Graphs - Introduction

Fundamental Algorithms

Rodica Potolea, Camelia Lemnaru and Ciprian Oprișa

Technical University of Cluj-Napoca
Computer Science Department

Agenda

1. Graph Representation
2. Minimum Spanning Tree (MST)
   - Kruskal's Algorithm
   - Prim's Algorithm
3. Breadth-First Search (BFS)

Agenda

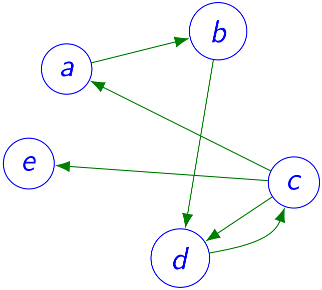
1. Graph Representation
2. Minimum Spanning Tree (MST)
   - Kruskal's Algorithm
   - Prim's Algorithm
3. Breadth-First Search (BFS)

Graph Representation

$G = (V, E)$

- V - node/vertex set
- E - edge set, $E \subseteq V \times V$

Representation:



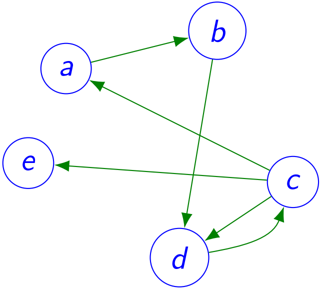
Graph Representation

$G = (V, E)$

- V - node/vertex set
- E - edge set, $E \subseteq V \times V$

Representation:

- adjacency matrix



	a	b	c	d	e
a	0	1	0	0	0
b	0	0	0	1	0
c	1	0	0	1	1
d	0	0	1	0	0
e	0	0	0	0	0

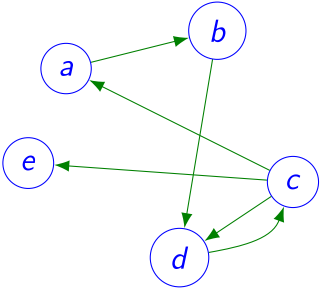
Graph Representation

$G = (V, E)$

- V - node/vertex set
- E - edge set, $E \subseteq V \times V$

Representation:

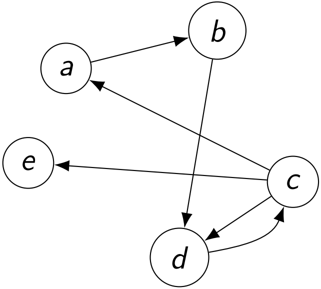
- adjacency matrix
- adjacency lists



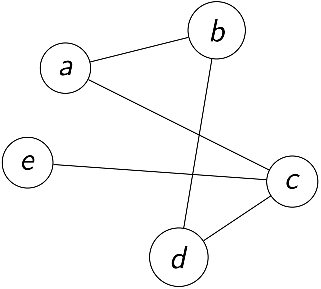
	a	b	c	d	e
a	0	1	0	0	0
b	0	0	0	1	0
c	1	0	0	1	1
d	0	0	1	0	0
e	0	0	0	0	0

- $a \rightarrow \{b\}$
- $b \rightarrow \{d\}$
- $c \rightarrow \{a, d, e\}$
- $d \rightarrow \{c\}$
- $e \rightarrow \emptyset$

Concepts (1)

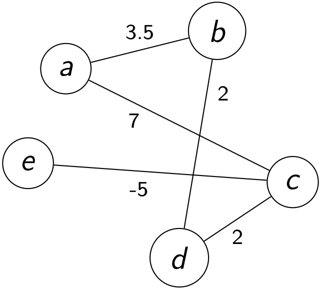


Directed graph



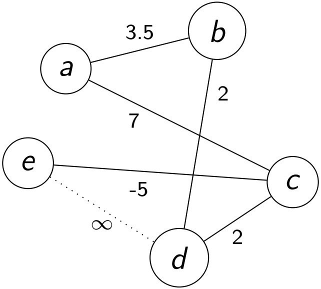
Undirected graph

Concepts (2)



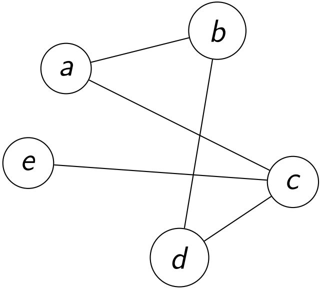
Weighted graph

Concepts (2)



Weighted graph

Concepts (3)



Node degree

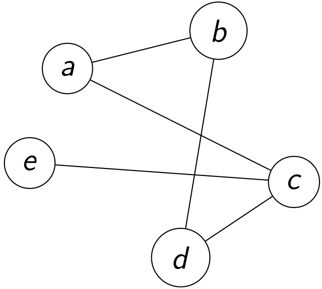
$a.deg = 2$

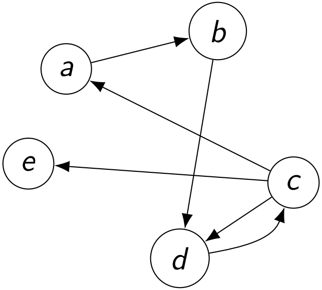
$b.deg = 2$

$c.deg = 3$

...

Concepts (3)





Node degree

$a.deg = 2$

$b.deg = 2$

$c.deg = 3$

...

In–degree and out–degree

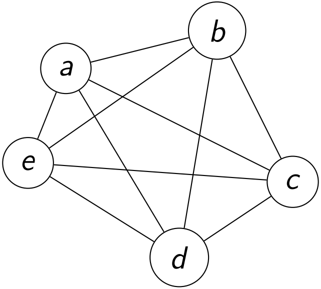
$a.deg_{in} = 1, a.deg_{out} = 1$

$c.deg_{in} = 1, c.deg_{out} = 3$

$e.deg_{in} = 1, e.deg_{out} = 0$

...

Concepts (4)



Complete graph

$$E = \{(u, v) \in V \times V, u \neq v\}$$

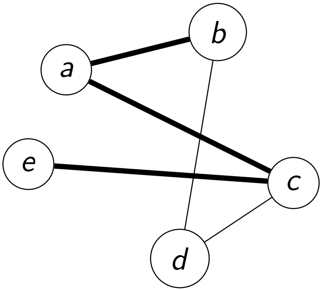
Concepts (5)

The distance between two nodes

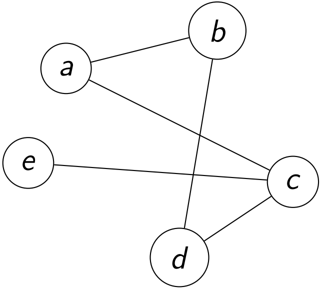
The shortest path (in number of edges) between the two nodes.

Graph diameter

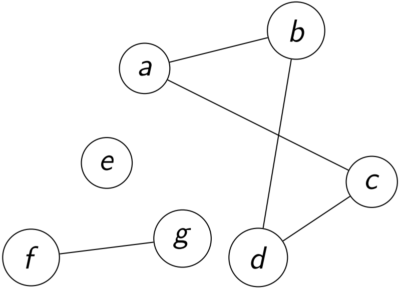
The longest distance between (any) two nodes in the graph.



Concepts (6)

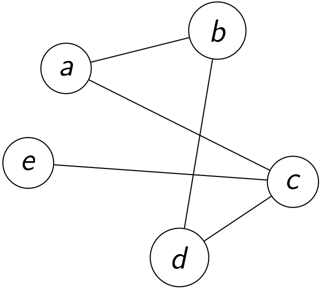


Connected graph

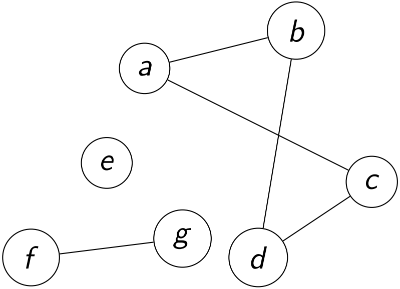


Unconnected graph

Concepts (6)



Connected graph



Unconnected graph

Strongly connected graph

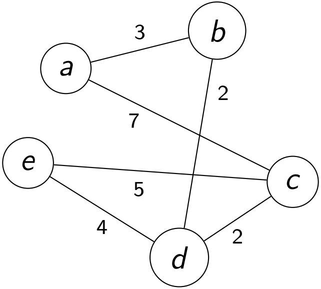
Directed graph, in which we have a path from any node to any other node, considering the edge direction.

