CP Template

Prepared By: sagorahmedmunna

Contents

1 Custom Codes

```
freopen("input.in", "r", stdin);
freopen("output.out", "w", stdout);
ios_base::sync_with_stdio(0), cin.tie(0);
// pragma
#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2,fma")
#pragma GCC optimize ("O3,unroll-loops")
#pragma GCC optimize ("-ffloat-store")
#pragma GCC target ("bmi,bmi2,lzcnt,popcnt")
#pragma GCC target
↪  ("sse,sse2,sse3,ssse3,sse4,abm,mmx")
// random number generator
mt19937_64
↪  rng(chrono::steady_clock::now().time_since_epoch().count());
int num = rng() % limit;
int64_t rnd(int64_t l, int64_t r) { // can handle
↪  negative range also
  if (l > r) swap(l, r);
  return l + (rng() % (r - l + 1));
}
// ignore white space
cin.ignore(); or cin >> ws;
// STL merge function: merge two vectors or sets
vector<int> a = {1, 2, 7}, b = {4, 2, 3};
sort(a.begin(), a.end());
sort(b.begin(), b.end());
vector<int> c(a.size() + b.size());
merge(a.begin(), a.end(), b.begin(), b.end(),
↪  c.begin());

set<int> a = {3, 2, 1}, b = {4, 5, 1};
set<int> c;
merge(a.begin(), a.end(), b.begin(), b.end(),
↪  inserter(c, c.begin()));
// tuple
tuple<int, int, int> tup = {1, 2, 3};
int first = get<0>(tup);
// custom compare function
struct item {
  int a, b;
};
bool cmp(item& a, item& b) {
  if(a.a != b.a) return a.a < b.a;
  return a.b > b.b;
}
bool custom(pair<int, int>& a, pair<int, int>& b) {
  if(a.first != b.first) return a.first > b.first;
  return a.second < b.second;
}
// custom compare in set
struct cmp{
  bool operator() (const pair<int, int>& a, const
  ↪  pair<int, int>& b) const {
    if (a.first != b.first) return a.first > b.first;
    return a.second < b.second;
  }
};
set<pair<int, int>, cmp> a;
// greater functions
priority_queue<int, vector<int>, greater<int>>
set<int, greater<int>>
map<int, int, greater<int>>
// fill with elements
fill(v.begin(), v.end(), -1);
// fill for array
fill(&dp[0][0][0], &dp[0][0][0] + N * M * N, INF);
// string to number
stoi(num);
// number to string
to_string(num);
// Binary to decimal
int n = stoi(binary, 0, 2);
// decimal to binary
string binary = bitset<64>(n).to_string();
binary.erase(0, binary.find_first_not_of('0'));
// count digit
int d = log10(num) + 1;
// set ith bit
n | (1 << i)
// unset ith bit
n & ~(1 << i)
// toggle or inverse ith bit
n ^ (1 << i)
__builtin_popcountll(x)
__builtin_clzll(x)
__builtin_ctzll(x)
```

2 Articulation Bridge

```
vector<int> adj[N];
int t = 0;
vector<int> tin(N, -1), low(N);
vector<array<int, 2>> ab;
void dfs (int u, int p) {
  tin[u] = low[u] = t++;
  for (int v: adj[u]) {
    if (v != p) {
      if (tin[v] != -1) {
        low[u] = min(low[u], tin[v]);
      } else {
        dfs(v, u);
        if (tin[u] < low[v]) {
          ab.push_back({u, v});
        }
        low[u] = min(low[u], low[v]);
      }
    }
  }
}
```

3 Articulation Point

```
vector<int> adj[N];
int t = 0;
vector<int> tin(N, -1), low(N), ap;
void dfs (int u, int p) {
  tin[u] = low[u] = t++;
  int is_ap = 0, child = 0;
  for (int v: adj[u]) {
    if (v != p) {
      if (tin[v] != -1) {
        low[u] = min(low[u], tin[v]);
      } else {
        child++;
        dfs(v, u);
        if (tin[u] <= low[v]) {
          is_ap = 1;
        }
        low[u] = min(low[u], low[v]);
      }
    }
  }
  if ((u != p or child > 1) and is_ap) {
    ap.push_back(u);
  }
}
```

4 Bellman Ford

```
struct st {
  int a, b, cost;
} e[N];
const int INF = 2e9;
int32_t main() {
  int n, m;
  cin >> n >> m;
  for(int i = 0; i < m; i++) cin >> e[i].a >> e[i].b
  ↪  >> e[i].cost;
  int s;
  cin >> s;//is there any negative cycle which is
  ↪  reachable from s?
  vector<int> d (n, INF);//for finding any cycle(not
  ↪  necessarily from s) set d[i] = 0 for all i
  d[s] = 0;
  vector<int> p (n, -1);
  int x;
  for (int i=0; i<n; ++i) {
    x = -1;
    for (int j=0; j<m; ++j) {
      if (d[e[j].a] < INF) {
        if (d[e[j].b] > d[e[j].a] + e[j].cost) {
          d[e[j].b] = max (-INF, d[e[j].a] +
          ↪  e[j].cost);//for overflow
          p[e[j].b] = e[j].a;
          x = e[j].b;
        }
      }
    }
  }
  if (x == -1) cout << "No negative cycle from "<<s;
  else {
    int y = x; //x can be on any cycle or reachable
    ↪  from some cycle
    for (int i=0; i<n; ++i) y = p[y];
    vector<int> path;
    for (int cur=y; ; cur=p[cur]) {
      path.push_back (cur);
      if (cur == y && path.size() > 1) break;
```

```
    }
    reverse (path.begin(), path.end());
    cout << "Negative cycle: ";
    for (int i=0; i<path.size(); ++i) cout << path[i]
    ↪  << ' ';
  }
  return 0;
}
// ## Bellman-ford (V * E)
vector<int> bellman_ford(int s){
  vector<int> dis(n, I);
  dis[s]=0;
  while(1){
    int any=0;
    for (auto& e: ed){
      if(dis[e.u]<I){
        if(dis[e.u]+e.cost < dis[e.v]){
          dis[e.v] = dis[e.u]+e.cost;
          any=1;
        }
      }
    }
    if(!any)  break;
  }
  return dis;
}
```

5 Binary & Ternary Search

```
//Binary Search (integer)
int lo = 0, hi = n - 1;
while (lo <= hi) {
  int mid = (lo + hi) / 2;
  if (f(mid)) hi = mid - 1;
  else lo = mid + 1;
}
//Binary Search (double)
double lo = 0, hi = inf;
int itr = 50;
while (itr--) {
  double mid = (lo + hi) / 2;
  if (f(mid)) hi = mid;
  else lo = mid;
}
//Ternary Search (integer, max)
int lo = 0, hi = n - 1;
while (lo < hi) {
  // int mid1 = lo + (hi - lo) / 3;
  // int mid2 = hi - (hi - lo) / 3;
  int mid = (lo + hi) / 2;
  if (f(mid) < f(mid + 1)) lo = mid + 1;
  else  hi = mid;
}
//Ternary Search (double, max)
double lo = 0, hi = inf;
int itr = 50;
while (itr--) {
  double mid1 = lo + (hi - lo) / 3;
  double mid2 = hi - (hi - lo) / 3;
  if (f(mid1) < f(mid2)) lo = mid1;
  else  hi = mid2;
}
```

6 2D BIT Range Update & Range Query

```
const int N = 1009;
struct BIT2D {
  long long M[N][N][2], A[N][N][2];
  BIT2D() {
    memset(M, 0, sizeof M);
    memset(A, 0, sizeof A);
  }
  void upd2(long long t[N][N][2], int x, int y, long
  ↪  long mul, long long add) {
    for(int i = x; i < N; i += i & -i) {
      for(int j = y; j < N; j += j & -j) {
        t[i][j][0] += mul;
        t[i][j][1] += add;
      }
    }
  }
  void upd1(int x, int y1, int y2, long long mul, long
  ↪  long add) {
    upd2(M, x, y1, mul, -mul * (y1 - 1));
    upd2(M, x, y2, -mul, mul * y2);
    upd2(A, x, y1, add, -add * (y1 - 1));
    upd2(A, x, y2, -add, add * y2);
  }
  void upd(int x1, int y1, int x2, int y2, long long
  ↪  val) {
    upd1(x1, y1, y2, val, -val * (x1 - 1));
    upd1(x2, y1, y2, -val, val * x2);
  }
  long long query2(long long t[N][N][2], int x, int y)
  ↪  {
    long long mul = 0, add = 0;
    for(int i = y; i > 0; i -= i & -i) {
      mul += t[x][i][0];
      add += t[x][i][1];
    }
    return mul * y + add;
  }
  long long query1(int x, int y) {
    long long mul = 0, add = 0;
    for(int i = x; i > 0; i -= i & -i) {
      mul += query2(M, i, y);
      add += query2(A, i, y);
    }
    return mul * x + add;
  }
  long long query(int x1, int y1, int x2, int y2) {
    return query1(x2, y2) - query1(x1 - 1, y2) -
    ↪  query1(x2, y1 - 1) + query1(x1 - 1, y1 - 1);
  }
};
```

7 Centroid Decomposition

```
struct CentroidDecomposition {
  using T = vector<vector<int>>;
  int n;
  vector<int> sz, is_cen, cpar, cdep;
  CentroidDecomposition(T& adj, int root = 1) {
    n = (int)adj.size() + 1;
    sz.resize(n), is_cen.resize(n), cpar.resize(n),
    ↪  cdep.resize(n);
    Decompose(root, -1, 0, adj);
  }
  void Cal_sz(int u, int p, T& adj) {
    sz[u] = 1;
    for (auto& v : adj[u]) {
      if (v != p && !is_cen[v]) {
        Cal_sz(v, u, adj);
        sz[u] += sz[v];
      }
    }
  }
  int Get_cen(int u, int p, int csz, T& adj) {
    for (auto& v : adj[u]) {
      if (v != p && !is_cen[v] && (2 * sz[v] > csz)) {
        return Get_cen(v, u, csz, adj);
      }
    }
    return u;
  }
  void Decompose(int u, int p,  int d, T& adj) {
    Cal_sz(u, p, adj);
    int c = Get_cen(u, p, sz[u], adj);
    is_cen[c] = 1, cpar[c] = p, cdep[c] = d;
    for (auto& v : adj[c]) {
      if (!is_cen[v]) {
        Decompose(v, c, d + 1, adj);
      }
    }
  }
};
```

8 Closest Index Where Each Element in Distinct

```
// 0 based, closest index where each element is
↪  distinct
array<vector<int>, 2>
↪  closestDistinctElementIndex(vector<int>& a) {
  int n = a.size(), mxIdx = -1;
  map<int, int> mp;
  vector<int> l(n);
  for (int i = 0; i < n; i++) {
    if (mp.count(a[i])) mxIdx = max(mxIdx, mp[a[i]]);
    mp[a[i]] = i, l[i] = i - mxIdx;
  }
  mp.clear(), mxIdx = n;
  vector<int> r(n);
  for (int i = n - 1; i >= 0; i--) {
    if (mp.count(a[i])) mxIdx = min(mxIdx, mp[a[i]]);
    mp[a[i]] = i, r[i] = mxIdx - i;
  }
  return {l, r};
}
```

9 Closest Min Max

```
// closest left, right index where current element is max or min
// closest max or min element from the current element
array<vector<int>, 2> closest_min_element(vector<int> &a) {
  int n = a.size();
  vector<int> l(n), r(n);
  stack<int> st;
  st.push(-1);
  for (int i = 0; i < n; i++) {
    while (st.top() != -1 && a[st.top()] > a[i]) st.pop();
    l[i] = st.top() + 1; // closest index where current element is min
    // l[i] = st.top(); // closest min element index
    st.push(i);
  }
  while (!st.empty()) st.pop();
  st.push(n);
  for (int i = n - 1; i >= 0; i--) {
    while (st.top() != n && a[st.top()] >= a[i]) st.pop();
    r[i] = st.top() - 1; // closest index where current element is min
    // r[i] = st.top(); // closest min element index
    st.push(i);
  }
  return {l, r};
}
array<vector<int>, 2> closest_max_element(vector<int> &a) {
  int n = a.size();
  vector<int> l(n), r(n);
  stack<int> st;
  st.push(-1);
  for (int i = 0; i < n; i++) {
    while (st.top() != -1 && a[st.top()] < a[i]) st.pop();
    l[i] = st.top() + 1;
    // l[i] = st.top(); // closest max element index
    st.push(i);
  }
  while (!st.empty()) st.pop();
  st.push(n);
  for (int i = n - 1; i >= 0; i--) {
    while (st.top() != n && a[st.top()] <= a[i]) st.pop();
    r[i] = st.top() - 1;
    // r[i] = st.top(); // closest max element index
    st.push(i);
  }
  return {l, r};
}
```

10 Compress Array

```
void CompressArray(vector<int>& a) {
  auto b = a;
  sort(b.begin(), b.end());
  b.erase(unique(b.begin(), b.end()), b.end());
  for (auto& ai : a) {
    ai = lower_bound(b.begin(), b.end(), ai) - b.begin();
  }
}
```

