Referência

Sumário

1 Template

Compiler

```
alias codec++='g++ -std=c++14 -static -lm -O2 -Wshadow -Wfatal-errors
-D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC -DLOCAL_DEFINE $1'
```

Template

```
#pragma GCC optimize("Ofast")
ios_base::sync_with_stdio(0);
cin.tie(0);
#ifdef LOCAL_DEFINE
    cerr << "\nTime: " << 1.0 * clock() / CLOCKS_PER_SEC << " s.\n";
#endif
```

2 Data Structure

BIT

```
const int MAXN = 1e5;
int bit[MAXN];
void update(int x,int v){
        while(x<MAXN){
                bit[x]+=v;
                x+=(x&-x);
        }
}
int sum(int x){
        int s=0;
        while(x>0){
                s+=bit[x];
                x-=(x&-x);
        }
        return s;
}
```

BIT2D

```
int bit[100000][100000],N,M;
int sum(int x,int y){
        int resp=0;
        for(int i=x;i>0;i-=(i&-i))
                for(int j=y;j>0;j-=(j&-j))
                        resp+=bit[i][j];
        return resp;
}
void update(int x,int y,int val){
        for(int i=x;i<N;i+=(i&-i))
                for(int j=y;j<M;j+=(j&-j))
                        bit[i][j]+=val;
}
```

SegTree with Lazy

```
struct SegTreeLazy{
        int tree[400000]={0},lazy[400000]={0},arr[100000];// tree e lazy 4*tam_arr
        void build(int node,int left,int right){
                if(left==right){
                        tree[node]=arr[left];
                        return ;
                }
                int mid=(left+right)/2;
                build(2*node,left,mid);
                build(2*node+1,mid+1,right);
                tree[node]= tree[2*node]+tree[2*node+1];
        }
        void update(int node,int left,int right,int l,int r,int value){
                if(lazy[node]){
                        tree[node]=(right-left+1)*lazy[node];
                        if(right!=left){
                                lazy[2*node]=lazy[node];
                                lazy[2*node+1]=lazy[node];
                        }
                        lazy[node]=0;
                }
                if(left>r or l>right)
                        return ;
                if(left>=l and r>=right){
                        tree[node]=(right-left+1)*value;
                        if(right!=left){
                                lazy[2*node]=value;
                                lazy[2*node+1]=value;
                        }
                        return ;
                }
                int mid=(left+right)/2;
                update(2*node,left,mid,l,r,value);
                update(2*node+1,mid+1,right,l,r,value);
                tree[node]=tree[2*node]+tree[2*node+1];
        }
        int sum(int node,int l,int r,int left,int right){
                if(lazy[node]){
                        tree[node]=(right-left+1)*lazy[node];
                        if(right!=left){
                                lazy[2*node]=lazy[node];
                                lazy[2*node+1]=lazy[node];
                        }
                        lazy[node]=0;
                }
                if(left>r or l>right)
                        return 0;
                if(left>=l and r>=right)
                        return tree[node];
                int mid=(left+right)/2;
                return sum(2*node,l,r,left,mid)+sum(2*node+1,l,r,mid+1,right);
        }
};
typedef struct SegTreeLazy seg;
void pointupdate(int node,int left,int right,int idx,double value){
        if(left==right){
                tree[node]=value;
                return ;
        }
        int mid=(left+right)/2;
        if(left<=idx and idx<=mid)
                pointupdate(2*node,left,mid,idx,value);
        else
                pointupdate(2*node+1,mid+1,right,idx,value);
        tree[node]=tree[2*node]+tree[2*node+1];
}
```

SegTree Iterative

```
struct SegIterative{
        int tree[200000]={0};// tree e lazy 2*tam_arr
        int n=1e5;//size Seg
        void build(){
                for(int i=n-1;i>0;i--) tree[i] = tree[i<<1] + tree[i<<1|1];
        }
        void update(int p,int val){
                for(tree[p+=n]=val;p>1;p>>=1) tree[p>>1] = tree[p]+tree[p^1];
```

```
        }
        int query(int l,int r){ // [l,r)
                int resp=0;
                for(l+=n,r+=n;l<r;l>>=1,r>>=1){
                        if(l&1) resp+=tree[l++];
                        if(r&1) resp+=tree[--r];
                }
                return resp;
        }
};
// main input
for(int i=0;i<n;i++)
        cin >> SegIterative.tree[i+n];
SegIterative.build();
```

Sparse Table

```
int table[MAXN][MAXN],arr[MAXN];
void buildSparseTable(int N){
        for(int i=0;i<N;i++)
                table[i][0]=arr[i];
        for(int j=1;(1LL<<j)<=N;j++)
                for(int i=0;(i+(1LL<<j))<=N;i++)
                        table[i][j]=min(table[i][j-1],table[i+(1LL<<(j-1))][j-1]);
}
int query(int l,int r){
        int j=log2(r-l+1);
        return min(table[l][j] , table[r-(1LL<<j)+1][j]);
}
```

Union Find

```
int find(int x){
        return (pai[x]==x)?x:pai[x]=find(pai[x]);
}
void join(int x,int y){
        x=find(x);
        y=find(y);
        if(x==y) return ;
        pai[x]=y;
}
```

Treap

```
#include <bits/stdc++.h>
using namespace std;
#define inf 0x3f3f3f3f
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
struct node {
        int x, priori, size,query;
        node *l, *r;
        node(int _x) : x(_x), priori(rng()), query(_x), size(1), l(NULL), r(NULL) {}
};
class Treap {
private:
        node* root;
        int size(node* t) { return t ? t->size : 0; }
        int query(node *t){ return t ? t->query: -inf; } //Query type
        node* refresh(node* t) {
                if (!t) return t;
                t->size = 1 + size(t->l) + size(t->r);
                t->query = t->x;
                t->query = max(query(t),max(query(t->r),query(t->l))); //Query type
                return t;
        }
        void split(node* &t, int k, node* &a, node* &b) {
                node* aux;
                if (!t) a = b = NULL;
                else if (t->x < k) {
                        split(t->r, k, aux, b);
                        t->r = aux;
                        a = refresh(t);
                }
                else {
                        split(t->l, k, a, aux);
                        t->l = aux;
                        b = refresh(t);
                }
        }
        node* merge(node* a, node* b) {
                if (!a or !b) return a ? a : b;
                if (a->priori < b->priori) {
                        a->r = merge(a->r, b);
                        return refresh(a);
                }
                else {
                        b->l = merge(a, b->l);
                        return refresh(b);
                }
        }
        node* count(node* t, int k) {
                if (!t) 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);
        }
        node* nth(node* t, int n) { // 1-idexed
                if (!t) return NULL;
                if (n <= size(t->l)) return nth(t->l, n);
                else if (n == size(t->l) + 1) return t;
                else return nth(t->r, n-size(t->l)-1);
        }
        void del(node* &t) {
                if (!t) return;
                if (t->l) del(t->l);
                if (t->r) 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(k)) return false;
                node *a, *b;
```

```
                split(root, k, a, b);
                root = merge(merge(a, new node(k)), b);
                return true;
        }
        bool erase(int k) {
                node * f = count(root, k);
                if (!f) 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(int n) {
                node* ans = nth(root, n);
                return ans ? ans->x : -1;
        }
        int query(int l, int r) { // range [l,r) -> os elementos
                if (l > r) swap(l, r);
                node *a, *b, *c, *d;
                split(root, l, a, d);
                split(d, r, b, c);
                int ans = query(b);
                root = merge(a, merge(b, c));
                return ans;
        }
};
```

