cpts 515 Take-home Final Exam

Requirements: 1. Please type your solutions and submit the PDF, 2. You must work on your own while collaboration is prohibited, 3. If you use internet resources, please cite the source. If you have questions in understanding the problems, you may call or text or email me at 509-338-5089. But I will not comment on your solutions.

By definition, an algorithm is deterministic and must halt on all inputs. If your algorithm involves probability, involves nondeterminism, and/or doesnt halt on all inputs, your algorithm is not even an algorithm! We have special names for algorithms that use probabilities: randomized algorithms. We have special names for algorithms that use nondeterminism: nondeterministic algorithms.

As I said many times during the lectures, nowadays, when we design algorithms, we almost always assemble those classic and famous algorithms like Dijkstra's shortest path algorithm together in a creative way so that we can come up with a solution. In this case, you dont need to repeat the details of those famous algorithms. Instead, you can simply use them as a library function call.

Let G be a directed graph where each node has a color and has a positive integer weight. Multiple nodes can share the same color and/or the same weight. In particular, there is a designated start node and there is a designated end node. A walk α is a path of the graph that starts with the start node and ends with the end node. Notice that a walk may have an unbounded length; e.g., a walk can contain (nested) loops.

Consider a walk $\alpha = v_1 \cdots v_k$ for some k in G. That is, as we have said, the v_1 is the start node, the v_k is the end node, and for each $1 \leq i < k$, $\langle v_i, v_{i+1} \rangle$ is an edge in G. The weight $W(\alpha)$ of the walk α is the sum of all weights of all nodes on the walk; i.e.,

$$W(\alpha) = w(v_1) + \cdots + w(v_k)$$

where we use $w(v)$ to denote the weight of node v in G.

We now consider three colors: red, yellow, green. Notice that some nodes of G may have other colors like blue or purple. For the walk α, we use $\#_{red}(\alpha), \#_{yellow}(\alpha)$ and $\#_{green}(\alpha)$ to denote respectively the number of red nodes, the number of yellow nodes, and the number of green nodes on the

walk α. The walk α is *good* if, on the walk, all red nodes are before all yellow nodes, and all yellow nodes are before all green nodes. The walk α is *balanced* if $\#_{red}(\alpha) = \#_{yellow}(\alpha) = \#_{green}(\alpha)$.

Prob 0 (20%). Recall that an efficient algorithm is one that runs in deterministic polynomial time.

- (10%) Design **an efficient algorithm** to decide whether there is a good walk α in G.
- (10%) In case when there is indeed such a good walk in G, design **an efficient algorithm** to find a good walk in G with minimal weight.

For this problem, your algorithms shall be clearly presented so that there is no confusion if one chooses to implement it. Your solutions must start with explaining in English the intuition behind your algorithms and then are followed by describing the algorithms themselves. You dont have to write code/psuedo code to describe your algorithms. Instead, you may choose to clearly describe your algorithms in English.

Prob 1 (30%).

- (15%) Design **an efficient algorithm** to decide whether there is a good and balanced walk α in G.
- (15%) In case when there is indeed such a good and balanced walk in G, design **an efficient algorithm** to find a good and balanced walk in G with minimal weight.

For this problem, your algorithms shall be clearly presented so that there is no confusion if one chooses to implement it. Your solutions must start with explaining in English the intuition behind your algorithms and then are followed by describing the algorithms themselves. You dont have to write code/psuedo code to describe your algorithms. Instead, you may choose to clearly describe your algorithms in English.

Prob 2 (30%).

- (15%) Design an algorithm to decide whether there is a balanced walk in G.

- (15%) In case when there is indeed such a balanced walk in G, design an algorithm to find a balanced walk in G with minimal weight.

For this problem, your algorithms shall be sketched so that one can understand the ideas behind your algorithms and one is convinced that your algorithms work on all graphs (instead of a small example graph).

Prob 3 (20%). Take one problem from the above three problems as an example to describe an application in Computer Science that directly uses the algoithms or their generalized forms in the problem. Your description shall be clear and relevant and shall take roughly one page.

For problems 0, 1, and 2, I will grade on your algorithms' correctness and efficiency as well as the clarity of your presentation. For problem 3, if your write up is messy, irrelevant, and/or fundamentally flawed, I will only give you a small partial credit to the problem.

If your algorithm is not correct (e.g., has a major flaw that can not be fixed eaily), at most 50% partial credit (out of the maximal possible score of the algorithm) will be given to the algorithm.

