

Contents

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

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 11l*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 MillerRabin other

```

1 //input should < 2^63 - 1 (max prime
    :9223372036854775783)
2 typedef unsigned long long ull;
3
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
15 ull bigmod(ull a, ull d, ull n) {
16     if(d==0) return 1LL;
17     if(d==1) return a % n;
18     return mul(bigmod(mul(a, a, n), d/2, n),
19                 d%2?a:1, n);
20 }
21 const bool PRIME = 1, COMPOSITE = 0;
22 bool miller_rabin(ull n, ull a) {
23     if(__gcd(a, n) == n) return PRIME;
24     if(__gcd(a, n) != 1) return COMPOSITE;
25     ull d = n-1, r = 0, res;
26     while(d%2==0) { ++r; d/=2; }
27     res = bigmod(a, d, n);
28     if(res == 1 || res == n-1) return PRIME;
29     while(r-->0) {
30         res = mul(res, res, n);
31         if(res == n-1) return PRIME;
32     }
33     return COMPOSITE;
34 }
35
36 bool isprime(ull n) {
37     if(n==1)
38         return COMPOSITE;
39     ull as[7] = {2, 325, 9375, 28178, 450775,
40                 9780504, 1795265022};
41     for(int i=0; i<7; i++)
42         if(miller_rabin(n, as[i]) == COMPOSITE)
43             return COMPOSITE;
44     return PRIME;
45 }

```

2.5 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
5                               // type of data
6         for(int j = 0; j < 8; j++)
7             mat[i][j] = mat[i][j] / tmp;
8         for(int j = 0; j < 7; j++) {
9             if(i == j)
10                continue;
11             Frac ratio = mat[j][i]; // Frac ->
12                                   // the type of data
13             for(int k = 0; k < 8; k++)
14                 mat[j][k] = mat[j][k] - ratio * mat[i][k];
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 ,
22                                     maxflow=c
23         eg[ei]=E(b,c);
24         e[a].PB(ei);
25         ei++;
26         eg[ei]=E(a,0);
27         e[b].PB(ei);
28         ei++;
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[v-1]==-1) {
37             int n=qu[ql++];
38             VI &v=e[n];
39             for(int i=v[0]-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].
    cost,d[v].S+1)<d[eg[id].to]) {
47                     d[eg[id].to]=MP(d[v].F+eg[id].
    cost,d[v].S+1);
48                     if(!inq[eg[id].to]) {
49                         que.push(eg[id].to);
50                         inq[eg[id].to]=1;
51                     }
52                 }
53             }
54         }
55         return d[n-1].F<INF;
56     }
57     PIL go(ll cb=cINF) {
58         // cost_bound
59         if(!SPFA()) return MP(0,0);
60         pe.clear();
61         int fl=INF;
62         for(int v=n-1;v!=0;) {
63             for(int id:e[v]) {
64                 int u=eg[id].to;
65                 const E& t=eg[id^1];
66                 if(t.ca>0 && MP(d[u].F+t.cost,d[u].
    S+1)==d[v]) {
67                     fl=min(fl, t.ca);
68                     v=u;
69                     pe.PB(id^1);
70                     break;
71                 }
72             }
73         }
74         if(d[n-1].F>0) fl=min(1ll*fl, cb/d[n-1].F);
75         for(int id:pe) {
76             eg[id].ca-=fl;
77             eg[id^1].ca+=fl;
78         }
79         return MP(fl, 1ll*fl*d[n-1].F);
80     }
81     PIL max_flow() {
82         PIL ans=MP(0,0),temp;
83         while((temp=go()).F>0) {
84             ans.F+=temp.F;
85             ans.S+=temp.S;
86         }
87         return ans;
88     }
89 } 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;

```

```

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
23             -i]);
24         while(s[i+z[i]+1]==s[i-z[i]-1])
25             z[i]++;
26         if(z[i]+i>bst+z[bst])
27             bst=i;
28     }
29     /*for(int i=1;i<len;i++)
30         putchar(s[i]);
31     puts("");
32     for(int i=1;i<len;i++)
33         printf("%d",z[i]);
34     puts("");*/
35     bool yes=0;
36     for(int i=3;i<len;i+=2)
37         if(z[(i+1)/2]==i/2 && z[(i+len
38             )/2]==(len-i-1)/2)
39             yes=1;
40     if(yes)
41         puts("www");
42     else
43         puts("vvvvvv");
44     return 0;

```

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         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             {

