

Contents

1	Basic	1
1.1	default code	1
1.2	.vimrc	1
2	math	1
2.1	ext gcd	1
2.2	FFT	2
2.3	NTT	2
2.4	orFFT	2
2.5	andFFT	3
2.6	xorFFT	3
2.7	MillerRabin other	3
2.8	Guass	4
3	flow	4
3.1	dinic	4
3.2	min-cost-max-flow	4
4	string	5
4.1	KMP	5
4.2	Z-value	5
4.3	Z-value-palindrome	6
4.4	Suffix Array ($O(N \log N)$)	6
4.5	Aho-Corasick	7
4.6	Aho-Corasick-2016ioicamp	7
4.7	Palindrome Automaton	8
4.8	Suffix Automaton (bcw)	8
5	graph	9
5.1	Bipartite matching ($O(N^3)$)	9
5.2	KM ($O(N^4)$)	9
5.3	general graph matching (bcw)	10
5.4	minimum general graph weighted matching (bcw)	11
5.5	Max clique (bcw)	12
5.6	EdgeBCC	12
5.7	VertexBCC	13
5.8	Dominating Tree	13
5.9	Them.	14
6	data structure	14
6.1	Treap	14
6.2	copy on write treap	15
6.3	copy on write segment tree	16
6.4	Treap+ (HOJ 92)	17
6.5	Leftist Tree	19
6.6	Link Cut Tree	20
6.7	Heavy Light Decomposition	21
6.8	Disjoint Sets + offline skill	21
6.9	2D Segment Tree	22
7	geometry	23
7.1	Basic	23
7.2	Smallest circle problem	24
8	Others	25
8.1	Random	25
8.2	Fraction	25

1 Basic

1.1 default code

```

1 #include <bits/stdc++.h>
2 #define PB push_back
3 #define MP make_pair
4 #define F first
5 #define S second
6 #define SZ(x) ((int)(x).size())
7 #define ALL(x) (x).begin(),(x).end()
8 #ifdef _DEBUG_
9     #define debug(...) printf(__VA_ARGS__)
10 #else
11     #define debug(...) (void)0
12 #endif
13 using namespace std;
14 typedef long long ll;
15 typedef pair<int,int> PII;
16 typedef vector<int> VI;
17
18 int main() {
19     return 0;
20 }
```

1.2 .vimrc

```

1 color torte
2 syn on
3 set guifont=Consolas:h16: smd nu hls ru
4 set sc ai si ts=4 sm sts=4 sw=4
5 map <F9> <ESC>:w<CR>:!g++ % -o %< -O2 -Wall
   -Wshadow -Wno-unused-result -std=c++0x
   <CR>
6 map <S-F9> <ESC>:w<CR>:!g++ % -o %< -O2 -
   Wall -Wshadow -Wno-unused-result -
   D_DEBUG_ -std=c++0x<CR>
7 map <F5> <ESC>:!./%<<CR>
8 map <F6> <ESC>:w<CR>ggVG"+y
9 map <S-F5> <ESC>:!./%< < %<.in<CR>
10 imap <Home> <ESC>^i
11 com INPUT sp %<.in
```

2 math

2.1 ext gcd

```

1 // find one solution (x,y) of ax+by=gcd(
   a,b)
2 void ext_gcd(int a,int b,int &g,int &x,int
   &y)
3 {
4     if(!b){ g=a; x=1; y=0; }
5     else{ ext_gcd(b, a%b, g, y, x); y -= x*(a
   /b); }
6 }
```

2.2 FFT

```

1 typedef complex<double> CD;
2
3 const double PI=acos(-1.0);
4 inline CD ang(double t) { return CD(cos(t),
    sin(t)); }
5
6 int rev_int(int x,int lgn) {
7     int re=0;
8     for(int i=0;i<lgn;i++) {
9         re=(re<<1)+(x&1);
10        x>>=1;
11    }
12    return re;
13 }
14 void fft(CD* A, int lgn, bool inv=false) {
15     int n=1<<lgn;
16     for(int i=0;i<n;i++)
17         if(i<rev_int(i, lgn)) swap(A[i], A[
            rev_int(i, lgn)]);
18     for(int i=1;i<n;i*=2) {
19         CD W(1.0, 0.0), Wn;
20         if(inv) Wn=ang(-PI/i);
21         else Wn=ang(PI/i);
22         for(int j=0;j<n;j++) {
23             if(j&i) {
24                 W=CD(1.0, 0.0);
25                 continue;
26             }
27             CD x=A[j], y=A[j+i]*Wn;
28             A[j]=x+y;
29             A[j+i]=x-y;
30             W*=Wn;
31         }
32     }
33     if(inv)
34         for(int i=0;i<n;i++)
35             A[i]/=n;
36 }

```

2.3 NTT

```

1 //      MOD      Wn_      LGN
2 //      5767169   177147 19
3 //      7340033   2187 20
4 //      2013265921 440564289 27
5 const int MOD=786433;
6 const int Wn_=5; // 25 625
7 const int LGN=18; // 17 16
8 inline int add(int x,int y) { return (x+y)%
    MOD; }
9 inline int mul(int x,int y) { return 111*x*
    y%MOD; }
10 inline int sub(int x,int y) { return (x-y+
    MOD)%MOD; }
11
12 int pW[MOD]; // power of Wn
13 int divN;
14 int inv(int a) {
15     int re=1, k=MOD-2, t=a;
16     while(k) {

```

```

17         if(k%2) re=mul(re, t);
18         k/=2;
19         t=mul(t, t);
20     }
21     return re;
22 }
23 void NTTinit(int lgn) { // call every time
    using new lgn !
24     int Wn=Wn_;
25     for(int i=lgn;i<LGN;i++) Wn=mul(Wn,Wn);
26     divN=inv(1<<lgn);
27     pW[0]=1;
28     for(int i=1;i<LGN;i++) {
29         pW[i]=mul(pW[i-1], Wn);
30         if(pW[i]==1) break;
31     }
32 }
33
34 int rev_int(int x,int lgn) {
35     int re=0;
36     for(int i=0;i<lgn;i++) {
37         re=(re<<1)+(x&1);
38         x>>=1;
39     }
40     return re;
41 }
42 void ntt(int *A,int lgn,bool inv=false) {
43     int n=1<<lgn;
44     for(int i=0;i<n;i++)
45         if(i<rev_int(i,lgn))
46             swap(A[i], A[rev_int(i,lgn)]);
47     for(int i=1;i<n;i*=2) {
48         int W=1, Wn;
49         if(inv) Wn=pW[n-(n/2/i)];
50         else Wn=pW[n/2/i];
51         for(int j=0;j<n;j++) {
52             if(j&i) {
53                 W=1;
54                 continue;
55             }
56             int x=A[j], y=mul(A[j+i],W);
57             A[j]=add(x,y);
58             A[j+i]=sub(x,y);
59             W=mul(W,Wn);
60         }
61     }
62     if(inv)
63         for(int i=0;i<n;i++)
64             A[i]=mul(A[i],divN);
65 }

```

2.4 orFFT

```

1 //      1  1
2 // T =  1  0
3 //      0  1
4 //T-1=  1 -1
5 vector<ll> transform(vector<ll> P, bool
    inverse) {
6     for(int len = 1; 2 * len <= SZ(P); len
        <= 1) {

```

```

7   for(int i = 0; i < SZ(P); i += 2 * len) {
8       for(int j = 0; j < len; j++) {
9           ll u = P[i + j];
10          ll v = P[i + len + j];
11          if (!inverse) {
12              P[i + j] = u + v;
13              P[i + len + j] = u - v;
14          } else {
15              P[i + j] = u - v;
16              P[i + len + j] = u + v;
17          }
18      }
19  }
20  }
21  return P;
22 }

```

2.5 andFFT

```

1 //      0   1
2 // T =  1   1
3 //     -1   1
4 //T-1=  1   0
5 vector<ll> transform(vector<ll> P, bool
6     inverse) {
7     for(int len = 1; 2 * len <= SZ(P); len
8         <<= 1) {
9         for(int i = 0; i < SZ(P); i += 2 * len) {
10            for(int j = 0; j < len; j++) {
11                ll u = P[i + j];
12                ll v = P[i + len + j];
13                if (!inverse) {
14                    P[i + j] = u + v;
15                    P[i + len + j] = u - v;
16                } else {
17                    P[i + j] = u - v;
18                    P[i + len + j] = u + v;
19                }
20            }
21        }
22    }
23    return P;
24 }

```

2.6 xorFFT

```

1 //      1   1
2 // H =  1  -1
3 //      /sqrt(2)
4 vector<ll> FWHT(vector<ll> P, bool inverse) {
5     {
6         for(int len = 1; 2 * len <= SZ(P); len
7             <<= 1) {
8             for(int i = 0; i < SZ(P); i += 2 * len) {
9                 for(int j = 0; j < len; j++) {
10                    ll u = P[i + j];
11                    ll v = P[i + len + j];

```

```

10          P[i + j] = u + v;
11          P[i + len + j] = u - v;
12      }
13  }
14  }
15  if (inverse) {
16      for (int i = 0; i < SZ(P); i++)
17          P[i] = P[i] / SZ(P);
18  }
19  return P;
20 }

```

2.7 MillerRabin other

```

1 //input should < 2^63 - 1 (max prime
2 // :9223372036854775783)
3 typedef unsigned long long ull;
4 ull mul(ull a, ull b, ull n) {
5     ull r = 0;
6     a %= n, b %= n;
7     while(b) {
8         if(b&1) r = (a+r>=n ? a+r-n : a+r);
9         a = (a+a>=n ? a+a-n : a+a);
10        b >>= 1;
11    }
12    return r;
13 }
14 ull bigmod(ull a, ull d, ull n) {
15     if(d==0) return 1LL;
16     if(d==1) return a % n;
17     return mul(bigmod(mul(a, a, n), d/2, n),
18         d%2?a:1, n);
19 }
20 const bool PRIME = 1, COMPOSITE = 0;
21 bool miller_rabin(ull n, ull a) {
22     if(__gcd(a, n) == n) return PRIME;
23     if(__gcd(a, n) != 1) return COMPOSITE;
24     ull d = n-1, r = 0, res;
25     while(d%2==0) { ++r; d/=2; }
26     res = bigmod(a, d, n);
27     if(res == 1 || res == n-1) return PRIME;
28     while(r--) {
29         res = mul(res, res, n);
30         if(res == n-1) return PRIME;
31     }
32     return COMPOSITE;
33 }
34 bool isprime(ull n) {
35     if(n==1) return COMPOSITE;
36     ull as[7] = {2, 325, 9375, 28178, 450775,
37         9780504, 1795265022};
38     for(int i=0; i<7; i++)
39         if(miller_rabin(n, as[i]) == COMPOSITE)
40             return COMPOSITE;
41     return PRIME;
42 }
43 }

