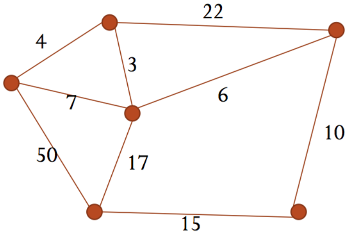
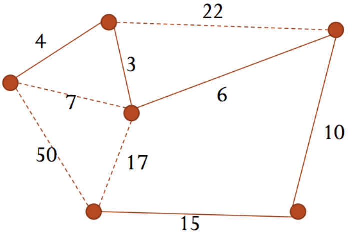
COL106: Data Structures and Algorithms

Ragesh Jaiswal, IIT Delhi

Greedy Algorithms: Minimum Spanning Tree (MST)

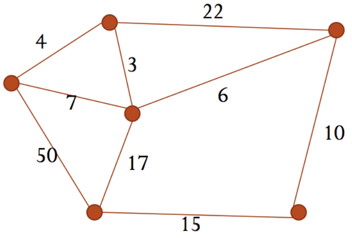
- Spanning Tree: Given a strongly connected graph $G = (V, E)$, a *spanning tree* of G is a subgraph $G' = (V, E')$ such that G' is a tree.
- Minimum Spanning Tree (MST): Given a strongly connected weighted graph $G = (V, E)$, a *Minimum Spanning Tree* of G is a spanning tree of G of minimum total weight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

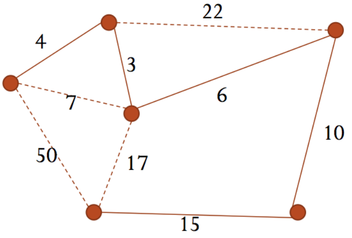




Problem

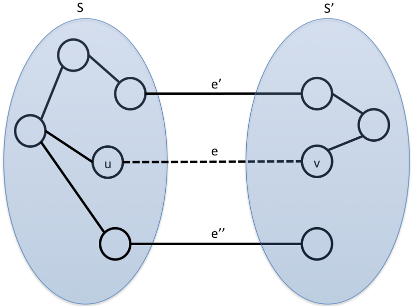
Given a weighted graph G where all the edge weights are distinct, give an algorithm for finding the MST of G.





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.



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

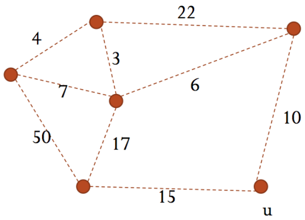
`Kruskal's Algorithm(G)`

- $S \leftarrow E$; $T \leftarrow \{\}$
- 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)`

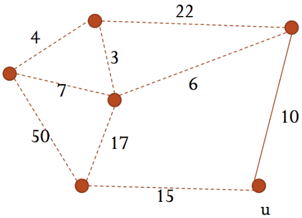
- $S \leftarrow \{u\}$ *//u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$



Algorithm

`Prim's Algorithm(G)`

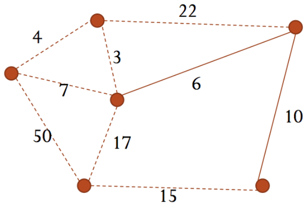
- $S \leftarrow \{u\}$ *//u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$



Algorithm

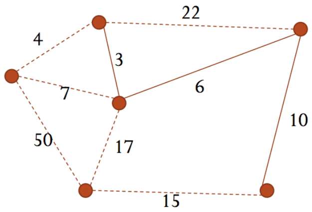
`Prim's Algorithm(G)`

- $S \leftarrow \{u\}$ *//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

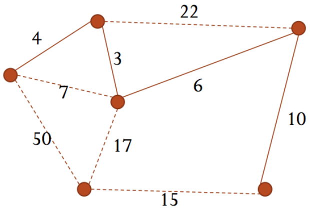
`Prim's Algorithm(G)`
- $S \leftarrow \{u\}$ *//u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$



Algorithm

`Prim's Algorithm(G)`

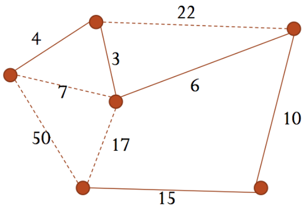
- $S \leftarrow \{u\}$ *//u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$



Algorithm

`Prim's Algorithm(G)`

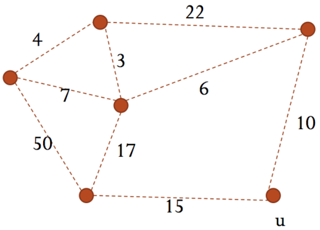
- $S \leftarrow \{u\}$ *//u is an arbitrary vertex in the graph*
- $T \leftarrow \{\}$
- While S does not contain all vertices
  - Let $e = (v, w)$ be the minimum weight edge between S and $V \setminus S$
  - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \cup \{w\}$



