Contents

1 Useful Tips **2**
- 1.1 Run with key in VS Code 2

2 Formula **2**
- 2.1 Area Formula 2
- 2.2 Perimeter Formulas 2
- 2.3 Volume Formula 2
- 2.4 Surface Area Formula 2
- 2.5 Triangles 2
- 2.6 Summation Of Series 3
- 2.7 Logarithmic Basics 3
- 2.8 Series 3
  - 2.8.1 Catalan Series 3
  - 2.8.2 Arithmetic Series 3
  - 2.8.3 Geometric Series 3
  - 2.8.4 Derangement Series 3
  - 2.8.5 nth Fibonacci Golden Ratio 3
- 2.9 Miscellaneous 3

3 Graph Theory **3**
- 3.1 BFS 3
- 3.2 DFS 3
- 3.3 Dijkstra Algorithm 4
- 3.4 Bellman Ford 4
- 3.5 Floyed Warshall Algorithm 4
- 3.6 Kruskal Algorithm (MST) 4
- 3.7 Prims Algorithm (MST) 5
- 3.8 Strongly Connected Components . 5
- 3.9 LCA 5
- 3.10 Max Flow 6
- 3.11 Cycle Detection in Directed Graph 7
- 3.12 Cycle Detection in Undirected Graph 7
- 3.13 Bipartite Graph Check 7

4 Data Structures **8**
- 4.1 Segment Tree 8
- 4.2 Segment Tree Lazy 8
- 4.3 Fenwick Tree 8
- 4.4 Disjoint Set 9
- 4.5 TRIE 9
- 4.6 Set Balancing 10
- 4.7 Ordered Set 10

5 Algorithms **10**
- 5.1 KMP 10
- 5.2 Monotonic Stack (Immediate Small) 10
- 5.3 Kadane's Algorithm 10
- 5.4 2D Prefix Sum 11
- 5.5 Next Greater Element 11

6 Number Theory **11**
- 6.1 Prime Numbers Under 1000 11
- 6.2 Divisibility Rules for Numbers . . 11
- 6.3 Divisor Count 11
- 6.4 Leap Year 11
- 6.5 Number of Leap Year in Between . 12
- 6.6 Binary Exponentiation: $(a\hat{b})$ . . . 12
- 6.7 Binary Exponentiation: $(a\hat{b}\hat{c})$ . . . 12
- 6.8 Power 12
- 6.9 nCr 12
- 6.10 Check Prime -O(sqrt(n)) 12
- 6.11 Prime Factorization 12
- 6.12 Miller Rabin 12
- 6.13 Sieve 13
- 6.14 BitSet Sieve 13
- 6.15 Segment Sieve 13
- 6.16 Nth Prime 13
- 6.17 Divisors 13
- 6.18 Log a base b 14
- 6.19 Count 1's from 0 to n 14
- 6.20 Primes Upto 1e9 14
- 6.21 Extd GCD 15
- 6.22 Factorial Mod 15
- 6.23 Modular Operations 15
- 6.24 10 base to M base conversion . . . 16
- 6.25 M base to 10 base conversion . . . 16
- 6.26 Matrix Exponentiation 16

7 Dynamic Programming **16**
- 7.1 LCS (Longest Common Subsequence) 16
- 7.2 LIS (Longest Increasing Subsequence) 16
- 7.3 SOS (Sum Of Subsets) 17

8 Strings **17**
- 8.1 Double Hashing 17
- 8.2 Large Number Multiplication . . . 18
- 8.3 Large Number Summation 18
- 8.4 Large Number Division 19
- 8.5 Large Number Subtraction 19

9 Stress Testing **20**
- 9.1 Bash Stress File 20
- 9.2 C++ Generator File 20

10 Game Theory **20**
- 10.1 Nim Game 20
- 10.2 Miser Nim 20
- 10.3 Grundy Game 20

1 Useful Tips

1.1 Run with key in VS Code

Add the following code in `keybindings.json` file in VS Code, to run onpen git bash terminal, in.txt and out.txt file and press `f5` to run the code.