11 Convex Hull

```
const double PI = acos((double)-1.0);
using ll = long long;
struct PT {
  ll x, y;
  bool operator < (const PT &p) const {
    return x == p.x ? y < p.y : x < p.x;
  }
};
ll area(PT a, PT b, PT c) {
  return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}
vector<PT> ConvexHull(vector<PT> p) {
  int n = p.size(), m = 0;
  if (n < 3) return p;
  vector<PT> hull(n + n);
  sort(p.begin(), p.end());
  for (int i = 0; i < n; ++i) {
    while (m > 1 and area(hull[m - 2], hull[m - 1], p[i]) <= 0) --m;
    hull[m++] = p[i];
  }
  for (int i = n - 2, j = m + 1; i >= 0; --i) {
    while (m >= j and area(hull[m - 2], hull[m - 1], p[i]) <= 0) --m;
    hull[m++] = p[i];
  }
  hull.resize(m - 1);
  return hull;
}
```

12 DFS with LCA

```
struct DFS {
  int n, k, t = 0;
  vector<int> tin, tout, depth, height, subtree_size, heavy;
  vector<bool> is_leaf;
  vector<vector<int>> parent;
  DFS() {}
  DFS(vector<vector<int>>& adj, int root = 1) {
    n = (int)adj.size() + 1;
    k = __lg(n) + 1;
    tin.resize(n), tout.resize(n), depth.resize(n), height.resize(n), subtree_size.resize(n), heavy.assign(n, - 1), is_leaf.assign(n, 1);
    parent = vector<vector<int>> (n, vector<int> (k));
    dfs(root, root, adj);
  }
  void dfs(int u, int p, vector<vector<int>>& adj) {
    tin[u] = ++t;
    subtree_size[u] = 1;
    parent[u][0] = p;
    for (int i = 1; i < k; i++) {
      if (parent[u][i - 1] != -1) parent[u][i] = parent[parent[u][i - 1]][i - 1];
      else parent[u][i] = -1;
    }
    for (auto& v : adj[u]) {
      if (v != p) {
        depth[v] = depth[u] + 1;
        is_leaf[u] = 0;
        dfs(v, u, adj);
        height[u] = max(height[u], height[v] + 1);
        subtree_size[u] += subtree_size[v];
        if (heavy[u] == -1 || subtree_size[heavy[u]] < subtree_size[v]) {
          heavy[u] = v;
        }
      }
    }
    tout[u] = ++t;
  }
  bool is_ancestor(int u, int v) {
    return tin[u] <= tin[v] && tout[v] <= tout[u];
  }
  int kth_parent(int u, int kth) {
    for (int i = k - 1; i >= 0; i--) {
      if (kth & (1 << i)) {
        u = parent[u][i];
        if (u == -1) return u;
      }
    }
    return u;
  }
  int lca(int u, int v) {
    if (is_ancestor(u, v)) return u;
    for (int i = k - 1; i >= 0; i--) {
      if (!is_ancestor(parent[u][i], v)) {
        u = parent[u][i];
      }
    }
    return parent[u][0];
  }
  int dis(int u, int v)
    return depth[u] + depth[v] - 2 * depth[lca(u, v)];
};
```

13 Dijkstra

```
using ll = long long;
const ll INF = 1.1e17;
vector<ll> Dijkstra(vector<vector<array<ll, 2>>>& adj, int source = 1) {
  int n = (int)adj.size();
  vector<ll> dis(n, INF);
  vector<bool> vis(n);
  dis[source] = 0;
```

```
  priority_queue<array<ll, 2>, vector<array<ll, 2>>,
  ↪ greater<array<ll, 2>>> pq;
  pq.push({0, source});
  while (!pq.empty()) {
    auto [d, u] = pq.top();
    pq.pop();
    if (vis[u]) continue;
    vis[u] = 1;
    for (auto& [v, c] : adj[u]) {
      if (dis[v] > d + c) {
        dis[v] = d + c;
        pq.push({dis[v], v});
      }
    }
  }
  return dis;
}
```

14 Digit DP

```
#2 countWithExactDigitCount // f(0, 0, 1, 0)
int dp[11][11][2][2];
int targetDigit, targetCnt;
int f(int i, int digitCnt, int tight, int started) {
  if (i == n) {
    return targetCnt == digitCnt;
  }
  auto& ret = dp[i][digitCnt][tight][started];
  if (~ret) return ret;
  ret = 0;
  int limit = tight ? (s[i] - '0') : 9;
  for (int d = 0; d <= limit; d++) {
    int newTight = tight && (d == limit);
    int newStarted = started || (d > 0);
    int newDigitCnt = digitCnt;
    if (newStarted) newDigitCnt += (d == targetDigit);
    ret += f(i + 1, newDigitCnt, newTight,
    ↪ newStarted);
  }
  return ret;
}
#6 countNumbersWithOnly1234 // f(0, 1, 0)
int dp[11][2][2];
int f(int i, int tight, int started) {
  if (i == n) {
    return started;
  }
  auto& ret = dp[i][tight][started];
  if (~ret) return ret;
  ret = 0;
  int limit = tight ? (s[i] - '0') : 9;
  for (int d = 0; d <= limit; d++) {
    int newTight = tight && (d == limit);
    int newStarted = started || (d > 0);
    if (d > 4) continue;
    if (!newStarted && d == 0) ret += f(i + 1,
    ↪ newTight, newStarted);
    else if (d >= 1 && d <= 4) ret += f(i + 1,
    ↪ newTight, newStarted);
  }
  return ret;
}
#8 countDistinctDigitNumbers // f(0, 0, 1, 0)
int dp[11][1025][2][2];
int f(int i, int mask, int tight, int started) {
  if (i == n) return started;
  auto& ret = dp[i][mask][tight][started];
  if (~ret) return ret;
  ret = 0;
  int limit = tight ? (s[i] - '0') : 9;
  for (int d = 0; d <= limit; d++) {
    int newTight = tight && (d == limit);
    int newStarted = started || (d > 0);
    int newMask = mask;
    if (newStarted) {
      if (mask & (1 << d)) continue;
      newMask |= (1 << d);
    }
    ret += f(i + 1, newMask, newTight, newStarted);
  }
  return ret;
}
#9 countWithAnyRepeatedDigits -> n -
↪ f(countDistinctDigitNumbers)
#10 countNumbersWithKDistinctDigits // f(0, 0, 1, 0)
int k, dp[11][1025][2][2];
int f(int i, int mask, int tight, int started) {
  if (i == n) return started &&
  ↪ (__builtin_popcount(mask) == k);
  auto& ret = dp[i][mask][tight][started];
  if (~ret) return ret;
  ret = 0;
  int limit = tight ? (s[i] - '0') : 9;
  for (int d = 0; d <= limit; d++) {
    int newTight = tight && (d == limit);
    int newStarted = started || (d > 0);
    int newMask = mask;
    if (newStarted) newMask |= (1 << d);
    ret += f(i + 1, newMask, newTight, newStarted);
  }
  return ret;
}
#11 count Numbers Divisible By k and Contains No Digit
↪ m // f(0, 0, 1, 0)
int dp[11][100][2][2];
int k, m;
int f(int i, int rem, int tight, int started) {
  if (i == n) return started && (rem == 0);
  auto& ret = dp[i][rem][tight][started];
  if (~ret) return ret;
  ret = 0;
  int limit = tight ? (s[i] - '0') : 9;
  for (int d = 0; d <= limit; d++) {
    if (d == m) continue;
    int newTight = tight && (d == limit);
    int newStarted = started || (d > 0);
    int newRem = (rem * 10 + d) % k;
    ret += f(i + 1, newRem, newTight, newStarted);
  }
  return ret;
}
--> find sum of integers and conunt at same time
#12 sum and count of all numbers x that has at most k
↪ distinct digits // f(0, 0, 1, 0)
const int mod = 998244353;
int64_t n, k, Pow10[20];
array<int64_t, 2> dp[20][1025][2][2]; // dp[0] -> cnt,
↪ dp[1] -> sum;
string s;
array<int64_t, 2> f(int i, int mask, int tight, int
↪ started) {
  if (i == n) {
    return {started && (__builtin_popcount(mask) <=
    ↪ k), 0};
  }
  auto& ret = dp[i][mask][tight][started];
  if (~ret[0]) return ret;
  ret = {0, 0};
  int limit = tight ? (s[i] - '0') : 9;
  for (int d = 0; d <= limit; d++) {
    int newTight = tight && (d == limit);
    int newStarted = started || (d > 0);
    int newMask = mask;
    if (newStarted) newMask |= (1 << d);
    auto [currCnt, currSum] = f(i + 1, newMask,
    ↪ newTight, newStarted);
    auto& [cnt, sum] = ret;
    cnt = (cnt + currCnt) % mod;
    sum = (sum + (currCnt * d % mod) * Pow10[n - i -
    ↪ 1] % mod) % mod;
    sum = (sum + currSum) % mod;
  }
  return ret;
}
int64_t Cnt(int64_t num) {
  if (num <= 0) return 0;
  memset(dp, -1, sizeof dp);
  s = to_string(num);
  n = s.size();
  Pow10[0] = 1;
  for (int i = 1; i < 20; i++) {
    Pow10[i] = (Pow10[i - 1] * 10) % mod;
  }
  return f(0, 0, 1, 0)[1];
}
--> find digit sum upto n, this is optimized version,
↪ use memset just once
string s;
int64_t dp[20][2][2];
int64_t cnt[20][2][2];
pair<int64_t, int64_t> f(int i, int tight, int
↪ started) {
  if (i < 0) return {0, started ? 1 : 0}; // {sum,
  ↪ count of valid numbers}
  if (~dp[i][tight][started] && !tight) return
  ↪ {dp[i][tight][started], cnt[i][tight][started]};
  int64_t totalSum = 0, totalCount = 0;
```

```
  int limit = tight ? (s[i] - '0') : 9;
  for (int d = 0; d <= limit; d++) {
    int newTight = tight && (d == limit);
    int newStarted = started || (d > 0);
    auto [nextSum, nextCount] = f(i - 1, newTight,
    ↪  newStarted);
    if (newStarted) {
      totalSum += d * nextCount + nextSum;
      totalCount += nextCount;
    } else {
      totalSum += nextSum;
      totalCount += nextCount;
    }
  }
  dp[i][tight][started] = totalSum;
  cnt[i][tight][started] = totalCount;
  return {totalSum, totalCount};
}
int64_t sumOfAllDigits(int64_t num) {
  if (num < 0) return 0;
  s = to_string(num);
  reverse(s.begin(), s.end());
  return f(s.size() - 1, 1, 0).first;
}
memset(dp, -1, sizeof dp);
memset(cnt, -1, sizeof cnt);

// Optimized versions -->
## with only 1 memset, reverse the number then
↪  calculate from (n - 1) ->
## so when some state is already calculated and tight
↪  == 0, then return 'ret', or else again calculate
↪  the state
#13 count of 3 = 6 = 9, and count of 3 > 0
const int mod = 1e9 + 7, N = 17;
string s;
int dp[51][N][N][N][2]; // f(s.size() - 1, 0, 0, 0, 1)
int f(int i, int three, int six, int nine, int tight)
↪  {
  if (three > 16 || six > 16 || nine > 16) return 0;
  if (i < 0) {
    return (three > 0) && (three == six) && (six ==
    ↪  nine);
  }
  auto& ret = dp[i][three][six][nine][tight];
  if (ret != -1 && !tight) return ret;
  ret = 0;
  int limit = tight ? (s[i] - '0') : 9;
  for (int d = 0; d <= limit; d++) {
    int newTight = tight && (d == limit);
    ret = (ret + f(i - 1, three + (d == 3), six + (d
    ↪  == 6), nine + (d == 9), newTight)) % mod;
  }
  return ret;
}
// optimized, without tight state
// when the state is not tight then return 'ret' also
↪  memoize the dp;
int dp[51][N][N][N];
int f(int i, int three, int six, int nine, int tight)
↪  {
  if (three > 16 || six > 16 || nine > 16) return 0;
  if (i < 0) {
    return (three > 0) && (three == six) && (six ==
    ↪  nine);
  }
  auto ret = dp[i][three][six][nine];
  if (ret != -1 && !tight) return ret;
  ret = 0;
  int limit = tight ? (s[i] - '0') : 9;
  for (int d = 0; d <= limit; d++) {
    int newTight = tight && (d == limit);
    ret = (ret + f(i - 1, three + (d == 3), six + (d
    ↪  == 6), nine + (d == 9), newTight)) % mod;
  }
  if (!tight) dp[i][three][six][nine] = ret;
  return ret;
}
int Cnt(string a) {
  reverse(a.begin(), a.end());
  s = a;
  return f(s.size() - 1, 0, 0, 0, 1);
}
int check(string& s) {
  // problem condition
}
void solve() {
  string a, b;
  cin >> a >> b;
  cout << Cnt(b) - Cnt(a) + check(a); // check if only
  ↪  string a can satisfy the condition
}


// calculate between l and r in one function
int dp[51][2][2][18][18][18];
string l, r;
int f(int i, int tightLower, int tightUpper, int c3,
↪  int c6, int c9) {
  if (c3 >= 17 || c6 >= 17 || c9 >= 17) return 0;
  if (i < 0) {
    return c3 && c3 == c6 && c6 == c9;
  }
  int& ret =
  ↪  dp[i][tightLower][tightUpper][c3][c6][c9];
  if (~ret && !tightLower && !tightUpper) return ret;
  int lo = tightLower ? l[i] - '0' : 0;
  int hi = tightUpper ? r[i] - '0' : 9;
  ret = 0;
  for (int d = lo; d <= hi; d++) {
    int newTightLower = tightLower && (d == lo);
    int newTightUpper = tightUpper && (d == hi);
    ret = (ret + f(i - 1, newTightLower,
    ↪  newTightUpper, c3 + (d == 3), c6 + (d == 6),
    ↪  c9 + (d == 9))) % mod;
  }
  return ret;
}
// optimized, without (tightLower, tightUpper) state
int f(int i, int tightLower, int tightUpper, int c3,
↪  int c6, int c9) {
  if (c3 >= 17 || c6 >= 17 || c9 >= 17) return 0;
  if (i < 0) {
    return c3 && c3 == c6 && c6 == c9;
  }
  int ret = dp[i][c3][c6][c9];
  if (~ret && !tightLower && !tightUpper) return ret;
  int lo = tightLower ? l[i] - '0' : 0;
  int hi = tightUpper ? r[i] - '0' : 9;
  ret = 0;
  for (int d = lo; d <= hi; d++) {
    int newTightLower = tightLower && (d == lo);
    int newTightUpper = tightUpper && (d == hi);
    ret = (ret + f(i - 1, newTightLower,
    ↪  newTightUpper, c3 + (d == 3), c6 + (d == 6),
    ↪  c9 + (d == 9))) % mod;
  }
  if (!tightLower && !tightUpper) dp[i][c3][c6][c9] =
  ↪  ret;
  return ret;
}
void solve() {
  cin >> l >> r;
  int n = r.size();
  while (l.size() < n) l = "0" + l;
  reverse(l.begin(), l.end());
  reverse(r.begin(), r.end());
  cout << f(n - 1, 1, 1, 0, 0, 0) << '\n';
}
```

