Contents

1 Basic

1.1 default code

```cpp
#include <bits/stdc++.h>
#define PB push_back
#define MP make_pair
#define F first
#define S second
#define SZ(x) ((int)(x).size())
#define ALL(x) (x).begin(),(x).end()
#ifdef _DEBUG_
  #define debug(...) printf(__VA_ARGS__)
#else
  #define debug(...) (void)0
#endif
using namespace std;
typedef long long ll;
typedef pair<int,int> PII;
typedef vector<int> VI;

int main() {
  return 0;
}
```

1.2 .vimrc

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

2 math

2.1 ext gcd

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

2.2 FFT

```
typedef complex<double> CD;

const double PI=acos(-1.0);
inline CD ang(double t) { return CD(cos(t), sin(t)); }

int rev_int(int x,int lgn) {
  int re=0;
  for(int i=0;i<lgn;i++) {
    re=(re<<1)+(x&1);
    x>>=1;
  }
  return re;
}
void fft(CD* A, int lgn, bool inv=false) {
  int n=1<<lgn;
  for(int i=0;i<n;i++)
    if(i<rev_int(i, lgn)) swap(A[i], A[rev_int(i, lgn)]);
  for(int i=1;i<n;i*=2) {
    CD W(1.0, 0.0), Wn;
    if(inv) Wn=ang(-PI/i);
    else Wn=ang(PI/i);
    for(int j=0;j<n;j++) {
      if(j&i) {
        W=CD(1.0, 0.0);
        continue;
      }
      CD x=A[j], y=A[j+i]*W;
      A[j]=x+y;
      A[j+i]=x-y;
      W*=Wn;
    }
  }
  if(inv)
    for(int i=0;i<n;i++)
      A[i]/=n;
}
```

2.3 NTT

```
//     MOD      Wn_       LGN
//     5767169     177147 19
//     7340033       2187 20
// 2013265921 440564289 27
const int MOD=786433;
const int Wn_=5; // 25 625
const int LGN=18;// 17  16
inline int add(int x,int y) { return (x+y)%MOD; }
inline int mul(int x,int y) { return 1ll*x*y%MOD; }
inline int sub(int x,int y) { return (x-y+MOD)%MOD; }

int pW[MOD]; // power of Wn
int divN;
int inv(int a) {
  int re=1, k=MOD-2, t=a;
  while(k) {
    if(k%2) re=mul(re, t);
    k/=2;
    t=mul(t, t);
  }
  return re;
}
void NTTinit(int lgn) { // call every time using new lgn !
  int Wn=Wn_;
  for(int i=lgn;i<LGN;i++) Wn=mul(Wn,Wn);
  divN=inv(1<<lgn);
  pW[0]=1;
  for(int i=1;;i++) {
    pW[i]=mul(pW[i-1], Wn);
    if(pW[i]==1) break;
  }
}

int rev_int(int x,int lgn) {
  int re=0;
  for(int i=0;i<lgn;i++) {
    re=(re<<1)+(x&1);
    x>>=1;
  }
  return re;
}
void ntt(int *A,int lgn,bool inv=false) {
  int n=1<<lgn;
  for(int i=0;i<n;i++)
    if(i<rev_int(i,lgn))
      swap(A[i], A[rev_int(i,lgn)]);
  for(int i=1;i<n;i*=2) {
    int W=1, Wn;
    if(inv) Wn=pW[n-(n/2/i)];
    else Wn=pW[n/2/i];
    for(int j=0;j<n;j++) {
      if(j&i) {
        W=1;
        continue;
      }
      int x=A[j], y=mul(A[j+i],W);
      A[j]=add(x,y);
      A[j+i]=sub(x,y);
      W=mul(W,Wn);
    }
  }
  if(inv)
    for(int i=0;i<n;i++)
      A[i]=mul(A[i],divN);
}
```

2.4 MillerRabin other

```
//input should < 2^63 - 1 (max prime :9223372036854775783)
typedef unsigned long long ull;

ull mul(ull a, ull b, ull n) {
  ull r = 0;
  a %= n, b %= n;
  while(b) {
    if(b&1) r = (a+r>=n ? a+r-n : a+r);
    a = (a+a>=n ? a+a-n : a+a);
    b >>= 1;
  }
  return r;
}

ull bigmod(ull a, ull d, ull n) {
  if(d==0)  return 1LL;
  if(d==1) return a % n;
  return mul(bigmod(mul(a, a, n), d/2, n),
      d%2?a:1, n);
}

const bool PRIME = 1, COMPOSITE = 0;
bool miller_rabin(ull n, ull a) {
  if(__gcd(a, n) == n)  return PRIME;
  if(__gcd(a, n) != 1)  return COMPOSITE;
  ull d = n-1, r = 0, res;
  while(d%2==0) { ++r; d/=2; }
  res = bigmod(a, d, n);
  if(res == 1 || res == n-1)  return PRIME;
  while(r--) {
    res = mul(res, res, n);
    if(res == n-1)  return PRIME;
  }
  return COMPOSITE;
}

bool isprime(ull n) {
  if(n==1)
    return COMPOSITE;
  ull as[7] = {2, 325, 9375, 28178, 450775,
      9780504, 1795265022};
  for(int i=0; i<7; i++)
    if(miller_rabin(n, as[i]) == COMPOSITE)
        return COMPOSITE;
  return PRIME;
}
```

2.5 Guass

```
// be care of the magic number 7 & 8
void guass() {
  for(int i = 0; i < 7; i++) {
    Frac tmp = mat[i][i]; // Frac -> the
        type of data
    for(int j = 0; j < 8; j++)
      mat[i][j] = mat[i][j] / tmp;
    for(int j = 0; j < 7; j++) {
      if(i == j)
        continue;
      Frac ratio = mat[j][i]; // Frac ->
          the type of data
      for(int k = 0; k < 8; k++)
        mat[j][k] = mat[j][k] - ratio * mat
            [i][k];
    }
  }
}
```

3 flow

3.1 dinic

```
const int MAXV=300;
const int MAXE=10000;
const int INF=(int)1e9+10;
// ^ config those things

struct E {
  int to,co;//capacity
  E(int t=0,int c=0):to(t),co(c) {}
}eg[2*MAXE];

// source:0  sink:n-1
struct Flow {
  VI e[MAXV];
  int ei,v;
  void init(int n) {
    v=n;
    ei=0;
    for(int i=0;i<n;i++)
      e[i]=VI();
  }
  void add(int a,int b,int c) { //a to b ,
      maxflow=c
    eg[ei]=E(b,c);
    e[a].PB(ei);
    ei++;
    eg[ei]=E(a,0);
    e[b].PB(ei);
    ei++;
  }

  int d[MAXV],qu[MAXV],ql,qr;
  bool BFS() {
    memset(d,-1,v*sizeof(int));
    ql=qr=0;
    qu[qr++]=0;
    d[0]=0;
    while(ql<qr && d[v-1]==-1) {
      int n=qu[ql++];
      VI &v=e[n];
      for(int i=SZ(v)-1;i>=0;i--) {
        int u=v[i];
        if(d[eg[u].to]==-1 && eg[u].co>0) {
          d[eg[u].to]=d[n]+1;
          qu[qr++]=eg[u].to;
        }
      }
    }
    return d[v-1]!=-1;
  }
  int ptr[MAXV];
  int go(int n,int p) {
    if(n==v-1)
      return p;
    VI &u=e[n];
    int temp;
    for(int i=ptr[n];i<SZ(u);i++) {
      if(d[n]+1!=d[eg[u[i]].to] || eg[u[i
          ]].co==0)
        continue;
      if((temp=go(eg[u[i]].to,min(p,eg[u[i
         ]].co)))==0)
        continue;
      eg[u[i]].co-=temp;
      eg[u[i]^1].co+=temp;
      ptr[n]=i;
      return temp;
    }
    ptr[n]=SZ(u);
    return 0;
  }
  int max_flow() {
    int ans=0,temp;
    while(BFS()) {
      for(int i=0;i<v;i++)
        ptr[i]=0;
      while((temp=go(0,INF))>0)
        ans+=temp;
    }
    return ans;
  }
}flow;
```

