Algoritmi distribuiti su reti sincrone

Per questo capitolo si può fare riferimento al libro "Distributed Algorithms" di Nancy Lynch, Morgan Kaufmann Publishers, San Mateo, CA, 1996.

Introduzione alle reti sincrone

Una rete di agenti consiste nella collezione di "elementi di calcolo" localizzati ai nodi di un grafo orientato. Tali elementi di calcolo si chiamano comunemente *processori* e si possono sia riferire ad un hardware che ad un software. I canali di comunicazione attraverso i quali i processori scambiano informazione sono rappresentati dagli archi del grafo.

Nelle reti sincrone si suppone che i vari processori prendono delle decisioni o agiscono contemporaneamente. L'evoluzione dello stato della rete procede a "cicli di clock". L'ipotesi di simultaneità è forte ma la risoluzione di problemi su reti sincrone aiuta nella comprensione e nella soluzione di problemi su reti asincrone.

Le informazioni che i processori si scambiano si basano su un alfabeto $\mathcal{A}$, l'assenza di un messaggio si rappresenta dall'elemento $\epsilon \in \mathcal{A}$. Ogni processore i, più formalmente, consiste in varie componenti:

- state_i, un insieme (non necessariamente finito) di stati;
- start_i, un insieme non vuoto di state_i di stati iniziali;
- msgs_i una funzione che dato uno stato in state_i e un nodo della stella uscente da i fornisce un messaggio, i.e. un elemento di $\mathcal{A}$.
- trans_i, una funzione di transizione di stato che dato uno stato in state_i e un vettore di elementi di $\mathcal{A}$ (di dimensione pari alla stella entrante in i), fornisce un elemento di state_i.

Misure di complessità

Per gli algoritmi distribuiti su reti sincrone solitamente si utilizzano due misure di complessità: "complessità temporale" (time complexity) e la "complessità di comunicazione" (communication complexity).

La complessità temporale si misura in termini di "cicli di clock" necessari per ottenere l'uscita desiderata o fino a che tutti i processori non subiscano un alt.

La complessità di comunicazione è invece misurata in numero di messaggi non nulli che sono stati spediti attraverso la rete (in alcuni casi si considerano i bit trasmessi).

La complessità di comunicazione diventa importante quando si può avere un congestionamento della rete magari dovuto alla presenza di più algoritmi distribuiti che evolvono in contemporanea sulla rete.

Algoritmi di Leader Election

Quando i sistemi distribuiti si basano su politiche client-server con un unico server (leader) e più clients è necessario avere un sistema automatico di *Elezione del Leader* (*Leader Election*). Infatti nel caso in cui il leader subisce un guasto è necessario determinarne uno nuovo altrimenti la rete non è più in grado di funzionare.

Il primo algoritmo di Leader Election è quello descritto da Gerard LeLann nel 1977 per reti ad anello (un ciclo di dimensione arbitraria).

Consideriamo invece il caso di una rete rappresentata da un grafo $G = (V, E)$ orientato e fortemente connesso. Supponiamo che ogni processore (nodo) abbia un identificatore unico. In altre parole ad ogni nodo è associato un numero che lo identifica ed è diverso dagli identificatori degli altri nodi (non necessariamente consecutivi).

Si richiede che alla fine dell'algoritmo uno e un solo processore si classifica come leader cambiando una componente del proprio stato al valore "*leader*". In alcuni casi tutti gli altri processori assegnano al loro stato il valore "*non leader*".

Per questo tipo di problema non è necessario che i processori conoscano il numero totale di processori della rete (numero n di nodi).

Algoritmo FloodMax

Questo semplice algoritmo consente ai processori di classificarsi come "leader" o "non leader" e necessita della conoscenza del diametro D del grafo. Il nome dell'algoritmo viene dal fatto che il massimo valore identificativo dei processori viene fatto girare attraverso la rete. L'idea è quella che ogni processore comunica ai vicini il massimo valore identificativo che ha ricevuto dall'inizio dell'algoritmo. Dopo un numero di cicli di clock pari al diametro della rete ogni processore è non leader se ha un identificativo minore rispetto ai valori ricevuti altrimenti è il leader.

Più formalmente lo stato state_i di un nodo è dato da due identificativi: u, l'identificativo caratterizzante il nodo stesso e $max - uid$, il valore massimo tra gli identificativi ricevuti.

Entrambi i valori vengono inizializzati al valore dell'identificativo del nodo.

Un altra componente dello stato dell'agente è la classificazione tra *"non–noto, non–leader, leader"*, inizializzata a non–noto.

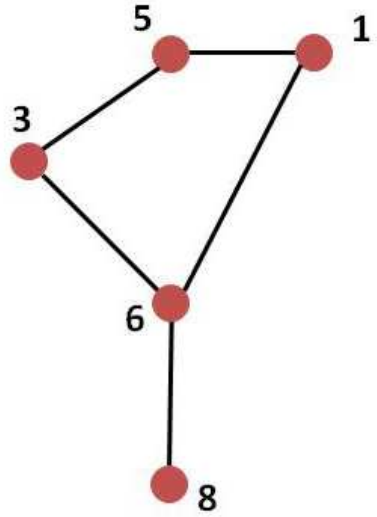
Infine, lo stato tiene traccia anche del numero di cicli di clock con un valore *rounds*, inizializzato a zero.

La componente msgs_i è tale che: se $rounds < D$ allora ad ogni ciclo di clock (o ogni round) l'agente invia il valore corrente di $max - uid$ a tutti i nodi della stella uscente.