In case when an efficient algorithm is required and your algorithm is actually not efficient, at most 50% partial credit (out of the maximal possible score of the algorithm) will be given to the algorithm under the condition that your algorithm is actually correct.

Final Exam

Sheryl Mathew (11627236)

December 14, 2018

1. **Problem 0**

a. **Does Good Walk Exist**

Intuition:

Algorithm:

Given: Consider a directed graph G with n nodes. Each node has a color and a positive weight.

Let: isGoodWalk be a Boolean value which tells whether there exists a good walk or not. Whenever there isGoodWalk is False, the function terminates there itself and return the value isGoodWalk.

Step 1: Let the start node v_1 be of color blue with some positive weight and the end node v_k be of color purple with some positive weight. If no blue node exists then add a new node v_1 with color blue and weight 0. If no purple node exists then add a new node v_k with color purple and weight 0. This will now be the graph G

Step 2: Make 3 different copies of the above graph G - G_1, G_2, G_3

Step 3: Set $Q_1 = \{v_1, v_k, yellow, green\}$. Traverse through the graph G_1 and drop all the nodes in G_1 which are present in Q_1 except v_1, v_k since they are the start and the end nodes. This will give a graph with nodes which contains red nodes but not yellow and green nodes.

Step 4: Set $Q_2 = \{v_1, v_k, red, green\}$. Traverse through the graph G_2 and drop all the nodes in G_2 which are present in Q_2 except v_1, v_k since they are the start and the end nodes. This will give a graph with nodes which contains yellow nodes but not red and green nodes.

Step 5: Set $Q_3 = \{v_1, v_k, red, yellow\}$. Traverse through the graph G_3 and drop all the nodes in G_3 which are present in Q_3 except v_1, v_k since they are the start and the end nodes. This will give a graph with nodes which contains green nodes but not red and yellow nodes.

Step 6: Glue the v_k of G_1 which is a purple node with the v_1 of G_2 which is a blue node. Glue the v_k of G_2 which is a purple node with the v_1 of G_3 which is a blue node. This

will generate a graph G'. The node v_1 of G_1 will be the start node of G' and the node v_k of G_3 will be the end node of G'

Step 7: Perform Depth-First-Search over G' till we reach v_k starting from v_1. Whenever we reach v_k set isGoodWalk = True. This is because G' will contain red nodes followed by yellow and green nodes. If we never reach v_k then set isGoodWalk = False since no path exists from v_1 to v_k

b. Good Walk with Minimum Weight

Step 1: Check the value of isGoodWalk from Problem 0 part (a). If isGoodWalk = True then proceed to Step 2. If isGoodWalk = False then return "No good walk with minimum weight exists"

Step 2: Convert the graph G' in such a way that for all the edges in G', assign the weight on the edges with $w(v)$. $w(v)$ is the weight on node v. Add a new node $v_k{}'$ f weight $w(v_k)$ and add an edge from the end node $v_k{}'$ to G'.

Step 3: Using Dijkstra's algorithm to find the shortest path of G' from v_1 to v_k. This will be the good walk with minimum weight.

2. Problem 1

a. Does Good and Balanced Walk Exist

Given: Consider a directed graph G with n nodes. Each node has a color and a positive weight.

Let: isGoodBalancedWalk be a Boolean value which tells whether there exists a good and balanced walk or not. Whenever there isGoodBalancedWalk is False, the function terminates there itself and return the value isGoodBalancedWalk.

Step 1: Check the value of isGoodWalk from Problem 0 part (a). If isGoodWalk = True then proceed to Step 2. If isGoodWalk = False then return "No good and balanced walk exists".

Step 2: Find if path exists between v_1 and v_k using DFS in O(V+E) time.

Step 3: Create a hash list in this case to hold multiple hashes.

Step 4: Create three hash tables to hold the number of red nodes, green nodes and yellow nodes on a given path.

Step 5: Create a hash table of the paths with key as paths and the value will be the below mentioned hash table with the red, green and yellow nodes as keys and the count of each as values.

a. Use a hash list to store the hashes we will be using in the algorithm.
b. One hash we use will be a hash table to keep track of paths.
c. Use another hash table to keep track of nodes that were previously in a path from v to v_k.
d. Use an array to keep track of temporary paths during DFS.
e. Use an integer variable to keep count of number of paths.
f. As each leftmost node is visited during DFS, the following are possible:
    - If the node is v_k, increment the number of paths by one.
    - If the node is not v_k, newly visited and not adjacent to a node that is part of another path, add node to temporary path array. Continue DFS.
    - If the node is not v_k, newly visited and adjacent to a node that is part of another path, increment the number of paths by one, concatenate path sequence in temporary path array and continue DFS.
g. At the end of such DFS operation, one hash table will contain all possible paths from v to v_k and the integer value will contain the total number of paths from v to v_k.

