# Final Project Report (Summer 2023 Semester)

# Smarter Travel Google Map

## Optimizing Best Visiting Route with Multiple Destinations for Google Map

**Ahmad El Idrissi Amiri, Nishtha Sawhney, Yiming Zhu**
**Under supervision of Dr. Justin Kennedy**

## Introduction

In an era characterized by rapid globalization and technological advancement, the ability to efficiently navigate through diverse locations has become an essential aspect of modern life. Whether it's for business or leisure, individuals frequently find themselves faced with the challenge of visiting multiple destinations in the most time-effective and cost-efficient manner. While navigation platforms like Google Maps have revolutionized the way we find our way around, they still lack a crucial feature: the ability to provide users with the optimal route that encompasses multiple destinations in a single journey.

The **"Smarter Travel: Optimizing Best Visiting Route with Multiple Destinations for Google Maps"** project seeks to address this pressing need by harnessing the power of advanced algorithms to compute the shortest path that allows an individual to visit a set of multiple destinations, starting from a designated origin point. By incorporating the **Shortest Path** and **Held-Karp algorithms**, this project aims to revolutionize the way individuals plan and execute multi-destination trips, saving time, reducing travel costs, and enhancing the overall travel experience.

Traditionally, planning trips involving several destinations has been a labor-intensive process, requiring individuals to manually calculate various route options, consider traffic conditions, and account for distances between locations. This project aims to automate and optimize this process, enabling users to effortlessly input their desired destinations and obtain a seamlessly integrated route that ensures minimal time and distance traveled.

The Held-Karp algorithm, a well-established technique in the field of computer science and operations research, forms the backbone of this project. By leveraging its combinatorial optimization capabilities, the algorithm allows us to efficiently calculate the shortest possible route while visiting all specified destinations. Paired with the existing infrastructure of Google Maps, this project introduces a groundbreaking feature that caters to a wide array of users, including tourists, delivery drivers, sales representatives, and professionals on the move.

In the subsequent sections of this project, we will delve into the underlying principles of the algorithms being used, outline the technical implementation process, and provide illustrative examples to showcase the effectiveness of our approach. Through this project, we hope to bridge the gap in multi-destination route planning on Google Maps, contributing to a smarter and more connected world of travel.

## Motivation

**Yiming:**

My deep passion for geography and my unwavering enthusiasm for travel have led me to a realization that modern navigation tools, despite their convenience, lack a crucial feature: the ability to efficiently optimize routes for visiting multiple destinations. As an avid user of Google Maps in my daily life, I've often encountered the frustration of planning complex itineraries manually. This frustration, coupled with my exposure to the concepts covered in our CS5800 Algorithms class, has ignited a strong motivation within me to bridge this gap. I am excited to harness the power of algorithms like minimum spanning trees and Prim's algorithm to enhance Google Maps' capabilities, making multi-destination route planning a seamless and time-efficient experience for fellow travel enthusiasts like me. Through this project, I aspire to bridge the gap between technology and wanderlust, providing a solution that empowers travelers to embark on more meaningful, efficient, and memorable adventures.

**Ahmad:**

I am motivated to embark on this research project because I firmly believe in the power of technology and data-driven solutions to address real-world challenges. In an era marked by increasing connectivity and globalization, optimizing travel routes and minimizing distances between desired destinations has become an urgent need. Witnessing the rising gas prices and their impact on travelers, it becomes evident that we must find innovative ways to help individuals save time and money while reducing their carbon footprint. I am driven by the potential to create a user-friendly model that empowers tourists and travelers with efficient, eco-friendly routes, allowing them to enjoy seamless journeys without compromising the environment. By combining my passion for technology, data analytics, and environmental consciousness, this research holds the promise of making a tangible difference in people's lives while contributing positively to the sustainability of our planet.

**Nishtha:**

I absolutely love road trips and rely on Google Maps for navigation. However, I wish it had a feature to find the shortest routes between different cities or places. This would save time, gas, and reduce my carbon footprint. When my friend Yiming proposed this idea, I got really excited about the project's potential. Implementing this concept in Google Maps would bring immense benefits, making road trips more economical, less stressful, and encouraging people to explore new destinations. It could revolutionize travel planning, fostering a sense of adventure and discovery. As a tech-savvy traveler, I believe this feature would be incredibly useful, enhancing the travel experience for millions. I eagerly await the day when Google Maps incorporates this feature, unlocking a world of possibilities for travelers everywhere.

## Video Link

https://www.youtube.com/watch?v=4KauMjdlQ3Q

## Analysis

### What did we do to approach the problem?

To address the challenge of optimizing the best visiting route with multiple destinations on Google Maps, our project employs a multi-faceted approach that combines graph theory and optimization techniques. The core idea revolves around transforming the map into a graph, where locations are nodes and the distances between them are edges. This graph representation allows us to apply various algorithms to find the most efficient route.

The first step involves collecting data from Google & Google Map to get a list of distances between each pair of consecutive destinations.

The next challenge is to determine the optimal order in which these destinations should be visited. This is where the Travel Salesman Problem and  Held-Karp algorithms come into play. The algorithm aims to connect all nodes (destinations) with the least possible total edge weight (distance) from a certain starting node. While there are various algorithms to achieve this, we've chosen to focus on the Held-Karp's algorithm due to its efficiency and suitability for this scenario.

