

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	
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	
5 graph	
5.1 Bipartite matching($O(N^3)$)	
5.2 KM($O(N^4)$)	
5.3 Max clique(bcw)	
5.4 EdgeBCC	
5.5 VerticeBCC	
5.6 Dominating Tree	
5.7 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: nu sc ai si ts=4
   sm sts=4 sw=4
4
5 map <F9> <ESC>:w<CR>:!g++ % -o %< -O2 -Wall
   -Wno-unused-result -std=c++0x<CR>
6 map <S-F9> <ESC>:w<CR>:!g++ % -o %< -O2 -
   Wall -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 }
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<n;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 }

```

2.3 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);

```

2.4 MillerRabin other

```

1 /* Miller Rabin code from ioicamp */
2 #include <bits/stdc++.h>
3 #define PB push_back
4 #define MP make_pair
5 #define F first
6 #define S second
7 #define SZ(x) ((int)(x).size())
8 #define ALL(x) (x).begin(),(x).end()
9 #ifdef _DEBUG_
10     #define debug(...) printf(__VA_ARGS__)
11 #else
12     #define debug(...) 0
13 #endif

```

```

14 using namespace std;
15 typedef long long ll;
16 typedef pair<int,int> PII;
17 typedef vector<int> VI;
18
19 ll mul(ll a, ll b, ll n) {
20     ll r = 0;
21     a %= n, b %= n;
22     while(b) {
23         if(b&1) r = (a+r>=n ? a+r-n : a+r);
24         a = (a+a>=n ? a+a-n : a+a);
25         b >>= 1;
26     }
27     return r;
28 }
29
30 ll bigmod(ll a, ll d, ll n) {
31     if(d==0) return 1LL;
32     if(d==1) return a % n;
33     return mul(bigmod(mul(a, a, n), d/2, n),
34                 d%2?a:1, n);
35 }
36
37 const bool PRIME = 1, COMPOSITE = 0;
38 bool miller_rabin(ll n, ll a) {
39     if(__gcd(a, n) == n) return PRIME;
40     if(__gcd(a, n) != 1) return COMPOSITE;
41     ll d = n-1, r = 0, res;
42     while(d%2==0) { ++r; d/=2; }
43     res = bigmod(a, d, n);
44     if(res == 1 || res == n-1) return PRIME;
45     while(r-->0) {
46         res = mul(res, res, n);
47         if(res == n-1) return PRIME;
48     }
49     return COMPOSITE;
50 }
51
52 bool isprime(ll n) {
53     if(n==1) return COMPOSITE;
54     ll as[7] = {2, 325, 9375, 28178, 450775,
55                 9780504, 1795265022};
56     for(int i=0; i<7; i++)
57         if(miller_rabin(n, as[i]) == COMPOSITE)
58             return COMPOSITE;
59     return PRIME;
60 }

```

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) continue;
10            Frac ratio = mat[j][i]; // Frac ->
11                                   // the type of data

```

```

11         for(int k = 0; k < 8; k++)
12             mat[j][k] = mat[j][k] - ratio * mat
13             [i][k];
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 ,
22                                     // maxflow=c
23         eg[ei]=E(b,c);
24         e[a].PB(eg[ei]);
25         ei++;
26         eg[ei]=E(a,0);
27         e[b].PB(eg[ei]);
28         ei++;
29     }
30     int d[MAXV],qu[MAXV],ql,qr;
31     bool BFS() {
32         memset(d,-1,v*sizeof(int));
33         ql=qr=0;
34         qu[qr++]=0;
35         d[0]=0;
36         while(ql<qr && d[v-1]==-1) {
37             int n=qu[ql++];
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[qr++]=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]
57         ].co==0)
58         continue;
59     if((temp=go(eg[u[i]].to,min(p,eg[u[i]
60         ].co)))==0)
61         continue;
62     eg[u[i]].co-=temp;
63     eg[u[i]^1].co+=temp;
64     ptr[n]=i;
65     return temp;
66 }
67 ptr[n]=SZ(u);
68 return 0;
69 }
70 int max_flow() {
71     int ans=0,temp;
72     while(BFS()) {
73         for(int i=0;i<v;i++)
74             ptr[i]=0;
75         while((temp=go(0,INF))>0)
76             ans+=temp;
77     }
78 }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 VI KMP_match(const char *S,const int *F,
12     const char *T) {
13     VI ans;
14     int p=-1;
15     for(int i=0;T[i];i++) {
16         while(p!=-1 && S[p+1]!=T[i])
17             p=F[p];
18         if(S[p+1]==T[i])
19             p++;
20         if(!S[p+1]) {
21             ans.PB(i-p);
22             p=F[p];
23         }
24     }
25     return ans;
26 }

```

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 #include <bits/stdc++.h>
3 #define pb push_back
4 #define F first
5 #define S second
6 #define SZ(x) ((int)(x).size())
7 #define MP make_pair
8 using namespace std;
9 typedef long long ll;
10 typedef pair<int,int> PII;
11 typedef vector<int> VI;
12
13 char in[100100];
14 char s[200100];
15 int z[200100];
16
17 int main()
18 {
19     while(gets(in))
20     {
21         int len=1;
22         for(int i=0;in[i];i++)
23         {
24             s[len++]='*';
25             s[len++]=in[i];
26         }
27         s[len]=0;
28         z[0]=0;
29         z[1]=0;
30         int bst=1;
31         for(int i=1;i<len;i++)
32         {
33             z[i]=min(bst+z[bst]-i,z[bst+bst-i]);
34             while(s[i+z[i]+1]==s[i-z[i]-1])
35                 z[i]++;
36             if(z[i]+i>bst+z[bst])
37                 bst=i;
38         }
39         /*for(int i=1;i<len;i++)
40             putchar(s[i]);
41         puts("");
42         for(int i=1;i<len;i++)
43             printf("%d",z[i]);
44         puts("");*/
45         bool yes=0;
46         for(int i=3;i<len;i+=2)

```

```

47         if(z[(i+1)/2]==i/2 && z[(i+len)/2]==(len-i-1)/2)
48             yes=1;
49         if(yes)
50             puts("www");
51         else
52             puts("vvvvvvv");
53     }
54     return 0;
55 }

