Contents

Data Structures

Sparse Table

```
int a[N], st[LG + 1][N];
void preprocess() {
    for (int i = 1; i <= n; ++i) st[0][i] = a[i];
    for (int j = 1; j <= LG; ++j)
        for (int i = 1; i + (1 << j) - 1 <= n; ++i)
            st[j][i] = min(st[j - 1][i], st[j - 1][i +
                (1 << (j - 1))]);
}

int query(int l, int r) {
    int k = __lg(r - l + 1);
    return min(st[k][l], st[k][r - (1 << k) + 1]);
}

//query sum:
int querySum(int l, int r) {
    int len = r - l + 1;
    int sum = 0;
    for (int j = 0; (1 << j) <= len; ++j)
        if (len >> j & 1) {
            sum = sum + st[j][l];
            l = l + (1 << j);
        }
    return sum;
}
```

Fenwick Tree

```
void update(int i, int val){
    for (; i <= n; i += i & -i) bit[i] += val;
}

int get(int i){
    int res = 0;
    for (; i; i -= i & -i) res += bit[i];
    return res;
}
```

Segment Tree

```
struct Segment_tree{
  int st[4 * N], lazy[4 * N];

    void apply(int id, int c){
        update(st[id], c);
        update(lazy[id], c);
    }

  void down(int id, int l, int r){
    int c = lazy[id]; lazy[id] = 0;
        apply(id << 1, c); apply (id << 1 | 1, c);
  }

  void build(int id, int l, int r){
    if (l == r){
      st[id] = a[l];
      return;
    }

    int mid = (l + r) >> 1;
    build(id << 1, l, mid);
    build(id << 1 | 1, mid + 1, r);

    st[id] = merge(st[id << 1], st[id << 1 | 1]);
  }

  void update(int id, int l, int r, int u, int v, int
      x){
    if (r < u || v < l) return;
    if (u <= l && r <= v){
      apply(id, x);
      return;
    }

    down(id, l, r);

    int mid = (l + r) >> 1;
    update(id << 1, l, mid, u, v, x);
    update(id << 1 | 1, mid + 1, r, u, v, x);

    st[id] = merge(st[id << 1], st[id << 1 | 1]);
  }

  int get(int id, int l, int r, int u, int v){
    if (r < u || v < l) return -INF;
    if (u <= l && r <= v) return st[id];

    down(id, l, r);

    int mid = (l + r) >> 1;
    return merge(get(id << 1, l, mid, u, v), get(id
        << 1 | 1, mid + 1, r, u, v));
  }
} ST;
```

Sigma Tree

```
struct Sigma_Tree{
    int st[2 * N];

    void init(){
        For(i, 1, n) st[i + n - 1] = a[i];
        ForD(i, n - 1, 1) st[i] = merge(st[i << 1],
            st[i << 1 | 1]);
    }

  void update(int p, int val){
    p += n - 1;
    st[p] = val;
    for (; p > 1; p >>= 1) st[p >> 1] = merge(st[p],
        st[p ^ 1]);
  }

    int get(int l, int r){
        int res = 0;
        for (l += n - 1, r += n - 1; l <= r; l >>= 1,
             r >>= 1){
            if (l & 1) res = merge(res, st[l++]);
            if (!(r & 1)) res = merge(res, st[r--]);
        }
        return res;
    }
} ST;
```

Persistent Segment Tree

```
struct Node {
    int left, right; // ID of left child & right
        child
    long long ln; // Max value of node
    Node() {}
    Node(long long ln, int left, int right) : ln(ln),
         left(left), right(right) {}
```

```
} it[N]; // Each node has a position in this array,
    called ID
int nNode;

int ver[N]; // ID of root in each version

// Update max value of a node
inline void refine(int cur) {
    it[cur].ln = max(it[it[cur].left].ln, it[it[cur].
        right].ln);
}

// Update a range, and return new ID of node
int update(int l, int r, int u, int x, int oldId) {
    if (l == r) {
        ++nNode;
        it[nNode] = Node(x, 0, 0);
        return nNode;
    }

    int mid = (l + r) >> 1;
    int cur = ++nNode;

    if (u <= mid) {
        it[cur].left = update(l, mid, u, x, it[oldId
            ].left);
        it[cur].right = it[oldId].right;
        refine(cur);
    }
    else {
        it[cur].left = it[oldId].left;
        it[cur].right = update(mid+1, r, u, x, it[
            oldId].right);
        refine(cur);
    }

    return cur;
}

// Get max of range. Same as usual IT
int get(int nodeId, int l, int r, int u, int v) {
    if (v < l || r < u) return -1;
    if (u <= l && r <= v) return it[nodeId].ln;

    int mid = (l + r) >> 1;
    return max(get(it[nodeId].left, l, mid, u, v),
        get(it[nodeId].right, mid+1, r, u, v));
}


// When update:
    ++nVer;
    ver[nVer] = update(1, n, u, x, ver[nVer-1]);

// When query:
    res = get(ver[t], 1, n, u, v);
```

Hash Map

```
//faster than unordered_map
struct hash_map {
    const static int SZ = 2e4 + 9;
    int nxt[SZ >> 3], val[SZ >> 3];
    int key[SZ >> 3];
    int h[SZ + 5], cnt;
    vector<int>vec;

    void clear(){
        for (int i : vec) h[i] = 0;
        for (int i = 1; i <= cnt; i++)
            val[i] = nxt[i] = 0, key[i] = 0;

        vec.clear();
        cnt = 0;
    }

    int hash(int u) {
        return u % SZ;
    }

    int &operator[](int u) {
        int x = hash(u);

        for (int i = h[x]; i; i = nxt[i])
            if (key[i] == u) return val[i];

        if (!h[x]) vec.push_back(x);

        ++cnt;
        key[cnt] = u;
        val[cnt] = 0;
        nxt[cnt] = h[x];
        h[x] = cnt;
        return val[cnt];
    }

    int qry(int u) {
        int x = hash(u);
        for (int i = h[x]; i; i = nxt[i])
            if (key[i] == u) return val[i];

        return 0;
    }
} hs;
```

