Divide & Conquer DP: `C(a,c) + C(b,d) <= C(a,d) + C(b,c) where a <= b <= c <= d`

Number Theory:

```
namespace nt {
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
    int64_t rand(int64_t l, int64_t r) { return l + rng() % (r - l + 1); }
    template <typename T = int64_t, typename U = __int128_t>
    T pow(T a, T b, T mod) {
        a %= mod;
        T r = 1;
        for(T i = b; i; i >>= 1, a = (U)a * a % mod) if(i & 1) {
            r = (U)r * a % mod;
        }
        return r;
    }
    template <typename T>
    T gcd(T a, T b, T &x, T &y) {
        if(!b) {
            x = 1, y = 0;
            return a;
        }
        T x1, y1;       T g = gcd(b, a % b, x1, y1);
        x = y1;        y = x1 - y1 * (a / b);
        return g;
    }
    int64_t inverse(int64_t a, int64_t m) {
        int64_t x, y;
        int64_t g = gcd(a, m, x, y);
        if(g > 1)      return -1;
        x = (x % m + m) % m;
        return x;
    }
    // Returns minimum x for which a ^ x % m = b % m, a and m are coprime.
    int solve(int a, int b, int m) {
        a %= m, b %= m;
        int n = sqrt(m) + 1, an = 1;
        for (int i = 0; i < n; ++i) an = (an * 1LL * a) % m;
        unordered_map<int, int> vals;
        for (int q = 0, cur = b; q <= n; ++q) {
            vals[cur] = q, cur = (cur * 1LL * a) % m;
        }
        for (int p = 1, cur = 1; p <= n; ++p) {
            cur = (cur * 1LL * an) % m;
            if (vals.count(cur)) {
                int ans = n * p - vals[cur];
                return ans;
            }
        }
        return -1;
    }
```

```
namespace MillerRabin{
    bool isComposite(int64_t x, int64_t a, int64_t d, int64_t s) {
        int64_t y = pow(a, d, x);
        if(y == 1 || y == x - 1) return false;
        for(int64_t r = 1; r < s; r++) {
            y = (__int128_t)y * y % x;
            if(y == x - 1) return false;
        }
        return true;
    }
    bool isPrime(int64_t x) {
        if(x < 2) return false;
        for(int64_t a: sieve(40)) if(a == x)    return true;
        int64_t r = 0, d = x - 1;
        while(d % 2 == 0)  d /= 2,     ++r;
        for(int64_t a: sieve(40)) if(isComposite(x, a, d, r))
                return false;
        return true;
    }
}
namespace Rho{
    int64_t f(int64_t x, int64_t c, int64_t mod) {
        return ((__int128_t)x * x + c) % mod;
    }

    int64_t brent(int64_t n) {
        int64_t x = rand(2, n), g = 1, q = 1, xs, y, c = rand(1, n);
        int m = 128, l = 1;
        while (g == 1) {
            y = x;
            for (int i = 1; i < l; i++)   x = f(x, c, n);
            int k = 0;
            while (k < l && g == 1) {
                xs = x;
                for (int i = 0; i < m && i < l - k; i++) {
                    x = f(x, c, n),q=(__int128_t)q * abs(y - x) % n;
                }
                g = __gcd(q, n), k += m;
            }
            l *= 2;
        }
        if (g == n) {
            do {
                xs = f(xs, c, n);
                g = __gcd(abs(xs - y), n);
            } while (g == 1);
        }
        return g;
    }
```

```
        vector <int64_t> factor(int64_t n) {
            if(n == 1)return {};
            if(MillerRabin::isPrime(n)) {return {n};}
            int64_t dx = n;
            while(dx == n)  dx = brent(n);
            auto L = factor(dx), R = factor(n / dx);
            L.insert(L.end(), R.begin(), R.end());
            return L;
        }
    }
}
using namespace nt;
```

