CSI - 3105 Design & Analysis of Algorithms Course 12

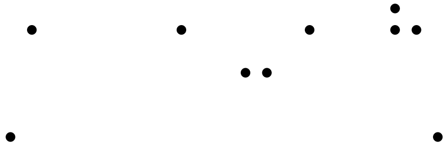
Jean-Lou De Carufel

Fall 2019

Kruskal Algorithm (1956)

Approach : Maintain a forest. In each step, add an edge of minimum weight that does not create a cycle.

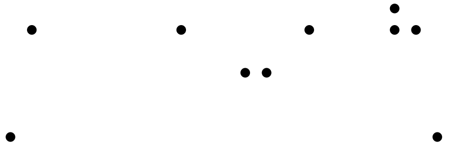
Start : At the beginning, each vertex is a (trivial) tree.



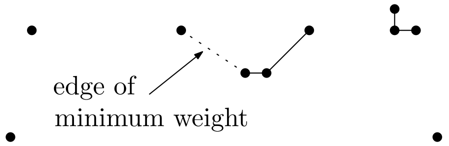
Kruskal Algorithm (1956)

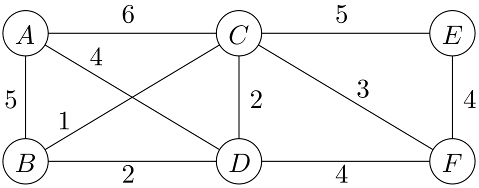
Approach : Maintain a forest. In each step, add an edge of minimum weight that does not create a cycle.

Start : At the beginning, each vertex is a (trivial) tree.



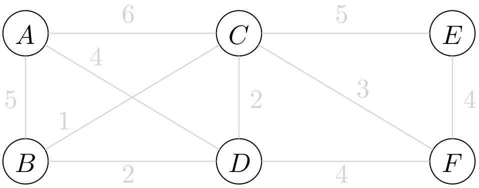
One Iteration : Combine two trees using an edge of minimum weight.

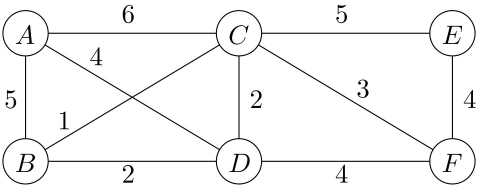




Sor the edges by weight:

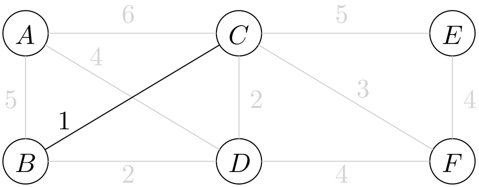
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

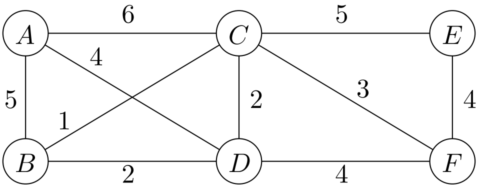




Sor the edges by weight:

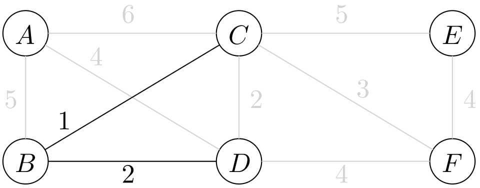
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

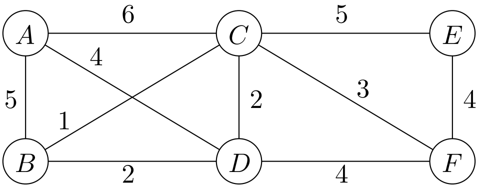




Sor the edges by weight:

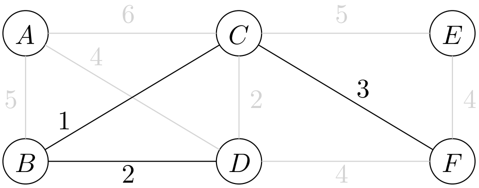
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

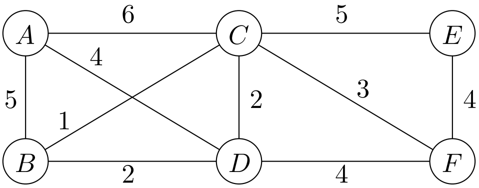




Sor the edges by weight:

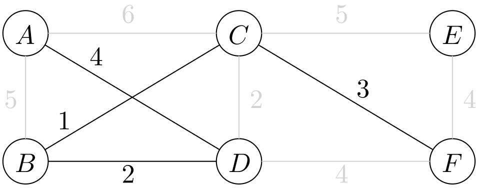
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

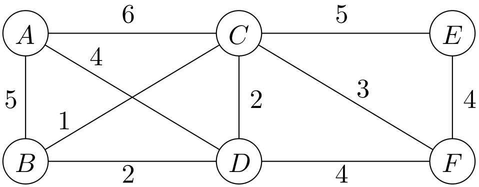




Sor the edges by weight:

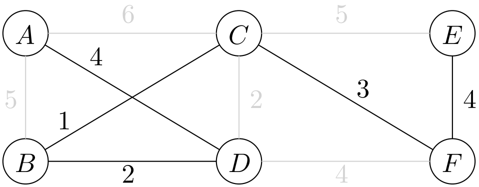
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$





Sor the edges by weight:

$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$



Total weight: 14

Algorithm *Kruskal*(G)

Input: $G = (V, E)$, where $V = \{x_1, x_2, ..., x_n\}$ and $m = |E|$.
Output: A minimum spanning tree of G.

```
1: Sort the edges of $E$ by weight using Merge Sort: $e_1, e_2, ..., e_m$
2: for $i = 1$ to $n$ do
3:     $V_i = \{x_i\}$
4: end for
5: $T = \{ \}$
6: for $k = 1$ to $m$ do
7:     let $u_k$ and $v_k$ be the vertices of $e_k$.
8:     let $i$ be the index such that $u_k \in V_i$
9:     let $j$ be the index such that $v_k \in V_j$
10:    if $i \neq j$ then
11:        $V_i = V_i \cup V_j$
12:        $V_j = \{ \}$
13:        $T = T \cup \{\{u_k, v_k\}\}$
14:    end if
15: end for
16: return $T$
```

Running time:

- Sorting: $O(m \log(m)) = O(m \log(n))$ time (do you see why?)

Running time:

- Sorting: $O(m \log(m)) = O(m \log(n))$ time (do you see why?)
- First For-loop: $O(n)$ time

Running time:

- Sorting: $O(m\log(m)) = O(m\log(n))$ time (do you see why?)
- First For-loop: $O(n)$ time
- Second For-loop:
  - Store T in a linked list. Total time to maintain this list: $O(n)$ time
  - Store the sets V_i using the Union-Find data structure.

Running time:

- Sorting: $O(m\log(m)) = O(m\log(n))$ time (do you see why?)
- First For-loop: $O(n)$ time
- Second For-loop:
  - Store T in a linked list. Total time to maintain this list: $O(n)$ time
  - Store the sets V_i using the Union-Find data structure.

  In this second For-Loop, we do
  - $2m$ **Find** operations
  - $n-1$ **Union** operations

  So in total for the second For-Loop:

$$O(n) + O(m + n\log(n)) \text{ time}$$

Running time:

- Sorting: $O(m\log(m)) = O(m\log(n))$ time (do you see why?)
- First For-loop: $O(n)$ time
- Second For-loop:
  - Store T in a linked list. Total time to maintain this list: $O(n)$ time
  - Store the sets V_i using the Union-Find data structure.

  In this second For-Loop, we do
  - $2m$ **Find** operations
  - $n-1$ **Union** operations

  So in total for the second For-Loop:

$$O(n) + O(m + n\log(n)) \text{ time}$$

So the total time is

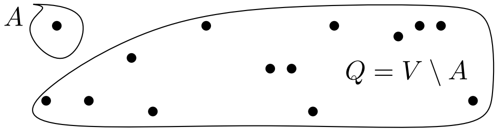
$$O(m\log(n)) + O(n) + O(m + n\log(n)) = O(m\log(n))$$

Do you see why?

Conclusion: Kruskal computes an MST in $O(m \log(n))$ time.

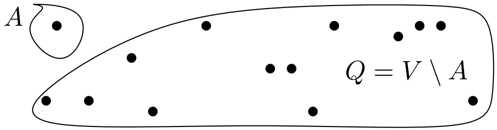
Prim Algorithm (1957) [Jarník (1930), Dijkstra (1959)]

Start : • A is a set consisting of one (arbitrary) vertex of V.
• T is an empty set of edges.

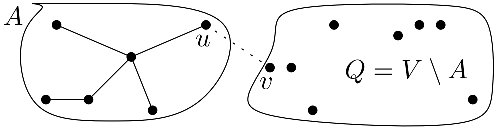


Prim Algorithm (1957) [Jarník (1930), Dijkstra (1959)]

Start :
- A is a set consisting of one (arbitrary) vertex of V.
- T is an empty set of edges.

