DISTRIBUTED SPANNING TREES

LN 2

SPANNING TREES

- Spanning trees have many applications in computer networks as they provide a subgraph of less number of links than the original network resulting in lowered communications.

SPANNING TREES

- A spanning tree of a connected, undirected graph G(V,E) is its subgraph T (V,E) that covers (spans) all vertices of G .

- A spanning tree of a connected graph G can also be defined as a maximal set of edges of G that contains no cycle, or as a minimal set of edges that connect all vertices.

- A spanning forest of G is a subgraph of G that consists of a spanning tree in each connected component of G .

FLOOD

- Many applications in computer networks require sending a message to all nodes in the network that is called the broadcast .

- A natural way of performing broadcast in a network without any formed structure is to simply forward any incoming message to all neighbor nodes except the neighbor that has sent the message.

FLOOD

Algorithm 4.1 *Flood*

```
int i, j
boolean visited ← false
message types msg

if i = root then                                        ▷ root initiates flooding
    send msg to Γ(i)
    visited ← true
end if

receive flood(j)                                        ▷ flood may be received many times
if visited = false then                                 ▷ msg received first time
    send msg to Γ(i) \ {j}
    visited ← true
else                                                    ▷ msg received before
    discard msg
end if
```

Flood Analysis

- **Theorem 4.1** The message complexity of Flood is $O(m)$ where m is the number of edges of G, and the time complexity of Flood is $\Theta(d)$ where d is the diameter of G.

- **Proof.** Since each edge connects two nodes and is used to deliver a message at least once and at most twice when two nodes send msg concurrently, there will be a total of 2m messages at most, and therefore, $Msg(Flood) = O(m)$. The longest time for the broadcast message to reach any node in the graph G is the distance between two farthest nodes of the graph, which is the diameter, and hence, $Time(Flood) = \Theta(d)$.

Flooding-Based Asynchronous Spanning Tree Construction

- We can use the algorithm Flood by some modifications to build a spanning tree originating from the initiator root for broadcasting. We assume that it is required that each node in the tree except the leaf nodes should know the identifiers of its children and all nodes except the root should know their parents in the end.

- Any node that wants to build a broadcast tree initiates the algorithm and becomes the root of the spanning tree to be formed.

Flood_ST Algorithm

Algorithm 4.2 *Flood_ST*

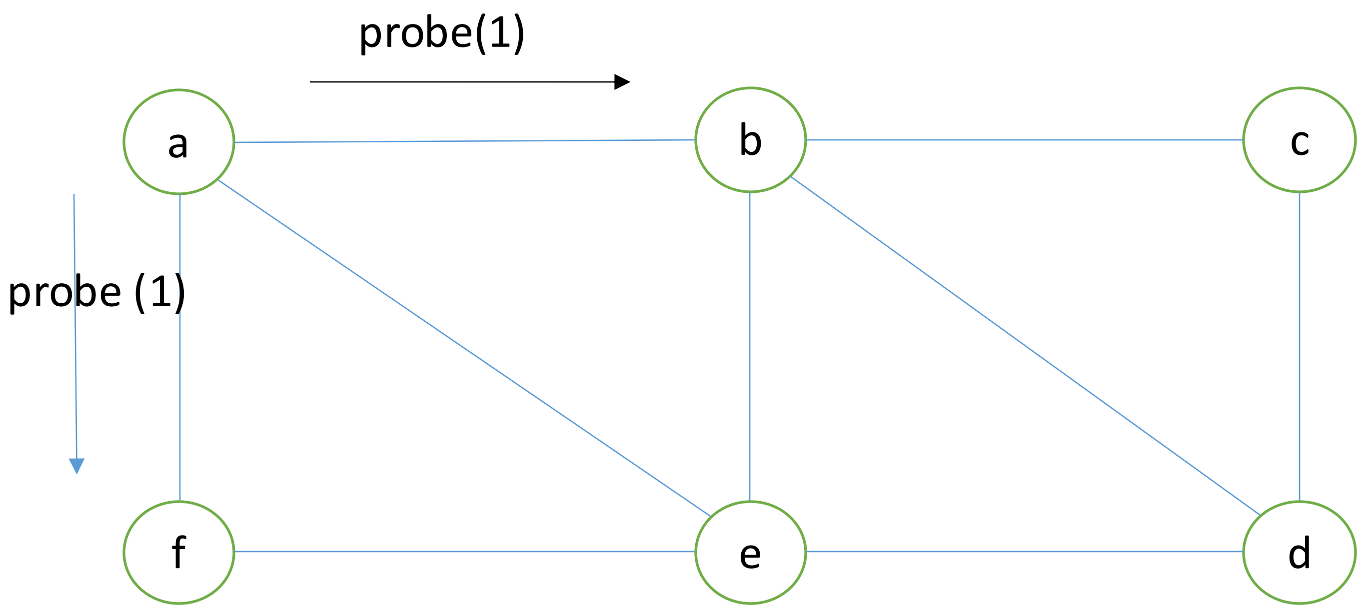
int *parent* $\leftarrow \perp$
set of int *childs* $\leftarrow \varnothing$, *others* $\leftarrow \varnothing$
message types *probe*, *ack*, *reject*

if $i = root$ **then** $\triangleright$ root initiates tree construction
  send *probe* to $\Gamma(i)$
  parent $\leftarrow i$
end if

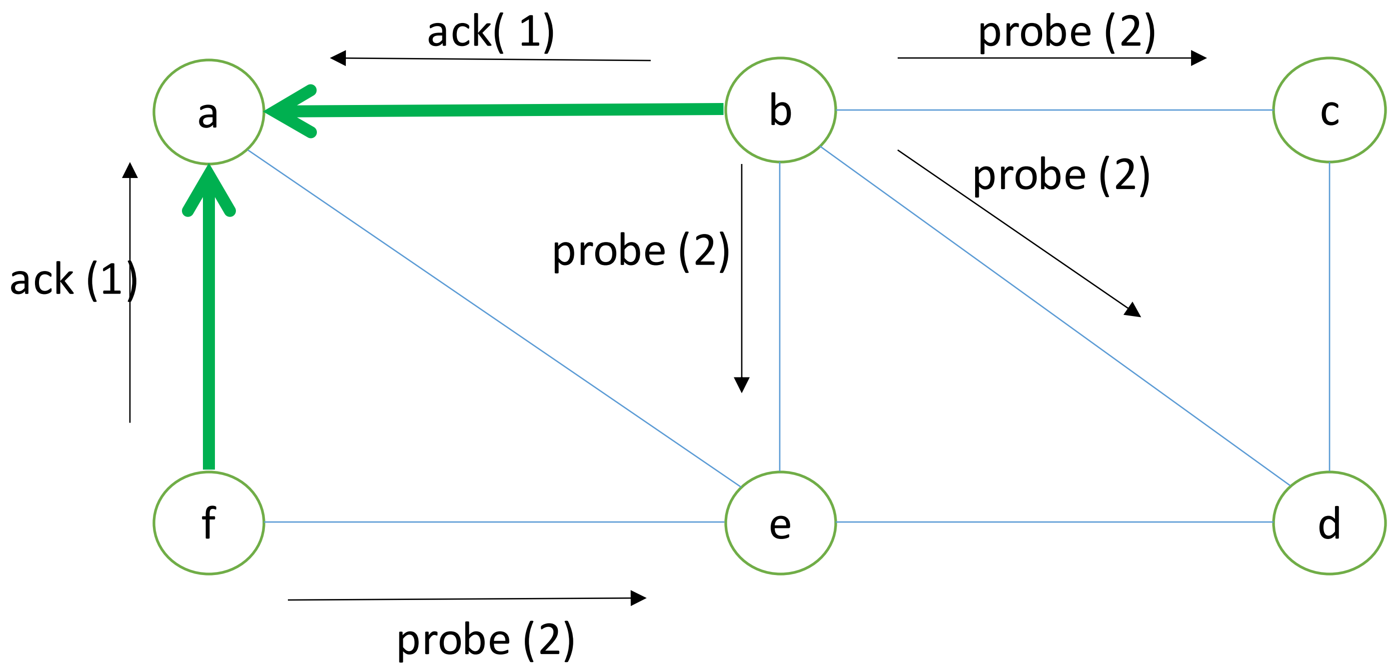
Flood_ST Algorithm

while $(childs \cup others) \neq (\Gamma(i) \backslash \{parent\})$ **do**
    receive $msg(j)$
    case $msg(j).type$ **of**
        <u>*probe*</u>: **if** $parent = \perp$ **then** ▷ *probe* received first time
                $parent \leftarrow j$
                send *ack* to j
                send *probe* to $\Gamma(i) \backslash \{j\}$
            else ▷ *probe* received before
                send *reject* to j
        <u>*ack*</u>: $childs \leftarrow childs \cup \{j\}$ ▷ include j in children
        <u>*reject*</u>: $others \leftarrow others \cup \{j\}$ ▷ include j in unrelated neighbors
end while

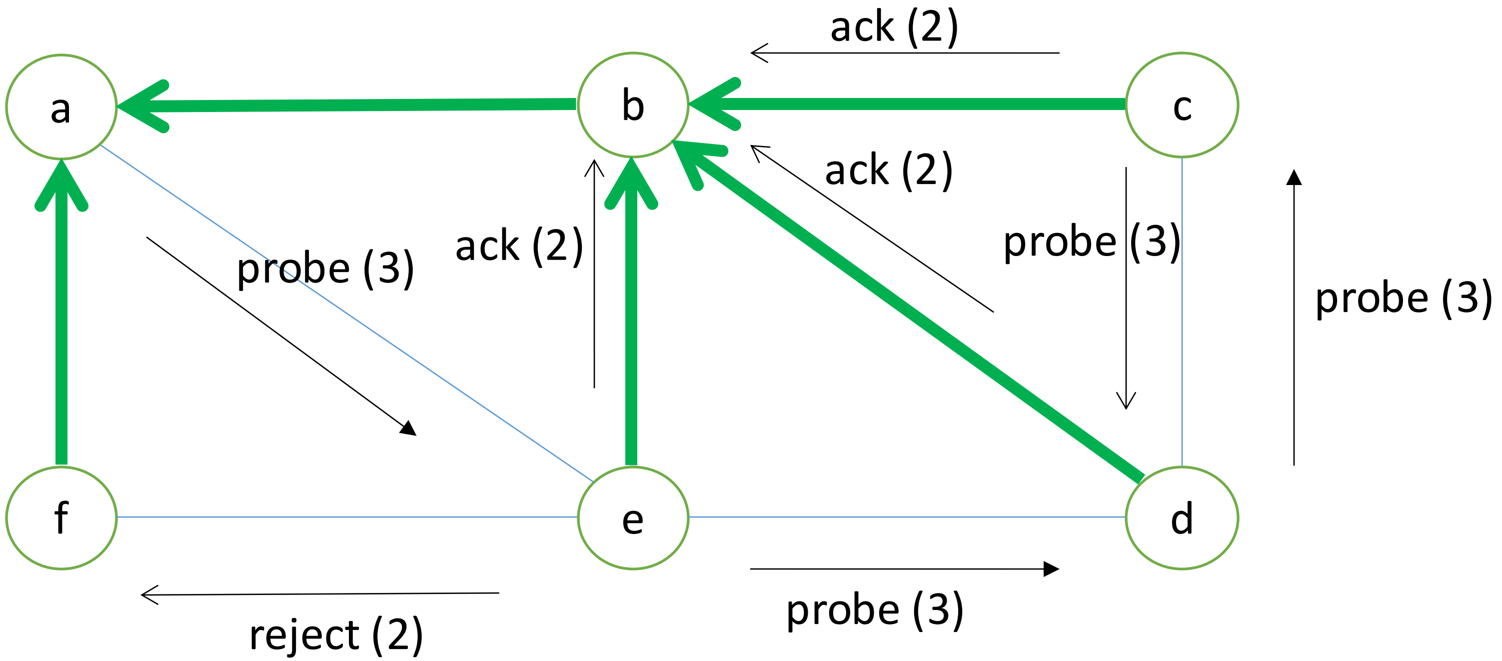
An example Spanning Tree in Flood_ST Alg.



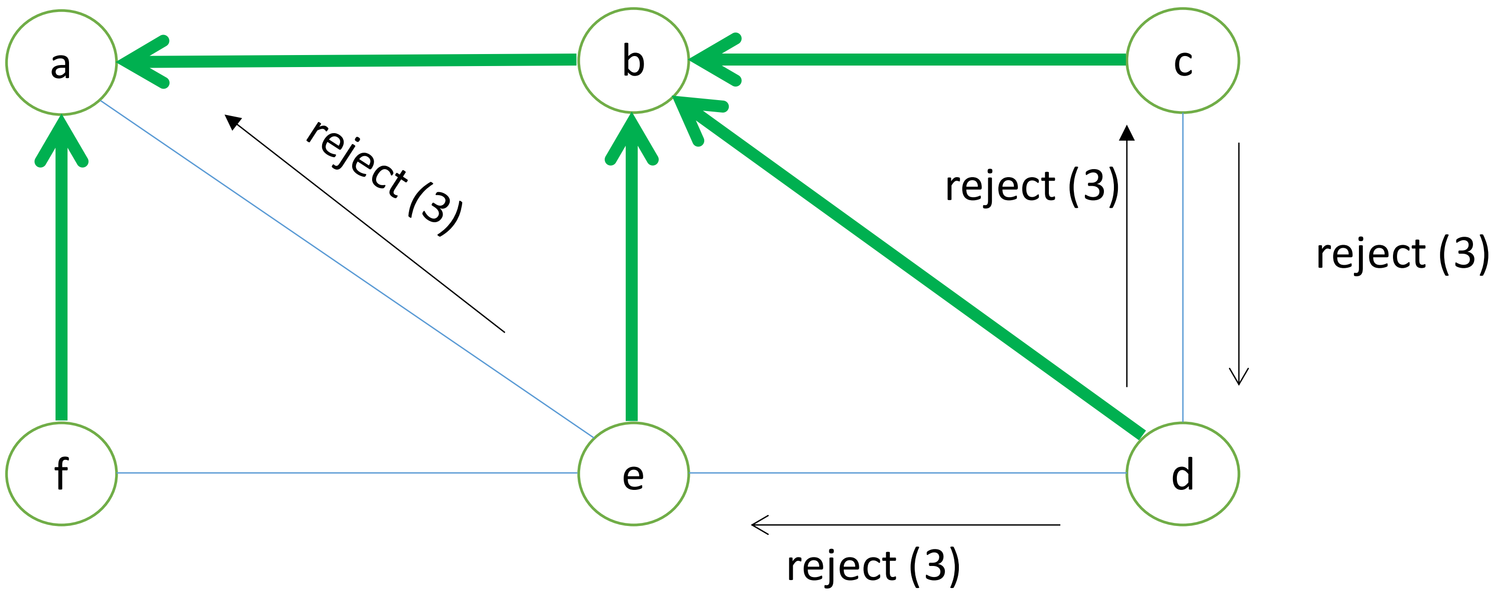
An example Spanning Tree in Flood_ST Alg.