FFT:

```
namespace FFT {
    typedef long long ll;
    typedef long double ld;
    struct base {
        typedef double T; T re, im;
        base() :re(0), im(0) {}
        base(T re) :re(re), im(0) {}
        base(T re, T im) :re(re), im(im) {}
        base operator + (const base& o) const { return base(re + o.re, im +
o.im); }
        base operator - (const base& o) const { return base(re - o.re, im -
o.im); }
        base operator * (const base& o) const { return base(re * o.re - im *
o.im, re * o.im + im * o.re); }
        base operator * (ld k) const { return base(re * k, im * k); }
        base conj() const { return base(re, -im); }
    };
    const int N = 21;
    const int MAXN = (1 << N);
    base w[MAXN];
    base f1[MAXN];
    int rev[MAXN];
    void build_rev(int k) {
        static int rk = -1;
        if (k == rk)return; rk = k;
        int K=1<<k;
        for (int i = 1; i <= K; i++) {
            int j = rev[i - 1], t = k - 1;
            while (t >= 0 && ((j >> t) & 1)) { j ^= 1 << t; --t; }
            if (t >= 0) { j ^= 1 << t; --t; }
            rev[i] = j;
        }
    }
    void fft(base *a, int k) {
        build_rev(k);
        int n = 1 << k;
        for (int i = 0; i < n; i++) if (rev[i] > i) swap(a[i], a[rev[i]]);
        for (int l = 2, ll = 1; l <= n; l += l, ll += ll) {
            if (w[ll].re == 0 && w[ll].im == 0) {
                ld angle = M_PI / ll;
                base ww(cosl(angle), sinl(angle));
                if (ll > 1) for (int j = 0; j < ll; ++j) {
                    if (j & 1) w[ll + j] = w[(ll + j) / 2] * ww;
                    else w[ll + j] = w[(ll + j) / 2];
                }
                else w[ll] = base(1, 0);
            }
            for (int i = 0; i < n; i += l) for (int j = 0; j < ll; j++) {
                base v = a[i + j], u = a[i + j + ll] * w[ll + j];
                a[i + j] = v + u; a[i + j + ll] = v - u;
            }
        }
    }
    vector<int> mul(const vector<int>& a, const vector<int>& b) {
        int k = 1;
        int ABsize=(int)(a.size()) + (int)(b.size());
        while ((1 << k) < ABsize) ++k;
        int n = (1 << k);
        for (int i = 0; i < n; i++) f1[i] = base(0, 0);
        int Asize=(int)(a.size());
        int Bsize=(int)(b.size());
        for (int i = 0; i < Asize; i++) f1[i] = f1[i] + base(a[i], 0);
        for (int i = 0; i < Bsize; i++) f1[i] = f1[i] + base(0, b[i]);
        fft(f1, k);
        for (int i = 0; i < 1 + n / 2; i++) {
            base p = f1[i] + f1[(n - i) % n].conj();
            base _q = f1[(n - i) % n] - f1[i].conj();
            base q(_q.im, _q.re);
            f1[i] = (p * q) * 0.25;
            if (i > 0) f1[(n - i)] = f1[i].conj();
        }
        for (int i = 0; i < n; i++) f1[i] = f1[i].conj();
        fft(f1, k);
        vector<int> r(ABsize);
        int Rsize=(int)(r.size());
        for (int i = 0; i < Rsize; i++) {
            r[i] = ll(f1[i].re / n + 0.5);
        }
        return r;
    }
}
```

Derangement Series:

```
Dn = n * Dn-1 + (-1) ^ n   for n>=2
```

Dinic:

```
struct edge {
    int u, v;
    long long capacity, flow = 0;
    edge(int u, int v, long long capacity) : u(u), v(v), capacity(capacity) {}
};
struct Dinic {
    int s, t, id = 0, n;
    const long long INF = 0x3f3f3f3f3f3f3f3f;
    vector <edge> edges;
    vector <vector <int> > adj;
    vector <int> lvl, ptr;
    queue  <int> Q;
    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        lvl.resize(n);
        ptr.resize(n);
    }
    void addEdge(int u, int v, long long capacity) {
        edges.push_back(edge(u, v, capacity));
        edges.push_back(edge(v, u, 0));
        adj[u].push_back(id);
        adj[v].push_back(id + 1);
        id += 2;
    }
    int bfs() {
        fill(lvl.begin(), lvl.end(), -1);
        fill(ptr.begin(), ptr.end(),  0);
        Q.push(s);
        lvl[s] = 0;
        while(!Q.empty()) {
            int u = Q.front();
            Q.pop();
            for(int i = 0; i < adj[u].size(); i++) {
                if(lvl[edges[adj[u][i]].v] != -1 || edges[adj[u][i]].capacity -
edges[adj[u][i]].flow == 0)
                    continue;
                lvl[edges[adj[u][i]].v] = lvl[u] + 1;
                Q.push(edges[adj[u][i]].v);
            }
        }
        return lvl[t] != -1;
    }
    int dfs(int u, long long pushed) {
       if(!pushed || u == t)
            return pushed;
       for(int& i = ptr[u]; i < adj[u].size(); i++) {
            int idx = adj[u][i];
            int v = edges[idx].v;
            if(lvl[v] != lvl[u] + 1)
                continue;
            long long x = dfs(v, min(pushed, edges[idx].capacity -
edges[idx].flow));
            if(x > 0) {
                edges[idx ^ 0].flow += x;
                edges[idx ^ 1].flow -= x;
                return x;
            }
        }
        return 0;
    }
    long long maxFlow() {
        long long flow = 0, x;
        while(bfs())
           while(x = dfs(s, INF))
                flow += x;
        return flow;
    }
};
```

