SAN FRANCISCO STATE UNIVERSITY
Computer Science Department

CSC510– Analysis of Algorithms
Algorithm Challenge 4: Backtracking

Instructor: Jose Ortiz

Full Name: Edel Jhon Cenario
Student ID: 921121224

Assignment Instructions. Must read!

Note: Failure to follow the following instructions in detail will impact your grade negatively.

1. This algorithm challenge is worth 9%, and will be graded using a grading point scale where the maximum possible grade is 9 points
2. Handwriting work or screenshots of your work are not allowed. In addition, all the pseudocode done in this algorithm challenge must be done using LaTeX. Students who fail to meet this policy won't get credit for their work. Note that for pseudocode, I only want to see the compiled PDF psudocode, instead of the code to create the pseudocode.
3. Each section of this algorithm challenge is worth 2.25 points
4. Take into account that in this type of assignments, I am more interested in all the different approaches you take to solve the problem rather than on the final solution.

Problem Statement

Given a $N * N$ $Matrix$ M filled with positive integers, find all the possible cells $M(i, j)$ where indexes i and j are unique and the sum of those cells is maximized or minimized for all the possible solutions found.

The formal definition of the problem is the following:

Let $\{P_1, P_2, ..., P_k, ..., P_n\}$ be a set of solutions for this problem where $P_k = \{M(i,j)_1 + M(i,j)_2 + M(i,j)_3 + ... + M(i,j)_{m-1} + M(i,j)_m\} = S$ is a set of coordinates for integers values in a matrix, and S the sum of those integers for that solution P_k. The S sum is valid only if:

1. All the indexes i *and* j for that sum of P_k are unique
2. The integer in $M(i, j)$ is not zero
3. Index j in $M(i,j)_x$ must be the same as index i in $M(i,j)_{x+1}$
4. Index i in $M(i,j)_1$ and index j in $M(i,j)_m$ must be zero for all the solutions P_k
5. A possible solution P_k is only considered an optimal solution if the sum S of all its integers is the minimum or the maximum sum S from all the solutions P_k
6. All the vertices but the source vertex must be visited only once. The source vertex is visited twice because it plays the role of the source and destination vertex in this algorithm

For example, given the following matrix M filled with integers and zeros find all the possible results that met the above conditions.

0	3	6	0
3	0	2	5
6	2	0	1
0	5	1	0

All possible solutions are:

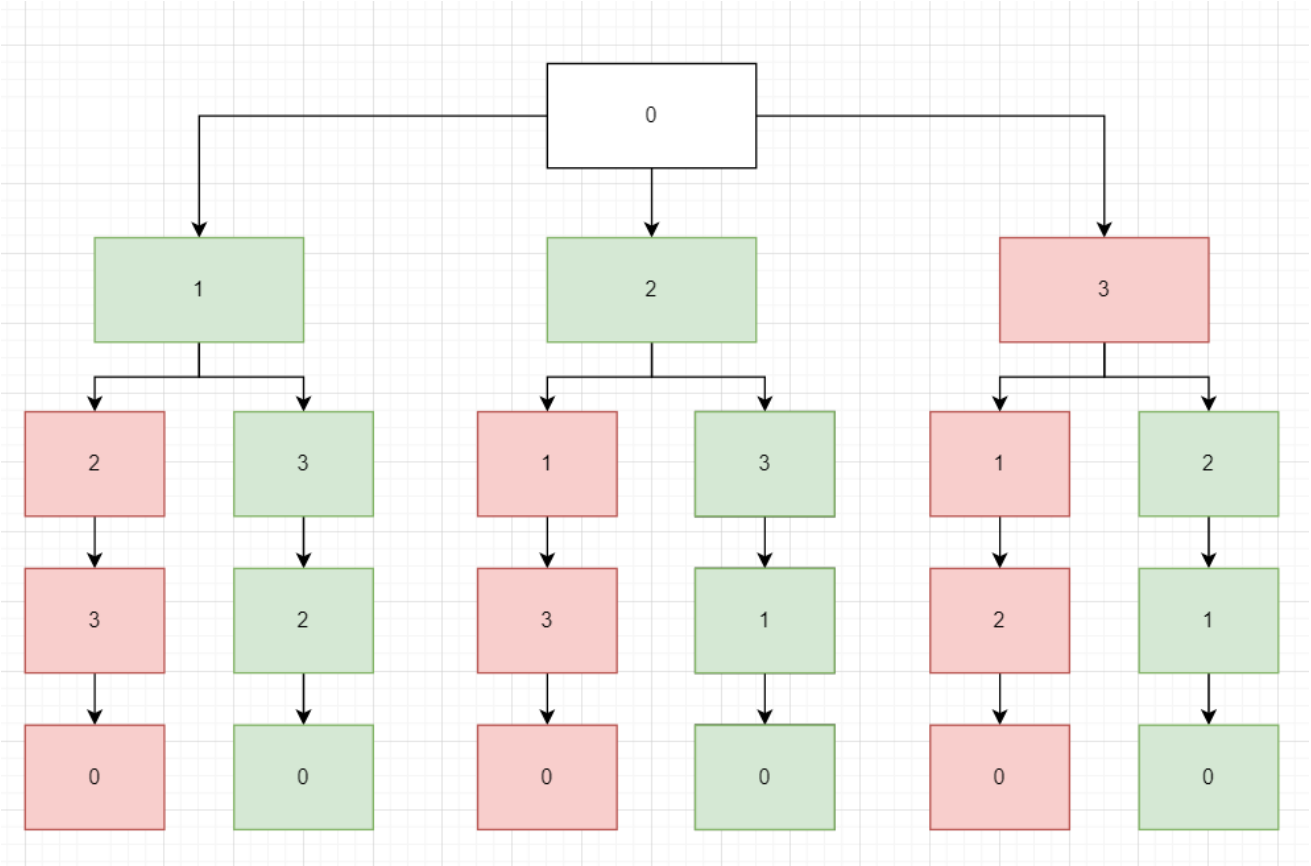
1. $P = \{M[0][1] + M[1][3] + M[3][2] + M[2][0]\} = 15$
2. $P = \{M[0][2] + M[2][3] + M[3][1] + M[1][0]\} = 15$

