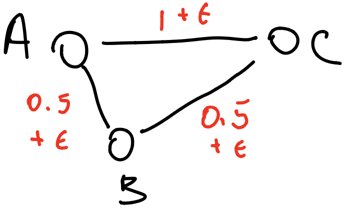
Shortest Paths

Hug's Slides

0.0001

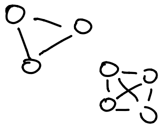
ε = very small #



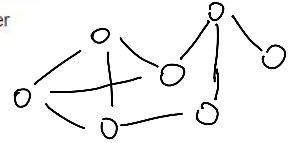
What?

- Preface: We've learned about BFS and DFS traversals, but those don't have edge weights (which are important for ex. in Google Maps)
- Dijkstra's Demo
    - Finds shortest path from one source node to **ALL** other nodes

- Dijkstra is just BFS if there are no edge weights or identical edge weights
- ***May Fail*** for negative edge weights! (won't if only neg weights are from start node)
- **Exam Tip:** If multiple paths give same distance and you want to find the one with the fewest edges, add a tiny constant number to each edge of the graph to ensure path with the least amount of edges is returned



- **General Steps:** implement recursively < DFS: Stack, BFS: Queue   Dijkstra: Priority Queue
    1. Insert all vertices into priority queue initialized with priority infinity
    2. Remove the vertex at top of queue, if a shorter distance is found from source to vertex **change the priority** of the vertex in the queue to the smaller number



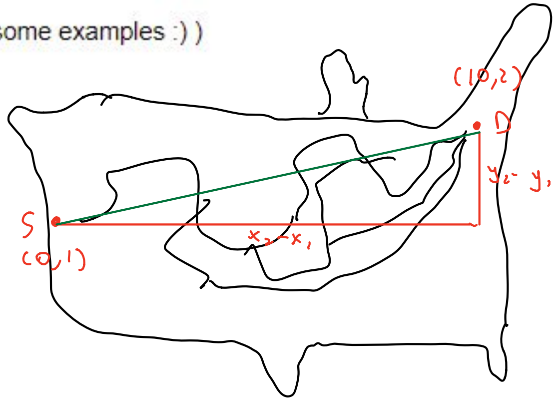
	# Operations	Cost per operation	Total cost
PQ add	V	O(log V)	O(V log V)
PQ removeSmallest	V	O(log V)	O(V log V)
PQ changePriority	E	O(log V)	O(E log V)

Assuming E > V, Total runtime is O(E log V) (when # V is large)

What? Continued

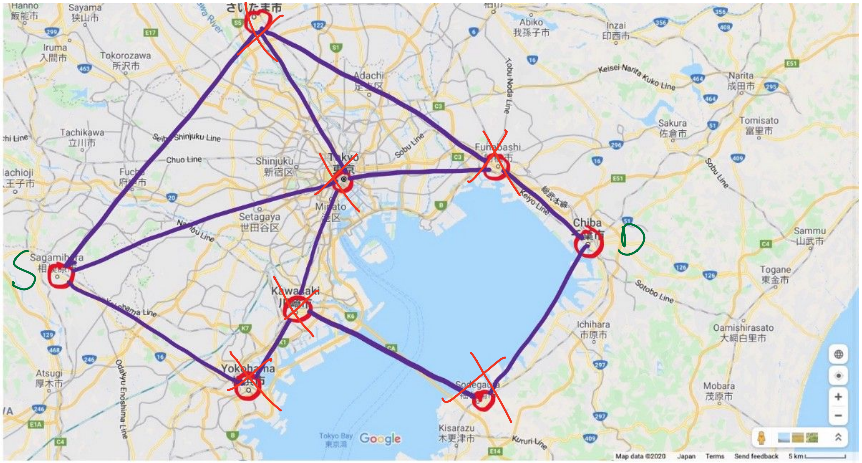
- A* Demo
  - Finds shortest path from one source node to **ONE** other nodes
    - Same as Dijkstra's, store priority as distance from source + heuristic (estimated distance to goal)
    - Unlike Dijkstra's, may not need to visit all vertices! Stop once the goal is visited
    - Runtime depends heavily on the heuristic function (take 188 for some examples :))
- An example heuristic is Manhattan Distance:

$(x_2 - x_1) + (y_2 - y_1)$



Why?