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

Run CPP File
`g++ .\test.cpp -o test && .\test`

Run CPP File with Input File
`g++ .\test.cpp -o test && .\test < in.txt > out.txt`

Big Integer C++ __int128_t

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

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

Integer - Binary Conversion in C++

```
bitset<size>(val).to_string();
(int)bitset<size>(val).to_ulong
    ();
```

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

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

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 (Pick's formula) $Area = a + \frac{b}{2} - 1$ (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

- $c^k + c^{k+1} + ... + c^n = c^{n+1} - c^k$
- $1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$
- $1^2 + 2^2 + 3^2 + ... + n^2 = \frac{n(n+1)(2n+1)}{6}$
- $1^3 + 2^3 + 3^3 + ... + n^3 = (\frac{n(n+1)}{2})^2$
- $1^4+2^4+3^4+...+n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$

Sum of first n odd numbers n^2

2.7 Logarithmic Basics

- $\log_b 1 = 0$
- $\log_b b = 1$
- $\log_b(AB) = \log_b A + \log_b B$
- $\log_b \left(\frac{A}{B}\right) = \log_b A - \log_b B$
- $\log_b A^x = x \log_b A$
- $\log_a c = \log_a b \cdot \log_b c$
- $b^{\log_b a} = a$
- $x \log_b y = y \log_b x$
- $\log_a b = \frac{1}{\log_b a}$
- $\log_a x = \frac{\log_b x}{\log_b a}$

2.8 Series

2.8.1 Catalan Series

Series: $1, 1, 2, 5, 14, 42, 132, 429, \ldots$
Equation: $C_n = \sum_{k=0}^{n-1} C_k \cdot C_{n-1-k}$

2.8.2 Arithmetic Series

$a_n = a + (n-1) \cdot d$
$S_n = \frac{n}{2} \cdot (2a + (n-1) \cdot d)$

2.8.3 Geometric Series

$a_n = a \cdot r^{n-1}$
$S_n = \frac{a(1-r^n)}{1-r} \quad \text{for} \quad r \neq 1$

2.8.4 Derangement Series

Series: $0, 1, 2, 9, 44, 265, 1854, 14833, 133496,$
$1334961, 14684570, \ldots$
$D_n = (n-1)D_{n-2} + (n-1)D_{n-1}$

2.8.5 nth Fibonacci Golden Ratio

$$f_n = \frac{\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)}{\sqrt{5}}$$

2.9 Miscellaneous

- $2^{100} = 2^{50} * 2^{50}$
- $\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$
- $logn! = log1 + log2 + ... + logn$
- Number of occurrence of a prime number p in $n!$ is $\lfloor \frac{n}{p} \rfloor + \lfloor \frac{n}{p^2} \rfloor + \lfloor \frac{n}{p^3} \rfloor + ... + 0$
- Number of divisors of $p^x q^y$ where p and q are prime is $(x+1) * (y+1)$
- Sum of divisors of $p^x q^y$ where p and q are prime is $(1+p+p^2+...+p^x)(1+q+q^2+...+q^y)$
- Golden Ratio $\Phi \approx 1.618034$
- nth Fibonacci number $F_n = \frac{\Phi n - (1-\Phi)n}{\sqrt{5}}$
- $n(A \cup B) = n(A) + n(B) - n(A \cap B)$
- If $A \cap B = \emptyset$, then $n(A \cup B) = n(A) + n(B)$
- $n(A - B) + n(A \cap B) = n(A)$
- $n(B - A) + n(A \cap B) = n(B)$
- $n(A \cup B) = n(A - B) + n(A \cap B) + n(B - A)$
- $n(A \cup B \cup C) = n(A) + n(B) + n(C) - n(A \cap B) - n(B \cap C) - n(C \cap A) + n(A \cap B \cap C)$
- Area of a regular polygon: $\frac{na^2 \cot\left(\frac{180}{n}\right)}{4}$
- Apex point angle of a regular polygon: $\left(\frac{2n-4}{n}\right) \times 90°$

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;
}
```

3.11 Cycle Detection in Directed Graph

```
// Cycle detection in directed graph
bool isCyclicUtil(vector<vector<int>>&
     adj, int u, vector<bool>& visited
    , vector<bool>& recStack) {
    if (!visited[u]) {
        // Mark the current node as
            visited // and part of
            recursion stack
        visited[u] = true;
        recStack[u] = true;
        // Recur for all the vertices
            adjacent // to this vertex
        for (int x : adj[u]) {
            if (!visited[x] &&
                isCyclicUtil(adj, x,
                    visited, recStack)
                    )
                return true;
            else if (recStack[x])
                return true;
        }
    }
    // Remove the vertex from
        recursion stack
    recStack[u] = false;
    return false;
}
// Function to detect cycle in a
    directed graph
bool isCyclic(vector<vector<int>>& adj
    , int V) {
    vector<bool> visited(V, false);
    vector<bool> recStack(V, false);
    // Call the recursive helper
        function to // detect cycle in
         different DFS trees
    for (int i = 0; i < V; i++) {
        if (!visited[i] &&
            isCyclicUtil(adj, i,
                visited, recStack))
            return true;
    }
    return false;
}
```

3.12 Cycle Detection in Undirected Graph