```

2.8 Guass

```

1 // be care of the magic number 7 & 8
2 void guass() {
3     for(int i = 0; i < 7; i++) {
4         Frac tmp = mat[i][i]; // Frac -> the
           type of data
5         for(int j = 0; j < 8; j++)
6             mat[i][j] = mat[i][j] / tmp;
7         for(int j = 0; j < 7; j++) {
8             if(i == j)
9                 continue;
10            Frac ratio = mat[j][i]; // Frac ->
           the type of data
11            for(int k = 0; k < 8; k++)
12                mat[j][k] = mat[j][k] - ratio * mat
                    [i][k];
13        }
14    }
15 }

```

3 flow

3.1 dinic

```

1 const int MAXV=300;
2 const int MAXE=10000;
3 const int INF=(int)1e9+10;
4 // ^ config those things
5
6 struct E {
7     int to,co;//capacity
8     E(int t=0,int c=0):to(t),co(c) {}
9 }eg[2*MAXE];
10
11 // source:0 sink:n-1
12 struct Flow {
13     VI e[MAXV];
14     int ei,v;
15     void init(int n) {
16         v=n;
17         ei=0;
18         for(int i=0;i<n;i++)
19             e[i]=VI();
20     }
21     void add(int a,int b,int c) { //a to b ,
           maxflow=c
22         eg[ei]=E(b,c);
23         e[a].PB(ei);
24         ei++;
25         eg[ei]=E(a,0);
26         e[b].PB(ei);
27         ei++;
28     }
29
30     int d[MAXV],qu[MAXV],ql,qv;
31     bool BFS() {
32         memset(d,-1,v*sizeof(int));
33         ql=qv=0;
34         qu[qv++]=0;
35         d[0]=0;

```

```

36     while(ql<qv && d[qv-1]==-1) {
37         int n=qu[qv++];
38         VI &v=e[n];
39         for(int i=SZ(v)-1;i>=0;i--) {
40             int u=v[i];
41             if(d[eg[u].to]==-1 && eg[u].co>0) {
42                 d[eg[u].to]=d[n]+1;
43                 qu[qv++]=eg[u].to;
44             }
45         }
46     }
47     return d[v-1]!=-1;
48 }
49 int ptr[MAXV];
50 int go(int n,int p) {
51     if(n==v-1)
52         return p;
53     VI &u=e[n];
54     int temp;
55     for(int i=ptr[n];i<SZ(u);i++) {
56         if(d[n]+1!=d[eg[u[i]].to] || eg[u[i]
           ].co==0)
57             continue;
58         if((temp=go(eg[u[i]].to,min(p,eg[u[i]
           ]).co))>0)
59             continue;
60         eg[u[i]].co-=temp;
61         eg[u[i]^1].co+=temp;
62         ptr[n]=i;
63         return temp;
64     }
65     ptr[n]=SZ(u);
66     return 0;
67 }
68 int max_flow() {
69     int ans=0,temp;
70     while(BFS()) {
71         for(int i=0;i<v;i++)
72             ptr[i]=0;
73         while((temp=go(0,INF))>0)
74             ans+=temp;
75     }
76     return ans;
77 }
78 }flow;

```

3.2 min-cost-max-flow

```

1 typedef pair<int,ll> PIL;
2 const int MAXV=60;
3 const int MAXE=6000;
4 const int INF=(int)1e9+10;
5 const ll cINF=(ll)1e18+10;
6 // ^ config those things
7
8 struct E {
9     int to,ca,cost;//capacity, cost
10    E(int t=0,int c=0,int co=0):to(t),ca(c),
           cost(co) {}
11 }eg[2*MAXE];
12
13 // source:0 sink:n-1

```

```

14 struct Flow {
15     VI e[MAXV];
16     int ei,n;
17     void init(int n_) {
18         n=n_;
19         ei=0;
20         for(int i=0;i<n;i++)
21             e[i]=VI();
22     }
23     void add(int a,int b,int c,int d) {
24         //a to b ,maxflow=c, cost=d
25         eg[ei]=E(b,c,d);
26         e[a].PB(ei);
27         ei++;
28         eg[ei]=E(a,0,-d);
29         e[b].PB(ei);
30         ei++;
31     }
32
33     PII d[MAXV]={};
34     bool inq[MAXV]={};
35     queue<int> que;
36     VI pe;
37     bool SPFA() {
38         fill(d, d+n, MP(INF,INF));
39         d[0]=MP(0,0);
40         que.push(0);
41         inq[0]=1;
42         while(!que.empty()) {
43             int v=que.front(); que.pop();
44             inq[v]=0;
45             for(int id:e[v]) {
46                 if(eg[id].ca>0 && MP(d[v].F+eg[id].
47                     cost,d[v].S+1)<d[eg[id].to]) {
48                     d[eg[id].to]=MP(d[v].F+eg[id].
49                         cost,d[v].S+1);
50                     if(!inq[eg[id].to]) {
51                         que.push(eg[id].to);
52                         inq[eg[id].to]=1;
53                     }
54                 }
55             }
56             return d[n-1].F<INF;
57         }
58     }
59     PIL go(ll cb=cINF) {
60         // cost_bound
61         if(!SPFA()) return MP(0,0);
62         pe.clear();
63         int fl=INF;
64         for(int v=n-1;v!=0;) {
65             for(int id:e[v]) {
66                 int u=eg[id].to;
67                 const E& t=eg[id^1];
68                 if(t.ca>0 && MP(d[u].F+t.cost,d[u].
69                     S+1)==d[v]) {
70                     fl=min(fl, t.ca);
71                     v=u;
72                     pe.PB(id^1);
73                     break;
74                 }
75             }
76         }
77     }

```

```

74         if(d[n-1].F>0) fl=min(1ll*fl, cb/d[n
75             -1].F);
76         for(int id:pe) {
77             eg[id].ca-=fl;
78             eg[id^1].ca+=fl;
79         }
80         return MP(fl, 1ll*fl*d[n-1].F);
81     }
82     PIL max_flow() {
83         PIL ans=MP(0,0),temp;
84         while((temp=go()).F>0) {
85             ans.F+=temp.F;
86             ans.S+=temp.S;
87         }
88         return ans;
89     }
90 } flow;

```

4 string

4.1 KMP

```

1 void KMP_build(const char *S,int *F) {
2     int p=F[0]=-1;
3     for(int i=1;S[i];i++) {
4         while(p!=-1 && S[p+1]!=S[i])
5             p=F[p];
6         if(S[p+1]==S[i])
7             p++;
8         F[i]=p;
9     }
10 }
11
12 VI KMP_match(const char *S,const int *F,
13     const char *T) {
14     VI ans;
15     int p=-1;
16     for(int i=0;T[i];i++) {
17         while(p!=-1 && S[p+1]!=T[i])
18             p=F[p];
19         if(S[p+1]==T[i])
20             p++;
21         if(!S[p+1]) {
22             ans.PB(i-p);
23             p=F[p];
24         }
25     }
26     return ans;
27 }

```

4.2 Z-value

```

1 void Z_build(const char *S,int *Z) {
2     Z[0]=0;
3     int bst=0;
4     for(int i=1;S[i];i++) {
5         if(Z[bst]+bst<i) Z[i]=0;
6         else Z[i]=min(Z[bst]+bst-i,Z[i-bst]);
7         while(S[Z[i]]==S[i+Z[i]]) Z[i]++;
8     }
9 }

```

```

8   if(Z[i]+i>Z[bst]+bst) bst=i;
9   }
10 }

```

4.3 Z-value-palindrome

```

1 // AC code of NTUJ1871
2 char in[100100];
3 char s[200100];
4 int z[200100];
5
6 int main()
7 {
8     while(gets(in))
9     {
10         int len=1;
11         for(int i=0;in[i];i++)
12         {
13             s[len++]='*';
14             s[len++]=in[i];
15         }
16         s[len]=0;
17         z[0]=0;
18         z[1]=0;
19         int bst=1;
20         for(int i=1;i<len;i++)
21         {
22             z[i]=min(bst+z[bst]-i,z[bst+bst-i]);
23             while(s[i+z[i]+1]==s[i-z[i]-1])
24                 z[i]++;
25             if(z[i]+i>bst+z[bst])
26                 bst=i;
27         }
28         bool yes=0;
29         for(int i=3;i<len;i+=2)
30             if(z[(i+1)/2]==i/2 && z[(i+len)/2]==(
31                 len-i-1)/2)
32                 yes=1;
33         if(yes)
34             puts("www");
35         else
36             puts("vvvvvv");
37     }
38     return 0;

```

4.4 Suffix Array($O(N\log N)$)

```

1 const int SASIZE=100020; // >= (max length
   of string + 20)
2 struct SA{
3     char S[SASIZE]; // put target string into
   S[0:(len-1)]
4     // you can change the type of S into int
   if required
5     // if the string is in int, please avoid
   number < 0
6     int R[SASIZE*2],SA[SASIZE];
7     int tR[SASIZE*2],tSA[SASIZE];
8     int cnt[SASIZE],len; // set len
   before calling build()

```

```

9     int H[SASIZE];
10
11 void build_SA() {
12     int maxR=0;
13     for(int i=0;i<len;i++)
14         R[i]=S[i];
15     for(int i=0;i<=len;i++)
16         R[len+i]=-1;
17     memset(cnt,0,sizeof(cnt));
18     for(int i=0;i<len;i++)
19         maxR=max(maxR,R[i]);
20     for(int i=0;i<len;i++)
21         cnt[R[i]+1]++;
22     for(int i=1;i<=maxR;i++)
23         cnt[i]+=cnt[i-1];
24     for(int i=0;i<len;i++)
25         SA[cnt[R[i]]++]=i;
26     for(int i=1;i<len;i*=2)
27     {
28         memset(cnt,0,sizeof(int)*(maxR+10));
29         memcpy(tSA,SA,sizeof(int)*(len+10));
30         memcpy(tR,R,sizeof(int)*(len+i+10));
31         for(int j=0;j<len;j++)
32             cnt[R[j]+1]++;
33         for(int j=1;j<=maxR;j++)
34             cnt[j]+=cnt[j-1];
35         for(int j=len-i;j<len;j++)
36             SA[cnt[R[j]]++]=j;
37         for(int j=0;j<len;j++)
38         {
39             int k=tSA[j]-i;
40             if(k<0)
41                 continue;
42             SA[cnt[R[k]]++]=k;
43         }
44         int num=0;
45         maxR=0;
46         R[SA[0]]=num;
47         for(int j=1;j<len;j++)
48         {
49             if(tR[SA[j-1]]<tR[SA[j]] || tR[SA[j]
50                 -1]+i<tR[SA[j]+i])
51                 num++;
52             R[SA[j]]=num;
53             maxR=max(maxR,R[SA[j]]);
54         }
55     }
56 void build_H() {
57     memset(H,0,sizeof(int)*(len+10));
58     for(int i=0;i<len;i++)
59     {
60         if(R[i]==0)
61             continue;
62         int &t=H[R[i]];
63         if(i>0)
64             t=max(0,H[R[i-1]]-1);
65         while(S[i+t]==S[SA[R[i]-1]+t]) t++;
66     }
67 }
68 }sa;

