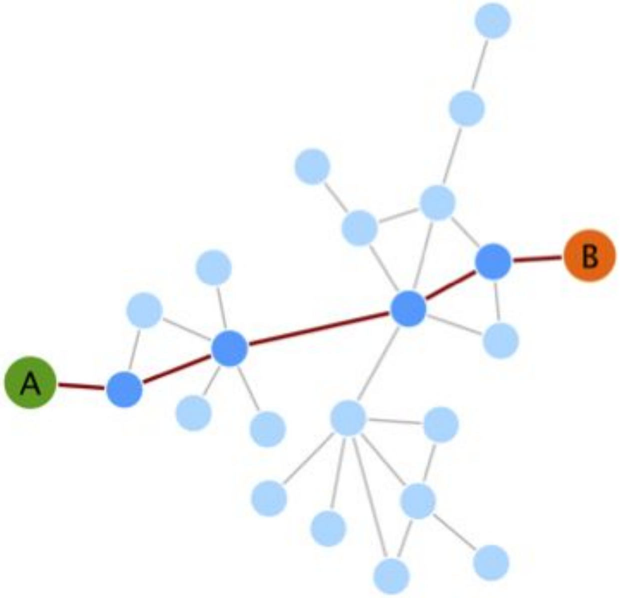
Dijkstra’s Algorithm

SCTG Lunch & Learn

What is Dijkstra's Algorithm?

Dijkstra's algorithm is used to find the shortest path between 2 nodes in a graph

Named after the Dutch computer scientist Edsger Dijkstra (1930-2002)

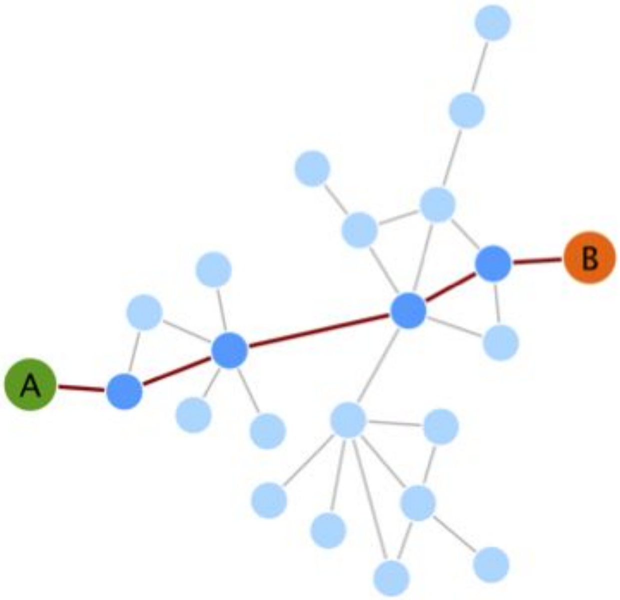


What is a graph?

A set of vertices (nodes) and connecting edges (arcs)

Example:

- Road/highway system
- Facebook/social network
- The internet

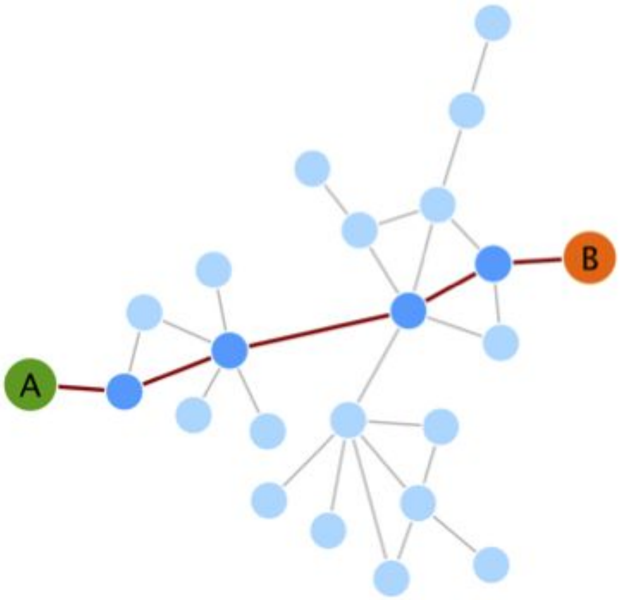


What is Dijkstra's Algorithm?

