Artificial Intelligence (CS303)

Practice 2

Requirements

- Graph: define a weighed directed-graph
- start: start node in the search
- end: end node in the search
- distances

Implement A* search in python

1. Read the graph file and initialize *Graph*, *start*, *end*, *distances* ;
2. *res* = AStarSearch(*Graph*,*start*,*end*,*distances*) ;
3. Visualize the search process iteratively with *res*, which specifies the temporary search tree at the current search step ;
4. Print the final route (*result*) found by A* search.

Requirements

Implement A* search in python

provided in the test block

1. Read the graph file and initialize *Graph*, *start*, *end*, *distances* ;

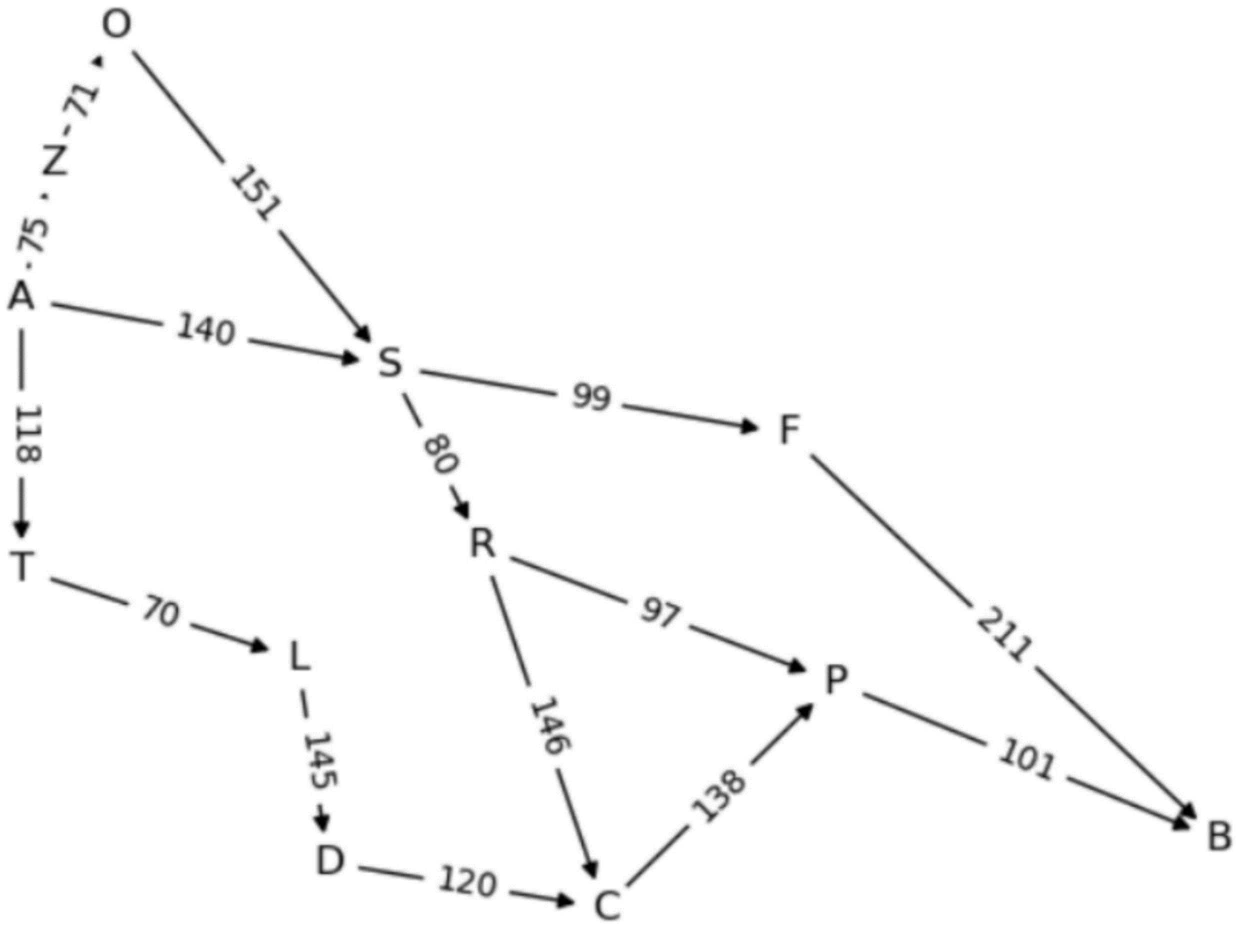
2. *res* = AStarSearch(*Graph*,*start*,*end*,*distances*) ;

3. Visualize the search process iteratively with *res*, which specifies the temporary search tree at the current search step ;

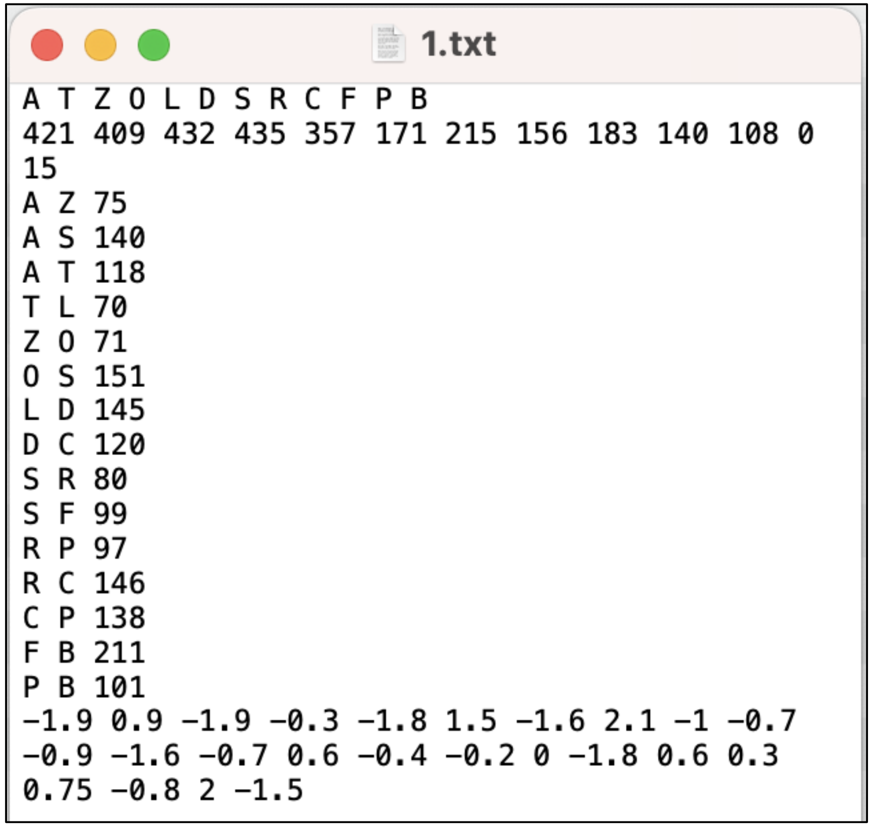
4. Print the final route (*result*) found by A* search.

