A Brief Introduction to Greedy Algorithms

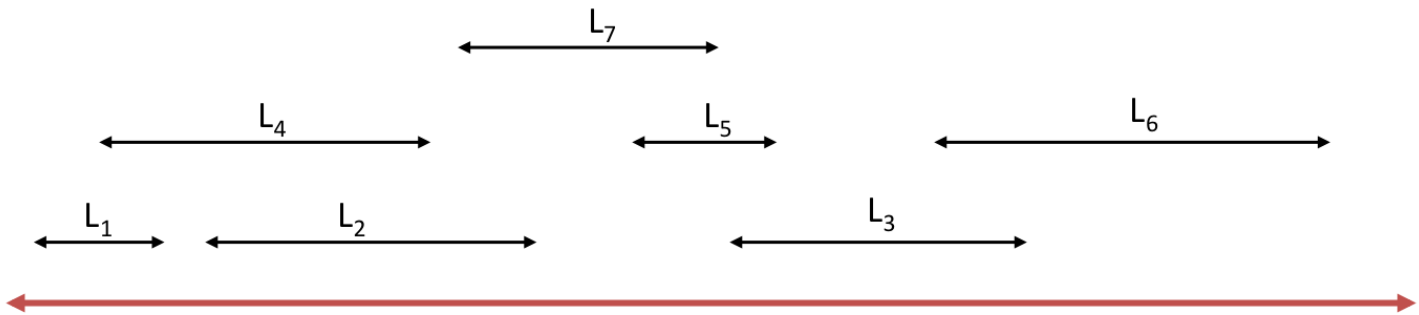
Ragesh Jaiswal, CSE, UCSD

Greedy Algorithms

- "*A local (greedy) decision rule leads to a globally optimal solution.*"

Problem

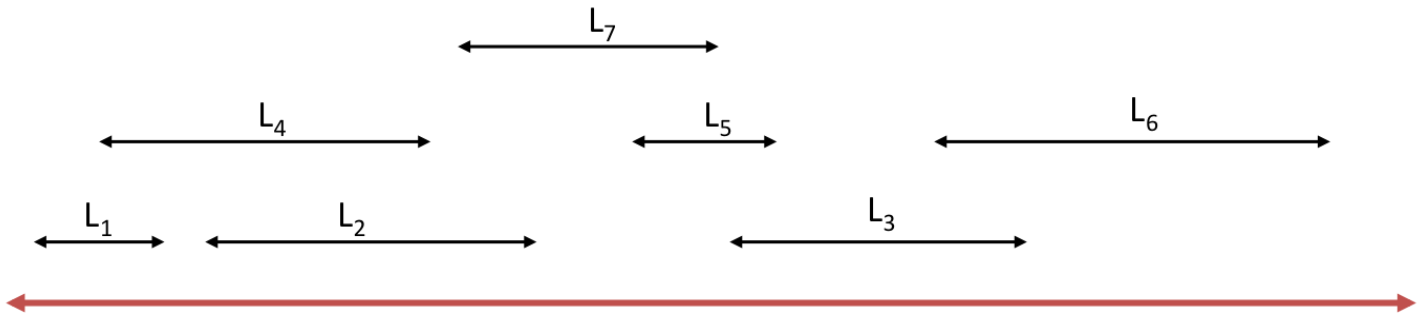
Interval scheduling: Given a set of n intervals of the form $(S(i), F(i))$, find the largest subset of non-overlapping intervals.



Problem

Interval scheduling: Given a set of n intervals of the form $(S(i), F(i))$, find the largest subset of non-overlapping intervals.

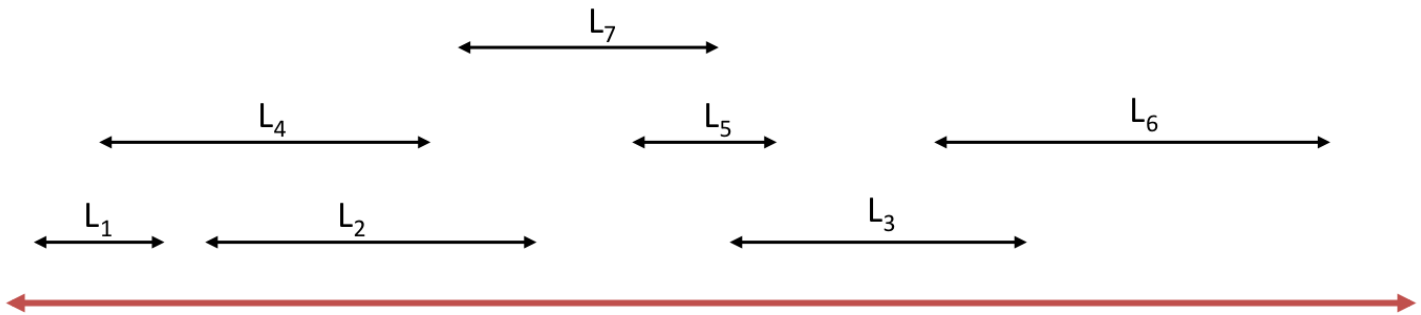
- Candidate greedy choices:
  - Earliest start time
  - Smallest duration
  - Least overlapping



Problem

Interval scheduling: Given a set of n intervals of the form $(S(i), F(i))$, find the largest subset of non-overlapping intervals.

- Candidate greedy choices:
  - Earliest start time
  - Smallest duration
  - Least overlapping
  - Earliest finish time



Problem

Interval scheduling: Given a set of n intervals of the form $(S(i), F(i))$, find the largest subset of non-overlapping intervals.

Algorithm

`GreedySchedule`

- Initialize R to contain all intervals
- While R is not empty
  - Choose an interval $(S(i), F(i))$ from R that has the smallest value of $F(i)$
  - Delete all intervals in R that overlaps with $(S(i), F(i))$

Problem

Interval scheduling: Given a set of n intervals of the form $(S(i), F(i))$, find the largest subset of non-overlapping intervals.

Algorithm

`GreedySchedule`

- Initialize R to contain all intervals
- While R is not empty
  - Choose an interval $(S(i), F(i))$ from R that has the smallest value of $F(i)$
  - Delete all intervals in R that overlaps with $(S(i), F(i))$

- Question: Let O denote some optimal subset and A by the subset given by `GreedySchedule`. Can we show that $A = O$?

- Question: Let O denote some optimal subset and A by the subset given by `GreedySchedule`. Can we show that $A = O$?
- Question Can we show that $|O| = |A|$?

- Question: Let O denote some optimal subset and A by the subset given by `GreedySchedule`. Can we show that $A = O$?
- Question Can we show that $|O| = |A|$?
- Yes we can! We will use "greedy stays ahead" method to show this.

Proof

Let $a_1, a_2, ..., a_k$ be the sequence of requests that `GreedySchedule` picks and $o_1, o_2, ..., o_l$ be the requests in O sorted in non-decreasing order by finishing time.

- Claim 1: $F(a_1) \leq F(o_1)$.

- Question: Let O denote some optimal subset and A by the subset given by `GreedySchedule`. Can we show that $A = O$?
- Question Can we show that $|O| = |A|$?
- Yes we can! We will use "greedy stays ahead" method to show this.

Proof

Let $a_1, a_2, ..., a_k$ be the sequence of requests that `GreedySchedule` picks and $o_1, o_2, ..., o_l$ be the requests in O sorted in non-decreasing order by finishing time.

- Claim 1: $F(a_1) \leq F(o_1)$.
- Claim 2: If $F(a_1) \leq F(o_1)$, $F(a_2) \leq F(o_2)$, ..., $F(a_{i-1}) \leq F(o_{i-1})$, then $F(a_i) \leq F(o_i)$.

- Question: Let O denote some optimal subset and A by the subset given by `GreedySchedule`. Can we show that $A = O$?
- Question Can we show that $|O| = |A|$?
- Yes we can! We will use "greedy stays ahead" method to show this.

Proof

- Let $a_1, a_2, ..., a_k$ be the sequence of requests that `GreedySchedule` picks and $o_1, o_2, ..., o_l$ be the requests in O sorted in non-decreasing order by finishing time.
- We will show by induction that $\forall i, F(a_i) \leq F(o_i)$
- Claim 1 (base case): $F(a_1) \leq F(o_1)$.
- Claim 2 (inductive step): If $F(a_1) \leq F(o_1)$, $F(a_2) \leq F(o_2)$, ..., $F(a_{i-1}) \leq F(o_{i-1})$, then $F(a_i) \leq F(o_i)$.
- `GreedySchedule` could not have stopped after a_k.

Problem

Interval scheduling: Given a set of n intervals of the form $(S(i), F(i))$, find the largest subset of non-overlapping intervals.

Algorithm

`GreedySchedule`

- Initialize R to contain all intervals
- While R is not empty
  - Choose an interval $(S(i), F(i))$ from R that has the smallest value of $F(i)$
  - Delete all intervals in R that overlaps with $(S(i), F(i))$

- Running time?

Problem

Interval scheduling: Given a set of n intervals of the form $(S(i), F(i))$, find the largest subset of non-overlapping intervals.

