UNIVERSITY OF HELSINKI

Contents

1 src/string/suffixarray.cpp

```cpp
// TCR
// Suffix array in O(n log^2 n)
// ~300ms runtime for 10^5 character string, ~2000ms for 5*10^5
// You can change vector<int> s to string s
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;

vector<int> suffixArray(vector<int> s) {
        int n=s.size();
        vector<int> k(n);
        for (int i=0;i<n;i++) {
                k[i]=s[i];
        }
        vector<pair<pair<int, int>, int> > v(n);
        for (int t=1;t<=n;t*=2) {
                for (int i=0;i<n;i++) {
                        int u=-1;
                        if (i+t<n) u=k[i+t];
                        v[i]={{k[i], u}, i};
                }
                sort(v.begin(), v.end());
                int c=0;
                for (int i=0;i<n;i++) {
                        if (i>0&&v[i-1].F!=v[i].F) c++;
                        k[v[i].S]=c;
                }
                if (c==n-1) break;
        }
        vector<int> sa(n);
        for (int i=0;i<n;i++) sa[k[i]]=i;
        return sa;
}
```

2 src/string/lcparray.cpp

```cpp
// TCR
```

```cpp
// Constructs LCP array from suffix array in O(n) time
// You can change vector<int> s to string s
#include <bits/stdc++.h>
using namespace std;

vector<int> lcpArray(vector<int> s, vector<int> sa) {
        int n=s.size();
        int k=0;
        vector<int> ra(n), lcp(n);
        for (int i=0;i<n;i++) ra[sa[i]]=i;
        for (int i=0;i<n;i++) {
                if (k) k--;
                if (ra[i]==n-1) {
                        k=0;
                        continue;
                }
                int j=sa[ra[i]+1];
                while (k<n&&s[(i+k)%n]==s[(j+k)%n]) k++;
                lcp[ra[i]]=k;
                if (ra[(sa[ra[i]]+1)%n]>ra[(sa[ra[j]]+1)%n]) k=0;
        }
        return lcp;
}
```

3   src/string/aho-corasick.cpp

```cpp
// TCR
// Aho-Corasick algorithm
// Building of automaton is O(L) where L is total length of dictionary
// Matching is O(n + number of matches), O(n sqrt(L)) in the worst case
// Add dictionary using addString and then use pushLinks
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;

struct AhoCorasick {
        vector<map<char, int> > g;
        vector<int> link;
        vector<int> tlink;
        vector<int> te;

        // Use 1-indexing in id
        void addString(const string& s, int id) {
                int tn=0;
                for (int i=0;i<(int)s.size();i++) {
                        if (g[tn][s[i]]==0) {
                                g[tn][s[i]]=g.size();
                                g.push_back(map<char, int>());
                                link.push_back(0);
                                tlink.push_back(0);
                                te.push_back(0);
                        }
                        tn=g[tn][s[i]];
                }
                te[tn]=id;
        }

        void pushLinks() {
                queue<int> bfs;
                bfs.push(0);
                while (!bfs.empty()) {
                        int x=bfs.front();
                        bfs.pop();
                        for (auto nx:g[x]) {
                                int l=link[x];
                                while (l!=-1&&g[l].count(nx.F)==0) l=link[l];
                                if (l!=-1) link[nx.S]=g[l][nx.F];
                                bfs.push(nx.S);
                                if (te[link[nx.S]]) {
                                        tlink[nx.S]=link[nx.S];
                                }
                                else{
                                        tlink[nx.S]=tlink[link[nx.S]];
                                }
                        }
                }
        }
};

// Returns matches {id, endpos}
vector<pair<int, int> > match(const string& s) {
        int tn=0;
        vector<pair<int, int> > re;
        for (int i=0;i<(int)s.size();i++) {
                while (tn!=-1&&g[tn].count(s[i])==0) tn=link[tn];
                if (tn==-1) tn=0;
                tn=g[tn][s[i]];
                int f=tlink[tn];
                if (te[tn]) re.push_back({te[tn], i});
                while (f) {
                        re.push_back({te[f], i});
                        f=tlink[f];
                }
```

```
                }
            }
            return re;
        }

        AhoCorasick() {
            g.push_back(map<char, int>());
            link.push_back(-1);
            tlink.push_back(0);
            te.push_back(0);
        }
};
```

4   src/string/z.cpp

```
// TCR
// Computes the Z array in linear time
// z[i] is the length of the longest common prefix of substring
// starting at i and the string
// You can use string s instead of vector<int> s
// z[0]=0 by definition
#include <bits/stdc++.h>
using namespace std;

vector<int> zAlgo(vector<int> s) {
        int n=s.size();
        vector<int> z(n);
        int l=0;
        int r=0;
        for (int i=1;i<n;i++) {
                z[i]=max(0, min(z[i-l], r-i));
                while (i+z[i]<n&&s[z[i]]==s[i+z[i]]) z[i]++;
                if (i+z[i]>r) {
                        l=i;
                        r=i+z[i];
                }
        }
        return z;
}
```

5   src/string/suffixautomaton.cpp

```
// TCR
// Online suffix automaton construction algorithm
// Time complexity of adding one character is amortized O(1)
#include <bits/stdc++.h>
```

```
using namespace std;

struct SuffixAutomaton {
        vector<map<char, int> > g;
        vector<int> link;
        vector<int> len;
        int last;
        void addC(char c) {
                int p=last;
                int t=link.size();
                link.push_back(0);
                len.push_back(len[last]+1);
                g.push_back(map<char, int>());
                while (p!=-1&&g[p].count(c)==0) {
                        g[p][c]=t;
                        p=link[p];
                }
                if (p!=-1) {
                        int q=g[p][c];
                        if (len[p]+1==len[q]) {
                                link[t]=q;
                        }
                        else {
                                int qq=link.size();
                                link.push_back(link[q]);
                                len.push_back(len[p]+1);
                                g.push_back(g[q]);
                                while (p!=-1&&g[p][c]==q) {
                                        g[p][c]=qq;
                                        p=link[p];
                                }
                                link[q]=qq;
                                link[t]=qq;
                        }
                }
                last=t;
        }
        SuffixAutomaton() : SuffixAutomaton("") {}
        SuffixAutomaton(string s) {
                last=0;
                g.push_back(map<char, int>());
                link.push_back(-1);
                len.push_back(0);
                for (int i=0;i<(int)s.size();i++) {
                        addC(s[i]);
                }
        }
```

```
            }
        vector<int> terminals() {
                vector<int> t;
                int p=last;
                while (p>0) {
                        t.push_back(p);
                        p=link[p];
                }
                return t;
        }
};
```

