PROBLEM SET 1 - SOLUTIONS
COMS W1004 Fall 2016
30 Points

S&G 7th edition: 1.10, 1.11, 2.18. 2.25, 3.31, 3.36

Problem 1.10: 5 points

a) The final output of the algorithm is 2.

Step	I	J	R
Initial	30	22	
1	30	22	8
2	22	8	6
3	8	6	2
4	6	2	0

b) The algorithm will fail because of a divide by 0 error. See the table below. The algorithm can be fixed by checking if J=0 between the first and second step. If J is 0, the algorithm should print an error message and stop.

Step	I	J	R
Initial	12	0	
1	12	0	ERROR

Problem 1.11: 5 points

This algorithm is very computationally intensive. That is, we must find all possible combinations of routes between 20 cities, such that we have the minimum distance travelled overall, and that we visit each city only once. Thus, we start with a choice of any one of the 20 cities. From there, we can continue to any of the 19 remaining cities (assuming all cities are interconnected). From this point, we continue onwards until we have only one city left to visit. Determining the number of paths can be represented as:

$$20!\ paths\ =\ 20\ \cdot\ 19\ \cdot\ 18\ \cdot\ ...\ \cdot 1 \approx 2.43\ \times 10^{18}\ paths$$

From here, we can divide the total number of paths by the rate at which the computer analyzes paths,

$$\frac{2.43 \times 10^{18}\ paths}{10{,}000{,}000\ paths/sec} = 2.43 \times 10^{11} sec\ \times \frac{1\ min}{60\ sec} \times \frac{1\ hour}{60\ min} \times \frac{1\ day}{24\ hours} \times \frac{1\ year}{365\ days}$$
$$= 7.71\ \times 10^{3} years$$

As we can see, this algorithm will take a very long time. Therefore, this algorithm is not feasible for use on any modern machine.

However, there are other algorithms we can implement that can obtain a reasonable solution to this problem. One such algorithm is Dijkstra's algorithm (you will learn this algorithm later in the semester). Below is a greedy algorithm, similar to Dijkstra's algorithm, that follows the domain of the problem.

Step 1	Create an empty set for visited cities.
Step 2	Pick any city at random from the n cities, and add that city to the set of visited cities.
Step 3	Of all the roads between the $n-1$ other cities and the current city, pick the road that is the minimum driving distance between both cities.
Step 4	Check if the city on the selected road has already been visited, if it has ignore this road and choose the next smallest distance. If it has not been visited, add the city to the set of visited cities.
Step 5	Repeat Step 3 and 4 until all the cities have been added to the set of visited cities.

The algorithm picks the shortest path between the current city and any of its unvisited neighbors until all the cities in the graph of cities has been added to the set of visited cities. We are attempting to minimize the total distance travelled; however, there is a possibility that the algorithm does not minimize the total distance. Thus, we have an approximation of the minimum distance. In most

cases, this algorithm will generate an answer close to the minimum, but in a few cases, the algorithm could perform poorly. The overall running time of this algorithm will depend on how it is implemented. However, theoretically we can see that the runtime is $O(n^2)$ where n is the number of cities within the problem. Thus, we have

$$\frac{20^2\ paths}{10,000,000\ paths/sec} = \frac{1}{25000}\ sec$$

The running time here is definitely an improvement over the running time of the previous algorithm.

Problem 2.18: 5 points

a) The output is:

```
There is a match at position 10
There is a match at position 23
There is a match at position 28
```

b) One possible modification to the algorithm is to add the two checks.

i) First, when the first character of the pattern, P_1, matches the character in the string $T_{k+(i-1)}$, check that $T_{k+(i-1)}$ is the first character in the string or that the previous character, $T_{k+(i-1)-1}$, is a space or a punctuation mark.

ii) Second, when the last character of the pattern, P_m, matches the character in the string $T_{k+(i-1)}$, check that Tk is the last character in the string or that the next character, $T_{k+(i-1)+1}$, is a space or a punctuation mark.

