Team Notebook:

IU_Waving_Flag

Number Theory:

1. Sieve

```
const int N = 1e5 + 9;
int spf[N];
vector<int> primes;
void sieve() {
  for(int i = 2; i < N; i++) {
    if (spf[i] == 0) spf[i] = i, primes.push_back(i);
    int sz = primes.size();
    for (int j = 0; j < sz && i * primes[j] < N && primes[j]
<= spf[i]; j++) {
      spf[i * primes[j]] = primes[j];
    }
  }
}
```

2. Segment Sieve

```
ll segmentedSieve(ll L, ll R) {
  ll ok = 0;
  bool isPrime[R - L + 1];
  for (int i = 0; i <= R - L + 1; i++)
    isPrime[i] = true;
  if (L == 1)
    isPrime[0] = false;
  for (ll i = 0; prime[i] * prime[i] <= R; i++) {
    ll curPrime = prime[i];
    ll base = curPrime * curPrime;
    if (base < L) {
      base = ((L + curPrime - 1) / curPrime) * curPrime;
    }
    for (ll j = base; j <= R; j += curPrime) {
      isPrime[j - L] = false;
    }
  }
  for (int i = 0; i <= R - L; i++) {
    if (isPrime[i] == true) {
      ok++;
    }
  }
  return ok;
}
```

3. Euler Phi Precalculate

```
void Eulerphi() {
  for (int i = 1; i <= sz; i++) phi[i] = i;
  for (int i = 2; i <= sz; i++) {
    if (phi[i] == i) {
      for (int j = i; j <= sz; j += i) {
        phi[j] = (phi[j] / i) * (i - 1);
      }
    }
  }
}
```

4. Bigmod

```
  int bigmod(int a, int b, int M) {
    if (b == 0) return 1 % M;
    int x = bigmod(a, b / 2, M);
    x = (x * 1 ll * x) % M;
    if (b % 2 == 1) x = (x * 1 ll * a) % M;
    return x;
  }
```

5. Euler Phi(single)

```
ll Eulerphi(ll n) {
  ll idx = 0, pf = prime[idx], ans = n;
  while (pf * pf <= n) {
    if (n % pf == 0) ans -= ans / pf;
    while (n % pf == 0) n /= pf;
    pf = prime[++idx];
  }
  if (n != 1) ans -= ans / n;
  return ans;
}
```

6. NCR DP

```
ll dp[66][33];
ll nCr(int n, int r) {
  if (n == r) return dp[n][r] = 1;
  if (r == 0) return dp[n][r] = 1;
  if (r == 1) return dp[n][r] = (i64) n;
  if (dp[n][r]) return dp[n][r];
  return dp[n][r] = nCr(n - 1, r) + nCr(n - 1, r - 1);
}
```

7. The number of relative primes in a given interval:

```
  int solve(int n, int r) {
    vector < int > p;
    for (int i = 2; i * i <= n; ++i)
      if (n % i == 0) {
        p.push_back(i);
        while (n % i == 0)
          n /= i;
      }
    if (n > 1)
      p.push_back(n);
    int sum = 0;
    for (int msk = 1; msk < (1 << p.size()); ++msk) {
      int mult = 1,
        bits = 0;
      for (int i = 0; i < (int) p.size(); ++i)
        if (msk & (1 << i)) {
          ++bits;
          mult *= p[i];
        }
      int cur = r / mult;
      if (bits % 2 == 1)
        sum += cur;
      else
        sum -= cur;
    }
    return r – sum;
}
```

8. Inverse mod: (a / b) % m = a * bigmod(b, mod - 2, mod)

9. NOD/SOD:

$n = p_1^{a1} * p_2^{a2} * \ldots\ldots\ldots * p_n^{an}$

$NOD(n) = (a1+1)(a2+1)\ldots\ldots(a_n+1)$

$SOD(n) = (p_1^{a1+1}-1)/(p_1-1) * (p_2^{a2+1}-1)/(p_2-1)\ldots$

10.PHI

$phi(n) = n(1-1/p_1)*(1-1/p_2)*\ldots.*(1-1/p_k)$

Here, p = is prime factor of n.