15 Distinct Subsequence

```
const int mod = 1e9 + 7;
int distinctSubseq(string &a) {
  vector<int> last(26); // for array, use map
  int res = 1;
  for(auto& ai : a) {
    int curr = (2LL * res - last[ai - 'a']) % mod;
    if (curr < 0) curr += mod;
    last[ai - 'a'] = res;
    res = curr;
  }
  return (res - 1 + mod) % mod; // without empty set
}
```

16 DSU Kruskal's Algorithm MST

```
struct DSU {
  vector<int> parent, sz;
  DSU(int n) {
    parent.resize(n + 1), sz.resize(n + 1, 1);
    iota(parent.begin(), parent.end(), 0);
  }
  int Find(int u) {
    if (u == parent[u]) return u;
    return parent[u] = Find(parent[u]);
  }
```

```
  bool Is_connected(int u, int v) {
    return Find(u) == Find(v);
  }
  void Merge(int u, int v) {
    u = Find(u), v = Find(v);
    if (u != v) {
      if (sz[u] < sz[v]) swap(u, v);
      parent[v] = u;
      sz[u] += sz[v];
    }
  }
};
```

17 Exclution DP

```
vector<int> f(n), g(n);
for (int i = 1; i < N; i++) {
  for (int j = i; j < N; j += i) {
    f[i] += cnt[j];
  }
  f[i] = nC2(f[i]);
}
for (int i = N - 1; i > 0; i--) {
  g[i] = f[i];
  for (int j = i * 2; j < N; j += i) {
    g[i] -= g[j];
  }
}
// g[i] = how many pairs have gcd i
```

18 Fenwick Tree 2D

```
struct FenWick2D {
  int n, m;
  vector<vector<long long>> ft;
  FenWick2D() {}
  FenWick2D(int _n, int _m) {
    n = _n, m = _m;
    ft.assign(n + 1, vector<long long> (m + 1));
  }
  void Add(int x, int y, long long val) {
    for(int i = x; i <= n; i += i & -i) {
      for(int j = y; j <= m; j += j & -j) {
        ft[i][j] += val;
      }
    }
  }
  long long Csum(int x, int y) {
    long long res = 0;
    for(int i = x; i > 0; i -= i & -i) {
      for(int j = y; j > 0; j -= j & -j) {
        res += ft[i][j];
      }
    }
    return res;
  }
  long long Rsum(int x1, int y1, int x2, int y2) {
    return Csum(x2, y2) - Csum(x1 - 1, y2) - Csum(x2,
    ↪  y1 - 1) + Csum(x1 - 1, y1 - 1);
  }
};
```

19 Fenwick Tree BIT

```
struct FenWickTree {
  int n;
  vector<long long> ft;
  FenWickTree() {}
  FenWickTree(int n) { Initial(n); }
  FenWickTree(vector<int>& a) {
    Initial((int)a.size());
    Build(a);
  }
  void Initial(int _n) {
    n = _n;
    ft.assign(n + 1, 0);
  }
  void Build(vector<int>& a) {
    for (int i = 0; i < (int)a.size(); i++) {
      Add(i, i, a[i]); #change
    }
  }
  void Add(int idx, long long val) {
    for (int i = idx; i <= n; i += i & -i) {
      ft[i] += val;
    }
  }
  void Add(int l, int r, long long val) {
    Add(l, val);
    Add(r + 1, -val);
  }
  long long Csum(int idx) {
    long long res = 0;
    for (int i = idx; i > 0; i -= i & -i) {
      res += ft[i];
    }
    return res;
  }
  long long Rsum(int l, int r) {
    return Csum(r) - Csum(l - 1);
  }
};
```

20 FFT

```
struct cplx {
  long double a, b;
  cplx(long double a = 0, long double b = 0) : a(a),
  ↪  b(b) {}
  const cplx operator + (const cplx &c) const { return
  ↪  cplx(a + c.a, b + c.b); }
  const cplx operator - (const cplx &c) const { return
  ↪  cplx(a - c.a, b - c.b); }
  const cplx operator * (const cplx &c) const { return
  ↪  cplx(a * c.a - b * c.b, a * c.b + b * c.a); }
  const cplx operator / (const long double &d) const {
  ↪  return cplx(a / d, b / d); }
};
const long double PI = acos(-1);
vector<int> rev;
void Preprocess(int sz) {
  if ((int)rev.size() == sz) return;
  rev.resize(sz);
  rev[0] = 0;
  int lg_n = __builtin_ctz(sz);
  for (int i = 1; i < sz; ++i) {
    rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (lg_n -
    ↪  1));
  }
}
void fft(vector<cplx> &a, bool inv = 0) {
  int n = a.size();
  for (int i = 1; i < n - 1; ++i) {
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  }
  for (int len = 2; len <= n; len <<= 1) {
    long double t = 2 * PI / len * (inv ? -1 : 1);
    cplx wlen = {cosl(t), sinl(t)};
    for (int st = 0; st < n; st += len) {
      cplx w(1);
      for (int i = 0; i < len / 2; ++i) {
        cplx ev = a[st + i], od = a[st + i + len / 2]
        ↪  * w;
        a[st + i] = ev + od;
        a[st + i + len / 2] = ev - od;
        w = w * wlen;
      }
    }
  }
  if (inv) {
    for (cplx& z : a) z = z / n;
  }
}
vector<long long> multiply(vector<int> &a, vector<int>
↪  &b) {
  int n = a.size(), m = b.size(), sz = 1;
  while(sz < n + m - 1) sz <<= 1;
  vector<cplx> x(sz), y(sz), z(sz);
  for(int i = 0 ; i < sz; i++) {
    x[i] = i < n ? cplx(a[i], 0) : cplx(0, 0);
    y[i] = i < m ? cplx(b[i], 0) : cplx(0, 0);
  }
  Preprocess(sz);
  fft(x), fft(y);
  for(int i = 0; i < sz; i++) z[i] = x[i] * y[i];
  fft(z, 1);
  vector<long long> c(n + m - 1);
  for(int i = 0; i < n + m - 1; i++) c[i] =
  ↪  round(z[i].a);
  return c;
}
auto c = mutiply(a, b);
```

21 Floyd Warshall

```
void FloydWarshall(vector<vector<long long>>& adj) {
  int n = (int)adj.size();
```

```
  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        adj[i][j] = min(adj[i][j], adj[i][k] +
        ↪  adj[k][j]);
        adj[i][i] = 0;
      }
    }
  }
}
```

22 Geometry

```
const double PI = acos((double)-1.0);
using ld = long double;
using ll = long long;
struct point { ld x, y; };
void manhattanToChebychev(int& x, int& y) {
  int X = x + y, Y = x - y;
  x = X, y = Y;
}
void chebyshevToManhattan(int& x, int& y) {
  int X = (x + y) / 2, Y = (x - y) / 2;
  x = X, y = Y;
}
ll manhattanDistance(point& p1, point& p2) {
  return abs(p1.x - p2.x) + abs(p1.y - p2.y);
}
ll chebyshevDistance(point& p1, point& p2) {
  return max(abs(p1.x - p2.x), abs(p1.y - p2.y));
}
// determines the relative position (cross product) of
↪  a point (p3)
// with respect to the line passing through (p1) and
↪  (p2)
// if (d > 0): point (p3) is to the left of the line
// if (d < 0): point (p3) is to the right of the line
// if (d == 0): point (p3) lies exactly on the line.
ll determinant(point p1, point p2, point p3) {
  ll d = (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y)
  ↪  * (p3.x - p1.x);
  return d;
}
bool arePointsOnSameSide(point p1, point p2, point p3,
↪  point p4) {
  ll d1 = determinant(p1, p2, p3);
  ll d2 = determinant(p1, p2, p4);
  return (d1 * d2) > 0;
}
bool doLinesIntersect(point p1, point p2, point p3,
↪  point p4) {
  ll d1 = determinant(p3, p4, p1);
  ll d2 = determinant(p3, p4, p2);
  ll d3 = determinant(p1, p2, p3);
  ll d4 = determinant(p1, p2, p4);
  return (d1 * d2 < 0) && (d3 * d4 < 0);
}
// distance of two points
ld distance(point& p1, point& p2) {
  return sqrt((p2.x - p1.x) * (p2.x - p1.x) + (p2.y -
  ↪  p1.y) * (p2.y - p1.y));
}
// a point inside cricle
// x1, y1 = point of circle, r = radius of circle
// x2, y2 = target point
bool isInside(int x1, int y1, int x2, int y2, int r) {
  return ((x2 - x1) * (x2 - x1)) + ((y2 - y1) * (y2 -
  ↪  y1)) <= r * r;
}
ld toDegrees(ld radians) {
  return radians * (180.0 / PI);
}
// circle with radius r
struct Circle {
  ld r;
  Circle(ld _r) { r = _r; }
  ld Diameter() { return 2 * r; }
  ld Circumference() { return 2 * PI * r; }
  ld Area() { return PI * r * r; }
  // if theta in radians, make it to degrees
  // all the calculations are in degree
  ld Sector_Area(ld theta) { return (theta / 360.0) *
  ↪  Area(); }
  ld Arc_Length(ld theta) { return (theta / 360.0) *
  ↪  Circumference(); }
  ld Chord_Length(ld theta) { return 2 * r *
  ↪  sin((theta * PI) / 360.0); }
  ld Segment_Area(ld theta) { return
  ↪  Sector_Area(theta) - (0.5 * r * r * sin((theta *
  ↪  PI) / 180.0)); }
  ld Circumscribed_Square_Area() { return 4 * r * r; }
  ld Inscribed_Square_Area() { return 2 * r * r; }
  ld Sphere_Surface_Area() { return 4 * PI * r * r; }
  ld Sphere_Volume() { return (4.0 / 3.0) * PI * r * r
  ↪  * r; }
  ld Cylinder_Surface_Area(ld h) { return 2 * PI * r *
  ↪  (r + h); }
  ld Cylinder_Volume(ld h) { return PI * r * r * h; }
  ld Cone_Surface_Area(ld l) { return PI * r * (r +
  ↪  l); }
  ld Cone_Volume(ld h) { return (1.0 / 3.0) * PI * r *
  ↪  r * h; }
  ld Annulus_Area(ld R) { return PI * (r * r - R * R);
  ↪  }
};
// triangle with three sides: a, b, c
struct Triangle {
  ld a, b, c, s;
  Triangle(point x, point y, point z) {
    a = distance(x, y);
    b = distance(y, z);
    c = distance(z, x);
    s = (a + b + c) / 2;
  }
  Triangle(ld _a, ld _b, ld _c) {
    a = _a, b = _b, c = _c;
    s = (a + b + c) / 2;
  }
  ld Area() { return sqrt(s * (s - a) * (s - b) * (s -
  ↪  c)); }
  ld Perimeter() { return a + b + c; }
  ld Angle_A_Radians() { return acos((b * b + c * c -
  ↪  a * a) / (2 * b * c)); }
  ld Angle_B_Radians() { return acos((a * a + c * c -
  ↪  b * b) / (2 * a * c)); }
  ld Angle_C_Radians() { return acos((a * a + b * b -
  ↪  c * c) / (2 * a * b)); }
  ld Angle_A_Degrees() { return
  ↪  toDegrees(Angle_A_Radians()); }
  ld Angle_B_Degrees() { return
  ↪  toDegrees(Angle_B_Radians()); }
  ld Angle_C_Degrees() { return
  ↪  toDegrees(Angle_C_Radians()); }
  ld Inradius() { return sqrt((s - a) * (s - b) * (s -
  ↪  c) / s); }
  ld Circumradius() { return (a * b * c) / (4 *
  ↪  Area()); }
  ld Exradius_A() { return sqrt((s - a) * (s - b) * (s
  ↪  - c) / (s - a)); }
  ld Exradius_B() { return sqrt((s - a) * (s - b) * (s
  ↪  - c) / (s - b)); }
  ld Exradius_C() { return sqrt((s - a) * (s - b) * (s
  ↪  - c) / (s - c)); }
  ld Altitude_A() { return 2 * Area() / a; }
  ld Altitude_B() { return 2 * Area() / b; }
  ld Altitude_C() { return 2 * Area() / c; }
  ld Median_A() { return sqrt((2 * b * b + 2 * c * c -
  ↪  a * a) / 4); }
  ld Median_B() { return sqrt((2 * a * a + 2 * c * c -
  ↪  b * b) / 4); }
  ld Median_C() { return sqrt((2 * a * a + 2 * b * b -
  ↪  c * c) / 4); }
};
// n sided polygon (n-gon) with side length a
struct Polygon {
  int n;
  ld a;
  Polygon(int _n, ld _a) { n = _n, a = _a; }
  ld Perimeter() { return n * a; }
  ld Exterior_Angle_Degrees() { return 360.0 / n; }
  ld Exterior_Angle_Radians() { return (2 * PI) / n; }
  ld Interior_Angle_Degrees() { return (n - 2) * 180.0
  ↪  / n; }
  ld Interior_Angle_Radians() { return PI - (2 * PI /
  ↪  n); }
  ld Circumradius() { return a / (2 * sin(PI / n)); }
  ld Inradius() { return a / (2 * tan(PI / n)); }
  ld Area() { return (n * a * a) / (4 * tan(PI / n));
  ↪  }
  ld Central_Angle_Degrees() { return 360.0 / n; }
  ld Central_Angle_Radians() { return (2 * PI) / n; }
  ll Diagonals_count() { return n * (n - 3) / 2; }
  ld Diagonal_length() { return a / sin(PI / n); }
  ld Height() {
```

```
    if (n & 1) return 2 * Inradius();
    else return Inradius() + Circumradius();
  }
  ld Width() {
    if (n & 1) {
      return 2 * Circumradius() * sin((n - 1) * PI /
      ↪  (2 * n));
    } else {
      if ((n / 2) & 1) return 2 * Circumradius();
      else return 2 * Inradius();
    }
  }
};
```

