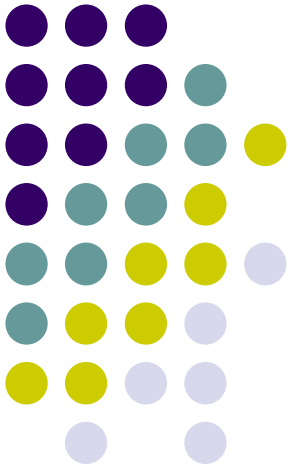
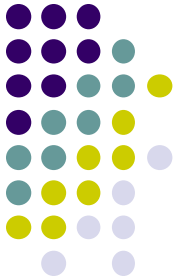
Κατανεμημένα Συστήματα

Μάθημα #8

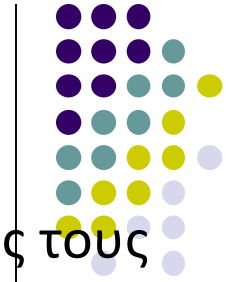


Περιεχόμενα



- **Κατασκευή Δένδρου Επικάλυψης**
- **Κατασκευή DFS δένδρου**
- **Εκλογή Αρχηγού σε Δένδρο**
- **Εκλογή Αρχηγού σε Ισχυρά Συνδεδεμένο Γράφο - Αλγόριθμος του Εξαναγκασμού**
- **Εκλογή Αρχηγού σε Γενικό Δίκτυο -Αλγόριθμος FloodMax**
- **Εκλογή Αρχηγού σε Γενικό Δίκτυο -Αλγόριθμος OptFloodMax**

Εκπομπή (broadcast) μηνύματος



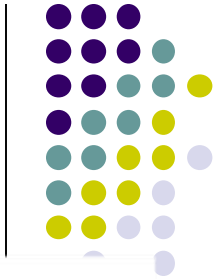
Εκπομπή (broadcast) μηνύματος Μ από έναν κόμβο p_r σε όλους τους άλλους κόμβους ενός **γράφου n διεργασιών δοθέντος δένδρου επικάλυψης (spanning tree) με ρίζα p_r**

Περιγραφή:

- Ο p_r αρχικά στέλνει το Μ στα παιδιά του
- Όταν ένας επεξεργαστής λαμβάνει το Μ από τον πατέρα του, το στέλνει στα παιδιά του και τερματίζει.

Ο αλγόριθμος έχει **πολυπλοκότητα επικοινωνίας n-1** και **χρονική πολυπλοκότητα d**, όπου d είναι το βάθος του δένδρου επικάλυψης του γράφου των διεργασιών.

Εκπομπή (broadcast) μηνύματος



Algorithm 1 Spanning tree broadcast algorithm.

Initially $\langle M \rangle$ is in transit from p_r to all its children in the spanning tree.

Code for p_r:

1: upon receiving no message: // first computation event by p_r
2: terminate

Code for p_i, $0 \leq i \leq n-1, i \neq r$:

3: upon receiving $\langle M \rangle$ from parent:
4: send $\langle M \rangle$ to all children
5: terminate

Συλλογή πληροφορίας σε έναν κόμβο

Convergecast - Συλλογή στον κόμβο p_r της μέγιστης τιμής πληροφοριών που έχουν οι κόμβοι γράφου n διεργασιών, δοθέντος δένδρου επικάλυψης (spanning tree) με ρίζα p_r και βάθους d

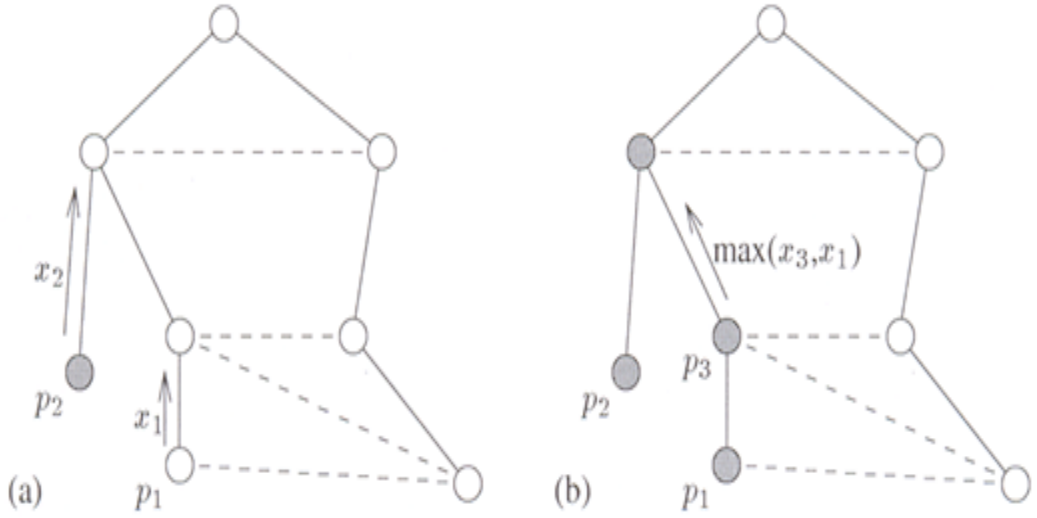
- Κάθε κόμβος-φύλλο του δένδρου p_i ξεκινά με μια μεταβλητή x_i, στην οποία είναι αποθηκευμένη η πληροφορία του και την οποία αποστέλλει στον πατέρα του
- Σε κάθε βήμα, κάθε κόμβος p_j που παρέλαβε μήνυμα από όλα τα παιδιά του, αποστέλλει στον πατέρα του τη μέγιστη από τις τιμές των μεταβλητών που παρέλαβε

  και της x_j.

Χρονική Πολυπλοκότητα: O(d)

Πολυπλοκότητα

Επικοινωνίας: n-1



Εκπομπή πληροφορίας από έναν κόμβο

Εκπομπή πολλαπλών αποδεκτών από την ρίζα p_r σε γράφο n διεργασιών, m ακμών & διαμέτρου D, χωρίς γνώση του δένδρου επικάλυψης.

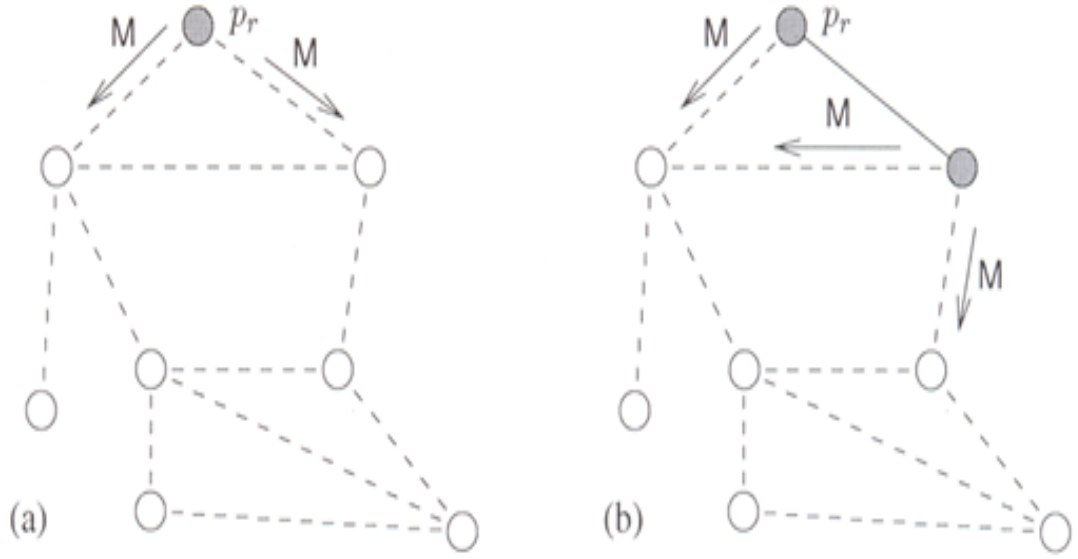
Αλγόριθμος Flooding

- Η p_r στέλνει το μήνυμα M σε όλους του γείτονες του.
- Όταν κάποια διεργασία p_i λαμβάνει το M για πρώτη φορά από κάποια διεργασία $\mathbf{p_j}$, το στέλνει σε όλους τους γείτονές της **εκτός από την $\mathbf{p_j}$.**

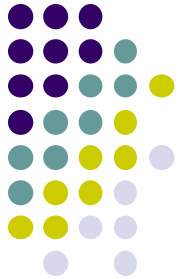
Χρονική Πολυπλοκότητα: D

Πολυπλοκότητα

Επικοινωνίας: 2m-(n-1).



Κατασκευή δένδρου επικάλυψης



Αλγόριθμος Spanning Tree //Τροποποίηση του Αλγορίθμου Flooding

- **Η p_r στέλνει μήνυμα Μ σε όλους τους γείτονές της**
- **Όταν μια διεργασία p_i λαμβάνει το Μ για πρώτη φορά από μία διεργασία, ορίζει τη διεργασία αυτή να είναι η γονική της διεργασία (στο δένδρο επικάλυψης που θα προκύψει τελικά).**
- **Έστω ότι η διεργασία που επιλέγεται ως γονική είναι η p_j. Η p_i στέλνει μήνυμα τύπου <parent> στην p_j και μήνυμα τύπου <already> σε όλες τις άλλες διεργασίες από τις οποίες έλαβε το Μ**
- **Η p_i στέλνει το Μ σε όλους τους γείτονές της και όταν λάβει απάντηση από αυτούς τερματίζει.**

Algorithm 2 Modified flooding algorithm to construct a spanning tree: code for processor p_i, $0 \leq i \leq n-1$.

Initially $parent = \perp$, $children = \emptyset$, and $other = \emptyset$.

