

# Gracias Demetrio



## Contents

## 1 Data structures

### 1.1 Segment tree

```

1  #define oper min
2  #define NEUT INF
3  struct STree { // segment tree for min over integers
4      vector<int> st; int n;
5      STree(int n): st(4*n+5, NEUT), n(n) {}
6      void init(int k, int s, int e, int *a){
7          if(s+1==e){st[k]=a[s];return;}
8          int m=(s+e)/2;
9          init(2*k, s, m, a); init(2*k+1, m, e, a);
10         st[k]=oper(st[2*k], st[2*k+1]);
11     }
12     void upd(int k, int s, int e, int p, int v){
13         if(s+1==e){st[k]=v;return;}
14         int m=(s+e)/2;
15         if(p<m)upd(2*k, s, m, p, v);
16         else upd(2*k+1, m, e, p, v);
17         st[k]=oper(st[2*k], st[2*k+1]);
18     }
19     int query(int k, int s, int e, int a, int b){
20         if(s>=b || e<=a)return NEUT;
21         if(s>=a && e<=b)return st[k];
22         int m=(s+e)/2;
23         return oper(query(2*k, s, m, a, b), query(2*k+1, m, e, a, b));
24     }
25     void init(int *a){init(1, 0, n, a);}
26     void upd(int p, int v){upd(1, 0, n, p, v);}
27     int query(int a, int b){return query(1, 0, n, a, b);}
28 }; // usage: STree rmq(n); rmq.init(x); rmq.upd(i, v); rmq.query(s, e);

```

### 1.2 Segment tree - Lazy propagation

```

1  struct STree { // example: range sum with range addition
2      vector<int> st, lazy; int n;
3      STree(int n): st(4*n+5, 0), lazy(4*n+5, 0), n(n) {}
4      void init(int k, int s, int e, int *a){
5          lazy[k]=0; // lazy neutral element
6          if(s+1==e){st[k]=a[s];return;}
7          int m=(s+e)/2;
8          init(2*k, s, m, a); init(2*k+1, m, e, a);

```

```

9     st[k]=st[2*k]+st[2*k+1]; // operation
10 }
11 void push(int k, int s, int e){
12     if(!lazy[k])return; // if neutral, nothing to do
13     st[k]+=(e-s)*lazy[k]; // update st according to lazy
14     if(s+1<e){ // propagate to children
15         lazy[2*k]+=lazy[k];
16         lazy[2*k+1]+=lazy[k];
17     }
18     lazy[k]=0; // clear node lazy
19 }
20 void upd(int k, int s, int e, int a, int b, int v){
21     push(k,s,e);
22     if(s>=b||e<=a)return;
23     if(s>=a&&e<=b){
24         lazy[k]+=v; // accumulate lazy
25         push(k,s,e);return;
26     }
27     int m=(s+e)/2;
28     upd(2*k,s,m,a,b,v);upd(2*k+1,m,e,a,b,v);
29     st[k]=st[2*k]+st[2*k+1]; // operation
30 }
31 int query(int k, int s, int e, int a, int b){
32     if(s>=b||e<=a)return 0; // operation neutral
33     push(k,s,e);
34     if(s>=a&&e<=b)return st[k];
35     int m=(s+e)/2;
36     return query(2*k,s,m,a,b)+query(2*k+1,m,e,a,b); // operation
37 }
38 void init(int *a){init(1,0,n,a);}
39 void upd(int a, int b, int v){upd(1,0,n,a,b,v);}
40 int query(int a, int b){return query(1,0,n,a,b);}
41 }; // usage: STree rmq(n);rmq.init(x);rmq.upd(s,e,v);rmq.query(s,e);
    
```

### 1.3 Segment tree - Persistence

```

1 #define oper(a,b) min(a,b)
2 #define NEUT INF
3 struct STree { // persistent segment tree for min over integers
4     vector<int> st, L, R; int n,sz,rt;
5     STree(int n): st(1,NEUT),L(1,0),R(1,0),n(n),rt(0),sz(1){}
6     int new_node(int v, int l=0, int r=0){
7         int ks=sz(st);
    
```

```

8     st.pb(v);L.pb(l);R.pb(r);
9     return ks;
10 }
11 int init(int s, int e, int *a){ // not necessary in most cases
12     if(s+1==e)return new_node(a[s]);
13     int m=(s+e)/2,l=init(s,m,a),r=init(m,e,a);
14     return new_node(oper(st[l],st[r]),l,r);
15 }
16 int upd(int k, int s, int e, int p, int v){
17     int ks=new_node(st[k],L[k],R[k]);
18     if(s+1==e){st[ks]=v;return ks;}
19     int m=(s+e)/2,ps;
20     if(p<m)ps=upd(L[ks],s,m,p,v),L[ks]=ps;
21     else ps=upd(R[ks],m,e,p,v),R[ks]=ps;
22     st[ks]=oper(st[L[ks]],st[R[ks]]);
23     return ks;
24 }
25 int query(int k, int s, int e, int a, int b){
26     if(e<=a||b<=s)return NEUT;
27     if(a<=s&&e<=b)return st[k];
28     int m=(s+e)/2;
29     return oper(query(L[k],s,m,a,b),query(R[k],m,e,a,b));
30 }
31 int init(int *a){return init(0,n,a);}
32 int upd(int k, int p, int v){return rt=upd(k,0,n,p,v);}
33 int upd(int p, int v){return upd(rt,p,v);} // update on last root
34 int query(int k,int a, int b){return query(k,0,n,a,b);}
35 }; // usage: STree rmq(n);root=rmq.init(x);new_root=rmq.upd(root,i,v);
    rmq.query(root,s,e);
    
```

### 1.4 Segment tree - 2D

```

1 int n,m;
2 int a[MAXN][MAXN],st[2*MAXN][2*MAXN];
3 void build(){
4     for(i,0,n)for(j,0,m)st[i+n][j+m]=a[i][j];
5     for(i,0,n)for(int j=m-1;j-->0)
6         st[i+n][j]=op(st[i+n][j<<1],st[i+n][j<<1|1]);
7     for(int i=n-1;i-->0)for(j,0,2*m)
8         st[i][j]=op(st[i<<1][j],st[i<<1|1][j]);
9 }
10 void upd(int x, int y, int v){
11     st[x+n][y+m]=v;
    
```

```

12 for(int j=y+m;j>1;j>=1)st[x+n][j>1]=op(st[x+n][j],st[x+n][j^1]);
13 for(int i=x+n;i>1;i>=1)for(int j=y+m;j>=1)
14     st[i>1][j]=op(st[i][j],st[i^1][j]);
15 }
16 int query(int x0, int x1, int y0, int y1){
17     int r=NEUT;
18     for(int i0=x0+n,i1=x1+n;i0<i1;i0>=1,i1>=1){
19         int t[4],q=0;
20         if(i0&1)t[q++]=i0++;
21         if(i1&1)t[q++]--i1;
22         fore(k,0,q)for(int j0=y0+m,j1=y1+m;j0<j1;j0>=1,j1>=1){
23             if(j0&1)r=op(r,st[t[k]][j0++]);
24             if(j1&1)r=op(r,st[t[k]][--j1]);
25         }
26     }
27     return r;
28 }
    
```

## 1.5 Sparse table (static RMQ)

```

1 #define oper min
2 int st[K][1<<K];int n; // K such that 2^K>n
3 void st_init(int *a){
4     fore(i,0,n)st[0][i]=a[i];
5     fore(k,1,K)fore(i,0,n-(1<<k)+1)
6         st[k][i]=oper(st[k-1][i],st[k-1][i+(1<<(k-1))]);
7 }
8 int st_query(int s, int e){
9     int k=31-__builtin_clz(e-s);
10    return oper(st[k][s],st[k][e-(1<<k)]);
11 }
    
```

## 1.6 Fenwick tree

```

1 int ft[MAXN+1]; // for more dimensions, make ft multi-dimensional
2 void upd(int i0, int v){ // add v to i0th element (0-based)
3     // add extra fors for more dimensions
4     for(int i=i0+1;i<=MAXN;i+=i&-i)ft[i]+=v;
5 }
6 int get(int i0){ // get sum of range [0,i0]
7     int r=0;
8     // add extra fors for more dimensions
9     for(int i=i0;i; i-=i&-i)r+=ft[i];
10    return r;
    
```

```

11 }
12 int get_sum(int i0, int i1){ // get sum of range [i0,i1] (0-based)
13     return get(i1)-get(i0);
14 }
    
```

## 1.7 Wavelet tree

```

1 struct WT {
2     vector<int> wt[1<<20];int n;
3     void init(int k, int s, int e){
4         if(s+1==e)return;
5         wt[k].clear();wt[k].pb(0);
6         int m=(s+e)/2;
7         init(2*k,s,m);init(2*k+1,m,e);
8     }
9     void add(int k, int s, int e, int v){
10        if(s+1==e)return;
11        int m=(s+e)/2;
12        if(v<m)wt[k].pb(wt[k].back()),add(2*k,s,m,v);
13        else wt[k].pb(wt[k].back()+1),add(2*k+1,m,e,v);
14    }
15    int query0(int k, int s, int e, int a, int b, int i){
16        if(s+1==e)return s;
17        int m=(s+e)/2;
18        int q=(b-a)-(wt[k][b]-wt[k][a]);
19        if(i<q)return query0(2*k,s,m,a-wt[k][a],b-wt[k][b],i);
20        else return query0(2*k+1,m,e,wt[k][a],wt[k][b],i-q);
21    }
22    void upd(int k, int s, int e, int i){
23        if(s+1==e)return;
24        int m=(s+e)/2;
25        int v0=wt[k][i+1]-wt[k][i],v1=wt[k][i+2]-wt[k][i+1];
26        if(!v0&&!v1)upd(2*k,s,m,i-wt[k][i]);
27        else if(v0&&v1)upd(2*k+1,m,e,wt[k][i]);
28        else if(v0)wt[k][i+1]--;
29        else wt[k][i+1]++;
30    }
31    void init(int _n){n=_n;init(1,0,n);} // (values in range [0,n])
32    void add(int v){add(1,0,n,v);}
33    int query0(int a, int b, int i){ // ith element in range [a,b]
34        return query0(1,0,n,a,b,i); // (if it was sorted)
35    }
36    void upd(int i){ // swap positions i,i+1
    
```

```

37     upd(1,0,n,i);
38 }
39 };
    
```

## 1.8 STL extended set

```

1  #include<ext/pb_ds/assoc_container.hpp>
2  #include<ext/pb_ds/tree_policy.hpp>
3  using namespace __gnu_pbds;
4  typedef tree<int,null_type,less<int>,rb_tree_tag,
5      tree_order_statistics_node_update> ordered_set;
6  // find_by_order(i) -> iterator to ith element
6  // order_of_key(k) -> position (int) of lower_bound of k
    
```

## 1.9 STL rope

```

1  #include <ext/rope>
2  using namespace __gnu_cxx;
3  rope<int> s;
4  // Sequence with O(log(n)) random access, insert, erase at any position
5  // s.push_back(x);
6  // s.insert(i,r) // insert rope r at position i
7  // s.erase(i,k) // erase subsequence [i,i+k)
8  // s.substr(i,k) // return new rope corresponding to subsequence [i,i+k)
9  // s[i] // access ith element (cannot modify)
10 // s.mutable_reference_at(i) // acces ith element (allows modification)
11 // s.begin() and s.end() are const iterators (use mutable_begin(),
    mutable_end() to allow modification)
    
```

## 1.10 Treap (as BST)

```

1  typedef struct item *pitem;
2  struct item {
3      int pr,key,cnt;
4      pitem l,r;
5      item(int key):key(key),pr(rand()),cnt(1),l(0),r(0) {}
6  };
7  int cnt(pitem t){return t?t->cnt:0;}
8  void upd_cnt(pitem t){if(t)t->cnt=cnt(t->l)+cnt(t->r)+1;}
9  void split(pitem t, int key, pitem& l, pitem& r){ // l: < key, r: >= key
10     if(!t)l=r=0;
11     else if(key<t->key)split(t->l,key,l,t->l),r=t;
12     else split(t->r,key,t->r,r),l=t;
13     upd_cnt(t);
    
```

```

14 }
15 void insert(pitem& t, pitem it){
16     if(!t)t=it;
17     else if(it->pr>t->pr)split(t,it->key,it->l,it->r),t=it;
18     else insert(it->key<t->key?t->l:t->r,it);
19     upd_cnt(t);
20 }
21 void merge(pitem& t, pitem l, pitem r){
22     if(!l||!r)t=l?l:r;
23     else if(l->pr>r->pr)merge(l->r,l->r,r),t=l;
24     else merge(r->l,l,r->l),t=r;
25     upd_cnt(t);
26 }
27 void erase(pitem& t, int key){
28     if(t->key==key)merge(t,t->l,t->r);
29     else erase(key<t->key?t->l:t->r,key);
30     upd_cnt(t);
31 }
32 void unite(pitem &t, pitem l, pitem r){
33     if(!l||!r){t=l?l:r;return;}
34     if(l->pr<r->pr)swap(l,r);
35     pitem p1,p2;split(r,l->key,p1,p2);
36     unite(l->l,l->l,p1);unite(l->r,l->r,p2);
37     t=l;upd_cnt(t);
38 }
39 pitem kth(pitem t, int k){
40     if(!t)return 0;
41     if(k==cnt(t->l))return t;
42     return k<cnt(t->l)?kth(t->l,k):kth(t->r,k-cnt(t->l)-1);
43 }
44 pair<int,int> lb(pitem t, int key){ // position and value of lower_bound
45     if(!t)return {0,1<<30}; // (special value)
46     if(key>t->key){
47         auto w=lb(t->r,key);w.fst+=cnt(t->l)+1;return w;
48     }
49     auto w=lb(t->l,key);
50     if(w.fst==cnt(t->l))w.snd=t->key;
51     return w;
52 }
    
```

## 1.11 Treap (implicit key)

```

1  // example that supports range reverse and addition updates, and range
    
```

```

    sum query
2 // (commented parts are specific to this problem)
3 typedef struct item *pitem;
4 struct item {
5     int pr,cnt,val;
6     // int sum; // (paramters for range query)
7     // bool rev;int add; // (parameters for lazy prop)
8     pitem l,r;
9     item(int val): pr(rand()),cnt(1),val(val),l(0),r(0)/*,sum(val),rev(0),
        add(0)*/ {}
10 };
11 void push(pitem it){
12     if(it){
13         /*if(it->rev){
14             swap(it->l,it->r);
15             if(it->l)it->l->rev^=true;
16             if(it->r)it->r->rev^=true;
17             it->rev=false;
18         }
19         it->val+=it->add;it->sum+=it->cnt*it->add;
20         if(it->l)it->l->add+=it->add;
21         if(it->r)it->r->add+=it->add;
22         it->add=0;*/
23     }
24 }
25 int cnt(pitem t){return t?t->cnt:0;}
26 // int sum(pitem t){return t?push(t),t->sum:0;}
27 void upd_cnt(pitem t){
28     if(t){
29         t->cnt=cnt(t->l)+cnt(t->r)+1;
30         // t->sum=t->val+sum(t->l)+sum(t->r);
31     }
32 }
33 void merge(pitem& t, pitem l, pitem r){
34     push(l);push(r);
35     if(!l||!r)t=l?l:r;
36     else if(l->pr>r->pr)merge(l->r,l->r,r),t=l;
37     else merge(r->l,l,r->l),t=r;
38     upd_cnt(t);
39 }
40 void split(pitem t, pitem& l, pitem& r, int sz){ // sz:desired size of l
41     if(!t){l=r=0;return;}
42     push(t);

