SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Diplomski stručni studij Primijenjeno računarstvo

[GRAF ALGORITMI]: S E M I N A R

ChristmasTravel

Student: Anamarija Papić

Nositeljica predmeta: Ljiljana Despalatović, v. pred.

Split, 18. prosinca 2023.

1. Opis problema

Općeprihvaćena istina je da je film "Die Hard" najbolji božićni film svih vremena. I baš kao John McClane na početku filma, milijuni ljudi svake godine lete kući za Božić. Ovaj problem odnosi se na zračna putovanja. Yippie-ki-yay!

1. Opis problema

U Absurdistanu postoji N zračnih luka. Do sada je postojala samo jedna aviokompanija (vlasništvo Velikog Vizira) koja je pružala letove između svih zračnih luka. Međutim, Veliki Vizir primijetio je da aviokompanije ponekad imaju problema s popunjavanjem aviona, što Velikog Vizira stoji novca. Veliki Vizir unajmio je skupinu analitičara, i nakon nekoliko mjeseci otkrili su problem u osnovi: mogućnost letenja neparnog broja segmenata. Obične avionske karte u Absurdistanu ponekad se nazivaju hobbit kartama (kako idu "tamo i natrag"). Na primjer, ako osoba leti iz A preko B do C, možemo očekivati da će se kasnije vratiti iz C preko B do A, putujući ukupno četiri segmenta. Ako bi umjesto toga uzeli izravan let iz C u A, ukupan broj segmenata letova naših putnika sada bi bio neparan, a to je razlog zašto se svi avioni (svaki s parnim brojem sjedala) ne mogu popuniti točno. Ili, barem, to je ono što su analitičari rekli Velikom Viziru, prije nego što su uzeli svoju plaću i brzo napustili zemlju.

1. Opis problema

Kako god bilo, Veliki Vizir odlučio je potpuno riješiti ovaj problem. Njegova odluka bila je sljedeća:

- Od ovog trenutka će postojati A zasebnih aviokompanija u Absurdistanu. (Svaka od njih naravno vlasništvo Velikog Vizira.)
- Za svaki par zračnih luka, točno jedna od tih aviokompanija mora letjeti izravne letove između njih, u oba smjera.
- Svaka aviokompanija mora zapravo izvršiti neke letove.
- Karta se mora kupiti s određenom aviokompanijom. Karta mora sadržavati niz letova koje nudi ta aviokompanija. Letovi moraju biti uzastopni (svaki počinje tamo gdje je prethodni završio), a posljednji let mora vratiti putnika na zračnu luku gdje je karta započela.
- Ne smije biti moguće kupiti kartu s neparnim brojem letova.

Saznajte može li se plan Velikog Vizira provesti. Ako ne može, vratite prazan niz String[]. Ako može, vratite String[] s N elemenata, svaki sadrži N znakova: '-' na glavnoj dijagonali i jedno od prvih A velikih slova engleske abecede svugdje drugdje. Za svaki različiti i j, znakovi na [i][j] i [j][i] u povratnoj vrijednosti moraju biti jednaki i predstavljati aviokompaniju koja leti između zračnih luka i i j. Bilo koji važeći odgovor bit će prihvaćen.

2. Definicija i ograničenja

Definition

```
Class:              ChristmasTravel
Method:             plan
Parameters:         int, int
Returns:            String[]
Method signature:String[] plan(int N, int A)
(be sure your method is public)
```

