CSI - 3105 Design & Analysis of Algorithms
Course 12

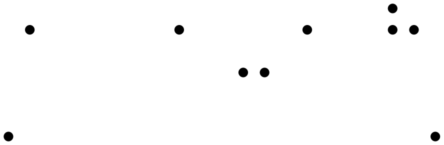
Jean-Lou De Carufel

Fall 2019

Kruskal Algorithm (1956)

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

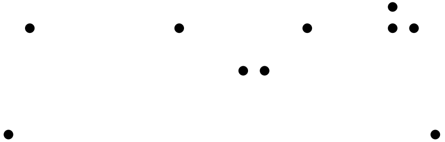
Start : At the beginning, each vertex is a (trivial) tree.



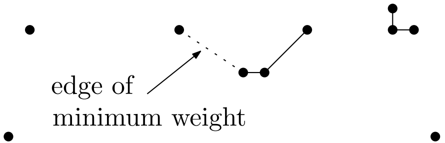
Kruskal Algorithm (1956)

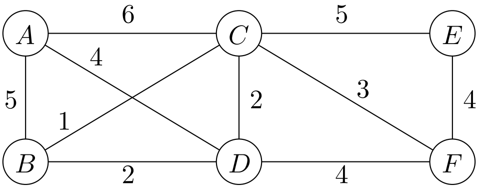
Approach : Maintain a forest. In each step, add an edge of minimum weight that does not create a cycle.

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



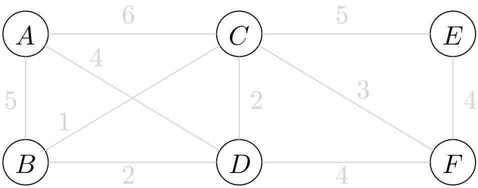
One Iteration : Combine two trees using an edge of minimum weight.

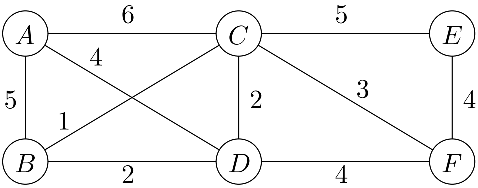




Sor the edges by weight:

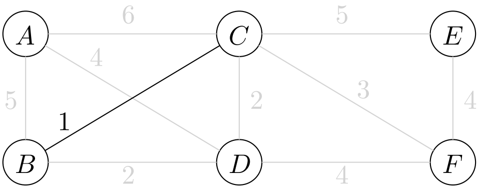
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

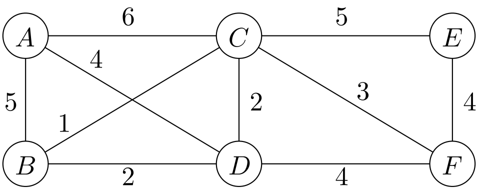




Sor the edges by weight:

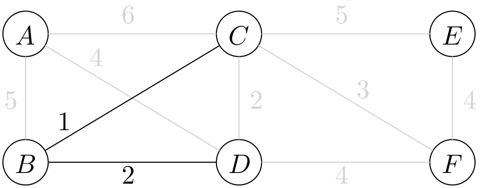
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

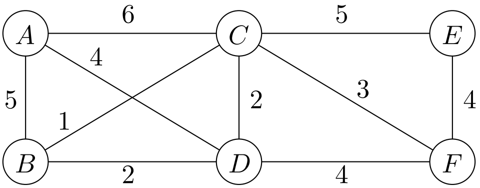




Sor the edges by weight:

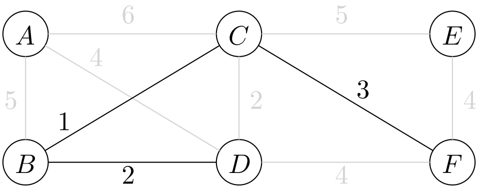
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

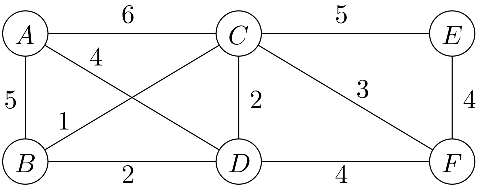




Sor the edges by weight:

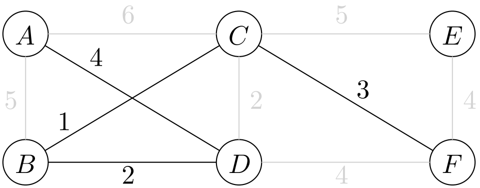
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$

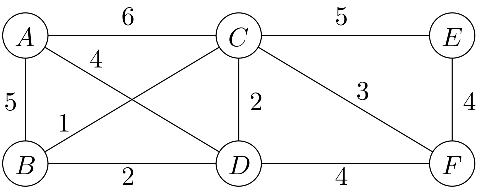




Sor the edges by weight:

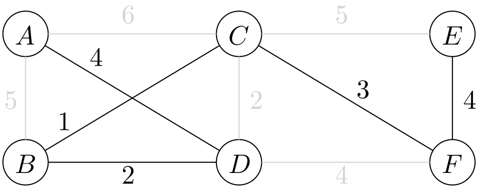
$$BC, BD, CD, CF, AD, DF, EF, AB, CE, AC$$





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.

```
Sort the edges of E by weight using Merge Sort: e_1, e_2, ..., e_m
for i = 1 to n do
    V_i = {x_i}
end for
T = { }
for k = 1 to m do
    let u_k and v_k be the vertices of e_k.
    let i be the index such that u_k ∈ V_i
    let j be the index such that v_k ∈ V_j
    if i ≠ j then
        V_i = V_i ∪ V_j
        V_j = { }
        T = T ∪ {{u_k, v_k}}
    end if
end for
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

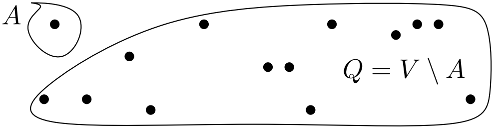
$$O(m\log(n)) + O(n) + O(m + n\log(n)) = O(m\log(n))$$

Do you see why?

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

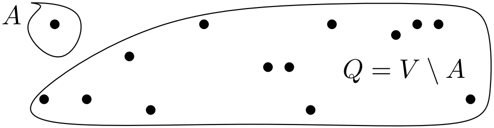
Prim Algorithm (1957) [Jarník (1930), Dijkstra (1959)]

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

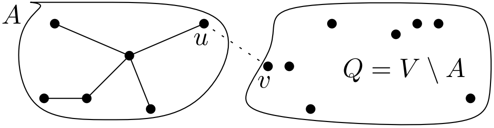


Prim Algorithm (1957) [Jarník (1930), Dijkstra (1959)]

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

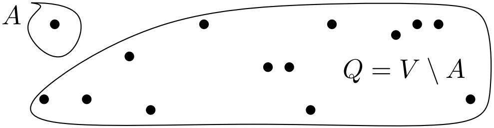


One Iteration :
- Take an edge $\{u, v\}$ of minimum weight such that $u \in A$ and $v \in Q$.
- Add the edge $\{u, v\}$ to T.
- Move v from Q to A.

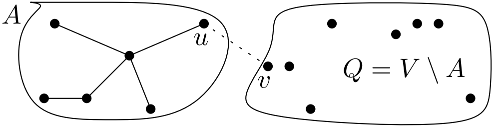


Prim Algorithm (1957) [Jarník (1930), Dijkstra (1959)]

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.

Let $r \in V$ be an arbitrary vertex.
$A = \{r\}$
$T = \{\ \}$
while $A \neq V$ **do**
    find an edge $\{u, v\} \in E$ of minimum weight such that $u \in A$ and $v \in V \setminus A$.
    $A = A \cup \{v\}$
    $T = T \cup \{\{u, v\}\}$
end while
return X

Algorithm *Prim*(G)

Input: $G = (V, E)$

Output: A minimum spanning tree of G.