6   src/geometry/convexhull.cpp

```
// TCR
// Computes the convex hull of given set of points in O(n log n)
// Uses Andrew's algorithm
// The points on the edges of the hull are not listed
// Change > to >= in ccw function to list the points on the edges
// Returns points in counterclockwise order
#include <bits/stdc++.h>
#define X real()
#define Y imag()
using namespace std;
typedef long double ld;
typedef long long ll;
// Coordinate type
typedef ll CT;
typedef complex<CT> co;

bool ccw(co a, co b, co c) {
        return ((c-a)*conj(b-a)).Y>0;
}

vector<co> convexHull(vector<co> ps) {
        auto cmp = [](co a, co b) {
                if (a.X==b.X) return a.Y<b.Y;
                else return a.X<b.X;
        };
        sort(ps.begin(), ps.end(), cmp);
        ps.erase(unique(ps.begin(), ps.end()), ps.end());

        int n=ps.size();
        if (n<=2) return ps;

        vector<co> hull;
```

```
            hull.push_back(ps[0]);
        for (int d=0;d<2;d++) {
                if (d) reverse(ps.begin(), ps.end());
                int s=hull.size();
                for (int i=1;i<n;i++) {
                        while ((int)hull.size()>s&&!ccw(hull[hull.size()-2],
hull.back(), ps[i])) {
                                hull.pop_back();
                        }
                        hull.push_back(ps[i]);
                }
        }
        hull.pop_back();
        return hull;
}
```

7   src/geometry/anglesort.cpp

```
// TCR
// Comparasion function for sorting points around origin
// Points are sorted in clockwise order
/*
122
143
443*/
#include <bits/stdc++.h>
#define X real()
#define Y imag()
#define F first
#define S second
using namespace std;
typedef long double ld;
typedef long long ll;
// Coordinate type
typedef ld CT;
typedef complex<CT> co;

bool ccw(co a, co b, co c) {
        return ((c-a)*conj(b-a)).Y>0;
}

int ar(co x) {
        if (x.Y>=0&&x.X<0) return 1;
        if (x.X>=0&&x.Y>0) return 2;
        if (x.Y<=0&&x.X>0) return 3;
        return 4;
```

```
}

bool cp(co p1, co p2) {
        if (ar(p1)!=ar(p2)) {
                return ar(p1)<ar(p2);
        }
        return ccw({0, 0}, p2, p1)>0;
}
```

8 src/geometry/minkowskisum.cpp

```cpp
// TCR
// Computes the Minkowski sum of 2 convex polygons in O(n+m log n+m)
// Returns convex polygon in counterclockwise order
// The points on the edges of the hull are listed
// The convex hulls must be in counterclockwise order
#include <bits/stdc++.h>
#define X real()
#define Y imag()
#define F first
#define S second
using namespace std;
typedef long double ld;
typedef long long ll;
// Coordinate type
typedef ll CT;
typedef complex<CT> co;

ll ccw(co a, co b, co c) {
        return ((c-a)*conj(b-a)).Y;
}

int ar(co x) {
        if (x.Y>=0&&x.X<0) return 1;
        if (x.X>=0&&x.Y>0) return 2;
        if (x.Y<=0&&x.X>0) return 3;
        return 4;
}

bool cp(pair<co, pair<int, int> > p1, pair<co, pair<int, int> > p2) {
        if (ar(p1.F)!=ar(p2.F)) {
                return ar(p1.F)<ar(p2.F);
        }
        assert((ccw({0, 0}, p1.F, p2.F)==0)==(ccw({0, 0}, p2.F, p1.F)==0));
        if (ccw({0, 0}, p1.F, p2.F)==0){
                return p1.S>p2.S;
```

```cpp
        }
        return ccw({0, 0}, p2.F, p1.F)>0;
}

vector<co> minkowski(vector<co>&a, vector<co>&b) {
        int n=a.size();
        int m=b.size();
        if (n==0) return b;
        if (m==0) return a;
        if (n==1) {
                vector<co> ret(m);
                for (int i=0;i<m;i++) {
                        ret[i]=b[i]+a[0];
                }
                return ret;
        }
        if (m==1) {
                vector<co> ret(n);
                for (int i=0;i<n;i++) {
                        ret[i]=a[i]+b[0];
                }
                return ret;
        }
        vector<pair<co, pair<int, int> > > pp;
        int f1=0;
        int f2=0;
        for (int i=0;i<n;i++) {
                if (ccw(a[(i-1+n)%n], a[i], a[(i+1)%n])!=0) {
                        f1=i;
                        break;
                }
        }
        for (int i=0;i<n;i++) {
                pp.push_back({a[(i+1+f1)%n]-a[(i+f1)%n], {1, i}});
        }
        for (int i=0;i<m;i++) {
                if (ccw(b[(i-1+m)%m], b[i], b[(i+1)%m])!=0) {
                        f2=i;
                        break;
                }
        }
        for (int i=0;i<m;i++) {
                pp.push_back({b[(i+1+f2)%m]-b[(i+f2)%m], {2, i}});
        }
        sort(pp.rbegin(), pp.rend(), cp);
        co s={0, 0};
```

```
        co ad={0, 0};
        for (int i=0;i<(int)pp.size();i++) {
                s+=pp[i].F;
                if (pp[i].S.F!=pp[i+1].S.F) {
                        if (pp[i].S.F==1) ad=a[(pp[i].S.S+1+f1)%n]+b[(pp[i+1].S.S+f2)%m];
                        else ad=b[(pp[i].S.S+1+f2)%m]+a[(pp[i+1].S.S+f1)%n];
                        ad-=s;
                        break;
                }
        }
        s=ad;
        vector<co> ret(pp.size());
        for (int i=0;i<(int)pp.size();i++) {
                ret[i]=s;
                s+=pp[i].F;
        }
        return ret;
}
```