Implicit Treap

```
#include <bits/stdc++.h>
using namespace std;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
struct node {
        int y, v, query, size;
        bool rev;
        node *l, *r;
        node(int _v) : v(_v), query(_v), y(rng()),
                size(1), l(NULL), r(NULL), rev(false) {}
};
/*
Para swap de string O(log N), substituir int v, por char v
N o possui query. Inserir os char's da string na treap
*/
class ImplicitTreap {
private:
        node* root;
        int size(node* t) { return t ? t->size : 0; }
        int query(node* t) { return t ? t->query : 0; }
        node* refresh(node* t) {
                if (t == NULL) return t;
                t->size = 1 + size(t->l) + size(t->r);
                t->query = t->v + query(t->l) + query(t->r);
                if (t->l != NULL) t->l->rev ^= t->rev;
                if (t->r != NULL) t->r->rev ^= t->rev;
                if (t->rev) {
                        swap(t->l, t->r);
                        t->rev = false;
                }
                return t;
        }
        void split(node* &t, int k, node* &a, node* &b) {
                refresh(t);
                node * aux;
                if (!t) a = b = NULL;
                else if (size(t->l) < k) {
                        split(t->r, k-size(t->l)-1, aux, b);
                        t->r = aux;
                        a = refresh(t);
                }
                else {
                        split(t->l, k, a, aux);
                        t->l = aux;
                        b = refresh(t);
                }
        }
        node* merge(node* a, node* b) {
                refresh(a); refresh(b);
                node* aux;
                if (!a or !b) return a ? a : b;
                if (a->y < b->y) {
                        a->r = merge(a->r, b);
                        return refresh(a);
                }
                else {
                        b->l = merge(a, b->l);
                        return refresh(b);
                }
        }
        node* at(node* t, int n) {
                if (!t) return t;
                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);
        }
        void del(node* &t) {
                if (!t) return;
                if (t->l) del(t->l);
                if (t->r) 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;
                split(root, n, a, b);
                root = merge(merge(a, new node(v)), 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);
                return ans ? ans->v : -1;
        }
        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 = query(b);
                root = merge(a, merge(b, c));
                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->rev ^= 1;
                root = merge(a, merge(b, c));
        }
};
```

Ordered Set

```
#include<bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>,rb_tree_tag,
        tree_order_statistics_node_update> ordered_set;
int32_t main(){
        ordered_set S;
        S.insert(1);
        S.insert(2);
        S.insert(5);
        S.insert(6);
        //      *S.find_by_order(pos)
        cout << *S.find_by_order(1) << '\n'; // 2
        cout << *S.find_by_order(2) << '\n'; // 5
        cout << *S.find_by_order(4) << '\n'; // 0 -> not find
        //              S.find_by_order(size) size do ordered_set
        cout<<((end(S)==S.find_by_order(3))?"TRUE":"FALSE")<<endl;
        //      S.order_of_key(val)
        cout << S.order_of_key(3) << '\n'; // 2
        cout << S.order_of_key(-1) << '\n';// 0
        cout << S.order_of_key(6) << '\n'; // 3
        //retorna val<=*S.find_by_order(S.order_of_key(val))
        cout << *S.find_by_order(S.order_of_key(3)) << '\n';
}
```

3 Dynamic Programming

DigitDP

```
int dp[20][1000][2];//apenas para 1 digito
//varicoes apenas no retorno e no segundo parametro
int digitDP(int idx, int sum, int can, vector<int> &digit){
        if(idx == (int)digit.size())
                return sum%mod;
        if(dp[idx][sum][can] != -1)
                return dp[idx][sum][can];
        int ans = 0;
        for(int i = 0; i < 10; i++)
          if(can or i <= digit[idx])
            ans =(ans+ digitDP(idx+1,sum+i,can or i<digit[idx], digit)) % mod;
        return dp[idx][sum][can] = ans % mod;
}
int query(int x){
        memset(dp, -1, sizeof(dp));
        vector<int> digit;
        if(x==0) digit.push_back(0);
        while(x){
                digit.push_back(x%10);
                x /= 10;
        }
        reverse(digit.begin(), digit.end());
        return digitDP(0, 0, 0, digit);
}
```

DP tree

```
int dp(int v,bool flag){
        if(pd[v][flag]!=-1) return pd[v][flag];
        int cas1=0,cas2=0;
        if(flag) cas1=A[v];
        for(int u:G[v]){
                if(u!=pai[v]){
                        pai[u]=v;
                        cas1+=dp(u,false);
                        cas2+=dp(u,true);
                }
        }
        if(flag) return pd[v][flag]=max(cas1,cas2);
        return pd[v][flag]=cas2;
}
```

DP Broken Profile

```
int n, m;
vector < vector<long long> > d;
void calc (int x = 0, int y = 0, int mask = 0, int next_mask = 0)
{
    if (x == n)
        return;
    if (y >= m)
```

```
            d[x+1][next_mask] += d[x][mask];
        else
        {
            int my_mask = 1 << y;
            if (mask & my_mask)
                calc (x, y+1, mask, next_mask);
            else
            {
                calc (x, y+1, mask, next_mask | my_mask);
                if (y+1 < m && ! (mask & my_mask) && ! (mask & (my_mask << 1)))
                    calc (x, y+2, mask, next_mask);
            }
        }
}
int main()
{
    cin >> n >> m;
    d.resize (n+1, vector<long long> (1<<m));
    d[0][0] = 1;
    for (int x=0; x<n; ++x)
        for (int mask=0; mask<(1<<m); ++mask)
            calc (x, 0, mask, 0);
    cout << d[n][0];
}
```

Kadane 2D

```
int A[N+1][N+1],pd[N+1][N+1];
for(int i=1;i<=N;i++)
        for(int j=1;j<=N;j++){
                scanf("%lld",&A[i][j]);
                pd[i][j]=pd[i][j-1]+A[i][j];
        }
int ans=0;
for(int i=1;i<=N;i++)
        for(int j=i+1;j<=N;j++){
                int sum=0;
                for(int k=1;k<=N;k++){
                        sum += pd[k][j] - pd[k][i-1];
                        if(sum<0) sum = 0;
                        ans = max(ans, sum);
                }
        }
```