Read the graph file and Initialize

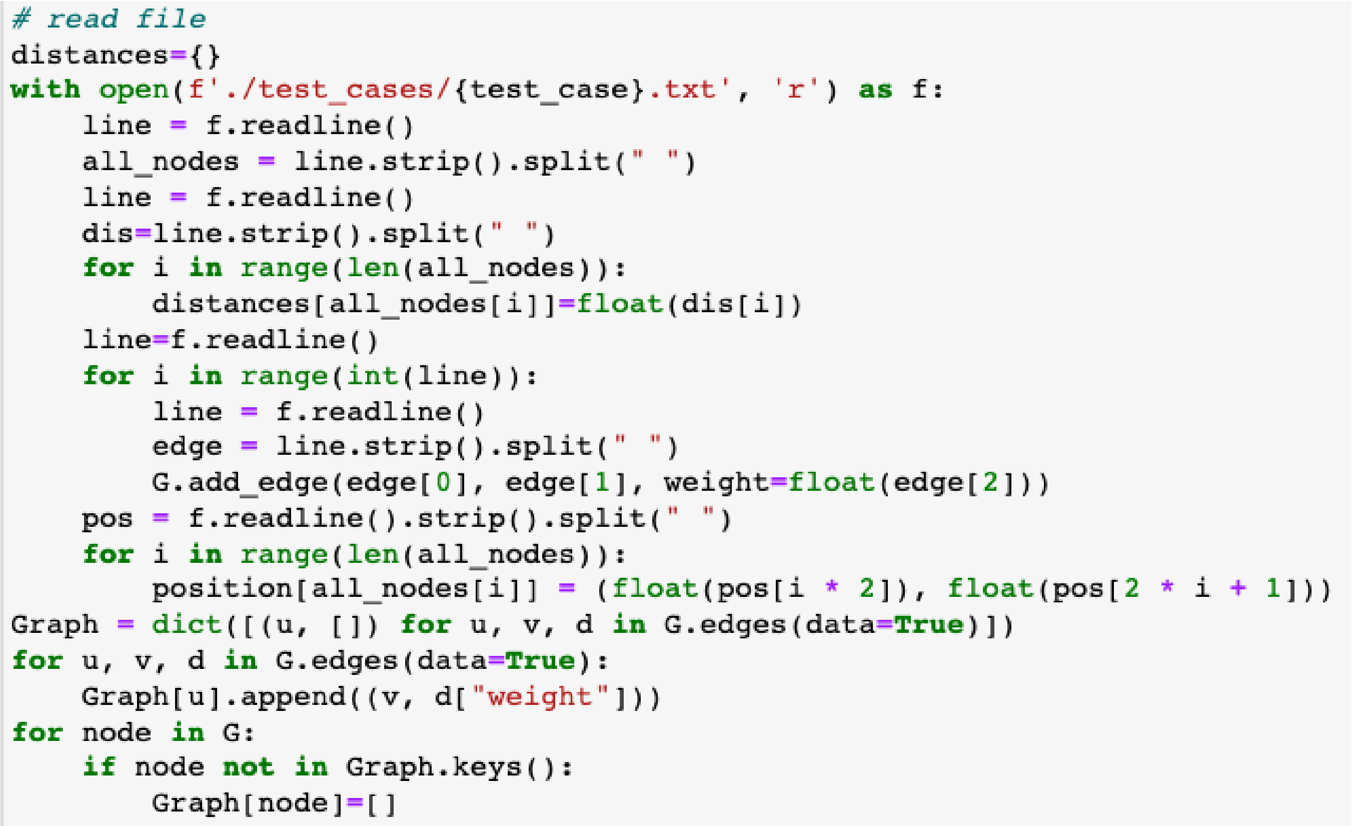
- A graph example



Read the graph file and Initialize



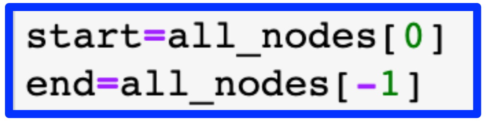
```
A T Z O L D S R C F P B
421 409 432 435 357 171 215 156 183 140 108 0
15
A Z 75
A S 140
A T 118
T L 70
Z O 71
O S 151
L D 145
D C 120
S R 80
S F 99
R P 97
R C 146
C P 138
F B 211
P B 101
-1.9 0.9 -1.9 -0.3 -1.8 1.5 -1.6 2.1 -1 -0.7
-0.9 -1.6 -0.7 0.6 -0.4 -0.2 0 -1.8 0.6 0.3
0.75 -0.8 2 -1.5
```



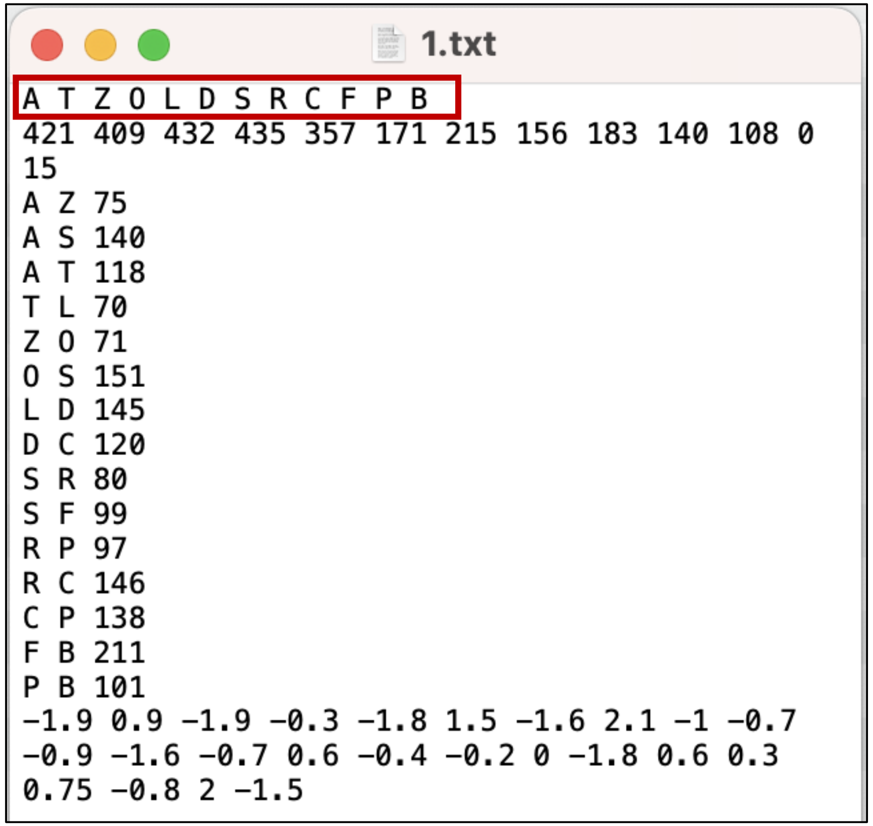
```
# read file
distances={}
with open(f'./test_cases/{test_case}.txt', 'r') as f:
    line = f.readline()
    all_nodes = line.strip().split(" ")
    line = f.readline()
    dis=line.strip().split(" ")
    for i in range(len(all_nodes)):
        distances[all_nodes[i]]=float(dis[i])
    line=f.readline()
    for i in range(int(line)):
        line = f.readline()
        edge = line.strip().split(" ")
        G.add_edge(edge[0], edge[1], weight=float(edge[2]))
    pos = f.readline().strip().split(" ")
    for i in range(len(all_nodes)):
        position[all_nodes[i]] = (float(pos[i * 2]), float(pos[2 * i + 1]))
Graph = dict([(u, []) for u, v, d in G.edges(data=True)])
for u, v, d in G.edges(data=True):
    Graph[u].append((v, d["weight"]))
for node in G:
    if node not in Graph.keys():
        Graph[node]=[]
```

Read the graph file and Initialize

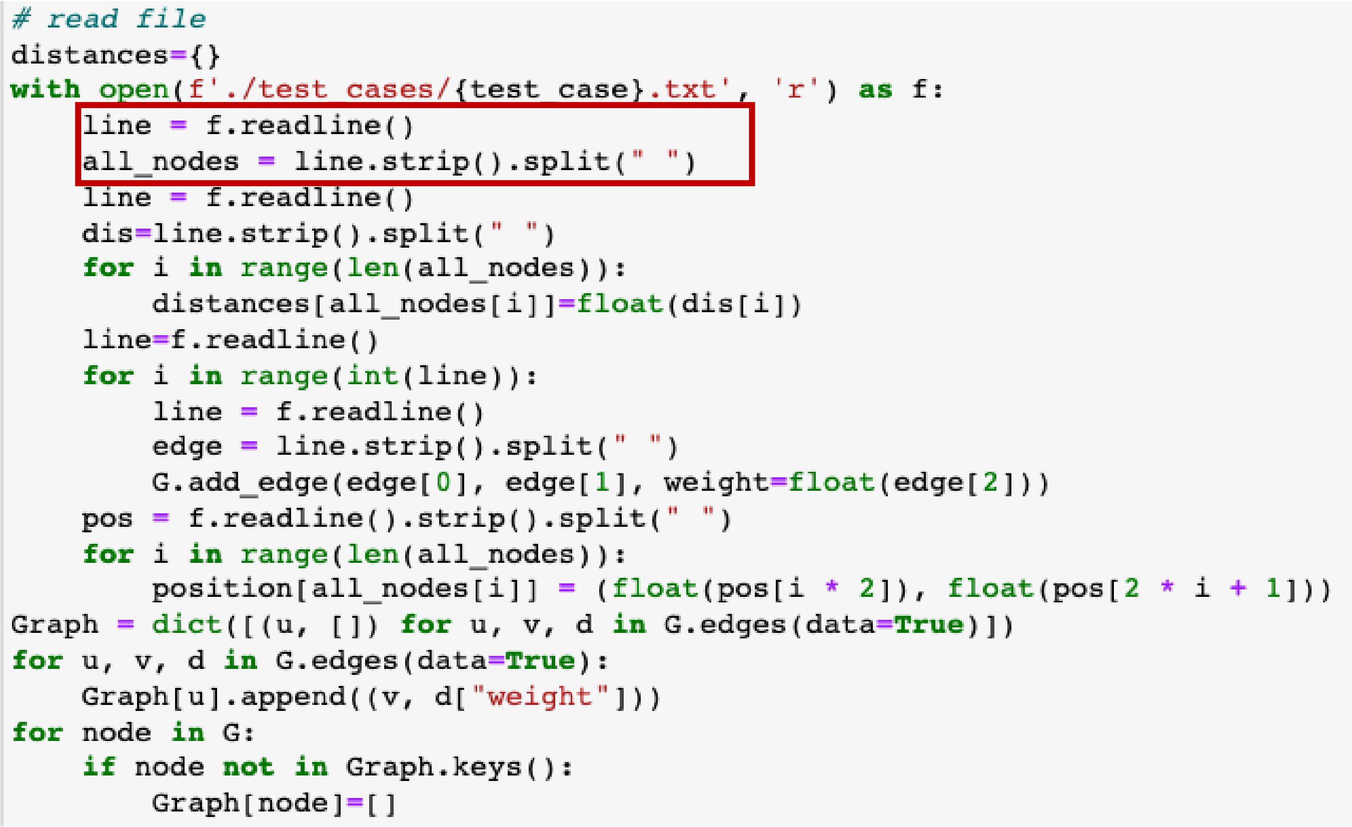
12 nodes



```
start=all_nodes[0]
end=all_nodes[-1]
```



