Contents

Ad hoc

Go up for ultras

```
#include <cstdio>
#include <cmath>
#include <cstring>
#include <algorithm>

using namespace std;
int v[100005];
int seg[4000005];
int seg2[4000005];

int esquerda(int p){
   return 2 * p;
}

int direita(int p){
   return 2 * p + 1;
}

int query(int p, int i, int j, int L, int R){
   if(j < L || i > R) return 1000000000;
   if(i <= L && j >= R) return seg[p];
   int meio = (L + R) / 2;
   return min(query(esquerda(p), i, j, L, meio),
      query(direita(p), i, j, meio + 1, R));
}

int findL(int p, int i, int L, int R, int v){
   if(i <= L) return -1;
   if(v >= seg2[p]) return -1;
   if(L == R) return L;

   int meio = (L + R) / 2;
   int ret = -1;
   if(i > meio+1 && seg2[direita(p)] > v)
      ret = findL(direita(p), i, meio+1, R, v);
   if(ret != -1) return ret;
   return findL(esquerda(p), i, L, meio, v);
}

int findR(int p, int i, int L, int R, int v){
   if(i > R) return -1;
   if(v >= seg2[p]) return -1;
   if(L == R) return L;

   int meio = (L + R) / 2;
   int ret = -1;
   if(i < meio && seg2[esquerda(p)] > v)
      ret = findR(esquerda(p), i, L, meio, v);
   if(ret != -1) return ret;
   return findR(direita(p), i, meio+1, R, v);
}
int constructMax(int L, int R, int p){
   if(L == R) return seg2[p] = v[L];
   int mid = (L + R) / 2;
   return seg2[p] = max(constructMax(L, mid, esquerda(p)),
      constructMax(mid+1, R, direita(p)));
}

int constructMin(int L, int R, int p){
   if(L == R) return seg[p] = v[L];
   int mid = (L + R) / 2;
   return seg[p] = min(constructMin(L, mid, esquerda(p)),
      constructMin(mid+1, R, direita(p)));
}

int main(){
   int n;

   while(scanf("%d", &n) == 1){
      memset(seg, 0, sizeof(seg));
      memset(seg2, 0, sizeof(seg2));

      for(int i = 0; i < n; i++)
         scanf("%d", v + i);

      constructMax(0, n-1, 1);
      constructMin(0, n-1, 1);

      bool first = true, ultra, left, right;
      for(int i = 0; i < n; i++){
         ultra = false;
         if((i == 0 || v[i] > v[i-1]) && (i == n-1 || v[i] > v[i+1])){
            int id = findL(1, i, 0, n-1, v[i]);
            int id2 = findR(1, i, 0, n-1, v[i]);
            if(id == -1 && id2 == -1){
               if(v[i] >= 150000)
                  ultra = true;
            }
            else{
               left = false;
               if(id == -1) left = true;
               else{
                  int d = query(1, id+1, i-1, 0, n-1);
                  if(v[i] - d >= 150000) left = true;
               }
               right = false;
               if(id2 == -1) right = true;
               else{
                int d = query(1, i+1, id2-1, 0, n-1);
                if(v[i] - d >= 150000) right = true;
             }
             if(left && right){
                ultra = true;
             }
          }
          if(ultra){
             if(!first) printf("␣");
             printf("%d", i+1);

             first = false;
          }
       }
    }
    printf("\n");
 }

 return 0;
}
```

Estrutura de Dados

Consultas Horríveis

```
#include <stdio.h>
#include <string.h>

long long st[800004];
long long troca[800004];

void update(int p, int ini, int fim, int b, int e, long long valor){
   st[p] += (troca[p] * (fim - ini + 1));
   troca[(p << 1)] += troca[p];
   troca[(p << 1) + 1] += troca[p];
   troca[p] = 0;

   if(b > fim || e < ini) return;

   st[p] += ((fim < e ? fim : e) - (ini > b ? ini : b) + 1) * valor;

   if(b <= ini && e >= fim){
      troca[(p << 1)] += valor;
      troca[(p << 1) + 1] += valor;
      return;
   }

   int meio = (ini + fim) >> 1;
   if(b <= meio) update((p << 1), ini, meio, b, e, valor);
   if(e > meio) update((p << 1) + 1, meio+1, fim, b, e, valor);
}

long long query(int p, int ini, int fim, int b, int e){
   st[p] += (troca[p] * (fim - ini + 1));
   troca[(p << 1)] += troca[p];
   troca[(p << 1) + 1] += troca[p];

   troca[p] = 0;

   if(b > fim || e < ini) return 0;
   if(b <= ini && e >= fim) return st[p];

   int meio = (ini + fim) >> 1;

   return query((p << 1), ini, meio, b, e) + query((p << 1) + 1, meio+1, fim, b, e);
}

int main(){
   int t, n, c, i, p, q, v;

   scanf("%d", &t);
   while(t--){
      scanf("␣%d␣%d", &n, &c);

      memset(st, 0, sizeof(st));
      memset(troca, 0, sizeof(troca));
      while(c--){
         scanf("␣%d␣%d␣%d", &i, &p, &q);
         if(!i){
            scanf("␣%d", &v);
            update(1, 0, n-1, p-1, q-1, v);
         }
         else printf("%lld\n", query(1, 0, n-1, p-1, q-1));
      }
   }

   return 0;
}
```

Contra Ataque Ramsay

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <list>
#include <vector>

#define LADO 800
#define SEG_LADO 1400000
```

```
using namespace std;

class Sgtree{
    private:
        int arv[SEG_LADO];
    public:
        int query(int L, int R, int D, int U, int l, int r, int d, int u, int p);
        void update(int L, int R, int D, int U, int pos_x, int pos_y, int num, int p);
        Sgtree(){memset(arv, 0, SEG_LADO<<2);};
}tipo_segtree;


typedef struct _sold{
    int x,y;
    bool pai;
}tipo_soldado;


int Sgtree::query(int L, int R, int D, int U, int l, int r, int d, int u, int p) {
    if(r < L || l > R || d > U || u < D)
        return 0;
    if(L >= l && R <= r && U <= u && D >= d){
        return arv[p];
    }
    int tmp1, tmp2, tmp3, tmp4;
    tmp1 = query(L, (L+R)>>1, D, (U+D)>>1, l, r, d, u, (p<<2)+1);
    tmp2 = query(L, (L+R)>>1, ((U+D)>>1) + 1, U, l, r, d, u, (p<<2)+2);
    tmp3 = query(((L+R)>>1) + 1, R, D, (U+D)>>1, l, r, d, u, (p<<2)+3);
    tmp4 = query(((L+R)>>1) + 1, R, ((U+D)>>1) + 1, U, l, r, d, u, (p<<2)+4);

    tmp1 = max(tmp1, tmp2);
    tmp1 = max(tmp1, tmp3);
    return max(tmp1, tmp4);
}

void Sgtree::update(int L, int R, int D, int U, int pos_x, int pos_y, int num, int p){
    if(L == R && U == D){
        arv[p] = num;
    }
    else{
        if(pos_x <= (L+R)>>1){
            if(pos_y <= (U+D)>>1)
                update(L, (L+R)>>1, D, (U+D)>>1, pos_x, pos_y, num, (p<<2)+1);
            else
                update(L, (L+R)>>1, ((U+D)>>1) + 1, U, pos_x, pos_y, num, (p<<2)+2);
        }
        else{
            if(pos_y <= (U+D)>>1)
                update(((L+R)>>1) + 1, R, D, (U+D)>>1, pos_x, pos_y, num, (p<<2)+3);
            else
                update(((L+R)>>1) + 1, R, ((U+D)>>1) + 1, U, pos_x, pos_y, num, (p<<2)+4);
```

