = e->n) {
41         int v = e->v;
42         if(!vis[v] && e->w && dfs(v,t)) {
43             path[u] = v;
44             return 1;
45         }
46     }
47     return 0;
48 }
49 int find_path(int s, int t) {
50     memset(vis, 0, sizeof(vis));
51     return dfs(s,t);
52 }

```

```

53 int max_flow(int s, int t) {
54     int flow = 0;
55     while(find_path(s,t)) {
56         int i = 0;
57         int minf = find_edge(s,path[s])->w;
58         for(i = path[s]; i != t; i = path[i])
59             minf = min(minf, find_edge(i,path[i])->w);
60         for(i = s; i != t; i = path[i]) {
61             grant_e(i, path[i], -minf);
62             grant_e(path[i], i, minf);
63         }
64         flow += minf;
65     }
66     return flow;
67 }
68
69 /* Dinic */
70 #define N 1000
71 #define INF 1000000000
72
73 struct Edge {
74     int from,to,cap,flow;
75     Edge(int u,int v,int c,int f):from(u),to(v),cap(c),flow(f) {}
76 };
77
78 struct Dinic {
79     int n,m,s,t;//结点数, 边数 (包括反向弧), 源点编号, 汇点编号
80     vector<Edge>edges;//边表, dges[e]和dges[e^1]互为反向弧
81     vector<int>G[N];//邻接表, G[i][j]表示结点i的第j条边在e数组中的编号
82     bool vis[N]; //BFS的使用
83     int d[N]; //从起点到i的距离
84     int cur[N]; //当前弧下标
85
86     void addedge(int from,int to,int cap) {
87         edges.push_back(Edge(from,to,cap,0));
88         edges.push_back(Edge(to,from,0,0));
89         int m=edges.size();
90         G[from].push_back(m-2);
91         G[to].push_back(m-1);
92     }
93
94     bool bfs() {
95         memset(vis,0,sizeof(vis));
96         queue<int>Q;
97         Q.push(s);
98         d[s]=0;
99         vis[s]=1;
100         while(!Q.empty()) {
101             int x=Q.front();
102             Q.pop();
103             for(int i=0; i<G[x].size(); i++) {
104                 Edge&e=edges[G[x][i]];
105                 if(!vis[e.to]&&e.cap>e.flow) { //只考虑残量网络中的弧
106                     vis[e.to]=1;
107                     d[e.to]=d[x]+1;
108                     Q.push(e.to);
109                 }
110             }
111         }
112     }
113     return vis[t];
114 }
115
116 int dfs(int x,int a) { //x表示当前结点, a表示目前为止的最小残量

```



```

117     if(x==t||a==0)return a;//a等于0时及时退出，此时相当于断路了
118     int flow=0,f;
119     for(int&i=cur[x]; i<G[x].size(); i++) { //从上次考虑的弧开始，注意要使用引用，同
        时修改cur[x]
120         Edge&e=edges[G[x][i]);//e是一条边
121         if(d[x]+1==d[e.to]&&(f=dfs(e.to,min(a,e.cap-e.flow)))>0) {
122             e.flow+=f;
123             edges[G[x][i]^1].flow-=f;
124             flow+=f;
125             a-=f;
126             if(!a)break;//a等于0及时退出，当a!=0,说明当前节点还存在另一个曾广路分支。
127         }
128     }
129 }
130 return flow;
131 }
132
133 int Maxflow(int s,int t) { //主过程
134     this->s=s,this->t=t;
135     int flow=0;
136     while(bfs()) { //不停地用bfs构造分层网络，然后用dfs沿着阻塞流增广
137         memset(cur,0,sizeof(cur));
138         flow+=dfs(s,INF);
139     }
140     return flow;
141 }
142 };
143
144 /* ISAP */
145 struct Edge {
146     int from,to,cap,flow;
147 };
148 const int maxn=650;
149 const int INF=0x3f3f3f3f;
150 struct ISAP {
151     int n,m,s,t;//结点数，边数（包括反向弧），源点编号，汇点编号
152     vector<Edge>edges;
153     vector<int>G[maxn];
154     bool vis[maxn];
155     int d[maxn];
156     int cur[maxn];
157     int p[maxn];
158     int num[maxn];
159     void AddEdge(int from,int to,int cap) {
160         edges.push_back((Edge) {
161             from,to,cap,0
162         });
163         edges.push_back((Edge) {
164             to,from,0,0
165         });
166         m=edges.size();
167         G[from].push_back(m-2);
168         G[to].push_back(m-1);
169     }
170     bool RevBFS() {
171         memset(vis,0,sizeof(vis));
172         queue<int>Q;
173         Q.push(t);
174         d[t]=0;
175         vis[t]=1;
176         while(!Q.empty()) {
177             int x=Q.front();
178             Q.pop();
179             for(int i=0; i<G[x].size(); i++) {

```

```

180     Edge &e =edges[G[x][i]^1];
181     if(!vis[e.from]&&e.cap>e.flow) {
182         vis[e.from]=1;
183         d[e.from]=d[x]+1;
184         Q.push(e.from);
185     }
186 }
187 }
188 return vis[s];
189 }
190 int Augment() {
191     int x=t, a=INF;
192     while(x!=s) {
193         Edge &e = edges[p[x]];
194         a= min(a,e.cap-e.flow);
195         x=edges[p[x]].from;
196     }
197     x=t;
198     while(x!=s) {
199         edges[p[x]].flow+=a;
200         edges[p[x]^1].flow-=a;
201         x=edges[p[x]].from;
202     }
203     return a;
204 }
205 int Maxflow(int s,int t,int n) {
206     this->s=s,this->t=t,this->n=n;
207     int flow=0;
208     RevBFS();
209     memset(num,0,sizeof(num));
210     for(int i=0; i<n; i++) {
211         num[d[i]]++;
212     }
213     int x=s;
214     memset(cur,0,sizeof(cur));
215     while(d[s]<n) {
216         if(x==t) {
217             flow+=Augment();
218             x=s;
219         }
220         int ok=0;
221         for(int i=cur[x]; i<G[x].size(); i++) {
222             Edge &e =edges[G[x][i]];
223             if(e.cap>e.flow&&d[x]==d[e.to]+1) {
224                 ok=1;
225                 p[e.to]=G[x][i];
226                 cur[x]=i;
227                 x=e.to;
228                 break;
229             }
230         }
231         if(!ok) {
232             int m=n-1;
233             for(int i=0; i<G[x].size(); i++) {
234                 Edge &e =edges[G[x][i]];
235                 if(e.cap>e.flow)
236                     m=min(m,d[e.to]);
237             }
238             if(--num[d[x]]==0)
239                 break;
240             num[d[x]=m+1]++;
241             cur[x]=0;
242             if(x!=s)
243                 x=edges[p[x]].from;

```

```

244     }
245 }
246     return flow;
247 }
248 };
249 int main() {
250     int n,m,a,b,c,res;
251     while(scanf("%d%d",&m,&n)!=EOF) {
252         ISAP tmp;
253         for(int i=0; i<m; i++) {
254             scanf("%d%d%d",&a,&b,&c);
255             tmp.AddEdge(a,b,c);
256         }
257         res=tmp.Maxflow(1,n,n);
258         printf("%d\n",res);
259     }
260     return 0;
261 }

```

## 4.7 Strongly Connected Component

```

1  /* Kosaraju */
2  #define MAXN    10010
3  #define MAXM    100010
4  struct edge {
5      int v;
6      edge *n;
7      edge(void):v(0),n(NULL) {}
8      edge(int vv, edge *nn):v(vv),n(nn) {}
9  };
10 int nE;
11 edge E[MAXM<<1];
12 edge *ori[MAXN];
13 edge *inv[MAXN];
14 void add_edge(edge *front[], int u, int v) {
15     int ne = ++nE;
16     E[ne] = edge(v, u[front]);
17     u[front] = &(E[ne]);
18 }
19 void connect(int u, int v) {
20     add_edge(ori, u, v);
21     add_edge(inv, v, u);
22 }
23
24 int vis[MAXN];
25 int vst[MAXN];
26 void first_dfs(int u, int &sig) {
27     vis[u] = 1;
28     for(edge *e = u[ori]; e; e = e->n)
29         if(!vis[e->v])
30             first_dfs(e->v, sig);
31     vst[++sig] = u;
32 }
33 int mark[MAXN];
34 void second_dfs(int u, int sig) {
35     vis[u] = 1;
36     mark[u] = sig;
37     for(edge *e = u[inv]; e; e = e->n)
38         if(!vis[e->v])
39             second_dfs(e->v, sig);
40 }
41

```

```

42 int N, M;
43
44 int kosaraju(void) {
45     int i;
46     int sig = 0;
47     for(i = 0; i <= N; ++i) vis[i] = 0;
48     for(i = 1; i <= N; ++i) {
49         if(!vis[i])
50             first_dfs(i, sig);
51     }
52     sig = 1;
53     for(i = 0; i <= N; ++i) vis[i] = 0;
54     for(i = N; i > 0; --i) {
55         if(!vis[vst[i]])
56             second_dfs(vst[i], sig++);
57     }
58     for(i = 1; i <= N; ++i)
59         if(mark[i] != 1)
60             return 0;
61     return 1;
62 }
63
64
65 void clear(void) {
66     nE = 0;
67     for(int i = 0; i <= N; ++i) {
68         ori[i] = inv[i] = NULL;
69     }
70 }
71
72 /* Tarjan */
73 #define MAXN    10010
74 #define MAXM    100010
75 struct edge {
76     int v;
77     edge *n;
78     edge(void):v(0),n(NULL) {}
79     edge(int vv, edge *nn):v(vv),n(nn) {}
80 };
81 typedef edge *ep;
82 int nE;
83 edge E[MAXM];
84 edge *front[MAXN];
85 void add_edge(int u, int v) {
86     int ne = ++nE;
87     E[ne] = edge(v, u[front]);
88     u[front] = &(E[ne]);
89 }
90
91 int mark[MAXN];
92 int dfn[MAXN], low[MAXN];
93 int stk[MAXN];
94 int stk_top;
95
96 void tardfs(int u, int stamp, int &scc) {
97     mark[u] = 1;
98     dfn[u] = low[u] = stamp;
99     stk[stk_top++] = u;
100     for(ep e = u[front]; e; e = e->n) {
101         if(0 == mark[e->v]) tardfs(e->v, ++stamp, scc);
102         if(1 == mark[e->v]) low[u] = min(low[u], low[e->v]);
103     }
104     if(dfn[u] == low[u]) {
105         ++scc;

```

