1. **Password validation**

import java.util.\*;

public class PasswordChecker {

public static List<String> checkPasswords(int n, String[] passwords, int k) {

Map<String, Integer> counts = new HashMap<>();

List<String> results = new ArrayList<>();

for (String password : passwords) {

if (!counts.containsKey(password)) {

counts.put(password, 1);

results.add("ACCEPT");

} else if (counts.get(password) < k) {

counts.put(password, counts.get(password) + 1);

results.add("ACCEPT");

} else {

results.add("REJECT");

}

}

return results;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

String[] passwords = new String[n];

for (int i = 0; i < n; i++) {

passwords[i] = sc.next();

}

int k = sc.nextInt();

List<String> results = checkPasswords(n, passwords, k);

for (String result : results) {

System.out.println(result);

}

}

}

**2.AutoCorrectType**

import java.util.\*;

public class Autocorrect {

public static List<List<String>> getSearchResults(List<String> words, List<String> queries) {

Map<String, List<String>> anagrams = new HashMap<>();

for (String word : words) {

char[] letters = word.toCharArray();

Arrays.sort(letters);

String sortedWord = new String(letters);

if (!anagrams.containsKey(sortedWord)) {

anagrams.put(sortedWord, new ArrayList<>());

}

anagrams.get(sortedWord).add(word);

}

List<List<String>> results = new ArrayList<>();

for (String query : queries) {

char[] letters = query.toCharArray();

Arrays.sort(letters);

String sortedQuery = new String(letters);

if (anagrams.containsKey(sortedQuery)) {

List<String> result = anagrams.get(sortedQuery);

Collections.sort(result);

results.add(result);

} else {

results.add(new ArrayList<>());

}

}

return results;

}

public static void main(String[] args) {

List<String> words = Arrays.asList("duel", "speed", "dule", "cars");

List<String> queries = Arrays.asList("spede", "deul");

List<List<String>> results = getSearchResults(words, queries);

System.out.println(results);

}

}

3.**prdouct Data Management**

import java.util.ArrayList;

import java.util.List;

public class ProductDataManager {

public static List<List<String>> getMatchingProducts(List<List<String>> products, List<List<String>> queries) {

List<List<String>> result = new ArrayList<>();

for (List<String> query : queries) {

String type = query.get(0);

String param = query.get(1);

List<String> matchingProducts = new ArrayList<>();

if (type.equals("Type1")) {

for (List<String> product : products) {

if (product.get(2).equals(param)) {

matchingProducts.add(product.get(0));

}

}

} else if (type.equals("Type2")) {

int price = Integer.parseInt(param);

for (List<String> product : products) {

int productPrice = Integer.parseInt(product.get(1));

if (productPrice < price) {

matchingProducts.add(product.get(0));

} else {

break;

}

}

} else if (type.equals("Type3")) {

int price = Integer.parseInt(param);

for (int i = products.size() - 1; i >= 0; i--) {

List<String> product = products.get(i);

int productPrice = Integer.parseInt(product.get(1));

if (productPrice > price) {

matchingProducts.add(0, product.get(0));

} else {

break;

}

}

}

result.add(matchingProducts);

}

return result;

}

}

**4.process Scheduler**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.PriorityQueue;

public class ProcessScheduler {

public static int getMinCores(List<Integer> start, List<Integer> end) {

int n = start.size();

List<int[]> processes = new ArrayList<>();

for (int i = 0; i < n; i++) {

processes.add(new int[] { start.get(i), end.get(i) });

}

Collections.sort(processes, (a, b) -> a[0] - b[0]);

PriorityQueue<Integer> heap = new PriorityQueue<>();

for (int i = 0; i < n; i++) {

int s = processes.get(i)[0];

int e = processes.get(i)[1];

if (!heap.isEmpty() && heap.peek() <= s) {

heap.poll();

}

heap.offer(e);

}

return heap.size();

}

public static void main(String[] args) {

List<Integer> start = Arrays.asList(1, 3, 4);

List<Integer> end = Arrays.asList(3, 5, 6);

System.out.println(getMinCores(start, end));

}

**6. even difference:**

public static int findLongestSubsequence(List<Integer> arr) {

int n = arr.size();

int maxLength = 0;

for (int i = 1; i < (1 << n); i++) {

List<Integer> subsequence = new ArrayList<>();

for (int j = 0; j < n; j++) {

if ((i & (1 << j)) > 0) {

subsequence.add(arr.get(j));

}

}

Collections.sort(subsequence);

int diffSum = 0;

for (int j = 1; j < subsequence.size(); j++) {

diffSum += subsequence.get(j) - subsequence.get(j-1);

}

if (diffSum % 2 == 0 && subsequence.size() > maxLength) {

maxLength = subsequence.size();

}

}

return maxLength;

}

public static void main(String[] args) {

List<Integer> arr = Arrays.asList(2, 4, 1, 7);

System.out.println(findLongestSubsequence(arr));

}

}

7.Faulty Server:

Faulty server detection(all test case passing)

import java.util.\*;

class Result {

/\*

\* Complete the 'countFaults' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER n

\* 2. STRING\_ARRAY logs

\*/

public static int countFaults(int n, List<String> logs) {

Map<String, Integer> errorCount = new HashMap<>();

int replacements = 0;

for (String log : logs) {

String[] parts = log.split("\\s+");

String serverId = parts[0];