23 Hash Map

```
#include <ext/pb_ds/assoc_container.hpp>
struct splitmix64_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM =
    ↪  std::chrono::steady_clock::now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};
template <typename K, typename V, typename Hash =
↪  splitmix64_hash>
using hash_map = __gnu_pbds::gp_hash_table<K, V,
↪  Hash>;
template <typename K, typename Hash = splitmix64_hash>
using hash_set = hash_map<K, __gnu_pbds::null_type,
↪  Hash>;
```

24 Hashing with Update

```
const int MAX = int(1e6) + 9;
vector<long long> MOD = {1909999999, 1999990999};
vector<array<long long, 2>> pw(MAX), ipw(MAX);
array<long long, 2> bs = {137, 277};
int BinExp(long long a, long long b, int mod) {
  a %= mod;
  int res = 1;
  while (b) {
    if (b & 1) res = (res * a) % mod;
    a = (a * a) % mod;
    b >>= 1;
  }
  return res;
}
void Preprocess() {
  pw[0][0] = pw[0][1] = 1;
  for (int i = 1; i < MAX; i++) {
    pw[i][0] = (pw[i - 1][0] * bs[0]) % MOD[0];
    pw[i][1] = (pw[i - 1][1] * bs[1]) % MOD[1];
  }
  ipw[0][0] = ipw[0][1] = 1;
  long long ip1 = BinExp(bs[0], MOD[0] - 2, MOD[0]);
  long long ip2 = BinExp(bs[1], MOD[1] - 2, MOD[1]);
  for (int i = 1; i < MAX; i++) {
    ipw[i][0] = (ipw[i - 1][0] * ip1) % MOD[0];
    ipw[i][1] = (ipw[i - 1][1] * ip2) % MOD[1];
  }
}
struct Hashing {
  int n;
  vector<array<long long, 2>> hs;
  Hashing(string& s) {
    if (pw[2][0] == 0) Preprocess();
    n = s.size();
    hs.resize(n + 1);
    for (int i = 0; i < n; i++) {
      hs[i + 1][0] = (hs[i][0] + (pw[i][0] * s[i]) %
      ↪  MOD[0]) % MOD[0];
      hs[i + 1][1] = (hs[i][1] + (pw[i][1] * s[i]) %
      ↪  MOD[1]) % MOD[1];
    }
  }
  array<long long, 2> get_hash(int l, int r) { // 0
  ↪  based query
    l++, r++; // 1 based hashing
    long long res1 = ((hs[r][0] - hs[l - 1][0]) +
    ↪  MOD[0]) % MOD[0];
    long long res2 = ((hs[r][1] - hs[l - 1][1]) +
    ↪  MOD[1]) % MOD[1];
    res1 = (res1 * ipw[l - 1][0]) % MOD[0];
    res2 = (res2 * ipw[l - 1][1]) % MOD[1];
    //return res1 << 31 | res2;
    return {res1, res2};
  }
};
auto Hash_Merge(array<long long, 2> left, array<long
↪  long, 2> right, int left_sz) {
  for (int i = 0; i < 2; i++) {
    (right[i] *= pw[left_sz][i]) %= MOD[i];
    (left[i] += right[i]) %= MOD[i];
  }
  return left;
};
// # with update, find palindrome O(nlogn)
const int N = 1e6 + 9;
int power(long long n, long long k, const int mod) {
  int ans = 1 % mod;
  n %= mod;
  if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}
using T = array<int, 2>;
const T MOD = {127657753, 987654319};
const T p = {137, 277};
T operator + (T a, int x) {return {(a[0] + x) %
↪  MOD[0], (a[1] + x) % MOD[1]};}
T operator - (T a, int x) {return {(a[0] - x + MOD[0])
↪  % MOD[0], (a[1] - x + MOD[1]) % MOD[1]};}
T operator * (T a, int x) {return {(int)((long long)
↪  a[0] * x % MOD[0]), (int)((long long) a[1] * x %
↪  MOD[1])};}
T operator + (T a, T x) {return {(a[0] + x[0]) %
↪  MOD[0], (a[1] + x[1]) % MOD[1]};}
T operator - (T a, T x) {return {(a[0] - x[0] +
↪  MOD[0]) % MOD[0], (a[1] - x[1] + MOD[1]) %
↪  MOD[1]};}
T operator * (T a, T x) {return {(int)((long long)
↪  a[0] * x[0] % MOD[0]), (int)((long long) a[1] *
↪  x[1] % MOD[1])};}
ostream& operator << (ostream& os, T hash) {return os
↪  << "(" << hash[0] << ", " << hash[1] << ")";}
T pw[N], ipw[N];
void prec() {
  pw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    pw[i] = pw[i - 1] * p;
  }
  ipw[0] =  {1, 1};
  T ip = {power(p[0], MOD[0] - 2, MOD[0]), power(p[1],
  ↪  MOD[1] - 2, MOD[1])};
  for (int i = 1; i < N; i++) {
    ipw[i] = ipw[i - 1] * ip;
  }
}
struct Hashing {
  int n;
  string s; // 1 - indexed
  vector<array<T, 2>> t; // (normal, rev) hash
  array<T, 2> merge(array<T, 2> l, array<T, 2> r) {
    l[0] = l[0] + r[0];
    l[1] = l[1] + r[1];
    return l;
  }
  void build(int node, int b, int e) {
    if (b == e) {
      t[node][0] = pw[b] * s[b];
      t[node][1] = pw[n - b + 1] * s[b];
      return;
    }
    int mid = (b + e) >> 1, l = node << 1, r = l | 1;
    build(l, b, mid);
    build(r, mid + 1, e);
    t[node] = merge(t[l], t[r]);
  }
  void upd(int node, int b, int e, int i, char x) {
    if (b > i || e < i) return;
    if (b == e && b == i) {
      t[node][0] = pw[b] * x;
      t[node][1] = pw[n - b + 1] * x;
      return;
    }
    int mid = (b + e) >> 1, l = node << 1, r = l | 1;
```

```
    upd(l, b, mid, i, x);
    upd(r, mid + 1, e, i, x);
    t[node] = merge(t[l], t[r]);
  }
  array<T, 2> query(int node, int b, int e, int i, int
  ↪  j) {
    if (b > j || e < i) return {T({0, 0}), T({0, 0})};
    if (b >= i && e <= j) return t[node];
    int mid = (b + e) >> 1, l = node << 1, r = l | 1;
    return merge(query(l, b, mid, i, j), query(r, mid
    ↪  + 1, e, i, j));
  }
  Hashing() {}
  Hashing(string _s) {
    n = _s.size();
    s = "." + _s;
    t.resize(4 * n + 1);
    build(1, 1, n);
  }
  void upd(int i, char c) {
    upd(1, 1, n, i, c);
    s[i] = c;
  }
  T get_hash(int l, int r) { // 1 - indexed
    return query(1, 1, n, l, r)[0] * ipw[l - 1];
  }
  T rev_hash(int l, int r) { // 1 - indexed
    return query(1, 1, n, l, r)[1] * ipw[n - r];
  }
  T get_hash() return get_hash(1, n);
  bool is_palindrome(int l, int r) return get_hash(l,
  ↪  r) == rev_hash(l, r);
};
```

25 HLD

```
struct HeavyLightDecomposition {
  int t = 0;
  vector<int> tin, depth, subtree_size, parent, heavy,
  ↪  head, euler;
  SegmentTreeIterative<int> sg;
  HeavyLightDecomposition(int root,
  ↪  vector<vector<int>>& adj, vector<int>& values) {
    int n = (int)adj.size() + 1;
    tin.resize(n), depth.resize(n),
    ↪  subtree_size.resize(n), parent.resize(n),
    ↪  heavy.assign(n, -1), head.resize(n),
    ↪  euler.resize(n);
    Dfs(root, root, adj);
    Decompose(root, root, adj, values);
    sg = SegmentTreeIterative<int> (euler);
  }
  void Dfs(int u, int p, vector<vector<int>>& adj) {
    subtree_size[u] = 1;
    parent[u] = p;
    for (auto& v : adj[u]) {
      if (v != p) {
        depth[v] = depth[u] + 1;
        Dfs(v, u, adj);
        subtree_size[u] += subtree_size[v];
        if (heavy[u] == -1 || subtree_size[heavy[u]] <
        ↪  subtree_size[v]) {
          heavy[u] = v;
        }
      }
    }
  }
  void Decompose(int u, int h, vector<vector<int>>&
  ↪  adj, vector<int>& values) {
    tin[u] = ++t;
    euler[t] = values[u];
    head[u] = h;
    if (heavy[u] != -1) {
      Decompose(heavy[u], h, adj, values);
    }
    for (auto& v : adj[u]) {
      if (v != parent[u] && v != heavy[u]) {
        Decompose(v, v, adj, values);
      }
    }
  }
  void Update(int u, int val) {
    sg.Update(tin[u], val);
  }
  int neutral = 0; #change
  int Merge(int a, int b) {
    return max(a, b); #change
  }
  int PathQuery(int a, int b) {
    int res = neutral;
    for (; head[a] != head[b]; b = parent[head[b]]) {
      if (depth[head[a]] > depth[head[b]]) {
        swap(a, b);
      }
      res = Merge(res, sg.Query(tin[head[b]],
      ↪  tin[b]));
    }
    if (depth[a] > depth[b]) {
      swap(a, b);
    }
    res = Merge(res, sg.Query(tin[a], tin[b]));
    return res;
  }
};
HeavyLightDecomposition hld(1, adj, a);
// add segment tree
```

26 Hopcroft Karp

```
// O(√V * E)
// works for only directed graph, or on two graphs
// 1 to n is left graph, n + 1 to n + m is right graph
// make directed edges for two graph
// match pairs -> if (i < match[i]) (i <-> match[i])
const int N = 2e5 + 9, INF = 1.1e9;
vector<int> adj[N];
int n, m, match[N], dist[N];
bool bfs() {
  queue<int> q;
  for (int i = 1; i <= n; ++i) {
    if (!match[i]) dist[i] = 0, q.emplace(i);
    else dist[i] = INF;
  }
  dist[0] = INF;
  while (!q.empty()) {
    int u = q.front(); q.pop();
    if (!u) continue;
    for (int v : adj[u]) {
      if (dist[match[v]] == INF) {
        dist[match[v]] = dist[u] + 1,
        q.emplace(match[v]);
      }
    }
  }
  return dist[0] != INF;
}
bool dfs (int u) {
  if (!u) return 1;
  for (int v : adj[u]) {
    if (dist[match[v]] == dist[u] + 1 and
    ↪  dfs(match[v])) {
      match[u] = v, match[v] = u;
      return 1;
    }
  }
  dist[u] = INF;
  return 0;
}
int max_matching() {
  int ret = 0;
  while (bfs()) {
    for (int i = 1; i <= n; ++i) {
      ret += !match[i] and dfs(i);
    }
  }
  return ret;
}
```

