Contents

1 src/other/bittricks.cpp

```cpp
// TCR
#include <bits/stdc++.h>
using namespace std;
int main(){
    // Iterate all submasks in increasing order. Does not list 0.
    int mask=13;
    for (int sub=0;(sub=(sub-mask)&mask);) {
        cout<<sub<<endl;// Should print 1 4 5 8 9 12 13
    }cout<<endl;
    // Iterate all submasks in decreasing order. Does not list 0.
    for (int sub=mask;sub;sub=(sub-1)&mask) {
        cout<<sub<<endl;// Should print 13 12 9 8 5 4 1
    }cout<<endl;
```

```
    int n=24;
    cout<<(n&-n)<<endl;// Smallest bit set. Should print 8
    cout<<__builtin_popcountll(n)<<endl;// Remember ll when using 64bit
    // Compute the next number that has the same number of bits set as n
    // Returns -1 for 0
    int t=n|(n-1);
    int w=(t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(n) + 1));
    cout<<w<<endl;// Should print 33
}
```

2 src/other/flags.txt

```
// TCR
Warnings: -Wall -Wextra -pedantic -Wshadow -Wformat=2 -Wfloat-equal -Wconversion -Wlogical-op
-Wcast-qual -Wcast-align
Runtime checks, these might make the code much slower: -D_GLIBCXX_DEBUG
-D_GLIBCXX_DEBUG_PEDANTIC -D_FORTIFY_SOURCE=2 -fsanitize=address -fsanitize=undefined
-fno-sanitize-recover -fstack-protector
Use these: -std=c++11 -O2 -Wall -Wextra -Wshadow
```

3 src/other/xmodmap.txt

```
// TCR
xmodmap -pke > lol
49 vasen yl
133 windows
less greater less greater bar bar bar
xmodmap lol
xmodmap -pm
xmodmap -e "remove mod4 = Super_L"
(clear mod4)
```

4 src/other/numbers.txt

```
// TCR
Primes
999999937, 999999999999999989, 2013265920268435457 = 2^28*13*223*2587099 + 1
Highly divisible numbers
840, 32 divisors
720720, 240 divisors
735134400, 1344 divisors
963761198400, 6720 divisors
866421317361600, 26880 divisors
897612484786617600, 103680 divisors
```

5 src/string/duval.cpp

```
// TCR
// Finds the Lyndon decomposition of a string in O(n)
// Returns the Lyndon substrings as inclusive intervals
#include <bits/stdc++.h>
using namespace std;
vector<pair<int, int> > duval(vector<int> s) {
    int k=-1;
    vector<pair<int, int> > ret;
    while (k+1<(int)s.size()) {
        int i=k+1;
        for (int j=k+2;;j++){
            if (j>=(int)s.size()||s[i]>s[j]) {
                while (k<i) {
                    ret.push_back({k+1, k+(j-i)});
                    k+=(j-i);
                }
                break;
            }
            else if(s[i]<s[j]) i=k+1;
            else i++;
        }
    }
    return ret;
}
```

6 src/string/suffixarray.cpp

```
// TCR
// Suffix array in O((n+S) log n)
// S is the size of alphabet, meaning that 0<=s[i]<S for all i
// You can change vector<int> s to string s. In that case S is 256
#include <bits/stdc++.h>
using namespace std;
vector<int> suffixArray(vector<int> s, int S) {
    int n=s.size();int N=n+S;
    vector<int> sa(n), ra(n);
    for(int i=0;i<n;i++) {sa[i]=i;ra[i]=s[i];}
    for(int k=0;k<n;k?k*=2:k++) {
        vector<int> nsa(sa), nra(n), cnt(N);
        for(int i=0;i<n;i++) nsa[i]=(nsa[i]-k+n)%n;
        for(int i=0;i<n;i++) cnt[ra[i]]++;
        for(int i=1;i<N;i++) cnt[i]+=cnt[i-1];
        for(int i=n-1;i>=0;i--) sa[--cnt[ra[nsa[i]]]]=nsa[i];
        int r=0;
        for(int i=1;i<n;i++) {
            if(ra[sa[i]]!=ra[sa[i-1]]) r++;
            else if(ra[(sa[i]+k)%n]!=ra[(sa[i-1]+k)%n]) r++;
            nra[sa[i]]=r;
        }
        ra=nra;
    }
    return sa;
}
```

7 src/string/lcparray.cpp

```
// TCR
// 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(), 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;
}
```

8 src/string/aho-corasick.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, tlink, 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);
    }
};
```

9 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;
}
```

10 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, 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;
    }
};
```

11 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());
        size_t s=hull.size();
        for (int i=1;i<n;i++) {
            while (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;
}
```

12 src/geometry/halfplaneintersection.cpp

```
// TCR
// getHPI returns the points of the half place intersection in the ccw order
// The allowed half plane is the left side of the p1 -> p2 vector
// maxD defines the bounding square so that the resulting polygon is never infinite
// May return many points even though the intersection is empty.
// Compute the area to check the emptiness.
// May return duplicate points and is generally kind of numerically unstable.
// coordinates<=1e6: eps in [1e-12, 1e-7] for ld and eps in [1e-9, 1e-7] for double
#include <bits/stdc++.h>
#define X real()
#define Y imag()
using namespace std;
typedef long double ld;
typedef complex<ld> co;
const ld eps=1e-12;
const ld maxD=1e8;
ld ccw(co a, co b) {
    return (b*conj(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(co p1, co p2) {
    if (ar(p1)!=ar(p2)) return ar(p1)>ar(p2);
    return ccw(p2, p1)<0;
}
struct hp_t {
    co a, b;
    hp_t(co p1, co p2) {
        a=p1;
        b=(p2-p1)/abs(p2-p1);
    }
    ld d(co p) const {
        return ccw(b, p-a);
    }
    bool operator==(const hp_t& o) const {
        return abs(b.X-o.b.X)<eps&&abs(b.Y-o.b.Y)<eps;
    }
    bool operator<(const hp_t& o) const {
        if ((*this)==o) {
            return d(o.a)<-eps;
        }
        return cp(b, o.b);
    }
};
co getI(hp_t a, hp_t b) {
    ld c=ccw(a.b, b.b);
    assert(!(abs(c)<eps));
    return ccw(b.a, b.b)*a.b/c+ccw(a.b, a.a)*b.b/c;
}
vector<co> getHPI(vector<hp_t> hp) {
    hp.push_back({{-maxD, -maxD}, {maxD, -maxD}});
    hp.push_back({{maxD, -maxD}, {maxD, maxD}});
    hp.push_back({{maxD, maxD}, {-maxD, maxD}});
    hp.push_back({{-maxD, maxD}, {-maxD, -maxD}});
    sort(hp.begin(), hp.end());
    hp.erase(unique(hp.begin(), hp.end()), hp.end());
    int del=0;
    vector<co> p;
    for (int i=1;i<(int)hp.size();i++) {
        while ((int)p.size()>del&&hp[i].d(p.back())<eps) p.pop_back();
        while ((int)p.size()>del&&hp[i].d(p[del])<eps) del++;
        if (del==(int)p.size()&&ccw(hp[p.size()].b, hp[i].b)<eps) return {};
        co np=getI(hp[i], hp[p.size()]);
        if (hp[del].d(np)>-eps) {
            p.push_back(np);
            hp[p.size()]=hp[i];
        }
    }
    rotate(p.begin(), p.begin()+del, p.end());
    rotate(hp.begin(), hp.begin()+del, hp.begin()+p.size()+1);
    p.resize((int)p.size()-del);
    if (p.empty()) return p;
    p.push_back(getI(hp[0], hp[p.size()]));
    return p;
}
```

