Parallel And Sequential Data Structures and Algorithms

Minimum Spanning Trees, Borůvka's Algorithm

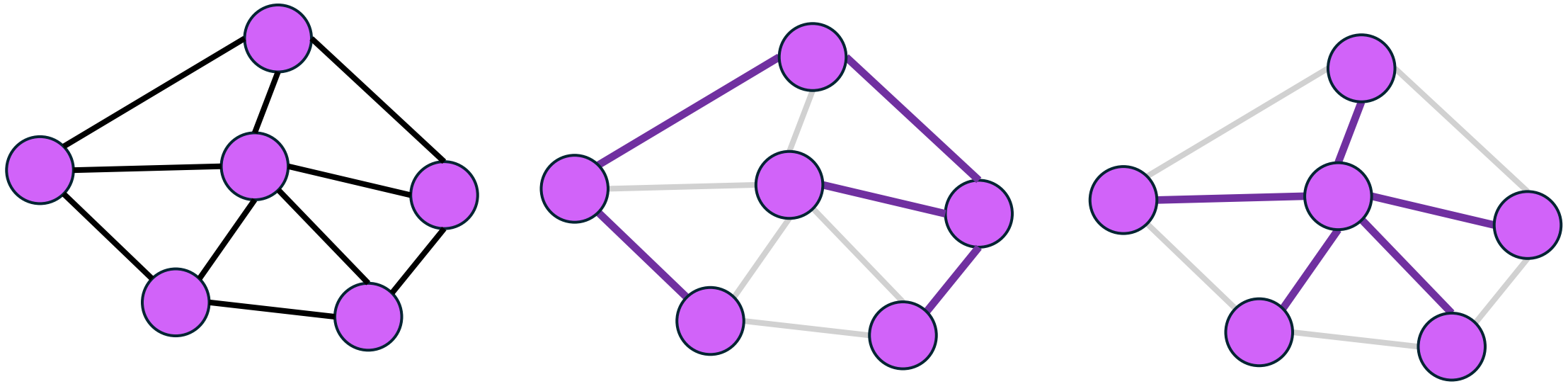
Learning Objectives

- Review the concept of minimum spanning trees (MSTs)
- See and prove some important properties of MSTs that underpin most MST algorithms
- Review Prim's and Kruskal's (sequential) algorithms for MSTs
- Learn **Borůvka's** parallel MST algorithm

Minimum Spanning Trees

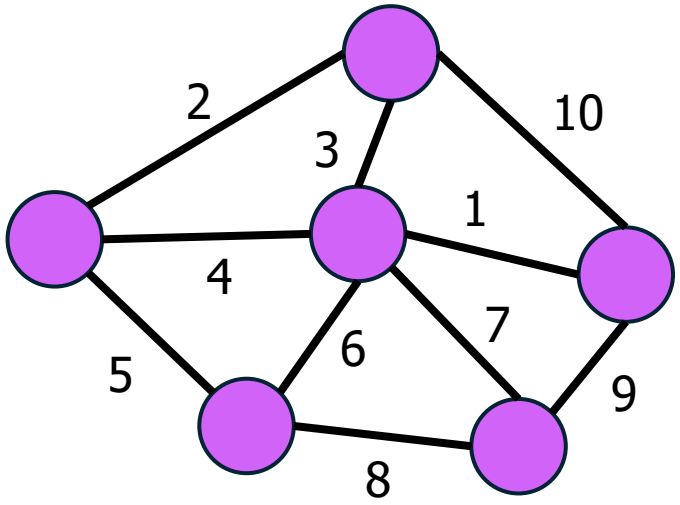
Spanning Trees

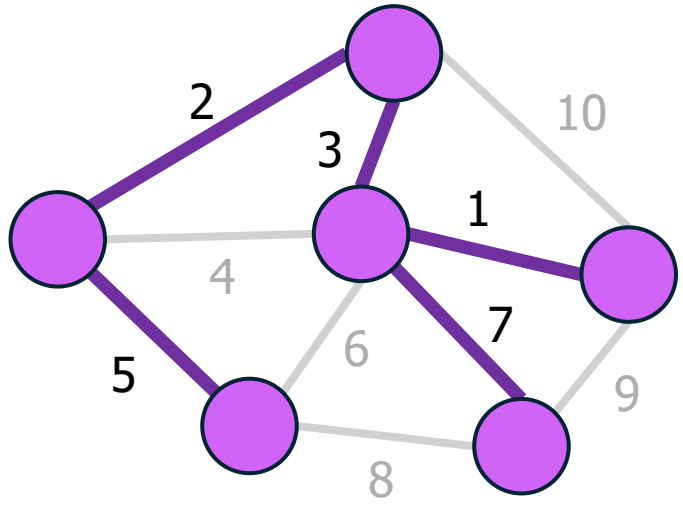
Definition (Spanning Tree): A spanning tree of an undirected, connected graph $G = (V, E)$ is a tree on V, i.e., a connected graph $T = (V, E')$ where $E' \subseteq E$.