3.2 min-cost-max-flow

```
typedef pair<int,ll> PIL;
const int MAXV=60;
const int MAXE=6000;
const int INF=(int)1e9+10;
const ll cINF=(ll)1e18+10;
// ^ config those things

struct E {
  int to,ca,cost;//capacity, cost
  E(int t=0,int c=0,int co=0):to(t),ca(c),
     cost(co) {}
}eg[2*MAXE];

// source:0  sink:n-1
struct Flow {
  VI e[MAXV];
  int ei,n;
  void init(int n_) {
    n=n_;
    ei=0;
    for(int i=0;i<n;i++)
      e[i]=VI();
  }
  void add(int a,int b,int c,int d) {
    //a to b ,maxflow=c, cost=d
    eg[ei]=E(b,c,d);
    e[a].PB(ei);
    ei++;
    eg[ei]=E(a,0,-d);
    e[b].PB(ei);
    ei++;
  }

  PII d[MAXV]={};
  bool inq[MAXV]={};
  queue<int> que;
  VI pe;
  bool SPFA() {
    fill(d, d+n, MP(INF,INF));
    d[0]=MP(0,0);
    que.push(0);
    inq[0]=1;
    while(!que.empty()) {
      int v=que.front(); que.pop();
      inq[v]=0;
      for(int id:e[v]) {
        if(eg[id].ca>0 && MP(d[v].F+eg[id].
            cost,d[v].S+1)<d[eg[id].to]) {
          d[eg[id].to]=MP(d[v].F+eg[id].
             cost,d[v].S+1);
          if(!inq[eg[id].to]) {
            que.push(eg[id].to);
            inq[eg[id].to]=1;
          }
        }
      }
    }
    return d[n-1].F<INF;
  }
  PIL go(ll cb=cINF) {
    // cost_bound
    if(!SPFA()) return MP(0,0);
    pe.clear();
    int fl=INF;
    for(int v=n-1;v!=0;) {
      for(int id:e[v]) {
        int u=eg[id].to;
        const E& t=eg[id^1];
        if(t.ca>0 && MP(d[u].F+t.cost,d[u].
            S+1)==d[v]) {
          fl=min(fl, t.ca);
          v=u;
          pe.PB(id^1);
          break;
        }
      }
    }
    if(d[n-1].F>0) fl=min(1ll*fl, cb/d[n
       -1].F);
    for(int id:pe) {
      eg[id].ca-=fl;
      eg[id^1].ca+=fl;
    }
    return MP(fl, 1ll*fl*d[n-1].F);
  }
  PIL max_flow() {
    PIL ans=MP(0,0),temp;
    while((temp=go()).F>0) {
      ans.F+=temp.F;
      ans.S+=temp.S;
    }
    return ans;
  }
} flow;
```

4 string

4.1 KMP

```
void KMP_build(const char *S,int *F) {
  int p=F[0]=-1;
  for(int i=1;S[i];i++) {
    while(p!=-1 && S[p+1]!=S[i])
      p=F[p];
    if(S[p+1]==S[i])
      p++;
    F[i]=p;
  }
}

VI KMP_match(const char *S,const int *F,const char *T) {
  VI ans;
  int p=-1;
  for(int i=0;T[i];i++) {
    while(p!=-1 && S[p+1]!=T[i])
      p=F[p];
    if(S[p+1]==T[i])
      p++;
    if(!S[p+1]) {
      ans.PB(i-p);
      p=F[p];
    }
  }
  return ans;
}
```

4.2 Z-value

```
void Z_build(const char *S,int *Z) {
  Z[0]=0;
  int bst=0;
  for(int i=1;S[i];i++) {
    if(Z[bst]+bst<i) Z[i]=0;
    else Z[i]=min(Z[bst]+bst-i,Z[i-bst]);
    while(S[Z[i]]==S[i+Z[i]]) Z[i]++;
    if(Z[i]+i>Z[bst]+bst) bst=i;
  }
}
```

4.3 Z-value-palindrome

```
// AC code of NTUJ1871
char in[100100];
char s[200100];
int z[200100];

int main()
{
    while(gets(in))
    {
        int len=1;
        for(int i=0;in[i];i++)
        {
            s[len++]=’*’;
            s[len++]=in[i];
        }
        s[len]=0;
        z[0]=0;
        z[1]=0;
        int bst=1;
        for(int i=1;i<len;i++)
        {
            z[i]=min(bst+z[bst]-i,z[bst+bst-i]);
            while(s[i+z[i]+1]==s[i-z[i]-1])
                z[i]++;
            if(z[i]+i>bst+z[bst])
                bst=i;
        }
        /*for(int i=1;i<len;i++)
            putchar(s[i]);
        puts(””);
        for(int i=1;i<len;i++)
            printf(”%d”,z[i]);
        puts(””);*/
        bool yes=0;
        for(int i=3;i<len;i+=2)
            if(z[(i+1)/2]==i/2 && z[(i+len)/2]==(len-i-1)/2)
                yes=1;
        if(yes)
            puts(”www”);
        else
            puts(”vvvvvv”);
    }
    return 0;
}
```

4.4 Suffix Array($O(NlogN)$)

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

  void build_SA() {
    int maxR=0;
    for(int i=0;i<len;i++)
      R[i]=S[i];
    for(int i=0;i<=len;i++)
      R[len+i]=-1;
    memset(cnt,0,sizeof(cnt));
    for(int i=0;i<len;i++)
      maxR=max(maxR,R[i]);
    for(int i=0;i<len;i++)
      cnt[R[i]+1]++;
    for(int i=1;i<=maxR;i++)
      cnt[i]+=cnt[i-1];
    for(int i=0;i<len;i++)
      SA[cnt[R[i]]++]=i;
    for(int i=1;i<len;i*=2)
    {
      memset(cnt,0,sizeof(int)*(maxR+10));
      memcpy(tSA,SA,sizeof(int)*(len+10));
      memcpy(tR,R,sizeof(int)*(len+i+10));
      for(int j=0;j<len;j++)
        cnt[R[j]+1]++;
      for(int j=1;j<=maxR;j++)
        cnt[j]+=cnt[j-1];
      for(int j=len-i;j<len;j++)
        SA[cnt[R[j]]++]=j;
      for(int j=0;j<len;j++)
      {
        int k=tSA[j]-i;
        if(k<0)
          continue;
        SA[cnt[R[k]]++]=k;
      }
      int num=0;
      maxR=0;
      R[SA[0]]=num;
      for(int j=1;j<len;j++)
      {
        if(tR[SA[j-1]]<tR[SA[j]] || tR[SA[j-1]+i]<tR[SA[j]+i])
          num++;
        R[SA[j]]=num;
        maxR=max(maxR,R[SA[j]]);
      }
    }
  }
  void build_H() {
    memset(H,0,sizeof(int)*(len+10));
    for(int i=0;i<len;i++)
    {
      if(R[i]==0)
        continue;
      int &t=H[R[i]];
      if(i>0)
        t=max(0,H[R[i-1]]-1);
      while(S[i+t]==S[SA[R[i]-1]+t]) t++;
    }
  }
}sa;
```

4.5 Aho-Corasick

```
// AC code of UVa 10679
struct Trie {
  int c;
  bool fi=0;
  Trie *fail,*ch[52];
  Trie():c(0){memset(ch,0,sizeof(ch));}
}trie[1000100];

char m[1010],f[100100];
Trie *str[1010],*na,*root;

inline int c_i(char a) {
  return (a>=’A’ && a<=’Z’) ? a-’A’ : a-’a’+26;
}

void insert(char *s,int num) {
  Trie *at=root;
  while(*s) {
    if(!at->ch[c_i(*s)])
      at->ch[c_i(*s)]=new (na++) Trie();
    at=at->ch[c_i(*s)],s++;
  }
  str[num]=at;
}

Trie *q[1000100];
int ql,qr;

void init() {
  ql=qr=-1;
  q[++qr]=root;
  root->fail=NULL;
  while(ql<qr) {
    Trie *n=q[++ql],*f;
    for(int i=0;i<52;i++) {
      if(!n->ch[i])
        continue;
      f=n->fail;
      while(f && !f->ch[i])
        f=f->fail;
      n->ch[i]->fail=f?f->ch[i]:root;
      q[++qr]=n->ch[i];
    }
  }
}

void go(char *s) {
  Trie*p=root;
  while(*s) {
    while(p && !p->ch[c_i(*s)])
      p=p->fail;
    p=p?p->ch[c_i(*s)]:root;
    p->fi=1;
    s++;
  }
}

void AC() {
  for(int i=qr;i>0;i--)
    q[i]->fail->c+=q[i]->c;
}

int main() {
  int T,q;
  scanf(”%d”,&T);
  while(T--) {
    na=trie;
    root=new (na++) Trie();
    scanf(”%s”,f);
    scanf(”%d”,&q);
    for(int i=0;i<q;i++) {
      scanf(”%s”,m);
      insert(m,i);
    }
    init();
    go(f);
    for(int i=0;i<q;i++)
      puts(str[i]->fi?”y”:”n”);
  }
  return 0;
}
```

4.6 Aho-Corasick-2016ioicamp

