MIST_Untitled

Last modified: November 17, 2025

Contents

1 C++

1.1 template

```
/*
c++:
ios_base::sync_with_stdio(false);
cin.tie(nullptr), cout.tie(nullptr);

python:
import sys
input = sys.stdin.readline
sys.stdout.write("------")
*/
```

1.2 random

```
#define accuracy chrono::steady_clock::
    now().time_since_epoch().count()
mt19937 rng(accuracy);

ll rand(ll l, ll r) {
  uniform_int_distribution<ll> ludo(l,
      r);
  return ludo(rng);
}
```

1.3 gp_hash

```
#include <ext/pb_ds/assoc_container.hpp
    >
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename p, typename q> using
     ht = gp_hash_table<p, q>;
```

1.4 pbds

```
#include <ext/pb_ds/assoc_container.hpp
    >
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T>
using o_set = tree<T, null_type, less<T
    >, rb_tree_tag,
    tree_order_statistics_node_update>;
// find_by_order(k) - returns an
    iterator to
// the k-th largest element (0 indexed)
    ;
// order_of_key(k)-the number of
    elements in // the set that are
    strictly
// smaller than k;
```

1.5 debug

```
string to_string(const string &s) {
    return '"' + s + '"'; }
string to_string(const char *s) {
    return to_string(string(s)); }
string to_string(const char c) { return
    "'" + string(1, c) + "'"; }
string to_string(bool b) { return b ? "
    true" : "false"; }
template <typename A, typename B>
    string to_string(pair<A, B> p) {
  return "(" + to_string(p.first) + ",
      " + to_string(p.second) + ")";
}
template <typename A> string to_string(
    A v) {
  string res = "{";
  for (const auto &x : v) {
    res += to_string(x) + ", ";
  }
  res += "}";
  return res;
}
void debug_out() { cerr << endl; }
template <typename Head, typename...
    Tail> void debug_out(Head H, Tail...
    T) {
  cerr << " " << to_string(H);
  debug_out(T...);
}
#define dbg(...)
   \
  cerr << __LINE__ << ": [" << #
      __VA_ARGS__ << "] = ", debug_out(
      __VA_ARGS__)
```

1.6 stress

```
#!/usr/bin/env bash
wrong="solution"
correct="brute"
gen="gen"
g++ -g solution.cpp -DONPC -o "$wrong"
g++ -g brute.cpp -DONPC -o "$correct"
g++ -g gen.cpp -DONPC -o "$gen"

for ((testNum=0;testNum<$1;testNum++))
do
        ./$gen 2>/dev/null > stdinput
        ./$correct < stdinput 2>/dev/
            null > outSlow
        ./$wrong < stdinput 2>/dev/null
            > outWrong
        H1=`md5sum outWrong`
        H2=`md5sum outSlow`
        if !(cmp -s "outWrong" "outSlow
            ")
        then
        echo "Error found!"
        echo "Input:"
        cat stdinput
        echo "Wrong Output:"
        cat outWrong
        echo "Slow Output:"
        cat outSlow
        exit
        fi
done
echo Passed $1 tests
# Usage: ./contest.sh times
```

1.7 vscode

```
{
  "key" : "f5",
  "command" : "workbench.action.
      terminal.sendSequence",
  "args" : {
    "text" : "g++ ${
        fileBasenameNoExtension}.cpp -o
         ${fileBasenameNoExtension} &&
        ./ ${fileBasenameNoExtension} <
        in.txt> out.txt\n "
  }
}
```

2 Dsa

2.1 KMP

```
vector<ll> createLPS(string pattern) {
  ll n = pattern.length(), idx = 0;
  vector<ll> lps(n);
  for (ll i = 1; i < n;) {
    if (pattern[idx] == pattern[i]) {
      lps[i] = idx + 1;
      idx++, i++;
    } else {
      if (idx != 0)
        idx = lps[idx - 1];
      else
        lps[i] = idx, i++;
    }
  }
  return lps;
}
ll kmp(string text, string pattern) {
  ll cnt_of_match = 0, i = 0, j = 0;
  vector<ll> lps = createLPS(pattern);
  while (i < text.length()) {
    if (text[i] == pattern[j])
      i++, j++; // i = text, j =
          pattern
    else {
      if (j != 0)
        j = lps[j - 1];
      else
        i++;
    }
    if (j == pattern.length()) {
      cnt_of_match++;
      // the index where match found ->
          (i - pattern.length());
      j = lps[j - 1];
    }
  }
  return cnt_of_match;
}
```

2.2 Hashing

```
const ll N = 2e5 + 5;
const ll MOD1 = 127657753, MOD2 =
    987654319;
const ll p1 = 137, p2 = 277;
ll ip1, ip2;
pair<ll, ll> pw[N], ipw[N];
void prec() {
  pw[0] = {1, 1};
  for (ll i = 1; i < N; i++) {
    pw[i].first = 1LL * pw[i - 1].first
         * p1 % MOD1;
    pw[i].second = 1LL * pw[i - 1].
        second * p2 % MOD2;
  }
  ip1 = binaryExp(p1, MOD1 - 2, MOD1);
  ip2 = binaryExp(p2, MOD2 - 2, MOD2);
  ipw[0] = {1, 1};
  for (ll i = 1; i < N; i++) {
    ipw[i].first = 1LL * ipw[i - 1].
        first * ip1 % MOD1;
    ipw[i].second = 1LL * ipw[i - 1].
        second * ip2 % MOD2;
  }
}
struct Hashing {
  ll n;
  string s;                  // 0 -
      indexed
  vector<pair<ll, ll>> hs; // 1 -
      indexed
  Hashing() {}
  Hashing(string _s) {
    n = _s.size();
    s = _s;
    hs.emplace_back(0, 0);
    for (ll i = 0; i < n; i++) {
      pair<ll, ll> p;
      p.first = (hs[i].first + 1LL * pw
          [i].first * s[i] % MOD1) %
          MOD1;
      p.second = (hs[i].second + 1LL *
          pw[i].second * s[i] % MOD2) %
          MOD2;
      hs.push_back(p);
    }
  }
  pair<ll, ll> get_hash(ll l, ll r) {
    // 1 - indexed
    assert(1 <= l && l <= r && r <= n);
    pair<ll, ll> ans;
    ans.first =
        (hs[r].first - hs[l - 1].first
            + MOD1) * 1LL * ipw[l - 1].
            first % MOD1;
    ans.second = (hs[r].second - hs[l -
        1].second + MOD2) * 1LL *
                 ipw[l - 1].second %
                     MOD2;
    return ans;
  }
  pair<ll, ll> get_hash() { return
      get_hash(1, n); }
};
```