```
        }
        int tmp1, tmp2;
        tmp1 = max(arv[(p<<2)+1], arv[(p<<2)+2]);
        tmp2 = max(arv[(p<<2)+3], arv[(p<<2)+4]);
        arv[p] = max(tmp1, tmp2);
    }
}

vector<int> G[50001];
Sgtree *tree;
tipo_soldado soldado[50001];
bool is_root[50001];
int ans = 1;
void dfs(int no){
    int tmp = tree->query(0, LADO, 0, LADO, 0 ,soldado[no].x-1, 0, soldado[no].y-1, 0
    tmp++;
    tree->update(0, LADO, 0, LADO, soldado[no].x, soldado[no].y, tmp, 0);

    ans = max(ans, tmp);
    for(int j : G[no])
        dfs(j);
    tree->update(0, LADO, 0, LADO, soldado[no].x, soldado[no].y, 0, 0);
}


int main() {
    int N, M, i, j, no_pai, no_filho;
    tree = new Sgtree();

    scanf("%d␣%d", &N, &M);
    for(i = 1; i <= N; i++){
        scanf("%d", &soldado[i].x);
        scanf("%d", &soldado[i].y);
        soldado[i].x += 400;
        soldado[i].y += 400;
    }

    for(i = 1; i <= N; i++) is_root[i] = 1;
    for(i = 1; i <= M; i++){
        scanf("%d␣%d", &no_filho, &no_pai);
        G[no_pai].push_back(no_filho);
        is_root[no_filho] = 0;
    }

    for(i = 1; i <= N; i++){
        if(!is_root[i]) continue;
        if(!G[i].empty())
            dfs(i);
    }
    printf("%d\n", ans);
    return 0;
}
```

Sparse Table

```
#include <bits/stdc++.h>

#define mp make_pair
#define ff first
#define ss second

using namespace std;

typedef long long ll;

int st1[200005][20], st2[200005][20];

void build(int n){
   int i,j;
   for(j = 1; 1<<j <= n; j++){
      for(i = 0; i +(1<<j) <= n; i++){
         st1[i][j] = max(st1[i][j-1],st1[i+(1<<(j-1))][j-1]);
         st2[i][j] = min(st2[i][j-1],st2[i+(1<<(j-1))][j-1]);
      }
   }
}

int getA(int l, int r){
   int num = r-l+1,k=0;
   while(num>=2) num/=2, k++;
   return max(st1[l][k], st1[r-(1<<k)+1][k]);
}

int getB(int l, int r){
   int num = r-l+1,k=0;
   while(num>=2) num/=2, k++;
   return min(st2[l][k], st2[r-(1<<k)+1][k]);
}

int bsearch1(int i, int n){
   int mid,l=i-1,r=n;
   while(l+1 < r){
      mid = (l+r)>>1;
      int a = getA(i, mid);
      int b = getB(i, mid);
      if(a < b) l = mid;
      else r = mid;
   }
   return r;
}

int bsearch2(int i, int n){
   int mid,l=i-1,r=n;
   while(l+1 < r){
      mid = (l+r)>>1;
      int a = getA(i, mid);
      int b = getB(i, mid);
      if(a <= b) l = mid;
      else r = mid;
   }
   return r;
}

int main(){
   int n,i,j;

   scanf("%d", &n);
   for(i = 0; i < n; i++) scanf("%d", &st1[i][0]);
   for(i = 0; i < n; i++) scanf("%d", &st2[i][0]);
   build(n);
   ll ans = 0;
   for(i = 0; i < n; i++){
      int x = bsearch1(i,n);
      int y = bsearch2(i,n);
      if(y > x)
         ans += (y-x);
   }
   printf("%lld\n", ans);

   return 0;
}
```

Paradigmas

Garota Hiperativa

```
#include <cstdio>
#include <cstring>
#include <utility>
#include <algorithm>
#define MOD (100000000)
using namespace std;

int n, m, dp[104][104];
pair <int, int> activity[105];

int f(int prev, int next){
   if(dp[prev][next] != -1) return dp[prev][next];

   if(activity[next].second == m) return 1;
   int aux = 0;
   for(int i = next+1; i < n; i++)
      if(activity[i].first > activity[next].first &&
         activity[i].first <= activity[next].second &&
         activity[i].second > activity[next].second &&
         (prev == next || activity[i].first > activity[prev].second))
         aux = (aux + f(next, i)) % MOD;

   return dp[prev][next] = aux;
}

int main(){

   while(scanf("%d␣%d", &m, &n) == 2 && n + m){
      for(int i = 0; i < n; i++)
         scanf("%d␣%d", &activity[i].first, &activity[i].second);

      memset(dp, -1, sizeof dp);
      sort(activity, activity+n);

      int ans = 0;
      for(int i = 0; i < n && activity[i].first == 0; i++)
         ans = (ans + f(i, i)) % MOD;

      printf("%d\n", ans);
   }

   return 0;
}
```

Hard Problem

```
#include <bits/stdc++.h>

#define oo 1000000000000000000

using namespace std;

string s[100005];
string r[100005];
int n, c[100005];

long long dp[100005][2];
bool mark[100005][2];

long long f(int x, int rev){
      if(mark[x][rev]) return dp[x][rev];
      mark[x][rev] = 1;

      if(x == n) return 0;
      if(x == 0) return min(c[x] + f(x+1, 1), f(x+1, 0));

      bool canforward = false, canreverse = false;
      long long ans = oo;

      if(rev){
            if(s[x] >= r[x-1]) canforward = true;
            if(r[x] >= r[x-1]) canreverse = true;
      }
      else{
            if(s[x] >= s[x-1]) canforward = true;
            if(r[x] >= s[x-1]) canreverse = true;
      }

      if(canforward) ans = min(ans, f(x+1, 0));
      if(canreverse) ans = min(ans, c[x]+f(x+1, 1));

      return dp[x][rev] = ans;
}

int main(){

      scanf("%d", &n);

      for(int i = 0; i < n; i++)
            scanf("%d", c+i);

      for(int i = 0; i < n; i++){
            cin >> s[i];
            r[i] = s[i];
            reverse(r[i].begin(), r[i].end());
      }

      long long ret = f(0, 0);
      printf("%lld\n", ret == oo ? -1 : ret);

      return 0;
}
```

Cartões

```
#include <stdio.h>

long long max(long long a, long long b){
   return (a > b ? a : b);
}
long long min(long long a, long long b){
   return (a < b ? a : b);
}
```

```
int main(){
   int n, i, j;
   int cards[10005];
   long long dp[10005];

   while(scanf("%d", &n) == 1){
      for(i = 0; i < n; i++)
         scanf("%d", cards + i);

      for(i = 0; i < n; i++)
         dp[i] = max(cards[i], cards[i+1]);

      for(j = 3; j <= n; j++)
         if(j & 1)
            for(i = 0; i < n; i++)
               dp[i] = min(dp[i], dp[i+1]);
         else
            for(i = 0; i < n; i++)
               dp[i] = max(cards[i+j-1] + dp[i], cards[i] + dp[i+1]);

      printf("%lld\n", dp[0]);
   }

   return 0;
}
```

FFT

```
typedef complex<double> base;

void fft (vector<base> & a, bool invert) {
      int n = (int) a.size();
      if (n == 1) return;

      vector<base> a0 (n/2), a1 (n/2);
      for (int i=0, j=0; i<n; i+=2, ++j) {
            a0[j] = a[i];
            a1[j] = a[i+1];
      }
      fft (a0, invert);
      fft (a1, invert);

      double ang = 2*PI/n * (invert ? -1 : 1);
      base w (1), wn (cos(ang), sin(ang));
      for (int i=0; i<n/2; ++i) {
            a[i] = a0[i] + w * a1[i];
            a[i+n/2] = a0[i] - w * a1[i];
            if (invert)
                  a[i] /= 2, a[i+n/2] /= 2;
            w *= wn;
      }
}
void multiply (const vector<int> & a, const vector<int> & b, vector<int> & res) {
      vector<base> fa (a.begin(), a.end()), fb (b.begin(), b.end());
      size_t n = 1;
      while (n < max (a.size(), b.size())) n <<= 1;
      n <<= 1;
      fa.resize (n), fb.resize (n);

      fft (fa, false), fft (fb, false);
      for (size_t i=0; i<n; ++i)
            fa[i] *= fb[i];
      fft (fa, true);

      res.resize (n);
      for (size_t i=0; i<n; ++i)
            res[i] = int (fa[i].real() + 0.5);
}
```

FFT iterativa

