

```

1  /// Bismillahi-r-Rahmani-r-Rahim
2  #include<bits/stdc++.h>
3  using namespace std;
4  #define debug(args...){ string _s = #args;replace(_s.begin(),_s.end(),',',' ');stringstream
5  _ss(_s);istream_iterator<string>_it(_ss);err(_it, args);} cout<<endl;
6  void err(istream_iterator<string> it) {}
7  template<typename T, typename... Args>
8  void err(istream_iterator<string> it, T a, Args... args) {cerr << *it << "=" << a <<
9  ", "; err(++it,
10 args...);}
11 #define ll long long int
12 #define MAX 2134567891
13 #define PF(a) cout<<a<<endl;
14 #define pf(a) printf("%lld", a);
15 #define sf(a) scanf("%lld", &a);
16 #define fr(i,n) for(i=0;i<n;i++)
17 #define rep(i,n) for(i=1;i<=n;i++)
18 #define rev(i,a,n) for(i=n;i>=a;i--)
19 #define FOR(i,a,n) for(i=a;i<=n;i++)
20 #define ALL(n) n.begin(),n.end()
21 #define mem(x,n) memset(x,n,sizeof(x));
22 //int fx[]={+1,-1,+0,+0};
23 //int fy[]={+0,+0,+1,-1};
24 //int fx[]={+0,+0,+1,-1,-1,+1,-1,+1}; // Kings Move
25 //int fy[]={-1,+1,+0,+0,+1,+1,-1,-1}; // Kings Move
26 //int fx[]={-2,-2,-1,-1,1,1,2,2}; // Knights Move
27 //int fy[]={-1,1,-2,2,-2,2,-1,1}; // Knights Move
28 #define TC(t) printf("Case %lld: ",t);
29 #define ans(t,c) printf("Case %lld: %lld\n",t,c);
30 #define SETP(n) cout<<setprecision(n)<<fixed;
31 #define READ freopen("F:\\Project\\Test_Case.txt","r",stdin)
32 #define WRITE freopen("F:\\Project\\Output_Test.txt","w",stdout)
33 #define IO ios_base::sync_with_stdio(0); cin.tie(0);cout.tie(0);
34 #define PAIR pair<ll,ll>
35 #define MP make_pair
36 #define pb push_back
37 #define eb emplace_back
38 #define ff first
39 #define ss second
40 #define NL printf("\n");
41 #define bug(a) cout<<#a<<" "<<a<<" ";
42 #define hlw printf("hlw\n");
43 #define hii printf("hii\n");
44 #define NN 111
45 #define MOD (ll)1e9+7 /// 10^9+7
46 #define N (ll)1e6+7 ///10^6->6 zero after 1 **
47 ll x[N],y[N],z[N],n;
48 string s,S;
49 vector<ll>v;
50 //bitset<N>B;
51 //map <ll,ll> mp;
52 /// priority_queue<ll, vector<ll>, greater<ll> > pq;
53 int main()
54 {
55     //IO;
56     //while(1)
57     //READ;WRITE;
58     {
59         ll a=0,b=0,c=0,d,e,f,g,i,j,k,l,m,p,q,r,u,w,t,tc=1;

```

```

59 ll in,loc,val,sz,lo,hi,mid,mn=MAX,mx=0,sum=0,ans=0;
60 //cin>>tc;
61 rep(t,tc)
62 {
63
64 }
65 }
66 return 0;
67 }
68 /// Division MOD needs BigMod(a,n-2)
69 //////////////////////////////////////
STRING////////////////////////////////////
70 unsigned bernstein_hash ( void *key, int len )
71 {
72 unsigned char *p = key;
73 unsigned h = 0;
74 int i;
75 for ( i = 0; i < len; i++ )
76 h = 33 * h + p[i];
77 return h;
78 }
79 /// string matching
80 vector<int> rabin_karp_HASH(string const& s, string const& t) {
81 const int p = 31;
82 const int m = 1e9 + 9;
83 int S = s.size(), T = t.size();
84 vector<long long> p_pow(max(S, T));
85 vector<long long> h(T + 1, 0);
86 long long h_s = 0;
87 vector<int> occurrences;
88 p_pow[0] = 1;
89 for (int i = 1; i < (int)p_pow.size(); i++)
90 p_pow[i] = (p_pow[i-1] * p) % m;
91 for (int i = 0; i < T; i++)
92 h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
93 for (int i = 0; i < S; i++)
94 h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;
95 for (int i = 0; i + S - 1 < T; i++)
96 {
97 long long cur_h = (h[i+S] + m - h[i]) % m;
98 if (cur_h == h_s * p_pow[i] % m)
99 occurrences.push_back(i);
100 }
101 return occurrences;
102 }
103 /// KMP with LPS (find pattern)
104 void LPS()
105 {
106 ll i,j,l=pat.size();
107 i=0,j=-1;
108 lps[i]=j;
109 while(i<l)
110 {
111 while(pat[i]!=pat[j] && j>=0)
112 j=lps[j];
113 i++,j++;
114 lps[i]=j;
115 }
116 }
117 ll KMP(string txt)

