目 次

1 備忘録

1.1 テンプレート

```cpp
#[[#include]]# <iostream>
#[[#include]]# <algorithm>
#[[#include]]# <iomanip>
#[[#include]]# <map>
#[[#include]]# <set>
#[[#include]]# <queue>
#[[#include]]# <stack>
#[[#include]]# <numeric>
#[[#include]]# <bitset>
#[[#include]]# <cmath>

static const int MOD = 1000000007;
using ll = long long;
using u32 = uint32_t;
using namespace std;

template<class T> constexpr T INF = ::numeric_limits<T>::max()/32*15+208;

int main() {

    return 0;
}
```

1.2 makev chmax chmin

```cpp
template <class T, class U>
vector<T> make_v(U size, const T& init){ return vector<T>(static_cast<size_t>(size), init); }

template<class... Ts, class U>
auto make_v(U size, Ts... rest) { return vector<decltype(make_v(rest...))>(static_cast<size_t>(size), make_v(rest...)); }

template<class T> void chmin(T &a, const T &b){ a = (a < b ? a : b); }
template<class T> void chmax(T &a, const T &b){ a = (a > b ? a : b); }
```

1.3 ファイルを作成する

テンプレートを書いたら、bash を開いて、以下のコマンドを打つ。

```
for i in {A..H}; do cp main.cpp $i.cpp; done
```

2　使いそうなライブラリ

2.1　繰り返し二乗法

```cpp
template <class T>
T pow_ (T x, T n, T M){
    uint64_t u = 1, xx = x;
    while (n > 0){
        if (n&1) u = u * xx % M;
        xx = xx * xx % M;
        n >>= 1;
    }
    return static_cast<T>(u);
};
```

2.2　約数列挙

```cpp
template<class T>
vector<T> divisor(T n){
    vector<T> ret;
    for(T i = 1; i * i <= n; i++) {
        if(n % i == 0) {
            ret.push_back(i);
            if(i * i != n) ret.push_back(n / i);
        }
    }
    sort(begin(ret), end(ret));
    return(ret);
}
```

2.3　素数列挙・素因数分解

```cpp
vector<int> get_prime(int n){
    if(n <= 1) return vector<int>();
    vector<bool> is_prime(n+1, true);
    vector<int> prime;
    is_prime[0] = is_prime[1] = 0;
    for (int i = 2; i <= n; ++i) {
        if(is_prime[i]) prime.emplace_back(i);
        for (auto &&j : prime){
            if(i*j > n) break;
            is_prime[i*j] = false;
            if(i % j == 0) break;
        }
    }
    return prime;
}
const auto primes = get_prime(65535);

template<class T>
vector<T> prime_factor(T n){
    vector<T> res;
    for (auto &&i : primes) {
        while (n % i == 0){
            res.emplace_back(i);
            n /= i;
        }
    }
    if(n != 1) res.emplace_back(n);
    return res;
}
```

2.4 ダイクストラ法

```
template <typename T>
struct edge {
    int from, to; T cost;
    edge(int to, T cost) : from(-1), to(to), cost(cost) {}
    edge(int from, int to, T cost) : from(from), to(to), cost(cost) {}
};

template <typename T>
vector<T> dijkstra(int s,vector<vector<edge<T>>> &G){
    size_t n=G.size();
    vector<T> d(n, INF<T>);
    priority_queue<pair<T, int>,vector<pair<T, int>>,greater<>> Q;
    d[s]=0;
    Q.emplace(0,s);
    while(!Q.empty()){
        T cost; int i;
        tie(cost, i) = Q.top(); Q.pop();
        if(d[i] < cost) continue;
        for (auto &&e : G[i]) {
            auto cost2 = cost + e.cost;
            if(d[e.to] <= cost2) continue;
            d[e.to] = cost2;
            Q.emplace(d[e.to], e.to);
        }
    }
    return d;
}
```

2.5 kruskal 法 (最小全域木)

```
template <typename T>
struct edge {
    int from, to;
    T cost;

    edge(int to, T cost) : from(-1), to(to), cost(cost) {}
    edge(int from, int to, T cost) : from(from), to(to), cost(cost) {}

    explicit operator int() const {return to;}
};

class UnionFind {
    vector<int> uni;
    int n;
public:
    explicit UnionFind(int n) : uni(static_cast<u32>(n), -1) , n(n){};

    int root(int a){
        if (uni[a] < 0) return a;
        else return (uni[a] = root(uni[a]));
    }

    bool unite(int a, int b) {
        a = root(a);
        b = root(b);
        if(a == b) return false;
        if(uni[a] > uni[b]) swap(a, b);
        uni[a] += uni[b];
        uni[b] = a;
        return true;
    }
```

```
};

template< typename T >
T kruskal(vector<edge<T>> &G, int V)
{
    sort(begin(G), end(G), [](const edge< T > &a, const edge< T > &b) { return
    ↪  (a.cost < b.cost); });
    UnionFind tree(V);
    T ret = 0;
    for(auto &e : G) {
        if(tree.unite(e.from, e.to)) ret += e.cost;
    }
    return (ret);
}
```

