

# 目次

<b>1</b>	<b>備忘録</b>	<b>2</b>
1.1	テンプレート	2
1.2	makev chmax chmin	2
1.3	ファイルを作成する	2
<b>2</b>	<b>使いそうなライブラリ</b>	<b>3</b>
2.1	繰り返し二乗法	3
2.2	約数列挙	3
2.3	素数列挙・素因数分解	3
2.4	ダイクストラ法	4
2.5	kruskal 法 (最小全域木)	4
2.6	Union-Find	5
2.7	正方行列	5
2.8	Dinic 法 (最大流)	6
2.9	幾何ライブラリ	8
2.10	木上の非再帰 dfs	10
<b>3</b>	<b>まあまあ使いそうなライブラリ</b>	<b>10</b>
3.1	階乗ライブラリ	10
3.2	Binary Indexed Tree (Fenwick Tree)	11
3.3	Segment Tree	11
3.4	遅延伝播 Segment Tree	12
3.5	modint(固定 MOD)	13
3.6	行列	14
<b>4</b>	<b>あまり使わなさそうなライブラリ</b>	<b>15</b>
4.1	中国剰余定理	15
4.2	拡張ユークリッドの互除法	16
4.3	ベルマンフォード法	16
4.4	二部グラフの最大マッチング	16
4.5	最大独立集合	17
4.6	最近共通祖先 (LCA)	18
4.7	Convex-Hull-Trick	20
4.8	スライド最小値	20
4.9	ポテンシャル付き Union-Find	21
4.10	modint(実行時 MOD)	22
4.11	Xor-Shift	22

# 1 備忘録

## 1.1 テンプレート

```
1  #include <iostream>
2  #include <algorithm>
3  #include <iomanip>
4  #include <map>
5  #include <set>
6  #include <queue>
7  #include <stack>
8  #include <numeric>
9  #include <bitset>
10 #include <cmath>
11
12 static const int MOD = 1000000007;
13 using ll = long long;
14 using u32 = uint32_t;
15 using namespace std;
16
17 template<class T> constexpr T INF = ::numeric_limits<T>::max()/32*15+208;
18
19 int main() {
20
21     return 0;
22 }
```

## 1.2 makev chmax chmin

```
1  template <class T, class U>
2  vector<T> make_v(U size, const T& init){ return vector<T>(static_cast<size_t>(size),
   →  init); }
3
4  template<class... Ts, class U>
5  auto make_v(U size, Ts... rest) { return
   →  vector<decltype(make_v(rest...))>(static_cast<size_t>(size), make_v(rest...)); }
6
7  template<class T> void chmin(T &a, const T &b){ a = (a < b ? a : b); }
8  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。

```
1  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

```
1  template <class T>
2  T pow_ (T x, T n, T M){
3      uint64_t u = 1, xx = x;
4      while (n > 0){
5          if (n&1) u = u * xx % M;
6          xx = xx * xx % M;
7          n >>= 1;
8      }
9      return static_cast<T>(u);
10 };
```

### 2.2 約数列挙

```
1  template<class T>
2  vector<T> divisor(T n){
3      vector<T> ret;
4      for(T i = 1; i * i <= n; i++) {
5          if(n % i == 0) {
6              ret.push_back(i);
7              if(i * i != n) ret.push_back(n / i);
8          }
9      }
10     sort(begin(ret), end(ret));
11     return(ret);
12 }
```

### 2.3 素数列挙・素因数分解

```
1  vector<int> get_prime(int n){
2      if(n <= 1) return vector<int>();
3      vector<bool> is_prime(n+1, true);
4      vector<int> prime;
5      is_prime[0] = is_prime[1] = 0;
6      for (int i = 2; i <= n; ++i) {
7          if(is_prime[i]) prime.emplace_back(i);
8          for (auto &&j : prime){
9              if(i*j > n) break;
10             is_prime[i*j] = false;
11             if(i % j == 0) break;
12         }
13     }
14     return prime;
15 }
16 const auto primes = get_prime(65535);
17
18 template<class T>
19 vector<T> prime_factor(T n){
20     vector<T> res;
21     for (auto &&i : primes) {
22         while (n % i == 0){
23             res.emplace_back(i);
24             n /= i;
25         }
26     }
27     if(n != 1) res.emplace_back(n);
28     return res;
29 }
```

## 2.4 ダイクストラ法

```
1  template <typename T>
2  struct edge {
3      int from, to; T cost;
4      edge(int to, T cost) : from(-1), to(to), cost(cost) {}
5      edge(int from, int to, T cost) : from(from), to(to), cost(cost) {}
6  };
7
8  template <typename T>
9  vector<T> dijkstra(int s, vector<vector<edge<T>>> &G){
10     size_t n=G.size();
11     vector<T> d(n, INF<T>);
12     priority_queue<pair<T, int>, vector<pair<T, int>>, greater<>> Q;
13     d[s]=0;
14     Q.emplace(0, s);
15     while(!Q.empty()){
16         T cost; int i;
17         tie(cost, i) = Q.top(); Q.pop();
18         if(d[i] < cost) continue;
19         for (auto &&e : G[i]) {
20             auto cost2 = cost + e.cost;
21             if(d[e.to] <= cost2) continue;
22             d[e.to] = cost2;
23             Q.emplace(d[e.to], e.to);
24         }
25     }
26     return d;
27 }
```