Agenda

1. Graph Representation
2. Minimum Spanning Tree (MST)
   - Kruskal's Algorithm
   - Prim's Algorithm
3. Breadth-First Search (BFS)

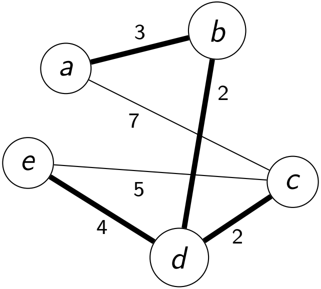
Minimum Spanning Tree (MST)

- Given a weighted, undirected, graph $G = (V, E)$, $w : V \times V \to \mathbb{R}$
  - $w(u, v)$ - weight/cost of the edge between u and v
- Find $G' = (V, T)$
  - $T \subseteq E$, $|T| = |V| - 1$, T contains no cycle
  - $\sum_{(u,v) \in T} w(u, v)$ is minimum



Minimum Spanning Tree (MST)

- Given a weighted, undirected, graph $G = (V, E)$, $w : V \times V \rightarrow \mathbb{R}$
  - $w(u, v)$ - weight/cost of the edge between u and v
- Find $G' = (V, T)$
  - $T \subseteq E$, $|T| = |V| - 1$, T contains no cycle
  - $\sum_{(u,v) \in T} w(u, v)$ is minimum



MST Applications

- network design
  - phone, electrical, hidraulic, computer, road

MST Applications

- network design
  - phone, electrical, hidraulic, computer, road
- approximation algorithms for NP-hard problems
  - TSP (*traveling salesman problem*)
  - Steiner trees

MST Applications

- network design
  - phone, electrical, hidraulic, computer, road
- approximation algorithms for NP-hard problems
  - TSP (*traveling salesman problem*)
  - Steiner trees
- indirect applications
  - max bottleneck path
  - LDPC codes for transmission error correction
  - feature learning in face recognition
  - reducing storage space in protein amino-acid sequencing
  - Ethernet auto-configuration for cycle avoidance

Kruskal's Algorithm - Approach

- *greedy* approach
- build the spanning tree by repeatedly linking partial trees
- initially, each node is a tree
- in each step, link the two closest sub-trees
  - by an edge which will be added to the MST
  - the sub-trees should be distinct (the edge must not close a cycle)
- stop when we have selected $|V| - 1$ edges
- how can we know that an edge links two different trees, and not two different branches of the same tree? (i.e. cycle)
  - use a disjoint set forest

Kruskal's Algorithm

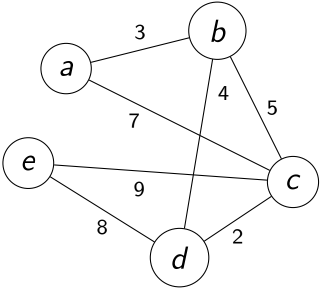
MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET(v)
sort the edges $G.E$ by weight w
for each edge $(u, v) \in G.E$
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)
return A

Kruskal's Algorithm

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```



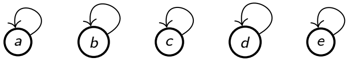
$$G.E = \{(a,b)_3, (a,c)_7, (b,c)_5, (b,d)_4, (c,d)_2, (c,e)_9, (d,e)_8\}$$

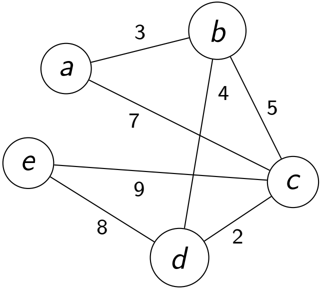
$$A = \{\}$$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```



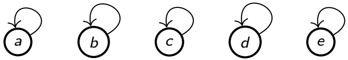


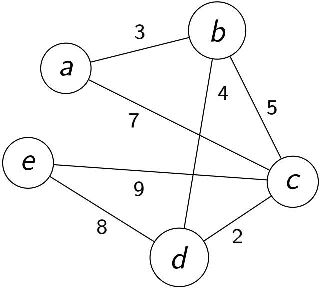
$$G.E = \{(a,b)_3, (a,c)_7, (b,c)_5, (b,d)_4, (c,d)_2, (c,e)_9, (d,e)_8\}$$

$$A = \{\}$$

Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```





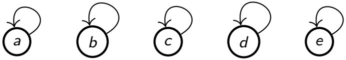
$$G.E = \{(c,d)_2, (a,b)_3, (b,d)_4, (b,c)_5, (a,c)_7, (d,e)_8, (c,e)_9\}$$

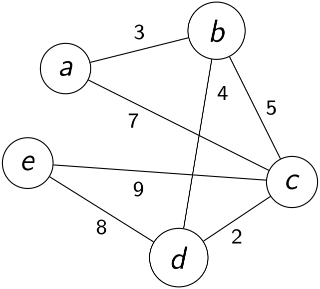
$$A = \{\}$$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET(v)
sort the edges $G.E$ by weight w
for each edge $(u, v) \in G.E$
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)
return A





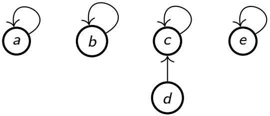
$$G.E = \{(c, d)_2, (a, b)_3, (b, d)_4, (b, c)_5, (a, c)_7, (d, e)_8, (c, e)_9\}$$

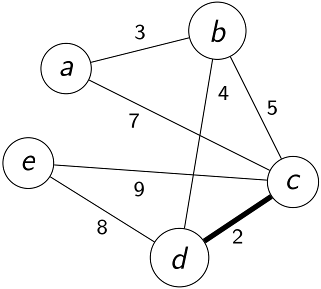
$$A = \{\}$$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```



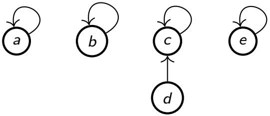


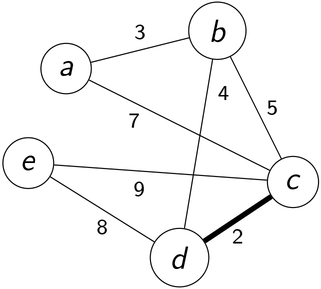
$$G.E = \{(c, d)_2, (a, b)_3, (b, d)_4, (b, c)_5, (a, c)_7, (d, e)_8, (c, e)_9\}$$

$$A = \{(c, d)\}$$

Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
      MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
      if FIND-SET(u) ≠ FIND-SET(v)
            A = A ∪ {(u, v)}
            UNION(u, v)
return A
```





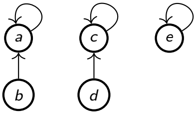
$G.E = \{(c,d)_2, (a,b)_3, (b,d)_4, (b,c)_5, (a,c)_7, (d,e)_8, (c,e)_9\}$

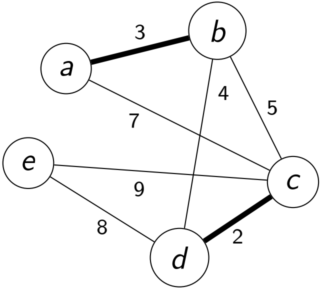
$A = \{(c,d)\}$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
  MAKE-SET(v)
sort the edges $G.E$ by weight w
for each edge $(u, v) \in G.E$
  if FIND-SET$(u) \neq$ FIND-SET(v)
    $A = A \cup \{(u, v)\}$
    UNION(u, v)
return A





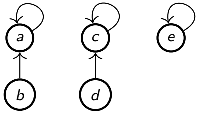
$$G.E = \{(c, d)_2, (a, b)_3, (b, d)_4, (b, c)_5, (a, c)_7, (d, e)_8, (c, e)_9\}$$

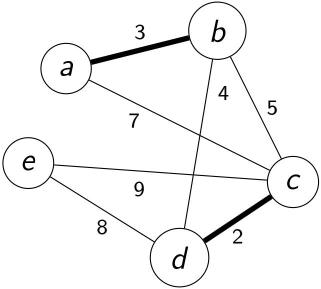
$$A = \{(c, d), (a, b)\}$$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```





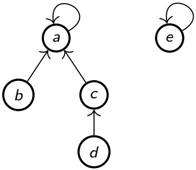
$$G.E = \{(c,d)_2, (a,b)_3, (b,d)_4, (b,c)_5, (a,c)_7, (d,e)_8, (c,e)_9\}$$

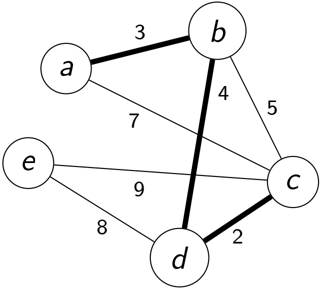
$$A = \{(c,d), (a,b)\}$$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```





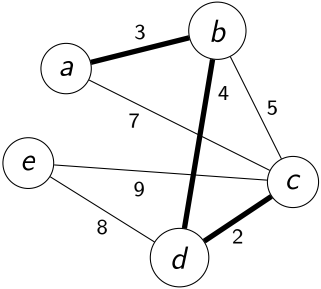
$$G.E = \{(c, d)_2, (a, b)_3, (b, d)_4, (b, c)_5, (a, c)_7, (d, e)_8, (c, e)_9\}$$

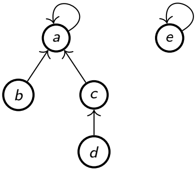
$$A = \{(c, d), (a, b), (b, d)\}$$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
      MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
      if FIND-SET(u) ≠ FIND-SET(v)
            A = A ∪ {(u, v)}
            UNION(u, v)
return A
```





