CSC006P1M: Design and Analysis of Algorithms
Lecture 08 *(Greedy Method-II)*

Sumit Kumar Pandey

September 01, 2022

Minimum Spanning Tree

Problem

Given an undirected connected weighted graph $G = (V, E)$, find a spanning tree T of G of minimum cost.

Spanning Tree

A spanning tree T of an undirected graph G is a subgraph that is a tree which includes all vertices of a graph.

Minimum Spanning Tree

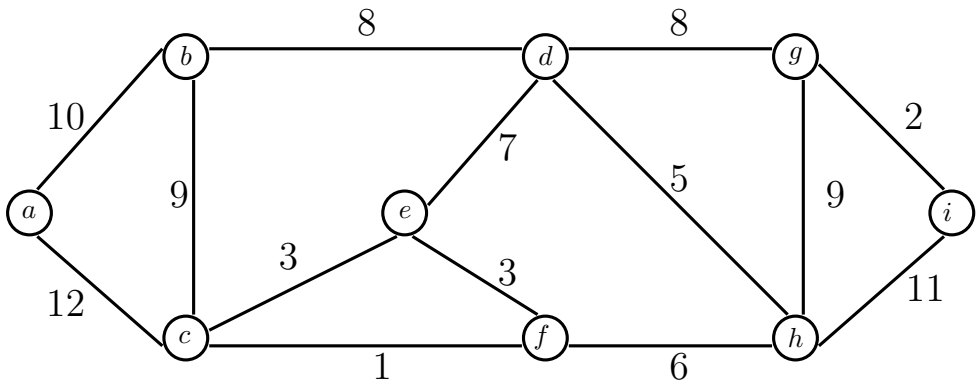
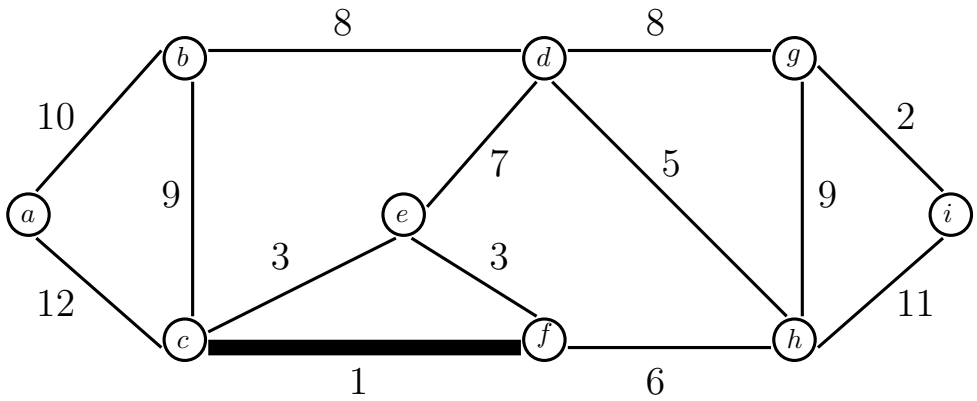


Figure: An Undirected Connected Graph

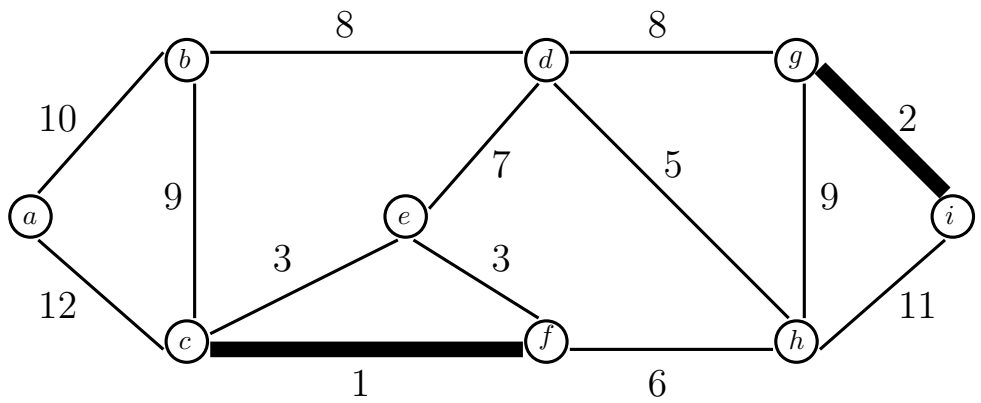
Minimum Spanning Tree

Greedy Method: Choose an edge with a minimum weight. Break the tie arbitrarily.



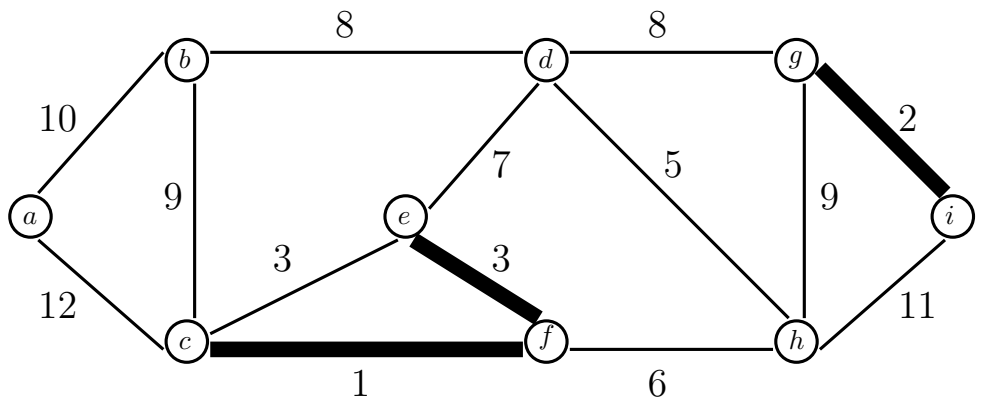
Minimum Spanning Tree

Greedy Method: Choose a minimum weight edge from the remaining edges in such a manner that the chosen edge does not form a cycle. Break the tie arbitrarily.



