CS325 MT1

1) Show that the Hamiltonian Cycle problem for directed graphs is in NP-Complete.
Solution
a) Show you can verify a solution in Polynomial time
b) Show that Ham-Cycle (undirected) which is in NP-Complete can be reduced to Ham-Cycle-Directed
Transform an undirected graph G into a directed graph H by replacing each edge vw of G by edges in each direction vw and wv. Then if there's a Hamiltonian cycle in G, there's one in H. For each edge in the cycle in G, the cycle in H uses the dge from a vw and wv pair that corresponds to the direction the edge was traversed in G. If there is a Hamiltonian cycle in H, then there is on ein G, simply replace either edge vw and wv in the cycle in H by the corresponding undirected edge in G.

3. Suppose you have four production plants for making cars. Each works a little differently in terms of labor needed, materials, and energy used per car:

Location	labor hrs	materials	energy
plant1	2	3	15
plant2	3	4	10
plant3	4	5	9
plant4	5	6	7

Suppose we need to produce at least 400 cars at plant 3 according to a labor agreement. We have 3300 hours of labor and 4000 units of material available. We are allowed to use at most 12000 units of energy, and we want to maximize the number of cars produced. Formulate this as a linear programming problem: (1) what are the variables, (2) what is the objective in terms of these variables, and (3) what are the constraints.

Solution:

1. What are the variables? x1,x2,x3,x4, where xi denotes the number of cars at plant i.

2. What is our objective? maximize x1 + x2 + x3 + x4.

3. What are the constraints?

xi ≥ 0 (for all i)

x3 ≥ 400

2x1 + 3x2 + 4x3 + 5x4 ≤ 3300

3x1 + 4x2 + 5x3 + 6x4 ≤ 4000

15x1 + 10x2 + 9x3 + 7x4 ≤ 12000

Q2. Macrosoft has a 24-hour-a-day, 7-days-a-week toll free hotline that is being set up to answer questions regarding a new product. The following table summarizes the number of full-time equivalent employees (FTEs) that must be on duty in each time block. Macrosoft may hire both full-time and part-time employees.
□ _Full-timers work 8-hour shifts with hourly wages of $15.20
□ _Part-timers work 4-hour shifts with hourly wages of $12.95.
□ _Employees may start work only at the beginning of 1 of the 6 intervals.
□ _Part-time employees can only answer 5 calls in the time a full-time employee can answer 6 calls. (i.e., a part-time employee is only 5/6 of a full-time employee.)
□ _At least two-thirds of the employees working at any one time must be full-time employees.
CHART (cover each shift and one for each y)
Formulate an LP to determine how to staff the hotline at minimum cost.

Decision Variables

x_t = # of full-time employees that begin the day at the start of interval t and work for 8 hours
y_t = # of part-time employees that are assigned interval t

$$\min \quad \overset{(8\times 15.20)}{121.6}(x_1+\cdots+x_6) + \overset{(4\times 12.95)}{51.8}(y_1+\cdots+y_6)$$

$$\begin{aligned}
\text{s.t.}\quad & x_1 + x_6 + \tfrac{5}{6}y_1 \ge 15 \\
& x_1 + x_2 + \tfrac{5}{6}y_2 \ge 10 \\
& x_2 + x_3 + \tfrac{5}{6}y_3 \ge 40 \\
& x_3 + x_4 + \tfrac{5}{6}y_4 \ge 70 \\
& x_4 + x_5 + \tfrac{5}{6}y_5 \ge 40 \\
& x_5 + x_6 + \tfrac{5}{6}y_6 \ge 35
\end{aligned}$$

All shifts must be covered

PT employee is 5/6 FT employee

$$\begin{aligned}
x_1 + x_6 &\ge \tfrac{2}{3}(x_6 + x_1 + y_1) \\
x_1 + x_2 &\ge \tfrac{2}{3}(x_1 + x_2 + y_2) \\
&\vdots \\
x_5 + x_6 &\ge \tfrac{2}{3}(x_5 + x_6 + y_6)
\end{aligned}$$

At least 2/3 workers must be full time

$x_t \ge 0,\ y_t \ge 0 \quad t = 1,2,\ldots,6$ Nonnegativity

4. Prove that 4-SAT is NP-complete. **Instance:** A collection of clauses C where each clause contains exactly 4 literals. **Question:** Is there a truth assignment to x such that each clasue is satisfied?