```
typedef complex<double> base;

void fft (vector<base> & a, bool invert) {
      int n = (int) a.size();

      for (int i=1, j=0; i<n; ++i) {
            int bit = n >> 1;
            for (; j>=bit; bit>>=1)
                  j -= bit;
            j += bit;
            if (i < j)
                  swap (a[i], a[j]);
      }

      for (int len=2; len<=n; len<<=1) {
            double ang = 2*PI/len * (invert ? -1 : 1);
            base wlen (cos(ang), sin(ang));
            for (int i=0; i<n; i+=len) {
                  base w (1);
                  for (int j=0; j<len/2; ++j) {
                        base u = a[i+j], v = a[i+j+len/2] * w;
                        a[i+j] = u + v;
                        a[i+j+len/2] = u - v;
                        w *= wlen;
                  }
            }
      }
      if (invert)
            for (int i=0; i<n; ++i)
                  a[i] /= n;
}
```

NTT

```
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1<<20;

void fft (vector<int> & a, bool invert) {
      int n = (int) a.size();

      for (int i=1, j=0; i<n; ++i) {
            int bit = n >> 1;
            for (; j>=bit; bit>>=1)
                  j -= bit;
            j += bit;
            if (i < j)
                  swap (a[i], a[j]);
      }

      for (int len=2; len<=n; len<<=1) {
            int wlen = invert ? root_1 : root;
            for (int i=len; i<root_pw; i<<=1)
                  wlen = int (wlen * 1ll * wlen % mod);
            for (int i=0; i<n; i+=len) {
                  int w = 1;
                  for (int j=0; j<len/2; ++j) {
                        int u = a[i+j], v = int (a[i+j+len/2] * 1ll * w % mod);
                        a[i+j] = u+v < mod ? u+v : u+v-mod;
                        a[i+j+len/2] = u-v >= 0 ? u-v : u-v+mod;
                        w = int (w * 1ll * wlen % mod);
                  }
            }
      }
      if (invert) {
            int nrev = reverse (n, mod);
            for (int i=0; i<n; ++i)
                  a[i] = int (a[i] * 1ll * nrev % mod);
      }
}
```

Matemática

Quantos zeros e quantos digitos?

```
#include <bits/stdc++.h>

using namespace std;

int main(){
      vector<pair<int,int>> v(1000);

      for(int i = 1; i < 1000; i++){
            pair<int, int> last = {1, 1};
            int tmp = i;
            for(int j = 2; j*j <= tmp; j++){
                  if(tmp % j == 0){
                        last = {j, 0};
                        while(tmp % j == 0){
                              tmp /= j;
                              last.second++;
                        }
                  }
            }
            if(tmp > 1){
                  last = {tmp, 1};
            }
            v[i] = last;
      }

      int n, b;

      while(scanf("%d␣%d", &n, &b) == 2){
            double tmp1 = 0;
            for(int i = 1; i <= n; i++)
                  tmp1 += log(i);
            int digits = 1 + (int)(tmp1 / log(b) + 1e-9);

            int tmp = v[b].first, count = 0;
            while(tmp <= n){
                  count += n / tmp;
                  tmp *= v[b].first;
            }

            int zeros = count / v[b].second;

            printf("%d␣%d\n", zeros, digits);
      }


      return 0;
}
```

RSA com Euclides Extendido

```
#include <bits/stdc++.h>

#define ff first
#define ss second
#define mp make_pair
```

```
using namespace std;

int pot(int base, int exp, int mod){
	int resp = 1;

	while(exp){
		if(exp & 1)
			resp = ((long long)resp * base) % mod;
		base = ((long long)base * base) % mod;

		exp >>= 1;
	}
	return resp;
}

int totiente(int n){
	long long i;
	int ans = 1, tmp;
	for(i = 2; i*i <= n; i++){
		if(n % i == 0){
			tmp = 0;
			ans *= i-1;
			while(n % i == 0){ n /= i; tmp++; }
			ans *= pot(i, tmp-1, n);
		}
	}

	if(n > 1){
		ans *= n-1;
	}

	return ans;
}

pair<int, pair<int, int> > euclidesExt(int a, int b) {
	if(b == 0) return mp(a, mp(1, 0));

	pair<int, pair<int, int> > ret = euclidesExt(b, a%b);
	pair<int, int> tmp = ret.ss;

	ret.ss.ff = tmp.ss;
	ret.ss.ss = tmp.ff - tmp.ss*(a/b);

	return ret;
}

// a*b = 1 (mod c) <-> a*b + c*k = 1
int invmult(int a, int b){
	return (euclidesExt(a, b).ss.ff + b) % b;
}

int main(){
	int n, e, c, tot;

	scanf("%d␣%d␣%d", &n, &e, &c);

	tot = totiente(n);

	int d = invmult(e, tot);

	int m = pot(c, d, n);

	printf("%d\n", m);

	return 0;
}
```

Grafos

Floid Fill

Lazy Painting

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <algorithm>
#define ff first
#define ss second

using namespace std;

vector<vector<bool> > mat;
vector<vector<int> > prox;

int n, m, h, w, q, r, c, total;

int dx[] = {-1, 0, 0, 1};
int dy[] = {0, 1, -1, 0};

void dfs(int x, int y){
	if(mat[x][y] == true) return;
	total--;
	mat[x][y] = true;

	int a, b;
	for(int i = 0; i < 4; i++){
		a = dx[i] + x;
		b = dy[i] + y;
		if(a < r || a >= r+h || b < c || b >= c+w) continue;
```

```
            dfs(a, b);
        }
}



int main(){

        scanf("%d␣%d␣%d␣%d␣%d", &n, &m, &h, &w, &q);
        mat.assign(n, vector<bool> (m, false));
        prox.assign(n, vector<int> (m, 1));

        total = n*m;

        for(int i = 0; i < q; i++){
                scanf("%d␣%d", &r, &c); r--; c--;
                for(int j = r; j < r+h; ){
                        dfs(j, c);
                        int tmp = prox[j][c];
                        prox[j][c] = max(prox[j][c], r+h - j);
                        j += tmp;
                }
                printf("%d\n", total);
        }


        return 0;
}
```