```

4.5 Aho-Corasick

```

1 // AC code of UVA 10679
2 struct Trie {
3     int c;
4     bool fi=0;
5     Trie *fail,*ch[52];
6     Trie():c(0){memset(ch,0,sizeof(ch));}
7 }trie[1000100];
8
9 char m[1010],f[100100];
10 Trie *str[1010],*na,*root;
11
12 inline int c_i(char a) {
13     return (a>='A' && a<='Z') ? a-'A' : a-'a'
14         +26;
15 }
16 void insert(char *s,int num) {
17     Trie *at=root;
18     while(*s) {
19         if(!at->ch[c_i(*s)])
20             at->ch[c_i(*s)]=new (na++) Trie();
21         at=at->ch[c_i(*s)],s++;
22     }
23     str[num]=at;
24 }
25
26 Trie *q[1000100];
27 int ql,qr;
28
29 void init() {
30     ql=qr=-1;
31     q[++qr]=root;
32     root->fail=NULL;
33     while(ql<qr) {
34         Trie *n=q[++ql],*f;
35         for(int i=0;i<52;i++) {
36             if(!n->ch[i])
37                 continue;
38             f=n->fail;
39             while(f && !f->ch[i])
40                 f=f->fail;
41             n->ch[i]->fail=f?f->ch[i]:root;
42             q[++qr]=n->ch[i];
43         }
44     }
45 }
46
47 void go(char *s) {
48     Trie*p=root;
49     while(*s) {
50         while(p && !p->ch[c_i(*s)])
51             p=p->fail;
52         p=p->ch[c_i(*s)]:root;
53         p->fi=1;
54         s++;
55     }
56 }
57
58 void AC() {
59     for(int i=qr;i>0;i--)
60         q[i]->fail->c+=q[i]->c;

```

```

61 }
62
63 int main() {
64     int T,q;
65     scanf("%d",&T);
66     while(T--) {
67         na=trie;
68         root=new (na++) Trie();
69         scanf("%s",f);
70         scanf("%d",&q);
71         for(int i=0;i<q;i++) {
72             scanf("%s",m);
73             insert(m,i);
74         }
75         init();
76         go(f);
77         for(int i=0;i<q;i++)
78             puts(str[i]->fi?"y":"n");
79     }
80     return 0;
81 }

```

4.6 Aho-Corasick-2016ioicamp

```

1 // AC code of 2016ioicamp 54
2 const int MAXNM=100010;
3 int pp[MAXNM];
4
5 const int sizz=100010;
6 int nx[sizz][26],spt;
7 int fl[sizz],efl[sizz],ed[sizz];
8 int len[sizz];
9 int newnode(int len_=0) {
10     for(int i=0;i<26;i++)nx[spt][i]=0;
11     ed[spt]=0;
12     len[spt]=len_;
13     return spt++;
14 }
15 int add(char *s,int p) {
16     int l=1;
17     for(int i=0;s[i];i++) {
18         int a=s[i]-'a';
19         if(nx[p][a]==0) nx[p][a]=newnode(1);
20         p=nx[p][a];
21         l++;
22     }
23     ed[p]=1;
24     return p;
25 }
26 int q[sizz],qs,qe;
27 void make_fl(int root) {
28     fl[root]=efl[root]=0;
29     qs=qe=0;
30     q[qe++]=root;
31     for(;qs!=qe;){
32         int p=q[qs++];
33         for(int i=0;i<26;i++) {
34             int t=nx[p][i];
35             if(t==0) continue;
36             int tmp=fl[p];
37             for(;tmp&&nx[tmp][i]==0;) tmp=fl[tmp];

```



```

38     fl[t]=tmp?nx[tmp][i]:root;
39     efl[t]=ed[fl[t]]?fl[t]:efl[fl[t]];
40     q[qe++]=t;
41 }
42 }
43 }
44 char s[MAXNM];
45 char a[MAXNM];
46
47 int dp[MAXNM][4];
48
49 void mmax(int &a,int b) {
50     a=max(a,b);
51 }
52
53 void match(int root) {
54     int p=root;
55     for(int i=1;s[i];i++) {
56         int a=s[i]-'a';
57         for(;p&nx[p][a]==0;p=fl[p]);
58         p=p?nx[p][a]:root;
59         for(int j=1;j<=3;j++)
60             dp[i][j]=dp[i-1][j];
61         for(int t=p;t;t=efl[t]) {
62             if(!ed[t])
63                 continue;
64             for(int j=1;j<=3;j++)
65                 mmax(dp[i][j],dp[i-len[t]][j-1]+(pp
66                     [i]-pp[i-len[t]]));
67         }
68     }
69 }
70
71 int main() {
72     int T;
73     scanf("%d",&T);
74     while(T--) {
75         int n,m;
76         scanf("%d%d",&n,&m);
77         scanf("%s",s+1);
78         for(int i=1;i<=n;i++)
79             scanf("%d",pp+i);
80         for(int i=1;i<=n;i++)
81             pp[i]+=pp[i-1];
82         spt=1;
83         int root=newnode();
84         for(int i=0;i<m;i++) {
85             scanf("%s",a);
86             add(a,root);
87         }
88         make_fl(root);
89         for(int i=1;i<=n;i++)
90             dp[i][1]=dp[i][2]=dp[i][3]=0;
91         match(root);
92         printf("%d\n",dp[n][3]);
93     }
94     return 0;

```

4.7 Palindrome Automaton

```
1|const int MAXN=100050;
```

```

2|char s[MAXN];
3|int n; // n: string length
4
5|typedef pair<PII,int> PD;
6|vector<PD> pal;
7
8|int ch[MAXN][26], fail[MAXN], len[MAXN],
9|    cnt[MAXN];
10|int edp[MAXN];
11|int nid=1;
12|int new_node(int len_) {
13|    len[nid]=len_;
14|    return nid++;
15|}
16|void build_pa() {
17|    int odd_root=new_node(-1);
18|    int even_root=new_node(0);
19|    fail[even_root]=odd_root;
20|    int cur=even_root;
21|    for(int i=1;i<=n;i++) {
22|        while(1) {
23|            if(s[i-len[cur]-1] == s[i]) break;
24|            cur=fail[cur];
25|        }
26|        if(ch[cur][s[i]-'a']==0) {
27|            int nt=ch[cur][s[i]-'a']=new_node(len
28|                [cur]+2);
29|            int tmp=fail[cur];
30|            while(tmp && s[i-len[tmp]-1]!=s[i])
31|                tmp=fail[tmp];
32|            if(tmp==0) fail[nt]=even_root;
33|            else {
34|                assert(ch[tmp][s[i]-'a']);
35|                fail[nt]=ch[tmp][s[i]-'a'];
36|            }
37|            edp[nt]=i;
38|        }
39|        cur=ch[cur][s[i]-'a'];
40|        cnt[cur]++;
41|    }
42|    for(int i=nid-1;i>even_root;i--) {
43|        cnt[fail[i]]+=cnt[i];
44|        pal.PB( MP( MP(edp[i]-len[i]+1, len[i])
45|            , cnt[i]) ));

```

4.8 Suffix Automaton(bcw)

```

1|// par : fail link
2|// val : a topological order ( useful for
3|// go[x] : automata edge ( x is integer in
4|// [0,26) )
5|struct SAM{
6|    struct State{
7|        int par, go[26], val;
8|        State () : par(0), val(0){ FZ(go); }
9|        State (int _val) : par(0), val(_val){
10|            FZ(go); }

```



```

10 };
11 vector<State> vec;
12 int root, tail;
13
14 void init(int arr[], int len){
15     vec.resize(2);
16     vec[0] = vec[1] = State(0);
17     root = tail = 1;
18     for (int i=0; i<len; i++){
19         extend(arr[i]);
20     }
21 void extend(int w){
22     int p = tail, np = vec.size();
23     vec.PB(State(vec[p].val+1));
24     for ( ; p && vec[p].go[w]==0; p=vec[p].
25         par)
26         vec[p].go[w] = np;
27     if (p == 0){
28         vec[np].par = root;
29     } else {
30         if (vec[vec[p].go[w]].val == vec[p].
31             val+1){
32             vec[np].par = vec[p].go[w];
33         } else {
34             int q = vec[p].go[w], r = vec.size
35             ();
36             vec.PB(vec[q]);
37             vec[r].val = vec[p].val+1;
38             vec[q].par = vec[np].par = r;
39             for ( ; p && vec[p].go[w] == q; p=
40                 vec[p].par)
41                 vec[p].go[w] = r;
42         }
43     }
44     tail = np;
45 }
46 };

```

5 graph

5.1 Bipartite matching($O(N^3)$)

```

1 // NTUJ1263
2 bool is(ll x)
3 {
4     ll l=1,r=2000000,m;
5     while(l<=r)
6     {
7         m=(l+r)/2;
8         if(m*m==x)
9             return 1;
10        if(m*m<x)
11            l=m+1;
12        else
13            r=m-1;
14    }
15    return 0;
16 }
17
18 VI odd,even;
19 int in[300];

```

```

20 VI e[300];
21 int match[300];
22 bool vis[300];
23
24 bool DFS(int x)
25 {
26     vis[x]=1;
27     for(int u:e[x])
28     {
29         if(match[u]==-1 || (!vis[match[u]]&&DFS
30             (match[u])))
31         {
32             match[u]=x;
33             match[x]=u;
34             return 1;
35         }
36     }
37     return 0;
38 }
39 int main()
40 {
41     int N;
42     while(scanf("%d",&N)==1)
43     {
44         odd.clear();
45         even.clear();
46         for(int i=0;i<N;i++)
47             e[i].clear();
48         for(int i=0;i<N;i++)
49         {
50             scanf("%d",&in[i]);
51             if(in[i]%2==0)
52                 even.pb(i);
53             else
54                 odd.pb(i);
55         }
56         for(int i:even)
57             for(int j:odd)
58                 if(is(1ll*in[i]*in[i]+1ll*in[j]*in[
59                     j]) && __gcd(in[i],in[j])==1)
60                     e[i].pb(j), e[j].pb(i);
61         int ans=0;
62         fill(match,match+N,-1);
63         for(int i=0;i<N;i++)
64             if(match[i]==-1)
65             {
66                 fill(vis,vis+N,0);
67                 if(DFS(i))
68                     ans++;
69             }
70         printf("%d\n",ans);
71     }
72     return 0;

```

5.2 KM($O(N^4)$)

```

1 const int INF=1016; //> max(a[i][j])
2 const int MAXN=650;
3 int a[MAXN][MAXN]; // weight [x][y] , two
4     set of vertex