```

```

118 {
119     pat=txt; reverse(ALL(pat));
120     LPS(pat);
121     ll i,j,n,m;
122     n=txt.size();
123     m=pat.size();
124     i=j=0;
125     while(i<n)
126     {
127         while(j>=0 && txt[i]!=pat[j])
128             j=lps[j];
129         i++,j++;
130     }
131     return j;
132 }
133 string sub_pal(string s) /// Find Prefix Sub_Palindrome Linear
134 {
135     string a = s;
136     reverse(a.begin(), a.end());
137     a = s + "#" + a;
138     //cout<<a<<endl;
139     ll c = 0, pref[99]={0};
140     for (int i = 1; i < (int)a.size(); i++)
141     {
142         cout<<"C " <<c<<endl;
143         while (c != 0 && a[c] != a[i])
144             c = pref[c - 1];
145         if (a[c] == a[i])
146             c++;
147         pref[i] = c;
148     }
149     return s.substr(0, c);
150 }
151 string Manacher(string s) /// longest subpalindrome
152 {
153     string T="#";/// Transform S to T
154     for(int i=0;i<s.size();i++)
155         T+=s.substr(i,1)+"#";
156     int P[T.size()+5]={0}; // Array to record longest palindrome
157     int center=0,boundary=0,maxLen=0,resCenter=0;
158     for(int i=1;i<T.size()-1;i++)
159     {
160         int iMirror=2*center-i; // calc mirror i = center-(i-center)
161         if(i<boundary)
162             P[i]=min(boundary-i,P[iMirror]);
163         while(i-1-P[i]>=0 && i+1+P[i]<=T.size()-1 && T[i+1+P[i]]==T[i-1-P[i]]) // Attempt to
164             expand palindrome centered at i
165             P[i]++;
166         if(i+P[i]>boundary)
167         { // update center and boundary
168             center = i;
169             boundary = i+P[i];
170         }
171         if(P[i]>maxLen)
172         { // update result
173             maxLen = P[i];
174             resCenter = i;
175         }
176     }
177     return s.substr((resCenter - maxLen)/2, maxLen);

```

```

178 }
179 vector<int>z_algo(string s) /// finds all occurrences of a pattern linear
180 {
181     int i,l,r,n;
182     n=s.length();
183     vector<int> z(n);
184     for (i = 1, l = 0, r = 0; i < n; ++i)
185     {
186         if (i <= r)
187             z[i] = min (r - i + 1, z[i - 1]);
188         while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ///Checking character and ++1
189             ++z[i];
190         if (i + z[i] - 1 > r)
191             l = i, r = i + z[i] - 1;
192     }
193     return z;
194 }
195 //////////////////////////////////////////////////Sparse Table
196 ll st[22][N],x[N],logs[N];
197 void build(ll n)/// 0
198 {
199     ll i,j,k;
200     logs[1]=0; for(i=2;i<=n;i++)logs[i]= logs[i/2]+1;
201     for(i=0;i<n;i++)st[0][i]=x[i];
202     for(i=1; (1<<i) <n; i++)
203     {
204         for(j=0; j+(1<<i)<=n; j++) /// 1<<i = current_len
205         {
206             st[i][j]=min(st[i-1][j], st[i-1][j + (1<<i-1)]);
207         }
208     }
209 }
210 ll query(ll l, ll r)
211 {
212     ll pow = logs[r-l+1]; ///log2(r-l+1);
213     return min(st[pow][l], st[pow][r-(1<<pow)+1]);
214 }
215 build(n); cout<<query(l,r)<<endl;
216 //////////////////////////////////////////////////Segment
Tree////////////////////////////////////
217 ll tree[4*N],tr[N],lazy[4*N];
218 void build(ll in,ll L,ll R)
219 {
220     if(L==R)
221     {
222         tree[in]=tr[L];
223         return;
224     }
225     ll mid=(L+R)/2;
226     build(in*2,L,mid);
227     build(in*2+1,mid+1,R);
228     tree[in]=min(tree[in*2],tree[in*2+1]); /// Change Function
229 }
230 void lazy_update (ll in,ll L,ll R,ll x,ll y,ll val)
231 {
232     if(x>y)return;
233     if(lazy[in]!=0)
234     {
235         tree[in]+=lazy[in];
236         if(L!=R)