13 src/geometry/anglesort.cpp

```
// TCR
// Comparasion function for sorting points around origin
// Points are sorted in clockwise order
// Works with integers and doubles
/*122
  143
  443*/
#include <bits/stdc++.h>
#define X real()
#define Y imag()
using namespace std;
typedef long double ld;
typedef long long ll;
typedef complex<ll> 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;
}
```

14 src/geometry/hullhulltan.cpp

```
// TCR
// O(log n log m)
// poinHullTan
// Finds the common tangents of a convex polygon and a point
// The polygon should be strictly convex and in counterclockwise order
// Pointhulltan returns {-1, -1} if the point is inside the polygon, otherwise
// it returs {maximal, minimal} vertices in terms of visibility from point p
// Remember to implement the special case n<=2
// Points on the boundary are considered to be inside
// hullHullTan
// Finds the common tangents of two convex polygons
// All of the conditions as above and it probably does not work if n<=2 or m<=2
// 1 is maximal and -1 is minimal
#include <bits/stdc++.h>
#define X real()
#define Y imag()
using namespace std;
typedef long long ll;
typedef complex<ll> co;
ll ccw(co a, co b, co c) {
    return ((c-a)*conj(b-a)).Y;
}
bool up(co p, vector<co>& h, int a, int b, int d) {
    int n=h.size();
    return (ll)d*ccw(p, h[(a+n)%n], h[(b+n)%n])<=0;
}
int getTanP(co p, vector<co>& h, int d) {
    int n=h.size();int mi=0;int ma=n;
    while (mi+1<ma) {
        int mid=(mi+ma)/2;
        if (up(p, h, mi, mi+1, d)) {
            if (up(p, h, mid+1, mid, d)) ma=mid;
            else if(up(p, h, mi, mid, d)) mi=mid;
            else ma=mid;
        }
        else {
            if (up(p, h, mid, mid+1, d)) mi=mid;
            else if(up(p, h, mid, mi, d)) mi=mid;
            else ma=mid;
        }
    }
    int step=0;
    if (d==1) {
        ma%=n;
        while (up(p, h, ma, ma+1, d)) {
            ma=(ma+1)%n;step++;
            assert(step<2);
        }
        return ma;
    }
    else {
        while (up(p, h, mi, mi+1, d)) {
            mi=(mi+1)%n;step++;
            if (step>=3) return -1;
        }
        if (up(p, h, mi, mi-1, d)) mi=(mi-1+n)%n;
        return mi;
    }
}
pair<int, int> pointHullTan(co p, vector<co>& h) {
    if ((int)h.size()<=2) return {0, 0};
    int t1=getTanP(p, h, -1);
    if (t1==-1) return {-1, -1};
    return {getTanP(p, h, 1), t1};
}
bool up2(vector<co>& h1, vector<co>& h, int a, int b, int d1, int d2) {
    int n=h.size();int k=getTanP(h[(b+n)%n], h1, d1);
    return (ll)d2*ccw(h[(a+n)%n], h[(b+n)%n], h1[k])<=0;
}
pair<int, int> getTanH(vector<co>& h1, vector<co>& h, int d1, int d2) {
    int n=h.size();int mi=0;int ma=n;
    while (mi+1<ma) {
        int mid=(mi+ma)/2;
        if (up2(h1, h, mi, mi+1, d1, d2)) {
            if (up2(h1, h, mid+1, mid, d1, d2)) ma=mid;
            else if(up2(h1, h, mi, mid, d1, d2)) mi=mid;
            else ma=mid;
        }
        else {
            if (up2(h1, h, mid, mid+1, d1, d2)) mi=mid;
            else if(up2(h1, h, mid, mi, d1, d2)) mi=mid;
            else ma=mid;
        }
    }
    int step=0;
    if (d2==1) {
        ma%=n;
        while (up2(h1, h, ma, ma+1, d1, d2)) {
            ma=(ma+1)%n;step++;
            assert(step<2);
        }
        return {getTanP(h[ma], h1, d1), ma};
    }
```

```
    else {
        while (up2(h1, h, mi, mi+1, d1, d2)) {
            mi=(mi+1)%n;step++;
            assert(step<3);
        }
        if (up2(h1, h, mi, mi-1, d1, d2)) mi=(mi-1+n)%n;
        return {getTanP(h[mi], h1, d1), mi};
    }
}
vector<pair<int, int> > hullHullTan(vector<co>& h1, vector<co>& h2) {
    vector<pair<int, int> > ret;
    ret.push_back(getTanH(h1, h2, 1, 1));
    ret.push_back(getTanH(h1, h2, 1, -1));
    ret.push_back(getTanH(h1, h2, -1, 1));
    ret.push_back(getTanH(h1, h2, -1, -1));
    return ret;
}
```

