CS 344 - Fall 2022
Exam 2
100 points total + 15 points extra credit

Instructions

Academic Integrity Policy You must complete this exam entirely alone. You may not discuss the exam with any students in the class, or with anyone outside the class.

The exam is open notes, so you can look at any files/videos posted on canvas and your own notes. You may also use either of the two books recommended for this course. You are not allowed to use any other sources, *and in particular you are not allowed to use any online source.*

Academic Integrity Statement You should type the following text on the first page of your midterm.

I have not discussed this exam with anyone except the professor Aaron Bernstein. I have not used any online resources during the exam except for those accessible from the Canvass website.

What to Write

- On the first page write your full name, netID, and the academic integrity statement above. Do not write anything else.
- Start each problem on a new page.
- Typed assignments receive 2 extra points.
- If your assignment is hand-written please write very legibly.
- Assignment should be submitted as a single PDF in the same way as a HW problem.

Skipping Problems Remember that if you skip a problem you get 25% credit. If you give an answer that is on the right track you will get more than 25% in partial credit, so don't be afraid to attempt problems if you think

you know how to approach them. But keep in mind that if you simply don't know the answer, writing a very wrong answer may get 0%.

To skip a problem, you have to write "I don't know" *and nothing else.* You can skip an individual problem part such as e.g. problem 1 part 3 and then get 25% credit for that part.

The only exception is the extra credit problem: you do not get any credit for leaving that blank and it's harder to get partial credit on that problem.

The 344 Library: Throughout the test, any algorithm you design may use any of the functions or data structures we covered in class without explanation. In particular, here are some functions from our library on graph algorithms that you may find it useful to take advantage of.

- The following functions take as input a specific source $s \in V$ and output dist(s,v) for all vertices $v \in V$.
  - BFS(G,s) – only works on unweighted graph. Runtime $O(|E|)$.
  - Dijkstra(G,s) – only works if graph has non-negative weights. Runtime $O(|E| \log(|V|))$.
  - DAG-SP(G,s) – computes shortest distances from source s if the input graph G is a DAG. This works on graphs that have positive and negative edge weights. Runtime is $O(|E|)$.
- TopSort(G). After running this algorithm, you can assume that the vertex set $V = \{v_1, ..., v_n\}$ is in topological order. Runtime is $O(|E|)$.

Graph Problems For all problems that involve a graph, you can assume that $|E| \geq |V|/2$.

1 Problem 1 (31 points)

typed exam: For this problem (and this problem only), you can hand-draw whatever figures you want, or even hand-write your solution to the whole problem, and still get the two points for a typed exam. This applies to all parts of this problem.

1.1 Part 1 – 13 points

Draw a graph G with the following properties

- The graph G is a DAG
- The graph G has *exactly* 7 vertices. You should label them a,b,c,d,e,f,g
- The graph G has *exactly* four possible topological orders.

WHAT TO WRITE: your final answer should include

- A drawing of the graph.
- The four possible topological orderings of your graph. For example, if you write b,d,e,f,c,a as an ordering that means that in the ordering b comes before d comes before e and so on.
- Please make your drawing extremely legible. In particular, draw everything large, as that will make it easier for the graders to interpret.

GRADING NOTE: If you draw a graph that has exactly three topological orderings, or exactly five topological orderings, and you correctly write down all the orderings, then you will get 9 out of 13 points.

1.2 Part 2 (13 points)

Consider the graph G in Figure 1 below. Now consider running Dijkstra(G,s). (s is the bottom left vertex.)

Draw a table which indicates what the algorithm looks like after each execution of the while loop inside Dijkstra's. In particular, for each iteration of the loop you should indicate

- which vertex is explored in that iteration

- what is the label $d(v)$ for *every* vertex v at the end of that iteration.
- So all in all you should draw a table with 8 rows and 9 columns. Each row corresponds to an iteration of the while loop: so you can label the rows "iteration 1", "iteration 2", and so on up to "iteration 8". You then have nine columns. The first column you should label "explored vertex", and this column indicates which vertex is explored in that iteration. You then have 8 other columns labeled d(C), d(D), d(H), d(J),d(L),d(M),d(P),d(S), which show the label of each vertex at the end of every iteration.

