Name:NOMAN IJAZ Roll number: 21F-9513

**Programming language(s) Name:java**

1. package dal;

public class KnapSackMain

{

public static void main(String[] args) {

int profits[] = { 10, 30, 55,25,60};

int weights[] = { 16, 12, 4, 11,13 };

int maxWeight = 18;

int profitLength = profits.length;

int tableRows = 0;

int tableColoms = 0;

tableRows = profitLength + 1;

tableColoms = maxWeight + 1;

int i, w;

int table[][] = new int[profitLength + 1][maxWeight + 1];

for (i = 0; i <= profitLength; i++) {

for (w = 0; w <= maxWeight; w++) {

if (i == 0 || w == 0) {

table[i][w] = 0;

} else if (weights[i - 1] <= w) {

table[i][w] = Math.max(profits[i - 1] + table[i - 1][w - weights[i - 1]], table[i - 1][w]);

} else {

table[i][w] = table[i - 1][w];

}

}

}

for (int a = 0; a < tableRows; a++) {

for (int b = 0; b < tableColoms; b++) {

System.out.print(table[a][b] + " ");

}

System.out.println();

}

System.out.println();

System.out.println("Maximum Profit !! = " + table[profitLength][maxWeight]);

System.out.println("Profit value are as !! ");

for (int i1 = 0; i1 < profits.length; i1++) {

System.out.println (profits[i1]);

}

System.out.println();

System.out.println("weight value are as !! ");

for (int i1 = 0; i1 < weights.length; i1++) {

System.out.println (weights[i1]);

}

System.out.println();

}

}

**########### Code in C++ language**

**## Code for GRAPH THEORY problem**

#include <iostream>

#include <limits>

using namespace std;

struct Edge {

int src, dest;

double weight;

};

struct Graph {

int V, E;

struct Edge\* edge;

};

struct Graph\* createGraph(int V, int E) {

struct Graph\* graph = new Graph;

graph->V = V;

graph->E = E;

graph->edge = new Edge[E];

return graph;

}

void printPath(int parent[], int j) {

// Base Case: If j is the source vertex

if (parent[j] == -1)

return;

printPath(parent, parent[j]);

cout << " -> " << j;

}

void printArr(double dist[], int parent[], int n, int src) {

cout << "Vertex Distance from Source Path\n";

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

if (i != src) {

cout << src << " to " << i << " (" << dist[i] << ") " << src;

printPath(parent, i);

cout << endl;

}

}

}

void BellmanFord(struct Graph\* graph, int src) {

int V = graph->V;

int E = graph->E;

double\* dist = new double[V];

int\* parent = new int[V];

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

dist[i] = numeric\_limits<double>::infinity();

parent[i] = -1;

}

dist[src] = 0.0;

for (int i = 1; i <= V - 1; i++) {

for (int j = 0; j < E; j++) {

int u = graph->edge[j].src;

int v = graph->edge[j].dest;

double weight = graph->edge[j].weight;

if (dist[u] != numeric\_limits<double>::infinity() && dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

parent[v] = u;

}

}

}

for (int i = 0; i < E; i++) {

int u = graph->edge[i].src;

int v = graph->edge[i].dest;

double weight = graph->edge[i].weight;

if (dist[u] != numeric\_limits<double>::infinity() && dist[u] + weight < dist[v]) {

cout << "Graph contains negative weight cycle";

delete[] dist;

delete[] parent;

return;

}

}

printArr(dist, parent, V, src);

// Clean up allocated memory

delete[] dist;

delete[] parent;

}

int main() {

int V, E;

cout << "Enter the number of vertices: ";

cin >> V;

cout << "Enter the number of edges: ";

cin >> E;

struct Graph\* graph = createGraph(V, E);

cout << "Enter edge details (source destination weight):" << endl;

for (int i = 0; i < E; ++i) {

cout << "Enter Source for edge " << (i + 1) << ": ";

cin >> graph->edge[i].src;

cout << "Enter destination for edge " << (i + 1) << ": ";

cin >> graph->edge[i].dest;

cout << "Enter weight for edge " << (i + 1) << ": ";

cin >> graph->edge[i].weight;

}

int src;

cout << "Enter the source vertex: ";

cin >> src;

BellmanFord(graph, src);

system("pause");

return 0;

}

**## Code for ASSIGNMENT problem**

#include <iostream>

using namespace std;

void inputMatrix(double matrix[][3], int size) {

cout << "Enter the matrix (" << size << "x" << size << "):\n";

