CS180 Final

Thomas Joseph Garcia

TOTAL POINTS

94 / 100

QUESTION 1

1 Problem 1 (25 / 25)

- **- 0 Correct**
- - 8 Did not show the technique of reduction (adding nodes)
- - 4 Did not show NP-ness of G
- - 25 Blank

QUESTION 2

2 Problem 2 (25 / 25)

- **- 0 Correct**
- - 8 (b) not correct/not complete
- - 10 (a) blank
- - 15 (b) blank/ near blank/ showed nothing relevant

QUESTION 3

3 Problem 3 (22 / 25)

- - 0 Correct
- - 10 for x1..xkXi, what if there does exist Xm->XjXi,Xj->xk but we convert xk-1xk into some Xj first, then there exist Xm->XjXi
- - 12 they are not only ABC
- - 20 irrelevant
- **- 3 almost correct but when combine to new entry**
- - 18 no recursion function
- - 25 blank

QUESTION 4

4 Problem 4 (22 / 25)

- - 0 Correct
- - 5 (a). No answer of showing an order exists
- - 4 (a). Should be the topological sorting of the shortest path tree of the genenral graph
- - 3 (a). It is a general graph, not necessary a DAG. There could be no topological order in a general graph
- - 1 (a). Doesn't specify what is the order according the shortest path.
- - 10 (b). No answer of showing an order
- - 8 (b). Should be topological sorting of the DAG
- - 1 (b). No explanation why topological sorting works
- - 3 (b). How do you know there must be a edge between v_i and v_i+1
- - 10 (c). No answer
- - 10 (c). need to prove that such Ea-Eb alternation appears at most n/2 times in any shortest path. Each phase, at least one node in Ea/Eb is accurate
- - 4 (c). Not a correct proof why Ea-Eb alternation appears at most n/2 times
- - 3 (c). Didn't show that any shortest path is at most n-1 edges and so there are n/2 alternations
- **- 3 (c) Didn't show in each phase, at least one node in Ea/Eb is accurate**
- - 3 (a). you try to give an algorithm. However, you didn't show the algorithm will terminate eventually with the desired order. The coorect answer should be the topological sorting of the shortest path tree of the genenral graph
- - 4 (a). You didn't prove that after putting k edges, f(v)s for other nodes will not change. In fact, f(v) may change and your poof is not correct.

CS 180: Introduction to Algorithms and Complexity
Final Exam (June 7, 2016)

Name	Thomas Garcia
UID	904606029
Section	D

1	2	3	4	Total

★ **Please write your answers ONLY on the front of the exam pages. Anything on the back of page will NOT be graded.**

★ **Print your name, UID and section number in the boxes above, and print your name at the top of every page.**

- The exam is closed book. You can bring one sheet letter size cheat sheet.
- There are 4 problems. Each problem is worth 25 points.
- Do not write code using C or some programming language. Use English or clear and simple pseudo-code. Explain the idea of your algorithm and show why it works.
- Your answer are supposed to be in a simple and understandable manner. Sloppy answers are expected to receiver fewer points.
- Brute force (exponential time) algorithm does not get any point.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.

1. Show the following problem is NP-complete: given an undirected graph G, does G has a spanning tree T such that each vertex in T has a degree of either 1 or 5 ? (Hint: reduction from an undirected (s, t) Hamiltonian path) **[25 pts]**

For each node v in the graph G, add 3 new nodes that connect to v, and also add two new nodes which connects all of the original nodes. . To make a spanning tree, each original node must connect to all three of its new nodes, which means they cannot have a degree of one; so to solve this problem, each of the original nodes will have a degree of 5 while each of the new nodes will have a degree of 1. If it connects to 3 nodes, that means it must connect to two other nodes. If those two other nodes are original nodes, it has a degree of 2 in the original graph, while if 1 is one of the universal new nodes (connected to all), it has a degree of 1 in the in the spanning tree original of graph. Only two nodes can connect to the universal nodes, b/c there are only two nodes of degree 1 in a Hamiltonian path. A node cannot connect to both universal nodes, because then it would become disconnected from the other original nodes, meaning it would not be a spanning tree. One other problem would be if the universal nodes ended up connected to 5 original nodes, but that would lead to either a cycle or a disconnected graph, which is not a spanning tree. Therefore, every spanning tree of degrees 1&5 in this new graph corresponds to a spanning tree of degrees 1&2 in the original, which is a hamiltonian path.

