**MIN MAX**

MaxMinFile.java

public class MaxMinfile {

    Double max;

    Double min;

}

Main.java

import java.io.BufferedReader;

import java.io.FileReader;

import java.util.ArrayList;

class Main{

    public static void main(String args[])

    {

        BufferedReader b;

        String line;

        int size = 0;

        String Date[]=new String[25];

        Double Open[]=new Double[25];

        Double High[]=new Double[25];

        Double Low[]=new Double[25];

        Double Close[]=new Double[25];

        String heading[]=new String[6];             //To store heading name

        ArrayList<Double[]> al=new ArrayList<>();   //creating Arraylist to store Array

        try{

            int i=0;

            b=new BufferedReader(new FileReader("Stock.csv"));

            heading=b.readLine().split(",");

            //printing heading

            System.out.println("\nPrinting Stock Prices:");

            for(int j=0;j<5;j++){

                if(j==0)

                    System.out.print("\n"+heading[j]+"       ");

                else if(j==2)

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

                else

                    System.out.print(heading[j]+' ');

            }

            System.out.println();

            while((line=b.readLine())!=null){

                String a[]=line.split(",");

                Date[i]=a[0];

                Open[i]=Double.parseDouble(a[1]);

                High[i]=Double.parseDouble(a[2]);

                Low[i]=Double.parseDouble(a[3]);

                Close[i]=Double.parseDouble(a[4]);

                //printing csv File

                System.out.println(Date[i]+" "+Open[i]+" "+High[i]+" "+Low[i]+" "+Close[i]);

                i++;

            }

            System.out.println("..................................");

            //size of Array

            size=i;

            al.add(Open);

            al.add(High);

            al.add(Low);

            al.add(Close);

        }

        catch(Exception e){System.out.println(e);}

        int i=0,j=size-1;

        MaxMinfile m=new MaxMinfile();

        Main MainObj=new Main();

        for(int z=0;z<4;z++){

            System.out.println("For the "+heading[z+1]+" coloumn:");

            System.out.println("Using Brute Force:");

            MainObj.BruteForce(size,al.get(z));

            System.out.println("Using Divide and Conquer:");

            MainObj.divideAndConquer(i, j, m, al.get(z));

            System.out.println("    Minimum Stock Price: "+m.min);

            System.out.println("    Maximum Stock Price: "+m.max);

            System.out.println("..................................");

        }

    }

    //Method for BruteForce

    void BruteForce(int size, Double arr[]){

        Double max=arr[0];

        Double min=arr[0];

        for(int j=1;j<size;j++){

            if(arr[j]>max){

                max=arr[j];

            }

        }for(int j=1;j<size;j++){

            if(arr[j]<min){

                min=arr[j];

            }

        }

        System.out.println("    Minimum Stock Price: "+min);

        System.out.println("    Maximum Stock Price: "+max);

    }

    //Method for Divide and Conquer

    void divideAndConquer(int i, int j,MaxMinfile m, Double a[]){

        if(i==j){

            m.min=a[i];

            m.max=a[i];

        }

        else if(i==j-1){

            if(a[i] > a[j] ){

                m.min=a[j];

                m.max=a[i];

            }

            else{

                m.min=a[i];

                m.max=a[j];

            }

        }

        else{

            MaxMinfile m1=new MaxMinfile();

            MaxMinfile m2=new MaxMinfile();

            int mid=(i+j)/2;

            divideAndConquer(i    , mid,m1,a);

            divideAndConquer(mid+1, j  ,m2,a);

            if(m1.min > m2.min){

                m.min=m2.min;

            }

            else{

                m.min=m1.min;

            }

            if(m1.max> m2.max){

                m.max=m1.max;

            }

            else{

                m.max=m2.max;

            }

        }

    }

}

**Quick Sort**

#include<iostream>

using namespace std;

int partition(int a[],int p,int r);

void quicksort(int a[],int p,int r);

void swap(int \*a,int \*b);

void printsortedarray(int a[],int n);

int main(){

    int n=10;

    int a[n]={6,9,2,7,8,1,3,7,10,5};

    int p=0,r=n-1;

    cout<<"\nArray before Sorting:\n";

    printsortedarray(a, n);

    cout<<"\n\nArray after Sorting:\n";

    quicksort(a,p,r);

    printsortedarray(a, n);

