

# Paths in Graphs: Most Direct Route

Michael Levin

Higher School of Economics

Graph Algorithms  
Data Structures and Algorithms

# Outline

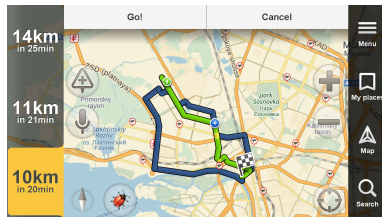
- 1 Paths and Distances
- 2 Breadth-first Search
- 3 Implementation and Analysis
- 4 Proof of Correctness
- 5 Shortest-path Tree

# Applications

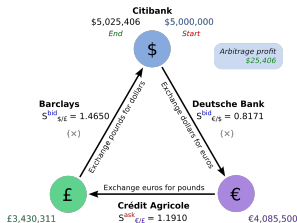
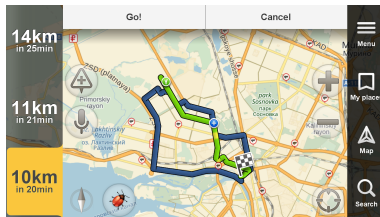
# Applications



# Applications



# Applications



By John Shandy - Own work, CC BY-SA 3.0

## The most direct route

What is the minimum number of flight segments to get from Hamburg to Moscow?

# The most direct route

What is the minimum number of flight segments to get from Hamburg to Moscow?

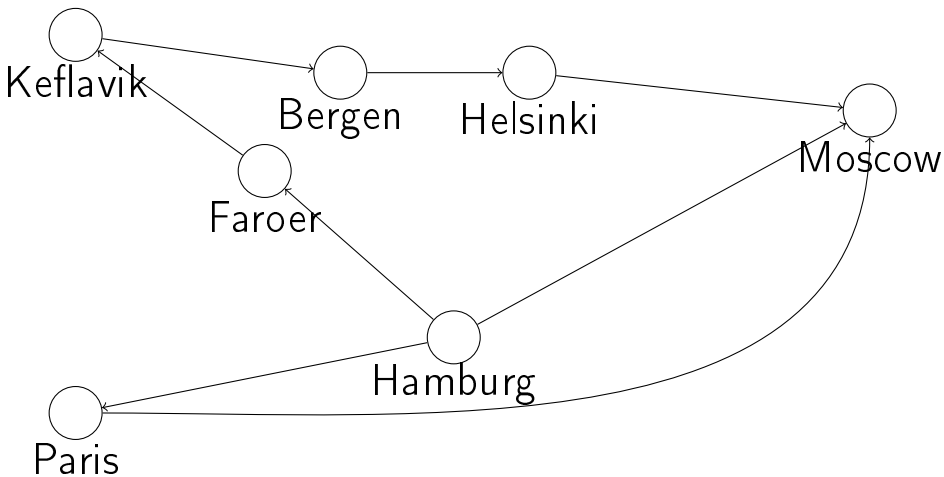


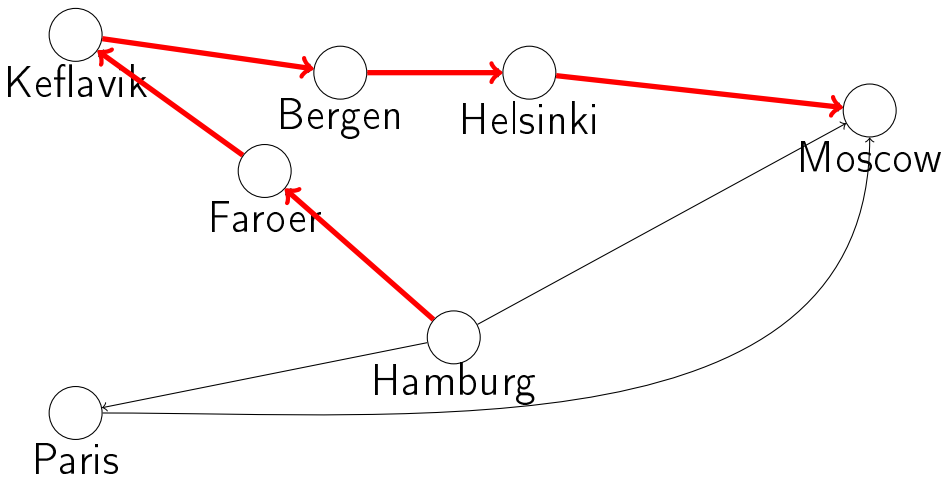


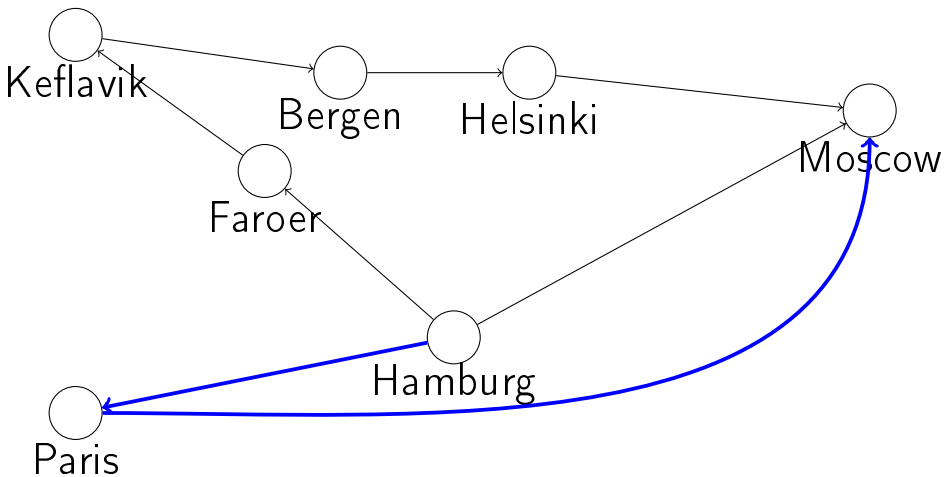
# The most direct route

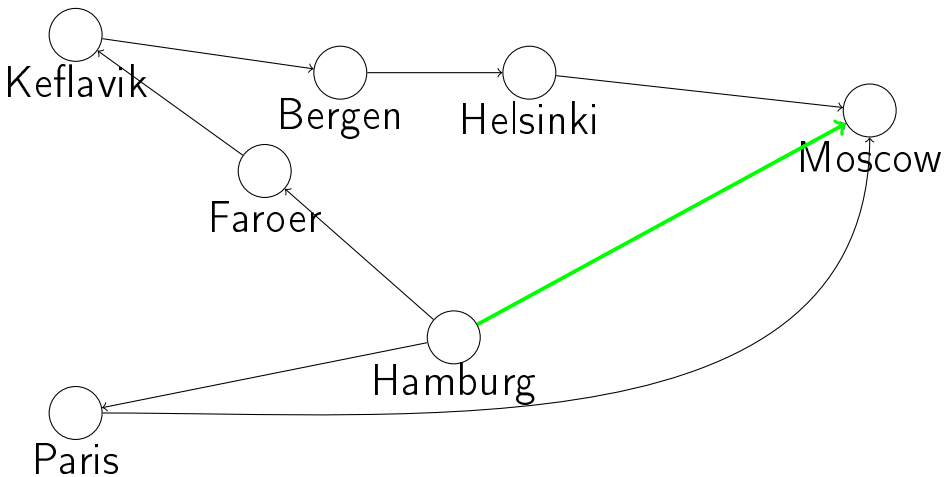
What is the minimum number of flight segments to get from Hamburg to Moscow?





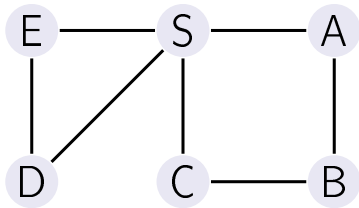






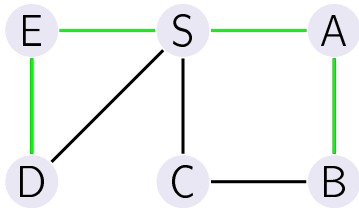
# Paths and lengths

Length of the path  $L(P)$  is the number of edges in the path.



# Paths and lengths

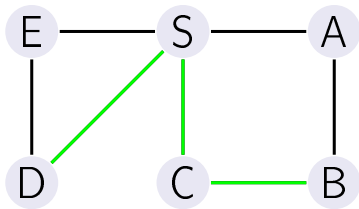
Length of the path  $L(P)$  is the number of edges in the path.



$$L(D - E - S - A - B) = 4$$

# Paths and lengths

Length of the path  $L(P)$  is the number of edges in the path.



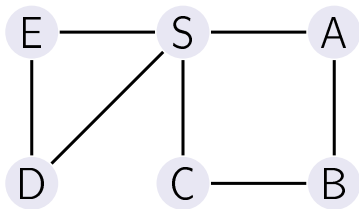
$$L(D - E - S - A - B) = 4$$

$$L(D - S - C - B) = 3$$



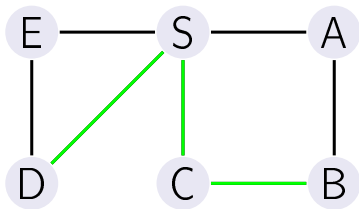
# Distances

The **distance** between two vertices is the length of the shortest path between them.



# Distances

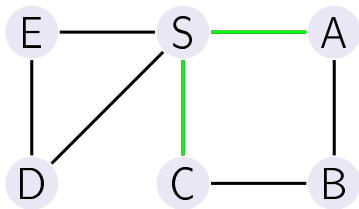
The **distance** between two vertices is the length of the shortest path between them.



$$d(D, B) = 3$$

# Distances

The **distance** between two vertices is the length of the shortest path between them.

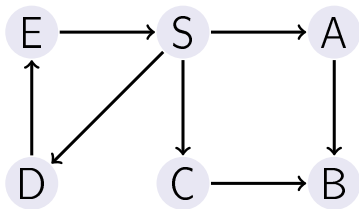


$$d(D, B) = 3$$

$$d(C, A) = 2$$

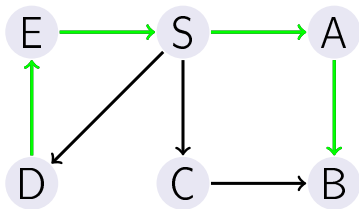
# Distances

The **distance** between two vertices is the length of the shortest path between them.



# Distances

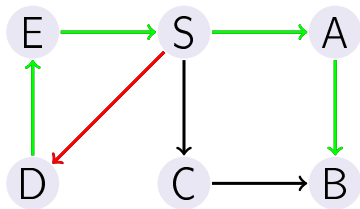
The **distance** between two vertices is the length of the shortest path between them.



$$d(D, B) = 4$$

# Distances

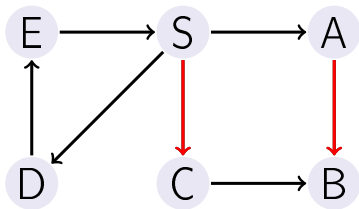
The **distance** between two vertices is the length of the shortest path between them.



$$d(D, B) = 4$$

# Distances

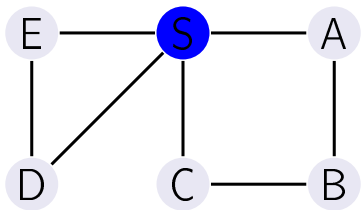
The **distance** between two vertices is the length of the shortest path between them.



$$d(D, B) = 4$$

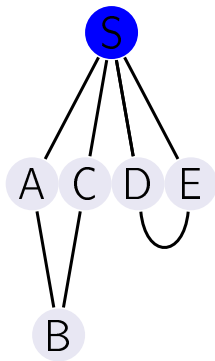
$$d(C, A) = \infty$$

# Distances

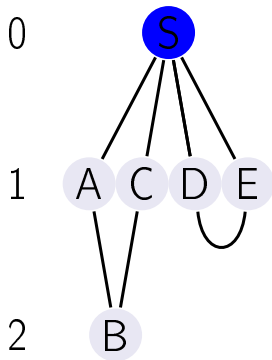




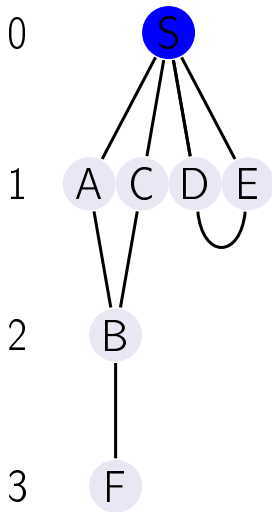
# Distance layers



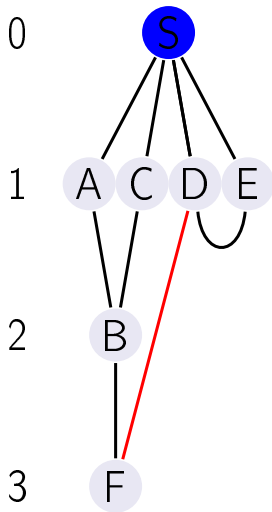
# Distance layers



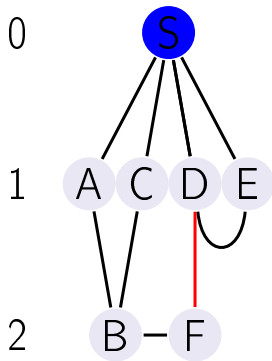
# Distance layers



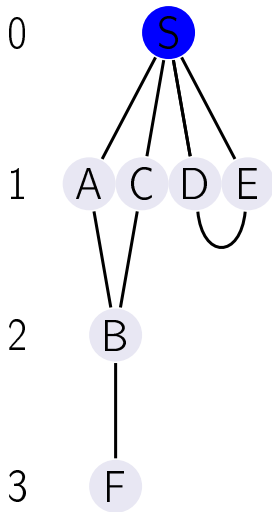
# Distance layers



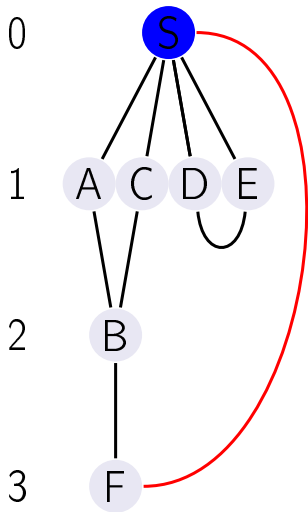
# Distance layers



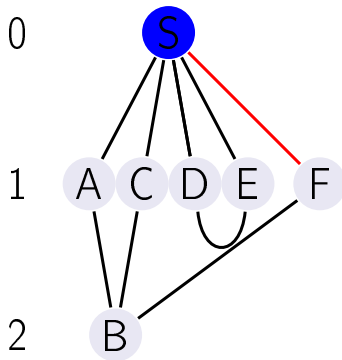
# Distance layers



# Distance layers

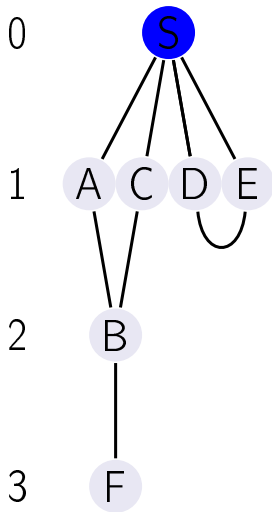


# Distance layers

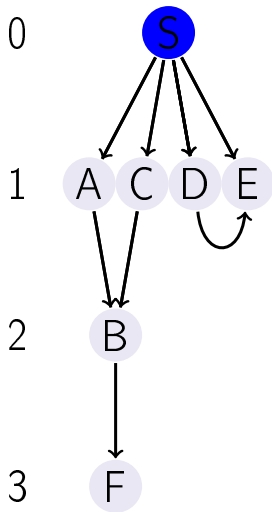




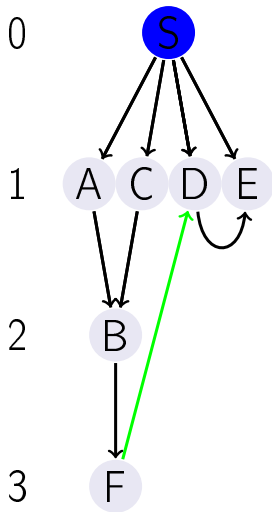
# Distance layers



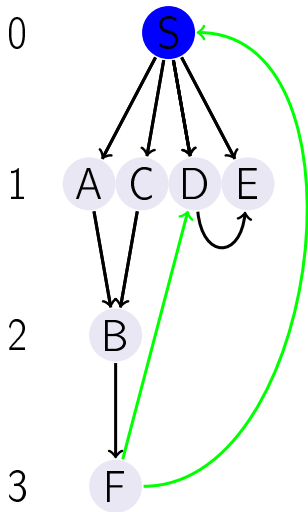
# Distance layers



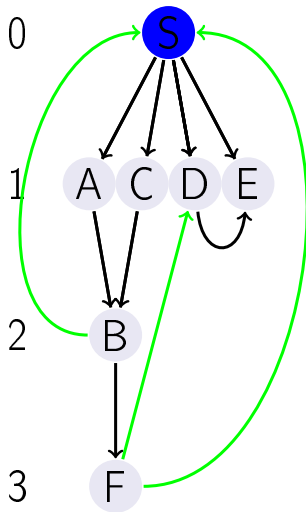
# Distance layers



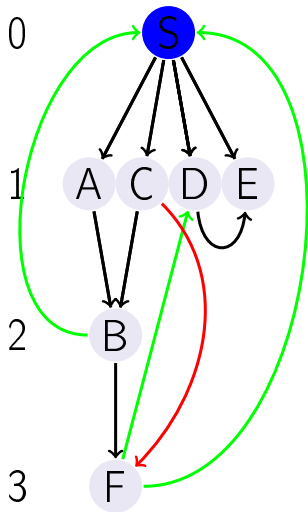
# Distance layers



# Distance layers

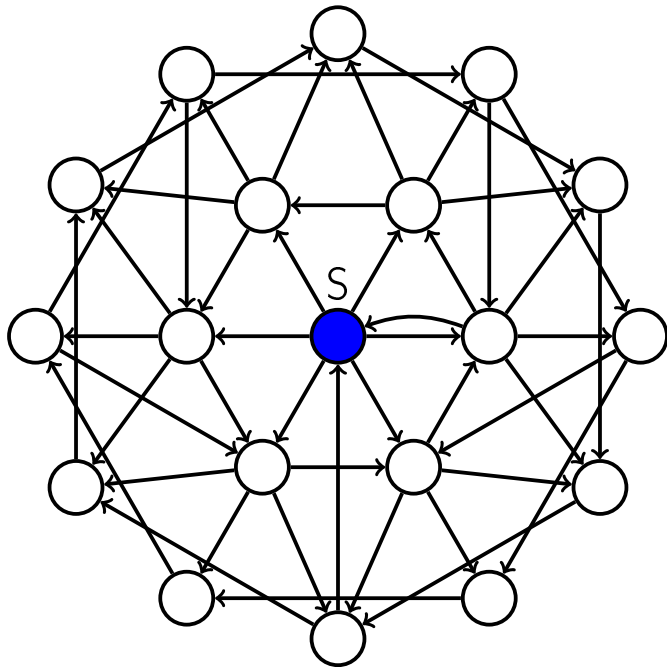


# Distance layers

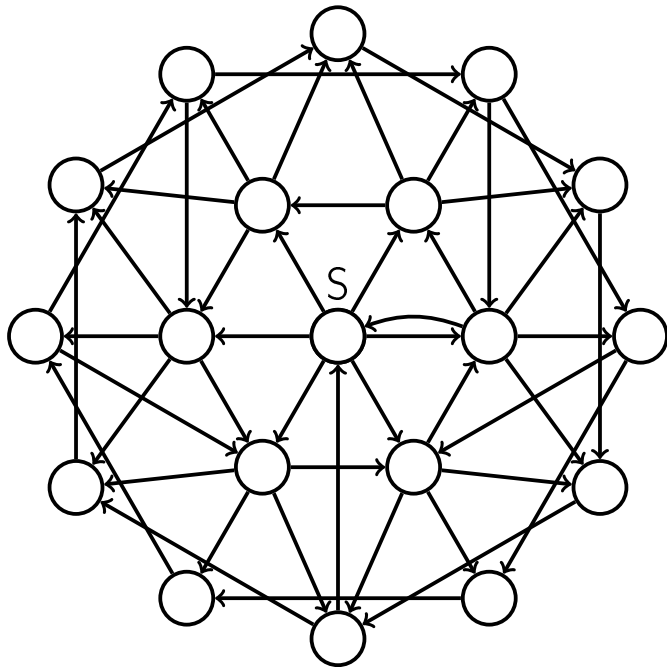


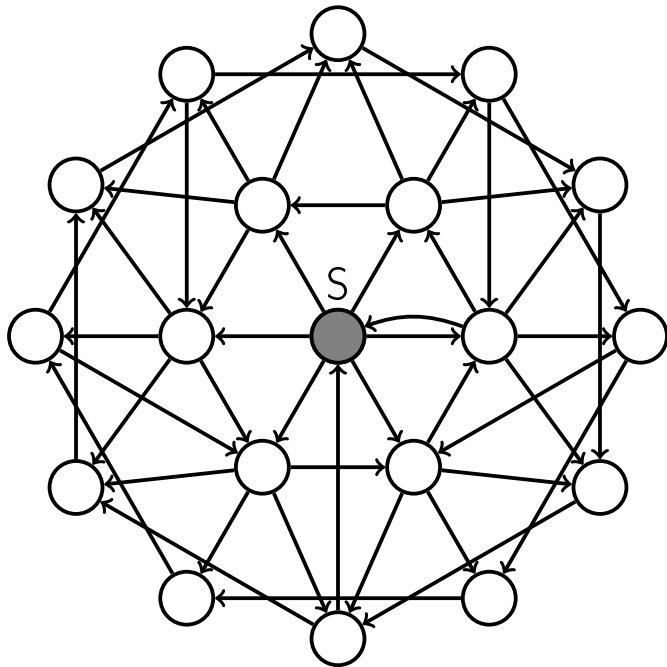
# Outline

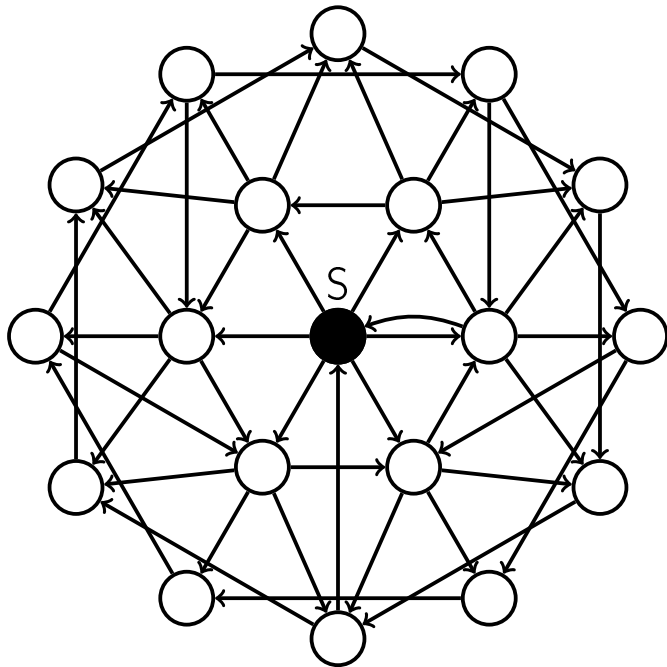
- 1 Paths and Distances
- 2 Breadth-first Search
- 3 Implementation and Analysis
- 4 Proof of Correctness
- 5 Shortest-path Tree

