Previous Code

Ariana Herbst

Compiled 11/4/16

2 BFS\_BellmanFord\_SignalStrength.java

3 Backtracking\_NQueens.java

5 BinarySearch\_ProfileLayout.java

7 BipartiteMatching\_DancingPartners.java

8 Combinations\_Subsets\_PrimeCombinations.java

9 DijkstraVariation\_BigTruck.java

11 Dijkstra\_GetShorty.java

13 Dijkstras.java

14 FloodFill\_LakesInBerland.java

16 FloodFill\_LeakyCeiling.java

18 FloodFill\_Rings.java

20 Geometry\_GameOfPool.java

21 Geometry\_Ternary\_Binary\_SpaceJunk

23 GreedyDP\_EpidemicInMonstropolis.java

25 HashTable\_CodeJam1B\_Digits .java

27 LineSweep\_CodeConD.java

28 MaxFlow\_BookClub.java

31 MaxFlow\_JobPostings.java

34 MaxFlow\_LeapingLizards.java

38 MinimumSpanningTree\_SupplyChain.java

40 NumberTheory\_PrimeSeive\_TheMagical3.java

42 Permutations\_RandomElections.java

43 Recursion\_CantoringAlong.java

44 Recursion\_DiskGame.java

45 Recursion\_TheEndOfTheWorld.java

46 SegmentTree\_AccountMaximumQueries.java

48 Simulation\_PyramidOfGlasses.java

49 TopologicalSorting\_DuelingPhilosophers.java

50 TravelingSalesman\_DanceRecital

51 Trees\_CountYourCousins.java

52 Trees\_KittenInATree.java

53 Tries\_RandomStrings.java

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/69

\* https://icpcarchive.ecs.baylor.edu/index.php?option=com\_onlinejudge&Itemid=8&page=show\_problem&problem=2200

\* @author Ariana Herbst

\* September 10, 2016

\*/

public class BFS\_BellmanFord\_SignalStrength {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int N = sc.nextInt();

ArrayList<ArrayList<Connection>> map = new ArrayList<ArrayList<Connection>>();

double[] path = new double[N];

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

{

map.add(new ArrayList<Connection>());

}

double[] mults = new double[N];

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

{

mults[n] = sc.nextDouble();

int K = sc.nextInt();

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

{

map.get(n).add(new Connection(sc.nextInt(), sc.nextDouble()));

}

}

path[0] = mults[0] ;

for (int a = 0; a < N - 1 ; a++)

{

for (int u = 0; u < N; u++) // u is index of parent node

{

for (Connection v : map.get(u))

{

path[v.target] = Math.max(path[v.target],

path[u] \* v.strength \* mults[v.target]);

}

}

}

System.out.printf("%.2f", path[N - 1]);

}

static class Connection

{

double strength;

int target;

public Connection(int t, double s)

{

target = t;

strength = s;

}

}

}

import java.util.\*;

public class Backtracking\_NQueens {

static int N;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

N = sc.nextInt();

fill(new State(new int[N][N], 0));

}

public static boolean fill(State st)

{

if (st.complete())

{

print(st);

return true;

}

else

{

int row = st.nextRow;

//try to find a spot for a queen

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

{

//may have found a place for a queen in this row

if (st.board[row][k] == 0)

{

//make each following spot be unavailable

State copy = new State(copy(st), st.nextRow);

copy.board[row][k] = 1;

for (int r = row + 1; r < N; r++)

{

int c = r - row;

if (k + c < N)

copy.board[r][k + c] = -1;

if (k - c >= 0)

copy.board[r][k - c] = -1;

copy.board[r][k] = -1;

}

if(fill(new State(copy.board, st.nextRow + 1)))

return true;

else

{

st.board[row][k] = 0;

}

}

}

return false;

}

}

public static int[][] copy(State st)

{

int[][] copy = new int[N][N];

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

{

int[] temp = st.board[r].clone();

copy[r] = temp;

}

return copy;

}

public static void print(State st)

{

StringBuilder sb = new StringBuilder();

for(int[] r : st.board)

{

for(int c : r)

{

if (c == 1)

{

sb.append("X");

}

else

{

sb.append(".");

}

}

sb.append("\n");

}

System.out.print(sb + "\n\n");

}

public static class State

{

int[][] board;

int nextRow;

public State(int[][] BOARD, int row)

{

board = BOARD;

nextRow = row;

}

public boolean complete()

{

return nextRow == N;

}

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/16

\* @author Ariana Herbst

\* April 14, 2016

\*/

public class BinarySearch\_ProfileLayout

{

static int P,C;

static long max;

static long[] heights;

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

P = sc.nextInt();

C = sc.nextInt();

heights = new long[P];

max = 0;

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

{

heights[i] = sc.nextInt();

max = Math.max(max, heights[i]);

}

long testMax = 10000000000L;

long testMin = 0;

System.out.print(binarySearch(heights, testMin, testMax, 0));

}

private static long binarySearch(long[] a, long fromIndex, long toIndex, long key)

{

int run = 0;

long low = fromIndex;

long high = toIndex - 1;

while (low <= high) {

run++;

long mid = (low + high) >>> 1;

long midVal = sparePixels(mid);

if (run > 100)

{

return mid + 1;

}

if (midVal < key)

low = mid - 1;

else if (midVal > key)

high = mid + 1;

else

return mid;

}

return -(low + 1);

}

public static long sparePixels(long height)

{

if (height < max)

{

return height - max;

}

int index = 0;

long col = 1;

long spare;

long temp = height;

boolean bufferNeeded = false;

while (index < heights.length)

{

if ((temp - heights[index]) >= 0 )

{

temp -= heights[index];

temp -= 10;

}

else {

temp = height;

col++;

index--;

}

index++;

}

return ((C - col)\*height) + height - temp;

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/35

\* @author Ariana Herbst

\* May 18, 2016

\*/

public class BipartiteMatching\_DancingPartners {

static int M,F,P;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

M = sc.nextInt();

F = sc.nextInt();

P = sc.nextInt();

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

for (int i = 0; i < M + F; i++) {

g.add(new ArrayList<Integer>());

}

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

g.get(sc.nextInt()).add(sc.nextInt() + M);

}

int matchings = findMaxMatching(g);

int lonelyppl = M + F - 2 \* matchings;

System.out.print(lonelyppl + "");

}

public static int findMaxMatching(ArrayList<ArrayList<Integer>> graph) {

int N = graph.size();

int matches = 0;

int[] matchedTo = new int[N];

Arrays.fill(matchedTo, -1);

for (int node = 0; node < N; node++) {

boolean[] visited = new boolean[N];

if (findMatchingNode(node, graph, matchedTo, visited)) {

matches++;

}

}

return matches;

}

public static boolean findMatchingNode(

int node,

ArrayList<ArrayList<Integer>> graph,

int[] matchedTo,

boolean[] visited

) {

if (visited[node])

return false;

visited[node] = true;

for (int neighbor : graph.get(node)) {

if (matchedTo[neighbor] == -1 ||

findMatchingNode(matchedTo[neighbor], graph, matchedTo, visited))

{

matchedTo[neighbor] = node;

return true;

}

}

return false;

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/25

\* @author Ariana Herbst

\* April 5, 2016

\*/

public class Combinations\_Subsets\_PrimeCombinations

{

static int[] factors;

static ArrayList<Integer> subset;

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

subset = new ArrayList<Integer>();

StringBuilder sb = new StringBuilder();

factors = new int[n];

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

{

factors[i] = sc.nextInt();

}

makeSubsets(factors, 0, 1);

Collections.sort(subset);

for (Integer i: subset)

{

sb.append(i + " ");

}

System.out.print(sb);

}

public static void makeSubsets(int[] arr, int ind, int multiple) {

if ( ind == arr.length) {

subset.add(multiple);

}

else {

makeSubsets(arr, ind + 1, multiple);

multiple \*= arr[ind];

makeSubsets(arr, ind + 1, multiple);

multiple /= arr[ind];

}

}

}

import java.util.\*;

/\*\*

\* https://naq16.kattis.com/problems/bigtruck

\* 2016 ICPC North American Qualifier Contest

\* @author Ariana Herbst

\* September 24, 2016

\*/

public class DijkstraVariation\_BigTruck {

static int N, M;

static ArrayList<ArrayList<Edge>> edges;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

N = sc.nextInt();

int[] items = new int[N];

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

{

items[n] = sc.nextInt();

}

M = sc.nextInt();

edges = new ArrayList<ArrayList<Edge>>();

int p;

for(p = 0; p < N; p++) { //build empty graph

ArrayList<Edge> temp = new ArrayList<Edge>();

edges.add(temp);

