CSC510– Analysis of Algorithms
Algorithm Challenge 3: Backtracking

Instructor: Jose Ortiz

Full Name: Wing Lee
Student ID: 920558688

Assignment Instructions for students. Must read!

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

1. All the assignments MUST have the student full name, and student id. If this information is not there, we won't grade your assignment, and it will be considered not submitted.
2. This homework is worth 15%. This homework has two problems. The second problem is divided into several parts.
3. Handwriting work is allowed as long as the work is clear and readable. If we can't understand your work, then we can't grade it.
4. Students must turn this assignment in on Canvas in PDF format, assignments submitted in a file format other than PDF will be considered not submitted and graded as such.
5. Blank assignments, or assignments missing work will be graded as such. No exceptions!. It is the responsibility of students to check if the assignment was submitted in the correct format.
6. Please read the late work policies stated in the syllabus. Homework submissions that do not follow these policies will get an unsatisfactory grade.

Problem Statement

Given a $N * N$ *Matrix* M filled with non-negative 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 considered optimal only 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.

	A	B	C	D
A	0	3	6	0
B	3	0	2	5
C	6	2	0	1
D	0	5	1	0

$P1 = \{M[0][1]\} = 3$

$P1 = \{M[0][1], M[1][3]\}$

3 5

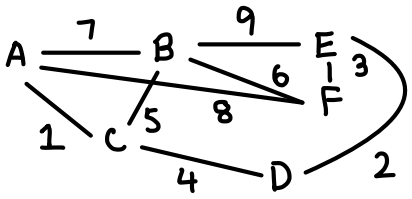
All possible solutions are:

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. Note that both of the solutions represent the maximized and minimized solutions for this problem.

Your work starts here

1. (5 points) Create a state-space tree to design a backtracking algorithm to find all the solutions for the following matrix M. Note that approaches other than creating a state-space tree for this algorithm won't get credit:



	A	B	C	D	E	F
A	0	7	1	0	0	8
B	7	0	5	0	9	6
C	1	5	0	4	0	0
D	0	0	4	0	2	0
E	0	9	0	2	0	3
F	8	6	0	0	3	0

	A	B	C	D	E	F	Min
A	∞	6	0	∞	∞	7	1
B	2	∞	0	∞	4	1	5
C	0	4	∞	3	∞	∞	1
D	∞	∞	2	∞	0	∞	2
E	∞	7	∞	0	∞	1	2
F	5	3	∞	∞	0	∞	3
Min	0	3	0	0	0	1	

Bound(A) = 14 + 4 = 18

	A	B	C	D	E	F
A	∞	3	0	∞	∞	6
B	2	∞	0	∞	4	0
C	0	1	∞	3	∞	∞
D	∞	∞	2	∞	0	∞
E	∞	4	∞	0	∞	0
F	5	0	∞	∞	0	∞

Bound (A,B)

	A	B	C	D	E	F	min
A	∞	∞	∞	∞	∞	∞	∞
B	∞	∞	0	∞	4	0	0
C	0	∞	∞	3	∞	∞	0
D	∞	∞	2	∞	0	∞	0
E	∞	∞	∞	0	∞	0	0
F	5	∞	∞	∞	0	∞	0
min	0	∞	0	0	0	0	

Bound(A,B) = cost(A,B) + cost(A) + Reduction

3 + 18 + 0

= 21

Bound (A,C)

	A	B	C	D	E	F	min
A	∞	∞	∞	∞	∞	∞	∞
B	0	∞	∞	∞	4	0	0
C	∞	0	∞	2	∞	∞	1
D	∞	∞	∞	∞	0	∞	0
E	∞	4	∞	0	∞	0	0
F	3	0	∞	∞	0	∞	0
Min	2	0	∞	0	0	0	

Bound(A,C) = Cost(A,C) + Cost(A) + Reduction

0 + 18 + 3

= 21

Bound (A,F)

	A	B	C	D	E	F	Min
A	∞	∞	∞	∞	∞	∞	∞
B	2	∞	0	∞	4	∞	0
C	0	1	∞	3	∞	∞	0
D	∞	∞	2	∞	0	∞	0
E	∞	4	∞	0	∞	∞	0
F	∞	0	∞	∞	0	∞	0
Min	0	0	0	0	0	∞	

Bound (A,F) = cost(A,F) + cost(A) + Reduction

6 + 18 + 2

= 24

Bound (A,B,C)

