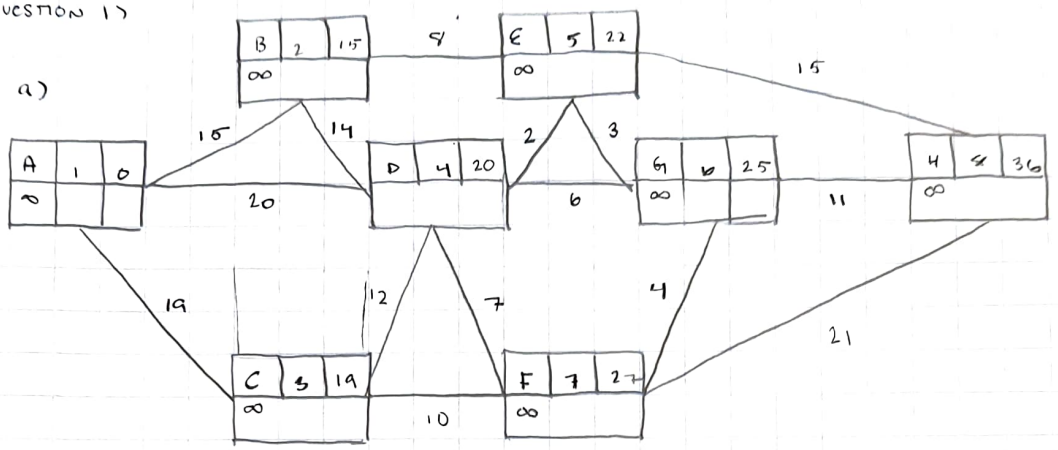
CSE 3500 Algorithms & Complexity

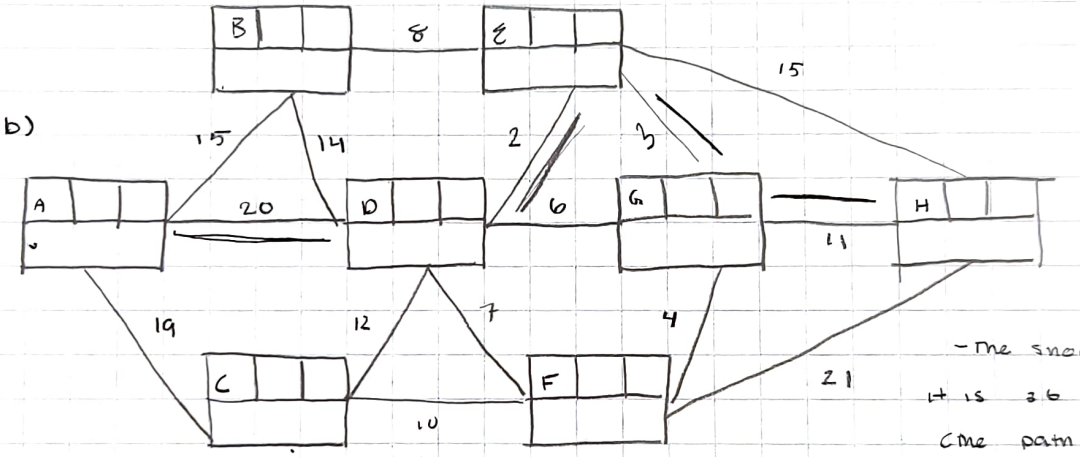
Homework #1

Question 1)

a)



b)

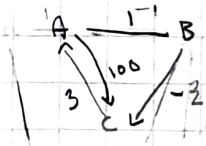


- The shortest path from A to H is 36 (the path is shown above)

Question 2)

The reason that the Dijkstra algorithm encounters issues &/or produces inaccurate results when negative edge weights are introduced is b/c the algorithm relies on the assumption that once a vertex's shortest path is found it will not change. This assumption holds true with pos weights b/c the path is never made shorter. (Unlike with neg weights). One of the main issues with neg weights within the algorithm is the possibility of creating "cycles of neg weights" which is a path that when traversed reduces the total path length. If such a cycle exists, the algorithm gets stuck in a loop, continuously reducing the path length with a final sol. The presence of the negative cycle disturbs the algorithms reliance of the shortest vertex found

for example if one were to use the Dijkstra's algorithm for this graph that contains neg weights



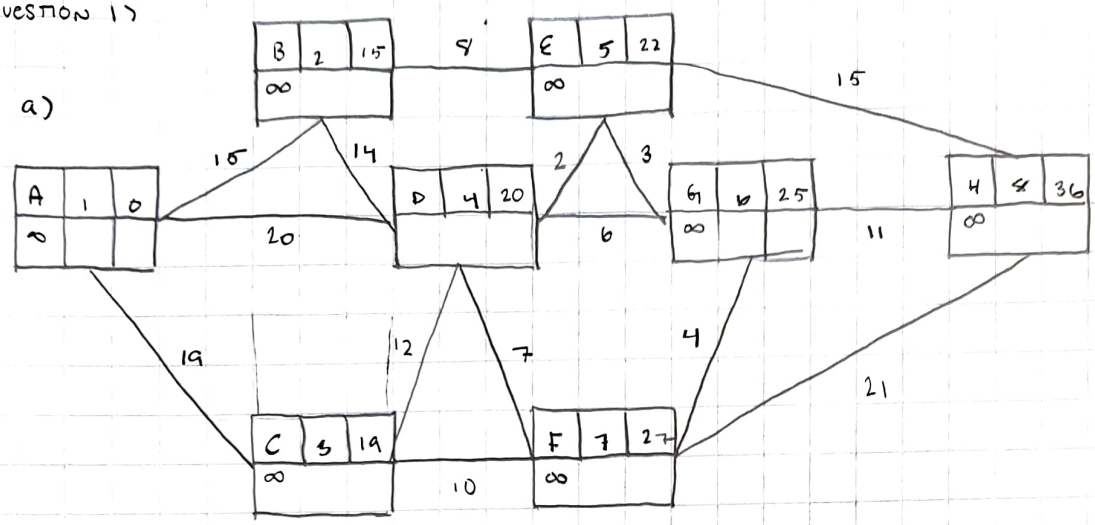
you would set A=0, vertex B=1, & C=∞, then the algorithm would update the distance to A to -2 visiting C. If the algorithm further visits (revisits) A through the edge C to A, it would further reduce the distance to A creating a neg cycle (A→C→A) this causes the basically infinite reduction of distance to A with it converging to the shortest path.

