The Best Taco Bell Path

Aryan Indarapu
Ashwin Jain
Rafael Wersom
Alejandro Pulido

What is the shortest path of Taco Bell stops from Champaign to Chicago?

Our Goals

- Gather data about Taco Bell locations in Illinois from Taco Bell's website
- Build a graph of Taco Bell locations using that data
- Use Dijkstra's Algorithm to determine the shortest path through Taco Bells from Champaign to Chicago
- Understand the betweenness centrality of both the Champaign and Chicago Taco Bell locations
- Determine connectedness between two nodes using BFS

Data Scraping/Parsing

- Used BeautifulSoup library to scrape data from Taco Bell website
- To make graph building easier, the latitude and longitude of the nearest locations were replaced with their CSV id
- Certain locations were out of the state of Illinois
  - Replaced these location IDs with a -1
- Challenges
  - Some hyperlinks had multiple locations within them
  - Finding a unique tag that corresponded to lat/long values

Graph Structure

- TacoBellGraph is the name of the class used for the graph structure composed of TacoBellNode(s)
- Each TacoBellNode stores the information from one Taco Bell in Illinois (id, address, latitude, and longitude)
- TacoBellGraph is a weighted (distance) and directed graph
- Each TacoBellNode also has degree 3 of outgoing edges to the 3 closest Taco Bells
- TacoBellGraph uses Adjacency Lists to store the information about the edges

The Algorithms

- Tested Dijkstra's on a bunch of smaller graphs to make sure it was working
  - To allow for betweenness centrality algorithm to work, if there is no path between two nodes, the function returns an empty vector
- All of the shortest paths between pairs of vertices are unique, so betweenness centrality will always be an integer

What is the shortest path of Taco Bell stops from Champaign to Chicago?

Start: 512 E. Green Street
1707 S. Neil Street
582 Main NW
195 South Creek Drive
5737 W. Monee Manhattan Road
413 Sauk Trail
201 S Halstead St.
2945 West 159th Street
12716 Ashland Ave.
1644 W 95th St
7906 S. Western Avenue
5350 S Pulaski
4614 S Damen Ave
255 W Garfield Blvd
3365 S Martin Luther King Drive
End: 407 S. Dearborn

Answering the Leading Question

- Input: Champaign ID and Chicago ID in Taco Bell graph
- Function Run: Dijkstra's Search Algorithm
- Output: The list of Taco Bell Nodes from Champaign to Chicago
    - Used CSV to get addresses and output as a list of addresses instead

Reaching Our Other Goals

- Taco Bell graph was successfully built and tested using adjacency lists
- BFS is able to detect if a path between two nodes are connected, since the graph is not fully connected
- As hypothesized, the betweenness centrality of the Chicago Taco Bell is significantly higher (377) than the Champaign Taco Bell (80)
    - Shows that the Chicago Taco Bell will have a lot more paths through it

Thank You For Watching!

Dijkstra's Algorithm

- Allows us to find the best path from one location to another using distance between locations to find the shortest path
- Challenges:
  - Implementing a priority queue that functions correctly
    - We need to use a pair to give the distance from origin as well as the ID that corresponds with the distance.
    - Priority queue uses max heap, so we need to use negative values internally to allow the highest priority be the node with the shortest distance from the origin
  - Ensuring that our algorithm reaches the target before terminating the priority queue phase
    - We handle this by throwing an runtime_error to give us a insight when testing that our algorithm actually reaches the end
  - Some locations are out of the scope of our dataset
    - We marked these in nearest location id as -1
  - This causes our BFS to throw a out_of_bounds error as our visited/previous vectors would attempt to access index = -1
  - We handled this by making sure that we skip any id that is -1

Breadth First Search

- BFS allows us to see if two nodes are within a connected graph as some pairs of nodes have no paths between each other
- We would finish when our current node from our queue is our destination id
  - We would then build a vector starting at our destination node and using our previous vector until we reach our starting node (when previous[index] = -1)
- If we never found our destination we simply return a empty vector
- Challenges
  - Just like in our Dijkstra's Search Algorithm, we also saw problems with edges having destination id = -1
  - We implemented a safe check to skip those as well