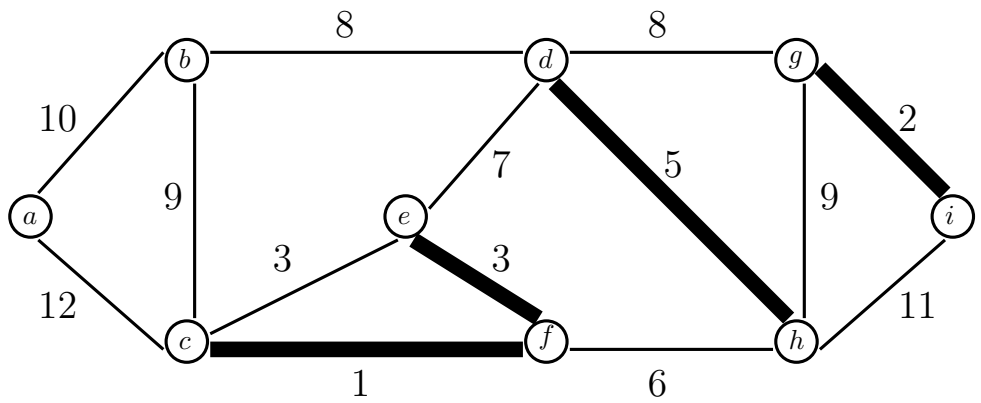
Minimum Spanning Tree

Greedy Method: Choose a minimum weight edge from the remaining edges in such a manner that the chosen edge does not form a cycle. Break the tie arbitrarily.



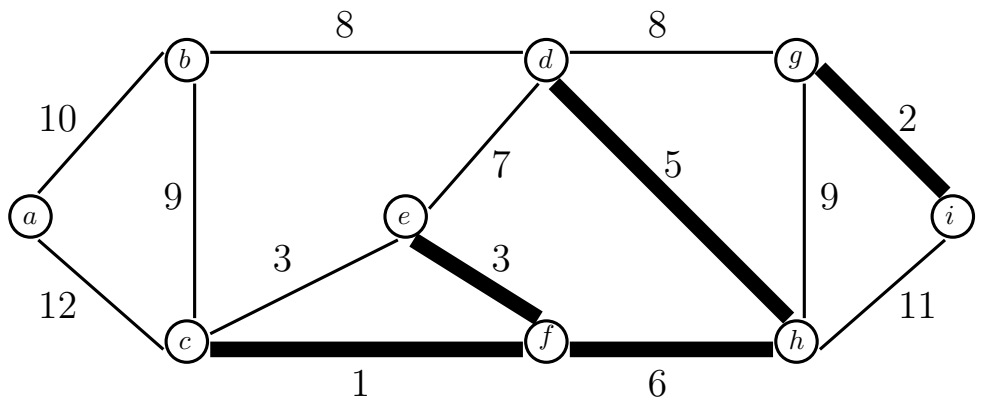
Minimum Spanning Tree

Greedy Method: Choose a minimum weight edge from the remaining edges in such a manner that the chosen edge does not form a cycle. Break the tie arbitrarily.



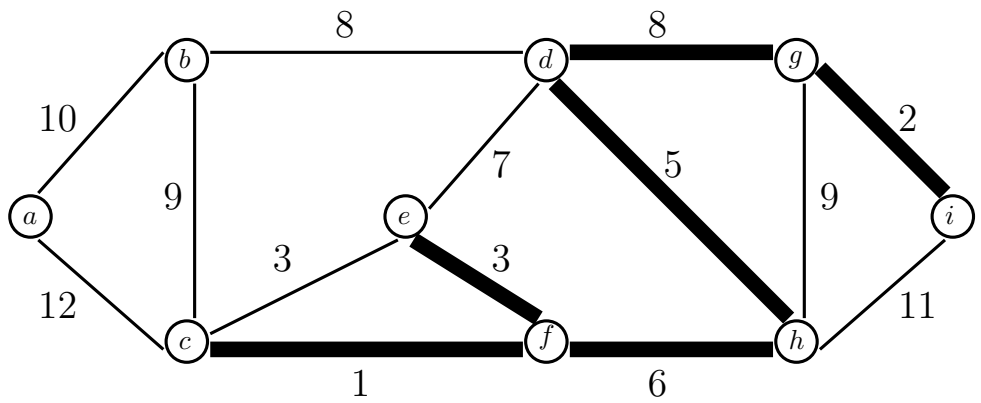
Minimum Spanning Tree

Greedy Method: Choose a minimum weight edge from the remaining edges in such a manner that the chosen edge does not form a cycle. Break the tie arbitrarily.



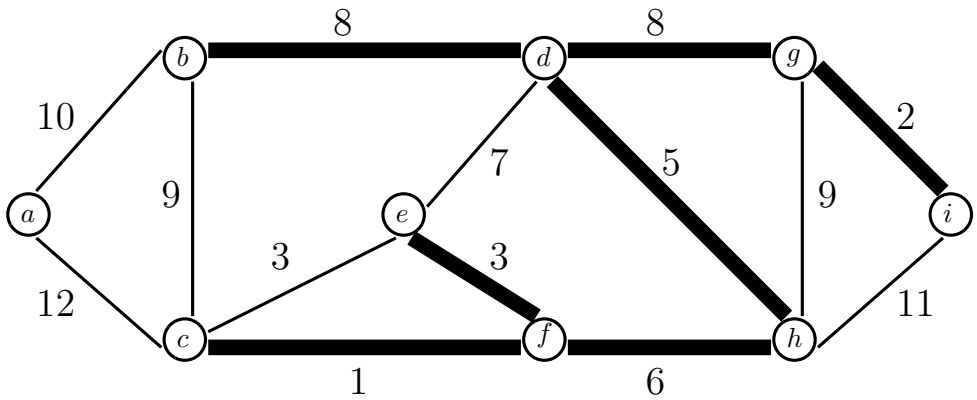
Minimum Spanning Tree

Greedy Method: Choose a minimum weight edge from the remaining edges in such a manner that the chosen edge does not form a cycle. Break the tie arbitrarily.