Minimum Spanning Tree

Definition (Minimum Spanning Tree): A minimum spanning tree (MST) of a weighted, undirected, connected graph is a spanning tree of minimum possible total weight





Uniqueness of Edge Weights

- MSTs are easier if we assume the edge weights are unique
- But this sounds like a substantial loss of generality!

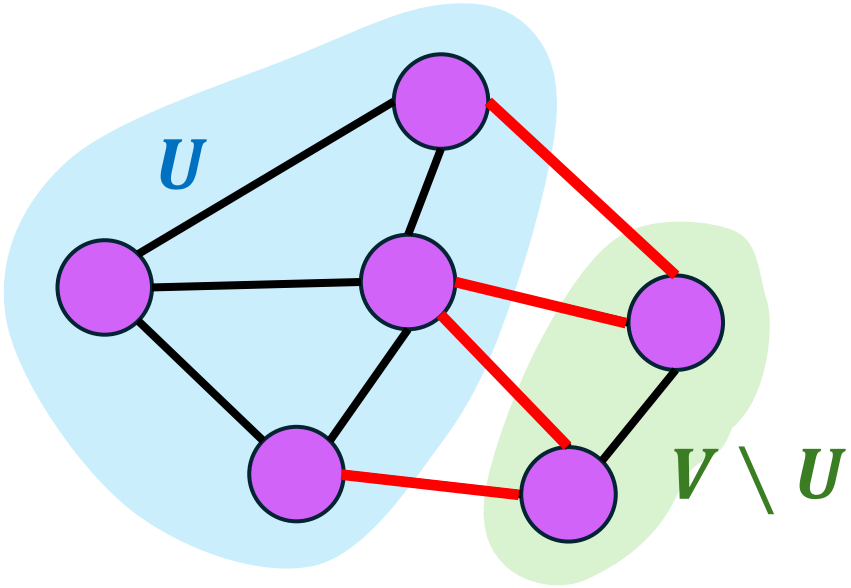
We can assume the edge weights are unique **without loss of generality**

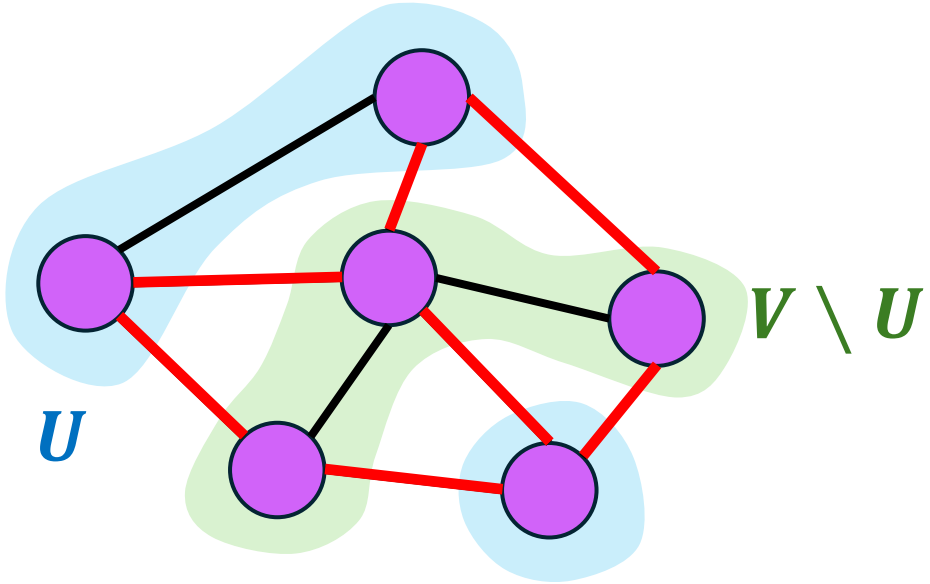
- Just introduce a tiebreaker for edge weights
- For example, give each edge a label from 0 to $m - 1$
- Tiebreak equal weights by considering the label

Theorem (Unique MST): In a weighted undirected graph with unique edge weights, there exists a unique MST