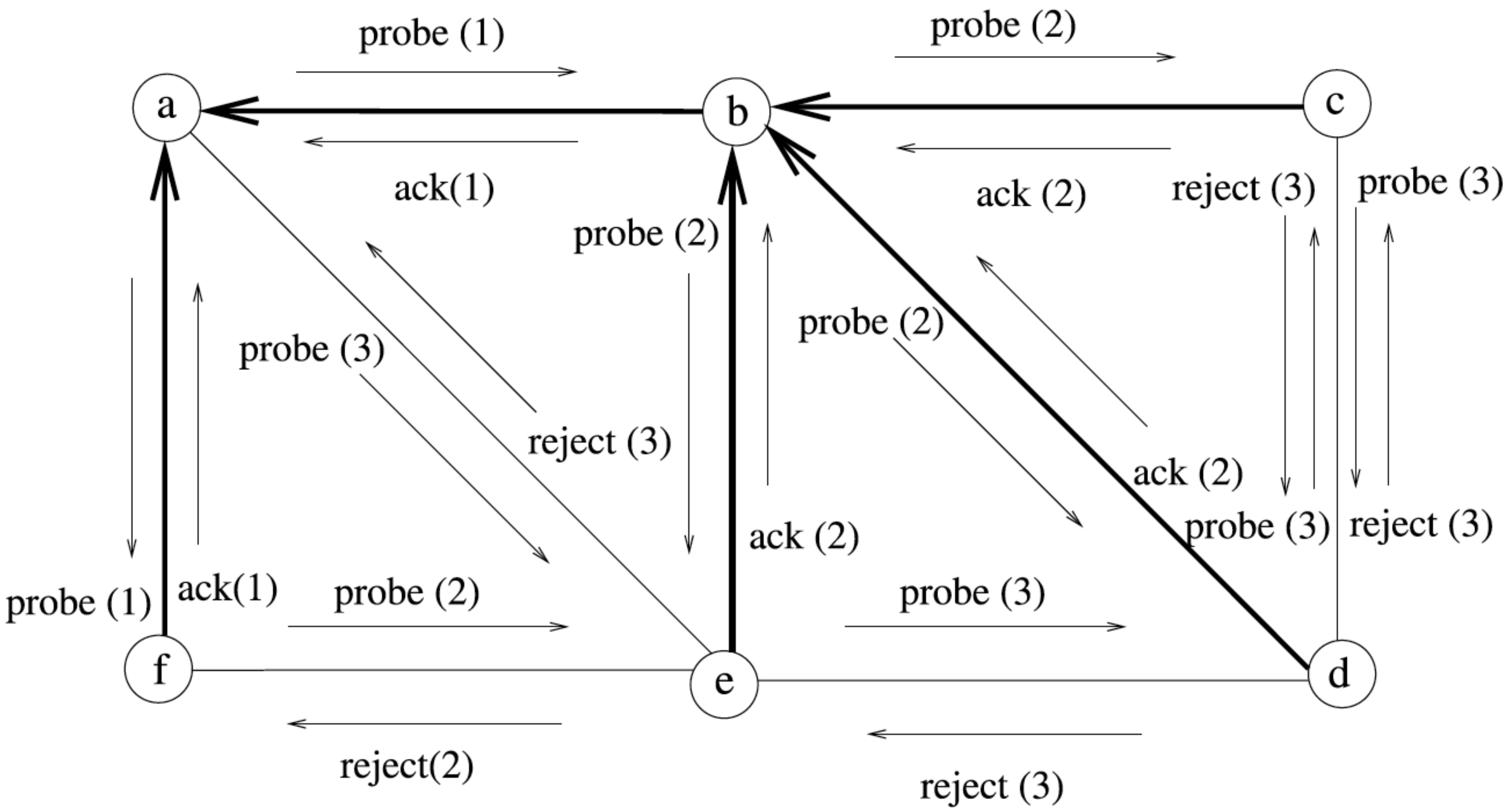
An example Spanning Tree in Flood_ST Alg.



An example Spanning Tree in Flood_ST Alg.



An example Spanning Tree in Flood_ST Alg.



Flood_ST Analysis

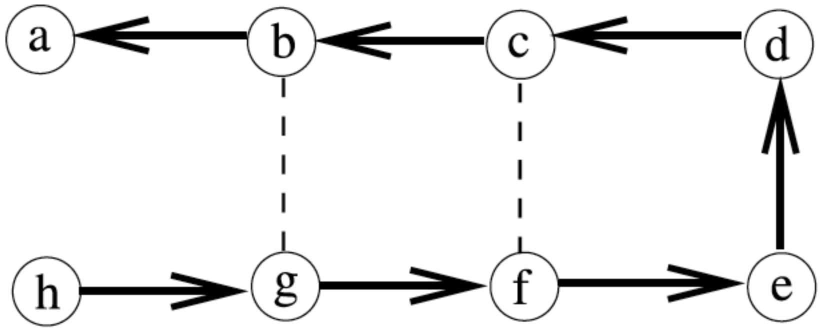
- Theorem 4.2 The message complexity of algorithm Flood_ST is O(m) where m is the number of edges of G, and it builds a tree T of maximum depth n−1. Assuming that there is at least one message transfer at each time unit, its time complexity is O(n).

- Proof. Proof Each edge of G will be traversed at least twice with probe and ack, or with probe and reject messages, or at most four times in the case of two nodes attempting to send each other probe messages concurrently. They will both reply with reject messages resulting in four messages for this edge for a total of 4m messages. Therefore, Msg(Flood_ ST) = O(m).

Flood_ST Analysis

- Proof. The depth of the tree constructed is O(n) considering the longest path. Assuming that there is at least one message transaction at each time step, the time complexity is bounded by the longest path in the graph, which has a length of n− 1.

Fig. 4.2 An example of the longest path



Flood_ST Analysis

- The problem with the Flood _ST algorithm is that although the root and nodes determine that their part of algorithm is over, they are not aware that the algorithm has terminated globally.

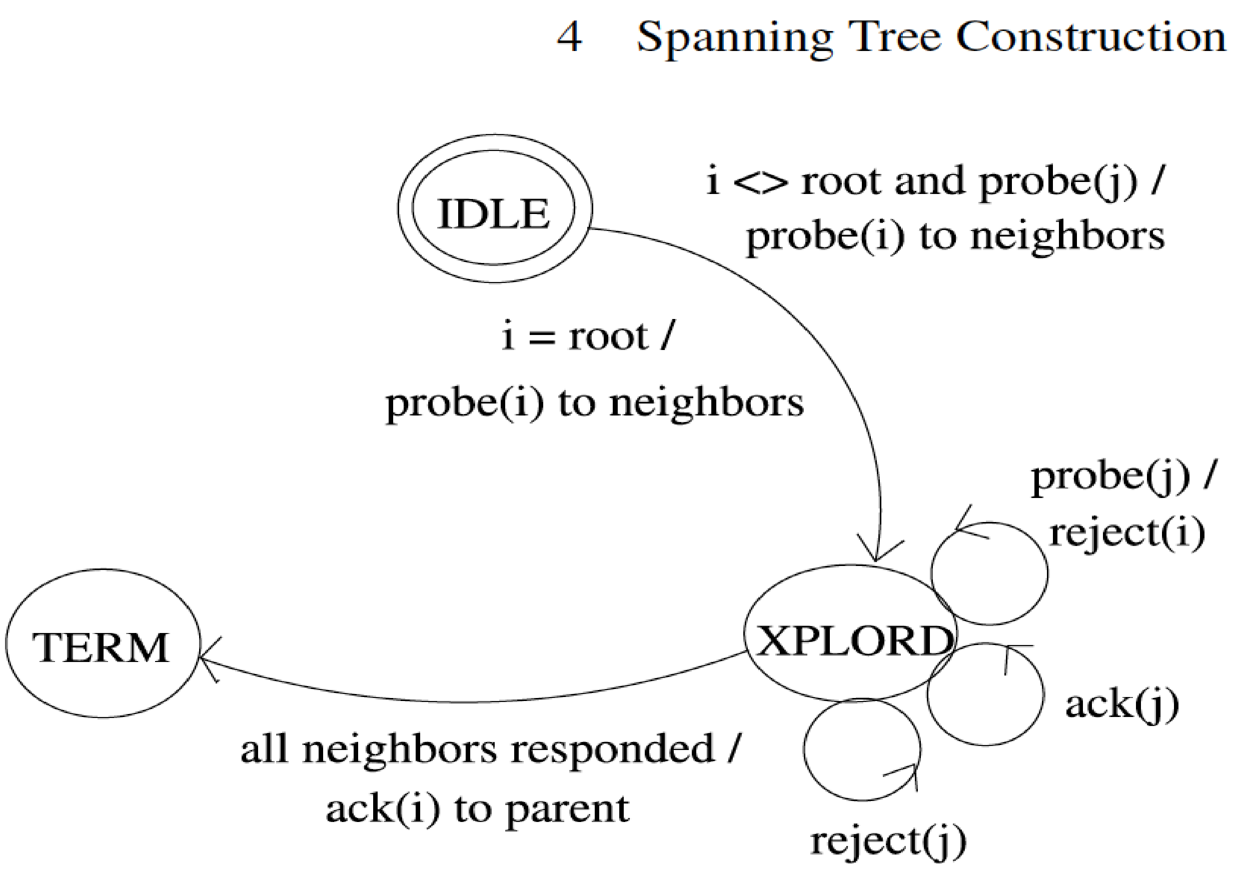
- The algorithm described in the next section provides the necessary modification to Flood _ST so that the nodes know when the algorithm has finished at least in their subtrees.

An Asynchronous Spanning Tree Algorithm with Termination Detection

- As a further attempt to build a spanning tree asynchronously, we will modify the previous algorithm so that termination of the construction is detected by the nodes.

- The modification is achieved by the nodes delaying the sending of the ack message to their parents until they receive ack or reject messages from their neighbors, rather than replying to their parents immediately.

An Asynchronous Spanning Tree Algorithm with Termination Detection



with Termination Detection

Algorithm 4.3 *Term_ST*

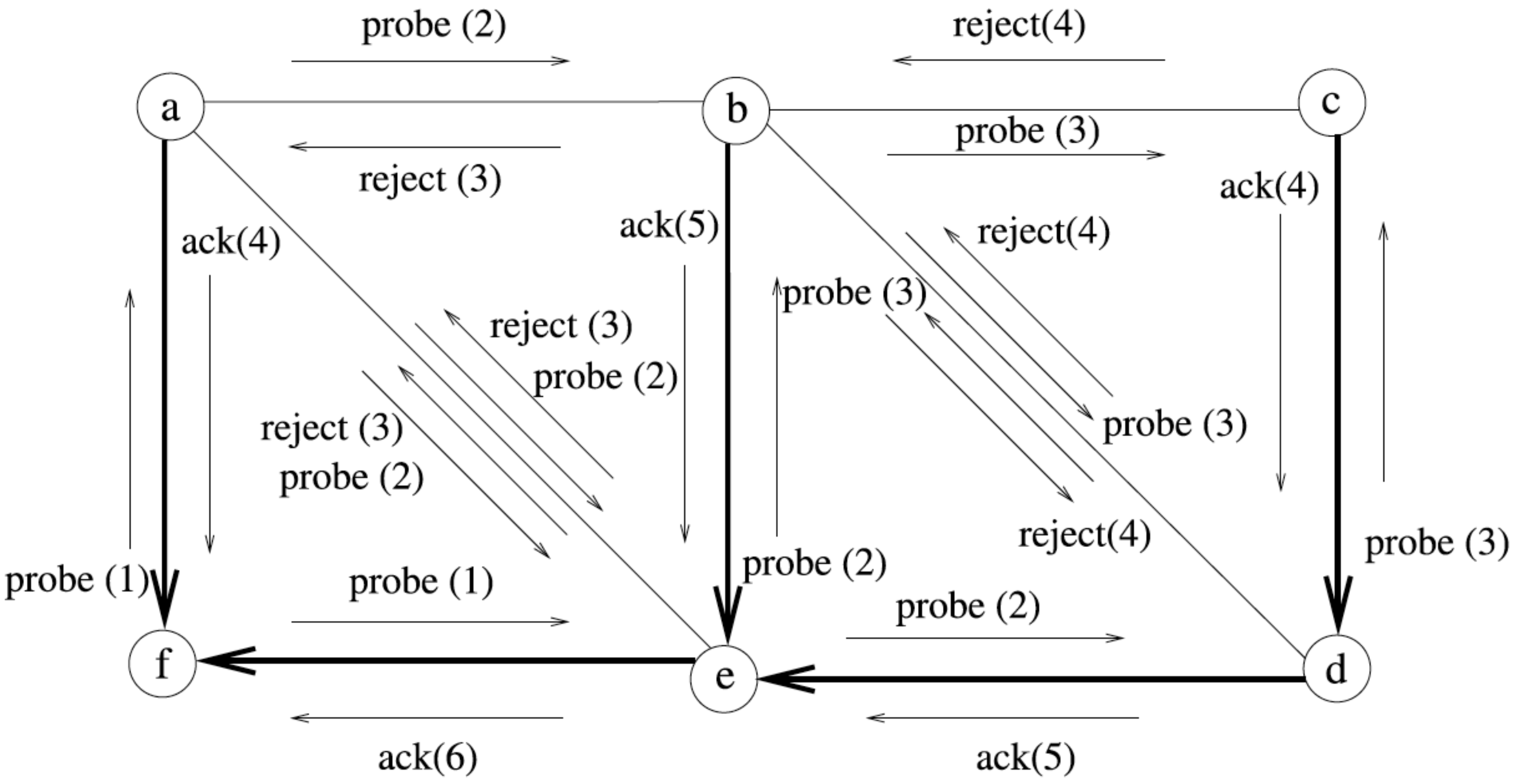
int $currstate \leftarrow IDLE$, $parent \leftarrow \perp$
set of int $childs \leftarrow \varnothing$, $others \leftarrow \varnothing$
message types *probe*, *ack*, *reject*
if $i =$ root **then**
    send *probe* to $\Gamma(i)$
    $currstate \leftarrow XPLORD$
end if

while $(childs \cup others) \neq (\Gamma(i) \setminus \{parent\})$ **do**
    receive$(msg(j))$;
    case *currstate* **of**
        IDLE:
            case $msg(j).type$ **of**
                probe: $parent \leftarrow j$ ▷ *probe* received first time
                    send *probe* to $\Gamma(i) \setminus \{j\}$
                    $currstate \leftarrow XPLORD$

with Termination Detection

```
            XPLORD:
                case msg(j).type of
                    probe:     send reject to j           ▷ probe received before
                    ack:       childs ← childs ∪ {j}
                    reject:    others ← others ∪ {j}
end while
if i ≠ root then                                          ▷ convergecast ack to root
    send ack(i) to parent
end if
currstate ← TERM
```

An Example Scenario With Term_ST



Analysis of TERM_ST

- **Theorem 4.3** The message complexity of algorithm Term_ST is O(m), and assuming that there is at least one message transfer at each time unit, its time complexity is O(n).

- Proof. The message complexity can be determined as in the Flood _ST algorithm to result in O(m) . Due to the asynchronous operation, messages may take the longest path of length $n-1$ to form the tree, and since there will be n− 1 more steps for the reply messages to be gathered at the root along this longest path, there will be a total of $2n - 2$ time steps at most, considering there is at least one message transfer at each time unit. Time(Term _ST) is therefore O(n) . Otherwise, the time to construct a spanning tree is unbounded.

Tarry’s Spanning Tree Algorithm

- The algorithm based on these two rules:
  - 1. A process never forwards the token twice through the same channel.
  - 2. A non-initiator forwards the token to its parent, the node from which it received the token for the first time, only if there is no other channel left according to Rule 1.