```

```

43     if(sz<=cnt(t->l))split(t->l,l,t->l,sz),r=t;
44     else split(t->r,t->r,r,sz-1-cnt(t->l)),l=t;
45     upd_cnt(t);
46 }
47 void output(pitem t){ // useful for debugging
48     if(!t)return;
49     push(t);
50     output(t->l);printf("%d",t->val);output(t->r);
51 }
52 // use merge and split for range updates and queries

```

## 1.12 Treap (with node father)

```

1 // node father is useful to keep track of the chain of each node
2 // alternative: splay tree
3 // IMPORTANT: add pointer f in struct item
4 void merge(pitem& t, pitem l, pitem r){
5     push(l);push(r);
6     if(!l||!r)t=l?l:r;
7     else if(l->pr>r->pr)merge(l->r,l->r,r),l->r->f=t=l;
8     else merge(r->l,l,r->l),r->l->f=t=r;
9     upd_cnt(t);
10 }
11 void split(pitem t, pitem& l, pitem& r, int sz){
12     if(!t){l=r=0;return;}
13     push(t);
14     if(sz<=cnt(t->l)){
15         split(t->l,l,t->l,sz);r=t;
16         if(l)l->f=0;
17         if(t->l)t->l->f=t;
18     }
19     else {
20         split(t->r,t->r,r,sz-1-cnt(t->l));l=t;
21         if(r)r->f=0;
22         if(t->r)t->r->f=t;
23     }
24     upd_cnt(t);
25 }
26 void push_all(pitem t){
27     if(t->f)push_all(t->f);
28     push(t);
29 }
30 pitem root(pitem t, int& pos){ // get root and position for node t

```

```

31 push_all(t);
32 pos=cnt(t->l);
33 while(t->f){
34     pitem f=t->f;
35     if(t==f->r)pos+=cnt(f->l)+1;
36     t=f;
37 }
38 return t;
39 }
    
```

### 1.13 Link-Cut tree

```

1 typedef struct item *pitem;
2 struct item {
3     int pr;bool rev;
4     pitem l,r,f,d;
5     item():pr(rand()),l(0),r(0),f(0),d(0),rev(0){}
6 };
7 void push(pitem t){
8     if(t&& t->rev){
9         swap(t->l,t->r);
10        if(t->l)t->l->rev^=1;
11        if(t->r)t->r->rev^=1;
12        t->rev=0;
13    }
14 }
15 void merge(pitem& t, pitem l, pitem r){
16     push(l);push(r);
17     if(!l||!r)t=l?l:r;
18     else if(l->pr>r->pr)merge(l->r,l->r,r),l->r->f=t=l;
19     else merge(r->l,l,r->l),r->l->f=t=r;
20 }
21 void push_all(pitem t){
22     if(t->f)push_all(t->f);
23     push(t);
24 }
25 void split(pitem t, pitem& l, pitem& r){
26     push_all(t);
27     l=t->l;r=t->r;t->l=t->r=0;
28     while(t->f){
29         pitem f=t->f;t->f=0;
30         if(t==f->l){
31             if(r)r->f=f;
    
```

```

32         f->l=r;r=f;
33     }
34     else {
35         if(l)l->f=f;
36         f->r=l;l=f;
37     }
38     t=f;
39 }
40 if(l)l->f=0;
41 if(r)r->f=0;
42 }
43 pitem path(pitem p){return p->f?path(p->f):p;}
44 pitem tail(pitem p){push(p);return p->r?tail(p->r):p;}
45 pitem expose(pitem p){
46     pitem q,r,t;
47     split(p,q,r);
48     if(q)tail(q)->d=p;
49     merge(p,p,r);
50     while(t=tail(p),t->d){
51         pitem d=t->d;t->d=0;
52         split(d,q,r);
53         if(q)tail(q)->d=d;
54         merge(p,p,d);merge(p,p,r);
55     }
56     return p;
57 }
58 pitem root(pitem v){return tail(expose(v));}
59 void evert(pitem v){expose(v)->rev^=1;v->d=0;}
60 void link(pitem v, pitem w){ // make v son of w
61     evert(v);
62     pitem p=path(v);
63     merge(p,p,expose(w));
64 }
65 void cut(pitem v){ // cut v from its father
66     pitem p,q;
67     expose(v);split(v,p,q);v->d=0;
68 }
69 void cut(pitem v, pitem w){evert(w);cut(v);}
    
```

### 1.14 Convex hull trick (static)

```

1 typedef ll tc;
2 struct Line{tc m,h;};
    
```

```

3 struct CHT { // for minimum (for maximum just change the sign of lines)
4     vector<Line> c;
5     int pos=0;
6     tc in(Line a, Line b){
7         tc x=b.h-a.h,y=a.m-b.m;
8         return x/y+(x%y?!((x>0)^(y>0)):0); // ==ceil(x/y)
9     }
10    void add(tc m, tc h){ // m's should be non increasing
11        Line l=(Line){m,h};
12        if(c.size()&&m==c.back().m){
13            l.h=min(h,c.back().h);c.pop_back();if(pos)pos--;
14        }
15        while(c.size()>1&&in(c.back(),l)<=in(c[c.size()-2],c.back())){
16            c.pop_back();if(pos)pos--;
17        }
18        c.pb(l);
19    }
20    inline bool fbin(tc x, int m){return in(c[m],c[m+1])>x;}
21    tc eval(tc x){
22        // O(log n) query:
23        int s=0,e=c.size();
24        while(e-s>1){int m=(s+e)/2;
25            if(fbin(x,m-1))e=m;
26            else s=m;
27        }
28        return c[s].m*x+c[s].h;
29        // O(1) query (for ordered x's):
30        while(pos>0&&fbin(x,pos-1))pos--;
31        while(pos<c.size()-1&&!fbin(x,pos))pos++;
32        return c[pos].m*x+c[pos].h;
33    }
34 };

```

### 1.15 Convex hull trick (dynamic)

```

1 typedef ll tc;
2 const tc is_query=-(1LL<<62); // special value for query
3 struct Line {
4     tc m,b;
5     mutable multiset<Line>::iterator it,end;
6     const Line* succ(multiset<Line>::iterator it) const {
7         return (++it==end? NULL : &*it);}
8     bool operator<(const Line& rhs) const {

```

```

9         if(rhs.b!=is_query)return m<rhs.m;
10        const Line *s=succ(it);
11        if(!s)return 0;
12        return b-s->b<(s->m-m)*rhs.m;
13    }
14 };
15 struct HullDynamic : public multiset<Line> { // for maximum
16     bool bad(iterator y){
17         iterator z=next(y);
18         if(y==begin()){
19             if(z==end())return false;
20             return y->m==z->m&&y->b<=z->b;
21         }
22         iterator x=prev(y);
23         if(z==end())return y->m==x->m&&y->b<=x->b;
24         return (x->b-y->b)*(z->m-y->m)>=(y->b-z->b)*(y->m-x->m);
25     }
26     iterator next(iterator y){return ++y;}
27     iterator prev(iterator y){return --y;}
28     void add(tc m, tc b){
29         iterator y=insert((Line){m,b});
30         y->it=y->end=end();
31         if(bad(y)){erase(y);return;}
32         while(next(y)!=end()&&bad(next(y)))erase(next(y));
33         while(y!=begin()&&bad(prev(y)))erase(prev(y));
34     }
35     tc eval(tc x){
36         Line l=*lower_bound((Line){x,is_query});
37         return l.m*x+l.b;
38     }
39 };

```

### 1.16 Gain-cost-set

```

1 // stores pairs (benefit,cost) (erases non-optimal pairs)
2 struct GCS {
3     set<pair<int,int> > s;
4     void add(int g, int c){
5         pair<int,int> x={g,c};
6         auto p=s.lower_bound(x);
7         if(p!=s.end()&&p->snd<=x.snd)return;
8         if(p!=s.begin()){ // erase pairs with less benefit
9             --p;           // and more cost

```



```

10     while(p->snd>=x.snd){
11         if(p==s.begin()){s.erase(p);break;}
12         s.erase(p--);
13     }
14 }
15 s.insert(x);
16 }
17 int get(int gain){ // min cost for some benefit
18     auto p=s.lower_bound((pair<int,int>){gain,-INF});
19     int r=p==s.end()?INF:p->snd;
20     return r;
21 }
22 };

```

## 1.17 Disjoint intervals

```

1 // stores disjoint intervals as [first, second)
2 struct disjoint_intervals {
3     set<pair<int,int> > s;
4     void insert(pair<int,int> v){
5         if(v.fst>=v.snd) return;
6         auto at=s.lower_bound(v);auto it=at;
7         if(at!=s.begin()&&(--at)->snd>=v.fst)v.fst=at->fst,--it;
8         for(;it!=s.end()&&it->fst<=v.snd;s.erase(it++))
9             v.snd=max(v.snd,it->snd);
10        segs.insert(v);
11    }
12 };

```

## 2 Graphs

### 2.1 Topological sort

```

1 vector<int> g[MAXN];int n;
2 vector<int> tsort(){ // lexicographically smallest topological sort
3     vector<int> r;priority_queue<int> q;
4     vector<int> d(2*n,0);
5     for(i,0,n)for(j,0,g[i].size())d[g[i][j]]++;
6     for(i,0,n)if(!d[i])q.push(-i);
7     while(!q.empty()){
8         int x=-q.top();q.pop();r.pb(x);
9         for(i,0,g[x].size()){
10             d[g[x][i]]--;

```

```

11         if(!d[g[x][i]])q.push(-g[x][i]);
12     }
13 }
14 return r; // if not DAG it will have less than n elements
15 }

```

### 2.2 Kruskal (+ Union-Find)

```

1 int uf[MAXN];
2 void uf_init(){memset(uf,-1,sizeof(uf));}
3 int uf_find(int x){return uf[x]<0?x:uf[x]=uf_find(uf[x]);}
4 bool uf_join(int x, int y){
5     x=uf_find(x);y=uf_find(y);
6     if(x==y)return false;
7     if(uf[x]>uf[y])swap(x,y);
8     uf[x]+=uf[y];uf[y]=x;
9     return true;
10 }
11 vector<pair<ll,pair<int,int> > > es; // edges (cost,(u,v))
12 ll kruskal(){ // assumes graph is connected
13     sort(es.begin(),es.end());uf_init();
14     ll r=0;
15     for(i,0,es.size()){
16         int x=es[i].snd.fst,y=es[i].snd.snd;
17         if(uf_join(x,y))r+=es[i].fst; // (x,y,c) belongs to mst
18     }
19     return r; // total cost
20 }

```

### 2.3 Dijkstra

```

1 vector<pair<int,int> > g[MAXN]; // u->[(v,cost)]
2 ll dist[MAXN];
3 void dijkstra(int x){
4     memset(dist,-1,sizeof(dist));
5     priority_queue<pair<ll,int> > q;
6     dist[x]=0;q.push({0,x});
7     while(!q.empty()){
8         x=q.top().snd;ll c=-q.top().fst;q.pop();
9         if(dist[x]!=c)continue;
10        for(i,0,g[x].size()){
11            int y=g[x][i].fst; ll c=g[x][i].snd;
12            if(dist[y]<0||dist[x]+c<dist[y])
13                dist[y]=dist[x]+c,q.push({-dist[y],y});

```



```

14     }
15 }
16 }

```

## 2.4 Bellman-Ford

```

1 int n;
2 vector<pair<int,int> > g[MAXN]; // u->[(v,cost)]
3 ll dist[MAXN];
4 void bford(int src){ // O(nm)
5     fill(dist,dist+n,INF);dist[src]=0;
6     fore(_,0,n)fore(x,0,n)if(dist[x]!=INF)for(auto t:g[x]){
7         dist[t.fst]=min(dist[t.fst],dist[x]+t.snd);
8     }
9     fore(x,0,n)if(dist[x]!=INF)for(auto t:g[x]){
10        if(dist[t.fst]>dist[x]+t.snd){
11            // neg cycle: all nodes reachable from t.fst have -INF distance
12            // to reconstruct neg cycle: save "prev" of each node, go up from
13            // t.fst until repeating a node. this node and all nodes between
14            // the two occurrences form a neg cycle
15        }
16    }
17 }

```

## 2.5 Floyd-Warshall

```

1 // g[i][j]: weight of edge (i, j) or INF if there's no edge
2 // g[i][i]=0
3 ll g[MAXN][MAXN];int n;
4 void floyd(){ // O(n^3) . Replaces g with min distances
5     fore(k,0,n)fore(i,0,n)if(g[i][k]<INF)fore(j,0,n)if(g[k][j]<INF)
6         g[i][j]=min(g[i][j],g[i][k]+g[k][j]);
7 }
8 bool inNegCycle(int v){return g[v][v]<0;}
9 bool hasNegCycle(int a, int b){ // true iff there's neg cycle in between
10     fore(i,0,n)if(g[a][i]<INF&&g[i][b]<INF&&g[i][i]<0)return true;
11     return false;
12 }

```

## 2.6 Strongly connected components (+ 2-SAT)

```

1 // MAXN: max number of nodes or 2 * max number of variables (2SAT)
2 bool truth[MAXN]; // truth[cmp[i]]=value of variable i (2SAT)
3 int nvar;int neg(int x){return MAXN-1-x;} // (2SAT)

```

```

4 vector<int> g[MAXN];
5 int n,lw[MAXN],idx[MAXN],qidx,cmp[MAXN],qcmp;
6 stack<int> st;
7 void tjn(int u){
8     lw[u]=idx[u]=++qidx;
9     st.push(u);cmp[u]=-2;
10    for(int v:g[u]){
11        if(!idx[v]||cmp[v]==-2){
12            if(!idx[v]) tjn(v);
13            lw[u]=min(lw[u],lw[v]);
14        }
15    }
16    if(lw[u]==idx[u]){
17        int x,l=-1;
18        do{x=st.top();st.pop();cmp[x]=qcmp;if(min(x,neg(x))<nvar)l=x;}
19        while(x!=u);
20        if(l!=-1)truth[qcmp]=(cmp[neg(l)]<0); // (2SAT)
21        qcmp++;
22    }
23 }
24 void scc(){
25     memset(idx,0,sizeof(idx));qidx=0;
26     memset(cmp,-1,sizeof(cmp));qcmp=0;
27     fore(i,0,n)if(!idx[i])tjn(i);
28 }
29 // Only for 2SAT:
30 void addor(int a, int b){g[neg(a)].pb(b);g[neg(b)].pb(a);}
31 bool satisf(int _nvar){
32     nvar=_nvar;n=MAXN;scc();
33     fore(i,0,nvar)if(cmp[i]==cmp[neg(i)])return false;
34     return true;
35 }