Closest Pair:

```
vector <pair <int, int> > v ;
int dist(pair <int, int> &a, pair <int, int> &b) {
    return (a.first - b.first) * (a.first - b.first) + (a.second - b.second) *
(a.second - b.second);
}
int res = 1e18;
int f(int l, int r)  {
    if(r - l <= 3) {
        for(int i = l; i <= r; i++)
            for(int j = l; j < i; j++)
                res = min(res, dist(v[i], v[j]));
        return res;
    }
    int mid = (l + r) >> 1;
    int x = f(l, mid);
    int y = f(mid, r);
    res = min(x, y);
    vector <pair<int, int>> a;
    for(int i = l; i <= r; i++) if(abs(v[i].first - v[mid].first) < res)
a.push_back(v[i]);
    sort(a.begin(), a.end(), [](pair <int, int> p, pair <int, int> q) {
        return p.second < q.second;
    });
    int z = ceil(sqrt(res));
    for(int i = 0; i < a.size(); i++)
        for(int j = i - 1; j >= 0 && a[i].second - a[j].second < z; j--)
            res = min(res, dist(a[i], a[j]));
    return res;
}
```

Suffix Automaton:

```
struct SA{
       struct state{
             int len, link, nxt[26];
             state() : len(0), link(-1) {
```

```
                        fill(begin(nxt), end(nxt), 0);
                }
        };
        vector <state> d; int last = 0;
        SA() {
                d.push_back(state());
        }
        void add(char ch) {
                int cur = d.size(); d.push_back(state());
                d[cur].len = d[last].len + 1;
                int p = last;
                while(p != -1 && !d[p].nxt[ch - 'a']) {
                        d[p].nxt[ch - 'a'] = cur;
                        p = d[p].link;
                }
                if(p == -1) {
                        d[cur].link = 0;
                } else {
                        int q = d[p].nxt[ch - 'a'];
                        if(d[q].len == d[p].len + 1) {
                                d[cur].link = q;
                        } else {
                                int clone = d.size(); d.push_back(d[q]);
                                d[clone].len = d[p].len + 1;
                                while(p != -1 && d[p].nxt[ch - 'a'] == q) {
                                        d[p].nxt[ch - 'a'] = clone;
                                        p = d[p].link;
                                }
                                d[cur].link = d[q].link = clone;
                        }
                }
                last = cur;
        }
};
```

EERTREE:

```
struct eertree{
        struct node{
                int nxt[26], len, link;
                node() : len(0), link(-1) {
                        fill(nxt, nxt + 26, 0);
                }
        };
        vector <node> t; int p;
        eertree() : p(2) {
                t = vector <node> (3);
                t[1].len = -1;
                t[2].len = 0;
                t[2].link = t[1].link = 1;
        }
        int add(int pos, string &str) {
```

```
			while(str[pos - t[p].len - 1] != str[pos]) p = t[p].link;
			int ch = str[pos] - 'a', x = t[p].link, r = 0;
			while(str[pos - t[x].len - 1] != str[pos]) x = t[x].link;
			if(!t[p].nxt[ch]) {
				r = 1;
				int y = t.size();
				t[p].nxt[ch] = y; t.push_back(node());
				t[y].len = t[p].len + 2;
				t[y].link = t[y].len == 1 ? 2 : t[x].nxt[ch];
			}
			p = t[p].nxt[ch];
			return r;
		}
};
```

Suffix Array:

```
struct SA{
	int n; vector <int> lcp, sa, rank;
	vector <vector <int> > t;
	SA() {}
	SA(string str) : n(str.size()) {
		vector <int> p(n), c(n), cnt(max(1 << 8, n), 0);
		for(int i = 0; i < n; i++) cnt[str[i]]++;
		for(int i = 1; i < (1 << 8); i++) cnt[i] += cnt[i - 1];
		for(int i = 0; i < n; i++) p[--cnt[str[i]]] = i;
		int cc = 1; c[p[0]] = 0;
		for(int i = 1; i < n; i++) {cc += str[p[i]] != str[p[i - 1]];
c[p[i]] = cc - 1;}
		vector <int> pn(n), cn(n);
		for(int h = 0; (1 << h) < n; h++) {
			for(int i = 0; i < n; i++) pn[i] = (p[i] - (1 << h) + n) %
n;
			fill(cnt.begin(), cnt.begin() + cc, 0);
			for(int i = 0; i < n; i++) cnt[c[pn[i]]]++;
			for(int i = 1; i < cc; i++) cnt[i] += cnt[i - 1];
			for(int i = n - 1; i >= 0; i--) p[--cnt[c[pn[i]]]] = pn[i];
			cc = 1; cn[p[0]] = 0;
			for(int i = 1; i < n; i++) {
				pair <int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) %
n]}, prv = {c[p[i - 1]], c[(p[i - 1] + (1 << h)) % n]};
				cc += (prv != cur);
				cn[p[i]] = cc - 1;
			}
			c.swap(cn);
		}
		sa = p; rank.resize(n); lcp.resize(n, 0);
		for(int i = 0; i < n; i++) rank[sa[i]] = i;
		int k = 0;
		for(int i = 0; i < n; i++) {
			if(rank[i] == n - 1) {k = 0; continue;}
			int j = p[rank[i] + 1];
```

```
                    while(i + k < n && j + k < n && str[i + k] == str[j + k])
++k;
                    lcp[rank[i]] = k;
                    if(k) --k;
                }
            }
            void build_rmq() {
                int l = 32 - __builtin_clz(n);
                t = vector <vector <int> > (l, vector <int> (n, 0));
                for(int i = 0; i < n; i++) t[0][i] = lcp[i];
                for(int i = 1; i < l; i++) {
                    for(int j = 0; j + (1 << i) - 1 < n; j++) {
                        t[i][j] = min(t[i - 1][j], t[i - 1][j + (1 << (i -
1))]);
                    }
                }
            }
            int query(int l, int r) {
                int h = 31 - __builtin_clz(r - l + 1);
                return min(t[h][l], t[h][r - (1 << h) + 1]);
            }
            int find_left(int i, int k) {
                int l = 1, r = rank[i] - 1, j = rank[i];
                while(l <= r) {
                    int mid = (l + r) >> 1;
                    if(query(mid, rank[i] - 1) >= k) j = mid, r = mid - 1;
                    else l = mid + 1;
                }
                return j;
            }
            int find_right(int i, int k) {
                int l = rank[i] + 1, r = n - 1, j = rank[i];
                while(l <= r) {
                    int mid = (l + r) >> 1;
                    if(query(rank[i], mid - 1) >= k) j = mid, l = mid + 1;
                    else r = mid - 1;
                }
                return j;
            }
}sa;
```

