## EECS 336: Algorithms — Problem X.1 Solution

Before we present the four-part DP solution, we need some notation. Note that you can represent a planting solution as a $4 \times n$ matrix of 1s and 0s - 1 in entry $(i, j)$ if there is a tree in that spot, 0 otherwise. Using this notation, it will be helpful to enumerate all the possible configurations of a column of this matrix. The set of possible configurations is $C = \{0000, 0001, 0010, 0100, 1000, 1001, 1010, 0101\}$, where, for example, 0101 denotes "plant trees in rows 2 and 4."

The intuition driving our DP algorithm will be that the configuration of column $j$ dictates the possible configurations for the rest of the matrix. In particular, column $j+1$ must be compatible with the configuration of column $j$: if column $j$ is 0101, the column $j+1$ may only be $0000, 0010, 1000$, or 1010. For each configuration $c \in C$, let comp$(c)$ be the set of configurations in $C$ which are compatible with $c$. There are finitely many configurations in $C$, so this can be hard-coded into our algorithm.

Finally, we will need notation for the total quality you get from planting configuration $c = c_1c_2c_3c_4$ in column $j$. This can be computed as $\text{qual}(j, c) = \sum_{i:c_i=1} q_{ij}$. For example, $\text{qual}(j, 1001) = q_{1j} + q_{4j}$.

**Subproblem:** For any $j$ and $c \in C$, let OPT($j$,$c$) be the optimal quality from plantings of columns $j, \ldots, n$ given that column $j$ must be compatible with a column of configuration $c$ on the left.

**Recurrence:** $\text{OPT}(j,c) = \max_{d \in \text{comp}(c)} [\text{qual}(j,d) + \text{OPT}(j+1,d)]$

**Proof of Recurrence:** To solve the subproblem for $i$ and $c$, we need to decide: of all the configurations of column $j$ which are compatible with $c$, which do I choose? If you choose $d$, you get qual($j$,$d$) in tree growth from column $j$, and then you must optimally plant columns $j+1$ through $n$, and column $j+1$ must be compatible with $d$. The optimal solution will chooose the best of these options.

**Base Cases:** $\text{OPT}(n+1,c) = 0$ for all $c$.

**Iterative Algorithm:**

> Memo[][] = new int[n+1][8] [Note: there are eight configurations.]
>
> Compute qual($j$,$c$) for all $j$ and $c$. Store these in an array.
>
> Compute comp($c$) for each $c \in C$.
>
> For all $c \in C$, set Memo[$n+1$][$c$]=0.
>
> For j from $n$ down to 1
>
> > For all $c \in C$, set Memo[$j$][$c$]=$\max_{d \in \text{comp}(c)}$ (qual($j$,$d$)+Memo[$j+1$][$d$])
>
> Return $\max_{c \in C}$ Memo[1][$c$]

**Runtime:** Computing qual($j$,$c$) for all $j$ and $c$ takes $O(n)$ time, as $|C| = 8$. Computing comp($c$) takes $O(1)$ time. Filling in the base cases also takes $O(1)$ time. The main work comes from filling in the rest of the memo table. The table is of size $O(n)$, and it takes $O(1)$ work to fill in the table, so the runtime is $O(n)$.