In our pursuit to solve the Smarter Travel project efficiently, we embarked on a journey of algorithm exploration. Initially, we explored the Minimum Spanning Tree (MST) approach, considering Prim's algorithm, only to realize that the Smarter Travel project goes beyond the realm of an MST problem due to its requirement of visiting all locations exactly once. Subsequently, we delved into Dijkstra's algorithm, but its greedy nature and inability to account for visiting all destinations proved unsuitable for the comprehensive Smarter Travel project. Fueled by the determination to conquer this challenge, we finally determined that this is a Traveling Salesman Problem (TSP), and we encountered the Held-Karp algorithm—a dynamic programming marvel. Recognizing its potential to address the TSP intricacies, we embraced it as the optimal solution. This algorithm, tailored for the TSP, intelligently considers all possible combinations of city orders, ensuring the shortest path while visiting every location. Through this iterative process, we honed in on Held-Karp as the ideal candidate to tackle the complexities of the TSP and pave the way for efficient route optimization.

## Traveling Salesman Problem and Held-Karp Algorithms

The Traveling Salesman Problem (TSP) is a classic optimization problem in the field of operations research and computer science. The problem can be stated as follows: Given a list of cities and the distances between each pair of cities, find the shortest possible route that visits each city exactly once and returns to the starting city.

The TSP has many real-world applications beyond just a traveling salesman, such as route optimization for delivery trucks, circuit board manufacturing, and tour planning for sightseeing or exploration. It's classified as a combinatorial optimization problem and is known to be NP-hard, meaning that as the number of cities increases, finding the optimal solution becomes increasingly computationally expensive.

The Held-Karp algorithm is a dynamic programming approach used to solve the Traveling Salesman Problem with a time complexity of O(n^2 * 2^n), where n is the number of cities (in our specific example, n is the number of locations we want to visit in Boston). This algorithm is an improvement over the brute-force approach, which would require checking all possible permutations of city orders (n! permutations).

The Held-Karp algorithm breaks down the TSP into subproblems, where each subproblem is defined by a set of cities to visit and the ending city of the current route. By solving these subproblems and using memoization to store their solutions, the algorithm efficiently computes the optimal route for the TSP.

The key insight of the algorithm is that the shortest path from the starting city to any other city, passing through a subset of intermediate cities, can be computed recursively based on previously solved subproblems. The algorithm builds a memoization table to store the optimal distances for each combination of visited cities, which significantly reduces redundant calculations and speeds up the overall process.

While the Held-Karp algorithm is effective for small to moderate-sized instances of the TSP, it becomes impractical for larger instances due to its exponential time complexity. For very large instances, approximation algorithms and heuristics are often used to find near-optimal solutions in a reasonable amount of time.

In summary, the Traveling Salesman Problem challenges us to find the shortest route that visits a set of cities exactly once and returns to the starting city. The Held-Karp algorithm provides an efficient way to solve small to moderate-sized instances of the TSP by utilizing dynamic programming and memoization to avoid redundant calculations.

## Why Held-Karp Algorithm?

The choice of algorithm to solve the Traveling Salesman Problem (TSP) depends on various factors, including the problem instance size, the desired level of optimality, and the available computational resources. The Held-Karp algorithm is one of several approaches available for solving the TSP, and its selection can be influenced by its advantages and limitations:

<u>Advantages of the Held-Karp Algorithm:</u>

- ❖ Optimality: The Held-Karp algorithm guarantees to find the optimal solution for the TSP. It systematically considers all possible combinations of city orders and calculates the shortest route among them. This is especially valuable when optimality is a critical requirement.

- ❖ Exact Solution: The algorithm provides an exact solution, meaning that it will find the best possible route that minimizes the total distance traveled. This can be crucial in scenarios where even small deviations from the optimal solution are not acceptable.

- ❖ No Heuristics: Unlike some approximation algorithms and heuristics, the Held-Karp algorithm doesn't rely on making assumptions or approximations that might lead to suboptimal solutions. It directly computes the optimal solution through its dynamic programming approach.

<u>Limitations of the Held-Karp Algorithm:</u>

- ❖ Exponential Time Complexity: The Held-Karp algorithm has an exponential time complexity of O(n^2 * 2^n), where n is the number of cities. This means that it becomes computationally infeasible for larger instances of the TSP. As the number of cities increases, the computation time grows exponentially.

- ❖ Memory Usage: The algorithm's memory usage also increases exponentially with the number of cities. Storing solutions for all possible combinations of visited cities can require significant memory resources, making it impractical for very large instances.

- ❖ Scaling Issues: Due to its exponential nature, the algorithm's performance deteriorates rapidly as the number of cities increases. This can limit its usability for real-world instances with a large number of destinations.

Choosing Held-Karp Algorithm or Alternatives:

The decision to use the Held-Karp algorithm or alternative approaches depends on the trade-offs between solution quality and computation time/resources:

- For Small Instances: When dealing with relatively small instances of the TSP (e.g., a dozen cities), the Held-Karp algorithm can provide the optimal solution, making it suitable for tasks that demand precision.

- For Large Instances: For larger instances with a significant number of cities, approximation algorithms like nearest neighbor, genetic algorithms, and simulated annealing may be more practical. While they don't guarantee optimality, they can find good solutions in a reasonable amount of time.

- Balancing Precision and Resources: If optimality is paramount and computational resources are available, the Held-Karp algorithm can be used. However, for larger instances, it might be necessary to explore hybrid approaches that use Held-Karp for smaller subproblems and heuristics or approximations for larger portions of the problem.

In essence, choosing the Held-Karp algorithm depends on the problem's size and the importance of obtaining the absolute optimal solution.

In the subsequent sections of our project, we will delve into the technical implementation of Held-Karp algorithm, showcasing its step-by-step execution and presenting real-world examples to highlight its effectiveness in solving the multi-destination route optimization problem. Through this approach, we aim to empower users with a tool that not only saves time and resources but also enhances their ability to make the most of their travel plans.

