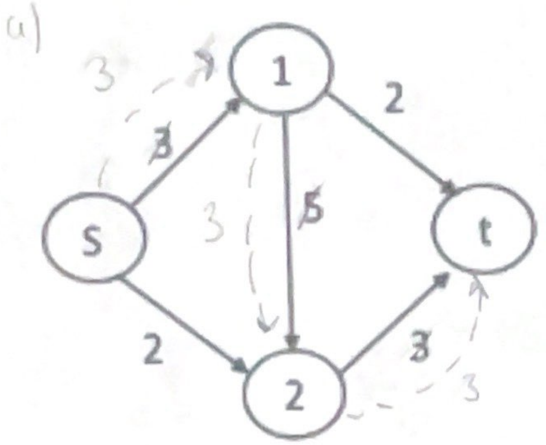
Grade: 5x20 = 100

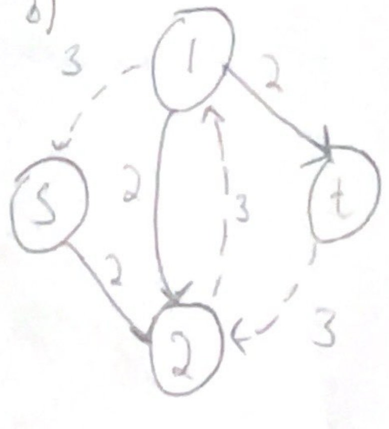
Name: Colin Henkel UCFID: 5485049

1. While learning Ford-Fulkerson algorithm we learned that the greedy approach of solving the network flow problem may not give the best maxflow for some cases.
   For example, the following network will give maxflow = 3 if you first choose augmenting path S-1-2-t.
   However, if you choose the augmenting paths in the following sequences, you will end up with a better maxflow = 5. [s-1-t], [s-2-t], and [s-1-2-t]

a)



b)



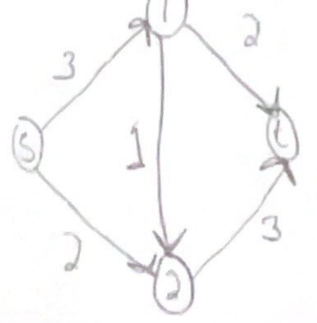
We have learned that residual graph can help us to overcome this issue as it helps us to automatically perform undo operation as part of the process. Note that the back edges of the residual graph are also used while finding the augmenting path.

As part of your answer, find the max flow for the above graph and fulfill the following requirements:

a) you must use [s-1-2-t] as your first augmenting path to see whether residual graph can help us to do undo and get the maxflow as we failed to get it without using the residual graph.
b) *You have to show the residual graph by updating the given graph during this process (use dotted line for this).*
c) *Draw the final flow graph that shows the final flow through this graph.*
d) *What is the value of the max flow?*

You can overwrite the above graph as part of your answer and use the space provided on the right side of the graph to show the final flow graph.

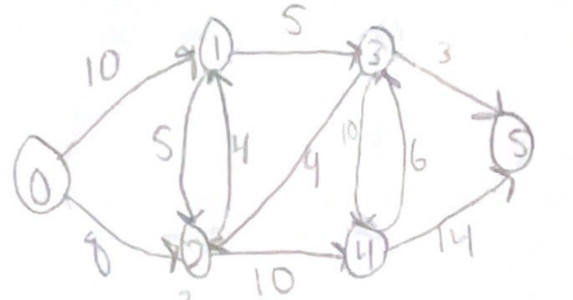
c, d)



maxflow = 5

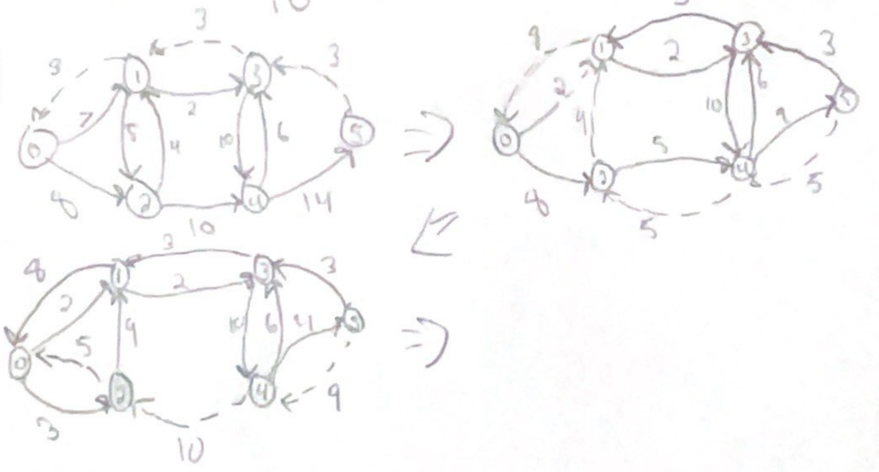
2. In this question, you will analyze the graph network shown in the uploaded code for network flow. See the network flow module's Main.java file. The code is thoroughly discussed in the recording on that module and also discussed briefly in the class.
   a) Draw the graph/initial residual graph based on the adjacency matrix.
   b) For each function call to isPathExist:
      a. Show the augmenting path returned by the function and write the bottleneck value.
      b. Update the graph based on the flow (use dotted line for the back edges)
      c. Update the maxflow
   c) Write the final maxflow based on the above steps.

