The Castle Problem

Programming Assignment 4

A long time ago, a mighty kingdom was invaded by the horde! Whenever there was an attack, the peasants would take refuge in the closest castle while the brave knights would leave the stronghold to confront the nasty orcs.

All over the kingdom, there were many castles spread. Nonetheless, the good and preoccupied king decided to raise yet another one. He asked his personal advisor—a former apprentice in the mysterious arts of data—to find the best location: the location should guarantee that the longest distance that any city or village had to its closest castle was as small as possible.

The apprentice unfurled a dusty vellum-bound map of the kingdom and started to work...

Problem Definition

Input: integers ***n***, ***m***, ***s***, a list of *m* roads (tuples) and a list of *s* scenarios.

- There are *n* locations numbered from 0 to n-1. Each location contains a city or village, and each city/village lies on a location.
- There are *m* (bidirectional) roads. Each road connects two locations v and w (with no locations in between), and is given as the tuple (v,w) or, equivalently, as (w,v). Every road has length exactly 1. The peasants and knights move only on roads.
- There are *s* scenarios. Each scenario consists of a list of locations. These are the locations that already contain a castle.

The task: For each scenario, the task is to determine a location for the new castle such that the maximum distance between any location and its closest castle is minimized. If there are more than one best locations, then choose the one with the highest number (meaning highest priority). Also, don't choose any location that already contains a castle.

Output: a list of *s* integers (the best location for each scenario).

Example

Input:

Output:

```
ret = [1, 6, 9, 4, 7]
```

Send Feedback

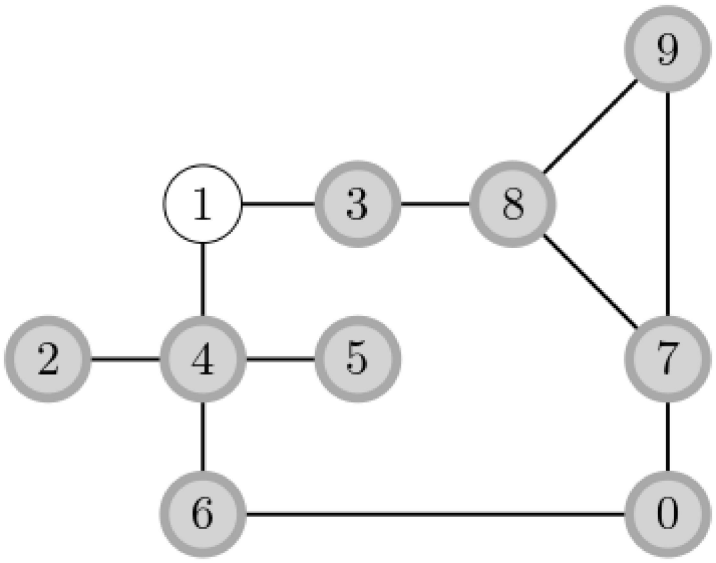
```
n = 10, m = 11, s = 5

roads =
[(1, 3)
 (3, 8)
 (8, 9)
 (9, 7)
 (8, 7)
 (7, 0)
 (0, 6)
 (6, 4)
 (4, 5)
 (4, 2)
 (1, 4)]

scenarios =
[[0,2,3,4,5,6,7,8,9],
 [],
 [4],
 [6,8],
 [9,1,8]]
```

Scenario 0:

```
[0,2,3,4,5,6,7,8,9]
```



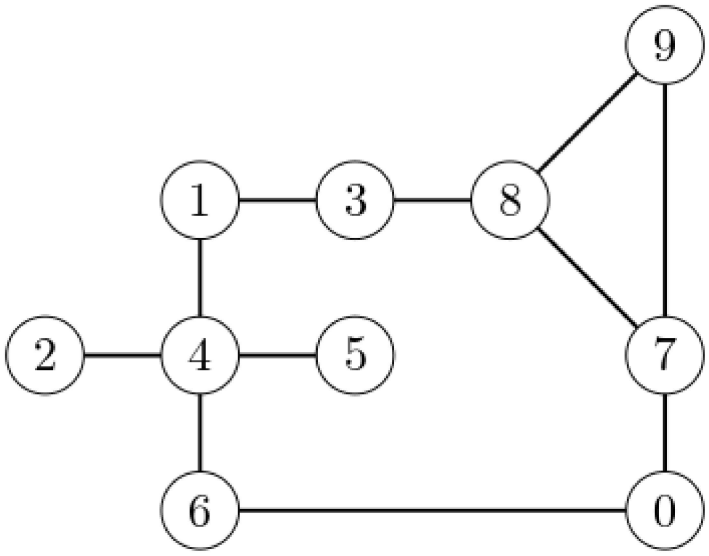
The input corresponds to the picture above. Locations are circles, roads are line segments. Locations already containing castles are shaded in gray.

