04_02_Token_&_Traversal

1. Token Loss
    1.1. Background
    1.2. Token Loss in a Ring
2. Traversal Algorithms
    2.1. Target
    2.2. Spinning Tree
    2.3. Basic ideas
    2.4. Tarry's traversal algorithm
    2.5. Cheung's DFS algorithm
    2.6. Awerbuch's Depth-First Search Algorithm
    2.7. Cidon's Depth-First Search Algorithm
    Remark of 4 algorithms: About parent

1. Token Loss

1.1. Background

1. Some DAs (such as termination detection in a ring) require the use of a token
2. Such a token may be lost, then a new token has to be generated

1.2. Token Loss in a Ring

Idea:

use two tokens, t_0 and t_1, with one detecting the loss of the other

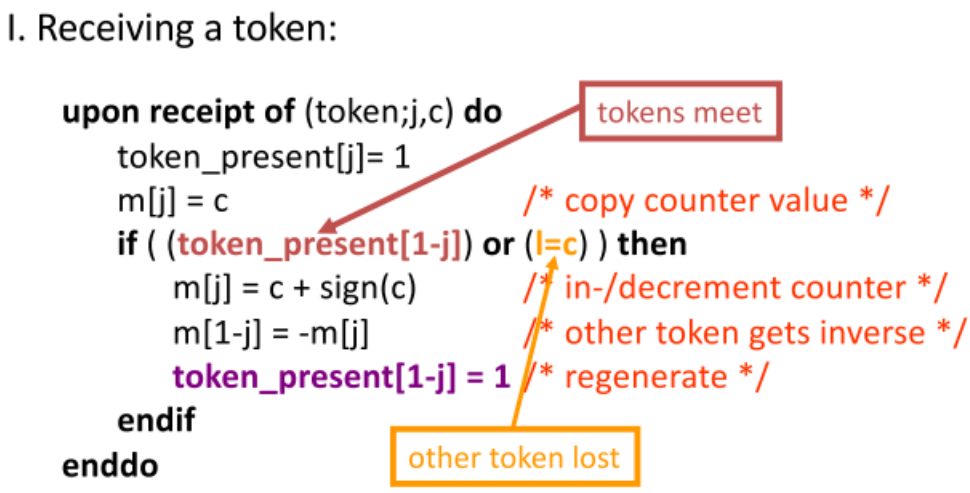
Preparation

1. Data structures in the **tokens**:
    a. their **id** (0 or 1) –
    b. a **counter c** which is equal to plus (in t_0) or minus (in t_1) the number of times the tokens have met
2. Data structures in **every node**:
    a. the **value l** of the counter of the token that visited it last
    b. an **array m[0,1]** with the values of the counters in the tokens

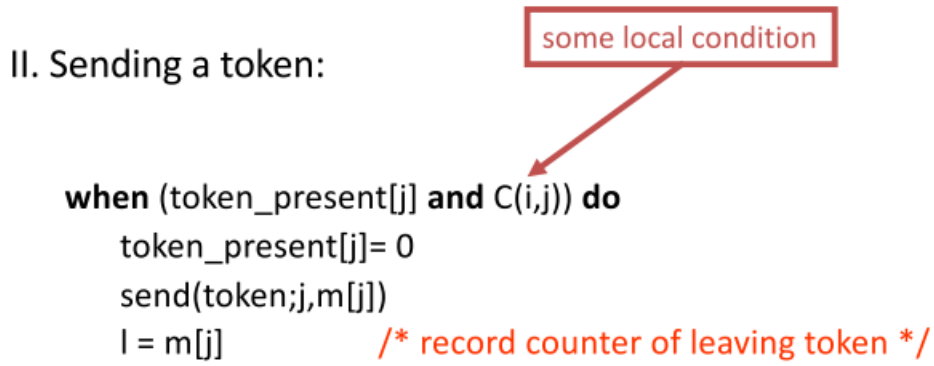
Procedure

1. when token t_0 passes through some node P for the second time in a row,
    - while in the meantime t_1 has not passed through P,
    - and in the meantime t_0 and t_1 have not met in another node,
    - then t_1 has been lost
2. Condition for detection of loss: l=c

Implementation



Token_Loss_Implementation_1



Token_Loss_Implementation_2

Understanding（Also the prove of Correctness）

Code

In receiving token, the **or** predict is actually a merge of two predict, because when the first part is true, the generation command has no effect on the system.