```

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[tR[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]-1]+i<tR[SA[j]+i])
50             num++;
51         R[SA[j]]=num;
52         maxR=max(maxR,R[SA[j]]);
53     }
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 #include <cstdio>
3 #include <cstring>
4 #include <new>
5
6 struct Trie {
7     int c;
8     bool fi=0;
9     Trie *fail,*ch[52];
10     Trie():c(0){memset(ch,0,sizeof(ch));}
11 }trie[1000100];
12
13 char m[1010],f[100100];
14 Trie *str[1010],*na,*root;
15
16 inline int c_i(char a) {
17     return (a>='A' && a<='Z') ? a-'A' : a-'a'+26;
18 }
19
20 void insert(char *s,int num) {
21     Trie *at=root;
22     while(*s) {
23         if(!at->ch[c_i(*s)])
24             at->ch[c_i(*s)]=new (na++) Trie();
25         at=at->ch[c_i(*s)],s++;
26     }
27     str[num]=at;
28 }
29
30 Trie *q[1000100];
31 int ql,qr;
32
33 void init() {
34     ql=qr=-1;
35     q[++qr]=root;

```

```

36 root->fail=NULL;
37 while(q1<qr) {
38     Trie *n=q[++q1],*f;
39     for(int i=0;i<52;i++) {
40         if(!n->ch[i])
41             continue;
42         f=n->fail;
43         while(f && !f->ch[i])
44             f=f->fail;
45         n->ch[i]->fail=f?f->ch[i]:root;
46         q[++qr]=n->ch[i];
47     }
48 }
49 }
50
51 void go(char *s) {
52     Trie*p=root;
53     while(*s) {
54         while(p && !p->ch[c_i(*s)])
55             p=p->fail;
56         p=p?p->ch[c_i(*s)]:root;
57         p->fi=1;
58         s++;
59     }
60 }
61
62 void AC() {
63     for(int i=qr;i>0;i--)
64         q[i]->fail->c+=q[i]->c;
65 }
66
67 int main() {
68     int T,q;
69     scanf("%d",&T);
70     while(T--) {
71         na=trie;
72         root=new (na++) Trie();
73         scanf("%s",f);
74         scanf("%d",&q);
75         for(int i=0;i<q;i++) {
76             scanf("%s",m);
77             insert(m,i);
78         }
79         init();
80         go(f);
81         for(int i=0;i<q;i++)
82             puts(str[i]->fi?"y":"n");
83     }
84     return 0;
85 }

```

4.6 Aho-Corasick-2016ioicamp

```

1 // AC code of 2016ioicamp 54
2 #include <bits/stdc++.h>
3 #define PB push_back
4 #define MP make_pair
5 #define F first
6 #define S second
7 #define SZ(x) ((int)(x).size())
8 #define ALL(x) (x).begin(),(x).end()
9 #ifdef _DEBUG_
10 #define debug(...) printf(__VA_ARGS__)

```

```

11 #else
12 #define debug(...) (void)0
13 #endif
14 using namespace std;
15 typedef long long ll;
16 typedef pair<int,int> PII;
17 typedef vector<int> VI;
18
19 const int MAXNM=100010;
20 int pp[MAXNM];
21
22 const int sizz=100010;
23 int nx[sizz][26],spt;
24 int fl[sizz],efl[sizz],ed[sizz];
25 int len[sizz];
26 int newnode(int len_=0) {
27     for(int i=0;i<26;i++)nx[spt][i]=0;
28     ed[spt]=0;
29     len[spt]=len_;
30     return spt++;
31 }
32 int add(char *s,int p) {
33     int l=1;
34     for(int i=0;s[i];i++) {
35         int a=s[i]-'a';
36         if(nx[p][a]==0) nx[p][a]=newnode(l);
37         p=nx[p][a];
38         l++;
39     }
40     ed[p]=1;
41     return p;
42 }
43 int q[sizz],qs,qe;
44 void make_fl(int root) {
45     fl[root]=efl[root]=0;
46     qs=qe=0;
47     q[qe++]=root;
48     for(;qs!=qe;) {
49         int p=q[qs++];
50         for(int i=0;i<26;i++) {
51             int t=nx[p][i];
52             if(t==0) continue;
53             int tmp=fl[p];
54             for(;tmp&&nx[tmp][i]==0;) tmp=fl[tmp];
55             fl[t]=tmp?nx[tmp][i]:root;
56             efl[t]=ed[fl[t]]?fl[t]:efl[fl[t]];
57             q[qe++]=t;
58         }
59     }
60 }
61 char s[MAXNM];
62 char a[MAXNM];
63
64 int dp[MAXNM][4];
65
66 void mmax(int &a,int b) {
67     a=max(a,b);
68 }
69
70 void match(int root) {
71     int p=root;
72     for(int i=1;s[i];i++) {
73         int a=s[i]-'a';
74         for(;p&&nx[p][a]==0;p=fl[p]);

```

```

75 p=p?nx[p][a]:root;
76 for(int j=1;j<=3;j++)
77 dp[i][j]=dp[i-1][j];
78 for(int t=p;t;t=efl[t]) {
79     if(!ed[t])
80         continue;
81     for(int j=1;j<=3;j++)
82         mmax(dp[i][j],dp[i-len[t]][j-1]+(pp
            [i]-pp[i-len[t]]));
83 }
84 }
85 }
86
87 int main() {
88     int T;
89     scanf("%d",&T);
90     while(T--){
91         int n,m;
92         scanf("%d%d",&n,&m);
93         scanf("%s",s+1);
94         for(int i=1;i<=n;i++)
95             scanf("%d",pp+i);
96         for(int i=1;i<=n;i++)
97             pp[i]+=pp[i-1];
98         spt=1;
99         int root=newnode();
100        for(int i=0;i<m;i++) {
101            scanf("%s",a);
102            add(a,root);
103        }
104        make_fl(root);
105        for(int i=1;i<=n;i++)
106            dp[i][1]=dp[i][2]=dp[i][3]=0;
107        match(root);
108        printf("%d\n",dp[n][3]);
109    }
110    return 0;
111 }

```

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],
    cnt[MAXN];
9 int edp[MAXN];
10 int nid=1;
11 int new_node(int len_) {
12     len[nid]=len_;
13     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
        [cur]+2);
28     int tmp=fail[cur];
29     while(tmp && s[i-len[tmp]-1]!=s[i])
        tmp=fail[tmp];
30     if(tmp==0) fail[nt]=even_root;
31     else {
32         assert(ch[tmp][s[i]-'a']);
33         fail[nt]=ch[tmp][s[i]-'a'];
34     }
35     edp[nt]=i;
36 }
37 cur=ch[cur][s[i]-'a'];
38 cnt[cur]++;
39 }
40 for(int i=nid-1;i>even_root;i--) {
41     cnt[fail[i]]+=cnt[i];
42     pal.PB( MP( MP(edp[i]-len[i]+1, len[i])
        , cnt[i]) ));
43 }
44 }