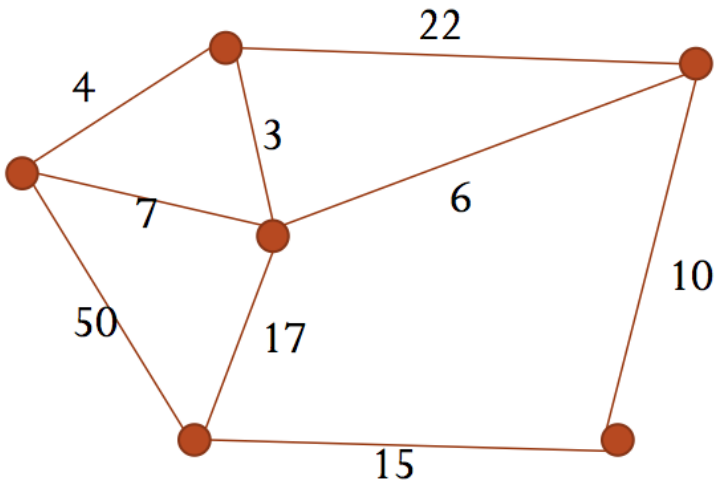
Algorithm

`GreedySchedule`

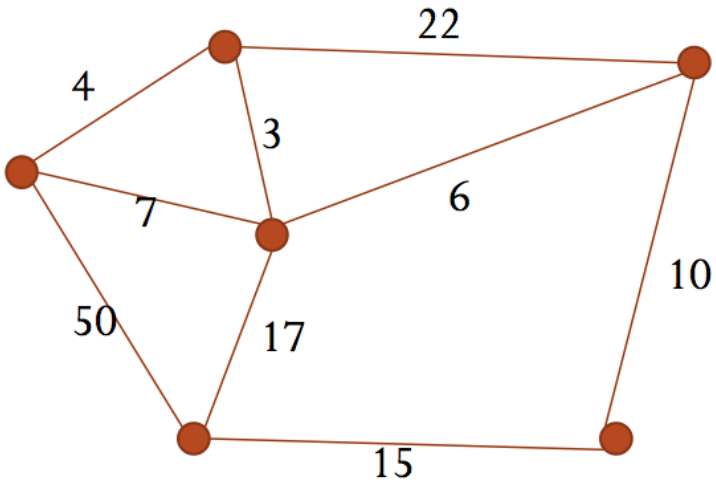
- While R is not empty
- Choose an interval $(S(i), F(i))$ from R that has the smallest value of $F(i)$
- Delete all intervals in R that overlaps with $(S(i), F(i))$

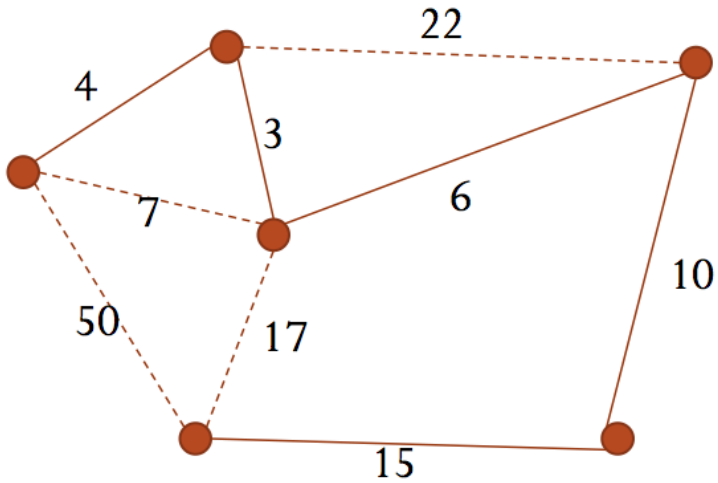
- Running time? $O(n \log n)$

- Spanning Tree: Given a strongly connected graph $G = (V, E)$, a *spanning tree* of G is a subgraph $G' = (V, E')$ such that G' is a tree.
- Minimum Spanning Tree (MST): Given a strongly connected weighted graph $G = (V, E)$, a *Minimum Spanning Tree* of G is a spanning tree of G of minimum total weight (i.e., sum of weight of edges in the tree).



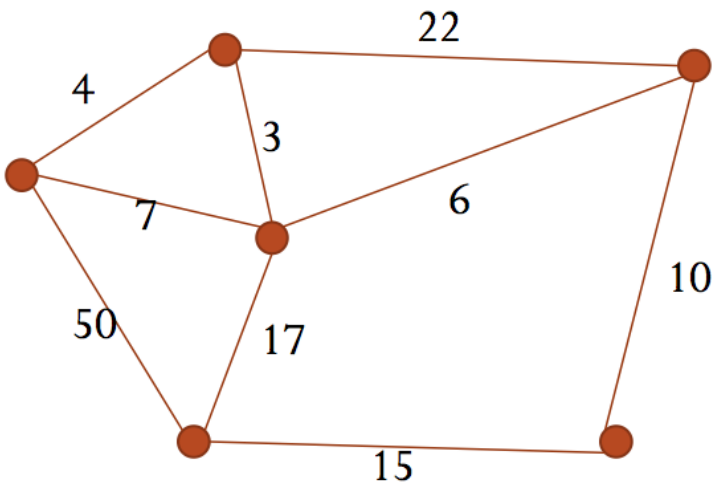
- Spanning Tree: Given a strongly connected graph $G = (V, E)$, a *spanning tree* of G is a subgraph $G' = (V, E')$ such that G' is a tree.
- Minimum Spanning Tree (MST): Given a strongly connected weighted graph $G = (V, E)$, a *Minimum Spanning Tree* of G is a spanning tree of G of minimum total weight (i.e., sum of weight of edges in the tree).

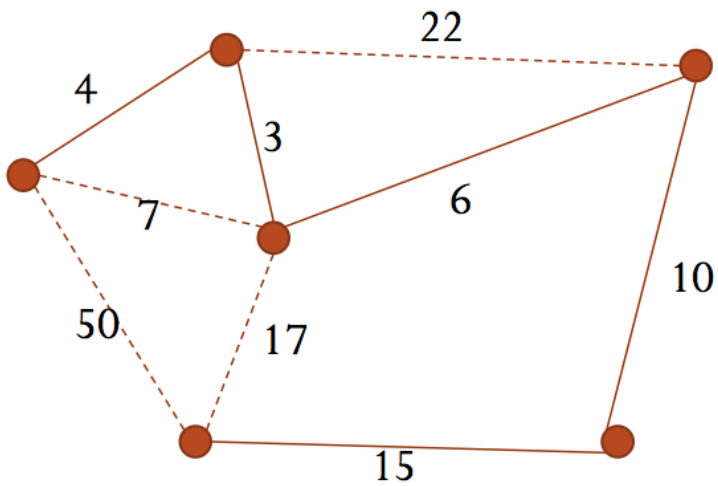




Problem

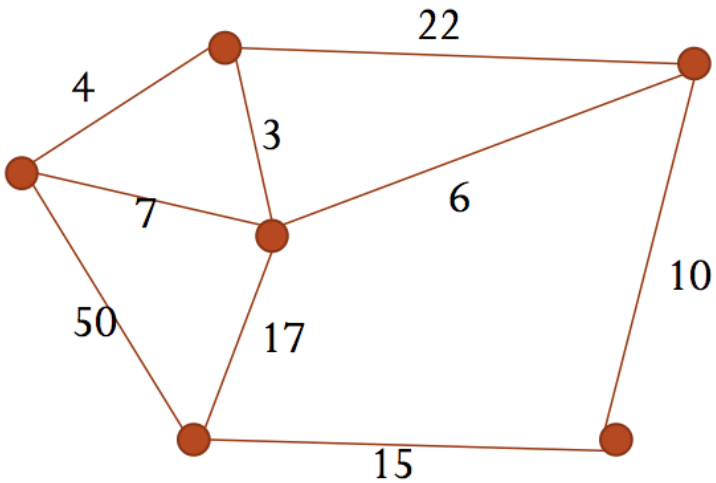
Given a weighted graph G where all the edge weights are distinct, give an algorithm for finding the MST of G.

