Outline

Dynamic Programming

- Break up a problem into many overlapping sub-problems
- Build solutions for larger and larger sub-problems
- Use a table to store solutions for sub-problems for reuse

Comparison with greedy algorithms

- Greedy algorithm: each step is making a small progress towards constructing the solution
- Dynamic programming: the whole solution is constructed in the last step

Comparison with divide and conquer

- Divide and conquer: an instance is broken into many independent sub-instances, which are solved separately.
- Dynamic programming: the sub-instances we constructed are overlapping.

Definition of Cells for Problems We Learnt

- Weighted interval scheduling: $opt[i]$ = value of instance defined by jobs $\{1, 2, \cdots, i\}$
- Subset sum, knapsack: $opt[i, W']$ = value of instance with items $\{1, 2, \cdots, i\}$ and budget W'
- Longest common subsequence: $opt[i, j]$ = value of instance defined by $A[1..i]$ and $B[1..j]$
- Shortest paths in DAG: $f[v]$ = length of shortest path from s to v
- Matrix chain multiplication, optimum binary search tree: $opt[i, j]$ = value of instances defined by matrices i to j

Outline

Important notations/algorithms

- Introduction:
  - Asymptotic analysis: O, Ω, Θ, compare the orders

Important notations/algorithms

- Introduction:
  - Asymptotic analysis: O, Ω, Θ, compare the orders
  - Polynomial time (efficient algorithm), exponential time

Important notations/algorithms

- Introduction:
  - Asymptotic analysis: O, Ω, Θ, compare the orders
  - Polynomial time (efficient algorithm), exponential time
- Graph Basics:

Important notations/algorithms

- Introduction:
  - Asymptotic analysis: O, Ω, Θ, compare the orders
  - Polynomial time (efficient algorithm), exponential time
- Graph Basics:
  - Undirected graph, directed graph

Important notations/algorithms

- Introduction:
  - Asymptotic analysis: O, Ω, Θ, compare the orders
  - Polynomial time (efficient algorithm), exponential time
- Graph Basics:
  - Undirected graph, directed graph
  - Two representations: adjacency matrix, linked lists

Important notations/algorithms

- Introduction:
  - Asymptotic analysis: O, Ω, Θ, compare the orders
  - Polynomial time (efficient algorithm), exponential time
- Graph Basics:
  - Undirected graph, directed graph
  - Two representations: adjacency matrix, linked lists
  - Path, cycle, tree, directed acyclic graph, bipartite graph

Important notations/algorithms

- Introduction:
  - Asymptotic analysis: O, Ω, Θ, compare the orders
  - Polynomial time (efficient algorithm), exponential time
- Graph Basics:
  - Undirected graph, directed graph
  - Two representations: adjacency matrix, linked lists
  - Path, cycle, tree, directed acyclic graph, bipartite graph
  - Connectivity problem: BFS and DFS algorithm

Important notations/algorithms

- Introduction:
  - Asymptotic analysis: O, Ω, Θ, compare the orders
  - Polynomial time (efficient algorithm), exponential time
- Graph Basics:
  - Undirected graph, directed graph
  - Two representations: adjacency matrix, linked lists
  - Path, cycle, tree, directed acyclic graph, bipartite graph
  - Connectivity problem: BFS and DFS algorithm
  - Testing Bipartiteness problem: test-bipartiteness-BFS or test-bipartiteness-DFS algorithm

Important notations/algorithms

- Introduction:
  - Asymptotic analysis: O, Ω, Θ, compare the orders
  - Polynomial time (efficient algorithm), exponential time
- Graph Basics:
  - Undirected graph, directed graph
  - Two representations: adjacency matrix, linked lists
  - Path, cycle, tree, directed acyclic graph, bipartite graph
  - Connectivity problem: BFS and DFS algorithm
  - Testing Bipartiteness problem: test-bipartiteness-BFS or test-bipartiteness-DFS algorithm
  - Topological Ordering problem: topological-sort algorithm

Important notations/algorithms

- Greedy algorithms: safety strategy+self reduce
  - Box Packing problem: greedy algorithm

Important notations/algorithms

- Greedy algorithms: safety strategy+self reduce
  - Box Packing problem: greedy algorithm
  - Interval Scheduling problem: schedule algorithm

Important notations/algorithms

- Greedy algorithms: safety strategy+self reduce
  - Box Packing problem: greedy algorithm
  - Interval Scheduling problem: schedule algorithm
  - Interval Partitioning problem: partition algorithm

Important notations/algorithms

- Greedy algorithms: safety strategy+self reduce
  - Box Packing problem: greedy algorithm
  - Interval Scheduling problem: schedule algorithm
  - Interval Partitioning problem: partition algorithm
  - Offline Caching problem: FIF algorithm

Important notations/algorithms

- Greedy algorithms: safety strategy+self reduce
  - Box Packing problem: greedy algorithm
  - Interval Scheduling problem: schedule algorithm
  - Interval Partitioning problem: partition algorithm
  - Offline Caching problem: FIF algorithm
  - Priority Queue: heap

Important notations/algorithms

- Greedy algorithms: safety strategy+self reduce
  - Box Packing problem: greedy algorithm
  - Interval Scheduling problem: schedule algorithm
  - Interval Partitioning problem: partition algorithm
  - Offline Caching problem: FIF algorithm
  - Priority Queue: heap
  - Huffman Code problem: prefix code notation, Huffman algorithm

Important notations/algorithms

- Greedy algorithms: safety strategy+self reduce
  - Box Packing problem: greedy algorithm
  - Interval Scheduling problem: schedule algorithm
  - Interval Partitioning problem: partition algorithm
  - Offline Caching problem: FIF algorithm
  - Priority Queue: heap
  - Huffman Code problem: prefix code notation, Huffman algorithm
  - Exercise problems: Job scheduling with deadline, clustering problem, Coin Problem, Weighted scheduling problem

Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)

Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm

Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm
  - Selection problem: selection algorithm based on quicksort

Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm
  - Selection problem: selection algorithm based on quicksort
  - Polynomial Multiplication problem: multiply algorithm

Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm
  - Selection problem: selection algorithm based on quicksort
  - Polynomial Multiplication problem: multiply algorithm
  - Recurrences: recursive-tree method and Master Theorem

Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm
  - Selection problem: selection algorithm based on quicksort
  - Polynomial Multiplication problem: multiply algorithm
  - Recurrences: recursive-tree method and Master Theorem
  - Fibonacci number problem: power algorithm

Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm
  - Selection problem: selection algorithm based on quicksort
  - Polynomial Multiplication problem: multiply algorithm
  - Recurrences: recursive-tree method and Master Theorem
  - Fibonacci number problem: power algorithm
  - Exercise problems: Modular Exponentiation Problem, Matrix Multiplication, Closest Pair, Convex Hull

Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule

Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
  - Subset Sum problem: DP algorithm + Recovering optimal schedule

Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
  - Subset Sum problem: DP algorithm + Recovering optimal schedule
  - Knapsack problem: DP algorithm + Recovering optimal schedule

Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
  - Subset Sum problem: DP algorithm + Recovering optimal schedule
  - Knapsack problem: DP algorithm + Recovering optimal schedule
  - Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule

Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
  - Subset Sum problem: DP algorithm + Recovering optimal schedule
  - Knapsack problem: DP algorithm + Recovering optimal schedule
  - Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
  - Edit distance with insertions and deletions problem: apply algorithm for LCS problem

Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
  - Subset Sum problem: DP algorithm + Recovering optimal schedule
  - Knapsack problem: DP algorithm + Recovering optimal schedule
  - Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
  - Edit distance with insertions and deletions problem: apply algorithm for LCS problem
  - Edit distance with insertions, deletions and replacing problem

Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
  - Subset Sum problem: DP algorithm + Recovering optimal schedule
  - Knapsack problem: DP algorithm + Recovering optimal schedule
  - Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
  - Edit distance with insertions and deletions problem: apply algorithm for LCS problem
  - Edit distance with insertions, deletions and replacing problem
  - Shortest Path in Directed Acyclic Graph (DAG): Shortest Paths in DAG algorithm + print-path algorithm

Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
  - Subset Sum problem: DP algorithm + Recovering optimal schedule
  - Knapsack problem: DP algorithm + Recovering optimal schedule
  - Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
  - Edit distance with insertions and deletions problem: apply algorithm for LCS problem
  - Edit distance with insertions, deletions and replacing problem
  - Shortest Path in Directed Acyclic Graph (DAG): Shortest Paths in DAG algorithm + print-path algorithm
  - Matrix Chain Multiplication problem: matrix-chain-multiplication algorithm + print-optimal-order alg

Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
  - Subset Sum problem: DP algorithm + Recovering optimal schedule
  - Knapsack problem: DP algorithm + Recovering optimal schedule
  - Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
  - Edit distance with insertions and deletions problem: apply algorithm for LCS problem
  - Edit distance with insertions, deletions and replacing problem
  - Shortest Path in Directed Acyclic Graph (DAG): Shortest Paths in DAG algorithm + print-path algorithm
  - Matrix Chain Multiplication problem: matrix-chain-multiplication algorithm + print-optimal-order alg
  - Optimum Binary Search Tree Problem: Optimum Binary Search Tree alg + Print Tree alg

CSE 431/531: Algorithm Analysis and Design (Spring 2024)

Graph Algorithms

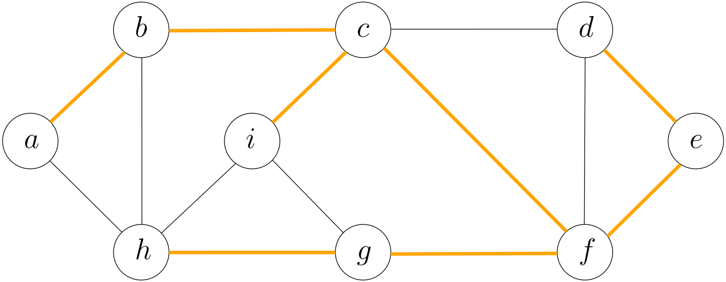
Lecturer: Kelin Luo

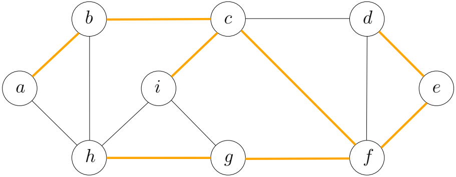
Department of Computer Science and Engineering
University at Buffalo

Outline

Spanning Tree

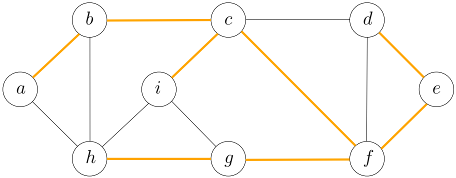
Def. Given a connected graph $G = (V, E)$, a spanning tree $T = (V, F)$ of G is a sub-graph of G that is a tree including all vertices V.