La funzione di transizione aggiorna le variabili di stato come segue:

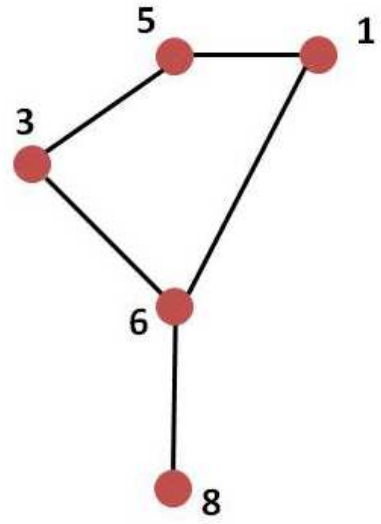
- $rounds := rounds + 1$;
- $max - uid := \max(\{max - uid\} \cup U)$;
- se $rounds = D$ allora se $max - uid = u$, $status := leader$ altrimenti $status := non - leader$.

Sia i_{max} l'indice del nodo con identificatore massimo e u_{max} il valore dell'identificatore massimo.



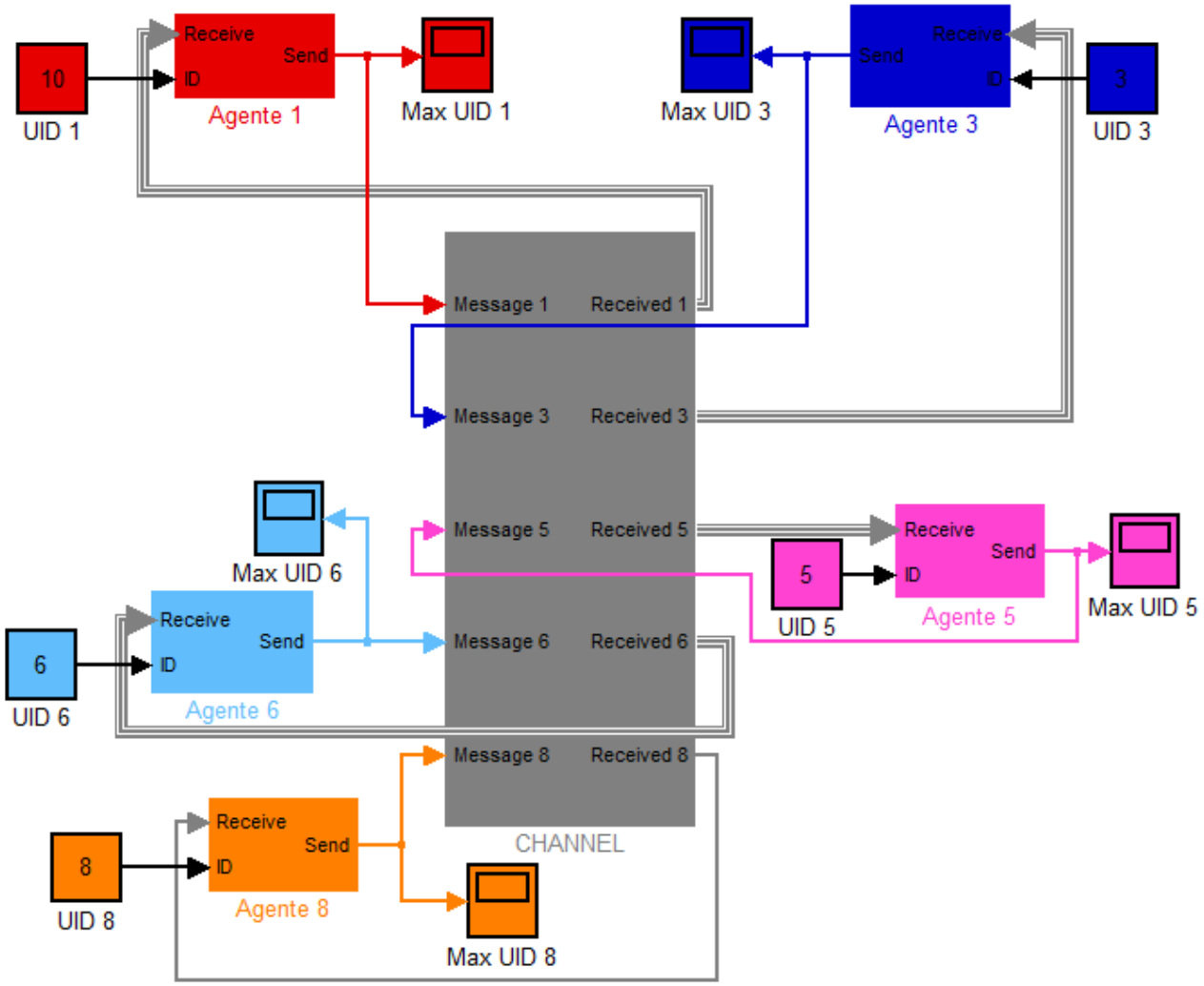
u	Invia	Riceve	Max-uid	Leader	Rounds
3			3	ignoto	0
5			5	ignoto	0
1			1	ignoto	0
6			6	ignoto	0
8			8	ignoto	0