Data Structure:

1. DSU

```
int find_set(int v) {
  if (v == parent[v]) return v;
  return parent[v] = find_set(parent[v]);
}
void union_set(int a, int b) {
  a = find_set(a);
  b = find_set(b);
  if (a != b) {
    parent[b] = a;
  }
}
```

2. Small TO Large

```
void dfs(int u, int par) {
  bucket[u] = u;
  for (int v: adj[u]) {
    if (v != par) {
      dfs(v, u);
    }
  }
  st[bucket[u]].insert(ar[u]);
  for (int v: adj[u]) {
    if (v != par) {
      int a = st[bucket[u]].size();
      int b = st[bucket[v]].size();
      if (b > a) swap(bucket[u], bucket[v]);
      for (int p: st[bucket[v]]) st[bucket[u]].insert(p);
    }
  }
}
```

3. TRIE

```
void add(string s) {
  int cur = 1;
  for (char ch: s) {
    if (!par[cur][ch - '0']) par[cur][ch - '0'] = ++nodecnt;
    cur = par[cur][ch – '0'];  }
  mark[cur]++; }
void del(string s) {
  int cur = 1;
  for (char ch: s) {
    cur = par[cur][ch - '0'];
  }
  mark[cur]--;
}
void dfs(int u) {
  subcnt[u] = mark[u];
  for (int i = 0; i < 26; i++) {
    int v = par[u][i];
    if (v) { dfs(v);   subcnt[u] += subcnt[v];
    }
  }
}
int query(string s) {
  int cur = 1;
  for (char ch: s) {
    if (!par[cur][ch - '0']) return 0;
    cur = par[cur][ch - '0'];
  }
  return mark[cur];
}
```

4. Segment Tree(Build):

```
void init(int node, int start, int end) {
  if (start == end) {
    //here base value setup
    return;
  }
  int left = node * 2;
  int right = node * 2 + 1;
  int mid = (start + end) / 2;
  init(left, start, mid)
  init(right, mid + 1, end);
  //clear lazy here
}
```

5. Segment Tree Update:

```
void update(int node, int start, int end, int l, int r) {
  if (lazy[node] != 0) {
    if (start != end) {
      lazy[node * 2] += lazy[node];
      lazy[node * 2 + 1] += lazy[node];
      lazy[node] = 0;
    }
  }
  if (start > end || start > r || end < l) return;
  if (start >= l && end <= r) {
    //update the segment
    return;
  }
  int left = node * 2;
  int right = node * 2 + 1;
  int mid = (start + end) / 2;
  update(left, start, mid, l, r);
  update(right, mid + 1, end, l, r);
  //merge answer of two sides
}
```

6. Segment Tree(query):

```
int query(int node, int start, int end, int l) {
  if (start > end || start > l || end < l) return;
  if (lazy[node] != 0) {
    if (start != end) {
      lazy[node * 2] += lazy[node];
      lazy[node * 2 + 1] += lazy[node];
      lazy[node] = 0;
    }
  }
  if (start >= l && end <= l) {
    // return value
    return;
  }
  int left = node * 2;
  int right = node * 2 + 1;
  int mid = (start + end) / 2;
  int p1 = query(left, start, mid, l);
  int p2 = query(right, mid + 1, end, l);
  //merge p1 & p2
}
```