```

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

```

1 const int SASIZE=100020; // >= (max length
2     of string + 20)
3 struct SA{
4     char S[SASIZE]; // put target string into
5     S[0:(len-1)]
6     // you can change the type of S into int
7     if required
8     // if the string is in int, please avoid
9     number < 0
10     int R[SASIZE*2],SA[SASIZE];
11     int tR[SASIZE*2],tSA[SASIZE];
12     int cnt[SASIZE],len; // set len
13     before calling build()
14     int H[SASIZE];
15
16 void build_SA() {
17     int maxR=0;
18     for(int i=0;i<len;i++)
19         R[i]=S[i];
20     for(int i=0;i<len;i++)
21         R[len+i]=-1;
22     memset(cnt,0,sizeof(cnt));
23     for(int i=0;i<len;i++)
24         maxR=max(maxR,R[i]);
25     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 inline int c_i(char a) {
12     return (a>='A' && a<='Z') ? a-'A' : a-'a'
13         +26;
14 }
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?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[i]-pp[i-len[t]]));
66         }
67     }
68 }
69
70 int main() {
71     int T;
72     scanf("%d",&T);
73     while(T--) {
74         int n,m;
75         scanf("%d%d",&n,&m);
76         scanf("%s",s+1);
77         for(int i=1;i<=n;i++)
78             scanf("%d",pp+i);
79         for(int i=1;i<=n;i++)
80             pp[i]+=pp[i-1];
81         spt=1;
82         int root=newnode();
83         for(int i=0;i<m;i++) {
84             scanf("%s",a);
85             add(a,root);
86         }
87         make_fl(root);
88         for(int i=1;i<=n;i++)
89             dp[i][1]=dp[i][2]=dp[i][3]=0;
90         match(root);
91         printf("%d\n",dp[n][3]);
92     }
93     return 0;
94 }

```

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++;

```



```

14 }
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]) ));
46     }
47 }

```

4.8 Suffix Automaton(bcw)

```

1 // par : fail link
2 // val : a topological order ( useful for
3 // DP )
4 // go[x] : automata edge ( x is integer in
5 // [0,26) )
6
7 struct SAM{
8     struct State{
9         int par, go[26], val;
10         State () : par(0), val(0){ FZ(go); }
11         State (int _val) : par(0), val(_val){
12             FZ(go); }
13     };
14     vector<State> vec;
15     int root, tail;
16
17 void init(int arr[], int len){
18     vec.resize(2);
19     vec[0] = vec[1] = State(0);
20     root = tail = 1;
21     for (int i=0; i<len; i++)
22         extend(arr[i]);
23 }
24 void extend(int w){
25     int p = tail, np = vec.size();

```

```

26     vec.PB(State(vec[p].val+1));
27     for ( ; p && vec[p].go[w]==0; p=vec[p].
28         par)
29         vec[p].go[w] = np;
30     if (p == 0){
31         vec[np].par = root;
32     } else {
33         if (vec[vec[p].go[w]].val == vec[p].
34             val+1){
35             vec[np].par = vec[p].go[w];
36         } else {
37             int q = vec[p].go[w], r = vec.size
38                 ();
39             vec.PB(vec[q]);
40             vec[r].val = vec[p].val+1;
41             vec[q].par = vec[np].par = r;
42             for ( ; p && vec[p].go[w] == q; p=
43                 vec[p].par)
44                 vec[p].go[w] = r;
45         }
46     }
47     tail = np;
48 }
49 };

```

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;

```



```

32     match[x]=u;
33     return 1;
34 }
35 }
36 return 0;
37 }
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[j]) && __gcd(in[i],in[j])==1)
59                     e[i].pb(j), e[j].pb(i);
60         int ans=0;
61         fill(match,match+N,-1);
62         for(int i=0;i<N;i++)
63             if(match[i]==-1)
64             {
65                 fill(vis,vis+N,0);
66                 if(DFS(i))
67                     ans++;
68             }
69         printf("%d\n",ans);
70     }
71     return 0;
72 }

```

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
   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]]&&DFS(match[N+i]))) {

```