Tarry’s Spanning Tree Algorithm

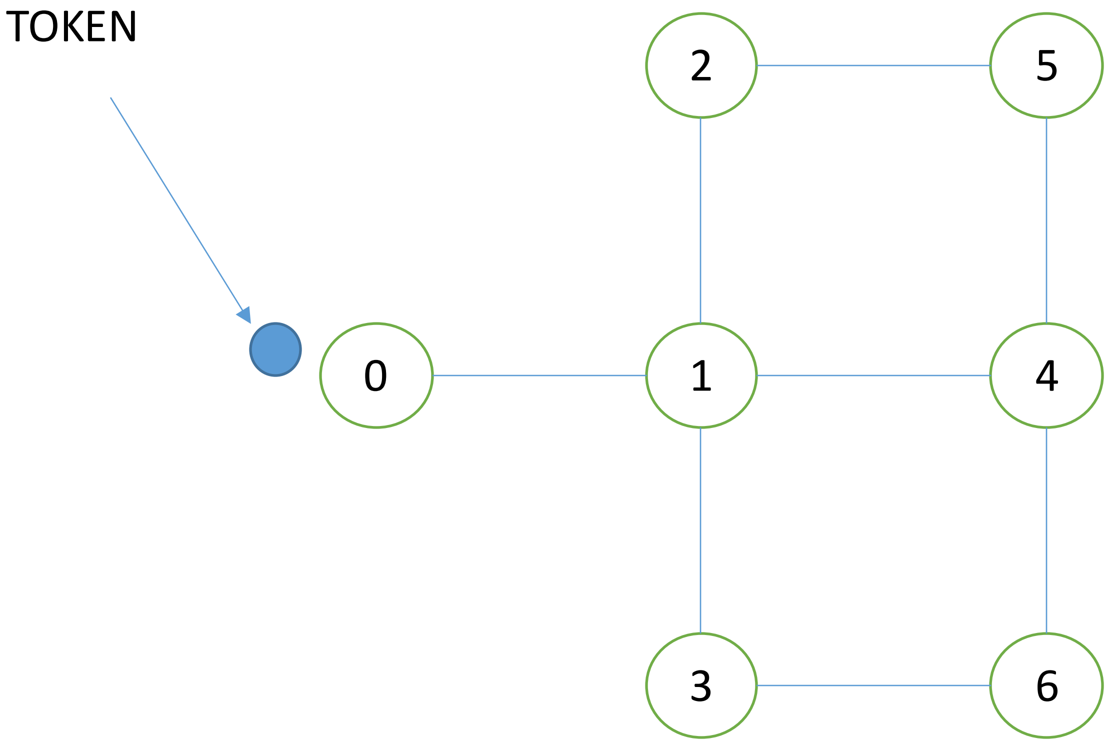
Algorithm 4.4 *Tarry_ST*

```
int parent ← ⊥
boolean used[n] ← {false}
message types token

if i = root then                                    ▷ root starts the search
    send token(i) to any j ∈ Γ(i)
    used[j] ← true, parent ← i
end if

while true do
    receive token(j)
    if parent = ⊥ then                              ▷ token first time
        parent ← j
    end if
```

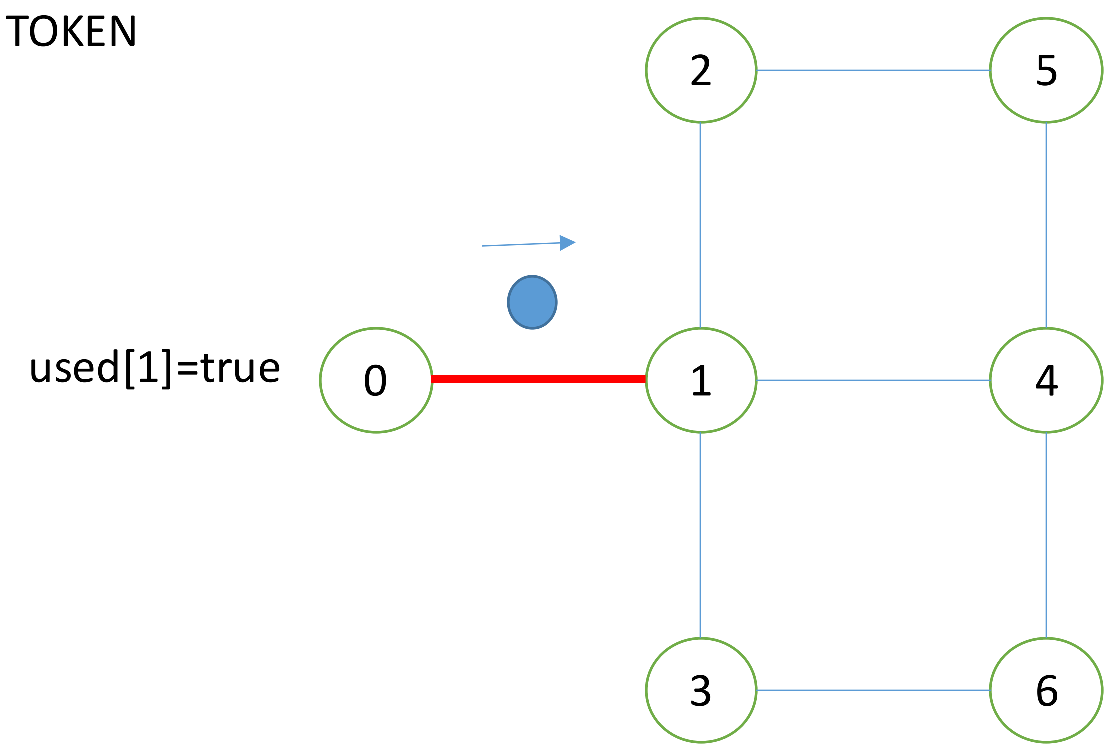
Tarry’s Spanning Tree Algorithm Cont’d

```
    if ∃j ∈ (Γ(i) \ parent) : ¬used[j] then    ▷ choose an unsearched neighbor ≠ parent
        send token to j
        used[j] ← true
    else
        if i ≠ root then
            used[parent] ← true                 ▷ all neighbors searched
            send token to parent
        end if
        exit                                    ▷ terminate
    end if
end while
```