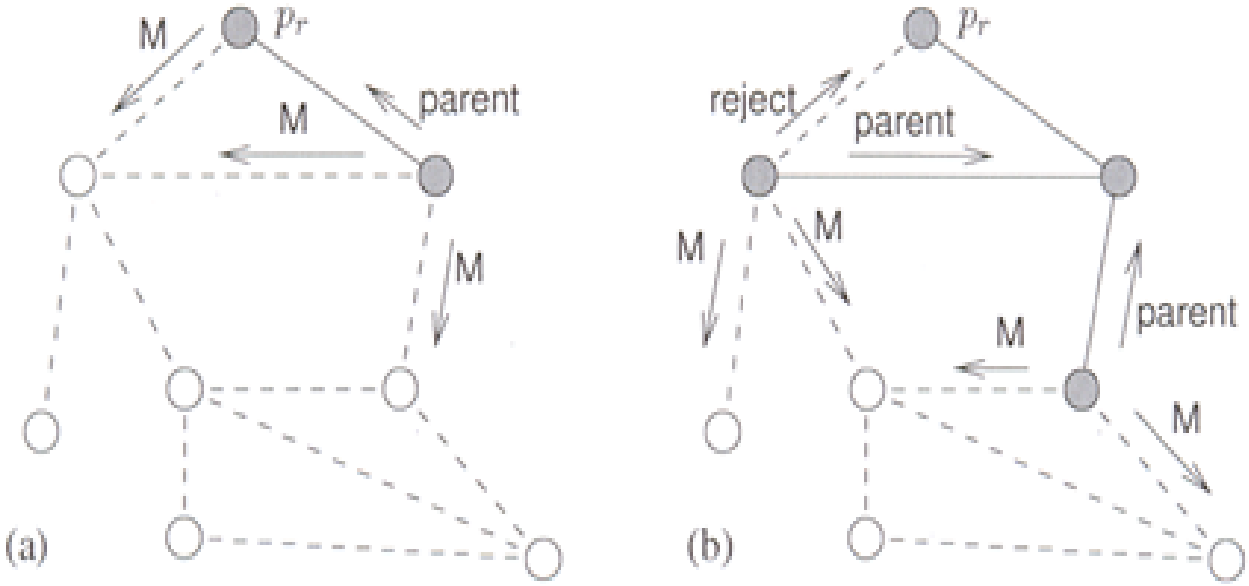
```
1:  upon receiving no message:
2:      if p_i = p_r and parent = ⊥ then          // root has not yet sent ⟨M
3:          send ⟨M⟩ to all neighbors
4:          parent := p_i

5:  upon receiving ⟨M⟩ from neighbor p_j:
6:      if parent = ⊥ then                        // p_i has not received ⟨M⟩ befo
7:          parent := p_j
8:          send ⟨parent⟩ to p_j
9:          send ⟨M⟩ to all neighbors except p_j
10:     else send ⟨already⟩ to p_j

11: upon receiving ⟨parent⟩ from neighbor p_j:
12:     add p_j to children
13:     if children ∪ other contains all neighbors except parent then
14:         terminate

15: upon receiving ⟨already⟩ from neighbor p_j:
16:     add p_j to other
17:     if children ∪ other contains all neighbors except parent then
18:         terminate
```

Κατασκευή δένδρου επικάλυψης



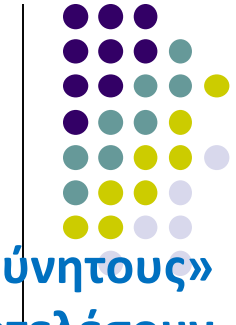
Εύκολο να δείξουμε ότι δεν υπάρχει κύκλος και ότι

υπάρχει μονοπάτι από τη ρίζα προς κάθε άλλο κόμβο.

Σε περίπτωση σύγχρονης εκτέλεσης το δένδρο που προκύπτει είναι Bread First Search (BFS) δένδρο.

Σε περίπτωση ασύγχρονης εκτέλεσης το δένδρο δεν είναι αναγκαστικά BFS.

Κατασκευή DFS (Depth-First-Search) Δένδρου Επικάλυψης δεδομένου του κόμβου ρίζα



- Κάθε κόμβος διατηρεί ένα σύνολο unexplored από «ανεξερεύνητους» γειτονικούς κόμβους και ένα σύνολο των κόμβων που θα αποτελέσουν παιδιά του στο DFS.
- Η ρίζα στέλνει το Μ σε έναν από τους γείτονές της τον οποίο διαγράφει από το σύνολο unexplored.
- Όταν ένας κόμβος p_i λάβει το Μ για πρώτη φορά από κάποιο κόμβο p_j, ο p_i σημειώνει τον p_j ως τον πατέρα του στο DFS. Στη συνέχεια, επιλέγει έναν από ανεξερεύνητους γείτονες του (unexplored) και του προωθεί το μήνυμα. Αν το σύνολο unexplored είναι κενό, ο p_i στέλνει ένα μήνυμα τύπου <parent> στον πατέρα του και τερματίζει.
- Αν ο p_i δεν λαμβάνει το Μ για πρώτη φορά, στέλνει ένα μήνυμα τύπου <already> στον p_j και τον διαγράφει από το unexplored.
- Όταν ένας κόμβος λάβει μήνυμα τύπου <parent> ή <already>, στέλνει το μήνυμα σε έναν ακόμη ανεξερεύνητους γείτονές του. Αν το unexplored είναι κενό, ο p_i στέλνει ένα μήνυμα τύπου <parent> στον πατέρα του και τερματίζει.

Algorithm 3 Depth-first search spanning tree algorithm for a specified root: code for processor p_i, $0 \leq i \leq n-1$.

Initially $parent = \perp$, $children = \emptyset$, $unexplored =$ all neighbors of p_i

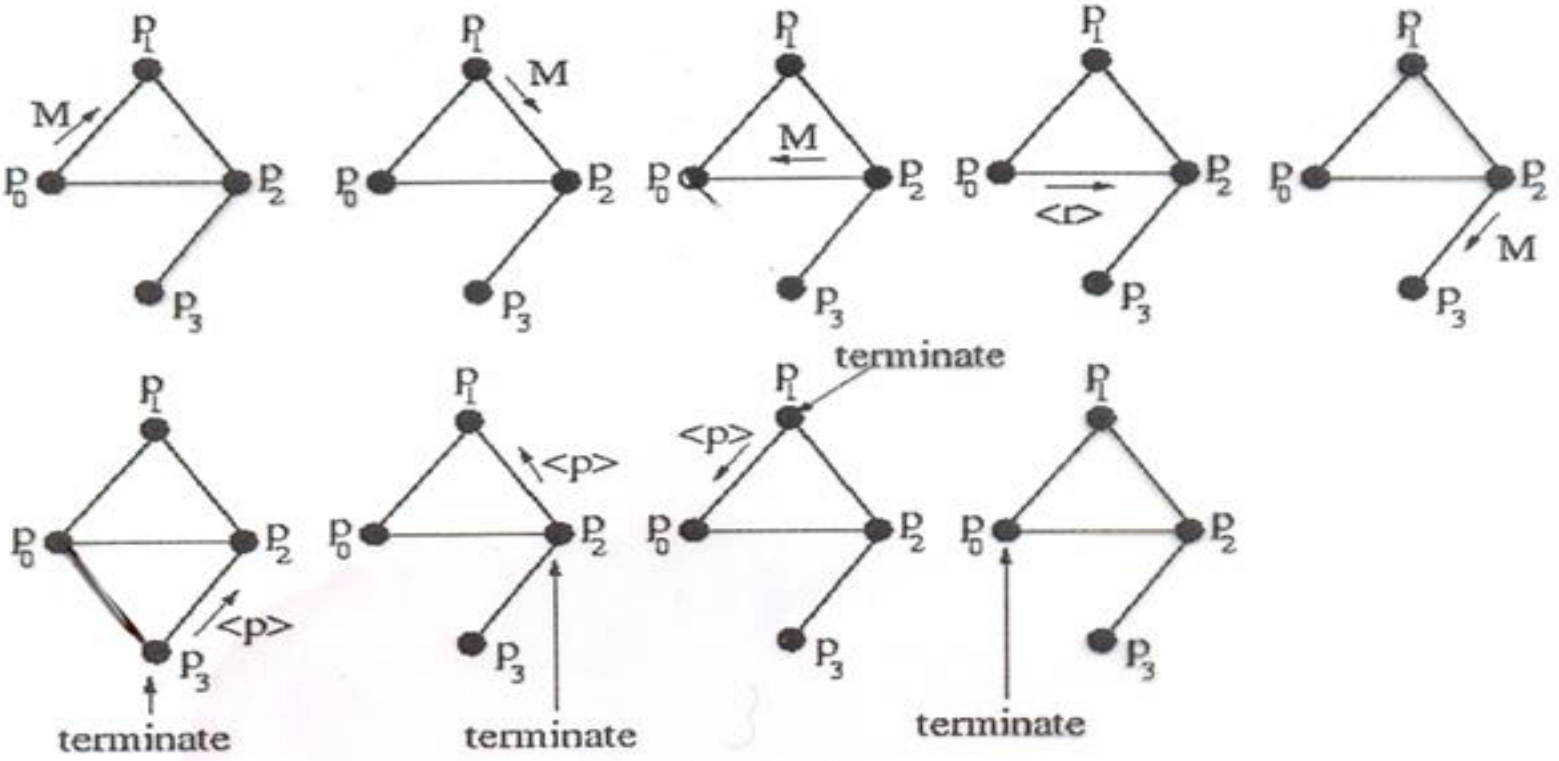
```
1:  upon receiving no message:
2:      if p_i = p_r and parent = ⊥ then                    // root wakes up
3:      parent := p_i
4:      explore()

5:  upon receiving ⟨M⟩ from p_j:
6:      if parent = ⊥ then                                  // p_i has not received ⟨M⟩ before
8:          parent := p_j
9:          remove p_j from unexplored
10:         explore()
11:     else
12:         send ⟨already⟩ to p_j                           // already in tree
13:         remove p_j from unexplored
14: upon receiving ⟨already⟩ from p_j:
15:     explore()

16: upon receiving ⟨parent⟩ from p_j:
17:     add p_j to children
18:     explore()
```

```
19: procedure explore():
20:     if unexplored ≠ ∅ then
21:         let p_k be a processor in unexplored
22:         remove p_k from unexplored
23:         send ⟨M⟩ to p_k
24:     else
25:         if parent ≠ p_i then send ⟨parent⟩ to parent
26:         terminate                    // DFS subtree rooted at p_i has been built
```

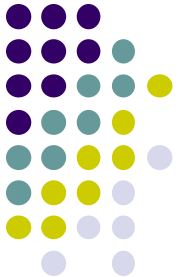
Παράδειγμα Εκτέλεσης του DFS



<r> μήνυμα τύπου already <p> μήνυμα τύπου parent

$unexplored_0 = \{2\}$	$parent_0 = nil$	$children_0 = \{\}$
$unexplored_1 = \{0,2\}$	$parent_1 = nil$	$children_1 = \{\}$
$unexplored_2 = \{0,1,2\}$	$parent_2 = nil$	$children_2 = \{\}$
$unexplored_3 = \{2\}$	$parent_3 = nil$	$children_3 = \{\}$

Πολυπλοκότητα του DFS



Λήμμα: Σε κάθε εκτέλεση του DFS αλγορίθμου, κατασκευάζεται ένα DFS δένδρο επικάλυψης του γράφου διεργασιών με ρίζα τον κόμβο p_r που ξεκινά τον αλγόριθμο.