CSE 3500 Algorithms & Complexity

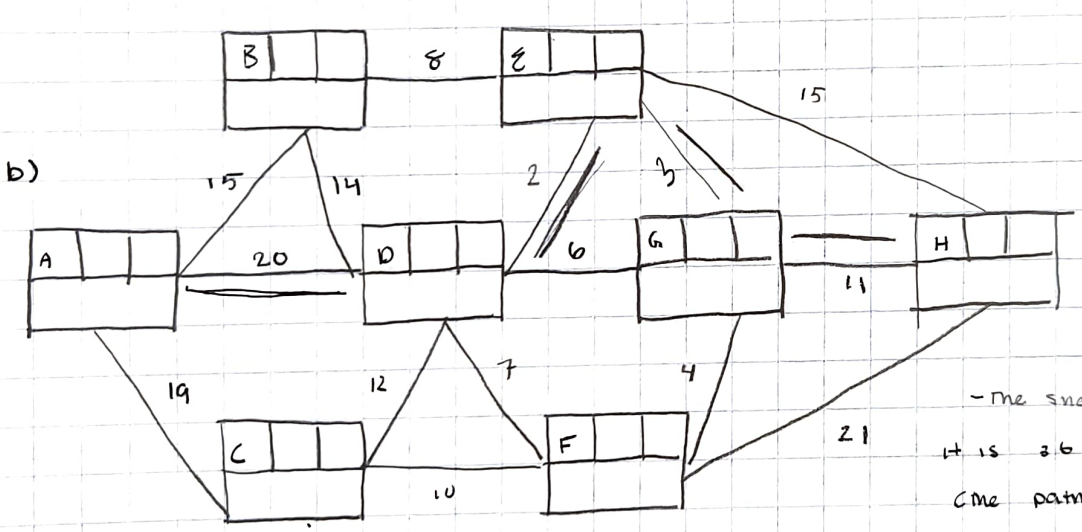
Homework 4

QUESTION 1)

a)



b)



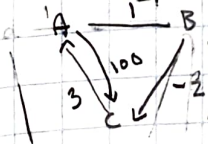
- The shortest path from A to H is 36

(the path is shown above)

QUESTION 2)

The reason that the Dijkstra algorithm encounters issues ?, or produces inaccurate results when negative edge weights are introduced is b/c the algorithm relies on the assumption that once a vertex's shortest path is found it will not change. This assumption hold true with pos weights b/c the path is never make shorter. (unlike with neg weights). One of the main issues with neg weights within the algorithm is the possibility of creating "cycles of neg weights" which is a path that when traversed reduces the total path length. If such a cycle exists, the algorithm gets stuck in a loop, continuously reducing the path length w/t a final sol. The presence of the negative cycle disrupts the algorithms reliance of the shortest vertex found

for example if one were to use the Dijkstra's algorithm for this graph that contains neg weights



you would set A=0, vertex B=1, & C=∞, then the algorithm would update the distance to A to -2 visiting C. If the algorithm further visits (revisits) A through the edge C to A, it would further reduce the distance to A creating a neg cycle (A→C→A) this causes the basically infinite reduction of distance to A w/t it converging to the shortest path.

QUESTION 3)

a) the first strategy may not always yield an optimal solution. this is b/c it doesn't consider the weight of the item in relation to its value. an item with a high value but also a high weight may not be the best choice if it prevents you from including other items that collectively have a higher total val

- example
  - Item 1: wt = 8 lbs, val = 10 , Item 2: wt = 2 lbs, val = 4

selecting item two rather than item 1 in this instance would be better & provide a higher val than item 1

b) the 2nd strategy may also not yield an optimal result b/c whilst you could get more items it does not consider value. For instance with (item 1: wt = 8 lbs, val = 10, Item 2 wt = 2 val = 4) using the 2nd strat, you would select the one with the lowest weight, however if you use a fractional value of item one you could have a higher val than item 2

c) By approaching the strategy, by focusing around the calculation of the ratio of val to wght will yield an optimal solution. this strategy ensures that when items are selected, you maintain the highest val, relative to weight, allowing one to maximize the total value w/t exceeding the limit

d) Def knapsack Alg (weight, value, limit):

```
n = len(val)
ratio[] = array of len(n)
#Ratio calculation:
for i in len(n-1)
    ratios[i] = values[i] / weights[i]

sort everything
total weight : Val = 0

For each item in sorted list
    - if adding the entire item does not exceed weight limit, add
the items weight & value to wt tot & tot+val
    - if adding the entire item exceeds wt limit, add item fractional
that is within weight limit whilst adjusting total wt & val

-return total val : max achievable val
```

e) The time complexity of the algorithm would be O(n log n)

f) the reason this algorithm provides the most optimal solution is b/c consistently selects the highest val to wt ratio, rather than prioritizing one over the other ensures that the algorithm maximizes the total val of items within the knapsack w/t exceeding the weight limit, this algorithm in conclusion guarentees the max possible tot val can be achieved with the wght constraint.