Algorithm

It actually consider the situation in the ring as a “chasing problem”:

when token t_0 passes through some node P for the second time in a row,

1. while in the meantime t_1 has not passed through P → means lost or has been bypassed by t_0

2. and in the meantime t_0 and t_1 have not met in another node, $\rightarrow$ means not be bypassed

And there is a clarification about why a token will not be regenerated twice:

1. when a token is regenerated in P, m[0] and m[1] in P are set to values these variables never had before in any node
2. so the tokens leave P with a new value for c
3. so, the value of l in any other node cannot be equal to such a value of c

2. Traversal Algorithms

2.1. Target

In order to disseminate information sensibly in a connected network, we want to **generate a spinning tree** in an undirected network

2.2. Spinning Tree

A spinning tree is the sparsest type of sub-network that keeps the system connected

2.3. Basic ideas

1. a special message (a **token**) traverses the network, starting at some originator
2. at the end of the algorithm, the token **returns** at the originator
3. having every node designating the node from which it receives the TOKEN for the **first time** as its **parent** (final) (this parent is not similar to some parent latter), a spanning tree is create

2.4. Tarry's traversal algorithm

Assumption

1. This algorithm is used in a **general undirected network**

Preparation

1. Every node:
    a. maintains **a set of its links**, excluding its parent, along which it **has not yet sent the token**
2. Message Types: **TOKEN**, **TOKEN BACK**

Procedure

1. When a node receives the token,
    a. if the set is not empty, it send the token on **random one** of the links in the set, and updates the set
    b. is empty, sent the token **back** to its parent
2. The parent link of each nodes construct a spinning tree

Understanding

The set of the parents link can connect the graph together without a circle:

1. That is because each node has only one parent(final), and the originator has no parent.
2. so, Link number=parent number=node number-1

Together with the proof of the correctness part, it make sense!

Correctness

Correctness is to **ensure every node has received the Token.**

1. The token ends in the originator
2. the TOKEN will be sent twice on every edge in the network, once in either direction (means no circle)

- when the token arrives in a node which is not the originator, #arrivals = #departures – 1, so it always has an option to forward the token
- See the ppt

Complexity

the message and time complexities are both equal to 2 ·|E| (|E| number of edges)

2.5. Cheung's DFS algorithm

Prepare

Message Types: TOKEN, TOKEN BACK

Procedure

1. when a node receives the token **for the first time**,
    a. it forwards it on **an unused link**, if any
    b. otherwise, it sends the token **back to its parent** (parent link)
2. when a node has received the token **already previously**, it **sends the token back immediately**

Understanding

Very similar to the previous algorithm. Just with different way to choose link, this algorithm is the Depth-First way

Complexity

the message and time complexities are both equal to 2 ·|E| (|E| number of edges)

2.6. Awerbuch's Depth-First Search Algorithm

Basic Idea

Reduce time complexity by **avoiding "useless" sends** of the Token to nodes that have already been "visited" by the Token

Prepare

Message Types: TOKEN, TOKEN BACK, VISITED, ACK

Procedure

- When a node receives a **TOKEN** (will be the first and only time):

    - sends a message **VISITIED** to all neighbours except: its parent + the node to which it will send the TOKEN.
- waiting for **ACK** of the VISITED
    - After Receiving all ACK, send **TOKEN** to a non-visited node through a unused link
- If no non-visited node or no unused link, send the token to its **parent**
- The originator will send the **VISITED** to all its neighbours when it is going to generate the first TOKEN

Understanding

With the procedure above:

1. a node will only receive a token only once
2. when a node receives the token back, it knows the sender is one of its children

Complexity

V for set of vertex, E for set of edges

Token: through all **edges of spinning tree**

- time complexity: 2(|V|-1)
- message complexity: 2(|V|-1)

VISITED & ACK： at most each edge has two VISITED + two ACK

- time complexity: 2 (|V|-1)
- message complexity: 4|E|

2.7. Cidon's Depth-First Search Algorithm

Main Idea

still use "visited" messages, but don't use acks anymore

Prepare

Message Types: TOKEN, TOKEN BACK, VISITED

Main Process

1. When a node receives a **TOKEN** (will be the first and only time):
    - sends a message **VISITIED** to all neighbours except: its parents + the node to which it will send the TOKEN (assume not visited).
    - send to the node that is chosen previously
    - If no non-visited node or no unused link, send the token to its parent
2. If the node received a **VISITED** signal from the target it chosen previously, then choose another that it knows has not been visited
3. If a node received a second **TOKEN**, just **ignore** it
4. The originator will send the VISITED to all its neighbours when it is going to generate the first TOKEN

Understanding

By using the mistake-revise mechanism, all nodes will still receive at least one token, so they can ensure a parent, and the parent links will become a spinning tree.

Complexity

- Message complexity: 3 |E|= 2 VISITIED + 1 TOKEN (at most)
- Time complexity: we assume the system is synchronous to evaluate the time complexity: 2(|V|-1)

Remark of 4 algorithms: About parent

In Tarry and Cheung

when a node receives the TOKEN back from a node to which it has previously sent it, it does not always mean that it is the parent of the latter node

- Tarry's: all parent link will receives a TOKEN Back, but they are not the final parent
- Cheung: all parent link will receives a TOKEN Back, but they are not the final parent

So, if it is required that not only the children know their parents but also the parents know their children (the first one is used to send TOKEN Back, the second one is used to ensure the tree)

messages returning the TOKEN to its sender have to include an indication whether the sender is the parent of the node that returns the TOKEN or not.

In Awerbuch and Cidon

ll nodes to which a node (correctly) sends the TOKEN automatically are the latter's children