Problem 2.25: 5 points

Pseudocode is as follows:

```
V = sequence
i = 0
while i < (length of V - 1) AND V[i] is not -1
     If V[i] equal to V[i+1]
          Print "Yes"
          Stop
Print "No"
```

Problem 3.31: 5 points

a) The values are:

i) $floor(1.2) = 1$

ii) $floor(2.3) = 2$

iii) $floor(8.9) = 8$

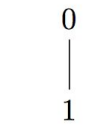
iv) $floor(-4.6) = -5$

b) The completed table:

n	$floor(lg\ n)$
2	1
3	1
4	2
5	2
6	2
7	2
8	3

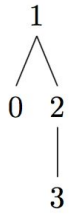
c) The completed table and trees:

n	Number of Compares, Worst Case
2	2
3	2
4	3
5	3
6	3
7	3
8	4

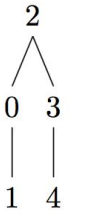


For $n = 2$

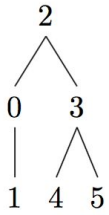
For $n = 3$



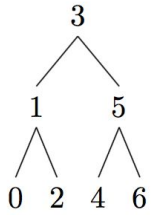
For $n = 4$



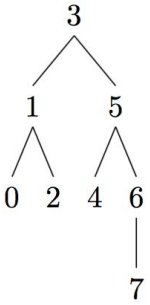
For $n = 5$



For $n = 6$



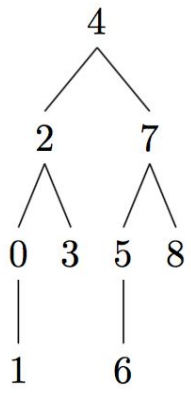
For $n = 7$



For $n = 8$

d) The formula is $f(n) = floor(lg\ n) + 1$. Testing for $n = 9$ we have:

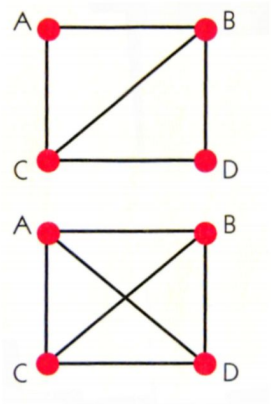
$$f(9) = \lfloor \lg 9 \rfloor + 1 = 4$$



For $n = 9$

Problem 3.36: 5 points

a) The top graph has an Euler path, one possible is B-D-C-A-B-C. The bottom does not have a Euler path.

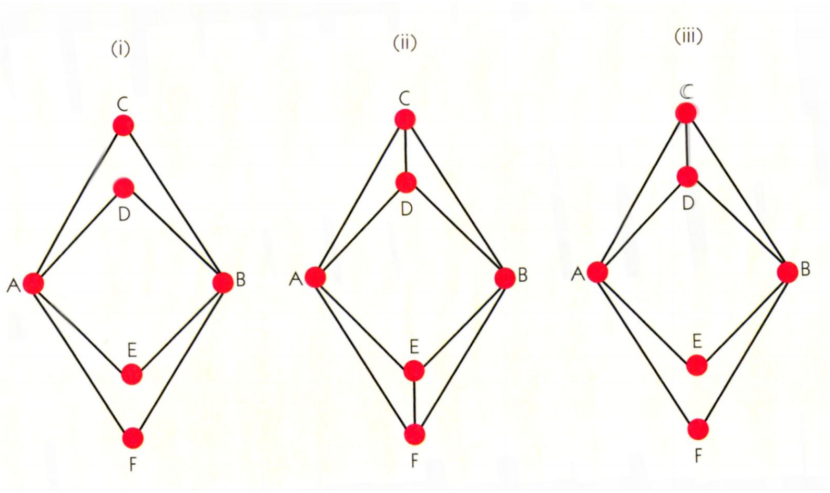


b) As follows:

i) There are 0 odd nodes, so a Euler path exists in this graph. One such is C-A-F-B-D-A-E-B-C.

ii) There are 4 odd nodes, so a Euler path does not exist in this graph.

iii) There are 2 odd nodes, so a Euler path exists in this graph. One such is C-A-F-B-D-C-B-E-A-D.



c) The order of magnitude of this algorithm where the “work unit” is checking whether an edge exists is $O(n^2)$. The outer while loop will iterate over all n nodes in the graph, while the inner while loop will iterate n times.

d) The Euler path problem is not intractable. Intractable problems are defined as problems that can be solved in theory, but in practice take too long to be useful -- that is, problems that are super-polynomial. But the algorithm given above solves the Euler path problem in polynomial time.