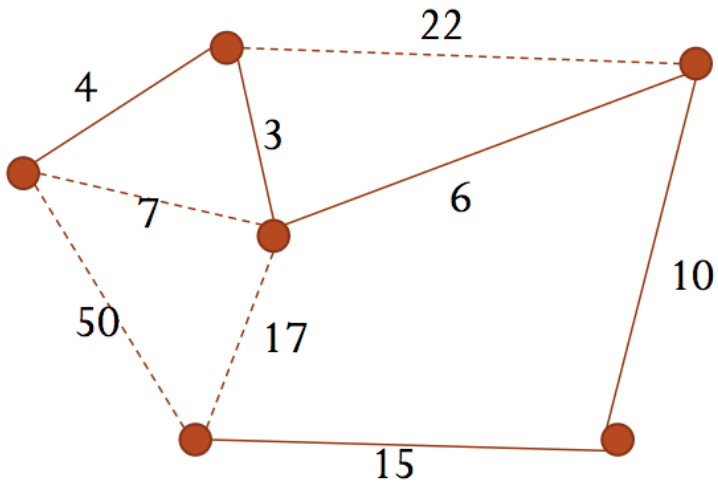




Theorem

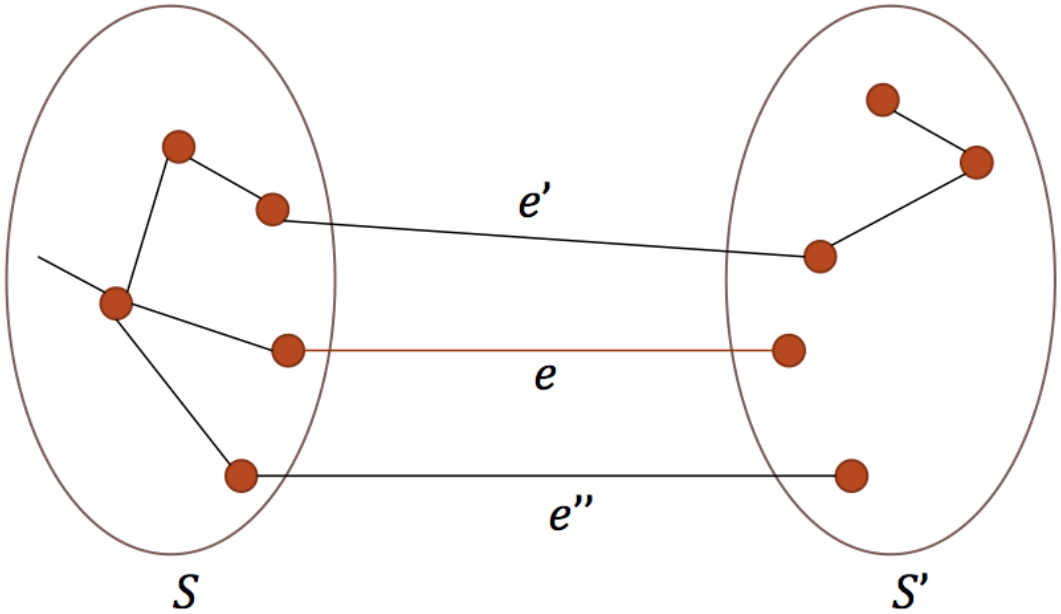
Cut property: Given a weighted graph $G = (V, E)$ where all the edge weights are distinct. Consider a non-empty proper subset S of V and $S' = V \setminus S$. Let e be the least weighted edge between any pair of vertices (u, v), where u is in S and v is in S'. Then e is necessarily present in *all* MSTs of G.





Theorem

Cut property: Given a weighted graph $G = (V, E)$ where all the edge weights are distinct. Consider a non-empty proper subset S of V and $S' = V \setminus S$. Let e be the least weighted edge between any pair of vertices (u, v), where u is in S and v is in S'. Then e is necessarily present in *all* MSTs of G.



Algorithm

`Prim's Algorithm(G)`

- $S \leftarrow \{u\}$ //*u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$

Algorithm

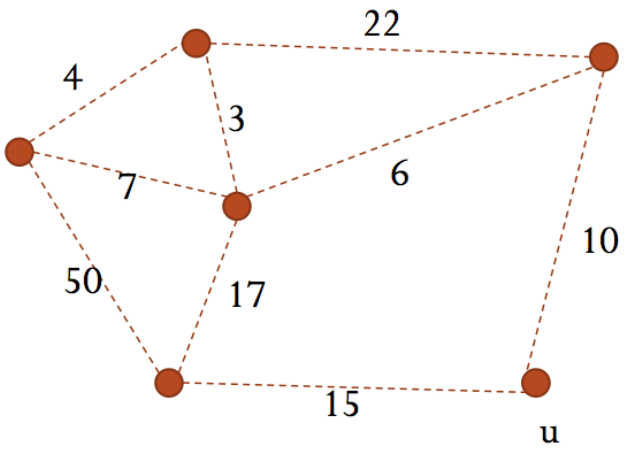
`Kruskal's Algorithm(G)`

- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$

Algorithm

`Prim's Algorithm(G)`

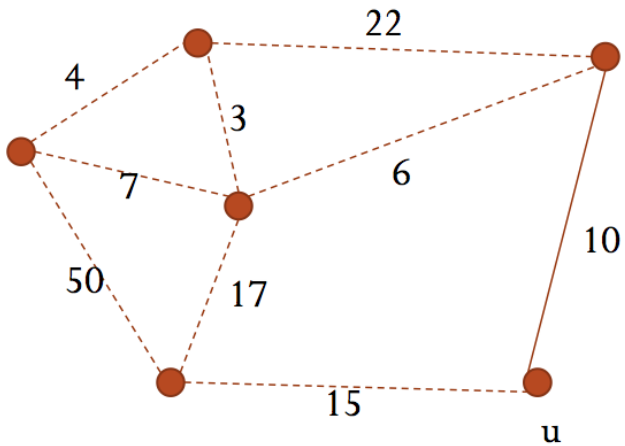
- $S \leftarrow \{u\}$ *//u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$



Algorithm

`Prim's Algorithm(G)`

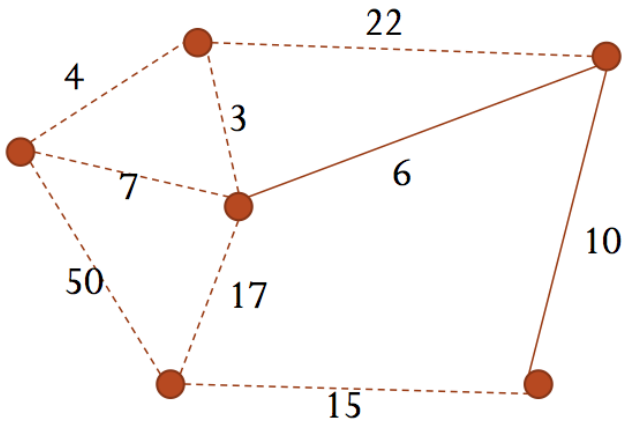
- $S \leftarrow \{u\}$ *//u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$



Algorithm

`Prim's Algorithm(G)`

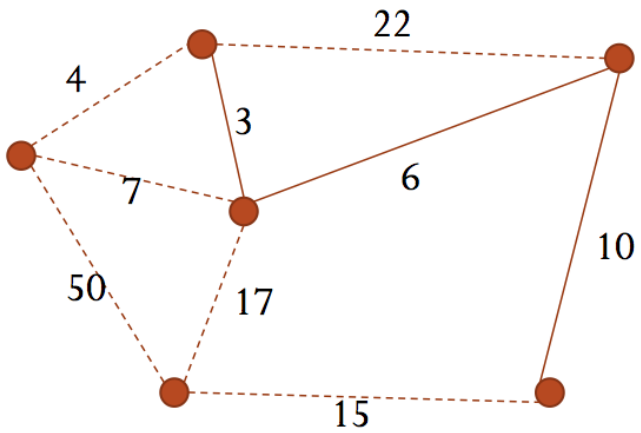
- $S \leftarrow \{u\}$ *//u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$



Algorithm

`Prim's Algorithm(G)`

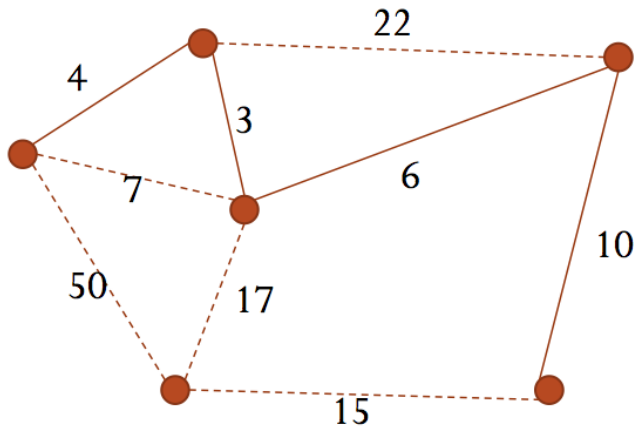
- $S \leftarrow \{u\}$ *//u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$



Algorithm

`Prim's Algorithm(G)`

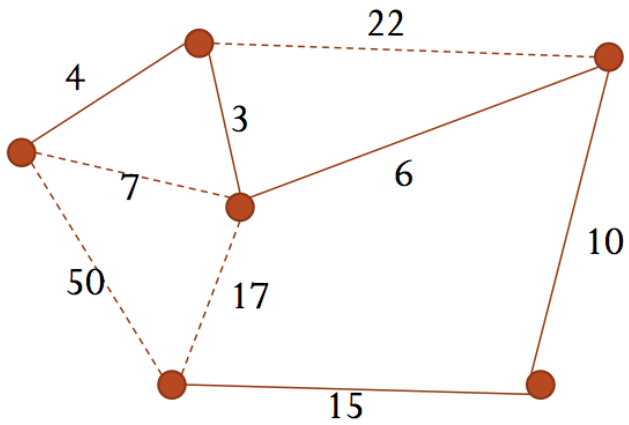
- $S \leftarrow \{u\}$ //*u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$



Algorithm

`Prim's Algorithm(G)`

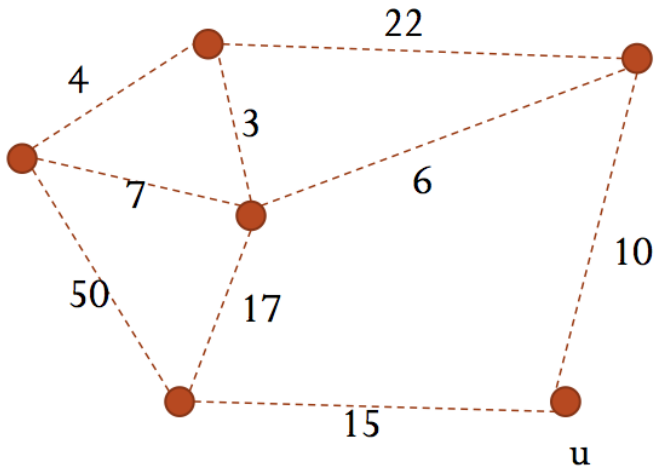
- $S \leftarrow \{u\}$ *//u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$