```

## 2.7 Articulation - Bridges - Biconnected

```

1 vector<int> g[MAXN];int n;
2 struct edge {int u,v,comp;bool bridge;};
3 vector<edge> e;
4 void add_edge(int u, int v){
5     g[u].pb(e.size());g[v].pb(e.size());
6     e.pb(edge){u,v,-1,false};
7 }
8 int D[MAXN],B[MAXN],T;

```

```

9  int nbc; // number of biconnected components
10 int art[MAXN]; // articulation point iff !=0
11 stack<int> st; // only for biconnected
12 void dfs(int u,int pe){
13     B[u]=D[u]=T++;
14     for(int ne:g[u])if(ne!=pe){
15         int v=e[ne].u^e[ne].v^u;
16         if(D[v]<0){
17             st.push(ne);dfs(v,ne);
18             if(B[v]>D[u])e[ne].bridge = true; // bridge
19             if(B[v]>=D[u]){
20                 art[u]++; // articulation
21                 int last; // start biconnected
22                 do {
23                     last=st.top();st.pop();
24                     e[last].comp=nbc;
25                 } while(last!=ne);
26                 nbc++; // end biconnected
27             }
28             B[u]=min(B[u],B[v]);
29         }
30         else if(D[v]<D[u])st.push(ne),B[u]=min(B[u],D[v]);
31     }
32 }
33 void doit(){
34     memset(D,-1,sizeof(D));memset(art,0,sizeof(art));
35     nbc=T=0;
36     fore(i,0,n)if(D[i]<0)dfs(i,-1),art[i]--;
37 }

```

## 2.8 Chu-Liu (minimum spanning arborescence)

```

1 //O(n*m) minimum spanning tree in directed graph
2 //returns -1 if not possible
3 //included i-th edge if take[i]!=0
4 typedef int tw; tw INF=1ll<<30;
5 struct edge{int u,v,id;tw len;};
6 struct ChuLiu{
7     int n; vector<edge> e;
8     vector<int> inc,dec,take,pre,num,id,vis;
9     vector<tw> inw;
10    void add_edge(int x, int y, tw w){
11        inc.pb(0); dec.pb(0); take.pb(0);

```

```

12    e.pb({x,y,SZ(e),w});
13 }
14 ChuLiu(int n):n(n),pre(n),num(n),id(n),vis(n),inw(n){}
15 tw doit(int root){
16     auto e2=e;
17     tw ans=0; int eg=SZ(e)-1,pos=SZ(e)-1;
18     while(1){
19         fore(i,0,n) inw[i]=INF,id[i]=vis[i]=-1;
20         for(auto ed:e2 if(ed.len<inw[ed.v])){
21             inw[ed.v]=ed.len; pre[ed.v]=ed.u;
22             num[ed.v]=ed.id;
23         }
24         inw[root]=0;
25         fore(i,0,n) if(inw[i]==INF) return -1;
26         int tot=-1;
27         fore(i,0,n){
28             ans+=inw[i];
29             if(i!=root)take[num[i]]++;
30             int j=i;
31             while(vis[j]!=i&&j!=root&&id[j]<0)vis[j]=i,j=pre[j];
32             if(j!=root&&id[j]<0){
33                 id[j]=++tot;
34                 for(int k=pre[j];k!=j;k=pre[k]) id[k]=tot;
35             }
36         }
37         if(tot<0)break;
38         fore(i,0,n) if(id[i]<0)id[i]=++tot;
39         n=tot+1; int j=0;
40         fore(i,0,SZ(e2)){
41             int v=e2[i].v;
42             e2[j].v=id[e2[i].v];
43             e2[j].u=id[e2[i].u];
44             if(e2[j].v!=e2[j].u){
45                 e2[j].len=e2[i].len-inw[v];
46                 inc.pb(e2[i].id);
47                 dec.pb(num[v]);
48                 take.pb(0);
49                 e2[j++].id=++pos;
50             }
51         }
52         e2.resize(j);
53         root=id[root];
54     }

```

```

55 while(pos>eg){
56     if(take[pos]>0) take[inc[pos]]++, take[dec[pos]]--;
57     pos--;
58 }
59 return ans;
60 }
61 };

```

## 2.9 LCA - Binary Lifting

```

1 vector<int> g[1<<K];int n; // K such that 2^K>=n
2 int F[K][1<<K],D[1<<K];
3 void lca_dfs(int x){
4     fore(i,0,g[x].size()){
5         int y=g[x][i];if(y==F[0][x])continue;
6         F[0][y]=x;D[y]=D[x]+1;lca_dfs(y);
7     }
8 }
9 void lca_init(){
10     D[0]=0;F[0][0]=-1;
11     lca_dfs(0);
12     fore(k,1,K)fore(x,0,n)
13         if(F[k-1][x]<0)F[k][x]=-1;
14         else F[k][x]=F[k-1][F[k-1][x]];
15 }
16 int lca(int x, int y){
17     if(D[x]<D[y])swap(x,y);
18     for(int k=K-1;k>=0;--k)if(D[x]-(1<<k)>=D[y])x=F[k][x];
19     if(x==y)return x;
20     for(int k=K-1;k>=0;--k)if(F[k][x]!=F[k][y])x=F[k][x],y=F[k][y];
21     return F[0][x];
22 }

```

## 2.10 Heavy-Light decomposition

```

1 vector<int> g[MAXN];
2 int wg[MAXN],dad[MAXN],dep[MAXN]; // weight,father,depth
3 void dfs1(int x){
4     wg[x]=1;
5     for(int y:g[x])if(y!=dad[x]){
6         dad[y]=x;dep[y]=dep[x]+1;dfs1(y);
7         wg[x]+=wg[y];
8     }
9 }

```

```

10 int curpos,pos[MAXN],head[MAXN];
11 void hld(int x, int c){
12     if(c<0)c=x;
13     pos[x]=curpos++;head[x]=c;
14     int mx=-1;
15     for(int y:g[x])if(y!=dad[x]&&(mx<0||wg[mx]<wg[y]))mx=y;
16     if(mx>=0)hld(mx,c);
17     for(int y:g[x])if(y!=mx&&y!=dad[x])hld(y,-1);
18 }
19 void hld_init(){dad[0]=-1;dep[0]=0;dfs1(0);curpos=0;hld(0,-1);}
20 int query(int x, int y, STree& rmq){
21     int r=NEUT;
22     while(head[x]!=head[y]){
23         if(dep[head[x]]>dep[head[y]])swap(x,y);
24         r=oper(r,rmq.query(pos[head[y]],pos[y]+1));
25         y=dad[head[y]];
26     }
27     if(dep[x]>dep[y])swap(x,y); // now x is lca
28     r=oper(r,rmq.query(pos[x],pos[y]+1));
29     return r;
30 }
31 // for updating: rmq.upd(pos[x],v);
32 // queries on edges: - assign values of edges to "child" node
33 // - change pos[x] to pos[x]+1 in query (line 28)

```

## 2.11 Centroid decomposition

```

1 vector<int> g[MAXN];int n;
2 bool tk[MAXN];
3 int fat[MAXN]; // father in centroid decomposition
4 int szt[MAXN]; // size of subtree
5 int calcsz(int x, int f){
6     szt[x]=1;
7     for(auto y:g[x])if(y!=f&&!tk[y])szt[x]+=calcsz(y,x);
8     return szt[x];
9 }
10 void cdfs(int x=0, int f=-1, int sz=-1){ // 0(nlogn)
11     if(sz<0)sz=calcsz(x,-1);
12     for(auto y:g[x])if(!tk[y]&&szt[y]*2>=sz){
13         szt[x]=0;cdfs(y,f,sz);return;
14     }
15     tk[x]=true;fat[x]=f;
16     for(auto y:g[x])if(!tk[y])cdfs(y,x);

```

```

17 }
18 void centroid(){memset(tk,false,sizeof(tk));cdfs();}

```

## 2.12 Parallel DFS

```

1 struct Tree {
2     int n,z[2];
3     vector<vector<int>> g;
4     vector<int> ex,ey,p,w,f,v[2];
5     Tree(int n):g(n),w(n),f(n){}
6     void add_edge(int x, int y){
7         p.pb(g[x].size());g[x].pb(ex.size());ex.pb(x);ey.pb(y);
8         p.pb(g[y].size());g[y].pb(ex.size());ex.pb(y);ey.pb(x);
9     }
10    bool go(int k){ // returns true if it finds new node
11        int& x=z[k];
12        while(x>=0&&
13            (w[x]==g[x].size()||w[x]==g[x].size()-1&&(g[x].back()^1)==f[x]))
14            x=f[x]>=0?ex[f[x]]:-1;
15        if(x<0)return false;
16        if((g[x][w[x]]^1)==f[x])w[x]++;
17        int e=g[x][w[x]],y=ey[e];
18        f[y]=e;w[x]++;w[y]=0;x=y;
19        v[k].pb(x);
20        return true;
21    }
22    vector<int> erase_edge(int e){
23        e*=2; // erases eth edge, returns smaller component
24        int x=ex[e],y=ey[e];
25        p[g[x].back()]=p[e];
26        g[x][p[e]]=g[x].back();g[x].pop_back();
27        p[g[y].back()]=p[e^1];
28        g[y][p[e^1]]=g[y].back();g[y].pop_back();
29        f[x]=f[y]=-1;
30        w[x]=w[y]=0;
31        z[0]=x;z[1]=y;
32        v[0]={x};v[1]={y};
33        bool d0=true,d1=true;
34        while(d0&&d1)d0=go(0),d1=go(1);
35        if(d1)return v[0];
36        return v[1];
37    }
38 };

```

## 2.13 Eulerian path

```

1 // Directed version (uncomment commented code for undirected)
2 struct edge {
3     int y;
4     // list<edge>::iterator rev;
5     edge(int y):y(y){}
6 };
7 list<edge> g[MAXN];
8 void add_edge(int a, int b){
9     g[a].push_front(edge(b));//auto ia=g[a].begin();
10    // g[b].push_front(edge(a));auto ib=g[b].begin();
11    // ia->rev=ib;ib->rev=ia;
12 }
13 vector<int> p;
14 void go(int x){
15     while(g[x].size()){
16         int y=g[x].front().y;
17         //g[y].erase(g[x].front().rev);
18         g[x].pop_front();
19         go(y);
20     }
21     p.push_back(x);
22 }
23 vector<int> get_path(int x){ // get a path that begins in x
24     // check that a path exists from x before calling to get_path!
25     p.clear();go(x);reverse(p.begin(),p.end());
26     return p;
27 }

```

## 2.14 Dynamic connectivity

```

1 struct UnionFind {
2     int n,comp;
3     vector<int> uf,si,c;
4     UnionFind(int n=0):n(n),comp(n),uf(n),si(n,1){
5         fore(i,0,n)uf[i]=i;}
6     int find(int x){return x==uf[x]?x:find(uf[x]);}
7     bool join(int x, int y){
8         if((x=find(x))==y=find(y))return false;
9         if(si[x]<si[y])swap(x,y);
10        si[x]+=si[y];uf[y]=x;comp--;c.pb(y);
11        return true;

```

```

12 }
13 int snap(){return c.size();}
14 void rollback(int snap){
15     while(c.size()>snap){
16         int x=c.back();c.pop_back();
17         si[uf[x]]-=si[x];uf[x]=x;comp++;
18     }
19 }
20 };
21 enum {ADD,DEL,QUERY};
22 struct Query {int type,x,y};
23 struct DynCon {
24     vector<Query> q;
25     UnionFind dsu;
26     vector<int> mt;
27     map<pair<int,int>,int> last;
28     DynCon(int n):dsu(n){
29     void add(int x, int y){
30         if(x>y)swap(x,y);
31         q.pb((Query){ADD,x,y});mt.pb(-1);last[{x,y}]=q.size()-1;
32     }
33     void remove(int x, int y){
34         if(x>y)swap(x,y);
35         q.pb((Query){DEL,x,y});
36         int pr=last[{x,y}];mt[pr]=q.size()-1;mt.pb(pr);
37     }
38     void query(){q.pb((Query){QUERY,-1,-1});mt.pb(-1);}
39     void process(){ // answers all queries in order
40         if(!q.size())return;
41         fore(i,0,q.size())if(q[i].type==ADD&&mt[i]<0)mt[i]=q.size();
42         go(0,q.size());
43     }
44     void go(int s, int e){
45         if(s+1==e){
46             if(q[s].type==QUERY) // answer query using DSU
47                 printf("%d\n",dsu.comp);
48             return;
49         }
50         int k=dsu.snap(),m=(s+e)/2;
51         for(int i=e-1;i>=m;--i)if(mt[i]>=0&&mt[i]<s)dsu.join(q[i].x,q[i].y);
52         go(s,m);dsu.rollback(k);
53         for(int i=m-1;i>=s;--i)if(mt[i]>=e)dsu.join(q[i].x,q[i].y);
54         go(m,e);dsu.rollback(k);

```

```

55 }
56 };

```

## 2.15 Edmond's blossom (matching in general graphs)

```

1 vector<int> g[MAXN];
2 int n,m,mt[MAXN],qh,qt,q[MAXN],ft[MAXN],bs[MAXN];
3 bool inq[MAXN],inb[MAXN],inp[MAXN];
4 int lca(int root, int x, int y){
5     memset(inp,0,sizeof(inp));
6     while(1){
7         inp[x=bs[x]]=true;
8         if(x==root)break;
9         x=ft[mt[x]];
10    }
11    while(1){
12        if(inp[y=bs[y]])return y;
13        else y=ft[mt[y]];
14    }
15 }
16 void mark(int z, int x){
17     while(bs[x]!=z){
18         int y=mt[x];
19         inb[bs[x]]=inb[bs[y]]=true;
20         x=ft[y];
21         if(bs[x]!=z)ft[x]=y;
22     }
23 }
24 void contr(int s, int x, int y){
25     int z=lca(s,x,y);
26     memset(inb,0,sizeof(inb));
27     mark(z,x);mark(z,y);
28     if(bs[x]!=z)ft[x]=y;
29     if(bs[y]!=z)ft[y]=x;
30     fore(x,0,n)if(inb[bs[x]]){
31         bs[x]=z;
32         if(!inq[x])inq[q[++qt]=x]=true;
33     }
34 }
35 int findp(int s){
36     memset(inq,0,sizeof(inq));
37     memset(ft,-1,sizeof(ft));
38     fore(i,0,n)bs[i]=i;

```

```

39 inq[q[qh=qt=0]=s]=true;
40 while(qh<=qt){
41     int x=q[qh++];
42     for(int y:g[x])if(bs[x]!=bs[y]&&mt[x]!=y){
43         if(y==s|mt[y]>=0&&ft[mt[y]]>=0)contr(s,x,y);
44         else if(ft[y]<0){
45             ft[y]=x;
46             if(mt[y]<0)return y;
47             else if(!inq[mt[y]])inq[q[++qt]=mt[y]]=true;
48         }
49     }
50 }
51 return -1;
52 }
53 int aug(int s, int t){
54     int x=t,y,z;
55     while(x>=0){
56         y=ft[x];
57         z=mt[y];
58         mt[y]=x;mt[x]=y;
59         x=z;
60     }
61     return t>=0;
62 }
63 int edmonds(){ // O(n^2 m)
64     int r=0;
65     memset(mt,-1,sizeof(mt));
66     fore(x,0,n)if(mt[x]<0)r+=aug(x,findp(x));
67     return r;
68 }

```

## 3 Math

### 3.1 Identities

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

$$C_n \sim \frac{4^n}{n^{3/2} \sqrt{\pi}}$$

$\sigma(n) = O(\log(\log(n)))$  (number of divisors of  $n$ )

$$F_{2n+1} = F_n^2 + F_{n+1}^2$$

$$F_{2n} = F_{n+1}^2 - F_{n-1}^2$$

$$\sum_{i=1}^n F_i = F_{n+2} - 1$$

$$F_{n+i}F_{n+j} - F_nF_{n+i+j} = (-1)^n F_i F_j$$

(Möbius Inv. Formula) Let  $g(n) = \sum_{d|n} f(d)$ , then  $f(n) = \sum_{d|n} g(d) \mu\left(\frac{n}{d}\right)$ .

