Nitin Chaudhary
9921103163
F7
APS Lab Week 11_12
Ans 1-

The approach which I have used to solve this problem is told by neeraj sir in the class.
The time complexity of the below code is: O(n)
The below code is giving wrong output for the strings which contains repeated vowels.

```
#include <bits/stdc++.h>
using namespace std;
bool isVovel(char c){
    return (c=='a'||c=='e'||c=='i'||c=='o'||c=='u');
}
int solve(string arr ,int sizei,int k){
  int dp[sizei];
  if(isVovel(arr[0]))dp[0]=1;
  else dp[0]=0;
  for(int i=1;i<sizei;i++){
    if(isVovel(arr[i]))dp[i]=dp[i-1]+1;
    else dp[i]=dp[i-1];
  }
  //for(int i=0;i<sizei;i++)cout<<dp[i]<<" ";

int i = 0, j = 0, ans = -1;
    while (j < sizei) {
        if (dp[j] - dp[i] == k) {
            //int temp=ans;
            ans = max(ans, j - i);
            //if(temp!=ans)cout<<endl<<i<<" "<<j<<endl;
            j++;
            i++;
        } else if (dp[j] - dp[i] > k) {
            i++;
        } else {
            j++;
```

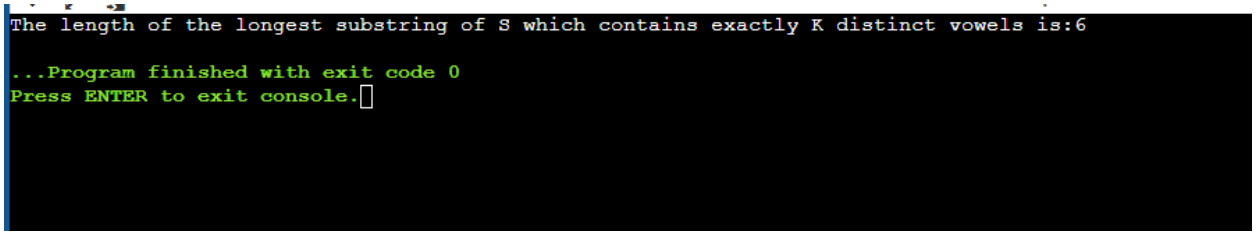
```
        }
    }
  return ans;
}
int main()
{
    string s = "artyebui";
    int k=2;
    cout<<"The length of the longest substring of S which contains exactly K distinct
vowels is:"<<solve(s,8,k);
    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
bool isVovel(char c){
    return (c=='a'||c=='e'||c=='i'||c=='o'||c=='u');
}
int solve(string arr ,int sizei,int k){
  int dp[sizei];
  unordered_set<char>us;
  if(isVovel(arr[0])){
      dp[0]=1;
      us.insert(arr[0]);
  }
  else dp[0]=0;
  for(int i=1;i<sizei;i++){
    if(isVovel(arr[i])&&us.find(arr[i])==us.end()){
        dp[i]=dp[i-1]+1;
        us.insert(arr[i]);
    }
    else dp[i]=dp[i-1];
```

```
  }
  //for(int i=0;i<sizei;i++)cout<<dp[i]<<" ";

int i = 0, j = 0, ans = -1;
    while (j < sizei) {
        if (dp[j] - dp[i] == k) {
            //int temp=ans;
            ans = max(ans, j - i+1);
            //if(temp!=ans)cout<<endl<<i<<" "<<j<<endl;
            j++;
            //i++;
        } else if (dp[j] - dp[i] > k) {
            i++;
        } else {
            j++;
        }
    }
  return ans;
}
int main()
{
    //string s = "nitiiijeetiuch";
    string s = "nittttjehtluch";
    int k=2;
    cout<<"The length of the longest substring of S which contains exactly K distinct vowels
is:"<<solve(s,14,k);
    return 0;
}
```

