Java\Bitwise\Bit Manipulations.java

```
Operaciones a nivel de bits.

n & 1      -> Verifica si n es impar o no
n & (1<<k) -> Verifica si el k-esimo bit esta encendido o no
n | (1<<k) -> Enciende el k-esimo bit
n & ~(1<<k)     -> Apaga el k-esimo bit
n ^ (1<<k) -> Invierte el k-esimo bit
~n         -> Invierte todos los bits
n & -n          -> Devuelve el bit encendido mas a la derecha
~n & (n+1) -> Devuelve el bit apagado mas a la derecha
n | (n+1)  -> Enciende el bit apagado mas a la derecha
n & (n-1)  -> Apaga el bit encendido mas a la derecha
```

```
//  O(n · 2^n )

    static List<List<Integer>> generateSubsets(int[] nums) {
        List<List<Integer>> subsets = new ArrayList<>();
        int n = nums.length;
        // Total number of subsets = 2^n

        // Generate all subsets
        for (int i = 0; i < 1 << n; i++) {
            List<Integer> subset = new ArrayList<>();
            for (int j = 0; j < n; j++) {
                if ((i & (1 << j)) != 0) {
                    subset.add(nums[j]);
                }
            }
            subsets.add(subset);
        }
        return subsets;
    }

    static void solve() throws IOException {
        int arr[] = { 1, 2, 3 };
        List<List<Integer>> x = generateSubsets(arr);
        for (List<Integer> i : x) {
            sa.println(i);
        }
        /*
         * []
         * [1]
         * [2]
         * [1, 2]
         * [3]
         * [1, 3]
         * [2, 3]
         * [1, 2, 3]
         */
    }
//
```