Aho:

```
struct AC {
    static const int K = 26;
    struct node {
        int nxt[K], link, leaf, par;
        char pch;
        node(int par = -1, char pch = '$') : par(par), pch(pch) {
            fill(begin(nxt), end(nxt), -1);
            leaf = false;
            link = -1;
        }
    };
    vector <node> t;
    vector <vector <int> > ad;
    vector <int> st, en, mp;
    int dt = 0;
    AC() {
        mp = vector <int> (1 << 8, 0);
        for(char ch = 'a'; ch <= 'z'; ++ch) mp[ch] = ch - 'a';
        t.resize(1);
    }
    int add(string str) {
        int ptr = 0;
        for(auto ch: str) {
            if(t[ptr].nxt[mp[ch]] < 0) {
                t[ptr].nxt[mp[ch]] = t.size();
                t.push_back(node(ptr, ch));
            }
            ptr = t[ptr].nxt[mp[ch]];
        }
        t[ptr].leaf = true;
        return ptr;
    }
    int get_link(int v) {
        if(t[v].link == -1) {
            t[v].link = (!v || !t[v].par) ? 0 : go(get_link(t[v].par),
t[v].pch);
        }
        return t[v].link;
    }
    int go(int v, char ch) {
        if(t[v].nxt[mp[ch]] < 0) {
            t[v].nxt[mp[ch]] = !v ? 0 : go(get_link(v), ch);
        }
        return t[v].nxt[mp[ch]];
    }
    void dfs(int v) {
        st[v] = ++dt;
        for(int u: ad[v])
            dfs(u);
        en[v] = dt;
    }
    void calc() {
        int k = t.size();
        st.resize(k);
        en.resize(k);
        ad.resize(k);
        for(int i = 1; i < k; i++) ad[get_link(i)].push_back(i);
        dfs(0);
    }
```

```
}ac;
```