```
// Cycle detection in undirected graph
bool isCyclicUtil(int v, vector<vector
    <int>>& adj,bool visited[], int
    parent) {
    // Mark the current node as
        visited
    visited[v] = true;
    // Recur for all the vertices //
        adjacent to this vertex
    for (int i : adj[v]) {
        // If an adjacent vertex is
            not visited,// then recur
            for that adjacent
        if (!visited[i]) {
            if (isCyclicUtil(i, adj,
                visited, v))
                return true;
        }
        // If an adjacent vertex is
            visited and is not parent
            of current vertex,then
            there exists a cycle in
            the graph.
        else if (i != parent)
            return true;
    }
    return false;
}
// Returns true if the graph contains
// a cycle, else false.
bool isCyclic(int V, vector<vector<int
    >>& adj) {
    // Mark all the vertices as not
        visited
    bool* visited = new bool[V]{false
        };
    // Call the recursive helper
        function
    // to detect cycle in different
        DFS trees
    for (int u = 0; u < V; u++) {
        // Don't recur for u if it is
            already visited
        if (!visited[u])
            if (isCyclicUtil(u, adj,
                visited, -1))
                return true;
    }
    return false;
}
```

3.13 Bipartite Graph Check

```
bool dfs(int v, int c){
  vis[v]=1;
  col[v]=c;
  for(int child : ar[v]){
    if(vis[child]==0){
    if(!dfs(child,c^1))
      return false;
    }
    else
      if(col[v]==col[child])
        return false;
```

```
  }
  return true;
}
```

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 Disjoint Set

```
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);
  }
}
```

4.7 Ordered Set

```
#include <ext/pb_ds/assoc_container.
    hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define ordered_set tree<int,
    null_type,less<int>, rb_tree_tag,
    tree_order_statistics_node_update>

ordered_set o_set;

// insert function to insert in
// ordered set same as SET STL
o_set.insert(5);
o_set.insert(1);
o_set.insert(2);

// Finding the second smallest element
// in the set using * because
//  find_by_order returns an iterator
*(o_set.find_by_order(1))

// Finding the number of elements
// strictly less than k=4
o_set.order_of_key(4)
```

5 Algorithms

5.1 KMP

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

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

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2005;
int a[N][N], pref[N][N];
int main(){
  int n, m;
  cin >> n >> m;
  for (int i = 1; i <= n; i++){
    for (int j = 1; j <= m; j++){
      cin >> a[i][j];
    }
  }
  for (int x = 1; x <= n; x++){
    for (int y = 1; y <= m; y++){
      pref[x][y] = a[x][y] + pref[x][y
          - 1] + pref[x - 1][y] -
         pref[x - 1][y - 1];
    }
  }
  int q;
  cin >> q;
  while (q--){
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    int sum = pref[x2][y2] - pref[x1 -
        1][y2] - pref[x2][y1 - 1] +
       pref[x1 - 1][y1 - 1];
    cout << sum << "\n";
  }
  return 0;
}
```

5.5 Next Greater Element

```
ll output[1000005];
void nextGreaterElement(ll x[], ll n)
    {
    stack<ll> s;
    s.push(0);
    for (ll i = 0; i < n; i++) {
        while (!s.empty() && x[s.top()
            ] <= x[i]) {
            output[s.top()] = i;
            s.pop();
        }
        s.push(i);
    }
    while (!s.empty()) {
        output[s.top()] = -1;
        s.pop();
    }
}
```

6 Number Theory

6.1 Prime Numbers Under 1000

The prime numbers under 1000 are:

- 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997

6.2 Divisibility Rules for Numbers

3 Sum of digits divisible by 3.

4 Last two digits divisible by 4.

5 Ends in 0 or 5.

6 Divisible by both 2 and 3.

7 Double the last digit, subtract it from the rest; check if the result is divisible by 7. Repeat if necessary.

8 Last three digits divisible by 8.

9 Sum of digits divisible by 9.

11 Difference between sums of digits in odd and even places divisible by 11.

6.3 Divisor Count

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

6.4 Leap Year

```
bool isLeap(ll n)
{
  if (n % 100 == 0)
  if (n % 400 == 0)return true;
    else return false;
  if (n % 4 == 0) return true;
  else return false;
}
```