```
Let r ∈ V be an arbitrary vertex.
A = {r}
T = { }
while A ≠ V do
    find an edge {u, v} ∈ E of minimum weight such that u ∈ A and
    v ∈ V \ A.
    A = A ∪ {v}
    T = T ∪ {{u, v}}
end while
return X
```

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

Algorithm *Prim*(G)

Input: $G = (V, E)$

Output: A minimum spanning tree of G.

Let $r \in V$ be an arbitrary vertex.
$A = \{r\}$
$T = \{\ \}$
while $A \neq V$ **do**
    find an edge $\{u, v\} \in E$ of minimum weight such that $u \in A$ and $v \in V \setminus A$.
    $A = A \cup \{v\}$
    $T = T \cup \{\{u, v\}\}$
end while
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.

Let $r \in V$ be an arbitrary vertex.
$A = \{r\}$
$T = \{\ \}$
while $A \neq V$ **do**
    find an edge $\{u, v\} \in E$ of minimum weight such that $u \in A$ and $v \in V \setminus A$.
    $A = A \cup \{v\}$
    $T = T \cup \{\{u, v\}\}$
end while
return X

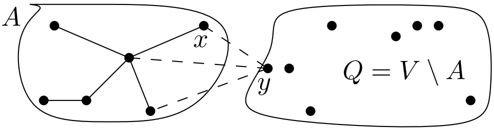
How to find such an edge $\{u, v\}$?
By brute force, it takes $O(|E|)$ time.
So the total running time becomes $O(|V| \cdot |E|)$.

To improve the running time, we maintain extra information.

To improve the running time, we maintain extra information.

For each vertex y in Q,

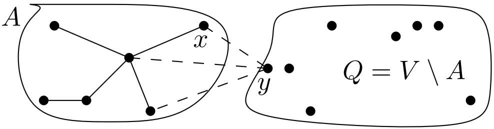
$$\begin{aligned} \mathit{minweight}(y) &: \text{minimum weight of any edge between } y \text{ and a vertex of } A \\ \mathit{closest}(y) &: \text{vertex } x \text{ in } A \text{ for which } \mathit{wt}(x,y) = \mathit{minweight}(y) \end{aligned}$$



To improve the running time, we maintain extra information.

For each vertex y in Q,

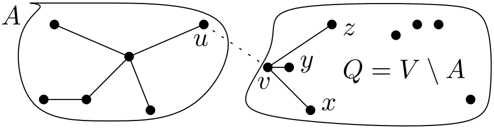
$$\begin{aligned} \mathit{minweight}(y) &: \text{minimum weight of any edge between } y \text{ and a vertex of } A \\ \mathit{closest}(y) &: \text{vertex } x \text{ in } A \text{ for which } \mathit{wt}(x,y) = \mathit{minweight}(y) \end{aligned}$$



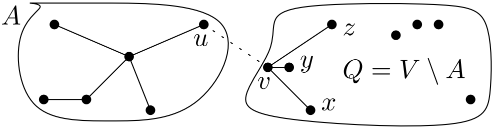
Observe that: a shortest edge $\{u, v\}$ connecting A and Q has weight

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

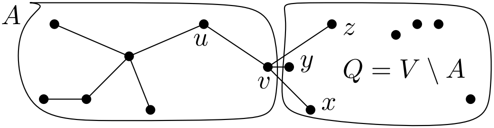
What happens if we move v from Q to A?



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.

Let $r \in V$ be an arbitrary vertex
$A = \{r\}$
$T = \{\ \}$
for each vertex $y \neq r$ **do**
    $minweight(y) = \infty$
    $closest(y) = \text{NIL}$
end for
for each edge $\{r, y\}$ **do**
    $minweight(y) = wt(r, y)$
    $closest(y) = r$
end for
$Q = V \setminus \{r\}$
$k = 1$ // Stores the size of A
while $k \neq n$ **do**
    Let v be the vertex of Q for which $minweight(v)$ is minimum
    $u = closest(v)$
    $A = A \cup \{v\}$
    $Q = Q \setminus \{v\}$
    $T = T \cup \{\{u, v\}\}$
    $k = k + 1$
    for each edge $\{v, y\}$ **do**
        if $y \in Q$ and $wt(v, y) < minweight(y)$ **then**
            $minweight(y) = wt(v, y)$
            $closest(y) = v$
        end if
    end for
end while
return T

update y's

- 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.

- 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).

- 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.