**DAA SKILL 6**

**Name: K. Sai Sadwiik**

**Id: 2300032291**

**Greedy Florist**

**Program:**

**1. import java.io.\*;**

**2. import java.math.\*;**

**3. import java.security.\*;**

**4. import java.text.\*;**

**5. import java.util.\*;**

**6. import java.util.concurrent.\*;**

**7. import java.util.regex.\*;**

**8.**

**9. public class Solution {**

**10.**

**11. // Complete the getMinimumCost function below.**

**12. static int getMinimumCost(int k, int[] c) {**

**13. Arrays.sort(c);**

**14. int totalCost = 0;**

**15. int numFlowers = c.length;**

**16.**

**17. int[] purchases = new int[k];**

**18.**

**19. for (int i=numFlowers-1;i>=0;i--) {**

**20.**

**21. int friendIndex = (numFlowers-1-i)%k;**

**22.**

**23. totalCost += (purchases[friendIndex]+1)\*c[i];**

**24.**

**25. purchases[friendIndex]++;**

**26. }**

**27.**

**28. return totalCost;**

**29.**

**30. }**

**31.**

**32. private static final Scanner scanner = new Scanner(System.in);**

**33.**

**34. public static void main(String[] args) throws IOException {**

**35. BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));**

**36.**

**37. String[] nk = scanner.nextLine().split(" ");**

**38.**

**39. int n = Integer.parseInt(nk[0]);**

**40.**

**41. int k = Integer.parseInt(nk[1]);**

**42.**

**43. int[] c = new int[n];**

**44.**

**45. String[] cItems = scanner.nextLine().split(" ");**

**46. scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");**

**47.**

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

**49. int cItem = Integer.parseInt(cItems[i]);**

**50. c[i] = cItem;**

**51. }**

**52.**

**53. int minimumCost = getMinimumCost(k, c);**

**54.**

**55. bufferedWriter.write(String.valueOf(minimumCost));**

**56. bufferedWriter.newLine();**

**57.**

**58. bufferedWriter.close();**

**59.**

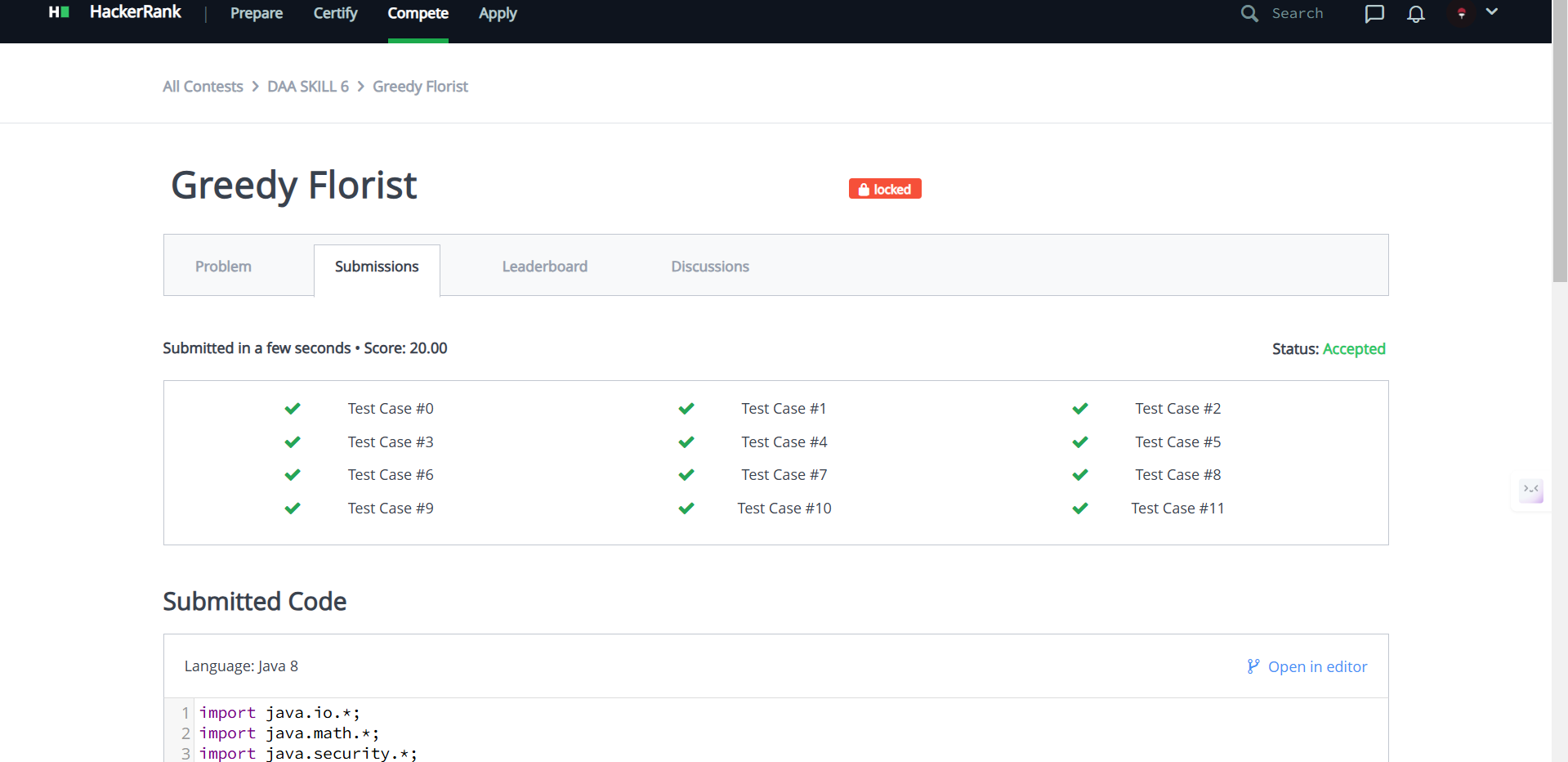