Step 6: Create an integer value to count the number of "good paths."

Step 7: While visiting each node during DFS, the following are the possibilities:

a. If the node is v_k, add the node at the end of the temporary path array, increment the color values for Red, Green and Yellow and check the hash tables with Red, Green, Yellow values.
    1. If Green count = Yellow count = Red Count in hash-table, add the temporary path in progress to the Paths hash table and increment the good paths variable.
    2. Otherwise, do nothing.
b. If the node is not v_k, newly visited and not adjacent to a node that is part of another path, increment the color values for Red, Green and Yellow and add node to temporary path array. Continue DFS.
c. If the node is not v_k, newly visited and adjacent to a node that is part of another path, add the node at the end of the temporary path array, concatenate the rest of the path onto the temporary path array, add the temporary path in progress to the Paths hash table and increment the good paths variable.

Step 8: At the end of DFS, the good paths variable will contain the total number of "good paths" between v and v_k. Along with the hash table with the best walks.

Step 9: If the hash table is empty then isGoodBalancedWalk = False else isGoodBalancedWalk= True

Since both the above algorithms, the use of hash lists and hash maps along with DFS ensures linear time complexity.

b. Good and Balanced Walk with Minimum Weight

Step 1: Check the value of isGoodBalancedWalk from Problem 1 part (a). If isGoodBalancedWalk = True then proceed to Step 2. If isGoodBalancedWalk = False then return "No good and balanced walk with minimum weight exists"

Step 2: Convert all the paths in the hash table in such a way that for all the edges in the path, assign the weight on the edges with $w(v)$. $w(v)$ is the weight on node v. Add a new node v_k' and add an edge of weight $w(v_k)$ from the end node v_k' to the path

Step 3: For first path in the hash table. Calculate the sum of all the weights in the path.

Step 4: Store the path in variable "best_path" and the weight in "min_weight".

Step 5: For second path in hash table. .Calculate the sum of all the weights in the path and store in temp_weight and the path in temp_path.

Step 6: Compare the weight obtained for second path with min_weight. If the value in temp_weight < min_weight. Update min_weight with temp_weight and best_path with temp_path. Else do nothing

Step 7: For all the paths in the hash table repeat Step 5 and 6.

Step 8: The value of best_path at the end of the algorithm will be the good and balanced walk with minimum weight.

3. Problem 2

a. Does Balanced Walk Exist

Given: Consider a directed graph G with n nodes. Each node has a color and a positive weight.

Let: isBalancedWalk be a Boolean value which tells whether there exists a balanced walk or not. Whenever there isBalancedWalk is False, the function terminates there itself and return the value isBalancedWalk.

Step 1: Let graph G be a Deterministic Finite-state Automata (DFA). From Problem 1 (a) we have all the possible balanced walks along with the number of red, yellow and green nodes.

Step 2: A regular expression can be constructed from G by passing all the possible balanced walks from Step 1. We collapse nested Kleene stars in a regular expression to create the linear programming problem.

Step 3: Solve the linear programming problem as follows: X is the number of red nodes, Y is the number of yellow nodes and Z is the number of green nodes.

x,y,z can repeat any arbitrary number of times in the code with the constraint that the total number of equations are limited by the number of lines of code. Since 2 lines of code are already taken, we have maximum 8 lines of code to write to the equations.

We can rewrite all equations involving y and z in terms of x. This is because we assume a linear relationship between x, y and z.

There are total 10 lines in the code, two lines are just declaring variables and returning x, so in the remaining 8 lines we simply assign x some arbitrary values of x1 through x7.

Write them in the following form:

X = a1.x1+b1.x2+c1.x3+....+g1.x7 + k1
X= a2.x1+b2.x2+c2.x3+.....+g2.x7 + k2
X= a3.x1+b3.x2+c3.x3+.....+g3.x7 + k3
. . .
. . .
. . .
. . .
X= a7.x1+b7.x2+c7.x7+.....+g7.x7 + k7
X=a8.x1+b8.x2+c8.x3+.....+g8.x7 + k8

These could also be written as

X -a1.x1-b1.x2-c1.x3....-g1.x7 = k1
X -a2.x1-b2.x2-c2.x3 ...-g2.x7 = k2
X -a3.x1-b3.x2-c3.x3....-g3.x7 = k3
. . .
. . .
. . .
X-a7.x1-b7.x2-c7.x3.....-g7.x7 = k7
X-a8.x1-b8.x2-c8.x3….-g8.x7 = k8

Where k1...k8 are some arbitrary constants.