Algorithm

`Kruskal's Algorithm(G)`

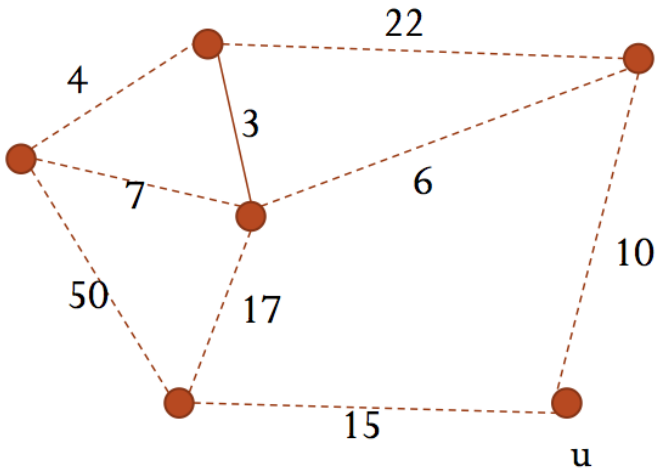
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$



Algorithm

`Kruskal's Algorithm(G)`

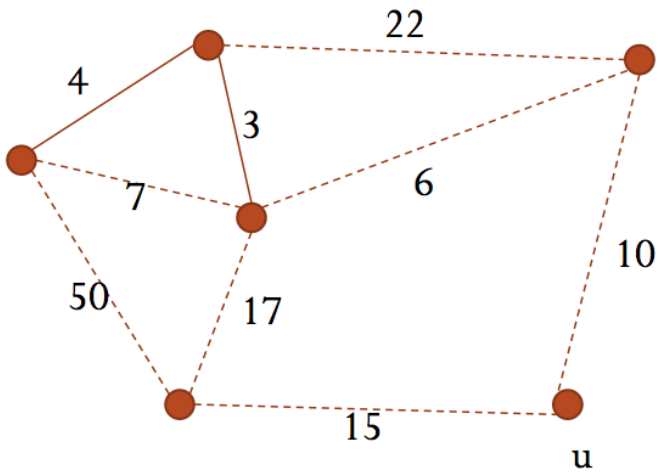
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$



Algorithm

`Kruskal's Algorithm(G)`

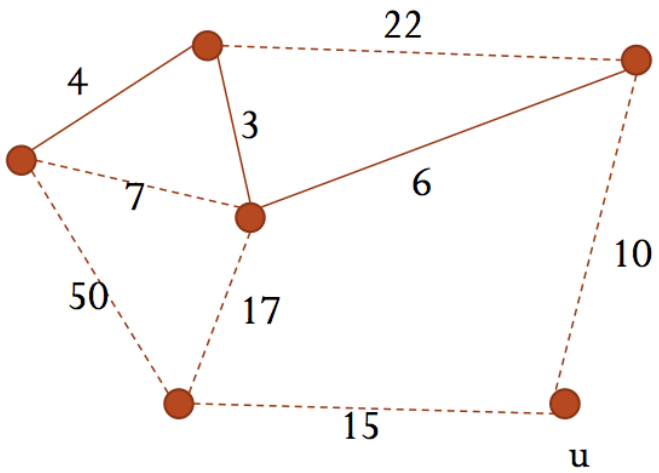
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$



Algorithm

`Kruskal's Algorithm(G)`

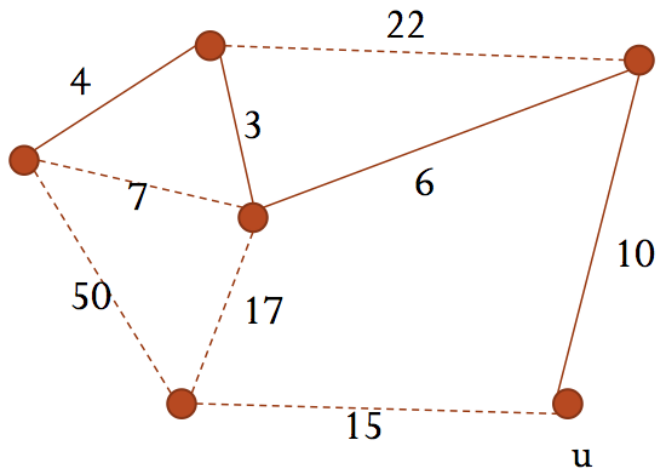
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$



Algorithm

`Kruskal's Algorithm(G)`

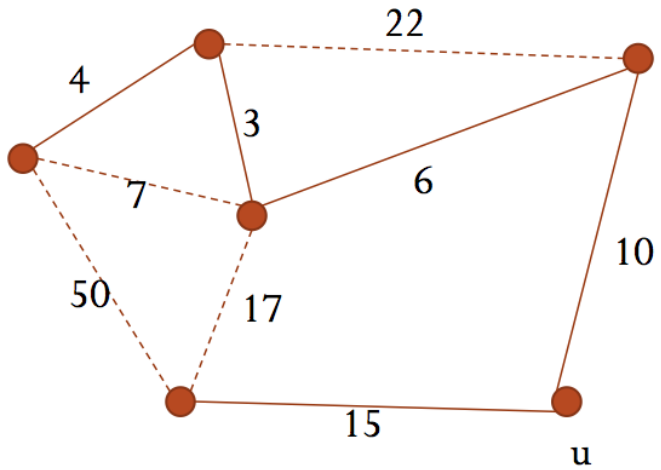
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$



Algorithm

`Kruskal's Algorithm(G)`

- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$



Algorithm

`Prim's Algorithm(G)`

- $S \leftarrow \{u\}$ //*u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$

- What is the running time of Prim's algorithm?

Algorithm

Prim's Algorithm(G)

- $S \leftarrow \{u\}$ //*u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$

- What is the running time of Prim's algorithm? $O(|E| \cdot \log |V|)$

Algorithm

`Kruskal's Algorithm(G)`

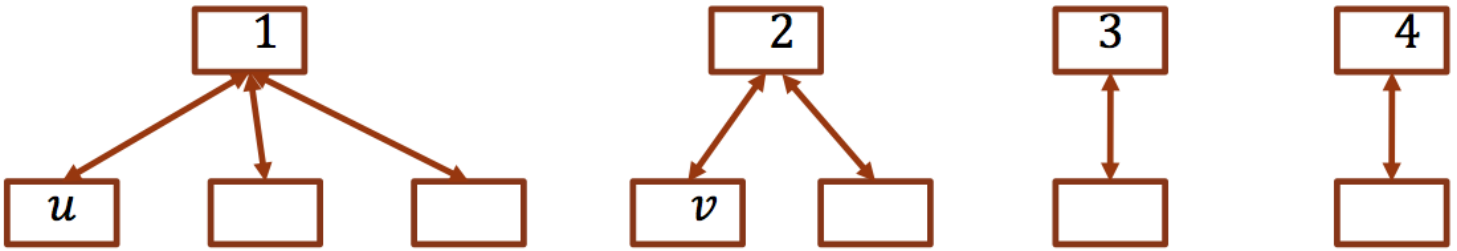
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$

Algorithm

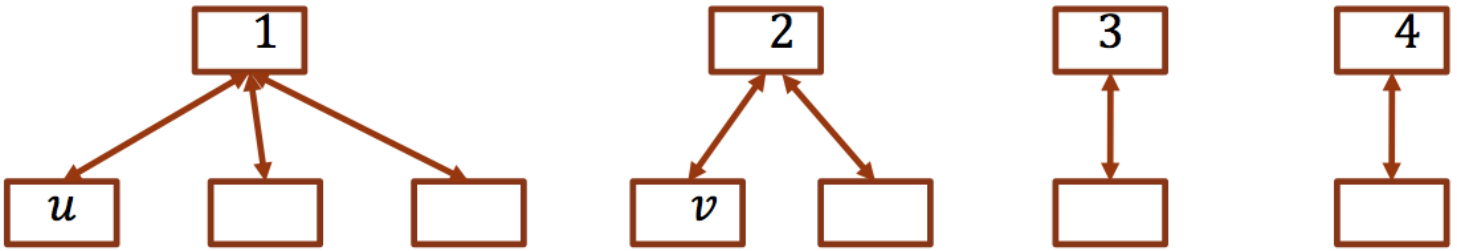
`Kruskal's Algorithm(G)`

- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - *//Note that $G' = (V, T)$ contains dicsonnected components*
  - Let e be the minimum weight edge in the set S
  - ~~If e does not create a cycle in T~~
  - If u and v are in different components of G'
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$

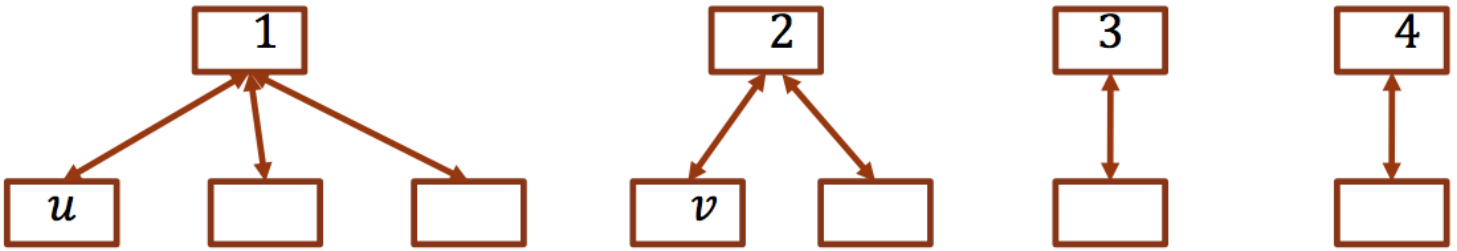
- Union-Find: Used for storing partition of a set of elements. The following two operations are supported:
  1. *Find*(v): Find the partition to which the element v belongs.
  2. *Union*(u, v): Merge the partition to which u belongs with the partition to which v belongs.
- Consider the following data structure.