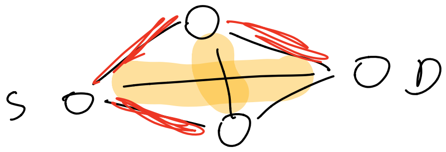
- Finding shortest path between a bunch of places (not just from point A to point B, but through many nodes in a graph like structure) is very common
- A* helps improve runtime on Dijsktra by having only ONE nodes in mind rather than ALL other nodes

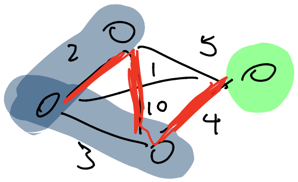


MST

Hug’s Slides

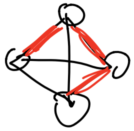
What?





- Goal is to find the set of edges within MINIMUM weight that connects all nodes
  - Tree-like structure, whereas shortest path can have cycles

- Tree (no cycles, connected) that includes all vertices in a graph with minimum weight, *only works on ***undirected graphs***
- <u>**Cut Property**</u>: assign graph's nodes to 2 different sets (**cut**); given any cut, minimum weight **crossing edge** (edge from one set to the other) is in the MST
- <u>**Cycle Property:**</u> The largest edge in a cycle will not be in the MST.
- If edges are *NOT UNIQUE*, there is a chance the MST is *NOT UNIQUE*
- Can't use Dijkstra's method (not exactly, anyways), no notion of a "source node"
- Results with <u>*V - 1 edges*</u> (given that trees have no cycles and must be connected, why can't the number of edges be anything else?)



Exam Tip - To see if adding an edge gives better MST, consider the largest edge of the MST coming out of that vertex

2 Famous Algorithms

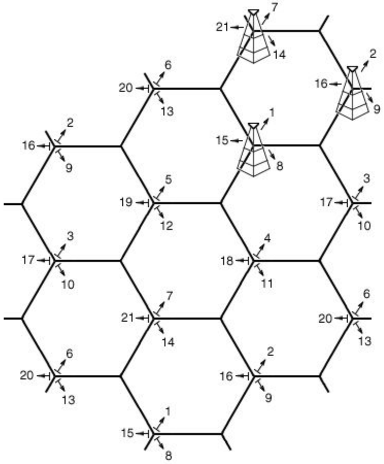
E log(V) - Dijkstra's

- Prim's Algorithm (Demo):
  - Very similar to Dijkstra's with one caveat - consider distance from TREE, not SOURCE
- Kruskal's Algorithm (Demo): — E log(E) - sorting edges
  - Sort ~~Consider~~ edges from smallest weight to largest, add the smallest edge to the MST unless it creates a cycle, and stop at V - 1 edges
  - Application of **Disjoint Sets**!

Why?

- MSTs are slightly different than Shortest Paths - aims to find minimum edges that COVERS a set of nodes, and not traveling along nodes
- Use cases include cell phone tower networks, maximum flow (CS 170 preview)

NP-Hard Problems

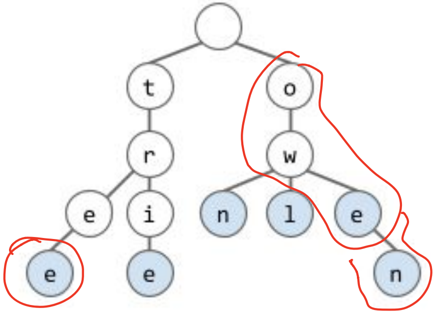


Tries

[Hug's Slides]

What?

- Used for 1 specific purpose - find if a String is a prefix of another String
  - Absolutely beautiful creatures - highlighted nodes represent the end of a word
  - Used in Autocomplete (Strings broken into chars)
  - Good to review **add** and **keysWithPrefix** methods
  - Random runtimes (Also check runtime table):
    - *Inserting N* strings length *M* -> **Θ(NM)**
    - *Finding all keys (length L) with a prefix of another key* -> **Θ(NL)**
    - *Finding the longest key that is a prefix of another key* -> **Θ(NL)**



This trie contains [**'tree'**, **'trie'**, **'own'**, **'owl'**, **'owe'**, **'owen'**]

Why?

- Used in any AutoComplete implementation on search engines
- Faster than BST or Hash Maps for their specific purpose (similar to how priority queue has one very specific purpose)
- Implementation