}

//add edges to graph

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

int s = sc.nextInt() - 1;

int e = sc.nextInt() - 1;

int l = sc.nextInt();

edges.get(s).add(new Edge( e, l, i, items[e]));

edges.get(e).add(new Edge (s, l, i, items[s]));

}

PriorityQueue<Edge> queue = new PriorityQueue<Edge>();

for(Edge e : edges.get(0)) {

e.totalItems += items[0];

queue.add(e);

}

boolean found = false;

Set<Integer> seen = new HashSet<Integer>();

int total = 0;

int toEnd = Integer.MAX\_VALUE / 4;

int maxItems = 0;

int[] leastPath = new int[N];

Arrays.fill(leastPath, Integer.MAX\_VALUE / 41);

while (!queue.isEmpty()) {

Edge e = queue.poll();

if (found && e.cost > toEnd)

{

System.out.print(toEnd + " " + maxItems);

System.exit(0);

}

if (e.cost <= leastPath[e.target]) //if we have not seen it

{

leastPath[e.target] = e.cost;

if (e.target == N - 1) { //we've reached the end

found = true;

total = e.cost;

toEnd = e.cost;

maxItems = Math.max(maxItems, e.totalItems);

}

else {

for (Edge k : edges.get(e.target)) {

queue.add(new Edge(k.target, k.cost + e.cost, p++, e.totalItems + k.totalItems));

}

}

}

}

if (!found)

System.out.print("impossible");

else

System.out.print(toEnd + " " + maxItems);

}

static class Edge implements Comparable<Edge> {

public int target;

public int cost;

public int id;

public int totalItems;

public Edge (int target, int cost, int ID, int items) {

this.target = target;

this.cost = cost;

this.id = ID;

this.totalItems = items;

}

public int compareTo(Edge other) {

int temp = cost - other.cost;

if (temp != 0)

return temp;

return id - other.id;

}

}

}

import java.util.\*;

/\*\*

\* https://open.kattis.com/problems/getshorty

\* @author Ariana Herbst

\* September 22, 2016

\*/

public class Dijkstra\_GetShorty {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while(true)

{

int N = sc.nextInt();

int M = sc.nextInt();

if (N == 0 && M == 0)

{

System.exit(0);

}

double[] best = new double[N];

ArrayList<ArrayList<Edge>> corrs = new ArrayList<ArrayList<Edge>>();

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

{

corrs.add(new ArrayList<Edge>());

}

int m;

for (m = 0; m < M; m++)

{

int a = sc.nextInt();

int b = sc.nextInt();

double d = sc.nextDouble();

corrs.get(a).add(new Edge(b, d, m));

corrs.get(b).add(new Edge(a, d, m));

}

PriorityQueue<Edge> queue = new PriorityQueue<Edge>();

for (Edge e : corrs.get(0)) {

queue.add(e); }

boolean found = false;

Set<Integer> seen = new HashSet<Integer>();

double total = 1;

while (!found)

{

Edge e = queue.poll();

if (!seen.contains(e.target))

{

seen.add(e.target);

if (e.target == N - 1)

{

found = true;

total = e.shrink;

}

else

{

for (Edge v : corrs.get(e.target))

{

queue.add(new Edge(v.target, v.shrink \* e.shrink, m++));

}

}

}

}

System.out.printf("%.4f\n", total);

}

}

static class Edge implements Comparable<Edge>

{

int target;

double shrink;

int id;

public Edge(int t, double d, int id)

{

target = t;

shrink = d;

this.id = id;

}

public int compareTo(Edge other)

{

double temp = other.shrink - shrink;

if (temp != 0)

{

if (temp > 0)

return 1;

else

return -1;

}

return Integer.compare(this.id, other.id);

}

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/22

\* @author Ariana Herbst

\* May 25, 2016

\*/

public class Dijkstras {

static int N, E, S, T;

static ArrayList<ArrayList<Edge>> edges;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

N = sc.nextInt();

E = sc.nextInt();

S = sc.nextInt();

T = sc.nextInt();

edges = new ArrayList<ArrayList<Edge>>();

int p;

for(p = 0; p < N; p++) {

ArrayList<Edge> temp = new ArrayList<Edge>();

edges.add(temp);

}

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

int s = sc.nextInt();

int e = sc.nextInt();

int l = sc.nextInt();

edges.get(s).add(new Edge( e, l, i));

edges.get(e).add(new Edge (s, l, i));

}

PriorityQueue<Edge> queue = new PriorityQueue<Edge>();

for(Edge e : edges.get(S)) {

queue.add(e);

}

boolean found = false;

Set<Integer> seen = new HashSet<Integer>();

int total = 0;

while (!found ) {

Edge e = queue.poll();

if (!seen.contains(e.x))

{

seen.add(e.x);

if (e.x == T) {

found = true;

total = e.length;

}

else {

for (Edge k : edges.get(e.x)) {

queue.add(new Edge(k.x, k.length + e.length, p++));

}

}

}

}

System.out.print(total + "");

}

static class Edge implements Comparable<Edge> {

public int length;

public int x;

public int id;

public Edge (int X, int L, int ID) {

x = X;

length = L;

id = ID;

}

public int compareTo(Edge other) {

int temp = length - other.length;

if (temp != 0)

return temp;

return id - other.id;

}

}

}

import java.util.\*;

/\*\*

\* http://codeforces.com/contest/723/problem/D

\* @author Ariana Herbst

\* October 3rd, 2016

\*/

public class FloodFill\_LakesInBerland {

static int N, M , K;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

N = sc.nextInt();

M = sc.nextInt();

K = sc.nextInt();

boolean[][] area = new boolean[N][M];

sc.nextLine();

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

{

String next = sc.next();

for (int c = 0; c < M; c++)

{

if (next.charAt(c) == '.')

area[r][c] = true;

}

}

boolean[][] copy = new boolean[N][M];

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

copy[i] = Arrays.copyOf(area[i], M);

ArrayList<Lake> lakes = new ArrayList<Lake>();

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

{

for (int c = 0; c < M; c++)

{

if (copy[r][c])

{

int size = fill(r, c, copy);

if (size != 0)

{

lakes.add(new Lake(new Point(r, c), size));

}

}

}

}

int lakesToRemove = lakes.size() - K;

Collections.sort(lakes);

int landAdded = 0;

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

{

Lake l = lakes.get(k);

landAdded += l.size;

fill(l.p.r, l.p.c, area);

}

System.out.print(landAdded);

StringBuilder sb = new StringBuilder();

for (boolean[] row : area)

{

sb.append("\n");

for (boolean b : row)

{

if (b)

sb.append("."); //if it's a lake

else

sb.append("\*");

}

}

System.out.print(sb);

}

public static int fill(int r, int c, boolean[][] arr)

{

Queue<Point> explore = new ArrayDeque<Point>();

explore.add(new Point(r, c));

int size = 0;

while (!explore.isEmpty())

{

Point p = explore.poll();

if (arr[p.r][p.c]) {

if (isBorder(p.r, p.c)) //not a lake

{

do

{

if (arr[p.r][p.c]) {

arr[p.r][p.c] = false;

for (int[] d : dirs)

{

if (isValid(p.r + d[0], p.c + d[1]))

explore.add(new Point(p.r + d[0], p.c + d[1]));

}

arr[p.r][p.c] = false;

//turn everything into land to make sure we don't check this again.

}

p = explore.poll();

} while (p != null);

return 0;

}

//if this is water, keep exploring

arr[p.r][p.c] = false;

size++;

for (int[] d : dirs)

{

if (isValid(p.r + d[0], p.c + d[1]))

explore.add(new Point(p.r + d[0], p.c + d[1]));

}

}

}

return size;

}

public static int[][] dirs = { {1, 0}, {-1, 0}, {0, 1}, {0, -1} };

public static class Lake implements Comparable<Lake>{

Point p;

int size;

public Lake (Point p, int size) { this.p = p; this.size = size; }

public int compareTo(Lake other) { return Integer.compare(this.size, other.size); }

}

public static class Point {

int r, c;

public Point (int r, int c) { this.r = r; this.c = c;}

}

public static boolean isValid(int r, int c)

{

if (r >= 0 && r < N && c >= 0 && c < M)

return true;

return false;

}

public static boolean isBorder(int r, int c)

{

if (c == 0 || c == M - 1 || r == 0 || r == N - 1)

return true;

return false;

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/30

\* @author Ariana Herbst

\* May 15, 2016

\*/

public class FloodFill\_LeakyCeiling {

static int N;

static int[][] states;

static boolean[][] visited;

static Queue<Cor> leaks = new ArrayDeque<Cor>();

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

N = sc.nextInt();

states = new int[N][N];

visited = new boolean[N][N];

sc.nextLine();