### 3.2 Theorems

- 1 (Tutte) A graph,  $G = (V, E)$ , has a perfect matching **if and only if for** every subset  $U$  of  $V$ , the subgraph induced by  $V - U$  has at most  $|U|$  connected components with an odd number of vertices.
- 2 Petersens Theorem. Every cubic, bridgeless graph contains a perfect matching.
- 3 (Dilworth) In any finite partially ordered set, the maximum number of elements in any antichain equals the minimum number of chains in any partition of the set into chains
- 4 Pick:  $A=I+B/2-1$  (area of polygon, points inside, points on border)

### 3.3 Integer floor division

```

1 void floordiv(ll x, ll y, ll& q, ll& r) { // (for negative x)
2     q=x/y;r=x%y;
3     if((r!=0)&&((r<0)!=(y<0)))q--,r+=y;
4 }

```

### 3.4 Sieve of Eratosthenes

```

1 int cr[MAXN]; // -1 if prime, some not trivial divisor if not
2 void init_sieve(){
3     memset(cr,-1,sizeof(cr));
4     fore(i,2,MAXN)if(cr[i]<0)for(ll j=1LL*i*i;j<MAXN;j+=i)cr[j]=i;
5 }
6 map<int,int> fact(int n){ // must call init_cribe before
7     map<int,int> r;
8     while(cr[n]>=0)r[cr[n]]++,n/=cr[n];
9     if(n>1)r[n]++;
10    return r;
11 }

```

### 3.5 Generate divisors

```

1 void div_rec(vector<ll>& r, vector<pair<ll,int> >& f, int k, ll c){
2     if(k==f.size()){r.pb(c);return;}
3     fore(i,0,f[k].snd+1)div_rec(r,f,k+1,c),c*=f[k].fst;
4 }
5 vector<ll> divisors(vector<pair<ll,int> > f){
6     vector<ll> r; // returns divisors given factorization
7     div_rec(r,f,0,1);

```

```

8   return r;
9 }

```

### 3.6 Pollard's rho

```

1 ll gcd(ll a, ll b){return a?gcd(b%a,a):b;}
2 ll mulmod(ll a, ll b, ll m) {
3     ll r=a*b-(ll)((long double)a*b/m+.5)*m;
4     return r<0?r+m:r;
5 }
6 ll expmod(ll b, ll e, ll m){
7     if(!e)return 1;
8     ll q=expmod(b,e/2,m);q=mulmod(q,q,m);
9     return e&1?mulmod(b,q,m):q;
10 }
11 bool is_prime_prob(ll n, int a){
12     if(n==a)return true;
13     ll s=0,d=n-1;
14     while(d%2==0)s++,d/=2;
15     ll x=expmod(a,d,n);
16     if((x==1)|| (x+1==n))return true;
17     fore(_,0,s-1){
18         x=mulmod(x,x,n);
19         if(x==1)return false;
20         if(x+1==n)return true;
21     }
22     return false;
23 }
24 bool rabin(ll n){ // true iff n is prime
25     if(n==1)return false;
26     int ar[]={2,3,5,7,11,13,17,19,23};
27     fore(i,0,9)if(!is_prime_prob(n,ar[i]))return false;
28     return true;
29 }
30 ll rho(ll n){
31     if(!(n&1))return 2;
32     ll x=2,y=2,d=1;
33     ll c=rand()%n+1;
34     while(d==1){
35         x=(mulmod(x,x,n)+c)%n;
36         y=(mulmod(y,y,n)+c)%n;
37         y=(mulmod(y,y,n)+c)%n;
38         if(x>y)d=gcd(x-y,n);

```

```

39     else d=gcd(y-x,n);
40 }
41 return d==n?rho(n):d;
42 }
43 void fact(ll n, map<ll,int>& f){ //O (lg n)^3
44     if(n==1)return;
45     if(rabin(n)){f[n]++;return;}
46     ll q=rho(n);fact(q,f);fact(n/q,f);
47 }
48 // optimized version: replace rho and fact with the following:
49 const int MAXP=1e6+1; // sieve size
50 int sv[MAXP]; // sieve
51 ll add(ll a, ll b, ll m){return (a+=b)<m?a-a-m;}
52 ll rho(ll n){
53     static ll s[MAXP];
54     while(1){
55         ll x=rand()%n,y=x,c=rand()%n;
56         ll *px=s,*py=s,v=0,p=1;
57         while(1){
58             *py++=y=add(mulmod(y,y,n),c,n);
59             *py++=y=add(mulmod(y,y,n),c,n);
60             if((x=*px++)==y)break;
61             ll t=p;
62             p=mulmod(p,abs(y-x),n);
63             if(!p)return gcd(t,n);
64             if(++v==26){
65                 if((p=gcd(p,n))>1&&p<n)return p;
66                 v=0;
67             }
68         }
69         if(v&&(p=gcd(p,n))>1&&p<n)return p;
70     }
71 }
72 void init_sv(){
73     fore(i,2,MAXP)if(!sv[i])for(ll j=i;j<MAXP;j+=i)sv[j]=i;
74 }
75 void fact(ll n, map<ll,int>& f){ // call init_sv first!!!
76     for(auto&& p:f){
77         while(n%p.fst==0){
78             p.snd++;
79             n/=p.fst;
80         }
81     }

```



```

82 if(n<MAXP)while(n>1)f[sv[n]]++,n/=sv[n];
83 else if(rabin(n))f[n]++;
84 else {ll q=rho(n);fact(q,f);fact(n/q,f);}
85 }

```

### 3.7 Simpson's rule

```

1 double integrate(double f(double), double a, double b, int n=10000){
2     double r=0,h=(b-a)/n,fa=f(a),fb;
3     fore(i,0,n){fb=f(a+h*(i+1));r+=fa+4*f(a+h*(i+0.5))+fb;fa=fb;}
4     return r*h/6.;
5 }

```

### 3.8 Polynomials

```

1 typedef int tp; // type of polynomial
2 template<class T=tp>
3 struct poly { // poly<> : 1 variable, poly<poly<>>: 2 variables, etc.
4     vector<T> c;
5     T& operator[](int k){return c[k];}
6     poly(vector<T>& c):c(c){}
7     poly(initializer_list<T> c):c(c){}
8     poly(int k):c(k){}
9     poly(){}
10    poly operator+(poly<T> o){
11        int m=c.size(),n=o.c.size();
12        poly res(max(m,n));
13        fore(i,0,m)res[i]=res[i]+c[i];
14        fore(i,0,n)res[i]=res[i]+o.c[i];
15        return res;
16    }
17    poly operator*(tp k){
18        poly res(c.size());
19        fore(i,0,c.size())res[i]=c[i]*k;
20        return res;
21    }
22    poly operator*(poly o){
23        int m=c.size(),n=o.c.size();
24        poly res(m+n-1);
25        fore(i,0,m)fore(j,0,n)res[i+j]=res[i+j]+c[i]*o.c[j];
26        return res;
27    }
28    poly operator-(poly<T> o){return *this+(o*-1);}
29    T operator()(tp v){

```

```

30    T sum(0);
31    for(int i=c.size()-1;i>=0;--i)sum=sum*v+c[i];
32    return sum;
33 }
34 };
35 // example: p(x,y)=2*x^2+3*x*y-y+4
36 // poly<poly<>> p={{4,-1},{0,3},{2}}
37 // printf("%d\n",p(2)(3)) // 27 (p(2,3))
38 set<tp> roots(poly<> p){ // only for integer polynomials
39     set<tp> r;
40     while(!p.c.empty()&&!p.c.back())p.c.pop_back();
41     if(!p(0))r.insert(0);
42     if(p.c.empty())return r;
43     tp a0=0,an=abs(p[p.c.size()-1]);
44     for(int k=0;!a0;a0=abs(p[k+1]));
45     vector<tp> ps,qs;
46     fore(i,1,sqrt(a0)+1)if(a0%i==0)ps.pb(i),ps.pb(a0/i);
47     fore(i,1,sqrt(an)+1)if(an%i==0)qs.pb(i),qs.pb(an/i);
48     for(auto pt:ps)for(auto qt:qs)if(pt%qt==0){
49         tp x=pt/qt;
50         if(!p(x))r.insert(x);
51         if(!p(-x))r.insert(-x);
52     }
53     return r;
54 }
55 pair<poly<>,tp> ruffini(poly<> p, tp r){ // returns pair (result,rem)
56     int n=p.c.size()-1;
57     vector<tp> b(n);
58     b[n-1]=p[n];
59     for(int k=n-2;k>=0;--k)b[k]=p[k+1]+r*b[k+1];
60     return {poly<>(b),p[0]+r*b[0]};
61 }
62 // only for double polynomials
63 pair<poly<>,poly<>> polydiv(poly<> p, poly<> q){ // returns pair (
64     result,rem)
65     int n=p.c.size()-q.c.size()+1;
66     vector<tp> b(n);
67     for(int k=n-1;k>=0;--k){
68         b[k]=p.c.back()/q.c.back();
69         fore(i,0,q.c.size())p[i+k]-=b[k]*q[i];
70         p.c.pop_back();
71     }
72     while(!p.c.empty()&&abs(p.c.back())<EPS)p.c.pop_back();

```

```

72     return {poly<>(b),p};
73 }
74 // only for double polynomials
75 poly<> interpolate(vector<tp> x, vector<tp> y){ //TODO TEST
76     poly<> q={1},S={0};
77     for(tp a:x)q=poly<>({-a,1})*q;
78     fore(i,0,x.size()){
79         poly<> Li=ruffini(q,x[i]).fst;
80         Li=Li*(1.0/Li(x[i])); // change for int polynomials
81         S=S+Li*y[i];
82     }
83     return S;
84 }

```

### 3.9 Bairstow

```

1 double pget(poly<>& p, int k){return k<p.c.size()?p[k]:0;}
2 poly<> bairstow(poly<> p){ // returns polynomial of degree 2 that
3     int n=p.c.size()-1; // divides p
4     assert(n>=3&&abs(p.c.back())>EPS);
5     double u=p[n-1]/p[n],v=p[n-2]/p[n];
6     fore(_,0,ITER){
7         auto w=polydiv(p,{v,u,1});
8         poly<> q=w.fst,r0=w.snd;
9         poly<> r1=polydiv(q,{v,u,1}).snd;
10        double c=pget(r0,1),d=pget(r0,0),g=pget(r1,1),h=pget(r1,0);
11        double det=1/(v*g*g+h*(h-u*g)),uu=u;
12        u=-det*(-h*c+g*d);v=-det*(-g*v*c+(g*uu-h)*d);
13    }
14    return {v,u,1};
15 }
16 void addr(vector<double>& r, poly<>& p){
17     assert(p.c.size()<=3);
18     if(p.c.size()<=1)return;
19     if(p.c.size()==2)r.pb(-p[0]/p[1]);
20     if(p.c.size()==3){
21         double a=p[2],b=p[1],c=p[0];
22         double d=b*b-4*a*c;
23         if(d<-0.1)return; // huge epsilon because of bad precision
24         d=d>0?sqrt(d):0;r.pb((-b-d)/2/a);r.pb((-b+d)/2/a);
25     }
26 }
27 }

```

```

28 vector<double> roots(poly<> p){
29     while(!p.c.empty()&&abs(p.c.back())<EPS)p.c.pop_back();
30     fore(i,0,p.c.size())p[i]/=p.c.back();
31     vector<double> r;int n;
32     while((n=p.c.size()-1)>=3){
33         poly<> q=bairstow(p);addr(r,q);
34         p=polydiv(p,q).fst;
35         while(p.c.size()>n-1)p.c.pop_back();
36     }
37     addr(r,p);
38     return r;
39 }

```

### 3.10 Fast Fourier Transform

```

1 // MAXN must be power of 2 !!
2 // MOD-1 needs to be a multiple of MAXN !!
3 // big mod and primitive root for NTT:
4 typedef ll tf;
5 typedef vector<tf> poly;
6 const tf MOD=2305843009255636993,RT=5;
7 // FFT
8 struct CD {
9     double r,i;
10    CD(double r=0, double i=0):r(r),i(i){}
11    double real()const{return r;}
12    void operator/=(const int c){r/=c, i/=c;}
13 };
14 CD operator*(const CD& a, const CD& b){
15     return CD(a.r*b.r-a.i*b.i,a.r*b.i+a.i*b.r);}
16 CD operator+(const CD& a, const CD& b){return CD(a.r+b.r,a.i+b.i);}
17 CD operator-(const CD& a, const CD& b){return CD(a.r-b.r,a.i-b.i);}
18 const double pi=acos(-1.0);
19 // NTT
20 /*
21 struct CD {
22     tf x;
23     CD(tf x):x(x){}
24     CD(){}
25 };
26 CD operator*(const CD& a, const CD& b){return CD(mulmod(a.x,b.x));}
27 CD operator+(const CD& a, const CD& b){return CD(addmod(a.x,b.x));}
28 CD operator-(const CD& a, const CD& b){return CD(submod(a.x,b.x));}

```

```

29 vector<tf> rts(MAXN+9,-1);
30 CD root(int n, bool inv){
31     tf r=rts[n]<0?rts[n]=pm(RT,(MOD-1)/n):rts[n];
32     return CD(inv?pm(r,MOD-2):r);
33 }
34 */
35 CD cp1[MAXN+9],cp2[MAXN+9];
36 int R[MAXN+9];
37 void dft(CD* a, int n, bool inv){
38     fore(i,0,n)if(R[i]<i)swap(a[R[i]],a[i]);
39     for(int m=2;m<=n;m*=2){
40         double z=2*pi/m*(inv?-1:1); // FFT
41         CD wi=CD(cos(z),sin(z)); // FFT
42         // CD wi=root(m,inv); // NTT
43         for(int j=0;j<n;j+=m){
44             CD w(1);
45             for(int k=j,k2=j+m/2;k2<j+m;k++,k2++){
46                 CD u=a[k];CD v=a[k2]*w;a[k]=u+v;a[k2]=u-v;w=w*wi;
47             }
48         }
49     }
50     if(inv)fore(i,0,n)a[i]/=n; // FFT
51     //if(inv){ // NTT
52     //    CD z(pm(n,MOD-2)); // pm: modular exponentiation
53     //    fore(i,0,n)a[i]=a[i]*z;
54     //}
55 }
56 poly multiply(poly& p1, poly& p2){
57     int n=p1.size()+p2.size()+1;
58     int m=1,cnt=0;
59     while(m<=n)m+=m,cnt++;
60     fore(i,0,m){R[i]=0;fore(j,0,cnt)R[i]=(R[i]<<1)|((i>>j)&1);}
61     fore(i,0,m)cp1[i]=0,cp2[i]=0;
62     fore(i,0,p1.size())cp1[i]=p1[i];
63     fore(i,0,p2.size())cp2[i]=p2[i];
64     dft(cp1,m,false);dft(cp2,m,false);
65     fore(i,0,m)cp1[i]=cp1[i]*cp2[i];
66     dft(cp1,m,true);
67     poly res;
68     n-=2;
69     fore(i,0,n)res.pb((tf)floor(cp1[i].real()+0.5)); // FFT
70     //fore(i,0,n)res.pb(cp1[i].x); // NTT
71     return res;

