Competitive Programmer's CodeBook

MIST_EaglesExpr

Syed Mafijul Islam, 202214105
Md. Tanvin Sarkar Pallab, 202214062
Shihab Ahmed, 202314049

October 13, 2024

Contents

1 Useful Tips

Big Integer C++ __int128_t

C++ FastIO
ios::sync_with_stdio(false); cin.tie(nullptr);

Python FastIO
import sys;
input = sys.stdin.readline

Input From File
freopen("input.txt", "r", stdin);

Python Array Input
list(map(int, input().split()))

Vim Setup

```
"install xclip, vim-gtk3
set nocp ai bs=2 hls ic is lbr ls
    =2 mouse=a nu ru sc scs smd so
    =3 sw=4 ts=4
filetype plugin indent on
syn on
map gA m'ggVG"+y''

inoremap {<CR> {<CR>}<Esc>ko
inoremap [<CR> [<CR>]<Esc>ko
inoremap (<CR> (<CR>)<Esc>ko
autocmd FileType cpp map <F9> :w<
    CR> :!clear; g++ % -DONPC -o
    %:r && ./%:r<CR>
nnoremap = gg=G
```

2 Formula

2.1 Area Formula

Rectangle $Area = length * width$

Square $Area = Side * Side$

Triangle $Area = \frac{1}{2} * length * width$

Circle $Area = \pi * radius^2$

Parallelogram $Area = base * height$

Pyramid Base $Area = \frac{1}{2} * base * slantHeight$

Polygon

a $Area = \frac{1}{2} |\sum_{n=1}^{n-1} (x_i y_{i+1})|$

b $Area = a + \frac{b}{2} - 1$ (for int coordinates). Here $a =$ int points inside polygon and $b =$ int points outside polygon.

2.2 Perimeter Formulas

Rectangle $Perimeter = 2 * (length + width)$

Square $Perimeter = 4 * side$

Triangle $Perimeter = 4 * side$

Circle $Perimeter = 2 * \pi * radius$

2.3 Volume Formula

Cube $Volume = side^3$

Rect Prism $Volume = length * width * height$

Cylinder $Volume = \pi * radius^2 * height$

Sphere $Volume = \frac{4}{3} * \pi * radius^3$

Pyramid $Volume = \frac{1}{3} * baseArea * height$

2.4 Surface Area Formula

Cube $SurfaceArea = 6 * side^2$

Rectangle Prism $SurfaceArea = 2 * (length * width + length * height + width * height)$

Cylinder $SurfaceArea = 2 * \pi * radius * (radius + height)$

Sphere $SurfaceArea = 4 * \pi * radius^2$

Pyramid $SurfaceArea = basearea + \frac{1}{2} * perimeterOfBase * slantHeight$

2.5 Triangles

Side Lengths a, b, c

Semi Perimeter $p = \frac{a+b+c}{2}$

Area $A = \sqrt{p(p-a)(p-b)(p-c)}$

Circumstance $R = \frac{abc}{4A}$

In Radius $r = \frac{A}{p}$

2.6 Summation Of Series

- $c^k + c^{k+1} + ... + c^n = c^{n+1} - c^k$
- $1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$
- $1^2 + 2^2 + 3^2 + ... + n^2 = \frac{n(n+1)(2n+1)}{6}$
- $1^3 + 2^3 + 3^3 + ... + n^3 = (\frac{n(n+1)}{2})^2$

2.7 Miscellaneous

- $2^{100} = 2^{50} * 2^{50}$
- $\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$
- $logn! = log1 + log2 + ... + logn$

3 Graph Theory

All about graph.