}

void quicksort(int a[],int p,int r){

    int q;

    if(p<r){

        q=partition(a,p,r);

        quicksort(a,p,q-1);

        quicksort(a,q+1,r);

    }

}

int partition(int a[],int p,int r){

    int pivot=a[r];

    int i=p-1;

    for(int j=p ; j<=r-1 ; ++j){

        if(a[j] < pivot){

            ++i;

            swap(&a[i],&a[j]);

        }

    }

    ++i;

    swap(&a[r],&a[i]);

    return i;

}

void swap(int \*a,int \*b){

    int temp=\*a;

    \*a=\*b;

    \*b=temp;

}

void printsortedarray(int a[], int n){

    for(int i=0;i<n;++i){

        cout<<a[i]<<" ";

    }

}

**Prims**

#include <bits/stdc++.h>

using namespace std;

#define MAX 9999999

#define N 6

int main()

{

    int g[N][N] = {{0, 4, 0, 0, 0, 2},

                   {4, 0, 6, 0, 0, 3},

                   {0, 6, 0, 3, 0, 1},

                   {0, 0, 3, 0, 2, 0},

                   {0, 0, 0, 2, 0, 4},

                   {2, 3, 1, 0, 4, 0}};

    int Node\_Selected[N] = {0, 0, 0, 0, 0, 0};

    int num\_edge = 0;

    Node\_Selected[0] = 1;

    int min\_cost = 0;

    cout << "Edge : Weight\n";

    while (num\_edge < N - 1)

    {

        int min = MAX;

        int a = 0;

        int b = 0;

        for (int m = 0; m < N; ++m)

        {

            if (Node\_Selected[m])

            {

                for (int n = 0; n < N; n++)

                {

                    if ((!Node\_Selected[n]) && g[m][n])

                    {

                        if (min > g[m][n])

                        {

                            min = g[m][n];

                            a = m;

                            b = n;

                        }

                    }

                }

            }

        }

        cout << a << "-" << b << " : " << g[a][b] << endl;

        min\_cost += g[a][b];

        Node\_Selected[b] = 1;

        num\_edge += 1;

    }

    return 0;

}

**LCS**

public class Main {

    public static void main(String args[]){

        String s1=new String("AGCCCTAAGGGCTACCTAGCTT");

        String s2=new String("GACAGCCTACAAGCGTTAGCTTG");

        int size1=s1.length() +1;

        int size2=s2.length() +1;

        int [][]a=new int[size1][size2];

        String [][]b=new String[size1][size2];

        int i=0,j=0;

        for( i=1;i<size1;i++){

            for( j=1;j<size2;j++){

                if(s1.charAt(i-1)==s2.charAt(j-1)){

                    a[i][j]=a[i-1][j-1]+1;

                    b[i][j]="d";

                }

                else if(a[i-1][j] >= a[i][j-1]){

                    a[i][j]=a[i-1][j];

                    b[i][j]="u";

                }

                else{

                    a[i][j]=a[i][j-1];

                    b[i][j]="l";

                }

            }

        }

        System.out.println("\nThe Path array:\n");

        for(int z=0;z<size1;z++){

            for(int y=0;y<size2;y++){

                System.out.print(a[z][y]+"  ");

            }

            System.out.println();

        }

        System.out.println("\nThe Direction array:\n");

        for(int z=1;z<size1;z++){

            for(int y=1;y<size2;y++){

                System.out.print(b[z][y]+" ");

            }

            System.out.println();

        }

        Main obj= new Main();

        System.out.println("\nThe longest common substring is:\n");

        obj.print(i-1,j-1,a,b,s1);

    }

    void print(int i, int j, int a[][],String b[][],String s1){

        if(i==0 || j==0){

            return ;

        }

        else if(b[i][j]=="d"){

            print(i-1,j-1,a,b,s1);

            System.out.print(s1.charAt(i-1)+" ");

        }

        else if(b[i][j] == "u"){

            print(i-1,j,a,b,s1);

