# FIT4010 Assignment 2

## Semester 2, 2014

Jonathan Poltak Samosir- 2271 3603

## Eulerian Graphs

### Task 1

#### Program specifications

The program for this task is written in Python 2.7 without any external libraries (all imports should be in the standard libraries in the Python 2.7 distribution, or utility classes written by myself).

Essentially, my program implements [Hierholzer's algorithm](http://en.wikipedia.org/wiki/Eulerian_path#Hierholzer.27s_algorithm) to find Eulerian cycles. Hierholzer’s algorithm works by finding a cycle from a given vertex, hence I have had to supply multiple start points to find multiple different circuits in the given graph.

#### Difficulties

Many difficulties were encountered in this part of the assignment. Most notably was finding every possible Euler circuit in a given graph. After looking around at some solutions to this problem, it was pretty clear that it’s not as simple as it sounds to calculate this number. The best solution seemed to be the BEST theorem (pun not intentional), which calculates the number of Euler circuits as per the following formula:

is apparently the number of *arborescences* in the graph G. In less crazy language, I understand, in my sleep-derived state, an arborescence to be the number of rooted trees in a given graph. The main thing to take away from it, is that the number of arborscences in a graph increases dramatically as the number of nodes does, as can be seen here:

[*http://oeis.org/A000081*](http://oeis.org/A000081)

Hence, the number of arborescences multiplied with the product of all the factorials of all the degrees in the graph would be rather large, and you’d end up with an extremely complicated algorithm to get the number of Euler circuits.

Instead, I have opted for a far simpler, but not very smart and full-coverage way of doing it. Basically, I have performed Hierholzer’s algorithm on a graph, starting at each vertex in the graph. Hence the algorithm is performed *n* times for an *n* vertex graph, returning *n* different Euler circuits. I then also take the reverse of each circuit, as they will also be Eurler circuits, ending up with *2n* Euler circuits. From these circuits, I can calculate the strings from the edge labels and using a longest common substring algorithm, return the longest length substrings that can be found from the provided sequence and all the Euler circuits that my algorithm implementation finds.

Note that I do not get every possible Euler circuit from the graph, hence the longest common substring returned is not always correct. **This is the major limitation of my supplied code.** Given more time, I would have attempted implementation of the BEST theorem and continued to permeate through different paths in the graph using Hierholzer’s algorithm until I found all possible Euler circuits, then attempted to find the longest common substring.

### Task 2

The following table presents the results of the previously described code run on each of the supplied graphs. Note that given the limitations I have specified above, the following answers (sadly) may not be correct.

