Zain Nawaz

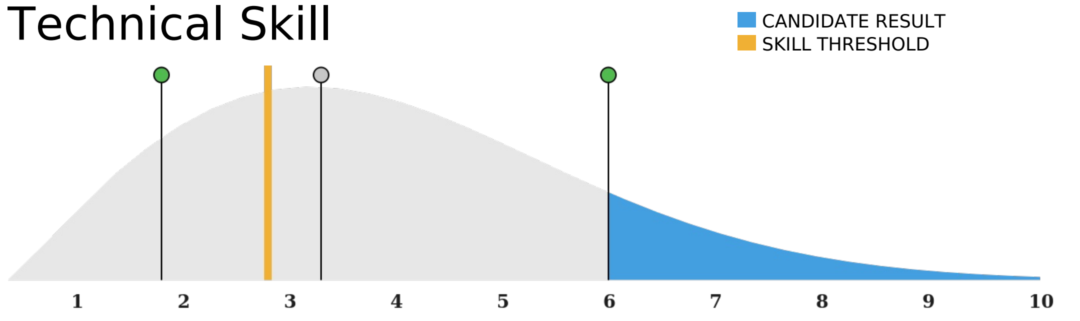
E-MAIL
zainnawaz2703@gmail.com

DATE OF LAST ACTIVITY
2020-09-09

TEST
Programming Aptitude Test

LANGUAGES USED
C, C++

Technical Skill



○ Strongly recommended regarding technical potential.
○ Recommended regarding technical potential.
● More information needed.
○ Not recommended if technical ability is crucial.

TASKS IN TEST	CORRECT	SUBMISSIONS	PLAGIARISM	LANGUAGE
EASY **A New Alphabet**	✓	15	✗	C
MEDIUM **Exploding Batteries**	✗	0	✓	None
HARD **Shibuya Crossing**	✓	10	✓	C++

The automated plagiarism check detected submissions with similar code in our database. If you need help investigating the plagiarism, please contact us at contact@kattis.com and provide the following submission IDs: 251142

EASY

A New Alphabet

SHOWN AS	MEMORY LIMIT	CORRECTNESS	PLAGIARISM
Problem A	**1024 MB**	✓ **Accepted**	✗ **Detected**
CPU TIME LIMIT	LANGUAGE USED	ATTEMPTS	EXPLANATIONS
1 second	**C**	**15**	✓ **Added**

Photo by r. nial bradshaw (https://www.flickr.com/photos/zionfiction/9588998186

A New Alphabet has been developed for Internet communications. While the glyphs of the new alphabet don't necessarily improve communications in any meaningful way, they certainly make us *feel cooler*.

You are tasked with creating a translation program to speed up the switch to our more *elite* New Alphabet by automatically translating ASCII plaintext symbols to our new symbol set.

The new alphabet is a one-to-many translation (one character of the English alphabet translates to anywhere between 1 and 6 other characters), with each character translation as follows:

Original	New	English Description	Original	New	English Description
a	@	at symbol	n	[]\[]	brackets, backslash, brackets
b	8	digit eight	o	0	digit zero
c	(	open parenthesis	p	|D	bar, capital D
d	|)	bar, close parenthesis	q	(,)	parenthesis, comma, parenthesis
e	3	digit three	r	|Z	bar, capital Z
f	#	number sign (hash)	s	$	dollar sign
g	6	digit six	t	']['	quote, brackets, quote
h	[-]	bracket, hyphen, bracket	u	|_|	bar, underscore, bar
i	|	bar	v	\/	backslash, forward slash
j	_|	underscore, bar	w	\/\/	four slashes
k	|<	bar, less than	x	}{	curly braces
l	1	digit one	y	`/	backtick, forward slash
m	[]\/[]	brackets, slashes, brackets	z	2	digit two

For instance, translating the string "Hello World!" would result in:

```
[-]3110 \/\/0|Z1|)!
```

Note that uppercase and lowercase letters are both converted, and any other characters remain the same (the exclamation point and space in this example).

