IITM-CS2800-2025 : Design and Analysis of Algorithms

Practice Problem Sheet #9 Released On : Apr 26, 10 PM

1. Say we want to compute the length of the shortest path between every pair of vertices in a weighted graph. This is called the *all-pairs shortest path problem*. If we use the Bellman-Ford algorithm directly, which takes $O(mn)$ time, for all $n$ possible destinations $t$, this would take time $O(mn^2)$ time. We will now see a Dynamic-Programming algorithm that runs in time $O(n^3)$

   How do we divide it into subproblems?

   (a) One way would be to do so by length, i.e., consider paths of length $k$ in terms of paths of length $k-1$ plus another edge. Write this down as a recurrence and as an algorithm. Analyse the algorithm.

   (b) A more efficient idea is to consider which vertices we are allowed to use. The idea is that instead of increasing the number of edges in the path, we will increase the set of vertices we allow as intermediate nodes in the path. So, lets try the subproblems denoted by $D[u][v][k]$ to be the length of the shortest path from $u$ to $v$ using intermediate vertices $\{1, 2, \ldots, k\}$.

   (c) To show the recurrence relation, we need to consider two cases. For the pair $(u, v)$, either the shortest path using the intermediate vertices $\{1, 2, \ldots, k\}$ goes through $k$ or it does not. Use the above consideration to prove:

   $$D[u][v][k] = \min\{D[u][v][k-1], D[u][k][k-1] + D[k][v][k-1]\}$$

   Write down an appropriate basecase.

   (d) Argue that the above formulation leads to a DP based algorithm that runs in time $O(n^3)$.

   (e) The above DP uses $O(n^3)$ space as well. Can you reduce the space to $O(n^2)$ using a similar observation that we made in class for Bellman-Ford algorithm?

2. Given a complete directed graph $G$ with $n$ vertices with weights assigned to all $m = n^2$ edges, the Traveling Salesperson Problem (TSP) asks to find the shortest route that visits all vertices in a graph exactly once and returns to the start. We will try to design algorithms for this problem:

   (a) One possible solution to the problem is to try out all possible routes. How many routes are possible with the above constraint. Write down a running time bound for this trivial algorithm in terms of $n$. Show that this is asymptotically $2^{O(n \log n)}$.

   (b) We will do better than above using Dynamic Programming strategy with an algorithm running in time $O(2^n n^2)$.

   How do we design the subproblems? This one is trickier and we will try a couple of the possibilities to get it right. Suppose we want to make a tour of some subset of nodes

$S$. Can we relate the cost of an optimal tour to a smaller version of the problem? In particular, suppose we call out a particular vertex $t$ , and then ask whether it is possible to relate the cost of the optimal tour of $S \setminus \{t\}$ and $S$. It doesn't seem so, because its not clear how we would splice $t$ into the tour formed by $S \setminus \{t\}$ without additional information.

*What additional information can we add?* Firstly, it may be easier to work with paths than cycles when we add vertices incrementally. How about adding starting and ending vertices? So lets fix an arbitrary starting vertex $x$ , and now consider the cheapest path that starts at $x$, visits all of the vertices in $S$ and ends at a specific vertex $t$. Can we find any substructure in this much more specific object? Well, yes, we know that the optimal path from $x$ to $t$ must have some second-last vertex $t'$ and the path from $x$ to $t'$ must be an optimal such path using the vertices $S \setminus \{t\}$.

Define the subproblem as:

$C(S, t) =$ The min-cost path starting at $x$ and ending at $t$ going through all vertices in $S$

(c) It turns out in the above case, the final solution that we want is not directly one of the subproblems (like in the Bellman-Ford algorithm case, we knew that we want to compute $OPT(n-1, s)$. Write down the solution to the TSP problem in terms of $C(S, t)$ for appropriate $S$'s and $t$'s (you may need many of them).

(d) Prove the recurrence:

$$C(S,t) = \begin{cases} w(x,t) & \text{if } S = \{x,t\} \\ \min\limits_{\substack{t' \in S \\ t' \notin \{x,t\}}} C(S \setminus \{t\}, t') + w(t',t) & \text{otherwise} \end{cases}$$

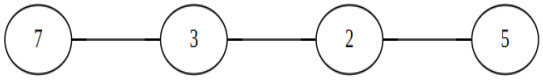
3. Recall the maximum independent set (MaxIS) that we discussed in class. For an undirected graph $G(V, E)$, a subset of vertices $S \subseteq V$ is said to be independent if $\forall (u, v) \in E$, either $u \notin S$ or $v \notin S$. We discussed about finding the maximum sized independent set in a given graph $G$ and stated that we do know only exponential time algorithms (in terms of $n$) for this problem.

   In this problem, we will consider a weighted version of this problem denoted by MaxWIS. We are given an undirected graph $G(V, E)$ with non-negative weights assigned to vertices $w : V \to \mathbb{Z}^+$, we want to find independent set $S \subseteq V$ with maximum weight, where the weight of a subset $S$ is defined to be the sum of the weights of the vertices contained in $S$ - that is, $w(S) = \sum_{u \in S} w(u)$.

   (a) Can you provide a reduction from the MaxIS problem to MaxWIS problem? As discussed in class - informally this means that given an instance of MaxIS problem to solve, you should produce an instance of MaxWIS problem (that is define the graph and the weight function) such that a solution to your MaxWIS instance will give you a solution to the MaxIS instance as well. Write down the description of the instance of MaxWIS problem that you produce as a part of the reduction, as precisely as you can.

   (b) Note that the above reduction implies that designing an algorithm for MaxWIS problem is probably hard. We will now design an algorithm in a special case where the given graph

is simply a path graph with weights[1] - see an example graph shown below.




We will use dynamic programming. First step is design subproblems and a recurrence relation to "smaller subproblems". A reasonable subproblem is to consider the independent set when the path contains only first $i$ vertices. *Hint: There are only two possibilities for the final vertex with regards to the independent set of maximum weight - either it is contained in it, or it is not.* Design a DP based algorithm and write down the pseudocode.

4. We practice reductions explicitly in this exercise.

   We have seen the problem of multiplying two $n \times n$ matrices and designed an algorithm which runs in time $O(n^3)$. We saw improved algorithms for this problem using divide and conquer strategy.

   We consider a special case of multiplying two *symmetric* $n \times n$ matrices - recall that an $n \times n$ matrix is symmetric if $(i, j)$-th entry is same as the $(j, i)$-th entry for $i, j \in \{1, 2, \ldots n\}$. Now the problem of multiplying two given symmetric matrices is potentially an easier problem. We show in this exericse that it is not the case.

   We give a reduction from the problem of multiplying two matrices to the case of multiplying two symmetric matrices. Given two $n \times n$ matrices $A$ and $B$, can you produce two symmetric matrices $A'$ and $B'$ such that if you are given $A'B'$, you can simply "read off" the matrix $AB$ from that result? Explicitly write down $A'$ and $B'$ in terms of $A$ and $B$ in such a way that computing $A'$ and $B'$ given $A$ and $B$ can be done efficiently.

5. We practice reduction as a tool for algorithm designers in this question.

   Suppose that you have a collection of n locations in $2D$ space expressed as $(x, y)$ pairs. You are interested in determining the $k$ points closest to your current location $p$. Design an $O(n)$-time algorithm for finding the $k$ points closest to $p$. For simplicity, you can assume that no two points are at the same distance to $p$. (*Hint: can you recast this question as a question of finding the k-th smallest element in an array? Think about what the array should contain so that the k smallest elements will give you the required k points. Write down your argument formally.*).

[1] Can you solve maximum independent set when the path graph is unweighted easily?