```

106     do {
107         low[stk[stk_top-1]] = scc;
108         mark[stk[stk_top-1]] = 2;
109     } while(stk[(stk_top-)-1] != u);
110 }
111 }
112
113 int tarjan(int n) {
114     int scc = 0, lay = 1;
115     for(int i = 1; i <= n; ++i)
116         if(0 == mark[i])
117             tardfs(i, lay, scc);
118     return scc;
119 }
120
121 int N, M;
122
123 void clear(void) {
124     nE = 0;
125     for(int i = 0; i <= N; ++i) {
126         i[front] = NULL;
127         mark[i] = low[i] = 0;
128     }
129     stk_top = 0;
130 }
131
132 /* Garbow */
133 #define MAXN    10010
134 #define MAXM    100010
135
136 struct edge {
137     int v;
138     edge *n;
139     edge(void):v(0),n(NULL) {}
140     edge(int vv, edge *nn):v(vv),n(nn) {}
141 };
142 typedef edge *ep;
143
144 int nE;
145 edge E[MAXM];
146 edge *front[MAXN];
147 void add_edge(int u, int v) {
148     int ne = ++nE;
149     E[ne] = edge(v, u[front]);
150     u[front] = &(E[ne]);
151 }
152
153 int stk1[MAXN], stk1t;
154 int stk2[MAXN], stk2t;
155 int low[MAXN], belg[MAXN];
156
157 void garbowdfs(int u, int lay, int &scc) {
158     stk1[++stk1t] = u;
159     stk2[++stk2t] = u;
160     low[u] = ++lay;
161     for(ep e=u[front]; e; e = e->n) {
162         if(!low[e->v]) garbowdfs(e->v, lay, scc);
163         else if (0 == belg[e->v])
164             while(low[stk2[stk2t]] > low[e->v])
165                 --stk2t;
166     }
167     if(stk2[stk2t] == u) {
168         stk2t--;
169         scc++;

```

```

170     do {
171         belg[stk1[stk1t]] = scc;
172     } while(stk1[stk1t--] != u);
173 }
174 }
175
176 int grabow(int n) {
177     int i;
178     int scc = 0, lay = 0;
179     for(i = 0; i <= n; ++i) {
180         belg[i] = low[i] = 0;
181     }
182     for(i = 1; i <= n; ++i)
183         if(0 == low[i])
184             garbowdfs(i, lay, scc);
185     return scc;
186 }
187
188 int N, M;
189
190 void clear(void) {
191     nE = 0;
192     for(int i = 0; i <= N; ++i) {
193         front[i] = NULL;
194     }
195 }

```

#### 4.8 Perfect elimination ordering

求弦图的最大团数/最小色数的时候，只要在完美消除序列上从后往前贪心染色即可。  
而求最大独立集/最小团覆盖的时候，只要在完美消除序列上从前往后贪心取点即可。

```

1  /**
2   * BZOJ 1006
3   * [HNOI2008] 神奇的国度
4   * 最大势法求完美消除序列
5   * by Abreto<m@abreto.net>.
6   */
7  #include <cassert>
8  #include <cstdio>
9  #include <vector>
10 #include <bitset>
11 #include <algorithm>
12
13 using namespace std;
14 typedef vector<int> vi;
15 typedef vi::iterator vii;
16 #define pb push_back
17 #define MAXN 10100
18 #define MAXM 1000100
19
20 int n;
21 vi g[MAXN];
22 int ans;
23
24 struct node_t {
25     int v;
26     node_t *nxt;
27 } node[MAXM << 2];
28 int used;
29 node_t *new_node(void) {
30     return node + (used ++);
31 }

```

```

32
33 node_t *f[MAXN]; /* head */
34 void lkto(int pos, int item) {
35     node_t *t = new_node();
36     t->v = item;
37     t->nxt = f[pos];
38     f[pos] = t;
39 }
40
41 int usedby[MAXN];
42 int color[MAXN];
43 bitset<MAXN> added;
44 int label[MAXN], max_label;
45 void mcs(void) {
46     for (int i = 1; i <= n; i++) lkto(0, i);
47     for (int i = n; i > 0; i--) {
48         node_t *cur = f[max_label];
49         assert(cur != NULL);
50         while (added.test(cur->v)) { /* already added */
51             cur = cur->nxt;
52             while (NULL == cur)
53                 cur = f[ --max_label ];
54         }
55         f[ max_label ] = cur->nxt;
56         while (max_label && NULL == f[max_label]) max_label--;
57         int u = cur->v;
58         added.set(u);
59         /* the i-th is u */
60         for (vii it = g[u].begin(); it != g[u].end(); it++) {
61             int v = *it;
62             if (!added.test(v)) {
63                 label[v] ++;
64                 max_label = max(max_label, label[v]);
65                 lkto(label[v], v);
66             }
67             usedby[color[v]] = i;
68         }
69         for (int j = 1; j <= n; j++)
70             if (usedby[j] != i) {
71                 color[u] = j;
72                 break;
73             }
74         ans = max(ans, color[u]);
75     }
76 }
77 }
78
79 int main(void) {
80     int m;
81     scanf("%d%d", &n, &m);
82     while (m--) {
83         int ai, bi;
84         scanf("%d%d", &ai, &bi);
85         g[ai].pb(bi);
86         g[bi].pb(ai);
87     }
88     mcs();
89     printf("%d\n", ans);
90     return 0;
91 }

```

## 5 Math

### 5.1 Euler Function

```

1  /* Euler function phi(x), by Abreto<m@abreto.net>. */
2
3  #define MAXX    3000000
4
5  int phi[MAXX];
6  void get_euler(void) {
7      int i = 0, j = 0;
8      phi[1] = 1;
9      for(i = 2; i < MAXX; ++i)
10         if(!phi[i])
11             for(j = i; j < MAXX; j += i) {
12                 if(!phi[j]) phi[j] = j;
13                 phi[j] = phi[j]/i * (i-1);
14             }
15 }
```

### 5.2 Möbius Function

```

1  void sieve() {
2      fill(isPrime, isPrime + maxn, 1);
3      mu[1] = 1, num = 0;
4      for (int i = 2; i < maxn; ++i) {
5          if (isPrime[i]) primes[num++] = i, mu[i] = -1;
6          static int d;
7          for (int j = 0; j < num && (d = i * primes[j]) < maxn; ++j) {
8              isPrime[d] = false;
9              if (i % primes[j] == 0) {
10                  mu[d] = 0;
11                  break;
12              } else mu[d] = -mu[i];
13          }
14      }
15 }
```

### 5.3 Number Theory Inverse

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int n=10000000;      /* */
5  const long long mod=1e9+7; /* prime required. */
6
7  long long fact[n],fiv[n],inv[n];
8
9  int main() {
10     fact[0]=fact[1]=1;
11     fiv[0]=fiv[1]=1;
12     inv[1]=1;
13     for (int i=2; i<n; i++) {
14         fact[i]=fact[i-1]*i%mod;
15         inv[i]=(mod-mod/i)*inv[mod%i]%mod;
16         fiv[i]=inv[i]*fiv[i-1]%mod;
17     }
18     for (int i=1; i<n; i++) {
19         if (fact[i]*fiv[i]%mod!=1) printf("fact_wrong:%d\n",i);
20         if (inv[i]*i%mod!=1)      printf("inv_wrong:%d\n",i);
21     }
```



```

22 | cout<<"complete"<<endl;
23 | return 0;
24 | }

```

## 5.4 Chinese Remainder Theorem

$$x \equiv a_i \pmod{m_i}$$

```

1 | /* Chinese Remainder Theorem, by Abreto<m@abreto.net>. */
2 | #include "euler.c"
3 |
4 | #define MAXN    64
5 |
6 | typedef long long int ll;
7 |
8 | ll quickpow(ll a, ll b, ll mod) {
9 |     ll ret = 1, base = a;
10 |    while(b > 0) {
11 |        if(b & 1) ret = (ret * base) % mod;
12 |        base = (base * base) % mod;
13 |        b >>= 1;
14 |    }
15 |    return ret;
16 | }
17 |
18 | ll N;
19 | ll a[MAXN], m[MAXN]; /* a and m is indexed from 0. */
20 | ll x, M;
21 |
22 | void naive_crt(void) {
23 |     int i = 0;
24 |     ll Mi[MAXN], nMi[MAXN];
25 |     ll t[MAXN];
26 |
27 |     M = 1;
28 |     for(i = 0; i < N; ++i)
29 |         M *= a[i];
30 |     for(i = 0; i < N; ++i)
31 |         Mi[i] = M / a[i];
32 |     get_euler();
33 |     for(i = 0; i < N; ++i)
34 |         nMi[i] = quickpow(Mi[i], phi[a[i]]-1, a[i]);
35 |     for(i = 0; i < N; ++i)
36 |         t[i] = ((a[i] * Mi[i]) % M) * nMi[i] % M;
37 |     for(i = 0; i < N; ++i)
38 |         x = (x + t[i]) % M;
39 | }

```

## 5.5 Linear congruences

```

1 | #include <cstdio>
2 | #include <cassert>
3 | #include <cstdlib>
4 |
5 | using namespace std;
6 |
7 | class mod_equ_resolver {
8 |     typedef long long int ll;
9 |     ll a, m;
10 |    inline void gurantee(void) {

```

```

11     if ( a < 0 ) {
12         ll k = ( -a ) / m;
13         a += ( k + 1ll ) * m;
14         a = ( a + m ) % m;
15     } else {
16         a %= m;
17     }
18     // printf("x = %lld (mod %lld)\n",a, m);
19 }
20 public:
21     mod_equ_resolver(void) {
22         a = 0ll;
23         m = 1ll;
24     }
25     ll exgcd(ll m, ll n, ll &x, ll &y) {
26         if ( 0 == n ) {
27             x = 1;
28             y = 0;
29             return m;
30         }
31         ll g = exgcd( n, m % n, x, y );
32         ll t = x;
33         x = y;
34         y = t - m / n * y;
35         return g;
36     }
37     int onemore(ll a2, ll m2) {
38         ll x, y;
39         ll g = exgcd( m, m2, x, y );
40         assert(x*m+y*m2==g);
41         a2 = (a2 + m2) % m2;
42         if ( abs( a2 - a ) % g ) return -1;
43         ll newm = m / g * m2;
44         ll newa = a + ( a2 - a ) / g * x * m;
45         a = newa;
46         m = newm;
47         gurantee();
48         return 0;
49     }
50     ll resolve(void) {
51         return a;
52     }
53 };

```

Usage: For

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \vdots \\ x \equiv a_k \pmod{m_k} \end{cases}$$

run

```

1 mod_equ_resolver solver;
2 for (int i = 1; i <= k; i++)
3     solver.onemore(a[i], m[i]);

```

then the solution is

$$x \equiv \text{solver.a} \pmod{\text{solver.m}}$$

## 5.6 FFT

```

1 #include <cmath>
2 using namespace std;

```

```

3 namespace fft {
4 #define eps (1e-9)
5 template < typename T = double >
6 struct dbl {
7     T x;
8     dbl(void):x(0.0) {}
9     template <typename U>
10    dbl(U a):x((T)a) {}
11    inline char sgn(void) {
12        return ((x>=eps)&&(x<=eps))?(0):((x>eps)?(1):(-1));
13    }
14    inline T tabs(void) {
15        return ((x>=eps)&&(x<=eps))?(0.0):((x>eps)?(x):(-x));
16    }
17    inline dbl abs(void) {
18        return dbl(tabs());
19    }
20    template <typename U> inline dbl &operator=(const U b) {
21        x=(T)b;
22        return (*this);
23    }
24    inline T *operator&(void) {
25        return &x;
26    }
27    inline dbl operator-(void) const {
28        return dbl(-x);
29    }
30    inline dbl operator+(const dbl &b) const {
31        return dbl(x+b.x);
32    }
33    inline dbl operator-(const dbl &b) const {
34        return dbl(x-b.x);
35    }
36    inline dbl operator*(const dbl &b) const {
37        return dbl(x*b.x);
38    }
39    inline dbl operator/(const dbl &b) const {
40        return dbl(x/b.x);
41    }
42    template <typename U> inline dbl operator^(const U &b) const {
43        T ret=1.0,base=x;
44        while(b) {
45            if(b&1)ret*=base;
46            base*=base;
47            b>>=1;
48        }
49        return dbl(ret);
50    }
51    inline dbl operator+=(const dbl &b) {
52        return dbl(x+=b.x);
53    }
54    inline dbl operator-=(const dbl &b) {
55        return dbl(x-=b.x);
56    }
57    inline dbl operator*=(const dbl &b) {
58        return dbl(x*=b.x);
59    }
60    inline dbl operator/=(const dbl &b) {
61        return dbl(x/=b.x);
62    }
63    template <typename U> inline dbl operator^=(const U &b) {
64        dbl tmp=(*this)^b;
65        *this=tmp;
66        return tmp;