        }

        else{

            print(i,j-1,a,b,s1);

        }

    }

}

**Matrix Chain Multiplication**

#include<iostream>

#include<limits.h>

using namespace std;

// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n

int matrix\_chain\_multiplication(int a[], int n)

{

    int m[n][n];

    int i, j, k, L, num;

    //number of multiplications are 0(zero) when there is only one matrix

    for (i=1; i<n; i++)

        m[i][i] = 0;

    //Here L is chain length. It varies from length 2 to length n.

    for (L=2; L<n; L++)

    {

        for (i=1; i<n-L+1; i++)

        {

            j = i+L-1;

            m[i][j] = INT\_MAX;  //assigning to maximum value

            for (k=i; k<=j-1; k++)

            {

                num = m[i][k] + m[k+1][j] + a[i-1]\*a[k]\*a[j];

                if (num < m[i][j])

                {

                    m[i][j] = num;    //if number of multiplications found less that number will be updated.

                }

            }

        }

    }

    return m[1][n-1];   //returning the final answer which is M[1][n]

}

int main()

{

    int n,i;

    cout<<"Enter number of matrices\n";

    cin>>n;

    n++;

    int arr[n];

    cout<<"Enter dimensions \n\n";

    for(i=0;i<n;i++)

    {

        cout<<"Enter D"<<i<<":-";

        cin>>arr[i];

    }

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

    cout<<"Minimum number of multiplications is "<<matrix\_chain\_multiplication(arr, size);

    return 0;

}

**HUFFMAN**

#include <bits/stdc++.h>

using namespace std;

//Creating a Huffman tree node

struct MinHeapNode {

    // One of the input characters

    char data;

    // Frequency of the character

    unsigned freq;

    // Left and right child

    MinHeapNode \*left, \*right;

    MinHeapNode(char data, unsigned freq)

    {

        left = right = NULL;

        this->data = data;

        this->freq = freq;

    }

};

// Comparing two heap nodes (needed in min heap)

struct compare {

    bool operator()(MinHeapNode\* l, MinHeapNode\* r)

    {

        return (l->freq > r->freq);

    }

};

// Prints huffman codes from the root of Huffman Tree.

void printCodes(struct MinHeapNode\* root, string s)

{

    if (!root)

        return;

    if (root->data != '$')

        cout << root->data << ": " << s << "\n";

    printCodes(root->left, s + "0");

    printCodes(root->right, s + "1");

}

//Function to build Huffman Tree and print its code by traversing the Huffman Tree.

void HuffmanCodes(char data[], int freq[], int size)

{

    struct MinHeapNode \*left, \*right, \*top;

    // Create a min heap & inserts all characters of data[]

    priority\_queue<MinHeapNode\*, vector<MinHeapNode\*>, compare> minHeap;

    for (int i = 0; i < size; ++i)

        minHeap.push(new MinHeapNode(data[i], freq[i]));

    // Iterate while size of heap doesn't become 1

    while (minHeap.size() != 1) {

        // Extract the two minimum

        // freq items from min heap

        left = minHeap.top();

        minHeap.pop();

        right = minHeap.top();

        minHeap.pop();

        top = new MinHeapNode('$', left->freq + right->freq);

        top->left = left;

        top->right = right;

        minHeap.push(top);

    }

    //Calling printCodes method

    printCodes(minHeap.top(), "");

}

int main()

{

    char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };

    int freq[] = {  5, 9, 12, 13, 16, 45 };

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

    HuffmanCodes(arr, freq, size);

    return 0;

}

**HAMILTANION CYCLE**

#include <bits/stdc++.h>

using namespace std;

bool hasCycle;

bool isSafe(int v, int graph[][5], vector<int> path, int pos)

{

    // If the new vertex is adjacent to the previous vertex

    if (graph[path[pos - 1]][v] == 0)

        return false;

    // If the new vertex is already included in the path

    for (int i = 0; i < pos; i++)

        if (path[i] == v)

            return false;

    return true;

}

void FindHamCycle(int graph[][5], int pos, vector<int> path, bool visited[], int N)