## 2.5 kruskal 法 (最小全域木)

```
1  template <typename T>
2  struct edge {
3      int from, to;
4      T cost;
5
6      edge(int to, T cost) : from(-1), to(to), cost(cost) {}
7      edge(int from, int to, T cost) : from(from), to(to), cost(cost) {}
8
9      explicit operator int() const {return to;}
10 };
11
12 class UnionFind {
13     vector<int> uni;
14     int n;
15 public:
16     explicit UnionFind(int n) : uni(static_cast<u32>(n), -1) , n(n){};
17
18     int root(int a){
19         if (uni[a] < 0) return a;
20         else return (uni[a] = root(uni[a]));
21     }
22
23     bool unite(int a, int b) {
24         a = root(a);
25         b = root(b);
26         if(a == b) return false;
27         if(uni[a] > uni[b]) swap(a, b);
28         uni[a] += uni[b];
29         uni[b] = a;
30         return true;
31     }
```

```

32 };
33
34 template< typename T >
35 T kruskal(vector<edge<T>> &G, int V)
36 {
37     sort(begin(G), end(G), [](const edge< T > &a, const edge< T > &b) { return
38         ↪ (a.cost < b.cost); });
39     UnionFind tree(V);
40     T ret = 0;
41     for(auto &e : G) {
42         if(tree.unite(e.from, e.to)) ret += e.cost;
43     }
44     return (ret);
45 }

```

## 2.6 Union-Find

```

1 class UnionFind {
2     int n;
3     vector<int> uni;
4 public:
5     explicit UnionFind(int n) : uni(static_cast<u32>(n), -1) , n(n){};
6
7     int root(int a){
8         if (uni[a] < 0) return a;
9         else return (uni[a] = root(uni[a]));
10    }
11
12    bool unite(int a, int b) {
13        a = root(a);
14        b = root(b);
15        if(a == b) return false;
16        if(uni[a] > uni[b]) swap(a, b);
17        uni[a] += uni[b];
18        uni[b] = a;
19        return true;
20    }
21
22    int size(int i){ return -uni[root(i)]; }
23    bool same(int a, int b) { return root(a) == root(b); }
24 };

```

## 2.7 正方形行列

```

1 template<class T, size_t SIZE>
2 struct SquareMatrix {
3     using ar = array<T, SIZE>;
4     using mat = array<ar, SIZE>;
5     mat A;
6     SquareMatrix() = default;
7     static SquareMatrix I(T e){
8         SquareMatrix X;
9         for (int i = 0; i < SIZE; ++i) {
10             X[i][i] = e;
11         }
12         return X;
13     }
14
15     inline const ar &operator[](int k) const{ return (A.at(k)); }
16     inline ar &operator[](int k) { return (A.at(k)); }
17     SquareMatrix &operator+= (const SquareMatrix &B){

```

```

18     for (int i = 0; i < SIZE; ++i) {
19         for (int j = 0; j < SIZE; ++j) {
20             (*this)[i][j] += B[i][j];
21         }
22     }
23     return (*this);
24 }
25
26 SquareMatrix &operator-= (const SquareMatrix &B) {
27     for (int i = 0; i < SIZE; ++i) {
28         for (int j = 0; j < SIZE; ++j) {
29             (*this)[i][j] -= B[i][j];
30         }
31     }
32     return (*this);
33 }
34
35 SquareMatrix &operator*=(const SquareMatrix &B) {
36     SquareMatrix C;
37     for (int i = 0; i < SIZE; ++i) {
38         for (int j = 0; j < SIZE; ++j) {
39             for (int k = 0; k < SIZE; ++k) {
40                 C[i][j] += ((*this)[i][k] * B[k][j]);
41             }
42         }
43     }
44     A.swap(C.A);
45     return (*this);
46 }
47
48 SquareMatrix pow(ll n) const {
49     SquareMatrix a = (*this), res = I(T(1));
50     while(n > 0){
51         if(n & 1) res *= a;
52         a *= a;
53         n >>= 1;
54     }
55     return res;
56 }
57 SquareMatrix operator+(const SquareMatrix &B) const {return SquareMatrix(*this)
    ↪ += B;}
58 SquareMatrix operator-(const SquareMatrix &B) const {return SquareMatrix(*this)
    ↪ -= B;}
59 SquareMatrix operator*(const SquareMatrix &B) const {return SquareMatrix(*this)
    ↪ *= B;}
60 };

```

## 2.8 Dinic 法 (最大流)

```

1  template<class T>
2  class Dinic {
3      struct edge {
4          int to{}, rev{};
5          T cap;
6          edge() = default;
7          edge(int to, T cap, int rev):to(to), cap(cap), rev(rev) {}
8      };
9      int n{};
10     vector<vector<edge>> G;
11     vector<int> level, iter;
12 public:
13     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;

```

```

77     }
78 };