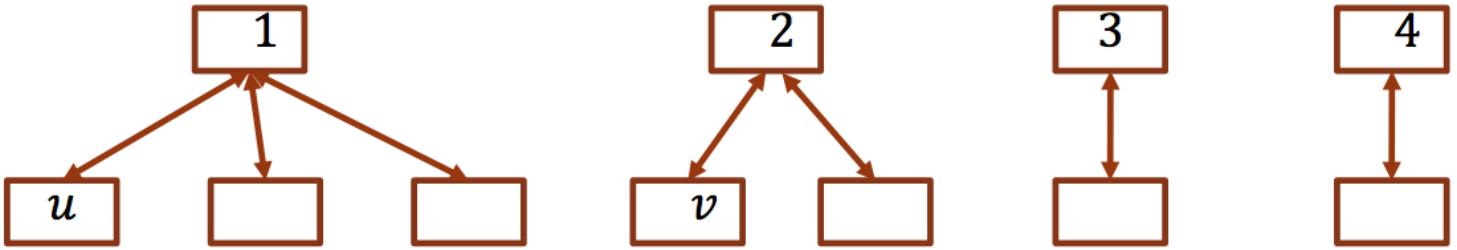
- Suppose we start from a full partition (i.e., each partition contains one element).
- How much time does the following operation take:
  - *Find*(v):
  - *Union*(u, v):



- Suppose we start from a full partition (i.e., each partition contains one element).
- How much time does the following operation take:
  - *Find*(v): $O(1)$
  - *Union*(u, v):



- Suppose we start from a full partition (i.e., each partition contains one element).
- How much time does the following operation take:
  - *Find*(v): $O(1)$
  - *Union*(u, v):
    - Claim: Performing k union operations takes $O(k \log k)$ time in the worst case.



Algorithm

`Prim's Algorithm(G)`

- $S \leftarrow \{u\}$ //*u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$

- What is the running time of Prim's algorithm?

Algorithm

Prim's Algorithm(G)

- $S \leftarrow \{u\}$ //*u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$

- What is the running time of Prim's algorithm? $O(|E| \cdot \log |V|)$

Algorithm

`Kruskal's Algorithm(G)`

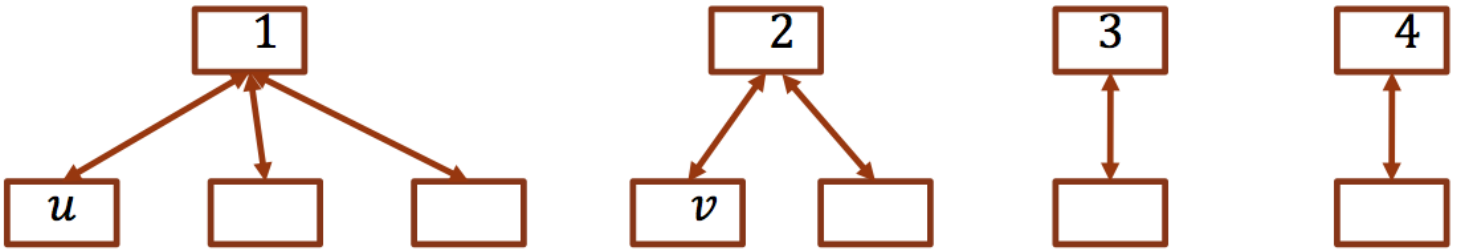
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$

Algorithm

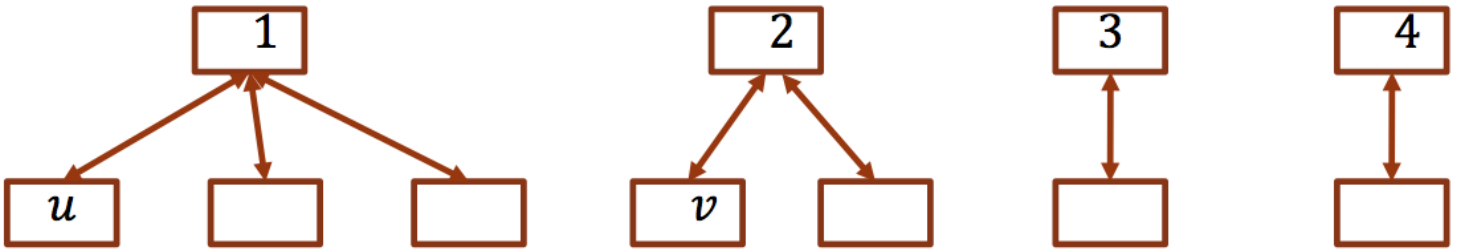
`Kruskal's Algorithm(G)`

- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - *//Note that $G' = (V, T)$ contains dicsonnected components*
  - Let e be the minimum weight edge in the set S
  - ~~If e does not create a cycle in T~~
  - If u and v are in different components of G'
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$

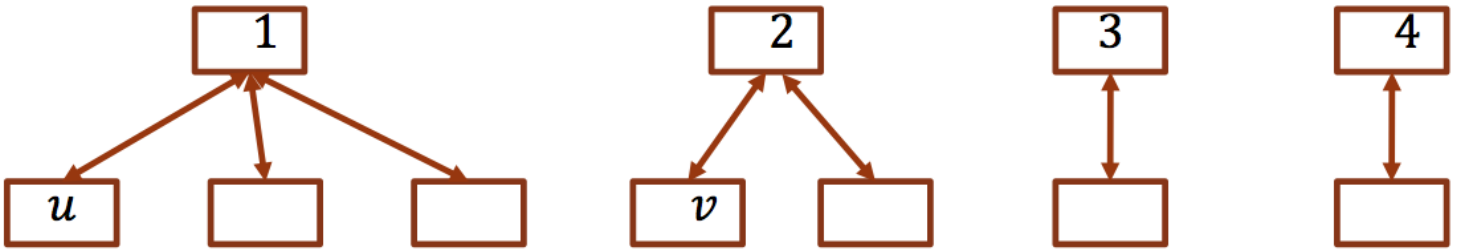
- Union-Finds: Used for storing partition of a set of elements. The following two operations are supported:
  1. *Find*(v): Find the partition to which the element v belongs.
  2. *Union*(u, v): Merge the partition to which u belongs with the partition to which v belongs.
- Consider the following data structure.



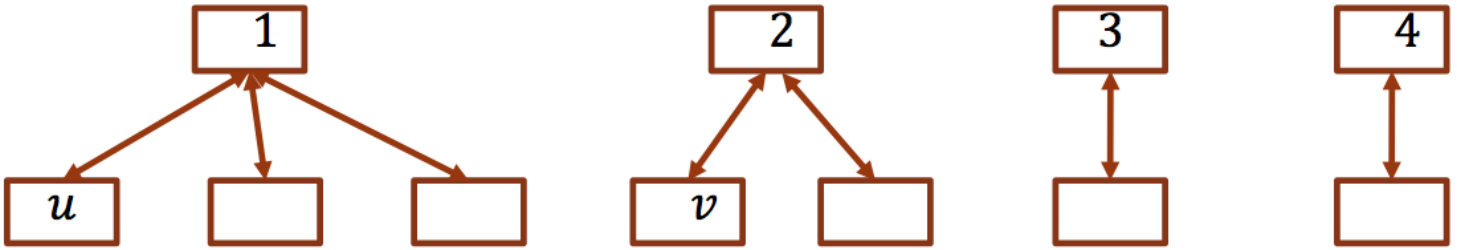
- Suppose we start from a full partition (i.e., each partition contains one element).
- How much time does the following operation take:
  - *Find*(v):
  - *Union*(u, v):



- Suppose we start from a full partition (i.e., each partition contains one element).
- How much time does the following operation take:
  - *Find*(v): $O(1)$
  - *Union*(u, v):



- Suppose we start from a full partition (i.e., each partition contains one element).
- How much time does the following operation take:
  - *Find*(v): $O(1)$
  - *Union*(u, v):
    - Claim: Performing k union operations takes $O(k \log k)$ time in the worst case.



- Kruskal's algorithm using Union-Find.

Algorithm

```
Kruskal's Algorithm(G)
```
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - *//Note that $G' = (V, T)$ contains dicsonnected components*
  - Let e be the minimum weight edge in the set S
  - ~~If e does not create a cycle in T~~
  - ~~If u and v are in different components of G'~~
  - If $(Find(u) \neq Find(v))$
    - $T \leftarrow T \cup \{e\}$
    - $Union(u, v)$
  - $S \leftarrow S \setminus \{e\}$

- What is the running time of the above algorithm?

- Kruskal's algorithm using Union-Find.