6.5 Number of Leap Year in Between

```
ll calNum(ll year) {
return (year / 4) - (year / 100) +
 (year / 400);
}
ll leapNum(ll l, ll r) {
  l--;
  return calNum(r) - calNum(l);
}
```

6.6 Binary Exponentiation: (ab̂)

```
ll binaryExp(ll base,ll power,ll MOD =
     mod) {
    ll res = 1;
    while (power) {
        if (power & 1)
            res = (res * base) % MOD;
        base = ((base%MOD)*(base%MOD))
           %MOD;
        power /= 2;
    }
    return res;
}
```

6.7 Binary Exponentiation: (abĉ)

```
ll binaryExp(ll base, ll power, ll
    modulo=mod){
    ll and = 1;
    while (power){
        if (power % 2 == 1)
            ans = (ans * base) %
                modulo;
        base = (base * base) % modulo;
        power /= 2;
    }
    return ans;
}
//Function call: binaryExp(a,binaryExp
    (b,c,mod),mod);
```

6.8 Power

```
ll x = (ll)(pow(base, power) + 1e-18);
```

6.9 nCr

```
int mod = 1e9 + 7;
const int MAX = 1e7 + 5;
vector<int> fact(MAX), ifact(MAX), inv
    (MAX);
void factorial() {
    inv[1] = fact[0] = ifact[0] = 1;
    for (int i = 2; i < MAX; i++)
        inv[i]=inv[mod%i]*(mod-mod/i)%
            mod;
    for (int i = 1; i < MAX; i++)
        fact[i] = (fact[i - 1] * i) %
            mod;
    for (int i = 1; i < MAX; i++)
        ifact[i]=ifact[i-1]*inv[i] %
            mod;
}
int nCr(int n, int r) {
    if (r < 0 || r > n)
        return 0;
    return (int)fact[n] * ifact[r] %
        mod * ifact[n - r] % mod;
}
// first call factorial() function
// then for nCr just call nCr(n,r)
```

6.10 Check Prime -O(sqrt(n))

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

6.11 Prime Factorization

```
vector<ll> factor(ll n) {
    vector<ll>factors;
    while (n % 2 == 0) {
        factors.push_back(2);
        n /= 2;
    }
    while (n % 3 == 0) {
        factors.push_back(3);
        n /= 3;
    }
    for (ll i = 5; i * i <= n; i += 6)
         {
        while (n % i == 0) {
            factors.push_back(i);
            n /= i;
        }
        while (n % (i + 2) == 0) {
            factors.push_back(i + 2);
            n /= (i + 2);
        }
    }
    if (n > 2) factors.push_back(n);
    return factors;
}
```

6.12 Miller Rabin

```
// Ret true if n is prime,unit64_t n
    =11; fun call MillerRabin(n)
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)
    {
  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;
}
```

6.13 Sieve

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

6.14 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);
  }
}
```

6.15 Segment Sieve

```
void SegmentSieve(ll L, ll R){
  //call sieve first
    if (L == 1)
        L++;
    ll maxN = R - L + 1;
    ll a[maxN] = {0};
    for (auto p : prime){
        if (p * p <= R)
        {
            ll x = (L / p) * p;
            if (x < L)
                x += p;
            for (ll i = x; i <= R; i
                += p)
            {
                if (i != p)
                    a[i - L] = 1;
            }
        }
        else
            break;}
    for (ll i = 0; i < maxN; i++)
        if (a[i] == 0)
            cout << i + L << endl;
}
```

6.16 Nth Prime

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

6.17 Divisors

```
constexpr ll N = 1000005;
ll Prime[N + 4], kk;
bool notPrime[N + 5];
void SieveOf() {
    notPrime[1] = true;
    Prime[kk++] = 2;
    for (ll i = 4; i <= N; i += 2)
        notPrime[i] = true;
    for (ll i = 3; i <= N; i += 2) {
        if (!notPrime[i]) {
            Prime[kk++] = i;
            for (ll j = i * i; j <= N;
                 j += 2 * i) notPrime[
                j] = true;
        }
    }
}
void Divisors(ll n) {
    ll sum = 1, total = 1;
    ll mnP = INT_MAX, mxP = INT_MIN,
        cntP = 0, totalP = 0;
    for (ll 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]);
            ll k = 0;
            cntP++;
            while (n % Prime[i] == 0)
                {
                k++;
                n /= Prime[i];
            }
            sum *= (k + 1);  // Number
                 of Divisors
            totalP += k;
            ll s = 0, p = 1;
            while (k-- >= 0) {
                s += p;
                p *= Prime[i];
            };
            total *= s;  // Sum of
                Divisors
        }
    }
    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";
}
```

6.18 Log a base b

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

6.19 Count 1's from 0 to n

