# Question1- superheroes

A team of superheroes is being assembled for a mission. Each superhero has a specific power, represented by the characters s, t, i, f, or x. Each team must consist of exactly one superhero skilled in each power.

Given a string powers, where each character represents a superhero's power, determine the maximum number of teams that can be formed. Each superhero can be part of only one team.

**Input Format:**

* A single string powers of length n (1 ≤ n ≤ 500,000), where each character belongs to the set {s, t, i, f, x}.

**Output Format:**

* Print a single integer denoting the maximum number of complete teams that can be formed.

**Example Input and Output:**

**Example 1:**

Input:

powers = "stifxstifx"

Output:

2

Explanation: The string contains two sets of all required powers. Two teams can be formed: stifx and stifx.

**Example 2:**

Input:

powers = "stifxstifxfxx"

Output:

2

Explanation: There are three superheroes with power x, but only two superheroes with power f. Hence, only two complete teams can be formed.

**Example 3:**

Input:

powers = "stix"

Output:

0

Explanation: Since at least one power is missing (f), no team can be formed.

# Question2 - The Perfect Orchestra

An orchestra needs musicians skilled in different instruments to perform a symphony. Each musician is skilled in exactly one instrument, represented by the characters v, d, p, g, or k. A complete orchestra must have one musician for each instrument.

Given a string skills, where each character represents a musician’s skill, determine the maximum number of orchestras that can be formed. Each musician can only play in one orchestra.

**Input Format:**

* A single string skills of length n (1 ≤ n ≤ 500,000), where each character belongs to the set {v, d, p, g, k}.

**Output Format:**

* Print a single integer denoting the maximum number of complete orchestras that can be formed.

**Example Input and Output:**

**Example 1:**

Input:

skills = "vdpgkvdpgk"

Output:

2

Explanation: The string contains two sets of all required skills. Two orchestras can be formed: vdpgk and vdpgk.

**Example 2:**

Input:

skills = "vdpgkvdpgkk"

Output:

2

Explanation: There are three musicians skilled in k, but only two musicians skilled in g. Hence, only two complete orchestras can be formed.

**Example 3:**

Input:

skills = "vdpg"

Output:

0

Explanation: Since at least one instrument is missing (k), no orchestra can be formed.

# Question3 – School Team

The School of Excellence is conducting a team-building activity. Each student is skilled in exactly one subject, represented by the characters p, c, m, b, or z.

Given a string skills where each character denotes a student's skill, your task is to determine the maximum number of complete teams that can be formed, following these rules:

1. A team consists of exactly 5 students, each skilled in one of the five subjects (p, c, m, b, z).
2. Each student can be part of only one team.
3. If any subject is missing or insufficient in number, fewer teams can be formed.

**Input Format:**

* A single string skills of length n (1 ≤ n ≤ 500,000) where each character belongs to the set {p, c, m, b, z}.

**Output Format:**

* Print a single integer denoting the maximum number of complete teams that can be formed.

**Example Input and Output:**

**Example 1:**

Input:

skills = "pcmbzpcmbz"

Output:

2

Explanation: The string has two sets of all required skills. Two teams can be formed: pcmbz and pcmbz.

**Example 2:**

Input:

skills = "pcmbzmpcmbzz"

Output:

1

Explanation: There are at least 2 students skilled in z but only one student in c. Therefore, only one complete team can be formed.

**Example 3:**

Input:

skills = "pcmp"

Output:

0

Explanation: Since at least one subject is missing (b, z), no team can be formed.

# Question 4 - Chair Simulation Problem

In a workroom, new employees require chairs to work. Initially, there are no chairs in the workroom. The following actions are possible:

1. C: A new employee arrives, and a new chair is purchased if no chair is available.
2. R: An employee leaves for a meeting, freeing up their chair.
3. U: An employee returns from a meeting and takes an available chair. If no chair is available, a new chair is purchased.
4. L: An employee leaves the workroom permanently, freeing up their chair.

Given a series of simulations, each represented as a string of these actions, determine the minimum number of chairs required for each simulation.