Java\Data Structures\MultiSet.java

```
static class MultiSet<T> {

        HashMap<T, Integer> fre;
        TreeSet<T> set;
        int size;

        public MultiSet() {
            set = new TreeSet<>();
            fre = new HashMap<>();
            size = 0;
        }

        public void add(T elem) {
            if (fre.get(elem) == null || fre.get(elem) == 0) {
                fre.put(elem, 0);
                set.add(elem);
            }
            fre.put(elem, fre.get(elem) + 1);
            size++;
        }

        public void remove(T elem) {
            fre.put(elem, fre.get(elem) - 1);
            if (fre.get(elem) == 0) {
                set.remove(elem);
            }
            size--;
        }

        public boolean contains(T elem) {
            return set.contains(elem);
        }
    }
```

```
static class SegmentTree<T> {
    int n;
    ArrayList<T> st;
    T neutro;
    BinaryOperator<T> oper;

    SegmentTree(ArrayList<T> a, BinaryOperator<T> op, T neutro) {
        n = a.size();
        st = new ArrayList<>(Collections.nCopies(n * 4, null));
        this.neutro = neutro;
        oper = op;

        build(1, 0, n - 1, a);
    }

    void build(int v, int tl, int tr, ArrayList<T> a) {
        if (tl == tr) {
            st.set(v, a.get(tl));
            return;
        }
        int tm = (tr + tl) / 2;
        build(v * 2, tl, tm, a);
        build(v * 2 + 1, tm + 1, tr, a);
        st.set(v, oper.apply(st.get(v * 2), st.get(v * 2 + 1)));
    }

    T query(int v, int tl, int tr, int l, int r) {
        if (tl > r || tr < l)
            return neutro;
        if (l <= tl && tr <= r)
            return st.get(v);
        int tm = (tl + tr) / 2;
        return oper.apply(query(v * 2, tl, tm, l, r), query(v * 2 + 1, tm + 1, tr, l, r));
    }

    void upd(int v, int tl, int tr, int pos, T val) {
        if (tl == tr) {
            st.set(v, val);
            return;
        }
        int tm = (tr + tl) / 2;
        if (pos <= tm)
            upd(v * 2, tl, tm, pos, val);
        else
            upd(v * 2 + 1, tm + 1, tr, pos, val);
        st.set(v, oper.apply(st.get(v * 2), st.get(v * 2 + 1)));
    }

    void upd(int pos, T val) {
        upd(1, 0, n - 1, pos, val);
    }
    // l inclusive  r exclusive
    T query(int l, int r) {
        return query(1, 0, n - 1, l, r);
    }
}


 //Example

 /*
 ArrayList<Long> arr = new ArrayList<>();
 BinaryOperator<Long> oper = (a, b) -> Math.min(a, b);
 Long neutro = Long.valueOf(1000000007);
 SegmentTree<Long> st = new SegmentTree<Long>(arr, oper, neutro);
 Long QUERY = st.query(l, r);
 System.out.println(QUERY == null ? 0 : QUERY); -> validar query que no sea null
 */
 // llenar sparse table O(n long n)

    static class Sparse_Table {
        int n, K;

        int st[][];

        int oper(int a, int b) {
            return a + b;
        }

        public Sparse_Table(List<Integer> a) {
            this.n = a.size();
            this.K = (int) (log(n) / log(2) + 1);
            st = new int[n + 1][K];

            for (int i = 0; i < n; i++)
                st[i][0] = a.get(i);

            for (int j = 0; j < K - 1; j++) {
                for (int i = 0; i + (1 << (j + 1)) <= n; ++i) {
                    st[i][j + 1] = oper(st[i][j], st[i + (1 << j)][j]);
                }

            }
        }

        long sum(int l, int r) {
            long sum = 0;
            for (int j = K; j >= 0; j--) {
                if ((1 << j) <= r - l + 1) {
                    sum += st[l][j];
                    l += (1 << j);
                }
            }
            return sum;
        }

        int query(int l, int r) {
            int k = 31 - Integer.numberOfLeadingZeros(r - l + 1);
            return oper(st[l][k], st[r - (1 << k) + 1][k]);
        }
    }
```

```
static final int N = 1000;

    static int dp[][] = new int[N][N];

    static int n;
    static int w[] = new int[N];
    static int f[] = new int[N];

    static void solve() throws IOException {

        n = en.nextInt();
        for (int i = 0; i < n; i++) w[i] = en.nextInt();
        for (int i = 0; i < n; i++) f[i] = en.nextInt();
        int m = en.nextInt();
        int ans = 0;
        for (int i = n - 1; i >= 0; i--) {
            for (int p = 0; p <= m; p++) {
                dp[i][p] = dp[i + 1][p];
                if (p >= w[i]) {
                    dp[i][p] = max(ans, dp[i + 1][p - w[i]] + f[i]);
                }
            }
        }
        sa.write(dp[0][m] + "\n");

    }
```

```
static void bfs(ArrayList<Edge> graph[], boolean vis[], int start) {
        StringBuilder path = new StringBuilder();

        Queue<Integer> q = new LinkedList<>();
        q.add(start);

        while (!q.isEmpty()) {
            int cur = q.remove();
            if (vis[cur] == false) {
                path.append(cur).append(' ');
                vis[cur] = true;
                for (int i = 0; i < graph[cur].size(); i++) {
                    Edge e = graph[cur].get(i);
                    q.add(e.dest);
                }
            }
        }
        System.out.println(path);
    }


    static void dfs(ArrayList<Edge> graph[], int cur, boolean vis[]) {
        System.out.print(cur+" ");
        vis[cur] = true;
        for (int i = 0; i < graph[cur].size(); i++) {
            Edge e = graph[cur].get(i);
            if (!vis[e.dest]) {
                dfs(graph, e.dest, vis);
            }
        }
    }


    static void solve() throws IOException {
        int vertex = 7;
        ArrayList<Edge> graph[] = new ArrayList[vertex];
        for (int i = 0; i < graph.length; i++) {
            graph[i] = new ArrayList<Edge>();
        }

        graph[0].add(new Edge(0, 1));
        graph[0].add(new Edge(0, 2));
        graph[1].add(new Edge(1, 0));
        graph[1].add(new Edge(1, 3));
        graph[2].add(new Edge(2, 0));
        graph[2].add(new Edge(2, 4));
        graph[3].add(new Edge(3, 1));
        graph[3].add(new Edge(3, 4));
        graph[3].add(new Edge(3, 5));
        graph[4].add(new Edge(4, 2));
        graph[4].add(new Edge(4, 3));
        graph[4].add(new Edge(4, 5));
        graph[5].add(new Edge(5, 3));
        graph[5].add(new Edge(5, 4));
        graph[5].add(new Edge(5, 6));
        graph[6].add(new Edge(6, 5));

        boolean vis[] = new boolean[vertex];
        /*for (int i = 0; i < vertex; i++) {
            if (!vis[i]) {
                bfs(graph, vis, i);
            }
        }*/
        for (int i = 0; i < vertex; i++) {
            if(!vis[i]){
                dfs(graph, i, vis);
            }
        }

    }
```

Java\Graph\bfs grid.java

```
public class BFS_matriz {



    /*
     *    https://cses.fi/problemset/task/1193/
     *    un mapa en forma de matriz, llegar desde A hasta B
     */


    static final int MAX_N = 1001;
    static boolean vis[][] = new boolean[MAX_N][MAX_N];
    static char ar[][] = new char[MAX_N][MAX_N];
    static char br[][] = new char[MAX_N][MAX_N];
    static LinkedList<Character> path = new LinkedList<>();
    static int n, m;

    static boolean isValid(int x, int y) {
        if (x < 1 || x > n || y < 1 || y > m) return false;
        if (ar[x][y] == '#' || vis[x][y]==true) return false;
        return true;
    }

    static boolean bfs(int x, int y) {
        LinkedList<pair<Integer>> q = new LinkedList<>();
        q.add(new pair<Integer>(x, y));
        vis[x][y] = true;

        while (!q.isEmpty()) {
            int a = q.getFirst().first;
            int b = q.getFirst().second;
            q.removeFirst();
            if(ar[a][b] == 'B'){
                while(true){
                    path.add(br[a][b]);
                    if(path.getLast() == 'L') b++;
                    if(path.getLast() == 'R') b--;
                    if(path.getLast() == 'U') a++;
                    if(path.getLast() == 'D') a--;

                    if(a==x && b==y) break;
                }
                return true;
            }
            //left
™   if(isValid(a , b - 1)){
                br[a][b-1] = 'L';
                q.add(new pair<Integer>(a, b-1));
                vis[a][b-1] = true;
            }
™
™   //right
™   if(isValid(a , b + 1)){
                br[a][b+1] = 'R';
                q.add(new pair<Integer>(a, b+1));
                vis[a][b+1] = true;
            }
™
™   //up
™   if(isValid(a - 1, b)) {
                br[a - 1][b] = 'U';
                q.add(new pair<Integer>(a-1, b));
                vis[a-1][b] = true;
            }

™
™   //down
™   if(isValid(a + 1, b)){
                br[a + 1][b] = 'D';
                q.add(new pair<Integer>(a+1, b));
                vis[a+1][b] = true;
            }

        }
        return false;

    }

    static void solve() throws IOException {
        n = en.nextInt();
        m = en.nextInt();

        int x= 0, y= 0;
        for (int i = 1; i <= n; i++) {
            String line = en.next();
            for (int j = 1; j <= m; j++) {
                ar[i][j] =line.charAt(j-1);
                if(ar[i][j]=='A') {
                    x=i;
                    y=j;
                }
            }
        }

        if(bfs(x, y)){
            sa.println("YES\n"+path.size());
            StringBuilder ans = new StringBuilder();
            while(!path.isEmpty()) ans.append(path.removeLast());
            sa.println(ans);
        }else sa.println("NO");

    }

     static class pair<T> {
        T first;
        T second;
        public pair(T f, T s) {
            first = f;
            second = s;
        }
    }


}
```