```

```

4 int N; // two set: each set have exactly N
   vertex
5 int match[MAXN*2], weight[MAXN*2];
6 bool vis[MAXN*2];
7
8 bool DFS(int x) {
9     vis[x]=1;
10    for(int i=0;i<N;i++) {
11        if(weight[x]+weight[N+i]!=a[x][i])
12            continue;
13        vis[N+i]=1;
14        if(match[N+i]==-1 || (!vis[match[N+i]]
15            ]&&DFS(match[N+i]))) {
16            match[N+i]=x;
17            match[x]=N+i;
18            return 1;
19        }
20    }
21    return 0;
22 }
23 int KM() {
24     fill(weight, weight+N*N, 0);
25     for(int i=0;i<N;i++) {
26         for(int j=0;j<N;j++)
27             weight[i]=max(weight[i], a[i][j]);
28     }
29     fill(match, match+N*N, -1);
30     for(int u=0;u<N;u++) {
31         fill(vis, vis+N*N, 0);
32         while(!DFS(u)) {
33             int d=INF;
34             for(int i=0;i<N;i++) {
35                 if(!vis[i]) continue;
36                 for(int j=0;j<N;j++)
37                     if(!vis[N+j])
38                         d=min(d, weight[i]+weight[N+j]-
39                             a[i][j]);
40             }
41             for(int i=0;i<N;i++)
42                 if(vis[i])
43                     weight[i]-=d;
44             for(int i=N;i<N*N;i++)
45                 if(vis[i])
46                     weight[i]+=d;
47             fill(vis, vis+N*N, 0);
48         }
49         int ans=0;
50         for(int i=0;i<N*N;i++) ans+=weight[i];
51     }
52 }
53
54 queue<int> qe;
55 int st,ed;
56 int nb;
57 int bk[MAXN],djs[MAXN];
58 int ans;
59 void init(int _V) {
60     V = _V;
61     FZ(el); FZ(pr);
62     FZ(inq); FZ(inp); FZ(inb);
63     FZ(bk); FZ(djs);
64     ans = 0;
65 }
66 void add_edge(int u, int v) {
67     el[u][v] = el[v][u] = 1;
68 }
69 int lca(int u,int v) {
70     memset(inp,0,sizeof(inp));
71     while(1) {
72         u = djs[u];
73         inp[u] = true;
74         if(u == st) break;
75         u = bk[pr[u]];
76     }
77     while(1) {
78         v = djs[v];
79         if(inp[v]) return v;
80         v = bk[pr[v]];
81     }
82     return v;
83 }
84 void upd(int u) {
85     int v;
86     while(djs[u] != nb) {
87         v = pr[u];
88         inb[djs[u]] = inb[djs[v]] = true;
89         u = bk[v];
90         if(djs[u] != nb) bk[u] = v;
91     }
92 }
93 void blo(int u,int v) {
94     nb = lca(u,v);
95     memset(inb,0,sizeof(inb));
96     upd(u); upd(v);
97     if(djs[u] != nb) bk[u] = v;
98     if(djs[v] != nb) bk[v] = u;
99     for(int tu = 1; tu <= V; tu++)
100         if(inb[djs[tu]]) {
101             djs[tu] = nb;
102             if(!inq[tu]){
103                 qe.push(tu);
104                 inq[tu] = 1;
105             }
106         }
107 }
108 void flow() {
109     memset(inq,false,sizeof(inq));
110     memset(bk,0,sizeof(bk));
111     for(int i = 1; i <= V;i++)
112         djs[i] = i;
113     while(qe.size()) qe.pop();
114     qe.push(st);
115     inq[st] = 1;
116     ed = 0;

```

5.3 general graph matching(bcw)

```

1 #define FZ(x) memset(x,0,sizeof(x))
2 struct GenMatch { // 1-base
3     static const int MAXN = 250;
4     int V;
5     bool el[MAXN][MAXN];
6     int pr[MAXN];
7     bool inq[MAXN],inp[MAXN],inb[MAXN];

```

```

72 while(qe.size()) {
73     int u = qe.front(); qe.pop();
74     for(int v = 1; v <= V; v++)
75         if(el[u][v] && (djs[u] != djs[v])
76             && (pr[u] != v)) {
77             if((v == st) || ((pr[v] > 0) &&
78                 bk[pr[v]] > 0))
79                 blo(u,v);
80             else if(bk[v] == 0) {
81                 bk[v] = u;
82                 if(pr[v] > 0) {
83                     if(!inq[pr[v]]) qe.push(pr[v]);
84                 } else {
85                     ed = v;
86                     return;
87                 }
88             }
89         }
90 void aug() {
91     int u,v,w;
92     u = ed;
93     while(u > 0) {
94         v = bk[u];
95         w = pr[v];
96         pr[v] = u;
97         pr[u] = v;
98         u = w;
99     }
100 }
101 int solve() {
102     memset(pr,0,sizeof(pr));
103     for(int u = 1; u <= V; u++)
104         if(pr[u] == 0) {
105             st = u;
106             flow();
107             if(ed > 0) {
108                 aug();
109                 ans ++;
110             }
111         }
112     return ans;
113 }
114 } gm;

```

5.4 minimum general graph weighted matching(bcw)

```

1 struct Graph {
2     // Minimum General Weighted Matching (
3     // Perfect Match) 0-base
4     static const int MXN = 105;
5     int n, edge[MXN][MXN];
6     int match[MXN],dis[MXN],onstk[MXN];
7     vector<int> stk;
8     void init(int _n) {
9         n = _n;
10        for (int i=0; i<n; i++)

```

```

12        for (int j=0; j<n; j++)
13            edge[i][j] = 0;
14    }
15    void add_edge(int u, int v, int w) {
16        edge[u][v] = edge[v][u] = w;
17    }
18    bool SPFA(int u){
19        if (onstk[u]) return true;
20        stk.PB(u);
21        onstk[u] = 1;
22        for (int v=0; v<n; v++){
23            if (u != v && match[u] != v && !onstk
24                [v]){
25                int m = match[v];
26                if (dis[m] > dis[u] - edge[v][m] +
27                    edge[u][v]){
28                    dis[m] = dis[u] - edge[v][m] +
29                        edge[u][v];
30                    onstk[v] = 1;
31                    stk.PB(v);
32                    if (SPFA(m)) return true;
33                    stk.pop_back();
34                    onstk[v] = 0;
35                }
36            }
37            onstk[u] = 0;
38            stk.pop_back();
39            return false;
40        }
41    }
42    int solve() {
43        // find a match
44        for (int i=0; i<n; i+=2){
45            match[i] = i+1;
46            match[i+1] = i;
47        }
48        while (true){
49            int found = 0;
50            for (int i=0; i<n; i++)
51                dis[i] = onstk[i] = 0;
52            for (int i=0; i<n; i++){
53                stk.clear();
54                if (!onstk[i] && SPFA(i)){
55                    found = 1;
56                    while (SZ(stk)>=2){
57                        int u = stk.back(); stk.
58                            pop_back();
59                        int v = stk.back(); stk.
60                            pop_back();
61                        match[u] = v;
62                        match[v] = u;
63                    }
64                }
65            }
66            if (!found) break;
67        }
68        int ret = 0;
69        for (int i=0; i<n; i++)
70            ret += edge[i][match[i]];
71        ret /= 2;
72        return ret;
73    }
74 }graph;

```

5.5 Max clique(bcw)

```

1 class MaxClique {
2 public:
3     static const int MV = 210;
4
5     int V;
6     int el[MV][MV/30+1];
7     int dp[MV];
8     int ans;
9     int s[MV][MV/30+1];
10    vector<int> sol;
11
12    void init(int v) {
13        V = v; ans = 0;
14        FZ(el); FZ(dp);
15    }
16
17    /* Zero Base */
18    void addEdge(int u, int v) {
19        if(u > v) swap(u, v);
20        if(u == v) return;
21        el[u][v/32] |= (1<<(v%32));
22    }
23
24    bool dfs(int v, int k) {
25        int c = 0, d = 0;
26        for(int i=0; i<(V+31)/32; i++) {
27            s[k][i] = el[v][i];
28            if(k != 1) s[k][i] &= s[k-1][i];
29            c += __builtin_popcount(s[k][i]);
30        }
31        if(c == 0) {
32            if(k > ans) {
33                ans = k;
34                sol.clear();
35                sol.push_back(v);
36                return 1;
37            }
38            return 0;
39        }
40        for(int i=0; i<(V+31)/32; i++) {
41            for(int a = s[k][i]; a ; d++) {
42                if(k + (c-d) <= ans) return 0;
43                int lb = a&(-a), lg = 0;
44                a ^= lb;
45                while(lb!=1) {
46                    lb = (unsigned int)(lb) >> 1;
47                    lg ++;
48                }
49                int u = i*32 + lg;
50                if(k + dp[u] <= ans) return 0;
51                if(dfs(u, k+1)) {
52                    sol.push_back(v);
53                    return 1;
54                }
55            }
56        }
57        return 0;
58    }
59
60    int solve() {
61        for(int i=V-1; i>=0; i--) {

```

```

62            dfs(i, 1);
63            dp[i] = ans;
64        }
65        return ans;
66    }
67 };

```

5.6 EdgeBCC

```

1 const int MAXN=1010;
2 const int MAXM=5010;
3 VI e[MAXN];
4 int low[MAXN],lvl[MAXN],bel[MAXN];
5 bool vis[MAXN];
6 int cnt;
7 VI st;
8 void DFS(int x,int l,int p) {
9     st.PB(x);
10    vis[x]=1;
11    low[x]=lvl[x]=1;
12    bool top=0;
13    for(int u:e[x]) {
14        if(u==p && !top) {
15            top=1;
16            continue;
17        }
18        if(!vis[u]) {
19            DFS(u,l+1,x);
20        }
21        low[x]=min(low[x],low[u]);
22    }
23    if(x==1 || low[x]==1) {
24        while(st.back()!=x) {
25            bel[st.back()]=cnt;
26            st.pop_back();
27        }
28        bel[st.back()]=cnt;
29        st.pop_back();
30        cnt++;
31    }
32 }
33 int main() {
34     int T;
35     scanf("%d",&T);
36     while(T--) {
37         int N,M,a,b;
38         scanf("%d%d",&N,&M);
39         fill(vis,vis+N+1,0);
40         for(int i=1;i<=N;i++)
41             e[i].clear();
42         while(M--) {
43             scanf("%d%d",&a,&b);
44             e[a].PB(b);
45             e[b].PB(a);
46         }
47         cnt=0;
48         DFS(1,0,-1);
49         /******/
50     }
51     return 0;
52 }

```

5.7 VerticeBCC

```

1 const int MAXN=10000;
2 const int MAXE=100000;
3
4 VI e[MAXN+10];
5 vector<PII> BCC[MAXE];
6 int bccnt;
7 vector<PII> st;
8 bool vis[MAXN+10];
9 int low[MAXN+10], level[MAXN+10];
10
11 void DFS(int x,int p,int l) {
12     vis[x]=1;
13     level[x]=low[x]=l;
14     for(int u:e[x]) {
15         if(u==p)
16             continue;
17         if(vis[u]) {
18             if(level[u]<l) {
19                 st.PB(MP(x,u));
20                 low[x]=min(low[x],level[u]);
21             }
22         }
23         else {
24             st.PB(MP(x,u));
25             DFS(u,x,l+1);
26             if(low[u]>=l) {
27                 PII t=st.back();
28                 st.pop_back();
29                 while(t!=MP(x,u)) {
30                     BCC[bccnt].PB(t);
31                     t=st.back();
32                     st.pop_back();
33                 }
34                 BCC[bccnt].PB(t);
35                 bccnt++;
36             }
37             low[x]=min(low[x],low[u]);
38         }
39     }
40 }
41
42 int main() {
43     int T,N,M;
44     scanf("%d",&T);
45     while(T--) {
46         scanf("%d%d",&N,&M);
47         for(int i=0;i<N;i++)
48             e[i].clear();
49         int cnt=0;
50         while(1) {
51             int x,y;
52             scanf("%d%d",&x,&y);
53             if(x==-1 && y==-1)
54                 break;
55             cnt++;
56             e[x].PB(y);
57             e[y].PB(x);
58         }
59         for(int i=0;i<N;i++) { // no multi-edge
60             sort(ALL(e[i]));