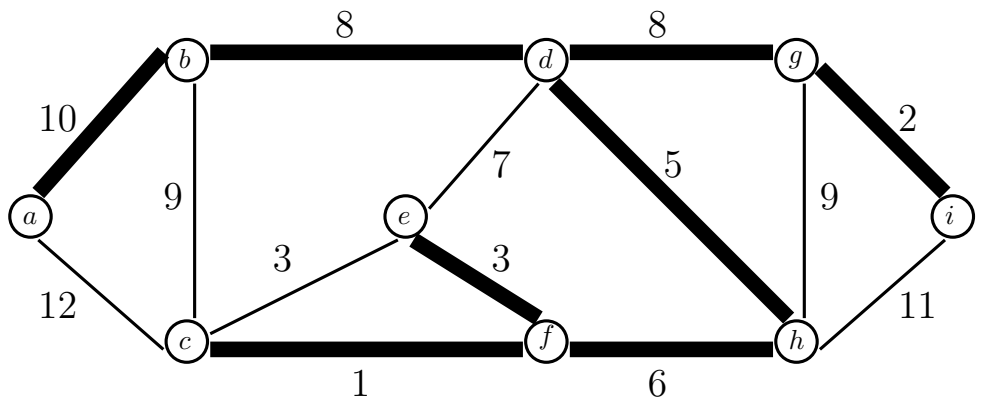
Minimum Spanning Tree

Greedy Method: Choose a minimum weight edge from the remaining edges in such a manner that the chosen edge does not form a cycle. Break the tie arbitrarily.



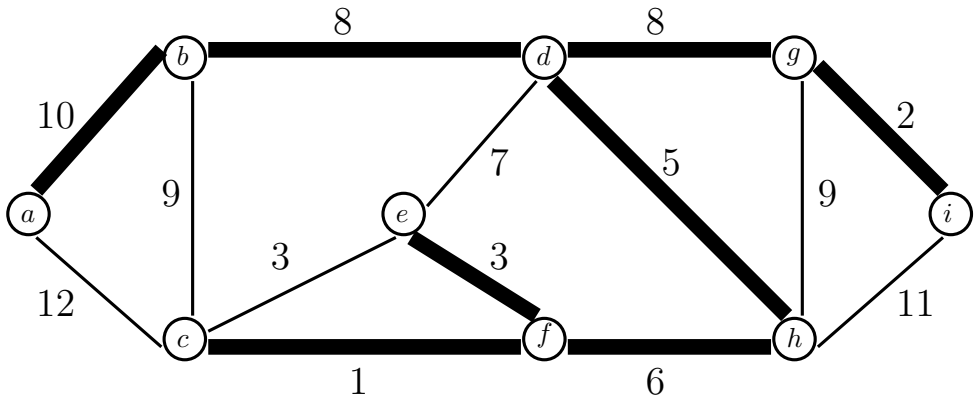
Minimum Spanning Tree

Greedy Method: Choose a minimum weight edge from the remaining edges in such a manner that the chosen edge does not form a cycle. Break the tie arbitrarily.



Minimum Spanning Tree

Final Minimum Spanning Tree:



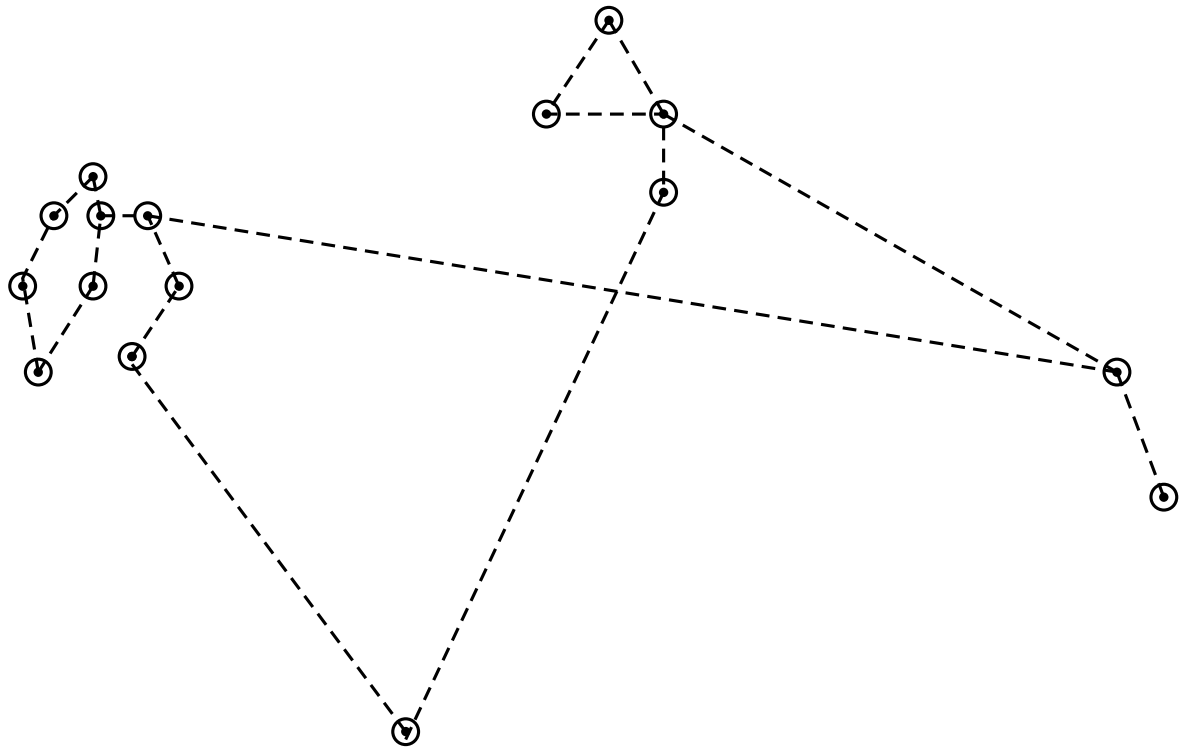
Cost of the tree = Sum of all weights in the tree = 43.

Proof of the algorithm

Let the graph be $G = (V, E)$ where $|V| = n$.

- Base Case: Start with the forest of n isolated vertices.

