

NoteBook

- Template

```
#include <bits/stdc++.h>
#define all( ) x.begin(),x.end()
#define ff first
#define ss second

using namespace std;

typedef long long ll;
typedef vector<ll> vi;
typedef pair<ll,ll> pii;
typedef vector<pii> vii;
const ll INF = numeric_limits<ll>::max();

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    return 0;
}
```

Problem Solving Paradigms

Binary Search

- Implementation

```
ll binarySearch(vi array, ll value) {
    ll left = 0;
    ll right = array.size() - 1;
    ll middle;
    while (left <= right) {
        middle = (left + right) / 2;
        if (value == array[middle])
            return middle;
        if (value < array[middle])
            right = middle - 1;
        else
            left = middle + 1;
    }
}
```

```

    }
    return -1; // not found
}

```

- Lower bound

```

ll lowerBound(vi array, ll value) {
    ll left = 0;
    ll right = array.size(); // not n - 1;
    ll middle;
    while (left < right) {
        middle = (left + right) / 2;
        if (value <= array[middle])
            right = middle + 1;
        else
            left = middle;
    }
    return left;
}

```

Bit Manipulation

X	Y	X or Y	X & Y	X ^ Y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Applications

- Count the number of ones in the binary representation of the given number

```

ll numberBits(ll x) {
    ll count = 0;
    while (x) {
        x &= (x - 1); // To turn off of the rightmost bits
        count++;
    }
    return count;
}

```

- How to generate all subsets of a set

```
void subsets(vi array, ll n) {
    for (ll i = 0; i < (1 << n); i++) {
        for (ll j = 0; j < n; j++)
            if (i & (1 << j))
                cout << array[j] << ' ';
        cout << '\n';
    }
}
```

Trick with bits

- $x \ll 1$ is equal that $x * 2$
- $x \gg 1$ is equal that $x / 2$
- $x \gg 2$ is equal that $x / 4$
- $x | (1 \ll j)$ To turn on the j -th bit of the x
- $x \& (1 \ll j)$ To check if the j -th bit of x is on
- $x \& \sim(1 \ll j)$ To turn off the j -th bit of x
- $x \wedge (1 \ll j)$ To toggle (flip the status of) of the j -th bit of the x
- $x \& (-x)$ To get the value of the least significant bits (first of the right)
- $(1 \ll x) - 1$ To turn on *all* bits in a set of size x
- $x \&\& \sim(x \& (x - 1))$ To check if x is power of 2

Math

Number Theory

- Sieve of Eratosthenes

```
const ll MAX_N = 1e7;
bitset<MAX_N> sieve;
sieve.set(); // all bits in true
vi primes;
primes.reserve(MAX_N / log(MAX_N));
for (ll i = 2; i < MAX_N; i++)
    if (sieve[i]) {
        for (ll j = i * i; j < MAX_N; j += i)
            sieve[j] = false;
        primes.push_back(i);
    }
```

- Functions Involving Prime Factors

- Count the number of different prime factors of `n`

```
void numPF(ll n) {
    vi factors;
    for (ll pf: primes) {
        if (pf * pf > n)
            break;
        while (n % pf == 0) {
            n /= pf;
            factors.push_back(pf);
        }
    }

    if (n != 1)
        factors.push_back(pf);
}
```

- Count the number of divisor of `n`

```
ll numDiv(ll n) {
    ll power, answer = 1;
    for (ll pf: primes) {
        if (pf * pf > n)
            break;
        power = 0;
        while (n % pf == 0) {
            n /= pf;
            power++;
        }
        answer *= (power + 1);
    }

    if (n != 1)
        answer *= 2;
    return answer;
}
```

- Sum of divisors of `n`

```
ll numDiv(ll n) {
    ll power, answer = 1;
    for (ll pf: primes) {
        if (pf * pf > n)
            break;
        power = 0;
```

```

        while (n % pf == 0) {
            n /= pf;
            power++;
        }
        answer *= (pow(pf, power + 1) - 1) / (pf - 1);
    }

    if (n != 1)
        answer *= (pow(pf, 2) - 1) / (n - 1);
    return answer;
}

```

- Count the number of positive integers $< n$ that are relatively prime to n .

```

ll eulerPhi(ll n) {
    ll answer;
    for (ll pf: primes) {
        if (pf * pf > n)
            break;
        if (n % pf == 0)
            answer -= answer / pf;

        while (n % pf == 0)
            n /= pf;
    }

    if (n != 1)
        answer -= answer / n;
}

```

- Pollard's rho Integer Factoring Algorithm find a divisor of n . This is used for factoring integers with 64 bits. Pollard's rho can factor an integer n if n is a large prime or is one.

```

ll mulmod(ll a, ll b, ll c) { // return (a * b) % c
    ll x = 0, y = a % c;
    while (b > 0) {
        if (b % 2 == 1) // odd
            x = (x + y) % c;
        y = (y * 2) % c;
        b /= 2;
    }
    return x % c;
}

ll pollardRho(ll n) {
    ll i = 0, k = 2;
    ll x = 3, y = 3;
    while (true) {
        i++;

```

```

x = (mulmod(x, x, n) + n - 1) % n;
ll d = gcd(abs(y - x), n);
if (d != 1 && d != n)
    return d;
if (i == k) {
    y = x;
    k *= 2;
}
}
}