LCA

Colonia

```
#include <bits/stdc++.h>

#define mp make_pair
#define ff first
#define ss second

using namespace std;

typedef long long ll;

vector<pair<int,int>> G[100005];
int lvl[100005], p[100005], anc[100005][20], logg;
ll dist[100005];

void dfs(int v, int l, ll d){
    lvl[v] = l;
    dist[v] = d;
    for(pair<int,int> &x : G[v]){
        if(lvl[x.ff] == -1){
            p[x.ff] = v;
            dfs(x.ff, l+1,d+x.ss);
        }
    }
}

int LCA(int u, int v){
    if(lvl[u] < lvl[v]) swap(u,v);

    for(int i = logg; i >= 0; i--)
        if(lvl[u] - (1<<i) >= lvl[v])
            u = anc[u][i];

    if(u == v) return u;

    for(int i = logg; i >= 0; i--){
        if(anc[u][i] != -1 && anc[u][i] != anc[v][i]){
            u = anc[u][i];
            v = anc[v][i];
        }
    }

    return anc[u][0];
}

int main() {
    int a,b,n,q,i,j;

        while(scanf("%d", &n),n){
                for(int i = 1; i <= n-1; i++){
                        scanf("%d%d", &a,&b);
                        G[i].push_back(mp(a,b));
                        G[a].push_back(mp(i,b));
                        lvl[i] = -1;
                }
                dfs(0,0,0);
                for(logg = 0; 1<<logg <= n; logg++);
                for(i = 0; i < n; i++)
                        for(j = 0; j < logg; j++)
                          anc[i][j] = -1;
                for(i = 0; i < n; i++)
                        anc[i][0] = p[i];
                for(j = 1; j <= logg; j++)
                        for(i = 0; i < n; i++)
                                if(anc[i][j-1] != -1)
                                        anc[i][j] = anc[ anc[i][j-1] ][j-1];
                scanf("%d", &q);
                while(q--){
                        scanf("%d%d", &a,&b);
                        int x = LCA(a,b);
                        ll ans = dist[a]+dist[b]-2*dist[x];
                if(q) printf("%lld␣", ans);
                else printf("%lld", ans);
            }
            printf("\n");
            for(i = 0; i < n; i++) G[i].clear();
        }

   return 0;
}
```

Jogo da Memoria

```
#include <bits/stdc++.h>

using namespace std;

#define ff first
#define ss second
#define pb push_back

const int oo = 1000000000;

typedef long long ll;
typedef pair<int,int> pii;
typedef vector<pii> vii;
typedef vector<int> vi;
typedef vector<vi> vvi;

int p[50005], L[50005], vis[50005], n, root;
vvi g;

int dfs(int u){
      vis[u] = 1;
      for(int i = 0; i < g[u].size(); i++){
            if(vis[ g[u][i] ] == 0){
                  p[ g[u][i] ] = u;
                  L[ g[u][i] ] = L[u]+1;
                  dfs(g[u][i]);
            }
      }
}

int dp[50005][20];

int f(int u, int level){
      if(dp[u][level] != -1) return dp[u][level];

      if(level == 0) return p[u];

      return dp[u][level] = f(f(u, level-1), level-1);
}

int lg(int n){
      int ans = 0;
      while(n)
            ans++, n/=2;
      return ans;
}

int LCA(int u, int v){
      if(L[u] < L[v]) swap(u, v);
      int k = lg(L[u]);

      for(int k = lg(L[u]); k >= 0; k--)
            if(L[ f(u, k) ] >= L[v])
                  u = f(u, k);

      if(u == v) return u;

      for(int k = lg(L[u]); k >= 0; k--){
            if(f(u, k) != f(v, k)){
                  u = f(u, k);
                  v = f(v, k);
            }
      }

      return p[u];
}


int main(){
      int a, b;
      scanf("%d", &n);

      vvi c(n+1);
      for(int i = 1; i <= n; i++){
            scanf("%d", &a);
            c[a].pb(i);
      }

      g.assign(n+1, vi());
      for(int i = 1; i < n; i++){
            scanf("%d␣%d", &a, &b);
            g[a].pb(b);
            g[b].pb(a);
      }

      memset(vis, 0, sizeof vis);
      root = 1;
      p[root] = root;
      L[root] = 0;
      dfs(root);
      memset(dp, -1, sizeof dp);

      int ans = 0;
      for(int i = 1; i <= n/2; i++)
            ans += L[ c[i][0] ] + L[ c[i][1] ] - 2*L[ LCA(c[i][0], c[i][1]) ];

      printf("%d\n", ans);
}
```

Query on Tree II

```
#include <bits/stdc++.h>

using namespace std;
#define MAXL 18
#define mp make_pair
#define ff first
#define ss second

typedef long long ll;
typedef pair<int,int> ii;

int p[10100], L[10100], vis[10100], dp[10100][MAXL], pw[10100], dp2[10100][MAXL];

vector<vector<ii> > g;

void dfs(int u){
      vis[u] = 1;

      for(ii &v : g[u]){
            if(!vis[v.ff]){
                  p[v.ff] = u;
                  L[v.ff] = L[u]+1;
                  pw[v.ff] = v.ss;
                  dfs(v.ff);
            }
      }
}

int f(int u, int j){
      if(j == 0) return p[u];
      if(dp[u][j] != -1) return dp[u][j];
      return dp[u][j] = f(f(u, j-1), j-1);
}

ll h(int u, int j){
      if(j == 0) return pw[u];
      if(dp2[u][j] != -1) return dp2[u][j];
      return dp2[u][j] = h(u, j-1) + h(f(u, j-1), j-1);
}

int LCA(int u, int v){
      if(L[u] < L[v]) swap(u, v);

      for(int i = MAXL-1; i >= 0; i--)
            if(L[ f(u, i) ] >= L[v])
                  u = f(u, i);

      if(u == v) return v;

      for(int i = MAXL-1; i >= 0; i--)
            if(f(u, i) != f(v, i))
                  u = f(u, i), v = f(v, i);

      return p[u];
}


long long DIST(int u, int v){
      if(L[u] < L[v]) swap(u, v);

      ll ans = 0;

      for(int i = MAXL-1; i >= 0; i--)
            if(L[ f(u, i) ] >= L[v])
                  ans += h(u, i), u = f(u, i);

      if(u == v) return ans;

      for(int i = MAXL-1; i >= 0; i--)
            if(f(u, i) != f(v, i))
                  ans += h(u, i) + h(v, i), u = f(u, i), v = f(v, i);

      ans += pw[u] + pw[v];
      return ans;
}

int KTH(int a, int b, int c){
      int lca = LCA(a, b);
      int u = a, v = b;
      if(c > L[a] - L[lca] + 1){
            c -= L[a] - L[lca] + 1;
            for(int i = MAXL-1; i >= 0; i--)
                  if(L[ f(v, i) ] - L[lca] >= c)
                        v = f(v, i);
            return v;
      }
      else{
            for(int i = MAXL-1; i >= 0; i--)
                  if(L[a] - L[ f(u, i) ] + 1 <= c)
                        u = f(u, i);
            return u;
```

```
	}
	return 0;
}

int main(){
	int n, T, a, b, c;

	scanf("%d", &T);

	for(int t = 1; t <= T; t++){
		scanf("%d", &n);
		g.assign(n+1, vector<ii>());

		for(int i = 1; i < n; i++){
			scanf("%d␣%d␣%d", &a, &b, &c);
			g[a].push_back(ii(b, c));
			g[b].push_back(ii(a, c));
		}

		memset(vis, 0, sizeof vis);
		memset(p, 0, sizeof p);
		memset(L, 0, sizeof L);
		L[1] = 0;
		p[1] = 1;
		dfs(1);

		memset(dp, -1, sizeof dp);
		memset(dp2, -1, sizeof dp2);

		char s[10];

		while(scanf("␣%s", s) == 1 && s[1] != 'O'){
			if(s[0] == 'D'){
				scanf("%d␣%d", &a, &b);
				printf("%lld\n", DIST(a, b));
			}
			else{
				scanf("%d␣%d␣%d", &a, &b, &c);
				printf("%d\n", KTH(a, b, c));
			}
		}
		printf("\n");
	}
}
```