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

for (int j = 0; j < size; j++) {

cout << "Enter element at position (" << i + 1 << "," << j + 1 << "): ";

cin >> matrix[i][j];

}

}

}

void inputVector(double vector[], int size) {

cout << "Enter the vector (" << size << " elements):\n";

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

cout << "Enter element at position " << i + 1 << ": ";

cin >> vector[i];

}

}

void printMatrix(double matrix[][3], int size) {

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

for (int j = 0; j < size; j++) {

cout << matrix[i][j] << " ";

}

cout << "\n";

}

}

void printVector(double vector[], int size) {

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

cout << vector[i] << " ";

}

cout << "\n";

}

int main() {

int size = 3;

double Mat[3][3];

double MatB[3][3];

double b[3];

double C[3];

double cb[3];

inputMatrix(Mat, size);

inputMatrix(MatB, size);

inputVector(b, size);

inputVector(C, size);

inputVector(cb, size);

double B\_Inverse[3][3];

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

for (int j = 0; j < size; j++) {

B\_Inverse[i][j] = MatB[i][j];

}

}

double MatB\_Inverse\_Mul\_Mat[3][3];

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

for (int j = 0; j < size; j++) {

MatB\_Inverse\_Mul\_Mat[i][j] = 0;

for (int k = 0; k < size; k++) {

MatB\_Inverse\_Mul\_Mat[i][j] += B\_Inverse[i][k] \* Mat[k][j];

}

}

}

double Cb\_Mul\_MatB\_Inverse\_Mul\_Mat\_Minus\_C[3];

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

Cb\_Mul\_MatB\_Inverse\_Mul\_Mat\_Minus\_C[i] = 0;

for (int j = 0; j < size; j++) {

Cb\_Mul\_MatB\_Inverse\_Mul\_Mat\_Minus\_C[i] += cb[j] \* MatB\_Inverse\_Mul\_Mat[j][i];

}

Cb\_Mul\_MatB\_Inverse\_Mul\_Mat\_Minus\_C[i] -= C[i];

}

double xb[3];

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

xb[i] = 0;

for (int j = 0; j < size; j++) {

xb[i] += B\_Inverse[i][j] \* b[j];

}

}

double Cb\_Inverse\_MatB[3][3];

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

for (int j = 0; j < size; j++) {

Cb\_Inverse\_MatB[i][j] = cb[i] \* B\_Inverse[i][j];

}

}

double MatB\_Inverse\_Mul\_b[3];

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

MatB\_Inverse\_Mul\_b[i] = 0;

for (int j = 0; j < size; j++) {

MatB\_Inverse\_Mul\_b[i] += B\_Inverse[i][j] \* b[j];

}

}

double Cb\_Mul\_MatB\_Inverse\_Mul\_b[3];

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

Cb\_Mul\_MatB\_Inverse\_Mul\_b[i] = 0;

for (int j = 0; j < size; j++) {

Cb\_Mul\_MatB\_Inverse\_Mul\_b[i] += Cb\_Inverse\_MatB[i][j] \* b[j];

}

}

cout << "MatB\_Inverse\_Mul\_Mat:\n";

printMatrix(MatB\_Inverse\_Mul\_Mat, size);

cout << "Cb\_Mul\_MatB\_Inverse\_Mul\_Mat\_Minus\_C:\n";

printVector(Cb\_Mul\_MatB\_Inverse\_Mul\_Mat\_Minus\_C, size);

cout << "xb:\n";

printVector(xb, size);

cout << "Cb\_Inverse\_MatB:\n";

printMatrix(Cb\_Inverse\_MatB, size);

cout << "MatB\_Inverse\_Mul\_b:\n";

printVector(MatB\_Inverse\_Mul\_b, size);

cout << "Cb\_Mul\_MatB\_Inverse\_Mul\_b:\n";

printVector(Cb\_Mul\_MatB\_Inverse\_Mul\_b, size);

system("pause");

return 0;

}

**## Code for SIMPLEX Integer problem**

package dal;

import java.io.\*;

import java.util.\*;

public class simplex {

static float[][] matrix;

static String[] rowLabels;

static String[] basicVariablesList;

static boolean isMinimizationProblem = false;

static List<Float> zjValues = new ArrayList<>();

static List<Integer> cj = new ArrayList<>();

static boolean canPrintOutput = true;

static List<Integer> cb = new ArrayList<>();

static List<Float> xb = new ArrayList<>();

static List<String> b = new ArrayList<>();

static List<String> variables = new ArrayList<>();

static List<String> artificialVariablesList = new ArrayList<>();

static List<Float> variablesDifference = new ArrayList<>();

static int bigM = 10000;

public static void namingvariable(int row, int coloumn) {

basicVariablesList = new String[coloumn + 1];

basicVariablesList[0] = "c";

for (int x = 0; x < coloumn; x++) {

basicVariablesList[x + 1] = "s" + (x + 1);