Knapsack-with-backtracking

```
int n,c;
vector<pii> v;
int res,aux;
double c2,aux2;
void bt(int i){
        if(i == n) return;
        aux2 = 0; c2 = c;
        for(int j=i; j<n && c2; j++){
                if(v[j].fi <= c2){
                        c2 -= v[j].fi; aux2 += v[j].se;
                } else {
                        aux2 += (v[j].se*c2)/v[j].fi;
                        c2 = 0;
                }
        }
        if(aux2 + aux <= res) return;
        if(v[i].fi <= c){
                c -= v[i].fi;
                aux += v[i].se;
                if(aux > res) res = aux;
                bt(i+1);
                aux -= v[i].se;
                c += v[i].fi;
        }
        bt(i+1);
}
int32_t main(){
        ios::sync_with_stdio(false);cin.tie(0);
        cin>>n>>c;
        FOR(i,0,n){
                int x,y; cin>>x>>y;
                v.pb({x,y});
        }
        sort(all(v),[](pii a, pii b){
                return (a.se+0.0)/a.fi > (b.se+0.0)/b.fi;
        });
        bt(0);
        cout<<res<<endl;
        return 0;
}
```

Knapsack

```
int peso[MAXobj],valor[MAXobj],tab[MAXobj][MAXpeso];
int knapsack(int obj, int aguenta){
        if(tab[obj][aguenta]>=0)
                return tab[obj][aguenta];
        if(obj==N or !aguenta)
                return tab[obj][aguenta]=0;
        int nao_coloca=knapsack(obj+1, aguenta);
        if(peso[obj]<=aguenta){
                int coloca=valor[obj]+knapsack(obj+1, aguenta-peso[obj]);
                return tab[obj][aguenta]=max(coloca, nao_coloca);
        }
        return tab[obj][aguenta]=nao_coloca;
}
```

LCS iterativa

```
#include<bits/stdc++.h>
using namespace std;
void mmax(int &a,int b){
        a = max(a,b);
}
int32_t main(){
    cin >> s1 >> s2;
    int sz1=(int)s1.size(),sz2=(int)s2.size();
```

```
        for(int i=0;i<=sz1;i++)
                for(int j=0;j<=sz2;j++){
                        if(i) mmax(pd[i][j],pd[i-1][j]);
                        if(j) mmax(pd[i][j],pd[i][j-1]);
                        if(i<sz1 and j<sz2)
                                if(s1[i]==s2[j])
                                        mmax(pd[i+1][j+1],pd[i][j]+1);
                }
        string out="";
        int i=sz1,j=sz2,p=pd[sz1][sz2];
        while(p--){
                while(pd[i][j]==pd[i-1][j]) i--;
                while(pd[i][j]==pd[i][j-1]) j--;
                i--,j--;
                out+=s1[i];
        }
        reverse(all(out));
        cout << out << '\n';
}
```

LCS

```
//Dois vetores A e B
int dp(int a,int b){
        if(a>=N or b>=M) return 0;
        if(pd[a][b]!=-1) return pd[a][b];
        if(A[a]==B[b])
                return pd[a][b]=max(dp(a+1,b+1)+1,max(dp(a+1,b),dp(a,b+1)));
        return pd[a][b]=max(dp(a+1,b),dp(a,b+1));
}
//Um vetor A
int pd(int id,int v){
        if(id<0) return 0;
        if(dp[id][Map[v]]!=-1) return dp[id][Map[v]];
        if(A[id]<=v)
                return dp[id][Map[v]]=max(pd(id-1,A[id])+1,pd(id-1,v));
        return dp[id][Map[v]]=pd(id-1,v);
}
```

LIS

```
vector<char> lis(string &str){
        vector<char> pilha,resp;
        int pos[300002],pai[300002];
        for(int i=0;i<str.size();i++){
        vector<char>::iterator it=upper_bound(pilha.begin(),pilha.end(),str[i]);
                                //lower_bound -> elementos distintos
                int p=it-pilha.begin();
                if(it==pilha.end()) pilha.push_back(str[i]);
                else *it=str[i];
                pos[p]=i;
                if(p==0) pai[i]=-1;
                else pai[i]=pos[p-1];
        }
        int p=pos[pilha.size()-1];
        while(p>=0){
                resp.push_back(str[p]);
                p=pai[p];
        }
        reverse(resp.begin(),resp.end());
        return resp;
}
```

SubConjuntos

```
v[0]=1;
for(auto valor:Valores)
        FORI(int i=valor_MAX-valor;i>=0;i--)
                if(v[i])
                        v[i+valor]++;
```

TSP

```
int tsp(int bitmask,int id){//O((2^n)*(n^2))
        if(memo[bitmask][id]!=-1)
                return memo[bitmask][id];
        if(bitmask==((1<<N)-1))
                return dist[id][0];
        int ans=INT_MAX;
        for(int i=0;i<N;i++)
                if(!(bitmask&(1<<i)))
                        ans=min(ans,tsp((bitmask|(1<<i)),i)+dist[id][i]);
        return memo[bitmask][id]=ans;
}
```

4 Geometry

Convex Hull

```
#define X first
#define Y second
typedef pair<int, int> ii;
int cross(ii O, ii A, ii B){
        return (((A.X - O.X) * (B.Y - O.Y)) - ((A.Y - O.Y) * (B.X - O.X)));
}
vector<ii> ConvexHull(vector<ii> P){
        if(P.size() <= 1) return P;
        vector<ii> H(2*P.size());
        int k = 0;
        sort(P.begin(), P.end());
        //lower hull
        for(int i = 0; i < P.size(); i++){
                while(k >= 2 and cross(H[k-2], H[k-1], P[i]) < 0) k--;
                H[k++] = P[i];
        }
        //upper hull
        for(int i = P.size()-2, l = k + 1; i >= 0; i--){
                while(k >= l and cross(H[k-2], H[k-1], P[i]) < 0) k--;
                H[k++] = P[i];
        }
        H.resize(k-1);
        return H;
}
int main(){
        int n, x, y;
        vector<ii> P;
        cin >> n;
        while(n--){
                cin >> x >> y;
                P.push_back({x, y});
        }
        vector<ii> H = ConvexHull(P);
        for(int i = 0; i < H.size(); i++)
                cout << H[i].X << ' ' << H[i].Y << '\n';
}
```

Pontos e Retas

```
#define x first
#define y second.first
#define z second.second
typedef pair<double,pair<double,double> > point;
//[x,y,1] -> ponto
//[x,y,z] -> reta
// a*x+b*y+c*z=0
point reta(point a,point b){
        point resp;
        resp.x = a.y*b.z-a.z*b.y;
        resp.y = a.z*b.x-a.x*b.z;
        resp.z = a.x*b.y-a.y*b.x;
        return resp;
}
point intercessao(point a,point b){
        point i = reta(a,b);
        if(i.z!=0){//reduz ao ponto, i.z==0 -> retas paralelas
                i.x/=i.z;
                i.y/=i.z;
                i.z/=i.z;
        }
        return i;
}
reta1 = reta(p1,p2);
intercessao1= intercessao(reta1,reta2);
//se existe intercessao no intervalor p1 p2
if(intercessao1.y>0.0 and intercessao1.y<min(p1.y,p2.y) and
intercessao1.x>min(p1.x,p2.x) and intercessao1.x<max(p1.x,p2.x))
```

