HW #2 - Problem 1

Ryan St.Pierre (ras70)

September 20, 2017

Problem 1A

Goal: Prove that if there is a solution, it is at most 2m-1.

Proof by contradiction

Assume (towards contradiction) that there is a solution A of q potions such that the total amount of healing is $2m-1$. Let S be the array of healing amount associated with each potion in the solution set. In other words let S be the array $[w_1...w_q]$ where w_k corresponds to the k^{th} potion used to heal the Pokemon.

Let $a[i]$ be the sum of values in S from index 1 to i (inclusive). Given the assumed solution A it must follow that $a[q] = 2m - 1$. Additionally, given the constraint of the problem (i.e. the healing power for all potions must be less than m), $1 < S[q] < m$ must hold. Using these two facts produces the following:

$$a[q-1] = a[q] - S[q]$$

Therefore:

$$m \leq a[q-1] \leq 2m - 2$$

This shows for any valid value of $S[q]$ there exists a solution B of $q-1$ potions such that the amount of healing is between m and $2m-2$. Any these possible solutions are both valid (have healing power $\geq m$) and have lower healing power than A. Since all valid solutions must minimize the total healing power and a solution B exists with healing power less than A, A must not be a valid solution. Thus, by contradiction A cannot be a valid solution. Therefore the solution corresponding to the total healing $2m-1$ cannot exist.

It can easily be seen that there is a solution that has total healing power equal to $2m-2$. This is shown below:

m = 4 with available potions = [3, 3]

The only valid solution is S = {3,3}
This solution has total healing value, h, satisfying
$$h = 6 = 2 * (4) - 2 = 2m - 2$$

Since all potions have integer healing values it must hold true that $2m-2$ is the upper bound of valid solutions, since it has been proven a solution with healing power $2m-1$ cannot exist.

Problem 1B

Let p be the list of available potions. If we consider the i^{th} potion in p we can either chose to include it in our summation of values from 1 to i or not. The two possible outcomes of this decision are described below:

- *(don't include i).* If the i^{th} potion is not included then the first $1...i+1$ values must be used to create the total healing j. In other words, $a[i-1, j]$ must be true.
- *(include i).* If the i^{th} is not included then the first $1...i+1$ values must be used to create the total healing $j - w_i$, since potion i provides w_i healing. In other words, $a[i-1, j-w_i]$ must be true.

We can produce the value j if *either* of these above cases is correct. However, case two only holds true if $w_i < j$. If $w_i > j$ and the i^{th} potion is included the value of j can never be achieved because all values in the potion list are positive. Therefore:

$$a[i,j] = \begin{cases} a[i-1,j] \;||\; a[i-1, j-w_i], & \text{if } w_i \geq j \\ a[i-1,j], & \text{otherwise} \end{cases}$$

However, having a conditional in the transition function becomes a bit cumbersome. To avoid this conditional we can handle the $w_i \geq j$ case by defining behavior for $a[i,j]$ when $j < 0$. This is described in greater depth in the *Base Cases* section below. Without the conditional the transition function is given as:

$$\boldsymbol{a[i,j] = a[i-1,j] \;||\; a[i-1, j-w_i]}$$

Base Cases:

There are three base cases, all described in greater length below.

$$a[i,0] = \text{ TRUE for } 1 \leq i \leq n \tag{1}$$

Zero healing power can be achieved for any number of potions given. This is achieved by including no potions in the solution set. This even holds for the case when i equals zero (when you have no potions to chose from).

$$a[0, j] = \text{ FALSE for } 1 \leq j \leq 2m - 1 \tag{2}$$

This base case (2) corresponds to attempting to satisfy a non-zero healing power with no potions. This base case returns false because a finite healing power can not be supplied with no potions.

$$a[i, j] = \text{ FALSE for } j < 0 \tag{3}$$

This base case (3) states that any attempt to crate a total negative healing power with i potions is unsuccessful. This falls directly from the constraint in the problem statement where the healing value of all potions is assumed to be positive.

Problem 1C

Algorithm

Once the proper transfer function has been established (as was done in Problem 1B) it can be used to define the proper algorithm. The algorithm can be summarized in the following steps:

1. Set the base case values for a corresponding from Base Case 1 and Base Case 2 from Problem 1B above.

2. Calculate $a[i][j]$ for all values i: 1 to n and j: 1 to $2m - 1$. This covers all the possible solutions since all the potions given are considered and given the proof from Problem 1A.

3. Find the smallest possible value of $j \geq m$ for which $a[i][j]$ is true, where i ranges from 1 to n. In actuality this smallest possible value of j can be checked and stored when the values of $a[i][j]$ are being calculated in part 2 (please reference the pseudo-code for how this is done).

For further clarity please reference the pseudo-code given below.

Visualizing the DP Table

The DP table can be visualized as a 2 dimensional grid. If we allow i to be represented by the columns in the table and j by the rows in the table then the columns range from 0 to n and the rows range from 0 to $2m-1$ (a further

description of these bounds and how they were chosen is given below). The first row of the DP table is full of true values. This corresponds to the base case (Case 1) where $a[i, 0]$ is true for all i between 1 and n. The first column (excluding the first cell at (0,0)) is full of false values. This corresponds to the base case (Case 2) where $a[0, j]$ is false for all j between 0 and $2m - 1$.

