**Roll No…………….. Total No. of Pages:……**

**ST-1 (SET-VI)**

**6th SEMESTER 2023-24**

**CS192- Advanced Data Structures**

**Time allowed: 90 Minutes Max. Marks: 40**

**General Instructions:**

* **Follow the instructions given in each section.**
* **Make sure that you attempt the questions in order.**

**SECTION-A (10\*1 mark=10 marks)**

***(All questions are compulsory)***

1. The KMP algorithm is based on which principle?
   1. Divide and Conquer
   2. Greedy approach
   3. Sliding Window
   4. **Prefix-suffix matching**
2. In the Z function algorithm, the Z value of a character is defined as:
   1. **The length of the longest prefix of the text that matches the suffix starting at that character**
   2. The length of the longest suffix of the text that matches the prefix ending at that character
   3. The number of occurrences of the character in the text
   4. The number of occurrences of the character in the pattern
3. The condition for terminating binary search is:
   1. The element is found
   2. The entire array has been searched
   3. **The low index becomes greater than the high index**
   4. The target value is equal to the mid element
4. The average-case time complexity of binary search is:
   1. O(1)
   2. **O(log n)**
   3. O(n)
   4. O(n^2)
5. The dot product of two vectors is \_\_\_\_\_\_\_\_\_\_\_.
   1. **A scalar quantity**
   2. A vector quantity
   3. A complex number
   4. A matrix
6. Vectors can be added by \_\_\_\_\_\_\_\_\_\_\_.
   1. **Adding corresponding elements**
   2. Subtracting corresponding elements
   3. Multiplying corresponding elements
   4. Dividing corresponding elements
7. How can we check if two strings are rotations of each other using the frequency array approach?
   1. Compare their lengths
   2. Sort both strings and compare them
   3. **Count occurrences of characters in both strings and compare the frequency arrays**
   4. Use the two-pointer technique to compare characters
8. Given two strings "ABC" and "CDE," what will be the output of their longest common prefix length?
   1. **0**
   2. 1
   3. 2
   4. 3
9. In a prefix array, what is the value at index 0?
   1. The sum of all elements in the array.
   2. The difference between the first and last element in the array.
   3. The product of all elements in the array.
   4. **The value of the first element in the array.**
10. What is the main advantage of using a frequency array over iterating through an array to count occurrences?
    1. **Reduced time complexity.**
    2. Improved memory efficiency.
    3. Simplicity of implementation.
    4. Elimination of duplicate elements.

**SECTION-B (5\*2 mark=10 marks)**

***(All questions are compulsory)***

**SECTION-C(Coding Question) (2x5 marks=5 marks)**

Q16) Given an array of integers, write a C program to compute the prefix sum array and print the results.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | {1,2,3,4,5} | {23,43,12,45} | {2,5,7,3,2,1} |
| **Output** | Prefix Sum Array: 1 3 6 10 15 | Prefix Sum Array: 23 66 78 123 | Prefix Sum Array: 2 7 14 17 19 20 |

Solution :

**#include <stdio.h>**

**#define MAX\_SIZE 100**

**void prefixSum(int arr[], int prefix[], int size) {**

**int i;**

**prefix[0] = arr[0];**

**// Calculating the prefix sum**

**for (i = 1; i < size; i++) {**

**prefix[i] = prefix[i - 1] + arr[i];**

**}**

**}**

**int main() {**

**int arr[] = {2,5,7,3,2,1};**

**int size = sizeof(arr) / sizeof(arr[0]);**

**int prefix[MAX\_SIZE];**

**prefixSum(arr, prefix, size);**

**printf("Prefix Sum Array: ");**

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

**printf("%d ", prefix[i]);**

**}**

**return 0;**

**}**

Q17) Given two strings 's1' and 's2,' check if 's2' contains a permutation of 's1.'