```
int cntOnes(int n) {
  // count number of 1s till n
  int cnt = 0;
  for(int i=1;i<=n;i<<=1) {
    int x = (n + 1) / (i << 1);
    cnt += x * i;
    if((n + 1) % i && n & i) cnt += (n
        + 1) % i;
  }
  return cnt;
}
```

6.20 Primes Upto 1e9

```
// credit: min_25
// takes 0.5s for n = 1e9
vector<int> sieve(const int N, const
    int Q = 17, const int L = 1 << 15)
     {
  static const int rs[] = {1, 7, 11,
      13, 17, 19, 23, 29};
  struct P {
    P(int p) : p(p) {}
    int p; int pos[8];
  };
  auto approx_prime_count = [] (const
      int N) -> int {
    return N > 60184 ? N / (log(N) -
        1.1)
                     : max(1., N / (
                         log(N) -
                         1.11)) + 1;
  };

  const int v = sqrt(N), vv = sqrt(v);
  vector<bool> isp(v + 1, true);
  for (int i = 2; i <= vv; ++i) if (
      isp[i]) {
    for (int j = i * i; j <= v; j += i
        ) isp[j] = false;
  }

  const int rsize = approx_prime_count
      (N + 30);
  vector<int> primes = {2, 3, 5}; int
      psize = 3;
  primes.resize(rsize);

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

```
  }

  vector<unsigned char> pre(prod, 0xFF
      );
  for (size_t pi = 0; pi < pbeg; ++pi)
       {
    auto pp = sprimes[pi]; const int p
        = pp.p;
    for (int t = 0; t < 8; ++t) {
      const unsigned char m = ~(1 << t
          );
      for (int i = pp.pos[t]; i < prod
          ; i += p) pre[i] &= m;
    }
  }

  const int block_size = (L + prod -
      1) / prod * prod;
  vector<unsigned char> block(
      block_size); unsigned char*
      pblock = block.data();
  const int M = (N + 29) / 30;

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

6.21 Extd GCD

```
// return {x,y} such that ax+by=gcd(a,
    b)
pair<int, int>
ext_gcd(int a, int b){
  if (b == 0) return {1, 0};
  else{
    pair<int, int> tmp = ext_gcd(b, a
        % b);
    return {tmp.second,tmp.first - (a
        / b) * tmp.second};
  }
}
```

6.22 Factorial Mod

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

6.23 Modular Operations

```
// Addition:
int mod_add(int a, int b, int MOD =
    mod){
    a = a % MOD, b = b % MOD;
    return (((a + b) % MOD) + MOD) %
        MOD;
}
// Subtraction:
int mod_sub(int a, int b, int MOD =
    mod){
    a = a % MOD, b = b % MOD;
    return (((a - b) % MOD) + MOD) %
        MOD;
}
// Multiplication:
int mod_mul(int a, int b, int MOD =
    mod){
    a = a % MOD, b = b % MOD;
    return (((a * b) % MOD) + MOD) %
        MOD;
}
// Division:
//call binary Exponential Function
    here.
int mminvprime(int a, int b) { return
    binaryExp(a, b - 2, b); }
//call modular multiplication here.
int mod_div(int a, int b, int MOD =
    mod) {
    a = a % MOD, b = b % MOD;
    return (mod_mul(a, mminvprime(b,
        MOD), MOD) + MOD) % MOD;
}
//only for prime MOD
```

6.24 10 base to M base conversion

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

6.25 M base to 10 base conversion

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

6.26 Matrix Exponentiation

```
#define vvi vector<vector<ll>>
ll n, m;
vvi matixMulti(vvi &a, vvi &b) {
    vvi res(n, vector<ll>(n, 0));
    for (ll i = 0; i < n; i++) {
        for (ll j = 0; j < n; j++) {
            for (ll k = 0; k < n; k++)
                {
                res[i][j] = (res[i][j]
                    + (a[i][k] * b[k
                    ][j]) % mod) % mod
                    ;
            }
        }
    }
    return res;
}

vvi martixExp(vvi &base, ll power) {
    vvi identity(n, vector<ll>(n, 0));
    for (ll i = 0; i < n; i++)
        identity[i][i] = 1;

    while (power > 0) {
        if (power % 2) {
            identity = matixMulti(base
                , identity);
        }
        base = matixMulti(base, base);
        power /= 2;
    }
    return identity;
}
```

7 Dynamic Programming

7.1 LCS (Longest Common Subsequence)

