Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department of Computer Science
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

17th November 2016

Data Structures & Algorithm — Solutions to Sheet 8 — AS 16

Solution 8.1 *Mars mission.*

Definition of the DP table: We use an $m \times n$-table T, and $T[i,j]$ contains the maximum achievable value of the rock samples on a south-east path from $(1,1)$ to (i,j).

Computation of an entry: For (i,j) with $1 < i,j \leq n$ we observe the following: if a path ends at the position (x,y), then we have get there either from above or from the left. Thus, the maximum value we can achieve at that position is exactly

$$T[i,j] = A[i,j] + \max\{T[i-1,j], T[i,j-1]\}, \tag{1}$$

since the value $A[i,j]$ of the rock sample at the position (i,j) must be added to the maximum value of the samples collected on the way to this position. For the cases at the top and at the left side we define:

- $T[1,1] = A[1,1] = 0$, because $(1,1)$ is the starting position,
- $T[i,1] = A[i,1] + T[i-1,1]$ for $i > 1$, because the position $(i,1)$ can only be reached from above from the position $(i-1,1)$,
- $T[1,j] = A[1,j] + T[1,j-1]$ for $j > 1$, because the position $(1,j)$ can only be reached from the left from the position $(1,j-1)$.

Calculation order: The entry $T[i,j]$ depends only on entries for smaller values of i and j. Therefore, the entries can be calculated, for example, increasingly in the value of $i = 1,\ldots,m$ and for the same i, increasingly in the value of $j = 1,\ldots,n$.

Extracting the solution: In the end, the maximum achievable *value* of the rock samples is stored in the entry $T[m,n]$. To reconstruct the path itself, we start in the entry $T[m,n]$ and output (m,n). Next, we check if $T[m,n] = A[m,n] + T[m-1,n]$. If this is the case, then we came from above, i.e., from the entry $(m-1,n)$, and we continue there. Otherwise, $T[m,n] = A[m,n]+T[m,n-1]$, and we came from the left, so we proceed with the entry $(m,n-1)$. The reconstruction continues until the starting position $(1,1)$ is reached. In the end, we have determined the positions of the path in reverse order.

Running time: The table has size $m \cdot n$ and each entry can be computed in time $\Theta(1)$. Thus, the computation of the maximum value can be done in time $\Theta(mn)$.

The reconstructed path has length $m + n - 1$. Since for each position we can decide in time $\Theta(1)$ whether we came from the top or from the left, the whole path can be determined in time $\Theta(m+n)$. Together with the time needed to fill the table, we get the overall running time of $\Theta(mn)$.

Solution 8.2 *Hike.*

For simplicity, we define $G := \sum_{i=1}^{n} g_i$.

Definition of the DP table: We use a table T of size $(n+1) \times (\lfloor G/2 \rfloor + 1)$ with entries that are either "true" or "false". For $0 \le i \le n$ and $0 \le g \le \lfloor G/2 \rfloor$ let $T[i,g] =$ true if and only if there exists a subset $K \subseteq \{1, \dots, i\}$ of the first i items with a total weight of exactly g, i.e. $\sum_{k \in K} g_k = g$.

Computation of an entry: We distinguish three cases:

- $T[i,0] =$ true for each $i \in \{0, \dots, n\}$, because the set $\{1, \dots, i\}$ contains the empty set $K = \emptyset$ with weight 0.
- $T[0,g] =$ false for each $g \in \{1, \dots, \lfloor G/2 \rfloor\}$, because we cannot get a total weight $g > 0$ using the empty set.
- For all $i \in \{1, \dots, n\}$ and $g \in \{1, \dots, \lfloor G/2 \rfloor\}$ we set

$$T[i,g] = \begin{cases} T[i-1,g] & \text{if } g_i > g \\ T[i-1,g] \vee T[i-1,g-g_i] & \text{otherwise.} \end{cases}$$

  For $g_i > g$, the item i weights more than the allowed total weight of the selection. Hence, this item cannot be selected and it holds that $T[i,g] = T[i-1,g]$. Otherwise, item i can be used or not used. That is, we either need to get weight $g - g_i$ with the items $\{1, \dots, i-1\}$ (if we use item i), or we have to get weight g (if we do not use i).

Calculation order: We compute the entries $T[i,g]$ with increasing value of i and for equal values of i with increasing value of g.

Extracting the solution: For each $g \in \{0, \dots, \lfloor G/2 \rfloor\}$ the entry $T[n,g] =$ true if and only if there exists a subset of items in $\{1, \dots, n\}$ with total weight g. First, we determine the maximum value g_{max} with $T[n, g_{max}] =$ true, i.e. the largest possible *weight* of a backpack with weight $\le \lfloor G/2 \rfloor$.