Properties of MSTs

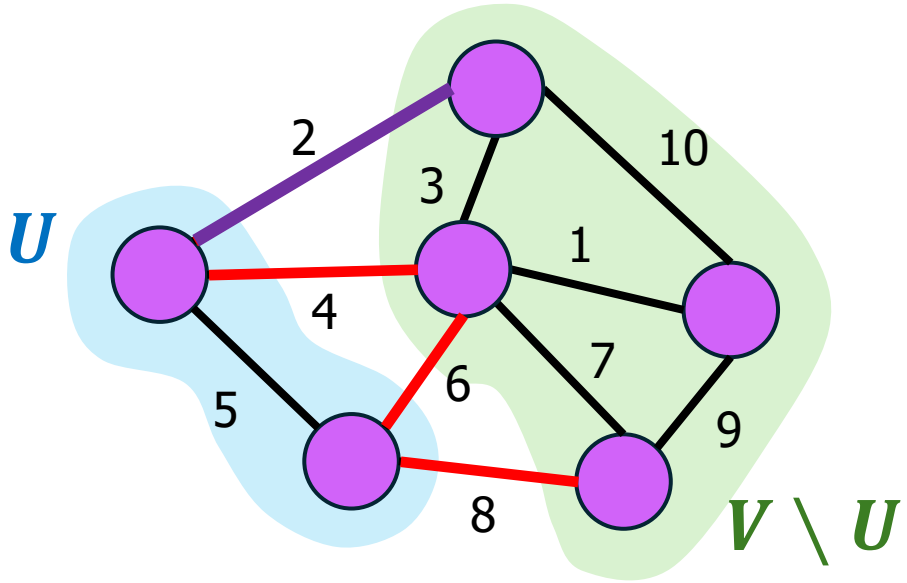
Definition (Cut): In an undirected graph $G = (V, E)$, a **cut** is a partition of the vertices V into two non-empty disjoint subsets U and $V \setminus U$. An edge $(u, v) \in E$ **crosses** the cut if $u \in U$ and $v \notin U$.





Properties of MSTs

Theorem (Light-Edge/Cut/Blue Rule): In a weighted undirected graph $G = (V, E)$, for **any** cut $(U, V \setminus U)$, the lightest edge crossing the cut must be in the minimum spanning tree of G.



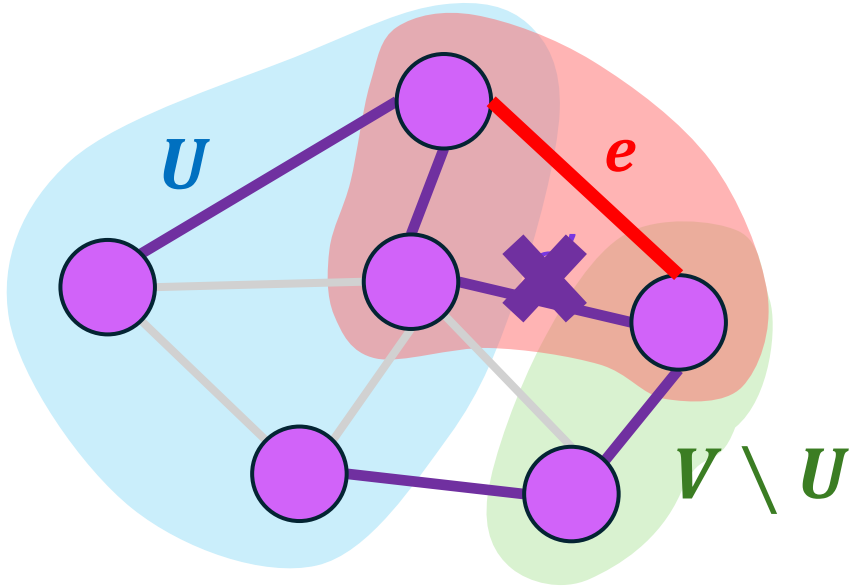
If there are duplicate edge weights, **for any lightest edge**, there exists an MST containing it.

Properties of MSTs

Theorem (Light-Edge/Cut/Blue Rule): In a weighted undirected graph $G = (V, E)$, for **any** cut $(U, V \setminus U)$, the lightest edge crossing the cut must be in the minimum spanning tree of G.