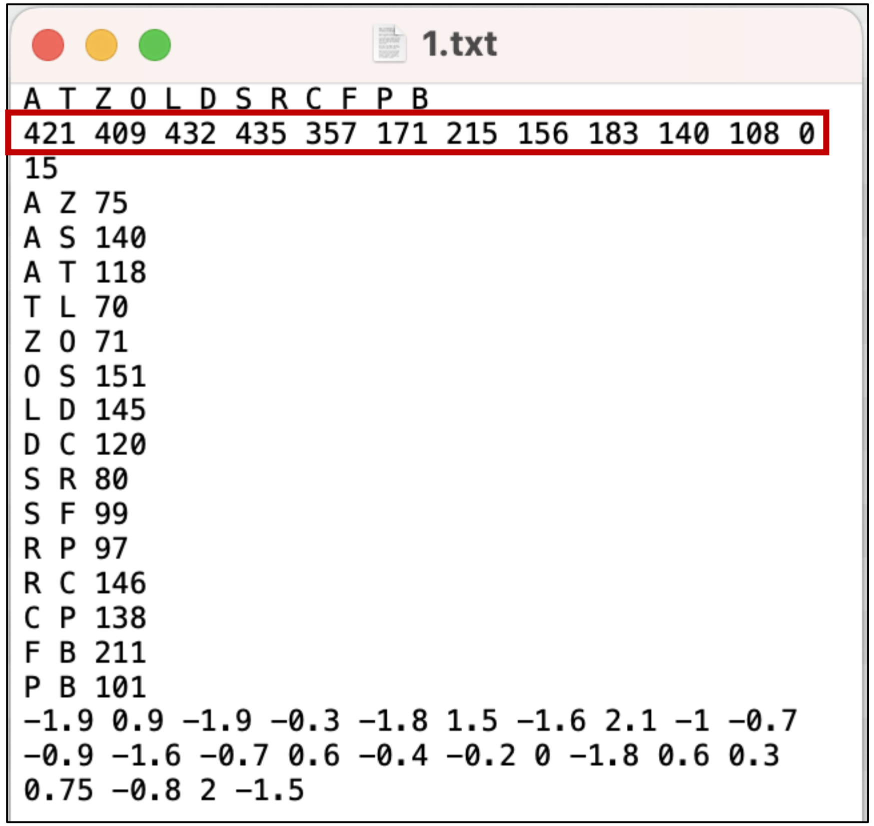
```
A T Z O L D S R C F P B
421 409 432 435 357 171 215 156 183 140 108 0
15
A Z 75
A S 140
A T 118
T L 70
Z O 71
O S 151
L D 145
D C 120
S R 80
S F 99
R P 97
R C 146
C P 138
F B 211
P B 101
-1.9 0.9 -1.9 -0.3 -1.8 1.5 -1.6 2.1 -1 -0.7
-0.9 -1.6 -0.7 0.6 -0.4 -0.2 0 -1.8 0.6 0.3
0.75 -0.8 2 -1.5
```



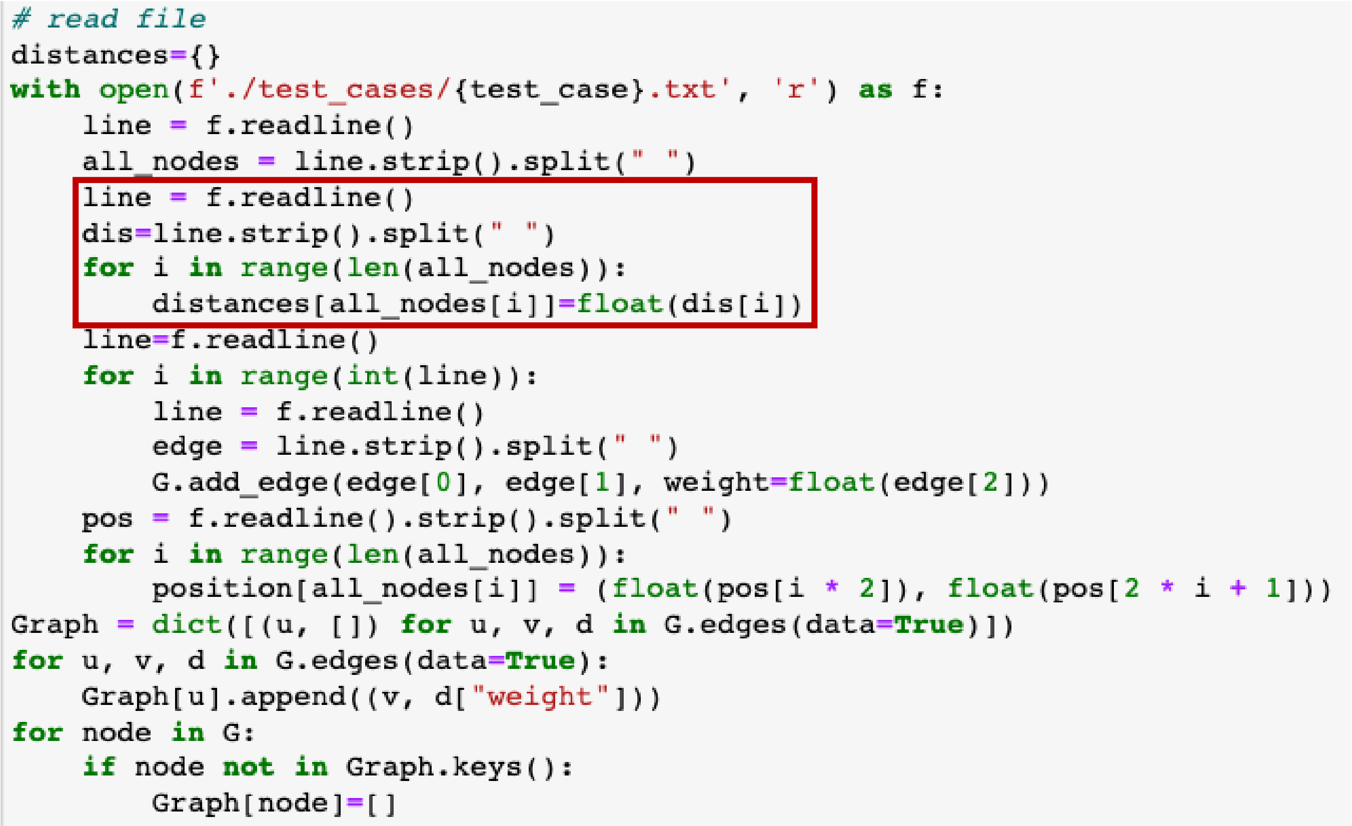
```
# read file
distances={}
with open(f'./test cases/{test_case}.txt', 'r') as f:
    line = f.readline()
    all_nodes = line.strip().split(" ")
    line = f.readline()
    dis=line.strip().split(" ")
    for i in range(len(all_nodes)):
        distances[all_nodes[i]]=float(dis[i])
    line=f.readline()
    for i in range(int(line)):
        line = f.readline()
        edge = line.strip().split(" ")
        G.add_edge(edge[0], edge[1], weight=float(edge[2]))
    pos = f.readline().strip().split(" ")
    for i in range(len(all_nodes)):
        position[all_nodes[i]] = (float(pos[i * 2]), float(pos[2 * i + 1]))
Graph = dict([(u, []) for u, v, d in G.edges(data=True)])
for u, v, d in G.edges(data=True):
    Graph[u].append((v, d["weight"]))
for node in G:
    if node not in Graph.keys():
        Graph[node]=[]
```