}

rowLabels = new String[row + coloumn + 2];

rowLabels[0] = "z";

for (int x = 0; x < row; x++) {

rowLabels[x + 1] = "x" + (x + 1);

}

for (int x = 0; x < coloumn; x++) {

rowLabels[row + x + 1] = "s" + (x + 1);

}

rowLabels[row + coloumn + 1] = "b";

}

public static void diffCalculation() {

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

float val = 0;

for (int j = 0; j < matrix.length; j++) {

val += cb.get(j) \* matrix[j][i];

}

zjValues.add(val);

variablesDifference.add(zjValues.get(i) - cj.get(i));

}

}

public static boolean ifItIsNotOptimal() {

diffCalculation();

boolean status = false;

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

if (variablesDifference.get(i) < 0) {

status = true;

break;

}

}

return status;

}

public static void zeroOneEntry(int row, int minimumRow) {

float arrival = matrix[minimumRow][row];

xb.set(minimumRow, xb.get(minimumRow) / arrival);

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

matrix[minimumRow][i] = matrix[minimumRow][i] / arrival;

}

for (int i = 0; i < matrix.length; i++) {

if (i != minimumRow) {

float temporary = -matrix[i][row];

for (int j = 0; j < cj.size(); j++) {

matrix[i][j] = temporary \* matrix[minimumRow][j] + matrix[i][j];

}

xb.set(i, temporary \* xb.get(minimumRow) + xb.get(i));

}

}

}

public static void printMatrix() {

for (int i = 0; i < matrix.length; i++) {

for (int j = 0; j < matrix[i].length; j++) {

System.out.print(matrix[i][j] + "\t");

}

System.out.println();

}

}

public static void displayItteration() {

boolean notPrinted = true;

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

for (int j = 0; j < b.size(); j++) {

if (artificialVariablesList.get(i).equals(b.get(j)) && xb.get(j) > 0) {

notPrinted = false;

break;

}

}

}

if (!notPrinted) {

System.out.println("No Feasible Solution Exists");

} else {

System.out.println("Basic Feasible Solution");

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

boolean status = false;

for (int j = 0; j < b.size(); j++) {

if (variables.get(i).equals(b.get(j))) {

System.out.println(variables.get(i) + " = " + xb.get(j));

status = true;

break;

}

}

if (!status) {

System.out.println(variables.get(i) + " = " + 0);

}

}

float optimalSolution = 0;

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

optimalSolution += cb.get(i) \* xb.get(i);

}

if (!isMinimizationProblem) {

System.out.println("Optimal Value = " + optimalSolution);

} else {

System.out.println("Optimal Value = " + -optimalSolution);

}

}

}

public static void bigMOpt() {

System.out.println("Select the Type of the Problem:\n" + "Maximization\tPress 1\n" + "Minimization\tPress 2!");

Scanner inputData = new Scanner(System.in);

int pattern = inputData.nextInt();

System.out.print("Enter number of constraints: ");

int coloumns = inputData.nextInt();

System.out.print("Enter number of variables: ");

int rows = inputData.nextInt();

int[] zbigM = new int[rows];

System.out.println("Please Enter your objective function:");

for (int i = 0; i < rows; i++) {

if (pattern == 1) {

zbigM[i] = inputData.nextInt();

} else {

zbigM[i] = -inputData.nextInt();

isMinimizationProblem = true;

}

String str = "x" + (i + 1);

variables.add(str);

}

for (int i = 0; i < zbigM.length; i++) {

cj.add(zbigM[i]);

}

int index = rows;

List<List<Integer> > temp = new ArrayList<>();

int slackvariable = 1;

int artifcal = 1;

for (int j = 0; j < coloumns; j++) {

System.out.println("Please enter LHS values of your constraints " + (j + 1) + ": ");

List<Integer> left = new ArrayList<>();

for (int i = 0; i < rows; i++) {

int num = inputData.nextInt();

left.add(num);

}

System.out.println("Select Inequality: \n" + " <=\tPress 1\n" + " >=\tPress 2\n" + " =\tPress 3");

int choice = inputData.nextInt();