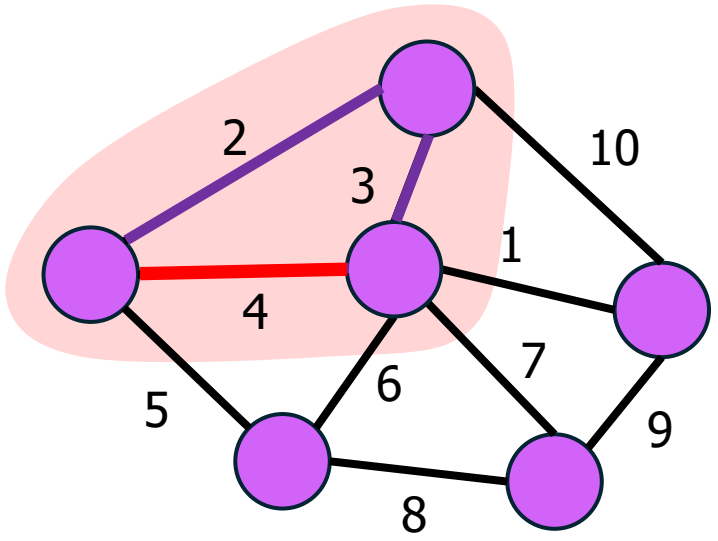
Proof: Let $\boldsymbol{e}$ *be the lightest edge across* $(\boldsymbol{U}, \boldsymbol{V \setminus U})$

- *Let* $\boldsymbol{T}$ *be a spanning tree not containing* $\boldsymbol{e}$
- *Consider the graph* $T \cup \{e\}$*, it contains a cycle!*
- *Some edge* $\boldsymbol{e'}$ *in* T *in this cycle crosses the cut*
- *Replace* e' *with* e *and we still have a spanning tree*
- *But since* $w(e) < w(e')$ *it has a lower weight*
- *Therefore, T is not an MST*



Properties of MSTs

Theorem (Heavy-Edge/Cycle/Red Rule): In a weighted undirected graph $G = (V, E)$, for **any** cycle in G, the heaviest edge on the cycle is not in the MST of G.



If there are duplicate edge weights, **for any heaviest edge**, there exists an MST not containing it

Proof: Very similar to the light-edge rule

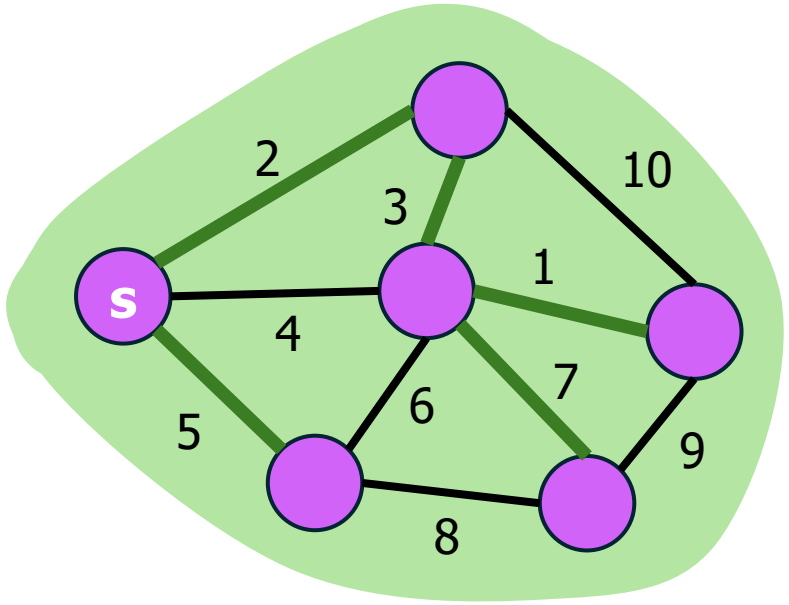
- *Also, you saw it in 15-122*

Sequential Algorithms

Prim's (Jarnik's) Algorithm

Algorithm (Prim's MST Algorithm):

- Start at an arbitrary source vertex s
- Until the tree is fully connected:
  - Add the lightest edge that connects the current tree to a new vertex



Theorem (Correctness): Prim's Algorithm is correct.

