INTRODUCTION TO PYTHON MATH6005

GROUP COURSEWORK

DEADLINE: MONDAY, 08th JANUARY 2024, 06:00 PM

University Of Southampton

Data and Decision Analytics

Group Members:

Serial Number	Name	Student ID
1	Vyom Khanna	28965736
2	Zixu Hou	34955399
3	Jixin Zhao	34619232
4	Akanimoh Udoh	35177705

Contents

Introduction

- In the realm of graph theory and network optimization, Dijkstra's Algorithm stands as a cornerstone algorithm renowned for its pivotal role in solving the single-source shortest path problem.
- First conceived by Dutch computer scientist Edsger W. Dijkstra in 1956, the algorithm has since become a fundamental tool in computer science, operations research, and various engineering disciplines.
- Its efficacy lies in its ability to navigate through weighted graphs, determining the shortest path from a specified source node to all other nodes in the graph.
- As an algorithmic marvel, Dijkstra's Algorithm plays a critical role in diverse applications, including transportation network planning, routing protocols in computer networks, and resource allocation in distributed systems.
- This academic exploration aims to delve into the intricacies of Dijkstra's Algorithm, elucidating its underlying principles, algorithmic mechanisms, and the mathematical foundation that renders it a versatile and indispensable tool in solving complex optimization problems.
- By understanding the algorithm's structure and its inherent advantages and limitations, researchers and practitioners gain valuable insights into its applicability and significance in addressing real-world challenges.
- As we embark on this academic journey, we seek to unravel the algorithm's elegance, unraveling the theoretical constructs that empower its computational prowess and exploring the myriad contexts in which it finds utility.

Application of Dijkstra Algorithm

Dijkstra's Algorithm, celebrated for its efficiency in solving the single-source shortest path problem, finds application in a plethora of real-world scenarios across various domains. The algorithm's versatility and reliability make it an indispensable tool in addressing complex optimization challenges. Some prominent applications of Dijkstra's Algorithm include:

- Transportation Networks
- Telecommunication Networks
- Robotics and Autonomous Vehicles
- Resource Management in Distributed Systems
- Urban Planning and Traffic Management

- Critical Infrastructure Networks
- Airline and Flight Routing
- Supply Chain and Logistics

Advantages of Dijkstra Method

Dijkstra's Algorithm possesses several advantages that contribute to its widespread use in various applications. Here are some key advantages of Dijkstra's Algorithm:

- Optimality
- Applicability to Weighted Graphs
- Versatility
- Efficiency for Sparse Graphs
- Ease of Implementation
- Distributed Implementation
- Simplicity in Path Reconstruction
- No Negative Cycles

Limitations of Dijkstra Method

- Non-Negative Edge Weights
- No Handling of Negative Cycles
- Inapplicability to Networks with Changing Costs
- Memory and Computational Complexity
- Single-Source Shortest Path
- Not Suitable for All-Pairs Shortest Path Problem
- Greedy Nature
- Vulnerability to Graph Structure

Correctness of Dijkstra’s Algorithm

Key Property: A subpath of a shortest path is a shortest path.

Let p = s → v be a shortest path from vertex s to vertex v. For any subpath of p, let the path q = u → w from vertex u to vertex w. Then path q is the shortest path from vertex u to vertex w.

Proof. Suppose that the path from u to w is the subpath of the shortest path from s to v. Then $\delta(s, v) = d[s,u] + d[u,w] + d[w,v]$, $d[u,w] > \delta(u,w)$. If replacing $d[u,w]$ by $\delta(u,w)$ gives a better path, then the original path is not the shortest path, which contradicts the definition of $\delta(s, v)$.

Theorem. For all $u,v,x \in V$, we have $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$.

Lemma 1. Initializing $d[s] = 0$ and $d[v] = \infty$ for all $v \in V - \{s\}$ establishes $d[v] \geq \delta(s, v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps.

Proof. Suppose not. Let v be the first vertex for which $d[v] < \delta(s, v)$, and let u be the vertex that caused $d[v]$ to change: $d[v] = d[u] + w(u, v)$.

$d[v] < \delta(s, v)$ supposition

$\leq \delta(s, u) + \delta(u, v)$ triangle inequality

$\leq \delta(s, u) + w(u, v)$ shortest path ≤ specific path

$\leq d[u] + w(u, v)$, v is the first violation

in contradiction to $d[v] = d[u] + w(u, v)$.