```

```

67     }
68     inline bool operator==(const dbl &b) const {
69         return (0 == ((*this)-b).sgn());
70     }
71     inline bool operator!=(const dbl &b) const {
72         return (0 != ((*this)-b).sgn());
73     }
74     inline bool operator<(const dbl &b) const {
75         return (-1 == ((*this)-b).sgn());
76     }
77     inline bool operator<=(const dbl &b) const {
78         return (((*this)==b) || ((*this)<b));
79     }
80     inline bool operator>(const dbl &b) const {
81         return (b < (*this));
82     }
83     inline bool operator>=(const dbl &b) const {
84         return (((*this)==b) || ((*this)>b));
85     }
86     template <typename U> inline operator U() const {
87         return (U)x;
88     }
89     inline char operator[](unsigned n) {
90         if(n >= 0) {
91             long long int ret=x;
92             while(n--) {
93                 ret/=10;
94             }
95             return (ret%10);
96         } else {
97             T ret=x;
98             n=-n;
99             while(n--)ret*=10.0;
100            return ((long long int)ret)%10;
101        }
102    }
103 };
104 template <typename T>
105 struct Complex {
106     T x,y; /* x + iy */
107     Complex(void):x(T()),y(T()) {}
108     Complex(T xx):x(xx) {}
109     Complex(T xx,T yy):x(xx),y(yy) {}
110     inline Complex operator-(void) const {
111         return Complex(-x,-y);
112     }
113     inline Complex operator+(const Complex& b) const {
114         return Complex(x+b.x,y+b.y);
115     }
116     inline Complex operator-(const Complex& b) const {
117         return Complex(x-b.x,y-b.y);
118     }
119     inline Complex operator*(const Complex& b) const {
120         return Complex(x*b.x-y*b.y,x*b.y+y*b.x);
121     }
122     inline Complex operator/(const Complex& b) const {
123         T bo=b.x*b.x+b.y*b.y;
124         return Complex((x*b.x+y*b.y)/bo,(y*b.x-x*b.y)/bo);
125     }
126     inline Complex& operator+=(const Complex& b) {
127         Complex tmp=(*this)+b;
128         (*this)=tmp;
129         return (*this);
130     }

```

```

131 inline Complex& operator--=(const Complex& b) {
132     Complex tmp=(*this)-b;
133     (*this)=tmp;
134     return (*this);
135 }
136 inline Complex& operator*=(const Complex& b) {
137     Complex tmp=(*this)*b;
138     (*this)=tmp;
139     return (*this);
140 }
141 inline Complex& operator/=(const Complex& b) {
142     Complex tmp=(*this)/b;
143     (*this)=tmp;
144     return (*this);
145 }
146 inline friend Complex operator+(const T& a, const Complex& b) {
147     return Complex(a)+b;
148 }
149 inline friend Complex operator-(const T& a, const Complex& b) {
150     return Complex(a)-b;
151 }
152 inline friend Complex operator*(const T& a, const Complex& b) {
153     return Complex(a)*b;
154 }
155 inline friend Complex operator/(const T& a, const Complex& b) {
156     return Complex(a)/b;
157 }
158 };
159 typedef dbl<> Double;
160 typedef Complex<Double> ComplexD;
161 typedef long long int ll;
162 const int maxn = 2000000; /* !! */
163 const Double pi(acos(-1.0));
164
165 void build(ComplexD _P[], ComplexD P[], int n, int m, int curr, int &cnt) {
166     if(m == n) {
167         _P[curr] = P[cnt++];
168     } else {
169         build(_P, P, n, m*2, curr, cnt);
170         build(_P, P, n, m*2, curr+m, cnt);
171     }
172 }
173
174 void FFT(ComplexD P[], int n, int oper) { /* n should be 2^k. */
175     static ComplexD _P[maxn];
176     int cnt = 0;
177     build(_P, P, n, 1, 0, cnt);
178     copy(_P, _P+n, P);
179     for(int d = 0; (1<<d)<n; ++d) {
180         int m = 1<<d;
181         int m2 = m*2;
182         Double p0 = pi / m * oper;
183         ComplexD unit_p0(cos(p0.x), sin(p0.x));
184         for(int i = 0; i < n; i += m2) {
185             ComplexD unit(1,0);
186             for(int j = 0; j < m; ++j) {
187                 ComplexD &P1 = P[i+j+m], &P2 = P[i+j];
188                 ComplexD t = unit * P1;
189                 P1 = P2 - t;
190                 P2 = P2 + t;
191                 unit *= unit_p0;
192             }
193         }
194     }

```

```

195     if(-1 == oper) {
196         for(int i = 0; i < n; ++i)
197             P[i] /= Double(n);
198     }
199 }
200 }

```

## 5.7 NTT

```

1  #include<bits/stdc++.h>
2  #define ll long long
3  const int N=262144;
4  const ll MOD=500000000001507329LL;//998244353 1004535809
5  using namespace std;
6  int n,m;
7  ll a[N],b[N],x[N],y[N];
8  ll wn[25];
9  ll Mul(ll x,ll y) { //乘法超ll用快速乘，主函数也需要用
10     ll ans=(x*y-(ll)((long double)x/MOD*y+1e-8)*MOD);
11     return ans<0?ans+MOD:ans;
12 }
13 ll Qpow(ll a,ll b,ll M) {
14     ll ans=1;
15     a%=M;
16     while(b) {
17         if(b&1) ans=Mul(ans,a);
18         a=Mul(a,a);
19         b>>=1;
20     }
21     return ans;
22 }
23 void Getwn() { //主函数预处理getwn()
24     for(int i=0; i<25; i++) {
25         wn[i]=Qpow(3,(MOD-1)/(1<<i),MOD);
26     }
27 }
28 void NTT(ll *x,int n,int rev) {
29     int i,j,k,ds;
30     ll w,u,v;
31     for(i=1,j=n>>1,k=n>>1; i<n-1; i++,k=n>>1) {
32         if(i<j) swap(x[i],x[j]);
33         while(j>=k) j-=k,k>>=1;
34         if(j<k) j+=k;
35     }
36     for(i=2,ds=1; i<=n; i<=<=1,ds++) {
37         for(j=0; j<n; j+=i) {
38             w=1;
39             for(k=j; k<j+i/2; k++) {
40                 u=x[k];
41                 v=Mul(w,x[k+i/2]);
42                 x[k]=(u+v)%MOD;
43                 x[k+i/2]=(u-v+MOD)%MOD;
44                 w=Mul(w,wn[ds]);
45             }
46         }
47     }
48     if(rev==1) {
49         for(i=1; i<n/2; i++) swap(x[i],x[n-i]);
50         w=Qpow(n,MOD-2,MOD);
51         for(i=0; i<n; i++) x[i]=Mul(x[i],w);
52     }
53 }

```

```

54 int main() {
55     Getwn();
56     while(~scanf("%d%d",&n,&m)) {
57         for(int i=0; i<n; i++)scanf("%lld",&a[i]);
58         for(int i=0; i<m; i++)scanf("%lld",&b[i]);
59         int len=1,s=n+m;
60         while(len<s)len<=1;
61         for(int i=n; i<len; i++)a[i]=0;
62         for(int i=m; i<len; i++)b[i]=0;
63         NTT(a,len,1);
64         NTT(b,len,1);
65         for(int i=0; i<len; i++)a[i]=Mul(a[i],b[i]);
66         NTT(a,len,-1);
67         for(int i=0; i<=s; i++)printf("%lld_",a[i]);
68         puts("");
69     }
70 }
71
72 // -----
73 #include<cstdio>
74 #include<iostream>
75 #include<cstring>
76 #include<cmath>
77 #include<complex>
78 using namespace std;
79 typedef long long LL;
80 const LL MOD=998244353,g=3,gi=332748118;
81 const LL N=1000005;
82 LL n,m;
83 LL a[N],b[N];
84 LL pow(LL x,LL y) {
85     if (y==1) return x;
86     LL lalal=pow(x,y>>1);
87     lalal=lalal*lalal%MOD;
88     if (y&1) lalal=lalal*x%MOD;
89     return lalal;
90 }
91 void ntt(LL *a,LL n,LL o) {
92     if (n==1) return ;
93     LL k=(n>>1);
94     LL w=1,wn=pow(o==1?g:gi,(MOD-1)/n),a0[k],a1[k];
95     for (LL u=0; u<k; u++) {
96         LL i=u*2;
97         a0[u]=a[i];
98         a1[u]=a[i+1];
99     }
100     ntt(a0,k,o);
101     ntt(a1,k,o);
102     for (LL u=0; u<k; u++) {
103         a[u]=a0[u]+w*a1[u]%MOD;
104         a[u]=(a[u]%MOD+MOD)%MOD;
105         a[u+k]=a0[u]-w*a1[u];
106         a[u+k]=(a[u+k]%MOD+MOD)%MOD;
107         w=w*wn%MOD;
108     }
109 }
110 void ntt(LL *a,LL n,LL op) {
111     for (LL u=0; u<n; u++) bin[u]=(bin[u>>1]>>1)|((u&1)*(n>>1));
112     for (LL u=0; u<n; u++) if (u<bin[u]) swap(a[u],a[bin[u]]);
113     for (LL u=1; u<n; u<=1) {
114         LL wn=pow(op==1?g:gi,(MOD-1)/(u<<1)),w,t;
115         for (LL i=0; i<n; i=i+(u<<1)) {
116             w=1;
117             for (LL k=0; k<u; k++) {

```

```

118     t=w*a[u+i+k]%MOD;
119     a[u+i+k]=(a[i+k]-t+MOD)%MOD;
120     a[i+k]=(a[i+k]+t)%MOD;
121     w=w*wn%MOD;
122 }
123 }
124 }
125 if(op==1) {
126     LL Inv=pow(n,MOD-2);
127     for(LL i=0; i<n; i++) a[i]=a[i]*Inv%MOD;
128 }
129 }
130
131 int main() {
132     scanf("%I64d%I64d",&n,&m);
133     for (LL u=0; u<=n; u++) scanf("%I64d",&a[u]);
134     for (LL u=0; u<=m; u++) scanf("%I64d",&b[u]);
135     m=m+n;
136     n=1;
137     while (n<=m) n<<=1;
138     ntt(a,n,1);
139     ntt(b,n,1);
140     for (LL u=0; u<=n; u++) a[u]*=b[u];
141     ntt(a,n,-1);
142     LL inv=pow(n,MOD-2);
143     for (LL u=0; u<=m; u++) printf("%I64d",a[u]*inv%MOD);
144     return 0;
145 }

