Solutions to Homework 2 Practice Problem

Note: *these solutions serve as examples for the level of detail we expect in your DP and D&C solutions. As with HW1, in some cases we have added optional commentary to explain the intuition behind the solution - this content will be highlighted in the narrative.*

[DPV] Problem 6.17 – Making change I (unlimited supply)

(a) Define the entries of your table in words. E.g., $T(i)$ or $T(i, j)$ is ....

Our subproblem considers making change for ever increasing amounts until we get to the value v: for all $1 \leq w \leq v$, if can make change w with coins of denominations $x_1, \ldots, x_n$ we let $T(w) = \texttt{TRUE}$, otherwise $T(w) = \texttt{FALSE}$.

Let $T(w)$ = TRUE or FALSE if we can make change w using coins $x_1, \ldots, x_n$

(b) State recurrence for entries of table in terms of smaller subproblems.

Informally, the recurrence does the following: for each denomination x_i, if $T(w - x_i)$ is TRUE *for at least one value of i, set $T(w)$ to be* TRUE. *Else set it* FALSE.

Base Case:

$$T(0) = \texttt{TRUE}$$

Recurrence:

$$T(w) = \text{False} \bigvee_i T(w - x_i) \text{ where } x_i \leq w \ \forall \ 1 \leq i \leq n$$

where $\vee$ is the OR logical operator.

(c) Write pseudocode for your algorithm to solve this problem.

```
$T(0) = \texttt{TRUE}$
for $w = 1$ to $v$ do
   $T(w) = \texttt{FALSE}$
   for $i = 1$ to $n$ do
      if $x_i \leq w$ then
         $T(w) = T(w) \vee T(w - x_i)$
return $T(v)$
```

(d) Analyze the running time of your algorithm.

The nested loops are $O(v)$ and $O(n)$, with updates in order $O(1)$, this has running time $O(nv)$ which dominates all the constant time operations.

[DPV] Problem 6.18 – Making change II

(a) Define the entries of your table in words. E.g., $T(i)$ or $T(i, j)$ is ....

This problem is very similar to the knapsack problem without repetition that we saw in class. First of all, let's identify the subproblems. Since each denomination is used at most once, consider the situation for x_n. There are two cases, either

- We do not use x_n then we need to use a subset of $x_1, \ldots, x_{n-1}$ to form value v;
- We use x_n then we need to use a subset of $x_1, \ldots, x_{n-1}$ to form value $v - x_n$. Note this case is only possible if $x_n \leq v$.

If either of the two cases is `TRUE`*, then the answer for the original problem is* `TRUE`*, otherwise it is* `FALSE`*. These two subproblems can depend further on some subproblems defined in the same way recursively, namely, a subproblem considers a prefix of the denominations and some value.*

We define a $n \times v$ sized table D as:

$$D(i,j) = \{\texttt{TRUE} \text{ or } \texttt{FALSE} \text{ where there is a subset of the coins of denominations } x_1, \ldots, x_i \text{ to form the value } j.\}$$

Our final answer is stored in the entry $D(n, v)$.

(b) State recurrence for entries of table in terms of smaller subproblems.

The base cases are

$$D(i,0) = \texttt{TRUE} \text{ for all } 0 \leq i \leq n$$
$$D(0,j) = \texttt{FALSE} \text{ for all } 1 \leq j \leq v$$

Analogous to the above scenario with denomiation x_n we have the following recurrence relation for $D(i, j)$: (Recall, $\vee$ denotes Boolean OR)

For $1 \leq i \leq n$ and $1 \leq j \leq v$ then we have:

$$D(i,j) = \begin{cases} D(i-1,j) \vee D(i-1, j-x_i) & \text{if } x_i \leq j \\ D(i-1,j) & \text{if } x_i > j. \end{cases}$$

(c) Write pseudocode for your algorithm to solve this problem.

The algorithm for filling in the table is the following.

```
for $i = 0$ to $n$ do
  $D(i, 0) =$ TRUE
for $j = 1$ to $v$ do
  D$(0, j) =$ FALSE
for $i = 1$ to $n$ do
  for $j = 1$ to $v$ do
    if $x_i \leq j$ then
      $D(i, j) \leftarrow D(i-1, j) \vee D(i-1, j-x_i)$
    else
      $D(i, j) \leftarrow D(i-1, j)$
return $D(n, v)$
```

(d) Analyze the running time of your algorithm.

Each entry takes $O(1)$ time to compute, and there are $O(nv)$ entries. Hence, the total running time is $O(nv)$.

[DPV] 4.11 (length of shortest cycle on a graph)

Solution: As presented, we are given a ***directed graph*** with ***positive edge weights***. These conditions assure that there is no negative-weight cycle within the graph. How do we find the length of the shortest cycle? Note that the shortest cycle visiting vertices u and v has length $d(u,v) + d(v,u)$, hence the answer we are looking for is

$$\min_{u,v}(d(u,v) + d(v,u))$$

Notice that if the quantity above is equal to infinity it means the graph is acyclic, and we report that.