**Input Format:**

1. An integer n (1 ≤ n ≤ 100), the number of simulations.
2. An array of n strings, where each string represents one simulation and has a length of up to 10,000.

**Output Format:**

* An array of integers where each integer represents the minimum number of chairs required for the corresponding simulation.

**Example Input and Output:**

**Input:**

3

["CCRRL", "RUCLC", "CCCC"]

**Output:**

3

2

4

**Explanation:**

1. For the first simulation, "CCRRL":
   * Action flow:
     + C: Buy 1 chair (Total: 1, Available: 0).
     + C: Buy 1 chair (Total: 2, Available: 0).
     + R: Chair freed (Total: 2, Available: 1).
     + R: Chair freed (Total: 2, Available: 2).
     + L: Chair freed (Total: 2, Available: 1).
   * Minimum chairs required: 3.
2. For the second simulation, "RUCLC":
   * Minimum chairs required: 2.
3. For the third simulation, "CCCC":
   * Minimum chairs required: 4.

### Solution

import java.util.\*;

public class ChairSimulation {

public static int[] minChairs(String[] simulations) {

int n = simulations.length;

int[] results = new int[n];

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

String simulation = simulations[i];

int totalChairs = 0, availableChairs = 0;

for (char action : simulation.toCharArray()) {

if (action == 'C') {

if (availableChairs > 0) {

availableChairs--; // Use an available chair

} else {

totalChairs++; // Buy a new chair

}

} else if (action == 'R') {

availableChairs++; // Free up a chair

} else if (action == 'U') {

if (availableChairs > 0) {

availableChairs--; // Use an available chair

} else {

totalChairs++; // Buy a new chair

}

} else if (action == 'L') {

availableChairs++; // Free up a chair

}

}

results[i] = totalChairs;

}

return results;

}

public static void main(String[] args) {

String[] simulations = {"CCRRL", "RUCLC", "CCCC"};

int[] results = minChairs(simulations);

for (int res : results) {

System.out.println(res);

}

}

}

# Question 5 - Conference Room Allocation

A company is hosting multiple meetings in a single conference room. Participants arrive and leave based on the following actions:

* **A**: A new participant arrives in the room. If there is an available seat, they take it. Otherwise, a new seat is added.
* **L**: A participant leaves the room, freeing up their seat.

At the start of each simulation, there are no seats in the conference room. For each simulation, determine the **minimum number of seats required** to accommodate all participants during the meeting.

**Input Format:**

1. An integer n (1 ≤ n ≤ 100), the number of simulations.
2. An array of n strings, where each string describes a sequence of actions in a simulation (A and L).

**Output Format:**

* An array of integers, where each integer represents the **minimum number of seats required** for the corresponding simulation.

**Constraints:**

1. 1 ≤ n ≤ 100
2. 1 ≤ length of each simulation ≤ 10,000

**Example Input:**

3

["AALL", "AAAL", "AAAA"]

**Example Output:**

2

3

4

**Explanation:**

1. For the first simulation, "AALL":
   * Actions:
     + A: Add 1 seat (Total: 1, Available: 0).
     + A: Add 1 seat (Total: 2, Available: 0).
     + L: Free 1 seat (Total: 2, Available: 1).
     + L: Free 1 seat (Total: 2, Available: 2).
   * Minimum seats required: **2**.
2. For the second simulation, "AAAL":
   * Actions:
     + A: Add 1 seat (Total: 1, Available: 0).
     + A: Add 1 seat (Total: 2, Available: 0).
     + A: Add 1 seat (Total: 3, Available: 0).
     + L: Free 1 seat (Total: 3, Available: 1).
   * Minimum seats required: **3**.
3. For the third simulation, "AAAA":
   * Actions:
     + A: Add 1 seat (Total: 1, Available: 0).
     + A: Add 1 seat (Total: 2, Available: 0).
     + A: Add 1 seat (Total: 3, Available: 0).
     + A: Add 1 seat (Total: 4, Available: 0).
   * Minimum seats required: **4**.

