1 Template

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <map>
#include <set>
#include <queue>
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cstring>

using namespace std;
using ll = long long;

#define all(c) (c).begin(), (c).end()
#define rep(i,n) for(int i=0;i<(int)(n);i++)
#define pb(e) push_back(e)
#define mp(a, b) make_pair(a, b)
#define fr first
#define sc second

const ll INF=1e9;
const ll MOD=1e9+7;
int dx[4]={1,0,-1,0};
int dy[4]={0,1,0,-1};

int main() {
    return 0;
}
```

2 Graph

2.1 二部マッチング

```cpp
/*
 * 二部マッチング
 * O(|E||V|)
 *
 * http://poj.org/problem?id=3041
 */

struct BipartiteMatching {
    vector<vector<int> > G;
    vector<int> match;
    vector<bool> used;

    int V;
    BipartiteMatching(int V):V(V){
        G.resize(V);
        match.resize(V);
        used.resize(V);
    }

    void add_edge(int u,int v) {
        G[u].push_back(v);
        G[v].push_back(u);
    }

    bool dfs(int v) {
        used[v]=true;
        for(int i=0;i<G[v].size();i++){
            int u=G[v][i];
            int w=match[u];
            if(w<0||!used[w]&&dfs(w)) {
                match[v]=u;
                match[u]=v;
                return true;
            }
        }
        return false;
    }

    int operator() () {
        int res=0;
        match.assign(V,-1);
        for(int v=0;v<V;v++){
            if(match[v]<0) {
                used.assign(V,0);
                if(dfs(v)) res++;
            }
        }
        return res;
    }
};
```

2.2 ダイクストラ

```cpp
using Weight = ll;
struct Edge {
    int to;
    Weight cost;
};
struct Node {
    int v;
    Node(int v) : v(v) {}
    bool operator<(const Node &rhs) const { return tie(v) < tie(rhs.v); }
    bool operator==(const Node &rhs) const { return tie(v) == tie(rhs.v); }
};

namespace std {
template <>
struct hash<Node>{
    size_t operator()(const Node &node) const {
        size_t seed = 0;

        size_t v_hash = hash<int>()(node.v);

```

```
        seed ^= v_hash + 0x9e3779b9 + (seed << 6) + (seed >> 2);
        return seed;
    }
};
}
struct State : public Node {
    Weight cost;
    State(Node node, Weight cost) : Node(node), cost(cost) {}
    bool operator<(const State &rhs) const { return cost > rhs.cost; }
};

unordered_map<Node, Weight> dijkstra(const vector<vector<Edge>> &adj,
                        const Node &source) {
    unordered_map<Node, Weight> dist;
    priority_queue<State> que;

    que.push(State(source, 0));
    dist[source] = 0;

    while (que.size()) {
        State s = que.top();
        que.pop();
        Node cur{s.v};

        for (auto u : adj[s.v]) {
            Node next{u.to};
            if (!dist.count(next) || dist[next] > dist[cur] + u.cost) {
                dist[next] = dist[cur] + u.cost;
                que.push(State(next, dist[next]));
            }
        }
    }

    return dist;
}
```

2.3 最大流

```
/*
 * 最大流
 * Ford 法 Fulkerson
 * O(F|E|)
 *
 * http://poj.org/problem?id=3281
 *
 */

struct FordFulkerson{
    struct Edge {
        int to,cap,rev;
        Edge(int to=0,int cap=0,int rev=0) :
            to(to),cap(cap),rev(rev){}
    };

    int V;
    vector<vector<Edge> > G;
    vector<bool> used;

    FordFulkerson(int V) : V(V) {
        G.resize(V);
        used.assign(V,false);
    }

    void add_edge(int from,int to,int cap) {
        G[from].pb(Edge(to,cap,G[to].size()));
        G[to].pb(Edge(from,0,G[from].size()-1));
    }

    int dfs(int v,int t,int f) {
        if(v==t) return f;
        used[v]=true;
        for(int i=0;i<G[v].size();i++) {
            Edge &e=G[v][i];

            if(!used[e.to] && e.cap>0) {
                int d=dfs(e.to,t,min(f,e.cap));
                if(d>0) {
                    e.cap-=d;
                    G[e.to][e.rev].cap+=d;
                    return d;
                }
            }
        }
        return 0;
    }

