运筹学期末大作业

统计1602

何琦榕

16271066

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pulp import *
import networkx as nx
import collections
import csv
import sys
import copy
import networkx as nx
import copy as cp
from collections import defaultdict
import bellmanford as bf
from networkx.algorithms.flow import network_simplex, min_cost_flow
%load_ext autoreload
%autoreload 2
%matplotlib inline
%config InlineBackend.figure_format='retina'
```

1. shortest-path problem

Algorithm for the Shortest-Path Problem

- $Objective\ of\ nth\ iteration$: Find the nth nearest node to the origin (to be repeated for n = 1, 2, . . . until the nth nearest node is the destination.

- $Input\ for\ nth\ iteration$: n - 1 nearest nodes to the origin (solved for at the previous iterations), including their shortest path and distance from the origin.(These nodes, plus the origin, will be called solved nodes; the others are unsolved nodes.)

- $Candidates\ for\ nth\ nearest\ node$: Each solved node that is directly connected by a link to one or more unsolved nodes provides one candidate—the unsolved node with the shortest connecting link to this solved node. (Ties provide additional candidates.)

- $Calculation\ of\ nth\ nearest\ node$: For each such solved node and its candidate, add the distance between them and the distance of the shortest path from the origin to this solved node. The candidate with the smallest such total distance is the nth nearest node (ties provide additional solved nodes), and its shortest path is the one generating this distance.

Example I

Amy has just purchased a new car for $22,000. The cost of maintaining a car during a year depends on its age at the beginning of the year:

Age of car (years)	**0**	**1**	**2**	**3**	**4**
Annual maintenance cost ($)	2,000	3,000	4,000	8,000	12,000

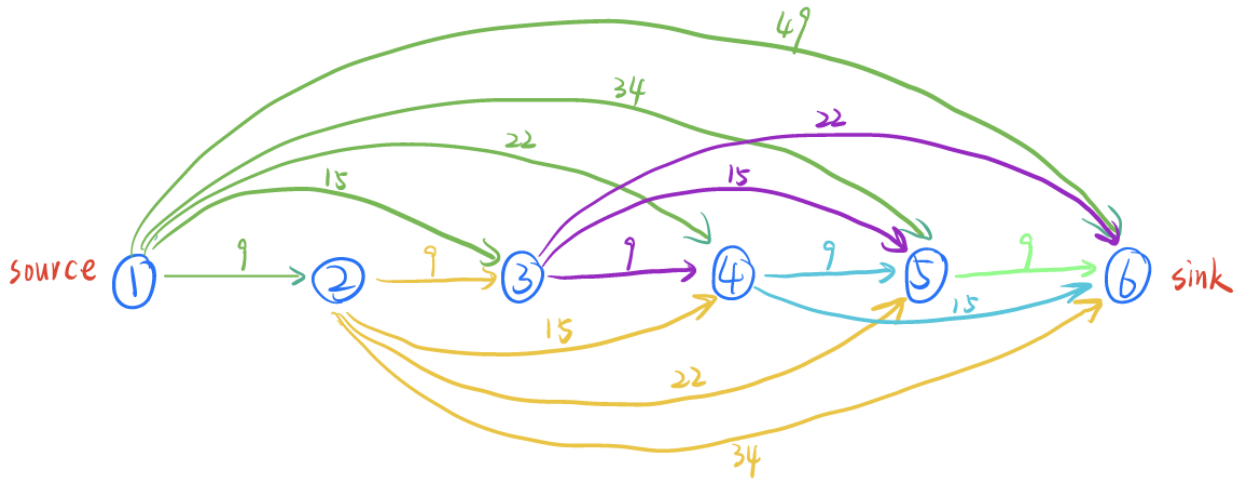
To avoid the high maintenance costs associated with an older car, Amy may trade in Amy's car and purchase a new car. The price Amy receiveed on a trade-in depends on the age of the car at the time of the trade-in:

Age of car (years)	**1**	**2**	**3**	**4**	**5**
Trade-in price ($)	15,000	12,000	9,000	5,000	2,000

For now, assume that at any time, it costs $22,000 to purchase a new car. Amy's goal is to minimize the net cost (purchasing costs + maintenance costs − money received in trade-ins) incurred over the next five years.

Age of car (years)	**1**	**2**	**3**	**4**	**5**
net cost ($)	9,000	15,000	22,000	34,000	49,000

- 1. Formulate this problem as a shortest path problem.
- 2. Solve the shortest path formulation: give the shortest path length and a shortest path.
- 3. Interpret the shortest path length and shortest path in the context of the problem.



mode i $\leftrightarrow$ beginming of year i
edge (i, j) $\longleftrightarrow$ purchase new machine at the beginming of year i and keep it watil year j
path from 1 to 6: each mode in the path $\longleftrightarrow$ when to buy a new machine
length of path $\longleftrightarrow$ total cost incurred over 5 years

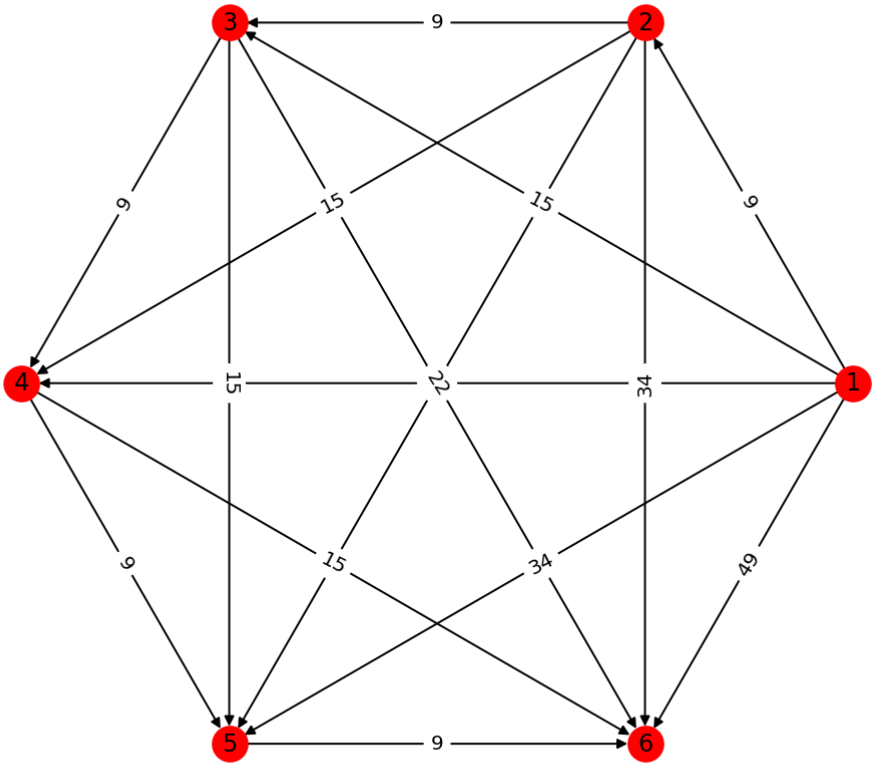
Our mission:

- obtain the length of a shortest path
- obtain the nodes/edges in a shortest path

Step1: Building a graph in networkx

In [2]:

```
# 创建空图
G = nx.DiGraph()
# 添加边
G.add_node(1,color='red')
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)
G.add_node(6,color='green')
# 添加边(1,2)长度为9
G.add_edge(1, 2, length=9)
G.add_edge(1, 3, length=15,color='r')
G.add_edge(1, 4, length=22)
G.add_edge(1, 5, length=34)
G.add_edge(1, 6, length=49)

# 添加从节点2出发的边
G.add_edge(2, 3, length=9)
G.add_edge(2, 4, length=15)
G.add_edge(2, 5, length=22)
G.add_edge(2, 6, length=34)

# 添加从节点3出发的边
G.add_edge(3, 4, length=9)
G.add_edge(3, 5, length=15)
G.add_edge(3, 6, length=22,color='r')

# 添加从节点4出发的边
G.add_edge(4, 5, length=9)
G.add_edge(4, 6, length=15)

# 添加从节点5出发的边
G.add_edge(5, 6, length=9)

fig, ax = plt.subplots(figsize = (10, 8))

nx.draw_networkx(
    G, ax=ax,
    pos=nx.shell_layout(G)
)
nx.draw_networkx_edge_labels(
    G, ax=ax, pos=nx.shell_layout(G),
    edge_labels=dict(((u, v), G[u][v]['length'])
                     for u, v in G.edges)
)

ax.set_aspect(1)
ax.set_axis_off()

plt.show()
```



Step2: Accessing edge information

- Two nodes are **adjacent** if they are endpoints of the same edge.
- We can find the nodes adjacent to node 3 by outgoing edges like this:

In [3]:
```
# Print nodes adjacent to node 2 by outgoing edges
print("nodes adjacent to node 2 by outgoing edges:\n",G[2])
print("\nlength of G2->G5:",G[2][5]["length"])
```

```
nodes adjacent to node 2 by outgoing edges:
 {3: {'length': 9}, 4: {'length': 15}, 5: {'length': 22}, 6: {'length': 3
4}}

length of G2->G5: 22
```

- We see that $(2, 3)$ is an edge in G with length 9, as desired.
- Same goes for $(2, 4)$, $(2, 5)$, and $(2, 6)$
- We can find the length of edge $(2, 5)$ directly, which is 22

Step3: To find the shortest path from node 1 to node 6 in the graph G

In [4]:
```
path_length, path_nodes, negative_cycle = bf.bellman_ford(G, source=1, targ
print(f"Is there a negative cycle? {negative_cycle}")
print(f"Shortest path length: {path_length}")
print(f"Shortest path: {path_nodes}")
```

```
Is there a negative cycle? False
Shortest path length: 37
Shortest path: [1, 3, 6]
```

Step4: Interpreting the results

- The length of the shortest path represents the minimum cost of owning and maintaining a car over the next 5 years, which in this case is 37,000 dollars.
- The nodes in the shortest path indicate when to buy a new car (and trade-in the old car). In this case, we should buy a new car in year 1 and year 3.

2.k-shortest pathes problem(K条最短路径问题)

K条最短路径算法：Yen's Algorithm

- 算法背景
  - K 最短路径问题是最短路径问题的扩展和变形。1959 年，霍夫曼(Hoffman) 和帕夫雷(Pavley)在论文中第一次提出k 最短路径问题。 k 最短路径问题通常包括两类：有限制的k 最短路问题和无限制的K 最短路问题。 前者要求最短路径集合不含有回路，而后者对所求得的最短路径集合无限制。

- 算法简介

- Yen's算法是Yen 在1971 年提出的以其名字命名 的Yen 算法。Yen's算法采用了递推法中的偏离路径算法思想，适用于非负权边的有向无环图结构。

- 算法思想
  - 算法可分为两部分，算出第1条最短路径P(1)，然后在此基础上依次依次算出其他的K-1条最短路径。在求P(i+1) 时，将P(i)上除了终止节点外的所有节点都视为偏离节点，并计算每个偏离节点到终止节点的最短路径，再与之前的P(i)上起始节点到偏离节点的路径拼接，构成候选路径，进而求得最短偏离路径。

- 算法实现：

In [5]:
```
def k_shortest_paths(G, source, target, k, weight = 'length'):
    # G is a networkx graph
    # source and target are the labels for the source and target of the pat
    # k is the amount of desired paths
    # weight = 'length' assumes a weighed graph.