27 Hungarian Min Assignment

```
// returns the max sum and permuation
// for minimum, convert each value to negative.
template<typename T>
pair<T, vector<int> > hungarian(const vector<vector<T>
↪  > &a) {
  if (a.empty()) return {0, {}};
  int n = a.size() + 1, m = a[0].size() + 1;
  vector<int> p(m), ans(n - 1);
  vector<T> u(n), v(m);
  for (int i = 1; i < n; i++) {
    p[0] = i;
    int now = 0;
    vector<int> pre(m, -1), vis(m + 1);
    vector<T> dis(m, numeric_limits<T>::max());
    do {
      vis[now] = true;
      int t = p[now], nxt;
```

```
      T d = numeric_limits<T>::max();
      for (int j = 1; j < m; j++)
        if (!vis[j]) {
          T cur = -a[t - 1][j - 1] - u[t] - v[j];
          if (cur < dis[j]) dis[j] = cur, pre[j] =
          ↪  now;
          if (dis[j] < d) d = dis[j], nxt = j;
        }
      for (int j = 0; j < m; j++) {
        if (vis[j]) u[p[j]] += d, v[j] -= d;
        else dis[j] -= d;
      }
      now = nxt;
    } while (p[now]);
    while (now) {
      int t = pre[now];
      p[now] = p[t], now = t;
    }
  }
  for (int i = 1; i < m; i++)
    if (p[i]) ans[p[i] - 1] = i - 1;
  return {v[0], ans};
}
```

28 KMP

```
// length of longest proper prefix of P[0...i] that is
↪  also a suffix of P[0...i]
// Pattern: a b a b a c a
// Index:   0 1 2 3 4 5 6
// LPS[]:   0 0 1 2 3 0 1
vector<int> get_pi(string& s) {
  int n = s.size();
  vector<int> pi(n);
  for (int i = 1, j = 0; i < n; i++) {
    if (s[i] == s[j]) pi[i] = ++j;
    else if (j == 0) pi[i] = 0;
    else j = pi[j - 1], i--;
  }
  return pi;
}
// count the number of p occurs in s
string s, p;
s = p + "#" + s;
vector<int> pi = get_pi(s);
int ans = 0;
for (auto& i : pi) ans += (i == p.size());
```

29 Longest Path in DAG

```
vector<int> adj[N];
int dp[N], vis[N], n;
int dfs (int u) {
  vis[u] = dp[u] = 1;
  for (int v : adj[u]) {
    if (!vis[v]) dfs(v);
    dp[u] = max(dp[u], dp[v] + 1);
  } return dp[u];
}
int longestpath() {
  int ans = 0;
  for (int i = 1; i <= n; i++)
    if (!vis[i]) ans = max(ans, dfs(i));
  return ans;
}
```

30 Manacher

```
struct Manacher { // 0 based
  int n;
  vector<int> p;
  Manacher(string& s) {
    n = s.size();
    p.resize(2 * n);
    build(s);
  }
  void build(string& s) {
    for (int i = 0, j = 0, k; i < n * 2; i += k, j =
    ↪  max(j - k, 0)) {
      while (i >= j && i + j + 1 < n * 2 && s[(i - j)
      ↪  / 2] == s[(i + j + 1) / 2]) ++j;
      p[i] = j;
      for (k = 1; i >= k && j >= k && p[i - k] != j -
      ↪  k; ++k) {
        p[i + k] = min(p[i - k], j - k);
      }
    }
  }
  bool is_palindrome(int l, int r) {
    int len = (r - l + 1), idx = -1;
    if (len & 1) idx = (l + len / 2) * 2;
    else idx = (l + len / 2 - 1) * 2 + 1;
    return p[idx] >= len;
  }
  int odd_length_of_center_i(int i) {return p[i * 2];
  ↪  }
  int even_length_of_center_i(int i) { return p[i * 2
  ↪  + 1]; }
};
```

31 Merge Sort Tree

```
// add ordered multiset
struct MergeSortTree {
  int size = 1;
  vector<ordered_multiset<int>> st; #change
  MergeSortTree() {}
  MergeSortTree(int n) { Initial(n); }
  MergeSortTree(vector<int>& a) {
    Initial((int)a.size() - 1);
    Build(1, 1, size, a);
  }
  void Initial(int _n) {
    size = _n;
    int tree_size = 1;
    while (tree_size < _n) tree_size *= 2;
    st.resize(tree_size * 2);
  }
  void Build(int u, int s, int e, vector<int>& a) {
    if (s == e) { #change
      st[u].insert(a[s]);
      return;
    }
    int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
    Build(v, s, m, a);
    Build(w, m + 1, e, a);
    for (int i = s; i <= e; i++) st[u].insert(a[i]);
  }
  void Update(int u, int s, int e, int k, int prev,
  ↪  int curr) {
    if (s == e) { #change

      ↪  st[u].erase(st[u].find_by_order(st[u].order_of_key
      st[u].insert(curr);
      return;
    }
    int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
    if (k <= m) Update(v, s, m, k, prev, curr);
    else Update(w, m + 1, e, k, prev, curr);
    #change

    ↪  st[u].erase(st[u].find_by_order(st[u].order_of_key(pr
    st[u].insert(curr);
  }
  void Update(int k, int prev, int curr) {
    Update(1, 1, size, k, prev, curr);
  }
  int Query(int u, int s, int e, int l, int r, int
  ↪  val) {
    if (e < l || r < s) { #change
      return 0;
    }
    if (l <= s && e <= r) { #change
      return st[u].order_of_key(val);
    }
    int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
    int lsum = Query(v, s, m, l, r, val);
    int rsum = Query(w, m + 1, e, l, r, val);
    return lsum + rsum; #change
  }
  int Query(int l, int r, int val) {
    return Query(1, 1, size, l, r, val);
  }
};
```

32 MEX with TRIE

```
const int N = 2e5 * 22 + 9;
int nxt[N][2], cnt[N], intCnt[N], node = 2;
void insert(int x) {
  int u = 1;
  for(int i = 20; i >= 0; i--) {
    int bit = (x >> i) & 1;
    if(!nxt[u][bit]) nxt[u][bit] = node++;
    u = nxt[u][bit];
    cnt[u]++;
  }
```

```
  intCnt[u]++;
}
int find(int x) {
  int u = 1;
  for (int i = 20; i >= 0; i--) {
    int bit = (x >> i) & 1;
    if (!nxt[u][bit]) return 0;
    u = nxt[u][bit];
  }
  return intCnt[u];
}
void erase(int x) {
  if (find(x) == 0) return;
  int u = 1;
  for (int i = 20; i >= 0; i--) {
    int bit = (x >> i) & 1;
    int v = nxt[u][bit];
    cnt[v]--;
    u = v;
  }
  intCnt[u]--;
}
int mex() {
  int u = 1, ret = 0;
  for(int i = 20; i >= 0; i--) {
    if(u == 0 || !nxt[u][0]) return ret;
    if(cnt[nxt[u][0]] >= (1 << i)) {
      u = nxt[u][1];
      ret |= (1 << i);
    } else {
      u = nxt[u][0];
    }
  }
  return ret;
}
```

33 Minimum Expression

```
// lexicographically smallest rotation
int minimumExpression(string s) {
  s = s + s;
  int i = 0, j = 1, k = 0, len = s.size();
  while(i + k < len && j + k < len) {
    if(s[i + k] == s[j + k]) k++;
    else if(s[i + k] < s[j + k]) { j = max(j + k + 1,
    ↪  i + 1); k = 0; }
    else { i = max(i + k + 1, j + 1); k = 0; }
  } return min(i, j);
}
```

34 Number Theory

```
const int N = 1e6 + 9;
vector<int> lpf(N), gpf(N);
vector<array<int, 2>> factors[N];
// sieve for finding lowest prime and highest prime
↪  upto N
for (int i = 2; i < N; i++) {
  if (lpf[i] == 0) {
    for (int j = i; j < N; j += i) {
      gpf[j] = i;
      if (!lpf[j]) lpf[j] = i;
    }
  }
}
// find all factors upto N
for (int i = 2; i < N; i++) {
  int num = i;
  while (num > 1) {
    int p = lpf[num], cnt = 0;
    while (num > 1 && num % p == 0) {
      cnt++;
      num /= p;
    }
    factors[i].push_back({p, cnt});
  }
}
int num = 10;
int total_divisors = 1;
long long sum_of_divisors = 1;
for (auto& [p, c] : factors[num]) {
  total_divisors *= (c + 1);
  sum_of_divisors *= (pow(p, c + 1) - 1) / (p - 1);
}
// (a ^ b) % p   (Binary Exponentiation)
int BinExp(long long a, long long b, int mod) {
  a %= mod;
  int res = 1;
  while (b) {
    if (b & 1) res = (res * a) % mod;
    a = (a * a) % mod;
    b >>= 1;
  }
  return res;
}
// (a * b) % p   (Binary Multiplication)
int BinMul(long long a, long long b, int mod) {
  a %= mod;
  int res = 0;
  while (b) {
    if (b & 1) res = (res + a) % mod;
    a = (a + a) % mod;
    b >>= 1;
  }
  return res;
}
// (a ^ -1) % mod (inverse of a number)
BinExp(a, mod - 2);
// (a / b) % mod
BinMul(a, BinExp(b, mod - 2, mod), mod);
// (a ^ (b ^ c)) % mod
BinExp(a, BinExp(b, c, mod - 1), mod);

// Permutations and Combinations
struct Combinatorics {
  vector<long long> fact, inv, ifact;
  Combinatorics(int n) {
    fact.assign(n + 1, 1), inv.assign(n + 1, 1),
    ↪  ifact.assign(n + 1, 1);
    inv[0] = 0;
    for (int i = 2; i <= n; i++) fact[i] = (fact[i -
    ↪  1] * i) % mod;
    for (int i = 2; i <= n; i++) inv[i] = mod - (mod /
    ↪  i) * inv[mod % i] % mod;
    for (int i = 2; i <= n; i++) ifact[i] = (ifact[i -
    ↪  1] * inv[i]) % mod;
  }
  int nPr(int n, int r) { // Permutations
    if (n < r) return 0;
    return (fact[n] * ifact[n - r]) % mod;
  }
  int nCr(int n, int r) { // Combinations
    return (nPr(n, r) * ifact[r]) % mod;
  }
  int nCRr(int n, int r) { // Combinations with
  ↪  repetition
    return (nPr(n + r - 1, r) * ifact[r]) % mod;
  }
} comb(N);

// phi of single integer
int n = 10;
long long num = n;
long long phi_of_n = n;
for (long long i = 2; i * i <= num; i++) {
  if (num % i == 0) {
    while (num % i == 0) num /= i;
    phi_of_n -= phi_of_n / i;
  }
}
if (num > 1) phi_of_n -= phi_of_n / num;

// phi upto N
vector<int> phi(N);
// initial 0 to N
iota(phi.begin(), phi.end(), 0);
for (int i = 2; i < N; i++) {
  if (phi[i] == i) {
    for (int j = i; j < N; j += i) {
      phi[j] -= phi[j] / i;
    }
  }
}

// gcd sum -> ∑ gcd(i, n) for 1 <= i <= n; (n <= 1e9)
// gcd(1, n) + gcd(2, n) + .. + gcd(n, n)
long long sum = 0;
for (int i = 1; 1LL * i * i <= n; i++) {
  if (n % i == 0) {
    sum += i * phi(n / i);
    if (n / i != i) sum += (n / i) * phi(n / (n / i));
  }
}
// all pair gcd sum from 1 to N (N <= 4e6)
// ∑ i to n - 1, ∑ j + 1 to n [gcd(i, j)]
for (int i = 1; i < N; i++) {
  for (int j = i; j < N; j += i) sum[j] += i * phi[j /
  ↪  i];
```

```
}
for (int i = 1; i < N; i++) sum[i] += sum[i - 1] - i;

// lcm sum -> ∑ lcm(i, n) for 1 <= i <= n; (n <= 1e6)
// lcm(1, n) + lcm(2, n) + .. + lcm(n, n)
phi[1] = 2; // phi[1] should be 2 for this algorithm
for (int i = 1; i < N; i++) {
  for (int j = i; j < N; j += i) lcm_sum[j] += 1LL * j
  ↪  * phi[i] * i / 2;
}
// all pair lcm sum from 1 to N (N <= 1e6)
// ∑ i to n - 1, ∑ j + 1 to n [lcm(i, j)]
phi[1] = 2; // phi[1] should be 2 for this algorithm
for (int i = 1; i < N; i++) {
  for (int j = i; j < N; j += i) lcm_sum[j] += (long
  ↪  long) j * (1LL * i * phi[i] / 2);
}
for (int i = 1; i < N; i++) lcm_sum[i] += lcm_sum[i -
↪  1] - i;
```

