DELAY ESTIMATION AND SHORTEST FLIGHT PATH ANALYSIS

CME 4422 INTRODUCTION TO GRAPH THEORY

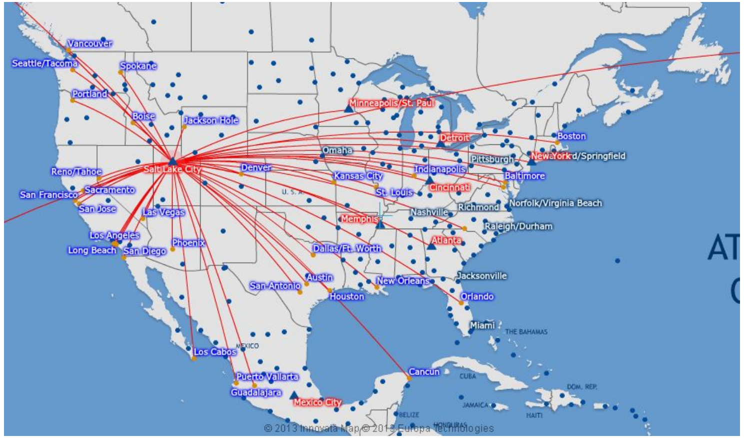
2014510028 Çağrı Anıl ERBEY
2017510072 Hüseyin Emrecan TAN
2017510076 Oğuzhan TÜRKMEN

CONTENTS

- INTRODUCTION
- DATASET
- PREPROCESSING
- DATA MINING MODELS
- GRAPH ALGORITHMS
- CONCLUSION

INTRODUCTION

- Flight information between US cities in October 2014 is used.
- The estimation of delays that may occur in flights was made using many features.
- The shortest flight between the two cities was tried to be found.



LIBRARIES

- Networkx
- Pandas
- Numpy
- Scikit-Learn
- Matplotlib

DATASET

Dataset Name: International Air Transportation Performance

Dataset Source: Kaggle

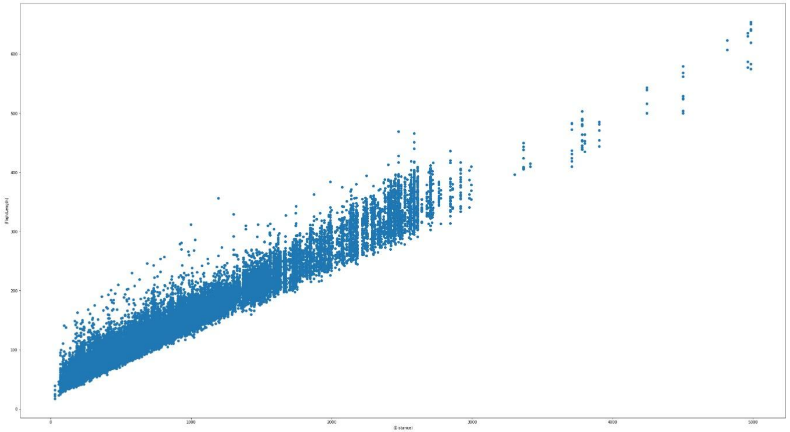
```
RangeIndex: 49101 entries, 0 to 49100
Data columns (total 50 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Year                   49101 non-null  int64
 1   Quarter                49101 non-null  int64
 2   Month                  49101 non-null  int64
 3   DayofMonth             49101 non-null  int64
 4   DayOfWeek              49101 non-null  int64
 5   FlightDate             49101 non-null  object
 6   UniqueCarrier          49101 non-null  object
 7   TailNum                49042 non-null  object
 8   FlightNum              49101 non-null  int64
 9   Origin                 49101 non-null  object
 10  OriginCityName         49101 non-null  object
 11  OriginState            49101 non-null  object
 12  OriginStateFips        49101 non-null  int64
 13  OriginStateName        49101 non-null  object
 14  Dest                   49101 non-null  object
 15  DestCityName           49101 non-null  object
 16  DestState              49101 non-null  object
 17  DestStateFips          49101 non-null  int64
 18  DestStateName          49101 non-null  object
 19  CRSDepTime             49101 non-null  int64
 20  DepTime                48599 non-null  float64
 21  DepDelay               48599 non-null  float64
 22  DepDelayMinutes        48599 non-null  float64
 23  DepDel15               48599 non-null  float64
 24  DepartureDelayGroups   48599 non-null  float64
 25  DepTimeBlk             49101 non-null  object
 26  TaxiOut                48583 non-null  float64
 27  WheelsOff              48583 non-null  float64
 28  WheelsOn               48567 non-null  float64
 29  TaxiIn                 48567 non-null  float64
 30  CRSArrTime             49101 non-null  int64
 31  ArrTime                48567 non-null  float64
 32  ArrDelay               48495 non-null  float64
 33  ArrDelayMinutes        48495 non-null  float64
 34  ArrDel15               48495 non-null  float64
 35  ArrivalDelayGroups     48495 non-null  float64
 36  ArrTimeBlk             49101 non-null  object
 37  Cancelled              49101 non-null  int64
 38  CancellationCode       522 non-null    object
 39  Diverted               49101 non-null  int64
 40  CRSElapsedTime         49101 non-null  int64
 41  Distan                 48495 non-null  float64
 42  AirTime                48495 non-null  float64
 43  Distance               49101 non-null  int64
 44  DistanceGroup          49101 non-null  int64
 45  CarrierDelay           9124 non-null   float64
 46  WeatherDelay           9124 non-null   float64
 47  NASDelay               9124 non-null   float64
 48  SecurityDelay          9124 non-null   float64
 49  LateAircraftDelay      9124 non-null   float64
```