Fluxo Máximo

The Cool Monkeys

```
#include <bits/stdc++.h>
using namespace std;

#define ff first
#define ss second
#define pb push_back
#define mp make_pair

const int oo = 1000000000;

typedef pair<int,int> ii;

vector<vector<int> > g;

int m, na, nb, t, source, target, ha[505], hb[505];
int mat[2020][2020], p[2020], vis[2020];

int back(int u, int minEdge){
	if(u == source) return minEdge;
	int f = back(p[u], min(minEdge, mat[ p[u] ][u]));
	mat[ p[u] ][u] -= f;
	mat[u][ p[u] ] += f;
	return f;
}

int maxflow(){
	int mf = 0, f = 1;
	while(f){
		queue<int> q;
		q.push(source);
		memset(vis, 0, sizeof vis);

		vis[source] = 1;
		p[source] = source;

		while(!q.empty()){
			int u = q.front(); q.pop();

			if(u == target) break;

			for(int i = 0; i < g[u].size(); i++){
				int v = g[u][i];
				if(mat[u][v] > 0 && vis[v] != 1){
					vis[v] = 1;
					p[v] = u;
					q.push(v);
				}
			}
		}

		if(vis[target] != 1) break;
		f = back(target, oo);
		mf += f;
```

```
        }
        return mf;
}

int buildRun(int *ha, int na, int *hb, int nb){
        g.assign(2020, vector<int>());
        sort(ha, ha+na, greater<int>());
        sort(hb, hb+nb);

        memset(mat, 0, sizeof mat);

        int cnt = 0;
        for(int i = 0; i < na; i++){
                for(int j = 0; j < nb; j++){
                        int vin = i;
                        int uin = na+j;
                        int vout = na+nb+i;
                        int uout = na+nb+na+j;
                        if(abs(ha[i] - hb[j]) < t){
                                // vout -> uin
                                // uout -> vin
                                // printf("%d -> %d\n", i, j);
                                g[vout].pb(uin);
                                g[uin].pb(vout);
                                g[uout].pb(vin);
                                g[vin].pb(uout);
                                mat[vout][uin] = oo;
                                mat[uout][vin] = oo;
                        }
                }
        }

        for(int i = 0; i < na; i++){
                int vin = i;
                int vout = na+nb+i;
                g[vin].pb(vout);
                g[vout].pb(vin);
                mat[vin][vout] = 1;
        }


        for(int j = 0; j < nb; j++){
                int uin = na+j;
                int uout = na+nb+na+j;
                g[uin].pb(uout);
                g[uout].pb(uin);
                mat[uin][uout] = 1;
        }

        for(int i = 0; i < m; i++){
                int vin = i;
                int uout = na+nb+na+i;
                g[source].pb(vin);
                g[vin].pb(source);
                mat[source][vin] = 1;

                g[uout].pb(target);
                g[target].pb(uout);
                mat[uout][target] = 1;
        }

        return maxflow();
}

int main(){
        source = 2018;
        target = 2019;

        scanf("%d␣%d␣%d␣%d", &m, &na, &nb, &t);

        for(int i = 0; i < na; i++)
                scanf("%d", ha+i);
        for(int i = 0; i < nb; i++)
                scanf("%d", hb+i);

        if(buildRun(ha, na, hb, nb) == m || buildRun(hb, nb, ha, na) == m) printf("S\n");
        else printf("N\n");

        return 0;
}
```

Batalha Naval

```
#include <bits/stdc++.h>

using namespace std;

int pX[200200], pY[200200], distX[200200], distY[200200];
int n, ptrx = 1, ptry = 100100;

vector<vector<int> > g(200200);

bool bfs(){
   bool found = false;
   queue<int> Q;

   for(int i = 1; i < ptrx; i++)
      if(pX[i] == -1)
         Q.push(i);

   memset(distX, 0, sizeof distX);
   memset(distY, 0, sizeof distY);
   while(!Q.empty()){
      int u = Q.front(); Q.pop();
      for(int i = 0; i < (int)g[u].size(); i++){
         int v = g[u][i];
         if(distY[v] == 0){
            distY[v] = distX[u]+1;
            if(pY[v] == -1) found = true;
            else{
               distX[ pY[v] ] = distY[v]+1;
               Q.push(pY[v]);
            }
         }
      }
   }
   return found;
}

bool dfs(int u){
   for(int i = 0; i < (int)g[u].size(); i++){
      int v = g[u][i];
      if(distX[u]+1 == distY[v]){
         distY[v] = 0; // "apaga" o vertice para a dfs
         if(pY[v] == -1 || dfs(pY[v])){
            pX[u] = v, pY[v] = u;
            return true;
         }
      }
   }
   return false;
}

int f(){
   int ans = 0;
   memset(pX, -1, sizeof pX);
   memset(pY, -1, sizeof pY);

   while(bfs())
      for(int i = 1; i < ptrx; i++)
         if(pX[i] == -1 && dfs(i)) // eh possivel escolher um par para i
            ans++;

   return ans;
}

int main(){
   int x, y;

   map<int, int> mapx;
   map<int, int> mapy;

   while(scanf("%d", &n) == 1){
      g.assign(200200, vector<int>());

      for(int i = 0; i < n; i++){
         scanf("%d␣%d", &x, &y);

         if(mapx.find(x) == mapx.end()) mapx[x] = ptrx++;
         if(mapy.find(y) == mapy.end()) mapy[y] = ptry++;

         x = mapx[x];
         y = mapy[y];

         g[x].push_back(y);
         g[y].push_back(x);
      }

      printf("%d\n", f());
   }
   return 0;
}
```

Dinic

```
// Network flow (Dinic's algorithm)
//
// Given a directed weighted graph, source, and sink, computes the maximum flow
// from source to sink.
//
// To use, call init(n), then add edges using edge(x, y, c1, c2), and finally
// call run(src, sink).
//
// Functions:
// - init(n) initializes the algorithm with the given number of nodes
// - edge(x, y, c1, c2) adds edges x->y of capacity c1 and y->x of capacity c2
// - run(src, sink) runs the algorithm and returns the total flow
//
// Time complexity: O(V^2 * E)
//
// Constants to configure:
// - MAXV is the maximum number of vertices
// - MAXE is the maximum number of edges (i.e. twice the calls to function edge)
// - oo is the "infinity" value

namespace Dinic {
 const int MAXV = 1000100;
 const int MAXE = 1000100;
 const llint oo = 1e18;

 int V, E;
 int last[MAXV], dist[MAXV], curr[MAXV];
 int next[MAXE], adj[MAXE]; llint cap[MAXE];
```

```
  void init(int n) {
   V = n;
   E = 0;
   for(i = 0; i < V; i++) last[i] = -1;
  }

  void edge(int x, int y, llint c1, llint c2) {
   adj[E] = y; cap[E] = c1; next[E] = last[x]; last[x] = E++;
   adj[E] = x; cap[E] = c2; next[E] = last[y]; last[y] = E++;
  }

  llint push(int x, int sink, llint flow) {
   if (x == sink) return flow;

   for (int &e = curr[x]; e != -1; e = next[e]) {
    int y = adj[e];

    if (cap[e] && dist[x] + 1 == dist[y])
     if (llint f = push(y, sink, min(flow, cap[e])))
      return cap[e] -= f, cap[e^1] += f, f;
   }
   return 0;
  }

  llint run(int src, int sink) {
   llint ret = 0;
   for (;;) {
    for(i = 0; i < V; i++) curr[i] = last[i];
    for(i = 0; i < V; i++) dist[i] = -1;

    queue<int> Q;
    Q.push(src), dist[src] = 0;

    while (!Q.empty()) {
     int x = Q.front(); Q.pop();

     for (int e = last[x]; e != -1; e = next[e]) {
      int y = adj[e];
      if (cap[e] && dist[y] == -1) Q.push(y), dist[y] = dist[x] + 1;
     }
    }
    if (dist[sink] == -1) break;

    while (llint f = push(src, sink, oo)) ret += f;
   }
   return ret;
  }
}
```