Resumo Geometria

```
//Distancia entre dois pontos
double dist=sqrt((X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2));
//Equacao da Circunferencia (Centro (a,b) e Raio r)
(x-a)^2+(y-b)^2=r^2
//Check se um circulo esta dentro do outro
if(dist<=(R1-R2))
//Check se um circulo esta fora do outro
if(dist>=(R1+R2))
//Condicao de existencia de um triangulo de lados A,B,C
if((abs(A-B)<C and C<(A+B)) and (abs(A-C)<B and B<(A+C)) and (abs(B-C)<A and A<(B+C)))
//Formulas para um triangulo com lados a,b,c
   Semi-Perimetro => p = (a+b+c)/2
             Area => A = sqrt(p(p-a)(p-b)(p-c))
             Area => A = bc.sin(alpha)/2
          Altura => h = 2A/b
    Raio Inscrito => r = A/p
Raio Curcunscrito => R = (abc)/(4A)
```

5 Graph

BFS

```
void bfs(int v){
        queue<int> q;
        q.push(v);
        cor[v]=1;
        while(!q.empty()){
                int u=q.front();
                q.pop();
                for(int w:G[u])
                if(!cor[w]){
                        q.push(w);
                        cor[w]=1;
                        dist[w]=dist[u]+1;
                }
        }
}
```

DFS

```
void dfs(int v){
        cor[v]=1;
        for(int u:G[v])
                if(!cor[u]){
                        dist[u]=dist[v]+1;
                        dfs(u);
                }
}
```

DFS bridge

```
void dfs(int u,int p){
        cor[u]=1;
        d[u]=low[u]=tempo++;
        for(int &v:G[u])
                if(!cor[v]){
                        dfs(v,u);
                        low[u]=min(low[u],low[v]);
                        if(low[v]>d[u])
                                //ponte encontrada se entrar aqui
                }else if(v!=p)
                        low[u]=min(low[u],d[v]);
}
```

Dijkstra Paridade

```
int dist[100002][3];
// paridade da chegada no ponto,%3 (quantidade de arestas usadas)
//Dijkstra com paridade ate o destino, acessar dist[z][X] na main
void dijkstra(int v,int z){
        memset(dist,63,sizeof dist);
        priority_queue< pair<int,pair<int,int> > > pq;
        dist[v][0]=0;
        pq.push({0,{v,0}});
        while(!pq.empty()){
                int d=-pq.top().f;
                int u=pq.top().s.f;
                int p=pq.top().s.s;
                pq.pop();
                if(u==z) continue;
                if(d>dist[u][0] and d>dist[u][1] and d>dist[u][2]) continue;
                for(pair<int,int> j:G[u]){
                        int w=j.s,_d=j.f;
                        if(p==2){//max paridade
                                if(dist[w][0]>_d+dist[u][2]){
                                        dist[w][0]=_d+dist[u][2];
                                        pq.push({-dist[w][0],{w,0}});
                                }
                        }else{
                                if(dist[w][p+1]>_d+dist[u][p]){
                                        dist[w][p+1]=_d+dist[u][p];
                                        pq.push({-dist[w][p+1],{w,p+1}});
                                }
                        }
                }
        }
}
```

Dinic

```
const int MAXN = 3e3;
struct Edge{
        int v,flow,c;
        Edge(){};
        Edge(int _v,int _c ,int _flow){
                v = _v; c = _c; flow = _flow;
        }
};
typedef struct Edge edge;
vector<edge> Edges;
vector<int> G[MAXN];
int nivel[MAXN],send[MAXN];
inline void add_Edge(int u,int v,int cap,int rev = 0){
        Edges.push_back(Edge(v ,cap ,0));
        G[u].push_back((int)Edges.size() - 1);
        Edges.push_back(Edge(u ,rev ,0));
        G[v].push_back((int)Edges.size() - 1);
}
inline bool bfs(int s,int t){
        memset(nivel,-1,sizeof nivel);
        nivel[s] = 0;
        queue<int> q;
        q.push(s);
        while(!q.empty()){
                int u = q.front();
                q.pop();
                for(auto &w:G[u])
                        if(nivel[Edges[w].v]<0 and Edges[w].flow < Edges[w].c ){
                                nivel[Edges[w].v] = nivel[u]+1;
```

```
                        q.push(Edges[w].v);
                }
        }
        return (nivel[t]>=0);
}
int sendFlow(int s,int t,int flow){
        if(s==t) return flow;
        for(int &i= send[s];i< (int)G[s].size(); i++){
                int e = G[s][i];
                if(nivel[Edges[e].v]==(nivel[s]+1) and Edges[e].flow<Edges[e].c){
                        if(int a = sendFlow(Edges[e].v, t,
                        min(flow, Edges[e].c - Edges[e].flow))){
                                Edges[e].flow += a;
                                Edges[e^1].flow -=a;
                                return a;
                        }
                }
        }
        return 0;
}
inline int Dinic(int s, int t){
        int mf=0;
        while(bfs(s,t)){
                memset(send,0,sizeof send);
                while(int flow = sendFlow(s, t, INT_MAX))
                        mf+= flow;
        }
        return mf;
}
```

FloodFill

```
auto MAT[N][M];
int dr[]={0,0,1,-1};
int dc[]={1,-1,0,0};
int floodfill(int r,int c,auto k,auto p){
        if(r>=N or r<0 or c>=M or c<0)
                return 0;
        if(MAT[r][c]!=p)
                return 0;
        int ans=1;
        for(int i=0;i<4;i++)
                ans+=floodfill(r+dr[di],c+dc[di],k,p);
        return ans;
}
```

Floyd Warshall

```
memset(dist,63,sizeof dist);
for(int k = 1;k <= n;k++)
        for(int i = 1;i <= n;i++)
                for(int j = 1;j <= n;j++)
                        dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
```