PREPROCESSING

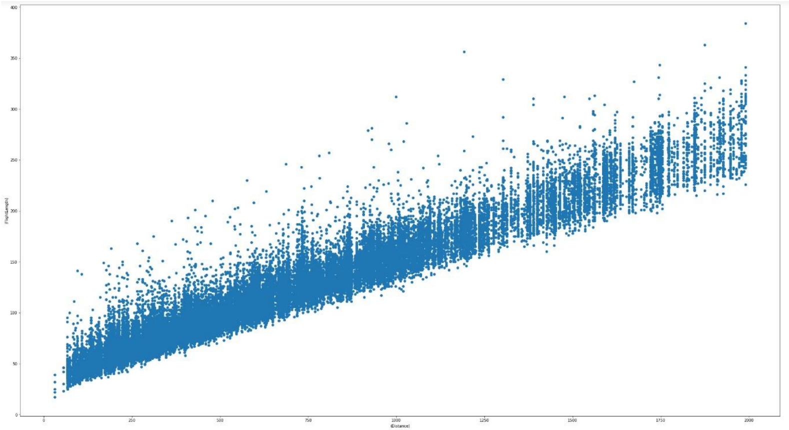
- Dropped Columns
- Delay Classification
- Flight Range Calculations
- KNN Imputer for NaN values
- Converted Data Types
- Generated New Columns
- Outlier Detection