7. Range Sum Query(SPARSE TABLE):

```
int query(int L, int R) {
  int seg = R - L + 1;
  int k = pw[seg];
  int sum = 0;
  for (int K = k; K >= 0; K--) {
    if ((1 << K) <= (R - L + 1)) {
      sum += Tab[L][K];
      L += (1 << K);
    }
  }
  return sum;
}
signed main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> ar[i];
    for (int i = 1; i <= n; i++) Tab[i][0] = ar[i];
    pw[1] = 0;
    for (int i = 1; i <= n; i++) {
      pw[i] = pw[i / 2] + 1;
    }
    int len = log2(n);
    for (int k = 1; k <= len; k++) {
      for (int i = 1; i + (1 << k) - 1 <= n; i++) {
        int end_point = i + (1 << (k - 1));
        Tab[i][k] = Tab[i][k - 1] + Tab[end_point][k - 1];
      }
    }
```

8. LCA

```
void dfs(int u, int par) {
  Tab[u][0] = par;
  for (int v: adj[u]) {
    if (v != par) {
      d[v] = d[u] + 1;
      dfs(v, u);
    }
  }
}
int parent(int u, int k) {
  for (int mask = 20; mask >= 0; mask--) {
    if ((k >> mask) & 1) // if kth bit is on
    {
      u = Tab[u][mask];
    }
  }
  return u;
}
int Lca(int u, int v) {
  if (d[u] < d[v]) swap(u, v);
  int k = d[u] - d[v];
  u = parent(u, k);
  if (u == v) return v;
  for (int mask = 20; mask >= 0; mask--) {
    if (Tab[u][mask] != Tab[v][mask]) {
      u = Tab[u][mask];
      v = Tab[v][mask];
    }
  }
  return Tab[u][0];
}
```

9. MO' S Algorithm:

```
void remove(idx);
void add(idx);
int get_answer();

int block_size;
bool cmp(pair<int, int> p, pair<int, int> q) {
  // optimized sorting
  if (p.first / block_size != q.first / block_size) return p < q;
  return (p.first / block_size & 1) ? (p.second < q.second)
                                    : (p.second > q.second);
}
struct Query {
  int l, r, idx;
  // normal sorting // don't use both at the same time;
  bool operator<(Query other) const {
    return make_pair(l / block_size, r) <
           make_pair(other.l / block_size, other.r);
  }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
  block_size = sqrt(n);
  vector<int> answers(queries.size());
  sort(queries.begin(), queries.end());

  int cur_l = 0;
  int cur_r = -1;

  for (Query q : queries) {
    while (cur_l > q.l) {
      cur_l--;
      add(cur_l);
    }
    while (cur_r < q.r) {
      cur_r++;
      add(cur_r);
    }
    while (cur_l < q.l) {
      remove(cur_l);
      cur_l++;
    }
    while (cur_r > q.r) {
      remove(cur_r);
      cur_r--;
    }
    answers[q.idx] = get_answer();
  }
  return answers;
}
```

1. Dijkstra

```
#define infinity 1 << 30
const int N = 10009;
vector < int > vec[N], cost[N];
struct node {
  int u, cost;
  node(int _u, int _cost) {
    u = _u;
    cost = _cost;
  }
  bool operator < (const node & p) const {
    return p.cost < cost;
  }
};
void dijkstra(int n, int s) {
  int dis[N + 1];
  for (int i = 1; i <= n; i++) {
    dis[i] = infinity;
  }
  priority_queue < node > q;
  q.push(node(s, 0));
  dis[s] = 0;
  while (!q.empty()) {
    node top = q.top();
    q.pop();
    int u = top.u;
    int val = top.cost;
    if (dis[u] != val) continue;
    int sz = vec[u].size();
    for (int i = 0; i < sz; i++) {
      int v = vec[u][i];
      if (dis[u] + cost[u][i] < dis[v]) {
        dis[v] = dis[u] + cost[u][i];
        q.push(node(v, dis[v]));
      }
    }
  }
  for (int i = 1; i <= n; i++) {
    cout << s << "-->" << i << " " << dis[i] << endl;
  }
}
```