3.1 BFS

```
void bfs(int start, int target = -1) {
  queue<int> q;
  q.push(start);
  vis[start] = true;
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int i : adj[u]) {
      if (!vis[i]) {
        vis[i] = true;
        q.push(i);
      }
    }
  }
}
```

3.2 DFS

```
map<int,vector<int>> adj;
map<int,int>visited,parent,level,color
    ;

void dfs(int start)
{
  visited[start]=1;
  for (auto child : adj[start])
  {
    if (!visited[child])
    {
      dfs(child);
    }
  }
  visited[start]=2;
}
```

3.3 Dijkstra Algorithm

```
void Dijkstra(int start) {
  // vector<pair<int, int>> adj[N];
  priority_queue<pair<int, int>,
      vector<pair<int, int>>, greater<
      pair<int, int>>> pq;
  pq.push({0, start});
  while (!pq.empty()) {
    auto it = pq.top();
    pq.pop();
    int wt = it.first;
    int u = it.second;
    if (vis[u])
      continue;
    vis[u] = 1;
    for (pair<int, int> i : adj[u]) {
      int adjWt = i.second;
      int adjNode = i.first;
      if (dist[adjNode] > wt + adjWt)
          {
        dist[adjNode] = wt + adjWt;
        pq.push({dist[adjNode],
            adjNode});
      }
    }
  }
}
```

3.4 Bellman Ford

```
vector<int> dist;
vector<int> parent;
vector<vector<pair<int, int>>> adj;
// resize the vectors from main
    function
void bellmanFord(int num_of_nd, int
    src) {
  dist[src] = 0;
  for (int step = 0; step < num_of_nd;
       step++) {
    for (int i = 1; i <= num_of_nd; i
        ++) {
      for (auto it : adj[i]) {
        int u = i;
        int v = it.first;
        int wt = it.second;
        if (dist[u] != inf &&
            ((dist[u] + wt) < dist[v])
            ) {
          if (step == num_of_nd - 1) {
            cout << "Negative cycle
                found\n ";
                return;
          }
          dist[v] = dist[u] + wt;
          parent[v] = u;
        }
      }
    }
  }
  for (int i = 1;i <= num_of_nd; i++)
    cout << dist[i] << " ";
  cout << endl;
}
```

3.5 Floyed Warshall Algorithm

```
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
typedef vector<int> VI;
typedef vector<VI> VVI;
bool FloydWarshall(VVT &w, VVI &prev)
    {
  int n = w.size();
  prev = VVI(n, VI(n, -1));
  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int 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;
        }
      }
    }
  }
  for (int i = 0; i < n; i++)
    if (w[i][i] < 0)
      return false;
```

```
  return true;
}
```

3.6 Kruskal Algorithm (MST)

```
vector<pair<int, pair<int, int>>> Krushkal(vector<pair<int, pair<int, int>>> &edges, int n) {
  sort(edges.begin(), edges.end());
  vector<pair<int, pair<int, int>>> ans;
  DisjointSet D(n);
  for (auto it : edges) {
    if (D.findUPar(it.second.first) != D.findUPar(it.second.second)) {
      ans.push_back({it.first, {it.second.first, it.second.second}});
      D.unionBySize(it.second.first, it.second.second);
    }
  }
  return ans;
}
```

3.7 Prims Algorithm (MST)

```
void Prims(int start) {
  // map<int, vector<pair<int, int>>> adj, ans;
  priority_queue<pair<int, pair<int, int>>, vector<pair<int, pair<int, int>>>, greater<pair<int, pair<int, int>>>> pq;
  pq.push({0, {start, -1}});
  while (!pq.empty()) {
    auto it = pq.top();
    pq.pop();
    int wt = it.first;
    int u = it.second.first;
    int v = it.second.second;
    if (vis[u]) continue;
    vis[u] = 1;
    if (v != -1) ans[u].push_back({v, wt});
    for (pair<int, int> i : adj[u]) {
      int adjWt = i.second;
      int adjNode = i.first;
      if (!vis[adjNode]) pq.push({adjWt, {adjNode, u}});
    }
  }
}
```

