cs 5800 - hw4

Yijing Xiao

October 2021

1 Exercise 16.1-3

1.1 Selecting the activity with the least duration from activity times $\{(0,5),(4,7),(6,10)\}$ will result in selecting (4, 7) and none other. Clearly, this is worst than the optimal solution obtained by selecting the two activities (0,5) and (6,10).

1.2 The activity with the minimum overlap in activity times $\{(0,3),(0,3),(0,5),(0,5),(4,7),(6,9),(8,11),(10,13),(10,13),(10,13),(12,13),(12,13)\}$ is (6,9) in the top row. However, selecting this activity eliminates the possibility of selecting the optimal solution of (0,3), (4,7), (8,11) and (12,13).

1.3 Selecting the activity with earliest starting time in activity time $\{(0,6),(3,4),(5,8)\}$ will oly have a single activity (0,6), whereas the optimal solution would be to pick the two other activities (3,4) and (5,8).

2 Exercise 16.2-3

Assume that the knapsack can carry at most W pounds, and there has n items. The weights of items put in the array W[n] and the values of items put in the V[n], the result will put in the X[n]. .

Pseudocode 1 Solving the 0-1 knapsack problem by using the greedy strategy

Input: The capacity of knapsack W, weights W[n] and value V[n]
Output: The result array X[n]

```
1: function KNAPSACK(W, W[n], V[n])
2:     ▷ Suppose that W[n] is sorted by increasing weight and V[n] is the same as their order when sorted by decreasing value.
3:     Initialize the result array X[n] ← 0
4:     i ← 0
5:     temp ← W
6:     while X[n] == 0 and i < n do
7:         if W[i] ≤ temp then
8:             X[i] ← V[i]                          ▷ put the ith item into the knapsack
9:             temp ← temp − W[i]
10:            i++
11:        end if
12:    end while
13:    return X[n]
14: end function
```

Since the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing value, we could just choose the items from the biggest to small until there is no more space. Since the sorting takes $O(n \log_n)$ and iterating through the list takes $O(n)$, the running time would be $O(n \log_n)$.

Proof for correctness:
Suppose that best solution doesn't not include the greedy choice: $SOL = (r_1, r_2, r_3, ...)$ quantities chosen of these items, and that r_1 is not the max quantity available (of max quality item), $r_1 < w_1$. Create a new solution SOL' from SOL by taking more of item 1 and less of the others.

$$e = min(r_2, w_1 - r_1), SOL' = (r_1 + e, r_2 - e, r_3, r_4, ...)$$

$$Value(SOL') - Value(SOL) = e(q_1 - q_2) > 0$$

which means SOL' is better than SOL
Contradicting that SOL is the best solution, therefore, best solution should include greedy choice.

3 Exercise 16.2 -4

Professor should check whether he can make it to the next refilling location without stopping at this one: If he can, skip this one otherwise fill up.

Professor doesn't need to know how much water he has or how far the next refilling location is because at each fill up, he can determine which is the next location at which he will need to stop. Suppose the distances between refilling locations are in array D[n]: .

Pseudocode 2 determine which water stops the professor should make

Input: the distances between refilling locations D[n]
Output: The water stop R[n]

```
1:  function MinimumStop(D[n])
2:      distance ← 0
3:      for d in D[n] do
4:          if distance + d > m then
5:              ▷ we need to mark current location to refilling stop R[n]. The number of water stops increase by one
6:              distance ← 0
7:          else
8:              distance ← distance + d
9:          end if
10:     end for
11:     return R[n]
12: end function
```

By following the algorithm shown above, the professor will not make a water stop if the distance from last refill to the upcoming refill is less than m (he can skate m miles before running out of water). He will stop for refill only when he cannot reach the next point at any point.

Proof for correctness:
Let SOL be any optimal solution which has professor stop at location $o_1, o_2, ..., o_k$. Let k denote the furthest stopping point we can reach from the starting point. Then we may replace o_1 by k to create a modified solution SOL', and $o_2 - o_1 < o_2 - k$.

In other words, we can actually make it to the position without running out of waster. Since SOL' has the same number of refilling stops, we conclude that k is contained in some optimal solution. Therefore, our algorithm works,

The running time is $O(n)$ where n is the number of refilling spots.

4 Exercise 16.2-5

We could use greedy strategy to solve this problem.