Bipartite Matching:

```
struct bpm{
	const int inf = 0x3f3f3f3f;
	vector <vector <int> > G;
	vector <int> match, dist;
	int n, m;
	bpm(int n, int m) : n(n), m(m) {
		G.resize(n + 1);
		match.resize(n + m + 1, 0);
		dist.resize(n + 1);
	}
	void add_edge(int i, int j) {
		G[i].push_back(n + j);
	}
	bool bfs() {
		queue <int> Q;
		fill(dist.begin(), dist.end(), inf);
		for(int i = 1; i <= n; i++) if(!match[i]) {
			dist[i] = 0;
			Q.push(i);
		}
		while(Q.size()) {
			int u = Q.front(); Q.pop();
			if(!u) continue;
			for(int v: G[u]) if(dist[match[v]] == inf) {
				dist[match[v]] = 1 + dist[u];
				Q.push(match[v]);
			}
		}
		return dist[0] != inf;
	}
	bool dfs(int u) {
		if(!u) return true;
		for(int x: G[u]) {
			int v = match[x];
			if(dist[v] == dist[u] + 1 && dfs(v)) {
				match[u] = x;
				match[x] = u;
				return true;
			}
		}
		dist[u] = inf;
		return false;
	}
	int get() {
		int ans = 0;
		while(bfs()) {
			for(int i = 1; i <= n; i++) if(!match[i] && dfs(i)) ++ans;
```

```
            }
            return ans;
        }
};
```
BCC:
```
vector <int> adj[N];
int dis[N], low[N], col[N], ins[N], t, id;
stack <int> stk;
void BCC(int u, int p) {
    dis[u] = low[u] = ins[u] = ++t; stk.push(u);
    for(int v: adj[u]) {
        if(!dis[v]) {
            BCC(v, u);
            low[u] = min(low[u], low[v]);
        } else if(ins[v] && p != v)
            low[u] = min(low[u], dis[v]);
    }
    if(dis[u] == low[u]) {
        ++id; int v;
        do {
            v = stk.top(); stk.pop();
            ins[v] = 0; col[v] = id;
        } while(v != u);
    }
}
```
Dynamic Connectivity:
```
struct DSU{
    vector <int> p, sz, stk;
    DSU(int n) {
        p.resize(n + 1); sz.resize(n + 1, 1);
        for(int i = 1; i <= n; i++) p[i] = i;
    }
    int find(int x) {
        return (p[x] == x) ? p[x] : find(p[x]);
    }
    int merge(int x, int y) {
        if((x = find(x)) ^ (y = find(y))) {
            if(sz[x] > sz[y]) swap(x, y);
            p[x] = y;
            sz[y] += sz[x];
            stk.push_back(x);
            return 1;
        }
        return 0;
    }
    void roll_back(int t) {
        while(stk.size() > t) {
            int x = stk.back();
            stk.pop_back();
            sz[p[x]] -= sz[x];
            p[x] = x;
        }
    }
};
void update(int v, int l, int r, int L, int R, pair <int, int> e) {
    if(l > R || r < L) return;
    if(l >= L && r <= R) {
        edges[v].push_back(e);
        return;
    }
    int mid = (l + r) / 2;
    update(v * 2, l, mid, L, R, e);
    update(v * 2 + 1, mid + 1, r, L, R, e);
}
void build(int v, int l, int r, DSU &d) {
    int tm = d.stk.size();
    for(auto e: edges[v]) d.merge(e.first, e.second);
    if(l == r) {
        for(auto Q: query[l]) ans[Q.second] = d.sz[d.find(Q.first)];
    } else {
        int mid = (l + r) / 2;
        build(v * 2, l, mid, d);
        build(v * 2 + 1, mid + 1, r, d);
    }
    d.roll_back(tm);
}
```

MCMF:

```
const int mxN = 110;
const int inf = 2e9;
struct Edgee {
  int to, cost, cap, flow, backEdge;
};
struct MCMF {
  int s, t, n;
  vector<Edgee> g[mxN];
  MCMF(int _s, int _t, int _n) {
    s = _s, t = _t, n = _n;
  }
  void addEdge(int u, int v, int cost, int cap) {
    Edgee e1 = { v, cost, cap, 0, g[v].size() };
    Edgee e2 = { u, -cost, 0, 0, g[u].size() };
    g[u].push_back(e1); g[v].push_back(e2);
  }
  pair<int, int> minCostMaxFlow() {
    int flow = 0, cost = 0;
    vector<int> state(n), from(n), from_edge(n), d(n);
    deque<int> q;
    while (true) {
      for (int i = 0; i < n; i++)
        state[i] = 2, d[i] = inf, from[i] = -1;
      state[s] = 1; q.clear(); q.push_back(s); d[s] = 0;
      while (!q.empty()) {
        int v = q.front(); q.pop_front(); state[v] = 0;
        for (int i = 0; i < (int) g[v].size(); i++) {
          Edgee e = g[v][i];
          if (e.flow >= e.cap || d[e.to] <= d[v] + e.cost)
            continue;
          int to = e.to; d[to] = d[v] + e.cost;
          from[to] = v; from_edge[to] = i;
          if (state[to] == 1) continue;
          if (!state[to] || (!q.empty() && d[q.front()] > d[to]))
            q.push_front(to);
          else q.push_back(to);
          state[to] = 1;
        }
      }
      if (d[t] == inf) break;
      int it = t, addflow = inf;
      while (it != s) {
        addflow = min(addflow,
            g[from[it]][from_edge[it]].cap
            - g[from[it]][from_edge[it]].flow);
        it = from[it];
      }
      it = t;
      while (it != s) {
        g[from[it]][from_edge[it]].flow += addflow;
        g[it][g[from[it]][from_edge[it]].backEdge].flow -= addflow;
        cost += g[from[it]][from_edge[it]].cost * addflow;
        it = from[it];
      }
      flow += addflow;
    }
    return {cost,flow};
  }
};
```

HLD:

```
int curPos,depth[],headofCurrentChain[],heavyChild[],parent[],pos[];
int DFS(int cur) {
    int childSize,size=1,maxChildSize=0;
    for(int x : adj[cur]) {
        if(x ^ parent[cur]) {
        parent[x]=cur; depth[x]=depth[cur]+1; childSize=DFS(x); size+=childSize;
if(childSize > maxChildSize) { heavyChild[cur]=x; maxChildSize=childSize; }
         } } return size; }
void Decompose(int cur,int headNode) {
    pos[cur]=++curPos; headofCurrentChain[cur]=headNode;
if(heavyChild[cur])Decompose(heavyChild[cur],headNode);
```

```
    for(int x : adj[cur]) {
        if(x != parent[cur]  &&  x != heavyChild[cur])Decompose(x,x);
    } }
void HeavyLightDecomposition() { DFS(1); curPos=0; Decompose(1,1); }
long long HLDQuery(int x,int y) {
    long long ans=0,curVal;
    while(headofCurrentChain[x] != headofCurrentChain[y]) {
if(depth[headofCurrentChain[x]] > depth[headofCurrentChain[y]])swap(x,y);
curVal=SegmentTreeQuery(1,1,n,pos[headofCurrentChain[y]],pos[y]); ans+=curVal;
y=parent[headofCurrentChain[y]]; }
   if(depth[x] > depth[y])swap(x,y);//x-lca of given (x,y)
curVal=SegmentTreeQuery(1,1,n,pos[x],pos[y]); return ans+curVal; }
```