```

5 graph

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

```

1 // NTUJ1263
2 #include <bits/stdc++.h>
3 #define pb push_back
4 #define F first
5 #define S second
6 #define SZ(x) ((int)(x).size())
7 #define MP make_pair
8 using namespace std;
9 typedef long long ll;
10 typedef pair<int,int> PII;
11 typedef vector<int> VI;
12
13 bool is(ll x)
14 {
15     ll l=1,r=2000000,m;
16     while(l<=r)
17     {
18         m=(l+r)/2;
19         if(m*m==x)
20             return 1;
21         if(m*m<x)
22             l=m+1;
23         else
24             r=m-1;
25     }
26     return 0;
27 }
28
29 VI odd,even;
30 int in[300];
31 VI e[300];

```



```

32 int match[300];
33 bool vis[300];
34
35 bool DFS(int x)
36 {
37     vis[x]=1;
38     for(int u:e[x])
39     {
40         if(match[u]==-1 || (!vis[match[u]]&&DFS
41             (match[u])))
42         {
43             match[u]=x;
44             match[x]=u;
45             return 1;
46         }
47     }
48     return 0;
49 }
50 int main()
51 {
52     int N;
53     while(scanf("%d",&N)==1)
54     {
55         odd.clear();
56         even.clear();
57         for(int i=0;i<N;i++)
58             e[i].clear();
59         for(int i=0;i<N;i++)
60         {
61             scanf("%d",in+i);
62             if(in[i]%2==0)
63                 even.pb(i);
64             else
65                 odd.pb(i);
66         }
67         for(int i:even)
68             for(int j:odd)
69                 if(is(1ll*in[i]*in[i]+1ll*in[j]*in[
70                     j]) && __gcd(in[i],in[j])==1)
71                     e[i].pb(j), e[j].pb(i);
72         int ans=0;
73         fill(match,match+N,-1);
74         for(int i=0;i<N;i++)
75             if(match[i]==-1)
76             {
77                 fill(vis,vis+N,0);
78                 if(DFS(i))
79                     ans++;
80             }
81         printf("%d\n",ans);
82     }
83     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
  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 i=0;i<N;i++) {
31         fill(vis, vis+N*N, 0);
32         while(!DFS(i)) {
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     }
50     int ans=0;
51     for(int i=0;i<N*N;i++) ans+=weight[i];
52     return ans;
53 }

```

5.3 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)
47                     >> 1;
48                 lg ++;
49             }
50             int u = i*32 + lg;
51             if(k + dp[u] <= ans) return 0;
52             if(dfs(u, k+1)) {
53                 sol.push_back(v);
54                 return 1;
55             }
56         }
57     }
58     return 0;
59 }
60
61 int solve() {
62     for(int i=V-1; i>=0; i--) {
63         dfs(i, 1);
64         dp[i] = ans;
65     }
66     return ans;
67 };

```

5.4 EdgeBCC

```

1 const int MAXN=1010;
2 const int MAXM=5010;
3 VI e[MAXN];
4 int low[MAXN],lv1[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]=lv1[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 }

```

5.5 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;

```

5.6 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 }

```

5.7 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 #include <cstdlib>
2 #include <cstdio>
3 #include <algorithm>
4
5 using namespace std;
6
7 typedef long long ll;
8
9 const int N = 100000 + 10;
10
11 struct Treap {
12     static Treap mem[N], *pmem;
13
14     int sz, pri;
15     ll val, sum, add;
16     Treap *l, *r;
17
18     Treap() {}
19     Treap(ll _val):
20         l(NULL), r(NULL), sz(1), pri(rand()),
21         val(_val), sum(_val), add(0) {}
22 } Treap::mem[N], *Treap::pmem = Treap::mem;
23
24 Treap* make(ll val) {
25     return new (Treap::pmem++) Treap(val);
26 }
27
28 inline int sz(Treap *t) {
29     return t ? t->sz : 0;
30 }
31
32 inline ll sum(Treap *t) {
33     return t ? t->sum + t->add * sz(t) : 0;
34 }
35
36 inline void add(Treap *t, ll x) {

```

```

36     t->add += x;
37 }
38
39 void push(Treap *t) {
40     t->val += t->add;
41     if(t->l) t->l->add += t->add;
42     if(t->r) t->r->add += t->add;
43     t->add = 0;
44 }
45
46 void pull(Treap *t) {
47     t->sum = sum(t->l) + sum(t->r) + t->val;
48     t->sz = sz(t->l) + sz(t->r) + 1;
49 }
50
51 Treap* merge(Treap *a, Treap *b) {
52     if(!a || !b) return a ? a : b;
53     else if(a->pri > b->pri) {
54         push(a);
55         a->r = merge(a->r, b);
56         pull(a);
57         return a;
58     }
59     else {
60         push(b);
61         b->l = merge(a, b->l);
62         pull(b);
63         return b;
64     }
65 }
66
67 void split(Treap* t, int k, Treap *&a,
68     Treap *&b) {
69     if(!t) a = b = NULL;
70     else if(sz(t->l) < k) {
71         a = t;
72         push(a);
73         split(t->r, k - sz(t->l) - 1, a->r, b);
74         pull(a);
75     }
76     else {
77         b = t;
78         push(b);
79         split(t->l, k, a, b->l);
80         pull(b);
81     }
82 }
83
84 int main() {
85     srand(105105);
86
87     int n, q;
88     scanf("%d%d", &n, &q);
89
90     Treap *t = NULL;
91     for(int i = 0; i < n; i++) {
92         ll tmp;
93         scanf("%LLd", &tmp);
94         t = merge(t, make(tmp));
95     }
96
97     while(q--) {
98         char c;
99         int l, r;
100         scanf("\n%c %d %d", &c, &l, &r);

```

```

100 Treap *tl = NULL, *tr = NULL;
101 if(c == 'Q') {
102     split(t, l - 1, tl, t);
103     split(t, r - l + 1, t, tr);
104     printf("%lld\n", sum(t));
105     t = merge(tl, merge(t, tr));
106 }
107 else {
108     ll x;
109     scanf("%lld", &x);
110     split(t, l - 1, tl, t);
111     split(t, r - l + 1, t, tr);
112     add(t, x);
113     t = merge(tl, merge(t, tr));
114 }
115 }
116 }
117 return 0;
118 }
119 }