String

Trie 1

```
struct node{
  node *g[26];
  node(){
    rep(i, 26) g[i] = NULL;
  }
} *root = new node();

void Insert(string s){
  node *p = root;
  for (char t: s){
    if (p->g[t - 'a'] == NULL)
      p->g[t - 'a'] = new node();

    p = p->g[t - 'a'];
  }
}
```

Trie 2

```
int nNode = 0;
int g[N][26];

void Insert(string s){
    int p = 0;
    for (char t: s){
        if (!g[p][t - 'a']) g[p][t - 'a'] = ++nNode;
        p = g[p][t - 'a'];
    }
}
```

Hash

```
ll getHashT(int i, int j) {
    return (hashT[j] - mul(hashT[i - 1], POW[j - i +
        1]) + MOD) % MOD;
}

// Precalculate base^i
for (int i = 1; i <= lenT; i++)
    POW[i] = (POW[i - 1] * base) % MOD;

// Calculate hash value of T[1..i]
for (int i = 1; i <= lenT; i++)
    hashT[i] = (hashT[i - 1] * base + (T[i] - 'a' +
        1)) % MOD;
```

KMP

```
//prefix function: length of the longest prefix of
    the substring s[1..i] that is also a suffix of
    this same substring
int k = 0;
For(i, 2, n){ //1-indexed
    while (k && s[k + 1] != s[i]) k = kmp[k];
    kmp[i] = (s[k + 1] == s[i]) ? ++k : 0;
}
```

Manacher

```
vector<int> manacher_odd(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 0, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = 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);
}

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);
}
```

Aho - Corasick

```
namespace Trie{
  struct Node{
    int child[26], p = -1, cnt = 0;
    char pch;
    int link = -1, go[26];
    Node(int p = -1, char ch = '#'): p(p), pch(ch){
      fill(begin(child), end(child), -1);
          fill(begin(go), end(go), -1);
    }
  };
  vector<Node> g(1);

  void add(string s){
    int v = 0;
    for (char t: s){
      int c = t - 'a';
      if (g[v].child[c] == -1){
        g[v].child[c] = g.size();
        g.emplace_back(v, t);
      }
      v = g[v].child[c];
    }
    g[v].cnt++;
  }

  int go(int v, char c);

  int get_link(int v){
    if (g[v].link == -1){
      if (!v || !g[v].p) g[v].link = 0;
      else g[v].link = go(get_link(g[v].p), g[v].pch)
          ;
    }
    return g[v].link;
  }

  int go(int v, char t){
    int c = t - 'a';
    if (g[v].go[c] == -1){
      if (g[v].child[c] != -1) g[v].go[c] = g[v].
          child[c];
      else g[v].go[c] = (v == 0) ? 0 : go(get_link(v)
          , t);
    }
    return g[v].go[c];
  }
}
```

Aho - Corasick (BFS)

```
struct trie{
  struct Node{
    Node *child[26], *link;
    int cnt = 0;
    Node(){
      cnt = 0;
      rep(i, 26) child[i] = NULL;
      link = NULL;
    }
  } *root = new Node();


  void add(string &s){
    Node* p = root;
    for (char &t: s){
      int c = t - 'a';
      if (p->child[c] == NULL) p->child[c] = new Node
          ();;
      p = p->child[c];
    }
    p->cnt++;
  }

  void AhoCorasick(){
    root->link = root;
    queue<Node*> q; q.push(root);
    while (!q.empty()){
      Node* p = q.front(); q.pop();
      rep(i, 26) if (p->child[i]){
```

```
        Node* k = p->link;
        while (k != root && k->child[i] == NULL) k =
            k->link;

        if (k->child[i] && k != p) p->child[i]->link
            = k->child[i];
        else p->child[i]->link = root;

        p->child[i]->cnt += p->child[i]->link->cnt;
        q.push(p->child[i]);
      }
    }
  }
};
```

SQRT Decomposition

MO

```
struct query{
  int l, r, id;
}

bool cmp(const query &a, const query &b){
    if(a.l / S != b.l / S) return a.l < b.l;

    if((a.l / S) & 1)
        return a.r < b.r;
    else
        return a.r > b.r
}

long long res = 0;

void update(...);

//////////////////////////////
sort(q + 1, q + Q + 1, cmp);

int l = 1, r = 0;
for(int i = 1; i <= Q; i++){
    while(l < q[i].l) update(a[l++], ...);
    while(l > q[i].l) update(a[--l], ...);
    while(r < q[i].r) update(a[++r], ...);
    while(r > q[i].r) update(a[r--], ...);
    ans[q[i].id] = res;
}
```

Graph

Joint and Bridge

```
void dfs(int u, int pre) {
    int child = 0;
    num[u] = low[u] = ++timer;
    for (int v: g[u]) {
        if (v == pre) continue;
        if (!num[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] == num[v]) bridge++;
            child++;
            if (u == pre){
                if (child > 1) joint[u] = true;
            }
            else if (low[v] >= num[u]) joint[u] = true
                ;
        }
        else low[u] = min(low[u], num[v]);
    }
}
```

SCC

```
void dfs(int u) {
    num[u] = low[u] = ++timer;
    st.push(u);
    for (int v : g[u]) {
        if (!num[v]){
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
        else low[u] = min(low[u], num[v]);
    }
    if (low[u] == num[u]) {
        scc++;
        int v;
        do {
            v = st.top();
            st.pop();
            num[v] = INF;
        }
        while (v != u);
    }
}
```

Topology Sort 1

```
//u -> v
//++deg[v]
for (int u = 1; u <= n; ++u)
    if (!deg[u]) q.push(u);

while (!q.empty()) {
    int u = q.front();
    q.pop();
    topo.push_back(u);
    for (auto v : g[u]) {
        deg[v]--;
        if (!deg[v]) q.push(v);
    }
}
```

Topology Sort 2

```
void dfs(int u) {
    visit[u] = 1;
    for (auto v : g[u]) {
        assert(visit[v] != 1);
        //graph contains a cycle
        if (!visit[v]) dfs(v);
    }
    topo.push(u);
    visit[u] = 2;
}
```