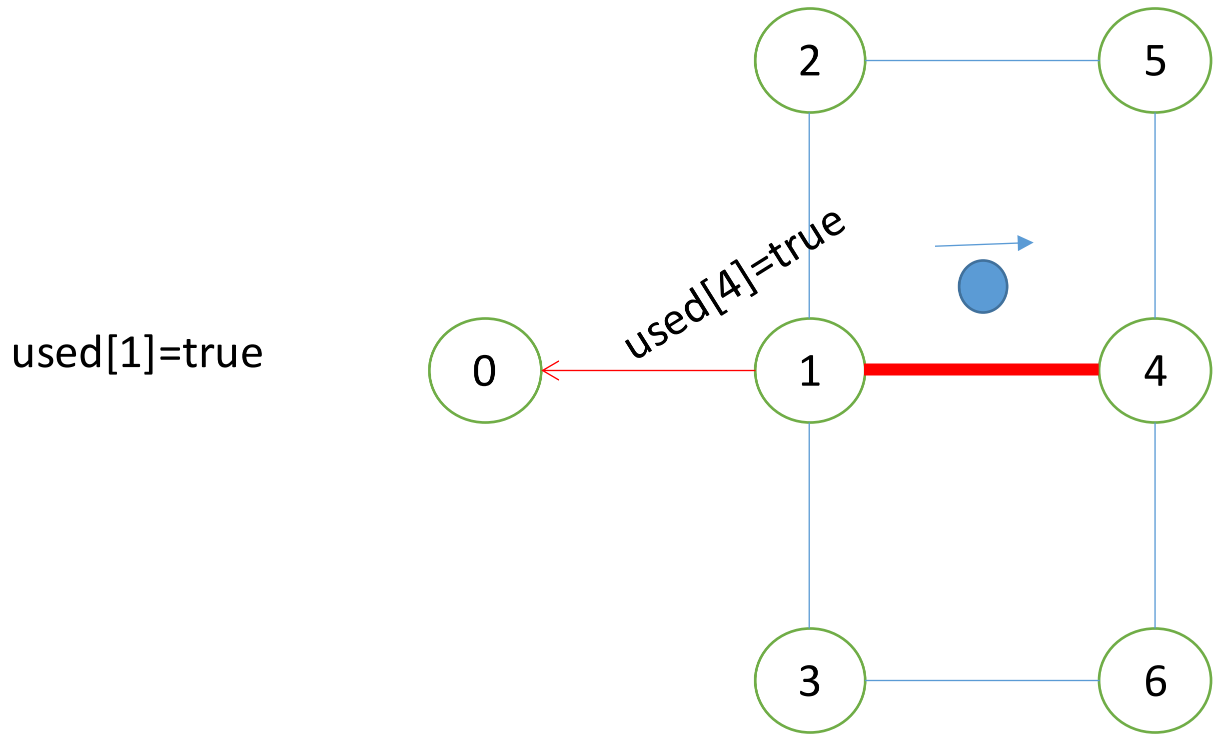
Tarry's Spanning Tree Example Scenario



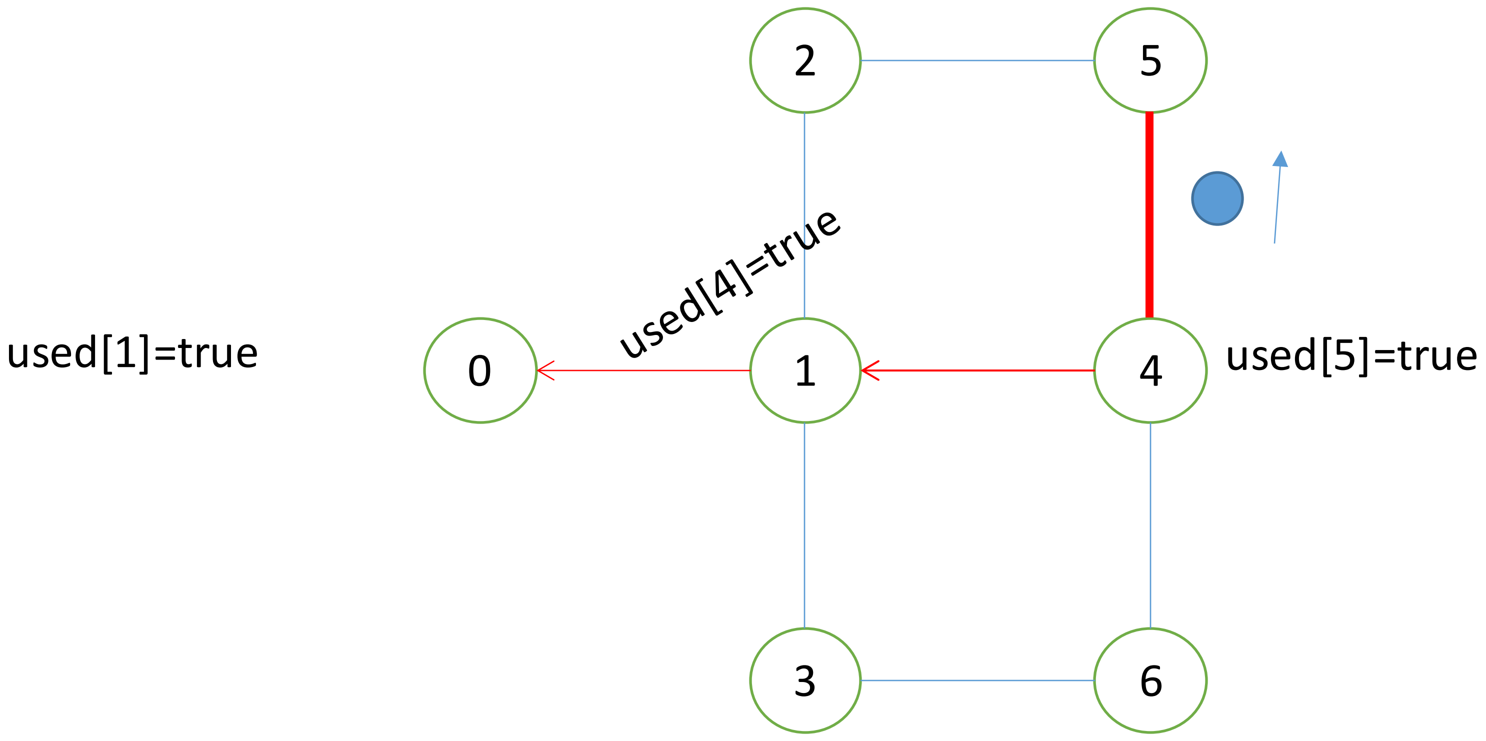
Tarry’s Spanning Tree



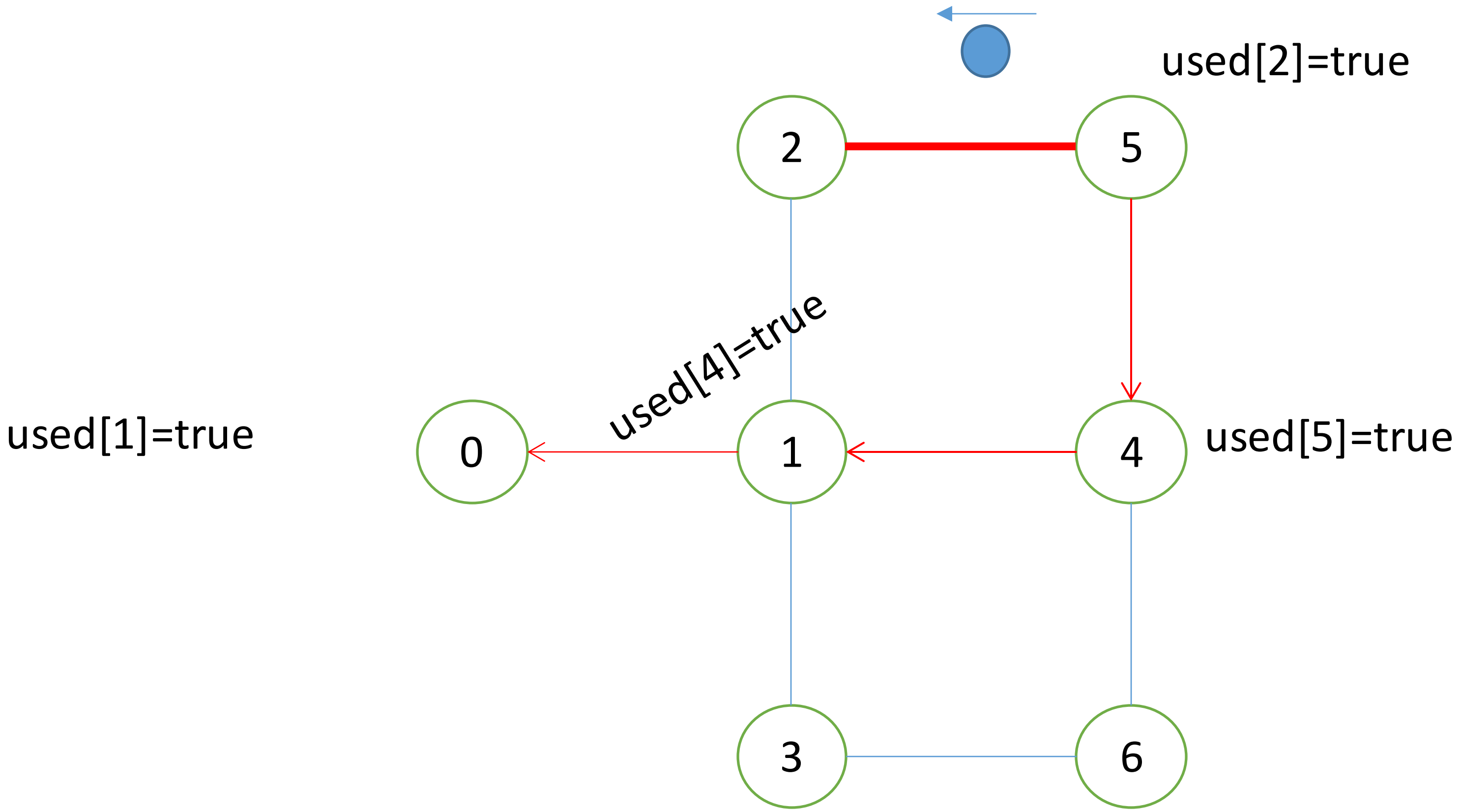
Tarry's Spanning Tree



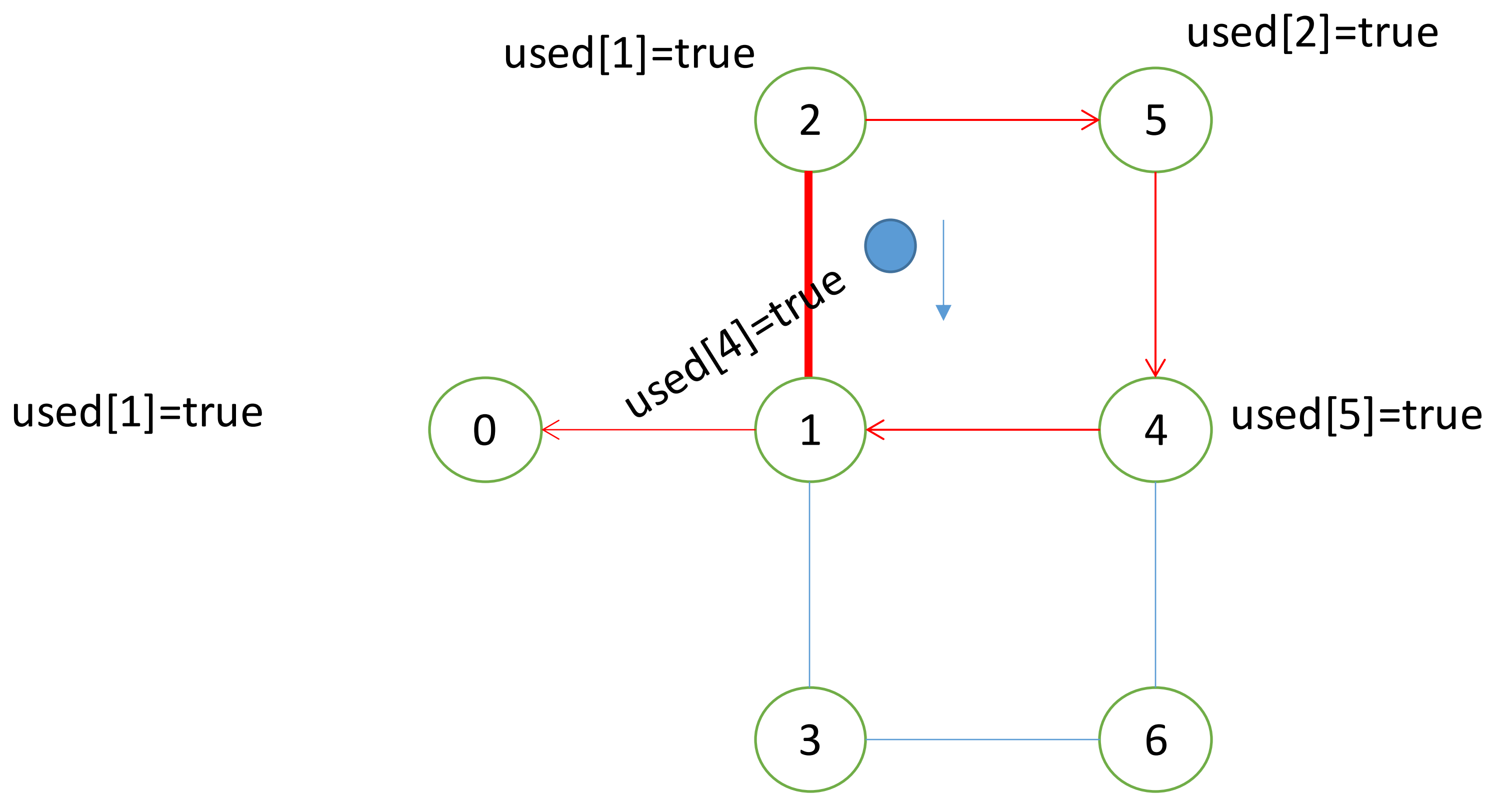
Tarry’s Spanning Tree



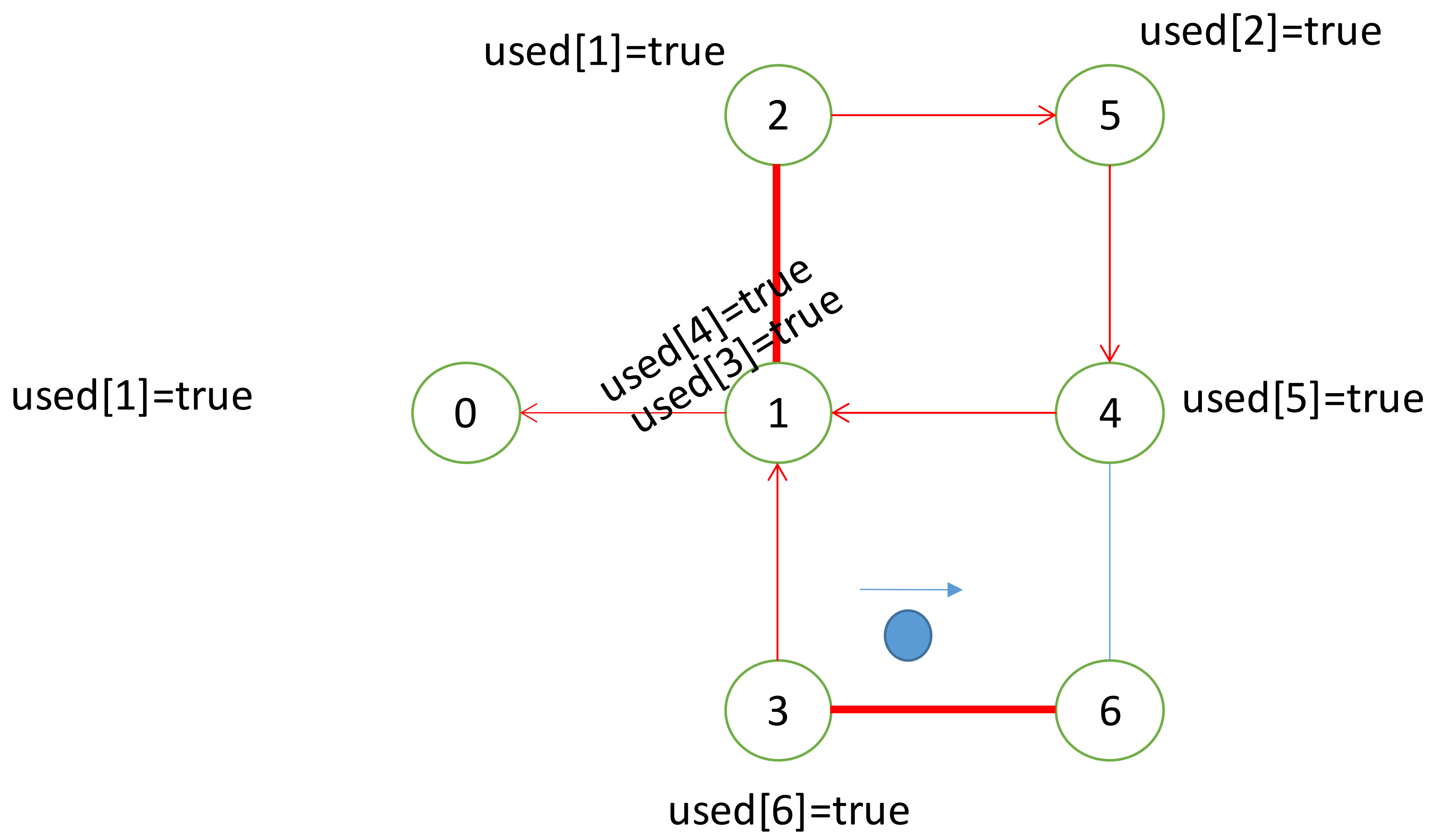
Tarry’s Spanning Tree



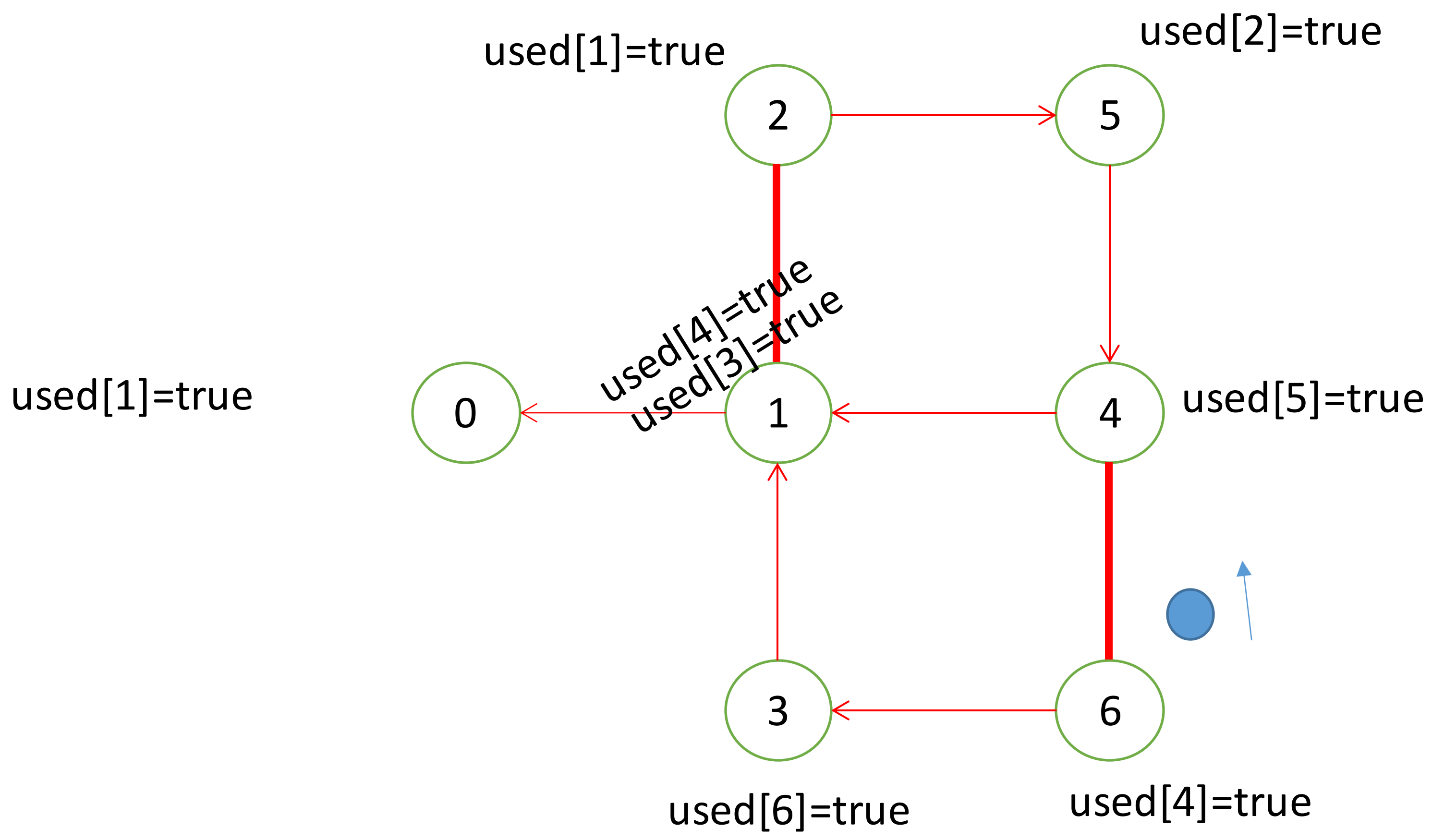
Tarry’s Spanning Tree



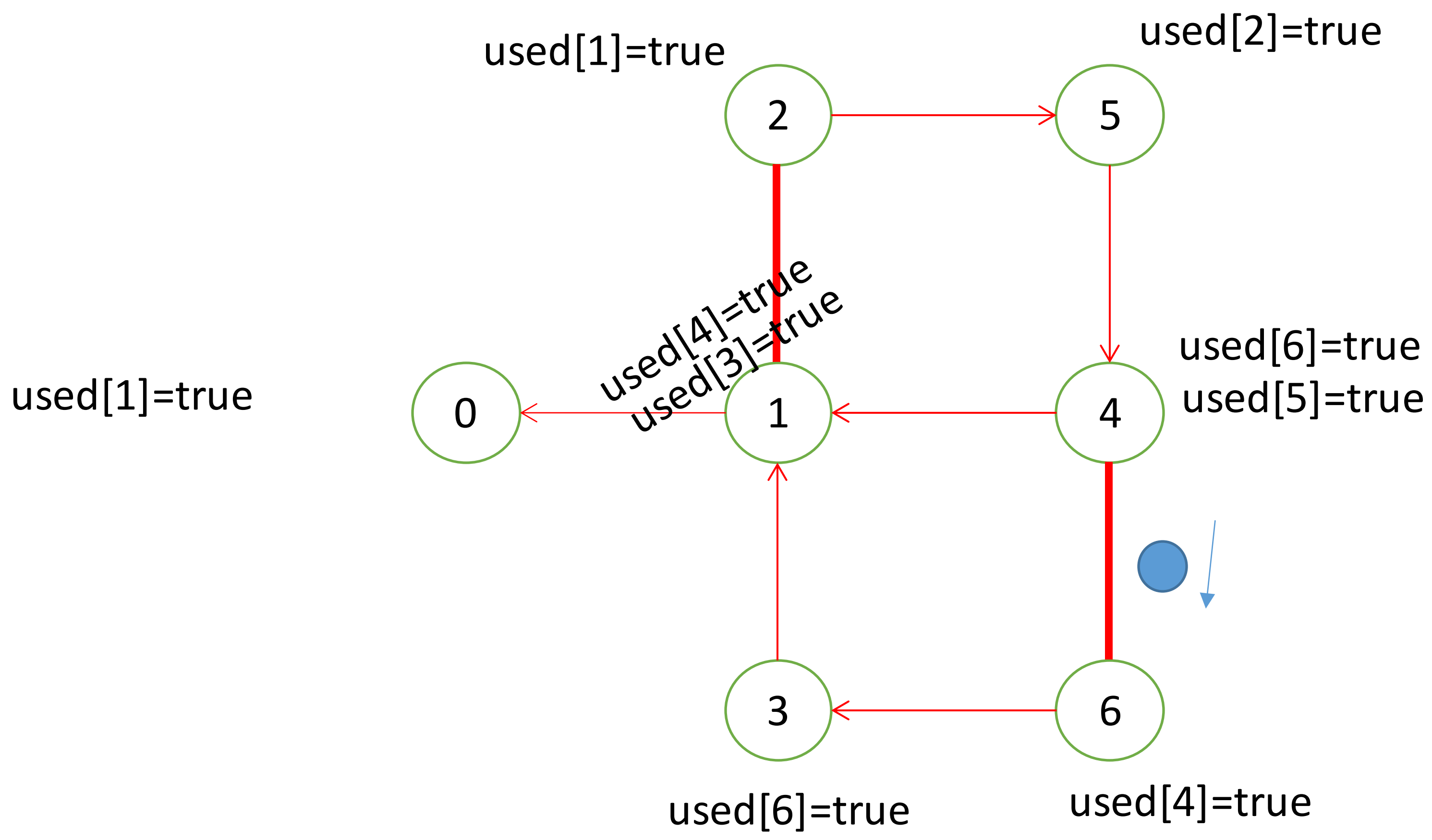
Tarry's Spanning Tree



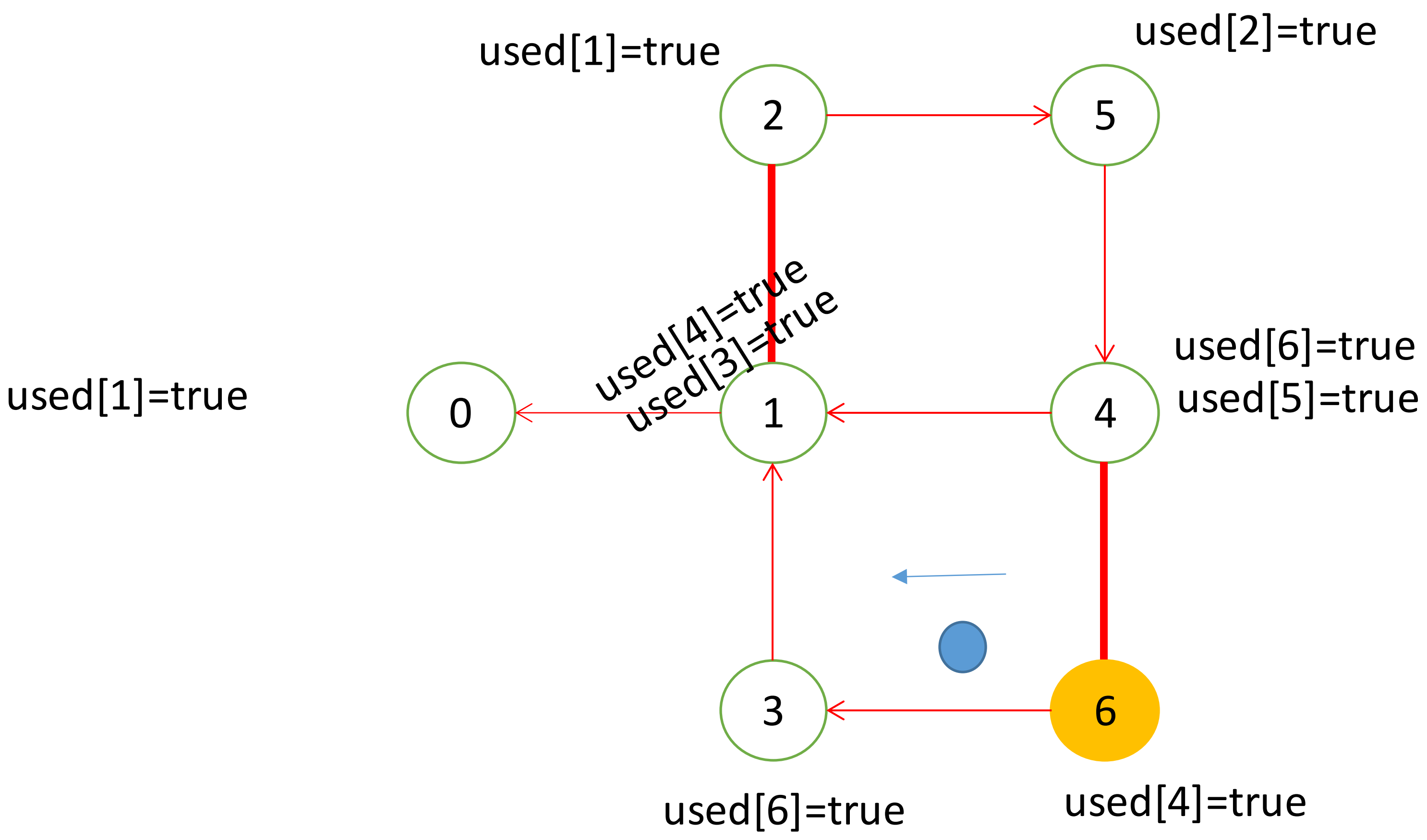
Tarry's Spanning Tree



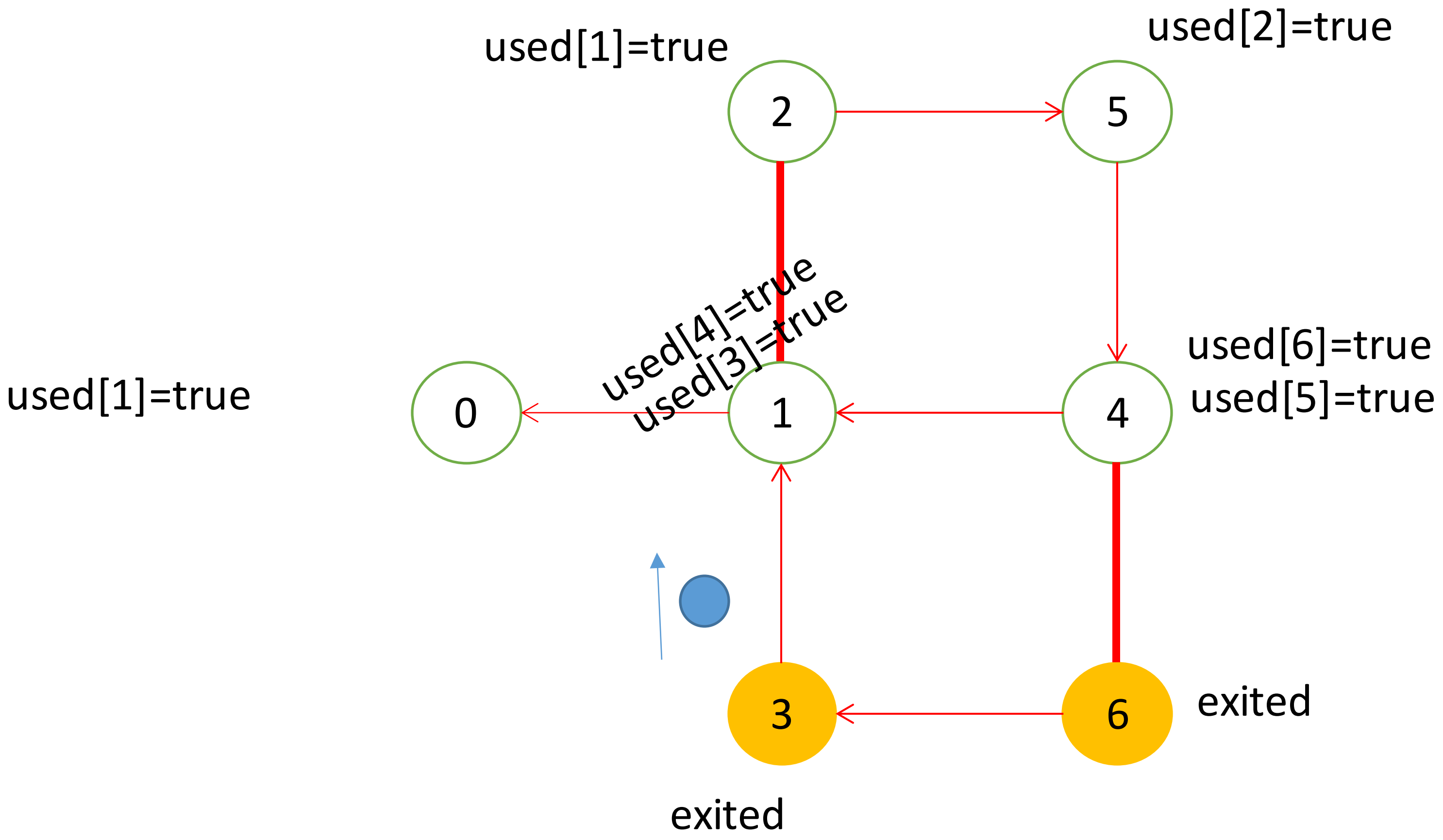
Tarry’s Spanning Tree



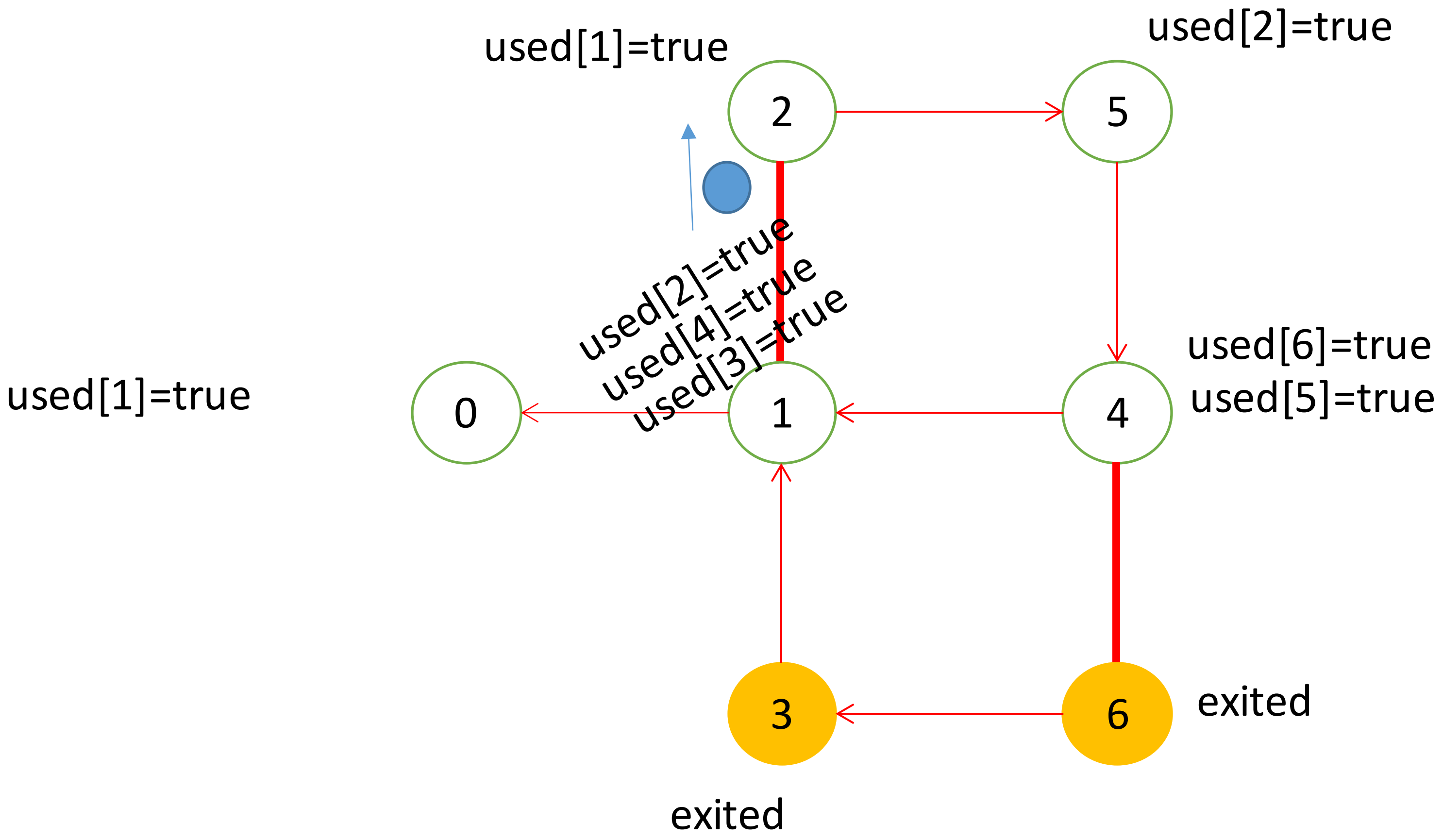
Tarry’s Spanning Tree



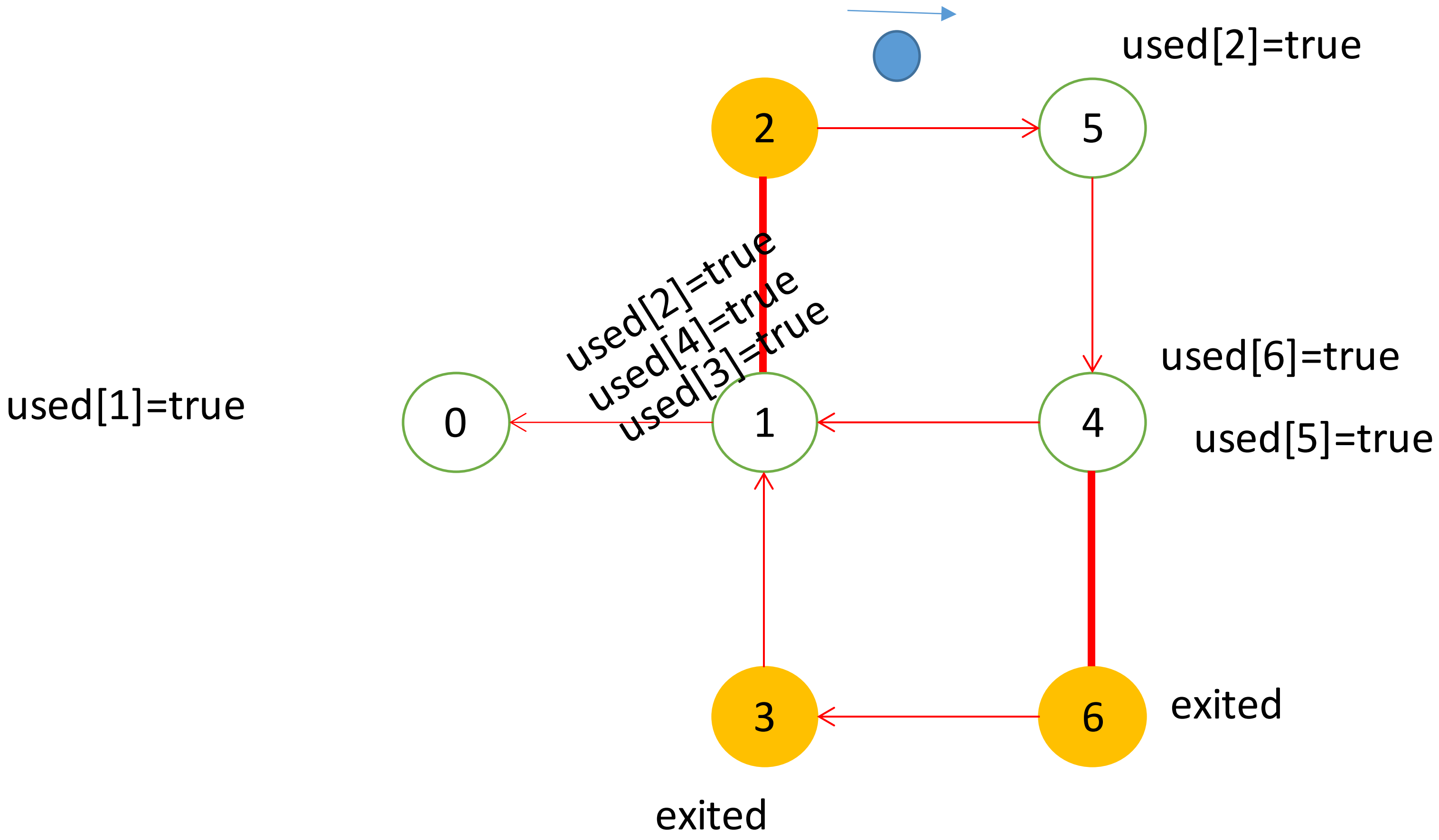
Tarry's Spanning Tree



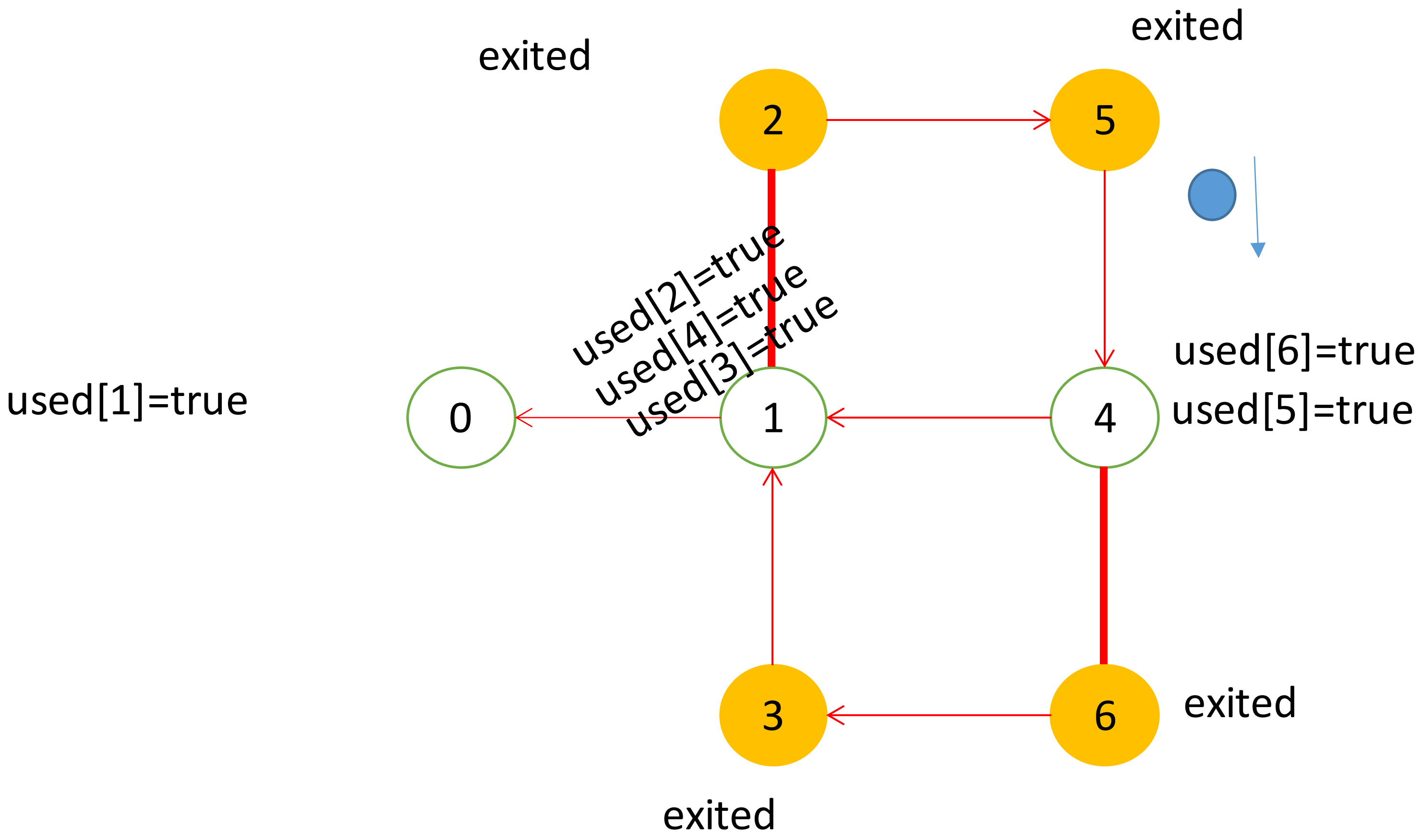
Tarry’s Spanning Tree



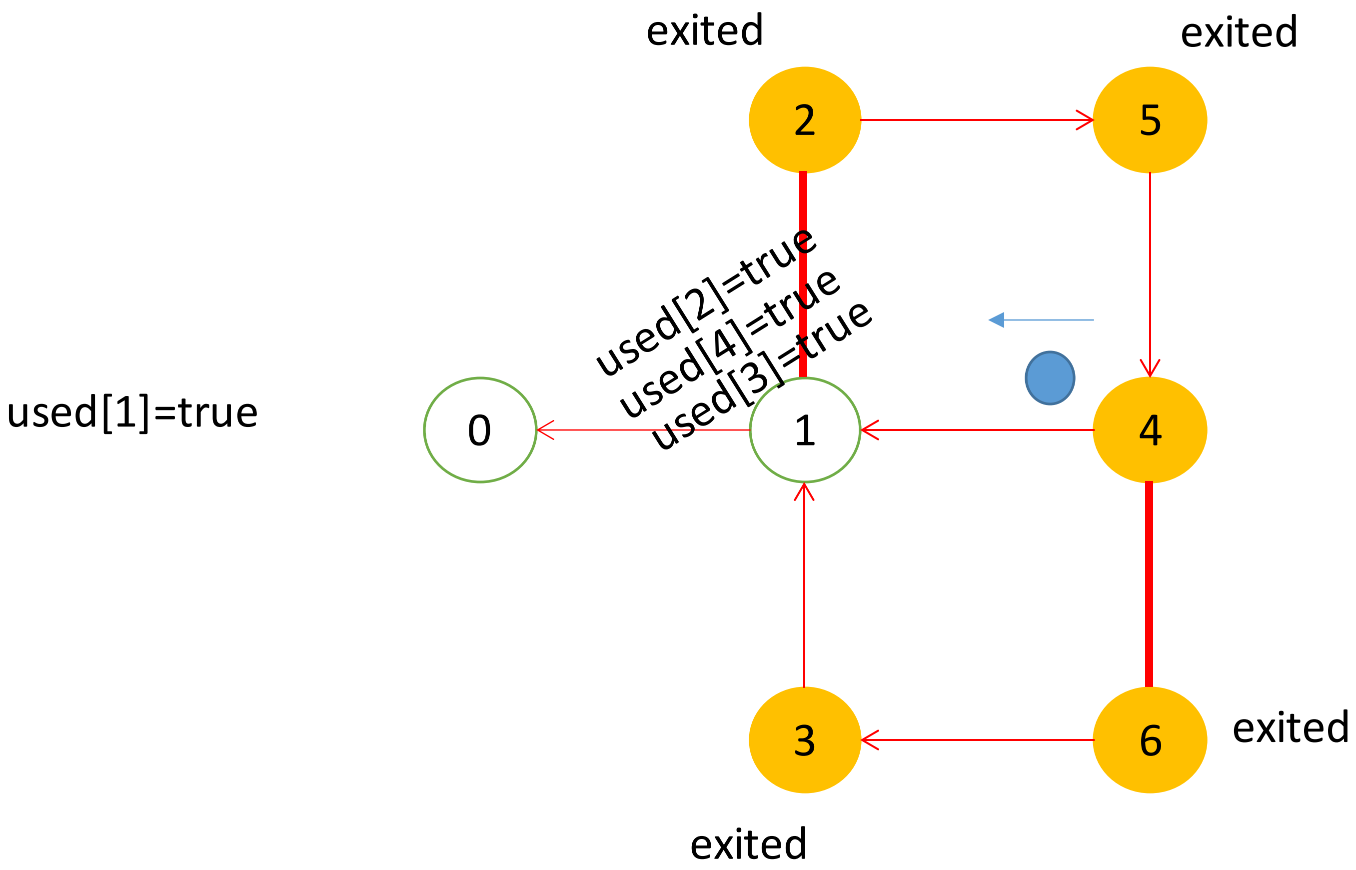
Tarry’s Spanning Tree