In this representation the value of any cell in the table is equal to the value of the cell directly to its left logically or'd with the value one column to the left and w_i rows up (if this location is in bounds).

The solution to this problem corresponds to the first row, starting at m, that has at least one true value for all of its columns. If a row has at least one true value for all of its columns it means that there exists some i in which potions 1...i can be used to make the total healing power that corresponds to that row.

Ordering/Bounds

Transition function:

$$a[i, j] = a[i - 1, j] \;||\; a[i - 1, j - w_i]$$

From the transition function it is clear that the pair (i, j) is dependent on $(i-1, j)$, $(i-1, j-w_i)$, where $j - w_i$ can fall anywhere between 0 and $j - 1$. Thus, when $a[i, j]$ is calculated the values $a[i-1, j]$ and $a[i-1, 0...j-1]$ must already be calculated. In the DP table this can be interpreted as values to the left and above of the value being calculated.

The bounds for the i and j are given as $[0, n]$ and $[0, 2m-1]$ respectively. The bounds for i falls directly from the number of potions in the list since all combinations of potions in the list must be checked. The bounds for j fall from the possible solution set. It was proven in Problem 1A that a given solution is bound by $2m - 1$ (actually $2m - 2$, but we will use $2m - 1$ to be consistent with the proof statement in Problem 1a).

An ordering that ensures $a[i, j]$ has the values necessary to be calculated is given below:

```
i: 0 -> n
j: 0 -> 2m-1
```

The values of $i = 0$ and $j = 0$ correspond to the base cases of the algorithm. Thus, once these have been accounted for the necessary iterations through the variables i and j needed are:

```
i: 1 -> n
j: 1 -> 2m-1
```

Pseudo-code

```
computeSmallestW(int [] potions, m) {

    int n = potions.length
    minW = INTEGER.MAX

    a[0,0] = TRUE

    \\ base cases
    for i = 1 to n
        a[i,0] = TRUE

    for j = 1 to 2m-1
        a[0,j] = FALSE

    for i = 1 to n {
        for j = 1 to 2m-1 {
            // Transfer function
            wi = potions[i] //assumes potions is one-indexed
            if (wi < j)
                a[i, j] = a[i-1, j] || a[i-1, j-wi]
            else
                // corresponds to Base Case 3
                a[i, j] = a[i-1, j]

            if (a[i,j] && j>=w) *
                minW = min(minW, j)
        }
    }
    return minW*
}
```

*Given the way the values are iterated we can return at this point. However, this is not done to generate all values of the DP grid.

Running time

From the pseudo-code above the algorithm has a loop from 1 to n, a loop for 1 to 2m-1, and two nested for loops ranging from 1 to n and from 1 to 2m-1. The work within these loops is constant (simple comparisons and setting of values). Thus, the running time of this algorithm can be given by $O(n + (2m - 1) + n * (2m - 1))$, which is bounded by $O(nm)$. This is the desired running time given in the problem statement. To see more clearly how $O(n + (2m - 1) + n * (2m - 1))$, is bounded by $O(nm)$ reference below:

$$\begin{aligned} O(n + (2m - 1) + n * (2m - 1)) &= O(n(2m - 1 + 1) + (2m - 1)) \\ &= O(2m * n + 2m - 1) \\ &= O(2m(n + 1) - 1) \\ &= O(Am(n + B) - C) \\ &\quad \text{where A, B, and C are constants} \\ &= O(m * n) \end{aligned}$$

Proof of correctness

Let $(i, j) < (i', j')$ if $i < i'$ or ($i = i'$ and $j < j'$). Also let a[i,j] be the value of the transfer function a for values i and j.

Proof by induction:

Induction Hypothesis: The algorithm is correct for all values of $a[i, j]$ where $(i, j) < (i', j')$.
Base cases:

$$a[i, 0] = \text{ TRUE for } 0 \leq i \leq n$$

This base case corresponds to requiring zero potion to supply zero healing.

$$a[0, j] = \text{ FALSE for } 1 \leq j \leq 2m - 1$$

This base case corresponds to attempting to satisfy a non-zero healing power with no potions. This base case returns false because a finite healing power can not be supplied with no potions.

$$a[i, j] = \text{ FALSE for } j < 0$$

This base case states that any attempt to crate a total negative healing power with i potions is unsuccessful. This falls directly from the constraint

in the problem statement where the healing value of all potions is assumed to be positive.

Inductive step:

When computing $a[i', j']$ by induction hypothesis $a[i' - 1][j']$ and $a[i' - 1][j' - w_i]$, have been computed correctly (given $w_i > 0$).

The algorithm calculates the value of $a[i' - 1][j' - 1]$ by taking the logical OR of these two results, which are correct. These two cases correspond to the two scenarios in which i' potions can be used to produce exactly j' healing - when the i'^{th} potion is used and when it isn't.

Since the algorithm records the smallest j' where $a[i'][j']$ is true and $j' > m$ and $a[i'][j']$ is computed correctly, it must follow that the algorithm returns the smallest possible total healing greater than m.