|  |  |
| --- | --- |
| Graph file | Results |
| Fig1TEST | Eulerian Circuit: [0, 2, 4, 0, 3, 2, 1, 0]  Symbols on path: TGCCATC  Sequence: GTGCTCC  Matched Substring: TGC  Found sequence: 1-3 and subsequence: 0-2 |
| TEST1 | Eulerian Circuit: [0, 1, 6, 8, 9, 0, 3, 5, 4, 3, 7, 9, 2, 5, 8, 7, 6, 2, 1, 4, 0]  Symbols on path: ACGAGCTCAGTATCCCGTAA  Sequence: TTACGAGTAA  Matched Substring: ACGAG  Found sequence: 2-6 and subsequence: 0-4 |
| TESTA | Eulerian Circuit: [6, 1, 5, 2, 1, 3, 9, 0, 7, 9, 4, 0, 8, 4, 2, 8, 6, 3, 7, 5, 6]  Symbols on path: TGGTTAGAGTGAGTCGCTCA  Sequence: TTCCGCTCAC  Matched Substring: CGCTCA  Found sequence: 3-8 and subsequence: 14-19 |
| TESTB | Eulerian Circuit: [6, 4, 8, 0, 4, 5, 9, 7, 2, 3, 5, 6, 8, 7, 6, 9, 8, 1, 5, 2, 9, 3, 7, 0, 3, 4, 1, 2, 0, 1, 6]  Symbols on path: ACGTTAGCTCTACGCAGCGTTCGTCTTGTC  Sequence: TTAACGCCTT  Matched Substring: ACGC  Found sequence: 3-6 and subsequence: 11-14 |
| TESTC | Eulerian Circuit: [1, 7, 2, 9, 1, 4, 9, 3, 1, 8, 2, 6, 1, 2, 5, 9, 6, 5, 0, 4, 7, 6, 3, 0, 6, 8, 4, 3, 7, 5, 4, 2, 0, 9, 8, 3, 5, 8, 7, 0, 1]  Symbols on path: AGACGACTAGCCATATGAGTGGACGGAGGGCCGGAACGCC  Sequence: TTACGCCTGT  Matched Substring: ACGCC  Found sequence: 2-6 and subsequence: 35-39 |
| TESTD | Eulerian Circuit: [3, 4, 5, 2, 7, 8, 6, 0, 7, 4, 2, 1, 5, 9, 1, 6, 3, 8, 0, 9, 3]  Symbols on path: GACCACTGATCATGACATGC  Sequence: CCAC  Matched Substring: CCAC  Found sequence: 0-3 and subsequence: 2-5 |
| TESTE | Eulerian Circuit: [1, 5, 4, 8, 3, 0, 7, 2, 5, 7, 6, 3, 4, 9, 2, 6, 5, 9, 6, 8, 0, 4, 7, 1, 8, 9, 1, 3, 2, 0, 1]  Symbols on path: CAAGGCCAGCTGTCGGAATTTGGTCCTGGT  Sequence: CGCC  Matched Substring: GCC  Found sequence: 1-3 and subsequence: 4-6 |
| TESTF | ulerian Circuit: [0, 5, 3, 4, 0, 8, 1, 3, 8, 9, 4, 6, 5, 7, 4, 8, 5, 2, 8, 7, 6, 2, 1, 5, 9, 7, 0, 9, 2, 3, 0, 6, 1, 9, 6, 3, 7, 2, 4, 1, 0]  Symbols on path: CTTCCGGAGCCACAGCCGGAAACTGTGTTCAGCAGCAGGT  Sequence: GTTA  Matched Substring: GTT  Found sequence: 0-2 and subsequence: 26-28 |
| TESTG | Eulerian Circuit: [5, 2, 11, 10, 0, 4, 6, 1, 9, 13, 0, 3, 4, 5, 8, 9, 2, 7, 12, 8, 1, 13, 12, 6, 10, 3, 7, 11, 5]  Symbols on path: CCGAGGAATCAACTGCTGTGCGCCTAGA  Sequence: CATAGGATTC  Matched Substring: AGGA  Found sequence: 3-6 and subsequence: 3-6 |
| TESTH | Eulerian Circuit: [4, 10, 7, 11, 6, 8, 2, 6, 9, 0, 10, 12, 3, 7, 9, 13, 5, 1, 3, 13, 0, 2, 11, 8, 4, 1, 12, 5, 4]  Symbols on path: CCTCGGACCGTGCGTTGTAGGTCCTATG  Sequence: CGGG  Matched Substring: CGG  Found sequence: 0-2 and subsequence: 3-5 |