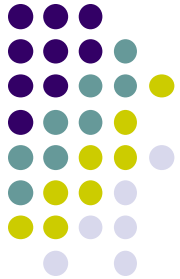
Λήμμα: Η πολυπλοκότητα επικοινωνίας του DFS αλγορίθμου είναι O(m).

Απόδειξη: Κάθε κόμβος (διεργασία) στέλνει το M το πολύ μια φορά σε κάθε μια από τις ακμές που πρόσκεινται σε αυτόν.

Κάθε κόμβος που παραλαμβάνει το M αποστέλλει το πολύ ένα μήνυμα ως απάντηση σε κάθε μια από τις ακμές που πρόσκεινται σε αυτόν.

Άρα συνολικά αποστέλλονται το πολύ O(m) μηνύματα.

Πολυπλοκότητα του DFS



Λήμμα: Η χρονική πολυπλοκότητα του αλγορίθμου DFS-ST είναι O(m).

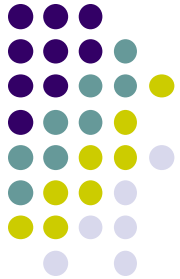
Απόδειξη:

Αφότου η ρίζα εκτελέσει το πρώτο της βήμα και πριν αυτή τερματίσει, υπάρχει πάντα ακριβώς ένα μήνυμα υπό αποστολή.

Σε κάθε ακμή δεν στέλνονται ποτέ περισσότερα από 2 μηνύματα.

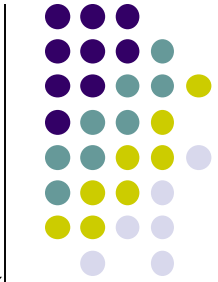
Υπάρχουν m ακμές στο σύστημα.

Εκλογή Αρχηγού σε Δένδρο



- Δίκτυα με τοπολογία δένδρου
- Δίκτυα στα οποία κατασκευάζεται ένα spanning tree
- Κάθε διεργασία γνωρίζει τα IDs των γειτόνων της
- Αρχηγός εκλέγεται η διεργασία με το μικρότερο ID
- Initiators (εναρκτές)
  - τουλάχιστον το σύνολο των φύλλων του δένδρου
  - όλες ξεκινούν τον αλγόριθμο στέλνοντας ένα μήνυμα <tok, ID>
- Non- Initiators
  - Οι υπόλοιπες διεργασίες στο δένδρο

Αλγόριθμος Εκλογής Αρχηγού σε Δένδρο



Κάθε διεργασία

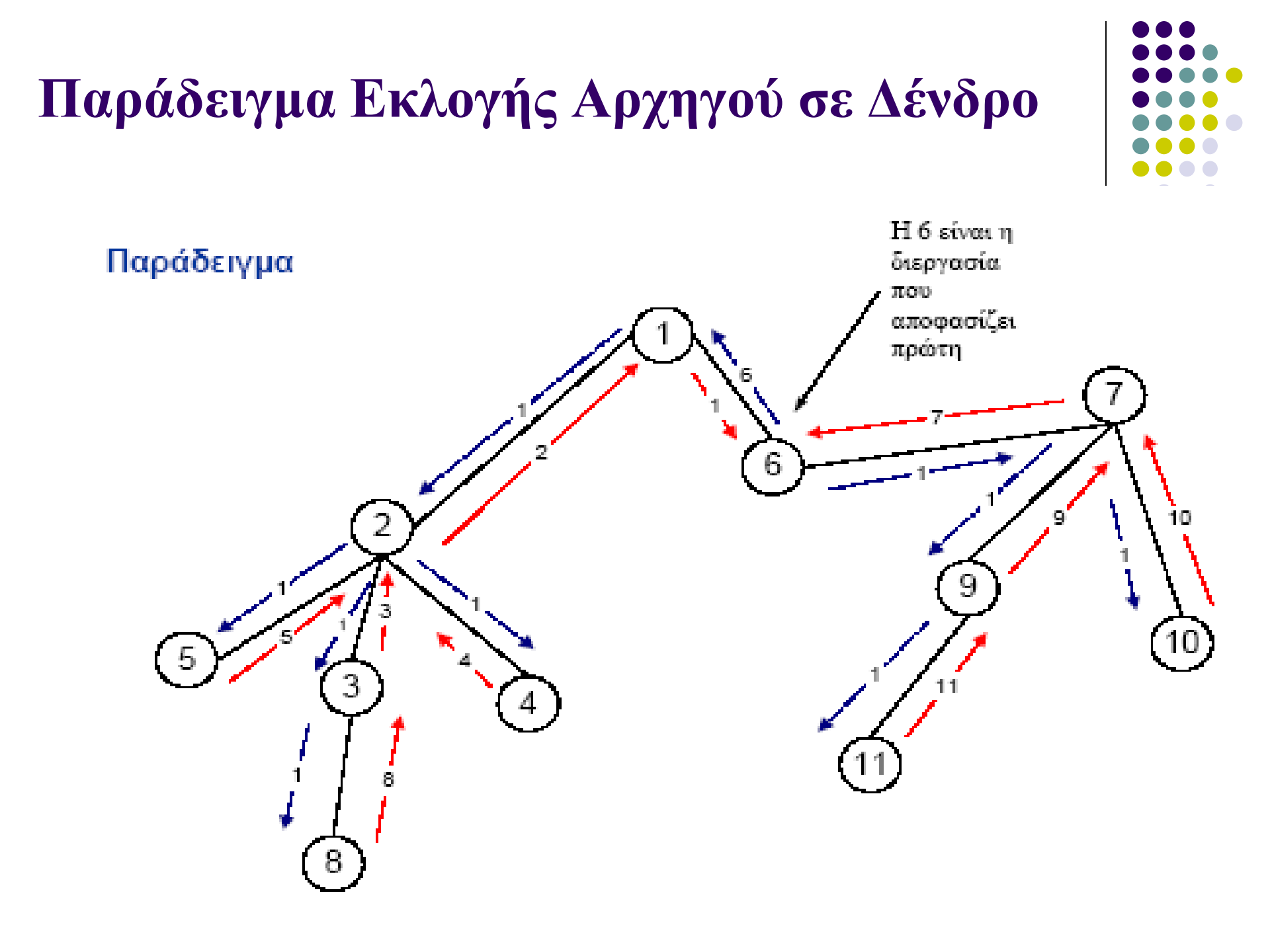
Περιμένει να λάβει μηνύματα <tok, ID> από όλους τους γείτονές της εκτός από το πολύ έναν, έστω τον P_0

Όταν ικανοποιηθεί η συνθήκη αυτή

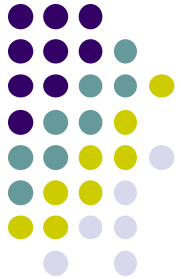
{*αρχικά ισχύει μόνο για τα φύλλα του δέντρου*}

- Υπολογίζει το min ID από τα ID's που έχει λάβει και το δικό της
- Στέλνει μήνυμα <tok, min ID > στον P_0
- Περιμένει μήνυμα <tok, ID> από τον P_0
- Υπολογίζει το νέο min ID από τα ID's που έχει λάβει και το δικό της
  - Αν τώρα min ID είναι το δικό της ανακηρύσσεται σε κατάσταση αρχηγού (ισχύος) – διαφορετικά ανακηρύσσεται σε κατάσταση χαμένου (μη ισχύος)
- Στέλνει σε όλους τους γείτονές της (εκτός του P_0) μήνυμα <tok,min ID >

Παράδειγμα Εκλογής Αρχηγού σε Δένδρο



Παράδειγμα Εκλογής Αρχηγού σε Δένδρο



- **5, 8, 4, 11, 10: στέλνουν <tok, min ID>**
- **3, 9: στέλνουν <tok, min ID>**
- **2,7: στέλνουν <tok, min ID>**
- **1: στέλνει <tok, min ID> (=1)**
- **6: στέλνει <tok, min ID> (=6) στην 1**

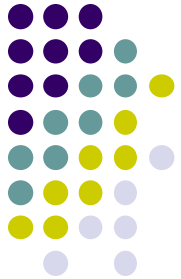
{ 11 μηνύματα <tok, min ID> }

- **6: αποφασίζει lost**
- **1: αποφασίζει leader**
- **6, 7, 9, 1, 2, 3 : στέλνουν <tok, 1> στους απογόνους τους (εκτός P_0)**

(1) (2) (1) (1) (3) (1) ***{ 9 μηνύματα <tok, 1> }***

{ συνολικά 20 μηνύματα = 2N-2 =2x11 -2 }

Εκλογή Αρχηγού σε Δένδρο



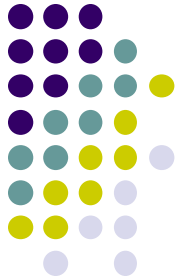
Πρόβλημα

- **Αν δεν είναι όλα τα φύλλα του δένδρου initiators τότε ο αλγόριθμος δεν δουλεύει !**

Αντιμετώπιση προβλήματος

- **Προστίθεται επιπλέον φάση *(Wake Up)* που «ξυπνάει» όλες τις υπόλοιπες διεργασίες και τις κάνει initiators**

