Algorithms: Design & Analysis

Shortest Distance Problem

Maaz Saeed | Syed Ammar Mahdi | Syeda Zainab Fatima

Introduction

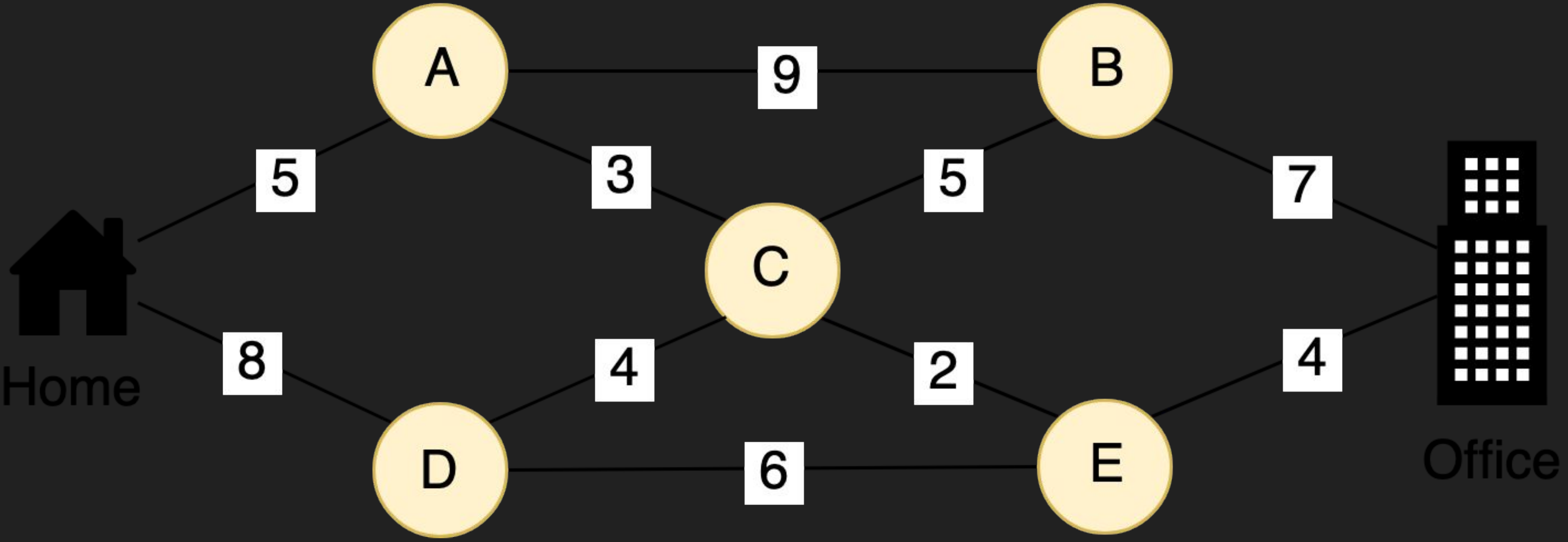
The ‘shortest path problem’, in graph theory, is the problem of finding the minimum distance required to travel one a given node, to another.

The problem arises regardless of whether your graph is directed, undirected, or mixed.

Think about how often you come across this problem in practical life, not just graphs.

