CS170 - Spring 2010 Section April 5th, 2010

Dynamic Programming

1. The LAF (League of Acrobatic Ferrets) is putting together a tricky routine where a chain of ferrets will swing from a bar K centimeters above the crowd (the first ferret holds the bar and the rest hold the ankles of the ferret above it). The goal of the routine is to have the bottom ferret swing exactly 1 centimeter from the crowd (i.e., the ferret chain should have total length $K-1$), thereby creating a superbly thrilling experience. Each ferret i out of the n members of LAF has a length l_i in centimeters.

   a. Design an dynamic programming algorithm to determine whether this stunt can be pulled off, and which ferrets should do it.

   b. Identify other DP problems that are equivalent to this problem (and don't involve ferrets) and explain briefly what the equivalence is.

   Solution

   a. We solve a more general problem: you are given totals $T_1, \ldots, T_k$ and sizes $a_1, \ldots, a_n$ such that $\sum_j T_j = \sum_i a_i := T$. We wish to output disjoint subsets $S_1, \ldots, S_k$ of $[n]$ such that $\sum_{i \in S_j} a_i = T_j$ for all j. That is, we wish to partition the $\{a_i\}$ to make the sums $\{T_j\}$. Let $L[t_1, \ldots, t_k, i]$ represent whether or not we can partition $a_1, \ldots, a_i$ to sum to the $\{t_j\}$. Then we do the following:

      1. Initialize $L[0, \ldots, 0, 0] = \text{true}$.
      2. Recursively (with caching!) compute

      $$L[t_1, \ldots, t_k, i] = \max_{j : t_j - a_i \geq 0} L[t_1, \ldots, t_{j-1}, (t_j - a_i), t_{j+1}, \ldots, t_k, i-1]$$

         and record the j that worked (if any) in $J[t_1, \ldots, t_k, i]$, starting with $L[T1, \ldots, T_k, n]$.
      3. If we return 'true', we can retrace our steps for the solution.

      The runtime of this algorithm is $\Theta(nk\Pi T_j)$, since there are up to $n\Pi T_j$ entries in our table, and we loop over all k choices for each entry. Actually, it's really only $\Pi_{j<k} T_j$ since if we use recursion the total t_k is determined in this setting by the first $k-1$ sums.
      To solve the k-partition problem, set $T_i = T/k$ for $i \in [k]$ (where $T = \sum a_i$ as above). To solve the ferret problem, set $T_1 = K - 1$ and $T_2 = T?T_1$ (here we use $a_i = l_i$ of course). The runtimes for these cases come out to $\Theta(nT^{k-1})$ and $\Theta(nK)$, respectively.

2. Correction of last section: You are given a sequence of n positive integers, $a_1, \ldots, a_n$, say $(4, 5, 3, 2, 1)$. You are asked to choose among them a set of numbers *no two of which are adjacent*, so that the sum is as large as possible. In the present case, the correct answer is $4, 3, 1$. You want to solve this problem by dynamic programming, with prefixes of the sequence as subproblems.

   (d) Suppose that each number a_i comes with its *reach* r_i such that, if a_i is included in the sum, then any other number between $i - r_i$ and $i + r_i$ must be excluded. That is, the original problem was the $r_i = 1$ case. Write the formula for $S(i+1)$ now.

Solution

We define the subproblem $S(i)$ to be the maximum sum achievable when using the elements $a_1, \ldots, a_i$, and including a_i in the chosen set.
The formula for this subproblem is

$$S(i) = a_i + \max_{j<i \text{ s.t. } \max\{r_i, r_j\}<(i-j)} S(j)$$

After computing $S(i)$ for $i = 1, \ldots, n$ we find the maximum $S(i)$ and trace back from it's solution to find the subset of elements we pick.