| TESTI | Eulerian Circuit: [1, 11, 13, 10, 4, 13, 3, 5, 4, 0, 13, 9, 10, 12, 4, 6, 11, 2, 0, 10, 8, 9, 2, 7, 11, 8, 7, 6, 0, 5, 12, 6, 3, 2, 12, 1, 5, 9, 7, 1, 8, 3, 1]  Symbols on path: CTAATGAGCTTCGTGAACTCGCATCATTAGCACGATTATCTC  Sequence: AGATGAGCGG  Matched Substring: ATGAGC  Found sequence: 2-7 and subsequence: 3-8 |
| TESTJ | Eulerian Circuit: [0, 10, 4, 2, 5, 13, 11, 0, 5, 8, 0, 13, 9, 4, 6, 2, 10, 11, 2, 9, 5, 12, 10, 6, 8, 3, 7, 4, 3, 13, 1, 3, 12, 6, 1, 11, 12, 9, 7, 8, 1, 7, 0]  Symbols on path: CAACAACAACACCACAACTACATGAGAGCTTTCAGCCAAATT  Sequence: GACC  Matched Substring: ACC  Found sequence: 1-3 and subsequence: 10-12 |
| TESTK | Eulerian Circuit: [3, 7, 1, 12, 3, 4, 2, 3, 9, 1, 0, 13, 12, 6, 9, 7, 6, 11, 0, 9, 13, 3, 8, 1, 10, 12, 0, 2, 7, 5, 8, 6, 4, 0, 7, 10, 8, 2, 11, 8, 12, 5, 9, 4, 11, 1, 6, 5, 2, 13, 11, 5, 10, 13, 4, 10, 3]  Symbols on path: ATCGGGCTCTCATAGTTACGATTGCGCATCGTTCATAATCACCTGACCATTCATAA  Sequence: AACAATCACC  Matched Substring: AATCACC  Found sequence: 3-9 and subsequence: 36-42 |
| TESTL | Eulerian Circuit: [13, 1, 8, 9, 12, 10, 9, 7, 8, 10, 0, 1, 10, 7, 3, 1, 5, 3, 0, 12, 13, 9, 5, 7, 6, 11, 12, 6, 2, 12, 5, 13, 6, 3, 8, 5, 11, 1, 4, 0, 6, 4, 13, 2, 0, 8, 11, 4, 2, 10, 4, 7, 2, 3, 9, 11, 13]  Symbols on path: CAATGATAGACGATTTCTCGAGTCATCACCCGACTGTTAGTCTATTTATCCCAGGC  Sequence: CCGA  Matched Substring: CCGA  Found sequence: 0-3 and subsequence: 29-32 |
| TESTM | Eulerian Circuit: [7, 3, 13, 18, 12, 1, 10, 9, 5, 16, 14, 9, 7, 5, 2, 17, 14, 19, 0, 17, 11, 12, 16, 3, 6, 2, 10, 6, 1, 0, 8, 4, 11, 13, 15, 8, 18, 4, 15, 19, 7]  Symbols on path: AGGTATCGATTATGGGCCTTTAGCACGAACTATTTCAATG  Sequence: TTATATGGGC  Matched Substring: TATGGGC  Found sequence: 3-9 and subsequence: 10-16 |
| TESTN | Eulerian Circuit: [1, 17, 13, 4, 8, 14, 16, 6, 10, 9, 5, 14, 18, 6, 0, 16, 19, 11, 15, 12, 11, 13, 2, 3, 15, 7, 3, 4, 7, 19, 10, 18, 1, 5, 2, 0, 17, 9, 12, 8, 1]  Symbols on path: GGGGGGGCGTCGTGGAGAGCATCATGATGATTCGAGCACG  Sequence: ATTA  Matched Substring: ATT  Found sequence: 0-2 and subsequence: 29-31 |
| TESTO | Eulerian Circuit: [11, 9, 12, 7, 3, 1, 15, 3, 19, 9, 8, 0, 2, 3, 10, 1, 9, 14, 1, 17, 7, 15, 13, 2, 19, 15, 17, 12, 14, 0, 15, 8, 19, 12, 6, 7, 11, 0, 16, 18, 1, 16, 8, 11, 13, 10, 12, 5, 0, 4, 7, 5, 18, 19, 16, 6, 8, 10, 9, 4, 6, 5, 14, 13, 17, 10, 18, 2, 11, 17, 4, 14, 6, 2, 5, 13, 18, 4, 3, 16, 11]  Symbols on path: ATAACTCAATAGCTCATTCAAGGGCTAAACGGACTTTACTATAGGCTATTCCGTACACTAGGCCTTTCTTGTTCTGTTTT  Sequence: AGGGCTCGTT  Matched Substring: AGGGCT  Found sequence: 0-5 and subsequence: 20-25 |
