Aashish Yadavally
Fall 2019

Homework II

CSCI 6470
September 30, 2019

Question 1

```
Input: a graph G = (V, E); 'm' edges; 'n' vertices
Output: "Yes" iff the vertices in V can be colored with the 3 colors

GRAPH-3-COLORING (V, E, m, n)
    array <- GET-INITIAL-COMBINATION(V, E, n)
    for each combination in combinations
        counter <- 0
        for each edge in E
            color_v1_edge = GET-COLOR(edge[0], combination)
            color_v2_edge = GET-COLOR(edge[1], combination)
            if color_v1_edge is equal to color_v2_edge
                break
            else
                counter <- counter + 1
        if counter is equal to m
            return "yes"
end function

GET-COMBINATIONS (n, array, i)
    combinations <- empty list
    if i <- n
        append array to combinations
        return
    array[i] <- 'B' // Label for color-1
    GET-COMBINATIONS(n, array, i + 1)
    array[i] <- 'R' // Label for color-2
    GET-COMBINATIONS(n, array, i + 1)
    array[i] <- 'Y' // Label for color-3
    GET-COMBINATIONS(n, array, i + 1)
end function
```

Time complexity upper bound for the algorithm:

$f(n) \leq (m) * 3^n$, where 'm' is number of edges

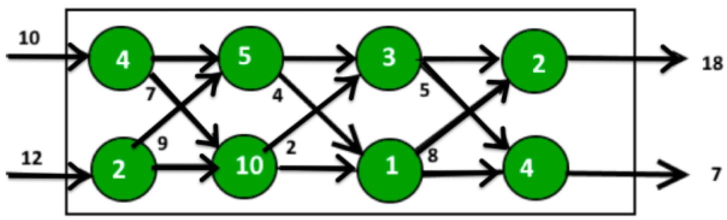
$f(n) = O(m.3^n)$

$\therefore$ Upper bound is given by $O(m.3^n)$

Time complexity lower bound for the algorithm:
Assuming the first combination given by GET-COMBINATIONS is the combination of color labels for the set of vertices in the graph-coloring algorithm, the lower bound would be $O(m)$

$\therefore$ A lower bound is given by $O(m)$

Question 2



The dynamic programming table for the above assembly line is:

j	1	2	3	4
f_1	14	19	22	24
f_2	14	24	24	28

$$\text{Minimum finish time} = min \begin{cases} 24 + 18 = 42 \\ 28 + 7 = 35 \end{cases}$$

$\implies$ Minimum finish time, $f* = 35$

Question 3

X	"6"	"6"	"6"	"2"	"4"	"5"	"1"
F	0.0833	0.013194	0.00209	0.00084	0.00013	2.115e-05	3.349e-06
L	0.25	0.1125	0.050625	0.00456	0.00041	3.69e-05	3.321e-06

It can be seen that $3.349e-06 > 3.321e-06$, i.e, $P(LLLFFFF) > P(LLLLLLL)$. Thus, most probable sequence of dice used is LLLFFFF.

Question 4

	X	0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
{w=3, v=70}	1	0	0	0	70	70	70	70
{w=3, v=80}	2	0	0	0	80	80	80	150
{w=4, v=100}	3	0	0	0	80	100	100	150
{w=2, v=55}	4	0	0	55	80	100	135	155

APPENDIX

Python script for Question 3

```
#!/usr/bin/env python3
def CASINO_DICE_DECODING(sequence: str, switch_F2L: float,
                         switch_L2F: float, p_F: list, p_L: list) -> None:
    """
    Implements algorithm for CASINO DICE DECODING PROBLEM.

    Arguments
    ---------
        sequence    : Input sequence of digits
        switch_F2L  : Fair to loaded dice switch probability
        switch_L2F  : Loaded to fair dice switch probability
        p_F         : Probabilities of faces of fair dice
        p_L         : Probabilities of faces of loaded dice
    """
    dice_faces = [str(i) for i in range(1, 7)]
    F_die_prob = dict(zip(dice_faces, p_F))
    L_die_prob = dict(zip(dice_faces, p_L))
    S_F = [F_die_prob[top] for top in sequence]
    S_L = [L_die_prob[top] for top in sequence]
    table = [[], []]
    for i in range(len(sequence)):
        if i == 0: # Base Case
            table[0].append(0.5 * S_F[0])
            table[1].append(0.5 * S_L[0])
        else:
            table[0].append(max(table[0][i-1] * (1 - switch_F2L) * S_F[i],
                                table[1][i-1] * (switch_L2F) * S_F[i]))
            table[1].append(max(table[1][i-1] * (1 - switch_L2F) * S_L[i],
                                table[0][i-1] * (switch_F2L) * S_L[i]))
        print(f'Iteration [{i}] ==========>')
        print(table)

if __name__ == "__main__":
    sequence = "6662451"
    switch_F2L = 0.05; switch_L2F = 0.1
    p_F = [1/6, 1/6, 1/6, 1/6, 1/6, 1/6]
    p_L = [1/10, 1/10, 1/10, 1/10, 1/10, 1/2]
    CASINO_DICE_DECODING(sequence, switch_F2L, switch_L2F, p_F, p_L)
```