CS1010E: Programming Methodology

PE2 (Saturday, 10 April 2021)

Instructions

Read **ALL** instructions carefully. Read **ALL** questions carefully first before attempting them.

1. **Submission Content**:
   - Each question will be given a template file (*e.g.*, `QuestionN.py`).
   - Comment out all the given **test cases** and `import` statements before submission.
   - If any implementation is given, you are not allowed to change them.
   - You are not allowed to *add*, *remove*, or *modify* any parameters from functions.
   - You are not allowed to change the *name* of the given function.
   - Unless otherwise stated, you are not allowed to add/change `import`.
   - General restrictions should not be violated or you will be **heavily penalised**.
2. **LumiNUS Submissions**:
   - You are NOT allowed to change the filename.
   - You should submit within the given 2 hours. NO additional time will be given.
   - You should submit to the correct folder with your name on LumiNUS.
   - Late submission will be **heavily penalised**.
3. **Coursemology Submissions**:
   - You are responsible for finalizing your submission to avoid being **heavily penalised**
   - You should not submit at the last minute since Coursemology is not designed to handle large workload.
   - If you see that Coursemology is *under maintenance*, it is due to large workload and not an actual maintenance. No extension will be given.
   - Your LumiNUS and Coursemology submission should be **identical**. Any changes will be heavily penalized.
   - There are no test cases on Coursemology and our testing will use test cases that are *different* from the sample runs given.
4. **Screen Recording Submissions**:
   - Failure to submit your screen recording, missing parts of screen recording or failure to keep your proctoring camera will result in your PE being marked **ZERO**.
   - Late submission will be **heavily penalized**.
5. **Testing**: *No change from usual instructions.*
6. **Good Programming Practice**: *No change from usual instructions.*
7. **Plagiarism Warning**:
   - Any form of plagiarism including (*but not limited to*) copying of code, viewing online source, *etc.*will be marked **ZERO** and reported to the University immediately.
     - Any attempt at modifying plagiarised code including (*but not limited to*) adding comments, changing variable names, changing order of lines/functions, adding redundant statements, *etc.* *will not be able to fool the system*.

Failure to follow any of the instructions above will result in deduction of your marks.

Files

- `PE2.pdf`
- `Question1.py`
- `Question2.py`
- `Question3.py`

Coursemology

- `PE2`

LumiNUS

- `PE2 File Submission`
- `PE2 Video Submission`

Questions

Question 1: Calculus

In Assignment 4, we describe a polynomial function $P(x)$ with a single variable x as follows:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \cdots + a_2 x^2 + a_1 x^1 + a_0$$

The representation on Python is a list as follows: `[a0, a1, a2, ..., an]`. The derivative of $P(x)$ denoted by $P'(x)$ is the polynomial function:

$$P'(x) = n \times a_n x^{n-1} + (n-1) \times a_{n-1} x^{n-2} + \cdots + 2 \times a_2 x^1 + a_1$$

In this question, you are also given an implementation of `compute(poly, x)`.

General Restrictions

- The input polynomial `poly` should not be modified.
- The function `compute(poly, x)` should not be modified.
- You are NOT allowed to use the the following Python ***built-in*** string (`str`) or list (`list`) methods/functions in Question 1:.

`encode()`	`split()`	`find()`	`index()`
`decode()`	`rsplit()`	`join()`	`strip()`
`replace()`	`translate()`	`rfind()`	`sort()`
`partition()`	`map()`	`rsplit()`	`sorted()`
`rpartition()`	`count()`	`rstrip()`	`pop()`
`chr()`	`ord()`	`filter()`	`sum()`

continue on the next page...

1.1 Iterative Derivative [20 marks]

Question

Write the *iterative* function `derivative_I(poly)` to return a new polynomial that is the derivative of the given polynomial `poly`.

Assumptions

- `poly` is a valid polynomial such that `0 < len(poly) <= n+1` where `n` is the degree of the polynomial
- The polynomial `poly` will have a degree of at least `1`

Restrictions

- You are not allowed to modify the input polynomial `poly`.
- The resulting derivative `res` should also be a valid polynomial such that `0 < len(res) <= n+1` where `n` is the degree of the resulting derivative polynomial.
- You may not use recursive function(s) to solve this.

Sample Run #1

```
>>> derivative_I([3, -9, 1, 2])
[-9, 2, 6]
```

NOTE: The derivative of the polynomial $P(x) = 2x^3 + x^2 - 9x + 3$ is the polynomial $P'(x) = 6x^2 + 2x - 9$.