3.8 Strongly Connected Components

```
vector<bool> visited; // keeps track of which vertices are already visited

// runs depth first search starting at vertex v.
// each visited vertex is appended to the output vector when dfs leaves it.
void dfs(int v, vector<vector<int>> const &adj, vector<int> &output) {
  visited[v] = true;
  for (auto u : adj[v])
    if (!visited[u])
      dfs(u, adj, output);
  output.push_back(v);
}

// input: adj -- adjacency list of G
// output: components -- the strongy connected components in G
// output: adj_cond -- adjacency list of G^SCC (by root vertices)
void scc(vector<vector<int>> const &adj, vector<vector<int>> &components, vector<vector<int>> &adj_cond) {
  int n = adj.size();
  components.clear(), adj_cond.clear();

  vector<int> order; // will be a sorted list of G's vertices by exit time

  visited.assign(n, false);

  // first series of depth first searches
  for (int i = 0; i < n; i++)
    if (!visited[i])
      dfs(i, adj, order);

  // create adjacency list of G^T
  vector<vector<int>> adj_rev(n);
  for (int v = 0; v < n; v++)
    for (int u : adj[v])
      adj_rev[u].push_back(v);

  visited.assign(n, false);
  reverse(order.begin(), order.end());

  vector<int> roots(n, 0); // gives the root vertex of a vertex's SCC

  // second series of depth first searches
  for (auto v : order)
    if (!visited[v]) {
      std::vector<int> component;
      dfs(v, adj_rev, component);
      components.push_back(component);
      int root = *min_element(begin(component), end(component));
      for (auto u : component)
        roots[u] = root;
    }
```

```
  // add edges to condensation graph
  adj_cond.assign(n, {});
  for (int v = 0; v < n; v++)
    for (auto u : adj[v])
      if (roots[v] != roots[u])
        adj_cond[roots[v]].push_back(
            roots[u]);
}
```

3.9 LCA

```
struct LCA {
  vector<int> height, euler, first,
      segtree, parent;
  vector<bool> visited;
  vector<vector<int>> jump;

  int n;

  LCA(vector<vector<int>> &adj, int
      root = 0) {
    n = adj.size();
    height.resize(n);
    first.resize(n);
    parent.resize(n);
    euler.reserve(n * 2);
    visited.assign(n, false);
    dfs(adj, root);
    int m = euler.size();
    segtree.resize(m * 4);
    build(1, 0, m - 1);

    jump.resize(n, vector<int>(32, -1)
        );

    for(int i=0;i<n;i++) {
        jump[i][0] = parent[i];
    }

    for(int j=1;j<20;j++) {
        for(int i=0;i<n;i++) {
            int mid = jump[i][j-1];
            if(mid != -1) jump[i][j] =
                jump[mid][j-1];
        }
    }
  }

  void dfs(vector<vector<int>> &adj,
      int node, int h = 0) {
    visited[node] = true;
    height[node] = h;
    first[node] = euler.size();
    euler.push_back(node);
    for (auto to : adj[node]) {
      if (!visited[to]) {
        parent[to] = node;
        dfs(adj, to, h + 1);
        euler.push_back(node);
      }
    }
  }

  void build(int node, int b, int e) {
    if (b == e) {
      segtree[node] = euler[b];
    } else {
      int mid = (b + e) / 2;
      build(node << 1, b, mid);
      build(node << 1 | 1, mid + 1, e)
          ;
      int l = segtree[node << 1], r =
          segtree[node << 1 | 1];
      segtree[node] = (height[l] <
          height[r]) ? l : r;
    }
  }

  int query(int node, int b, int e,
      int L, int R) {
    if (b > R || e < L)
      return -1;
    if (b >= L && e <= R)
      return segtree[node];
    int mid = (b + e) >> 1;

    int left = query(node << 1, b, mid
        , L, R);
    int right = query(node << 1 | 1,
        mid + 1, e, L, R);
    if (left == -1)
      return right;
    if (right == -1)
      return left;
    return height[left] < height[right
        ] ? left : right;
  }

  int lca(int u, int v) {
    int left = first[u], right = first
        [v];
    if (left > right)
      swap(left, right);
    return query(1, 0, euler.size() -
        1, left, right);
  }

  int kthParent(int u, int k) {
      for(int i=0;i<19;i++) {
          if(k & (1LL<<i)) u = jump[u
              ][i];
      }
      return u;
  }
};
```