9   src/geometry/basic.cpp

```cpp
// TCR
// Basic geometry functions using complex numbers
// Mostly copied from https://github.com/ttalvitie/libcontest/
/* Useful functions of complex number class
        CT abs(co x): Length
        CT norm(co x): Square of length
        CT arg(co x): Angle
        co polar(CT length, CT angle): Complex from polar components
*/
#include <bits/stdc++.h>
#define X real()
#define Y imag()
using namespace std;
typedef long double ld;
typedef long long ll;
// Coordinate type
typedef ld CT;
typedef complex<CT> co;

// Return true iff points a, b, c are CCW oriented.
bool ccw(co a, co b, co c) {
        return ((c-a)*conj(b-a)).Y>0;
}

// Return true iff points a, b, c are collinear.
```

```cpp
// Note: doesn't make much sense with non-integer CT.
bool collinear(co a, co b, co c) {
        return ((c-a)*conj(b-a)).Y==0;
}

// Rotate x with agle ang
co rotate(co x, CT ang) {
        return x*polar((CT)1, ang);
}

// Check whether segments [a, b] and [c, d] intersect.
// The segments must not be collinear. Doesn't handle edge cases (endpoint of
// a segment on the other segment) consistently.
bool intersects(co a, co b, co c, co d) {
        return ccw(a, d, b)!=ccw(a, c, b)&&ccw(c, a, d)!=ccw(c, b, d);
}

// Interpolate between points a and b with parameter t.
co interpolate(CT t, co a, co b) {
        return a+t*(b-a);
}

// Return interpolation parameter between a and b of projection of v to the
// line defined by a and b.
// Note: no rounding behavior specified for integers.
CT projectionParam(co v, co a, co b) {
        return ((v-a)/(b-a)).X;
}

// Compute the distance of point v from line a..b.
// Note: Only for non-integers!
CT pointLineDistance(co p, co a, co b) {
        return abs(((p-a)/(b-a)).Y)*abs(b-a);
}

// Compute the distance of point v from segment a..b.
// Note: Only for non-integers!
CT pointSegmentDistance(co p, co a, co b) {
        co z=(p-a)/(b-a);
        if(z.X<0) return abs(p-a);
        if(z.X>1) return abs(p-b);
        return abs(z.Y)*abs(b-a);
}

// Return interpolation parameter between a and b of the point that is also
// on line c..d.
```

```cpp
// Note: Only for non-integers!
CT intersectionParam(co a, co b, co c, co d) {
        co u=(c-a)/(b-a);
        co v=(d-a)/(b-a);
        return (u.X*v.Y-u.Y*v.X)/(v.Y-u.Y);
}
```

10 src/geometry/closestpoints.cpp

```cpp
// TCR
// Returns square of distance between closest 2 points
// O(n log n)
#include <bits/stdc++.h>
#define X real()
#define Y imag()
#define F first
#define S second
using namespace std;
typedef long long ll;
typedef complex<ll> co;

const ll inf=2e18;

ll csqrt(ll x) {
        ll r=sqrt(x);
        while (r*r<x) r++;
        while (r*r>x) r--;
        return r;
}

ll sq(ll x) {
        return x*x;
}

ll closestPoints(vector<co> points) {
        int n=points.size();
        vector<pair<ll, ll> > ps(n);
        for (int i=0;i<n;i++) {
                ps[i]={points[i].X, points[i].Y};
        }
        sort(ps.begin(), ps.end());
        int i2=0;
        ll d=inf;
        set<pair<ll, ll> > pss;
        for (int i=0;i<n;i++) {
                while (i2<i&&sq(ps[i].F-ps[i2].F)>d) {
```

```cpp
                        pss.erase({ps[i2].S, ps[i2].F});
                        i2++;
                }
                auto it=pss.lower_bound({ps[i].S-csqrt(d), -inf});
                for (;it!=pss.end();it++) {
                        if (sq(it->F-ps[i].S)>d) break;
                        d=min(d, sq(it->F-ps[i].S)+sq(it->S-ps[i].F));
                }
                pss.insert({ps[i].S, ps[i].F});
        }
        return d;
}
```

11 src/datastructure/orderedset.cpp

```cpp
// TCR
// Sample code on how to use g++ ordered set
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace std;
using namespace __gnu_pbds;
//using namespace pb_ds;

typedef tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>
ordered_set;

int main() {
        ordered_set X;
        X.insert(1);
        X.insert(4);
        cout<<*X.find_by_order(1)<<endl; // 4
        cout<<X.order_of_key(3)<<endl;   // 1
}
```

12 src/datastructure/treap.cpp

```cpp
// TCR
// Treap implementation with pointers
// Expected running time of split and merge is O(log n)
#include <bits/stdc++.h>
using namespace std;

typedef struct node* pnode;
struct node {
```

```cpp
        pnode l,r;
        int pr,c;
        node() {
                l=0;
                r=0;
                c=1;
                pr=rand();
        }
};

// Returns the size of the subtree t
int cnt(pnode t) {
        if (t) return t->c;
        return 0;
}

// Updates the size of the subtree t
void upd(pnode t) {
        if (t) t->c=cnt(t->l)+cnt(t->r)+1;
}

// Put lazy updates here
void push(pnode t) {
        if (t) {
                // Do lazy update
        }
}

// Merges trees l and r into tree t
void merg(pnode& t, pnode l, pnode r) {
        push(l);
        push(r);
        if (!l) t=r;
        else if(!r) t=l;
        else {
                if (l->pr>r->pr) {
                        merg(l->r, l->r, r);
                        t=l;
                }
                else {
                        merg(r->l, l, r->l);
                        t=r;
                }
        }
        upd(t);
}
```

```cpp
// Splits tree t into trees l and r
// Size of tree l will be k
void split(pnode t, pnode& l, pnode& r, int k) {
        if (!t) {
                l=0;
                r=0;
                return;
        }
        else {
                push(t);
                if (cnt(t->l)>=k) {
                        split(t->l, l, t->l, k);
                        r=t;
                }
                else {
                        split(t->r, t->r, r, k-cnt(t->l)-1);
                        l=t;
                }
        }
        upd(t);
}
```

