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

**ST-1 (SET-III)**

**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. In binary search, the worst-case scenario occurs when:
   1. **The element is not present in the array**
   2. The element is in the middle of the array
   3. The array contains duplicate elements
   4. The array is sorted indescending order
2. Binary search is NOT suitable for which of the following scenarios?
   1. Searching for a word in a dictionary
   2. **Finding the maximum value in an unsorted array**
   3. Determining if an element exists in a sorted array
   4. Finding the closest value to a given target in a sorted array
3. Which sorting algorithm is not an in-place sorting algorithm?
   1. Bubble Sort
   2. **Heap Sort**
   3. Quick Sort
   4. Selection Sort
4. The Radix Sort algorithm sorts elements based on their:
   1. Comparison with a pivot element
   2. **Binary representation**
   3. Frequency of occurrence
   4. Position in the original array
5. The Z function algorithm finds the longest common prefix of a text with itself in:
   1. **O(N)**
   2. O(M)
   3. O(N+M)
   4. O(N\*M)
6. The KMP algorithm is particularly efficient for pattern searching in:
   1. **Texts with repeating patterns**
   2. Texts with large alphabets
   3. Texts with long patterns
   4. Texts with random characters
7. What is the primary benefit of using a frequency array?
   1. It allows constant-time access to array elements.
   2. It provides efficient sorting of array elements.
   3. **It allows counting occurrences of elements in an array.**
   4. It simplifies the computation of prefix sums.
8. Which of the following operations can be performed efficiently using prefix arrays?
   1. Finding the maximum element in the array.
   2. Finding the minimum element in the array.
   3. Computing the average of the array elements.
   4. **Calculating the sum of elements in a given range.**
9. How can we efficiently find the most frequent character in a string using a frequency array?
   1. Sort the string and find the character with the highest frequency
   2. Use a priority queue to track the character frequencies
   3. **Iterate through the frequency array and find the character with the highest count**
   4. Convert the string into a set of characters and find the most frequent one
10. The Manacher's algorithm is used to find the:
    1. **Longest Palindromic Substring**
    2. Longest Common Subsequence
    3. Longest Increasing Subsequence
    4. Shortest Palindromic Substring

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

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

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

Q16) Write a C program to count the occurrences of a given substring in a string.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Enter a string: spring  Enter a substring: ring | Enter a string: points  Enter a substring: int | Enter a string: reel  Enter a substring: e |
| **Output** | Number of occurrences: 1 | Number of occurrences: 1 | Number of occurrences: 2 |

Solution :

**#include <stdio.h>**

**#include <string.h>**

**// Function to count occurrences of a substring**

**int countOccurrences(char str[], char sub[]) {**

**int count = 0;**

**int subLen = strlen(sub);**

**for (int i = 0; i <= strlen(str) - subLen; i++) {**

**int j;**

**for (j = 0; j < subLen; j++) {**

**if (str[i + j] != sub[j])**

**break;**

**}**

**if (j == subLen)**

**count++;**

**}**

**return count;**

**}**

**int main() {**

**char str[100], sub[100];**

**printf("Enter a string: ");**

**gets(str);**

**printf("Enter a substring: ");**

**gets(sub);**

**int occurrences = countOccurrences(str, sub);**

**printf("Number of occurrences: %d", occurrences);**

**return 0;**

**}**

Q17) Given an array of integers, write a program to find the length of the longest subarray with a sum of zero using the prefix sum array.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | {4, 2, -3, 1, 6, -3, -1, 5, 1} | {1, 2, 3} | {6,8,-8,-6,2} |
| **Output** | Length of the longest subarray with a sum of zero: 5 | Length of the longest subarray with a sum of zero: 0 | Length of the longest subarray with a sum of zero: 4 |

Solution :

