CSCI 570 - Fall 2019 - HW 4

Junzhe Liu, 2270250947

October 17, 2019

1 Graded Problems

1.1 From the lecture, you know how to use dynamic programming to solve the 0-1 knapsack problem where each item is unique and only one of each kind is available. Now let us consider knapsack problem where you have infinitely many items of each kind. Namely, there are n different types of items. All the items of the same type i have equal size w_i and value v_i. You are offered with infinitely many items of each type. Design a dynamic programming algorithm to compute the optimal value you can get from a knapsack with capacity W .

Suppose array $d[w]$ represents the maximum value in the bag when the sum of the weights carried in the bag is smaller than w.

$$d[w] = \begin{cases} 0 & \text{if } w = 0 \\ 0 & \text{if } w_i > w \\ \max_i \{v_i + d[w - w_i]\} & \text{otherwise} \end{cases} \tag{1}$$

Initially, the bag has no value because it's empty. Furthermore, if all the weights are greater than the bag's volume, then none of them can be put inside, the value should also be zero. After that, we chose the best item which maximizes the sum $v_i + d[w - w_i]$, and put it in the bag.

1.2 Given a non-empty string s and a dictionary containing a list of unique words, design a dynamic programming algorithm to determine if s can be segmented into a space-separated sequence of one or more dictionary words. If s = "$algorithmdesign$" and your dictionary contains "$algorithm$" and "$design$". Your algorithm should answer Yes as s can be segmented as "$algorithmdesign$".

I came up with a recursive algorithm(or actually a divide-conquer). If the string s is exactly a word in the dictionary, then no split is needed, return true directly. If split is necessary, then we try to split the string at every possible position and check recursively if the two substrings can be splited into the words from the dictionary.

Algorithm 1 Determine whether a string can be splited into a set of words in the dictionary

```
function isInDic(string $s$)
    if then$s$ is in Dictionary:
        return true;
    end if
    for index $i$ from 1 to s.length()-1 do (both inclusive)
        Split $s$ at index $i$, makes two substrings: $left$ and $right$;
        if then$isInDic(left)$ and $isInDic(right)$ is true:
            return true
        end if
    end for
    return false
end function
```

1.3 Given n balloons, indexed from 0 to $n-1$. Each balloon is painted with a number on it represented by array nums. You are asked to burst all the balloons. If the you burst balloon i you will get $nums[left] \cdot nums[i] \cdot nums[right]$ coins. Here left and right are adjacent indices of i. After the burst, the left and right then becomes adjacent. You may assume $nums[-1] = nums[n] = 1$ and they are not real therefore you can not burst them. Design an dynamic programming algorithm to find the maximum coins you can collect by bursting the balloons wisely. Analyze the running time of your algorithm.

Here is an example. If you have the nums arrays equals [3, 1, 5, 8]. The optimal solution would be 167, where you burst balloons in the order of 1, 5 3 and 8. The left balloons after each step is:

$$[3,1,5,8] \rightarrow [3,5,8] \rightarrow [3,8] \rightarrow [8] \rightarrow [] \tag{2}$$

And the coins you get equals:

$$167 = 3 \cdot 1 \cdot 5 + 3 \cdot 5 \cdot 8 + 1 \cdot 3 \cdot 8 + 1 \cdot 8 \cdot 1 \tag{3}$$

Suppose all the array elements are positive integers. We denote $d[i][j]$ is the maximum value of earned coins for a sub-array from index i to j. Let's suppose that mid is the last balloon to be burst in this sub-array, then $d[i][j]$ should be:

$$d[i][j] = \begin{cases} A[i] & \text{if } i = j \\ \max_{i \le mid \le j} \{A[mid] \cdot A[i-1] \cdot A[j+1] + d[i][mid-1] + d[mid+1][j]\} & \text{otherwise} \end{cases} \tag{4}$$

Initially, $d[i][i] = A[i]$ for all i. Then $d[1][N]$ will be the answer to the question. What's worth mentioning is that we should build up this matrix by growing the length of the interval, that is to say:

```
for length = 1; length <= N-1; length++ do
    for i = 1; i <= N; i++ do
        j = i + length;
        ...renew d[i][k]...
    end for
end for
```

1.4 Kleinberg and Tardos, Chapter 6, Exercise 5.

Sentence segmentation: Given a long string of letters $y = y_1 y_2 \cdots y_n$, we denote $d[i]$ as the maximum quality of the substring $y_1 \cdots y_i$, where $i = 1, \ldots, n$. We use bottom-up technique to build this array. Initially, $i = 1$, $d[i] = d[1] = quality(y_i)$. Then for any $1 < i \leq n$, we consider where to split the sentence, we will choose the largest

$$d[j] = \begin{cases} \text{quality}(1 \ldots n) & \text{if } j = 1 \\ \max_{1 \leq i < j} \left\{ d[i] + \text{quality}\left(substring(i \ldots j)\right) \right\} & \text{otherwise} \end{cases} \tag{5}$$

The solution to this problem will be: $d[n]$.

2 Practice Problems

2.1 Kleinberg and Tardos, Chapter 6, Exercise 6.

2.2 Kleinberg and Tardos, Chapter 6, Exercise 10.

We denote $d_A[i]$ as the maximum value of the works in minutes 1 through i that ends on machine A, and define $d_B[i]$ analogously for B.

For A, the occurence equation will be:

$$d_A[i] = a_i + \max \left\{ d_A[i-1], d_B[i-2] \right\} \tag{6}$$

similar for B:

$$d_B[i] = b_i + \max \left\{ d_B[i-1], d_A[i-2] \right\} \tag{7}$$

when we approach $d_A[n]$ and $d_B[n]$, choose $\max\{d_A[n], d_B[n]\}$

2.3 Kleinberg and Tardos, Chapter 6, Exercise 24