Final Exam

(2023.12.20)

시간: 75 분

주의 사항

1.시험 "시작" 전에 페이지를 넘기지 말 것.

2.**제출하는 답안지 각 페이지에 이름 및 학번을 반드시 쓸 것.**

3."끝" 하는 말에 모든 행동을 정지 할 것.

4.[T/F]로 표시된 문제들은 해당 문제의 statement 가 참이면 [T/F]

표시의 T 에 동그라미, 아니면 F 에 동그라미를 할 것.

5. 각 [T/F] 문제 4 점, 각 XXX 4 점. 점수가 명시된 문제는 해당 점수로 채점.

부분점수 없음.

[이름]:

[학번]:

1. **[T/F]** There is no a directed graph G=(V,E), a source vertex s in V, and a set of tree edges E_π such that the unique simple path in the graph (V, E_π) from s to v for each vertex v in V is a shortest path in G, but the set of edges E_π cannot be produced by running BFS on G, no matter how the vertices are ordered in each adjacency list.

2. **[T/F]** The value u.d assigned to a vertex u by BFS is independent of the order in which the vertices appear in each adjacency list.

3. **[T/F]** If a directed graph G contains a path from u to v, and if u.d < v.d by DFS(G), then v is a descendant of u in the depth-first forest produced.

4. **[T/F]** If a directed graph G contains a path from u to v, then DFS(G) always results in v.d ≤ u.f

5. **[T/F]** A vertex u of a directed graph can end up in a depth-first tree containing only u, even though u has both incoming and outgoing edges in G.

6. **[T/F]** An undirected graph is acyclic if and only if a DFS on the graph yields no back edges. Therefore, an algorithm can be designed that checks whether or not a given undirected graph G=(V,E) contains a cycle in O(V) time, independent of |E|.

7. **[T/F]** Another way to perform topological sorting on a dag G=(V,E) is to repeatedly find a vertex of in-degree 0, output it, and remove it and all of its outgoing edges from the graph.

8. **[T/F]** For any directed graph G, the transpose of the SCC graph of G^T is not necessarily the same as the SCC graph of G.

9. **[T/F]** Suppose we replace the edges within each SCC by one simple, directed cycle and then remove redundant edges between SCC's. Since there must be at least k edges within an SCC that has k vertices, a single directed cycle of k edges gives the k-vertex SCC with the fewest possible edges. In other words, this implies that we can create a directed graph G'=(V,E') from a given directed graph G=(V,E') such that (a) G' has the same SCCs as G, (b) G' has the same SCC graph as G, and (c) E' is as small as possible.

10. **[T/F]** For the activity-selection problem (as in the text), suppose that instead of always selecting the first activity to finish, consider selecting the activity of least duration from among those that are compatible with previously selected activities. This is correct and produces an optimal solution.

11. **[T/F]** The choice made by a greedy algorithm may depend on choices so far, and possibly, if any, future choices or on the solutions to subproblems. A greedy strategy usually progresses in a top-down fashion, making one greedy choice after another, reducing each given problem instance to a smaller one.

12. **[T/F]** Let C be an alphabet in which each character c in C has frequency c.freq. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit. This fact shows that the optimal-substructure property holds for the problem of determining an optimal prefix code.

Write proper answers to the questions below. If XXX *is included, write a proper answer for each* XXX.

[**No partial Credit.**]

13. A student designed the following algorithm **A** to find the minimum length <u>in terms of the number of edges</u> in negative-weight cycles in a graph G=(V,E) with |V|=n.

    A: Run SLOW-ALL-PAIRS-SHORTEST-PATHS on G. Look at the diagonal elements of $L^{(m)}$. Return the first (encountered) value of m for which one (or more) of the diagonal elements $l_{ii}^{(m)}$ is negative. If m reaches n+1, then stop and return ∞ (declaring that there is no negative-weight cycle in G).

    (a) The following is an argument for the correctness. Assume that a negative-weight cycle exists in G. Let m* = min{ m | m ≤ n and $l_{ii}^{(m)} < 0$ for some i in V}. Then the graph has a negative-weight cycle with m* edges that goes from vertex i to itself. Because SLOW-ALL-PAIRS-SHORTEST-PATHS chekcs all **XXX1**, the cycle must be the negative-weight cycle with the minimum length. On the other hand, if there is no negative $l_{ii}^{(m)}$ element for some i in V for all m ≤ n, , there is no negative-weight cycle in the graph because **XXX2**.

    (b) What is the time complexity of the algorithm **A** by Θ in terms of n, m*?

