CS 218, Spring 2017 Homework 6
Abhishek Kumar SRIVASTAVA May 24, 2017
Student ID: 861307778

I certify that this submission represents my own original work

---

Solution 1. . **Given :** Set of n objects and have m sequence of operations to perform consists of MAKE-SET, FIND-SET and LINK operations. LINK operation occurs before every FIND-SET.

From their algorithms we know that time complexity of MAKE-SET is $O(1)$, time complexity of LINK is also $O(1)$ and time complexity of FIND-SET is $O(h)$ where h is height of node in the SET tree. But since we are using both path compression and union by rank heuristics, after first FIND-SET operation on a node, root node will become its parent and also of all the intermediate nodes which are in the path from searched node to the root node and thus further operations will be of $O(1)$ for all these nodes.

Using accounting method for the amortized analysis the charging scheme will be:

1. Charge \$2 for MAKE-SET operation:
   - Pay \$1 for performing MAKE-SET operation for that node.
   - Reserve \$1 for that node. This will be used either when FIND-SET operation is performed on this node to change its parent from current node to the root node or if it is an intermediate node in the path when FIND-SET operation is performed on different node.
2. Charge \$1 for LINK operation. This will be paid for linking two different nodes using union by rank heuristics.
3. Charge \$1 for FIND-SET operation. This will be paid to return the root node of the SET tree when the node is at level 1. While traversing the path from node to the root node and changing its parent to root node will be paid using the reserve from MAKE-SET operation and it will bring it to level 1. Thus credit invariant holds since credit never goes to negative.

So total number of operations performed are m and lets assume n MAKE-SET operations are performed, p LINK operations are performed and $m-n-p$ FIND-SET operations are performed, thus total amount charged will be $\$2n + \$p + \$(m-n-p) = \$m + \$n$. So the time complexity will be T(n) = O(m) + O(n) but since m > n. Thus,

$$\boxed{\mathbf{T(n) = O(m)}}$$

Since above amortized analysis does not depend on the rank of nodes and how they are linked but is dependent upon path compression since we are able to bring the nodes at level 1 and don't have to pay again and again the $\$h$ amount each time. Thus the time complexity will be the same as above even if we do not perform union by rank heuristics and only use path compression heuristics. This time complexity will change in case if path compression heuristics is not used.

Solution 2. Given : Coins are minted with denominations of $\{d_1, ..., d_k\}$ units and we have to devise algorithm to change n units using minimum number of coins.

Consider the following pseudocode for the greedy algorithm. Sorting the list of denominations in decreasing order of its values since we want to minimize the number of coins so taking the coins of highest amount is a good greedy choice. Sorting the list will takes $\mathbf{O(k \cdot logk)}$ time assuming it has k number of denominations. The loop is executed maximum of k times each time doing constant number of computation *i.e* dividing the amount by denomination value thus has time complexity of $\mathbf{O(n)}$. Therefore the time complexity for the greedy algorithm is:

$$\boxed{\mathbf{T(n) = O(k\ logk)}} \tag{1}$$

Algorithm 1

```
function MinimumCoins(Amount, Denominations[ ])
    count = 0                      ▷ Variable to store the final count of selected denominations
    sort(Denominations[ ])         ▷ Sorting Denominations in decreasing order
    num = 0                        ▷ Stores number of coins for respective denomination
    while Amount > 0 do            ▷ Repeat till all amount is converted
        if Denominations[i] ≤ Amount then
            num = Amount / Denominations[i]          ▷ Take floor value
            count += num                             ▷ Adding number of coins for each denomination
            Amount = Amount - (num · Denominations[i])   ▷ Left Amount to be exchanged
        end if
    end while
return count
end function
```

Greedy choice property:

The greedy choice in this problem is picking the denomination value with highest value. We can get the maximum number of coins which can be exchanged for this denomination by dividing the total amount by the denomination value. Since we are exchanging maximum number of coins for a denomination, the amount left to be exchange will be the remainder of amount which cannot be divided. And this can be repeated with the next highest denomination value.

Let d_1 be the highest denomination value & d_2 some other denomination value less than d_1. By our greedy property picking d_1 will give optimal solution. We have to show that by picking d_2 will not lead to the optimal solution which is least number of coins. Let n be the amount to be exchanges and assuming using d_1 and d_2 we can exchange the n amount. The number of coins when d_1 is picked first and d_2 is picked will be $\left\lfloor \frac{n}{d_1} \right\rfloor + \left\lfloor \frac{n'}{d_2} \right\rfloor$. n' is left amount $n - (\left\lfloor \frac{n}{d_1} \right\rfloor \cdot d_1)$ after d_1 is exchanged. When d_2 is picked first and d_1 is picked later the total number of coins will be $\left\lfloor \frac{n}{d_2} \right\rfloor + \left\lfloor \frac{n''}{d_1} \right\rfloor$. Since n'' will be less than d_2 value therefore n''/d_1 will give 0. If choosing d_2 leads to better result we have to prove that:

$$\left\lfloor \frac{n}{d_1} \right\rfloor + \left\lfloor \frac{n'}{d_2} \right\rfloor > \left\lfloor \frac{n}{d_2} \right\rfloor$$

$$\left\lfloor \frac{n}{d_1} \right\rfloor + \left\lfloor \frac{n}{d_2} \right\rfloor - (\left\lfloor \frac{n}{d_1} \right\rfloor \cdot \frac{d_1}{d_2}) > \left\lfloor \frac{n}{d_2} \right\rfloor$$

$$\left\lfloor \frac{n}{d_1} \right\rfloor > \left\lfloor \frac{n}{d_1} \right\rfloor \cdot \frac{d_1}{d_2}$$

$$d_2 > d_1$$

Which is contradiction to our assumption. That's why our greedy assumption is correct.

Optimal substructure property:
We have proved that the optimal solution contains the greedy choice. Suppose U is an optimal solution for the problem. U' be the solution for the subproblem formed after exclusion of the greedy choice. If U' is not optimal, then there exists some U'' which is the optimal solution of the subproblem. In that case we can include the greedy choice in U'' and form a better solution than U. But as U is the optimal solution this is a contradiction. Therefore U' is optimal which proves the optimal substructure.

2. Lets say we were trying to find minimum number of coins in the change for 12 (or any multiple of 6). The greedy algorithm would give us total of 3 coins (1 coin of 10 and 2 coins of 1). But the optimal solution has 2 coins of 6. Therefore the greedy algorithm does not always give the minimum number of coins in a country whose denominations are $\{1, 6, 10\}$.

3. Consider the following pseudocode, where we A is the amount (in this problem $\mathbf{A} = \mathbf{n}$) and D is array of denominations (**where length of D is k**). The outer loop runs n times and during each of these iterations the inner loop runs k times. Therefore the running time of the algorithm can be given by

$$\boxed{\mathbf{T(n) = O(nk)}}$$

Algorithm 2

```
function MinimumNumberOfCoins(A, D)
    minimumCoin                      ▷ Array of size n to store the solutions of sub-problems
    for a do ← 1 to A
        minimumCoin[a] ← ∞
        for d do in D
            if d<a && minimumCoins[a-d] + 1 < minimumCoins[a] then
                minimumCoins[a] ← minimumCoins[a-d] + 1
            end if
        end for
    end for
return minimumCoins[A]
end function
```