35 Ordered Set

```
#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>;
template <class T> using ordered_multiset = tree<T,
↪  null_type, less_equal<T>, rb_tree_tag,
↪  tree_order_statistics_node_update>;
template <class T, class R> using ordered_map =
↪  tree<T, R, less<T>, rb_tree_tag,
↪  tree_order_statistics_node_update>;
// Sorting Descending (or Equal)
template <class T> using ordered_set = tree<T,
↪  null_type, greater<T>, rb_tree_tag,
↪  tree_order_statistics_node_update>;
template <class T> using ordered_multiset = tree<T,
↪  null_type, greater_equal<T>, rb_tree_tag,
↪  tree_order_statistics_node_update>;
template <class T, class R> using ordered_map =
↪  tree<T, R, greater<T>, rb_tree_tag,
↪  tree_order_statistics_node_update>;
// Fucntions
st.order_of_key(k); // count of elements smaller than
↪  k. - O(log n)
st.find_by_order(k); // Returns the iterator for the
↪  kth element (0 based index). - O(log n)
st.size() - st.order_of_key(k + 1); // count of
↪  elements greater than k. - O(log n)
st.order_of_key(r + 1) - st.order_of_key(l); // count
↪  of elements between l to r
st.erase(st.find_by_order(st.order_of_key(k))); //
↪  erase in multiset
```

36 SCC

```
// clear everything first, 1 based
// reverse scc for topological order
const int N = 5e5 + 10;
vector<int> adj[N], trans[N], scc[N];
int col[N], vis[N], idx = 0, n, m;
stack<int> st;
void dfs(int u) {
  vis[u] = 1;
  for (int v : adj[u]) if (!vis[v]) dfs(v);
  st.push(u);
}
void dfs2(int u) {
  col[u] = idx;
  scc[idx].push_back(u);
  for (int v : trans[u]) if (!col[v]) dfs2(v);
}
void findSCC() {
  for (int i = 1; i <= n; i++)
    if(!vis[i]) dfs(i);
  for (int u = 1; u <= n; u++)
    for (int v : adj[u])
      trans[v].push_back(u);
  while (!st.empty()) {
    int u = st.top(); st.pop();
    if (col[u]) continue;
    idx++; dfs2(u);
  }
}
// take input and call findSCC();
```

37 Segment Tree Iterative Point Update Range Query

```
template <class T> struct SegmentTreeIterative {
  int n = 1;
  vector<T> st;
  SegmentTreeIterative() {}
  SegmentTreeIterative(int n) { Initial(n); }
  SegmentTreeIterative(vector<int>& a) {
    Initial((int)a.size() - 1);
    Build(a);
  }
  void Initial(int _n) {
    n = _n;
    int tree_size = 1;
    while (tree_size <= n) tree_size *= 2;
    st.resize(tree_size * 2);
  }
  T neutral = INT_MAX; #change
  T Merge(T& a, T& b) { #change
    return min(a, b);
  }
  void Build(vector<int>& a) {
    for (int i = 1; i <= n; ++i) {
      st[n + i] = a[i];
    }
    for (int u = n - 1; u > 0; --u) {
      st[u] = Merge(st[u << 1], st[u << 1 | 1]);
    }
  }
  void Update(int idx, T val) {
    st[idx += n] = val;
    for (idx /= 2; idx; idx /= 2) {
      st[idx] = Merge(st[idx << 1], st[idx << 1 | 1]);
    }
  }
  T Query(int l, int r) {
    T res = neutral;
    for (l += n, 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;
  }
};
```

38 Segment Tree Lazy

```
const long long INF = 1.1e17;
struct node { #change
  long long sum, lazy_add;
  node() {
    sum = 0;
    lazy_add = INF;
  }
};
struct SegmentTreeLazy {
  int size = 1;
  vector<node> st;
  SegmentTreeLazy() {}
  SegmentTreeLazy(int n) { Initial(n); }
  SegmentTreeLazy(vector<int>& a) {
    Initial((int)a.size() - 1);
    Build(1, 0, size, a);
  }
  void Initial(int _n) {
    size = _n;
    int tree_size = 1;
    while (tree_size <= size) tree_size *= 2;
    st.resize(tree_size * 2);
  }
  node Make_node(long long val) { #change
    node res;
    res.sum = val;
    res.lazy_add = INF;
    return res;
  }
  node Merge(node& l, node& r) { #change
    node res;
    res.sum = l.sum + r.sum;
    return res;
  }
  void Push(int u, int l, int r) { #change
    if (st[u].lazy_add == INF) return;
    if (l != r) {
      int v = 2 * u, w = 2 * u + 1;
      if (st[v].lazy_add != INF) st[v].lazy_add +=
      ↪  st[u].lazy_add;
```

```
    else st[v].lazy_add = st[u].lazy_add;
    if (st[w].lazy_add != INF) st[w].lazy_add +=
    ↪ st[u].lazy_add;
    else st[w].lazy_add = st[u].lazy_add;
  }
  st[u].sum += (r - l + 1) * st[u].lazy_add;
  st[u].lazy_add = INF;
}
void Build(int u, int s, int e, vector<int>& a) {
  if (s == e) {
    st[u] = Make_node(a[s]);
    return;
  }
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  Build(v, s, m, a);
  Build(w, m + 1, e, a);
  st[u] = Merge(st[v], st[w]);
}
void Update(int u, int s, int e, int l, int r, long
↪ long val) {
  Push(u, s, e);
  if (e < l || r < s) return;
  if (l <= s && e <= r) { #change
    if (st[u].lazy_add != INF) st[u].lazy_add +=
    ↪ val;
    else st[u].lazy_add = val;
    Push(u, s, e);
    return;
  }
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  Update(v, s, m, l, r, val);
  Update(w, m + 1, e, l, r, val);
  st[u] = Merge(st[v], st[w]);
}
void Update(int l, int r, long long val) {
  Update(1, 0, size, l, r, val);
}
node Query(int u, int s, int e, int l, int r) {
  Push(u, s, e);
  if (e < l || r < s) { #change
    return node();
  }
  if (l <= s && e <= r) return st[u];
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  node lsum = Query(v, s, m, l, r);
  node rsum = Query(w, m + 1, e, l, r);
  return Merge(lsum, rsum);
}
node Query(int l, int r) {
  return Query(1, 0, size, l, r);
}
int Idx_query(int u, int s, int e, int l, int r) {
  Push(u, s, e);
  if (e < l || r < s) return -1;
  if (s == e) return st[u].mn == 0 ? s : -1;
  int v = u << 1, w = v | 1, m = s + e >> 1;
  int lq = Idx_query(v, s, m, l, r);
  if (lq == -1) {
    return Idx_query(w, m + 1, e, l, r);
  }
  return lq;
}
int Idx_query(int l, int r) {
  return Idx_query(1, 0, size, l, r);
}
};
```

39 Segment Tree

```
struct node { #change
  long long sum, pref, suff, ans;
  node() {
    sum = pref = suff = ans = 0;
  }
};
struct SegmentTree {
  int size = 1;
  vector<node> st;
  SegmentTree() {}
  SegmentTree(int n) { Initial(n); }
  SegmentTree(vector<int>& a) {
    Initial((int)a.size() - 1);
    Build(1, 0, size, a);
  }
  void Initial(int _n) {
    size = _n;
    int tree_size = 1;
    while (tree_size <= size) tree_size *= 2;
    st.resize(tree_size * 2);
  }
  node Make_node(int val) { #change
    node res;
    res.sum = val;
    res.pref = res.suff = res.ans = max(0, val);
    return res;
  }
  node Merge(node& l, node& r) { #change
    node res;
    res.sum = l.sum + r.sum;
    res.pref = max(l.pref, l.sum + r.pref);
    res.suff = max(r.suff, r.sum + l.suff);
    res.ans = max(max(l.ans, r.ans), l.suff + r.pref);
    return res;
  }
  void Build(int u, int s, int e, vector<int>& a) {
    if (s == e) {
      st[u] = Make_node(a[s]);
      return;
    }
    int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
    Build(v, s, m, a);
    Build(w, m + 1, e, a);
    st[u] = Merge(st[v], st[w]);
  }
  void Update(int u, int s, int e, int k, long long
  ↪ val) {
    if (s == e) { #change
      st[u] = Make_node(val);
      return;
    }
    int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
    if (k <= m) Update(v, s, m, k, val);
    else Update(w, m + 1, e, k, val);
    st[u] = Merge(st[v], st[w]);
  }
  void Update(int k, long long val) {
    Update(1, 0, size, k, val);
  }
  node Query(int u, int s, int e, int l, int r) {
    if (e < l || r < s) { #change
      return node();
    }
    if (l <= s && e <= r) return st[u];
    int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
    node lsum = Query(v, s, m, l, r);
    node rsum = Query(w, m + 1, e, l, r);
    return Merge(lsum, rsum);
  }
  node Query(int l, int r) {
    return Query(1, 0, size, l, r);
  }
};
```

40 SOS DP

```
const int mod = 1e9 + 7;
const int B = 20, N = (1 << B) + 9;
int f[N], g[N], pairF[N], pairG[N], subseqF[N],
↪ subseqG[N], Pow2[N];
void sub(int& a, int& b) { // a = a - b
  a -= b;
  if (a < 0) a += mod;
}
int nC2(int n) {
  return (n * 1LL * (n - 1) / 2) % mod;
}
void forward1() { // sum over subset
  for (int bit = 0; bit < B; bit++) {
    for (int i = 0; i < N; i++) {
      if (i & (1 << bit)) {
        f[i] += f[i ^ (1 << bit)];
      }
    }
  }
}
void backward1() { // exclude sebset sum
  for (int bit = 0; bit < B; bit++) {
    for (int i = N - 1; i >= 0; i--) {
      if (i & (1 << bit)) {
        // f[i] -= f[i ^ (1 << bit)];
        sub(pairF[i], pairF[i ^ (1 << bit)]);
        sub(subseqF[i], subseqF[i ^ (1 << bit)]);
      }
    }
  }
}
void forward2() { // sum over superset
  for (int bit = 0; bit < B; bit++) {
```

```
    for (int i = N - 1; i >= 0; i--) {
      if (i & (1 << bit)) {
        g[i ^ (1 << bit)] += g[i];
      }
    }
  }
}
void backward2() { // exclude superset sum
  for (int bit = 0; bit < B; bit++) {
    for (int i = 0; i < N; i++) {
      if (i & (1 << bit)) {
        // g[i ^ (1 << bit)] -= g[i];
        sub(pairG[i ^ (1 << bit)], pairG[i]);
        sub(subseqG[i ^ (1 << bit)], subseqG[i]);
      }
    }
  }
}
void sosDP(vector<int>& a) {
  Pow2[0] = 1;
  for (int i = 1; i < N; i++) {
    Pow2[i] = (Pow2[i - 1] * 2) % mod;
  }
  for (auto& ai : a) {
    f[ai]++;
    g[ai]++;
  }
  forward1();
  forward2();
  for (int i = 0; i < N; i++) {
    // all pair (i < j)
    pairF[i] = nC2(f[i]);
    pairG[i] = nC2(g[i]);
    // subsequence
    subseqF[i] = Pow2[f[i]] - 1;
    subseqG[i] = Pow2[g[i]] - 1;
  }
  backward1();
  backward2();
}
int subMaskOf(int x) { // x | y = x
  return f[x];
}
int superMaskOf(int x) { // x & y = x
  return g[x];
}
int countPairsWithAnd(int x) { // y & z = x (i < j)
  return pairG[x];
}
int countPairsWithOr(int x) { // y | z = x (i < j)
  return pairF[x];
}
int countSubseqWithAnd(int x) {
  return subseqG[x];
}
int countSubseqWithOr(int x) {
  return subseqF[x];
}
sosDP(a); // clear everything
```

41 Sparse Table RMQ

```
const int N = (int)2e5 + 9;
int lg[N];
void Preprocess() {
  for (int i = 2; i < N; ++i) {
    lg[i] = lg[i / 2] + 1;
  }
}
template <class T> struct RMQ {
  int n = 1, LOG = 1;
  vector<vector<T>> st;
  T Merge(T& a, T& b) {
    return min(a, b); #change
  }
  RMQ() {}
  RMQ(vector<T>& a) {
    if (lg[2] == 0) Preprocess();
    n = (int)a.size(), LOG = __lg(n) + 1;
    st.assign(n, vector<T> (LOG));
    for (int j = 0; j < LOG; j++) {
      for (int i = 0; i + (1 << j) - 1 < n; i++) {
        if (j == 0) st[i][j] = a[i];
        else st[i][j] = Merge(st[i][j - 1], st[i + (1
        ↪  << (j - 1))][j - 1]);
      }
    }
  }
  T Query(int l, int r) {
    // if (l > r) return 0;
    int k = lg[r - l + 1];
    return Merge(st[l][k], st[r - (1 << k) + 1][k]);
  }
};
```

42 Stress Testing

```
// rename file -> testing.sh
set -e
g++ -std=c++17 gen.cpp -o gen
g++ -std=c++17 main.cpp -o main
g++ -std=c++17 brute.cpp -o brute
for((i = 1; ; ++i)); do
  echo $i
  ./gen $i > in
  ./main < in > out
  ./brute < in > out2
  diff -w out out2 || break
done
```

