Authors: Daniel Oh (20063998), Evelyn Yach (20071956)

# **Programming**

## Eulerian Cycle

1. Why you think your algorithm is correct (whether you program worked on the sample data or not).

Our algorithm uses builds a stack to keep track of which vertices come after the others, while also generating a list of edges that have already been visited. If an edge has already been visited, it will not be ever represented on the stack. Once the last vertex is determined to have no more traversable edges, the stack then starts popping the vertices and adds them to the path until either the stack is empty or there is a vertex with an untraversed path.

2. Provide an estimate of the time and space complexity of your algorithm.

Time Complexity: O(E)

Space Complexity: O(V2 + E)

For space complexity, I put down V2 mainly because of the fact that there is a possibility of every vertex having an edge that goes to every other vertex (e.g. a triangle shaped graph)

3. Add three-unit tests using the Rosalind sample data, and some of your own. There must be at least one positive and one negative unit test.

See main.py

## Contigs

1. Why you think your algorithm is correct (whether you program worked on the sample data or not).

Our algorithm constructs a graph which denotes possible continuations for contigs. It then uses this graph to generate a complete list of contigs.

2. Provide an estimate of the time and space complexity of your algorithm.

Time Complexity: O(V + E)

Space Complexity: O(V + E)

3. Add three-unit tests using the Rosalind sample data, and some of your own. There must be at least one positive and one negative unit test.

See main.py

# **Theory**

## 2.1 Lesson 3.3

Fuck

## 2.2 Peaceful Placement of Queens

1. What is the smallest n such that n be peacefully placed?

n = 0

n = 1, if there must be at least one Queen

n = 4, if there must be greater than one Queen

2. Write a recursive algorithm that either places the n Queen’s or determines that no such placement is possible.

recursiveStep(r, n):  
set a base case, where if r exceeds n then immediately return True (indicating successful queen placement)  
for n-times  
 a = check if there is a piece on the same row to the left  
 b = check if there is a piece on the upper left diagonal  
 c = check if there is a piece on the lower left diagonal  
 if a, b and c are false

place a queen at the current position  
 if recursiveStep(r+1, n) can place a queen  
 return that queens can be placed  
 if the program is here no queens was found, remove the earlier placed queen  
if no queens can be placed, return that no queen can be placed

nQueens(n):  
make a n x n game board   
if recursiveStep(0, n) returns false  
 no solution exists so return nothing  
otherwise return a filled out board

3. Modify the algorithm so that it counts all peaceful placements.

recursiveStep(r, n):  
set a base case, where if r exceeds n then immediately return True (indicating successful queen placement)  
for n-times  
 a = check if there is a piece on the same row to the left  
 b = check if there is a piece on the upper left diagonal  
 c = check if there is a piece on the lower left diagonal  
 if a, b and c are false

place a queen at the current position  
 if recursiveStep(r+1, n) can place a queen  
 return that queens can be placed  
 if the program is here no queens was found, remove the earlier placed queen  
if no queens can be placed, return that no queen can be placed

nQueens(n):  
make a n x n game board  
while there is a solution being generated

update variables in recursive step so its starting at different points every iteration, so it generates a different board every time

if recursiveStep(r, n) returns false

no solution exists, flag for while loop to end

otherwise store a filled out board