1.1 - Convex Hull Trick Decrescente - Mínimo(nlogn)

```
typedef long long int ll;

struct pt{
    ll x, y;
    pt(){x=y=0;}
    pt(ll a, ll b) : x(a), y(b) {}
};

struct line{
    ll a, b;
    line(){a=b=0;}
    line(ll i, ll j) : a(i), b(j) {}

    ll value_at(ll x){
        return a*x + b;
    }
};


struct cht{

    int sz;
    vector<line> ch;

    cht(){ch.clear(); sz=0;}

    bool can_pop(line ant, line top, line at){
        return (top.b - ant.b)*(ant.a - at.a) >= (at.b - ant.b)*(ant.a -
top.a);
    }

    void add_line(line L){//retas ordenadas decrescente
        while(sz>1 && can_pop(ch[sz-2], ch[sz-1], L)){
            ch.pop_back();
            sz--;
        }
        ch.push_back(L);
        sz++;
    }

    ll query(int x){//query de minimo

        int ini=0, fim = sz-1, meio, ans;
        ans = sz-1;
```

```
        while(ini<=fim){
            meio = (ini+fim)/2;

            if(ch[meio].value_at(x) > ch[meio+1].value_at(x)){
                ini = meio+1;
            }else{
                fim = meio-1;
                ans = meio;
            }
        }
        return ch[ans].value_at(x);
    }

};
```

1.2 - Convex Hull Trick Decrescente - Mínimo(Linear)

```
typedef long long int ll;

struct pt{
    ll x, y;
    pt(){x=y=0;}
    pt(ll a, ll b) : x(a), y(b) {}
};

struct line{
    ll a, b;
    line(){a=b=0;}
    line(ll i, ll j) : a(i), b(j) {}

    ll value_at(ll x){
        return a*x + b;
    }
};

struct cht{
    int sz, pos;
    vector<line> ch;

    cht(){ch.clear(); sz=pos=0;}

    bool can_pop(line ant, line top, line at){
        return (top.b - ant.b)*(ant.a - at.a) >= (at.b - ant.b)*(ant.a -
top.a);
    }
```

```
    void add_line(line L){
        while(sz>1 && can_pop(ch[sz-2], ch[sz-1], L)){
            ch.pop_back();
            sz--;
        }
        ch.push_back(L);
        sz++;
    }

    ll query(int x){
        int ans = sz-1;
        for(int i=pos; i<sz-1; i++){
            if(ch[i].value_at(x) > ch[i+1].value_at(x)) pos++;
            else{
                ans=i;
                break;
            }
        }
        return ch[ans].value_at(x);
    }
};
```

1.3 - Convex Hull Trick Crescente - Máximo(nlogn)

```
typedef long long int ll;

struct line{
    ll a, b;
    line(){a=b=0;}
    line(ll i, ll j) : a(i), b(j) {}

    ll value_at(ll x){
        return a*x + b;
    }
};


struct cht{

    int sz;
    vector<line> ch;

    cht(){ch.clear(); sz=0;}

    bool can_pop(line ant, line top, line at){
        return (top.b - ant.b)*(ant.a - at.a) >= (at.b - ant.b)*(ant.a -
top.a);
```

```
    }

    void add_line(line L){
        while(sz>1 && can_pop(ch[sz-2], ch[sz-1], L)){
            ch.pop_back();
            sz--;
        }
        ch.push_back(L);
        sz++;
    }

    ll query(ll x){

        int ini=0, fim = sz-1, meio, ans;
        ans = sz-1;

        while(ini<=fim){
            meio = (ini+fim)/2;

            if(ch[meio].value_at(x) < ch[meio+1].value_at(x)){
                ini = meio+1;
            }else{
                fim = meio-1;
                ans = meio;
            }
        }
        return ch[ans].value_at(x);
    }

};
```

1.4 - Convex Hull Trick Crescente - Máximo(Linear)

```
typedef long long int ll;

struct line{
    ll a, b;
    line(){a=b=0;}
    line(ll i, ll j) : a(i), b(j) {}

    ll value_at(ll x){
        return a*x + b;
    }
};
```

```
struct cht{

    int sz, pos;
    vector<line> ch;

    cht(){ch.clear(); sz=pos=0;}

    bool can_pop(line ant, line top, line at){
        return (top.b - ant.b)*(ant.a - at.a) >= (at.b - ant.b)*(ant.a -
top.a);
    }

    void add_line(line L){
        while(sz>1 && can_pop(ch[sz-2], ch[sz-1], L)){
            ch.pop_back();
            sz--;
        }
        ch.push_back(L);
        sz++;
    }

    ll query(int x){
        int ans = sz-1;
        for(int i=pos; i<sz-1; i++){
            if(ch[i].value_at(x) < ch[i+1].value_at(x)) pos++;
            else{
                ans=i;
                break;
            }
        }
        return ch[ans].value_at(x);
    }

};
```

1.5 - Convex Hull Trick Crescente - Mínimo(variação)

```
typedef long long int ll;

struct line{

      ll a, b, id;

      line(){ a=b=0;}
      line(ll x, ll y, ll c) : a(x), b(y), id(c) {}
```

```
	ll value_at(ll x){
		return a*x + b;
	}

};

struct cht{

	int sz, pos;
	vector<line> ch;

	cht(){ch.clear(); sz=pos=0;}

	bool can_pop(line ant, line top, line at){
		ll p = (ant.b - at.b)*(top.a - ant.a);
		ll q = (ant.b - top.b)*(at.a - ant.a);
		return p>=q;
	}

	void add_line(line at){
		ll sz = ch.size();
		while(sz>1 && can_pop(ch[sz-2], ch[sz-1], at)) {
			ch.pop_back();
			sz--;
		}
		ch.push_back(at);
	}


	bool check(ll i, ll x){

		ll at = ch[i].value_at(x), ant = ch[i-1].value_at(x);
		if(ant > at) return true;
		return false;
	}

	ll query(ll x){

		ll ini, fim, meio, meio_, sz = ch.size();
		ini = 1; fim = sz-1;

		ll ans=0;

		while(ini<=fim){

			meio = (ini+fim)/2;

			if(check(meio, x)) {
```

```
                        ini = meio+1;
                        ans = meio;
                    }else fim = meio-1;
            }
            return ch[ans].value_at(x);
        }

};
```

1.6 - Divide and Conquer

```
typedef long long int ll;

ll n, K, dp[2][N];

void build_cost(){
    //depende do problema
}

ll get_cost(ll i, ll j){
    //depende do problema
}

void func(ll at, ll l, ll r, ll optL, ll optR){
    if(l>r) return;
    ll mid = (l+r)>>1;
    ll opt = 1;
    ll best = dp[at^1][mid];

    for(ll i=optL; i<=min(mid-1, optR); i++){
        ll c = get_cost(i+1, mid);
        if(dp[at^1][i]+c < best){
            best = dp[at^1][i] + c;
            opt = i;
        }
    }

    dp[at][mid] = best;

    func(at, l, mid-1, optL, opt);
    func(at, mid+1, r, opt, optR);
}


int main(){
```

```
    while(scanf("%lld %lld", &n, &K)!=EOF){
      //entrada
        build_cost();
        for(ll i=1; i<=n; i++){
            dp[1][i] = get_cost(1, i);
        }
        ll at=0;
        for(ll k=2; k<=K; k++){
            func(at, 1, n, 1, n);
            at^=1;
        }
        at^=1;
        printf("%lld\n", dp[at][n]);
    }
}
```

1.7 - Knuth

```
typedef long long int ll;

int n, opt[N][N];
ll acc[N], dp[N][N];
string answer;

void knuth(){

    for(int i=1; i<=n; i++) {
        dp[i][i] = acc[i]-acc[i-1];
        opt[i][i] = i;
    }

    for(int s = 2; s<=n; s++){
        for(int l=1; l+s-1<=n; l++){
            int r = l+s-1;

            int optL = opt[l][r-1];
            int optR = opt[l+1][r];
            int opt_ = optL;
            ll best = inf;

            for(int i=optL; i<=min(optR, r-1); i++){
                if(dp[l][i] + dp[i+1][r] < best){
                    best = dp[l][i]+dp[i+1][r];
                    opt_ = i;
                }
```

```
            }
            if(best == inf) best = 0;
            opt[l][r] = opt_;
            dp[l][r] = best+acc[r]-acc[l-1];
        }
    }
}

void solve(int l ,int r){//recupera resposta
    if(r<l) return;
    if(l == r){
        cout << answer << endl;
        return;
    }

    answer.push_back('0');
    solve(l, opt[l][r]);
    answer.back()='1';
    solve(opt[l][r]+1, r);
    answer.pop_back();
}

int main(){
    ios_base::sync_with_stdio(0); cin.tie(0);
    while(cin >> n){
        for(int i=1; i<=n; i++){
            cin >> acc[i];
            acc[i]+=acc[i-1];
        }

        knuth();
        solve(1, n);
    }
}
```

2 - Estruturas de dados

2.1 - BIT

```
struct BIT{
    #define LOGMAX 22
    #define N 101010

    int bit[N];
    BIT(){};
```

```
    void clear(){
        memset(bit, 0, sizeof bit);
    }

    void update(int pos, int v){
        for(; pos<N; pos+=(pos&(-pos))) bit[pos]+=v;
    }

    int sum(int pos){
        int s=0;
        for(; pos; pos-=(pos&(-pos))) s+=bit[pos];
        return s;
    }

    int kth(int k){

        int ans=0;
        for(int j=LOGMAX; j>=0; j--){
            if(ans+(1<<j) >= N) continue;

            if(bit[ans+(1<<j)]<k){
                ans+=(1<<j);
                k-=bit[ans];
            }
        }
        return ans+1;
    }

    int query(int l, int r){
        if(l > r) return 0;
        return sum(r) - sum(l-1);
    }
};
```

2.2 - BIT2D

```
struct BIT2D{
    #define MAXN 1010

    int bit[MAXN][MAXN];
    BIT2D(){}

    void reset(){
        memset(bit, 0, sizeof bit);
    }
```

```
    void update(int a, int b, int val){
        for(int x = a;  x < MAXN;  x+= (x & -x) ){

            for(int y = b;  y < MAXN;  y+= (y & -y) ){
                bit[x][y] += val;
            }

        }
    }

    int sum(int a, int b){
        int ans = 0;
        for(int x=a;  x;  x-= (x & -x)){

            for(int y=b;  y;  y-= (y & -y) ){
                ans += bit[x][y];
            }

        }
        return ans;
    }

    int query(int i1, int j1, int i2, int j2){
        return sum(i2, j2) + sum(i1-1, j1-1) - sum(i1-1,j2) - sum(i2,j1-1);
    }
};
```

2.3 - Exponenciação de Matriz

```
struct mat{

      ll m[N][N];

      mat(){ memset(m, 0, sizeof m); }
};

mat mult(mat a, mat b, ll na, ll mb, ll c){
//obs: considerar passagem de parametros por referencia
//multiplica duas matrizes (na x c)*(c x mb)

      mat ans;

      for(ll i=0; i<na; i++)
            for(ll j=0; j<mb; j++)
                  for(ll k=0; k<c; k++)
                        ans.m[i][j] = (ans.m[i][j] + a.m[i][k]*b.m[k][j])%MOD;
```

```
	return ans;
}

mat identity(){
	mat ans;
	for(ll i=0; i<N; i++) ans.m[i][i] = 1;
	return ans;
}

mat mat_pow(mat base, ll p){//obs: considerar passagem de parametros por ref.

	mat ans = identity();
	while(p>0){
		if(p&1) ans = mult(ans, base, N, N, N);
		base = mult(base, base, N, N, N);
		p>>=1;
	}
	return ans;
}


mat build(){
	//constroi a matriz de transição. Depende do problema

}


int main(){

	mat base, ans, T;
	T = build();//monta a matriz de transição

	//monta a matriz do caso base

	ans = mat_pow(T, expoente);//exponencia
	ans = mult(ans, base, _, _, _);//multiplica pelo caso base
}
```

