0/13 Questions Answered

Assignment 2 Questions

Q1
0 Points

Assignment 2 has 9 questions. We provide you with starting codes. Please fill in the blanks in the starting codes.
Please download the starting code here.

- Please submit your code to **"Assignment 2 Auto-grader"** in the Gradescope.
- Q2-Q7: exercises for Python basic syntax, where you will solve them using Python built-in data types and structures, such as list, dictionary, loops, and also recursions;
- Q8-Q9: exercises for OOP syntax; the code of these questions will be reviewed manually.
- Q10: exercise for computational thinking, where you must design and realize a solution using Python. You may try it after having this Wednesday's class.

Save Answer

Q2 Comparing tuples

0.5 Points

In Python, a function can be an argument to another function. The following code snippet takes Python's built-in function print and passes it as an argument:

```
def my_print(p, a_str):
    p("hello {}".format(a_str))


>>> my_print(print, "ICS")
hello ICS
```

Please put your code in find_max.py.

Q2.1

0.1 Points

Write a function order(p, q, n) that takes two multi-element tuples p and q, and compares the element at index n, returns True if p[n] > q[n]. Examples:

```
>>> order((1, 2, 3), (2, 1, 4), 0)
False
>>> order((1, 2, 3), (2, 1, 4), 1)
True
>>> order((1, 2, 3), (2, 1, 4), 2)
False
```

Save Answer

Q2.2

0.2 Points

Use order to implement a function first_max(order_f, l, n). l is a list of tuples; the function returns the first largest tuple in l. The comparison of order is done using the order you implemented in B.1 (i.e., a tuple p is larger than q if p[n] > q[n]).

Example: ('B', 6), ('X', 6) and ('P', 6) are all the largest on index 1, but ('B', 6) is the correct result because it is the first largest:

```
>>> tuplst = [('X', 5), ('B', 6), ('P', 4), ('X', 3), ('B', 5), ('P', 6)]
>>> print(first_max(order, tuplst, 1)
('B', 6)
```

Save Answer

Q2.3
0.2 Points

Write a function last_max(order_f, l, n); this time returns the last one.

Requirements:
1. you can not invoke first_max on a reversed list;
2. you must not modify the order function, and you can only use order_f to test order of two tuples.

Example: this time ('P', 6) is the last largest

```
>>> tuplst = [('X', 5), ('B', 6), ('P', 4), ('X', 3), ('B', 5), ('P', 6)]
>>> print(last_max(order, tuplst, 1))
('P', 6)
```

Save Answer

Q3 Selection sort
0.5 Points

Selection sort is another simple sorting algorithm that sorts a list by repeatedly finding the minimum element from the unsorted part of the list and moving it to the beginning (Recall the figure in the slides of the Lecture). The pseudo-code is as follows,

```
selection_sort(a_list) # a_list is a list of N numbers.
  for i from 0 to N-1
    min_j = i
    for j from i to N-1
      if a_list[j] < a_list[min_j]
         min_j = j
    swap a_list[i] and a_list[min_j]
```

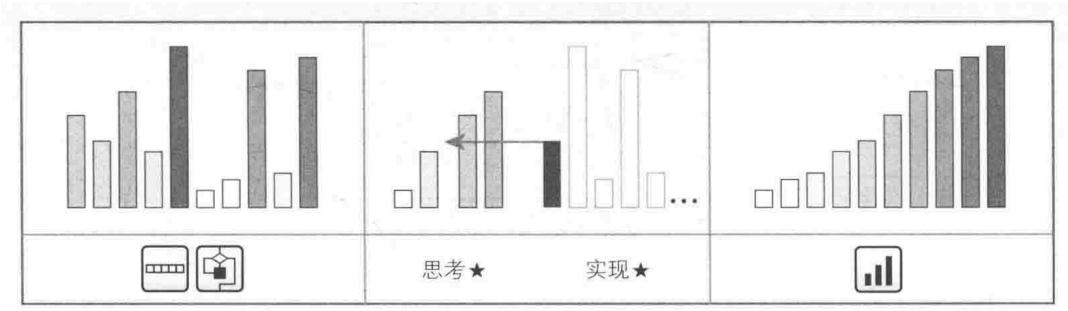
Please write your code in selection_sort.py.