2.6 Union-Find

```
class UnionFind {
    int n;
    vector<int> uni;
public:
    explicit UnionFind(int n) : uni(static_cast<u32>(n), -1) , n(n){};

    int root(int a){
        if (uni[a] < 0) return a;
        else return (uni[a] = root(uni[a]));
    }

    bool unite(int a, int b) {
        a = root(a);
        b = root(b);
        if(a == b) return false;
        if(uni[a] > uni[b]) swap(a, b);
        uni[a] += uni[b];
        uni[b] = a;
        return true;
    }

    int size(int i){ return -uni[root(i)]; }
    bool same(int a, int b) { return root(a) == root(b); }
};
```

2.7 正方行列

```
template<class T, size_t SIZE>
struct SquareMatrix {
    using ar = array<T, SIZE>;
    using mat = array<ar, SIZE>;
    mat A;
    SquareMatrix() = default;
    static SquareMatrix I(T e){
        SquareMatrix X;
        for (int i = 0; i < SIZE; ++i) {
            X[i][i] = e;
        }
        return X;
    }

    inline const ar &operator[](int k) const{ return (A.at(k)); }
    inline ar &operator[](int k) { return (A.at(k)); }
    SquareMatrix &operator+= (const SquareMatrix &B){
```

```
        for (int i = 0; i < SIZE; ++i) {
            for (int j = 0; j < SIZE; ++j) {
                (*this)[i][j] += B[i][j];
            }
        }
        return (*this);
    }

    SquareMatrix &operator-= (const SquareMatrix &B){
        for (int i = 0; i < SIZE; ++i) {
            for (int j = 0; j < SIZE; ++j) {
                (*this)[i][j] -= B[i][j];
            }
        }
        return (*this);
    }

    SquareMatrix &operator*=(const SquareMatrix &B) {
        SquareMatrix C;
        for (int i = 0; i < SIZE; ++i) {
            for (int j = 0; j < SIZE; ++j) {
                for (int k = 0; k < SIZE; ++k) {
                    C[i][j] += ((*this)[i][k] * B[k][j]);
                }
            }
        }
        A.swap(C.A);
        return (*this);
    }

    SquareMatrix pow(ll n) const {
        SquareMatrix a = (*this), res = I(T(1));
        while(n > 0){
            if(n & 1) res *= a;
            a *= a;
            n >>= 1;
        }
        return res;
    }
    SquareMatrix operator+(const SquareMatrix &B) const {return SquareMatrix(*this)
    ↪  += B;}
    SquareMatrix operator-(const SquareMatrix &B) const {return SquareMatrix(*this)
    ↪  -= B;}
    SquareMatrix operator*(const SquareMatrix &B) const {return SquareMatrix(*this)
    ↪  *= B;}
};
```

2.8 Dinic 法 (最大流)

```
template<class T>
class Dinic {
    struct edge {
        int to{}, rev{};
        T cap;
        edge() = default;
        edge(int to, T cap, int rev):to(to), cap(cap), rev(rev) {}
    };
    int n{};
    vector<vector<edge>> G;
    vector<int> level, iter;
public:
    Dinic() = default;
```

```
14      explicit Dinic(int sz): n(sz), G(n), level(n), iter(n){}
15      void add_edge(int from, int to, T cap, bool directed){
16          G[from].emplace_back(to, cap, G[to].size());
17          G[to].emplace_back(from, directed?0:cap, G[from].size()-1);
18      }
19
20      void bfs(int s){
21          fill(level.begin(), level.end(), -1);
22          queue<int> Q;
23          level[s] = 0;
24          Q.emplace(s);
25          while(!Q.empty()){
26              int v = Q.front(); Q.pop();
27              for (auto &&e : G[v]) {
28                  if(e.cap > 0 && level[e.to] < 0){
29                      level[e.to] = level[v]+1;
30                      Q.emplace(e.to);
31                  }
32              }
33          }
34      }
35
36      T dfs(int v, int t, T f){
37          if(v == t) return f;
38          for (int &i = iter[v]; i < G[v].size(); ++i) {
39              edge &e = G[v][i];
40              if(e.cap > 0 && level[v] < level[e.to]){
41                  T d = dfs(e.to, t, min(f, e.cap));
42                  if(d > 0){
43                      e.cap -= d;
44                      G[e.to][e.rev].cap += d;
45                      return d;
46                  }
47              }
48          }
49          return 0;
50      }
51
52      T flow(int s, int t, T lim = INF<T>){
53          T fl = 0;
54          while(true){
55              bfs(s);
56              if(level[t] < 0 || lim == 0) return fl;
57              fill(iter.begin(), iter.end(), 0);
58              T f;
59              while((f=dfs(s,t, lim))>0){
60                  fl += f;
61                  lim -= f;
62              }
63          }
64      }
65
66      bool back_edge(int s, int t, int from, int to){
67          for (auto &&e : G[from]) {
68              if(e.to == to) {
69                  if(e.cap == 0 && flow(from, to, 1) == 0){
70                      flow(from, s, 1);
71                      flow(t, to, 1);
72                      return true;
73                  }
74              }
75          }
76          return false;
```

```
    }
};
```

