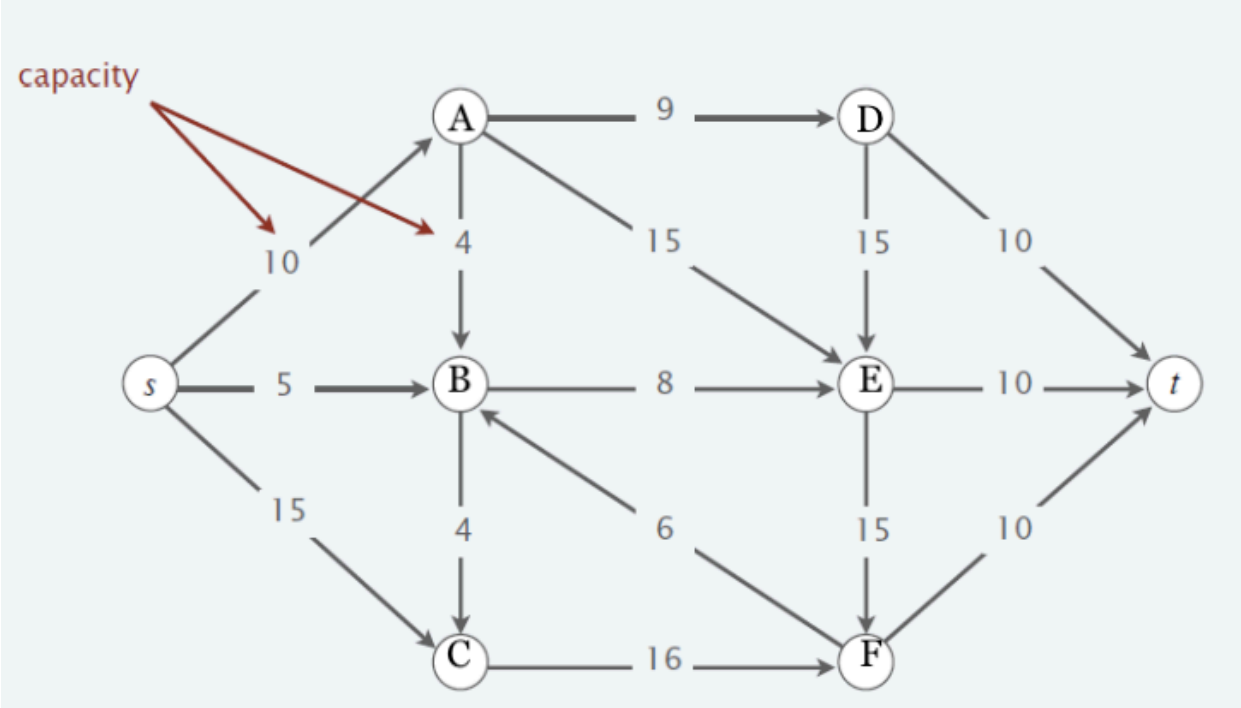
MATH 3172 Bonus Assignment

Coding the Max Flow/ Min Cut Problem in Python

By Jonghoon Park 216865313

April 30th, 2023

In this assignment, we try to solve a maximum flow problem. Below is the network we want to solve by the Ford Fulkerson Problem.



Let's begin by importing the necessary Python libraries. We use the library **networkx**, which is a Python package that will help us create the structure of the above network, as well as numpy, pandas, and matplotlib.

In [510]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
```

In [511]:

```
edges = pd.DataFrame({'from':['S', 'S', 'S', 'A', 'A', 'A', 'B', 'B',
                      'to':  ['A', 'B', 'C', 'D', 'E', 'B', 'E', 'C',
                    'weights':[10,  5,  15,   9,   15,  4,   8,   4,
                          })
#show edges and weights:
edges
```

Out[511]:

	from	to	weights
0	S	A	10
1	S	B	5
2	S	C	15
3	A	D	9
4	A	E	15
5	A	B	4
6	B	E	8
7	B	C	4
8	C	F	16
9	D	T	10
10	D	E	15
11	E	F	15
12	E	T	10
13	F	T	10
14	F	B	6

Next we call the DiGraph() function from the networkx library, which is a function that will define the edges with the weights in a directed graph.

We define a for loop to iterrate through the dataframe we defined as "edges" to extract the 0th, 1st, and 2nd value of each row, and assigns them as an edge to the graph.

In [512]:
```
graph = nx.DiGraph()
for i, row in edges.iterrows():
    graph.add_edge(row[0], row[1], weight=row[2])
# graph.edges(data=True)
graph.edges(data=True)
```

Out[512]:
```
OutEdgeDataView([('S', 'A', {'weight': 10}), ('S', 'B', {'weight': 5}
), ('S', 'C', {'weight': 15}), ('A', 'D', {'weight': 9}), ('A', 'E',
{'weight': 15}), ('A', 'B', {'weight': 4}), ('B', 'E', {'weight': 8})
, ('B', 'C', {'weight': 4}), ('C', 'F', {'weight': 16}), ('D', 'T', {
'weight': 10}), ('D', 'E', {'weight': 15}), ('E', 'F', {'weight': 15}
), ('E', 'T', {'weight': 10}), ('F', 'T', {'weight': 10}), ('F', 'B',
{'weight': 6})])
```

Here we have an more simplified view of the nodes and the edges.

In [513]:
```
graph.nodes
```

Out[513]:
```
NodeView(('S', 'A', 'B', 'C', 'D', 'E', 'F', 'T'))
```

In [514]:
```
graph.edges
```

Out[514]:
```
OutEdgeView([('S', 'A'), ('S', 'B'), ('S', 'C'), ('A', 'D'), ('A', 'E
'), ('A', 'B'), ('B', 'E'), ('B', 'C'), ('C', 'F'), ('D', 'T'), ('D',
'E'), ('E', 'F'), ('E', 'T'), ('F', 'T'), ('F', 'B')])
```

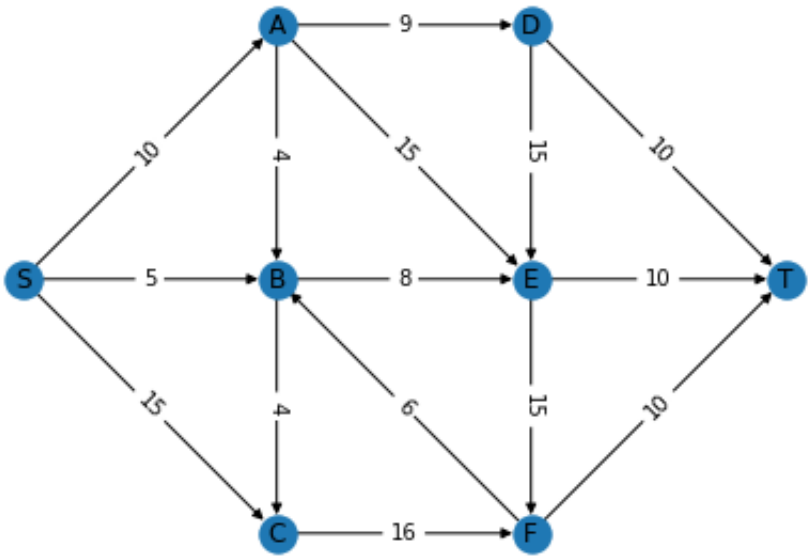
In [515]:
```
graph.add_node('S',pos=(0,1))
graph.add_node('A',pos=(1,2))
graph.add_node('B',pos=(1,1))
graph.add_node('C',pos=(1,0))
graph.add_node('D',pos=(2,2))
graph.add_node('E',pos=(2,1))
graph.add_node('F',pos=(2,0))
graph.add_node('T',pos=(3,1))
graph.nodes(data=True)
```

Out[515]:
```
NodeDataView({'S': {'pos': (0, 1)}, 'A': {'pos': (1, 2)}, 'B': {'pos'
: (1, 1)}, 'C': {'pos': (1, 0)}, 'D': {'pos': (2, 2)}, 'E': {'pos': (
2, 1)}, 'F': {'pos': (2, 0)}, 'T': {'pos': (3, 1)}})
```

Now, we want to assign the weights to the edges and and the positions of each nodes to visualize our directed graph.

the function get_node_attributes will extract the euclidean coordinates from the nodes, and the function get_edge_attributes will extract the weights from the edges.

In [516]:
```
weight = nx.get_edge_attributes(graph,'weight')
position = nx.get_node_attributes(graph,'pos')
nx.draw(graph,position, with_labels=True)
nx.draw_networkx_edge_labels(graph,position,edge_labels = weight)
plt.show()
```



Now, it is our goal to find the maximum flow between the starting node S and the ending node T. To solve this problem, we use Google's Operational Research Library called **ORTools**. This library contains a solver called the CP_Model, such that it can solve mathematical optimization problems such as integer problems subject to constraints.

More information can be found here:
https://developers.google.com/optimization/cp/cp_solver
(https://developers.google.com/optimization/cp/cp_solver)

In [517]:
```
pip install ortools
```

```
Requirement already satisfied: ortools in /opt/miniconda3/lib/python3
.9/site-packages (9.6.2534)
Requirement already satisfied: scipy>=1.10.0 in /opt/miniconda3/lib/p
ython3.9/site-packages (from ortools) (1.10.1)
Requirement already satisfied: numpy>=1.13.3 in /opt/miniconda3/lib/p
ython3.9/site-packages (from ortools) (1.23.1)
Requirement already satisfied: absl-py>=0.13 in /opt/miniconda3/lib/p
ython3.9/site-packages (from ortools) (1.4.0)
Requirement already satisfied: protobuf>=4.21.12 in /opt/miniconda3/l
ib/python3.9/site-packages (from ortools) (4.22.1)
Note: you may need to restart the kernel to use updated packages.
```

In [518]:
```
from ortools.sat.python import cp_model
flow_model = cp_model.CpModel()
#import the solver cp_model,and name our model flow_model
```

Now we must define the decision variables for the solver to use. First we define an empty dictionary called vars. Then we iterate over the list of edges in the pandas dataframe we defined as **edges** to assign the edge capacity as a decision variable in our objective function.

The function **NewIntVar** is pre-defined as a function that takes in the parameters (min, max, name). We set max to 152 since total edge weights add upto 152.

In [640]:
```
variables = {}
for edge in graph.edges:
    variables[edge[0], edge[1]] = flow_model.NewIntVar(0,152,'edge_%s_
    print('Decision variable: ', variables[edge[0], edge[1]],
          'for edge:', (edge[0], edge[1]))
```