Lemma 2. Let u be v's predecessor on a shortest path from s to v. Then, if $d[u] = \delta(s, u)$ and edge (u, v) is relaxed, we have $d[v] = \delta(s, v)$ after the relaxation.

Proof. Observe that $\delta(s, v) = \delta(s, u) + w(u, v)$. Suppose that $d[v] > \delta(s, v)$ before the relaxation (Otherwise, we're done). Then, the test $d[v] > d[u] + w(u, v)$ succeeds, because $d[v] > \delta(s, v) = \delta(s, u) + w(u, v) = d[u] + w(u, v)$, and the algorithm sets $d[v] = d[u] + w(u, v) = \delta(s, v)$.

Graph Theory

Graph theory is a branch of mathematics that deals with the study of graphs, which are mathematical structures used to model pairwise relationships between objects. A graph consists of a set of vertices (or nodes) and a set of edges connecting pairs of vertices. These vertices and edges can represent a wide variety of entities and relationships, making graph theory a versatile and powerful tool in various fields.

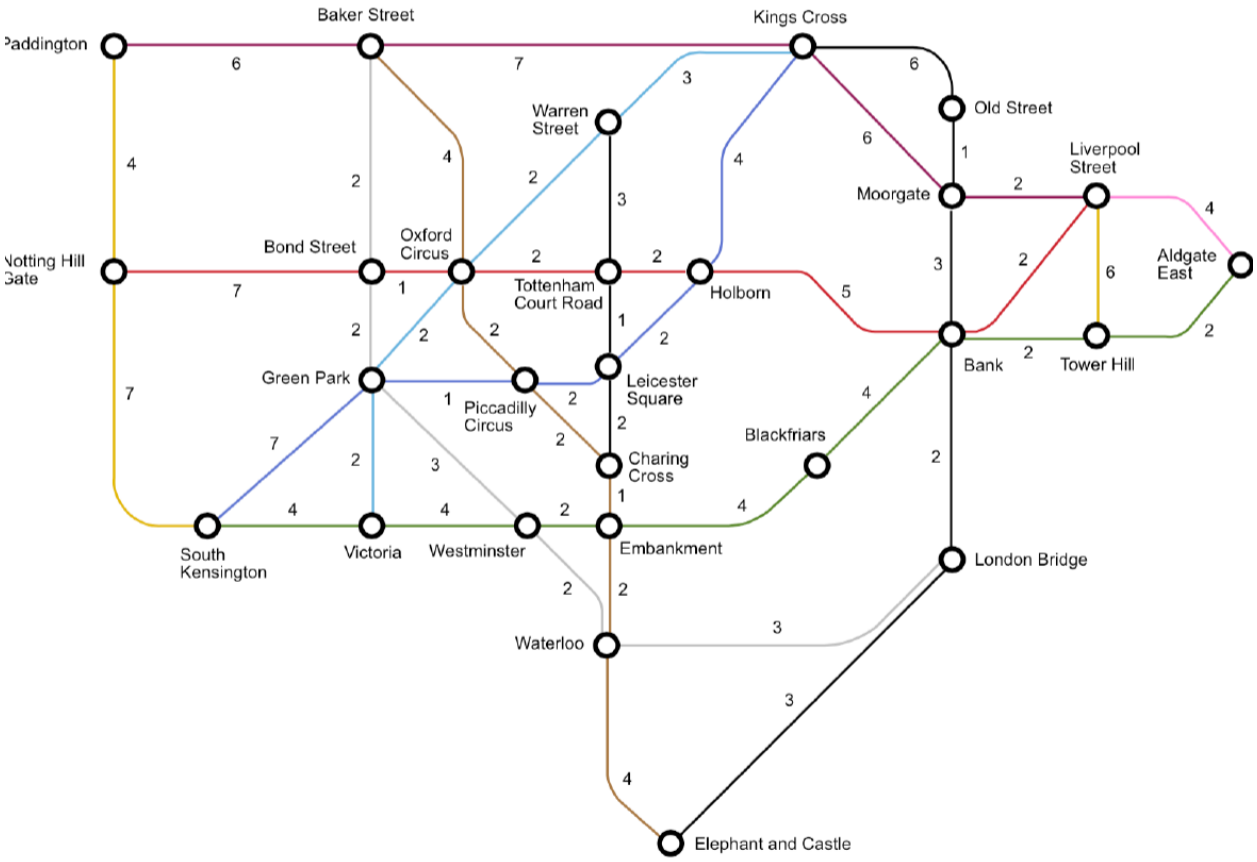
Here are some fundamental concepts in graph theory:

- Vertices (Nodes): These are the fundamental units in a graph, representing entities such as cities, people, or any other objects.
- Edges: Edges connect pairs of vertices and represent relationships between them. An edge may be directed (with an arrow indicating a one-way relationship) or undirected (with no direction, representing a mutual relationship).
- Graphs: A graph is a collection of vertices and edges. It can be either directed or undirected, depending on whether the edges have a specific direction.
- Degrees: The degree of a vertex is the number of edges connected to it. In a directed graph, vertices have both an in-degree (number of incoming edges) and an out-degree (number of outgoing edges).
- Paths and Cycles: A path in a graph is a sequence of vertices where each adjacent pair is connected by an edge. A cycle is a closed path, where the first and last vertices are the same.
- Connected Graphs: A graph is connected if there is a path between every pair of vertices. Otherwise, it is disconnected.

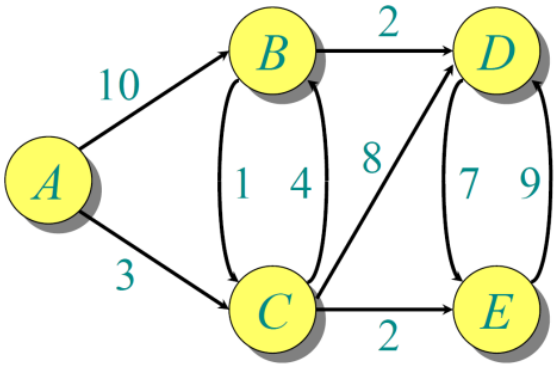
Graph theory has applications in various fields such as computer science, biology, social science, linguistics, and operations research. It is used to model and solve problems related to networks, transportation systems, social networks, circuit design, optimization, and much more. The study of graph algorithms and properties of graphs forms a significant part of graph theory, contributing to the development of efficient algorithms for problem-solving in different domains.

Question

Your task is to write a Python script that, given the rail network in Figure, computes the shortest paths within the network given any starting station



Test Network Figure:



Objective

- The overarching aim of this study is to elucidate the operational intricacies of the proposed algorithm through the meticulous computation of a minimum of four distinct journeys within the network. The algorithmic output will be rigorously documented, encompassing both the temporal duration of each journey and the optimal path to be traversed.

- This research further aspires to scrutinize the algorithm's fidelity and computational efficiency. The correctness assessment involves a meticulous examination of the algorithm's output against predetermined benchmarks for the selected journeys, ensuring its precision in solution generation. Concurrently, the efficiency evaluation seeks to quantify the computational resources expended during the algorithm's execution, thereby establishing a quantitative measure of its effectiveness.

- In addition to evaluating the algorithm within its immediate context, this study endeavors to investigate its reusability across diverse networks. The assessment will discern the algorithm's adaptability and generalizability, determining its efficacy when applied to networks with varying topologies and characteristics.

- To facilitate comprehension and utilization by a broader audience, a comprehensive documentation endeavor is undertaken. The documentation aims to elucidate the intricacies of the code to a level where external users can effortlessly grasp its functionality and seamlessly integrate it into their applications. This entails not only descriptive comments within the code but also a separate document delineating the algorithm's architecture, methodologies, and pertinent parameters.

- The culmination of this research involves the rigorous testing of the algorithm, subjected to a battery of scenarios to assess its robustness and reliability across diverse inputs and network configurations. This iterative testing process aims to validate the algorithm's performance under varying conditions, ensuring its practical viability and effectiveness across a spectrum of real-world applications.