Αλγόριθμος Wake Up



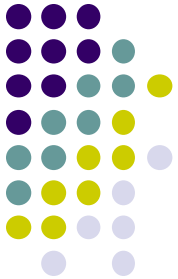
Κάθε διεργασία initiator

- **στέλνει ένα μήνυμα <wake up> σε κάθε γείτονά της**
- **περιμένει να λάβει μηνύματα <wake up> από όλους τους γείτονές της**
- **αρχίζει την εκτέλεση του αλγορίθμου εκλογής**

Κάθε διεργασία Non- Initiator όταν λάβει ένα μήνυμα <wake up>

- **γίνεται initiator**
- **στέλνει μηνύματα <wake up> σε κάθε γείτονά της**
- **περιμένει να λάβει μηνύματα <wake up> από όλους τους γείτονές της**
- **αρχίζει την εκτέλεση του αλγορίθμου εκλογής**

Αλγόριθμος Wake Up

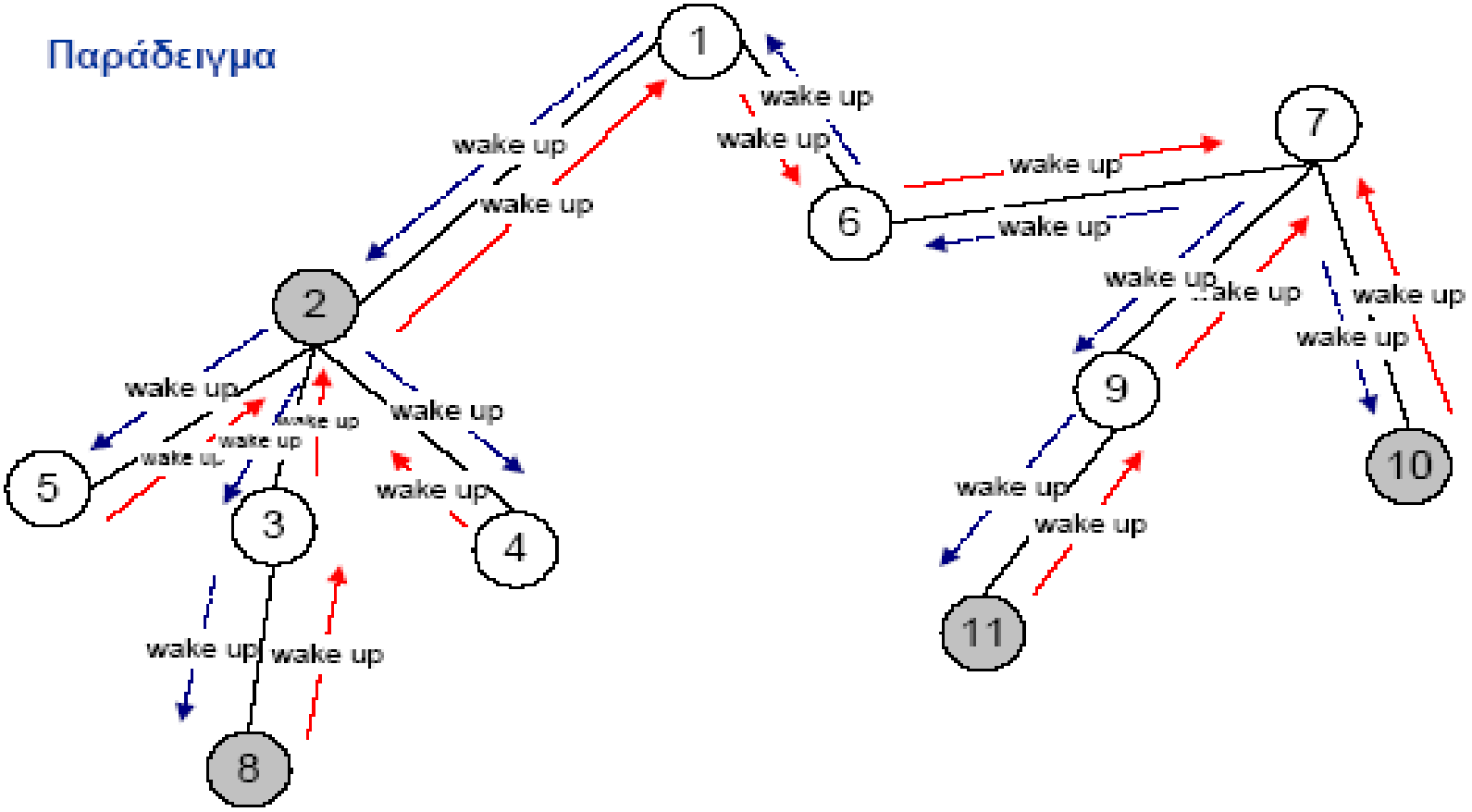


ws_p : *true* αν η διεργασία *p* έχει στείλει <*wake up*>
wr_p : # <*wake up*> μηνυμάτων που έχει λάβει η διεργασία
$Neigh_p$: τα αναγνωριστικά των γειτόνων της *p*

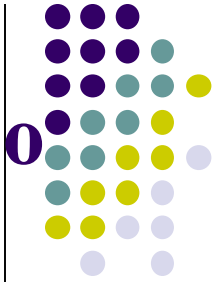
```
{
 Initialization: for all  p do { ws_p :=  false;   wr_p :=  0 }

   if p is initiator then
            { ws_p := true;
              for all q ∈ Neigh_p do send <wake up> to q }
    while wr_p < #Neigh_p do
                { receive <wake up>;
                  wr_p := wr_p + 1;
                   if not ws_p then
                          { ws_p :=true;
                            for all q ∈ Neigh_p do send <wake up> to q }
                 }
}
```

Παράδειγμα εκτέλεσης Wake Up



Πολυπλοκότητα Εκλογής Αρχηγού σε Δένδρο



Wake up

– **κάθε διεργασία ξεκινάει την εκτέλεση του αλγορίθμου αφού έχει λάβει το μήνυμα <wake up> από κάθε γείτονά της**

– **Από κάθε κανάλι επικοινωνίας στέλνονται 2 μηνύματα <wake up>**

– **Συνολικά στέλνονται 2N - 2 μηνύματα**

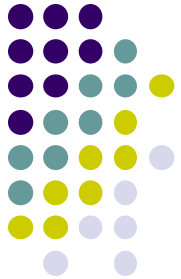
Εκλογή αρχηγού

– **Απαιτούνται 2N - 2 μηνύματα για την Wake up φάση**

– **Τα μηνύματα <tok, ID> που στέλνονται είναι 2 για κάθε κανάλι, ήτοι 2N - 2 μηνύματα**

Άρα, ο συνολικός αριθμός μηνυμάτων είναι 4N - 4

Εκλογή αρχηγού σε συνδεδεμένο γράφο – Αλγόριθμος του Εξαναγκασμού (Garcia-Molina Bully Algorithm)



Μοντέλο

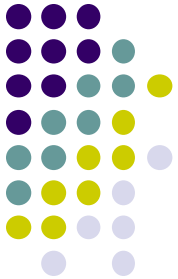
- **κάθε διεργασία έχει ένα ID**
- **κάθε διεργασία γνωρίζει τα IDs όλων των υπολοίπων**
- **αρχηγός εκλέγεται η διεργασία με το μέγιστο ID**
- **μια διεργασία μπορεί να αποτυγχάνει και να επανέρχεται οποτεδήποτε**

Initiator μπορεί να είναι

– **μια διεργασία που διαπιστώνει ότι ο τρέχων αρχηγός δεν είναι ενεργός (π.χ. έχει πολλή ώρα να λάβει μήνυμα από αυτόν)**

– **μια διεργασία που επανέρχεται**

Αλγόριθμος του Εξαναγκασμού



Έστω ότι Initiator είναι η διεργασία P

1. Αν η P έχει το μεγαλύτερο ID

– ανακηρύσσεται αρχηγός

– γνωστοποιεί το ID της στις διεργασίες με μικρότερο ID

2. Αν η P δεν έχει το μεγαλύτερο ID

– Στέλνει σε όλες τις διεργασίες με **μεγαλύτερο ID** από αυτήν ένα μήνυμα **<election>**

– Περιμένει ένα μήνυμα **<OK>** από αυτές τις διεργασίες

– Αν μέσα σε ένα συγκεκριμένο χρονικό όριο

Δεν λάβει καμία απάντηση <OK>

- **ανακηρύσσεται αρχηγός** ***/*όλες οι διεργασίες με μεγαλύτερο id έχουν αποτύχει***

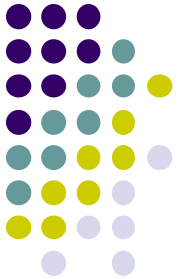
- το γνωστοποιεί στις διεργασίες με **μικρότερο ID**

Λάβει κάποια απάντηση <OK>

- περιμένει το ID του αρχηγού

- Αν μέσα σε ένα χρονικό όριο δεν λάβει το ID του αρχηγού, ξαναστέλνει μήνυμα **<election>**