15 src/geometry/minkowskisum.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;
typedef complex<ll> 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;
    }
    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;
}
```

16 src/geometry/basic.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;
ld eps=1e-12;
// 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.
// Note: doesn't make much sense with non-integer CT.
bool collinear(co a, co b, co c) {
    return abs(((c-a)*conj(b-a)).Y)<eps;
}
// 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.
// Note: Only for non-integers!
// x=a*(1-t)+b*t
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);
}
// Intersection points of circles with centers p1 and p2 with radiuses r1 and r2
// The first return value is the number of intersection points, 3 for infinite
pair<int, pair<co, co> > circleIntersection(co p1, CT r1, co p2, CT r2) {
    if (norm(p1-p2)>(r1+r2)*(r1+r2)||norm(p1-p2)<(r1-r2)*(r1-r2))
        return {0, {{0, 0}, {0, 0}}};
    if (abs(p1-p2)<eps&&abs(r1-r2)<eps)
        return {3, {{p1.X, p1.Y+r1}, {p1.X+r1, p1.Y}}};
    CT a=abs(p1-p2);
    CT x=(r1*r1-r2*r2+a*a)/(2*a);
    co v1={x, sqrt(r1*r1-x*x)};
    co v2={x, -sqrt(r1*r1-x*x)};
    v1=v1*(p2-p1)/a+p1;
    v2=v2*(p2-p1)/a+p1;
    if (abs(v1-v2)<eps) return {1, {v1, v1}};
    return {2, {v1, v2}};
}
// Intersection of lines a..b and c..d
// Only for doubles
pair<int, co> lineIntersection(co a, co b, co c, co d) {
    if (collinear(a, b, c)&&collinear(a, b, d)) {
        return {2, a};
    }
    else if(abs(((b-a)/(c-d)).Y)<eps) {
        return {0, {0, 0}};
    }
    else {
        CT t=intersectionParam(a, b, c, d);
        return {1, a*(1-t)+b*t};
    }
}
// Is b between a and c
// Only for doubles
int between(co a, co b, co c) {
```

```
    return abs(abs(a-b)+abs(b-c)-abs(a-c))<eps;
}
// Intersection of segments a..b and c..d
// Only for doubles
// The first return value is the number of intersection points, 2 for infinite
// The second values are the endpoints of the intersection segment
pair<int, pair<co, co> > segmentIntersection(co a, co b, co c, co d) {
    if (abs(a-b)<eps) {
        if (between(c, a, d)) return {1, {a, a}};
        else return {0, {0, 0}};
    }
    else if (abs(c-d)<eps) {
        if (between(a, c, b)) return {1, {c, c}};
        else return {0, {0, 0}};
    }
    else if (collinear(a, b, c)&&collinear(a, b, d)) {
        if (((b-a)/(d-c)).X<0) swap(c, d);
        co beg;
        if (between(a,c,b)) beg=c;
        else if (between(c,a,d)) beg=a;
        else return {0, {{0, 0}, {0, 0}}};
        co en=d;
        if (between(c, b, d)) en=b;
        if (abs(beg-en)<eps) return {1, {beg, beg}};
        return {2, {beg, en}};
    }
    else if(abs(((b-a)/(c-d)).Y)<eps) {
        return {0, {0, 0}};
    }
    else {
        CT u=intersectionParam(a, b, c, d);
        CT v=intersectionParam(c, d, a, b);
        if (u<-eps||u>1+eps||v<-eps||v>1+eps) {
            return {0, {{0, 0}, {0, 0}}};
        }
        else {
            co p=a*(1-u)+b*u;
            return {1, {p, p}};
        }
    }
}
// Returns a point from the ray bisecting the non-reflex angle abc.
// Only for doubles. Returns 0 if the points are collinear.
pair<co,int> angleBisector(co a, co b, co c) {
    if (collinear(a,b,c)) return {{0, 0}, 0};
    co aa=(a-b)/abs(a-b);
    co cc=(c-b)/abs(c-b);
    co bb=sqrt(aa/cc);
    return {b+bb*cc, 1};
}
```

17 src/geometry/closestpoints.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) {
            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;
}
```

18 src/geometry/welzl.cpp

```
// TCR
// Minimum enclosing circle in expected O(n) time
// Remove duplicate points before using
#include <bits/stdc++.h>
#define X real()
#define Y imag()
#define F first
#define S second
```

```
using namespace std;
typedef long double ld;
typedef complex<ld> co;
pair<co, ld> md2(vector<co> R) {
    if (R.size()==0) {
        return {{0, 0}, -1};
    } else if (R.size()==1) {
        return {R[0], 0};
    } else if (R.size()==2) {
        return {(R[0]+R[1])/(ld)2.0, hypot(R[0].X-R[1].X, R[0].Y-R[1].Y)/2.0};
    } else {
        ld s=(co(0, 1)*(R[0]-R[2])*conj(R[2]-R[1])).Y/((R[0]-R[1])*conj(R[2]-R[1])).Y;
        co c=(R[0]+R[1])/(ld)2.0+co(0, 1)*s*((R[0]-R[1])/(ld)2.0);
        return {c, hypot(R[0].X-c.X, R[0].Y-c.Y)};
    }
}
pair<co, ld> md(vector<co>& P, int i, vector<co> R) {
    if (i==(int)P.size()||R.size()==3) {
        return md2(R);
    } else {
        auto D=md(P, i+1, R);
        if (hypot(P[i].X-D.F.X, P[i].Y-D.F.Y)>D.S) {
            R.push_back(P[i]);
            D=md(P, i+1, R);
        }
        return D;
    }
}
pair<co, ld> minEnclosing(vector<co> P) {
    random_shuffle(P.begin(), P.end());
    return md(P, 0, vector<co>());
}
```

19 src/geometry/3dconvexhull.cpp

```
// TCR
// 3d convex hull in O(n^2 log n)
// first 4 points should not be coplanar
struct v3 {
    ld x,y,z;
    v3() : x(0), y(0), z(0) {}
    v3(ld xx, ld yy, ld zz) : x(xx), y(yy), z(zz) {};
};
v3 operator+(v3 a, v3 b) {
    return v3(a.x+b.x, a.y+b.y, a.z+b.z);
}
v3 operator-(v3 a, v3 b) {
    return v3(a.x-b.x, a.y-b.y, a.z-b.z);
}
v3 operator*(v3 a, v3 b) {
    return v3(a.y*b.z-a.z*b.y,a.z*b.x-a.x*b.z,a.x*b.y-a.y*b.x);
}
ld operator^(v3 a, v3 b) {
    return a.x*b.x+a.y*b.y+a.z*b.z;
}
bool onSameHalfSpace(v3 a, v3 b, v3 p1, v3 p2, v3 p3) {
    v3 hlp = (p2-p1)*(p3-p1);
    return (hlp^(a-p1))*(hlp^(b-p1))>=0;
}
int n;
v3 pts[1010];
map<pair<pair<int,int>,int>,int> convHull;
void toggleHull(int i1, int i2, int i3, int ref) {
    if (convHull.count({{i1,i2},i3})) {
        convHull.erase({{i1,i2},i3});
    } else convHull[{{i1,i2},i3}]=ref;
}
void makeHull() {
    convHull[{{0,1},2}]=3;
    convHull[{{0,1},3}]=2;
    convHull[{{0,2},3}]=1;
    convHull[{{1,2},3}]=0;
    for (int i=4;i<n;i++) {
        vector<pair<pair<int,int>,pair<int,int>>> toChange;
        for (auto hullFace : convHull) {
            int i1=hullFace.F.F.F;
            int i2=hullFace.F.F.S;
            int i3=hullFace.F.S;
            v3 pt1=pts[i1];
            v3 pt2=pts[i2];
            v3 pt3=pts[i3];
            v3 ref=pts[hullFace.S];
            if (!onSameHalfSpace(pts[i],ref,pt1,pt2,pt3)) {
                toChange.push_back({{i1,i2},{i,i3}});
                toChange.push_back({{i1,i3},{i,i2}});
                toChange.push_back({{i2,i3},{i,i1}});
                toChange.push_back({{i1,i2},{i3,i}});
            }
        }
        for (auto diff : toChange) {
            toggleHull(diff.F.F,diff.F.S,diff.S.F, diff.S.S);
        }
    }
}
```

20 src/datastructure/orderedset.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
}
```