```
// AC code of 2016ioicamp 54
const int MAXNM=100010;
int pp[MAXNM];

const int sizz=100010;
int nx[sizz][26],spt;
int fl[sizz],efl[sizz],ed[sizz];
int len[sizz];
int newnode(int len_=0) {
  for(int i=0;i<26;i++)nx[spt][i]=0;
  ed[spt]=0;
  len[spt]=len_;
  return spt++;
}
int add(char *s,int p) {
  int l=1;
  for(int i=0;s[i];i++) {
    int a=s[i]-'a';
    if(nx[p][a]==0) nx[p][a]=newnode(l);
    p=nx[p][a];
    l++;
  }
  ed[p]=1;
  return p;
}
int q[sizz],qs,qe;
void make_fl(int root) {
  fl[root]=efl[root]=0;
  qs=qe=0;
  q[qe++]=root;
  for(;qs!=qe;) {
    int p=q[qs++];
    for(int i=0;i<26;i++) {
      int t=nx[p][i];
      if(t==0) continue;
      int tmp=fl[p];
      for(;tmp&&nx[tmp][i]==0;) tmp=fl[tmp
          ];
      fl[t]=tmp?nx[tmp][i]:root;
      efl[t]=ed[fl[t]]?fl[t]:efl[fl[t]];
      q[qe++]=t;
    }
  }
}
char s[MAXNM];
char a[MAXNM];

int dp[MAXNM][4];

void mmax(int &a,int b) {
  a=max(a,b);
}

void match(int root) {
  int p=root;
  for(int i=1;s[i];i++) {
    int a=s[i]-'a';
    for(;p&&nx[p][a]==0;p=fl[p]);
    p=p?nx[p][a]:root;
    for(int j=1;j<=3;j++)
      dp[i][j]=dp[i-1][j];
    for(int t=p;t;t=efl[t]) {
      if(!ed[t])
        continue;
      for(int j=1;j<=3;j++)
        mmax(dp[i][j],dp[i-len[t]][j-1]+(pp
            [i]-pp[i-len[t]]));
    }
  }
}

int main() {
  int T;
  scanf("%d",&T);
  while(T--) {
    int n,m;
    scanf("%d%d",&n,&m);
    scanf("%s",s+1);
    for(int i=1;i<=n;i++)
      scanf("%d",pp+i);
    for(int i=1;i<=n;i++)
      pp[i]+=pp[i-1];
    spt=1;
    int root=newnode();
    for(int i=0;i<m;i++) {
      scanf("%s",a);
      add(a,root);
    }
    make_fl(root);
    for(int i=1;i<=n;i++)
      dp[i][1]=dp[i][2]=dp[i][3]=0;
    match(root);
    printf("%d\n",dp[n][3]);
  }
  return 0;
}
```

4.7 Palindrome Automaton

```
const int MAXN=100050;
char s[MAXN];
int n; // n: string length

typedef pair<PII,int> PD;
vector<PD> pal;

int ch[MAXN][26], fail[MAXN], len[MAXN],
    cnt[MAXN];
int edp[MAXN];
int nid=1;
int new_node(int len_) {
  len[nid]=len_;
  return nid++;
}

void build_pa() {
  int odd_root=new_node(-1);
  int even_root=new_node(0);
  fail[even_root]=odd_root;
  int cur=even_root;
  for(int i=1;i<=n;i++) {
    while(1) {
      if(s[i-len[cur]-1] == s[i]) break;
      cur=fail[cur];
    }
    if(ch[cur][s[i]-’a’]==0) {
      int nt=ch[cur][s[i]-’a’]=new_node(len[cur]+2);
      int tmp=fail[cur];
      while(tmp && s[i-len[tmp]-1]!=s[i]) tmp=fail[tmp];
      if(tmp==0) fail[nt]=even_root;
      else {
        assert(ch[tmp][s[i]-’a’]);
        fail[nt]=ch[tmp][s[i]-’a’];
      }
      edp[nt]=i;
    }
    cur=ch[cur][s[i]-’a’];
    cnt[cur]++;
  }
  for(int i=nid-1;i>even_root;i--) {
    cnt[fail[i]]+=cnt[i];
    pal.PB( MP( MP(edp[i]-len[i]+1, len[i]) , cnt[i]) );
  }
}
```

4.8 Suffix Automaton(bcw)

```
// par : fail link
// val : a topological order ( useful for DP )
// go[x] : automata edge ( x is integer in [0,26) )

struct SAM{
  struct State{
    int par, go[26], val;
    State () : par(0), val(0){ FZ(go); }
    State (int _val) : par(0), val(_val){ FZ(go); }
  };
  vector<State> vec;
  int root, tail;

  void init(int arr[], int len){
    vec.resize(2);
    vec[0] = vec[1] = State(0);
    root = tail = 1;
    for (int i=0; i<len; i++)
      extend(arr[i]);
  }
  void extend(int w){
    int p = tail, np = vec.size();
    vec.PB(State(vec[p].val+1));
    for ( ; p && vec[p].go[w]==0; p=vec[p].par)
      vec[p].go[w] = np;
    if (p == 0){
      vec[np].par = root;
    } else {
      if (vec[vec[p].go[w]].val == vec[p].val+1){
        vec[np].par = vec[p].go[w];
      } else {
        int q = vec[p].go[w], r = vec.size();
        vec.PB(vec[q]);
        vec[r].val = vec[p].val+1;
        vec[q].par = vec[np].par = r;
        for ( ; p && vec[p].go[w] == q; p=vec[p].par)
          vec[p].go[w] = r;
      }
    }
    tail = np;
  }
};
```

5 graph

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

```
// NTUJ1263
bool is(ll x)
{
  ll l=1,r=2000000,m;
  while(l<=r)
  {
    m=(l+r)/2;
    if(m*m==x)
      return 1;
    if(m*m<x)
      l=m+1;
    else
      r=m-1;
  }
  return 0;
}

VI odd,even;
int in[300];
VI e[300];
int match[300];
bool vis[300];

bool DFS(int x)
{
  vis[x]=1;
  for(int u:e[x])
  {
    if(match[u]==-1 || (!vis[match[u]]&&DFS(match[u])))
    {
      match[u]=x;
      match[x]=u;
      return 1;
    }
  }
  return 0;
}

int main()
{
  int N;
  while(scanf(”%d”,&N)==1)
  {
    odd.clear();
    even.clear();
    for(int i=0;i<N;i++)
      e[i].clear();
    for(int i=0;i<N;i++)
    {
      scanf("%d",in+i);
      if(in[i]%2==0)
        even.pb(i);
      else
        odd.pb(i);
    }
    for(int i:even)
      for(int j:odd)
        if(is(1ll*in[i]*in[i]+1ll*in[j]*in[
            j]) && __gcd(in[i],in[j])==1)
          e[i].pb(j), e[j].pb(i);
    int ans=0;
    fill(match,match+N,-1);
    for(int i=0;i<N;i++)
      if(match[i]==-1)
      {
        fill(vis,vis+N,0);
        if(DFS(i))
          ans++;
      }
    printf("%d\n",ans);
  }
  return 0;
}
```

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

```
const int INF=1016; //> max(a[i][j])
const int MAXN=650;
int a[MAXN][MAXN]; // weight [x][y] , two
    set of vertex
int N; // two set: each set have exactly N
    vertex
int match[MAXN*2], weight[MAXN*2];
bool vis[MAXN*2];

bool DFS(int x) {
  vis[x]=1;
  for(int i=0;i<N;i++) {
    if(weight[x]+weight[N+i]!=a[x][i])
        continue;
    vis[N+i]=1;
    if(match[N+i]==-1 || (!vis[match[N+i
        ]]&&DFS(match[N+i]))) {
      match[N+i]=x;
      match[x]=N+i;
      return 1;
    }
  }
  return 0;
}

int KM() {
  fill(weight, weight+N+N, 0);
  for(int i=0;i<N;i++) {
    for(int j=0;j<N;j++)
      weight[i]=max(weight[i], a[i][j]);
  }
  fill(match, match+N+N, -1);
  for(int u=0;u<N;u++) {
    fill(vis, vis+N+N, 0);
    while(!DFS(u)) {
      int d=INF;
      for(int i=0;i<N;i++) {
        if(!vis[i]) continue;
        for(int j=0;j<N;j++)
          if(!vis[N+j])
            d=min(d, weight[i]+weight[N+j]-
                a[i][j]);
      }
      for(int i=0;i<N;i++)
        if(vis[i])
          weight[i]-=d;
      for(int i=N;i<N+N;i++)
        if(vis[i])
          weight[i]+=d;
      fill(vis, vis+N+N, 0);
    }
  }
  int ans=0;
  for(int i=0;i<N+N;i++) ans+=weight[i];
  return ans;
}
```

5.3 general graph matching(bcw)

```
#define FZ(x) memset(x,0,sizeof(x))
struct GenMatch { // 1-base
  static const int MAXN = 250;
  int V;
  bool el[MAXN][MAXN];
  int pr[MAXN];
  bool inq[MAXN],inp[MAXN],inb[MAXN];
  queue<int> qe;
  int st,ed;
  int nb;
  int bk[MAXN],djs[MAXN];
  int ans;
  void init(int _V) {
    V = _V;
    FZ(el); FZ(pr);
    FZ(inq); FZ(inp); FZ(inb);
    FZ(bk); FZ(djs);
    ans = 0;
  }
  void add_edge(int u, int v) {
    el[u][v] = el[v][u] = 1;
  }
  int lca(int u,int v) {
    memset(inp,0,sizeof(inp));
    while(1) {
      u = djs[u];
      inp[u] = true;
      if(u == st) break;
      u = bk[pr[u]];
    }
    while(1) {
      v = djs[v];
      if(inp[v]) return v;
      v = bk[pr[v]];
    }
    return v;
  }
  void upd(int u) {
    int v;
    while(djs[u] != nb) {
      v = pr[u];
      inb[djs[u]] = inb[djs[v]] = true;
      u = bk[v];
      if(djs[u] != nb) bk[u] = v;
    }
  }
  void blo(int u,int v) {
    nb = lca(u,v);
    memset(inb,0,sizeof(inb));
    upd(u); upd(v);
    if(djs[u] != nb) bk[u] = v;
    if(djs[v] != nb) bk[v] = u;
    for(int tu = 1; tu <= V; tu++)
      if(inb[djs[tu]]) {
        djs[tu] = nb;
        if(!inq[tu]){
          qe.push(tu);
          inq[tu] = 1;
        }
      }
  }
  void flow() {
    memset(inq,false,sizeof(inq));
    memset(bk,0,sizeof(bk));
    for(int i = 1; i <= V;i++)
      djs[i] = i;

