Assignment 4

The code is available at: https://github.com/ErlendKK/DAT600-Algorithmic-Theory/tree/main/Assignment%204

Problem 1

Let

x = amount of Product X produced
y = amount of Product Y produced
m = houres of machine time
c = houres of craftsman time

Constraints:
m <= 40
c <= 35
0,25x + 0,33y = m
0,33x + 0,50y = c

0,25x + 0,33y <= 40;
0,33x + 0,50y <= 35;
x >= 10;
y >= 0;

Objective function:
Revenues = 200x + 300y
Costs = 100m + 20c

Maximize: z = Revenues – Costs
= 200x + 300y - 100m – 20c
= 200x + 300y – 100*(0,25x + 0,33y) – 20*(0,33x + 0,50y)
= 168,4x + 256,7y

Slackified problem formulation:

-168,4x - 256,7y + z = 0
0,25x + 0,33y + sm = 40
0,33x + 0,50y + sc = 35
x >= 10;
y >= 0; sm >= 0; sc >= 0;

Initial simplex tableau:

x	y	sm	sc	Z	C	
0,25	0,33	1	0	0	40	sm
0,33	0,5	0	1	0	35	sc
-168,4	-256,7	0	0	1	0	z

Programmatic solution.

The problem is solved in scipy.

```python
from scipy.optimize import linprog

'''
Objective function
Maximize: -168,4x - 256,7y + z = 0;
'''
c = [-168.4, -256.7]


'''
Subject to
0,25x + 0,33y <= 40;
0,33x + 0,50y <= 35;
x >= 10;
y >= 0;
'''
A_ub = [[0.25, 0.33], [0.33, 0.5]]
b_ub = [40, 35]
x_bounds = (10, None)
y_bounds = (0, None)

def solve_problem_one():
    result = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=[x_bounds, y_bounds], method='highs')
    print(f"Optimal verdi: {-result.fun}")
    print(f"x-verdier: {result.x}")

solve_problem_one()
```

Results:
Optimal verdi: 17958.78
x-verdier: [10. 63.4]

Problem 2: Maximum Flow

Problem 2 a)

let
G = (V, E) be a graph with a set V of vertices and set E of edges
s = the source vertex
t = the sink vertex
u and v = any pair of connected vertices
f(u,v) = the flow on the edge connecting u and v.
C(u, v) = the capacity (maximum flow) on the edge connecting u and v.
x(u,v) = a boolean representing included/ not included in the minimum cut

Objective function:
Minimize: z = sum(c(u,v) * x(u,v)) – (the sum of capacities for the edges crossing the cut)

Constraints:
f(u,v) <= c(u,v) for all u,v in V
f(u,v) >= 0 for all u,v in V

The problem is solved with puLP

```python
import pulp
import numpy as np

G = np.array([
    [0, 14, 25,  0,  0,  0,  0], # s
    [0,  0,  0, 21,  0,  0,  0], # V1
    [0,  0,  0,  3,  0,  7,  0], # V2
    [0,  6, 13,  0, 10,  5,  0], # V3
    [0,  0,  0,  0,  0,  0, 20], # V4
    [0,  0,  0,  0, 10,  0, 10], # V5
    [0,  0,  0,  0,  0,  0,  0]  # t
])


def solve_problem_two_a(G):
    num_nodes = G.shape[0]
    edges = [(i, j) for i in range(num_nodes) for j in range(num_nodes) if G[i, j] > 0]
    prob = pulp.LpProblem("MinimumCut", pulp.LpMinimize)

    # source set (1) or sink set (0)
    in_source_set = pulp.LpVariable.dicts("in_source_set", range(num_nodes), cat=pulp.LpBinary)
    prob += in_source_set[0] == 1
    prob += in_source_set[num_nodes - 1] == 0

    # included/excluded from the cut -> {1,0}
    x = pulp.LpVariable.dicts("x", edges, cat=pulp.LpBinary)


    # Objective Function:
    # Minimize the sum of the capacities of the cut edges

    prob += pulp.lpSum([G[i][j] * x[(i, j)] for (i, j) in edges])


    for (i, j) in edges:
        prob += x[(i, j)] >= in_source_set[i] - in_source_set[j]


    status = prob.solve(pulp.PULP_CBC_CMD(msg=False))
    cut_edges = [(i, j) for (i, j) in edges if pulp.value(x[(i, j)]) == 1]

    print(f"Problem 2a")
    print(f"Minimum Cut Value: {pulp.value(prob.objective)}")
    print(f"Edges in the cut: {cut_edges}\n")

solve_problem_two_a(G)
```

Result:
Minimum Cut Value: 22.0
Edges in the cut: [(2, 5), (3, 4), (3, 5)]

Problem 2 b)

Based on the textbook (29.25 – 29.28):

Objective function (29.25):
Maximize : sum(fsv – fsv) over the set of vertices sharing a boundary with s.

Subject to:

f(u,v) <= c(u,v) for all u,v in V
sum(f(u,v)) = sum(f(v,u)), for all u,v in V – {s, t}

f(u,v) >= 0 for all u,v in V

According to the textbook this can be rewritten so that it has O(V + E) constraints. I think the solution is to rewite the V^2 constrains in terms of edges:

Constraint	Size
f(u,v) <= c(u,v) for all (u,v) in E :	E
sum(f(u,v)) = sum(f(v,u)), for all u,v in V – {s, t}	V
f(u,v) >= 0 for all (u,v) in E	E

=> O(2E + V) = O(V + E)

The problem is solved in scipy.

```
from scipy.sparse import csr_matrix
from scipy.sparse.csgraph import maximum_flow

def solve_problem_two_b(G):
    graph = csr_matrix(G)
    source = 0
    sink = len(G) - 1
    result = maximum_flow(graph, source, sink)
    print(f"Problem 2b")
    print(f"Maximum flow: {result.flow_value}")

solve_problem_two_b(G)
```

Result: 22