Tutorial: Dynamic Programming

Problem 1. Design a DP algorithm for the following problem. We are given a sequence of positive weights $w_1, \ldots, w_{2n}$. We want to partition all numbers from 1 to $2n$ into n pairs $(a_1, b_1), \ldots, (a_n, b_n)$ so that

- Each number from 1 to $2n$ appears in exactly one pair.
- $1 \leq b_i - a_i \leq 2$ for all $i \in \{1, \ldots, 2n\}$

For example, if $n = 2$, then $(1, 2), (3, 4)$ and $(1, 3), (2, 4)$ are feasible solution. The goal is to find a feasible solution that maximizes $\sum_{i=1}^{n} w_{a_i} w_{b_i}$.

Solution.
Let $A[k]$ be the maximum weight of a feasible partition of $1, \ldots, 2k$. $A[1] = w_{a_1} w_{b_1}$. For $k > 1$, there are two choices for which number to pair $2k$ with. It can be either paired with $2k - 1$, or with $2k - 2$. In the first case, we are left with the numbers $1, \ldots, 2(k-1)$. In the second case, $2k - 1$ must be paired with $2k - 3$ and we are left with the numbers $1, \ldots, 2(k-2)$. So we get

$$A[k] = \max\{w_{2k} w_{2k-1} + A[k-1], w_{2k} w_{2k-2} + w_{2k-1} w_{2k-3} + A[k-2]\}.$$

There are n subproblems, and we need constant time for each, hence the running time of a DP implementation of the above recursion is $O(n)$.

Problem 2. We are given a tree $T = (V, E)$ and positive edge weights w_e for $e \in E$. We say that a set of edges $A \subseteq E$ is good if for every $e \in A$, there exists exactly one edge $e' \in A - \{e\}$ that shares a vertex with e The goal is to find a good subset of edges A that maximizes $\sum_{e \in A} w_e$. Design a DP algorithm for the problem.

Hint: Define two DP tables $A[u]$ and $B[u]$ indexed by vertices $u \in V$.

Solution.
Let $A[u]$ be the maximum weight of a good subset of edges for the subtree T_u rooted at u. Let $B[u]$ be the maximum weight of a good subset A of edges for the subtree T_u such that there is exactly one edge in A incident to u. If u is a leaf, then $A[u] = 0$ and $B[u] = 0$. If u is not a leaf, then there are three cases: Either none, one or two edges incident to u are selected to be in a good subset of maximum weight. In the second case, if (u, w) is the edge selected, then there must be exactly one edge other than (u, w) incident to w that is selected. In the third case, if (u, w_1) and (u, w_2) are the edges selected, then there cannot be edges other than (u, w_1) and (u, w_2) that are incident to w_1 and w_2. So, we see that

$$A[u] = \max \begin{cases} \sum_{v \text{ child of } u} A[v], \\ \max_{w \text{ child of } u} w_{(u,w)} + B[w] + \sum_{v \neq w \text{ child of } u} A[v], \\ \max_{w_1, w_2 \text{ children of } u} w_{(u,w_1)} + w_{(u,w_2)} + \sum_{z \text{ child of } w_1} A[z] + \sum_{z \text{ child of } w_2} A[z] \\ + \sum_{v \neq w_1, w_2 \text{ child of } u} A[v], \end{cases}$$

$$B[u] = \max_{w \text{ child of } u} w_{(u,w)} + B[w] + \sum_{v \neq w \text{ child of } u} A[v].$$

During the execution of the algorithm every node is visited at most as many times as its the number of its siblings, giving a running time quadratic on the size of T.