Save Answer

Q4 Insertion sort
0.5 Points

Insertion sort is a sorting algorithm that is quite efficient for small or nearly sorted lists. Its idea is like the way we arrange cards when playing card games. Once we have a hand that is already sorted, and when we pick up a new card, we want to find the correct place to insert it to maintain the sorted order. So, we have to compare the rank of the new card with the ranks of the cards in our hands one by one and insert it after the card whose rank is smaller than it.

Now, we have a list of numbers and want to sort them using the insertion sort. You can imagine each number is a card. At the very beginning, you have the first number in your hand; then, you need to find the correct place for the 2nd number by comparing it with the 1st one. If it is greater than the 1st number, you should insert the 2nd number after it; otherwise, insert the number in front of it. And for the following numbers, the 3rd, 4th, 5th, and so on, you can sort them in the same way, i.e.,
1. comparing the number with the sorted numbers;
2. if there is a number that is smaller than it, inserting it after the number;
3. otherwise, put the number at the beginning of the sorted number list.



You may refer to the following pseudocode,

```
insertion_sort(a_list) #a_list is a list of N numbers
  for i from 1 to N-1
    v = a_list[i]
    j = i - 1
    while j >= 0 and a_list[j] > v
       a_list[j+1] = a_list[j]
       decrement j by 1
    a_list[j+1] = v
```

Please write your code in insertion_sort.py.

Save Answer

Q5 Shell sort
0.5 Points

Shell sort, also known as diminishing increment sort, is an algorithm that improves upon the insertion sort by allowing the exchange of items that are far apart.

The algorithm can be viewed as running the insertion sort with a sequence of gaps. It starts by sorting pairs of elements far apart from each other, then progressively reducing the gap between elements to be compared and exchanged. The final stage of the algorithm is a standard insertion sort (i.e., gap=1), but by that time, the array has been partially sorted, making the insertion step more efficient. The pseudo-code is as follows,

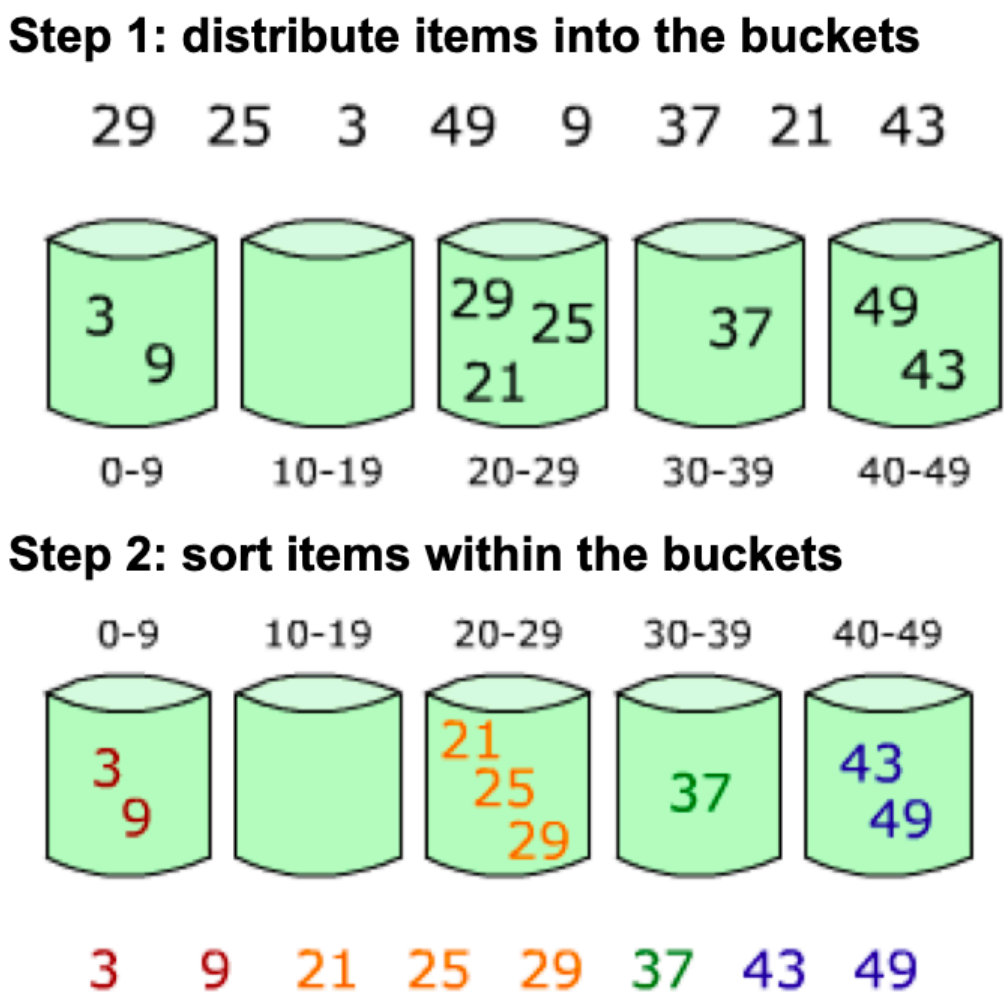
```
insertion_sort_with_gap(a_list, g)
# a_list is a list of N numbers, and g is the gap
  for i from g to N-1
    v = a_list[i]
    j = i - g
    while j >= 0 and a_list[j] > v
       a_list[j+g] = a_list[j]
       j = j - g
    a_list[j+g] = v

shell_sort(a_list):
# G is a list of M gaps in the descending order
# in this assignment, we simply set G = [5, 3, 1]
  G = [a list of gaps]
  for i from 0 to M-1
    insertion_sort_with_gap(a_list, G[i])
```

