Task 8 : Arrays -Introduction, memory allocation (with row/column major), operations (insert, delete, search) sorted and unsorted array, suffix array, subsequence and subarray. -CO3-K3

Raman loves Mathematics a lot. One day his maths teacher gave him an interesting problem. He was given an array 'A' consisting of 'n' integers, he was needed to find the maximum value
Input:

- First line of input contains an integer *T* denoting number of test cases.
- Each test case contains two lines, first line contains integer *n* where *n* is the number of elements in array
- Second line contains *n* space separated integers

A_i. Output:
Print the maximum value of the above give expression, for each test case separated in a new line

Sample Input:

2

3

1 2 5

4

1 2 3 4

Sample output:

5

4

Algorithm:

1. Read the number of elements n in the array A
2. Read the array A
3. Initialize max_val to -1
4. Repeat the following steps for all possible pairs (i, j) where 1 <= i < j <= n:

i. Calculate the value of the expression (A[i] * A[j]) + (i - j)

ii. If the calculated value is greater than max_val, set max_val to the calculated value

5. Print the value of max_val for the current test case
6. End

Program:

```
#include
<stdio.h>
```

```
void rotate(int arr[], int n, int k) {
    k = k % n;
    int temp[k];

  for (int i = 0; i < k; i++) {
        temp[i] = arr[n - k + i];
    }
    for (int i = n - 1; i >= k; i--) {
        arr[i] = arr[i - k];
    }
    for (int i = 0; i < k; i++) {
        arr[i] = temp[i];
    }
}

int main()
    { int n,
    k;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the array elements:
    "); for (int i = 0; i < n; i++) {

 scanf("%d", &arr[i]);
    }
    printf("Enter the number of positions to rotate: ");
    scanf("%d", &k);
    rotate(arr, n, k);
    printf("The rotated array is: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

OUTPUT:

Enter the size of the array: 10

1Enter the array elements: 1 2 3 4 5 6 7 8 9 0

Enter the number of positions to rotate: 5

The rotated array is: 6 7 8 9 0 1 2 3 4 5

Problem:

Students have become secret admirers of grade. They find the course exciting and the professors amusing. After a superb Mid Semester examination its now time for the results. The TAs have released the marks of students in the form of an array, where arr[i] represents the marks of the ith student. Since you are a curious kid, you want to find all the marks that are not smaller than those on its right side in the array.

Input Format

The first line of input will contain a single integer n denoting the number of students.

The next line will contain n space separated integers representing the marks of students.

Output Format
Output all the integers separated in the array from left to right that are not smaller than those on its right side.

Constraints

1 <= n <= 1000000

0 <= arr[i] <= 10000

Sample input:

4

5 7 3 6

Sample output:

7

Algorithm:

1. Read the input values of n and the array arr[].
2. Initialize a variable max as the first element of the array arr[0].
3. Traverse the array arr[] from right to left and for each element arr[i] do the following: a. If arr[i] >= max, then set max = arr[i] and print arr[i].
4. End.

Program:

```
#include <stdio.h>

int maxSubArray(int arr[], int n) {
   int max_so_far = arr[0];
   int max_ending_here = arr[0];

   for (int i = 1; i < n; i++) {
      max_ending_here = (arr[i] > max_ending_here + arr[i]) ? arr[i] : max_ending_here + arr[i];
      max_so_far = (max_so_far > max_ending_here) ? max_so_far : max_ending_here;
   }

   return max_so_far;
}

int main()
   { int n;
   printf("Enter the size of array: ");
   scanf("%d", &n);

   int arr[n];
   printf("Enter the elements of array: ");
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   int max_sum = maxSubArray(arr, n);
   printf("The maximum subarray sum is: %d\n", max_sum);

   return 0;
}
```

OUTPUT:

Enter the size of array: 5
Enter the elements of array: 1 2 3 4 5

The maximum subarray sum is: 15

Result: Thus the program is executed and verified successfully

Task 9: String Processing- Basics, String functions (based on different language), String operations, Two algorithms for string pattern matching - Naïve approach and Robin karp approach.-CO3-K3

Merge two strings

Given two strings S1 and S2 as input, the task is to merge them alternatively i.e. the first character of S1 then the first character of S2 and so on till the strings end.
NOTE: Add the whole string if other string is empty.
Input:
The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case contains two strings S1 and S2.
Output:
For each test case, in a new line, print the merged string.
Constraints:
1 <= T <= 100
1 <= |S1|, |S2| <= 10^4
Example:
Input:
2

Hello
Bye abc
def
Output:
HBeylel
o adbecf

Algorithm:

a. Read the first string S1
b. Read the second string S2
c. Initialize two variables i and j to 0
d. Initialize an empty string mergedStr
e. Repeat the following steps while i is less than the length of S1 and j is less than the length of S2:
i. Append the i-th character of S1 to mergedStr
ii. Append the j-th character of S2 to mergedStr
iii. Increment i by 1
iv. Increment j by 1
f. If i is less than the length of S1, append the remaining characters of S1 to mergedStr
g. If j is less than the length of S2, append the remaining characters of S2 to mergedStr
h. Print mergedSt

Program:

```
#include <stdio.h>
#include <string.h>
```

```
int main()
    { int t;
    scanf("%d", &t); // read the number of test cases
```

```
    while (t--) {
        char s1[10005], s2[10005], ans[20005];
        scanf("%s %s", s1, s2); // read the two strings

        int len1 = strlen(s1), len2 =
        strlen(s2); int i, j, k;
        i = j = k = 0;

        // merge the strings alternatively
        while (i < len1 && j < len2) {
            ans[k++] =
            s1[i++]; ans[k++]
            = s2[j++];
        }

