Recurring Section 11

Aniruth – 5 PM

Announcements

- This week covers a variety of algorithms and sorts.
  - These are all pretty complex and very important, so I would highly recommend you pay very close attention.
  - I anticipate we will talk about the algorithms next time.
- We’re almost there! Keep yourself going, because we are really close to the finish line.
  - We have some review stuff planned for next week to help you all before the final!

Content Review

Big Picture

- Two main classifications of graph algorithms in this class:
  - Minimum Spanning Trees
    - Prim's
      - Relies on cut property
    - Kruskal's
      - Relies on cycle detection and sorted edges
  - Shortest Paths (specifies a start vertex)
    - Dijkstra's
      - Relies on non-negative edges
      - If you're curious on how to make it work with all edges, check out Bellman-Ford (out of scope)
    - A*
      - Relies on a good heuristic
      - Specifies an end vertex too

Prim’s/A*

- These slides were ones I made during my interview for course staff. They’re a bit on the long side.

Graph Algorithms

MSTs (Cut Property, Prim's), A*

Aniruth Narayanan

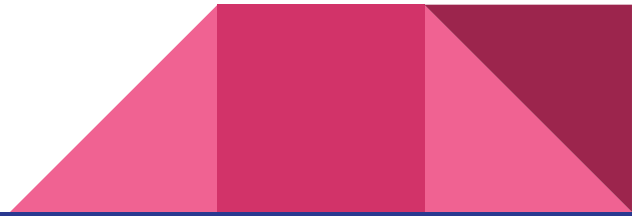
Minimum Spanning Tree (Review)

Minimum: Least edge weights

Spanning: Connects all nodes

Tree: Specifies the type of graph

Useful Property: Trees have |V|-1 edges



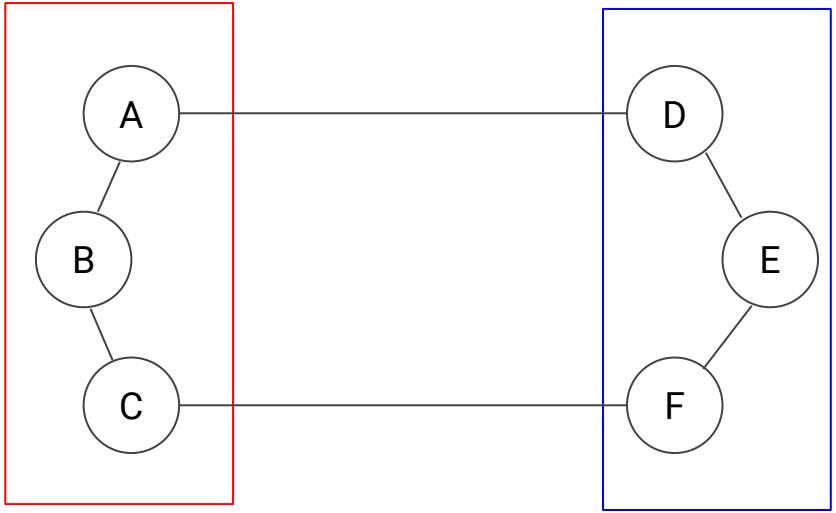
Cut Property

The Cut Property:

When assigning nodes to two non-empty sets, the minimum weighted edge between the two sets is in the MST.

To connect the two sets, there must be a **connecting edge** - MSTs are spanning.

The minimum weighted edge between the two sets is used in the MST.



For example, consider two groups of friends trying to communicate with each other.

Someone in one group must know someone in the other group for the two groups to be able to pass messages.

Extension to Prim's

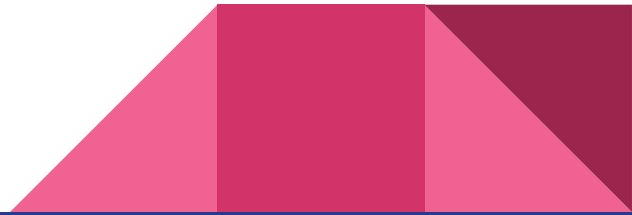
How can we use the cut property to create a generalized algorithm to create a MST?

Each "usage" of the cut property gives one edge to include in the MST.

To get every edge in the MST, the cut property must be applied repeatedly - which is Prim's algorithm.