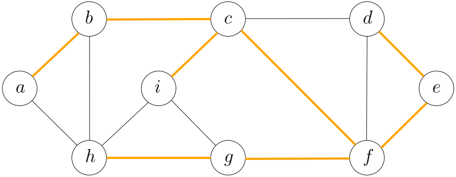
Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;



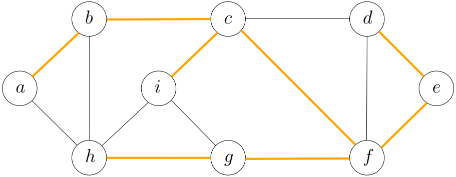
Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;



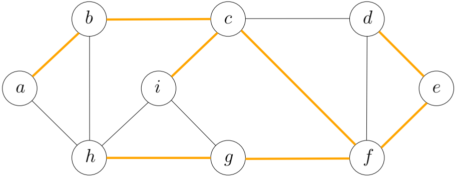
Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;
- T is connected and has $n - 1$ edges;



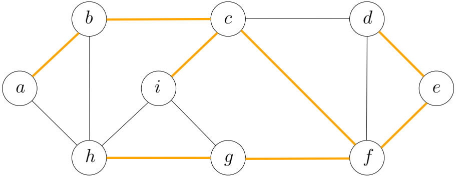
Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;
- T is connected and has $n - 1$ edges;
- T is acyclic and has $n - 1$ edges;



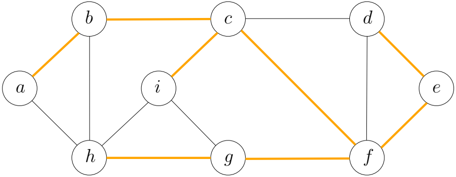
Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;
- T is connected and has $n - 1$ edges;
- T is acyclic and has $n - 1$ edges;
- T is minimally connected: removal of any edge disconnects it;



Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;
- T is connected and has $n - 1$ edges;
- T is acyclic and has $n - 1$ edges;
- T is minimally connected: removal of any edge disconnects it;
- T is maximally acyclic: addition of any edge creates a cycle;



Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;
- T is connected and has $n - 1$ edges;
- T is acyclic and has $n - 1$ edges;
- T is minimally connected: removal of any edge disconnects it;
- T is maximally acyclic: addition of any edge creates a cycle;
- T has a unique simple path between every pair of nodes.

- How to find a spanning tree?
  - BFS

- How to find a spanning tree?
  - BFS
  - DFS

Minimum Spanning Tree (MST) Problem

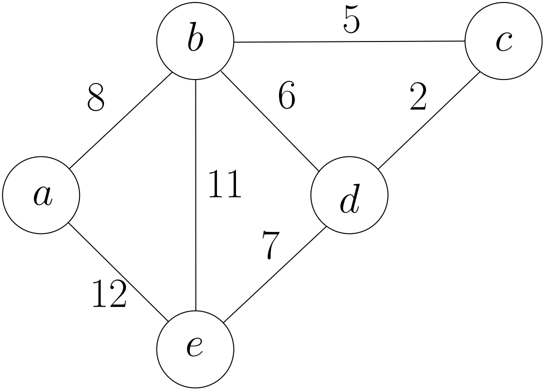
Input: Graph $G = (V, E)$ and edge weights $w : E \to \mathbb{R}$

Output: the spanning tree T of G with the minimum total weight

Minimum Spanning Tree (MST) Problem

Input: Graph $G = (V, E)$ and edge weights $w : E \to \mathbb{R}$

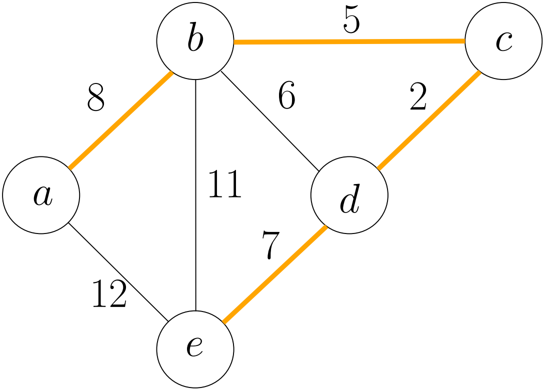
Output: the spanning tree T of G with the minimum total weight



Minimum Spanning Tree (MST) Problem

Input: Graph $G = (V, E)$ and edge weights $w : E \to \mathbb{R}$

Output: the spanning tree T of G with the minimum total weight



Recall: Steps of Designing A Greedy Algorithm

- Design a "reasonable" strategy
- Prove that the reasonable strategy is "safe" (key, usually done by "exchanging argument")
- Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem (usually trivial)

Def. A choice is "safe" if there is an optimum solution that is "consistent" with the choice

Recall: Steps of Designing A Greedy Algorithm

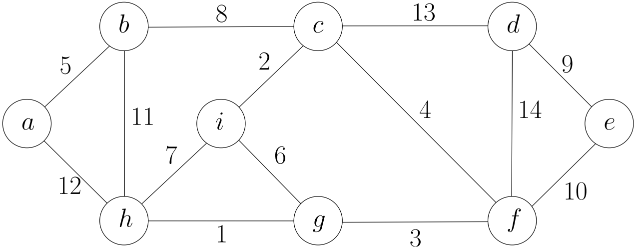
- Design a "reasonable" strategy
- Prove that the reasonable strategy is "safe" (key, usually done by "exchanging argument")
- Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem (usually trivial)

Def. A choice is "safe" if there is an optimum solution that is "consistent" with the choice

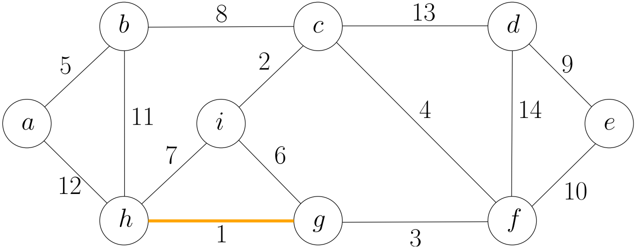
Two Classic Greedy Algorithms for MST

- Kruskal's Algorithm
- Prim's Algorithm

Outline



Q: Which edge can be safely included in the MST?