```

## 5.8 Fast Walsh–Hadamard transform

- 异或

$$\mathcal{F}\{A\} = [\mathcal{F}\{A_0\} + \mathcal{F}\{A_1\}, \mathcal{F}\{A_0\} - \mathcal{F}\{A_1\}]$$

$$\mathcal{F}^{-1}\{A\} = \left[ \mathcal{F}^{-1}\left\{\frac{A_0 + A_1}{2}\right\}, \mathcal{F}^{-1}\left\{\frac{A_0 - A_1}{2}\right\} \right]$$

- 按位与

$$\mathcal{F}\{A\} = [\mathcal{F}\{A_0\} + \mathcal{F}\{A_1\}, \mathcal{F}\{A_1\}]$$

$$\mathcal{F}^{-1}\{A\} = [\mathcal{F}^{-1}\{A_0\} - \mathcal{F}^{-1}\{A_1\}, \mathcal{F}^{-1}\{A_1\}]$$

- 按位或

$$\mathcal{F}\{A\} = [\mathcal{F}\{A_0\}, \mathcal{F}\{A_1\} + \mathcal{F}\{A_0\}]$$

$$\mathcal{F}^{-1}\{A\} = [\mathcal{F}^{-1}\{A_0\}, \mathcal{F}^{-1}\{A_1\} - \mathcal{F}^{-1}\{A_0\}]$$

```

1 void FWT(int a[],int n) {
2     for(int d=1; d<n; d<<=1)
3         for(int m=d<<1,i=0; i<n; i+=m)
4             for(int j=0; j<d; j++) {
5                 int x=a[i+j],y=a[i+j+d];
6                 a[i+j]=(x+y)%mod,a[i+j+d]=(x-y+mod)%mod;
7                 //xor:a[i+j]=x+y,a[i+j+d]=(x-y+mod)%mod;
8                 //and:a[i+j]=x+y;
9                 //or:a[i+j+d]=x+y;
10            }
11 }
12
13 void UFWT(int a[],int n) {
14     for(int d=1; d<n; d<<=1)
15         for(int m=d<<1,i=0; i<n; i+=m)

```



```

16     for(int j=0; j<d; j++) {
17         int x=a[i+j],y=a[i+j+d];
18         a[i+j]=1LL*(x+y)*rev%mod,a[i+j+d]=(1LL*(x-y)*rev%mod+mod)%mod;
19         //xor:a[i+j]=(x+y)/2,a[i+j+d]=(x-y)/2;
20         //and:a[i+j]=x-y;
21         //or:a[i+j+d]=y-x;
22     }
23 }
24 void solve(int a[],int b[],int n) {
25     FWT(a,n);
26     FWT(b,n);
27     for(int i=0; i<n; i++) a[i]=1LL*a[i]*b[i]%mod;
28     UFWT(a,n);
29 }

```

## 5.9 Lucas

```

1  /* Lucas, by Abreto<m@abreto.net>. */
2
3  struct __lucas {
4      static const int maxp = 1000000;
5      typedef long long int ll;
6      int p;
7      int f[maxp]; // fiv[maxp], inv[maxp];
8      inline int mul(const int a, const int b) {
9          ll z = 1ll * a * b;
10         z -= z / p * p;
11         return z;
12     }
13     int qow(int a, int x) {
14         int ret = 1;
15         while (x) {
16             if (1 & x) ret = mul(ret, a);
17             a = mul(a, a);
18             x >>= 1;
19         }
20         return ret;
21     }
22     void init(int np) {
23         p = np;
24         // return; // uncomment this line if use binom()
25         f[0] = f[1] = 1;
26         // fiv[0] = fiv[1] = 1;
27         // inv[1] = 1;
28         for (int i = 2; i < p; i++) {
29             f[i] = mul(f[i - 1], i);
30             // inv[i] = mul(p - p / i, inv[p % i]);
31             // fiv[i] = mul(fiv[i - 1], inv[i]);
32         }
33     }
34     int C(int n, int k) {
35         if (n < k) return 0;
36         return mul(f[n], qow(mul(f[k], f[n - k]), p - 2));
37     }
38     /** use following if get TLE { */
39     int binom(int n, int k) {
40         if (n < k) return 0;
41         if (k > n - k) k = n - k;
42         int a = 1, b = 1;
43         while (k) {
44             a = mul(a, n);
45             b = mul(b, k);

```

```

46     n--;
47     k--;
48 }
49 return mul(a, qow(b, p - 2));
50 }
51 /** } —— */
52 int operator()(int n, int k) {
53     if (0 == k) return 1;
54     if (n < p && k < p) return C(n, k);
55     return mul(C(n % p, k % p), (*this)(n / p, k / p));
56 }
57 } lucas;

```

## 5.10 Linear Programming

```

1  /* 线性规划 */
2  #include<bits/stdc++.h>
3
4  using namespace std;
5  const int Maxn=110,Maxm=59;
6  class Simplex {
7      /*
8       功能:
9       接受有n个约束, m个基本变量的方程组a[0~n][0~m]
10      a[0][]存放需要最大化的目标函数, a[][0]存放常数
11      Base[]存放基本变量的id,初始为1~m
12      Rest[]存放松弛变量的id,初始为m+1~m+n
13      返回此线性规划的最小值ans
14      要求方案的话, Base[]中的变量值为0,Rest[]中的变量值为相应行的[0]
15      如果solve
16      返回1,说明运行正常ans是它的最大值
17      返回0,说明无可行解
18      返回-1,说明解没有最大值
19      测试:
20      m=2,n=3
21      double a[4][3]={
22          {0,1,3},
23          {8,-1,1},
24          {-3,1,1},
25          {2,1,-4}
26      };
27      solve=1,ans=64/3;
28      注意ac不了可能是eps的问题
29      */
30  public:
31      static const double Inf;
32      static const double eps;
33      int n,m;
34      double a[Maxn][Maxm];
35      int Base[Maxm],Rest[Maxn];
36      double val[Maxm];
37      double ans;
38      void pt() {
39          for(int i=0; i<=n; i++) {
40              for(int j=0; j<=m; j++)printf("%.2f ",a[i][j]);
41              puts("");
42          }
43      }
44      void pivot(int x,int y) { //将第x个非基本变量和第y个基本变量调换
45          swap(Rest[x],Base[y]);
46          double tmp=-1./a[x][y];
47          a[x][y]=-1.;

```

```

48     for(int j=0; j<=m; j++)a[x][j]*=tmp;
49     for(int i=0; i<=n; i++) {
50         if(i==x||fabs(a[i][y])<eps)continue;
51         tmp=a[i][y];
52         a[i][y]=0;
53         for(int j=0; j<=m; j++)a[i][j]+=tmp*a[x][j];
54     }
55 }
56 bool opt() {
57     while(1) {
58         int csi=0;
59         for(int i=1; i<=m; i++)if(a[0][i]>eps&&(!csi||Base[i]<Base[csi]))csi=i;
60         if(!csi)break;
61         int csj=0;
62         double cur;
63         for(int j=1; j<=n; j++) {
64             if(a[j][csi]>=eps)continue;
65             double tmp=-a[j][0]/a[j][csi];
66             if(!csj||tmp+eps<cur||(fabs(tmp-cur)<eps&&Rest[j]<Rest[csj]))csj=j,cur=tmp;
67         }
68         if(!csj)return 0;
69         pivot(csj,csi);
70     }
71     ans=a[0][0];
72     return 1;
73 }
74 bool init() {
75     ans=0;
76     for(int i=1; i<=m; i++)Base[i]=i;
77     for(int i=1; i<=n; i++)Rest[i]=m+i;
78     int cs=1;
79     for(int i=2; i<=n; i++)if(a[i][0]<a[cs][0])cs=i;
80     if(a[cs][0]>=eps)return 1;
81     static double tmp[Maxm];
82     for(int i=0; i<=m; i++)tmp[i]=a[0][i],a[0][i]=0;
83     for(int i=1; i<=n; i++)a[i][m+1]=1.;
84     a[0][m+1]=-1.;
85     Base[m+1]=m+n+1;
86     pivot(cs,++m);
87     opt();
88     m--;
89     if(a[0][0]<=eps)return 0;
90     cs=-1;
91     for(int i=1; i<=n; i++) {
92         if(Rest[i]>m+n) {
93             cs=i;
94             break;
95         }
96     }
97     if(cs>=1) {
98         int nxt=-1;
99         m++;
100         for(int i=1; i<=m; i++)if(a[cs][i]>eps||a[cs][i]<=eps) {
101             nxt=i;
102             break;
103         }
104         pivot(cs,nxt);
105         m--;
106     }
107     for(int i=1; i<=m; i++) {
108         if(Base[i]>m+n) {
109             swap(Base[i],Base[m+1]);
110             for(int j=0; j<=n; j++)a[j][i]=a[j][m+1];
111             break;

```

```

112     }
113 }
114 for(int i=1; i<=m; i++)a[0][i]=0;
115 a[0][0]=tmp[0];
116 for(int i=1; i<=m; i++)if(Base[i]<=m)a[0][i]=tmp[Base[i]];
117 for(int i=1; i<=n; i++) {
118     if(Rest[i]<=m) {
119         for(int j=0; j<=m; j++)a[0][j]+=tmp[Rest[i]]*a[i][j];
120     }
121 }
122 return 1;
123 }
124 void getval() {
125     for(int i=1; i<=m; i++)val[i]=0;
126     for(int i=1; i<=n; i++)if(Rest[i]<=m)val[Rest[i]]=a[i][0];
127     //for(int i=1; i<=m; i++)printf("%.2f ",val[i]);puts("");
128 }
129 int solve() {
130     if(!init())return 0;
131     if(!opt())return -1;
132     getval();
133     return 1;
134 }
135 } solver;
136 const double Simplex:: Inf=1e80;
137 const double Simplex:: eps=1e-8;
138 int main() {
139     int m,n,type;
140     scanf("%d%d%d",&m,&n,&type);
141     solver.a[0][0]=0;
142     for(int i=1; i<=m; i++)scanf("%lf",&solver.a[0][i]);
143     for(int i=1; i<=n; i++) {
144         for(int j=1; j<=m+1; j++) {
145             if(j==m+1)scanf("%lf",&solver.a[i][0]);
146             else {
147                 scanf("%lf",&solver.a[i][j]);
148                 solver.a[i][j]=-solver.a[i][j];
149             }
150         }
151     }
152     solver.m=m,solver.n=n;
153     int rep=solver.solve();
154     if(rep==0)puts("Infeasible");
155     else if(rep==-1)puts("Unbounded");
156     else {
157         printf("%.12f\n",solver.ans);
158         if(type==1) {
159             for(int i=1; i<=m; i++)printf("%.12f%c",solver.val[i],i==m?'\\n':' ');
160         }
161     }
162 }

```

## 5.11 Big Prime Test

```

1 #include <iostream>
2 #include <cstdlib>
3 using namespace std;
4 typedef long long LL;
5 LL minfactor, p[11] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
6 LL gcd(LL a, LL b) {
7     return b ? gcd(b, a % b) : a;
8 }