DETECTED OUTLIERS

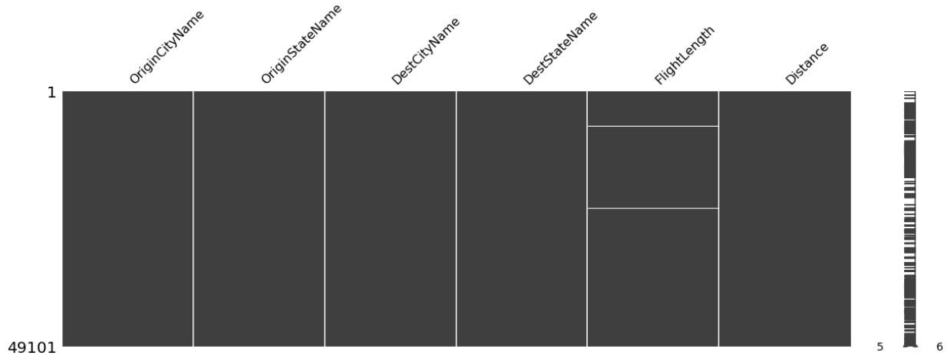
BEFORE



AFTER

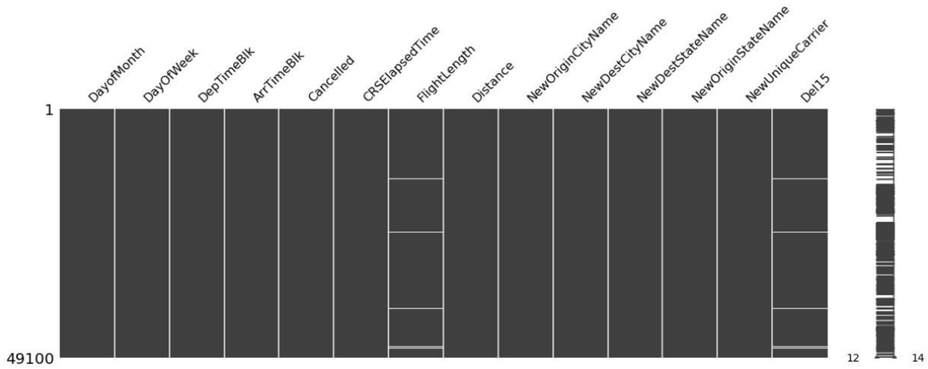


<AxesSubplot:>



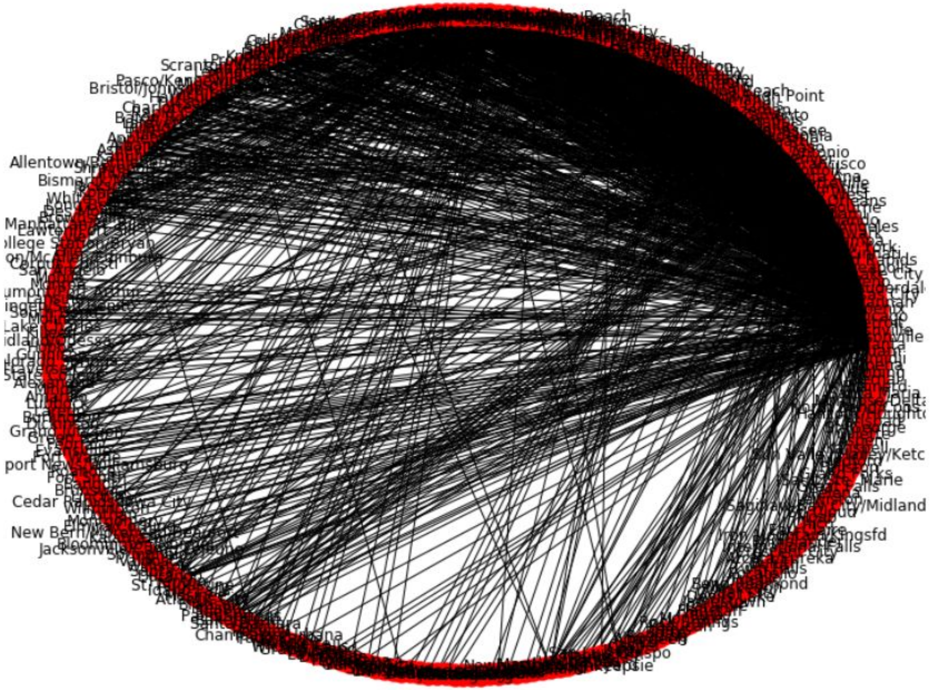
- NaN Values for Graph Dataset

<AxesSubplot:>

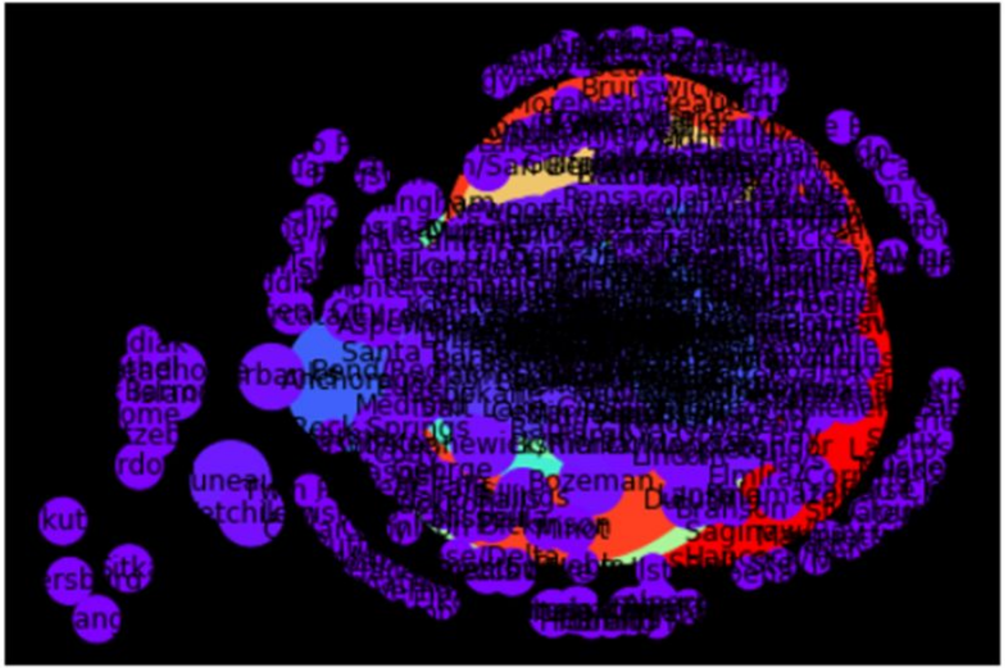


- NaN Values for Mining Dataset

Roads between cities with no distance



Cities in graph with distance



Datasets after Preprocessing

- Dataset of the Graph Side

```
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   OriginCityName   49101 non-null  object
 1   OriginStateName  49101 non-null  object
 2   DestCityName     49101 non-null  object
 3   DestStateName    49101 non-null  object
 4   FlightLength     49101 non-null  float64
 5   Distance         49101 non-null  int64
dtypes: float64(1), int64(1), object(4)
memory usage: 2.2+ MB
```

Datasets after Preprocessing

- Dataset of the Data Mining Side

```
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   DayofMonth           46130 non-null  int32
 1   DayOfWeek            46130 non-null  int32
 2   DepTimeBlk           46130 non-null  int32