Max Flow

```
struct edge{
  int to, rev, flow, cap;
};

void add_edge(int u, int v, int cap){
  edge e1 = {v, sz(g[v]), 0, cap};
    edge e2 = {u, sz(g[u]), 0 , 0};
    g[u].pb(e1); g[v].pb(e2);
}
```

```
bool bfs(){
  memset(dist, 0x3f, sizeof dist);
  queue<int> q;
  q.push(source); dist[source] = 0;
  while (!q.empty()){
    int u = q.front(); q.pop();
    for (edge e: g[u]){
      int v = e.to, flow = e.flow, cap = e.cap;
      if (flow < cap && minimize(dist[v], dist[u] +
          1))
        q.push(v);
    }
  }
  return dist[sink] < INF;
}

int dfs(int u, int mn){
  if (u == sink) return mn;
  for (int &i = lazy[u]; i < sz(g[u]); ++i){
    auto &[v, rev, flow, cap] = g[u][i];
    if (dist[v] == dist[u] + 1 && flow < cap){
      int cur = dfs(v, min(mn, cap - flow));
      if (cur > 0){
        flow += cur;
        g[v][rev].flow -= cur;
        return cur;
      }
    }
  }
  return 0;
}

int main(){
    //...
    int res = 0;
    while (bfs()){
        memset(lazy, 0, sizeof lazy);
        while (int del = dfs(source, INF))
            res += del;
    }

    cout << res;
    return 0;
}
```

Bipartite Matching

```
bool dfs(int u){
  if (seen[u]) return 0;
  seen[u] = 1;

  for (int v: g[u])
    if (!mt[v] || dfs(mt[v]))
      return mt[v] = u, 1;

  return 0;
}

//memset(mt, 0, sizeof mt);
//For(i, 1, n){
    //memset(seen, 0, sizeof seen);
    //dfs(i);
//}
```

HLD

```
void dfs(int u){
  sz[u] = 1;
  for (int v: g[u]) if (v != par[u]){
    par[v] = u;
    dfs(v);
    sz[u] += sz[v];
  }
}

void hld(int u){
  if (!Head[nChain]) Head[nChain] = u;
  idChain[u] = nChain;

  pos[u] = ++timer;
  node[timer] = u;

  int bigC = 0;
  for (int v: g[u]) if (v != par[u])
    if (!bigC || sz[v] > sz[bigC])
      bigC = v;

  if (bigC) hld(bigC);
  for (int v: g[u]) if (v != par[u] && v != bigC){
    ++nChain;
    hld(v);;
  }
}

//LCA
int LCA(int u, int v){
  while (idChain[u] != idChain[v]){
    if (idChain[u] > idChain[v])
      u = par[Head[idChain[u]]];
    else
      v = par[Head[idChain[v]]];
  }

  if (h[u] < h[v]) return u;
  return v;
}

int get(int u, int v){
  int res = 0;
  while (idChain[u] != idChain[v]){
    if (idChain[u] > idChain[v]){
        maximize(res, ST.get(pos[Head[idChain[u]]],
            pos[u]));
        u = par[Head[idChain[u]]];
    }
    else{
        maximize(res, ST.get(pos[Head[idChain[v]]],
            pos[v]));
        v = par[Head[idChain[v]]];
    }
  }

  if (pos[u] < pos[v])
    maximize(res, ST.get(pos[u], pos[v]));
  else
    maximize(res, ST.get(pos[v], pos[u]));

  return res;
}
```

DSU on tree

```
void dfs(int u, int prev = -1){
  in[u] = ++timer; node[timer] = u;
  for (int v: g[u]) if (v != prev)
    dfs(v, u);
```

```
  out[u] = timer;
}

#define sz(u) out[u] - in[u]

void calc(int u, int prev = -1){
  int bigC = 0;
  for (int v: g[u]) if (v != prev)
    if (sz(v) > sz(bigC))
      bigC = v;

  for (int v: g[u]) if (v != prev && v != bigC){
        calc(v, u);
        //reset(v)...
    }

  if (bigC) calc(bigC, u);

  for (int v: g[u]) if (v != prev && v != bigC){
    For(t, in[v], out[v]){
      int x = node[t];
      //...
    }
  }
}
```

Centroid Decomposition

```
int size(int u, int prev){
  sz[u] = 1;
  for (int v: g[u]) if (!del[v] && v != prev)
    sz[u] += size(v, u);
  return sz[u];
}

int centroid(int u, int prev){
  for (int v: g[u]) if (!del[v] && v != prev)
    if (sz[v] > n/2)
      return centroid(v, u);
  return u;
}

void dfs(int u, int prev){
    in[u] = ++timer; node[timer] = u;
    for (int v: g[u]) if (!del[v] && v != prev){
        dfs(v, u);
        //...
    }
    out[u] = timer;
}

void calc(int u){
  n = size(u, 0);
  u = centroid(u, 0);

  timer = 0;
  dfs(u, 0);

    for (int v: g[u]) if (!del[v]){
        //subtree v...
    }
    //reset
  del[u] = 1;
  for (auto [v, c]: g[u]) if (!del[v])
    calc(v);
}
```

Centroid Tree (CT)

Centroid Tree properties:

- Centroid tree height $\leq \log(n)$
- $LCA(u, v)$ in CT lies on the path from u to v in the original tree

```
int size(int u, int prev){
  sz[u] = 1;
  for (int v: g[u]) if (v != prev && !del[v]){
    sz[u] += size(v, u);
  }
    return sz[u];
}

int centroid(int u, int prev, int m){
  for (int v: g[u]) if (v != prev && !del[v])
    if (sz[v] > m/2)
      return centroid(v, u, m);
  return u;
}

int cd(int u){
  int m = size(u);
  u = centroid(u, 0, m);
  del[u] = 1;
  for (int v: g[u]) if (!del[v]){
    v = cd(v);
    par[v] = u;
  }
  return u;
}

//example problems:
void solve(){
    dfs(1, 0); init(); //to calculate the dist(u, v)
         from the original tree
  cd(1);

  memset(d, 0x3f, sizeof d);

  c[1] = 1; //color
  int pp = 1;
  while (pp){
    minimize(d[pp], dist(pp, 1));
    pp = par[pp];
  }

  while (q--){
    int t; cin >> t;
    if (t == 1){
      int u; cin >> u;
      c[u] = 1;

      int p = u;
      while (p){
        minimize(d[p], dist(p, u));
        p = par[p];
      }
    }
    else{
      int u; cin >> u;
      if (c[u]) {
              cout << 0 << endl; continue;
          }

      int p = u, res = INF;
```

```
      while(p){
        minimize(res, dist(u, p) + d[p]);
        p = par[p];
      }

      cout << res << endl;
    }
  }
}
```