Q: Which edge can be safely included in the MST?

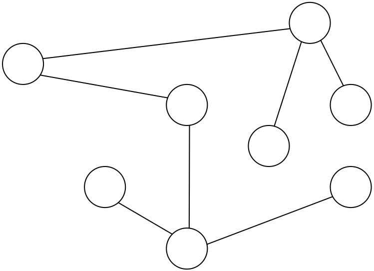
A: The edge with the smallest weight (lightest edge).

Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Proof.

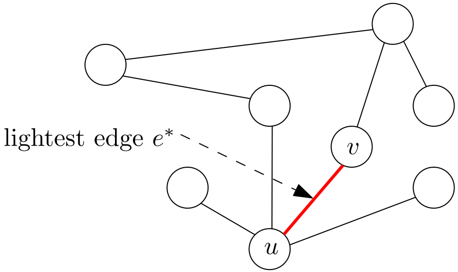
- Take a minimum spanning tree T



Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Proof.

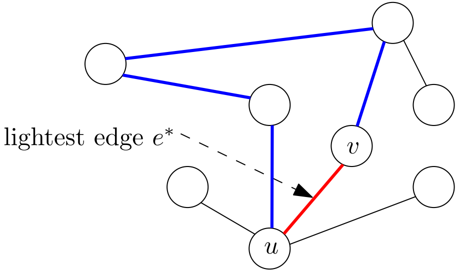
- Take a minimum spanning tree T
- Assume the lightest edge e^* is not in T



Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Proof.

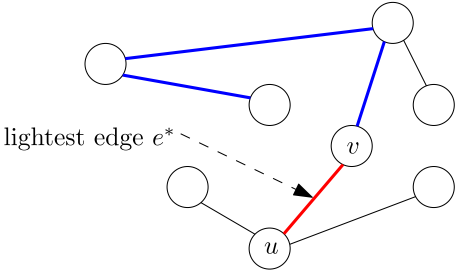
- Take a minimum spanning tree T
- Assume the lightest edge e^* is not in T
- There is a unique path in T connecting u and v



Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Proof.

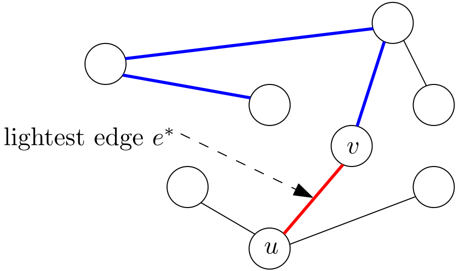
- Take a minimum spanning tree T
- Assume the lightest edge e^* is not in T
- There is a unique path in T connecting u and v
- Remove any edge e in the path to obtain tree T'



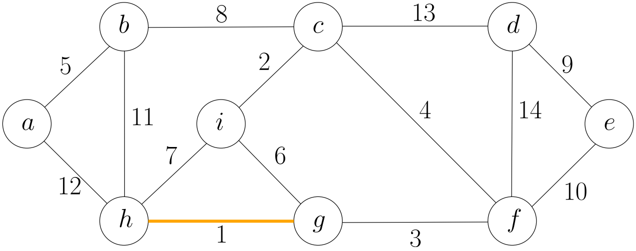
Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Proof.

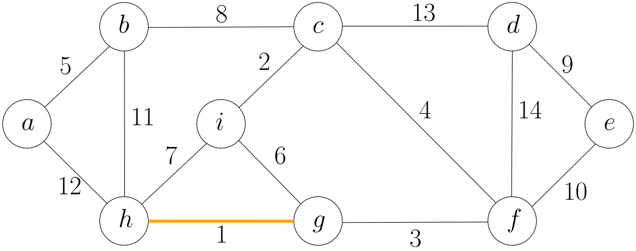
- Take a minimum spanning tree T
- Assume the lightest edge e^* is not in T
- There is a unique path in T connecting u and v
- Remove any edge e in the path to obtain tree T'
- $w(e^*) \leq w(e) \implies w(T') \leq w(T)$: T' is also a MST $\square$



Is the Residual Problem Still a MST Problem?

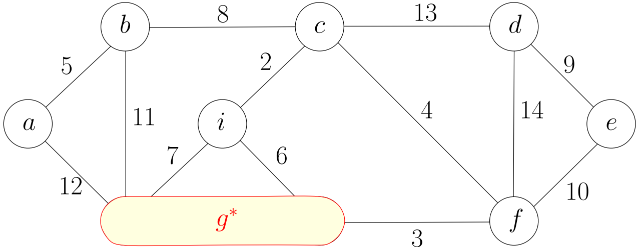


Is the Residual Problem Still a MST Problem?



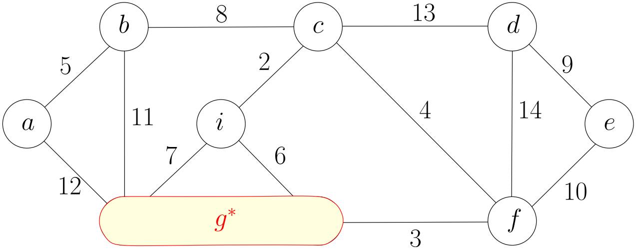
- Residual problem: find the minimum spanning tree that contains edge (g, h)

Is the Residual Problem Still a MST Problem?



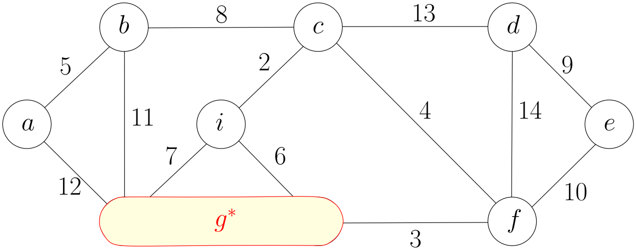
- Residual problem: find the minimum spanning tree that contains edge (g, h)
- Contract the edge (g, h)

Is the Residual Problem Still a MST Problem?