3.10 Max Flow

```
const int N = 505;
int capacity[N][N];
int vis[N], p[N];
int n, m;

int bfs(int s, int t) {
  memset(vis, 0, sizeof vis);
  queue<int> qu;
  qu.push(s);
  vis[s] = 1;
```

```
  while (!qu.empty()) {
    int u = qu.front();
    qu.pop();
    for (int i = 0; i <= n + m + 2; i
        ++) {
      if (capacity[u][i] > 0 && !vis[i
          ]) {
        p[i] = u;
        vis[i] = 1;
        qu.push(i);
      }
    }
  }
  return vis[t] == 1;
}

int maxflow(int s, int t) {
  int cnt = 0;
  while (bfs(s, t)) {
    int cur = t;
    while (cur != s) {
      int prev = p[cur];
      capacity[prev][cur] -= 1;
      capacity[cur][prev] += 1;
      cur = prev;
    }
    cnt++;
  }
  return cnt;
}
```

4 Data Structures

Different Data Structures.

4.1 Segment Tree

```
constexpr int N = 100005;
int arr[N], seg[N];

void build(int ind, int low, int high)
    {
  if (low == high) {
    seg[ind] = arr[low];
    return;
  }
  int mid = (low + high) / 2;
  build(2 * ind + 1, low, mid);
  build(2 * ind + 2, mid + 1, high);
  seg[ind] = seg[2 * ind + 1] + seg[2
      * ind + 2];
}
int query(int ind, int low, int high,
    int l, int r) {
  if (low >= l && high <= r) return
      seg[ind];
  if (low > r || high < l) return 0;
  int mid = (low + high) / 2;
  int left = query(2 * ind + 1, low,
      mid, l, r);
  int right = query(2 * ind + 2, mid +
      1, high, l, r);
  return left + right;
}
void update(int ind, int low, int high
    , int node, int val) {
  if (low == high) {
    seg[ind] = val;
    return;
  }
  int mid = (low + high) / 2;
  if (low <= node && node <= mid)
      update(2 * ind + 1, low, mid,
      node, val);
  else update(2 * ind + 2, mid + 1,
      high, node, val);
  seg[ind] = seg[2 * ind + 1] + seg[2
      * ind + 2];
}
```

4.2 Segment Tree Lazy

```
const int N = 1e5 + 5;
int arr[N], seg[N << 2], lz[N << 2];

void pull(int node, int l, int r) {
  seg[node] = seg[node << 1] + seg[
      node << 1 | 1];
}

void push(int node, int l, int r) {
  int mid = (l + r) >> 1;
  lz[node << 1] += lz[node];
  seg[node << 1] += lz[node] * (mid -
      l + 1);
  lz[node << 1 | 1] += lz[node];
  seg[node << 1 | 1] += lz[node] * (r
      - mid);
  lz[node] = 0;
}

void build(int node, int l, int r) {
  if(l == r) {
    seg[node] = arr[l];
    lz[node] = 0;
    return;
  }
  int mid = (l + r) >> 1;
  build(node << 1, l, mid);
  build(node << 1 | 1, mid + 1, r);
  pull(node, l, r);
}

void update(int node, int l, int r,
    int ql, int qr, int val) {
  if(qr < l || r < ql) return;
  if(ql <= l && r <= qr) {
    seg[node] += val;
    lz[node] += val;
    return;
  }

  push(node, l, r);

  int mid = (l + r) >> 1;
  update(node << 1, l, mid, ql, qr,
      val);
```

```
  update(node << 1 | 1, mid + 1, r, ql
      , qr, val);

  pull(node, l, r);
}

int query(int node, int l, int r, int
   ql, int qr) {
  if(qr < l || r < ql) return 0;
  if(ql <= l && r <= qr) return seg[
     node];

  push(node, l, r);

  int mid = (l + r) >> 1;
  return query(node << 1, l, mid, ql,
     qr) + query(node << 1 | 1, mid +
      1, r, ql, qr);
}
```

4.3 Fenwick Tree

```
int fenwick[N];

void update(int ind, int val) {
  while (ind < N) {
    fenwick[ind] += val;
    ind += ind & -ind;
  }
}
int query(int ind) {
  int sum = 0;
  while (ind > 0) {
    sum += fenwick[ind];
    ind -= ind & -ind;
  }
  return sum;
}
```