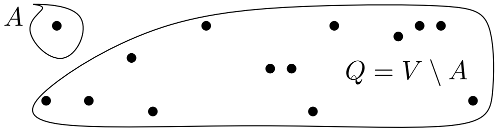


One Iteration :
- Take an edge $\{u, v\}$ of minimum weight such that $u \in A$ and $v \in Q$.
- Add the edge $\{u, v\}$ to T.
- Move v from Q to A.

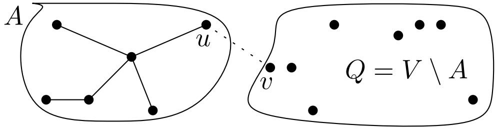


Prim Algorithm (1957) [Jarník (1930), Dijkstra (1959)]

Start :
- A is a set consisting of one (arbitrary) vertex of V.
- T is an empty set of edges.



One Iteration :
- Take an edge $\{u, v\}$ of minimum weight such that $u \in A$ and $v \in Q$.
- Add the edge $\{u, v\}$ to T.
- Move v from Q to A.



Repeat until $A = V$ (i.e. $Q = \{\ \}$)!

```
Algorithm  Prim(G)
Input:  $G = (V, E)$
Output:  A minimum spanning tree of $G$.
1: Let $r \in V$ be an arbitrary vertex.
2: $A = \{r\}$
3: $T = \{\ \}$
4: while $A \neq V$ do
5:     find an edge $\{u, v\} \in E$ of minimum weight such that $u \in A$ and
       $v \in V \setminus A$.
6:     $A = A \cup \{v\}$
7:     $T = T \cup \{\{u, v\}\}$
8: end while
9: return  $X$
```

```
Algorithm  $Prim(G)$
Input:  $G = (V, E)$
Output:  A minimum spanning tree of $G$.
1: Let $r \in V$ be an arbitrary vertex.
2: $A = \{r\}$
3: $T = \{ \}$
4: while $A \neq V$ do
5:     find an edge $\{u, v\} \in E$ of minimum weight such that $u \in A$ and
       $v \in V \setminus A$.
6:     $A = A \cup \{v\}$
7:     $T = T \cup \{\{u, v\}\}$
8: end while
9: return  $X$
```

How to find such an edge $\{u, v\}$?

```
Algorithm  Prim(G)
Input:  $G = (V, E)$
Output:  A minimum spanning tree of $G$.
1: Let $r \in V$ be an arbitrary vertex.
2: $A = \{r\}$
3: $T = \{\ \}$
4: while $A \neq V$ do
5:     find an edge $\{u, v\} \in E$ of minimum weight such that $u \in A$ and
       $v \in V \setminus A$.
6:     $A = A \cup \{v\}$
7:     $T = T \cup \{\{u, v\}\}$
8: end while
9: return  $X$
```

How to find such an edge $\{u, v\}$?
By brute force, it takes $O(|E|)$ time.

```
Algorithm  $Prim(G)$
Input:  $G = (V, E)$
Output:  A minimum spanning tree of $G$.
1: Let $r \in V$ be an arbitrary vertex.
2: $A = \{r\}$
3: $T = \{\ \}$
4: while $A \neq V$ do
5:     find an edge $\{u, v\} \in E$ of minimum weight such that $u \in A$ and
       $v \in V \setminus A$.
6:     $A = A \cup \{v\}$
7:     $T = T \cup \{\{u, v\}\}$
8: end while
9: return  $X$
```

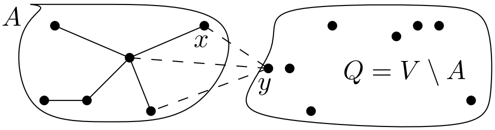
How to find such an edge $\{u, v\}$?
By brute force, it takes $O(|E|)$ time.
So the total running time becomes $O(|V| \cdot |E|)$.

To improve the running time, we maintain extra information.

To improve the running time, we maintain extra information.

For each vertex y in Q,

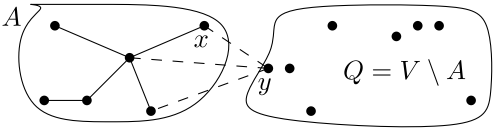
$minweight(y)$	: minimum weight of any edge between y and a vertex of A
$closest(y)$	: vertex x in A for which $wt(x, y) = minweight(y)$



To improve the running time, we maintain extra information.

