**Transform to Palindrome**

--------------soluciones---------------------

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

class Solution

{

static int max(int x, int y) { return (x > y) ? x : y; }

static int longest(int[] seq)

{

int n = seq.Length;

int i, j, cl;

int[,] L = new int[n, n];

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

L[i, i] = 1;

for (cl = 2; cl <= n; cl++)

{

for (i = 0; i < n - cl + 1; i++)

{

j = i + cl - 1;

if (seq[i] == seq[j] && cl == 2)

L[i, j] = 2;

else if (seq[i] == seq[j])

L[i, j] = L[i + 1, j - 1] + 2;

else

L[i, j] = max(L[i, j - 1], L[i + 1, j]);

}

}

return L[0, n - 1];

}

static void Main(String[] args)

{

string[] tokens\_n = Console.ReadLine().Split(' ');

int n = Convert.ToInt32(tokens\_n[0]);

int k = Convert.ToInt32(tokens\_n[1]);

int m = Convert.ToInt32(tokens\_n[2]);

List<int>[] graph = new List<int>[n];

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

{

graph[i] = new List<int>();

}

for (int a0 = 0; a0 < k; a0++)

{

string[] t = Console.ReadLine().Split(' ');

int x = Convert.ToInt32(t[0]);

int y = Convert.ToInt32(t[1]);

graph[x - 1].Add(y - 1);

graph[y - 1].Add(x - 1);

}

int[] map = new int[n];

bool[] visited = new bool[n];

Stack<int> stack = new Stack<int>();

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

{

if (!visited[i])

{

visited[i] = true;

stack.Push(i);

while (stack.Count > 0)

{

int num = stack.Pop();

map[num] = i;

for (int j = 0; j < graph[num].Count; j++)

{

if (!visited[graph[num][j]])

{

stack.Push(graph[num][j]);

visited[graph[num][j]] = true;

}

}

}

}

}

int[] a = Console.ReadLine().Split(' ').Select(int.Parse).ToArray();

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

{

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

}

int result = longest(a);

Console.WriteLine(result);

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

class Solution

{

public static List<Transform> transforms = new List<Transform>();

public static int[] a;

public static Dictionary<int, int> exists = new Dictionary<int, int>();

public static List<List<int>> groups = new List<List<int>>();

public static int m;

static void Main(String[] args)

{

string[] tokens\_n = Console.ReadLine().Trim().Split(' ');

int n = Convert.ToInt32(tokens\_n[0]);

int k = Convert.ToInt32(tokens\_n[1]);

m = Convert.ToInt32(tokens\_n[2]);

for (int a0 = 0; a0 < k; a0++)

{

string[] tokens\_x = Console.ReadLine().Trim().Split(' ');

int x = Convert.ToInt32(tokens\_x[0]);

int y = Convert.ToInt32(tokens\_x[1]);

transforms.Add(new Transform(x, y));

}

string[] a\_temp = Console.ReadLine().Trim().Split(' ');

a = Array.ConvertAll(a\_temp, Int32.Parse);

int groupcount = 0;

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

{

if (exists.ContainsKey(a[i]))

continue;

exists[a[i]] = groupcount;

groups.Add(new List<int>());

groups[groupcount].Add(a[i]);

groupcount++;

}

for (int i = 0; i < transforms.Count; i++)

{

if (!exists.ContainsKey(transforms[i].x))

{

exists[transforms[i].x] = groupcount;

groups.Add(new List<int>());

groups[groupcount].Add(transforms[i].x);

groupcount++;

}

if (!exists.ContainsKey(transforms[i].y))

{

exists[transforms[i].y] = groupcount;

groups.Add(new List<int>());

groups[groupcount].Add(transforms[i].y);

groupcount++;

}

if (exists[transforms[i].x] < exists[transforms[i].y])

{

int old = exists[transforms[i].y];

for (int j = 0; j < groups[old].Count; j++)

{

int item = groups[old][j];

groups[exists[transforms[i].x]].Add(item);

exists[item] = exists[transforms[i].x];

}

}

else if (exists[transforms[i].x] > exists[transforms[i].y])

{

int old = exists[transforms[i].x];

for (int j = 0; j < groups[old].Count; j++)

{

int item = groups[old][j];

exists[item] = exists[transforms[i].y];

groups[exists[transforms[i].y]].Add(item);

}

}

}

int[] pos = new int[m];

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

{

pos[i] = exists[a[i]];

}

Console.WriteLine(findPalindrome(pos));

}

public static int findPalindrome(int[] A)

{

int[][] LP = new int[A.Length][];

for (var i = 0; i < A.Length; i++)

LP[i] = new int[A.Length];

for (int i = 0; i < A.Length; i++)

{

LP[i][i] = 1;

}

for (int sublen = 2; sublen <= A.Length; sublen++)

{

for (int i = 0; i <= LP.Length - sublen; i++)

{

int j = i + sublen - 1;

if (A[i] == A[j] && sublen == 2)

{

LP[i][j] = 2;

}

else if (A[i] == A[j])

{

LP[i][j] = LP[i + 1][j - 1] + 2;

}

else

{

LP[i][j] = Math.Max(LP[i + 1][j], LP[i][j - 1]);

}

}

}

return LP[0][LP.Length - 1];

}

}

class Transform

{

public int x;

public int y;

public Transform(int x, int y)

{

this.x = x;

this.y = y;

}

}