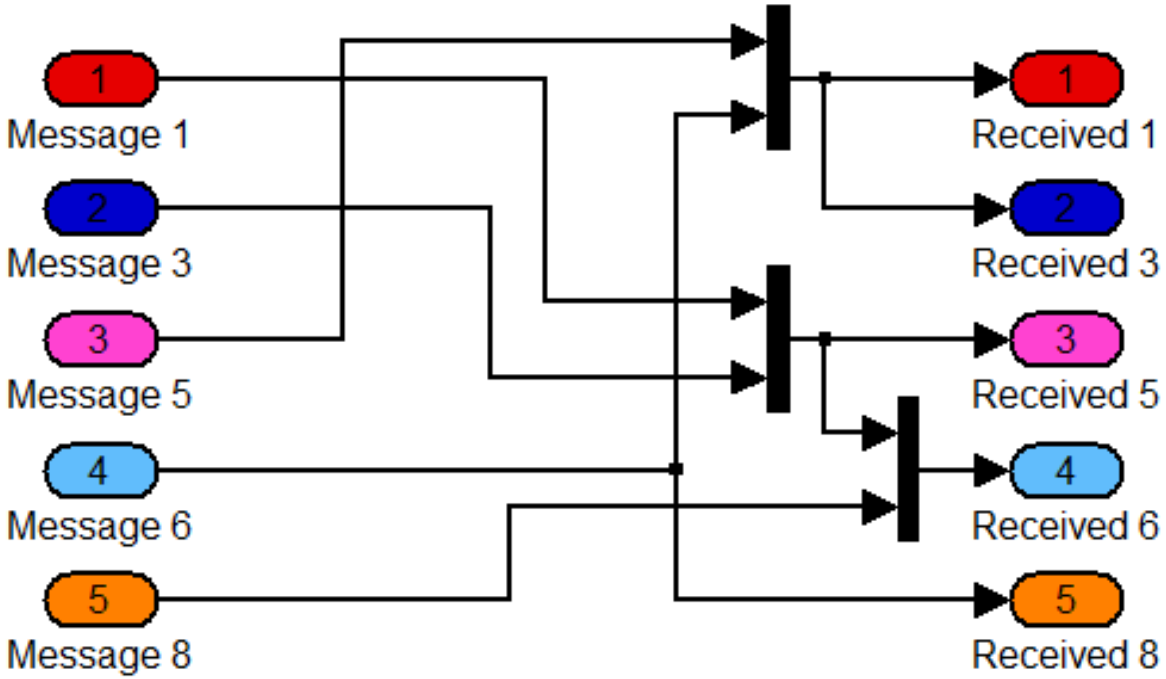
u	Invia	Riceve	Max-uid	Leader	Rounds
3	3	5,6	**6**	ignoto	1
5	5	3,1	5	Ignoto	1
1	1	5,6	**6**	ignoto	1
6	6	3,1,8	**8**	ignoto	1
8	8	6	8	ignoto	1



u	Invia	Riceve	Max-uid	Leader	Rounds
3	6	5,8	8	ignoto	2
5	5	6,6	6	ignoto	2
1	6	5,8	8	ignoto	2
6	8	6,6,8	8	ignoto	2
8	8	8	8	ignoto	2

u	Invia	Riceve	Max-uid	Leader	Rounds
3	8	6,8	8	No	3
5	6	8,8	8	No	3
1	8	6,8	8	No	3
6	8	8,8,8	8	No	3
8	8	8	8	Leader	3


10
UID 1
Receive
ID
Send
Agente 1
Max UID 1
Max UID 3
Receive
Send
ID
Agente 3
3
UID 3
Message 1
Received 1
Message 3
Received 3
Max UID 6
Message 5
Received 5
Receive
Send
ID
5
UID 5
Agente 5
Max UID 5
6
UID 6
Receive
ID
Send
Agente 6
Message 6
Received 6
Message 8
Received 8
CHANNEL
8
UID 8
Receive
ID
Send
Agente 8
Max UID 8


1
Message 1
2
Message 3
3
Message 5
4
Message 6
5
Message 8
1
Received 1
2
Received 3
3
Received 5
4
Received 6
5
Received 8

Teorema 6. *Con l'algoritmo FloodMax il processore i_{max} ha come uscita il valore leader e tutti gli altri processori hanno valore di uscita non–leader.*

Dimostrazione. Dato l'algoritmo FloodMax si ha che

$$\forall r, j \text{ dopo } r \text{ cicli di clock } rounds_j = r,$$

$$\forall r, j \text{ dopo } r \text{ cicli di clock } max - uid_j \leq u_{max},$$

da ciò segue che

per $0 \leq r \leq D$ e $\forall j$ dopo r cicli di clock, se la distanza da i_{max} a j è al più r allora $max - uid_j = u_{max}$.

Applicando quanto ottenuto per $r = D - 1$ e applicando un passo dell'algoritmo si ha che

Dopo D cicli di clock, $status_{i_{max}} = leader$ e $statu_j = non - leader \forall j \neq i_{max}$

□

Analisi della complessità

Il leader è eletto dopo un tempo pari a D mentre il numero totale di messaggi attraverso la rete è di $D|E|$ dove $|E|$ rappresenta il numero di archi del grafo.

Un modo per diminuire la complessità di comunicazione è quella di consentire ai processori di inviare il messaggio solo se il valore $max - uid$ è stato aggiornato.

Algoritmi di ricerca su grafi

Gli algoritmi di ricerca su grafi consentono di esaminare il grafo e le sue caratteristiche.