Algorithm to find the shortest path between 2 nodes in a directed graph.

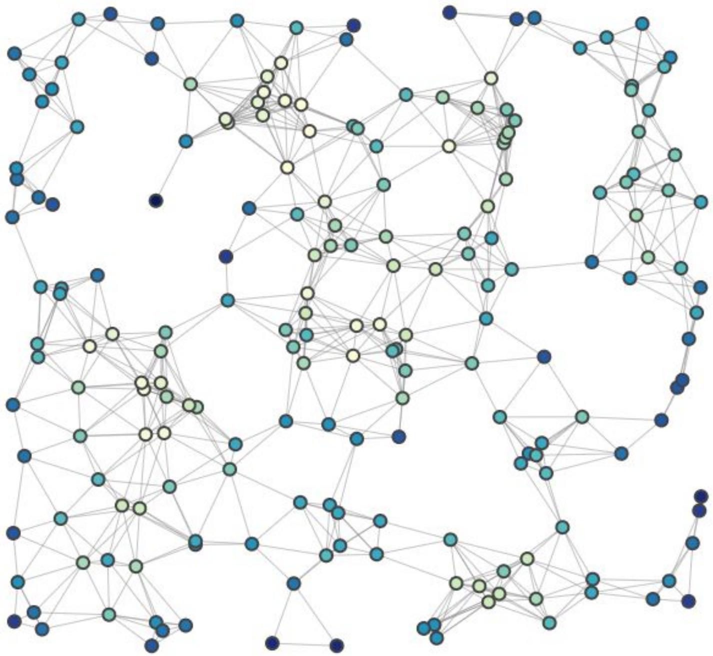
Shortest path is not the fewest number of edges traversed. It is the shortest path along weighted edges.

Note: Path is not *necessarily* distance. Could be number of connections, time of traversal, etc



Alternatives & Motivation

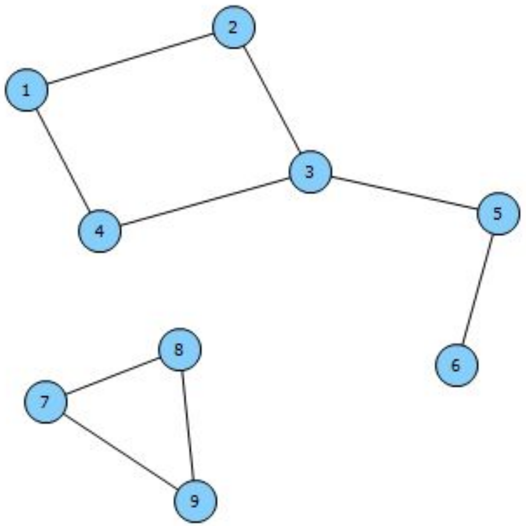
- BFS or DFS
- Find all paths
- Time Efficiency (Big O)



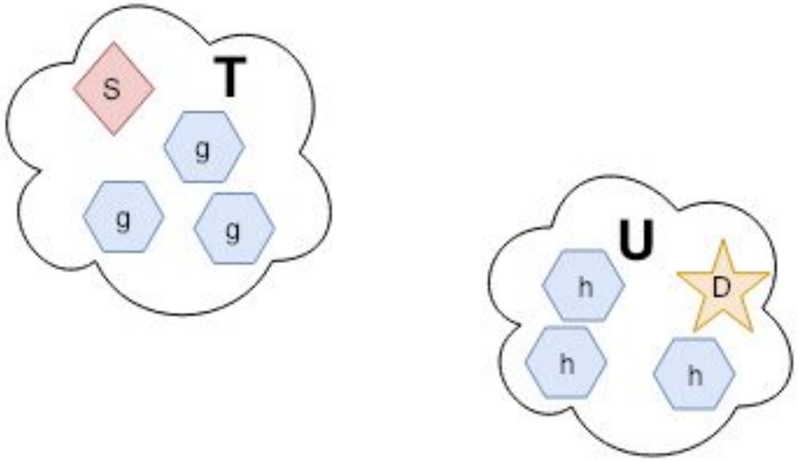
Dijkstra's Algorithm

Conditions:

- All nodes are known in advance (or discover all in pre-processing step)
- Graph is directed
- Edges have non-negative weight
- The nodes are connected*



Computation



S: Source/starting node

D: Destination/ending node

T: Set of nodes we have visited (traversed to)

U: Set of nodes we have not visited (un-visited)

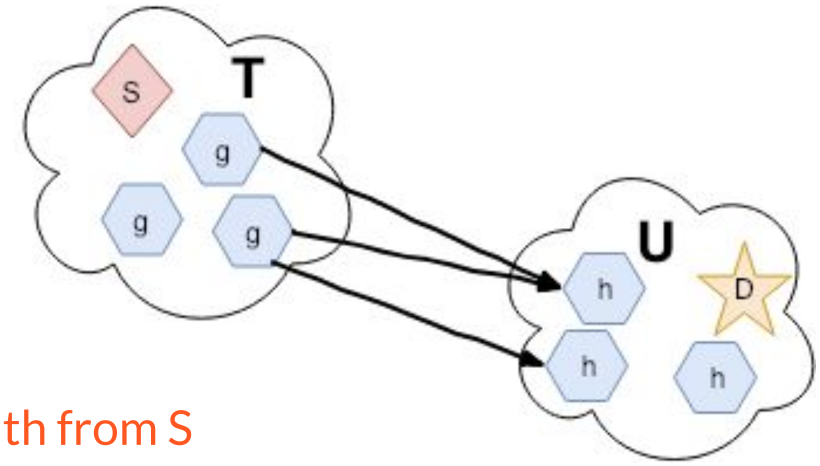
g: A node in the set T

h: A node in the set U

Computation

While D is not in T

- Find all edges (e) that cross *from* T *to* U
- Determine the edge, e*, with shortest path from S [through g] to h along e*
    - Dijkstra's greedy criterion
- Add h to T, remove from U
- Note: g does not need to be h from last iteration



Bookkeeping

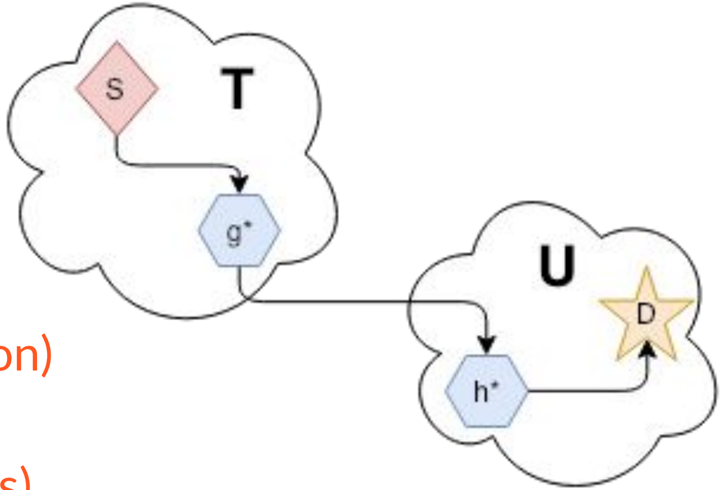
Whenever node is added to T, need to keep track of:

- ➔ **Total distance traveled**
  Shortest distance from S to g (so we know how far we will need to go)
- ➔ **Path traveled**
  Path from S to g (so we know how to traverse the graph)