2.4 - Merge Sort Tree

```
int n, k, q;
int v[MAXN];

struct MERGESORT_TREE{
    vector<int> st[4*MAXN];
```

```
MERGESORT_TREE(){}
void reset(){
    for (int i = 0; i < 4*MAXN; i++){
        st[i].clear();
    }
}

vector<int> merge(const vector<int> &a, const vector<int> &b){
    vector<int> ans;
    int i = 0, j = 0;
    while (ans.size() < k){
        if(i==a.size() && j==b.size()) break;
        if(i==a.size()){
            ans.pb(b[j++]);
        }else if(j==b.size()){
            ans.pb(a[i++]);
        }else{
            if(a[i] > b[j]){
                ans.pb(a[i++]);
            }else{
                ans.pb(b[j++]);
            }
        }
    }
    return ans;
}

void build(int no, int l, int r){
    if(l==r){
        st[no].pb(v[l]);
        return;
    }
    int nxt = 2*no;
    int mid = (l+r)/2;
    build(nxt, l, mid);
    build(nxt+1, mid+1, r);
    st[no] = merge(st[nxt], st[nxt+1]);
}

vector<int> query(int no, int l, int r, int i, int j){
    vector<int> ans;
    if(r<i || l>j) return ans;
    if(i<=l && r<=j) return st[no];
    int nxt = 2*no;
    int mid = (l+r)/2;

    return merge(query(nxt, l, mid, i, j), query(nxt+1, mid+1, r, i, j));
```

```
    }

};

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    cin >> n >> k >> q;
    MERGESORT_TREE tr;

    for (int i = 0; i < n; i++)
    {
        cin >> v[i];
    }
    tr.build(1, 0, n-1);

    vector<int> res;
    int l, r;
    ll ans;
    for (int i = 0; i < q; i++)
    {
        cin >> l >> r;
        l--; r--;
        res.clear();
        res = tr.query(1, 0, n-1, l, r);

        ans = res[0];
        for (int j = 1; j < res.size(); j++)
        {
            if(res[j]!=0){
                ans = (ans * 1LL * res[j])%MOD;
            }
        }

        cout << ans << "\n";
    }


    return 0;
}
```

2.5 - Segment Tree

```
int v[MAXN];
```

```
struct SEGTREE{
    int st[MAXN * 4];

    SEGTREE(){}

    void reset(){
        memset(st, 0, sizeof st);
    }

    int merge(int a, int b){
        return a+b;
    }

    void build(int no, int l, int r){
        if(l==r){
            st[no] = v[l];
            return;
        }
        int mid = (l+r)>>1;
        int nxt = no<<1;
        build(nxt, l, mid);
        build(nxt+1, mid+1, r);
        st[no] = merge(st[nxt], st[nxt+1]);
    }

    int query(int no, int l, int r, int i, int j){
        if(i<=l && r<=j) return st[no];
        if(i>r || j<l) return 0;

        int mid = (l+r)>>1;
        int nxt = no<<1;
        return merge(query(nxt, l, mid, i, j), query(nxt+1, mid+1, r, i, j));
    }

    void update(int no, int l, int r, int pos, int val){
        if(pos<l || pos>r) return;
        if(l==r){
            st[no] = val;
            return;
        }

        int mid=(l+r)>>1;
        int nxt = no<<1;
        update(nxt, l, mid, pos, val);
        update(nxt+1, mid+1, r, pos, val);
        st[no] = merge(st[nxt], st[nxt+1]);
    }
};
```

2.6 - Segment Tree + Lazy Propagation

```
int v[MAXN];

struct SEGTREE_LAZY{
    int st[MAXN * 4];
    int lazy[MAXN * 4];

    SEGTREE_LAZY(){}

    void reset(){
        memset(st, 0, sizeof st);
        memset(lazy, 0, sizeof lazy);
    }

    int merge(int a, int b){
        return a+b;
    }

    void build(int no, int l, int r){
        if(l==r){
            st[no] = v[l];
            lazy[no] = 0;
            return;
        }
        int mid = (l+r)>>1;
        int nxt = no<<1;

        build(nxt, l, mid);
        build(nxt+1, mid+1, r);

        st[no] = merge(st[nxt], st[nxt+1]);
        lazy[no] = 0;
    }

    void propagate(int no, int l, int r){
        if(!lazy[no]) return;

        int mid = (l+r)>>1;
        int nxt = no<<1;

        st[no] += (r-l+1)*lazy[no];

        if(l!=r){
            lazy[nxt] += lazy[no];
```

```
            lazy[nxt+1] += lazy[no];
        }
        lazy[no] = 0;
    }

    int query(int no, int l, int r, int i, int j){
        propagate(no, l, r);

        if(i<=l && r<=j) return st[no];
        if(i>r || j<l) return 0;

        int mid = (l+r)>>1;
        int nxt = no<<1;
        return merge(query(nxt, l, mid, i, j), query(nxt+1, mid+1, r, i, j));
    }

    void update(int no, int l, int r, int i, int j, int val){
        propagate(no, l, r);

        if(i>r || j<l) return;
        if(i<=l && r<=j){
            lazy[no] += val;
            propagate(no, l, r);
            return;
        }

        int mid = (l+r)>>1;
        int nxt = no<<1;

        update(nxt, l, mid, i, j, val);
        update(nxt+1, mid+1, r, i, j, val);

        st[no] = merge(st[nxt], st[nxt+1]);
    }
};
```

2.7 - Segment Tree Dinâmica

```
#define N 101010

typedef long long int ll;

struct no{
    ll val, lazy;
```

```
no *left, *right;
no() : val(0), lazy(0), left(NULL), right(NULL) {}



void do_lazy(int l, int r){
    if(lazy==0) return;
    val+= ((r-l)+1)*lazy;
    if(l<r){
        if(!left) left = new no();
        if(!right) right = new no();
        left->lazy+=lazy;
        right->lazy+=lazy;
    }
    lazy = 0;
}

void update(int l, int r, int a, int b, ll v){
    do_lazy(l, r);

    if(l>b || r<a) return;
    if(a<=l && b>=r) {
        lazy+=v;
        do_lazy(l, r);
        return;
    }

    int mid = (l+r)>>1;
    if(left == NULL) left = new no();
        left->update(l, mid, a, b, v);

    if(right == NULL) right = new no();
        right->update(mid+1, r, a, b, v);

    val = left->val + right->val;
}

ll query(int l, int r, int a, int b){
    do_lazy(l, r);

    if(l>b || r<a) return 0;
    if(a<=l && b>=r) return val;

    int mid = (l+r)>>1;
    ll x = (left) ? left->query(l, mid, a, b) : 0;
    ll y = (right) ? right->query(mid+1, r, a, b) : 0;
    return x+y;
}
```

```
    void destroy(){//nem todo problema precisa, mas pode dar merda se nao
destruir
        if(left) {
            left->destroy();
            free(left);
        }
        if(right) {
            right->destroy();
            free(right);
        }
        return;
    }
};
```

2.8 - Sparse Table

```
struct SparseTable{

    #define N 101010
    #define M 20

    int n, table[N][M];

    SparseTable() : n(0) {}

    SparseTable(int a) : n(a) {}

    void build(){//pressupoe que table[i][0] ja esteja calculado pra todo i
        for(int j=1; j<M; j++){
            for(int i=0; i+(1<<j)<=n; i++){
            //0-indexado. Pra 1-indexado faça: for(int i=1; i+(1<<j)<=n+1; i++)

                  table[i][j]= min(table[i][j-1],table[i+(1<<(j-1))][j-1]);
                  //se for soma, eh so trocar min por soma
            }
        }
    }

    int query_min(int l, int r){// pressupoe que l<=r
        int k = 31 - __builtin_clz(r-l+1);
        //se as variaveis forem long long, faça 63 - __builtin_clz(r-l+1)

        return min(table[l][k], table[r-(1<<k)+1][k]);
    }
```

```
    int query_soma(int l, int r){
    //pressupoe que a sparse table calculada seja de soma

        int ans=0;
        for(int j=M-1; j>=0; j--){
            if(l+(1<<j) > r+1) continue;
            ans+=table[l][j];
            l+=(1<<j);
        }
        return ans;
    }

};
```

2.9 - Persistent Segment Tree - Estática

```
/*
 *  SPOJ - MKTHNUM
 */

#include <bits/stdc++.h>

using namespace std;

#define N 101010

struct no{
    int l, r, val;
    no() : l(0), r(0), val(0) {}
}st[10101010];


int n, q, root[N], vet[N], inv[N], aux[N], nxt;

int update(int no1, int l, int r, int pos, int v){

    int no2 = nxt++;
    st[no2] = st[no1];
    if(l == r){
        st[no2].val+=v;
        return no2;
    }

    int mid = (l+r)>>1;
    if(pos<=mid) st[no2].l = update(st[no1].l, l, mid, pos, v);
    if(pos>mid) st[no2].r = update(st[no1].r, mid+1, r, pos, v);
```

```
    st[no2].val = st[st[no2].l].val + st[st[no2].r].val;
    return no2;
}

int query_k(int no1, int no2, int l, int r, int k){
    if(l == r) return l;
    int x = st[st[no2].l].val - st[st[no1].l].val;
    int mid = (l+r)>>1;


    if(x >= k) return query_k(st[no1].l, st[no2].l, l, mid, k);
    return query_k(st[no1].r, st[no2].r, mid+1, r, k-x);
}


int main(){

    scanf("%d %d", &n, &q);
    for(int i=1; i<=n; i++){
        scanf("%d", &vet[i]);
        aux[i] = vet[i];
    }

    sort(aux+1, aux+n+1);
    root[0] = 0;
    nxt = 1;
    for(int i=1; i<=n; i++){
        int a = lower_bound(aux+1, aux+n+1, vet[i]) - aux;
        inv[a] = vet[i];
        vet[i] = a;
        root[i] = update(root[i-1], 1, n, a, 1);
    }

    int a, b, c;
    for(int i=0; i<q; i++){
        scanf("%d %d %d", &a, &b, &c);
        printf("%d\n", inv[query_k(root[a-1], root[b], 1, n, c)]);
    }

}
```

2.10 - Persistent Segment Tree - Dinâmica

```
/*
 *  SPOJ - MKTHNUM
 */
```

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

using namespace std;

#define N 101010
#define inf 1000000100

struct no{

    no *left, *right;
    int val;

    no() : val(0), left(NULL), right(NULL) {}

    int join(no *a, no *b){
        int x = a ? a->val : 0;
        int y = b ? b->val : 0;
        return x+y;
    }

    no * update(int l, int r, int pos, int v){

        no *at = new no();
        *at = *this;

        if(l == r){
            at->val+=v;
            return at;
        }

        int mid = (l+r)>>1;

        if(pos<=mid){
            if(!left) left = new no();
            at->left = left->update(l, mid, pos, v);
        }else{
            if(!right) right = new no();
            at->right = right->update(mid+1, r, pos, v);
        }

        at->val = join(at->left, at->right);
        return at;
    }

};


no *root[N];
```

```
int vet[N], aux[N], inv[N];

int query_k(no *no1, no *no2, int l, int r, int k){

    if(l == r) return l;
    int a = (no1 && no1->left) ? no1->left->val : 0;
    int b = (no2 && no2->left) ? no2->left->val : 0;
    int x = b-a;

    int mid = (l+r)>>1;
    if(x>=k) return query_k( no1 ? no1->left : NULL, no2 ? no2->left : NULL, l,
mid, k );

 return query_k( no1 ? no1->right : NULL, no2 ? no2->right : NULL, mid+1, r,
k-x );

}


int main(){

    int n, q;
    scanf("%d %d", &n, &q);
    root[0] = new no();

    for(int i=1; i<=n; i++){
        scanf("%d", &vet[i]);
        aux[i] = vet[i];
    }
    int a, b, c;
    sort(aux+1, aux+n+1);
    for(int i=1; i<=n; i++){
        a = lower_bound(aux+1, aux+n+1, vet[i]) - aux;
        inv[a] = vet[i];
        vet[i] = a;
        root[i] = root[i-1]->update(1, n, vet[i], 1);
    }

    for(int i=0; i<q; i++){
        scanf("%d %d %d", &a, &b, &c);
        printf("%d\n", inv[query_k(root[a-1], root[b], 1, n, c)]);
    }
}
```

2.11 - Wavelet Tree

```
/*
 *  E da final brasileira de 2016
 */

#include <bits/stdc++.h>

using namespace std;