**60. scanner.close();**

**61. }**

**62. }**

**63.**

**TESTCASES:**

****

**Max Min**

**Program:**

**1. import java.io.\*;**

**2. import java.math.\*;**

**3. import java.security.\*;**

**4. import java.text.\*;**

**5. import java.util.\*;**

**6. import java.util.concurrent.\*;**

**7. import java.util.function.\*;**

**8. import java.util.regex.\*;**

**9. import java.util.stream.\*;**

**10. import static java.util.stream.Collectors.joining;**

**11. import static java.util.stream.Collectors.toList;**

**12.**

**13. class Result {**

**14.**

**15. /\***

**16. \* Complete the 'maxMin' function below.**

**17. \***

**18. \* The function is expected to return an INTEGER.**

**19. \* The function accepts following parameters:**

**20. \* 1. INTEGER k**

**21. \* 2. INTEGER\_ARRAY arr**

**22. \*/**

**23.**

**24. public static int maxMin(int k, List<Integer> arr) {**

**25. // Write your code here**

**26. Collections.sort(arr);**

**27.**

**28. int minUnfairness = Integer.MAX\_VALUE;**

**29.**

**30. for (int i=0;i<=arr.size()-k;i++) {**

**31. int currentUnfairness = arr.get(i+k-1)-arr.get(i);**

**32. minUnfairness = Math.min(minUnfairness,currentUnfairness);**

**33. }**

**34.**

**35. return minUnfairness;**

**36.**

**37. }**

**38.**

**39. }**

**40.**

**41. public class Solution {**

**42. public static void main(String[] args) throws IOException {**

**43. BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));**

**44. BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));**

**45.**

**46. int n = Integer.parseInt(bufferedReader.readLine().trim());**

**47.**

**48. int k = Integer.parseInt(bufferedReader.readLine().trim());**

**49.