Solutions to exercises of lecture 6

Exercise 6.1. Download the excel sheet subsetsum.xls from `http://www.win.tue.nl/~jnederlo/2MMD30/`. It was used to solve the instance

$$w = \{3, 20, 58, 90, 267, 493, 869, 961, 1000, 1153, 1246, 1598, 1766, 1922\}, t = 5842$$

of Subset Sum to find the solution $20, 58, 90, 869, 961, 1000, 1246, 1598$. Are there more solutions? If so, can you find one?

Solution: The excel sheet uses the following recurrence: for $i = 0, \ldots, 14$, $t = 0, \ldots, 5842$ let $A[i, j]$ be the number of subsets X from $\{1, \ldots, i\}$ such that $\sum_{e \in X} w_e = j$. We see that

$$A[i,j] = \begin{cases} 0 & \text{if } i = 0, j \neq 0, \quad (6.1) \\ A[i-1, j] + A[i-1, j-w_i] & \text{otherwise.} \quad (6.2) \end{cases}$$

Note that for convenience in excel we use a variant of the recurrence from the lecture notes. In the excel sheet we see the table entries $A[i, j]$ being computed. From the cell $(7842, 16)$ we see there are 5 solutions, and from $(7842, 15)$ we see there are 4 subsets X of $\{1, \ldots, 13\}$ with $\sum_{e \in X} w_e = t$, so 1 solution uses the integer 1922 and from the first 13 integers it will pick integers summing to $5842 - 1922$, so we continue to look at cell $(5920, 14)$ and see the unique solution containing 1922 does not contain $1766, 1598, 1246$ but contains 1153. Continuing like this we arrive at the solution $1922, 1766, 1246, 493, 90, 58$.

Exercise 6.2. How many integers in $\{1, \ldots, 100\}$ are not divisible by 2, 3 or 7?

Solution: Use inclusion exclusion. Let $U = \{1, \ldots, 100\}$, $P_1 = \{i \in U : i \text{ is not a multiple of } 2\}$, $P_2 = \{i \in U : i \text{ is not a multiple of } 2\}$, $P_3 = \{i \in U : i \text{ is not a multiple of } 2\}$. We need to compute $|P_1 \cap P_2 \cap P_3|$ which equals by the inclusion exclusion formula

$$|U| - |\overline{P_1}| - |\overline{P_2}| - |\overline{P_3}| + |\overline{P_1} \cap \overline{P_2}| + |\overline{P_2} \cap \overline{P_3}| + |\overline{P_1} \cap \overline{P_3}| - |\overline{P_1} \cap \overline{P_2} \cap \overline{P_3}|,$$

and these terms are more easily computed since, e.g., $|\overline{P_2} \cap \overline{P_3}|$ is the number integers that are simultaneously multiple of 3 and 7 (so equivalently, since 3 and 7 are co prime, a multiple of 21) which are $21, 42, 63$ and 84.

$$100 - 50 - 33 - 14 + 16 + 4 + 7 - 2 = 28$$

Exercise 6.3. At the 5th of December it is common in the Netherlands to buy presents for each other. To do this when there are n persons $p_1, \dots, p_n$ celebrating together, there are various processes to pick a random permutation $f : \{1, \dots, n\} \leftrightarrow \{1, \dots, n\}$. We call a permutation good if $f(i) \neq i$ for every i. Suppose $n = 5$, how many good permutations are there?

Solution: Use inclusion exclusion. Let U be all permutations, and for $i = 1, \dots, 5$ let P_i be the set of permutations f such that $f(i) \neq i$. We see that the number of good permutations (also called derangements) equals $|\cap_{i=1}^{5} P_i|$. We note that $|\cap_{i \in F} \overline{P_i}|$ only depends on $|F|$ by symmetry and

$$|\cap_{i \in F} \overline{P_i}| = \text{number of permutations where } f(i) = i \text{ for every } i \in F = (5 - |F|)!$$