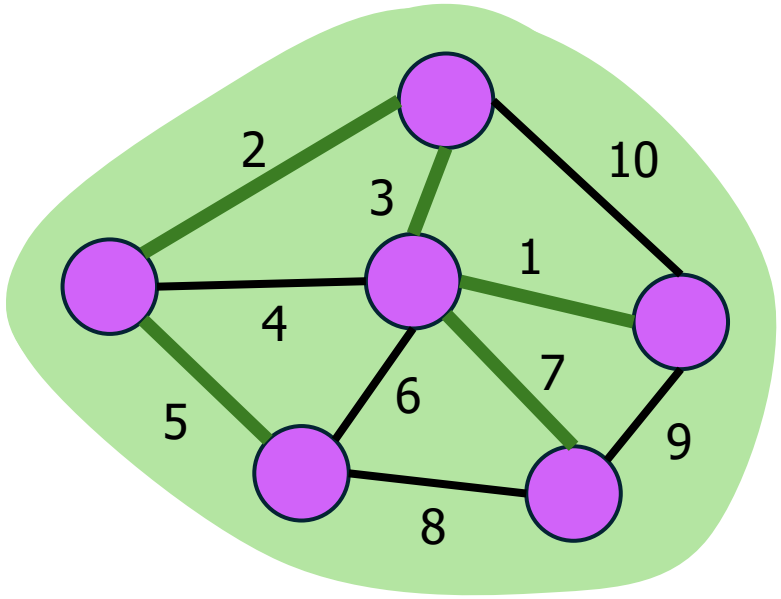
Proof: The light-edge property!

- *Every edge added by Prim's Algorithm is the lightest edge crossing the cut (current tree, everything else)*

Kruskal's Algorithm

Algorithm (Kruskal's MST Algorithm):

- Sort the edges of the graph by weight, lightest to heaviest
- For each edge (u, v) in order, add it to the MST if u and v are not already connected



Theorem (Correctness):
Kruskal's Algorithm is correct.

Proof: The light-edge property again!

- *For any edge added by Kruskal's, can you think of a cut that shows that it must be in the MST?*

Efficient Implementations

- Both Prim's and Kruskal's algorithm require **efficient data structures** to implement with good cost bounds

Theorem (Prim's Cost): Prim's Algorithm runs in $O(m \log n)$ cost.

*Use an efficient **priority queue** to maintain the set of edges adjacent to the current tree and to select the lightest*

Theorem (Kruskal's Cost): Kruskal's Algorithm runs in $O(m \log n)$ cost.

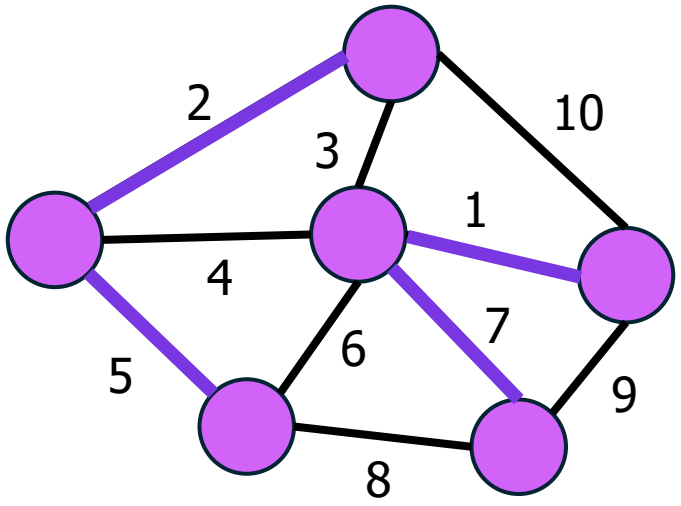
*Sorting the edges costs $O(m \log n)$. Then, use an efficient **union find** data structure to maintain whether u and v are connected in $O(\log n)$ per query.*

Borůvka's Algorithm

A Parallel MST Algorithm

- Both Prim's and Kruskal's algorithms are completely sequential, they add one edge at a time to the tree/forest

Can we **find many edges at once (in parallel)** that are all guaranteed to be a part of the MST?



Claim (Vertex Bridges): In a graph with unique weights, for each vertex, the lightest edge adjacent to it is in the MST

*Proof: The light-edge property **again**!*

- *For each u, consider the cut* $(\{u\}, V \setminus \{u\})$

Borůvka's Algorithm – High Level

- Borůvka finds, in parallel, the "**vertex bridge**" (lightest adjacent edge) for each vertex.
- All these are guaranteed to be MST edges by the light-edge rule