21 src/datastructure/dynamichull.cpp

```
// TCR
// Data structure that maintains a set of lines in O(log n) query time
// Operations: insert line, find the highest line at x coordinate x
// Works with integers and doubles
// Cast too large integers to doubles when comparing to avoid overflow
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll isQuery=-(1LL<<62);
struct Line {
    ll m, b;int id;
    Line(ll m_, ll b_, int id_) : m(m_), b(b_), id(id_) {}
    mutable multiset<Line>::iterator it,e;
    const Line* succ() const {
        return next(it)==e ? 0 : &*next(it);
    }
    bool operator<(const Line& rhs) const {
        if (rhs.b!=isQuery) return m<rhs.m;
        const Line* s=succ();
        if (!s) return 0;
        ll x=rhs.m;
        return b-s->b<(s->m-m)*x;
    }
};
struct DynamicHull : public multiset<Line> {
    bool bad(iterator y) {
        auto z=next(y);
        if (y==begin()) {
            if (z==end()) return 0;
            return y->m==z->m&&y->b<=z->b;
        }
        auto x=prev(y);
        if (z==end()) return y->m==x->m&&y->b<=x->b;
        return (x->b-y->b)*(z->m-y->m)>=(y->b-z->b)*(y->m-x->m);
    }
    void insertLine(ll m, ll b, int id) {
        auto y=insert({m, b, id});
        y->it=y;y->e=end();
        if (bad(y)) {erase(y);return;}
        while (next(y)!=end()&&bad(next(y))) erase(next(y));
        while (y!=begin()&&bad(prev(y))) erase(prev(y));
    }
    pair<ll, int> getMax(ll x) {
        auto l=*lower_bound({x, isQuery, 0});
        return {l.m*x+l.b, l.id};
    }
};
```

22 src/datastructure/linkcut.cpp

```
// TCR
// Link/cut tree. All operations are amortized O(log n) time
// This implementation supports finding minimum value on a path (usual MST case)
// evert(x) makes x the root
// expose(x, y) now the path from x to y is in the tree whose root is x
#include <bits/stdc++.h>
using namespace std;
struct Node {
    Node* c[2], *p;
    int id, rev;
    int v, mv;
    int isr() {
        return !p||(p->c[0]!=this&&p->c[1]!=this);
    }
    int dir() {
        return p->c[1]==this;
    }
    void upd(){
        mv=v;
        if (c[0]) mv=min(mv, c[0]->mv);
        if (c[1]) mv=min(mv, c[1]->mv);
    }
    void setc(Node* s, int d) {
        c[d]=s;
        if (s) s->p=this;
        upd();
    }
    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, int val) : id(i), v(val), mv(val) {
        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;x->upd();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 expose(Node* x, Node* y) {
    evert(x);expose(y);splay(x);
}
void cut(Node* x, Node* y) {
    expose(x, y);x->c[1]=0;x->upd();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;
}
Node* getMin(Node* x) {
    if (x->v==x->mv) {
        splay(x);
        return x;
    }
    if (x->c[0]&&x->c[0]->mv==x->mv) return getMin(x->c[0]);
    else return getMin(x->c[1]);
}
Node* getMinP(Node* x, Node* y) {
    expose(x, y);
        return getMin(x);
    }
};
```

23 src/datastructure/treap.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 {
    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) {}// 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);
}
// 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);
}
```

24 src/datastructure/HLD.cpp

```
// TCR
// Builds Heavy-light decomposition of tree in O(n) time
// getPath returns decomposed path from a to b in a vector which contains
// {{u, v}, {index[u], index[v]}} index[u]<=index[v], depth[u]<=depth[v]
// lca(a, b) is in the last path of the vector
// Uses 1-indexing
#include <bits/stdc++.h>
using namespace std;
struct HLD {
    vector<int> aps, pRoot, pLI, pRI, nPath, nPathId, p;
    int index;
    void dfs1(vector<int>* g, int x) {
        aps[x]=1;
        for (int nx:g[x]) {
            if (nx!=p[x]) {
                p[nx]=x;dfs1(g, nx);
                aps[x]+=aps[nx];
            }
        }
    }
    void dfs2(vector<int>* g, int x, int path, int pi) {
        if (path==-1) {
            path=pRoot.size();
            pRoot.push_back(x);
            pLI.push_back(index);
            pRI.push_back(index);
        }
        nPath[x]=path;
        nPathId[x]=pi;
        pRI[path]=index++;
        int ma=0;
        for (int nx:g[x]){
            if (nx!=p[x]&&aps[nx]>aps[ma]) ma=nx;
        }
        if (ma) dfs2(g, ma, path, pi+1);
        for (int nx:g[x]){
            if (nx!=p[x]&&nx!=ma) dfs2(g, nx, -1, 0);
        }
    }
    HLD(vector<int>* g, int n) : aps(n+1), nPath(n+1), nPathId(n+1), p(n+1) {
        index=0;dfs1(g, 1);
        dfs2(g, 1, -1, 0);
    }
    vector<pair<pair<int, int>, pair<int, int> > > getPath(int a, int b) {
        vector<pair<pair<int, int>, pair<int, int> > > ret;
        while (nPath[a]!=nPath[b]) {
            int pa=nPath[a];
            int pb=nPath[b];
            if (pa>pb) {
                ret.push_back({{pRoot[pa], a}, {pLI[pa], pLI[pa]+nPathId[a]}});
                a=p[pRoot[pa]];
            }
            else {
                ret.push_back({{pRoot[pb], b}, {pLI[pb], pLI[pb]+nPathId[b]}});
                b=p[pRoot[pb]];
            }
        }
        int pa=nPath[a];
        if (nPathId[a]>nPathId[b]) swap(a, b);
        ret.push_back({{a, b}, {pLI[pa]+nPathId[a], pLI[pa]+nPathId[b]}});
        return ret;
    }
};
```

25 src/math/berlekampmassey.cpp