Ford Fulkerson

```
const int MAX = 1e4;
struct edge{
        int v, f, c;
        edge(){}
        edge(int _v, int _f, int _c){
                v = _v, f = _f, c = _c;
        }
};
vector<edge> edges;
vector<int> G[MAX];
int tempo = 1, cor[MAX], pai[MAX],N,M;
void add_edge(int u, int v, int cp, int rc){
        edges.push_back(edge(v, 0, cp));
        G[u].push_back(edges.size()-1);
        edges.push_back(edge(u, 0, rc));
        G[v].push_back(edges.size()-1);
}
int dfs(int s, int t, int f){
        if(s == t) return f;
        cor[s] = tempo;
        for(int e : G[s])
        if(cor[edges[e].v] < tempo and edges[e].c-edges[e].f > 0)
                if(int a = dfs(edges[e].v, t, min(f, edges[e].c-edges[e].f))){
                        edges[e].f += a;
                        edges[e^1].f -= a;
                        return a;
                }
        return 0;
}
int MaxFlow(int s, int t){
        int mf = 0;
        while(int a = dfs(s, t, inf))
                mf += a, tempo++;
        return mf;
}
int main(){
        int s=0,t=1000;// pontos de base para a passagem do fluxo
        REP(i,N)
                add_edge(s,i+1,1,0);//pontos de entrada
        REP(i,M)
                add_edge(i+N+1,t,1,0);//pontos de saida
        REP(i,N)
                REP(j,M)
                        if(abs((A[i])-B[j])<=1){
                                add_edge(i+1,j+N+1,1,0);//pontos intermediarios
                        }
        cout << MaxFlow(s,t) << '\n';//Chamada da funcao
}
```

Kahn

```
vector<int> G[50002];
int grau[50002];
int main(){
        int N,M;
        cin >> N >> M;
        while(M--){
                int a,b;
                cin >> a >> b;
                G[a].push_back(b);
                grau[b]++;
```

```
        }
        vector<int> ts;
        set<int> S;
        for(int i=0;i<N;i++)
                if(!grau[i]) S.insert(i);
        int b=0;
        while(!S.empty()){ //Prioriza os menores valores
                int a=*S.begin();
                ts.push_back(a);
                S.erase(S.begin());
                for(auto &u:G[a]){
                        grau[u]--;
                        if(!grau[u])
                                S.insert(u);
                }
        }
        if((int)ts.size()<N){
                cout << "*\n";
        }else{
                for(int i=0;i<ts.size();i++)
                        cout << ts[i] << '\n';
        }
}
```

Kosaraju

```
int N,M,t,dist[MAXN],gr[MAXN],ciclo,out[MAXN],cor[MAXN];
vi G[MAXN],Gl[MAXN];
stack<int> s;
//OBS: o pedido da questao ira alterar os dados retirados das dfs`s
void dfs1(int v){
        cor[v]=1;
        for(int u:G[v])
                if(!cor[u])
                        dfs1(u);
        dist[v]=t++;
        s.push(v);
}
void dfs2(int v,int di){
        cor[v]=0;
        gr[v]=ciclo;
        for(int u:Gl[v])
                if(cor[u] and di>dist[u])
                        dfs2(u,di);
}
int main(){
        cin >> N >> M;
        while(M--){
                int a,b;
                cin >> a >> b;
                G[b].pb(a);
                Gl[a].pb(b);
        }
        FOR(i,1,N+1)
                if(!cor[i])
                        dfs1(i);
        while(!s.empty()){
                int k=s.top();
                s.pop();
                if(cor[k])
                        dfs2(k,dist[k]),ciclo++;
        }
}
```

Kruskal

```
#define f first
#define s second
int find(int a){
        return (pai[a]==-1)?a:pai[a]=find(pai[a]);
}
void join(int a,int b){
        pai[find(a)]=find(b);
}
int main(){
        priority_queue<T,vector<T>,greater<T>> pq;
        while(!pq.empty()){
                iii a=pq.top(); // iii == pair<int,pair<int,int>>  | T == iii
                pq.pop();
                if(find(a.s.f)!=find(a.s.s)){
                        join(a.s.f,a.s.s);
                        mst.push_back(a);//minimum spanning tree
                }
        }
}
```

Kuhn

```
//Minimum edge cover == Maximum Cardinality Bipartite Matching == MVC
//MIS(Maximum Independent Set)+MVC(max matching) = N(n s)
bool kuhn(int u)//max matching
{
        if(cor[u] == tempo)
                return 0;
        cor[u] = tempo;
        //random_shuffle(G[u].begin(), G[u].end(), [](int x){ return rand() % x; });
        for(const int &v : G[u])
                if(!b[v] or kuhn(b[v]))
                        return b[v] = u;
        return 0;
}
int main(){
        tempo = 1;
        int ans = 0;
        for(int i = 1; i <= na; i++)
                ans += kuhn(i), tempo++;
}
```

LCA

```
int nivel[200002],ancestral[200002][20],table[200002][20];
vector< pair<int,int> > MST[200002];
void dfs(int v){
        for(pair<int,int> u:MST[v])
                if(nivel[u.second]==-1){
                        ancestral[u.second][0]=v;
                        table[u.second][0]=u.first;
```

```
                        nivel[u.second]=nivel[v]+1;
                        dfs(u.second);
                        sz[v]+=sz[u];
                }
        sz[v]++;
}
pair<int,int> LCA(int u,int v){
        if(nivel[u]<nivel[v]) swap(u,v);
        int m=0;
        for(int i=19;i>=0;i--)
                if(nivel[u]-(1<<i) >= nivel[v]){
                        m=max(m,table[u][i]);
                        u=ancestral[u][i];
                }
        if(u==v) return {v,m};
        for(int i=19;i>=0;i--)
                if(ancestral[u][i]!=-1 and ancestral[u][i]!=ancestral[v][i]){
                        m=max(m,max(table[u][i],table[v][i]));
                        u=ancestral[u][i],v=ancestral[v][i];
                }
        return {ancestral[u][0],max(m,max(table[u][0],table[v][0]))};
}
int main(){
        memset(nivel,-1,sizeof nivel);
        memset(pai,-1,sizeof pai);
        memset(ancestral,-1,sizeof ancestral);
        //input, grafo
        dfs(1);//grafo 1-idexado
        for(int i = 1; i<20; ++i)
                for(int j=1; j<=N; ++j)
                if(ancestral[j][i-1]!=-1){
                        ancestral[j][i] = ancestral[ancestral[j][i-1]][i-1];
                        table[j][i]=max(table[ancestral[j][i-1]][i-1],
                                        table[j][i-1]);
                }
        //Query -> elemento propagado na sparse table "table",k.second
        pair<int,int> k=LCA(Arestas[m].second.first,Arestas[m].second.second);
}//dist entre dois pontos com LCA nivel[a]+nivel[b]-2*nivel[lca]
// N-sz[aux1]-sz[aux2]
```

6 Math

Crivo

```
int primos[m+1];//m = valor maximo desejado
void crivo(){
        primos[1]=1;
        for(int i=2;i<=m;i++)
                if(primos[i]==0){
                        for(int j=2;i*j<=m;j++)
                                primos[i*j]=i;
                }
}
```