Virtual Tree

```
void dfs(int u){
  in[u] = ++timer;
  for (int v: g[u]) if (v != up[u][0]){
    up[v][0] = u;
    For(j, 1, 17) up[v][j] = up[up[v][j - 1]][j - 1];
    dfs(v);
  }
  out[u] = timer;
}

bool is_anc(int u, int v){
  if (!u) return 1;
  return in[u] <= in[v] && in[v] <= out[u];
}

//short LCA
int lca(int u, int v){
  if (is_anc(u, v)) return u;
  ForD(j, 17, 0){
        if (!is_anc(up[u][j], v)){
            u = up[u][j];
        }
    }
  return up[u][0];
}

bool cmp(int u, int v){
  return in[u] < in[v];
}

void query(){
  cin >> k;
  For(i, 1, k) cin >> a[i], sz[a[i]] = 1;

  sort(a + 1, a + k + 1, cmp);
  For(i, 1, k - 1) a[i + k] = lca(a[i], a[i + 1]);

  sort(a + 1, a + k + k, cmp);
  k = unique(a + 1, a + k + k) - a - 1;

  stack<int> st; st.push(a[1]);
  For(i, 2, k){
    while (!is_anc(st.top(), a[i])) st.pop();
    g[st.top()].pb(a[i]);
    st.push(a[i]);
  }

  res = 0; calc(a[1]);
  cout << res << endl;

  For(i, 1, k) sz[a[i]] = 0, g[a[i]].clear();
}

void solve(){
  //...
  dfs(1);
  For(i, 1, n) g[i].clear();
  while (q--) query();
}
```

DP

Digit DP

```
int f(int id, bool sml, ...){
    if (id < 0) return ...;
    if (!sml && dp[id][...] != -1) return dp[id
        ][...];

    int lim = sml ? a[id] : 9;
    int res = ...;
    For(c, 0, lim){
        update(res, f(id - 1, sml && c == lim, ...));
    }
    if (!sml) dp[id][...] = res;
    return res;
}

int get(int x){
    int n = 0;
    while (x){
        a[n++] = x % 10;
        x /= 10;
    }
    return f(n - 1, 1, ...);
}
```

SOS DP

```
for (int k = 0; k < n; k++)
    for (int mask = 0; mask < (1 << n); mask++)
        if (mask & (1 << k))
            dp[mask] += dp[mask ^ (1 << k)];
```

DP DNC

Solving problems that have the form:

$$dp(i,j) = \min_{0 \le k \le j} \Big(dp(i-1, k-1) + C(k,j) \Big)$$

The cost function has to satisfie the quadrangle inequality: $C(a,c) + C(b,d) \le C(a,d) + C(b,c)$ for all $a \le b \le c \le d$.

Example:
$C(j,i) = f(i-j)$ where f is convex.
$C(j,i) = (i-j)$
$C(j,i) = (i-j)^2$

```
int m, n;
vector<int> dp_before(n + 1), dp_cur(n + 1);

// cost function
int C(int l, int r);

// calculate dp_cur[l], ..., dp_cur[r]
void compute(int l, int r, int optl, int optr) {
    if (l > r) return;

    int mid = (l + r) >> 1;
    pair<int, int> best = {INT_MAX, -1};
    //calculate dp_cur[mid] & opt[i][mid] depend on
        dp_before and cost func
    for (int k = optl; k <= min(mid, optr); ++k) {
        minimize(best, {dp_before[k] + C(k, mid), k})
    }
```

```
    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

int solve() {
    for (int i = 0; i <= n; ++i)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; ++i) {
        compute(0, n, 0, n);
        dp_before = dp_cur;
    }

    return dp_before[n];
}
```

Convex Hull Trick

Adding lines $y = kx + m$ and querying minimum values at integer x.

```
struct Line {
    int k, m;
    mutable int p;

    int eval(int x){
      return k * x + m;
    }

    bool operator < (const Line& l) const {
        return k < l.k;
    }
    bool operator < (const int &x) const {
        return p < x;
    }
};

struct ConvexHull : multiset<Line, less<>> {
    int div(int a, int b) {
        return a / b - ((a ^ b) < 0 && a % b);
    }

    bool bad(iterator x, iterator y) {
        if(y == end()) {
            x->p = LINF;
            return 0;
        }

        if(x->k == y->k) x->p = x->m > y->m ? LINF :
             -LINF;
        else x->p = div(y->m - x->m, x->k - y->k);

        return x->p >= y->p;
    }

    void add(int k, int m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (bad(y, z)) z = erase(z);

        if(x != begin() && bad(--x, y)) bad(x, y =
             erase(y));
        while((y = x) != begin() && (--x)->p >= y->p)
              bad(x, erase(y));
    }

    int query(int x) {
        assert(!empty());
        Line l = *lower_bound(x);
        return l.eval(x);
    }
} CH;

//If you want maximum, just flip signs:
CH.add(-k, -m);
int res = -CH.query(x);
```

1D1D Optimization

Solving problems that have the form:

$$dp(i) = \min_{0 \le j < i} \Big(dp(j) + C(j, i) \Big),$$

```
struct item {
  int l, r, p;
};

long long w(int j, int i) {
    //cost function
}

void solve() {
  deque<item> dq;
  dq.push_back({1, n, 0});
  for (int i = 1; i <= n; ++i) {
    f[i]=f[dq.front().p]+w(dq.front().p,i);
    ++dq.front().l;

    if (dq.front().l > dq.front().r) {
      dq.pop_front();
    }

    while (!dq.empty()) {
      auto [l, r, p] = dq.back();
      if (f[i] + w(i, l) < f[p] + w(p, l)) {
        dq.pop_back();
      }
      else break;
    }

    if (dq.empty()) {
      dq.push_back({i + 1, n, i});
      // h[i+1]=h[i+2]=...=h[n]=i
    }
    else {
      auto& [l, r, p] = dq.back();
      int low = l, high = r;
      int pos = r + 1, mid;
      while (low <= high) {
        mid = (low + high) / 2;
        if (f[i] + w(i, mid) < f[p] + w(p, mid)) {
          pos = mid, high = mid - 1;
        }
        else {
          low = mid + 1;
        }
      }

      r = pos - 1;
      if (pos <= n) {
        dq.push_back({pos, n, i});
        // h[pos]=h[pos+1]=...=h[n]=i
      }
    }
  }
```

```
}
```