```
// TCR
// Berlekamp massey
// Give a sequence of integers in constructor and query with get(index)
// Use just the solve to get the coefficients of the recursion
// remember to negate the coefficients
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll powmod(ll a, ll p, ll modd) {
    if (p==0) return 1;
    if (p%2==0) {
        a=powmod(a, p/2, modd);
        return (a*a)%modd;
    }
    return (a*powmod(a, p-1, modd))%modd;
}
ll invp(ll a, ll p) {
    return powmod(a, p - 2, p);
}
vector<ll> solve(vector<ll> S, ll mod) {
    vector<ll> C = {1};
    vector<ll> B = {1};
    ll L = 0;ll m = 1;ll b = 1;ll N = S.size();
    for (ll i = 0; i < N; i++) {
        ll d = S[i];
        for (ll j = 1; j <= L; j++) {
```

```
            d += C[j]*S[i - j];d %= mod;
        }
        if (d == 0) {
            m++;
        } else if (2*L <= i) {
            vector<ll> T = C;
            ll a = (invp(b, mod)*d)%mod;
            for (int j=0;j<i+1-2*L;j++) {
                C.push_back(0);
            }
            L=i+1-L;
            for (ll j = m; j <= L; j++) {
                C[j] -= a*B[j - m];C[j] %= mod;
            }
            B = T;b = d;m = 1;
        } else {
            ll a = (invp(b, mod)*d)%mod;
            for (ll j = m; j < m+(int)B.size(); j++) {
                C[j] -= a*B[j - m];C[j] %= mod;
            }
            m++;
        }
    }
    for (ll i = 0; i <= L; i++) {
        C[i] += mod;C[i] %= mod;
    }
    return C;
}
struct LinearRecurrence {
    vector<vector<ll> > mat;
    vector<ll> seq;
    ll mod;
    vector<vector<ll> > mul(vector<vector<ll> > a, vector<vector<ll> > b) {
        int n=a.size();
        vector<vector<ll> > ret(n);
        for (int i=0;i<n;i++){
            ret[i].resize(n);
            for (int j=0;j<n;j++){
                ret[i][j]=0;
                for (int k=0;k<n;k++){
                    ret[i][j]+=a[i][k]*b[k][j];
                    ret[i][j]%=mod;
                }
            }
        }
        return ret;
    }
    vector<vector<ll> > pot(vector<vector<ll> > m, ll p) {
        if (p==1) return m;
        if (p%2==0) {
            m=pot(m, p/2);
            return mul(m, m);
        }
        else {
            return mul(m, pot(m, p-1));
        }
    }
    ll get(ll index) {
        if (index<(ll)mat.size()) {
            return seq[index];
        }
        vector<vector<ll> > a=pot(mat, index-(ll)mat.size()+1);
        ll v=0;
        for (int i=0;i<(int)mat.size();i++) {
            v+=a[0][i]*seq[(int)mat.size()-i-1];
            v%=mod;
        }
        return v;
    }
    LinearRecurrence(vector<ll> S, ll mod_) {
        seq=S;
        mod=mod_;
        vector<ll> C=solve(S, mod);
        int n=C.size()-1;
        mat.resize(n);
        for (int i=0;i<n;i++) {
            mat[i].resize(n);
        }
        for (int i=0;i<n;i++) {
            mat[0][i]=(mod-C[i+1])%mod;
        }
        for (int i=1;i<n;i++) {
            mat[i][i-1]=1;
        }
    }
};
```

26 src/math/crt.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}};
}
```

27 src/math/fftmod.cpp

```
// TCR
// Precise FFT modulo mod
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long long lll;
// Number of form (2^25)*k+1
const lll mod=2113929217; // between 2*10^9 and 2^31
// Number whose order mod mod is 2^25
const lll root=1971140334;
const lll root_pw=1<<25;
// 128 bit
// typedef __int128 lll;
// const lll mod=2013265920268435457; // between 2*10^18 and 2^61
// const lll root=1976010382590097340;
// const lll root_pw=1<<28;
lll pot(lll x, lll 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;
}
lll inv(lll x) {
    return pot(x, mod-2);
}
vector<lll> fft (vector<lll> a, int d) {
    lll 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) {
        lll 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) {
            lll w = 1;
            for (int j=0;j<len/2;j++) {
                lll u = a[i+j];
                lll 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) {
        lll nrev=inv(n);
        for (int i=0;i<n;i++) a[i]=(a[i]*nrev)%mod;
    }
    return a;
}
vector<lll> conv(const vector<ll>& a, const vector<ll>& b) {
    int as=a.size(), bs=b.size();
    int n=1;
    while (n<as+bs-1) n*=2;
    vector<lll> aa(n*2), bb(n*2);
    for (int i=0;i<as;i++) aa[i]=a[i];
    for (int i=0;i<bs;i++) bb[i]=b[i];
    aa=fft(aa, 1);bb=fft(bb, 1);
    vector<lll> 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<lll> c=conv(a, b);
    for (lll t:c) {
        cout<<(ll)t<<endl;
    }
}
```

28 src/math/gaussjordan.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;
// 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;
}
// 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};
    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;
}
```

29 src/math/fht.cpp

```
// TCR
// Fast Hadamard Transform for computing xor, and, or, convolutions in O(n log n)
// The length of the input vector must be a power of 2
// Works also in mod arithmetic, just remember to handle negative numbers and
// the division in inverse xor
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
void fht(vector<ll>& a, vector<vector<ll> > m) {
    for (int len=1;2*len<=(int)a.size();len*=2) {
        for (int i=0;i<(int)a.size();i+=2*len) {
            for (int j=0;j<len;j++) {
```

```
                ll u=a[i+j];
                ll v=a[i+len+j];
                a[i+j]=m[0][0]*u+m[0][1]*v;
                a[i+len+j]=m[1][0]*u+m[1][1]*v;
}}}}
void xorTr(vector<ll>& a) {fht(a, {{1, 1}, {1, -1}});}
void xorInv(vector<ll>& a) {
    fht(a, {{1, 1}, {1, -1}});
    for (int i=0;i<(int)a.size();i++) a[i]/=(ll)a.size();
}
void andTr(vector<ll>& a) {fht(a, {{0, 1}, {1, 1}});}
void andInv(vector<ll>& a) {fht(a, {{-1, 1}, {1, 0}});}
void orTr(vector<ll>& a) {fht(a, {{1, 1}, {1, 0}});}
void orInv(vector<ll>& a) {fht(a, {{0, 1}, {1, -1}});}
int main() {// Should print 92 73 78 69
    vector<ll> a={3, 2, 7, 1};
    vector<ll> b={5, 4, 9, 6};
    xorTr(a);xorTr(b);
    vector<ll> c(4);
    for (int i=0;i<4;i++) c[i]=a[i]*b[i];
    xorInv(c);
    for (ll t:c) cout<<t<<endl;
}
```

30 src/math/simplex.cpp