```

6.2 copy on write treap

```

1 #include <cstdlib>
2 #include <cstdio>
3 #include <algorithm>
4 #include <climits>
5 #include <cstring>
6
7 using namespace std;
8
9 const int N = 1000000 + 10;
10
11 struct Treap {
12     char val;
13     int sz, refs;
14     Treap *l, *r;
15
16     Treap() {}
17     Treap(char _val):
18         val(_val), sz(1), refs(0), l(NULL),
19         r(NULL) {}
20 };
21 Treap* make(Treap* t) {
22     return new Treap(*t);
23 }
24
25 Treap* make(char _val) {
26     return new Treap(_val);
27 }
28
29 void print_ref(Treap* t) {
30     if(!t) return;
31     print_ref(t->l);
32     printf("%d ", t->refs);
33     print_ref(t->r);
34 }
35
36 void print(Treap* t) {
37     if(!t) return;
38     print(t->l);
39     putchar(t->val);
40     print(t->r);
41 }
42
43 void takeRef(Treap* t) {
44     if(t) t->refs++;
45 }
46
47 void dropRef(Treap* t) {
48     if(t) {
49         char c = t->val;
50         t->refs--;
51         if(t->refs <= 0) {
52             dropRef(t->l);
53             dropRef(t->r);
54             delete t;
55         }
56     }
57 }
58
59 int sz(Treap* t) {
60     return t ? t->sz : 0;
61 }
62
63 int rnd(int m) {
64     static int x = 851025;
65     return (x = (x*0xdefaced+1) & INT_MAX)
66         % m;
67 }
68
69 void pull(Treap* t) {
70     t->sz = sz(t->l) + sz(t->r) + 1;
71 }
72
73 Treap* merge(Treap* a, Treap* b) {
74     if(!a || !b) {
75         Treap* t = a ? make(a) : make(b);
76         t->refs = 0;
77         takeRef(t->l);
78         takeRef(t->r);
79         return t;
80     }
81
82     Treap* t;
83     if( rnd(a->sz+b->sz) < a->sz ) {
84         t = make(a);
85         t->refs = 0;
86         t->r = merge(a->r, b);
87         takeRef(t->l);
88         takeRef(t->r);
89     }
90     else {
91         t = make(b);
92         t->refs = 0;
93         t->l = merge(a, b->l);
94         takeRef(t->l);
95         takeRef(t->r);
96     }
97     pull(t);
98     return t;
99 }
100
101 void split(Treap* t, int k, Treap* &a,
102     Treap* &b) {
103     if(!t) a = b = NULL;

```

```

103     else if(sz(t->l) < k) {
104         a = make(t);
105         a->refs = 0;
106         split(a->r, k-sz(t->l)-1, a->r, b);
107         takeRef(a->l);
108         takeRef(a->r);
109         pull(a);
110     }
111     else {
112         b = make(t);
113         b->refs = 0;
114         split(b->l, k, a, b->l);
115         takeRef(b->l);
116         takeRef(b->r);
117         pull(b);
118     }
119 }
120
121 void print_inorder(Treap* t) {
122     if(!t) return;
123     putchar(t->val);
124     print_inorder(t->l);
125     print_inorder(t->r);
126 }
127
128 char s[N];
129
130 int main() {
131     int m;
132     scanf("%d", &m);
133     scanf("%s", s);
134     int n = strlen(s);
135     int q;
136     scanf("%d", &q);
137
138     Treap* t = NULL;
139     for(int i = 0; i < n; i++) {
140         Treap *a = t, *b = make(s[i]);
141         t = merge(a, b);
142         dropRef(a);
143         dropRef(b);
144     }
145
146     while(q--) {
147         int l, r, x;
148         scanf("%d%d%d", &l, &r, &x);
149         r++;
150
151         Treap *a, *b, *c, *d;
152         a = b = c = d = NULL;
153         split(t, l, a, b);
154         dropRef(a);
155         split(b, r-l, c, d);
156         dropRef(b);
157         dropRef(d);
158         split(t, x, a, b);
159         dropRef(t);
160         Treap* t2 = merge(c, b);
161         dropRef(b);
162         dropRef(c);
163         t = merge(a, t2);
164         dropRef(a);
165         dropRef(t2);
166
167         if(t->sz > m) {
168             Treap* t2 = NULL;
169             split(t, m, t2, a);
170             dropRef(a);
171             dropRef(t);
172             t = t2;
173         }
174     }
175
176     print(t);
177     putchar('\n');
178
179     return 0;
180 }

```

6.3 copy on write segment tree

```

1 #include <cstdlib>
2 #include <cstdio>
3 #include <algorithm>
4 #include <vector>
5
6 using namespace std;
7
8 const int N = 50000 + 10;
9 const int Q = 10000 + 10;
10
11 struct Seg {
12     static Seg mem[N*80], *pmem;
13
14     int val;
15     Seg *tl, *tr;
16
17     Seg() :
18         tl(NULL), tr(NULL), val(0) {}
19
20     Seg* init(int l, int r) {
21         Seg* t = new (pmem++) Seg();
22         if(l != r) {
23             int m = (l+r)/2;
24             t->tl = init(l, m);
25             t->tr = init(m+1, r);
26         }
27         return t;
28     }
29
30     Seg* add(int k, int l, int r) {
31         Seg* _t = new (pmem++) Seg(*this);
32         if(l==r) {
33             _t->val++;
34             return _t;
35         }
36
37         int m = (l+r)/2;
38         if(k <= m) _t->tl = tl->add(k, l, m);
39         else _t->tr = tr->add(k, m+1, r);
40
41         _t->val = _t->tl->val + _t->tr->val;
42         return _t;
43     }
44 } Seg::mem[N*80], *Seg::pmem = mem;
45
46 int query(Seg* ta, Seg* tb, int k, int l,
47           int r) {