## How we collect data

In the data collection phase of our research project, Google Maps played a pivotal role in obtaining the distances between different places seamlessly. By using the directions feature on Google Maps, we could effortlessly access the precise distances in miles between various desired destinations. This automated process not only saved valuable time but also ensured the accuracy of the distance measurements. The collected data was then meticulously organized into a comprehensive table, allowing for easy comparison and analysis. Google Maps' reliable and up-to-date information significantly contributed to the robustness of the research. Thanks to this efficient data collection approach, we could focus on the core aspects of the research, refining the model and delivering optimal results for travelers seeking time-saving, eco-friendly routes to their preferred destinations.

## Project Scope and Assumptions

In any complex project, defining the scope and making well-considered assumptions is crucial to managing time and resources effectively. For our project, we acknowledge the need for streamlined focus due to time constraints. As such, the following assumptions will serve as the foundation upon which we build our solution:

1. Single Means of Transportation (Car):
Given the time limitations, we will exclusively consider travel by car as the means of transportation. While there are various modes of transportation available, restricting our analysis to car travel simplifies route planning and algorithm implementation.

2. Single Visitor per Trip:
To maintain simplicity and expedite development, we assume that each trip is undertaken by a single visitor. This avoids the complexities associated with group dynamics, preferences, and diverse schedules.

3. Exploration Limited to Boston:
While the world is full of exciting destinations, focusing solely on exploring places within Boston allows us to work within a well-defined geographical area. This concentration helps in optimizing the route while disregarding potential long-distance travel planning.

4. Ignoring Visiting Time:
In this initial iteration, we will not consider the time spent at each location. Our primary goal is to optimize the route itself rather than the duration of stay at individual destinations. This enables us to focus on the core challenges of multi-destination route planning.

5. Excluding Opening Hours:
Although the operating hours of locations play a significant role in travel planning, we will omit this factor in our project. While this assumption simplifies our implementation, it's important to recognize that real-world travel planning would require consideration of opening and closing times.

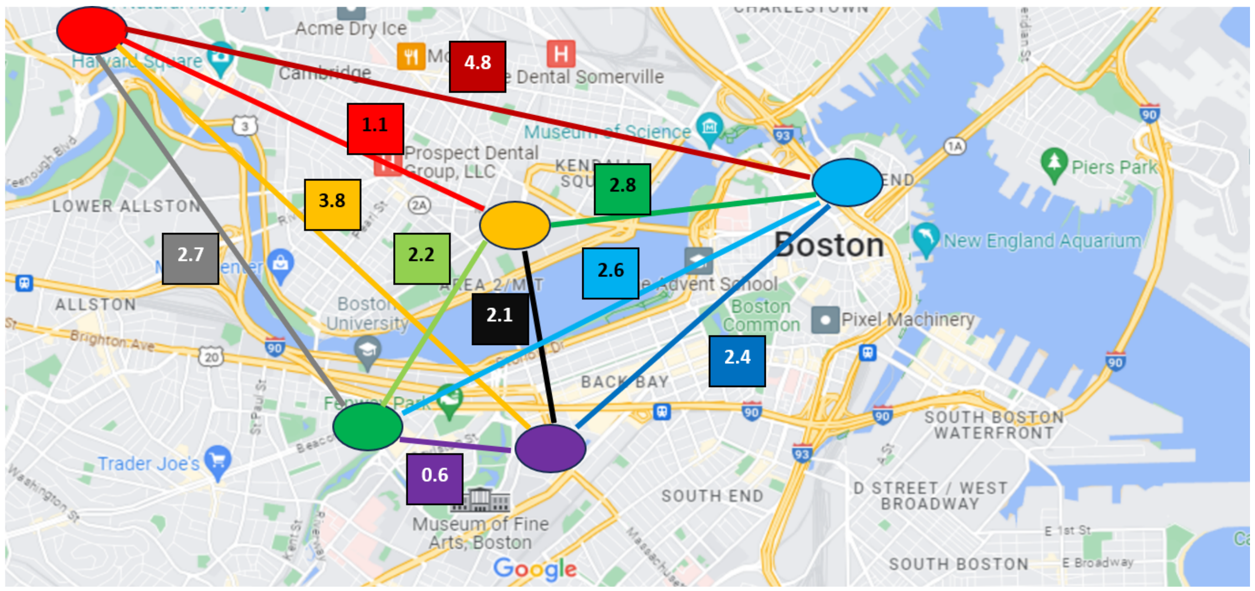
6. Shortest Distance Implies Shortest Time:
In our constrained scenario, we will operate under the assumption that the shortest distance between two locations translates directly to the shortest travel time. While this might not always hold true due to factors like traffic, road conditions, and speed limits, it provides a reasonable starting point for our simplified approach.

By carefully outlining these assumptions, we create a clear framework for our project's objectives and limitations. This focused approach will allow us to efficiently develop and present a functional solution within the given time frame. However, it's important to acknowledge that these assumptions simplify the real-world complexity of multi-destination travel planning, and further iterations could incorporate more factors for a comprehensive travel optimization tool. We will talk more in the “What’s Next” section.

**Three Different Test Cases**

To thoroughly evaluate our algorithm, we designed three distinctive case studies, each centered around an individual seeking to explore a selection of renowned Boston landmarks. Within each scenario, the person aims to visit between three to five different places from the list of famous Boston destinations, including Harvard, MIT, Boston Common, Chinatown, Isabella Museum, Museum of Fine Arts, Seaport, Prudential, and, of course, Northeastern University.

