ARYAMAN MISHRA

19BCE1027

1.a)

import java.util.\*;

class SubSequence

{

// Main program to call all functions

public static void main (String[] args)

{

Scanner sc=new Scanner(System.in);

String str1,str2;

int m,n;

boolean result;

System.out.println("Enter 2 Strings.");

str1=sc.nextLine();

str2=sc.nextLine();

m = str1.length();

n = str2.length();

result = isSubSequence(str1, str2, m, n);

if(result==true)

System.out.println("Yes");

else

System.out.println("No");

}

// Returns true if str1[] is a subsequence of str2[]

// m is length of str1 and n is length of str2

static boolean isSubSequence(String str1, String str2, int m, int n)

{

// Base Cases

if (m == 0)

return true;

if (n == 0)

return false;

// If last characters of two strings are matching

if (str1.charAt(m-1) == str2.charAt(n-1))

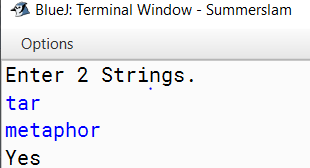
return isSubSequence(str1, str2, m-1, n-1);

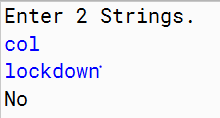
// If last characters are not matching

return isSubSequence(str1, str2, m, n-1);

}

}





1.b) import java.util.\*;

class Combination

{

/\*Main program to call and check for above function\*/

public static void main (String[] args)

{

Scanner sc=new Scanner(System.in);

int n,k,i;

System.out.println("Enter elements to be put into array.");

n=sc.nextInt();

int arr[]=new int[n];

System.out.println("Enter a number for permutations.");

k=sc.nextInt();

if(n==0 || k==0)

{

System.out.println("No permutations available."); //base case

}

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

{

System.out.println("Enter elements in array");

arr[i]=sc.nextInt();

}

printCombination(arr, n, k);

}

/\* arr[] ---> Input Array

data[] ---> Temporary array to store current combination

start & end ---> Staring and Ending indexes in arr[]

index ---> Current index in data[]

k ---> Size of a combination to be printed \*/

static void combinationUtil(int arr[], int data[], int start, int end, int index, int k)

{

// Current combination is ready to be printed, print it

if (index == k)

{

for (int j=0; j<k; j++)

System.out.print(data[j]+" ");

System.out.println("");

return;

}

// replace index with all possible elements. The condition

// "end-i+1 >= r-index" makes sure that including one element

// at index will make a combination with remaining elements

// at remaining positions

for (int i=start; i<=end && end-i+1 >= k-index; i++)

{

data[index] = arr[i];

combinationUtil(arr, data, i+1, end, index+1, k);

}

}

// The main function that prints all combinations of size r

// in arr[] of size n. This function mainly uses combinationUtil()

static void printCombination(int arr[], int n, int r)

{

// A temporary array to store all combination one by one

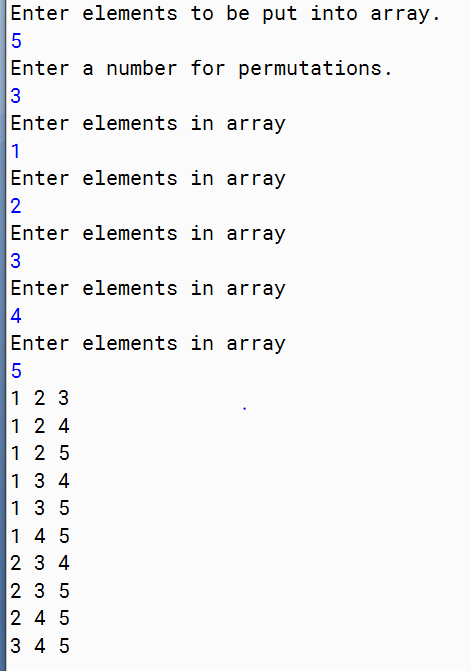
int data[]=new int[r];

// Print all combination using temprary array 'data[]'

combinationUtil(arr, data, 0, n-1, 0, r);

}

}



3) #include<stdio.h>

// structure of node of doubly linked list

struct Node {

int data;

struct Node\* next, \*prev;

};

// function to count triplets in a sorted doubly linked list

// whose sum is equal to a given value 'x'

int countTriplets(struct Node\* head, int x)