#define N 101010
#define inf 1e9

int n, vet[N], q;

struct wavelet{
    int low, high;
    vector<int> b;
    wavelet *left, *right;

    wavelet(int *from, int *to, int l, int h){//l e h sao o menor e o maior
elemento do alfabeto
        low = l, high = h;
        if(from == to || l == h) return;

        int mid = (l+h)>>1;

        auto f = [mid](int i){ return i<=mid; };

        b.push_back(0);
        for(int *it = from; it!=to; it++){
            b.push_back( b.back() + f(*it) );
        }

        int *pivo = stable_partition(from, to, f);
        left = new wavelet(from, pivo, l, mid);
        right = new wavelet(pivo, to, mid+1, h);
    }

    int kth(int l, int r, int k){
        if(low == high) return low;
        int lb = b[l-1];
        int rb = b[r];
        int c = rb-lb;
        if(c>=k) return left->kth(lb+1, rb, k);
        else return right->kth(l-lb, r-rb, k-c);
    }
```

```
    bool esq(int p){
        return b[p] == b[p-1]+1;
    }

    void update(int p){//swap p e p+1
        if(low == high) return;

        if(esq(p) && !esq(p+1)){
            swap(b[p], b[p+1]);
            b[p]--;
            return;
        }

        if(!esq(p) && esq(p+1)){
            b[p]++;
            return;
        }
        if(esq(p)) left->update(b[p]);
        else right->update(p-b[p]);
    }

};

int main(){

    scanf("%d %d", &n, &q);
    for(int i=1; i<=n; i++) scanf("%d", &vet[i]);
    wavelet *root = new wavelet(vet+1, vet+n+1, 0, inf);
    int a, b, c;
    char op;
    while(q--){
        scanf(" %c", &op);
        if(op == 'Q'){
            scanf("%d %d %d", &a, &b, &c);
            printf("%d\n", root->kth(a, b, c));
        }else{
            scanf("%d", &a);
            root->update(a);
        }
    }
}
```

2.12 - Wavelet Tree + Toggle

```
/*
 *  ILKQUERY 2 - toggle
```

```
 */

#include <bits/stdc++.h>

using namespace std;

#define N 101010
#define inf 1000000001

int vet[N], n, q, state[N];

typedef long long int ll;

struct BIT{
    vector<int> bit;
    int sz;

    BIT(){ bit.clear(); sz=0;}

    BIT(int n){
        sz=n;
        bit.assign(n+1, 0);
    }

    void update(int pos, int v){
        for(; pos<=sz; pos+= (pos&(-pos))) bit[pos]+=v;
    }

    int sum(int pos){
        int ans=0;
        for(; pos; pos-=(pos&(-pos))) ans+=bit[pos];
        return ans;
    }
};


struct wavelet{
    int low, high;
    vector<int> b;
    BIT bit;//a bit guarda a quantidade de elementos inativos no intervalo
    wavelet *left, *right;


    wavelet(int *from, int *to, int l, int h){
        low = l, high = h;
        left = right = NULL;

        bit = BIT(to-from+1);
```

```
        if(from == to || l==h) return;


        int mid = int( (ll(l) + ll(h) )>>1LL);

        auto f = [mid](int i){ return i<=mid; };

        b.push_back(0);
        for(int *it = from;  it!=to;  it++){
            b.push_back(b.back()+f(*it));
        }

        int *pivo = stable_partition(from, to, f);
        left = new wavelet(from, pivo, l, mid);
        right = new wavelet(pivo, to, mid+1, h);
    }

    int count_active(int l, int r){
        int x= (r-l+1) - bit.sum(r) + bit.sum(l-1);//qtd de elementos ativos:
|range| - qtd inativos no range
        return x;
    }

    void toggle(int pos, int v){
        bit.update(pos, v);
        if(low == high) return;

        int rb = b[pos];
        int lb = b[pos-1];
        int c = rb-lb;

        if(c) left->toggle(lb+1, v);
        else right->toggle(pos-rb, v);
    }

    int query(int l, int r, int k){//quantos elementos igual a k ativos existem
no intervalo
        if(l>r) return 0;
        if(low == high) return (low == k) ? count_active(l, r) : 0;

        int mid = int( (ll(low)+ ll(high))>>1LL );
        int rb = b[r];
        int lb = b[l-1];
        if(k<=mid) return (left) ? left->query(lb+1, rb, k) : 0;
        else return (right) ? right->query(l-lb, r-rb, k) : 0;
    }
};
```

```
wavelet *WT;

int main(){

    scanf("%d %d", &n, &q);
    int menor=inf, maior=-inf;
    for(int i=1; i<=n; i++){
        scanf("%d", &vet[i]);
        maior = max(maior, vet[i]);
        menor = min(menor, vet[i]);
        state[i] = 1;
    }

    WT = new wavelet(vet+1, vet+n+1, menor, maior);

    int op, a, b, k;
    while(q--){
        scanf("%d", &op);
        if(op){
            scanf("%d", &a); a++;
            if(state[a]) WT->toggle(a, 1);
            else WT->toggle(a, -1);

            state[a]^=1;
        }else{
            scanf("%d %d %d", &a, &b, &k); a++; b++;
            printf("%d\n", WT->query(a, b, k));
        }
    }
}
```

2.13 - Treap

```
#include <cstdio>
#include <set>
#include <algorithm>
using namespace std;

//Treap para arvore binária de busca
struct node{
    int x, y, size;
    node *l, *r;
    node(int _x){
        x = _x;
        y = rand();
        size = 1;
```

```
        l = r = NULL;
    }
};

//10 vezes mais lento que Red-Black....
//Tome uma array de pontos (x,y) ordenados por x. u é ancestral de v se e
somente se y(u) é maior que todos os elementos de u a v, v incluso!
//Split separa entre k-1 e k.
class Treap{
private:
    node* root;
    void refresh(node* t){
        if (t == NULL) return;
        t->size = 1;
        if (t->l != NULL)
            t->size += t->l->size;
        if (t->r != NULL)
            t->size += t->r->size;
    }
    void split(node* &t, int k, node* &a, node* &b){
        node * aux;
        if(t == NULL){
            a = b = NULL;
            return;
        }
        else if(t->x < k){
            split(t->r, k, aux, b);
            t->r = aux;
            refresh(t);
            a = t;
        }
        else{
            split(t->l, k, a, aux);
            t->l = aux;
            refresh(t);
            b = t;
        }
    }
    node* merge(node* &a, node* &b){
        node* aux;
        if(a == NULL) return b;
        else if(b == NULL) return a;
        if(a->y < b->y){
            aux = merge(a->r, b);
            a->r = aux;
            refresh(a);
            return a;
        }
```

```
        else{
            aux = merge(a, b->l);
            b->l = aux;
            refresh(b);
            return b;
        }
    }
    node* count(node* t, int k){
        if(t == NULL) return NULL;
        else if(k < t->x) return count(t->l, k);
        else if(k == t->x) return t;
        else return count(t->r, k);
    }
    int size(node* t){
        if (t == NULL) return 0;
        else return t->size;
    }
    node* nth_element(node* t, int n){
        if (t == NULL) return NULL;
        if(n <= size(t->l)) return nth_element(t->l, n);
        else if(n == size(t->l) + 1) return t;
        else return nth_element(t->r, n-size(t->l)-1);
    }
    void del(node* &t){
        if (t == NULL) return;
        if (t->l != NULL) del(t->l);
        if (t->r != NULL) del(t->r);
        delete t;
        t = NULL;
    }
public:
    Treap(){ root = NULL; }
    ~Treap(){ clear(); }
    void clear(){ del(root); }
    int size(){ return size(root); }
    bool count(int k){ return count(root, k) != NULL; }
    bool insert(int k){
        if(count(root, k) != NULL) return false;
        node *a, *b, *c, *d;
        split(root, k, a, b);
        c = new node(k);
        d = merge(a, c);
        root = merge(d, b);
        return true;
    }
    bool erase(int k){
        node * f = count(root, k);
        if(f == NULL) return false;
```

```
        node *a, *b, *c, *d;
        split(root, k, a, b);
        split(b, k+1, c, d);
        root = merge(a, d);
        delete f;
        return true;
    }
    int nth_element(int n){
        node* ans = nth_element(root, n);
        if (ans == NULL) return -1;
        else return ans->x;
    }

};

/*
 * TEST MATRIX
 */

int vet[10000009];

void test(){
    set<int> s;
    Treap t;
    int N = 1000000;
    for(int i=0; i<N; i++){
        int n = rand()%1000;
        if(!s.count(n)){
            s.insert(n);
            t.insert(n);
            //if(!t.insert(n)) printf("error inserting %d in treap!\n", n);
            //printf("inserted %d\n", n);
        }
        else{
            s.erase(n);
            t.erase(n);
            //if(!t.erase(n)) printf("error erasing %d in treap!\n", n);
            //printf("erased %d\n", n);
        }
        n = rand()%1000;
        if (s.count(n) != t.count(n)){
            printf("failed test %d, s.count(%d) = %d, t.count(%d) = %d\n", i,
n, s.count(n), n, t.count(n));
        }
    }
    s.clear();
    t.clear();
    for(int i=0; i<N; i++){
```

```
        vet[i] = i+1;
    }
    random_shuffle(vet, vet+N);
    for(int i=0; i<N; i++){
        t.insert(vet[i]);
    }
    for(int i=1; i<=N; i++){
        if (t.nth_element(i) != i){
            printf("failed test %d\n", i);
        }
    }
}