 3   ArrTimeBlk           46130 non-null  int32
 4   Cancelled            46130 non-null  int32
 5   CRSElapsedTime       46130 non-null  int32
 6   FlightLength         46130 non-null  int32
 7   Distance             46130 non-null  int32
 8   NewOriginCityName    46130 non-null  int32
 9   NewDestCityName      46130 non-null  int32
 10  NewDestStateName     46130 non-null  int32
 11  NewOriginStateName   46130 non-null  int32
 12  NewUniqueCarrier     46130 non-null  int32
 13  Del15                46130 non-null  int32
dtypes: int32(14)
memory usage: 2.8 MB
```

DATA MINING ALGORITHMS

- **Support Vector Machine**
- **Logistic Regression**
- **k - Nearest Neighbours**

DATA MINING ALGORITHMS

(SUPPORT VECTOR MACHINE)

```python
cv = KFold(n_splits=10, random_state=1, shuffle=True)
# create model
model = svm.SVC(kernel='poly', C=100.0)
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

Accuracy: 0.899 (0.004)

	precision	recall	f1-score	support
0	0.89	1.00	0.94	3639
1	1.00	0.54	0.70	974
accuracy			0.90	4613
macro avg	0.95	0.77	0.82	4613
weighted avg	0.91	0.90	0.89	4613

DATA MINING ALGORITHMS

(LOGISTIC REGRESSION)

