HW8

Seyed Armin Vakil Ghahani
PSU ID: 914017982
CSE-565 Fall 2018
Collaboration with: Sara Mahdizadeh Shahri, Soheil Khadirsharbiyani, Muhammad Talha Imran

October 24, 2018

Problem 1. Recursive algorithm

Solution

- a) The time complexity of calculating a_n is $T(n) = T(n-1) + T(n-2)$. We know that $T(n-1) = T(n-2) + T(n-3) \geq T(n-2)$. As a result:

  $$T(n) \geq 2 * T(n-2) = 2 * [T(n-3) + T(n-4)] \geq 2 * [2 * T(n-4)] = 2^2 * T(n-4)$$

  We can continue this operation until reaching $T(0)$ or $T(1)$. After that, we have this inequality: $T(n) \geq 2^{n/2} T(0)$ (W.L.O.G I suppose that n is even.).

  As a result, $T(n) \in \Omega(2^{n/2})$.

- b) With using the memoization, for each n, we just calculate the value of a_n once, and consequently, the running time would be $\theta(n)$.

  By using the dynamic programming, the running time would not change because for each n, we just calculate the value of a_n once, again.

- c) At first, I am going to prove $a_n \in \Omega(2^n)$ by induction. For $n = 0$ and $n = 1$, it is obvious based on the baseline values. Suppose that, for each $n \leq k$, $a_n \in \Omega(2^n)$. Now, we are going to prove this equation for $n = k + 1$.

  $$a_{k+1} = 2 * a_{k-1} + a_k, \forall n \leq k : a_n \in \Omega(2^n)$$

  $$a_k \leq c * 2^k, a_{k-1} \leq c * 2^{k-1}$$

  $$a_{k+1} \leq 2 * c * 2^{k-1} + c * 2^k \Rightarrow a_{k+1} \leq c * 2^{k+1} \Rightarrow a_{k+1} \in \Omega(2^{k+1})$$

  For the second step, we should do the same thing for proving $a_n \in O(2^n)$. We just have to change the $\leq$s to $\geq$. By proving these two, it is proven that $a_n \in \theta(2^n)$.

Problem 2. Knapsack

Solution

- a) We define $M[i, w]$, the maximum value that we can get if we use some of the first i items in such a way that their weights equal to w. If we cannot do this, we set the value of $M[i, w]$ to $-\infty$.
- b) Based on the definition of M, we cannot make the weight of $w > 0$ with zero items. So, the value of $M[0, w]$ for each $w > 0$ is $-\infty$. However, we can make the weight of zero without any items, and consequently, the value of $M[0, 0] = 0$.
- c)

$$M[i,w] = \left\{ \begin{array}{cc} M[i-1,w] & \text{if } w_i > w \\ M[i-1,w] & \text{if } M[i-1,w-w_i] = -\infty \\ max(M[i-1,w], v_i + M[i-1,w-w_i]) & \text{o.w.} \end{array} \right\}$$

- d) The running time of this problem is the same as the standard Knapsack problem because we just change the initial value of the $M[i, w]$ table, and add an if statement to the recurrence equation which do not change the time complexity of the algorithm. As a result, the time complexity of this algorithm would be $O(nW)$.
- e) The parts b and c are changed a little. However, part d is not changed.
- f) The difference between the standard knapsack and this problem is just in the definition of this problem. We know that in the standard knapsack, the weight of the knapsack could be 0, 1, 2, ..., W-1, W. So, if we want the weight of the knapsack to be w, we can check the value of $M[n, w]$. So, we can get the maximum of $M[n, 0], M[n, 1], M[n, 2], ..., M[n, W-1], M[n, W]$ to get the result of the standard knapsack.

Problem 3. Subset sum

Solution

- a) $M[0, 0] = true, \forall w > 0 : M[0, w] = false$

$$M[i,w] = \left\{ \begin{array}{cc} M[i-1,w] & \text{if } w_i > w \\ M[i-1,w] & \text{if } M[i-1,w-w_i] = false \\ true & \text{if } M[i-1,w-w_i] = true \end{array} \right\}$$

- b) In this case, we should define $M[i, W]$, the number of subsets $S = \{1, ..., i\}$ that $\sum_{j \in S} w_j = W$. So, for initialization the values of M, the value of $M[0, 0]$ should be one, and $\forall w > 0 : M[0, w] = 0$. The recurrence equation in this case would be like this: $M[0, 0] = true, \forall w > 0 : M[0, w] = false$

$$M[i,w] = \left\{ \begin{array}{cc} M[i-1,w] & \text{if } w_i > w \\ M[i-1,w] + M[i-1,w-w_i] & \text{o.w} \end{array} \right\}$$

The pseudo-code is as follows:

```
M[0,0] = 1;
for i = 1 to W
   M[0,i] = 0;
for i = 1 to n
   for j = 1 to W
      if j < w[i]
         M[i,j] = M[i-1,j];
      else
         M[i,j] = M[i-1,j] + M[i-1,j-w[i]];
return M[n,W]
```

- c) We are going to prove this problem by induction on n. For the first case, when $n = 0$ and $S = 0$, there is no way to create the value of $w > 0$, and consequently, the value of $M[0, w]$ for each $w > 0$ is zero. There is just one case that we do not choose anything from S and the sum would be zero. As a result, $M[0, 0]$ is equal to one.

  Now, suppose that every value of $M[k, i]$ for each $0 \leq i \leq W$ is calculated correctly, and we are going to calculate the values of $M[k+1, i]$ for each $0 \leq i \leq W$. According to the pseudo-code of this algorithm, the number of ways that we can create i is two ways:

  - Do not picking the item $k + 1$: In this case, the number of ways to create i is equal to $M[k, i]$, which is correctly calculated in previous step.
  - Picking the item $k + 1$: In this case, the number of ways to create i is equal to create $i - w_{k+1}$ from previous items that is $M[k, i - wk + 1]$, which is correctly evaluated based on the induction assumption.

  As a result, the value of $M[k + 1, i]$ for each $0 \leq i \leq W$ is going to calculate correctly with the values of array M in previous row.