Input

Input contains one line of text, terminated by a newline. The text may contain any characters in the ASCII range 32–126 (space through tilde), as well as 9 (tab). Only characters listed in the above table (A–Z, a–z) should be translated; any non-alphabet characters should be printed (and not modified). Input has at most 10 characters.
000

Output

Output the input text with each letter (lowercase and uppercase) translated into its New Alphabet counterpart.

Sample Input 1

```
All your base are belong to us.
```

Sample Output 1

```
@11 `/0|_||Z 8@$3 @|Z3 8310[]\[]6 '][' 0 |_|$.
```

Sample Input 2

```
What's the Frequency, Kenneth?
```

Sample Output 2

```
\/\/[-]@']['' $ '][' [-]3 #|Z3(,)|_|3[]\[](`/, |<3[]\[][]\[]3']['[-]?
```

Sample Input 3

```
A new alphabet!
```

Sample Output 3

```
@ []\[]3\/\/ @1|D[-]@83']['!
```

Task1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

void alphabets(int position, int length, char sub[])
{  int c=0;
   char alpha[]="@8(|)3#6[-]|_||<1[]\\/[][]\\[]0|D(,)|Z$'][']|_|\\/\\/\\/}{`/2";
   while (c < length) {    //Using position and length to find the substrng
      sub[c] = alpha[position+c];
      c++;
   }
}

void Comparison(char str)
{
  int length, position, j =0;
  char sub[10]="";
       // For comaprison
        if('a'== str || 'A' == str)
        {
            position=0,length=1;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('b'== str || 'B' == str)
        {
            position=1,length=1;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('c'== str || 'C' == str)
        {
            position=2,length=1;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('d'== str || 'D' == str)
        {
            position=3,length=2;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('e'== str || 'E' == str)
        {
            position=5,length=1;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('f'== str || 'F' == str)
        {
            position=6,length=1;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('g'== str || 'G' == str)
        {
            position=7,length=1;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('H'== str || 'h' == str)
        {
            position=8,length=3;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('i'== str || 'I' == str)
        {
            position=11,length=1;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('j'== str || 'J' == str)
        {
            position=12,length=2;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('k'== str || 'K' == str)
        {
            position=14,length=2;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('l'== str || 'L' == str)
        {
            position=16,length=1;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('m'== str || 'M' == str)
        {
            position=17,length=6;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('n'== str || 'N' == str)
        {
            position=23,length=5;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('o'== str || 'O' == str)
        {
            position=28,length=1;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('p'== str || 'P' == str)
        {
            position=29,length=2;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('Q'== str || 'q' == str)
        {
            position=31,length=3;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('r'== str || 'R' == str)
        {
            position=34,length=2;
            alphabets(position,length,sub);
            printf("%s",sub);

        }
        else if('s'== str || 'S' == str)
        {
            position=36,length=1;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('t'== str || 'T' == str)
        {
            position=37,length=4;
            alphabets(position,length,sub);
            printf("%s",sub);
         }
        else if('u'== str || 'U' == str)
        {
            position=41,length=3;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('v'== str || 'V' == str)
        {
            position=44,length=2;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('w'== str || 'W' == str)
        {
            position=46,length=4;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('x'== str || 'X' == str)
        {
            position=50,length=2;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('y'== str || 'Y' == str)
        {
            position=52,length=2;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if('z'== str || 'Z' == str)
        {
            position=54,length=1;
            alphabets(position,length,sub);
            printf("%s",sub);
        }
        else if(' '== str)
        {

              printf("%c",str);
        }
        else
        {

            printf("%c",str);