    while(qe.size()) qe.pop();
    qe.push(st);
    inq[st] = 1;
    ed = 0;
    while(qe.size()) {
      int u = qe.front(); qe.pop();
      for(int v = 1; v <= V; v++)
        if(el[u][v] && (djs[u] != djs[v])
            && (pr[u] != v)) {
          if((v == st) || ((pr[v] > 0) &&
              bk[pr[v]] > 0))
            blo(u,v);
          else if(bk[v] == 0) {
            bk[v] = u;
            if(pr[v] > 0) {
              if(!inq[pr[v]]) qe.push(pr[v
                  ]);
            } else {
              ed = v;
              return;
            }
          }
        }
    }
  }
  void aug() {
    int u,v,w;
    u = ed;
    while(u > 0) {
      v = bk[u];
      w = pr[v];
      pr[v] = u;
      pr[u] = v;
      u = w;
    }
  }
  int solve() {
    memset(pr,0,sizeof(pr));
    for(int u = 1; u <= V; u++)
      if(pr[u] == 0) {
        st = u;
        flow();
        if(ed > 0) {
          aug();
          ans ++;
        }
      }
    return ans;
  }
} gm;
```

5.4 minimum general graph weighted matching(bcw)

```
struct Graph {
  // Minimum General Weighted Matching (
      Perfect Match) 0-base
  static const int MXN = 105;

  int n, edge[MXN][MXN];
  int match[MXN],dis[MXN],onstk[MXN];
  vector<int> stk;

  void init(int _n) {
    n = _n;
    for (int i=0; i<n; i++)
      for (int j=0; j<n; j++)
        edge[i][j] = 0;
  }
  void add_edge(int u, int v, int w) {
    edge[u][v] = edge[v][u] = w;
  }
  bool SPFA(int u){
    if (onstk[u]) return true;
    stk.PB(u);
    onstk[u] = 1;
    for (int v=0; v<n; v++){
      if (u != v && match[u] != v && !onstk
          [v]){
        int m = match[v];
        if (dis[m] > dis[u] - edge[v][m] +
            edge[u][v]){
          dis[m] = dis[u] - edge[v][m] +
              edge[u][v];
          onstk[v] = 1;
          stk.PB(v);
          if (SPFA(m)) return true;
          stk.pop_back();
          onstk[v] = 0;
        }
      }
    }
    onstk[u] = 0;
    stk.pop_back();
    return false;
  }

  int solve() {
    // find a match
    for (int i=0; i<n; i+=2){
      match[i] = i+1;
      match[i+1] = i;
    }
    while (true){
      int found = 0;
      for (int i=0; i<n; i++)
        dis[i] = onstk[i] = 0;
      for (int i=0; i<n; i++){
        stk.clear();
        if (!onstk[i] && SPFA(i)){
          found = 1;
          while (SZ(stk)>=2){
            int u = stk.back(); stk.
                pop_back();
            int v = stk.back(); stk.
                pop_back();
            match[u] = v;
            match[v] = u;
          }
        }
      }
      if (!found) break;
    }
    int ret = 0;
    for (int i=0; i<n; i++)
      ret += edge[i][match[i]];
    ret /= 2;
    return ret;
  }
}graph;
```

5.5 Max clique(bcw)

```
class MaxClique {
public:
    static const int MV = 210;

    int V;
    int el[MV][MV/30+1];
    int dp[MV];
    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

```
const int MAXN=1010;
const int MAXM=5010;
VI e[MAXN];
int low[MAXN],lvl[MAXN],bel[MAXN];
bool vis[MAXN];
int cnt;
VI st;
void DFS(int x,int l,int p) {
  st.PB(x);
  vis[x]=1;
  low[x]=lvl[x]=l;
  bool top=0;
  for(int u:e[x]) {
    if(u==p && !top) {
      top=1;
      continue;
    }
    if(!vis[u]) {
      DFS(u,l+1,x);
    }
    low[x]=min(low[x],low[u]);
  }
  if(x==1 || low[x]==l) {
    while(st.back()!=x) {
      bel[st.back()]=cnt;
      st.pop_back();
    }
    bel[st.back()]=cnt;
    st.pop_back();
    cnt++;
  }
}
int main() {
  int T;
  scanf("%d",&T);
  while(T--) {
    int N,M,a,b;
    scanf("%d%d",&N,&M);
    fill(vis,vis+N+1,0);
    for(int i=1;i<=N;i++)
      e[i].clear();
    while(M--) {
      scanf("%d%d",&a,&b);
      e[a].PB(b);
      e[b].PB(a);
    }
    cnt=0;
    DFS(1,0,-1);
    /*****/
  }
  return 0;
}
```

5.7 VerticeBCC

```
const int MAXN=10000;
const int MAXE=100000;

VI e[MAXN+10];
vector<PII> BCC[MAXE];
int bccnt;
vector<PII> st;
bool vis[MAXN+10];
int low[MAXN+10],level[MAXN+10];

void DFS(int x,int p,int l) {
  vis[x]=1;
  level[x]=low[x]=l;
  for(int u:e[x]) {
    if(u==p)
      continue;
    if(vis[u]) {
      if(level[u]<l) {
        st.PB(MP(x,u));
        low[x]=min(low[x],level[u]);
      }
    }
    else {
      st.PB(MP(x,u));
      DFS(u,x,l+1);
      if(low[u]>=l) {
        PII t=st.back();
        st.pop_back();
        while(t!=MP(x,u)) {
          BCC[bccnt].PB(t);
          t=st.back();
          st.pop_back();
        }
        BCC[bccnt].PB(t);
        bccnt++;
      }
      low[x]=min(low[x],low[u]);
    }
  }
}

int main() {
  int T,N,M;
  scanf("%d",&T);
  while(T--) {
    scanf("%d%d",&N,&M);
    for(int i=0;i<N;i++)
      e[i].clear();
    int cnt=0;
    while(1) {
      int x,y;
      scanf("%d%d",&x,&y);
      if(x==-1 && y==-1)
        break;
      cnt++;
      e[x].PB(y);
      e[y].PB(x);
    }
    for(int i=0;i<N;i++) { // no multi-edge
      sort(ALL(e[i]));
      e[i].erase(unique(ALL(e[i])),e[i].end
          ());
    }
    fill(vis,vis+N,0);
    while(bccnt)
      BCC[--bccnt].clear();
    DFS(0,-1,0);
    /***/
  }
  return 0;
}
```

5.8 Dominating Tree

```
const int MAXN = 200000 + 10;

VI e[MAXN], re[MAXN];
int par[MAXN], num[MAXN], t, rn[MAXN];
int sd[MAXN], id[MAXN];
PII p[MAXN];
VI sdom_at[MAXN];

void dfs(int u) {
  num[u] = ++t;
  rn[t] = u;
  for(int v : e[u]) {
```

```cpp
    if(num[v])  continue;
    par[v] = u;
    dfs(v);
  }
}

void LINK(int x, int y) {
  p[x].F = y;
  if(sd[y] < sd[p[x].S])  p[x].S = y;
}

int EVAL(int x) {
  if(p[p[x].F].F != p[x].F) {
    int w = EVAL(p[x].F);
    if(sd[w] < sd[p[x].S])  p[x].S = w;
    p[x].F = p[ p[x].F ].F;
  }
  return p[x].S;
}

void DominatingTree(int n) {
  // 1-indexed
  par[1] = 1;
  fill(num, num+n+1, 0);
  fill(rn, rn+n+1, 0);
  t = 0;
  dfs(1);

  for(int i=1; i<=n; i++) {
    p[i] = MP(i, i);
  }
  for(int i=1; i<=n; i++) {
    sd[i] = (num[i] ? num[i] : MAXN+10);
    id[i] = i;
  }
  for(int i=n; i>1; i--) {
    int v = rn[i];
    if(!v)  continue;
    for(int u : re[v]) {
      int w = EVAL(u);
      sd[v] = min(sd[v], sd[w]);
    }
    sdom_at[rn[sd[v]]].PB(v);
    LINK(v, par[v]);

    for(int w : sdom_at[par[v]]) {
      int u = EVAL(w);
      id[w] = (sd[u]<sd[w] ? u : par[v]);
    }
    sdom_at[par[v]].clear();
  }

  for(int i=2; i<=n; i++) {
    int v = rn[i];
    if(!v)  break;
    if(id[v] != rn[sd[v]])  id[v] = id[id[v
        ]];
  }
}
```