Code

```
def dijkstra(graph, start):
    '''
    Given the graph containing distances between different nodes and the starting node,
```

```
it returns a shortest time-taking route to travel all the nodes on the graph.

Keyword arguments:
graph (dictionary) -- A dictionary representing the graph where keys are nodes and
                      values are dictionaries of neighboring nodes with their corresponding distances.
start (string) -- The name of the starting node.

Returns:
result (dictionary): A dictionary containing the shortest time-taking route information.
    Each node is a key with a sub-dictionary containing 'distance' (total distance to reach the node)
    and 'path' (the shortest path to reach the node from the starting node).
'''
# Initialize distances with infinity for all nodes except the start node
distances = {node: float('infinity') for node in graph}
distances[start] = 0

# Use a list to keep track of nodes with the smallest distances
priority_queue = [(0, start)]

# Track the predecessors for each node
predecessors = {node: None for node in graph}

while priority_queue:
    # Find the node with the smallest distance in the priority queue
    current_distance, current_node = min(priority_queue)

    # Remove the current node from the priority queue
    priority_queue.remove((current_distance, current_node))

    # Check if the current distance is already greater than the known distance
    if current_distance > distances[current_node]:
        continue

    # Iterate through neighbors of the current node
    for neighbor, weight in graph[current_node].items():
        distance = current_distance + weight

        # Update the distance and predecessor if a shorter path is found
        if distance < distances[neighbor]:
            distances[neighbor] = distance
            predecessors[neighbor] = current_node
            priority_queue.append((distance, neighbor))

# Build and return the result including distances and paths
```

```
    result = {node: {'distance': distances[node], 'path': get_path(start, node, predecessors)} for node in graph}
    return result

def get_path(start, end, predecessors):
    path = []
    current_node = end
    while current_node is not None:
        path.insert(0, current_node)
        current_node = predecessors[current_node]
    return path


# Testing of algorithm on a smaller problem as a coding sanity check
test_graph = {
    'A': {'B': 10, 'C': 3},
    'B': {'C': 1, 'D': 2},
    'C': {'B': 4, 'D': 8, 'E': 2},
    'D': {'E': 7},
    'E': {'D': 9}
}

starting_node = 'A'
destination_node = 'D'
result = dijkstra(test_graph, starting_node)

distance = result[destination_node]['distance']
path = result[destination_node]['path']
print(f"Starting station : {starting_node}")
print(f"Destination station : {destination_node}")
print(f"Time : {distance} minutes")
print(f"Route : {path}")


# Import the graph of London underground:
London_graph = {
    'Paddington': {'Notting Hill Gate': 4, 'Baker Street': 6},
    'Notting Hill Gate': {'Paddington': 4, 'Bond Street': 7, 'South Kensington': 7},
    'South Kensington': {'Notting Hill Gate': 7, 'Green Park': 7, 'Victoria': 4},
    'Baker Street': {'Paddington': 6, 'Bond Street': 2, 'Oxford Circus': 4, 'Kings Cross': 7},
    'Bond Street': {'Notting Hill Gate': 7, 'Baker Street': 2, 'Green Park': 2, 'Oxford Circus': 1},
    'Green Park': {'South Kensington': 7, 'Bond Street': 2, 'Victoria': 2, 'Oxford Circus': 2, 'Piccadilly Circus': 1, 'Westminster': 3},
    'Victoria': {'South Kensington': 4, 'Green Park': 2, 'Westminster': 4},
    'Oxford Circus': {'Baker Street': 4, 'Bond Street': 1, 'Green Park': 2, 'Piccadilly Circus': 2, 'Warren Street': 2, 'Tottenham Court
Road': 2},
```

```
    'Piccadilly Circus': {'Green Park': 1, 'Oxford Circus': 2, 'Leicester Square': 2, 'Charing Cross': 2},
    'Westminster': {'Green Park': 3, 'Victoria': 4, 'Embankment': 2, 'Waterloo': 2},
    'Warren Street': {'Oxford Circus': 2, 'Tottenham Court Road': 3, 'Kings Cross': 3},
    'Tottenham Court Road': {'Oxford Circus': 2, 'Warren Street': 3, 'Leicester Square': 1, 'Holborn': 2},
    'Leicester Square': {'Piccadilly Circus': 2, 'Tottenham Court Road': 1, 'Charing Cross': 2, 'Holborn': 2},
    'Charing Cross': {'Piccadilly Circus': 2, 'Leicester Square': 2, 'Embankment': 1},
    'Embankment': {'Westminster': 2, 'Charing Cross': 1, 'Waterloo': 2, 'Blackfriars': 4},
    'Waterloo': {'Westminster': 2, 'Embankment': 2, 'Elephant and Castle': 4},
    'Holborn': {'Tottenham Court Road': 2, 'Leicester Square': 2, 'Kings Cross': 4, 'Bank': 5},
    'Elephant and Castle': {'Waterloo': 4, 'London Bridge': 3},
    'Kings Cross': {'Baker Street': 7, 'Warren Street': 3, 'Holborn': 4, 'Old Street': 6, 'Moorgate': 6},
    'Blackfriars': {'Embankment': 4, 'Bank': 4},
    'Old Street': {'Kings Cross': 6, 'Moorgate': 1},
    'Moorgate': {'Kings Cross': 6, 'Old Street': 1, 'Bank': 3, 'Liverpool Street': 2},
    'Bank': {'Holborn': 5, 'Blackfriars': 4, 'Moorgate': 3, 'London Bridge': 2, 'Liverpool Street': 2, 'Tower Hill': 2},
    'London Bridge': {'Waterloo': 3, 'Elephant and Castle': 3, 'Bank': 2},
    'Liverpool Street': {'Moorgate': 2, 'Bank': 2, 'Tower Hill': 6, 'Aldgate East': 4},
    'Tower Hill': {'Bank': 2, 'Liverpool Street': 6, 'Aldgate East': 2},
    'Aldgate East': {'Liverpool Street': 4, 'Tower Hill': 2}
}

starting_station = 'Westminster'
destination_station = 'Waterloo'
result = dijkstra(London_graph, starting_station)

distance = result[destination_station]['distance']
path = result[destination_station]['path']
print(f"Starting station : {starting_station}")
print(f"Destination station : {destination_station}")
print(f"Time : {distance} minutes")
print(f"Route : {path}")
```