Knapsack on Tree

Problem: Given a tree T with N vertices rooted at vertex 1 ($1 \le N \le 5000$). The i-th vertex has a value C_i and a constraint K_i ($|C_i| \le 10^9$, $1 \le K_i \le N$). Choose a subset of vertices such that, in the subtree of every vertex i, there are at most K_i chosen vertices, and the total sum of the chosen vertices' values is maximized.

```
void calc(int V){
    int n = child[V].size();

    for(int v_i: child[V]) {
        calc(v_i);
    }

    for(int i = 0; i <= n; i++)fill(fV[i], fV[i] + N
         + 1, -INF);
    fV[0][0] = 0;

    for(int i = 1; i <= n; i++){
        int v_i = child[V][i - 1];
        for(int a = 0; a <= sz[V]; a++){
            for(int b = 0; b <= sz[v_i]; b++){
                fV[i][a+b] = max(fV[i][a+b], fV[i-1][a
                     ] + dp[v_i][b]);
        }
        sz[V] += sz[v_i];
    }

    for(int k = 0; k <= N; k++){
        if(k > K[V])dp[V][k] = -INF;
        else {
            if(k > 0)dp[V][k] = max(fV[n][k], fV[n][k
                 -1] + C[V]);
            else dp[V][k] = fV[n][k];
        }
    }
    sz[V]++;
}

long long solve() {
    calc(1);
    return *max_element(dp[1], dp[1] + N + 1);
}
```

DP on Broken Profile

count the number of ways you can fill an $n \times m$ grid using 1×2 and 2×1 tiles. ($1 \le n \le 10, 1 \le m \le 1000$)

```
dp[0][0] = 1;
for (int j = 0; j < m; j++)
    for (int i = 0; i < n; i++) {
        for (int mask = 0; mask < (1 << n); mask++) {
            dp[mask][1] = dp[mask ^ (1 << i)][0]; //
                 Horizontal or no tile
            if (i && !(mask & (1 << i)) && !(mask & (1
                 << i - 1))) // Vertical tile
                dp[mask][1] += dp[mask ^ (1 << i - 1)
                     ][0];
            if (dp[mask][1] >= MOD) dp[mask][1] -= MOD
                 ;
        }
        for (int mask = 0; mask < (1 << n); mask++)
            dp[mask][0] = dp[mask][1];
    }
cout << dp[0][0] << '\n';
```

Math

Euler's totient function

```
int phi(int n) {
    int res = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) n /= i;
            res -= res / i;
        }
    }
    if (n > 1) res -= res / n;
    return res;
}
```

Euler's totient function from 1 to N

```
void preCompute(int n) {
    iota(phi, phi + N, 0); //phi[i] = i
    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

Modular Inverse

```
//if MOD is a prime number then phi(MOD)= MOD - 1
int inv(int x, int MOD){
    return Pow(x, phi(MOD) - 1);
}
```

Extended Euclidean Algorithm

```
//computing gcd(a, b) and finding (x, y) that
//ax + by = gcd(a, b)

//recursive version
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1; y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

//iterative version
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
```

```
    return a1;
}
```

Diophantine

```
bool find_any_solution(int a, int b, int c, int &x0,
     int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) return false;

    x0 *= c / g; y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

//all the solutions have the form:
//x = x0 + k * b/g
//y = y0 - k * b/g


//IN A GIVEN INTERVAL:
void shift(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int minx,
      int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g)) return
        0;
    a /= g; b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift(x, y, a, b, (minx - x) / b);
    if (x < minx) shift(x, y, a, b, sign_b);
    if (x > maxx) return 0;
    int lx1 = x;

    shift(x, y, a, b, (maxx - x) / b);
    if (x > maxx) shift(x, y, a, b, -sign_b);
    int rx1 = x;

    shift(x, y, a, b, -(miny - y) / a);
    if (y < miny) shift(x, y, a, b, -sign_a);
    if (y > maxy) return 0;
    int lx2 = x;

    shift(x, y, a, b, -(maxy - y) / a);
    if (y > maxy) shift(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2) swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx) return 0;
    return (rx - lx) / abs(b) + 1;
}
```

Chinese Remainder Theorem

```
// Combine two congruences:
// x = a1 (mod m1), x = a2 (mod m2)
// Returns (x, lcm) or (-1,-1) if no solution
pair<ll, ll> crt2(ll a1, ll m1, ll a2, ll m2) {
    int x, y;
    ll g = gcd(m1, m2, x, y);

    if ((a2 - a1) % g != 0) {
        return {-1, -1}; // no solution
    }

    ll lcm = m1 / g * m2;

    ll k = (a2 - a1) / g;
    ll mult = (1LL * x * k) % (m2 / g);

    ll ans = (a1 + m1 * mult) % lcm;
    if (ans < 0) ans += lcm;

    return {ans, lcm};
}

//solve a system of congruences:
//x = a1 (mod m1)
//x = a2 (mod m2)
//...
//x = ak (mod mk)
pair<ll, ll> crt(vector<ll> a, vector<ll> m) {
    pair<ll,ll> res = {a[0], m[0]};
    for (int i = 1; i < sz(a); i++) {
        res = crt2(res.first, res.second, a[i], m[i])
            ;
        if (res.first == -1) return {-1,-1};
    }
    return res;
}

//x = sol.first (mod sol.second)
```

Rabin-Miller primality test

```
bool test(ll a, ll n, ll k, ll m){
    ll mod = Pow(a, m, n);
    if (mod == 1 || mod == n - 1) return 1;
    for (int l = 1; l < k; ++l){
        mod = (mod * mod) % n;
        if (mod == n - 1) return 1;
    }
    return 0;
}