```
Decision variable:  edge_S_A for edge: ('S', 'A')
Decision variable:  edge_S_B for edge: ('S', 'B')
Decision variable:  edge_S_C for edge: ('S', 'C')
Decision variable:  edge_A_D for edge: ('A', 'D')
Decision variable:  edge_A_E for edge: ('A', 'E')
Decision variable:  edge_A_B for edge: ('A', 'B')
Decision variable:  edge_B_E for edge: ('B', 'E')
Decision variable:  edge_B_C for edge: ('B', 'C')
Decision variable:  edge_C_F for edge: ('C', 'F')
Decision variable:  edge_D_T for edge: ('D', 'T')
Decision variable:  edge_D_E for edge: ('D', 'E')
Decision variable:  edge_E_F for edge: ('E', 'F')
Decision variable:  edge_E_T for edge: ('E', 'T')
Decision variable:  edge_F_T for edge: ('F', 'T')
Decision variable:  edge_F_B for edge: ('F', 'B')
```

In here we define the constraints for each of the entering and leaving edges to each nodes.

We define that for any nodes that are not entering nor exiting nodes (A, B, C, D, E, F), the entering capacity must equal the exiting capacity, such that:

$$\sum_{allnodes} x_{i,j} + \sum_{allnodes} x_{j,i} - flow = 0$$

And if they are entering or exiting nodes:

$$-\sum_{exitingnodes} x_{S,i} + flow = 0$$

$$-\sum_{exitingnodes} x_{i,T} + flow = 0$$

In [641]:
```
flow = flow_model.NewIntVar(0,152,'flow')

for node in graph.nodes:
    in_edges = graph.in_edges(node)
    out_edges = graph.out_edges(node)
    print('total constraint on node:', node)
    edge_sum = sum(variables[edge[0], edge[1]] for edge in in_edges) -
    if(node == 'S'):
        flow_model.Add(edge_sum == -flow)
        print(edge_sum == -flow)
    elif(node == 'T'):
        flow_model.Add(edge_sum == flow)
        print(edge_sum == flow)
    else:
        flow_model.Add(edge_sum == 0)
        print(edge_sum, '== 0')
```

```
total constraint on node: S
((-((edge_S_A + edge_S_B) + edge_S_C) + 0) + (1 * flow)) == 0
total constraint on node: A
(edge_S_A + -((edge_A_D + edge_A_E) + edge_A_B)) == 0
total constraint on node: B
(((edge_S_B + edge_A_B) + edge_F_B) + -(edge_B_E + edge_B_C)) == 0
total constraint on node: C
((edge_S_C + edge_B_C) + -edge_C_F) == 0
total constraint on node: D
(edge_A_D + -(edge_D_T + edge_D_E)) == 0
total constraint on node: E
(((edge_A_E + edge_B_E) + edge_D_E) + -(edge_E_F + edge_E_T)) == 0
total constraint on node: F
((edge_C_F + edge_E_F) + -(edge_F_T + edge_F_B)) == 0
total constraint on node: T
(((edge_D_T + edge_E_T) + edge_F_T) + -flow) == 0
```

Define integer bounds on the constraints:

In [642]:
```
for edge in graph.edges:
    print('Constraint on: ', edge)
    maxflow = graph.get_edge_data(*edge)['weight']
    flow_model.Add(variables[edge[0], edge[1]] <= maxflow)
    print(variables[edge[0], edge[1]] <= maxflow)
```

```
Constraint on:  ('S', 'A')
edge_S_A <= 10
Constraint on:  ('S', 'B')
edge_S_B <= 5
Constraint on:  ('S', 'C')
edge_S_C <= 15
Constraint on:  ('A', 'D')
edge_A_D <= 9
Constraint on:  ('A', 'E')
edge_A_E <= 15
Constraint on:  ('A', 'B')
edge_A_B <= 4
Constraint on:  ('B', 'E')
edge_B_E <= 8
Constraint on:  ('B', 'C')
edge_B_C <= 4
Constraint on:  ('C', 'F')
edge_C_F <= 16
Constraint on:  ('D', 'T')
edge_D_T <= 10
Constraint on:  ('D', 'E')
edge_D_E <= 15
Constraint on:  ('E', 'F')
edge_E_F <= 15
Constraint on:  ('E', 'T')
edge_E_T <= 10
Constraint on:  ('F', 'T')
edge_F_T <= 10
Constraint on:  ('F', 'B')
edge_F_B <= 6
```

Define the problem as maximization:

In [643]:
```
flow_model.Maximize(flow)
```

Now we solve the problemn. CpModel is a function for creating the maximization model with the decision variables and constraints, and the CPSolver actually solves the model.