```

```

9  LL qmult(LL a, LL b, LL mod) { // 快速乘模
10     LL sum = 0;
11     while (b) {
12         if (b & 1) {
13             sum += a;
14             if (sum >= mod) sum -= mod; // 此处无需用%, %运算比减法慢很多
15         }
16         b >>= 1, a <<= 1;
17         if (a >= mod) a -= mod;
18     }
19     return sum;
20 }
21 LL qpow(LL a, LL b, LL mod) { // 快速幂模
22     LL res = 1;
23     while (b) {
24         if (b & 1) res = qmult(res, a, mod);
25         b >>= 1;
26         a = qmult(a, a, mod);
27     }
28     return res;
29 }
30 bool prime_test(LL n, LL a) { // 对整数n, 底数a进行测试, 返回true表示通过测试
31     LL p = qpow(a, n - 1, n);
32     if (p != 1) return false;
33     else { // 二次探测
34         LL s = n - 1;
35         while (!(s & 1) && p == 1) {
36             s >>= 1;
37             p = qpow(a, s, n);
38         }
39         if (p == 1 || p == n - 1) return true;
40         else return false;
41     }
42 }
43 bool Miller_Rabin(LL n) { // 对整数n进行Miller_Rabin素数测试, 返回true表示通过测试
44     if (n <= 29) { // if这一块其实可以不用
45         for (int i = 0; i < 10; i++) {
46             if (n == p[i]) return true;
47         }
48         return false;
49     }
50     for (int i = 0; i < 10; i++) { // 利用前10个素数作为底数测试的正确率已经非常高
51         if (gcd(n, p[i]) == 1 && !prime_test(n, p[i])) return false;
52     }
53     return true;
54 }
55 LL randf(LL x, LL n, LL c) { // 满足要求的产生伪随机数函数
56     return (qmult(x, x, n) + c) % n;
57 }
58 LL pollard_rho(LL n, LL c) { // 查找n的因数, c为上面函数要用的随机数, c也可自己指定
    (但要有变化)
59     LL x = rand() % n, y = x, i = 1, k = 2, p; // 随机生成随机数的初始值, 也可自己指定
60     while (true) {
61         i++;
62         x = randf(x, n, c);
63         p = gcd(y - x + n, n);
64         if (p > 1 && p < n) return p;
65         if (y == x) return n; // 判圈, 返回n表示查找失败, 要更新随机种子重新查找
66         if (i == k) {
67             y = x; // 更新范围和记录的数
68             k <<= 1;
69         }
70     }
71 }

```

```

72 void find_factor(LL n) { // 查找所有因数
73     if (Miller_Rabin(n)) {
74         minfactor = min(minfactor, n);
75         return ;
76     }
77     LL p = n;
78     while (p == n) p = pollard_rho(n, rand() % (n - 1) + 1); // 查找失败则更新随机种子
79     // 重新查找, 直到找到因子
80     find_factor(p); // 递归查找更小因子
81     find_factor(n / p);
82 }
83 int main() {
84     int t;
85     cin >> t;
86     while (t--) {
87         LL N;
88         cin >> N;
89         if (Miller_Rabin(N)) cout << "Prime" << endl;
90         else {
91             minfactor = N;
92             find_factor(N);
93             cout << minfactor << endl;
94         }
95     }
96     return 0;
97 }

```

### 5.11.1 Miller Rabin

```

1  /* Miller-Rabin Prime Test, by Abreto<m@abreto.net>. */
2
3  namespace miller_rabin {
4
5  typedef long long int ll;
6
7  inline ll add(const ll a, const ll b, const ll mod) {
8      ll z = a + b;
9      if (z >= mod) z -= mod;
10     return z;
11 }
12 inline ll mul(ll a, ll b, const ll mod) {
13     ll z = 0;
14     if (a >= mod) a %= mod;
15     if (b >= mod) b %= mod;
16     while (b) {
17         if (1 & b) z = add(z, a, mod);
18         a = add(a, a, mod);
19         b >>= 1;
20     }
21     return z;
22 }
23
24 ll qow(ll a, ll x, ll mod) {
25     ll ret = 1ll;
26     while (x) {
27         if (1 & x) ret = mul(ret, a, mod);
28         a = mul(a, a, mod);
29         x >>= 1;
30     }
31     return ret;
32 }
33

```

```

34 const int K = 5;
35 const int p[] = {
36     2, 3, 7, 61, 24251
37 };
38 const ll strong = 4685624825598111;
39 /* 46 856 248 255 981 in (0, 1e16) */
40
41 bool mr(ll n, int k) {
42     ll d = n - 1;
43     int s = 0;
44     while (d > 1 && 0 == (d & 1)) {
45         s++;
46         d >>= 1;
47     }
48     for (int i = 0; i < k; i++) {
49         ll a = (i < K) ? p[i] : (1 + rand() % (n - 1));
50         ll x = qow(a, d, n);
51         for (int j = 0; j < s; j++) {
52             ll xp = mul(x, x, n);
53             if (1 == xp && x != 1 && x != n-1) return false;
54             x = xp;
55         }
56         if (x != 1) return false;
57     }
58     return true;
59 }
60
61 /* 2,3,5,7,11,13 */
62 const int pre[] = {3, 5, 7, 11, 13};
63 bool test(ll n, int k = 5) {
64     if (2 == n) return true;
65     if (0 == (n & 1)) return false;
66     if (strong == n) return false;
67     for (int i = 0; i < 5; i++) {
68         if (n == pre[i]) return true;
69         if (n == n / pre[i] * pre[i])
70             return false;
71     }
72     return mr(n, k);
73 }
74
75 }

```

### 5.11.2 Pollard's rho

```

1  /* Pollard's rho, by Abreto<m@abreto.net>. */
2
3  namespace pollards_rho {
4
5  typedef long long int ll;
6
7  inline ll add(const ll a, const ll b, ll mod) {
8      ll z = a + b;
9      if (z >= mod) z -= mod;
10     return z;
11 }
12 inline ll mul(ll a, ll b, ll mod) {
13     ll z = 0ll;
14     if (a >= mod) a -= a / mod * mod;
15     if (b >= mod) b -= b / mod * mod;
16     while (b) {
17         if (1 & b) z = add(z, a, mod);
18         a = add(a, a, mod);

```

```

19     b >= 1;
20 }
21 return z;
22 }
23
24 ll gcd(ll m, ll n) {
25     return (0 == n) ? m : gcd(n, m % n);
26 }
27
28 ll find(ll n, int c = -1) {
29     ll x = rand() % n;
30     ll y = x, k = 2;
31     for (int i = 2; ; i++) {
32         x = add(mul(x, x, n), (n + c) % n, n);
33         ll d = gcd(y - x + n, n); // change to abs(y - x) if get WA
34         if (1 != d && n != d) return d;
35         if (y == x) return n;
36         if (i == k) {
37             y = x;
38             k <<= 1;
39         }
40     }
41 }
42
43 /** usage:
44  * void find(ll n, int c = 107)
45  * {
46  *     if (1 == n) return;
47  *     if (miller-rabin(n) )
48  *     {
49  *         n is a prime;
50  *         return;
51  *     }
52  *     ll p = n, k = c;
53  *     while (p >= n) p = pollards_rho(p, k--);
54  *     find(p, c);
55  *     find(n/p, c);
56  * }
57  **/
58
59 }

```

## 5.12 Montgomery modular multiplication

```

1  /* — Montgomery modular algorithm { — */
2  struct Mod64 {
3      typedef long long ll;
4      typedef unsigned long long u64;
5      typedef __int128_t i128;
6      typedef __uint128_t u128;
7      Mod64() :n_(0) {}
8      Mod64(u64 n) :n_(init(n)) {}
9      static u64 init(u64 w) {
10         return reduce(u128(w) * r2);
11     }
12 }
13 static void set_mod(u64 m) {
14     mod = m;
15     assert(mod & 1);
16     inv = m;
17     for (int i = 0; i < 5; ++i) inv *= 2 - inv * m;
18     r2 = -u128(m) % m;

```



```

19 }
20 static u64 reduce(u128 x) {
21     u64 y = u64(x >> 64) - u64((u128(u64(x)*inv)*mod) >> 64);
22     return ll(y)<0 ? y + mod : y;
23 }
24 Mod64& operator += (Mod64 rhs) {
25     n_ += rhs.n_ - mod;
26     if (ll(n_)<0) n_ += mod;
27     return *this;
28 }
29 Mod64 operator + (Mod64 rhs) const {
30     return Mod64(*this) += rhs;
31 }
32 Mod64& operator -= (Mod64 rhs) {
33     n_ -= rhs.n_;
34     if (ll(n_)<0) n_ += mod;
35     return *this;
36 }
37 Mod64 operator - (Mod64 rhs) const {
38     return Mod64(*this) -= rhs;
39 }
40 Mod64& operator *= (Mod64 rhs) {
41     n_ = reduce(u128(n_)*rhs.n_);
42     return *this;
43 }
44 Mod64 operator * (Mod64 rhs) const {
45     return Mod64(*this) *= rhs;
46 }
47 u64 get() const {
48     return reduce(n_);
49 }
50 static u64 mod, inv, r2;
51 u64 n_;
52 };
53
54 Mod64::u64 Mod64::mod, Mod64::inv, Mod64::r2;
55 /* — } Montgomery modular algorithm — */
56
57 /**
58  * usage:
59  * First, Mod64::set_mod();
60  * Mod64 a, b, c(init_val);
61  * a = b * c;
62  * printf("%llu\n", a.get());
63  */

```

### 5.13 Berlekamp Massey

```

1  /* Berlekamp Massey by HoldZhu. */
2  #include <cstdio>
3  #include <vector>
4
5  using namespace std;
6
7  namespace BerlekampMassey {
8      const int mod = 1e9 + 7;
9      int L, m, b, n;
10     vector<int> s, C, B;
11     void init() {
12         s.clear();
13         C.clear();
14         B.clear();

```

```

15 C.push_back(1);
16 B.push_back(1);
17 L = n = 0;
18 m = b = 1;
19 }
20 int pow_mod(int a, int k) {
21     int s = 1;
22     while (k) {
23         if (k & 1)
24             s = 1ll * s * a % mod;
25         a = 1ll * a * a % mod;
26         k >>= 1;
27     }
28     return s;
29 }
30 void update(int d) {
31     s.push_back(d);
32     for (int i = 1; i <= L; ++i)
33         d = (d + 1ll * C[i] * s[n - i] % mod) % mod;
34     if (d == 0)
35         ++m;
36     else if (2 * L <= n) {
37         vector<int> T = C;
38         C.resize(n + 1 - L + 1);
39         for (int i = L + 1; i <= n + 1 - L; ++i)
40             C[i] = 0;
41         for (int i = 0; i < B.size(); ++i)
42             C[i + m] = (C[i + m] + mod - 1ll * d * pow_mod(b, mod - 2) % mod * B[i] % mod)
43                 % mod;
44         L = n + 1 - L;
45         B = T;
46         b = d;
47         m = 1;
48     } else {
49         for (int i = 0; i < B.size(); ++i)
50             C[i + m] = (C[i + m] + mod - 1ll * d * pow_mod(b, mod - 2) % mod * B[i] % mod)
51                 % mod;
52         ++m;
53     }
54     ++n;
55 }
56 void output() {
57     printf("F(n)=");
58     for (int i = 1; i < C.size(); ++i) {
59         int output = (mod - C[i]) % mod;
60         if (output > mod / 2)
61             output -= mod;
62         printf("%s%d*F(n-%d)", (output < 0 || i == 1) ? "" : "+", output, i);
63     }
64     puts("");
65 }
66 void output_code_for() {
67     static const char *name = "dp";
68     static const char *index = "i";
69     static const char *upperbound = "maxn";
70     puts("//_Generated_by_Berlekamp-Massey_algorithm");
71     for (int i = 1; i < C.size(); ++i) {
72         printf("%s[%d]=%d;\n", name, i - 1, s[i - 1]);
73     }
74     printf("for(int i=%d;i<%s;++i)\n", (int)C.size() - 1, upperbound);
75     printf("_s[%s]=(", name, index);
76     for (int i = 1; i < C.size(); ++i) {
77         int output = (mod - C[i]) % mod;
78         if (output > mod / 2)