**

**50. List<Integer> arr = IntStream.range(0, n).mapToObj(i -> {**

**51. try {**

**52. return bufferedReader.readLine().replaceAll("\\s+$", "");**

**53. } catch (IOException ex) {**

**54. throw new RuntimeException(ex);**

**55. }**

**56. })**

**57. .map(String::trim)**

**58. .map(Integer::parseInt)**

**59. .collect(toList());**

**60.**

**61. int result = Result.maxMin(k, arr);**

**62.**

**63. bufferedWriter.write(String.valueOf(result));**

**64. bufferedWriter.newLine();**

**65.**

**66. bufferedReader.close();**

**67. bufferedWriter.close();**

**68. }**

**69. }**

**70.**

**TESTCASES:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Goodland Electricity**

**Program:**

**1. import java.io.\*;**

**2. import java.math.\*;**

**3. import java.security.\*;**

**4. import java.text.\*;**

**5. import java.util.\*;**

**6. import java.util.concurrent.\*;**

**7. import java.util.function.\*;**

**8. import java.util.regex.\*;**

**9. import java.util.stream.\*;**

**10. import static java.util.stream.Collectors.joining;**

**11. import static java.util.stream.Collectors.toList;**

**12.**

**13. class Result {**

**14.**

**15. /\***

**16. \* Complete the 'pylons' function below.**

**17. \***

**18. \* The function is expected to return an INTEGER.**

**19. \* The function accepts following parameters:**

**20. \* 1. INTEGER k**

**21. \* 2. INTEGER\_ARRAY arr**

**22. \*/**

**23.**

**24. public static int pylons(int k, List<Integer> arr) {**

**25. // Write your code here**

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