More formally:

1. Start from some vertex, assign it to be in one set and every other vertex to be in another set.
2. Repeatedly, add the minimum weight edge between the two sets until there are |V|-1 edges.

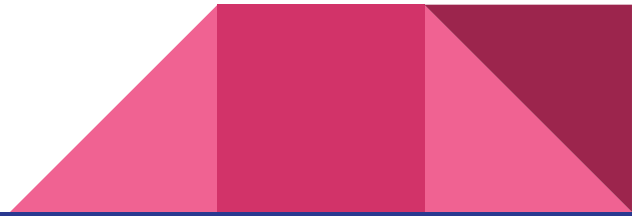


Implementation of Prim's

Implementation is similar to Dijkstra's, using distTo, edgeTo, and a *fringe PQ* (priority queue) to relax edges.

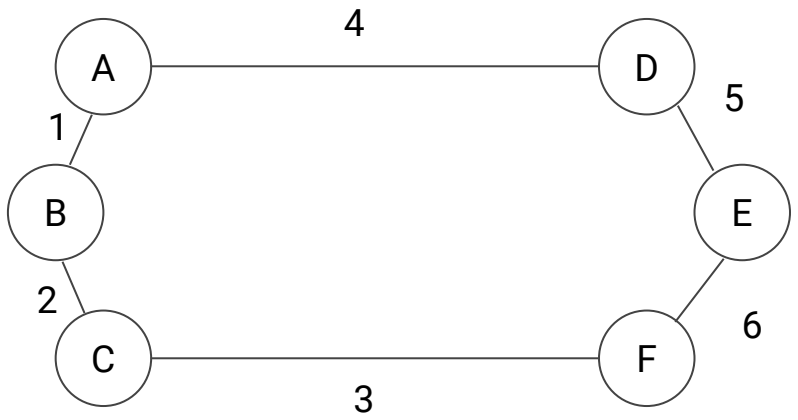
How are priority queues implemented (which data structure - important for runtime)?

The start vertex has distTo of 0 and every other vertex has distTo of infinity. The fringe stores the vertices in order of distTo.



Example of Prim's

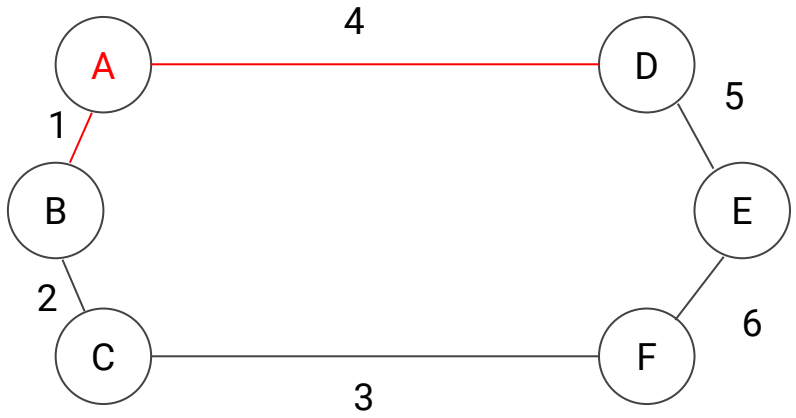
Vertex	distTo	edgeTo
A	0	-
B	inf	-
C	inf	-
D	inf	-
E	inf	-
F	inf	-



Fringe: [(A: 0), (B: inf), (C: inf), (D: inf), (E: inf), (F: inf)]

Example of Prim's

Vertex	distTo	edgeTo
A	0	-
B	~~inf~~ 1	A
C	inf	-
D	~~inf~~ 4	A
E	inf	-
F	inf	-



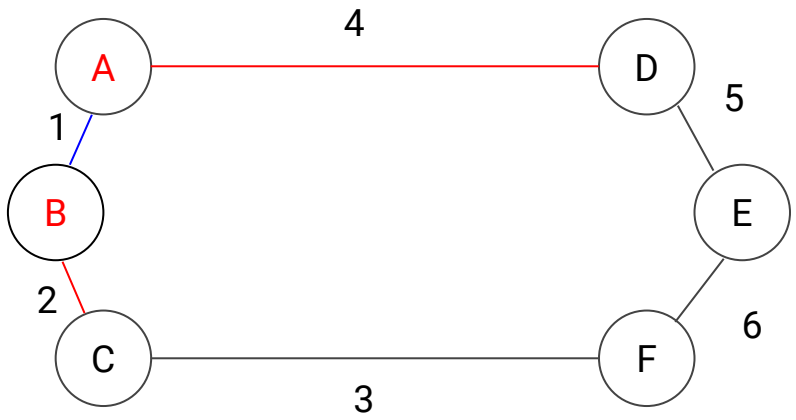
Fringe: [(A: 0), (B: inf), (C: inf), (D: inf), (E: inf), (F: inf)]