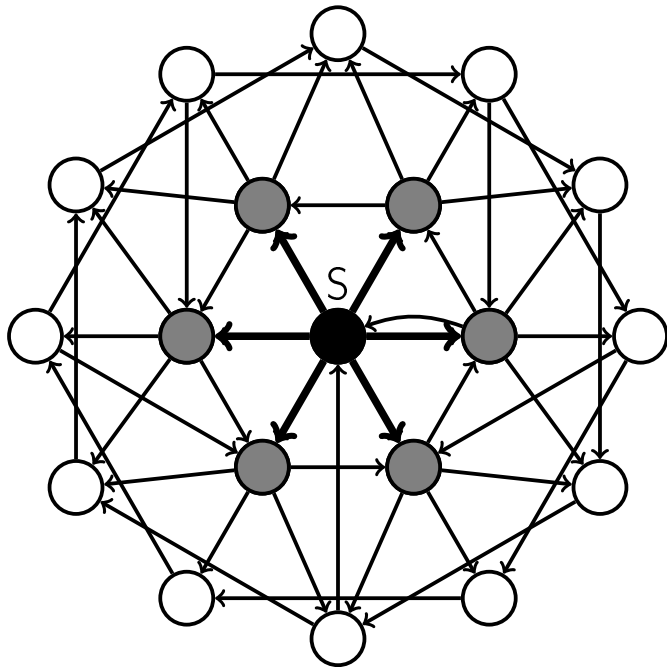


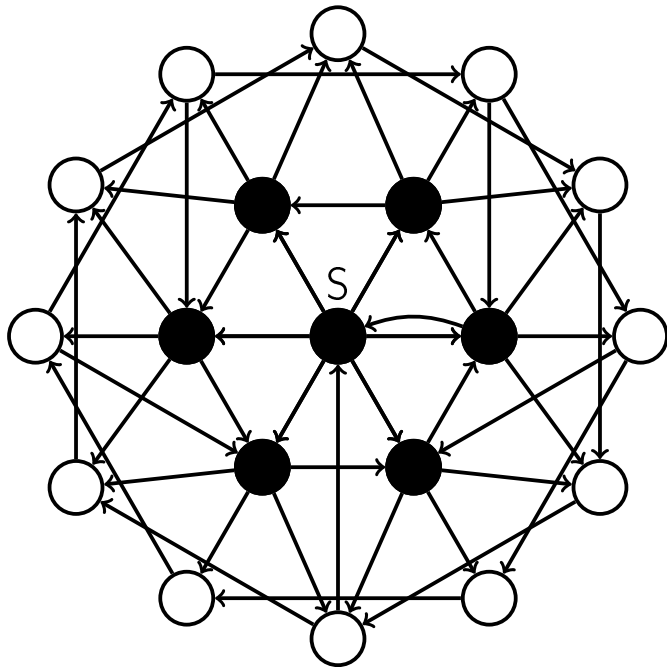


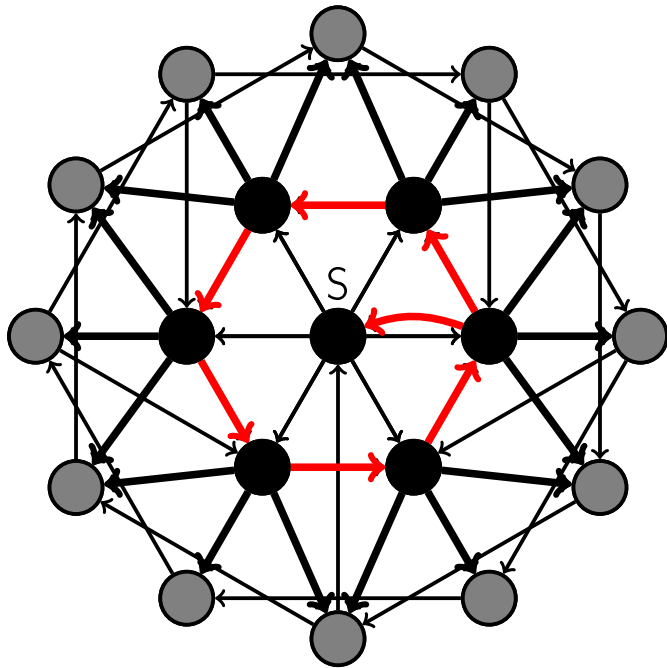


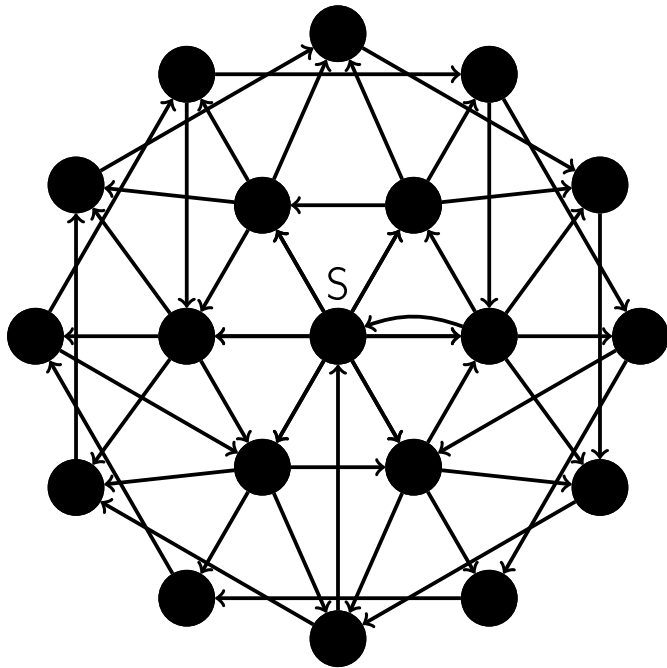


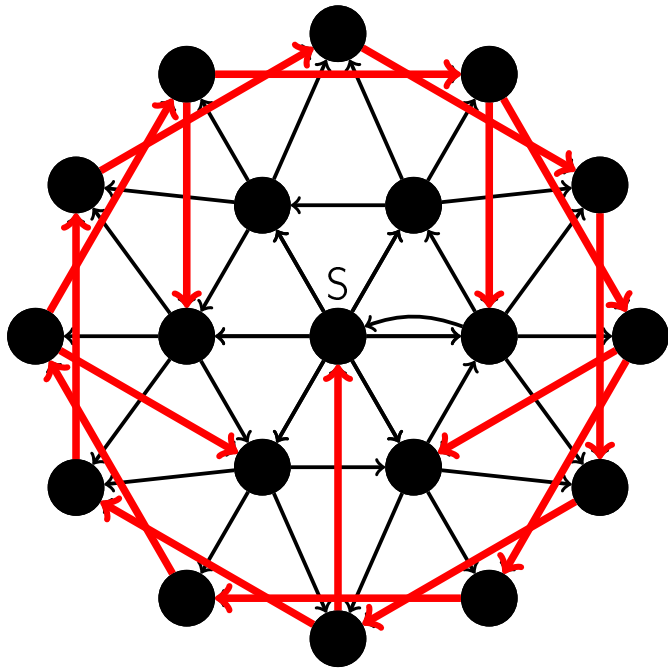




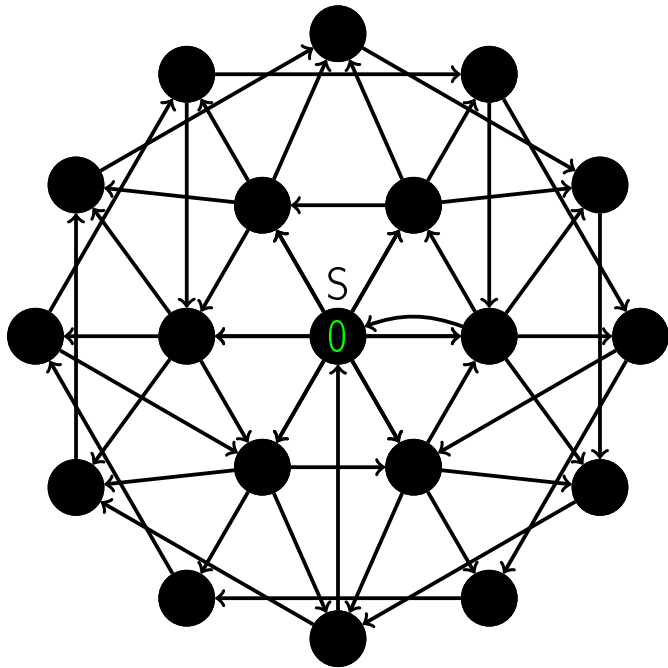


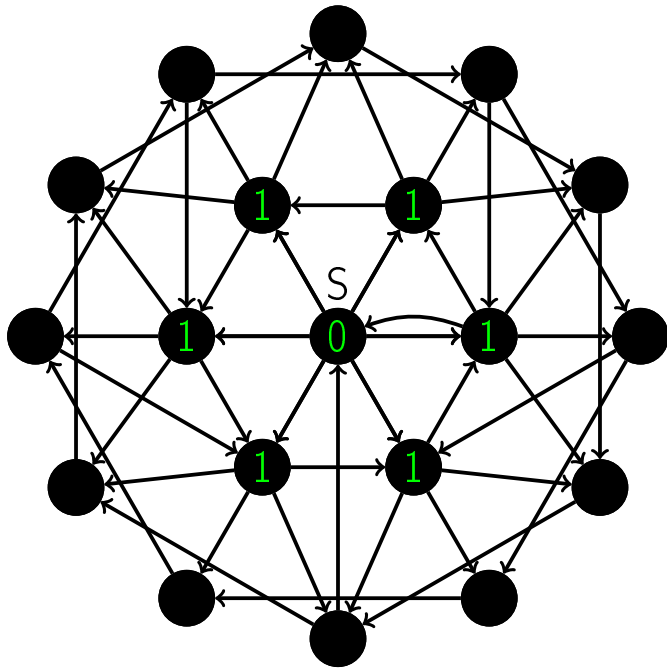


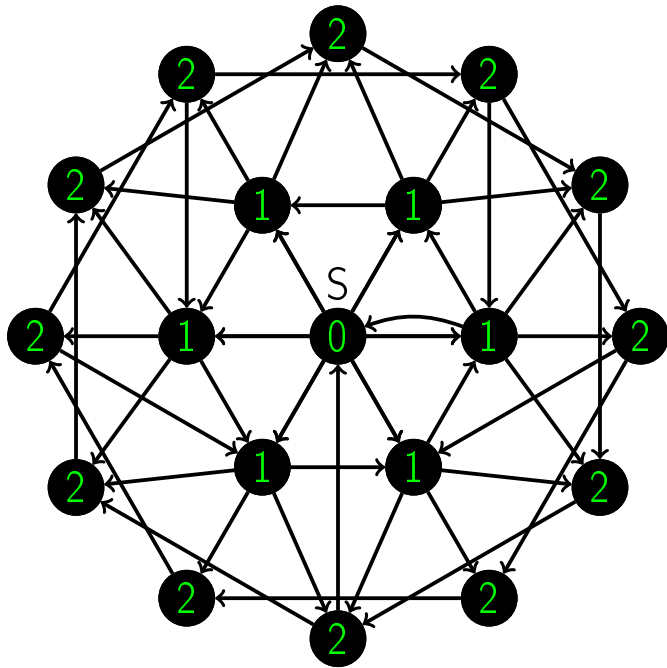


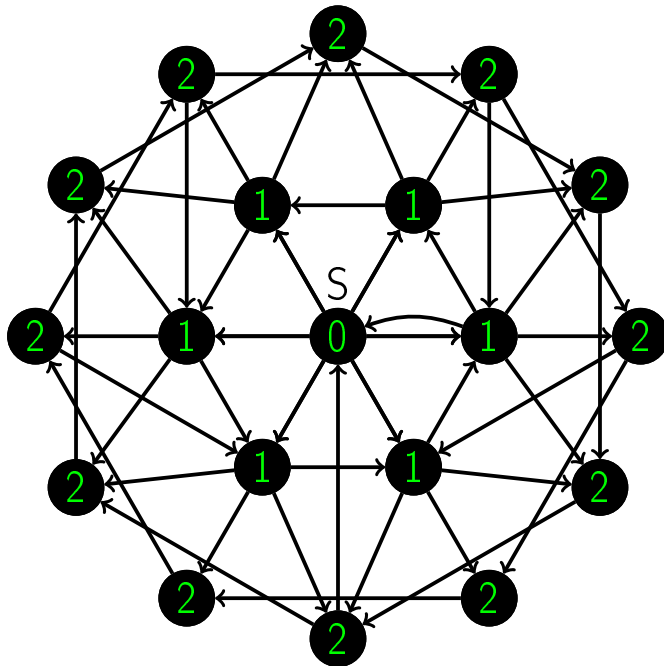


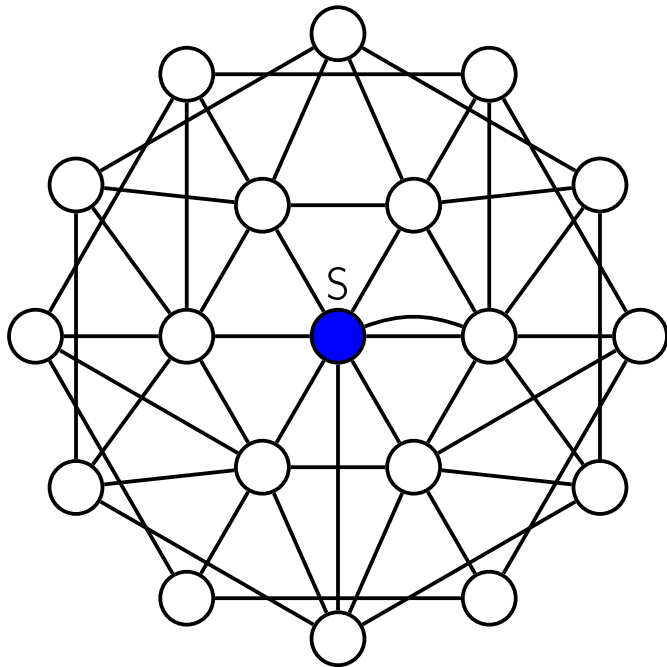


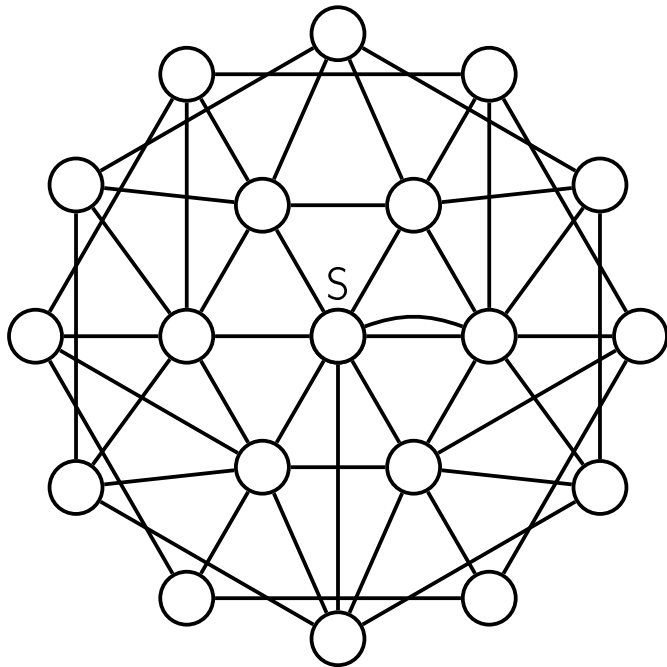


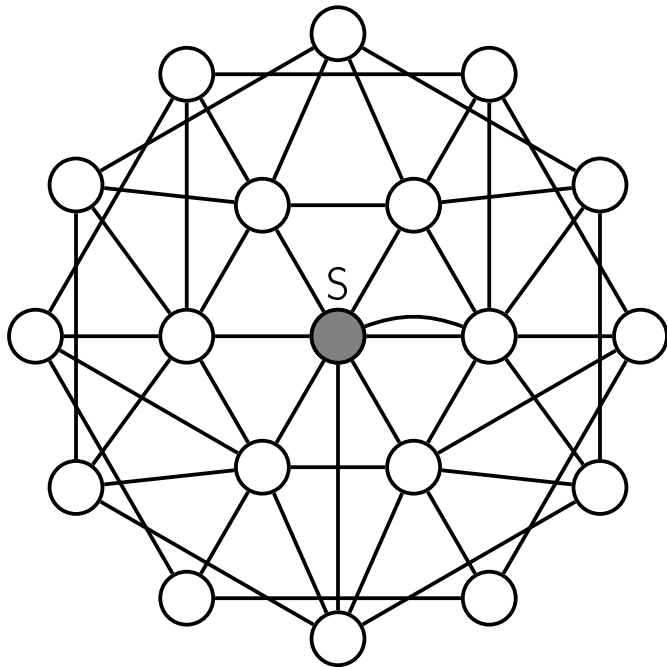


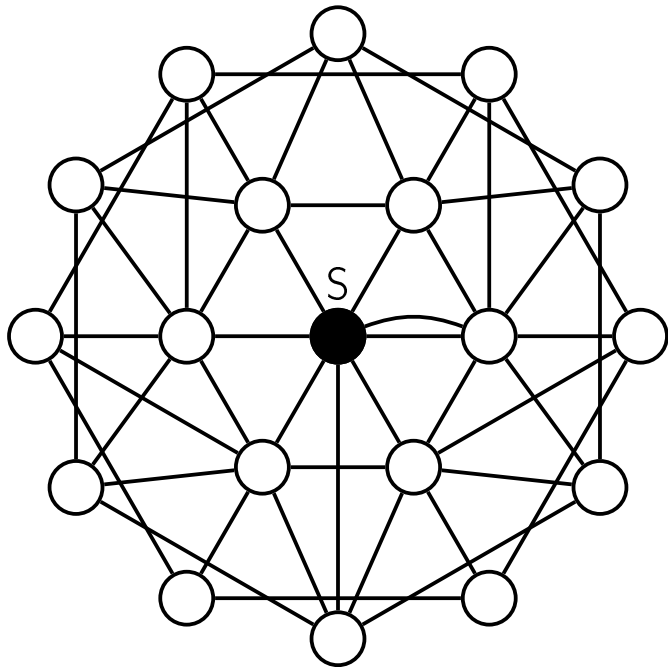




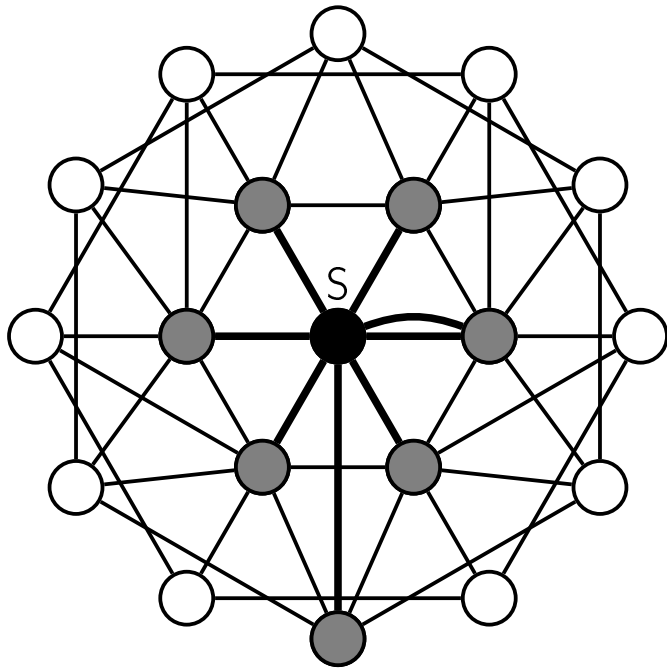


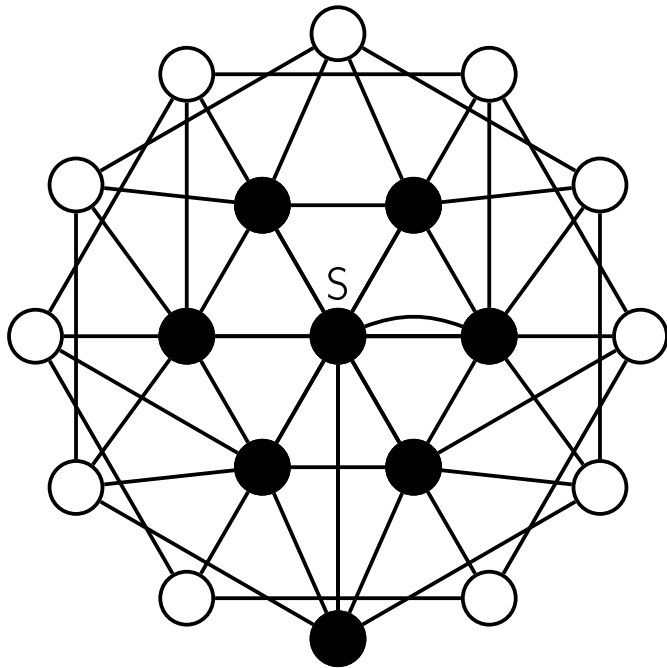


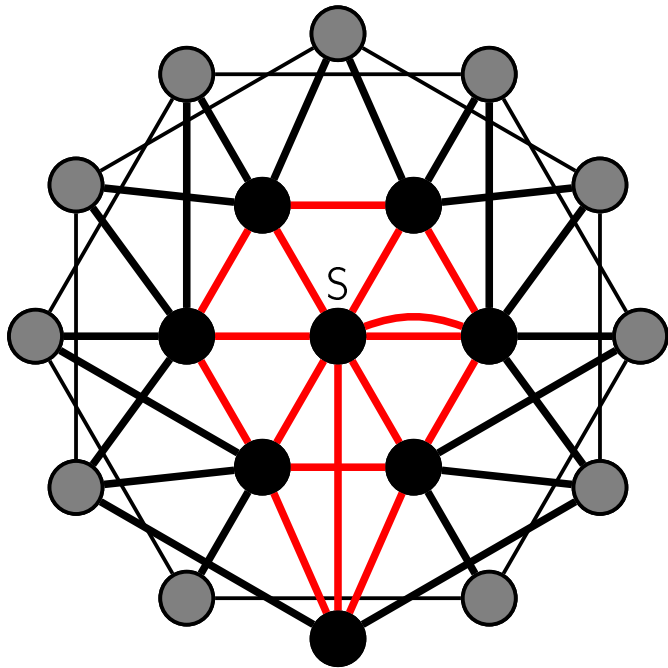