        }

}

int main()
{
    char c;
    while (scanf("%c", &c)>0)
    { Comparison(c);}
    return 0;
}
```

Explanations provided by the candidate

Explanation of the problem: Storing alphabets in a single string then using position and size to extract that information by simple if-else. "251128" is not my solution. Just wanted to confirm that run time error was due to memory.

Explanation of the Solution: No explanation given

MEDIUM

Exploding Batteries

SHOWN AS	MEMORY LIMIT	CORRECTNESS	PLAGIARISM
Problem B	**1024 MB**	✗ **Not attempted**	✓ **Not detected**
CPU TIME LIMIT	LANGUAGE USED	ATTEMPTS	EXPLANATIONS
1 second	**Not attempted**	**0**	✗ **Not Added**

Oops! The latest shipment of laptop batteries has a minor issue. Nothing major, just a slight tendency to explode when being pushed too hard, that's all. Some of our customers keep pestering us about this tiny defect however, and the media just won't give us a break.

Shipping new batteries to all customers is very expensive, so we've come up with an ingenious way to fix this issue. See, the thing is, the battery can only explode when it's being pushed too hard (i.e. when it is delivering too many Amperes). So we're going to include an ammeter (a device measuring the current) and shut down the laptop when critical levels are reached.

There's a slight problem though. We need to figure out what that critical level is. We now know from, umm, "field testing", that the laptop can explode when delivering n milliAmperes (mA), but it is possible that explosion could occur below that level. We need to figure out the highest level which is still safe, since turning off the laptop is not very appreciated by customers, so we don't want to do it needlessly.

You have been given two laptop batteries and a testing device. The testing device will make the battery connected to it (one battery can be connected at a time) deliver x mA, where x is an integer. The testing device is constructed in such a way that the battery will always explode if there's any risk of explosion at that current.

You don't want to spend too much time testing (time is money), so you want to run as few tests as possible while still figuring out the maximum current which is safe. Given Murphy's Law (everything that can go wrong will go wrong), we want to minimize the number of tests needed in the worst case.

Remember that you only have access to two batteries, so when the first has exploded, you cannot afford to blow the second one up without being sure you've found the correct answer.

Input

Input consists of multiple lines, one line per case (at most 200 test cases). Each line contains a single positive integer n between 1 and $4\,711$ inclusive, giving the current at which the battery is known to explode. Input is terminated by a line containing 0, which should not produce any output.

Output

For each case, print one line containing the worst-case number of tests to perform when using an optimal testing strategy.

Sample Input 1

```
1
2
10
23
0
```

Sample Output 1

```
0
1
4
7
```

Kattis

HARD

Shibuya Crossing

SHOWN AS	MEMORY LIMIT	CORRECTNESS	PLAGIARISM
Problem C	**1024 MB**	✓ **Accepted**	✓ **Not detected**
CPU TIME LIMIT	LANGUAGE USED	ATTEMPTS	EXPLANATIONS
1 second	**C++**	**10**	✓ **Added**

The Shibuya scramble crossing in Tokyo is infamous for being heavily used, resulting in people bumping into each other. The crossing can be modeled as a convex polygon, where the n people about to cross initially stand at a point that is on the perimeter of the polygon and in its lower half. When the traffic lights change, each person starts to walk towards a unique point on the perimeter in the upper half of the polygon. The path each person takes may look like spaghetti (it may even cross itself), but it will never leave the polygon and no two paths will cross more than once.

Shibuya Crossing by Guwashi (http://commons.wikimedia.org/wiki/File:Shibuya_Night_(HDR).jpg)

Oskar who is a badass geek observes the crossing from the Starbucks nearby. He has numbered the people in the crossing consecutively 1 through n in counter-clockwise order (starting with the person at the very left). Sadly he doesn't know the intended paths of the people at the crossing, but he has gathered some intelligence telling him exactly which persons' paths will cross one another (and this information is consistent with the physical reality).

Being a nerd he obviously knows about Murphy's Law saying "Anything that can go wrong, will go wrong!". So all people who could possibly bump into each other, i.e., all people whose paths cross, will actually bump into each other! He now asks himself, "After all the n people have crossed, what is the size of the largest group of people where everyone has bumped into each other?". Now that is a geeky and tough question, can you help him?

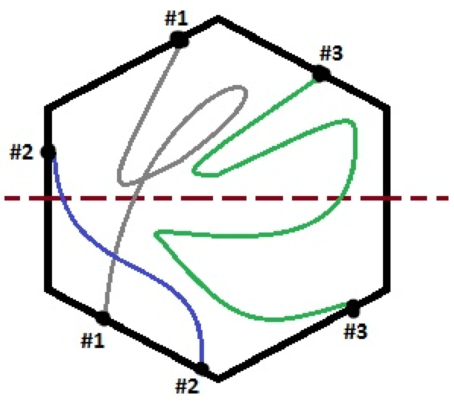


Figure 1: A beautiful illustration of a possible interpretation of the first sample test case.

Input

The first line contains an integer $1 \leq n \leq 800$, the number of people at the crossing, and an integer $0 \leq m \leq 10\,000$, the number of paths that will cross, i.e., intersect one another. The next m lines each contain two integers a and b, $1 \leq a < b \leq n$, meaning that the path taken by person a will cross the path taken by person b. (No pair will occur twice in the input.)

Output

Output a single integer giving the size of the largest group of people where everyone has bumped into one another.

Sample Input 1

