Massively Parallel Computation

Christian Wulff-Nilsen
Algorithmic Techniques for Modern Data Models
DTU

November $21, 2025$

Overview for today

- MPC: Comparison with LOCAL/CONGEST models
- Summing Numbers
- Sorting
- Minimum Spanning Tree

Comparison with LOCAL/CONGEST

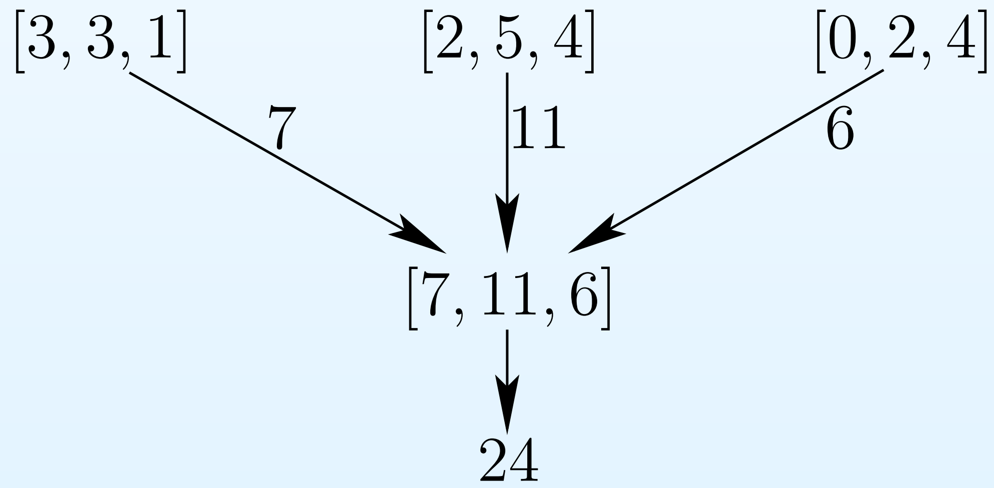
- LOCAL/CONGEST model:
  - Communication network represented by a graph $G = (V, E)$
  - Input is the graph itself
  - Each node outputs its part of the solution
  - No limit on the memory of each machine (node)
  - LOCAL model: No limit on message size
  - CONGEST model: $O(\log n)$ message size, $n = |V|$
- MPC model:
  - Every machine can communicate directly with every other (complete graph)
  - Limit on machine memory, typically much smaller than input
  - Input is arbitrarily distributed among the machines
  - Total message size sent/received by a machine limited by its memory
- For all models: minimize the number of communication rounds

The MPC Model: Parameters

- Input size: N
- Machine memory: S (measured in words)
- Typically $S = N^c$, constant $c < 1$ (S polynomially smaller than N)
- Input arbitrarily distributed among the machines
- Number of machines: M
- We need $M \geq N/S$ (why?)
- Typically, $M = O(N/S)$
- Total size of messages entering or leaving a machine: $\leq S$
- Output: can be stored in a single machine or distributed among multiple machines

Warm-up: Summing Numbers

- Input: list of N numbers
- Output: sum of these numbers
- Assume $S = M = \Theta(\sqrt{N})$
- Solving the problem in 2 rounds:
  - Each machine computes the sum of its numbers
  - It then sends the sum to machine 1
  - Machine 1 computes and outputs the sum of numbers received



Warm-up: Summing Numbers

- Input: list of N numbers
- Output: sum of these numbers
- Assume $S = M = \Theta(\sqrt{N})$
- Solving the problem in 2 rounds:
  - Each machine computes the sum of its numbers
  - It then sends the sum to machine 1
  - Machine 1 computes and outputs the sum of numbers received
- Space and bandwidth bounds are satisfied:
  - Each machine sends only a single word
  - Total size of messages received by machine 1 is $\leq M = S$

Sorting

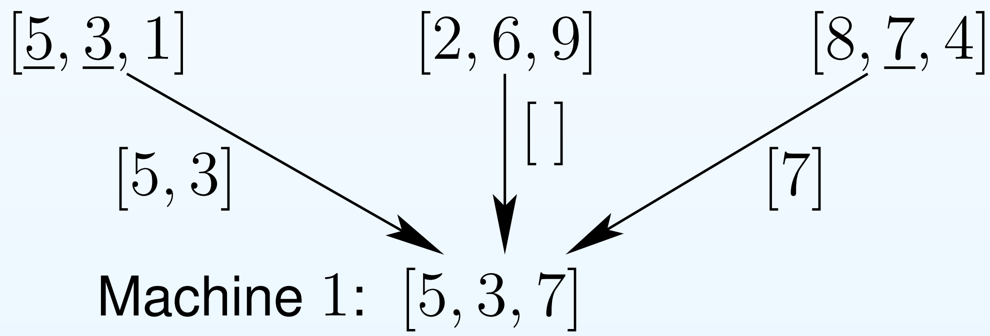
- Input: list of N numbers
- Output: list of all N pairs (x, r) where:
  - x is an element of the list
  - r is the rank of x in the sorted list
- Example:

$$\text{Input: } [7, 2, 9, 5]$$
$$\text{Sorted order: } [2, 5, 7, 9]$$
$$\text{Possible output: } [(7, 3), (2, 1), (9, 4), (5, 2)]$$

- Both input and output are spread among multiple machines
- Space available per machine: $S = N^{\epsilon}$, constant $\epsilon \in (0, 1)$
- Machines available: $M = \Theta(N/S) = \Theta(N^{1-\epsilon})$
- Goal: sort in $O(1)$ rounds
- Our algorithm is a variant of QuickSort for MPC

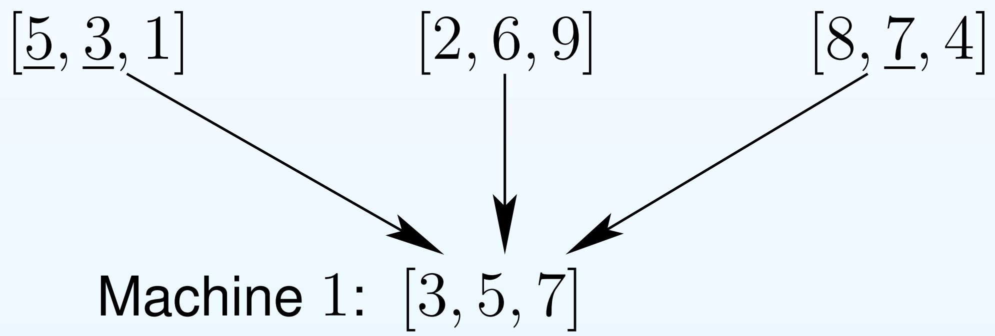
Algorithm for Sorting: Part 1

- Each machine samples each of its elements with some probability p
- Each machine sends its samples to machine 1
- Machine 1 sorts the samples received from all machines
- It then sends the sorted list to all machines