```

```

237 {
238     lazy[in*2]+=lazy[in];
239     lazy[in*2+1]+=lazy[in];
240 }
241 lazy[in]=0;
242 }
243 if(x>R || y<L)return;
244 if(x<=L && y>=R)
245 {
246     tree[in]+=val;
247     if(L!=R)
248     {
249         lazy[in*2]+=val;
250         lazy[in*2+1]+=val;
251     }
252     return;
253 }
254 ll mid=(L+R)/2;
255 lazy_update(in*2,L,mid,x,y,val);
256 lazy_update(in*2+1,mid+1,R,x,y,val);
257 tree[in]=tree[in*2]+tree[in*2+1];
258 }
259 ll lazy_query(ll in,ll L,ll R,ll x,ll y)
260 {
261     if(x>y)return 0;
262     if(lazy[in]!=0)
263     {
264         tree[in]+=lazy[in];
265         if(L!=R)
266         {
267             lazy[in*2]+=lazy[in];
268             lazy[in*2+1]+=lazy[in];
269         }
270         lazy[in]=0;
271     }
272     if(x>R || y<L)return 0;
273     if(x<=L && y>=R)
274         return tree[in];
275     ll p,q,mid=(L+R)/2;
276     p=lazy_query(in*2,L,mid,x,y);
277     q=lazy_query(in*2+1,mid+1,R,x,y);
278     return p+q;
279 }
280 void update(ll in,ll L,ll R,ll pos,ll val)
281 {
282     if(pos>R||L>pos)return;
283     if(L==R&&pos==L)
284     {
285         tree[in]+=val; /// Change Function
286         return;
287     }
288     ll mid=(L+R)/2;
289     update(in*2,L,mid,pos,val);
290     update(in*2+1,mid+1,R,pos,val);
291     tree[in]=tree[in*2]+tree[in*2+1]; /// Change Function
292 }
293 ll query(ll L,ll R,ll in,ll i,ll j)
294 {
295     if(j<L||i>R)return MAX;
296     //return 0;

```

```

297 if(L>=i&&j>=R)return tree[in];
298 ll p,q,mid=(L+R)/2;
299 p=query(L,mid,in*2,i,j);
300 q=query(mid+1,R,in*2+1,i,j);
301 return min(p,q); /// Change Function
302 }
303 build(1,1,n); cout<<query(1,n,1,a,b)<<endl;
304 lazy_update(1,1,n,a,b,c); cout<<lazy_query(1,n,1,a,b)<<endl;
305 ////////////////////////////////////DP////////////////////////////////////
306 ll LCS(char p[],char q[],int a,int b)
307 {
308     ///All loop will work through 1 to n/m here...
309     int i,j,k;
310     rep(i,a)
311     x[i][0]=0;
312     rep(i,b)
313     x[0][i]=0;
314     rep(i,a)
315     rep(j,b)
316     {
317         if(p[i]==q[j])x[i][j]=x[i-1][j-1]+1;
318         else x[i][j]=max(x[i][j-1],x[i-1][j]);
319     }
320     return x[a][b];
321 }
322 ll LIS(ll n)
323 {
324     ll i,a,in=0,st,en,mid,ans=-1;
325     ar[1]=INT_MIN;
326     rep(i,n)
327     {
328         a=x[i];
329         if(in==0 || a>ar[in])
330         {
331             cout<<"Appending "<<a<<" in "<<1+in<<endl;
332             ar[++in]=a;
333         }
334         else if(a<x[1])
335             ar[1]=a;
336         else
337         {
338             st=1,en=in;
339             while(st<=en)
340             {
341                 mid=(st+en)/2;
342                 if(ar[mid]<a)
343                     st=mid+1;
344                 else en=mid-1;
345             }
346             ar[st]=a;
347             cout<<mid<<" mid\n";
348         }
349         cout<<"i "<<i<<" a "<<a<<" in "<<in<<endl;
350     }
351     return in;
352 }
353 ////////////////////////////////////BIT////////////////////////////////////
354 void update(ll pos,ll val)
355 {
356     while(pos<=n)