Algorithm

Kruskal's Algorithm(G)
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - //*Note that* $G' = (V, T)$ *contains dicsonnected components*
  - Let e be the minimum weight edge in the set S
  - ~~If e does not create a cycle in T~~
  - ~~If u and v are in different components of G'~~
  - If $(Find(u) \neq Find(v))$
    - $T \leftarrow T \cup \{e\}$
    - $Union(u, v)$
  - $S \leftarrow S \setminus \{e\}$

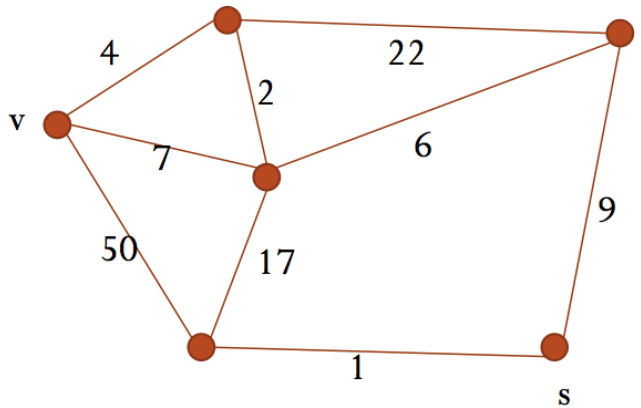
- What is the running time of the above algorithm? $O(|E| \cdot \log |V|)$

- Path length: Let $G = (V, E)$ be a weighted directed graph. Given a path in G, the length of a path is defined to be the sum of lengths of the edges in the path.
- Shortest path: The shortest path from u to v is the path with minimum length.

- Path length: Let $G = (V, E)$ be a weighted directed graph. Given a path in G, the length of a path is defined to be the sum of lengths of the edges in the path.
- Shortest path: The shortest path from u to v is the path with minimum length.

Problem

Single source shortest path: Given a weighted, directed graph $G = (V, E)$ with positive edge weights and a source vertex s, find the shortest path from s to all other vertices in the graph.



Problem

Single source shortest path: Given a weighted, directed graph $G = (V, E)$ with positive edge weights and a source vertex s, find the shortest path from s to all other vertices in the graph.

- Claim 1: Shortest path is a *simple* path.

Problem

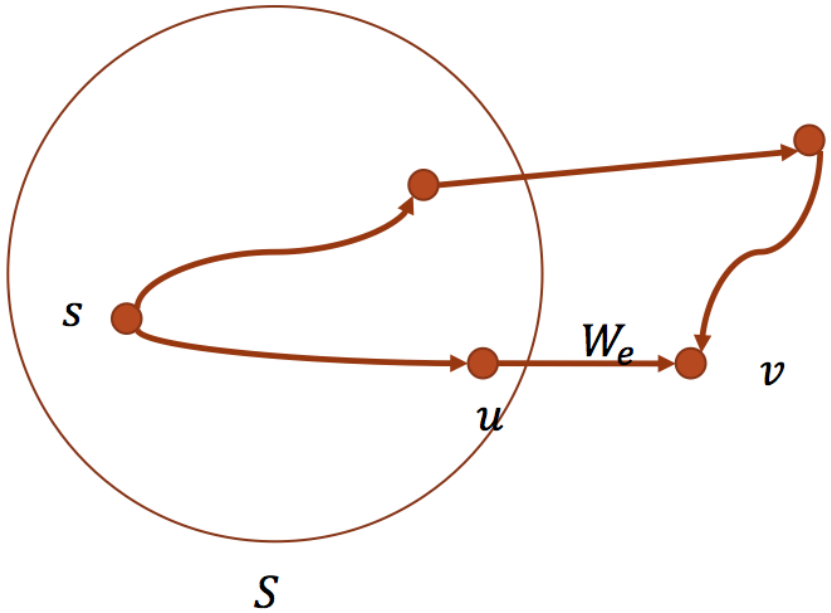
Single source shortest path: Given a weighted, directed graph with positive edge weights $G = (V, E)$ and a source vertex s, find the shortest path from s to all other vertices in the graph.

- Claim 1: Shortest path is a *simple* path.
- Claim 2: Let S be a subset of vertices containing s such that we know the shortest path length $l(s, u)$ from s to any vertex in $u \in S$. Let $e = (u, v)$ be an edge such that
  1. $u \in S$, $v \in V \setminus S$,
  2. $(l(s, u) + W_e)$ is the least among all such cut edges.

  Then $l(s, v) = l(s, u) + W_e$.

- Claim 2: Let S be a subset of vertices containing s such that we know the shortest path length $l(s,u)$ from s to any vertex in $u \in S$. Let $e = (u,v)$ be an edge such that
  1. $u \in S$, $v \in V \setminus S$,
  2. $(l(s,u) + W_e)$ is the least among all such cut edges.

  Then $l(s,v) = l(s,u) + W_e$.



- Claim 2: Let S be a subset of vertices containing s such that we know the shortest path length $l(s,u)$ from s to any vertex in $u \in S$. Let $e = (u,v)$ be an edge such that
  1. $u \in S$, $v \in V \setminus S$,
  2. $(l(s,u) + W_e)$ is the least among all such cut edges.

  Then $l(s,v) = l(s,u) + W_e$.

Algorithm

`Dijkstra's Algorithm(`G, s`)`

- $S \leftarrow \{s\}$
- $d(s) \leftarrow 0$
- While S does not contain all vertices in G
  - Let $e = (u,v)$ be a cut edge across $(S, V \setminus S)$ with minimum value of $d(u) + W_e$
  - $d(v) \leftarrow d(u) + W_e$
  - $S \leftarrow S \cup \{v\}$

- Claim 2: Let S be a subset of vertices containing s such that we know the shortest path length $l(s,u)$ from s to any vertex in $u \in S$. Let $e = (u,v)$ be an edge such that
  1. $u \in S$, $v \in V \setminus S$,
  2. $(l(s,u) + W_e)$ is the least among all such cut edges.

  Then $l(s,v) = l(s,u) + W_e$.

Algorithm

`Dijkstra's Algorithm`(G,s)
- $S \leftarrow \{s\}$
- $d(s) \leftarrow 0$
- While S does not contain all vertices in G
  - Let $e = (u,v)$ be a cut edge across $(S, V \setminus S)$ with minimum value of $d(u) + W_e$
  - $d(v) \leftarrow d(u) + W_e$
  - $S \leftarrow S \cup \{v\}$

- What is the running time of the above algorithm?

Greedy (Approximation) Algorithms

Greedy (Approximation) Algorithms

- For some problems, even though the greedy strategy does not give an optimal solution but it might give a solution that is provably close to the optimal solution.

- Covering set: Let S be a set containing n elements. A set of subsets $\{S_1, ..., S_m\}$ of S is called a covering set if each element in S is present in at least one of the subsets $S_1, ..., S_m$.

Problem

Set Cover: Given a set S containing n elements and m subsets $S_1, ..., S_m$ of S. Find a covering set of S of minimum cardinality.

- Example
  - $S = \{a, b, c, d, e, f\}$
  - $S_1 = \{a, b\}$, $S_2 = \{a, c\}$, $S_3 = \{b, c\}$, $S_4 = \{d, e, f\}$, $S_5 = \{e, f\}$
  - $\{S_1, S_2, S_3, S_4\}$ is a covering set.
  - $\{S_1, S_2, S_4\}$ is a covering set of minimum cardinality.

Problem

Set Cover: Given a set S containing n elements and m subsets $S_1, ..., S_m$ of S. Find a covering set of S of minimum cardinality.

- Application: There are n villages and the government is trying to figure out which villages to open schools at so that it has to open minimum number of schools. The constraint is that no children should have to walk more than 3 miles to get to a school.

Problem

Set Cover: Given a set S containing n elements and m subsets $S_1, ..., S_m$ of S. Find a covering set of S of minimum cardinality.

- Greedy strategy: Give preference to the subsets that covers the most number of (remaining) elements.

Algorithm

`GreedySetCover`$(S, S_1, ..., S_m)$

- $T \leftarrow \{\}$; $R \leftarrow S$
- While R is not empty:
  - Pick a subset S_i that covers the maximum number of elements in R
  - $T \leftarrow T \cup \{S_i\}$; $R \leftarrow R - S_i$

Problem

Set Cover: Given a set S containing n elements and m subsets $S_1, ..., S_m$ of S. Find a covering set of S of minimum cardinality.

- Greedy strategy: Give preference to the subsets that covers the most number of (remaining) elements.

Algorithm

`GreedySetCover`$(S, S_1, ..., S_m)$
- $T \leftarrow \{\}$; $R \leftarrow S$
- While R is not empty:
  - Pick a subset S_i that covers the maximum number of elements in R
  - $T \leftarrow T \cup \{S_i\}$; $R \leftarrow R - S_i$

- Counterexample: $S = \{a, b, c, d, e, f, g, h\}, S_1 = \{a, b, c, d, e\}, S_2 = \{a, b, c, f\}, S_3 = \{d, e, g, h\}$.

Algorithm

`GreedySetCover`$(S, S_1, ..., S_m)$

- $T \leftarrow \{\}$; $R \leftarrow S$
- While R is not empty:
  - Pick a subset S_i that covers the maximum number of elements in R
  - $T \leftarrow T \cup \{S_i\}$; $R \leftarrow R - S_i$

- Claim 1: Let k be the cardinality of any optimal covering set. Then the greedy algorithm outputs a covering set with cardinality at most $k \cdot \ln n$.

Algorithm

`GreedySetCover`$(S, S_1, ..., S_m)$

- $T \leftarrow \{\}$; $R \leftarrow S$
- While R is not empty:
  - Pick a subset S_i that covers the maximum number of elements in R
  - $T \leftarrow T \cup \{S_i\}$; $R \leftarrow R - S_i$

- Claim 1: Let k be the cardinality of any optimal covering set. Then the greedy algorithm outputs a covering set with cardinality at most $k \cdot \ln n$.

Proof of Claim 1

- Let N_t be the number of uncovered elements after t iterations of the loop.
- Claim 1.1: $N_t \leq (1 - 1/k) \cdot N_{t-1}$.

