CMPSC 465 Assignment - Dynamic Programming

Name: Anthony Vallin PSU Username: aav5195

Format Requirement

- Algorithms in pseudo code **MUST** be placed in code blocks/fences (5%), and use either the `cpp` or `java` syntax highlighter.
- Algorithms should follow the pseudo code standard described in handout 1. (2%)
- Do NOT change the template except the answer portion. (5%)
- Formulas and equations should be in math mode using Latex math symbols. (5%)
    - Markdown math tutorial: http://tug.ctan.org/info/undergradmath/undergradmath.pdf
    - Two ways to enter math mode:
        - Insert a pair of dollar signs: $ your equations go here $. This is the inline math mode.
        - Insert a pair of double-dollar signs: $$ your equations go here $$, which produces a standalone equation/formula set.

Problem Set

Problem 1

What does dynamic programming have in common with divide-and-conquer? What is a principal difference between them?

Answer:

They both set up a recurrence relating a solution to a larger instance to solutions of some smaller instances. The difference being that dynamic programming solves smaller instances once, while divide-and-conquer may solve the same sub problem more than once.

Problem 2

Solve the instance 5, 1, 2, 10, 6 of the coin-row problem. You need to provide 1) a table that stores solutions to subproblems, 2) steps that compute the solutions to the subproblems.

Answer:

max amount: 15

②

coin index	0	1	2	3	4	5
coin value	---	5	1	2	10	6
F( )	0	5	5	7	15	(15)
	F(0)	F(1)	F(2)	F(3)	F(4)	F(5)

$F(1) = 5$

$F(2) = \begin{cases} 1, & \text{pick-up } C_2 \\ F(1) = 5, & \text{otherwise} \end{cases}$

$F(3) = \begin{cases} 2 + F(1) = 7, & \text{pick-up } C_3 \\ F(2) = 5 \end{cases}$

$F(4) = \begin{cases} 10 + F(2) = 15, & \text{pick-up } C_4 \\ F(3) = 7, & \text{otherwise} \end{cases}$

$F(5) = \begin{cases} 6 + F(3) = 13, & \text{pick-up } C_5 \\ F(4) = 15, & \text{otherwise} \end{cases}$

Max amount = 15

Problem 3

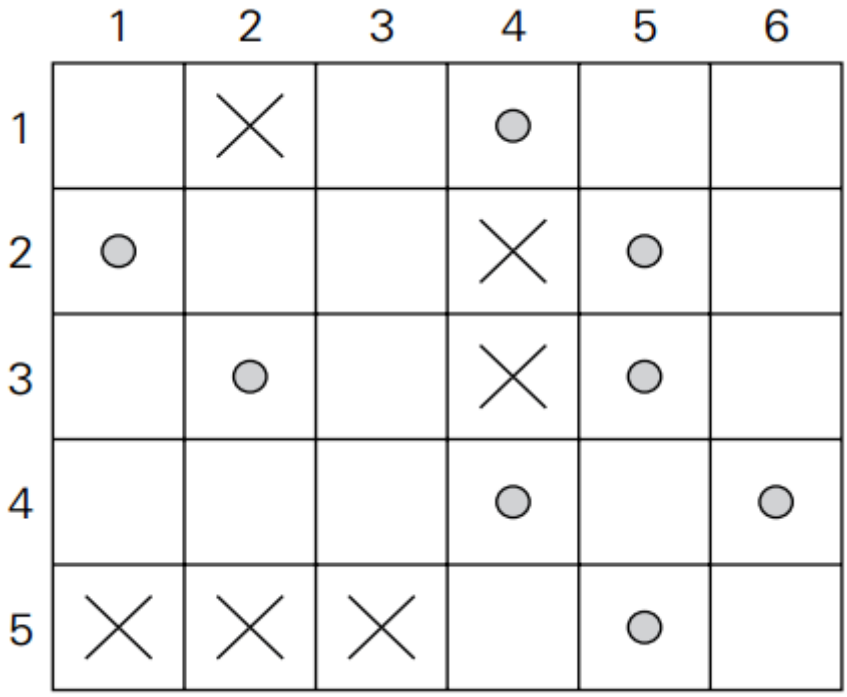
Show that the time efficiency of solving the coin-row problem by exhaustive search is at least exponential.

Answer:

Exhaustive search requires that every possible combination be checked before finding the max amount. The time efficiency for the total number of combinations is $\frac{C(2n,n)}{n+1}$, which equals $\theta(n!)$.

Problem 4

How would you modify the dynamic programming algorithm for the coin collecting problem if some cells on the board are inaccessible for the robot? Apply your algorithm to the board below, where the inaccessible cells are shown by X's. How many optimal paths are there for this board? You need to provide 1) a modified recurrence relation, 2) a pseudo code description of the algorithm, and 3) a table that stores solutions to the subproblems.