Min Cost Max Flow

```
// Min-cost max-flow (uses Dijkstra's algorithm)
//
// Given a directed weighted graph, source, and sink, computes the minimum cost
// of the maximum flow from source to sink.
// This version uses Dijkstra's algorithm and gives good performance on all
// kinds of graphs.
//
// To use, call init(n), then add edges using edge(x, y, c, w), and finally
// call run(src, sink).
//
// Functions:
// - init(n) initializes the algorithm with the given number of nodes
// - edge(x, y, c, w) adds an edge x->y with capacity c and weight w
// - run(src, sink) runs the algorithm and returns {total_cost, total_flow}
//
// Time complexity: O(V * E^2 log E)
//
// Constants to configure:
// - MAXV is the maximum number of vertices
// - MAXE is the maximum number of edges (i.e. twice the calls to function edge)
// - oo is the "infinity" value

namespace Mcmf {
 const int MAXV = 1000100;
 const int MAXE = 1000100;
 const llint oo = 1e18;

 int V, E;
 int last[MAXV], how[MAXV]; llint dist[MAXV];
 int next[MAXE], from[MAXE], adj[MAXE]; llint cap[MAXE], cost[MAXE];

 struct cmpf {
  bool operator () (int a, int b) {
   if (dist[a] != dist[b]) return dist[a] < dist[b];
   return a < b;
  }
 };
 set<int, cmpf> S;

 void init(int n) {
  V = n;
  E = 0;
  REP(i, V) last[i] = -1;
 }

 void edge(int x, int y, llint c, llint w) {
  from[E] = x; adj[E] = y; cap[E] = c; cost[E] = +w; next[E] = last[x]; last[x] = E++;
  from[E] = y; adj[E] = x; cap[E] = 0; cost[E] = -w; next[E] = last[y]; last[y] = E++;
 }

 pair<llint, llint> run(int src, int sink) {
  llint total = 0;
```

```
    llint flow = 0;

    for (;;) {
      REP(i, V) dist[i] = oo;
      dist[src] = 0;

      for (;;) {
        bool done = true;
        REP(x, V) for (int e = last[x]; e != -1; e = next[e]) {
          if (cap[e] == 0) continue;

          int y = adj[e];
          llint val = dist[x] + cost[e];

          if (val < dist[y]) {
            dist[y] = val;
            how[y] = e;
            done = false;
          }
        }
        if (done) break;
      }

      if (dist[sink] >= oo / 2) break;

      llint aug = cap[how[sink]];
      for (int i = sink; i != src; i = from[how[i]])
        aug = min(aug, cap[how[i]]);

      for (int i = sink; i != src; i = from[how[i]]) {
        cap[how[i]] -= aug;
        cap[how[i]^1] += aug;
        total += cost[how[i]] * aug;
      }
      flow += aug;
    }
    return {total, flow};
  }
}
```

Min Cost Max Flow(DFS)

```
// Min-cost max-flow (uses DFS)
//
// Given a directed weighted graph, source, and sink, computes the minimum cost
// of the maximum flow from source to sink.
// This version uses DFS to find shortest paths and gives good performance on
// very "shallow" graphs: graphs which have very short paths between source
// and sink (e.g. at most 10 edges).
// In such cases this algorithm can be orders of magnitude faster than the
// Dijkstra version.
//
// To use, call init(n), then add edges using edge(x, y, c, w), and finally
// call run(src, sink).
//
// Functions:
// - init(n) initializes the algorithm with the given number of nodes
// - edge(x, y, c, w) adds an edge x->y with capacity c and weight w
// - run(src, sink) runs the algorithm and returns {total_cost, total_flow}
//
// Time complexity: O(V * E^3)
//
// Constants to configure:
// - MAXV is the maximum number of vertices
// - MAXE is the maximum number of edges (i.e. twice the calls to function edge)
// - oo is the "infinity" value

namespace Mcmf {
  const int MAXV = 1000100;
  const int MAXE = 1000100;
  const llint oo = 1e18;

  int V, E;
  int last[MAXV], curr[MAXV], bio[MAXV]; llint pi[MAXV];
  int next[MAXE], adj[MAXE]; llint cap[MAXE], cost[MAXE];

  void init(int n) {
    V = n;
    E = 0;
    REP(i, V) last[i] = -1;
    REP(i, V) pi[i] = 0;
  }

  void edge(int x, int y, llint c, llint w) {
    adj[E] = y; cap[E] = c; cost[E] = +w; next[E] = last[x]; last[x] = E++;
    adj[E] = x; cap[E] = 0; cost[E] = -w; next[E] = last[y]; last[y] = E++;
  }

  llint push(int x, int sink, llint flow) {
    if (x == sink) return flow;
    if (bio[x]) return 0;
    bio[x] = true;

    for (int &e = curr[x]; e != -1; e = next[e]) {
      int y = adj[e];

      if (cap[e] && pi[x] == pi[y] + cost[e])
        if (llint f = push(y, sink, min(flow, cap[e])))
          return cap[e] -= f, cap[e^1] += f, f;
    }
    return 0;
  }
```

```
 pair<llint, llint> run(int src, int sink) {
  llint total = 0;
  llint flow = 0;
  pi[src] = oo;

  for (;;) {
   REP(i, V) bio[i] = false;
   REP(i, V) curr[i] = last[i];

   while (llint f = push(src, sink, oo)) {
    total += pi[src] * f;
    flow += f;
    REP(i, V) bio[i] = false;
   }

   llint inc = oo;
   REP(x, V) if (bio[x]) {
    for (int e = last[x]; e != -1; e = next[e]) {
     int y = adj[e];
     if (cap[e] && !bio[y]) inc = min(inc, pi[y] + cost[e] - pi[x]);
    }
   }
   if (inc == oo) break;

   REP(i, V) if (bio[i]) pi[i] += inc;
  }
  return {total, flow};
 }
}
```

Centroid

Junior e Falta de Ideias

```
#include <bits/stdc++.h>

#define ff first
#define ss second
#define mp make_pair

using namespace std;

typedef long long ll;

vector<pair<int,int>> G[500005];
int subtree[500005], treesize, k;
bool vis[500005];
ll dist[500005], ans;

int dfs(int v, int p){
	subtree[v] = 1;
	for(pair<int,int> x : G[v])
		if(x.ff != p && !vis[x.ff]) subtree[v] += dfs(x.ff,v);
	return subtree[v];
}

int centroid(int v, int p){
	for(pair<int,int> x : G[v]){
		if(x.ff == p || vis[x.ff]) continue;
		if(subtree[x.ff]*2 > treesize) return centroid(x.ff,v);
	}
	return v;
}

void procurar_ans(int v, int p, int d_atual, ll custo){
	ans = min(ans, dist[k-d_atual] + custo);
	if(d_atual == k) return;
	for(pair<int,int> x : G[v]){
		if(!vis[x.ff] && x.ff != p)
			procurar_ans(x.ff,v,d_atual+1,custo+x.ss);
	}
}

void atualiza_distancia(int v, int p, int d_atual, ll custo){
	dist[d_atual] = min(dist[d_atual], custo);
	if(d_atual == k) return;
	for(pair<int,int> x : G[v]){
		if(!vis[x.ff] && x.ff != p)
			atualiza_distancia(x.ff,v,d_atual+1,custo+x.ss);
	}
}

void decomp(int v, int p){
	treesize = dfs(v,v);
	// if(treesize < k) return;
	int cent = centroid(v,v);
	vis[cent] = 1;

	for(int i = 1; i <= treesize; i++)
		dist[i] = 1e18;

	for(pair<int,int> x : G[cent]){
		if(!vis[x.ff]){
			procurar_ans(x.ff,v,1,x.ss);
			atualiza_distancia(x.ff,v,1,x.ss);
		}
	}

	for(pair<int,int> x : G[cent]){
		if(!vis[x.ff])
			decomp(x.ff, v);
	}
}
```

```
int main(){
	int n,i,a,b;

	scanf("%d%d", &n,&k);
	for(i = 2; i <= n; i++){
		scanf("%d%d", &a,&b);
		G[i].push_back(mp(a,b));
		G[a].push_back(mp(i,b));
	}
	ans = 1e18;
	decomp(1,-1);

	printf("%lld\n", ans == 1e18 ? -1 : ans);

	return 0;
}
```