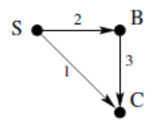
Example $\Phi = (x_1 \vee \neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_3 \vee x_2 \vee x_4)$

Ans: To show that 4-SAT is NP-complete, we prove that 4-SAT is in NP and NP-hard.
First, 4-SAT is in NP, we can write a nondeterministic polynomial-time algorithm which takes a 4-SAT instance and a proposed truth assignment as input. This algorithm evaluates the 4-SAT instance with the truth assignment. If the 4-SAT instance evaluates to true, the algorithm outputs *yes*; otherwise, the algorithm outputs *no*. This runs in polynomial time.
To prove that 4-SAT is NP-hard, we reduce 3-SAT to 4-SAT as follows. Let ϕ denote an instance of 3-SAT. We convert ϕ to a 4-SAT instance ϕ' by turning each clause $(x \vee y \vee z)$ in ϕ to $(x \vee y \vee z \vee h) \wedge (x \vee y \vee z \vee \neg h)$, where h is a new variable. Clearly this is polynomial-time doable.

⇒ If a given clause $(x \vee y \vee z)$ is satisfied by a truth assignment, then $(x \vee y \vee z \vee h) \wedge (x \vee y \vee z \vee \neg h)$ is satisfied by the same truth assignment with h arbitrarily set. Thus if ϕ is satisfiable, ϕ' is satisfiable.

⇐ Suppose ϕ' is satisfied by a truth assignment T. Then $(x \vee y \vee z \vee h) \wedge (x \vee y \vee z \vee \neg h)$ must be true under T. As h and $\neg h$ assume different truth values, $x \vee y \vee z$ must be true under T as well. Thus ϕ is satisfiable.

5. Let G = (V, E) be a DAG, where each edge is annotated with some positive length. Let s be a source vertex in G. Suppose we run Dijkstra's algorithm to compute the distance from s to each vertex v □ _V, and then order the vertices in increasing order of their distance from s. Are we guaranteed that this is a valid topological sort of G? Answer: No, explanation below
Answer:



Comment: Sorting by increasing distance gives S, C, B; but B must precede C in any valid topological sort.

6. Consider a connected weighted directed graph G = (V, E, w). Define the *fatness* of a path P to be the maximum weight of any edge in P. Give an efficient algorithm that, given such a graph and two vertices u,v ∈V, finds the minimum possible fatness of a path from u to v in G.

We can see that that fatness must be the weight of one of the edges, so we sort all edge weights and perform a binary search. To test whether or not a path with a fatness no more than x exists, we perform a breadth-first search that only walks edges with weight less than or equal to x. If we reach v, such a path exists.

If such a path exists, we recurse on the lower part of the range we are searching, to see if a tighter fatness bound also applies. If such a path does not exist, we recurse on the upper part of the range we are searching, to relax that bound. When we find two neighboring values, one of which works and one of which doesn't, we have our answer. This takes $O((V+E)\lg E)$ time.

8. A Hamiltonian path in a graph G=(V,E) is a simple that includes every vertex in V. Design an algorithm to determine if a directed acyclic graph (DAG) G has a Hamiltonian path. Your algorithm should run in O(V+E). Provide a written description of your algorithm including why it works, pseudocode and an explanation of the running time.
Solution: Compute a topological sort and check if there is an edge between each consecutive pair of vertices in the topological order. If each consecutive pair of vertices are connected, then every vertex in the DAG is connected, which indicates a Hamiltonian path exists. Running time for the topological sort is O(V+E), running time for the next step is O(V) so total running time is O(V+E).

