SAN FRANCISCO STATE UNIVERSITY
Computer Science Department

CSC510 – Analysis of Algorithms
Algorithm Challenge 5: Branch and Bound

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.

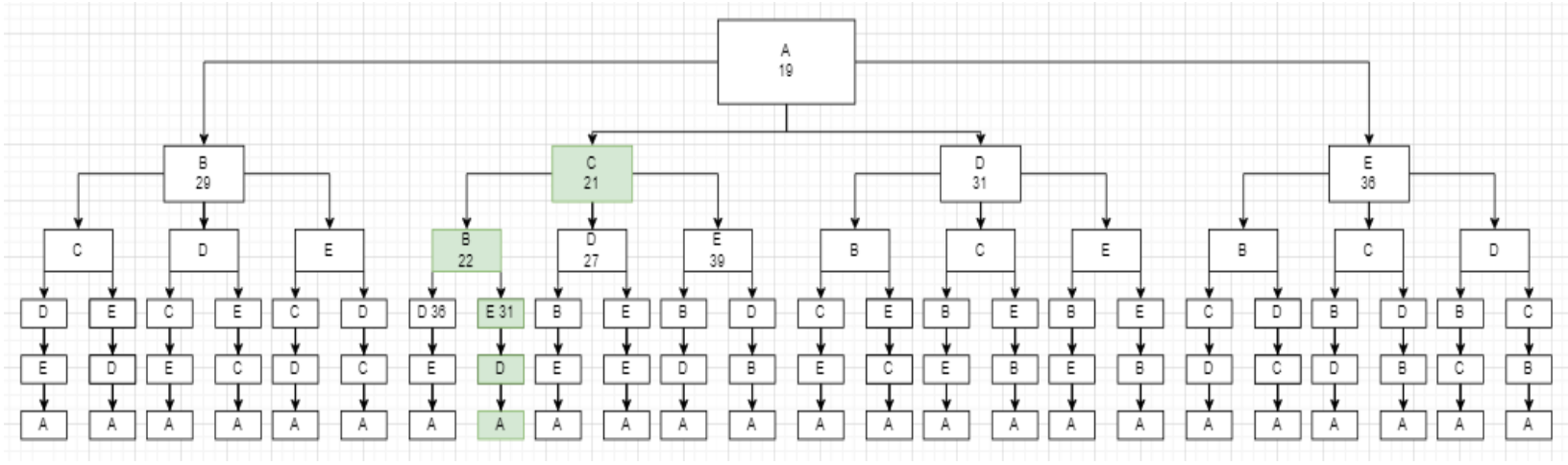
1. This algorithm challenge is worth 9%, and will be graded using a grading point scale where the maximum possible grade is 9 points
2. Handwriting work or screenshots of your work are not allowed. In addition, all the pseudocode done in this algorithm challenge must be done using LaTeX. Students who 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 3 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.

Your Work Starts Here

Given a a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the original city?

	A	B	C	D	E
A	0	14	4	10	20
B	14	0	7	8	7
C	4	5	0	7	16
D	11	7	9	0	2
E	18	7	17	4	0

1. Create a branch and bound algorithm to solve this problem **considering exit nodes first.**. Note that it needs to be done with a space- state tree to get credit



Now, let us discuss the Exit Nodes

A -> B
B -> min { C, D, E }
C -> min {D, E, A }
D -> min{ C, E, A }
E -> min { C, D, A }