```

```

357 {
358     x[pos]+=val;
359     pos+=(pos & -pos);
360 }
361 }
362 ll query(ll pos)
363 {
364     ll sum=0;
365     while(pos)
366     {
367         sum+=x[pos];
368         pos-=(pos & -pos);
369     }
370     return sum;
371 }
372 rep(i,n)
373 {cin>>a; update(i,a); /// 1-based}
374 cout<<query(4)<<" "<<query(2)<<" Ans "<<query(4)-query(2)<<endl;
375 //////////////////////////////////MATH////////////////////////////////////
376 ll spf[N]; vector<ll>primes;
377 void sieve() ///with SPF
378 {
379     for(int i = 2; i < N; i++)
380     {
381         if (spf[i] == 0) spf[i] = i, primes.push_back(i);
382         int sz = primes.size();
383         for (int j=0; j<sz && i*primes[j]<N && primes[j]<=spf[i]; j++)
384         {
385             spf[i * primes[j]] = primes[j];
386         }
387     }
388 }
389 ll nCr(ll n,ll r) /// nCr DP
390 {
391     ll &ret=dp[n][r];
392     if(~ret)return ret;
393     if(n==r)return ret=1;
394     if(r==1)return ret=n;
395     return ret=nCr(n-1,r)+nCr(n-1,r-1);
396 }
397 ll bigmod(ll n,ll p,ll MOD) /// finds n ^ p % MOD
398 {
399     if(p==0)return 1;
400     ll x=bigmod(n,p/2,MOD);
401     x=(x*x)%MOD;
402     if(p%2)x=(x*n)%MOD;
403     return x;
404 }
405 ll precal_nCr(ll n, ll r) /// larger inputs and MOD required
406 {
407     /// Precal Starts Here
408     fact[1] = 1;
409     for(ll i=2; i<n; i++) fact[i] = (i*fact[i-1])%MOD;
410     invfact[n-1] = bigmod(fact[n-1], MOD-2, MOD);
411     for (ll i=n-2; i>=0; i--) invfact[i] = (invfact[i+1]*(i+1))%MOD;
412     /// Precal Ends Here
413     if (r<0 || r>n) return 0;
414     return (fact[n]*(invfact[r]*invfact[n-r])%MOD)%MOD;
415 }
416 void permutation(string s,int i,int n)

```

```

417 {
418     if(i==n){cout<<s<<endl;return ;}
419     for(int j=i;j<=n;j++)
420     {
421         swap(s[i],s[j]);
422         permutation(s,i+1,n);
423     }
424 }
425 ll mod_inverse(ll a,ll mod)
426 {
427     return bigmod(a,mod-2,mod);
428 }
429 void allPossibleSubset(int n)
430 {
431     for(ll mask = 0; mask < (1 << n); mask++) {
432         ll sum_of_this_subset = 0;
433         for(int i = 0; i < n; i++)
434         {
435             if(mask & (1 << i)) {
436                 sum_of_this_subset += x[i];
437             }
438         }
439     }
440 }
441 /// Find numbers of co-prime of N which are less than N
442 void totient()
443 {
444     ll i,j,k;
445     for(i=1;i<=N;i++)phi[i]=i;
446     for(i=2;i<=N;i++)
447     {
448         if(phi[i]==i)
449         {
450             for(j=i;j<=N;j+=i)
451             {
452                 phi[j]= (phi[j]*(i-1))/i;
453             }
454         }
455     }
456 }
457 /// Find eulerphi for any numbers with prime pre-calculated
458 int eulerPhi ( int n ) {
459     int res = n;
460     int sqrtn = sqrt ( n );
461     for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
462         if ( n % prime[i] == 0 ) {
463             while ( n % prime[i] == 0 ) {
464                 n /= prime[i];
465             }
466             sqrtn = sqrt ( n );
467             res /= prime[i];
468             res *= prime[i] - 1;
469         }
470     }
471     if ( n != 1 ) {
472         res /= n;
473         res *= n - 1;
474     }
475     return res;
476 }

