Best First Search

GRAPH = {\

'Arad': {'Sibiu': 140, 'Zerind': 75, 'Timisoara': 118},\

'Zerind': {'Arad': 75, 'Oradea': 71},\

'Oradea': {'Zerind': 71, 'Sibiu': 151},\

'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu': 80},\

'Timisoara': {'Arad': 118, 'Lugoj': 111},\

'Lugoj': {'Timisoara': 111, 'Mehadia': 70},\

'Mehadia': {'Lugoj': 70, 'Drobeta': 75},\

'Drobeta': {'Mehadia': 75, 'Craiova': 120},\

'Craiova': {'Drobeta': 120, 'Rimnicu': 146, 'Pitesti': 138},\

'Rimnicu': {'Sibiu': 80, 'Craiova': 146, 'Pitesti': 97},\

'Fagaras': {'Sibiu': 99, 'Bucharest': 211},\

'Pitesti': {'Rimnicu': 97, 'Craiova': 138, 'Bucharest': 101},\

'Bucharest': {'Fagaras': 211, 'Pitesti': 101, 'Giurgiu': 90, 'Urziceni': 85},\

'Giurgiu': {'Bucharest': 90},\

'Urziceni': {'Bucharest': 85, 'Vaslui': 142, 'Hirsova': 98},\

'Hirsova': {'Urziceni': 98, 'Eforie': 86},\

'Eforie': {'Hirsova': 86},\

'Vaslui': {'Iasi': 92, 'Urziceni': 142},\

'Iasi': {'Vaslui': 92, 'Neamt': 87},\

'Neamt': {'Iasi': 87}\

}

def bestfirst(source, destination):

"""Optimal path from source to destination using straight line distance heuristic

:param source: Source city name

:param destination: Destination city name

:returns: Heuristic value, cost and path for optimal traversal”””

# HERE THE STRAIGHT LINE DISTANCE VALUES ARE IN REFERENCE TO BUCHAREST AS THE DESTINATION

straight\_line = {\

'Arad': 366,\

'Zerind': 374,\

'Oradea': 380,\

'Sibiu': 253,\

'Timisoara': 329,\

'Lugoj': 244,\

'Mehadia': 241,\

'Drobeta': 242,\

'Craiova': 160,\

'Rimnicu': 193,\

'Fagaras': 176,\

'Pitesti': 100,\

'Bucharest': 0,\

'Giurgiu': 77,\

'Urziceni': 80,\

'Hirsova': 151,\

'Eforie': 161,\

'Vaslui': 199,\

'Iasi': 226,\

'Neamt': 234\

}

from queue import PriorityQueue

priority\_queue, visited = PriorityQueue(), {}

priority\_queue.put((straight\_line[source], 0, source, [source]))

visited[source] = straight\_line[source]

while not priority\_queue.empty():

(heuristic, cost, vertex, path) = priority\_queue.get()

if vertex == destination:

return heuristic, cost, path

for next\_node in GRAPH[vertex].keys():

current\_cost = cost + GRAPH[vertex][next\_node]

heuristic = straight\_line[next\_node]

if not next\_node in visited or visited[next\_node] >= heuristic:

visited[next\_node] = heuristic

priority\_queue.put((heuristic, current\_cost,next\_node, path + [next\_node]))

def main():

"""Main function"""

print('ENTER SOURCE :', end=' ')

source = input().strip()

print('ENTER GOAL :', end=' ')

goal = input().strip()

if source not in GRAPH or goal not in GRAPH:

print('ERROR: CITY DOES NOT EXIST.')

else:

print('\nBFS PATH:')

heuristic, cost, optimal\_path = bestfirst(source, goal)

print('PATH COST =', cost)

print(' -> '.join(city for city in optimal\_path))

if \_\_name\_\_ == '\_\_main\_\_':

main()

OUTPUT

ENTER SOURCE : Sibiu

ENTER GOAL : Bucharest

BFS PATH:

PATH COST = 310

Sibiu -> Fagaras -> Bucharest