```

## 2.9 幾何ライブラリ

```

1  using real = double;
2  real EPS = 1e-10;
3  struct Point {
4      real x, y;
5      Point& operator+=(const Point a) { x += a.x; y += a.y; return *this; }
6      Point& operator-=(const Point a) { x -= a.x; y -= a.y; return *this; }
7      Point& operator*=(const real k) { x *= k; y *= k; return *this; }
8      Point& operator/=(const real k) { x /= k; y /= k; return *this; }
9      Point operator+(const Point a) const {return Point(*this) += a; }
10     Point operator-(const Point a) const {return Point(*this) -= a; }
11     Point operator*(const real k) const {return Point(*this) *= k; }
12     Point operator/(const real k) const {return Point(*this) /= k; }
13     bool operator<(const Point &a) const { return (x != a.x ? x < a.x : y < a.y); }
14     explicit Point(real a = 0, real b = 0) : x(a), y(b) {};
15 };
16
17 istream& operator>> (istream& s, Point& P){
18     s >> P.x >> P.y;
19     return s;
20 }
21
22 inline real dot(Point a, Point b){ return a.x*b.x + a.y*b.y; }
23 inline real cross(Point a, Point b){ return a.x*b.y - a.y*b.x; }
24 inline real abs(Point a){ return sqrt(dot(a, a)); }
25
26 static constexpr int COUNTER_CLOCKWISE = 1;
27 static constexpr int CLOCKWISE = -1;
28 static constexpr int ONLINE_BACK = 2;
29 static constexpr int ONLINE_FRONT = -2;
30 static constexpr int ON_SEGMENT = 0;
31
32 int ccw(Point a, Point b, Point c){
33     b -= a; c -= a;
34     if(cross(b, c) > EPS) return COUNTER_CLOCKWISE;
35     if(cross(b, c) < -EPS) return CLOCKWISE;
36     if(dot(b, c) < 0) return ONLINE_BACK;
37     if(abs(b) < abs(c)) return ONLINE_FRONT;
38     return ON_SEGMENT;
39 }
40 struct Segment {
41     Point a, b;
42     Segment(Point x, Point y) : a(x), b(y) {};
43 };
44
45 bool intersect(Segment s, Segment t){
46     return (ccw(s.a, s.b, t.a)*ccw(s.a, s.b, t.b) <= 0 &&
47             ccw(t.a, t.b, s.a)*ccw(t.a, t.b, s.b) <= 0);
48 }
49
50 double distance(Segment s, Point c){
51     if(dot(s.b-s.a, c-s.a) < EPS) return abs(c-s.a);
52     if(dot(s.a-s.b, c-s.b) < EPS) return abs(c-s.b);
53     return abs(cross(s.b-s.a, c-s.a)) / abs(s.a-s.b);
54 }
55
56 double distance(Segment s, Segment t){
57     if(intersect(s, t)) return 0.0;

```



```

58     return min({distance(s, t.a), distance(s, t.b),
59                distance(t, s.a), distance(t, s.b)});
60 }
61 Point crossPoint(Segment s, Segment t){
62     real d1 = abs(cross(s.b-s.a, t.b-t.a));
63     real d2 = abs(cross(s.b-s.a, s.b-t.a));
64     if(d1 < EPS && d2 < EPS) return t.a;
65     return t.a+(t.b-t.a)*d2/d1;
66 }
67
68 Point project(Segment s, Point p){
69     Point Q = s.b-s.a;
70     return s.a + Q*(dot(p-s.a, Q) / dot(Q, Q));
71 }
72
73 Point refrect(Segment s, Point p){
74     Point Q = project(s, p);
75     return Q*2-p;
76 }
77
78 bool isOrthogonal(Segment s, Segment t){
79     return fabs(dot(s.b-s.a, t.b-t.a)) < EPS;
80 }
81
82 bool isparallel(Segment s, Segment t){
83     return fabs(cross(s.b-s.a, t.b-t.a)) < EPS;
84 }
85
86 using polygon = vector<Point>;
87
88 real area(polygon &v){
89     if(v.size() < 3) return 0.0;
90     real ans = 0.0;
91     for (int i = 0; i+1 < v.size(); ++i) {
92         ans += cross(v[i], v[i+1]);
93     }
94     ans += cross(v.back(), v.front());
95     return ans/2;
96 }
97
98 polygon convex_hull(polygon v){
99     int n = v.size();
100    sort(v.begin(), v.end());
101    int k = 0;
102    polygon ret(n*2);
103    for (int i = 0; i < n; ++i) {
104        while(k > 1 && cross(ret[k-1]-ret[k-2], v[i]-ret[k-1]) < 0) k--;
105        ret[k++] = v[i];
106    }
107    for(int i = n-2, t=k; i >= 0; i--){
108        while(k > t && cross(ret[k-1]-ret[k-2], v[i]-ret[k-1]) < 0) k--;
109        ret[k++] = v[i];
110    }
111    ret.resize(k-1);
112    return ret;
113 }
114
115 bool isconvex(polygon &P){
116     int n = P.size();
117     for (int i = 0; i < n; ++i) {
118         if(ccw(P[(i+n-1)%n], P[i], P[(i+1)%n]) == CLOCKWISE) return false;
119     }
120     return true;