Strings

YATG

```
#include <bits/stdc++.h>

#define ff first
#define ss second
#define mp make_pair
#define oo 1000000000

using namespace std;

int n, k, mat[300005][26], ans;

int dfs(int u){
	int ret = oo;
	ans++;

	for(int i = 0; i < 26; i++){
		if(mat[u][i]){
			ret = min(ret, dfs(mat[u][i]));
		}
	}
	if(ret == oo){
		ret = 0;
		ans++;
	}
	ret++;
	if(ret > k){
		ans++;
		ret = 1;
	}
	return ret;
}

int main(){

	scanf("%d␣%d", &n, &k);

	char s[100005];

	int ptr = 1;

	for(int i = 0; i < n; i++){
		scanf("␣%s", s);
		int node = 0;
		for(int j = 0; s[j]; j++){
			int letra = s[j]-'a';
			if(!mat[node][letra])
				mat[node][letra] = ptr++;
			node = mat[node][letra];
		}
	}

	for(int i = 0; i < 26; i++)
		if(mat[0][i])
			dfs(mat[0][i]);

	ans += n;

	printf("%d\n", ans);

	return 0;
}
```

Vasiliy's Multiset

```
#include <bits/stdc++.h>

#define oo 1000000000000000000

using namespace std;

int mat[6400640][2];
int cnt[6400640][2];

int main(){
	int n, x, ptr, next = 1;
	char c;

	scanf("%d", &n);
	ptr = 0;
	for(int i = 30; i >= 0; i--){
		if(!mat[ptr][0]) mat[ptr][0] = next++;
		cnt[ptr][0]++;
		ptr = mat[ptr][0];
	}

	for(int i = 0; i < n; i++){
		scanf("␣%c␣%d", &c, &x);

		if(c == '+'){
			ptr = 0;
			for(int i = 30; i >= 0; i--){
				if((1 << i) & x){
					if(!mat[ptr][1]) mat[ptr][1] = next++;
					cnt[ptr][1]++;
					ptr = mat[ptr][1];
				}
				else{
					if(!mat[ptr][0]) mat[ptr][0] = next++;
					cnt[ptr][0]++;
					ptr = mat[ptr][0];
				}
			}
		}
		else if(c == '-'){
			ptr = 0;
			for(int i = 30; i >= 0; i--){
				if((1 << i) & x){
					cnt[ptr][1]--;
					ptr = mat[ptr][1];
				}
				else{
					cnt[ptr][0]--;
					ptr = mat[ptr][0];
				}
			}
		}
		else{
			int ans = 0;
			ptr = 0;
			for(int i = 30; i >= 0; i--){
				if((1 << i) & x){
					if(cnt[ptr][0] > 0){
						ans += (1 << i);
						ptr = mat[ptr][0];
					}
					else ptr = mat[ptr][1];
				}
				else{
					if(cnt[ptr][1] > 0){
						ans += (1 << i);
						ptr = mat[ptr][1];
					}
					else ptr = mat[ptr][0];
				}
			}
			printf("%d\n", ans);
		}
	}

	return 0;
}
```

Trie ponteiro

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

struct node{
	bool is_end;
	int prefixes, words, maxsize;
	struct node* edge[26];
	//initialize()
	//addword(vertex, word)
	//countPrefixes(vertex, prefix)
	//countWords(vertex, word)

	node(){
		maxsize = 0;
		prefixes = 0;
		words = 0;
		is_end = false;
		for(int i = 0; i < 26; i++) edge[i] = NULL;
	}

	void addWord(string word, int tam){
		maxsize = max(tam, maxsize);
		if(word.empty()){
			prefixes++;
			words++;
		}
		else{
			prefixes++;
			int k = word[0] - 'a';
			if(edge[k] == NULL){
				node *p1 = new node();
				p1->addWord(word.substr(1,word.size()-1), tam );
				edge[k] = p1;
			}
```

```
                else
                      edge[k]->addWord(word.substr(1,word.size()-1), tam );
          }
    }

    int countWords(string word){
          if(word.empty())
                return words;
          int k = word[0] - 'a';
          if(edge[k] == NULL)
                return 0;
          return edge[k]->countWords(word.substr(1,word.size()-1) );
    }

    int countPrefixes(string word){
          if(word.empty())
                return prefixes;
          int k = word[0] - 'a';
          if(edge[k] == NULL)
                return 0;
          return edge[k]->countPrefixes(word.substr(1,word.size()-1) );
    }

    int countSize(string word){
          if(word.empty())
                return maxsize;
          int k = word[0] - 'a';
          if(edge[k] == NULL)
                return 0;
          return edge[k]->countSize(word.substr(1,word.size()-1) );
    }
};

int main() {
    int n,m;
    string s;

    while(scanf("%d", &n) != EOF){
          node *trie = new node();
          while(n--){
                cin >> s;
                trie->addWord(s, s.size());
          }
          scanf("%d", &m);
          while(m--){
                cin >> s;
                int x = trie->countPrefixes(s);
                if(x == 0)
                      cout << "-1\n";
                else
                      cout << x << "␣" << trie->countSize(s) << endl;
          }
    }


    return 0;
}
```

Trie

```
#include <bits/stdc++.h>

#define oo 1000000000000000000

using namespace std;

int mat[6400640][2];
int cnt[6400640][2];

int main(){
    int n, x, ptr, next = 1;
    char c;

    scanf("%d", &n);

    ptr = 0;
    for(int i = 30; i >= 0; i--){
          if(!mat[ptr][0]) mat[ptr][0] = next++;
          cnt[ptr][0]++;
          ptr = mat[ptr][0];
    }

    for(int i = 0; i < n; i++){
          scanf("␣%c␣%d", &c, &x);

          if(c == '+'){
                ptr = 0;
                for(int i = 30; i >= 0; i--){
                      if((1 << i) & x){
                            if(!mat[ptr][1]) mat[ptr][1] = next++;
                            cnt[ptr][1]++;
                            ptr = mat[ptr][1];
                      }
                      else{
                            if(!mat[ptr][0]) mat[ptr][0] = next++;
                            cnt[ptr][0]++;
                            ptr = mat[ptr][0];
                      }
                }
          }
          else if(c == '-'){
                ptr = 0;
                for(int i = 30; i >= 0; i--){
```