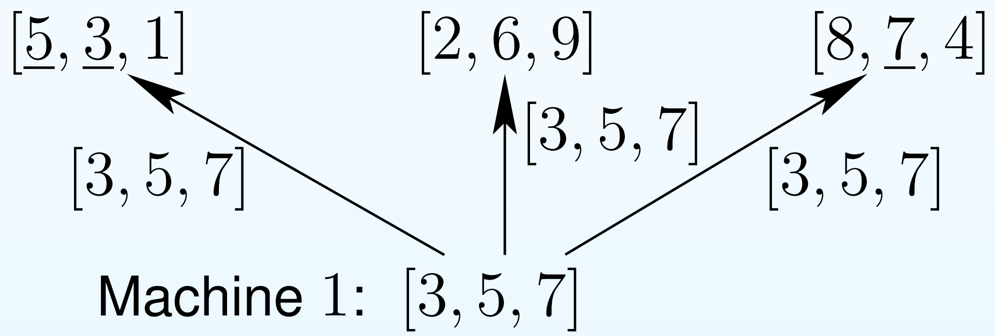
Algorithm for Sorting: Part 1

- Each machine samples each of its elements with some probability p
- Each machine sends its samples to machine 1
- Machine 1 sorts the samples received from all machines
- It then sends the sorted list to all machines



Algorithm for Sorting: Part 1

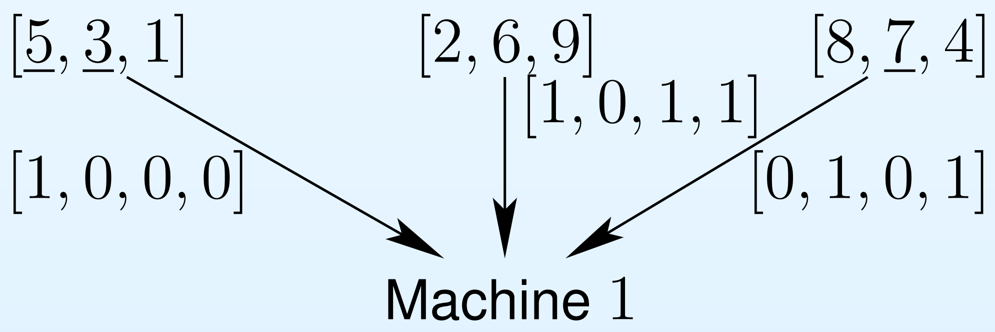
- Each machine samples each of its elements with some probability p
- Each machine sends its samples to machine 1
- Machine 1 sorts the samples received from all machines
- It then sends the sorted list to all machines



Algorithm for Sorting: Part 1

- Each machine samples each of its elements with some probability p
- Each machine sends its samples to machine 1
- Machine 1 sorts the samples received from all machines
- It then sends the sorted list to all machines
- S_i: elements between sample $i-1$ and i in sorted order
- Each machine counts the number of its elements in S_i for each i
- It then sends these counts back to machine 1
- From these counts, machine 1 computes $|S_i|$ for each i

Output: $[\overbrace{1, 2}^{S_1}, \underline{3}, \overbrace{\;4\;}^{S_2}, \underline{5}, \overbrace{\;6\;}^{S_3}, \underline{7}, \overbrace{8, 9}^{S_4}]$



Algorithm for Sorting: Part 1

- Each machine samples each of its elements with some probability p
- Each machine sends its samples to machine 1
- Machine 1 sorts the samples received from all machines
- It then sends the sorted list to all machines
- S_i: elements between sample $i-1$ and i in sorted order
- Each machine counts the number of its elements in S_i for each i
- It then sends these counts back to machine 1
- From these counts, machine 1 computes $|S_i|$ for each i

Output: $[\overbrace{1, 2}^{S_1}, \underline{3}, \overbrace{4}^{S_2}, \underline{5}, \overbrace{6}^{S_3}, \underline{7}, \overbrace{8, 9}^{S_4}]$

$[\underline{5}, \underline{3}, 1]$ $[2, 6, 9]$ $[8, \underline{7}, 4]$

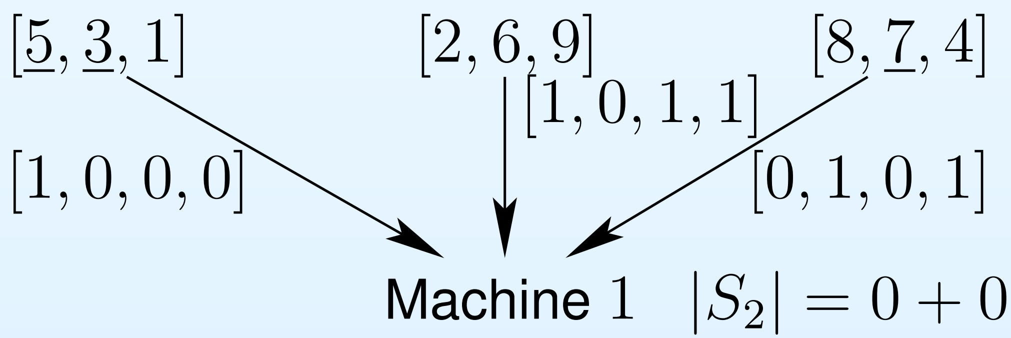
$[1, 0, 0, 0]$ $[1, 0, 1, 1]$ $[0, 1, 0, 1]$

Machine 1 $|S_1| = 1 + 1 + 0 = 2$

Algorithm for Sorting: Part 1

- Each machine samples each of its elements with some probability p
- Each machine sends its samples to machine 1
- Machine 1 sorts the samples received from all machines
- It then sends the sorted list to all machines
- S_i: elements between sample $i-1$ and i in sorted order
- Each machine counts the number of its elements in S_i for each i
- It then sends these counts back to machine 1
- From these counts, machine 1 computes $|S_i|$ for each i