//check if n is a prime number
bool RabinMiller(ll n){
    if (n == 2 || n == 3 || n == 5 || n == 7) return
        1;
    if (n < 11) return 0;

    ll k = 0, m = n - 1;
    while (!(m & 1)){
        m >>= 1;
        k++;
    }

    const static int repeatTime = 3;
    for (int i = 0; i < repeatTime; ++i){
        ll a = rand() % (n - 3) + 2;
        if (!test(a, n, k, m)) return 0;
    }
    return 1;
}
```

Geometry

Dot Product

```
//remember that u * v = 0 -> u is perdendicular with
    v
//or (u, v) = pi/2
double dotProduct(Vector u, Vector v){
    return u.x * v.x + u.y * v.y;
}
```

Angle

```
//u * v = |u| * |v| * cos(theta)
//-> theta = acos (u * v / (|u| * |v|))
double Cos(Vector u, Vector v){
    return dotProduct(u, v)/(u.len * v.len);
}

double theta(Vector u, Vector v){
    return acos(Cos(u, v));
}
```

Cross Product

```
//u * v = |u| * |v| * sin(theta)
//u * v = u.x * v.y - u.y * v.x
//|u * v| = area of a parallelogram formed by
    adjacent vectors u and v
//= double the area of the triangle
double crossProduct(Vector u, Vector v){
    return u.x * v.y - u.y * v.x;
}
```

Distance from a Point to a line

$d(C, AB) = |\vec{AB} * \vec{AC}|/AB$

```
// Compute the distance from AB to C
// if isSegment is true, AB is a segment, not a line.
double linePointDist(Point A, Point B, Point C, bool
    isSegment){
    double res = abs(cross(A, B, C)) / dist(A, B);
    if (isSegment){
        int dot1 = dot(B, A, C);
        if (dot1 < 0) return distance(B, C);
        int dot2 = dot(A, B, C);
        if (dot2 < 0) return distance(A, C);
    }
    return res;
}
```

Template 1

```
struct vec {
    db x, y;
    vec(db _x = 0, db _y = 0) : x(_x), y(_y) {}
    db dot(const vec &other) { // Compute the dot
        product
        return x * other.x + y * other.y;
    }
    db cross(const vec &other) { // Compute the cross
         product
        return x * other.y - y * other.x;
    }
    db length() const {
        return sqrt(x * x + y * y);
    }
};
using point = vec; // or use 'typedef vec point'
vec operator - (const point &B, const point &A) { //
    vecAB = B - A
    return vec(B.x - A.x, B.y - A.y);
}

// if isSegment is true, AB is a segment, not a line.
db linePointDist(const point &A, const point &B,
    const point &C, bool isSegment) {
    db dist = abs((B - A).cross(C - A)) / (A - B).
        length();
    if (isSegment) {
        db dot1 = (A - B).dot(C - B);
        if (dot1 < 0) return (B - C).length();
        db dot2 = (B - A).dot(C - A);
        if (dot2 < 0) return (A - C).length();
    }
    return dist;
}
```

Intersection of 2 lines and bla bla bla (I have no time bro)

Lines will have the form: $ax + by = c$.

$\vec{AB} \times \vec{AC} > 0 \Rightarrow A, B, C$ are counterclockwise.

$\vec{AB} \times \vec{AC} < 0 \Rightarrow A, B, C$ are clockwise.

$\vec{AB} \times \vec{AC} = 0 \Rightarrow A, B, C$ are collinear.

```
const double eps = 1e-9;
int sign(double x) {
    if (x > eps) return 1;
    if (x < -eps) return -1;
    return 0;
}
double cross(Vec AB, Vec AC) {
    return AB.x * AC.y - AC.x * AB.y;
}
double dot(Vec AB, Vec AC) {
    return AB.x * AC.x + AB.y * AC.y;
}
//intersection of 2 segments
bool intersect(Point A, Point B, Point C, Point D) {
    int ABxAC = sign(cross(B - A, C - A));
    int ABxAD = sign(cross(B - A, D - A));
    int CDxCA = sign(cross(D - C, A - C));
    int CDxCB = sign(cross(D - C, B - C));
    if (ABxAC == 0 || ABxAD == 0 || CDxCA == 0 ||
        CDxCB == 0) {
        // C on segment AB if ABxAC = 0 and CA.CB <=
            0
        if (ABxAC == 0 && sign(dot(A - C, B - C)) <=
            0) return true;
        if (ABxAD == 0 && sign(dot(A - D, B - D)) <=
            0) return true;
        if (CDxCA == 0 && sign(dot(C - A, D - A)) <=
            0) return true;
        if (CDxCB == 0 && sign(dot(C - B, D - B)) <=
            0) return true;
        return false;
    }
    return (ABxAC * ABxAD < 0 && CDxCA * CDxCB < 0);
}
```

Circle passing through 3 points

```
struct Point {
    double x, y;
    Point() { x = y = 0.0; }
```

```
    Point(double x, double y) : x(x), y(y) {}

    Point operator + (const Point &a) const { return
        Point(x + a.x, y + a.y); }
    Point operator - (const Point &a) const { return
        Point(x - a.x, y - a.y); }
    Point operator * (double k) const { return Point(
        x * k, y * k); }
    Point operator / (double k) const { return Point(
        x / k, y / k); }
};

struct Line { // Ax + By = C
    double a, b, c;
    Line(double a = 0, double b = 0, double c = 0) :
        a(a), b(b), c(c) {}
    Line(Point A, Point B) {
        a = B.y - A.y;
        b = A.x - B.x;
        c = a * A.x + b * A.y;
    }
};

Line Perpendicular_Bisector(Point A, Point B) {
    Point M = (A + B) / 2;
    Line d = Line(A, B);
    // the equation of a perpendicular line has the
        form: -Bx + Ay = D
    double D = -d.b * M.x + d.a * M.y;
    return Line(-d.b, d.a, D);
}

//Intersection of 2 Perpendicular Bisector is the
    center of the circle
```

Symmetry

```
struct Line { // Ax + By = C
    double a, b, c;
    Line(double a = 0, double b = 0, double c = 0) :
        a(a), b(b), c(c) {}
};