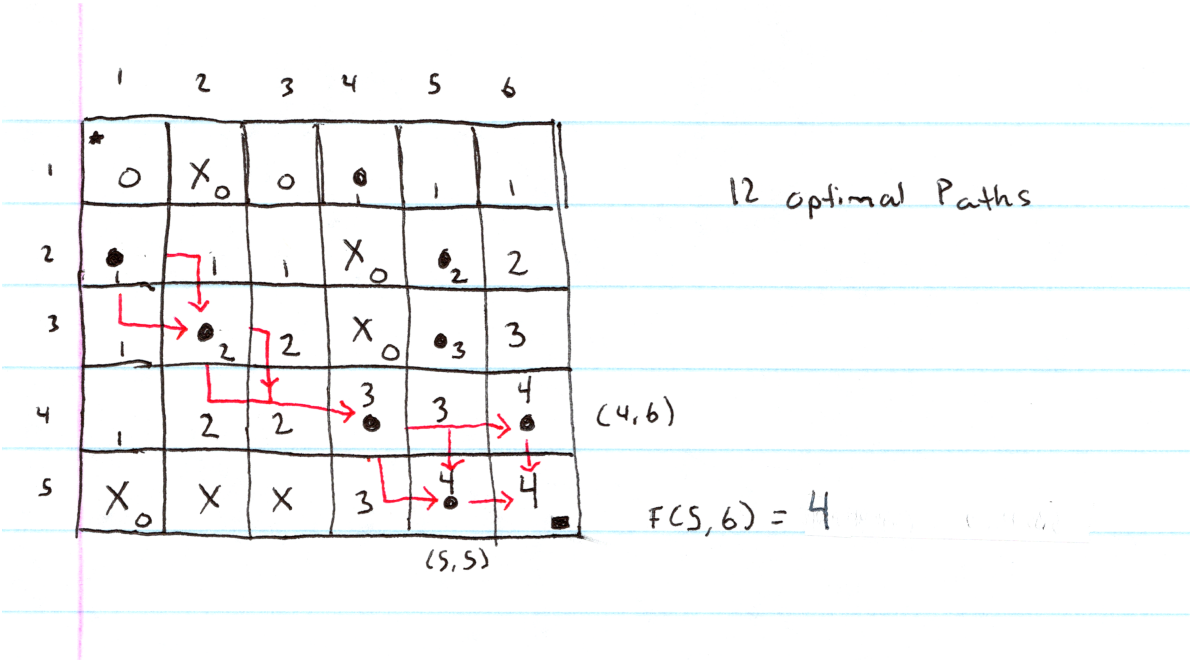
Answer:

Modified recurrence relation:

$$F(i,j) = \begin{cases} maxF(i-1,j), F(i,j-1) & \text{for } 1 <= \text{i} <= \text{n}, 1 <= \text{j} <= \text{m} \\ 0 & \text{for inaccessible block} \end{cases}$$

```
Algorithm RobotCoinCollectionObstacle(C[1..m, 1..m])
//Applies dynamic programming to compute the largest number of coins a robot can
collect //on a nXm board that contains inaccessible cells
//Input: Matrix C[1..n,1..m] whose elements equal 1 if coin is in cell or 0 for
cells without coins and 0 for inaccessible cells
//Output: Largest number of coins the robot can get from cell(m,n)
F[1,1] <- C[1,1]
for j <- 2 to m do
    F[1,j] <- F[1, j-1] + C[1,j]
for i <- 2 to n do
    F[i,1] <- F[i-1,1] + C[i,1]
    for j <- 2 to m do
        if cell inaccessible //Modification happens here
            F[i,j] <- 0
        else
            F[i,j] <- max(F[i-1,j], F[i, j-1]) + C[i,j]
return F[n,m]
```

12 optimal paths.



Problem 5

Design a `dynamic programming` algorithm for the following problem. Find the **maximum total sale price** that can be obtained by cutting a rod of n units long into integer-length pieces if the sale price of a piece i units long is p_i for $i = 1, 2, \ldots, n$. What are the time and space efficiencies of your algorithm?

Answer:

```
Algorithm BottomUpCutRod(p,n)
//Calculates maximum total sale price that can be obtained by cutting a rod of n
units
//Input: Array containing sale price of a piece i units long and a rod cut n
units long.
//Output: Maximum total sales price obtained by cutting a rod of n units.
rod[0..n]
rod[0] <- 0
for j <- 1 to n
    q <- p[j]
    for i <- 1 to j
        q <- max(q, p[i] + rod[j-i])
    rod[j] <- q
return rod[n]
```

Time efficiency = $\theta(n^2)$
Space efficiency = $\theta(n)$