	A	B	C	D	E	F	
A	∞	∞	∞	∞	∞	∞	∞
B	∞	∞	∞	∞	∞	∞	∞
C	∞	∞	∞	0	∞	∞	3
D	∞	∞	∞	∞	0	∞	0
E	∞	∞	∞	0	∞	0	0
F	5	∞	∞	∞	0	∞	0
	5	∞	∞	0	0	0	

Bound(A,B,C) = cost(B,C) + cost(B) + Reduction

0 + 21 + 5 + 3

= 29

Bound (A,B,E)

	A	B	C	D	E	F	min
A	∞	∞	∞	∞	∞	∞	∞
B	∞	∞	∞	∞	∞	∞	∞
C	0	∞	∞	3	∞	∞	0
D	∞	∞	0	∞	∞	∞	2
E	∞	∞	∞	0	∞	0	0
F	0	∞	∞	∞	∞	∞	5
min	0	∞	0	0	∞	0	

Bound(A,B,E) = cost(B,E) + cost(B) + Reduction

4 + 21 + 7

= 32

Bound (A,B,F)

	A	B	C	D	E	F	min
A	∞	∞	∞	∞	∞	∞	∞
B	∞	∞	∞	∞	∞	∞	∞
C	0	∞	∞	3	∞	∞	0
D	∞	∞	2	∞	0	∞	0
E	∞	∞	∞	0	∞	∞	0
F	5	∞	∞	∞	0	∞	0
min	0	∞	2	0	0	∞	

Bound(A,B,F) = cost(B,F) + cost(B) + reduction

0 + 21 + 2

= 23

Bound (A,C,B)

	A	B	C	D	E	F	min
A	∞	∞	∞	∞	∞	∞	∞
B	∞	∞	∞	∞	4	0	0
C	∞	∞	∞	∞	∞	∞	∞
D	∞	∞	∞	∞	0	∞	0
E	∞	∞	∞	0	∞	0	0
F	3	∞	∞	∞	0	∞	0
min	3	∞	∞	0	0	0	

Bound(A,C,B) = cost(C,B) + cost(C) + Reduction

1 + 21 + 3

= 25

Bound (A,C,D)

	A	B	C	D	E	F	
A	∞	∞	∞	∞	∞	∞	∞
B	0	∞	∞	∞	4	0	0
C	∞	∞	∞	∞	∞	∞	∞
D	∞	∞	∞	∞	0	∞	0
E	∞	4	∞	∞	∞	0	0
F	3	0	∞	∞	0	∞	0
	0	0	∞	∞	0	0	

Bound(A,C,D) = cost(C,D) + cost(C) + Reduction

3 + 21 + 0

= 24

Bound (A,B,F,E)

	A	B	C	D	E	F	
A	∞	∞	∞	∞	∞	∞	∞
B	∞	∞	∞	∞	∞	∞	∞
C	0	∞	∞	3	∞	∞	0
D	∞	∞	0	∞	∞	∞	2
E	∞	∞	∞	0	∞	∞	0
F	∞	∞	∞	∞	∞	∞	∞
	0	∞	0	0	∞	∞	

Bound(A,B,F,E) = cost(F,E) + cost(F) + reduction

0 + 23 + 2

= 25

Bound (A,F,B)

	A	B	C	D	E	F	min
A	∞	∞	∞	∞	∞	∞	∞
B	3	∞	0	∞	4	∞	0
C	0	∞	∞	3	∞	∞	0
D	∞	∞	2	∞	0	∞	0
E	∞	∞	∞	0	∞	∞	0
F	∞	∞	∞	∞	∞	∞	∞
min	0	∞	0	0	0	∞	

Bound(A,F,B) = cost(F,B) + cost(F) + reduction

= 0 + 24 + 0

= 24

Bound (A,F,B,E)

	A	B	C	D	E	F	
A	∞	∞	∞	∞	∞	∞	∞
B	∞	∞	∞	∞	∞	∞	∞
C	0	∞	∞	3	∞	∞	0
D	∞	∞	0	∞	∞	∞	2
E	∞	∞	∞	0	∞	∞	0
F	∞	∞	∞	∞	∞	∞	∞
	0	∞	0	0	∞	∞	

Bound(A,F,B,E) = cost(B,E) + cost(B) + reduction

4 + 24 + 2

= 30

Bound(A,F,E)

	A	B	C	D	E	F	
A	∞	∞	∞	∞	∞	∞	∞
B	3	∞	0	∞	∞	∞	0
C	0	0	∞	3	∞	∞	0
D	∞	∞	0	∞	∞	∞	2
E	∞	4	∞	0	∞	∞	0
F	∞	∞	∞	∞	∞	∞	∞
	0	1	0	0	∞	∞	