Read the graph file and Initialize

distances: distance from each node to the end node

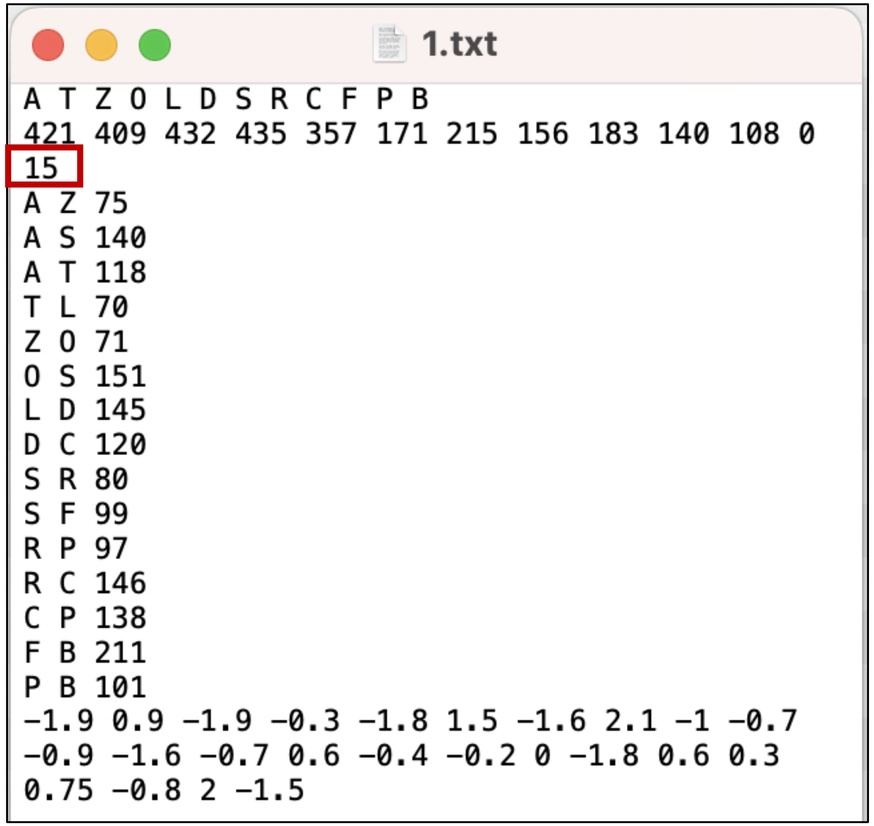


```
A T Z O L D S R C F P B
421 409 432 435 357 171 215 156 183 140 108 0
15
A Z 75
A S 140
A T 118
T L 70
Z O 71
O S 151
L D 145
D C 120
S R 80
S F 99
R P 97
R C 146
C P 138
F B 211
P B 101
-1.9 0.9 -1.9 -0.3 -1.8 1.5 -1.6 2.1 -1 -0.7
-0.9 -1.6 -0.7 0.6 -0.4 -0.2 0 -1.8 0.6 0.3
0.75 -0.8 2 -1.5
```



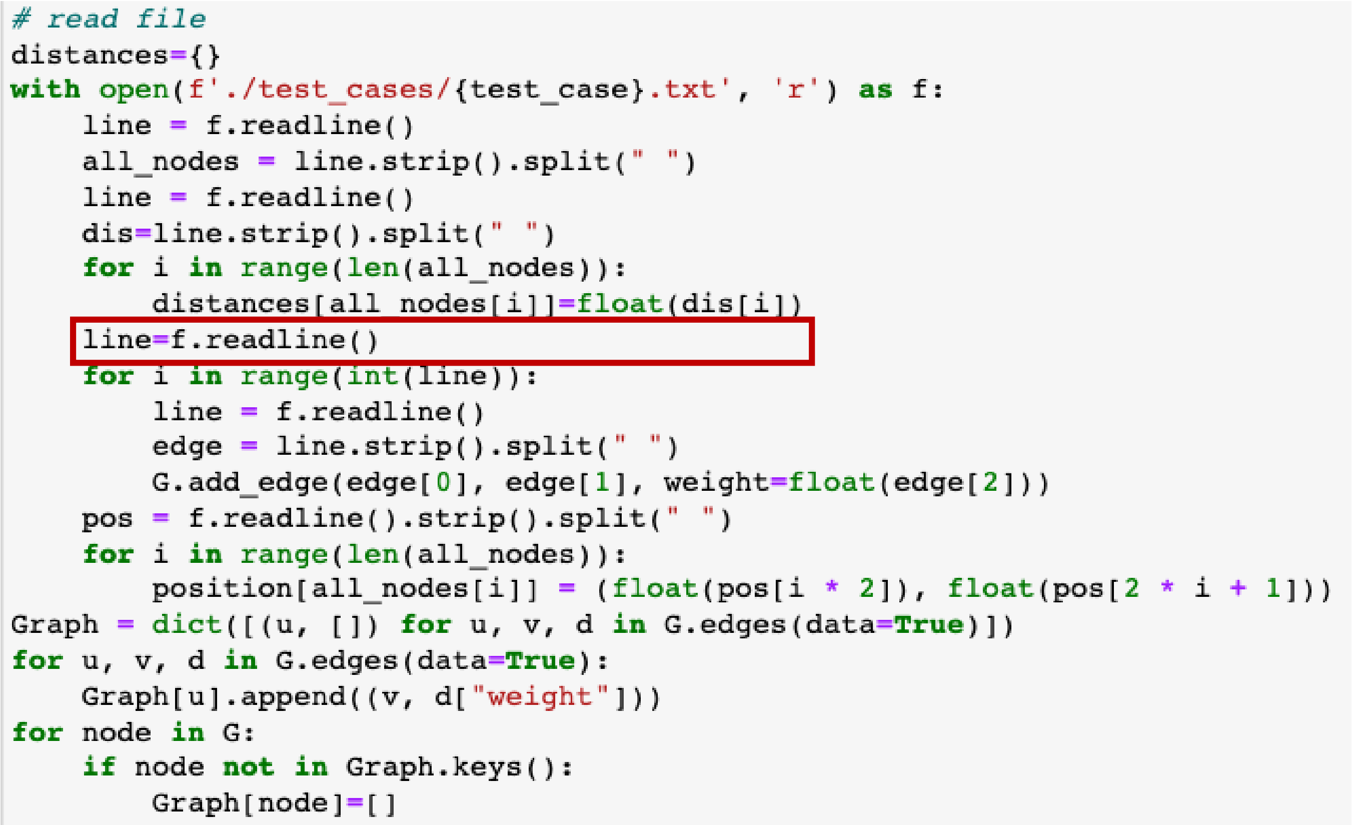
```python
# read file
distances={}
with open(f'./test_cases/{test_case}.txt', 'r') as f:
    line = f.readline()
    all_nodes = line.strip().split(" ")
    line = f.readline()
    dis=line.strip().split(" ")
    for i in range(len(all_nodes)):
        distances[all_nodes[i]]=float(dis[i])
    line=f.readline()
    for i in range(int(line)):
        line = f.readline()
        edge = line.strip().split(" ")
        G.add_edge(edge[0], edge[1], weight=float(edge[2]))
    pos = f.readline().strip().split(" ")
    for i in range(len(all_nodes)):
        position[all_nodes[i]] = (float(pos[i * 2]), float(pos[2 * i + 1]))
Graph = dict([(u, []) for u, v, d in G.edges(data=True)])
for u, v, d in G.edges(data=True):
    Graph[u].append((v, d["weight"]))
for node in G:
    if node not in Graph.keys():
        Graph[node]=[]
```

Read the graph file and Initialize



```
A T Z O L D S R C F P B
421 409 432 435 357 171 215 156 183 140 108 0
15
A Z 75
A S 140
A T 118
T L 70
Z O 71
O S 151
L D 145
D C 120
S R 80
S F 99
R P 97
R C 146
C P 138
F B 211
P B 101
-1.9 0.9 -1.9 -0.3 -1.8 1.5 -1.6 2.1 -1 -0.7
-0.9 -1.6 -0.7 0.6 -0.4 -0.2 0 -1.8 0.6 0.3
0.75 -0.8 2 -1.5
```