```

```

77     output -= mod;
78     printf("%s%d*s[%s-%d]%%mod", (output < 0 || i == 1) ? "" : "+", output, name,
        index, i);
79 }
80 puts(")%mod+mod)%mod;");
81 }
82 void output_code_matrix() {
83     // TODO
84 }
85 };
86
87 /** usage */
88 int usage() {
89     // int arr[12] = {2, 24, 96, 416, 1536, 5504, 18944, 64000, 212992, 702464,
        2301952, 7512064};
90     int arr[] = {3, 20, 119, 696, 4059, 23660, 137903, 803760, 4684659};
91     BerlekampMassey::init();
92     for (auto ai : arr) {
93         BerlekampMassey::update(ai);
94     }
95     printf("Formule:␣");
96     BerlekampMassey::output();
97     printf("Code:␣\n");
98     BerlekampMassey::output_code_for();
99     return 0;
100 }

```

## 5.14 Inclusion–exclusion principle

### 5.14.1 General form

若  $A_1, A_2, \dots, A_n$  为有限集, 则

$$\left| \bigcup_{i=1}^n A_i \right| = \sum_{\emptyset \neq J \subseteq \{1, 2, \dots, n\}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right|$$

### 5.14.2 A generalization

若

$$g(A) = \sum_{S \subseteq A} f(S)$$

则

$$f(A) = \sum_{S \subseteq A} (-1)^{|A|-|S|} g(S)$$

更一般的, 如果  $S$  是多重集合 (multiset), 那么

$$f(A) = \sum_{S \subseteq A} \mu(A - S) g(S)$$

其中

- 当  $S$  是含有偶数个元素的集合 (没有重复元素) 时,  $\mu(S) = 1$
- 当  $S$  是含有奇数个元素的集合 (没有重复元素) 时,  $\mu(S) = -1$
- 当  $S$  含有重复元素时,  $\mu(S) = 0$ .

### 5.14.3 Applications

#### I. 乱序排列

如果集合  $A$  含有  $n$  个元素, 则乱序排列的数目为  $[n!/e]$ ,  $[x]$  表示最接近  $x$  的整数.

### 5.15 Lindström—Gessel—Viennot lemma

对于一张无边权的 DAG 图, 给定  $n$  个起点和对应的  $n$  个终点, 这  $n$  条不相交路径的方案数为

$$\begin{vmatrix} e(a_1, b_1) & e(a_1, b_2) & \cdots & e(a_1, b_n) \\ e(a_2, b_1) & e(a_2, b_2) & \cdots & e(a_2, b_n) \\ \vdots & \vdots & \ddots & \vdots \\ e(a_n, b_1) & e(a_n, b_2) & \cdots & e(a_n, b_n) \end{vmatrix} \quad (\text{该矩阵的行列式})$$

其中  $e(a, b)$  为图上  $a$  到  $b$  的方案数.

## 6 String

### 6.1 Hash

```

1 /* Common hash for any substrings. */
2
3 typedef unsigned long long int llu;
4 #define MAXN 1000000
5 int n;
6 char s[MAXN];
7 llu H[MAXN], xP[MAXN], P = 9999111;
8 void init(void) {
9     int i = 0;
10    xP[0] = 111;
11    for(i = 1; i < MAXN; ++i) xP[i] = xP[i-1] * P;
12    H[n] = 0;
13    for(i = n-1; i >= 0; --i) H[i] = H[i+1]*P + s[i];
14 }
15 #define HASH(i,l) (H[i] - H[i+l]*xP[l])

```

### 6.2 KMP

```

1 /* KMP, by Abreto<m@abreto.net>. */
2 #include <string.h>
3
4 /* !!NEED IMPROVING!! */
5
6 #define MAXL (1000010)
7
8 char W[MAXL], T[MAXL];
9 int f[MAXL];
10 int lW, lT;
11
12 int count(void) {
13     int cnt = 0;
14     int i, j;
15     lW = strlen(W);
16     lT = strlen(T);
17     // — self-matching
18     f[0] = j = -1;
19     for(i = 1; i < lW; i++) {
20         while( j >= 0 && W[j+1] != W[i] )

```

```

21     j = f[j];
22     if( W[j+1] == W[i] ) j++;
23     f[i] = j;
24 }
25 // ——
26 j = -1;
27 for(i = 0; i < lT; i++) {
28     while( j >= 0 && W[j+1] != T[i] )
29         j = f[j];
30     if( W[j+1] == T[i] ) j++;
31     if( j == lW-1 ) {
32         cnt++;
33         j = f[j];
34     }
35 }
36 return cnt;
37 }

```

### 6.3 exKMP

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  namespace exkmp {
5
6  const int maxn = 1000100, maxm = 1000100;
7
8  int n, m;
9  char S[maxn], T[maxm];
10
11 /* the length of longest prefix between T[i..m-1] and T[0..m-1] */
12 int nxt[maxm];
13 /* the length of longest prefix between S[i..n-1] and T[0..m-1] */
14 int ex[maxn];
15
16 void getsize() {
17     n = strlen(S);
18     m = strlen(T);
19 }
20
21 void self(void) {
22     int q = 1, p = 0;
23     nxt[0] = m;
24     while (1 + p < m && T[1 + p] == T[p]) p++;
25     nxt[1] = p;
26     for (int i = 2; i < m; i++) {
27         int l = nxt[i - q];
28         if (i + l - 1 < p) {
29             nxt[i] = l;
30         } else {
31             int j = max(0, p - i + 1);
32             while (i + j < m && T[i + j] == T[j]) j++;
33             nxt[i] = j;
34             p = i + j - 1;
35             q = i;
36         }
37     }
38 }
39
40 void run(void) {
41     int q = 0, p = 0;
42     self();

```

```

43 while (p < n && p < m && S[p] == T[p]) p++;
44 ex[0] = p;
45 p--;
46 for (int i = 1; i < n; i++) {
47     int l = nxt[i - q];
48     if (i + l - 1 < p) {
49         ex[i] = l;
50     } else {
51         int j = max(0, p - i + 1);
52         while (i + j < n && S[i + j] == T[j]) j++;
53         ex[i] = j;
54         p = i + j - 1;
55         q = i;
56     }
57 }
58 }
59
60 void inspect(void) {
61     printf("S:");
62     for (int i = 0; i < n; i++) putchar(S[i]);
63     puts("");
64     printf("T:");
65     for (int i = 0; i < m; i++) putchar(T[i]);
66     puts("");
67     printf("next:");
68     for (int i = 0; i < m; i++) printf(" %d", nxt[i]);
69     puts("");
70     printf("extend:");
71     for (int i = 0; i < n; i++) printf(" %d", ex[i]);
72     puts("");
73 }
74
75 } // exkmp

```

## 6.4 Suffix Array

```

1  /* Suffix Array, copied. */
2
3  #define MAXN    (200010)
4  namespace mzry_sa {
5  int wx[MAXN],wy[MAXN],*x,*y,wss[MAXN],wv[MAXN];
6
7  bool dacmp(int *r,int n,int a,int b,int l) {
8      return a+l<n && b+l<n && r[a]==r[b]&&r[a+l]==r[b+l];
9  }
10 void da(int str[],int sa[],int rank[],int height[],int n,int m) {
11     int *s = str;
12     int *x=wx,*y=wy,*t,p;
13     int i,j;
14     for(i=0; i<m; i++)wss[i]=0;
15     for(i=0; i<n; i++)wss[x[i]=s[i]]++;
16     for(i=1; i<m; i++)wss[i]+=wss[i-1];
17     for(i=n-1; i>=0; i--)sa[--wss[x[i]]]=i;
18     for(j=1,p=1; p<n && j<n; j*=2,m=p) {
19         for(i=n-j,p=0; i<n; i++)y[p++]=i;
20         for(i=0; i<n; i++)if(sa[i]-j>=0)y[p++]=sa[i]-j;
21         for(i=0; i<n; i++)wv[i]=x[y[i]];
22         for(i=0; i<m; i++)wss[i]=0;
23         for(i=0; i<n; i++)wss[wv[i]]++;
24         for(i=1; i<m; i++)wss[i]+=wss[i-1];
25         for(i=n-1; i>=0; i--)sa[--wss[wv[i]]]=y[i];
26         for(t=x,x=y,y=t,p=1,i=1,x[sa[0]]=0; i<n; i++)

```

```

27     x[sa[i]]=dacmp(y,n,sa[i-1],sa[i],j)?p-1:p++;
28 }
29 for(int i=0; i<n; i++) rank[sa[i]]=i;
30 for(int i=0,j=0,k=0; i<n; height[rank[i++]]=k)
31     if(rank[i]>0)
32         for(k?k--:0,j=sa[rank[i]-1];
33             i+k < n && j+k < n && str[i+k]==str[j+k];
34             k++);
35 }
36 }
37
38 /*
39 Suffix array O(n lg^2 n)
40 LCP table O(n)
41 */
42 #include <cstdio>
43 #include <algorithm>
44 #include <cstring>
45
46 using namespace std;
47
48 #define REP(i, n) for (int i = 0; i < (int)(n); ++i)
49
50 namespace SuffixArray {
51     const int MAXN = 1 << 21;
52     char * S;
53     int N, gap;
54     int sa[MAXN], pos[MAXN], tmp[MAXN], lcp[MAXN];
55
56     bool sufCmp(int i, int j) {
57         if (pos[i] != pos[j])
58             return pos[i] < pos[j];
59         i += gap;
60         j += gap;
61         return (i < N && j < N) ? pos[i] < pos[j] : i > j;
62     }
63
64     void buildSA() {
65         N = strlen(S);
66         REP(i, N) sa[i] = i, pos[i] = S[i];
67         for (gap = 1; gap <= N; gap <<= 1) {
68             sort(sa, sa + N, sufCmp);
69             REP(i, N - 1) tmp[i + 1] = tmp[i] + sufCmp(sa[i], sa[i + 1]);
70             REP(i, N) pos[sa[i]] = tmp[i];
71             if (tmp[N - 1] == N - 1) break;
72         }
73     }
74
75     void buildLCP() {
76         for (int i = 0, k = 0; i < N; ++i) if (pos[i] != N - 1) {
77             for (int j = sa[pos[i] + 1]; S[i + k] == S[j + k];)
78                 ++k;
79             lcp[pos[i]] = k;
80             if (k) --k;
81         }
82     }
83 } // end namespace SuffixArray
84
85 namespace HashSuffixArray {
86     const int
87     MAXN = 1 << 21;
88
89     typedef unsigned long long hash;
90

```