Alcuni algoritmi di ricerca consentono di costruire un albero di copertura e questo, nell'ambito delle reti di agenti, è utile per la comunicazione ad esempio minimizzando il massimo tempo impiegato da un messaggio per andare da un processore a tutti gli altri processori della rete.

Breadth–First

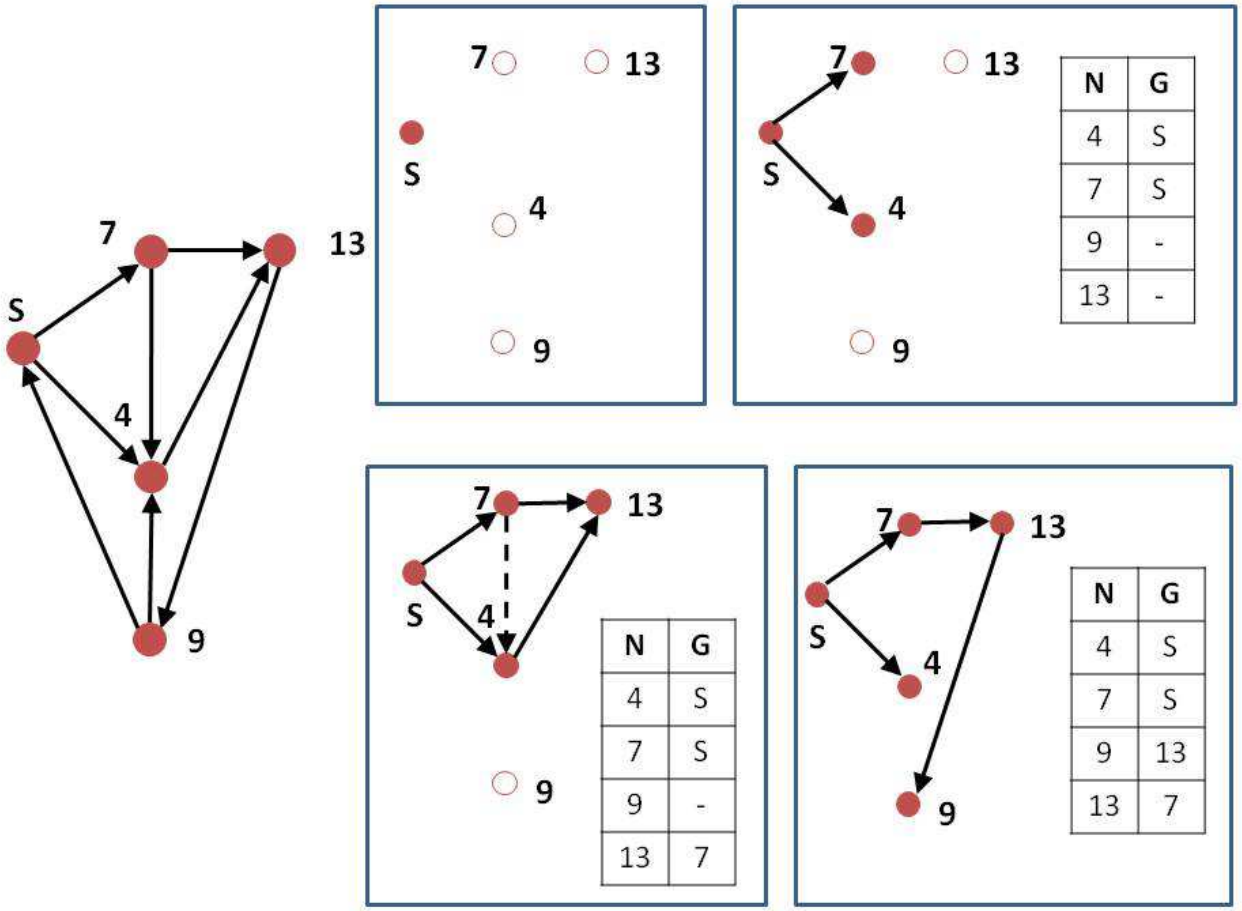
Dato un grafo G orientato e fortemente connesso, la ricerca di tipo Breadth–First (Ricerca in ampiezza) genera un albero (orientato) di copertura del grafo a partire da un nodo sorgente. La ricerca avviene in maniera esaustiva fino a quando non si sono visitati tutti i nodi del grafo.

L'algoritmo decentralizzato che determina l'albero di copertura Breadth–First fa in modo di assegnare ad ogni processore diverso da s un valore alla componente *genitore* dello stato pari al nodo che nell'albero risulta essere genitore del processore.

La caratteristica dell'albero con sorgente s, trovato con un algoritmo di tipo Breadth–First, è quella che ogni nodo a distanza d da s in G risulta a profondità d nell'albero.

In questo caso si suppone che ogni processore ha un identificativo ma non conosce la dimensione o il diametro della rete.

L'algoritmo parte con il solo processore s *marcato*. Ad ogni passo un nodo che è passato da non marcato a marcato (e solo al primo passo dopo la marcatura!) spedisce un messaggio *search* ai nodi della sua stella uscente. Ogni volta che un nodo non marcato riceve un messaggio *search* viene marcato e assegna alla componente *genitore* il valore di uno dei nodi da cui ha ricevuto il messaggio.