```


[illegible]

```

537 ll i, in;
538 last[s]=k;
539 nodes[k]=s;
540 depth[k++]=d;
541 fr(i,v[s].size())
542 {
543   in=v[s][i];
544   if(vis[in])continue;
545   vis[in]=1;
546   walk(in,d+1);
547   nodes[k]=s;
548   depth[k++]=d;
549 }
550 }
551 void sparse_table(ll n)/// 0 based indexing
552 {
553   ll node_a,node_b,i,j,k;
554   for(i=0;i<n;i++)st[0][i]=i; /// storing nodes, not values
555   for(i=1; (1<<i) <n; i++)
556   {
557     for(j=0; j+(1<<i)<=n; j++) /// 1<<i = current_len
558     {
559       node_a=st[i-1][j];
560       node_b=st[i-1][j + (1<<i-1)];
561       st[i][j] = depth[node_a]<=depth[node_b]? node_a:node_b; /// For RMQ
562     }
563   }
564 }
565 ll LCA(ll l,ll r)
566 {l=last[l],r=last[r];if(l>r)swap(l,r);ll pow = log2(r-l+1);ll a,b;a=st[pow][l];
  b=st[pow][r-(1<<pow)+1];return nodes[depth[a]<=depth[b]? a:b];}
567 int main(){ ///0 based indexing
568   vis[0]=1;
569   walk(0,0);
570   sparse_table(2*n-1);
571   cin>>a>>b; cout<<LCA(a-1,b-1)+1<<endl;
572 }
573 /* author : s@if */
574 #include<bits/stdc++.h>
575 #include<ext/pb_ds/assoc_container.hpp>
576 using namespace __gnu_pbds;
577 using namespace std;
578 #define NIL -1
579 #define INF 1e9
580 #define EPS 1e-9
581 #define SAIF main
582 #define fi first
583 #define sec second
584 #define MAX INT_MAX
585 #define ll long long
586 #define PI acos(-1.0)
587 #define MOD 1000000007
588 #define PLL pair<ll,ll>
589 #define PII pair<int,int>
590 #define ull unsigned long long
591 #define For(i,a,b) for(int i=a;i<=(int)b;i++)
592 typedef tree<int, null_type, less<int>, rb_tree_tag,
593           tree_order_statistics_node_update> new_data_set;
594 /**find_by_order(k) gives the kth element;

```

```

595 //order_of_key(item) gives the index(number of element strictly less than item) of
    item;
596 inline int in() {int x; scanf("%d", &x); return x; }
597 bool Check(int N , int pos) { return (bool) (N & (1<<pos));}
598 int Set(int N, int pos) { return N = N | (1<<pos);}
599 int fx[]={+0,+0,+1,-1,-1,+1,-1,+1}; // King's move
600 int fy[]={-1,+1,+0,+0,+1,+1,-1,-1};
601 int hx[]={-2,-2,-1,+1,+2,+2,-1,+1}; // Knight's move
602 int hy[]={+1,-1,+2,+2,-1,+1,-2,-2};
603 int dx[]={+1,-1,+0,+0};
604 int dy[]={+0,+0,+1,-1};
605 const int MAXN = (int)2e5+9;
606 /*
607 // Hashing
608 ll base = 247, M = 1000000007;
609 ll Hash[MAXN], power[MAXN], L;
610 void init(void)
611 {
612     power[0] = 1; Hash[0] = 0;
613     for(int i=1; i<MAXN; i++)
614     {
615         power[i] = (power[i-1]*base)%M;
616     }
617 }
618 void Hashing(string s)
619 {
620     L = s.size();
621     ll h = 0;
622     for(int i=1; i<=L; i++)
623     {
624         ll tmp = (h*base)%M;
625         tmp = (tmp+s[i-1]-'a'+1)%M;
626         Hash[i] = h = tmp;
627     }
628     return;
629 }
630 ll HashOf(string p)
631 {
632     int l = p.size();
633     ll h = 0;
634     for(int i=1; i<=l; i++)
635     {
636         ll tmp = (h*base)%M;
637         tmp = (tmp+p[i-1]-'a'+1)%M;
638         h = tmp;
639     }
640     return h;
641 }
642 ll HashOfSubstring(int l, int r)
643 {
644     ll a, b, ret;
645     a = Hash[l-1], b = Hash[r];
646     a = (a*power[r-l+1])%M;
647     ret = (b-a+M)%M;
648
649     return ret;
650 }
651 int FindPattern(string p)
652 {
653     int i, l = p.size();