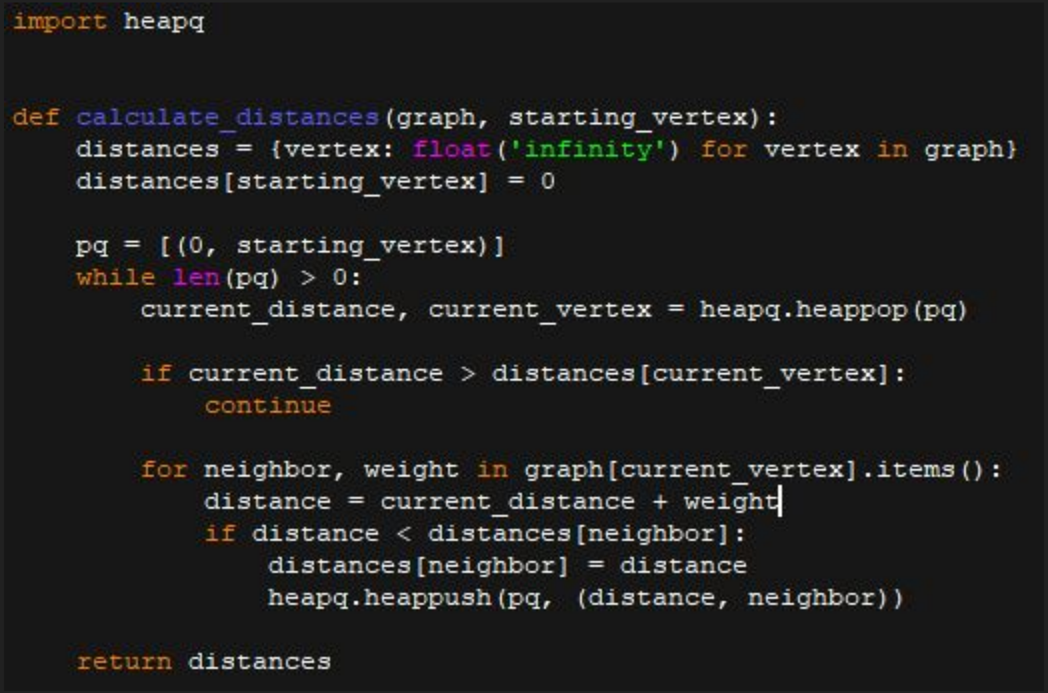
So how do we solve this problem?

Dijkstra’s Algorithm



(Newline, 2020)

Dijkstra’s Algorithm: How does it work?



```python
import heapq


def calculate_distances(graph, starting_vertex):
    distances = {vertex: float('infinity') for vertex in graph}
    distances[starting_vertex] = 0

    pq = [(0, starting_vertex)]
    while len(pq) > 0:
        current_distance, current_vertex = heapq.heappop(pq)

        if current_distance > distances[current_vertex]:
            continue

        for neighbor, weight in graph[current_vertex].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))

    return distances
```

(Bradfield CS, 2020)

Dijkstra’s Algorithm: Theoretical Analysis

Building the dictionary of distances: O(V)