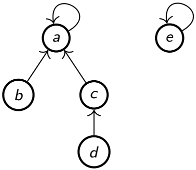
$$G.E = \{(c,d)_2, (a,b)_3, (b,d)_4, (b,c)_5, (a,c)_7, (d,e)_8, (c,e)_9\}$$

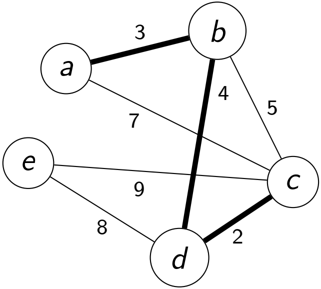
$$A = \{(c,d), (a,b), (b,d)\}$$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```





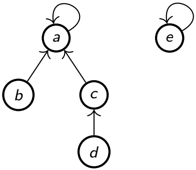
$G.E = \{(c,d)_2, (a,b)_3, (b,d)_4, (b,c)_5, (a,c)_7, (d,e)_8, (c,e)_9\}$

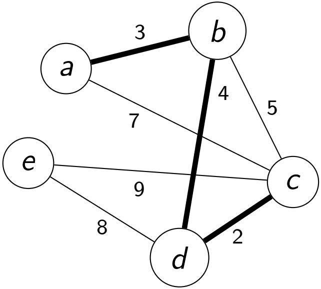
$A = \{(c,d), (a,b), (b,d)\}$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```





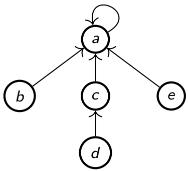
$$G.E = \{(c,d)_2, (a,b)_3, (b,d)_4, (b,c)_5, (a,c)_7, (d,e)_8, (c,e)_9\}$$

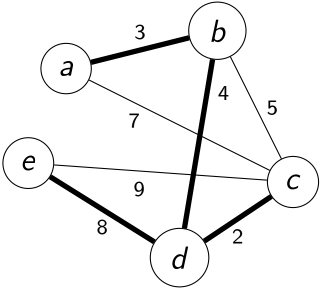
$$A = \{(c,d), (a,b), (b,d)\}$$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
      MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
      if FIND-SET(u) ≠ FIND-SET(v)
            A = A ∪ {(u, v)}
            UNION(u, v)
return A
```





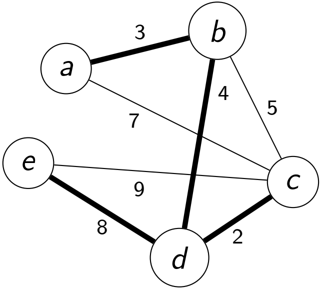
$$G.E = \{(c,d)_2, (a,b)_3, (b,d)_4, (b,c)_5, (a,c)_7, (d,e)_8, (c,e)_9\}$$

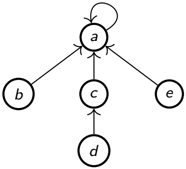
$$A = \{(c,d), (a,b), (b,d), (d,e)\}$$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET(v)
sort the edges $G.E$ by weight w
for each edge $(u, v) \in G.E$
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)
return A





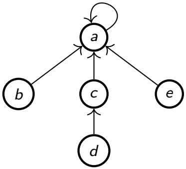
$$G.E = \{(c, d)_2, (a, b)_3, (b, d)_4, (b, c)_5, (a, c)_7, (d, e)_8, (c, e)_9\}$$

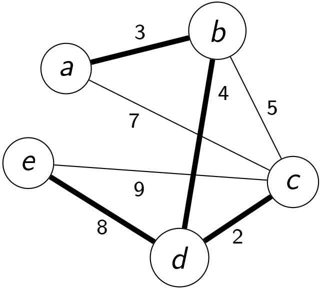
$$A = \{(c, d), (a, b), (b, d), (d, e)\}$$

Kruskal's Algorithm

MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET(v)
sort the edges $G.E$ by weight w
for each edge $(u, v) \in G.E$
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)
return A





$$G.E = \{(c,d)_2, (a,b)_3, (b,d)_4, (b,c)_5, (a,c)_7, (d,e)_8, (c,e)_9\}$$

$$A = \{(c,d), (a,b), (b,d), (d,e)\}$$

Kruskal's Algorithm - Complexity

MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET(v)
sort the edges $G.E$ by weight w
for each edge $(u, v) \in G.E$
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)
return A

Kruskal's Algorithm - Complexity

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
      MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
      if FIND-SET(u) ≠ FIND-SET(v)
            A = A ∪ {(u, v)}
            UNION(u, v)
return A
```

$O(V)$

Kruskal's Algorithm - Complexity

MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET(v) $O(V)$
sort the edges $G.E$ by weight w $O(E \log E)$
for each edge $(u, v) \in G.E$
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)
return A

Kruskal's Algorithm - Complexity

MST-KRUSKAL(G, w)

$A = \emptyset$

for each vertex $v \in G.V$
    MAKE-SET(v) $O(V)$

sort the edges $G.E$ by weight w $O(E \log E)$

for each edge $(u, v) \in G.E$ $O(E)$
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v) $\times O(\alpha(V))$