```

14        match[N+i]=x;
15        match[x]=N+i;
16        return 1;
17    }
18 }
19 return 0;
20 }
21
22 int KM() {
23     fill(weight, weight+N*N, 0);
24     for(int i=0;i<N;i++) {
25         for(int j=0;j<N;j++)
26             weight[i]=max(weight[i], a[i][j]);
27     }
28     fill(match, match+N*N, -1);
29     for(int u=0;u<N;u++) {
30         fill(vis, vis+N*N, 0);
31         while(!DFS(u)) {
32             int d=INF;
33             for(int i=0;i<N;i++) {
34                 if(!vis[i]) continue;
35                 for(int j=0;j<N;j++)
36                     if(!vis[N+j])
37                         d=min(d, weight[i]+weight[N+j]-a[i][j]);
38             }
39             for(int i=0;i<N;i++)
40                 if(vis[i])
41                     weight[i]-=d;
42             for(int i=N;i<N*N;i++)
43                 if(vis[i])
44                     weight[i]+=d;
45             fill(vis, vis+N*N, 0);
46         }
47     }
48     int ans=0;
49     for(int i=0;i<N*N;i++) ans+=weight[i];
50     return ans;
51 }

```

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];
8     queue<int> qe;
9     int st,ed;
10    int nb;
11    int bk[MAXN],djs[MAXN];
12    int ans;
13    void init(int _V) {
14        V = _V;
15        FZ(el); FZ(pr);
16        FZ(inq); FZ(inp); FZ(inb);
17        FZ(bk); FZ(djs);
18        ans = 0;
19    }
20    void add_edge(int u, int v) {

```

```

21     el[u][v] = el[v][u] = 1;
22 }
23 int lca(int u,int v) {
24     memset(inp,0,sizeof(inp));
25     while(1) {
26         u = djs[u];
27         inp[u] = true;
28         if(u == st) break;
29         u = bk[pr[u]];
30     }
31     while(1) {
32         v = djs[v];
33         if(inp[v]) return v;
34         v = bk[pr[v]];
35     }
36     return v;
37 }
38 void upd(int u) {
39     int v;
40     while(djs[u] != nb) {
41         v = pr[u];
42         inb[djs[u]] = inb[djs[v]] = true;
43         u = bk[v];
44         if(djs[u] != nb) bk[u] = v;
45     }
46 }
47 void blo(int u,int v) {
48     nb = lca(u,v);
49     memset(inb,0,sizeof(inb));
50     upd(u); upd(v);
51     if(djs[u] != nb) bk[u] = v;
52     if(djs[v] != nb) bk[v] = u;
53     for(int tu = 1; tu <= V; tu++)
54         if(inb[djs[tu]]) {
55             djs[tu] = nb;
56             if(!inq[tu]){
57                 qe.push(tu);
58                 inq[tu] = 1;
59             }
60         }
61 }
62 void flow() {
63     memset(inq,false,sizeof(inq));
64     memset(bk,0,sizeof(bk));
65     for(int i = 1; i <= V;i++)
66         djs[i] = i;
67
68     while(qe.size()) qe.pop();
69     qe.push(st);
70     inq[st] = 1;
71     ed = 0;
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                             );
85                     }
86                 }
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;

```

```

82     } else {
83         ed = v;
84         return;
85     }
86 }
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
6     int n, edge[MXN][MXN];
7     int match[MXN],dis[MXN],onstk[MXN];
8     vector<int> stk;
9
10    void init(int _n) {
11        n = _n;
12        for (int i=0; i<n; i++)
13            for (int j=0; j<n; j++)
14                edge[i][j] = 0;
15    }
16    void add_edge(int u, int v, int w) {
17        edge[u][v] = edge[v][u] = w;
18    }
19    bool SPFA(int u){
20        if (onstk[u]) return true;
21        stk.PB(u);
22        onstk[u] = 1;
23        for (int v=0; v<n; v++){
24            if (u != v && match[u] != v && !onstk
25                [v]){