```

```

654     ll h1 = HashOf(p);
655
656     for(i=1; i<=L-l+1; i++)
657     {
658         int x = i, y = i+l-1;
659         ll h2 = HashOfSubstring(x, y);
660         if(h1==h2) return i-1;
661     }
662
663     return -1;
664 }
665 */
666 /*
667 // trie
668 struct node
669 {
670     bool mark;
671     node *next[30];
672     node()
673     {
674         mark=false;
675
676         for(int i=0;i<26;i++)
677         {
678             next[i]=NULL;
679         }
680     }
681 };
682 node *root;
683 void add(string s)
684 {
685     int l=s.size();
686
687     node *curr=root;
688     for(int i=0;i<l;i++)
689     {
690         int id=s[i]-'a';
691         if(curr->next[id]==NULL)
692             curr->next[id]=new node();
693
694         curr=curr->next[id];
695     }
696
697     curr->mark=true;
698 }
699 bool _search(string s)
700 {
701     int l=s.size();
702
703     node *curr=root;
704     for(int i=0;i<l;i++)
705     {
706         int id=s[i]-'a';
707         if(curr->next[id]==NULL)
708             curr->next[id]=new node();
709         curr=curr->next[id];
710     }
711     return curr->mark;
712 }
713 void del(node *curr)

```

```

714 {
715     for(int i=0;i<26;i++)
716     {
717         if(curr->next[i])
718             del(curr->next[i]);
719     }
720     delete(curr);
721 }
722 */
723 /*
724 // KMP
725 void kmp(string T, string P)
726 {
727     int n=strlen(T);
728     int m=strlen(P);
729     int pi[m+9], i, now;
730
731     now=pi[0]=-1;
732     for(i=1;i<m;i++)
733     {
734         while(now!=-1 && P[now+1]!=P[i])
735         {
736             now=pi[now];
737         }
738         if(P[now+1]==P[i])
739             pi[i]= ++now;
740         else
741             pi[i]=now=-1;
742     }
743     int cnt=0;
744     now=-1;
745     for(i=0;i<n;i++)
746     {
747         while(now!=-1 && P[now+1]!=T[i])
748         {
749             now=pi[now];
750         }
751         if(P[now+1]==T[i])
752             now++;
753         else
754             now=-1;
755
756         if(now==m-1)
757         {
758             cnt++;
759             now=pi[now];
760         }
761     }
762 }
763 printf("Case %d: %d\n",++t,cnt);
764
765     return;
766 }*/
767 /*
768 // Articulation Point
769 int vis[MAXN], d[MAXN], low[MAXN], art[MAXN], Tm;
770 vector<int>adj[MAXN];
771 void init(int n)
772 {
773     for(int i=0; i<=n; i++)

```

```

774     {
775         vis[i] = 0; art[i] = 0, Tm = 0;
776         adj[i].clear();
777     }
778 }
779 void find_articulation_point(int u)
780 {
781     Tm++; d[u] = low[u] = Tm;
782     vis[u] = 1; int child = 0;
783
784     for(int i=0; i<adj[u].size(); i++)
785     {
786         int v = adj[u][i];
787
788         if(vis[v]==1)
789         {
790             low[u] = min(low[u], d[v]);
791         }
792         else
793         {
794             child++;
795             find_articulation_point(v);
796             low[u] = min(low[u], low[v]);
797             if(d[u]<=low[v] && u!=1) art[u] = 1;
798         }
799     }
800     if(u==1 && child>1) art[u] = 1;
801 }
802 */
803 /*
804 // SCC
805 vector<int>component[MAXN];
806 vector<int>g[MAXN];
807 vector<int>rev[MAXN];
808 stack<int>stk;
809 int n,mark;
810 int vis[MAXN];
811 void dfs1(int cur)
812 {
813     vis[cur]=1;
814
815     for(int i=0;i<g[cur].size();i++)
816     {
817         int v=g[cur][i];
818
819         if(!vis[v])
820             dfs1(v);
821     }
822
823     stk.push(cur);
824 }
825 void dfs2(int cur,int mark)
826 {
827     vis[cur]=1;
828     component[mark].push_back(cur);
829     for(int i=0;i<rev[cur].size();i++)
830     {
831         int v=rev[cur][i];
832
833         if(!vis[v])