2.9 幾何ライブラリ

```
using real = double;
real EPS = 1e-10;
struct Point {
    real x, y;
    Point& operator+=(const Point a) { x += a.x; y += a.y;  return *this; }
    Point& operator-=(const Point a) { x -= a.x; y -= a.y;  return *this; }
    Point& operator*=(const real k) { x *= k; y *= k;  return *this; }
    Point& operator/=(const real k) { x /= k; y /= k;  return *this; }
    Point operator+(const Point a) const {return Point(*this) += a; }
    Point operator-(const Point a) const {return Point(*this) -= a; }
    Point operator*(const real k) const {return Point(*this) *= k; }
    Point operator/(const real k) const {return Point(*this) /= k; }
    bool operator<(const Point &a) const { return (x != a.x ? x < a.x : y < a.y); }
    explicit Point(real a = 0, real b = 0) : x(a), y(b) {};
};

istream& operator>> (istream& s, Point& P){
    s >> P.x >> P.y;
    return s;
}

inline real dot(Point a, Point b){ return a.x*b.x + a.y*b.y; }
inline real cross(Point a, Point b){ return a.x*b.y - a.y*b.x; }
inline real abs(Point a){ return sqrt(dot(a, a)); }

static constexpr int COUNTER_CLOCKWISE = 1;
static constexpr int CLOCKWISE = -1;
static constexpr int ONLINE_BACK = 2;
static constexpr int ONLINE_FRONT = -2;
static constexpr int ON_SEGMENT = 0;

int ccw(Point a, Point b, Point c){
    b -= a; c -= a;
    if(cross(b, c) > EPS) return COUNTER_CLOCKWISE;
    if(cross(b, c) < -EPS) return CLOCKWISE;
    if(dot(b, c) < 0) return ONLINE_BACK;
    if(abs(b) < abs(c)) return ONLINE_FRONT;
    return ON_SEGMENT;
}
struct Segment {
    Point a, b;
    Segment(Point x, Point y) : a(x), b(y) {};
};

bool intersect(Segment s, Segment t){
    return (ccw(s.a, s.b, t.a)*ccw(s.a, s.b, t.b) <= 0 &&
            ccw(t.a, t.b, s.a)*ccw(t.a, t.b, s.b) <= 0);
}

double distance(Segment s, Point c){
    if(dot(s.b-s.a, c-s.a) < EPS) return abs(c-s.a);
    if(dot(s.a-s.b, c-s.b) < EPS) return abs(c-s.b);
    return abs(cross(s.b-s.a, c-s.a)) / abs(s.a-s.b);
}

double distance(Segment s, Segment t){
    if(intersect(s, t)) return 0.0;
```

```
    return min({distance(s, t.a), distance(s, t.b),
                distance(t, s.a), distance(t, s.b)});
}
Point crossPoint(Segment s, Segment t){
    real d1 = abs(cross(s.b-s.a, t.b-t.a));
    real d2 = abs(cross(s.b-s.a, s.b-t.a));
    if(d1 < EPS && d2 < EPS) return t.a;
    return t.a+(t.b-t.a)*d2/d1;
}

Point project(Segment s, Point p){
    Point Q = s.b-s.a;
    return s.a + Q*(dot(p-s.a, Q) / dot(Q, Q));
}

Point refrect(Segment s, Point p){
    Point Q = project(s, p);
    return Q*2-p;
}

bool isOrthogonal(Segment s, Segment t){
    return fabs(dot(s.b-s.a, t.b-t.a)) < EPS;
}

bool isparallel(Segment s, Segment t){
    return fabs(cross(s.b-s.a, t.b-t.a)) < EPS;
}

using polygon = vector<Point>;

real area(polygon &v){
    if(v.size() < 3) return 0.0;
    real ans = 0.0;
    for (int i = 0; i+1 < v.size(); ++i) {
        ans += cross(v[i], v[i+1]);
    }
    ans += cross(v.back(), v.front());
    return ans/2;
}

polygon convex_hull(polygon v){
    int n = v.size();
    sort(v.begin(),v.end());
    int k = 0;
    polygon ret(n*2);
    for (int i = 0; i < n; ++i) {
        while(k > 1 && cross(ret[k-1]-ret[k-2], v[k]-ret[k-1]) < 0) k--;
        ret[k++] = v[i];
    }
    for(int i = n-2, t=k; i >= 0; i--){
        while(k > t && cross(ret[k-1]-ret[k-2], v[i]-ret[k-1]) < 0) k--;
        ret[k++] = v[i];
    }
    ret.resize(k-1);
    return ret;
}

bool isconvex(polygon &P){
    int n = P.size();
    for (int i = 0; i < n; ++i) {
        if(ccw(P[(i+n-1)%n], P[i], P[(i+1)%n]) == CLOCKWISE) return false;
    }
    return true;
```

```
}
```