2.3 BigInteger

```
struct BigInteger {
  string str;
  // Constructor to initialize
  // BigInteger with a string
  BigInteger(string s) { str = s; }
  // Overload + operator to add
  // two BigInteger objects
  BigInteger operator+(const BigInteger
      &b) {
    string a = str, c = b.str;
    ll alen = a.length(), clen = c.
        length();
    ll n = max(alen, clen);
    if (alen > clen)
      c.insert(0, alen - clen, '0');
    else if (alen < clen)
      a.insert(0, clen - alen, '0');
    string res(n + 1, '0');
    ll carry = 0;
    for (ll i = n - 1; i >= 0; i--) {
            ll digit=(a[i -'0')+(c[i]-'
               0')
    +carry;
            carry = digit / 10;
            res[i + 1] = digit % 10 + '
               0';
    }
    if (carry == 1) {
            res[0] = '1';
            return BigInteger(res);
    } else
            return BigInteger(res.
               substr(1));
  }
  // Overload - operator to subtract
  // first check which number is
      greater and then subtract
  BigInteger operator-(const BigInteger
      &b) {
    string a = str;
    string c = b.str;
    ll alen = a.length(), clen = c.
        length();
    ll n = max(alen, clen);
    if (alen > clen)
            c.insert(0, alen - clen, '0
               ');
    else if (alen < clen)
            a.insert(0, clen - alen, '0
               ');
    if (a < c) {
            swap(a, c);
            swap(alen, clen);
    }
    string res(n, '0');
    ll carry = 0;
    for (ll i = n - 1; i >= 0; i--) {
            ll digit = (a[i] - '0') - (
               c[i] - '0') - carry;
            if (digit < 0)
              digit += 10, carry = 1;
            else
              carry = 0;
            res[i] = digit + '0';
    }
    // remove leading zeros
    ll i = 0;
    while (i < n && res[i] == '0')
            i++;
    if (i == n)
            return BigInteger("0");
    return BigInteger(res.substr(i));
  }
  // Overload * operator to multiply
  // two BigInteger objects
  BigInteger operator*(const BigInteger
      &b) {
    string a = str, c = b.str;
    ll alen = a.length(), clen = c.
        length();
    ll n = alen + clen;
    string res(n, '0');
    for (ll i = alen - 1; i >= 0; i--)
        {
            ll carry = 0;
            for (ll j = clen - 1; j >=
               0; j--) {
              ll digit =
                  (a[i] - '0') * (c[j -
                      '0']) + (res[i +
                     j + 1] - '0') +
                     carry;
              carry = digit / 10;
              res[i + j + 1] = digit %
                 10 + '0';
            }
            res[i] += carry;
    }
    ll i = 0;
    while (i < n && res[i] == '0')
            i++;
    if (i == n)
            return BigInteger("0");
    return BigInteger(res.substr(i));
  }
  // Overload << operator to output
  // BigInteger object
  friend ostream &operator<<(ostream &
      out, const BigInteger &b) {
    out << b.str;
    return out;
  }
};
```

2.4 Kadane

```
// return maximum subarray sum.
ll kadense(ll arr[], ll n) {
  ll mxsm = arr[0], curr_s = arr[0];
  for (ll i = 1; i < n; i++) {
    curr_s = max(arr[i], curr_s + arr[i
        ]);
    mxsm = max(mxsm, curr_s);
  }
  return mxsm;
}
```

2.5 Segement_tree

```
class SEGMENT_TREE {
public:
  vector<ll> v;
  vector<ll> seg;
  SEGMENT_TREE(ll n) {
    v.resize(n + 5);
    seg.resize(4 * n + 5);
  }
  //! initially: ti = 1, low = 1, high
      = n
  //(number of elements in the array);
  void build(ll ti, ll low, ll high) {
    if (low == high) {
      seg[ti] = v[low];
      return;
    }
    ll mid = (low + high) / 2;
    build(2 * ti, low, mid);
    build(2 * ti + 1, mid + 1, high);
    seg[ti] = (seg[2 * ti] + seg[2 * ti
        + 1]);
  }
  //! initially: ti = 1, low = 1, high
      = n
  //(number of elements in the array),
  //(ql & qr)=user input in 1 based
      index;
  ll find(ll ti, ll tl, ll tr, ll ql,
      ll qr) {
    if (tl > qr || tr < ql) {
      return 0;
    }
    if (tl >= ql and tr <= qr)
      return seg[ti];
    ll mid = (tl + tr) / 2;
    ll l = find(2 * ti, tl, mid, ql, qr
        );
    ll r = find(2 * ti + 1, mid + 1, tr
        , ql, qr);
    return (l + r);
  }
  //! initially: ti = 1, tl = 1, tr = n
  //(number of elements in the array),
  // id = user input in 1 based
      indexing,
  // val = updated value;
  void update(ll ti, ll tl, ll tr, ll
      id, ll val) {
    if (id > tr or id < tl)
      return;
    if (id == tr and id == tl) {
      seg[til] = val;
      return;
    }
    ll mid = (tl + tr) / 2;
    update(2 * ti, tl, mid, id, val);
    update(2 * ti + 1, mid + 1, tr, id,
        val);
    seg[ti] = (seg[2 * ti] + seg[2 * ti
        + 1]);
  }
};
// use 1 based indexing;
```

2.6 Fenwick_tree

```
struct FenwickTree {
  vector<ll> bit; // binary indexed
      tree
  ll n;
  FenwickTree(ll n) {
    this->n = n;
    bit.assign(n, 0);
  }
  FenwickTree(vector<ll> a) :
      FenwickTree(a.size()) {
    for (size_t i = 0; i < a.size(); i
        ++)
      add(i, a[i]);
  }
  ll sum(ll r) {
    ll ret = 0;
    for (; r >= 0; r = (r & (r + 1)) -
        1)
      ret += bit[r];
    return ret;
  }
  ll sum(ll l, ll r) { return sum(r) -
      sum(l - 1); }
  void add(ll idx, ll delta) {
    for (; idx < n; idx = idx | (idx +
        1))
      bit[idx] += delta;
  }
};
// minimum
struct FenwickTreeMin {
  vector<ll> bit;
  ll n;
  const ll INF = (ll)1e9;
  FenwickTreeMin(ll n) {
    this->n = n;
    bit.assign(n, INF);
  }
  FenwickTreeMin(vector<ll> a) :
      FenwickTreeMin(a.size()) {
    for (size_t i = 0; i < a.size(); i
        ++)
      update(i, a[i]);
  }
  ll getmin(ll r) {
    ll ret = INF;
    for (; r >= 0; r = (r & (r + 1)) -
        1)
      ret = min(ret, bit[r]);
    return ret;
  }
  void update(ll idx, ll val) {
    for (; idx < n; idx = idx | (idx +
        1))
      bit[idx] = min(bit[idx], val);
  }
};
```