Fringe: [~~(A: 0),~~ (B: ~~inf~~ 1), (C: inf), (D: ~~inf~~ 4), (E: inf), (F: inf)]

Fringe: [(B: 1), (D: 4), (C: inf), (E: inf), (F: inf)]

Example of Prim's

Vertex	distTo	edgeTo
A	0	-
B	1	A
C	~~inf~~ 2	B
D	4	A
E	inf	-
F	inf	-



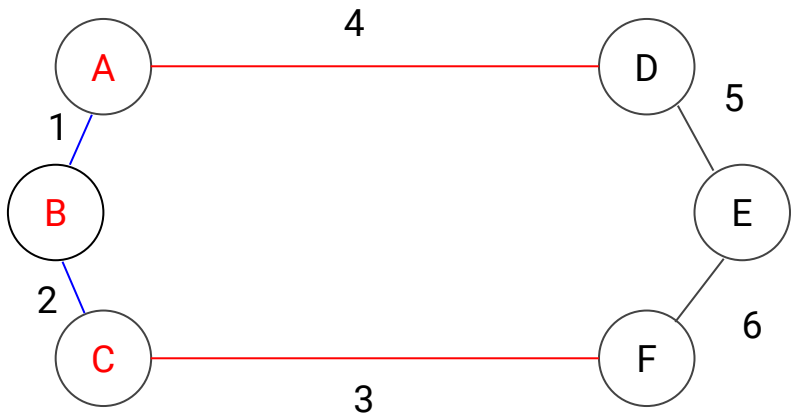
Fringe: [(B: 1), (D: 4), (C: inf), (E: inf), (F: inf)]

Fringe: [~~(B: 1)~~, (D: 4), (C: ~~inf~~ 2), (E: inf), (F: inf)]

Fringe: [(C: 2), (D: 4), (E: inf), (F: inf)]

Example of Prim's

Vertex	distTo	edgeTo
A	0	-
B	1	A
C	2	B
D	4	A
E	inf	-
F	~~inf~~ 3	C



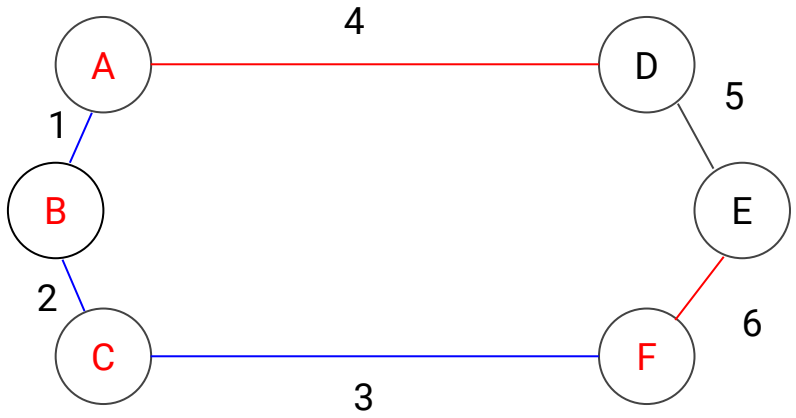
Fringe: [(C: 2), (D: 4), (E: inf), (F: inf)]

Fringe: [~~(C: 2)~~, (D: 4), (E: inf), (F: ~~inf~~ 3)]

Fringe: [(F: 3), (D: 4), (E: inf)]