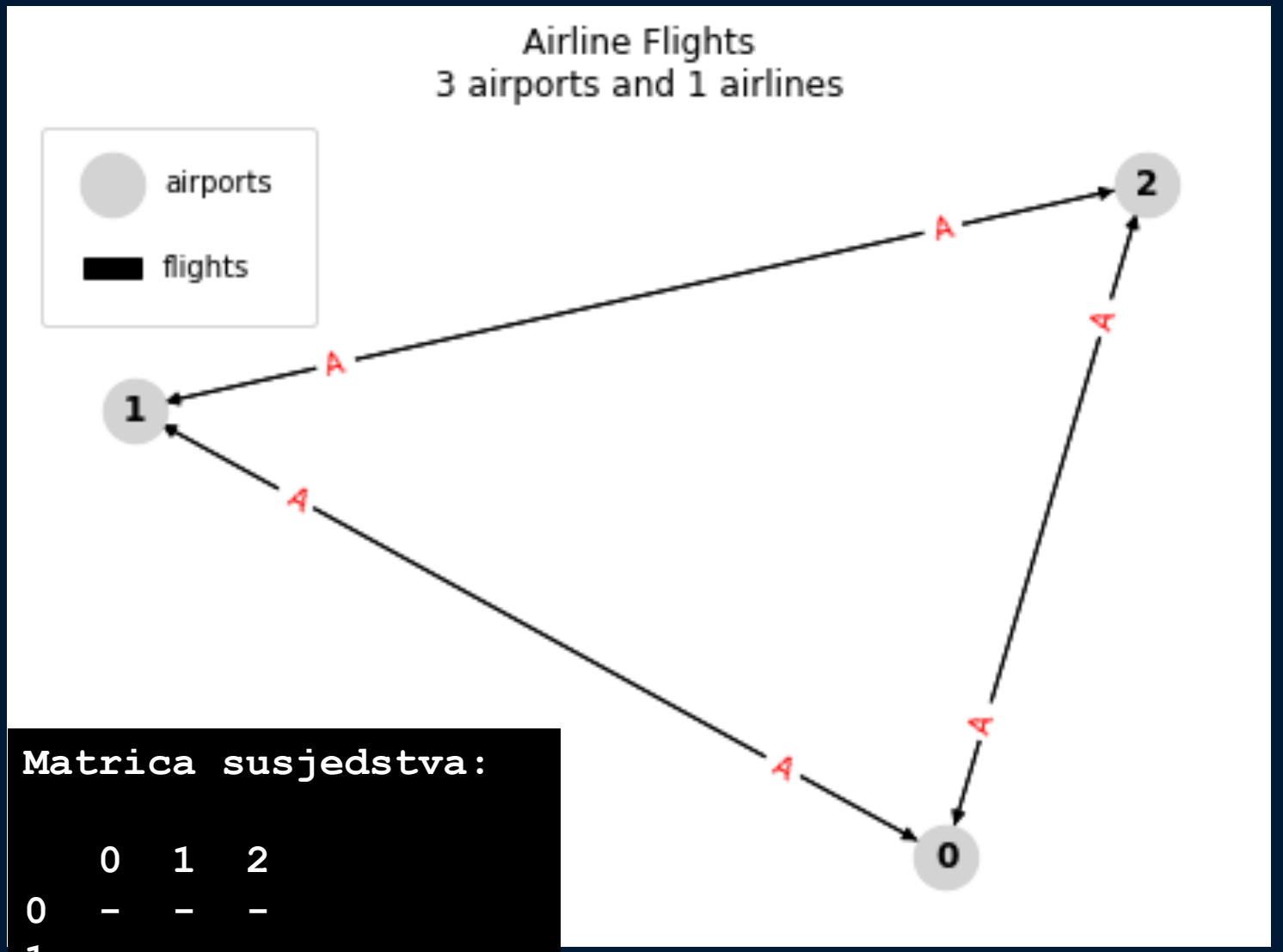
Constraints

- **N** will be between 1 and 100, inclusive.
- **A** will be between 1 and 26, inclusive.

3. Primjeri

0)

N = 3
A = 1
Returns: { }
There is no solution. If we had three airports and a single airline, people could buy a ticket with an odd number of flights. An example of such a ticket is a ticket with the flights 2 -> 0 -> 1 -> 2.



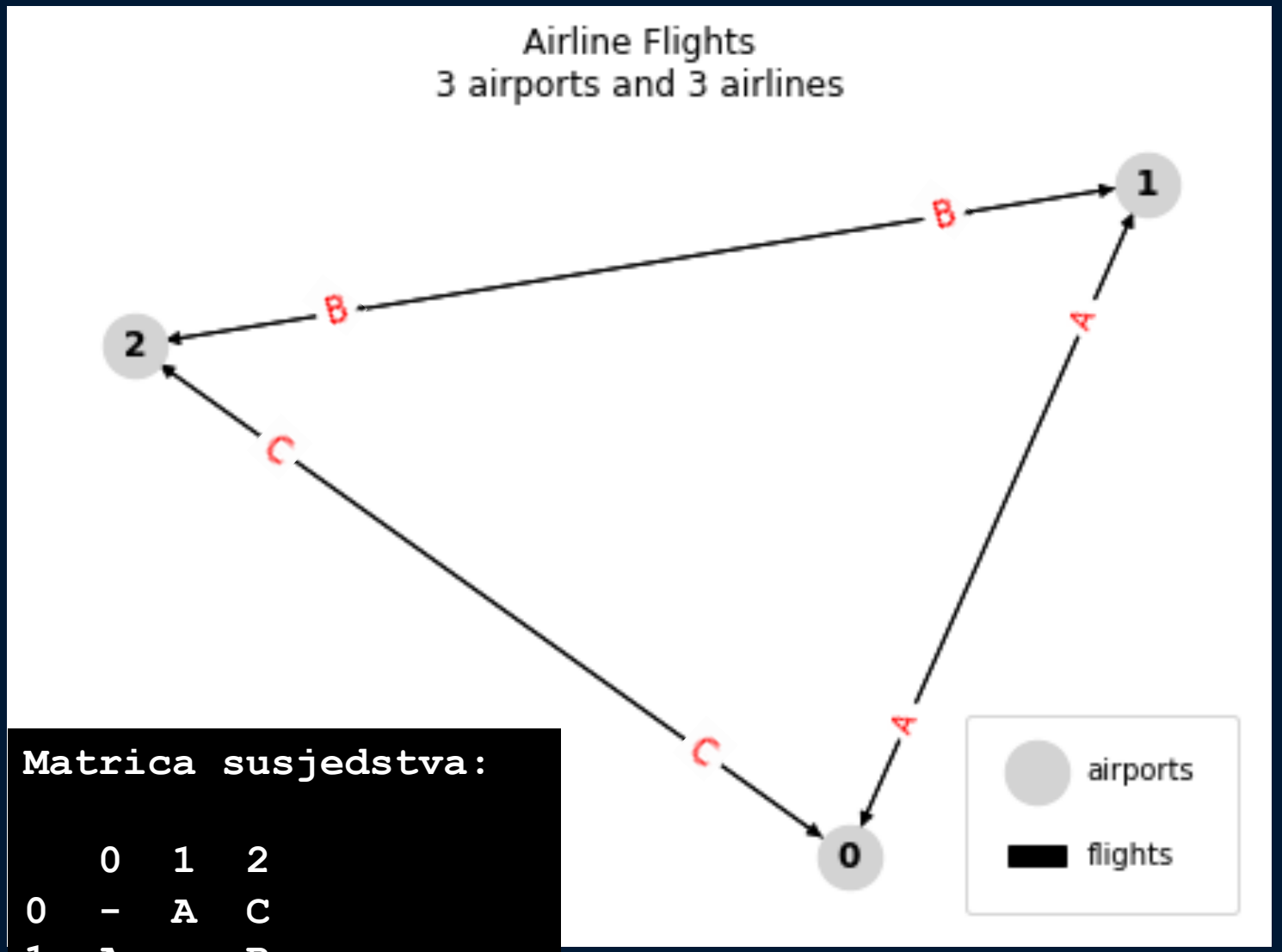
Matrica susjedstva:

	0	1	2
0	-	-	-
1	-	-	-
2	-	-	-

3. Primjeri

1)

N = 3
A = 3
Returns:
{"-AC", "A-B", "CB-"}
Remember that each airline must fly some segments, so each of the letters 'A', 'B' and 'C' must appear in the output for this test case (exactly twice). All correct answers for this test case can be obtained from the one shown in the example by permuting 'A', 'B', and 'C'.



Matrica susjedstva:

	0	1	2
0	-	A	C
1	A	-	B
2	C	B	-

3. Primjeri

2)

N = 5
A = 3
Returns:
{"-ABAB", "A-ACC",
"BA-AC", "ACA-B",
"BCCB-"}

Matrica susjedstva:

	0	1	2	3	4
0	-	A	B	A	B
1	A	-	C	C	C
2	B	A	-	A	C
3	A	C	A	-	B
4	B	C	C	B	-

The example output describes the following situation:

Airline 'A' executes the flights between the following pairs of airports: 0-1, 1-2, 2-3, 3-0.

Airline 'B' executes the flights between the following pairs of airports: 2-0, 0-4, 4-3.

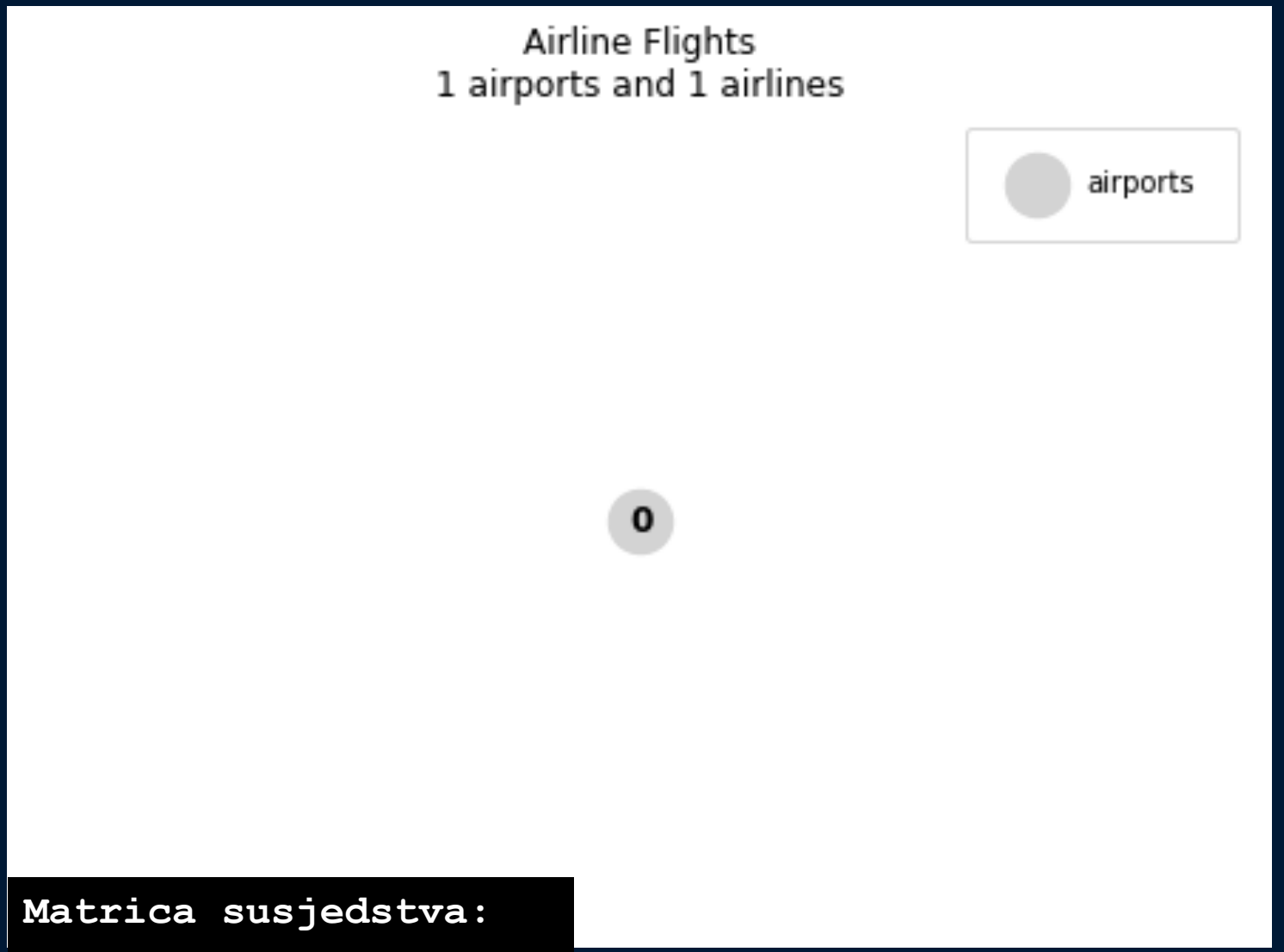
Airline 'C' executes the flights between the following pairs of airports: 3-1, 1-4, 4-2.