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

```cpp
const int N = 100000 + 10;

struct Treap {
  static Treap mem[N], *pmem;

  int sz, pri;
  ll val, sum, add;
  Treap *l, *r;

  Treap() {}
  Treap(ll _val):
    l(NULL), r(NULL), sz(1), pri(rand()),
        val(_val), sum(_val), add(0) {}
} Treap::mem[N], *Treap::pmem = Treap::mem;

Treap* make(ll val) {
  return new (Treap::pmem++) Treap(val);
}

inline int sz(Treap *t) {
  return t ? t->sz : 0;
}

inline ll sum(Treap *t) {
  return t ? t->sum + t->add * sz(t) : 0;
}

inline void add(Treap *t, ll x) {
  t->add += x;
}

void push(Treap *t) {
  t->val += t->add;
  if(t->l)  t->l->add += t->add;
  if(t->r)  t->r->add += t->add;
  t->add = 0;
}

void pull(Treap *t) {
  t->sum = sum(t->l) + sum(t->r) + t->val;
  t->sz = sz(t->l) + sz(t->r) + 1;
}

Treap* merge(Treap *a, Treap *b) {
  if(!a || !b)  return a ? a : b;
  else if(a->pri > b->pri) {
    push(a);
    a->r = merge(a->r, b);
    pull(a);
    return a;
  }
  else {
    push(b);
    b->l = merge(a, b->l);
    pull(b);
    return b;
  }
}

void split(Treap* t, int k, Treap *&a,
    Treap *&b) {
  if(!t)  a = b = NULL;
  else if(sz(t->l) < k) {
    a = t;
    push(a);
    split(t->r, k - sz(t->l) - 1, a->r, b);
    pull(a);
  }
  else {
    b = t;
    push(b);
    split(t->l, k, a, b->l);
    pull(b);
  }
}

int main() {
  srand(105105);

  int n, q;
  scanf("%d%d", &n, &q);

  Treap *t = NULL;
  for(int i = 0; i < n; i++) {
    ll tmp;
    scanf("%lld", &tmp);
    t = merge(t, make(tmp));
  }

  while(q--) {
    char c;
    int l, r;
    scanf("\n%c %d %d", &c, &l, &r);

    Treap *tl = NULL, *tr = NULL;
    if(c == 'Q') {
      split(t, l - 1, tl, t);
      split(t, r - l + 1, t, tr);
      printf("%lld\n", sum(t));
      t = merge(tl, merge(t, tr));
    }
    else {
      ll x;
      scanf("%lld", &x);
      split(t, l - 1, tl, t);
      split(t, r - l + 1, t, tr);
      add(t, x);
      t = merge(tl, merge(t, tr));
    }
  }

  return 0;
}
```

6.2 copy on write treap

```
const int N = 1000000 + 10;

struct Treap {
    char val;
    int sz, refs;
    Treap *l, *r;

    Treap() {}
    Treap(char _val):
        val(_val), sz(1), refs(0), l(NULL),
            r(NULL) {}
};

Treap* make(Treap* t) {
    return new Treap(*t);
}

Treap* make(char _val) {
    return new Treap(_val);
}

void print_ref(Treap* t) {
    if(!t)  return ;
    print_ref(t->l);
    printf("%d ", t->refs);
    print_ref(t->r);
}

void print(Treap* t) {
    if(!t)  return ;
    print(t->l);
    putchar(t->val);
    print(t->r);
}

void takeRef(Treap* t) {
    if(t)   t->refs++;
}

void dropRef(Treap* t) {
    if(t) {
        char c = t->val;
        t->refs--;
        if(t->refs <= 0) {
            dropRef(t->l);
            dropRef(t->r);
            delete t;
        }
    }
}

int sz(Treap* t) {
    return t ? t->sz : 0;
}

int rnd(int m) {
    static int x = 851025;
    return (x = (x*0xdefaced+1) & INT_MAX)
        % m;
}

void pull(Treap* t) {
    t->sz = sz(t->l) + sz(t->r) + 1;
}

Treap* merge(Treap* a, Treap* b) {
    if(!a || !b) {
        Treap* t = a ? make(a) : make(b);
        t->refs = 0;
        takeRef(t->l);
        takeRef(t->r);
        return t;
    }

    Treap* t;
    if( rnd(a->sz+b->sz) < a->sz) {
        t = make(a);
        t->refs = 0;
        t->r = merge(a->r, b);
        takeRef(t->l);
        takeRef(t->r);
    }
    else {
        t = make(b);
        t->refs = 0;
        t->l = merge(a, b->l);
        takeRef(t->l);
        takeRef(t->r);
    }

    pull(t);
    return t;
}

void split(Treap* t, int k, Treap* &a,
    Treap* &b) {
    if(!t)  a = b = NULL;
    else if(sz(t->l) < k) {
        a = make(t);
        a->refs = 0;
        split(a->r, k-sz(t->l)-1, a->r, b);
        takeRef(a->l);
        takeRef(a->r);
        pull(a);
    }
    else {
        b = make(t);
        b->refs = 0;
        split(b->l, k, a, b->l);
        takeRef(b->l);
        takeRef(b->r);
        pull(b);
    }
}

void print_inorder(Treap* t) {
    if(!t)  return ;
    putchar(t->val);
    print_inorder(t->l);
    print_inorder(t->r);
}

char s[N];

int main() {
    int m;
    scanf("%d", &m);
    scanf("%s", s);
    int n = strlen(s);
    int q;
    scanf("%d", &q);

    Treap* t = NULL;
    for(int i = 0; i < n; i++) {
        Treap *a = t, *b = make(s[i]);
        t = merge(a, b);
        dropRef(a);
        dropRef(b);
    }

    while(q--) {
        int l, r, x;
        scanf("%d%d%d", &l, &r, &x);
        r++;

        Treap *a, *b, *c, *d;
        a = b = c = d = NULL;
        split(t, l, a, b);
        dropRef(a);
        split(b, r-l, c, d);
        dropRef(b);
        dropRef(d);
        split(t, x, a, b);
        dropRef(t);
        Treap* t2 = merge(c, b);
        dropRef(b);
        dropRef(c);
        t = merge(a, t2);
        dropRef(a);
        dropRef(t2);

        if(t->sz > m) {
            Treap* t2 = NULL;
            split(t, m, t2, a);
            dropRef(a);
            dropRef(t);
            t = t2;
        }
    }

    print(t);
    putchar('\n');

    return 0;
}
```

6.3 copy on write segment tree

```
const int N = 50000 + 10;
const int Q = 10000 + 10;

struct Seg {
  static Seg mem[N*80], *pmem;

  int val;
  Seg *tl, *tr;

  Seg() :
    tl(NULL), tr(NULL), val(0) {}

  Seg* init(int l, int r) {
    Seg* t = new (pmem++) Seg();
    if(l != r) {
      int m = (l+r)/2;
      t->tl = init(l, m);
      t->tr = init(m+1, r);
    }
    return t;
  }

  Seg* add(int k, int l, int r) {
    Seg* _t = new (pmem++) Seg(*this);
    if(l==r) {
      _t->val++;
      return _t;
    }

    int m = (l+r)/2;
    if(k <= m)  _t->tl = tl->add(k, l, m);
    else     _t->tr = tr->add(k, m+1, r);

    _t->val = _t->tl->val + _t->tr->val;
    return _t;
  }
} Seg::mem[N*80], *Seg::pmem = mem;

int query(Seg* ta, Seg* tb, int k, int l, int r) {
  if(l == r)  return l;

  int m = (l+r)/2;

  int a = ta->tl->val;
  int b = tb->tl->val;
  if(b-a >= k)  return query(ta->tl, tb->tl, k, l, m);
  else       return query(ta->tr, tb->tr, k-(b-a), m+1, r);
};

struct Query {
  int op, l, r, k, c, v;

  bool operator<(const Query b) const {
    return c < b.c;
  }
} qs[Q];
int arr[N];
Seg *t[N];
vector<int> vec2;

int main() {
  int T;
  scanf("%d", &T);

  while(T--) {
    int n, q;
    scanf("%d%d", &n, &q);

    for(int i = 1; i <= n; i++) {
      scanf("%d", arr+i);
      vec2.push_back(arr[i]);
    }
    for(int i = 0; i < q; i++) {
      scanf("%d", &qs[i].op);
      if(qs[i].op == 1) scanf("%d%d%d", &qs[i].l, &qs[i].r, &qs[i].k);
      else  scanf("%d%d", &qs[i].c, &qs[i].v);

      if(qs[i].op == 2) vec2.push_back(qs[i].v);
    }
    sort(vec2.begin(), vec2.end());
    vec2.resize(unique(vec2.begin(), vec2.end())-vec2.begin());
    for(int i = 1; i <= n; i++) arr[i] = lower_bound(vec2.begin(), vec2.end(), arr[i]) - vec2.begin();
    int mn = 0, mx = vec2.size()-1;

    for(int i = 0; i <= n; i++) t[i] = NULL;
    t[0] = new (Seg::pmem++) Seg();
    t[0] = t[0]->init(mn, mx);
    int ptr = 0;
    for(int i = 1; i <= n; i++) {
      t[i] = t[i-1]->add(arr[i], mn, mx);
    }

    for(int i = 0; i < q; i++) {
      int op = qs[i].op;
      if(op == 1) {
        int l = qs[i].l, r = qs[i].r, k = qs[i].k;
        printf("%d\n", vec2[query(t[l-1], t[r], k, mn, mx)]);
      }
      if(op == 2) {
        continue;
      }
      if(op == 3) puts("7122");
    }

    vec2.clear();
    Seg::pmem = Seg::mem;
  }

  return 0;
}
```