    int max_flow(int s,int t) {
        int flow=0;
        while(1) {
            used.assign(V,false);
            int f=dfs(s,t,INF);
            if(f==0) break;
            flow+=f;
        }

        return flow;
    }
};
```

2.4 最小費用流

```

struct MinumumCostFlow {
    static const int MAX_V = 10004;
    typedef pair<int, int> P;
    struct Edge {
        int to, cost, cap, rev;
        Edge(int to = 0, int cap = 0, int cost = 0, int rev = 0)
            : to(to), cap(cap), cost(cost), rev(rev) {}
    };

    int V;
    vector<Edge> G[MAX_V];
    int h[MAX_V];
```

```
    int dist[MAX_V];
    int prev_v[MAX_V], prev_e[MAX_V];

    void add_edge(int from, int to, int cap, int cost) {
        G[from].push_back(Edge(to, cap, cost, G[to].size()));
        G[to].push_back(Edge(from, 0, -cost, G[from].size() - 1));
    }

    int operator()(int s, int t, int f) {
        int res = 0;
        fill(h, h + V, 0);

        while (f > 0) {
            priority_queue<P, vector<P>, greater<P>> que;
            fill(dist, dist + V, INF);
            dist[s] = 0;
            que.push(P(0, s));

            while (que.size()) {
                P p = que.top();
                que.pop();
                int v = p.second;
                if (dist[v] < p.first) continue;
                rep(i, G[v].size()) {
                    Edge &e = G[v][i];
                    if (e.cap > 0 &&
                        dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]) {
                        dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
                        prev_v[e.to] = v;
                        prev_e[e.to] = i;
                        que.push(P(dist[e.to], e.to));
                    }
                }
            }
            if (dist[t] == INF) return -1;
            rep(v, V) h[v] += dist[v];

            int d = f;
            for (int v = t; v != s; v = prev_v[v]) {
                d = min(d, G[prev_v[v]][prev_e[v]].cap);
            }
            f -= d;
            res += d * h[t];

            for (int v = t; v != s; v = prev_v[v]) {
                Edge &e = G[prev_v[v]][prev_e[v]];
                e.cap -= d;
                G[v][e.rev].cap += d;
            }
        }

        return res;
    }
};
```

2.5 最小共通祖先

```
// Verified http://abc014.contest.atcoder.jp/tasks/abc014_4
struct LCA {
    int N;
    int root;
    int log2_n;
    vector<vector<int>> parent;
    vector<int> depth;
    vector<vector<int>> G;
    LCA(int N,int root=0) : N(N), root(root) {
        log2_n = log2(N) + 1;
        parent.resize(log2_n);
        rep(i,log2_n) parent[i].resize(N);
        depth.resize(N);
        G.resize(N);
    }
    void add_edge(int v,int u) {
        G[v].pb(u);
    }
    void init() {
        dfs(root,-1,0);
        rep(k,log2_n-1) rep(v,N) {
            if(parent[k][v]<0) parent[k+1][v]=-1;
            else parent[k+1][v]=parent[k][parent[k][v]];
        }
    }
    void dfs(int v,int p,int d) {
        parent[0][v]=p;
        depth[v]=d;
        rep(i,G[v].size()) {
            if(G[v][i]!=p) dfs(G[v][i],v,d+1);
        }
    }
    int operator()(int u,int v) {
        if(depth[u]>depth[v]) swap(u,v);
        rep(k,log2_n) {
            if((depth[v]-depth[u])>>k&1) {
                v=parent[k][v];
            }
        }
        if(v==u) return u;

        for(int k=log2_n-1;k>=0;k--) {
            if(parent[k][u]!=parent[k][v]) {
                u=parent[k][u];
                v=parent[k][v];
            }
        }
        return parent[0][u];
    }
    int dist(int v,int u) {
        int a=operator()(v,u);
        return depth[v] - depth[a] + depth[u] - depth[a];
    }
};
```

3 Math

3.1 繰り返し 2 乗法

```
ll mod_pow(ll a, int n, int m) {
    if (n == 0)
        return 1;
    else if (n % 2 == 1)
        return (a * mod_pow((a * a) % m, n >> 1, m)) % m;
    else
        return mod_pow((a * a) % m, n >> 1, m);
}
```

3.2 コンビネーション