Euclides Estendido

```
// a*x + b*y == gcd(a,b)*K
int x,y;
int euclidesEst(int a,int b){
        if(a==0){
                x=0,y=1;
                return b;
        }
        int g = euclidesEst(b%a,a);
        int x1 = x,y1 = y;
        y1 = x;
        x1 = y - (b/a)*x;
        x = x1;
        y = y1;
        return g;
}
```

Exponenciação Rápida

```
const int mod = 1e9+7;
// (base^exp)%mod
int fast_expo(int base,int exp){
        if(exp==0) return 1;
        if(exp==1) return base;
        int ans=fast_expo(base,exp/2);
        ans= (ans*ans)%mod;
        if(exp&1) ans=(ans*base)%mod;
        return ans;
}
```

GCD

```
__gcd(a,b);
int gcd(int a,int b){
        return (b==0?a:gcd(b,a%b));
}
int lcm(int a,int b){
        return (a*b/gcd(a,b));
}
```

Inverso Multiplicativo

```
// inv(x) = x^(phi(m)-1) mod m, se x e m s o coprimos
int inv(int x, int mod){
        if(__gcd(x,mod)!=1) return -1; // n o existe
        int _phi = phi(mod) - 1;
        return fast_expo(x,_phi,mod);
}
// inv(x) = x^(m-2) mod m, se m    primo
int inv(int x,int mod){
        return fast_expo(x,m-2,mod);
}
/* Linear */
const int MOD =1e9+7;
int inv[MAXN];
inv[1] = 1;
for ( int i = 2; i < MAXN; i++ ) {
    inv[i] = (-(MOD/i) * inv[MOD%i] ) % MOD;
    inv[i] = inv[i] + MOD;
}
```

Mobius

```
const int MAXN=5e5;
int mob[MAXN+1];
bool vist[MAXN+1];
inline void mobius(){
        for(int i = 1; i < MAXN; i++)
                mob[i] = 1;
        for(int i = 2; i < MAXN; i++){
                if(vist[i]) continue;
                for(int j = i; j < MAXN; j += i){
                        vist[j] = true;
                        mob[j] *= -1;
                        if((j/i)%i==0)
                                mob[j] = 0;
                }
        }
}
```

Pollard Rho with Miller Rabin

```
#include<bits/stdc++.h>
using namespace std;
#define fi first
#define se second
#define int long long
long long llrand(){
        long long tmp = rand();
        return (tmp << 31) | rand();
}
long long add(long long a, long long b, long long c){
        long long ans = (a + b) % c;
    if(ans < 0) ans += c;
    return ans;
}
long long mulmod(long long a, long long b, long long c){
        long long ans = 0;
        while(b){
                if(b & 1) ans = add(ans, a, c);
                a = add(a, a, c);
                b /= 2;
        }
        return ans;
}
long long rho(long long n){
    if(n % 2 == 0) return 2;
    long long d = n;
    while(d == n){
        long long c = llrand() % n, x = llrand() % n, y = x;
        do{
            x = add(mulmod(x, x, n), c, n);
            y = add(mulmod(y, y, n), c, n);
            y = add(mulmod(y, y, n), c, n);
            d = __gcd(abs(x - y), n);
        }while(d == 1);
    }
    return d;
}
long long fexp(long long a, long long b, long long c){
        long long ans = 1;
        while(b){
                if(b & 1) ans = mulmod(ans, a, c);
                a = mulmod(a, a, c);
                b /= 2;
        }
        return ans;
}
bool miller(long long a, long long n){
    if (a >= n) return true;
    long long s = 0, d = n - 1;
    while(d%2 == 0 and d) d >>= 1, s++;
    long long x = fexp(a, d, n);
    if(x == 1 or x == n - 1) return true;
    for(int r = 0; r < s; r++, x = mulmod(x, x, n)){
        if (x == 1) return false;
        if (x == n-1) return true;
    }
    return false;
}
bool isprime(long long n){
    int base[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
    for(int i = 0; i < 12; i++)
        if(!miller(base[i], n))
            return false;
    return true;
}
vector<long long> fac;
void factors(long long n) // encontrar os fatores primos de N
{// Usar Miller-Rabin para testar se N    primo
    if(n == 1) return;
    if(isprime(n)){ fac.push_back(n); return; }
    long long d = rho(n);
    factors(d);
    factors(n / d);
```

```
}
main(){
        srand(time(0));
        int N;
        scanf("%lld",&N);
        factors(N);
        for(auto f:fac)
                printf("%lld ",f);
        printf("\n");
}
```

Totiente Euler

```
const int MAXN = 1e6+1;
void crivophi(){ //todos os phi no crivo
        for(int i=1;i<MAXN;i++)
                phi[i] = i;
        for(int i=2;i<MAXN;i++){
                if(phi[i] == i)
                        for(int j = i;j <MAXN;j += i) {
                                phi[j] /= i;
                                phi[j] *= i - 1;
                        }
        }
}
int phi(int n){ //o valor de apenas 1
        auto f = fatorar(n);
        int res = 1;
        for(auto x: f){
                int fator = x.first;
                int exp = x.second;
                res *= fast_expo(fator,exp-1);
                res *= fator-1;
        }
        return res;
}
```

7 Programming Techniques

Contagem Inversões

```
#define vi vector<int>
#define inf 0x3f3f3f3f
#define pb push_back
#define FOR(i,a,N) for(int i=a;i<N;i++)
#define REP(i,N) FOR(i,0,N)
int MergeSort(vi &A){
        if(A.size()<=1) return 0;
        vi B,C;
        REP(i,A.size()/2)
                B.pb(A[i]);
        FOR(i,A.size()/2,A.size())
                C.pb(A[i]);
        int ans=MergeSort(B)+MergeSort(C);
        B.pb(inf);
        C.pb(inf);
        int b=0,c=0;
        REP(i,A.size()){
                if(C[c]<B[b]){
                        A[i]=C[c++];
                        ans+=B.size()-b-1;
                }else
                        A[i]=B[b++];
        }
        return ans;
}
```

CoutingSort

```
int vet[max];
int frec[max];//vetor da frequencia dos elementos
void CoutingSort(){
        for(int i=0,c=0; i<max; i++)
                while(frec[i])
                        vet[c++]=i,frec[i]--;//coloca o valor no local certo
}
```

Max Sum

```
int max_sum(vector<int> &s){
        int resp=0,maior=0;
        for(int i=0;i<s.size();i++){
                maior=max(0,maior+s[i]);
                resp=max(resp,maior);
        }
        return resp;
}
```

