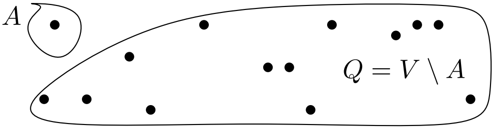
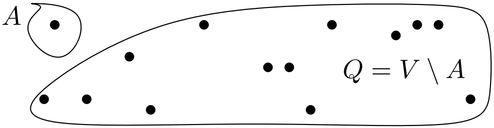
Prim Algorithm (1957) [Jarník (1930), Dijkstra (1959)]

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

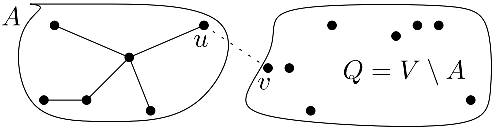


Prim Algorithm (1957) [Jarník (1930), Dijkstra (1959)]

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

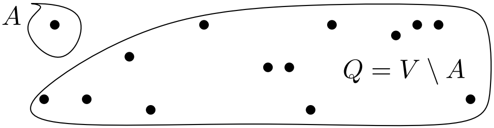


One Iteration :
- Take an edge $\{u, v\}$ of minimum weight such that $u \in A$ and $v \in Q$.
- Add the edge $\{u, v\}$ to T.
- Move v from Q to A.

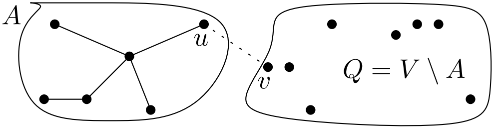


Prim Algorithm (1957) [Jarník (1930), Dijkstra (1959)]

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 $O(|V| \cdot |E|)$.

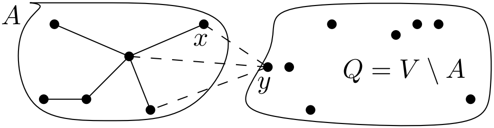
To improve the running time, we maintain extra information.

To improve the running time, we maintain extra information.

For each vertex y in Q,

$\mathit{minweight}(y)$: minimum weight of any edge between y and a vertex of A

$\mathit{closest}(y)$: vertex x in A for which $wt(x, y) = \mathit{minweight}(y)$

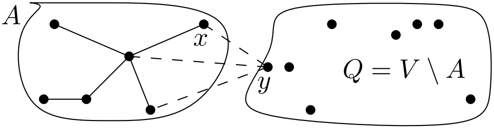


To improve the running time, we maintain extra information.

For each vertex y in Q,

$minweight(y)$: minimum weight of any edge between y and a vertex of A

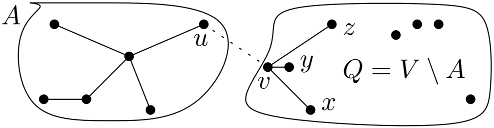
$closest(y)$: vertex x in A for which $wt(x, y) = minweight(y)$



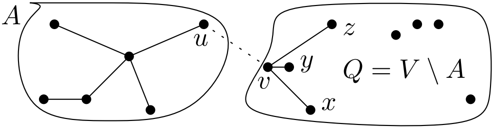
Observe that: a shortest edge $\{u, v\}$ connecting A and Q has weight

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

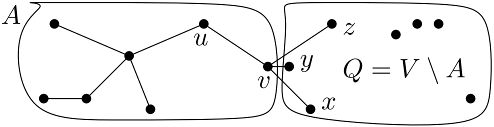
What happens if we move v from Q to A?



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.

```
Let r ∈ V be an arbitrary vertex
A = {r}
T = { }
for each vertex y ≠ r do
    minweight(y) = ∞
    closest(y) = NIL
end for
for each edge {r, y} do
    minweight(y) = wt(r, y)
    closest(y) = r
end for
Q = V \ {r}
k = 1            // Stores the size of A
while k ≠ n do
    Let v be the vertex of Q for which minweight(v) is minimum
    u = closest(v)
    A = A ∪ {v}
    Q = Q \ {v}
    T = T ∪ {{u, v}}
    k = k + 1
    for each edge {v, y} do
        if y ∈ Q and wt(v, y) < minweight(y) then
            minweight(y) = wt(v, y)
            closest(y) = v
        end if
    end for
end while
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 $\mathit{minweight}(v)$.
- 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.