```

```

47 if(l == r) return l;
48
49 int m = (l+r)/2;
50
51 int a = ta->t1->val;
52 int b = tb->t1->val;
53 if(b-a >= k) return query(ta->t1, tb->t1
54 , k, l, m);
55 else return query(ta->tr, tb->tr, k
56 -(b-a), m+1, r);
57 };
58
59 struct Query {
60 int op, l, r, k, c, v;
61
62 bool operator<(const Query b) const {
63 return c < b.c;
64 }
65 } qs[Q];
66 int arr[N];
67 Seg *t[N];
68 vector<int> vec2;
69
70 int main() {
71 int T;
72 scanf("%d", &T);
73
74 while(T--) {
75 int n, q;
76 scanf("%d%d", &n, &q);
77
78 for(int i = 1; i <= n; i++) {
79 scanf("%d", arr+i);
80 vec2.push_back(arr[i]);
81 }
82 for(int i = 0; i < q; i++) {
83 scanf("%d", &qs[i].op);
84 if(qs[i].op == 1) scanf("%d%d%d", &qs
85 [i].l, &qs[i].r, &qs[i].k);
86 else scanf("%d%d", &qs[i].c, &qs[i].
87 v);
88
89 if(qs[i].op == 2) vec2.push_back(qs[i]
90 l.v);
91 }
92 sort(vec2.begin(), vec2.end());
93 vec2.resize(unique(vec2.begin(), vec2.
94 end())-vec2.begin());
95 for(int i = 1; i <= n; i++) arr[i] =
96 lower_bound(vec2.begin(), vec2.end()
97 , arr[i]) - vec2.begin();
98 int mn = 0, mx = vec2.size()-1;
99
100 for(int i = 0; i <= n; i++) t[i] = NULL
101 ;
102 t[0] = new (Seg::pmem++) Seg();
103 t[0] = t[0]->init(mn, mx);
104 int ptr = 0;
105 for(int i = 1; i <= n; i++) {
106 t[i] = t[i-1]->add(arr[i], mn, mx);
107 }
108
109 for(int i = 0; i < q; i++) {
110 int op = qs[i].op;
111 if(op == 1) {
112 int l = qs[i].l, r = qs[i].r, k =
113 qs[i].k;
114 printf("%d\n", vec2[query(t[l-1], t
115 [r], k, mn, mx)]);
116 }
117 if(op == 2) {
118 continue;
119 }
120 if(op == 3) puts("7122");
121 }
122
123 vec2.clear();
124 Seg::pmem = Seg::mem;
125 }
126
127 return 0;
128 }

```

6.4 Treap+(H0J 92)

```

1 #include <cstdlib>
2 #include <cstdio>
3 #include <algorithm>
4 #include <cstring>
5
6 using namespace std;
7
8 const int INF = 103456789;
9
10 struct Treap {
11 int pri, sz, val, chg, rev, sum, lsum,
12 rsum, mx_sum;
13 Treap *l, *r;
14
15 Treap() {}
16 Treap(int _val) :
17 pri(rand()), sz(1), val(_val), chg(
18 INF), rev(0), sum(_val), lsum(
19 _val), rsum(_val), mx_sum(_val),
20 l(NULL), r(NULL) {}
21
22 int sz(Treap* t) {return t ? t->sz : 0;}
23 int sum(Treap* t) {
24 if(!t) return 0;
25 if(t->chg == INF) return t->sum;
26 else return t->chg*t->sz;
27 }
28 int lsum(Treap* t) {
29 if(!t) return -INF;
30 if(t->chg != INF) return max(t->chg,
31 (t->chg)*(t->sz));
32 if(t->rev) return t->rsum;
33 return t->lsum;
34 }
35 int rsum(Treap* t) {
36 if(!t) return -INF;
37 if(t->chg != INF) return max(t->chg,
38 (t->chg)*(t->sz));
39 if(t->rev) return t->lsum;
40 return t->rsum;
41 }
42 int mx_sum(Treap* t) {

```

```

38     if(!t) return -INF;
39     if(t->chg != INF) return max(t->chg,
    (t->chg)*(t->sz));
40     return t->mx_sum;
41 }
42
43 void push(Treap* t) {
44     if(t->chg != INF) {
45         t->val = t->chg;
46         t->sum = (t->sz) * (t->chg);
47         t->lsum = t->rsum = t->mx_sum = max
    (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     if(t->rev) {
53         swap(t->l, t->r);
54         if(t->l) t->l->rev ^= 1;
55         if(t->r) t->r->rev ^= 1;
56         t->rev = 0;
57     }
58 }
59
60 void pull(Treap* t) {
61     t->sz = sz(t->l)+sz(t->r)+1;
62     t->sum = sum(t->l)+sum(t->r)+t->val;
63     t->lsum = max(lsum(t->l), sum(t->l)+max
    (0, lsum(t->r))+t->val);
64     t->rsum = max(rsum(t->r), sum(t->r)+max
    (0, rsum(t->l))+t->val);
65     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);
66 }
67
68 Treap* merge(Treap* a, Treap* b) {
69     if(!a || !b) return a ? a : b;
70     if(a->pri > b->pri) {
71         push(a);
72         a->r = merge(a->r, b);
73         pull(a);
74         return a;
75     }
76     else {
77         push(b);
78         b->l = merge(a, b->l);
79         pull(b);
80         return b;
81     }
82 }
83
84 void split(Treap* t, int k, Treap* &a,
    Treap* &b) {
85     if(!t) {
86         a = b = NULL;
87         return ;
88     }
89     push(t);
90     if(sz(t->l) < k) {
91         a = t;
92         push(a);
93         split(t->r, k-sz(t->l)-1, a->r, b);
94         pull(a);
95     }
    else {
96         b = t;
97         push(b);
98         split(t->l, k, a, b->l);
99         pull(b);
100    }
101 }
102
103 void del(Treap* t) {
104     if(!t) return;
105     del(t->l);
106     del(t->r);
107     delete t;
108 }
109
110 int main() {
111     srand(7122);
112
113     int n, m;
114     scanf("%d%d", &n, &m);
115
116     Treap* t = NULL;
117     for(int i = 0; i < n; i++) {
118         int x;
119         scanf("%d", &x);
120         t = merge(t, new Treap(x));
121     }
122
123     while(m--) {
124         char s[15];
125         scanf("%s", s);
126
127         Treap *t1 = NULL, *tr = NULL, *t2 =
            NULL;
128
129         if(!strcmp(s, "INSERT")) {
130             int p, k;
131             scanf("%d%d", &p, &k);
132             for(int i = 0; i < k; i++) {
133                 int x;
134                 scanf("%d", &x);
135                 t2 = merge(t2, new Treap(x)
                    );
136             }
137             split(t, p, t1, tr);
138             t = merge(t1, merge(t2, tr));
139         }
140
141         if(!strcmp(s, "DELETE")) {
142             int p, k;
143             scanf("%d%d", &p, &k);
144             split(t, p-1, t1, t);
145             split(t, k, t, tr);
146             del(t);
147             t = merge(t1, tr);
148         }
149
150         if(!strcmp(s, "MAKE-SAME")) {
151             int p, k, l;
152             scanf("%d%d%d", &p, &k, &l);
153             split(t, p-1, t1, t);
154             split(t, k, t, tr);
155             if(t) t->chg = l;
156             t = merge(t1, merge(t, tr));
157         }
158     }