Q7. Let T be a complete binary tree with n vertices. Finding a path from the root of T to a given vertex v in T using breadth-first search takes **O(n)** .

Q8) **PSEUDOCODE:**
```
HAS_HAM_PATH(G)
1. Call DFS(G) to computer finishing time v.f for each vertex v.
2. As each vertex is finished, insert it into the front of the list
3. Iterate each through the lists of vertices in the list
4. If any pairs of consecutive vertices are not connected RETURN FALSE
5. After all pairs of vertices are examined RETURN TRUE
```

9. **Variable Definitions**
xt : Number of drivers scheduled at time *t*, *t* = 0, 4, 8, 12, 16, 20
We assume that this problem continues over an indefinite number of days with the same bus requirements and that *xt* is the number used in every day at time *t.*

10. Consider a variant of BIN-PACKING Problem where the bin size is b (not necessarily an integer) and each item has size less than b/3. This version is still NP-complete, though we won't prove this here. Your problem is to give an algorithm that *approximates* the optimal packing into bins, and prove a bound on the quality

The objective is to
minimize x0 + x4 + x8 + x12 + x16 + x20
Constraints
We need constraints that assure that the drivers scheduled at the times that cover the requirements of a particular interval sum to the number required. For the interval from time 0 to 4, drivers starting at time 20 of the previous day and time 0 of the current day cover the needs from time 0 to time 4. Then we write
x20 + x0 >= 4
The remaining constraints are:
x0 + x4 >= 8
x4 + x8 >=10
x8 + x12 >= 7
x12 + x16 >= 12
x16 + x20 >= 4
xi >= 0, xi an integer

of your approximation. In particular, prove that if your algorithm uses 3a+1 bins for some integer a, then the optimal algorithm uses at least 2a+1 bins.
Solution: You can use the Next-Fit algorithm where we keep one bin open at a time, look at each item in turn, and put it into the current bin if and only if it fits. If it doesn't fit, we open a new bin. Every bin except the last one we use must be filled to more than 2b/3, since we could only close it if a new item, of size less than b/3, failed to fit.
Thus if our algorithm uses 3a+1 bins, the first 3a bins contain a total size of more than 3a(2b/3) = 2ab. If we include the last partially filed las bin, then the 3a+1 bins have a size > 2ab. The optimal algorithm could not possibly fit this much size into 2a bins of size b, so it must use at least 2a+1 bins as desired.

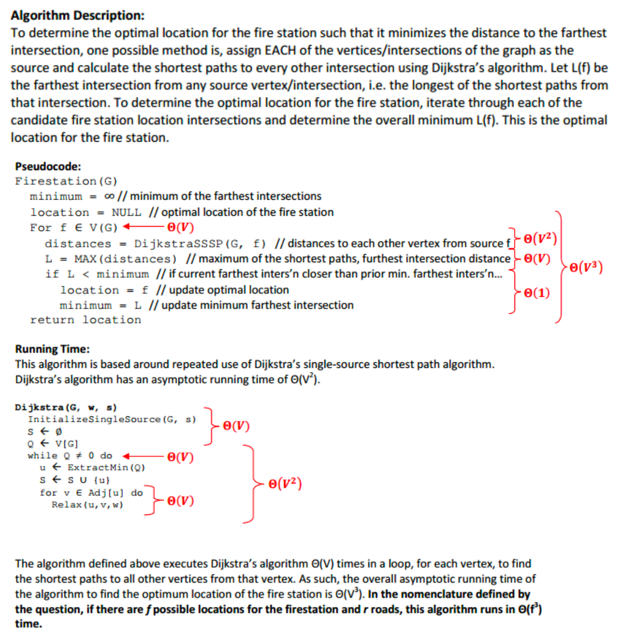
HW7 - Let X and Y be two decision problems. Suppose we know that X reduces to Y in polynomial time. Which of the following can we infer? Explain
a. If Y is NP-complete then so is X. False cannot be inferred
b. If X is NP-complete then so is Y. False cannot be inferred
c. If Y is NP-complete and X is in NP then X is NP-complete. False cannot be inferred
d. If X is NP-complete and Y is in NP then Y is NP-complete. TRUE
e. If X is in P, then Y is in P. False cannot be inferred
f. If Y is in P, then X is in P. TRUE

```
def minCoins(coins, amount):
  table = [None for x in range(amount + 1)]
  table[0] = []
  for i in range(1, amount + 1):
    for coin in coins:
      if coin > i: continue
      elif not table[i] or len(table[i - coin]) + 1 < len(table[i]):
        if table[i - coin] != None:
          table[i] = table[i - coin][:]
          table[i].append(coin)

  if table[-1] != None:
    print '%d coins: %s' % (len(table[-1]), table[-1])
  else:
    print 'No solution possible'
```