13   src/math/crt.cpp

```cpp
// TCR
// (Generalised) Chinese remainder theorem (for arbitrary moduli):
// Solves x from system of equations x == a_i (mod m_i), giving answer modulo m =
lcm(m_1,...,m_n)
// Runs in O(log(m)+n) time
// Overflows only if m overflows
// Returns {1, {x, m}} if solution exists, and {-1, {0,0}} otherwise
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef __int128 lll;

ll ee(ll ca, ll cb, ll xa, ll xb, ll&x) {
        if (cb) return ee(cb, ca%cb, xb, xa-(ca/cb)*xb, x);
        x = xa;
        return ca;
}

pair<int, pair<ll, ll>> crt(vector<ll> as, vector<ll> ms) {
        ll aa = 0, mm = 1, d, a, x;
        for (int i = 0; i < (int) as.size(); i++) {
```

```
                d = ee(ms[i], mm, 1, 0, x);
                if ((aa-as[i])%d) return {-1,{0,0}};
                a = ms[i]/d;
                mm *= a;
                aa = (as[i] + (aa-as[i])*(((lll)a*x)%mm))%mm;
        }
        if (aa < 0) aa += mm;
        return {1, {aa, mm}};
}
```

14 src/math/fftmod.cpp

```cpp
// TCR
// Precise FFT modulo mod
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
// Number of form (2^20)*k+1
const ll mod=1045430273;
// Number whose order mod mod is 2^20
const ll root=363;
const ll root_pw=1<<20;

// 128 bit
//const lll mod=2097152000007340033;
//const lll root=2014907510281342032;
//const lll root_pw=1<<20;

ll pot(ll x, ll p) {
        if (p==0) return 1;
        if (p%2==0) {
                x=pot(x, p/2);
                return (x*x)%mod;
        }
        return (x*pot(x, p-1))%mod;
}

ll inv(ll x) {
        return pot(x, mod-2);
}

vector<ll> fft (vector<ll> a, int d) {
        ll root_1=inv(root);
        int n=(int)a.size();
        for (int i=1,j=0;i<n;i++) {
                int bit=n>>1;
                for (;j>=bit;bit>>=1) {
                        j-=bit;
                }
                j+=bit;
                if (i<j) swap (a[i], a[j]);
        }
        for (int len=2;len<=n;len<<=1) {
                ll wlen=root;
                if (d==-1) {
                        wlen=root_1;
                }
                for (int i=len;i<root_pw;i<<=1) wlen=(wlen*wlen)%mod;
                for (int i=0;i<n;i+=len) {
                        ll w = 1;
                        for (int j=0;j<len/2;j++) {
                                ll u = a[i+j];
                                ll v = (a[i+j+len/2]*w)%mod;
                                if (u+v<mod) {
                                        a[i+j]=u+v;
                                }
                                else {
                                        a[i+j]=u+v-mod;
                                }
                                if (u-v>=0) {
                                        a[i+j+len/2]=u-v;
                                }
                                else {
                                        a[i+j+len/2]=u-v+mod;
                                }
                                w=(w*wlen)%mod;
                        }
                }
        }
        if (d==-1) {
                ll nrev=inv(n);
                for (int i=0;i<n;i++) a[i]=(a[i]*nrev)%mod;
        }
        return a;
}

vector<ll> conv(vector<ll> a, vector<ll> b) {
        int as=a.size();
        int bs=b.size();
        vector<ll> aa(as);
        vector<ll> bb(bs);
        for (int i=0;i<as;i++) {
```

```cpp
                aa[i]=a[i];
        }
        for (int i=0;i<bs;i++) {
                bb[i]=b[i];
        }
        int n=1;
        while (n<as+bs-1) n*=2;
        aa.resize(n*2);
        bb.resize(n*2);
        aa=fft(aa, 1);
        bb=fft(bb, 1);
        vector<ll> c(2*n);
        for (int i=0;i<2*n;i++) {
                c[i]=(aa[i]*bb[i])%mod;
        }
        c=fft(c, -1);
        c.resize(as+bs-1);
        return c;
}

int main() {
        // Shoud print 12 11 30 7
        vector<ll> a={3, 2, 7};
        vector<ll> b={4, 1};
        vector<ll> c=conv(a, b);
        for (ll t:c) {
                cout<<t<<endl;
        }
}
```