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

2.14 - Implicit Treap

```
#include <cstdio>
#include <vector>
#include <algorithm>
#include <ctime>
#define INF (1 << 30)
using namespace std;

const int neutral = 0; //comp(x, neutral) = x
int comp(int a, int b){
    return a + b;
}

//Treap para arvore binária de busca
struct node{
    int y, v, sum, size;
    bool swap;
    node *l, *r;
    node(int _v){
        v = sum = _v;
        y = rand();
        size = 1;
        l = r = NULL;
        swap = false;
    }
};
```

```
//10 vezes mais lento que Red-Black....
//Tome uma array de pontos (x,y) ordenados por x. u é ancestral de v se e
somente se y(u) é maior que todos os elementos de u a v, v incluso!
//Split separa entre em uma árvore com k elementos e outra com size-k.
class ImplicitTreap{
private:
    node* root;
    void refresh(node* t){
        if (t == NULL) return;
        t->size = 1;
        t->sum = t->v;
        if (t->l != NULL){
            t->size += t->l->size;
            t->sum = comp(t->sum, t->l->sum);
            t->l->swap ^= t->swap;
        }
        if (t->r != NULL){
            t->size += t->r->size;
            t->sum = comp(t->sum, t->r->sum);
            t->r->swap ^= t->swap;
        }
        if (t->swap){
            swap(t->l, t->r);
            t->swap = false;
        }
    }
    void split(node* &t, int k, node* &a, node* &b){
        refresh(t);
        node * aux;
        if(t == NULL){
            a = b = NULL;
            return;
        }
        else if(size(t->l) < k){
            split(t->r, k-size(t->l)-1, aux, b);
            t->r = aux;
            refresh(t);
            a = t;
        }
        else{
            split(t->l, k, a, aux);
            t->l = aux;
            refresh(t);
            b = t;
        }
    }
    node* merge(node* &a, node* &b){
        refresh(a);
```

```
        refresh(b);
        node* aux;
        if(a == NULL) return b;
        else if(b == NULL) return a;
        if(a->y < b->y){
            aux = merge(a->r, b);
            a->r = aux;
            refresh(a);
            return a;
        }
        else{
            aux = merge(a, b->l);
            b->l = aux;
            refresh(b);
            return b;
        }
    }
    node* at(node* t, int n){
        if (t == NULL) return NULL;
        refresh(t);
        if(n < size(t->l)) return at(t->l, n);
        else if(n == size(t->l)) return t;
        else return at(t->r, n-size(t->l)-1);
    }
    int size(node* t){
        if (t == NULL) return 0;
        else return t->size;
    }
    void del(node* &t){
        if (t == NULL) return;
        if (t->l != NULL) del(t->l);
        if (t->r != NULL) del(t->r);
        delete t;
        t = NULL;
    }
public:
    ImplicitTreap(){ root = NULL; }
    ~ImplicitTreap(){ clear(); }
    void clear(){ del(root); }
    int size(){ return size(root); }
    bool insertAt(int n, int v){
        node *a, *b, *c, *d;
        split(root, n, a, b);
        c = new node(v);
        d = merge(a, c);
        root = merge(d, b);
        return true;
    }
```

```
    bool erase(int n){
        node *a, *b, *c, *d;
        split(root, n, a, b);
        split(b, 1, c, d);
        root = merge(a, d);
        if (c == NULL) return false;
        delete c;
        return true;
    }
    int at(int n){
        node* ans = at(root, n);
        if (ans == NULL) return -1;
        else return ans->v;
    }
    int query(int l, int r){
        if (l>r) swap(l, r);
        node *a, *b, *c, *d;
        split(root, l, a, d);
        split(d, r-l+1, b, c);
        int ans = (b != NULL ? b->sum : neutral);
        d = merge(b, c);
        root = merge(a, d);
        return ans;
    }
    void reverse(int l, int r){
        if (l>r) swap(l, r);
        node *a, *b, *c, *d;
        split(root, l, a, d);
        split(d, r-l+1, b, c);
        if(b != NULL) b->swap ^= 1;
        d = merge(b, c);
        root = merge(a, d);
    }
};

/*
 * TEST MATRIX
 */

bool test(){
    srand(time(NULL));
    vector<int> v;
    ImplicitTreap t;
    int N = 10000;
    vector<int>::iterator it;
    bool toprint = false;
    for(int i=0, n, k, l, r; i<N; i++){
        if (i%5 == 0 && i > 0){
```

```
            n = rand()%((int)v.size());
            if (toprint) printf("deleting v[%d] = %d\n", n, v[n]);
            it = v.begin()+n;
            v.erase(it);
            t.erase(n);
        }
        else if (i%5 == 4){
            l = rand()%((int)v.size());
            r = rand()%((int)v.size());
            if (l>r) swap(l, r);
            if (toprint) printf("reversing %d to %d\n", l, r);
            for(int j=l; j<=r && j<=r-j+l; j++){
                swap(v[j], v[r-j+l]);
            }
            t.reverse(l, r);
        }
        else{
            n = rand()%((int)v.size()+1);
            k = rand()%1000;
            if (toprint) printf("inserting %d in pos %d\n", k, n);
            it = v.begin()+n;
            v.insert(it, k);
            t.insertAt(n, k);
        }
        if (toprint) printf("array: ");
        for(int j=0; j<(int)v.size(); j++){
            if (toprint) printf("%d ", v[j]);
            if (v[j] != t.at(j)){
                printf("test %d failed, v[%d] = %d, t.at(%d) = %d\n", i+1, j,
v[j], j, t.at(j));
                return false;
            }
        }
        if (toprint) printf("\n");
        l = rand()%((int)v.size());
        r = rand()%((int)v.size());
        if (l>r) swap(l, r);
        int ans = neutral;
        for(int j=l; j<=r; j++){
            ans = comp(ans, v[j]);
        }
        if (toprint) printf("sum(%d, %d) = %d = %d\n", l, r, ans, t.query(l,
r));
        if (ans != t.query(l, r)){
            printf("test %d failed, ans(%d, %d) = %d = %d\n", i, l, r, ans,
t.query(l, r));
            return false;
        }
```

```
    }
    return true;
}

int main(){
    if(test()) printf("all tests passed\n");
    return 0;
}
```

3 - Max Flow

3.1 - Dinic

```
#define N 50500//depende do problema
#define M 10100100//depende do problema
#define inf 10101010

typedef pair<int, int> ii;


struct ed{
    int to, c, f;
}edge[M];

int n, m, ptr[N], dist[N], curr, s, t;
vector<int> adj[N];
queue<int> q;


void add_edge(int a, int b, int c, int r){

    edge[curr].to = b;
    edge[curr].c = c;
    edge[curr].f = 0;
    adj[a].push_back(curr++);

    edge[curr].to = a;
    edge[curr].c = r;
    edge[curr].f = 0;
    adj[b].push_back(curr++);
}

void build_graph(){

    s = curr = 0;
    t = N-2;
    //modelagem do grafo
}
```

```
bool bfs(){

    q.push(s);
    memset(dist, -1, sizeof dist);
    dist[s] = 0;

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

        for(int i=0; i<adj[u].size(); i++){
            int e = adj[u][i];
            int v = edge[e].to;
            int w = edge[e].c - edge[e].f;

            if(dist[v] != -1 || w<=0) continue;

            q.push(v);
            dist[v] = dist[u]+1;
        }
    }

    return dist[t]!=-1;
}


int dfs(int u, int f){

    if(u == t) return f;

    for(; ptr[u]<adj[u].size(); ptr[u]++){

        int e = adj[u][ptr[u]];
        int v = edge[e].to;
        int w = edge[e].c - edge[e].f;

        if(dist[v]!=dist[u]+1) continue;

        if(w>0){
            if(int a = dfs(v, min(f, w))){
                edge[e].f+=a;
                edge[e^1].f-=a;
                return a;
            }
        }
    }
    return 0;
}
```

```
int dinic(){

    int flow = 0;

    while(1){

        if(!bfs()) break;

        memset(ptr, 0, sizeof ptr);

        while(int a = dfs(s, inf)){
            flow+=a;
        }
    }
    return flow;
}



int main(){

    //le grafo
    build_graph();

    int mf = dinic();

}
```

3.2 - Edmonds Karp

```
struct ed{
    int to, c, f;
}edge[M];


int n, m, seen[N], tempo, curr, p[N], nxt[N], dist[N], s, t;
vector<int> adj[N];


void add_edge(int a, int b, int c, int rev){

    edge[curr].to = b;
    edge[curr].c = c;
    edge[curr].f = 0;
```

```
    adj[a].push_back(curr++);

    edge[curr].to = a;
    edge[curr].c = rev;
    edge[curr].f = 0;
    adj[b].push_back(curr++);
}

build_graph(){
      //depende do problema
}

int augment(){
    int ans = inf;
    for(int u=t, e = p[u];     u!=s;        u = edge[e^1].to, e = p[u]){
        int w = edge[e].c - edge[e].f;
        ans = min(ans, w);
    }

    for(int u=t, e = p[u];     u!=s;        u = edge[e^1].to, e = p[u]){
        edge[e].f+=ans;
        edge[e^1].f-=ans;
    }
    return ans;
}

int bfs(){

    p[t] = -1;
    queue<int> q;
    q.push(s);

    while(q.size()){
        int u = q.front(); q.pop();
        if(u == t) break;
        for(int i=0; i<adj[u].size(); i++){
            int e = adj[u][i];
            int v = edge[e].to;
            if(seen[v] < tempo && edge[e].c - edge[e].f > 0){
                q.push(v);
                seen[v] = tempo;
                p[v] = e;
            }
        }
    }
    if(p[t] == -1) return 0;
    return augment();
}
```

```
int edmonds_karp(){
    int flow=0;
    memset(seen, 0, sizeof seen);
    tempo = 1;

    while(int a = bfs()){
        flow+=a;
        tempo++;
    }
    return flow;
}


int main(){

    cin >> n >> m;
    build_graph();
    cout << "Max flow = " << edmonds_karp() << endl;
}
```

3.3 - Ford Fulkerson

```
#define N 10040//depende do problema
#define M 1010101//depende do problema
#define inf 10101010//depende do problema

struct ed{
    int to, c, f;
}edge[M];


int n, curr, seen[N], tempo, s, t;
vector<int> adj[N];


void add_edge(int a, int b, int c, int r){

    edge[curr].to = b;
    edge[curr].c = c;
    edge[curr].f = 0;
    adj[a].push_back(curr++);

    edge[curr].to = a;
    edge[curr].c = r;
```

```
    edge[curr].f = 0;
    adj[b].push_back(curr++);
}


void build_graph(){

    s = curr = 0;
    t = N-2;
    //modelagem do grafo
}

int dfs(int u, int f){

    if(u == t) return f;

    seen[u] = tempo;

    for(int i=0; i<adj[u].size(); i++){
        int e = adj[u][i];
        int v = edge[e].to;
        int w = edge[e].c - edge[e].f;

        if(seen[v]<tempo && w>0){
            if(int a = dfs(v, min(f, w))){
                edge[e].f+=a;
                edge[e^1].f-=a;
                return a;
            }
        }
    }
    return 0;
}


int ford_fulk(){

    memset(seen, 0, sizeof seen);
    tempo = 1;
    int flow = 0;

    while(int a = dfs(s, inf)){
        flow+=a;
        tempo++;
    }
    return flow;
}
```

```
int main(){


    //le grafo

    //monta o grafo

    build_graph();

    int mf = ford_fulk();
}
```

3.4 - Min Cost Max Flow

```
struct ed{
    ll to, c, f, cost;
}edge[M];


ll n, k, dist[N], p[N], seen[N], curr, s, t;
vector<int> adj[N];

void add_edge(ll a, ll b, ll c, ll cost){//arestas indo com custo positivo, e
voltando com custo negativo

    edge[curr] = {b, c, 0, cost};
    adj[a].push_back(curr++);

    edge[curr] = {a, 0, 0, -cost};
    adj[b].push_back(curr++);
}


void build_graph(){

    s = curr = 0;
    t = N-2;
    //modelagem do grafo
}

ll augment(){
    ll mf = inf;
    ll ans = 0;
    for(ll u = t, e = p[u]; u!=s; u = edge[e^1].to, e = p[u]){
        mf = min(mf, edge[e].c - edge[e].f);
```

```
    }
    for(ll u = t, e = p[u]; u!=s; u = edge[e^1].to, e = p[u]){
        ans += mf*edge[e].cost;
        edge[e].f+=mf;
        edge[e^1].f-=mf;
    }
    return ans;
}

ll SPF(){

    for(ll i=0; i<N; i++) dist[i] = inf;
    p[s] = p[t] = -1;

    dist[s] = 0;  seen[s] = 1;
    queue<int> q; q.push(s);

    while(q.size()){

        ll u = q.front(); q.pop();

        seen[u] = 0;
        for(ll i=0; i<adj[u].size(); i++){
            ll e = adj[u][i];
            ll v = edge[e].to;
            ll w = edge[e].c - edge[e].f;

            if(w>0 && dist[v] > dist[u]+edge[e].cost){
                dist[v] = dist[u]+edge[e].cost;
                p[v] = e;
                if(!seen[v]){
                    seen[v] = 1;
                    q.push(v);
                }
            }
        }

    }

    if(p[t] == -1) return inf;
    return augment();

}

ll MCMF(){
    ll ans = 0;
    while(1) {
        ll a = SPF();
```

```
        if(a == inf) break;
        ans+=a;
    }
    return ans;
}


int main(){


    //leitura do grafo

    build_graph();

    ll x = MCMF();

    //
}
```

3.5 - Resumão de flow

RESUMO DOS ALGORITMOS CLASSICOS DE FLOW
Min-Path-Cover:

Minimo numero de caminhos para visitar todos os vertices num DAG
Constroi o grafo bipartido Vout / Vin, add todas as arestas u-v:
out(u) - in(v).
add aresta s-out(u) pra todo u, e in(u)-t pra todo u.
Todas as arestas com capacidade 1.

Edge-disjoint/independent paths

Encontre o maior numero de caminhos que nao compartilham nenhuma
aresta(edge-disjoint) no caminho de s-t, num grafo qualquer.
Encontre o maior numero de caminhos que nao compartilham nenhuma aresta e
nenhum vertice(independent path) no caminho de s-t, num grafo qualquer.
Coloque o peso de cada aresta igual a 1, e pra independent paths coloque
capacidade 1 em cada vertice tambem.

Max Weighted Independent Set
Grafo bipartido, cada vertice tem um peso, coloque peso[u] como
capacidade da aresta s-u, e todas as outras arestas como infinito.

COMPLEXIDADE DOS ALGORITMOS
grafos genéricos:
Ford fulkerson: O(f*E)
Edmonds Karp: VE^2
Dinic: V^2E

Pra grafos bipartidos a complexidade do Dinic é O(sqrt(V)*E), e a do Ford geralmente é V^2 dependendo do problema.