        // add the remaining characters from s1 or s2
        while (i < len1) ans[k++] = s1[i++];
        while (j < len2) ans[k++] = s2[j++];

        ans[k] = '\0'; // add null terminator to the merged string

        printf("%s\n", ans); // print the merged string
    }

    return 0;
}
```

Input:
2
Hello
Bye abc
def
Output:
HBeylel
o adbecf

Taking input

You are given two inputs: a(integer), and b(string). You need to take the input and print a and b separated by a space.

Input Format:

First line of input conatins number of testcases T. T testcases follow. For each testcase, there will be one line of input containing a and b.

Output Format:

For each testcase, print a and b separated by a space.

Your Task:

This is a function problem. You need to write the command to take input of a and b inside the function inputData()

Taking input

Constraints:

1 <= T <=

10
$1 <= a <= 10^6$
Input:
2
5

Hell
o 7
Geeks
Output:
5 Hello
7 Geek

Algorithm:

1. Read the number of testcases T from the input.
2. Loop through T testcases
    a. Read the integer a from input.
    b. Read the string b from input using fgets() or scanf("%[^\n]s", b).
    c. Print the values of a and b separated by a space using printf() function.
3. End loop.

Program:

```
#include <iostream>
using namespace std;

void inputData()
   { int t, a;
   string b;
   cin >> t;
   while(t--)
   {
      cin >> a >> b;
      cout << a << " " << b << endl;
   }
}

int main() {
   inputData()
   ; return 0;
}
```

Input:
2
5
Hell
o 7
Geeks
Output:
5 Hello
7 Geek

Result: Thus the program is executed and verified successfully.

Task 10: Bit Map-Introduction, XOR, AND, OR, right shift, left shift.-CO3-K3

Reverse bits of a given 32 bits unsigned integer.
Note:
Note that in some languages, such as Java, there is no unsigned integer type. In this case, both input

and output will be given as a signed integer type. They should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in

Example 2 above, the input represents the signed integer -3 and the output represents the signed integer -1073741825.

Example 1:

Input: n = 00000010100101000001111010011100

Output: 964176192 (00111001011110000010100101000000)

Explanation: The input binary string 00000010100101000001111010011100 represents the unsigned integer 43261596, so return 964176192 which its binary representation is 00111001011110000010100101000000.

Example 2:

Input: n = 11111111111111111111111111111101

Output: 3221225471 (10111111111111111111111111111111)

Explanation: The input binary string 11111111111111111111111111111101 represents the unsigned integer 4294967293, so return 3221225471 which its binary representation is 10111111111111111111111111111111.

Constraints:

The input must be a binary string of length 32.

Algorithm:

1. Take the 32-bit unsigned integer as input in binary format.
2. Convert the binary string into a character array.
3. Initialize two pointers: one pointing to the beginning of the array (i.e., the most significant bit), and the other pointing to the end of the array (i.e., the least significant bit).
4. Swap the bits at the two pointers and increment the first pointer and decrement the second pointer until they meet or cross each other.
5. Convert the modified character array back to a binary string.
6. Convert the binary string to an integer and return the result.

Program:

```
#include <stdio.h>

unsigned int reverseBits(unsigned int num) {
   unsigned int reversed = 0;
   int bits = sizeof(num) * 8;

   for (int i = 0; i < bits; i++) {
      if (num & (1 << i)) {
         reversed |= 1 << (bits - 1 - i);
      }
   }

   return reversed;
}

int main() {
   unsigned int num = 10;
   printf("%u\n", num);
   printf("%u\n",
   reverseBits(num)); return 0;
}
```

OUTPUT:

10
1342177280

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the Hamming weight).

Note:

Note that in some languages, such as Java, there is no unsigned integer type. In this case, the input will be given as a signed integer type. It should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.

In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in Example 3, the input represents the signed integer. -3.

Example 1:

Input: n = 00000000000000000000000000001011

Output: 3

Explanation: The input binary string 00000000000000000000000000001011 has a total of three '1' bits.

Example 2:

Input: n = 00000000000000000000000010000000

Output: 1

Explanation: The input binary string 00000000000000000000000010000000 has a total of one '1' bit.

Example 3:

Input: n = 11111111111111111111111111111101

Output: 31

Explanation: The input binary string 11111111111111111111111111111101 has a total of thirty one '1' bits.

Constraints:

The input must be a binary string of length 32.

3. The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Give two integers x and y, return the Hamming distance between them.

Example 1:

Input: x = 1, y = 4

Output: 2

Explanation:

1 (0 0 0 1)

4 (0 1 0 0)

↑ ↑

The above arrows point to positions where the corresponding bits are different.

Example 2:

Input: x = 3, y = 1

Output: 1

Constraints:

0 <= x, y <= 231 – 1

Algorithm:

Algorithm for counting number of 1 bits in an unsigned integer:

1. Initialize a variable count to 0.
2. Loop through each bit in the 32-bit integer.
3. If the current bit is a 1, increment the count by 1.
4. After looping through all the bits, return the count.

Algorithm for computing the Hamming distance between two integers:

1. Initialize a variable count to 0.
2. Loop through each bit in the 32-bit integers x and y.

3. If the current bit in x is different from the current bit in y, increment the count by 1.
4. After looping through all the bits, return the count.

Program:

```
#include <stdio.h>

unsigned int hammingDistance(unsigned int x, unsigned int y)

 {
   unsigned int dist = 0;
   unsigned int val = x ^ y; // bitwise XOR of x and y

   while (val) {
      // count the number of set bits (i.e., 1s) in val
      dist++;
      val &= val - 1;
   }

   return dist;
}

int main() {
   unsigned int x = 10;
   unsigned int y = 15;
   printf("%u\n", hammingDistance(x, y));
   return 0;
```

OUTPUT:

2

Result: Thus the program is executed and verified successfully.

executed and verified successfully.