```
cv = KFold(n_splits=10, random_state=1, shuffle=True)
# create model
model = LogisticRegression()
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.865 (0.014)
```

	precision	recall	f1-score	support
0	0.87	0.94	0.90	3639
1	0.69	0.46	0.55	974
accuracy			0.84	4613
macro avg	0.78	0.70	0.73	4613
weighted avg	0.83	0.84	0.83	4613

DATA MINING ALGORITHMS

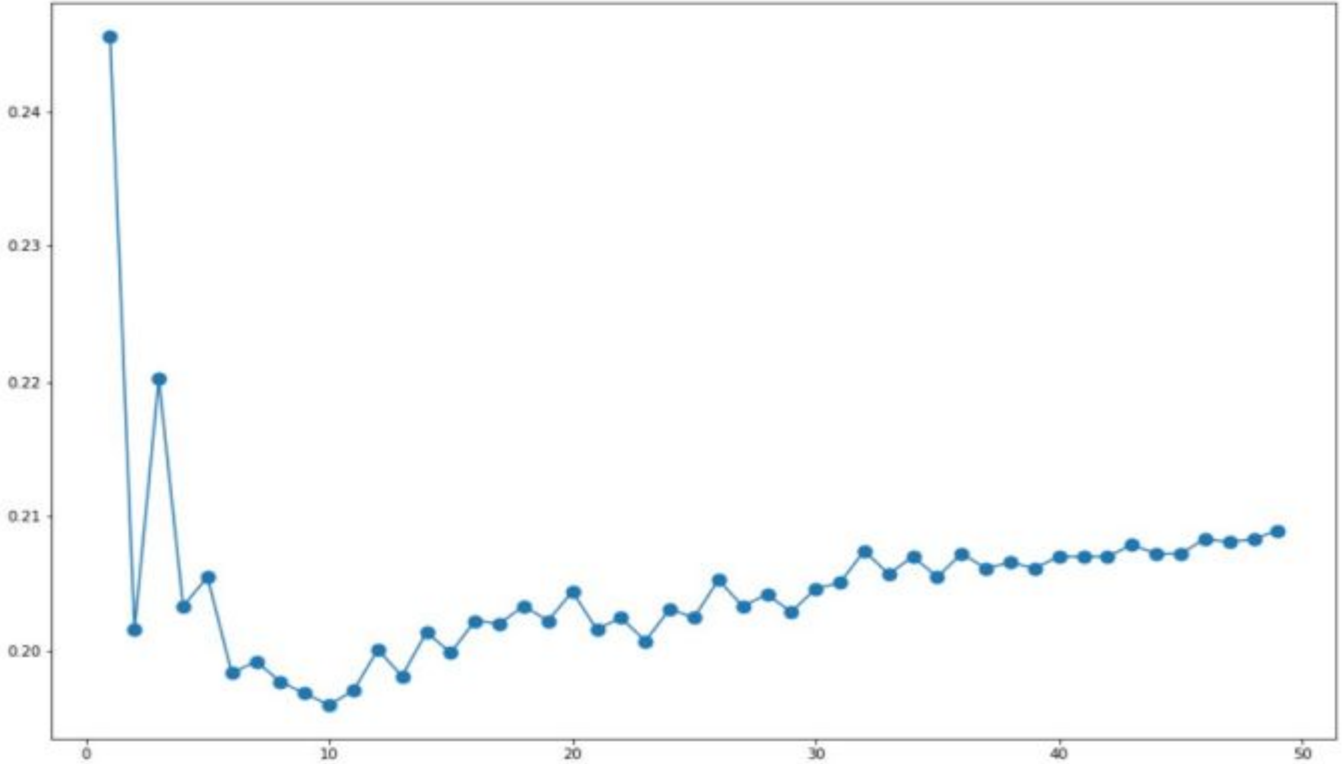
(K - NEAREST NEIGHBOURS)

```
cv = KFold(n_splits=10, random_state=1, shuffle=True)
# create model
model = KNeighborsClassifier(n_neighbors = 10)
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