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

{

String line = sc.nextLine();

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

{

char k = line.charAt(c);

if(k == ' ')

{

states[r][c] = 10;

}

else if(k == 'X')

{

states[r][c] = -1;

}

else

{

states[r][c] = -2;

leaks.add(new Cor(r, c, 0));

}

}

}

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

boolean[] temp = new boolean[N];

Arrays.fill(temp, false);

visited[i] = temp;

}

while(!leaks.isEmpty())

{

Cor cor = leaks.remove();

spread(cor);

}

print();

}

public static void print() {

StringBuilder sb = new StringBuilder();

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

{

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

{

if(states[r][c] == -1)

{

sb.append("X");

}

else if(!visited[r][c])

sb.append(" ");

else if(states[r][c] == -2)

{

sb.append("o");

}

else if(states[r][c] == 0)

{

sb.append(" ");

}

else if(states[r][c] > 9)

{

sb.append("9");

}

else {

sb.append(states[r][c]);

}

}

sb.append("\n");

}

System.out.print(sb);

}

public static void spread(Cor cor)

{

int r = cor.x;

int c = cor.y;

if(!visited[r][c])

states[r][c] = Math.min(cor.min, states[r][c]);

visited[r][c] = true;

for (int i = 0; i < 4; i++)

{

int x = dirs[i][0] + r;

int y = dirs[i][1] + c;

Cor temp = new Cor(x,y, cor.min + 1);

if (!isBorder(temp) && !visited[x][y] && states[r][c] != -1)

{

leaks.add(temp);

}

}

}

public static boolean isBorder(Cor cor)

{

if (cor.x < 0 || cor.x >= N)

return true;

if (cor.y < 0 || cor.y >= N)

return true;

return false;

}

static class Cor

{

public int x;

public int y;

public int min;

public Cor(int X, int Y, int MIN) {

x = X;

y = Y;

min = MIN;

}

}

static int[][] dirs = new int[][] {

{ 1, 0 },

{-1, 0},

{0, 1},

{0, -1}

};

}

import java.util.\*;

/\*\*

\* https://open.kattis.com/problems/rings2

\* @author Ariana Herbst

\* October 8, 2016

\*/

public class FloodFill\_Rings {

static Queue<Point> queue;

static int[][] states;

static int N, M;

static int maxLevel = 0;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

N = sc.nextInt();

M = sc.nextInt();

queue = new ArrayDeque<Point>();

states = new int[N][M];

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

{

char[] line = sc.next().toCharArray();

for (int c = 0; c < M; c++)

{

switch(line[c]) {

case '.':

states[r][c] = 0;

queue.add(new Point(r, c, 0));

break;

default:

states[r][c] = -1;

break;

}

}

}

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

for (int c = 0; c < M; c++)

{

if (isBorder(r, c))

queue.add(new Point(r, c, 1));

}

//fill

while (!queue.isEmpty())

{

Point p = queue.poll();

if (states[p.r][p.c] <= 0) //if unvisite3d

{

if (states[p.r][p.c] != 0) {

states[p.r][p.c] = p.l;

}

//System.out.println(p.l);

//print();

maxLevel = Math.max(maxLevel, p.l);

for (int[] dir : dirs)

{

int r = p.r + dir[0];

int c = p.c + dir[1];

if (inBounds(r, c) && states[r][c] == -1)

queue.add(new Point(r, c, p.l + 1));

}

}

}

print();

}

public static void print()

{

int pad;

String wall;

if (maxLevel < 10)

{

pad = 2;

wall = "..";

}

else

{

pad = 3;

wall = "...";

}

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

{

for (int c = 0; c < M; c++)

{

int k = states[r][c];

switch(k) {

case 0:

System.out.print(wall);

break;

default:

int l = (k + "").length();

for (int i = 0; i < pad - l; i++ )

{

System.out.print(".");

}

System.out.print(k);

break;

}

}

System.out.print("\n");

}

}

public static class Point {

int r, c, l;

public Point(int row, int col, int lev) {

r = row;

c = col;

l = lev;

}

}

public static boolean inBounds(int r, int c) {

return !(r < 0 || r >= N || c < 0 || c >= M);

}

public static boolean isBorder(int r, int c ) {

return (r == 0 || r == N - 1 || c == 0 || c == M - 1 );

}

public static int[][] dirs = new int[][] {

{-1, 0},

{1, 0},

{0, -1},

{0, 1}

};

}

import java.util.\*;

import java.awt.\*;

/\*\*

\* https://pcs.spruett.me/problems/23

\* @author Ariana Herbst

\* March 31st, 2016

\*/

public class Geometry\_GameOfPool

{

static double X;

static double Y;

static double EPS = 1e-10;

static ArrayList<Double> angles;

static ArrayList<Point> points;

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

X = sc.nextDouble();

Y = sc.nextDouble();

angles = new ArrayList<Double>();

points = new ArrayList<Point>();

points.add(new Point( 0, 2));

points.add(new Point( 1, 2));

points.add(new Point( 2, 1));

points.add(new Point( 2, 0));

points.add(new Point( 1, -1));

points.add(new Point( 0, -1));

points.add(new Point( -1, 0));

points.add(new Point( -1, 1));

for(Point p : points)

{

angles.add(getAngle(p));

}

Collections.sort(angles);

for(Double d : angles)

{

System.out.print( d + "\n");

}

}

public static double getAngle(Point target)

{

double angle = Math.toDegrees(Math.atan2(target.y - Y, target.x - X));

if(angle < 0)

{

angle += 360;

}

return angle;

}

static boolean floatEq(double a, double b)

{

return Math.abs(a - b) <= EPS;

}

}

import java.util.Scanner;

/\*\*

\* https://open.kattis.com/problems/junk

\* @author Ariana Herbst and Daniel Moyer

\* October 22, 2016

\*/

public class Geometry\_Ternary\_Binary\_SpaceJunk {

static Vect s, j;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int T = sc.nextInt();

for (int t = 0; t < T; t++)

{

//create shuttle

s= new Vect(sc.nextInt(), sc.nextInt(), sc.nextInt(), sc.nextInt(),

sc.nextInt(), sc.nextInt(), sc.nextInt());

//create junk

j= new Vect(sc.nextInt(), sc.nextInt(), sc.nextInt(), sc.nextInt(),

sc.nextInt(), sc.nextInt(), sc.nextInt());

//check to see if the distance function is increasing or decreasing between 0 and .0001.

//if the distance function is increasing, we know that the objects will never touch because they

//are always moving away from each other.

//if the distance function is decreasing, we know that the objects may or may not touch.

double t0 = func(0);

double t1 = func(.0001);

if ((t1 - t0) >= -100\*Math.ulp((t1 - t0))) {

System.out.println("No collision");

continue;

}

//finding a right bound for the ternary search

//by finding a value of time for which the distance between the objects is greater

//than the starting distance between the objects.

double rightBound = 1;

while (func(rightBound) <= t0)

rightBound \*= 2;

//min gives the time at which the objects are closest together

double min = ternarySearch(0, rightBound);

//val gives the closest possible distance between the objects, i.e.,

//if val <= 0, the objects collide.

double val = func(min);

if (val >= 1000.0 \* Math.ulp(min)) {

System.out.println("No collision");

continue;

}

//We have to use a binary search to find the first value of time for which the distance

//between the two objects = 0. i.e., we have to find the first zero of our distance function.

//it's guaranteed that this value of time will be between t = 0 and t = min,

//as long as func(min) < 0.

double ans = binarySearch(0, min, 0);

System.out.println(String.format("%.3f", ans));

}

sc.close();

}

//input: value of time

//output: the distance between the shuttle and the junk at given time.

public static double func (double t)

{

double dx = Math.pow((s.x - j.x) + t \* (s.vx - j.vx), 2);

double dy = Math.pow((s.y - j.y) + t \* (s.vy - j.vy), 2);

double dz = Math.pow((s.z - j.z) + t \* (s.vz - j.vz), 2);

return Math.sqrt(dx + dy + dz) - (s.r + j.r);

}

//stores the coordinates, radius, and velocity

public static class Vect

{

int x, y, z, r, vx, vy, vz;

public Vect(int x, int y, int z, int r, int vx, int vy, int vz)

{

this.x = x; this.y = y; this.z = z; this.r = r;

this.vx = vx; this.vy = vy; this.vz = vz;

}

}

//The following code is taken from Virginia Tech ACM ICPC Handbook//

static double binarySearch(double low, double high, double y)

{

while ((high - low) > Math.max(1e-16, 10 \* Math.ulp(high))) {

double mid = (low + high) / 2.0;

double midVal = -1 \*func(mid);

if (midVal < y)

low = mid;

else

high = mid;