4.4 DisjointSet

```
class DisjointSet {
  vector<int> parent, sz;

 public:
  DisjointSet(int n) {
    sz.resize(n + 1);
    parent.resize(n + 2);
    for (int i = 1; i <= n; i++)
       parent[i] = i, sz[i] = 1;
  }
  int findUPar(int u) { return parent[
     u] == u ? u : parent[u] =
     findUPar(parent[u]); }
  void unionBySize(int u, int v) {
    int a = findUPar(u);
    int b = findUPar(v);
    if (sz[a] < sz[b]) swap(a, b);
    if (a != b) {
      parent[b] = a;
      sz[a] += sz[b];
    }
  }
};
```

4.5 TRIE

```
const int N = 26;
class Node {
 public:
  int EoW;
  Node* child[N];
  Node() {
    EoW = 0;
    for (int i = 0; i < N; i++) child[
        i] = NULL;
  }
};

void insert(Node* node, string s) {
  for (size_t i = 0; i < s.size(); i
      ++) {
    int r = s[i] - 'A';
    if (node->child[r] == NULL) node->
        child[r] = new Node();
    node = node->child[r];
  }
  node->EoW += 1;
}

int search(Node* node, string s) {
  for (size_t i = 0; i < s.size(); i
      ++) {
    int r = s[i] - 'A';
    if (node->child[r] == NULL) return
         0;
  }
  return node->EoW;
}

void print(Node* node, string s = "")
    {
  if (node->EoW) cout << s << "\n";
  for (int i = 0; i < N; i++) {
    if (node->child[i] != NULL) {
      char c = i + 'A';
      print(node->child[i], s + c);
    }
  }
}

bool isChild(Node* node) {
  for (int i = 0; i < N; i++)
    if (node->child[i] != NULL) return
         true;
  return false;
}

bool isJunc(Node* node) {
  int cnt = 0;
  for (int i = 0; i < N; i++) {
    if (node->child[i] != NULL) cnt++;
  }
  if (cnt > 1) return true;
  return false;
}

int trie_delete(Node* node, string s,
```

```
    int k = 0) {
  if (node == NULL) return 0;
  if (k == (int)s.size()) {
    if (node->EoW == 0) return 0;
    if (isChild(node)) {
      node->EoW = 0;
      return 0;
    }
    return 1;
  }
  int r = s[k] - 'A';
  int d = trie_delete(node->child[r],
      s, k + 1);
  int j = isJunc(node);
  if (d) delete node->child[r];
  if (j) return 0;
  return d;
}

void delete_trie(Node* node) {
  for (int i = 0; i < 15; i++) {
    if (node->child[i] != NULL)
        delete_trie(node->child[i]);
  }
  delete node;
}
```

4.6 Set Balancing

```
// return middle element of the set
void balance(multiset<int> right,
    multiset<int> &left) {
  while (true) {
    int st = right.size();
    int sl = left.size();
    if (st == sl || st == sl + 1)
        break;
    if (st < sl) right.insert(left.
        begin()), left.erase(left.
        begin());
    else left.insert(right.rbegin()),
        right.erase(right.rbegin());
  }
}

void insert_in_set(multiset<int> &
    right, multiset<int> &left, int
    value) {
  if (right.emptleft()) right.insert(
      value);
  else {
    auto it = right.end();
    it--;
    if (value < *it) right.insert(
        value);
    else left.insert(value);
  }
}
```

5 Algorithms

All about algorithms.