{

struct Node\* ptr1, \*ptr2, \*ptr3;

int count = 0;

for (ptr1 = head; ptr1 != NULL; ptr1 = ptr1->next)

for (ptr2 = ptr1->next; ptr2 != NULL; ptr2 = ptr2->next)

for (ptr3 = ptr2->next; ptr3 != NULL; ptr3 = ptr3->next)

if ((ptr1->data \* ptr2->data \* ptr3->data) == x)

count++;

return count;

}

void insert(struct Node\*\* head, int data)

{

// allocate node

struct Node\* temp = new Node();

// put in the data

temp->data = data;

temp->next = temp->prev = NULL;

if ((\*head) == NULL)

(\*head) = temp;

else {

temp->next = \*head;

(\*head)->prev = temp;

(\*head) = temp;

}

}

int main()

{

// start with an empty doubly linked list

struct Node\* head = NULL;

int i,n,a;

printf("Enter number of elements\n");

scanf("%d",&n);

printf("Insert values in sorted order\n");

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

{

scanf("%d",&a);

insert(&head,a);

}

printf("Enter value of x\n");

int x;

scanf("%d",&x);

printf("Count=%d\n",countTriplets(head, x));

return 0;

}

#include<stdio.h>

#include <cstdlib>

/\* structure for a node in circular

linked list \*/

struct Node

{

int data;

struct Node \*next;

};

// To create a new node of circular

// linked list

Node \*newNode(int data)

{

Node \*temp = new Node;

temp->next = temp;

temp->data = data;

}

/\* Function to find the only person left

after one in every k-th node is killed

in a circle of n nodes \*/

void getJosephusPosition(int k, int n)

{

// Create a circular linked list of

// size N.

Node \*head = newNode(1);

Node \*prev = head;

for (int i = 2; i <= n; i++)

{

prev->next = newNode(i);

prev = prev->next;

}

prev->next = head; // Connect last

// node to first

/\* while only one node is left in the

linked list\*/

Node \*ptr1 = head, \*ptr2 = head;

while (ptr1->next != ptr1)

{

// Find m-th node

int count = 1;

while (count != k)

{

ptr2 = ptr1;

ptr1 = ptr1->next;

count++;

}

/\* Remove the m-th node \*/

ptr2->next = ptr1->next;

free(ptr1);

ptr1 = ptr2->next;

}

printf ("Last person left standing is %d\n ", ptr1->data);

}

int main()

{

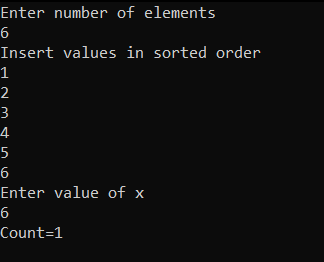
int n,k;

printf("Input the total number of persons n and a number k which indicates that k-1 persons are skipped and kth person is killed in circle.\n");

scanf("%d%d",&n,&k);

getJosephusPosition(k, n);

return 0;

} 

4)

#include<stdio.h>

#include <cstdlib>

/\* structure for a node in circular

linked list \*/

struct Node

{

int data;

struct Node \*next;

};

// To create a new node of circular

// linked list

Node \*newNode(int data)

{

Node \*temp = new Node;

temp->next = temp;

temp->data = data;

}

/\* Function to find the only person left

after one in every k-th node is killed

in a circle of n nodes \*/

void getJosephusPosition(int k, int n)

{

// Create a circular linked list of

// size N.

Node \*head = newNode(1);

Node \*prev = head;

for (int i = 2; i <= n; i++)

{

prev->next = newNode(i);

prev = prev->next;

}

prev->next = head; // Connect last

// node to first

/\* while only one node is left in the

linked list\*/

Node \*ptr1 = head, \*ptr2 = head;

while (ptr1->next != ptr1)

{

// Find m-th node

int count = 1;

while (count != k)

{

ptr2 = ptr1;

ptr1 = ptr1->next;

count++;

}

/\* Remove the m-th node \*/

ptr2->next = ptr1->next;

free(ptr1);

ptr1 = ptr2->next;

}

printf ("Last person left standing is %d\n ", ptr1->data);

}

int main()

{

int n,k;

printf("Input the total number of persons n and a number k which indicates that k-1 persons are skipped and kth person is killed in circle.\n");

scanf("%d%d",&n,&k);

getJosephusPosition(k, n);

return 0;

}