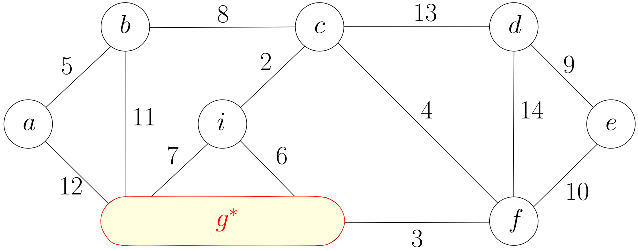


- Residual problem: find the minimum spanning tree that contains edge (g, h)
- Contract the edge (g, h)
- Residual problem: find the minimum spanning tree in the contracted graph

Contraction of an Edge (u, v)

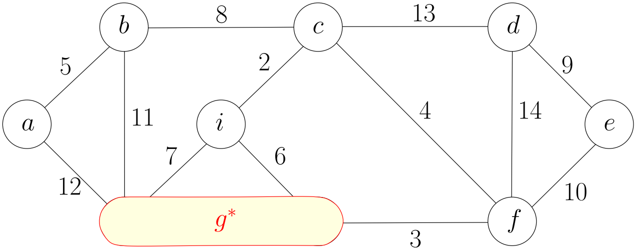


Contraction of an Edge (u, v)



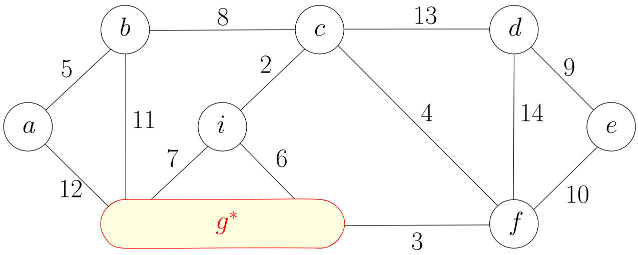
- Remove u and v from the graph, and add a new vertex u^*

Contraction of an Edge (u, v)



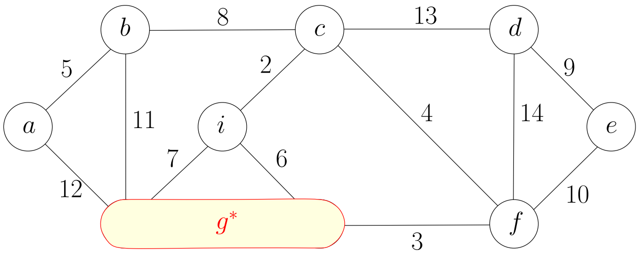
- Remove u and v from the graph, and add a new vertex u^*
- Remove all edges (u, v) from E

Contraction of an Edge (u, v)



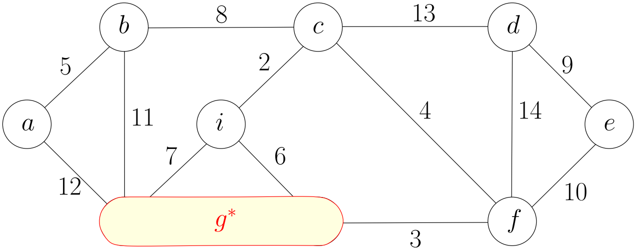
- Remove u and v from the graph, and add a new vertex u^*
- Remove all edges (u, v) from E
- For every edge $(u, w) \in E, w \neq v$, change it to (u^*, w)

Contraction of an Edge (u, v)



- Remove u and v from the graph, and add a new vertex u^*
- Remove all edges (u, v) from E
- For every edge $(u, w) \in E, w \neq v$, change it to (u^*, w)
- For every edge $(v, w) \in E, w \neq u$, change it to (u^*, w)

Contraction of an Edge (u, v)



- Remove u and v from the graph, and add a new vertex u^*
- Remove all edges (u, v) from E
- For every edge $(u, w) \in E, w \neq v$, change it to (u^*, w)
- For every edge $(v, w) \in E, w \neq u$, change it to (u^*, w)
- May create parallel edges! E.g. : two edges (i, g^*)

Greedy Algorithm

Repeat the following step until G contains only one vertex:

1. Choose the lightest edge e^*, add e^* to the spanning tree
2. Contract e^* and update G be the contracted graph

Greedy Algorithm

Repeat the following step until G contains only one vertex:

1. Choose the lightest edge e^*, add e^* to the spanning tree
2. Contract e^* and update G be the contracted graph

Q: What edges are removed due to contractions?

Greedy Algorithm

Repeat the following step until G contains only one vertex:

1. Choose the lightest edge e^*, add e^* to the spanning tree
2. Contract e^* and update G be the contracted graph

Q: What edges are removed due to contractions?