2.7 Segment_tree_lazy

```
class SEGMENT_TREE {
public:
  vector<ll> v;
  vector<ll> seg;
  vector<ll> lazy;
  SEGMENT_TREE(ll n) {
    v.resize(n + 5, 0);
    seg.resize(4 * n + 5, 0);
    lazy.resize(4 * n + 5, 0);
  }
  void pull(ll ti) { seg[ti] = (seg[2 *
      ti] & seg[2 * ti + 1]); }
  void push(ll ti, ll tl, ll tr) {
    if (lazy[ti] == 0)
      return;
    seg[ti] |= lazy[ti];
    if (tl != tr) {
      lazy[2 * ti] |= lazy[ti];
      lazy[2 * ti + 1] |= lazy[ti];
    }
    lazy[ti] = 0;
  }
  //! llially: ti = 1, low = 1, high =
      n(number of elements in the array)
      ;
  void build(ll ti, ll low, ll high) {
    lazy[ti] = 0;
    if (low == high) {
      seg[ti] = v[low];
      return;
    }
    ll mid = (low + high) / 2;
    build(2 * ti, low, mid);
    build(2 * ti + 1, mid + 1, high);
    pull(ti);
  }
  //! llially: ti = 1, low = 1, high =
      n(number of elements in the array)
      , (ql
  //! & qr) = user input in 1 based
      indexing;
  ll query(ll ti, ll tl, ll tr, ll ql,
      ll qr) {
    push(ti, tl, tr);
    if (tl > qr || tr < ql) {
      return (1LL << 32) - 1;
    }
    if (tl >= ql and tr <= qr)
      return seg[ti];
    ll mid = (tl + tr) / 2;
    ll l = query(2 * ti, tl, mid, ql,
        qr);
    ll r = query(2 * ti + 1, mid + 1,
        tr, ql, qr);
    return (l & r);
  }
  //! llially: ti = 1, tl = 1, tr = n(
      number of elements in the array),
      id =
  //! user input in 1 based indexing,
      val = updated value;
  void update(ll ti, ll tl, ll tr, ll
      idL, ll idR, ll val) {
    push(ti, tl, tr);
    if (idR < tl or tr < idL)
      return;
    if (idL <= tl and tr <= idR) {
      lazy[ti] |= val;
      push(ti, tl, tr);
      return;
    }
    ll mid = (tl + tr) / 2;
    update(2 * ti, tl, mid, idL, idR,
        val);
    update(2 * ti + 1, mid + 1, tr, idL
        , idR, val);
    pull(ti);
  }
  // use 1 based indexing for input and
      queries and update;
};
```

2.8 Trie

```
const ll N = 26;
class Node {
public:
  ll EoW;
  Node *child[N];
  Node() {
    EoW = 0;
    for (ll i = 0; i < N; i++)
      child[i] = NULL;
  }
};
void insert(Node *node, string s) {
  for (size_t i = 0; i < s.size(); i++)
      {
    ll r = s[i] - 'A';
    if (node->child[r] == NULL)
      node->child[r] = new Node();
    node = node->child[r];
  }
  node->EoW += 1;
}
ll search(Node *node, string s) {
  for (size_t i = 0; i < s.size(); i++)
      {
    ll r = s[i] - 'A';
    if (node->child[r] == NULL)
      return 0;
  }
  return node->EoW;
}
void prll(Node *node, string s = "") {
  if (node->EoW)
    cout << s << "\n";
  for (ll i = 0; i < N; i++) {
    if (node->child[i] != NULL) {
      char c = i + 'A';
      prll(node->child[i], s + c);
    }
  }
}
bool isChild(Node *node) {
  for (ll i = 0; i < N; i++)
    if (node->child[i] != NULL)
      return true;
  return false;
}
bool isJunc(Node *node) {
  ll cnt = 0;
  for (ll i = 0; i < N; i++) {
    if (node->child[i] != NULL)
      cnt++;
  }
  if (cnt > 1)
    return true;
  return false;
}
ll trie_delete(Node *node, string s, ll
    k = 0) {
  if (node == NULL)
    return 0;
  if (k == (ll)s.size()) {
    if (node->EoW == 0)
      return 0;
    if (isChild(node)) {
      node->EoW = 0;
      return 0;
    }
    return 1;
  }
  ll r = s[k] - 'A';
  ll d = trie_delete(node->child[r], s,
      k + 1);
  ll j = isJunc(node);
  if (d)
    delete node->child[r];
  if (j)
    return 0;
  return d;
}
void delete_trie(Node *node) {
  for (ll i = 0; i < 15; i++) {
    if (node->child[i] != NULL)
      delete_trie(node->child[i]);
  }
  delete node;
}
```

2.9 DSU

```
class DisjollSet {
  vector<ll> par, sz, minElmt, maxElmt,
      cntElmt;
public:
  DisjollSet(ll n) {
    par.resize(n + 1);
    sz.resize(n + 1, 1);
    minElmt.resize(n + 1);
    maxElmt.resize(n + 1);
    cntElmt.resize(n + 1, 1);
    for (ll i = 1; i <= n; i++)
      par[i] = minElmt[i] = maxElmt[i]
          = i;
  }
  ll findUPar(ll u) {
    if (u == par[u])
      return u;
    return par[u] = findUPar(par[u]);
  }
  void unionBySize(ll u, ll v) {
    ll pU = findUPar(u);
    ll pV = findUPar(v);
    if (pU == pV)
      return;
    if (sz[pU] < sz[pV])
      swap(pU, pV);
    par[pV] = pU;
    sz[pU] += sz[pV];
    cntElmt[pU] += cntElmt[pV];
    minElmt[pU] = min(minElmt[pU],
        minElmt[pV]);
    maxElmt[pU] = max(maxElmt[pU],
        maxElmt[pV]);
  }
  ll getMinElementIntheSet(ll u) {
      return minElmt[findUPar(u)]; }
  ll getMaxElementIntheSet(ll u) {
      return maxElmt[findUPar(u)]; }
  ll getNumofElementIntheSet(ll u) {
      return cntElmt[findUPar(u)]; }
};
```