Java\Graph\Dijkstra.java

```java
import java.util.*;

public class Dijkstra {

    static class Pair {
        int first;
        int second;

        Pair(int first, int second) {
            this.first = first;
            this.second = second;
        }

        @Override
        public String toString() {
            return "(" + first + " , " + second + ")";
        }
    }

    static final int MAXN = 100003;
    static final int INF = Integer.MAX_VALUE;

    static List<List<Pair>> adjList;
    static int n;
    static int[] dist = new int[MAXN];
    static int[] par = new int[MAXN];
    static BitSet isDone = new BitSet(MAXN);

    // dijkstra(int source, int des)
    static boolean dijkstra(int s, int t) {
        PriorityQueue<Pair> pq = new PriorityQueue<>(Comparator.comparingInt(p -> p.first));
        Arrays.fill(dist, INF);

        pq.add(new Pair(0, s));
        dist[s] = 0;
        par[s] = -1;

        while (!pq.isEmpty()) {
            int u = pq.poll().second;

            if (u == t)
                return true;

            isDone.set(u, true);

            for (Pair pr : adjList.get(u)) {
                int v = pr.first;
                int w = pr.second;

                if (!isDone.get(v) && dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                    pq.add(new Pair(dist[v], v));
                    par[v] = u;
                }
            }
        }

        return false;
    }


    /*      Dijkstra
     *      shortest path undirected graph
     *      https://codeforces.com/contest/20/problem/C
     */

    static void solve() throws IOException {
        n = en.nextInt();
        int m = en.nextInt();

        adjList = new ArrayList<>(n + 3);
        for (int i = 0; i < n + 3; i++) {
            adjList.add(new ArrayList<>());
        }

        dist = new int[MAXN];
        par = new int[MAXN];
        isDone = new BitSet(MAXN);

        int u, v, w;
        for (int i = 0; i < m; i++) {
            u = en.nextInt();
            v = en.nextInt();
            w = en.nextInt();
            adjList.get(u).add(new Pair(v, w));
            adjList.get(v).add(new Pair(u, w));
        }
        //path is found
        if (dijkstra(1, n)) {
            StringBuilder ans = new StringBuilder();
            List<Integer> path = new ArrayList<>();
            for (v = n; v != -1; v = par[v]) {
                path.add(v);
            }
            //reversing path
            for (int i = path.size()-1; i >=0; i--) {
                ans.append(path.get(i)).append(" ");
            }
            sa.println(ans);
        } else {
            sa.println("-1");
        }
    }
}
```

```
/*
     * primero llamar a sieve() y
     * luego a factorize()
     */

    static final long MAX = 1000000; // 10^6

    static long[] primediv = new long[(int) MAX];
    static List<Long> primes = new ArrayList<>();

    static void sieve() {
        for (int i = 0; i < MAX; i++)
            primediv[i] = i;
        int root = (int) sqrt(MAX) + 1;
        for (int i = 2; i < MAX; i++) {
            if (primediv[i] != i)
                continue;
            primes.add((long) i);
            if (i > root)
                continue;
            for (int j = i * i; j < MAX; j += i)
                primediv[j] = i;
        }
    }

    static Map<Long, Integer> factorize(long n) { // n <= 10^12
        Map<Long, Integer> factors = new HashMap<>();
        for (int i = 0; i < primes.size() && n >= MAX; ++i) {
            while (n % primes.get(i) == 0) {
                factors.put(primes.get(i), factors.getOrDefault(primes.get(i), 0) + 1);
                n /= primes.get(i);
            }
        }
        if (n >= MAX) {
            factors.put(n, factors.getOrDefault(n, 0) + 1);
            return factors;
        }
        while (n > 1) {
            factors.put(primediv[(int) n], factors.getOrDefault(primediv[(int) n], 0) + 1);
            n /= primediv[(int) n];
        }
        return factors;
    }

    public static void main(String[]args) throws IOException {
™
™
        sieve();

        Map<Long, Integer> ans = factorize(2500);

        for (Map.Entry<Long, Integer> entry : ans.entrySet()) {
            long prime = entry.getKey();
            int count  =  entry.getValue();
            for (int j = 0; j < count; j++) {
                System.out.println(prime);
            }
        }
    }
```