return A

Kruskal's Algorithm - Complexity

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
        MAKE-SET(v)
sort the edges G.E by weight w
for each edge (u, v) ∈ G.E
        if FIND-SET(u) ≠ FIND-SET(v)
                A = A ∪ {(u, v)}
                UNION(u, v)
return A
```

$O(V)$

$O(E \log E)$

$O(E)$

$\times O(\alpha(V))$

Complexity: $O(E \log E)$

Kruskal's Algorithm - Disscussion

- Is the solution unique?
  - on the given example
  - in general

Kruskal's Algorithm - Disscussion

- Is the solution unique?
  - on the given example
  - in general
- How can we obtain other solutions?

Kruskal's Algorithm - Disscussion

- Is the solution unique?
  - on the given example
  - in general
- How can we obtain other solutions?
- How can we obtain all solutions?
  - How many solutions are there, in the worst case?

Kruskal's Algorithm - Correctness

- Let T be the MST built by MST-KRUSKAL.
- Consider $G.E = \{e_1, e_2, \ldots, e_m\}$ the edges of the graph, ordered ascendingly by weight.

Kruskal's Algorithm - Correctness

- Let T be the MST built by MST-KRUSKAL.
- Consider $G.E = \{e_1, e_2, \ldots, e_m\}$ the edges of the graph, ordered ascendingly by weight.
- Assume that T is not minimal, and that $T^\star$ is the MST $(w(T^\star) < w(T))$ having the longest common edge prefix with T.

Kruskal's Algorithm - Correctness

- Let T be the MST built by MST-KRUSKAL.
- Consider $G.E = \{e_1, e_2, \ldots, e_m\}$ the edges of the graph, ordered ascendingly by weight.
- Assume that T is not minimal, and that $T^\star$ is the MST ($w(T^\star) < w(T)$) having the longest common edge prefix with T.
- Let $e \in G.E$ be the first edge (considering their order) which doesn't belong to $T.E \cap T^\star.E$

Kruskal's Algorithm - Correctness

- Let T be the MST built by MST-KRUSKAL.
- Consider $G.E = \{e_1, e_2, \ldots, e_m\}$ the edges of the graph, ordered ascendingly by weight.
- Assume that T is not minimal, and that $T^\star$ is the MST ($w(T^\star) < w(T)$) having the longest common edge prefix with T.
- Let $e \in G.E$ be the first edge (considering their order) which doesn't belong to $T.E \cap T^\star.E$
  - Kruskal would not have excluded that edge, unless it closed a cycle
  - $\Rightarrow e \in T.E$ and $e \notin T^\star.E$

Kruskal's Algorithm - Correctness

- Let T be the MST built by MST-KRUSKAL.
- Consider $G.E = \{e_1, e_2, \ldots, e_m\}$ the edges of the graph, ordered ascendingly by weight.
- Assume that T is not minimal, and that $T^\star$ is the MST $(w(T^\star) < w(T))$ having the longest common edge prefix with T.
- Let $e \in G.E$ be the first edge (considering their order) which doesn't belong to $T.E \cap T^\star.E$
  - Kruskal would not have excluded that edge, unless it closed a cycle
  - $\Rightarrow e \in T.E$ and $e \notin T^\star.E$
- If $e = (x, y)$, there is a path in $T^\star$, P, from x to y (which cannot be the direct edge, because, $(x, y) \notin T^\star.E$).
- If all edges in P had a weight smaller than e, Kruskal-MST would have selected them prior to e (e is the first edge for which T and $T^\star$ "disagree"), and e would close a cycle.
- $\Rightarrow \exists e' \in P, w(e) \leq w(e')$.

Kruskal's Algorithm - Correctness

- Let T be the MST built by MST-KRUSKAL.
- Consider $G.E = \{e_1, e_2, \ldots, e_m\}$ the edges of the graph, ordered ascendingly by weight.
- Assume that T is not minimal, and that $T^\star$ is the MST ($w(T^\star) < w(T)$) having the longest common edge prefix with T.
- Let $e \in G.E$ be the first edge (considering their order) which doesn't belong to $T.E \cap T^\star.E$
    - Kruskal would not have excluded that edge, unless it closed a cycle
    - $\Rightarrow e \in T.E$ and $e \notin T^\star.E$
- If $e = (x, y)$, there is a path in $T^\star$, P, from x to y (which cannot be the direct edge, because, $(x, y) \notin T^\star.E$).
- If all edges in P had a weight smaller than e, Kruskal-MST would have selected them prior to e (e is the first edge for which T and $T^\star$ "disagree"), and e would close a cycle.
- $\Rightarrow \exists e' \in P, w(e) \leq w(e')$.
- Let $T_1 = T^\star \cup \{e\} \setminus \{e'\}$. $\Rightarrow w(T_1) \leq w(T^\star)$.
- T_1 is a MST with a longer common prefix with T than $T^\star$ - contradiction.

Prim's Algorithm - Approach

- *greedy* approach
- start at any node
- in every step (out of the $|V| - 1$ steps)
  - select the closest node to the current tree
  - add the node and the corresponding edge to the tree

Prim's Algorithm

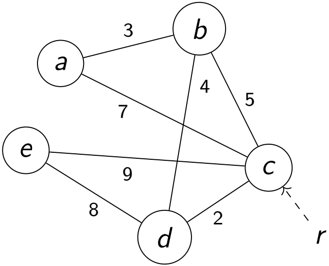
MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

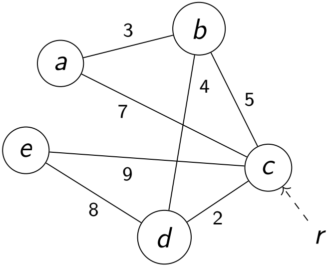


	π	key

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



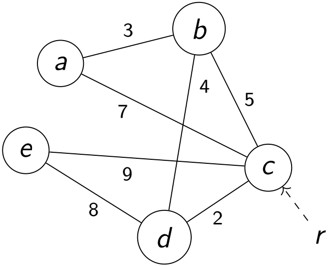
	π	key
a	NIL	∞
b	NIL	∞
c	NIL	∞
d	NIL	∞
e	NIL	∞

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

c



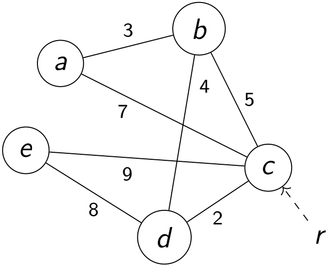
	π	key
a	NIL	∞
b	NIL	∞
c	NIL	0
d	NIL	∞
e	NIL	∞

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

c



	π	*key*
c	NIL	0
a	NIL	∞
b	NIL	∞
d	NIL	∞
e	NIL	∞

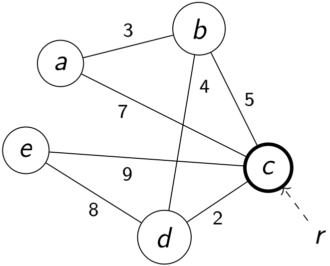
Q

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

c



	π	key
c	NIL	0
a	NIL	∞
b	NIL	∞
d	NIL	∞
e	NIL	∞

$V \setminus Q$

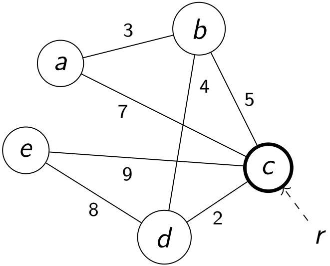
Q

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

c



	π	key
c	NIL	0
a	NIL	∞
b	NIL	∞
d	NIL	∞
e	NIL	∞

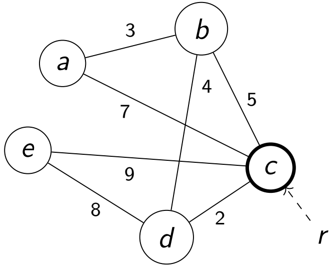
$V \setminus Q$

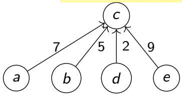
Q

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



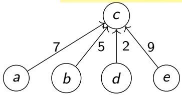


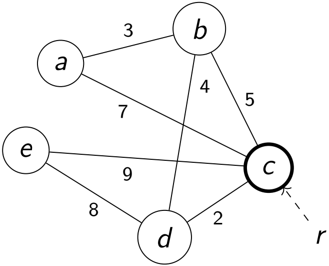
	π	key	
c	NIL	0	$V \setminus Q$
a	c	7	Q
b	c	5	
d	c	2	
e	c	9	

Prim's Algorithm

MST-PRIM(G, w, r)

for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$ // priority queue
while $Q \neq \emptyset$
    $u = \text{EXTRACT-MIN}(Q)$
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$



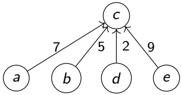


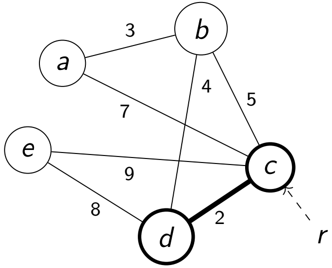
	π	key	
c	NIL	0	$V \setminus Q$
d	c	2	Q
b	c	5	
a	c	7	
e	c	9	

Prim's Algorithm

$\text{MST-PRIM}(G, w, r)$

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



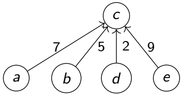


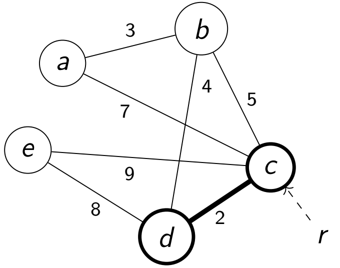
	π	key	
c	NIL	0	$V \setminus Q$
d	c	2	
b	c	5	Q
a	c	7	
e	c	9	

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```





	π	key
c	NIL	0
d	c	2
b	c	5
a	c	7
e	c	9

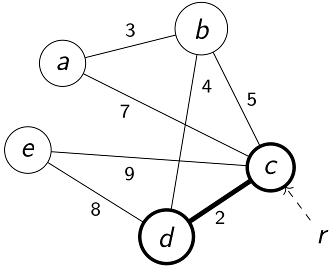
$V \setminus Q$: rows c, d

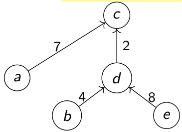
Q: rows b, a, e

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



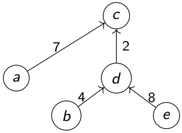


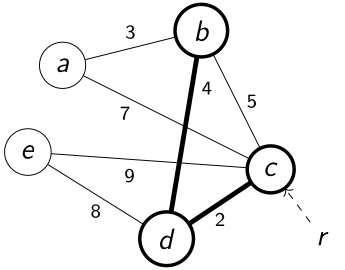
	π	key	
c	NIL	0	$V \setminus Q$
d	c	2	
b	d	4	Q
a	c	7	
e	d	8	

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```





	π	key
c	NIL	0
d	c	2
b	d	4
a	c	7
e	d	8

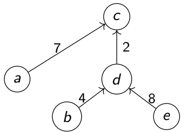
$V \setminus Q$ (rows c, d, b)

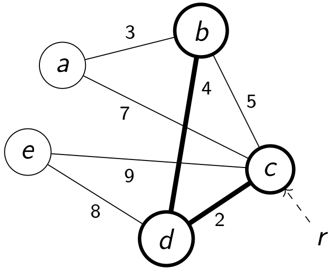
Q (rows a, e)

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```





	π	key
c	NIL	0
d	c	2
b	d	4
a	c	7
e	d	8

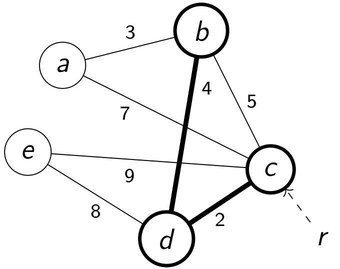
$V \setminus Q$ (rows c, d, b)

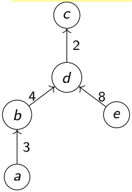
Q (rows a, e)

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```





	π	key
c	NIL	0
d	c	2
b	d	4
a	b	3
e	d	8

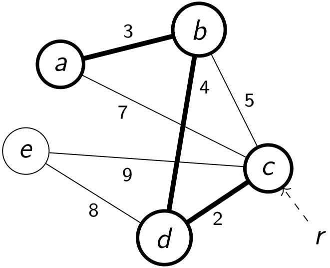
$V \setminus Q$: rows c, d, b

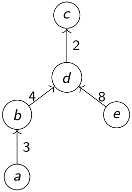
Q: rows a, e

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V  // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```





	π	key
c	NIL	0
d	c	2
b	d	4
a	b	3
e	d	8

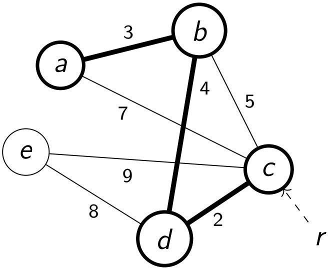
$V \setminus Q$ (rows c, d, b, a)

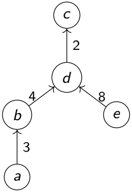
Q (row e)

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



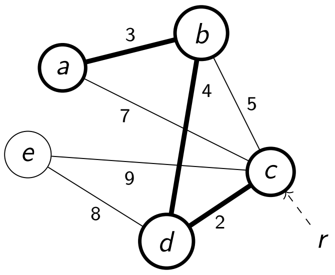


	π	*key*	
c	NIL	0	$V \setminus Q$
d	c	2	
b	d	4	
a	b	3	
e	d	8	Q

Prim's Algorithm

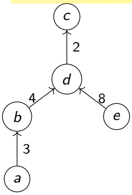
MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```



	π	key
c	NIL	0
d	c	2
b	d	4
a	b	3
e	d	8

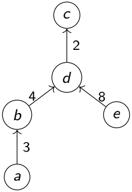
$V \setminus Q$ (rows c, d, b, a); Q (row e)

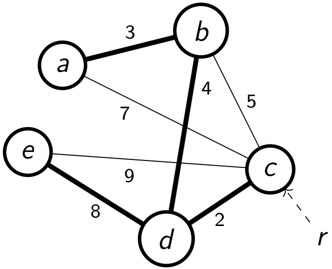


Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V  // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```





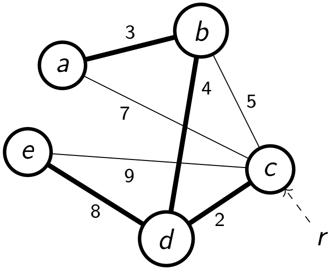
	π	key
c	NIL	0
d	c	2
b	d	4
a	b	3
e	d	8

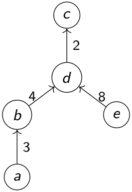
$V \setminus Q$

Prim's Algorithm

MST-PRIM(G, w, r)

for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$ // priority queue
while $Q \neq \emptyset$
    $u = \text{EXTRACT-MIN}(Q)$
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$





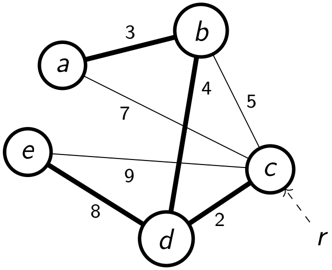
	π	key
c	NIL	0
d	c	2
b	d	4
a	b	3
e	d	8

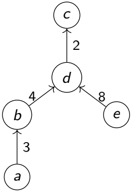
$V \setminus Q$

Prim's Algorithm

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```





	π	*key*
c	NIL	0
d	c	2
b	d	4
a	b	3
e	d	8

$V \setminus Q$

Prim's Algorithm - Complexity