```

- Modified Sieve (DP)

- Sieve of Eratosthenes

```

vi numDiffPF(MAX_N);
for (ll i = 0; i < MAX_N; i++)
    if (numDiffPF[i])
        for (ll j = i; j < MAX_N; j += i)
            numDiffPF[j]++;

```

- Euler Totient

```

for (ll i = 0; i < MAX_N; i++)
    eulerPhi[i] = i;
for (ll i = 2; i < MAX_N; i++)
    if (eulerPhi[i] == i) // i is a prime
        for (ll j = i; j < MAX_N; j += i)
            eulerPhi[j] = (eulerPhi[j] / i) * (i - 1);

```

- Extended Euclid: Solving Linear Diophantine Equation

```

void extendedEuclid(ll a, ll b, ll &x, ll &y, ll &d) {
    if (b == 0) {
        x = 1;
        y = 0;
        d = a;
        return;
    }
    extendedEuclid(b, a % b, x, y, d);
    ll x1 = y;
    ll y1 = x - (a / b) * y;
    x = x1;
    y = y1;
}

```

- Modulo Arithmetic

- $(a + b) \% c = ((a \% c) + (b \% c)) \% c$
- $(a * b) \% c = ((a \% c) * (b \% c)) \% c$
- $(a - b) \% c = ((a \% c) - (b \% c)) \% c$
- $(a / b) \% c = ((a \% c) / (b \% c)) \% c$
- $(a ^ b) \% c = (2 * (a ^ {\lfloor b / 2 \rfloor} \% c)) \% c$ b is even
- $(a / b) \% c = ((a \% c) * (b^{-1} \% c)) \% c$
- $(x + y - z) \% c = ((a \% c) + (b \% c) - (z \% c) + c) \% c$

- Greatest Common Divisor and Least Common Multiple

```
11 gcd(11 a, 11 b) {  
    return b == 0 ? a : gcd(b, a % b);  
}  
  
11 lcd(11 a, 11 b) {  
    return (a * b) / gcd(a, b);  
}
```

Data Structures

- Union-Find Disjoint Sets

```
class UnionFind {  
private:  
    vi parent;  
    vi heighth;  
  
public:  
    UnionFind (11 n) {  
        parent = vi(n);  
        heighth = vi(n, 1);  
        for (11 i = 0; i < n; i++)  
            parent[i] = i;  
    }  
  
    11 findSet(11 p) {  
        11 root = p, aux;  
        while (root != parent[root])  
            root = parent[root];  
        while (p != root) {  
            aux = parent[p];  
            parent[p] = root;  
            p = aux;  
        }  
    }
```

```

        return root;
    }

    bool isSameSet(ll p, ll q) {
        return findSet(p) == findSet(q);
    }

    void unionSet(ll p, ll q) {
        ll rootP = findSet(p);
        ll rootQ = findSet(q);
        if (rootP == rootQ)
            return;
        if (heighth[rootP] < heighth[rootQ])
            parent[rootP] = rootQ;
        else {
            parent[rootQ] = rootP;
            if (heighth[rootP] == heighth[rootQ])
                heighth[rootP]++;
        }
    }
};

```

- Segment Tree
- Binary Indexed (Fenwick) Tree

Graph

- **Depth first search:** Return the number of component conected. It is important fill the `color` with "white"
 - Recursive

```

ll dfs (vector<vi> graph, vector<string> &color, vector<ll> &path, ll node) {
    ll totalMarquet = 1;
    color[node] = "gray";
    for (auto neighbor : graph[node])
        if (color[neighbor] == "white") {
            path[neighbor] = node;
            totalMarquet += dfs(graph, color, path, neighbor);
        }

    color[node] = "black";
    return totalMarquet;
}

```

- Iterative


```

11 dfs (vector<vi> graph, vector<string> &color, vector<ll> &path, ll initNode) {

    ll node, totalMarquet = 1;
    stack<ll> nodeList;
    nodeList.push(initNode);
    color[node] = "gray";

    while (!nodeList.empty()) {
        node = nodeList.top();
        nodeList.pop();
        for (auto neighbor: graph[node])
            if (color[neighbor] == "white") {
                nodeList.push(neighbor);
                path[neighbor] = node;
                color[neighbor] = "gray";
                totalMarquet++;
            }
        color[node] = "black";
    }
    return totalMarquet;
}

```

- Build Path

```

11 longPath(vector<ll> path, ll endNode) {
    ll answer = 0;
    ll currentNode = endNode;
    while (path[currentNode] != -1) {
        cout << currentNode << ' ';
        currentNode = path[currentNode];
        answer++;
    }
    cout << currentNode << '\n';
    return answer;
}

```

- For test

```

// int main
ll n, l, r, initNode;
cin >> n >> initNode;
vector<vi> graph(n, vi());
vector<string> color(n, "white");
vector<ll> path(n);
path[initNode] = -1;
while (n) {
    n--;
}

```

```

    cin >> l >> r;
    graph[l].push_back(r);
}

cout << dfs(graph, color, path, initNode) << '\n';
for (ll i = 0; i < (ll)path.size(); i++)
    cout << i << ' ' << path[i] << '\n';

cout << '\n';
cout << longPath(path, 4) << '\n';

```

- test case

```

7 0
0 1
1 4
3 4
0 3
0 2
5 6

```

- **Breath first search:**