```

```

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

```

```

        int s[MV][MV/30+1];
        vector<int> sol;

        void init(int v) {
            V = v; ans = 0;
            FZ(el); FZ(dp);
        }

        /* Zero Base */
        void addEdge(int u, int v) {
            if(u > v) swap(u, v);
            if(u == v) return;
            el[u][v/32] |= (1<<(v%32));
        }

        bool dfs(int v, int k) {
            int c = 0, d = 0;
            for(int i=0; i<(V+31)/32; i++) {
                s[k][i] = el[v][i];
                if(k != 1) s[k][i] &= s[k-1][i];
                c += __builtin_popcount(s[k][i]);
            }
            if(c == 0) {
                if(k > ans) {
                    ans = k;
                    sol.clear();
                    sol.push_back(v);
                    return 1;
                }
                return 0;
            }
            for(int i=0; i<(V+31)/32; i++) {
                for(int a = s[k][i]; a ; d++) {
                    if(k + (c-d) <= ans) return
                        0;
                    int lb = a&(-a), lg = 0;
                    a ^= lb;
                    while(lb!=1) {
                        lb = (unsigned int)(lb)
                            >> 1;
                        lg ++;
                    }
                    int u = i*32 + lg;
                    if(k + dp[u] <= ans) return
                        0;
                    if(dfs(u, k+1)) {
                        sol.push_back(v);
                        return 1;
                    }
                }
            }
            return 0;
        }

        int solve() {
            for(int i=V-1; i>=0; i--) {
                dfs(i, 1);
                dp[i] = ans;
            }
            return ans;
        }
    };

```

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]=1;
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]>=1) {
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     }

```

```
69 return 0;
70 }
```

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
20 inline int sz(Treap *t) {
21     return t ? t->sz : 0;
22 }
23
24 inline ll sum(Treap *t) {
25     return t ? t->sum + t->add * sz(t) : 0;
26 }
27
28 inline void add(Treap *t, ll x) {
29     t->add += x;
30 }
```

```

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++) {
84         ll tmp;
85         scanf("%lld", &tmp);
86         t = merge(t, make(tmp));
87     }
88
89     while(q--) {
90         char c;
91         int l, r;
92         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),
11         r(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;
31     print(t->l);
32     putchar(t->val);
33     print(t->r);
34 }
35
36 void takeRef(Treap* t) {
37     if(t) t->refs++;
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
94 void split(Treap* t, int k, Treap* &a,
95           Treap* &b) {
96     if(!t) a = b = NULL;
97     else if(sz(t->l) < k) {
98         a = make(t);
99         a->refs = 0;
100        split(a->r, k-sz(t->l)-1, a->r, b);
101        takeRef(a->l);
102        takeRef(a->r);
103        pull(a);
104    }
105    else {
106        b = make(t);
107        b->refs = 0;
108        split(b->l, k, a, b->l);
109        takeRef(b->l);
110        takeRef(b->r);
111        pull(b);
112    }
113 }
114
115 void print_inorder(Treap* t) {
116     if(!t) return;
117     putchar(t->val);
118     print_inorder(t->l);
119     print_inorder(t->r);
120 }
121
122 char s[N];
123
124 int main() {
125     int m;
126     scanf("%d", &m);
127     scanf("%s", s);
128     int n = strlen(s);
129     int q;
130     scanf("%d", &q);
131
132     Treap* t = NULL;
133     for(int i = 0; i < n; i++) {
134         Treap *a = t, *b = make(s[i]);
135         t = merge(a, b);
136         dropRef(a);
137         dropRef(b);
138     }
139
140     while(q--) {
141         int l, r, x;
142         scanf("%d%d%d", &l, &r, &x);
143         r++;
144
145         Treap *a, *b, *c, *d;
146         a = b = c = d = NULL;
147         split(t, l, a, b);
148         dropRef(a);
149         split(b, r-l, c, d);
150         dropRef(b);
151         dropRef(d);
152         split(t, x, a, b);
153         dropRef(t);
154         Treap* t2 = merge(c, b);
155         dropRef(b);
156         dropRef(c);
157         t = merge(a, t2);
158         dropRef(a);
159         dropRef(t2);
160
161         if(t->sz > m) {
162             Treap* t2 = NULL;
163             split(t, m, t2, a);
164             dropRef(a);
165             dropRef(t);
166             t = t2;
167         }
168     }

```



```

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 l;
42
43     int m = (l+r)/2;
44
45     int a = ta->tl->val;
46     int b = tb->tl->val;
47     if(b-a >= k) return query(ta->tl, tb->tl,
48         k, l, m);
49     else return query(ta->tr, tb->tr, k,
50         m+1, r);
51 }

