Quiz #5: Solutions

1. If a DP algorithm needs to solve $\Theta(n^2)$ subproblems, like the ones discussed in lectures, then its space complexity is at least $\Theta(n^2)$ because we need to store the results of all subproblems.

   Solution: False. A counterexample is the linear-space alignment. This is pretty common.

2. If a DP formulation consists of $2n^2$ subproblems, then we need to solve most of them before we can get the final answer, i.e., $\Theta(n^2)$ subproblems excluding some trivial boundary cases.

   Solution: False. For example, in the edit distance problem, if we are sure the distance won't exceed $k = o(n)$ (e.g., a constant or $\Theta(\log n)$), we can skip subproblems $\Theta(k)$-unit far away from the diagonal line, i.e., $\mathrm{OPT}(i, j)$ with roughly $|i - j| > k$, and only solve the rest $O(nk)$ subproblems. This is not very common, though, because we usually don't know k in advance, and we have to start over to try a larger k.

   In general, if the recurrence is equivalent to that of the shortest distance problem and we believe the ultimate distance is relatively short, we can adopt Dijkstra's algorithm to solve less subproblems at the cost of extra $O(\log n)$ time at every visited subproblem.

3. Assume the recurrence is $f(n) = \max_{i:1\le i<n} \{f(i) + \mathrm{cost}(i, n)\}$ and assume the runtime of $\mathrm{cost}(i, n)$ is $O(g(n))$. And then, we can solve $f(n)$ in $O(ng(n))$ time, if we have already solved $f(1), f(2), \ldots, f(n-1)$, and if we don't use any acceleration.

   Solution: True. We need to solve $O(n)$ tasks and each task takes $O(g(n))$ time.

4. For any DP formulation, we have to solve the subproblems in one unique order.

   Solution: False. For example, in the sequence alignment problem, we can solve them row by row, column by column, or in order of increasing $i + j$ value. Actually, any topological order in the dependency graph works. However, some orders are better than the rest in that we can reduce the space complexity and/or accelerate the DP algorithm in certain orders.

5. The single-source shortest path problem in a DAG can be formulated as a DP.

   Solution: True. The boundary case are 1) $\mathrm{dist}(s) = 0$ for the source s, and 2) $\mathrm{dist}(u) = +\infty$ for any other nodes without incoming edges. The recurrence is $\mathrm{dist}(u) = \min_{v:(v,u)\in E}\{\mathrm{dist}(v) + w(v, u)\}$. We can solve them in any topological order.

6. For any two subproblems a and b in a DP formulation, $\mathrm{OPT}(a)$ and $\mathrm{OPT}(b)$ are allowed to depend on each other mutually.

   Solution: The mutual dependency between a and b is allowed only if we can eliminate at least one dependency before we solve any of a and b. Similar to the previous question, the shortest path problem in any graph can also be formulated as a DP, and there is a natural order of these subproblems, which is in increasing distance. But the values of subproblems, i.e., the distances, depend on each other if the nodes are in the same cycle. This means we don't know any feasible order initially. Following Dijkstra's algorithm, when we visit node u and calculate its distance, we correctly eliminate its dependency on any unvisited nodes. This indicates, however, we can figure out one feasible order as we solve subproblems one by one.

   In contrast, if edges may have negative weights, we can't correctly eliminate dependencies before we solve each subproblem, which means we have to solve them more than once or even infinitely many times, so this is not regarded as a proper DP. When there are mutual dependencies and the recurrence is more complicated, we can't even tell if there exists a unique solution.

7. We can solve the knapsack problem by A* with a zero heuristic or any non-DP-like heuristic, although the runtime might be exponential in n and/or W.

Solution: True. We use exactly the same recurrence and boundary case, but we don't store the values of subproblems and solve it again every time we meet one. Basically, we are enumerating every combination. Since there are roughly 2^n combinations, the overall runtime is $O(2^n)$ or $O(n2^n)$.

If we store intermediate values, i.e., we allow memorization, then this A* becomes DP. We usually don't store them, because we can't afford the exponential space, although the space is polynomial in this question.

8. The configuration of a maximum flow in a network is always unique.

Solution: False. In the following graph, the capacity of every edge is one. The max flow can be $s \to 1 \to 3 \to t$ and $s \to 2 \to 3 \to t$, and also can be any weighted combination of them.