}

return (low + high) / 2.0;

}

static double ternarySearch(double left, double right)

{

while (true)

{

if (Math.abs(right - left) < 10 \* Math.ulp(right))

return (left + right) / 2;

double leftThird = (2 \* left + right) / 3;

double rightThird = (left + 2 \* right) / 3;

if (func(leftThird) < func(rightThird))

right = rightThird;

else

left = leftThird;

}

}

}

import java.util.\*;

/\*\*

\* http://codeforces.com/contest/733/problem/C

\* @author Ariana Herbst

\* November 1, 2016

\*/

public class GreedyDP\_EpidemicInMonstropolis {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

ArrayList<Long> in = new ArrayList<Long>();

ArrayList<Long> out = new ArrayList<Long>();

int N = sc.nextInt();

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

in.add(sc.nextLong());

}

int K = sc.nextInt();

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

out.add(sc.nextLong());

}

int i = in.size() - 1;

int o = out.size() - 1;

long sum = 0;

StringBuilder ans = new StringBuilder();

while (i >= 0 && o >= 0)

{

while (sum < out.get(o) && i >= 0)

{

sum += in.get(i);

i--;

}

i++;

if (sum > out.get(o))

{

System.out.println("NO");

System.exit(0);

}

if (sum == out.get(o))

{

int iterations = in.size() - i - 1;

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

{

int bigIndex = -1;

long bigNum = -1;

for (int k = in.size() - 2; k >= i; k--) //find eating monster

{

if (in.get(k) > in.get(k + 1) && in.get(k) > bigNum)

{

bigIndex = k;

bigNum = in.get(k);

}

else if (in.get(k) < in.get(k + 1) && in.get(k + 1) > bigNum)

{

bigIndex = k + 1;

bigNum = in.get(k + 1);

}

}

if (bigIndex > i && bigNum > in.get(bigIndex - 1)) //eat left

{

ans.append((bigIndex + 1) + " L\n");

in.set(bigIndex - 1, bigNum + in.get(bigIndex - 1));

in.remove(bigIndex);

}

else if (bigIndex < in.size() - 1 && bigNum > in.get(bigIndex + 1)) //eat right

{

ans.append((bigIndex + 1) + " R\n");

in.set(bigIndex, bigNum + in.get(bigIndex + 1));

in.remove(bigIndex + 1);

}

else if (bigIndex == -1)

{

System.out.println("NO");

System.exit(0);

}

}

long a = in.remove(i);

long b = out.remove(o);

if (a != b)

{

System.out.println("NO");

System.exit(0);

}

sum = 0;

o--;

i--;

}

}

if (in.size() == 0 && out.size() == 0) {

System.out.println("YES");

System.out.println(ans);

}

else

System.out.println("NO");

}

}

import java.util.\*;

/\*\*

\* https://code.google.com/codejam/contest/dashboard?c=11254486

\* @author Ariana Herbst

\* April 30, 2016

\*/

public class HashTable\_CodeJam1B\_Digits {

static int N;

public static List<String> in;

static HashMap<Integer, Integer> nums;

static Map<Character, Integer> let;

public static void main(String[] args) {

StringBuilder output = new StringBuilder();

Scanner sc = new Scanner(System.in);

N = sc.nextInt();

in = new ArrayList();

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

{

in.add(sc.next());

}

//decipher

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

let = new HashMap();

for (int j = 0; j < in.get(i).length(); j++) {

Character temp = in.get(i).charAt(j);

Integer val = let.get(temp);

if ( val != null) {

let.put(temp, val + 1);

} else {

let.put(temp, 1);

}

}

nums = new HashMap<Integer, Integer>();

nums.put(0, getNums('Z', "ZERO"));

nums.put(2, getNums('W', "TWO"));

nums.put(6, getNums('X', "SIX"));

nums.put(8, getNums('G', "EIGHT"));

nums.put(3, getNums('H', "THREE"));

nums.put(7, getNums('S', "SEVEN"));

nums.put(5, getNums('V', "FIVE"));

nums.put(4, getNums('F', "FOUR"));

nums.put(1, getNums('O', "ONE"));

nums.put(9, getNums('I', "NINE"));

StringBuilder testCase = new StringBuilder();

for(int k = 0; k < 10; k++)

{

Integer temp = nums.get(k);

int freq;

if (temp == null)

{

freq = 0;

}

else {

freq = temp;

}

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

{

testCase.append("" + k);

}

}

output.append("Case #" + (i + 1) + ": " + testCase + "\n");

//System.out.print(output);

let.clear();

}

String printme = output.substring(0, output.length() - 1);

System.out.print(printme);

}

public static int getNums(char c, String word)

{

Integer freq = let.get(c);

if ((freq != null) && !(freq.equals(0))) {

//System.out.print( c );

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

{

//System.out.print(word + " " + i);

int t = let.get(word.charAt(i));

let.put(word.charAt(i), t - freq);

}

return freq;

}

return 0;

}

}

import java.util.\*;

/\*\*

\* Bloomberg Codecon 2016

\* Given a list of event starting and ending times, what is

\* the maximum number of events occurring at any time?

\* Input: the first line will have an integer N for the number

\* of events listed. The following N lines will list the starting

\* and ending times of the i-th event with the format

\* [start time] | [end time]

\* Output: The maximum number of events occurring at one time.

\* @author Ariana Herbst

\* September 22, 2016

\*/

public class LineSweep\_CodeConD {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int N = sc.nextInt();

ArrayList<Event> events = new ArrayList<Event>();

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

{

String next = sc.next();

next = next.substring(next.indexOf("|") + 1);

String s1 = next.substring(0, next.indexOf("|"));

String s2 = next.substring(next.indexOf("|"));

s2 = s2.substring(1);

events.add(new Event(Integer.parseInt(s1), 1));

events.add(new Event(Integer.parseInt(s2), 0));

}

Collections.sort(events);

int c = 0;

int m = 0;

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

{

Event e = events.get(i);

if (e.state == 0)

{

c--;

}

else

{

c++;

}

m = Math.max(m, c);

}

System.out.print(m);

}

static class Event implements Comparable<Event>

{

int time;

int state;

public Event(int t, int s)

{

time = t;

state = s;

}

public int compareTo(Event other)

{

int c = Integer.compare(time, other.time);

if (c != 0)

{

return c;

}

return Integer.compare(state, other.state);

}

}

}

import java.util.\*;

/\*\*

\* Southwestern Europe Regional Contest (SWERC) 2014

\* https://pcs.spruett.me/problems/120

\* @author Ariana Herbst

\* October 12, 2016

\*/

public class MaxFlow\_BookClub {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int N = sc.nextInt();

int M = sc.nextInt();

FordFulkerson ff = new FordFulkerson(N \* 2 + 2);

int src = N \* 2;

int snk = N \* 2 + 1;

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

{

ff.link(ff.nodes.get(src), ff.nodes.get(n), 1);

ff.link(ff.nodes.get(n + N), ff.nodes.get(snk), 1);

}

for (int m = 0; m < M; m++)

{

int u = sc.nextInt();

int v = sc.nextInt() + N;

ff.link(ff.nodes.get(u), ff.nodes.get(v), 1);

}

int matches = ff.getMaxFlow(ff.nodes.get(src), ff.nodes.get(snk));

if (matches == N)

System.out.print("YES");

else

System.out.print("NO");

}

//The following code is taken from Virginia Tech ACM ICPC Handbook//

public static class Node {

private Node() { }

List<Edge> edges = new ArrayList<Edge>();

int index;

}

public static class Edge {

boolean forward; //true: edge is in original graph

//false: edge is a backward edge

Node from, to; //nodes connected

int flow;

final int capacity;

Edge dual; //reference to this edge's dual

long cost; //only used for MinCost.

protected Edge(Node s, Node d, int c, boolean f)

{

forward = f;

from = s;

to = d;

capacity = c;

}

int remaining() { return capacity - flow; }

void addFlow(int amount) {

flow += amount;

dual.flow -= amount;

}

}

public static abstract class MaxFlowSolver {

List<Node> nodes = new ArrayList<Node>();

public void link(Node n1, Node n2, int capacity) {

Edge e12 = new Edge(n1, n2, capacity, true);

Edge e21 = new Edge(n2, n1, 0, false);

e12.dual = e21;

e21.dual = e12;

n1.edges.add(e12);

n2.edges.add(e21);

}

public void link(Node n1, Node n2, int capacity, long cost)

{

Edge e12 = new Edge(n1, n2, capacity, true);

Edge e21 = new Edge(n2, n1, 0, false);

e12.dual = e21;

e21.dual = e12;

n1.edges.add(e12);

n2.edges.add(e21);

e12.cost = cost;

e21.cost = -cost;