```
                if((1 << i) & x){
                    cnt[ptr][1]--;
                    ptr = mat[ptr][1];
                }
                else{
                    cnt[ptr][0]--;
                    ptr = mat[ptr][0];
                }
            }
        }
        else{
            int ans = 0;
            ptr = 0;
            for(int i = 30; i >= 0; i--){
                if((1 << i) & x){
                    if(cnt[ptr][0] > 0){
                        ans += (1 << i);
                        ptr = mat[ptr][0];
                    }
                    else ptr = mat[ptr][1];
                }
                else{
                    if(cnt[ptr][1] > 0){
                        ans += (1 << i);
                        ptr = mat[ptr][1];
                    }
                    else ptr = mat[ptr][0];
                }
            }
            printf("%d\n", ans);
        }
    }

    return 0;
}
```

Aho Corasick

```
#include <bits/stdc++.h>

#define ff first
#define ss second
#define mp make_pair

using namespace std;

typedef long long ll;

int trie[1000005][52], fn[1000005];
int ptr;
bool passou[1000005];
vector<int> final_vec;

int insert(char *str){
    int v = 0;

    for(int i = 0; str[i]; i++){
        int to = str[i] >= 'a' ? str[i]-'a'+26 : str[i]-'A';
        if(trie[v][to])
            v = trie[v][to];
        else
            v = trie[v][to] = ptr++;
    }

    return v;
}

void init_aho(){
    queue<int> Q;

    Q.push(0);

    while(!Q.empty()){
        int t = Q.front(); Q.pop();

        for(int i = 0; i < 52; i++){
            if(trie[t][i]){
                int x = trie[t][i];
                Q.push(x);

                if(t){
                    fn[x] = fn[t];

                    while(fn[x] && trie[fn[x]][i] == 0) fn[x] = fn[fn[x]];
                    if(trie[fn[x]][i]) fn[x] = trie[fn[x]][i];
                    trie[x][i] = fn[x];
                }
            }
        }
    }
}

int main(){
    int t,q;
    char s[100005], s2[1005];

    scanf("%d", &t);
    while(t--){
        scanf("␣%s", s);

        memset(trie, 0, sizeof trie);
        memset(passou, 0, sizeof passou);
        memset(fn, 0, sizeof fn);
```

```
		final_vec.clear();

		ptr = 1;

		scanf("%d", &q);

		// build trie
		while(q--){
			scanf("␣%s", s2);
			final_vec.push_back(insert(s2));
		}

		// build failures
		init_aho();

		// simulate
		int v = 0;
		for(int i = 0; s[i]; i++){
			int to = s[i] >= 'a' ? s[i]-'a'+26 : s[i]-'A';
			v = trie[v][to];
			else{
				while(v){
					v = fn[v];
					passou[v] = 1;
					if(trie[v][to]){
						v = trie[v][to];
						break;
					}
				}
			}
			passou[v] = 1;
		}

		for(int i = 0; i < final_vec.size(); i++){
			if(passou[final_vec[i]]) printf("y\n");
			else printf("n\n");
		}
	}

	return 0;
}
```

Suffix Array

```
// Suffix array
//
// Given a string s of length N, function suffix_array(s, N) computes an array
// of sorted suffixes: the i-th sorted suffix starts from index srt[i].second.
// Time complexity: O(N log^2 N)
//
// Function lcp(a, b) computes the length of longest common prefix of suffixes
// s[a..] and s[b..].
// Time complexity: O(log N)
//
// Constants to configure:
// - MAX is the maximum value of N
// - LG is ceil(log2(MAX)) + 1

const int MAX = 100100;
const int LG = 17 + 1;

int L;
pair<llint, int> srt[MAX];
llint buc[LG][MAX];

int lcp(int a, int b) {
	int ret = 0;
	for(int i = L-1; i >= 0; --i){
		int s = 1 << i;
		if (a+s <= N && b+s <= N && buc[i][a] == buc[i][b])
			a += s, b += s, ret += s;
	}
	return ret;
}

void suffix_array(char *s, int N){
	for(i = 0; i < N; i++) buc[0][i] = s[i] + 1;

	for(L = 0; (1<<L) < 2*N; ++L){
		for(x = 0; x < N; x++) srt[x] = {buc[L][x] << 30, x};
		for (int x = (1<<L); x < N; x++) srt[x-(1<<L)].first += buc[L][x];
		sort(srt, srt+N);

		int pos = 1;
		for(x = 0; x < N; x++){
			pos += i && srt[i-1].first < srt[i].first;
			buc[L+1][srt[i].second] = pos;
		}
	}
}
```

Z Algorithm

```
// Z Algorithm
//
// Given a string s of length N, computes an array z, where z[i] is the length
// of longest substring starting from index i which is also a prefix of s.
// More information: http://codeforces.com/blog/entry/3107
//
// Time complexity: O(N)

void z_algorithm(char *s, int N, int *z) {
  z[0] = N;
  int L = -1, R = -1;

  FOR(i, 1, N) {
    z[i] = i >= R ? 0 : min(R-i, z[i-L]);
    while (i+z[i] < N && s[i+z[i]] == s[z[i]]) ++z[i];
    if (i+z[i] > R) L = i, R = i+z[i];
  }
}
```

Min rotation

```
// Lexicographically minimum rotation of a sequence
//
// Given a sequence s of length N, min_rotation(s, N) returns the start index
// of the lexicographically minimum rotation.
//
// Note: array s must be of length of at least 2 * N.
//
// Time complexity: O(N)

int min_rotation(int *s, int N) {
  REP(i, N) s[N+i] = s[i];

  int a = 0;
  REP(b, N) REP(i, N) {
    if (a+i == b || s[a+i] < s[b+i]) { b += max(0, i-1); break; }
    if (s[a+i] > s[b+i]) { a = b; break; }
  }
  return a;
}
```

All palindrome

```
// Finds all palindromes in a string
//
// Given a string s of length N, finds all palindromes as its substrings.
//
// After calling manacher(s, N, rad), rad[x] will be the radius of the largest
// palindrome centered at index x / 2.
// Example:
// s = b a n a n a a
// rad = 0000102010010
//
// Note: Array rad must be of length at least twice the length of the string.
// Also, "invalid" characters are denoted by -1, therefore the string must not
// contain such characters.
//
// Time complexity: O(N)
//
// Constants to configure:
// - MAX is the maximum length of the string

void manacher(char *s, int N, int *rad) {
  static char t[2*MAX];
  int m = 2*N - 1;

  REP(i, m) t[i] = -1;
  REP(i, N) t[2*i] = s[i];

  int x = 0;
  FOR(i, 1, m) {
    int &r = rad[i] = 0;
    if (i <= x+rad[x]) r = min(rad[x+x-i], x+rad[x]-i);
    while (i-r-1 >= 0 && i+r+1 < m && t[i-r-1] == t[i+r+1]) ++r;
    if (i+r >= x+rad[x]) x = i;
  }

  REP(i, m) if (i-rad[i] == 0 || i+rad[i] == m-1) ++rad[i];
  REP(i, m) rad[i] /= 2;
}
```

Geometria

Dividindo a coca

```
#include <cstdio>
#include <cmath>
using namespace std;

int main(){
    int c, n, l, b, H, i;
    double B, h, sup, inf, mid, bb, v;

    scanf("%d", &c);

    while(c--){
        scanf("%d␣%d␣%d␣%lf␣%d", &n, &l, &b, &B, &H);

        sup = H;
        inf = 0;

        while(sup - inf > 1e-9){
            mid = (sup + inf) / 2;

            bb = b + (B - b)*mid/sup;

            v = (M_PI*mid*(bb*bb + bb*b+ b*b))/3;
            if(v*n > l){
                sup = mid;
                B = bb;
        }
        else inf = mid;
    }
    printf("%.2lf\n", sup);
}

return 0;
```

```
}
// r*r = B*B*H/h

// H/(R-r) = h/(R1-r)
// R1-r = (R-r)*h/H
// R1 = r + (R-r)*h/H
```