Point intersect(Line d1, Line d2) {
    double det = d1.a * d2.b - d2.a * d1.b;
    // det != 0 because d1 is perpendicular to d2
    return Point((d2.b * d1.c - d1.b * d2.c) / det, (
        d1.a * d2.c - d2.a * d1.c) / det);
}

Point Symmetry(Point X, Line d) {
    // the equation of a perpendicular line has the
        form: -Bx + Ay = D
    double D = -d.b * X.x + d.a * X.y;
    Line d2 = Line(-d.b, d.a, D);
    Point Y = intersect(d, d2);
    Point X2 = Point(2 * Y.x - X.x, 2 * Y.y - X.y);
    return X2;
}
```

Rotation

To rotate $A(x, y)$ counterclockwise by an angle theta around the origin, we can easily use this formula:

$x' = xcos\theta - ysin\theta$

$y' = xsin\theta + ycos\theta$

```
Point Rotations(Point A, Point C, double rad) {
    Point A2 = A - C;
    Point B2 = Point(A2.x * cos(rad) - A2.y * sin(rad
        ), A2.x * sin(rad) + A2.y * cos(rad));
    Point B = B2 + C;
    return B;
}
```

Area of a Polygon

```
double polygonArea(const vector<Point>& poly) {
    int n = poly.size();
    double area = 0.0;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        area += poly[i].x * poly[j].y - poly[j].x *
            poly[i].y;
    }
    return fabs(area) / 2.0;
}
```

Relative position of a point to a polygon $O(N)$

```
//using Area
//Time Complexity: O(N) per query
PointPolygonPosition position(Polygon plg, Point p) {
    long long sSumTris = 0;
    for (int i = 0; i < plg.nVertices; i++) {
        int i1 = (i + 1) % plg.nVertices;
        Polygon tri(p, plg.vertices[i], plg.vertices[
            i1]);
        auto sTri = tri.area2(); //2 * area
        if (!sTri) {
            return BOUNDARY;
        }
        sSumTris += sTri;
    }
    return (sSumTris == plg.area2() ? INSIDE :
        OUTSIDE);
}
```

Relative position of a point to a polygon $O(logN)$

```
//using binary search
bool isCW(Point a, Point b, Point c) {
    return (Vector(a, b) ^ Vector(a, c)) < 0;
}

PointPolygonPosition position(Polygon plg, Point p) {
    // Check if P is on A_1A_n
    Vector pa1(p, plg.vertices[0]);
    Vector pan(p, plg.vertices[plg.nVertices - 1]);
    if (pa1 ^ pan == 0) { //cross product
        if (1ll * pa1.x * pan.x <= 0) {
            return BOUNDARY;
        }
        return OUTSIDE;
    }

    int l = 1, r = plg.nVertices;
    while (r - l > 1) {
        int mid = (l + r) >> 1;
        if (isCW(plg.vertices[0], p, plg.vertices[mid
            ])) {
            l = mid;
        } else {
            r = mid;
        }
```

```
    }
    int k = 1;
    if (k == plg.nVertices - 1) {
        return OUTSIDE;
    }

    // Check if P is on the triangle
    if (Vector(p, plg.vertices[k]) ^ Vector(p, plg.
        vertices[k + 1]) == 0) {
        return BOUNDARY;
    }
    long long ss = 0;
    ss += Polygon(p, plg.vertices[0], plg.vertices[k
        ]).area2();
    ss += Polygon(p, plg.vertices[k], plg.vertices[k
        + 1]).area2();
    ss += Polygon(p, plg.vertices[k + 1], plg.
        vertices[0]).area2();
    if (ss == Polygon(plg.vertices[0], plg.vertices[k
        ],
                    plg.vertices[k + 1]).area2()) {
        return INSIDE;
    }
    return OUTSIDE;
}
```

Pick Theorem

$A = I + B/2 - 1$

- A = Area of the Polygon
- I = Number of interior lattice points (strictly inside the polygon)
- B = Number of boundary lattice points (on the polygon edges)

Convex Hull (Graham scan)

```
// Cross Product of AB and AC
long long cross(const Point &A, const Point &B, const
     Point &C) {
    return 1LL * (B.x - A.x) * (C.y - A.y) - 1LL * (C
        .x - A.x) * (B.y - A.y);
}

// A -> B -> C clockwise (-1), collinear (0),
    counterclockwise (1)
int ccw(const Point &A, const Point &B, const Point &
    C) {
    long long S = cross(A, B, C);
    if (S < 0) return -1;
    if (S == 0) return 0;
    return 1;
}

//convex hull listed in counterclockwise order
vector<Point> convexHull(vector<Point> p, int n) {
    for (int i = 1; i < n; ++i) {
        if (p[0].y > p[i].y || (p[0].y == p[i].y && p
            [0].x > p[i].x)) {
            swap(p[0], p[i]);
        }
    }

    sort(p.begin() + 1, p.end(), [&p](const Point &A,
         const Point &B) {
        int c = ccw(p[0], A, B);
        if (c > 0) return true;
        if (c < 0) return false;
        return A.x < B.x || (A.x == B.x && A.y < B.y)
            ;
    });

    vector<Point> hull;
    hull.push_back(p[0]);

    for (int i = 1; i < n; ++i) {
        while (hull.size() >= 2 && ccw(hull[hull.size
            () - 2], hull.back(), p[i]) < 0) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }
    return hull;
}
```

Convex Hull (Monotone chain algorithm)

```
bool ccw(const Point &A, const Point &B, const Point
    &C) {
    return 1LL * (B.x - A.x) * (C.y - A.y) - 1LL * (C
        .x - A.x) * (B.y - A.y) > 0;
}

vector<Point> convexHull(vector<Point> p, int n) {
    sort(p.begin(), p.end(), [](const Point &A, const
         Point &B) {
        if (A.x != B.x) return A.x < B.x;
        return A.y < B.y;
    });

    vector<Point> hull;
    hull.push_back(p[0]);

    for (int i = 1; i < n; ++i) {
        while (hull.size() >= 2 && ccw(hull[hull.size
            () - 2], hull.back(), p[i])) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }

    for (int i = n - 2; i >= 0; --i) {
        while (hull.size() >= 2 && ccw(hull[hull.size
            () - 2], hull.back(), p[i])) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }

    if (n > 1) hull.pop_back();

    return hull;
}
```

Algebra

Matrix Multiplication

Example:

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \begin{bmatrix}1\cdot5+2\cdot7 & 1\cdot6+2\cdot8\\3\cdot5+4\cdot7 & 3\cdot6+4\cdot8\end{bmatrix} = \begin{bmatrix}19 & 22\\43 & 50\end{bmatrix}$$