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.
The above algorithm is simply DFS with an extra stack. So time complexity is same as DFS which is **O(V+E).**
Problem: undirected graph and each edge weight is changed by 1.
- Minimum Spanning tree does NOT change
- Shortest paths MAY change
2. In the bottleneck-path problem, you are given a graph G with edge weights, two vertices s and t and a particular weight W; your goal is to find a path from s to t in which every edge has at least weight W. (a) Describe an efficient algorithm to solve this problem. **Perform a BFS of G starting at s and ignoring all edges with weight < W until edge t is reached. If t is not reached before the BFS terminates then there is no path from s to t with weight at least W.**
(b) What is the running time of youor algorithm. **O(E+V)**

Optimal Fire Station Route



Algorithm Description:
To determine the optimal location for the fire station such that it minimizes the distance to the farthest intersection, one possible method is, assign EACH of the vertices/intersections of the graph as the source and calculate the shortest paths to every other intersection using Dijkstra's algorithm. Let L(f) be the farthest intersection from any source vertex/intersection, i.e. the longest of the shortest paths from that intersection. To determine the optimal location for the fire station, iterate through each of the candidate fire station location intersections and determine the overall minimum L(f). This is the optimal location for the fire station.

Pseudocode:

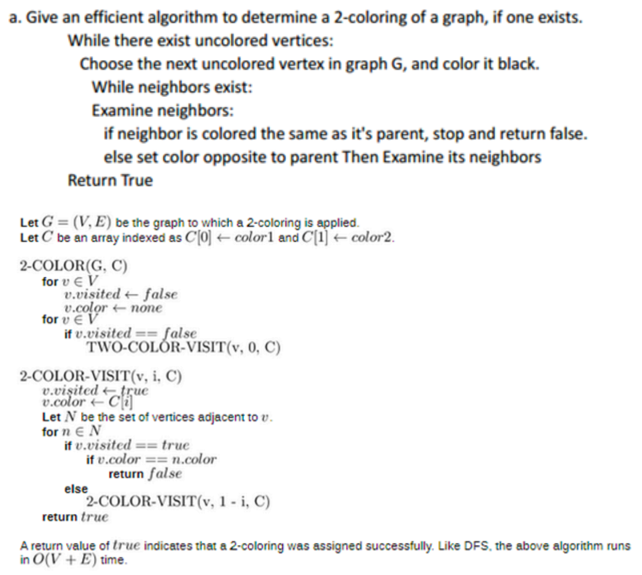
```
Firestation(G)
  minimum = ∞ // minimum of the farthest intersections
  location = NULL // optimal location of the fire station
  For f ∈ V(G)
    distances = DijkstraSSSP(G, f) // distances to each other vertex from source f
    L = MAX(distances) // maximum of the shortest paths, furthest intersection distance
    if L < minimum // if current farthest inters'n closer than prior min. farthest inters'n...
      location = f // update optimal location
      minimum = L // update minimum farthest intersection
  return location
```