HELPFUL RESOURCE: see the files on Canvas in the folder "Dijkstra Practice" for an example of the kind of table you should write. See also the solutions for HW 5 problem 1 part 1.

SUGGESTION: I purposefully chose a confusing-looking graph so that you can't just eyeball the shortest distances, and have to use Dijkstra's instead. In particular, if you overlook even a single edge you might get the totally wrong distances. So i really recommend printing out the graph, taking out a pen, and going through the steps of Dijkstra as systematically as possible.

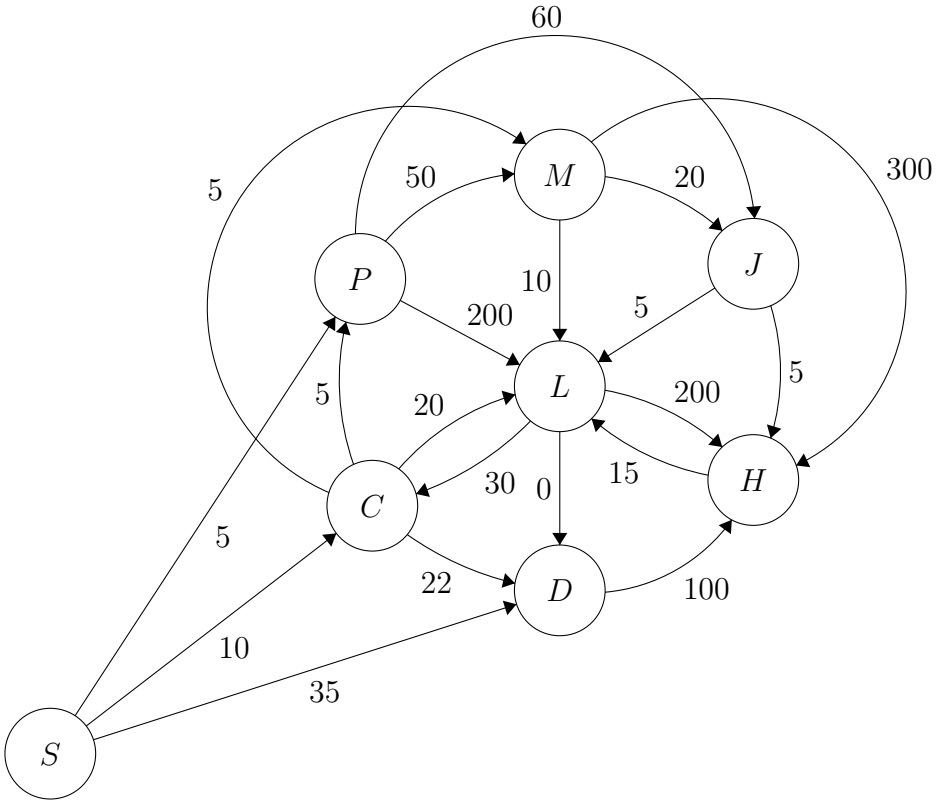
1.3 part 3 – 5 points

Let G be an *directed, unweighted* graph, and let s be some source. Recall that for any two vertices a, b, we define dist(a,b) to be the length of the *shortest* path from a to b – i.e. number of edges on the shortest path.

Now, say that there is some vertex x in the graph where you know that dist(s,x) = 7 and also that G contains an edge (x, y). For each of the five cases below, say POSSIBLE if there could possibly exists a graph G with this value of dist(s,y), and NOT POSSIBLE if there cannot exist such a graph G.

1. dist(s,y) = 5
2. dist(s,y) = 6
3. dist(s,y) = 7
4. dist(s,y) = 8

Figure 1: Figure for problem 1 part 2



5. dist(s,y) = 9

NOTE: Recall that that for all five cases above, we are are assuming that G is directed and unweighted, that dist(s,x) = 7 and that G contains an edge (x, y)

CLARIFICATION TO ABOVE NOTE: G contains edge (x, y) means that there is an edge from x to y. This does not say anything about an edge from y to x; G *could* also contain an edge from y to x, but it does *not necessarily* contain an edge from y to x.

