CS3230: Assignment for Week 8 Solutions

Due: Sunday, 27th Mar 2022, 11:59 pm SGT.

---

1. We can add a third parameter k that indicates that the chosen set of items is of size at most k:

$$m[i,j,k] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \text{ or } k = 0 \\ \max\{m[i-1, j-w_i, k-1] + v_i,\ m[i-1,j,k]\} & \text{if } w_i \le j \\ m[i-1,j,k] & \text{otherwise} \end{cases}$$

The second case says that if item i is chosen, then we maximize the value from choosing at most $k-1$ out of the first $i-1$ items subject to the weight constraint (which is reduced by w_i); otherwise, we simply maximize the value from choosing at most k out of the first $i-1$ items subject to the weight constraint (which is the same as before).

The running time is $O(nWR)$. To achieve this running time, we fill in the table in the order $m[0,*,*], m[1,*,*], m[2,*,*], \dots, m[n,*,*]$. Since the value of $m[i,j,k]$ depends only on $m[i-1,*,*]$, we have all the necessary values to compute $m[i,j,k]$. There are $O(nWR)$ entries, and computing each entry takes time $O(1)$. The final answer is then $m[n,W,R]$.

2. For $i = 0, 1, \dots, n$, let $m[i]$ denote the maximum sum of elements in $A[1\,..\,i]$ no two of which are adjacent. We start with $m[0] = 0$. When we consider $m[i]$, there are two choices: either we include $A[i]$, or we don't. If we include $A[i]$, we cannot include $A[i-1]$, and by a standard "cut-and-paste" argument, we must include an optimal solution from $A[1\,..\,i-2]$. Similarly, if we don't include $A[i]$, we must include an optimal solution from $A[1\,..\,i-1]$. It follows that

$$m[i] = \begin{cases} 0 & \text{if } i = 0 \\ \max\{0, A[1]\} & \text{if } i = 1 \\ \max\{m[i-2] + A[i],\ m[i-1]\} & \text{otherwise} \end{cases}$$

The running time is $O(n)$. Indeed, we can fill in the array m in the order $m[0], m[1], \dots, m[n]$; filling in each $m[i]$ only relies on $m[i-2]$ and $m[i-1]$ and takes $O(1)$ time. The final answer is then $m[n]$.

3. Note that if you assign helper i to task b_i for each $i = 1, 2, \ldots, n$, then by independence, the probability that all tasks are completed is $p_{1,b_1} p_{2,b_2} \cdot \ldots p_{n,b_n}$. So we want to find the maximum value that this product can attain.

   (a) For a set $S \subseteq \{1, 2, \ldots, n\}$, if S has size k, let $m[S]$ be the maximum probability[1] that we can complete all tasks in S by assigning them to helpers $1, 2, \ldots, k$. We have the following recurrence:

   $$m[S] = \begin{cases} 1 & \text{if } S = \varnothing \\ \max_{j \in S} \left(m[S \setminus \{j\}] \cdot p_{k,j} \right) & \text{otherwise} \end{cases}$$

   Here, if S is empty, the probability of completing all tasks in S is trivially 1. Otherwise, we try assigning to helper k each task $j \in S$, with success probability $p_{k,j}$. The success probability of the k tasks in S is then maximized when the assignment of the tasks in $S \setminus \{j\}$ to helpers $1, 2, \ldots, k-1$ is optimal; this optimal probability is $m[S \setminus \{j\}]$.

   The running time is $O(n \cdot 2^n)$. To achieve this, we fill in the table from smaller sets S to larger ones. There are $O(2^n)$ entries, and computing each entry takes time $O(n)$. The final answer is then $m[\{1, 2, \ldots, n\}]$.

   (b) If you check all possible assignments, this would take time $\Theta(n \cdot n!)$, which is higher than the time that the dynamic programming algorithm in part (a) takes.

[1] In order to implement sets as array indices, we can represent a set S as an n-bit integer, where bit i is set to 1 if and only if $i \in S$.