

TRAVERSAL

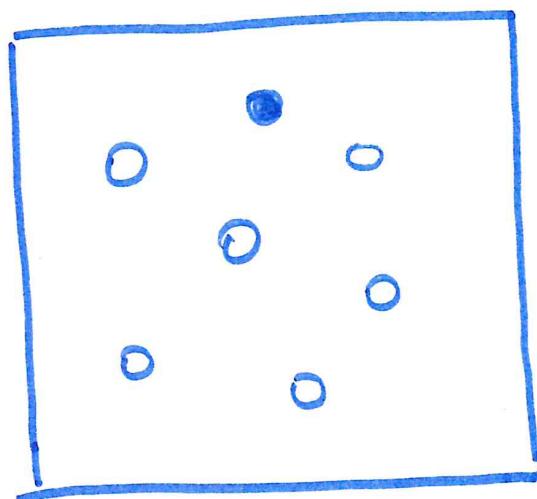
Problema: ogni entità della rete
deve essere visitata

ma

sequenzialmente, cioè una
dopo l'altra

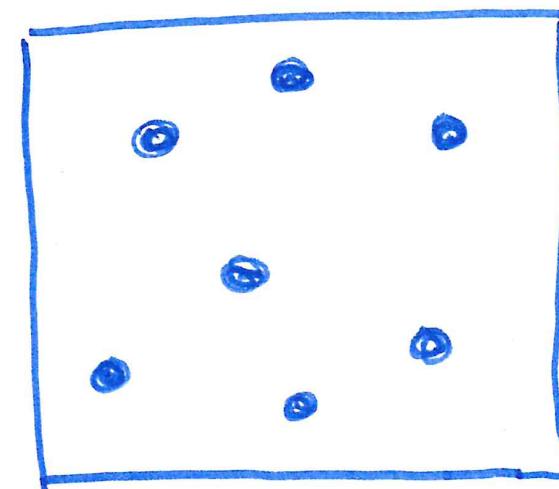
Applicazioni: per la gestione delle
risorse condivise

conf. iniziale



Tutte unvisited
(eccetto una)

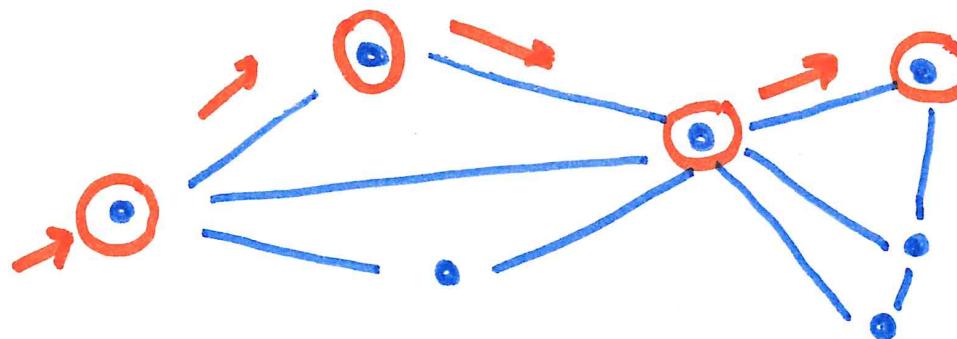
conf. finale



Tutte visited
ma una alla volta

Protocollo Depth-first Traversal

Visita in profondità di un grafo:



"Si scende sempre verso il vicino non ancora visitato"

IDEA: usiamo un messaggio particolare

TOKEN T

quando un nodo riceve T
diventa VISITATO

ma ad ogni istante di tempo
"viaggia al più un solo T"

Passi principali :

1) Un nodo che riceve T per la prima volta:

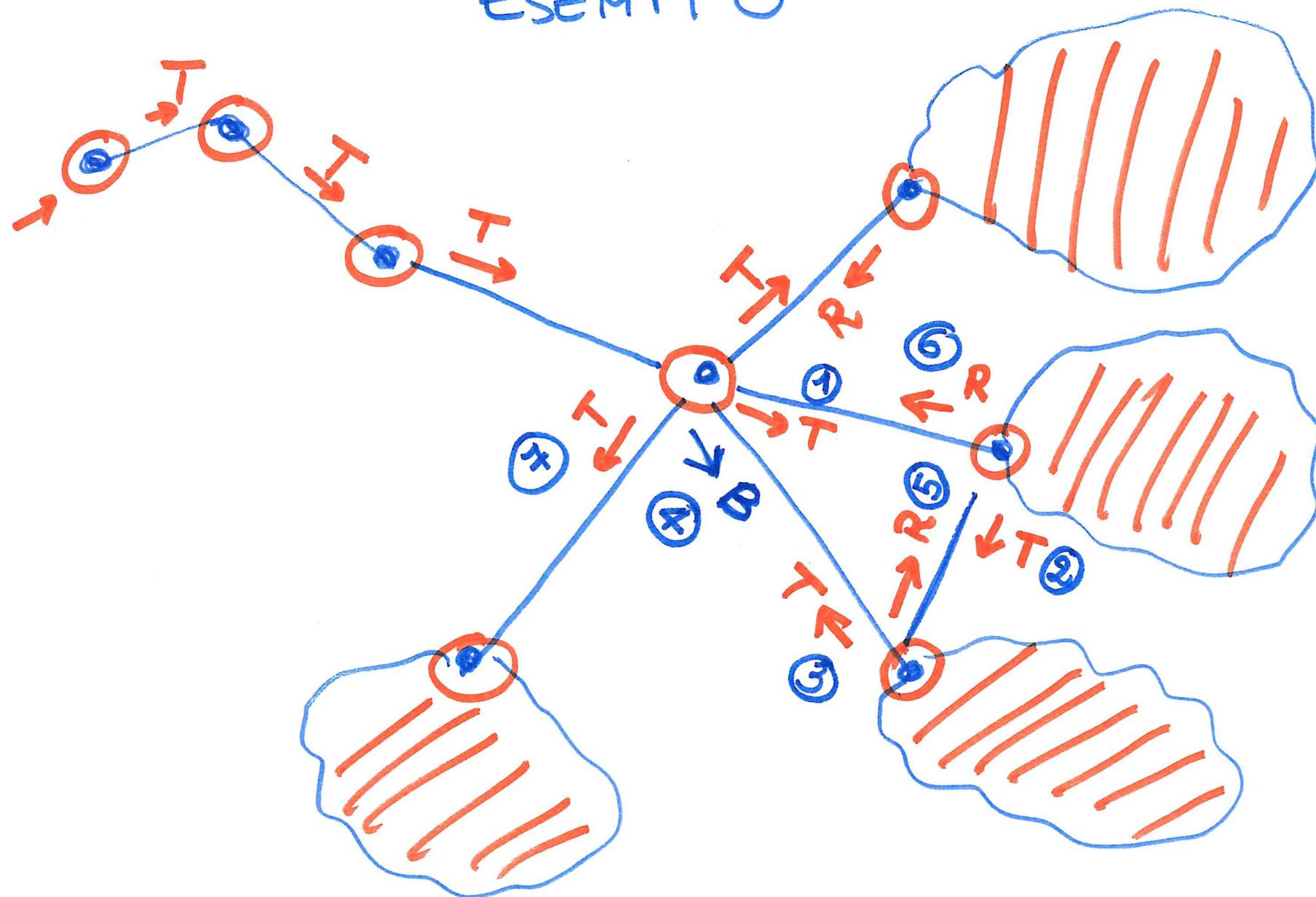
- ricorda il sender
- fa una lista dei suoi vicini non visitati
- invia T ad uno di essi
- aspetta un messaggio da quest'ultima entità :
RETURN
BACK-EDGE

2) Il vicino che riceve T:

- se è il primo T che riceve ripete 1)
- altrimenti (è già stato visitato)
e specifica un BACK-EDGE

3) Solo dopo aver finito la lista
dei vicini non visitati deve inviare
un RETURN al sender

ESEMPIO



Riassumendo abbiamo Tre Tipi
di messaggi:

- T Token
- B Back-edge
- R Return

Protocollo DF- TRAVERSAL

$S = \{ \text{initiator}, \text{idle}, \text{visited}, \text{done} \}$

$S_{\text{INIT}} = \{ \text{initiator}, \text{idle} \}$

$S_{\text{TERM}} = \{ \text{done} \}$

Restrizioni = RI

Protocols

Initiator

Spontaneously

```
{   initiator = True ;  
    Unvisited = N( $\infty$ ) ;  
    VISIT ;  
}
```

IDLE

Receiving (T)

```
{   entry = sender;  
    unvisited = N( $\alpha$ ) - sender;  
    initiator = false;  
    VISIT;  
}
```

Procedure VISIT

```
{   if unvisited ≠ φ Then
    {   next ← unvisited ;
        send (T) To next;
        become visited;
    }
else
{   if (initiator = false) Then
    send (Return) to entry ;
    become done;
}
}
```

Visited

Receiving (Return)

{ VISIT; }

Receiving (Back-edge)

{ VISIT; }

Receiving (T)

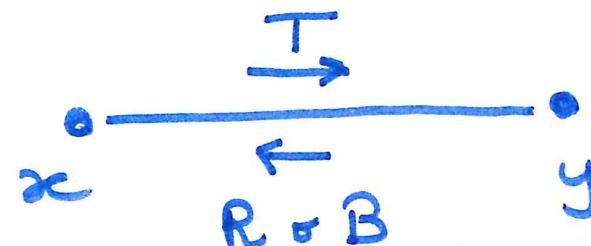
{ Unvisited = Unvisited - sender;
send (Back-edge) To sender;

}

Complessità Depth - First Traversal

OSSERVAZIONE:

-)



→ Traversal è sequenziale



$$T[\text{DF-Traversal}] = M[\text{DF-Traversal}]$$

||

$\leq m$

Lower Bound al problema TRAVERSAL

- $M \lceil \text{DFS-Traversal} \rceil \geq m *$
- $T \lceil \text{BFS-Traversal} \rceil \geq h-1 **$
- ** ogni nodo viene visitato in sequenza
- * Vale il Teo per BCAST

Nel caso di un grafo
CONNESSO

$$n-1 \leq m \leq \frac{n \cdot (n-1)}{2}$$

"

$$\frac{n^2 - n}{2} = O(n^2)$$



DF-Traversal è ottimo

- per la quantità di messaggi
- ma nel caso peggiore non per il Tempo

$$n^2 \longleftrightarrow n$$

Osservazione:

Il problema per il costo del tempo è che:
ad ogni istante di tempo viaggia
un messaggio solo



Soluzione: Introduciamo concorrenza
magari aggiungeremo una q.tà opportuna
di messaggi ($O(m)$)

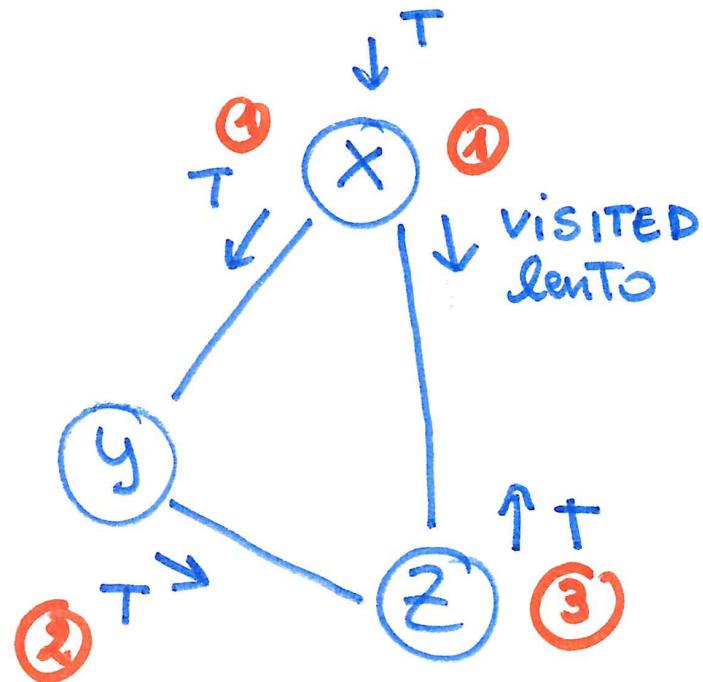
Domanda:

Possiamo evitare di inviare T
su un link back-edge?