2. 0 - 1 BFS

```
vector < int > d(n, INF);
d[s] = 0;
deque < int > q;
q.push_front(s);
while (!q.empty()) {
  int v = q.front();
  q.pop_front();
  for (auto edge: adj[v]) {
    int u = edge.first;
    int w = edge.second;
    if (d[v] + w < d[u]) {
      d[u] = d[v] + w;
      if (w == 1)
        q.push_back(u);
      else
        q.push_front(u);
    }
  }
}
```

3. TopSort using kahn's algo

```
int degree[100007];
priority_queue < int > q;
bool topsort() {
  for (i = 1; i <= n; i++) {
    if (!degree[i]) q.push(-i);
  }

  while (!q.empty()) {
    ll u = abs(q.top());
    vec.push_back(u);
    q.pop();
    for (i = 0; i < adj[u].size(); i++) {
      ll v = adj[u][i];
      degree[v]--;
      if (!degree[v]) {
        q.push(-v);
      }
    }
  }
}
```

4. Floyd Warshall

```
for (int k = 0; k < n; ++k) {
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      if (d[i][k] < INF && d[k][j] < INF)
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
    }
  }
}
```

5. Diameter of Every Subtree:

```
 void dfs(int u, int p) {
    for (int v: adj[u]) {
      if (v != p) {
        dfs(v, u);
        // dis[u]+=dis[v];
      }
    }
    dis[u] = 0;
    priority_queue < int > q;
    for (int v: adj[u]) {
      if (v != p) {
        dis[u] = max(dis[u], dis[v] + 1);
        q.push(dis[v]);
      }
    }
    int cnt = 0;
    while (!q.empty()) {
      ans[u] += (q.top() + 1);
      q.pop();
      cnt++;
      if (cnt == 2) break;
    }
  }
```

1.Policy Based Data Structure:

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

/*
find_by_order(k)=returns ans iterator to the k-th largest
element(0 index);

order_of_key(k)=the number of items in a set that are
strictly smaller than our item

Declaration: ordered_set<int> x;
Insert: x.insert(val)

x.find_by_order(n)
x.order_of_key(n)

(end(x)==x.find_by_order(n))//true when no element of
order n
*/
```

2.Bitwise:

1. Checkbit:

```
bool checkbit(int n, int k) {
  return ((n >> k) & 1);
}
```

2. SetBit:

```
int setbit(int n, int k) {
  //kth bit of n is being set by this operation
  return ((1 << k) | n);
}
```

3. Toggle Bit:

```
int toggleBit(int n, int k) {
  return (n ^ (1 << (k - 1)));
}
```

4.Others:

```
#define ok cerr << "Line " << __LINE__ << " : " << "ok"
<< endl

#define DBG(a) cerr << "Line " << __LINE__ << " : " <<
#a << " = " << (a) << endl

#define fastio {
 ios_base::sync_with_stdio(false);
 cin.tie(NULL);
}
```

Stress Test:

```
set -e
g++ code.cpp -o code
g++ gen.cpp -o gen
g++ brute.cpp -o brute
for((i = 1; ; ++i)); do
   ./gen $i > in
   ./code < in > myAnswer
   ./brute < in > correctAnswer
   diff -Z myAnswer correctAnswer > /dev/null || break
   echo "Passed test: "  $i
done
echo "WA on the following test:"
cat in
echo "Your answer is:"
cat myAnswer
echo "Correct answer is:"
cat correctAnswer
```

Random Number Generator:

```
mt19937
rng(chrono::steady_clock::now().time_since_epoch().cou
nt());

int my_rand(int l, int r) {
   return uniform_int_distribution<int>(l, r) (rng);
}
```

Hashing:

```
const int N = 1e6 + 10;

long long bigmod(long long a, long long b, long long m)
{
  if(b == 0) return 1;

  long long x = bigmod(a, b/2, m);
  x = (x * x) % m;
  if(b % 2 == 1) x = (x * a) % m;

  return x;
}

const int M1 = 127657753, M2 = 987654319;
const int p1 = 137, p2 = 277;

int ip1, ip2;

pair < int, int > pw[N], ipw[N];

void init() {
   pw[0] = {1, 1};

   for (int i = 1; i < N; i ++) {
      pw[i].first = (1LL * pw[i - 1].first * p1) % M1;
      pw[i].second = (1LL * pw[i - 1].second * p2) % M2;
   }

   int ip1 = bigmod(p1, M1 - 2, M1);
   int ip2 = bigmod(p2, M2 - 2, M2);

   ipw[0] = {1, 1};

   for (int i = 1; i < N; i ++) {
      ipw[i].first = (1LL * ipw[i - 1].first * ip1) % M1;
      ipw[i].second = (1LL * ipw[i - 1].second * ip2) %
M2;
   }
}

struct Hash {

   int n;

   string s;

   vector < pair < int, int > > hs; // 1 - based

   Hash(){}

   Hash(string r) {

      int n = r.size();

      s = r;

      hs.push_back({0, 0});

  for (int i = 1; i < n; i ++) {
        pair < int, int > p;

        p.first = (hs[i].first + 1LL * pw[i].first * s[i] %
M1) % M1;
        p.second = (hs[i].second + 1LL * pw[i].second *
s[i] % M2) % M2;

        hs.push_back(p);
     }
   }

   pair < int, int > get_hash(int l, int r) {
   pair < int, int > res;

      res.first = (hs[r].first - hs[l - 1].first + M1) * 1LL *
ipw[l - 1].first % M1;
      res.second = (hs[r].second - hs[l - 1].second + M2) *
1LL * ipw[l - 1].second % M2;

      return res;
   }

   pair < int, int > get_hash() {
      return get_hash(1, n);
   }

};
```

Hashing Rules:

subset hash :
{x, y, z} = b ^ x + b ^ y + b ^ z;

grid hash :
(x1,y1) (x1, y2)
(x2, y1) (x2, y2);

need 2 base bx, by : g[x1][y1] * bx ^ x1 * by ^ y1 + g[x1][y2] * bx ^ x1 * by ^ y2;

tree hash:

h[i] = digit * base ^ power + (h[i - 1]);

here,
digit = size of subtree;
power = level of node;

other way use parenthesis "()" , make hash and sort them.

Reverse Hash:

```
const int M1 = 127657753, M2 = 987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair < int, int > pw[N], ipw[N];
void init() {
   pw[0] = {1, 1};
   for (int i = 1; i < N; i ++) {
      pw[i].first = (1LL * pw[i - 1].first * p1) % M1;
      pw[i].second = (1LL * pw[i - 1].second * p2) % M2;
   }
   int ip1 = bigmod(p1, M1 - 2, M1);
   int ip2 = bigmod(p2, M2 - 2, M2);

   ipw[0] = {1, 1};

   for (int i = 1; i < N; i ++) {
      ipw[i].first = (1LL * ipw[i - 1].first * ip1) % M1;
      ipw[i].second = (1LL * ipw[i - 1].second * ip2) %
M2;
   }
}
struct Hash {
   int n;
   string s;
   vector < pair < int, int > > hs; // 1 - based
   vector < pair < int, int > > rhs; // 1 - based
   Hash(){}
   Hash(string r) {
      int n = r.size();
      s = r;
      hs.push_back({0, 0});
      rhs.push_back({0, 0});
      for (int i = 0; i < n; i ++) {
         pair < int, int > p, rp;
         p.first = (hs[i].first + 1LL * pw[i].first * s[i] %
M1) % M1;
         rp.first = (rhs[i].first + 1LL * pw[i].first * s[n - i -
1] % M1) % M1;
         p.second = (hs[i].second + 1LL * pw[i].second *
s[i] % M2) % M2;
         rp.second = (rhs[i].second + 1LL * pw[i].second *
s[n - i - 1] % M2) % M2;

     hs.push_back(p);
 rhs.push_back(rp);
      }
   }

   pair < int, int > get_hash(int l, int r) {
      pair < int, int > res;