15   src/math/gaussjordan.cpp

```cpp
// TCR
// Solves system of linear equations in O(n m^2)
// Using doubles or mod 2
// Using doubles might have large precision errors or overflow
// Returns 0 if no solution exists, 1 if there is one solution
// or 2 if infinite number of solutions exists
// If at least one solution exists, it is returned in ans
// You can modify the general algorithm to work mod p by using modular inverse
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;
const ld eps=1e-12;
```

```cpp
// Using doubles
int gaussD (vector<vector<ld> > a, vector<ld>& 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) {
                                ld 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++) {
                ld 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 2;
        }
        return 1;
}
```

```cpp
// mod 2
// n is number of rows m is number of variables
const int M=4;
int gaussM(vector<bitset<M> > a, int n, int m, bitset<M-1>& ans) {
        vector<int> where (m, -1);
        for (int col=0,row=0;col<m&&row<n;col++) {
                for (int i=row;i<n;i++) {
                        if (a[i][col]) {
                                swap (a[i], a[row]);
                                break;
                        }
                }
                if (!a[row][col]) continue;
                where[col]=row;

                for (int i=0;i<n;i++) {
                        if (i!=row&&a[i][col]) {
                                a[i]^=a[row];
                        }
                }
                row++;
        }
        ans=0;
        for (int i=0;i<m;i++) {
                if (where[i]!=-1) ans[i]=a[where[i]][m];
        }
        for (int i=0;i<n;i++) {
                int sum=0;
                for (int j=0;j<m;j++) {
                        sum^=ans[j]*a[i][j];
                }
                if (sum!=a[i][m]){
                        return 0;
                }
        }
        for (int i=0;i<m;i++){
                if (where[i]==-1) return 2;
        }
        return 1;
}

int main() {
        // Should output 2, 1 2 0
        vector<vector<ld> > d(3);
        d[0]={3, 3, -15, 9};
        d[1]={1, 0, -2, 1};
```

```cpp
        d[2]={2, -1, -1, 0};
        vector<ld> da;
        cout<<gaussD(d, da)<<endl;
        cout<<da[0]<<" "<<da[1]<<" "<<da[2]<<endl;

        // Should output 1, 110
        // Note that bitsets are printed in reverse order
        bitset<M> r1("0110");
        bitset<M> r2("1101");
        bitset<M> r3("0111");
        vector<bitset<M> > m={r1, r2, r3};
        bitset<M-1> ma;
        cout<<gaussM(m, 3, 3, ma)<<endl;
        cout<<ma<<endl;
}
```

16   src/math/fft.cpp

```cpp
// TCR
// Fast Fourier transform and convolution using it
// O(n log n)
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
typedef long long ll;
typedef complex<ld> co;
const ld PI=atan2(0, -1);

vector<co> fft(vector<co> x, int d) {
        int n=x.size();
        for (int i=0;i<n;i++) {
                int u=0;
                for (int j=1;j<n;j*=2) {
                        u*=2;
                        if (i&j) u++;
                }
                if (i<u) {
                        swap(x[i], x[u]);
                }
        }
        for (int m=2;m<=n;m*=2) {
                co wm=exp(co{0, d*2*PI/m});
                for (int k=0;k<n;k+=m) {
                        co w=1;
                        for (int j=0;j<m/2;j++) {
                                co t=w*x[k+j+m/2];
```

```
                        co u=x[k+j];
                        x[k+j]=u+t;
                        x[k+j+m/2]=u-t;
                        w*=wm;
                    }
                }
            }
            if (d==-1) {
                    for (int i=0;i<n;i++) {
                            x[i]/=n;
                    }
            }
            return x;
}

vector<ll> conv(vector<ll> a, vector<ll> b) {
        int as=a.size();
        int bs=b.size();
        vector<co> aa(as);
        vector<co> bb(bs);
        for (int i=0;i<as;i++) {
                aa[i]=a[i];
        }
        for (int i=0;i<bs;i++) {
                bb[i]=b[i];
        }
        int n=1;
        while (n<as+bs-1) n*=2;
        aa.resize(n*2);
        bb.resize(n*2);
        aa=fft(aa, 1);
        bb=fft(bb, 1);
        vector<co> c(2*n);
        for (int i=0;i<2*n;i++) {
                c[i]=aa[i]*bb[i];
        }
        c=fft(c, -1);
        c.resize(as+bs-1);
        vector<ll> r(as+bs-1);
        for (int i=0;i<as+bs-1;i++) {
                r[i]=(ll)round(c[i].real());
        }
        return r;
}

int main() {
```

```
        // Shoud print 12 11 30 7
        vector<ll> a={3, 2, 7};
        vector<ll> b={4, 1};
        vector<ll> c=conv(a, b);
        for (ll t:c) {
                cout<<t<<endl;
        }
}
```

17 src/math/miller-rabin.cpp

```
// TCR
// Deterministic Miller-Rabin primality test
// Works for all 64 bit integers
// Support of 128 bit integers is required to test over 32 bit integers
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef __int128 lll;

lll powmod(lll a, lll p, lll mod) {
        if (p==0) return 1;
        if (p%2==0) {
                a=powmod(a, p/2, mod);
                return (a*a)%mod;
        }
        return (a*powmod(a, p-1, mod))%mod;
}

bool is_w(ll a, ll even, ll odd, ll p) {
        lll u = powmod(a, odd, p);
        if (u==1) return 0;
        for (ll j=1;j<even;j*=2) {
                if (u==p-1) return 0;
                u*=u;
                u%=p;
        }
        return 1;
}

bool isPrime(ll p) {
        if (p==2) return 1;
        if (p<=1||p%2==0) return 0;
        ll odd=p-1;
        ll even=1;
        while (odd%2==0) {
```

```
                even*=2;
                odd/=2;
        }
        ll b[7]={2, 325, 9375, 28178, 450775, 9780504, 1795265022};
        for (ll i=0;i<7;i++) {
                ll a=b[i]%p;
                if (a==0) return 1;
                if (is_w(a, even, odd, p)) return 0;
        }
        return 1;
}
```

18   src/math/primitiveroot.cpp

```
// TCR
// Computes primitive root
// O(sqrt(n))
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

ll pot(ll x, ll p, ll mod) {
        if (p==0) return 1;
        if (p%2==0) {
                x=pot(x, p/2, mod);
                return (x*x)%mod;
        }
        return (x*pot(x, p-1, mod))%mod;
}

ll primitiveRoot(ll p) {
        vector<ll> fact;
        ll phi=p-1;
        ll n=phi;
        for (ll i=2;i*i<=n;i++) {
                if (n%i==0) {
                        fact.push_back(i);
                        while (n%i==0) n/=i;
                }
        }
        if (n>1) fact.push_back (n);
        for (ll res=2;res<=p;res++) {
                bool ok = true;
                for (int i=0;i<(int)fact.size()&&ok;i++) ok&=pot(res, phi/fact[i],
p)!=1;
                if (ok)  return res;
```

```
        }
        return -1;
}

int main() {
        cout<<primitiveRoot(1000000007)<<endl;// should print 5
}
```