Max Sum 2D

```
for(int i=1;i<=N;i++)
        for(int j=1;j<=M;j++){
                scanf("%d",&A[i][j]);
                dp[i][j]=dp[i-1][j]+dp[i][j-1]-dp[i-1][j-1]+A[i][j];
                for(int k=0;k<j;k++)
                        for(int l=0;l<i;l++)
                                saida = max(saida,
                                        dp[i][j]-dp[i][k]-dp[l][j]+dp[l][k]);
        }
```

prefix Sum 2D

```
int pref[n][m],arr[n][m];
void build(){
pref[1][1] = arr[1][1];
for(int i=2;i<=m;i++)
        pref[1][i] = pref[1][i-1]+arr[1][i];
for(int i=2;i<=n;i++)
        pref[i][1] = pref[i-1][1]+arr[i][1];
for(int i=2;i<=n;i++)
        for(int j=2;j<=m;j++)
                pref[i][j] = pref[i-1][j]+pref[i][j-1]-pref[i-1][j-1]+arr[i][j];
}
```

8 String

KMP

```
int arr[10002];
string txt,pattern;
void build(){
        int i=0,j=1;
        while(j<pattern.size()){
                if(pattern[i]==pattern[j])
                        arr[j]= ++i;
                else{
                        i=0;
                        if(pattern[i]==pattern[j])
                                arr[j]=++i;
                }
                j++;
        }
}
int matching(){
        int i=0,j=0,cont=0;
        while(j<txt.size()){
                if(pattern[i]==txt[j])i++,j++;
                else if(i)i=arr[i-1];
                else j++;
                if(i==pattern.size())//matching na posicao return j-M
                        cont++;//quantidade de matching's
        }
        return cont;
}
int32_t main(){
        cin >> pattern >> txt;
        build();
        cout << matching() << '\n';
}
```

Manacher

```
const int MAX = 100000;
int lps[2*MAX+5];
char s[MAX];//input
int manacher(){
    int n = strlen(s);
    string p(2*n+3,'#');
    p[0] = '^';
    for(int i = 0; i < n; i++)
        p[2*(i+1)] = s[i];
    p[2*n+2] = '$';
    int k = 0, r = 0, m = 0;
    int l = p.length();
    for(int i = 1; i < l; i++){
        int o = 2*k - i;
        lps[i] = (r > i) ? min(r-i, lps[o]) : 0;
        while(p[i + 1 + lps[i]] == p[i - 1 - lps[i]])
            lps[i]++;
        if(i + lps[i] > r) k = i, r = i + lps[i];
```

```
        m = max(m, lps[i]);
    }
    return m;
}
```

String Hashing

```
//s[i]*p^i mod m
long long compute_hash(string const& s) {
    const int p = 911382323;
    const int m = 972663749;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) * p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
```

Trie

```
// URI - 2087 - Conjuntos Bons e Ruins
#include<bits/stdc++.h>
using namespace std;
struct NodeTrie{
        map<int,NodeTrie*> filho;
        int fim;
        NodeTrie(){
                fim = 0;
        }
};
bool flag2;
inline void insert(NodeTrie *root, string str){
        NodeTrie *node = root;
        for(auto c:str){
                int id = c-'a';
                if(node->filho.find(id)==node->filho.end()){
                        node->filho[id] = new NodeTrie();
                        flag2=true;
                }
                node = node->filho[id];
        }
        node->fim++;
}
int cont=0;
bool flag;
void busca(NodeTrie *node){
        if(flag) return ;
        cont+=(node->fim);
        if(cont==2){
                flag=true;
                return ;
        }
        for(auto it = node->filho.begin();it!=node->filho.end();it++)
                busca(it->second);
        cont-=(node->fim);
}
```

```
int N;
bool cmp(string s,string s2){
        return (int)s.size()>(int)s2.size();
}
int main(){
        while(scanf("%d",&N) and N){
                cont=0;
                NodeTrie *root = new NodeTrie();
                vector<string> S;
                for(int n=0;n<N;n++){
                        string s="";
                        char c[102];
                        scanf("%s",c);
                        s=c;
                        S.push_back(s);
                }
                sort(S.begin(),S.end(),cmp);
                for(auto &s:S){
                        flag2=false;
                        insert(root,s);
                        cont+=flag2;
                }
                if(cont==N){
                        puts("Conjunto Bom");
                }else
                        puts("Conjunto Ruim");
        }
}
```

Trie Static

```
int x[MAX_QUANTIDADE_ELEMENTOS][TAMANHO_ALFABETO]
inline void build(string &s){
        int i = 0, v = 0;
        for(int i=0;i < s.size();i++){
                if(x[v][s[i]-'a'] == 0)//s[i]-'A'
                        v = x[v][s[i]-'a'] = id ++;
                else
                        v = x[v][s[i]-'a'];
        }
        // end string in v
}
```

Trie Vector

```
struct Trie{
        map<char,int> filho;
        int cont;
        Trie(){
                cont=0;
        }
};
vector<Trie> trie;
inline void add(string &adc){
        root=0;
        for(auto &c:adc){
                if(trie[root].filho.count(c)==0){
                        trie[root].filho[c] = trie.size();
                        trie.push_back(Trie());
                }
                root = trie[root].filho[c];
        }
        trie[root].cont++;
}
inline void init(){
        trie.clear();
        trie.push_back(Trie());
}
```

9 Miscelania

Matrix

```
#define int long long
struct matrix{
        int n,m;
        vector<vector<int>>M;
        matrix(int _n,int _m){
                n = _n;
                m = _m;
                M = vector<vector<int>>(n,vector<int> (m));
        }
        matrix friend operator*(const matrix &m1,const matrix &m2){
                matrix result(m1.n,m2.m);
                for(int i=0;i< m1.n;i++)
                        for(int j=0;j<m2.m;j++)
                                for(int k=0;k<m1.m;k++)
                                        result.M[i][j]+=(m1.M[i][k]*m2.M[k][j]); //MOD
                return result;
        }
        matrix operator^(int exp){
                matrix resp(n,m);
                for(int i = 0; i<n ;i++) resp.M[i][i] = 1;
                matrix aux(n,m);
                aux.M = M;
                while(exp){
                        if(exp&1) resp = resp*aux,exp--;
                        else aux = aux*aux,exp/=2;
                }
                return resp;
        }
        void print(){
                for(int i=0;i<n;i++){
                        for(int j=0;j<m;j++)
                                printf("%lld ",M[i][j]);
                        puts("");
                }
        }
};
int32_t main(){
        matrix base(2,2);
        base.M[0][1] = base.M[1][0] = base.M[1][1] = 1;
        matrix Fib(2,1);
        Fib.M[0][0] = 1;//0 - base de Fib
        Fib.M[1][0] = 1;
        for(int i=0;i<=10;i++){
                matrix f = base^i;
                matrix fib = f*Fib;
                printf("%lld: %lld\n",i,fib.M[0][0]);
        }
}
```