```

```

49 struct Query {
50     int op, l, r, k, c, v;
51
52     bool operator<(const Query b) const {
53         return c < b.c;
54     }
55 } qs[Q];
56 int arr[N];
57 Seg *t[N];
58 vector<int> vec2;
59
60 int main() {
61     int T;
62     scanf("%d", &T);
63
64     while(T--) {
65         int n, q;
66         scanf("%d%d", &n, &q);
67
68         for(int i = 1; i <= n; i++) {
69             scanf("%d", arr+i);
70             vec2.push_back(arr[i]);
71         }
72         for(int i = 0; i < q; i++) {
73             scanf("%d", &qs[i].op);
74             if(qs[i].op == 1) scanf("%d%d%d", &qs[i].l, &qs[i].r, &qs[i].k);
75             else scanf("%d%d", &qs[i].c, &qs[i].v);
76
77             if(qs[i].op == 2) vec2.push_back(qs[i].v);
78         }
79         sort(vec2.begin(), vec2.end());
80         vec2.resize(unique(vec2.begin(), vec2.end())-vec2.begin());
81         for(int i = 1; i <= n; i++) arr[i] = lower_bound(vec2.begin(), vec2.end(), arr[i]) - vec2.begin();
82         int mn = 0, mx = vec2.size()-1;
83
84         for(int i = 0; i <= n; i++) t[i] = NULL;
85
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     vec2.clear();
105     Seg::pmem = Seg::mem;
106 }
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,
5          rsum, mx_sum;
6      Treap *l, *r;
7
8      Treap() {}
9      Treap(int _val) :
10         pri(rand()), sz(1), val(_val), chg(
11             INF), rev(0), sum(_val), lsum(
12             _val), rsum(_val), mx_sum(_val)
13         , l(NULL), r(NULL) {}
14 };
15
16 int sz(Treap* t) {return t ? t->sz : 0;}
17 int sum(Treap* t) {
18     if(!t) return 0;
19     if(t->chg == INF) return t->sum;
20     else return t->chg*t->sz;
21 }
22 int lsum(Treap* t) {
23     if(!t) return -INF;
24     if(t->chg != INF) return max(t->chg,
25         (t->chg)*(t->sz));
26     if(t->rev) return t->rsum;
27     return t->lsum;
28 }
29 int rsum(Treap* t) {
30     if(!t) return -INF;
31     if(t->chg != INF) return max(t->chg,
32         (t->chg)*(t->sz));
33     if(t->rev) return t->lsum;
34     return t->rsum;
35 }
36 int mx_sum(Treap* t) {
37     if(!t) return -INF;
38     if(t->chg != INF) return max(t->chg,
39         (t->chg)*(t->sz));
40     return t->mx_sum;
41 }
42 void push(Treap* t) {
43     if(t->chg != INF) {
44         t->val = t->chg;
45         t->sum = (t->sz) * (t->chg);
46         t->lsum = t->rsum = t->mx_sum = max
47             (t->sum, t->val);
48         if(t->l) t->l->chg = t->chg;
49         if(t->r) t->r->chg = t->chg;
50         t->chg = INF;
51     }
52
53     if(t->rev) {
54         swap(t->l, t->r);
55         if(t->l) t->l->rev ^= 1;
56         if(t->r) t->r->rev ^= 1;
57         t->rev = 0;
58     }
59 }
60
61 void pull(Treap* t) {
62     t->sz = sz(t->l)+sz(t->r)+1;
63     t->sum = sum(t->l)+sum(t->r)+t->val;
64     t->lsum = max(lsum(t->l), sum(t->l)+max
65         (0, lsum(t->r))+t->val);
66     t->rsum = max(rsum(t->r), sum(t->r)+max
67         (0, rsum(t->l))+t->val);
68     t->mx_sum = max(max(mx_sum(t->l),
69         mx_sum(t->r)), max(0, rsum(t->l))+
70         max(0, lsum(t->r))+t->val);
71 }
72
73 Treap* merge(Treap* a, Treap* b) {
74     if(!a || !b) return a ? a : b;
75     if(a->pri > b->pri) {
76         push(a);
77         a->r = merge(a->r, b);
78         pull(a);
79         return a;
80     }
81     else {
82         push(b);
83         b->l = merge(a, b->l);
84         pull(b);
85         return b;
86     }
87 }
88
89 void split(Treap* t, int k, Treap* &a,
90     Treap* &b) {
91     if(!t) {
92         a = b = NULL;
93         return ;
94     }
95     push(t);
96     if(sz(t->l) < k) {
97         a = t;
98         push(a);
99         split(t->r, k-sz(t->l)-1, a->r, b);
100         pull(a);
101     }
102     else {
103         b = t;
104         push(b);
105         split(t->l, k, a, b->l);
106         pull(b);
107     }
108 }
109
110 void del(Treap* t) {
111     if(!t) return;
112     del(t->l);
113     del(t->r);
114     delete t;
115 }