Testing and Output

Route 1

Paddington -> Charing Cross

```
In [2]: runfile('D:/8212/Python/Dijkstra 2.py', wdir='D:/8212/Python')
Starting station : Paddington
Destination station : Charing Cross
Time : 13 minutes
Route : ['Paddington', 'Baker Street', 'Bond Street', 'Oxford Circus', 'Piccadilly
Circus', 'Charing Cross']
```

Route 2

Victoria -> Liverpool Street

```
In [3]: runfile('D:/8212/Python/Dijkstra 2.py', wdir='D:/8212/Python')
Starting station : Victoria
Destination station : Liverpool Street
Time : 14 minutes
Route : ['Victoria', 'Green Park', 'Piccadilly Circus', 'Leicester Square',
'Holborn', 'Bank', 'Liverpool Street']
```

Route 3

Embankment -> Kings Cross

```
In [4]: runfile('D:/8212/Python/Dijkstra 2.py', wdir='D:/8212/Python')
Starting station : Embankment
Destination station : Kings Cross
Time : 9 minutes
Route : ['Embankment', 'Charing Cross', 'Leicester Square', 'Holborn', 'Kings
Cross']
```

Route 4

London Bridge -> Oxford Circus

```
In [5]: runfile('D:/8212/Python/Dijkstra 2.py', wdir='D:/8212/Python')
Starting station : London Bridge
Destination station : Oxford Circus
Time : 10 minutes
Route : ['London Bridge', 'Waterloo', 'Westminster', 'Green Park', 'Oxford Circus']
```

Meeting Record

Meeting 1

Date: 27th November

Duration: 60 minutes

Location: Hartley Library

Objective: Introduction and Information Sharing on Dijkstra's Algorithm

Summary:

The inaugural meeting took place on the 27th of November and spanned a duration of 60 minutes. The venue for the meeting was the Hartley Library. The primary objective of this initial encounter was to facilitate introductions among the Group Members and initiate the sharing of knowledge pertaining to the Dijkstra Algorithm.

During the meeting, participants engaged in introductory discussions, providing an opportunity for each individual to present themselves. This session aimed to establish a foundational understanding of the diverse backgrounds and perspectives of the participants, fostering a collaborative environment for subsequent interactions.

The central focus of the meeting was the exchange of information related to the Dijkstra Algorithm. Participants shared insights, experiences, and any existing knowledge they possessed about the algorithm. This collective sharing of information set the stage for subsequent in-depth discussions and analyses in the forthcoming meetings.

The meeting successfully achieved its introductory objectives, creating a platform for collaborative exploration and discussion on the intricacies of Dijkstra's Algorithm. The shared insights will serve as a valuable foundation for the continued exploration and understanding of this algorithm in subsequent meetings.