**27. int count = 0;**

**28. int i = 0;**

**29.**

**30. while (i < n) {**

**31.**

**32. int plantLocation = -1;**

**33.**

**34.**

**35. for (int j = Math.min(i + k - 1, n - 1); j >= Math.max(i - k + 1, 0); j--) {**

**36. if (arr.get(j) == 1) {**

**37. plantLocation = j;**

**38. break;**

**39. }**

**40. }**

**41.**

**42. if (plantLocation == -1) {**

**43. return -1;**

**44. }**

**45.**

**46.**

**47. count++;**

**48. i = plantLocation + k;**

**49. }**

**50.**

**51. return count;**

**52. }**

**53.**

**54. }**

**55.**

**56. public class Solution {**

**57. public static void main(String[] args) throws IOException {**

**58. BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));**

**59. BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));**

**60.**

**61. String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");**

**62.**

**63. int n = Integer.parseInt(firstMultipleInput[0]);**

**64.**

**65. int k = Integer.parseInt(firstMultipleInput[1]);**

**66.**

**67. List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))**

**68. .map(Integer::parseInt)**

**69. .collect(toList());**

**70.**

**71. int result = Result.pylons(k, arr);**

**72.**

**73. bufferedWriter.write(String.valueOf(result));**

**74. bufferedWriter.newLine();**

**75.**

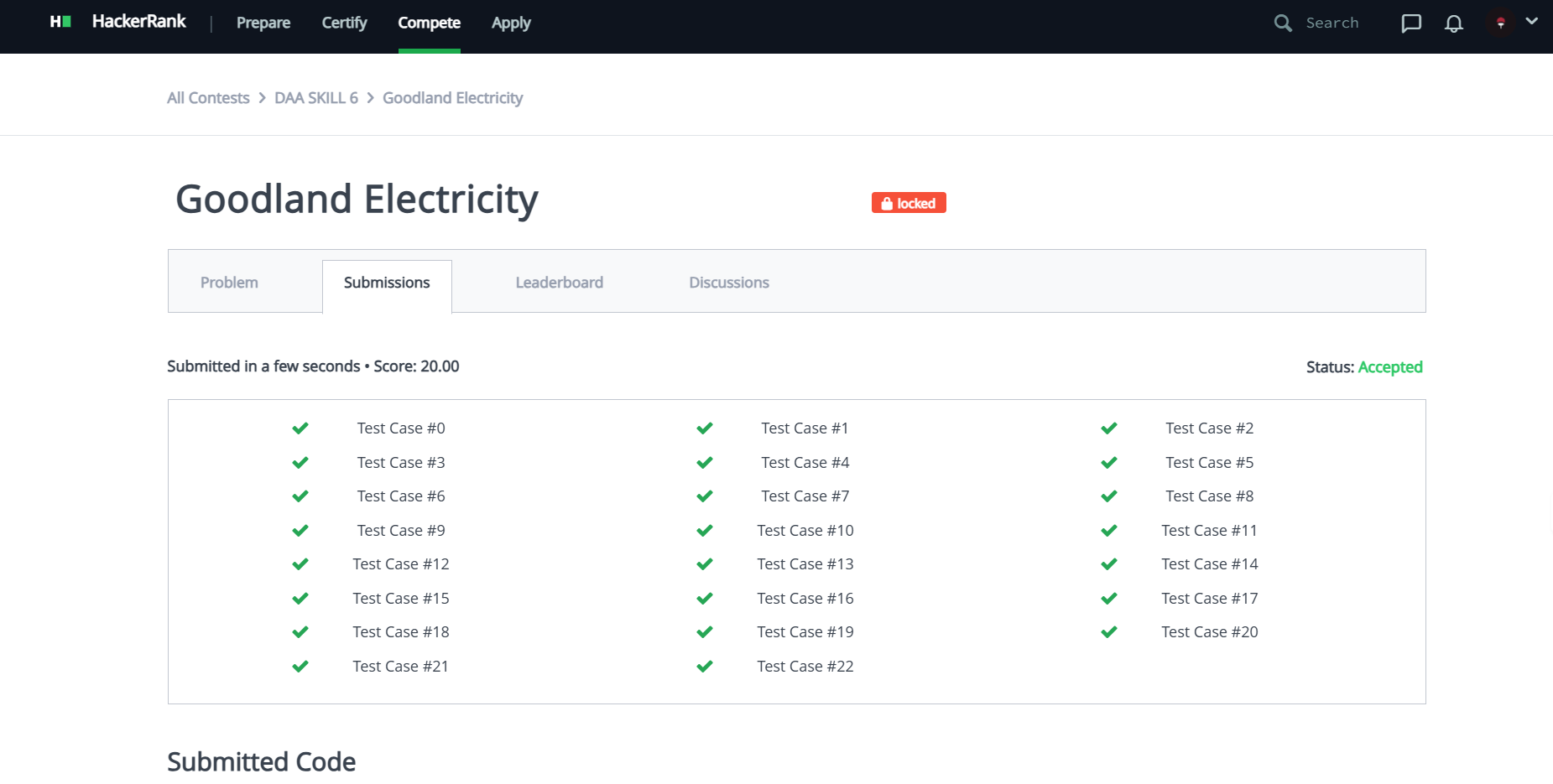
**76. bufferedReader.close();**

**77. bufferedWriter.close();**

**78. }**

**79. }**

**TESTCASES:**

****

**Cloudy Day**

**Program:**

**1. #!/bin/python3**

**2.**

**3. import math**

**4. import os**

**5. import random**

**6. import re**

**7. import sys**

**8.**

**9. #**

**10. # Complete the 'maximumPeople' function below.**

**11. #**

**12. # The function is expected to return a LONG\_INTEGER.**

**13. # The function accepts following parameters:**

**14. # 1. LONG\_INTEGER\_ARRAY p**

**15. # 2. LONG\_INTEGER\_ARRAY x**

**16. # 3. LONG\_INTEGER\_ARRAY y**

**17. # 4. LONG\_INTEGER\_ARRAY r**

**18. #**

**19.