Idea: Un nodo non visitato che riceve
 T comunica l'evento ai vicini
mandandolo un messaggio VISITED
(chiamamente in contemp. con un T)

\Rightarrow I vicini che ricevono VISITED aggiornano
la propria lista Unvisited

Esempio :



- z riceve visited: capisce l'errore e spedisce T ad un'altra entità oppure invia un Return e sarà questa entità che lo riceve a spedire un altro T
- z riceve T : capisce che z è visitato e lo rimuove dalla sua lista

Domanda:

abbiamo evitato di inviare T sui back-edge?

Risp: No!

Nuova complessità :

- Messaggi :

$$2^{n-2} \leftarrow T + \text{Return}$$

$$2^{m-(n-1)} \leftarrow \text{Visited}$$

$$2(m-(n-1)) \leftarrow \begin{array}{l} \text{errori} \\ \text{invio dei } T \\ \text{sui backedge} \end{array}$$

$$\Theta(m)$$

$$\frac{\rightarrow T}{T \leftarrow}$$

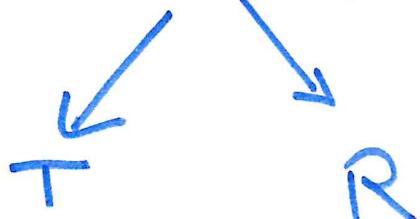
- Tempo :

Parliamo di tempo ideale :

no ritardi dei msg

⇒ gli errori non capitano

Inoltre visited msg solo in sovrapposizioni
con T

$$\Rightarrow \text{Tempo} : 2(n-1) = O(n)$$


DF^* \leftarrow protocollo modificato
con questa idea dei msg visited

ed è OTTIMO per:

- M [DF^*]
- T [DF^*]

Protocol DF*

$S = \{ \text{INITIATOR}, \text{IDLE}, \text{AVAILABLE}, \text{VISITED}, \text{DONE} \}$

$S_{\text{INIT}} = \{ \text{INITIATOR}, \text{IDLE} \}$

$S_{\text{TERM}} = \{ \text{DONE} \}$

Initiator

Spontaneously

{ initiator = True ;

Unvisited = $N(x)$;

nextT \leftarrow Unvisited ;

send T to nextT ;

send visited to $N(x) - \text{nei}$

they become VISITED

IDLE

Receiving (T)

{ Unvisited = $N(\infty)$;
FIRST-VISIT;

}

Receiving (Visited)

{ Unvisited = $N(\infty) - \text{sender}$;
become AVAILABLE

}

AVAILABLE

Receiving (T)

FIRST-VISIT;

Receiving (Visited)

{ Unvisited =
Unvisited - sender;

}

VISITED

Receiving (VISITED)

{ Unvisited = Unvisited - sender;
if (next = sender) then
 VISIT;
}

Receiving (T)

{ Unvisited = Unvisited - sender;
if (next = sender) then
 VISIT;

}

Receiving (Return)

{ VISIT;

}

Procedure FIRST-VISIT

```
{ initiator = false;  
entry = sender;
```

Unvisited = Unvisited - sender;

if (unvisited ≠ Ø) Then

{ next ∈ unvisited;

→ send (T) To next;

→ send (visited) To N(x) -
{ entry, next }

} become VISITED;

else

→ { send (Return) To entry;

→ send (visited) To N(x)-entry

become DONE;

}

}

Procedure VISIT

```
{ if Unvisited ≠ φ Then
    { next ← unvisited;
      → send (T) To next;
    }
  else
    { if (not initiator) then
      → send (Return) To entry;
      → become DONE;
    }
}
```