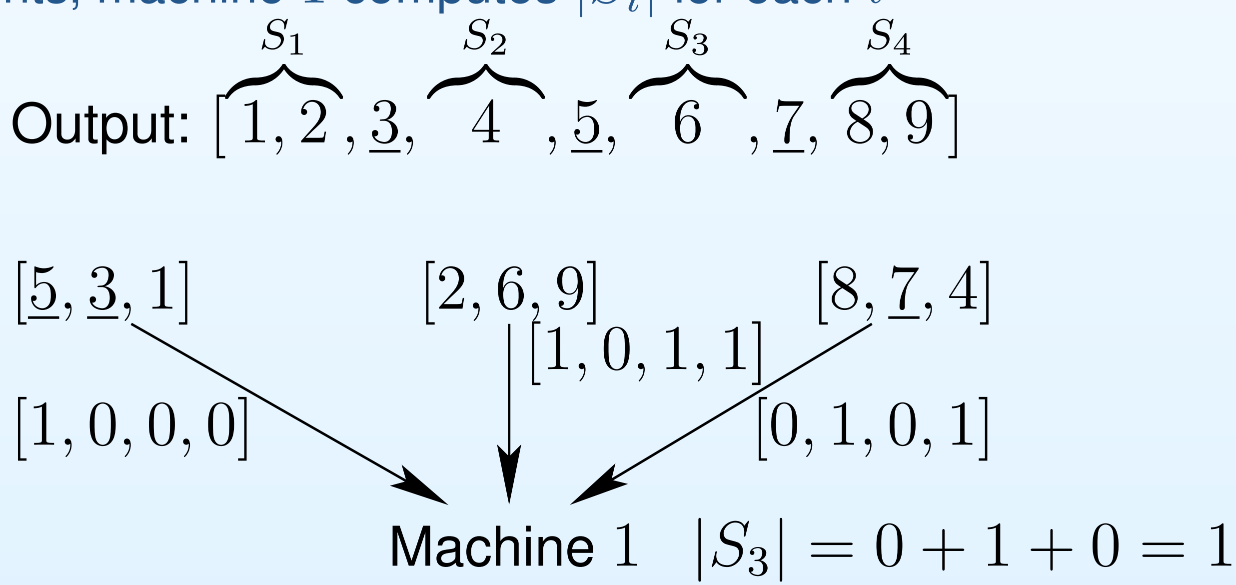
Output: $[\overbrace{1,2}^{S_1},\underline{3},\overbrace{4}^{S_2},\underline{5},\overbrace{6}^{S_3},\underline{7},\overbrace{8,9}^{S_4}]$



$|S_2| = 0 + 0 + 1 = 1$

Algorithm for Sorting: Part 1

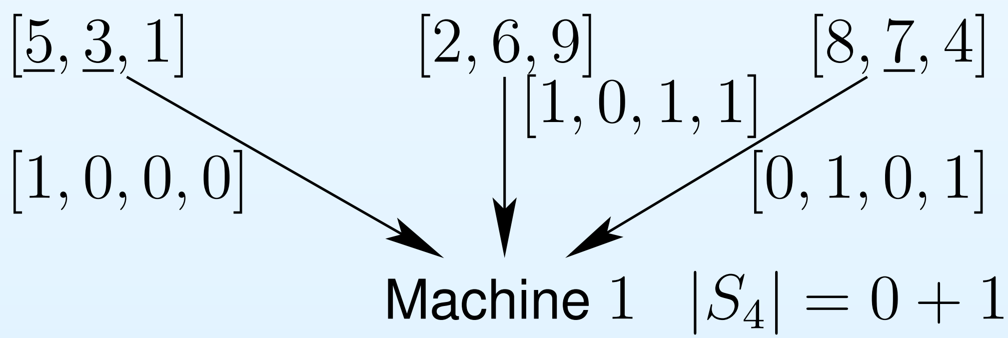
- Each machine samples each of its elements with some probability p
- Each machine sends its samples to machine 1
- Machine 1 sorts the samples received from all machines
- It then sends the sorted list to all machines
- S_i: elements between sample $i-1$ and i in sorted order
- Each machine counts the number of its elements in S_i for each i
- It then sends these counts back to machine 1
- From these counts, machine 1 computes $|S_i|$ for each i



Algorithm for Sorting: Part 1

- Each machine samples each of its elements with some probability p
- Each machine sends its samples to machine 1
- Machine 1 sorts the samples received from all machines
- It then sends the sorted list to all machines
- S_i: elements between sample $i-1$ and i in sorted order
- Each machine counts the number of its elements in S_i for each i
- It then sends these counts back to machine 1
- From these counts, machine 1 computes $|S_i|$ for each i

Output: $[\overbrace{1,2}^{S_1}, \underline{3}, \overbrace{4}^{S_2}, \underline{5}, \overbrace{6}^{S_3}, \underline{7}, \overbrace{8,9}^{S_4}]$

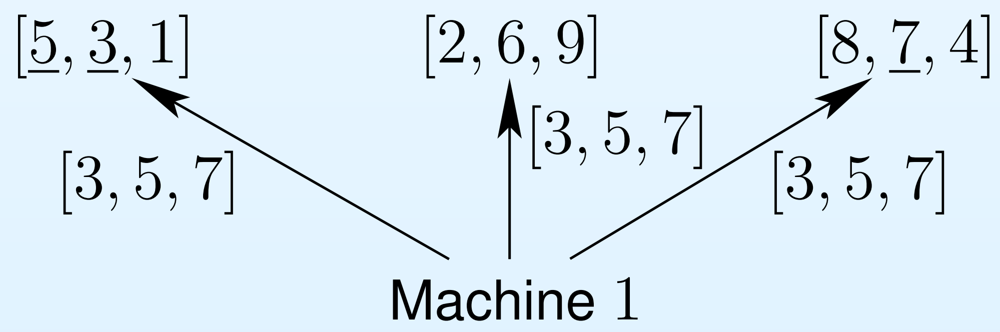


$|S_4| = 0 + 1 + 1 = 2$

Algorithm for Sorting: Part 1

- Each machine samples each of its elements with some probability p
- Each machine sends its samples to machine 1
- Machine 1 sorts the samples received from all machines
- It then sends the sorted list to all machines
- S_i: elements between sample $i-1$ and i in sorted order
- Each machine counts the number of its elements in S_i for each i
- It then sends these counts back to machine 1
- From these counts, machine 1 computes $|S_i|$ for each i
- From sizes $|S_i|$, machine 1 computes the rank of samples and sends them to all machines

Output: $[\overbrace{1,2}^{S_1}, \underline{3}, \overbrace{4}^{S_2}, \underline{5}, \overbrace{6}^{S_3}, \underline{7}, \overbrace{8,9}^{S_4}]$



Algorithm for Sorting: What Remains

- Remaining problem: compute the ranks of non-sampled elements
- Recall: each machine knows the rank of each sample
- Hence, if a machine can find the local rank in S_i of an element x, it can obtain the global rank of x
- This means that the S_i-sets can be sorted recursively
- Problem: S_i may be too large to fit on one machine
- Solution: assign a number of machines proportional to $|S_i|$

Algorithm for Sorting: Part 2

- Machine 1 computed $|S_1|, |S_2|, \ldots$ in Part 1
- It uses this to partition $[M]$ into sub-intervals where sub-interval $[l_i, u_i]$ contains (indices of) machines that should sort S_i
- Example with $M = 10$:

$$|S_1| = 5 \quad |S_2| = 25 \quad |S_3| = 20$$
$$[l_1, u_1] = [1, 1]$$
$$[l_2, u_2] = [2, 6]$$
$$[l_3, u_3] = [7, 10]$$

Algorithm for Sorting: Part 2

- Machine 1 computed $|S_1|, |S_2|, \ldots$ in Part 1
- It uses this to partition $[M]$ into sub-intervals where sub-interval $[l_i, u_i]$ contains (indices of) machines that should sort S_i
- Machine 1 sends all l_i- and u_i-indices to all machines
- Each machine then does the following for each of its elements x:
  - Let $S_i \ni x$
  - Send x to a random machine in $[l_i, u_i]$
- Together, machines in $[l_i, u_i]$ recursively sort S_i
- The recursion stops when $|S_i| \leq S$; a single machine then sorts S_i