```

```
121 }
```

## 2.10 木上の非再帰 dfs

```
1 deque<int> Q;
2 stack<int> s;
3 int cnt = 0;
4 vector<int> visited(n, 0), num(n);
5 s.emplace(0);
6 while(!s.empty()){
7     int a = s.top(); s.pop();
8     visited[a]++;
9     num[a] = cnt++;
10    Q.emplace_front(a);
11    for (auto &&i : v[a]) {
12        if(!visited[i]) s.emplace(i);
13    }
14 }
```

## 3 まあまあ使いそうなライブラリ

### 3.1 階乗ライブラリ

```
1 template <class T>
2 T pow_ (T x, T n, T M){
3     uint64_t u = 1, xx = x;
4     while (n > 0){
5         if (n&1) u = u * xx % M;
6         xx = xx * xx % M;
7         n >>= 1;
8     }
9     return static_cast<T>(u);
10 };
11
12 template <class T> class Factorial {
13     T mod;
14     vector<uint64_t> facts, factinv;
15
16 public:
17     Factorial(int n, T mod) : facts(static_cast<u32>(n+1)),
18         ↪ factinv(static_cast<u32>(n+1)), mod(mod) {
19         facts[0] = 1;
20         for (int i = 1; i < n+1; ++i) facts[i] = facts[i-1]*i % mod;
21         factinv[n] = pow_(facts[n], static_cast<uint64_t>(mod - 2),
22             ↪ static_cast<uint64_t>(mod));
23         for (int i = n-1; i >= 0; --i) factinv[i] = factinv[i+1] * (i+1) % mod;
24     }
25
26     T fact(int k) const {
27         if(k >= 0) return static_cast<T>(facts[k]);
28         else return static_cast<T>(factinv[-k]);
29     }
30
31     T operator[] (const int &k) const {
32         if(k >= 0) return static_cast<T>(facts[k]);
33         else return static_cast<T>(factinv[-k]);
34     }
35
36     T C(int p, int q) const {
37         if(q < 0 || p < q) return 0;
38         return static_cast<T>(facts[p]* factinv[q] % mod * factinv[p-q] % mod);
39     }
40 }
```

```

37     }
38
39     T P(int p, int q) const {
40         if(q < 0 || p < q) return 0;
41         return static_cast<T>((facts[p] * factinv[p-q]) % mod);
42     }
43
44     T H(int p, int q) const {
45         if(p < 0 || q < 0) return 0;
46         return static_cast<T>(q == 0 ? 1 : C(p+q-1, q));
47     }
48 };

```

### 3.2 Binary Indexed Tree (Fenwick Tree)

```

1  template<class T>
2  class BIT {
3      vector<T> bit;
4  public:
5      BIT(int n): bit(vector<T>(n+1, 0)){}
6
7      T sum(int k){
8          T ret = 0;
9          for (++k; k > 0; k -= (k & -k)) ret += bit[k];
10         return ret;
11     }
12
13     void add(int k, T x){
14         for (++k; k < bit.size(); k += (k & -k)) bit[k] += x;
15     }
16 };

```

### 3.3 Segment Tree

```

1  template <class M>
2  struct SegmentTree{
3      using T = typename M::T;
4      int sz;
5      vector<T> seg;
6      explicit SegmentTree(int n) {
7          sz = 1;
8          while(sz < n) sz <<= 1;
9          seg.assign(2*sz, M::e());
10     }
11
12     void set(int k, const T &x){ seg[k + sz] = x; }
13
14     void build(){
15         for (int i = sz-1; i > 0; --i) seg[i] = M::f(seg[2*i], seg[2*i+1]);
16     }
17
18     void update(int k, const T &x){
19         k += sz;
20         seg[k] = x;
21         while (k >= 1) seg[k] = M::f(seg[2*k], seg[2*k+1]);
22     }
23
24     T query(int a, int b){
25         T l = M::e(), r = M::e();
26         for(a += sz, b += sz; a < b; a >>= 1, b >>= 1){
27             if(a & 1) l = M::f(l, seg[a++]);

```

```

28         if(b & 1) r = M::f(seg[--b], r);
29     }
30     return M::f(l, r);
31 }
32
33 T operator[](const int &k) const { return seg[k + sz]; }
34 };
35
36
37 struct Monoid{
38     using T = int;
39     static T f(T a, T b) { return min(a, b); }
40     static T e() { return INF<int>; }
41 };