We would like to solve these set of equations but we have no idea if the origin is a feasible point or not, since this is an arbitrary set of equations.

Also the constraint is $x<0$.

<u>Converting to linear optimization problem:</u>

To convert this to a linear optimization problem, we introduce artificial variables called M1, M2 through M8 to each of the equations respectively in the following manner.

We must have an objective function Min (M1+ M2+M3+....M8) subject to constraints below, where a1...a8, b1..b8, c1...c8 etc are some arbitrary coefficients in the equation :

X -a1.x1-b1.x2-c1.x3....-g1.x7 + M1 = k1
X -a2.x1-b2.x2-c2.x3 ...-g2.x7 + M2 = k2
X -a3.x1-b3.x2-c3.x3....-g3.x7 + M3 = k3
. . .
. . .
. . .
X-a7.x1-b7.x2-c7.x3.....-g7.x7 + M7 = k7
X-a8.x1-b8.x2-c8.x3….-g8.x7 + M8 = k8

Where M10, M2 0, M3 0….M8 0.

Thus we must solve an LP with starting point with M1 = k1, M2=k2, M3=k3….M8=k8 and X=x1=x2=x3…=x7 = 0.

We perform linear programming using Simplex method to get a return value for Min (M1+M2+M3+...M8), subject to all above constraints.

Repeat LP on the above equation until we get return value for Min (M1+M2+M3+....M8).

If return value for Min (M1+M2+M3+...M8) is zero, set isBalancedWalk = True, else isBalancedWalk = False

b. Balanced Walk with Minimum Weight

Step 1: Check the value of isBalancedWalk from Problem 3 part (a). If isBalancedWalk = True then proceed to Step 2. If isBalancedWalk = False then return "No balanced walk with minimum weight exists"

Step 2: Convert the graph G' in such a way that for all the edges in G', assign the weight on the edges with $w(v)$. $w(v)$ is the weight on node v. Add a new node $v_k{}'$ f weight $w(v_k)$ and add an edge from the end node $v_k{}'$ to G'.

Step 3: Using Dijkstra's algorithm to find the shortest path of G' from v_1 to v_k. This will be the balanced walk with minimum weight.

4. Problem 3

Problem Selected: Problem 0 part (a) – Nurse Scheduling System

Problem Statement: Consider a pediatric ward which provides 24 x 7 Services. The head nurse is in charge of scheduling nurses to staff shift. This process of allocating nursing staff is a very important task in hospital management. This is because it is necessary to have the right skill mix and number of staff to each shift. This scheduling policy has a direct impact on satisfaction of the nurses and therefore affects the turnover. We can understand this intuitively since the schedule will require the nurses to work very difficult shift combinations which will affect the safety and quality of patient care. The schedule should be developed in such a way that the nurse's dissatisfaction is minimized.

Objective: To generate efficient and reliable nurses' schedule.

Criteria: We need to develop the right combination of nurses for each shift in such a way that we can remove various conflicts which normally create problems in the nurse's schedule

Proposed Solution: A graph is constructed and the nodes of the graph represents the different types of nurses. The vertices are then colored using Greedy algorithm approach. Now we will obtain a graph with different colors. Using Problem 0 approach we need to find the different good walks from the graph which are possible. The different good walks will help to provide different optimal nurse schedules so that there will be no bias and all the nurses will work in different shifts without any one group of nurses working in the same shift always.

Situation: Consider the following types of nurses - Senior Ward Assistant (SWA), Principal Enrolled Nurse (PEN), Rotation Nurse (RN), Health Extension Workers (HEW), Staff Nurse (SN) and Enrolled Nurse (EN). They are named as SN1, SN2, SN3, SN4, PEN, RN1, RN2, EN1, EN2, SWA, HEW1, HEW2, HEW3, HEW4 and HEW5. But the constraint is that Nurses with different skills levels can be in same shift but only the junior or senior nurses cannot be in same shift. There are some nurses who do not like to work together. So they must be put in to different groups. Sometimes hospital management also wants to put some nurses in to different shifts to create high quality roster.