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | s1=ab  s2=eidbaooo | s1=car  s2=racecar | s1=ring  s2=night |
| **Output** | s2 contains a permutation of s1: true | s2 contains a permutation of s1: true | s2 contains a permutation of s1: false |

Solution :

**#include <stdio.h>**

**#include <stdbool.h>**

**#include <string.h>**

**#define MAX\_CHARS 256**

**bool compareArrays(int arr1[], int arr2[]) {**

**for (int i = 0; i < MAX\_CHARS; i++) {**

**if (arr1[i] != arr2[i]) {**

**return false; // Return false if any count is different**

**}**

**}**

**return true; // Return true if all counts are the same**

**}**

**bool checkPermutation(char\* s1, char\* s2) {**

**int s1Count[MAX\_CHARS] = {0}; // Frequency array for characters in s1**

**int s2Count[MAX\_CHARS] = {0}; // Frequency array for characters in s2**

**int s1Len = strlen(s1);**

**int s2Len = strlen(s2);**

**if (s1Len > s2Len) {**

**return false; // If s1 is longer than s2, s2 cannot contain a permutation of s1**

**}**

**for (int i = 0; i < s1Len; i++) {**

**s1Count[s1[i]]++; // Increment the count of character in s1**

**s2Count[s2[i]]++; // Increment the count of character in s2**

**}**

**for (int i = s1Len; i < s2Len; i++) {**

**if (compareArrays(s1Count, s2Count)) { // Compare the frequency arrays**

**return true;**

**}**

**s2Count[s2[i]]++; // Increment the count of the current character in s2**

**s2Count[s2[i - s1Len]]--; // Decrement the count of the character leaving the window**

**}**

**return compareArrays(s1Count, s2Count); // Compare the frequency arrays at the end**

**}**

**int main() {**

**char s1[] = "ring";**

**char s2[] = "night";**

**bool result = checkPermutation(s1, s2);**

**printf("s2 contains a permutation of s1: %s\n", (result) ? "true" : "false");**

**return 0;**

**}**

**SECTION-D (Coding Question)(1x10 mark=10 mark)**

Q18) Given two strings s and t. Return the minimum number of operations required to convert s to t.

The possible operations are permitted:

Insert a character at any position of the string.

Remove any character from the string.

Replace any character from the string with any other character.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | s = "mumbai", t = "momba" | s = " chopper", t = " shopner" | s = " pizza", t = " pasta" |
| **Output** | 2 | 2 | 3 |

Solution :

**#include <bits/stdc++.h>**

**using namespace std;**

**class Solution {**

**public:**

**// Function to calculate the minimum edit distance between two strings 's' and 't'.**

**int editDistance(string s, string t) {**

**int m = s.length(); // Length of string 's'.**

**int n = t.length(); // Length of string 't'.**

**// Create a 2D DP array to store the minimum edit distance for all substrings of 's' and 't'.**

**int dp[m + 1][n + 1];**

**// Initializing the base cases of the DP array.**

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

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

**if (i == 0)**

**dp[i][j] = j; // If string 's' is empty, the minimum edit distance is the length of string 't'.**

**else if (j == 0)**

**dp[i][j] = i; // If string 't' is empty, the minimum edit distance is the length of string 's'.**

**else if (s[i - 1] == t[j - 1])**

**dp[i][j] = dp[i - 1][j - 1]; // If the characters match, no edit is required, take the diagonal value.**

**else**

**dp[i][j] = 1 + min(dp[i - 1][j - 1], min(dp[i][j - 1], dp[i - 1][j])); // Otherwise, choose the minimum of three edit operations (insert, remove, or replace).**

**}**

**}**

**return dp[m][n]; // Return the minimum edit distance between the strings 's' and 't'.**

**}**

**};**

**int main() {**

**string s, t;**

**cin >> s >> t; // Input the two strings.**

**Solution ob; // Create an object of the Solution class.**

**// Call the editDistance function to calculate the minimum edit distance.**

**int ans = ob.editDistance(s, t);**

**cout << ans << "\n"; // Output the minimum edit distance.**

**return 0;**

**}**