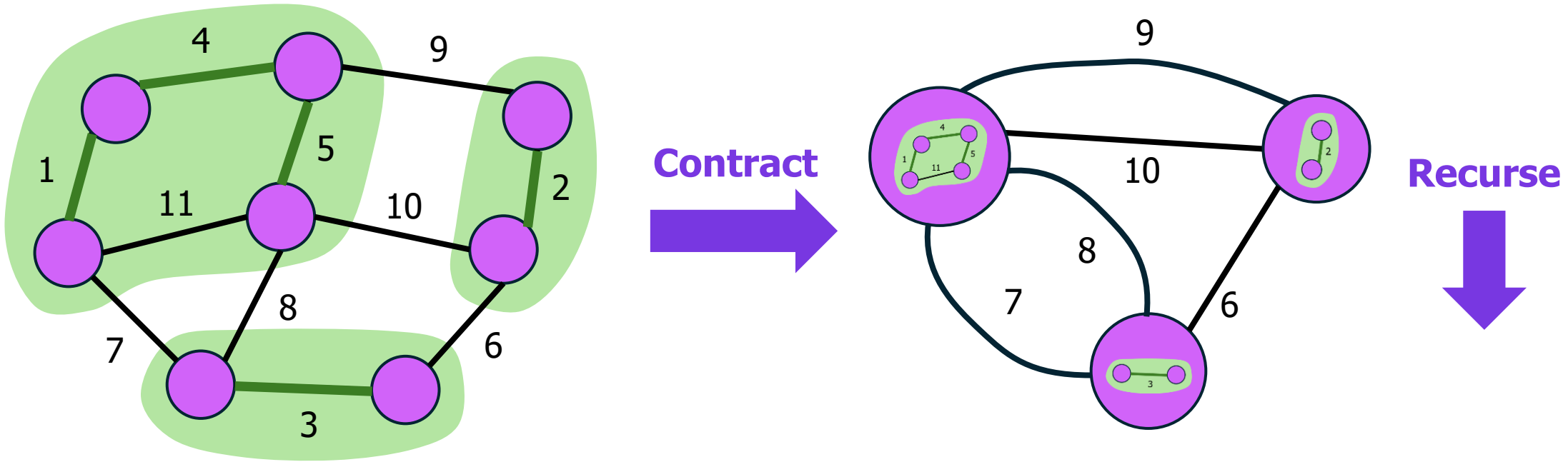
How many vertex bridges will we find?

- At least $n/2$ and at most $n - 1$, so we find **at least half** of the minimum spanning tree

Idea: Find the vertex bridges, then recursively find the rest of the MST

Borůvka's Algorithm – High Level

Question: After finding the vertex bridges, what smaller graph do we recursively find the MST of?



Borůvka's Algorithm – High Level

Algorithm (Borůvka's MST Algorithm):

1. For each vertex in parallel, find its lightest adjacent edge
2. **Contract these edges to form a smaller graph**
3. Recursively find the MST of the contracted graph
4. Return the vertex bridges and the edges of the contracted MST

- Steps 1, 3, & 4 are straightforward.
- Step 1 is just a map over each vertex and a reduce over the neighbors the vertex
- Step 2 can be implemented by finding the **connected components** induced by the vertex bridges

The Contraction Step

Algorithm (Borůvka's Contraction Step):

1. Find the connected components of the graph induced by the vertex bridges (this can be implemented using **star contraction**)
2. Filter the edges whose endpoints are in the same component
3. Relabel the endpoints of the remaining edges with the connected components that they belong to

- Using the **star contraction** algorithm from last lecture, Step 1 costs
  - $O\left(m \log^2(n)\right)$ work in expectation,
  - $O\left(\log^3(n)\right)$ span w.h.p.
- Both other steps cost at most $O(m)$ work and $O\left(\log(n)\right)$ span

The Cost

Theorem (Borůvka's Cost): On a connected undirected weighted graph with n vertices and m edges, Borůvka's algorithm can be implemented in $O(m \log^3(n))$ expected work and $O(\log^4(n))$ span w.h.p.

Proof:

- *The contraction step costs $O(m \log^2(n))$ expected work and $O(\log^3(n))$ span w.h.p.*
- *Contraction reduces the number of vertices to at most $n/2$*
- *Therefore, the algorithm will finish after $O(\log(n))$ rounds*

Almost Efficient, But Not Quite

- $O\left(m \log^3(n)\right)$ expected work and $O\left(\log^4(n)\right)$ span w.h.p. is almost efficient compared to our sequential algorithms
- Prim's and Kruskal's cost $O\left(m \log(n)\right)$ work, so we are over by just a factor of $\log^2(n)$