4 - Grafos

4.1 - Bellman Ford

```
vii Grafo[MAXN];
int dist[MAXN];
int parent[MAXN];
vi pathToDest;
int n;
bool hasNegativeCycle;

int BellmanFord(int source, int dest){
    int custo, v;
    hasNegativeCycle = false;
    for (int i = 0; i < n; i++){
        dist[i] = 1e8;
        parent[i] = -1;
    }
    dist[source]=0;
    parent[source]=source;

    for (int j = 0; j < n-1; j++)//roda n-1 vezes
    {
        for (int u = 0; u < n; u++)
        {
            for (int i = 0; i < Grafo[u].size(); i++)
            {
                v = Grafo[u][i].first;
                custo = Grafo[u][i].second;
                if(dist[v] > dist[u] + custo){
                    dist[v] = dist[u] + custo;
                    parent[v] = u;
                }
            }
        }
    }

    //se quiser saber quais vertices estao no ciclo é só adicionar outr for de
0 até 5, por exemplo, e ver qual distancia diminuiu. Se rodar só uma vez
```

dependendo da configuração das aresta pode ser que não ache todos do ciclo, por isso é melhor rodar uma quantidade X de vezes, o ideal seria X = n

```
    for (int u = 0; !hasNegativeCycle && u < n; u++)
    {
        for (int i = 0; !hasNegativeCycle && i < Grafo[u].size(); i++)
        {
            v = Grafo[u][i].first;
            custo = Grafo[u][i].second;

            if(dist[v] > dist[u] + custo)//se depois de n-1 iterações ainda
existe um caminho menor, existe um ciclo negativo
                hasNegativeCycle = true;
        }
    }

    if(!hasNegativeCycle){
        pathToDest.clear();
        v = dest;
        while(v!=source){
            pathToDest.push_back(v);
            v = parent[v];
            //~ cout << v << endl;
        }
        pathToDest.push_back(source);
    }
    return dist[dest];
}

/*
limpa();
BellmanFord(origem, destino) retorna o menor caminho. Se tiver ciclo negativo a
variável hasNegativeCycle vai ser true.
*/
```

4.2 - Centroid Decomposition

```
/*
 *  Cf 161D : quantos pares de vertices com distancia = k
 */

//ATENCAO: Prestar atenção nos caminhos que começam no centroid, e na
contribuição de cada centroid na resposta final

int n, k, dist[N], h[N], sz[N], block[N];
ll answer;
```

```
vector<int> adj[N];


void build_sz(int u, int p){
    sz[u] = 1;
    for(int v : adj[u]){
        if(v == p || block[v]) continue;
        build_sz(v, u);
        sz[u]+=sz[v];
    }
}

int find_centroid(int u, int p, int tam){
    for(int v : adj[u]){
        if(v == p || block[v]) continue;
        if(sz[v]*2 > tam) return find_centroid(v, u, tam);
    }
    return u;
}

void dfs(int u, int p, int d){
    dist[d]++;
    for(int v : adj[u]){
        if(v == p || block[v]) continue;
        dfs(v, u, d+1);
    }
}

void solve(int u, int p, int d){

    if(d>=k) return;
    answer+= (ll)dist[k-d];
    for(int v : adj[u]){
        if(v == p || block[v]) continue;
        solve(v, u, d+1);
    }
}

void decompose(int u){

    build_sz(u, u);
    u = find_centroid(u, u, sz[u]);
    block[u] = 1;

    for(int v : adj[u]){
        if(block[v]) continue;
        solve(v, u, 1);
        dfs(v, u, 1);
```

```
    }

    answer+= (ll)dist[k];
    for(int i=1; dist[i] > 0; i++) dist[i] = 0;

    for(int v : adj[u]){
        if(block[v]) continue;
        decompose(v);
    }
}


int main(){

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

    answer = 0;
    decompose(1);
    printf("%lld\n", answer);
}
```

4.3 - Dijkstra

```
int n, m, dist[N], pai[N], s, t;
vector<ii> adj[N];

int dijkstra(){
    memset(pai, -1, sizeof pai);
    for(int i=0; i<n; i++) dist[i] = inf;

    dist[s] = 0;
    priority_queue< ii, vector<ii>, greater<ii> > pq;
    pq.push(ii(0, s));

    while(pq.size()){
        ii foo = pq.top(); pq.pop();
        int u = foo.S, d = foo.F;

        if(dist[u] < d) continue;
```

```
        for(ii f : adj[u]){
            int v = f.F, w = f.S;
            if(dist[v] > dist[u]+w){
                pai[v] = u;
                dist[v] = dist[u]+w;
                pq.push(ii(dist[v], v));
            }
        }
    }
    return (pai[t] == -1) ? -1 : dist[t];
}
```

4.4 - Flood Fill

```
char vis[MAXN][MAXN];
char grid[MAXN][MAXN];
int n, m;
int dx[]={1,0,-1,0};
int dy[]={0,1,0,-1};

bool pode(int x, int y){
    return x>=0 && x<n && y>=0 && y<m && !vis[x][y] && grid[x][y] == 'A';
}

void dfs(int x, int y){
    vis[x][y] = 1;
    grid[x][y] = 'T';

    for (int i = 0; i < 4; i++)
    {
        if(pode(x+dx[i], y+dy[i])){
            dfs(x+dx[i], y+dy[i]);
        }
    }
}
```

4.5 - Floyd Warshall - All Pairs of Shortest Paths + Recuperação de caminho

```
int n;
int dist[MAXN][MAXN];
int pai[MAXN][MAXN];
```

```
void reset(){
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j++) {
            dist[i][j] = INF;
            if(i==j) dist[i][j]=0;

            pai[i][j] = i;
        }
    }
}

void printPath(int i, int j) {
    if (i != j) printPath(i, pai[i][j]);
    printf(" %d", j+1);
}

int main(){
    int m;
    cin >> n >> m;
    reset();

    int u, v, w;
    for (int i = 0; i < m; i++)
    {
        cin >> u >> v >> w;
        u--; v--;
        dist[u][v] = w;
        dist[v][u] = w;
    }

    for(int k = 0; k < n; k++){
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                if(dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                    pai[i][j] = pai[k][j];
                }
            }
        }
    }

    while (cin >> u >> v)
    {
        u--; v--;
        cout << "dist = " << dist[u][v] << "\n";
        cout << "path = "; printPath(u, v); cout << "\n";
    }
```

4.6 - Floyd Warshall - Fecho Transitivo

```
//inicializa com 1 onde tem aresta e 0 onde não tem

for (int k = 0; k < V; k++)
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            dist[i][j] |= (dist[i][k] & dist[k][j]);
```

4.7 - Floyd Warshall - Minimax

```
Minimax: arv. ger. min e maior aresta
Maximin: arv. ger. max e menor aresta
*/
int N, E;
int main()
{
    int i, u, v, w, q;
    int g[200][200];
    int caso=1;

    while(scanf("%d %d %d", &N, &E, &q), N != 0) {
        for (int i = 1; i <= N; i++)
        {
            for (int j = 1; j <= N; j++)
            {
                g[i][j]=10000000;
                if(i==j) g[i][j]=0;
            }
        }

        for(i = 0; i < E; i++) {
            scanf("%d %d %d", &u, &v, &w);
            g[u][v]=w;
            g[v][u]=w;
        }

        for(int k = 1; k <= N; k++)
           for(int i = 1; i <= N; i++)
              for(int j = 1; j <= N; j++)
                 g[i][j] = min(g[i][j], max(g[i][k], g[k][j]));//pega a maior
aresta do caminho (so existe um caminho, é uma arvore)
```

```
    }

    return 0;
}
```

4.8 - Kosaraju - Componentes Fortemente Conexas

```
int n, m;
vector<int> g[MAXN];
vector<int> t[MAXN];//grafo transposto
char vis[MAXN];
stack<int> p;

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

    int v;
    if(op == 1){
        for (int i = 0; i < g[u].size(); i++)
        {
            v = g[u][i];
            if(!vis[v]){
                dfs(v, op);
            }
        }
        p.push(u);
    }else{
        for (int i = 0; i < t[u].size(); i++)
        {
            v = t[u][i];
            if(!vis[v]){
                dfs(v, op);
            }
        }
    }
}

int kosaraju(){//retorna quantas componentes fortemente conexas existe
    memset(vis, 0, sizeof vis);

    while (!p.empty())
        p.pop();

    for (int i = 0; i < n; i++)
    {
        if(!vis[i]) dfs(i, 1);
```

```
    }

    int u;
    int qtd = 0;
    memset(vis, 0, sizeof vis);
    while (!p.empty())
    {
        u = p.top();
        p.pop();
        if(!vis[u]){
            qtd++;
            dfs(u, 0);
        }
    }

    return qtd;
}

void reset(){
    for (int i = 0; i < n; i++)
    {
        g[i].clear();
        t[i].clear();
    }
}

int main(){
    reset();
    //le o grafo normal e transposto
    int ans = kosaraju();

    return 0;
}
```

4.9 - LCA (logN Padrão)

```
ll lca[N][LOGMAX], h[N];
ll minAresta[N][LOGMAX];

void dfs(ll x, ll ult, ll peso_ult_x) {
    lca[x][0] = ult;
    minAresta[x][0] = peso_ult_x;

    for(ll i = 1; i < LOGMAX; ++i){
        lca[x][i] = lca[lca[x][i - 1]][i - 1];
        minAresta[x][i] = min(minAresta[x][i-1], minAresta[lca[x][i-1]][i-1]);
```

```
    }
    ll y;
    for(ll i=0; i<g[x].size(); i++) {
        y = g[x][i].first;
        if(y == ult) continue;
        h[y] = h[x] + 1;
        dfs(y, x, g[x][i].second);
    }
}

ll getLca(ll a, ll b) {
    menorAresta = 10000000;
    if(h[a] < h[b]) swap(a, b);
    ll d = h[a] - h[b];
    for(ll i = LOGMAX - 1; i >= 0; --i){
        if((d >> i) & 1){
            menorAresta = min(menorAresta, minAresta[a][i]);
            a = lca[a][i];
        }
    }
    if(a == b) return a;
    for(ll i = LOGMAX - 1; i >= 0; --i){
        if(lca[a][i] != lca[b][i]){
            menorAresta = min(menorAresta, minAresta[a][i]);
            menorAresta = min(menorAresta, minAresta[b][i]);
            a = lca[a][i];
            b = lca[b][i];
        }
    }
    menorAresta = min(menorAresta, minAresta[a][0]);
    menorAresta = min(menorAresta, minAresta[b][0]);
    return lca[a][0];
}
```

4.10 - LCA com RMQ Query O(1)

```
//SPOJ LCA

#include <bits/stdc++.h>

using namespace std;

#define N 101010
#define M 22

int n, vet[N<<1], in[N], h[N<<1], dist[N], table[N<<1][M], tempo;
vector<int> adj[N];
```

```
void dfs(int u, int d, int pai){

    in[u] = tempo;
    h[tempo] = dist[u] = d+1;
    vet[tempo++] = u;

    for(int v : adj[u]){
        if(v == pai) continue;
        dfs(v, d+1, u);
        h[tempo] = d+1;
        vet[tempo++] = u;
    }
}

void build_table(){
    int sz = tempo;
    for(int i=0; i<sz; i++) table[i][0] = vet[i];
    for(int j=1; j<M; j++){
        for(int i=0; i+(1<<j)<=sz; i++){
            int u = table[i][j-1];
            int v = table[i+(1<<(j-1))][j-1];
            table[i][j] = (dist[u] < dist[v]) ? u : v;
        }
    }
}

int query(int l, int r){
    int k = 31 - __builtin_clz(r-l+1);
    int u = table[l][k];
    int v = table[r-(1<<k)+1][k];
    return (dist[u] < dist[v]) ? u : v;
}

int get_lca(int u, int v){
    if(in[u] > in[v]) swap(u, v);
    return query(in[u], in[v]);
}

int main(){
        //le a árvore
        tempo = 0;
        dfs(1, 0, -1);//supondo que a raiz da arvore seja o vertice 1
        build_table();

}
```