43 Sublime Build

```
// check the gcc version and replace
// name the file -> cp.sublime-build
// ubuntu
{
  "cmd" : ["g++ -std=c++14 $file_name -o
  ↪  $file_base_name && timeout 4s
  ↪  ./$file_base_name<input.txt>output.txt"],
  "selector" : "source.c",
  "shell": true,
  "working_dir" : "$file_path"
}
// windows
{
  "cmd": ["g++.exe","-std=c++14", "${file}", "-o",
  ↪  "${file_base_name}.exe", "&&" ,
  ↪  "${file_base_name}.exe<input.txt>output.txt"],
  "selector":"source.cpp",
  "shell":true,
  "working_dir":"$file_path"
}
// mac
{
  "cmd" : ["g++-14 $file_name -o $file_base_name &&
  ↪  gtimeout 4s
  ↪  ./$file_base_name<input.txt>output.txt"],
  "selector" : "source.c",
  "shell": true,
  "working_dir" : "$file_path"
}
```

44 Suffix Array

```
/*
  for integer, just change string to vector<int> and
  ↪  minimum value of vector must be >= 1
  for integer, lim will be the maximum value of the
  ↪  array
  LCP of suffix (sa[i], sa[i + 1]) = lcp[i + 1]
  O(nlogn)
*/
array<vector<int>, 2> get_sa(string& s, int lim = 128)
↪  {
  int n = s.size() + 1, k = 0, a, b;
  vector<int> x(begin(s), end(s) + 1), y(n), sa(n),
  ↪  lcp(n), ws(max(n, lim)), rank(n);
  x.back() = 0;
  iota(begin(sa), end(sa), 0);
  for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim
  ↪  = p) {
    p = j, iota(begin(y), end(y), n - j);
    for (int i = 0; i < n; ++i) if (sa[i] >= j) y[p++]
    ↪  = sa[i] - j;
    fill(begin(ws), end(ws), 0);
    for (int i = 0; i < n; ++i) ws[x[i]]++;
    for (int i = 1; i < lim; ++i) ws[i] += ws[i - 1];
    for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
    swap(x, y), p = 1, x[sa[0]] = 0;
    for (int i = 1; i < n; ++i) a = sa[i - 1], b =
    ↪  sa[i],
      x[b] = (y[a] == y[b] && y[a + j] == y[b + j]) ?
      ↪  p - 1 : p++;
  }
  for (int i = 1; i < n; ++i) rank[sa[i]] = i;
  for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
    for (k && k--, j = sa[rank[i] - 1]; s[i + k] ==
    ↪  s[j + k]; k++);
```

```
  sa.erase(sa.begin()), lcp.erase(lcp.begin());
  return {sa, lcp};
}

/*
  O(|S| + |alphabet|) Suffix Array
  LIM := max{s[i]} + 2
  LCP of suffix (sa[i], sa[i + 1]) = lcp[i]
*/
void inducedSort(const vector<int>& vec, int
↪  val_range, vector<int>& SA, const vector<int>& sl,
↪  const vector<int>& lms_idx) {
  vector<int> l(val_range, 0), r(val_range, 0);
  for (int c : vec) {
    ++r[c]; if (c + 1 < val_range) ++l[c + 1];
  }
  partial_sum(l.begin(), l.end(), l.begin());
  partial_sum(r.begin(), r.end(), r.begin());
  fill(SA.begin(), SA.end(), -1);
  for (int i = lms_idx.size() - 1; i >= 0; --i)
  ↪  SA[--r[vec[lms_idx[i]]]] = lms_idx[i];
  for (int i : SA) if (i > 0 and sl[i - 1]) SA[l[vec[i
  ↪  - 1]]++] = i - 1;
  fill(r.begin(), r.end(), 0);
  for (int c : vec) ++r[c];
  partial_sum(r.begin(), r.end(), r.begin());
  for (int k = SA.size() - 1, i = SA[k]; k; --k, i =
  ↪  SA[k]) {
    if (i and !sl[i - 1]) SA[--r[vec[i - 1]]] = i - 1;
  }
}
vector<int> suffixArray(const vector<int>& vec, int
↪  val_range) {
  const int n = vec.size();
  vector<int> sl(n), SA(n), lms_idx;
  for (int i = n - 2; i >= 0; --i) {
    sl[i] = vec[i] > vec[i + 1] or (vec[i] == vec[i +
    ↪  1] and sl[i + 1]);
    if (sl[i] and !sl[i + 1]) lms_idx.emplace_back(i +
    ↪  1);
  }
  reverse(lms_idx.begin(), lms_idx.end());
  inducedSort(vec, val_range, SA, sl, lms_idx);
  vector<int> new_lms_idx(lms_idx.size()),
  ↪  lms_vec(lms_idx.size());
  for (int i = 0, k = 0; i < n; ++i) {
    if (SA[i] > 0 and !sl[SA[i]] and sl[SA[i] - 1])
    ↪  new_lms_idx[k++] = SA[i];
  }
  int cur = 0; SA[n - 1] = 0;
  for (int k = 1; k < new_lms_idx.size(); ++k) {
    int i = new_lms_idx[k - 1], j = new_lms_idx[k];
    if (vec[i] ^ vec[j]) {
      SA[j] = ++cur; continue;
    }
    bool flag = 0;
    for (int a = i + 1, b = j + 1; ; ++a, ++b) {
      if (vec[a] ^ vec[b]) {
        flag = 1; break;
      }
      if ((!sl[a] and sl[a - 1]) or (!sl[b] and sl[b -
      ↪  1])) {
        flag = !(!sl[a] and sl[a - 1] and !sl[b] and
        ↪  sl[b - 1]); break;
      }
    }
    SA[j] = flag ? ++cur : cur;
  }
  for (int i = 0; i < lms_idx.size(); ++i) lms_vec[i]
  ↪  = SA[lms_idx[i]];
  if (cur + 1 < lms_idx.size()) {
    auto lms_SA = suffixArray(lms_vec, cur + 1);
    for (int i = 0; i < lms_idx.size(); ++i)
    ↪  new_lms_idx[i] = lms_idx[lms_SA[i]];
  }
  inducedSort(vec, val_range, SA, sl, new_lms_idx);
  ↪  return SA;
}
vector<int> getSuffixArray(const string& s, const int
↪  LIM = 128) { // change limit for integer array,
↪  (integer > 0)
  vector<int> vec(s.size() + 1);
  copy(begin(s), end(s), begin(vec)); vec.back() =
  ↪  '!';
  auto ret = suffixArray(vec, LIM);
  ret.erase(ret.begin()); return ret;
}
// build RMQ on it to get LCP of any two suffix
vector<int> getLCParray(const string& s, const
↪  vector<int>& SA) {
  int n = s.size(), k = 0;
  vector<int> lcp(n), rank(n);
  for (int i = 0; i < n; ++i) rank[SA[i]] = i;
  for (int i = 0; i < n; ++i, k ? --k : 0) {
    if (rank[i] == n - 1) {
      k = 0; continue;
    }
    int j = SA[rank[i] + 1];
    while (i + k < n and j + k < n and s[i + k] == s[j
    ↪  + k]) ++k;
    lcp[rank[i]] = k;
  }
  lcp[n - 1] = 0; return lcp;
}
int lower_bound(string& s, string& t, vector<int>& sa)
↪  {
  int n = s.size(), m = t.size(), lo = 0, hi = n - 1;
  while (lo <= hi) {
    int mid = (lo + hi) / 2;
    if (s.substr(sa[mid], m) < t) lo = mid + 1;
    else hi = mid - 1;
  }
  return lo;
}
int upper_bound(string& s, string& t, vector<int>& sa)
↪  {
  int n = s.size(), m = t.size(), lo = 0, hi = n - 1;
  while (lo <= hi) {
    int mid = (lo + hi) / 2;
    if (s.substr(sa[mid], m) <= t) lo = mid + 1;
    else hi = mid - 1;
  }
  return lo;
}
int find_occurrence(string& s, string& t, vector<int>&
↪  sa) {
  return upper_bound(s, t, sa) - lower_bound(s, t,
  ↪  sa);
}
const int N = 1e6 + 9;
int64_t distPref[N];
void kthSubstringDistinctPreprocess(vector<int>& sa,
↪  vector<int>& lcp) {
  int last = 0, n = sa.size();
  for (int i = 0; i < n; i++) {
    distPref[i] = n - sa[i] - last;
    if (i) distPref[i] += distPref[i - 1];
    last = lcp[i];
  }
}
array<int, 2> kthSubstringDistinctPos(string& s,
↪  int64_t k, vector<int>& sa, vector<int>& lcp) {
  int n = s.size(); k--;
  int i = upper_bound(distPref, distPref + n, k) -
  ↪  distPref;
  int len = k - (i == 0 ? 0 : distPref[i - 1]) + (i ==
  ↪  0 ? 0 : lcp[i - 1]) + 1;
  return {i, len};
}
string kthSubstringDistinct(string& s, int64_t k,
↪  vector<int>& sa, vector<int>& lcp) {
  auto [i, len] = kthSubstringDistinctPos(s, k, sa,
  ↪  lcp);
  return s.substr(sa[i], len);
}
int64_t pref[N];
void kthSubstringPreprocess(vector<int>& sa) {
  int n = sa.size();
  for (int i = 0; i < n; i++) {
    pref[i] = n - sa[i];
    if (i) pref[i] += pref[i - 1];
  }
}
string kthSubstring(string& s, int64_t k, vector<int>&
↪  sa, vector<int>& lcp) {
  int n = s.size();
  int64_t lo = 1, hi = distPref[n - 1];
  while (lo <= hi) {
    int64_t mid = (lo + hi) >> 1;
    auto [i, len] = kthSubstringDistinctPos(s, mid,
    ↪  sa, lcp);
    int64_t totCnt = (i == 0 ? 0 : pref[i - 1]) + len;
    int mn = len;
    for (int j = i; j < n; j++) {
      mn = min(mn, lcp[j]);
```

```
      if (mn == 0) break;
      totCnt += mn;
    }
    if (totCnt < k) lo = mid + 1;
    else hi = mid - 1;
  }
  return kthSubstringDistinct(s, lo, sa, lcp);
}
int main() {
  string s;
  auto sa = getSuffixArray(s);
  auto lcp = getLCParray(s, sa);
  kthSubstringDistinctPreprocess(sa, lcp);
  kthSubstringPreprocess(sa);
}
```

45 Topological Sort

```
bool topological_sort(vector<vector<int>>& adj) {
  int n = adj.size();
  vector<int> in_degree(n + 1);
  for (int u = 1; u <= n; u++) {
    for (auto& v : adj[u]) in_degree[v]++;
  }
  // without sorting the order
  queue<int> q;
  // sort with lexicographically
  // priority_queue<int, vector<int>, greater<int>> q;
  for (int u = 1; u <= n; u++) {
    if (in_degree[u] == 0) {
      q.push(u);
    }
  }
  if (q.empty()) return 0;
  vector<int> order;
  while (!q.empty()) {
    # change the queue operation
    int u = q.front();
    q.pop();
    for (auto& v : adj[u]) {
      in_degree[v]--;
      if (in_degree[v] == 0) {
        q.push(v);
      }
    }
    order.push_back(u);
  }
  return (int)order.size() == n;
}
```

46 Trie

```
// iterative
const int N = 1e6 + 9, A = 26;
int nxt[N][A], pref_cnt[N], word_cnt[N], node = 2;
void insert(string& s) {
  int u = 1;
  for (auto& c : s) {
    int idx = c - 'a';
    if (!nxt[u][idx]) nxt[u][idx] = node++;
    u = nxt[u][idx];
    pref_cnt[u]++;
  }
  word_cnt[u]++;
}
int countPref(string& s) {
  int u = 1;
  for (auto& c : s) {
    int idx = c - 'a';
    if (!nxt[u][idx]) return 0;
    u = nxt[u][idx];
  }
  return pref_cnt[u];
}
int find(string& s) {
  int u = 1;
  for (auto& c : s) {
    int idx = c - 'a';
    if (!nxt[u][idx]) return 0;
    u = nxt[u][idx];
  }
  return word_cnt[u];
}
void erase(string& s) {
  if (find(s) == 0) return;
  int u = 1;
  for (auto& c : s) {
    int idx = c - 'a';
    int v = nxt[u][idx];
    pref_cnt[v][idx]--;
    u = v;
  }
  word_cnt[u]--;
}
// trie integer iterative (max xor, min xor)
const int N = 2e5 * 31 + 9, A = 2;
int nxt[N][A], pref_cnt[N], int_cnt[N], node = 2;
void insert(int num) {
  int u = 1;
  for (int bit = 30; bit >= 0; bit--) {
    int idx = (num >> bit) & 1;
    if (!nxt[u][idx]) nxt[u][idx] = node++;
    u = nxt[u][idx];
    pref_cnt[u]++;
  }
  int_cnt[u]++;
}
int find(int num) {
  int u = 1;
  for (int bit = 30; bit >= 0; bit--) {
    int idx = (num >> bit) & 1;
    if (!nxt[u][idx]) return 0;
    u = nxt[u][idx];
  }
  return int_cnt[u];
}
void erase(int num) {
  if (find(num) == 0) return;
  int u = 1;
  for (int bit = 30; bit >= 0; bit--) {
    int idx = (num >> bit) & 1;
    int v = nxt[u][idx];
    pref_cnt[v]--;
    u = v;
  }
  int_cnt[u]--;
}
int maxXor(int num) {
  int res = 0, u = 1;
  for (int bit = 30; bit >= 0; bit--) {
    int idx = (num >> bit) & 1;
    int flip = idx ^ 1;
    if (nxt[u][flip] && pref_cnt[nxt[u][flip]]) {
      res += (1 << bit);
      u = nxt[u][flip];
    } else {
      u = nxt[u][idx];
    }
  }
  return res;
}
int minXor(int num) {
  int res = 0, u = 1;
  for (int bit = 30; bit >= 0; bit--) {
    int idx = (num >> bit) & 1;
    if (nxt[u][idx] && pref_cnt[nxt[u][idx]]) {
      u = nxt[u][idx];
    } else {
      res += (1 << bit);
      u = nxt[u][idx ^ 1];
    }
  }
  return res;
}
```