Meeting 2

Date: 4th December 2023

Duration: 90 minutes

Location: Hartley Library

Objective: Information Gathering, Planning, and Winter Vacation Coordination

Summary:
The second meeting transpired on the 4th of December 2023, lasting for a duration of 90 minutes and taking place at the Hartley Library. The overarching goals of this meeting encompassed the acquisition of additional information from diverse sources, collaborative planning, and a mutual understanding of each participant's plans for the upcoming winter vacations.

During the session, participants actively engaged in the collection of information pertinent to the subject matter, leveraging resources such as Google, Geekforgeeks, Stackoverflow, Github and content accessible on the Blackboard platform. The incorporation of these varied sources aimed to enrich the collective knowledge base, ensuring a comprehensive understanding of the intricacies surrounding the Dijkstra Algorithm.

A pivotal component of this meeting involved the collaborative development of a strategic planner. This planner served as a roadmap delineating the proposed trajectory for future meetings and collective research endeavors. The structured planning phase facilitated a synchronized approach, enhancing the efficiency and effectiveness of the group's exploration of Dijkstra's Algorithm.

Additionally, the meeting provided an opportune moment for participants to openly discuss and comprehend each other's plans for the forthcoming winter vacations. This exchange of information fostered a supportive environment within the group, allowing for considerations of individual timelines and potential adjustments to the collaborative schedule.

Meeting 3

Date: 10th December 2023

Duration: 90 minutes

Location: Hartley Library

Objective: Collaborative Engagement and Code Testing

Summary:
The third meeting convened on the 10th of December 2023, lasting for a duration of 90 minutes and taking place at the Hartley Library. The primary focus of this meeting was to foster an

inclusive and collaborative environment, ensuring equal participation from all four group members in various activities related to the exploration of Dijkstra's Algorithm.

Throughout the meeting, an ethos of shared engagement prevailed, with each group member actively participating in discussions, planning, and implementation activities. The seamless collaboration was underscored by a collective awareness of individual contributions, allowing for a cohesive understanding of the progress made by each team member.

The synergy within the group was particularly evident in the dynamic exchange of ideas. All participants actively brought forth concepts, suggestions, and insights relevant to the exploration of Dijkstra's Algorithm. This inclusive approach not only enriched the depth of the discussions but also contributed to a comprehensive exploration of diverse perspectives within the group.

An integral aspect of the meeting involved the practical implementation of the algorithm. The group collectively worked on creating a sample test network, providing a practical framework for testing and validating the codes developed. This hands-on approach not only reinforced the theoretical understanding of the algorithm but also served as a crucial step in ensuring the robustness and accuracy of the group's code implementations.

Meeting 4

Date: 18th December 2023

Duration: 3 hours

Location: Hartley Library

Objective: Completion of Coding and Testing Phase

Summary:

The fourth meeting transpired on the 18th of December 2023, extending over a comprehensive 3-hour duration and taking place at the Hartley Library. The principal aim of this meeting was to conclude the coding and testing phase, particularly considering the impending vacation plans of most group members.

Given the shared understanding of the upcoming vacation commitments, the group collectively decided to channel their efforts into completing the coding and testing aspects of the project during this meeting. The strategic allocation of the available time and resources allowed the group to maximize productivity and capitalize on the completion of outstanding coursework.

The meeting was notably productive, as the group successfully finalized the code implementation and conducted thorough testing. The completion of other concurrent coursework obligations further facilitated an environment conducive to concentrated and effective collaboration.

Towards the conclusion of the meeting, the group accomplished the creation of a test path for a sanity check, ensuring the fundamental correctness of the algorithm implementation. Subsequently, the group focused on coding for the actual problem statement. Four specific test paths were meticulously chosen for evaluation: Paddington to Charing Cross, Victoria to Liverpool Street, Embankment to Kings Cross, and London Bridge to Oxford Circus.

Upon obtaining the algorithm's output, the group undertook a manual verification process, aligning the results with anticipated outcomes. Remarkably, the output was verified to be accurate, affirming the precision of the implemented code. Additionally, random tests were conducted on alternative paths to further validate the algorithm's robustness and reliability.