```

```

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(t1, 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, t1, t);
166
167             split(t, k, t, tr);
168             printf("%d\n", sum(t));
169             t = merge(t1, merge(t, tr));
170         }
171         if(!strcmp(s, "MAX-SUM")) {
172             printf("%d\n", mx_sum(t));
173         }
174     }
175     return 0;
176 }

```

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 }

```

6.6 Link Cut Tree

```

46
47 int main() {
48     int T,n,x,o,size;
49     bool bst,bqu,bpq;
50     scanf("%d",&T);
51     while(T--) {
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--) {
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)
86             count++;
87         if(bqu)
88             count++;
89         if(bpq)
90             count++;
91         if(count>1)
92             puts("not sure");
93         else if(count==0)
94             puts("impossible");
95         else if(bst)
96             puts("stack");
97         else if(bqu)
98             puts("queue");
99         else if(bpq)
100             puts("priority queue");
101     }
102     return 0;
103 }

```

```

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     }

```

```

56     rotate(x);
57 }
58 }
59
60 inline int access(int x) {
61     int last = 0;
62     while(x) {
63         splay(x);
64         node[x].ch[1] = last;
65         pull(x);
66         last = x;
67         x = node[x].pa;
68     }
69     return last;
70 }
71
72 inline void make_root(int x) {
73     node[access(x)].rev ^= 1;
74     splay(x);
75 }
76
77 inline void link(int x, int y) {
78     make_root(x);
79     node[x].pa = y;
80 }
81
82 inline void cut(int x, int y) {
83     make_root(x);
84     access(y);
85     splay(y);
86     node[y].ch[0] = 0;
87     node[x].pa = 0;
88 }
89
90 inline void cut_parent(int x) {
91     x = access(x);
92     splay(x);
93     node[node[x].ch[0]].pa = 0;
94     node[x].ch[0] = 0;
95     pull(x);
96 }
97
98 inline int find_root(int x) {
99     x = access(x);
100    while(node[x].ch[0]) x = node[x].ch[0];
101    splay(x);
102    return x;
103 }
104
105 int find_mx(int x) {
106     if(node[x].val == node[x].mx) return x;
107     return node[node[x].ch[0]].mx == node[x].mx
108         ? find_mx(node[x].ch[0]) : find_mx(
109             node[x].ch[1]);
110 }
111
112 inline void change(int x, int b){
113     splay(x);
114     node[x].data=b;
115     up(x);
116 }
117
118 inline int query_lca(int u,int v){
119     /*retrun: sum of weight of vertices on the
120        chain (u->v)
121
122 sum: total weight of the subtree
123 data: weight of the vertex */
124     access(u);
125     int lca=access(v);
126     splay(u);
127     if(u==lca){
128         return node[lca].data+node[node[lca].ch
129             [1]].sum;
130     }else{
131         return node[lca].data+node[node[lca].ch
132             [1]].sum+node[u].sum;
133     }
134 }
135 }

```

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
9  void find_max_son(int u) {
10     sz[u] = 1;
11     max_son[u] = -1;
12     for(int i=0; i<SZ(e[u]); i++) {
13         PII tmp = e[u][i];