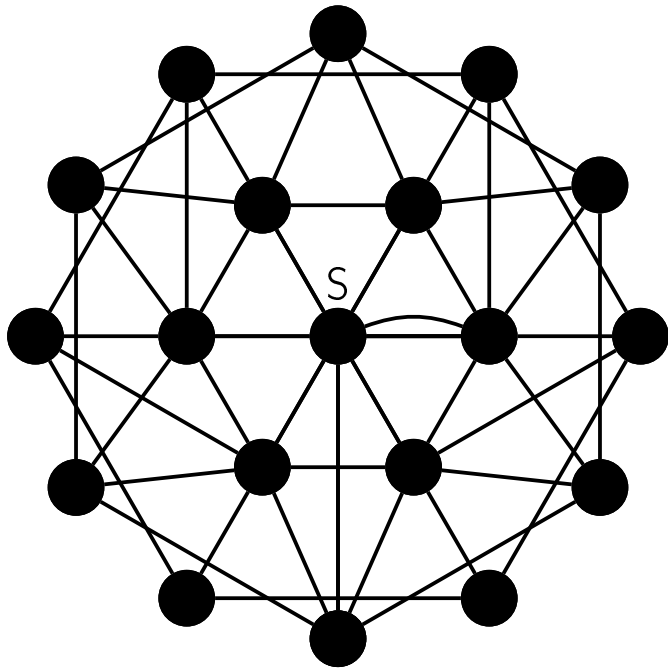


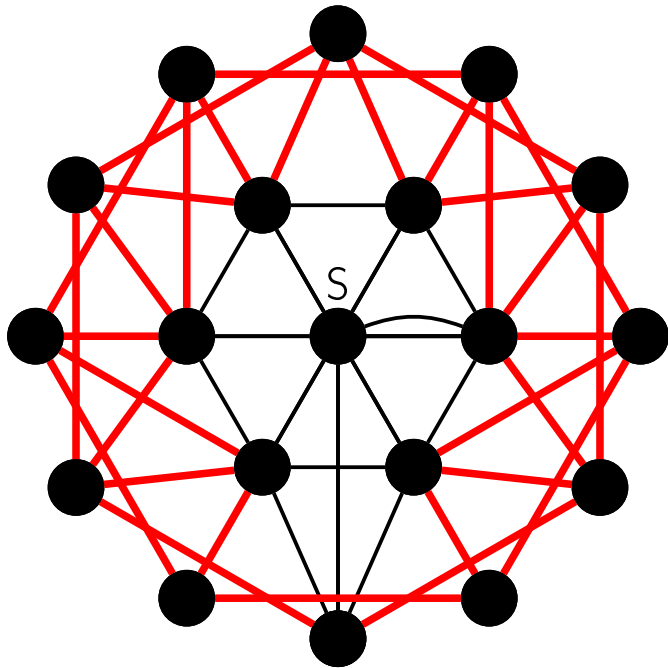


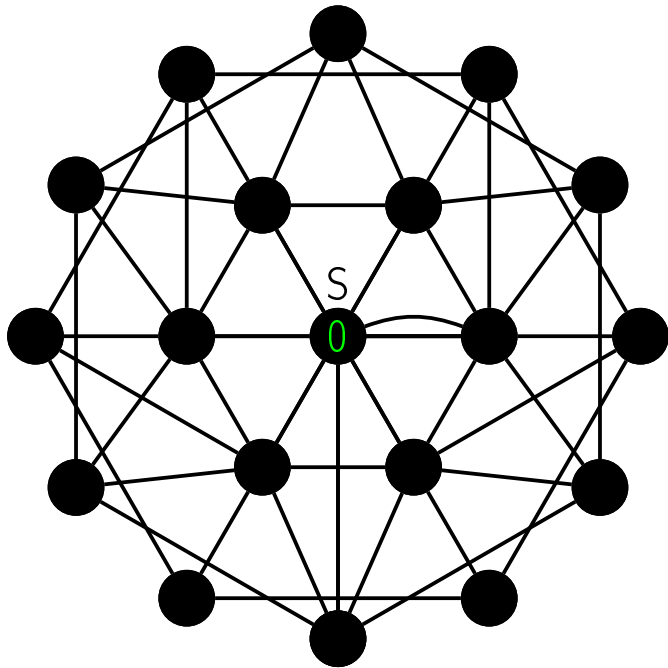


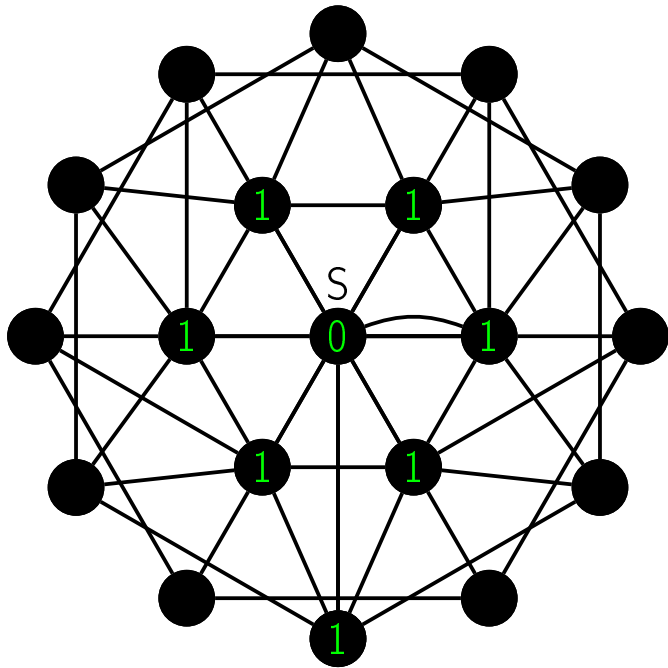


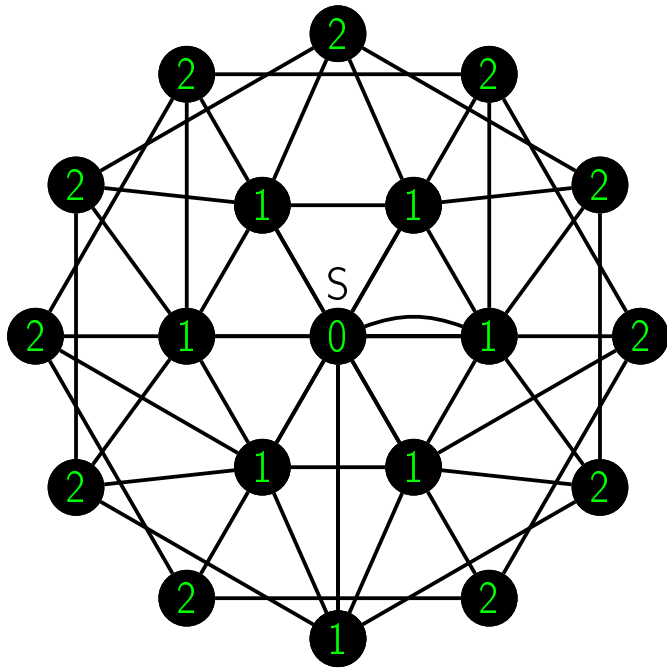




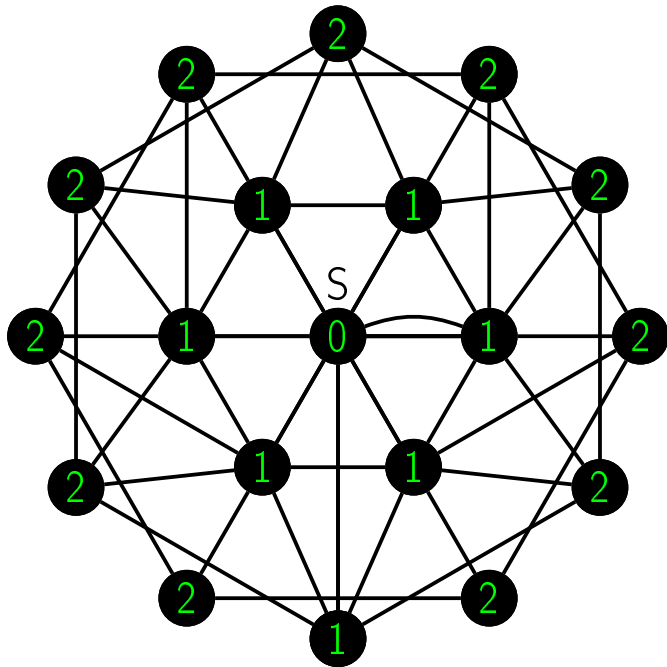


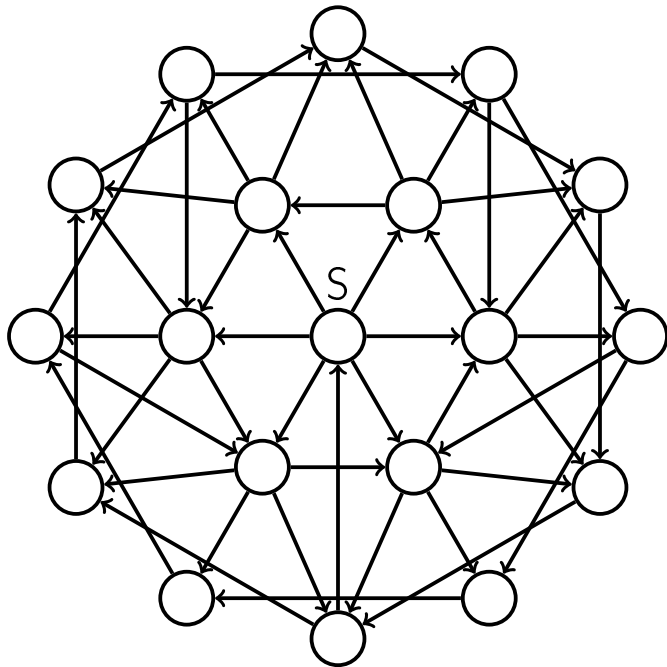


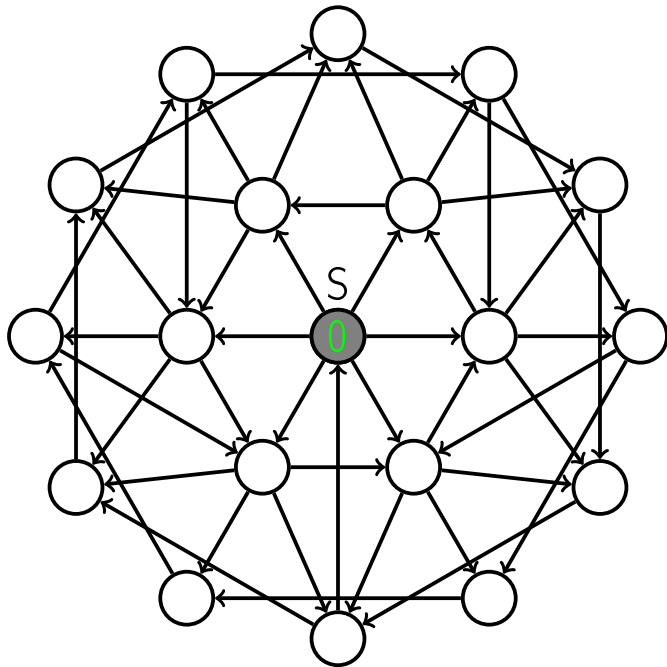


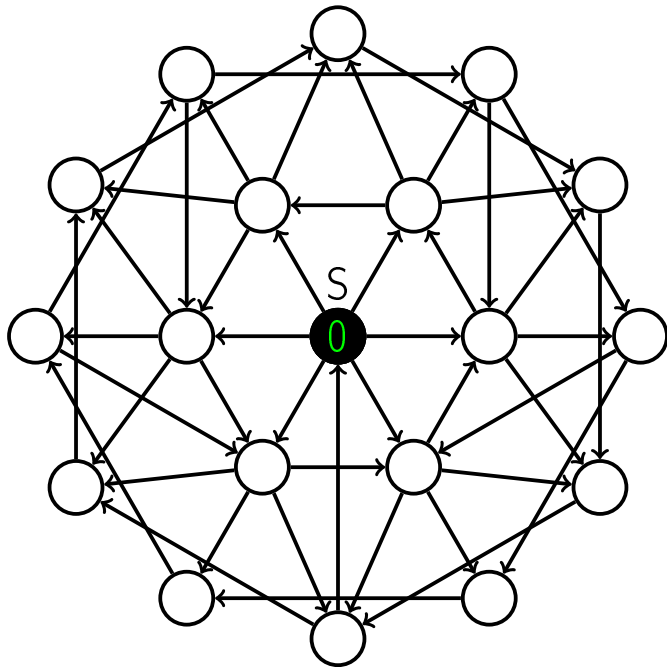


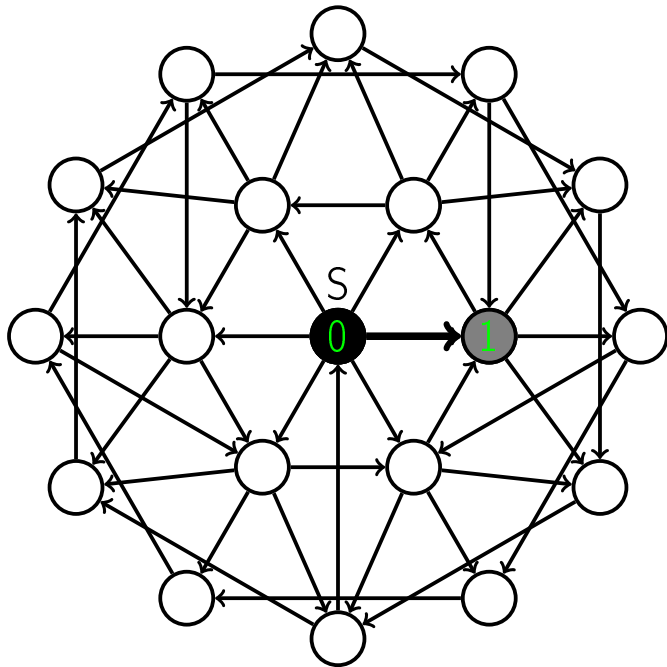


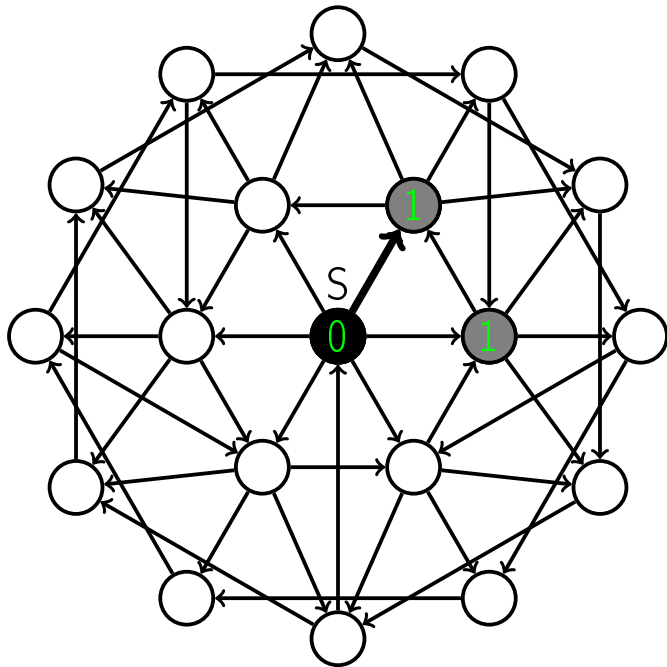


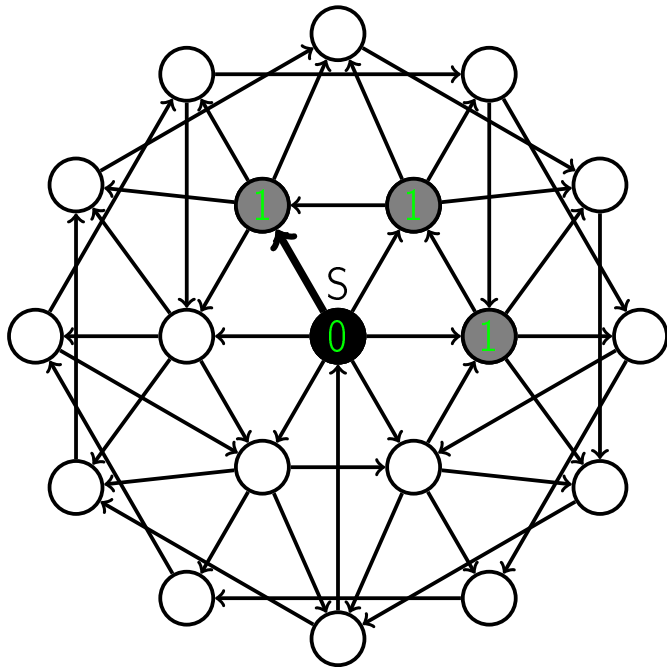


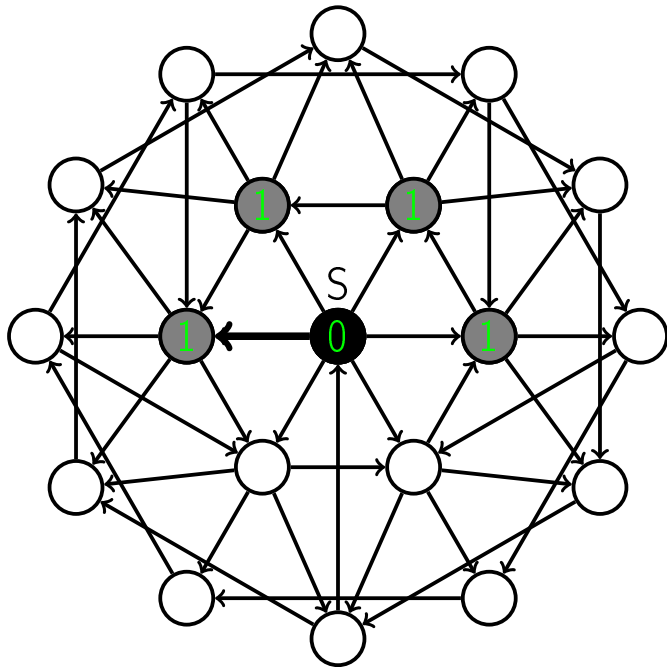




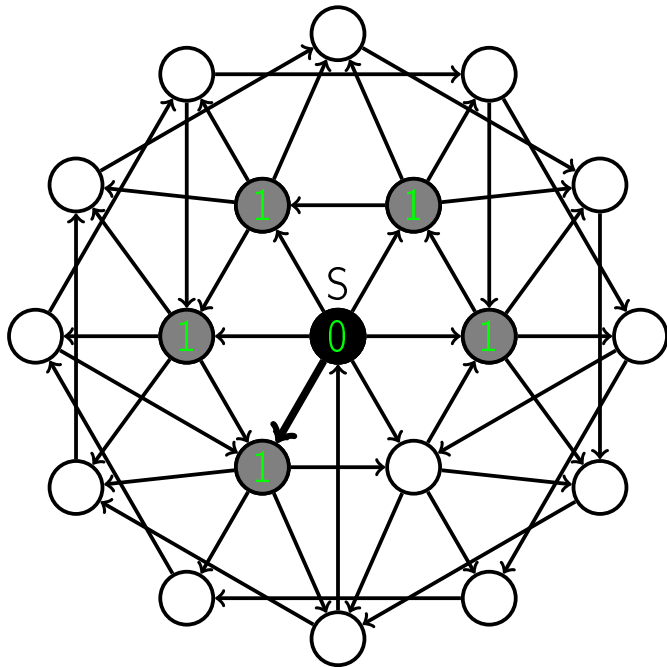


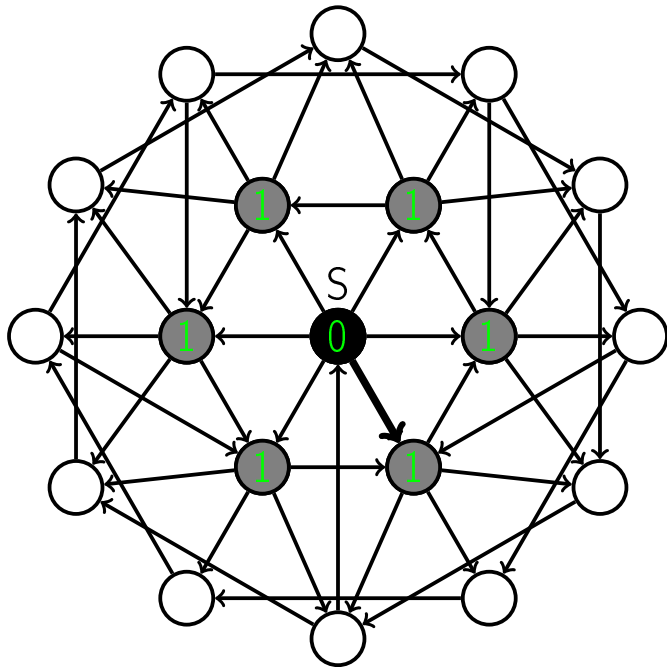


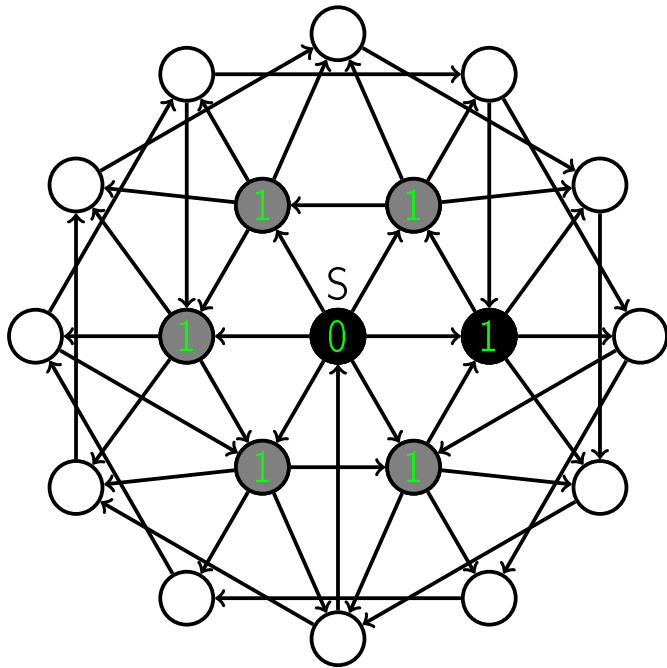