Java\Math\Generate permutation.java

```
// O(n!)

static void generatePermutation(ArrayList<Integer> arr, int n) {
        if (n == 1) {
            printArr(arr);
        } else {
            for (int i = 0; i < n; i++) {
                generatePermutation(arr, n - 1);
                if (n % 2 == 0) {
                    swap(arr, i, n - 1);
                } else {
                    swap(arr, 0, n - 1);
                }
            }
        }
    }

    static void swap(ArrayList<Integer> arr, int i, int j) {
        int temp = arr.get(i);
        arr.set(i, arr.get(j));
        arr.set(j, temp);
    }

    static void printArr(ArrayList<Integer> arr) {
        for (int num : arr) {
            sa.print(num + " ");
        }
        sa.println();
    }
```

Java\Math\Ocurrencias de multiplos.java

```java
import static java.lang.Math.*;
import java.util.*;
import java.io.*;

static void ocurrencia_Multiplos(int nums[]) {
        int n = nums.length;
        int ocurrences[] = new int[n + 1];
        HashMap<Integer, Integer> mp = new HashMap<>();

        for (int i = 0; i < n; i++) {
            if (nums[i] <= n) {
                mp.put(nums[i], mp.getOrDefault(nums[i], 0) + 1);
            }
        }

        for (int i = 1; i <= n; i++) {
            if (mp.getOrDefault(i, 0) == 0) {
                continue;
            }
            for (int j = i; j <= n; j += i) {
                ocurrences[j] += mp.getOrDefault(i, 0);
            }
        }

        /* multiplo mas repetido y menos repetido */

        long MAX = 0;
        long MIN = Long.MAX_VALUE;

        for (int i : ocurrences) {
            MAX =  max(MAX,  i);
            MIN = min(MIN, i);
        }
        sa.println(MAX);
    }
```

Java\Math\Sieve of Eratosthenes.java

```
/* Guarda en primes los numeros primos menores o iguales a MAX */


static final int MAX = 10000000;
static ArrayList<Integer> primes = new ArrayList<>();
static boolean sieve[] = new boolean[MAX + 5];

static void calculatePrimes() {
    sieve[0] = sieve[1] = true;
    int i;
    for (i = 2; i * i <= MAX; i++) {
        if (!sieve[i]) {
            primes.add(i);
            for (int j = i * i; j <= MAX; j += i)
                sieve[j] = true;
        }
    }
    for (; i <= MAX; i++) {
        if (!sieve[i]) {
            primes.add(i);
        }
    }
}
```

Java\String\Aho Corasick 1.java

```
//AhoCorasick


static class node {
        int child[];
        int p;
        char c;
        int suffixLink;
        int endLink;
        int id;

        node(int p, char c) {
            this.p = p;
            this.c = c;
            this.child = new int[26];
            Arrays.fill(child, -1);
            this.suffixLink = -1;
            this.endLink = -1;
            this.id = -1;
        }
    }

    static class AhoCorasick {
        ArrayList<node> t;
        ArrayList<Integer> lengths;
        int wordCount;

        AhoCorasick() {
            t = new ArrayList<>();
            t.add(new node(-1, (char) 0));
            lengths = new ArrayList<>();
            wordCount = 0;
        }

        void add(String s) {
            int u = 0;
            for (int i = 0; i < s.length(); i++) {
                char c = s.charAt(i);
                int idx = c - 'a';
                if (t.get(u).child[idx] == -1) {
                    t.get(u).child[idx] = t.size();
                    t.add(new node(u, c));
                }
                u = t.get(u).child[idx];
            }
            t.get(u).id = wordCount++;
            lengths.add(s.length());
        }

        void link(int u) {
            if (u == 0) {
                t.get(u).suffixLink = 0;
                t.get(u).endLink = 0;
                return;
            }
            if (t.get(u).p == 0) {
                t.get(u).suffixLink = 0;
                if (t.get(u).id != -1)
                    t.get(u).endLink = u;
                else
                    t.get(u).endLink = t.get(t.get(u).suffixLink).endLink;
                return;
            }
            int v = t.get(t.get(u).p).suffixLink;
            char c = t.get(u).c;
            while (true) {
                if (t.get(v).child[c - 'a'] != -1) {
                    t.get(u).suffixLink = t.get(v).child[c - 'a'];
                    break;
                }
                if (v == 0) {
                    t.get(u).suffixLink = 0;
                    break;
                }
                v = t.get(v).suffixLink;
            }
            if (t.get(u).id != -1)
                t.get(u).endLink = u;
            else
                t.get(u).endLink = t.get(t.get(u).suffixLink).endLink;
        }

        void build() {
            Queue<Integer> Q = new LinkedList<>();
            Q.add(0);
            while (!Q.isEmpty()) {
                int u = Q.poll();
                link(u);
                for (int v = 0; v < 26; v++) {
                    if (t.get(u).child[v] != -1) {
                        Q.add(t.get(u).child[v]);
                    }
                }
            }
        }

        int match(String text) {
            int u = 0;
            int ans = 0;
            for (int j = 0; j < text.length(); j++) {
                int i = text.charAt(j) - 'a';
                while (true) {
                    if (t.get(u).child[i] != -1) {
                        u = t.get(u).child[i];
                        break;
                    }
                    if (u == 0)
                        break;
                    u = t.get(u).suffixLink;
                }
                int v = u;
                while (true) {
                    v = t.get(v).endLink;

                    if (v == 0) break;


                    ans++;

                    /*
                     ====================================================================
                     * Found word #" + t.get(v).id + " at position " + idx
                     *
                     * palabra encontrada, crear String words[] y almacenar durante la lectura
                     * String found = words[t.get(v).id];
                     *
                     =====================================================================
                     * Si queremos almacenar los indices en los que aparece (inicio, fin)
                     * se puede usar un ArrayList[] o un HasMap<String, List<pair<Integer,
Integer>>>
                     * int idx = j + 1 - lengths.get(t.get(v).id);
                     * int inicio = idx - found.length() + 1;
                     * int fin = idx;
                     * mp[t.get(v).id].add(new pair<Integer, Integer>(inicio, fin));
                     =====================================================================
                     *
                     */

                    v = t.get(v).suffixLink;
                }
            }
            return ans;
        }
    }
```