**Case 1**: Nishtha's friend, Sarah, is visiting from Washington DC and wants to see Harvard, MIT, MFA, and have dinner in Chinatown. We're figuring out the best order for her trip, starting from Nishtha's house near Brigham Circle. Using this information, like the starting point, Sarah's places to visit, and the distances between each spot, we're running our algorithm to find the optimal order for her to see all those places.



| | |
|---|---|
| 🔴 | **Harvard** |
| 🟢 | **Brigham Circle** |
| 🟡 | **MIT** |
| 🟣 | **MFA** |
| 🔵 | **Chinatown** |

**Distance Data :**

Brigham Circle to Harvard: 2.7 miles ; Brigham Circle to MIT: 2.2 miles

Brigham Circle to MFA: 0.6 miles ; Brigham Circle to Chinatown: 2.6 miles

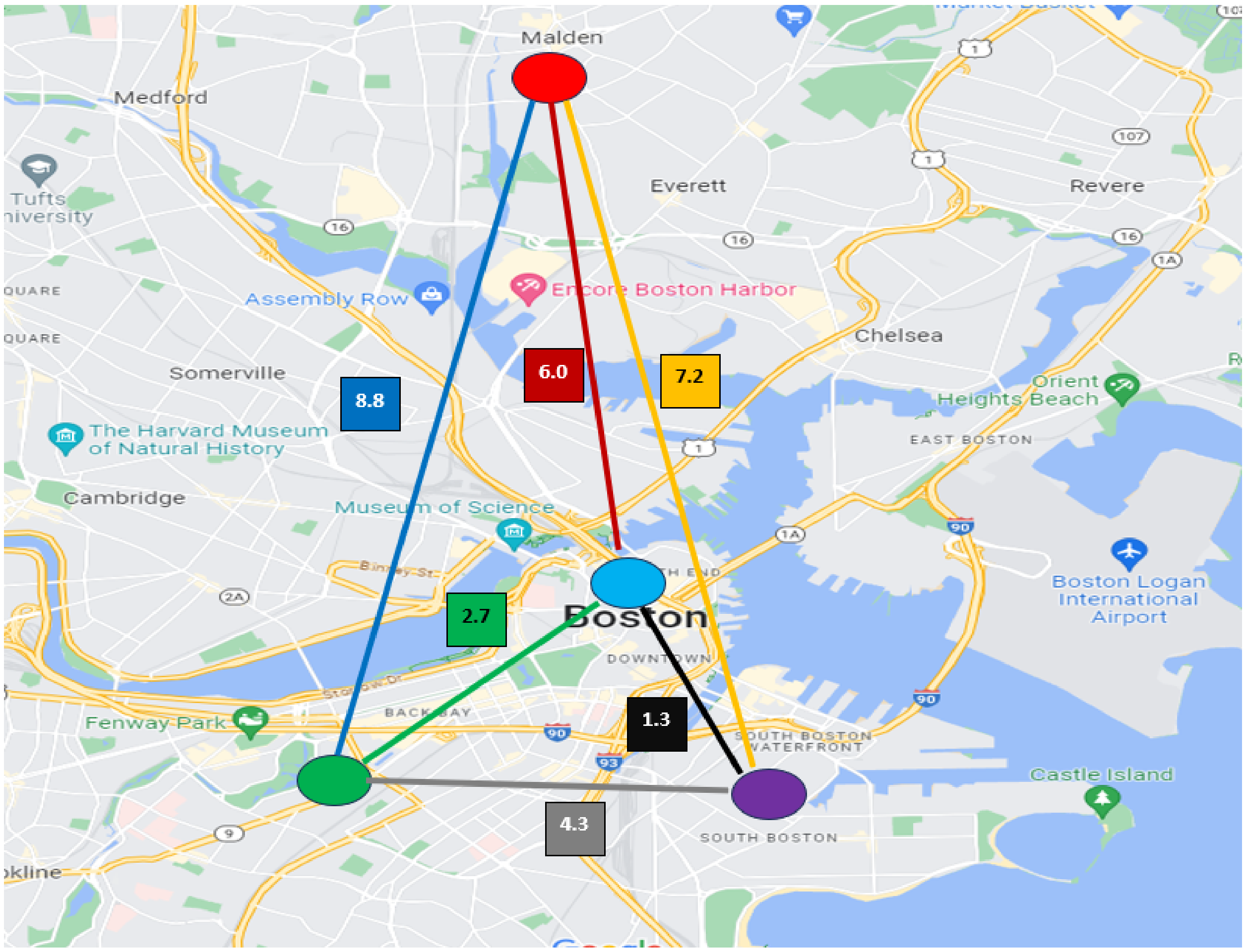
Harvard to MIT: 1.1 miles ; Harvard to MFA: 3.8 miles

Harvard to Chinatown: 4.8 miles ; MIT to MFA: 2.1 miles

MIT to Chinatown: 2.8 miles ; MFA Chinatown : 2.4 miles

After manually looking at all possible routes and distances, we found the following route had the shortest minimal distance of 9.6 miles: Brigham -> MFA -> Chinatown -> MIT -> Harvard -> Brigham.

**Case 2**: Yiming and her friend, Amber, are excited to explore Boston Common, ICA, and Isabella Stewart Museum. To kick start their adventure, they'll begin from Malden Center subway station. Armed with essential details, including the starting point, places to visit, and distances between each spot, we're employing our algorithm to determine the most efficient order for them to experience all these fantastic locations.



| | |
|---|---|
| 🔴 | **Malden Center** |
| 🟢 | **Isabella Museum** |
| 🟣 | **Seaport** |
| 🔵 | **Boston Common** |