{

    // If all vertices traversed in Hamiltonian Cycle

    if (pos == N)

    {

        // If there exists an edge from the last vertex to the first vertex

        if (graph[path[path.size() - 1]][path[0]] != 0)

        {

            // path will include source vertex and then print the entire path traversed

            path.push\_back(0);

            // printing the path

            for (int i = 0; i < path.size(); i++)

            {

                if(i==path.size()-1){

                    cout<<path[i];

                }

                else{

                    cout << path[i] << " -> ";

                }

            }

            cout << endl;

            // Removing the source vertex added first

            path.pop\_back();

            // Updating the hasCycle variable

            hasCycle = true;

        }

        return;

    }

    // repeat for different vertices

    for (int v = 0; v < N; v++)

    {

        // Checking if this vertex is safe to be added to the Cycle being traversed

        if (isSafe(v, graph, path, pos) && !visited[v])

        {

            path.push\_back(v);

            visited[v] = true;

            // using recursion to frame rest of the way

            FindHamCycle(graph, pos + 1, path, visited, N);

            // Removing current vertex from path and process other vertices

            visited[v] = false;

            path.pop\_back();

        }

    }

}

void hamCycle(int graph[][5], int N)

{

    hasCycle = false;

    //path vector for storing the path.

    vector<int> path;

    //selecting 0 as start vertex

    path.push\_back(0);

    // Keeps the track of the visited vertices

    bool visited[N];

    for (int i = 0; i < N; i++)

        visited[i] = false;

    //marking 0th vertex as visited

    visited[0] = true;

    FindHamCycle(graph, 1, path, visited, N);

    if (!hasCycle)

    {

        cout << "No Hamiltonian Cycle"<< "possible " << endl;

        return;

    }

}

int main(){

    int graph[][5]=

     {{0, 1, 0, 1, 0},

    {1, 0, 1, 1, 1},

    {0, 1, 0, 0, 1},

    {1, 1, 0, 0, 1},

    {0, 1, 1, 1, 0},

    };

    hamCycle(graph, 5);

    return 0;

}

**Graph Coloring**

#include <bits/stdc++.h>

using namespace std;

// Total Number of vertices in the graph

#define V 4

void printSolution(int color[]);

//This Function checks whether the edge exits or not

//and whether the color to be filled in the new vertex

//is already used by uts adjacent vertices.

bool isSafe(int v, bool graph[V][V], int color[], int c)

{

    for (int i = 0; i < V; i++)

        if (graph[v][i] && c == color[i])

            return false;

    return true;

}

// A recursive function to solve m coloring problem

bool graphColoringUtil(bool graph[V][V], int m, int color[],int v)

{

    //Base Condition: if all vertices are assigned a color, return True

    if (v == V)

        return true;

    //Consider this vertex v and try different colors

    for (int c = 1; c <= m; c++) {

        // Check if assignment of color c to v is fine

        if (isSafe(v, graph, color, c)) {

            color[v] = c;

            // recur to assign colors to rest of the vertices

            if (graphColoringUtil(graph, m, color, v + 1)

                == true)

                return true;

            // If assigning color c doesn't lead to a solution then remove it

            color[v] = 0;

        }

    }

    // If no color can be assigned to this vertex then return false

    return false;

}

//This Function returns False if one of the vertex cannot be colored.

//Elsem it print the solution and return True.

bool graphColoring(bool graph[V][V], int m)

{

    // Initialize all color values as 0.

    // This initialization is needed

    // correct functioning of isSafe()

    int color[V];

    for (int i = 0; i < V; i++)

        color[i] = 0;

    // Call graphColoringUtil() for vertex 0

    if (graphColoringUtil(graph, m, color, 0) == false) {

        cout << "Solution does not exist";

        return false;

    }

    // Print the solution

    printSolution(color);

    return true;

}

/\* A utility function to print solution \*/

void printSolution(int color[])

{

    cout << "Solution Exists:"

        << " Following are the assigned colors"

        << "\n";

    for (int i = 0; i < V; i++)

        cout << " " << color[i] << " ";

    cout << "\n";

}

// Driver code

int main()

{

    /\*

    (3)---(2)

    |    / |

    |   /  |

    |  /   |

    (0)---(1)

    \*/

    bool graph[V][V] = {

        { 0, 1, 1, 1 },

        { 1, 0, 1, 0 },

        { 1, 1, 0, 1 },

        { 1, 0, 1, 0 },

    };

    // Number of colors

    int m = 3;

    graphColoring(graph, m);

    return 0;

}

**TSP**

// CPP program to implement traveling salesman

// problem using naive approach.