```



```

159     if(!strcmp(s, "REVERSE")) {
160         int p, k;
161         scanf("%d%d", &p, &k);
162         split(t, p-1, tl, t);
163         split(t, k, t, tr);
164         if(t) t->rev ^= 1;
165         t = merge(tl, merge(t, tr));
166     }
167
168     if(!strcmp(s, "GET-SUM")) {
169         int p, k;
170         scanf("%d%d", &p, &k);
171         split(t, p-1, tl, t);
172         split(t, k, t, tr);
173         printf("%d\n", sum(t));
174         t = merge(tl, merge(t, tr));
175     }
176
177     if(!strcmp(s, "MAX-SUM")) {
178         printf("%d\n", mx_sum(t));
179     }
180 }
181
182 return 0;
183 }
184

```

6.5 Leftist Tree

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct Left {
5     Left *l,*r;
6     int v,h;
7     Left(int v_) : v(v_), h(1), l(0), r(0) {}
8 };
9
10 int height(Left *p) { return p ? p -> h : 0 ; }
11
12 Left* combine(Left *a,Left *b) {
13     if(!a || !b) return a ? a : b ;
14     Left *p ;
15     if( a->v > b->v ) {
16         p = a;
17         p -> r = combine( p -> r , b );
18     }
19     else {
20         p = b;
21         p -> r = combine( p -> r , a );
22     }
23     if( height( p->l ) < height( p->r ) )
24         swap( p->l , p->r );
25     p->h = min( height( p->l ) , height( p->r ) ) + 1;
26     return p;
27 }
28 Left *root;
29
30 void push(int v) {
31     Left *p = new Left(v);
32     root = combine( root , p );
33 }
34 int top() { return root? root->v : -1; }
35 void pop() {
36     if(!root) return;
37     Left *a = root->l , *b = root->r ;
38     delete root;
39     root = combine( a , b );
40 }
41 void clear(Left* &p) {
42     if(!p)
43         return;
44     if(p->l) clear(p->l);
45     if(p->r) clear(p->r);
46     delete p;
47     p = 0 ;
48 }
49
50 int main() {
51     int T,n,x,o,size;
52     bool bst,bqu,bpq;
53     scanf("%d",&T);
54     while(T-->0) {
55         bst=bqu=bpq=1;
56         stack<int> st;
57         queue<int> qu;
58         clear(root);
59         size=0;
60         scanf("%d",&n);
61         while(n-->0) {
62             scanf("%d%d",&o,&x);
63             if(o==1)
64                 st.push(x),qu.push(x),push(x),size++;
65             else if(o==2) {
66                 size--;
67                 if(size<0)
68                     bst=bqu=bpq=0;
69                 if(bst) {
70                     if(st.top()!=x)
71                         bst=0;
72                     st.pop();
73                 }
74                 if(bqu) {
75                     if(qu.front()!=x)
76                         bqu=0;
77                     qu.pop();
78                 }
79                 if(bpq) {
80                     // printf("(%d)\n",top());
81                     if(top()!=x)
82                         bpq=0;
83                     pop();
84                 }
85             }
86         }
87         int count=0;
88         if(bst) count++;
89         if(bqu) count++;
90         if(bpq) count++;
91         if(count>1)
92             puts("not sure");
93     }
94 }

```

```

97     else if(count==0)
98         puts("impossible");
99     else if(bst)
100         puts("stack");
101     else if(bqu)
102         puts("queue");
103     else if(bpq)
104         puts("priority queue");
105 }
106 return 0;
107 }

```

6.6 Link Cut Tree

```

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 const int MAXN = 100000 + 10;
19
20 struct SplayTree {
21     int val, mx, ch[2], pa;
22     bool rev;
23     void init() {
24         val = mx = -1;
25         rev = false;
26         pa = ch[0] = ch[1] = 0;
27     }
28 } node[MAXN*2];
29
30 inline bool isroot(int x) {
31     return node[node[x].pa].ch[0]!=x && node[
        node[x].pa].ch[1]!=x;
32 }
33
34 inline void pull(int x) {
35     node[x].mx = max(node[x].val, max(node[
        node[x].ch[0]].mx, node[node[x].ch
        [1]].mx));
36 }
37
38 inline void push(int x) {
39     if(node[x].rev) {
40         node[node[x].ch[0]].rev ^= 1;
41         node[node[x].ch[1]].rev ^= 1;
42         swap(node[x].ch[0], node[x].ch[1]);
43         node[x].rev ^= 1;
44     }
45 }
46

```

```

47 void push_all(int x) {
48     if(!isroot(x)) push_all(node[x].pa);
49     push(x);
50 }
51
52 inline void rotate(int x) {
53     int y = node[x].pa, z = node[y].pa, d =
        node[y].ch[1]==x;
54     node[x].pa = z;
55     if(!isroot(y)) node[z].ch[node[z].ch
        [1]==y] = x;
56     node[y].ch[d] = node[x].ch[d^1];
57     node[node[x].ch[d^1]].pa = y;
58     node[x].ch[!d] = y;
59     node[y].pa = x;
60     pull(y);
61     pull(x);
62 }
63
64 void splay(int x) {
65     push_all(x);
66     while(!isroot(x)) {
67         int y = node[x].pa;
68         if(!isroot(y)) {
69             int z = node[y].pa;
70             if((node[z].ch[1]==y) ^ (node[y].ch
                [1]==x)) rotate(y);
71             else rotate(x);
72         }
73         rotate(x);
74     }
75 }
76
77 inline int access(int x) {
78     int last = 0;
79     while(x) {
80         splay(x);
81         node[x].ch[1] = last;
82         pull(x);
83         last = x;
84         x = node[x].pa;
85     }
86     return last;
87 }
88
89 inline void make_root(int x) {
90     node[access(x)].rev ^= 1;
91     splay(x);
92 }
93
94 inline void link(int x, int y) {
95     make_root(x);
96     node[x].pa = y;
97 }
98
99 inline void cut(int x, int y) {
100     make_root(x);
101     access(y);
102     splay(y);
103     node[y].ch[0] = 0;
104     node[x].pa = 0;
105 }
106
107 inline void cut_parent(int x) {
108     x = access(x);

```