```
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 (Longest Increasing Subsequence)

```
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";
}
```

7.3 SOS (Sum Of Subsets)

```
void SOS (vector<int> &v) {
  const int BITS = log2(*max_element(v
      .begin(), v.end())) + 1;
  vector<int> freq(1 << BITS, 0);
  for (int mask : v) freq[mask]++;

  vector<vector<int>> dp(BITS + 1,
      vector<int> (1 << BITS, 0));
  for (int mask = 0; mask < 1 << BITS;
       mask++) {
    dp[0][mask] = freq[mask];
  }

  for (int bits = 1; bits <= BITS;
      bits++) {
    for (int mask = 0; mask < 1 <<
        BITS; mask++) {
      if ((mask & (1 << (bits - 1)))
          == 0) {
        dp[bits][mask] = dp[bits - 1][
            mask];
      }
      else {
        int other_mask = mask - (1 <<
            (bits - 1));
        dp[bits][mask] = dp[bits - 1][
            mask] + dp[bits - 1][
            other_mask];
      }
    }
  }

  for (int mask : v) cout << dp[BITS][
      mask] << '\n';
}
// dp[bits][mask] means left most '
    bits' of submasks can be differ
// for dp[1][11] 01, 00 are now allow
    because leftmost 1 bit can be
    differ. 10 and 11 are allowed.
// for travarsing all the submask of a
     mask we can use
// submask = mask
// do {
// submask = (submask - 1) & mask
// } while (submask)
```

8 Strings

8.1 Double Hashing

Need power() from **??**

```
const int N = 1e6 + 9;
const int MOD1 = 127657753, MOD2 =
    987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void prec() {
  pw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    pw[i].first = 1LL * pw[i - 1].
        first * p1 % MOD1;
    pw[i].second = 1LL * pw[i - 1].
        second * p2 % MOD2;
  }
  ip1 = power(p1, MOD1 - 2, MOD1);
  ip2 = power(p2, MOD2 - 2, MOD2);
  ipw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    ipw[i].first = 1LL * ipw[i - 1].
        first * ip1 % MOD1;
    ipw[i].second = 1LL * ipw[i - 1].
        second * ip2 % MOD2;
  }

}
struct Hashing {
  int n;
  string s; // 0 - indexed
  vector<pair<int, int>> hs; // 1 -
      indexed
  Hashing() {}
  Hashing(string _s) {
    n = _s.size();
    s = _s;
    hs.emplace_back(0, 0);
    for (int i = 0; i < n; i++) {
      pair<int, int> p;
      p.first = (hs[i].first + 1LL *
          pw[i].first * s[i] % MOD1) %
           MOD1;
      p.second = (hs[i].second + 1LL *
          pw[i].second * s[i] % MOD2)
          % MOD2;
      hs.push_back(p);
    }
  }
  pair<int, int> get_hash(int l, int r
      ) { // 1 - indexed
    assert(1 <= l && l <= r && r <= n)
        ;
    pair<int, int> ans;
    ans.first = (hs[r].first - hs[l -
        1].first + MOD1) * 1LL * ipw[l
         - 1].first % MOD1;
    ans.second = (hs[r].second - hs[l
        - 1].second + MOD2) * 1LL *
        ipw[l - 1].second % MOD2;
    return ans;
```