```
MST-PRIM(G, w, r)
for each u ∈ G.V
        u.key = ∞
        u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
                if v ∈ Q and w(u, v) < v.key
                        v.π = u
                        v.key = w(u, v)
```

Prim's Algorithm - Complexity

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

$O(V)$

Prim's Algorithm - Complexity

MST-PRIM(G, w, r)

for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$ // priority queue
while $Q \neq \emptyset$
    $u = \text{EXTRACT-MIN}(Q)$
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$

$O(V)$

$O(V)$ - build heap

Prim's Algorithm - Complexity

MST-PRIM(G, w, r)

for each $u \in G.V$
  $u.key = \infty$
  $u.\pi = \text{NIL}$ $O(V)$
$r.key = 0$
$Q = G.V$ // priority queue $O(V)$ - build heap
while $Q \neq \emptyset$ $|V|$ steps
  $u = \text{EXTRACT-MIN}(Q)$
  for each $v \in G.Adj[u]$
    if $v \in Q$ and $w(u, v) < v.key$
      $v.\pi = u$
      $v.key = w(u, v)$

Prim's Algorithm - Complexity

MST-PRIM(G, w, r)

for each $u \in G.V$
  $u.key = \infty$
  $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$ // priority queue
while $Q \neq \emptyset$
  $u = \text{EXTRACT-MIN}(Q)$
  for each $v \in G.Adj[u]$
    if $v \in Q$ and $w(u, v) < v.key$
      $v.\pi = u$
      $v.key = w(u, v)$

$O(V)$

$O(V)$ - build heap

$|V|$ steps

$O(\log V)$

Prim's Algorithm - Complexity

MST-PRIM(G, w, r)

for each $u \in G.V$
    $u.key = \infty$
    $u.\pi =$ NIL
$r.key = 0$
$Q = G.V$ // priority queue
while $Q \neq \emptyset$
    $u =$ EXTRACT-MIN(Q)
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$

$O(V)$

$O(V)$ - build heap

$|V|$ steps

$O(\log V)$

each neighbor for every node, so $|E|$ steps in total

Prim's Algorithm - Complexity

MST-PRIM(G, w, r)

for each $u \in G.V$
  $u.key = \infty$
  $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$ // priority queue
while $Q \neq \emptyset$
  $u = \text{EXTRACT-MIN}(Q)$
  for each $v \in G.Adj[u]$
    if $v \in Q$ and $w(u, v) < v.key$
      $v.\pi = u$
      $v.key = w(u, v)$

$O(V)$

$O(V)$ - build heap

$|V|$ steps

$O(\log V)$

each neighbor
for every node,
so $|E|$ steps in total

binary heap: $O(\log V)$

Fibonacci heap:
$O(1)$ amortized

Prim's Algorithm - Complexity

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V // priority queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

$O(V)$

$O(V)$ - build heap

$|V|$ steps

$O(\log V)$

each neighbor for every node, so $|E|$ steps in total

binary heap: $O(\log V)$

Fibonacci heap: $O(1)$ amortized

binary heap implementation: $O(V + V + V \log V + E \log V) = O(E \log V)$

Prim's Algorithm - Complexity

MST-PRIM(G, w, r)

for each $u \in G.V$	
$\quad u.key = \infty$	$O(V)$
$\quad u.\pi = \text{NIL}$	
$r.key = 0$	
$Q = G.V$ // priority queue	$O(V)$ - build heap
while $Q \neq \emptyset$	$\lvert V \rvert$ steps
$\quad u = \text{EXTRACT-MIN}(Q)$	$O(\log V)$
$\quad$ **for** each $v \in G.Adj[u]$	each neighbor for every node, so $\lvert E \rvert$ steps in total
$\quad\quad$ **if** $v \in Q$ and $w(u, v) < v.key$	
$\quad\quad\quad v.\pi = u$	
$\quad\quad\quad v.key = w(u, v)$	binary heap: $O(\log V)$ Fibonacci heap: $O(1)$ amortized

binary heap implementation: $O(V + V + V \log V + E \log V) = O(E \log V)$
Fibonacci heap implementation: $O(V + V + V \log V + E) = O(E + V \log V)$

Prim's Algorithm - Disscussion

- Advantage over Kruskal's algorithm
  - builds a rooted tree
  - more efficient in the Fibonacci heap implementation

Prim's Algorithm - Disscussion

- Advantage over Kruskal's algorithm
  - builds a rooted tree
  - more efficient in the Fibonacci heap implementation
- Is the solution unique?

Prim's Algorithm - Disscussion

- Advantage over Kruskal's algorithm
  - builds a rooted tree
  - more efficient in the Fibonacci heap implementation
- Is the solution unique?
- Is the algorithm deterministic?

Prim's Algorithm - Correctness

- Let T be the MST build by MST-PRIM.
- Let $T.E = \{e_1, e_2, \ldots, e_{n-1}\}$ be the sequence of edges, in the order they are selected by the algorithm.

Prim's Algorithm - Correctness

- Let T be the MST build by MST-PRIM.
- Let $T.E = \{e_1, e_2, \ldots, e_{n-1}\}$ be the sequence of edges, in the order they are selected by the algorithm.
- Assume that T is not minimal, and that $T^\star$ is the MST, $(w(T^\star) < w(T))$, which includes the longest prefix $E' = \{e_1, e_2, \ldots, e_{i-1}\}$ from $T.E$, while the edge $e_i = (x, y) \notin T^\star.E$. We denote by T' the nodes added to T before the edge e_i.

Prim's Algorithm - Correctness

- Let T be the MST build by MST-PRIM.
- Let $T.E = \{e_1, e_2, \ldots, e_{n-1}\}$ be the sequence of edges, in the order they are selected by the algorithm.
- Assume that T is not minimal, and that $T^\star$ is the MST, $(w(T^\star) < w(T))$, which includes the longest prefix $E' = \{e_1, e_2, \ldots, e_{i-1}\}$ from $T.E$, while the edge $e_i = (x, y) \notin T^\star.E$. We denote by T' the nodes added to T before the edge e_i.
- In $T^\star$ there is a path from x to y. As $y \notin T'$, there is in the path from x to y an edge (a, b) with $a \in T'$ and $b \notin T'$.
- Let $T_1 = T^\star \cup \{(x, y)\} \setminus \{(a, b)\}$ - also a spanning tree.

Prim's Algorithm - Correctness

- Let T be the MST build by MST-PRIM.
- Let $T.E = \{e_1, e_2, \ldots, e_{n-1}\}$ be the sequence of edges, in the order they are selected by the algorithm.
- Assume that T is not minimal, and that $T^\star$ is the MST, $(w(T^\star) < w(T))$, which includes the longest prefix $E' = \{e_1, e_2, \ldots, e_{i-1}\}$ from $T.E$, while the edge $e_i = (x, y) \notin T^\star.E$. We denote by T' the nodes added to T before the edge e_i.
- In $T^\star$ there is a path from x to y. As $y \notin T'$, there is in the path from x to y an edge (a, b) with $a \in T'$ and $b \notin T'$.
- Let $T_1 = T^\star \cup \{(x, y)\} \setminus \{(a, b)\}$ - also a spanning tree.
- We have 3 cases:
  - if $w(x, y) < w(a, b)$, then $w(T_1) < w(T^\star) \Rightarrow T^\star$ is not minimal
  - if $w(x, y) > w(a, b)$, then MST-PRIM would have selected (a, b) instead of (x, y)
  - if $w(x, y) = w(a, b)$, then $w(T_1) = w(T^\star)$, so T_1 is minimal and contains a longer prefix of $T.E$

Prim's Algorithm - Correctness

- Let T be the MST build by MST-PRIM.
- Let $T.E = \{e_1, e_2, \ldots, e_{n-1}\}$ be the sequence of edges, in the order they are selected by the algorithm.
- Assume that T is not minimal, and that $T^\star$ is the MST, $(w(T^\star) < w(T))$, which includes the longest prefix $E' = \{e_1, e_2, \ldots, e_{i-1}\}$ from $T.E$, while the edge $e_i = (x, y) \notin T^\star.E$. We denote by T' the nodes added to T before the edge e_i.
- In $T^\star$ there is a path from x to y. As $y \notin T'$, there is in the path from x to y an edge (a, b) with $a \in T'$ and $b \notin T'$.
- Let $T_1 = T^\star \cup \{(x, y)\} \setminus \{(a, b)\}$ - also a spanning tree.
- We have 3 cases:
  - if $w(x, y) < w(a, b)$, then $w(T_1) < w(T^\star) \Rightarrow T^\star$ is not minimal
  - if $w(x, y) > w(a, b)$, then MST-PRIM would have selected (a, b) instead of (x, y)
  - if $w(x, y) = w(a, b)$, then $w(T_1) = w(T^\star)$, so T_1 is minimal and contains a longer prefix of $T.E$
- All three cases result in a contradiction, so $T^\star$ does not exist $\Rightarrow T$ is minimal.

Agenda

Breadth-First Search (BFS)

- start at a given (source) node s
- produce a tree rooted in s
- all reachable nodes at distance k from the source will be discovered prior to all those at distance $k + 1$
- the resulting tree contains the minimum length paths from s to all nodes reachable from s

BFS Algorithm - Approach

- associate the current attributes to the nodes:
    - *color*: WHITE, GRAY, BLACK
    - *d*: distance from the source *s*
    - π: parent node in BFS tree
- the algorithm starts by adding the source node to a queue and coloring it GRAY
- while the queue still has elements
    - extract the first node from it
    - color with GRAY all neighbors of this node that haven't yet been discovered, and add them to the queue
    - color the extracted node with BLACK

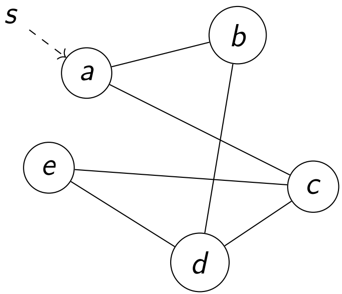
BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

BFS Algorithm

BFS(G, s)

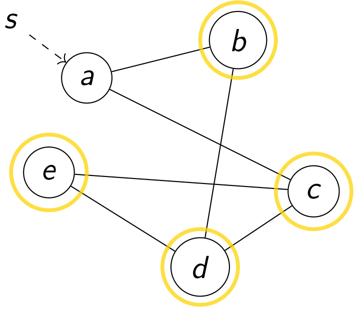
```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