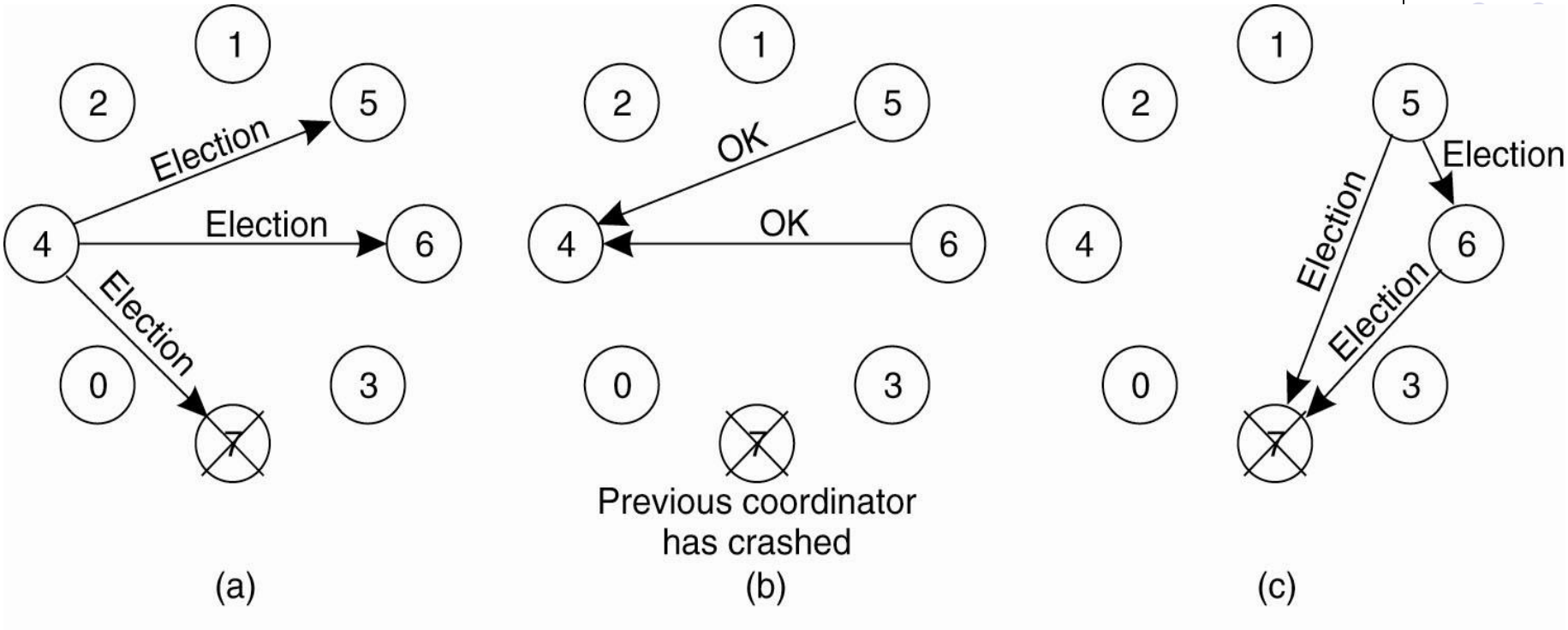
Αλγόριθμος του Εξαναγκασμού



Οι Non-initiators διεργασίες

- Μια διεργασία με ID μεγαλύτερο από αυτό της P λαμβάνει το μήνυμα **<election>**
- στέλνει απάντηση **<OK>** στην P
- παίρνει το ρόλο του **initiator**

Παράδειγμα Εκτέλεσης του Αλγορίθμου του Εξαναγκασμού (1)

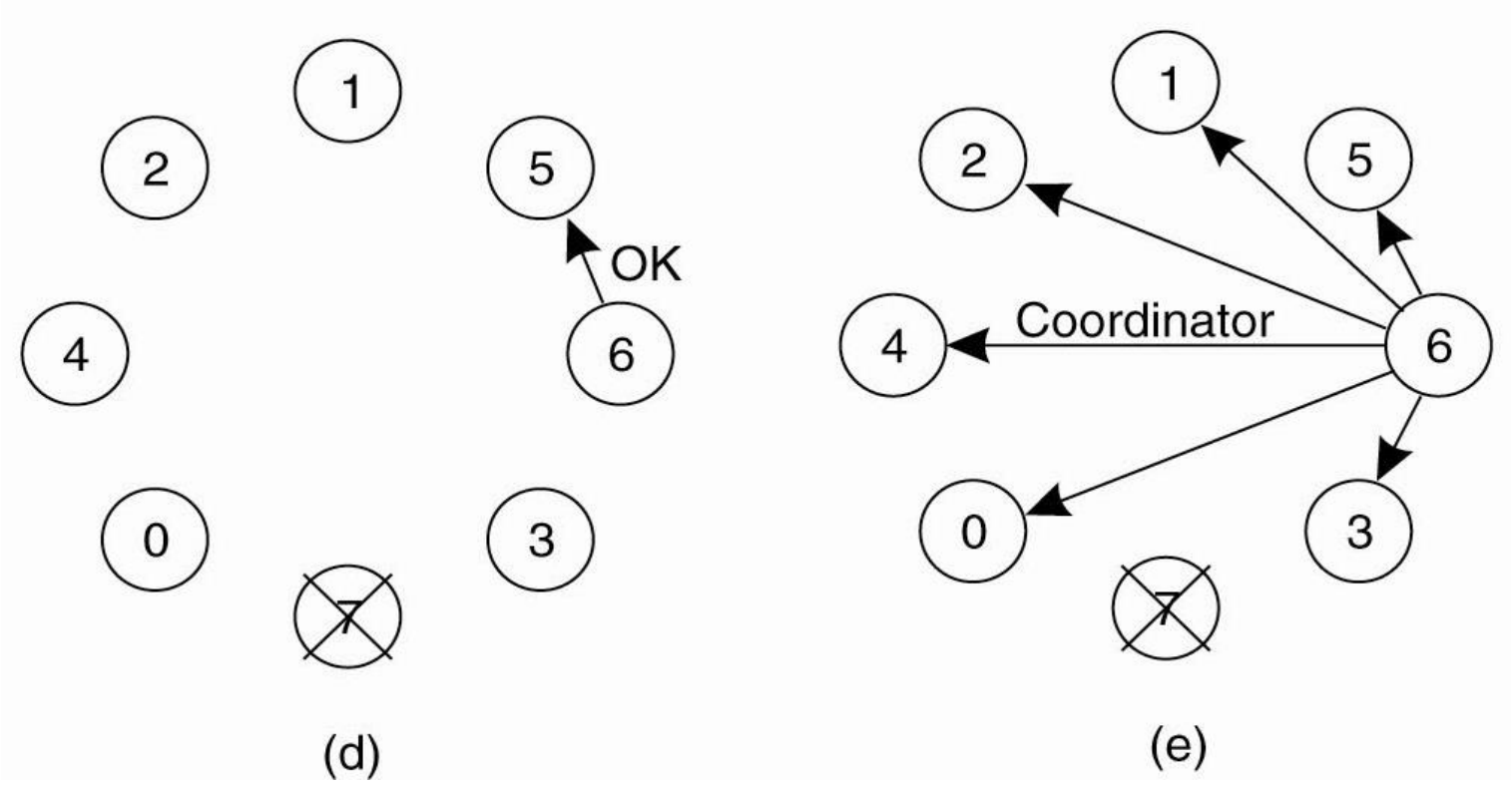


(a) **Process 4 first notices 7 is down and holds an election.**

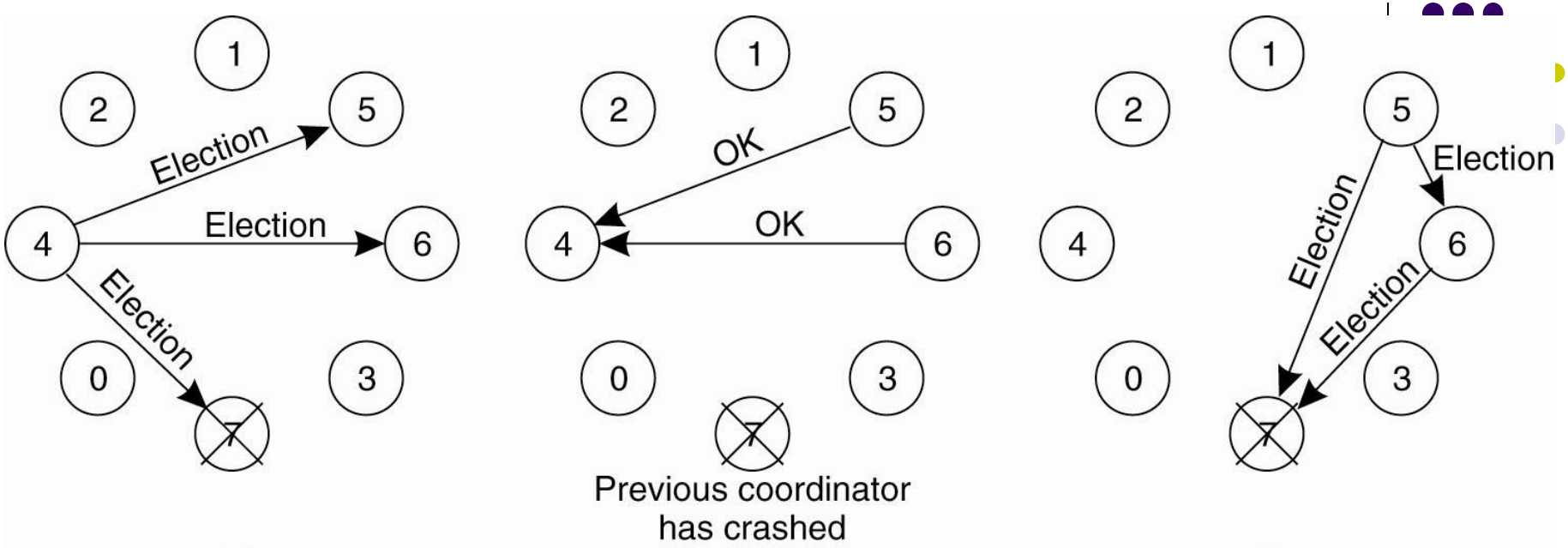
(b) **Processes 5 and 6 respond, telling 4 to stop.**

(c) **Now 5 and 6 each hold an election.**

Παράδειγμα Εκτέλεσης του Αλγορίθμου του Εξαναγκασμού (2)



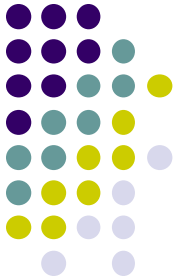
(d) Process 6 tells 5 to stop.
(e) Process 6 wins and tells everyone.