Algorithm:

Step 1: Considering different skill levels, the nurses were put into groups.

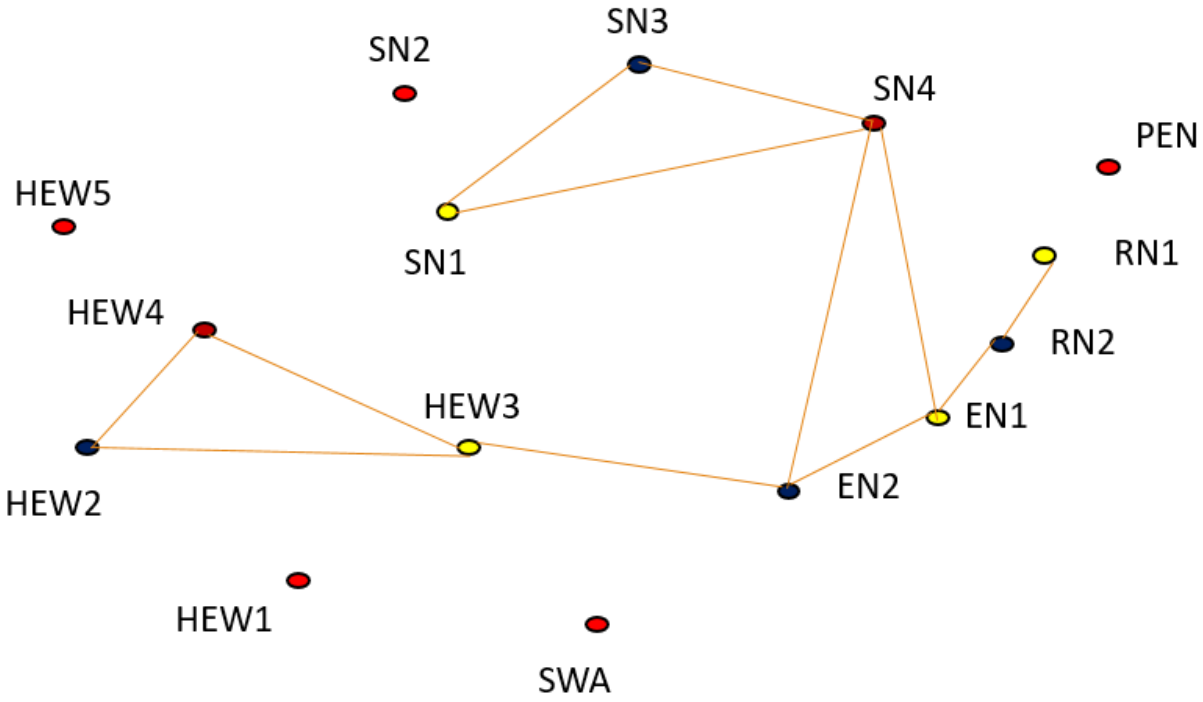
A	SN1,SN3,SN4
B	SN4,EN1,EN2
C	RN1,RN2
D	EN2,HEW3
E	HEW2,HEW3,HEW4
F	RN1,EN1
G	EN1,EN2

Step 2: Create adjacency matrix based on above table.

Adjacency Matrix

	SN1	SN2	SN3	SN4	PEN	RN1	RN2	EN1	EN2	SWA	HEW1	HEW2	HEW3	HEW4	HEW5
SN1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
SN2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SN3	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
SN4	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0
PEN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RN1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
RN2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
EN1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
EN2	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0
SWA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HEW1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HEW2	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
HEW3	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0
HEW4	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
HEW5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Step 3: Construct colored graph from the above adjacency matrix using greedy algorithm



Step 4: Nurse Groups after applying the graph coloring process. These will be the nodes. G1, G2, G3 and G4 will be green, yellow, red and pink nodes.

G1	SN1,EN1,RN1,HEW3
G2	SN3,EN2,RN2,HEW2
G3	SN4,HEW4
G4	SN2,PEN,SWA,HEW1,HEW5

Step 5: These nodes will be repeated in a random manner repeatedly to generate a random graph every month for different time period slots for 24X7. This is done so that there will be no partiality among the different groups of nurses. We need to schedule them one after the

Step 6: Problem 0 part (a) will be used to generate a graph G with good walks such that G1, G2, G3, G4 all will come one after the other in a sequential manner to ensure maximum efficiency.

Step 7: From the graph G we can find different walks using DFS. Each walk will be the nurse scheduling for 24X7.