GRADING NOTE: if you get one case wrong you get 3 out of 5 points. If you get 2 or more cases wrong you get 0 points.

2 Problem 2 (27 points)

Say that you are given a directed graph $G = (V, E)$ and a source s. Say that all the edge weights of G are non-negative except that there is EXACTLY ONE edge (x, y) that has negative weight. You can assume that the algorithm is told which edge (x, y) has negative weight.

Part 1 (12 points) Write pseudocode for an algorithm that in $O(|E| \log(|V|))$ time detects whether G contains a negative-weight cycle. The algorithm should return "YES CYCLE" if there is a negative-weight cylce and "NO CYCLE" if there is not a negative weight cycle. (Recall that a negative-weight cycle is simply a cycle C such that the sum of all edge-weights on C is < 0.)

Part 2 (15 points) Consider the graph G above and say that you have the additional guarantee that G does NOT contain a negative-weight cycle. Write pseudocode for an algorithm that in $O(|E| \log(|V|))$ time computes dist(s,v) for all vertices v.

IMPORTANT NOTE: For both of these problems, you *must* use graph transformation. That is, your solution to both parts should create a new graph G' which is slightly different from G and then run Dijkstra's algorithm in G'. You might need to run Dijkstra's algorithm in G' more than once, and note that you can run Dijkstra from whatever source you want; make sure to always indicate which source you are running it from. You can then use the distances you learn in G' to figure out the answer you need about G.

NOTE: in the solution I have in mind, the graph G' is only very slightly different from G.

3 Problem 3 (15 points)

In this problem we consider a variant of subset sum. Consider the following problem:

- INPUT:
  - A set $S = \{s_0, ..., s_{n-1}\}$ with n positive integers.You can think of S as an array, so you can access any $s_i = S[i]$ in $O(1)$ time.
  - Each element $s_i \in S$ is colored either red or blue. You can think of each color as a parameter in the objects s_i. So you can access any s_i.color in $O(1)$ time.
  - A target positive integer B
- OUTPUT: the algorithm should output TRUE if there exists a subset $S' \subseteq S$ such that $\sum_{x \in S'} = B$ and S' contains at most 100 red elements. The algorithm should output FALSE otherwise. Note that you don't need to output the set S' itself; just true/false. Also note that the number of blue elements in S' doesn't matter; the only restriction is that S' contains at most 100 red elements.

The problem: For this problem, I wrote most of the pseudocode for you. All you have to do is fill in the two answer lines in the pseudocode. To make the pseudocode more understandable, I will tell you what table entry T[i][j][k] corresponds to.

The table entries My pseudocode will set T[i][j][k] to be TRUE if there exists a subset S' with the following properties

- $S' \subseteq \{s_1, \ldots s_i\}$. That is, S' only uses the first i elements of S.
- $\sum_{x \in S'} = j$. That is, the elements of S' sum to j.
- The number of red elements in S' is exactly k.

My Pseudocode

1. Initialize table $T[0...n-1][0...B][0...k]$ with all FALSE
2. If s_0.color == blue then set $T[0][s_0][0]$ = TRUE

3. If s_0.color == red then set $T[0][s_0][1]$ = TRUE

4. For $i = 1$ to $n - 1$

   (a) For $j = 0$ to B

      i. For $k = 0$ to 100

         A. If $T[i-1][j][k]$ == TRUE then set $T[i][j][k] \leftarrow$ TRUE

         B. Else if s_i.color == red and $k = 0$ then set $T[i][j][k] \leftarrow$ FALSE

         C. Else if s_i.color == blue

            - If $j = s_i$ and $k = 0$
              - $T[i][j][k] \leftarrow$ TRUE \\ $S' = \{s_i\}$ is a solution.
            - Else If [ANSWER 1 GOES HERE]
              - Set $T[i][j][k]$ = TRUE