14         int v = tmp.F;
15         if(v == p[u]) continue;
16
17         p[v] = u;
18         dep[v] = dep[u]+1;
19         val[v] = tmp.S;
20         find_max_son(v);
21         if(max_son[u]<0 || sz[v]>sz[ max_son[u]
22             ]) max_son[u] = v;
23         sz[u] += sz[v];
24     }
25 }
26
27 void build_link(int u, int top) {
28     link[u] = ++cnt;
29     link_top[u] = top;
30     if(max_son[u] > 0) build_link(max_son[u]
31         ], top);
32     for(int i=0; i<SZ(e[u]); i++) {
33         PII tmp = e[u][i];
34         int v = tmp.F;
35         if(v==p[u] || v==max_son[u]) continue;
36
37         build_link(v, v);
38     }
39 }
40
41 int query(int a, int b) {
42     int res = -1;
43     int ta = link_top[a], tb = link_top[b];
44     while(ta != tb) {
45         if(dep[ta] < dep[tb]) {
46             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 <= 1 && r <= y2) return val;
21         else if(r < y1 || y2 < 1) 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;
28             return gcd(a, b);
29         }
30     }
31     void update1D(int x, int y, ll num, int l
32         , int r) {
33         if(l == r) {
34             val = num;
35             return ;
36         }
37         /*
38         if(tmp < 0 && !tl && !tr) {
39             tmp = val = num;
40             _x = x;
41             _y = y;
42             return ;
43         }
44         else if(tmp >= 0) {
45             int m = (l+r)/2;
46             if(_y <= m) {
47                 if(!tl) tl = new Seg1D();
48                 tl->update1D(_x, _y, tmp, l, m);
49             }
50             else {
51                 if(!tr) tr = new Seg1D();
52                 tr->update1D(_x, _y, tmp, m+1, r);
53             }
54             tmp = _x = _y = -1;
55         }
56         */
57         int m = (l+r)/2;
58         if(y <= m) {
59             if(!tl) tl = new Seg1D();
60             tl->update1D(x, y, num, l, m);
61         }
62         else {
63             if(!tr) tr = new Seg1D();
64             tr->update1D(x, y, num, m+1, r);
65         }
66         ll a = tl ? tl->val : 0;
67         ll b = tr ? tr->val : 0;
68         val = gcd(a, b);
69     }
70 };
71
72 struct Seg2D {
73     Seg2D *tl, *tr;
74     Seg1D *t2;
75     Seg2D() :
76         tl(NULL), tr(NULL), t2(NULL) {}
77     ll query2D(int x1, int x2, int y1, int y2
78        , int l, int r) {
79         if(x1 <= 1 && r <= x2) {
80             if(!t2) t2 = new Seg1D();
81             return t2->query1D(x1, x2, y1, y2, 0,
82                 C-1);
83         }
84         else if(x2 < 1 || r < x1) return 0;
85         else {
86             int m = (l+r)/2;
87             ll a = tl ? tl->query2D(x1, x2, y1,
88                 y2, l, m) : 0,
89                 b = tr ? tr->query2D(x1, x2, y1,
90                 y2, m+1, r) : 0;
91             return gcd(a, b);
92         }
93     }
94     void update2D(int x, int y, ll num, int l
95         , int r) {
96         int m = (l+r)/2;
97         if(l == r) {
98             if(!t2) t2 = new Seg1D();
99             t2->update1D(x, y, num, 0, C-1);
100             return ;
101         }