Output-



Below is the correct working code of above question understood from gfg-

```
#include <bits/stdc++.h>
using namespace std;
```

```
#define MAX 128
bool isVowel(char x)
{
    return (x == 'a' || x == 'e' || x == 'i' ||
            x == 'o' || x == 'u');
}

int KDistinctVowel(char s[], int k)
{
    int n = strlen(s);
    int c[MAX];
    memset(c, 0, sizeof(c));
    int result = -1;

    for (int i = 0, j = -1; i < n; ++i) {

        int x = s[i];
        if (isVowel(x)) {
            if (++c[x] == 1) {
                --k;
            }
        }
        while (k < 0) {

            x = s[++j];
            if (isVowel(x)) {
                if (--c[x] == 0) {
                    ++k;
                }
            }
        }
        if (k == 0)
            result = max(result, i - j);
    }
    return result;
}
int main(void)
{
    char s[] = "artyebui";
```

```
    int k = 2;
    cout <<"the length of the longest substring of S which contains exactly K distinct
vowels."<< KDistinctVowel(s, k);
    return 0;
}
```

```
the length of the longest substring of S which contains exactly K distinct vowels.6

...Program finished with exit code 0
Press ENTER to exit console.
```

Ans 2-

```
//This question can be easily solved using the kdanes algorithm.
#include <bits/stdc++.h>
using namespace std;
int solve(int s[],int size){
    int curSum=0;
    int maxSum=INT_MIN;
    int curIndex;
    int maxIndex=0;
   for(int i=0;i<size;i++){
       curSum+=s[i];
       if(curSum>maxSum){
           maxSum=curSum;
           maxIndex=curIndex;
       }
       if(curSum<0){
           curIndex=i+1;
           curSum=0;
       }
   }
   int temp=0;
   cout<<"The subsequence which will create the maximum sum will be :\n";
   while(temp!=maxSum){
       cout<<s[maxIndex]<<" ";
       temp+=s[maxIndex];
       maxIndex++;
   }
   return maxSum;
}
int main()
{
```

```
    int s[]={5,15,-30,10,-5,40,10};
    int size=7;
    int ans=solve(s,size);
    cout<<"\nThe maximum sum will be :"<<ans<<endl;
    return 0;
}
```

Output-

```
The subsequence which will create the maximum sum will be :
10 -5 40 10
The maximum sum will be :55


...Program finished with exit code 0
Press ENTER to exit console.
```

The solution of the above problem by using the approach of creating an array to maintain the sum as told by sir in class-

```
#include <bits/stdc++.h>
using namespace std;
int solve(int arr[] ,int size){
  int dp[size];
  dp[0]=arr[0];
  for(int i=1;i<size;i++){
      if(dp[i-1]>0)dp[i]=dp[i-1]+arr[i];
      else dp[i]=arr[i];
  }
  int ans=INT_MIN;
  for(int i=0;i<size;i++){
      ans=max(dp[i],ans);
  }
  return ans;
}
int main()
{
    int s[]={5,15,-30,10,-5,40,10};
```

```
    int size=7;
    int ans=solve(s,size);
    cout<<"\nThe maximum sum will be :"<<ans<<endl;
    return 0;
}
```

Output-

```
The maximum sum will be :55


...Program finished with exit code 0
Press ENTER to exit console.
```

Ans 3-

//The below given code is written by me as per the class lecture. //Ask how to code the trace back to get what operation to be done at what index.

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
string src="vish";
int col=src.size();
string dst="vishal";
int rows=dst.size();
int arr[rows][col];
for(int j=0;j<col;j++)arr[0][j]=j;
for(int i=0;i<rows;i++)arr[i][0]=i;
for(int i=1;i<rows;i++){
    for(int j=1;j<col;j++){
        if(dst[i]==src[j])arr[i][j]=arr[i-1][j-1];
        else{
            arr[i][j]=(min(arr[i-1][j-1],min(arr[i][j-1],arr[i-1][j])))+1;
        }
    }
}
```

```
cout<<"Number of operations required will be "<< arr[rows-1][col-1]<<endl;
return 0;
}
```

Output-

```
Number of operations required will be 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Ans 4-

Code-

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int p[]={40, 20, 30, 10, 30};
  int n=5;
  int m[5][5];
  for (int i = 1; i < n; i++)m[i][i] = 0;
  for(int d=1;d<n-1;d++){
     for(int i=1;i<n-d;i++){
        int j=i+d;
        int minval=INT_MAX;
        for(int k=i;k<=j-1;k++){
           int a=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
           minval=min(minval,a);
        }
        m[i][j]=minval;
     }
  }
   cout<<"Minimum number of operations required are:"<<m[1][n-1]<<endl;
   return 0;
}
```

Output-

```
Minimum number of operations required are:26000


...Program finished with exit code 0
Press ENTER to exit console.
```

Ans 5-

Code-