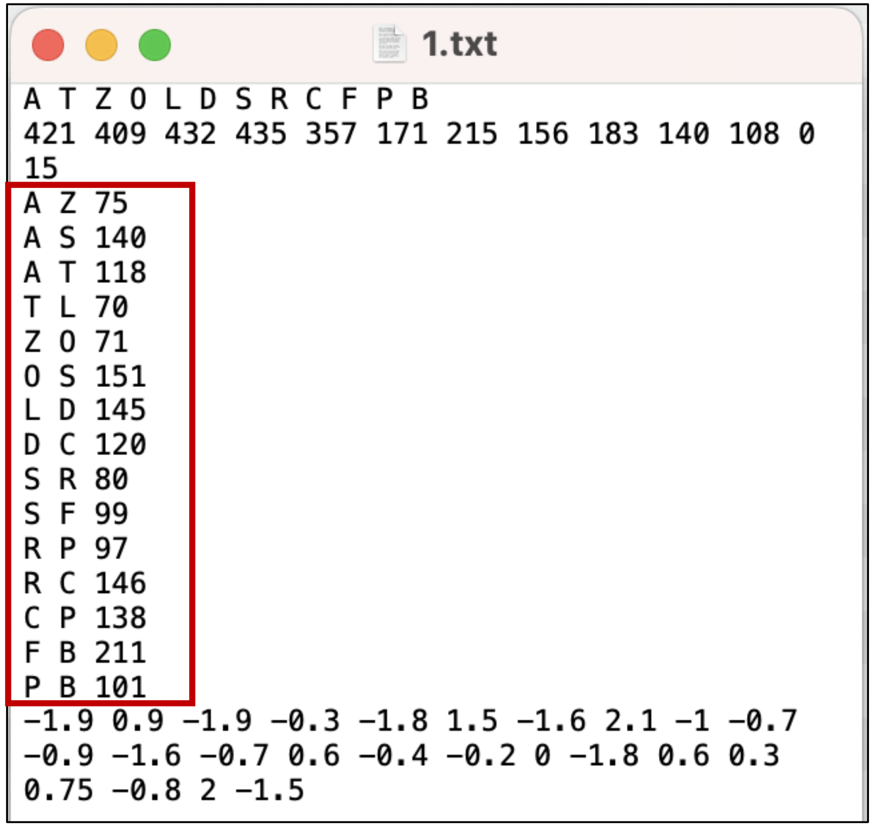
Number of edges



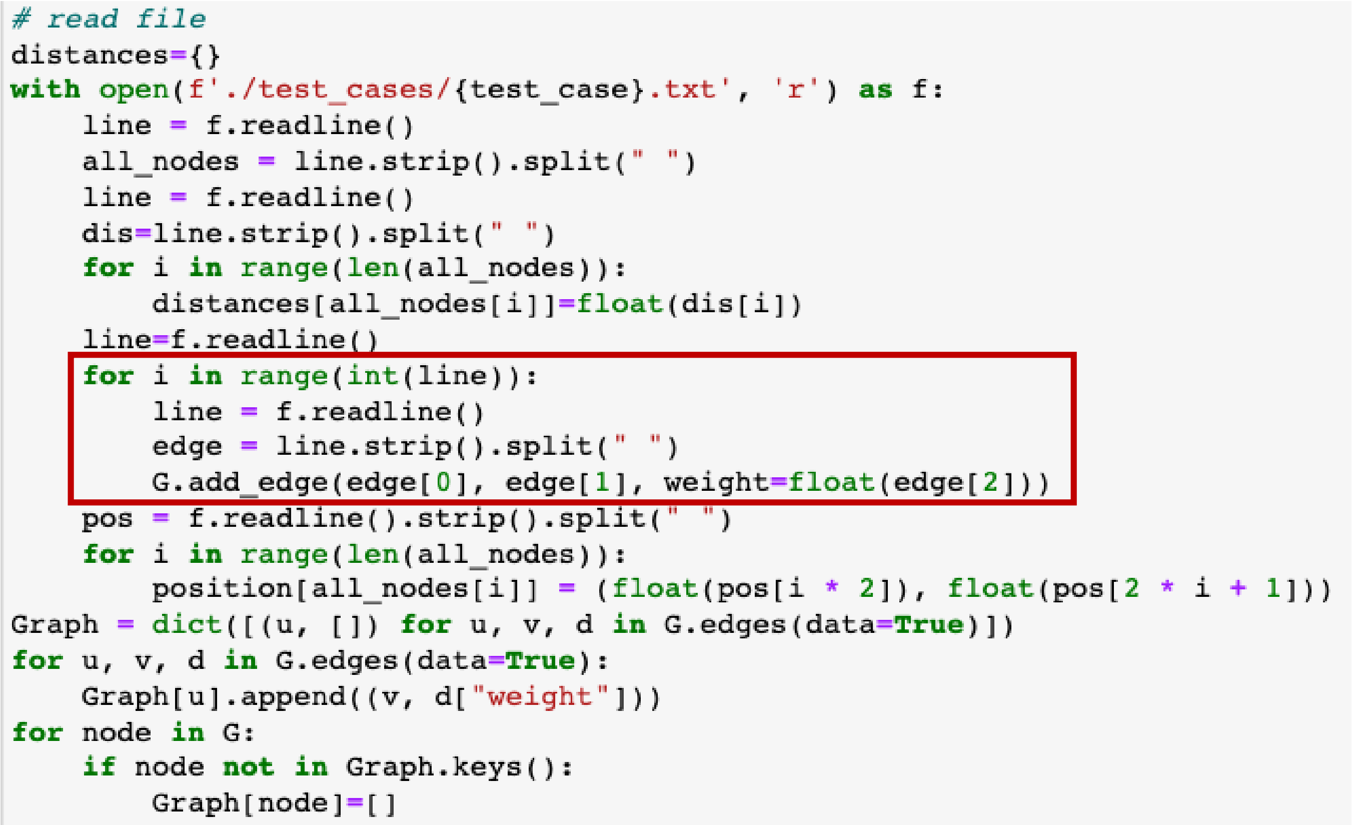
```python
# read file
distances={}
with open(f'./test_cases/{test_case}.txt', 'r') as f:
    line = f.readline()
    all_nodes = line.strip().split(" ")
    line = f.readline()
    dis=line.strip().split(" ")
    for i in range(len(all_nodes)):
        distances[all_nodes[i]]=float(dis[i])
    line=f.readline()
    for i in range(int(line)):
        line = f.readline()
        edge = line.strip().split(" ")
        G.add_edge(edge[0], edge[1], weight=float(edge[2]))
    pos = f.readline().strip().split(" ")
    for i in range(len(all_nodes)):
        position[all_nodes[i]] = (float(pos[i * 2]), float(pos[2 * i + 1]))
Graph = dict([(u, []) for u, v, d in G.edges(data=True)])
for u, v, d in G.edges(data=True):
    Graph[u].append((v, d["weight"]))
for node in G:
    if node not in Graph.keys():
        Graph[node]=[]
```

Read the graph file and Initialize

Edge (start, end, weight)



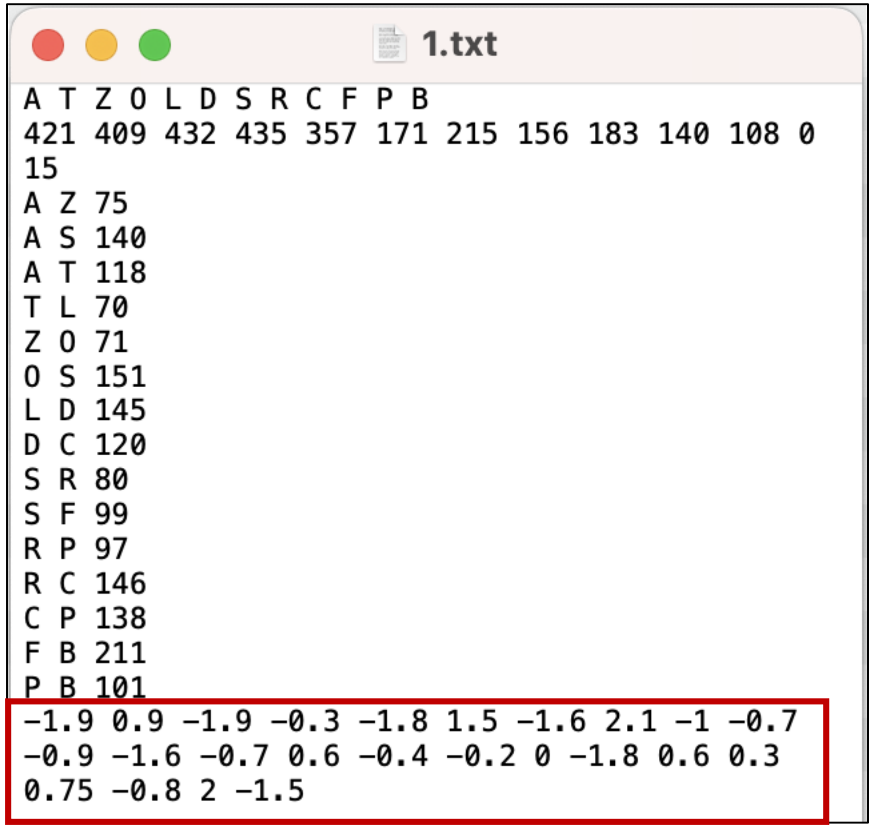
```
A T Z O L D S R C F P B
421 409 432 435 357 171 215 156 183 140 108 0
15
A Z 75
A S 140
A T 118
T L 70
Z O 71
O S 151
L D 145
D C 120
S R 80
S F 99
R P 97
R C 146
C P 138
F B 211
P B 101
-1.9 0.9 -1.9 -0.3 -1.8 1.5 -1.6 2.1 -1 -0.7
-0.9 -1.6 -0.7 0.6 -0.4 -0.2 0 -1.8 0.6 0.3
0.75 -0.8 2 -1.5
```