System.out.print("Please enter b of your constraint: ");

int bRight = inputData.nextInt();

xb.add((float) Math.abs(bRight));

for (int i = 0; i < index; i++) {

left.add(0);

}

if (bRight < 0) {

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

left.set(i, -left.get(i));

}

switch (choice) {

case 1:

left.set(index++, -1);

cj.add(0);

left.set(index++, 1);

cj.add(-bigM);

String s = "s" + (slackvariable++);

variables.add(s);

String a = "a" + (artifcal++);

variables.add(a);

break;

case 2:

left.set(index++, 1);

cj.add(0);

String s1 = "s" + (slackvariable++);

variables.add(s1);

break;

default:

left.set(index++, 1);

cj.add(-bigM);

String a1 = "a" + (artifcal++);

variables.add(a1);

break;

}

} else {

switch (choice) {

case 1:

left.set(index++, 1); // to add slack

cj.add(0);

String s = "s" + (slackvariable++);

variables.add(s);

break;

case 2:

left.set(index++, -1); // to add surplus

cj.add(0);

left.set(index++, 1);

cj.add(-bigM);

String s1 = "s" + (slackvariable++);

variables.add(s1);

String a = "a" + (artifcal++);

variables.add(a);

break;

default:

left.set(index++, 1);

cj.add(-bigM);

String a1 = "a" + (artifcal++);

variables.add(a1);

break;

}

}

temp.add(left);

}

matrix = new float[coloumns][temp.get(temp.size() - 1).size()];

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

for (int j = 0; j < temp.get(i).size(); j++) {

matrix[i][j] = temp.get(i).get(j);

}

}

for (int i = 0; i < matrix.length; i++) {

for (int j = rows; j < cj.size(); j++) {

if (matrix[i][j] == 1.0) {

b.add(variables.get(j));

cb.add(cj.get(j));

}

}

}

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

if (cj.get(i) == -bigM) {

artificialVariablesList.add(variables.get(i));

}

}

makeSolutionOptimal();

if (canPrintOutput) {

displayItteration();

}

}

public static void makeSolutionOptimal() {

int iteration = 1;

while (ifItIsNotOptimal() && iteration < 4) {

int col = getColoumn();

float minRatio = Float.MAX\_VALUE;

int min\_index = 0;

boolean state = false;

for (int j = 0; j < matrix.length; j++) {

if (matrix[j][col] > 0) {

state = true;

float rowRatio = xb.get(j) / matrix[j][col];

if (rowRatio < minRatio) {

minRatio = rowRatio;

min\_index = j;

}

}

}

if (!state) {

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

canPrintOutput = false;

break;

} else {

System.out.println("\tIteration " + iteration);

System.out.println("Entering Basic Variable is: " + variables.get(col));

System.out.println("Leaving Basic Variable is: " + b.get(min\_index));

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

boolean stateVar = false;

for (int j = 0; j < b.size(); j++) {

if (variables.get(i).equals(b.get(j))) {

System.out.println(variables.get(i) + " = " + xb.get(j));

stateVar = true;

break;

}

}

if (!stateVar) {

System.out.println(variables.get(i) + " = " + 0);

}

}

b.set(min\_index, variables.get(col));

cb.set(min\_index, cj.get(col));

zeroOneEntry(col, min\_index);

iteration++;

}

zjValues = new ArrayList<>();

variablesDifference = new ArrayList<>();

printMatrix();

}

}

public static int getColoumn() {

int col = 0;

float min = variablesDifference.get(0);

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

if (variablesDifference.get(i) < min) {

col = i;

min = variablesDifference.get(i);

}

}

return col;

}

public static void main(String[] args) {

bigMOpt();

}

}

**########### Code java language**

**## Code for Simplex decimal (BIG M )problem**

package dal;

import java.util.Scanner;

import java.util.List;

import java.util.ArrayList;

public class simplex {

static float[][] matrix;

static String[] rowLabels;

static String[] basicVariablesList;

static boolean isMinimizationProblem = false;

static List<Float> zjValues = new ArrayList<>();

static List<Float> cj = new ArrayList<>();

static boolean canPrintOutput = true;

static List<Float> cb = new ArrayList<>();

static List<Float> xb = new ArrayList<>();

static List<String> b = new ArrayList<>();

static List<String> variables = new ArrayList<>();

static List<String> artificialVariablesList = new ArrayList<>();

static List<Float> variablesDifference = new ArrayList<>();

static float bigM = 10000.0f;

public static void namingVariables(int rows, int columns) {

basicVariablesList = new String[columns + 1];

basicVariablesList[0] = "c";

for (int x = 0; x < columns; x++) {

basicVariablesList[x + 1] = "s" + (x + 1);