```
#include <bits/stdc++.h>
using namespace std;
string solve(string s1,string s2,int row,int col){
    int dp[row+1][col+1];
    for(int j=0;j<=col;j++)dp[0][j]=0;
    for(int i=0;i<=row;i++)dp[i][0]=0;
    for(int i=1;i<=row;i++){
        for(int j=1;j<=col;j++){
            if(s1[i]==s2[j])dp[i][j]=dp[i-1][j-1]+1;
            else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
           // cout<<dp[i][j]<<" ";
        }
       // cout<<endl;
    }
    int i=row;
    int j=col;
    cout<<"The length of longest common subsequence available is:"<<dp[i][j]<<endl;
    string answer="";
    while(i>0&&j>0){
        if(s1[i-1]==s2[j-1]){
            answer+=s1[i-1];
            i--;
            j--;
```

```
        }
        else if(dp[i-1][j]>dp[i][j-1]){
            i--;
        }
        else j--;
        //cout<<answer<<endl;
    }
    reverse(answer.begin(),answer.end());
    return answer;
}
int main()
{
    //My code of trace back fails for below commented test case. This is happening
because we are not handling the tie up cases in the traceback
    //string s1 ="ACADB";
    //string s2 = "CBDB";
    string s1 ="nitin";
    string s2 = "iin";
    int row=s1.size();
    int col=s2.size();
    string ans=solve(s1,s2,row,col);
    cout<<"The longest common subsequence which is available in both the given strings
is: "<<ans<<endl;
    return 0;
}
```

Output-

```
The length of longest common subsequence available is:3
The longest common subsequence which is available in both the given strings is: iin


...Program finished with exit code 0
Press ENTER to exit console.
```

Ans 6-

This code is taken from gfg because I am not able to implement this completely by own.

```
#include<bits/stdc++.h>
using namespace std;
```

```
class NAryTree
{
	int N; // No. of nodes in Tree

	// Pointer to an array containing list of children
	list<int> *adj;

	// A function used by getMinIter(), it basically does postorder
	void getMinIterUtil(int v, int minItr[]);
public:
	NAryTree(int N); // Constructor

	// function to add a child w to v
	void addChild(int v, int w);

	// The main function to find minimum iterations
	int getMinIter();

	static int compare(const void * a, const void * b);
};

NAryTree::NAryTree(int N)
{
	this->N = N;
	adj = new list<int>[N];
}

// To add a child w to v
void NAryTree::addChild(int v, int w)
{
	adj[v].push_back(w); // Add w to v's list.
}

void NAryTree::getMinIterUtil(int u, int minItr[])
{
	minItr[u] = adj[u].size();
	int *minItrTemp = new int[minItr[u]];
	int k = 0, tmp = 0;
	// Recur for all the vertices adjacent to this vertex
	list<int>::iterator i;
	for (i = adj[u].begin(); i!= adj[u].end(); ++i)
	{
		getMinIterUtil(*i, minItr);
		minItrTemp[k++] = minItr[*i];
	}
	qsort(minItrTemp, minItr[u], sizeof (int), compare);
	for (k = 0; k < adj[u].size(); k++)
	{
		tmp = minItrTemp[k] + k + 1;
		minItr[u] = max(minItr[u], tmp);
	}
	delete[] minItrTemp;
}
```

```
// The function to do PostOrder traversal. It uses
// recursive getMinIterUtil()
int NAryTree::getMinIter()
{
	// Set minimum iteration all the vertices as zero
	int *minItr = new int[N];
	int res = -1;
	for (int i = 0; i < N; i++)
		minItr[i] = 0;

	// Start Post Order Traversal from Root
	getMinIterUtil(0, minItr);
	res = minItr[0];
	delete[] minItr;
	return res;
}

int NAryTree::compare(const void * a, const void * b)
{
		return ( *(int*)b - *(int*)a );
}

// Driver function to test above functions
int main()
{
	// TestCase 1
	NAryTree tree1(17);
	tree1.addChild(0, 1);
	tree1.addChild(0, 2);
	tree1.addChild(0, 3);
	tree1.addChild(0, 4);
	tree1.addChild(0, 5);
	tree1.addChild(0, 6);

	tree1.addChild(1, 7);
	tree1.addChild(1, 8);
	tree1.addChild(1, 9);

	tree1.addChild(4, 10);
	tree1.addChild(4, 11);

	tree1.addChild(6, 12);

	tree1.addChild(7, 13);
	tree1.addChild(7, 14);
	tree1.addChild(10, 15);
	tree1.addChild(11, 16);

	cout << "TestCase 1 - Minimum Iteration: "
		<< tree1.getMinIter() << endl;