2.10 木上の非再帰 dfs

```
deque<int> Q;
stack<int> s;
int cnt = 0;
vector<int> visited(n, 0), num(n);
s.emplace(0);
while(!s.empty()){
    int a = s.top(); s.pop();
    visited[a]++;
    num[a] = cnt++;
    Q.emplace_front(a);
    for (auto &&i : v[a]) {
        if(!visited[i]) s.emplace(i);
    }
}
```

3 まあまあ使いそうなライブラリ

3.1 階乗ライブラリ

```
template <class T>
T pow_ (T x, T n, T M){
    uint64_t u = 1, xx = x;
    while (n > 0){
        if (n&1) u = u * xx % M;
        xx = xx * xx % M;
        n >>= 1;
    }
    return static_cast<T>(u);
};

template <class T> class Factorial {
    T mod;
    vector<uint64_t> facts, factinv;

public:
    Factorial(int n, T mod) : facts(static_cast<u32>(n+1)),
    ↪  factinv(static_cast<u32>(n+1)), mod(mod) {
        facts[0] = 1;
        for (int i = 1; i < n+1; ++i) facts[i] = facts[i-1]*i % mod;
        factinv[n] = pow_(facts[n], static_cast<uint64_t>(mod - 2),
        ↪  static_cast<uint64_t>(mod));
        for (int i = n-1; i >= 0; --i) factinv[i] = factinv[i+1] * (i+1) % mod;
    }

    T fact(int k) const {
        if(k >= 0) return static_cast<T>(facts[k]);
        else return static_cast<T>(factinv[-k]);
    }

    T operator[](const int &k) const {
        if(k >= 0) return static_cast<T>(facts[k]);
        else return static_cast<T>(factinv[-k]);
    }

    T C(int p, int q) const {
        if(q < 0 || p < q) return 0;
        return static_cast<T>(facts[p]*  factinv[q] % mod * factinv[p-q] % mod);
```

```
    }

    T P(int p, int q) const {
        if(q < 0 || p < q) return 0;
        return static_cast<T>((facts[p] * factinv[p-q]) % mod);
    }

    T H(int p, int q) const {
        if(p < 0 || q < 0) return 0;
        return static_cast<T>(q == 0 ? 1 : C(p+q-1, q));
    }
};
```

3.2 Binary Indexed Tree (Fenwick Tree)

```
template<class T>
class BIT {
    vector<T> bit;
public:
    BIT(int n): bit(vector<T>(n+1, 0)){}

    T sum(int k){
        T ret = 0;
        for (++k; k > 0; k -= (k & -k)) ret += bit[k];
        return ret;
    }

    void add(int k, T x){
        for (++k; k < bit.size(); k  += (k & -k)) bit[k] += x;
    }
};
```

3.3 Segment Tree

```
template <class M>
struct SegmentTree{
    using T = typename M::T;
    int sz;
    vector<T> seg;
    explicit SegmentTree(int n) {
        sz = 1;
        while(sz < n) sz <<= 1;
        seg.assign(2*sz, M::e());
    }

    void set(int k, const T &x){ seg[k + sz] = x; }

    void build(){
        for (int i = sz-1; i > 0; --i) seg[i] = M::f(seg[2*i], seg[2*i+1]);
    }

    void update(int k, const T &x){
        k += sz;
        seg[k] = x;
        while (k >>= 1) seg[k] = M::f(seg[2*k], seg[2*k+1]);
    }

    T query(int a, int b){
        T l = M::e(), r = M::e();
        for(a += sz, b += sz; a < b; a >>=1, b>>=1){
            if(a & 1) l = M::f(l, seg[a++]);
```

```
            if(b & 1) r = M::f(seg[--b], r);
        }
        return M::f(l, r);
    }

    T operator[](const int &k) const { return seg[k + sz]; }
};


struct Monoid{
    using T = int;
    static T f(T a, T b) { return min(a, b); }
    static T e() { return INF<int>; }
};
```