```

### 3.4 遅延伝播 Segment Tree

```

1  template <class M>
2  struct LazySegmentTree{
3      using T = typename M::T;
4      using L = typename M::L;
5      int sz, height{};
6      vector<T> seg; vector<L> lazy;
7      explicit LazySegmentTree(int n) {
8          sz = 1; while(sz < n) sz <<= 1, height++;
9          seg.assign(2*sz, M::e());
10         lazy.assign(2*sz, M::l());
11     }
12
13     void set(int k, const T &x){
14         seg[k + sz] = x;
15     }
16
17     void build(){
18         for (int i = sz-1; i > 0; --i) seg[i] = M::f(seg[i<<1], seg[(i<<1)|1]);
19     }
20
21     T reflect(int k){ return lazy[k] == M::l() ? seg[k] : M::g(seg[k], lazy[k]); }
22
23     void eval(int k){
24         if(lazy[k] == M::l()) return;
25         lazy[(k<<1)|0] = M::h(lazy[(k<<1)|0], lazy[k]);
26         lazy[(k<<1)|1] = M::h(lazy[(k<<1)|1], lazy[k]);
27         seg[k] = reflect(k);
28         lazy[k] = M::l();
29     }
30     void thrust(int k){ for (int i = height; i; --i) eval(k>>i); }
31     void recalc(int k) { while(k >= 1) seg[k] = M::f(reflect((k<<1)|0),
32         ↪ reflect((k<<1)|1)); }
33     void update(int a, int b, const L &x){
34         thrust(a += sz); thrust(b += sz-1);
35         for (int l = a, r = b+1; l < r; l >>= 1, r >>= 1) {
36             if(l&1) lazy[l] = M::h(lazy[l], x), l++;
37             if(r&1) --r, lazy[r] = M::h(lazy[r], x);
38         }
39         recalc(a);
40         recalc(b);
41     }
42
43     T query(int a, int b){ // [l, r)
44         thrust(a += sz);
45         thrust(b += sz-1);

```

```

45     T ll = M::e(), rr = M::e();
46     for(int l = a, r = b+1; l < r; l >>=1, r>>=1) {
47         if (l & 1) ll = M::f(ll, reflect(l++));
48         if (r & 1) rr = M::f(reflect(--r), rr);
49     }
50     return M::f(ll, rr);
51 }
52 };
53
54 struct Monoid{
55     using T = ll;
56     using L = ll;
57     static T f(T a, T b) { return min(a, b); }
58     static T g(T a, L b) {
59         if(b == e()) return a; else return b;
60     }
61     static L h(L a, L b) {
62         if(b == e()) return a; else return b;
63     }
64     static T e() { return 0; }
65     static L l() { return 0; }
66 };

```

### 3.5 modint(固定MOD)

```

1  template<ll M = 1000000007>
2  struct modint{
3      ll val;
4      modint(): val(0){}
5      template<typename T>
6      explicit modint(T t){val = t%M; if(val < 0) val += M;}
7
8      modint pow(ll k){
9          modint res(1), x(val);
10         while(k){
11             if(k&1) res *= x;
12             x *= x;
13             k >>= 1;
14         }
15         return res;
16     }
17     template<typename T>
18     modint& operator=(T a){ val = a%M; if(val < 0) val += M; return *this; }
19     modint inv() {return pow(M-2);}
20     modint& operator+=(modint a){ val += a.val; if(val >= M) val -= M; return *this;}
21     modint& operator-=(modint a){ val += M-a.val; if(val >= M) val -= M; return
        ↳ *this;}
22     modint& operator*=(modint a){ val = 1LL*val*a.val%M; return *this;}
23     modint& operator/=(modint a){ return (*this) *= a.inv();}
24     modint operator+(modint a) const {return modint(val) +=a;}
25     modint operator-(modint a) const {return modint(val) -=a;}
26     modint operator*(modint a) const {return modint(val) *=a;}
27     modint operator/(modint a) const {return modint(val) /=a;}
28     modint operator-(){ return modint(-val);}
29     bool operator==(const modint a) const {return val == a.val;}
30     bool operator!=(const modint a) const {return val != a.val;}
31     bool operator<(const modint a) const {return val < a.val;}
32 };
33
34 using mint = modint<MOD>;

```

### 3.6 行列

```
1  template<class T>
2  struct matrix {
3      vector<vector<T>> A;
4      matrix() = default;
5      matrix(size_t n, size_t m) : A(n, vector<T>(m)) {}
6      explicit matrix(size_t n) : A(n, vector<T>(n)) {};
7      size_t height() const { return (A.size()); }
8      size_t width() const { return (A[0].size()); }
9
10     const vector<T> &operator [] (int k) const { return A[k]; }
11     vector<T> &operator [] (int k) { return A[k]; }
12
13     static matrix I(size_t n){
14         matrix mat(n);
15         for (int i = 0; i < n; ++i) mat[i][i] = 1;
16         return mat;
17     }
18
19     matrix &operator+= (const matrix &B){
20         size_t h = height(), w = width();
21         for (int i = 0; i < h; ++i) {
22             for (int j = 0; j < w; ++j) {
23                 (*this)[i][j] += B[i][j];
24             }
25         }
26     }
27
28     matrix &operator-= (const matrix &B){
29         size_t h = height(), w = width();
30         for (int i = 0; i < h; ++i) {
31             for (int j = 0; j < w; ++j) {
32                 (*this)[i][j] -= B[i][j];
33             }
34         }
35     }
36
37     matrix &operator*=(const matrix &B)
38     {
39         size_t n = height(), m = B.width(), p = width();
40         matrix C (n, m);
41         for (int i = 0; i < n; ++i) {
42             for (int j = 0; j < m; ++j) {
43                 for (int k = 0; k < p; ++k) {
44                     C[i][j] = (C[i][j] + (*this)[i][k] * B[k][j]);
45                 }
46             }
47         }
48         A.swap(C.A);
49         return (*this);
50     }
51
52     template <class U>
53     matrix &operator%=(const U &m){
54         for (int i = 0; i < height(); ++i) {
55             for (int j = 0; j < width(); ++j) {
56                 (*this)[i][j] %= m;
57             }
58         }
59     }
60
61     matrix pow(ll n) const {
```

```

62     matrix a = (*this), res = I(height());
63     while(n > 0){
64         if(n & 1) res *= a;
65         a *= a;
66         n >>= 1;
67     }
68     return res;
69 }
70 matrix operator+(const matrix &A) const {return matrix(*this) += A;}
71 matrix operator-(const matrix &A) const {return matrix(*this) -= A;}
72 matrix operator*(const matrix &A) const {return matrix(*this) *= A;}
73 template <class U>
74 matrix operator%(const U &m) const {return matrix(*this) %= m;}
75
76 };

