Homework Set 2

Suggested Solution

CSCI 204.01

Prof Meng

Assigned: Monday, 03/02/2020

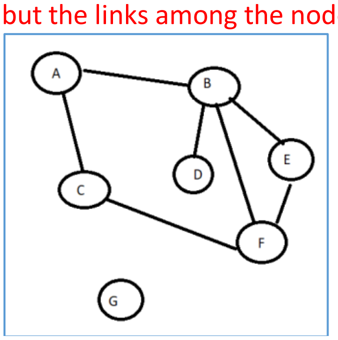
Due: Monday, 03/16/2020

1. For the given graph of cities represented as Python dictionary, following the algorithm of Depth First Search (stack solution), complete the following tasks.

```
graph = {'A': set(['B', 'C']),
         'B': set(['A', 'D', 'E', 'F']),
         'C': set(['A', 'F'']),
         'D': set(['B']),
         'E': set(['B', 'F'])
         'F': set(['B', 'C', 'E'])
         'G': set([])}
```

   a. Draw the diagram represented by the above Python dictionary;
   b. Demonstrate the algorithm how to find if there is a path between the city of 'A' and 'F' by drawing the changes of the stack;
   c. Demonstrate the algorithm how to find if there is a path between the city of 'C' and 'E';
   d. Demonstrate the algorithm how to find if there is a path between the city of 'A' and 'G'.

   a. The graph should look like the following. The shape and the location of the nodes may vary, but the links among the nodes should be the same.



   b. S = stack(), assume the right side of the list is the top of the stack
      S = [A]
      S.pop(), S.push(B), S.push(C)
      S = [B,C]
      S.pop(), S.push(F)
      S = [B, F]
      S.pop()
      'F' is the target! So there is a path from A to F
   c. S = stack(), assume the right side of the list is the top of the stack
      S = [C]
      S.pop(), S.push(A), S.push(F)

S = [A, F]
S.pop(), S.push(B), S.push(E)
S = [A, B, E]
S.pop()
'E' is the target! So there is a path from C to E

d. S = stack(), assume the right side of the list is the top of the stack
S = [A]
S.pop(), S.push(B), S.push(C)
S = [B, C]
S.pop(), S.push(F)
S = [B, F]
S.pop(), S.push(E)
S = [B, E]
S.pop(), nothing can be pushed as all E's linked nodes have been visited
S = [B]
S.pop(), S.push(D)
S = [D]
S.pop(), nothing can be pushed as D's linked node B has been visited
S = []
Algorithm stops. 'A' can't reach 'G'

2. Do the same using the Breadth First Search (queue solution) using the same data.

a. S = queue(), assume the right side of the list is the end (tail) of the queue
S = [A]
S.deq(), S.enq(B), S.enq(C)
S = [B,C]
S.deq(), S.enq(D), S.enq(E), S.enq(F)
S = [C, D, E, F]
S.deq(), nothing to enq as C's connections A and F have been visited
S = [D, E, F]
S.deq(), nothing to enq as D's connection B has been visited
S = [E, F]
S.deq(), nothing to enq as E's connections B and F have been visited
S = [F]
S.deq(), nothing to enq as F's connections B, C, and E all have been visited
'F' is the target! So there is a path from 'A' to 'F'

b. S = queue(), assume the right side of the list is the end (tail) of the queue
S = [C]
S.deq(), S.enq(A), S.enq(F)
S = [A, F]
S.deq(), S.enq(B)
S = [F, B]
S.deq(), S.enq(E)
S = [B,E]

S.deq(), S.enq(D)
S = [E, D]
S.deq()
'E' is the target! So there is a path from C to E

3. S = queue(), assume the right side of the list is the end (tail) of the queue
   S = [A]
   S.deq(), S.enq(B), S.enq(C)
   S = [B, C]
   S.deq(), S.enq(D), S.enq(E), S.enq(F)
   S = [C, D, E, F]
   S.deq(), nothing to enq as C's connections A and F have been visited
   S = [D, E, F]
   S.deq(), nothing can be enqed as D's connection B has been visited
   S = [E, F]
   S.deq(), nothing to enq as E's connections B and F have been visited
   S = [F]
   S.deq(), nothing can be enqed as F's linked node B has been visited
   S = []
   Algorithm stops. 'A' can't reach 'G'

4. Write a function using stack ADT called **is_palindrome(s)** that takes a string as the parameter and returns **True** if the string represents a palindrome, **False** otherwise. You can assume all functions in a standard stack ADT are defined for you.

```
def is_palindrome(s):
  stack = stack()
  for c in s:
    stack.push(c)
  for c in s:
    if c != stack.pop()
       return False
  return True
```

5. Given a circular queue of capacity of 6 using an array, assuming all other functions are defined,
   a. Define the two functions is_full() and is_empty(). You can choose how these two functions are defined.
   b. Show how the content of the queue evolves when inserting the integers 2, 3, 4, 5, 6 into the queue. When is the queue full? Why?

   a. Assume that the initial condition of the queue is front == back == 0 when the queue is empty,

```
def is_empty(self):
    return self.front == self.back

def is_full(self):
```

return ((self.back + 1) % n == self.front)   # n == 6 in our case

b. Empty queue [_], front == 0, back == 0, an underscore '_' means an empty spot.
   enq(2), enq(3), enq(4), enq(5), enq(6) result in the following
   [_, 2, 3, 4, 5, 6] and front == 0, back == 5. At this point, the queue is full because (back + 1) % 6 == front (Note that enq() would have to increment the value of 'back' by one first before putting the item into the queue.

6. For each of the following situations, which of these ADTs (1 through 4) would be most appropriate to represent the data: (1) a queue; (2) a stack; (3) a list; (4) none? Briefly explain your answer(s).
   a. The customers at a deli counter who take numbers to mark their turn
      Queue, this is a first-come-first-serve queue.
   b. An alphabetic list of names
      List, depending on what the applications with this list of names. The data structure could be changed into others.
   c. Integers that need to be sorted
      List: as we may need to alter the locations of these integers.
   d. A grocery list ordered by the occurrences of the items in the store
      Queue, as an order is maintained, it is best to use a queue.
   e. A list of tasks to be completed in chronological order
      Queue, as an order is maintained, it is best to use a queue.
   f. Airplanes that are approaching an airport, waiting to land
      Queue, they have to land in the order of arrival
   g. People who are put on hold when they call a travel agency to make hotel reservations
      Queue, they will be serviced according to the order of arrival
   h. A collection of papers submitted by students that needs to be graded, facing up (i.e., the cover page is facing up)
      Stack: if the professor so chooses, grading the papers from the top of the pile down.