Problem Set 2, Problem 3 **Jihong Tang**
COMPSCI 260 NetID: jt290
Work Date: 09/24/2018 Due Date: Fri 09/28/2018 5pm

Introduction to the problem

Problem 3 from problem set 2 is all about the pebbling on grid problem and the dynamic programming solution to it.

For question 3a, we need to enumerate all possible patterns that can be achieved in one single column under the given conditions and some limitations.

For question 3b, we are asked to design an $\Theta(n)$ dynamic programming algorithm for computing a valid placement of maximum value on the original grid, which means to get the optimal solution to the problem.

For question 3c, we need to implement our designed algorithm in python and take the given input to get some outputs.

For question 3d, we are offered a chance to think of an example grid to let the optimal solution included at least one empty row.

Problem 3a

I found the number of distinct valid pebble placements that can occur in a single row is eight. The placements can be shown as following(counting grids from left to right):

1. None pebble in neither one of the four grids
2. One pebble in the first grid
3. One pebble in the second grid
4. One pebble in the third grid
5. One pebble in the fourth grid
6. Two pebbles in the first and third grid respectively
7. Two pebbles in the first and fourth grid respectively
8. Two pebbles in the second and fourth grid respectively

Problem 3b

For problem 3b and 3c, I worked out an iterative dynamic programming algorithm of which time complexity is $\Theta(n)$.

Algorithm discription

My algorithm starts with creating a data structure to store the compatible information between different patterns. In python, I chose to use dictionary. Let's name it as C_j for all possible pattern j. Afterwards this dictionary will play an important role in the iterative step.

For the solving procedure, I will scan the 2D data array from the first row to the last row, and I will find out the optimal solution to the k^{th} row with pattern j subproblems at each column. I will use one dictionary Opt_k to store the optimal solution when k^{th} row is the last row in our subproblems. Opt_k will contain eight elements, each one is the maximum value when the last column is in pattern j. I will use one variable $N_{k,j}$ to store the value of k^{th} row in pattern j. The relationship between different subproblems can be illustrated as Equation1

$$Opt_k[j] = \max_{i \subseteq C_j}(N_{k,j} + Opt_{k-1}[i]) \tag{1}$$

When we make the subproblem bigger and bigger, finallly getting $k = n$, we have worked out the Opt_n to find our final result. Just compare the optimal values in different pattern j in the dictionary Opt_n to find the fianl maximum value in an optimal displacement.

Algorithm analysis

To argue that this algorithm is correct, I would make the statements as following. The iterative dynamic programming algorithm focuses on subproblems with k^{th} and pattern j at each new row, it included all possible subproblems in the iterative process. In addition, I tested several test sample after the inplement in 3c to prove the algorithm will work. The detailed information about the test sample can be found in **pebbles.py**.

About the time complexity, since some preparation work including the creatig of compatible dictionary and data inputing and some calculation work at each column including finding the maximum value and calculating the pattern value are both limited to a constant amount of time, my algorithm will run in $\Theta(n)$ time. The major part of the comsuming time is to iterate through n columns and each column do a constant amount of calculation work mentioned above.

Problem 3c

The code solution of the $\Theta(n)$ dynamic programming algorithm can be found in **pebbles.py**.

From my code, the maximum value of a valid placement for this sample grid is 10753. I have run some own-created sample to test my program and some detailed comments can be seen in the coding file.

Problem 3d

To work out the sample grid where the optimal solution is obtained by not placing anything on at least one of the rows, I firstly think about the condition when this solution will be taken. I believe when a row with small values is sandwiched by two rows with large values, sometimes not choosing any value on the small value row will be a better choice especially when choosing some values from the low value row will make some large value unable to be chosen.

After the thinking procedure shown in above, I would like to provide the simple example as shown in Table1.

Table 1: One simple example to question 4d

1000	500	900	600
10	20	30	40
500	1000	600	900

In this example, the optimal solution is obtained by not placing anything on the second row.

Citation

For the iterative dynamic programming algorithm, I gave some hints to my classmate Xi Li to complete her own algorithm design and code implement including the idea that storing the optimal solution to each pattern at each new column.