```

```

72 }

```

### 3.11 Fast Hadamard Transform

```

1 ll c1[MAXN+9],c2[MAXN+9]; // MAXN must be power of 2 !!
2 void fht(ll* p, int n, bool inv){
3     for(int l=1;2*l<=n;l*=2)for(int i=0;i<n;i+=2*l)fore(j,0,l){
4         ll u=p[i+j],v=p[i+l+j];
5         if(!inv)p[i+j]=u+v,p[i+l+j]=u-v; // XOR
6         else p[i+j]=(u+v)/2,p[i+l+j]=(u-v)/2;
7         //if(!inv)p[i+j]=v,p[i+l+j]=u+v; // AND
8         //else p[i+j]=-u+v,p[i+l+j]=u;
9         //if(!inv)p[i+j]=u+v,p[i+l+j]=u; // OR
10        //else p[i+j]=v,p[i+l+j]=u-v;
11    }
12 }
13 // like polynomial multiplication, but XORing exponents
14 // instead of adding them (also ANDing, ORing)
15 vector<ll> multiply(vector<ll>& p1, vector<ll>& p2){
16     int n=1<<(32-__builtin_clz(max(SZ(p1),SZ(p2))-1));
17     fore(i,0,n)c1[i]=0,c2[i]=0;
18     fore(i,0,SZ(p1))c1[i]=p1[i];
19     fore(i,0,SZ(p2))c2[i]=p2[i];
20     fht(c1,n,false);fht(c2,n,false);
21     fore(i,0,n)c1[i]*=c2[i];
22     fht(c1,n,true);
23     return vector<ll>(c1,c1+n);
24 }

```

### 3.12 Karatsuba

```

1 typedef ll tp;
2 #define add(n,s,d,k) fore(i,0,n)(d)[i]+=(s)[i]*k
3 tp* ini(int n){tp *r=new tp[n];fill(r,r+n,0);return r;}
4 void karatsura(int n, tp* p, tp* q, tp* r){
5     if(n<=0)return;
6     if(n<35)fore(i,0,n)fore(j,0,n)r[i+j]+=p[i]*q[j];
7     else {
8         int nac=n/2,nbd=n-n/2;
9         tp *a=p,*b=p+nac,*c=q,*d=q+nac;
10        tp *ab=ini(nbd+1),*cd=ini(nbd+1),*ac=ini(nac*2),*bd=ini(nbd*2);
11        add(nac,a,ab,1);add(nbd,b,ab,1);
12        add(nac,c,cd,1);add(nbd,d,cd,1);
13        karatsura(nac,a,c,ac);karatsura(nbd,b,d,bd);

```

```

14     add(nac*2,ac,r+nac,-1);add(nbd*2,bd,r+nac,-1);
15     add(nac*2,ac,r,1);add(nbd*2,bd,r+nac*2,1);
16     karatsura(nbd+1,ab,cd,r+nac);
17     free(ab);free(cd);free(ac);free(bd);
18 }
19 }
20 vector<tp> multiply(vector<tp> p0, vector<tp> p1){
21     int n=max(p0.size(),p1.size());
22     tp *p=ini(n),*q=ini(n),*r=ini(2*n);
23     fore(i,0,p0.size())p[i]=p0[i];
24     fore(i,0,p1.size())q[i]=p1[i];
25     karatsura(n,p,q,r);
26     vector<tp> rr(r,r+p0.size()+p1.size()-1);
27     free(p);free(q);free(r);
28     return rr;
29 }

```

### 3.13 Diophantine

```

1 pair<ll,ll> extendedEuclid (ll a, ll b){ //a * x + b * y = gcd(a,b)
2     ll x,y;
3     if (b==0) return {1,0};
4     auto p=extendedEuclid(b,a%b);
5     x=p.snd;
6     y=p.fst-(a/b)*x;
7     if(a*x+b*y==gcd(a,b)) x=-x, y=-y;
8     return {x,y};
9 }
10 pair<pair<ll,ll>,pair<ll,ll> > diophantine(ll a,ll b, ll r) {
11     //a*x+b*y=r where r is multiple of gcd(a,b);
12     ll d=gcd(a,b);
13     a/=d; b/=d; r/=d;
14     auto p = extendedEuclid(a,b);
15     p.fst*=r; p.snd*=r;
16     assert(a*p.fst+b*p.snd==r);
17     return {p,{-b,a}}; // solutions: p+t*ans.snd
18 }

```

### 3.14 Modular inverse

```

1 ll inv(ll a, ll mod) { //inverse of a modulo mod
2     assert(gcd(a,mod)==1);
3     pl sol = extendedEuclid(a,mod);
4     return ((sol.fst%mod)+mod)%mod;

```

```

5 }

```

### 3.15 Chinese remainder theorem

```

1 #define mod(a,m) (((a)%m+m)%m)
2 pair<ll,ll> sol(tuple<ll,ll,ll> c){ //requires inv, diophantine
3     ll a=get<0>(c), x1=get<1>(c), m=get<2>(c), d=gcd(a,m);
4     if(d==1) return {mod(x1*inv(a,m),m), m};
5     else return x1%d ? ii({-1LL,-1LL}) : sol(make_tuple(a/d,x1/d,m/d));
6 }
7 pair<ll,ll> crt(vector< tuple<ll,ll,ll> > cond) { // returns: (sol, lcm)
8     ll x1=0,m1=1,x2,m2;
9     for(auto t:cond){
10         tie(x2,m2)=sol(t);
11         if((x1-x2)%gcd(m1,m2))return {-1,-1};
12         if(m1==m2)continue;
13         ll k=diophantine(m2,-m1,x1-x2).fst.snd,l=m1*(m2/gcd(m1,m2));
14         x1=mod((__int128)m1*k+x1,1);m1=l;
15     }
16     return sol(make_tuple(1,x1,m1));
17 } //cond[i]={ai,bi,mi} ai*xi=bi (mi); assumes lcm fits in ll

```

### 3.16 Mobius

```

1 short mu[MAXN] = {0,1};
2 void mobius(){
3     fore(i,1,MAXN)if(mu[i])for(int j=i+i;j<MAXN;j+=i)mu[j]-=mu[i];
4 }

```

### 3.17 Matrix exponentiation

```

1 typedef vector<vector<ll> > Matrix;
2 Matrix ones(int n) {
3     Matrix r(n,vector<ll>(n));
4     fore(i,0,n)r[i][i]=1;
5     return r;
6 }
7 Matrix operator*(Matrix &a, Matrix &b) {
8     int n=SZ(a),m=SZ(b[0]),z=SZ(a[0]);
9     Matrix r(n,vector<ll>(m));
10    fore(i,0,n)fore(j,0,m)fore(k,0,z)
11        r[i][j]+=a[i][k]*b[k][j],r[i][j]%mod;
12    return r;
13 }

```

```

14 Matrix be(Matrix b, ll e) {
15     Matrix r=ones(SZ(b));
16     while(e){if(e&1LL)r=r*b;b=b*b;e/=2;}
17     return r;
18 }
    
```

### 3.18 Matrix reduce and determinant

```

1 double reduce(vector<vector<double> >& x){ // returns determinant
2     int n=x.size(),m=x[0].size();
3     int i=0,j=0;double r=1.;
4     while(i<n&&j<m){
5         int l=i;
6         fore(k,i+1,n)if(abs(x[k][j])>abs(x[l][j]))l=k;
7         if(abs(x[l][j])<EPS){j++;r=0.;continue;}
8         if(l!=i){r=-r;swap(x[i],x[l]);}
9         r*=x[i][j];
10        for(int k=m-1;k>=j;k--)x[i][k]/=x[i][j];
11        fore(k,0,n){
12            if(k==i)continue;
13            for(int l=m-1;l>=j;l--)x[k][l]-=x[k][j]*x[i][l];
14        }
15        i++;j++;
16    }
17    return r;
18 }
    
```

### 3.19 Simplex

```

1 vector<int> X,Y;
2 vector<vector<double> > A;
3 vector<double> b,c;
4 double z;
5 int n,m;
6 void pivot(int x,int y){
7     swap(X[Y],Y[X]);
8     b[X]/=A[X][Y];
9     fore(i,0,m)if(i!=y)A[X][i]/=A[X][Y];
10    A[X][Y]=1/A[X][Y];
11    fore(i,0,n)if(i!=x&&abs(A[i][Y])>EPS){
12        b[i]-=A[i][Y]*b[X];
13        fore(j,0,m)if(j!=y)A[i][j]-=A[i][Y]*A[X][j];
14        A[i][Y]=-A[i][Y]*A[X][Y];
15    }
    
```

```

16 z+=c[Y]*b[X];
17 fore(i,0,m)if(i!=y)c[i]-=c[Y]*A[X][i];
18 c[Y]=-c[Y]*A[X][Y];
19 }
20 pair<double,vector<double> > simplex( // maximize c^T x s.t. Ax<=b, x>=0
21     vector<vector<double> > _A, vector<double> _b, vector<double> _c){
22     // returns pair (maximum value, solution vector)
23     A=_A;b=_b;c=_c;
24     n=b.size();m=c.size();z=0.;
25     X=vector<int>(m);Y=vector<int>(n);
26     fore(i,0,m)X[i]=i;
27     fore(i,0,n)Y[i]=i+m;
28     while(1){
29         int x=-1,y=-1;
30         double mn=-EPS;
31         fore(i,0,n)if(b[i]<mn)mn=b[i],x=i;
32         if(x<0)break;
33         fore(i,0,m)if(A[X][i]<-EPS){y=i;break;}
34         assert(y>=0); // no solution to Ax<=b
35         pivot(x,y);
36     }
37     while(1){
38         double mx=EPS;
39         int x=-1,y=-1;
40         fore(i,0,m)if(c[i]>mx)mx=c[i],y=i;
41         if(y<0)break;
42         double mn=1e200;
43         fore(i,0,n)if(A[i][Y]>EPS&&b[i]/A[i][Y]<mn)mn=b[i]/A[i][Y],x=i;
44         assert(x>=0); // c^T x is unbounded
45         pivot(x,y);
46     }
47     vector<double> r(m);
48     fore(i,0,n)if(Y[i]<m)r[Y[i]]=b[i];
49     return {z,r};
50 }
    
```

### 3.20 Discrete log

```

1 //returns x such that a^x = b (mod m) or -1 if inexistent
2 ll discrete_log(ll a,ll b,ll m) {
3     a%=m, b%=m;
4     if(b == 1) return 0;
5     int cnt=0;
    
```

```

6   ll tmp=1;
7   for(int g=__gcd(a,m);g!=1;g=__gcd(a,m)) {
8       if(b%g) return -1;
9       m/=g, b/=g;
10      tmp = tmp*a/g%m;
11      ++cnt;
12      if(b == tmp) return cnt;
13  }
14  map<ll,int> w;
15  int s = ceil(sqrt(m));
16  ll base = b;
17  fore(i,0,s) {
18      w[base] = i;
19      base=base*a%m;
20  }
21  base=fastpow(a,s,m);
22  ll key=tmp;
23  fore(i,1,s+2) {
24      key=base*key%m;
25      if(w.count(key)) return i*s-w[key]+cnt;
26  }
27  return -1;
28 }

```

### 3.21 Berlekamp Massey

```

1  typedef vector<int> vi;
2  vi BM(vi x){
3      vi ls,cur;int lf,ld;
4      fore(i,0,SZ(x)){
5          ll t=0;
6          fore(j,0,SZ(cur))t=(t+x[i-j-1]*(ll)cur[j])%MOD;
7          if((t-x[i])%MOD==0)continue;
8          if(!SZ(cur)){cur.resize(i+1);lf=i;ld=(t-x[i])%MOD;continue;}
9          ll k=-(x[i]-t)*fast_pow(ld,MOD-2)%MOD;
10         vi c(i-lf-1);c.pb(k);
11         fore(j,0,SZ(ls))c.pb((-ls[j]*k)%MOD);
12         if(SZ(c)<SZ(cur))c.resize(SZ(cur));
13         fore(j,0,SZ(cur))c[j]=(c[j]+cur[j])%MOD;
14         if(i-lf+SZ(ls)>=SZ(cur))ls=cur,lf=i,ld=(t-x[i])%MOD;
15         cur=c;
16     }
17     fore(i,0,SZ(cur))cur[i]=(cur[i]%MOD+MOD)%MOD;

```

```

18     return cur;
19 }

```

### 3.22 Linear Rec

```

1  //init O(n^2log) query(n^2 logk)
2  //input: terms: first n term; trans: transition function; MOD; LOG=mxlog
3  //output calc(k): kth term mod MOD
4  //example: {1,1} {2,1} an=2*a_(n-1)+a_(n-2); calc(3)=3 calc(10007)
5  //          =71480733
6  struct LinearRec{
7      typedef vector<int> vi;
8      int n; vi terms, trans; vector<vi> bin;
9      vi add(vi &a, vi &b){
10         vi res(n*2+1);
11         fore(i,0,n+1)fore(j,0,n+1)res[i+j]=(res[i+j]*1LL+(ll)a[i]*b[j])%MOD;
12         for(int i=2*n; i>n; --i){
13             fore(j,0,n)res[i-1-j]=(res[i-1-j]*1LL+(ll)res[i]*trans[j])%MOD;
14             res[i]=0;
15         }
16         res.erase(res.begin()+n+1,res.end());
17         return res;
18     }
19     LinearRec(vi &terms, vi &trans):terms(terms),trans(trans){
20         n=SZ(trans);vi a(n+1);a[1]=1;
21         bin.pb(a);
22         fore(i,1,LOG)bin.pb(add(bin[i-1],bin[i-1]));
23     }
24     int calc(int k){
25         vi a(n+1);a[0]=1;
26         fore(i,0,LOG)if((k>>i)&1)a=add(a,bin[i]);
27         int ret=0;
28         fore(i,0,n)ret=((ll)ret+(ll)a[i+1]*terms[i])%MOD;
29         return ret;
30     };

```

### 3.23 Tonelli Shanks

```

1  ll legendre(ll a, ll p){
2      if(a%p==0)return 0; if(p==2)return 1;
3      return fpow(a,(p-1)/2,p);
4  }