We set $i \leftarrow n$ and $g \leftarrow g_{max}$ and continue as follows. If $i = 0$, we have found a subset with weight g_{max}. Otherwise, we compare $T[i,g]$ and $T[i-1,g]$. If $T[i,g] = T[i-1,g]$, we set $i \leftarrow i-1$ and continue. Otherwise, it must hold that $T[i,g] = T[i-1,g-g_i]$. We print i and set $i \leftarrow i-1$ and $g \leftarrow g - g_i$, and continue. We put all items that we printed during the procedure in the backpack of Alice and the other items in the backpack of Bob (or vice versa).

Running time: The DP table has $\Theta(nG)$ entries and the value of each entry can be computed in constant time using the previously computed entries. Therefore, the table can be filled in $\Theta(nG)$ running time. To reconstruct a solution, we need n additional steps. Each step takes constant time. The overall running time of the algorithm is in $\Theta(nG)$, which is pseudo-polynomial.

Solution 8.3 *Numerical Puzzle.*

a) *Definition of the DP table:* We define a table S of size $(n+1) \times (\sigma+1)$. The entry $S[n', \sigma']$ is 'true' if the sum σ' can be obtained using the first n' digits, otherwise it is 'false'.

Computation of an entry: We set $S[0][0] \leftarrow$ 'true', and for all $0 < \sigma' \leq \sigma$ we set $S[0, \sigma'] \leftarrow$ 'false'. To compute the entry $S[n', \sigma']$, we iterate over all $i = 1, 2, \ldots, n'$ and calculate the number z composed of the last i digits. If $z > \sigma'$ we stop, otherwise we check the entry $S[n' - i, \sigma' - z]$. This entry indicates whether the number $\sigma' - z$ can be formed using the first $n' - i$ digits. If so, we can form σ' by adding the last i digits. We then set $S[n', \sigma'] \leftarrow$ 'true' and stop the iteration. Otherwise, we continue with the next choice for i. If there are no choices left for i, we set $S[n', \sigma'] \leftarrow$ 'false', since it is not possible to form the sum.

Calculation order: Every entry $S[n', \sigma']$ depends only on entries for smaller values of n' and σ'. We can therefore sort the entries, for example, by increasing values of n', and for the same n' by increasing values of σ'.

Extracting the solution: The solution is stored in the entry $S[n, \sigma]$.

Running time: The number of entries is in $\Theta(n\sigma)$. For every entry $S[n', \sigma']$ we iterate over the values $i = 1, 2, \ldots, n'$. We also need to compute the number formed by the last i digits. We can derive this number in constant time from the number that was formed using the last $i - 1$ digits (or infer it directly if $i = 1$). Alternatively, we can calculate these values in advance for every possible combination of n' and i.

We have an overall running time of $\mathcal{O}(n^2\sigma)$. Since the running time depends on σ, it is *not* polynomial but only *pseudo-polynomial.* The running time is polynomial if it is known in advance that σ is bounded by a polynomial in n (i.e., $\sigma = \mathcal{O}(n^c)$ for a constant c).

b) We proceed exactly like while calculating the entries. In order to find all possible arrangements, we develop a recursive algorithm List, that gets three parameters n' (initially n), σ' (initially σ), and a character sequence *str* (initially empty). We iterate over all $i = 1, 2, \ldots, n'$ like we do when computing an entry and compute in each round the number z that consists of the last i digits. If $S[n'-i, \sigma'-z] =$ 'false', we do not have to do anything and continue with the next value of i. On the other hand, if $S[n' - i, \sigma' - z] =$ 'true', we recursively call $\textsc{List}(n'-i, \sigma'-z, \text{“}z + \text{”} \oplus str)$ (where “z + ” $\oplus$ *str* is the concatenation of the character sequences “z + ” and *str*). It is important to note that – in contrast to the computation of the table – we do not stop the loop if some i with $S[n' - i, \sigma' - z] =$ 'true' is found, but we consider all possibilities until either $i = n'$ or $z > \sigma'$. The recursion terminates if $n' = 0$ and the algorithm returns the character sequence *str* (without the last character that is always “+”).

The algorithm might not seem to be efficient at first sight, because there are apparently many recursive calls. However, one can notice that a recursive call only occurs if it leads to a possible arrangement. If we consider the recursion tree of the algorithm, we note that in each leaf a possible arrangement is reported. As n' is decreased by 1 in every step, the length of a path between two leaves in the recursion tree is at most $2n$. The algorithm needs only $\mathcal{O}(n)$ per node and in each leaf a solution is reported. Therefore, the overall runtime is in $\mathcal{O}(n^2M)$, where M is the number of reported arrangements.

One can even design the algorithm such that the time between reporting two character sequences is in $\mathcal{O}(n)$ by storing the largest i with $S[n' - i, \sigma' - z] =$ 'true' while computing each entry of the table. As n' is decreased the quicker, the larger i gets, i.e. the more time is necessary to iterate i, the running time reduces to $\mathcal{O}(nM)$. As every reported character sequence has a length in $\Theta(n)$ (exactly n digits and up to $n-1$ plus signs) and as there are M possible arrangements, the running time is linear in the size of the output!