**

**20. def maximumPeople(p, x, y, r):**

**21.**

**22. # sort x and p by x**

**23. x, p = zip(\*sorted(zip(x, p)))**

**24.**

**25. not\_visited=[i for i in range(len(y))]**

**26.**

**27. # sort by starting point**

**28. not\_visited.sort(key=lambda x: y[x] - r[x])**

**29.**

**30. to\_visit=[]**

**31.**

**32. sum\_sunny=dict()**

**33. max\_pop=sum(p)**

**34.**

**35. cloud\_pop=[0]\*len(y)**

**36.**

**37. idx\_x=0**

**38.**

**39. while idx\_x < len(x):**

**40. pos=x[idx\_x]**

**41. population=p[idx\_x]**

**42. j=0**

**43. while j < len(not\_visited):**

**44. element = not\_visited[j]**

**45. if y[element] - r[element] <= pos:**

**46. to\_visit.append((element, y[element]+r[element]))**

**47. del not\_visited[j]**

**48. else:**

**49. break**

**50.**

**51. j=0**

**52. to\_check=None**

**53. while j < len(to\_visit):**

**54. element, max\_pos = to\_visit[j]**

**55. if max\_pos - pos >= 0:**

**56. to\_check=element**

**57. if idx\_x not in sum\_sunny:**

**58. sum\_sunny[idx\_x]=population**

**59. else:**

**60. to\_check=None**

**61. break**

**62. j+=1**

**63. else:**

**64. del to\_visit[j]**

**65.**

**66. idx\_x+=1**

**67. if to\_check is not None:**

**68. cloud\_pop[to\_check]+=population**

**69.**

**70. for pop in sum\_sunny.values():**

**71. max\_pop-=pop**

**72.**

**73. return max(cloud\_pop) + max\_pop**

**74.**

**75. if \_\_name\_\_ == '\_\_main\_\_':**

**76. fptr = open(os.environ['OUTPUT\_PATH'], 'w')**

**77.**

**78. n = int(input().strip())**

**79.**

**80. p = list(map(int, input().rstrip().split()))**

**81.**

**82. x = list(map(int, input().rstrip().split()))**

**83.**

**84. m = int(input().strip())**

**85.**

**86. y = list(map(int, input().rstrip().split()))**

**87.**

**88. r = list(map(int, input().rstrip().split()))**

**89.**

**90. result = maximumPeople(p, x, y, r)**

**91.**

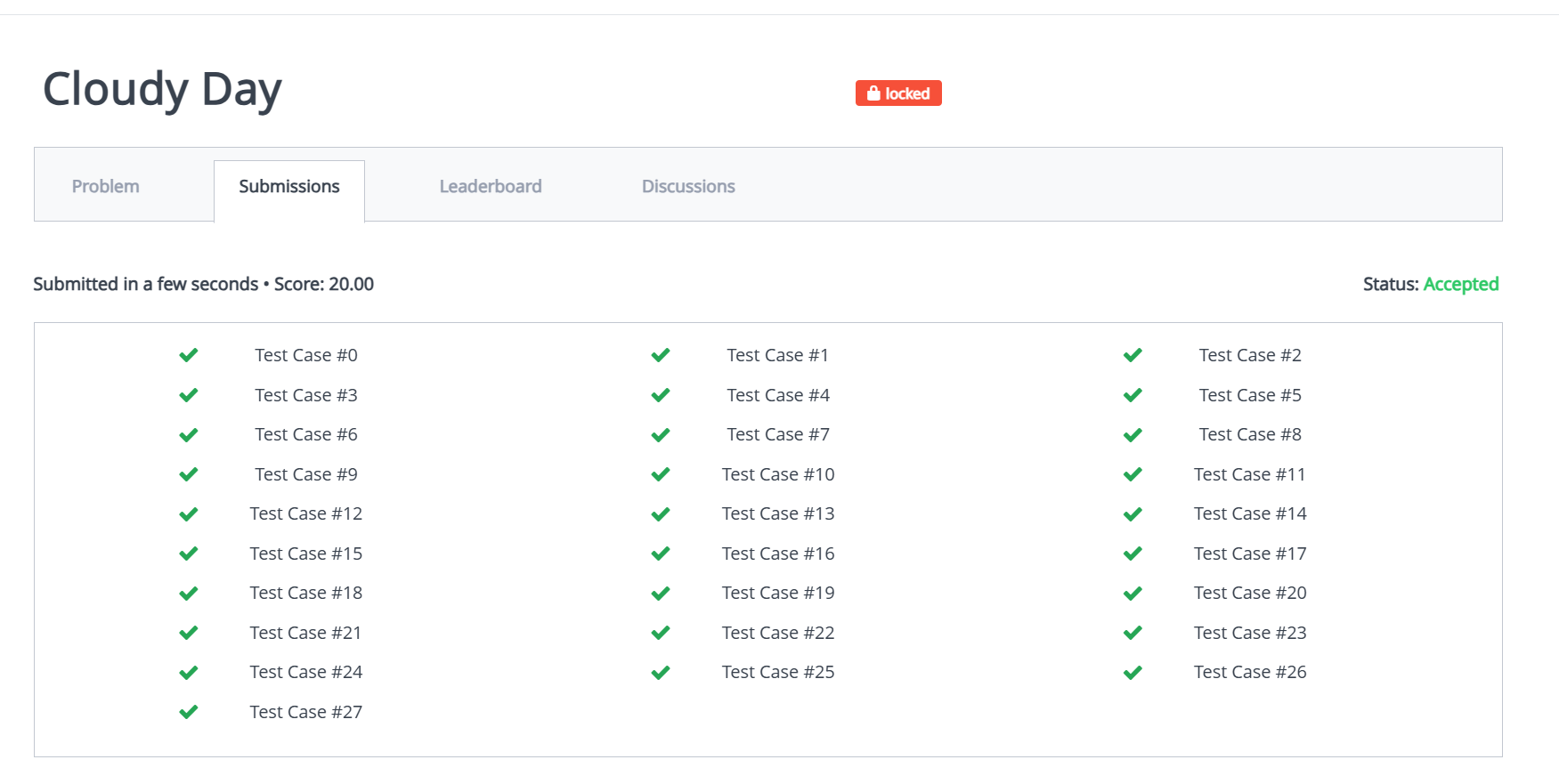