```
/*
 * 前処理O(N)
 * クエリO(1)
 *
 * Verified
 * http://yukicoder.me/problems/184
 */

template<class T,size_t N,T MOD>
struct Combination {
    typedef T int_type;
    vector<int_type> fact,factr,inv;
    Combination() {
        fact.resize(2*N+1);
        factr.resize(2*N+1);
        inv.resize(2*N+1);
        fact[0]=factr[0]=1;
        inv[1]=1;
        for(int i=2;i<=2*N;i++) inv[i] = inv[MOD % i] * (MOD - MOD / i) % MOD;
        for(int i=1;i<=2*N;i++) fact[i]=fact[i-1]*i%MOD, factr[i]=factr[i-1]*inv[i]%MOD;
    }

    int_type C(int n, int r) {
        if(r<0 || r>n) return 0;
        return factr[r]*fact[n]%MOD*factr[n-r]%MOD;
    }

    int_type P(int n,int r) {
        if(r<0 || r>n) return 0;
        return fact[n]*factr[n-r]%MOD;
    }

    int_type H(int n, int r) {
        if(n==0 && r==0) return 1;
        return C(n+r-1,r);
    }
};
```

4 DataStructure

4.1 Segment Tree

```
// Verified
// http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DSL_2_A
//


template<class T>
struct SegTree {
    typedef T int_type;
    static const size_t MAX_N = 1 << 17;
    static const int_type INIT_VAL = (int_type(1)<<31)-1;
    int_type data[2 * MAX_N - 1];
    size_t n;
    SegTree(size_t n__) {
        n=1;
        while(n < n__) n*=2;
        rep(i,2*n-1) data[i]=INIT_VAL;
    }
    void update(size_t k,int_type a) {
        k+=n-1;
        data[k]=a;
        while(k>0) {
            k=(k-1)/2;
            data[k]=min(data[k*2+1],data[k*2+2]);
        }
    }

    int_type query(size_t a,size_t b,size_t k,size_t l,size_t r) {
        if(r<=a || b<=l) return INIT_VAL;
        if(a<=l && r<=b) return data[k];
        else {
            int_type vl = query(a,b,k*2+1,l,(l+r)/2);
            int_type vr = query(a,b,k*2+2,(l+r)/2,r);
            return min(vl,vr);
        }
    }
    int_type query(size_t a,size_t b) {
        return query(a,b,0,0,n);
    }
};
```

4.2 StarrySky Tree

```
// Verified
// http://code-festival-2015-final-open.contest.atcoder.jp/tasks/codefestival_2015_final_d
// http://judge.u-aizu.ac.jp/onlinejudge/cdescription.jsp?cid=RitsCamp16Day3&pid=F

template<class T>
struct StarrySkyTree {
    using int_type = T;
    vector<T> data;
    vector<T> lazy;
    int N;
```

```
    StarrySkyTree(int n) {
        N=1;
        while(N<n) N<<=1;
        data.assign(2*N-1, 0);
        lazy.assign(2*N-1, 0);
    }
    // [a,b)
    void add(int a,int b, int_type val) {
        add(a,b,val,0,0,N);
    }
    int_type get(int a,int b) {
        return get(a,b,0,0,N);
    }

    void add(int a,int b,int_type val, int k,int l,int r) {
        if(r<=a || b<=l) return;
        if(a<=l && r<=b) {
            lazy[k]+=val;
            return;
        }
        add(a,b,val,k*2+1,l,(l+r)/2);
        add(a,b,val,k*2+2,(l+r)/2,r);
        data[k]=max(data[k*2+1]+lazy[k*2+1], data[k*2+2]+lazy[k*2+2]);
    }

    int_type get(int a,int b, int k,int l, int r) {
        if(r<=a || b<=l) return -INF;
        if(a<=l && r<=b) return data[k]+lazy[k];
        auto lval = get(a,b,k*2+1,l,(l+r)/2);
        auto rval = get(a,b,k*2+2,(l+r)/2,r);
        return max(lval, rval)+lazy[k];
    }
};
```

4.3 Treap

```
int xor128() {
    static int x = 123456789, y = 362436069, z = 521288629, w = 88675123;
    int t;
    t = (x ^ (x << 11));
    x = y;
    y = z;
    z = w;
    return (w = (w ^ (w >> 19)) ^ (t ^ (t >> 8)));
}

struct Node {
    int val;
    Node *ch[2];
    // 優先度
    int pri;
    // 部分木の個数
    int cnt;
    // 部分木の値の和
    int s;
    Node(int val, int p) : val(val), pri(p), cnt(1), s(val) {
        ch[0] = ch[1] = nullptr;
    }
};
int count(Node *n) { return n == nullptr ? 0 : n->cnt; }
int sum(Node *n) { return n == nullptr ? 0 : n->s; }