**#include <stdio.h>**

**int findLongestSubarray(int arr[], int n) {**

**int maxLength = 0; // Variable to store the length of the longest subarray**

**int sum = 0; // Variable to store the prefix sum**

**int prefixSum[n]; // Array to store the prefix sums**

**// Initialize the prefix sum array with 0**

**prefixSum[0] = 0;**

**// Calculate the prefix sum for each element in the array**

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

**sum += arr[i];**

**prefixSum[i + 1] = sum;**

**}**

**// Find the longest subarray with sum 0**

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

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

**// If the sum of the subarray is 0 and the length is greater than maxLength, update maxLength**

**if (prefixSum[j] - prefixSum[i] == 0 && (j - i) > maxLength) {**

**maxLength = j - i;**

**}**

**}**

**}**

**return maxLength;**

**}**

**int main() {**

**int arr[] = {6,8,-8,-6,2};**

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

**int length = findLongestSubarray(arr, n);**

**printf("Length of the longest subarray with a sum of zero: %d\n", length);**

**return 0;**

**}**

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

Q18) Given an array of lowercase strings A[] of size N, determine if the strings can be chained together to form a circle.

A string X can be chained together with another string Y if the last character of X is same as first character of Y. If every string of the

array can be chained with exactly two strings of the array(one with the first character and second with the last character of the string), it will form a circle.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | N = 3  A[] = { "abc", "bcd", "cdf" } | N = 4  A[] = { "ab" , "bc", "cd", "da" } | N = 2  A[] = { "cat" , "tac" } |
| **Output** | 0 | 1 | 1 |

Solution :

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

**using namespace std;**

**class Solution**

**{**

**public:**

**// Depth-First Search (DFS) function to find if a circular chain can be formed**

**void dfs(int src, vector<string> &arr, int n, int &cnt, vector<int> &visited)**

**{**

**visited[src] = 1; // Mark the current node as visited**

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

**{**

**if (!visited[i]) // If the node is not visited yet**

**{**

**// Check if the last character of the current string matches the first character of the next string**

**if (arr[src][arr[src].length() - 1] == arr[i][0])**

**{**

**cnt++; // Increment the count of strings in the circular chain**

**dfs(i, arr, n, cnt, visited); // Recursively call DFS on the next string in the chain**

**}**

**}**

**}**

**return;**

**}**

**// Function to check if a circular chain can be formed using the given strings**

**int isCircle(int n, vector<string> arr)**

**{**

**// Create a frequency map to count occurrences of characters at the beginning and end of strings**

**unordered\_map<char, int> mp;**

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

**{**

**mp[arr[i][0]]++;**

**mp[arr[i][arr[i].length() - 1]]++;**

**}**

**bool flag = 1;**

**for (auto it : mp)**

**{**

**if (it.second % 2 == 1)**

**{**

**flag = false; // If there is any character with an odd frequency, a circular chain is not possible**

**break;**

**}**

**}**

**// Special case when only one string is provided**

**if (n == 1)**

**{**

**if (arr[0][0] == arr[0][arr[0].length() - 1])**

**return 1; // If the single string is already a circular chain, return true**

**return 0; // Otherwise, return false**

**}**

**vector<int> visited(n, 0); // Create a vector to mark visited strings during DFS**

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

**{**

**int cnt = 1; // Initialize the count of strings in the current chain to 1**

**dfs(i, arr, n, cnt, visited); // Start DFS from the current string**

**if (cnt == n)**

**return 1 && flag; // If the count of strings in the chain is equal to the total number of strings, return true**

**}**

**return 0 && flag; // If no circular chain is found, return false**

**}**

**};**

**int main()**

**{**

**int N;**

**cin >> N;**

**vector<string> A;**

**string s;**

**for (int i = 0; i < N; i++)**

**{**

**cin >> s;**

**A.push\_back(s);**

**}**

**Solution ob;**

**cout << ob.isCircle(N, A) << endl; // Output 1 if a circular chain can be formed, 0 otherwise**

**return 0;**

**}**