6.4 Treap+(HOJ 92)

```
const int INF = 103456789;

struct Treap {
    int pri, sz, val, chg, rev, sum, lsum, rsum, mx_sum;
    Treap *l, *r;

    Treap() {}
    Treap(int _val) :
        pri(rand()), sz(1), val(_val), chg(INF), rev(0), sum(_val), lsum(_val), rsum(_val), mx_sum(_val), l(NULL), r(NULL) {}
};

```

```cpp
int sz(Treap* t) {return t ? t->sz : 0;}
int sum(Treap* t) {
    if(!t)  return 0;
    if(t->chg == INF)   return t->sum;
    else    return t->chg*t->sz;
}
int lsum(Treap* t) {
    if(!t)  return -INF;
    if(t->chg != INF)   return max(t->chg,
       (t->chg)*(t->sz));
    if(t->rev)  return t->rsum;
    return t->lsum;
}
int rsum(Treap* t) {
    if(!t)  return -INF;
    if(t->chg != INF)   return max(t->chg,
       (t->chg)*(t->sz));
    if(t->rev)  return t->lsum;
    return t->rsum;
}
int mx_sum(Treap* t) {
    if(!t)  return -INF;
    if(t->chg != INF)   return max(t->chg,
       (t->chg)*(t->sz));
    return t->mx_sum;
}

void push(Treap* t) {
    if(t->chg != INF) {
        t->val = t->chg;
        t->sum = (t->sz) * (t->chg);
        t->lsum = t->rsum = t->mx_sum = max
           (t->sum, t->val);
        if(t->l)    t->l->chg = t->chg;
        if(t->r)    t->r->chg = t->chg;
        t->chg = INF;
    }
    if(t->rev) {
        swap(t->l, t->r);
        if(t->l)    t->l->rev ^= 1;
        if(t->r)    t->r->rev ^= 1;
        t->rev = 0;
    }
}

void pull(Treap* t) {
    t->sz = sz(t->l)+sz(t->r)+1;
    t->sum = sum(t->l)+sum(t->r)+t->val;
    t->lsum = max(lsum(t->l), sum(t->l)+max
       (0, lsum(t->r))+t->val);
    t->rsum = max(rsum(t->r), sum(t->r)+max
       (0, rsum(t->l))+t->val);
    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);
}

Treap* merge(Treap* a, Treap* b) {
    if(!a || !b)    return a ? a : b;
    if(a->pri > b->pri) {
        push(a);
        a->r = merge(a->r, b);
        pull(a);
        return a;
    }
    else {
        push(b);
        b->l = merge(a, b->l);
        pull(b);
        return b;
    }
}

void split(Treap* t, int k, Treap* &a,
    Treap* &b) {
    if(!t) {
        a = b = NULL;
        return ;
    }
    push(t);
    if(sz(t->l) < k) {
        a = t;
        push(a);
        split(t->r, k-sz(t->l)-1, a->r, b);
        pull(a);
    }
    else {
        b = t;
        push(b);
        split(t->l, k, a, b->l);
        pull(b);
    }
}

void del(Treap* t) {
    if(!t)  return;
    del(t->l);
    del(t->r);
    delete t;
}

int main() {
    srand(7122);

    int n, m;
    scanf("%d%d", &n, &m);

    Treap* t = NULL;
    for(int i = 0; i < n; i++) {
        int x;
        scanf("%d", &x);
        t = merge(t, new Treap(x));
    }

    while(m--) {
        char s[15];
        scanf("%s", s);

        Treap *tl = NULL, *tr = NULL, *t2 =
           NULL;

        if(!strcmp(s, "INSERT")) {
            int p, k;
            scanf("%d%d", &p, &k);
            for(int i = 0; i < k; i++) {
                int x;
                scanf("%d", &x);
                t2 = merge(t2, new Treap(x)
                   );
            }
```

```
      split(t, p, tl, tr);
      t = merge(tl, merge(t2, tr));
    }

    if(!strcmp(s, ”DELETE”)) {
      int p, k;
      scanf(”%d%d”, &p, &k);
      split(t, p-1, tl, t);
      split(t, k, t, tr);
      del(t);
      t = merge(tl, tr);
    }

    if(!strcmp(s, ”MAKE-SAME”)) {
      int p, k, l;
      scanf(”%d%d%d”, &p, &k, &l);
      split(t, p-1, tl, t);
      split(t, k, t, tr);
      if(t)   t->chg = l;
      t = merge(tl, merge(t, tr));
    }

    if(!strcmp(s, ”REVERSE”)) {
      int p, k;
      scanf(”%d%d”, &p, &k);
      split(t, p-1, tl, t);
      split(t, k, t, tr);
      if(t)   t->rev ^= 1;
      t = merge(tl, merge(t, tr));
    }

    if(!strcmp(s, ”GET-SUM”)) {
      int p, k;
      scanf(”%d%d”, &p, &k);
      split(t, p-1, tl, t);
      split(t, k, t, tr);
      printf(”%d\n”, sum(t));
      t = merge(tl, merge(t, tr));
    }

    if(!strcmp(s, ”MAX-SUM”)) {
      printf(”%d\n”, mx_sum(t));
    }
  }

  return 0;
}
```

6.5 Leftist Tree

```
struct Left {
 Left *l,*r;
 int v,h;
 Left(int v_) : v(v_), h(1), l(0), r(0) {}
};

int height(Left *p) { return p ? p -> h : 0 ; }

Left* combine(Left *a,Left *b) {
 if(!a || !b) return a ? a : b ;
 Left *p ;
 if( a->v > b->v) {
  p = a;
  p -> r = combine( p -> r , b );
 }
 else {
  p = b;
  p -> r = combine( p -> r , a );
 }
 if( height( p->l ) < height( p->r ) )
  swap( p->l , p->r );
 p->h = min( height( p->l ) , height( p->r ) ) + 1;
 return p;
}
Left *root;

void push(int v) {
 Left *p = new Left(v);
 root = combine( root , p );
}
int top() { return root? root->v : -1; }
void pop() {
 if(!root) return;
 Left *a = root->l , *b = root->r ;
 delete root;
 root = combine( a , b );
}
void clear(Left* &p) {
 if(!p)
  return;
 if(p->l) clear(p->l);
 if(p->r) clear(p->r);
 delete p;
 p = 0 ;
}

int main() {
 int T,n,x,o,size;
 bool bst,bqu,bpq;
 scanf(”%d”,&T);
 while(T--) {
  bst=bqu=bpq=1;
  stack<int> st;
  queue<int> qu;
  clear(root);
  size=0;
  scanf(”%d”,&n);
  while(n--) {
   scanf(”%d%d”,&o,&x);
   if(o==1)
    st.push(x),qu.push(x),push(x),size++;
   else if(o==2) {
    size--;
    if(size<0)
     bst=bqu=bpq=0;
    if(bst) {
     if(st.top()!=x)
      bst=0;
     st.pop();
    }
    if(bqu) {
     if(qu.front()!=x)
      bqu=0;
     qu.pop();
    }
```

```
        if(bpq) {
        //  printf("(%d)\n",top());
          if(top()!=x)
            bpq=0;
          pop();
        }
      }
    }
    int count=0;
    if(bst)
      count++;
    if(bqu)
      count++;
    if(bpq)
      count++;

    if(count>1)
      puts("not sure");
    else if(count==0)
      puts("impossible");
    else if(bst)
      puts("stack");
    else if(bqu)
      puts("queue");
    else if(bpq)
      puts("priority queue");
  }
  return 0;
}
```