4.11 - MST - Árvore Geradora Mínima

```
int n, comp[N], m;
vector<iii> edge;

void init(){
    edge.clear();
    for(int i=0; i<=n; i++) comp[i] = i;
}

int find(int i){
    return (comp[i] == i) ? i : comp[i] = find(comp[i]);
}

bool same(int i, int j) {
    return find(i) == find(j);
}

void join(int i, int j){
    comp[find(i)] = find(j);
}

int MST(){

    sort(edge.begin(), edge.end());
    int ans=0;
    for(int i=0; i<m; i++){
        int u = edge[i].S.F, v = edge[i].S.S, w = edge[i].F;
        if(!same(u, v)){
            join(u, v);
            ans+=w;
        }
    }
    return ans;
}

int main(){
    while(scanf("%d %d", &n, &m)){
        if(!n && !m) break;
        init();
        int a, b, c, tot=0;
        for(int i=0; i<m; i++){
            scanf("%d %d %d", &a,&b,&c);
            edge.push_back(iii(c, ii(a, b)));
            tot+=c;
        }
```

```
        printf("%d\n", tot-MST());
    }

}
```

4.12 - Ordenação Topológica - DFS

```
int n, m;
vector<int> g[MAXN];
char vis[MAXN];
vector<int> ts;

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

    int v;
    for (int i = 0; i < g[u].size(); i++)
    {
        v = g[u][i];
        if(!vis[v]){
            dfs(v);
        }
    }
    ts.pb(u);
}

int main(){
    // le o grafo
    // chama dfs
    // ordenação topológica invertida vai estar em ts
    return 0;
}
```

4.13 - Ordenação Topológica - Kahn

```
int grauEntrada[MAXN], u, v;
vector<int> g[MAXN];
vector<int> topoSort;
/*
    - Mantem na fila os vertices que nao tem aresta de entrada
    - Remove todas as arestas que saem de u, e diminui o grau de entrada de
cada vizinho v de u
    - Se v passou a ter grau de entrada 0, adiciona ele na fila
```

```
    - Repete o processo até a fila esvaziar
*/

void Kahn(){
    queue<int> q;
    for (int i = 0; i < n; i++)
    {
        if(grauEntrada[i]==0) q.push(i);
    }

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

        for (int i = 0; i < g[u].size(); i++)
        {
            v = g[u][i];
            grauEntrada[v]--;
            if(grauEntrada[v]==0){
                q.push(v);
            }
        }
        g[u].clear();
    }
}

void limpa(){
    for (int i = 0; i < n; i++)
    {
        g[i].clear();
        grauEntrada[i]=0;
    }
    nome.clear();
    mapa.clear();
    topoSort.clear();
}

int main()
{
    limpa();
    //monta grafo
    Kahn();
    //percorre topoSort e printa
    return 0;
}
```

4.14 - Tarjan - Pontos/Pontes de articulação

```
#define N 101010
#define GRAY 1

int n, m, seen[N], in[N], low[N], tempo, root, bridges, AP;
vector<int> adj[N];

void tarjan(int u, int p){

    seen[u] = GRAY;
    in[u] = low[u] = tempo++;
    int any, child=any=0;

    for(int v : adj[u]){
        if(v == p) continue;

        if(!seen[v]){
            child++;
            tarjan(v, u);

            low[u] = min(low[u], low[v]);

            if(low[v] >= in[u]) any=1;
            if(low[v] > in[u]) bridges++;

        }else low[u] = min(low[u], in[v]);
    }

    if(child>1 && u == root) AP++;//caso especial: raiz é um vertice de
articulacao
    else if(any && u!=root) AP++;
}


int main(){
    int a, b;
    scanf("%d %d", &n, &m);
    for(int i=0;i<m; i++){
        scanf("%d %d", &a, &b);
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    root = 1;
    bridges = tempo = AP = 0;
```

```
    tarjan(1, 0);

    printf("Articulation points: %d\n", AP);
    printf("Bridges: %d\n", bridges);
}
```

4.15 - Tarjan - Componentes Fortemente Conexas

```
#define N 101010
#define GRAY 1
#define BLACK 2

int n, m, seen[N], low[N], in[N], comp[N], tempo, comp_cont;
vector<int> adj[N];
stack<int> pilha;

void tarjan_scc(int u){

    seen[u] = GRAY;
    in[u] = low[u] = tempo++;
    pilha.push(u);

    for(int v : adj[u]){
        if(seen[v] == BLACK) continue;

        if(!seen[v]){
            tarjan_scc(v);

            low[u] = min(low[v], low[u]);
        }else low[u] = min(low[u], in[v]);
    }

    if(low[u] == in[u]){
        comp_cont++;
        while(pilha.size()){
            int v = pilha.top(); pilha.pop();
            seen[v] = BLACK;
            comp[v] = comp_cont;
            if(u == v) break;
        }
    }

}


int main(){
```

```
    int a, b, op;
    scanf("%d %d", &n, &m);
    for(int i=0; i<m; i++){//recebe um grafo direcionado
        scanf("%d %d", &a, &b);
        adj[a].push_back(b);
    }

    memset(seen, 0, sizeof seen);
    comp_cont=tempo=0;

    for(int i=1; i<=n; i++){
        if(!seen[i]) tarjan_scc(i);
    }
    printf("%d\n", comp_cont == 1);//printa 1 se for fortemente conexo
}
```

4.16 - Tarjan - Grafo das Componentes Biconectadas

```
// O Grafo gerado eh uma árvore (ou uma floresta se for desconexo)

#define N 101010
#define GRAY 1

int n, seen[N], in[N], low[N], id[N], tempo, bridges, diametro;
vector<int> adj[N], tr[N];//tr: arvore das componentes biconectadas
stack<int> pilha;

void tarjan(int u, int p, int op){//op == 0: calcula pra cada vertice qual
componente que ele faz parte (id)

    seen[u] = GRAY;
    in[u] = low[u] = tempo++;

    if(!op) pilha.push(u);

    for(int v : adj[u]){
        if(v == p) continue;

        if(!seen[v]){
            tarjan(v, u, op);

            if(!op && low[v] > in[u]){
                while(pilha.size()){
                    int x = pilha.top(); pilha.pop();
                    id[x] = v;
                    if(v == x) break;
```

```
                }
            }
            if(op && low[v]>in[u]){
                tr[id[u]].push_back(id[v]);
                tr[id[v]].push_back(id[u]);
            }

            low[u] = min(low[u], low[v]);
        }else low[u] = min(low[u], in[v]);
    }
}

void build_tarjan(int op){
    tempo = bridges = 0;
    memset(seen, 0, sizeof seen);

    tarjan(1, 0, op);

    if(op) return;
    while(pilha.size()){
        int x = pilha.top(); pilha.pop();
        id[x] = 1;
    }
}

int main(){

    int a, b, tc, m;
    scanf("%d %d", &n, &m);
    for(int i=0; i<m; i++){
        scanf("%d %d", &a, &b);
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    build_tarjan(0);
    build_tarjan(1);

    //processa a arvore
}
```

4.17 - Tarjan - Grafo das Componentes Fortemente Conexas

```
//responde qual vertice alcanca a maior quantidade de vertices num grafo com
N<=100000
```

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

using namespace std;

#define N 101010
#define GRAY 1
#define BLACK 2

int n, m, seen[N], low[N], dp[N], in[N], comp[N], sz[N], tempo, comp_cont;
vector<int> adj[N], g[N];//adj eh o grafo normal, g eh o grafo das componentes
stack<int> pilha;

void tarjan_scc(int u, int op){//op == 0: calcula as scc de cada vertice, op ==
1: monta o grafo das scc

    seen[u] = GRAY;
    in[u] = low[u] = tempo++;
    pilha.push(u);

    for(int v : adj[u]){
        if(seen[v] == BLACK) {
            if(op == 1)   g[comp[u]].push_back(comp[v]);
            continue;
        }

        if(!seen[v]){
            tarjan_scc(v, op);

            if(op == 1 && seen[v] == BLACK){
                g[comp[u]].push_back(comp[v]);
            }

            low[u] = min(low[v], low[u]);
        }else low[u] = min(low[u], in[v]);
    }

    if(low[u] == in[u]){
        comp_cont++;

        while(pilha.size()){
            int v = pilha.top(); pilha.pop();
            seen[v] = BLACK;

            if(!op) comp[v] = comp_cont;
            if(!op) sz[comp_cont]++;


            if(u == v) break;
```

```
        }
    }

}


void build_tarjan(int op){
    memset(seen, 0, sizeof seen);
    comp_cont=tempo=0;

    for(int i=1; i<=n; i++){
        if(!seen[i]) tarjan_scc(i, op);
    }

    if(!op) return;

    for(int i=1; i<=comp_cont; i++){//tira as arestas repetidas do grafo das
scc
        if(!g[i].size()) continue;

        sort(g[i].begin(), g[i].end());

        g[i].resize( distance( g[i].begin(),  unique(g[i].begin(), g[i].end())
) );//tira repetições
    }
}


void solve(int u){
    if(dp[u]!=0) return;
    dp[u] = sz[u];
    for(int v : g[u]){
        solve(v);
        dp[u]+=dp[v];
    }
}


int main(){
    int a, b, op;
    scanf("%d %d", &n, &m);
    for(int i=0; i<m; i++){//recebe um grafo direcionado
        scanf("%d %d %d", &a, &b, &op);
        adj[a].push_back(b);
        if(op == 2) adj[b].push_back(a);
    }

    build_tarjan(0);
```

```
    build_tarjan(1);
    memset(dp, 0, sizeof dp);

    for(int i=1; i<=comp_cont; i++) solve(i);

    int ans=1;
    for(int i=1; i<=n; i++){
        if(dp[comp[i]] > dp[comp[ans]]) ans=i;
    }

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

4.18 - Shortest Path Faster - Menor caminho chinês

```
int n, m, dist[N], pai[N], in[N], s, t;
vector<ii> adj[N];

int SPF(){
    memset(pai, -1, sizeof pai);
    memset(in, 0, sizeof in);
    for(int i=0; i<n; i++) dist[i] = inf;
    dist[s] = 0;
    queue<int> q;
    q.push(s);

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

        in[u] = 0;
        for(ii f : adj[u]){
            int v = f.F, w = f.S;
            if(dist[v] > dist[u]+w){
                pai[v] = u;
                dist[v] = dist[u]+w;
                if(!in[v]){
                    q.push(v);
                    in[v] = 1;
                }
            }
        }
    }
    return (pai[t] == -1) ? -1 : dist[t];
}

int main(){
```

```
    int tc, a, b, c, x=1;
    scanf("%d", &tc);
    while(tc--){
        scanf("%d %d %d %d", &n, &m, &s, &t);
        for(int i=0; i<=n; i++) adj[i].clear();
        for(int i=0; i<m; i++){
            scanf("%d %d %d", &a, &b, &c);
            adj[a].push_back(ii(b, c));
            adj[b].push_back(ii(a, c));
        }
        a = SPF();

        if(a>=0) printf("Case #%d: %d\n", x++, a);
        else printf("Case #%d: unreachable\n", x++);
    }
}
```

4.19 - Union Find

```
int n, sz[N], comp[N], cont_comp, maior;

void init(){
    cont_comp = n;
    maior = 1;
    for(int i=0; i<=n; i++) {
        comp[i] = i;
        sz[i] = 1;
    }
}

int find(int i){
    return (comp[i] == i) ? i : comp[i] = find(comp[i]);
}

bool same(int i, int j){
    return find(i) == find(j);
}

void join(int i, int j){
    int x = find(i), y = find(j);
    if(x == y) return;

    comp[y] = x;
    sz[x]+=sz[y];
    sz[y] = 0;
    cont_comp--;
```

```
    maior = max(maior, sz[x]);
}

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

    init();

    while(q--){
        scanf(" %c", &op);
        if(op == 'T'){
            printf("%d %d\n", cont_comp, maior);
            continue;
        }
        scanf("%d %d", &a, &b);

        if(op=='F') {
            join(a, b);
        }else{
            printf(find(a) == find(b) ? "sim\n" : "nao\n");
        }
    }
}
```