Meeting 5

Date: 4th January 2024

Duration: 3 hours

Location: Hartley Library

Objective: Progress Review, Code Re-evaluation, and Report Planning

Summary:

The fifth meeting convened on the 4th of January 2024, spanning a duration of 3 hours and held at the Hartley Library. This marked the first collaborative session post the new year, focusing on an extensive review of the work accomplished thus far, including a thorough examination of the code and the initiation of the report and documentation phase.

The meeting commenced with a comprehensive overview of the progress made by the group since the last meeting. Participants engaged in a reflective discussion, revisiting the implemented code, and collectively assessing the outcomes of prior testing and verification processes. This retrospective analysis aimed to ensure a holistic understanding of the existing project status.

To reaffirm the stability and accuracy of the implemented code, the group conducted a re-run of their algorithms, meticulously comparing the new outputs with previously verified results. The

consistent outcomes reinforced the group's confidence in the reliability of their Dijkstra Algorithm implementation.

Subsequently, the focus shifted towards the planning and initiation of the report and documentation phase. The group collectively decided on the key elements to be included in the report, outlining the structure and content parameters. To facilitate seamless collaboration and editing, the report was instantiated using Google Docs, ensuring accessibility for all members from diverse locations.

The group concluded the meeting with a preliminary discussion on the framework of the report, delineating the sections, key findings, and the overall narrative structure. The collaborative use of Google Docs facilitated ongoing contributions and edits, fostering a dynamic and inclusive approach to report development.

Meeting 6

Date: 6th January 2024

Duration: 2 hours

Location: Hartley Library

Objective: Final Review of Report and Code Optimization

Summary:

The sixth and final meeting convened on the 6th of January 2024, encompassing a 2-hour duration and held at the Hartley Library. The principal focus of this conclusive session was to conduct a comprehensive review of the project's report and code, ensuring precision, clarity, and optimization in preparation for submission.

Commencing the meeting, the group engaged in a meticulous examination of the report, critically assessing each section to guarantee coherence and completeness. The review aimed to ascertain that the report elucidated the project's objectives, methodologies, and findings in a manner accessible to individuals unfamiliar with the codebase. Participants made deliberate efforts to enhance clarity, making minor amendments to language and structure to optimize overall comprehension.

Simultaneously, a parallel review of the implemented code was undertaken. Every detail was cross-checked to verify its accuracy, adherence to coding conventions, and optimal efficiency.

The group ensured that the codebase was not only functional but also well-organized, following best practices in software development.

Particular attention was directed towards making the report and code comprehensible for individuals who had not directly participated in the project. This involved providing adequate explanations, comments, and documentation within the code and report to facilitate ease of understanding for a diverse audience.

As a culmination of the meeting, the group collaboratively made minor amendments to both the report and code, optimizing language, clarity, and structure. The final iterations were crafted with meticulous attention to detail, reflecting the group's commitment to delivering a polished and professional submission.

Collectively, the group determined that the report and code were refined and ready for submission, marking the successful conclusion of their collaborative exploration of Dijkstra's Algorithm. The meeting encapsulated the culmination of efforts, emphasizing precision, clarity, and optimal functionality in the project's final deliverables.

Conclusion

- The implemented code has been rigorously assessed and validated, affirming its efficiency and accuracy within the specified network. Its design supports ease of reuse, providing a versatile tool for subsequent applications. A preliminary sanity check was conducted on a sample network, validating the program's functionality and correctness.

- A notable aspect of the code's implementation is its meticulous documentation. The documentation is structured comprehensively, ensuring that individuals unfamiliar with the intricacies of the code can readily comprehend and utilize it. This approach aligns with best practices in software development, fostering accessibility and promoting the potential for future collaborative efforts.

- Even for individuals lacking proficiency or possessing only rudimentary knowledge of Python, the code remains accessible. A simple adaptation involves renaming the starting and destination stations, allowing users to obtain desired output without necessitating an in-depth understanding of the underlying codebase.

- The confidence in the code's applicability extends to its potential use by Transport for London (TFL) for journey optimization within their network. Multiple tests have been conducted on both the specified network and a deliberately constructed sample network. These tests have collectively affirmed the reliability, accuracy, and practical viability of the implemented code in diverse scenarios.

References

- https://github.com/dmahugh/dijkstra-algorithm
- https://stackoverflow.com/questions/22897209/dijkstras-algorithm-in-python
- https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
- https://blackboard.soton.ac.uk/ultra/courses/_219524_1/cl/outline