```

22 int sz[MAXN], max_son[MAXN], p[MAXN], dep[
    MAXN];
23 int link[MAXN], link_top[MAXN], cnt;
24
25 void find_max_son(int u) {
26     sz[u] = 1;
27     max_son[u] = -1;
28     for(int i=0; i<SZ(e[u]); i++) {
29         PII tmp = e[u][i];
30         int v = tmp.F;
31         if(v == p[u]) continue;
32
33         p[v] = u;
34         dep[v] = dep[u]+1;
35         val[v] = tmp.S;
36         find_max_son(v);
37         if(max_son[u]<0 || sz[v]>sz[ max_son[u]
            ]) max_son[u] = v;
38         sz[u] += sz[v];
39     }
40 }
41
42 void build_link(int u, int top) {
43     link[u] = ++cnt;
44     link_top[u] = top;
45     if(max_son[u] > 0) build_link(max_son[u]
        ], top);
46     for(int i=0; i<SZ(e[u]); i++) {
47         PII tmp = e[u][i];
48         int v = tmp.F;
49         if(v==p[u] || v==max_son[u]) continue;
50
51         build_link(v, v);
52     }
53 }
54
55 int query(int a, int b) {
56     int res = -1;
57     int ta = link_top[a], tb = link_top[b];
58     while(ta != tb) {
59         if(dep[ta] < dep[tb]) {
60             swap(a, b);
61             swap(ta, tb);
62         }
63
64         res = max(res, seg->qry(link[ta], link[
            a], 1, cnt));
65         ta = link_top[a=p[ta]];
66     }
67
68     if(a != b) {
69         if(dep[a] > dep[b]) swap(a, b);
70         a = max_son[a];
71         res = max(res, seg->qry(link[a], link[b
            ], 1, cnt));
72     }
73
74     return res;
75 }

```

6.8 Disjoint Sets + offline skill

```
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 const int MAXN = 300000 + 10;
19
20 bool q[MAXN];
21
22 struct DisJointSet {
23     int p[MAXN], sz[MAXN], gps;
24     vector<pair<int*, int> > h;
25     VI sf;
26
27     void init(int n) {
28         for(int i=1; i<=n; i++) {
29             p[i] = i;
30             sz[i] = 1;
31         }
32         gps = n;
33     }
34
35     void assign(int *k, int v) {
36         h.PB(MP(k, *k));
37         *k = v;
38     }
39
40     void save() {
41         sf.PB(SZ(h));
42     }
43
44     void load() {
45         int last = sf.back(); sf.pop_back();
46         while(SZ(h) != last) {
47             auto x = h.back(); h.pop_back();
48             *x.F = x.S;
49         }
50     }
51
52     int find(int x) {
53         return x==p[x] ? x : find(p[x]);
54     }
55
56     void uni(int x, int y) {
57         x = find(x), y = find(y);
58         if(x == y) return;
59         if(sz[x] < sz[y]) swap(x, y);
60         assign(&sz[x], sz[x]+sz[y]);
61         assign(&p[y], x);
62         assign(&gps, gps-1);
63     }
64 } djs;
65
66 struct Seg {
67     vector<PII> es;
68     Seg *tl, *tr;
69
70     Seg() {}
71     Seg(int l, int r) {
72         if(l == r) tl = tr = NULL;
73         else {
74             int m = (l+r) / 2;
75             tl = new Seg(l, m);
76             tr = new Seg(m+1, r);
77         }
78     }
79
80     void add(int a, int b, PII e, int l, int r) {
81         if(a <= l && r <= b) es.PB(e);
82         else if(b < l || r < a) return;
83         else {
84             int m = (l+r) / 2;
85             tl->add(a, b, e, l, m);
86             tr->add(a, b, e, m+1, r);
87         }
88     }
89
90     void solve(int l, int r) {
91         djs.save();
92         for(auto p : es) djs.uni(p.F, p.S);
93
94         if(l == r) {
95             if(q[l]) printf("%d\n", djs.gps);
96         }
97         else {
98             int m = (l+r) / 2;
99             tl->solve(l, m);
100             tr->solve(m+1, r);
101         }
102
103         djs.load();
104     }
105 };
106
107 map<PII, int> prv;
108
109 int main() {
110     freopen("connect.in", "r", stdin);
111     freopen("connect.out", "w", stdout);
112
113     int n, k;
114     scanf("%d%d\n", &n, &k);
115     if(!k) return 0;
116
117     Seg *seg = new Seg(1, k);
118     djs.init(n);
119     for(int i=1; i<=k; i++) {
120         char op = getchar();
121         if(op == '?') {
122             q[i] = true;
123             op = getchar();
124         }
125         else {
126             int u, v;
127             scanf("%d%d\n", &u, &v);
128             if(u > v) swap(u, v);
129             PII eg = MP(u, v);
130             int p = prv[eg];

```

```

131     if(p) {
132         seg->add(p, i, eg, 1, k);
133         prv[eg] = 0;
134     }
135     else prv[eg] = i;
136 }
137 }
138 for(auto p : prv) {
139     if(p.S) {
140         seg->add(p.S, k, p.F, 1, k);
141     }
142 }
143
144 seg->solve(1, k);
145
146 return 0;
147 }

```

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

```

```

97     if(!tr) tr = new Seg2D();
98     ll a = t1->t2 ? t1->t2->query1D(1, m, y
        , y, 0, C-1) : 0,
99     b = tr->t2 ? tr->t2->query1D(m+1, r,
        y, y, 0, C-1) : 0;
100    if(!t2) t2 = new Seg1D();
101    t2->update1D(x, y, gcd(a, b), 0, C-1);
102 }
103 };