4.20 - Todos os menores caminhos com Dijkstra

```
int n, m;
int g[600][600];
int origem, destino;
set<int> parent[600];
char vis[600];
int dist[600];

void solve(int atual, int nxt){
    if(atual == origem){
        cout << origem << "\n";
        return;
    }

    cout << atual << " ";
    for (auto i : parent[atual])
    {
        int v = i;
        solve(v, atual);
```

```
    }
}

int dij(){
    priority_queue<pair<int, int> >pq;
    pq.push(mp(0, origem));
    parent[origem].insert(origem);
    dist[origem] = 0;

    int u, w, v;
    while (!pq.empty())
    {
        u = pq.top().S;
        pq.pop();
        if(vis[u]) continue;
        vis[u] = 1;

        if(u==destino) return dist[destino];
        for (int i = 0; i < n; i++)
        {
            if(g[u][i]){
                v = i;
                w = g[u][i];
                if(dist[u] + w <= dist[v]){
                    if(dist[u] + w < dist[v]) parent[v].clear();//se achou
caminho menor: limpa vetor de parent
                    parent[v].insert(u);
                    dist[v] = dist[u] + w;
                    pq.push(mp(-dist[v], v));
                }
            }
        }

    }
    return -1;
}

int main()
{
    reset();
    cout << dij() << endl;
    solve(destino, destino);//printa os caminhos invertidos: destino ... origem
    return 0;
}
```

4.21 - 2-SAT

```
vector<int> Grafo[MAXN], Transposto[MAXN];
int n, m, cnt;
int vis[MAXN];
int componente[MAXN];
stack<int> pilha;
map<string, int> mapa;
bool sat;
int ans[MAXN];

void limpa(){
    for (int i = 0; i <= MAXN; i++)
    {
        Grafo[i].clear();
        Transposto[i].clear();
    }
}
//da pra acessar a negacao de um elemento fazendo o xor. Deve ser indexado
como:
//true: x*2
//false: x*2 + 1
//CODIGO SENDO INDEXADO A PARTIR DE 0***************

void addEdge(int u, int v){
    Grafo[u^1].pb(v);// !u -> v
    Grafo[v^1].pb(u);// !v -> u

    Transposto[v].pb(u^1);//Grafo transposto pra rodar o kosaraju
    Transposto[u].pb(v^1);
}

void dfs1(int u){
    if (!vis[u])
    {
        vis[u]=1;
        for (int i = 0; i < Grafo[u].size(); i++)
        {
            int v = Grafo[u][i];
            if(!vis[v]) dfs1(v);
        }
        pilha.push(u);
    }
}

void dfs2(int u){
    if (!vis[u])
    {
```

```
        vis[u]=1;
        componente[u] = cnt;
        for (int i = 0; i < Transposto[u].size(); i++)
        {
            int v = Transposto[u][i];
            if(!vis[v]) dfs2(v);
        }
    }
}

void Kosaraju(){
    memset(vis, 0, sizeof vis);
    while(!pilha.empty()) pilha.pop();
    for (int i = 0; i < 2*n; i++)
        if(!vis[i]) dfs1(i);//visita todos os vertices

    memset(vis, 0, sizeof vis);
    memset(componente, 0, sizeof componente);
    cnt=0;
    int u;

    while(!pilha.empty()){
        u = pilha.top(); pilha.pop();
        if(!vis[u]){
            dfs2(u);
            cnt++;
        }
        ans[u/2] = 1-u%2;//atribui valores aos elementos. Se for satisfativel
da pra usar esse vetor
    }

    sat=true;
    for (int i = 0; i < n; i++)
    {
        if(componente[2*i] == componente[2*i + 1]) sat = false;//se estão na
mesma componente a formula nao tem solucao
    }
}
```

5 - Strings

5.1 - Aho-Corasick

```
string s, txt;
```

```
int cont;//contador global de nós

struct no{
    #define ALF 130 //depende do problema

    no *pai, *suffix_link, *nxt[ALF];
    char c;
    int fim, num;

    no(char letra, int id){
        c = letra;
        for(int i=0; i<ALF; i++) nxt[i] = NULL;
        pai = suffix_link = NULL;
        fim = 0;
        num = id;
    }


    void insert(int i){
        if(i == s.size()){
            fim++;
            return;
        }

        int letra = s[i]-'A';
        if(!nxt[letra]){
            nxt[letra] = new no(s[i], cont++);
            nxt[letra]->pai = this;
        }
        nxt[letra]->insert(i+1);
    }

    void build_sf(){

        queue<no*> q;
        for(int i=0; i<ALF; i++)
            if(nxt[i]) q.push(nxt[i]);

        while(q.size()){
            no *u = q.front(); q.pop();

            no *tmp = u->pai->suffix_link;

            char letra = u->c - 'A';

            while(tmp && !tmp->nxt[letra])   tmp = tmp->suffix_link;

            u->suffix_link = (tmp) ? tmp->nxt[letra] : this;
```

```
            u->fim += u->suffix_link->fim;

            for(int i=0; i<ALF; i++)
                if(u->nxt[i]) q.push(u->nxt[i]);
        }
    }

    void destroy(){
        for(int i=0; i<ALF; i++){
            if(nxt[i]){
                nxt[i]->destroy();
                delete nxt[i];
            }
        }
    }

};

no *root;

no *climb(no *u, char letra){
    no *tmp = u;
    while(tmp && !tmp->nxt[letra]) tmp = tmp->suffix_link;
    return tmp ? tmp->nxt[letra] : root;
}

int query(int pos, no *u){//exemplo de query, mas varia de problema pra
problema
    if(pos==txt.size()) return u->fim;
    return u->fim + query(pos+1, climb(u, txt[pos]-'A'));
}
```

5.2 - Hash

```
#define MAXN 100100
const ll A = 1009;
const ll MOD = 9LL + 1e18;
ll pot[MAXN];

ll normalize(ll r){
    while(r<0) r+=MOD;
    while(r>=MOD) r-=MOD;
    return r;
}

ll mul(ll a, ll b){//(a*b)%MOD
```

```
    ll q = ll((long double)a*b/MOD);
    ll r = a*b - MOD*q;
    return normalize(r);
}

ll add(ll hash, ll c){
    return (mul(hash, A) + c)%MOD;
}

void buildPot(){
    for (int i = 0; i < MAXN; i++)
    {
        pot[i] = i ? mul(pot[i-1], A) : 1LL;
    }
}

struct Hash{
    string s;
    ll hashNormal, hashInvertida;
    ll accNormal[MAXN], accInvertida[MAXN];

    Hash(){}
    Hash(string _s){
        s = _s;
    }

    void build(){
        accNormal[0] = 0LL;
        for (int i = 1; i <= (int)s.size(); i++){
            accNormal[i] = add(accNormal[i-1], s[i-1]-'a'+1);
        }
        hashNormal = accNormal[(int)s.size()];

        accInvertida[s.size()] = 0LL;
        for (int i = s.size()-1; i >= 0; i--){
            accInvertida[i] = add(accInvertida[i+1], s[i]-'a'+1);
        }
        hashInvertida = accInvertida[0];

    }

    ll getRangeNormal(int l, int r){//pega a hash da substring (l, r) na string
normal (abcd - [0, 2] = abc)
        if(l>r) return 0LL;
        ll ans = (accNormal[r+1] - mul(accNormal[l], pot[r-l+1]))%MOD;
        return normalize(ans);
    }
```

```
    ll getRangeInvertido(int l, int r){//pega a hash da substring (l, r) na
string invertida (abcd - [0, 2] = cba)
        if(l>r) return 0LL;
        ll ans = (accInvertida[l] - mul(accInvertida[r+1], pot[r-l+1]))%MOD;
        return normalize(ans);
    }
};

int main () {
    buildPot();//cuidado com o limite do MAXN
    //resolve o problema
    Hash H = Hash(str);
    H.build();

    return 0;
}
```

5.3 - Hash - Maior Substring Palindromo(nlogn)

```
Hash H;

bool ok(int tam){
    int l = 0;
    int r = tam-1;

    while (r < (int)H.s.size())
    {
        if(H.getRangeNormal(l, r) == H.getRangeInvertido(l, r)){
            return true;
        }

        l++; r++;
    }

    return false;
}

int longestPalindromicSubstring(string s, string &res){
//retorna o tamanho da maior substring palindromo
    H = Hash(s);
    H.build();

    int lo = 1;
    int hi = (int)s.size();
    int mid;
    int ans = 0;
```

```
    while (lo <= hi)
    {
        mid = (lo+hi)/2;
        if(ok(mid) || ok(mid+1)){
            lo = mid+1;
            ans = max(ans, mid);
        }else{
            hi = mid-1;
        }
    }

    //recupera a primeira string palindromo de tamanho ans
    res.clear();
    int l = 0, r = ans-1;
    while (r < (int)H.s.size())
    {
        if(H.getRangeNormal(l, r) == H.getRangeInvertido(l, r)){
            res = H.s.substr(l, ans);
            break;
        }
    }

    return ans;
}

int main(){
    //le a string
    // chama a função
}
```

5.4 - KMP

```
string s, txt;
int n, m, p[N];

void kmp(){

    p[0] = 0;
    for(int i=1; i<n; i++){

        p[i] = p[i-1];
        while(txt[p[i]] != txt[i] && p[i]) p[i] = p[p[i]-1];

        if(txt[p[i]] == txt[i]) p[i]++;
```

```
    }
    for(int i=0; i<n; i++) printf("p[%d] = %d\n", i, p[i]);
}


int main(){

    getline(cin, s);
    txt = s+"$";//importante
    getline(cin, s);
    txt+=s;

    n = txt.size();
    cout << txt << endl;
    kmp();

}
```

5.5 - Rabin Karp

```
int rabin_karp(string &text, string &pattern){
//retorna a posição da primeira ocorrência do padrão no texto, ou -1, se não
existir

    Hash T = Hash(text);
    Hash P = Hash(pattern);
    T.build();
    P.build();

    int l = 0, r = pattern.size()-1;
    while (r < (int)text.size())
    {
        if(T.getRangeNormal(l, r) == P.hashNormal){
            return l;
        }
        l++; r++;
    }

    return -1;
}
```

5.6 - Suffix Array nlogn + LCP Array

```
int n, sa[N], tmpsa[N], rk[N], tmprk[N], cont[N], lcp[N], inv[N];
```

```
string s;

void radix(int  k){

    memset(cont, 0, sizeof cont);
    int maxi = max(300, n);

    for(int  i=0; i<n; i++){
        cont[ (i+k)<n ?  rk[i+k] : 0 ]++;
    }

    for(int  i=1; i<maxi; i++) cont[i]+=cont[i-1];

    for(int  i=n-1; i>=0; i--){
        tmpsa[ --cont[ ( sa[i]+k ) < n   ?  rk[sa[i]+k] : 0 ] ] = sa[i];
    }
    for(int  i=0; i<n; i++) sa[i] = tmpsa[i];
}


void build_SA(){

    for(int  i=0; i<n; i++){
        rk[i] = s[i];
        sa[i] = i;
    }

    for(int  k=1; k<n; k<<=1){

        radix(k);
        radix(0);

        tmprk[sa[0]] = 0;
        int  r = 0;
        for(int  i=1; i<n; i++){
            tmprk[sa[i]] = (rk[sa[i]] == rk[sa[i-1]] && rk[sa[i]+k] ==
rk[sa[i-1]+k]) ? r : ++r;
        }

        for(int  i=0; i<n; i++){
            rk[sa[i]] = tmprk[sa[i]];
        }

        if(rk[sa[n-1]] == n-1) break;
    }

}
```

```
void build_lcp(){

    for(int  i=0; i<n; i++){
        inv[sa[i]] = i;
    }
    int  L=0;
    for(int  i=0; i<n; i++){
        if(inv[i] == 0){
            lcp[inv[i]] = 0;
            continue;
        }
        int  prev = sa[inv[i]-1];
        while(i+L<n && prev+L<n && s[i+L] == s[prev+L]) L++;

        lcp[inv[i]] = L;
        L = max(L-1, 0);
    }
}


int  solve(){
    //depende do problema
}

int main(){
    ios_base::sync_with_stdio(0); cin.tie(0);
    getline(cin, s);
    s.push_back('$');

    n = s.size();

    build_SA();
    build_lcp();

    solve();
}
```