3.4 遅延伝播 Segment Tree

```
template <class M>
struct LazySegmentTree{
    using T = typename M::T;
    using L = typename M::L;
    int sz, height{};
    vector<T> seg; vector<L> lazy;
    explicit LazySegmentTree(int n) {
        sz = 1; while(sz < n) sz <<= 1, height++;
        seg.assign(2*sz, M::e());
        lazy.assign(2*sz, M::l());
    }

    void set(int k, const T &x){
        seg[k + sz] = x;
    }

    void build(){
        for (int i = sz-1; i > 0; --i) seg[i] = M::f(seg[i<<1], seg[(i<<1)|1]);
    }

    T reflect(int k){ return lazy[k] == M::l() ? seg[k] : M::g(seg[k], lazy[k]); }

    void eval(int k){
        if(lazy[k] == M::l()) return;
        lazy[(k<<1)|0] = M::h(lazy[(k<<1)|0], lazy[k]);
        lazy[(k<<1)|1] = M::h(lazy[(k<<1)|1], lazy[k]);
        seg[k] = reflect(k);
        lazy[k] = M::l();
    }
    void thrust(int k){ for (int i = height; i; --i) eval(k>>i); }
    void recalc(int k) { while(k >>= 1) seg[k] = M::f(reflect((k<<1)|0),
    ↪ reflect((k<<1)|1));}
    void update(int a, int b, const L &x){
        thrust(a += sz); thrust(b += sz-1);
        for (int l = a, r = b+1;l < r; l >>=1, r >>= 1) {
            if(l&1) lazy[l] = M::h(lazy[l], x), l++;
            if(r&1) --r, lazy[r] = M::h(lazy[r], x);
        }
        recalc(a);
        recalc(b);
    }

    T query(int a, int b){ // [l, r)
        thrust(a += sz);
        thrust(b += sz-1);
```

```
        T ll = M::e(), rr = M::e();
        for(int l = a, r = b+1; l < r; l >>=1, r>>=1) {
            if (l & 1) ll = M::f(ll, reflect(l++));
            if (r & 1) rr = M::f(reflect(--r), rr);
        }
        return M::f(ll, rr);
    }
};

struct Monoid{
    using T = ll;
    using L = ll;
    static T f(T a, T b) { return min(a, b); }
    static T g(T a, L b) {
        if(b == e()) return a; else return b;
    }
    static L h(L a, L b) {
        if(b == e()) return a; else return b;
    }
    static T e() { return 0; }
    static L l() { return 0; }
};
```

3.5 modint(固定 MOD)

```
template<ll M = 1000000007>
struct modint{
    ll val;
    modint(): val(0){}
    template<typename T>
    explicit modint(T t){val = t%M; if(val < 0) val += M;}

    modint pow(ll k){
        modint res(1), x(val);
        while(k){
            if(k&1) res *= x;
            x *= x;
            k >>= 1;
        }
        return res;
    }
    template<typename T>
    modint& operator=(T a){ val = a%M; if(val < 0) val += M; return *this; }
    modint inv() {return pow(M-2);}
    modint& operator+=(modint a){ val += a.val; if(val >= M) val -= M; return *this;}
    modint& operator-=(modint a){ val += M-a.val; if(val >= M) val -= M; return
    ↪  *this;}
    modint& operator*=(modint a){ val = 1LL*val*a.val%M; return *this;}
    modint& operator/=(modint a){ return (*this) *= a.inv();}
    modint operator+(modint a) const {return modint(val) +=a;}
    modint operator-(modint a) const {return modint(val) -=a;}
    modint operator*(modint a) const {return modint(val) *=a;}
    modint operator/(modint a) const {return modint(val) /=a;}
    modint operator-(){ return modint(-val);}
    bool operator==(const modint a) const {return val == a.val;}
    bool operator!=(const modint a) const {return val != a.val;}
    bool operator<(const modint a) const {return val < a.val;}
};

using mint = modint<MOD>;
```

3.6 行列

```
template<class T>
struct matrix {
    vector<vector<T>> A;
    matrix() = default;
    matrix(size_t n, size_t m) : A(n, vector<T>(m)) {}
    explicit matrix(size_t n) : A(n, vector<T> (n)) {};
    size_t height() const { return (A.size()); }
    size_t width() const { return (A[0].size()); }

    const vector<T> &operator [] (int k) const { return A[k]; }
    vector<T> &operator[] (int k) { return A[k]; }

    static matrix I(size_t n){
        matrix mat(n);
        for (int i = 0; i < n; ++i) mat[i][i] = 1;
        return mat;
    }

    matrix &operator+= (const matrix &B){
        size_t h = height(), w = width();
        for (int i = 0; i < h; ++i) {
            for (int j = 0; j < w; ++j) {
                (*this)[i][j] += B[i][j];
            }
        }
    }

    matrix &operator-= (const matrix &B){
        size_t h = height(), w = width();
        for (int i = 0; i < h; ++i) {
            for (int j = 0; j < w; ++j) {
                (*this)[i][j] -= B[i][j];
            }
        }
    }

    matrix &operator*=(const matrix &B)
    {
        size_t n = height(), m = B.width(), p = width();
        matrix C (n, m);
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                for (int k = 0; k < p; ++k) {
                    C[i][j] = (C[i][j] + (*this)[i][k] * B[k][j]);
                }
            }
        }
        A.swap(C.A);
        return (*this);
    }

    template <class U>
    matrix &operator%= (const U &m){
        for (int i = 0; i < height(); ++i) {
            for (int j = 0; j < width(); ++j) {
                (*this)[i][j] %= m;
            }
        }
    }

    matrix pow(ll n) const {
```

```
        matrix a = (*this), res = I(height());
        while(n > 0){
            if(n & 1) res *= a;
            a *= a;
            n >>= 1;
        }
        return res;
    }
    matrix operator+(const matrix &A) const {return matrix(*this) += A;}
    matrix operator-(const matrix &A) const {return matrix(*this) -= A;}
    matrix operator*(const matrix &A) const {return matrix(*this) *= A;}
    template <class U>
    matrix operator%(const U &m) const {return matrix(*this) %= m;}

};
```