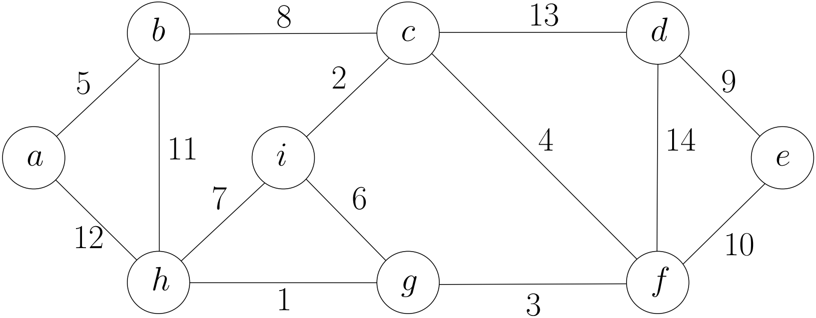
A: Edge (u, v) is removed if and only if there is a path connecting u and v formed by edges we selected

Greedy Algorithm

MST-Greedy(G, w)

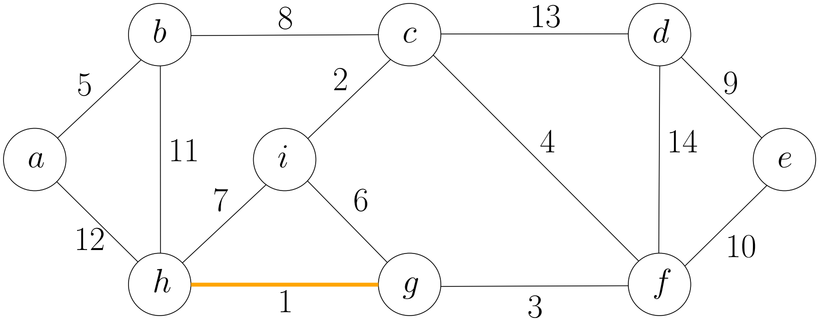
$F \leftarrow \emptyset$
sort edges in E in non-decreasing order of weights w
for each edge (u, v) in the order **do**
    if u and v are not connected by a path of edges in F **then**
        $F \leftarrow F \cup \{(u, v)\}$
return (V, F)

Kruskal's Algorithm: Example



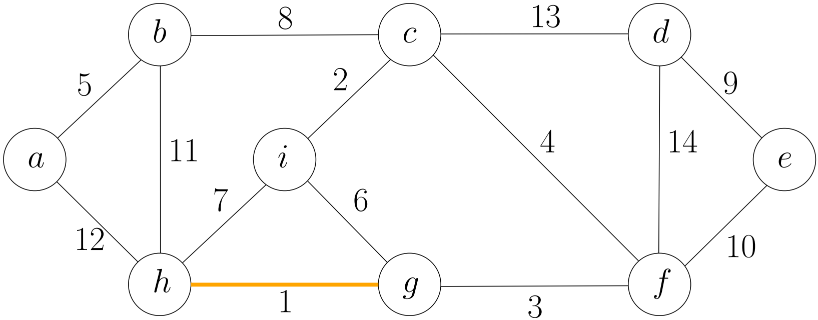
Sets: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}$

Kruskal's Algorithm: Example



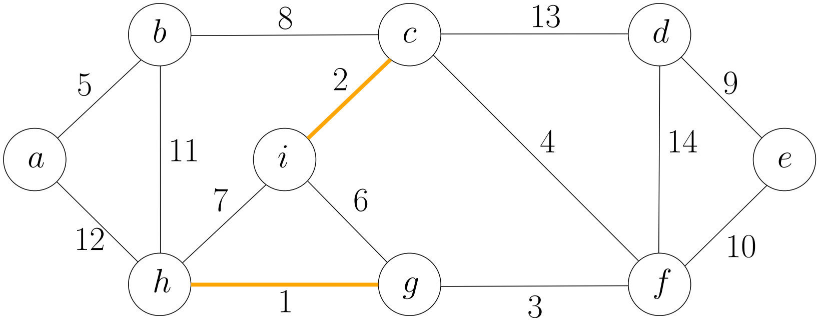
Sets: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}$

Kruskal's Algorithm: Example



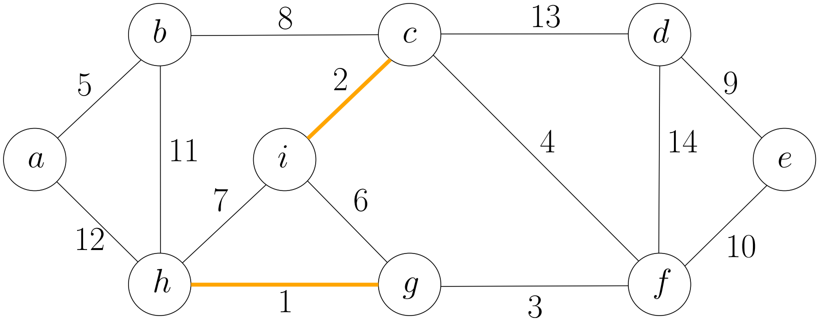
Sets: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g, h\}, \{i\}$

Kruskal's Algorithm: Example



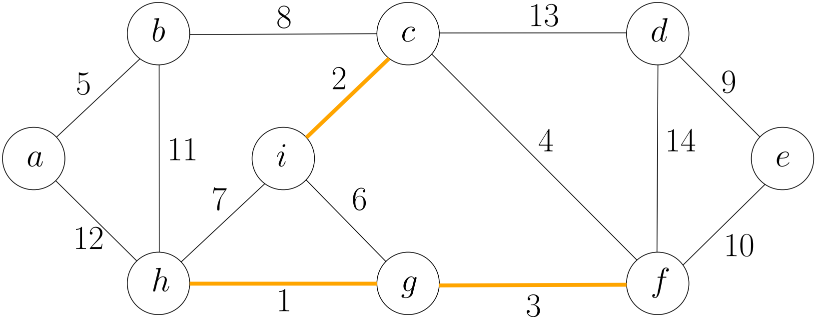
Sets: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g, h\}, \{i\}$