since for all elements in F we have only one choice and for $i \notin F$ there are no restrictions so f can be any permutation restricted to $\{1, \dots, 5\} \setminus F$. By inclusion exclusion we see that $|\cap_{i=1}^{5} P_i|$ equals

$$\begin{aligned}
\sum_{F \subseteq \{1,\dots,5\}} (-1)^{|F|}(5 - |F|)! &= \sum_{i=0}^{5} (-1)^i \binom{5}{i} (5 - i)! \\
&= \binom{5}{0} 5! - \binom{5}{1} \cdot 4! + \binom{5}{2} 3! - \binom{5}{3} 2! + \binom{5}{4} 1! - \binom{5}{5} 0! \\
&= 120 - 120 + 60 - 20 + 5 - 1 = 44.
\end{aligned}$$

Exercise 6.4. The n'th Fibonacci number f_n is defined as follows: $f_1 = 1$,$f_2 = 1$ and for $n > 2$, $f_n = f_{n-1} + f_{n-2}$. What is the running time of the following algorithm to compute f_n?

Algorithm FIB2(n)
Output: f_n
Initiate a table F with $F[i] = -1$ for $i = 1, \dots, n$
return FIBREC(n).

Algorithm FIBREC(n)
Output: f_n
if $n = 1$ or $n = 2$ **then return** 1
if $F[n] = -1$ **then**
    $x \leftarrow$ FIBREC($n - 1$)+FIBREC($n - 2$)
    $F[n] \leftarrow x$
    return x.
else
    return $F[n]$.

Solution: $O(n)$ time. To see this, note that in the execution, the condition at Line 2 applies only once for every n. Intuitively, we could still look at the recursion tree of this algorithm, but it will be a very unbalanced tree of depth n where, if the left child is evaluated before the right child, the right child is a leaf since the condition at Line 2 will not apply.

Exercise 6.5. Let G be bipartite graph with parts A, B, $|A| = |B| = n$. Use inclusion exclusion to count the number of perfect matchings of G in $O^*(2^n)$ time. Can you do with polynomial space?

Solution: Use inclusion exclusion. A *pseudo-matching* of $G = (A \dot\cup B, E)$ is a set of edges $M \subseteq E$ such that for every $a \in A$ there exists exactly one $e \in M$ incident to a. Note that if $M \subseteq E$ is a pseudo-matching, and for every $b \in B$ there exists an edge $e \in M$ incident to b, then M is a perfect matching. Thus, if U is the set of all pseudo-matchings of G and for every $b \in B$, P_b is the set of all pseudo-matchings M such that there exists *at least*[1] one $e \in M$ containing b, then $|\cap_{b\in B} P_b|$ is the number of perfect matchings of G. By inclusion exclusion we have that

$$|\bigcap_{b\in B} P_b| = \sum_{F\subseteq B} (-1)^{|F|} |\bigcap_{b\in F} \overline{P_b}|.$$

Now, note that $|\bigcap_{b\in F} \overline{P_b}|$ is the number of pseudo-matchings that do contain any edge incident to a vertex of F, which is the number of pseudo-matchings in $G[A \cup B \setminus F, E]$. This number if easily computed in polynomial time: in a pseudo-matching every vertex in A needs to pick a neighbor in B but these choices are independent so we see that

$$|\bigcap_{b\in F} \overline{P_b}| = \prod_{a\in A} |N(a) \setminus F|,$$

so this can clearly be computed in polynomial time and thus the inclusion exclusion formula can be computed in $O^*(2^n)$ time.

Exercise 6.6. In the Weighted Steiner Tree problem we are given a graph $G = (V, E)$, a weight function $\omega : E \to \mathbb{N}$ and a set of terminals $T \subseteq V$. We need to find a connected tree (S, E') minimizing $\sum_{e\in E'} \omega(e)$ such that $T \subseteq S \subseteq V$. Solve this problem in time $O^*(2^{n-k})$, where $|V| = n$ and $|T| = k$.