Java\String\Aho Corasick 2.java

```
// Aho_corasick implementacion corta

   //para poder imprimir las palabras
   //que van apareciendo -> crear ArrayList<String> p;


   static class Aho_Corasick {

       final int N = 200000;
       int trie[][] = new int[N][256];
       int nodes = 0;
       int fail[] = new int[N];
       int end_word[] = new int[N];

       //almacena los indices en los que aparece una cadena (inicio, fin)

       LinkedHashMap<String, ArrayList<pair<Integer, Integer>>> mp = new LinkedHashMap<>();

       void add(String s, int idx) {
           int node = 0;
           for (int i = 0; i < s.length(); i++) {
               char ch = s.charAt(i);
               if (trie[node][ch] == 0) {
                   trie[node][ch] = ++nodes;
               }
               node = trie[node][ch];
           }
           end_word[node] = idx;
       }

       void build() {
           Queue<Integer> q = new LinkedList<>();
           for (int ch = 0; ch < 256; ch++) {
               if (trie[0][ch] > 0) {
                   q.add(trie[0][ch]);
               }
           }
           while (!q.isEmpty()) {
               int u = q.poll();
               for (int ch = 0; ch < 256; ch++) {
                   int v = trie[u][ch];
                   if (v != 0) {
                       fail[v] = fail[u];
                       while (fail[v] > 0 && trie[fail[v]][ch] == 0) {
                           fail[v] = fail[fail[v]];
                       }
                       fail[v] = trie[fail[v]][ch];
                       q.add(trie[u][ch]);
                   }
               }
           }
       }

       int count(String text) {
           int node = 0;
           int ans = 0;
           for (int i = 0; i < text.length(); i++) {
               char ch = text.charAt(i);
               while (node > 0 && trie[node][ch] == 0) {
                   node = fail[node];
               }
               node = trie[node][ch];
               int tmp_node = node;
               while (tmp_node > 0) {
                   if (end_word[tmp_node] > 0) {
                       ans++;
                       //String que aparece en text
                       String s = p.get(end_word[tmp_node] - 1);
                       //(inicio, fin)
                       int start = i - s.length() + 1;
                       int end = i;
                       mp.get(s).add(new pair<>(start, end));
```

```
                    // sa.print(p.get(word[tmp_node]-1)+"\n");

                }
                tmp_node = fail[tmp_node];
            }
        }
        return ans;
    }

}
```

Java\String\KMP.java

```java
 // usar prefix function
// cuenta las veces que p aparece en s

    static int kmp(String s, String p) {
        int n = s.length(), m = p.length(), cnt = 0;
        int pf[] = prefix_function(p);
        for (int i = 0, j = 0; i < n; i++) {
            while (j > 0 && s.charAt(i) != p.charAt(j)) j = pf[j - 1];
            if (s.charAt(i) == p.charAt(j)) j++;
            if (j == m) {
                cnt++;
                j = pf[j - 1];
            }
        }
        return cnt;

    }
```