BIT:

```
long long sum[2][N+2];
void Update(int num,int idx,long long val)
{
    while(idx <= N) sum[num][idx]+=val; idx+=idx & (-idx);
}
long long Query(int num,int idx)
{
    long long ans=0;
    while(idx)ans+=sum[num][idx]; idx-=idx & (-idx);
    return ans;
}
void RangeUpdate(int l,int r,long long val)
{Update(0,l,val),Update(0,r+1,-val),Update(1,l,val*(l-1)),Update(1,r+1,-val*r);}
long long PrefixSum(int idx) return Query(0,idx)*idx-Query(1,idx);
long long RangeQuery(int l,int r) { return PrefixSum(r)-PrefixSum(l-1); }
```

Segment Tree:

```
const int inf = 0x3f3f3f3f, N = 3e5;
struct RMQ {
        vector <int> t;
        int n;
        RMQ(int n) : n(n) {
                t.resize(n << 1);
        }
        void build(int a[]) {
                for(int i = 0; i < n; i++) t[n + i] = a[i];
                for(int i = n - 1; i > 0; --i) t[i] = min(t[i << 1], t[i << 1 |
1]);
        }
        void modify(int p, int v) {
                for(t[p += n] = v; p > 1; p >>= 1) t[p >> 1] = min(t[p], t[p ^
1]);
        }
        int query(int l, int r) { int res = inf;
                for(l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
                        if(l & 1) res = min(res, t[l++]);
                        if(r & 1) res = min(res, t[--r]);
                }
                return res;
            }
};
```

Treap:

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
struct treap {
    struct node{
        int value = 0, priority = rng(), size = 1;
        node *l = NULL, *r = NULL;
        node(int x) : value(x) {}
    } *root = NULL;

    int size(node *v) {
        return !v ? 0 : v -> size;
    }
    void recalc(node *v) {
        if(!v) return;
        v -> size = size(v -> l) + size(v -> r) + 1;
    }
    node *merge(node *p, node *q) {
        if(!p || !q) return !p ? q : p;
        if(p -> priority < q -> priority) {
            p -> r = merge(p -> r, q);
            recalc(p);
            return p;
        } else {
            q -> l = merge(p, q -> l);
            recalc(q);
            return q;
        }
    }
    pair <node *, node *> split(node *v, int cnt) {
        if(!v) return {NULL, NULL};
        if(size(v -> l) >= cnt) {
            auto [p, q] = split(v -> l, cnt);
            v -> l = q;
            recalc(v);
            return {p, v};
        } else {
            auto [p, q] = split(v -> r, cnt - size(v -> l) - 1);
            v -> r = p;
            recalc(v);
            return {v, q};
        }
    }
    void insert(int x) {
        root = merge(root, new node(x));
    }
};
```

2-SAT:

```
// 0-based indexing
vector<int>adj[N];
int getNode(int x)   ///Convert a choice to node
{
    int p=abs(x);
    p=(p-1)*2;
    if(x<0)p^=1;
    return p;
}
int getNodeVal(int x)   ///Convert a node to choice
{
    int p=1;
    if(x&1) p=-1, x-=1;
    x/=2, x++, p*=x;
    return p;
}
///Always pass getNode() value to the folloing function
///For example if we want to mustTrue 5 then the call will be
mustTrue(getNode(5))
void mustTrue(int a)   /// A is True
{
    adj[a^1].emplace_back(a);
}
void xorClause(int a, int b)   /// A ^ B clause
{
    //!a->b !b->a a->!b b->!a
    adj[a^1].emplace_back(b);
    adj[a].emplace_back(b^1);
    adj[b^1].emplace_back(a);
    adj[b].emplace_back(a^1);
}
void orClause(int a, int b)   /// A || B clause
{
    //!a->b !b->a
    adj[a^1].emplace_back ( b );
    adj[b^1].emplace_back ( a );
}
void andClause(int a, int b)   /// A && B clause
{
    mustTrue(a);
    mustTrue(b);
}
/// Out of all possible option, at most one is true
void atMostOneClause(int a[], int n, bool flag)
{
    int i,j;
    if(!flag)   /// At most one can be false
    {
        for(i=0; i<n; i++)a[i] = a[i] ^ 1;
    }
    for(i = 0; i<n; i++)
    {
        for(j = i+1; j<n; j++)
        {
            orClause( a[i] ^ 1, a[j] ^ 1 ); /// !a || !b both being true not
allowed
        }
    }
}
///SCC variables
stack<int>scc;
int component[N],disc[N],low[N],Time, scc_count;
///2-SAT variables
deque<int>sat;
int isSAT[N];
void allClear(int n)
{
    n <<= 1;
    for(int i=0; i < n; i++)
    {
        isSAT[i]=-1;
        adj[i].clear();
        low[i]=disc[i]=component[i]=0;
    }
    Time=0;
    scc_count=0;
    while(!scc.empty())scc.pop();
    sat.clear();
}