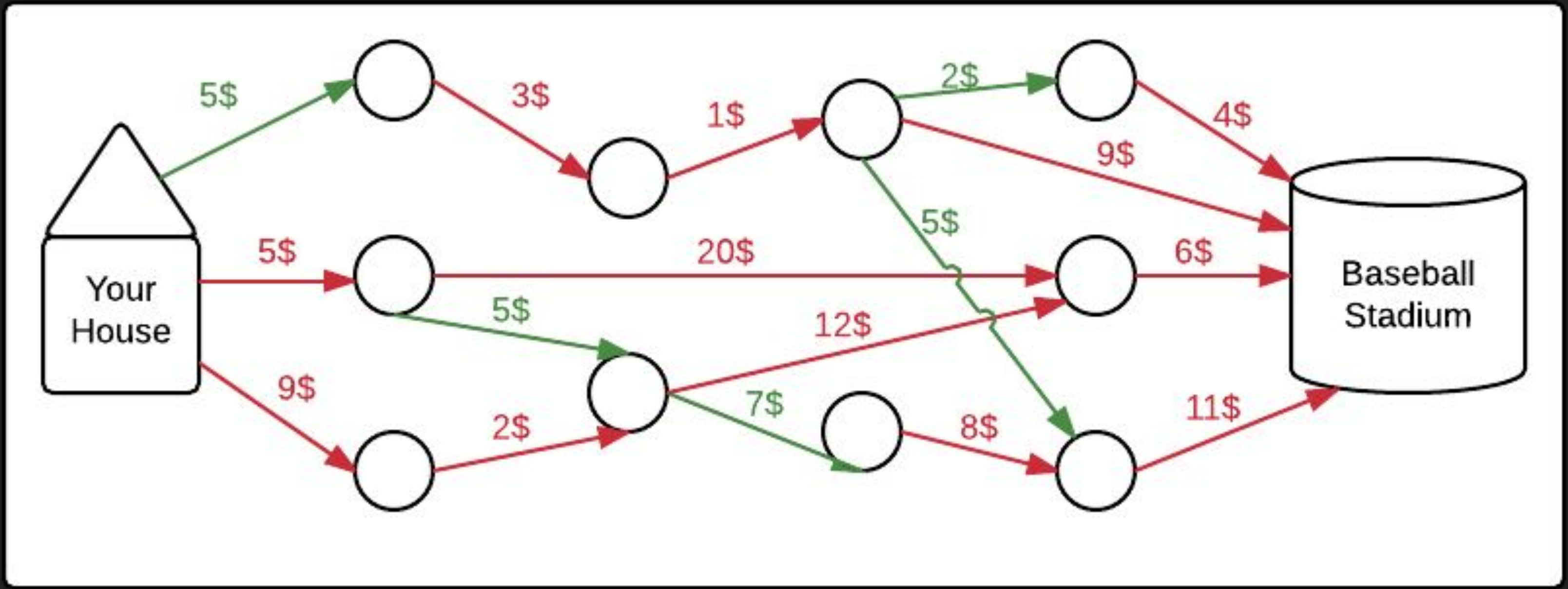
While loop: O(E)

For loop: O(E)

Adding/Removing priority queues: O(Log E)

Total theoretical running time: O(V + E Log E)

Bellman Ford Algorithm



("Bellman-Ford Algorithm | Brilliant Math & Science Wiki", 2020)

Bellman Ford algorithm has the same end result as Dijkstra’s.

Then why do we have a separate algorithm?

Bellman Ford’s algorithm works even with negative weighted edges whereas Dijkstra doesn’t.

Bellman Ford Algorithm

1. Initialize distances from source to all vertices as infinite. Create an array **dist** that contains these distances.

2. Run a loop **|V| -1** times, where **V** is the number of vertices.
   a. For each edge **u-v**, check if dist[v] > dist[u] + weight of edge **u-v.**
   b. If true, update dist[v] as dist[u] + weight of edge

3. This step checks for negative cycle. If dist [v] > dist[u] + weight of edge **u-v** then algorithm reports a negative cycle
   This step confirms that the output is the shortest distance indeed.

Bellman Ford’s ALgorithm Analysis

Step 1 -> Initialization takes O(V) time

Step 2 -> Inner loop takes O(E (V-1)) time **or** O(E V) time

Step 3 -> Detection of negative cycles takes O(E) time

Overall time taken by the algorithm -> O (V E)

Dijkstra's vs Bellman Ford’s :

Dijkstra’s Algorithm	**Bellman Ford’s Algorithm**
Greedy Algorithm	Not greedy
Can't work on negative edges or cycles	Works with negative edges but not cycles
Theoretical Run time: O(V + E log E)	O(V E)
Empirical Run time: 0.0079322 seconds	0.0540228 seconds

References

Newline.co. 2020. [online] Available at: <https://www.newline.co/books/javascript-algorithms/dijkstras-algorithm> [Accessed 30 November 2020].

Bradfieldcs.com. 2020. Shortest Path With Dijkstra’S Algorithm. [online] Available at: <https://bradfieldcs.com/algos/graphs/dijkstras-algorithm/> [Accessed 30 November 2020].

Bellman-Ford Algorithm | Brilliant Math & Science Wiki. Brilliant.org. (2020). Retrieved 1 December 2020, from https://brilliant.org/wiki/bellman-ford-algorithm/.

Thank You!