**92. fptr.write(str(result) + '\n')**

**93.**

**94. fptr.close()**

**95.**

**TESTCASES:**

****

**The Power Sum**

**Program:**

**1. import java.io.\*;**

**2. import java.math.\*;**

**3. import java.security.\*;**

**4. import java.text.\*;**

**5. import java.util.\*;**

**6. import java.util.concurrent.\*;**

**7. import java.util.function.\*;**

**8. import java.util.regex.\*;**

**9. import java.util.stream.\*;**

**10. import static java.util.stream.Collectors.joining;**

**11. import static java.util.stream.Collectors.toList;**

**12.**

**13. class Result {**

**14.**

**15. /\***

**16. \* Complete the 'powerSum' function below.**

**17. \***

**18. \* The function is expected to return an INTEGER.**

**19. \* The function accepts following parameters:**

**20. \* 1. INTEGER X**

**21. \* 2. INTEGER N**

**22. \*/**

**23.**

**24. public static int powerSum(int X, int N) {**

**25. return powerSumHelper(X, N, 1);**

**26. }**

**27.**

**28. private static int powerSumHelper(int X, int N, int currentNum) {**

**29.**

**30. if (X == 0) {**

**31. return 1;**

**32. }**

**33.**

**34. if (X < 0) {**

**35. return 0;**

**36. }**

**37.**

**38. int count = 0;**

**39.**

**40. for (int i = currentNum; ; i++) {**

**41. int power = (int) Math.pow(i, N);**

**42. if (power > X) {**

**43. break;**

**44. }**

**45. count += powerSumHelper(X - power, N, i + 1);**

**46. }**

**47.**

**48. return count;**

**49. }**

**50. }**

**51.