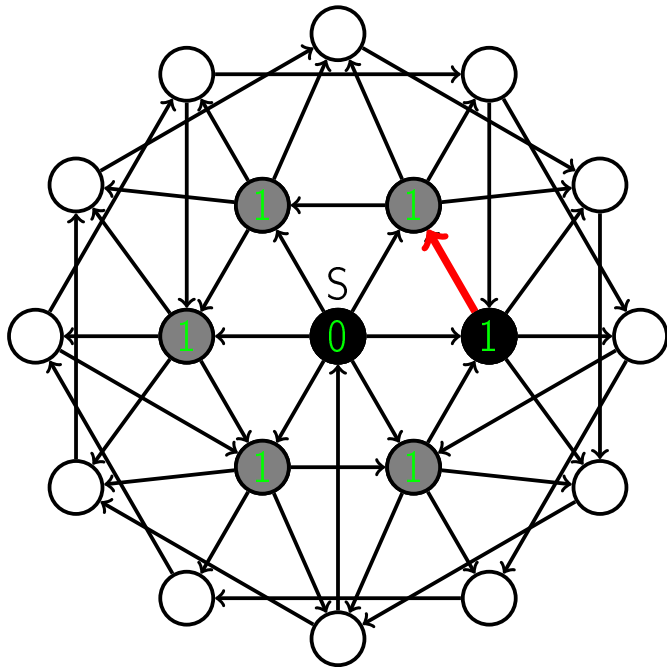


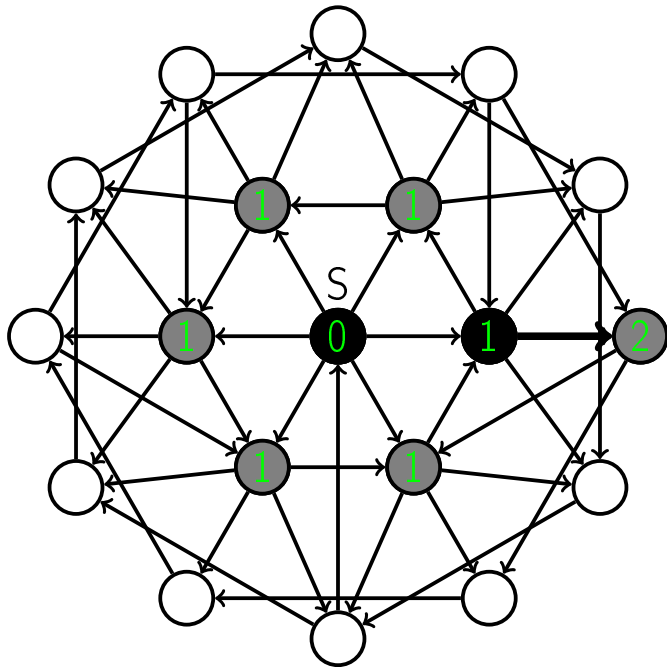


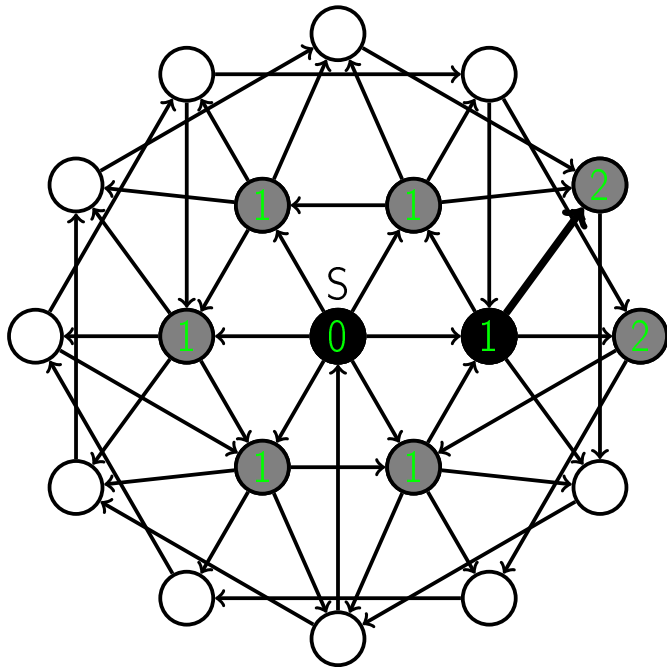


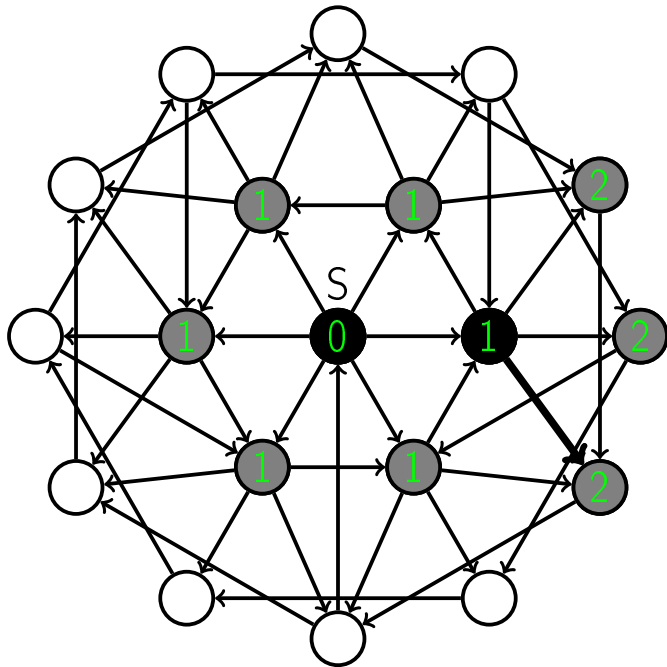


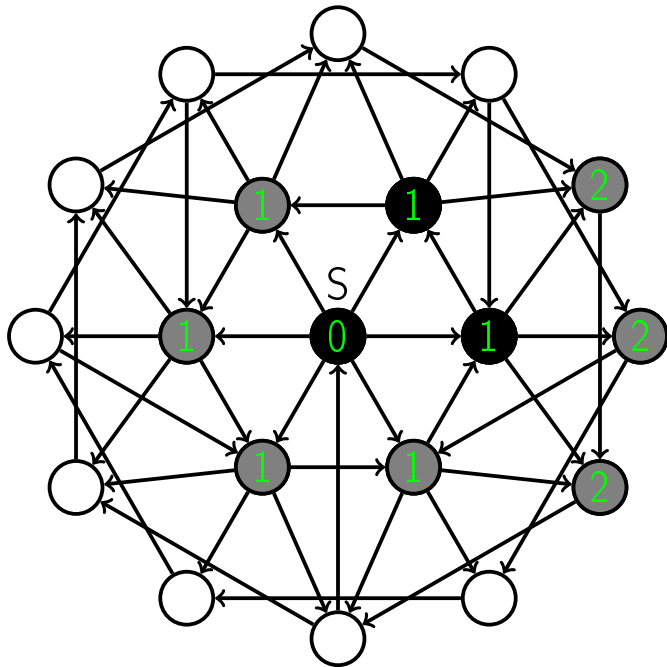




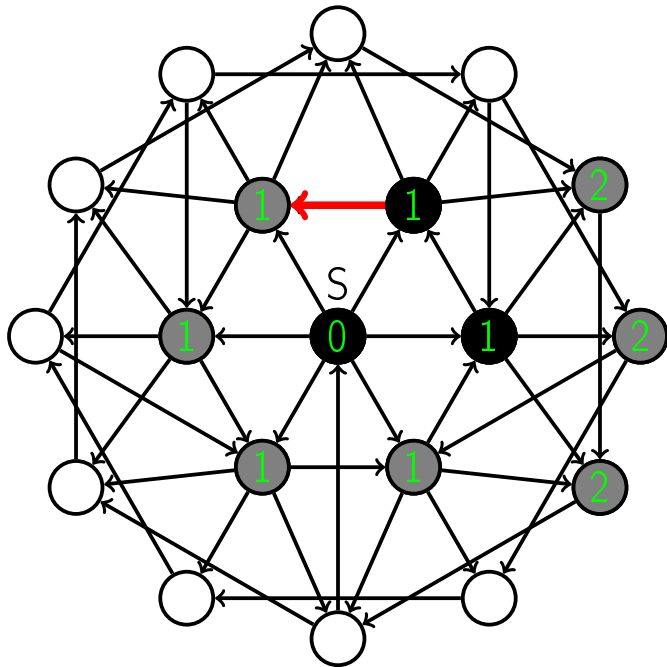


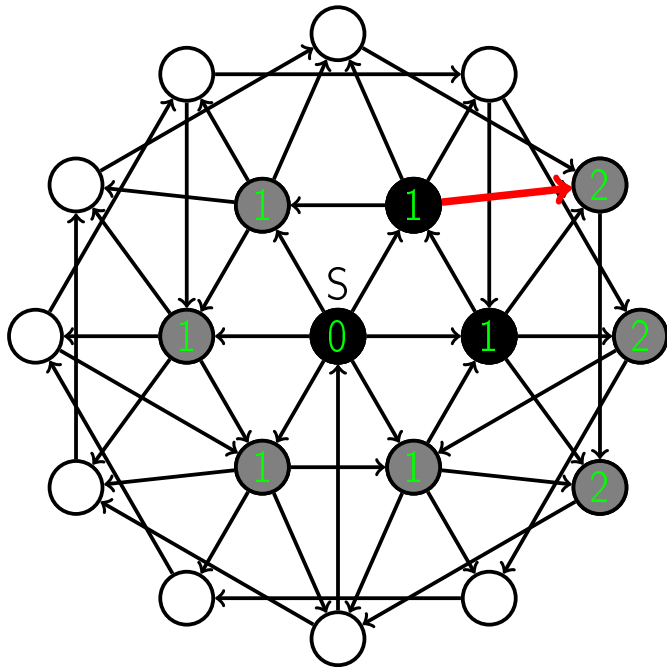


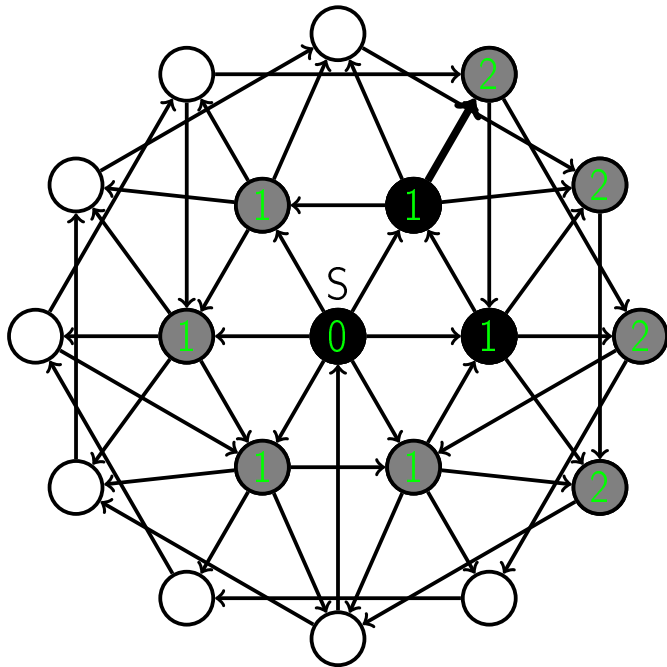


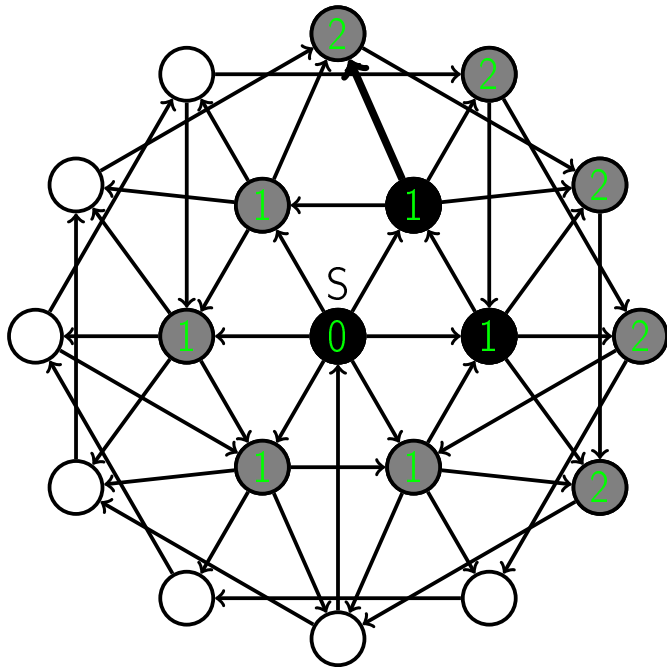


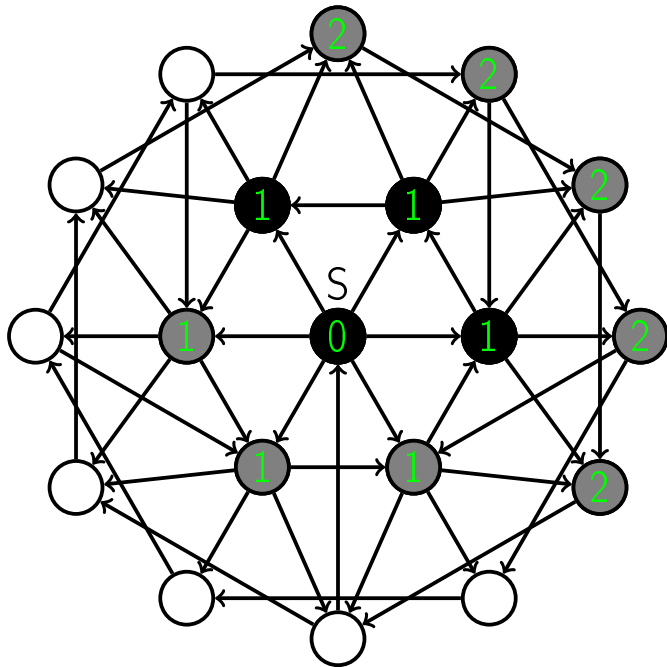


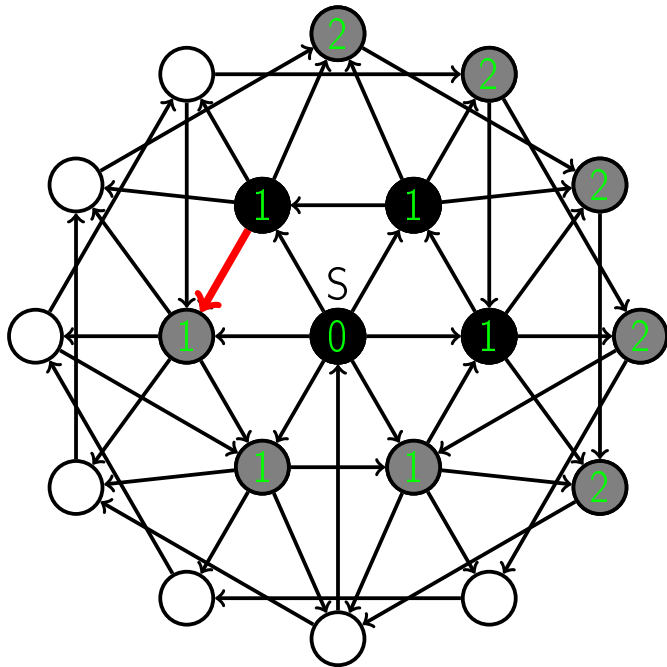


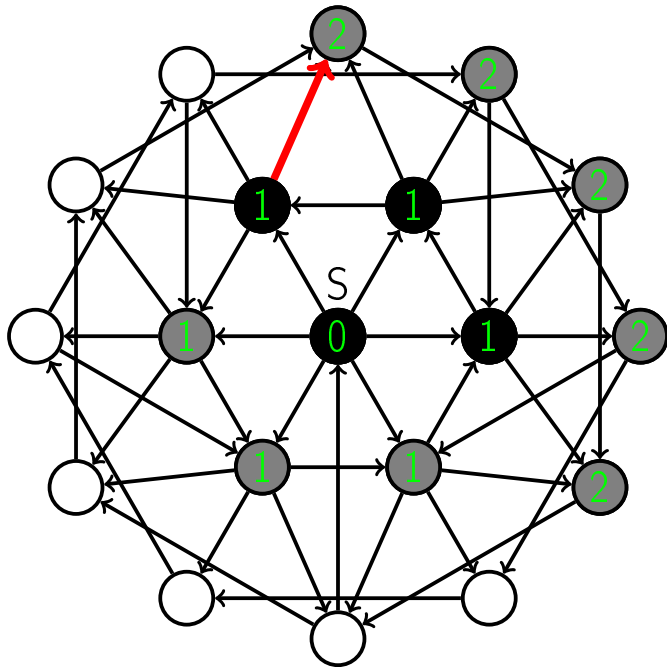


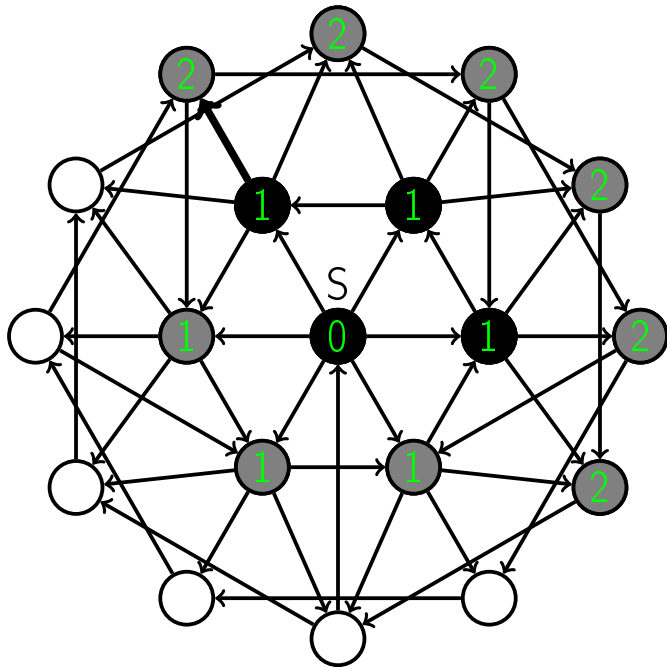




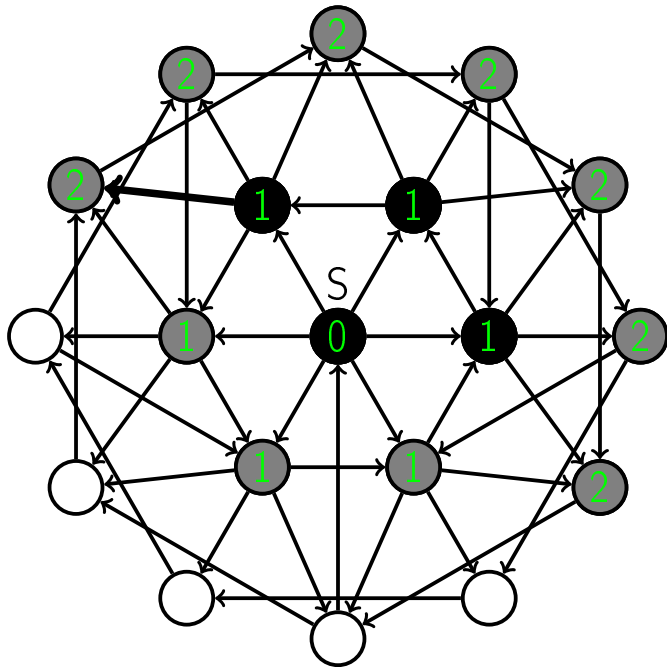


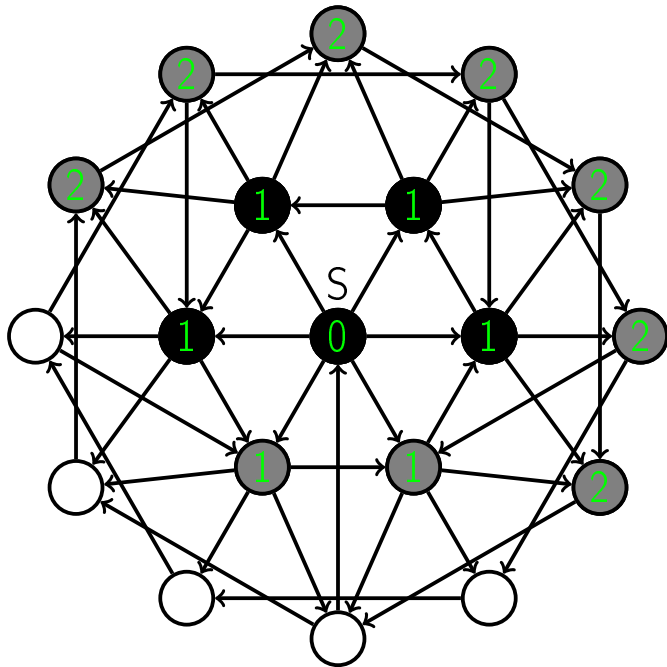


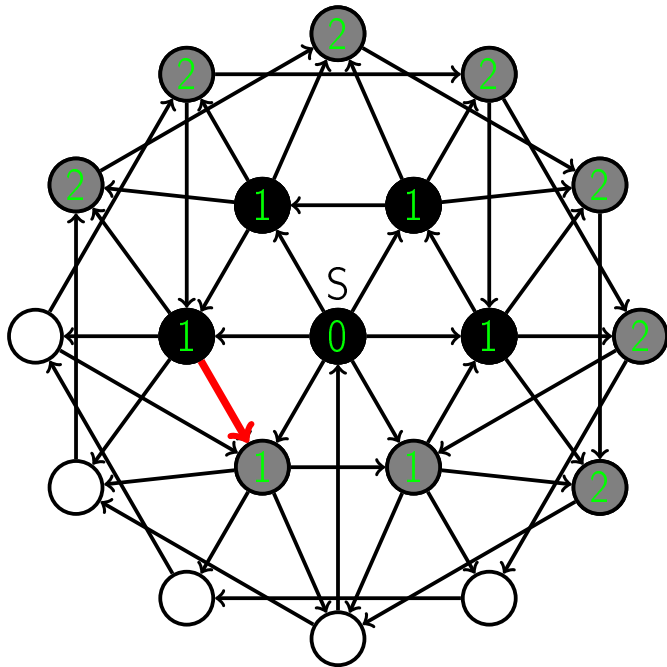