Our approach: run `Floyd-Warshall` on the given graph to get an array $\{d(u,v)\}_{u,v \in V}$ of the distances between any pair of points. It is assumed that at the beginning any (u,v) pair which is not represented by an edge $[(u,v) \notin E]$ is set $d(u,v) = \infty$. Now, inspect the diagonal in the array and our shortest path is the $\min(d(v,v))$ across all vertex $v \in V$

To see why this algorithm is correct: note first that every cycle visiting vertices u and v can be broke into two directed paths: from u to v and from v back to u. Each such path achieves its minimum length at $d(u,v)$ and $d(v,u)$ respectively, hence the cycle of minimal length is given by this sum.

`Floyd-Warshall` takes $O(|V|^3)$ and the minimum can be found in $O(|V|)$, leading to an $O(|V|^3)$ runtime.

[DPV] 4.21 (Currency trading)

Solution: We are presented with a directed graph problem, so hopefully we can use a known algorithm as the basis for our solution. The key is to recognize two things: (a) the currency calculation is a product (b) we want to maximize that product. We solve for the former by converting the exchange rates to logs (recalling that $\log(a) + \log(b) = \log(a \times b)$), and solve for the latter by using the negative of the log for our edge weight (flipping maximization to minimization).

So, for part (a), we create a graph where the vertices represent countries, and the edges between them have a weight $w_{i,j} = -\log r_{i,j}$. We then run Bellman-Ford from vertex s to vertex t, and the minimal weight path represents the most advantageous sequence of currency trades. This takes $O(|V||E|)$ time.

Why does this work? By converting the exchange rates to logs and then negating that value the weight of the shortest path (the sum of the negated log values) will represent the series of exchanges which maximizes the product of the exchange rates. Note that since edge weights are negative we cannot use Dijkstra's.

For part (b), we run one more iteration of Bellman-Ford, and if any of the distances (weights) change, we have detected a negative cycle – such a cycle represents the trading anomaly which allows for infinite profit.

[DPV] Problem 2.1 – Practice Multiplication

The product is 28,830. See ***2.1 Practice Multiplication.pdf*** for the value calculated at each level of the recursive stack.

[DPV] Problem 2.5 – Recurrence

Solution:
(a) $T(n) = 2T(n/3) + 1 = O(n^{\log_3 2})$ by the Master theorem

(b) $T(n) = 5T(n/4) + n = O(n^{\log_4 5})$ by the Master theorem

(c) $T(n) = 7T(n/7) + n = O(n \log_7 n) = O(n \log n)$ by the Master theorem

(d) $T(n) = 9T(n/3) + n^2 = O(n^2 \log_3 n) = O(n^2 \log n)$ by the Master theorem

(e) $T(n) = 8T(n/2) + n^3 = O(n^3 \log_2 n) = O(n^3 \log n)$ by the Master theorem

(f) $T(n) = 49T(n/25) + n^{3/2} \log n = O(n^{3/2} \log n)$
Hint: the contribution of level i of the recursion is $(\frac{49}{125})^i O(n^{3/2} \log n)$.

(g) $T(n) = T(n-1) + 2 = O(n)$

(h)

$$T(n) = T(n-1) + n^c = \sum_{i=1}^{n} i^c + T(0) = O(n^{c+1})$$

(i)

$$T(n) = T(n-1) + c^n = \sum_{i=1}^{n} c^i + T(0) = O(c^n)$$

(j)

$$T(n) = 2T(n-1) + 1 = \sum_{i=0}^{n-1} 2^i + 2^n T(0) = O(2^n)$$

(k)

$$T(n) = T(\sqrt{n}) + 1 = \sum_{i=0}^{k} 1 + T(b)$$

where k is an integer such that $n^{\frac{1}{2^k}}$ is a small constant b (the size of the base case). This implies that $k = O(\log \log n)$ and $T(n) = O(\log \log n)$.

[DPV] Problem 2.7 – Roots of unity

Solution:

For the sum, use the geometric series equality to get

$$1 + \omega + \omega^2 + \cdots + \omega^{n-1} = \frac{\omega^n - 1}{\omega - 1} = 0.$$

For the product, since $1 + 2 + \cdots + (n-1) = \frac{(n-1)n}{2}$ we get

$$1\omega\omega^2 \ldots \omega^{n-1} = \omega^{\frac{(n-1)n}{2}}$$

which equals 1 if n is odd and $\omega^{\frac{n}{2}} = -1$ for n even (remember that $\omega = e^{\frac{2\pi i}{n}}$).

[DPV] Problem 2.14 – Erase Duplicates
Solution:

This is a straight forward application of the MergeSort algorithm.

STEP1: Sort the given list. The running time of this step is $O(n\log(n))$.

STEP2: For $i = 1$ to $n - 1$: if $a_i = a_{i+1}$, delete a_i from the list. Output the final list. This list has no repeated terms. The running time of this step is $O(n)$.

Why the algorithm is correct: the list after step 1 is sorted, so if there are repeated terms of a_j for some j, they are all consecutive. STEP2 guarantees a comparison of all elements with both of its neighbors and elimination of one of the copies every time we found one. Hence no copies survived after STEP2 is completed.

The running time is $O(n\log(n) + n) = O(n\log(n))$.

Alternatively, you can modify the Merge subroutine to eliminate copies at the same time that the merging is done. As this change adds constant work at each level to test for and remove duplicates, this additional step does not impact the overall run time.)