}

rowLabels = new String[rows + columns + 2];

rowLabels[0] = "z";

for (int x = 0; x < rows; x++) {

rowLabels[x + 1] = "x" + (x + 1);

}

for (int x = 0; x < columns; x++) {

rowLabels[rows + x + 1] = "s" + (x + 1);

}

rowLabels[rows + columns + 1] = "b";

}

public static boolean ifItIsNotOptimal() {

diffCalculation();

boolean status = false;

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

if (variablesDifference.get(i) < 0) {

status = true;

break;

}

}

return status;

}

public static void zeroOneEntry(int row, int minimumRow) {

float arrival = matrix[minimumRow][row];

xb.set(minimumRow, xb.get(minimumRow) / arrival);

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

matrix[minimumRow][i] = matrix[minimumRow][i] / arrival;

}

for (int i = 0; i < matrix.length; i++) {

if (i != minimumRow) {

float temporary = -matrix[i][row];

for (int j = 0; j < cj.size(); j++) {

matrix[i][j] = temporary \* matrix[minimumRow][j] + matrix[i][j];

}

xb.set(i, temporary \* xb.get(minimumRow) + xb.get(i));

}

}

}

public static void bigMOpt() {

System.out.println("Select the Type of the Problem:\n" + "Maximization\tPress 1\n" + "Minimization\tPress 2!");

Scanner inputData = new Scanner(System.in);

int pattern = inputData.nextInt();

System.out.print("Enter the number of constraints: ");

int columns = inputData.nextInt();

System.out.print("Enter the number of variables: ");

int rows = inputData.nextInt();

float[] zbigM = new float[rows];

System.out.println("Please Enter your objective function:");

for (int i = 0; i < rows; i++) {

if (pattern == 1) {

zbigM[i] = inputData.nextFloat();

} else {

zbigM[i] = -inputData.nextFloat();

isMinimizationProblem = true;

}

String str = "x" + (i + 1);

variables.add(str);

}

for (int i = 0; i < zbigM.length; i++) {

cj.add(zbigM[i]);

}

int index = rows;

List<List<Float>> temp = new ArrayList<>();

int slackVariable = 1;

int artificial = 1;

for (int j = 0; j < columns; j++) {

System.out.println("Please enter LHS values of your constraints " + (j + 1) + ": ");

List<Float> left = new ArrayList<>();

for (int i = 0; i < rows; i++) {

float num = inputData.nextFloat();

left.add(num);

}

System.out.println("Select Inequality: \n" + " <=\tPress 1\n" + " >=\tPress 2\n" + " =\tPress 3");

int choice = inputData.nextInt();

System.out.print("Please enter b of your constraint: ");

float bRight = inputData.nextFloat();

xb.add(Math.abs(bRight));

for (int i = 0; i < index; i++) {

left.add(0.0f);

}

if (bRight < 0) {

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

left.set(i, -left.get(i));

}

switch (choice) {

case 1:

left.set(index++, -1.0f);

cj.add(0.0f);

left.set(index++, 1.0f);

cj.add(-bigM);

String s = "s" + (slackVariable++);

variables.add(s);

String a = "a" + (artificial++);

variables.add(a);

break;

case 2:

left.set(index++, 1.0f);

cj.add(0.0f);

String s1 = "s" + (slackVariable++);

variables.add(s1);

break;

default:

left.set(index++, 1.0f);

cj.add(-bigM);

String a1 = "a" + (artificial++);

variables.add(a1);

break;

}

} else {

switch (choice) {

case 1:

left.set(index++, 1.0f);

cj.add(0.0f);

String s = "s" + (slackVariable++);

variables.add(s);

break;

case 2:

left.set(index++, -1.0f);

cj.add(0.0f);

left.set(index++, 1.0f);

cj.add(-bigM);

String s1 = "s" + (slackVariable++);

variables.add(s1);

String a = "a" + (artificial++);

variables.add(a);

break;

default:

left.set(index++, 1.0f);

cj.add(-bigM);

String a1 = "a" + (artificial++);

variables.add(a1);

break;

}

}

temp.add(left);

}

matrix = new float[columns][temp.get(temp.size() - 1).size()];

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

for (int j = 0; j < temp.get(i).size(); j++) {

matrix[i][j] = temp.get(i).get(j);

}

}

for (int i = 0; i < matrix.length; i++) {

for (int j = rows; j < cj.size(); j++) {

if (matrix[i][j] == 1.0f) {

b.add(variables.get(j));

cb.add(cj.get(j));