Example of Prim's

Vertex	distTo	edgeTo
A	0	-
B	1	A
C	2	B
D	4	A
E	~~inf~~ 6	F
F	3	C



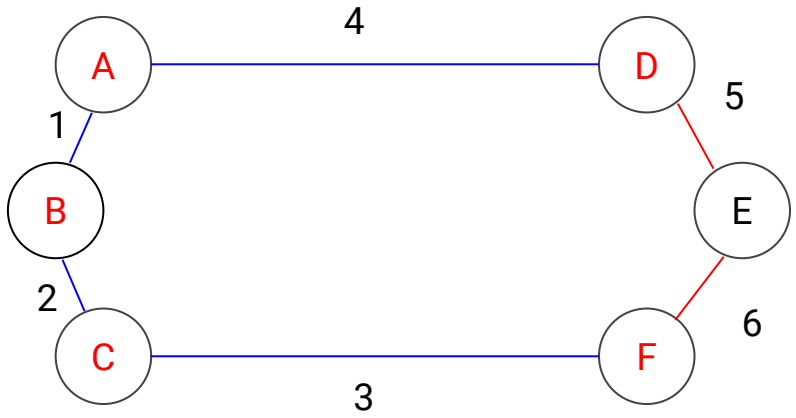
Fringe: [(F: 3), (D: 4), (E: inf)]

Fringe: [~~(F: 3)~~, (D: 4), (E: ~~inf~~ 6)]

Fringe: [(D: 4), (E: 6)]

Example of Prim's

Vertex	distTo	edgeTo
A	0	-
B	1	A
C	2	B
D	4	A
E	~~6~~ 5	~~F~~ D
F	3	C



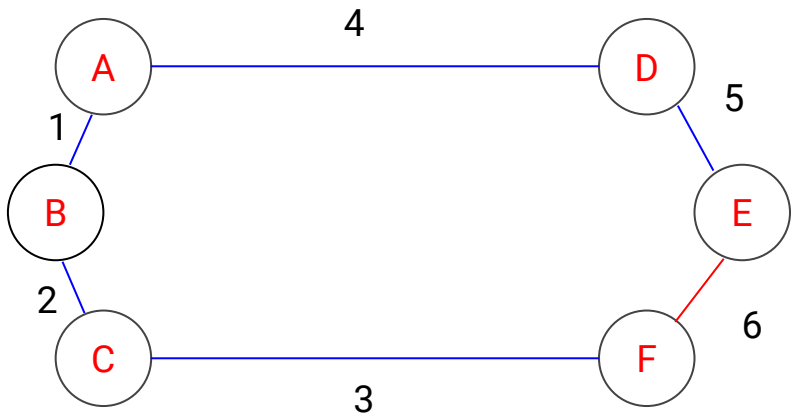
Fringe: [(D: 4), (E: 6)]

Fringe: [~~(D: 4)~~, (E: ~~6~~ 5)]

Fringe: [(E: 5)]

Example of Prim's

Vertex	distTo	edgeTo
A	0	-
B	1	A
C	2	B
D	4	A
E	5	D
F	3	C



Fringe: [(E: 5)]

Fringe: [~~(E: 5)~~]

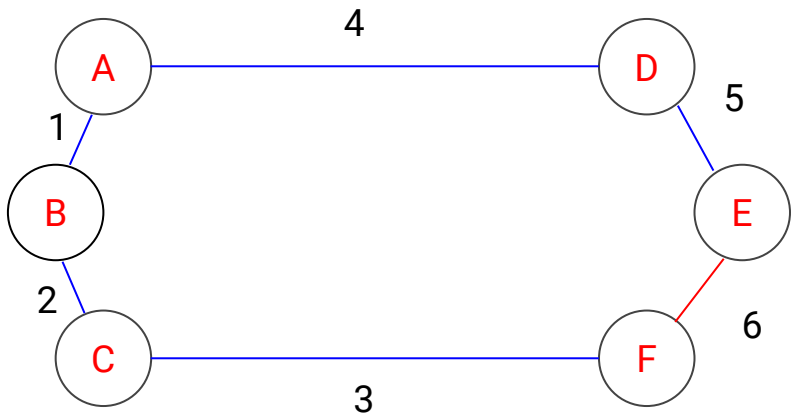
Fringe: [] - it's empty! We're done!

AB, AD, BC, CF, DE are the 5 edges for the MST of the 6 vertex graph above.

Total weight: 1 + 2 + 3 + 4 + 5 = 15

Example of Prim's

Vertex	distTo	edgeTo
A	0	-
B	1	A
C	2	B
D	4	A
E	5	D
F	3	C



There's an interesting "shortcut" here.