void tarjan_SCC(int u)
{
    disc[u]=low[u]=++Time;
    scc.emplace(u);
    for(int i=0; i<adj[u].size(); i++)
    {
        int v=adj[u][i];
        if(disc[v]==0)tarjan_SCC(v);
        if(disc[v]!=-1)low[u]=min(low[u],low[v]);
    }
    if(low[u] == disc[u])
    {
        scc_count++;
        int v;
        do
        {
            v=scc.top();
            scc.pop();
            sat.emplace_back(v);
            component[v]=scc_count;
            disc[v]=-1;
        }
        while(v != u);
    }
}

bool checkSAT(int const& n)
{
    int i,x;
    for(i=1;i<=n;i++)
    {
        x=getNode(i);
        if(!disc[x])tarjan_SCC(x);
        x=getNode(-i);
        if(!disc[x])tarjan_SCC(x);
    }
    while(!sat.empty())   ///Assigning valid values to candidates
    {
        int x=sat.front();
        sat.pop_front();
        if(isSAT[x]==-1)
        {
            isSAT[x]=1;
            x=getNode(-getNodeVal(x)); ///Getting opposite value
            isSAT[x]=0;
        }
    }

    ///Checking whether satisfiability is possible or not
    bool check=1;
    for(i=1; i<=n && check; i++)
    {
        check=(component[getNode(i)] != component[getNode(-i)]);
    }
    return check;
}
```

NTT:

```
struct base
{
    double x, y;
    base()
    {
        x = y = 0;
    }
    base(double x, double y): x(x), y(y) { }
};
inline base operator + (base a, base b)
{
    return base(a.x + b.x, a.y + b.y);
}
inline base operator - (base a, base b)
{
    return base(a.x - b.x, a.y - b.y);
}
inline base operator * (base a, base b)
{
    return base(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);
}
inline base conj(base a)
{
    return base(a.x, -a.y);
}

int lim = 1;
vector<int> rev = {0, 1};
const double PI = acosl(- 1.0);
vector<base> roots = {{0, 0}, {1, 0}};

void EnsureBase(int p)
{
    if(p <= lim) return;
    rev.resize(1 << p), roots.resize(1 << p);
    for(int i = 0; i < (1 << p); i++) rev[i] = (rev[i >> 1] >> 1) + ((i & 1)  <<
(p - 1));
    while(lim < p)
    {
        double angle = 2 * PI / (1 << (lim + 1));
        for(int i = 1 << (lim - 1); i < (1 << lim); i++)
        {
            roots[i << 1] = roots[i];
            double angle_i = angle * (2 * i + 1 - (1 << lim));
            roots[(i << 1) + 1] = base(cos(angle_i), sin(angle_i));
        }
        lim++;
    }
}
void FFT(vector<base> &a, int n = -1)
{
    if(n == -1) n = a.size();
    int zeros = __builtin_ctz(n);
    EnsureBase(zeros);
    int shift = lim - zeros;
    for(int i = 0; i < n; i++) if(i < (rev[i] >> shift)) swap(a[i], a[rev[i] >>
shift]);
```

```
    for(int k = 1; k < n; k <<= 1)
    {
        for(int i = 0; i < n; i += 2 * k)
        {
            for(int j = 0; j < k; j++)
            {
                base z = a[i + j + k] * roots[j + k];
                a[i + j + k] = a[i + j] - z;
                a[i + j] = a[i + j] + z;
            }
        }
    }
}