For each vertex y in Q,

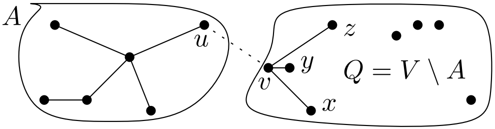
$$\begin{aligned} \mathit{minweight}(y) &: \text{minimum weight of any edge between } y \text{ and a vertex of } A \\ \mathit{closest}(y) &: \text{vertex } x \text{ in } A \text{ for which } \mathit{wt}(x,y) = \mathit{minweight}(y) \end{aligned}$$



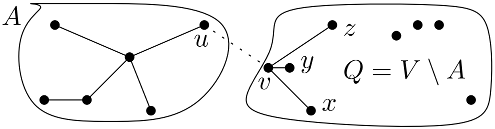
Observe that: a shortest edge $\{u, v\}$ connecting A and Q has weight

$$\min_{y \in Q} \{\mathit{minweight}(y)\}.$$

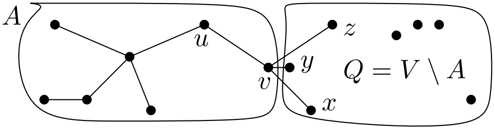
What happens if we move v from Q to A?



What happens if we move v from Q to A?



We update *minweight*(w) and *closest*(w) for $w = x, y, z$.



Algorithm $Prim(G)$

```
Input:   $G = (V, E)$
Output:  A minimum spanning tree of $G$.
 1: Let $r \in V$ be an arbitrary vertex
 2: $A = \{r\}$
 3: $T = \{\ \}$
 4: for each vertex $y \neq r$ do
 5:     $minweight(y) = \infty$
 6:     $closest(y) = \text{NIL}$
 7: end for
 8: for each edge $\{r, y\}$ do
 9:     $minweight(y) = wt(r, y)$
10:     $closest(y) = r$
11: end for
12: $Q = V \setminus \{r\}$
13: $k = 1$              // Stores the size of $A$
14: while $k \neq n$ do
15:     Let $v$ be the vertex of $Q$ for which $minweight(v)$ is minimum
16:     $u = closest(v)$
17:     $A = A \cup \{v\}$
18:     $Q = Q \setminus \{v\}$
19:     $T = T \cup \{\{u, v\}\}$
20:     $k = k + 1$
21:     for each edge $\{v, y\}$ do
22:         if $y \in Q$ and $wt(v, y) < minweight(y)$ then
23:             $minweight(y) = wt(v, y)$
24:             $closest(y) = v$
25:         end if
26:     end for
27: end while
28: return $T$
```

- Store the vertices of Q in a min-heap.
  For each vertex $v \in Q$, the key of v is $\mathit{minweight}(v)$.
- Store T in a list.
- With each vertex of V, store one bit indicating whether the vertex belongs to A or to Q.

- Store the vertices of Q in a min-heap.
  For each vertex $v \in Q$, the key of v is $minweight(v)$.
- Store T in a list.
- With each vertex of V, store one bit indicating whether the vertex belongs to A or to Q.

Running time:

- Up to the while loop: $O(n)$ time
  (this includes the time to build the heap).

- Store the vertices of Q in a min-heap.
  For each vertex $v \in Q$, the key of v is $\mathit{minweight}(v)$.
- Store T in a list.
- With each vertex of V, store one bit indicating whether the vertex belongs to A or to Q.

Running time:

- Up to the while loop: $O(n)$ time
  (this includes the time to build the heap).
- One iteration of the while-loop:
  - *extract_min*: $O(\log(n))$ time
  - At most $\mathit{degree}(v)$ many *decrease_key* operations:
    $O(\mathit{degree}(v) \cdot \log(n))$ time

- Store the vertices of Q in a min-heap.
  For each vertex $v \in Q$, the key of v is $\mathit{minweight}(v)$.
- Store T in a list.
- With each vertex of V, store one bit indicating whether the vertex belongs to A or to Q.

Running time:

- Up to the while loop: $O(n)$ time
  (this includes the time to build the heap).
- One iteration of the while-loop:
  - *extract_min*: $O(\log(n))$ time
  - At most $\mathit{degree}(v)$ many *decrease_key* operations:
    $O(\mathit{degree}(v) \cdot \log(n))$ time
- Total time for the while-loop:

$$O\left(\sum_{v \in V} \mathit{degree}(v) \cdot \log(n)\right) = O(2m \log(n)) = O(m \log(n))$$

Conclusion: Prim computes an MST in $O(m \log(n))$ time.