5.1 KMP

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

vector<int> find_matches(string text,
    string pat) {
    int n = pat.length(), m = text.
        length();
    string s = pat + "$" + text;
    vector<int> pi = prefix_function(s
        ), ans;
    for (int i = n; i <= n + m; i++) {
        if (pi[i] == n) {
            ans.push_back(i - 2 * n);
        }
    }
    return ans;
}
```

5.2 Monotonic Stack (Immediate Small)

```
for (int i = n - 1; i >= 0; i--) {
  while (!stk.empty() && v[i] >= v[stk
      .top()]) stk.pop();
  ind[i] = stk.empty() ? -1 : stk.top
      ();
  stk.push(i);
}
// 3 1 5 4 10
// 2 2 4 4 -1
```

5.3 Kadane's Algorithm

```
int maxSubArraySum(vector<int> &a) {
  int size = a.size();
  int maxTill = INT_MIN, maxEnd = 0;
  for (int i = 0; i < size; i++) {
    maxEnd = maxEnd + a[i];
    if (maxTill < maxEnd) maxTill =
        maxEnd;
    if (maxEnd < 0) maxEnd = 0;
  }
  return maxTill;
}
```

5.4 2D Prefix Sum

```
pref[i][j] = a[i][j] + pref[i - 1][j]
    + pref[i][j - 1] - pref[i - 1][j -
     1];
```

6 Number Theory/Math

All about math.

6.1 nCr

```
int inverseMod(int a, int m) { return
    power(a, m - 2); }

int nCr(int n, int r, int m = mod){
    if(r==0) return 1;
    if(r>n) return 0;
    return (fact[n] * inverseMod((fact[
        r] * fact[n-r]) % m , m)) % m;
}
```

6.2 Power

```
int power(int base, int n, int m = mod
    ) {
  if (n == 0) return 1;
  if (n & 1) {
    int x = power(base, n / 2);
    return ((x * x) % m * base) % m;
  }
  else {
    int x = power(base, n / 2);
    return (x * x) % m;
  }
}
```

6.3 Miller Rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod)
     {
  u64 result = 1;
  base %= mod;
  while (e) {
    if (e & 1) result = (u128)result *
         base % mod;
    base = (u128)base * base % mod;
    e >>= 1;
  }
  return result;
}

bool check_composite(u64 n, u64 a, u64
     d, int s) {
  u64 x = binpower(a, d, n);
  if (x == 1 || x == n - 1) return
      false;
  for (int r = 1; r < s; r++) {
    x = (u128)x * x % n;
    if (x == n - 1) return false;
  }
  return true;
};

bool MillerRabin(u64 n, int iter = 5)
    { // returns true if n is
      probably prime, else returns false
      .
  if (n < 4) return n == 2 || n == 3;
  int s = 0;
  u64 d = n - 1;
  while ((d & 1) == 0) {
    d >>= 1;
    s++;
  }

  for (int i = 0; i < iter; i++) {
    int a = 2 + rand() % (n - 3);
    if (check_composite(n, a, d, s))
        return false;
  }
  return true;
}
```

6.4 Sieve

```
const int N = 1e7 + 3;
vector<int> primes;
int notprime[N];

void sieve() {
  primes.push_back(2);
  for (int i = 2; i < N; i += 2) {
    notprime[i] = true;
  }
  for (int i = 3; i < N; i += 2) {
    if (!notprime[i]) {
      primes.push_back(i);
      for (int j = i * i; j < N; j +=
           2 * i) {
        notprime[j] = true;
      }
    }
  }
}
```

6.5 Inverse Mod

```
int modInverse(int a, int m) {
  int m0 = m, t, q;
  int x0 = 0, x1 = 1;
  if (m == 1) return 0;
  while (a > 1) {
    q = a / m;
    t = m;
    m = a % m, a = t;
    t = x0;
    x0 = x1 - q * x0;
    x1 = t;
  }
  if (x1 < 0) x1 += m0;
  return x1;
}
```

6.6 Bitset Sieve

```
const int sieve_size = 10000006;
bitset<sieve_size> sieve;

void Sieve() {
```

```
  sieve.flip();
  int finalBit = sqrt(sieve.size()) +
      1;
  for (int i = 2; i < finalBit; ++i) {
    if (sieve.test(i))
      for (int j = 2 * i; j <
          sieve_size; j += i) sieve.
          reset(j);
  }
}
```