Proof of the algorithm



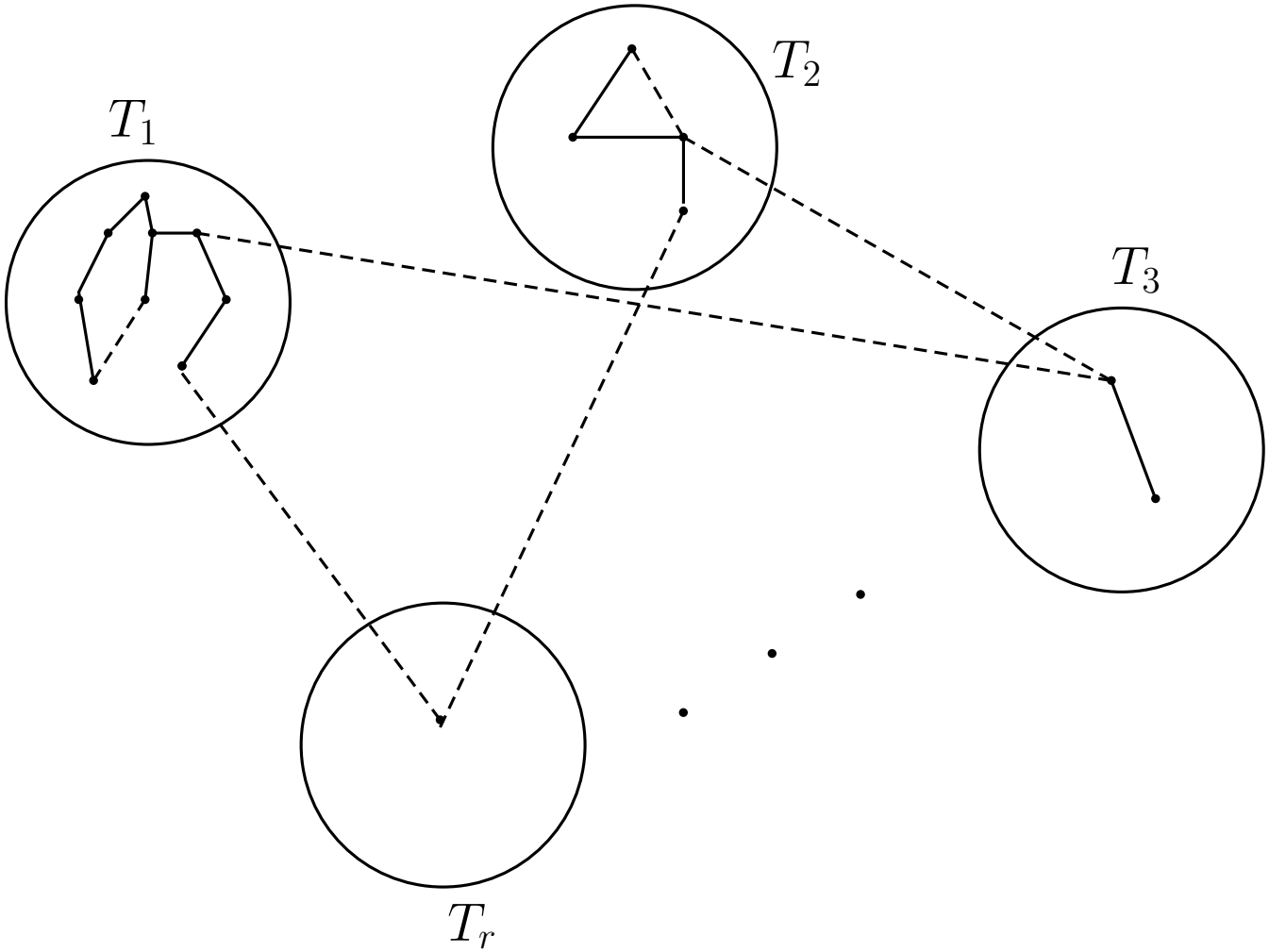
The forest of n isolated vertices. Each vertex is a tree. Each vertex is a trivial component.

Proof of the algorithm

Let the graph be $G = (V, E)$ where $|V| = n$.

- Base Case: Start with the forest of n isolated vertices.
- Assume that after k number of iterations,
  - The created graph has r many components and each component is a tree (an isolated vertex is a tree and a trivial component). Let these components be $T_1, T_2, \cdots, T_r$.
  - $T_1, T_2, \cdots, T_r$ are the subgraphs of some MST T. (Note that the base case satisfies this condition).

Proof of the algorithm



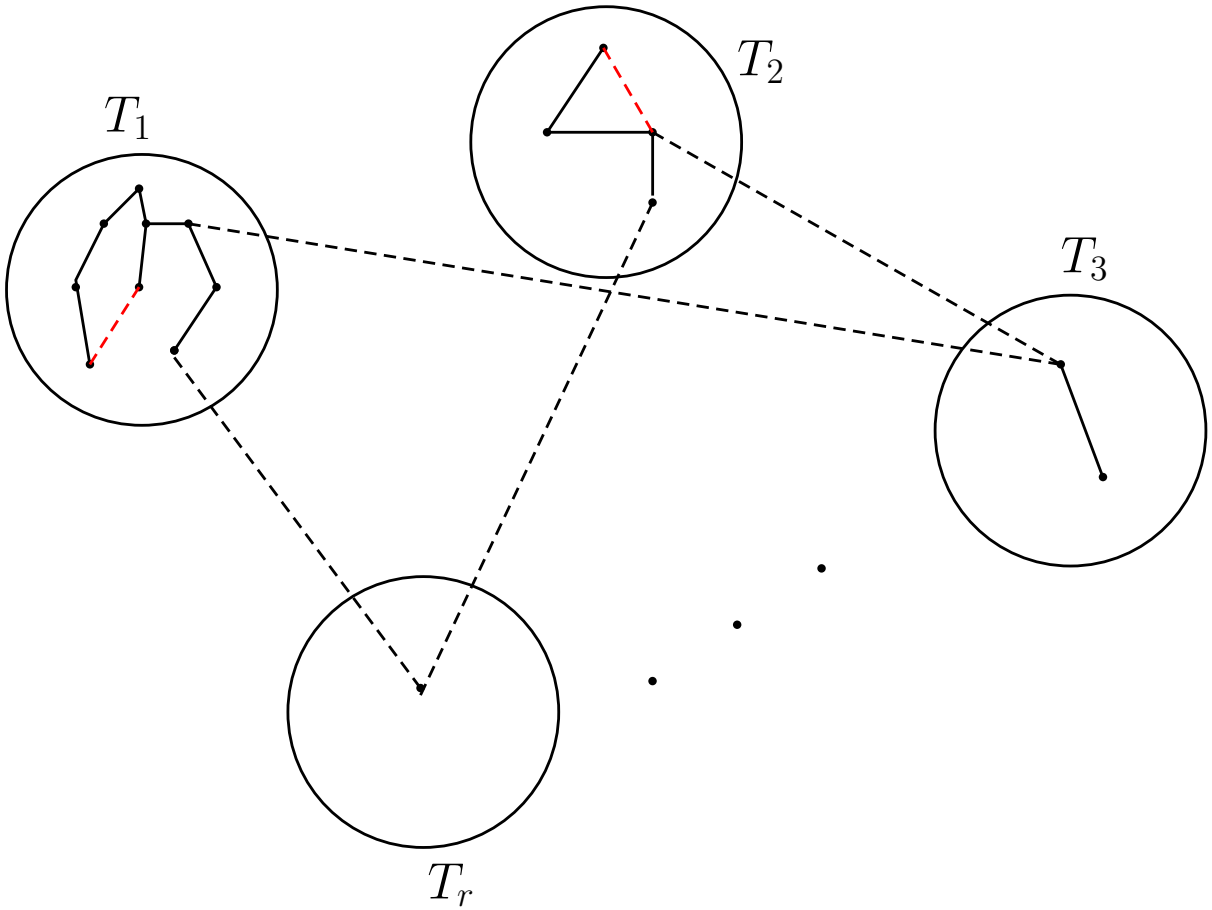
r many components - $T_1, T_2, T_3, \cdots, T_r$.