    A = [nx.dijkstra_path(G, source, target, weight = 'length')]
    A_len = [sum([G[A[0][l]][A[0][l + 1]]['length'] for l in range(len(A[0]
    B = []

    for i in range(0, k):
        for j in range(0, len(A[-1]) - 1):
            Gcopy = cp.deepcopy(G)
            spurnode = A[-1][j]
            rootpath = A[-1][:j + 1]
            for path in A:
                if rootpath == path[0:j + 1]: #and len(path) > j?
                    if Gcopy.has_edge(path[j], path[j + 1]):
                        Gcopy.remove_edge(path[j], path[j + 1])
                    if Gcopy.has_edge(path[j + 1], path[j]):
                        Gcopy.remove_edge(path[j + 1], path[j])
            for n in rootpath:
                if n != spurnode:
                    Gcopy.remove_node(n)
            try:
                spurpath = nx.dijkstra_path(Gcopy, spurnode, target, weight
                totalpath = rootpath + spurpath[1:]
                if totalpath not in B:
                    B += [totalpath]
            except nx.NetworkXNoPath:
                continue
        if len(B) == 0:
            break
        lenB = [sum([G[path[l]][path[l + 1]]['length'] for l in range(len(p
        B = [p for _,p in sorted(zip(lenB, B))]
        A.append(B[0])
        A_len.append(sorted(lenB)[0])
        B.remove(B[0])
        print(f"{i+1} shortest path is {A[i]}, the length is {A_len[i]}\n")
```

We computed all the feasible paths for example1

In [6]:
```
k_shortest_paths(G, 1, 6, k = 20, weight = 'length')
```

```
1 shortest path is [1, 3, 6], the length is 37

2 shortest path is [1, 4, 6], the length is 37

3 shortest path is [1, 2, 4, 6], the length is 39

4 shortest path is [1, 3, 4, 6], the length is 39

5 shortest path is [1, 3, 5, 6], the length is 39

6 shortest path is [1, 2, 3, 6], the length is 40

7 shortest path is [1, 2, 5, 6], the length is 40

8 shortest path is [1, 4, 5, 6], the length is 40

9 shortest path is [1, 2, 3, 4, 6], the length is 42

10 shortest path is [1, 2, 3, 5, 6], the length is 42

11 shortest path is [1, 2, 4, 5, 6], the length is 42

12 shortest path is [1, 3, 4, 5, 6], the length is 42

13 shortest path is [1, 2, 6], the length is 43

14 shortest path is [1, 5, 6], the length is 43

15 shortest path is [1, 2, 3, 4, 5, 6], the length is 45
```

From the result above, we can see there are 15 paths in total.

3. The Maximum Flow Problem

In general terms, the maximum flow problem can be described as follows:

- 1. All flow through a directed and connected network originates at one node, called the source, and terminates at one other node, called the sink.
- 2. All the remaining nodes are transshipment nodes.
- 3. Flow through an arc is allowed only in the direction indicated by the arrowhead, where the maximum amount of flow is given by the capacity of that arc. At the source, all arcs point away from the node. At the sink, all arcs point into the node.
- 4. The objective is to maximize the total amount of flow from the source to the sink. This amount is measured in either of two equivalent ways, namely, either the amount leav- ing the source or the amount entering the sink.

Example II

Max Flow Application | 电影运输

本例的目的是研究寻找最大流量的应用。目标是设计和实现一种算法，用于在具有不同供需需求的节点之间运输物料。

现在，假设电影发行人希望将电影的副本从加州(CA)运送到其他每个州。 要注意由于州与周之间要满足相邻，所以我们省略了夏威夷州(HI)和阿拉斯加州(AK)。因此，有48个单位要从加州(CA)运出，并且每个其他州收到1个单位。

数据集usa.dat列出了美国的相邻州, 每行列出两个相邻的状态

我们假定每条边的运输能力都相同，为20个单位

In [7]:
```
G = nx.Graph()
usa = open('usa.dat')
for line in usa:
    s1, s2 = line.strip().split()
    G.add_edge(s1, s2)
for state in G.nodes():
    if state != 'CA':
        G.node[state]['demand'] = 1
G.node['CA']['demand'] = -48