Bounding Size Received by a Machine: Part 1

- Max size allowed: $S = N^{\epsilon}$
- Expected number of samples received by machine: $\leq Np$
- Pick $p = N^{\epsilon/2}/2N$
- Chernoff bound: w.h.p., the actual size received is at most:

$$2Np \leq N^{\epsilon/2} \ll N^{\epsilon} = S$$

- W.h.p., machine 1 receives at most $1 + N^{\epsilon/2} = O(N^{\epsilon/2})$ counts from *each* machine
- This is $O(MN^{\epsilon/2}) = O(N^{1-\epsilon/2}) = O(N)$ in total
- Too much in one round!
- We deal with this by spreading the data transfer over $O(1/\epsilon)$ rounds using a *converge-cast tree* (exercise)

Bounding Size Received by a Machine: Part 2

- Expected number of pairs (l_i, u_i) received: $Np + 1$
- W.h.p., the actual number is at most $2(Np + 1) \ll S$ (Chernoff)
- For some x, $|S_i| = \Theta(x(u_i - l_i))$ for all i (proportionality)
- Elements of S_i are randomly assigned to machines in $[l_i, u_i]$
- Thus, each machine in $[l_i, u_i]$ receives $|S_i|/(u_i - l_i) = \Theta(x)$ elements in expectation
- Bounding x:

$$N \geq \sum_i |S_i| = \Theta(\sum_i x(u_i - l_i)) = \Theta(xM) \Rightarrow x = O(N/M)$$

- W.h.p., each machine receives $O(N/M)$ elements
- We assumed $M = CN/S$ for constant C
- For sufficiently big C, no machine receives more than S elements w.h.p.

Bounding Size Sent by a Machine

- Part 1:
  - Machine 1 sends (sorted) samples to all machines
  - In expectation, the total size sent is NpM
  - W.h.p., the actual size sent is at most

$$2NpM = 2N \cdot \frac{N^{\epsilon/2}}{2N} \cdot CN^{1-\epsilon} = O(N^{1-\epsilon/2}) = O(N)$$

  - This is too much in a single round as we only allow $S = N^{\epsilon}$
  - Solution: spread the data transfer over $O(1/\epsilon)$ rounds using a broadcast tree (exercise)
  - W.h.p., less than size S is sent per round per machine
- Part 2: same approach

Bounding Number of Rounds

- Part 1: dominated by $O(1/\epsilon)$ rounds for converge-cast and broadcast tree
- Part 2:
  - $O(1/\epsilon)$ rounds per recursion level
  - We will show that w.h.p., each $|S_i| = O(N^{1-\epsilon/3})$
  - Hence, the problem size is reduced by $\Omega(N^{\epsilon/3})$ per level
  - This implies that the total number of levels is $O(1/\epsilon)$
- Total number of rounds for algorithm: $O(1/\epsilon) \cdot O(1/\epsilon) = O(1)$ (w.h.p.)

Showing $|S_i| = O(N^{1-\epsilon/3})$ W.h.p.

- S_i: elements between sample $i-1$ and i in sorted order

$$
\begin{aligned}
\mathrm{P}[|S_i| \geq k] &= \overbrace{(1-p)^k}^{k \text{ unsampled in a row}} \\
&= \left(1 - \frac{N^{\epsilon/2}}{2N}\right)^k \\
&= \left(1 - \frac{1}{2N^{1-\epsilon/2}}\right)^k \\
&\leq \exp\left(-\frac{k}{2N^{1-\epsilon/2}}\right) \qquad\qquad (1 + x \leq e^x)
\end{aligned}
$$

- With $k = D \cdot 2N^{1-\epsilon/2} \ln N$ for constant D,

$$\mathrm{P}[|S_i| \geq k] \leq e^{-D \ln N} = N^{-D}$$

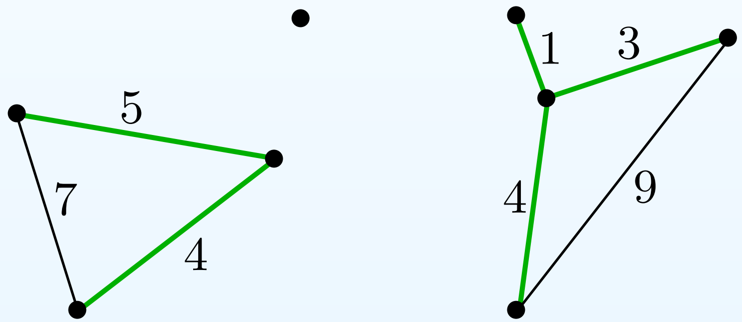
- Conclusion: w.h.p., $|S_i| = O(N^{1-\epsilon/2} \ln N) = O(N^{1-\epsilon/3})$

Minimum Spanning Tree (MST)

- $G = (V, E)$ connected, edge-weighted, undirected graph
- Let $\mathrm{MST}(E)$ denote an MST of E (same as MST of G)
- $n = |V|$, $m = |E|$
- Input: list of edges with weights ($N = m$)
- Output: $\mathrm{MST}(E)$
- $S = n^{1+\epsilon}$ for constant $\epsilon > 0$
- $M = \Theta(N/S) = \Theta(m/n^{1+\epsilon})$ (assume $m = \Omega(n^{1+\epsilon})$)
- Goal: compute $\mathrm{MST}(E)$ in $O(1)$ rounds

Minimum Spanning Forest (MSF)

- *Minimum spanning forest* of a graph $G = (V, E)$:
  - Has a minimum spanning tree for each connected component
  - Denoted by $\mathrm{MSF}(E)$
  - An MSF (green) with three connected components:



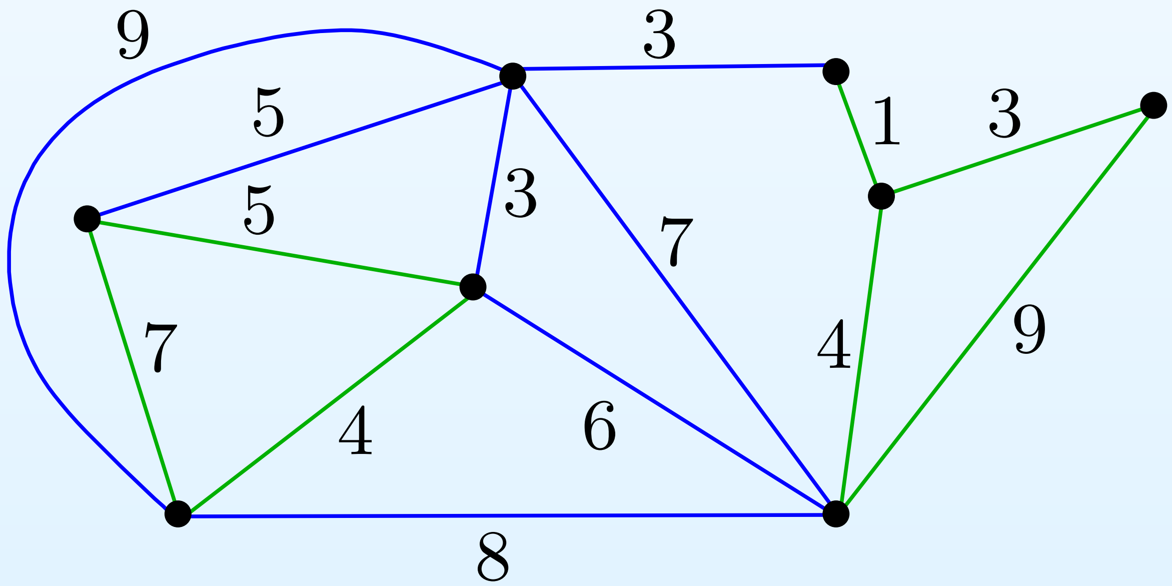
- For simplicity, assume that any graph we consider has a unique MSF
- This is without loss of generality (why?)