**Distance Data :**

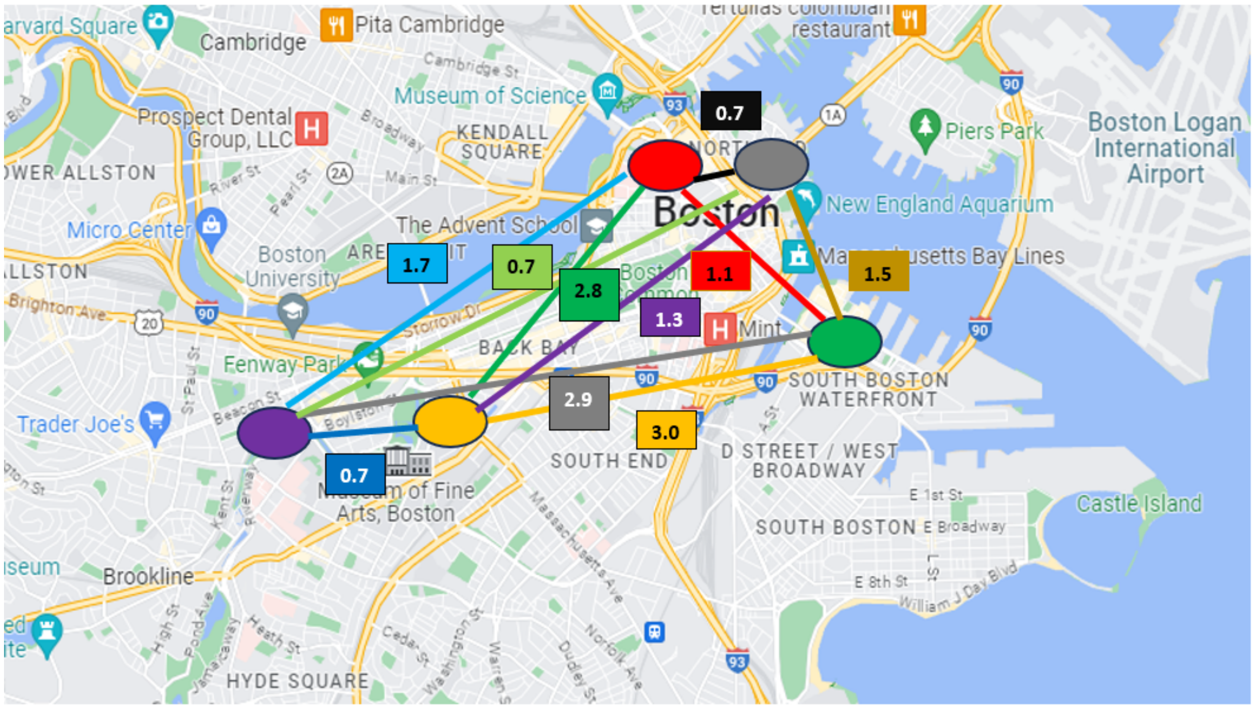
Malden center to Boston common : 6.0 miles ; Malden center to seaport: 7.2 miles

Malden center to Isabella museum : 8.8 miles ; Boston common to seaport : 1.3 miles

Boston common to Isabella museum : 2.7 miles ; Seaport to Isabella : 4.3 miles

After manually looking at all possible routes and distances, we found the following route had the shortest minimal distance of 20.0 miles: Malden Center -> ICA -> Boston Commons -> Isabella Museum -> Malden Center.

**Case 3**: Ahmad's mom has come all the way from Morocco to visit Boston for the first time. She has a lot of places she wants to visit, but we want to save time and cover all the areas efficiently. Her list includes exploring NEU, visiting Seaport, shopping at Prudential, and attending an all-day summer festival at Boston Commons. To plan the best route, we'll use our algorithm, starting from Downtown Crossing, to figure out the shortest distance and make the most of her trip.



| | |
|---|---|
| 🔴 | Downtown Crossing |
| 🟢 | Seaport |
| 🟣 | Prudential |
| 🟡 | Northeastern University |
| ⚫ | Boston Common |

**Distance Data :**

Downtown crossing to NEU : 2.8 miles ;Downtown crossing to Prudential : 1.7 miles