Save Answer

Q6 Bucket Sort
0.5 Points

We have learned bubble sort and merge sort. Another very useful sorting algorithm in practice is bucket sort. The intuition is simple:

- You throw items into a set of buckets; each is responsible for a fixed and constant range
- Sort items within a bucket using other sorting algorithms
- Go through the buckets in order; you are done!



Implement bucket sort: we will sort a list of random integers range in 0~99 and will have every bucket responsible for exactly the same range. So, the first bucket takes numbers from 0 to 9, the second from 10 to 19, and so on.

The following hints are ways to implement bucket sort using the above intuition:

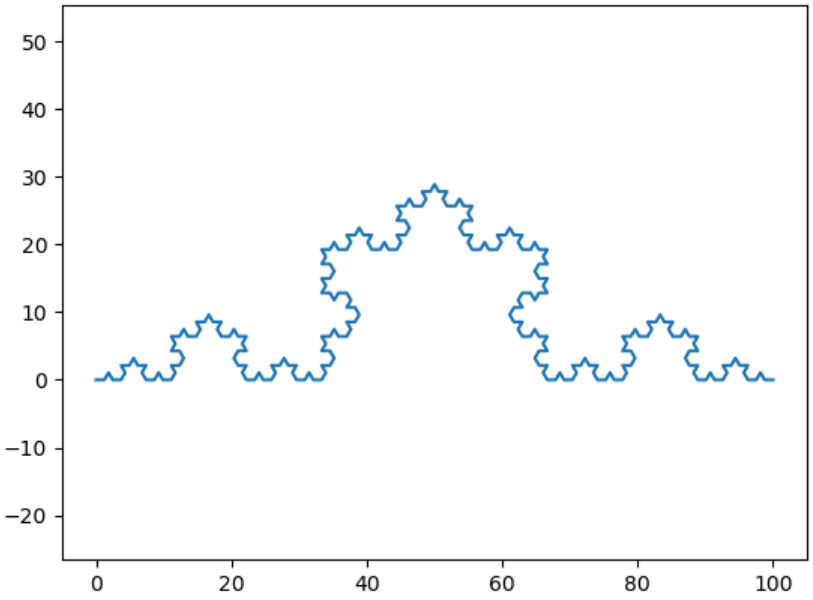
- Use the starting code provided from bucket_sort_student.py
- Use a dictionary to implement buckets, each bucket holds a list.
- Distribute items into the list in each bucket. In our case, if x is a number in the list, x//10 will find its bucket (Actually, here, 10 is the size of a bucket). If the bucket doesn't exist, start a list with just x; otherwise, append it to the list.
- For each bucket, sort its list; you can use built-in list sort, i.e., mylist.sort() will sort mylist in ascending order
- Then go through the dictionary in order of keys, using sorted(mydict.keys()).

