Homework 3

Kevin Martin
CIS675 - Syracuse University

February 9, 2020

1. Question 3

   (a) Pseudocode for a greedy algorithm to compute optimal order and minimize wait time:

   Sort the service time in ascending order, and serve them in order of increasing scheduling times.
   Let n be the list of customers that need to be solved, with each time required t_i as the relevent index in the array:

   def greedySort(n [])
   sort(n); // in ascending order
   currentJob = n[1]
   for i = 1 to n:
       currentJob = currentJob + t_i

   (b) Claim: the running time of the proposed algorithm is $O(nlogn)$

   Proof. Sorting the list to get the algorithm started is where most of the time is required. Sorting a list of n elements takes $O(nlogn)$ time. The rest of the algorithm can be completed in constant time, $O(1)$. Because the algorithm can be written as $c_1 + (c_1 + c_2) + (c_1 + c_2 + ...c_n)\frac{1}{n}$, we can see that c_1 repeats itself the most times. As such, because it is the shortest time, then no other ordering could be correct. Therefore the greedy strategy holds true. ■

2. Question 5

   (a) To represent the situation as a linear problem, we formulate as follows, letting x_1 be coffee mugs and x_2 be milk glasses:

Objective function	max $25x_1 + 20x_2$
Constraints	$20x_1 + 12x_2 \leq 1800$
	$x_1/15 + x_2/15 \leq 8$
	$x_1, x_2 \geq 0$

(b) Graph of feasible region:

(c) The coordinates of all vertices of the feasible region are:
(0, 0), (90, 0), (45, 75), (0, 120)

(d) The optimal product mix to maximize daily proift is:
45 coffee mugs at \$ 25 each and 90 milk glasses at \$ 20 each gives a total profit of \$2,625 per day. This is represented on the graph but the furthest out point on on the feasible region, represented by the tangential dotted line.

3. Question 3

(a) The adjacency matrix representation:

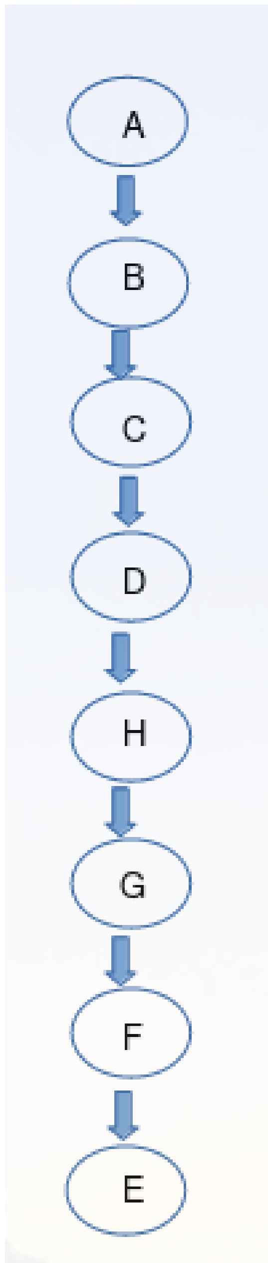
	A	B	C	D	E	F	G	H
A	0	1	0	0	0	1	0	0
B	0	0	1	0	1	0	0	0
C	0	0	0	1	0	0	0	0
D	0	1	0	0	0	0	0	1
E	0	0	0	1	0	0	1	0
F	0	0	0	0	1	0	1	0
G	0	0	0	0	0	1	0	0
H	0	0	0	0	0	0	1	0

(b) The adjacency list representation:
$A : B \rightarrow F$
$B : C \rightarrow E$
$C : D$
$D : B \rightarrow H$
$E : D \rightarrow G$
$F : E \rightarrow G$
$G : F$
$H : G$

(c) Table for intermediate visited, pre, and post values of all nodes:

	visited(v), pre(v), post(v) arrays																								
	v	v	pr	po	v	pr	po	v	pr	po	v	pr	po	v	pr	po	v	pr	po	v	pr	po	v	pr	po
A	0	1	1	16																					
B	0					2	15																		
C	0								3	14															
D	0											4	13												
H	0														5	12									
G	0																	6	11						
F	0																				7	10			
E	0																							8	9
	time																								

(d) Final DFS tree:



4. Question 4

   Pseudocode of an efficient algorithm to take in graph G(V, E), represetned by an adjacency list, and two vertices $x, y \in V$, where the output is the different paths from x to y in G:

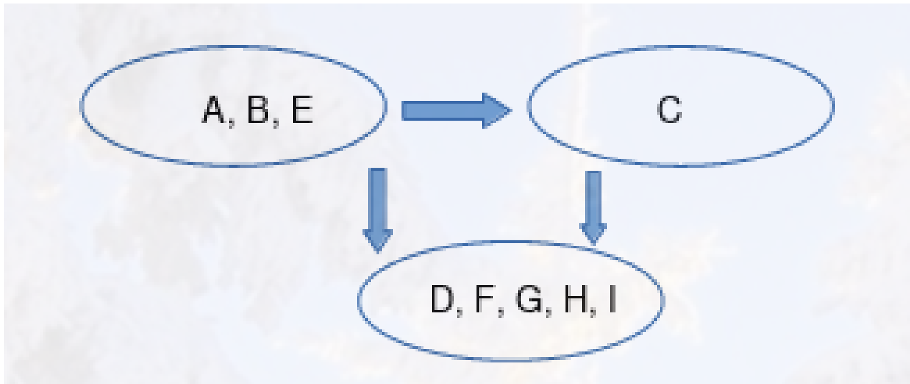
```
//wrapper function to set empty path
def diffPathsWrap(x, y, G)
for i = 1 to (size-1) do
        visited[i] = False
        path = [] //empty array for each path
diffPaths(x, y, G, path)



def diffPaths(x,y, visited, path)
visited[x]=True
if x == y
        return path
else
        for j in G[x] do
                if visited[j] == False
                        return diffPaths(x, y, visited, path)
                path[j].remove //remove current vertex from path
                visited[j] = False
```

5. Question 5

   (a) The order of the strongly connected componets is:
   $A \to B \to E$
   C
   $D \to H \to F \to I \to H \to D$

   (b) The source SCC is $A \to B \to E$ and the sink SCC is $D \to H \to F \to I \to H \to D$

(c) Metagraph:



(d) The minimum number of edges one must add to make the graph strongly connected is just one: from $D \to G$. If that edge were added, then we could get to any vertex from any other.