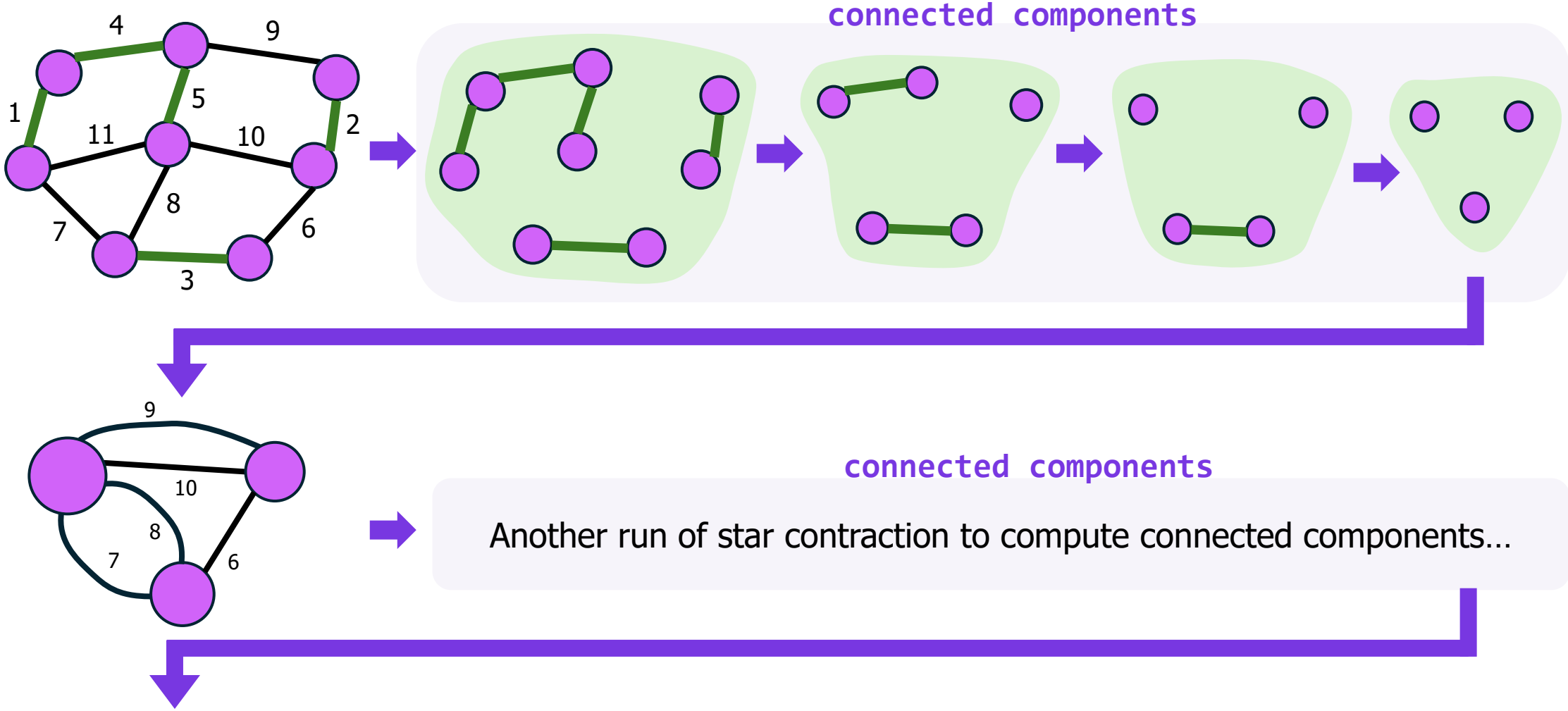
Question: Can we make Borůvka's as efficient as Prim's/Kruskal's?

Where is This Inefficient?

- Borůvka's **is** a graph contraction algorithm: we reduce the number of vertices by half and recurse on a smaller graph
- But… the contraction step of Borůvka's algorithm finds the connected components with respect to the vertex bridges…
- Which is solved using star contraction…

So, Borůvka's algorithm is calling a graph contraction algorithm on every step of a graph contraction algorithm!