Demo

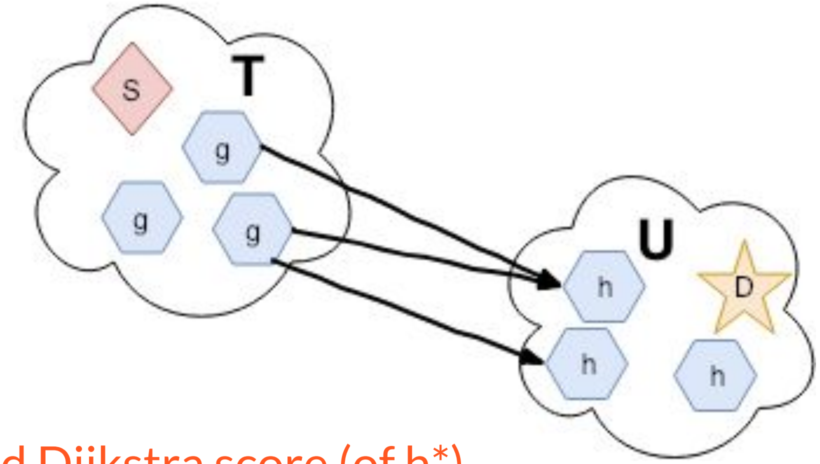
What is the fastest way to get a free lunch?

Proof of Correctness



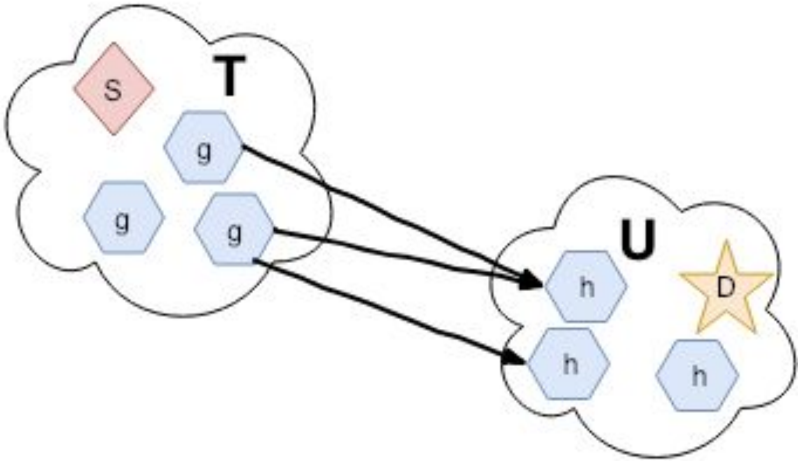
- Overall: determine some path from S to D
- Base: add S to V with length 0
- Iteration: add node h* to V
- Given S -> g* -> h* -> D,
    - S -> g* is shortest path to g* (by induction)
    - g* -> h* is some distance from g* to h*
    - h* -> D >= 0 (non-negative edge weights)
- But g* -> h* is shortest path (Dijkstra's greedy criterion)
- Therefore, chosen path S -> D is not longer than any other path from S->D

Run Time



- Heap: O(log(x)) inserts and extract min
- Heap of nodes in U
    - Value of node in U is smallest greed Dijkstra score (of h*)
    - If no edge from T to h, key of h is infinity (should never happen)
- Two rounds:
    - For each node h, determine shortest path from T to h
    - Determine shortest of all paths to h (result of 1st round)
- Every time node h* is removed from U, check every edge from h* into U and recalculate shortest path for that node

Run Time



- m = # edges; n = # nodes
- If path from S to every node, m >= n - 1
    - m + n <= 2m + 1 = O(m)
- We remove at most *n-1* nodes from heap of U -> O(n)
- Any edge will require at most 1 key recalculation (insert/delete) in heap of U
- Heap operations:
    - O(n) [extract min from heap] + O(m) [heap key recalculation]
    - O(m + n) operations = O(m) [from above]
    - Each heap operation takes log time -> O(m*log(n))

Further Enhancements

A* (A star) algorithm

- Extension of Dijkstra's algorithm
- "Best first search"
- Uses heuristic function to determine estimate of minimum distance from h to D
- Prevents going down paths in the "wrong direction"

Resources

Wikipedia:

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Stanford Graph Search and Data Structures Class

https://www.coursera.org/learn/algorithms-graphs-data-structures

Demo Code:

https://startupnextdoor.com/dijkstras-algorithm-in-python-3/

Code from the demo

https://github.com/ajstocchetti/dijkstra-lunch-and-learn

Questions?

If you were inspired and want to give your own Lunch & Learn, contact me