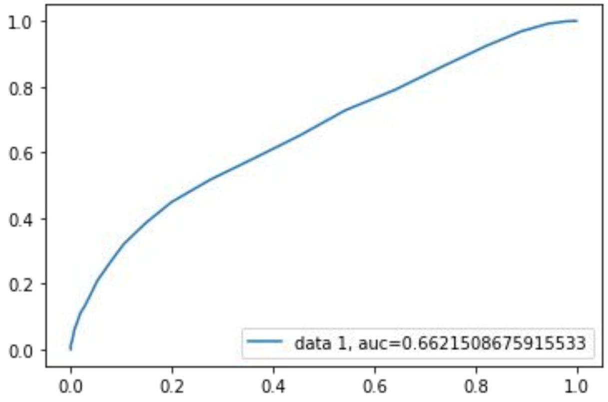
```
Accuracy: 0.798 (0.006)
```

	precision	recall	f1-score	support
0	0.82	0.95	0.88	3639
1	0.54	0.24	0.33	974
accuracy			0.80	4613
macro avg	0.68	0.59	0.60	4613
weighted avg	0.76	0.80	0.76	4613

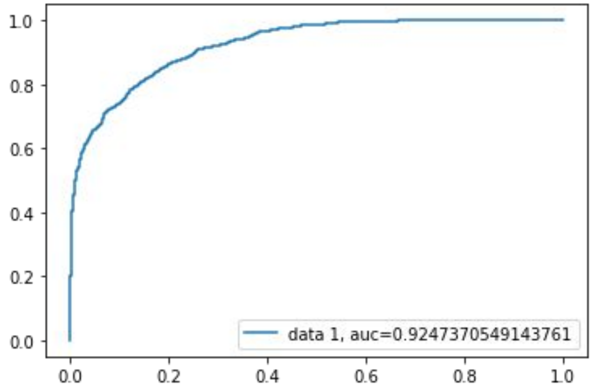
Finding Best Fit K Value



ROC CURVES



KNeighbors



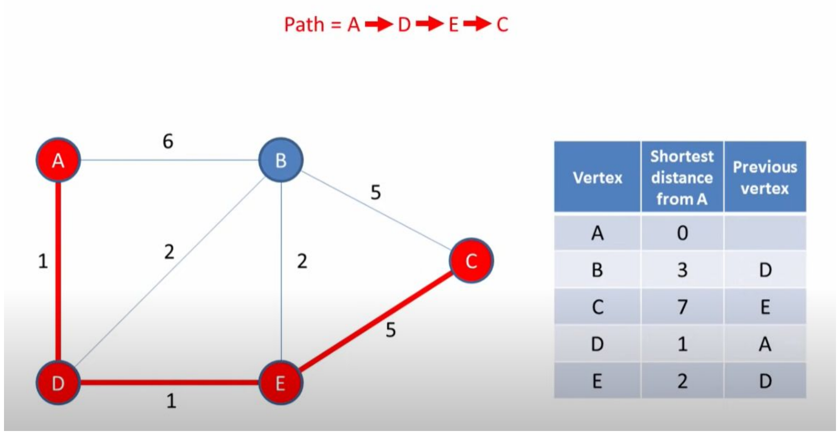
LogReg

GRAPH ALGORITHMS

- **Dijkstra's Shortest Path Algorithm**
- **A Star Shortest Path Algorithm**
- **Bellman Ford Shortest Path Algorithm**

Dijkstra's Shortest Path Algorithm

Example algorithm image:

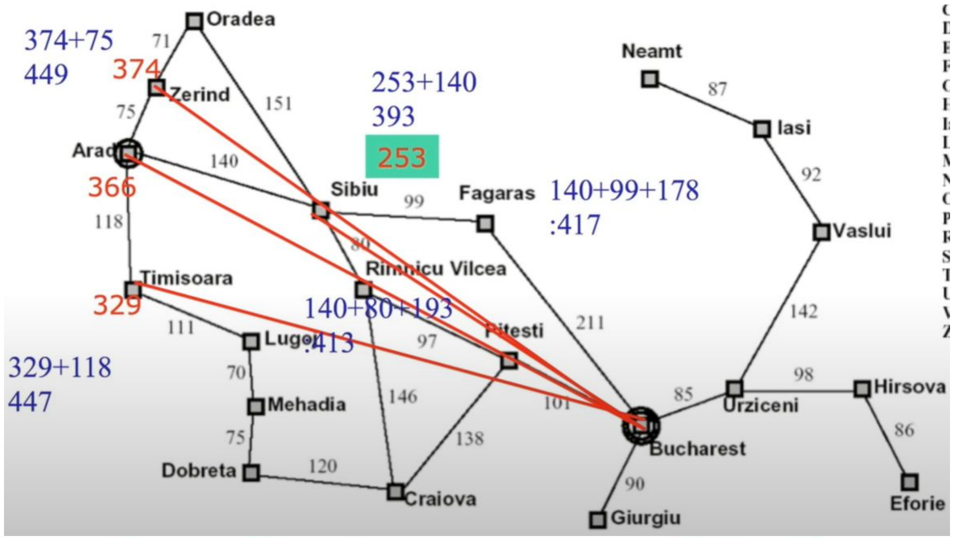



```python
ShortestCityPath = nx.dijkstra_path(G, source="Grand Forks", target="El Paso", weight='FlightLength')
print("Shortest path between Origin and Dest:", ShortestCityPath)
```

```
Shortest path between Origin and Dest: ['Grand Forks', 'Minneapolis', 'Los Angeles', 'El Paso']
```

