**Algorihms Questions**

How to proof the correctness of a greedy algorithm? https://cs.stackexchange.com/questions/59964/how-to-prove-greedy-algorithm-is-correct

concept quiz on NP problem https://www.youtube.com/watch?v=bczw7umoeRU

[True/False] If an optimization problem can be solved using a greedy algorithm, there must be a solution for this optimization problem using dynamic programming as well? True maybe ? Both depend on optimal structure of sub problems, greedy takes into account only one, but dp takes them all into account. It also depends on what u mean by "optimization" greedy and dp may both solve a problem, but they may not have the same solution, eg try solving knapsack with greedy, u won't get the optimal result that dp gets. So u can say dp has a better chance of getting the optimal solutiom but at the expense of higher time complexities. where did u find this question ? [here: https://cs.stackexchange.com/questions/23493/dynamic-programming-vs-greedy-algroithms]

[True/False] If an optimization problem can be solved using dynamic programming, there must be a solution for this problem using a greedy algorithm as well? "False

Dynamic programming requires the problem to have ""optimal substructure"" and ""overlapping subproblems"" properties

but greedy problem requires that the problem must exhibit ""greedy-choice"" property in addition to ""optimal substructure""

""greedy-choice"" means local optimal solution is global also

having a dynamic programming solution implies that the problem have ""optimal substructure"" and ""overlapping subproblems"" but does not imply that it has the ""greedy-choice"" property thus we can not assume that there is a greedy algorithm solution for it."

Clearly describe an algorithm, strictly better than O(n^2), that takes a positive integer s and a set A of n positive integers and returns a Boolean answer to the question whether there exist two distinct elements of A whose sum is exactly s. Evaluate its complexity Using 2 pointer algorithm technique, first sort the array and place a pointer at the first element and another at the last element. Repeatedly check if the sum of the 2 elements where the 2 pointers point is equal to the needed sum. If it is less, advance the left pointer (if sorted ascendingly), if it is greater than the needed sum, decrease the right pointer. Repeat till sum is found, or if the 2 pointers cross each other. If array is sorted: O(n), if not O(nlogn)

"Very difficult question

The halting problem is a famous problem in computer science and it is defined as follows: Given a Program P and an Input A, decide whether Program P will halt given input A. i.e. H(P,A)=1 iff Program P will halt given input A and H(P,A)=0 iff Program P will not halt ,runs for an infinite loop for example, given input A.

Prove that the halting problem is NP-Hard problem"

Why doesn't Dijkstra work for negative edges ? https://stackoverflow.com/questions/13159337/why-doesnt-dijkstras-algorithm-work-for-negative-weight-edges

[True/False] Bellman Ford can detect negative cycles but cannot calculate the short distance between any 2 vertices in this case TRUE

Prove that Super Mario Bros. is NP-Hard "(Page-1) http://courses.csail.mit.edu/6.890/fall14/lectures/L05\_images.pdf

(Page-3) https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2015/lecture-notes/MIT6\_046JS15\_lec16.pdf

(Page-4) https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2015/lecture-notes/MIT6\_046JS15\_writtenlec16.pdf

"

"This idea is repeated more than one time in different MIT exams.

Given weighted directed graph G with vertices V and Edges E, the weight of Edge E(u,v) represents the probability of sucess when moving from u to v. Describe an efficient algorithm to find a path from the start vertex s

to the destination vertex t that maximizes the probability of a successful traversal." Intuitively, we want to maximize P1 x P2 x P3 x P4.... = Π(Pi) which is equivalent of minimizing the value of (−Log(P1)) + (−Log(P2)) + (−Log(P3)) + .... = − Σ LogPi so we can change the weights of the each Edge in the Original graph w to −Log(w) and apply any single source shortest path algorithm. Note also that since we are talking about probabilites, each weight w of the orignal graph is less than 1, so the new weights −Log(w) will be always > 0 and we can use Dijkstra for better running time.

If we have a problem to be classified, and another NP-complete one. Which to reduce to the other? Reduce the NP-Complete one to the new problem to be classified. Also need to prove that the time of the decision version of the problem is polynomial

Check Problem 3 in the next link - Graphs. http://web.stanford.edu/class/cs161/Homework/Homework5/HW5\_sols.pdf

[True/False] If P equals NP, then NP equals NP-Complete TRUE

[True/False] If P equals NP, then NP equals NP-Hard FALSE

[True/False] Every problem in NP can be solved in exponential time. True. NP is contained in EXP.

[True/False] If a problem X can be reduced to a known NP-hard problem, then X must be NP-hard. False. The reverse, however, is true: if a known NP-hard problem can be reduced to X then X must be NP-hard.

"[True/False] Consider two positively weighted graphs G = (V, E, w) and G0 = (V, E, w0) with the same vertices V and edges E such that, for any edge e ∈ E, we have w0(e) = w(e)^2

.For any two vertices u, v ∈ V , any shortest path between u and v in G0 is also a shortest path in G." False. Assume we have two paths in G, one with weights 2 and 2 and another one with weight 3. The first one is shorter in G0 while the second one is shorter in G

[True/False] If SAT ≤ A, then A is NP-hard. TRUE

[True/False] Running a DFS on an UNDIRECTED graph G = (V, E) always produces the same number of cross edges, no matter what order the vertex list V is in and no matter what order the adjacency lists for each vertex are in True. DFS in an undirected graph never produces cross edges

[True/False] Every directed acyclic graph has exactly one topological ordering FALSE. Some priority constraints may be unspecified, and multiple orderings may be possible for a given DAG.

[True/False] Given a graph G = (V, E) with positive edge weights, the Bellman-Ford algorithm and Dijkstra’s algorithm can produce different shortest-path trees despite always producing the same shortest-path weights. True. Both algorithms are guaranteed to produce the same shortestpath weight, but if there are multiple shortest paths, Dijkstra’s will choose the shortest path according to the greedy strategy, and Bellman-Ford will choose the shortest path depending on the order of relaxations, and the two shortest path trees may be different

"[True/False] Consider a weighted directed graph G = (V, E, w) and let X be a shortest s-t path for s, t ∈ V . If we double the weight of every edge in the graph, setting w0(e) = 2w(e) for each e ∈ E, then X will still be a shortest s-t path in

(V, E, w0)." True. Any linear transformation of all weights maintains all relative path lengths, and thus shortest paths will continue to be shortest paths, and more generally all paths will have the same relative ordering. One simple way of thinking about this is unit conversions between kilometers and miles

[True/False] If a directed graph G is cyclic but can be made acyclic by removing one edge, then a depth-first search in G will encounter exactly one back edge False. You can have multiple back edges, yet it can be possible to remove one edge that destroys all cycles.

msh fahem mmkn shar7 aktar ^^ which row number ? row 27

how do you find the second shortest path? Using a modified Dijkstra algorithm: https://en.wikipedia.org/wiki/K\_shortest\_path\_routing#Algorithm