We can easily verify that none of these airlines can sell a valid ticket with an odd number of flights.

3. Primjeri

3)

```
N = 1
A = 1
Returns: { }
If there is just one
airport, no flights are
possible, so the only
airline in this country
does not fly anywhere.
This contradicts the
requirement that each
airline must actually fly
some flights.
```




```
Matrica susjedstva:

    0
0   -
```

3. Primjeri

4)

N = 6
A = 5
Returns:
{"-ACEDB", "A-BDCE",
"CB-AED", "EDA-BC",
"DCEB-A", "BEDCA-" }

Matrica susjedstva:

	0	1	2	3	4	5
0	-	A	C	E	D	B
1	A	-	B	D	C	E
2	C	B	-	A	E	D
3	E	D	A	-	B	C
4	D	C	E	B	-	A
5	B	E	D	C	A	-

In the example output, each airline has exactly one flight out of each airport.

Thus, obviously, each valid ticket people can buy just goes back and forth between two airports one or more times, and therefore there are no valid tickets with an odd number of flights.

Of course, this is just one of very many valid solutions.

4. Implementacija

- Provjera uvjeta
- Provjera broja aerodroma / avioprijevoznika
- Inicijalizacija matrice rezultata – plana letova
- Stvaranje liste avio-kompanija
- Popunjavanje matrice za parni N
  - Ako je N paran, popunjava matricu rezultata avio-kompanijama za direktne letove između svaka dva različita aerodroma.
- Popunjavanje matrice za neparan N
  - Ako je N neparan, prvo popunjava matricu za direktne letove, a zatim dodatno postavlja avio-kompanije za letove prema i iz zadnjeg aerodroma kako bi se osiguralo da nema karata s neparnim brojem letova.
- Pretvaranje rezultata u listu stringova i vraćanje

```
21  class ChristmasTravel:
22      def plan(self, N, A):
23          """
24          Determines if the Grand Vizier's plan for air travel can be carried out.
25
26          Parameters:
27          - N (int): Number of airports in Absurdistan (1 to 100, inclusive).
28          - A (int): Number of separate airline companies (1 to 26, inclusive).
29
30          Returns:
31          - list of str: If the plan can be carried out, returns a list of N strings,
32                         each containing N characters. If not, returns an empty list.
33          """
34          if not (1 <= N <= 100 and 1 <= A <= 26):
35              raise ValueError("Input parameters do not meet the constraints.")
36
37          # If there is just one airport, no flights are possible.
38          # Also check if number of airlines is sufficent.
39          if N % 2 == 1 and A < N // 2 or A == 1:
40              return []
41
42          res = [['-'] * N for _ in range(N)]
43
44          airlines = [chr(ord('A') + i) for i in range(A)]
45
46          # Handle even N
47          if N % 2 == 0:
48              for i in range(N):
49                  for j in range(i + 1, N):
50                      res[i][j] = res[j][i] = airlines[(i + j) % A]
51          # Handle odd N
52          else:
53              for i in range(N - 1):
54                  for j in range(i + 1, N):
55                      res[i][j] = res[j][i] = airlines[(i + j) % A]
56              for i in range(N - 1):
57                  res[i][N - 1] = res[N - 1][i] = airlines[(i + N - 1) % A]
58
59          return [''.join(res[i]) for i in range(N)]
60
```

```python
    def print_airline_flights(self, N, A):
        """
        Prints the flights executed by each airline.

        Parameters:
        - N (int): Number of airports in Absurdistan (1 to 100, inclusive).
        - A (int): Number of separate airline companies (1 to 26, inclusive).
        """
        flights = self.plan(N, A)

        if not flights:
            print('No flights are possible.')
            return

        airlines = [chr(ord('A') + i) for i in range(A)]

        for i, airline in enumerate(airlines):
            print(f"Airline '{airline}' executes the flights between the following pairs of airports: ", end='')
            airport_pairs = []
            for j in range(len(flights)):
                for k in range(j + 1, len(flights[j])):
                    if flights[j][k] == airline:
                        airport_pairs.append((j, k))
            print(', '.join([f'{pair[0]}-{pair[1]}' for pair in airport_pairs]))
```