a)



c) maxflow = 15

b)



3. Watch the uploaded recording on Longest common subsequence (LCS), recursive solution without dynamic programming and for the following two strings answer the following questions:

A = "ad" A= a d (0 1) B= a b c d (0 1 2 3)

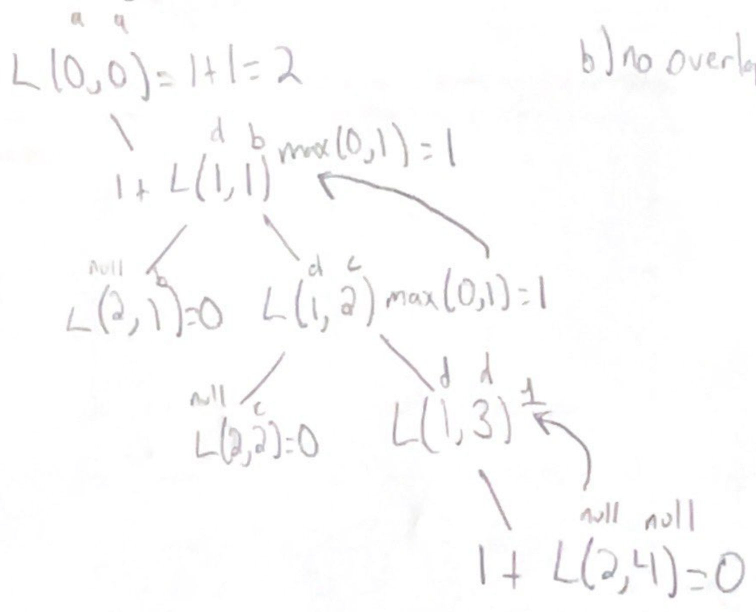
B = "abcd"

a) Clearly draw the recursion tree based on the pseudo code discussed in the recording. Use the exact same approach while drawing it and use multiple colors to clearly show the function call, the characters at those positions, what is being returned for a particular recursive call as like shown in the recording.

b) In the recursion tree, identify overlapping/repeated function calls and put star(*) on those function calls. List them below and write their count, i.e., how many ties they are being called.

c) Derive recurrence relation for the pseudo code discussed in the recording and solve this to find run-time using master theorem.

a)



b) no overlap as good input case

c) $T(n)=2T(n-1)+1$

$a=2,\ b=1,\ f(n)=1$, case 3 $a>1$

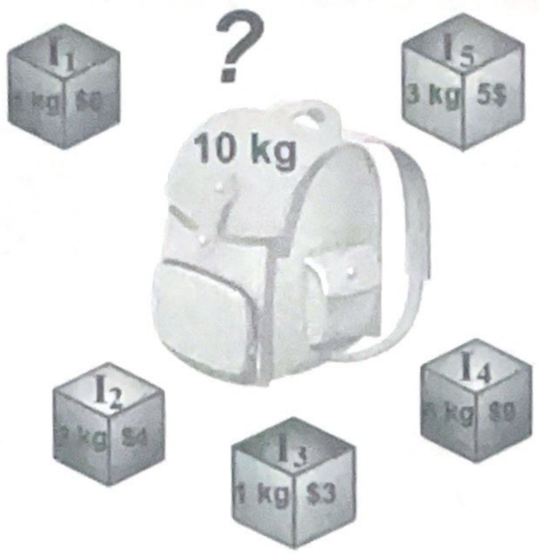
$T(n)=O(2^n \cdot 1)=O(2^n)$

4. As discussed in the lab. Answer the following questions related to 0/1 knapsack:

6 3 1
X ✓ 9+5+3=17

9 6
X 9+6=15

4 2 1 3
✓ 6+4+3+5=18



1) Fill out the dynamic programming table for the knapsack problem.
2) Trace back through the table to find the items in the knapsack.

item/weight	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	$\boxed{6}$	6	6	6	6	6	6
2	0	0	4	4	$\boxed{6}$	6	10	10	10	10	10
3	0	3	4	7	7	$\boxed{9}$	10	13	13	13	13
4	0	3	4	7	7	$\boxed{9}$	10	13	13	16	16
5	0	3	4	7	8	9	12	13	14	16	$\boxed{18}$

16 = 1,2,3,5
9 = 1,3
6 = 1

Items in Knapsack 1,2,3,5

Value of the Knapsack: 18

5. Derive the recurrence relation of Karatsuba's algorithm for large integer multiplication (divide and conquer approach) and use master theorem to derive the run-time.

```
Karatsuba(x, y):
  terminating conditions
  else{
    n= Math.max(x,y)
    half = n/2
    a= x / Math.pow(10, half)
    b= x % Math.pow(10, half)
    c= y / Math.pow(10, half)
    d= y / Math.pow(10, half)
    ac = Karatsuba(a, c)  T(n/2)
    bd= karatsuba(b, d)  T(n/2)
    ad_bc = Karatsuba(a+b, c+d) - ac - bd  T(n/2)
    return ac * Math.pow(10, 2*half) + ad_bc * Math.pow(10, half) + bd  θ(n)
```

Karatsuba:

$T(n) = 3T\left(\frac{n}{2}\right) + \theta(n)$

$a=3, b=2, f(n) = \theta(n), k=1$

$\log_2 3 > 1$, case 1

$T(n) = \theta(n^{\log_2 3})$

Master:

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$