```

91 const hash BASE = 137;
92
93 int N;
94 char * S;
95 int sa[MAXN];
96 hash h[MAXN], hPow[MAXN];
97
98 #define getHash(lo, size) (h[lo] - h[(lo) + (size)] * hPow[size])
99
100 inline bool sufCmp(int i, int j) {
101     int lo = 1, hi = min(N - i, N - j);
102     while (lo <= hi) {
103         int mid = (lo + hi) >> 1;
104         if (getHash(i, mid) == getHash(j, mid))
105             lo = mid + 1;
106         else
107             hi = mid - 1;
108     }
109     return S[i + hi] < S[j + hi];
110 }
111
112 void buildSA() {
113     N = strlen(S);
114     hPow[0] = 1;
115     for (int i = 1; i <= N; ++i)
116         hPow[i] = hPow[i - 1] * BASE;
117     h[N] = 0;
118     for (int i = N - 1; i >= 0; --i)
119         h[i] = h[i + 1] * BASE + S[i], sa[i] = i;
120
121     stable_sort(sa, sa + N, sufCmp);
122 }
123
124 } // end namespace HashSuffixArray
125
126 namespace lrj_sa {
127     const int MAXN = 1000;
128     char s[MAXN]; /* 原始字符数组 (最后一个字符应必须是0, 而前面的字符必须非0) */
129     int sa[MAXN], t[MAXN], t2[MAXN], c[MAXN], n; /* n seems to be the length of s. */
130     /* every character is in [0,m-1] */
131     void build_sa(int m) {
132         int i, *x = t, *y = t2;
133         for(i = 0; i < m; ++i) c[i] = 0;
134         for(i = 0; i < n; ++i) c[x[i]=s[i]]++;
135         for(i = 1; i < m; ++i) c[i] += c[i-1];
136         for(i = n-1; i >= 0; --i) sa[--c[x[i]]] = i;
137         for(int k = 1; k <= n; k <= 1) {
138             int p = 0;
139             for(i = n-k; i < n; ++i) y[p++] = i;
140             for(i = 0; i < n; ++i) if(sa[i] >= k) y[p++] = sa[i]-k;
141             for(i = 0; i < m; ++i) c[i] = 0;
142             for(i = 0; i < n; ++i) c[x[y[i]]]++;
143             for(i = 0; i < m; ++i) c[i] += c[i-1];
144             for(i = n-1; i >= 0; --i) sa[--c[x[y[i]]]] = y[i];
145             swap(x,y);
146             p = 1;
147             x[sa[0]] = 0;
148             for(i = 1; i < n; ++i)
149                 x[sa[i]] = y[sa[i-1]]==y[sa[i]] && y[sa[i-1]+k]==y[sa[i]+k] ? p-1:p++;
150             if(p >= n) break;
151             m = p;
152         }
153     }
154     int rank[MAXN], height[MAXN];

```



```

155 void get_height(void) {
156     int i,j,k = 0;
157     for(i = 0; i < n; ++i) rank[sa[i]] = i;
158     for(i = 0; i < n; ++i) {
159         if(k) k--;
160         j = sa[rank[i]-1];
161         while(s[i+k]==s[j+k]) k++;
162         height[rank[i]] = k;
163     }
164 }
165 } // end namespace lrj_sa

```

## 6.5 Aho-Corasick Automata

```

1  /* Aho-Corasick automaton algorithm, by Abreto<m@abreto.net>. */
2
3  #define MAXN 500500
4  #define NALPHA 26
5  #define FIRSA 'a'
6
7  /* pointer version => { */
8  struct vtx {
9      vtx *nxt[NALPHA];
10     vtx *fail;
11     int end;
12 } vtxs[MAXN];
13 int nvtxs;
14 void myclr(void) {
15     nvtxs = 0;
16 }
17 vtx *new_vtx(void) {
18     vtx *ret = vtxs+(nvtxs++);
19     for(int i = 0; i < NALPHA; i++)
20         ret->nxt[i] = NULL;
21     ret->fail = NULL;
22     ret->end = 0;
23     return ret;
24 }
25 void myins(vtx *root, char const *s) {
26     for( ; *s ; s++ ) {
27         int of = (*s) - FIRSA;
28         if ( NULL == root->nxt[of] ) {
29             root->nxt[of] = new_vtx();
30         }
31         root = root->nxt[of];
32     }
33     root->end++;
34 }
35 void build_ac(vtx *root) {
36     queue<vtx *> q;
37     q.push(root);
38     while(!q.empty()) {
39         vtx *p = q.front();
40         q.pop();
41         for(int i = 0; i < NALPHA; i++) {
42             if( NULL == p->nxt[i] ) continue;
43             if( root == p ) p->nxt[i]->fail = root;
44             else {
45                 vtx *t = p->fail;
46                 while ( t && NULL == t->nxt[i] ) {
47                     t = t->fail;
48                 }

```

```

49     if (t) p->nxt[i]->fail = t->nxt[i];
50     else p->nxt[i]->fail = root;
51 }
52 /* version[1] { */
53 p->nxt[i]->end += p->nxt[i]->fail->end; /* update this sum, add its existing
    prefix to this. */
54 /* } */
55 q.push(p->nxt[i]);
56 }
57 }
58 }
59 int qry(vtx *root, char const *s) {
60     vtx *p = root;
61     int cnt = 0;
62     for( ; *s ; s++ ) {
63         int of = (*s) - FIRSTA;
64         while( p != root && NULL == p->nxt[of] ) {
65             p = p->fail;
66         }
67         if (p->nxt[of]) p = p->nxt[of];
68         cnt += p->end; // correct when version[1] exists.
69         // if version[1] not exists, you need to add all ends from this vertex up.
70         //for( vtx *t = p ; t ; t = t->fail )
71         // cnt += t->end;
72     }
73     return cnt;
74 }
75 /* } */
76
77 /* — usage (of pointer version) — */
78 #include <bits/stdc++.h>
79 using namespace std;
80
81 char S[1000100];
82 char pat[64];
83
84 int main(void) {
85     int T, N;
86     vtx *root = NULL;
87     scanf("%d", &T);
88     while(T--) {
89         myclr();
90         root = new_vtx();
91         scanf("%s", S);
92         scanf("%d", &N);
93         while(N--) {
94             scanf("%s", pat);
95             myins(root, pat);
96         }
97         build_ac(root);
98         printf("%d\n", qry(root, S));
99     }
100     return 0;
101 }

```

## 6.6 Manacher

```

1 char t[MAXL<<1];
2 int p[MAXL<<1];
3 int manacher(char *s) {
4     int i;
5     int sl = strlen(s);

```

```

6   int pos = 0, mxr = 0;
7   int ret = 0;
8   t[0] = '^';
9   for(i = 0; i < sl; ++i) {
10      t[i*2+1] = '#';
11      t[i*2+2] = s[i];
12  }
13  t[sl*2+1] = '#';
14  t[sl*2+2] = '$';
15  sl = sl*2+2;
16  for(i = 1; i < sl; ++i) {
17      if(i <= mxr) {
18          p[i] = min(p[2*pos - i], mxr-i+1);
19      } else {
20          p[i] = 1;
21      }
22      while( t[i-p[i]] == t[i+p[i]] ) p[i]++;
23      if( i + p[i] - 1 > mxr ) {
24          mxr = i+p[i]-1;
25          pos = i;
26      }
27      ret = max(ret, p[i]-1);
28  }
29  return ret;
30 }

```

## 7 Utility

### 7.1 IO plug-in

```

1  /* I/O Plug-in, by Abreto <m@abreto.net>. */
2  #include <stdio.h>
3
4  #if ( _WIN32 || __WIN32__ || _WIN64 || __WIN64__ )
5  #define INT64 "%I64d"
6  #else
7  #define INT64 "%lld"
8  #endif
9
10 #if ( _WIN32 || __WIN32__ || _WIN64 || __WIN64__ )
11 #define UNS64 "%I64u"
12 #else
13 #define UNS64 "%llu"
14 #endif
15
16 #define ISDIGIT(x) ((x>='0')&&(x<='9'))
17 int readn(int *n) {
18     int c=0;
19     *n=0;
20     for(; !ISDIGIT(c); c=getchar());
21     for(; ISDIGIT(c); c=getchar()) *n=(*n)*10+c-'0';
22     return (*n);
23 }
24 void putn(int n) {
25     int ns[16]= {0,n%10},nd=1;
26     while(n/=10)ns[++nd]=n%10;
27     while(nd)putchar(ns[nd--]+'0');
28 }

```

  

```

1  #include <cstdio>
2
3  class abio {

```

```

4 static const unsigned BUF_SZ = 65536;
5 FILE *istream, *ostream;
6 char ibuf[BUF_SZ], obuf[BUF_SZ];
7 bool reached_eof;
8 size_t ip, isz;
9 size_t op, osz;
10 inline void clear_ibuf(void) {
11     ip = isz = 0u;
12 }
13 inline void clear_obuf(void) {
14     op = osz = 0u;
15 }
16 inline void clear_buffer(void) {
17     reached_eof = false;
18     clear_ibuf();
19     clear_obuf();
20 }
21 inline size_t read_buffer(void) {
22     isz = std::fread(ibuf, sizeof(char), BUF_SZ, istream);
23     ip = 0;
24     return isz;
25 }
26 inline size_t write_buffer(void) {
27     if(osz) {
28         size_t ret = std::fwrite(obuf+op, sizeof(char), osz-op, ostream);
29         op += ret;
30         if(op == osz) clear_obuf();
31         return ret;
32     }
33     return 0;
34 }
35 inline abio &reach_eof(void) {
36     reached_eof = true;
37     return (*this);
38 }
39 public:
40 static const char endl = '\n';
41 abio(FILE *input = stdin, FILE *output = stdout) {
42     this->istream = input;
43     this->ostream = output;
44     clear_buffer();
45 }
46 abio(const char *input, const char *output) {
47     this->istream = std::fopen(input, "r");
48     this->istream = std::fopen(output, "w+");
49     clear_buffer();
50 }
51 ~abio(void) {
52     write_buffer();
53     std::fclose(istream);
54     std::fclose(ostream);
55 }
56 operator bool() const {
57     return (!reached_eof);
58 }
59 inline int getchar(void) {
60     if(isz == ip) read_buffer();
61     if(isz == ip) return EOF;
62     return ibuf[ip++];
63 }
64 inline int putchar(int ch) {
65     if(osz == BUF_SZ) write_buffer();
66     if(osz == BUF_SZ) return EOF;
67     return (obuf[osz++] = ch);

```