Downtown crossing to seaport : 1.1 miles; Downtown crossing to BC: 0.7 miles

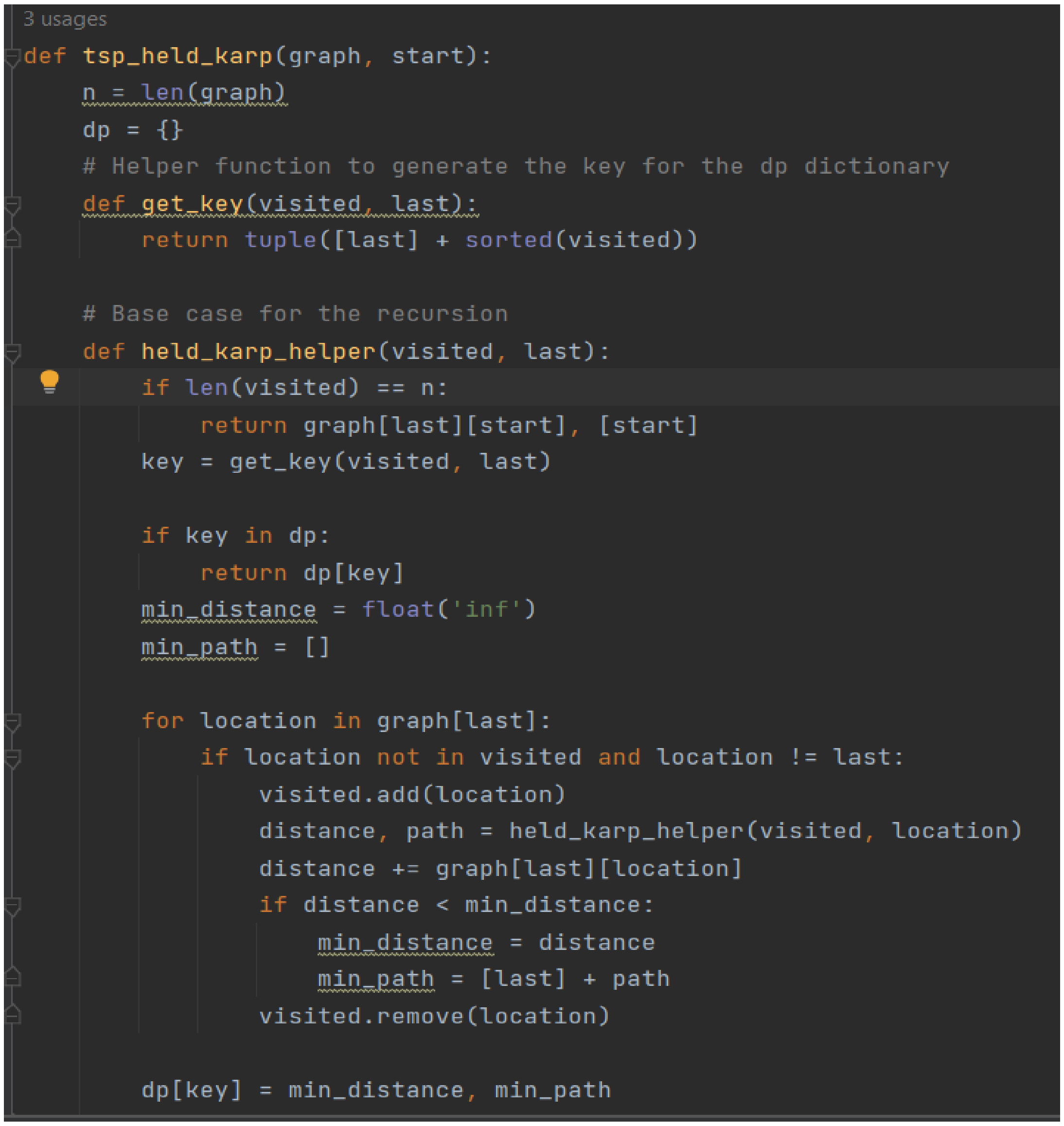
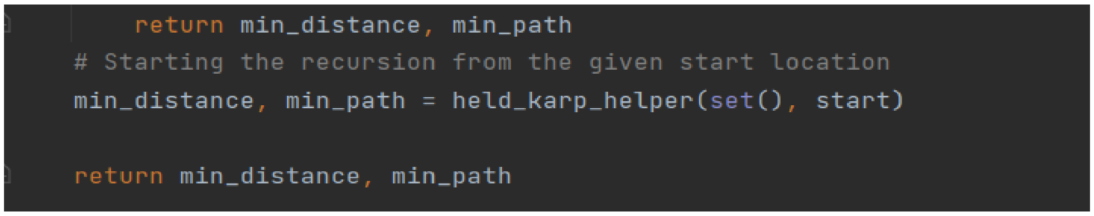
NEU to Prudential : 0.7 miles ; Northeastern University to seaport : 3.0 miles

NEU to Boston common: 1.3 miles ; Prudential to seaport: 2.9 miles

Prudential to Boston common: 0.7 miles ; Seaport to Boston common: 1.5 miles.

After manually looking at all possible routes and distances, we found the following route had the shortest minimal distance of 6.20 miles: Downtown Crossing-> Seaport -> Northeastern -> Prudential -> Boston Commons -> Downtown Crossing.

## Code sample

```
3 usages
def tsp_held_karp(graph, start):
    n = len(graph)
    dp = {}
    # Helper function to generate the key for the dp dictionary
    def get_key(visited, last):
        return tuple([last] + sorted(visited))

    # Base case for the recursion
    def held_karp_helper(visited, last):
        if len(visited) == n:
            return graph[last][start], [start]
        key = get_key(visited, last)

        if key in dp:
            return dp[key]
        min_distance = float('inf')
        min_path = []

        for location in graph[last]:
            if location not in visited and location != last:
                visited.add(location)
                distance, path = held_karp_helper(visited, location)
                distance += graph[last][location]
                if distance < min_distance:
                    min_distance = distance
                    min_path = [last] + path
                visited.remove(location)

        dp[key] = min_distance, min_path
```

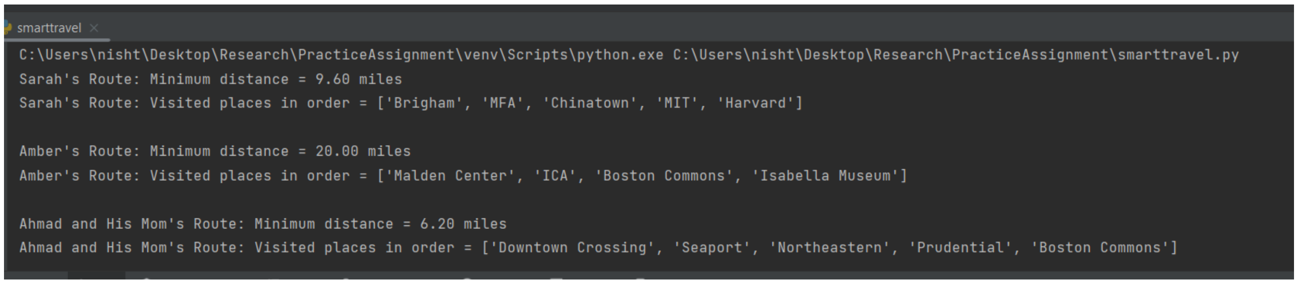
```
        return min_distance, min_path
    # Starting the recursion from the given start location
    min_distance, min_path = held_karp_helper(set(), start)

    return min_distance, min_path
```

**Explanation for Our Code:**

Our code implements the Held-Karp algorithm to help our three different visitors (Visitor 1, Visitor 2, and Visitor 3), each with their own distance graph representing the distances between various locations they want to visit. The function tsp_held_karp takes two inputs: the graph representing distances between locations and the starting location for the visitor. It uses dynamic programming and memoization to efficiently compute the minimum distance and the optimal path for each visitor to visit all the locations exactly once and return to the starting location. The algorithm iteratively explores different paths, keeping track of the minimum distance and corresponding path found so far. The output displays the minimum distance and the sequence of locations in order to achieve that minimum distance for each visitor. The code then applies the algorithm to three different visitor graphs, prints the minimum distances in two decimal places, and shows the order of visited places for each visitor.