6.6 Link Cut Tree

```
const int MAXN = 100000 + 10;

struct SplayTree {
  int val, mx, ch[2], pa;
  bool rev;
  void init() {
    val = mx = -1;
    rev = false;
    pa = ch[0] = ch[1] = 0;
  }
} node[MAXN*2];

inline bool isroot(int x) {
  return node[node[x].pa].ch[0]!=x && node[
      node[x].pa].ch[1]!=x;
}

inline void pull(int x) {
  node[x].mx = max(node[x].val, max(node[
      node[x].ch[0]].mx, node[node[x].ch
      [1]].mx));
}

inline void push(int x) {
  if(node[x].rev) {
    node[node[x].ch[0]].rev ^= 1;
    node[node[x].ch[1]].rev ^= 1;
    swap(node[x].ch[0], node[x].ch[1]);
    node[x].rev ^= 1;
  }
}

void push_all(int x) {
  if(!isroot(x))  push_all(node[x].pa);
  push(x);
}

inline void rotate(int x) {
  int y = node[x].pa, z = node[y].pa, d =
      node[y].ch[1]==x;
  node[x].pa = z;
  if(!isroot(y))  node[z].ch[node[z].ch
      [1]==y] = x;
  node[y].ch[d] = node[x].ch[d^1];
  node[node[x].ch[d^1]].pa = y;
  node[x].ch[!d] = y;
  node[y].pa = x;
  pull(y);
  pull(x);
}

void splay(int x) {
  push_all(x);
  while(!isroot(x)) {
    int y = node[x].pa;
    if(!isroot(y)) {
      int z = node[y].pa;
      if((node[z].ch[1]==y) ^ (node[y].ch
          [1]==x)) rotate(y);
      else  rotate(x);
    }
    rotate(x);
  }
}

inline int access(int x) {
  int last = 0;
  while(x) {
    splay(x);
    node[x].ch[1] = last;
    pull(x);
    last = x;
    x = node[x].pa;
  }
  return last;
}

inline void make_root(int x) {
  node[access(x)].rev ^= 1;
  splay(x);
}

inline void link(int x, int y) {
  make_root(x);
  node[x].pa = y;
}

inline void cut(int x, int y) {
  make_root(x);
  access(y);
  splay(y);
  node[y].ch[0] = 0;
  node[x].pa = 0;
}

inline void cut_parent(int x) {
```

```cpp
  x = access(x);
  splay(x);
  node[node[x].ch[0]].pa = 0;
  node[x].ch[0] = 0;
  pull(x);
}

inline int find_root(int x) {
  x = access(x);
  while(node[x].ch[0])  x = node[x].ch[0];
  splay(x);
  return x;
}

int find_mx(int x) {
  if(node[x].val == node[x].mx) return x;
  return node[node[x].ch[0]].mx==node[x].mx ? find_mx(node[x].ch[0]) : find_mx(node[x].ch[1]);
}

inline void change(int x,int b){
    splay(x);
    node[x].data=b;
    up(x);
}
inline int query_lca(int u,int v){
/*retrun: sum of weight of vertices on the chain (u->v)
sum: total weight of the subtree
data: weight of the vertex */
  access(u);
  int lca=access(v);
  splay(u);
  if(u==lca){
    return node[lca].data+node[node[lca].ch[1]].sum;
  }else{
    return node[lca].data+node[node[lca].ch[1]].sum+node[u].sum;
  }
}
```

6.7 Heavy Light Decomposition

```cpp
const int MAXN = 10000 + 10;

vector<PII> e[MAXN];
int val[MAXN];
int sz[MAXN], max_son[MAXN], p[MAXN], dep[MAXN];
int link[MAXN], link_top[MAXN], cnt;

void find_max_son(int u) {
  sz[u] = 1;
  max_son[u] = -1;
  for(int i=0; i<SZ(e[u]); i++) {
    PII tmp = e[u][i];
    int v = tmp.F;
    if(v == p[u]) continue;

    p[v] = u;
    dep[v] = dep[u]+1;
    val[v] = tmp.S;
    find_max_son(v);
    if(max_son[u]<0 || sz[v]>sz[ max_son[u] ])  max_son[u] = v;
    sz[u] += sz[v];
  }
}

void build_link(int u, int top) {
  link[u] = ++cnt;
  link_top[u] = top;
  if(max_son[u] > 0)  build_link(max_son[u], top);
  for(int i=0; i<SZ(e[u]); i++) {
    PII tmp = e[u][i];
    int v = tmp.F;
    if(v==p[u] || v==max_son[u])  continue;

    build_link(v, v);
  }
}

int query(int a, int b) {
  int res = -1;
  int ta = link_top[a], tb = link_top[b];
  while(ta != tb) {
    if(dep[ta] < dep[tb]) {
      swap(a, b);
      swap(ta, tb);
    }

    res = max(res, seg->qry(link[ta], link[a], 1, cnt));
    ta = link_top[a=p[ta]];
  }

  if(a != b) {
    if(dep[a] > dep[b]) swap(a, b);
    a = max_son[a];
    res = max(res, seg->qry(link[a], link[b], 1, cnt));
  }

  return res;
}
```

6.8 Disjoint Sets + offline skill

```cpp
const int MAXN = 300000 + 10;

bool q[MAXN];

struct DisJointSet {
  int p[MAXN], sz[MAXN], gps;
  vector<pair<int*, int> > h;
  VI sf;

  void init(int n) {
    for(int i=1; i<=n; i++) {
      p[i] = i;
      sz[i] = 1;
    }
    gps = n;
  }

  void assign(int *k, int v) {
    h.PB(MP(k, *k));
    *k = v;
  }

  void save() {
    sf.PB(SZ(h));
  }

  void load() {
    int last = sf.back(); sf.pop_back();
    while(SZ(h) != last) {
      auto x = h.back();  h.pop_back();
      *x.F = x.S;
    }
  }

  int find(int x) {
    return x==p[x] ? x : find(p[x]);
  }

  void uni(int x, int y) {
    x = find(x), y = find(y);
    if(x == y)  return ;
    if(sz[x] < sz[y]) swap(x, y);
    assign(&sz[x], sz[x]+sz[y]);
    assign(&p[y], x);
    assign(&gps, gps-1);
  }
} djs;

struct Seg {
  vector<PII> es;
  Seg *tl, *tr;

  Seg() {}
  Seg(int l, int r) {
    if(l == r)  tl = tr = NULL;
    else {
      int m = (l+r) / 2;
      tl = new Seg(l, m);
      tr = new Seg(m+1, r);
    }
  }

  void add(int a, int b, PII e, int l, int
      r) {
    if(a <= l && r <= b)  es.PB(e);
    else if(b < l || r < a) return ;
    else {
      int m = (l+r) / 2;
      tl->add(a, b, e, l, m);
      tr->add(a, b, e, m+1, r);
    }
  }

  void solve(int l, int r) {
    djs.save();
    for(auto p : es)  djs.uni(p.F, p.S);

    if(l == r) {
      if(q[l])  printf("%d\n", djs.gps);
    }
    else {
      int m = (l+r) / 2;
      tl->solve(l, m);
      tr->solve(m+1, r);
    }

    djs.load();
  }
};

map<PII, int> prv;

int main() {
  freopen("connect.in", "r", stdin);
  freopen("connect.out", "w", stdout);

  int n, k;
  scanf("%d%d\n", &n, &k);
  if(!k)  return 0;

  Seg *seg = new Seg(1, k);
  djs.init(n);
  for(int i=1; i<=k; i++) {
    char op = getchar();
    if(op == '?') {
      q[i] = true;
      op = getchar();
    }
    else {
      int u, v;
      scanf("%d%d\n", &u, &v);
      if(u > v) swap(u, v);
      PII eg = MP(u, v);
      int p = prv[eg];
      if(p) {
        seg->add(p, i, eg, 1, k);
        prv[eg] = 0;
      }
      else  prv[eg] = i;
    }
  }
  for(auto p : prv) {
    if(p.S) {
      seg->add(p.S, k, p.F, 1, k);
    }
  }

  seg->solve(1, k);

    return 0;
}
```

6.9 2D Segment Tree

```
struct Seg1D {
  Seg1D *tl, *tr;
  ll val;
  // ll tmp;
  //int _x, _y;
  Seg1D() :
    tl(NULL), tr(NULL), val(0), tmp(-1), _x
        (-1), _y(-1) {}
```

```cpp
  ll query1D(int x1, int x2, int y1, int y2
      , int l, int r) {
    /*
    if no Brian improvement, dont need to
        pass x1 and x2
    if(tmp >= 0) {
      if(x1<=_x&&_x<=x2 && y1<=_y&&_y<=y2)
            return tmp;
      else  return 0;
    }
    */
    if(y1 <= l && r <= y2)  return val;
    else if(r < y1 || y2 < l) return 0;
    else {
      int m = (l+r)/2;
      ll a = tl ? tl->query1D(x1, x2, y1,
          y2, l, m) : 0,
         b = tr ? tr->query1D(x1, x2, y1,
             y2, m+1, r) : 0;
      return gcd(a, b);
    }
  }
  void update1D(int x, int y, ll num, int l
      , int r) {
    if(l == r) {
      val = num;
      return ;
    }
    /*
    if(tmp < 0 && !tl && !tr) {
      tmp = val = num;
      _x = x;
      _y = y;
      return ;
    }
    else if(tmp >= 0) {
      int m = (l+r)/2;
      if(_y <= m) {
        if(!tl) tl = new Seg1D();
        tl->update1D(_x, _y, tmp, l, m);
      }
      else {
        if(!tr) tr = new Seg1D();
        tr->update1D(_x, _y, tmp, m+1, r);
      }
      tmp = _x = _y = -1;
    }*/
    int m = (l+r)/2;
    if(y <= m) {
      if(!tl) tl = new Seg1D();
      tl->update1D(x, y, num, l, m);
    }
    else {
      if(!tr) tr = new Seg1D();
      tr->update1D(x, y, num, m+1, r);
    }
    ll a = tl ? tl->val : 0;
    ll b = tr ? tr->val : 0;
    val = gcd(a, b);
  }
};
struct Seg2D {
  Seg2D *tl, *tr;
  Seg1D *t2;
  Seg2D() :
    tl(NULL), tr(NULL), t2(NULL) {}
  ll query2D(int x1, int x2, int y1, int y2
      , int l, int r) {
    if(x1 <= l && r <= x2) {
      if(!t2) t2 = new Seg1D();
      return t2->query1D(x1, x2, y1, y2, 0,
          C-1);
    }
    else if(x2 < l || r < x1) return 0;
    else {
      int m = (l+r)/2;
      ll a = tl ? tl->query2D(x1, x2, y1,
          y2, l, m) : 0,
         b = tr ? tr->query2D(x1, x2, y1,
             y2, m+1, r) : 0;
      return gcd(a, b);
    }
  }
  void update2D(int x, int y, ll num, int l
      , int r) {
    int m = (l+r)/2;
    if(l == r) {
      if(!t2) t2 = new Seg1D();
      t2->update1D(x, y, num, 0, C-1);
      return ;
    }
    if(x <= m) {
      if(!tl) tl = new Seg2D();
      tl->update2D(x, y, num, l, m);
    }
    else {
      if(!tr) tr = new Seg2D();
      tr->update2D(x, y, num, m+1, r);
    }
    if(!tl) tl = new Seg2D();
    if(!tr) tr = new Seg2D();
    ll a = tl->t2 ? tl->t2->query1D(l, m, y
        , y, 0, C-1) : 0,
       b = tr->t2 ? tr->t2->query1D(m+1, r,
           y, y, 0, C-1) : 0;
    if(!t2) t2 = new Seg1D();
    t2->update1D(x, y, gcd(a, b), 0, C-1);
  }
};
```