2.10 HLD

```
ll par[N], sub_tree_sz[N], heavy[N],
    wt_from_parent[N], depth[N], head[N
    ],
    position[N];
vector<pair<ll, ll>> gd[N];
// HLD part start
ll dfs(ll node, ll p) {
  par[node] = p;
  sub_tree_sz[node] = 1;
  heavy[node] = -1;

  for (auto [v, w] : gd[node]) {
    if (v == p)
```

```
      continue;
    depth[v] = depth[node] + 1;
    wt_from_parent[v] = w;
    sub_tree_sz[node] += dfs(v, node);
    if (heavy[node] == -1 ||
       sub_tree_sz[v] > sub_tree_sz[
       heavy[node]]) {
      heavy[node] = v;
    }
  }
  return sub_tree_sz[node];
}
ll pos;
void decompose(ll node, ll hd) {
  head[node] = hd;
  position[node] = ++pos;
  if (heavy[node] != -1) {
    decompose(heavy[node], hd);
  }
  for (auto [v, w] : gd[node]) {
    if (v != par[node] && v != heavy[
       node]) {
      decompose(v, v);
    }
  }
}
// HLD part end

// in main function
ll n, m;
cin >> n;
SEGMENT_TREE seg(n); // Lazy if needed
vector<ll> edge_u(n), edge_v(n),
   edge_node(n);

for (int i = 1; i < n; i++) {
  ll u, v, wt = 1;
  cin >> u >> v >> wt;
  gd[u].push_back({v, wt});
  gd[v].push_back({u, wt});
  edge_u[i] = u;
  edge_v[i] = v;
}

dfs(1, -1);
pos = 0;
decompose(1, 1);

for (int i = 1; i <= n; i++) {
  // seg.v[position[i]] = val[i];  //
     for node value
  seg.v[position[i]] = wt_from_parent[i
     ]; // for edge value
}

// work on a specific edge
for (int i = 1; i < n; i++) {
  ll u = edge_u[i], v = edge_v[i];
  edge_node[i] = (depth[u] > depth[v])
     ? u : v;
}

seg.build(1, 1, n);

auto updatePath = [&](ll u, ll v, ll x)
    {
  while (head[u] != head[v]) {
    if (depth[head[u]] < depth[head[v
       ]])
      swap(u, v);
    seg.update(1, 1, n, position[head[u
       ]], position[u], x);
    u = par[head[u]];
  }
  if (depth[u] > depth[v])
    swap(u, v);
  // edge value
  if (u != v) {
    seg.update(1, 1, n, position[u] +
       1, position[v], x);
  }
  // node value
  // seg.update(1, 1, n, position[u],
     position[v], x);
};
auto querypath = [&](ll u, ll v) {
  ll ans = -inf;
  while (head[u] != head[v]) {
    if (depth[head[u]] < depth[head[v
       ]])
      swap(u, v);
    ans = max(ans, seg.query(1, 1, n,
       position[head[u]], position[u]))
       ;
    u = par[head[u]];
  }
  if (depth[u] > depth[v])
    swap(u, v);
  // upward + downward
  if (u != v) {
    ans = max(ans, seg.query(1, 1, n,
       position[u] + 1, position[v]));
  }
  // only upward
//   ans = max(ans, seg.query(1, 1, n,
   position[u], position[v])); // for
   node value
  return ans;
};
seg.update(1, 1, n, position[edge_node[
   s]], position[edge_node[s]], x); //
   single point update. if path update
   need call update path
cout << querypath(x, s) << '\n';
```

2.11 Manacher

```
struct Manacher {
  vector<ll> p[2];
  string s;
  // p[1][i] = (max odd length
     palindrome centered at i) / 2 [
     floor division]
  // p[0][i] = same for even, it
     considers the right center
  // e.g. for s = "abbabba", p[1][3] =
     3, p[0][2] = 2
  Manacher(string s) {
    this->s = s;
    ll n = s.size();
    p[0].resize(n + 1);
    p[1].resize(n);
    for (ll z = 0; z < 2; z++) {
      for (ll i = 0, l = 0, r = 0; i <
         n; i++) {
        ll t = r - i + !z;
        if (i < r)
          p[z][i] = min(t, p[z][l + t])
             ;
        ll L = i - p[z][i], R = i + p[z
           ][i] - !z;
        while (L >= 1 && R + 1 < n && s
           [L - 1] == s[R + 1])
          p[z][i]++, L--, R++;
        if (R > r)
          l = L, r = R;
      }
    }
  }
  bool is_palindrome(ll l, ll r) {
    ll mid = (l + r + 1) / 2, len = r -
       l + 1;
    return 2 * p[len % 2][mid] + len %
       2 >= len;
  }
  string get_palin(ll i, bool odd =
     true) {
    ll len = p[odd][i];
    return s.substr(i - len, 2 * len +
       1 - !odd);
  }
};
```

3 Dynamic Programming

3.1 LCS

```
/*
```

```
Fact about LCS:
1. Longest Increasing Substring
To solve this, we just care about when
    two char equals. Rest of the things
    should be neglected.
2. Longest Palindromic Subsequence(LPS)
To solve this, we just take a new
    string which is the reverse of the
    original string. Then just call the
    LCS function to find LPS.
3. Minimum insertions to make a string
    palindrome To solve this, we just
    basically do string length - LPS.
    Why this?
     Let's take an example: string s =
         aabca; Let's say aca is our LPS.
          Now we find how many char we
         need to insert to make the
         string palindrome while our LPS
         is fixed.
     a ab c a now to make the string
         palindrome we just need to
         insert the reverse of ab after c
         . So the new string looks like a
          ab c ba a
4. Minimum Number of Deletions and
    Insertions to make the string equals
     To solve this we just find the LCS
    of those string then just do: n + m
    - 2 * LCS.length() where n, m =
    strings length
*/
```

3.2 MCM

```
// TC: O(n ^ 3)
const ll N = 1005;
vector<ll> v;
ll dp[N][N], mark[N][N];
ll MCM(ll i, ll j) {
  if (i == j)
    return dp[i][j] = 0;
  if (dp[i][j] != -1)
    return dp[i][j];
  ll mn = INT_MAX;
  for (ll k = i; k < j; k++) {
    ll x = mn;
    mn = min(mn, MCM(i, k) + MCM(k + 1,
        j) + v[i - 1] * v[k] * v[j]);
    if (x != mn)
      mark[i][j] = k;
  }
  return dp[i][j] = mn;
}

void print_order(ll i, ll j) {
  if (i == j)
    cout << "X" << i;
  else {
    cout << "(";
    print_order(i, mark[i][j]);
    print_order(mark[i][j] + 1, j);
    cout << ")";
  }
}
// memset(dp, -1, sizeof dp);
// print_order(1, n);
```

3.3 LIS_length

```
vector<ll> v = {7, 3, 5, 3, 6, 2, 9,
    8};
vector<ll> seq;
/*
here we basically check is the current
    element from v is greater than the
    last element of the sequence. if it
    is then push it to the seq array and
     if not then replace that index
    value. let's take an example:
v = 7 3 5 3 6 2 9 8
1st iteration seq = 7;
2nd iteration seq = 3;
3rd iteration seq = 3 5;
4th iteration seq = 3 3;
5th iteration seq = 3 3 6;
6th iteration seq = 2 3 6;
7th iteration seq = 2 3 6 9;
8th iteration seq = 2 3 6 8;
*/
for (auto i : v) {
  auto id = lower_bound(seq.begin(),
      seq.end(), i);
  if (id == seq.end())
    seq.push_back(i);
  else
    seq[id - seq.begin()] = i;
}
cout << seq.size() << endl;
```