4 あまり使わなさそうなライブラリ

4.1 中国剰余定理

```
template <class T>
T pow_ (T x, T n, T M){
    uint64_t u = 1, xx = x;
    while (n > 0){
        if (n&1) u = u * xx % M;
        xx = xx * xx % M;
        n >>= 1;
    }
    return static_cast<T>(u);
};


template<typename T>
T extgcd(T a, T b, T &x ,T &y){
    for (T u = y = 1, v = x = 0; a; ) {
        ll q = b/a;
        swap(x -= q*u, u);
        swap(y -= q*v, v);
        swap(b -= q*a, a);
    }
    return b;
}

template<typename T>
T mod_inv(T x, T m){
    T s, t;
    extgcd(x, m, s, t);
    return (m+s)% m;
}

pair<ll, ll> CRT(const vector<pair<ll, ll>> &a){
    ll r = 0, M = 1;
    for (int i = 0; i < a.size(); ++i) {
        ll p, q;
        ll d = extgcd(M, a[i].second, p, q);
        if((a[i].first - r )%d != 0) return make_pair(0, -1);
        ll tmp = (a[i].first - r) / d * p % (a[i].second / d);
        r += M * tmp;
        M *= a[i].second/d;
    }
    return make_pair((r+M) % M, M);
}
```

4.2 拡張ユークリッドの互除法

```
template<typename T>
T extgcd(T a, T b, T &x ,T &y){
    for (T u = y = 1, v = x = 0; a; ) {
        ll q = b/a;
        swap(x -= q*u, u);
        swap(y -= q*v, v);
        swap(b -= q*a, a);
    }
    return b;
}

```

4.3 ベルマンフォード法

```
template <typename T>
struct edge {
    int from, to;
    T cost;

    edge(int to, T cost) : from(-1), to(to), cost(cost) {}
    edge(int from, int to, T cost) : from(from), to(to), cost(cost) {}

    explicit operator int() const {return to;}
};

template <typename T>
vector<T> bellman_ford(int s, int V,vector<edge<T> > &G){
    const T INF = numeric_limits<T>::max();
    vector<T> d(V, INF);
    d[s] = 0;
    for (int i = 0; i < V - 1; ++i) {
        for (auto &&e : G) {
            if (d[e.from] == INF) continue;
            d[e.to] = min(d[e.to], d[e.from] + e.cost);
        }
    }
    for (auto &&e : G) {
        if(d[e.from] == INF) continue;
        if(d[e.from] + e.cost < d[e.to]) return vector<T> ();
    }
    return d;
}
```

4.4 二部グラフの最大マッチング

```
class Bipartite_Matching {
    vector<vector<int>> G;
    vector<int> match, used, alive;
    int t;
public:
    explicit Bipartite_Matching(int n): t(0), G(n), match(n, -1), used(n, 0),
    ↪  alive(n, -1){};

    void connect(int a, int b){
        G[a].emplace_back(b);
        G[b].emplace_back(a);
    }

    bool dfs(int x){
        used[x] = t;
```

```
        for (auto &&i : G[x]) {
            int w = match[i];
            if(alive[i] == 0) continue;
            if(w == -1 || (used[w] != t && dfs(w))){
                match[x] = i;
                match[i] = x;
                return true;
            }
        }
        return false;
    }

    int matching() {
        int ans = 0;
        for (int i = 0; i < G.size(); ++i) {
            if(alive[i] == 0) continue;
            if(match[i] == -1) {
                ++t;
                ans += dfs(i);
            }
        }
        return ans;
    }
};
```