BFS Algorithm

BFS(G, s)

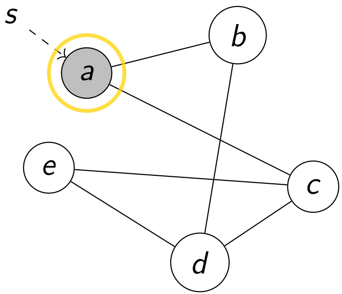
```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



BFS Algorithm

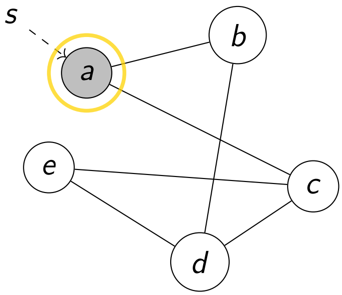
BFS(G, s)

```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



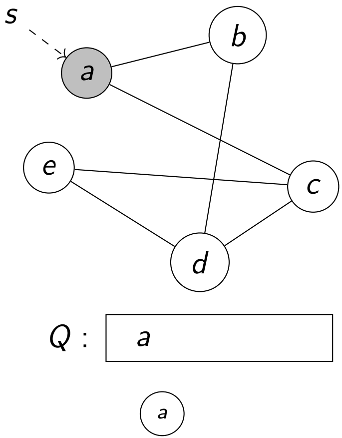
BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



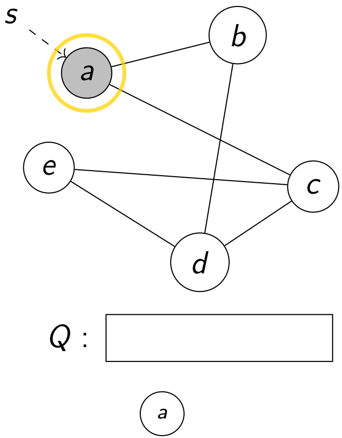
BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
                if v.color == WHITE
                        v.color = GRAY
                        v.d = u.d + 1
                        v.π = u
                        ENQUEUE(Q, v)
        u.color = BLACK
```



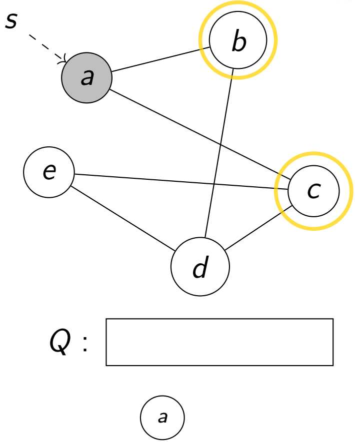
BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



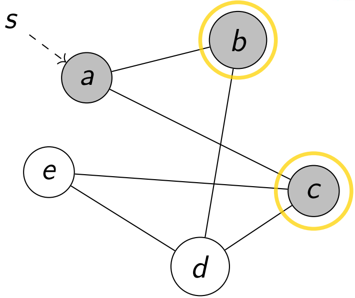
BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
                if v.color == WHITE
                        v.color = GRAY
                        v.d = u.d + 1
                        v.π = u
                        ENQUEUE(Q, v)
        u.color = BLACK
```