Sample Run #2

```
>>> derivative_I([-9, 2, 6])
[2, 12]
```

NOTE: The derivative of the polynomial $P(x) = 6x^2 + 2x - 9$ is the polynomial $P'(x) = 12x + 2$.

continue on the next page...

1.2 Recursive Derivative [20 marks]

Question

Write the *recursive* function `derivative_R(poly)` to return a new polynomial that is the derivative of the given polynomial `poly`.

Assumptions

- `poly` is a valid polynomial such that `0 < len(poly) <= n+1` where `n` is the degree of the polynomial
- The polynomial `poly` will have a degree of at least `1`

Restrictions

- You are not allowed to modify the input polynomial `poly`.
- The resulting derivative `res` should also be a valid polynomial such that `0 < len(res) <= n+1` where `n` is the degree of the resulting derivative polynomial.
- You may not use iterative constructs (*e.g.*, loop, list comprehensions, *etc.*) to solve this.
- The function `derivative_R` must be *recursive* (*i.e.*, it calls itself). The use of any recursive helper functions will not be counted as being recursive.

Sample Run #1

```
>>> derivative_R([3, -9, 1, 2])
[-9, 2, 6]
```

NOTE: The derivative of the polynomial $P(x) = 2x^3 + x^2 - 9x + 3$ is the polynomial $P'(x) = 6x^2 + 2x - 9$.

Sample Run #2

```
>>> derivative_R([-9, 2, 6])
[2, 12]
```

NOTE: The derivative of the polynomial $P(x) = 6x^2 + 2x - 9$ is the polynomial $P'(x) = 12x + 2$.

continue on the next page...

1.3 Newton's Method [10 marks]

Newton's method is a numerical analysis to find a root of a function. In other words, it finds the value of x such that $P(x) = 0$. The method works by producing successively better approximations until the desired precision. We start with an initial guess x_0. The next approximation is then:

$$x_1 = x_0 - \frac{P(x_0)}{P'(x_0)}$$

This next approximation x_1 is a better approximation than x_0. By repeating this process, we have:

$$x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)}$$

where x_n is the previous approximation and x_{n+1} is the next approximation.

Consider the polynomial $P(x) = 2x^3 + x^2 - 9x + 3$ below.

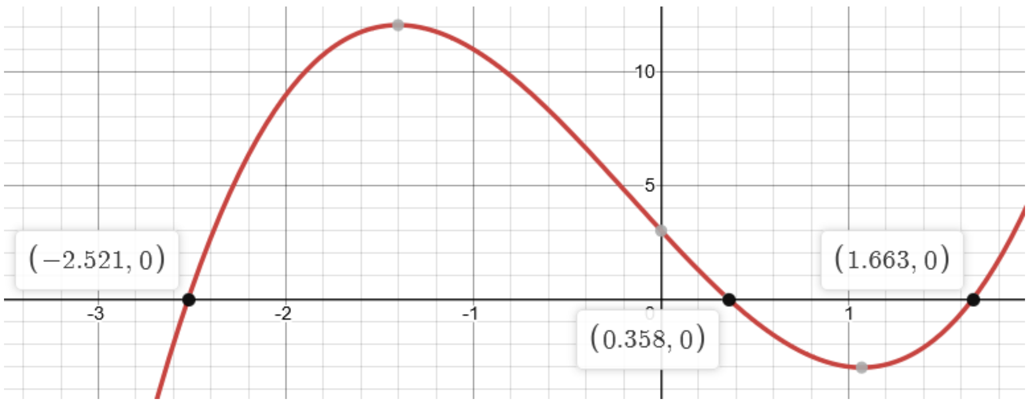


Figure 1: The graph of the polynomial $P(x) = 2x^3+x^2-9x+3$. The three roots are approximately 1.663, 0.358 and -2.521.

One of the roots is at $x = 1.663$. If we start the Newton's method with an initial guess of $x_0 = 2$, the first few iterations are shown below:

	x_n	$P(x_n)$	$P'(x_n)$	x_{n+1}
1	2	5	19	1.736842105263158
2	1.736842105263158	0.8638285464353412	12.573407202216067	1.6681392840992104
3	1.6681392840992104	0.05325968980519136	11.032410595128578	1.66331171814289
4	1.66331171814289	0.00025634022661336076	10.926258666554498	1.6632882572095151
5	1.6632882572095151	0.0000000000434626192	-	-

Notice how within 5 iterations, we already quickly converge to a result close enough to the root. We say that an answer x_n is *close enough* if $P(x_n) \approx 0$ within 5 decimal places. In other words $|P(x_n)| < 0.00001$ where $|P(x_n)|$ is the absolute value of $P(x_n)$. In this case, our *close enough* approximation is 1.6632882572095151.