1. Let x be the smallest element in a set S.
2. put all elements between [x, x+1] in a list.
3. Repeat.

Let S be the given set, y be the 0 and A be the result list. .

```
1: for i = 1 to n do
2:     if x_i > y then
3:         A ← A ∪ x_i, x_i + 1
4:         y ← x_i + 1
5:     end if
6: end for
7: return A
```

Proof for correctness:
For any intervals, let S(I) denote its left endpoint and F(I) denote its right endpoint. Let $SOL = \{o_1, o_2, ..., o_p\}$ be an optimal solution to the problem and $SOL' = \{r_1, r_2, ..., r_q\}$ be the solution produced by our algorithm described above.
For the base case intervals o_1 and r_1, both of these intervals contains x_1, therefore, $F(o_1) \leq x_1+1$. However, our greedy algorithm picks r_1 such as $S(r_1) = x_1$ and $F(r_1) = x_1 + 1$. That is, we conclude that $F(r_1) \geq F(o_1)$. Suppose that $F(r_i) \geq F(o_i)$ where $1 \leq i \leq k \leq p$ for all i, our greedy algorithm will choose $[x, x + 1]$ as the next interval and therefore $r_{k+1} = [x, x + 1]$.
According to the induction hypothesis, $F(r_k) \geq F(o_k)$ and therefore $F(o_k) < x$. This means that $S(o_{k+1}) \leq x$ and therefore $F(o_{k+1}) \leq x + 1 = F(r_{k+1})$. Clearly, our greedy algorithm is optimal solution.

The running time is $O(n)$.

5 Exercise 16.2-6

According to the weighted median (Book Problem 9.2), for n distinct elements $x_1, x_2, ..., x_n$ with positive weights $w_1, w_2, ..., w_n$ such that $\sum_{=1}^{n} kw_i = 1$, the weighted (lower) median is the elements x_k satisfying $\sum_{x_i<x_k} w_i < \frac{1}{2}$ and $\sum_{x_i>x_k} w_i \geq \frac{1}{2}$. We can find the median item in terms of value per unit weight in linear time and we can know if we can fit all items that are at least that valuable in the knapsack or not in linear time.

Use a linear time median algorithm to calculate the median m of $\frac{v_i}{w_i}$ ratios. Next, partition the items into three sets:

$$G_1 = \left\{ i : \frac{v_i}{w_i} > m \right\}, G_2 = \left\{ i : \frac{v_i}{w_i} = m \right\}, and\ G_3 = \left\{ i : \frac{v_i}{w_i} < m \right\};$$

this step takes linear time.

Compute $w_1 = \sum_{i \in G_1} w_i, w_2 = \sum_{i \in G_2} w_i, and\ w_3 = \sum_{i \in G_3} w_i$ the total weight of the items in set $G_1, G_2, and\ G_3$ respectively. .

Input: The median of set m, knapsack capacity W
Output: A set of items fitting in the knapsack maximizing the total profit

1: **if** $w_1 > W$ **then**
2: recurse on the set of items G_1 and knapsack capacity W
3: **else**
4: take all items in set G_1 and take as much of the items in set G_2 as will fit the remaining capacity $W - w_1$
5: **end if**
6: **if** $w_1 + w_2 \geq W$ **then** ▷ no capacity left after taking
7: we are done
8: **else**
9: after taking all the items in set G_1 and G_2, recurse on the set of items G_3 and knapsack capacity $W - w_1 - w_2$
10: **end if**

Let $T(n)$ denote the runtime of the algorithm. Since we can solve the problem when there is only one item in constant time, the recursion for the runtime is $T(n) = T(\frac{n}{2}) + cn$, which gives runtime of $O(n)$.

6 (Extra) Exercise 16.3-3

a = 0000000, b = 0000001, c = 000001, d = 00001, e = 0001, f = 001, g = 01, h = 1

7 Problem 16-1

7.1

Let the amount that need to be changing is A and we at least use $p_1 < p_2 < ... < p_n$ value;s coin (the type if coin's value is n). Let F(A) denote to be the fewest coins number when the total amount is A, such as F(0) = 0 when A = 0.

If we want to use greedy strategy to solve this problem, we could choose a greedy choice. We could check how many biggest value's coins we have to use (assume that we need m coins with p_k value), and check how many biggest value's coins we have to use in situation of $A - m \times p_k \geq p_j$. Then, repeat this process until the amount of remaining change drops to 0.