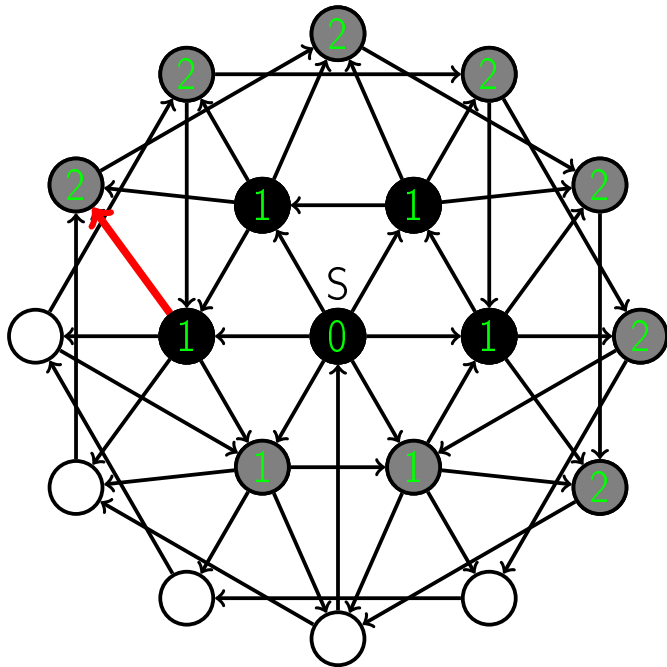


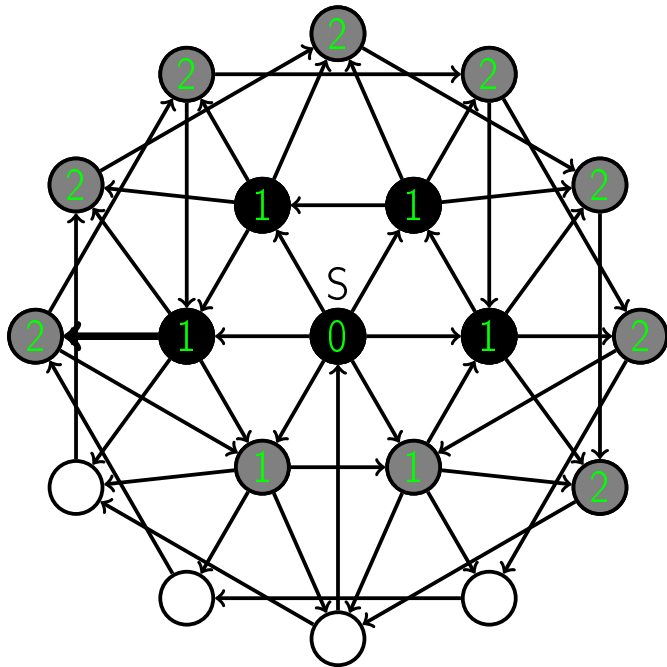


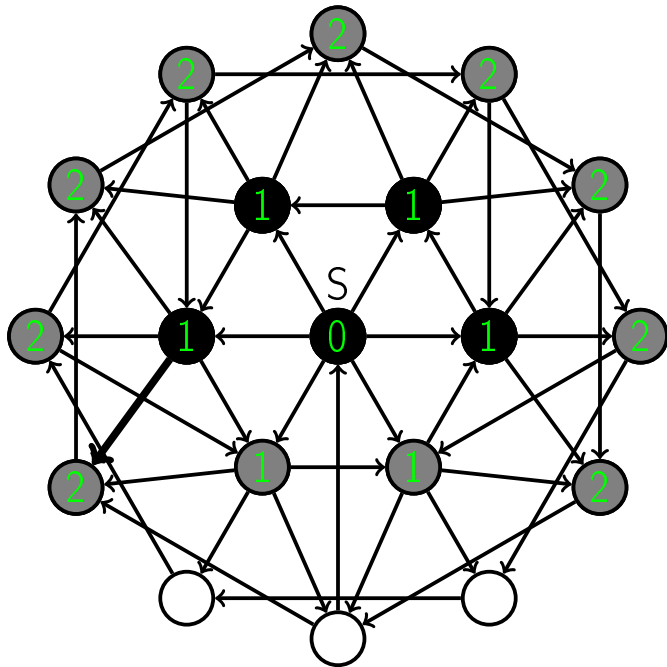


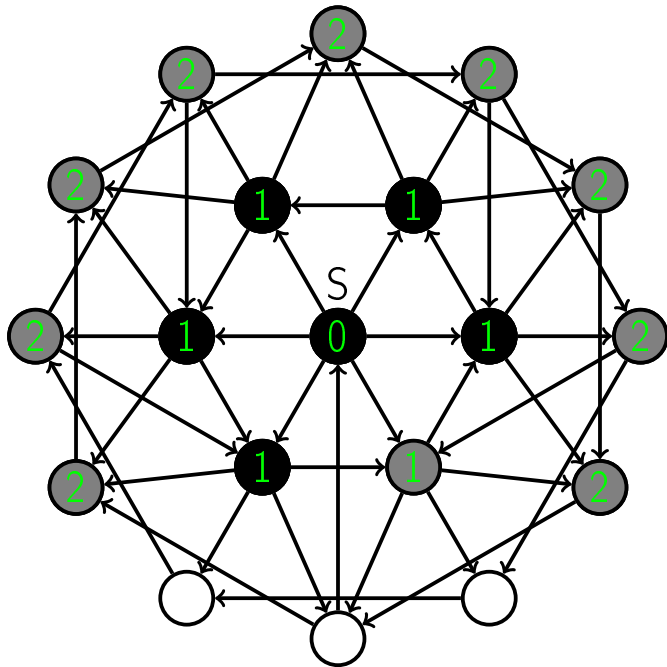


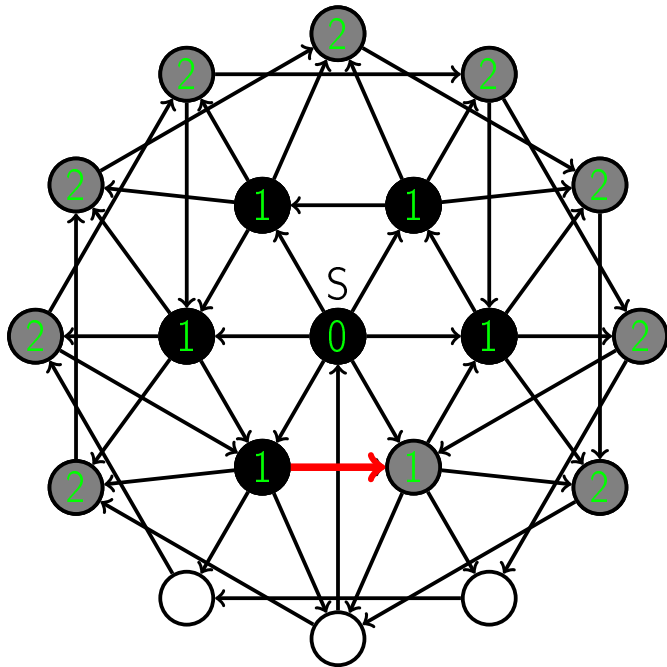




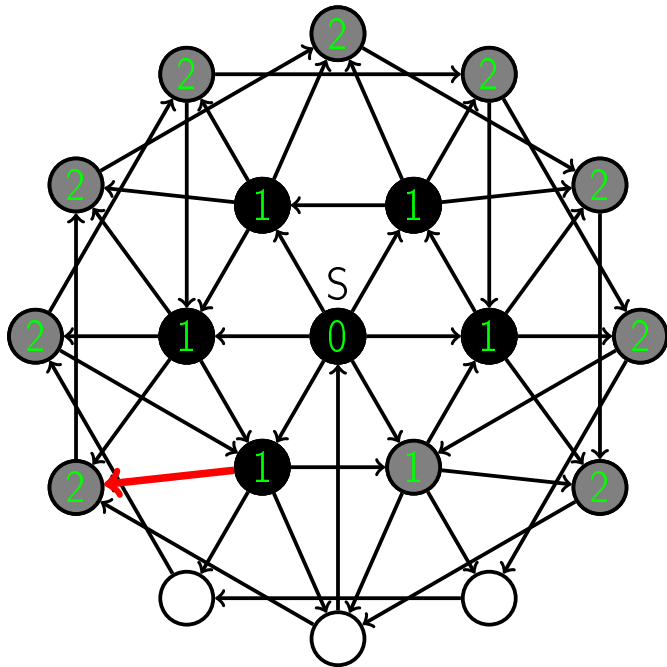


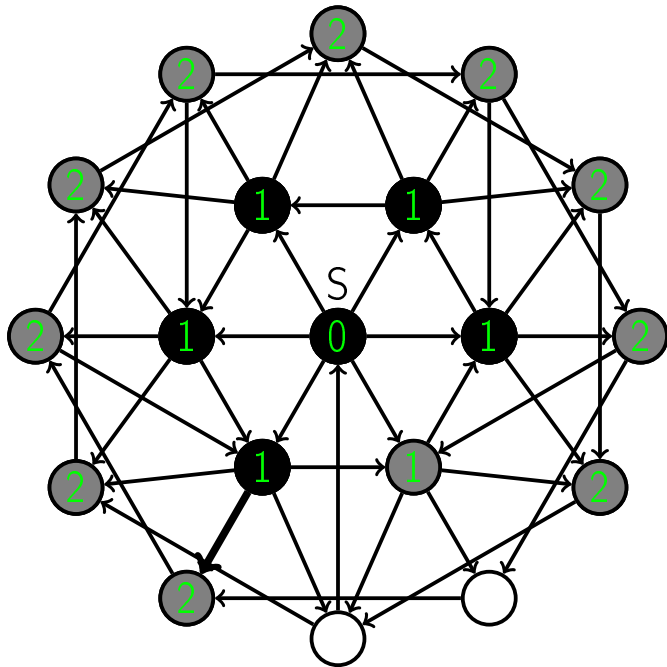


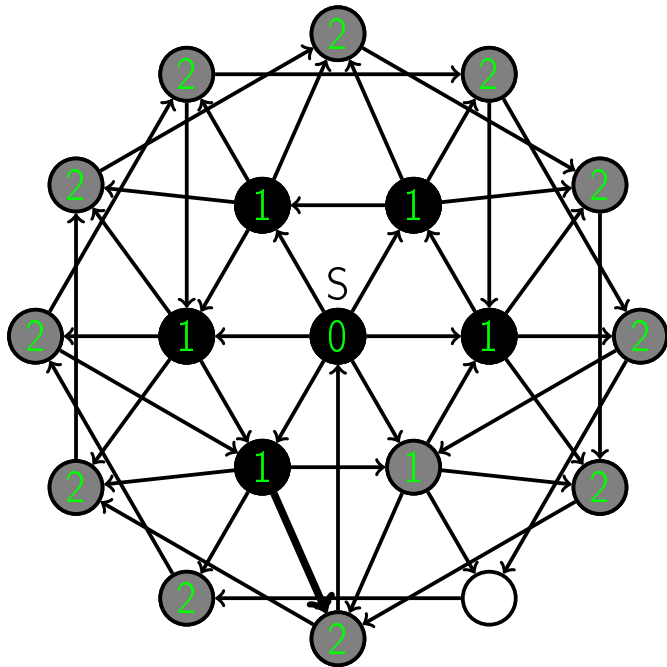


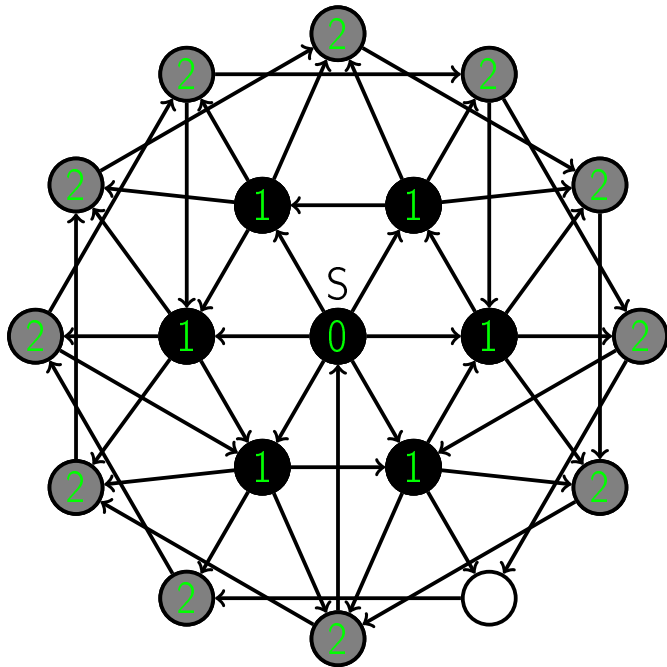


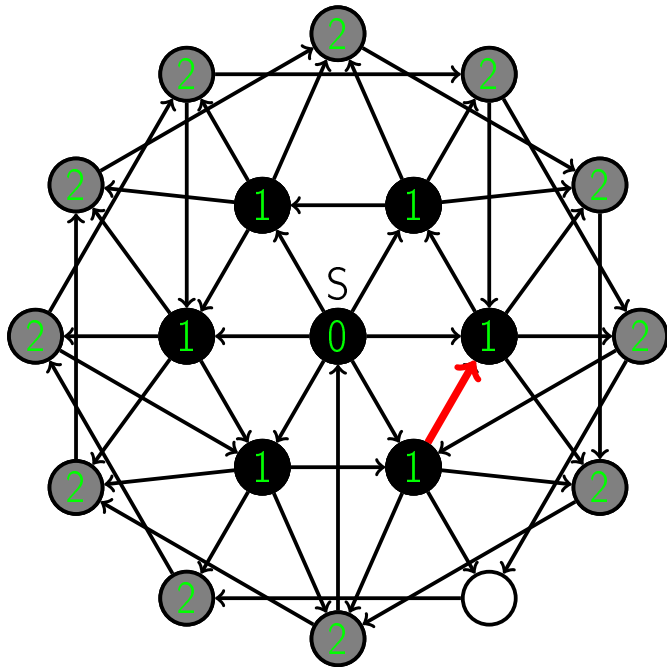


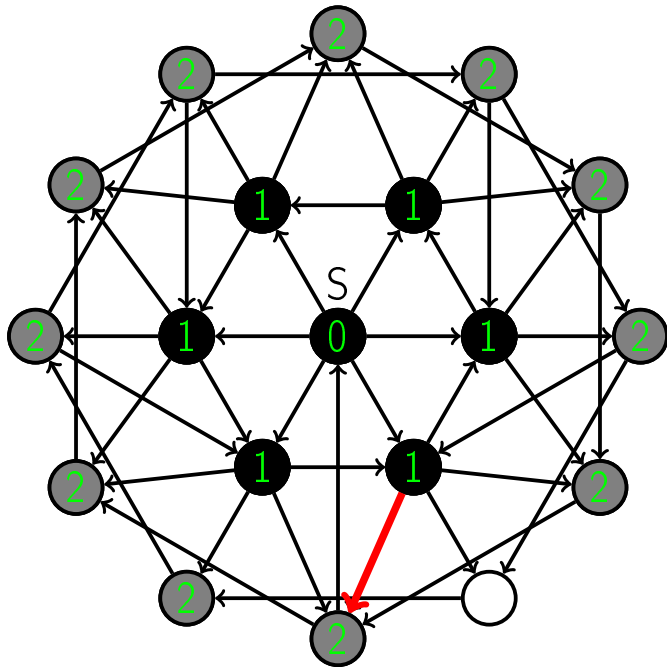


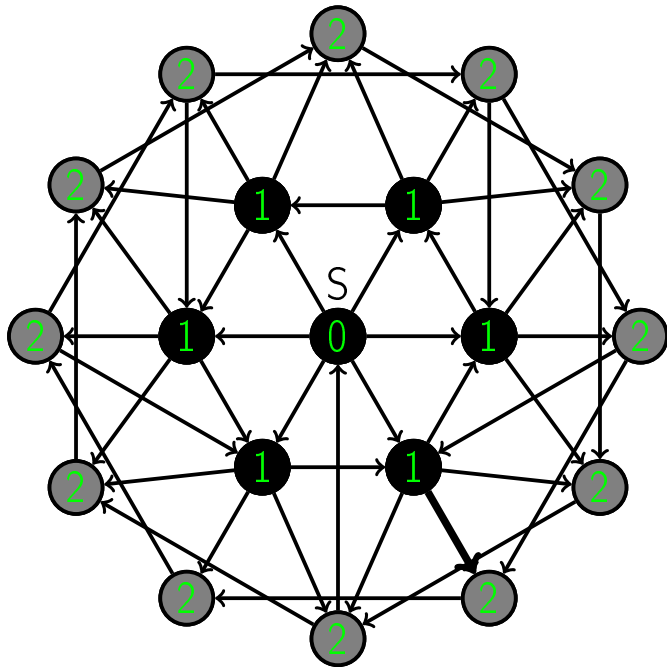


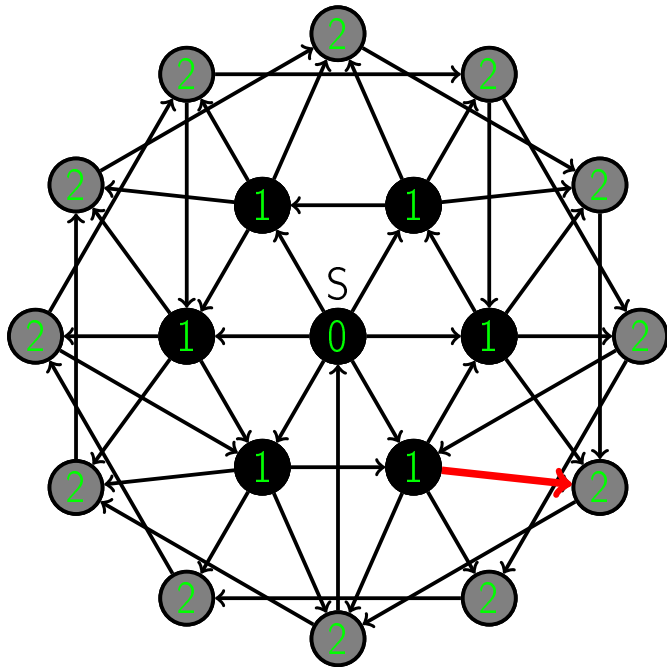




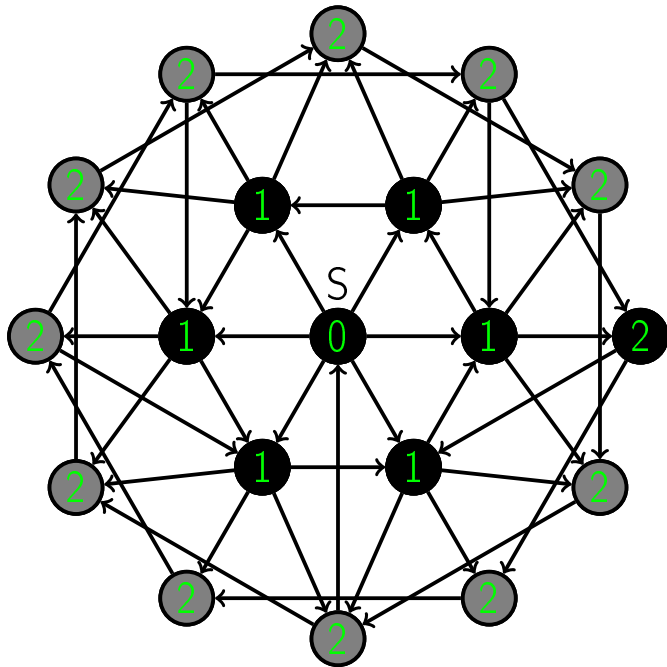


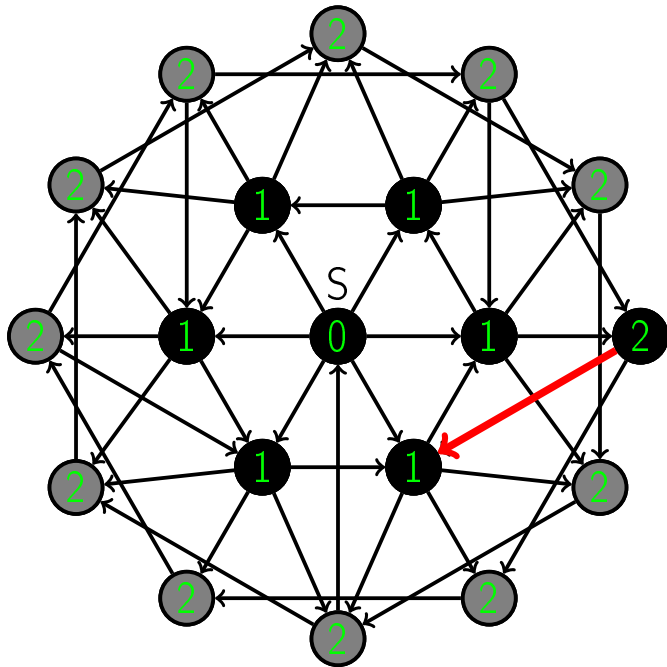


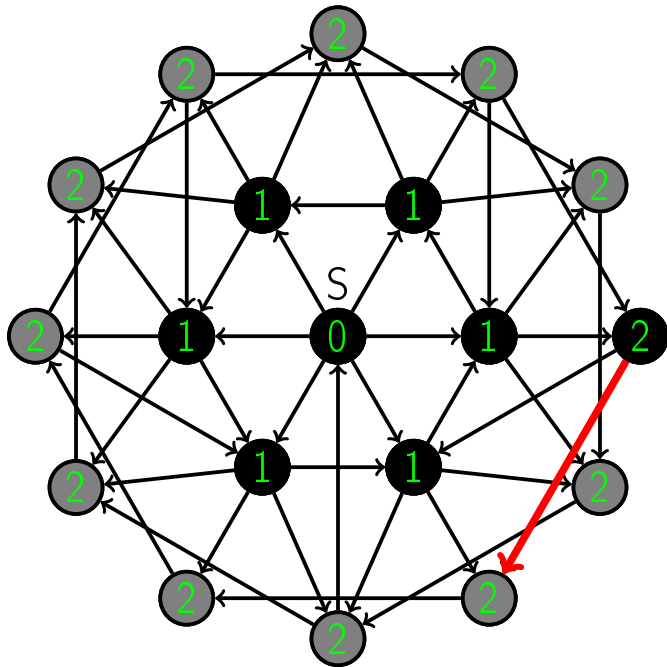