4 Graph

4.1 Dijkstra

```
// TC: O(V + ElogV)
typedef pair<ll, ll> pairi;
ll N = 20000 + 5;
vector<vector<pairi>> adj(N);
vector<ll> dis(N, inf), parent(N);

void dijkstra(ll src) {
  priority_queue<pairi, vector<pairi>,
      greater<pairi>> pq;
  dis[src] = 0;
  pq.push({0, src});
  while (pq.size()) {
    auto top = pq.top();
    pq.pop();
    for (auto i : adj[top.second]) {
      ll v = i.first;
      ll wt = i.second;
      if (dis[v] > dis[top.second] + wt
          ) {
        dis[v] = dis[top.second] + wt;
        pq.push({dis[v], v});
        parent[v] = top.second
      }
    }
  }
}
ll node = n;
while (parent[node] != node) {
  path.push_back(node);
  node = parent[node];
}
path.push_back(1);
```

4.2 BellmanFord

```
// TC : O(V.E)
vector<ll> dist;
vector<ll> parent;
vector<vector<pair<ll, ll>>> adj;
// resize the vectors from main
    function

void bellmanFord(ll n, ll src) {
  dist[src] = 0;
  for (ll step = 0; step < n; step) {
    for (ll i = 1; i <= n; i++) {
      for (auto it : adj[i]) {
        ll u = i;
        ll v = it.first;
        ll wt = it.second;
        if (dist[u] != inf && ((dist[u]
            + wt) < dist[v])) {
          if (step == n - 1) {
            cout << "Negative cycle
                found\n ";
            return;
          }
          dist[v] = dist[u] + wt;
          parent[v] = u;
        }
      }
    }
  }
  for (ll i = 1; i <= n; i++)
    cout << dist[i] << " ";
  cout << endl;
}
```

4.3 FloydWarshall

```
// TC : O(n ^ 3)
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
typedef vector<ll> VI;
typedef vector<VI> VVI;

bool FloydWarshall(VVT &w, VVI &prev) {
  ll n = w.size();
  prev = VVI(n, VI(n, -1));

  for (ll k = 0; k < n; k++) {
    for (ll i = 0; i < n; i++) {
      for (ll j = 0; j < n; j++) {
        if (w[i][j] > w[i][k] + w[k][j
           ]) {
          w[i][j] = w[i][k] + w[k][j];
          prev[i][j] = k;
        }
      }
    }
  }

  // check for negative weight cycles
  for (ll i = 0; i < n; i++)
    if (w[i][i] < 0)
      return false;
  return true;
}
```

4.4 Toposort

```
// TC : O(V + E)
map<ll, vector<ll>> adj;
map<ll, ll> degree;
set<ll> nodes;
vector<ll> ans;
// adj: graph input, degree: cnt
   indegree,
// node: unique nodes, ans: path
ll c = 0;
void topo_sort() {
  queue<ll> qu;
  // traverse all the nodes and check
     if its degree is 0 or not..
  for (ll i : nodes) {
    if (degree[i] == 0)
      qu.push(i);
  }
  while (!qu.empty()) {
    ll top = qu.front();
    qu.pop();
    ans.push_back(top);
    for (ll i : adj[top]) {
      degree[i]--;
      if (degree[i] == 0) {
        qu.push(i);
      }
    }
  }
}
```

4.5 Kruskal

```
// TC : O(ElogE)
typedef pair<ll, ll> edge;
class Graph {
  vector<pair<ll, edge>> G, T;
  vector<ll> parent;
  ll cost = 0;

public:
  Graph(ll n) {
    for (ll i = 0; i < n; i++)
      parent.push_back(i);
  }
  void add_edges(ll u, ll v, ll wt) { G
     .push_back({wt, {u, v}}); }
  ll find_set(ll n) {
    if (n == parent[n])
      return n;
    else
      return find_set(parent[n]);
  }

  void union_set(ll u, ll v) { parent[u
     ] = parent[v]; }

  void kruskal() {
    sort(G.begin(), G.end());
    for (auto it : G) {
      ll uRep = find_set(it.second.
         first);
      ll vRep = find_set(it.second.
         second);
      if (uRep != vRep) {
        cost += it.first;
        T.push_back(it);
        union_set(uRep, vRep);
      }
    }
  }
  ll get_cost() { return cost; }
  void print() {
    for (auto it : T)
      cout << it.second.first << " " <<
          it.second.second << ": " <<
         it.first << endl;
  }
};

// g.add_edges(u, v, wt);
// g.kruskal();
```

4.6 Prims

```
// TC: O(ElogV)
typedef pair<ll, ll> pll;
class Prims {
  map<ll, vector<pll>> graph;
  map<ll, ll> visited;

public:
  void addEdge(ll u, ll v, ll w) {
    graph[u].push_back({v, w});
    graph[v].push_back({u, w});
  }

  vector<ll> path(pll start) {
    vector<ll> ans;
    priority_queue<pll, vector<pll>,
       greater<pll>> pq;
    // cost vs node
    pq.push({start.second, start.first
       });
    while (!pq.empty()) {
      pair<ll, ll> curr = pq.top();
      pq.pop();
      if (visited[curr.second])
        continue;
      visited[curr.second] = 1;
      ans.push_back(curr.second);
      for (auto i : graph[curr.second])
          {
        if (visited[i.first])
          continue;
        pq.push({i.second, i.first});
      }
    }
    return ans;
  }
};
```

4.7 LCA

```
// TC: preprocessing O(nlogn), each
   query O(logn)
ll n, l;
vector<vector<ll>> adj;
ll timer;
```

```
vector<ll> tin, tout;
vector<vector<ll>> up;

void dfs(ll v, ll p) {
  tin[v] = ++timer;
  up[v][0] = p;
  for (ll i = 1; i <= l; ++i)
    up[v][i] = up[up[v][i - 1]][i - 1];

  for (ll u : adj[v]) {
    if (u != p)
      dfs(u, v);
  }

  tout[v] = ++timer;
}

bool is_ancestor(ll u, ll v) { return
    tin[u] <= tin[v] && tout[u] >= tout[
    v]; }

ll lca(ll u, ll v) {
  if (is_ancestor(u, v))
    return u;
  if (is_ancestor(v, u))
    return v;
  for (ll i = l; i >= 0; --i) {
    if (!is_ancestor(up[u][i], v))
      u = up[u][i];
  }
  return up[u][0];
}

void preprocess(ll root) {
  tin.resize(n);
  tout.resize(n);
  timer = 0;
  l = ceil(log2(n));
  up.assign(n, vector<ll>(l + 1));
  dfs(root, root);
}
```