Bound(A,F,E) = cost(F,E) + cost(F) + reduction

0 + 24 + 3

= 27

Bound (A,F,E,D)

	A	B	C	D	E	F	
A	∞	∞	∞	∞	∞	∞	∞
B	3	∞	0	∞	∞	∞	0
C	0	0	∞	∞	∞	∞	0
D	∞	∞	0	∞	∞	∞	0
E	∞	∞	∞	∞	∞	∞	∞
F	∞	∞	∞	∞	∞	∞	∞
	0	0	0	∞	∞	∞	

Bound(A,F,E,D) = cost(E,D) + cost(E) + reduction

0 + 27 + 0

= 27

matrix solutions: Trying out every solution that can transverse each node once

M[0][1] + M[1][2] + M[2][3] + M[3][4] + M[4][5] + M[5][0]

7 + 5 + 4 + 2 + 3 + 8

= 29

M[0][5] + M[5][1] + M[1][4] + M[4][3] + M[3][2] + M[2][0]

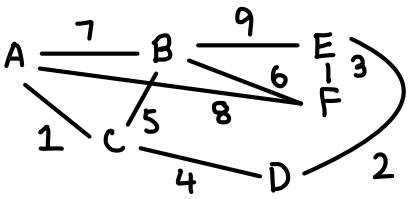
8 + 6 + 9 + 2 + 4 + 1

= 30

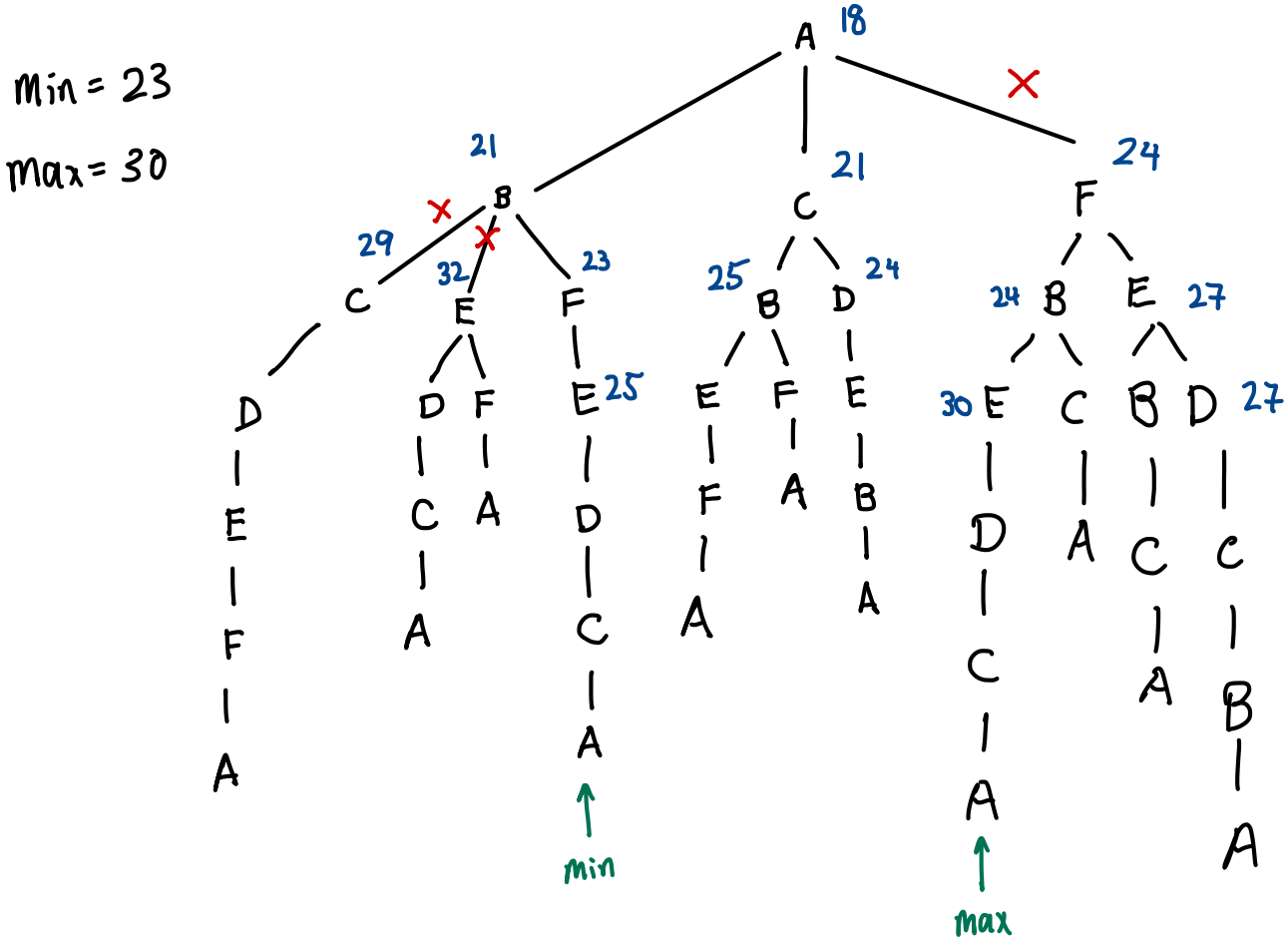
more next page

Your work starts here

1. (5 points) Create a state-space tree to design a backtracking algorithm to find all the solutions for the following matrix M. Note that approaches other than creating a state-space tree for this algorithm won't get credit:



	A	B	C	D	F	F
0 A	0	7	1	0	0	8
1 B	7	0	5	0	9	6
2 C	1	5	0	4	0	0
3 D	0	0	4	0	2	0
4 E	0	9	0	2	0	3
5 F	8	6	0	0	3	0



M[0][1]+M[1][5]+M[5][4]+M[4][3]+M[3][2]+M[2][0]
= 7 + 6 + 3 + 2 + 4 + 1 = 23

M[0][5]+ M[5][1] + M[1][4] + M[4][3] + M[3][2] + M[2][0]
= 8 + 6 + 9 + 2 + 4 + 1 = 30

2. (5 points) Create the pseudocode to solve all the instances of this problem based on your backtracking algorithm. Your pseudocode must be recursive, indented and organized into functions. Students writing code instead of pseudocode, or creating non recursive pseudocode won't get credit for this problem

```
Function main(matrix, Node, NodeNumber, Row , max, min):
    initialize: matrixLength = 6
    initialize: matrixWidth = 6
    initialize: matrix array of size N, Node
    if (Node Number +1 ) = N then
    Print: Node
    if ( Node = max OR Node = min) then
    Print: Node
Node <- n

foreach Node in range of Node Numbers +1 do
    if isSafe(matrix , Row , Node, column) then
        column[at index Row] <- Node
        main( matrix, Node, NodeNumber, Row+1)