Useful Property of MSFs

- Let $E_1, \ldots, E_k$ be a partition of E
- Let $M_i = \mathrm{MSF}(E_i)$ for $i = 1, \ldots, k$
- Then

$$\mathrm{MSF}(E) = \mathrm{MSF}\left(\bigcup_{i=1}^{k} E(M_i)\right)$$

- Example with blue set E_1 and green set E_2:

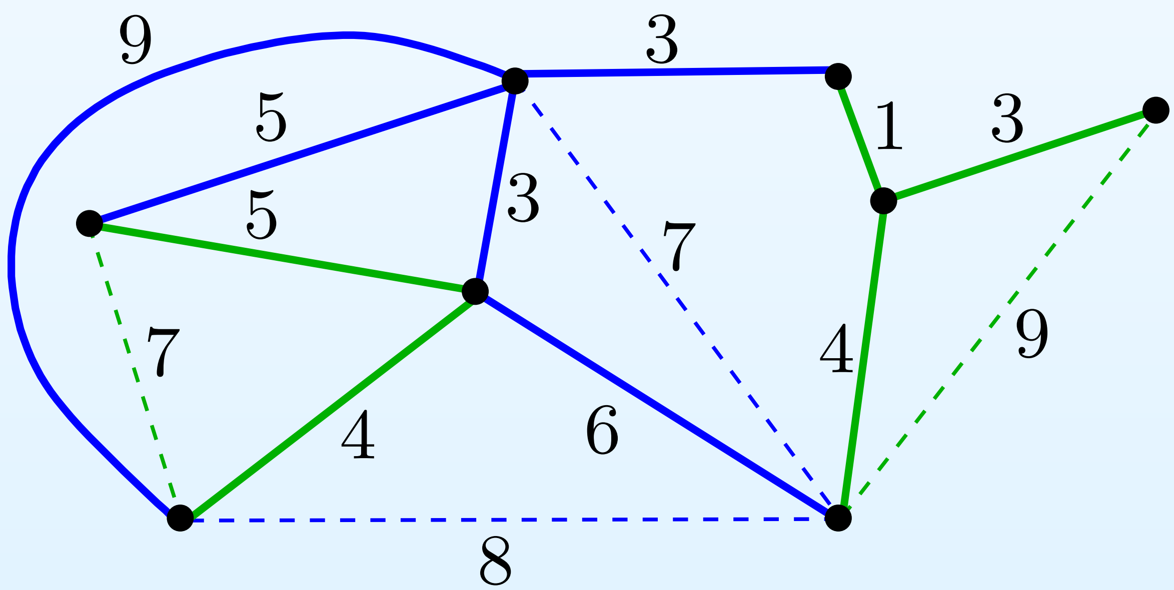


Useful Property of MSFs

- Let $E_1, \ldots, E_k$ be a partition of E
- Let $M_i = \mathrm{MSF}(E_i)$ for $i = 1, \ldots, k$
- Then

$$\mathrm{MSF}(E) = \mathrm{MSF}\left(\bigcup_{i=1}^{k} E(M_i)\right)$$

- M_1 and M_2 (solid edges):

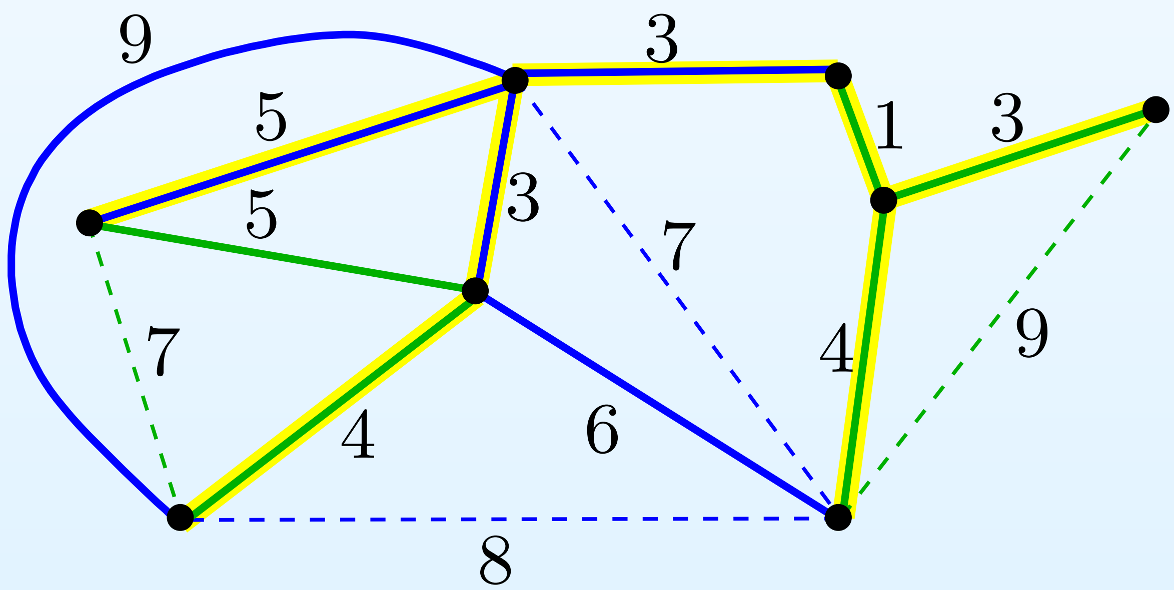


Useful Property of MSFs

- Let $E_1, \ldots, E_k$ be a partition of E
- Let $M_i = \mathrm{MSF}(E_i)$ for $i = 1, \ldots, k$
- Then

$$\mathrm{MSF}(E) = \mathrm{MSF}\left(\bigcup_{i=1}^{k} E(M_i)\right)$$

- $\mathrm{MSF}(E)$ in yellow is contained in $E(M_1) \cup E(M_2)$:



Shuffle/Filter Algorithm for $\mathrm{MST}(G)$

- Shuffle Algorithm for edge set E':
  - Pick $k = 2|E'|/n^{1+\epsilon}$ *active* machines
  - Distribute the edges of E' randomly among these
- Filter Algorithm for each active machine:
  - Compute a minimum spanning forest of assigned edge set
- Overall MST Algorithm with initial $E' = E$:
  - Run Shuffle Algorithm for edge set E'
  - Run Filter Algorithm; let $M_1, \ldots, M_k$ be its output
  - If $k = 1$ then output M_1 and terminate
  - Otherwise, recurse on edge set $\bigcup_{i=1}^{k} E(M_i)$
- By our MSF property, $\mathrm{MSF}(E') = \mathrm{MSF}(E)$
- Hence, if the algorithm terminates, it outputs

$$M_1 = \mathrm{MSF}(E') = \mathrm{MSF}(E) = \mathrm{MST}(G)$$

Showing Termination in $O(1/\epsilon)$ Rounds

- Recall that $k = 2|E'|/n^{1+\epsilon}$ is the number of active machines
- Let $k_1, k_2, \ldots$ be the sequence of k-values in the recursion
- Let $E'_1, E'_2, \ldots$ be the sequence of E'-sets in the recursion
- For $j > 1$, E'_j is the union of k_{j-1} forests so

$$|E'_j| \leq k_{j-1}(n-1)$$

Showing Termination in $O(1/\epsilon)$ Rounds

- Have shown $|E'_j| \le k_{j-1}(n-1)$ for each $j > 1$
- Hence,

$$k_j = \frac{2|E'_j|}{n^{1+\epsilon}} \le \frac{2k_{j-1}(n-1)}{n^{1+\epsilon}} < \frac{2}{n^{\epsilon}} k_{j-1}$$

- Also:

$$k_1 = \frac{2|E|}{n^{1+\epsilon}} \le \frac{2n^2}{n^{1+\epsilon}} = 2n^{1-\epsilon}$$

- Then

$$k_j < 2n^{1-\epsilon} \left(\frac{2}{n^{\epsilon}}\right)^{j-1} = n^{1-\epsilon j} 2^j$$

- Thus, termination after $O(1/\epsilon) = O(1)$ rounds

Bounding Space and Number of Machines

- In each round, $k = 2|E'|/n^{1+\epsilon} = 2|E'|/S$ active machines
- The expected number of edges assigned to each such machine is

$$\frac{|E'|}{k} = \frac{|E'|}{2|E'|/S} = \frac{1}{2}|S|$$

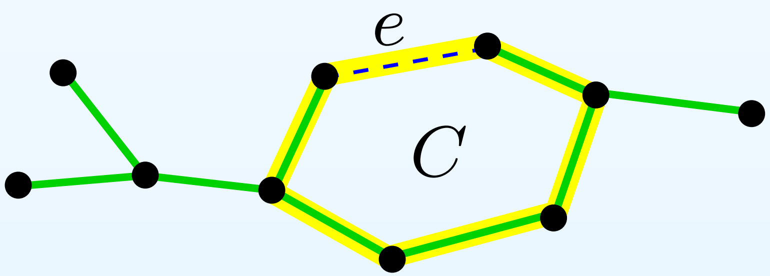
- With high probability, the actual number of edges assigned is at most twice its expectation (Chernoff bound)
- An active machine does not send more edges than it can store
- Thus, no machine exceeds S space with high probability
- Number of machines used: $O(|E|/S) = O(N/S)$

Proving $\mathrm{MSF}(E) = \mathrm{MSF}(\cup_{i=1}^{k} E(M_i))$

- Let $E' = \cup_{i=1}^{k} E(M_i)$

- Will show that for any $e = (u, v) \in \overbrace{E \setminus E'}^{\text{dashed edges}}$, $e \notin \mathrm{MSF}(E)$
- For some i, we have $e \in E_i \setminus E(M_i)$
- This implies that $E(M_i) \cup \{e\}$ has a cycle C containing e

M_i :

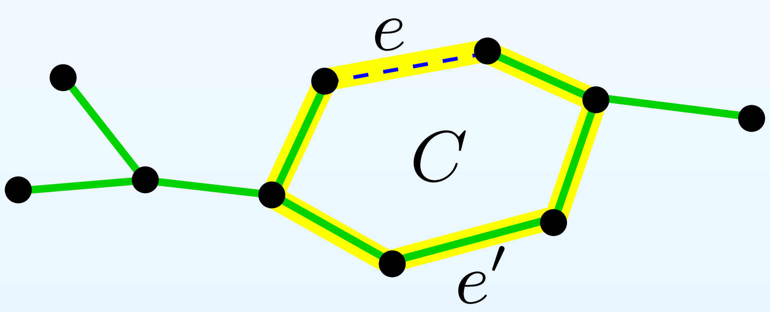


Proving $\mathrm{MSF}(E) = \mathrm{MSF}(\cup_{i=1}^{k} E(M_i))$

- Let $E' = \cup_{i=1}^{k} E(M_i)$

- Will show that for any $e = (u, v) \in \overbrace{E \setminus E'}^{\text{dashed edges}}$, $e \notin \mathrm{MSF}(E)$
- For some i, we have $e \in E_i \setminus E(M_i)$
- This implies that $E(M_i) \cup \{e\}$ has a cycle C containing e

M_i :

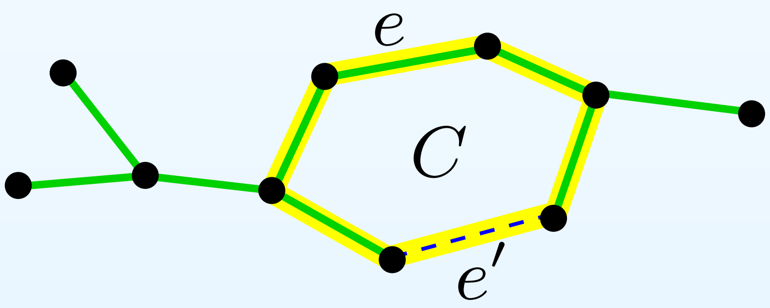


- e has maximal weight on C:
  - Otherwise, let $e' \in E(C)$ with $w(e') > w(e)$
  - Let $M_i' = (M_i \setminus \{e'\}) \cup \{e\}$
  - M_i' is an MSF of E_i of weight less than M_i, a contradiction

Proving $\mathrm{MSF}(E) = \mathrm{MSF}(\cup_{i=1}^{k} E(M_i))$

- Let $E' = \cup_{i=1}^{k} E(M_i)$
- Will show that for any $e = (u, v) \in \overbrace{E \setminus E'}^{\text{dashed edges}}$, $e \notin \mathrm{MSF}(E)$
- For some i, we have $e \in E_i \setminus E(M_i)$
- This implies that $E(M_i) \cup \{e\}$ has a cycle C containing e

M_i' :