Java\String\Prefix function.java

```java
static int[] prefix_function(String s) {
    int n = s.length();
    int pf[] = new int[n];
    pf[0] = 0;
    for (int i = 1, j = 0; i < n; i++) {
        while (j > 0 && s.charAt(i) != s.charAt(j)) j = pf[j - 1];
        if (s.charAt(i) == s.charAt(j))
            j++;
        pf[i] = j;
    }
    return pf;
}
```

```
/* letter frecuency in a range */

static void PrefixSumArray(String s) {
       int psa[][] = new int[26][s.length() + 1];
       for (int i = 1; i <= s.length(); i++) {
           for (int j = 0; j < 26; j++) {
               psa[j][i] = psa[j][i - 1];
           }
           char cur = s.charAt(i - 1);
           psa[cur - 'a'][i]++;
       }
       char c = 'e';
       int left = 1, right = s.length();

       sa.println(psa[c - 'a'][right] - psa[c - 'a'][left - 1]);
   }
```

```
//retorna los indices en los que empieza una ocurrencia

//los indices van desde 1


static List<Integer> rabin_karp(String pattern, String text) {
    int p = 31;
    int m = (int) 1e9 + 9;
    int S = pattern.length(), T = text.length();

    long p_pow[] = new long[max(S, T)];
    p_pow[0] = 1;
    for (int i = 1; i < (int) p_pow.length; i++) {
        p_pow[i] = (p_pow[i - 1] * p) % m;
    }

    long h[] = new long[T + 1];
    for (int i = 0; i < T; i++) {
        h[i + 1] = (h[i] + (text.charAt(i) - 'a' + 1) * p_pow[i]) % m;
    }
    long h_s = 0;
    for (int i = 0; i < S; i++) {
        h_s = (h_s + (pattern.charAt(i) - 'a' + 1) * p_pow[i]) % m;
    }

    List<Integer> occurences = new ArrayList<>();
    for (int i = 0; i + S - 1 < T; i++) {
        long cur_h = (h[i + S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m){
            occurences.add(i);
        }
    }
    return occurences;
}
```

Java\String\String matching with Bitset.java

```
// Muy lento
//String matching using a bitset



    static final int N =(int) 1e6+1;
    static BitSet[] mask = new BitSet[26];
    static String text;
    static String pattern;
    static BitSet startMask = new BitSet(N);

    static void computeMask(String text) {
        text = "#" + text; // to make text 1-indexed.
        for (int i = 0; i < 26; i++) {
            mask[i] = new BitSet(N);
        }
        for (int i = 1; i < text.length(); ++i) {
            int c = text.charAt(i) - 'a';
            mask[c].set(i);
        }
    }

    static int match(String pattern, String text) {
        if (pattern.length() > text.length()) {
            return 0;
        }
        pattern = "#" + pattern;
        startMask.set(0, N, true); // all bits are set to true
        for (int i = 1; i < pattern.length(); ++i) {
            int c = pattern.charAt(i) - 'a';
            startMask.and(mask[c].get(i, N));
        }
        return startMask.cardinality(); // cardinality() returns the number of true bits

    }


    static int rangePatternMatchCount(String pattern, int l, int r) {
        if (r - l + 1 < pattern.length()) {
            return 0;
        }
        pattern = "#" + pattern;
        startMask.set(0, N, true);
        for (int i = 1; i < pattern.length(); ++i) {
            int c = pattern.charAt(i) - 'a';
            startMask.and(mask[c].get(i, N));
        }
        return startMask.get(l - 1, r - pattern.length() + 2).cardinality();
    }

    static List<Integer> rangePatternPositions(String pattern, int l, int r) {
        int rangeCount = rangePatternMatchCount(pattern, l, r);
        List<Integer> positions = new ArrayList<>();
        for (int i = l - 1, idx = 0; i < r - pattern.length() + 2 && idx < rangeCount; ++i) {
            if (startMask.get(i)) {
                positions.add(i + 1);
                ++idx;
            }
        }
        return positions;
    }

    static void setBitForChar(int bit, char c) {
        mask[c - 'a'].set(bit);
    }

    static void unsetBitForChar(int bit, char c) {
        mask[c - 'a'].clear(bit);
    }

    static void update(int idx, char ch) {
        unsetBitForChar(idx, text.charAt(idx));
        text = text.substring(0, idx) + ch + text.substring(idx + 1);
        setBitForChar(idx, text.charAt(idx));
    }
```