}

}

}

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

if (cj.get(i) == -bigM) {

artificialVariablesList.add(variables.get(i));

}

}

makeSolutionOptimal();

if (canPrintOutput) {

displayIteration();

}

}

public static void printMatrix() {

for (int i = 0; i < matrix.length; i++) {

for (int j = 0; j < matrix[i].length; j++) {

System.out.print(matrix[i][j] + "\t");

}

System.out.println();

}

}

public static void displayIteration() {

boolean notPrinted = true;

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

for (int j = 0; j < b.size(); j++) {

if (artificialVariablesList.get(i).equals(b.get(j)) && xb.get(j) > 0) {

notPrinted = false;

break;

}

}

}

if (!notPrinted) {

System.out.println("No Feasible Solution Exists");

} else {

System.out.println("Basic Feasible Solution");

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

boolean status = false;

for (int j = 0; j < b.size(); j++) {

if (variables.get(i).equals(b.get(j))) {

System.out.println(variables.get(i) + " = " + xb.get(j));

status = true;

break;

}

}

if (!status) {

System.out.println(variables.get(i) + " = " + 0);

}

}

float optimalSolution = 0;

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

optimalSolution += cb.get(i) \* xb.get(i);

}

if (!isMinimizationProblem) {

System.out.println("Optimal Value = " + optimalSolution);

} else {

System.out.println("Optimal Value = " + -optimalSolution);

}

}

}

public static void makeSolutionOptimal() {

int iteration = 1;

while (ifItIsNotOptimal() && iteration < 10) {

int col = getColoumn();

float minRatio = Float.MAX\_VALUE;

int min\_index = 0;

boolean state = false;

for (int j = 0; j < matrix.length; j++) {

if (matrix[j][col] > 0) {

state = true;

float rowRatio = xb.get(j) / matrix[j][col];

if (rowRatio < minRatio) {

minRatio = rowRatio;

min\_index = j;

}

}

}

if (!state) {

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

canPrintOutput = false;

break;

} else {

System.out.println("\tIteration " + iteration);

System.out.println("Entering Basic Variable is: " + variables.get(col));

System.out.println("Leaving Basic Variable is: " + b.get(min\_index));

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

boolean stateVar = false;

for (int j = 0; j < b.size(); j++) {

if (variables.get(i).equals(b.get(j))) {

System.out.println(variables.get(i) + " = " + xb.get(j));

stateVar = true;

break;

}

}

if (!stateVar) {

System.out.println(variables.get(i) + " = " + 0);

}

}

b.set(min\_index, variables.get(col));

cb.set(min\_index, cj.get(col));

zeroOneEntry(col, min\_index);

iteration++;

}

zjValues = new ArrayList<>();

variablesDifference = new ArrayList<>();

printMatrix();

}

}

public static int getColoumn() {

int col = 0;

float min = variablesDifference.get(0);

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

if (variablesDifference.get(i) < min) {

col = i;

min = variablesDifference.get(i);

}

}

return col;

}

public static void diffCalculation() {

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

float val = 0;

for (int j = 0; j < matrix.length; j++) {

val += cb.get(j) \* matrix[j][i];

}

zjValues.add(val);

variablesDifference.add(zjValues.get(i) - cj.get(i));

}

}

public static void main(String[] args) {

bigMOpt();

}

}

**## Code for NORTH WEST problem**

package dal;

public class NorthWest {

public static void main(String[] args) {

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

int[][] costs = {

{5, 7, 8},

{4, 4, 6},

{6, 7, 7}

};

int[] supply = {70, 30, 50};

int[] demand = {65, 42, 43};

int currentRow = 0;

int currentColumn = 0;

int totalCost = 0;

while (currentRow < costs.length && currentColumn < costs[0].length) {

if (supply[currentRow] <= demand[currentColumn]) {

int quantity = supply[currentRow];

totalCost += quantity \* costs[currentRow][currentColumn];

demand[currentColumn] -= quantity;

supply[currentRow] = 0;

System.out.println(quantity + " \* " + costs[currentRow][currentColumn] + " at (" + currentRow + ", " + currentColumn + ")");

currentRow++;

} else {

int quantity = demand[currentColumn];

totalCost += quantity \* costs[currentRow][currentColumn];

supply[currentRow] -= quantity;

demand[currentColumn] = 0;

System.out.println(quantity + " \* " + costs[currentRow][currentColumn] + " at (" + currentRow + ", " + currentColumn + ")");

currentColumn++;