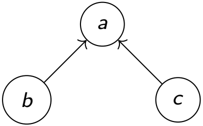


BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

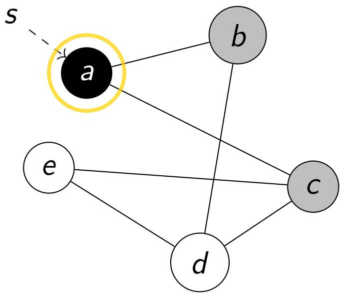


Q : $b \quad c$

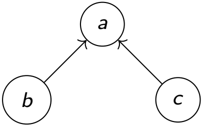


BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

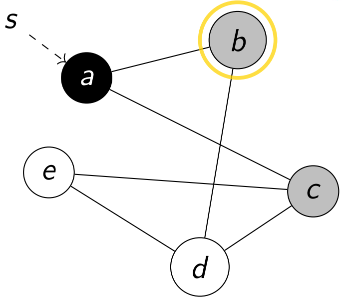


Q : b c

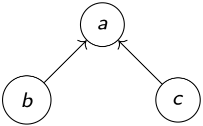


BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



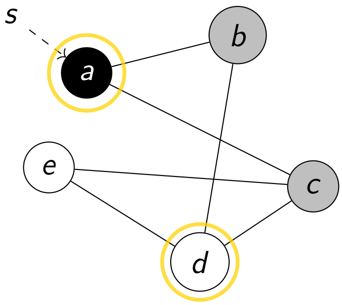
Q : c



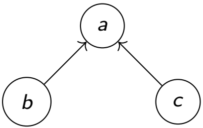
BFS Algorithm

BFS(G, s)

```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

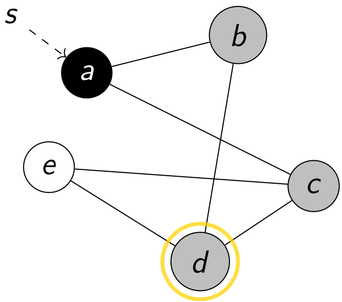


Q : c

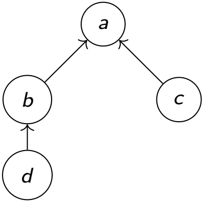


BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
                if v.color == WHITE
                        v.color = GRAY
                        v.d = u.d + 1
                        v.π = u
                        ENQUEUE(Q, v)
        u.color = BLACK
```

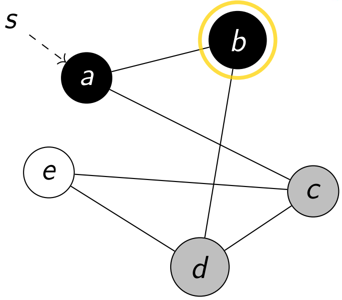


Q : $c \quad d$

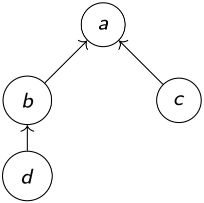


BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

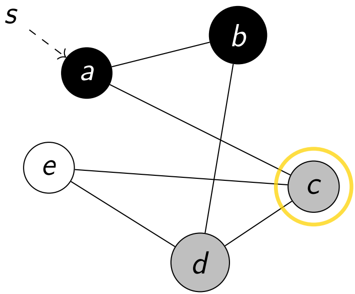


Q : c d

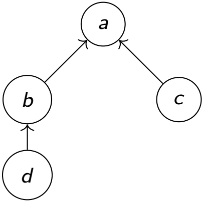


BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

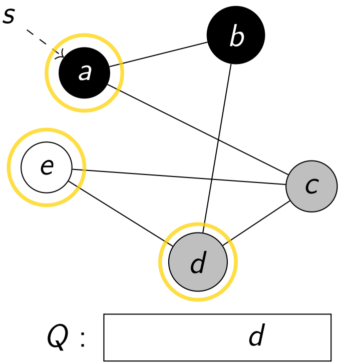


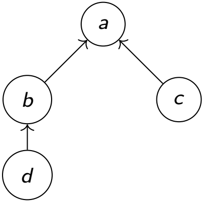
Q : | d |



BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

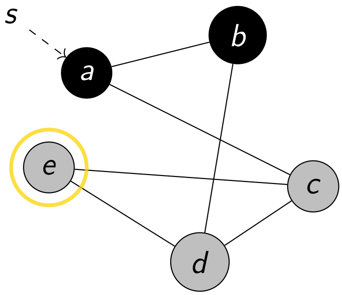




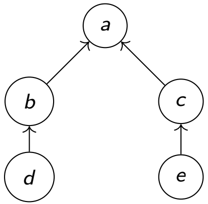
BFS Algorithm

BFS(G, s)

```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



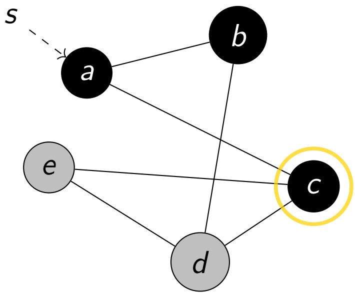
Q : d e



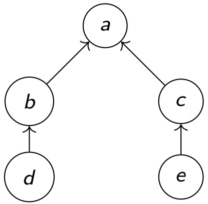
BFS Algorithm

BFS(G, s)

```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

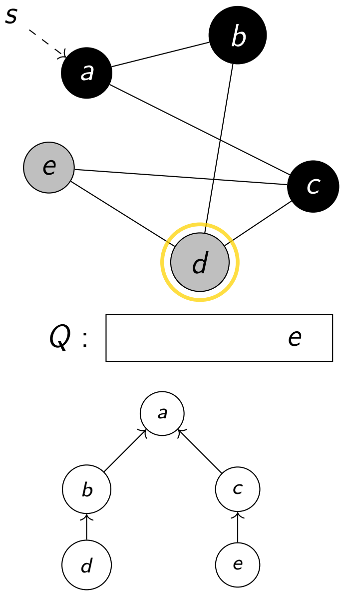


Q : d e



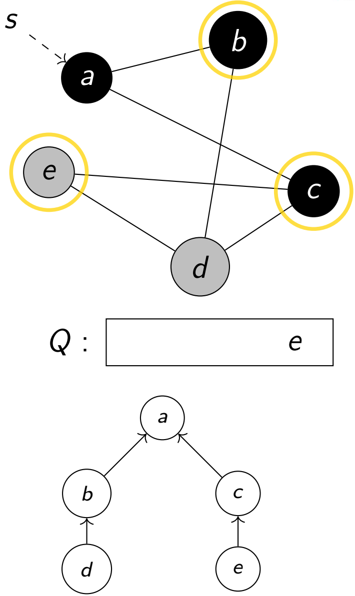
BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



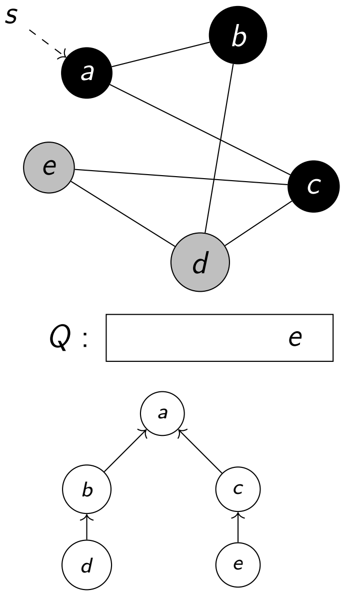
BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



BFS Algorithm

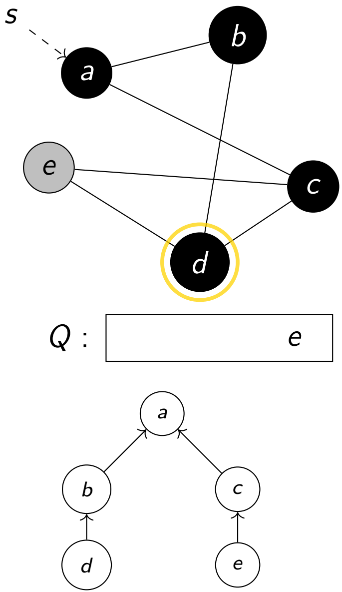
```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



BFS Algorithm

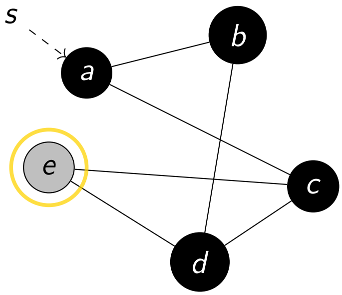
BFS(G, s)

```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

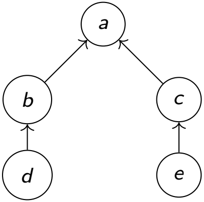


BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



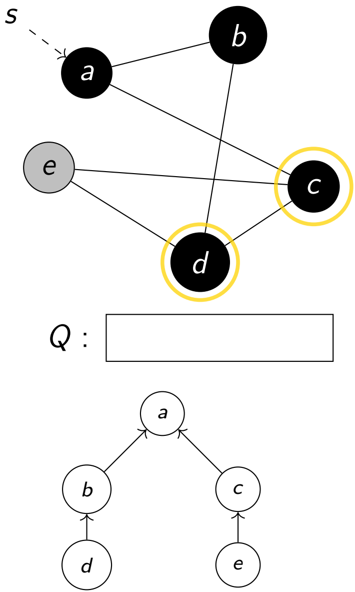
Q :