Java\String\Suffix Array.java

```
//primero llamar a init()
//luego match()


static class suffixArray {

        int K = 26; // tamaño del alfabeto
        int SA[];
        int occ[][];
        int count[];
        int n;

        void init(String s) {
            s += "$";
            n = s.length();
            SA = computeSuffixArray(s);
            occ = new int[n + 1][K];
            count = new int[K + 1];
            for (int i = 0; i < n; i++) {
                int t = s.charAt(SA[i] == 0 ? n - 1 : SA[i] - 1) - 'a';
                if (t >= 0) {
                    occ[i][t]++;
                    count[t + 1]++;
                }
                System.arraycopy(occ[i], 0, occ[i + 1], 0, K);
            }
            for (int i = 1; i < count.length; i++) {
                count[i] += count[i - 1];
            }
        }

        // Retorna indices (start, end) -> de cada aparicion de w en s

        // Si no se necesita un minimo de ocurrencias colocar minOcc = 0


        int[] match(String w, int minOcc) {
            int begin = 0, end = n - 1;
            // List<pair<Integer, Integer>> matches = new ArrayList<>();


            for (int j = w.length() - 1; end - begin + 1 >= minOcc && j >= 0; j--) {
                char c = w.charAt(j);
                int let = c - 'a';
                int nbegin = count[let] + (begin == 0 ? 0 : occ[begin - 1][let]) + 1;
                int nend = count[let] + occ[end][let];
                begin = nbegin;
                end = nend;
            }
            if (end - begin + 1 < minOcc) {
                return new int[] { -1 };
            }

            int t[] = new int[end - begin + 1];
            // se agregan los rangos en que aparece w en s

            for (int j = begin; j <= end; j++) {
                // int inicio = SA[j];
                // int fin = SA[j] + w.length() - 1;
                // matches.add(new pair<>(inicio, fin));
                t[j - begin] = SA[j];
            }
            // ordenar en caso de ser necesario o recorrer en reversa la lista


            Arrays.sort(t);
            return t;

        }

        int[] computeSuffixArray(CharSequence S) {
            int n = S.length();
            int[] sa = IntStream.range(0, n).mapToObj(i -> n - 1 - i)
                    .sorted((a, b) -> Character.compare(S.charAt(a),
S.charAt(b))).mapToInt(Integer::intValue)
                    .toArray();
            int[] classes = S.chars().toArray();
            for (int len = 1; len < n; len *= 2) {
                int[] c = classes.clone();
                for (int i = 0; i < n; i++) {
                    classes[sa[i]] = i > 0 && c[sa[i - 1]] == c[sa[i]] && sa[i - 1] + len < n
                            && c[sa[i - 1] + len / 2] == c[sa[i] + len / 2] ? classes[sa[i - 1
]] : i;
                }
                int[] cnt = IntStream.range(0, n).toArray();
                int[] s = sa.clone();
                for (int i = 0; i < n; i++) {
                    int s1 = s[i] - len;
                    if (s1 >= 0)
                        sa[cnt[classes[s1]]++] = s1;
                }
            }
            return sa;
        }

    }
```

```
static class Node {
        int cont;
        Node child[];

        public Node() {
            this.cont = 0;
            child = new Node[26];
        }
    }

    static class Trie {
        Node root;

        Trie() {
            root = new Node();
        }

        void insert(String s) {
            Node curr = root;
            for (int i = 0; i < s.length(); ++i) {
                char c = s.charAt(i);
                if (curr.child[c - 'a'] == null) {
                    curr.child[c - 'a'] = new Node();
                }
                curr.child[c - 'a'].cont++;
                curr = curr.child[c - 'a'];
            }
        }


        /*

            if (x.first == 0 && x.second == 0 || len < s.length()) {
                sa.write("NO ESTÁ\n");
            } else {
                sa.write("SI ESTÁ\n");
            }

         */
        pair<Integer, Integer> query(String s) {
            Node curr = root;
            for (int i = 0; i < s.length(); ++i) {
                char c = s.charAt(i);
                if (curr.child[c - 'a'] == null) {
                    return new pair<Integer, Integer>(i, curr.cont);
                }
                curr = curr.child[c - 'a'];
            }
            return new pair<Integer, Integer>(s.length(), curr.cont);
        }
```

Java\utils\InputReader.java

Mas rapido que el BufferedReader

```
   // en = new InputReader("input.txt");
   // sa = new BufferedWriter(new BufferedWriter(new FileWriter("output.txt")));


static class InputReader {

       InputStream stream;
       byte[] buf = new byte[1024];
       int curChar;
       int numChars;
       SpaceCharFilter filter;
       BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

       public InputReader() {
           this.stream = System.in;
       }

       public InputReader(String file) throws IOException {
           this.stream = new FileInputStream(file);
       }

       int read() {
           if (numChars == -1) {
               throw new InputMismatchException();
           }
           if (curChar >= numChars) {
               curChar = 0;
               try {
                   numChars = stream.read(buf);
               } catch (IOException e) {
                   throw new InputMismatchException();
               }

               if (numChars <= 0) {
                   return -1;
               }
           }
           return buf[curChar++];
       }

       String nextLine() {
           String str = "";
           try {
               str = br.readLine();
           } catch (IOException e) {
               e.printStackTrace();
           }
           return str;
       }

       int nextInt() {
           int c = read();
           while (isSpaceChar(c)) {
               c = read();
           }
           int sgn = 1;
           if (c == '-') {
               sgn = -1;
               c = read();
           }
           int res = 0;
           do {
               if (c < '0' || c > '9') {
                   throw new InputMismatchException();
               }
               res *= 10;
               res += c - '0';
               c = read();
           } while (!isSpaceChar(c));
           return res * sgn;
       }

       long nextLong() {
           int c = read();
           while (isSpaceChar(c)) {
               c = read();
           }
           int sgn = 1;
           if (c == '-') {
               sgn = -1;
               c = read();
           }
           long res = 0;
           do {
               if (c < '0' || c > '9') {
                   throw new InputMismatchException();
               }
               res *= 10;
               res += c - '0';
               c = read();
           } while (!isSpaceChar(c));
           return res * sgn;
       }

       double nextDouble() {
           int c = read();
           while (isSpaceChar(c)) {
               c = read();
           }
           int sgn = 1;
           if (c == '-') {
               sgn = -1;
               c = read();
           }
           double res = 0;
           while (!isSpaceChar(c) && c != '.') {
               if (c == 'e' || c == 'E') {
                   return res * Math.pow(10, nextInt());
               }
               if (c < '0' || c > '9') {
                   throw new InputMismatchException();
               }
               res *= 10;
               res += c - '0';
               c = read();
           }
           if (c == '.') {
               c = read();
               double m = 1;
               while (!isSpaceChar(c)) {
                   if (c == 'e' || c == 'E') {
                       return res * Math.pow(10, nextInt());
                   }
                   if (c < '0' || c > '9') {
                       throw new InputMismatchException();
                   }
                   m /= 10;
                   res += (c - '0') * m;
                   c = read();
               }
           }
           return res * sgn;
       }

       String next() {
           int c = read();
           while (isSpaceChar(c)) {
               c = read();
           }
           StringBuilder res = new StringBuilder();
           do {
               res.appendCodePoint(c);
               c = read();
           } while (!isSpaceChar(c));
           return res.toString();
       }
```

```
    boolean isSpaceChar(int c) {
        if (filter != null) {
            return filter.isSpaceChar(c);
        }
        return c == ' ' || c == '\n' || c == '\r' || c == '\t' || c == -1;
    }

    interface SpaceCharFilter {
        public boolean isSpaceChar(int ch);
    }

}
```