# Question 6 - Maximum Attendance Streak

A project manager wants to analyze employee attendance data. Given that m employees are working on a project, and the manager has the attendance data for n days, determine the **maximum number of consecutive days** on which **all employees were present**.

* If an employee is present on a given day, the corresponding data entry is Y.
* If an employee is absent on a given day, the corresponding data entry is N.

**Input Format:**

1. An integer m (1 ≤ m ≤ 10), the number of employees working on the project.
2. An integer n (1 ≤ n ≤ 100,000), the number of days in the attendance data.
3. An array of n strings, where each string represents the attendance of all m employees on a specific day.

**Output Format:**

* A single integer denoting the **maximum streak** of consecutive days where **all employees were present**.

**Constraints:**

1. 1 ≤ m ≤ 10
2. 1 ≤ n ≤ 100,000
3. Each data[i][j] ∈ {Y, N}

**Example Input:**

3

5

["YYY", "YYY", "YNN", "YYN", "YYN"]

**Example Output:**

2

**Explanation:**

* Day 1 and Day 2: All employees are present.
* Day 3: Employees 2 and 3 are absent.
* Day 4 and Day 5: Employee 3 is absent.
* Maximum streak of consecutive days where all employees are present is **2 days**.

## Solution

import java.util.\*;

public class MaxAttendanceStreak {

public static int maxStreak(int m, String[] data) {

int maxStreak = 0, currentStreak = 0;

for (String day : data) {

boolean allPresent = true;

// Check if all employees are present on this day

for (char c : day.toCharArray()) {

if (c == 'N') {

allPresent = false;

break;

}

}

if (allPresent) {

currentStreak++;

maxStreak = Math.max(maxStreak, currentStreak);

} else {

currentStreak = 0;

}

}

return maxStreak;

}

public static void main(String[] args) {

int m = 3;

String[] data = {"YYY", "YYY", "YNN", "YYN", "YYN"};

System.out.println(maxStreak(m, data)); // Output: 2

}

}

# Question 7 – Problem Statement:

Rosalind has been given a string s consisting of lowercase English letters and the character ?. She is required to replace all ? with appropriate lowercase letters such that the resulting string becomes a **palindrome**. Among all possible palindromes, she needs to find the **lexicographically smallest one**. If no such palindrome can be formed, return -1.

**Notes:**

* A **palindrome** reads the same backward as forward, e.g., "abba" or "racecar".
* A string a is lexicographically smaller than string b if at the first different position, the character in a comes before the character in b in the dictionary order.

**Input Format:**

* A single line contains the string sss.

**Output Format:**

* Output the lexicographically smallest palindrome or -1 if it is not possible to form one.

**Example Input/Output:**

**Sample Input 1:**

a?rt???

**Sample Output 1:**

aarrtaa

**Explanation:** The question marks can be replaced with a, r, a, and a to form the palindrome "aarrtaa".

**Sample Input 2:**

yh??tx

**Sample Output 2:**

diff

-1

**Explanation:** It is not possible to replace ? to make the string a palindrome.

### Solution

public class SmallestPalindrome {

public static String getSmallestPalindrome(String s) {

char[] chars = s.toCharArray();

int n = chars.length;

for (int i = 0; i < n / 2; i++) {

int j = n - i - 1;

if (chars[i] == '?' && chars[j] == '?') {

chars[i] = chars[j] = 'a'; // Replace both '?' with 'a'

} else if (chars[i] == '?') {

chars[i] = chars[j]; // Mirror the character

} else if (chars[j] == '?') {

chars[j] = chars[i]; // Mirror the character

} else if (chars[i] != chars[j]) {

return "-1"; // Not possible to form a palindrome

}

}

// Handle the middle character for odd-length strings

if (n % 2 == 1 && chars[n / 2] == '?') {

chars[n / 2] = 'a';

}

return new String(chars);

}

public static void main(String[] args) {

// Test Cases

System.out.println(getSmallestPalindrome("a?rt???")); // Output: aarrtaa

System.out.println(getSmallestPalindrome("yh??tx")); // Output: -1

System.out.println(getSmallestPalindrome("???")); // Output: aaa

}

}