Proof of the algorithm

Let the graph be $G = (V, E)$ where $|V| = n$.

- Base Case: Start with the forest of n isolated vertices.
- Assume that after k number of iterations,
  - The created graph has r many components and each component is a tree (an isolated vertex is a tree and a trivial component). Let these components be $T_1, T_2, \cdots, T_r$.
  - $T_1, T_2, \cdots, T_r$ are the subgraphs of some MST T. (Note that the base case satisfies this condition).
- At $k + 1^{st}$ iteration: If the new edge in $k + 1^{st}$ iteration contains end vertices from the same component, it will create a cycle which is not allowed. So, the new edge must have end vertices in two different components.

Proof of the algorithm

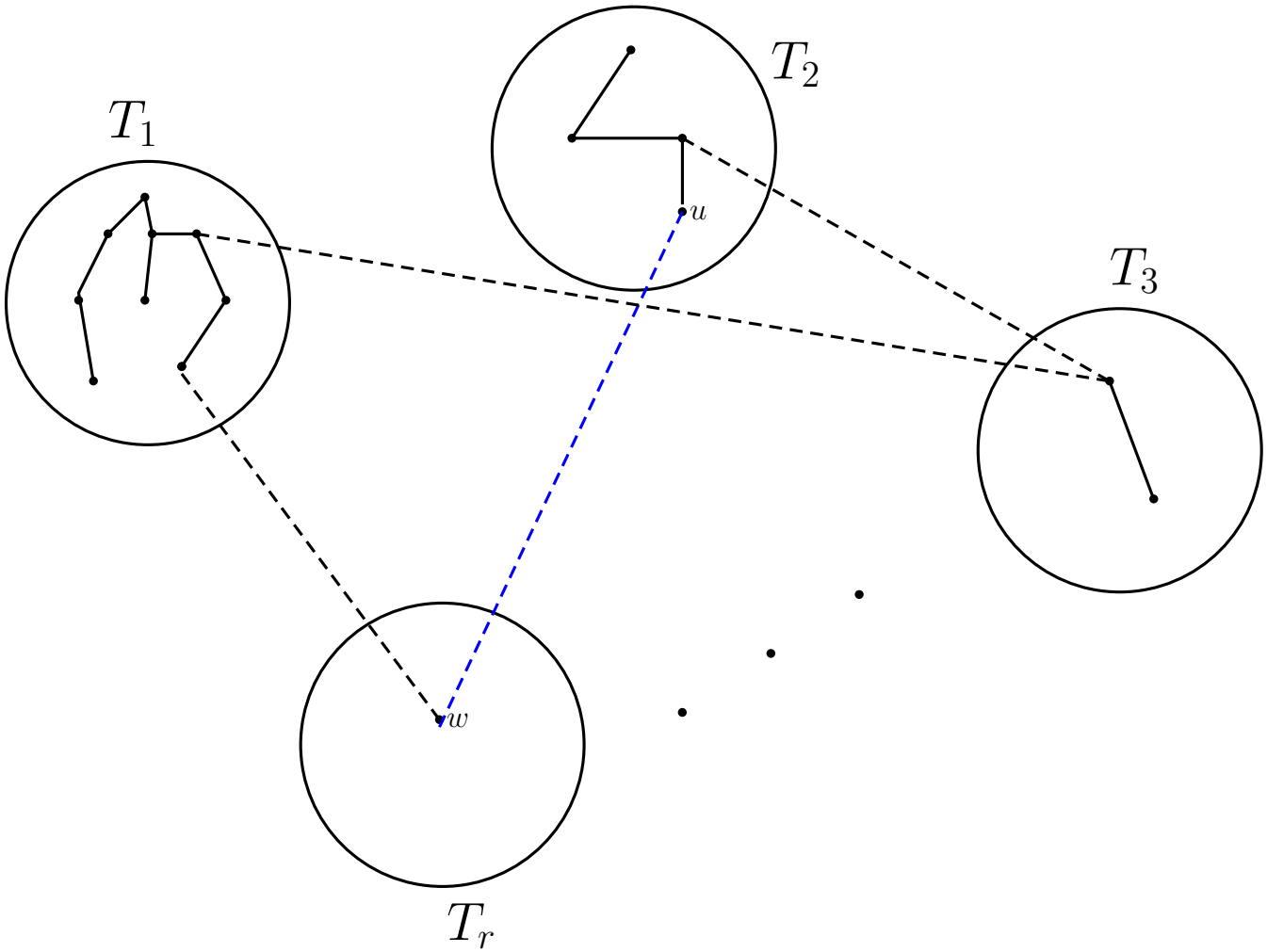


Edges whose end vertices are in the same component create cycles. See red dotted edges.

