CS 312: Algorithm Analysis

Homework Assignment #18

Question 1 (4 points): Give the reduced cost matrix and the lower bound given the following distance matrix for the TSP problem.

∞	7	3	12
3	∞	6	14
5	8	∞	6
9	3	5	∞

Question 2 (6 points): Assume you want to solve the non-repetitive (0-1) knapsack problem (i.e. exactly 0 or 1 of each item can be in the final knapsack). You have *n* items each with a value and weight, and a knapsack capacity of *W*.

a) How many possible solutions are there?
b) Show a high level diagram of the general search tree.
c) Give at least one reasonable bounding function.

1.

$$\begin{bmatrix} \infty & 7 & 3 & 12 \\ 3 & \infty & 6 & 14 \\ 5 & 8 & \infty & 6 \\ 9 & 3 & 5 & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & 4 & 0 & 9 \\ 0 & \infty & 3 & 11 \\ 0 & 3 & \infty & 1 \\ 6 & 0 & 2 & \infty \end{bmatrix} \begin{matrix} -3 \\ -3 \\ -5 \\ -3 \end{matrix}$$

$14 + 1 =$ **15** min cost

$$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$ ← (subtract this from)

$$= \begin{bmatrix} \infty & 4 & 0 & 8 \\ 0 & \infty & 3 & 10 \\ 0 & 3 & \infty & 0 \\ 6 & 0 & 2 & \infty \end{bmatrix}$$ = final Matrix! reduced cost

---

2. a. 2^n solutions,

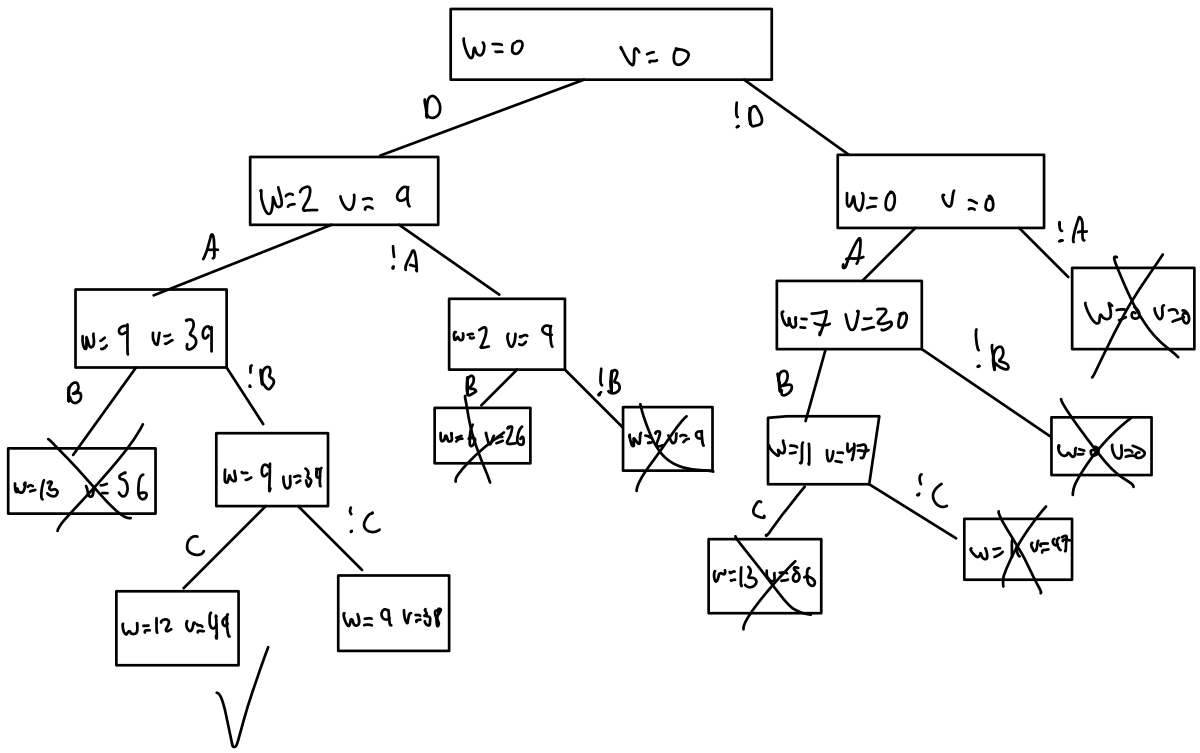
b.

