FIT1045 Algorithms and programming in Python, S2-2019
Assignment 2 (value 10%)
Due: Sunday 22nd September 2019, 11:55 pm.

Objectives

The **objectives of this assignment** are:

- To demonstrate the ability to implement algorithms using basic data structures and operations on them.
- To gain experience in designing an algorithm for a given problem description and implementing that algorithm in Python.
- To demonstrate an understanding of complexity, and to the ability to implement algorithms of a given complexity.

Submission Procedure

1. Save your files into a zip file called yourStudentID_yourFirstName_yourLastName.zip
2. Submit your zip file containing your solution to Moodle.
3. Your assignment will not be accepted unless it is a readable zip file.

Important Note: Please ensure that you have read and understood the university's policies on plagiarism and collusion available at `http://www.monash.edu.au/students/policies/academic-integrity.html`. You will be required to agree to these policies when you submit your assignment.
A common mistake students make is to use Google to find solutions to the questions. Once you have seen a solution it is often difficult to come up with your own version. The best way to avoid making this mistake is to avoid using Google. You have been given all the tools you need in workshops. If you find you are stuck, feel free to ask for assistance on Moodle, ensuring that you do not post your code.

Marks: This assignment will be marked both by the correctness of your code and by an interview with your lab demonstrator, to assess your understanding. This means that although the quality of your code (commenting, decomposition, good variable names etc.) will not be marked directly, it will help to write clean code so that it is easier for you to understand and explain.
This assignment has a total of 10 marks and contributes to 10% of your final mark. For each day an assignment is late, the maximum achievable mark is reduced by 10% of the total. For example, if the assignment is late by 3 days (including weekends), the highest achievable mark is 70% of 10, which is 7. Assignments submitted 7 days after the due date will normally not be accepted.
Detailed marking guides can be found at the end of each task. Marks are subtracted when you are unable to explain your code via a code walk-through in the assessment interview. **Readable code is the basis of a convincing code walk-through.**

Task 1: Complexity (4 Marks)

Create a Python module called `complexity.py`. Within this module implement the following task. For this module you may **not** use the Python built-in function `pow(x)` and you may **not** use the power operator `**`. You may not import any other libraries or modules.

Part A: Peaks (2 Marks)

Sometimes in life we care more about finding peaks than we care about finding the maximum. For example, during semester you know that the week containing the maximum amount of assessment is (probably) during exam block. But you will also have weeks in which that particular week contains more assessment than the week directly before it, or directly after it. As such, if you are planning on going to the movies with friends, you should probably do it in a week that is not an assessment peak.
Write a function `local_peak(lst)` that takes in a list of numbers, and returns the index of a local peak.

Input: a list `lst` containing two or more numbers, where no number is the same as its neighbour.

Output: an integer indicating the index of a local peak. If there is more than one peak in the given list, the index of any peak may be returned. A peak is defined as a number that is greater than its neighbours. A peak may be the first or last number in the list, in which case it must be greater than its only neighbour.

For full marks, this function must have a complexity of $O(log(N))$, where N is the length of `lst`. Be aware that some Python functions have a higher complexity than you might think. One example is slicing, which has a complexity of $O(k)$ where k is the length of the slice.

Examples

a) Calling `local_peak([0,2,4,6,5,2])` returns `3`.

b) Calling `local_peak([-7,-6,-2,-10,-5,-11])` returns *either* `2` or `4`.

c) Calling `local_peak([8.8,8.6,8.1,8.2,8.1,8.01])` returns *either* `0` or `3`.

Part B: Powers (2 Marks)

Write a function `power(n, p)` that takes a number and a power and returns the number raised to the given power.

Input: a number `n` and a non-negative integer `p`.

Output: a number that is the evaluation of n^p

For full marks, this function must have a complexity of $O(log(p))$.

Examples

a) Calling `power(2,8)` returns `256`.

Marking Guide (total 4 marks)

Marks are given for the correct behavior of the different functions:

(a) 1 mark for the correct behaviour of `local_peak`, and 1 mark for implementing the function with a complexity of $O(log(N))$. You must be able to explain why the function has a complexity of $O(log(N))$ to receive the full mark.

(b) 1 mark for the correct behaviour of `power`, and 1 mark for implementing the function with a complexity of $O(log(p))$. You must be able to explain why the function has a complexity of $O(log(p))$ to receive the full mark.

Note: marks will be deducted for including in submitted module function calls that are outside of function definitions, or print statements.