1
2
5
Election
4
Election
6
Election
0
3
7
1
2
5
OK
4
OK
6
0
3
7
Previous coordinator
has crashed
1
2
5
4
6
0
3
7
Election
Election
Election
1
2
5
4
6
0
3
7
OK
(d)
1
2
5
4
Coordinator
6
0
3
7
(e)

Μειονεκτήματα αλγορίθμου

- **Μεγάλος αριθμός μηνυμάτων - $O(n^2)$: n number of elections can take place in the system at the same time.**
- **Περισσότεροι από ένας αρχηγοί: If there is no guarantee on message delivery, two processes may declare themselves as a coordinator at the same time. E.g., P initiates an election and doesn't get any reply message from R (R has a higher ID than P). P announces itself as a coordinator and R also.**
- **Έναρξη διαδικασίας εκλογής χωρίς να χρειάζεται: If the coordinator is running unusually slowly (system is not working properly) or the link between a process and a coordinator is broken, any other process fails to detect the coordinator and initiates an election. But the coordinator is up, so in this case it is a redundant election.**

Αλγόριθμος του Εξαναγκασμού - Τροποποίηση



Έστω ότι Initiator είναι η διεργασία P

1. Αν η P έχει το μεγαλύτερο ID

– ανακηρύσσεται αρχηγός - γνωστοποιεί ID της στις διεργασίες με μικρότερο ID

2. Αν η P δεν έχει το μεγαλύτερο ID

– Στέλνει στη διεργασία **R** με το **μεγαλύτερο ID** ένα μήνυμα **<election>**

– Περιμένει ένα μήνυμα **<OK>** από αυτή τη διεργασία

– Αν μέσα σε ένα συγκεκριμένο χρονικό όριο

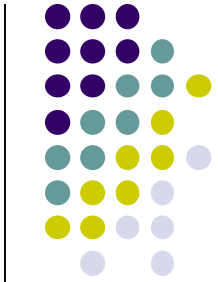
Δεν λάβει καμία απάντηση <OK>

- επαναλαμβάνει τη διαδικασία /*η R έχει αποτύχει

Λάβει κάποια απάντηση <OK>

- περιμένει το ID του αρχηγού /***η R ανακηρύσσεται αρχηγός**

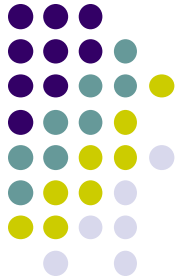
Εκλογή αρχηγού σε Γενικά Δίκτυα
Αλγόριθμος FloodMax



- Ο αλγόριθμος **FloodMax** έχει το προτέρημα της εύκολης υλοποίησης, και της **καλής απόδοσης** κάτω από ορισμένες συνθήκες χρονισμού του δικτύου (σύγχρονα συστήματα).
- Υποθέτει ότι οι διεργασίες **έχουν μοναδικές ταυτότητες** και οι διεργασίες **γνωρίζουν μόνο την δικιά τους** ταυτότητα.
- Υποθέτει ότι οι διεργασίες **συνθέτουν ένα συνεκτικό δίκτυο *n* διεργασιών και *m* καναλιών** (οποιασδήποτε τοπολογίας) και **γνωρίζουν τη διάμετρο *d*** του δικτύου.

- Ο αλγόριθμος **δεν απαιτεί** την ταυτόχρονη εκτέλεση των βημάτων από όλες τις διεργασίες.
- Ο αλγόριθμος βασίζεται σε λειτουργίες σύγκρισης των ταυτοτήτων των διεργασιών. Η βασική ιδέα του αλγορίθμου βρίσκεται στην επιβίωση και προώθηση της **μεγαλύτερης ταυτότητας** που υπάρχει στο δίκτυο. Στο τέλος της εκτέλεσης, κάθε διεργασία είναι σε θέση να γνωρίζει αυτή τη μέγιστη ταυτότητα.

Εκλογή αρχηγού σε Γενικά Δίκτυα
Αλγόριθμος FloodMax



Οι διεργασίες διατηρούν τις εξής μεταβλητές:

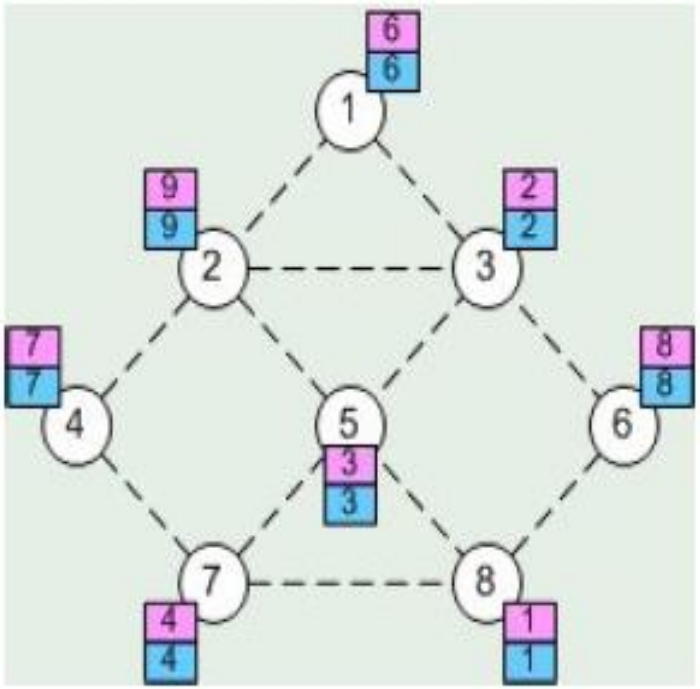
- την **μεταβλητή max_ID** με αρχική τιμή το ***ID*** της διεργασίας
- την **μεταβλητή *leader* ={*true,false*}** με αρχική τιμή ***false***

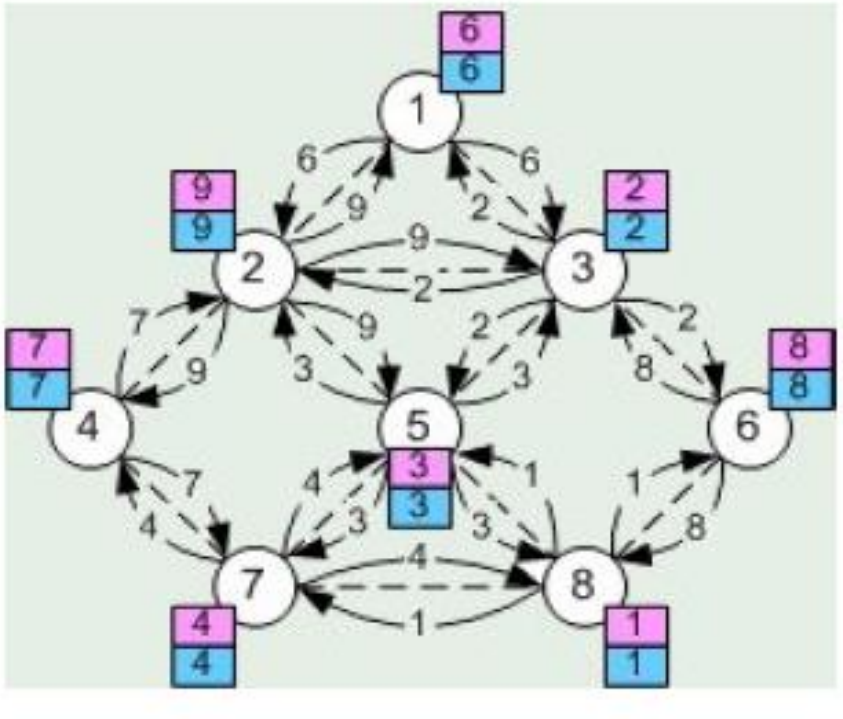
Αλγόριθμος FloodMax

- **Σε κάθε γύρο, οι διεργασίες εκπέμπουν την τιμή της max_ID σε όλους τους γείτονες.**
- **Μόλις λάβουν μία ταυτότητα από κάποιον γείτονα, την συγκρίνουν με την δική τους max_ID.**
- **Αν είναι μεγαλύτερη, θέτουν την μεταβλητή στην νέα τιμή.**
- **Μετά από *d* γύρους**

  αν η μεταβλητή max_ID ισούται με την ταυτότητα της διεργασίας, η διεργασία μεταβαίνει στην κατάσταση ισχύος θέτοντας την μεταβλητή *leader* στην τιμή *true*

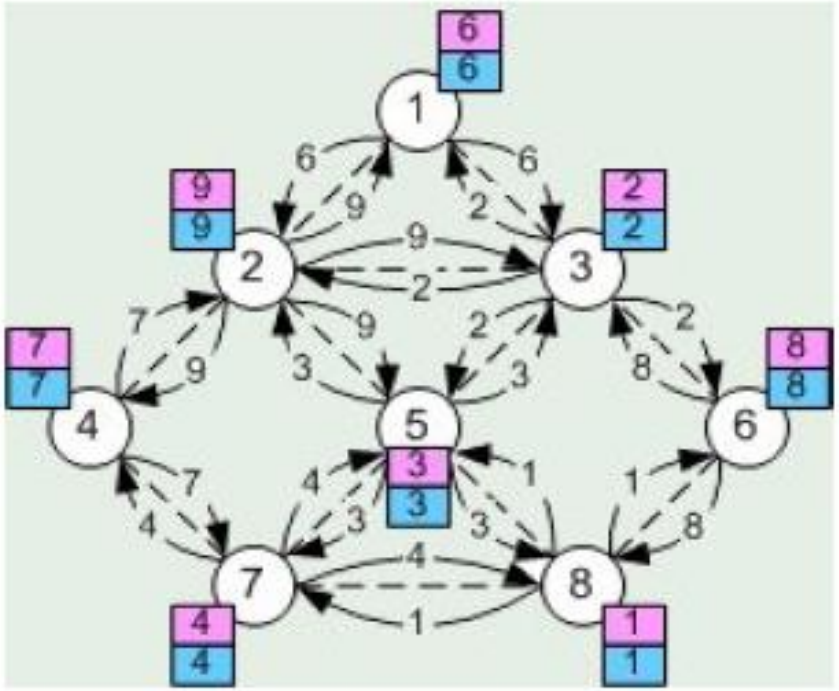
Παράδειγμα Εκτέλεσης του Αλγορίθμου FloodMax



