CS1010E: Programming Methodology

PE2 (2019/2020 Sem 1)

Files

- `PE2.pdf`
- `Question1.py`
- `Question2.py`
- `Question3.py`
- `a6_q2_template.py`

Coursemology

- `Past PE2 > CS1010E 2019/20 Sem 1`

Questions

Question 1: Super Fibonacci Sequence

A super Fibonacci sequence (SFS) is a sequence of integers with the property that, from the third term onwards, every term is the sum of all previous terms. An example is the sequence $1, 2, 3, 6, 12, \ldots$. Note that $3 = 1 + 2$, $6 = 1 + 2 + 3$ and $12 = 1 + 2 + 3 + 6$.

We assume the first term is always 1. However, we have the freedom to choose the second term. By setting different second term, we got different sequence such as:

- $1, 5, 6, 12, 24, 48, 96, 192, 384, 768, \ldots$
- $1, 6, 7, 14, 28, 56, 112, 224, 448, 896, \ldots$

1.1 Recursive SFS [10 marks]

Question

Write the *recursive* function `super_fib_R(term2, n)` that takes in two integers (`int`) and compute the first `n` terms of the super Fibonacci sequence given the chosen second term `term2`. Return the result as a list (`list`) of integer (`int`).

Restrictions

- You may not use iterative constructs (*e.g.*, loop, list comprehensions, *etc.*) to solve this.
- The function `super_fib_R` must be *recursive* (*i.e.*, it calls itself). The use of any recursive helper functions will not be counted as being recursive.

Assumptions

- `term2 > 0` and `n > 0`
- You do not need to worry about recursion depth that is greater than 994.

Sample Run #1

```
>>> super_fib_R(10, 10)
[1, 10, 11, 22, 44, 88, 176, 352, 704, 1408]
```

Sample Run #2

```
>>> super_fib_R(20, 7)
[1, 20, 21, 42, 84, 168, 336]
```

Sample Run #3

```
>>> super_fib_R(11, 994)[-1]%10000
5376
```

Sample Run #4

```
>>> super_fib_R(11, 994)[-2]%10000
2688
```

1.2 Iterative SFS [10 marks]

Question

Write the *iterative* function `super_fib_I(term2, upper)` that takes in two integers (`int`) and compute all terms of the super Fibonacci sequence given the chosen second term `term2` such that they are smaller than or equal to `upper`. Return the result as a list (`list`) of integer (`int`).

Restrictions

- You may not use recursive function(s) to solve this.

Assumptions

- `term2 > 0` and `upper > 0`

Notes

- The second parameter has *different meaning* from Question 1.1.

Sample Run #1

```
>>> super_fib_I(4, 100)
[1, 4, 5, 10, 20, 40, 80]
```

Sample Run #2

```
>>> super_fib_I(4, 160)
[1, 4, 5, 10, 20, 40, 80, 160]
```

Sample Run #3

```
>>> len(super_fib_I(20, 10**4321))
14352
```

Sample Run #4

```
>>> super_fib_I(20, 10**4321)[-1]%(10**10)
4087703552
```

1.3 Smallest Second [10 marks]

With different second term, the SFS will contain different set of numbers. Given a number n, we want to find the *smallest second term* that will generate an SFS that contains n. The above example in Question 1.1 shows that if the second term (*i.e.*, `term2`) is 10, the number $n = 44$ will be in the SFS. However, the number 44 is **NOT** in the SFS with 20 as the second term.

Question

Write the function `smallest_second(n)` that takes in an integer (`int`) `n` and returns the smallest second term such that the second term will generate an SFS that contains the number `n`. You may assume that `n > 1`.

