20.6.2
Implementing Prim’s Algorithm
Implementing Prim’s Algorithm

```
Prim_ComputeMST
E is the set of all edges in G
S = {1}
T is empty (* T will store edges of a MST *)
while S ≠ V do
    pick e = (v, w) ∈ E such that
    v ∈ S and w ∈ V - S
    e has minimum cost
    T = T ∪ e
    S = S ∪ w
return the set T
```

Analysis

1. Number of iterations = \( O(n) \), where \( n \) is number of vertices
2. Picking \( e \) is \( O(m) \) where \( m \) is the number of edges
3. Total time \( O(nm) \)
Implementing Prim’s Algorithm

**Prim ComputeMST**

- $E$ is the set of all edges in $G$
- $S = \{1\}$
- $T$ is empty (* $T$ will store edges of a MST *)

**while** $S \neq V$ **do**

- pick $e = (v, w) \in E$ such that $v \in S$ and $w \in V - S$
- $e$ has minimum cost
- $T = T \cup e$
- $S = S \cup w$

**return** the set $T$

**Analysis**

- Number of iterations = $O(n)$, where $n$ is number of vertices
- Picking $e$ is $O(m)$ where $m$ is the number of edges
- Total time $O(nm)$
Implementing Prim’s Algorithm

**Prim_ComputeMST**

- $E$ is the set of all edges in $G$
- $S = \{1\}$
- $T$ is empty (* $T$ will store edges of a MST *)

while $S \neq V$ do
  pick $e = (v, w) \in E$ such that
  - $v \in S$ and $w \in V - S$
  - $e$ has minimum cost
  $T = T \cup e$
  $S = S \cup w$

return the set $T$

**Analysis**

1. Number of iterations $= O(n)$, where $n$ is number of vertices
2. Picking $e$ is $O(m)$ where $m$ is the number of edges
3. Total time $O(nm)$
Implementing Prim’s Algorithm

**Prim_ComputeMST**

- $E$ is the set of all edges in $G$
- $S = \{1\}$
- $T$ is empty (* $T$ will store edges of a MST *)

while $S \neq V$ do

- pick $e = (v, w) \in E$ such that $v \in S$ and $w \in V - S$
- $e$ has minimum cost

- $T = T \cup e$
- $S = S \cup w$

return the set $T$

**Analysis**

1. Number of iterations = $O(n)$, where $n$ is number of vertices
2. Picking $e$ is $O(m)$ where $m$ is the number of edges
3. Total time $O(nm)$
## Implementing Prim’s Algorithm

### More Efficient Implementation

**Prim ComputeMST**

| E is the set of all edges in G |
| S = {1} |
| T is empty (* T will store edges of a MST *) |

for v \( \not\in S \), \( a(v) = \min_{w \in S} c(w, v) \)

for v \( \not\in S \), \( e(v) = w \) such that \( w \in S \) and \( c(w, v) \) is minimum

while \( S \neq V \) do

pick v with minimum \( a(v) \)

\( T = T \cup \{(e(v), v)\} \)

\( S = S \cup \{v\} \)

update arrays \( a \) and \( e \)

return the set \( T \)

Maintain vertices in \( V \setminus S \) in a priority queue with key \( a(v) \).
Implementing Prim’s Algorithm

More Efficient Implementation

```plaintext
Prim_ComputeMST

- \( E \) is the set of all edges in \( G \)
- \( S = \{1\} \)
- \( T \) is empty (* \( T \) will store edges of a MST *)

for \( v \not\in S \), \( a(v) = \min_{w \in S} c(w, v) \)

for \( v \not\in S \), \( e(v) = w \) such that \( w \in S \) and \( c(w, v) \) is minimum

while \( S \neq V \) do
  pick \( v \) with minimum \( a(v) \)
  \( T = T \cup \{(e(v), v)\} \)
  \( S = S \cup \{v\} \)

update arrays \( a \) and \( e \)

return the set \( T \)
```

Maintain vertices in \( V \setminus S \) in a priority queue with key \( a(v) \).
Implementing Prim’s Algorithm
More Efficient Implementation

```
Prim_ComputeMST
    E is the set of all edges in G
    S = {1}
    T is empty (* T will store edges of a MST *)
    for v ∉ S, a(v) = min_{w ∈ S} c(w, v)
    for v ∉ S, e(v) = w such that w ∈ S and c(w, v) is minimum
    while S ≠ V do
        pick v with minimum a(v)
        T = T ∪ {(e(v), v)}
        S = S ∪ {v}
        update arrays a and e
    return the set T
```

Maintain vertices in V \ S in a priority queue with key a(v).
20.6.3
Implementing Prim’s algorithm with priority queues
Priority Queues

Data structure to store a set $S$ of $n$ elements where each element $v \in S$ has an associated real/integer key $k(v)$ such that the following operations