47 XOR Basis

```
using ll = long long;
const int D = 60;
struct XorBasis {
  // number of subsequences of xor-sum X, for ith
  ↪  prefix -> 2 ^ (i - sz);
  ll basis[D] = {};
  int sz = 0, n = 0;
  void insert(ll x) {
    for (int i = D - 1; i >= 0; i--) {
      if (!((x >> i) & 1)) continue;
      if (!basis[i]) {
        basis[i] = x;
        sz++;
        break;
      }
      x ^= basis[i];
    } n++;
  }
  bool canRepresent(ll x) {
    for (int i = D - 1; i >= 0; i--) {
      if ((x >> i) & 1) x ^= basis[i];
    }
```

```
  return x == 0;
}
ll maxXor(ll x = 0) {
  for (int i = D - 1; i >= 0; i--) {
    if ((x ^ basis[i]) > x) x ^= basis[i];
  }
  return x;
}
ll minXor() { // except xor 0
  for (int i = 0; i < D; i++) {
    if (basis[i]) return basis[i];
  }
  return 0;
}
ll maxXorWith(ll x) {
  return maxXor(x);
}
ll minXorWith(ll x) {
  for (int i = D - 1; i >= 0; i--) {
    if ((x ^ basis[i]) < x) x ^= basis[i];
  }
  return x;
}
ll countDistinctXors() {
  return 1LL << sz;
}
ll kthXor(ll k) { // returns k + 1 th xor, 1st zor
↪  is 0
  ll res = 0;
  ll tot = countDistinctXors();
  if (tot < k) return -1;
  for (int i = D - 1; i >= 0; i--) {
    if (basis[i]) {
      ll low = tot / 2;
      if ((low < k && (res & 1 << i) == 0) ||
        (low >= k && (res & 1 << i) > 0)) res ^=
        ↪  basis[i];
      if (low < k) k -= low;
      tot /= 2;
    }
  }
  return res;
}
ll kthLargestXor(ll k) {
  return kthXor(countDistinctXors() - 1 - k);
}
ll kthXorAllCombinations(ll k) {
  if (n - sz > 60) return kthXor(1);
  ll totComb = 1LL << n;
  ll disComb = countDistinctXors();
  ll dupPerDis = totComb / disComb;
  ll disIdx = (k - 1) / dupPerDis + 1;
  return kthXor(disIdx);
}
ll kthLargestXorAllCombinations(ll k) {
  if (n - sz > 60) return
  ↪  kthXor(countDistinctXors());
  ll totComb = 1LL << n;
  ll disComb = countDistinctXors();
  ll dupPerDis = totComb / disComb;
    ll disIdx = (k - 1) / dupPerDis + 1;
    ll desIdx = disComb - disIdx + 1;
    return kthXor(desIdx);
  }
  ll countSubsetsLessThan(ll x) {
    ll lo = 0, hi = countDistinctXors();
    while (lo <= hi) {
      ll mid = (lo + hi) / 2;
      if (kthXor(mid) < x) lo = mid + 1;
      else hi = mid - 1;
    }
    return hi;
  }
  ll countSubsetsWithXor(ll x) {
    if (!canRepresent(x)) return 0;
    return 1LL << (n - sz); // if n > 60 use mod
  }
};
```

48 Z Function

```
// 0 based, longest substring lenght starting at
↪  position i that is also a prefix
// String: a b c a b c b b
// Index:  0 1 2 3 4 5 6 7
// Z[]:    - 0 0 3 0 0 0 0
vector<int> get_z(string& s) {
  int n = s.size(), l = 0, r = 0;
  vector<int> z(n);
  z[0] = n;
  for (int i = 1; i < n; i++) {
    if (i < r) z[i] = min(r - i, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]])
    ↪  z[i]++;
    if (i + z[i] > r) l = i, r = i + z[i];
  }
  return z;
}
```

49 Mathematical Formulas & Notes

1. **PI up to 31:** 3.14159265358979323846264338327 95
2. **PI value in CPP:** $2 * acos(0), 2 * asin(1), M\_PI$
3. **Formula for angle C using the Law of Cosines** $C = \cos^{-1}\left(\frac{a^2+b^2-c^2}{2ab}\right)$
4. **Sum remainder: n mod 1+n mod 2+ n mod 3 +.....+n mod m :** n*m-sum of divisors form 1 to n
5. **A number is divisible by 60 if and only if it is divisible by 3 and 20**
6. **All numbers greater than 1099 can be written as a sum of 11 and 111**
7. **Legendre's formula:** $\nu_p(n!) = \sum_{i=1}^{L} \left\lfloor \frac{n}{p^i} \right\rfloor$,where $L = \lfloor \log_p n \rfloor$
8. ${}^nC_r = \binom{n}{r} = \frac{n!}{r!(n-r)!}$, ${}^nP_r = \frac{n!}{(n-r)!}$
9. $(a-b)\%M = (a\%M - b\%M + M)\%M$
10. $(a/b)\%M = (a\%M * b^{-1}\%M)\%M$
11. $(a^b)\%M = ((a\%M)^b)\%M$
12. **Euler's Totient Function (ETF):** Count of numbers less than n that are co-prime to n is, $\phi(n) = n * \prod_{p|n}(1 - \frac{1}{p})$ Here, p = distinct prime factors of n
13. **Congruence:** $a \equiv b \pmod{M}$ if $a\%M = b\%M$ and $M \mid (a-b)$
14. **Euler's Theorem:** $a^{\phi(M)} \equiv 1 \pmod{M}$, where a and M are co-prime
15. **Fermat's Little Theorem:** $a^{M-1} \equiv 1 \pmod{M}$, where M is prime
16. $a^b \equiv a^{b \bmod \phi(M)} \pmod{M}$ or it can be written that, $a^b \bmod M = a^{b \bmod \phi(M)} \bmod M$ or, $a^b \bmod M = a^{b \bmod M-1} \bmod M$
17. **x steps forward or backward in a circular number range:** $newPos = l + ((pos - l + x)\%N) + N)\%N$ Where $N = l - r + 1$ (total numbers in range)
18. **Stars & Bars:** $x_1 + x_2 + \cdots + x_k = n$ with $x_i \geq 0$ has $\binom{n+k-1}{n}$ solutions and with $x_i > 0$ has $\binom{n-1}{k-1}$ solutions.
19. Number of subsequences of length k from an array of size n such that x appears at least once in the subsequence, where that x appears c times in the array: ${}^nC_k - {}^{n-c}C_k$
20. **Hockey-stick Identity:** $n > r, \sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}$

21. $a^k - b^k = (a-b)\cdot(a^{k-1}b^0 + a^{k-2}b^1 + \ldots + a^0b^{k-1})$
22. $ab \mod ac = a(b \mod c)$
23. $|a-b|+|b-c|+|c-a| = 2(\max(a,b,c)-\min(a,b,c))$
24. if $a\cdot b \le c$ then $a \le \left\lfloor \frac{c}{b} \right\rfloor$ Same for $<, \le, >, \ge$
25. For positive integer n & arbitrary real numbers m, x, $\left\lfloor \frac{\lfloor x/m \rfloor}{n} \right\rfloor = \left\lfloor \frac{x}{mn} \right\rfloor$ and $\left\lceil \frac{\lceil x/m \rceil}{n} \right\rceil = \left\lceil \frac{x}{mn} \right\rceil$

Simple Formulas

1. $(a \pm b)^3 = a^3 \pm 3a^2b + 3ab^2 \pm b^3$
2. $a^3 \pm b^3 = (a \pm b)(a^2 \mp ab + b^2)$
3. $\sin(A \pm B) = \sin A \cos B \pm \cos A \sin B$
4. $\cos(A \pm B) = \cos A \cos B \mp \sin A \sin B$
5. $\tan(A \pm B) = \frac{\tan A \pm \tan B}{1 \mp \tan A \tan B}$
6. $\sin A \pm \sin B = 2\sin\left(\frac{A \pm B}{2}\right)\cos\left(\frac{A \mp B}{2}\right)$
7. $\cos A + \cos B = 2\cos\left(\frac{A+B}{2}\right)\cos\left(\frac{A-B}{2}\right)$
8. $\cos A - \cos B = 2\sin\left(\frac{A+B}{2}\right)\sin\left(\frac{B-A}{2}\right)$
9. $\sin A \sin B = -\frac{1}{2}[\cos(A+B) - \cos(A-B)]$
10. $\cos A \cos B = \frac{1}{2}[\cos(A+B) + \cos(A-B)]$
11. $\sin A \cos B = \frac{1}{2}[\sin(A+B) + \sin(A-B)]$
12. $\sin 2\theta = 2\sin\theta\cos\theta = \frac{2\tan\theta}{1+\tan^2\theta}$
13. $\cos 2\theta = \cos^2\theta - \sin^2\theta = 2\cos^2\theta - 1$
    $= 1 - 2\sin^2\theta = \frac{1-\tan^2\theta}{1+\tan^2\theta}$
14. $\tan 2\theta = \frac{2\tan\theta}{1-\tan^2\theta}$
15. $\sin 3\theta = 3\sin\theta - 4\sin^3\theta$
16. $\cos 3\theta = 4\cos^3\theta - 3\cos\theta$
17. $\tan 3\theta = \frac{3\tan\theta - \tan^3\theta}{1 - 3\tan^2\theta}$
18. $1 + \cos 2\theta = 2\cos^2\theta$
19. $1 - \cos 2\theta = 2\sin^2\theta$
20. $1 \pm \sin 2\theta = (\cos\theta \pm \sin\theta)^2$
21. $\sin^{-1} x = \cos^{-1}\sqrt{1-x^2}$
22. $2\sin^{-1} x = \sin^{-1}(2x\sqrt{1-x^2})$
23. $2\cos^{-1} x = \cos^{-1}(2x^2 - 1)$
24. $2\tan^{-1} x = \tan^{-1}\left(\frac{2x}{1-x^2}\right)$
25. $2\tan^{-1} x = \sin^{-1}\left(\frac{2x}{1+x^2}\right)$
26. $2\tan^{-1} x = \cos^{-1}\left(\frac{1-x^2}{1+x^2}\right)$
27. $3\sin^{-1} x = \sin^{-1}(3x - 4x^3)$
28. $3\cos^{-1} x = \cos^{-1}(4x^3 - 3x)$
29. $3\tan^{-1} x = \tan^{-1}\left(\frac{3x - x^3}{1-3x^2}\right)$
30. $\sin^{-1} x \pm \sin^{-1} y = \sin^{-1}\left[x\sqrt{1-y^2} \pm y\sqrt{1-x^2}\right]$
31. $\cos^{-1} x \pm \cos^{-1} y = \cos^{-1}\left[xy \mp \sqrt{1-x^2}\sqrt{1-y^2}\right]$
32. $\tan^{-1} x \pm \tan^{-1} y = \tan^{-1}\left(\frac{x \pm y}{1 \mp xy}\right)$
33. $\tan^{-1} x + \tan^{-1} y + \tan^{-1} z = \tan^{-1}\left(\frac{x+y+z-xyz}{1-xy-yz-zx}\right)$

Short Formulas

1. $\sin(-\theta) = -\sin\theta$
2. $\cos(-\theta) = \cos\theta$
3. $\tan(-\theta) = -\tan\theta$
4. $\sin\left(\frac{\pi}{2} - \theta\right) = \cos\theta$
5. $\cos\left(\frac{\pi}{2} - \theta\right) = \sin\theta$
6. $\tan\left(\frac{\pi}{2} - \theta\right) = \cot\theta$
7. $\cot\left(\frac{\pi}{2} - \theta\right) = \tan\theta$
8. $\csc\left(\frac{\pi}{2} - \theta\right) = \sec\theta$
9. $\sec\left(\frac{\pi}{2} - \theta\right) = \csc\theta$
10. $\sin\left(\frac{\pi}{2} + \theta\right) = \cos\theta$
11. $\cos\left(\frac{\pi}{2} + \theta\right) = -\sin\theta$
12. $\tan\left(\frac{\pi}{4} + \theta\right) = \frac{1+\tan\theta}{1-\tan\theta}$
13. $\tan\left(\frac{\pi}{4} - \theta\right) = \frac{1-\tan\theta}{1+\tan\theta}$
14. $\sin(\pi + \theta) = -\sin\theta$
15. $\cos(\pi + \theta) = -\cos\theta$
16. $\sin(\pi - \theta) = \sin\theta$
17. $\cos(\pi - \theta) = -\cos\theta$
18. $\log_a a = 1$
19. $a^{\log_a b} = b$
20. $\log_b a = \frac{\log a}{\log b}$
21. $\log a^m = m \log a$