```

## 4 あまり使わなさそうなライブラリ

### 4.1 中国剰余定理

```

1  template <class T>
2  T pow_ (T x, T n, T M){
3      uint64_t u = 1, xx = x;
4      while (n > 0){
5          if (n&1) u = u * xx % M;
6          xx = xx * xx % M;
7          n >>= 1;
8      }
9      return static_cast<T>(u);
10 };
11
12
13 template<typename T>
14 T extgcd(T a, T b, T &x ,T &y){
15     for (T u = y = 1, v = x = 0; a; ) {
16         ll q = b/a;
17         swap(x -= q*u, u);
18         swap(y -= q*v, v);
19         swap(b -= q*a, a);
20     }
21     return b;
22 }
23
24 template<typename T>
25 T mod_inv(T x, T m){
26     T s, t;
27     extgcd(x, m, s, t);
28     return (m+s)% m;
29 }
30
31 pair<ll, ll> CRT(const vector<pair<ll, ll>> &a){
32     ll r = 0, M = 1;
33     for (int i = 0; i < a.size(); ++i) {
34         ll p, q;
35         ll d = extgcd(M, a[i].second, p, q);
36         if((a[i].first - r)%d != 0) return make_pair(0, -1);
37         ll tmp = (a[i].first - r) / d * p % (a[i].second / d);
38         r += M * tmp;
39         M *= a[i].second/d;
40     }
41     return make_pair((r+M) % M, M);
42 }

```

## 4.2 拡張ユークリッドの互除法

```
1  template<typename T>
2  T extgcd(T a, T b, T &x ,T &y){
3      for (T u = y = 1, v = x = 0; a; ) {
4          ll q = b/a;
5          swap(x -= q*u, u);
6          swap(y -= q*v, v);
7          swap(b -= q*a, a);
8      }
9      return b;
10 }
11
```

## 4.3 ベルマンフォード法

```
1  template <typename T>
2  struct edge {
3      int from, to;
4      T cost;
5
6      edge(int to, T cost) : from(-1), to(to), cost(cost) {}
7      edge(int from, int to, T cost) : from(from), to(to), cost(cost) {}
8
9      explicit operator int() const {return to;}
10 };
11
12 template <typename T>
13 vector<T> bellman_ford(int s, int V, vector<edge<T> > &G){
14     const T INF = numeric_limits<T>::max();
15     vector<T> d(V, INF);
16     d[s] = 0;
17     for (int i = 0; i < V - 1; ++i) {
18         for (auto &&e : G) {
19             if (d[e.from] == INF) continue;
20             d[e.to] = min(d[e.to], d[e.from] + e.cost);
21         }
22     }
23     for (auto &&e : G) {
24         if(d[e.from] == INF) continue;
25         if(d[e.from] + e.cost < d[e.to]) return vector<T> ();
26     }
27     return d;
28 }
```

## 4.4 二部グラフの最大マッチング

```
1  class Bipartite_Matching {
2      vector<vector<int>> G;
3      vector<int> match, used, alive;
4      int t;
5  public:
6      explicit Bipartite_Matching(int n): t(0), G(n), match(n, -1), used(n, 0),
7          ↪ alive(n, -1){};
8
9      void connect(int a, int b){
10         G[a].emplace_back(b);
11         G[b].emplace_back(a);
12     }
13
14     bool dfs(int x){
15         used[x] = t;
16     }
17 }
```



```

15     for (auto &&i : G[x]) {
16         int w = match[i];
17         if(alive[i] == 0) continue;
18         if(w == -1 || (used[w] != t && dfs(w))) {
19             match[x] = i;
20             match[i] = x;
21             return true;
22         }
23     }
24     return false;
25 }
26
27 int matching() {
28     int ans = 0;
29     for (int i = 0; i < G.size(); ++i) {
30         if(alive[i] == 0) continue;
31         if(match[i] == -1) {
32             ++t;
33             ans += dfs(i);
34         }
35     }
36     return ans;
37 }
38 };