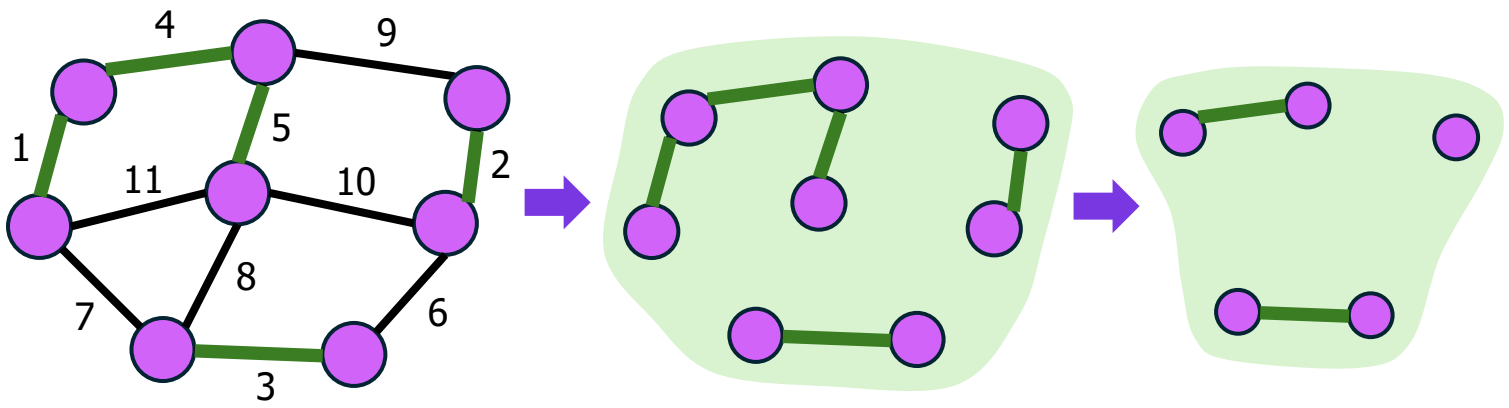
Contraction Within Contraction



So… Less Contraction?

- We use star contraction to compute connected components to contract the original graph and recursively find a smaller MST

the goal is just to **make the graph smaller**. We don't need to make it as small as possible. E.g., reducing the number of vertices to $\frac{3}{4}n$ is good



Instead of running star contraction to completion, just run **one round**.

Improved Algorithm

Algorithm (Improved Borůvka's Contraction Step):

1. Run **star partitioning** on the graph induced by the vertex bridges
2. Vertex bridges that contract become part of the MST
3. Filter the edges whose endpoints are in the same star
4. Relabel the endpoints of the remaining edges with the star centers they belong to

- Using the **star partitioning** algorithm from last lecture, Step 1 costs
  - $O(m \log(n))$ work in expectation,
  - $O(\log^2(n))$ span w.h.p.
- Steps 3 and 4 cost at most $O(m)$ work and $O(\log(n))$ span

The Improved Cost

Theorem (Borůvka's Cost): On a connected undirected weighted graph with n vertices and m edges, Borůvka's algorithm can be implemented in $O(m \log^2(n))$ expected work and $O(\log^3(n))$ span w.h.p.

Proof:

- *The contraction step costs* $O(m \log(n))$ *expected work and* $O(\log^2(n))$ *span w.h.p.*
- *Contraction reduces the number of vertices to at most* $3n/4$ *in expectation (a vertex contracts if it flips T and a neighbour flips H)*
- *Therefore, the algorithm will finish after* $O(\log(n))$ *rounds w.h.p.*

Still Not Quite There

- $O\left(m \log^2(n)\right)$ expected work and $O\left(\log^3(n)\right)$ span w.h.p. is even closer to our sequential algorithms, but still not there…

Theorem (Borůvka's Actual Cost): On a connected undirected weighted graph with n vertices and m edges, Borůvka's algorithm can be implemented in $O\left(m \log(n)\right)$ expected work and $O\left(\log^2(n)\right)$ span w.h.p.

How?

- *Need a better implementation of star partitioning that runs in $O(n + m)$ work and $O(\log n)$ span*
- *This is possible, but we don't have another lecture on graph contraction*

Final Algorithm Summary

Algorithm (Borůvka's MST Algorithm):

1. For each vertex in parallel, find its lightest adjacent edge ("bridges")
2. **Run star partitioning on the vertex bridges**
3. Filter the edges whose endpoints are in the same star
4. Contract the graph by relabeling the endpoints of the remaining edges with the star centers they belong to
5. Recursively find the MST of the contracted graph
6. Return the edges contracted by star partitioning and the edges of the contracted MST

Costs $O(m \log(n))$ expected work and $O(\log^2(n))$ span w.h.p.

Summary

- The minimum spanning tree problem has many interesting and efficient algorithms
- Prim's and Kruskal's Algorithms are sequential and cost $O(m \log n)$ work.
- Borůvka's Algorithm is a parallel algorithm that costs $O(m \log n)$ expected work and $O(\log^2 n)$ span w.h.p. if implemented properly, or $O(m \log^2 n)$ expected work and $O(\log^3 n)$ span w.h.p. if implemented with just what we have learned.