vector<int> Multiply(vector<int> &a, vector<int> &b, int eq = 0)
{
    int need = a.size() + b.size() - 1;
    int p = 0;
    while((1 << p) < need) p++;
    EnsureBase(p);
    int sz = 1 << p;
    vector<base> A, B;
    if(sz > (int)A.size()) A.resize(sz);
    for(int i = 0; i < (int)a.size(); i++)
    {
        int x = (a[i] % mod + mod) % mod;
        A[i] = base(x & ((1 << 15) - 1), x >> 15);
    }
    fill(A.begin() + a.size(), A.begin() + sz, base{0, 0});
    FFT(A, sz);
    if(sz > (int)B.size()) B.resize(sz);
    if(eq) copy(A.begin(), A.begin() + sz, B.begin());
    else
    {
        for(int i = 0; i < (int)b.size(); i++)
        {
            int x = (b[i] % mod + mod) % mod;
            B[i] = base(x & ((1 << 15) - 1), x >> 15);
        }
        fill(B.begin() + b.size(), B.begin() + sz, base{0, 0});
        FFT(B, sz);
    }
    double ratio = 0.25 / sz;
    base r2(0,  - 1), r3(ratio, 0), r4(0,  - ratio), r5(0, 1);
    for(int i = 0; i <= (sz >> 1); i++)
    {
        int j = (sz - i) & (sz - 1);
        base a1 = (A[i] + conj(A[j])), a2 = (A[i] - conj(A[j])) * r2;
        base b1 = (B[i] + conj(B[j])) * r3, b2 = (B[i] - conj(B[j])) * r4;
        if(i != j)
        {
            base c1 = (A[j] + conj(A[i])), c2 = (A[j] - conj(A[i])) * r2;
            base d1 = (B[j] + conj(B[i])) * r3, d2 = (B[j] - conj(B[i])) * r4;
            A[i] = c1 * d1 + c2 * d2 * r5;
            B[i] = c1 * d2 + c2 * d1;
        }
        A[j] = a1 * b1 + a2 * b2 * r5;
        B[j] = a1 * b2 + a2 * b1;
    }
    FFT(A, sz);
    FFT(B, sz);
    vector<int> res(need);
    for(int i = 0; i < need; i++)
    {
        long long aa = A[i].x + 0.5;
        long long bb = B[i].x + 0.5;
        long long cc = A[i].y + 0.5;
        res[i] = (aa + ((bb % mod) << 15) + ((cc % mod) << 30))%mod;
    }
    return res;
}

vector<int> Pow(vector<int>& a, int p)
{
    vector<int> res;
    res.emplace_back(1);
    while(p)
    {
        if(p & 1) res = Multiply(res, a);
        a = Multiply(a, a, 1);
        p >>= 1;
    }
    return res;
}
```

Ncr sum optimization with limit:

$$f_M(N) = \sum_{i=0}^{\min(N,M)} \binom{N}{i} \quad -> \quad f_M(N+1) = 2 \times f_M(N) - \binom{N}{M}$$

Distribution:

```
distinct m ball in distinct n box -> n * ( T(m-1, n-1) + T(m-1, n))
distinct m ball in identical  n box ->  S(m-1, n-1) +  n * S(m-1, n)
Box in circular / Flag pole with identical pole -> S(m-1, n-1)+(m-1)*S(m-1,n)
identical m ball in identical n box ->. P(m-1, n-1) +  n * P(m-n, n)
identical m ball in identical n box -> (m-1)c(n-1)
```

Sum of LCM:

$$SUM = \frac{n}{2}(\sum_{d|n}(\phi(d) \times d) + 1)$$

Eular:

There is a less known version of the last equivalence, that allows computing $x^n \bmod m$ efficiently for not coprime x and m. For arbitrary x, m and $n \geq \log_2 m$:

$$x^n \equiv x^{\phi(m) + [n \bmod \phi(m)]} \mod m$$

Linear Convex-hull trick:

```
using ii = pair <int, int>;
long double intersect(ii &a, ii &b) {
    return (long double)(a.second - b.second) / (b.first - a.first);
}

struct CHT {
    deque <ii> dq;
    void add(ii p) {
        while(dq.size() >= 2 and intersect(dq[dq.size() - 2], dq.back()) >= 
intersect(dq[dq.size() - 2], p))
            dq.pop_back();
        dq.push_back(p);
    }

    int query(int x) {
        while(dq.size() >= 2 and dq[0].first * x + dq[0].second >= dq[1].first * x 
+ dq[1].second)
            dq.pop_front();
        return dq[0].first * x + dq[0].second;
    }

    void clear() {
        while(dq.size()) dq.pop_back();
    }
};
```

Miscellaneous:

- **PBDS:**

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template<typename temp>using ordered_set = tree<temp, null_type, less<temp>, 
rb_tree_tag,tree_order_statistics_node_update>;
```

- **Unordered Map:**

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15; x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
}
size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
    }
};
unordered_map<long long, int, custom_hash> safe_map;
Manacher's algo
vector<int> manacher_odd(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 1, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}
//working with parities
vector<int> manacher(string s) {
    string t;
    for(auto c: s) {
        t += string("#") + c;
    }
    auto res = manacher_odd(t + "#");
    return vector<int>(begin(res) + 1, end(res) - 1);
}

//
```

$C_0 = 1 \; and \; C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \; for \; n \geq 0$

```
#include <iostream>
using namespace std;

// A recursive function to find nth catalan number
unsigned long int catalan(unsigned int n)
{
    // Base case
    if (n <= 1)
```

```
        return 1;

    // catalan(n) is sum of
    // catalan(i)*catalan(n-i-1)
    unsigned long int res = 0;
    for (int i = 0; i < n; i++)
        res += catalan(i) * catalan(n - i - 1);

    return res;
}

// Driver code
int main()
{
    for (int i = 0; i < 10; i++)
        cout << catalan(i) << " ";
    return 0;
}
```