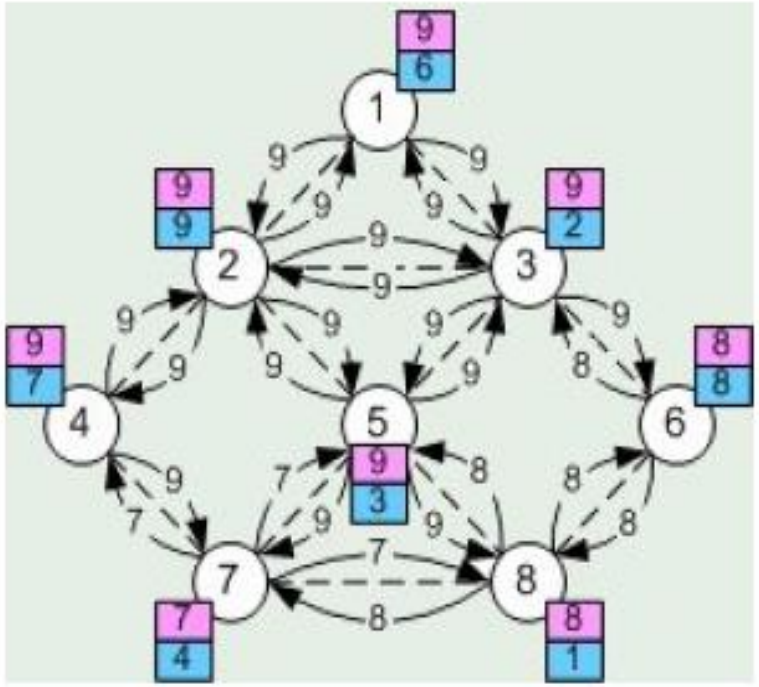


Πρώτος Γύρος

Παράδειγμα Εκτέλεσης του Αλγορίθμου FloodMax

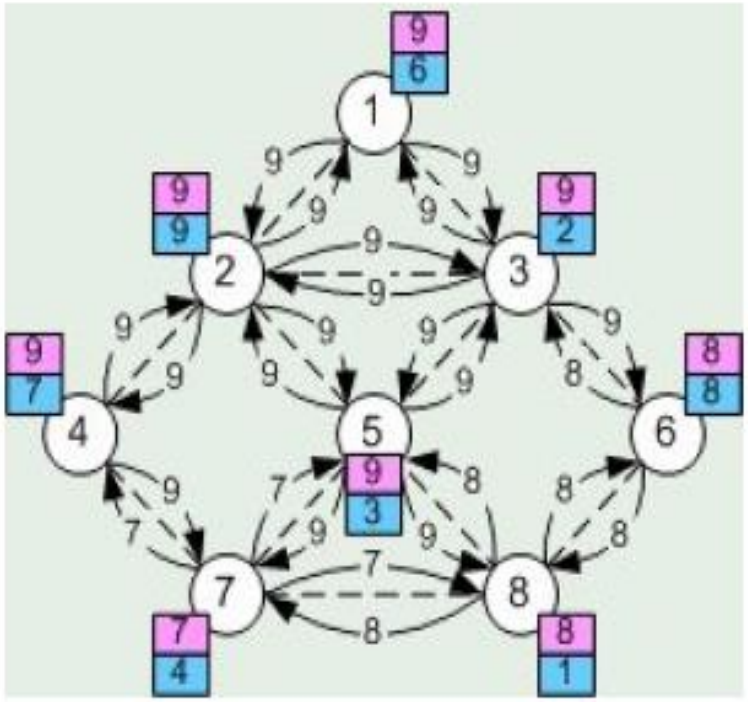


Πρώτος Γύρος

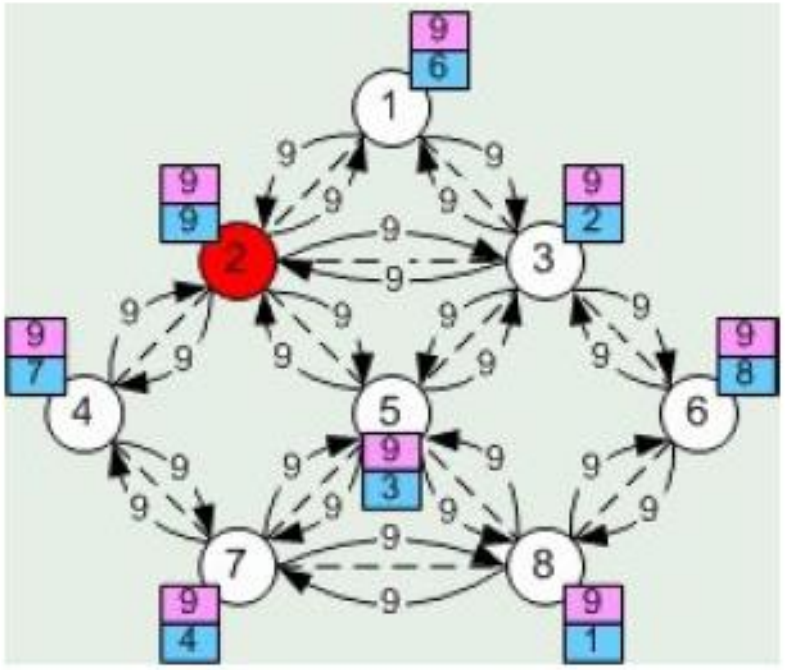


Δεύτερος Γύρος

Παράδειγμα Εκτέλεσης του Αλγορίθμου FloodMax

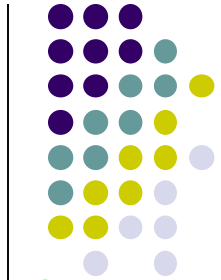


Δεύτερος Γύρος



Τρίτος Γύρος

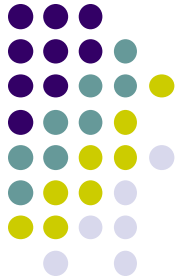
Τερματισμός Αλγορίθμου FloodMax



Συνήθως, μετά την εκλογή αρχηγού, όλες οι διεργασίες στο δίκτυο ενημερώνονται για την ταυτότητα του αρχηγού.

- Στον αλγόριθμο FloodMax αυτό επιτυγχάνεται στο τέλος του γύρου *d* όπου όλες οι διεργασίες θα έχουν αποθηκευμένη την ίδια max_ID. Θα γνωρίζουν δηλαδή τη διεργασία αρχηγό. Επίσης, για μία και μόνο διεργασία θα ισχύει ότι η max_ID θα ισούται με την ταυτότητα της διεργασίας.
- Στο τέλος, δηλαδή, της εκτέλεσης, θα υπάρχει μία διεργασία αρχηγός και όλες οι υπόλοιπες θα γνωρίζουν ποια είναι αυτή.

Απόδειξη ορθότητας του Αλγορίθμου FloodMax



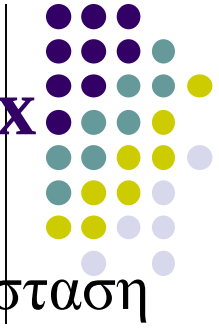
Έστω ID_k η ταυτότητα της διεργασίας k και ID_{max} η ταυτότητα της διεργασίας max με τη μεγαλύτερη ταυτότητα.

Τότε, στο τέλος του γύρου d η διεργασία max εκλέγεται αρχηγός και καμία άλλη διεργασία εκτός από την max δεν είναι σε κατάσταση *ισχύος*.

- Η χρονική πολυπλοκότητα είναι $O(d)$ - *είναι ο χρόνος που χρειάζεται η ταυτότητα της διεργασίας με τη μέγιστη ταυτότητα να φτάσει και στη πιο απομακρυσμένη διεργασία.*

- Η πολυπλοκότητα επικοινωνίας είναι $O(dm)$ - *Τα μηνύματα που ανταλλάσσονται εξαρτώνται από τον αριθμό m των καναλιών επικοινωνίας και τον αριθμό d των γύρων εκτέλεσης*

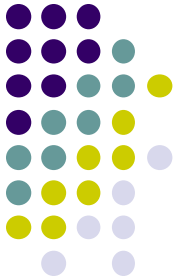
Απόδειξη ορθότητας του Αλγορίθμου FloodMax



Λήμμα: Η διεργασία με ταυτότητα ID_{max} τίθεται σε κατάσταση ισχύος θέτοντας *leader=true* στο τέλος του γύρου *d*.

Απόδειξη:
Η διαδικασία με ταυτότητα ID_{max} έχει αρχικά θέσει στη μεταβλητή max_ID την ταυτότητά της που ισούται με ID_{max} (έναρξη του αλγορίθμου).
Μετά απο *d* γύρους η διεργασία δεν έχει κάνει καμία αλλαγή στην max_ID καθώς για κάθε άλλη διεργασία ισχύει $ID_k < ID_{max}$. Έτσι μετά και από *d* γύρους για τη διεργασία με ταυτότητα ID_{max} θα ισχύει max_ID = ID_{max}. Επομένως, ισχύουν οι συνθήκες για να τεθεί σε κατάσταση ισχύος θέτοντας *leader=true*.

Εκλογή αρχηγού σε Γενικά Δίκτυα Αλγόριθμος OptFloodMax

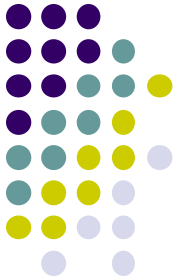


Βελτιώνει την απόδοση του FloodMax: Δεν πραγματοποιείται αποστολή μηνυμάτων στις γειτονικές διεργασίες που αφορά ήδη γνωστή ταυτότητα.