```

```

61             e[i].erase(unique(ALL(e[i])),e[i].end
62                 ());
63         }
64         fill(vis,vis+N,0);
65         while(bccnt)
66             BCC[--bccnt].clear();
67         DFS(0,-1,0);
68         /**/
69     }
70     return 0;

```

5.8 Dominating Tree

```

1 const int MAXN = 200000 + 10;
2
3 VI e[MAXN], re[MAXN];
4 int par[MAXN], num[MAXN], t, rn[MAXN];
5 int sd[MAXN], id[MAXN];
6 PII p[MAXN];
7 VI sdom_at[MAXN];
8
9 void dfs(int u) {
10     num[u] = ++t;
11     rn[t] = u;
12     for(int v : e[u]) {
13         if(num[v]) continue;
14         par[v] = u;
15         dfs(v);
16     }
17 }
18
19 void LINK(int x, int y) {
20     p[x].F = y;
21     if(sd[y] < sd[p[x].S]) p[x].S = y;
22 }
23
24 int EVAL(int x) {
25     if(p[p[x].F].F != p[x].F) {
26         int w = EVAL(p[x].F);
27         if(sd[w] < sd[p[x].S]) p[x].S = w;
28         p[x].F = p[p[x].F].F;
29     }
30     return p[x].S;
31 }
32
33 void DominatingTree(int n) {
34     // 1-indexed
35     par[1] = 1;
36     fill(num, num+n+1, 0);
37     fill(rn, rn+n+1, 0);
38     t = 0;
39     dfs(1);
40
41     for(int i=1; i<=n; i++) {
42         p[i] = MP(i, i);
43     }
44     for(int i=1; i<=n; i++) {
45         sd[i] = (num[i] ? num[i] : MAXN+10);
46         id[i] = i;
47     }
48     for(int i=n; i>1; i--) {

```

```

49 int v = rn[i];
50 if(!v) continue;
51 for(int u : re[v]) {
52     int w = EVAL(u);
53     sd[v] = min(sd[v], sd[w]);
54 }
55 sdom_at[rn[sd[v]]].PB(v);
56 LINK(v, par[v]);
57
58 for(int w : sdom_at[par[v]]) {
59     int u = EVAL(w);
60     id[w] = (sd[u] < sd[w] ? u : par[v]);
61 }
62 sdom_at[par[v]].clear();
63 }
64
65 for(int i=2; i<=n; i++) {
66     int v = rn[i];
67     if(!v) break;
68     if(id[v] != rn[sd[v]]) id[v] = id[id[v]
        ];
69 }
70 }

```

5.9 Them.

1. Max (vertex) independent set = Max clique on Complement graph
2. Min vertex cover = $|V|$ - Max independent set
3. On bipartite: Min vertex cover = Max Matching(edge independent)
4. Any graph with no isolated vertices: Min edge cover + Max Matching = $|V|$

6 data structure

6.1 Treap

```

1 const int N = 100000 + 10;
2
3 struct Treap {
4     static Treap mem[N], *pmem;
5
6     int sz, pri;
7     ll val, sum, add;
8     Treap *l, *r;
9
10    Treap() {}
11    Treap(ll _val):
12        l(NULL), r(NULL), sz(1), pri(rand()),
13        val(_val), sum(_val), add(0) {}
14 } Treap::mem[N], *Treap::pmem = Treap::mem;
15
16 Treap* make(ll val) {
17     return new (Treap::pmem++) Treap(val);
18 }
19 inline int sz(Treap *t) {

```

```

20     return t ? t->sz : 0;
21 }
22
23 inline ll sum(Treap *t) {
24     return t ? t->sum + t->add * sz(t) : 0;
25 }
26
27 inline void add(Treap *t, ll x) {
28     t->add += x;
29 }
30
31 void push(Treap *t) {
32     t->val += t->add;
33     if(t->l) t->l->add += t->add;
34     if(t->r) t->r->add += t->add;
35     t->add = 0;
36 }
37
38 void pull(Treap *t) {
39     t->sum = sum(t->l) + sum(t->r) + t->val;
40     t->sz = sz(t->l) + sz(t->r) + 1;
41 }
42
43 Treap* merge(Treap *a, Treap *b) {
44     if(!a || !b) return a ? a : b;
45     else if(a->pri > b->pri) {
46         push(a);
47         a->r = merge(a->r, b);
48         pull(a);
49         return a;
50     }
51     else {
52         push(b);
53         b->l = merge(a, b->l);
54         pull(b);
55         return b;
56     }
57 }
58
59 void split(Treap* t, int k, Treap *&a,
60     Treap *&b) {
61     if(!t) a = b = NULL;
62     else if(sz(t->l) < k) {
63         a = t;
64         push(a);
65         split(t->r, k - sz(t->l) - 1, a->r, b);
66         pull(a);
67     }
68     else {
69         b = t;
70         push(b);
71         split(t->l, k, a, b->l);
72         pull(b);
73     }
74 }
75
76 int main() {
77     srand(105105);
78
79     int n, q;
80     scanf("%d%d", &n, &q);
81
82     Treap *t = NULL;
83     for(int i = 0; i < n; i++) {

```

```

83     ll tmp;
84     scanf("%lld", &tmp);
85     t = merge(t, make(tmp));
86 }
87
88 while(q--) {
89     char c;
90     int l, r;
91     scanf("\n%c %d %d", &c, &l, &r);
92
93     Treap *tl = NULL, *tr = NULL;
94     if(c == 'Q') {
95         split(t, l - 1, tl, t);
96         split(t, r - l + 1, t, tr);
97         printf("%lld\n", sum(t));
98         t = merge(tl, merge(t, tr));
99     }
100    else {
101        ll x;
102        scanf("%lld", &x);
103        split(t, l - 1, tl, t);
104        split(t, r - l + 1, t, tr);
105        add(t, x);
106        t = merge(tl, merge(t, tr));
107    }
108 }
109
110 return 0;
111 }

```

6.2 copy on write treap

```

1  const int N = 1000000 + 10;
2
3  struct Treap {
4      char val;
5      int sz, refs;
6      Treap *l, *r;
7
8      Treap() {}
9      Treap(char _val):
10         val(_val), sz(1), refs(0), l(NULL), r(
11         NULL) {}
12 };
13
14 Treap* make(Treap* t) {
15     return new Treap(*t);
16 }
17
18 Treap* make(char _val) {
19     return new Treap(_val);
20 }
21
22 void print_ref(Treap* t) {
23     if(!t) return;
24     print_ref(t->l);
25     printf("%d ", t->refs);
26     print_ref(t->r);
27 }
28
29 void print(Treap* t) {
30     if(!t) return;

```

```

30     print(t->l);
31     putchar(t->val);
32     print(t->r);
33 }
34
35 void takeRef(Treap* t) {
36     if(t) t->refs++;
37 }
38
39 void dropRef(Treap* t) {
40     if(t) {
41         char c = t->val;
42         t->refs--;
43         if(t->refs <= 0) {
44             dropRef(t->l);
45             dropRef(t->r);
46             delete t;
47         }
48     }
49 }
50
51 int sz(Treap* t) {
52     return t ? t->sz : 0;
53 }
54
55 int rnd(int m) {
56     static int x = 851025;
57     return (x = (x*0xdefaced+1) & INT_MAX) %
58         m;
59 }
60
61 void pull(Treap* t) {
62     t->sz = sz(t->l) + sz(t->r) + 1;
63 }
64
65 Treap* merge(Treap* a, Treap* b) {
66     if(!a || !b) {
67         Treap* t = a ? make(a) : make(b);
68         t->refs = 0;
69         takeRef(t->l);
70         takeRef(t->r);
71         return t;
72     }
73
74     Treap* t;
75     if(rnd(a->sz+b->sz) < a->sz) {
76         t = make(a);
77         t->refs = 0;
78         t->r = merge(a->r, b);
79         takeRef(t->l);
80         takeRef(t->r);
81     }
82     else {
83         t = make(b);
84         t->refs = 0;
85         t->l = merge(a, b->l);
86         takeRef(t->l);
87         takeRef(t->r);
88     }
89
90     pull(t);
91     return t;
92 }

```



```

93 void split(Treap* t, int k, Treap* &a,
    Treap* &b) {
94     if(!t) a = b = NULL;
95     else if(sz(t->l) < k) {
96         a = make(t);
97         a->refs = 0;
98         split(a->r, k-sz(t->l)-1, a->r, b);
99         takeRef(a->l);
100        takeRef(a->r);
101        pull(a);
102    }
103    else {
104        b = make(t);
105        b->refs = 0;
106        split(b->l, k, a, b->l);
107        takeRef(b->l);
108        takeRef(b->r);
109        pull(b);
110    }
111 }

```

```

113 void print_inorder(Treap* t) {
114     if(!t) return;
115     putchar(t->val);
116     print_inorder(t->l);
117     print_inorder(t->r);
118 }
119
120 char s[N];
121
122 int main() {
123     int m;
124     scanf("%d", &m);
125     scanf("%s", s);
126     int n = strlen(s);
127     int q;
128     scanf("%d", &q);
129
130     Treap* t = NULL;
131     for(int i = 0; i < n; i++) {
132         Treap *a = t, *b = make(s[i]);
133         t = merge(a, b);
134         dropRef(a);
135         dropRef(b);
136     }
137
138     while(q--) {
139         int l, r, x;
140         scanf("%d%d%d", &l, &r, &x);
141         r++;
142
143         Treap *a, *b, *c, *d;
144         a = b = c = d = NULL;
145         split(t, l, a, b);
146         dropRef(a);
147         split(b, r-l, c, d);
148         dropRef(b);
149         dropRef(d);
150         split(t, x, a, b);
151         dropRef(t);
152         Treap* t2 = merge(c, b);
153         dropRef(b);
154         dropRef(c);
155         t = merge(a, t2);

```

```

156         dropRef(a);
157         dropRef(t2);
158
159         if(t->sz > m) {
160             Treap* t2 = NULL;
161             split(t, m, t2, a);
162             dropRef(a);
163             dropRef(t);
164             t = t2;
165         }
166     }
167
168     print(t);
169     putchar('\n');
170
171     return 0;
172 }

