Analysis of Algorithms - Homework II (Solutions)

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Problems

1. In the Knapsack problem, you are given a knapsack of capacity W and n objects $\{o_1, o_2, \ldots o_n\}$ with respective weights $\{w_1, w_2, \ldots, w_n\}$ and respective profit values $\{p_1, p_2, \ldots p_n\}$. The goal is to pack the objects into the knapsack in a manner that maximizes the profit of knapsack, without violating its capacity constraint. In class, we showed that if we are permitted to choose fractions of objects, then the problem can be solved by a greedy strategy. The $0/1$ Knapsack problem is a variant of the knapsack problem in which you *cannot* choose fractions of objects, i.e., each object is either selected or not. Argue with a counterexample that the greedy strategy does not work for the $0/1$ Knapsack problem.

   Solution: Consider the following input instance $\langle o_1, o_2, o_3 \rangle$:

   (a) $W = 5$.

   (b) $\langle p_1, p_2, p_3 \rangle = \langle 6, 10, 12 \rangle$.

   (c) $\langle w_1, w_2, w_3 \rangle = \langle 1, 2, 3 \rangle$.

   The greedy strategy for fractional knapsack would pick o_1 and o_2, whereas the optimal solution is clearly o_1 and o_3. □

2. Let $A = \{a_1, a_2, \ldots, a_n\}$ denote a set of positive integers that add up to N. Design an $O(n \cdot N)$ algorithm for determining whether there is a subset B of A, such that $\sum_{a_i \in B} a_i = \sum_{a_i \in A - B} a_i$.

   Solution: Observe that the problem is asking whether A can be partitioned into two parts which sum up to $\frac{N}{2}$. Without loss of generality, we assume that N is even; since if N is odd, the answer is "no"!

   We thus need to determine whether a subset of A adds up exactly to $\frac{N}{2}$.

   Accordingly, we define

   $$\begin{aligned} B[i,j] &= \textbf{true}, \text{ if some subset of } \{a_1, a_2, \ldots, a_i\} \text{ adds up exactly to } j. \\ &= \textbf{false}, \text{ otherwise.} \end{aligned}$$

   The entry of interest in this table is $B[n, \frac{N}{2}]$; if this entry is **true**, then there is indeed some subset of A which sums up to $\frac{N}{2}$.

   When filling in the entry for $B[i,j]$, we either select a_i to be included in the chosen subset or a_i is excluded. In the former case, $B[i,j] = B[i-1, j-a_i]$, since there must exist a subset of $\{a_1, a_2, \ldots, a_{i-1}\}$ that sums to $j - a_i$. In the latter case, $B[i,j] = B[i-1,j]$. Since $B[i,j]$ is **true**, if either choice is **true**, we conclude that

   $$B[i,j] = B[i-1, j-a_i] \vee B[i-1,j].$$

   The table $\mathbf{B}$ is filled up in bottom-up fashion, i.e., first $B[1,k]$ is computed for $k = 1, 2, \ldots, \frac{N}{2}$, then $B[2,k]$ is computed and so on. The computation of a single entry requires at most two lookups and an arithmetic computation,

i.e., each entry of the table can be computed in $O(1)$ time. Since there are $n \cdot \frac{N}{2}$ entries in the table, the entire table can be filled in $O(n \cdot N)$ time. □

3. Let G be an undirected spanning tree on n vertices and m edges. Argue that $m = n - 1$.

   Solution: We use mathematical induction to establish the above theorem.

   When $n = 1$ and G is a tree, we must have $m = 0$ and the basis is proven.

   Assume that whenever G is an undirected spanning tree on at most k vertices, the number of edges in G is one lower than the number of vertices.

   Now let G be an undirected spanning tree on $(k + 1)$ vertices. Remove any edge e in G; the removal of e breaks G into two non-empty disjoint components G_1 and G_2, since G is minimally connected. These two components are themselves spanning trees on $|G_1|$ vertices and $|G_2|$ vertices respectively, since they are also minimally connected. Clearly, $|G_1| \leq k$ and $|G_2 \leq k$ and as per the inductive hypothesis, they must have $|G_1| - 1$ and $|G_2| - 1$ edges respectively. It follows that G has $|G_1| - 1 + |G_2| - 1 + 1 = |G_1| + |G_2| - 1 = (k + 1) - 1 = k$ edges.

   Using the *second* principle of mathematical induction, we conclude that the theorem holds for all $n \geq 1$. □

4. Consider the DFS algorithm discussed in class. Argue that for any pair of nodes u and v, precisely one of the following two possibilities holds:

   (i) $[d[u], f[u]] \cap [d[v], f[v]] = \emptyset$.

   (ii) Either $[d[u], f[u]]$ is completely contained in $[d[v], f[v]]$ or vice versa.

   Solution: First note that both the above possibilities cannot simultaneously hold, since they describe mutually exclusive situations.

   Let us assume that neither possibility holds and suppose that there exists a pair of vertices (u, v), such that $d[u] < d[v] < f[u] < f[v]$. Since $d[u] < d[v]$, vertex u was discovered by the depth-first search before vertex v. Since $d[v] < f[u]$, it follows that v was discovered when u was yet to be "finished"; consequently v must be a descendant of u in the depth-first search tree. However, if v is a descendant of u, then as per the definition of depth-first search, v must be "finished" before u is, i.e., we have contradicted the hypothesis.

   In other words, we cannot have a vertex pair (u, v) such that $d[u] < d[v] < f[u] < f[v]$ and hence precisely one of the enumerated possibilities holds. □

5. Consider a directed weighted graph in which all edges, except those directed out of the source are positive. The edges out of the source can be positive or negative. Will Dijkstra's algorithm produce the correct shortest path distances when run on this graph? Justify your answer with a proof or counterexample.

   Solution: Dijkstra's algorithm works on this graph as well.

   Let $G = \langle V, E, w \rangle$ denote the graph, s denote the source vertex and Q denote the priority queue from which the vertices are extracted in increasing order of their $d[]$ values as per Dijkstra's algorithm. We first observe that there cannot be any negative cost cycles in G so that the shortest path from s to any vertex $u \in V$ is well-defined.

   Claim 1.1 *For all vertices, $u \in V$, $d[u] = \delta(s, u)$, when u is extracted from the queue.*

   Proof: The claim is clearly true for $u = s$ since in this case $d[u] = \delta(s, u) = 0$ and s is first vertex to be extracted from Q.

   Let $p = \langle s, v_1, v_2, \ldots, u \rangle$ denote the shortest path from the source s to some vertex u. If this path consists of a single edge then clearly $d[u] = \delta(s, u)$ when s is extracted from Q and its edges are relaxed. By the upper-bound property, $d[u]$ will not decrease as a result of future relaxations and hence the claim holds, when u is extracted from Q.

   Now consider the case in which the shortest path from s to u consists of r edges. We can decompose this path as $\langle (s, v_1), (v_1, v_2), \ldots, (v_{r-1}, v_r = u) \rangle$. By the principle of optimality, (s, v_1) must be shortest path from s to v_1 and $\langle v_1, v_2, \ldots u \rangle$ is the shortest path from v_1 to u in G. Once s is extracted from Q and its edges are relaxed, $d[v_1] = \delta(s, v_1)$. The problem of finding the shortest path from s to u is now identical to the problem of finding

the shortest path from v_1 to u. However, the edges on all paths from v_1 to u have positive weights and hence the correctness of Dijkstra's algorithm establishes that when u is extracted from Q, $d[u] = \delta(s, u)$. $\square$

It follows that Dijkstra's algorithm produces the correct output when run on G. $\square$