Gaussian Elimination

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define pb push_back
```

```cpp
#define inf 0x3f3f3f3f
const long double EPS =1e-9;
int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;
    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;
        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }
    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }
    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return inf;
    return 1;
}
int n,m,a;
int32_t main(){
        cin >> n >> m;
        vector<vector<double>> A(n,vector<double>(m));
        for(int i=0;i<n;i++){
                for(int j=0;j<m;j++){
                        cin >> A[i][j];
                }
        }
        vector<vector<double>> b(n,vector<double>(1));
        for(int i=0;i<n;i++){
                cin >> b[i][0];
        }
        for(int i=0;i<n;i++){
                A[i].pb(b[i][0]);
        }
        vector<double> x;
        if(gauss(A,x)==1){
                for(int j=0;j<x.size();j++){
                        cout << x[j] << '\n';
                }
        }
}
```

Gaussian Elimination XOR

```cpp
int N,T,K;
int msb(ull a){
    int i;
    for(i=0;a;i++)
        a>>=1;
    return i;
}
int main(){
    cin >> T;
    while(T--){
        cin >> N >> K;
        vull A(N),S,B[70];
        for(auto &a:A){
            cin >> a;
            B[msb(a)].pb(a);
        }
        FORI(i,64,1){
            if(B[i].size()!=0){
                FOR(j,1,B[i].size())
                    B[msb((B[i][0]^B[i][j]))].pb((B[i][0]^B[i][j]));
                S.pb(B[i][0]);
            }
        }
        ull saida=K;
        FOR(i,1,1<<S.size()){
            ull aux=K;
            REP(j,S.size())
                if(i&(1LL<<j)) aux^=S[j];
            saida=max(saida,aux);
        }
        cout << saida << '\n';
    }
}
```

Mo Algorithm

```cpp
//Questao do CF - Powerful array - 86/D
#define int long long
int N,Q,arr[200002],block,sum,OC[1000002],out[200002];
struct _Query{
        int i,L,R;//i -> indice da query, L -> left, R -> Right
        /*o indece serve para retornar para a ordem original dos pedidos*/
};
typedef struct _Query Query;
void add(int p){//Funcao pra add na resposta do range (varia)
        sum+=(p*((OC[p]<<1)+1));
        OC[p]++;
}
void rem(int p){//Funcao pra remover na resposta do range (varia)
        OC[p]--;
        sum-=(p*((OC[p]<<1)+1));
}
bool cmp(Query a,Query b){//sqrt Decomposition
```

```
        if(a.L/block!=b.L/block)
                return a.L < b.L;
        if((a.L/block)&1)
                return a.R<b.R;
        return a.R>b.R;
}
void MoAlgorithm(Query A[],int Q){
        int left=0,right=-1;
        for(int i=0;i<Q;i++){
                Query &q=A[i];
                while(left<q.L) rem(arr[left++]);
                while(left>q.L) add(arr[--left]);
                while(right<q.R) add(arr[++right]);
                while(right>q.R) rem(arr[right--]);
                out[q.i]=sum;//Criar o vetor out[] evita um Q*logQ
        }
}
main(){
        scanf("%lld %lld",&N,&Q);
        for(int i=0;i<N;i++)
                scanf("%lld",&arr[i]);
        block=(int)sqrt(N);
        Query A[Q];
        for(int i=0;i<Q;i++){
                scanf("%lld %lld",&A[i].L,&A[i].R);
                A[i].L--,A[i].R--;
                A[i].i=i;
        }
        sort(A,A+Q,cmp);//ordena em blocos de sqrt(N)
        MoAlgorithm(A,Q);
        for(int i=0;i<Q;i++)
                printf("%lld\n",out[i]);
}
```

SQRT Decomposition

```
int F[100002][320],A[100002],block;//[valMax][sqrt(N)]
void update(int id,int W){
        F[A[id]][id/block]--;
        F[W][id/block]++;
        A[id]=W;
}
int query(int x,int y,int W){
        int x1=x/block,y1=y/block,out=0;
        for(int i=x1+1;i<y1;i++)
                out+=F[W][i];
        if(x1==y1){
                for(int i=x;i<=y;i++)
                        if(A[i]==W)
                                out++;
                return out;
        }
        for(int i=x;(i/block)==x1;i++)
                if(A[i]==W)
                        out++;
        for(int i=y;(i/block)==y1;i--)
                if(A[i]==W)
                        out++;
        return out;
}
void build(){
        for(int i=1;i<=N;i++)
                F[A[i]][i/block]++;
}
```

Bits

```
int couting_bits(int N){
        int i;
        for(i=0;N;i++)
                N&=(N-1);
        return i;
}
__builtin_popcount(int N);
__builtin_popcountll(ll N);
//Portas Logicas
and 1 & 1 = 1, 0 & X = 0
or 1 | X = 1
xor 1 ^ 0 = 1, X ^ X = 0
Conjunto |=(1<<i);//inserir elemento
& intersecao de dois conjuntos
| uniao de dois conjuntos
```

STL

```
//vector<T>
vector<int> A,B;
A.push_back(x);          A.begin();
A.end();                 A.assign(N,val);
A.size();                A.pop_back();
A.erase(A.begin()+i);    A.clear();
A.front();               A.back();
//pair<T,T>
pair<int,int> a;
a.first;                 a.second;
a=make_pair(val1,val2);  a={val1,val2};
//queue<T>
queue<int> q;
q.empty();               q.size();
q.front();               q.back();
q.push(a);               q.pop();
//priority_queue<T>
priority_queue<T,vector<T>,greater<T> >
priority_queue<int> pq;
pq.empty();              pq.size();
pq.top();                pq.push(a);
pq.pop();
//stack<T>
stack<int> pilha;
pilha.empty();           pilha.size();
pilha.top();             pilha.push(val);
pilha.pop();
//set<T>
set<int> S;
S.begin();               S.end();
set<T>::iterator it=S.begin();
S.empty();               S.size();
S.insert(val);           S.clear();
```

```
S.erase(val);//ponteiro ou valor
if(S.find(val)!=S.end())
set<int>::iterator it=S.lower_bound(a);//retorna o ponteiro do elemento
set<int>::iterator it=S.upper_bound(b);//retorna o ponteiro do prox elemento
S.erase(S.lower_bound(a),S.upper_bound(b)); // remove[a,b]
//map<T,T>
map<string,int> Map;
Map.begin();                Map.end();
Map.empty();                Map.size();
Map.insert({"str",x});
pair< map<string,int>::iterator,bool> r;
r=Map.insert({"str",x});
if(r.second) x++;
Map.erase("str");//key or iterator or range(it,Map.end())
//deque<T>
deque<int> dq;
dq.size();                  dq.empty();
dq.front();                 dq.back();
dq.assign(N,val);           dq.push_back(val);
dq.push_front(val);         dq.pop_back();
dq.pop_front();             dq.clear();
//algorithm
lower_bound(v.begin(), v.end(), val);
upper_bound(v.begin(), v.end(), val);
sort(v.begin(), v.end(), cmp);
stable_sort(v.begin(), v.end(),cmp);
partial_sort(v.begin(),v.begin()+x,v.end(),cmp);
merge(first.begin(),first.end(),second.begin(),second.end(),v.begin());
next_permutation(v.begin(),v.end());
```