    else
        column[index at Row] <- 0
    end if
End foreach

Function isSafe( matrix, Row, Node, column):
    foreach col in matrix do
        Connected <- matrix[row][column]
        if Vertex is Connected and Nodes in column[col] then
            return False
    end foreach
    return true

Function matrixSol( matrix , Row, column):
    if matrix[Row][column] != 0
        if isSafe(matrix , Row , Node, column) then
            if matrix i [at start] != 0
                return false
            else
                Print: matrix[i][j]
            if matrix i [at start] != 0
                return false
            else
                Print: matrix[i][j]
main(matrix, Node, NodeNumber-1, Row , max, min)
```

3. (5 points) Find the complexity and time complexity of your backtracking algorithm. Note that you MUST use the back-substitution method for this problem. Show ALL your work to get credit.

$$T(n) = T(n-1) + n^7 + 2n + 1$$
$$= [T(n-2) + (n-1) + (n-1)^7 + 2(n-1) + 1] + n^7 + 2n + 1$$
$$= [[T(n-3) + (n-2) + (n-2)^7 + 2(n-2) + 1] + n - 1 + (n-1)^7 + 2(n-1) + 1] + n^7 + 2n + 1$$
$$= T(n-3) + (n-2) + (n-2)^7 + 2(n-2) + 1 + (n-1) + (n-1)^7 + 2(n-1) + 1 + n^7 + 2n + 1$$

$$T(n) \overset{k \text{ steps}}{=} T(n-k) + (n-(k-1)) + n(1^7 + 2^7 + 3^7 + \ldots + (n-1) + n^7 + 2n + 1$$

$$T(n) = T(1) + [1^7 + 2(1) + 1] + [2^7 + 2(2) + 1] + [3^7 + 2(3) + 1] + \ldots + (n-1) + [n^7 + 2n + 1]$$
$$= \frac{n^7(n+1)}{2} = O(n^8)$$

4. (2 points extra credit and optional) Create a program, using your favorite programming language, that implements your algorithm. To get credit for this problem students must provide some unit tests to test that your program performs as per your algorithm specifications. Create this program in an online editor and share here the link to your program. Broken links or code that doesn't work as per your algorithm specifications won't get credit.