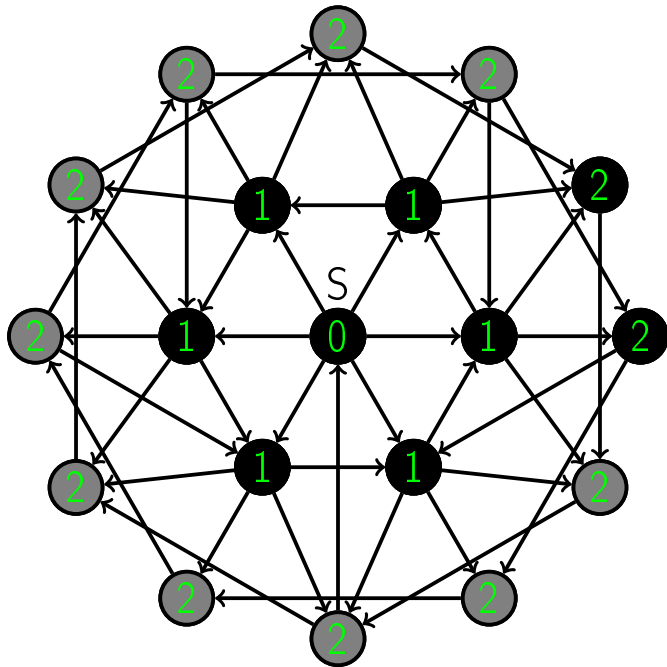


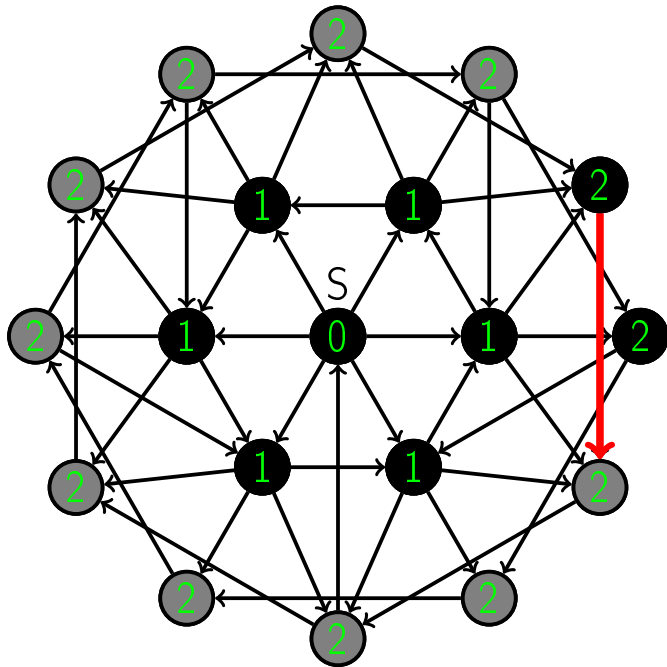


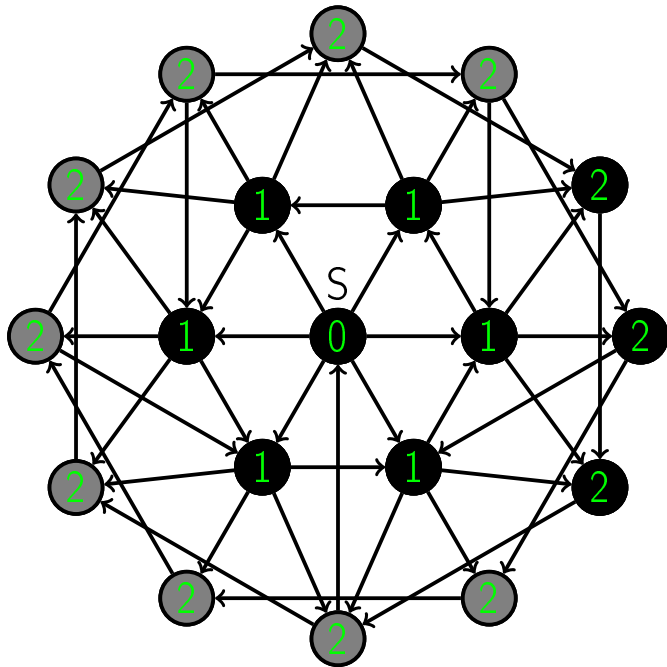


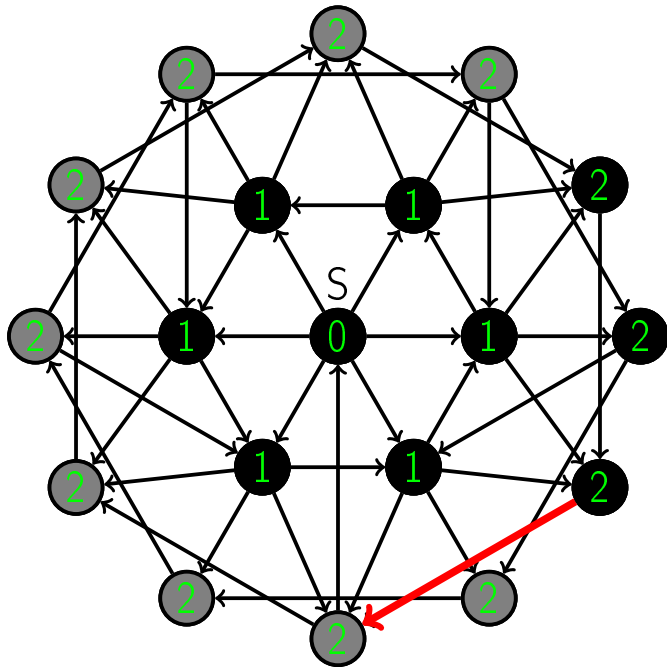


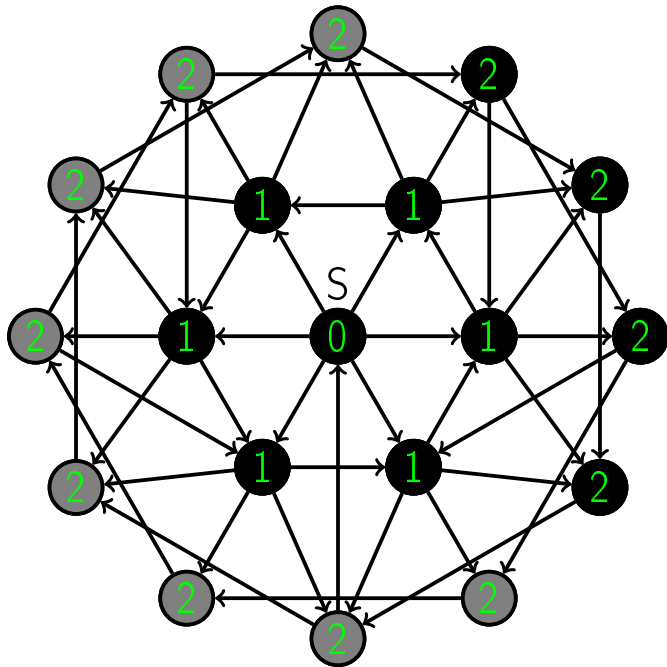




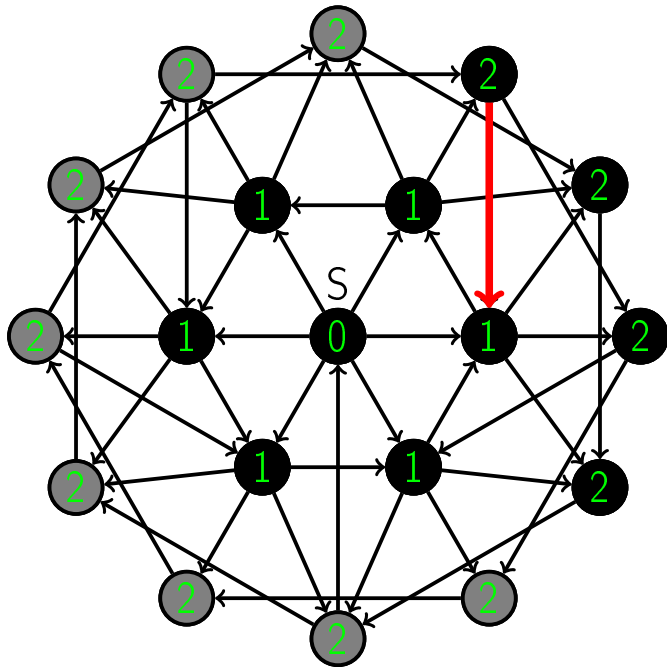


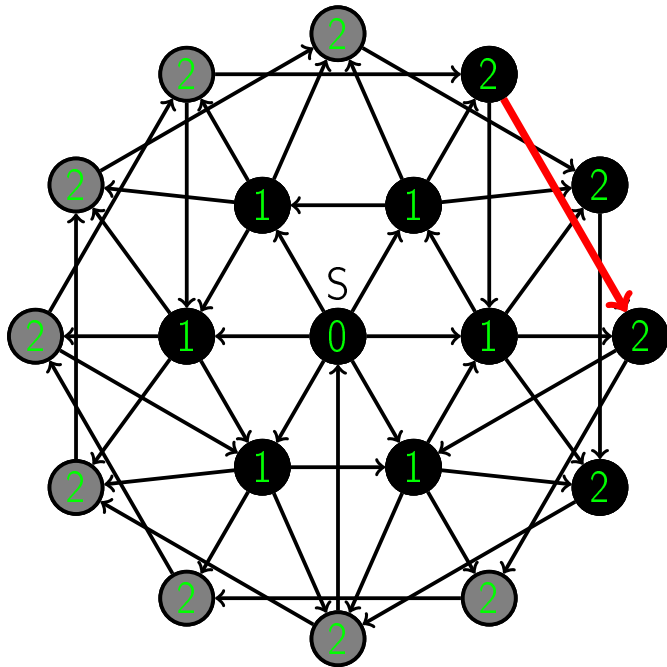


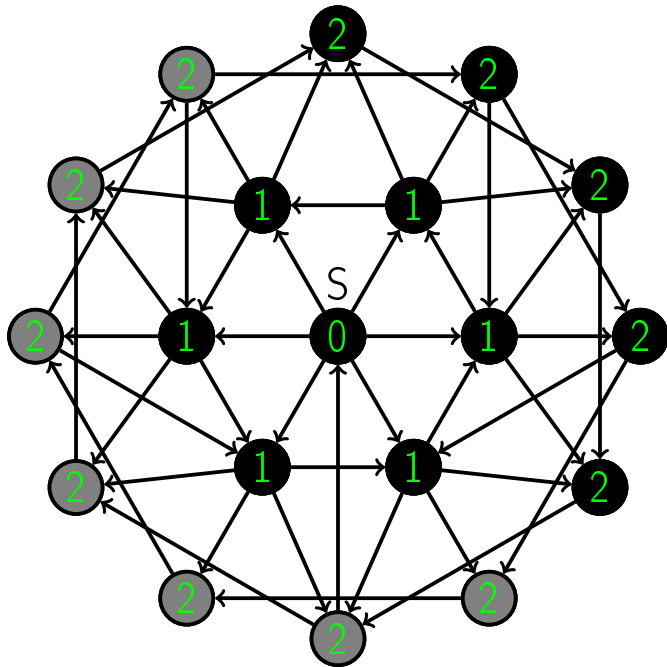


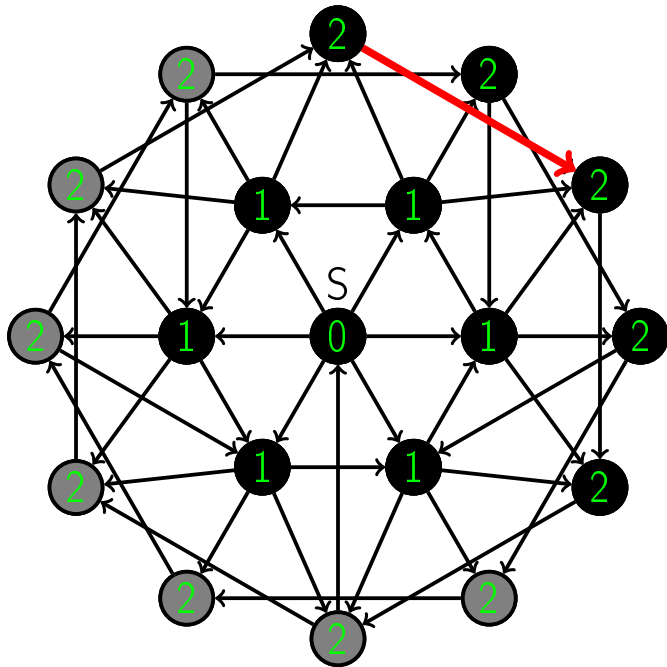


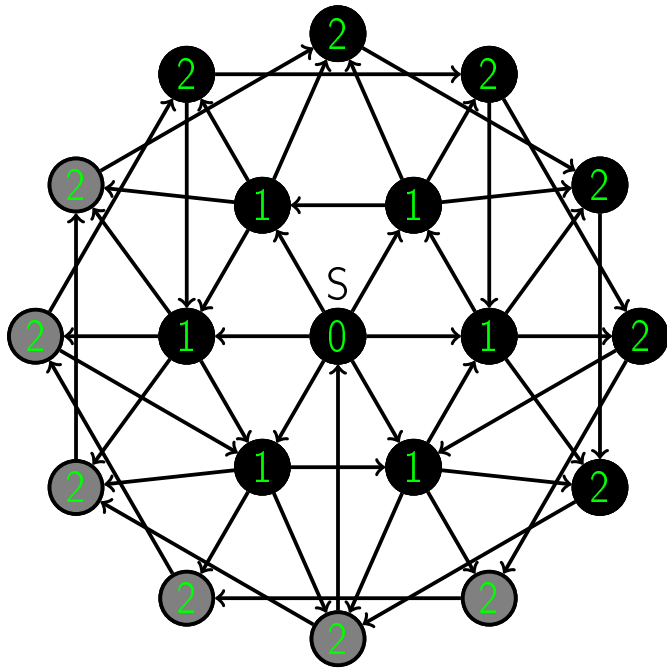


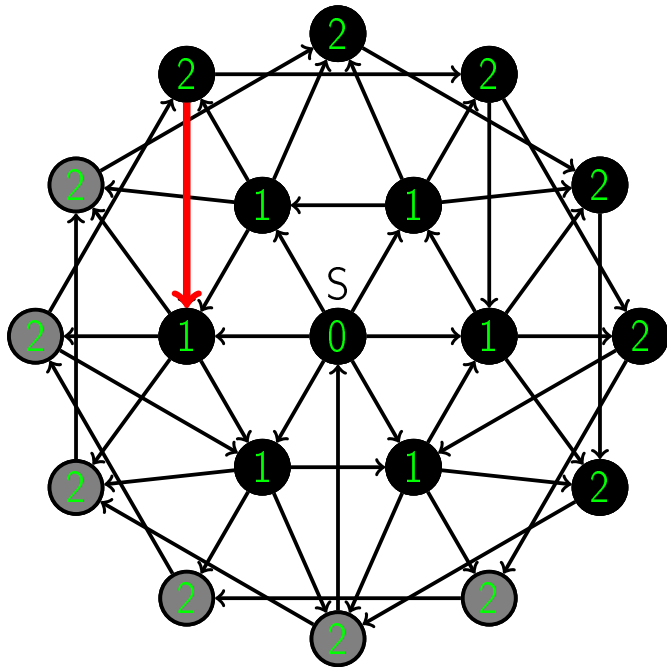


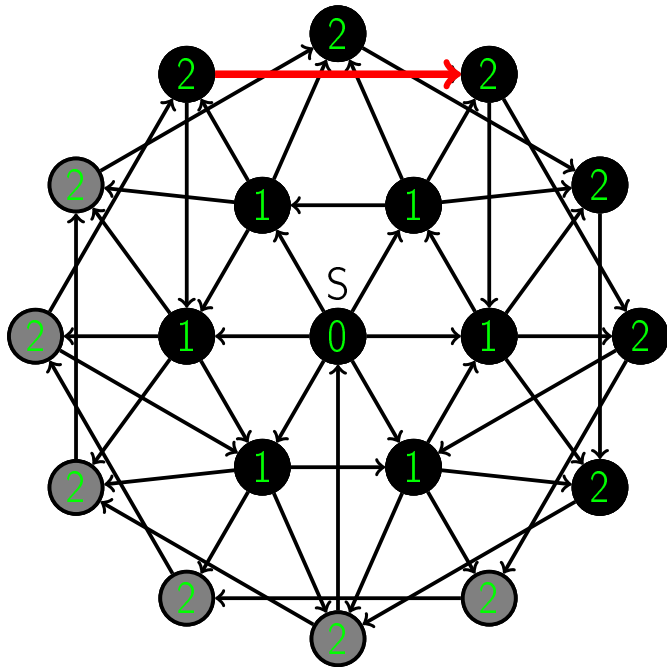


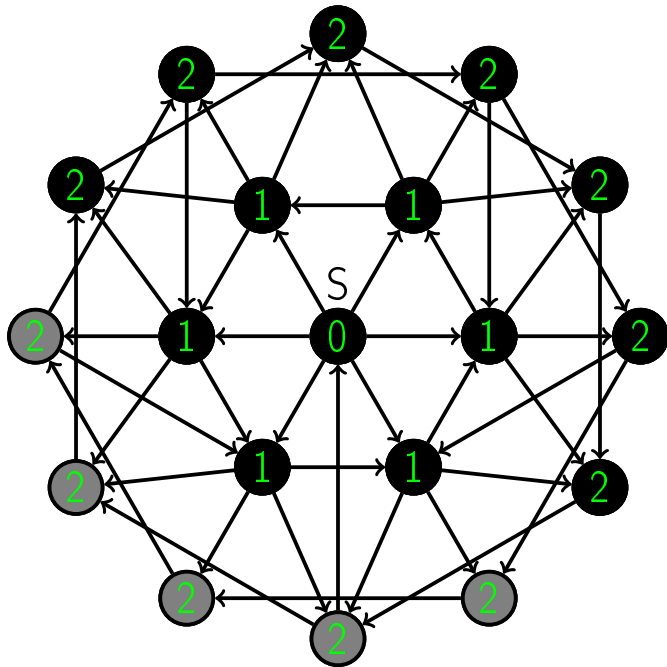




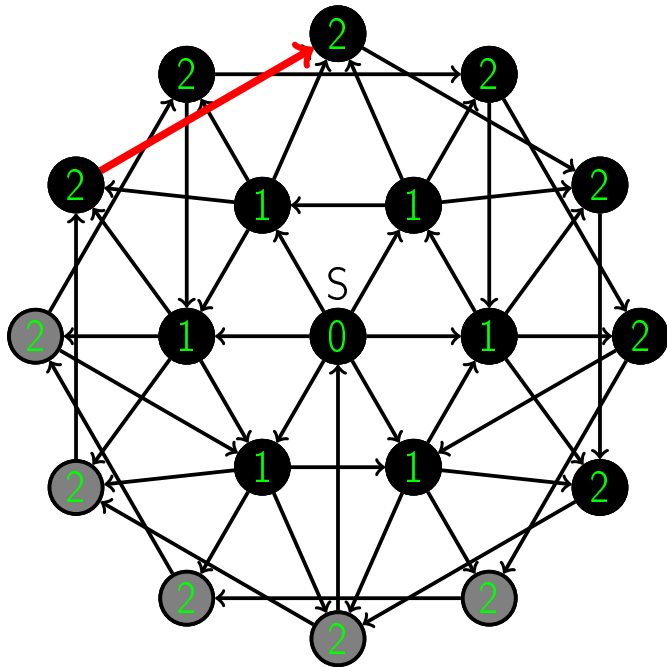


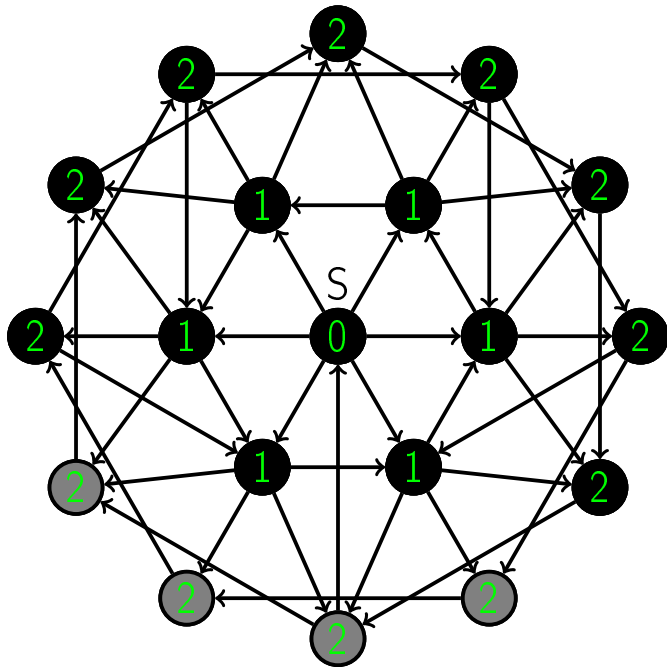


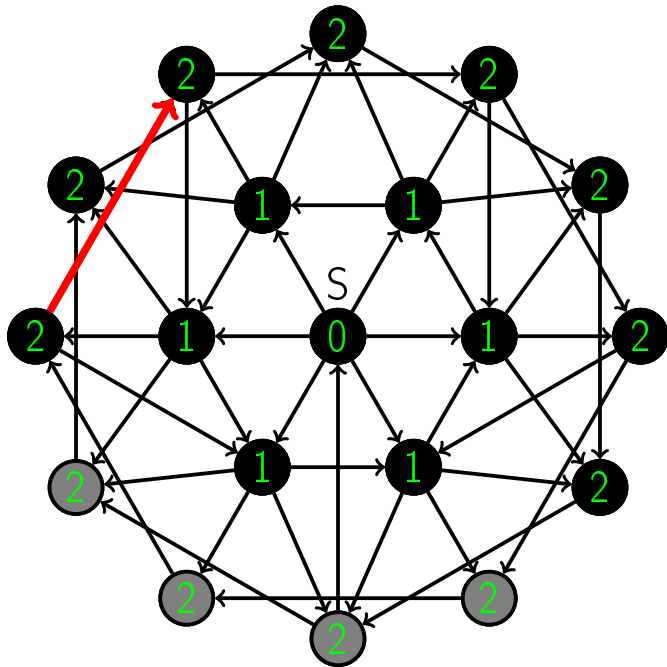