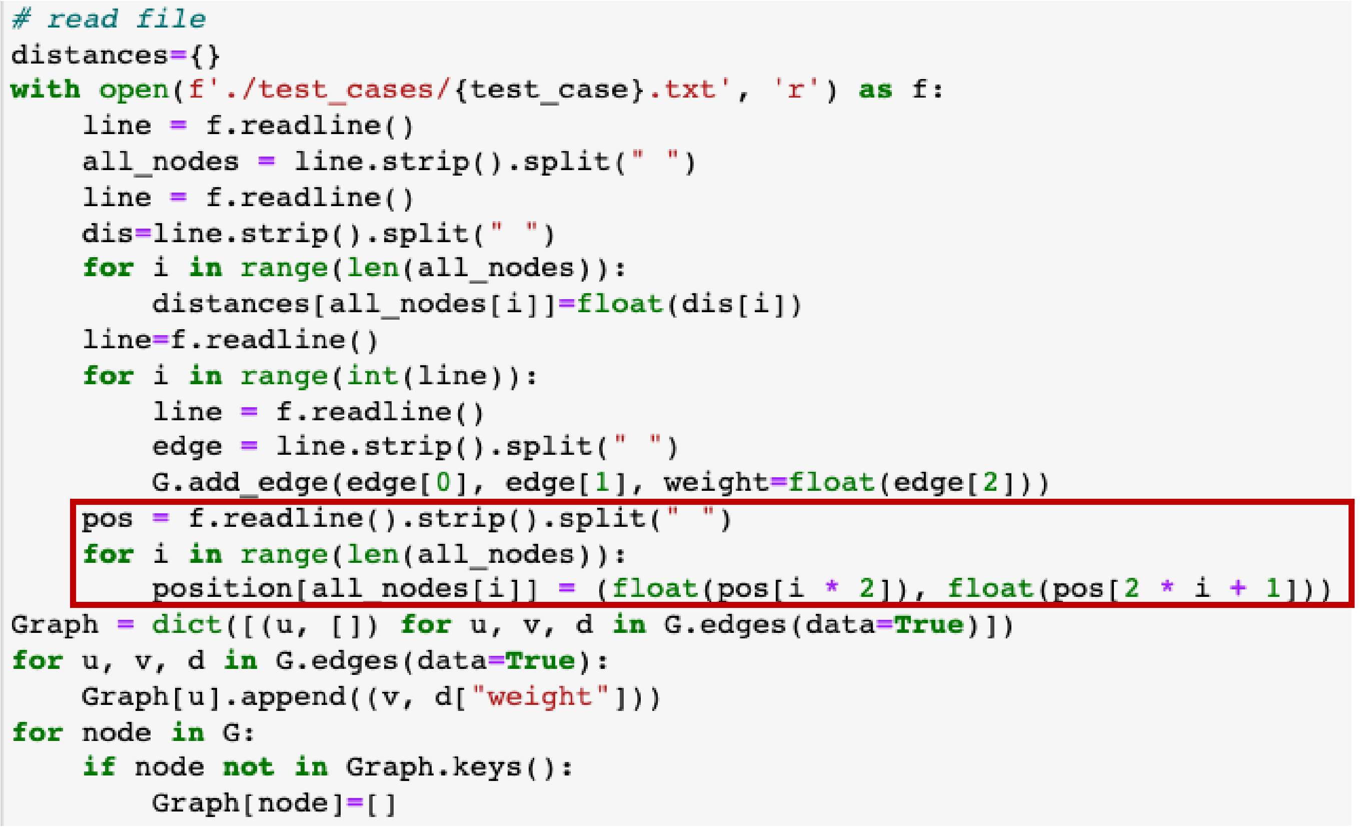
```
# read file
distances={}
with open(f'./test_cases/{test_case}.txt', 'r') as f:
    line = f.readline()
    all_nodes = line.strip().split(" ")
    line = f.readline()
    dis=line.strip().split(" ")
    for i in range(len(all_nodes)):
        distances[all_nodes[i]]=float(dis[i])
    line=f.readline()
    for i in range(int(line)):
        line = f.readline()
        edge = line.strip().split(" ")
        G.add_edge(edge[0], edge[1], weight=float(edge[2]))
    pos = f.readline().strip().split(" ")
    for i in range(len(all_nodes)):
        position[all_nodes[i]] = (float(pos[i * 2]), float(pos[2 * i + 1]))
Graph = dict([(u, []) for u, v, d in G.edges(data=True)])
for u, v, d in G.edges(data=True):
    Graph[u].append((v, d["weight"]))
for node in G:
    if node not in Graph.keys():
        Graph[node]=[]
```

Read the graph file and Initialize

Coordinate (x, y) of each node in the plot



```
A T Z O L D S R C F P B
421 409 432 435 357 171 215 156 183 140 108 0
15
A Z 75
A S 140
A T 118
T L 70
Z O 71
O S 151
L D 145
D C 120
S R 80
S F 99
R P 97
R C 146
C P 138
F B 211
P B 101
-1.9 0.9 -1.9 -0.3 -1.8 1.5 -1.6 2.1 -1 -0.7
-0.9 -1.6 -0.7 0.6 -0.4 -0.2 0 -1.8 0.6 0.3
0.75 -0.8 2 -1.5
```



```
# read file
distances={}
with open(f'./test_cases/{test_case}.txt', 'r') as f:
    line = f.readline()
    all_nodes = line.strip().split(" ")
    line = f.readline()
    dis=line.strip().split(" ")
    for i in range(len(all_nodes)):
        distances[all_nodes[i]]=float(dis[i])
    line=f.readline()
    for i in range(int(line)):
        line = f.readline()
        edge = line.strip().split(" ")
        G.add_edge(edge[0], edge[1], weight=float(edge[2]))
    pos = f.readline().strip().split(" ")
    for i in range(len(all_nodes)):
        position[all_nodes[i]] = (float(pos[i * 2]), float(pos[2 * i + 1]))
Graph = dict([(u, []) for u, v, d in G.edges(data=True)])
for u, v, d in G.edges(data=True):
    Graph[u].append((v, d["weight"]))
for node in G:
    if node not in Graph.keys():
        Graph[node]=[]
```