In the first scenario (scenario 0), there are already *n*-1 castles build. Thus, there is only one location left without a castle: location 1.

Thus, we have to choose location 1. Note that the maximum distance from any location to a closest castle is now zero.

Scenario 1:

```
[]
```



In the second scenario, there is no castle build yet.

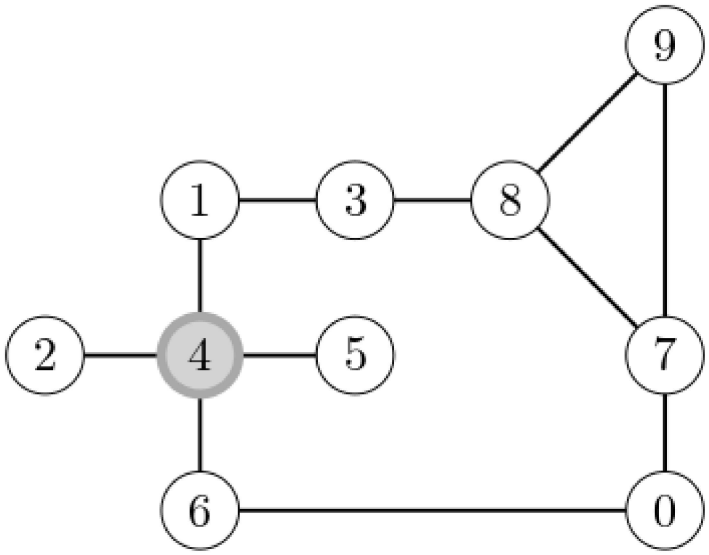
Send Feedback

If we build our castle on location either 0, 1, 3 or 6, then the maximum distance to any other location is at most 3. Note that a smaller distance is not possible.

Since 6 is the highest number among {0, 1, 3, 6}, we choose location 6.

Scenario 2:

```
[4]
```



In the third scenario, there is already a castle on location 4.

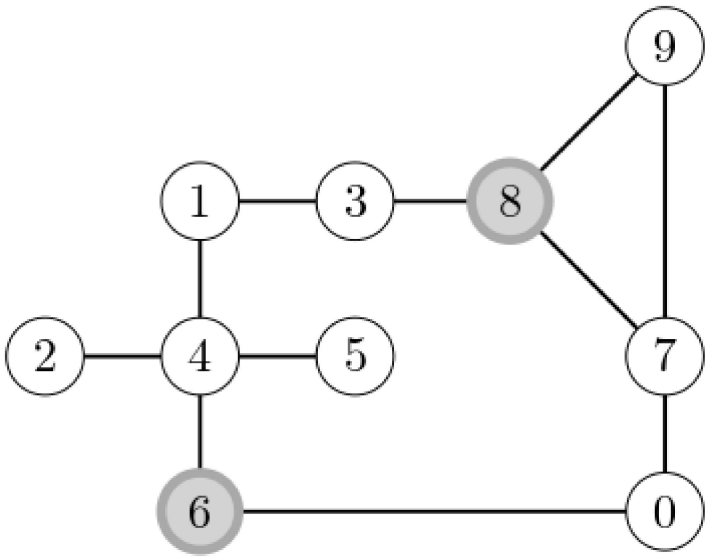
If we build our castle on location either 0, 3, 7, 8 or 9, then the maximum distance from any location to its closest castle is at most 2. Note that a smaller distance is not possible.

Since 9 is the highest number among {0, 3, 7, 8, 9}, we choose location 9.