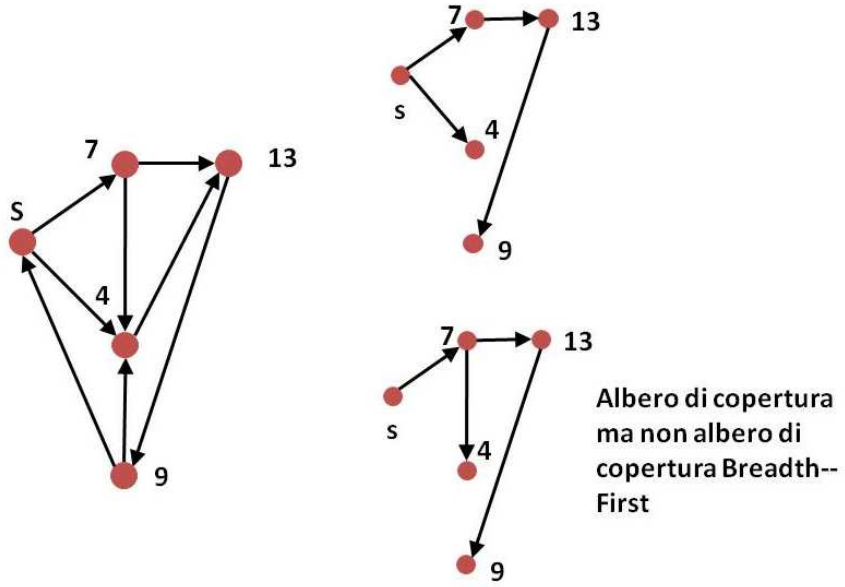

7
13
S
4
9
7
13
S
4
9
N G
4 S
7 S
9 -
13 -
7
13
S
4
9
N G
4 S
7 S
9 -
13 7
7
13
S
4
9
N G
4 S
7 S
9 13
13 7

Teorema 7. *L'algoritmo Breadth–First fornisce un albero di copertura Breadth–First.*

Dimostrazione. Per induzione si dimostra che dopo r cicli di clock ogni processore a distanza d da s con $1 \leq d \leq r$ ha il valore *genitore* definito. Inoltre, tale valore è associato ad un nodo a distanca $d-1$ da s. □

Analisi della complessità

La complessità temporale è al più di D cicli di clock. Il numero di messaggi trasmessi è invece pari a $|E|$.



Spedire messaggi in Broadcast

Per inviare un messaggio in broadcast nella rete è possibile utilizzare l'algoritmo Breadth–First leggermente modificato.

Infatti, se si vuole spedire un messaggio m a tutti i nodi della la rete il nodo che deve inviare un messaggio fa partire l'algoritmo in cui il messaggio m viene in un certo senso ancorato al messaggio *search* (in elettronica e telecomunicazioni si usa il termine di *piggybacking message*).

Ogni processore che riceve il messaggio *search* (con il messaggio m ancorato ad esso) inserisce a sua volta il messaggio m nel proprio messaggio *search*. Visto che l'algoritmo Breadth–First fornisce un albero di copertura, il messaggio m alla fine viene ricevuto da tutti i nodi.

Puntatore ai *figli*

Nel caso in cui si voglia fare in modo che ogni nodo non solo conosca l'identità dei propri *genitori* nell'albero Breadth–First ma anche l'identità dei *figli* è possibile modificare l'algoritmo Breadth–First. Nel caso di grafi non orientati una facile soluzione è quella di consentire ad ogni nodo di inviare un messaggio *genitore* o *non genitore* a tutti i nodi da cui ha ricevuto il messaggio *search* a seconda che siano stati scelti o meno come *genitori*. Se il grafo è orientato il messaggio *genitore* o *non genitore* non viene recapitato direttamente. In questo caso si può ancorare il messaggio (con l'identificativo del nodo genitore che deve riconoscere il messaggio e del nodo figlio) al messaggio *search* di un nuovo algoritmo Breadth–First che evolverà in parallelo con il precedente.

Esistono tecniche per determinare quando l'albero è stato costruito e cioè quando ogni nodo ha ricevuto risposta dai propri nodi *figli*.

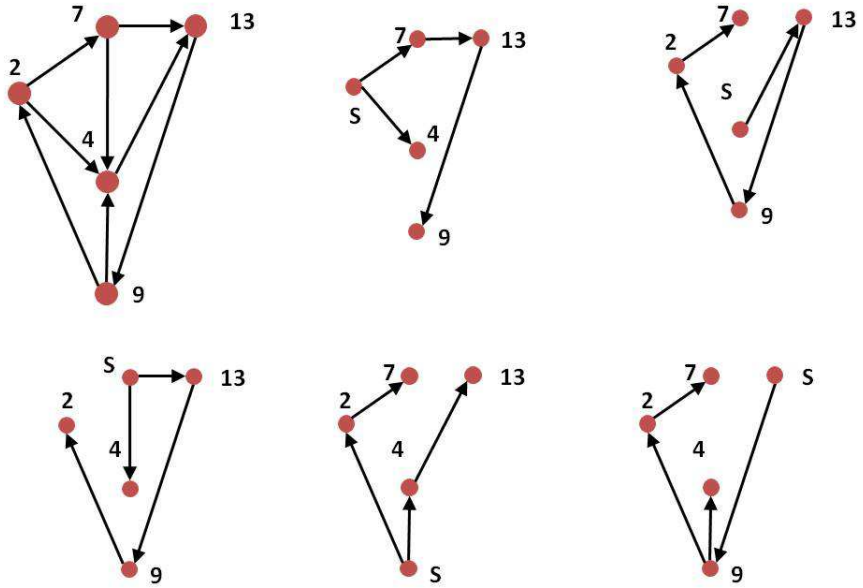


Figura 19: Algoritmo Breadth-First a partire da sorgenti diverse

L'algoritmo Breadth–First è molto utile per la costruzione di algoritmi decentralizzati.

