CS218, Winter 2024, Due: 11:59pm, Fri 2/16, 2024
Assignment #3 [1] Required Programming Due: 11:59pm, Fri 2/9, 2024
Name: Aaryan Bhagat Student ID: 862468325

---

1 (2 + 0.5 pts) Everything on sale!

1. (0.3 pt) Assume we have a current benefit of X for W dollars and in which we buy one item from each group. Assume we take an item p_j belonging to group g_i from which we have already taken an item p_i. If instead of p_j we buy a new item $p_k$in a new group g_k. So the net benefit we will now have is X + $(p_k - t_k)$. Hence its better to always buy from a new group than take multiple from the same group.

2. (0.3 pt)

   - Our dp is of the form dp[W][n][2]. Where dp[W][i][0] represents the maximum benfit possible for weight W and items till item i and the ith item is not chosen in the list. Similarly dp[W][i][1] means the same except that the ith item is chosen.
   - A recurrence relation will be of the form
     - $dp[W][i][0] = max(dp[W][i-1][0], dp[W-t_i][i-1][1])$. We are taking both cases of whether $i-1$ is chosen or not.
     - $dp[W][i][1] = dp[W-t_i][i-1][0] + p_i - t_i$ if p_{i-1} belongs to g_i. Taking only the 1 possible case of when $i-1$ is not chosen as it belongs to group g_i.
     - $dp[W][i][1] = max(dp[W-t_i][i-1][1] + p_i - t_i, dp[W-t_i][i-1][0] + p_i - t_i)$. We are taking both cases of whether $i-1$ is chosen or not.
   - Base case of $dp[W][0][0-1]$ is 0.
   - Another base case, when W = 0 then $dp[W][i][0-1] = 0$.
   - We return the value of $max(dp[W][n][0], dp[W][n][1])$
   - Note that it will not always be optimal regarding the number of items but it is optimal regarding the total benefit.

3. (0.6 pt)

   - In order to make the optimal choice with regarding to number, each element of dp matrix will be a tuple (benefit, count).

[1] Some of the problems are adapted from existing problems from online sources. Thanks to the original authors.

- In the dp recursion above the max operation will be replaced by a new function f. f will do a max operation on the benefit element of tuple and if its equal then will do a min operation on the count element.
- If $dp[W][i-1][0]$ is chosen then that means ith element is not included, the tuple of $dp[W][i]$, will be same as that of $dp[W][i-1][0]$.
- If $dp[W-t_i][i-1][1]+p_i-t_i$ is chosen then tuple of $dp[W][i]$ will get updated in both benefit and the count will 1 + count value in tuple of $dp[W-t_i][i-1][1]$.
- return function f of $dp[W][n][0-1]$
- Time complexity of this function is same as the above function which is O(N*W).

4. (0.4 pt) We choose an item when the second element in the dp recursion formula is chosen. So just maintain an array sort of and update it with element index i whenever the $dp[W-t_i][i-1][1]+p_i-t_i$ is chosen. Then print the array in reverse to give the order. No need to print in reverse though as order is not important, only the list of items.

5. (0.4 pt)

    - Here if we take lets suppose product p_i we can take many instances of it from 1 to the max amount given the weight W.
    - So our recursion value is modified, in our $dp[W][i][1]$, we modify the rhs of this equation to $dp[W-x*t_i][i-1][1]+x*(p_i-t_i)$ where x will go from 1 till it satisfies the condition $x*t_i <= w$ where w is the current budget left.
    - Complexity wise it will be O(nW^2)

6. (0.5 pt bonus)

    - We can modify the above code as, whenever the ith element is included, we do not decrease i to i-1.
    - Then the relation becomes
    - $dp[W][i][1] = dp[W-t_i][i][1]+p_i-t_i+dp[W-t_i][i-1][0]$ if i-1 is in the same group as i.
    - $dp[W][i][1] = max(dp[W-t_i][i-1][1]+p_i-t_i, dp[W-t_i][i][1]+p_i-t_i)$. We are taking max of 2 cases when i-1 chosen or when i is chosen as we can choose i again.
    - If current weight w goes ¡=0 then just fill 0.
    - Time complexity for this is O(nW)

2 Morse Code (1.5 pts)

1. (0.2 pt)
   (a) For UCR the string $.. - -. - .. - ..$
   (b) Other interpretation for this morse code string EPNPE (12), UCL(3),

2. (0.8 pt)
   (a) We create an array named dp which stores the minimum stroke morse code for each index i, $dp[0..n], index starts from 1 to n.$
   (b) Base case $dp[0] = 0$.
   (c) For $dp[1]$ just add manually seeing if its a -or . and add corresponding number of strokes as the value for $d[1]$
   (d) Iterate through the array, at $a[i]$, match all possible words from dictionary with the substring $1..i$. For e.g if letter S is there and it is ... then match from i-2 to i using the check function which is given to us and takes O(1) time. If it matches then the value will become $1 + dp[i - 3]$.
   (e) Do this for every possible word in dictionary for a specific index i. Then the minimum of all these values will be the final value for $dp[i]$.
   (f) Do this for i 1..n and return answer n;

3. (0.2 pt) Complexity is O(N*K) where N is the length of the string and K the size of the dictionary.

4. (0.3 pt)
   (a) The whole algorithm is shown below, the answer is 3 and string is ZG.

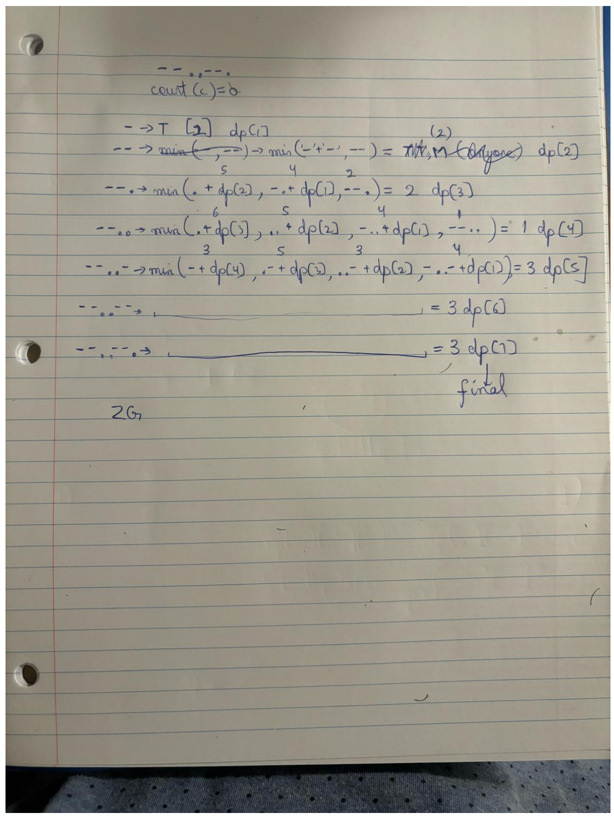

Figure 1: Caption