$At\ k+1^{st}\ iteration$

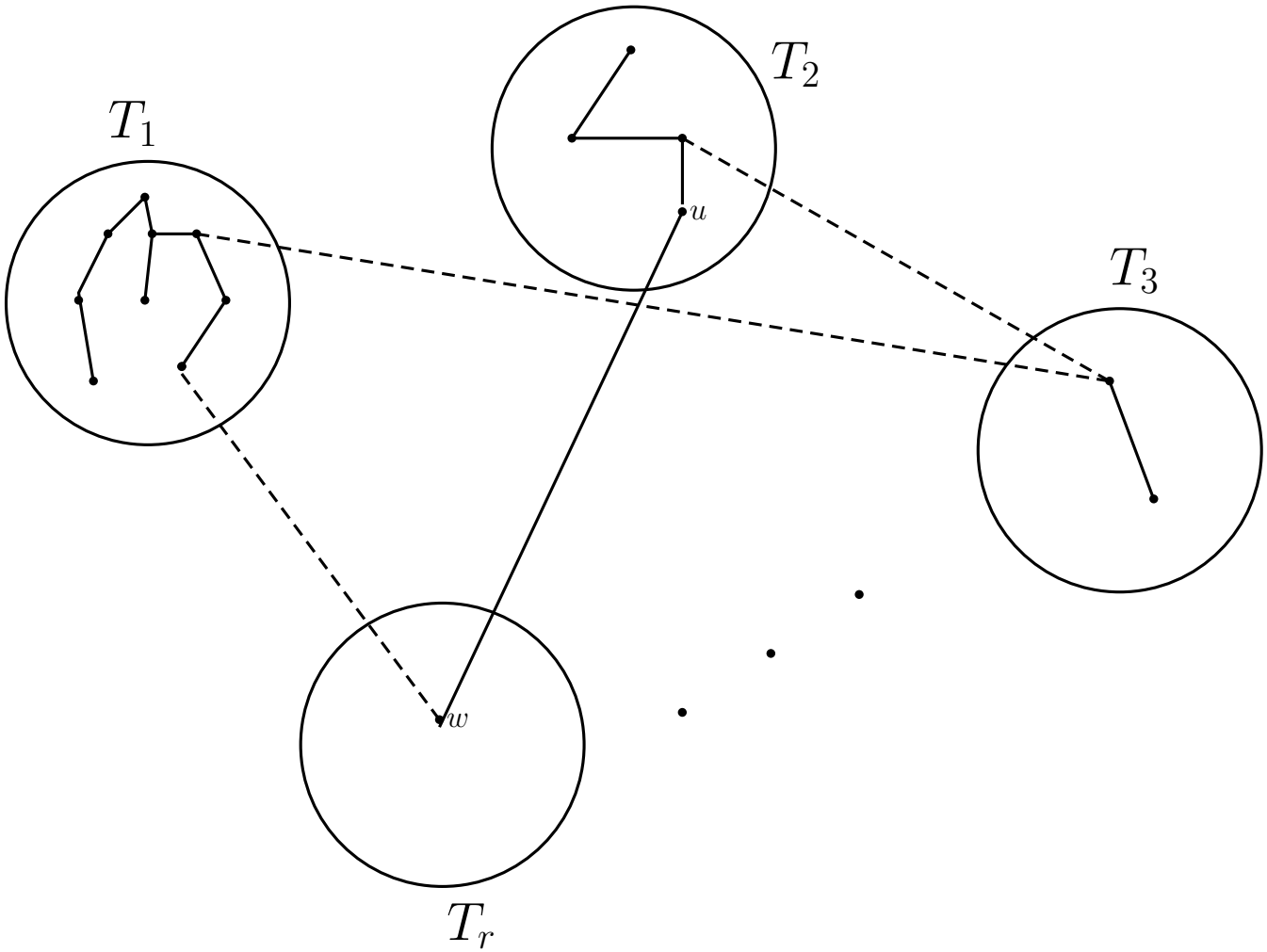
- Let E_k be the set of edges whose end vertices are in different components.
- Claim: The edge with the minimum weight (assuming unique) in E_k belongs to the MST T. Let (u, w) be that edge where $u \in V(T_i)$ and $w \in V(T_j)$ where $i \neq j$. ($V(T_l)$ denotes the set of vertices in the tree T_l).