Ad esempio abbiamo visto che se si vuole inviare un messaggio da un nodo s a tutti i nodi della rete il Breadth–First può essere utilizzato per costruire l'albero di comunicazione attraverso il quale un messaggio impiega dell'ordine del diametro D della rete per essere recapitato e il numero di messaggi è dell'ordine di n.

Un'altra possibile applicazione è quella della elezione di un Leader nel caso di una rete con identificativi ma in cui i processori non hanno informazioni sulla dimensione della rete.

Il problema si risolve inizializzando delle ricerche di tipo Breadth–First in parallelo sulla rete. Ogni processore usa l'albero ottenuto per calcolare il massimo valore identificativo. Questo è fattibile facendo inviare da ogni foglia (nodo senza figli) l'identificativo al proprio genitore (per ogni albero ottenuto).

Ogni nodo calcola il massimo dei valori ottenuti dai propri figli e invia a sua volta il valore al proprio genitore. Il valore che ogni nodo riceve come radice dell'albero calcolato con il Breadth–First è il massimo identificativo della rete. Se l'identificativo del nodo è pari al valore ottenuto allora si auto-elegge leader.

Con lo stesso tipo di tecnica si possono effettuare varie operazioni ed far si che ogni nodo della rete ottenga lo stesso valore.

Shortest-Path

Sia G un grafo pesato orientato e fortemente connesso i cui pesi siano non negativi. Si definisce costo del cammino la somma dei pesi degli archi del cammino stesso.

Si vuole determinare i cammini di costo minimo da un nodo sorgente s a tutti gli altri nodi della rete.

Se il costo sugli archi rappresentasse il ritardo nella comunicazione attraverso quel canale il problema riguarderebbe la ricerca di un canale di comunicazione da una sorgente ai vari nodi che minimizza il ritardo di comunicazione.

In questo caso è necessario che entrambi i nodi dell'arco siano a conoscenza del suo costo.

Si suppone inoltre che i processori conoscano il numero totale n di nodi della rete.

Infine, ogni processore dovrà calcolare (con le tecniche già viste) il proprio genitore in un particolare albero di cammini minimi e anche la sua distanza da s (ottenuta sommando il costo degli archi).

Un possibile algoritmo si basa su un algoritmo (Bellman-Ford) per il calcolo sequenziale di cammini minimi.

Durante l'esecuzione dell'algoritmo, ogni processore i tiene traccia della distanza d_i (attualmente nota) da s e del *genitore* che precede i nel cammino di lunghezza d_i.

Inizialmente $d_s = 0$ e $d_i = \infty$ per ogni $i \neq s$, la componente *genitore* non è definita.

Ad ogni passo ogni processore i invia il messaggio d_i a tutti i nodi della sua stella uscente. Dati i valori d_j ricevuti dai nodi della stella entrante, il processore i confronta il valore d_i con il minimo di tutti i valori $d_j + c(j, i)$ (dove $c(j, i)$ è il costo dell'arco da j ad i).

Se il minimo tra questi valori è più piccolo di d_i allora si aggiorna $d_i := d_k + c(k, i)$ e la componente *genitore* assume valore k dove k è il processore per cui il minimo dei $d_j + c(j, i)$ viene raggiunto.

Dopo $n-1$ cicli di clock il valore d_i è la distanza minima da s ad i e la componente *genitore* è il genitore dell'albero di cammini minimi.

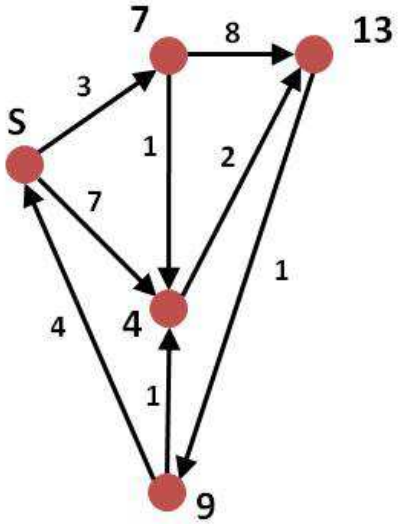
Teorema 8. *L'algoritmo Shortest–Path fornisce un albero di camini minimi.*

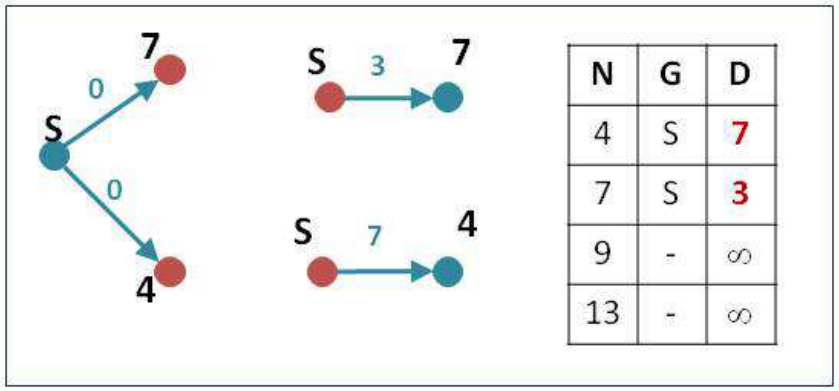
Dimostrazione. Per induzione si dimostra che dopo r cicli di clock ogni processore i ha componenti d_i e *genitore* corrispondenti al cammino minimo tra tutti i cammini da s ad i con al più r archi. □