String status = parts[1];

if (status.equals("error")) {

int count = errorCount.getOrDefault(serverId, 0);

count++;

errorCount.put(serverId, count);

if (count == 3) {

replacements++;

errorCount.put(serverId, 0);

}

} else {

errorCount.put(serverId, 0);

}

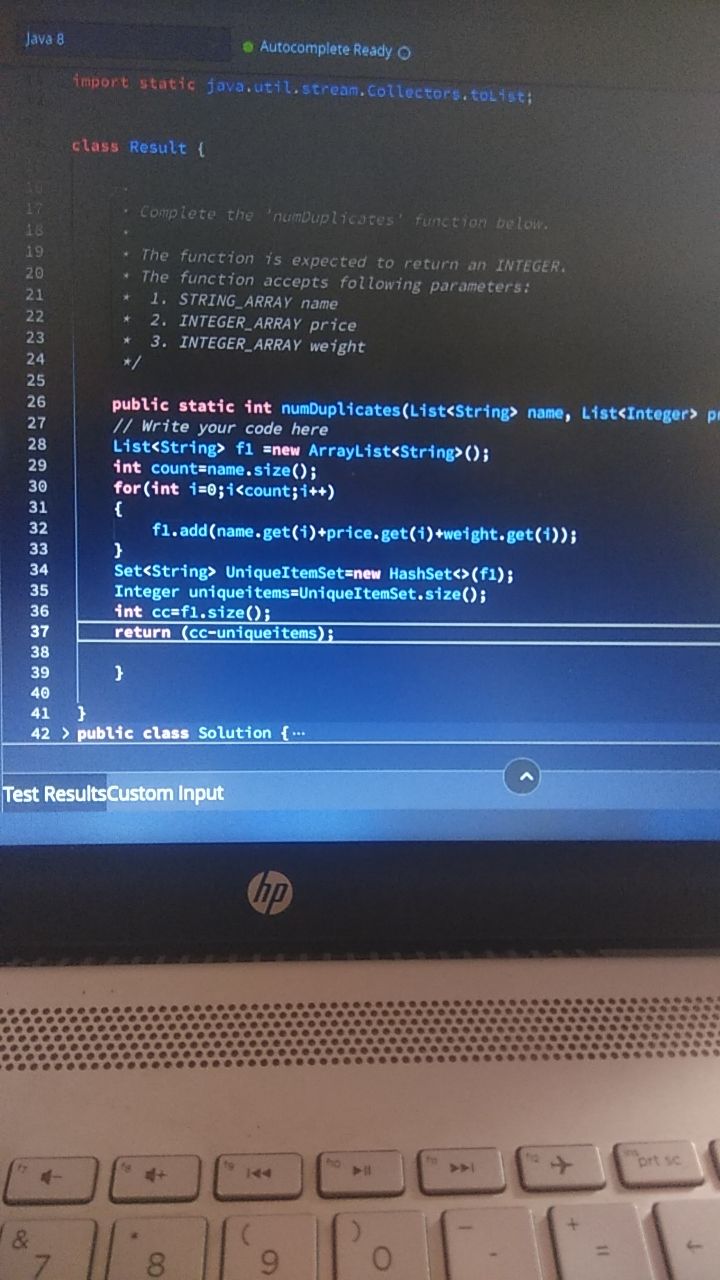
}

return replacements;

}

}

**9.Duplicate Products:**



**9.Password validation(another solution):**

public static List<String> getRegistrationStatus(List<String> passwords, int k) {

Map<String, Integer> passwordCount = new HashMap<>();

List<String> result = new ArrayList<>();

for (String password : passwords) {

int count = passwordCount.getOrDefault(password, 0);

if (count < k) {

passwordCount.put(password, count + 1);

result.add("ACCEPT");

} else {

result.add("REJECT");

}

}

return result;

}

**10.Application Type**

public class ApplicationManager {

public static List<String> getOpenApplications(List<String> commands) {

List<String> openApplications = new ArrayList<>();

for (String command : commands) {

if (command.startsWith("open")) {

String application = command.substring(5);

openApplications.add(application);

} else if (command.startsWith("close")) {

int k = Integer.parseInt(command.substring(6));

int size = openApplications.size();

int startIndex = Math.max(0, size - k);

openApplications.subList(startIndex, size).clear();

} else if (command.equals("clear")) {

openApplications.clear();

}

}

return openApplications;

}

}

**11.maximum Score:**

**public static long getMaximumScore(List<Integer> arr, int k) {**

**int n = arr.size();**

**List<Long> prefixSum = new ArrayList<>(n + 1);**

**prefixSum.add(0L);**

**for (int i = 1; i <= n; i++) {**

**prefixSum.add(prefixSum.get(i - 1) + arr.get(i - 1));**

**}**

**List<List<Long>> dp = new ArrayList<>(k + 1);**

**for (int i = 0; i <= k; i++) {**

**dp.add(new ArrayList<>(Collections.nCopies(n, 0L)));**

**}**

**for (int i = 1; i <= k; i++) {**

**for (int j = 0; j < n; j++) {**

**dp.get(i).set(j, prefixSum.get(n) - prefixSum.get(j));**

**for (int p = j + 1; p <= n - i; p++) {**

**dp.get(i).set(j, Math.max(dp.get(i).get(j), prefixSum.get(p) - prefixSum.get(j) + dp.get(i - 1).get(p)));**

**}**

**}**

**}**

**return dp.get(k).get(0);**

**}**