4. Implementacija – Dodatne funkcije

```python
    def visualize_plan(self, N, A):
        """
        Visualizes the Christmas Travel plan using a directed graph.
        Using NetworkX Python package.

        Parameters:
        - N (int): Number of airports in Absurdistan (1 to 100, inclusive).
        - A (int): Number of separate airline companies (1 to 26, inclusive).
        """
        flights = self.plan(N, A)

        if not flights:
            print('No valid flights plan to visualize.')
            return
        else:
            print('Visualizing flights plan, see Plots.')

        # Create a directed graph
        G = nx.DiGraph()

        # Add nodes representing airports
        for i in range(N):
            G.add_node(i, label=str(i))

        # Add edges representing flights
        for i in range(N):
            for j in range(i + 1, N):
                if flights[i][j] != '-':
                    G.add_edge(i, j, airline=flights[i][j])
                    G.add_edge(j, i, airline=flights[j][i])

        # Draw the graph
        pos = nx.spring_layout(G)
        edge_labels = {(i, j): flights[i][j] for i, j in G.edges()}

        nx.draw(G, pos, with_labels=True, node_size=500, node_color="lightgray", font_weight='bold')
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, label_pos=0.8, font_color='red')

        plt.title(f'ChristmasTravel: Flights Plan Visualization\n{N} airports and {A} airlines')
        plt.legend(['airports', 'flights'], borderpad=1.5, labelspacing=2)
        plt.show()
```

4. Implementacija – Testiranje

```python
import unittest
from ChristmasTravel import ChristmasTravel

class TestChristmasTravel(unittest.TestCase):
    def setUp(self):
        self.christmas_travel = ChristmasTravel()

    def check_valid_output(self, result, N, A):
        """
        Check if the output meets the specified criteria for a valid solution.

        Parameters:
        - result (list of str): The output to be checked.
        - N (int): Number of airports in Absurdistan.
        - A (int): Number of separate airline companies.
        """
        # Check if the output is a list
        self.assertIsInstance(result, list)

        if result:
            # Check if each element is a string of length N
            self.assertTrue(all(isinstance(row, str) and len(row) == N for row in result))

            # Check if '-' is on the main diagonal
            self.assertTrue(all(result[i][i] == '-' for i in range(N)))

            # Check if characters at [i][j] and [j][i] are equal and represent the airline
            for i in range(N):
                for j in range(i + 1, N):
                    self.assertEqual(result[i][j], result[j][i])

            # Check if each character is one of the first A uppercase letters or '-'
            valid_letters = set(chr(ord('A') + i) for i in range(A))
            self.assertTrue(all(char in valid_letters or char == '-' for row in result for char in row)
```