1. **makeQ**: create an empty queue
2. **findMin**: find the minimum key in $S$
3. **extractMin**: Remove $v \in S$ with smallest key and return it
4. **add** ($v$, $k(v)$): Add new element $v$ with key $k(v)$ to $S$
5. **Delete** ($v$): Remove element $v$ from $S$
6. **decreaseKey** ($v$, $k'(v)$): decrease key of $v$ from $k(v)$ (current key) to $k'(v)$ (new key). Assumption: $k'(v) \leq k(v)$
7. **meld**: merge two separate priority queues into one
Prim’s using priority queues

\[
E \text{ is the set of all edges in } G \\
S = \{1\} \\
T \text{ is empty (} T \text{ will store edges of a MST *)} \\
\text{for } v \not\in S, \ a(v) = \min_{w \in S} c(w, v) \\
\text{for } v \not\in S, \ e(v) = w \text{ such that } w \in S \text{ and } c(w, v) \text{ is minimum} \\
\text{while } S \neq V \text{ do} \\
\quad \text{pick } v \text{ with minimum } a(v) \\
\quad T = T \cup \{(e(v), v)\} \\
\quad S = S \cup \{v\} \\
\quad \text{update arrays } a \text{ and } e \\
\text{return the set } T
\]

Maintain vertices in \( V \setminus S \) in a priority queue with key \( a(v) \)

1. Requires \( O(n) \) extractMin operations
2. Requires \( O(m) \) decreaseKey operations
Prim’s using priority queues

\[ E \] is the set of all edges in \( G \)
\[ S = \{1\} \]
\( T \) is empty (* \( T \) will store edges of a MST *)

for \( v \not\in S \), \( a(v) = \min_{w \in S} c(w, v) \)

for \( v \not\in S \), \( e(v) = w \) such that \( w \in S \) and \( c(w, v) \) is minimum

while \( S \neq V \) do
    pick \( v \) with minimum \( a(v) \)
    \( T = T \cup \{(e(v), v)\} \)
    \( S = S \cup \{v\} \)
    update arrays \( a \) and \( e \)

return the set \( T \)

Maintain vertices in \( V \setminus S \) in a priority queue with key \( a(v) \)

⚠ Requires \( O(n) \) extractMin operations

⚠️ Requires \( O(m) \) decreaseKey operations
Prim’s using priority queues

\[
E \text{ is the set of all edges in } G \\
S = \{1\} \\
T \text{ is empty (} \ast \text{ } T \text{ will store edges of a MST } \ast\) \\
\text{for } v \not\in S, \ a(v) = \min_{w \in S} c(w, v) \\
\text{for } v \not\in S, \ e(v) = w \text{ such that } w \in S \text{ and } c(w, v) \text{ is minimum} \\
\text{while } S \neq V \text{ do} \\
\quad \text{pick } v \text{ with minimum } a(v) \\
\quad T = T \cup \{(e(v), v)\} \\
\quad S = S \cup \{v\} \\
\quad \text{update arrays } a \text{ and } e \\
\text{return the set } T
\]

Maintain vertices in \( V \setminus S \) in a priority queue with key \( a(v) \)

1. Requires \( O(n) \) extractMin operations
2. Requires \( O(m) \) decreaseKey operations
Running time of Prim’s Algorithm

\( O(n) \) extractMin operations and \( O(m) \) decreaseKey operations

1. Using standard Heaps, extractMin and decreaseKey take \( O(\log n) \) time. Total: \( O((m + n) \log n) \)

2. Using Fibonacci Heaps, \( O(\log n) \) for extractMin and \( O(1) \) (amortized) for decreaseKey. Total: \( O(n \log n + m) \).

Prim’s algorithm and Dijkstra’s algorithms are similar. Where is the difference?

Prim’s algorithm = Dijkstra where length of a path \( \pi \) is the weight of the heaviest edge in \( \pi \). (Bottleneck shortest path.)
Running time of Prim’s Algorithm

\( O(n) \) extractMin operations and \( O(m) \) decreaseKey operations

1. Using standard Heaps, extractMin and decreaseKey take \( O(\log n) \) time. Total: \( O((m + n) \log n) \)

2. Using Fibonacci Heaps, \( O(\log n) \) for extractMin and \( O(1) \) (amortized) for decreaseKey. Total: \( O(n \log n + m) \).

3. Prim’s algorithm and Dijkstra’s algorithms are similar. Where is the difference?

Prim’s algorithm = Dijkstra where length of a path \( \pi \) is the weight of the heaviest edge in \( \pi \). (Bottleneck shortest path.)
Running time of Prim’s Algorithm

\( O(n) \) extractMin operations and \( O(m) \) decreaseKey operations

1. Using standard Heaps, extractMin and decreaseKey take \( O(\log n) \) time. Total: \( O((m + n) \log n) \)
2. Using Fibonacci Heaps, \( O(\log n) \) for extractMin and \( O(1) \) (amortized) for decreaseKey. Total: \( O(n \log n + m) \).
3. Prim’s algorithm and Dijkstra’s algorithms are similar. Where is the difference?
4. Prim’s algorithm = Dijkstra where length of a path \( \pi \) is the weight of the heaviest edge in \( \pi \). (Bottleneck shortest path.)
THE END

... (for now)