Kruskal's Algorithm: Example



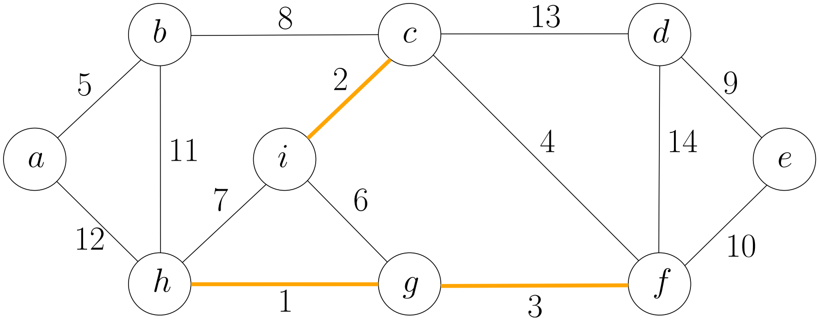
Sets: $\{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f\}, \{g, h\}$

Kruskal's Algorithm: Example



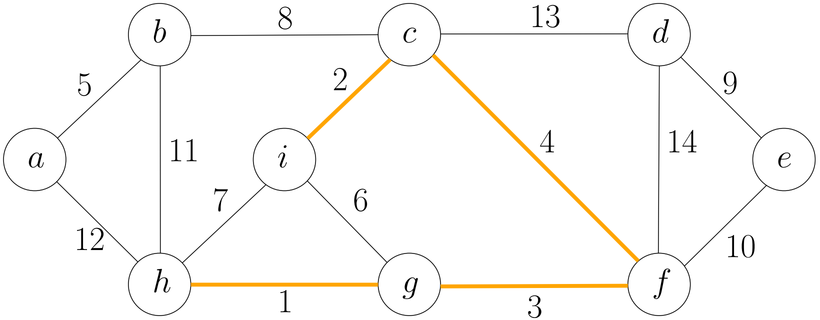
Sets: $\{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f\}, \{g, h\}$

Kruskal's Algorithm: Example



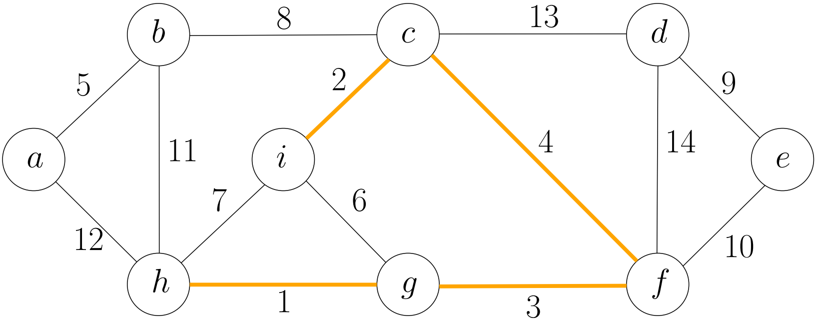
Sets: $\{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f, g, h\}$

Kruskal's Algorithm: Example



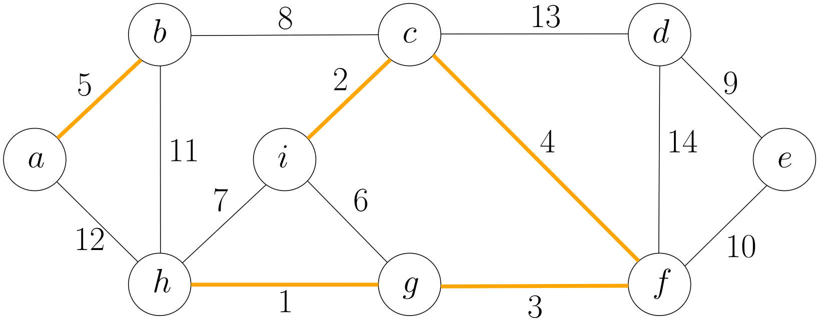
Sets: $\{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f, g, h\}$

Kruskal's Algorithm: Example



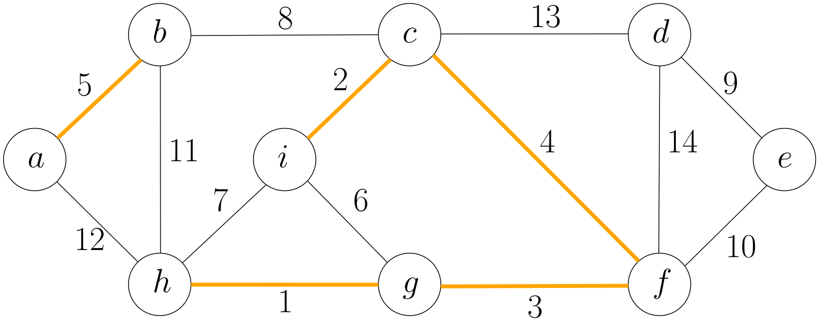
Sets: $\{a\}, \{b\}, \{c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



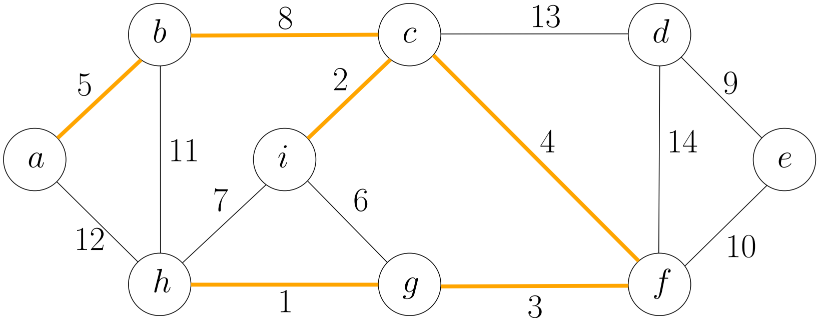
Sets: $\{a\}, \{b\}, \{c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



