---

1. You are given n items with non-negative integer weights $w(i)$ and a positive integer W. You have to find a subset $S \subseteq \{1, ..., n\}$ of indices such that $\sum_{i \in S} w(i)$ is maximised subject to $\sum_{i \in S} w(i) \leq W$.

2. You need to drive your electric car a total of M miles. It will need to stop along the way to recharge: there are n recharging stations on the road, at distances $m[1] < m[2] < m[3] < \cdots < m[n]$ from the starting point. The cost for recharging varies from station to station; it is $c[i]$ at the i^{th} station. The car can travel at most D miles on a single charge. Your goal is to find which charging stations you should stop to reach the destination while paying as little as possible. Assume that you begin with a full charge (at no cost). You may also assume that there is a solution: that is, the distance between consecutive charging stations is at most D and $m[1] \leq D$ and $M - m[n] \leq D$.

   (*For example, suppose* $M = 100$*,* $D = 40$*, and there are* $n = 6$ *stations at distances* $m[1:6] = [10, 20, 30, 50, 60, 80]$ *with costs* $c[1:6] = [5, 20, 30, 5, 10, 10]$*. Then the optimal solution is to stop at stations* $1, 4, 6$*, for a total cost of* 20*.*)

   Here is a subproblem that can be used for a Dynamic Programming solution: for $1 \leq i \leq n$, define

   $$T[i] = \text{optimal cost starting at position } m[i] \text{ on a full charge.}$$

   (*For instance, in the example above,* $T[5] = 0$ *since if you begin at position* $m[5] = 60$*, then you don't need to recharge before getting to the destination.*)

   For convenience, define $T[0]$ as the minimum cost from the starting point.

   (a) Give the full array $T[0:6]$ for the example above.

   (b) Give a rule by which the answer to any subproblem $T[i]$ can be determined once answers are known for "smaller" subproblems.

   (c) For a general instance with n stations, in what order should the subproblems $T[0], T[1], \ldots, T[n]$ be solved?

   (d) Write down a dynamic programming algorithm that implements this rule and returns the optimal cost. (You do not need to return the chosen locations.)

   (e) What is the running time of your algorithm?

3. A subsequence is *palindromic* if it is the same whether read left to right or right to left. For instance, the sequence

   $$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

   has many palindromic subsequences, including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is *not* palindromic). Devise an algorithm that takes a sequence $x[1 \ldots n]$ and returns the (length of the) longest palindromic subsequence. Its running time should be $O(n^2)$.