      res.first = (hs[r].first - hs[l - 1].first + M1) * 1LL *
ipw[l - 1].first % M1;
      res.second = (hs[r].second - hs[l - 1].second + M2) *
1LL * ipw[l - 1].second % M2;

      return res;
   }
   pair < int, int > get_hash() {
      return get_hash(1, n);
   }
   pair < int, int > get_rhash(int l, int r) {
      pair < int, int > res;
      res.first = (rhs[r].first - rhs[l - 1].first + M1)
* 1LL * ipw[l - 1].first % M1;
      res.second = (rhs[r].second - rhs[l - 1].second + M2) *
1LL * ipw[l - 1].second % M2;
      return res;
   }
   pair < int, int > get_rhash() {
      return get_rhash(1, n);
   }
};
```

Manacher's Algorithm:

```
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];
      }
      // p[i] --; if ith position is skip
   }
   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); // return
without first and last #
}
```

Prefix function (kmp) :

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

Counting the number of occurrences of each prefix :

```
vector<int> ans(n + 1);
for (int i = 0; i < n; i++)
    ans[pi[i]]++;
for (int i = n-1; i > 0; i--)
    ans[pi[i-1]] += ans[i];
for (int i = 0; i <= n; i++)
    ans[i]++;
```

MEX (minimal excluded) of a sequence:

```
class Mex {
private:
    map<int, int> frequency;
    set<int> missing_numbers;
    vector<int> A;
public:
    Mex(vector<int> const& A) : A(A) {
        for (int i = 0; i <= A.size(); i++)
            missing_numbers.insert(i);

        for (int x : A) {
            ++frequency[x];
            missing_numbers.erase(x);
        }
    }
    int mex() {
        return *missing_numbers.begin();
    }
    void update(int idx, int new_value) {
        if (--frequency[A[idx]] == 0)
            missing_numbers.insert(A[idx]);
        A[idx] = new_value;
        ++frequency[new_value];
        missing_numbers.erase(new_value);
    }
};
```

Finding the totient from 1 to n using the divisor sum property:

Time : nlong(n)

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i - 1;
    for (int i = 2; i <= n; i++)
        for (int j = 2 * i; j <= n; j += i)
            phi[j] -= phi[i];
}
```

Trie:

```
const int ALPHABET_SIZE = 27;
class TrieNode {
public:
    TrieNode() : isEndOfWord(false) {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            children[i] = -1; // Initialize to indicate no child
        }
    }

    bool isEndOfWord;
    int children[ALPHABET_SIZE];
};

class Trie {
public:
    vector < int > depth;
    Trie() {
        nodes.emplace_back(); // Create the root node
    }

    void insert(const string& word) {
        int nodeIdx = 0; // Start from the root
        for (char ch : word) {
            int index = ch - '0';
            if (nodes[nodeIdx].children[index] == -1) {
                nodes.emplace_back();
                nodes[nodeIdx].children[index] = nodes.size() -
1;
            }
            nodeIdx = nodes[nodeIdx].children[index];
        }
        nodes[nodeIdx].isEndOfWord = true;
    }

    int search(const string& word) {
        int nodeIdx = 0; // Start from the root
        for (char ch : word) {
            int index = ch - '0';
            if (nodes[nodeIdx].children[index] == -1) {
                return false;
            }
            nodeIdx = nodes[nodeIdx].children[index];
        }
        return nodes[nodeIdx].isEndOfWord;
    }
    void dfs (int u) {
        depth.emplace_back();
        depth[u] = nodes[u].isEndOfWord;
        for (int c = 0; c < ALPHABET_SIZE; c ++) {
            if (nodes[u].children[c] != -1) {
                int v = nodes[u].children[c];
                dfs (v);
                depth[u] += depth[v];
            }
        }
    }

private:
    vector<TrieNode> nodes;
};
```