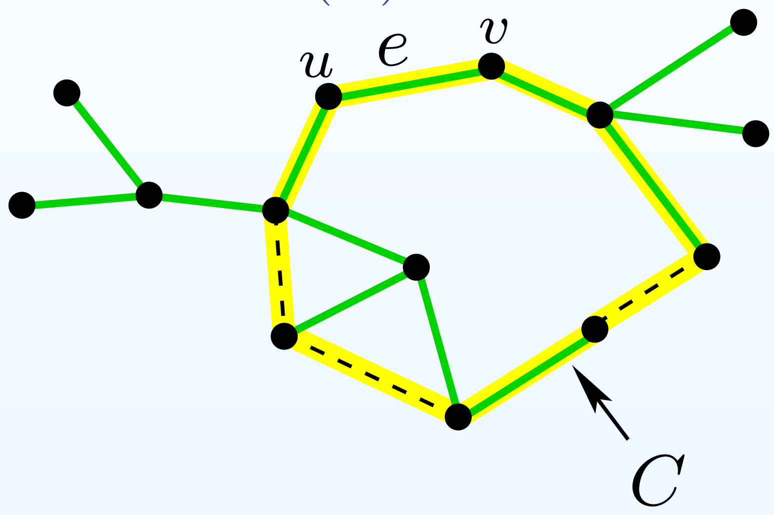


- e has maximal weight on C:
  - Otherwise, let $e' \in E(C)$ with $w(e') > w(e)$
  - Let $M_i' = (M_i \setminus \{e'\}) \cup \{e\}$
  - M_i' is an MSF of E_i of weight less than M_i, a contradiction

Proving $\mathrm{MSF}(E) = \mathrm{MSF}(\cup_{i=1}^{k} E(M_i))$

- Have shown: e has maximal weight on cycle C
- Assume for contradiction that $e \in \mathrm{MSF}(E)$

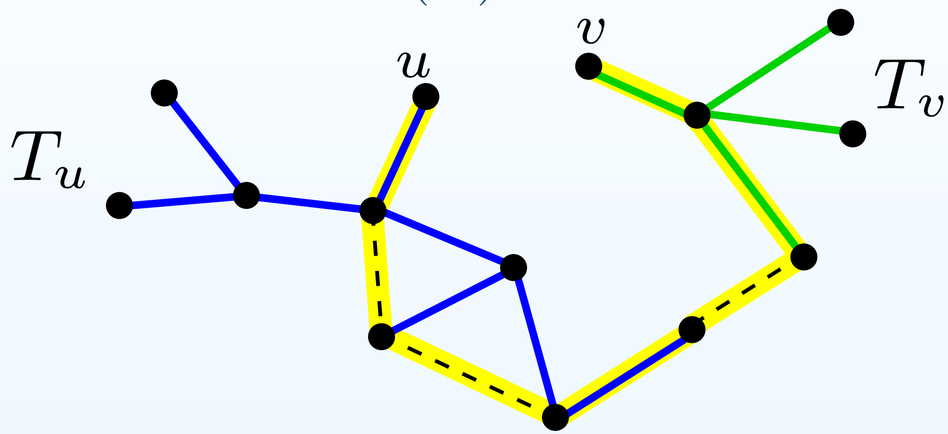
$\mathrm{MSF}(E)$



- Removing e from $\mathrm{MSF}(E)$ splits a tree into two trees: T_u containing u and T_v containing v

Proving $\mathrm{MSF}(E) = \mathrm{MSF}(\cup_{i=1}^{k} E(M_i))$

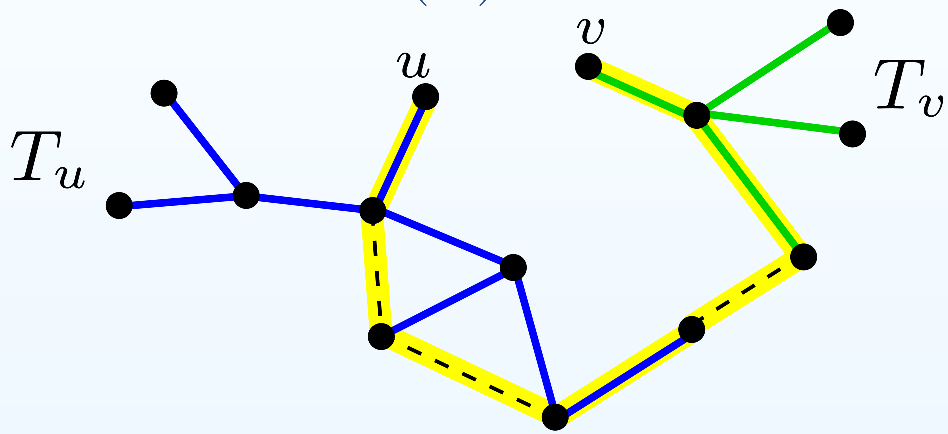
- Have shown: e has maximal weight on cycle C
- Assume for contradiction that $e \in \mathrm{MSF}(E)$



- Removing e from $\mathrm{MSF}(E)$ splits a tree into two trees: T_u containing u and T_v containing v

Proving $\mathrm{MSF}(E) = \mathrm{MSF}(\cup_{i=1}^{k} E(M_i))$

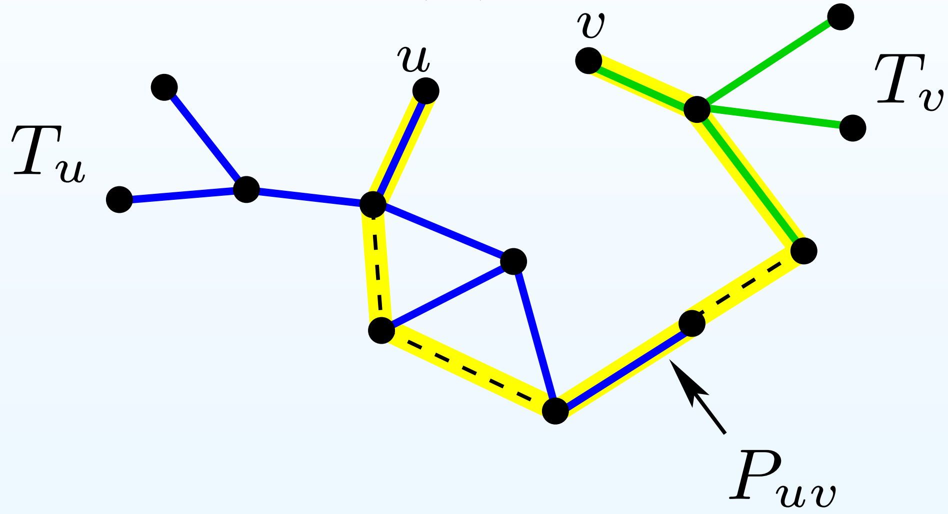
- Have shown: e has maximal weight on cycle C
- Assume for contradiction that $e \in \mathrm{MSF}(E)$



- Removing e from $\mathrm{MSF}(E)$ splits a tree into two trees: T_u containing u and T_v containing v
- $C \setminus \{e\}$ is a path P_{uv} in G from u to v