}

}

System.out.println("The initial feasible basic solution is " + totalCost);

}

}

**## Code for VOGAL MTHOD problem**

package dal;

import java.util.Arrays;

public class Vogalmethod {

public static void main(String[] args) {

System.out.println("VOGEL'S APPROXIMATION METHOD");

int[][] transportationCost = {

{5, 7, 8},

{4, 4, 6},

{6, 7, 7}

};

int[] supply = {70, 30, 50};

int[] demand = {65, 42, 43};

int n = transportationCost.length;

int m = transportationCost[0].length;

int answer = 0;

int a = 1000;

while (Arrays.stream(supply).max().getAsInt() != 0 || Arrays.stream(demand).max().getAsInt() != 0) {

int[] rowDiff = new int[n];

int[] colDiff = new int[m];

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

int[] rowValues = Arrays.copyOf(transportationCost[i], m);

Arrays.sort(rowValues);

rowDiff[i] = rowValues[1] - rowValues[0];

}

for (int j = 0; j < m; j++) {

int[] colValues = new int[n];

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

colValues[i] = transportationCost[i][j];

}

Arrays.sort(colValues);

colDiff[j] = colValues[1] - colValues[0];

}

int maximum1 = Arrays.stream(rowDiff).max().getAsInt();

int maximum2 = Arrays.stream(colDiff).max().getAsInt();

if (maximum1 >= maximum2) {

for (int index = 0; index < n; index++) {

if (rowDiff[index] == maximum1) {

int minimum1 = Arrays.stream(transportationCost[index]).min().getAsInt();

for (int j = 0; j < m; j++) {

if (transportationCost[index][j] == minimum1) {

int minimum2 = Math.min(supply[index], demand[j]);

answer += minimum2 \* minimum1;

supply[index] -= minimum2;

demand[j] -= minimum2;

System.out.println("Allocate " + minimum2 + " to cell (" + index + ", " + j + ") with cost " + minimum1);

if (demand[j] == 0) {

for (int r = 0; r < n; r++) {

transportationCost[r][j] = a;

}

} else {

Arrays.fill(transportationCost[index], a);

}

break;

}

}

break;

}

}

} else {

for (int ind = 0; ind < m; ind++) {

if (colDiff[ind] == maximum2) {

int minimum1 = a;

for (int j = 0; j < n; j++) {

minimum1 = Math.min(minimum1, transportationCost[j][ind]);

}

for (int ind2 = 0; ind2 < n; ind2++) {

if (transportationCost[ind2][ind] == minimum1) {

int mini2 = Math.min(supply[ind2], demand[ind]);

answer += mini2 \* minimum1;

supply[ind2] -= mini2;

demand[ind] -= mini2;

System.out.println("Allocate " + mini2 + " to cell (" + ind2 + ", " + ind + ") with cost " + minimum1);

if (demand[ind] == 0) {

for (int r = 0; r < n; r++) {

transportationCost[r][ind] = a;

}

} else {

Arrays.fill(transportationCost[ind2], a);

}

break;

}

}

break;

}

}

}

}

System.out.println("The basic feasible solution is " + answer);

}

}

**## Code for LEAST COST problem**

package dal;

public class LeastCost {

public static void main(String[] args) {

System.out.println("LEAST COST TRANSPORTATION");

int[][] transportationCost = {

{5, 7, 8},

{4, 4, 6},

{6, 7, 7}

};

int[] supply = {70, 30, 50};

int[] demand = {65, 42, 43};

int answer = 0;

while (true) {

int minimumCost = Integer.MAX\_VALUE;

int rowWithMinCost = -1;

int colWithMinCost = -1;

for (int i = 0; i < transportationCost.length; i++) {

for (int j = 0; j < transportationCost[0].length; j++) {

if (transportationCost[i][j] < minimumCost) {

minimumCost = transportationCost[i][j];

rowWithMinCost = i;

colWithMinCost = j;

}

}

}

if (minimumCost == Integer.MAX\_VALUE) {

break;

}

int location = Math.min(supply[rowWithMinCost], demand[colWithMinCost]);

supply[rowWithMinCost] -= location;

demand[colWithMinCost] -= location;

int cost = location \* transportationCost[rowWithMinCost][colWithMinCost];

answer += cost;

System.out.println(location + " \* " + transportationCost[rowWithMinCost][colWithMinCost] + " at (" + rowWithMinCost + ", " + colWithMinCost + ")");

transportationCost[rowWithMinCost][colWithMinCost] = Integer.MAX\_VALUE;