4.8 Rerooting

```
namespace reroot {
const auto exclusive = [](const auto &a
    , const auto &base,
                          const auto &
                              merge_into
                              , int
                              vertex) {
  int n = (int)a.size();
  using Aggregate = decay_t<decltype(
      base)>;
  vector<Aggregate> b(n, base);
  for (int bit = (int)__lg(n); bit >=
      0; --bit) {
    for (int i = n - 1; i >= 0; --i)
      b[i] = b[i >> 1];
    int sz = n - (n & !bit);
    for (int i = 0; i < sz; ++i) {
      int index = (i >> bit) ^ 1;
      b[index] = merge_into(b[index], a
          [i], vertex, i);
    }
  }
  return b;
};
// MergeInto : Aggregate * Value *
    Vertex(int) * EdgeIndex(int) ->
    Aggregate
// Base : Vertex(int) -> Aggregate
// FinalizeMerge : Aggregate * Vertex(
    int) * EdgeIndex(int) -> Value
const auto rerooter = [](const auto &g,
     const auto &base,
                        const auto &
                            merge_into,
                            const auto
                            &
                            finalize_merge
                            ) {
  int n = (int)g.size();
  using Aggregate = decay_t<decltype(
      base(0))>;
  using Value = decay_t<decltype(
      finalize_merge(base(0), 0, 0))>;
  vector<Value> root_dp(n), dp(n);
  vector<vector<Value>> edge_dp(n),
      redge_dp(n);

  vector<int> bfs, parent(n);
  bfs.reserve(n);
  bfs.push_back(0);
  for (int i = 0; i < n; ++i) {
    int u = bfs[i];
    for (auto v : g[u]) {
      if (parent[u] == v)
        continue;
      parent[v] = u;
      bfs.push_back(v);
    }
  }

  for (int i = n - 1; i >= 0; --i) {
    int u = bfs[i];
    int p_edge_index = -1;
    Aggregate aggregate = base(u);
    for (int edge_index = 0; edge_index
         < (int)g[u].size(); ++
        edge_index) {
      int v = g[u][edge_index];
      if (parent[u] == v) {
        p_edge_index = edge_index;
        continue;
      }
      aggregate = merge_into(aggregate,
          dp[v], u, edge_index);
    }
    dp[u] = finalize_merge(aggregate, u
        , p_edge_index);
  }

  for (auto u : bfs) {
    dp[parent[u]] = dp[u];
    edge_dp[u].reserve(g[u].size());
    for (auto v : g[u])
      edge_dp[u].push_back(dp[v]);
    auto dp_exclusive = exclusive(
        edge_dp[u], base(u), merge_into,
         u);
    redge_dp[u].reserve(g[u].size());
    for (int i = 0; i < (int)
        dp_exclusive.size(); ++i)
      redge_dp[u].push_back(
          finalize_merge(dp_exclusive[i
          ], u, i));
    root_dp[u] = finalize_merge(
        n > 1 ? merge_into(dp_exclusive
            [0], edge_dp[u][0], u, 0) :
            base(u), u,
        -1);
    for (int i = 0; i < (int)g[u].size
        (); ++i) {
      dp[g[u][i]] = redge_dp[u][i];
    }
  }

  return make_tuple(move(root_dp), move
      (edge_dp), move(redge_dp));
};
} // namespace reroot

int main() {
  ll n;
  cin >> n;
  vector<vector<ll>> g(n);
  // everything should be 0 based.

  using Aggregate = int;
  using Value = int;

  auto base = [](int vertex) ->
      Aggregate {
    // task here
  };
  auto merge_into = [](Aggregate
      vertex_dp, Value neighbor_dp, int
      vertex, int edge_index) ->
      Aggregate {
    // task here
  };
  auto finalize_merge = [](Aggregate
      vertex_dp, int vertex, int
      edge_index) -> Value {
    // task here
  };

  auto [reroot_result, edge_dp,
      redge_dp] = reroot::rerooter(g,
      base, merge_into, finalize_merge);
}
```

5 Number Theory

5.1 Leap_year

```
bool isLeap(ll n) {
  if (n % 100 == 0)
    return (n % 400 == 0);
  else
    return (n % 4 == 0);
}
// leap year between l and r
ll calNum(ll y) { return (y / 4) - (y /
     100) + (y / 400); }
ll leapNum(ll l, ll r) { return calNum(
    r) - calNum(--l); }
```

5.2 Print_calender

```
ll dayNumber(ll day, ll month, ll year)
     {
  ll t[] = {0, 3, 2, 5, 0, 3, 5, 1, 4,
      6, 2, 4};
  year -= month < 3;
  return (year + year / 4 - year / 100
      + year / 400 + t[month - 1] + day)
       % 7;
}
string getMonthName(ll monthNumber) {
  string months[] = {"January", "
      February", "March", "April", "May"
      , "June" "July", "August", "
      September", "October", "November",
       "December"};
  return (months[monthNumber]);
}
ll numberOfDays(ll monthNumber, ll year
    ) {
  if (monthNumber == 1 && isLeapYear(
      year))
    return 29;
  ll monthDays[] = {31, 28, 31, 30, 31,
       30, 31, 31, 30, 31, 30, 31};
  return (monthDays[monthNumber]);
}
void prllCalendar(ll year) {
  printf("       Calendar - %d\n\n",
      year);
  ll days;
  ll current = dayNumber(1, 1, year);
  // i: Iterate through all the months
  // j: Iterate through all the days of
       the month - i
  for (ll i = 0; i < 12; i++) {
    days = numberOfDays(i, year);
    cout << "          |" <<
        getMonthName(i).c_str() << "|"
        << endl;
    printf(" Sun Mon Tue Wed Thu Fri
        Sat\n");
    ll k;
    for (k = 0; k < current; k++)
      printf("    ");
    for (ll j = 1; j <= days; j++) {
      printf("%4d", j);
      if (++k > 6) {
        k = 0;
        cout << endl;
      }
    }
    if (k)
      cout << endl;
    cout << "
        -----------------------------\n";
    current = k;
  }
}
```

5.3 Binary_exponentiation

```
ll binaryExp(ll base, ll power, ll MOD
    = mod) {
  ll res = 1;
  while (power) {
    if (power & 1)
      res = (res * base) % MOD;
    base = ((base % MOD) * (base % MOD)
        ) % MOD;
    power /= 2;
  }
  return res;
}
/*
task: a ^ b ^ c
binaryExp(a, binaryExp(b, c, mod - 1),
    mod)
*/
```

5.4 Count_divisor

```
ll maxVal = 1e6 + 1;
vector<ll> countDivisor(maxVal, 0);
void countingDivisor() {
  for (ll i = 1; i < maxVal; i++)
    for (ll j = i; j < maxVal; j += i)
      countDivisor[j]++;
}
// TC: nlog(n)
// count the number of divisors of all
    numbers in a range.
```

5.5 Check_prime

```
bool prime(ll n) {
  if (n < 2)
    return false;
  if (n <= 3)
    return true;
  if (!(n % 2) || !(n % 3))
    return false;
  for (ll i = 5; i * i <= n; i += 6) {
    if (!(n % i) || !(n % (i + 2)))
      return false;
  }
  return true;
}
// TC: sqrt(n) / 6;
```