Question

Write the function `newton(poly, x0)` to find a root of polynomial `poly` using Newton's method with the initial guess of `x0`.

Assumptions

- `poly` is a valid polynomial such that `0 < len(poly) <= n+1` where `n` is the degree of the polynomial
- The polynomial `poly` will have a degree of at least `1`
- $-\infty <$ `x0` $< \infty$
- You are guaranteed that a root exists

Restrictions

- You are not allowed to modify the input polynomial `poly`.
- You must use the described Newton's method and stops as soon as the answer is *close enough*.
- Even though your answer will be accepted if it is within 5 decimal places, you are not allowed to round your answer (*e.g.*, using `round` or other techniques).

Sample Run #1

```
>>> newton([3, -9, 1, 2], 2)
1.6632882572095151
```

Sample Run #2

```
>>> newton([3, -9, 1, 2], 0)
0.3577244832082757
```

Sample Run #3

```
>>> newton([3, -9, 1, 2], -2)
-2.5210127632817647
```

Question 2: Cousins

In Assignment 5, we used dictionary to represent the ancestry tree. We then asked if two people are related or not. In this question, we will ask if two people are nth cousins or not.

First, recap the ancestry information. The following ancestry *extended* tree below:

- Ben is Amy's parent (*i.e.*, father)
- Ben is Tom's parent (*i.e.*, father)
- Amy is Frank's parent (*i.e.*, mother)
- Tom is May's parent (*i.e.*, father)
- Howard is Ben's parent (*i.e.*, father)
- George is Howard's parent (*i.e.*, father)
- Frank is Joe's parent (*i.e.*, father)
- May is Jane's parent (*i.e.*, mother)
- Joe is Alf's parent (*i.e.*, father)
- Jane is Liz's parent (*i.e.*, mother)

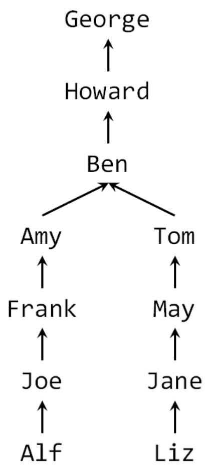


Figure 2: A simple ancestry tree.

can be represented as the following dictionary:

```
parent = {
  'Amy' : 'Ben' , 'Tom': 'Ben'    , 'Frank' : 'Amy',
  'May' : 'Tom' , 'Ben': 'Howard', 'Howard': 'George',
  'Jane': 'May' , 'Joe': 'Frank' ,  # Jane and Joe are second cousins
  'Liz' : 'Jane', 'Alf': 'Joe'      # Liz and Alf are third cousins
}
```

A cousin (*first cousin*) are relatives whose most recent common ancestor (*as defined in Assignment 5*) is the grandparent (*i.e.*, parent-of-parent). In the example above, Frank and May are cousins but Amy and Tom are not. We can extend the idea of cousin into nth cousin in which first cousin (*i.e.*, cousin) is just a specific instance of it.

More generally, cousin is a type of familial relationship in which people with a known common ancestor are both two or more generations away from their most recent common ancestor. This distinguishes a cousin from an ancestor, descendant, sibling, aunt, uncle, niece, or nephew.

– Wikipedia

In our example:

- Amy and Tom are not cousin because they are sibling.
- Frank and May are first cousin (*or simply, cousin*) and not second cousin.
- Joe and Jane are second cousin and not third cousin.
- Alf and Liz are third cousin and not fourth cousin.

General Assumptions

- Every name in the dictionary is unique.
- Every person has at most one parent.
- Your code should work any ancestry tree.
- There will be no cycle (*i.e.*, *I am my own grandparent scenario*).

continue on the next page...

General Restrictions

- The dictionary `tree` should not be modified.

2.1 Are You My Cousin? [15 marks]

Question

Write the function `is_cousin(name1, name2, tree)` to check if the person with the name `name1` is the cousin of the person with the name `name2` in the ancestry tree `tree`.

Assumptions

- `tree` is a valid ancestry tree as described above

Restrictions

- The input `tree` cannot be modified.

Sample Run #1

```
>>> is_cousin('Frank', 'May', parent)
True
```

Sample Run #2

```
>>> is_cousin('Jane', 'Joe', parent)
False
```

Sample Run #3

```
>>> is_cousin('Liz', 'Alf', parent)
False
```

2.2 N-th Cousin [30 marks]

Question

Write the function `nth_cousin(name1, name2, tree)` to find the value of n such that `name1` is the nth cousin of `name2` (*and vice versa since the relationship is symmetric*). If they are not nth cousin, return `-1`. Remember, the smallest value of n is `1` if they are cousins.