Assumptions

- `n > 1`

Sample Run #1

```
>>> smallest_second(2016)
62
```

Sample Run #2

```
>>> smallest_second(9876)
2468
```

Sample Run #3

```
>>> smallest_second(23592960)
44
```

Sample Run #4

```
>>> smallest_second(2651336998912)
9876
```

Question 2: Pizza Delivery 2.0

This question is based on Assignment 6. You may reuse your code here. However, you will build some new extra features on this part and we will only reward you for the new features *excluding* what you have done for the assignment. If you are not familiar with Assignment 6, you may refer to `a6_q2_template.py`.

For the original assignment, pizza shops only have coordinates. We would like to enrich the pizza shop property here to include the following attributes:

- `pos` : The position/coordinates as in Assignment 6.
- `name` : The name of the pizza shop (`str`).
- `radius` : The maximum radius of delivery of drone. In other words, the drone can only deliver pizza for a distance less than or equal to `radius`.
- `hour_s` : Opening hour of the pizza shop.
- `hour_e` : Closing hour of the pizza shop.

For simplicity, we will only records the hours (*i.e.*, `hour_s` and `hour_e`) in 24 hours format. For instance, if `hour_s = 10` and `hour_e = 20`, it means the pizza shop will open at 10:00 AM in the morning and close at 08:00 PM in the evening every day. To be precise, it will close at 08:00:00 PM and the last delivery drone will fly out at 07:59:59 PM. If a pizza shop opens 24 hours, we will have `hour_s = 0` and `hour_e = 24`.

2.1 PizzaShop [10 marks]

You are given the basic definition of the class `PizzaShop` with initial attributes. Your current task is to add a new method `distance_square_to(i, j)` such that it will return the *square* of the distance between itself and the coordinate `(i, j)`. The coordinate, similar to Assignment 6, will be given as a pair of integer (`tuple` of `int`).

Question

Add the method `distance_square_to(i, j)` to the class `PizzaShop`.

2.2 Delivery Map 2.0 [35 marks]

Similar to Assignment 6, we will draw the delivery map with the same return value format. In other words, it will be 2D array of symbols (`str`). However, unlike Assignment 6, we will be writing the following function `pd_map(r,c,all_shops,hour)` :

- `r` , `c` : Same as Assignment 6, the number of rows and columns of the map.
- `all_shops` : A list of instances of class `PizzaShop`.
- `hour` : The current time in hour defined in the same way as `hour_s` and `hour_e`.

Some modifications from the original `pd_map` :

- A pizza shop will NOT deliver to any house that has a distance more than the `radius`. If the distance is *exactly* the `radius`, it will still deliver.
- A pizza ship will only deliver when it is open according to `hour_s` and `hour_e`.
- On the map, the nearest pizza ship is shown by the first character of its name instead of numbers. Similar to Assignment 6, if a house has more than one nearest pizza shops, then an `'X'` will be shown instead. You may assume that no two pizza shops have the same first characters in their names and no pizza shops start with `'X'`.
- If there are any house not reachable from any pizza shop, put a period (*i.e.*, `'.'`) in the map.

For example, if we initialise the list of pizza shops as follows:

```
all_shop = [PizzaShop([3,3],'Ace',3,8,14), PizzaShop([6,6],'Bizza',4,12,22)]
```

With `r = 10` and `c = 12`, the three maps for different value of `hour` is shown as follows:

- `hour = 10`

```
...A........
.AAAAA......
.AAAAA......
AAAAAAA.....
.AAAAA......
.AAAAA......
...A........
............
............
............
```

- `hour = 12`

```
...A........
.AAAAA......
.AAAAAB.....
AAAAAAXBB...
.AAAAXBBBB..
.AAAXBBBBB..
..BXBBBBBBB.
...BBBBBBB..
...BBBBBBB..
....BBBBB...
```

- `hour = 16`

```
............
............
......B.....
....BBBBB...
...BBBBBBB..
...BBBBBBB..
..BBBBBBBBB.
...BBBBBBB..
...BBBBBBB..
....BBBBB...
```

Question

Write the function `pd_map(r,c,all_shops,hour)`.