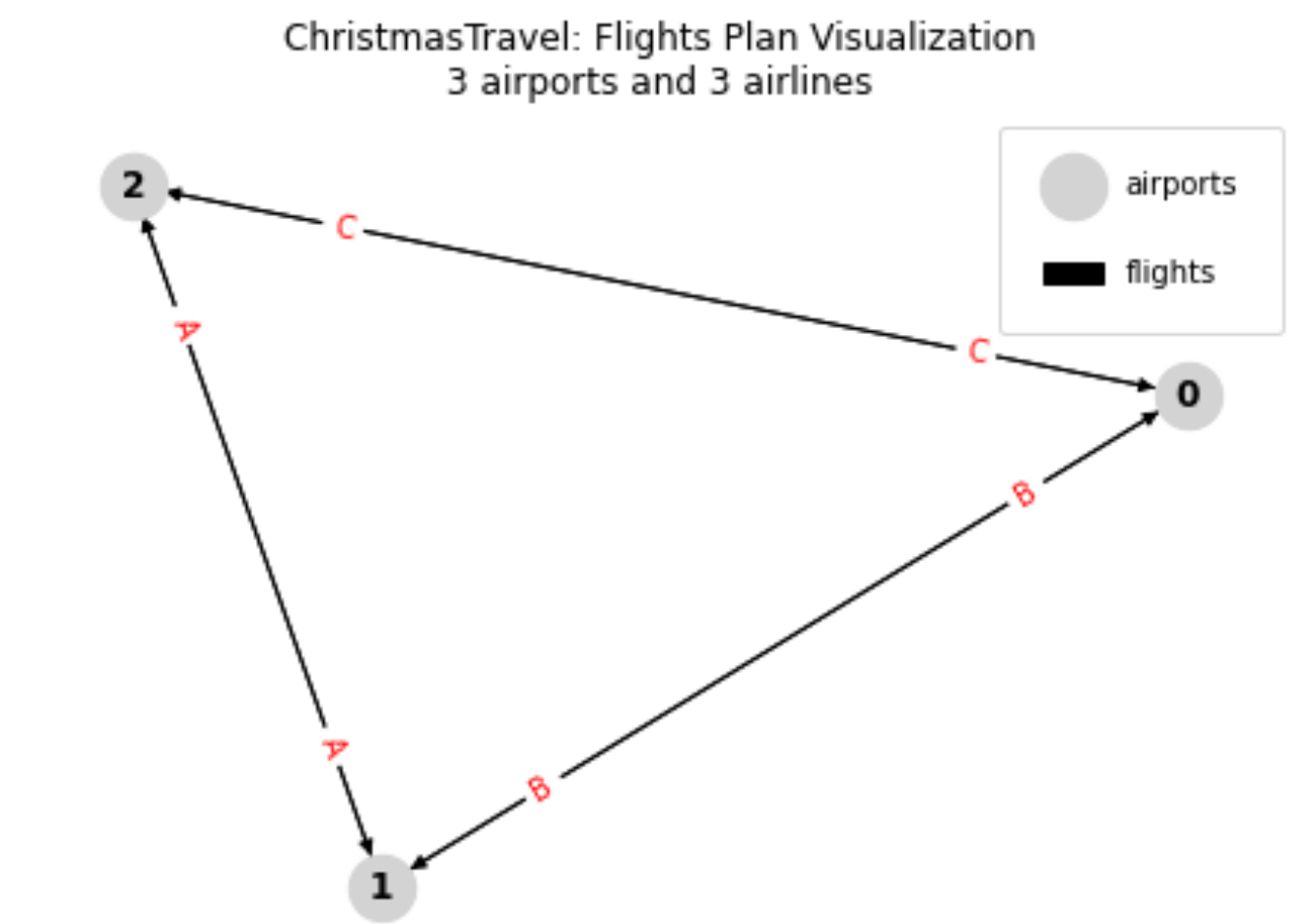
4. Implementacija – Izlaz funkcije

```python
def main():
    christmas_travel = ChristmasTravel()

    # Note, each example is just one of very many valid solutions.

    # Example 0)
    # []
    print('Example 0)\n\tN=3\n\tA=1')
    print('Returns:', christmas_travel.plan(3, 1))
    christmas_travel.print_airline_flights(3, 1)
    christmas_travel.visualize_plan(3, 1)

    # Example 1)
    # ["-AC", "A-B", "CB-"]
    print('\nExample 1)\n\tN=3\n\tA=3')
    print('Returns:', christmas_travel.plan(3, 3))
    christmas_travel.print_airline_flights(3, 3)
    christmas_travel.visualize_plan(3, 3)

    # Example 2)
    # ["-ABAB", "A-ACC", "BA-AC", "ACA-B", "BCCB-"]
    print('\nExample 2)\n\tN=5\n\tA=3')
    print('Returns:', christmas_travel.plan(5, 3))
    christmas_travel.print_airline_flights(5, 3)
    christmas_travel.visualize_plan(5, 3)

    # Example 3)
    # []
    print('\nExample 3)\n\tN=1\n\tA=1')
    print('Returns:', christmas_travel.plan(1, 1))
    christmas_travel.print_airline_flights(1, 1)
    christmas_travel.visualize_plan(1, 1)

    # Example 4)
    # ["-ACEDB", "A-BDCE", "CB-AED", "EDA-BC", "DCEB-A", "BEDCA-"]
    print('\nExample 4)\n\tN=6\n\tA=5')
    print('Returns:', christmas_travel.plan(6, 5))
    christmas_travel.print_airline_flights(6, 5)
    christmas_travel.visualize_plan(6, 5)

    # print(christmas_travel.plan(N, A))
    # christmas_travel.visualize_plan(N, A)

if __name__ == '__main__':
    main()
```

```
Matrica susjedstva:

   0  1  2
0  -  -  -
1  -  -  -
2  -  -  -

Example 0)
        N=3
        A=1
Returns: []
No flights are possible.
No valid flights plan to visualize.

…

Example 3)
        N=1
        A=1
Returns: []
No flights are possible.
No valid flights plan to visualize.

…

Matrica susjedstva:

   0
0  -
```

4. Implementacija – Izlaz funkcije

```
Example 1)
        N=3
        A=3
Returns: ['-BC', 'B-A', 'CA-']
Airline 'A' executes the flights between the following pairs of airports: 1-2
Airline 'B' executes the flights between the following pairs of airports: 0-1
Airline 'C' executes the flights between the following pairs of airports: 0-2
Visualizing flights plan, see Plots.
```