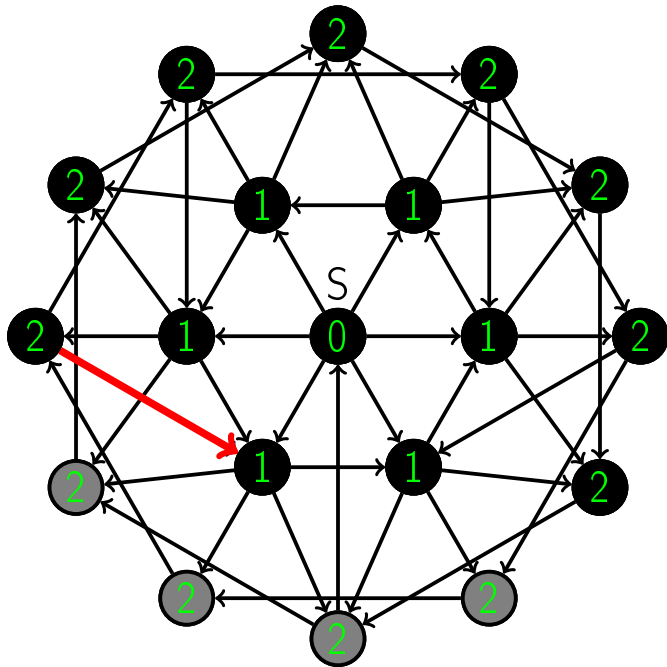


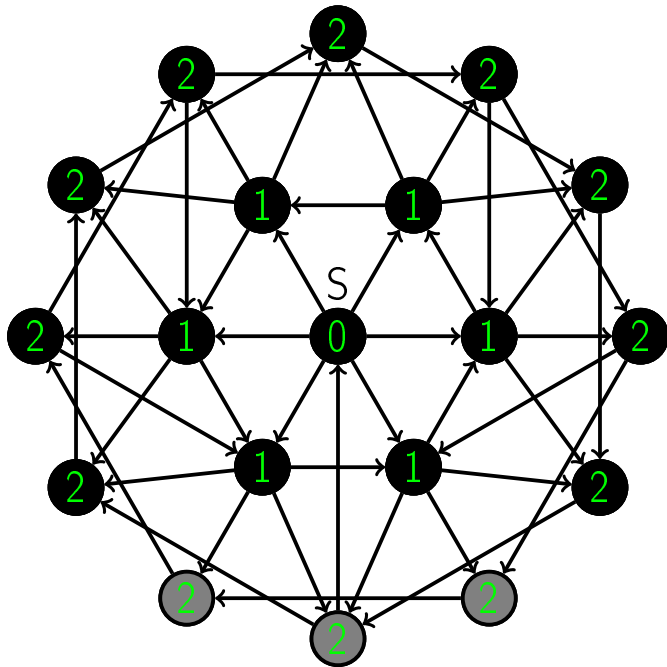


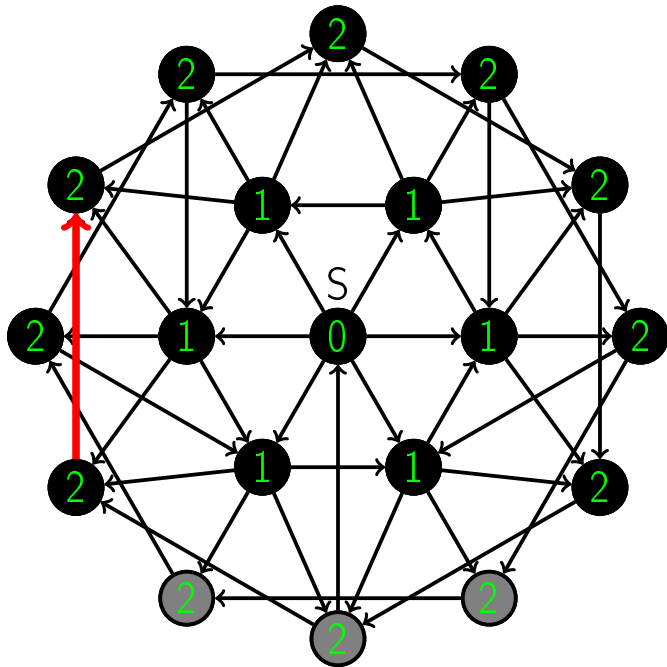


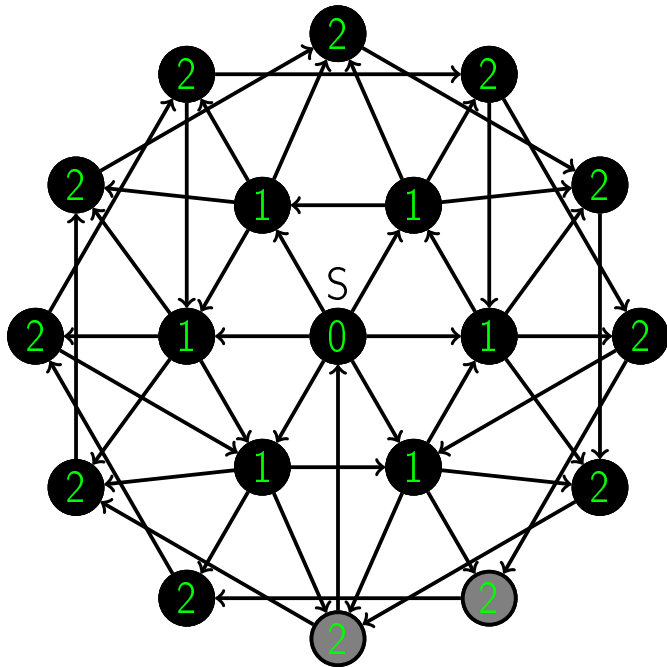


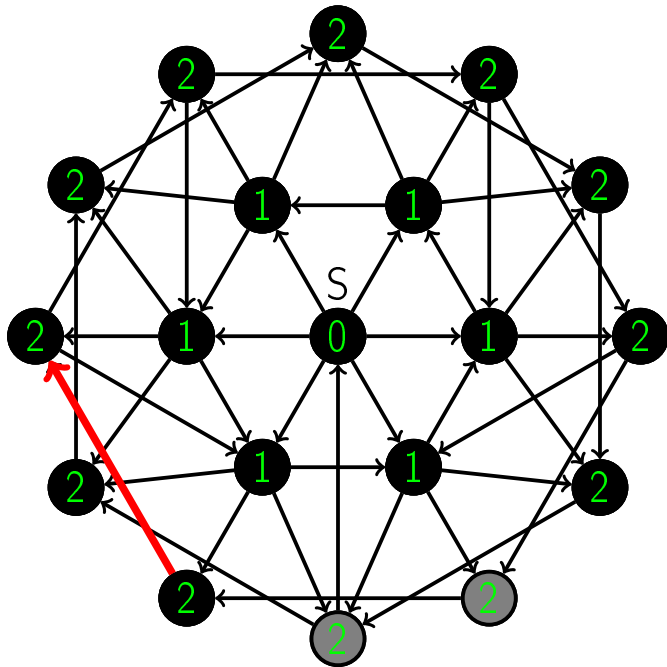




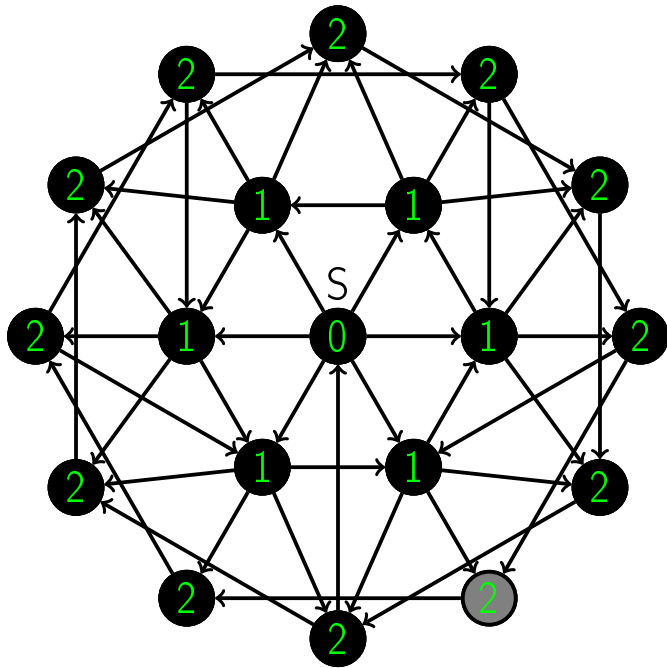


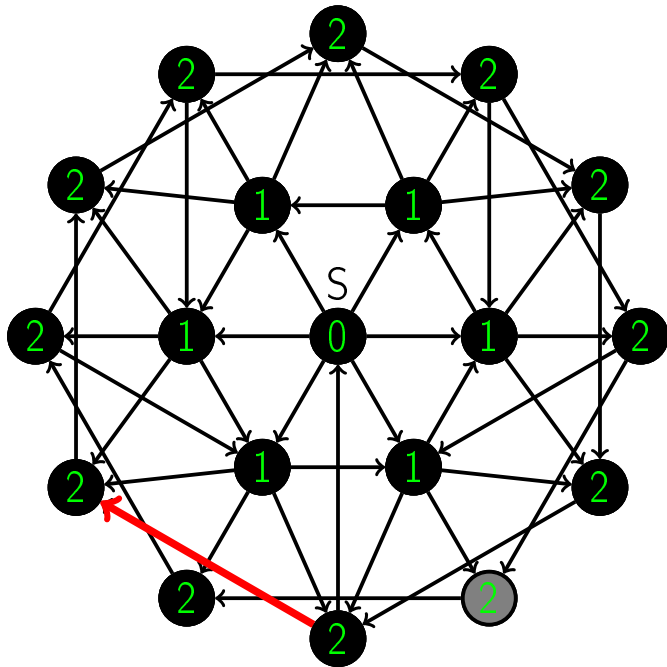


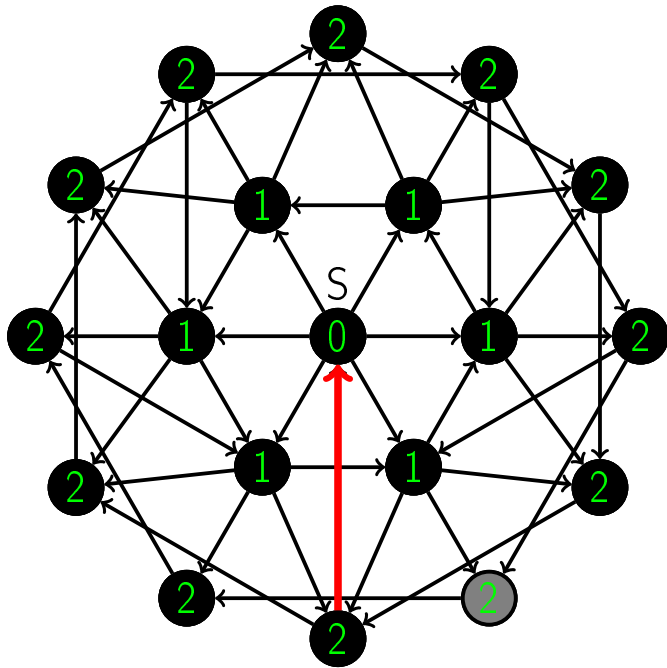


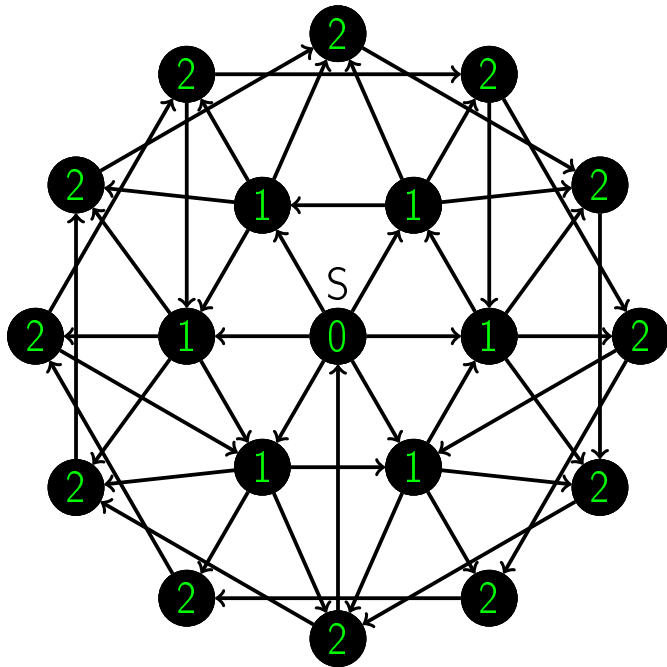


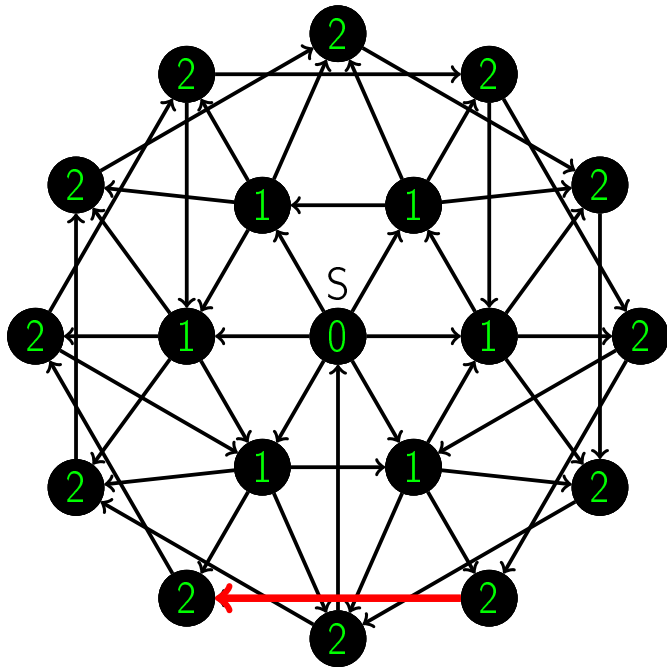


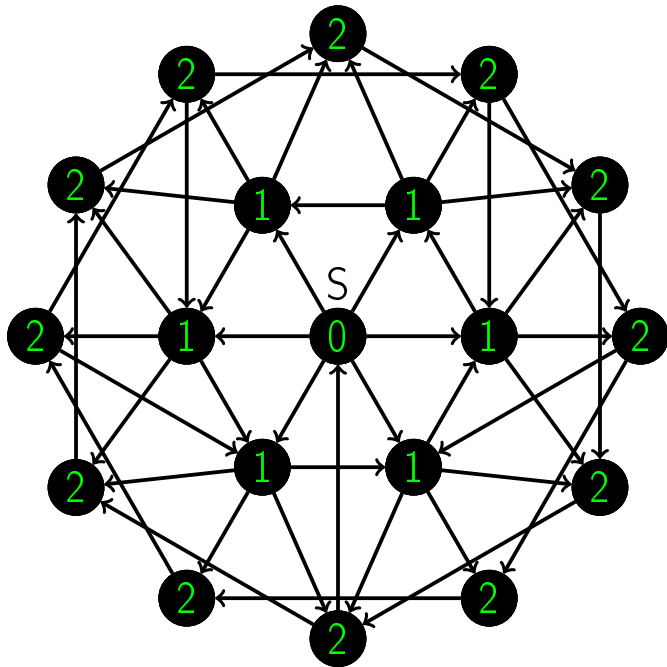


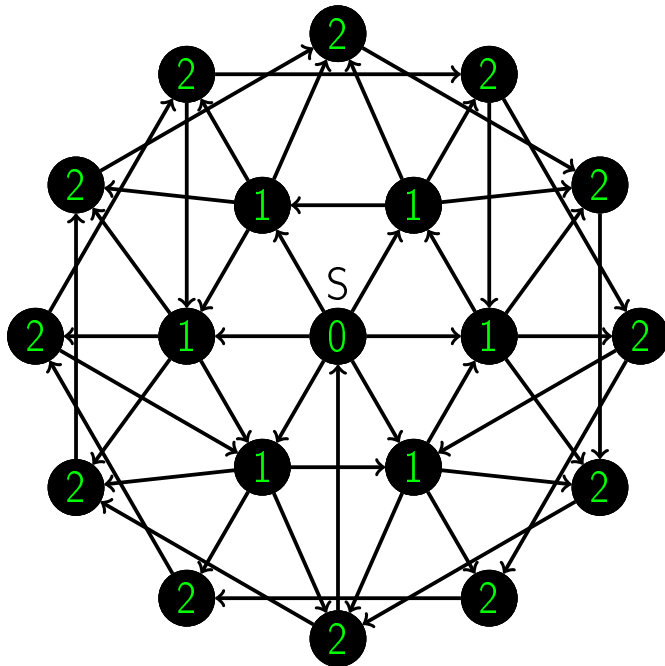












# Outline

- 1 Paths and Distances
- 2 Breadth-first Search
- 3 Implementation and Analysis
- 4 Proof of Correctness
- 5 Shortest-path Tree