4.5 最大独立集合

```
class IndependentSet {
    int n;
    vector<vector<int>> G;
    void dfs(int x, vector<bool> &visited, vector<bool> &gcan, vector<bool> &alive){
        stack<int> s;
        s.emplace(x);
        while(!s.empty()){
            int y = s.top();
            visited[y] = true;
            gcan[y] = true;
            s.pop();
            for (auto &&i : G[y]) {
                if(!visited[i] && alive[i]) s.emplace(i);
            }
        }
    }

    int ConnectedCase(vector<bool> can) {
        int pMax = -1, pMin = -1, Max = -1, Min = n+1, num = 0;
        for (int i = 0; i < n; ++i) {
            if(!can[i]) continue;
            ++num;
            int tnum = 0;
            for (auto &&j : G[i]) if(can[j]) ++tnum;
            if(Max < tnum) Max = tnum, pMax = i;
            if(Min > tnum) Min = tnum, pMin = i;
        }
        if(num == 1) return 1;
        if(Max <= 2){
            if(Min == 1) return (num+1)/2;
            else return num/2;
        }
        int ans = 0;
        vector<bool> ncan = can;
        if(Min < 2){
```

```
            ncan[pMin] = false;
            for (auto &&i : G[pMin]) ncan[i] = false;
            ans = max(ans, GeneralCase(ncan) + 1);
        }else {
            ncan[pMax] = false;
            for (auto &&i : G[pMax]) ncan[i] = false;
            int temp = GeneralCase(ncan);
            ans = max(ans, temp+1);
            ncan = can;
            ncan[pMax] = false;
            ans = max(ans, GeneralCase(ncan));
        }
        return ans;
    }

    int GeneralCase(vector<bool> alive) {
        if(n <= 1) return n;
        vector<bool> visited(n, 0);
        int res = 0;
        for (int i = 0; i < n; ++i) {
            if(!visited[i] && alive[i]){
                vector<bool> gcan(n, false);
                dfs(i, visited, gcan, alive);
                res += ConnectedCase(gcan);
            }
        }
        return res;
    }
public:
    explicit IndependentSet(int n): n(n), G(n) {}
    void add_edge(int u, int v){
        G[u].emplace_back(v);
        G[v].emplace_back(u);
    }
    int stable_set() {
        vector<bool> alive(n, true);
        return GeneralCase(alive);
    }
};
```

4.6 最近共通祖先 (LCA)

```
template <class M>
struct SegmentTree{
    using T = typename M::T;
    int sz;
    vector<T> seg;
    explicit SegmentTree(int n) {
        sz = 1;
        while(sz < n) sz <<= 1;
        seg.assign(2*sz, M::e());
    }

    void set(int k, const T &x){ seg[k + sz] = x; }

    void build(){
        for (int i = sz-1; i > 0; --i) seg[i] = M::f(seg[2*i], seg[2*i+1]);
    }

    void update(int k, const T &x){
        k += sz;
        seg[k] = x;
```

```
        while (k >>= 1) seg[k] = M::f(seg[2*k], seg[2*k+1]);
    }

    T query(int a, int b){
        T l = M::e(), r = M::e();
        for(a += sz, b += sz; a < b; a >>=1, b>>=1){
            if(a & 1) l = M::f(l, seg[a++]);
            if(b & 1) r = M::f(seg[--b], r);
        }
        return M::f(l, r);
    }

    T operator[](const int &k) const { return seg[k + sz]; }
};

struct Monoid{
    using T = pair<int, int>;
    static T f(T a, T b) { return min(a, b); }
    static T e() { return T(INF<int>, -1); }
};

class Graph {
    void dfs_euler(int v, int p, int d, int &k){
        id[v] = k;
        vs[k] = v;
        depth[k++] = d;
        for (auto &&u : G[v]) {
            if(u != p){
                dfs_euler(u, v, d+1, k);
                vs[k] = v;
                depth[k++] = d;
            }
        }
    }
public:
    int n;
    vector<vector<int>> G;
    vector<int> vs, depth, id;
    explicit Graph(int n) : n(n), G(n), vs(2*n-1), depth(2*n-1), id(n) {};
    void add_edge(int a, int b){
        G[a].emplace_back(b);
        G[b].emplace_back(a);
    }

    void eulertour(int root) {
        int k = 0;
        dfs_euler(root, -1, 0, k);
    }
};

class LCA {
    Graph G;
    SegmentTree<Monoid> seg;
public:
    explicit LCA(Graph G) : G(G), seg(2*G.n-1) {
        int n = G.n;
        for (int i = 0; i < 2*n-1; ++i) {
            seg.set(i, pair<int, int>(G.depth[i], i));
        }
        seg.build();
    };

    int lca(int u, int v){
```

```
        if(G.id[u] > G.id[v]) swap(u, v);
        return seg.query(G.id[u], G.id[v]+1).second;
    }
};
```

4.7 Convex-Hull-Trick

```
template<class T>
class monotonic_CHT {
    using P = pair<T, T>;
    vector<P> lines;
public:
    monotonic_CHT() {}

    bool check(P l1, P l2, P l3){
        if(l1 < l3) swap(l1, l3);
        return (l3.second - l2.second)*(l2.first-l1.first)
            >= (l2.second - l1.second)*(l3.first-l2.first);
    }
    void add(T a, T b){
        P line(a, b);
        while(lines.size() >= 2 && check(*(lines.end()-2), lines.back(), line))
        ↪  lines.pop_back();
        lines.emplace_back(line);
    }

    T f(int i, T x){
        return lines[i].first * x + lines[i].second;
    }

    T f(P line, T x){
        return line.first * x + line.second;
    }

    T get(T x){
        static int head = 0;
        while(lines.size()-head >= 2 && f(head, x) >= f(head+1, x)) ++head;
        return f(head, x);
    }
};
```