```
3 1
1 2
```

Sample Output 1

```
2
```

Sample Input 2

```
5 7
1 3
1 5
1 4
2 4
3 4
2 5
2 3
```

Sample Output 2

```
3
```

t3.cpp

```
#include <bits/stdc++.h>

// For debuging
using namespace std;
void printSet(set<int> s)
{
    cout << "Set: ";
    for (int x : s) {
        cout << x << " ";
    }
    cout << endl;
}
//for debugging
void printVector(vector<int> vec)
{

    cout << "Vector: ";
    for (int x : vec) {
        cout << x << " ";
    }
    cout << endl;
}
//for debugging
void printVector(vector<vector<int>> vec)
{
    cout << "2D Vector: ";
    for (int i = 0; i < vec.size(); i++) { // printing the 2D vector.
      for (int j = 0; j < vec[i].size(); j++)
       { cout << vec[i][j] << " ";}
       cout<<endl;
    }
}

using namespace std;

// Using Longest increasing sequence algorithm to find the largest group
int lis(vector<int> arr, int n)
{
    int i, j, max = 0;
    vector<int>lis(n);

    /* Initialize LIS values for all indexes */
    for (i = 0; i < n; i++)
        lis[i] = 1;

    /* Compute optimized LIS values in bottom up manner */
    for (i = 1; i < n; i++)
        for (j = 0; j < i; j++)
            if (arr[i] > arr[j] && lis[i] < lis[j] + 1)
                lis[i] = lis[j] + 1;

    /* Pick maximum of all LIS values */
    for (i = 0; i < n; i++)
        if (max < lis[i])
            max = lis[i];

    return max;
}

void ToplogoicalSort(vector<vector<int>> matrix, vector<int> &arr, vector<int> &degree)
{
    //Checking empty places
set<int> zeroin;
    for(int i = 0; i < (matrix).size(); i++) {
        if(degree[i] == 0) {
            zeroin.insert(i);
        }
    }
    //Implementing topolgical sort starting from thoe having zero connections
while(!zeroin.empty()) {
        int curr = *zeroin.begin();
        zeroin.erase(zeroin.begin());
        arr.push_back(curr);
        for(auto next : matrix[curr]) {
            degree[next]--;
            if(degree[next] == 0) {
                zeroin.insert(next);
            }
        }
    }

}

int main() {

    int n, m,n1, n2;;
    cin >> n >> m;

    vector<vector<int>> matrix(n);
    vector<int> degree(n,0);
    vector<int> arr;

    //Taking Inputs in reverse order get directed graph
    for(int i = 0; i < m; i++) {
        cin >> n1 >> n2;
        n1--; n2--;
        matrix[n2].push_back(n1);
        degree[n1]++;
    }
    ToplogoicalSort(matrix,arr,degree);
    //Lowest number having more connection using inverse to increase the pirty of LIS
    for(auto &i : arr) {
        i = i*(-1);
    }
    cout<<lis(arr,(arr).size());
}
```

Explanations provided by the candidate

Explanation of the problem: Taking input in vector array(Matrix), then from connection having 0 degree using topological sort from them. Printing the largest group by using the Longest Increasing sequence. Using LIS without DFS having no "time limit crashing" its working :)

Explanation of the Solution: No explanation given