5.6 SPF

```
// smallest prime factor using seive
const ll N = 1e7 + 5;
ll spf[N];
void smallestPrimeFactorUsingSeive() {
  for (ll i = 2; i < N; i++) {
    if (spf[i] == 0) {
      for (ll j = i; j < N; j += i) {
        if (spf[j] == 0)
          spf[j] = i;
      }
    }
  }
}

// smallest factor of a number
ll factor(ll n) {
  ll a;
  if (n % 2 == 0)
    return 2;
  for (a = 3; a * a <= n; a += 2) {
    if (n % a == 0)
      return a;
  }
  return n;
}
// complete factorization
ll r;
while (n > 1) {
  r = factor(n);
  cout << r << '\n';
  n /= r;
}
```

5.7 Seive

```
const ll N = 1e7 + 5;
ll prime[N];
void sieveOfEratosthenes() {
  for (ll i = 2; i < N; i++)
    prime[i] = 1;
  for (ll i = 4; i < N; i += 2)
    prime[i] = 0;
  for (ll i = 3; i * i < N; i++) {
    if (prime[i]) {
      for (ll j = i * i; j < N; j += i * 2)
        prime[j] = 0;
    }
  }
}
```

5.8 Optimize_seive

```
vector<ll> sieve(const ll N, const ll Q = 17, const ll L = 1 << 15) {
  static const ll rs[] = {1, 7, 11, 13, 17, 19, 23, 29};
  struct P {
    P(ll p) : p(p) {}
    ll p;
    ll pos[8];
  };
  auto approx_prime_count = [](const ll N) -> ll {
    return N > 60184 ? N / (log(N) - 1.1) : max(1., N / (log(N) - 1.11)) + 1;
  };
  const ll v = sqrt(N), vv = sqrt(v);
  vector<bool> isp(v + 1, true);
  for (ll i = 2; i <= vv; ++i)
    if (isp[i]) {
      for (ll j = i * i; j <= v; j += i)
        isp[j] = false;
    }
  const ll rsize = approx_prime_count(N + 30);
  vector<ll> primes = {2, 3, 5};
  ll psize = 3;
  primes.resize(rsize);

  vector<P> sprimes;
  size_t pbeg = 0;
  ll prod = 1;
  for (ll p = 7; p <= v; ++p) {
    if (!isp[p])
      continue;
    if (p <= Q)
      prod *= p, ++pbeg, primes[psize++] = p;
    auto pp = P(p);
    for (ll t = 0; t < 8; ++t) {
      ll j = (p <= Q) ? p : p * p;
      while (j % 30 != rs[t])
        j += p << 1;
      pp.pos[t] = j / 30;
    }
    sprimes.push_back(pp);
  }

  vector<unsigned char> pre(prod, 0xFF);
  for (size_t pi = 0; pi < pbeg; ++pi) {
    auto pp = sprimes[pi];
    const ll p = pp.p;
    for (ll t = 0; t < 8; ++t) {
      const unsigned char m = ~(1 << t);
      for (ll i = pp.pos[t]; i < prod; i += p)
        pre[i] &= m;
    }
  }
  const ll block_size = (L + prod - 1) / prod * prod;
  vector<unsigned char> block(block_size);
  unsigned char *pblock = block.data();
  const ll M = (N + 29) / 30;

  for (ll beg = 0; beg < M; beg += block_size, pblock -= block_size) {
    ll end = min(M, beg + block_size);
    for (ll i = beg; i < end; i += prod) {
      copy(pre.begin(), pre.end(), pblock + i);
    }
    if (beg == 0)
      pblock[0] &= 0xFE;
    for (size_t pi = pbeg; pi < sprimes.size(); ++pi) {
      auto &pp = sprimes[pi];
      const ll p = pp.p;
      for (ll t = 0; t < 8; ++t) {
        ll i = pp.pos[t];
        const unsigned char m = ~(1 << t);
        for (; i < end; i += p)
          pblock[i] &= m;
        pp.pos[t] = i;
      }
    }
    for (ll i = beg; i < end; ++i) {
      for (ll m = pblock[i]; m > 0; m &= m - 1) {
        primes[psize++] = i * 30 + rs[__builtin_ctz(m)];
      }
    }
  }
  assert(psize <= rsize);
  while (psize > 0 && primes[psize - 1] > N)
    --psize;
  primes.resize(psize);
  return primes;
}
// it takes 500ms for generating prime upto 1e9
```

5.9 nth_prime_number

```
vector<ll> nth_prime;
const ll MX = 86200005;
bitset<MX> visited;
void optimized_prime() {
  nth_prime.push_back(2);
  for (ll i = 3; i < MX; i += 2) {
    if (visited[i])
      continue;
    nth_prime.push_back(i);
    if (1ll * i * i > MX)
      continue;
    for (ll j = i * i; j < MX; j += i + i)
      visited[j] = true;
  }
}
```

5.10 nCr

```
// 1:
// more space, less time
const ll MAX = 1e7 + 5;
vector<ll> fact(MAX), ifact(MAX), inv(MAX);
```

```
void factorial() {
  inv[1] = fact[0] = ifact[0] = 1;
  for (ll i = 2; i < MAX; i++)
    inv[i] = inv[mod % i] * (mod - mod
        / i) % mod;
  for (ll i = 1; i < MAX; i++)
    fact[i] = (fact[i - 1] * i) % mod;
  for (ll i = 1; i < MAX; i++)
    ifact[i] = ifact[i - 1] * inv[i] %
        mod;
}
ll nCr(ll n, ll r) {
  if (r < 0 || r > n)
    return 0;
  return (ll)fact[n] * ifact[r] % mod *
      ifact[n - r] % mod;
}
// 2:
// less space, more time
const ll MAX = 1e7 + 10;
vector<ll> fact(MAX), inv(MAX);
void factorial() {
  fact[0] = 1;
  for (ll i = 1; i < MAX; i++)
    fact[i] = (i * fact[i - 1]) % mod;
}
ll binaryExp(ll a, ll n, ll M = mod){};
    // needs to implement
void inverse() {
  for (ll i = 0; i < MAX; ++i)
    inv[i] = binaryExp(fact[i], mod -
        2);
}
ll nCr(ll a, ll b) {
  if (a < b or a < 0 or b < 0)
    return 0;
  ll de = (inv[b] * inv[a - b]) % mod;
  return (fact[a] * de) % mod;
}
// 3:
// nCr mod m where m is not prime
ll C_mod_p(ll n, ll k, ll p) {
  if (k > n)
    return 0;
  vector<ll> fac(p);
  fac[0] = 1;
  for (int i = 1; i < p; i++)
    fac[i] = fac[i - 1] * i % p;

  ll res = 1;
  while (n || k) {
    ll ni = n % p, ki = k % p;
    if (ki > ni)
      return 0;
    res = res * fac[ni] % p * modInv(
        fac[ki], p) % p * modInv(fac[ni
        - ki], p) %
          p;
    n /= p;
    k /= p;
  }
  return res;
}
// compute nCr mod composite m (non-
    prime)
ll nCr_mod_m(ll n, ll k, ll m) {
  // Step 1: factorize m
  vector<int> primes;
  int tmp = m;
  for (int i = 2; i * i <= tmp; i++) {
    if (tmp % i == 0) {
      primes.push_back(i);
      while (tmp % i == 0)
        tmp /= i;
    }
  }
  if (tmp > 1)
    primes.push_back(tmp);

  // Step 2: compute result mod each
      prime
  vector<ll> rem, mod;
  for (int p : primes) {
    rem.push_back(C_mod_p(n, k, p));
    mod.push_back(p);
  }
  // Step 3: Chinese Remainder Theorem
      (combine)
  ll res = 0;
  for (int i = 0; i < (int)mod.size();
      i++) {
    ll Mi = m / mod[i];
    ll invMi = binaryExp(Mi, mod[i] -
        2, mod[i]); // modular inverse
    res = (res + rem[i] * Mi % m *
        invMi % m) % m;
  }
  return res;
}
```