Analisi della complessità

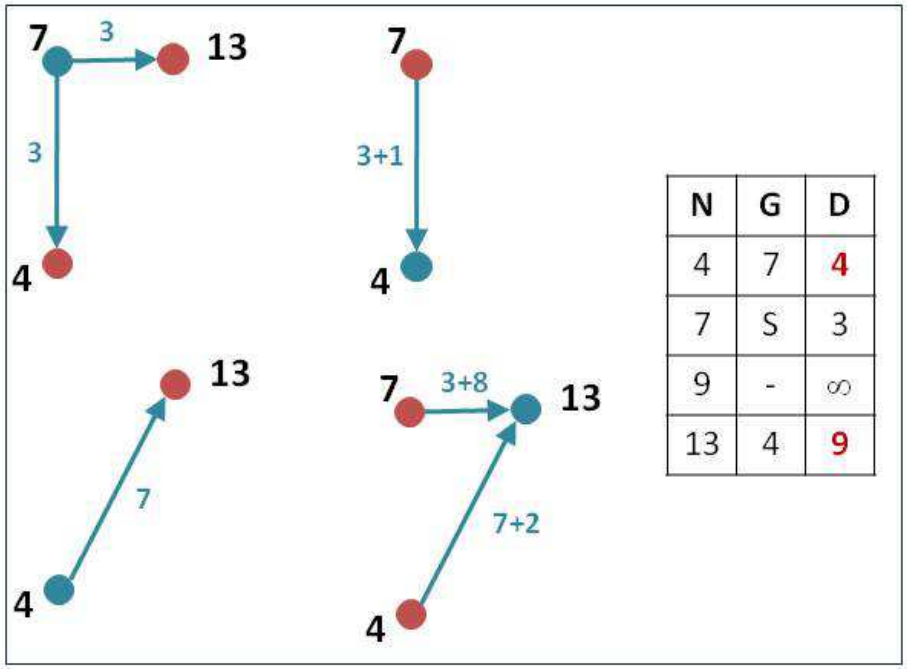
La complessità temporale è di $n-1$ cicli di clock. Il numero di messaggi trasmessi è invece pari a $(n-1)|E|$.

Esempio 12.

