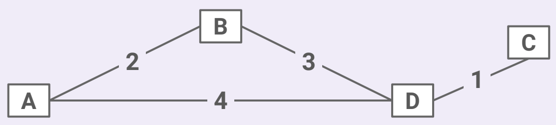
MSTs vs. SPTs

Is the MST for this graph also a shortest paths tree?

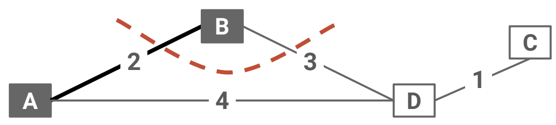
If so, using which node as the starting node for this SPT?



Repeated Application of Cut Property

Given a cut, the minimum-weight crossing edge must be in the minimum spanning tree.

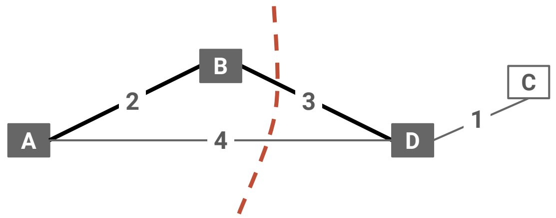
But other crossing edges can also be in the minimum spanning tree.



Repeated Application of Cut Property

Given a cut, the minimum-weight crossing edge must be in the minimum spanning tree.

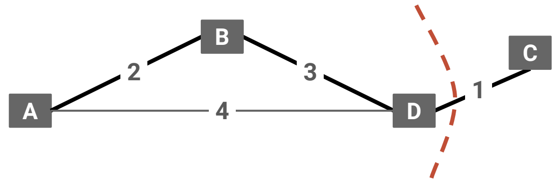
But other crossing edges can also be in the minimum spanning tree.



Repeated Application of Cut Property

Given a cut, the minimum-weight crossing edge must be in the minimum spanning tree.

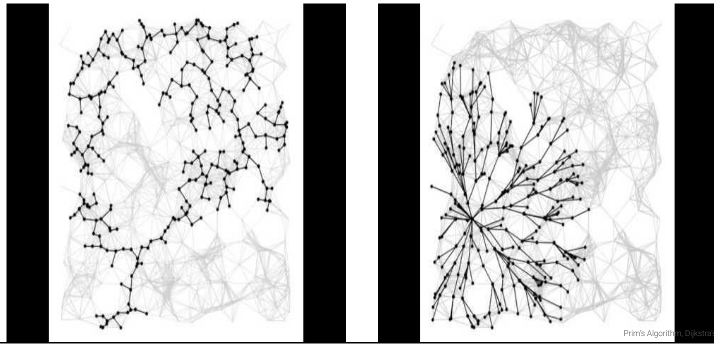
But other crossing edges can also be in the minimum spanning tree.



Conceptual Prim's Algorithm

Demo

Idea. Iteratively apply cut property from a source vertex, expanding the fringe as we go.



Prim's Algorithm, Dijkstra's Algorithm (Kevin Wayne/Princeton)

PQ.add(**s**, 0)

For all other vertices **v**, PQ.add(**v**, infinity)

While PQ is not empty:

  p = PQ.removeSmallest()
  Relax all edges from **p**

Relaxing an edge (**v**, **w**) with **weight**:

If distTo[**w**] > distTo[**v**] + **weight**:

  distTo[**w**] = distTo[**v**] + **weight**
  edgeTo[**w**] = **v**
  PQ.changePriority(**w**, distTo[**w**])

Dijkstra's Pseudocode

Invariants

edgeTo[**v**]: best known predecessor of **v**.

distTo[**v**]: best known distance of **s** to **v**.

PQ maintains vertices based on distTo.

Important properties

Always visits vertices in order of total distance from source. Relaxation always fails on edges to visited (white) vertices.

PQ.add(**s**, 0)

For all other vertices **v**, PQ.add(**v**, infinity)

While PQ is not empty:

  p = PQ.removeSmallest()
  Relax all edges from **p**

Relaxing an edge (**v**, **w**) with **weight**:

If **w** is in PQ and distTo[**w**] > **weight**:

  distTo[**w**] = **weight**
  edgeTo[**w**] = **v**
  PQ.changePriority(**w**, distTo[**w**])

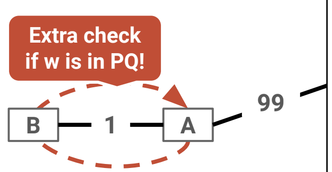
Prim's Pseudocode

Invariants

edgeTo[**v**]: best known predecessor of **v**.

distTo[**v**]: best known distance of **s** to **v**.

PQ maintains vertices based on distTo.



PQ.add(**s**, 0)

For all other vertices **v**, PQ.add(**v**, infinity)

While PQ is not empty:

  p = PQ.removeSmallest()
  Relax all edges from **p**

Relaxing an edge (**v**, **w**) with **weight**:

If **w** is in PQ and distTo[**w**] > **weight**:

  distTo[**w**] = **weight**
  edgeTo[**w**] = **v**
  PQ.changePriority(**w**, distTo[**w**])

Prim's Runtime Analysis

Same as Dijkstra's.