Algorithm

`GreedySetCover`$(S, S_1, ..., S_m)$

- $T \leftarrow \{\}$; $R \leftarrow S$
- While R is not empty:
  - Pick a subset S_i that covers the maximum number of elements in R
  - $T \leftarrow T \cup \{S_i\}$; $R \leftarrow R - S_i$

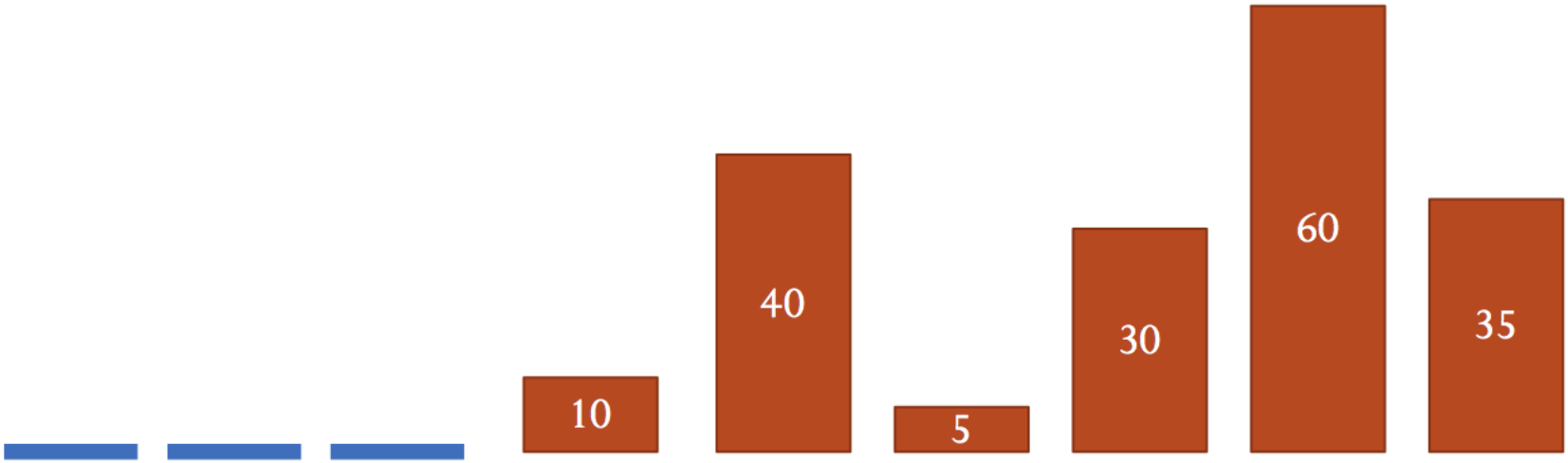
- Claim 1: Let k be the cardinality of any optimal covering set. Then the greedy algorithm outputs a covering set with cardinality at most $k \cdot \ln n$.

Proof of Claim 1

- Let N_t be the number of uncovered elements after t iterations of the loop.
- Claim 1.1: $N_t \leq (1 - 1/k) \cdot N_{t-1}$.
- Claim 1.2: $N_{k \cdot \ln n} < 1$.
  - Use the fact that $(1 - x) \leq e^{-x}$ and the equality holds only for $x = 0$.

Problem

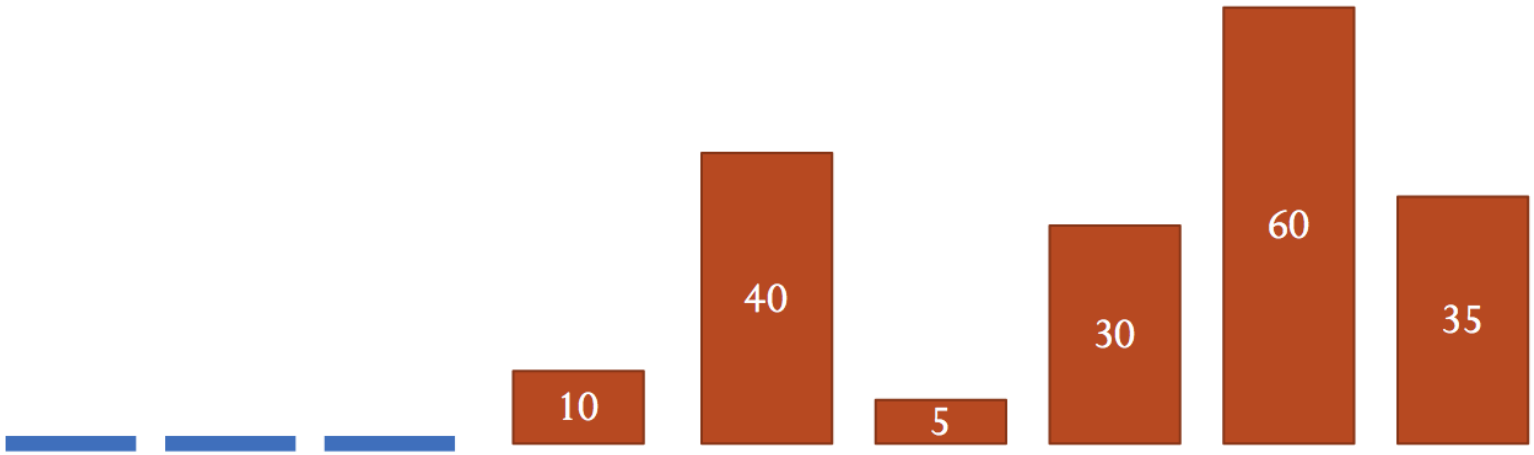
Minimum Makespan: You have m identical machines and n jobs. For each job i, you are given the duration of this job $d(i)$ that denotes the time that is required by any machine to perform this job. Assign these n jobs on m machine such that the maximum finishing time is minimized.



Problem

Minimum Makespan: You have m identical machines and n jobs. For each job i, you are given the duration of this job $d(i)$ that denotes the time that is required by any machine to perform this job. Assign these n jobs on m machine such that the maximum finishing time is minimized.

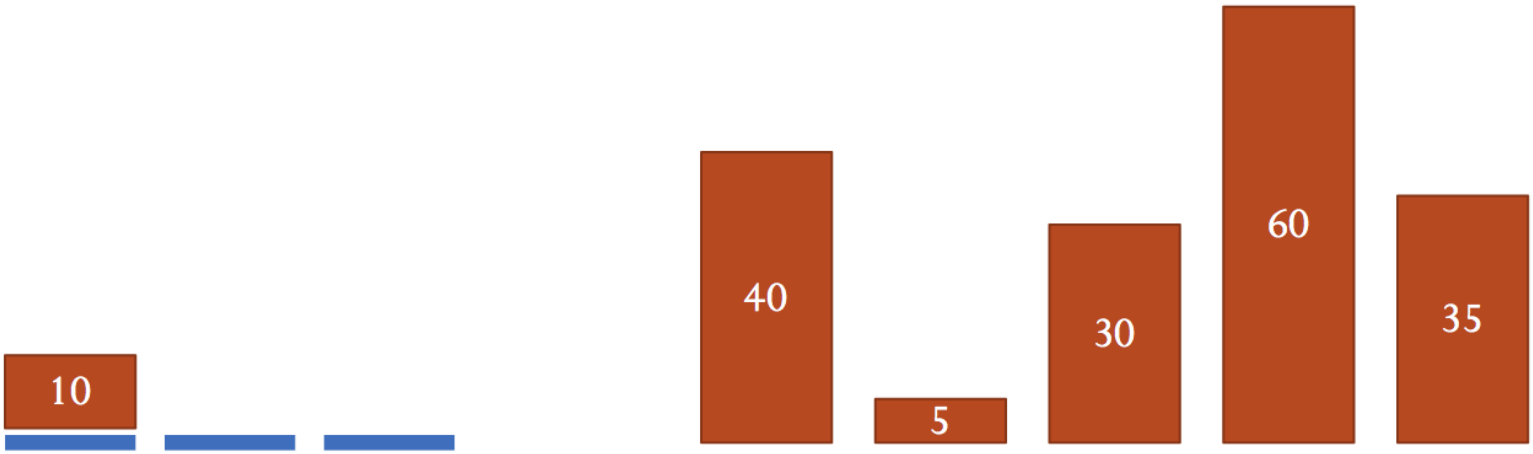
- Greedy strategy: Assign the next job to a machine with least load.



Problem

Minimum Makespan: You have m identical machines and n jobs. For each job i, you are given the duration of this job $d(i)$ that denotes the time that is required by any machine to perform this job. Assign these n jobs on m machine such that the maximum finishing time is minimized.

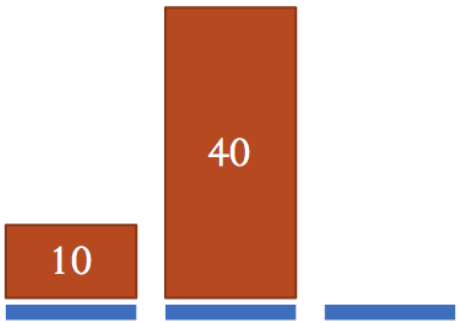
- Greedy strategy: Assign the next job to a machine with least load.