```



```

87     }
88     if(x <= m) {
89         if(!t1) t1 = new Seg2D();
90         t1->update2D(x, y, num, l, m);
91     }
92     else {
93         if(!tr) tr = new Seg2D();
94         tr->update2D(x, y, num, m+1, r);
95     }
96     if(!t1) t1 = new Seg2D();
97     if(!tr) tr = new Seg2D();
98     ll a = t1->t2 ? t1->t2->query1D(l, m, y
99         , y, 0, C-1) : 0,
100         b = tr->t2 ? tr->t2->query1D(m+1, r,
101             y, y, 0, C-1) : 0;
102     if(!t2) t2 = new Seg1D();
103     t2->update1D(x, y, gcd(a, b), 0, C-1);
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
31          return node(cos(rad)*x-sin(rad)*y, sin(
32              rad)*x+cos(rad)*y);
33      }
34  };
35
36  node __mirror(node normal, double constant,
37      node point){ //2D3D

```

```

38     double scale=(normal*point+constant)/((
39         normal*normal);
40     return point-normal*(2*scale);
41 }
42 node mirror(node p1, node p2, node p3){ //
43     2D3D
44     return __mirror((p2-p1).T(), (p2-p1).T()*
45         p1*(-1), p3);
46 }
47 double ori(const node& p1, const node& p2,
48     const node& p3){ // (p2-p1)^(p3-p1)
49     return (p2-p1)^(p3-p1);
50 }
51 bool intersect(const node& p1, const node&
52     p2, const node& p3, const node& p4){
53     return (ori(p1,p2,p3)*ori(p1,p2,p4)<0 &&
54         ori(p3,p4,p1)*ori(p3,p4,p2)<0);
55 }
56 pair<node,node> two_circle_intersect(node
57     p1, double r1, node p2, double r2){
58     double degree=acos(((p2-p1).len2()+r1*r1-
59         r2*r2)/(2*r1*(p2-p1).len()));
60     return make_pair(p1+(p2-p1).unit().rot(
61         degree)*r1, p1+(p2-p1).unit().rot(-
62         degree)*r1);
63 }
64 node intersectionPoint(node p1, node p2,
65     node p3, node p4){
66     double a123 = (p2-p1)^(p3-p1);
67     double a124 = (p2-p1)^(p4-p1);
68     return (p4*a123-p3*a124)/(a123-a124);
69 }
70 node inter(const node &p1, const node &v1,
71     const node &p2, const node &v2) //
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 void CircleInter(node o1, double r1, node
80     o2, double r2) {
81     if(r2>r1)
82         swap(r1, r2), swap(o1, o2);
83     double d = (o2-o1).len();
84     node v = (o2-o1).unit();
85     node t = v.TR();
86
87     double area;
88     vector<node> pts;
89     if(d > r1+r2+EPS)
90         area = 0;
91     else if(d < r1-r2)
92         area = r2*r2*PI;
93     else if(r2*r2+d*d > r1*r1){
94         double x = (r1*r1 - r2*r2 + d*d) / (2*d
95             );
96         double th1 = 2*acos(x/r1), th2 = 2*acos
97             ((d-x)/r2);
98         area = (r1*r1*(th1 - sin(th1)) + r2*r2
99             *(th2 - sin(th2))) / 2;
100         double y = sqrt(r1*r1 - x*x);

```

```

77 pts.PB(o1 + v*x + t*y), pts.PB(o1 + v*x 41 o = (p4*a123 + p3*a124) / (a123 + a124)
    - t*y);
78 } else { 42 }
79 double x = (r1*r1 - r2*r2 - d*d) / (2*d 43 r = (a-o).len();
    ); 44 }
80 double th1 = acos((d+x)/r1), th2 = acos 45
    (x/r2); 46 int main() {
81 area = r1*r1*th1 - r1*d*sin(th1) + r2* 47 srand(7122);
    r2*(PI-th2); 48
82 double y = sqrt(r2*r2 - x*x); 49 int m, n;
83 pts.PB(o2 + v*x + t*y), pts.PB(o2 + v*x 50 while(scanf("%d%d", &m, &n)) {
    - t*y); 51 if(!n && !m) return 0;
84 } 52
85 //Area: area 53 for(int i = 0; i < n; i++) scanf("%lf%
86 //Intersections: pts 54 lf", &p[i].x, &p[i].y);
87 } 55
56 for(int i = 0; i < n; i++)
57 swap(p[i], p[rand() % (i+1)]);
58
59 PT a = p[0], b = p[1], c(-1.0, -1.0), o
    = (a+b) / 2.0;
60 double r = (a-o).len();
61 for(int i = 2; i < n; i++) {
62 if((p[i]-o).len() <= r) continue;
63
64 a = p[i];
65 b = p[0];
66 c = (PT) {-1.0, -1.0};
67 update(a, b, c, o, r);
68 for(int j = 1; j < i; j++) {
69 if((p[j]-o).len() <= r) continue;
70
71 b = p[j];
72 c = (PT) {-1.0, -1.0};
73 update(a, b, c, o, r);
74
75 for(int k = 0; k < j; k++) {
76 if((p[k]-o).len() <= r) continue;
77
78 c = p[k];
79 update(a, b, c, o, r);
80 }
81 }
82
83 printf("%.3f\n", r);
84 }
85 }

```

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
    &r) {
34     if(c.x < 0.0) o = (a+b) / 2.0;
35     else {
36         PT p1 = (a+b)/2.0, p2 = p1 + (b-a).T();
37         PT p3 = (a+c)/2.0, p4 = p3 + (c-a).T();
38         double a123 = (p2-p1)%(p3-p1), a124 = (
            p2-p1)%(p4-p1);
39         if(a123 * a124 > 0.0) a123 = -a123;
40         else a123 = abs(a123), a124 = abs(a124
            );

```

8 Others

8.1 Random

```

1 const int seed=1;
2
3 mt19937 rng(seed);
4 int randint(int lb,int ub) { // [lb, ub]
5     return uniform_int_distribution<int>(lb,
6         ub)(rng);

```

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 };

```