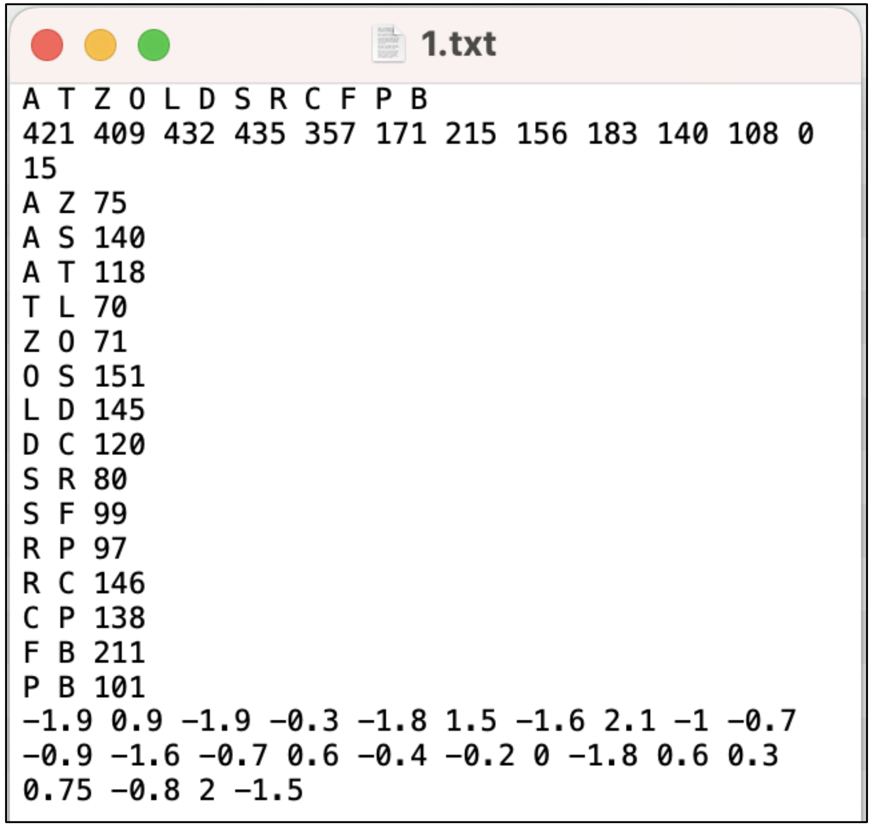
Read the graph file and Initialize

Create Graph

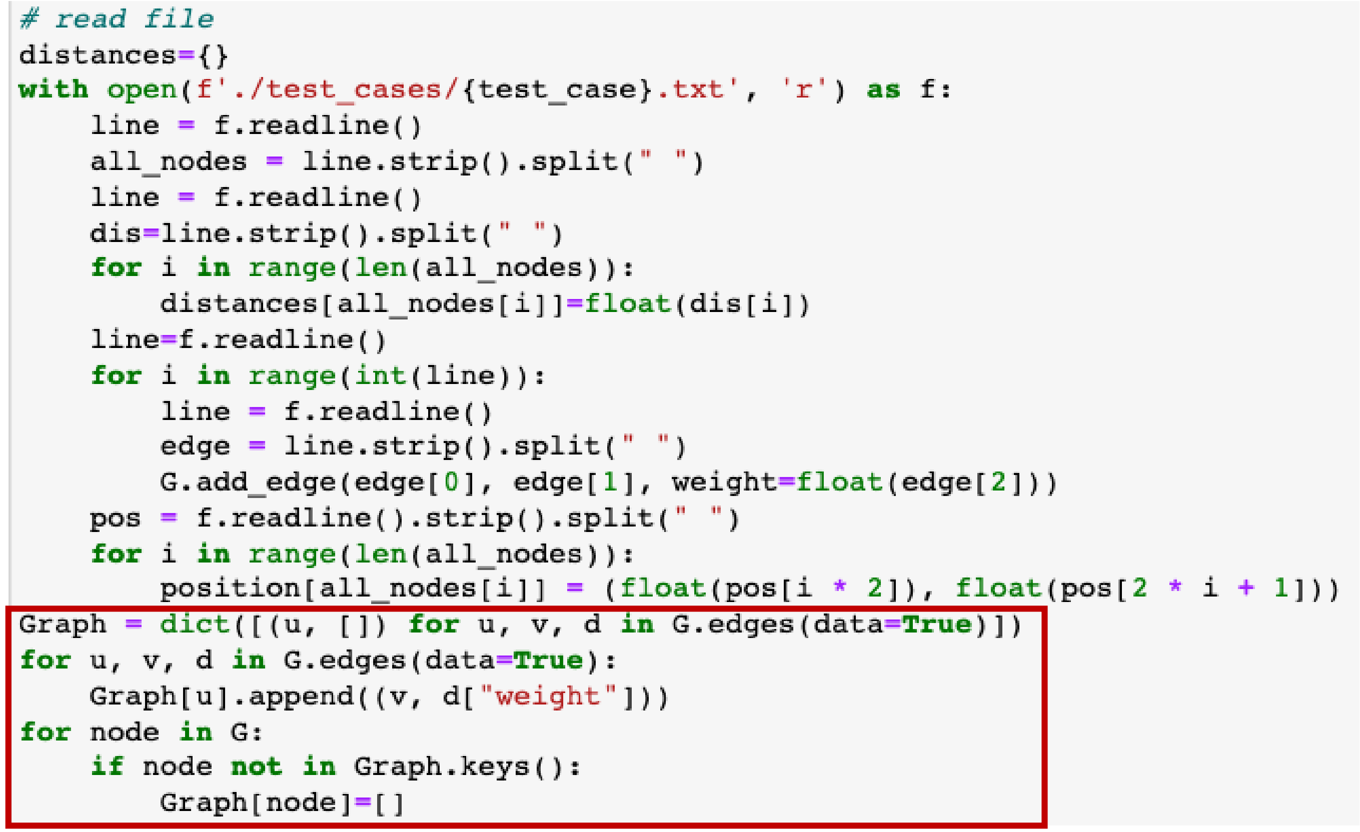
key: node

value: (end node, weight)

e.g., Graph["S"]:[('R', '80'), ('F', '99')]



```
A T Z O L D S R C F P B
421 409 432 435 357 171 215 156 183 140 108 0
15
A Z 75
A S 140
A T 118
T L 70
Z O 71
O S 151
L D 145
D C 120
S R 80
S F 99
R P 97
R C 146
C P 138
F B 211
P B 101
-1.9 0.9 -1.9 -0.3 -1.8 1.5 -1.6 2.1 -1 -0.7
-0.9 -1.6 -0.7 0.6 -0.4 -0.2 0 -1.8 0.6 0.3
0.75 -0.8 2 -1.5
```



```python
# read file
distances={}
with open(f'./test_cases/{test_case}.txt', 'r') as f:
    line = f.readline()
    all_nodes = line.strip().split(" ")
    line = f.readline()
    dis=line.strip().split(" ")
    for i in range(len(all_nodes)):
        distances[all_nodes[i]]=float(dis[i])
    line=f.readline()
    for i in range(int(line)):
        line = f.readline()
        edge = line.strip().split(" ")
        G.add_edge(edge[0], edge[1], weight=float(edge[2]))
    pos = f.readline().strip().split(" ")
    for i in range(len(all_nodes)):
        position[all_nodes[i]] = (float(pos[i * 2]), float(pos[2 * i + 1]))
Graph = dict([(u, []) for u, v, d in G.edges(data=True)])
for u, v, d in G.edges(data=True):
    Graph[u].append((v, d["weight"]))
for node in G:
    if node not in Graph.keys():
        Graph[node]=[]
```