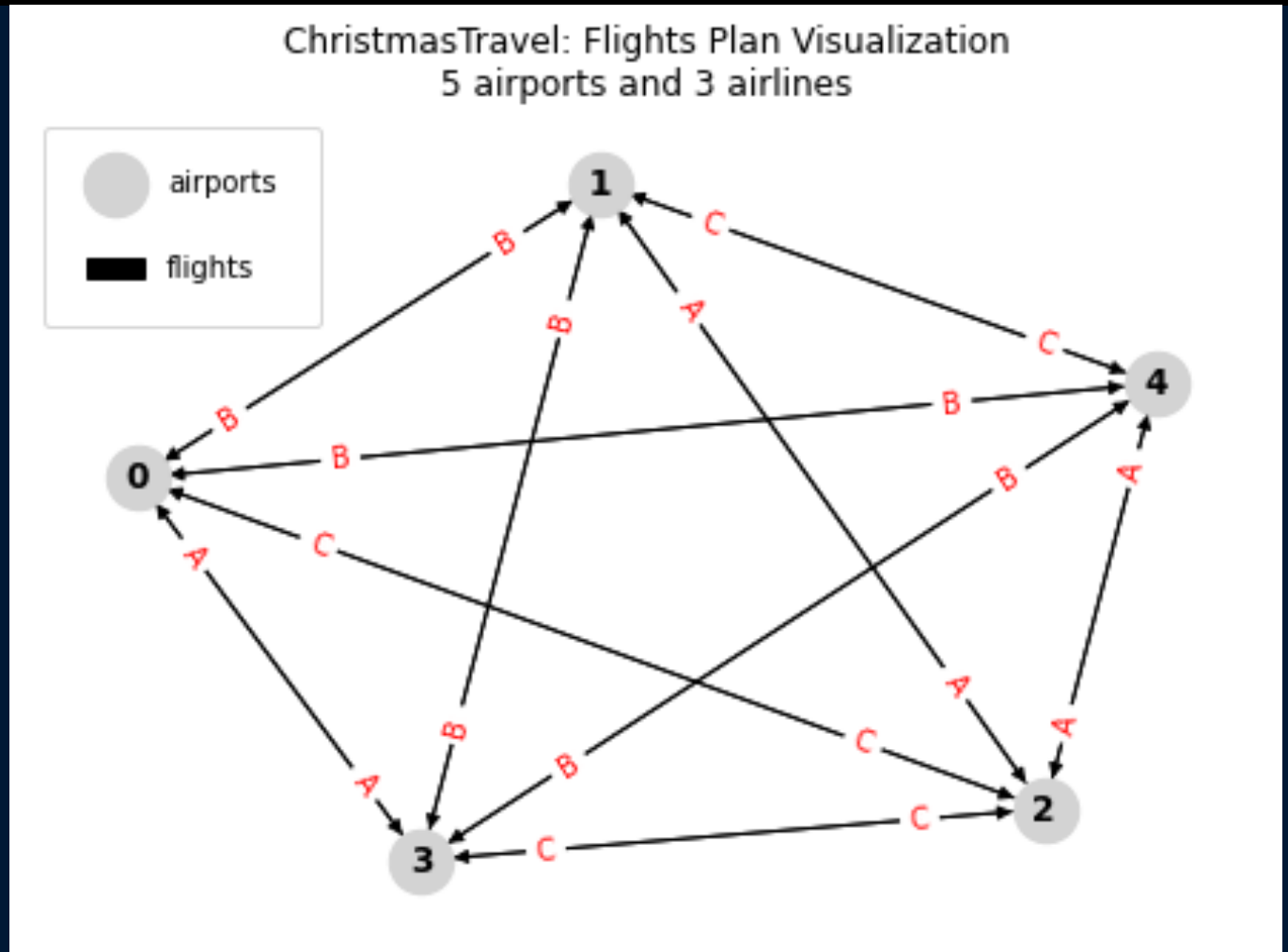


Matrica susjedstva:

	0	1	2
0	-	B	C
1	B	-	A
2	C	A	-

4. Implementacija – Izlaz funkcije

```
Example 2)
        N=5
        A=3
Returns: ['-BCAB', 'B-ABC', 'CA-CA', 'ABC-B', 'BCAB-']
Airline 'A' executes the flights between the following pairs of airports: 0-3, 1-2, 2-4
Airline 'B' executes the flights between the following pairs of airports: 0-1, 0-4, 1-3, 3-4
Airline 'C' executes the flights between the following pairs of airports: 0-2, 1-4, 2-3
Visualizing flights plan, see Plots.
```