}

System.out.println("The final feasible solution is " + answer);

}

}

**## Code for ASSIGNMENT problem**

package dal;

import java.util.\*;

public class AssignmentProblem {

public static int n;

public static int costing[][];

public static int maxingmatching;

public static int leftx[], lefty[];

public static int xy[];

public static int yx[];

public static boolean setS[], setT[];

public static int slack[];

public static int slackxvertex[];

public static int previous[];

public static void entry\_lable() {

Arrays.fill(leftx, 0);

Arrays.fill(lefty, 0);

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

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

leftx[x] = Math.max(leftx[x], costing[x][y]);

}

public static int assignmentProblem(int Arr[], int N) {

n = N;

costing = new int[n][n];

leftx = new int[n];

lefty = new int[n];

setS = new boolean[n];

setT = new boolean[n];

slack = new int[n];

slackxvertex = new int[n];

previous = new int[n];

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

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

costing[i][j] = -1 \* Arr[i \* n + j];

int ans = -1 \* assigmentproblem();

return ans;

}

public static void addvalue(int x, int previousx) {

setS[x] = true;

previous[x] = previousx;

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

if (leftx[x] + lefty[y] - costing[x][y] < slack[y]) {

slack[y] = leftx[x] + lefty[y] - costing[x][y];

slackxvertex[y] = x;

}

}

public static void printMatrix(int[][] matrix) {

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

for (int j = 0; j < n; j++) {

System.out.print(matrix[i][j] + "\t");

}

System.out.println();

}

System.out.println();

}

public static void aug() {

if (maxingmatching == n) return;

int x, y;

int q[] = new int[n], wr = 0, rd = 0;

Arrays.fill(setS, false);

Arrays.fill(setT, false);

Arrays.fill(previous, -1);

int root = -1;

for (x = 0; x < n; x++) {

if (xy[x] == -1) {

q[wr++] = root = x;

previous[x] = -2;

setS[x] = true;

break;

}

}

if (root == -1) {

return;

}

for (y = 0; y < n; y++) {

slack[y] = leftx[root] + lefty[y] - costing[root][y];

slackxvertex[y] = root;

}

while (true) {

while (rd < wr) {

x = q[rd++];

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

if (costing[x][y] == leftx[x] + lefty[y] && !setT[y]) {

if (yx[y] == -1) break;

setT[y] = true;

q[wr++] = yx[y];

addvalue(yx[y], x);

}

if (y < n)

break;

}

if (y < n)

break;

updatingvalues();

wr = rd = 0;

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

if (!setT[y] && slack[y] == 0) {

if (yx[y] == -1) {

x = slackxvertex[y];

break;

} else {

setT[y] = true;

if (!setS[yx[y]]) {

q[wr++] = yx[y];

addvalue(yx[y], slackxvertex[y]);

}

}

}

if (y < n) break;

}

if (y < n) {

maxingmatching++;

for (int cx = x, cy = y, ty; cx != -2; cx = previous[cx], cy = ty) {

ty = xy[cx];

yx[cy] = cx;

xy[cx] = cy;

}

aug();

}

}

public static void updatingvalues() {

int x, y;

int delta = 99999999;

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

if (!setT[y])

delta = Math.min(delta, slack[y]);

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

if (setS[x])

leftx[x] -= delta;

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

if (setT[y])

lefty[y] += delta;

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

if (!setT[y])

slack[y] -= delta;

}

public static int assigmentproblem() {

int ret = 0;

maxingmatching = 0;

xy = new int[n];

yx = new int[n];

Arrays.fill(xy, -1);

Arrays.fill(yx, -1);

entry\_lable();

aug();

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

ret += costing[x][xy[x]];

return ret;

}

public static void main(String[] args) {

int n = 5;

int Arr[] = {160, 130, 175, 190, 200, 135, 120, 130, 160, 175, 140, 110, 155, 170, 185, 50, 50, 80, 80, 110, 55, 35, 70, 80, 105};

AssignmentProblem ob = new AssignmentProblem();

// System.out.println("Original Cost Matrix:");

//ob.printMatrix(costing);

System.out.println("Optimal Solution:");

int result = ob.assignmentProblem(Arr, n);

System.out.println("Total Cost: " + result);

System.out.println("Optimal Assignment:");

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

System.out.println("Worker " + (char)('a' + i) + " is assigned to Job " + (xy[i] + 1) + " (Cost: " + Arr[i \* n + xy[i]] + ")");

}

}

}

######################################## END ######################################