Route Planning and Optimization using Greedy and DP Approach

Vidhi Parekh, Dhairya Amin, Sai Rahul Ponnana

Video Presentation Link

https://youtu.be/miC0t26wDcc

Project Introduction

When finding an optimal path between source and destination is in question, people usually talk about finding the shortest path. Considering real-life examples like food delivery, package/mail delivery, and public transport, a lot of elements such as cost, time, and convenience are factored into finding the best route. Due to the time constraint of the project and our group members' being students, we will only care about the distance of the route and will try to answer the question, “How can we find the shortest route between two locations?”.

Project Goals

- This program helps create a road trip or plan a flight route from one state or city to another with a variety of algorithms and traversals.
- Given an input and ending state this program will utilize a dataset of attractions in the United States to find the most efficient path while still hitting as many attractions as is possible.
- The algorithms used are Kruskal's MST algorithm, BFS, and Dijkstra's algorithm.

How can we find the shortest route between two locations?

We will find the shortest path using Kruskal's and Dijkstra's algorithms followed by a dynamic programming approach using the Bellman equation. Using the analytic and problem-solving skills learned from this class, we will explore how those two approaches will perform on our chosen dataset and figure out if they help us find the shortest route between two locations.

How do those algorithms perform on a real-world dataset?

Kruskal's algorithm is very easy to implement compared to Prim's algorithm because we don't have to care about the adjacency vertices in this algorithm. In this algorithm, Kruskal's algorithm starts from all the vertices of the graph. We choose the minimum weight edge vertice and start creating the minimum spanning tree. During creating the minimum spanning tree of a graph, we also have to take care of that graph, not creating any cycles because cycles should not be in the minimum spanning tree. We also wish to provide a comparison of the performances of all the algorithms on our dataset so that we can answer this question

Dataset

The cities.txt file in our repository includes different parameters and has source, destination and the hypothetical distance between the two cities(Source and Destination) in miles.

```
Chicago Boston 117.45
Minneapolis Chicago 148.27
SaltLakeCity Miami 183.43
WashingtonDC Houston 185.34
WashingtonDC SaltLakeCity 597.09
Denver Albuquerque 224.50
KansasCity Minneapolis 245.71
Nashville Minneapolis 269.04
KansasCity Nashville 287.16
Atlanta Nashville 301.76
```

Kruskal's Algorithm

When looking for the shortest path in a graph, we come across a term known as minimum spanning tree or MST. MST is a subset of the original graph that contains the same vertices and V-1 number of edges (V being the number of vertices in the original graph).

The sum of weights of edges of the MST is of the minimum value. One of the algorithms that is widely used to generate a minimum spanning tree is Kruskal's algorithm.

- In Kruskal's algorithm, we first sort the edges in an ascending order based on their weights, and we add nodes to the minimum spanning tree if it does not create a cycle.
- To check if an edge does not create a cycle, it uses a union-find algorithm. It uses a greedy approach by picking the edge with the least weight at each iteration so, at every turn it makes a local optimal choice and in the end we get a global optimal solution.
- The main advantage of using Kruskal's is that it will guarantee a path that has the least weight i.e. distance in our case.
- **Time complexity:** The algorithm takes O(ElogE) time where E is number of edges

```
print("The minimum spanning tree with Kruskal's algorithm is:")
g.kruskal()
```

```
The minimum spanning tree with Kruskal's algorithm is:
Chicago to Boston with distance 117.45 miles
Minneapolis to Chicago with distance 148.27 miles
SaltLakeCity to Miami with distance 183.43 miles
WashingtonDC to Houston with distance 185.34 miles
Houston to SaltLakeCity with distance 215.91 miles
Denver to Albuquerque with distance 224.50 miles
KansasCity to Minneapolis with distance 245.71 miles
Nashville to Minneapolis with distance 269.04 miles
Atlanta to Nashville with distance 301.76 miles
Dallas to LasVegas with distance 316.74 miles
Omaha to KansasCity with distance 317.31 miles
Chicago to Dallas with distance 327.20 miles
Denver to StLouis with distance 331.12 miles
Atlanta to Denver with distance 350.38 miles
Seattle to Sacramento with distance 359.56 miles
Albuquerque to Sacramento with distance 359.68 miles
Houston to Boston with distance 384.70 miles
Phoenix to Omaha with distance 402.34 miles
Charolette to Omaha with distance 604.25 miles
```

Dijkstra's Algorithm

Dijkstra's algorithm is a well-known algorithm that has broad applications in the industry concerning networks or GPS especially when the algorithm is customized to suit a company's needs. Since we only have the distance between two cities i.e. positive values, Dijkstra's will help us find the shortest path between a source city and all other cities in the dataset using the distances between them.

- We keep track of the shortest distances of nodes from the source city and those distances are initially set as infinity except for the source city which is 0.
- For a city whose shortest distance is yet to be calculated, we iterate through the adjacent cities of that city and then update those values only if the current distance plus the edge weight is less than the current distance.
- The shortest distance is kept track of greedily for each city. In the end, we can just return the shortest distance value of the destination city.
- **Time complexity:** The algorithm takes O(V^2) time where V is number of vertices

- Washington DC to Salt Lake City = 597.09 miles
- Washington DC to Houston = 185.34 miles
- Houston to Salt Lake City = 215.91 miles
- Dijkstra's accurately picks the shortest path as Washington DC to Houston to Salt Lake City with 401.25 miles

```
Direct distance of SaltLakeCity from WashingtonDC is 597.09 miles
Shortest distance of SaltLakeCity from WashingtonDC is 401.25 miles
```

Bellman Equation

Here the algorithm aims to approximate the function j() which provides the least path cost from a node to the target. Assuming we know this function, then we can write the Bellman equation:

- j(nodeA) = min [C(nodeA, nodeB) + j(nodeB)] over all nodeB which are neighbours of nodeA
- C() is the transition cost (or step cost) to go from nodeA to nodeB (this corresponds to the distance or edge cost)
- j() is the path cost from the node to the goal The algorithm finds the function j() by successive iterations.

For our graph, the solution is found in about 30 iterations. The algorithm outputs a distance matrix with all node to node distance as well as the output of j for all nodes (ie shortest distance from any node to the goal). These are then used to derive the shortest path sequence. Complexity of this approach is O(n^2) where n is the number of nodes (locations).

Comparing Runtime

Kruskal's

O(ElogE) or O(ElogV)

Dijkstra's

O(V^2)

Bellman Equation

O(V^3)

Conclusion

Upon the completion of this project, it has given us more insight as to how distance between locations affect route planning even though there are multiple variables that are at play in the industry. We've realized how broad route planning can be, we can use multiple algorithms with different computational complexities and we can modify existing algorithms to meet our goals. This project has also helped us reflect on what we have learned in the class and to translate it to practical knowledge.

Future Scope

Due to the time complexity of the project, we are unable to explore more algorithms and huge datasets. Also, we intend to work on dynamic programmatic approaches for further exploration. In the future, we would also like to extend this project to explore and compare more approaches to come to a more well-informed opinion.

Thank You