**

**52. public class Solution {**

**53. public static void main(String[] args) throws IOException {**

**54. BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));**

**55. BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));**

**56.**

**57. int X = Integer.parseInt(bufferedReader.readLine().trim());**

**58.**

**59. int N = Integer.parseInt(bufferedReader.readLine().trim());**

**60.**

**61. int result = Result.powerSum(X, N);**

**62.**

**63. bufferedWriter.write(String.valueOf(result));**

**64. bufferedWriter.newLine();**

**65.**

**66. bufferedReader.close();**

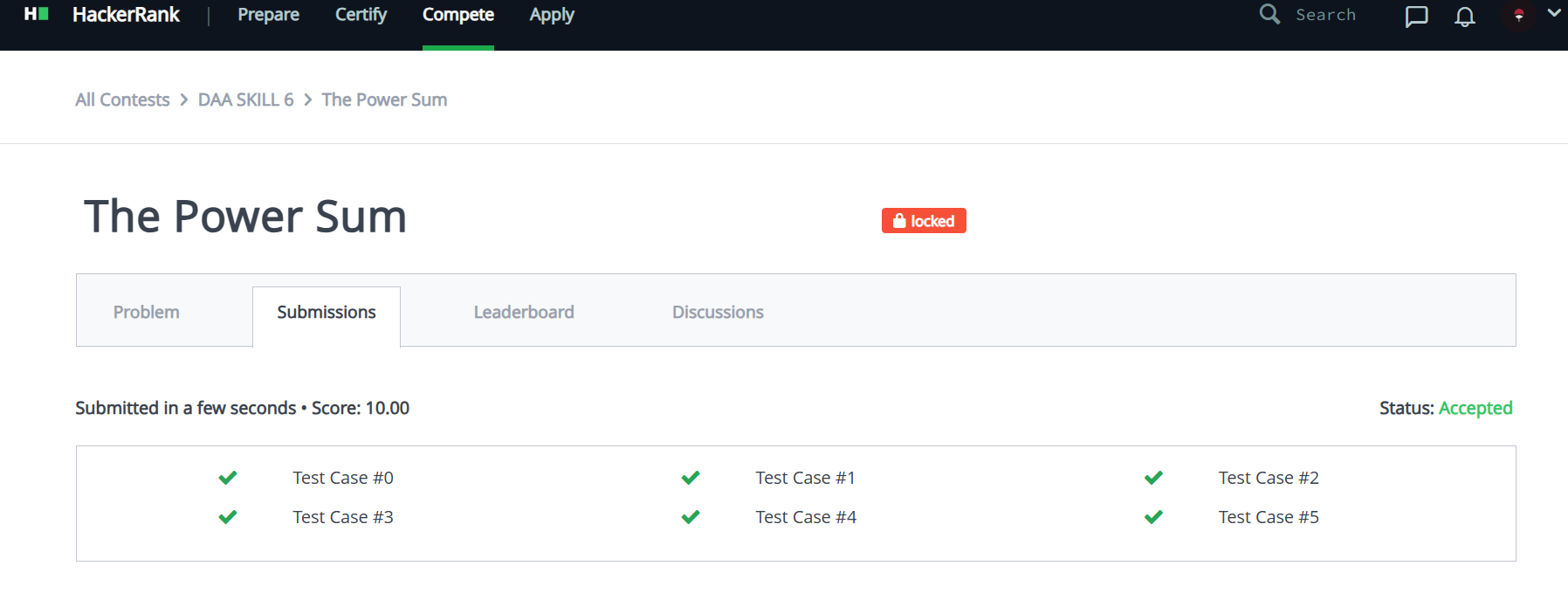
**67. bufferedWriter.close();**

**68. }**

**69. }**

**70.**

**TESTCASES:**

****

**Password Cracker**

**Program:**

**1. import java.io.\*;**

**2. import java.math.\*;**

**3. import java.security.\*;**

**4. import java.text.\*;**

**5. import java.util.\*;**

**6. import java.util.concurrent.\*;**

**7. import java.util.function.\*;**

**8. import java.util.regex.\*;**

**9. import java.util.stream.\*;**

**10. import static java.util.stream.Collectors.joining;**

**11. import static java.util.stream.Collectors.toList;**

**12.**

**13. class Result {**

**14.**

**15. /\***

**16. \* Complete the 'passwordCracker' function below.**

**17. \***

**18. \* The function is expected to return a STRING.**

**19. \* The function accepts following parameters:**

**20. \* 1. STRING\_ARRAY passwords**

**21. \* 2. STRING loginAttempt**

**22. \*/**

**23.**

**24. public static String passwordCracker(List<String> passwords, String loginAttempt) {**

**25. // Write your code here**

**26. int n = loginAttempt.length();**

**27. boolean[] dp = new boolean[n + 1];**

**28. int[] prev = new int[n + 1];**

**29.**

**30. dp[0] = true;**

**31.**

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

