Chapter 14: Graph algorithms I. - shortest path

The sample dataset for this lecture is given in the `airports.csv` and `airroutes.csv` files in the `data` folder. (The column separator is the `;` character.)

- The `airports.csv` file contains information about (larger) airports all over the world:
    1. IATA code (International Air Transport Association code, e.g. *BUD* for the Budapest Airport)
    2. ICAO code (International Civil Aviation Organization code, e.g. *LHBP* for the Budapest Airport)
    3. Name
    4. Number of runways
    5. Longest runway length (in foots)
    6. Elevation (in foots)
    7. Country
    8. Country region
    9. City
    10. Latitude
    11. Longitude
- The `airroutes.csv` consists of the direct flight relations between the airports, identifying them with their IATA code. The distance of the airports / length of the flight route is also given (in miles). The flights are directed, if there is a flight between both directions of two airports, then there will be two records in the file, with opposite direction.

Reading the dataset

Read the airport data

First read the airports data into a pandas *DataFrame*.

In [1]:

```
import pandas as pd

airports = pd.read_csv('../data/airports.csv', delimiter = ';')
display(airports)
```

	iata	icao	name	runways	longest	elevation	country	region	city	
0	ATL	KATL	Hartsfield - Jackson Atlanta International Air...	5	12390	1026	US	US-GA	Atlanta	33
1	ANC	PANC	Anchorage Ted Stevens	3	12400	151	US	US-AK	Anchorage	61
2	AUS	KAUS	Austin Bergstrom International Airport	2	12250	542	US	US-TX	Austin	30
3	BNA	KBNA	Nashville International Airport	4	11030	599	US	US-TN	Nashville	36
4	BOS	KBOS	Boston Logan	6	10083	19	US	US-MA	Boston	42
...	...	...	...	...	...	...	...	...	...	
3459	LNL	ZLLN	Cheng Xian Airport	1	9186	3707	CN	CN-62	Longnan	33
3460	XAI	ZHXY	Xinyang Minggang Airport	1	8858	4528	CN	CN-41	Xinyang	32
3461	YYA	ZGYY	Sanhe Airport	1	8530	230	CN	CN-43	Yueyang	29
3462	BQJ	UEBB	Batagay Airport	2	6562	699	RU	RU-SA	Batagay	67
3463	DPT	UEBD	Deputatskij Airport	1	7021	920	RU	RU-SA	Deputatskij	69

3464 rows × 11 columns

Note: the length of the longest runway and the elevation is given in foots.

Lets set the column `iata` as the index column, so each row of data will be accessible later by indexing the airports with their IATA code.

In [2]:

```
airports.set_index('iata', inplace=True)
display(airports)
```

iata	icao	name	runways	longest	elevation	country	region	city	la
ATL	KATL	Hartsfield - Jackson Atlanta International Air...	5	12390	1026	US	US-GA	Atlanta	33.63670
ANC	PANC	Anchorage Ted Stevens	3	12400	151	US	US-AK	Anchorage	61.17440
AUS	KAUS	Austin Bergstrom International Airport	2	12250	542	US	US-TX	Austin	30.19450
BNA	KBNA	Nashville International Airport	4	11030	599	US	US-TN	Nashville	36.12450
BOS	KBOS	Boston Logan	6	10083	19	US	US-MA	Boston	42.36430
...	...	...	...	...	...	...	...	...	.
LNL	ZLLN	Cheng Xian Airport	1	9186	3707	CN	CN-62	Longnan	33.78972
XAI	ZHXY	Xinyang Minggang Airport	1	8858	4528	CN	CN-41	Xinyang	32.54055
YYA	ZGYY	Sanhe Airport	1	8530	230	CN	CN-43	Yueyang	29.31250
BQJ	UEBB	Batagay Airport	2	6562	699	RU	RU-SA	Batagay	67.64777
DPT	UEBD	Deputatskij Airport	1	7021	920	RU	RU-SA	Deputatskij	69.39250

3464 rows × 10 columns

Reminder: the `set_index` function can be configured to modify the index in place or return a new *Dataframe* with the `inplace` parameter (defaults to `False`). It can also be configured to drop or keep the index column with the `drop` parameter (defaults to `True`).

The information of the Budapest Airport can now be accessed both by numerical and associative (string) indexing:

In [3]:

```
print('The Budapest airport by the numerical index:')
print(airports.iloc[111])
print()
print('The Budapest airport by the associative index:')
print(airports.loc['BUD'])
```

```
The Budapest airport by the numerical index:
icao                                                 LHBP
name         Budapest Ferenc Liszt International Airport
runways                                                 2
longest                                             12162
elevation                                             495
country                                                HU
region                                              HU-PE
city                                             Budapest
lat                                             47.436901
lon                                               19.2556
Name: BUD, dtype: object

The Budapest airport by the associative index:
icao                                                 LHBP
name         Budapest Ferenc Liszt International Airport
runways                                                 2
longest                                             12162
elevation                                             495
country                                                HU
region                                              HU-PE
city                                             Budapest
lat                                             47.436901
lon                                               19.2556
Name: BUD, dtype: object
```