Node *update(Node *n) {
    n->cnt = count(n->ch[0]) + count(n->ch[1]) + 1;
    n->s = sum(n->ch[0]) + sum(n->ch[1]) + n->val;
    return n;
}

Node *merge(Node *l, Node *r) {
    if (l == nullptr || r == nullptr) return l == nullptr ? r : l;
    if (l->pri > r->pri) {
        l->ch[1] = merge(l->ch[1], r);
        return update(l);
    } else {
        r->ch[0] = merge(l, r->ch[0]);
        return update(r);
    }
}

pair<Node *, Node *> split(Node *t, int k) {
    if (t == nullptr) return make_pair(nullptr, nullptr);
    int c = count(t->ch[0]);
    if (k <= c) {
        auto s = split(t->ch[0], k);
        t->ch[0] = s.second;
        return make_pair(s.first, update(t));
    } else {
        auto s = split(t->ch[1], k - c - 1);
        t->ch[1] = s.first;
        return make_pair(update(t), s.second);
    }
}

Node *insert(Node *t, int k, int val) {
    Node *m = new Node(val, xor128() % 1007);
    auto p = split(t, k);
    return merge(merge(p.first, m), p.second);
}

Node *erase(Node *t, int k) {
    auto p1 = split(t, k);
    auto p2 = split(p1.second, 1);
    return merge(p1.first, p2.second);
}

// を挿入するべき場所を探す val
int upper_bound(Node *t, int val) {
    if (t == nullptr) return 0;
    if (t->val > val)
        return upper_bound(t->ch[0], val);
    else
        return upper_bound(t->ch[1], val) + count(t->ch[0]) + 1;
}
```

```
// 番目の大きいやつ x
int get(Node *t, int x) {
    int c = count(t->ch[0]);
    if (c == x) return t->val;
    if (x < c)
        return get(t->ch[0], x);
    else
        return get(t->ch[1], x - c - 1);
}
```

4.4 UnionFind

```
#include <vector>
template<typename T>
class UnionFind {
    int size_;
    std::vector<T> par;
    std::vector<T> rank;

public:
    UnionFind(int size_) : size_(size_) {
        par.resize(size_);
        rank.resize(size_);

        for(int i=0; i<size_; i++) {
            par[i] = i;
            rank[i] = 0;
        }
    }

    T find(T x) {
        return par[x] == x ? x : par[x] = find(par[x]);
    }
    void unite(T x,T y) {
        x = find(x);
        y = find(y);
        if(x == y) return;

        if(rank[x] < rank[y]) {
            par[x] = y;
        }
        else {
            par[y] = x;
            if(rank[x] == rank[y]) rank[x]++;
        }
    }
};
```

5 String

5.1 ローリングハッシュ

```
struct RollingHash {
    typedef long long int_type;
    typedef pair<int_type,int_type> hash_type;

    int_type base1;
    int_type base2;
    int_type mod1;
    int_type mod2;
    vector<int_type> hash1;
    vector<int_type> hash2;
    vector<int_type> pow1;
    vector<int_type> pow2;

    RollingHash() : base1(1009), base2(1007), mod1(1000000007), mod2(1000000009) {}

    void init(const string &s) {
        int n = s.size();

        hash1.assign(n+1,0);
        hash2.assign(n+1,0);
        pow1.assign(n+1,1);
        pow2.assign(n+1,1);

        for(int i=0;i<n;i++) {
            hash1[i+1] = (hash1[i]+s[i]) * base1 % mod1;
            hash2[i+1] = (hash2[i]+s[i]) * base2 % mod2;
            pow1[i+1] = pow1[i] * base1 % mod1;
            pow2[i+1] = pow2[i] * base2 % mod2;
        }
    }

    hash_type get(int l,int r) {
        int_type t1 = ((hash1[r] - hash1[l] * pow1[r-l]) % mod1 + mod1) % mod1;
        int_type t2 = ((hash2[r] - hash2[l] * pow2[r-l]) % mod2 + mod2) % mod2;
        return make_pair(t1, t2);
    }