**33. for (String password : passwords) {**

**34. int len = password.length();**

**35. if (i >= len && dp[i - len] && loginAttempt.substring(i - len, i).equals(password)) {**

**36. dp[i] = true;**

**37. prev[i] = i - len;**

**38. break;**

**39. }**

**40. }**

**41. }**

**42.**

**43. if (!dp[n]) {**

**44. return "WRONG PASSWORD";**

**45. }**

**46.**

**47.**

**48. StringBuilder result = new StringBuilder();**

**49. for (int i = n; i > 0; ) {**

**50. int start = prev[i];**

**51. result.insert(0, loginAttempt.substring(start, i) + " ");**

**52. i = start;**

**53. }**

**54.**

**55. return result.toString().trim();**

**56. }**

**57. }**

**58.**

**59. public class Solution {**

**60. public static void main(String[] args) throws IOException {**

**61. BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));**

**62. BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));**

**63.**

**64. int t = Integer.parseInt(bufferedReader.readLine().trim());**

**65.**

**66. IntStream.range(0, t).forEach(tItr -> {**

**67. try {**

**68. int n = Integer.parseInt(bufferedReader.readLine().trim());**

**69.**

**70. List<String> passwords = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))**

**71. .collect(toList());**

**72.**

**73. String loginAttempt = bufferedReader.readLine();**

**74.**

**75. String result = Result.passwordCracker(passwords, loginAttempt);**

**76.**

**77. bufferedWriter.write(result);**

**78. bufferedWriter.newLine();**

**79. } catch (IOException ex) {**

**80. throw new RuntimeException(ex);**

**81. }**

**82. });**

**83.**

**84. bufferedReader.close();**

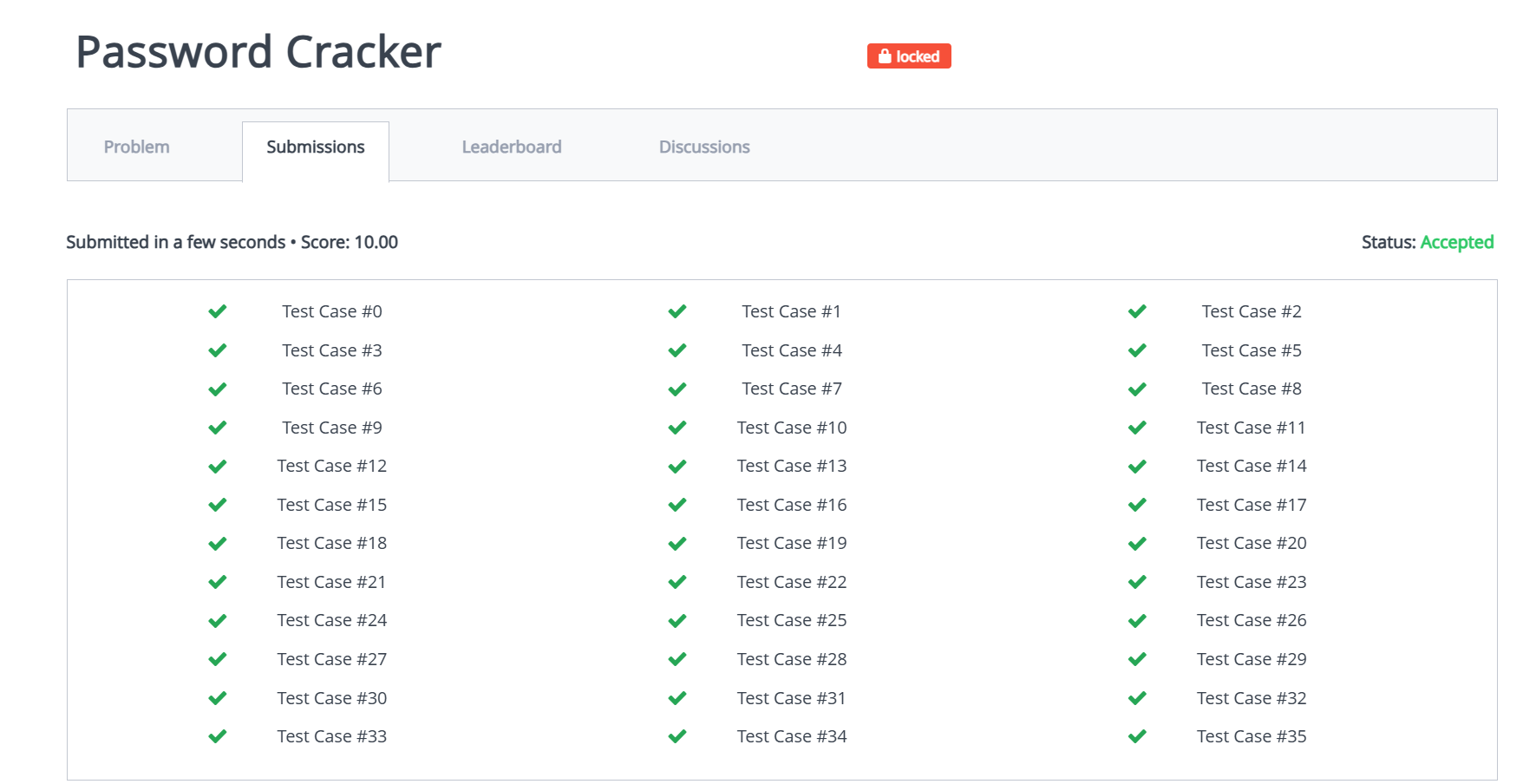
**85. bufferedWriter.close();**

**86. }**

**87. }**

**88.**

**TESTCASES:**

****