Tarry’s Spanning Tree

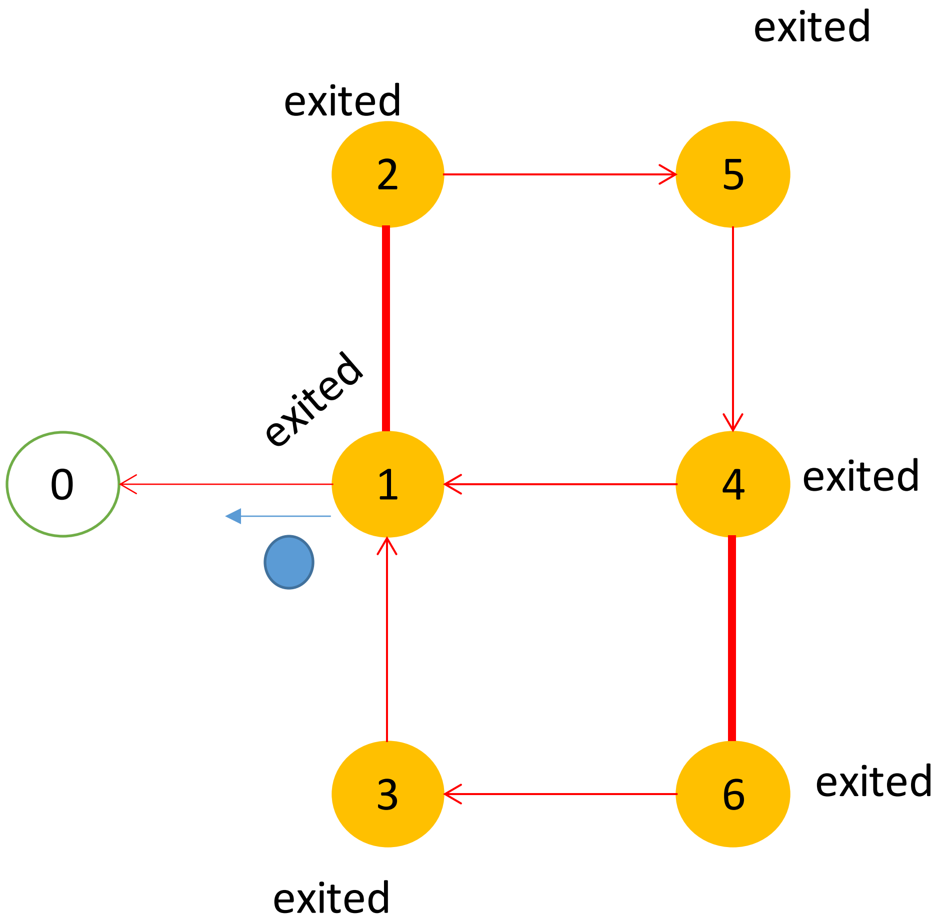


Tarry's Spanning Tree

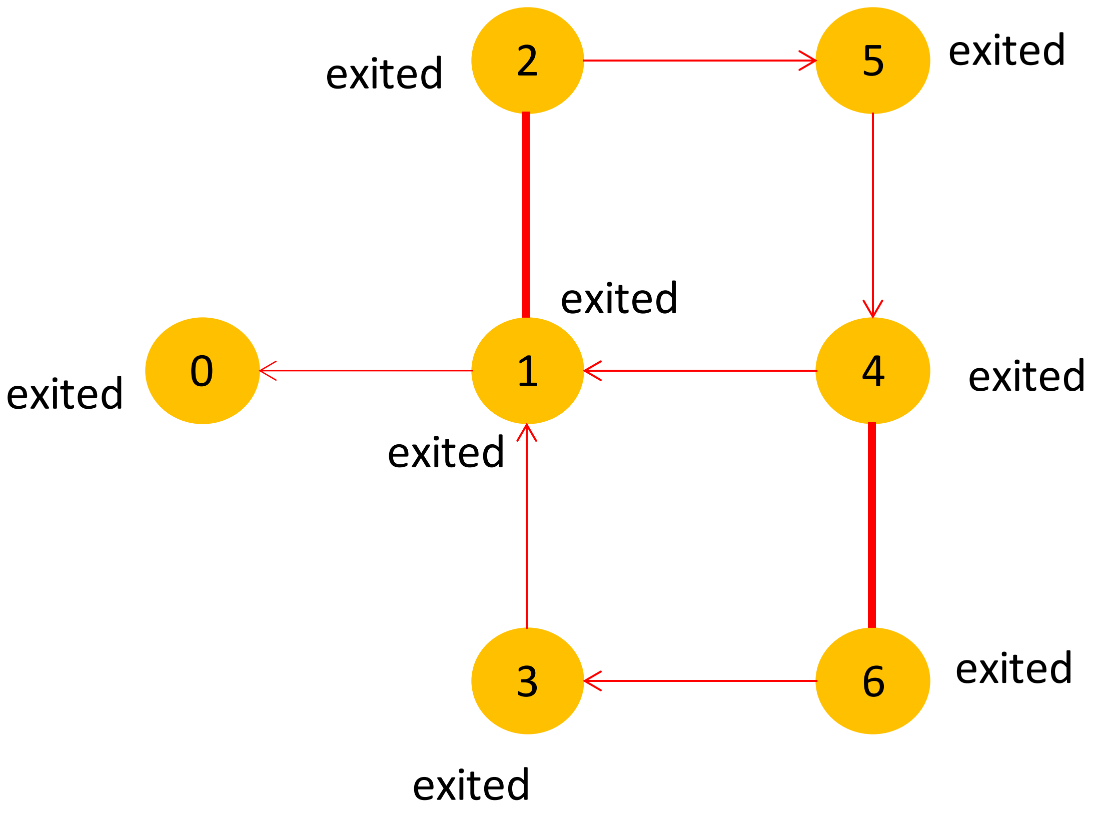


Tarry's Spanning Tree

used[1]=true



Tarry's Spanning Tree



Tarry's Spanning Tree Analysis

- Theorem 4.4 The time complexity of Tarry_ST is $\Theta(m)$, and its message complexity is also $\Theta(m)$.

- Proof Each edge is used to deliver a message exactly twice, once in each direction governed by the rules for a total of 2m times. Since there is a single point of activity at any time, there will be 2m steps, and hence Time(Tarry_ ST) = $\Theta(m)$ and Msg(Tarry_ ST) = $\Theta(m)$.

Convergecast and Broadcast

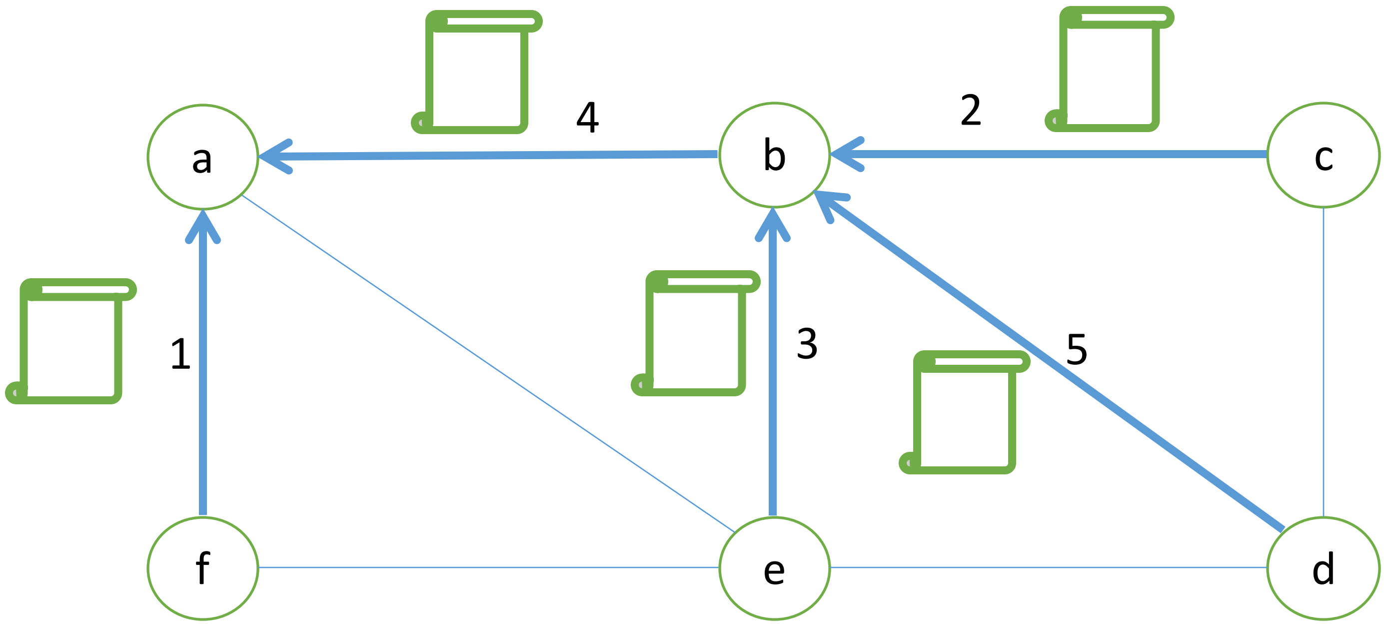
- In a computer network, it may be required to gather data from all nodes of the network to the root node of an already formed spanning tree.
- This operation, called the convergecast , is one of the key data transfer operations in a wireless sensor network.

