CMPSCI 311: Introduction to Algorithms **Spring 2018**

Discussion 6

3/23/2018 Name:

You will be randomly assigned groups to work on these problems in discussion section. List your group members on your worksheet and turn it in at the end of class. Write first and last names. Each group member should turn in their own paper.

1. **Knapsack.** Grids are provided for the calculations in this problem, but there may be cases where the grid is unnecessary.

   (a) Say you have items of weights [2, 3, 5, 6]. Find a sum of these weights that gets you as close as possible to 12 without going over. Use the method shown in lecture to find this.

   From lecture:

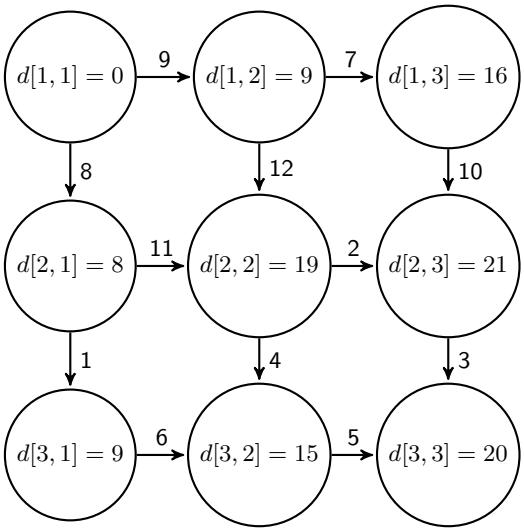
   $M[j, w] \leftarrow \max(M[j-1, w], w_j + M[j-1, w-w_j])$

   (b) Say you have items of weights [2, 4, 8, 10]. Find a sum of these weights that gets you as close as possible to 22 without going over.

   (c) Qualitatively, when is it faster to exhaustively calculate the sums of all combinations of weights to find the best set rather than using the algorithm shown in class?

   (d) Does the knapsack algorithm described in class work given negative weights? Why or why not?

2. **Grid Dijkstra.** Say you have a directed graph that resembles a grid. Every node has an edge going down and an edge going right, unless it is on the bottom or right perimeter. Node (i, j) is in the i^{th} row and the j^{th} column of the grid. All edges are positive. You want to be able to calculate the distance from node $(1, 1)$ to all other nodes using dynamic programming. Below is a picture of a grid we might take as an input.



   (a) Define a recursive way of calculating the minimal distance to $(1, 1)$. You may assume that (x, y) is not $(1, 1)$ and is not on the perimeter.
   $d[x, y] \leftarrow$

   (b) What values of d should be set at the start of the algorithm?

   (c) In what order should the values of d get updated?

3. **Bad Use of Dynamic Programming.** What is the Θ complexity of the number of bits of memory this algorithm will need? Design a more memory-efficient algorithm. What is your algorithm's memory complexity? By what factor is yours better?

Algorithm 1 `Factorial`$(factorial\text{-}of)$

```
cache[1] = 1
for i from 2 to factorial-of do
  cache[i] = cache[i − 1] × i
end for
return cache[factorial-of]
```