Proof of the algorithm



The edge $(u, w) \in E_k$

Proof of the algorithm

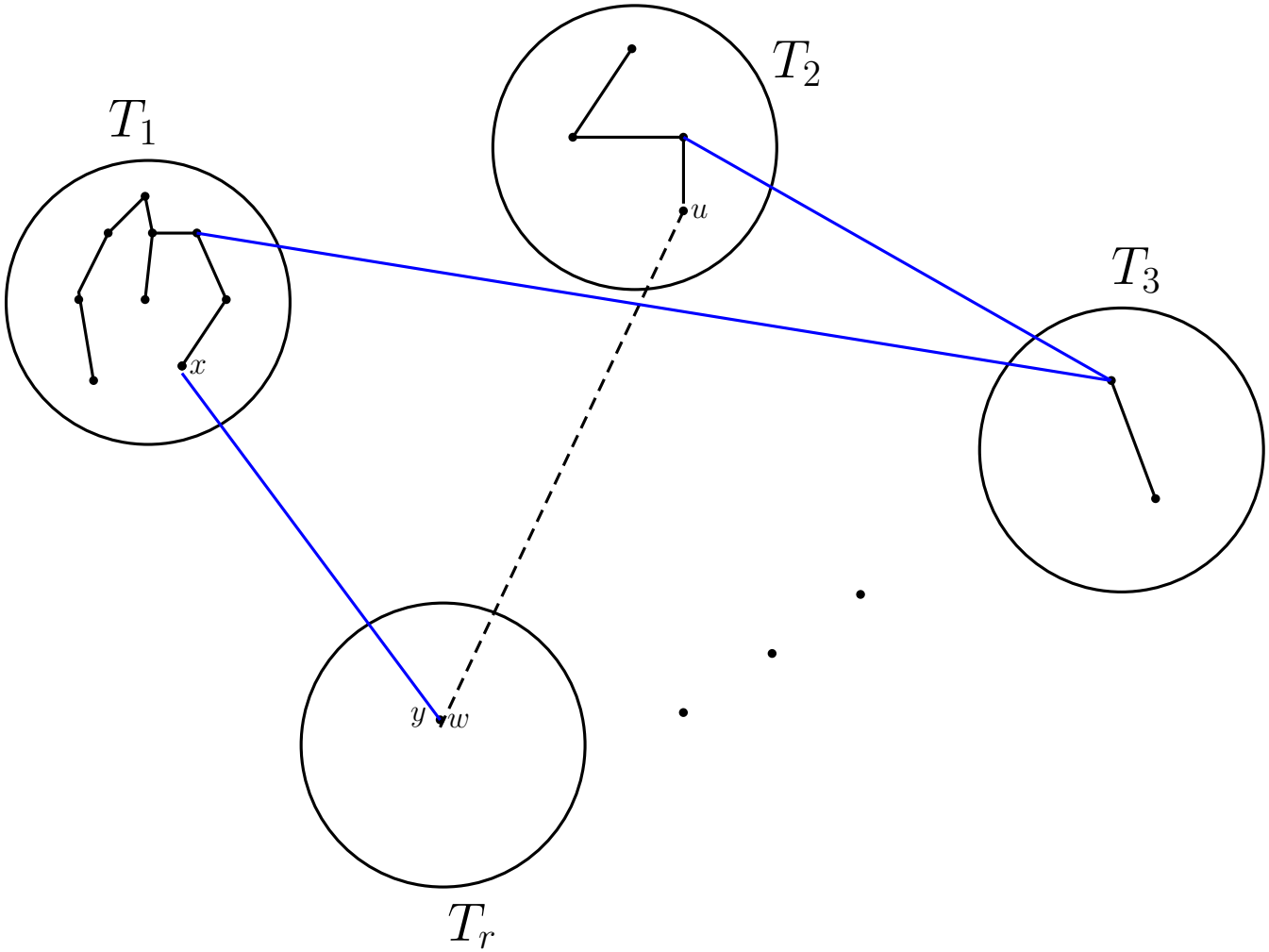


Claim: The edge (u, w) belongs to the final MST T.

$At\ k+1^{st}\ iteration$

- Let E_k be the set of edges whose end vertices are in different components.
- Claim: The edge with the minimum weight (assuming unique) in E_k belongs to the MST T. Let (u, w) be that edge where $u \in V(T_i)$ and $v \in V(T_j)$ where $i \neq j$. ($V(T_l)$ denotes the set of vertices in the tree T_l).
  - Proof by contradiction: Assume (u, w) is not in T.
  - Since T is a tree, there exist a unique path from u to w.
  - Since $u \in V(T_i)$ and $w \in V(T_j)$, there must be at least one edge (x, y) in this path that connects a vertex in T_l to a vertex in T_j (note that l may not be equal to i).

Proof of the algorithm

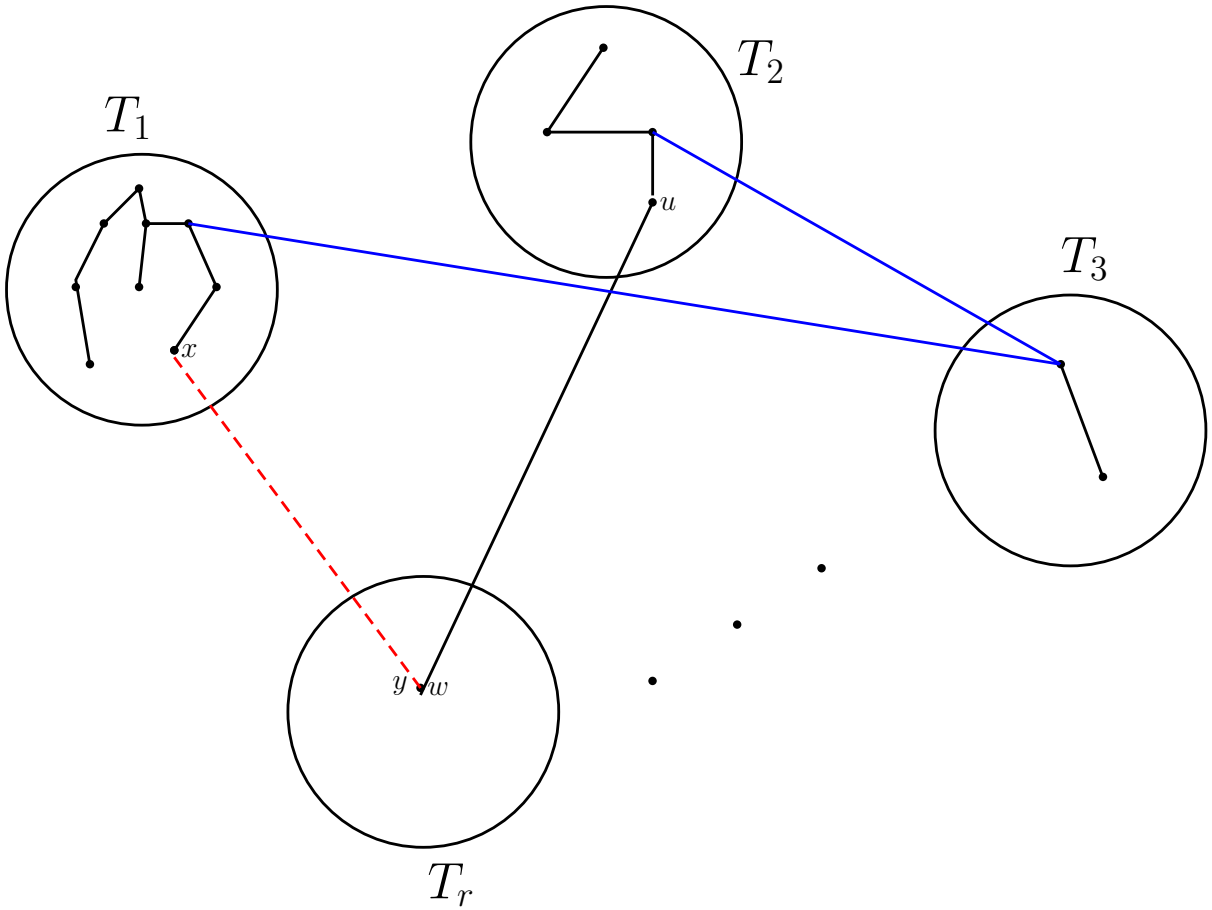


Consider a path from u to w in the final MST.