```

6.3 copy on write segment tree

```

1  const int N = 50000 + 10;
2  const int Q = 10000 + 10;
3
4  struct Seg {
5      static Seg mem[N*80], *pmem;
6
7      int val;
8      Seg *tl, *tr;
9
10     Seg() :
11         tl(NULL), tr(NULL), val(0) {}
12
13     Seg* init(int l, int r) {
14         Seg* t = new (pmem++) Seg();
15         if(l != r) {
16             int m = (l+r)/2;
17             t->tl = init(l, m);
18             t->tr = init(m+1, r);
19         }
20         return t;
21     }
22
23     Seg* add(int k, int l, int r) {
24         Seg* _t = new (pmem++) Seg(*this);
25         if(l==r) {
26             _t->val++;
27             return _t;
28         }
29
30         int m = (l+r)/2;
31         if(k <= m) _t->tl = tl->add(k, l, m);
32         else _t->tr = tr->add(k, m+1, r);
33
34         _t->val = _t->tl->val + _t->tr->val;
35         return _t;
36     }
37 } Seg::mem[N*80], *Seg::pmem = mem;
38
39 int query(Seg* ta, Seg* tb, int k, int l,
40     int r) {
41     if(l == r) return 1;

```

```

42 int m = (l+r)/2;
43
44 int a = ta->t1->val;
45 int b = tb->t1->val;
46 if(b-a >= k) return query(ta->t1, tb->t1
    , k, l, m);
47 else return query(ta->tr, tb->tr, k
    -(b-a), m+1, r);
48 };
49
50 struct Query {
51     int op, l, r, k, c, v;
52
53     bool operator<(const Query b) const {
54         return c < b.c;
55     }
56 } qs[Q];
57 int arr[N];
58 Seg *t[N];
59 vector<int> vec2;
60
61 int main() {
62     int T;
63     scanf("%d", &T);
64
65     while(T--) {
66         int n, q;
67         scanf("%d%d", &n, &q);
68
69         for(int i = 1; i <= n; i++) {
70             scanf("%d", arr+i);
71             vec2.push_back(arr[i]);
72         }
73         for(int i = 0; i < q; i++) {
74             scanf("%d", &qs[i].op);
75             if(qs[i].op == 1) scanf("%d%d%d", &qs
                [i].l, &qs[i].r, &qs[i].k);
76             else scanf("%d%d", &qs[i].c, &qs[i].
                v);
77
78             if(qs[i].op == 2) vec2.push_back(qs[i]
                .v);
79         }
80         sort(vec2.begin(), vec2.end());
81         vec2.resize(unique(vec2.begin(), vec2.
            end())-vec2.begin());
82         for(int i = 1; i <= n; i++) arr[i] =
            lower_bound(vec2.begin(), vec2.end
                (), arr[i]) - vec2.begin();
83         int mn = 0, mx = vec2.size()-1;
84
85         for(int i = 0; i <= n; i++) t[i] = NULL
            ;
86         t[0] = new (Seg::pmem++) Seg();
87         t[0] = t[0]->init(mn, mx);
88         int ptr = 0;
89         for(int i = 1; i <= n; i++) {
90             t[i] = t[i-1]->add(arr[i], mn, mx);
91         }
92
93         for(int i = 0; i < q; i++) {
94             int op = qs[i].op;
95             if(op == 1) {
96                 int l = qs[i].l, r = qs[i].r, k =
                    qs[i].k;
97                 printf("%d\n", vec2[query(t[l-1], t
                    [r], k, mn, mx)]);
98             }
99             if(op == 2) {
100                 continue;
101             }
102             if(op == 3) puts("7122");
103         }
104
105         vec2.clear();
106         Seg::pmem = Seg::mem;
107     }
108
109     return 0;
110 }

```

6.4 Treap+(HOJ 92)

```

1 const int INF = 103456789;
2
3 struct Treap {
4     int pri, sz, val, chg, rev, sum, lsum,
        rsum, mx_sum;
5     Treap *l, *r;
6
7     Treap() {}
8     Treap(int _val) :
9         pri(rand()), sz(1), val(_val), chg(INF)
        , rev(0), sum(_val), lsum(_val),
        rsum(_val), mx_sum(_val), l(NULL),
        r(NULL) {}
10 };
11
12 int sz(Treap* t) {return t ? t->sz : 0;}
13 int sum(Treap* t) {
14     if(!t) return 0;
15     if(t->chg == INF) return t->sum;
16     else return t->chg*t->sz;
17 }
18 int lsum(Treap* t) {
19     if(!t) return -INF;
20     if(t->chg != INF) return max(t->chg, (t
        ->chg)*(t->sz));
21     if(t->rev) return t->rsum;
22     return t->lsum;
23 }
24 int rsum(Treap* t) {
25     if(!t) return -INF;
26     if(t->chg != INF) return max(t->chg, (t
        ->chg)*(t->sz));
27     if(t->rev) return t->lsum;
28     return t->rsum;
29 }
30 int mx_sum(Treap* t) {
31     if(!t) return -INF;
32     if(t->chg != INF) return max(t->chg, (t
        ->chg)*(t->sz));
33     return t->mx_sum;
34 }
35

```

```

36 void push(Treap* t) {
37     if(t->chg != INF) {
38         t->val = t->chg;
39         t->sum = (t->sz) * (t->chg);
40         t->lsum = t->rsum = t->mx_sum = max(t->
            sum, t->val);
41         if(t->l) t->l->chg = t->chg;
42         if(t->r) t->r->chg = t->chg;
43         t->chg = INF;
44     }
45     if(t->rev) {
46         swap(t->l, t->r);
47         if(t->l) t->l->rev ^= 1;
48         if(t->r) t->r->rev ^= 1;
49         t->rev = 0;
50     }
51 }
52
53 void pull(Treap* t) {
54     t->sz = sz(t->l)+sz(t->r)+1;
55     t->sum = sum(t->l)+sum(t->r)+t->val;
56     t->lsum = max(lsum(t->l), sum(t->l)+max
        (0, lsum(t->r))+t->val);
57     t->rsum = max(rsum(t->r), sum(t->r)+max
        (0, rsum(t->l))+t->val);
58     t->mx_sum = max(max(mx_sum(t->l), mx_sum(
        t->r)), max(0, rsum(t->l))+max(0,
        lsum(t->r))+t->val);
59 }
60
61 Treap* merge(Treap* a, Treap* b) {
62     if(!a || !b) return a ? a : b;
63     if(a->pri > b->pri) {
64         push(a);
65         a->r = merge(a->r, b);
66         pull(a);
67         return a;
68     }
69     else {
70         push(b);
71         b->l = merge(a, b->l);
72         pull(b);
73         return b;
74     }
75 }
76
77 void split(Treap* t, int k, Treap* &a,
    Treap* &b) {
78     if(!t) {
79         a = b = NULL;
80         return ;
81     }
82     push(t);
83     if(sz(t->l) < k) {
84         a = t;
85         push(a);
86         split(t->r, k-sz(t->l)-1, a->r, b);
87         pull(a);
88     }
89     else {
90         b = t;
91         push(b);
92         split(t->l, k, a, b->l);
93         pull(b);
94     }
95 }
96
97 void del(Treap* t) {
98     if(!t) return;
99     del(t->l);
100    del(t->r);
101    delete t;
102 }
103
104 int main() {
105     srand(7122);
106
107     int n, m;
108     scanf("%d%d", &n, &m);
109
110     Treap* t = NULL;
111     for(int i = 0; i < n; i++) {
112         int x;
113         scanf("%d", &x);
114         t = merge(t, new Treap(x));
115     }
116
117     while(m--) {
118         char s[15];
119         scanf("%s", s);
120
121         Treap *t1 = NULL, *tr = NULL, *t2 =
            NULL;
122
123         if(!strcmp(s, "INSERT")) {
124             int p, k;
125             scanf("%d%d", &p, &k);
126             for(int i = 0; i < k; i++) {
127                 int x;
128                 scanf("%d", &x);
129                 t2 = merge(t2, new Treap(x));
130             }
131             split(t, p, t1, tr);
132             t = merge(t1, merge(t2, tr));
133         }
134
135         if(!strcmp(s, "DELETE")) {
136             int p, k;
137             scanf("%d%d", &p, &k);
138             split(t, p-1, t1, t);
139             split(t, k, t, tr);
140             del(t);
141             t = merge(t1, tr);
142         }
143
144         if(!strcmp(s, "MAKE-SAME")) {
145             int p, k, l;
146             scanf("%d%d%d", &p, &k, &l);
147             split(t, p-1, t1, t);
148             split(t, k, t, tr);
149             if(t) t->chg = l;
150             t = merge(t1, merge(t, tr));
151         }
152
153         if(!strcmp(s, "REVERSE")) {
154             int p, k;
155             scanf("%d%d", &p, &k);
156             split(t, p-1, t1, t);

```

```

157     split(t, k, t, tr);
158     if(t) t->rev ^= 1;
159     t = merge(tl, merge(t, tr));
160 }
161
162 if(!strcmp(s, "GET-SUM")) {
163     int p, k;
164     scanf("%d%d", &p, &k);
165     split(t, p-1, tl, t);
166     split(t, k, t, tr);
167     printf("%d\n", sum(t));
168     t = merge(tl, merge(t, tr));
169 }
170
171 if(!strcmp(s, "MAX-SUM")) {
172     printf("%d\n", mx_sum(t));
173 }
174 }
175
176 return 0;
177 }

```

6.5 Leftist Tree

```

1 struct Left {
2     Left *l,*r;
3     int v,h;
4     Left(int v_) : v(v_), h(1), l(0), r(0) {}
5 };
6
7 int height(Left *p) { return p ? p->h : 0; }
8
9 Left* combine(Left *a, Left *b) {
10     if(!a || !b) return a ? a : b;
11     Left *p;
12     if( a->v > b->v ) {
13         p = a;
14         p->r = combine( p->r , b );
15     }
16     else {
17         p = b;
18         p->r = combine( p->r , a );
19     }
20     if( height( p->l ) < height( p->r ) )
21         swap( p->l , p->r );
22     p->h = min( height( p->l ) , height( p->r ) ) + 1;
23     return p;
24 }
25 Left *root;
26
27 void push(int v) {
28     Left *p = new Left(v);
29     root = combine( root , p );
30 }
31 int top() { return root? root->v : -1; }
32 void pop() {
33     if(!root) return;
34     Left *a = root->l , *b = root->r ;
35     delete root;
36     root = combine( a , b );

```

```

37 }
38 void clear(Left* &p) {
39     if(!p)
40         return;
41     if(p->l) clear(p->l);
42     if(p->r) clear(p->r);
43     delete p;
44     p = 0;
45 }
46
47 int main() {
48     int T,n,x,o,size;
49     bool bst,bqu,bpq;
50     scanf("%d",&T);
51     while(T-->0) {
52         bst=bqu=bpq=1;
53         stack<int> st;
54         queue<int> qu;
55         clear(root);
56         size=0;
57         scanf("%d",&n);
58         while(n-->0) {
59             scanf("%d%d",&o,&x);
60             if(o==1)
61                 st.push(x),qu.push(x),push(x),size++;
62             else if(o==2) {
63                 size--;
64                 if(size<0)
65                     bst=bqu=bpq=0;
66                 if(bst) {
67                     if(st.top()!=x)
68                         bst=0;
69                     st.pop();
70                 }
71                 if(bqu) {
72                     if(qu.front()!=x)
73                         bqu=0;
74                     qu.pop();
75                 }
76                 if(bpq) {
77                     // printf("(%d)\n",top());
78                     if(top()!=x)
79                         bpq=0;
80                     pop();
81                 }
82             }
83         }
84         int count=0;
85         if(bst) count++;
86         if(bqu) count++;
87         if(bpq) count++;
88         if(count>1)
89             puts("not sure");
90         else if(count==0)
91             puts("impossible");
92         else if(bst)
93             puts("stack");
94         else if(bqu)
95             puts("queue");
96     }

