Outline

1. Weighted Interval Scheduling
2. Subset Sum Problem
3. Knapsack Problem
4. Longest Common Subsequence
   - Longest Common Subsequence in Linear Space
5. Shortest Paths in Directed Acyclic Graphs
6. Matrix Chain Multiplication
7. Optimum Binary Search Tree
8. Summary
9. Summary of Studies Until April
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: \( opt[i] = \text{value of instance defined by jobs } \{1, 2, \cdots, i\} \)
- Subset sum, knapsack: \( opt[i, W'] = \text{value of instance with items } \{1, 2, \cdots, i\} \text{ and budget } W' \)
- Longest common subsequence: \( opt[i, j] = \text{value of instance defined by } A[1..i] \text{ and } B[1..j] \)
- Shortest paths in DAG: \( f[v] = \text{length of shortest path from } s \text{ to } v \)
- Matrix chain multiplication, optimum binary search tree: \( opt[i, j] = \text{value of instances defined by matrices } i \text{ to } j \)
Outline

1. Weighted Interval Scheduling
2. Subset Sum Problem
3. Knapsack Problem
4. Longest Common Subsequence
   - Longest Common Subsequence in Linear Space
5. Shortest Paths in Directed Acyclic Graphs
6. Matrix Chain Multiplication
7. Optimum Binary Search Tree
8. Summary
9. Summary of Studies Until April
Introduction:

Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
Introduction:
- Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
- Polynomial time (efficient algorithm), exponential time
Introduction:

- Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
- Polynomial time (efficient algorithm), exponential time

Graph Basics:
Introduction:

- Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
- Polynomial time (efficient algorithm), exponential time

Graph Basics:

- Undirected graph, directed graph
Important notations/algorithms

- **Introduction:**
  - Asymptotic analysis: $O$, $\Omega$, $\Theta$, 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: $O$, $\Omega$, $\Theta$, 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: $O$, $\Omega$, $\Theta$, 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: $O$, $\Omega$, $\Theta$, 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: $O$, $\Omega$, $\Theta$, 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
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

Exercise problems: Modular Exponentiation Problem, Matrix Multiplication, Closest Pair, Convex Hull
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
- 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
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
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
Outline

1 Minimum Spanning Tree
   - Kruskal’s Algorithm
   - Reverse-Kruskal’s Algorithm
   - Prim’s Algorithm

2 Single Source Shortest Paths
   - Dijkstra’s Algorithm

3 Shortest Paths in Graphs with Negative Weights

4 All-Pair Shortest Paths and Floyd-Warshall
**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;
- $T$ 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.
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 $w : E \rightarrow \mathbb{R}$

**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 $w : E \rightarrow \mathbb{R}$

**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

1. Minimum Spanning Tree
   - Kruskal’s Algorithm
   - Reverse-Kruskal’s Algorithm
   - Prim’s Algorithm

2. Single Source Shortest Paths
   - Dijkstra’s Algorithm

3. Shortest Paths in Graphs with Negative Weights

4. All-Pair Shortest Paths and Floyd-Warshall
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 $T$

![Diagram of a minimum spanning tree](image)
**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.**

1. Take a minimum spanning tree $T$
2. Assume the lightest edge $e^*$ is not in $T$
3. There is a unique path in $T$ connecting $u$ and $v$
4. Remove any edge $e$ in the path to obtain tree $T'$
5. $w(e^*) \leq w(e) \implies w(T') \leq w(T)$: $T'$ is also a MST
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.
Residual problem: find the minimum spanning tree that contains edge \((g, h)\)
Residual problem: find the minimum spanning tree that contains edge $(g, h)$

Contract the edge $(g, h)$
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 \((u, v)\)

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

1: $F ← ∅$
2: sort edges in $E$ in non-decreasing order of weights $w$
3: for each edge $(u, v)$ in the order do
4: if $u$ and $v$ are not connected by a path of edges in $F$ then
5: $F ← F \cup \{(u, v)\}$
6: return $(V, F)$
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\}
Sets: \{a, b\}, \{c, i, f, g, h\}, \{d\}, \{e\}
Sets: \{a, b\}, \{c, i, f, g, h\}, \{d\}, \{e\}
Sets: \{a, b, c, i, f, g, h\}, \{d\}, \{e\}
Kruskal’s Algorithm: Example

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

1: $F \leftarrow \emptyset$
2: $S \leftarrow \{\{v\} : v \in V\}$
3: sort the edges of $E$ in non-decreasing order of weights $w$
4: for each edge $(u, v) \in E$ in the order do
5: $S_u \leftarrow$ the set in $S$ containing $u$
6: $S_v \leftarrow$ the set in $S$ containing $v$
7: if $S_u \neq S_v$ then
8: $F \leftarrow F \cup \{(u, v)\}$
9: $S \leftarrow S \setminus \{S_u\} \setminus \{S_v\} \cup \{S_u \cup S_v\}$
10: return $(V, F)$