After running the code, it gives the correct results of all the cases as shown below:



```
C:\Users\nisht\Desktop\Research\PracticeAssignment\venv\Scripts\python.exe C:\Users\nisht\Desktop\Research\PracticeAssignment\smarttravel.py
Sarah's Route: Minimum distance = 9.60 miles
Sarah's Route: Visited places in order = ['Brigham', 'MFA', 'Chinatown', 'MIT', 'Harvard']

Amber's Route: Minimum distance = 20.00 miles
Amber's Route: Visited places in order = ['Malden Center', 'ICA', 'Boston Commons', 'Isabella Museum']

Ahmad and His Mom's Route: Minimum distance = 6.20 miles
Ahmad and His Mom's Route: Visited places in order = ['Downtown Crossing', 'Seaport', 'Northeastern', 'Prudential', 'Boston Commons']
```

## What's Next?

As we discussed earlier, our initial problem-solving approach involved several assumptions, such as considering only cars as the mode of transportation, accommodating one visitor at a time, overlooking visiting and operating times for each location, and assuming that the shortest distance translates to the shortest time.

When conceptualizing this feature for Google Maps, we recognized the need to expand the transportation options. In order for this feature to fully function, we should incorporate walking and public transportation as well. This way, users can be presented with the best route that optimizes time efficiency, whether by a single mode of transportation or a combination of different modes, based on their preferences.

Real-time traffic information will be crucial to include in the directions, just as it's already available in Google Maps. By integrating this data, users can choose the most efficient route, taking into account the current traffic conditions, which ultimately helps them save time during their travels.

Another essential factor to consider is the operating hours of each location. For instance, if a location like the Museum of Fine Arts has limited operating hours from 10 am to 5 pm, the algorithm should avoid suggesting a route where it becomes impossible to visit the museum before it closes. Taking into account operating times ensures that the feature remains practical and efficient for users' needs.

The development of this feature for Google Maps presents a unique opportunity to revolutionize travel planning for users. By expanding the transportation options to include walking and public transportation, individuals can now access a more versatile and sustainable route selection. Integrating real-time traffic information offers the added benefit of making informed decisions to avoid delays and congested areas, ultimately saving valuable time.

Additionally, the consideration of operating hours for each location is crucial in providing practical and achievable itineraries. This ensures users can confidently explore their desired destinations without being constrained by opening and closing times.

By successfully incorporating multiple modes of transportation, real-time traffic updates, and operating hours into our algorithm, we are poised to deliver a highly efficient and user-centric feature. It aims to empower travelers to make the most of their time while exploring Boston or any other city, offering a seamless and enjoyable travel experience. With this new feature, Google Maps becomes an indispensable tool for users looking to optimize their journeys and create lasting memories in the places they visit.

## Conclusion

In conclusion, our research project was a resounding success, as we were able to develop an innovative and user-friendly solution to find the shortest path between different places for travelers. Leveraging the Held-Karp algorithm, we achieved accurate and efficient route planning, ensuring visitors can optimize their travel itineraries and save valuable time and gas. The model's simplicity and accessibility will make it easy for users to generate optimized routes effortlessly, satisfying their travel preferences and ensuring a seamless journey experience. By providing eco-friendly options that minimize gas consumption, our model not only benefits individual tourists but also contributes to global efforts in reducing carbon emissions and promoting sustainable travel practices. As travelers embrace our tool, they can embark on their adventures with confidence, knowing they have a reliable companion to guide them toward an enjoyable and environmentally conscious travel experience.

**Nishtha**:
During this project, I delved into the intriguing Traveling Salesman Problem (TSP) and its connection to our topic. I learned why the greedy approach using Dijkstra's method is inadequate for solving this problem. Eventually, we settled on the Held-Karp algorithm, which provided a more systematic way to tackle the TSP. However, it came with a drawback: its time complexity makes it less efficient for larger instances. This might explain why services like Google Maps have not yet incorporated this feature. But most importantly, this project taught me how to be adaptable and find the right balance between getting good results and making things work quickly and efficiently. I feel more confident in approaching complex problems and evaluating potential solutions critically, and this knowledge will undoubtedly benefit my future pursuits in computer science.

**Yiming:**
In the culmination of this project, the journey from research to implementation has been a rich learning experience. Navigating the intricate landscape of the Traveling Salesman Problem (TSP), I gained insights into the complexities of optimizing routes while visiting multiple destinations. The exploration of various algorithms, from Minimum Spanning Trees to Dijkstra's method, illuminated the nuances of each approach and revealed the unique demands of the TSP. Through careful consideration and teamwork, the selection of the Held-Karp algorithm emerged as a pivotal decision. Its dynamic programming prowess and systematic approach to solving subproblems showcased the power of algorithmic thinking. As a team, we harnessed this algorithm to craft an efficient solution, weaving together code that translated theory into practical route optimization. This project taught me the value of adaptability and the art of striking a balance between solution quality and computational efficiency. In this journey, I not only deepened my understanding of algorithms but also cultivated essential teamwork and problem-solving skills that will undoubtedly resonate in future endeavors.

**Ahmad:**
This research project has been an immensely fulfilling journey, and I couldn't be prouder of our achievements. As I prepare to present my research to potential job interviewers, I feel a strong sense of accomplishment and excitement. This project not only allowed me to explore the captivating world of advanced algorithms but also provided a platform to apply my CS5800 learnings to a real-world problem. Through this endeavor, I have deepened my understanding of complex algorithms and their practical applications, while honing my problem-solving and critical thinking skills. Looking back on this research journey, I am grateful for the knowledge and experiences it has bestowed upon me. I am eager to share how this project has contributed to my growth as a computer scientist and how it can positively impact travel efficiency and enjoyment worldwide.