```

```

100     else if(bpq)
101         puts("priority queue");
102     }
103     return 0;
104 }

```

6.6 Link Cut Tree

```

1  const int MAXN = 100000 + 10;
2
3  struct SplayTree {
4      int val, mx, ch[2], pa;
5      bool rev;
6      void init() {
7          val = mx = -1;
8          rev = false;
9          pa = ch[0] = ch[1] = 0;
10     }
11 } node[MAXN*2];
12
13 inline bool isroot(int x) {
14     return node[node[x].pa].ch[0]!=x && node[
15         node[x].pa].ch[1]!=x;
16 }
17
18 inline void pull(int x) {
19     node[x].mx = max(node[x].val, max(node[
20         node[x].ch[0]].mx, node[node[x].ch
21         [1]].mx));
22 }
23
24 inline void push(int x) {
25     if(node[x].rev) {
26         node[node[x].ch[0]].rev ^= 1;
27         node[node[x].ch[1]].rev ^= 1;
28         swap(node[x].ch[0], node[x].ch[1]);
29         node[x].rev ^= 1;
30     }
31 }
32
33 void push_all(int x) {
34     if(!isroot(x)) push_all(node[x].pa);
35     push(x);
36 }
37
38 inline void rotate(int x) {
39     int y = node[x].pa, z = node[y].pa, d =
40         node[y].ch[1]==x;
41     node[x].pa = z;
42     if(!isroot(y)) node[z].ch[node[z].ch
43         [1]==y] = x;
44     node[y].ch[d] = node[x].ch[d^1];
45     node[node[x].ch[d^1]].pa = y;
46     node[x].ch[!d] = y;
47     node[y].pa = x;
48     pull(y);
49     pull(x);
50 }
51
52 void splay(int x) {
53     push_all(x);
54     while(!isroot(x)) {

```

```

55         int y = node[x].pa;
56         if(!isroot(y)) {
57             int z = node[y].pa;
58             if((node[z].ch[1]==y) ^ (node[y].ch
59                 [1]==x)) rotate(y);
60             else rotate(x);
61         }
62         rotate(x);
63     }
64 }
65
66 inline int access(int x) {
67     int last = 0;
68     while(x) {
69         splay(x);
70         node[x].ch[1] = last;
71         pull(x);
72         last = x;
73         x = node[x].pa;
74     }
75     return last;
76 }
77
78 inline void make_root(int x) {
79     node[access(x)].rev ^= 1;
80     splay(x);
81 }
82
83 inline void link(int x, int y) {
84     make_root(x);
85     node[x].pa = y;
86 }
87
88 inline void cut(int x, int y) {
89     make_root(x);
90     access(y);
91     splay(y);
92     node[y].ch[0] = 0;
93     node[x].pa = 0;
94 }
95
96 inline void cut_parent(int x) {
97     x = access(x);
98     splay(x);
99     node[node[x].ch[0]].pa = 0;
100     node[x].ch[0] = 0;
101     pull(x);
102 }
103
104 inline int find_root(int x) {
105     x = access(x);
106     while(node[x].ch[0]) x = node[x].ch[0];
107     splay(x);
108     return x;
109 }
110
111 int find_mx(int x) {
112     if(node[x].val == node[x].mx) return x;
113     return node[node[x].ch[0]].mx==node[x].mx
114         ? find_mx(node[x].ch[0]) : find_mx(
115             node[x].ch[1]);
116 }
117
118 inline void change(int x,int b){

```

```

111     splay(x);
112     node[x].data=b;
113     up(x);
114 }
115 inline int query_lca(int u,int v){
116 /*retrun: sum of weight of vertices on the
117    chain (u->v)
118 sum: total weight of the subtree
119 data: weight of the vertex */
120     access(u);
121     int lca=access(v);
122     splay(u);
123     if(u==lca){
124         return node[lca].data+node[node[lca].ch
125             [1]].sum;
126     }else{
127         return node[lca].data+node[node[lca].ch
128             [1]].sum+node[u].sum;
129     }
130 }

```

6.7 Heavy Light Decomposition

```

1  const int MAXN = 10000 + 10;
2
3  vector<PII> e[MAXN];
4  int val[MAXN];
5  int sz[MAXN], max_son[MAXN], p[MAXN], dep[
6      MAXN];
7  int link[MAXN], link_top[MAXN], cnt;
8  void find_max_son(int u) {
9      sz[u] = 1;
10     max_son[u] = -1;
11     for(int i=0; i<SZ(e[u]); i++) {
12         PII tmp = e[u][i];
13         int v = tmp.F;
14         if(v == p[u]) continue;
15
16         p[v] = u;
17         dep[v] = dep[u]+1;
18         val[v] = tmp.S;
19         find_max_son(v);
20         if(max_son[u]<0 || sz[v]>sz[ max_son[u]
21             ]) max_son[u] = v;
22         sz[u] += sz[v];
23     }
24 }
25 void build_link(int u, int top) {
26     link[u] = ++cnt;
27     link_top[u] = top;
28     if(max_son[u] > 0) build_link(max_son[u]
29         , top);
30     for(int i=0; i<SZ(e[u]); i++) {
31         PII tmp = e[u][i];
32         int v = tmp.F;
33         if(v==p[u] || v==max_son[u]) continue;
34         build_link(v, v);
35     }
36 }

```

```

37
38 int query(int a, int b) {
39     int res = -1;
40     int ta = link_top[a], tb = link_top[b];
41     while(ta != tb) {
42         if(dep[ta] < dep[tb]) {
43             swap(a, b);
44             swap(ta, tb);
45         }
46
47         res = max(res, seg->qry(link[ta], link[
48             a], 1, cnt));
49         ta = link_top[a=p[ta]];
50     }
51     if(a != b) {
52         if(dep[a] > dep[b]) swap(a, b);
53         a = max_son[a];
54         res = max(res, seg->qry(link[a], link[b
55             ], 1, cnt));
56     }
57     return res;
58 }

```

6.8 Disjoint Sets + offline skill

```

1  const int MAXN = 300000 + 10;
2
3  bool q[MAXN];
4
5  struct DisJointSet {
6      int p[MAXN], sz[MAXN], gps;
7      vector<pair<int*, int> > h;
8      VI sf;
9
10     void init(int n) {
11         for(int i=1; i<=n; i++) {
12             p[i] = i;
13             sz[i] = 1;
14         }
15         gps = n;
16     }
17
18     void assign(int *k, int v) {
19         h.PB(MP(k, *k));
20         *k = v;
21     }
22
23     void save() {
24         sf.PB(SZ(h));
25     }
26
27     void load() {
28         int last = sf.back(); sf.pop_back();
29         while(SZ(h) != last) {
30             auto x = h.back(); h.pop_back();
31             *x.F = x.S;
32         }
33     }
34
35     int find(int x) {

```

```

36     return x==p[x] ? x : find(p[x]);
37 }
38
39 void uni(int x, int y) {
40     x = find(x), y = find(y);
41     if(x == y) return;
42     if(sz[x] < sz[y]) swap(x, y);
43     assign(&sz[x], sz[x]+sz[y]);
44     assign(&p[y], x);
45     assign(&gps, gps-1);
46 }
47 } djs;
48
49 struct Seg {
50     vector<PII> es;
51     Seg *tl, *tr;
52
53     Seg() {}
54     Seg(int l, int r) {
55         if(l == r) tl = tr = NULL;
56         else {
57             int m = (l+r) / 2;
58             tl = new Seg(l, m);
59             tr = new Seg(m+1, r);
60         }
61     }
62
63     void add(int a, int b, PII e, int l, int
64             r) {
65         if(a <= l && r <= b) es.PB(e);
66         else if(b < l || r < a) return;
67         else {
68             int m = (l+r) / 2;
69             tl->add(a, b, e, l, m);
70             tr->add(a, b, e, m+1, r);
71         }
72     }
73
74     void solve(int l, int r) {
75         djs.save();
76         for(auto p : es) djs.uni(p.F, p.S);
77
78         if(l == r) {
79             if(q[l]) printf("%d\n", djs.gps);
80         }
81         else {
82             int m = (l+r) / 2;
83             tl->solve(l, m);
84             tr->solve(m+1, r);
85         }
86         djs.load();
87     }
88 };
89
90 map<PII, int> prv;
91
92 int main() {
93     freopen("connect.in", "r", stdin);
94     freopen("connect.out", "w", stdout);
95
96     int n, k;
97     scanf("%d%d\n", &n, &k);
98     if(!k) return 0;

```

```

99
100 Seg *seg = new Seg(1, k);
101 djs.init(n);
102 for(int i=1; i<=k; i++) {
103     char op = getchar();
104     if(op == '?') {
105         q[i] = true;
106         op = getchar();
107     }
108     else {
109         int u, v;
110         scanf("%d%d\n", &u, &v);
111         if(u > v) swap(u, v);
112         PII eg = MP(u, v);
113         int p = prv[eg];
114         if(p) {
115             seg->add(p, i, eg, 1, k);
116             prv[eg] = 0;
117         }
118         else prv[eg] = i;
119     }
120 }
121 for(auto p : prv) {
122     if(p.S) {
123         seg->add(p.S, k, p.F, 1, k);
124     }
125 }
126
127 seg->solve(1, k);
128
129 return 0;
130 }

```

6.9 2D Segment Tree

```

1 struct Seg1D {
2     Seg1D *tl, *tr;
3     ll val;
4     // ll tmp;
5     //int _x, _y;
6     Seg1D() :
7         tl(NULL), tr(NULL), val(0), tmp(-1), _x
8         (-1), _y(-1) {}
9     ll query1D(int x1, int x2, int y1, int y2
10        , int l, int r) {
11         /*
12         if no Brian improvement, dont need to
13         pass x1 and x2
14         if(tmp >= 0) {
15             if(x1<=_x&&_x<=x2 && y1<=_y&&_y<=y2)
16                 return tmp;
17             else return 0;
18         }
19         */
20         if(y1 <= l && r <= y2) return val;
21         else if(r < y1 || y2 < l) return 0;
22         else {
23             int m = (l+r)/2;
24             ll a = tl ? tl->query1D(x1, x2, y1,
25                 y2, l, m) : 0;
26             b = tr ? tr->query1D(x1, x2, y1,
27                 y2, m+1, r) : 0;