G = nx.DiGraph(G)
uniform_capacity = 20
for (s1, s2) in G.edges():
    G.edges[s1, s2]['capacity'] = uniform_capacity
```

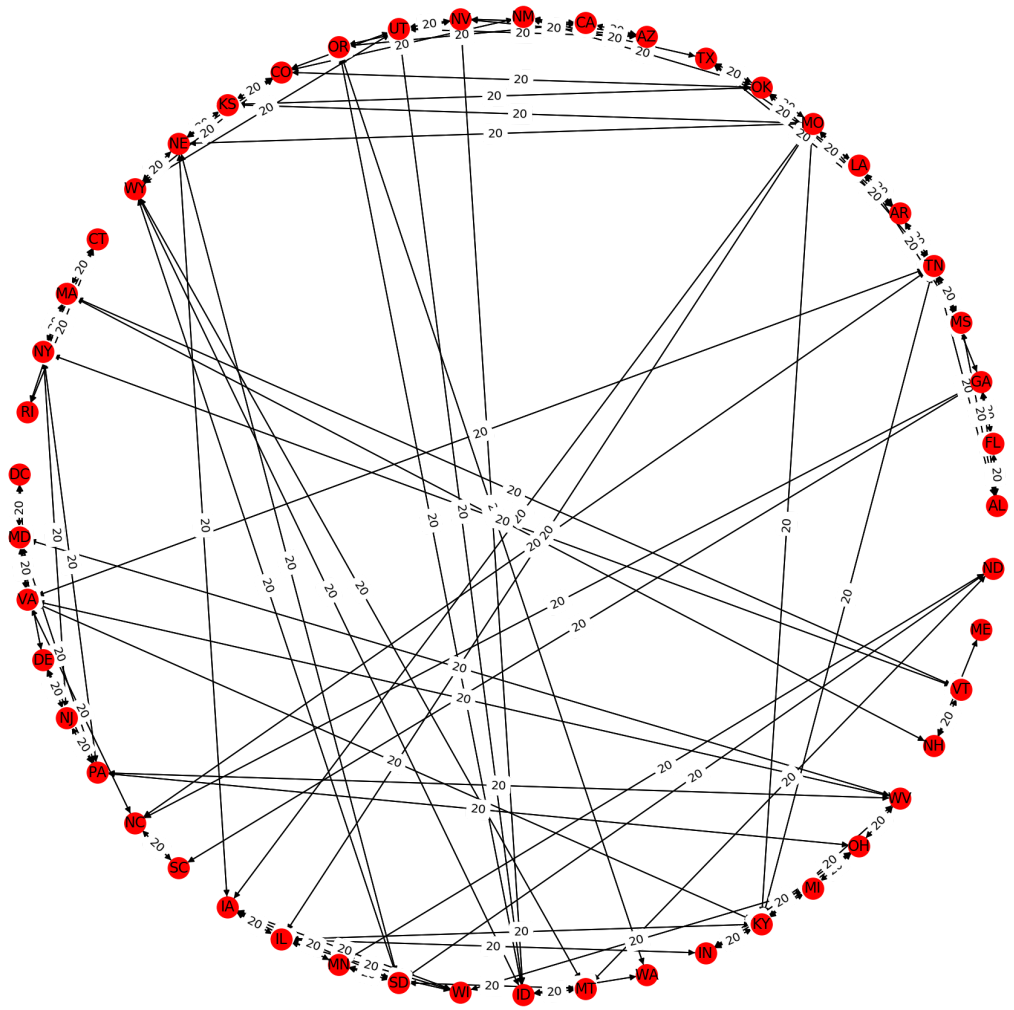
In [8]:
```
def flow_with_demands(graph):
    """Computes a flow with demands over the given graph"""
    # add super source node and super sink node
    graphNew=graph.copy()

    graphNew.add_node('s')
    graphNew.add_node('t')
    graphNew.node['s']['demand'] = 0
    graphNew.node['t']['demand'] = 0

    f=0

    # add adajcent edges and assign capacities
    for state in graphNew.nodes():
        d=graphNew.node[state]['demand']
        if d < 0:
            graphNew.add_edge('s',state)
            graphNew.edges['s',state]['capacity']=-d
            # compute the sum of demands
            f=f-d
        if d > 0:
            graphNew.add_edge(state,'t')
            graphNew.edges[state,'t']['capacity']=d

    flow_value, flow_dict = nx.maximum_flow(graphNew, 's', 't',capacity='ca

    del flow_dict['s']
    del flow_dict['t']

    for s1 in list(flow_dict):
        for s2 in list(flow_dict[s1]):
            if s2 =='t':
                del flow_dict[s1]['t']

    if flow_value == f:
        return(flow_dict)
    else:
        raise ValueError('NetworkXUnfeasible')
```

In [9]:
```
def divergence(flow):
    """Computes the total flow into each node according to the given flow d
    div=dict()
    for state in flow.keys():
        div[state]=0

    for s1 in div.keys():
        for s2 in flow[s1].keys():
            div[s1]=div[s1]-flow[s1][s2]
            div[s2]=div[s2]+flow[s1][s2]

    return(div)
```

结果展示

In [10]:

```
fig, ax = plt.subplots(figsize = (20, 18))

nx.draw_networkx(
    G, ax=ax,
    pos=nx.shell_layout(G)
)
nx.draw_networkx_edge_labels(
    G, ax=ax, pos=nx.shell_layout(G),
    edge_labels=dict(((u, v), G[u][v]['capacity'])
                     for u, v in G.edges)
)

ax.set_aspect(1)
ax.set_axis_off()

plt.show()
flow=flow_with_demands(G)
print(f'最大流为：{flow}\n')
div=divergence(flow)
flow_df=pd.DataFrame(flow)
print("在这里，由于篇幅限制，只展示了最大流前20个州的运输情况")
flow_df.iloc[:20,:20]
```



最大流为：{'AL': {'FL': 19, 'GA': 0, 'MS': 0, 'TN': 0}, 'FL': {'AL': 0, 'GA': 18}, 'GA': {'AL': 0, 'FL': 0, 'NC': 16, 'SC': 1, 'TN': 0}, 'MS': {'AL': 20, 'AR': 0, 'LA': 0, 'TN': 2}, 'TN': {'AL': 0, 'AR': 3, 'GA': 0, 'KY': 20, 'MO': 0, 'MS': 0, 'NC': 0, 'VA': 0}, 'AR': {'LA': 0, 'MO': 0, 'MS': 20, 'OK': 0, 'TN': 0, 'TX': 0}, 'LA': {'AR': 0, 'MS': 3, 'TX': 0}, 'MO': {'AR': 0, 'IA': 9, 'IL': 20, 'KS': 0, 'KY': 0, 'NE': 0, 'OK': 0, 'TN': 13}, 'OK': {'AR': 5, 'CO': 0, 'KS': 1, 'MO': 0, 'NM': 0, 'TX': 5}, 'TX': {'AR': 13, 'LA': 4, 'NM': 0, 'OK': 0}, 'AZ': {'CA': 0, 'NM': 20, 'NV': 0, 'UT': 0}, 'CA': {'AZ': 8, 'NV': 20, 'OR': 20}, 'NM': {'AZ': 0, 'CO': 0, 'OK': 6, 'TX': 13}, 'NV': {'AZ': 9, 'CA': 0, 'ID': 17, 'OR': 0, 'UT': 0}, 'UT': {'AZ': 4, 'CO': 0, 'ID': 0, 'NV': 0, 'WY': 0}, 'OR': {'CA': 0, 'ID': 11, 'NV': 7, 'WA': 1}, 'CO': {'KS': 20, 'NE': 0, 'NM': 0, 'OK': 6, 'UT': 0, 'WY': 0}, 'KS': {'CO': 0, 'MO': 20, 'NE': 0, 'OK': 0}, 'NE': {'CO': 7, 'IA': 0, 'KS': 0, 'MO': 13, 'SD': 0, 'WY': 0}, 'WY': {'CO': 20, 'ID': 0, 'MT': 0, 'NE': 0, 'SD': 0, 'UT': 0}, 'CT': {'MA': 6, 'NY': 0, 'RI': 0}, 'MA': {'CT': 0, 'NH': 4, 'NY': 1, 'RI': 2, 'VT': 0}, 'NY': {'CT': 6, 'MA': 0, 'NJ': 0, 'PA': 0, 'VT': 1}, 'RI': {'CT': 1, 'MA': 0}, 'DC': {'MD': 4, 'VA': 0}, 'MD': {'DC': 0, 'DE': 3, 'PA': 0, 'VA': 0, 'WV': 0}, 'VA': {'DC': 5, 'KY': 0, 'MD': 0, 'NC': 0, 'TN': 0, 'WV': 0}, 'DE': {'MD': 0, 'NJ': 2, 'PA': 0}, 'NJ': {'DE': 0, 'NY': 1, 'PA': 0}, 'PA': {'DE': 0, 'MD': 0, 'NJ': 0, 'NY': 6, 'OH': 0, 'WV': 1}, 'NC': {'GA': 0, 'SC': 0,