The number of runways the Budapest Airport can be fetched (or modified) now 4 possible ways:

In [4]:

```
print(airports.iloc[111]['runways'])
print(airports.loc['BUD']['runways'])
print(airports['runways'][111])
print(airports['runways']['BUD'])
```

```
2
2
2
2
```

Read the airroutes data

In [5]:

```
airroutes = pd.read_csv('../data/airroutes.csv', delimiter = ';')
display(airroutes)
```

	from	to	distance
0	ATL	AUS	811
1	ATL	BNA	214
2	ATL	BOS	945
3	ATL	BWI	576
4	ATL	DCA	546
...	...	...	...
50225	NRR	CPX	23
50226	LNL	PKX	708
50227	XAI	PKX	498
50228	YYA	PKX	726
50229	BQJ	DPT	178

50230 rows × 3 columns

Note: the distance is given in miles.

Build a graph

NetworkX has an integrated conversion for *pandas* DataFrames which can be used.
Lets create a directed graph (`networkx.DiGraph`) from the flights. The edges shall be weighted with the distance of the routes.

In [6]:

```
import networkx as nx

flight_graph = nx.from_pandas_edgelist(airroutes, 'from', 'to', ['distance'], cr
eate_using = nx.DiGraph)

print('Metadata for the BUD -> JFK edge: {0}'.format(flight_graph['BUD']['JFK'
]))
```

```
Metadata for the BUD -> JFK edge: {'distance': 4356}
```

Reminder: The 4th parameter defines which *Series* (columns) of the *DataFrame* shall be added to the edges as attributes. If `True` , all of the remaining columns will be added. If `None` , no edge attributes are added to the graph. Its default value is `None` .

Calculating the shortest path

NetworkX supports various [shortest path algorithms (https://networkx.org/documentation/stable/reference/algorithms/shortest_paths.html)]:

- *Dijkstra* and *Bellman-Ford* algorithm to compute shortest path between source and all other reachable nodes;
- *Floyd-Warshall* algorithm to find the shortest path between all node pairs.

Beside the algorithm-specific functions, NetworkX also provides a uniform interface to calculate the shortest paths from a starting point to a target (or to all):

```
nx.shortest_path(graph, source, target, weight, method)
```

The default algorithm is *Dijsktra*.

Example

Calculate the path between 2 user given airports with the minimal number of transfers.

In [7]:

```
from_airport = input("From airport: ")
to_airport = input("To airport: ")

if flight_graph.has_node(from_airport) and flight_graph.has_node(to_airport):
    route = nx.shortest_path(flight_graph, from_airport, to_airport)
    print("Route: {0}".format(route))

    length = 0
    for i in range(1, len(route)):
        length += flight_graph[route[i-1]][route[i]]['distance']
    print("Length: {0} mi".format(length))
else:
    print("Source or destination airport was not found.")
```

```
Route: ['BUD', 'JFK', 'CAN', 'PKX', 'YYA']
Length: 14194 mi
```

Calculate the shortest path by distance between 2 user given airports.

In [8]:

```
from_airport = input("From airport: ")
to_airport = input("To airport: ")

if flight_graph.has_node(from_airport) and flight_graph.has_node(to_airport):
    route = nx.dijkstra_path(flight_graph, from_airport, to_airport, 'distance')
    length = nx.dijkstra_path_length(flight_graph, from_airport, to_airport, 'di
stance')
    print("Route: {0} ({1} mi)".format(route, length))
else:
    print("Source or destination airport was not found.")
```

```
Route: ['BUD', 'SVO', 'HET', 'PKX', 'YYA'] (5339 mi)
```

Calculate the shortest between 2 user given airports by distance, but with the following additional conditions:

- airports with no longer runway than 8000 feets cannot be used;
- airports with only 1 runway has a 50% penalty of the distance.

In [9]:

```
def custom_distance(from_node, to_node, edge_attr):
    if airports.loc[to_node]['longest'] < 8000:
        return None
    if airports.loc[to_node]['runways'] == 1:
        return edge_attr['distance'] * 1.5
    return edge_attr['distance']

from_airport = input("From airport: ")
to_airport = input("To airport: ")

if flight_graph.has_node(from_airport) and flight_graph.has_node(to_airport):
    route = nx.dijkstra_path(flight_graph, from_airport, to_airport, custom_dist
ance)
    length = nx.dijkstra_path_length(flight_graph, from_airport, to_airport, cus
tom_distance)
    print("Route: {0} ({1} mi)".format(route, length))
else:
    print("Source airport was not found.")
```

```
Route: ['BUD', 'SVO', 'HET', 'PKX', 'YYA'] (5702.0 mi)
```

Calculate which airports can be reached from a starting, user given airport within a reach of also user given distance (in miles). Also compute the shorthest path by distance to each of them.