Question 3: Fun with Digits

This question is about a puzzle. We start with the following incorrect equation:

```
123456789 = 100
```

If we are allowed to put any numbers of `'+'` or `'-'` signs *in between* any digits on the left side of the equation, we may have some correct equations! For this particular case, there are altogether 11 ways of setting the above equation right:

```
1+2+3-4+5+6+78+9 = 100
1+2+34-5+67-8+9  = 100
1+23-4+5+6+78-9  = 100
1+23-4+56+7+8+9  = 100
12+3+4+5-6-7+89  = 100
12+3-4+5+67+8+9  = 100
12-3-4+5-6+7+89  = 100
123+4-5+67-89    = 100
123+45-67+8-9    = 100
123-4-5-6-7+8-9  = 100
123-45-67+89     = 100
```

The equation above is just one example, its left side can be changed from `123456789` to any set of digits and its right side can be any number. For instance, if the left side digits are `111111` and the right number is `121`, we have the following correct equations:

```
11+111-1 = 121
11-1+111 = 121
111+11-1 = 121
111-1+11 = 121
```

Additionally, the set of operators can also be extended from the set `{'+', '-'}` to include more operators such as `{'+', '-', '*', '%'}`. If we use the latter set of operators, the initial equation `12345 = 30` can be corrected into:

```
1+2*3*4+5 = 30
1%2+34-5  = 30
```

3.1 Equal To [25 marks]

Question

Write the function `equal_to(lhs, rhs, ops)` to return a set (`set`) of all correct equations in string (`str`) where:

- `lhs` : A string (`str`) containing the digits on the left side. We will not include the digit `0`.
- `rhs` : An integer (`int`) for the number on the right side.
- `ops` : A string (`str`) of all possible operators. We will limit the possible operators to only `'+'`, `'-'`, `'*'` and `'%'`.

Notes

- On the left side, you can put *at most* one operator between any pair of digits.
- On the left side, you may put *no operator* between any pair of digits.
- The symbol `'-'` is treated as subtraction instead of unary negative operator.

continue on the next page...

Hints

- You may use the built-in function `eval()` to evaluate an arithmetic expression given as string (`str`).

```
>>> x = 2
>>> eval('x*3')
6
```

- You may want to write the function `product(s,n)`:
  - The function takes in a string (`str`) `s` and an integer (`int`) `n`.
  - The function returns a list (`list`) of all *Cartesian* product of `s` with length `n`
  - The expression `product(s,n)` can be computed by adding each character in `s` to the result of `product(s,n-1)`.

```
>>> product('ab',3)
['aaa', 'aab', 'aba', 'abb', 'baa', 'bab', 'bba', 'bbb']
>>> product('xyz',2)
['xx', 'xy', 'xz', 'yx', 'yy', 'yz', 'zx', 'zy', 'zz']
```

- You may want to write the function `interleave(s1,s2)`:
  - The function takes in two strings (`str`) `s1` and `s2`.
  - The function returns a string (`str`) of the interleaving between `s1` and `s2`.
  - You should assume that `len(s1) == len(s2)` for simplicity.

```
>>> interleave('abc','123')
a1b2c3
```

Sample Run #1

```
>>> equal_to('199',100,'+')
{'1+99=100'}
```

Sample Run #2

```
>>> equal_to('123456789',100,'+')
{'1+2+3-4+5+6+78+9=100',
 '1+2+34-5+67-8+9=100',
 '1+23-4+5+6+78-9=100',
 '1+23-4+56+7+8+9=100',
 '12+3+4+5-6-7+89=100',
 '12+3-4+5+67+8+9=100',
 '12-3-4+5-6+7+89=100',
 '123+4-5+67-89=100',
 '123+45-67+8-9=100',
 '123-4-5-6-7+8-9=100',
 '123-45-67+89=100'}
```

Sample Run #3

```
>>> equal_to('111111',100,'+')
{'111-11*1=100', '111*1-11=100', '1*111-11=100', '111-1*11=100'}
```

– End of Paper –