```
// TCR
// Source: https://github.com/jaehyunp/stanfordacm/blob/master/code/Simplex.cc
// Two-phase simplex algorithm for solving linear programs of the form
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
// OUTPUT: value of the optimal solution (inf if unbounded
//         above, -inf if infeasible)
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
struct LPSolver {
    const ld eps=1e-9;
    const ld inf=1e30;
    int m, n;
    vector<int> N, B;
    vector<vector<ld> > D;
    LPSolver(vector<vector<ld> >& A, vector<ld>& b, vector<ld>& c) :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, vector<ld>(n + 2)) {
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) D[i][j] = A[i][j];}
        for (int i = 0; i < m; i++) {
            B[i] = n + i; D[i][n] = -1; D[i][n + 1] = b[i];}
        for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
        N[n] = -1; D[m + 1][n] = 1;
    }
    void Pivot(int r, int s) {
        ld inv = 1.0 / D[r][s];
        for (int i = 0; i < m + 2; i++) if (i != r)
            for (int j = 0; j < n + 2; j++) if (j != s)
                D[i][j] -= D[r][j] * D[i][s] * inv;
        for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
        for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
        D[r][s] = inv;
        swap(B[r], N[s]);
    }
    bool Simplex(int phase) {
        int x = phase == 1 ? m + 1 : m;
        while (true) {
            int s = -1;
            for (int j = 0; j <= n; j++) {
                if (phase == 2 && N[j] == -1) continue;
                if (s==-1||D[x][j]<D[x][s]||(D[x][j]==D[x][s]&&N[j]<N[s])) s=j;
            }
            if (D[x][s] > -eps) return true;
            int r = -1;
            for (int i = 0; i < m; i++) {
                if (D[i][s] < eps) continue;
                if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
                    ((D[i][n + 1]/D[i][s])==(D[r][n+1]/D[r][s])&&B[i]<B[r]))r=i;
            }
            if (r == -1) return false;
            Pivot(r, s);
        }
    }
    ld Solve(vector<ld>& x) {
        int r = 0;
        for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
        if (D[r][n + 1] < -eps) {
            Pivot(r, n);
            if (!Simplex(1) || D[m + 1][n + 1] < -eps) return -inf;
            for (int i = 0; i < m; i++) if (B[i] == -1) {
                int s = -1;
                for (int j = 0; j <= n; j++)
                if (s==-1||D[i][j]<D[i][s]||(D[i][j]==D[i][s]&&N[j]<N[s])) s=j;
                Pivot(i, s);
            }
        }
        if (!Simplex(2)) return inf;
        x = vector<ld>(n);
        for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
        return D[m][n + 1];
    }
};
int main(){
```

```
    const int m = 4;const int n = 3;
    ld _A[m][n] = {{ 6, -1, 0 },{ -1, -5, 0 },{ 1, 5, 1 },{ -1, -5, -1 }};
    ld _b[m] = { 10, -4, 5, -5 };
    ld _c[n] = { 1, -1, 0 };
    vector<vector<ld>> A(m);vector<ld> b(_b, _b + m);vector<ld> c(_c, _c + n);
    for (int i = 0; i < m; i++) A[i] = vector<ld>(_A[i], _A[i] + n);
    LPSolver solver(A, b, c);
    vector<ld> x;
    ld value = solver.Solve(x);
    cerr << "VALUE: " << value << endl; // VALUE: 1.29032
    cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
    for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
}
```

31 src/math/fft.cpp

```
// TCR
// Fast Fourier transform and convolution using it
// O(n log n)
// Is accurate with integers if the numbers of the result array are <= 4e15
// Also accurate if input <= 1e6 and the lengths of input arrays are 2e5
// Can be speed up by a factor of 2 by implementing the complex class
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
typedef long long ll;
typedef complex<ld> co;
const ld PI=atan2((ld)0, (ld)-1);
void fft(vector<co>&a, int n, int k) {
    vector<co> ww(n);
    ww[1]=co(1, 0);
    for (int t=0;t<k-1;t++) {
        co c=polar((ld)1, PI/n*(1<<(k-1-t)));
        int p2=(1<<t),p3=p2*2;
        for (int j=p2;j<p3;j++) ww[j*2+1]=(ww[j*2]=ww[j])*c;
    }
    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(a[i], a[u]);
    }
    for (int l=1;l<n;l*=2) {
        for (int i=0;i<n;i+=l) {
            for (int it=0,j=i+l,w=l;it<l;it++,i++,j++) {
                co t=a[j]*ww[w++];
                a[j]=a[i]-t;
                a[i]=a[i]+t;
            }
        }
    }
}
vector<ll> conv(const vector<ll>& a, const vector<ll>& b) {
    int as=a.size(), bs=b.size();
    if (as*bs==0) return {};
    int k=0;
    while ((1<<k)<as+bs-1) k++;
    int n=1<<k;
    vector<co> c(n+1);
    for (int i=0;i<n;i++) {
        if (i<as) c[i]=a[i];
        if (i<bs) c[i]={c[i].real(), (ld)b[i]};
    }
    fft(c, n, k);
    c[n]=c[0];
    for (int i=0;i<=n-i;i++) {
        c[i]=conj(c[i]*c[i]-conj(c[n-i]*c[n-i]))*co(0,(ld)1/n/4);
        c[n-i]=conj(c[i]);
    }
    fft(c, n, k);
    vector<ll> r(as+bs-1);
    for (int i=0;i<as+bs-1;i++) r[i]=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;
}
```

32 src/math/pollard-rho.cpp

```
// TCR
// Pollard Rho Integer factorization
// Support of 128 bit integers is required to factor over 32 bit integers
// requires isPrime function
// expected time complexity is O(n^(1/4))
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef __int128 lll;
void step(ll& x, ll n, ll c) {x=(lll)((lll)x*(lll)x+(lll)c)%n;}
void rFactor(ll n, map<ll, ll>& r) {
    while (n%2==0) {
        n/=2;r[2]++;
    }
    if (n==1) return;
    if (isPrime(n)) r[n]++;
    else {
        while (1) {
            ll x=rand()%n;ll y=x;
            ll c=rand()%n;
            for (ll i=0;i*i<=n;i++) {
                step(x, n, c);step(x, n, c);step(y, n, c);
                ll g=__gcd(max(x, y)-min(x, y), n);
                if (g==n) break;
```