7 geometry

7.1 Basic

```cpp
const double PI = acos(-1);
const double INF = 1e18;
const double EPS = 1e-8;

struct node {
  double x,y;
  node(double _x=0, double _y=0) : x(_x),y(
      _y) {}
  node operator+(const node& rhs) const
    { return node(x+rhs.x, y+rhs.y); }
  node operator-(const node& rhs) const
    { return node(x-rhs.x, y-rhs.y); }
  node operator*(const double& rhs) const
```

```cpp
    { return node(x*rhs, y*rhs); }
  node operator/(const double& rhs) const
    { return node(x/rhs, y/rhs); }
  double operator*(const node& rhs) const
    { return x*rhs.x+y*rhs.y; }
  double operator^(const node& rhs) const
    { return x*rhs.y-y*rhs.x; }
  double len2() const { return x*x+y*y; }
  double len() const { return sqrt(x*x+y*y); }
  node unit() const { return *this/len(); }
  node T() const { return node(-y,x); } // counter-clockwise
  node TR() const { return node(y,-x); } // clockwise
  node rot(double rad) const { // rotate counter-clockwise in rad
    return node(cos(rad)*x-sin(rad)*y, sin(rad)*x+cos(rad)*y);
  }
};

node __mirror(node normal, double constant, node point){ //2D3D
  double scale=(normal*point+constant)/(normal*normal);
  return point-normal*(2*scale);
}
node mirror(node p1, node p2, node p3){ // 2D3D
  return __mirror((p2-p1).T(), (p2-p1).T()*p1*(-1), p3);
}
double ori(const node& p1, const node& p2, const node& p3){ //□□□□□□□(□□□)
  return (p2-p1)^(p3-p1);
}
bool intersect(const node& p1, const node& p2, const node& p3, const node& p4){
  return (ori(p1,p2,p3)*ori(p1,p2,p4)<0 && ori(p3,p4,p1)*ori(p3,p4,p2)<0);
}
pair<node,node> two_circle_intersect(node p1, double r1, node p2, double r2){
  double degree=acos(((p2-p1).len2()+r1*r1-r2*r2)/(2*r1*(p2-p1).len()));
  return make_pair(p1+(p2-p1).unit().rot(degree)*r1, p1+(p2-p1).unit().rot(-degree)*r1);
}
node intersectionPoint(node p1, node p2, node p3, node p4){
  double a123 = (p2-p1)^(p3-p1);
  double a124 = (p2-p1)^(p4-p1);
  return (p4*a123-p3*a124)/(a123-a124);
}
node inter(const node &p1, const node &v1, const node &p2, const node &v2) // intersection
{
  if(fabs(v1^v2) < EPS)
    return node(INF, INF);
  double k = ((p2-p1)^v2) / (v1^v2);
  return p1 + v1*k;
}
void CircleInter(node o1, double r1, node o2, double r2) {
  if(r2>r1)
    swap(r1, r2), swap(o1, o2);
  double d = (o2-o1).len();
  node v = (o2-o1).unit();
  node t = v.TR();

  double area;
  vector<node> pts;
  if(d > r1+r2+EPS)
    area = 0;
  else if(d < r1-r2)
    area = r2*r2*PI;
  else if(r2*r2+d*d > r1*r1){
    double x = (r1*r1 - r2*r2 + d*d) / (2*d);
    double th1 = 2*acos(x/r1), th2 = 2*acos((d-x)/r2);
    area = (r1*r1*(th1 - sin(th1)) + r2*r2*(th2 - sin(th2))) / 2;
    double y = sqrt(r1*r1 - x*x);
    pts.PB(o1 + v*x + t*y), pts.PB(o1 + v*x - t*y);
  } else {
    double x = (r1*r1 - r2*r2 - d*d) / (2*d);
    double th1 = acos((d+x)/r1), th2 = acos(x/r2);
    area = r1*r1*th1 - r1*d*sin(th1) + r2*r2*(PI-th2);
    double y = sqrt(r2*r2 - x*x);
    pts.PB(o2 + v*x + t*y), pts.PB(o2 + v*x - t*y);
  }
  //Area: area
  //Intersections: pts
}
```

7.2 Smallist circle problem

```cpp
const int N = 1000000 + 10;

struct PT {
  double x, y;

  PT() {}
  PT(double x, double y):
    x(x), y(y) {}
  PT operator+(const PT &b) const {
    return (PT) {x+b.x, y+b.y};
  }
  PT operator-(const PT &b) const {
    return (PT) {x-b.x, y-b.y};
  }
  PT operator*(const double b) const {
    return (PT) {x*b, y*b};
  }
  PT operator/(const double b) const {
    return (PT) {x/b, y/b};
  }
  double operator%(const PT &b) const {
    return x*b.y - y*b.x;
  }

  double len() const {
    return sqrt(x*x + y*y);
  }
  PT T() const {
    return (PT) {-y, x};
  }
} p[N];

void update(PT a, PT b, PT c, PT &o, double
     &r) {
  if(c.x < 0.0) o = (a+b) / 2.0;
  else {
    PT p1 = (a+b)/2.0, p2 = p1 + (b-a).T();
    PT p3 = (a+c)/2.0, p4 = p3 + (c-a).T();
    double a123 = (p2-p1)%(p3-p1), a124 = (
        p2-p1)%(p4-p1);
    if(a123 * a124 > 0.0) a123 = -a123;
    else  a123 = abs(a123), a124 = abs(a124
        );
    o = (p4*a123 + p3*a124) / (a123 + a124)
        ;
  }
  r = (a-o).len();
}

int main() {
  srand(7122);

  int m, n;
  while(scanf("%d%d", &m, &n)) {
    if(!n && !m)  return 0;

    for(int i = 0; i < n; i++)  scanf("%lf%
        lf", &p[i].x, &p[i].y);

    for(int i = 0; i < n; i++)
      swap(p[i], p[rand() % (i+1)]);

    PT a = p[0], b = p[1], c(-1.0, -1.0), o
         = (a+b) / 2.0;
    double r = (a-o).len();
    for(int i = 2; i < n; i++) {
      if((p[i]-o).len() <= r) continue;

      a = p[i];
      b = p[0];
      c = (PT) {-1.0, -1.0};
      update(a, b, c, o, r);
      for(int j = 1; j < i; j++) {
        if((p[j]-o).len() <= r) continue;

        b = p[j];
        c = (PT) {-1.0, -1.0};
        update(a, b, c, o, r);

        for(int k = 0; k < j; k++) {
          if((p[k]-o).len() <= r) continue;

          c = p[k];
          update(a, b, c, o, r);
        }
      }
    }

    printf("%.3f\n", r);
  }
}
```

8 Others

8.1 Random

```
const int seed=1;

mt19937 rng(seed);
int randint(int lb,int ub) { // [lb, ub]
  return uniform_int_distribution<int>(lb,
      ub)(rng);
}
```

8.2 Fraction

```
struct Frac {
  ll a,b; //  a/b
  void relax() {
    ll g=__gcd(a,b);
    if(g!=0 && g!=1)
      a/=g, b/=g;
    if(b<0)
      a*=-1, b*=-1;
  }
  Frac(ll a_=0,ll b_=1): a(a_), b(b_) {
    relax();
  }
  Frac operator + (Frac x) {
    relax();
    x.relax();
    ll g=__gcd(b,x.b);
    ll lcm=b/g*x.b;
    return Frac(a*(lcm/b)+x.a*(lcm/x.b),lcm
        );
  }
  Frac operator - (Frac x) {
    relax();
    x.relax();
    Frac t=x;
    t.a*=-1;
    return *this+t;
  }
  Frac operator * (Frac x) {
    relax();
    x.relax();
    return Frac(a*x.a,b*x.b);
  }
  Frac operator / (Frac x) {
    relax();
    x.relax();
    Frac t=Frac(x.b,x.a);
    return (*this)*t;
  }
  bool operator < (Frac x) {
    ll lcm=b/__gcd(b,x.b)*x.b;
    return ( (lcm/b)*a < (lcm/x.b)*x.a );
  }
};
```