         D. Else if s_i.color == red

            - If $j = s_i$ and $k = 1$
              - $T[i][j][k] \leftarrow$ TRUE \\ $S' = \{s_i\}$ is a solution.
            - If [ANSWER 2 GOES HERE]
              - Set $T[i][j][k]$ = TRUE

5. For $k = 0$ to 100

   (a) If $T[n][B][k]$ = TRUE

      - **Return** TRUE

6. **Return** FALSE

The Problem: Your job is to fill in ANSWER 1 and ANSWER 2. Note that both ANSWER 1 and ANSWER 2 are the conditions of an if-statement: so if this condition is satisfied, then the algorithm sets $T[i][j][k]$ = TRUE. For both ANSWER 1 and ANSWER 2 you will want to use an AND-statement, since in each one there are actually two conditions that need to be satisfied (one of them is very simple).

The overall runtime of the algorithm should be $O(nB)$.

What to write: *Don't write anything except the answer lines.* So your final answer should take the form:

- Answer 1: your answer here. (one line of pseudocode.)
- Answer 2: your answer here. (one line of pseudocode.)

HINT: To make sure that the overall runtime of the algorithm is $O(nB)$, you will want to make sure that the if-conditions you write for ANSWER 1 and ANSWER 2 can both be checked in $O(1)$ time.

RELEVANT REVIEW MATERIAL: the lecture notes and lecture videos for subset sum, including the followup lecture on subset sum with exactly C elements. Also the solution to HW 4 problem 3.

4 Problem 4 (27 points)

Definition: We define a path in a graph to be *short* if it contains ≤ 100 edges. Note that we are looking at the number of edges, not the number of vertices.

The Problem:

- INPUT
  - An unweighted DAG $G = (V, E)$
  - Two specific vertices $s, t \in V$
- OUTPUT: the *number* of different short paths in G from s to t. You don't have to output the actual paths; you just have to figure out how many of them there are.

Show a dynamic programming algorithm that solves the above problem in $O(|E|)$ time. You only need to write pseudocode, nothing else.

NOTE: reminder that you can assume in this class that all arithmetic operations (multiplication, addition, etc.) take $O(1)$ time, even if the numbers are very big.

RELEVANT REVIEW MATERIAL: Here's some material we covered that is relevant to this problem

- The last lecture I posted (Monday Dec 12)
- The solution to HW 4 problem 2. Prince did a recitation on this problem and posted a recording online. (You might have to look through announcements from the past couple of weeks to find it.)

5 Problem 5 – 15 points. EXTRA CREDIT. (Easier than normal extra credit problems.)

Consider the following problem

Problem: EvenSubsetSum(S,B)

- INPUT:
  - a set S of positive integers such that all the numbers in S are even.
  - A positive integer B
- OUTPUT: TRUE if there exists a subset $S' \subseteq S$ such that $\sum_{x \in S'} = B$. FALSE if no such S' exists.

The Problem: Show that SubsetSum(S,B) $\leq_p$ EvenSubsetSum(S,B). Here, EvenSubsetSum is the probelm above, while SubsetSum is the regular TRUE/FALSE version of subset sum that we did in class (i.e. S can contain any positive integers (even or odd) and SubsetSum(S,B) returns TRUE if there is a subset of S that sums to B.)

What to Write You need to prove that ubsetSum(S,B) $\leq_p$ EvenSubset-Sum(S,B) by using the general template we showed in class for showing that one problem is $\leq_p$ some other problem. In particular, you will need to include

- Pseudocode. *Your pseudocode should be at most 8 lines long.* (You can get away with a lot fewer lines.)
- A justification that your pseudocode is correct. Your justification needs to be convincing to someone who has not seen this problem before.

HINT: the solution I have in find slightly alters both the set S and the target B.

HINT: the solution to this problem has nothing to do with problem 3 or with the dynamic programming algorithm for subset sum. If you are trying to use dynamic programming for this one you are way off track.

REMARK: Note that Since we already know that SubsetSum is NP-complete, and you are showing that SubsetSum is easier than EvenSubset-Sum, this implies that EvenSubsetSum is also NP-complete. You don't need to write anything about this; just pointing it out.