Pseudocode 3 making change for n cents using the fewest number of coins

Input: the change we have, the amount cents that need to be changing
Output: How many change coins and the changing coins' value

```
1: function GetChange(coins[n], amount)
2:     sort the coins array by increasing order
3:     i ← coins.length − 1                          ▷ we will traverse from biggest value
4:     while i ≥ 0 do
5:         if amount ≥ coins[i] then
6:             n ← amount/coins[i]
7:             change ← n + coins[i]
8:             amount ← amount − change
9:             result[i] ← [n, coins[i]]
10:        end if
11:    end while
12:    return result[n]
13: end function
```

Proof for correctness:
Assume that our greedy solution SOL is not optimal solution, let SOL' be an optimal solution. $SOL' = \{o_1, o_2, o_3, o_4\}$ **which is using** o_1 **quarters,** o_2 **dimes,** o_3 **nickels and** o_4 **pennies.**

Let C_1 **denote to be the largest number of quarters that can be used to make change for n cents,** n_1 **is the remaining after using** C_1 **quarters.** C_2 **is the largest number of dimes that can be used,** n_2 **is the amount remaining after using** C_2 **dimes.** C_3 **is the largest number of nickels that can be used, and** C_4 **is the largest number of pennies that can be used.**

$$C_1 = \frac{n}{25},\ n_1 = n - 25 \times C_1;\ C_2 = \frac{n_1}{10},\ n_2 = n_1 - 10 \times C_2;$$

$$C_3 = \frac{n_2}{5};\ c_4 = n_2 - 5 \times C_3$$

If $o_4 \geq 5$ **we can replace every 5 pennies with 1 nickels, reducing the number of coins used, so** $o_4 < 5$**.**
If $o_2 \geq 3$ **we can replace every 3 dimes with 1 quarter and 1 nickel, reducing the number of coins used, so** $o_2 < 3$**.**
If $o_3 \geq 2$ **we can replace every 2 nickels with 1 dime, reducing the number of coins used, so** $o_3 < 2$**.**
If $o_2 \times dimes + o_3 \times nickels + o_4 \times pennies \geq 25$ **we can replace 25 cents with 1 quarter, reducing the number of coins used. In other words, if suppose that** $10 \times o_2 + 5 \times o_3 + o_4 \geq 25$**, it only happen if** $o_2 = 2$ **and** $o_3 = 1$**. But in this case we can replace the 2 dimes and 1 nickel with 1 quarter. So** $10 \times o_2 + 5 \times o_3 + o_4 < 25$**.**

Since $n = 25 \times o_1 + 10 \times o_2 + 5 \times o_3 + o_4$**, we have just shown that** $c_1 = \frac{n}{25} = o_1$**. From the previous facts that** $o_3 < 2$ **and** $o_4 < 5$**, we have that** $5o_3 + o_4 < 10$**, so greedy chooses** $c_2 = o_2$ **dimes and then** $c_3 = o_3$ **nickels and** $c_4 = o_4$ **pennies, so our greedy solution is optimal solution.**

7.2

For $i = 0, 1, ..., k$, let o_i be the number of coins of denomination c^i used in an optimal solution to the problem of making change for n cents. Then for $i = 0, 1, ..., k-1$, we have $o_i < c$ that means if we use c^i to making changes then we at most could use the number of $c - 1$

To show that the greedy solution is optimal, we have to show that any non-greedy solution is not optimal. Assume that there is a non-greedy solution $SOL = m_0 \times c^0 + m_1 \times c^1 + ... + m_k \times c^k = \sum_{i=0}^{i} m_i \times c^i$, and a greedy solution $SOL' = n_0 \times c^0 + n_1 \times c^1 + ... + n_k \times c^k = \sum_{i=0}^{i} n_i \times c^i$. Since the greedy strategy as much as possible takes the biggest value of coin to make change, $SOL' = \sum_{i=0}^{i} n_i \times c^i \geq c^i$, whereas the non-greedy solution $SOL = \sum_{i=0}^{i} m_i \times c^i = n$ where n is the total amount that need to be changing.
Since we have $m_i \leq c - 1$ for $i = 0, 1, .., k$, then

$$SOL = \sum_{i=0}^{i} m_i \times c^i \leq \sum_{i=0}^{i} (c-1)c^i$$

$$= (c-1) \sum_{i=0}^{i} c^i$$

$$= (c-1) \frac{(1-c^i)}{1-c}$$

$$= c^i - 1 < c^i$$

Since our greedy solution$SOL' \geq c^i \Rightarrow SOL' > SOL$, we conclude that the non-greedy solution is not optimal.

7.3

Let the coin denominations be $\{1, 4, 6\}$, and the value to make change for be 8. The greedy solution would result in the collection of coins $\{6, 1, 1\}$ which uses 3 coins but the optimal solution would be $\{4, 4\}$ which uses only 2 coins.

8 Exercise 15.4-5

Build a temporary array for solving the largest increasing subsequences.
Traverse the target array, and try to insert target element to temporary array.
– If the target element is bigger than all elements of temporary, then add target element in the end of temporary array.
– Otherwise, use target element instead of the elements of temporary array.

I don't think this greedy algorithm would work, I will leave it for HW5 by using DP.