Scenario 3:

```
[6,8]
```

Send Feedback



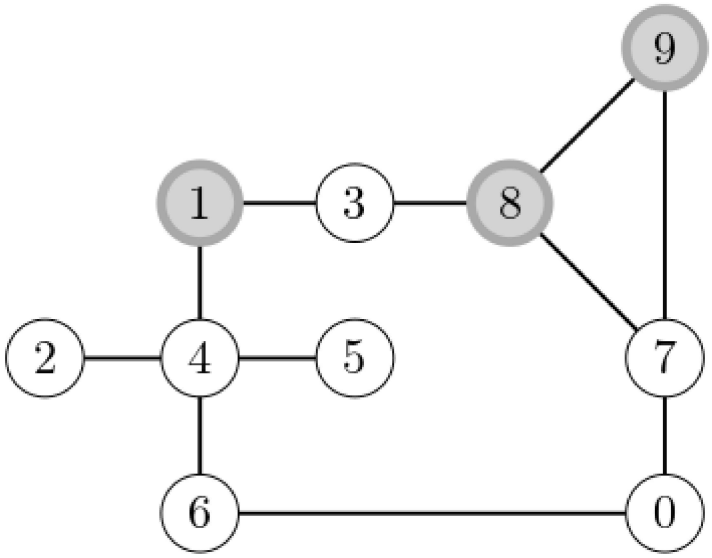
In the forth scenario, there are already two castles: one on location 6, the other one on location 8.

All locations that are neighbors to 6 or 8 have distance 1 to some castle. The only locations that have higher distance are 1, 2, and 5. If we build our castle on location 4, then their distance becomes 1. A smaller distance than 1 is not possible if the total number of castles is smaller than n.

Thus, we choose location 4.

Scenario 4:

```
[9,1,8]
```



In the fifth scenario, there are already three castles: on locations 1, 8 and 9.

Any location has already distance at most 2 to its closest castle. Independently, of where we build the next castle, the maximum distance will not decrease. Thus, any location is

Send Feedback

optimal with respect to the distance.

Thus, we choose the location with the highest priority that does not yet contain a castle: 7.

Details

You can assume that

- $1 \le n \le 250$
- $1 \le s \le 250$
- there is at most one road between any two locations,
- every location is reachable from every other one,
- there are no two castles on the same location, and
- there is at least one location without a castle.

Formally, the input of your program is given via the standard input and your output has to be written into the standard output. The format of the input and the output is described at the end of this page. However, you don't need to worry about it. The following code (one in Python and one in C++) automatically reads the input and writes the output. You can use it as a base of your program! (But you are free to write everything from scratch yourself.)

[Example Code in Python] (material/base.py) [Example Code in C++] (material/base.cpp)

Rules

- The **deadline is on Friday, 11 June** 1 minute before midnight (23:59 Polish time zone).
- Hint: model the problem as a **graph** and solve it with the help of graph algorithms from the lecture!
- Discussing the algorithm and the implementation with other people is not allowed.
- We will compare all solutions and take measures if we suspect plagiarism.
- Only C++ and Python are allowed.
- Using standard libraries is allowed—including numpy for Python (anything that works on the Szkopuł webserver is allowed).
- Upload your program on the Szkopuł webserver anytime before the deadline and how often you want. Shortly after each upload, you will see which tests you passed or failed and what your total score is.
- **Your last submission counts! Hence, please take care that your last submission is the one with the most points.**
- The initial test (Test0) corresponds to the example above and does not give you any points. Its input and output is provided below.
- The inputs of all the other tests are secret.
- You get 0 points for a test if your program crashes, has the wrong output, is too slow or takes too much memory.
- To get the full score (100 points), your program has to pass all the tests within the respective time and memory limits. There are two time limits: If you are below the first time limit, you get full points. If

you are between the two time limits, you get less points. If you are above the second time limit, you get 0 points. The second time limit is twice as large as the first one. In Szkopuł, only the value of the second time limit is displayed.

- **Activity points**: You can earn up to two activity points by submitting a test on Moodle. The test should contain an input and a corresponding output in the format as specified below. It will be published to help other students.
- In the code above, you find in the comments some code by which you can read the input from a given file (input.txt) and write the output to another file (your-output.txt) and compare it to the intended output (output.txt). By this, you can use the test data submitted by other students to test your code.

Format

Input:

- first line: n m s
- blank line
- next m lines: two distinct integers from 0,...,n-1
- blank line
- next s lines: an integer k followed by k pairwise different integers (that correspond to locations that contain already a castle); k and the locations are from 0,...,n-1

Output: s rows each consisting of an integer (the answer to the respective scenario)

Test0 has the following input and output. (Note that it corresponds to the example above.)

- [Input Test 0] (material/input0.txt) [Output Test 0] (material/output0.txt)