Please write your code in bucket_sort.py.

Save Answer

Q7 The Koch Curve
1 Point

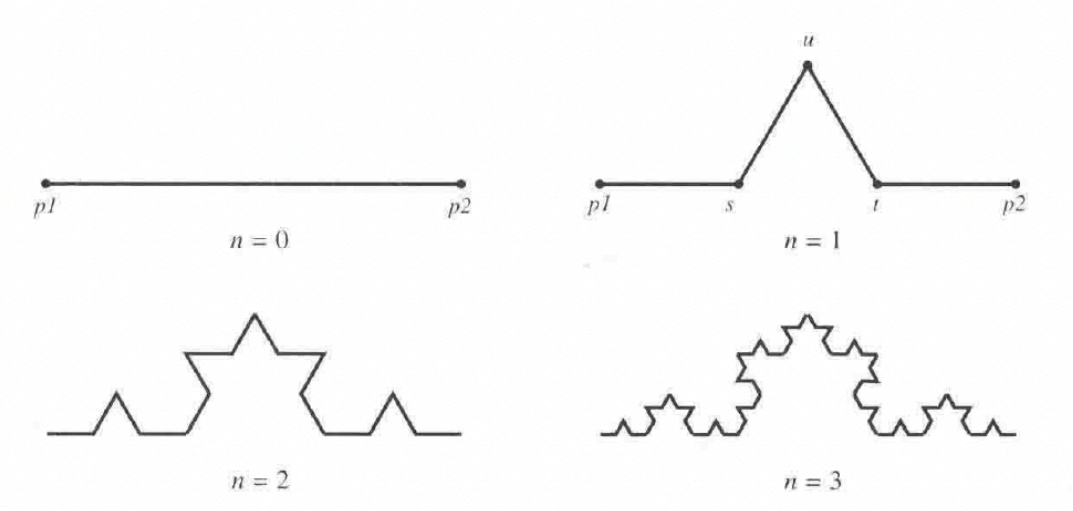
The Koch Curve is a mathematical curve and one of the earliest fractal curves to be described. It was introduced by the Swedish mathematician Helge von Koch in 1904. (See the following figure)



The Koch Curve is constructed using an iterative process. Given a line segment defined by two points $(p1, p2)$,

1. Divide the segment into three equal parts. Let the division points be s and t.
2. Replace the middle segment (s, t) with an equilateral triangle pointing outward. The new point is u, and (s, u, t) forms an equilateral triangle.
3. Repeat step 1 and step 2 for the line segment $(p1, s)$, (s, u), (u, t), and $(t, p2)$.

The following figure shows the construction of a Koch curve of $n = 3$, where n is the depth of the curve.