- The key to the operation of this algorithm is that any non-leaf node should wait data from all of its children before uploading the combined/processed data to its parent.

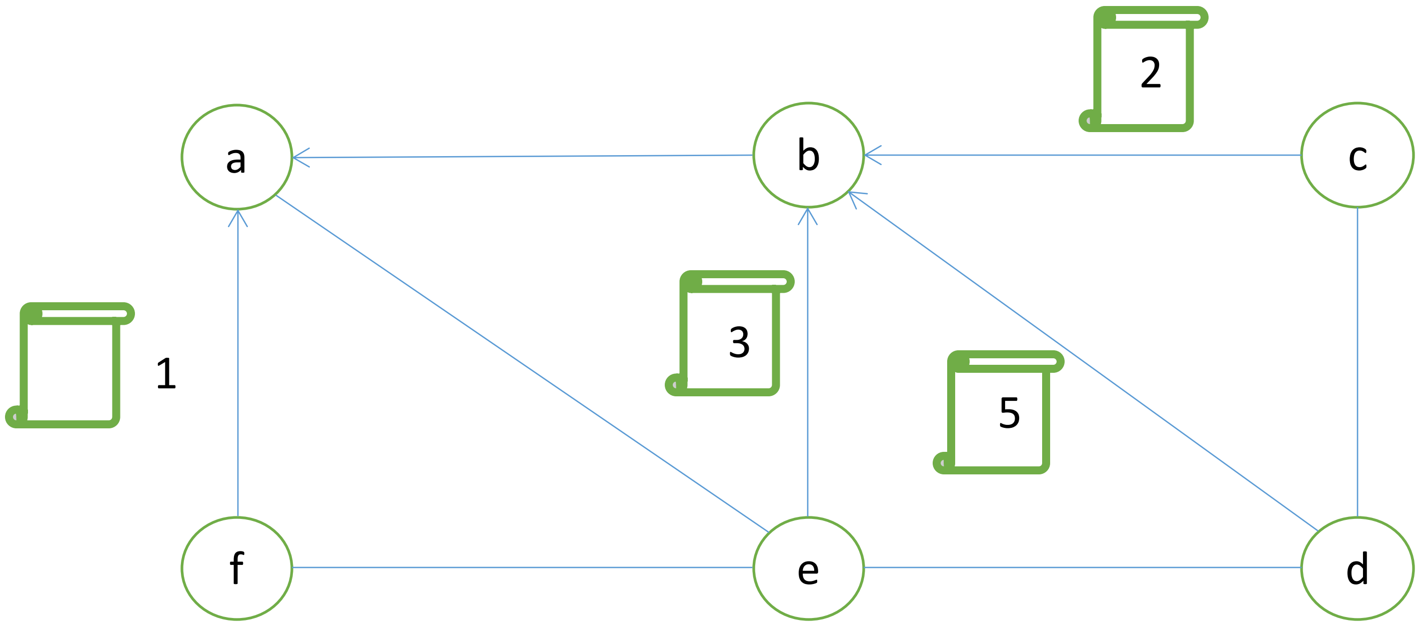
Algorithm 4.5 *Ccast_ST*

```
int parent
set of int childs; gathered ← ∅
message types convcast; msgs ← ∅

if childs = ∅ then                                         ▷ leaf nodes start convergecast
    send convcast to parent
else                                                       ▷ any intermediate node or root
    while childs ≠ gathered do            ▷ wait for convergecast messages from all children
        receive convcast(j)
        gathered ← gathered ∪ {j}
        msgs ← msgs ∪ convcast(j)
    end while
end if
if i ≠ root then
    combine msgs into convcast
    send convcast to parent
end if
```