```

```

5 ll tonelli_shanks(ll n, ll p){ // sqrt(n) mod p (p must be a prime)
6   assert(legendre(n,p)==1); if(p==2)return 1;
7   ll s=__builtin_ctzll(p-1), q=(p-1LL)>>s, z=rnd(1,p-1);
8   if(s==1)return fpow(n,(p+1)/4LL,p);
9   while(legendre(z,p)!=p-1)z=rnd(1,p-1);
10  ll c=fpow(z,q,p), r=fpow(n,(q+1)/2,p), t=fpow(n,q,p), m=s;
11  while(t!=1){
12    ll i=1, ts=(t*t)%p;
13    while(ts!=1)i++,ts=(ts*ts)%p;
14    ll b=c;
15    fore(_,0,m-i-1)b=(b*b)%p;
16    r=r*b%p;c=b*b%p;t=t*c%p;m=i;
17  }
18  return r;
19 }

```

## 4 Geometry

### 4.1 Point

```

1 struct pt { // for 3D add z coordinate
2   double x,y;
3   pt(double x, double y):x(x),y(y){}
4   pt(){}
5   double norm2(){return *this**this;}
6   double norm(){return sqrt(norm2());}
7   bool operator==(pt p){return abs(x-p.x)<=EPS&&abs(y-p.y)<=EPS;}
8   pt operator+(pt p){return pt(x+p.x,y+p.y);}
9   pt operator-(pt p){return pt(x-p.x,y-p.y);}
10  pt operator*(double t){return pt(x*t,y*t);}
11  pt operator/(double t){return pt(x/t,y/t);}
12  double operator*(pt p){return x*p.x+y*p.y;}
13  // pt operator^(pt p){ // only for 3D
14  //   return pt(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y*p.x);}
15  double angle(pt p){ // redefine acos for values out of range
16    return acos(*this*p/(norm()*p.norm()));}
17  pt unit(){return *this/norm();}
18  double operator%(pt p){return x*p.y-y*p.x;}
19  // 2D from now on
20  bool operator<(pt p)const{ // for convex hull
21    return x<p.x-EPS|| (abs(x-p.x)<=EPS&&y<p.y-EPS);}
22  bool left(pt p, pt q){ // is it to the left of directed line pq?
23    return (q-p)%(*this-p)>EPS;}

```

```

24 pt rot(pt r){return pt(*this/r,*this*r);}
25 pt rot(double a){return rot(pt(sin(a),cos(a)));}
26 };
27 pt ccw90(1,0);
28 pt cw90(-1,0);

```

### 4.2 Line

```

1 int sgn2(double x){return x<0?-1:1;}
2 struct ln {
3   pt p,pq;
4   ln(pt p, pt q):p(p),pq(q-p){}
5   ln(){}
6   bool has(pt r){return dist(r)<=EPS;}
7   bool seghas(pt r){return has(r)&&(r-p)*(r-(p+pq))<=EPS;}
8   // bool operator/(ln l){return (pq.unit()^l.pq.unit()).norm()<=EPS;}
9   // 3D
10  bool operator/(ln l){return abs(pq.unit()^l.pq.unit())<=EPS;} // 2D
11  bool operator==(ln l){return *this/l&&has(l.p);}
12  pt operator^(ln l){ // intersection
13    if(*this/l)return pt(DINF,DINF);
14    pt r=l.p+l.pq*((p-l.p)%pq/(l.pq%pq));
15    // if(!has(r)){return pt(NAN,NAN,NAN);} // check only for 3D
16    return r;
17  }
18  double angle(ln l){return pq.angle(l.pq);}
19  int side(pt r){return has(r)?0:sgn2(pq%(r-p));} // 2D
20  pt proj(pt r){return p+pq*((r-p)*pq/pq.norm2());}
21  pt ref(pt r){return proj(r)*2-r;}
22  double dist(pt r){return (r-proj(r)).norm();}
23  // double dist(ln l){ // only 3D
24  //   if(*this/l)return dist(l.p);
25  //   return abs((l.p-p)*(pq^l.pq))/(pq^l.pq).norm();
26  // }
27  ln rot(auto a){return ln(p,p+pq.rot(a));} // 2D
28 };
29 ln bisector(ln l, ln m){ // angle bisector
30   pt p=l^m;
31   return ln(p,p+l.pq.unit()+m.pq.unit());
32 }
33 ln bisector(pt p, pt q){ // segment bisector (2D)
34   return ln((p+q)*.5,p).rot(ccw90);
35 }

```



## 4.3 Circle

```

1 struct circle {
2     pt o; double r;
3     circle(pt o, double r):o(o),r(r){}
4     circle(pt x, pt y, pt z){o=bisector(x,y)^bisector(x,z);r=(o-x).norm()
5         ;}
6     bool has(pt p){return (o-p).norm()<=r+EPS;}
7     vector<pt> operator^(circle c){ // ccw
8         vector<pt> s;
9         double d=(o-c.o).norm();
10        if(d>r+c.r+EPS||d+min(r,c.r)+EPS<max(r,c.r))return s;
11        double x=(d*d-c.r*c.r+r*r)/(2*d);
12        double y=sqrt(r*r-x*x);
13        pt v=(c.o-o)/d;
14        s.pb(o+v*x-v.rot(ccw90)*y);
15        if(y>EPS)s.pb(o+v*x+v.rot(ccw90)*y);
16        return s;
17    }
18    vector<pt> operator^(ln l){
19        vector<pt> s;
20        pt p=l.proj(o);
21        double d=(p-o).norm();
22        if(d-EPS>r)return s;
23        if(abs(d-r)<=EPS){s.pb(p);return s;}
24        d=sqrt(r*r-d*d);
25        s.pb(p+l.pq.unit()*d);
26        s.pb(p-l.pq.unit()*d);
27        return s;
28    }
29    vector<pt> tang(pt p){
30        double d=sqrt((p-o).norm2()-r*r);
31        return *this^circle(p,d);
32    }
33    bool in(circle c){ // non strict
34        double d=(o-c.o).norm();
35        return d+r<=c.r+EPS;
36    }
37    double intertriangle(pt a, pt b){ // area of intersection with oab
38        if(abs((o-a)%(o-b))<=EPS)return 0.;
39        vector<pt> q={a},w=*this^ln(a,b);
40        if(w.size()==2)for(auto p:w)if((a-p)*(b-p)<-EPS)q.pb(p);
41        q.pb(b);

```

```

41    if(q.size()==4&&(q[0]-q[1])*(q[2]-q[1])>EPS)swap(q[1],q[2]);
42    double s=0;
43    for(i,0,q.size()-1){
44        if(!has(q[i])||!has(q[i+1]))s+=r*r*(q[i]-o).angle(q[i+1]-o)/2;
45        else s+=abs((q[i]-o)%(q[i+1]-o)/2);
46    }
47    return s;
48 }
49 };
50 vector<double> intercircles(vector<circle> c){
51     vector<double> r(SZ(c)+1); // r[k]: area covered by at least k circles
52     for(i,0,SZ(c)){ // O(n^2 log n) (high constant)
53         int k=1; Cmp s(c[i].o);
54         vector<pair<pt,int> > p={
55             {c[i].o+pt(1,0)*c[i].r,0},
56             {c[i].o-pt(1,0)*c[i].r,0}};
57         for(j,0,SZ(c))if(j!=i){
58             bool b0=c[i].in(c[j]),b1=c[j].in(c[i]);
59             if(b0&&(!b1||i<j))k++;
60             else if(!b0&&!b1){
61                 auto v=c[i]^c[j];
62                 if(SZ(v)==2){
63                     p.pb({v[0],1});p.pb({v[1],-1});
64                     if(s(v[1],v[0]))k++;
65                 }
66             }
67         }
68         sort(p.begin(),p.end(),
69             [&](pair<pt,int> a, pair<pt,int> b){return s(a.fst,b.fst);});
70         for(j,0,SZ(p)){
71             pt p0=p[j?j-1:SZ(p)-1].fst,p1=p[j].fst;
72             double a=(p0-c[i].o).angle(p1-c[i].o);
73             r[k]+=(p0.x-p1.x)*(p0.y+p1.y)/2+c[i].r*c[i].r*(a-sin(a))/2;
74             k+=p[j].snd;
75         }
76     }
77     return r;
78 }

```

## 4.4 Polygon

```

1 int sgn(double x){return x<-EPS?-1:x>EPS;}
2 struct pol {

```

```

3  int n;vector<pt> p;
4  pol(){
5  pol(vector<pt> _p){p=_p;n=p.size();}
6  double area(){
7      double r=0.;
8      fore(i,0,n)r+=p[i]%p[(i+1)%n];
9      return abs(r)/2; // negative if CW, positive if CCW
10 }
11 pt centroid(){ // (barycenter)
12     pt r(0,0);double t=0;
13     fore(i,0,n){
14         r=r+(p[i]+p[(i+1)%n])*(p[i]%p[(i+1)%n]);
15         t+=p[i]%p[(i+1)%n];
16     }
17     return r/t/3;
18 }
19 bool has(pt q){ // O(n)
20     fore(i,0,n)if(ln(p[i],p[(i+1)%n]).seghas(q))return true;
21     int cnt=0;
22     fore(i,0,n){
23         int j=(i+1)%n;
24         int k=sgn((q-p[j])%(p[i]-p[j]));
25         int u=sgn(p[i].y-q.y),v=sgn(p[j].y-q.y);
26         if(k>0&&u<0&&v>=0)cnt++;
27         if(k<0&&v<0&&u>=0)cnt--;
28     }
29     return cnt!=0;
30 }
31 void normalize(){ // (call before haslog, remove collinear first)
32     if(p[2].left(p[0],p[1]))reverse(p.begin(),p.end());
33     int pi=min_element(p.begin(),p.end())-p.begin();
34     vector<pt> s(n);
35     fore(i,0,n)s[i]=p[(pi+i)%n];
36     p.swap(s);
37 }
38 bool haslog(pt q){ // O(log(n)) only CONVEX. Call normalize first
39     if(q.left(p[0],p[1])||q.left(p.back(),p[0]))return false;
40     int a=1,b=p.size()-1; // returns true if point on boundary
41     while(b-a>1){ // (change sign of EPS in left
42         int c=(a+b)/2; // to return false in such case)
43         if(!q.left(p[0],p[c]))a=c;
44         else b=c;
45     }

```

```

46     return !q.left(p[a],p[a+1]);
47 }
48 pt farthest(pt v){ // O(log(n)) only CONVEX
49     if(n<10){
50         int k=0;
51         fore(i,1,n)if(v*(p[i]-p[k])>EPS)k=i;
52         return p[k];
53     }
54     if(n==SZ(p))p.pb(p[0]);
55     pt a=p[1]-p[0];
56     int s=0,e=n,ua=v*a>EPS;
57     if(!ua&&v*(p[n-1]-p[0])<=EPS)return p[0];
58     while(1){
59         int m=(s+e)/2;pt c=p[m+1]-p[m];
60         int uc=v*c>EPS;
61         if(!uc&&v*(p[m-1]-p[m])<=EPS)return p[m];
62         if(ua&&(!uc||v*(p[s]-p[m])>EPS))e=m;
63         else if(ua||uc||v*(p[s]-p[m])>=-EPS)s=m,a=c,ua=uc;
64         else e=m;
65         assert(e>s+1);
66     }
67 }
68 pol cut(ln l){ // cut CONVEX polygon by line l
69     vector<pt> q; // returns part at left of l.pq
70     fore(i,0,n){
71         int d0=sgn(l.pq%(p[i]-l.p)),d1=sgn(l.pq%(p[(i+1)%n]-l.p));
72         if(d0>=0)q.pb(p[i]);
73         ln m(p[i],p[(i+1)%n]);
74         if(d0*d1<0&&!(1/m))q.pb(l^m);
75     }
76     return pol(q);
77 }
78 double intercircle(circle c){ // area of intersection with circle
79     double r=0.;
80     fore(i,0,n){
81         int j=(i+1)%n;double w=c.intertriangle(p[i],p[j]);
82         if((p[j]-c.o)%(p[i]-c.o)>0)r+=w;
83         else r-=w;
84     }
85     return abs(r);
86 }
87 double callipers(){ // square distance of most distant points
88     double r=0; // prereq: convex, ccw, NO COLLINEAR POINTS

```

```

89     for(int i=0,j=n<2?0:1;i<j;++i){
90         for(;;j=(j+1)%n){
91             r=max(r,(p[i]-p[j]).norm2());
92             if((p[(i+1)%n]-p[i])%(p[(j+1)%n]-p[j])<=EPS)break;
93         }
94     }
95     return r;
96 }
97 };
98 // Dynamic convex hull trick
99 vector<pol> w;
100 void add(pt q){ // add(q), O(log^2(n))
101     vector<pt> p={q};
102     while(!w.empty()&&SZ(w.back().p)<2*SZ(p)){
103         for(pt v:w.back().p)p.pb(v);
104         w.pop_back();
105     }
106     w.pb(pol(chull(p)));
107 }
108 ll query(pt v){ // max(q*v:q in w), O(log^2(n))
109     ll r=-INF;
110     for(auto& p:w)r=max(r,p.farthest(v)*v);
111     return r;
112 }

```

#### 4.5 Plane

```

1 struct plane {
2     pt a,n; // n: normal unit vector
3     plane(pt a, pt b, pt c):a(a),n(((b-a)^(c-a)).unit()){
4         plane(){}
5     bool has(pt p){return abs((p-a)*n)<=EPS;}
6     double angle(plane w){return acos(n*w.n);}
7     double dist(pt p){return abs((p-a)*n);}
8     pt proj(pt p){inter(ln(p,p+n),p);return p;}
9     bool inter(ln l, pt& r){
10         double x=n*(l.p+l.pq-a),y=n*(l.p-a);
11         if(abs(x-y)<=EPS)return false;
12         r=(l.p*x-(l.p+l.pq)*y)/(x-y);
13         return true;
14     }
15     bool inter(plane w, ln& r){
16         pt nn=n^w.n;pt v=n^nn;double d=w.n*v;

```

```

17         if(abs(d)<=EPS)return false;
18         pt p=a+v*(w.n*(w.a-a)/d);
19         r=ln(p,p+nn);
20         return true;
21     }
22 };

```

#### 4.6 Radial order of points

```

1 struct Cmp { // IMPORTANT: add const in pt operator -
2     pt r;
3     Cmp(pt r):r(r){}
4     int cuad(const pt &a)const {
5         if(a.x>0&&a.y>=0)return 0;
6         if(a.x<=0&&a.y>0)return 1;
7         if(a.x<0&&a.y<=0)return 2;
8         if(a.x>=0&&a.y<0)return 3;
9         assert(a.x==0&&a.y==0);
10        return -1;
11    }
12    bool cmp(const pt& p1, const pt& p2)const {
13        int c1=cuad(p1),c2=cuad(p2);
14        if(c1==c2)return p1.y*p2.x<p1.x*p2.y;
15        return c1<c2;
16    }
17    bool operator()(const pt& p1, const pt& p2)const {
18        return cmp(p1-r,p2-r);
19    }
20 };

```

#### 4.7 Convex hull

```

1 // CCW order
2 // Includes collinear points (change sign of EPS in left to exclude)
3 vector<pt> chull(vector<pt> p){
4     if(SZ(p)<3)return p;
5     vector<pt> r;
6     sort(p.begin(),p.end()); // first x, then y
7     fore(i,0,p.size()){ // lower hull
8         while(r.size()>=2&&r.back().left(r[r.size()-2],p[i]))r.pop_back();
9         r.pb(p[i]);
10    }
11    r.pop_back();
12    int k=r.size();