19   src/math/diophantine.cpp

```
// TCR
// Solves ax+by=c in O(log a+b) time
// Returns {is, {x, y}}, is=0 if there is no solution
// Use __int128 for 64 bit numbers
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;
typedef long long ll;

ll ee(ll a, ll b, ll ca, ll cb, ll xa, ll xb, ll&x, ll&y) {
        if (cb==0) {
                x=xa;
                if (b==0) y=0;
                else y=(ca-a*xa)/b;
                return ca;
        }
        else return ee(a, b, cb, ca%cb, xb, xa-(ca/cb)*xb, x, y);
}

pair<int, pair<ll, ll> > solve(ll a, ll b, ll c) {
        if (c==0) return {1, {0, 0}};
        if (a==0&&b==0) return {0, {0, 0}};
        ll x,y;
        ll g=ee(a, b, a, b, 1, 0, x, y);
        if (abs(c)%g>0) return {0, {0, 0}};
        return {1, {x*(c/g), y*(c/g)}};
}
```

20   src/graph/stronglyconnected.cpp

```
// TCR
// Kosaraju's algorithm for strongly connected components O(V+E)
// Components will be returned in topological order
// Uses 1-indexing
#include <bits/stdc++.h>
```

```cpp
using namespace std;

struct SCC {
        vector<int> used;
        vector<vector<int> > g2;

        void dfs1(vector<int>* g, int x, vector<int>& ns) {
                if (used[x]==1) return;
                used[x]=1;
                for (int nx:g[x]) {
                        g2[nx].push_back(x);
                        dfs1(g, nx, ns);
                }
                ns.push_back(x);
        }

        void dfs2(int x, vector<int>& co) {
                if (used[x]==2) return;
                used[x]=2;
                co.push_back(x);
                for (int nx:g2[x]) {
                        dfs2(nx, co);
                }
        }

        // Returns strongly connected components of the graph in vector ret
        // n is the size of the graph, g is the adjacency list
        SCC(vector<int>* g, int n, vector<vector<int> >& ret) : used(n+1),
g2(n+1) {
                vector<int> ns;
                for (int i=1;i<=n;i++) {
                        dfs1(g, i, ns);
                }
                for (int i=n-1;i>=0;i--) {
                        if (used[ns[i]]!=2) {
                                ret.push_back(vector<int>());
                                dfs2(ns[i], ret.back());
                        }
                }
        }
};
```

21   src/graph/eulertour.cpp

// TCR

```cpp
// NOT TESTED PROPERLY
// Finds Euler tour of graph in O(E) time

// Parameters are the adjacency list, number of nodes,
// return value vector, and d=1 if the graph is directed
// Return array contains E+1 elements, the first and last
// elements are same

// Undefined behavior if Euler tour doesn't exist

// Note that Eulerian path can be reduced to Euler tour
// by adding an edge from the last vertex to the first

// In bidirectional graph edges must be in both direction
// Be careful to not add loops twice in case of bidirectional graph
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;

struct EulerTour {
        int dir;
        vector<vector<pair<int, int> > > g;
        vector<int> used;

        void dfs(int x, vector<int>& ret) {
                int t=x;
                vector<int> c;
                while (1) {
                        while (used[g[t].back().S]) g[t].pop_back();
                        auto nx=g[t].back();
                        g[t].pop_back();
                        used[nx.S]=1;
                        t=nx.F;
                        c.push_back(t);
                        if (t==x) break;
                }
                for (int a:c) {
                        ret.push_back(a);
                        while (g[a].size()>0&&used[g[a].back().S]) g[a].pop_back();
                        if (g[a].size()>0) dfs(a, ret);
                }
        }

        EulerTour(vector<int>* og, int n, vector<int>& ret, int d=0) : dir(d),
g(n+1) {
```

```
            int i2=0;
            for (int i=1;i<=n;i++) {
                    for (int nx:og[i]) {
                            if (d==1||nx<=i) {
                                    if (d==0&&nx<i) g[nx].push_back({i, i2});
                                    g[i].push_back({nx, i2++});
                            }
                    }
            }
            used.resize(i2);
            for (int i=1;i<=n;i++) {
                    if (g[i].size()>0) {
                            ret.push_back(i);
                            dfs(i, ret);
                            break;
                    }
            }
    }
};
```

22   src/graph/cutvertices.cpp

```cpp
// TCR
// Finds cutvertices and 2-vertex-connected components of graph
// 2-vertex-connected components are stored in bg
// Uses 1-indexing
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;

struct Biconnected {
        vector<int> cut, h, d, used;
        vector<map<int, vector<int> > > bg;
        vector<pair<int, int> > es;
        int cc;
        void dfs(vector<int>* g, int x, int p) {
                h[x]=d[x];
                int f=0;
                for (int nx:g[x]) {
                        if (nx!=p) {
                                if (!used[nx]) es.push_back({x, nx});
                                if (d[nx]==0) {
                                        f++;
                                        d[nx]=d[x]+1;
                                        int ts=es.size();
```

```cpp
                                        dfs(g, nx, x);
                                        h[x]=min(h[x], h[nx]);
                                        if (h[nx]>=d[x]) {
                                                cut[x]=1;
                                                while ((int)es.size()>=ts) {
                                                        auto e=es.back();
                                                        bg[e.F][cc].push_back(e.S);
                                                        bg[e.S][cc].push_back(e.F);
                                                        used[e.S]=1;
                                                        used[e.F]=1;
                                                        es.pop_back();
                                                }
                                                used[x]=0;
                                                cc++;
                                        }
                                }
                                h[x]=min(h[x], d[nx]);
                        }
                }
                if (p==0) {
                        if (f>1) cut[x]=1;
                        else cut[x]=0;
                }
        }

        Biconnected(vector<int>* g, int n) : cut(n+1), h(n+1), d(n+1), used(n+1),
bg(n+1) {
                cc=1;
                for (int i=1;i<=n;i++) {
                        if (d[i]==0) {
                                d[i]=1;
                                dfs(g, i, 0);
                        }
                }
        }
};
```