spanning tree of

2. Suppose we are given an array $A[1..m][1..n]$ of non-negative real numbers, such that the sum of each row or column is integer. We want to round A to an integer matrix, by replacing each entry x in A with either $\lfloor x \rfloor$ or $\lceil x \rceil$, without changing the sum of entries in any row or any column of A. For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

(a) Suppose you have a black-box which accepts a matrix as above and does the rounding for you but only under the restriction that the numbers are real numbers between 0 and 1. For example:

$$\begin{bmatrix} 0.2 & 0.4 & 0.4 \\ 0.9 & 0.0 & 0.1 \\ 0.9 & 0.6 & 0.5 \end{bmatrix}$$

Show how to use such a black-box to do the rounding for a general matrix in an extra linear time in the number of entries of the matrix. **[10 pts]**

(b) Give an algorithm that rounds a matrix of numbers between 0 and 1. (Hint: look at the topics of the exam. This problem has to do with the "missing" topic.) **[15 pts]**

a) Create two new mxn matrices, and for each entry in the original array, put the integer portion in the same position in the first new array, and the decimal portion in the same position in the second array. Pass the decimal array to the black box, and the perform elementwise addition (matrix addition) between the result and the integer array. Creating the two arrays is O(n), as is performing matrix addition, so the extra time is O(n).

b) Create two nodes s+t. Then create a node for each entry in the array, and connect it to s with an edge of infinite capacity; flow represents what value is stored in each entry. Then create a node for each row and column, and connect each entry node to the row & column it is in, with an edge of infinite capacity. Finally, connect each row and column node to t, with an edge of capacity equal to the sum of that row or column. Then run integer max flow on the graph. Because

(extra space at next page)

(extra space for problem 2)

all edges but the ones leading to t are infinite capacity, the max flow will be every row and column keeping its element sum, which is what we want.

3. A **grammar** G is a way of generating strings of "terminal" characters from a nonterminal symbol S, by applying simple substitution rules, called **productions**. If $B \rightarrow \beta$ is a production, then we can convert a string of the form $\alpha B \gamma$ into the string $\alpha\beta\gamma$. A grammar is in **Chomsky normal form** if every production is of the form "$A \rightarrow BC$" or "$A \rightarrow$ a", where A, B, and C are nonterminal characters (i.e., uppercase letter) and a is a terminal character (i.e., lowercase letter). For example:

> The grammar G is given by:
>
> - $S \rightarrow AB$; $S \rightarrow BC$
> - $A \rightarrow BA$; $A \rightarrow$ a
> - $B \rightarrow CC$; $B \rightarrow$ b
> - $C \rightarrow AB$; $C \rightarrow$ a
>
> The string "baaba" can be generated from S using this grammar by $S \rightarrow AB \rightarrow BACC \rightarrow$ ba$CC \rightarrow$ ba$AB$a $\rightarrow$ baaba

Design an $O(n^3)$ time dynamic programming algorithm for determining if string $x = x_1x_2...x_n$ can be generated from start symbol S. **[25 pts]**

Opt(i, i) ← the set of all nonterminal symbols that can produce X_i

Opt(i, j) ← for each $i \leq k < j$, the set of non-terminal symbols that can produce one nonterminal in opt(i, k) followed by one symbol in opt(k+1, j).

Opt(j, i) where $j > i$ are empty.

we fill entries in the matrix based on increasing value of $j - i$.

If opt(1, n) contains S, we have a solution.

This is only $O(n^3)$ in the case that each opt(i, j) only has 1 non terminal; otherwise it is $O(n^3 S)$, where S is the number of grammar rules.