A -> E = **20**
E -> {B, C, D } = **4**
B -> { C, D, E } = **7**
C -> { B, D, E } = **5**
D -> {B, C, E } = **2**

EQUALS = **31**

A -> D = **10**
D -> { B, C, E } = **2**
B -> { C, E, A } = **7**
C -> { B, E, A } = **4**
E -> { C B A } = **7**

EQUALS = **30**

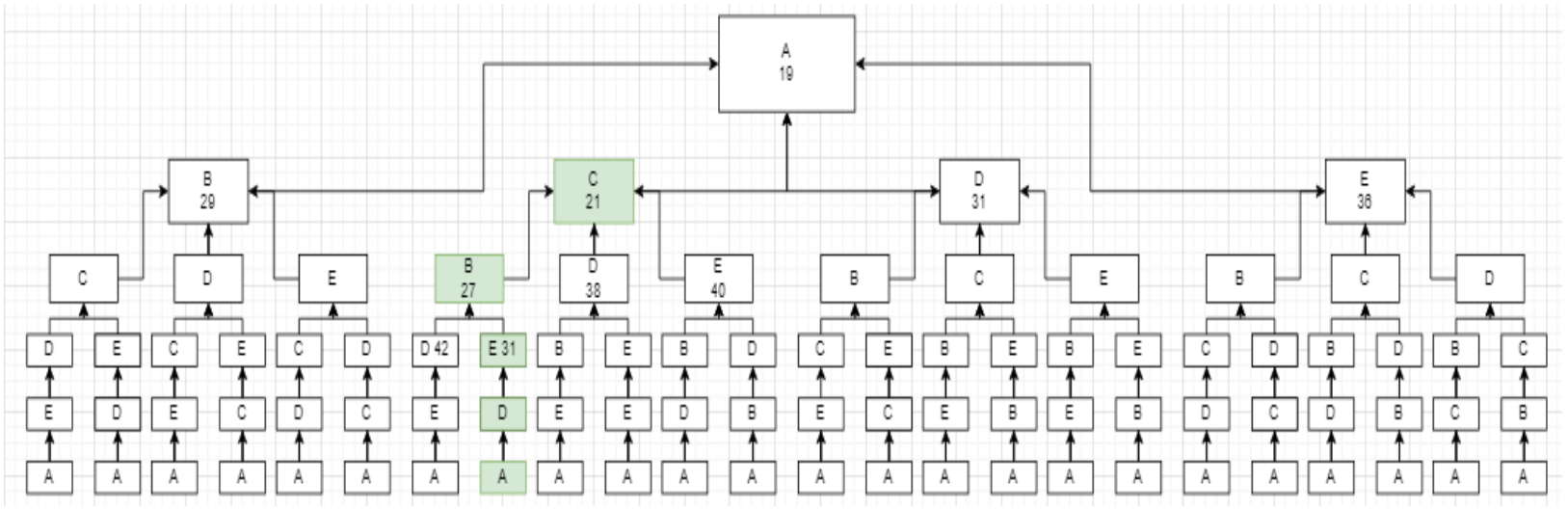
A -> B = **14**
B -> min { C, D, E } = **7**
C -> min { D, E, A } = **4**
D -> min { C, E, A } = **2**
E -> { C, D, A } = **4**

EQUALS = **31**

A -> C = **4**
C -> min { B, D, E} = **5**
B -> min { D, E, A } = **7**
D -> min { A, B, E } = **2**
E -> { A, D, B } = **4**

EQUALS = **22**

2.Create a branch and bound algorithm to solve this problem **considering entry nodes first.**. Note that it needs to be done with a space-state tree to get credit



ENTRY NODES

A <- B = **14**
A -> { C, D, E } = **4**
C -> { B, D, E } = **5**
D -> { C, B, E } = **2**
E -> { B, C, D } = **4**

EQUALS = **24**

A <- D = **11**
A -> { B, C, E } = **4**
B -> { D, C, E } = **7**
C -> { D, B, E } = **5**
E -> { D, B, C } = **4**

EQUALS = **31**

A <- C = **4**
A -> { B, D, E } = **4**
D -> { C, E, B } = **2**
B -> { C, D, E } = **7**
E -> { C, B, D } = **4**

EQUALS = **21**

A <- E = **18**
A -> { B, C, D } = **4**
C -> { B, D, E } = **5**
B -> { E, C, D } = **7**
D -> { E, B, C } = **2**

EQUALS = **34**

3. Write the pseudocode in LaTeX for both, exit and entry nodes first, and compute its time complexity (worst case) for your algorithm. Note that I am asking here for the time complexity and not the amount of work done by the algorithm. Show **ALL** your work to get credit.

```
Initialize: array NodesVisited [ citiesSub ]
            integer cost <- 0

function ExitENTRY ( cities, startPoint )
    1 <- NodesVisited
    if ( ( cities <- 2 ) AND ( m is not starting point )) then
        cost <- distance between start and m
        return cost
    else
        for ( j in n ) do
            for ( i in n AND NodesVisited is not 0 ) do
                if ( ( j is not equal i ) AND ( j is not equal m ) ) then
                    cost <- (ExitENTRY ( n - ( i ), j )
                end if
            end for
        end for
        return cost
    end if
end function
```

Let us now discuss the Time complexity of the function.
By looking at the code above, the algorithm we are using is recursive. With that being said, we are using a recursive approach that utilizes back substitution.

(n - 1) ! + n^2 + n2 + (n - 1) ! + n^2

2 (n - 1) ! + 4n^2

TIME COMPLEXITY: O(n!)