As you can see, solutions 1 and 2 met all the conditions above.

Your work starts here

1. Create a backtracking algorithm (using a state space tree) to find all the solutions from the example above. Note that the below matrix M is the same as the one from the example

0	3	6	0
3	0	2	5
6	2	0	1
0	5	1	0



Green is a solution, while Red is not.

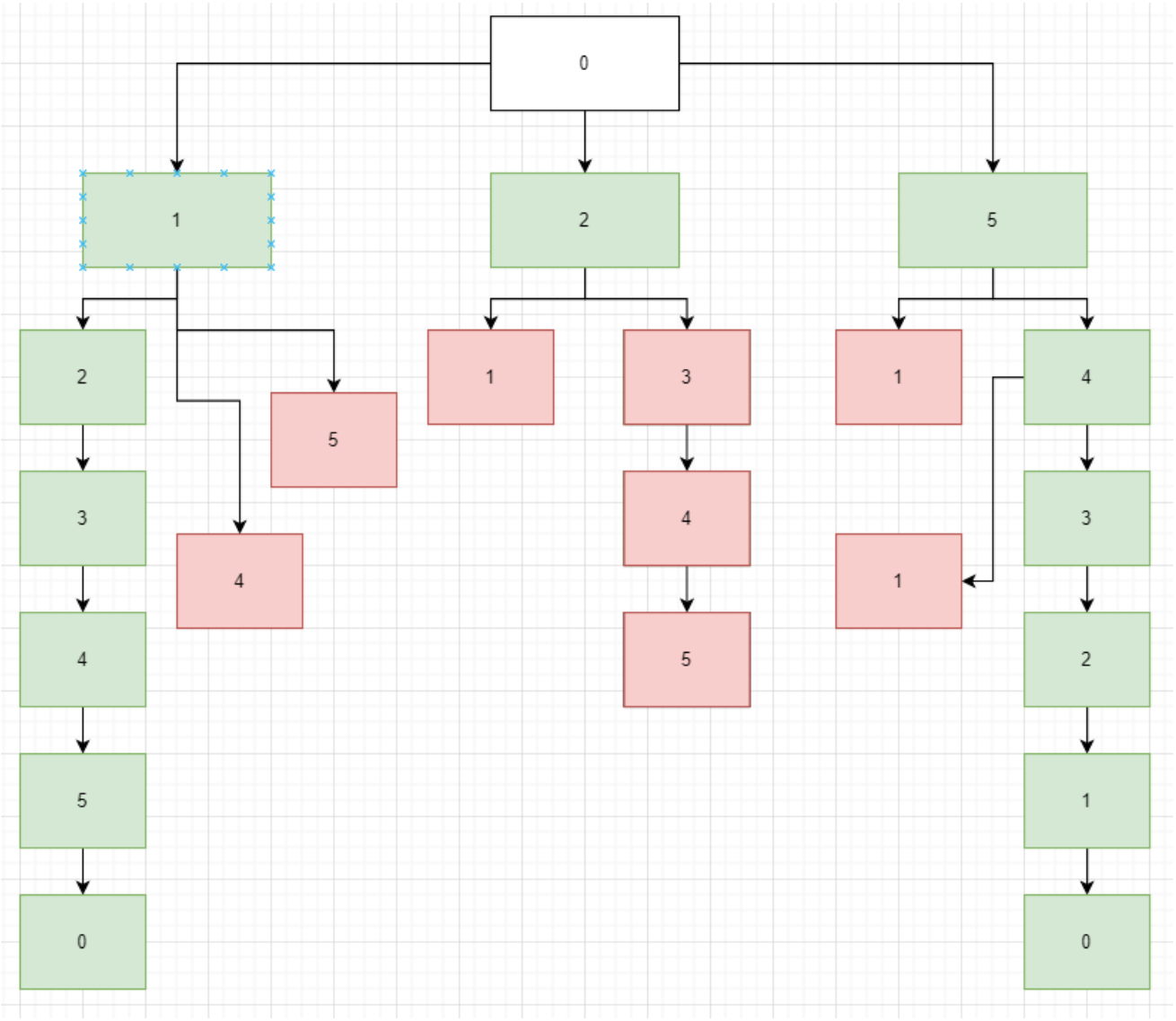
To further discuss this idea, let us look at the equation:

P = M[0] [1] + M[1] [3] + M[3] [2] + M [2] [0]
P = 3 + 5 + 1 + 6
P = 15

P = M[0] [2] + M[2] [3] + M[3] [1] + M[1] [0]
P = 6 + 1 + 5 + 3
P = 15

2. Use your backtracking algorithm to find all the solutions for the following matrix M:

0	7	1	0	0	8
7	0	5	0	9	6
1	5	0	4	0	0
0	0	4	0	2	0
0	9	0	2	0	3
8	6	0	0	3	0



Green is a solution, while Red is not.

P = M[0] [1] + M[1][2] + M[2] [3] + M [3] [4] + M [4] [5] + M [5] [0]
P = 7 + 5 + 4 + 2 + 3 + 8
P = 29

P = M[0] [5] + M[5] [4] + M[4] [3] + M[3] [2] + M[2] [1] + M[1] [0]
P = 8 + 3 + 2 + 4 + 5 + 7
P = 29

3. Based on your backtracking algorithm, create pseudocode (with LaTeX) that finds all the solutions for any given matrix M. Note that your pseudocode can be either iterative or recursive.

function GetSum (vertex, v, count, matrix, position, answer)

INITIALIZE: v = Array of boolean values: true means that it has been visited, while false means that it has not been visited. M = 2D Matrix. l = matrix length. i = 0. x = number of nodes. position = current position in the matrix.

Assert: i, x, and j cannot be negative. M cannot have negative integers.

```
if answer ≤ 0 then
        return 0                                                        T(0) = 0
end if
for from i to x, inclusive do                                           n
        if i = 0 and j = 0 then                                         1
                continue
        end if
        if count = x then
                answer = max(answer, vertex + M[ position ][ 0 ])       1
                return answer
        end if
        if v[ i ] = false then                                          1
                v[i] = true
                GETSUM(vertex + 1, v, count, matrix, position, answer)  T( n - 1 )
        end if
end for
return answer
end function
```

4. Analyze your backtracking algorithm and find its time and space complexities based on that analysis. Note that if your pseudocode is recursive, you must use the substitution method first, and then check your time complexity result using the Master Theorem. On the other hand, if your pseudecode is iterative, then you must use a step counting approach.

Our time complexity or the algorithm will be O(n!)
To further discuss the process, let us look at the
algorithm itself.

Base case: T(0) = 0
T(n) = n * [T(n - 1) + 3]
T(n) = n * [T(n - 2) + 3 + 3]
T(n) = n * [T(n - 3) + 3 + 3 + 3]
T(n) = n * [T(n - k) + k]

T(n) = n * n!
Therefore: Time Complexity = O(n!)