Task 2: Birthday Card (6 Marks)

Your good friend, Xanthe, who also lives in Melbourne, is turning 21 in a few weeks. To celebrate the occassion you would like to give Xanthe a birthday card that has been signed by all of her friends. The only problem is Xanthe is incredibly well travelled, and has friends all through the country and the world.

To figure out the best way to get the card to all of these friends before Xanthe's birthday, you have turned to graph theory. Every city and town in which a friend of Xanthe lives has been allocated a number between 1 and the number of friends (minus one), and Melbourne has been allocated the number 0. You have recorded postage times between locations as edge weights. You have found that all pairs of locations are such that the postage time from A to B is the same as the postage time from B to A, so *your graph is undirected*. You have also found that there is a postage connection between every pair of locations, so *your graph is connected and complete*. You have implemented this graph as an adjacency matrix, where each cell represents the *weight* of an edge. All postage times are given in integers, representing how many days the card would take to travel between the two locations. (I.e. an edge weight of 2 between A and B means it would take two days to get the card between location A and location B.)

Create a Python module called `card.py`. Within this module implement the following three tasks. You are encouraged to decompose the given tasks into additional functions. You may not import any other libraries or modules.

Part A: Time (1 Mark)

Write a function `post_time(graph, path)` that returns how long it will take Xanthe's card to travel the given path.

Input: a nested list `graph` that represents a graph as an adjacency matrix, that models the postage routes and times between Xanthe's friends; and a list of integers `path` that represents a proposed path in the graph. You may assume the given path exists in the graph.

Output: an integer that is the number of days it would take Xanthe's card to travel the given path.

Examples

```
postage1 = [ [0,1,1,3,2],
             [1,0,4,5,1],
             [1,4,0,1,3],
             [3,5,1,0,1],
             [2,1,3,1,0] ]

postage2 = [ [0,2,2,1,2],
             [2,0,1,1,2],
             [2,1,0,1,4],
             [1,1,1,0,1],
             [2,2,4,1,0] ]
```

The example graphs `postage1` and `postage2` are provided for illustrative purpose. Your implemented function must be able to handle arbitrary graphs in matrix form.

a) Calling `post_time(postage1, [0,1,4,3,2,0])` returns `5`.

b) Calling `post_time(postage1, [0,3,1,2,3])` returns `13`.

Part B: Where to post? (2.5 Marks)

Write a function `post_route(graph)` that uses a greedy approach to find a fast (but not necessarily the fastest) route by which to send the card. The metric for the greedy approach is: send the card to the location that takes the shortest time to get to that the card has not yet been visited; or, if all locations have been visited, send the card back to Melbourne. If there are multiple locations to chose from, all with the same distance, choose the location that has been allocated the smaller number.

Input: a nested list `graph` that represents a graph as an adjacency matrix, that models the postage routes and times between Xanthe's friends.

Output: a list of integers, where each integer is a location, ordered such that visiting each location in the order given creates a cycle that begins and ends in Melbourne. *The only number that can appear twice is 0.*

Examples

a) Calling `post_route(postage1)` returns `[0,1,4,3,2,0]`.

b) Calling `post_route(postage2)` returns `[0,3,1,2,4,0]`.

Part C: Can it be done? (2.5 Marks)

Write a function `on_time(graph,days)` that determines whether there exists a postage route that can be completed before Xanthe's birthday.

Input: a nested list `graph` that represents a graph as an adjacency matrix, that models the postage routes and times between Xanthe's friends; and a non-negative integer `days`, which is the number of days until Xanthe's birthday.

Output: a boolean, `True` if there exists a cycle beginning and ending at Melbourne, that visits every location, and that has a weight equal or less than `days`; otherwise `False`.

You must cite any lecture code you use.

Examples

a) Calling `on_time(postage1,5)` returns `True`, because there is a cycle in graph `postage1` with weight ≤ 5 that fits the criteria: `[0,1,4,3,2,0]`.

b) Calling `on_time(postage2,8)` returns `True`, because there is a cycle in graph `postage2` with weight ≤ 8 that fits the criteria: `[0,1,2,3,4,0]`.

c) Calling `on_time(postage2,5)` returns `False`, because there is no cycle in graph `postage2` with weight ≤ 5 that fits the criteria.

Marking Guide (total 6 marks)

Marks are given for the correct behavior of the different functions:

(a) 1 mark for `post_time`.

(b) 2.5 marks for `post_route`.

(c) 2.5 marks for `on_time`.

Note: marks will be deducted for including in submitted module function calls that are outside of function definitions, or print statements.