```
                else if(g>1) {
                    rFactor(n/g, r);
                    rFactor(g, r);
                    return;
                }
            }
        }
    }
}
map<ll, ll> factor(ll n) {
    map<ll, ll> ret;
    if (n>1) rFactor(n, ret);
    return ret;
}
```

33 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;
}
```

34 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
}
```

35 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)}};
}
```

36 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
// Returns strongly connected components of the graph in vector ret
// n is the size of the graph, g is the adjacency list
#include <bits/stdc++.h>
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);
    }
    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());
                }
            }
        }
};
```

37 src/graph/circulation.cpp

```
// TCR
// Min cost circulation
// O(VE) on average, probably something like O(ans * E) worst case
// Use by adding edges with addEdge and then calling minCostCirculation
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
template<int V, int E> struct Circulation {
    struct Edge {
        int a, b;
        ll ca, co;
    } es[E*2];
    int eu=0,cookie=1;
    int how[V+1], good[V+1], bio[V+1];
    ll dist[V+1];
    void addEdge(int from, int to, ll ca, ll co) {
        es[eu++]={from, to, ca, co};
        es[eu++]={to, from, 0, -co};
    }
    void reset() {
        for (int i=1;i<=V;i++) {
            dist[i]=0;how[i]=-1;bio[i]=0;
        }
    }
    bool relax() {
        bool ret=false;
        for (int e=0;e<eu;e++) {
            if (es[e].ca) {
                int x=es[e].a;int y=es[e].b;
                if (dist[x]+es[e].co<dist[y]) {
                    dist[y]=dist[x]+es[e].co;
                    how[y]=e;ret=true;
                }
            }
        }
        return ret;
    }
    ll cycle(int s, bool flip = false) {
        int x=s;ll c=es[how[x]].ca;
        do {
            int e=how[x];c=min(c, es[e].ca);x=es[e].a;
        } while (x!=s);
        ll cost=0;
        do {
            int e=how[x];
```

```
            if (flip) {
                es[e].ca-=c;es[e^1].ca+=c;
            }
            cost+=es[e].co*c;x=es[e].a;
        } while (x!=s);
        return cost;
    }
    ll push(int x) {
        for (cookie++;bio[x]!=cookie;x=es[how[x]].a) {
            if (!good[x]||how[x]==-1||es[how[x]].ca==0) return 0;
            bio[x]=cookie;good[x]=false;
        }
        return cycle(x)>=0?0:cycle(x, true);
    }
    ll minCostCirculation() {
        reset();
        ll cost=0;
        for (int step=0;step<2*V;step++) {
            if (step == V) reset();
            if (!relax()) continue;
            for (int i=1;i<=V;i++) good[i]=true;
            for (int i=1;i<=V;i++) if (ll w=push(i)) {cost+=w;step=0;}
        }
        return cost;
    }
};
```

38 src/graph/eulertour.cpp

```
// TCR
// 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;
            }
        }
    }
};
```

39 src/graph/cutvertices.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();
                    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);
            }
        }
    }
};
```

40 src/graph/scalingflow.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 {
    struct e {
        int to,r;
        ll f;
    };
    vector<vector<e> > 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 (auto& nx:g[x]) {
            if (used[nx.to]!=cc&&nx.f>=miv) {
                ll r=flow(nx.to, t, min(fl, nx.f), miv);
                if (r>0) {
                    nx.f-=r;g[nx.to][nx.r].f+=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) {
        g[a].push_back({b, (int)g[b].size(), c});
        g[b].push_back({a, (int)g[a].size()-1, 0});
    }
    MaxFlow(int n) : g(n+1), used(n+1) {}
};
```

41 src/graph/bridges.cpp

```
// TCR
// Finds bridges and 2-edge connected components of graph
// Component of vertex x is c[x]
// Edge is a 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]) {
```

```
            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);
    }
};
```

42 src/graph/matching.cpp

```
// TCR
// General matching in O(n^3) time
#include <bits/stdc++.h>
using namespace std;
struct Matching {
    int n;
    vector<int> vLabel, ma, s, u;
    queue<int> q;
    void rm(int x, int y) {
        int m=ma[x];ma[x]=y;
        if (ma[m]==x) {
            if (vLabel[x]<=n) {
                ma[m]=vLabel[x];
                rm(vLabel[x], m);
            }
            else {
                int a=1+(vLabel[x]-n-1)/n;
                int b=1+(vLabel[x]-n-1)%n;
                rm(a, b);rm(b, a);
            }
        }
    }
    void tr(int x) {
        for (int i=1;i<=n;i++) s[i]=ma[i];
        rm(x, x);
        for (int i=1;i<=n;i++) {
            if (ma[i]!=s[i]) u[i]++;
            ma[i]=s[i];
        }
    }
    void rl(int x, int y) {
        for (int i=1;i<=n;i++) u[i]=0;
        tr(x);tr(y);
        for (int i=1;i<=n;i++) {
            if (u[i]==1&&vLabel[i]<0) {
                vLabel[i]=n+x+(y-1)*n;
                q.push(i);
            }
        }
    }
    vector<pair<int, int> > solve(vector<int>* g) {
        for (int i=1;i<=n;i++) {
            if (ma[i]==0) {
                for (int j=1;j<=n;j++) vLabel[j]=-1;
                vLabel[i]=0;q.push(i);
                while (!q.empty()) {
                    int x=q.front();q.pop();
                    for (int y:g[x]) {
                        if (ma[y]==0&&i!=y) {
                            ma[y]=x;rm(x, y);
                            while (!q.empty()) q.pop();
                            break;
                        }
                        if (vLabel[y]>=0) {
                            rl(x, y);
                            continue;
                        }
                        if (vLabel[ma[y]]<0) {
                            vLabel[ma[y]]=x;
                            q.push(ma[y]);
                        }
                    }
                }
            }
        }
        vector<pair<int, int> > res;
        for (int i=1;i<=n;i++) if (ma[i]>i) res.push_back({i, ma[i]});
        return res;
    }
    Matching(int nn) : n(nn), vLabel(n+1), ma(n+1), s(n+1), u(n+1) {}
};
```

43 src/graph/mincostflow.cpp