Earlier, I said trees have |V| -1 edges.

Here, there are 6 edges with 6 vertices, so the question becomes which singular edge to remove.

It ends up that the largest weighted edge, EF, is removed, leaving behind the remaining 5.

This is just a conceptual check that happens to work here.

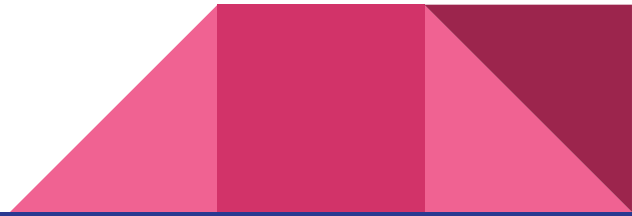
Runtime

Don’t memorize these slides; understand why at each step and for each component.

The algorithm has to do three things:

1. Insert vertices into the fringe
2. Delete the minimum from the fringe
3. Change the priority of vertices from the fringe

Break it down by step.



Calculating Runtime

How many vertices would need to be inserted?

What is the cost of each insertion?

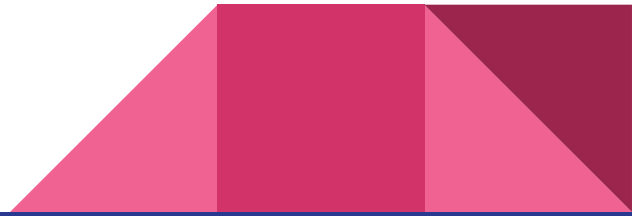
How many times would the minimum vertex be deleted?

What is the cost of each insertion?

How many times does each vertex need to be changed priority?

What is the cost of each change?

Total Runtime:



Calculating Runtime

How many vertices would need to be inserted? V

What is the cost of each insertion?

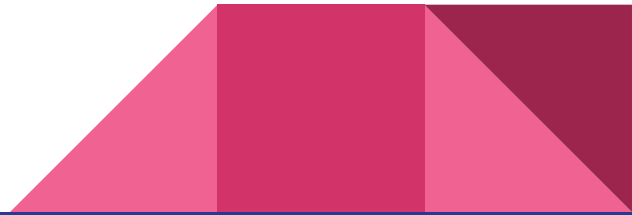
How many times would the minimum vertex be deleted?

What is the cost of each insertion?

How many times does each vertex need to be changed priority?

What is the cost of each change?

Total Runtime:



Calculating Runtime

How many vertices would need to be inserted? V

What is the cost of each insertion? O(logV)

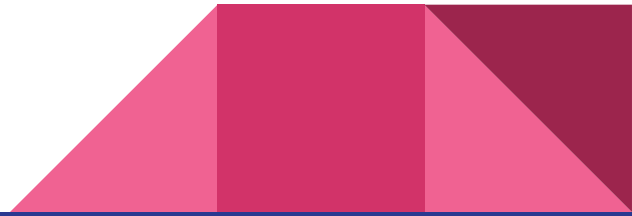
How many times would the minimum vertex be deleted?

What is the cost of each insertion?

How many times does each vertex need to be changed priority?

What is the cost of each change?

Total Runtime:



Calculating Runtime

How many vertices would need to be inserted? V

What is the cost of each insertion? O(logV)

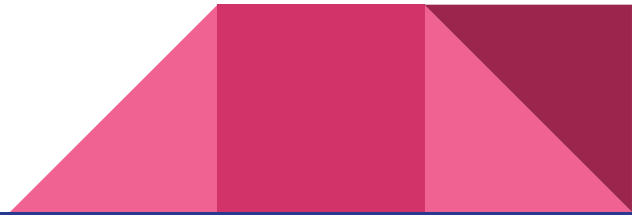
How many times would the minimum vertex be deleted? V

What is the cost of each insertion?

How many times does each vertex need to be changed priority?

What is the cost of each change?