$At\ k+1^{st}\ iteration$

- Let E_k be the set of edges whose end vertices are in different components.
- Claim: The edge with the minimum weight (assuming unique) in E_k belongs to the MST T. Let (u, w) be that edge where $u \in V(T_i)$ and $v \in V(T_j)$ where $i \neq j$. ($V(T_l)$ denotes the set of vertices in the tree T_l).
  - Proof by contradiction: Assume (u, w) is not in T.
  - Since T is a tree, there exist a unique path from u to w.
  - Since $u \in V(T_i)$ and $w \in V(T_j)$, there must be at least one edge (x, y) in this path that connects a vertex in T_l to a vertex in T_j (note that l may not be equal to i).
  - We assumed that the weight of (x, y) is higher than the weight of (u, w).
  - Now, add (u, w) in T and remove (x, y) to obtain a spanning tree with a lower cost, which is a contradiction.

Proof of the algorithm



Adding (u, w) and removing (x, y) produces another spanning tree T' whose cost is lower than that of T.

Proof of the algorithm

Wait!!

The proof is not complete.

We need to prove the following:

1. Merging two components yield a tree only. (trivial ... try yourself).
2. When the algorithm terminates, it produces a tree. It will be because

- It never chooses an edge that completes a cycle.
- If the final graph has more than one component, then we considered no edge joining two of them, because such an edge would be accepted. Since G is connected, some such edge exists and we considered it.

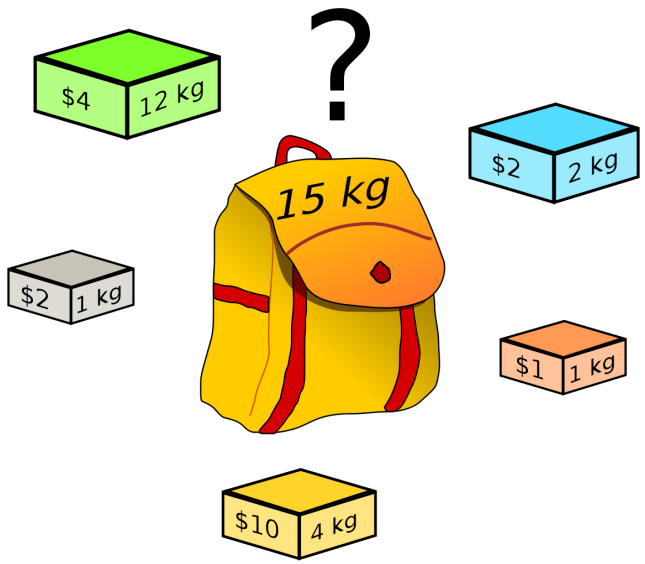
Thus the final graph is connected and acyclic, which makes it a tree.

Greedy method doesn't always provide an optimal solution.

Non-Optimal Solution by Greedy Method

Fractional Knapsack Problem:

In a Knapsack problem, we are given n objects and a knapsack or bag. Object i has a weight w_i and the knapsack has a capacity m. If a fraction x_i, $0 \leq x_i \leq 1$, of object i is placed into the knapsack, then a profit of $p_i x_i$ is earned.



Non-Optimal Solution by Greedy Method

Fractional Knapsack Problem

$$\text{Maximize } P = \sum_{i=1}^{n} p_i x_i$$

$$\text{subject to } \sum_{i=1}^{n} w_i x_i \leq m$$

$$\text{and } 0 \leq x_i \leq 1, 1 \leq i \leq n$$

The profits and weights are positive numbers.

Non-Optimal Solution by Greedy Method

Example

Consider the following instance of the knapsack problem, $n = 3, m = 20$, $(p_1, p_2, p_3) = (25, 24, 15)$, and $(w_1, w_2, w_3) = (18, 15, 10)$.

Lets solve by Greedy Approach.

1. Fill the knapsack by including next object with the largest profit.

Solution: $(x_1, x_2, x_3) = (1, 2/15, 0)$. $w_1x_1 + w_2x_2 + w_3x_3 = 20$.
$P = p_1x_1 + p_2x_2 + p_3x_3 = 28.2$.

Non-Optimal Solution by Greedy Method

Example

Consider the following instance of the knapsack problem, $n = 3, m = 20$, $(p_1, p_2, p_3) = (25, 24, 15)$, and $(w_1, w_2, w_3) = (18, 15, 10)$.

Lets be greedy again.

2. Fill the knapsack by capacity and use it up as slow as possible.

Solution: $(x_1, x_2, x_3) = (0, 2/3, 1)$. $w_1x_1 + w_2x_2 + w_3x_3 = 20$.
$P = p_1x_1 + p_2x_2 + p_3x_3 = 31$.

Optimal Solution by Greedy Method

Example

Consider the following instance of the knapsack problem, $n = 3, m = 20$, $(p_1, p_2, p_3) = (25, 24, 15)$, and $(w_1, w_2, w_3) = (18, 15, 10)$.

One more attempt. Lets strive to achieve a balance between the rate at which profit increases and the rate at which capacity is used.

$p_1/w_1 = 1.39, p_2/w_2 = 1.6, p_3/w_3 = 1.5$

3. Fill the knapsack by next object with the maximum profit per unit of capacity used.

Solution: $(x_1, x_2, x_3) = (0, 1, 1/2)$. $w_1x_1 + w_2x_2 + w_3x_3 = 20$.
$P = p_1x_1 + p_2x_2 + p_3x_3 = 31.5$.

This solution is optimal (Prove it!!).

Thank You