Hashing :

```
long long bigmod(long long a, long long b, long long m)
{
  if(b == 0) return 1;

  long long x = bigmod(a, b/2, m);
  x = (x * x) % m;
  if(b % 2 == 1) x = (x * a) % m;

  return x;
}
const int M1 = 127657753, M2 = 987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair < int, int > pw[N], ipw[N];
void init() {
  pw[0] = {1, 1};
  for (int i = 1; i < N; i ++) {
    pw[i].first = (1LL * pw[i - 1].first * p1) % M1;
    pw[i].second = (1LL * pw[i - 1].second * p2) % M2;
  }
  int ip1 = bigmod(p1, M1 - 2, M1);
  int ip2 = bigmod(p2, M2 - 2, M2);
  ipw[0] = {1, 1};
  for (int i = 1; i < N; i ++) {
    ipw[i].first = (1LL * ipw[i - 1].first * ip1) % M1;
    ipw[i].second = (1LL * ipw[i - 1].second * ip2) % M2;
  }
}
struct Hash {
  int n;
  string s;
  vector < pair < int, int > > hs, rhs; // 1 - based
  Hash(){}
  Hash(string r) {
    n = r.size();
    s = r;
    hs.push_back({0, 0});
    rhs.push_back({0, 0});
    for (int i = 0; i < n; i ++) {
      pair < int, int > p;
      p.first = (hs[i].first + 1LL * pw[i].first * s[i] % M1) %
M1;
      p.second = (hs[i].second + 1LL * pw[i].second * s[i] %
M2) % M2;
      hs.push_back(p);
    }
    reverse(s.begin(),s.end());
    for (int i = 0; i < n; i ++) {
      pair < int, int > p;
      p.first = (rhs[i].first + 1LL * pw[i].first * s[i] % M1) %
M1;
      p.second = (rhs[i].second + 1LL * pw[i].second * s[i]
% M2) % M2;
      rhs.push_back(p);
    }
  }
pair < int, int > get_hash(int l, int r) {
    pair < int, int > res;
    res.first = (hs[r].first - hs[l - 1].first + M1) * 1LL *
ipw[l - 1].first % M1;
    res.second = (hs[r].second - hs[l - 1].second + M2) *
1LL * ipw[l - 1].second % M2;
    return res;
  }

  pair < int, int > get_rhash(int l, int r) {
    pair < int, int > res;
    res.first = (rhs[r].first - rhs[l - 1].first + M1) * 1LL *
ipw[l - 1].first % M1;
    res.second = (rhs[r].second - rhs[l - 1].second + M2) *
1LL * ipw[l - 1].second % M2;
    return res;
  }

  pair < int, int > get_hash() {
    return get_hash(1, n);
  }
  pair < int, int > get_rhash() {
    return get_hash(1, n);
  }

};
```

Catalan Numbers:

```
int catalan[MAX];
void init() {
    catalan[0] = catalan[1] = 1;
    for (int i=2; i<=n; i++) {
        catalan[i] = 0;
        for (int j=0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan[i-j-1]) % MOD;
            if (catalan[i] >= MOD) {
                catalan[i] -= MOD;
            }
        }
    }
}
```

Analytical formula:

Cn = 1 / (n + 1) * (2n C n) [nCr formula]

nCk O(k) solutions:

```
int C(int n, int k) {
    double res = 1;
    for (int i = 1; i <= k; ++i)
        res = res * (n - k + i) / i;
    return (int)(res + 0.01);
}
```

Dsu:

```
class UnionFind {
public:
  UnionFind(int n) {
    parent.resize(n);
    rank.resize(n, 0);

    for (int i = 0; i < n; ++i) {
      parent[i] = i; // Each element is initially its own parent
zero based
    }
  }

  int find(int x) {
    if (parent[x] != x) {
      parent[x] = find(parent[x]); // Path compression
    }
    return parent[x];
  }

  bool same_set(int x, int y) {
    int rootX = find(x);
    int rootY = find(y);

    if (rootX == rootY) {
      return true;
    }
    return false;
  }

  void unite(int x, int y) {
    int rootX = find(x);
    int rootY = find(y);

    if (same_set(rootX, rootY)) {
      return; // Already in the same set
    }

    if (rank[rootX] < rank[rootY]) {
      parent[rootX] = rootY;
    } else if (rank[rootX] > rank[rootY]) {
      parent[rootY] = rootX;
    } else {
      parent[rootY] = rootX;
      ++rank[rootX];
    }
  }

private:
  vector<int> parent;
  vector<int> rank;
};
```