```

```

834     {
835         dfs2(v,mark);
836     }
837 }
838 }
839 void SCC(void)
840 {
841     cin>>n>>m;
842     while(m--)
843     {
844         cin>>u>>v;
845
846         g[u].push_back(v);
847         rev[v].push_back(u);
848     }
849     memset(vis,0,sizeof(vis));
850     for(i=1;i<=n;i++)
851         if(!vis[i])
852             dfs1(i);
853
854     memset(vis,0,sizeof(vis));
855     mark=0;
856     while(!stk.empty())
857     {
858         u=stk.top();
859         stk.pop();
860
861         if(!vis[u])
862         {
863             dfs2(u,++mark);
864         }
865     }
866     for(i=1;i<=mark;i++)
867     {
868         cout<<"component "<<i<<" : ";
869
870         for(j=0;j<component[i].size();j++)
871             cout<<component[i][j]<<" ";
872         cout<<endl;
873     }
874     cout<<endl;
875 }
876 */
877 /*
878 //LCA
879 int L[mx];
880 int P[mx][22];
881 int T[mx];
882 vector<int>g[mx];
883 void dfs(int from,int u,int dep)
884 {
885     T[u]=from;
886     L[u]=dep;
887     for(int i=0;i<(int)g[u].size();i++)
888     {
889         int v=g[u][i];
890         if(v==from) continue;
891         dfs(u,v,dep+1);
892     }
893 }

```

```

894 int lca_query(int N, int p, int q)
895 {
896     int tmp, log, i;
897
898     if (L[p] < L[q])
899         tmp = p, p = q, q = tmp;
900     log=1;
901     while(1) {
902         int next=log+1;
903         if((1<<next)>L[p])break;
904         log++;
905     }
906     for (i = log; i >= 0; i--)
907         if (L[p] - (1 << i) >= L[q])
908             p = P[p][i];
909
910     if (p == q)
911         return p;
912
913     for (i = log; i >= 0; i--)
914         if (P[p][i] != -1 && P[p][i] != P[q][i])
915             p = P[p][i], q = P[q][i];
916
917     return T[p];
918 }
919
920 void lca_init(int N)
921 {
922     memset (P,-1,sizeof(P));
923     int i, j;
924     for (i = 0; i < N; i++)
925         P[i][0] = T[i];
926
927     for (j = 1; 1 << j < N; j++)
928         for (i = 0; i < N; i++)
929             if (P[i][j - 1] != -1)
930                 P[i][j] = P[P[i][j - 1]][j - 1];
931 }
932 */
933 /*
934 // Discrete Logarithm
935
936 ll Discrete_Log(ll a, ll b, ll m)
937 {
938     if(a==0)
939     {
940         if(b==0) return 1;
941         else return -1;
942     }
943     a%=m, b%=m; ll g, k = 1, add = 0;
944     while((g=__gcd(a,m))>1)
945     {
946         if(b==k) return add;
947         if(b%g) return -1;
948
949         b/=g, m/=g; ++add;
950
951         k = (k*a/g)%m;
952     }
953     map<ll,ll>Map; ll n = sqrt(m)+1;

```



```

954
955     for(ll q=0, curr=b; q<=n; q++)
956     {
957         Map[curr] = q;
958         curr = (curr*a)%m;
959     }
960     ll an = 1;
961     for(ll p=1; p<=n; p++)
962         an = (an*a)%m;
963     for(ll p=1, curr=k; p<=n; p++)
964     {
965         curr = (curr*an)%m;
966
967         if(Map[curr])
968             return n*p-Map[curr]+add;
969     }
970     return -1;
971 }
972 */
973 void solve(void)
974 {
975     ll a, b, i,j,k,l,m,n,p,q,x,y,u,v,w,r,tc,t;
976     return;
977 }
978 int SAIF()
979 {
980     int tc, t = 0;
981     cin>>tc; while(tc--) solve();
982     return 0;
983 }
984 // read the question correctly (is y a vowel? what are the exact constraints?)
985 // look out for SPECIAL CASES (n=1?) and overflow (ll vs int?)
986

```