Total Runtime:



Calculating Runtime

How many vertices would need to be inserted? V

What is the cost of each insertion? O(logV)

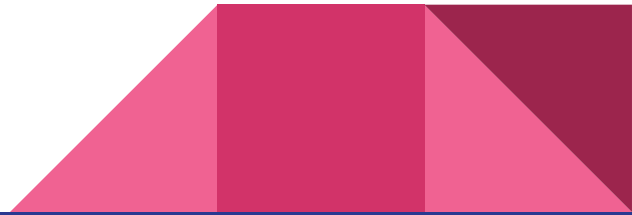
How many times would the minimum vertex be deleted? V

What is the cost of each insertion? O(logV)

How many times does each vertex need to be changed priority?

What is the cost of each change?

Total Runtime:



Calculating Runtime

How many vertices would need to be inserted? V

What is the cost of each insertion? O(logV)

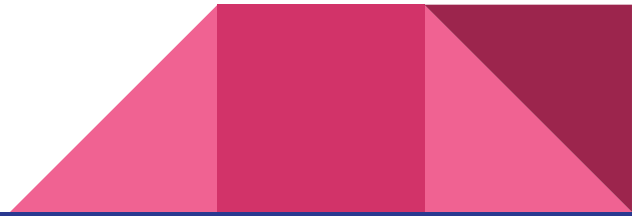
How many times would the minimum vertex be deleted? V

What is the cost of each insertion? O(logV)

How many times does each vertex need to be changed priority? O(E)

What is the cost of each change?

Total Runtime:



Calculating Runtime

How many vertices would need to be inserted? V

What is the cost of each insertion? O(logV)

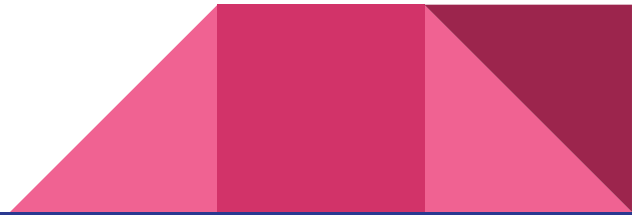
How many times would the minimum vertex be deleted? V

What is the cost of each insertion? O(logV)

How many times does each vertex need to be changed priority? O(E)

What is the cost of each change? O(logV)

Total Runtime:



Calculating Runtime

How many vertices would need to be inserted? V

What is the cost of each insertion? O(logV)

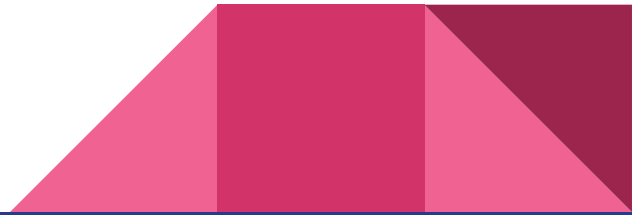
How many times would the minimum vertex be deleted? V

What is the cost of each insertion? O(logV)

How many times does each vertex need to be changed priority? O(E)

What is the cost of each change? O(logV)

Total Runtime: O(VlogV + ElogV) = *O(ElogV)*



More Advanced Prim’s Demo

https://docs.google.com/presentation/d/1GPizbySYMsUhnXSXKvbqV4UhPCvrt750MiqPPgU-eCY

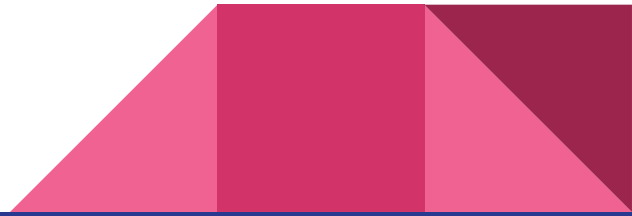
The above link is to a demo from lecture that follows the same procedure on a slightly more complicated graph.