A Star Shortest Path Algorithm

Example algorithm image:

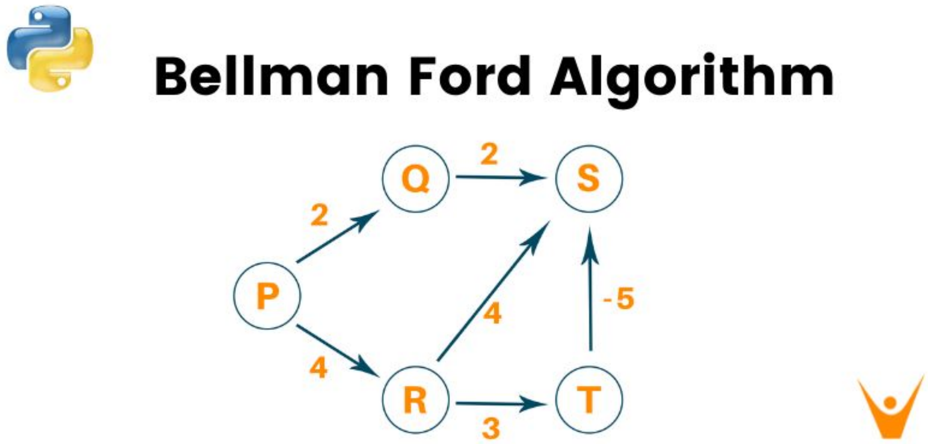



```
ShortestCityPath = nx.astar_path(G, source="Grand Forks", target="El Paso", heuristic=None, weight='FlightLength')
print("Shortest path between Origin and Dest:", ShortestCityPath)
```

```
Shortest path between Origin and Dest: ['Grand Forks', 'Minneapolis', 'Los Angeles', 'El Paso']
```

Bellman Ford Shortest Path Algorithm

Example algorithm image:




```
ShortestCityPath = nx.shortest_path(G, source="Grand Forks", target="El Paso", method='bellman-ford',weight='FlightLength')
print("Shortest path between Origin and Dest:", ShortestCityPath)
```

```
Shortest path between Origin and Dest: ['Grand Forks', 'Minneapolis', 'Los Angeles', 'El Paso']
```

```
Shortest path between Origin and Dest: ['Grand Forks', 'Minneapolis', 'Los Angeles', 'El Paso']
```

OriginCityName	OriginStateName	DestCityName	DestStateName	FlightLength
Minneapolis	Minnesota	Los Angeles	California	117.0

OriginCityName	OriginStateName	DestCityName	DestStateName	FlightLength
Los Angeles	California	El Paso	Texas	95.0

- 'Los Angeles' has been removed from the flight route.
- A new route was determined using the shortest path (Dijkstra's & A Star & Bellman Ford) algorithms.

```
Shortest path between Origin and Dest: ['Grand Forks', 'Minneapolis', 'Denver', 'El Paso']
```

OriginCityName	OriginStateName	DestCityName	DestStateName	FlightLength
Minneapolis	Minnesota	Denver	Colorado	232.0

OriginCityName	OriginStateName	DestCityName	DestStateName	FlightLength
Denver	Colorado	El Paso	Texas	103.0

THANKS FOR LISTENING!