23   src/graph/linkcut.cpp

```cpp
// TCR
// Link/cut tree. All operations are amortized O(log n) time
#include <bits/stdc++.h>
using namespace std;
struct Node {
        Node* c[2], *p;
        int id, rev;
```

```cpp
    int isr() {
            return !p||(p->c[0]!=this&&p->c[1]!=this);
    }
    int dir() {
            return p->c[1]==this;
    }
    void setc(Node* s, int d) {
            c[d]=s;
            if (s) s->p=this;
    }
    void push() {
            if (rev) {
                    swap(c[0], c[1]);
                    if (c[0]) c[0]->rev^=1;
                    if (c[1]) c[1]->rev^=1;
                    rev=0;
            }
    }
    Node(int i) : id(i) {
            c[0]=0;
            c[1]=0;
            p=0;
            rev=0;
    }
};
struct LinkCut {
    void rot(Node* x) {
            Node* p=x->p;
            int d=x->dir();
            if (!p->isr()) {
                    p->p->setc(x, p->dir());
            }
            else {
                    x->p=p->p;
            }
            p->setc(x->c[!d], d);
            x->setc(p, !d);
    }
    void pp(Node* x) {
            if (!x->isr()) pp(x->p);
            x->push();
    }
    void splay(Node* x) {
            pp(x);
            while (!x->isr()) {
                    if (x->p->isr()) rot(x);
                    else if(x->dir()==x->p->dir()) {
                            rot(x->p);
                            rot(x);
                    }
                    else {
                            rot(x);
                            rot(x);
                    }
            }
    }
    Node* expose(Node* x) {
            Node* q=0;
            for (;x;x=x->p) {
                    splay(x);
                    x->c[1]=q;
                    q=x;
            }
            return q;
    }
    void evert(Node* x) {
            x=expose(x);
            x->rev^=1;
            x->push();
    }
    void link(Node* x, Node* y) {
            evert(x);
            evert(y);
            splay(y);
            x->setc(y, 1);
    }
    void cut(Node* x, Node* y) {
            evert(x);
            expose(y);
            splay(x);
            x->c[1]=0;
            y->p=0;
    }
    int rootid(Node* x) {
            expose(x);
            splay(x);
            while(x->c[0]) {
                    x=x->c[0];
                    x->push();
            }
            splay(x);
            return x->id;
```

24 src/graph/scalingflow.cpp

```cpp
// TCR
// Scaling flow algorithm for maxflow
// O(E^2 log U), where U is maximum possible flow
// In practice O(E^2)
// Uses 1-indexing


#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;
typedef long long ll;

struct MaxFlow {
        // Use vector<map<int, ll> > for sparse graphs
        vector<vector<ll> > f;
        vector<vector<int> > g;
        vector<int> used;
        int cc;

        ll flow(int x, int t, ll fl, ll miv) {
                if (x==t) return fl;
                used[x]=cc;
                for (int nx:g[x]) {
                        if (used[nx]!=cc&&f[x][nx]>=miv) {
                                ll r=flow(nx, t, min(fl, f[x][nx]), miv);
                                if (r>0) {
                                        f[x][nx]-=r;
                                        f[nx][x]+=r;
                                        return r;
                                }
                        }
                }
                return 0;
        }

        // maxv is maximum expected maxflow
        ll getMaxFlow(int source, int sink, ll maxv) {
                cc=1;
                ll r=0;
                ll k=1;
                while (k*2<=maxv) k*=2;
                for (;k>0;k/=2) {
                        while (ll t=flow(source, sink, maxv, k)) {
                                r+=t;
                                cc++;
                        }
                        cc++;
                }
                return r;
        }

        void addEdge(int a, int b, ll c) {
                if (f[a][b]==0&&f[b][a]==0) {
                        g[a].push_back(b);
                        g[b].push_back(a);
                }
                f[a][b]+=c;
        }

        MaxFlow(int n) : f(n+1), g(n+1), used(n+1) {
                for (int i=1;i<=n;i++) {
                        f[i]=vector<ll>(n+1);
                }
        }
};
```

25 src/graph/bridges.cpp

```cpp
// TCR
// Finds bridges and 2-edge connected components of graph
// Component of vertex x is c[x]
// Edge is bridge iff its endpoints are in different components
// Graph in form {adjacent vertex, edge id}
// Uses 1-indexing
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;

struct Bridges {
        vector<int> c, h;
        void dfs(vector<pair<int, int> >* g, int x, int pe, int d, vector<int>&
ns) {
                if (h[x]) return;
                h[x]=d;
                ns.push_back(x);
                for (auto nx:g[x]) {
```

```cpp
                    if (nx.S!=pe) {
                            dfs(g, nx.F, nx.S, d+1, ns);
                            h[x]=min(h[x], h[nx.F]);
                    }
                }
                if (h[x]==d) {
                    while (ns.size()>0) {
                            int t=ns.back();
                            c[t]=x;
                            ns.pop_back();
                            if (t==x) break;
                    }
                }
            }
        }
        Bridges(vector<pair<int, int> >* g, int n) : c(n+1), h(n+1) {
                vector<int> ns;
                for (int i=1;i<=n;i++) {
                        dfs(g, i, -1, 1, ns);
                }
        }
};
```

