20.6.4
Implementing Kruskal’s Algorithm
Kruskal’s Algorithm

Kruskal\_ComputeMST

Initially $E$ is the set of all edges in $G$
$T$ is empty (* $T$ will store edges of a MST *)

while $E$ is not empty do
  choose $e \in E$ of minimum cost
  if $(T \cup \{e\}$ does not have cycles)
    add $e$ to $T$

return the set $T$

1. Presort edges based on cost. Choosing minimum can be done in $O(1)$ time
2. Do BFS/DFS on $T \cup \{e\}$. Takes $O(n)$ time
3. Total time $O(m \log m) + O(mn) = O(mn)$
**Kruskal’s Algorithm**

```
Kruskal_ComputeMST

Initially $E$ is the set of all edges in $G$
$T$ is empty (* $T$ will store edges of a MST *)

while $E$ is not empty do
    choose $e \in E$ of minimum cost
    if ($T \cup \{e\}$ does not have cycles)
        add $e$ to $T$

return the set $T$
```

1. Presort edges based on cost. Choosing minimum can be done in $O(1)$ time
2. Do BFS/DFS on $T \cup \{e\}$. Takes $O(n)$ time
3. Total time $O(m \log m) + O(mn) = O(mn)$
Kruskal’s Algorithm

Kruskal\_ComputeMST

Initially \( E \) is the set of all edges in \( G \)
\( T \) is empty (* \( T \) will store edges of a MST *)

while \( E \) is not empty do

choose \( e \in E \) of minimum cost

if \((T \cup \{e\} \) does not have cycles) add \( e \) to \( T \)

return the set \( T \)

1. Presort edges based on cost. Choosing minimum can be done in \( O(1) \) time
2. Do BFS/DFS on \( T \cup \{e\} \). Takes \( O(n) \) time
3. Total time \( O(m \log m) + O(mn) = O(mn) \)
Kruskal’s Algorithm

Kruskal\_ComputeMST
Initially \( E \) is the set of all edges in \( G \)
\( T \) is empty (* \( T \) will store edges of a MST *)
while \( E \) is not empty do
  choose \( e \in E \) of minimum cost
  if \( (T \cup \{e\} \text{ does not have cycles}) \)
    add \( e \) to \( T \)
return the set \( T \)

1. Presort edges based on cost. Choosing minimum can be done in \( O(1) \) time
2. Do BFS/DFS on \( T \cup \{e\} \). Takes \( O(n) \) time
3. Total time \( O(m \log m) + O(mn) = O(mn) \)
Kruskal’s Algorithm

**Kruskal**\_ComputeMST

Initially $E$ is the set of all edges in $G$

$T$ is empty (* $T$ will store edges of a MST *)

while $E$ is not empty do

choose $e \in E$ of minimum cost

if $(T \cup \{e\}$ does not have cycles)

add $e$ to $T$

return the set $T$

1. Presort edges based on cost. Choosing minimum can be done in $O(1)$ time
2. Do BFS/DFS on $T \cup \{e\}$. Takes $O(n)$ time
3. Total time $O(m \log m) + O(mn) = O(mn)$
Kruskal’s Algorithm

KruskalComputeMST

Initially $E$ is the set of all edges in $G$
$T$ is empty (* $T$ will store edges of a MST *)

while $E$ is not empty do
    choose $e \in E$ of minimum cost
    if ($T \cup \{e\}$ does not have cycles)
        add $e$ to $T$

return the set $T$

1. Presort edges based on cost. Choosing minimum can be done in $O(1)$ time
2. Do BFS/DFS on $T \cup \{e\}$. Takes $O(n)$ time
3. Total time $O(m \log m) + O(mn) = O(mn)$
Implementing Kruskal’s Algorithm Efficiently

```
Kruskal_ComputeMST
Sort edges in $E$ based on cost
$T$ is empty (* $T$ will store edges of a MST *)
each vertex $u$ is placed in a set by itself
while $E$ is not empty do
    pick $e = (u, v) \in E$ of minimum cost
    if $u$ and $v$ belong to different sets
        add $e$ to $T$
        merge the sets containing $u$ and $v$
return the set $T$
```

Need a data structure to check if two elements belong to same set and to merge two sets.
Using Union-Find data structure can implement Kruskal’s algorithm in $O((m + n) \log m)$ time.
Implementing Kruskal’s Algorithm Efficiently

Kruskal\_ComputeMST

Sort edges in $E$ based on cost

$T$ is empty (* $T$ will store edges of a MST *)

each vertex $u$ is placed in a set by itself

while $E$ is not empty do

pick $e = (u, v) \in E$ of minimum cost

if $u$ and $v$ belong to different sets

add $e$ to $T$

merge the sets containing $u$ and $v$

return the set $T$

Need a data structure to check if two elements belong to same set and to merge two sets.

Using Union-Find data structure can implement Kruskal’s algorithm in $O((m + n) \log m)$ time.
Implementing Kruskal’s Algorithm Efficiently

Kruskal\_ComputeMST

Sort edges in $E$ based on cost

$T$ is empty (* $T$ will store edges of a MST *)

each vertex $u$ is placed in a set by itself

while $E$ is not empty do

pick $e = (u, v) \in E$ of minimum cost

if $u$ and $v$ belong to different sets

add $e$ to $T$

merge the sets containing $u$ and $v$

return the set $T$

Need a data structure to check if two elements belong to same set and to merge two sets.

Using Union-Find data structure can implement Kruskal’s algorithm in $O((m + n) \log m)$ time.
THE END

...

(for now)