	// TestCase 2
	NAryTree tree2(3);
	tree2.addChild(0, 1);
```

```
tree2.addChild(0, 2);
cout << "TestCase 2 - Minimum Iteration: "
        << tree2.getMinIter() << endl;

// TestCase 3
NAryTree tree3(1);
cout << "TestCase 3 - Minimum Iteration: "
        << tree3.getMinIter() << endl;

// TestCase 4
NAryTree tree4(6);
tree4.addChild(0, 1);
tree4.addChild(1, 2);
tree4.addChild(2, 3);
tree4.addChild(3, 4);
tree4.addChild(4, 5);
cout << "TestCase 4 - Minimum Iteration: "
        << tree4.getMinIter() << endl;

// TestCase 5
NAryTree tree5(6);
tree5.addChild(0, 1);
tree5.addChild(0, 2);
tree5.addChild(2, 3);
tree5.addChild(2, 4);
tree5.addChild(2, 5);
cout << "TestCase 5 - Minimum Iteration: "
        << tree5.getMinIter() << endl;

// TestCase 6
NAryTree tree6(6);
tree6.addChild(0, 1);
tree6.addChild(0, 2);
tree6.addChild(2, 3);
tree6.addChild(2, 4);
tree6.addChild(3, 5);
cout << "TestCase 6 - Minimum Iteration: "
        << tree6.getMinIter() << endl;

// TestCase 7
NAryTree tree7(14);
tree7.addChild(0, 1);
tree7.addChild(0, 2);
tree7.addChild(0, 3);
tree7.addChild(1, 4);
tree7.addChild(2, 5);
tree7.addChild(2, 6);
tree7.addChild(4, 7);
tree7.addChild(5, 8);
tree7.addChild(5, 9);
tree7.addChild(7, 10);
tree7.addChild(8, 11);
tree7.addChild(8, 12);
tree7.addChild(10, 13);
```

```
cout << "TestCase 7 - Minimum Iteration: "
            << tree7.getMinIter() << endl;

// TestCase 8
NAryTree tree8(14);
tree8.addChild(0, 1);
tree8.addChild(0, 2);
tree8.addChild(0, 3);
tree8.addChild(0, 4);
tree8.addChild(0, 5);
tree8.addChild(1, 6);
tree8.addChild(2, 7);
tree8.addChild(3, 8);
tree8.addChild(4, 9);
tree8.addChild(6, 10);
tree8.addChild(7, 11);
tree8.addChild(8, 12);
tree8.addChild(9, 13);
cout << "TestCase 8 - Minimum Iteration: "
            << tree8.getMinIter() << endl;

// TestCase 9
NAryTree tree9(25);
tree9.addChild(0, 1);
tree9.addChild(0, 2);
tree9.addChild(0, 3);
tree9.addChild(0, 4);
tree9.addChild(0, 5);
tree9.addChild(0, 6);

tree9.addChild(1, 7);
tree9.addChild(2, 8);
tree9.addChild(3, 9);
tree9.addChild(4, 10);
tree9.addChild(5, 11);
tree9.addChild(6, 12);

tree9.addChild(7, 13);
tree9.addChild(8, 14);
tree9.addChild(9, 15);
tree9.addChild(10, 16);
tree9.addChild(11, 17);
tree9.addChild(12, 18);

tree9.addChild(13, 19);
tree9.addChild(14, 20);
tree9.addChild(15, 21);
tree9.addChild(16, 22);
tree9.addChild(17, 23);
tree9.addChild(19, 24);

cout << "TestCase 9 - Minimum Iteration: "
            << tree9.getMinIter() << endl;
```

```
        return 0;
}
```

Ans 7-

This question is very hard and I am not able to implement this completely.

The pseudo code is given below-