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

8.2 Large Number Multiplication

```
#include <bits/stdc++.h>
using namespace std;
// Multiplies str1 and str2, and
    prints result.
string multiply(string num1, string
    num2){
    int len1 = num1.size();
    int len2 = num2.size();
    if (len1 == 0 || len2 == 0)
        return "0";
    // will keep the result number in
        vector
    // in reverse order
    vector<int> result(len1 + len2, 0)
        ;
    // Below two indexes are used to
        find positions
    // in result.
    int i_n1 = 0;
    int i_n2 = 0;
    // Go from right to left in num1
    for (int i = len1 - 1; i >= 0; i
        --){
        int carry = 0;
        int n1 = num1[i] - '0';
        // To shift position to left
            after every
        // multiplication of a digit
            in num2
        i_n2 = 0;
        // Go from right to left in
            num2
        for (int j = len2 - 1; j >= 0;
            j--){
            // Take current digit of
                second number
            int n2 = num2[j] - '0';
            // Multiply with current
                digit of first number
            // and add result to
                previously stored
                result
            // at current position.
            int sum = n1 * n2 + result
                [i_n1 + i_n2] + carry;
            // Carry for next
                iteration
            carry = sum / 10;
            // Store result
            result[i_n1 + i_n2] = sum
                % 10;
            i_n2++;
        }
        // store carry in next cell
        if (carry > 0)
            result[i_n1 + i_n2] +=
                carry;
        // To shift position to left
            after every
        // multiplication of a digit
            in num1.
        i_n1++;
    }
    // ignore '0's from the right
    int i = result.size() - 1;
    while (i >= 0 && result[i] == 0)
        i--;
    // If all were '0's - means either
        both or
    // one of num1 or num2 were '0'
    if (i == -1)
        return "0";
    // generate the result string
    string s = "";
    while (i >= 0)
        s += std::to_string(result[i
            --]);
    return s;
}
// Driver code
int main(){
    string str1 = "
        12354214154545454545454544";
    string str2 = "
        17145465465465454544548544544545
        ";
    cin >> str1 >> str2;
    if ((str1.at(0) == '-' || str2.at
        (0) == '-') &&
        (str1.at(0) != '-' || str2.at
            (0) != '-'))
        cout << "-";
    if (str1.at(0) == '-')
        str1 = str1.substr(1);
    if (str2.at(0) == '-')
        str2 = str2.substr(1);
    cout << multiply(str1, str2);
    return 0;
}
```

8.3 Large Number Summation

```
#include <bits/stdc++.h>
using namespace std;
// Function for finding sum of larger
    numbers
string findSum(string str1, string
    str2){
    // Before proceeding further, make
        sure length
    // of str2 is larger.
    if (str1.length() > str2.length())
        swap(str1, str2);
    // Take an empty string for
        storing result
    string str = "";
    // Calculate length of both string
```

```
    int n1 = str1.length(), n2 = str2.
        length();
    int diff = n2 - n1;
    // Initially take carry zero
    int carry = 0;
    // Traverse from end of both
        strings
    for (int i = n1 - 1; i >= 0; i--){
        // Do school mathematics,
            compute sum of
        // current digits and carry
        int sum = ((str1[i] - '0') + (
            str2[i + diff] - '0') +
            carry);
        str.push_back(sum % 10 + '0');
        carry = sum / 10;}
    // Add remaining digits of str2[]
    for (int i = n2 - n1 - 1; i >= 0;
        i--){
        int sum = ((str2[i] - '0') +
            carry);
        str.push_back(sum % 10 + '0');
        carry = sum / 10;}
    // Add remaining carry
    if (carry)
        str.push_back(carry + '0');
    // reverse resultant string
    reverse(str.begin(), str.end());
    return str;}
// Driver code
int main(){
    string str1 = "12";
    string str2 = "198111";
    cin >> str1 >> str2;
    cout << findSum(str1, str2);
    return 0;}
```

8.4 Large Number Division

```
#include <bits/stdc++.h>
using namespace std;
// A function to perform division of
    large numbers
string longDivision(string number, int
     divisor){
    // As result can be very large
        store it in string
    string ans;
    // Find prefix of number that is
        larger
    // than divisor.
    int idx = 0;
    int temp = number[idx] - '0';
    while (temp < divisor)
        temp = temp * 10 + (number[++
            idx] - '0');

    // Repeatedly divide divisor with
        temp. After
    // every division, update temp to
        include one
    // more digit.
    while (number.size() > idx) {
        // Store result in answer i.e.
            temp / divisor
        ans += (temp / divisor) + '0';

        // Take next digit of number
        temp = (temp % divisor) * 10 +
            number[++idx] - '0';}

    // If divisor is greater than
        number
    if (ans.length() == 0)
        return "0";
    // else return ans
    return ans;}
// Driver program to test longDivision
    ()
int main(){
    string number = "
        12481632641282565 12";
    int divisor = 125;
    cout << longDivision(number,
        divisor);
    return 0;}
```