ArrayHeapMinPQ implementation.

- **V** adds, each O(log **V**) time.
- **V** removals, each O(log **V**) time.
- **E** contains, each O(log **V**) time.
- **E** changePriority, each O(log **V**) time.

Simple: O(**V** log **V** + **E** log **V**).

Assuming **E** > **V**, this is just O(**E** log **V**) for connected graphs.

Prim's Algorithm as a Modification of Dijkstra's

Demo

Prim's Algorithm is almost the same as Dijkstra's Algorithm.

Instead of measuring distance from the source, **Prim's considers distance from the tree**.

Visit order:

- Dijkstra's visits vertices in order of distance from the source.
- Prim's visits vertices in order of distance from the MST-under-construction.

Relaxation:

- Dijkstra's considers an edge better based on distance to source.
- Prim's considers an edge better based on distance to tree.

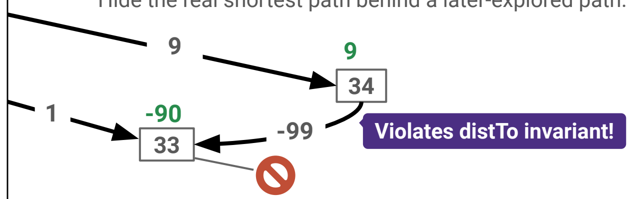
A Dijkstra's Algorithm Correctness

Dijkstra's algorithm. Visit vertices in order of distance from source.

On visit, **relax** every edge from the visited vertex.

Dijkstra's can fail if the graph has negative weight edges. **Give an example graph.**

Hide the real shortest path behind a later-explored path.



When poll is active, respond at **PollEv.com/kevinl**

Does Prim's algorithm work on graphs with negative edge weights?

- Always
- Sometimes
- Never
- Not enough information
- Not sure

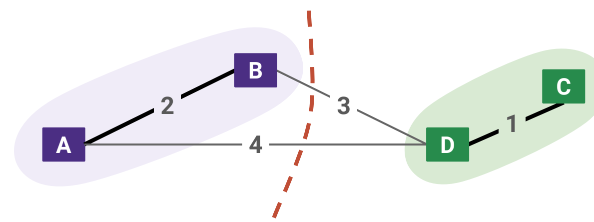
Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

Total Results

Repeated Application of Cut Property

Given a cut, the minimum-weight crossing edge must be in the minimum spanning tree.

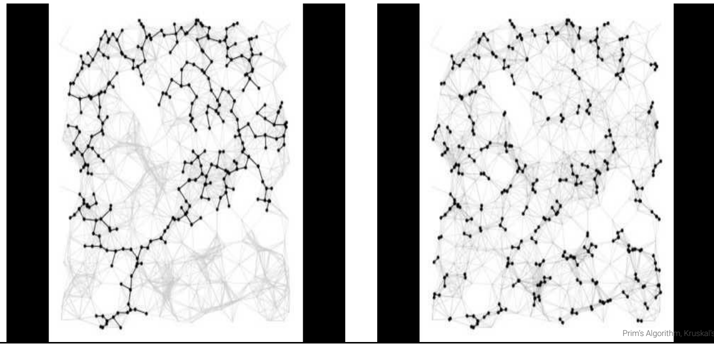
But other crossing edges can also be in the minimum spanning tree.



Conceptual Kruskal's Algorithm

Demo

Idea. Consider edges by increasing weight. Add edge to MST (mark black) unless doing so creates a cycle. Repeat until V-1 edges.



Prim's Algorithm. Kruskal's Algorithm (Kevin Wayne/Princeton)

Sort all edges by weight

While number of edges in MST < **V** - 1:

- **e** = next lightest edge
- If adding **e** doesn't cause a cycle:
  - Add **e** to the MST

Kruskal's Runtime Analysis

Simple graph: **E** < $\mathbf{V}^2$.

- Sorting: O(**E** log **E**) = O(**E** log **V**).
- **E** cycle-checks.
- **V** - 1 edges added to the MST.

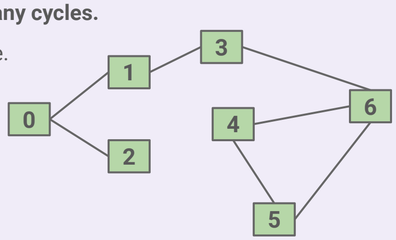
Cycle-finding runtime?

Cycle-finding?

Q Finding Cycles

Given a undirected graph, determine if it contains any cycles.

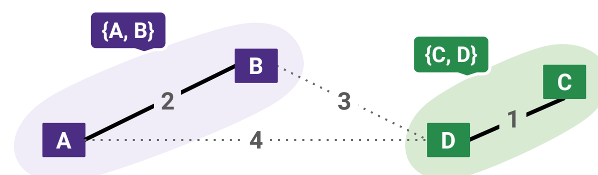
Use any data structure or algorithm from the course.



Finding Cycles: Connected Components

For each vertex **v**, its **connected component** is the set of all vertices that are connected to **v**.

Model connectedness in terms of sets of vertices. Keep track of the component (set) for **v**.




```
isConnected(B, D)
    connect(B, D)
```