```
// TCR
// Find minimum-cost k-flow
// O(VE) normalizing and O(E log V) for each augmenting path
// getKFlow augments at most k flow and returns {flow, cost}
// Uses 1-indexing
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;
typedef long long ll;
const ll inf=1e18;
```

```
template<int V, int E> struct MinCostFlow {
    struct Edge {
        int a, b;
        ll ca, co;
    } es[E*2];
    int eu=0,nmz=0;
    vector<int> g[V+1];
    ll p[V+1],d[V+1];
    int fr[V+1],u[V+1];
    void addEdge(int a, int b, ll ca, ll co) {
        nmz=0;
        es[eu++]={a, b, ca, co};
        es[eu++]={b, a, 0, -co};
        g[a].push_back(eu-2);
        g[b].push_back(eu-1);
    }
    void normalize(int source) {
        if (nmz) return;nmz=1;
        for (int i=1;i<=V;i++) {
            p[i]=inf;u[i]=0;
        }
        p[source]=0;
        queue<int> q;q.push(source);
        while (!q.empty()){
            int x=q.front();
            u[x]=0;q.pop();
            for (int e:g[x]) {
                if (es[e].ca>0&&p[x]+es[e].co<p[es[e].b]) {
                    p[es[e].b]=p[x]+es[e].co;
                    if (!u[es[e].b]) {
                        u[es[e].b]=1;
                        q.push(es[e].b);
                    }
                }
            }
        }
    }
    ll augment(int x, ll fl) {
        if (fr[x]==-1) return fl;
        ll r=augment(es[fr[x]].a, min(fl, es[fr[x]].ca));
        es[fr[x]].ca-=r;
        es[fr[x]^1].ca+=r;
        return r;
    }
    pair<ll, ll> flow(int source, int sink, ll mf) {
        priority_queue<pair<ll, int> > dij;
        for (int i=1;i<=V;i++) {
            u[i]=0;fr[i]=-1;d[i]=inf;
        }
        d[source]=0;
        dij.push({0, source});
        while (!dij.empty()) {
            auto x=dij.top();dij.pop();
            if (u[x.S]) continue;
            u[x.S]=1;
            for (int e:g[x.S]) {
                ll nd=d[x.S]+es[e].co+p[x.S]-p[es[e].b];
                if (es[e].ca>0&&nd<d[es[e].b]) {
                    d[es[e].b]=nd;
                    fr[es[e].b]=e;
                    dij.push({-nd, es[e].b});
                }
            }
        }
        ll co=d[sink]+p[sink];
        for (int i=1;i<=V;i++) {
            if (fr[i]!=-1) p[i]+=d[i];
        }
        if (u[sink]) {
            ll fl=augment(sink, mf);
            return {fl, fl*co};
        }
        else return {0, 0};
    }
    pair<ll, ll> getKFlow(int source, int sink, ll k) {
        ll fl=0;ll co=0;
        normalize(source);
        while (1) {
            pair<ll, ll> t=flow(source, sink, k);
            fl+=t.F;k-=t.F;co+=t.S;
            if (k==0||t.F==0) break;
        }
        return {fl, co};
    }
};
```

44 src/graph/dynamicconnectivity.cpp

```
// TCR
// O(n log^2 n) offline solution for the dynamic connectivity problem.
// The purpose of this code is not to be a black box but rather a template
// Optimize memory by using vector<int>& es instead of vector<Edge>& es
// Uses 1-indexing
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;
struct Edge {
    int a, b, l, r;
};
struct DynamicConnectivity {
    vector<int> u, us;
    vector<pair<int, int> > st;
    int getu(int a) {
        if (u[a]==a) return a;
        else return getu(u[a]);
    }
```

```
    int uni(int a, int b) {
        a=getu(a);
        b=getu(b);
        if (a==b) return 0;
        if (us[a]>us[b]) swap(a, b);
        st.push_back({a, b});
        u[a]=b;
        us[b]+=us[a];
        return 1;
    }
    void undo() {
        us[st.back().S]-=us[st.back().F];
        u[st.back().F]=st.back().F;
        st.pop_back();
    }
    void go(int l, int r, vector<Edge>& es) {
        int use=0;
        vector<Edge> nes;
        for (auto e:es) {
            if (e.l>r||e.r<l) continue;
            if (e.l<=l&&r<=e.r) use+=uni(e.a, e.b);
            else nes.push_back(e);
        }
        if (l<r) {
            go(l, (l+r)/2, nes);
            go((l+r)/2+1, r, nes);
        }
        for (int i=0;i<use;i++) undo();
    }
    DynamicConnectivity(int n) : u(n+1), us(n+1) {
        for (int i=1;i<=n;i++) {
            u[i]=i;
            us[i]=1;
        }
    }
};
```

45 src/graph/dinic.cpp

```
// TCR
// Dinic's algorithm for maxflow
// O(n^2 m) theoretical, really fast (near linear) in practice
// O(m^(3/2)) in unit network graphs
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll inf=1e18;
struct MaxFlow {
    struct edge {
        int a, b;
        ll c;
    };
    vector<edge> es;
    vector<vector<int> > g;
    vector<int> d, pt;
    MaxFlow(int n) : g(n+1), d(n+1), pt(n+1) {}
    void addEdge(int a, int b, ll c) {
        es.push_back({a, b, c});
        g[a].push_back((int)es.size()-1);
        es.push_back({b, a, 0});
        g[b].push_back((int)es.size()-1);
    }
    bool bfs(int source, int sink) {
        queue<int> q({source});
        fill(d.begin(), d.end(), (int)g.size()+1);
        d[source]=0;
        while(!q.empty()) {
            int x=q.front();q.pop();
            if (x==sink) break;
            for (int k:g[x]) {
                edge& e=es[k];
                if (e.c>0&&d[e.b]>d[e.a]+1) {
                    d[e.b]=d[e.a]+1;
                    q.push(e.b);
                }
            }
        }
        return d[sink]!=(int)g.size()+1;
    }
    ll flow(int x, int sink, ll fl=inf) {
        if (x==sink||fl==0) return fl;
        for (int& i=pt[x];i<(int)g[x].size();i++) {
            edge& e=es[g[x][i]];
            edge& oe=es[g[x][i]^1];
            if (d[e.b]==d[e.a]+1) {
                if (ll pf=flow(e.b, sink, min(e.c, fl))) {
                    e.c-=pf;
                    oe.c+=pf;
                    return pf;
                }
            }
        }
        return 0;
    }
    ll getMaxFlow(int source, int sink) {
        ll r=0;
        while (bfs(source, sink)) {
            fill(pt.begin(), pt.end(), 0);
            while (ll t=flow(source, sink)) r+=t;
        }
        return r;
    }
};
```