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. | 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. |
| 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? | 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 |
| 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.*  **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 | 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 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 | 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. |

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](https://www.geeksforgeeks.org/greedy-algorithms-set-5-prims-minimum-spanning-tree-mst-2/). **3)** List vertices visited in preorder walk of the constructed MST and add 1 at the end. | **Approximate** sinceThe cost of the output produced by the above algorithm is never more than twice the cost of best possible output |