```

```

13 for(int i=p.size()-1;i>=0;--i){ // upper hull
14     while(r.size()>=k+2&& r.back().left(r[r.size()-2],p[i]))r.pop_back();
15     r.pb(p[i]);
16 }
17 r.pop_back();
18 return r;
19 }
    
```

## 4.8 Dual from planar graph

```

1 vector<int> g[MAXN];int n; // input graph (must be connected)
2 vector<int> gd[MAXN];int nd; // output graph
3 vector<int> nodes[MAXN]; // nodes delimiting region (in CW order)
4 map<pair<int,int>,int> ps,es;
5 void get_dual(vector<pt> p){ // p: points corresponding to nodes
6     ps.clear();es.clear();
7     fore(x,0,n){
8         Cmp pc(p[x]); // (radial order of points)
9         auto comp=[&](int a, int b){return pc(p[a],p[b]);};
10        sort(g[x].begin(),g[x].end(),comp);
11        fore(i,0,g[x].size())ps[{x,g[x][i]}]=i;
12    }
13    nd=0;
14    fore(xx,0,n)for(auto yy:g[xx])if(!es.count({xx,yy})){
15        int x=xx,y=yy;gd[nd].clear();nodes[nd].clear();
16        while(!es.count({x,y})){
17            es[{x,y}]=nd;nodes[nd].pb(y);
18            int z=g[y][(ps[{y,x}]+1)%g[y].size()];x=y;y=z;
19        }
20        nd++;
21    }
22    for(auto p:es){
23        pair<int,int> q={p.fst.snd,p.fst.fst};
24        assert(es.count(q));
25        if(es[q]!=p.snd)gd[p.snd].pb(es[q]);
26    }
27    fore(i,0,nd){
28        sort(gd[i].begin(),gd[i].end());
29        gd[i].erase(unique(gd[i].begin(),gd[i].end()),gd[i].end());
30    }
31 }
    
```

## 4.9 Halfplane intersection

```

1 // polygon intersecting left side of halfplanes
2 struct halfplane:public ln{
3     double angle;
4     halfplane(){}
5     halfplane(pt a,pt b){p=a; pq=b-a; angle=atan2(pq.y,pq.x);}
6     bool operator<(halfplane b)const{return angle<b.angle;}
7     bool out(pt q){return pq%(q-p)<=-EPS;}
8 };
9 vector<pt> intersect(vector<halfplane> b){
10     vector<pt>bx={{DINF,DINF},{-DINF,DINF},{-DINF,-DINF},{DINF,-DINF}};
11     fore(i,0,4) b.pb(halfplane(bx[i],bx[(i+1)%4]));
12     sort(ALL(b));
13     int n=SZ(b),q=1,h=0;
14     vector<halfplane> c(SZ(b)+10);
15     fore(i,0,n){
16         while(q<h&&b[i].out(c[h]^c[h-1])) h--;
17         while(q<h&&b[i].out(c[q]^c[q+1])) q++;
18         c[++h]=b[i];
19         if(q<h&&abs(c[h].pq*c[h-1].pq)<EPS){
20             if(c[h].pq*c[h-1].pq<=0) return {};
21             h--;
22             if(b[i].out(c[h].p)) c[h]=b[i];
23         }
24     }
25     while(q<h-1&&c[q].out(c[h]^c[h-1]))h--;
26     while(q<h-1&&c[h].out(c[q]^c[q+1]))q++;
27     if(h-q<=1)return {};
28     c[h+1]=c[q];
29     vector<pt> s;
30     fore(i,q,h+1) s.pb(c[i]^c[i+1]);
31     return s;
32 }
    
```

## 5 Strings

### 5.1 KMP

```

1 vector<int> kmppre(string& t){ // r[i]: longest border of t[0,i]
2     vector<int> r(t.size()+1);r[0]=-1;
3     int j=-1;
4     fore(i,0,t.size()){
5         while(j>=0&&t[i]!=t[j])j=r[j];
6         r[i+1]=++j;
    }
    
```

```

7   }
8   return r;
9 }
10 void kmp(string& s, string& t){ // find t in s
11     int j=0;vector<int> b=kmppre(t);
12     fore(i,0,s.size()){
13         while(j>=0&&s[i]!=t[j])j=b[j];
14         if(++j==t.size())printf("Match at %d\n",i-j+1),j=b[j];
15     }
16 }

```

## 5.2 Z function

```

1 vector<int> z_function(string& s){
2     int l=0,r=0,n=s.size();
3     vector<int> z(s.size(),0); // z[i] = max k: s[0,k] == s[i,i+k]
4     fore(i,1,n){
5         if(i<=r)z[i]=min(r-i+1,z[i-l]);
6         while(i+z[i]<n&&s[z[i]]==s[i+z[i]])z[i]++;
7         if(i+z[i]-1>r)l=i,r=i+z[i]-1;
8     }
9     return z;
10 }

```

## 5.3 Manacher

```

1 int d1[MAXN]; //d1[i] = max odd palindrome centered on i
2 int d2[MAXN]; //d2[i] = max even palindrome centered on i
3 //s aabbaacaabbaa
4 //d1 111111711111
5 //d2 0103010010301
6 void manacher(string& s){
7     int l=0,r=-1,n=s.size();
8     fore(i,0,n){
9         int k=i>r?1:min(d1[l+r-i],r-i);
10        while(i+k<n&&i-k>=0&&s[i+k]==s[i-k])k++;
11        d1[i]=k--;
12        if(i+k>r)l=i-k,r=i+k;
13    }
14    l=0;r=-1;
15    fore(i,0,n){
16        int k=i>r?0:min(d2[l+r-i+1],r-i+1);k++;
17        while(i+k<=n&&i-k>=0&&s[i+k-1]==s[i-k])k++;
18        d2[i]=--k;

```

```

19     if(i+k-1>r)l=i-k,r=i+k-1;
20 }
21 }

```

## 5.4 Aho-Corasick

```

1 struct vertex {
2     map<char,int> next,go;
3     int p,link;
4     char pch;
5     vector<int> leaf;
6     vertex(int p=-1, char pch=-1):p(p),pch(pch),link(-1){}
7 };
8 vector<vertex> t;
9 void aho_init(){ //do not forget!!
10     t.clear();t.pb(vertex());
11 }
12 void add_string(string s, int id){
13     int v=0;
14     for(char c:s){
15         if(!t[v].next.count(c)){
16             t[v].next[c]=t.size();
17             t.pb(vertex(v,c));
18         }
19         v=t[v].next[c];
20     }
21     t[v].leaf.pb(id);
22 }
23 int go(int v, char c);
24 int get_link(int v){
25     if(t[v].link<0)
26         if(!v||!t[v].p)t[v].link=0;
27         else t[v].link=go(get_link(t[v].p),t[v].pch);
28     return t[v].link;
29 }
30 int go(int v, char c){
31     if(!t[v].go.count(c))
32         if(t[v].next.count(c))t[v].go[c]=t[v].next[c];
33         else t[v].go[c]=v==0?0:go(get_link(v),c);
34     return t[v].go[c];
35 }

```

## 5.5 Suffix automaton

```

1 struct state {int len,link;map<char,int> next;}; //clear next!!
2 state st[100005];
3 int sz,last;
4 void sa_init(){
5     last=st[0].len=0;sz=1;
6     st[0].link=-1;
7 }
8 void sa_extend(char c){
9     int k=sz++,p;
10    st[k].len=st[last].len+1;
11    for(p=last;p!=-1&&!st[p].next.count(c);p=st[p].link)st[p].next[c]=k;
12    if(p==-1)st[k].link=0;
13    else {
14        int q=st[p].next[c];
15        if(st[p].len+1==st[q].len)st[k].link=q;
16        else {
17            int w=sz++;
18            st[w].len=st[p].len+1;
19            st[w].next=st[q].next;st[w].link=st[q].link;
20            for(;p!=-1&&st[p].next[c]==q;p=st[p].link)st[p].next[c]=w;
21            st[q].link=st[k].link=w;
22        }
23    }
24    last=k;
25 }
26 // input: abcbcbc
27 // i,link,len,next
28 // 0 -1 0 (a,1) (b,5) (c,7)
29 // 1 0 1 (b,2)
30 // 2 5 2 (c,3)
31 // 3 7 3 (b,4)
32 // 4 9 4 (c,6)
33 // 5 0 1 (c,7)
34 // 6 11 5 (b,8)
35 // 7 0 2 (b,9)
36 // 8 9 6 (c,10)
37 // 9 5 3 (c,11)
38 // 10 11 7
39 // 11 7 4 (b,8)

```

## 5.6 Palindromic Tree

```

1 struct palindromic_tree{

```

```

2     static const int SIGMA=26;
3     struct Node{
4         int len, link, to[SIGMA];
5         ll cnt;
6         Node(int len, int link=0, ll cnt=1):len(len),link(link),cnt(cnt)
7         {
8             memset(to,0,sizeof(to));
9         }
10    };
11    vector<Node> ns;
12    int last;
13    palindromic_tree():last(0){ns.pb(Node(-1));ns.pb(Node(0));}
14    void add(int i, string &s){
15        int p=last, c=s[i]-'a';
16        while(s[i-ns[p].len-1]!=s[i])p=ns[p].link;
17        if(ns[p].to[c]){
18            last=ns[p].to[c];
19            ns[last].cnt++;
20        }else{
21            int q=ns[p].link;
22            while(s[i-ns[q].len-1]!=s[i])q=ns[q].link;
23            q=max(1,ns[q].to[c]);
24            last=ns[p].to[c]=SZ(ns);
25            ns.pb(Node(ns[p].len+2,q,1));
26        }
27    };

```

## 5.7 Suffix array (shorter but slower)

```

1 pair<int, int> sf[MAXN];
2 bool sacomp(int lhs, int rhs) {return sf[lhs]<sf[rhs];}
3 vector<int> constructSA(string& s){ // O(n log^2(n))
4     int n=s.size(); // (sometimes fast enough)
5     vector<int> sa(n),r(n);
6     for(i,0,n)r[i]=s[i];
7     for(int m=1;m<n;m*=2){
8         fore(i,0,n)sa[i]=i,sf[i]={r[i],i+m<n?r[i+m]:-1};
9         stable_sort(sa.begin(),sa.end(),sacomp);
10        r[sa[0]]=0;
11        fore(i,1,n)r[sa[i]]=sf[sa[i]]!=sf[sa[i-1]]?i:r[sa[i-1]];
12    }
13    return sa;

```

```
14 }
```

## 5.8 Suffix array

```
1 #define RB(x) (x<n?r[x]:0)
2 void csort(vector<int>& sa, vector<int>& r, int k){
3     int n=sa.size();
4     vector<int> f(max(255,n),0),t(n);
5     fore(i,0,n)f[RB(i+k)]++;
6     int sum=0;
7     fore(i,0,max(255,n))f[i]=(sum+=f[i])-f[i];
8     fore(i,0,n)t[f[RB(sa[i]+k)]]+=sa[i];
9     sa=t;
10 }
11 vector<int> constructSA(string& s){ // O(n logn)
12     int n=s.size(),rank;
13     vector<int> sa(n),r(n),t(n);
14     fore(i,0,n)sa[i]=i,r[i]=s[i];
15     for(int k=1;k<n;k*=2){
16         csort(sa,r,k);csort(sa,r,0);
17         t[sa[0]]=rank=0;
18         fore(i,1,n){
19             if(r[sa[i]]!=r[sa[i-1]]||RB(sa[i]+k)!=RB(sa[i-1]+k))rank++;
20             t[sa[i]]=rank;
21         }
22         r=t;
23         if(r[sa[n-1]]==n-1)break;
24     }
25     return sa;
26 }
```

## 5.9 LCP (Longest Common Prefix)

```
1 vector<int> computeLCP(string& s, vector<int>& sa){
2     int n=s.size(),L=0;
3     vector<int> lcp(n),plcp(n),phi(n);
4     phi[sa[0]]=-1;
5     fore(i,1,n)phi[sa[i]]=sa[i-1];
6     fore(i,0,n){
7         if(phi[i]<0){plcp[i]=0;continue;}
8         while(s[i+L]==s[phi[i]+L])L++;
9         plcp[i]=L;
10        L=max(L-1,0);
11    }
```

```
12 fore(i,0,n)lcp[i]=plcp[sa[i]];
13 return lcp; // lcp[i]=LCP(sa[i-1],sa[i])
14 }
```

## 5.10 Suffix Tree (Ukkonen's algorithm)

```
1 struct SuffixTree {
2     char s[MAXN];
3     map<int,int> to[MAXN];
4     int len[MAXN]={INF},fpos[MAXN],link[MAXN];
5     int node,pos,sz=1,n=0;
6     int make_node(int p, int l){
7         fpos[sz]=p;len[sz]=l;return sz++;}
8     void go_edge(){
9         while(pos>len[to[node][s[n-pos]]]){
10             node=to[node][s[n-pos]];
11             pos-=len[node];
12         }
13     }
14     void add(int c){
15         s[n++]=c;pos++;
16         int last=0;
17         while(pos>0){
18             go_edge();
19             int edge=s[n-pos];
20             int& v=to[node][edge];
21             int t=s[fpos[v]+pos-1];
22             if(v==0){
23                 v=make_node(n-pos,INF);
24                 link[last]=node;last=0;
25             }
26             else if(t==c){link[last]=node;return;}
27             else {
28                 int u=make_node(fpos[v],pos-1);
29                 to[u][c]=make_node(n-1,INF);
30                 to[u][t]=v;
31                 fpos[v]+=pos-1;len[v]-=pos-1;
32                 v=u;link[last]=u;last=u;
33             }
34             if(node==0)pos--;
35             else node=link[node];
36         }
37     }
```



38 };

## 5.11 Hashing

```

1 struct Hash {
2     int P=1777771,MOD[2],PI[2];
3     vector<int> h[2],pi[2];
4     Hash(string& s){
5         MOD[0]=999727999;MOD[1]=1070777777;
6         PI[0]=325255434;PI[1]=10018302;
7         fore(k,0,2)h[k].resize(s.size()+1),pi[k].resize(s.size()+1);
8         fore(k,0,2){
9             h[k][0]=0;pi[k][0]=1;
10            ll p=1;
11            fore(i,1,s.size()+1){
12                h[k][i]=(h[k][i-1]+p*s[i-1])%MOD[k];
13                pi[k][i]=(1LL*pi[k][i-1]*PI[k])%MOD[k];
14                p=(p*P)%MOD[k];
15            }
16        }
17    }
18    ll get(int s, int e){
19        ll h0=(h[0][e]-h[0][s]+MOD[0])%MOD[0];
20        h0=(1LL*h0*pi[0][s])%MOD[0];
21        ll h1=(h[1][e]-h[1][s]+MOD[1])%MOD[1];
22        h1=(1LL*h1*pi[1][s])%MOD[1];
23        return (h0<<32)|h1;
24    }
25 };