}

void link(int n1, int n2, int capacity)

{

link(nodes.get(n1), nodes.get(n2), capacity);

}

protected MaxFlowSolver(int n) {

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

addNode();

}

public abstract int getMaxFlow(Node src, Node snk);

public Node addNode() {

Node n = new Node();

n.index = nodes.size();

nodes.add(n);

return n;

}

List<Edge> getMinCut(Node src) {

Queue<Node> bfs = new ArrayDeque<Node>();

Set<Node> visited = new HashSet<Node>();

bfs.offer(src);

visited.add(src);

while(!bfs.isEmpty()) {

Node u = bfs.poll();

for (Edge e : u.edges) {

if (e.remaining() > 0 && !visited.contains(e.to)) {

visited.add(e.to);

bfs.offer(e.to);

}

}

}

List<Edge> minCut = new ArrayList<Edge>();

for (Node s : visited) {

for (Edge e : s.edges)

if (e.forward && !visited.contains(e.to))

minCut.add(e);

}

return minCut;

}

}

static class FordFulkerson extends MaxFlowSolver

{

FordFulkerson () { this(0); }

FordFulkerson (int n) { super(n); }

@Override

public int getMaxFlow(Node src, Node snk) {

int total = 0;

for (;;) {

//find an augmenting path and record its edges in prev

Edge [] prev = new Edge[nodes.size()];

int addedFlow = findAugmentingPath(src, snk, prev);

if (addedFlow == 0) break;

total += addedFlow;

//Go back along the path, and for each, move the

//added flow from the edge to its dual.

for (Edge edge = prev[snk.index];

edge != null;

edge = prev[edge.dual.to.index])

{

edge.addFlow(addedFlow);

}

}

return total;

}

int findAugmentingPath(Node src, Node snk, Edge [] from) {

Deque<Node> queue = new ArrayDeque<Node>();

queue.offer(src);

int N = nodes.size();

int [] minCapacity = new int[N];

boolean [] visited = new boolean[N];

visited[src.index] = true;

Arrays.fill(minCapacity, Integer.MAX\_VALUE);

while (queue.size() > 0) {

Node node = queue.poll();

if (node == snk)

return minCapacity[snk.index];

for (Edge edge : node.edges) {

Node dest = edge.to;

if (edge.remaining() > 0 && !visited[dest.index]) {

visited[dest.index] = true;

from[dest.index] = edge;

minCapacity[dest.index] = Math.min (minCapacity[node.index], edge.remaining()); ///Math.min???

if (dest == snk)

return minCapacity[snk.index];

queue.push(dest);

}

}

}

return 0; //no push

}

}

}

import java.util.\*;

/\*\*

\* 2013 University of Chicago Invitational Programming Contest

\* https://pcs.spruett.me/problems/122

\* @author Ariana Herbst

\* October 28, 2016

\*/

public class MaxFlow\_JobPostings {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while(true)

{

int N = sc.nextInt(); // # postings

int M = sc.nextInt(); // # students

if (N == 0 && M == 0)

System.exit(0);

int[][] sat = new int[3][4];

int k = -12;

for (int year = 2; year >= 0; year--)

for (int choice = 0; choice < 4; choice++) {

sat[year][choice] = k; //satisfaction matrix with negative values of those given

k++;

}

//We look for the minimum cost and maximum flow of this graph, where

//cost is (-1 \* satisfaction) and flow are jobs.

MinCostMaxFlowSolver ek = new EdmondsKarp();

Node[] students = new Node[M];

Node[] postings = new Node[N];

Node src = ek.addNode();

Node snk = ek.addNode();

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

{

postings[n] = ek.addNode();

ek.link(postings[n], snk, sc.nextInt(), 0);

}

for (int m = 0; m < M; m++)

{

students[m] = ek.addNode();

ek.link(src, students[m], 1);

int year = sc.nextInt() - 1;

for (int c = 0; c < 4; c++)

ek.link(students[m], postings[sc.nextInt()], 1, sat[year][c]);

}

long[] ans = ek.getMinCostMaxFlow(src, snk);

System.out.println(-1 \* ans[1]);

}

}

//The following code is taken from Virginia Tech ACM ICPC Handbook//

static abstract class MinCostMaxFlowSolver extends MaxFlowSolver {

//returns [maxflow, mincost]

abstract long [] getMinCostMaxFlow(Node src, Node snk);

//unavoidable boiler plate

MinCostMaxFlowSolver() { this(0); }

MinCostMaxFlowSolver(int n) { super(n); }

}

static class EdmondsKarp extends MinCostMaxFlowSolver

{

EdmondsKarp () { this(0); }

/\* Create a graph with n nodes. \*/

EdmondsKarp (int n) { super(n); }

long minCost;

@Override

public long [] getMinCostMaxFlow(Node src, Node snk) {

long maxflow = getMaxFlow(src, snk);

return new long [] { maxflow, minCost };

}

static final long INF = Long.MAX\_VALUE/4;

/\*\*

\* Maximize the flow, and simultaneously minimize its cost.

\* Code taken from judge solution to Chicago 2013/Job Postings

\* http://serjudging.vanb.org/wp-content/uploads/jobpostings\_artur.java

\*/

@Override

public long getMaxFlow(Node src, Node snk) {

final int n = nodes.size();

final int source = src.index;

final int sink = snk.index;

long flow = 0;

long cost = 0;

long[] potential = new long[n]; // allows Dijkstra to work with negative edge weights

while (true) {

Edge[] parent = new Edge[n]; // used to store an augmenting path

long[] dist = new long[n]; // minimal cost to vertex

Arrays.fill(dist, INF);

dist[source] = 0;

// Dijkstra on cost

PriorityQueue<Item> que = new PriorityQueue<Item>();

que.add(new Item(0, source));

while (!que.isEmpty()) {

Item item = que.poll();

if (item.dist != dist[item.v])

continue;

for (Edge e : nodes.get(item.v).edges) {

long temp = dist[item.v] + e.cost + potential[item.v] - potential[e.to.index];

// if can push some flow, and new cost is cheaper than push

if (e.capacity > e.flow && dist[e.to.index] > temp) {

dist[e.to.index] = temp;

parent[e.to.index] = e;

que.add(new Item(temp, e.to.index));

}

}

}

// couldn't reach sink

if (parent[sink] == null)

break;

// update potentials for Dijkstra

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

if (parent[i] != null)

potential[i] += dist[i];

// maximum flow that can be pushed

long augFlow = Long.MAX\_VALUE;

for (int i = sink; i != source; i = parent[i].from.index)

augFlow = Math.min(augFlow, parent[i].capacity - parent[i].flow);

// push the flow

for (int i = sink; i != source; i = parent[i].from.index) {

Edge e = parent[i];

e.addFlow(augFlow);

cost += augFlow \* e.cost;

}

flow += augFlow;

}

minCost = cost;

return flow;

}

static class Item implements Comparable<Item> {

long dist;

int v;

public Item(long dist, int v) {

this.dist = dist;

this.v = v;

}

public int compareTo(Item that) {

return Long.compare(this.dist, that.dist);

}

}

}

public static class Node {

private Node() {}

List<Edge> edges =new ArrayList<Edge>();

int index;

}

public static class Edge

{

boolean forward;

Node from, to;

int flow;

final long capacity;

Edge dual;

long cost;

protected Edge(Node s, Node d, long c, boolean f)

{

forward = f;

from = s;

to = d;

capacity = c;

}

long remaining() { return capacity - flow; }

void addFlow(long augFlow) {

flow += augFlow;

dual.flow -= augFlow;

}

}

public static abstract class MaxFlowSolver {

List<Node> nodes = new ArrayList<Node>();

public void link(Node n1, Node n2, int capacity)

{

Edge e12 = new Edge(n1, n2, capacity, true);

Edge e21 = new Edge(n2, n1, 0, false);

e12.dual = e21;

e21.dual = e12;

n1.edges.add(e12);

n2.edges.add(e21);

}

public void link(Node n1, Node n2, int capacity, long cost) {

Edge e12 = new Edge(n1, n2, capacity, true);

Edge e21 = new Edge(n2, n1, 0, false);

e12.dual = e21;

e21.dual = e12;

n1.edges.add(e12);

n2.edges.add(e21);

e12.cost = cost;

e21.cost = -cost;

}

void link(int n1, int n2, int capacity)

{

link(nodes.get(n1), nodes.get(n2), capacity);

}

protected MaxFlowSolver(int n) {

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

addNode();

}

protected MaxFlowSolver() { this(0); }

public abstract long getMaxFlow(Node src, Node snk);

public Node addNode() {

Node n = new Node();

n.index = nodes.size();

nodes.add(n);

return n;