```

7 geometry

7.1 Basic

```

1 // correct code of NPSC2013 senior-final pF
2
3 #include <bits/stdc++.h>
4 #define PB push_back
5 #define F first
6 #define S second
7 #define SZ(x) ((int)(x).size())
8 #define MP make_pair
9 using namespace std;
10 typedef long long ll;
11 typedef pair<int,int> PII;
12 typedef vector<int> VI;
13
14 typedef double db;
15 typedef pair<db, db> PDD;
16
17 PDD operator+(const PDD &a, const PDD &b) {
18     return MP(a.F+b.F, a.S+b.S);
19 }
20 PDD operator-(const PDD &a, const PDD &b) {
21     return MP(a.F-b.F, a.S-b.S);
22 }
23 PDD operator*(const PDD &a, const db &b) {
24     return MP(a.F*b, a.S*b);
25 }
26 PDD operator/(const PDD &a, const db &b) {
27     return MP(a.F/b, a.S/b);
28 }
29 db dot(const PDD &a, const PDD &b) {
30     return a.F*b.F + a.S*b.S;
31 }
32 db cross(const PDD &a, const PDD &b) {
33     return a.F*b.S - a.S*b.F;
34 }
35 db abs2(const PDD &a) {
36     return dot(a, a);
37 }
38 db abs(const PDD &a) {
39     return sqrt( abs2(a) );
40 }
41
42 const db PI = acos(-1);
43 const db INF = 1e18;
44 const db EPS = 1e-8;
45
46 PDD inter(const PDD &p1, const PDD &v1,
    const PDD &p2, const PDD &v2) //
    intersection

```

```

47 {
48     if(fabs(cross(v1, v2)) < EPS)
49         return MP(INF, INF);
50     db k = cross((p2-p1), v2) / cross(v1, v2)
        ;
51     return p1 + v1*k;
52 }
53 void CircleInter(PDD o1, db r1, PDD o2, db
    r2) {
54     if(r2>r1)
55         swap(r1, r2), swap(o1, o2);
56     db d = abs(o2-o1);
57     PDD v = o2-o1;
58     v = v / abs(v);
59     PDD t = MP(v.S, -v.F);
60
61     db area;
62     vector<PDD> pts;
63     if(d > r1+r2+EPS)
64         area = 0;
65     else if(d < r1-r2)
66         area = r2*r2*PI;
67     else if(r2*r2+d*d > r1*r1){
68         db x = (r1*r1 - r2*r2 + d*d) / (2*d);
69         db th1 = 2*acos(x/r1), th2 = 2*acos((d-
            x)/r2);
70         area = (r1*r1*(th1 - sin(th1)) + r2*r2
            *(th2 - sin(th2))) / 2;
71         db y = sqrt(r1*r1 - x*x);
72         pts.PB(o1 + v*x + t*y), pts.PB(o1 + v*x
            - t*y);
73     } else {
74         db x = (r1*r1 - r2*r2 - d*d) / (2*d);
75         db th1 = acos((d+x)/r1), th2 = acos(x/
            r2);
76         area = r1*r1*th1 - r1*d*sin(th1) + r2*
            r2*(PI-th2);
77         db y = sqrt(r2*r2 - x*x);
78         pts.PB(o2 + v*x + t*y), pts.PB(o2 + v*x
            - t*y);
79     }
80     //Area: area
81     //Intersections: pts
82 }
83
84 int main() {
85     return 0;
86 }

```

7.2 Smallest circle problem

```

1 #include <cstdlib>
2 #include <cstdio>
3 #include <algorithm>
4 #include <cmath>
5
6 using namespace std;
7
8 const int N = 1000000 + 10;
9
10 struct PT {
11     double x, y;
12 }

```



```

13 PT() {}
14 PT(double x, double y):
15     x(x), y(y) {}
16 PT operator+(const PT &b) const {
17     return PT {x+b.x, y+b.y};
18 }
19 PT operator-(const PT &b) const {
20     return PT {x-b.x, y-b.y};
21 }
22 PT operator*(const double b) const {
23     return PT {x*b, y*b};
24 }
25 PT operator/(const double b) const {
26     return PT {x/b, y/b};
27 }
28 double operator%(const PT &b) const {
29     return x*b.y - y*b.x;
30 }
31
32 double len() const {
33     return sqrt(x*x + y*y);
34 }
35 PT T() const {
36     return PT {-y, x};
37 }
38 } p[N];
39
40 void update(PT a, PT b, PT c, PT &o, double
    &r) {
41     if(c.x < 0.0) o = (a+b) / 2.0;
42     else {
43         PT p1 = (a+b)/2.0, p2 = p1 + (b-a).T();
44         PT p3 = (a+c)/2.0, p4 = p3 + (c-a).T();
45         double a123 = (p2-p1)%(p3-p1), a124 = (
            p2-p1)%(p4-p1);
46         if(a123 * a124 > 0.0) a123 = -a123;
47         else a123 = abs(a123), a124 = abs(a124
            );
48         o = (p4*a123 + p3*a124) / (a123 + a124)
            ;
49     }
50     r = (a-o).len();
51 }
52
53 int main() {
54     srand(7122);
55
56     int m, n;
57     while(scanf("%d%d", &m, &n)) {
58         if(!n && !m) return 0;
59
60         for(int i = 0; i < n; i++) scanf("%Lf
            Lf", &p[i].x, &p[i].y);
61
62         for(int i = 0; i < n; i++)
63             swap(p[i], p[rand() % (i+1)]);
64
65         PT a = p[0], b = p[1], c(-1.0, -1.0), o
            = (a+b) / 2.0;
66         double r = (a-o).len();
67         for(int i = 2; i < n; i++) {
68             if((p[i]-o).len() <= r) continue;
69
70             a = p[i];
71             b = p[0];

```

```

72         c = (PT) {-1.0, -1.0};
73         update(a, b, c, o, r);
74         for(int j = 1; j < i; j++) {
75             if((p[j]-o).len() <= r) continue;
76
77             b = p[j];
78             c = (PT) {-1.0, -1.0};
79             update(a, b, c, o, r);
80
81             for(int k = 0; k < j; k++) {
82                 if((p[k]-o).len() <= r) continue;
83
84                 c = p[k];
85                 update(a, b, c, o, r);
86             }
87         }
88     }
89
90     printf("%.3f\n", r);
91 }
92 }

```

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,
        ub)(rng);
6 }

```

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     Frac operator - (Frac x) {
21         relax();
22         x.relax();
23         Frac t=x;
24         t.a*=-1;

```



```
25     return *this+t;
26 }
27 Frac operator * (Frac x) {
28     relax();
29     x.relax();
30     return Frac(a*x.a,b*x.b);
31 }
32 Frac operator / (Frac x) {
33     relax();
34     x.relax();
35     Frac t=Frac(x.b,x.a);
36     return (*this)*t;
37 }
38 };
```