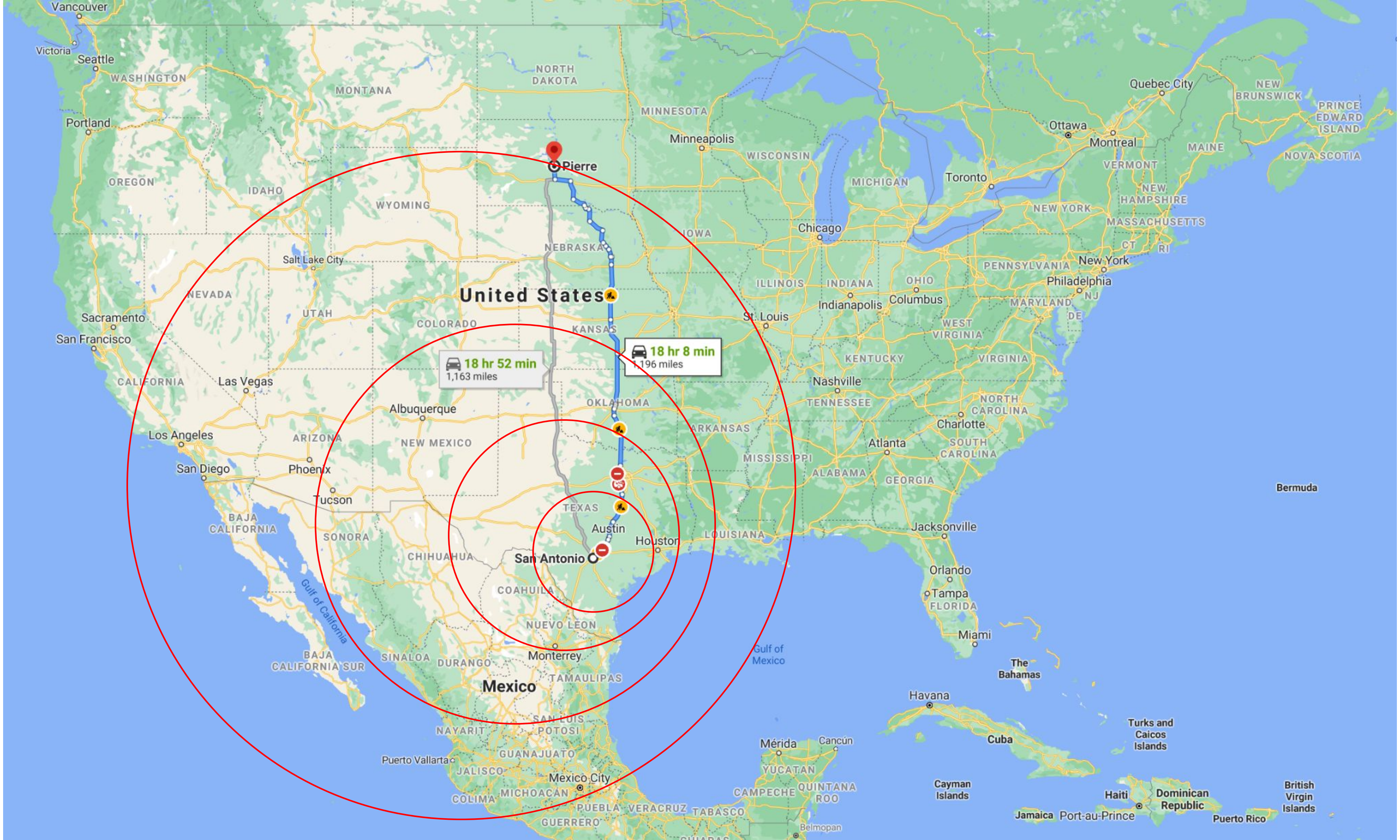
A*

What does Dijkstra’s do?

It finds the shortest path from a source node to every other node.

What if we want to find the shortest path from San Antonio, Texas, to Pierre, South Dakota?