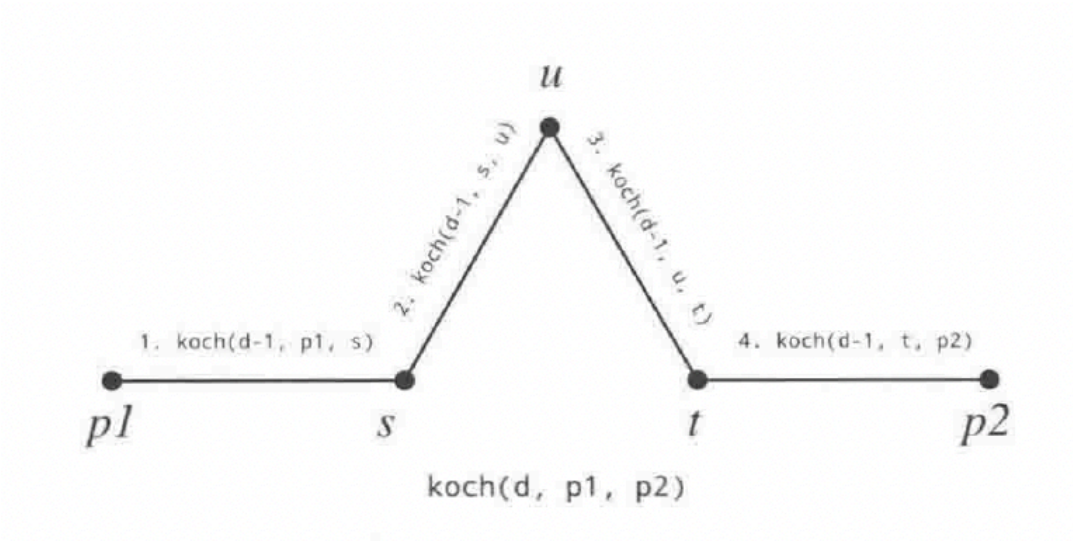


The construction of the Koch Curve can be expressed through recursive mathematical equations, and it exhibits self-similarity, meaning that smaller portions of the curve resemble the overall shape of the entire curve. The concept of the Koch Curve is foundational in the study of fractal geometry and has applications in various scientific and artistic fields.

You need to write a program to construct a Koch curve of n depth. Specifically, your program should return the coordinates of all segment terminal points of a n-depth Koch curve from $p1$ to $p2$.

Q7.1 Compute the coordinates of s, t, and u
0.5 Points

Given a segment $(p1, p2)$; let $p1.x$, $p1.y$, $p2.x$, and $p2.y$ represent the coordinates of $p1$ and $p2$, respectively.



For s, t, and u, the coordinates can be obtained by the following,
$s.x = (2 \times p1.x + 1 \times p2.x)/3$
$s.y = (2 \times p1.y + 1 \times p2.y)/3$
$t.x = (1 \times p1.x + 2 \times p2.x)/3$
$t.y = (1 \times p1.y + 2 \times p2.y)/3$
$u.x = (t.x - s.x) \times cos(\pi/3) - (t.y - s.y) \times sin(\pi/3) + s.x$
$u.y = (t.x - s.x) \times sin(\pi/3) + (t.y - s.y) \times cos(\pi/3) + s.y$

Please complete the functions get_s_coordinates, get_t_coordinates, and get_u_coordinates in koch_curve.py.

Save Answer

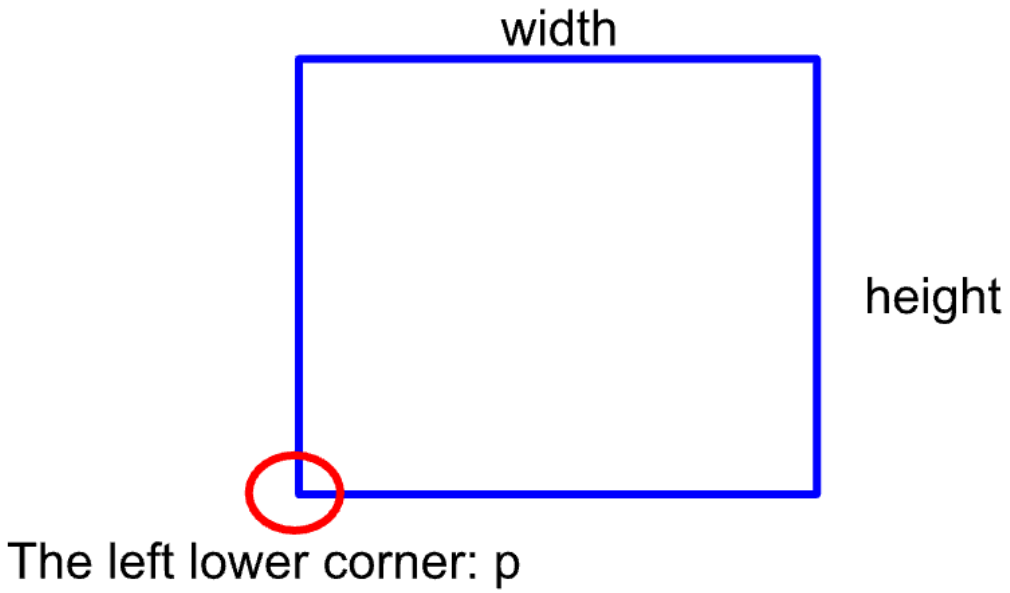
Q7.2
0.5 Points

We compute the coordinates recursively. For each segment in the Koch curve, we calculate the coordinates of the division points s, u, t, and return them in the order from $p1$ to $p2$, i.e., the return value is a list containing coordinates of $p1$, s, u, t, and $p2$; the coordinates of a point is a tuple of (x, y). Please complete the function koch_curve in koch_curve.py.

