

Homework 2A Solutions

1 4 and 7 Change

You need to make change for n cents, but only have coins of value 4 and 7. Design an algorithm which returns whether it is possible to make n cents with these coins.

Part A Let $T[i]$ represent a boolean value of whether it is possible to make i cents using 4 and 7 cent coins.

Part B Since $T[i]$ represents whether it is possible to make i cents with 4 or 7 cent coins, then $T[i] = (i \geq 4 \wedge T[i - 4]) \vee (i \geq 7 \wedge T[i - 7])$ and define our base case as $T[0] = \text{true}$ (it is possible to make 0 cents by using no coins).

This is correct as we know if it is possible to make the current value i minus 4 cents, we can add a 4 cent coin to that previous value. This relies on having those previous values filled in for the table, which our algorithm does.

Part C The pseudocode for this algorithm is below. Note that we are using short circuit evaluation to stop evaluation of an expression if we evaluate an expression that concludes a clause.

```

1 Function QuestionOne( $n$ ):
2   /* let  $T$  be an empty  $(n+1)$  table with default values of false */
3    $T \leftarrow []$ 
4    $T[0] \leftarrow \text{true}$ 
5   for  $i \in 1 \rightarrow n$  do
6     /* recurrence relation utilizing short circuit evaluation */
7      $T[i] = (i \geq 4 \wedge T[i - 4]) \vee (i \geq 7 \wedge T[i - 7])$ 
8   return  $T[n]$ 

```

Part D The runtime of this algorithm is $\mathcal{O}(n)$ as there is a single loop iterating from $1 \rightarrow n$ where each iteration performs a constant number of $\mathcal{O}(1)$ operations.

2 Minimum Coin Change

You are given a set of k coins each with a different value and a target amount n . Return the fewest number of coins that you need to make up the target amount. If it cannot be made, return -1 .

Part A Let $T[i]$ represent how many coins needed to make i cents with $T[i] = -1$ meaning that it is not possible to make i cents.

Part B Since we have a set of K coins, we know that $T[i]$ can be made by using any previous values (previous such that a coin from K can be added to get i) and adding 1. We minimize over all these options.

$$T[i] = \min_{c \in K: c \leq i \wedge T[i-c] \neq -1} T[i-c] + 1$$

We arrive at $T[i]$ by minimizing over all coins in K as long as the coin value is less than the target value ($c \leq i$) and it is possible to make $T[i-c]$ ($T[i-c] \neq -1$). Our base case is that $T[0] = 0$ since it takes 0 coins to make 0 cents. Values are initialized to -1 by default.

Part C The pseudocode for this algorithm is below.

```
1 Function QuestionTwo( $n, K$ ):  
   /* let  $T$  be an empty  $(n+1)$  table with default values of  $-1$  */  
2    $T \leftarrow []$   
3    $T[0] \leftarrow 0$   
4   for  $i \in 1 \rightarrow n$  do  
5       for  $c \in K$  do  
6            $m \leftarrow -1$   
7           if  $c \leq i \wedge T[i-c] \neq -1$  then  
8                $m \leftarrow \min(m, T[i-c] + 1)$   
9            $T[i] \leftarrow m$   
10  return  $T[n]$ 
```

Part D The runtime of this algorithm is $\mathcal{O}(nk)$ as we iterate over each value from $1 \rightarrow n$ and look through k coin values for each iteration.

3 Unloading for Christmas

Design an algorithm to unload packages from a ship efficiently where we can either remove packages one at a time (A, B, C) or by removing chunks of multiple packages (see original PDF for examples/detail).

Part A Let $T[i]$ represent the time that it takes to unload the first i packages.

Part B Since $T[i]$ is the time it takes to represent the first i packages, notice we can either naively unload a package or unload using chunks. The naive unloading strategy is $T[i - 1] + 1$ as it takes an additional unit of time to unload from the previously $i - 1$ unloaded packages.

In order to unload chunks, we must check each chunk $\in L$ and determine and if a substring ending in L is equal to that chunk. If a chunk fits, then it takes $T[i - c - 1] + 1$ time where c is the length of a chunk. To check the substring, we can simply check if $U[i - c : i]$ (inclusive) is equal to that chunk from L .

$$T[i] = \min \left(T[i - 1] + 1, \min_{\substack{c \in L \\ |c| < i \\ U[i - |c| : i] = c}} (T[i - |c| - 1] + 1) \right)$$

The base cases are $T[0] = 0$ since it takes no time to remove the first 0 packages. This recurrence is correct as the only options to remove packages is to remove individually or to remove entire chunks at a time. In order to remove a chunk, we find a chunk such that we won't be unloading more packages than there are to unload and also the chunk being removed is actually in the substring from $i - |c|$ to i (inclusive) where we let $|c|$ be notation for string size.

The pseudocode and runtime are on the next page.

Part C The pseudocode for this algorithm is below.

```

1 Function QuestionThree( $U, L$ ):
   | /* let  $T$  be an empty  $(n+1)$  table */
2   |  $T \leftarrow []$ 
   | /* takes 0 time to unload 0 packages */
3   |  $T[0] \leftarrow 0$ 
4   | for  $i \in 1 \rightarrow n$  do
5   |   |  $T[i] \leftarrow T[i-1] + 1$ 
6   |   | for  $c \in L$  do
7   |   |   | if  $|c| < i \wedge U[i - |c| : i]$  then
8   |   |   |   |  $T[i] \leftarrow \min(T[i], T[i - |c| - 1] + 1)$ 
9   | return  $T[n]$ 

```

Part D Let n be the length of the string U and k be the size of the set L , then the runtime for this algorithm is $\mathcal{O}(nk)$ as we iterate from $1 \rightarrow n$ and search through each $c \in L$ which is $\mathcal{O}(n) \times \mathcal{O}(k) = \mathcal{O}(nk)$.