Pierre
San Antonio
18 hr 52 min
1,163 miles
18 hr 8 min
1,196 miles
United States
Mexico
Vancouver
Victoria
Seattle
WASHINGTON
Portland
OREGON
IDAHO
MONTANA
NORTH DAKOTA
WYOMING
NEBRASKA
KANSAS
COLORADO
UTAH
NEVADA
Salt Lake City
Sacramento
San Francisco
CALIFORNIA
Las Vegas
Los Angeles
San Diego
ARIZONA
Phoenix
Tucson
NEW MEXICO
Albuquerque
OKLAHOMA
TEXAS
Austin
Houston
MINNESOTA
Minneapolis
WISCONSIN
IOWA
MICHIGAN
Chicago
ILLINOIS
INDIANA
Indianapolis
OHIO
Columbus
St. Louis
KENTUCKY
TENNESSEE
Nashville
ARKANSAS
MISSISSIPPI
ALABAMA
GEORGIA
Atlanta
LOUISIANA
Jacksonville
Orlando
Tampa
FLORIDA
Miami
SOUTH CAROLINA
NORTH CAROLINA
Charlotte
VIRGINIA
WEST VIRGINIA
MARYLAND
DE
NJ
Philadelphia
PENNSYLVANIA
New York
NEW YORK
CT
RI
MASSACHUSETTS
NEW HAMPSHIRE
VERMONT
MAINE
Toronto
Ottawa
Montreal
Quebec City
NEW BRUNSWICK
PRINCE EDWARD ISLAND
NOVA SCOTIA
Bermuda
Gulf of Mexico
The Bahamas
Havana
Cuba
Turks and Caicos Islands
Cayman Islands
Haiti
Dominican Republic
Port-au-Prince
Jamaica
Puerto Rico
British Virgin Islands
Mérida
Cancún
YUCATAN
CAMPECHE
QUINTANA ROO
TABASCO
Belmopan
VERACRUZ
PUEBLA
GUERRERO
Mexico City
MICHOACAN
COLIMA
JALISCO
GUANAJUATO
Puerto Vallarta
NAYARIT
SAN LUIS POTOSI
TAMAULIPAS
Monterrey
NUEVO LEON
COAHUILA
CHIHUAHUA
DURANGO
SINALOA
SONORA
BAJA CALIFORNIA
BAJA CALIFORNIA SUR
Gulf of California

Insight: End Vertex

Humans wouldn’t go outward in rings as they look for a path.

Instead, you would immediately go north from San Antonio - why?

Other paths are probably not going to work.

Can we do the same for Dijkstra’s? If so, what exactly are we trying to do?

Dijkstra’s doesn’t take in an end vertex into account during the stages of the algorithm.

A* is a way to bias Dijkstra’s summing the distance from the start vertex to the node, with a heuristic computing distance from the node to the end vertex.

Introducing the Heuristic

A heuristic is an estimation of the distance between a node and the end vertex.

Why can’t we just use the actual value between the node and the end vertex?

For example, if we’re going from San Antonio, Texas to Pierre, South Dakota, then we should probably head north than going south - which this heuristic could account for by finding the geographical distance between the intermediate cities visited.

We bias towards exploring up, because even though the distance from the start vertex is the same within a circle, accounting for the heuristic breaks that.

Conditions for a Working Heuristic

Admissible:

h(n, e) <= actual distance from n to e

The heuristic needs to underestimate the actual distance.

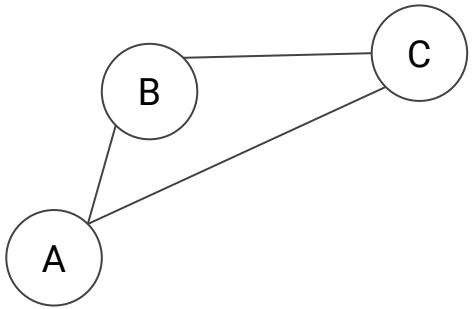
The exact reasoning is a 188 topic, but here's the intuition:

This is the optimistic approach - thinking that the heuristic is a shorter distance than it might actually be. This means you try more options, whereas if the heuristic is a longer distance, you may miss what is actually the shortest path.

Consistent:

h(s, e) <= d(s, n) + h(n, e)

Similar to the triangle inequality, the heuristic between two points needs to be less than or equal to the distance to an intermediate point plus the heuristic from the intermediate point to the end.



Kruskal's

- Central idea: make a sorted list of the edges, then iterate through them. For every edge, if they are not connected by a path to the existing set (the MST set), add it.
- Runtime:
  - You check every edge (at max), so E checks/possible additions
  - Each check for a path takes logE time with a disjoint set implementation (provided weighted quick union)
  - Each check for a path takes V time with a DFS/BFS implementation (look through all the vertices)

Dijkstra's

- There is a fringe of vertices to visit, with a priority queue (priority is based on distance from start vertex). We visit vertices and add them in our search. For every visit, for every neighbor, we skip if it's been visited, if it's not in the fringe we add it (with distance), or we update the distance if there's a shorter distance.
- This relaxing process is what is different than Prim's – in Prim's the distance is just the edge weight; here, it is the distance from the start vertex.
- We make up to E updates and adjust the fringe up to V times, making runtime ElogV + VlogV (generally can be simplified to ElogV directly)