26   src/graph/mincostflow.cpp

```cpp
// TCR
// Finds minimum-cost k-flow
// O(V E^2 log U), where U is maximum possible flow
// Finding augmenting path is O(V E), usually faster
// Uses scaling flow and finds augmenting path with SPFA
// Only 1-directional edges allowed
// Doesn't work if graph contains negative cost cycles
// Uses 1-indexing


#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;
typedef long long ll;
typedef long double ld;

const ll inf=1e18;

struct MinCostFlow {
        // Use vector<map<int, ll> > for sparse graphs
        vector<vector<ll> > f, c;
        vector<vector<int> > g;
```

```cpp
        vector<ll> d;
        vector<int> from, inq;
        queue<int> spfa;

        void relax(int x, ll di, int p) {
                if (di>=d[x]) return;
                d[x]=di;
                from[x]=p;
                if (!inq[x]) {
                        spfa.push(x);
                        inq[x]=1;
                }
        }

        ll augment(ll x, ll s, ll fl) {
                if (x==s) return fl;
                ll r=augment(from[x], s, min(fl, f[from[x]][x]));
                f[from[x]][x]-=r;
                f[x][from[x]]+=r;
                return r;
        }

        pair<ll, ll> flow(int s, int t, ll miv, ll kf) {
                int n=g.size()-1;
                for (int i=1;i<=n;i++) {
                        d[i]=inf;
                        inq[i]=0;
                }
                relax(s, 0, 0);
                while (!spfa.empty()) {
                        int x=spfa.front();
                        spfa.pop();
                        inq[x]=0;
                        for (int nx:g[x]) {
                                if (f[x][nx]>=miv) relax(nx, d[x]+c[x][nx], x);
                        }
                }
                if (d[t]<inf) {
                        ll fl=augment(t, s, kf);
                        return {fl, fl*d[t]};
                }
                return {0, 0};
        }

// maxv is maximum possible flow on a single augmenting path
// kf is inteded flow, set infinite for maxflow
```

```cpp
// returns {flow, cost}
pair<ll, ll> getKFlow(int source, int sink, ll maxv, ll kf) {
        ll r=0;
        ll k=1;
        ll co=0;
        while (k*2<=maxv) k*=2;
        for (;k>0&&kf>0;k/=2) {
                while (1) {
                        pair<ll, ll> t=flow(source, sink, k, kf);
                        r+=t.F;
                        kf-=t.F;
                        co+=t.S;
                        if (kf==0||t.F==0) break;
                }
        }
        return {r, co};
}

void addEdge(int a, int b, ll capa, ll cost) {
        if (f[a][b]==0&&f[b][a]==0) {
                g[a].push_back(b);
                g[b].push_back(a);
        }
        f[a][b]=capa;
        c[a][b]=cost;
        c[b][a]=-cost;
}

MinCostFlow(int n) : f(n+1), c(n+1), g(n+1), d(n+1), from(n+1), inq(n+1) {
        for (int i=1;i<=n;i++) {
                f[i]=vector<ll>(n+1);
                c[i]=vector<ll>(n+1);
        }
}
};
```

27   src/graph/dynamicconnectivity.cpp

```cpp
// TCR
// O(n log n) offline solution for dynamic connectivity problem.
// Query types:
// {1, {a, b}} add edge. If edge already exists nothing happns.
// {2, {a, b}} remove edge. If no edge exists nothing happens.
// {3, {0, 0}} count number of connected components.
// Uses 1-indexing
#include <bits/stdc++.h>
```

```cpp
#define F first
#define S second
using namespace std;

struct DynamicConnectivity {
        struct Edge {
                int a, b, l, r;
        };
        vector<int> ret, tq, id, is;
        vector<vector<int> > g;
        int dfs(int x, int c) {
                id[x]=c;
                int r=is[x];
                for (int nx:g[x])
                        if (!id[nx]) r|=dfs(nx, c);
                return r;
        }
        void go(int l, int r, int n, int out, vector<Edge>& es) {
                vector<Edge> nes;
                for (int i=1;i<=n;i++) {
                        g[i].clear();
                        id[i]=0;
                        is[i]=0;
                }
                for (auto e:es) {
                        if (e.l>r||e.r<l||e.a==e.b) continue;
                        if (e.l<=l&&r<=e.r) {
                                g[e.a].push_back(e.b);
                                g[e.b].push_back(e.a);
                        }
                        else {
                                nes.push_back(e);
                                is[e.a]=1;
                                is[e.b]=1;
                        }
                }
                int i2=1;
                for (int i=1;i<=n;i++) {
                        if ((int)g[i].size()>0||is[i]) {
                                if (!id[i]) {
                                        int a=dfs(i, i2);
                                        if (!a) out++;
                                        else i2++;
                                }
                        }
                        else {
```

```
                        out++;
                    }
                }
            for (auto&e:nes) {
                    e.a=id[e.a];
                    e.b=id[e.b];
            }
            if (l==r) {
                    if (tq[l]) ret[tq[l]-1]=out+i2-1;
            }
            else {
                    int m=(l+r)/2;
                    go(l, m, i2-1, out, nes);
                    go(m+1, r, i2-1, out, nes);
            }
        }
    }
    vector<int> solve(int n, vector<pair<int, pair<int, int> > > queries) {
            map<pair<int, int>, int> ae;
            tq.resize(queries.size());
            id.resize(n+1);
            is.resize(n+1);
            g.resize(n+1);
            int qs=0;
            vector<Edge> es;
            for (int i=0;i<(int)queries.size();i++) {
                    auto q=queries[i];
                    if (q.S.F>q.S.S) swap(q.S.F, q.S.S);
                    if (q.F==1) {
                            if (ae[q.S]==0) ae[q.S]=i+1;
                    }
                    else if(q.F==2) {
                            if (ae[q.S]) {
                                    es.push_back({q.S.F, q.S.S, ae[q.S]-1, i});
                                    ae[q.S]=0;
                            }
                    }
                    else if (q.F==3) {
                            tq[i]=1+qs++;
                    }
            }
            for (auto e:ae) {
                    if (e.S) es.push_back({e.F.F, e.F.S, e.S-1,
(int)queries.size()});
            }
            ret.resize(qs);
            if ((int)queries.size()>0) go(0, (int)queries.size()-1, n, 0, es);
```

```
            return ret;
        }
};
```