14. For the activity-selection problem (as in the text), suppose that instead of always selecting the first activity to finish, we instead select the last activity to start that is compatible with all previously selected activities. This is a new greedy algorithm for the problem but starting from the end rather than the beginning. Suppose that we are given a set $S = \{ a_1, a_2,..., a_n \}$ of activities, where $a_i=[s_i,f_i)$. Create a set $S'=\{ a'_1, a'_2,..., a'_n \}$, where $a'_i=[f_i,s_i)$. That is, a'_i is obtained by reversing

a_i. Observe that a subset M of S is mutually compatible if and only if **XXX1**. Thus, an optimal solution for S maps directly to an optimal solution for S' and vice versa. In other words, the new algorithm when run on S provides the same answer as the GREEDY-ACTIVITY-SELECTOR algorithm when run on **XXX2**. Therefore, the new algorithm is also correct and produces an optimal solution.

15. Consider running DFS(G) as given in the text for G=(V,E) and also consider tree, forward, back, cross edges. The edge (u,v) for u,v in V is

    (a). a **XXX1** edge if and only if u.*d* < v.*d* < v.*f* < u.*f* .
    (b). a **XXX2** edge if and only if v.d ≤ u.d < u.f ≤ v.f.
    (c). a **XXX3** edge if and only if v.*d* < v.*f* < u.*d* < u.*f* .

16. In the 0-1 knapsack problem, a thief robbing a store finds n items. The ith item is worth v_i dollars and weighs w_i pounds, where v_i and w_i are integers. The thief wants to take as valuable a load as possible, but he can carry at most W pounds in his knapsack, for some integer W. The solution is then based on the optimal-substructure observation: Let i be the highest-numbered item in an optimal solution S for W pounds and items 1,...,n. Then S' = S-{i} must be an optimal solution for **XXX1**, and the value of the solution S is v_i plus the value of the subproblem solution S'. We can express this relationship as follows by an optimality equation: Define c[i,w] to be the value of the solution for items 1,...,i and maximum weight w. Then, if i=0 or w=0, c[i,w] = 0. If $w_i > w$, c[i,w] = c[i-1,w]. If i > 0 and $w \geq w_i$, c[i,w] = **XXX2**. The last case says that the value of a solution for i items either includes item i, in which case it is i plus a subproblem solution for i - 1 items and the weight excluding w_i , or doesn't include item i, in which case it is a subproblem solution for i - 1 items and the same weight. Based on the DP-equation, this problem can be solved by a DP-based algorithm in Θ(**XXX3**) time.

17. Let G= (V,E) be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex s in V . Then, during its execution, BFS discovers every vertex v in V that is reachable from the source s, and upon termination, v.d = δ(s,v) for all v in V . To prove this, assume that some vertex receives a d value not equal to its shortest-path distance. Let v be the vertex with minimum δ(s,v) that receives such an incorrect d value. Because v is not equal to s and v. d ≥ δ(s,v), we should have that v. d > δ(s,v). Furthermore, v must be reachable from s. Let u be the vertex immediately preceding v on a shortest path from s to v. Because δ(s,u) < δ(s,v), and because of how we chose v, we have u.d = δ(u,v). Putting these properties together, we have
    v.d > **XXX1**= δ(s,u) + 1 = **XXX2** (*).

Consider the time when BFS chooses to dequeue vertex u from Q in BFS. If v is **XXX3**, then it was already removed from the queue and, we have v.d ≤ u.d, contradicting inequality (*). If v is **XXX4**, then it was painted into the color upon dequeuing some vertex w, which was removed from Q earlier than u and for v.d=w.d+1. However, **XXX5**. Therfore, v.d ≤ u.d+1, contradicting (*). If v is **XXX6**, then BFS sets v.d = u.d+1, contradicting (*).

18. [6pts] Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set {1,2,...,k} used in the Floyd-Warshall algorithm. Define $d_{ij}^{(k)}$ recursively.

19. [12pts] Let $X=<x_1,x_2,...,x_m>$ and $Y=<y_1,y_2,...,y_n>$ be sequences, and let $Z=<z_1,...,z_k>$ be any LCS of X and Y . In the longest-common-subsequence problem, we are given X and Y and wish to find a maximum length common subsequence Z (LCS) of X and Y. State the optimal-substructure property of an LCS by using X, Y, Z and their prefixes. Given a sequence X, the ith prefix of X, for i=1,...,m, is denoted by $X_i=<x_1,x_2,...,x_i>$.

20. [8pts]] Again, $X=<x_1,x_2,...,x_m>$ and $Y=<y_1,y_2,...,y_n>$ given sequences. The b table returned by the LCS-LENGTH algorithm (as in the text), which implements the optimal substucture property given in the above problem, enables us to construct an LCS of X and Y. We begin at b[m,n] and trace through the table by following the arrows. Write the recursive procedure PRINT-LCS(b,X,i,j) that allows us to print out an LCS of X and Y in the forward order. The initial call is PRINT-LCS(b,X,X.length,Y.length).