```

## 4.5 最大独立集合

```

1  class IndependentSet {
2      int n;
3      vector<vector<int>> G;
4      void dfs(int x, vector<bool> &visited, vector<bool> &gcan, vector<bool> &alive){
5          stack<int> s;
6          s.emplace(x);
7          while(!s.empty()){
8              int y = s.top();
9              visited[y] = true;
10             gcan[y] = true;
11             s.pop();
12             for (auto &&i : G[y]) {
13                 if(!visited[i] && alive[i]) s.emplace(i);
14             }
15         }
16     }
17
18     int ConnectedCase(vector<bool> can) {
19         int pMax = -1, pMin = -1, Max = -1, Min = n+1, num = 0;
20         for (int i = 0; i < n; ++i) {
21             if(!can[i]) continue;
22             ++num;
23             int tnum = 0;
24             for (auto &&j : G[i]) if(can[j]) ++tnum;
25             if(Max < tnum) Max = tnum, pMax = i;
26             if(Min > tnum) Min = tnum, pMin = i;
27         }
28         if(num == 1) return 1;
29         if(Max <= 2){
30             if(Min == 1) return (num+1)/2;
31             else return num/2;
32         }
33         int ans = 0;
34         vector<bool> ncan = can;
35         if(Min < 2){

```

```

36     ncan[pMin] = false;
37     for (auto &&i : G[pMin]) ncan[i] = false;
38     ans = max(ans, GeneralCase(ncan) + 1);
39 }else {
40     ncan[pMax] = false;
41     for (auto &&i : G[pMax]) ncan[i] = false;
42     int temp = GeneralCase(ncan);
43     ans = max(ans, temp+1);
44     ncan = can;
45     ncan[pMax] = false;
46     ans = max(ans, GeneralCase(ncan));
47 }
48 return ans;
49 }
50
51 int GeneralCase(vector<bool> alive) {
52     if(n <= 1) return n;
53     vector<bool> visited(n, 0);
54     int res = 0;
55     for (int i = 0; i < n; ++i) {
56         if(!visited[i] && alive[i]){
57             vector<bool> gcan(n, false);
58             dfs(i, visited, gcan, alive);
59             res += ConnectedCase(gcan);
60         }
61     }
62     return res;
63 }
64 public:
65     explicit IndependentSet(int n): n(n), G(n) {}
66     void add_edge(int u, int v){
67         G[u].emplace_back(v);
68         G[v].emplace_back(u);
69     }
70     int stable_set() {
71         vector<bool> alive(n, true);
72         return GeneralCase(alive);
73     }
74 };

```

## 4.6 最近共通祖先 (LCA)

```

1  template <class M>
2  struct SegmentTree{
3      using T = typename M::T;
4      int sz;
5      vector<T> seg;
6      explicit SegmentTree(int n) {
7          sz = 1;
8          while(sz < n) sz <= 1;
9          seg.assign(2*sz, M::e());
10     }
11
12     void set(int k, const T &x){ seg[k + sz] = x; }
13
14     void build(){
15         for (int i = sz-1; i > 0; --i) seg[i] = M::f(seg[2*i], seg[2*i+1]);
16     }
17
18     void update(int k, const T &x){
19         k += sz;
20         seg[k] = x;

```

```

21     while (k >= 1) seg[k] = M::f(seg[2*k], seg[2*k+1]);
22 }
23
24 T query(int a, int b){
25     T l = M::e(), r = M::e();
26     for(a += sz, b += sz; a < b; a >= 1, b >= 1){
27         if(a & 1) l = M::f(l, seg[a++]);
28         if(b & 1) r = M::f(seg[--b], r);
29     }
30     return M::f(l, r);
31 }
32
33 T operator[] (const int &k) const { return seg[k + sz]; }
34 };
35
36 struct Monoid{
37     using T = pair<int, int>;
38     static T f(T a, T b) { return min(a, b); }
39     static T e() { return T(INF<int>, -1); }
40 };
41
42 class Graph {
43     void dfs_euler(int v, int p, int d, int &k){
44         id[v] = k;
45         vs[k] = v;
46         depth[k++] = d;
47         for (auto &&u : G[v]) {
48             if(u != p){
49                 dfs_euler(u, v, d+1, k);
50                 vs[k] = v;
51                 depth[k++] = d;
52             }
53         }
54     }
55 public:
56     int n;
57     vector<vector<int>> G;
58     vector<int> vs, depth, id;
59     explicit Graph(int n) : n(n), G(n), vs(2*n-1), depth(2*n-1), id(n) {};
60     void add_edge(int a, int b){
61         G[a].emplace_back(b);
62         G[b].emplace_back(a);
63     }
64
65     void eulertour(int root) {
66         int k = 0;
67         dfs_euler(root, -1, 0, k);
68     }
69 };
70
71 class LCA {
72     Graph G;
73     SegmentTree<Monoid> seg;
74 public:
75     explicit LCA(Graph G) : G(G), seg(2*G.n-1) {
76         int n = G.n;
77         for (int i = 0; i < 2*n-1; ++i) {
78             seg.set(i, pair<int, int>(G.depth[i], i));
79         }
80         seg.build();
81     };
82
83     int lca(int u, int v){

```

```

84         if(G.id[u] > G.id[v]) swap(u, v);
85         return seg.query(G.id[u], G.id[v]+1).second;
86     }
87 };