```
#define vi vector<int>
struct Matrix{
  vector<vi> a;
  int r, c;
  Matrix(){
    a.clear(); r = c = 0;
  }
  Matrix(vector<vi> a, int r, int c): a(a), r(r), c(c
      ){}
  Matrix operator * (const Matrix &B) const{
    vector<vi> res(r, vi(B.c, 0));
    vector<vi> b = B.a;

    rep(i, r){
      rep(j, B.c){
        rep(k, c){
          res[i][j] += a[i][k] * b[k][j] % MOD;
          res[i][j] %= MOD;
        }
      }
    }

    return Matrix(res, r, B.c);
  }
};

Matrix Pow(Matrix A, int b){
  vector<vi> a(A.r, vi(A.r, 0));
  rep(i, A.r) a[i][i] = 1;

  Matrix res(a, A.r, A.c);

  while (b){
    if (b & 1) res = res * A;
    A = A * A;
    b >>= 1;
  }

  return res;
}
```

Fast Fourier Transform

```
namespace FFT{

    #define cd complex<long double>
    #define vc vector<cd>

    const long double PI = acosl(-1.0L);
    const int N = 1e6 + 5;
    int rev[N];

    void fft(vc &a, bool inverse = 0){
        int n = sz(a);
        rep(i, n) if (i < rev[i]){
            swap(a[i], a[rev[i]]);
        }

        for (int len = 2; len <= n; len <<= 1){
            cd wn = polar(1.0L, PI/len * (inverse ? -2
                : 2));
            for (int i = 0; i < n; i += len){
                cd w = 1;
                rep(j, len/2){
                    cd u = a[i + j];
                    cd v = a[i + j + len/2] * w;
                    a[i + j] = u + v;
                    a[i + j + len/2] = u - v;
                    w *= wn;
                }
            }
        }

        if (inverse){
            for (cd &x: a){
                x /= n;
            }
        }
    }

    vi operator * (const vi &a, const vi &b){
        if (a.empty() || b.empty()) return {};

        vc fa(all(a));
        vc fb(all(b));

        int n = 1, L = 0;
        while (n < sz(a) + sz(b) - 1) n <<= 1, ++L;
        rep(i, n){
            rev[i] = (rev[i >> 1] | (i & 1) << L) >>
                1;
        }


        fa.resize(n); fb.resize(n);

        fft(fa); fft(fb);

        rep(i, n) fa[i] *= fb[i];
        fft(fa, 1);

        n = sz(a) + sz(b) - 1;
        vi res(n);
        rep(i, n) res[i] = (int)(real(fa[i]) + 0.5);

        return res;
    }
}
```

Number Theory Transform

```
namespace NTT{
    const int MOD = 998244353;
    const int g = 3; //primitive root
    int rev[N];

    void ntt(vi &a, bool inverse = 0){
        int n = sz(a);
        rep(i, n) if (i < rev[i]){
            swap(a[i], a[rev[i]]);
        }

        for (int len = 2; len <= n; len <<= 1){
            int wn = Pow(g, (MOD - 1)/len);
            if (inverse) wn = Pow(wn, MOD - 2);

            for (int i = 0; i < n; i += len){
                int w = 1;
                rep(j, len/2){
                    int u = a[i + j];
                    int v = mul(w, a[i + j + len/2]);

                    a[i + j] = sum(u, v);
                    a[i + j + len/2] = dif(u, v);
```

```
                w = mul(w, wn);
            }
        }
    }

    if (inverse){
        int div_n = Pow(n, MOD - 2);
        rep(i, n) a[i] = mul(a[i], div_n);
    }
}

vi operator * (const vi &a, const vi &b){
    if (a.empty() || b.empty()) return {};

    vi fa(all(a)), fb(all(b));

    int n = 1, L = 0;
    while (n < sz(a) + sz(b) - 1) n <<= 1, ++L;
    rep(i, n){
        rev[i] = (rev[i >> 1] | (i & 1) << L) >>
            1;
    }

    fa.resize(n); fb.resize(n);
    ntt(fa); ntt(fb);

    rep(i, n) fa[i] = mul(fa[i], fb[i]);
    ntt(fa, 1);

    fa.resize(sz(a) + sz(b) - 1);
    return fa;
}
}
```

Combinatoric Formula

```
//DP version:
void preCompute(){
    for (int i = 0; i <= n; i++){
        C[i][0] = 1;
        for (int k = 1; k <= i; k++){
            C[i][k] = C[i - 1][k - 1] + C[i - 1][k];
        }
    }
}

//"you know what it is" version:
int C(int n, int k){
    if (n < k || k < 0) return 0;
    return mul(fact[n], mul(ifact[n - k], ifact[k]))
}
```

Catalan

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{1}{n+1}\binom{2n}{n}$$

Applications

- Number of ways to triangulate a convex polygon with $n+2$ vertices.
- Number of Dyck words of length $2n$ (strings of n X and n Y, every prefix has $\#\text{X} \geq \#\text{Y}$).
- Number of valid parentheses sequences with n pairs.

  For $n = 3$: $((())), (()()), (())(), ()(()), ()()()$.
- Number of ways to parenthesize $(n+1)$ factors. Example ($n = 3$):

  $$((ab)c)d,\ (a(bc))d,\ (ab)(cd),\ a((bc)d),\ a(b(cd))$$
- Number of full binary trees with n internal nodes.

Derangement

```
//Principle of Inclusion-Exclusion
int c = 1;
for (int i = 1; i <= n; i++) {
    c = (c * i) + (i % 2 == 1 ? -1 : 1);
    cout << c << ' ';
}

//DP
//dp[n] = (n - 1)(dp[n - 2] + dp[n - 1])
```