Assumptions

- `tree` is a valid ancestry tree as described above

Restrictions

- The input `tree` cannot be modified.

Hint

- You are given the ancestry tree above, make your own extensive tests to include non-cousins.

Sample Run #1

```
>>> nth_cousin('Frank', 'May', parent)
1
```

Sample Run #2

```
>>> nth_cousin('Jane', 'Joe', parent)
2
```

Sample Run #3

```
>>> nth_cousin('Liz', 'Alf', parent)
3
```

Question 3: Maze

In Tutorial Week 09, we solve a maze problem. However, there are many kinds of mazes with different restrictions. The general steps in solving a maze problem is summarised below:

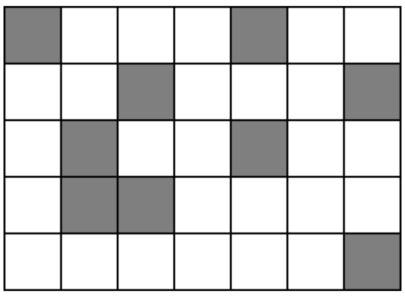
1. Start with the starting position (x, y) in my collection C and mark it as visited.
2. While C is non empty, do the following:
   (a) Select one position from C and let's call it (x', y').
   (b) Find all *neighbours* of (x', y') that are not yet visited and put it into C.
3. Check if the exit is visited.

Similar as in tutorial, we start from top-left and our objective is to go to the exit at bottom-right.

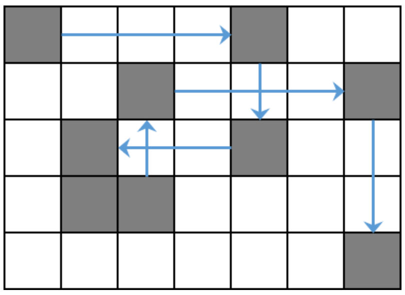
3.1 Amazing Slide [50 marks]

The sliding maze is a variation of the maze problem where you have stones on ice. You will slide on ice but if you reach a stone you can stop. This maze has no edge so you may slide off the maze and fall.

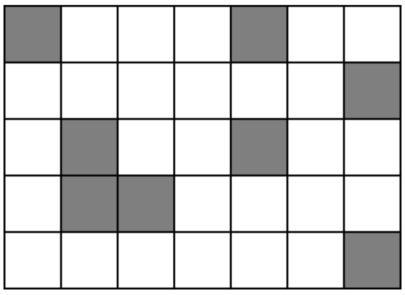
Let us indicate an ice with a white box and the stone with a grey box. Consider the following 5×7 maze below. If we start from position $(0, 0)$, we can only move to the right to reach the



position $(0, 4)$. Moving down will only cause us to fall of the edge. If our destination is the bottom right position at $(4, 7)$, this particular maze has a solution shown below. Note that from point $(2, 1)$ to $(3, 2)$ we can simply walk through $(3, 1)$ since they are connected.



Now consider the maze below. There is no solution for this particular maze.



We represent the ice with integer (`int`) `0` and the stone with integer (`int`) `1`. So, the solvable maze above is represented as:

```
maze1 = [[1,0,0,0,1,0,0],
         [0,0,1,0,0,0,1],
         [0,1,0,0,1,0,0],
         [0,1,1,0,0,0,0],
         [0,0,0,0,0,0,1]]
```

On the other hand, the unsolvable maze above is represented as:

```
maze2 = [[1,0,0,0,1,0,0],
         [0,0,0,0,0,0,1],
         [0,1,0,0,1,0,0],
         [0,1,1,0,0,0,0],
         [0,0,0,0,0,0,1]]
```

Question

Write the function `is_solvable(maze)` that returns `True` if the sliding maze is solvable and returns `False` if the maze is unsolvable. In other words, you need to check if there is a path from top-left to bottom-right.

Assumptions

- `maze` is a 2D array of integers (`int`)

Restrictions

- You are not allowed to modify the input array.

Notes

- For simplicity, you are given the following functions:
  1. `create_zero_matrix(r,c)` : Create a matrix of size `r` × `c` with all zero.
  2. `get_row(m)` : Returns the number of rows in the matrix m.
  3. `get_col(m)` : Returns the number of columns in the matrix m.
  4. `m_tight_print(m)` : Print the matrix m tightly.

Sample Run #1

```
>>> is_solvable(maze1)
True
```

Sample Run #2

```
>>> is_solvable(maze2)
False
```

– End of Paper –