8.5 Large Number Subtraction

```
#include <bits/stdc++.h>
using namespace std;
// Returns true if str1 is smaller
    than str2,
// else false.
bool isSmaller(string str1, string
    str2){
    // Calculate lengths of both
        string
    int n1 = str1.length(), n2 = str2.
        length();
    if (n1 < n2)
        return true;
    if (n2 < n1)
        return false;
    for (int i = 0; i < n1; i++) {
        if (str1[i] < str2[i])
            return true;
        else if (str1[i] > str2[i])
            return false;}
    return false;}
// Function for finding difference of
    larger numbers
string findDiff(string str1, string
    str2){
    // Before proceeding further, make
         sure str1
    // is not smaller
    if (isSmaller(str1, str2))
        swap(str1, str2);
    // Take an empty string for
        storing result
    string str = "";
    // Calculate lengths of both
        string
    int n1 = str1.length(), n2 = str2.
        length();
    int diff = n1 - n2;
```

```
    // Initially take carry zero
    int carry = 0;
    // Traverse from end of both
        strings
    for (int i = n2 - 1; i >= 0; i--)
        {
        // Do school mathematics,
            compute difference of
        // current digits and carry
        int sub = ((str1[i + diff] - '
            0') - (str2[i] - '0')
                    - carry);
        if (sub < 0) {
            sub = sub + 10;
            carry = 1;}
        else
            carry = 0;
        str.push_back(sub + '0');}
    // subtract remaining digits of
        str1[]
    for (int i = n1 - n2 - 1; i >= 0;
        i--) {
        if (str1[i] == '0' && carry) {
            str.push_back('9');
            continue;}
        int sub = ((str1[i] - '0') -
            carry);
        if (i > 0 || sub > 0) //
            remove preceding 0's
            str.push_back(sub + '0');
        carry = 0;}
    // reverse resultant string
    reverse(str.begin(), str.end());
    return str;}
// Driver code
int main(){
    string str1 = "88";
    string str2 = "1079";
    // Function call
    cout << findDiff(str1, str2);
    return 0;}
```

9 Stress Testing

9.1 Bash Stress File

```
#!/usr/bin/env bash

g++ -g "$1".cpp -DONPC -o "$1"
g++ -g "$2".cpp -DONPC -o "$2"
g++ -g "$3".cpp -DONPC -o "$3"

for ((testNum=0;testNum<$4;testNum++))
do
    ./$3 > stdinput
    ./$2 < stdinput > outSlow
    ./$1 < stdinput > outWrong
    H1=`md5sum outWrong`
    H2=`md5sum outSlow`
    if !(cmp -s "outWrong" "outSlow")
    then
        echo "Error found!"
        echo "Input:"
        cat stdinput
        echo "Wrong Output:"
        cat outWrong
        echo "Slow Output:"
        cat outSlow
        exit
    fi
done
echo Passed $4 tests

# ./stress.sh wrong correct gen times
```

9.2 C++ Generator File

```
#include <bits/stdc++.h>
using namespace std;

#define i64 long long
#define accuracy chrono::steady_clock
    ::now().time_since_epoch().count()

mt19937 rng(accuracy);

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

int main() {
  srand(accuracy);
  int t = 1;
  t = rand(1, 10), cout << t << '\n';
  while (t--) {
    // TODO
  }
}
```

10 Game Theory

10.1 Nim Game

The current player has a winning strategy if and only if the xor-sum of the pile sizes is non-zero.

10.2 Miser Nim

-Last player to remove stones loses. -Winning state if xor-sum of pile sizes is non-zero. -Exception: Each pile has one stone only. -Winning strategy: If there is only one pile of size greater than one,take all or all but one from that pile leaving an odd number one-size piles. Otherwise, same as normal nim.

10.3 Grundy Game

```
// The starting configuration is a
    single heap of objects. The two
    players take turn splitting a
    single heap into two heaps of
    different sizes. The player who
```

```
    can't make a move loses./ In each
    turn, a player can pick any pile
    and divide it into two unequal
    piles.
// If a player cannot do so, he/she
    loses the game.
int mex(vector<int> v) {
  sort(v.begin(), v.end());
  int ret = 0;
  for(int i=0; i<(int) v.size(); ++i)
      {
    if(v[i] == ret) ++ret;
    else if(v[i] > ret) break;
  }
  return ret;
}
const int N = 1e3 + 7;
int dp[N];
int g(int n) {
  if(n == 0) return 0;
  if(dp[n] != -1) return dp[n];

  vector<int> gsub;
  for(int i=1; i<n-i; ++i) {
    int cur = g(i) xor g(n-i);
    gsub.push_back(cur);
  }
  dp[n] = mex(gsub);
  return dp[n];
}
int main() {
  memset(dp, -1, sizeof dp);
int n;
  while(cin >> n) {
    if(g(n) > 0) cout << "First\n";
    else cout << "Second\n";
  }
}
```