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: $O(V^2 + E)$
>
> For space complexity, I put down V^2 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

0 0 0 0 1 1 0 0 1 0 1 1 1 1 0 1

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.

A recursive algorithm that places nQueens can be implemented using backtracking.

1) Create an n x n board.
2) Start at the leftmost column.
3) Base Case: if n queens are placed, return true.
4) Otherwise, For each row in the current column:
   i. If the current queen can be placed in this row without conflict (no queens present to the right), then place the queen at the present row/column pair, and recursively confirm.
   ii. If a queen at the current column/row pair leads to a solution, return true.
   iii. If a queen at the current column/row pair does not lead to a solution, remove the queen. Restart at sub-step i. with a new row.
5) If there are no more rows to try and none of the previous attempts have led to a solution, return false.

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

1) Create an n x n board.
2) Start at the leftmost column.
3) Base Case: if n queens are placed, return true.
   - If the base case triggered, save the current board layout, and increment the number of peaceful placements by one.
4) Otherwise, For each row in the current column:
   i. If the current queen can be placed in this row without conflict (no queens present to the right), then place the queen at the present row/column pair, and recursively confirm.
   ii. If a queen at the current column/row pair leads to a solution, return true.
   iii. If a queen at the current column/row pair does not lead to a solution, remove the queen. Restart at sub-step i. with a new row.

5) If there are no more rows to try and none of the previous attempts have led to a solution, return false.