Solution: Iterate over all relevant candidates for subset S, and for each such subset, compute a minimum spanning tree of $G[S]$. Return the minimum tree found. This gives a valid solution and also the optimum solution since once we fixed S the optimal way of connecting the vertex set is via a minimum spanning tree. The running time is $O^*(2^{n-k})$ since there are at most 2^{n-k} candidates for S and minimum spanning tree can be done in polynomial time.

Exercise 6.7. In the Set Partition and Set Cover problems we are given sets $A_1, \dots, A_m \subseteq U$ where $|U| = n$. In the Set Cover problem we need to find $X \subseteq \{1, \dots, m\}$ such that $\bigcup_{i\in X} A_i = U$. In the Set Partition problem we need to find $X \subseteq \{1, \dots, m\}$ such that $\bigcup_{i\in X} A_i = U$ and $A_i \cap A_j = \emptyset$ for every $i, j \in X$ with $i \neq j$. Solve both problems in $O^*(2^n)$ time. Can you also solve both in $O^*(2^n)$ time and polynomial space?[2] Note: $O^*(\cdot)$ suppresses factors polynomial in the *input size*, so $O^*(2^n m^{100})$ is also $O^*(2^n)$.

[1] Of course, the definition of perfect matching requires exactly one, but since we have only n edges, this is equivalent here.

[2] I do not expect you to reproduce the $O^*(2^n)$ time polynomial space algorithm for Set Partition.

Solution: Both problems can be done with both dynamic programming and inclusion exclusion. For dynamic programming, define table entries $A[i,Y]$ for $i = 1,\dots,m$ and $Y \subseteq U$ which contains the minimum size of a subset $X \subseteq \{1,\dots,i\}$ such that $\bigcup_{i\in X} A_i \subseteq X$ (and $\bigcup_{i\in X} A_i = X$ for Set Partition). We see for Set Cover that

$$A[i,Y] = \begin{cases} \infty & \text{if } i = 0, Y \neq \emptyset & (6.3)\\ 0 & \text{if } i = 0, Y = \emptyset, & (6.4)\\ \min\{A[i-1,Y], 1 + A[i-1, Y \setminus A_i]\} & \text{otherwise.} & (6.5)\end{cases}$$

And we see for Set Partition that

$$A[i,Y] = \begin{cases} \infty & \text{if } i = 0, Y \neq \emptyset & (6.6)\\ 0 & \text{if } i = 0, Y = \emptyset, & (6.7)\\ \min\{A[i-1,Y], 1 + A[i-1, Y \setminus A_i]\} & \text{if } A_i \subseteq Y & (6.8)\\ A[i-1,Y] & \text{otherwise.} & (6.9)\end{cases}$$

And $A[n,U]$ can be computed in $O^*(2^n)$ time in both cases.

For the inclusion-exclusion approach, set cover is quite a bit easier than set partition, so let us do that one first. We let $l = 1,\dots,m$ and will determine whether there exists X of size l. We apply inclusion exclusion with universe[3] $\{X \subseteq \{1,\dots,m\} : |X| = l\}$. For every $e \in U$ we define a property P_e which is $\{X \subseteq \{1,\dots,m\} : |X| = l \wedge e \in \bigcup_{i\in X} A_i\}$. We see that $|\bigcap_{e\in U} P_e| > 0$ if and only if there exists a set X as sought in the set cover problem of size l. Moreover, for $|\bigcap_{e\in U} \overline{P_e}|$ the choices of whether A_i is included or not are almost independent and in particular we see that

$$|\bigcap_{e\in F} \overline{P_e}| = \binom{|\{i \in \{1,\dots,m\} : A_i \cap F = \emptyset\}|}{l}.$$

Thus indeed the inclusion exclusion formula in this case can be evaluated in $O^*(2^n)$ time.