```

68 }
69 abio &read_int(int &x) {
70     int flag = 0, ch = getchar();
71     for (; (EOF!=ch)&&((ch<'0')||ch>'9')); ch=getchar(); if ('-' == ch) flag = 1;
72     if (EOF == ch) return (this->reach_eof());
73     x = 0;
74     for (; (ch>='0')&&(ch<='9'); ch=getchar()) x = x * 10 + (ch - '0');
75     if ( flag ) x *= (-1);
76     return (*this);
77 }
78 abio &read_ll(long long int &x) {
79     int flag = 0, ch = getchar();
80     for (; (EOF!=ch)&&((ch<'0')||ch>'9')); ch=getchar(); if ('-' == ch) flag = 1;
81     if (EOF == ch) return (this->reach_eof());
82     x = 0ll;
83     for (; (ch>='0')&&(ch<='9'); ch=getchar()) x = x * 10ll + (ch - '0');
84     if ( flag ) x *= (-1ll);
85     return (*this);
86 }
87 abio &read_unsigned(unsigned &x) {
88     int ch = getchar();
89     for(; (EOF != ch) && ((ch < '0') || (ch > '9'))); ch = getchar();
90     if (EOF == ch) return (this->reach_eof());
91     x = 0u;
92     for(; (ch >= '0') && (ch <= '9'); ch = getchar()) x = x * 10u + (ch - '0');
93     return (*this);
94 }
95 abio &read_ull(unsigned long long int &x) {
96     int ch = getchar();
97     for(; (EOF != ch) && ((ch < '0') || (ch > '9'))); ch = getchar();
98     if (EOF == ch) return (this->reach_eof());
99     x = 0ull;
100     for(; (ch >= '0') && (ch <= '9'); ch = getchar()) x = x * 10ull + (ch - '0');
101     return (*this);
102 }
103 /* set interrupt as '\n' to read a whole line. */
104 abio &read_s(char *s, const char interrupt = '\n') {
105     int ch = getchar();
106     while((EOF!=ch)&&(ch<'!'||ch>'~'))ch=getchar();
107     if(EOF==ch) return (this->reach_eof());
108     for(; (EOF!=ch)&&(interrupt!=ch)&&(ch>=' ' && ch<='~')); ch=getchar();(*s++)=ch;
109     (*s)=0;
110     return (*this);
111 }
112 abio &write_int(int x, char append = 0) {
113     int d[20],nd=0;
114     if(0==x) putchar('0');
115     if(x<0) {
116         putchar('-');
117         x=-x;
118     }
119     while(x) {
120         d[nd++]=x%10;
121         x/=10;
122     }
123     while(nd--)putchar('0'+d[nd]);
124     if(append)putchar(append);
125     return (*this);
126 }
127 abio &write_ll(long long int x, char append = 0) {
128     int d[20],nd=0;
129     if(0==x) putchar('0');
130     if(x<0) {
131         putchar('-');

```

```

132     x=-x;
133 }
134 while(x) {
135     d[nd++]=x%10;
136     x/=10;
137 }
138 while(nd--) putchar('0'+d[nd]);
139 if(append) putchar(append);
140 return (*this);
141 }
142 abio &write_unsigned(unsigned x, char append = 0) {
143     int d[20], nd=0;
144     if(0==x) putchar('0');
145     while(x) {
146         d[nd++]=x%10;
147         x/=10;
148     }
149     while(nd--) putchar('0'+d[nd]);
150     if(append) putchar(append);
151     return (*this);
152 }
153 abio &write_ull(unsigned long long int x, char append = 0) {
154     int d[20], nd=0;
155     if(0==x) putchar('0');
156     while(x) {
157         d[nd++]=x%10;
158         x/=10;
159     }
160     while(nd--) putchar('0'+d[nd]);
161     if(append) putchar(append);
162     return (*this);
163 }
164 abio &write_s(const char *s, char append = 0) {
165     while(*s) putchar(*s++);
166     if(append) putchar(append);
167     return (*this);
168 }
169 abio &operator>>(char &ch) {
170     ch = getchar();
171     if(EOF==ch) return (this->reach_eof());
172     return (*this);
173 }
174 abio &operator>>(int &x) {
175     return read_int(x);
176 }
177 abio &operator>>(long long int &x) {
178     return read_ll(x);
179 }
180 abio &operator>>(unsigned &x) {
181     return read_unsigned(x);
182 }
183 abio &operator>>(unsigned long long int &x) {
184     return read_ull(x);
185 }
186 abio &operator>>(char *s) {
187     return read_s(s);
188 }
189 abio &operator<<(const char ch) {
190     putchar(ch);
191     return (*this);
192 }
193 abio &operator<<(const int x) {
194     return write_int(x);
195 }

```

```

196     abio &operator<<(const long long int x) {
197         return write_ll(x);
198     }
199     abio &operator<<(const unsigned x) {
200         return write_unsigned(x);
201     }
202     abio &operator<<(const unsigned long long int x) {
203         return write_ull(x);
204     }
205     abio &operator<<(const char *s) {
206         return write_s(s);
207     }
208 } io;

```

## 7.2 Random Numbers

```

1  #include <algorithm>
2  #include <chrono>
3  #include <iostream>
4  #include <random>
5  #include <vector>
6  using namespace std;
7
8  const int N = 3000000;
9
10 double average_distance(const vector<int> &permutation) {
11     double distance_sum = 0;
12
13     for (int i = 0; i < N; i++)
14         distance_sum += abs(permutation[i] - i);
15
16     return distance_sum / N;
17 }
18
19 int main() {
20     /* use mt19937_64 if you want 64-bit random numbers */
21     mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
22     vector<int> permutation(N);
23
24     for (int i = 0; i < N; i++)
25         permutation[i] = i;
26
27     shuffle(permutation.begin(), permutation.end(), rng);
28     cout << average_distance(permutation) << '\n';
29
30     for (int i = 0; i < N; i++)
31         permutation[i] = i;
32
33     for (int i = 1; i < N; i++)
34         swap(permutation[i], permutation[uniform_int_distribution<int>(0, i)(rng)]);
35
36     cout << average_distance(permutation) << '\n';
37 }
38
39 // -----
40 // rand() * rand() % M
41 // or
42 // rand() << 16 | rand()
43 // maybe also work in some scene;

```

## 8 Appendix

### 8.1 C++ Reference

#### 8.1.1 STL

### 8.2 Java Reference

#### 8.2.1 Basic

##### Structure

```

1 import java.io.*;
2 import java.util.*;
3 import java.math.*;
4
5 public class Main {
6     public static final int maxn = 50050;
7     public static int[] int_array = new int[maxn]; /** Array */
8     public static int[] another_arr = {1, 2, 3, 5};
9     public static void main(String[] args) throws Exception {
10         Scanner cin = new Scanner(System.in);
11         int a = cin.nextInt(), b = cin.nextInt();
12         System.out.println(a + b);
13         for (int i = 0; i < 4; i++)
14             System.out.println(another_arr[i]);
15     }
16 }

```

##### Constant

```

1 public static final int A = 0;

```

##### Array

```

1 int [] a = new int[5];
2 int [] b = {10, 35, 45, 89, 90};

```

#### Bit op

The Java programming language also provides operators that perform bitwise and bit shift operations on integral types. The operators discussed in this section are less commonly used. Therefore, their coverage is brief; the intent is to simply make you aware that these operators exist.

The unary bitwise complement operator “~” inverts a bit pattern; it can be applied to any of the integral types, making every “0” a “1” and every “1” a “0”. For example, a byte contains 8 bits; applying this operator to a value whose bit pattern is “00000000” would change its pattern to “11111111”.

The signed left shift operator “<<” shifts a bit pattern to the left, and the signed right shift operator “>>” shifts a bit pattern to the right. The bit pattern is given by the left-hand operand, and the number of positions to shift by the right-hand operand. The unsigned right shift operator “>>>” shifts a zero into the leftmost position, while the leftmost position after “>>>” depends on sign extension.

The bitwise & operator performs a bitwise AND operation.

The bitwise ^ operator performs a bitwise exclusive OR operation.

The bitwise | operator performs a bitwise inclusive OR operation.

#### 8.2.2 BigInteger

Immutable arbitrary-precision integers. All operations behave as if `BigInteger` were represented in two's-complement notation (like Java's primitive integer types). `BigInteger` provides analogues to all of Java's primitive integer operators, and all relevant methods from `java.lang.Math`. Additionally, `BigInteger` provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations.

##### API

- ZERO, ONE, TEN



- Constructors

```
1 | public BigInteger(String val, int radix)
```

- primes

```
1 | /* prob of composite <= 2^(-100) */
2 | public BigInteger nextProbablePrime()
3 | /* return true if the probability that it is prime exceeds (1 - 1/(2^certainty))
   | */
4 | public boolean isProbablePrime(int certainty)
```

- valueOf

- 算术运算

```
1 | public BigInteger add(BigInteger val)
2 | public BigInteger subtract(BigInteger val)
3 | public BigInteger multiply(BigInteger val)
4 | public BigInteger divide(BigInteger val)
5 | public BigInteger[] divideAndRemainder(BigInteger val) // [quotient, remainder]
6 | public BigInteger remainder(BigInteger val)
7 | public BigInteger pow(int exponent)
8 | public BigInteger gcd(BigInteger val)
9 | public BigInteger abs()
10 | public BigInteger negate()
11 | public int signum() // -1, 0 or 1
12 | public BigInteger mod(BigInteger m) // always returns a non-negative BigInteger.
13 | public BigInteger modPow(BigInteger exponent, BigInteger m)
14 | public BigInteger modInverse(BigInteger m)
```

- 位运算

```
1 | public BigInteger shiftLeft(int n)
2 | public BigInteger shiftRight(int n)
3 | public BigInteger and(BigInteger val)
4 | public BigInteger or(BigInteger val)
5 | public BigInteger xor(BigInteger val)
6 | public BigInteger not()
7 | public boolean testBit(int n)
8 | public BigInteger setBit(int n)
9 | public BigInteger clearBit(int n)
10 | public BigInteger flipBit(int n)
11 | public int getLowestSetBit() // lowbit
12 | public int bitLength()
13 | public int bitCount()
```

- 比较

```
1 | public int compareTo(BigInteger val) // -1, 0 or 1 if this () val
2 | public BigInteger min(BigInteger val)
3 | public BigInteger max(BigInteger val)
```

- transform

```
1 | public int hashCode()
2 | public String toString(int radix)
3 | public int intValue()
4 | public long longValue()
```

### 8.2.3 BigDecimal

Immutable, arbitrary-precision signed decimal numbers. A BigDecimal consists of an arbitrary precision integer unscaled value and a 32-bit integer scale. If zero or positive, the scale is the number of digits to the right of the decimal point. If negative, the unscaled value of the number is multiplied by ten to the power of the negation of the scale. The value of the number represented by the BigDecimal is therefore  $(unscaledValue \times 10^{-scale})$ .

#### API

- Constructors

```
1 public BigDecimal(String val)
2 public BigDecimal(double val)
3 public BigDecimal(BigInteger val)
4 public BigDecimal(int val)
5 public static BigDecimal valueOf(long unscaledVal, int scale) // u * 10^(-scale)
6 public static BigDecimal valueOf(double val)
```

- arithmetic operation

```
1 public BigDecimal add(BigDecimal augend)
2 public BigDecimal subtract(BigDecimal subtrahend)
3 public BigDecimal multiply(BigDecimal multiplicand)
4 public BigDecimal divide(BigDecimal divisor)
5 public BigDecimal divideToIntegralValue(BigDecimal divisor)
6 public BigDecimal remainder(BigDecimal divisor)
7 public BigDecimal[] divideAndRemainder(BigDecimal divisor)
8 public BigDecimal pow(int n)
9 public BigDecimal abs()
10 public BigDecimal negate()
11 public int signum()
12 public int scale()
13 public int precision()
```

- transform

```
1 // RoundingMode.
2 // UP, DOWN, CEILING, FLOOR, HALF_UP, HALF_DOWN, HALF_EVEN
3 public MathContext(int setPrecision[, RoundingMode setRoundingMode])
4 public BigDecimal round(MathContext mc)
5 public int hashCode()
6 public String toString()
7 public String toPlainString()
8 public double doubleValue()
```

- comparison

```
1 public int compareTo(BigDecimal val)
2 public BigDecimal min(BigDecimal val)
3 public BigDecimal max(BigDecimal val)
```

### 8.2.4 Sorting

java.util.Arrays

```
1 public static void sort(int[] a[, int fromIndex, int toIndex]) // ascending numerical
  order.
2 // or parallelSort ?
```

### 8.3 Environment test items

- 一秒运算次数，带模一秒跑多少
- 行末空格
- `assert(0)` 是 WA 还是 RE
- `RAND_MAX` 大小