```

## 4.7 Convex-Hull-Trick

```

1  template<class T>
2  class monotonic_CHT {
3      using P = pair<T, T>;
4      vector<P> lines;
5  public:
6      monotonic_CHT() {}
7
8      bool check(P l1, P l2, P l3){
9          if(l1 < l3) swap(l1, l3);
10         return (l3.second - l2.second)*(l2.first-l1.first)
11             >= (l2.second - l1.second)*(l3.first-l2.first);
12     }
13     void add(T a, T b){
14         P line(a, b);
15         while(lines.size() >= 2 && check(*(lines.end()-2), lines.back(), line))
16             ↪ lines.pop_back();
17         lines.emplace_back(line);
18     }
19     T f(int i, T x){
20         return lines[i].first * x + lines[i].second;
21     }
22
23     T f(P line, T x){
24         return line.first * x + line.second;
25     }
26
27     T get(T x){
28         static int head = 0;
29         while(lines.size()-head >= 2 && f(head, x) >= f(head+1, x)) ++head;
30         return f(head, x);
31     }
32 };

```

## 4.8 スライド最小値

```

1  template<class T, class F>
2  class sliding_window {
3      vector<T> v;
4      deque<T> Q;
5      F f;
6  public:
7      int l, r;
8      explicit sliding_window(vector<T> &v, F f) : v(v), f(f), l(0), r(0) {};
9      void set(vector<T> &u){
10         v = u;
11         Q.clear();
12         l = 0; r = 0;
13     }
14     void reset(){
15         Q.clear();
16         l = 0, r = 0;
17     }
18     void slideL(){

```

```

19         if(Q.front() == l++) Q.pop_front();
20     }
21     void slideR() {
22         while(!Q.empty() && !f(v[Q.back()], v[r])) Q.pop_back();
23         Q.push_back(r++);
24     }
25     T get_index() {
26         if(l == r) return 0;
27         return Q.front();
28     }
29     T value() {
30         if(l == r) return 0;
31         return v[Q.front()];
32     }
33 };

```

## 4.9 ポテンシャル付き Union-Find

```

1  template <class T>
2  class WeightedUnionFind {
3      vector<int> uni;
4      vector<T> weights;
5      int n;
6  public:
7      explicit WeightedUnionFind(int n, T SUM_UNITY = 0) :
8          uni(static_cast<u32>(n), -1), n(n), weights(n, SUM_UNITY) {}
9
10     int root(int a) {
11         if (uni[a] < 0) return a;
12         else {
13             int r = root(uni[a]);
14             weights[a] += weights[uni[a]];
15             return (uni[a] = r);
16         }
17     }
18
19     bool unite(int a, int b, T w) {
20         w += weight(a); w -= weight(b);
21         a = root(a);
22         b = root(b);
23         if(a == b) return false;
24         if(uni[a] > uni[b]) swap(a, b), w = -w;
25         uni[a] += uni[b];
26         uni[b] = a;
27         weights[b] = w;
28         return true;
29     }
30
31     int size(int a) {
32         return -uni[root(a)];
33     }
34
35     T weight(T a) {
36         root(a);
37         return weights[a];
38     }
39     int diff(int x, int y) {
40         return weight(y) - weight(x);
41     }
42 };

```

## 4.10 modint(実行時MOD)

```
1 struct modint {
2     static ll &mod(){
3         static ll mod_ = 0;
4         return mod_;
5     }
6
7     static void set_mod(const ll x) { mod() = x; }
8     static ll M() {return mod(); }
9
10    ll val;
11    modint(): val(0){}
12    template<typename T>
13    explicit modint(T t){val = t%M(); if(val < 0) val += M();}
14
15    modint pow(ll k){
16        modint res(1), x(val);
17        while(k){
18            if(k&1) res *= x;
19            x *= x;
20            k >>= 1;
21        }
22        return res;
23    }
24    template<typename T>
25    modint& operator=(T a){ val = a%M(); if(val < 0) val += M(); return *this; }
26    modint inv() {return pow(M()-2);}
27    modint& operator+=(modint a){ val += a.val; if(val >= M()) val -= M(); return
    ↪ *this;}
28    modint& operator-=(modint a){ val += M()-a.val; if(val >= M()) val -= M(); return
    ↪ *this;}
29    modint& operator*=(modint a){ val = val*a.val%M(); return *this;}
30    modint& operator/=(modint a){ return (*this) *= a.inv();}
31    modint operator+(modint a) const {return modint(val) +=a;}
32    modint operator-(modint a) const {return modint(val) -=a;}
33    modint operator*(modint a) const {return modint(val) *=a;}
34    modint operator/(modint a) const {return modint(val) /=a;}
35    modint operator-(){return modint(-val); }
36    bool operator==(const modint a) const {return val == a.val;}
37    bool operator!=(const modint a) const {return val != a.val;}
38    bool operator<(const modint a) const {return val < a.val;}
39 };
```

## 4.11 Xor-Shift

```
1 #include <chrono>
2 class xor_shift {
3     uint32_t x, y, z, w;
4 public:
5     xor_shift() :
6     ↪ x(static_cast<uint32_t>((chrono::system_clock::now().time_since_epoch().count()) & ((1LL
7     ↪ << 32)-1))),
8     y(1068246329), z(321908594), w(1234567890) {};
9
10    uint32_t urand(){
11        uint32_t t;
12        t = x ^ (x << 11);
13        x = y; y = z; z = w;
14        w = (w ^ (w >> 19)) ^ (t ^ (t >> 8));
15        return w;
16    }
17 };
```

```

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  };

```