For set partition, we use a similar application, but we use that a tuple of sets form a set partition if and only if they form a set cover and there sizes add up to U: we use as universe in the inclusion exclusion formula the set $\{X \subseteq \{1,\dots,m\} : \sum_{i\in X} |A_i| = n\}$. For every $e \in U$ we define a property P_e which is $\{X \subseteq \{1,\dots,m\} : e \in \bigcup_{i\in X} A_i \wedge \sum_{i\in X} |A_i| = n\}$. We see that $|\bigcap_{e\in U} P_e| > 0$ if and only if there exists a set X as sought in the set partition problem of size l. Moreover, for $|\bigcap_{e\in U} \overline{P_e}|$ the choices of whether A_i is included or not are almost, and we can use dynamic programming: Define

$$A_F[i,j] = |\{X \subseteq \{1,\dots,i\} : \sum_{e\in X} |A_e| = n \wedge \text{ for every } e \in X : A_e \cap F = \emptyset\}|.$$

We see that $|\bigcap_{e\in F} \overline{P_e}| = A_F[n,n]$ and for fixed F, $A_F[n,n]$ can be computed in polynomial time using dynamic programming based on the recurrence

$$A_F[i,j] = \begin{cases} 0 & \text{if } i = 0, j \neq 0 & (6.10)\\ 1 & \text{if } i = 0, j = 0, & (6.11)\\ A_F[i-1,j] & \text{if } A_i \cap F \neq \emptyset & (6.12)\\ A_F[i-1,j] + A_F[i-1, j - |A_i|] & \text{otherwise.} & (6.13)\end{cases}$$

[3] which we won't denote by U this time since it already is used in the problem description

Thus indeed the inclusion exclusion formula in this case can be evaluated in $O^*(2^n)$ time.

Exercise 6.8.[Floyd Warshall] Suppose we are given a graph $G = (V, E)$ with for every edge $(u, v) \in E$ a distance $d(u, v)$. Let $V = \{1, \ldots, n\}$. The goal of this exercise if to compute for every pair $i, j \in V$ the shortest path from i to j in a total of $O(n^3)$ time. Show how to do this with dynamic programming. Specifically, let $d^{(k)}_{i,j}$ be the length of the shortest path from i to j for which all intermediate vertices are in the set $\{1, \ldots, k\}$ (see also p630 of the book 'Introduction to Algorithms' by Cormen et al.).

Solution:

$$d^{(k)}_{i,j} = \begin{cases} d(i,j) & \text{if } k = 0 \quad (6.14) \\ \min\{d^{(k-1)}_{i,j}, d^{(k-1)}_{i,k} + d^{(k-1)}_{k,j}\} & \text{if } k \geq 1. \quad (6.15) \end{cases}$$

And straightforward evaluation indeed results in an $O(n^3)$ time algorithm since the distances can be read off from $d^{(n)}_{i,j}$.

Exercise 6.9. First solve Knapsack in time $O(n \lg(v_{\max} n) v_{\max})$ where $v_{\max} = \max_i v_i$. How can you use is to solve Knapsack in time $O(n \lg(v_{\max} n) v_{\max}/S)$, if all values are divisible by S? Argue one could construct an approximation scheme for knapsack similar as in Subsection 6.1.1.[4]

Solution: For integers $i = 1, \ldots, n$ and $t = 1, \ldots, nv_{\max}$ define $A[i, t]$ to be $\min\{\sum_{e \in X} w_e : \sum_{e \in X} v_e \geq t\}$. We see that

$$A[i,j] = \begin{cases} 0 & \text{if } i = 0,\ j \leq 0, \quad (6.16) \\ \infty & \text{if } i = 0,\ j > 0, \quad (6.17) \\ \min\{A[i-1,j], w_i + A[i-1, j - v_i]\} & \text{otherwise.} \quad (6.18) \end{cases}$$

And the optimum value of the knapsack instance can be found by looking for the largest v such that $A[n, v] \leq W$. We can use this exactly as in Subsection 6.1.1 if all values are divisible by S by simply dividing all values by S and looking for the largest v such that $A[n, v] \leq W$ in the divided instance. By first rounding the values to the nearest multiple of S and then dividing by S this gives us an $O(n^3/\epsilon)$ approximation algorithm (e.g., if maximum value V is possible it returns a valid solution of value at least $(1-\epsilon)V$) for Knapsack.

[4] I do not expect you to reproduce the approximation scheme for Knapsack.