```



```

22     return gcd(a, b);
23 }
24 }
25 void update1D(int x, int y, ll num, int l
26 , int r) {
27     if(l == r) {
28         val = num;
29         return ;
30     }
31     /*
32     if(tmp < 0 && !t1 && !tr) {
33         tmp = val = num;
34         _x = x;
35         _y = y;
36         return ;
37     }
38     else if(tmp >= 0) {
39         int m = (l+r)/2;
40         if(_y <= m) {
41             if(!t1) t1 = new Seg1D();
42             t1->update1D(_x, _y, tmp, l, m);
43         }
44         else {
45             if(!tr) tr = new Seg1D();
46             tr->update1D(_x, _y, tmp, m+1, r);
47         }
48         tmp = _x = _y = -1;
49     }*/
50     int m = (l+r)/2;
51     if(y <= m) {
52         if(!t1) t1 = new Seg1D();
53         t1->update1D(x, y, num, l, m);
54     }
55     else {
56         if(!tr) tr = new Seg1D();
57         tr->update1D(x, y, num, m+1, r);
58     }
59     ll a = t1 ? t1->val : 0;
60     ll b = tr ? tr->val : 0;
61     val = gcd(a, b);
62 }
63 };
64 struct Seg2D {
65     Seg2D *t1, *tr;
66     Seg1D *t2;
67     Seg2D() :
68         t1(NULL), tr(NULL), t2(NULL) {}
69     ll query2D(int x1, int x2, int y1, int y2
70 , int l, int r) {
71         if(x1 <= l && r <= x2) {
72             if(!t2) t2 = new Seg1D();
73             return t2->query1D(x1, x2, y1, y2, 0,
74 C-1);
75         }
76         else if(x2 < l || r < x1) return 0;
77         else {
78             int m = (l+r)/2;
79             ll a = t1 ? t1->query2D(x1, x2, y1,
80 y2, l, m) : 0,
81 b = tr ? tr->query2D(x1, x2, y1,
82 y2, m+1, r) : 0;
83             return gcd(a, b);
84         }
85     }
86 }
87 };
88 void update2D(int x, int y, ll num, int l
89 , int r) {
90     int m = (l+r)/2;
91     if(l == r) {
92         if(!t2) t2 = new Seg1D();
93         t2->update1D(x, y, num, 0, C-1);
94         return ;
95     }
96     if(x <= m) {
97         if(!t1) t1 = new Seg2D();
98         t1->update2D(x, y, num, l, m);
99     }
100     else {
101         if(!tr) tr = new Seg2D();
102         tr->update2D(x, y, num, m+1, r);
103     }
104 }
105 };

```

7 geometry

7.1 Basic

```

1 const double PI = acos(-1);
2 const double INF = 1e18;
3 const double EPS = 1e-8;
4
5 struct node {
6     double x,y;
7     node(double _x=0, double _y=0) : x(_x),y(
8 _y) {}
9     node operator+(const node& rhs) const
10 { return node(x+rhs.x, y+rhs.y); }
11     node operator-(const node& rhs) const
12 { return node(x-rhs.x, y-rhs.y); }
13     node operator*(const double& rhs) const
14 { return node(x*rhs, y*rhs); }
15     node operator/(const double& rhs) const
16 { return node(x/rhs, y/rhs); }
17     double operator*(const node& rhs) const
18 { return x*rhs.x+y*rhs.y; }
19     double operator^(const node& rhs) const
20 { return x*rhs.y-y*rhs.x; }
21     double len2() const { return x*x+y*y; }
22     double len() const { return sqrt(x*x+y*y)
23 ; }
24     node unit() const { return *this/len(); }
25     node T() const { return node(-y,x); } //
26 counter-clockwise
27     node TR() const { return node(y,-x); } //
28 clockwise
29     node rot(double rad) const { // rotate
30 counter-clockwise in rad

```

```

26     return node(cos(rad)*x-sin(rad)*y, sin(
27         rad)*x+cos(rad)*y);
28 };
29
30 node __mirror(node normal, double constant,
31     node point){ //2D3D
32     double scale=(normal*point+constant)/(
33         normal*normal);
34     return point-normal*(2*scale);
35 }
36
37 node mirror(node p1, node p2, node p3){ //
38     2D3D
39     return __mirror((p2-p1).T(), (p2-p1).T()*
40         p1*(-1), p3);
41 }
42
43 double ori(const node& p1, const node& p2,
44     const node& p3){ //
45     return (p2-p1)^(p3-p1);
46 }
47
48 bool intersect(const node& p1, const node&
49     p2, const node& p3, const node& p4){
50     return (ori(p1,p2,p3)*ori(p1,p2,p4)<0 &&
51         ori(p3,p4,p1)*ori(p3,p4,p2)<0);
52 }
53
54 pair<node,node> two_circle_intersect(node
55     p1, double r1, node p2, double r2){
56     double degree=acos(((p2-p1).len2()+r1*r1-
57         r2*r2)/(2*r1*(p2-p1).len()));
58     return make_pair(p1+(p2-p1).unit().rot(
59         degree)*r1, p1+(p2-p1).unit().rot(-
60         degree)*r1);
61 }
62
63 node intersectionPoint(node p1, node p2,
64     node p3, node p4){
65     double a123 = (p2-p1)^(p3-p1);
66     double a124 = (p2-p1)^(p4-p1);
67     return (p4*a123-p3*a124)/(a123-a124);
68 }
69
70 node inter(const node &p1, const node &p2,
71     const node &p3, const node &p4) //
72     intersection
73 {
74     if(fabs(v1^v2) < EPS)
75         return node(INF, INF);
76     double k = ((p2-p1)^v2) / (v1^v2);
77     return p1 + v1*k;
78 }
79
80 void CircleInter(node o1, double r1, node
81     o2, double r2) {
82     if(r2>r1)
83         swap(r1, r2), swap(o1, o2);
84     double d = (o2-o1).len();
85     node v = (o2-o1).unit();
86     node t = v.TR();
87
88     double area;
89     vector<node> pts;
90     if(d > r1+r2+EPS)
91         area = 0;
92     else if(d < r1-r2)
93         area = r2*r2*PI;
94     else if(r2*r2+d*d > r1*r1){
95         double x = (r1*r1 - r2*r2 + d*d) / (2*d
96             );
97         double th1 = 2*acos(x/r1), th2 = 2*acos
98             ((d-x)/r2);
99         area = (r1*r1*(th1 - sin(th1)) + r2*r2
100             *(th2 - sin(th2))) / 2;
101         double y = sqrt(r1*r1 - x*x);
102         pts.PB(o1 + v*x + t*y), pts.PB(o1 + v*x
103             - t*y);
104     } else {
105         double x = (r1*r1 - r2*r2 - d*d) / (2*d
106             );
107         double th1 = acos((d+x)/r1), th2 = acos
108             (x/r2);
109         area = r1*r1*th1 - r1*d*sin(th1) + r2*
110             r2*(PI-th2);
111         double y = sqrt(r2*r2 - x*x);
112         pts.PB(o2 + v*x + t*y), pts.PB(o2 + v*x
113             - t*y);
114     }
115     //Area: area
116     //Intersections: pts
117 }

```

7.2 Smallest circle problem

```

1 const int N = 1000000 + 10;
2
3 struct PT {
4     double x, y;
5
6     PT() {}
7     PT(double x, double y):
8         x(x), y(y) {}
9     PT operator+(const PT &b) const {
10         return (PT) {x+b.x, y+b.y};
11     }
12     PT operator-(const PT &b) const {
13         return (PT) {x-b.x, y-b.y};
14     }
15     PT operator*(const double b) const {
16         return (PT) {x*b, y*b};
17     }
18     PT operator/(const double b) const {
19         return (PT) {x/b, y/b};
20     }
21     double operator%(const PT &b) const {
22         return x*b.y - y*b.x;
23     }
24
25     double len() const {
26         return sqrt(x*x + y*y);
27     }
28     PT T() const {
29         return (PT) {-y, x};
30     }
31 } p[N];
32
33 void update(PT a, PT b, PT c, PT &o, double
34     &r) {
35     if(c.x < 0.0) o = (a+b) / 2.0;
36     else {

```

```

36 PT p1 = (a+b)/2.0, p2 = p1 + (b-a).T(); 2
37 PT p3 = (a+c)/2.0, p4 = p3 + (c-a).T(); 3 mt19937 rng(seed);
38 double a123 = (p2-p1)%(p3-p1), a124 = ( 4 int randint(int lb,int ub) { // [lb, ub]
    p2-p1)%(p4-p1); 5 return uniform_int_distribution<int>(lb,
39 if(a123 * a124 > 0.0) a123 = -a123; 6 ub)(rng);
40 else a123 = abs(a123), a124 = abs(a124 6 }
    );
41 o = (p4*a123 + p3*a124) / (a123 + a124)
    ;
42 }
43 r = (a-o).len();
44 }
45
46 int main() {
47 srand(7122);
48
49 int m, n;
50 while(scanf("%d%d", &m, &n)) {
51 if(!n && !m) return 0;
52
53 for(int i = 0; i < n; i++) scanf("%lf%
    lf", &p[i].x, &p[i].y);
54
55 for(int i = 0; i < n; i++)
56 swap(p[i], p[rand() % (i+1)]);
57
58 PT a = p[0], b = p[1], c(-1.0, -1.0), o
    = (a+b) / 2.0;
59 double r = (a-o).len();
60 for(int i = 2; i < n; i++) {
61 if((p[i]-o).len() <= r) continue;
62
63 a = p[i];
64 b = p[0];
65 c = (PT) {-1.0, -1.0};
66 update(a, b, c, o, r);
67 for(int j = 1; j < i; j++) {
68 if((p[j]-o).len() <= r) continue;
69
70 b = p[j];
71 c = (PT) {-1.0, -1.0};
72 update(a, b, c, o, r);
73
74 for(int k = 0; k < j; k++) {
75 if((p[k]-o).len() <= r) continue;
76
77 c = p[k];
78 update(a, b, c, o, r);
79 }
80 }
81 }
82
83 printf("%.3f\n", r);
84 }
85 }

```

8.2 Fraction

```

1 struct Frac {
2     ll a,b; // a/b
3     void relax() {
4         ll g=__gcd(a,b);
5         if(g!=0 && g!=1)
6             a/=g, b/=g;
7         if(b<0)
8             a*=-1, b*=-1;
9     }
10    Frac(ll a_=0,ll b_=1): a(a_), b(b_) {
11        relax();
12    }
13    Frac operator + (Frac x) {
14        relax();
15        x.relax();
16        ll g=__gcd(b,x.b);
17        ll lcm=b/g*x.b;
18        return Frac(a*(lcm/b)+x.a*(lcm/x.b),lcm
19            );
20    }
21    Frac operator - (Frac x) {
22        relax();
23        x.relax();
24        Frac t=x;
25        t.a*=-1;
26        return *this+t;
27    }
28    Frac operator * (Frac x) {
29        relax();
30        x.relax();
31        return Frac(a*x.a,b*x.b);
32    }
33    Frac operator / (Frac x) {
34        relax();
35        x.relax();
36        Frac t=Frac(x.b,x.a);
37        return (*this)*t;
38    }
39    bool operator < (Frac x) {
40        ll lcm=b/__gcd(b,x.b)*x.b;
41        return ( (lcm/b)*a < (lcm/x.b)*x.a );
42    }
43 };

```

8 Others

8.1 Random

```
1 const int seed=1;
```