Java\utils\pair.java

```
//se puede usar en cualquier estructura

static class pair<F extends Comparable<F>, S extends Comparable<S>> implements Comparable
<pair<F, S>> {
        F first;
        S second;

        public pair(F first, S second) {
            this.first = first;
            this.second = second;
        }

        /*
         * • If object x is less than object y, return a negative number
         * • If object x is greater than object y, return a positive number
         * • If object x is equal to object y, return 0
         */
        @Override
        public int compareTo(pair<F, S> other) {
            // decresing order = return -first.compareTo(other.first);

            if (first == other.first) {
                return second.compareTo(other.second);
            } else {
                return first.compareTo(other.first);
            }
        }

        @Override
        public boolean equals(Object obj) {
            if (this == obj) {
                return true;
            }
            if (obj == null || getClass() != obj.getClass()) {
                return false;
            }
            pair<?, ?> other = (pair<?, ?>) obj;
            return Objects.equals(first, other.first) && Objects.equals(second, other.second);
        }

        @Override
        public int hashCode() {
            return 31 * first.hashCode() + second.hashCode();
        }

        @Override
        public String toString() {
            return "(" + first + ", " + second + ")";
        }
    }
```

Java\utils\Template.java

```java
import static java.lang.Math.*;
import java.util.*;
import java.io.*;

public class Template {

    static FastReader en = new FastReader();
    static BufferedWriter sa = new BufferedWriter(new OutputStreamWriter(System.out));

    static void solve() throws IOException {
        //en = new FastReader("input.txt");
        //sa = new BufferedWriter(new BufferedWriter(new FileWriter("output.txt")));




    }

    public static void main(String[] args) throws IOException {
        long t = 1;
        // t = en.nextLong();
        while (t-- > 0) {
            solve();
        }
        en.br.close();
        sa.close();
    }
    /*
     * number of digits = log10(x) + 1
     * number of bits = log2(x) + 1
     * number of times that we have to divide x by k = logk(x)
     */

    // ==== log(x) / log(base);

    static int gcd(int a, int b) {
        return b == 0 ? (a < 0 ? -a : a) : gcd(b, a % b);
    }

    static int lcm(int a, int b) {
        int lcm = (a / gcd(a, b)) * b;
        return lcm > 0 ? lcm : -lcm;
    }

    static void sort(int[] a) {
        Random get = new Random();
        for (int i = 0; i < a.length; i++) {
            int r = get.nextInt(a.length);
            int temp = a[i];
            a[i] = a[r];
            a[r] = temp;
        }
        Arrays.sort(a);
    }

    static class FastReader {
        BufferedReader br;
        StringTokenizer st;

        // Entrada standar
        public FastReader() {
            br = new BufferedReader(new InputStreamReader(System.in));
        }

        // Entrada por archivo
        public FastReader(String file) throws IOException {
            this.br = new BufferedReader(new FileReader(file));
        }

        String next() {
            while (st == null || !st.hasMoreElements()) {
                try {
                    st = new StringTokenizer(br.readLine());
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            return st.nextToken();
        }

        int nextInt() {
            return Integer.parseInt(next());
        }

        long nextLong() {
            return Long.parseLong(next());
        }

        double nextDouble() {
            return Double.parseDouble(next());
        }

        String nextLine() {
            String str = "";
            try {
                str = br.readLine();
            } catch (IOException e) {
                e.printStackTrace();
            }
            return str;
        }
    }

}
```

Java\utils\upper_bound and lower_bound.java

```
static int lower_bound(List<Long> a, long x) {
        int l = -1, r = a.size();
        while (l + 1 < r) {
            int m = (l + r) >>> 1;
            if (a.get(m) >= x)
                r = m;
            else
                l = m;
        }
        return r;
    }


    static int upper_bound(List<Long> a, long x) {
        int l = -1, r = a.size();
        while (l + 1 < r) {
            int m = (l + r) >>> 1;
            if (a.get(m) <= x)
                l = m;
            else
                r = m;
        }
        return l + 1;
    }
```