BFS Algorithm

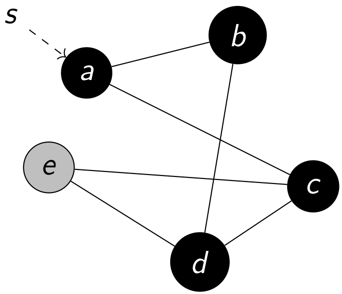
BFS(G, s)

```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

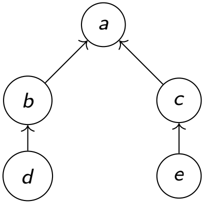


BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



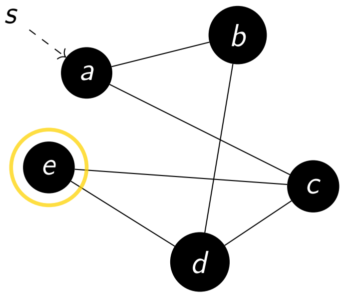
Q :



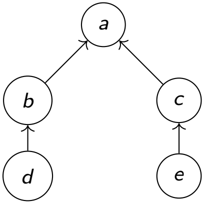
BFS Algorithm

BFS(G, s)

```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

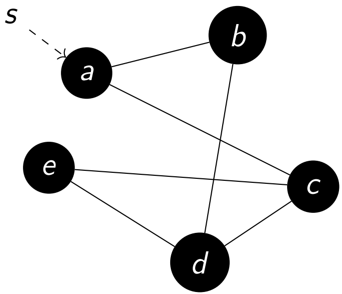


Q :

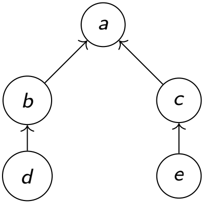


BFS Algorithm

```
BFS(G, s)
for each u ∈ G.V − {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
                if v.color == WHITE
                        v.color = GRAY
                        v.d = u.d + 1
                        v.π = u
                        ENQUEUE(Q, v)
        u.color = BLACK
```



Q :



BFS Algorithm - Complexity

```
BFS(G, s)
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

BFS Algorithm - Complexity

$\text{BFS}(G, s)$

for each $u \in G.V - \{s\}$
    $u.color = \text{WHITE}$
    $u.d = \infty$
    $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$ // regular queue
$\text{ENQUEUE}(Q, s)$
while $Q \neq \emptyset$
    $u = \text{DEQUEUE}(Q)$
    for each $v \in G.Adj[u]$
        if $v.color == \text{WHITE}$
            $v.color = \text{GRAY}$
            $v.d = u.d + 1$
            $v.\pi = u$
            $\text{ENQUEUE}(Q, v)$
    $u.color = \text{BLACK}$

$O(V)$

BFS Algorithm - Complexity

```
BFS(G, s)
for each u ∈ G.V − {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
                if v.color == WHITE
                        v.color = GRAY
                        v.d = u.d + 1
                        v.π = u
                        ENQUEUE(Q, v)
        u.color = BLACK
```

$O(V)$

$O(1)$

BFS Algorithm - Complexity

BFS(G, s)

```
for each u ∈ G.V − {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL                                  O(V)
s.color = GRAY
s.d = 0                                            O(1)
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅                                        |V| steps
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
                if v.color == WHITE
                        v.color = GRAY
                        v.d = u.d + 1
                        v.π = u
                        ENQUEUE(Q, v)
        u.color = BLACK
```

BFS Algorithm - Complexity

BFS(G, s)

```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

$O(V)$

$O(1)$

$|V|$ steps

each neighbor
for every node,
so $|E|$ steps in total

BFS Algorithm - Complexity

BFS(G, s)

```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

$O(V)$

$O(1)$

$|V|$ steps

each neighbor
for every node,
so $|E|$ steps in total

$O(1)$

BFS Algorithm - Complexity

BFS(G, s)

```
for each u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅ // regular queue
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Complexity: $O(V + E)$

$O(V)$

$O(1)$

$|V|$ steps

each neighbor
for every node,
so $|E|$ steps in total

$O(1)$

BFS Algorithm - Discussion

- How does the algorithm behave if the graph is not connected?

BFS Algorithm - Discussion

- How does the algorithm behave if the graph is not connected?
  - How do we modify the algorithm to completely traverse unconnected graphs?

BFS Algorithm - Discussion

- How does the algorithm behave if the graph is not connected?
  - How do we modify the algorithm to completely traverse unconnected graphs?
- What complexity would the algorithm have on an adjacency matrix graph representation?

BFS Algorithm - Discussion

- How does the algorithm behave if the graph is not connected?
  - How do we modify the algorithm to completely traverse unconnected graphs?
- What complexity would the algorithm have on an adjacency matrix graph representation?
- How can we use it do determine the diameter of a tree?

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

			-1	-1	-1	-1	
		-1					-1
	-1						-1
		-1					-1
		-1					-1
	-1		-1	-1	-1		-1
							-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \to k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1					-1
	-1						-1
		-1					-1
		-1					-1
	-1		-1	-1	-1		-1
							-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \to k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1					-1
	-1		1				-1
		-1					-1
		-1					-1
	-1		-1	-1	-1		-1
							-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \to k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1	2				-1
	-1	2	1	2			-1
		-1	2				-1
		-1					-1
	-1		-1	-1	-1		-1
							-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \to k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1	2	3			-1
	-1	2	1	2	3		-1
		-1	2	3			-1
		-1	3				-1
	-1		-1	-1	-1		-1
							-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
    - the neighbors of nodes $k \rightarrow k + 1$
    - $k = k + 1$

			-1	-1	-1	-1	
		-1	2	3	4		-1
	-1	2	1	2	3	4	-1
		-1	2	3	4		-1
		-1	3	4			-1
	-1		-1	-1	-1		-1
							-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \rightarrow k+1$
  - $k = k+1$

			-1	-1	-1	-1	
		-1	2	3	4	5	-1
	-1	2	1	2	3	4	-1
		-1	2	3	4	5	-1
		-1	3	4	5		-1
	-1		-1	-1	-1		-1
							-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \rightarrow k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1	2	3	4	5	-1
	-1	2	1	2	3	4	-1
		-1	2	3	4	5	-1
		-1	3	4	5	6	-1
	-1		-1	-1	-1		-1
							-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \rightarrow k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1	2	3	4	5	-1
	-1	2	1	2	3	4	-1
		-1	2	3	4	5	-1
		-1	3	4	5	6	-1
	-1		-1	-1	-1	7	-1
							-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \rightarrow k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1	2	3	4	5	-1
	-1	2	1	2	3	4	-1
		-1	2	3	4	5	-1
		-1	3	4	5	6	-1
	-1		-1	-1	-1	7	-1
						8	-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \to k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1	2	3	4	5	-1
	-1	2	1	2	3	4	-1
		-1	2	3	4	5	-1
		-1	3	4	5	6	-1
	-1		-1	-1	-1	7	-1
					9	8	-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \rightarrow k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1	2	3	4	5	-1
	-1	2	1	2	3	4	-1
		-1	2	3	4	5	-1
		-1	3	4	5	6	-1
	-1		-1	-1	-1	7	-1
				10	9	8	-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \rightarrow k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1	2	3	4	5	-1
	-1	2	1	2	3	4	-1
		-1	2	3	4	5	-1
		-1	3	4	5	6	-1
	-1		-1	-1	-1	7	-1
			11	10	9	8	-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \rightarrow k+1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1	2	3	4	5	-1
	-1	2	1	2	3	4	-1
		-1	2	3	4	5	-1
		-1	3	4	5	6	-1
	-1		-1	-1	-1	7	-1
		12	11	10	9	8	-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \rightarrow k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1	2	3	4	5	-1
	-1	2	1	2	3	4	-1
		-1	2	3	4	5	-1
		-1	3	4	5	6	-1
	-1	13	-1	-1	-1	7	-1
	13	12	11	10	9	8	-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \rightarrow k + 1$
  - $k = k + 1$

			-1	-1	-1	-1	
		-1	2	3	4	5	-1
	-1	2	1	2	3	4	-1
		-1	2	3	4	5	-1
		-1	3	4	5	6	-1
	-1	13	-1	-1	-1	7	-1
14	13	12	11	10	9	8	-1
	-1	-1	-1	-1	-1	-1	

Lee's Algorithm

- how do we find the closest exit in a maze?
- represent the maze as a matrix, in which 0 means free pass and -1 means wall
- we move using V4 or V8

- mark starting point with $k = 1$
- while we haven't reached the exit
  - the neighbors of nodes $k \rightarrow k + 1$
  - $k = k + 1$
- how do we rebuild the path?

			-1	-1	-1	-1	
		-1	2	3	4	5	-1
	-1	2	1	2	3	4	-1
		-1	2	3	4	5	-1
		-1	3	4	5	6	-1
	-1	13	-1	-1	-1	7	-1
14	13	12	11	10	9	8	-1
	-1	-1	-1	-1	-1	-1	

Bibliography

- Cormen, Thomas H., et al., *"Introduction to algorithms."*, MIT press, 2009, sec. 22.1, 22.2, ch. 23