# Breadth-first search

## BFS( $G, S$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty$   
 $\text{dist}[S] \leftarrow 0$   
 $Q \leftarrow \{S\}$  {queue containing just  $S$ }  
while  $Q$  is not empty:  
     $u \leftarrow \text{Dequeue}(Q)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] = \infty$ :  
             $\text{Enqueue}(Q, v)$   
             $\text{dist}[v] \leftarrow \text{dist}[u] + 1$ 
```

# Running time

## Lemma

The running time of breadth-first search is  $O(|E| + |V|)$ .

## Proof

# Running time

## Lemma

The running time of breadth-first search is  $O(|E| + |V|)$ .

## Proof


- Each vertex is enqueued at most once

# Running time

## Lemma

The running time of breadth-first search is  $O(|E| + |V|)$ .

## Proof

- Each vertex is enqueued at most once
  - Each edge is examined either once (for directed graphs) or twice (for undirected graphs)
- 

# Outline

- 1 Paths and Distances
- 2 Breadth-first Search
- 3 Implementation and Analysis
- 4 Proof of Correctness
- 5 Shortest-path Tree

# Reachability

## Definition

Node  $u$  is **reachable** from node  $S$  if there is a path from  $S$  to  $u$

## Lemma

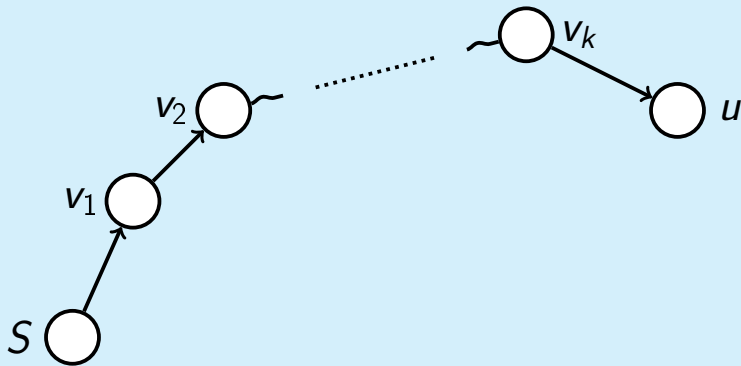
Reachable nodes are discovered at some point, so they get a finite distance estimate from the source. Unreachable nodes are not discovered at any point, and the distance to them stays infinite.

# Proof



- $u$  — reachable undiscovered closest to  $S$

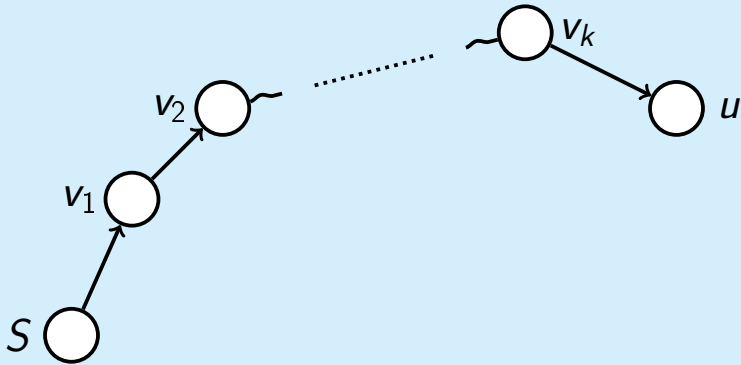
# Proof



- $u$  — reachable undiscovered closest to  $S$
- $S - v_1 - \dots - v_k - u$  — shortest path

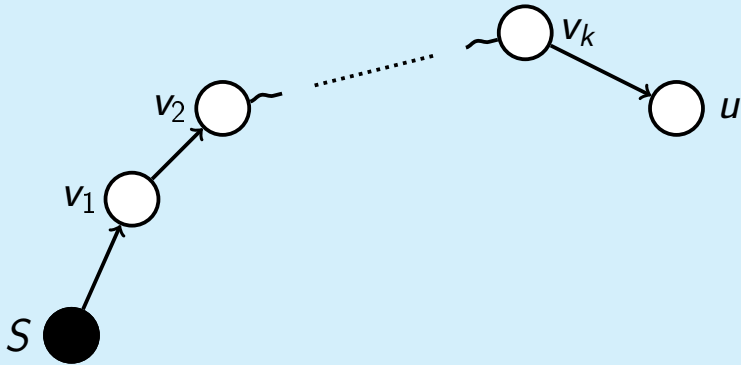


# Proof



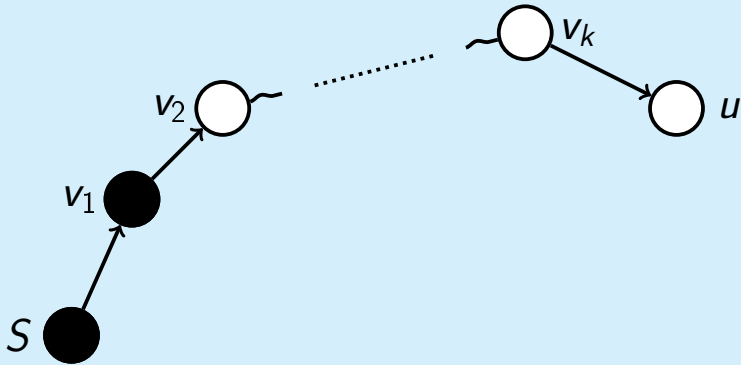
- $u$  — reachable undiscovered closest to  $S$
- $S - v_1 - \dots - v_k - u$  — shortest path
- $u$  is discovered while processing  $v_k$

# Proof



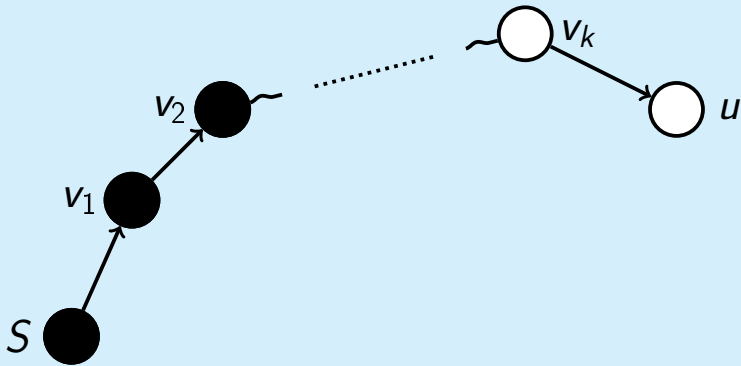
- $u$  — reachable undiscovered closest to  $S$
- $S - v_1 - \dots - v_k - u$  — shortest path
- $u$  is discovered while processing  $v_k$

# Proof



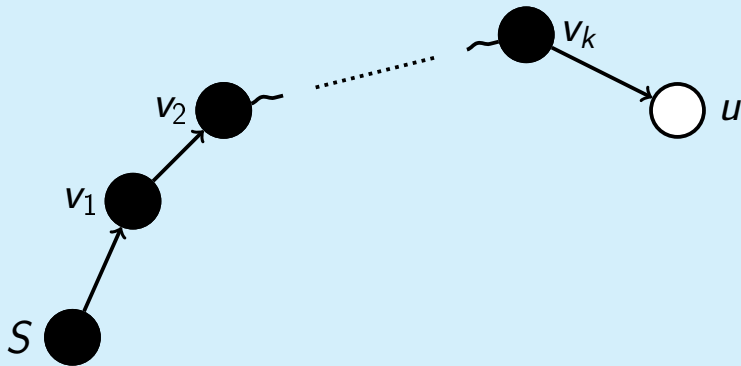
- $u$  — reachable undiscovered closest to  $S$
- $S - v_1 - \dots - v_k - u$  — shortest path
- $u$  is discovered while processing  $v_k$

# Proof



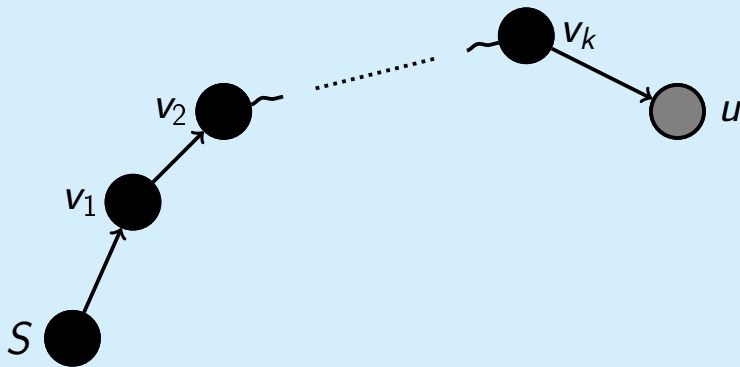
- $u$  — reachable undiscovered closest to  $S$
- $S - v_1 - \dots - v_k - u$  — shortest path
- $u$  is discovered while processing  $v_k$

# Proof



- $u$  — reachable undiscovered closest to  $S$
- $S - v_1 - \dots - v_k - u$  — shortest path
- $u$  is discovered while processing  $v_k$

# Proof



- $u$  — reachable undiscovered closest to  $S$
- $S - v_1 - \dots - v_k - u$  — shortest path
- $u$  is discovered while processing  $v_k$

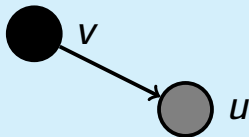
# Proof

$s$  ●

●  $u$

- $u$  — first unreachable discovered

# Proof

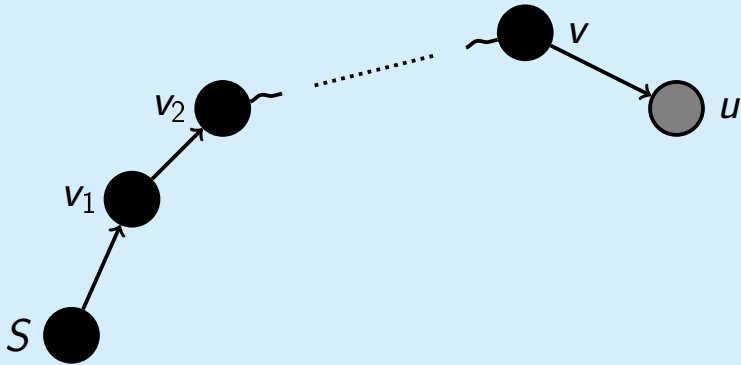


$s$  ●

- $u$  — first unreachable discovered
- $u$  was discovered while processing  $v$



# Proof



- $u$  — first unreachable discovered
- $u$  was discovered while processing  $v$
- $u$  is reachable through  $v$



# Order Lemma

## Lemma

By the time node  $u$  at distance  $d$  from  $S$  is dequeued, all the nodes at distance at most  $d$  have already been discovered (enqueued).

# Order Lemma Proof



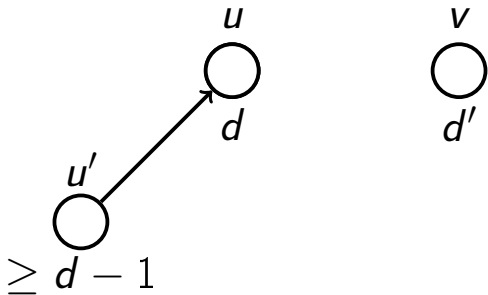
Consider the first time the order was broken

# Order Lemma Proof



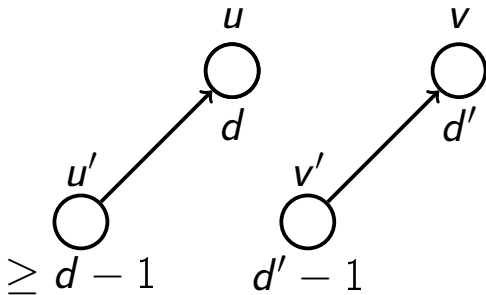
Consider the first time the order was broken  
 $d' \leq d$

# Order Lemma Proof



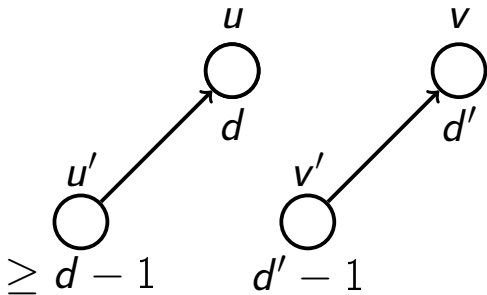
Consider the first time the order was broken  
 $d' \leq d$

# Order Lemma Proof



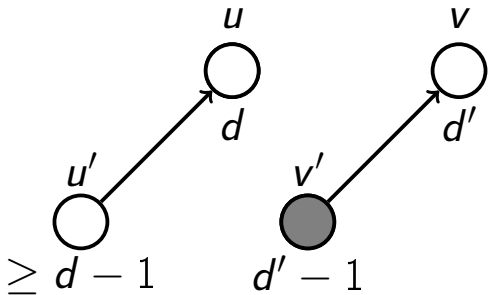
Consider the first time the order was broken  
 $d' \leq d$

# Order Lemma Proof



Consider the first time the order was broken  
 $d' \leq d \Rightarrow d' - 1 \leq d - 1$ , so  $v'$  was  
discovered before  $u'$  was dequeued

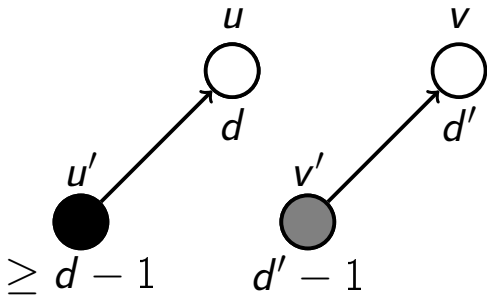
# Order Lemma Proof



Consider the first time the order was broken  
 $d' \leq d \Rightarrow d' - 1 \leq d - 1$ , so  $v'$  was  
discovered before  $u'$  was dequeued

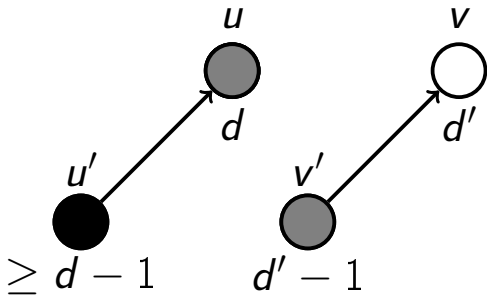


# Order Lemma Proof



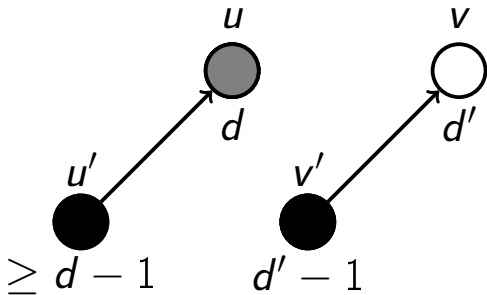
Consider the first time the order was broken  
 $d' \leq d \Rightarrow d' - 1 \leq d - 1$ , so  $v'$  was  
discovered before  $u'$  was dequeued

# Order Lemma Proof



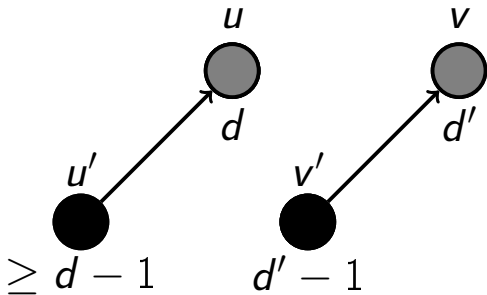
Consider the first time the order was broken  
 $d' \leq d \Rightarrow d' - 1 \leq d - 1$ , so  $v'$  was  
discovered before  $u'$  was dequeued

# Order Lemma Proof



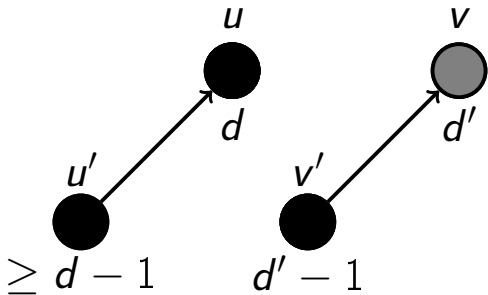
Consider the first time the order was broken  
 $d' \leq d \Rightarrow d' - 1 \leq d - 1$ , so  $v'$  was  
discovered before  $u'$  was dequeued

# Order Lemma Proof



Consider the first time the order was broken  
 $d' \leq d \Rightarrow d' - 1 \leq d - 1$ , so  $v'$  was  
discovered before  $u'$  was dequeued

# Order Lemma Proof



Consider the first time the order was broken  
 $d' \leq d \Rightarrow d' - 1 \leq d - 1$ , so  $v'$  was  
discovered before  $u'$  was dequeued

# Correct distances

## Lemma

When node  $u$  is discovered (enqueued),  $\text{dist}[u]$  is assigned exactly  $d(S, u)$ .

# Correct distances

## Proof

- Use mathematical induction

# Correct distances

## Proof

- Use mathematical induction
- Base: when  $S$  is discovered,  $\text{dist}[S]$  is assigned  $0 = d(S, S)$



# Correct distances

## Proof

- Use mathematical induction
- Base: when  $S$  is discovered,  $\text{dist}[S]$  is assigned  $0 = d(S, S)$
- Inductive step: suppose proved for all nodes at distance  $\leq k$  from  $S \rightarrow$  prove for nodes at distance  $k + 1$

# Correct distances

## Proof

- Take a node  $v$  at distance  $k + 1$  from  $S$

# Correct distances

## Proof

- Take a node  $v$  at distance  $k + 1$  from  $S$
- $v$  was discovered while processing  $u$

# Correct distances

## Proof

- Take a node  $v$  at distance  $k + 1$  from  $S$
- $v$  was discovered while processing  $u$
- $d(S, v) \leq d(S, u) + 1 \Rightarrow d(S, u) \geq k$

# Correct distances

## Proof

- Take a node  $v$  at distance  $k + 1$  from  $S$
- $v$  was discovered while processing  $u$
- $d(S, v) \leq d(S, u) + 1 \Rightarrow d(S, u) \geq k$
- $v$  is discovered after  $u$  is dequeued, so  $d(S, u) < d(S, v) = k + 1$

# Correct distances

## Proof

- Take a node  $v$  at distance  $k + 1$  from  $S$
- $v$  was discovered while processing  $u$
- $d(S, v) \leq d(S, u) + 1 \Rightarrow d(S, u) \geq k$
- $v$  is discovered after  $u$  is dequeued, so  $d(S, u) < d(S, v) = k + 1$
- So  $d(S, u) = k$ , and  $\text{dist}[v] \leftarrow \text{dist}[u] + 1 = k + 1$



# Queue property

Queue: 

$d$	$d$	$d$	$\dots$	$d$	$d$	$d + 1$	$d + 1$	$\dots$	$d + 1$
-----	-----	-----	---------	-----	-----	---------	---------	---------	---------

## Lemma

At any moment, if the first node in the queue is at distance  $d$  from  $S$ , then all the nodes in the queue are either at distance  $d$  from  $S$  or at distance  $d + 1$  from  $S$ . All the nodes in the queue at distance  $d$  go before (if any) all the nodes at distance  $d + 1$ .

# Queue property

## Proof

- All nodes at distance  $d$  were enqueued before first such node is dequeued, so they go before nodes at distance  $d + 1$



# Queue property

## Proof

- All nodes at distance  $d$  were enqueued before first such node is dequeued, so they go before nodes at distance  $d + 1$
- Nodes at distance  $d - 1$  were enqueued before nodes at  $d$ , so they are not in the queue anymore

# Queue property

## Proof

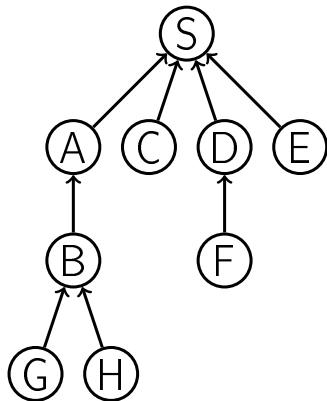
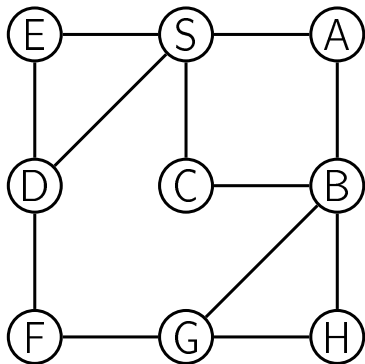
- All nodes at distance  $d$  were enqueued before first such node is dequeued, so they go before nodes at distance  $d + 1$
- Nodes at distance  $d - 1$  were enqueued before nodes at  $d$ , so they are not in the queue anymore
- Nodes at distance  $> d + 1$  will be discovered when all  $d$  are gone



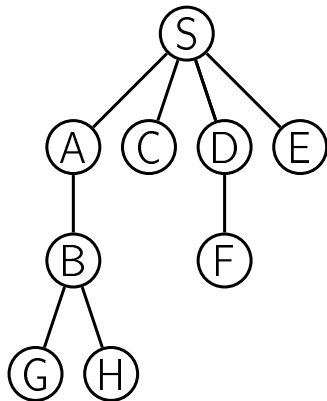
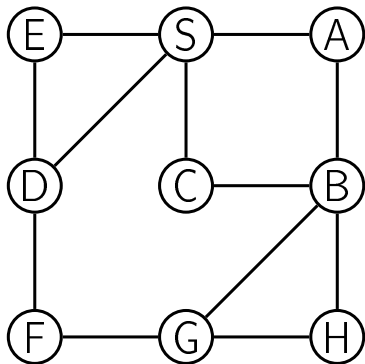
# Outline

- ① Paths and Distances
- ② Breadth-first Search
- ③ Implementation and Analysis
- ④ Proof of Correctness
- ⑤ Shortest-path Tree

# Shortest-path tree



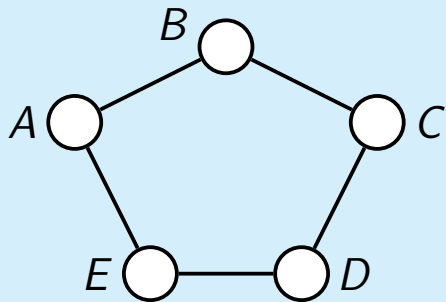
# Shortest-path tree



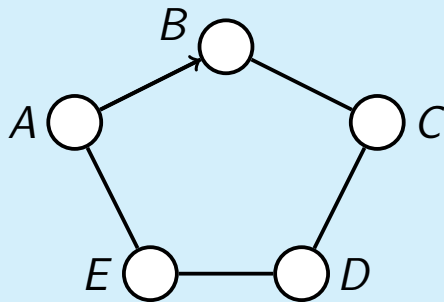
## Lemma

Shortest-path tree is indeed a tree, i.e. it doesn't contain cycles (it is a connected component by construction).

# Proof

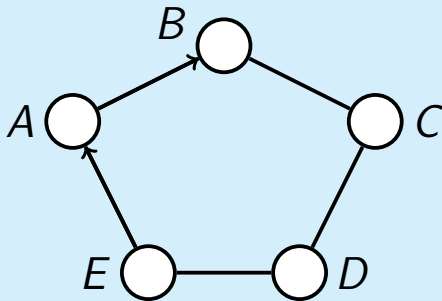


# Proof



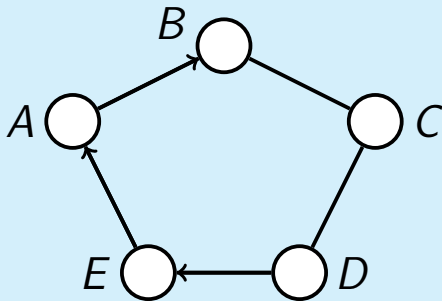


# Proof



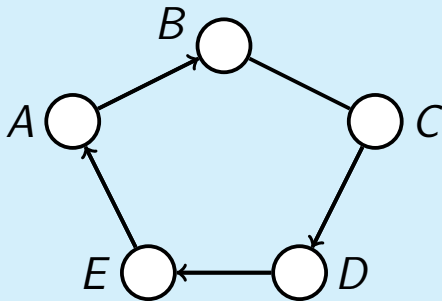
- Only one outgoing edge from each node

# Proof



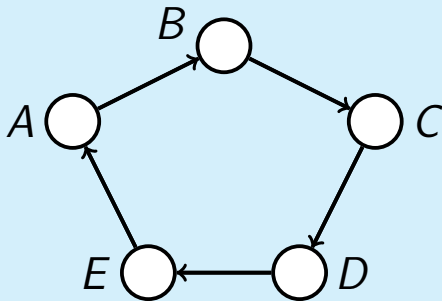
- Only one outgoing edge from each node

# Proof



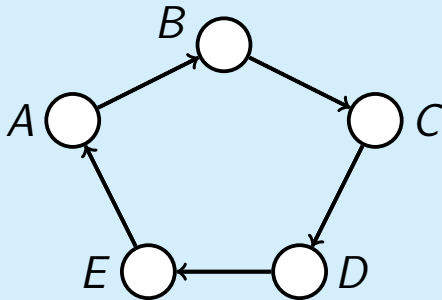
- Only one outgoing edge from each node

# Proof



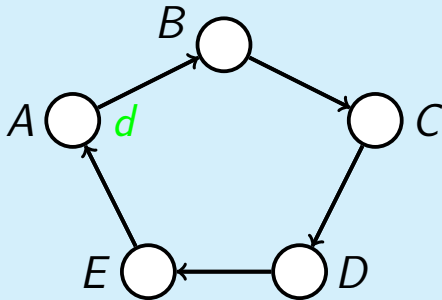
- Only one outgoing edge from each node

# Proof



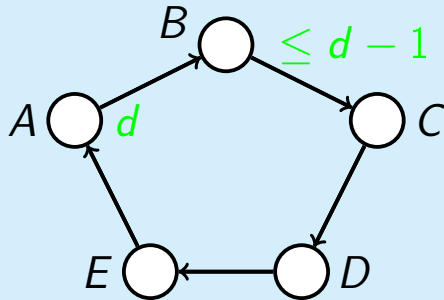
- Only one outgoing edge from each node
- Distance to  $S$  decreases after going by edge

# Proof



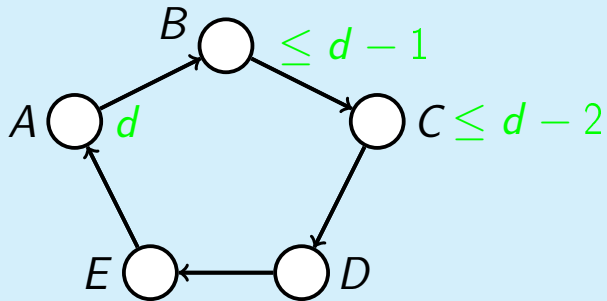
- Only one outgoing edge from each node
- Distance to  $S$  decreases after going by edge

# Proof



- Only one outgoing edge from each node
- Distance to  $S$  decreases after going by edge

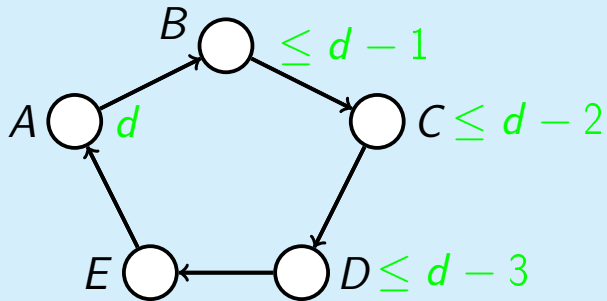
# Proof



- Only one outgoing edge from each node
- Distance to  $S$  decreases after going by edge

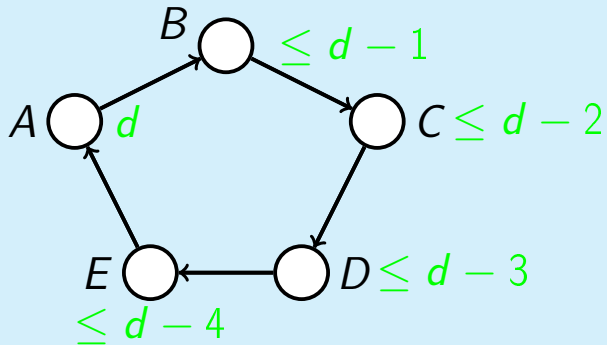


# Proof



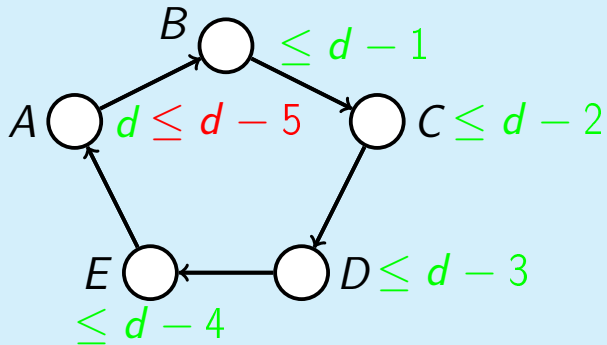
- Only one outgoing edge from each node
- Distance to  $S$  decreases after going by edge

# Proof



- Only one outgoing edge from each node
- Distance to  $S$  decreases after going by edge

# Proof



- Only one outgoing edge from each node
- Distance to  $S$  decreases after going by edge

# Constructing shortest-path tree

## BFS( $G, S$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty$ ,  $\text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[S] \leftarrow 0$   
 $Q \leftarrow \{S\}$  {queue containing just  $S$ }  
while  $Q$  is not empty:  
     $u \leftarrow \text{Dequeue}(Q)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] = \infty$ :  
             $\text{Enqueue}(Q, v)$   
             $\text{dist}[v] \leftarrow \text{dist}[u] + 1$ ,  $\text{prev}[v] \leftarrow u$ 
```

# Reconstructing Shortest Path

ReconstructPath( $S, u, \text{prev}$ )

```
result  $\leftarrow$  empty  
while  $u \neq S$ :  
    result.append( $u$ )  
     $u \leftarrow \text{prev}[u]$   
return Reverse(result)
```

# Conclusion

- Can find the minimum number of flight segments to get from one city to another

# Conclusion

- Can find the minimum number of flight segments to get from one city to another
- Can reconstruct the optimal path

# Conclusion

- Can find the minimum number of flight segments to get from one city to another
- Can reconstruct the optimal path
- Can build the tree of shortest paths from one origin



# Conclusion

- Can find the minimum number of flight segments to get from one city to another
- Can reconstruct the optimal path
- Can build the tree of shortest paths from one origin
- Works in  $O(|E| + |V|)$