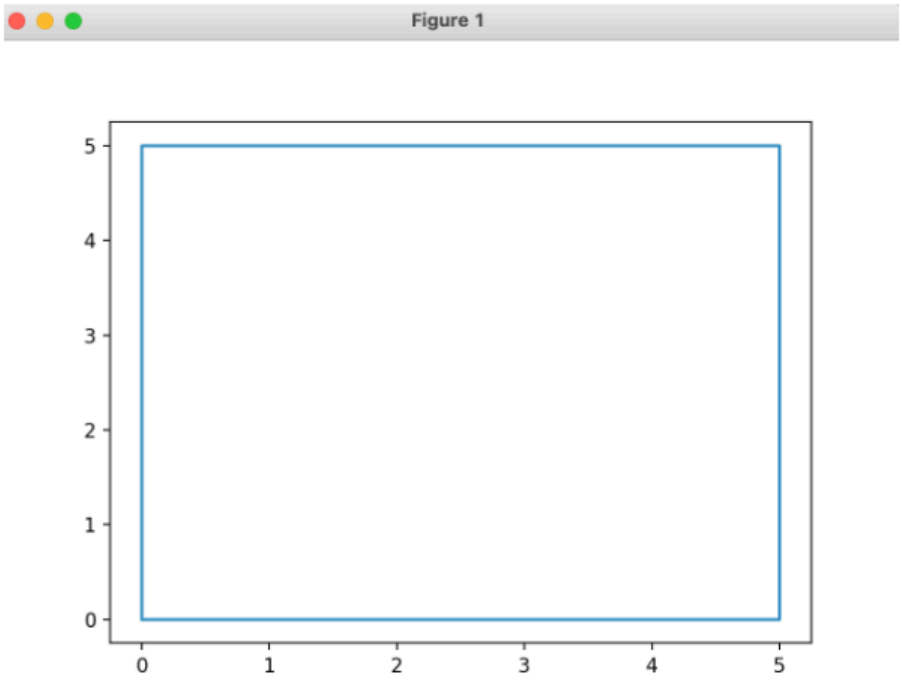
Save Answer

Q8 Rectangle
0.5 Points

We have a Rectangle class that is a subclass of the Polygon class. However, the Rectangle is initialized by the left lower corner, the width, and height (see the following figure), which is different from that of the Polygon class. Please note that the left lower corner P is an object of the Point class.



You need to write the __init__ method to initialize the Rectangle properly so that the methods that are **inherited** from the Polygon class can work **without** any modification. When you run the test code, the output should look like the following,



Please write your code in the rectangle.py, and upload it for review.

Save Answer

Q9 A hash table for contacts

0.5 Points

You will complete the Contacts class, which is essentially a hash table for storing contacts and their phone numbers. Specifically, the Contact class has a set of buckets and a hash function. It uses the hash function to convert the contact's name to the label of a bucket and then stores the contract's information in the bucket. When one needs to retrieve the information, he/she can input the name of the contact, and it will return the information.