Algorithm

`Kruskal's Algorithm(G)`

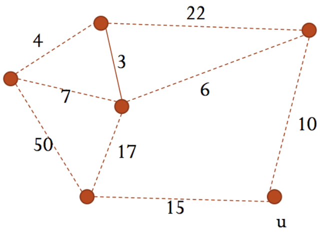
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$



Algorithm

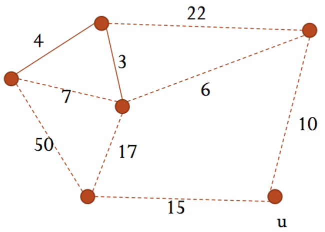
`Kruskal's Algorithm(G)`

- $S \leftarrow E$; $T \leftarrow \{\}$
- 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
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$



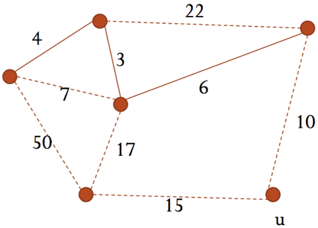
Algorithm

`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

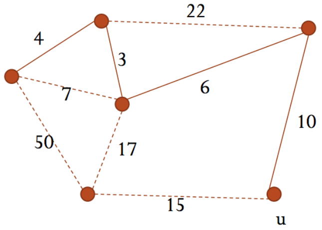
`Kruskal's Algorithm(G)`
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$



Algorithm

`Kruskal's Algorithm(G)`

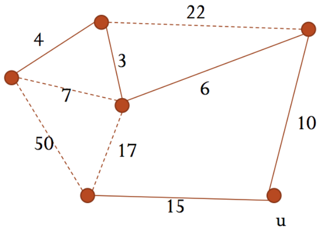
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$



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?
  $O(|E| \cdot \log |V|)$
  - Using a priority queue.

Algorithm

`Kruskal's Algorithm(G)`

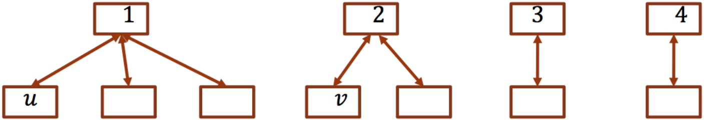
- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - Let e be the minimum weight edge in the set S
  - If e does not create a cycle in T
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$

Algorithm

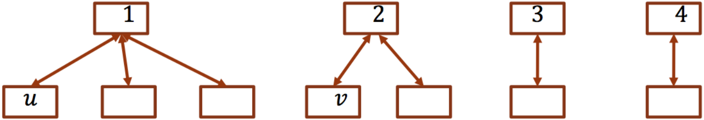
`Kruskal's Algorithm(G)`

- $S \leftarrow E$; $T \leftarrow \{\}$
- While the edge set T does not connect all the vertices
  - *//Note that $G' = (V, T)$ contains dicsonnected components*
  - Let $e = (u, v)$ 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'
    - $T \leftarrow T \cup \{e\}$
  - $S \leftarrow S \setminus \{e\}$

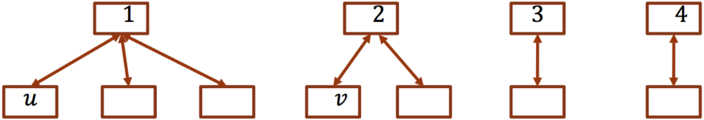
- Union-Find: Used for storing partition of a set of elements. The following two operations are supported:
  1. *Find*(v): Find the partition to which the element v belongs.
  2. *Union*(u, v): Merge the partition to which u belongs with the partition to which v belongs.
- Consider the following data 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 when starting from a full partition.
    - Proof sketch: For any element u, every time its pointer needs to be changed, the size of the partition that it belongs to at least doubles in size. This means that the pointer for u cannot change more than $O(\log k)$ times.

- Kruskal's algorithm using Union-Find.

Algorithm

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

- What is the running time of the above algorithm?

- Kruskal's algorithm using Union-Find.

Algorithm

`Kruskal's Algorithm(G)`

- $T \leftarrow \{\}$
- Let S be the list of edges sorted in increasing order of weight
- While the edge set T does not connect all the vertices
    - *//Note that $G' = (V, T)$ contains dicsonnected components*
    - Let $e = (u, v)$ be the next edge in the list S
    - ~~If e does not create a cycle in T~~
    - ~~If u and v are in different components of G'~~
    - If $(Find(u) \neq Find(v))$
        - $T \leftarrow T \cup \{e\}$
        - $Union(u, v)$

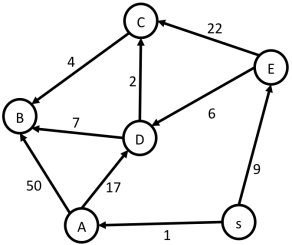
- What is the running time of the above algorithm? $O(|E| \cdot \log |V|)$

- 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.



End