5.7 - Suffix Array nlog^2n + LCP Array

```
//OBS: usa a struct Hash

int sa[MAXN], lcp[MAXN];

string s;
Hash H;
```

```
int getLCP(int a, int b){//pega o LCP entre o sufixo começando em a e o sufix
começando em b

    int lo = 0;
    int hi = min((int)s.size() - a, (int)s.size() - b);
    int mid;
    int ans = 0;

    while(lo <= hi){
        mid = (lo+hi)/2;
        if(H.getRangeNormal(a, a+mid-1) == H.getRangeNormal(b, b+mid-1)){
            lo = mid+1;
            ans = max(ans, mid);
        }else{
            hi = mid-1;
        }
    }

    return ans;
}

bool compareSA(int a, int b){//pega o LCP e compara o próximo caractere
    int len = getLCP(a, b);
    if(a+len == (int)s.size()) return true;
    if(b+len == (int)s.size()) return false;

    return s[a+len] < s[b+len];
}

void build_SA(){
    int tam = (int)s.size();
    for (int i = 0; i < tam; i++)
    {
        sa[i] = i;
    }
    sort(sa, sa + tam, compareSA);
}

void build_lcp(){
    lcp[0] = 0;
    for (int i = 1; i < (int)s.size(); i++)
    {
        lcp[i] = getLCP(sa[i], sa[i-1]);
    }
}

int main () {
    buildPot();//cuidado com o limite do MAXN
```

```
    cin >> s;
    H = Hash(s);
    H.build();

    build_SA();
    build_lcp();

    return 0;
}
```

5.8 - Trie Estática

```
int trie[MAXN][26];
char fim[MAXN];
int counter[MAXN];
string s;
int cnt = 2;

void add(){
    int no = 1;//1 é a raiz
    int c;

    for (int i = 0; i < (int)s.size(); i++)
    {
        c = s[i]-'a';
        if(!trie[no][c]){
            trie[no][c] = cnt++;
        }
        no = trie[no][c];
        counter[no]++;
    }
    fim[no] = 1;
}

int main(){
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cin >> s;
        add();
    }


    return 0;
}
```

5.9 - Trie Dinâmica

```
string s;

struct no{
    #define ALF 30 //depende do problema

    no *nxt[ALF];
    int cont, fim;
    char c;

    no(char k){
        c = k;
        for(int i=0; i<ALF; i++) nxt[i] = NULL;
        cont = fim = 0;
    }

    void insert(int i){
        cont++;
        if(i == s.size()){
            fim=1;
            return;
        }
        if(!nxt[s[i]-'a'])  nxt[s[i]-'a'] = new no(s[i]);

        return nxt[s[i]-'a']->insert(i+1);
    }

    void destroy(){
        for(int i=0; i<ALF; i++){
            if(nxt[i]) {
                nxt[i]->destroy();
                delete nxt[i];
            }
        }
    }

};

no *root;

int main(){
    ios_base::sync_with_stdio(0); cin.tie(0);
```

```
    root = new no('$');
    int n;
    cin >> n;
    for(int i=0; i<n; i++){
        cin >> s;
        root->insert(0);
    }

    // resolve problema

    root->destroy();
    delete root;
}
```

5.10 - Z-Algorithm

```
string s;
int z[N];

void calc_z(){

    memset(z, 0, sizeof z);

    int n = s.size();
    int l=0, r=0;

    for(int i=1; i<n; i++){
        if(i<=r)  z[i] = min(r-i+1, z[i-l]);

        while(i+z[i] < n && s[z[i]] == s[i+z[i]])
            z[i]++;

        if(i+z[i]-1 > r){
            l=i;
            r = i+z[i]-1;
        }
    }
}
```

6 - SQRT

6.1 - MO

```
struct query{
    int l, r, pos;
    query(){}
    query(int a, int b, int d){
        l = a;
        r = b;
        pos = d;
    }
};

//~ int block_size = sqrt(MAXN);

void add(int pos){
    //~ Faz alguma coisa: conta frequência, por exemplo
    //~ Adiciona o elemento v[pos] no intervalo
}

void del(int pos){
    //~ Faz alguma coisa: conta frequência, por exemplo
    //~ Remove o elemento v[pos] no intervalo
}

bool compare(query &a, query &b){
    if(a.l / block_size == b.l / block_size) return a.r < b.r;
    return a.l < b.l;
    //se o bloco do left for o mesmo, ordena pelo r, senão ordena por l
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)  //leitura do vetor de entrada
        cin >> v[i];

    int L, R;
    cin >> q;
    for (int i = 0; i < q; i++)  //leitura de query
    {
        cin >> L >> R;
        L--; R--;
        queries[i] = query(L, R, i);
    }
    sort(queries, queries+q, compare);//ordena as queries
    int currL = 0, currR = 0;

    for (int i = 0; i < q; i++)
    {
        L = queries[i].l;
```

```
        R = queries[i].r;
        while (currL < L)
            del(currL++);   //remove elemento da posição currL
        while (currR <= R)
            add(currR++);   //adiciona elemento da posição currR
        while (currL > L)
            add(--currL);   //adiciona elemento da posição currL-1
        while (currR > R+1)
            del(--currR);   //remove elemento da posição currR-1

        saida[queries[i].pos] = resposta;   //reordena a saída
    }
    for (int i = 0; i < q; i++)
        cout << saida[i] << "\n";
    return 0;
}
```

6.2 - MO em Árvore

```
//~ CONTAR QUANTOS PESOS DISTINTOS TEM NO CAMINHO DE U PRA V

struct query{
    int l, r, lca, pos;
    query(){}
    query(int a, int b, int c, int d){
        l = a;
        r = b;
        lca = c;
        pos = d;
    }
};

query queries[MAXN];
int lca[MAXN][LOG];
int valor[MAXN];
unordered_map<string, int> mapa;
unordered_map<string, int>::iterator it;
vector<int> g[MAXN];
int ini[MAXN];
int fim[MAXN];
int h[MAXN];
int ans[MAXN];
int f[MAXN];
int n, q;
vector<int> euler;
int block_size = 450;
```

```
int counter = 0;
int total = 0;
char vis[MAXN];

inline bool compare(const query &a, const query &b){
    if(a.l/block_size == b.l/block_size) return a.r < b.r;
    return a.l < b.l;
}

inline void dfs(int u, int pai){
    lca[u][0] = pai;
    for(int i = 1; i < LOG; i++)
        lca[u][i] = lca[lca[u][i-1]][i-1];

    euler.pb(u);
    int v;
    ini[u] = counter++;
    for (int i = 0; i < g[u].size(); i++)
    {
        v = g[u][i];
        if(v==pai) continue;
        h[v] = h[u]+1;
        dfs(v, u);
    }
    fim[u] = counter++;
    euler.pb(u);
}

inline int getLca(int u, int v){
    if(h[u] < h[v]) swap(u, v);
    int dist = abs(h[u]-h[v]);

    for (int i = LOG-1; i >= 0; i--)
    {
        if(dist & (1<<i))
            u = lca[u][i];
    }
    if(u==v) return u;

    for (int i = LOG-1; i >= 0; i--)
    {
        if(lca[u][i] != lca[v][i]){
            u = lca[u][i];
            v = lca[v][i];
        }
    }
    return lca[u][0];
}
```

```
inline void add(int pos){
    int u = euler[pos];
    int val = valor[u];
    if(vis[u]){
        f[val]--;
        if(f[val]==0) total--;
    }else{
        f[val]++;
        if(f[val]==1) total++;
    }
    vis[u] ^= 1;
}

inline void del(int pos){
    add(pos);
}

int main(){
    ios_base::sync_with_stdio (0);
    cin.tie (0);

    int nxtIdx=0, u, v;
    string s;
    cin >> n >> q;
    for (int i = 0; i < n; i++)
    {
        cin >> s;
        it = mapa.find(s);
        if(it == mapa.end()){
            mapa[s] = nxtIdx++;
        }
        valor[i] = mapa[s];
    }

    for (int i = 0; i < n-1; i++)
    {
        cin >> u >> v;
        u--; v--;
        g[u].pb(v);
        g[v].pb(u);
    }

    h[0] = 0;
    dfs(0, 0);
    int p;
    for (int i = 0; i < q; i++)
    {
```

```
    cin >> u >> v;
    u--; v--;
    if(ini[u] > ini[v]) swap(u, v);
    p = getLca(u, v);
    if(p==u){
        queries[i] = query(ini[u], ini[v], -1, i);
    }else{
        queries[i] = query(fim[u], ini[v], p, i);
    }
}

sort(queries, queries+q, compare);
int L, R;
int currL=0, currR=0;
for (int i = 0; i < q; i++)
{
    L = queries[i].l;
    R = queries[i].r;

    while (currR <= R)
    {
        add(currR);
        currR++;
    }
    while (currL > L)
    {
        add(currL-1);
        currL--;
    }
    while (currL < L)
    {
        del(currL);
        currL++;
    }
    while (currR > R+1)
    {
        del(currR-1);
        currR--;
    }

    if(queries[i].lca!=-1){
        add(ini[queries[i].lca]);
    }

    ans[queries[i].pos] = total;

    if(queries[i].lca!=-1){
        del(ini[queries[i].lca]);
```

```
        }
    }

    for (int i = 0; i < q; i++)
    {
        cout << ans[i] << "\n";
    }




    return 0;
}
```

6.3 - SQRT decomposition em blocos

```
int n, q;
vector<int> block[600];
int block_size = 600;
int v[100010];

int ini(int blocoAtual){ return blocoAtual*block_size; }
int fim(int blocoAtual){ return min(ini(blocoAtual+1) - 1, n-1); }

int func(int blocoAtual, int X){//calcula quantos elementos <= X tem no
blocoAtual
    int ans = upper_bound(block[blocoAtual].begin(), block[blocoAtual].end(),
X) - block[blocoAtual].begin();
    return ans;
}

void update(int pos, int val){//atualiza só o bloco afetado
    int valAntigo = v[pos];
    int blocoAtual = pos / block_size;
    v[pos] = val;

    for(int i = 0; i < block[blocoAtual].size(); i++){
        if(block[blocoAtual][i] == valAntigo){
            block[blocoAtual][i] = val;
            break;
        }
    }
    sort(block[blocoAtual].begin(), block[blocoAtual].end());//ordena o bloco
de novo
}
```

```
/*
int query(int L, int R, int X){
    int blocoL, blocoR;
    blocoL = L / block_size;
    blocoR = R / block_size;
    int pos;
    int ans = 0LL;

    for(pos = L; pos <= min(R, fim(blocoL)); pos++)
        if(v[pos] <= X) ans++;

    for (int i = blocoL+1; i <= blocoR-1; i++)
        ans += func(i, X);

    for(pos = max(pos, ini(blocoR)); pos <= R; pos++)
        if(v[pos] <= X) ans++;

    return ans;
}
*/

int query(int L, int R, int X){//retorna quantos elementos <= X tem em [L, R]
    int blocoL, blocoR;
    blocoL = L / block_size;
    blocoR = R / block_size;
    int pos;
    int ans = 0LL;
    //para blocos que não estão inteiros dentro do intervalo: percorre em O(n)
    //para blocos que estão inteiros dentr do intervalo: faz uma busca binária
pra saber quantos elementos <= X existe
    for (int i = 0; i < block_size; i++)
    {
        if(ini(i) > R) break;
        if(ini(i) >= L && fim(i) <= R) ans += func(i, X);
        else{
            for(int j=max(ini(i), L); j<=min(fim(i), R); j++) ans += (v[j] <=
X);
        }
    }
    return ans;
}

int main(){
    cin >> n >> q;

    for (int i = 0; i < n; i++)
    {
```

```
        cin >> v[i];
        block[i/block_size].pb(v[i]);//adiciona no bloco correspondente
    }
    for (int i = 0; i < block_size; i++)
    {
        if(block[i].size()==0) break;
        sort(block[i].begin(), block[i].end());//ordena cada bloco
    }

    char op;
    int L, R, X, pos, val;
    for (int i = 0; i < q; i++)
    {
        cin >> op;
        if(op=='C'){
            cin >> L >> R >> X;
            cout << query(L-1, R-1, X) << "\n";
        }else{
            cin >> pos >> val;
            update(pos-1, val);
        }
    }
    return 0;
}
```