Proving $\mathrm{MSF}(E) = \mathrm{MSF}(\cup_{i=1}^{k} E(M_i))$

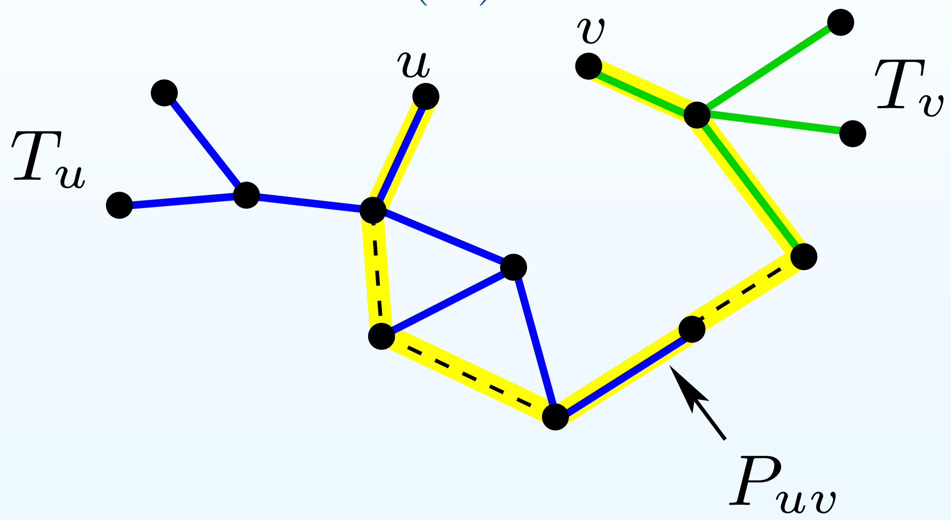
- Have shown: e has maximal weight on cycle C
- Assume for contradiction that $e \in \mathrm{MSF}(E)$



- Removing e from $\mathrm{MSF}(E)$ splits a tree into two trees: T_u containing u and T_v containing v
- $C \setminus \{e\}$ is a path P_{uv} in G from u to v

Proving $\mathrm{MSF}(E) = \mathrm{MSF}(\cup_{i=1}^{k} E(M_i))$

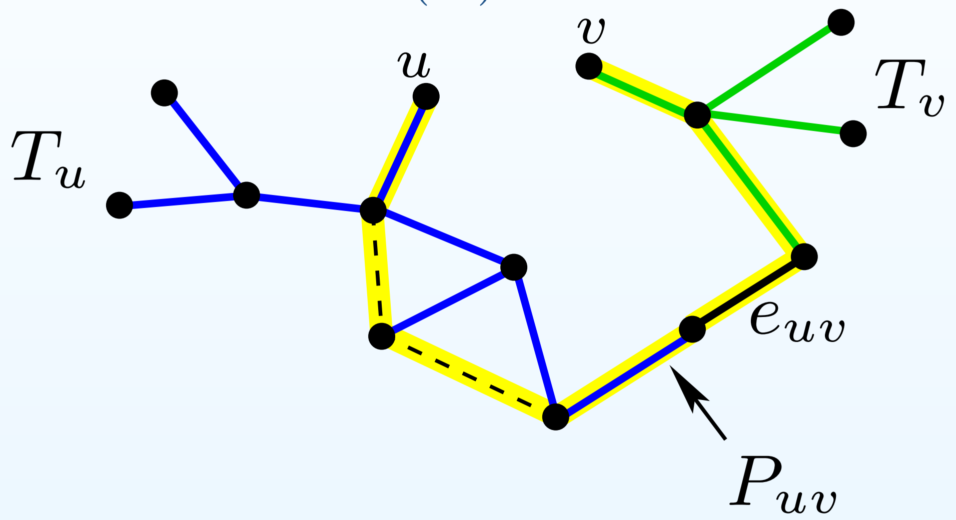
- Have shown: e has maximal weight on cycle C
- Assume for contradiction that $e \in \mathrm{MSF}(E)$



- Removing e from $\mathrm{MSF}(E)$ splits a tree into two trees: T_u containing u and T_v containing v
- $C \setminus \{e\}$ is a path P_{uv} in G from u to v
- Some edge $e_{uv} \in P_{uv}$ connects T_u with T_v

Proving $\mathrm{MSF}(E) = \mathrm{MSF}(\cup_{i=1}^{k} E(M_i))$

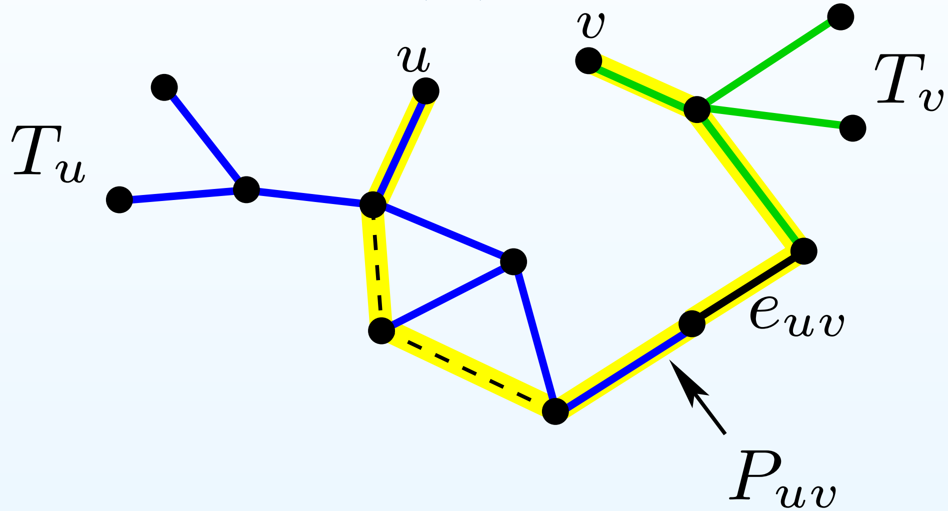
- Have shown: e has maximal weight on cycle C
- Assume for contradiction that $e \in \mathrm{MSF}(E)$



- Removing e from $\mathrm{MSF}(E)$ splits a tree into two trees: T_u containing u and T_v containing v
- $C \setminus \{e\}$ is a path P_{uv} in G from u to v
- Some edge $e_{uv} \in P_{uv}$ connects T_u with T_v

Proving $\mathrm{MSF}(E) = \mathrm{MSF}(\cup_{i=1}^{k} E(M_i))$

- Have shown: e has maximal weight on cycle C
- Assume for contradiction that $e \in \mathrm{MSF}(E)$



- Removing e from $\mathrm{MSF}(E)$ splits a tree into two trees: T_u containing u and T_v containing v
- $C \setminus \{e\}$ is a path P_{uv} in G from u to v
- Some edge $e_{uv} \in P_{uv}$ connects T_u with T_v
- By the property of C, reconnecting with e_{uv} does not increase weight
- This contradicts the uniqueness of $\mathrm{MSF}(E)$