}

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/119

\* 2005 ICPC Mid Central

\* @author Ariana Herbst

\* October 15, 2016

\*/

public class MaxFlow\_LeapingLizards {

public static int T, R, D, C;

public static ArrayList<Integer[]> dirs;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

T = sc.nextInt();

for (int t = 1; t <= T; t++)

{

R = sc.nextInt();

D = sc.nextInt();

dirs = createDirs(D);

MaxFlowSolver maxFlow = new FordFulkerson();

sc.nextLine();

char[] chars = sc.nextLine().toCharArray();

C = chars.length;

Node[][] left = new Node[R][C];

Node[][] right = new Node[R][C];

for (int r = 0; r < R; r++) //adds each node weight

{

for (int c = 0; c < C; c++)

{

left[r][c] = maxFlow.addNode();

right[r][c] = maxFlow.addNode();

maxFlow.link(left[r][c], right[r][c], ((int)chars[c] - '0'));

}

if (r != R - 1)

chars = sc.nextLine().toCharArray();

}

Node src = maxFlow.addNode(); //source node

Node snk = maxFlow.addNode(); //sink node

//connects adjacent nodes. If adjacent node is out of bounds,

//connects this node to snk

for (int r = 0; r < R; r++)

{

for (int c = 0; c < C; c++)

{

//to sink

boolean toSink = false;

for (Integer[] dir : dirs)

{

int i = r + dir[0];

int j = c + dir[1];

if (inBounds(i, j)) {

maxFlow.link(right[r][c], left[i][j], Integer.MAX\_VALUE);

}

else if (!toSink) {

maxFlow.link(right[r][c], snk, Integer.MAX\_VALUE);

toSink = true;

}

}

}

}

int totalLizards = 0;

for (int r = 0; r < R; r++)

{

chars = sc.nextLine().toCharArray();

for (int c = 0; c < C; c++)

{

if (chars[c] == 'L') {

maxFlow.link(src, left[r][c], 1);

totalLizards++;

}

}

}

int happyLizards = maxFlow.getMaxFlow(src, snk);

int sadLizards = totalLizards - happyLizards;

System.out.print("Case #" + t + ": ");

if (sadLizards >= 2)

System.out.println(sadLizards + " lizards were left behind.");

else if (sadLizards == 1)

System.out.println(sadLizards + " lizard was left behind.");

else if (sadLizards == 0)

System.out.println("no lizard was left behind.");

}

}

public static boolean inBounds(int r, int c) {

return ( r >= 0 && r < R && c >= 0 && c < C);

}

public static ArrayList<Integer[]> createDirs(int D)

{

ArrayList<Integer[]> temp = new ArrayList<Integer[]>();

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

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

if ((i \* i) + (j \* j) <= (D \* D) && !(i == 0 && j == 0)) {

temp.add(new Integer[] { i, j });

if (i != 0)

temp.add(new Integer[] {i \* -1, j});

if (j != 0)

temp.add(new Integer[] {i, j \* -1});

if (i != 0 && j != 0)

temp.add(new Integer[] {i \* -1, j \* -1});

}

}

}

return temp;

}

//The following code is taken from Virginia Tech ACM ICPC Handbook//

public static class Node {

private Node() { }

List<Edge> edges = new ArrayList<Edge>();

int index;

}

public static class Edge {

boolean forward; //true: edge is in original graph

//false: edge is a backward edge

Node from, to; //nodes connected

int flow;

final int capacity;

Edge dual; //reference to this edge's dual

long cost; //only used for MinCost.

protected Edge(Node s, Node d, int c, boolean f)

{

forward = f;

from = s;

to = d;

capacity = c;

}

int remaining() { return capacity - flow; }

void addFlow(int amount) {

flow += amount;

dual.flow -= amount;

}

}

public static abstract class MaxFlowSolver {

List<Node> nodes = new ArrayList<Node>();

public void link(Node n1, Node n2, int capacity) {

Edge e12 = new Edge(n1, n2, capacity, true);

Edge e21 = new Edge(n2, n1, 0, false);

e12.dual = e21;

e21.dual = e12;

n1.edges.add(e12);

n2.edges.add(e21);

}

public void link(Node n1, Node n2, int capacity, long cost)

{

Edge e12 = new Edge(n1, n2, capacity, true);

Edge e21 = new Edge(n2, n1, 0, false);

e12.dual = e21;

e21.dual = e12;

n1.edges.add(e12);

n2.edges.add(e21);

e12.cost = cost;

e21.cost = -cost;

}

void link(int n1, int n2, int capacity)

{

link(nodes.get(n1), nodes.get(n2), capacity);

}

protected MaxFlowSolver(int n) {

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

addNode();

}

public abstract int getMaxFlow(Node src, Node snk);

public Node addNode() {

Node n = new Node();

n.index = nodes.size();

nodes.add(n);

return n;

}

List<Edge> getMinCut(Node src) {

Queue<Node> bfs = new ArrayDeque<Node>();

Set<Node> visited = new HashSet<Node>();

bfs.offer(src);

visited.add(src);

while(!bfs.isEmpty()) {

Node u = bfs.poll();

for (Edge e : u.edges) {

if (e.remaining() > 0 && !visited.contains(e.to)) {

visited.add(e.to);

bfs.offer(e.to);

}

}

}

List<Edge> minCut = new ArrayList<Edge>();

for (Node s : visited) {

for (Edge e : s.edges)

if (e.forward && !visited.contains(e.to))

minCut.add(e);

}

return minCut;

}

}

static class FordFulkerson extends MaxFlowSolver

{

FordFulkerson () { this(0); }

FordFulkerson (int n) { super(n); }

@Override

public int getMaxFlow(Node src, Node snk) {

int total = 0;

for (;;) {

//find an augmenting path and record its edges in prev

Edge [] prev = new Edge[nodes.size()];

int addedFlow = findAugmentingPath(src, snk, prev);

if (addedFlow == 0) break;

total += addedFlow;

//Go back along the path, and for each, move the

//added flow from the edge to its dual.

for (Edge edge = prev[snk.index];

edge != null;

edge = prev[edge.dual.to.index])

{

edge.addFlow(addedFlow);

}

}

return total;

}

int findAugmentingPath(Node src, Node snk, Edge [] from) {

Deque<Node> queue = new ArrayDeque<Node>();

queue.offer(src);

int N = nodes.size();

int [] minCapacity = new int[N];

boolean [] visited = new boolean[N];

visited[src.index] = true;

Arrays.fill(minCapacity, Integer.MAX\_VALUE);

while (queue.size() > 0) {

Node node = queue.poll();

if (node == snk)

return minCapacity[snk.index];

for (Edge edge : node.edges) {

Node dest = edge.to;

if (edge.remaining() > 0 && !visited[dest.index]) {

visited[dest.index] = true;

from[dest.index] = edge;

minCapacity[dest.index] = Math.min (minCapacity[node.index], edge.remaining()); ///Math.min???

if (dest == snk)

return minCapacity[snk.index];

queue.push(dest);

}

}

}

return 0; //no push

}

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/8

\* @author Ariana Herbst

\* March 17, 2016

\*/

public class MinimumSpanningTree\_SupplyChain {

static int N;

static int M;

static List<Edge> l;

static Set minEdges;

static DisjointSets djs;

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

N = sc.nextInt();

M = sc.nextInt();

l = new ArrayList<Edge>();

minEdges = new HashSet<Edge>();

DisjointSets djs = new DisjointSets(N);

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

{

l.add(new Edge(sc.nextInt(), sc.nextInt(), sc.nextInt()));

}

Collections.sort(l);

for ( Edge e : l)

{

if ( djs.find(e.p1) != djs.find(e.p2) )

{

minEdges.add(e);

djs.union(e.p1, e.p2);

}

}

Iterator<Edge> it = minEdges.iterator();

int totalCost = 0;

while (it.hasNext())

{

totalCost += it.next().cost;

}

if (djs.N == 1)

System.out.print(totalCost + "");

else

System.out.print("not possible");

}

public static class Edge implements Comparable<Edge> {

int p1, p2;

int cost;

public Edge(int a, int b, int cost)

{

p1 = a;

p2 = b;

this.cost = cost;

}

public int compareTo(Edge other)

{

return Integer.compare(cost, other.cost);

}

}

public static class DisjointSets {

int N; // The number of disjoint items

int[] depth; // Depth of the set if < 0, else parent

int[] parent;

public DisjointSets(int numElements) {

this.N = numElements;

depth = new int[N];

parent = new int[N];

Arrays.fill(depth, 1);

Arrays.fill(parent, -1);