```
'TN': 9, 'VA': 6}, 'SC': {'GA': 0, 'NC': 0}, 'IA': {'IL': 0, 'MN': 0, 'M
O': 0, 'NE': 8, 'SD': 0, 'WI': 0}, 'IL': {'IA': 0, 'IN': 19, 'KY': 0, 'M
O': 0, 'WI': 0}, 'MN': {'IA': 0, 'ND': 15, 'SD': 0, 'WI': 0}, 'SD': {'I
A': 0, 'MN': 0, 'MT': 0, 'ND': 0, 'NE': 13, 'WY': 0}, 'WI': {'IA': 0, 'I
L': 0, 'MI': 0, 'MN': 16}, 'ID': {'MT': 2, 'NV': 0, 'OR': 0, 'UT': 5, 'W
A': 0, 'WY': 20}, 'MT': {'ID': 0, 'ND': 0, 'SD': 0, 'WY': 1}, 'WA': {'I
D': 0, 'OR': 0}, 'IN': {'IL': 0, 'KY': 0, 'MI': 18, 'OH': 0}, 'KY': {'I
L': 0, 'IN': 0, 'MO': 10, 'OH': 9, 'TN': 0, 'VA': 0, 'WV': 0}, 'MI': {'I
N': 0, 'OH': 0, 'WI': 17}, 'OH': {'IN': 0, 'KY': 0, 'MI': 0, 'PA': 8, 'W
V': 0}, 'WV': {'KY': 0, 'MD': 0, 'OH': 0, 'PA': 0, 'VA': 0}, 'NH': {'MA':
0, 'ME': 1, 'VT': 2}, 'VT': {'MA': 2, 'NH': 0, 'NY': 0}, 'ME': {'NH': 0},
'ND': {'MN': 0, 'MT': 0, 'SD': 14}}
```


在这里，由于篇幅限制，只展示了最大流前20个州的运输情况

Out[10]:

	AL	FL	GA	MS	TN	AR	LA	MO	OK	TX	AZ	CA	NM	NV	UT	OR	C
AL	NaN	0.0	0.0	20.0	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
AR	NaN	NaN	NaN	0.0	3.0	NaN	0.0	0.0	5.0	13.0	NaN	NaN	NaN	NaN	NaN	NaN	N
AZ	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	8.0	0.0	9.0	4.0	NaN	N
CA	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	0.0	NaN	0.0	N
CO	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	NaN	0.0	NaN	0.0	NaN	N
CT	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
DC	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
DE	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
FL	19.0	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
GA	0.0	18.0	NaN	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
IA	NaN	NaN	NaN	NaN	NaN	NaN	NaN	9.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
ID	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	17.0	0.0	11.0	N
IL	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
IN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
KS	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2(
KY	NaN	NaN	NaN	NaN	20.0	NaN	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
LA	NaN	NaN	NaN	0.0	NaN	0.0	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	N
MA	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
MD	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
ME	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

在这里NaN代表两边之间不直接相连，其余有数字（包括0）的边代表相连两点的运输量。

In [11]:
```
div
```

Out[11]:
```
{'AL': 1,
 'FL': 1,
 'GA': 1,
 'MS': 1,
 'TN': 1,
 'AR': 1,
 'LA': 1,
 'MO': 1,
 'OK': 1,
 'TX': 1,
 'AZ': 1,
 'CA': -48,
 'NM': 1,
 'NV': 1,
 'UT': 1,
 'OR': 1,
 'CO': 1,
 'KS': 1,
 'NE': 1,
 'WY': 1,
 'CT': 1,
 'MA': 1,
 'NY': 1,
 'RI': 1,
 'DC': 1,
 'MD': 1,
 'VA': 1,
 'DE': 1,
 'NJ': 1,
 'PA': 1,
 'NC': 1,
 'SC': 1,
 'IA': 1,
 'IL': 1,
 'MN': 1,
 'SD': 1,
 'WI': 1,
 'ID': 1,
 'MT': 1,
 'WA': 1,
 'IN': 1,
 'KY': 1,
 'MI': 1,
 'OH': 1,
 'WV': 1,
 'NH': 1,
 'VT': 1,
 'ME': 1,
 'ND': 1}
```

我们看到，供应点加州（CA）的所有电影都送出，而所有的需求点的需求都得到了满足。

4. The Minimun Cost Flow Problem

The minimum cost flow problem holds a central position among network optimization models, both because it encompasses such a broad class of applications and because it can be solved extremely efficiently. Like the maximum flow problem, it considers flow through a network with limited arc capacities. Like the shortest-path problem, it considers a cost (or distance) for flow through an arc. Like the transportation problem or assignment problem, it can consider multiple sources (supply nodes) and multiple destinations (demand nodes) for the flow, again with associated costs. In fact, all four of these previously studied problems are special cases of the minimum cost flow problem, as we will demonstrate shortly.

The reason that the minimum cost flow problem can be solved so efficiently is that it can be formulated as a linear programming problem so it can be solved by a streamlined version of the simplex method called the network simplex method. We describe this algorithm in the next section.

The minimum cost flow problem is described below:

- The network is a directed and connected network.
- At least one of the nodes is a supply node.
- At least one of the other nodes is a demand node.
- All the remaining nodes are transshipment nodes.
- Flow through an arc is allowed only in the direction indicated by the arrowhead, where the maximum amount of flow is given by the capacity of that arc. (If flow can occur in both directions, this would be represented by a pair of arcs pointing in opposite directions.)
- The network has enough arcs with sufficient capacity to enable all the flow generated at the supply nodes to reach all the demand nodes.
- The cost of the flow through each arc is proportional to the amount of that flow, where the cost per unit flow is known.
- The objective is to minimize the total cost of sending the available supply through the network to satisfy the given demand. (An alternative objective is to maximize the total profit from doing this.)

Consider a directed and connected network where the n nodes include at least one supply node and at least one demand node. The decision variables are:

$x_{ij} =$ flow through arc $i \rightarrow j$

and the given information includes

$c_{ij} =$ cost per unit flow through arc $i \rightarrow j$

$u_{ij} =$ arccapacity for arc $i \rightarrow j$

$b_i =$ net flow generated at node i

The value of b_i depends on the nature of node i, where

$b_i > 0$ if node i is a supply node,

$b_i < 0$ if node i is a demand node,

$b_i = 0$ if node i is a transshipment node.

The objective is to minimize the total cost of sending the available supply through the network to satisfy the given demand.

By using the convention that summations are taken only over existing arcs, the linear programming formulation of this problem is

$$Minimize: \sum_{(i,\, j) \in E} c_{ij}\, x_{ij}$$

Subject to:

1. $x_{ij} \leq u_{ij}$ for each $i,\ j \in V$
2. $\sum_{j \in V} x_{ji} - \sum_{j \in V} x_{ij} = b_i$ for each $i \in V$
3. $x_{ij} \geq 0$ for each $i,\ j \in V$

Feasible solutions property :A necessary condition for a minimum cost flow problem to have any feasible solutions is that

$$\sum_{i=1}^{n} b_i = 0$$

That is, the total flow being generated at the supply nodes equals the total flow being absorbed at the demand nodes. If the values of b_i provided for some application violate this condition, the usual interpretation is that either the supplies or the demands (whichever are in excess) actually represent upper bounds rather than exact amounts.

For many applications, b_i and u_{ij} will have integer values, and implementation will require that the flow quantities x_{ij} also be integer.

Integer solutions property : For minimum cost flow problems where every b_i and u_{ij} have integer values, all the basic variables in every basic feasible solution (including an optimal one) also have integer values.

Algorithm for Minimun Cost Flow Problem --- network simplex method

The network simplex method is a highly streamlined version of the simplex method for solving minimum cost flow problems. As such, it goes through the same basic steps at each iteration—finding the entering basic variable, determining the leaving basic variable, and solving for the new BF solution—in order to move from the current BF solution to a better adjacent one. However, it executes these steps in ways that exploit the special network structure of the problem without ever needing a simplex tableau.

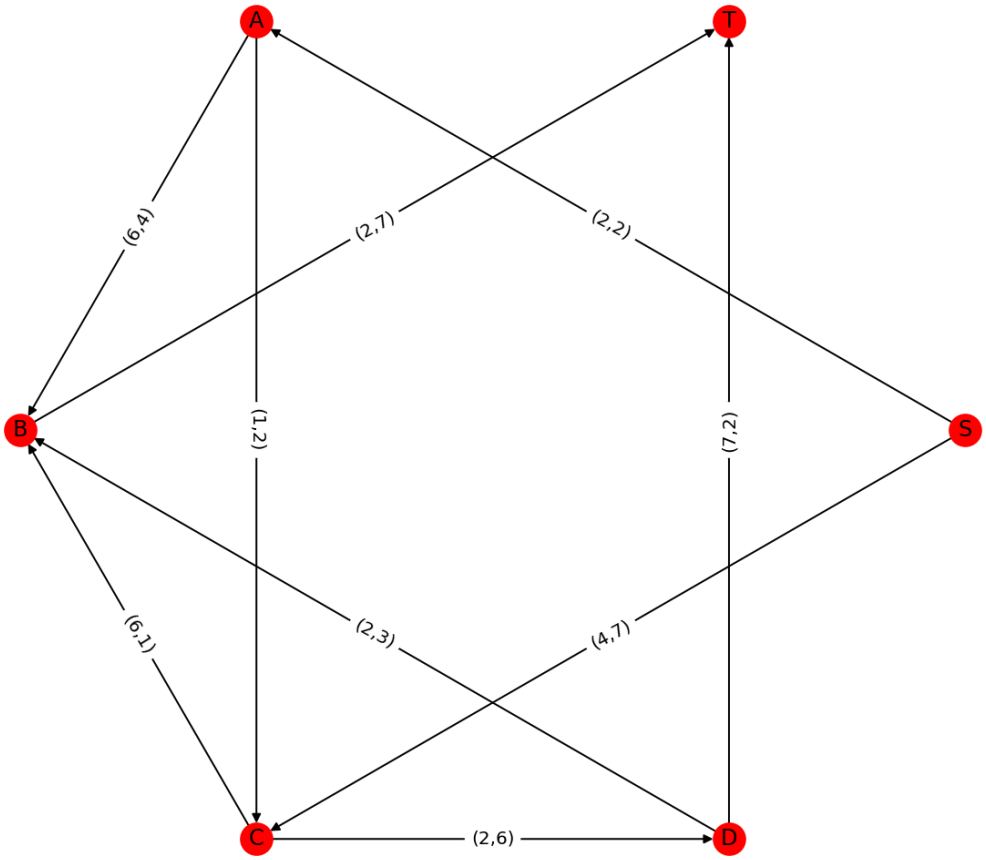
Example III

- Next, we are going to solve a Transshipment Problem.
  - Any minimum cost flow problem where each arc can carry any desired amount of flow is called a transshipment problem.

Build a network graph

In [12]:

```
import networkx as nx

# 创建有向图(Directed Graph)
G = nx.DiGraph()

# 加入顶点
G.add_node('S', demand=-5)
G.add_node('T', demand=5)
G.add_node('A')
G.add_node('B')
G.add_node('C')
G.add_node('D')

# 加入边
G.add_edge('S', 'A', capacity=2, weight=2)
G.add_edge('S', 'C', capacity=7, weight=4)
G.add_edge('A', 'B', capacity=4, weight=6)
G.add_edge('A', 'C', capacity=2, weight=1)
G.add_edge('C', 'B', capacity=1, weight=6)
G.add_edge('C', 'D', capacity=6, weight=2)
G.add_edge('D', 'B', capacity=3, weight=2)
G.add_edge('B', 'T', capacity=7, weight=2)
G.add_edge('D', 'T', capacity=2, weight=7)

# nx.nx_agraph.view_pygraphviz(G, prog='fdp')
flow_cost, flow_dict = nx.network_simplex(G)

fig, ax = plt.subplots(figsize = (12, 10))

nx.draw_networkx(
    G, ax=ax,
    pos=nx.shell_layout(G)
)
nx.draw_networkx_edge_labels(
    G, ax=ax, pos=nx.shell_layout(G),
    edge_labels=dict([((u, v,), "({},{})".format(
G[u][v]['weight'], G[u][v]['capacity']))for u, v, d in G.edges(data=True)])
)

ax.set_aspect(1)
ax.set_axis_off()

plt.show()
flow_df=pd.DataFrame(flow_dict)
```



As the graph above shows, we crested 6 nodes, one of them is supply node $\mathbf{s}$, and one of them is demand node $\mathbf{t}$, a,b,c,d are all transshipment nodes. The supply node supplys 5, and the demend node demands 5.

The texts on the arc is $(cost, capacity)$.

In [13]:
```
print(f"The minimum cost is {flow_cost}")
```