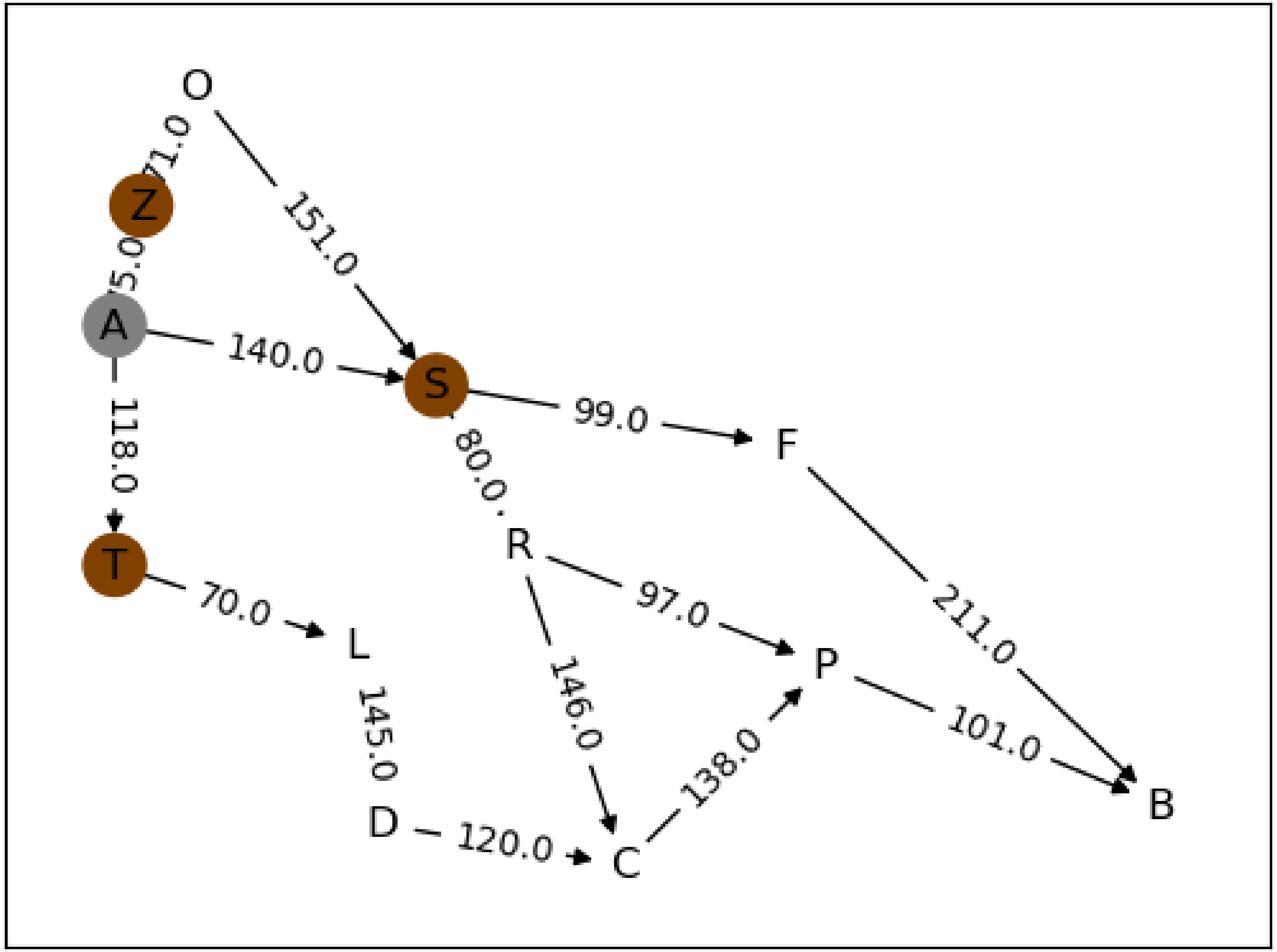
Visualization

- Search tree

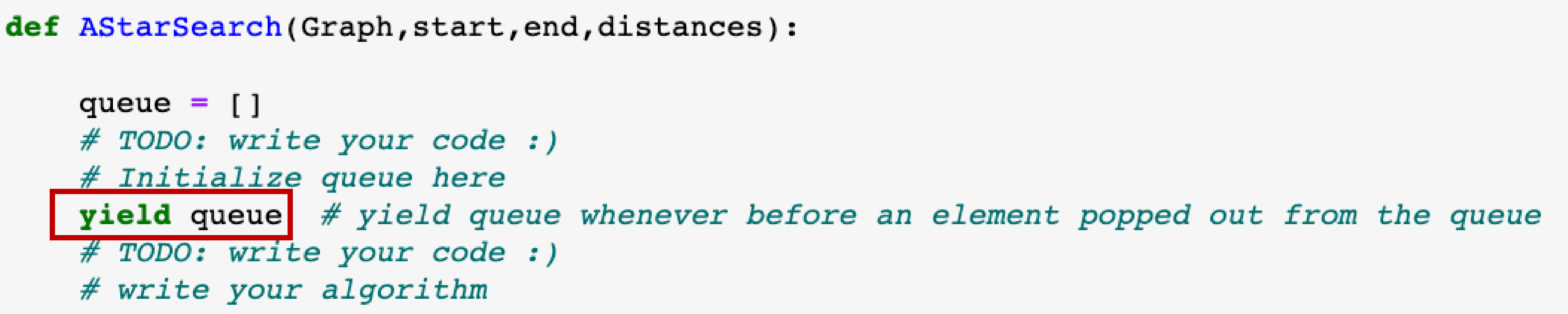
White: has not been visited

Brown : has been visited but not selected

Gray : selected



AStarSearch

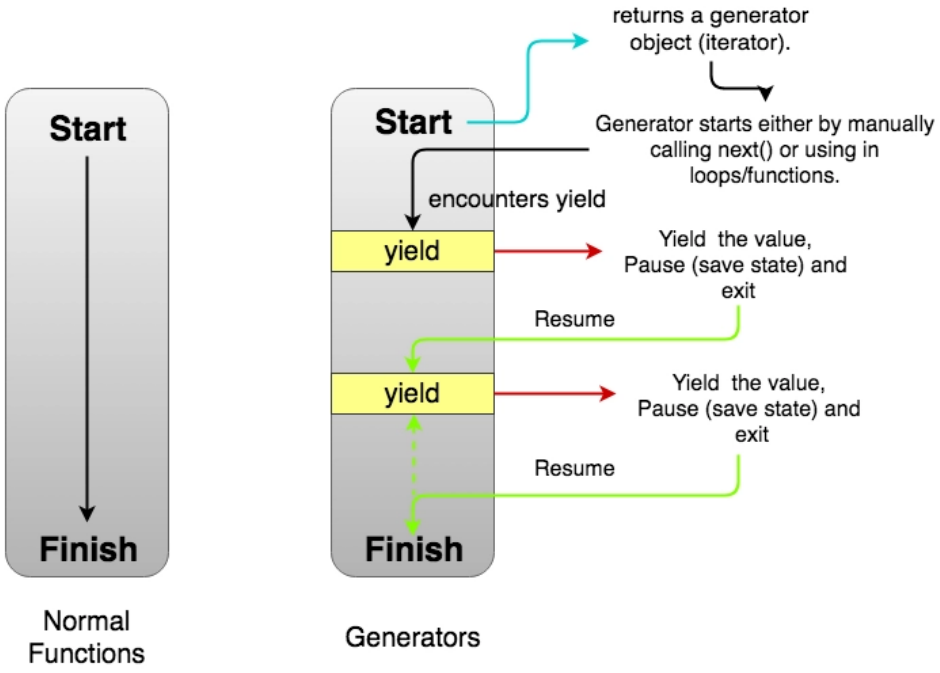


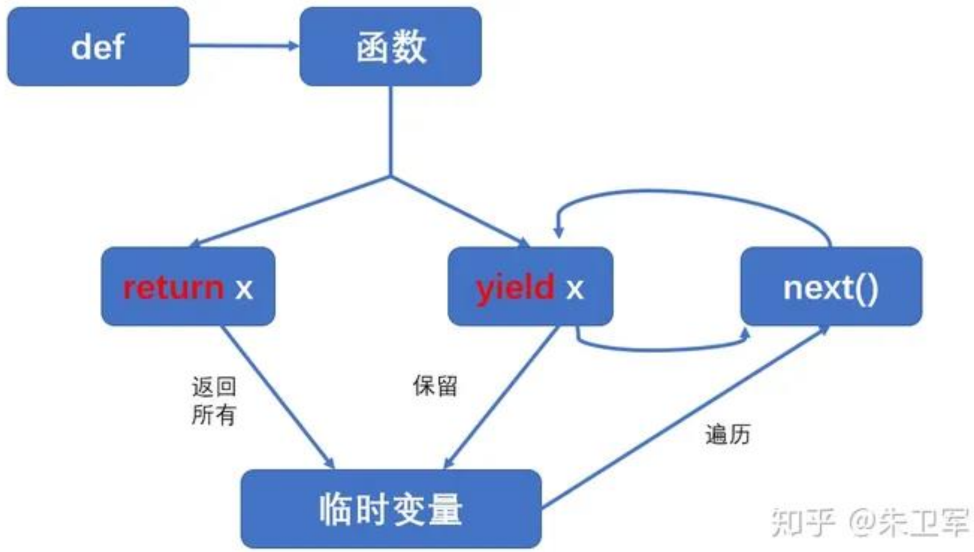
```
def AStarSearch(Graph,start,end,distances):

    queue = []
    # TODO: write your code :)
    # Initialize queue here
    yield queue  # yield queue whenever before an element popped out from the queue
    # TODO: write your code :)
    # write your algorithm
```

AStarSearch

- yield: generator


Start
Finish
Normal Functions
Start
returns a generator object (iterator).
Generator starts either by manually calling next() or using in loops/functions.
encounters yield
yield
Yield the value, Pause (save state) and exit
Resume
yield
Yield the value, Pause (save state) and exit
Resume
Finish
Generators



learnpython.py

Summary

Implement A* search in python

DDL: 22:00, Oct.13

1. Read the graph file and initialize *Graph*, *start*, *end*, *distances* ;
2. ***res* = AStarSearch(*Graph*,*start*,*end*,*distances*) ;**
3. Visualize the search process iteratively with *res*, which specifies the temporary search tree at the current search step ;
4. Print the final route (*result*) found by A* search.