## Side Note: transitioning in choosing topic

Changing the research topic turned out to be a great decision, and we take immense pride in the results of our research project. The challenges we encountered during our initial exploration of combining two techniques in bioinformatics to analyze DNA strands proved to be invaluable learning experiences. Those obstacles taught us to approach our new topic with a deeper understanding and greater tenacity. As a result, we were able to achieve a more successful outcome and make significant contributions to our new area of research. The experience reinforced the importance of adaptability and perseverance in scientific exploration. We are excited to showcase our research and the growth we have undergone as researchers, leveraging the lessons from our previous challenges to achieve a better outcome.

**Appendix (full code):**

```
def tsp_held_karp(graph, start):
     n = len(graph)
     dp = {}
# Helper function to generate the key for the dp dictionary
def get_key(visited, last):
     return tuple([last] + sorted(visited))
```

```
# Base case for the recursion
def held_karp_helper(visited, last):
     if len(visited) == n:
          return graph[last][start], [start]
     key = get_key(visited, last)

     if key in dp:
          return dp[key]
     min_distance = float('inf')
     min_path = []

     for location in graph[last]:
          if location not in visited and location != last:
               visited.add(location)
               distance, path = held_karp_helper(visited, location)
               distance += graph[last][location]
               if distance < min_distance:
                    min_distance = distance
                    min_path = [last] + path
               visited.remove(location)

     dp[key] = min_distance, min_path
     return min_distance, min_path
# Starting the recursion from the given start location
     min_distance, min_path = held_karp_helper(set(), start)

     return min_distance, min_path


# Sample graph representing distances between locations for Visitor 1
graph_visitor1 = {
'Brigham': {'Brigham': 0, 'Harvard': 2.7, 'MIT': 2.2, 'MFA': 0.6, 'Chinatown':
2.6},
'Harvard': {'Brigham': 2.7, 'Harvard': 0, 'MIT': 1.1, 'MFA': 3.8, 'Chinatown':
4.8},
'MIT': {'Brigham': 2.2, 'Harvard': 1.1, 'MIT': 0, 'MFA': 2.1, 'Chinatown':
2.8},
'MFA': {'Brigham': 0.6, 'Harvard': 3.8, 'MIT': 2.1, 'MFA': 0, 'Chinatown':
2.4},
'Chinatown': {'Brigham': 2.6, 'Harvard': 4.8, 'MIT': 2.8, 'MFA': 2.4,
'Chinatown': 0}
}

# Graph for Visitor 2
graph_visitor2 = {
'Malden Center': {'Malden Center': 0, 'Boston Commons': 6.0, 'ICA': 7.2,
'Isabella Museum': 8.8},
'Boston Commons': {'Malden Center': 6.0, 'Boston Commons': 0, 'ICA': 1.3,
'Isabella Museum': 2.7},
'ICA': {'Malden Center': 7.2, 'Boston Commons': 1.3, 'ICA': 0, 'Isabella
Museum': 4.3},
'Isabella Museum': {'Malden Center': 8.8, 'Boston Commons': 2.7, 'ICA': 4.3,
'Isabella Museum': 0}
}

# Graph for Visitor 3
graph_visitor3 = {
'Downtown Crossing': {'Downtown Crossing': 0, 'Northeastern': 2.8,
'Prudential': 1.7, 'Seaport': 1.1, 'Boston Commons': 0.7},
'Northeastern': {'Downtown Crossing': 2.8, 'Northeastern': 0, 'Prudential':
0.7, 'Seaport': 3.0, 'Boston Commons': 1.3},
'Prudential': {'Downtown Crossing': 1.7, 'Northeastern': 0.7, 'Prudential': 0,
'Seaport': 2.9, 'Boston Commons': 0.7},
'Seaport': {'Downtown Crossing': 1.1, 'Northeastern': 3.0, 'Prudential': 2.9,
'Seaport': 0, 'Boston Commons': 1.5},
'Boston Commons': {'Downtown Crossing': 0.7, 'Northeastern': 1.3,
'Prudential': 0.7, 'Seaport': 1.5, 'Boston Commons': 0}
}

# Calculate the minimum distance and path for Visitor 1, starting from
'Brigham'
min_distance_visitor1, path_visitor1 = tsp_held_karp(graph_visitor1,
'Brigham')

# Remove the duplicate 'Brigham' at the end of the path
path_visitor1.pop()

print("Sarah's Route: Minimum distance = {:.2f}
miles".format(min_distance_visitor1))
print("Sarah's Route: Visited places in order =", path_visitor1)
print("")

# Calculate the minimum distance and path for Visitor 2, starting from 'Malden
Center'
min_distance_visitor2, path_visitor2 = tsp_held_karp(graph_visitor2, 'Malden
Center')

# Remove the duplicate 'Malden Center' at the end of the path
path_visitor2.pop()

print("Amber's Route: Minimum distance = {:.2f}
miles".format(min_distance_visitor2))
print("Amber's Route: Visited places in order =", path_visitor2)
print("")
# Calculate the minimum distance and path for Visitor 3, starting from
'Downtown Crossing'
min_distance_visitor3, path_visitor3 = tsp_held_karp(graph_visitor3, 'Downtown
Crossing')
```

```python
# Remove the duplicate 'Downtown Crossing' at the end of the path
path_visitor3.pop()

print("Ahmad and His Mom's Route: Minimum distance = {:.2f}
miles".format(min_distance_visitor3))
print("Ahmad and His Mom's Route: Visited places in order =", path_visitor3)
print("")
```