| TESTP | Eulerian Circuit: [6, 19, 18, 11, 13, 10, 4, 0, 14, 10, 15, 13, 14, 1, 2, 5, 4, 2, 11, 19, 13, 6, 5, 17, 6, 18, 7, 1, 13, 9, 5, 3, 15, 5, 8, 19, 10, 7, 3, 18, 0, 9, 1, 10, 8, 2, 6, 9, 17, 8, 4, 3, 17, 15, 19, 14, 16, 8, 0, 7, 14, 4, 12, 11, 16, 1, 18, 16, 3, 2, 12, 0, 11, 9, 15, 12, 17, 16, 7, 12, 6]  Symbols on path: TTTTTACCTATTGACTCATGATTCCGGTATTGCTTGTGCGGATCTTTAGAGCATGCGCGATCGAGTAGAACACATCCGGA  Sequence: TTAG  Matched Substring: TTAG  Found sequence: 0-3 and subsequence: 45-48 |
| TESTQ | Eulerian Circuit: [2, 13, 1, 19, 5, 6, 13, 5, 18, 11, 13, 10, 14, 13, 18, 15, 13, 7, 0, 2, 11, 14, 18, 4, 9, 14, 6, 19, 9, 5, 4, 16, 5, 11, 8, 19, 16, 6, 4, 2, 19, 17, 14, 15, 5, 10, 4, 8, 5, 12, 0, 5, 2, 1, 15, 11, 4, 3, 5, 7, 6, 2, 10, 11, 6, 15, 16, 10, 18, 19, 14, 8, 10, 12, 19, 4, 14, 3, 10, 17, 2, 3, 13, 19, 10, 6, 9, 17, 0, 10, 15, 8, 16, 18, 6, 17, 11, 7, 10, 9, 11, 3, 7, 14, 1, 5, 17, 8, 6, 12, 11, 16, 14, 2, 15, 3, 18, 17, 13, 4, 15, 19, 7, 15, 12, 14, 0, 18, 9, 7, 17, 15, 9, 8, 3, 16, 7, 18, 8, 12, 3, 19, 0, 13, 8, 0, 3, 9, 0, 4, 7, 12, 9, 2, 12, 1, 11, 0, 6, 1, 18, 12, 17, 16, 12, 4, 1, 17, 3, 1, 9, 13, 16, 2, 7, 1, 0, 16, 1, 8, 2]  Symbols on path: AAAACCGATACTGGTTACCCCGTGCATAAAAGAGTAACCTATACTTTCAGTTAGGCACTCCAGAAACGCGCAGACGTATCGACGGCCCTGGACCTAATGAATAGTGACGCCGGTTGATCGTCTCTGTCCTACTAACTACACCTCAAGCAGACTGGCATGCATTGTGCCACCCAGTCAGTC  Sequence: CTGG  Matched Substring: CTGG  Found sequence: 0-3 and subsequence: 10-13 |
| TESTR | Eulerian Circuit: [13, 0, 18, 8, 10, 16, 23, 22, 14, 19, 27, 10, 26, 5, 3, 2, 7, 28, 4, 3, 8, 29, 7, 12, 16, 20, 14, 6, 21, 2, 22, 15, 6, 9, 12, 11, 24, 1, 18, 11, 21, 25, 13, 1, 28, 17, 24, 25, 20, 29, 27, 5, 19, 9, 0, 4, 17, 23, 15, 26, 13]  Symbols on path: AATTCCCTGCCCTCTCAGTCAGAGAAGCTTTTACATACCATAGTCCTTCGAGATACCGAA  Sequence: TATGTACCAC  Matched Substring: TACCA  Found sequence: 4-8 and subsequence: 35-39 |
| TESTS | Eulerian Circuit: [11, 16, 18, 14, 23, 8, 7, 19, 8, 29, 15, 13, 3, 22, 20, 3, 1, 28, 22, 27, 18, 12, 28, 19, 0, 23, 21, 20, 4, 9, 5, 15, 26, 16, 24, 6, 27, 2, 25, 13, 7, 24, 11, 10, 0, 17, 5, 12, 17, 10, 21, 2, 4, 26, 29, 1, 6, 14, 9, 25, 11]  Symbols on path: ATCTCCGCTCACGGGTGATTATCTTCGAGTGAACCCTAGCGTGAACGTTAGGTTTCCACC  Sequence: AACC  Matched Substring: AACC  Found sequence: 0-3 and subsequence: 31-34 |
| TESTT | Eulerian Circuit: [18, 11, 8, 21, 17, 27, 10, 9, 5, 4, 23, 27, 2, 7, 1, 10, 18, 22, 11, 25, 21, 15, 25, 28, 16, 0, 3, 1, 24, 29, 14, 17, 16, 19, 4, 9, 7, 13, 3, 26, 12, 29, 13, 5, 22, 12, 8, 20, 14, 26, 24, 6, 23, 28, 6, 20, 15, 19, 2, 0, 18]  Symbols on path: TGGGAACGTGAGATACCTAGCTCTGTTCTGTTATGGCATTGTATATAGGCAGCGGCCTCT  Sequence: ACCTAGGAAC  Matched Substring: ACCTAG  Found sequence: 0-5 and subsequence: 14-19 |