4.8 スライド最小値

```
template<class T, class F>
class sliding_window {
    vector<T> v;
    deque<T> Q;
    F f;
public:
    int l, r;
    explicit sliding_window(vector<T> &v, F f) : v(v), f(f), l(0), r(0) {};
    void set(vector<T> &u){
        v = u;
        Q.clear();
        l = 0; r = 0;
    }
    void reset(){
        Q.clear();
        l = 0, r = 0;
    }
    void slideL(){
```

```
        if(Q.front() == l++) Q.pop_front();
    }
    void slideR(){
        while(!Q.empty() && !f(v[Q.back()], v[r])) Q.pop_back();
        Q.push_back(r++);
    }
    T get_index()  {
        if(l == r) return 0;
        return Q.front();
    }
    T value()  {
        if(l == r) return 0;
        return v[Q.front()];
    }
};
```

4.9 ポテンシャル付き Union-Find

```
template <class T>
class WeightedUnionFind {
    vector<int> uni;
    vector<T> weights;
    int n;
public:
    explicit WeightedUnionFind(int n, T SUM_UNITY = 0) :
    uni(static_cast<u32>(n), -1) , n(n), weights(n,  SUM_UNITY){};

    int root(int a) {
        if (uni[a] < 0) return a;
        else {
            int r = root(uni[a]);
            weights[a] += weights[uni[a]];
            return (uni[a] = r);
        }
    }

    bool unite(int a, int b, T w) {
        w += weight(a); w -= weight(b);
        a = root(a);
        b = root(b);
        if(a == b) return false;
        if(uni[a] > uni[b]) swap(a, b), w = -w;
        uni[a] += uni[b];
        uni[b] = a;
        weights[b] = w;
        return true;
    }

    int size(int a){
        return -uni[root(a)];
    }

    T weight(T a){
        root(a);
        return weights[a];
    }
    int diff(int x, int y){
        return weight(y) - weight(x);
    }
};
```

4.10 modint(実行時 MOD)

```
struct modint {
    static ll &mod(){
        static ll mod_ = 0;
        return mod_;
    }

    static void set_mod(const ll x) { mod() = x; }
    static ll M() {return mod(); }

    ll val;
    modint(): val(0){}
    template<typename T>
    explicit modint(T t){val = t%M(); if(val < 0) val += M();}

    modint pow(ll k){
        modint res(1), x(val);
        while(k){
            if(k&1) res *= x;
            x *= x;
            k >>= 1;
        }
        return res;
    }
    template<typename T>
    modint& operator=(T a){ val = a%M(); if(val < 0) val += M(); return *this; }
    modint inv() {return pow(M()-2);}
    modint& operator+=(modint a){ val += a.val; if(val >= M()) val -= M(); return
     ↪  *this;}
    modint& operator-=(modint a){ val += M()-a.val; if(val >= M()) val -= M(); return
     ↪  *this;}
    modint& operator*=(modint a){ val = val*a.val%M(); return *this;}
    modint& operator/=(modint a){ return (*this) *= a.inv();}
    modint operator+(modint a) const {return modint(val) +=a;}
    modint operator-(modint a) const {return modint(val) -=a;}
    modint operator*(modint a) const {return modint(val) *=a;}
    modint operator/(modint a) const {return modint(val) /=a;}
    modint operator-(){return modint(-val); }
    bool operator==(const modint a) const {return val == a.val;}
    bool operator!=(const modint a) const {return val != a.val;}
    bool operator<(const modint a) const {return val < a.val;}
};
```

4.11 Xor-Shift

```
#include <chrono>
class xor_shift {
    uint32_t x, y, z, w;
public:
    xor_shift() :
     ↪  x(static_cast<uint32_t>((chrono::system_clock::now().time_since_epoch().count())&((1l
     ↪  << 32)-1))),
    y(1068246329), z(321908594), w(1234567890) {};

    uint32_t urand(){
        uint32_t t;
        t = x ^ (x << 11);
        x = y; y = z; z = w;
        w = (w ^ (w >> 19)) ^ (t ^ (t >> 8));
        return w;
    };

```

```
16      int rand(int n){
17          if(n < 0) return -rand(-n);
18          uint32_t t = numeric_limits<uint32_t>::max()/(n+1)*(n+1);
19          uint32_t e = urand();
20          while(e >= t) e = urand();
21          return static_cast<int>(e%(n+1));
22      }
23
24      int rand(int a, int b){
25          if(a > b) swap(a, b);
26          return a+rand(b-a);
27      }
28  };
```