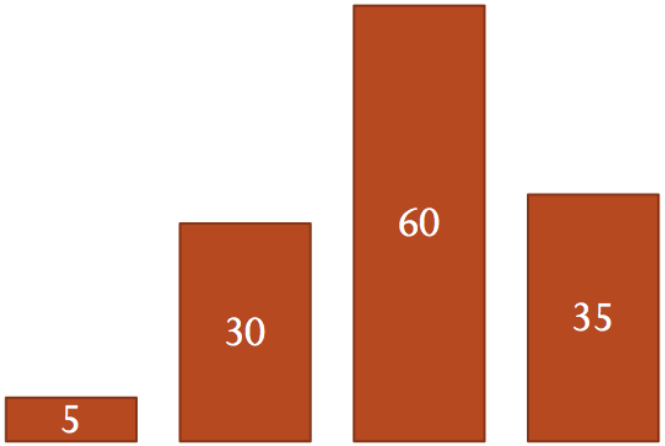


Problem

Minimum Makespan: You have m identical machines and n jobs. For each job i, you are given the duration of this job $d(i)$ that denotes the time that is required by any machine to perform this job. Assign these n jobs on m machine such that the maximum finishing time is minimized.

- Greedy strategy: Assign the next job to a machine with least load.

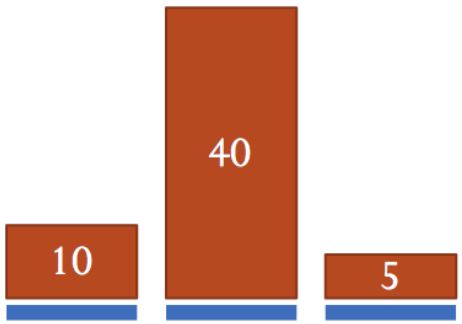


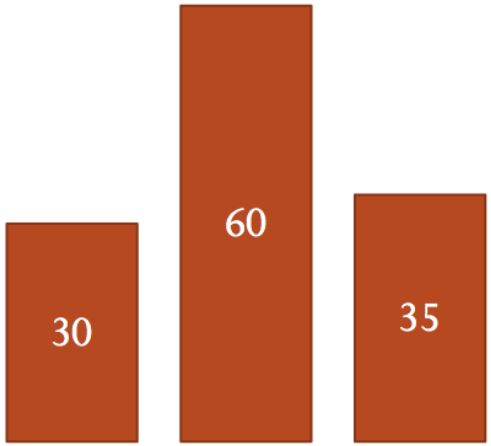


Problem

Minimum Makespan: You have m identical machines and n jobs. For each job i, you are given the duration of this job $d(i)$ that denotes the time that is required by any machine to perform this job. Assign these n jobs on m machine such that the maximum finishing time is minimized.

- Greedy strategy: Assign the next job to a machine with least load.

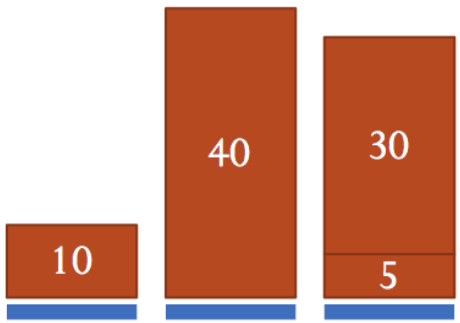


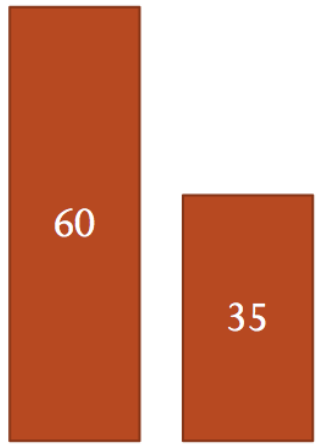


Problem

Minimum Makespan: You have m identical machines and n jobs. For each job i, you are given the duration of this job $d(i)$ that denotes the time that is required by any machine to perform this job. Assign these n jobs on m machine such that the maximum finishing time is minimized.

- Greedy strategy: Assign the next job to a machine with least load.

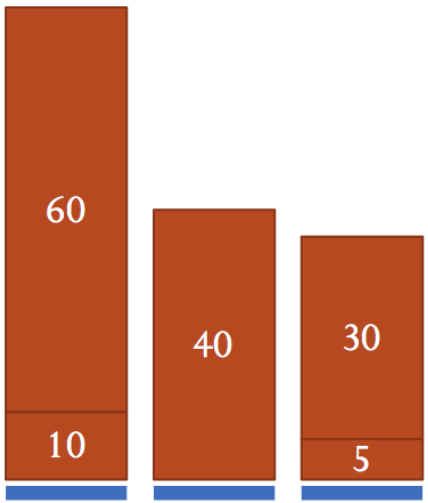




Problem

Minimum Makespan: You have m identical machines and n jobs. For each job i, you are given the duration of this job $d(i)$ that denotes the time that is required by any machine to perform this job. Assign these n jobs on m machine such that the maximum finishing time is minimized.

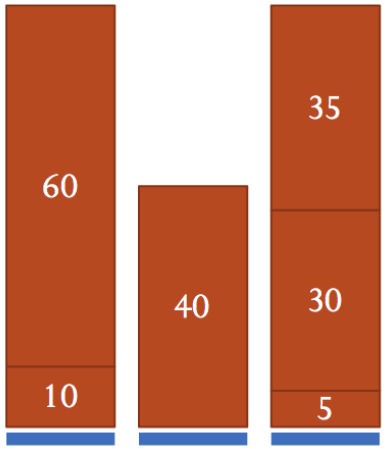
- Greedy strategy: Assign the next job to a machine with least load.



Problem

Minimum Makespan: You have m identical machines and n jobs. For each job i, you are given the duration of this job $d(i)$ that denotes the time that is required by any machine to perform this job. Assign these n jobs on m machine such that the maximum finishing time is minimized.

- Greedy strategy: Assign the next job to a machine with least load.

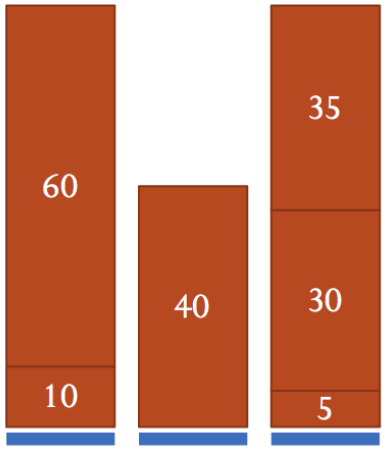


- Is this solution optimal?

Problem

Minimum Makespan: You have m identical machines and n jobs. For each job i, you are given the duration of this job $d(i)$ that denotes the time that is required by any machine to perform this job. Assign these n jobs on m machine such that the maximum finishing time is minimized.

- Greedy strategy: Assign the next job to a machine with least load.



- Is this solution optimal? No

Algorithm

`GreedyMakespan`

- While all jobs are not assigned
  - Assign the next job to a machine with least load

- Let OPT be the optimal value.
- Let G denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

Algorithm

`GreedyMakespan`

- While all jobs are not assigned
  - Assign the next job to a machine with least load

- Let OPT be the optimal value.
- Let G denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

Proof of Claim 1

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$

Algorithm

`GreedyMakespan`

- While all jobs are not assigned
  - Assign the next job to a machine with least load

- Let OPT be the optimal value.
- Let G denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

Proof of Claim 1

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$
- Claim 1.2: For any job t, $OPT \geq d(t)$.

Algorithm

`GreedyMakespan`

- While all jobs are not assigned
  - Assign the next job to a machine with least load

- Let OPT be the optimal value.
- Let G denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

Proof of Claim 1

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$
- Claim 1.2: For any job t, $OPT \geq d(t)$.
- Let the j^{th} machine finish last. Let i be the last job assigned to machine j. Let s be the start time of job i on machine j.
- Claim 1.3: $s \leq \frac{d(1)+d(2)+...+d(n)}{m}$

Algorithm

`GreedyMakespan`

- While all jobs are not assigned
  - Assign the next job to a machine with least load

- Let OPT be the optimal value.
- Let G denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

Proof of Claim 1

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$
- Claim 1.2: For any job t, $OPT \geq d(t)$.
- Let the j^{th} machine finish last. Let i be the last job assigned to machine j. Let s be the start time of job i on machine j.
- Claim 1.3: $s \leq \frac{d(1)+d(2)+...+d(n)}{m}$
- So, $G \leq s + d(i)$
- This implies that $G \leq \frac{d(1)+...+d(n)}{m} + d(i)$ (using claim 1.3)

Algorithm

`GreedyMakespan`

- While all jobs are not assigned
  - Assign the next job to a machine with least load

- Let OPT be the optimal value.
- Let G denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

Proof of Claim 1

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$
- Claim 1.2: For any job t, $OPT \geq d(t)$.
- Let the j^{th} machine finish last. Let i be the last job assigned to machine j. Let s be the start time of job i on machine j.
- Claim 1.3: $s \leq \frac{d(1)+d(2)+...+d(n)}{m}$
- So, $G \leq s + d(i)$
- This implies that $G \leq \frac{d(1)+...+d(n)}{m} + d(i)$ (using claim 1.3)
- This implies that $G \leq OPT + d(i)$ (using claim 1.1)

Algorithm

`GreedyMakespan`

- While all jobs are not assigned
  - Assign the next job to a machine with least load

- Let OPT be the optimal value.
- Let G denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

Proof of Claim 1

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$
- Claim 1.2: For any job t, $OPT \geq d(t)$.
- Let the j^{th} machine finish last. Let i be the last job assigned to machine j. Let s be the start time of job i on machine j.
- Claim 1.3: $s \leq \frac{d(1)+d(2)+...+d(n)}{m}$
- So, $G \leq s + d(i)$
- This implies that $G \leq \frac{d(1)+...+d(n)}{m} + d(i)$ (using claim 1.3)
- This implies that $G \leq OPT + d(i)$ (using claim 1.1)
- This implies that $G \leq OPT + OPT$ (using claim 1.2)

End