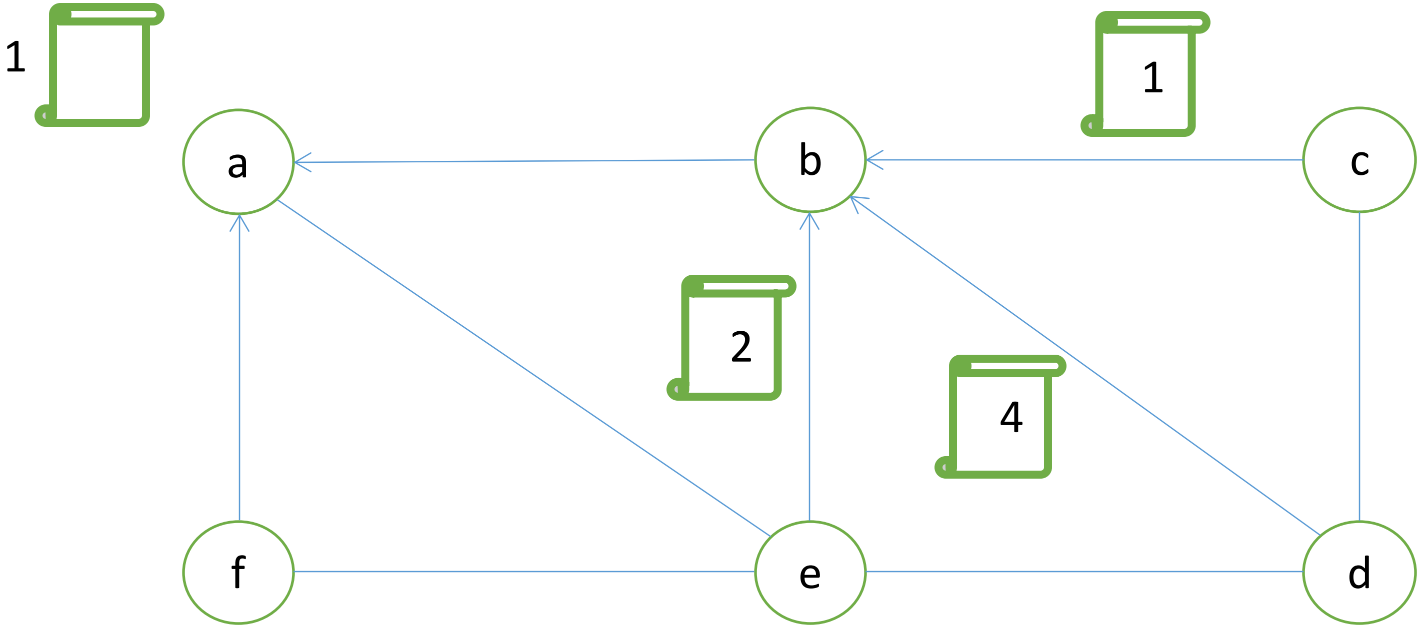
Convergecast Example Scenario



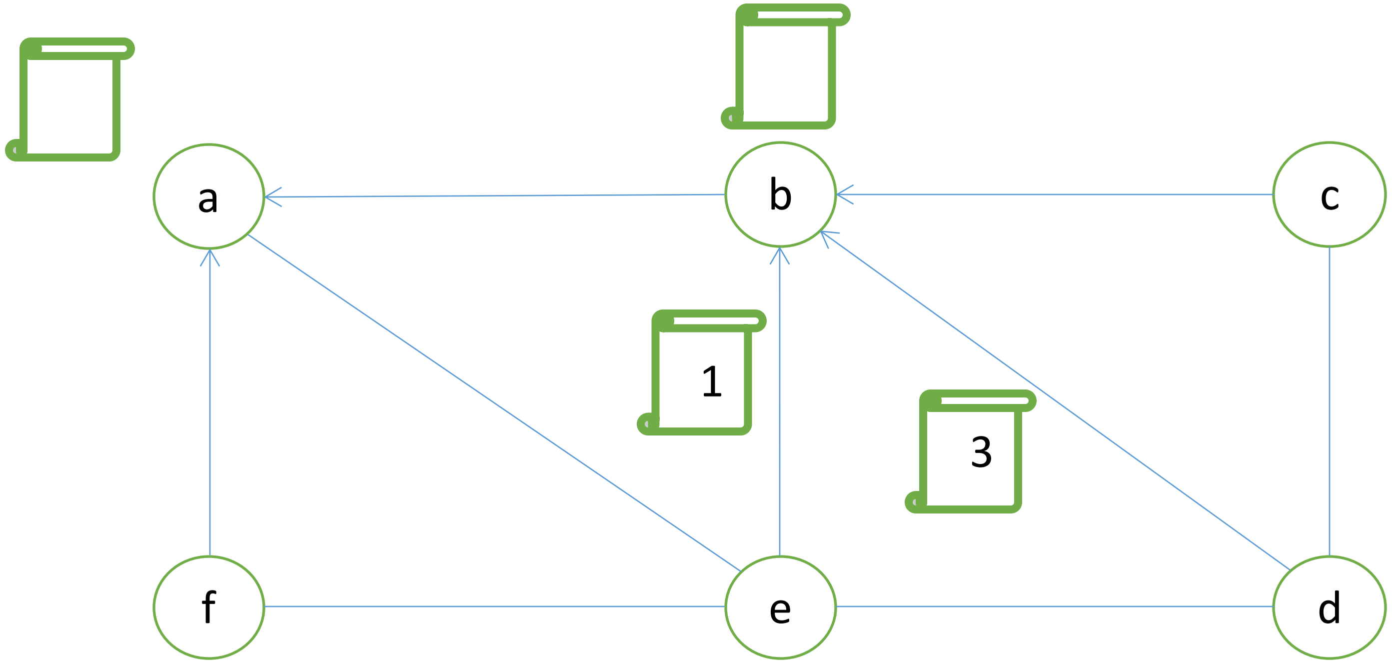
Convergecast Example Scenario



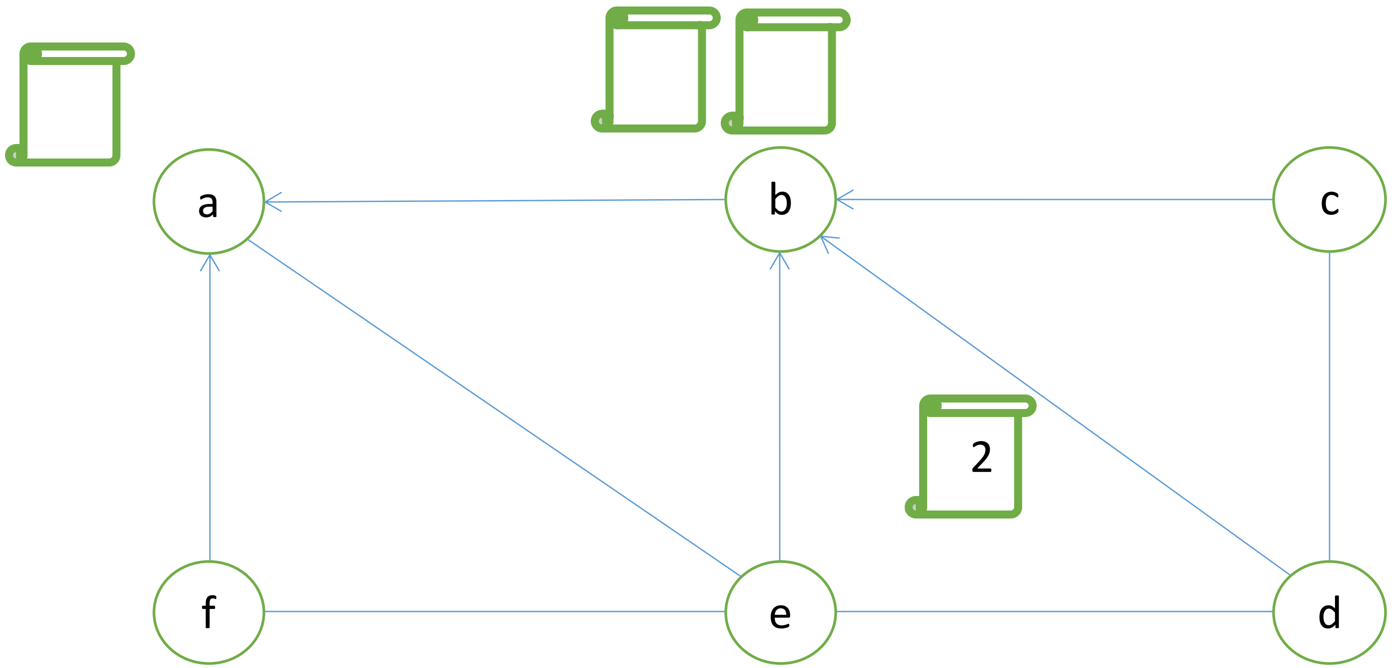
Convergecast Example Scenario



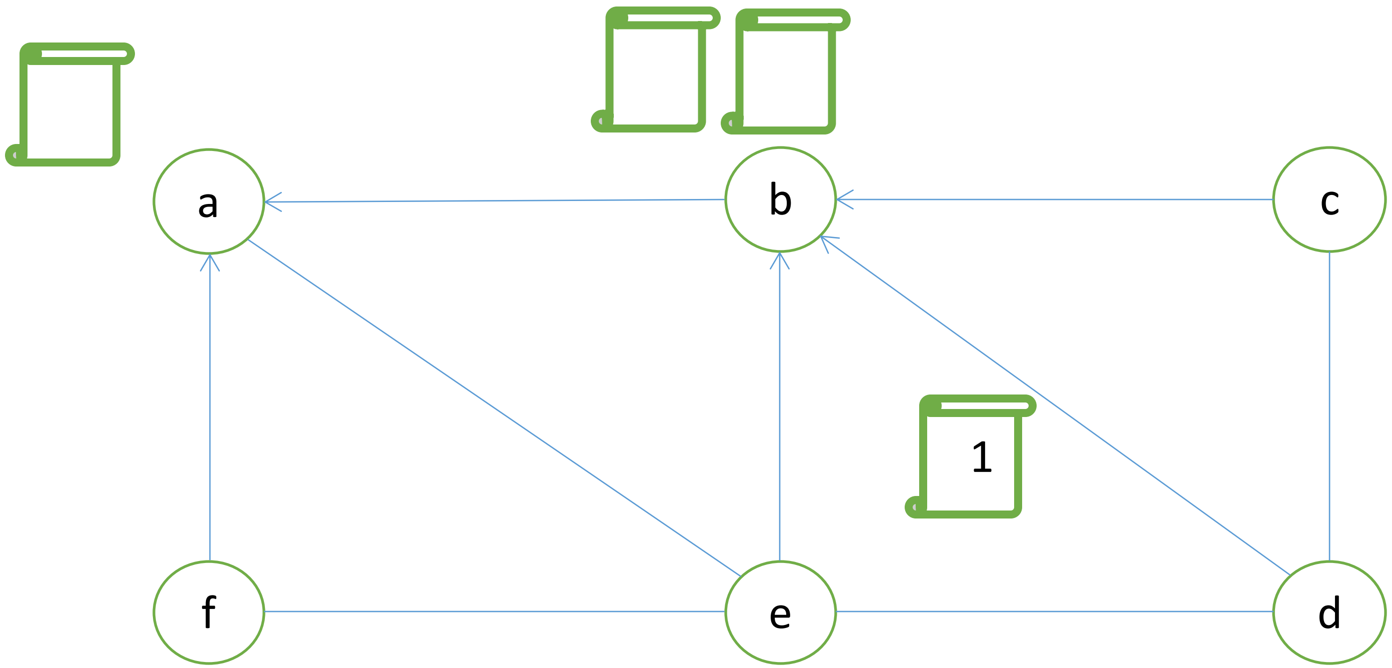
Convergecast Example Scenario



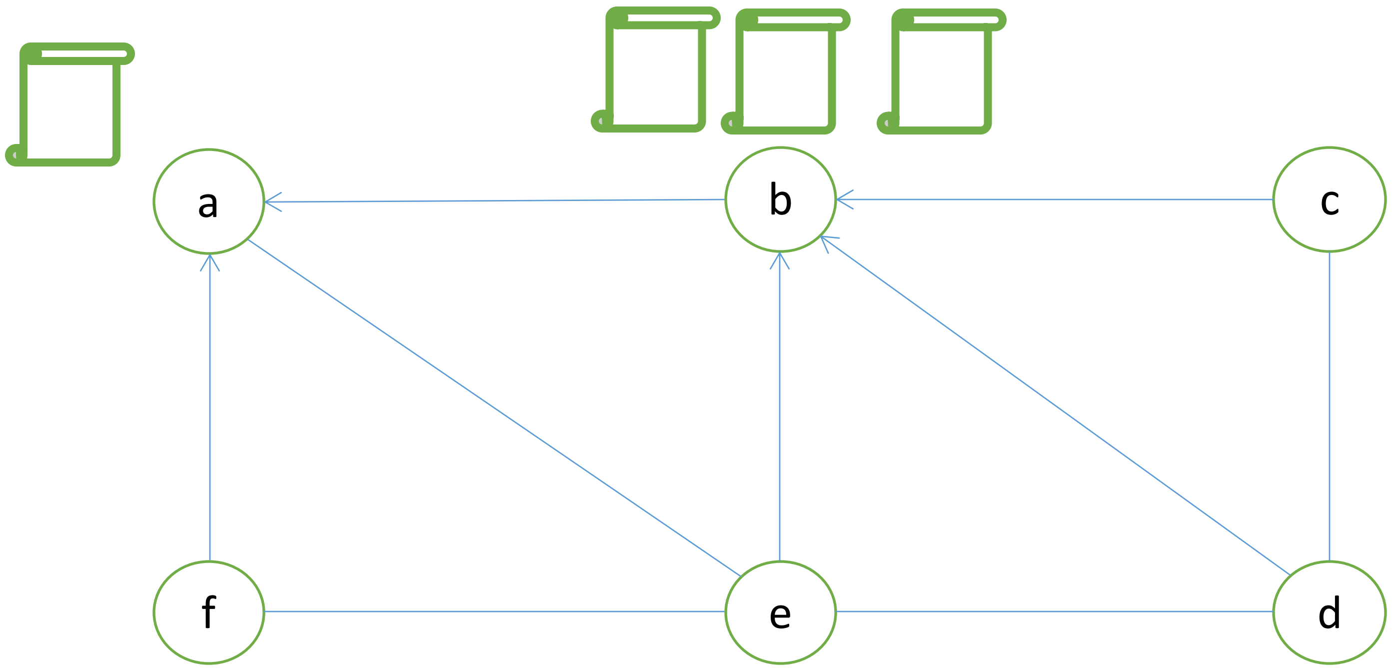
Convergecast Example Scenario



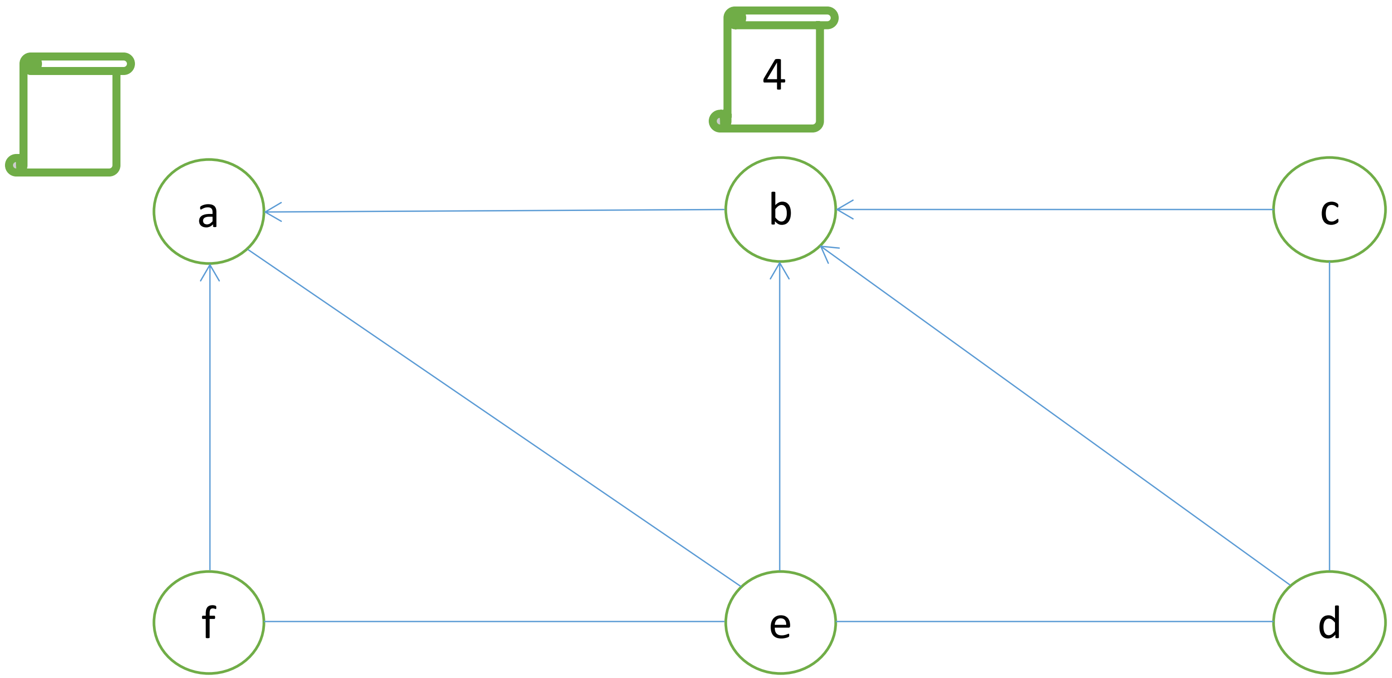
Convergecast Example Scenario



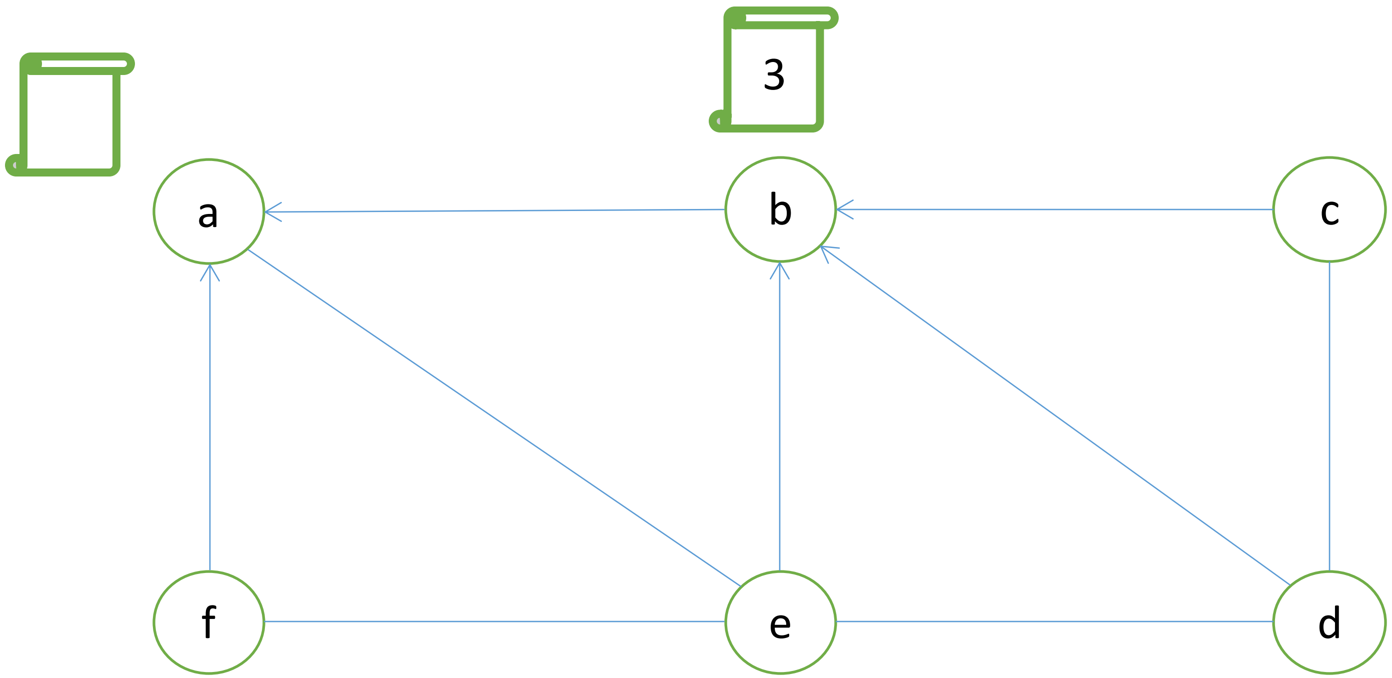
Convergecast Example Scenario



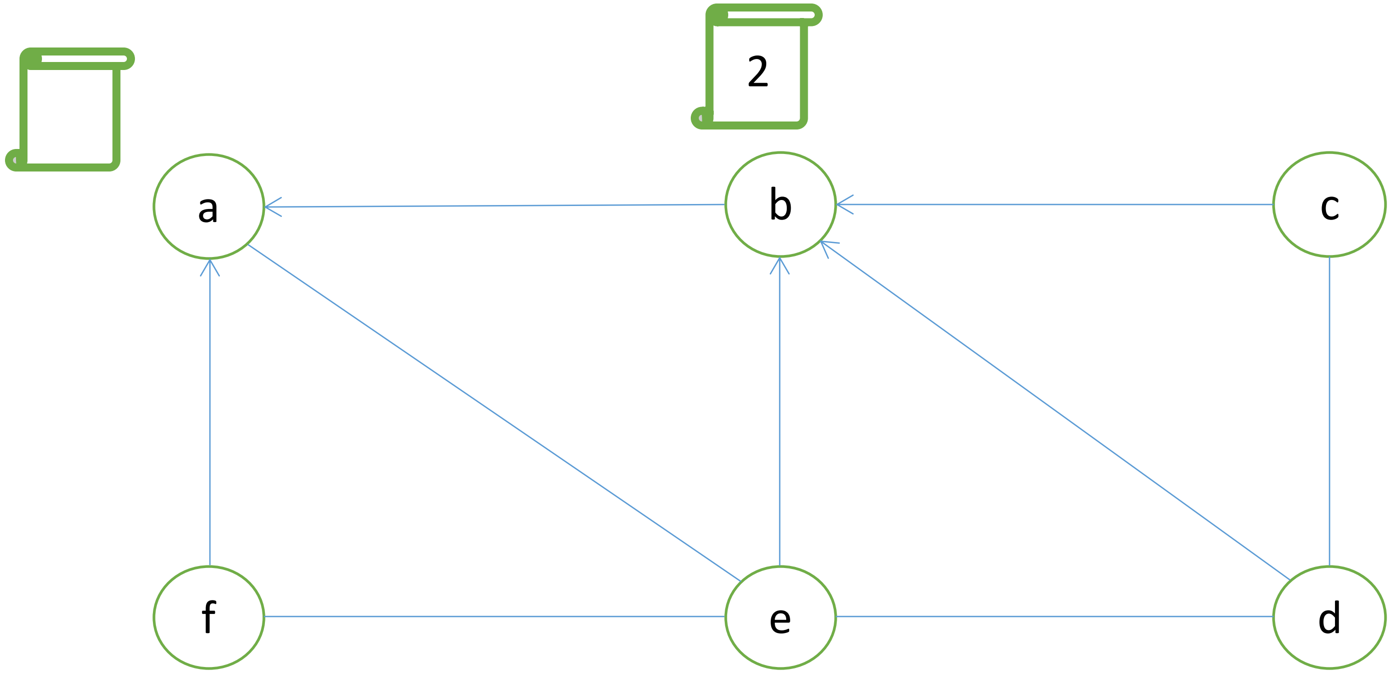
Convergecast Example Scenario



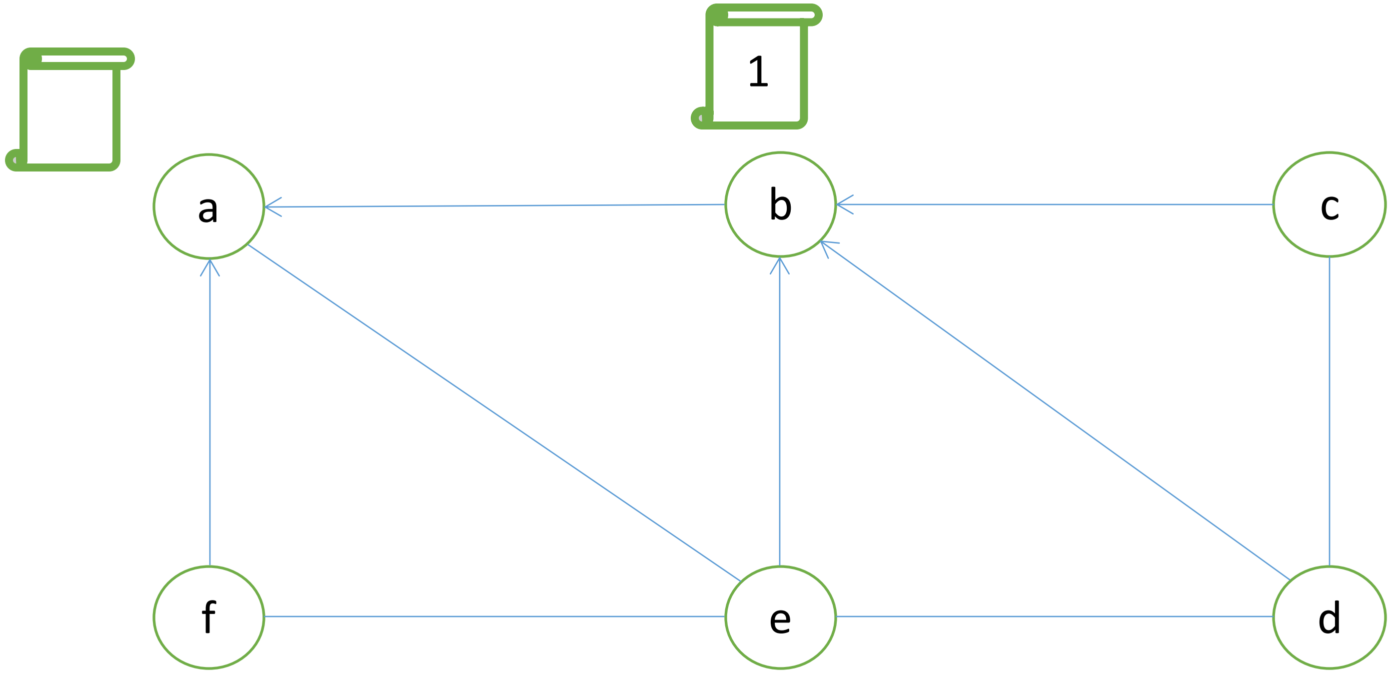
Convergecast Example Scenario



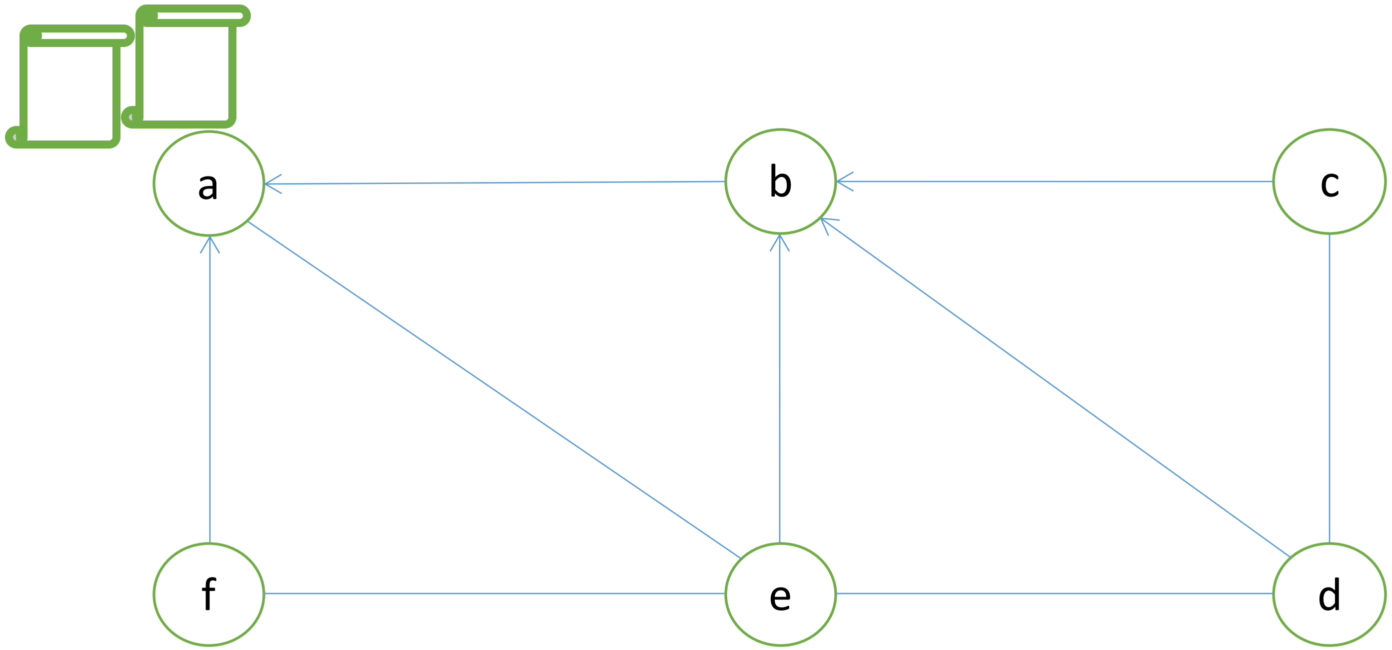
Convergecast Example Scenario



Convergecast Example Scenario



Convergecast Example Scenario



Broadcast

Algorithm 4.6 *Bcast_ST*

set of int $childs \leftarrow \emptyset$, $acked \leftarrow \emptyset$
message types $bcast$, ack
if $i = root$ **then**
    send $bcast$ to $childs$
end if
while $acked \neq childs$ **do** ▷ collect *acks* from *childs*
    receive $msg(j)$
    case $msg(j).type$ **of**
        $\underline{bcast}$: **if** $childs \neq \emptyset$ **then**
                send $bcast$ to $childs$
            else send ack to $parent$ ▷ start convergecast
        $\underline{ack}$: $acked \leftarrow acked \cup \{j\}$
end while
if $i \neq root$ **then**
    send ack to $parent$
end if

Broadcast and CCast Analysis

- Theorem 4.5. The message complexity of both Ccast_ST and ordinary broadcast algorithm Bcast_ST is $\Theta(n)$. The time complexity of both algorithms is $\Theta(depth(T))$, which would be at most $n - 1$.

- Proof In both algorithms, each edge of T is used to deliver a message once, and since the total number of edges of an n node tree is $n - 1$, there will be total of $n - 1$ messages. For the Bcast_ ST algorithm described, which provides convergecast operation, there will be a further $n - 1$ ack messages convergecast to the root, resulting in a total of $2n - 2$ messages. Time for broadcast or convergecast procedures, assuming that messages are transferred concurrently at each level, are the depth of the tree T , which would be at most $n-1$.