In [12]:

```
from_airport = input("From airport: ")
max_distance = int(input("Max distance: "))

if flight_graph.has_node(from_airport):
    lengths, routes = nx.single_source_dijkstra(flight_graph, from_airport, None
, max_distance, 'distance')
    for to_airport in routes.keys():
        print("{0} -> {1}: {2} ({3} mi)".format(from_airport, to_airport, routes
[to_airport], lengths[to_airport]))
else:
    print("Source airport was not found.")
```

```
JFK -> JFK: ['JFK'] (0 mi)
JFK -> BOS: ['JFK', 'BOS'] (186 mi)
JFK -> BWI: ['JFK', 'BWI'] (184 mi)
JFK -> DCA: ['JFK', 'DCA'] (213 mi)
JFK -> IAD: ['JFK', 'IAD'] (227 mi)
JFK -> PHL: ['JFK', 'PHL'] (93 mi)
JFK -> PWM: ['JFK', 'PWM'] (273 mi)
JFK -> ROC: ['JFK', 'ROC'] (263 mi)
JFK -> ORF: ['JFK', 'ORF'] (290 mi)
JFK -> BUF: ['JFK', 'BUF'] (300 mi)
JFK -> RIC: ['JFK', 'RIC'] (288 mi)
JFK -> SYR: ['JFK', 'SYR'] (208 mi)
JFK -> BTV: ['JFK', 'BTV'] (266 mi)
JFK -> ORH: ['JFK', 'ORH'] (149 mi)
JFK -> ACK: ['JFK', 'ACK'] (198 mi)
JFK -> HYA: ['JFK', 'HYA'] (195 mi)
JFK -> LGA: ['JFK', 'PHL', 'LGA'] (187 mi)
JFK -> HPN: ['JFK', 'PHL', 'HPN'] (207 mi)
JFK -> EWR: ['JFK', 'PHL', 'EWR'] (172 mi)
JFK -> MDT: ['JFK', 'PHL', 'MDT'] (176 mi)
JFK -> BDL: ['JFK', 'PHL', 'EWR', 'BDL'] (287 mi)
JFK -> ISP: ['JFK', 'PHL', 'ISP'] (222 mi)
JFK -> SWF: ['JFK', 'PHL', 'SWF'] (220 mi)
JFK -> ABE: ['JFK', 'PHL', 'ABE'] (148 mi)
JFK -> AVP: ['JFK', 'PHL', 'AVP'] (197 mi)
JFK -> PHF: ['JFK', 'PHL', 'PHF'] (294 mi)
JFK -> ELM: ['JFK', 'PHL', 'ELM'] (273 mi)
JFK -> SCE: ['JFK', 'PHL', 'SCE'] (246 mi)
JFK -> BGM: ['JFK', 'PHL', 'BGM'] (259 mi)
JFK -> ITH: ['JFK', 'PHL', 'ITH'] (285 mi)
JFK -> HVN: ['JFK', 'PHL', 'HVN'] (249 mi)
JFK -> IPT: ['JFK', 'PHL', 'IPT'] (222 mi)
JFK -> SBY: ['JFK', 'PHL', 'SBY'] (200 mi)
JFK -> LEB: ['JFK', 'BOS', 'LEB'] (295 mi)
JFK -> MVY: ['JFK', 'ACK', 'MVY'] (228 mi)
JFK -> PVC: ['JFK', 'BOS', 'PVC'] (231 mi)
JFK -> EWB: ['JFK', 'ACK', 'EWB'] (253 mi)
JFK -> HGR: ['JFK', 'IAD', 'HGR'] (282 mi)
```

Calculate which cities can be reached from a starting, user given city within a reach of also user given distance (in miles).

In [13]:

```
from_city = input("From city: ")
max_distance = int(input("Max distance: "))

from_airports = airports[airports['city'] == from_city].index
result = []
for from_airport in from_airports:
    routes = nx.single_source_dijkstra_path(flight_graph, from_airport, max_dist
ance, 'distance')
    to_airports = routes.keys()
    to_cities = [airports.loc[ap]['city'] for ap in to_airports]
    result += to_cities

result_unique = set(result) # remove duplicates
print(sorted(result_unique)) # sort the printed result
```

```
['Allentown', 'Baltimore', 'Binghamton', 'Boston', 'Buffalo', 'Burli
ngton', 'Elmira/Corning', 'Hagerstown', 'Harrisburg', 'Hartford', 'H
yannis', 'Islip', 'Ithaca', 'Lebanon', 'Manchester', "Martha's Viney
ard", 'Nantucket', 'New Bedford', 'New Haven', 'New York', 'Newark',
'Newburgh', 'Newport News', 'Norfolk', 'Philadelphia', 'Portland',
'Provincetown', 'Richmond', 'Rochester', 'Salisbury', 'State Colleg
e', 'Syracuse', 'Washington D.C.', 'White Plains', 'Wilkes-Barre/Scr
anton', 'Williamsport', 'Worcester']
```