```

## 5.12 Hashing with ll (using \_\_int128)

```

1 #define bint __int128
2 struct Hash {
3     bint MOD=212345678987654321LL,P=1777771,PI=106955741089659571LL;
4     vector<bint> h,pi;
5     Hash(string& s){
6         assert((P*PI)%MOD==1);
7         h.resize(s.size()+1);pi.resize(s.size()+1);
8         h[0]=0;pi[0]=1;
9         bint p=1;
10        fore(i,1,s.size()+1){
11            h[i]=(h[i-1]+p*s[i-1])%MOD;
12            pi[i]=(pi[i-1]*PI)%MOD;

```

```

13        p=(p*P)%MOD;
14    }
15 }
16 ll get(int s, int e){
17     return (((h[e]-h[s]+MOD)%MOD)*pi[s])%MOD;
18 }
19 };

```

## 6 Flow

### 6.1 Matching (slower)

```

1 vector<int> g[MAXN]; // [0,n)->[0,m]
2 int n,m;
3 int mat[MAXM];bool vis[MAXN];
4 int match(int x){
5     if(vis[x])return 0;
6     vis[x]=true;
7     for(int y:g[x])if(mat[y]<0||match(mat[y])){mat[y]=x;return 1;}
8     return 0;
9 }
10 vector<pair<int,int> > max_matching(){
11     vector<pair<int,int> > r;
12     memset(mat,-1,sizeof(mat));
13     fore(i,0,n)memset(vis,false,sizeof(vis)),match(i);
14     fore(i,0,m)if(mat[i]>=0)r.pb({mat[i],i});
15     return r;
16 }

```

### 6.2 Matching (Hopcroft-Karp)

```

1 vector<int> g[MAXN]; // [0,n)->[0,m]
2 int n,m;
3 int mt[MAXN],mt2[MAXN],ds[MAXN];
4 bool bfs(){
5     queue<int> q;
6     memset(ds,-1,sizeof(ds));
7     fore(i,0,n)if(mt2[i]<0)ds[i]=0,q.push(i);
8     bool r=false;
9     while(!q.empty()){
10        int x=q.front();q.pop();
11        for(int y:g[x]){
12            if(mt[y]>=0&&ds[mt[y]]<0)ds[mt[y]]=ds[x]+1,q.push(mt[y]);

```

```

13     else if(mt[y]<0)r=true;
14     }
15     }
16     return r;
17 }
18 bool dfs(int x){
19     for(int y:g[x])if(mt[y]<0||ds[mt[y]]==ds[x]+1&&dfs(mt[y])){
20         mt[y]=x;mt2[x]=y;
21         return true;
22     }
23     ds[x]=1<<30;
24     return false;
25 }
26 int mm(){
27     int r=0;
28     memset(mt,-1,sizeof(mt));memset(mt2,-1,sizeof(mt2));
29     while(bfs()){
30         fore(i,0,n)if(mt2[i]<0)r+=dfs(i);
31     }
32     return r;
33 }

```

### 6.3 Hungarian

```

1 typedef long double td; typedef vector<int> vi; typedef vector<td> vd;
2 const td INF=1e100;//for maximum set INF to 0, and negate costs
3 bool zero(td x){return fabs(x)<1e-9;}//change to x==0, for ints/ll
4 struct Hungarian{
5     int n; vector<vd> cs; vi L, R;
6     Hungarian(int N, int M):n(max(N,M)),cs(n,vd(n)),L(n),R(n){
7         fore(x,0,N)fore(y,0,M)cs[x][y]=INF;
8     }
9     void set(int x,int y,td c){cs[x][y]=c;}
10 td assign() {
11     int mat = 0; vd ds(n), u(n), v(n); vi dad(n), sn(n);
12     fore(i,0,n)u[i]=*min_element(ALL(cs[i]));
13     fore(j,0,n){v[j]=cs[0][j]-u[0];fore(i,1,n)v[j]=min(v[j],cs[i][j]-u[i]);}
14     L=R=vi(n, -1);
15     fore(i,0,n)fore(j,0,n)
16         if(R[j]==-1&&zero(cs[i][j]-u[i]-v[j])){L[i]=j;R[j]=i;mat++;break;}
17     for(;mat<n;mat++){
18         int s=0, j=0, i;

```

```

19     while(L[s] != -1)s++;
20     fill(ALL(dad),-1);fill(ALL(sn),0);
21     fore(k,0,n)ds[k]=cs[s][k]-u[s]-v[k];
22     for(;;){
23         j = -1;
24         fore(k,0,n)if(!sn[k]&&(j==-1||ds[k]<ds[j]))j=k;
25         sn[j] = 1; i = R[j];
26         if(i == -1) break;
27         fore(k,0,n)if(!sn[k]){
28             auto new_ds=ds[j]+cs[i][k]-u[i]-v[k];
29             if(ds[k] > new_ds){ds[k]=new_ds;dad[k]=j;}
30         }
31     }
32     fore(k,0,n)if(k!=j&&sn[k]){auto w=ds[k]-ds[j];v[k]+=w,u[R[k]]-=w;
33         ;}
34     u[s] += ds[j];
35     while(dad[j]>=0){int d = dad[j];R[j]=R[d];L[R[j]]=j;j=d;}
36     R[j]=s;L[s]=j;
37     td value=0;fore(i,0,n)value+=cs[i][L[i]];
38     return value;
39 }
40 };

```

### 6.4 Dinic

```

1 // Min cut: nodes with dist>=0 vs nodes with dist<0
2 // Matching MVC: left nodes with dist<0 + right nodes with dist>0
3 struct Dinic{
4     int nodes,src,dst;
5     vector<int> dist,q,work;
6     struct edge {int to,rev;ll f,cap;};
7     vector<vector<edge>> g;
8     Dinic(int x):nodes(x),g(x),dist(x),q(x),work(x){}
9     void add_edge(int s, int t, ll cap){
10         g[s].pb((edge){t,SZ(g[t]),0,cap});
11         g[t].pb((edge){s,SZ(g[s])-1,0,0});
12     }
13     bool dinic_bfs(){
14         fill(ALL(dist),-1);dist[src]=0;
15         int qt=0;q[qt++]=src;
16         for(int qh=0;qh<qt;qh++){
17             int u=q[qh];

```

```

18     fore(i,0,SZ(g[u])){
19         edge &e=g[u][i];int v=g[u][i].to;
20         if(dist[v]<0&&e.f<e.cap)dist[v]=dist[u]+1,q[qt++]=v;
21     }
22 }
23 return dist[dst]>=0;
24 }
25 ll dinic_dfs(int u, ll f){
26     if(u==dst)return f;
27     for(int &i=work[u];i<SZ(g[u]);i++){
28         edge &e=g[u][i];
29         if(e.cap<=e.f)continue;
30         int v=e.to;
31         if(dist[v]==dist[u]+1){
32             ll df=dinic_dfs(v,min(f,e.cap-e.f));
33             if(df>0){e.f+=df;g[v][e.rev].f-=df;return df;}
34         }
35     }
36     return 0;
37 }
38 ll max_flow(int _src, int _dst){
39     src=_src;dst=_dst;
40     ll result=0;
41     while(dinic_bfs()){
42         fill(ALL(work),0);
43         while(ll delta=dinic_dfs(src,INF))result+=delta;
44     }
45     return result;
46 }
47 };

```

## 6.5 Min cost max flow

```

1 typedef ll tf;
2 typedef ll tc;
3 const tf INFFLOW=1e9;
4 const tc INFCOST=1e9;
5 struct MCF{
6     int n;
7     vector<tc> prio, pot; vector<tf> curflow; vector<int> prevedge,
8         prevnode;
9     priority_queue<pair<tc, int>, vector<pair<tc, int>>, greater<pair<tc,
10         int>>> q;

```

```

9     struct edge{int to, rev; tf f, cap; tc cost;};
10    vector<vector<edge>> g;
11    MCF(int n):n(n),prio(n),curflow(n),prevedge(n),prevnode(n),pot(n),g(n)
12        {}
13    void add_edge(int s, int t, tf cap, tc cost) {
14        g[s].pb((edge){t,SZ(g[t]),0,cap,cost});
15        g[t].pb((edge){s,SZ(g[s])-1,0,0,-cost});
16    }
17    pair<tf,tc> get_flow(int s, int t) {
18        tf flow=0; tc flowcost=0;
19        while(1){
20            q.push({0, s});
21            fill(ALL(prio),INFCOST);
22            prio[s]=0; curflow[s]=INFFLOW;
23            while(!q.empty()) {
24                auto cur=q.top();
25                tc d=cur.fst;
26                int u=cur.snd;
27                q.pop();
28                if(d!=prio[u]) continue;
29                for(int i=0; i<SZ(g[u]); ++i) {
30                    edge &e=g[u][i];
31                    int v=e.to;
32                    if(e.cap<=e.f) continue;
33                    tc nprio=prio[u]+e.cost+pot[u]-pot[v];
34                    if(prio[v]>nprio) {
35                        prio[v]=nprio;
36                        q.push({nprio, v});
37                        prevnode[v]=u; prevedge[v]=i;
38                        curflow[v]=min(curflow[u], e.cap-e.f);
39                    }
40                }
41            }
42            if(prio[t]==INFCOST) break;
43            fore(i,0,n) pot[i]+=prio[i];
44            tf df=min(curflow[t], INFFLOW-flow);
45            flow+=df;
46            for(int v=t; v!=s; v=prevnode[v]) {
47                edge &e=g[prevnode[v]][prevedge[v]];
48                e.f+=df; g[v][e.rev].f-=df;
49                flowcost+=df*e.cost;
50            }

```

```

51     return {flow,flowcost};
52 }
53 };

```

## 7 Other

### 7.1 Mo's algorithm

```

1  int n,sq,nq; // array size, sqrt(array size), #queries
2  struct qu{int l,r,id;};
3  qu qs[MAXN];
4  ll ans[MAXN]; // ans[i] = answer to ith query
5  bool qcomp(const qu &a, const qu &b){
6      if(a.l/sq!=b.l/sq) return a.l<b.l;
7      return (a.l/sq)&1?a.r<b.r:a.r>b.r;
8  }
9  void mos(){
10     fore(i,0,nq)qs[i].id=i;
11     sq=sqrt(n)+.5;
12     sort(qs,qs+nq,qcomp);
13     int l=0,r=0;
14     init();
15     fore(i,0,nq){
16         qu q=qs[i];
17         while(l>q.l)add(--l);
18         while(r<q.r)add(r++);
19         while(l<q.l)remove(l++);
20         while(r>q.r)remove(--r);
21         ans[q.id]=get_ans();
22     }
23 }

```

### 7.2 Divide and conquer DP optimization

```

1  // O(knlogn). For 2D dps, when the position of optimal choice is non-
   decreasing as the second variable increases
2  int k,n,f[MAXN],f2[MAXN];
3  void doit(int s, int e, int s0, int e0, int i){
4      // [s,e): range of calculation, [s0,e0): range of optimal choice
5      if(s==e)return;
6      int m=(s+e)/2,r=INF,rp;
7      fore(j,s0,min(e0,m)){
8          int r0=something(i,j); // "something" usually depends on f

```

```

9      if(r0<r)r=r0,rp=j; // position of optimal choice
10     }
11     f2[m]=r;
12     doit(s,m,s0,rp+1,i);doit(m+1,e,rp,e0,i);
13 }
14 int doall(){
15     init_base_cases();
16     fore(i,1,k)doit(1,n+1,0,n,i),memcpy(f,f2,sizeof(f));
17     return f[n];
18 }

```

### 7.3 Dates

```

1  int dateToInt(int y, int m, int d){
2      return 1461*(y+4800+(m-14)/12)/4+367*(m-2-(m-14)/12*12)/12-
3          3*((y+4900+(m-14)/12)/100)/4+d-32075;
4  }
5  void intToDate(int jd, int& y, int& m, int& d){
6      int x,n,i,j;x=jd+68569;
7      n=4*x/146097;x-=(146097*n+3)/4;
8      i=(4000*(x+1))/1461001;x-=1461*i/4-31;
9      j=80*x/2447;d=x-2447*j/80;
10     x=j/11;m=j+2-12*x;y=100*(n-49)+i+x;
11 }
12 int DayOfWeek(int d, int m, int y){ //starting on Sunday
13     static int ttt[]={0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4};
14     y-=m<3;
15     return (y+y/4-y/100+y/400+ttt[m-1]+d)%7;
16 }

```

### 7.4 C++ stuff

```

1  // double inf
2  const double DINF=numeric_limits<double>::infinity();
3  // Custom comparator for set/map
4  struct comp {
5      bool operator()(const double& a, const double& b) const {
6          return a+EPS<b;}
7  };
8  set<double,comp> w; // or map<double,int,comp>
9  // Iterate over non empty subsets of bitmask
10 for(int s=m;s=(s-1)&m) // Decreasing order
11 for (int s=0;s=s-m&m;) // Increasing order
12 // Return the numbers the numbers of 1-bit in x

```

```

13 int __builtin_popcount (unsigned int x)
14 // Returns the number of trailing 0-bits in x. x=0 is undefined.
15 int __builtin_ctz (unsigned int x)
16 // Returns the number of leading 0-bits in x. x=0 is undefined.
17 int __builtin_clz (unsigned int x)
18 // x of type long long just add 'll' at the end of the function.
19 int __builtin_popcountll (unsigned long long x)
20 // Get the value of the least significant bit that is one.
21 v=(x&(-x))
    
```

## 7.5 Interactive problem tester template

```

1 #Easier method with bash commands:
2 #mkfifo fifo
3 #(. /solution < fifo) | (. /interactor > fifo)
4
5 # tester for cf 101021A (guess a number, queries: is it >=k?)
6 import random
7 import subprocess as sp
8 seed = random.randint(0, sys.maxint); random.seed(seed)
9 n=random.randint(1,1000000)
10 try:
11     p=sp.Popen(['./a.out'],stdin=sp.PIPE,stdout=sp.PIPE)
12     it=0
13     s=p.stdout.readline()
14     while it<25 and s and s[0]!='!':
15         k=int(s)
16         assert k>=1 and k<=1000000
17         if n>=k: p.stdin.write('>=\n')
18         else: p.stdin.write('<\n')
19         s=p.stdout.readline()
20         it+=1
21     assert s and s[0]=='!'
22     k=int(s.split()[1])
23     assert k==n
24 except:
25     print 'failed with seed %s' % seed
26     raise
    
```

## 7.6 Max number of divisors up to $10^n$

```

1 (0,1) (1,4) (2,12) (3,32) (4,64) (5,128) (6,240) (7,448) (8,768)
   (9,1344) (10,2304) (11,4032) (12,6720) (13,10752) (14,17280)
   (15,26880) (16,41472) (17,64512) (18,103680)
    
```

## 7.7 Template

```

1 #include <bits/stdc++.h>
2 #ifdef DEMETRIO
3 #define deb(...) fprintf(stderr,__VA_ARGS__)
4 #define deb1(x) cerr << #x << " = " << x << endl
5 #else
6 #define deb(...) 0
7 #define deb1(x) 0
8 #endif
9 #define pb push_back
10 #define mp make_pair
11 #define fst first
12 #define snd second
13 #define fore(i,a,b) for(int i=a,ThxDem=b;i<ThxDem;++i)
14 #define SZ(x) ((int)x.size())
15 using namespace std;
16 typedef long long ll;
17
18 int main(){
19     return 0;
20 }
    
```