}

public boolean union(int a, int b) {

int root1 = find(a);

int root2 = find(b);

if (root1 == root2)

return false;

if (depth[root2] < depth[root1]) {

parent[root2] = root1;

} else {

if (depth[root2] == depth[root1]) {

depth[root2]++;

}

parent[root1] = root2;

}

N--;

return true;

}

public int find(int a) {

if (parent[a] < 0) return a;

parent[a] = find(parent[a]);

return parent[a];

}

}

}

import java.util.\*;

import java.io.\*;

/\*\*

\* https://open.kattis.com/problems/magical3

\* @author Ariana Herbst and Daniel Moyer

\* October 22, 2016

\*/

public class NumberTheory\_PrimeSeive\_TheMagical3 {

private static List<Integer> primes;

public static void main(String[] args) throws NumberFormatException, IOException {

BufferedReader input = new BufferedReader(new InputStreamReader(System.in));

StringBuilder output = new StringBuilder();

String nextLine;

String newLine = System.lineSeparator();

PrimeSieve ps = new PrimeSieve(46350);

primes = ps.getPrimeNumbers();

while ((nextLine = input.readLine()) != null)

{

int N = Integer.parseInt(nextLine);

if (N == 0)

break;

else if (N == 3) {

output.append(4);

output.append(newLine);

continue;

}

N -= 3;

int k = smallestFactor(N);

if (k < 4)

output.append("No such base");

else

output.append(k);

output.append(newLine);

}

System.out.print(output.toString());

}

private static int smallestFactor(int k)

{

int i = 4;

for (i = 4; i <= 9; i++) {

if (k % i == 0)

return i;

}

for (i = 0; i < primes.size() && primes.get(i)\*primes.get(i) <= k; i++) {

if (k % primes.get(i) == 0)

return primes.get(i);

}

if (k % 3 == 0 && (k / 3 ) >= 4) {

return k / 3;

}

if (k % 2 == 0 && (k / 2) >= 4) {

return k / 2;

}

return k;

}

//The following code is taken from Virginia Tech ACM ICPC Handbook//

private static class PrimeSieve {

private boolean[] isComposite;

private int max;

private PrimeSieve(int max) {

this.max = max;

isComposite = new boolean[(max + 1) >> 1];

isComposite[0] = true;

for (int i = 3; i \* i <= max; i += 2) {

for (int j = i \* i; j <= max; j += i << 1) {

isComposite[j >> 1] = true;

}

}

}

private boolean isPrime(int num) {

boolean result = false;

if ((num & 1) == 0) {

result = (num == 2);

return result;

} else {

return !isComposite[num >> 1];

}

}

private ArrayList<Integer> getPrimeNumbers() {

int pi = (int) (Double.valueOf(max) / Math.log(max));

ArrayList<Integer> primeNums = new ArrayList<Integer>(pi);

// primeNums.add(2);

for (int i = 7; i <= max; i+= 2) {

if (isPrime(i)) {

primeNums.add(i);

}

}

return primeNums;

}

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/26

\* @author Ariana Herbst

\* April 5, 2016

\*/

public class Permutations\_RandomElections

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int r = sc.nextInt();

long k = 1;

for (int i = n; i > n - r; i--)

{

k \*= i;

}

System.out.print(k + "");

}

}

import java.util.\*;

/\*\*

\* 2002 MidAtlantic USA Regionals

\* https://pcs.spruett.me/problems/114

\* @author Ariana Herbst

\* October 5, 2016

\*/

public class Recursion\_CantoringAlong {

public static String blanks;

public static Map<Integer, String> mem = new HashMap<Integer, String>();

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

StringBuilder sb = new StringBuilder(" ");

mem.put(0, "-");

for (int i = 0; i < 18; i++)

sb.append(sb.toString());

blanks = sb.toString();

boolean notFirst = false;

//System.out.println(blanks.length());

while (sc.hasNext())

{

int N = sc.nextInt();

if (notFirst)

System.out.print("\n");

System.out.print(recur(N));

notFirst = true;

}

}

public static String recur(int N)

{

String m = mem.get(N);

if (m != null)

{

return m;

}

else {

StringBuilder sb = new StringBuilder();

String k = recur(N - 1);

sb.append(k);

String filled = blanks.substring(0, (int) Math.pow(3, N - 1));

sb.append(filled);

sb.append(k);

mem.put(N, sb.toString());

return sb.toString();

}

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/117

\* @author Ariana Herbst

\* October 5, 2016

\*/

public class Recursion\_DiskGame {

static Stack<Integer>[] stacks;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int N = sc.nextInt();

stacks = new Stack[3];

for (int n = 0; n < 3; n++)

stacks[n] = new Stack<Integer>();

for (int i = N - 1; i >= 0; i--)

stacks[0].add(i);

recur(0, 2, 1, N - 1);

}

public static void recur(int current, int target, int spare, int level)

{

if (stacks[current].peek() == level) //base case

{

int disk = stacks[current].pop();

stacks[target].push(disk);

char from = (char) (current + 'A');

char to = (char) (target + 'A');

System.out.println(from + " -> " + to);

}

else

{

recur(current, spare, target,level - 1);

int disk = stacks[current].pop();

stacks[target].push(disk);

char from = (char) (current + 'A');

char to = (char) (target + 'A');

System.out.println(from + " -> " + to);

recur(spare, target, current, level - 1);

}

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/116

\* @author Ariana Herbst

\* October 21, 2016

\*/

public class Recursion\_TheEndOfTheWorld {

static Stack<Integer>[] stacks;

static long count;

static char[] c;

static long[] moves = new long[63];

public static void main(String[] args) {

createMoves();

Scanner sc = new Scanner(System.in);

String next = sc.next();

while (!next.equals("X")) {

count = 0;

c = next.toCharArray();

char prev = 'B';

int i = c.length - 1;

boolean subtract = false;

for (; i >= 0; i--)

{

if (c[i] != prev) {

count += subtract? -1 \*moves[i] : moves[i];

subtract = subtract? false : true;

}

prev = c[i];

}

System.out.println(count);

next = sc.next();

}

}

public static void createMoves()

{

moves[0] = 1;

for (int i = 1; i < moves.length; i++) {

moves[i] = moves[i - 1] \* 2 + 1;

}

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/33

\* @author Ariana Herbst

\* April 19, 2016

\*/

public class SegmentTree\_AccountMaximumQueries {

static int N, M;

static SegmentTree st;

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

N = sc.nextInt();

M = sc.nextInt();

st = new SegmentTree(new Interval(0, N - 1));

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

{

if (sc.next().equals("ADD"))

{

st.update(sc.nextInt(), sc.nextInt());

}

else { //for MAX

System.out.print(st.rangeQuery(sc.nextInt(), sc.nextInt()) + "\n");

}

}

}

//The following code is taken from Virginia Tech ACM ICPC Handbook//

public static class SegmentTree {

SegmentTree left, right;

int val;

Interval range;

public static final int IDENTITY = Integer.MIN\_VALUE;

public int combine(int x, int y) {

return Math.max(x, y);

}

public SegmentTree (int l, int r) {

this(new Interval(l,r));

}

public SegmentTree (Interval i) {

range = i;

if (range.length() > 1) {

left = new SegmentTree(range.leftInterval());

right = new SegmentTree(range.rightInterval());

}

}

public int rangeQuery(int l, int r){

Interval range = new Interval(l, r);

return rangeQuery(range);

}

public int rangeQuery(Interval i) {

if (!this.range.intersects(i)) {

return IDENTITY;

}

else if (this.range.contained(i)) {

return this.val;

}

else {

return combine(this.left.rangeQuery(i), this.right.rangeQuery(i));

}

}

public void update(int i, int val) {

if (range.length() == 1) {

this.val += val;

}

else {

if (this.range.leftInterval().containsPoint(i)) {

this.left.update(i, val);

}

else {

this.right.update(i, val);

}

this.val = combine(this.left.val, this.right.val);

}

}

}

public static class Interval {

final int left, right;

public Interval(int l, int r) {

left = l;

right = r;

}

public int midPoint() {

return (left + right)/2;

}

public int length() {

return right - left + 1;

}

public boolean containsPoint(int i) {

return this.left <= i && i <= this.right;

}

public Interval leftInterval() {

return new Interval(left, midPoint());

}

public Interval rightInterval() {

return new Interval(midPoint()+1, right);

}

public boolean contained(Interval i) {

return i.left <= this.left && this.right <= i.right;

}

public boolean intersects(Interval i) {

return !(i.right < this.left || this.right < i.left);

}

}

}

import java.util.\*;;

/\*\*

\* http://codeforces.com/problemset/problem/676/B

\* @author Ariana Herbst

\* May 25, 2016

\*/

public class Simulation\_PyramidOfGlasses {

static int N, S;

static int glassesFilled;

static ArrayList<ArrayList<Glass>> tree;