Arbitrary-Precision Arithmetic:

```
const int base = 1000*1000*1000;
```

To read a long integer, read its notation into a string and then convert it to "digits":

```
for (int i=(int)strlen(s); i>0; i-=9) {
  s[i] = 0;
  a.push_back (atoi (i>=9 ? s+i-9 : s));
}
```

If the input can contain leading zeros, they can be removed as follows:

```
while (a.size() > 1 && a.back() == 0)
  a.pop_back();
```

Addition: (a by b)

```
int carry = 0;
for (size_t i=0; i<max(a.size(),b.size()) || carry; ++i) {
  if (i == a.size())
    a.push_back (0);
  a[i] += carry + (i < b.size() ? b[i] : 0);
  carry = a[i] >= base;
  if (carry)  a[i] -= base;
}
```

Subtraction:(a by b and a >= b)

```
int carry = 0;
for (size_t i=0; i<b.size() || carry; ++i) {
  a[i] -= carry + (i < b.size() ? b[i] : 0);
  carry = a[i] < 0;
  if (carry)  a[i] += base;
}
while (a.size() > 1 && a.back() == 0)
  a.pop_back();
```

Multiplication by long integer:

```
vector c (a.size()+b.size());
for (size_t i=0; i<a.size(); ++i)
  for (int j=0, carry=0; j<(int)b.size() || carry; ++j) {
    long long cur = c[i+j] + a[i] * 1ll * (j < (int)b.size() ?
b[j] : 0) + carry;
    c[i+j] = int (cur % base);
    carry = int (cur / base);
  }
while (c.size() > 1 && c.back() == 0)
  c.pop_back();
```

Division by short integer:

Divide long integer a by short integer b (b < base) store integer result in a, and reminder in carry:

```
int carry = 0;
for (int i=(int)a.size()-1; i>=0; --i) {
  long long cur = a[i] + carry * 1ll * base;
  a[i] = int (cur / b);
  carry = int (cur % b);
}
while (a.size() > 1 && a.back() == 0)
  a.pop_back();
```

Printing answer:

```
printf ("%d", a.empty() ? 0 : a.back());
for (int i=(int)a.size()-2; i>=0; --i)
  printf ("%09d", a[i]);
```

Gp hash table:

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

gp_hash_table<int, int, chash> table;
struct chash {
        // any random-ish large odd number will do
        const uint64_t C = uint64_t(2e18 * PI) + 71;
        // random 32-bit number
        const uint32_t RANDOM =

chrono::steady_clock::now().time_since_epoch().count();
        size_t operator()(uint64_t x) const {
                // see
https://gcc.gnu.org/onlinedocs/gcc/Other-Builtins.html
                return __builtin_bswap64((x ^
RANDOM) * C);
        }
};
```

Finding Bridge offline (N + M):

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
   visited[v] = true;
   tin[v] = low[v] = timer++;
   for (int to : adj[v]) {
      if (to == p) continue;
      if (visited[to]) {
         low[v] = min(low[v], tin[to]);
      } else {
         dfs(to, v);
         low[v] = min(low[v], low[to]);
         if (low[to] > tin[v])
            IS_BRIDGE(v, to);
      }
   }
}

void find_bridges() {
   timer = 0;
   visited.assign(n, false);
   tin.assign(n, -1);
   low.assign(n, -1);
   for (int i = 0; i < n; ++i) {
      if (!visited[i])
         dfs(i);
   }
}
```