Οι διεργασίες διατηρούν τις εξής μεταβλητές:

- την **μεταβλητή max_ID** με αρχική τιμή το ***ID*** της διεργασίας
- την **μεταβλητή *leader ={true,false}*** με αρχική τιμή ***false***
- την **μεταβλητή *send ={true,false}*** με αρχική τιμή ***true***

Εκλογή αρχηγού σε Γενικά Δίκτυα
Αλγόριθμος OptFloodMax



Αλγόριθμος OptFloodMax

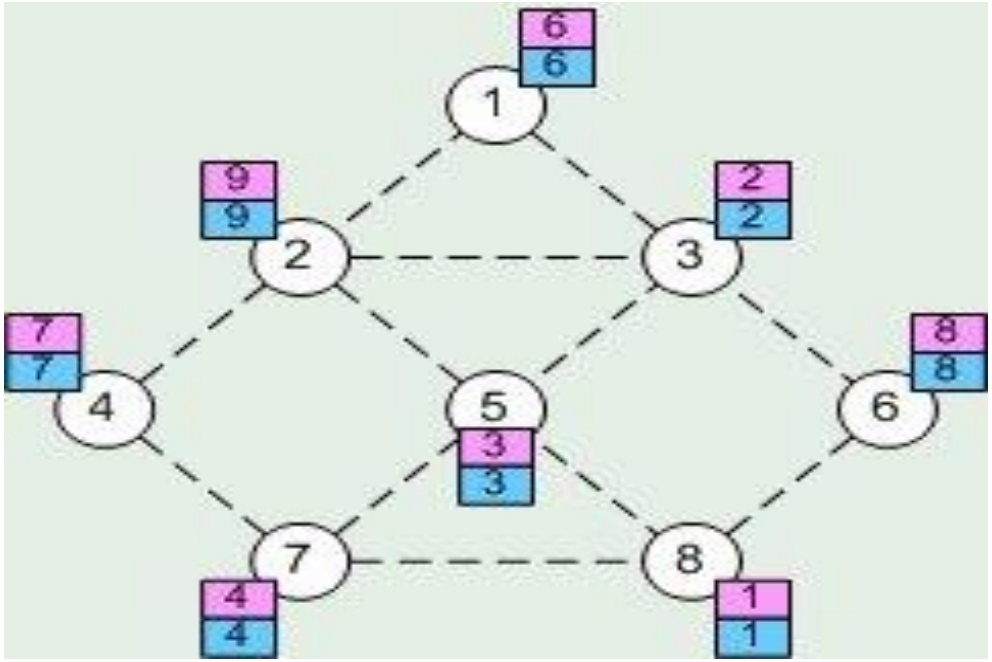
- **Στον πρώτο γύρο, οι διεργασίες εκπέμπουν την max_ID σε όλους τους γείτονες και θέτουν *send = false*.**

- **Μόλις λάβουν μία ταυτότητα από κάποιον γείτονα, την συγκρίνουν με την max_ID. Αν είναι μεγαλύτερη, θέτουν την μεταβλητή στην νέα τιμή και *send=true*.**

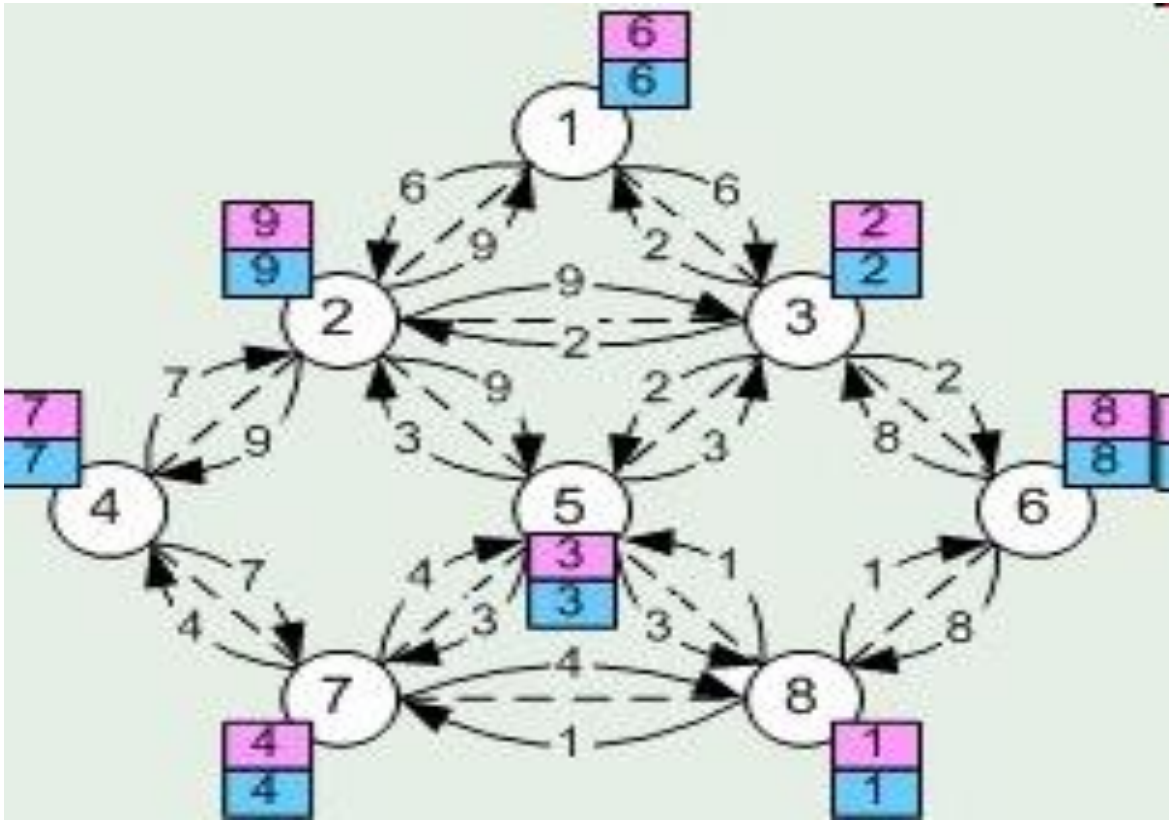
- **Σε κάθε γύρο πραγματοποιούν αποστολή μηνύματος αν *send = true*. Μετά το τέλος της αποστολής, κάθε διεργασία θέτει *send=false*.**

- **Μετά από *d* γύρους, αν η μεταβλητή ισούται με την ταυτότητα της διεργασίας, η διεργασία μεταβαίνει στην κατάσταση ισχύος θέτοντας *leader = true*.**

Παράδειγμα Εκτέλεσης του Αλγορίθμου OptFloodMax

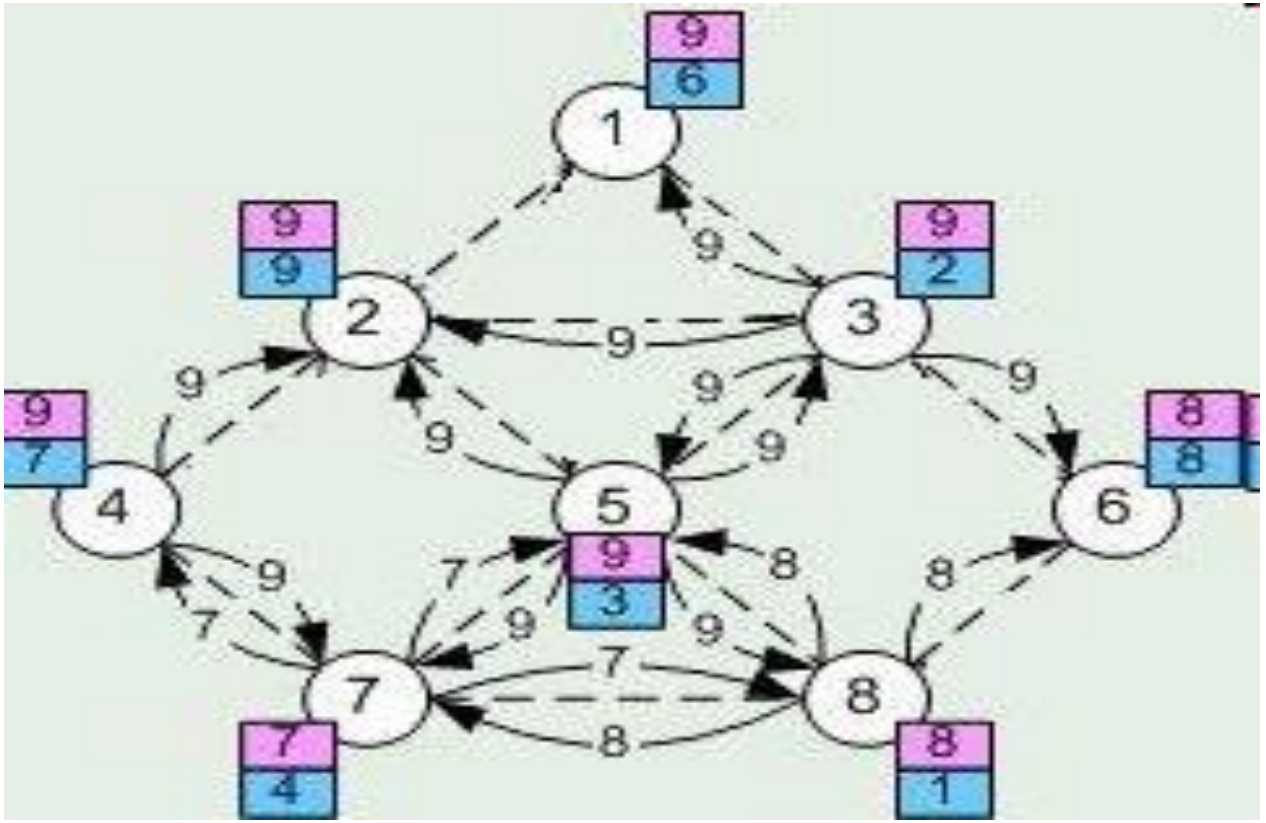


Παράδειγμα Εκτέλεσης του Αλγορίθμου OptFloodMax



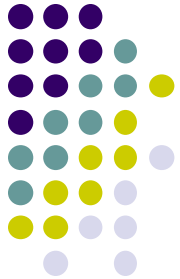
Πρώτος Γύρος

Παράδειγμα Εκτέλεσης του Αλγορίθμου OptFloodMax



Δεύτερος Γύρος

Παράδειγμα Εκτέλεσης του Αλγορίθμου OptFloodMax



- Στον τρίτο γύρο όλες οι διεργασίες έχουν ενημερωθεί για τη μέγιστη ταυτότητα. Η μόνη max_ID που κυκλοφορεί στο δίκτυο είναι η 9 που αντιστοιχεί στη διεργασία 2.

- Στο τέλος του γύρου *d* (δηλαδή στο τέλος του τρίτου γύρου), η διεργασία 2 (και μόνο αυτή) θέτει την μεταβλητή *leader=true*.

  Όλες οι άλλες διεργασίες θέτουν τις μεταβλητές *leader=false*.