```
The minimum cost is 50
```

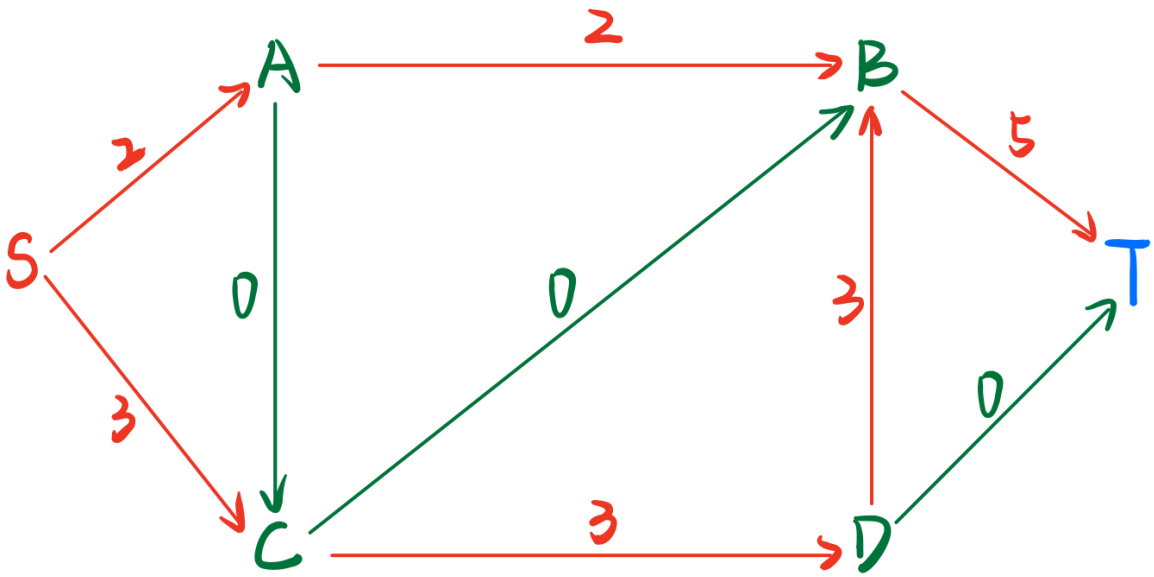
这样，我们就得到了这个网络的最小费用流的最小费用为 50，下面我们来看具体的最小费用流的网络

In [14]:
```
flow_df
```

Out[14]:

	S	T	A	B	C	D
A	2.0	NaN	NaN	NaN	NaN	NaN
B	NaN	NaN	2.0	NaN	0.0	3.0
C	3.0	NaN	0.0	NaN	NaN	NaN
D	NaN	NaN	NaN	NaN	3.0	NaN
T	NaN	NaN	NaN	5.0	NaN	0.0

Interpreting the results



最小费用流算法还可以用来解决最短路问题

要找到两个节点u和v之间的最短路径，给所有边无限的容量，给source节点1的需求为-1，给sink节点6的需求为1，然后运行网络单纯形。最小成本流的值将是u和v之间的距离，并且承载正流的边将指示路径。

下面我们展示如何用最小费用流算法解决我们的第一个例子$Example\ I$

In [15]:
```
G=nx.DiGraph()
G.add_weighted_edges_from([('1', '2' ,9), ('1' ,'3' ,15),('1', '4' ,22), ('
                           ('2' ,'3' ,9),('2', '4' ,15), ('2' ,'5' ,22), ('
                            ('3', '4' ,9), ('3' ,'5' ,15), ('3' ,'6' ,22),
                           ('4' ,'5' ,9), ('4' ,'6' ,15), ('5' ,'6' ,9)])
G.add_node('1', demand = -1)
G.add_node('6', demand = 1)
flowCost, flowDict = nx.network_simplex(G)
print(f"最短路径的长度为：{flowCost}")
```

```
最短路径的长度为：37
```

In [16]:
```
sorted([(u, v) for u in flowDict for v in flowDict[u] if flowDict[u][v] > 0
```

Out[16]:
```
[('1', '3'), ('3', '6')]
```

In [17]:
```
print(f"得到的最短路径为：{nx.shortest_path(G, '1', '6', weight = 'weight')}")
```

```
得到的最短路径为：['1', '3', '6']
```

可以看到，得出的结论与我们用最短路算法得出的结果是一致的。

5. Summary

In this project we explore the methods of solving Network Optimization Models problem and calculate them with python program.

The topic we discussed including $shortest-path\ problem$, $k-shortest\ pathes\ problem(Yen's\ Algorithm)$, $Maximum\ Flow\ Problem$, and $Minimun\ Cost\ Flow\ Problem$.

We introduced three examples.

- The first example is solved by three methods($shortest-path\ method$, $k-shortest\ pathes\ method(Yen's\ Algorithm)$, and $Minimun\ Cost\ Flow\ method$).
- The second example gives an application of maximum flow method.
- The third example is a min cost max flow problem solved by $Minimun\ Cost\ Flow\ simplex\ method$.

In [ ]:
```
```