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

Scanner sc = new Scanner(System.in);

N = sc.nextInt();

S = sc.nextInt();

tree = new ArrayList<ArrayList<Glass>>();

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

{

ArrayList<Glass> temp = new ArrayList<Glass>();

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

{

temp.add(new Glass(row, j));

}

tree.add(temp);

}

pour(tree.get(0).get(0), S, 1);

System.out.print(glassesFilled + "");

}

public static void pour(Glass glass, double sec, double rate)

{

if (glass.curr != 1)

{

double possibleSecondsSpent = (1 - glass.curr) / rate;

if (possibleSecondsSpent <= sec) {

glass.curr = 1;

sec -= possibleSecondsSpent;

glassesFilled++;

}

else

{

glass.curr += sec \* rate;

return;

}

}

if (glass.row < (tree.size() - 1))

{

pour(tree.get(glass.row + 1).get(glass.ind), sec, rate / 2);

pour(tree.get(glass.row + 1).get(glass.ind + 1), sec, rate / 2);

}

}

public static class Glass {

double curr;

int row;

int ind;

public Glass(int ROW, int INDEX)

{

row = ROW;

ind = INDEX;

curr = 0;

}

}

}

import java.util.\*;

/\*\*

\* Topological Sorting

\* https://pcs.spruett.me/problems/65

\* 2012 Southeast USA Regionals

\* @author Ariana Herbst

\* August 28, 2016

\*/

public class TopologicalSorting\_DuelingPhilosophers {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int N = sc.nextInt();

int M = sc.nextInt();

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

int[] inDeg = new int[N + 1];

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

{

outGoing.add(new ArrayList<Integer>());

}

for (int m = 0; m < M; m++) //create outGoing map, record # incoming edges

{

int u = sc.nextInt();

int v = sc.nextInt();

outGoing.get(u).add(v);

inDeg[v]++;

}

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

for (int i = 1; i <= N; i++) //find nodes w/ no incoming edges, add to sort

{

if (inDeg[i] == 0)

{

noInc.add(i);

}

}

boolean multSorts = false;

for (int i = 0; i < noInc.size(); i++) //topological sort

{

if (i < noInc.size() - 1)

{

multSorts = true;

}

int u = noInc.get(i);

for (Integer v : outGoing.get(u))

{

inDeg[v]--;

if (inDeg[v] == 0)

{

noInc.add(v);

}

}

}

if (noInc.size() == N)

{

if (multSorts)

{

System.out.print("2");

return;

}

else

{

System.out.print("1");

return;

}

}

else

{

System.out.print("0");

}

}

}

import java.util.\*;

/\*\*

\* https://open.kattis.com/contests/na16warmup1/problems/dancerecital

\* @author Ariana Herbst

\* October 1, 2016

\*/

public class TravelingSalesman\_DanceRecital {

static ArrayList<String> routines;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

routines = new ArrayList<String>();

int V = sc.nextInt();

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

{

routines.add(sc.next());

}

TSP tsp = new TSP(V);

int[] k = tsp.solve();

int min = Integer.MAX\_VALUE / 4;

for (Integer i : k)

min = Math.min(min, i);

System.out.print(min);

}

//The following code is taken from Virginia Tech ACM ICPC Handbook///////

static class TSP

{

int n;

TSP(int n) {this.n = n; }

int dist(int from, int to) {

if (from != to) //if different routines

{

String A = routines.get(from);

String B = routines.get(to);

//find number of shared dancers

int shared = 0;

for(int a = 0; a < A.length(); a++) {

if (B.contains(A.charAt(a) + "")) {

shared++;

}

}

return shared;

}

return 0;

}

int initialdist(int to) {

return 0;

}

int[] solve() {

final int[][]D = new int[n][n];

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

{

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

{

D[j][i] = dist(j, i);

}

}

int[][] dp = new int[1 << n] [n];

for (int[] row : dp)

Arrays.fill(row, Integer.MAX\_VALUE);

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

dp[1<<i][i] = initialdist(i);

for (int mask = 0; mask < 1 << n; mask++)

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

if ((mask & 1 << i) > 0)

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

if (i != j && (mask & 1 << j) > 0)

dp[mask][i] = Math.min(dp[mask][i],

dp[mask ^ (1<<i)][j] + D[j][i]);

return dp[(1<<n)-1];

}

}

}

import java.util.\*;

public class Trees\_CountYourCousins {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while (true)

{

int N = sc.nextInt();

int K = sc.nextInt();

if (N == 0 && K == 0)

{

return;

}

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

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

int prev = -1;

int pInd = -1;

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

{

ArrayList<Integer> temp2 = new ArrayList<Integer>();

downTree.add(temp2);

}

int[] names = new int[N + 1];

names[0] = -1;

upTree.add(-1);

for (int n = 1; n <= N; n++) //build upTree and downTree

{

int next = sc.nextInt();

names[n] = next;

if (next - prev > 1) //if they're not siblings

{

pInd++;

}

upTree.add(pInd);

downTree.get(pInd).add(n);

prev = next;

}

int target = -1;

for (int i = 1; i <= N;i++) //find index of the node you're trying to find

{

if (names[i] == K)

target = i;

}

int parent = upTree.get(target);

if (parent <= 0)

{

System.out.println("0");

continue;

}

int grandparent = upTree.get(parent);

if (grandparent <= 0)

{

System.out.println("0");

continue;

}

int cousins = 0;

for (Integer uncle : downTree.get(grandparent))

{

if (uncle != parent)

{

cousins += downTree.get(uncle).size();

}

}

System.out.println(cousins);

}

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/44

\* @author Ariana Herbst

\* May 11, 2016

\*/

public class Trees\_KittenInATree {

static Map<Integer, Integer> branches;

static int target;

static Set<Integer> possRoot;

static int root;

static Set<Integer> nodes;

public static void main(String[] args)

{

branches = new HashMap<Integer, Integer>();

possRoot = new HashSet<Integer>();

nodes = new HashSet<Integer>();

Scanner sc = new Scanner(System.in);

target = sc.nextInt();

sc.nextLine();

//makes an array slot for each number on each line

String[] next = sc.nextLine().split(" ");

//builds the tree with children pointing to parents

while (Integer.parseInt(next[0]) != -1)

{

//the parent node at the beginning of the line is a possible root for the entire tree

//we find the actual root later

possRoot.add(Integer.parseInt(next[0]));

//maps each child to the parent

for( int i = 1; i < next.length; i++)

{

if (!next[i].equals("")) { //just in case weird stuff makes the scanner not work

nodes.add(Integer.parseInt(next[i])); //used to find the root later

branches.put(Integer.parseInt(next[i]),Integer.parseInt(next[0])); //children point to parent

}

}

next = sc.nextLine().split(" ");

}

//starts finding the root

Iterator it = possRoot.iterator();

root = -1;

int j;

while (it.hasNext())

{

j = (int) it.next();

if (!nodes.contains(j))

root = j;

}

//we now have the root

//starts reaching for the root from the target

solve(target);

}

//reaches for the parent node until it reaches the root (and is done)

//this method is recursive

public static void solve(Integer j)

{

System.out.print(j + " ");

Integer k = branches.get(j); //gets the parent node

if (!k.equals(root))

{

solve(k); //calls "solve" for the parent node

}

else

{

System.out.print(k + ""); //done B-)

}

}

}

import java.util.\*;

/\*\*

\* https://pcs.spruett.me/problems/57

\* @author Ariana Herbst

\* August 23rd, 2016

\*/

public class Tries\_RandomStrings {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int N = sc.nextInt();

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

{

words.add(sc.next());

}

TrieNode root = new TrieNode(false);

TrieNode curr, child;

for (String str : words)

{

curr = root;

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

{

child = curr.children.get(str.charAt(i));

if (child != null) {

curr = child;

if (i == str.length() - 1) {

curr.endOfWord = true;

}

}

else

{

if (i == str.length() – 1) {

child = new TrieNode(true);

}

else {

child = new TrieNode(false);

}

curr.children.put(str.charAt(i), child);

curr = child;

}

}

} ///done building trie

String in = sc.next();

curr = root;

int c = 0;

int j;

TrieNode k;

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

j = i + 1;

k = curr.children.get(in.charAt(i));

while (k != null) {

if (k.endOfWord) {

c++;

}

curr = k;

k = curr.children.get(in.charAt(j));

j++;

}

}

System.out.print(c);

}

public static class TrieNode

{

Map<Character, TrieNode> children;

boolean endOfWord;

public TrieNode(boolean end)

{

children = new HashMap<Character, TrieNode>();

endOfWord = end;

}

}

}