(extra space at next page)

(extra space for problem 3)

4. In class we have seen the Bellman-Ford algorithm for a single source shortest path in a directed weighted graph $G = (V, E)$ with possible negative edges but not a negative cycle. The algorithm proceeds in $n-1$ phases where in each phase we *relax* all the edges in some order. Let $f(v)$ denote the distance of the shortest path between the source and node v. The algorithm starts with estimating $f(v)$ that assigns the value 0 to the source node s and value ∞ to all others. An *relaxation* of a directed edge $e = (u, v)$ is setting $f(v) \leftarrow \min\{f(v), f(u) + l(e)\}$, where $l(e)$ is the length of the edge e.

```
BELLMAN-FORD(s)
  f(s) ← 0
  for all vertices v ≠ s
      f(v) ← ∞
  repeat n − 1 times:
      for every edge u → v
          RELAX(u → v)
```

We want to see that the order of choosing edges to relax in a phase matters in how many phases we have to go until the next phase does not change any estimate.

(a) Show that there exists some order (that may be given to you by a Genie) in which in one phase all estimates will be the accurate distance of the shortest path (i.e., the second phase will not change any estimate $f(v)$). **[5 pts]**

(b) Show that if the graph is acyclic (DAG) you don't need the Genie and you can find the order in linear time. **[10 pts]**

(c) Suppose that we consider the following order of relaxing edges: we give source node s the identity 1 and all the other nodes with distinct positive integers from 2 to n. We partition all the edges E into $E_A \cup E_B$, where E_A contains edges from lower id to higher id and E_B contains edges from higher id to lower id. In each phase,we first relax the edges in E_A in *some particular order* from subproblem (b) and then relax edges in E_B in the same order. Consider the shortest path tree, obviously it starts with an edge from E_A (remember s has the lowest id). In the first part of each phase, any path in the shortest path tree that starts at an accurately estimated vertex and goes only through edges in E_A becomes accurately estimated, and in the second part of each phase, any path that starts at an accurately estimated vertex and goes only through edges in E_B becomes accurately estimated. The number of phases needed is the maximum number of such $E_A - E_B$ edge alternations on any path. Prove that this algorithm will calculate the final shortest path values in at most $n/2$ phases. **[10 pts]**

a) Because there are no negative cycles, the shortest path to any node is a simple path. Now, assume we have two vertices u, v such that u is the vertex immediately preceding v on the shortest path from s to v. If we already know the distance of the shortest path from s→u, we only need to relax one edge (u→v) to get the shortest distance from s to v. Therefore, if we relax all the edges in the path from s to v in order, we can get the shortest distance to v with relaxing each edge once. We can always do these in order, because the shortest path has no cycles. To handle multiple nodes, we simply need to take the ordering for each

(extra space at next page)

(extra space for problem 4)

node, & combine them so that each shortest path is still relaxed in order. If two shortest paths traverse the same series of nodes in reverse order this combination is impossible, but that will never happen, because for any series of nodes in a shortest path, they all have different distances from S, so all shortest paths that use that segment go in the same order. Therefore, if we relax edges such that for each shortest path, edges are relaxed in order from S to the end (which is always possible), we only need to relax each edge once.

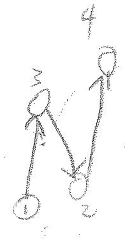
b) If a graph is a DAG, you can perform a topological sort, and then relax edges based on the topological ordering of the outgoing node (where the edge comes from). All parents of a node have lower topological orderings, so before we relax the outgoing edges of a node, we have already relaxed all preceding edges, meaning each shortest path is relaxed in order.

c) ~~There are no negative cycles in the graph, so all shortest paths are simple paths. Simple paths are trees, so each path has m-1 edges, where m is the number of nodes in the path. Simple paths do not repeat nodes, so the longest possible shortest path has n nodes and n-1 edges, where n is the number of nodes in the graph.~~

(extra space at next page)

(extra space for problem 4)

c) ~~For each phase, we relax all the edges coming out of the vertex in E_A with the smallest topological order, followed by all of the edges coming out of the vertex in E_B with smallest topological order (E_A & E_B have different orderings). Because they are done in topological order, by the~~



c) The longest simple path in the graph has n−1 edges in it. At most half of those edges cross between E_A & E_B, so at most $\frac{n}{2}$ passes are needed.