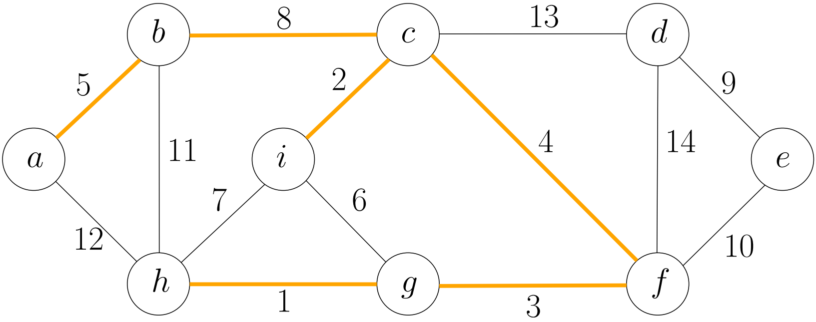
Sets: $\{a, b\}, \{c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



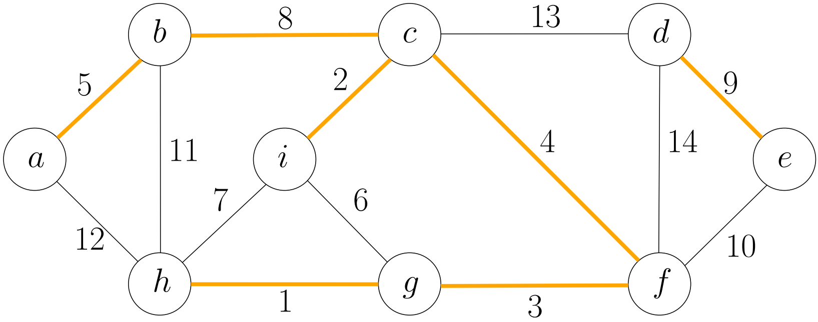
Sets: $\{a, b\}, \{c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



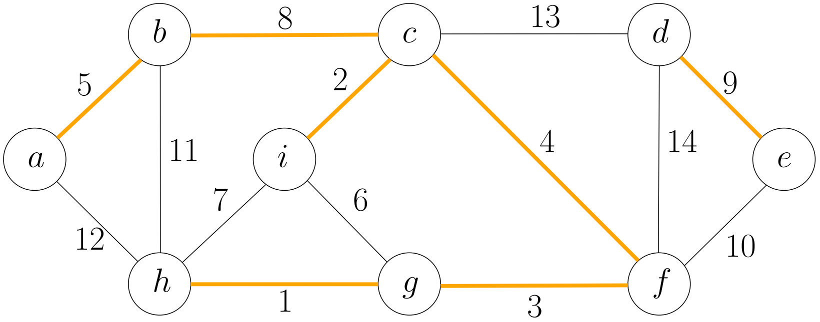
Sets: $\{a, b, c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



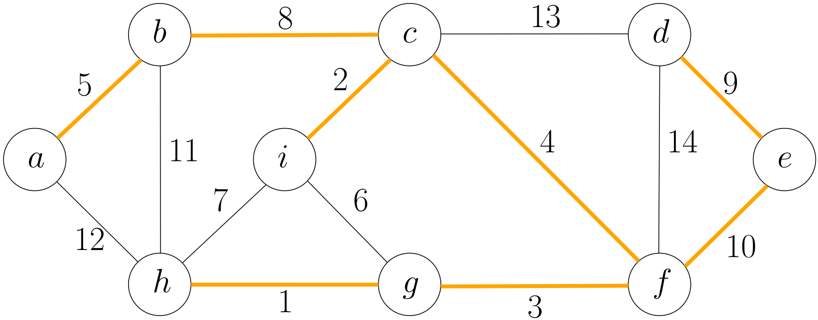
Sets: $\{a, b, c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



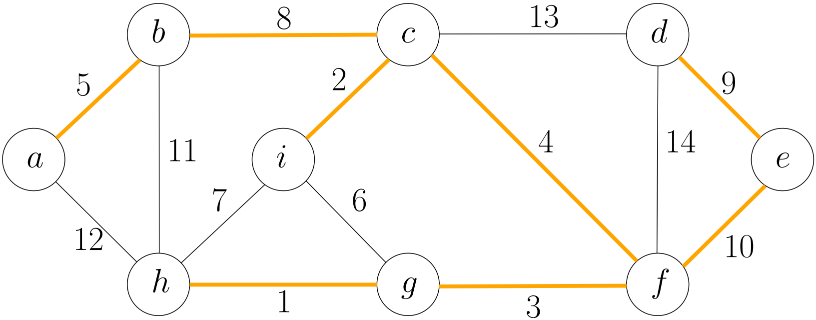
Sets: $\{a, b, c, i, f, g, h\}, \{d, e\}$

Kruskal's Algorithm: Example



Sets: $\{a, b, c, i, f, g, h\}, \{d, e\}$

Kruskal's Algorithm: Example



Sets: $\{a, b, c, i, f, g, h, d, e\}$

Kruskal's Algorithm: Efficient Implementation of Greedy Algorithm

MST-Kruskal(G, w)

$F \leftarrow \emptyset$
$\mathcal{S} \leftarrow \{\{v\} : v \in V\}$
sort the edges of E in non-decreasing order of weights w
for each edge $(u, v) \in E$ in the order **do**
    $S_u \leftarrow$ the set in $\mathcal{S}$ containing u
    $S_v \leftarrow$ the set in $\mathcal{S}$ containing v
    if $S_u \neq S_v$ **then**
        $F \leftarrow F \cup \{(u, v)\}$
        $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S_u\} \setminus \{S_v\} \cup \{S_u \cup S_v\}$
return (V, F)