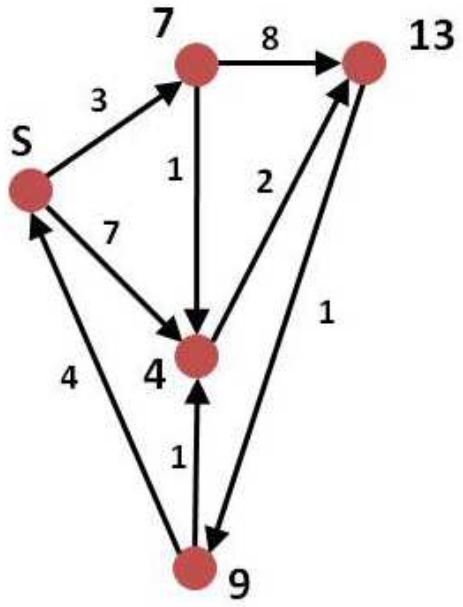


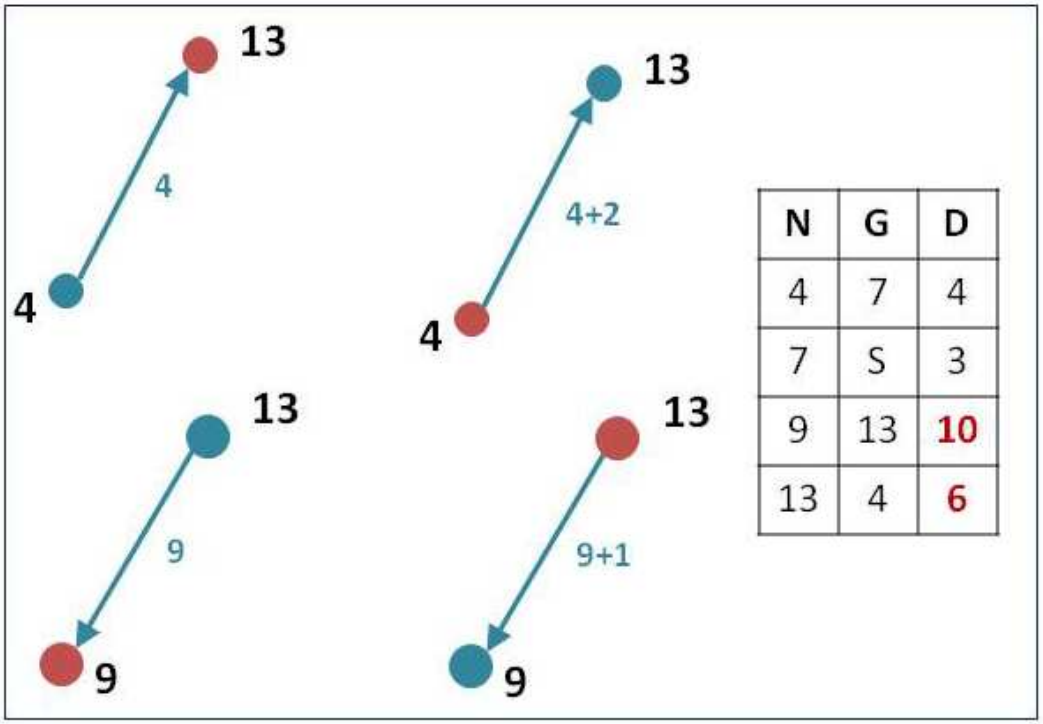


N	G	D
4	S	7
7	S	3
9	-	∞
13	-	∞

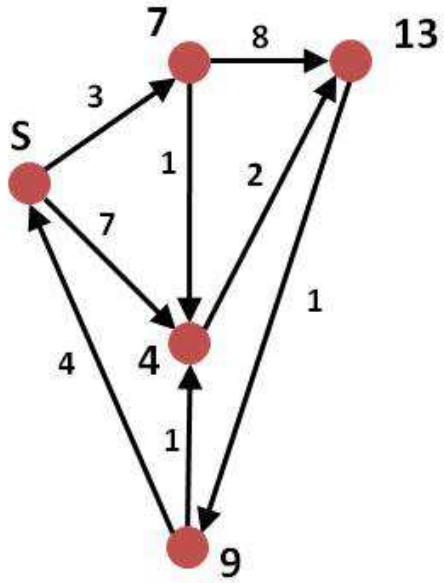


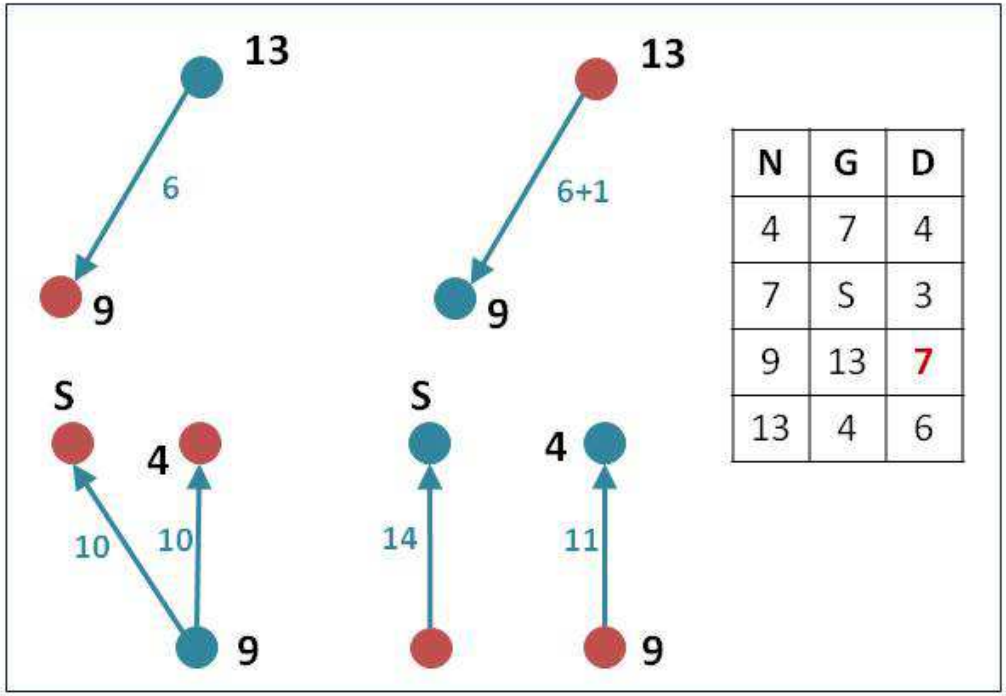
N	G	D
4	7	4
7	S	3
9	-	∞
13	4	9



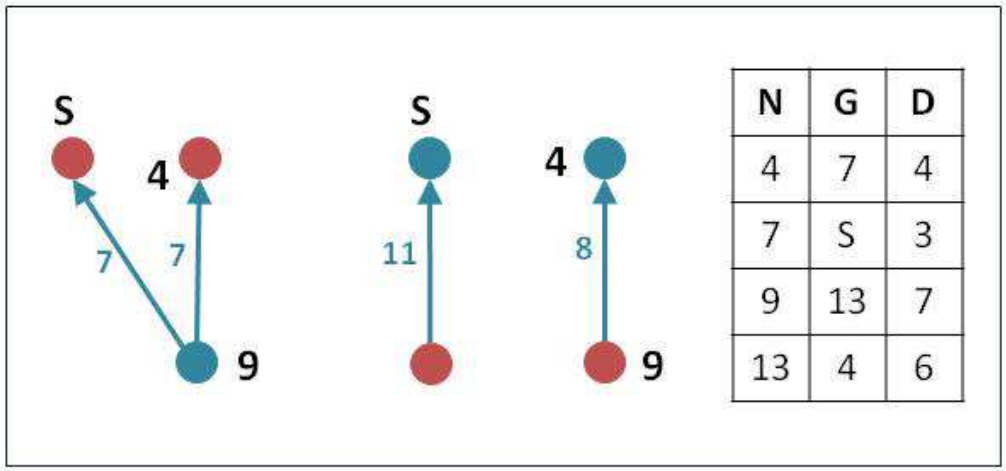


N	G	D
4	7	4
7	S	3
9	13	10
13	4	6





N	G	D
4	7	4
7	S	3
9	13	7
13	4	6



N	G	D
4	7	4
7	S	3
9	13	7
13	4	6