It has four methods:

- __init__: the method for initializing an instance of the Contacts class. It has two attributes:
  - self.buckets: it is a list of "buckets"; a bucket is a list, and its elements are the contacts' information, which is a tuple of (name, phonenumber).
  - self.numBuckets is a variable that represents the number of buckets. You need to specify a number for it when creating an instance of the Contacts class.
- _hash: it is the hash function. In this case, it returns the remainder of sumNameUnicode/numBuckets, where sumNameUnicode is the sum of the Unicode of **each** character in the name, and numBuckets is the number of buckets.
- addEntry appends the tuple (name, phone number) to a bucket in self.buckets. The hash function computes the bucket's index (i.e., the label).
- getValue: it returns the phone number of the input name.

```
class Contacts:
    """
    A class for storing contacts and their phone numbers
    You can refer the textbook: [MIT text],
    Introduction to Computation and Programming Using Python
    Chapter 10.3 Figure 10.7
    """

    def __init__(self, numBuckets):
        """Create an empty dictionary"""
        self.buckets = []
        self.numBuckets = numBuckets
        for i in range(numBuckets):
            self.buckets.append([])
        return

    def _hash(self, name):
        """Convert the name to the label of a bucket
        1. get the unicode of each character in the name[hint: you can use ord()]
        2. compute the remainder of (the_sum_of_all_unicodes/the_number_of_buckets)
        3. return the remainder
        """
        # insert your code here

        return

    def addEntry(self, name, phoneNumber):
        """adding the contact information to a bucket.
        1.find the label of the bucket;
        2.append the (name, phoneNumber) to it.
        """
        # insert your code here
        return

    def getValue(self, name):
        """retrive the contact information
        1. find the bucket where the name is stored
        2. get the phone number of the name in the bucket;
           if there is collision, take the phone number
           of the first element in the bucket.
        """
        return
```

When you run the test code, the output should look like this,

```
The phone of Tyrion Lannister is  66435
```

Please write your code in the hashing.py.

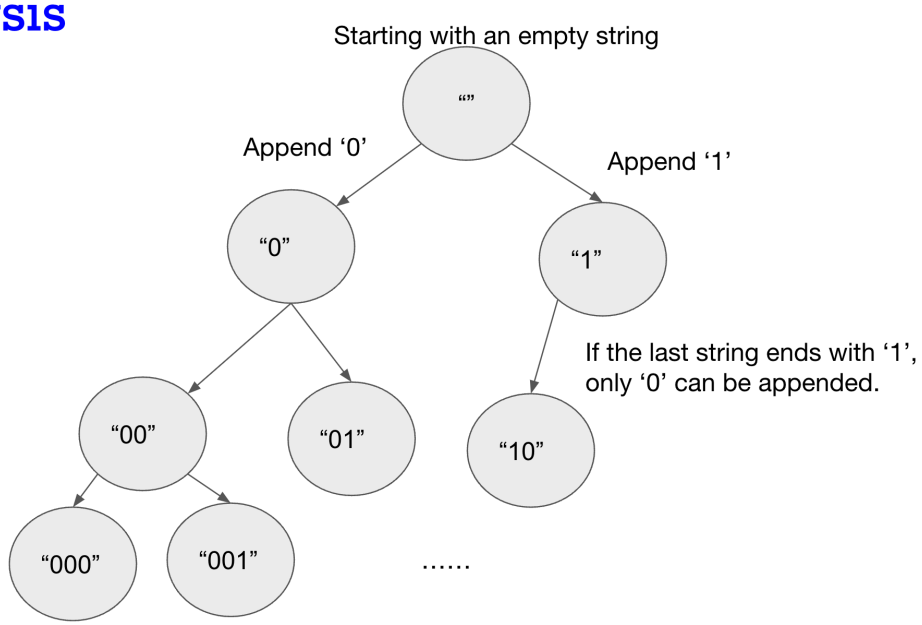
Save Answer

Q10 Generate all binary strings without consecutive 1s

0.5 Points

Write a function to generate binary strings of a given length n. The binary strings should not have consecutive 1s. You may design your algorithm by referring to the following figure.

Analysis



The following figures show the expected outputs of your function.

```
In [8]: n = 3

In [9]: binary_string = []

In [10]: binary_string_without_consecutive_ones(n, binary_string)
Out[10]: ['000', '001', '010', '100', '101']

In [11]: n = 4

In [12]: binary_string = []

In [13]: binary_string_without_consecutive_ones(n, binary_string)
Out[13]: ['0000', '0001', '0010', '0100', '0101', '1000', '1001', '1010']
```

Please write your code in the generate_binary_string.py.

Save Answer

Save All Answers

Submit & View Submission