```
MaxValue(X[0 .. 2n]):
for i ← n down to 0
Max[i, i] ← X[2k]
Min[i, i] ← X[2k]
for k ← i + 1 to n
localMax ← −∞
localMin ← ∞
for j ← i to k − 1
if X[2j + 1] = +
localMax ← max{localMax, Max[i, j] + Max[ j + 1, k]}
localMin ← min{localMin, Min[i, j] + Min[ j + 1, k]}
else
localMax ← max{localMax, Max[i, j] − Min[ j + 1, k]}
localMin ← min{localMin, Min[i, j] − Max[ j + 1, k]}
Max[i, k] ← localMax
Min[i, k] ← localMin
return Max[0, n]
```

Lab evaluation question asked by sir. So, please try this-

[Minimize the maximum difference between the heights - GeeksforGeeks]

Length of longest increasing subsequence-

```
#include <bits/stdc++.h>
using namespace std;
int solve(int arr[] ,int sizei){
  int dp[sizei];
  dp[0]=1;
  for(int i=1;i<sizei;i++){
      dp[i]=1;
```

```
      for(int j=0;j<i;j++){
          if(arr[i]>arr[j]&&dp[i]<dp[j]+1)dp[i]=dp[j]+1;
      }
  }
  int answer=0;
  for(int i=0;i<sizei;i++){
      if(answer<dp[i])answer=dp[i];
  }
  return answer;
}
int main()
{
    int s[]={10, 22, 9, 33, 21, 50, 41, 60};
    cout<<"The length of increasing subsequence is "<<solve(s,8);
    return 0;
}
```

Q-Find the length of the longest palindromic substring.

Ans-

Using recursion-

Ask why this below given code is giving error-

```
#include <iostream>

using namespace std;
int f(int i,int j,string s){
    if(i==j)return 1;
    int inner=0;
    int left=0;
    int right=0;
    if(s[i]==s[j]&&(j-i)>1){
        int temp=f(i+1,j-1,s);
        if(temp==j-i-1){
            inner=temp+2;
        }
    }
    if(j!=0)left=f(i,j-1,s);
    if(i!=s.length()-1)right=f(i+1,j,s);
    return max(inner,max(left,right));
}
```

```
int main()
{
    //Let f(i,j,s) means the length of the longest plindromic substring in the string s.
    string s="nigttin";
    cout<<"The length of the longest palindromic substring is:"<<f(0,5,s);
    return 0;
}
```

Q-Find if divisible by M using + or - between elements of the array.

Using recursion-

1-

```
#include<bits/stdc++.h>
using namespace std;
bool f(int i,int sum,int M,int arr[],int n){
    if(i==n){
        if(sum%M==0)return true;
        else return false;
    }
    //+
    bool plus=f(i+1,sum+arr[i],M,arr,n);
    //-
    bool minus=f(i+1,sum-arr[i],M,arr,n);
    return plus||minus;

}
int main(){
    int arr[] = {12,1,1};
    int M=12;
    int sum=arr[0];
    int n=3;
    if(f(1,sum,M,arr,n))cout<<"Y";
    else cout<<"No";
    return 0;
}
//12 3 1
```

2-

```
#include<bits/stdc++.h>
using namespace std;
bool f(int i,int sum,int M,int arr[]){
    if(i==0){
        int temp1=sum+arr[0];
        int temp2=arr[0]-sum;
        if(temp1%M==0||temp2%M==0)return true;
        else return false;
    }
    //+
    bool plus=f(i-1,sum+arr[i],M,arr);
    //-
    bool minus=f(i-1,sum-arr[i],M,arr);
    return plus||minus;

}
int main(){
    int arr[] = {12,1,1};
    vector<
    int M=12;
    int sum=0;
    if(f(2,sum,M,arr))cout<<"Y";
    else cout<<"No";
    return 0;
}
//12 3 1
```

Using DP memoization-

```
#include<bits/stdc++.h>
using namespace std;
bool f(int i,int sum,int M,int arr[],vector<bool>&dp){
    if(i==0){
```

```
        int temp1=sum+arr[0];
        int temp2=arr[0]-sum;
        if(temp1%M==0||temp2%M==0)return true;
        else return false;
    }
    if(dp[i]!=false)return dp[i];
    //+
    bool plus=f(i-1,sum+arr[i],M,arr,dp);
    //-
    bool minus=f(i-1,sum-arr[i],M,arr,dp);
    return dp[i]=plus||minus;
}
int main(){
    vector<bool>dp(3,false);
    int arr[] = {12,1,1};
    int M=12;
    int sum=0;
    if(f(2,sum,M,arr,dp))cout<<"Y";
    else cout<<"No";
    return 0;
}
//12 3 1
```