#include <bits/stdc++.h>

using namespace std;

#define V 4

// implementation of traveling Salesman Problem

int travllingSalesmanProblem(int graph[][V], int s)

{

// store all vertex apart from source vertex

vector<int> vertex;

for (int i = 0; i < V; i++)

if (i != s)

vertex.push\_back(i);

// store minimum weight Hamiltonian Cycle.

int min\_path = INT\_MAX;

do {

// store current Path weight(cost)

int current\_pathweight = 0;

// compute current path weight

int k = s;

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

current\_pathweight += graph[k][vertex[i]];

k = vertex[i];

}

current\_pathweight += graph[k][s];

// update minimum

min\_path = min(min\_path, current\_pathweight);

} while (

next\_permutation(vertex.begin(), vertex.end()));

return min\_path;

}

// Driver Code

int main()

{

// matrix representation of graph

int graph[][V] = { { 0, 10, 15, 20 },

{ 10, 0, 35, 25 },

{ 15, 35, 0, 30 },

{ 20, 25, 30, 0 } };

int s = 0;

cout << travllingSalesmanProblem(graph, s) << endl;

return 0;

}

**String Editing**

import java.util.\*;

public class StringEdit {

public static void main(String[] args) {

List<String> a=new ArrayList<>();

a.add("eel");

a.add("we");

a.add("weal");

a.add("welt");

a.add("well");

a.add("weld");

a.add("wee");

a.add("wen");

a.add("rel");

a.add("wet");

a.add("tel");

a.add("wed");

a.add("gel");

a.add("web");

a.add("Del");

//wel ['eel', 'we', 'weal', 'welt', 'well', 'weld', 'wee', 'wen', 'rel', 'wet', 'tel', 'wed', 'gel', 'web', 'Del']

System.out.println("Enter a word");

Scanner sc=new Scanner(System.in);

String input=sc.nextLine();

//Dictionary res;

editString(input,a);

}

private static void editString(String input, List<String> a) {

int temp;

int len1=input.length();

ArrayList<String> label=new ArrayList<>();

ArrayList<Integer> val=new ArrayList<>();

for(int i=0;i<a.size();i++)//For arrayListhelo

{

int len2=a.get(i).length();

int res[][]=new int[input.length()+2][len2+2];

for(int j=0;j<len1+1;j++)//for y input

{

for(int k=0;k<len2+1;k++)//for x destination

{

if(k==0 && j==0)

{

res[j][k]=0;

}

else if(j==0 && k!=0)

{

res[j][k]=k;

}

else if(j!=0 && k==0)

{

res[j][k]=j;

}

else{

if(Character.compare(input.charAt(j-1), a.get(i).charAt(k-1))==0)

{

temp=Math.min(res[j-1][k]+1, res[j][k-1]+1);

res[j][k]=Math.min(res[j-1][k-1], temp);

}

else{

temp=Math.min(res[j-1][k]+1, res[j][k-1]+1);

res[j][k]=Math.min(res[j-1][k-1]+2, temp);

}

}

}

}

label.add(a.get(i));

val.add(res[len1][len2]);

}

int temp1=0;

String tempString="";

for(int p=0;p<val.size();p++)//for y input

{

for(int t=0;t<val.size()-1;t++)//for x destination

{

if(val.get(t)>val.get(t+1))

{

temp1=val.get(t);

val.set(t,val.get(t+1));

val.set(t+1,temp1);

tempString=label.get(t);

label.set(t,label.get(t+1));

label.set(t+1,tempString);

}

}

}

System.out.println("\nBest 5 matches are:");

for(int p=0;p<5;p++)

{

System.out.println(label.get(p)+" : "+val.get(p));

}

}

}