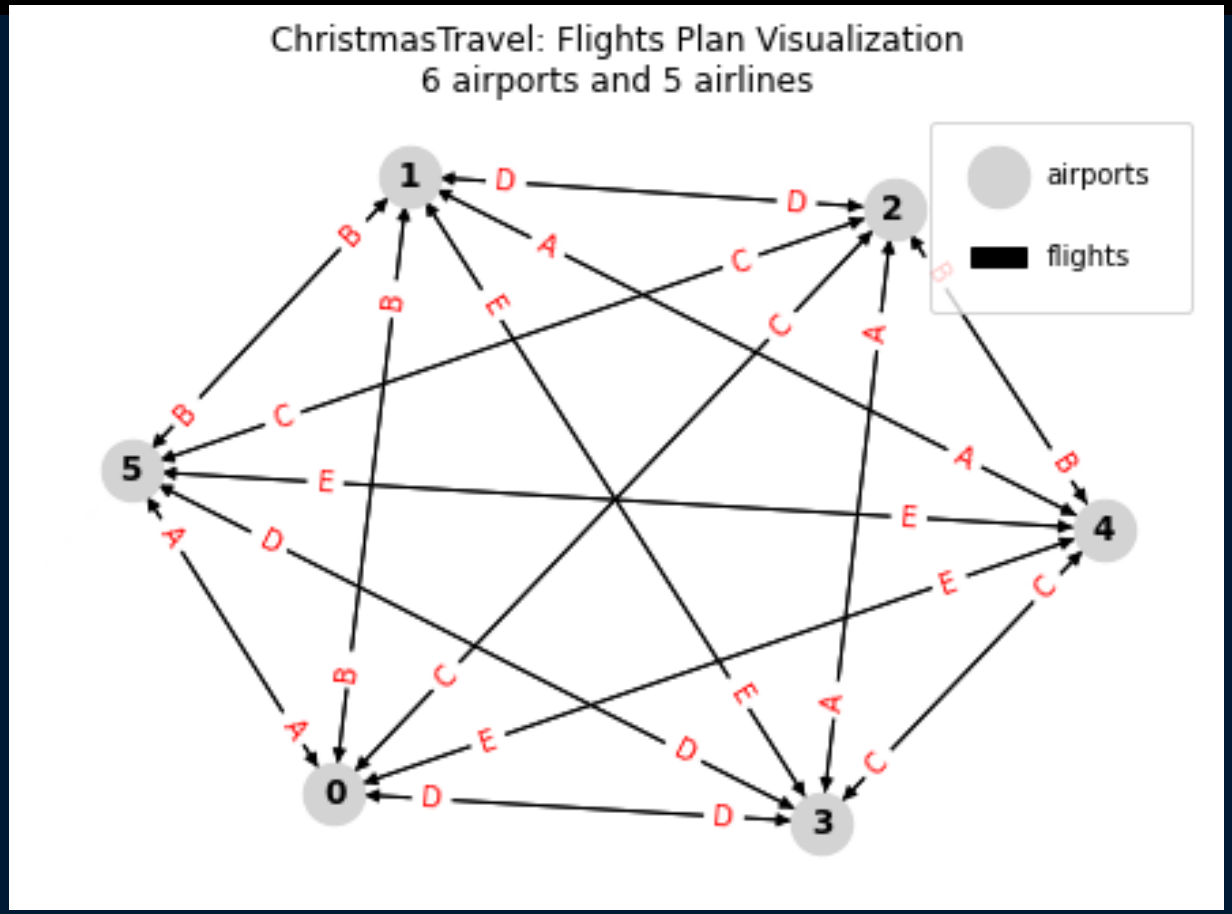


Matrica susjedstva:

	0	1	2	3	4
0	-	B	C	A	B
1	B	-	A	B	C
2	C	A	-	C	A
3	A	B	C	-	B
4	B	C	A	B	-

4. Implementacija – Izlaz funkcije

```
Example 4)
        N=6
        A=5
Returns: ['-BCDEA', 'B-DEAB', 'CD-ABC', 'DEA-CD', 'EABC-E', 'ABCDE-']
Airline 'A' executes the flights between the following pairs of airports: 0-5, 1-4, 2-3
Airline 'B' executes the flights between the following pairs of airports: 0-1, 1-5, 2-4
Airline 'C' executes the flights between the following pairs of airports: 0-2, 2-5, 3-4
Airline 'D' executes the flights between the following pairs of airports: 0-3, 1-2, 3-5
Airline 'E' executes the flights between the following pairs of airports: 0-4, 1-3, 4-5
Visualizing flights plan, see Plots.
```



Matrica susjedstva:

	0	1	2	3	4	5
0	-	B	C	D	E	A
1	B	-	D	E	A	B
2	C	D	-	A	B	C
3	D	E	A	-	C	D
4	E	A	B	C	-	E
5	A	B	C	D	E	-

4. Implementacija – Testiranje

```
51      def test_example_0(self):
52          """
53          Test case for the example 0 in the problem statement.
54          """
55          result = self.christmas_travel.plan(3, 1)
56          self.check_valid_output(result, 3, 1)
57
58      def test_example_1(self):
59          """
60          Test case for the example 1 in the problem statement.
61          """
62          result = self.christmas_travel.plan(3, 3)
63          self.check_valid_output(result, 3, 3)
64
65      def test_example_2(self):
66          """
67          Test case for the example 2 in the problem statement.
68          """
69          result = self.christmas_travel.plan(5, 3)
70          self.check_valid_output(result, 5, 3)
71
72      def test_example_3(self):
73          """
74          Test case for the example 3 in the problem statement.
75          """
76          result = self.christmas_travel.plan(1, 1)
77          self.check_valid_output(result, 1, 1)
78
79      def test_example_4(self):
80          """
81          Test case for the example 4 in the problem statement.
82          """
83          result = self.christmas_travel.plan(6, 5)
84          self.check_valid_output(result, 6, 5)
85
86  if __name__ == '__main__':
87      unittest.main()
88
```

```
.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
```

Hvala na pozornosti!