Running Time:
This algorithm is based around repeated use of Dijkstra's single-source shortest path algorithm.
Dijkstra's algorithm has an asymptotic running time of Θ(V^2).

```
Dijkstra(G, w, s)
  InitializeSingleSource(G, s)
  S ← ∅
  Q ← V[G]
  while Q ≠ 0 do
    u ← ExtractMin(Q)
    S ← S ∪ {u}
    for v ∈ Adj[u] do
      Relax(u,v,w)
```

The algorithm defined above executes Dijkstra's algorithm Θ(V) times in a loop, for each vertex, to find the shortest paths to all other vertices from that vertex. As such, the overall asymptotic running time of the algorithm to find the optimum location of the fire station is Θ(V^3). **In the nomenclature defined by the question, if there are *f* possible locations for the firestation and *r* roads, this algorithm runs in Θ(f^3) time.**

Graph-Color Efficient Algo. For 2 Color, use BFS/DFE and O(V+E) time.
Give an efficient algorithm to determine a 2-coloring of a graph, if one exists.



a. Give an efficient algorithm to determine a 2-coloring of a graph, if one exists.

```
While there exist uncolored vertices:
  Choose the next uncolored vertex in graph G, and color it black.
    While neighbors exist:
    Examine neighbors:
      if neighbor is colored the same as it's parent, stop and return false.
      else set color opposite to parent Then Examine its neighbors
Return True
```

Let $G = (V, E)$ be the graph to which a 2-coloring is applied.
Let C be an array indexed as $C[0] \leftarrow color1$ and $C[1] \leftarrow color2$.

```
2-COLOR(G, C)
  for $v \in V$
    $v.visited \leftarrow false$
    $v.color \leftarrow none$
  for $v \in V$
    if $v.visited == false$
      TWO-COLOR-VISIT(v, 0, C)

2-COLOR-VISIT(v, i, C)
  $v.visited \leftarrow true$
  $v.color \leftarrow C[i]$
  Let $N$ be the set of vertices adjacent to $v$.
  for $n \in N$
    if $v.visited == true$
      if $v.color == n.color$
        return $false$
    else
      2-COLOR-VISIT(v, 1 - i, C)
  return $true$
```

A return value of $true$ indicates that a 2-coloring was assigned successfully. Like DFS, the above algorithm runs in $O(V + E)$ time.

BFS and DFS are V+E and Dijkstra is V^2. Topological Sort, which is also O(V+E). | topological sort uses DFS | GIN = non-negative Integers |
3-SAT < HAM-PATH < HAM-CYCLE.|A Hamiltonian path in a graph is a simple path that visits every vertex exactly once. Provethat HAM-PATH = { (G, u, v): there is a Hamiltonian path from u to v in G} is NP-complete. You may use the fact that HAM-CYCLE is NP-complete
In many practical situations, the least costly way to go from a place u to a place w is to go directly, with no intermediate steps. Put another way, cutting out an intermediate stop never increases the cost. We formalize this notion by saying that the cost function c satisfies the triangle inequality if, for all vertices u; v; w in V, c(u,w) <= c(u,v) + c(c,w)" | for TSP approximation, you basically find an MST, then preorder visit the tree, giving you an at worst 2x longer trip (2-approximation)
Travelling Salesman Problem - Approximate w/ MST | **Triangle-Inequality:** The least distant path to reach a vertex j from i is always to reach j directly from i, rather than through some other vertex k (or vertices), i.e., dis(i, j) is always less than or equal to dis(i, k) + dist(k, j). The Triangle-Inequality holds in many practical situations. | When the cost function satisfies the triangle inequality, we can design an approximate algorithm for TSP that returns a tour whose cost is never more than twice the cost of an optimal tour. The idea is to use **M**inimum **S**panning **T**ree (MST). Following is the MST based algorithm.

Algorithm: 1) Let 1 be the starting and ending point for salesman.
2) Construct MST from with 1 as root using Prim's Algorithm.
3) List vertices visited in preorder walk of the constructed MST and add 1 at the end.

Approximate since The cost of the output produced by the above algorithm is never more than twice the cost of best possible output