5.11 Factorial_mod

```
// n! mod p : Here P is mod value
// For binaryExp we call 1.6 function
ll factmod(ll n, ll p) {
  ll res = 1;
  while (n > 1) {
    res = (res * binaryExp(p - 1, n / p
        , p)) % p;
    for (ll i = 2; i <= n % p; ++i)
      res = (res * i) % p;
    n /= p;
  }
  return (res % p);
}
```

5.12 PHI

```
// the positive integers less than or
    equal to n that are relatively prime
     to n.
ll phi(ll n) {
  ll result = n;
  for (ll i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      while (n % i == 0)
        n /= i;
      result -= result / i;
    }
  }
  if (n > 1)
    result -= result / n;
  return result;
}
// PHI of 1 to N
const int N = 1e6 + 9;
int phi[N];
int phiS[N];
void totient() {
  for (int i = 1; i < N; i++)
    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;
    }
  }
  phiS[0] = phi[0];
  for (int i = 1; i < N; i++)
    phiS[i] = phiS[i - 1] + phi[i];
}
```

5.13 Catalan

```
void catalan(ll n) {
  ll res = 1;
  cout << res << " ";
  for (ll i = 1; i < n; i++) {
    res = (res * (4 * i - 2)) / (i + 1)
      ;
    cout << res << " ";
  }
}
```

5.14 Extended_GCD

```
// return {x,y} such that ax + by = gcd
   (a,b)
ll extended_euclid(ll a, ll b, ll &x,
   ll &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  ll x1, y1;
  ll d = extended_euclid(b, a % b, x1,
     y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
ll inverse(ll a, ll m) {
  ll x, y;
  ll g = extended_euclid(a, m, x, y);
  if (g != 1)
    return -1;
  return (x % m + m) % m;
}
```

5.15 Large Mod

```
ll mod(string &num, ll a) {
  ll res = 0;
  for (ll i = 0; i < num.length(); i++)
    res = (res * 10 + num[i] - '0') % a
       ;
  return res;
}
```

5.16 Factorial_Divisor

```
ll factorialDivisors(ll n) {
  ll result = 1;
  for (ll i = 0; i < allPrimes.size();
     i++) {
    ll p = allPrimes[i];
    ll exp = 0;
    while (p <= n) {
      exp = exp + (n / p);
      p = p * allPrimes[i];
    }
    result = result * (exp + 1);
  }
  return result;
}
```

5.17 Number_conversion

```
// 10 - ary to m - ary
char a[16] = {'0', '1', '2', '3', '4',
   '5', '6', '7', '8', '9', 'A', 'B', '
   C', 'D', 'E', 'F'};
string tenToM(ll n, ll m) {
  ll temp = n;
  string result = "";
  while (temp != 0) {
    result = a[temp % m] + result;
    temp /= m;
  }
  return result;
}
```

```
// m - ary to 10 - ary
string num = "0123456789ABCDE";
ll mToTen(string n, ll m) {
  ll multi = 1;
  ll result = 0;
  for (ll i = n.size() - 1; i >= 0; i
     --) {
    result += num.find(n[i]) * multi;
    multi *= m;
  }
  return result;
}
```

5.18 Number_of_1_in_bit_till_N

```
ll cntOnes(ll n) {
  ll cnt = 0;
  for (ll i = 1; i <= n; i <<= 1) {
    ll x = (n + 1) / (i << 1);
    cnt += x * i;
    if ((n + 1) % i && n & i)
      cnt += (n + 1) % i;
  }
  return cnt;
}
```

5.19 Disarrangement

```
ll disarrange(ll n) {
  if (n == 1)
    return 0;
  if (n == 2)
    return 1;
  return (n - 1) * (disarrange(n - 1) +
     disarrange(n - 2));
}
// D(n) = (n!)/e
```

5.20 Millar_Rabin

```
bool check_composite(ll n, ll a, ll d,
   ll s) {
  ll x = binaryExp(a, d, n);
  if (x == 1 || x == n - 1)
    return false;
  for (ll r = 1; r < s; r++) {
    x = (u128)x * x % n;
    if (x == n - 1)
      return false;
  }
  return true;
};
bool MillerRabin(ll n, ll iter = 5) {
  // returns true if n is probably
     prime, else returns false.
  if (n < 4)
    return n == 2 || n == 3;
  ll s = 0;
  ll d = n - 1;
  while ((d & 1) == 0) {
    d >>= 1;
    s++;
  }
  for (ll i = 0; i < iter; i++) {
    ll a = 2 + rand() % (n - 3);
    if (check_composite(n, a, d, s))
      return false;
  }
  return true;
}
```

5.21 Modular_operation

```
// Addition :
ll mod_add(ll a, ll b, ll MOD = mod) {
  a = a % MOD, b = b % MOD;
  return (((a + b) % MOD) + MOD) % MOD;
}
// Subtraction :
ll mod_sub(ll a, ll b, ll MOD = mod) {
  a = a % MOD, b = b % MOD;
  return (((a - b) % MOD) + MOD) % MOD;
}
// Multiplication :
ll mod_mul(ll a, ll b, ll MOD = mod) {
  a = a % MOD, b = b % MOD;
  return (((a * b) % MOD) + MOD) % MOD;
}
// Division :
ll mminvprime(ll a, ll b) { return
   binaryExp(a, b - 2, b); }
ll mod_div(ll a, ll b, ll MOD = mod) {
  a = a % MOD, b = b % MOD;
  return (mod_mul(a, mminvprime(b, MOD)
     , MOD) + MOD) % MOD;
}
```