**Αναφορά: Εύρεση της Πιο Σύντομης Διαδρομής σε Δίκτυο Δρομολογίων**

**Εισαγωγή**

Αυτή η αναφορά περιγράφει την ανάπτυξη και εφαρμογή ενός αλγορίθμου που εντοπίζει την πιο σύντομη διαδρομή σε ένα δίκτυο δρομολογίων, χρησιμοποιώντας πραγματικά δεδομένα από το αρχείο **routes.csv**. Το αρχείο περιέχει πληροφορίες για διάφορους κόμβους (π.χ., πόλεις, αεροδρόμια) και τις μεταξύ τους συνδέσεις με διαφορετικά μέσα μετακίνησης.

**Δομή των Δεδομένων**

Το αρχείο **routes.csv** περιλαμβάνει τις εξής στήλες:

* **source**: Η αρχική τοποθεσία (πόλη/αεροδρόμιο).
* **destination**: Η τελική τοποθεσία.
* **type**: Το είδος της μετακίνησης (π.χ., boat, bus, plane, train, truck).
* **time**: Ο χρόνος μετάβασης.
* **cost**: Το κόστος της μετάβασης.

**Ανάγνωση των Δεδομένων**

Ο κώδικας αρχικά εισάγει και διαβάζει τα δεδομένα από το **routes.csv** μέσω της συνάρτησης read\_csv. Η συνάρτηση φροντίζει για την κανονικοποίηση των ονομάτων των πόλεων (μετατροπή σε κεφαλαία) και αγνοεί τυχόν ελλιπείς ή κατεστραμμένες γραμμές:

import csv

def read\_csv(filename):

csv\_data = []

with open(filename, mode='r', newline='', encoding='utf-8') as file:

reader = csv.reader(file)

for row in reader:

if len(row) < 5:

print(f"Skipping malformed row: {row}")

continue

from\_city = row[0].strip().upper()

to\_city = row[1].strip().upper()

travel\_mode = row[2].strip()

time = float(row[3])

cost = float(row[4])

csv\_data.append((from\_city, to\_city, travel\_mode, time, cost))

return csv\_data

**Δημιουργία του Γραφήματος**

Η συνάρτηση build\_graph δημιουργεί το γράφημα από τα δεδομένα που εισήχθησαν, προσθέτοντας διπλές εγγραφές για να θεωρηθεί το γράφημα αμφίδρομο:

def build\_graph(data):

graph = {}

for from\_city, to\_city, travel\_mode, time, cost in data:

if from\_city not in graph:

graph[from\_city] = []

if to\_city not in graph:

graph[to\_city] = []

graph[from\_city].append((to\_city, travel\_mode, time, cost))

graph[to\_city].append((from\_city, travel\_mode, time, cost))

return graph

**Υλοποίηση του Αλγορίθμου Dijkstra**

Ο αλγόριθμος Dijkstra χρησιμοποιείται για την εύρεση της συντομότερης διαδρομής. Η συνάρτηση dijkstra λαμβάνει ως παραμέτρους το γράφημα, την αρχική και τελική τοποθεσία, καθώς και το είδος βάρους (χρόνος ή κόστος):

import heapq

def dijkstra(graph, start, end, weight\_type="time"):

pq = [(0, start, [])] # Priority queue: (accumulated\_weight, current\_city, path)

visited = set()

while pq:

accumulated\_weight, current\_city, path = heapq.heappop(pq)

if current\_city in visited:

continue

visited.add(current\_city)

path = path + [current\_city]

if current\_city == end:

return accumulated\_weight, path

for neighbor, travel\_mode, time, cost in graph[current\_city]:

if neighbor not in visited:

weight = time if weight\_type == "time" else cost

heapq.heappush(pq, (accumulated\_weight + weight, neighbor, path))

return float("inf"), [] # Return infinity and empty path if no path found

**Παρουσίαση Αποτελεσμάτων**

Η εκτέλεση της κύριας συνάρτησης main() επιτρέπει στον χρήστη να εισαγάγει την αρχική και τελική τοποθεσία, επιστρέφοντας τις συντομότερες διαδρομές είτε με βάση τον χρόνο είτε με βάση το κόστος.

**Παράδειγμα Εκτέλεσης:**

* **Είσοδος**: Από την πόλη Α προς την πόλη Β.
* **Έξοδος**: Διαδρομή [Α, Γ, Β] με συνολικό χρόνο 5 ώρες ή κόστος 50 ευρώ.

**Συμπεράσματα**

Η υλοποίηση του αλγορίθμου Dijkstra επιτρέπει την εύρεση βέλτιστων διαδρομών με βάση διαφορετικά κριτήρια. Οι πραγματικές εφαρμογές του μπορούν να υποστηρίξουν συστήματα δρομολόγησης για επιβατικά και εμπορευματικά δίκτυα.