As talked about in the **Difficulties** section, the results may or may not be correct due to my program not generated all possible Euler circuits. Checking with another student who’s program also did not generate all possible Euler circuits, our results often varied, some of mine were better, some were worse, and some were the same result. This all depends on however our implementations decided to choose edges at each step to add to the Euler path, and they either had different heuristics or the same heuristic but different underlying data structures.

### Task 3

##### Question 1

Assuming that G always has a Eulerian circuit, then yes, the trail can be extended to form an Eulerian circuit. A graph containing an Eulerian circuit has the properties that every vertex in the graph will have an even degree.

##### Question 2

##### Question 3

## Eulerian Graphs

### Task 1

#### Program specifications

The program for this task is written in Python 2.7 without any external libraries (all imports should be in the standard libraries in the Python 2.7 distribution, or utility classes written by myself).

The program essentially implements the Ford-Fulkerson algorithm that calculate the max-flow of a directed graph with edge capacities from a source and sink vertices. As per assignment specifications, the source and sink vertices are the first vertex (0) and final vertex (n) respectively.

#### Difficulties

The main difficulty faced with this task was that I implemented my Ford-Fulkerson algorithm using a recursive method to traverse the graph edges and find paths. Often, when the supplied graphs were relatively large, the implementation would fail, as Python’s recursion limit would be reached. Hence, what I had to do was manually override Python’s default recursion limit, and set it to a large constant. So far, this has allowed the larger graphs to be attempted.

### Task 2

The following table presents the results of the previously described code run on each of the supplied graphs.

|  |  |
| --- | --- |
| Graph file | Max flow calculated |
| G1 | 2 |
| G2 | 17 |
| G3 | 0 (no way to get from source to sink) |
| G4 | 59 |
| G5 |  |
| G6 |  |
| G7 |  |
| G8 |  |
| G9 |  |
| G10 |  |
| G11 |  |
| G12 |  |
| G13 |  |
| G14 |  |
| G15 |  |
| G16 |  |
| G17 |  |
| G18 |  |
| G19 |  |
| G20 |  |
| G21 |  |
| G22 |  |
| G23 |  |
| G24 |  |
| G25 |  |

### Task 3

Note that

### Task 4

### Task 5

As discussed in Task 1, I have implemented the Ford-Fulkerson algorithm to find max flow. The actual Python implementation can be found in the **maxflow.py** file. As can be seen in the file, the algorithm is decomposed into two separate parts: a recursive path finding method, **\_findPath()**, and the calling max flow method, **maxFlow()**.

**\_findPath():**

This method works by giving it a source vertex and a destination vertex, it will recursively find a path from the source to the destination. At each recursive call, it will attempt to call itself for every outgoing edge from the current vertex, and will add that edge to the path. If in one of the calls, it realises it is stuck, and there is no way out of the current vertex, it will return None to the caller, which is then ignored and another edge is attempted.

In the case of a graph where there is no path from the source to the sink (such as G3), it will realise this after all edges are exhausted and end up returning None. This is then handled in the **maxFlow()** method.

The recursive base case is when the current vertex equals the destination vertex (the sink). This means that a path has been found and the path that has been generating at each recursive call is returned back up the call stack.

This path finding algorithm has a worst-case time complexity of the number of edges in the graph, hence , where E is the number of edges.

Note that when choosing edges to traverse from the current vertex, this algorithm does not consider those edges where the flow is greater or equal to the current capacity, or where the edge has already been traversed (to avoid cycles). This is most important for the next method.

**maxFlow():**

This method works by using the **\_findPath()** algorithm to get paths from the source to the sink, of which it uses to update the edge capacity and flows for each of the edges in the found path using the minimum value found in the calculated residuals. This is the heart of the Ford-Fulkerson algorithm, and will essentially continue to update edge capacities and flows in the graph while there are still unprocessed paths remaining in the graph.

The way it avoids getting into an infinite loop when it continues to get the paths is that the **\_findPath()** algorithm takes into account the current edge capacities and flow when adding edges to the path. Hence, if the current capacity of the edge is insufficient, it will not consider that edge for choosing.

Finally, to actually get the max flow of the graph, the sum of all the flows from outgoing edges from the source is computed. The same thing could be gotten from ingoing edges to the sink.

The worst-case time complexity for this algorithm is the complexity of **\_findPath()** multiplied by the max flow of the graph, as **\_findPath()** will continue to be run until the max flow is calculated (after all the edge capacities have been minimised). This leaves us with .

##### Question 1

##### Question 2

##### Question 3

##### Question 4

##### Question 5