6.7 Divisors

```
constexpr int N = 1000005;

int Prime[N + 4], kk;
bool notPrime[N + 5];
void SieveOf() {
  notPrime[1] = true;
  Prime[kk++] = 2;
  for (int i = 4; i <= N; i += 2)
      notPrime[i] = true;
  for (int i = 3; i <= N; i += 2) {
    if (!notPrime[i]) {
      Prime[kk++] = i;
      for (int j = i * i; j <= N; j +=
          2 * i) notPrime[j] = true;
    }
  }
}

void Divisors(int n) {
  int sum = 1, total = 1;
  int mnP = INT_MAX, mxP = INT_MIN,
      cntP = 0, totalP = 0;
  for (int i = 0; i <= N && Prime[i] *
       Prime[i] <= n; i++) {
    if (n % Prime[i] == 0) {
      mnP = min(mnP, Prime[i]);
      mxP = max(mnP, Prime[i]);
      int k = 0;
      cntP++;
      while (n % Prime[i] == 0) {
        k++;
        n /= Prime[i];
      }

      sum *= (k + 1);  // NOD
      totalP += k;
      int s = 0, p = 1;
      while (k-- >= 0) {
        s += p;
        p *= Prime[i];
      };
      total *= s;  // SOD
    }
  }
  if (n > 1) {
    cntP++, totalP++;
    sum *= 2;
    total *= (1 + n);
    mnP = min(mnP, n);
    mxP = max(mnP, n);
  }
  cout << mnP << " " << mxP << " " <<
      cntP << " " << totalP << " " <<
      sum << " " << total << "\n";
}
```

6.8 Euler's Totient Phi Function

```
const int N = 5000005;
int phi[N];
unsigned long long phiSum[N];
void phiCalc() {
  for (int i = 2; 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;
      }
    }
  }
  for (int i = 2; i < N; i++) {
    phiSum[i] = (unsigned long long)
        phi[i] * (unsigned long long)
        phi[i] + phiSum[i - 1];
  }
}
```

6.9 Log a base b

```
int logab (int a, int b){
  return log2(a) / log2(b);
}
```

7 Dynamic Programming

7.1 LCS

```
string s, t;
vector<vector<int>> dp(3003, vector<
    int>(3003, -1));
vector<vector<int>> mark(3003, vector<
    int>(3003));

int f(int i, int j) {
  if (i < 0 || j < 0) return 0;
  if (dp[i][j] != -1) return dp[i][j];
  int res = 0;
  if (s[i] == t[j]) {
    mark[i][j] = 1;
    res = 1 + f(i - 1, j - 1);
  }
  else {
    int iC = f(i - 1, j);
    int jC = f(i, j - 1);
    if (iC > jC) mark[i][j] = 2;
    else mark[i][j] = 3;
    res = max(iC, jC);
  }
  return dp[i][j] = res;
}

void printWay(int i, int j) {
  if (i < 0 || j < 0) return;
```

```
  if (mark[i][j] == 1) printWay(i - 1,
      j - 1), cout << s[i];
  else if (mark[i][j] == 2) printWay(i
      - 1, j);
  else if (mark[i][j] == 3) printWay(i
      , j - 1);
}
```

7.2 LIS

```
void lis(vector<int> &v) {
  int n = v.size();
  vector<int> dp(n + 1, 1), hash(n);
  int mx = 1, lastInd = 0;
  for (int i = 0; i < n; i++) {
    hash[i] = i;
    for (int prev = 0; prev < i; prev
        ++) {
      if (v[i] > v[prev] && 1 + dp[
          prev] > dp[i]) {
        dp[i] = 1 + dp[prev];
        hash[i] = prev;
      }
    }
    if (mx < dp[i]) {
      mx = dp[i];
      lastInd = i;
    }
  }
  vector<int> printSeq;
  printSeq.push_back(v[lastInd]);
  while (hash[lastInd] != lastInd) {
    lastInd = hash[lastInd];
    printSeq.push_back(v[lastInd]);
  }
  reverse(printSeq.begin(), printSeq.
      end());
  cout << mx << "\n";
  for (int i : printSeq) cout << i <<
      " ";
  cout << "\n";
}
```