    RollingHash::hash_type concat(hash_type h1, hash_type h2, int h2_len) {
        return make_pair((h1.fr*pow1[h2_len]+h2.fr)%mod1, (h1.sc*pow2[h2_len]+h2.sc)%mod2);
    }

};
```

6 Geometry

6.1 幾何

```
#include <complex>
#include <vector>
#include <cmath>
#include <utility>
#include <cassert>
#include <cmath>

const double EPS=1e-10;

#define equals(a, b) (fabs((a)-(b))<EPS)
#define X real()
```

```
#define Y imag()

using namespace std;

typedef complex<double> Point;
typedef complex<double> Vector;

struct Segment {
    Point p1, p2;
};
typedef Segment Line;

struct Circle {
    Point c;
    double r;
    Circle(Point c=Point(), double r=0.0) :
        c(c),r(r){}
};

typedef vector<Point> Polygon;

double dot(Vector a, Vector b) {
    return a.X*b.X + a.Y*b.Y;
}

double cross(Vector a,Vector b) {
    return a.X*b.Y - a.Y*b.X;
}

Point project(Segment s, Point p) {
    Vector base = s.p2-s.p1;
    double r=dot(p-s.p1,base) / norm(base);
    return s.p1+base*r;
}

Point reflect(Segment s, Point p) {
    return p+(project(s,p)-p)*2.0;
}

enum CCW {
    COUNTER_CLOCKWISE=1,
    CLOCKWISE=-1,
    ONLINE_BACK=2,
    ONLINE_FRONT=-2,
    ON_SEGMENT=0,
};

int ccw(Point p0, Point p1, Point p2) {
    Vector a=p1-p0;
    Vector b=p2-p0;
    if(cross(a,b)>EPS) return CCW::COUNTER_CLOCKWISE;
    if(cross(a,b)<-EPS) return CCW::CLOCKWISE;
    if(dot(a,b)<-EPS) return CCW::ONLINE_BACK;
    if(norm(a)<norm(b)) return CCW::ONLINE_FRONT;

    return CCW::ON_SEGMENT;
}

double getDistance(Point a,Point b) {
    return abs(a-b);
}

double getDistanceLP(Line l, Point p) {
    return abs(cross(l.p2-l.p1,p-l.p1)/abs(l.p2-l.p1));
}

bool intersect(Point p1,Point p2,Point p3,Point p4) {
    return (ccw(p1,p2,p3)*ccw(p1,p2,p4)<=0&&
            ccw(p3,p4,p1)*ccw(p3,p4,p2)<=0);
}

bool intersect(Segment s1,Segment s2) {
    return intersect(s1.p1,s1.p2,s2.p1,s2.p2);
}
bool intersect(Circle c,Line l) {
    return getDistanceLP(l,c.r)<=c.r;
}

Point getCrossPoint(Segment s1,Segment s2) {
    Vector base=s2.p2-s2.p1;
    double d1=abs(cross(base,s1.p1-s2.p1));
    double d2=abs(cross(base,s1.p2-s2.p1));
    double t=d1/(d1+d2);
    return s1.p1+(s1.p2-s1.p1)*t;
}

pair<Point, Point> getCrossPoints(Circle c, Line l) {
    // assert(intersect(c,l));
    Vector pr=project(l,c.c);
    Vector e=(l.p2-l.p1)/abs(l.p2-l.p1);
    double base=sqrt(c.r*c.r-norm(pr-c.c));

    return make_pair(pr+e*base,pr-e*base);

}

pair<Point,Point> getCrossPoints(Circle c1, Circle c2) {
    double d=abs(c1.c-c2.c);
    double a=acos((c1.r*c1.r+d*d-c2.r*c2.r)/(2*c1.r*d));
    double t=arg(c2.c-c1.c);

    return make_pair(c1.c+polar(c1.r,t+a),c1.c+polar(c1.r,t-a));
}

// IN 2, ON 1, OUT 0
int contains(Polygon g, Point p) {
    int n=g.size();
    bool x=false;
    for(int i=0;i<n;i++) {
        Point a=g[i]-p,b=g[(i+1)%n]-p;
        if(abs(cross(a,b))<EPS && dot(a,b)<EPS) return 1;
        if(a.Y>b.Y) swap(a,b);
        if(a.Y<EPS&&EPS<b.Y&&cross(a,b) > EPS) x=!x;
    }

```

```
126     return x?2:0;
127 }
```