$MAx = 12$

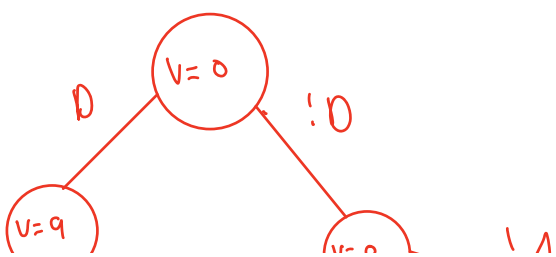
Item	W	V	Rate
P	2	9	4.5

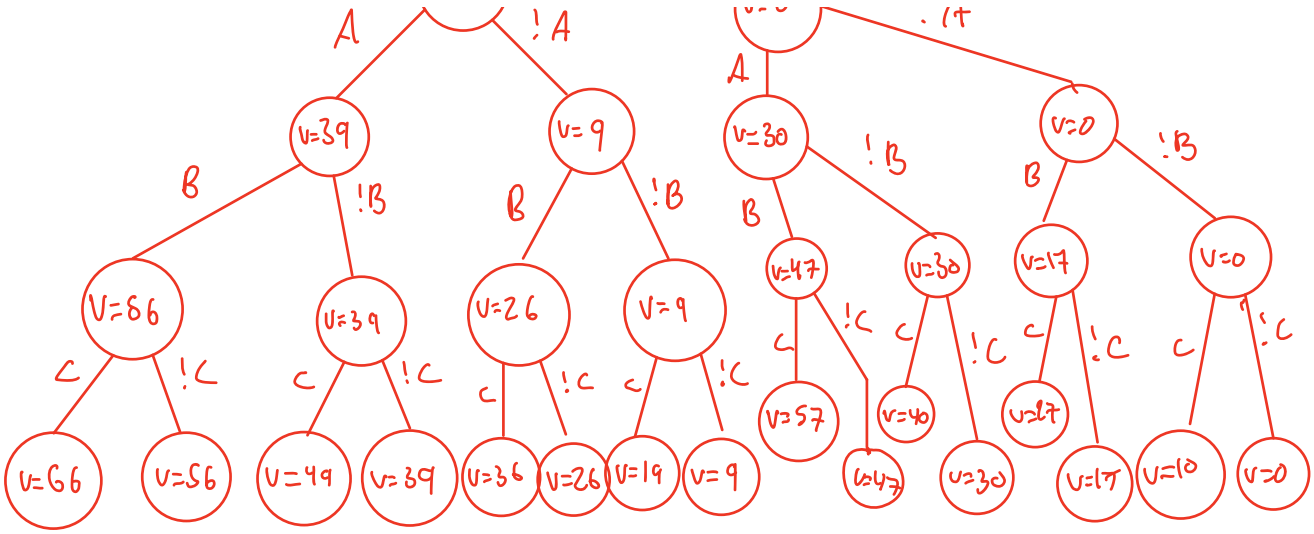
A	7	30	4.3
B	4	17	4.25
C	3	10	3.33

Best so far... ~~39~~ 49



(or this for b, I am confused on whether the search tree is looking for pruned edges or not)





C.

```
Branch(node):
    if node visited all other nodes:
        return cost-to-starting-node
    lower-bound = infinity
    min-node = node
    for every unvisited node:
        cost-to-node <- cost all unvisited node
        remove corresponding row and column from reduced matrix
        if reduced matrix needs to be reduces again
            reduce matrix again and
            get the min-cost after reducing
        new-lower-bound = node.cost + cost-to=node + min-cost
        current-unvisited-node.cost = new-lower-bound
        if new-lower-bound < lower-bound:
            lower-bound = new-lower-bound
            min-node = current-unvisited-node
    return lower-bound + branch(min-node)
```

*This algorithm does not account for the weight of a knapsack problem, if we need to account for the weight, we would add an additional check to see if the weight of an unvisited node would overflow the knapsack