Εισαγωγή

Μοντέλο

Βασικοί Αλγόριθμοι Γράφων

ΕΛΛΑΔΑ 2008 ΚΟΙΝΟΤΙΚΟ ΠΛΑΙΣΙΟ ΣΤΗΡΙΞΗΣ
Ανάπτυξη παντού. Ανάπτυξη για όλους.

ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ ΕΠΕΑΕΚ

ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Η ΠΑΙΔΕΙΑ ΣΤΗΝ ΚΟΡΥΦΗ
Επιχειρησιακό Πρόγραμμα
Εκπαίδευσης και Αρχικής
Επαγγελματικής Κατάρτισης

Κατανεμημένα Συστήματα

Ένα κατανεμημένο σύστημα είναι μια συλλογή από αυτόνομες διεργασίες οι οποίες έχουν τη δυνατότητα να επικοινωνούν μεταξύ τους.

Με βάση τον γενικό αυτό ορισμό, κατανεμημένα συστήματα είναι:

- ένα VLSI chip
- ένας στενά συνδεδεμένος πολυεπεξεργαστής με διαμοιραζόμενη μνήμη
- ένα τοπικό δίκτυο υπολογιστών
- το Internet

Περιοχές από όπου προέρχονται κλασικά προβλήματα στον κατανεμημένο υπολογισμό

- Λειτουργικά Συστήματα
- Δίκτυα Υπολογιστών
- Πολυ-επεξεργαστικά συστήματα
- Κατανεμημένες βάσεις δεδομένων
- Ανάπτυξη λογισμικού ανθεκτικού σε σφάλματα

Χαρακτηριστικά στα οποία οι κατανεμημένοι αλγόριθμοι διαφέρουν

- Μέθοδος διαδιεργασιακής επικοινωνίας
- Μοντέλο χρονισμού
- Μοντέλο αποτυχιών
- Προβλήματα που μελετώνται

Μη ντετερμινισμός

Ένας κατανεμημένος αλγόριθμος μπορεί να συμπεριφέρεται με διαφορετικό τρόπο σε διαφορετικές εκτελέσεις ακόμη και αν η είσοδος είναι ίδια. Οι λόγοι είναι ακόλουθοι:

- Πολλές διεργασίες με διαφορετικές ταχύτητες εκτελούνται ταυτόχρονα, ξεκινώντας πιθανόν διαφορετικές χρονικές στιγμές.
- Άγνωστοι χρόνοι παράδοσης μηνυμάτων
- Άγνωστη σειρά παράδοσης μηνυμάτων
- Αποτυχίες διεργασιών και συνδέσμων

Μετρικά Πολυπλοκότητας

- Χρόνος
- Μνήμη που απαιτείται από κάθε διεργασία
- Κόστος επικοινωνίας (αριθμός και μέγεθος μηνυμάτων, αριθμός και μέγεθος κοινών μεταβλητών)
- Αριθμός αποτυχημένων συνιστωσών

Μοντέλα Χρονισμού

- **Σύγχρονο Μοντέλο:** η εκτέλεση εξελίσσεται σε σύγχρονους γύρους (rounds).

- **Ασύγχρονο Μοντέλο:** οι διάφορες διεργασίες εκτελούνται με αυθαίρετη σειρά και με αυθαίρετες ταχύτητες.
- Σύστημα διαμοιραζομένης μνήμης
- Σύστημα μεταβίβασης-μηνύματος

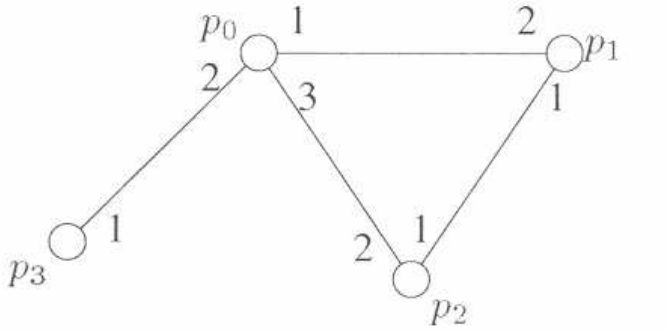
- **Μερικώς Σύγχρονο Μοντέλο:** υπάρχουν κάποιοι περιορισμοί στην σχετικό χρονισμό των γεγονότων, αλλά δεν είναι τόσο αυστηροί ώστε η εκτέλεση να εξελίσσεται σε γύρους (όπως στο σύγχρονο μοντέλο). Π.χ., μπορούν να υπάρχουν άνω και κάτω φράγματα για το χρόνο εκτέλεσης κάθε εντολής.

Μοντέλα Αποτυχίας

- **Αποτυχίες Κατάρρευσης:** διεργασίες μπορούν να σταματούν να εκτελούνται από κάποιο σημείο και μετά χωρίς προειδοποίηση.

- **Βυζαντινές Αποτυχίες:** διεργασίες μπορούν να συμπεριφέρονται με εντελώς αβίαστο τρόπο (δηλαδή να εκτελούν αυθαίρετο κώδικα που δεν σχετίζεται με τον αλγόριθμο και ακόμη να είναι εχθρικές).

Συστήματα Μεταβίβασης Μηνύματος



- n: αριθμός διεργασιών/επεξεργαστών (p_0, …, p_{n-1})
- Κάθε διεργασία μοντελοποιείται ως μια μηχανή καταστάσεων (state machine).
- Η κατάσταση π.χ., της διεργασίες p_0 αποτελείται από τις τοπικές της μεταβλητές, καθώς και από 6 πίνακες μηνυμάτων:
- $inbuf_0[1]$, …, $inbuf_0[3]$: μηνύματα που έχουν σταλεί στην p_0, τα οποία ωστόσο η p_0 δεν έχει ακόμη επεξεργαστεί.
- $outbuf_0[1]$, …, $outbuf_0[3]$: μηνύματα που έχουν σταλεί από την p0 σε κάθε μια από τις p_1, p_2, p_3, τα οποία δεν έχουν ακόμη παραδοθεί στις p_1, p_2, p_3.

Συστήματα Μεταβίβασης Μηνύματος

- *Κατάσταση πρόσβασης (access state)* μιας διεργασίας είναι η κατάστασή της χωρίς τους πίνακες outbuf.
- Κάθε διεργασία έχει μια αρχική κατάσταση στην οποία όλοι οι inbuf πίνακες είναι κενοί.
- Σε κάθε βήμα που εκτελείται από την p_0, η p_0 επεξεργάζεται όλα τα μηνύματα που βρίσκονται στους inbuf πίνακές της, η κατάστασή της p_0 αλλάζει, και αποστέλλεται το πολύ ένα μήνυμα προς κάθε γειτονική διεργασία.

Καθολικές Καταστάσεις (configurations)

Μια καθολική κατάσταση είναι ένα διάνυσμα που περιέχει τις καταστάσεις όλων των διεργασιών (μια κατάσταση για κάθε διεργασία).

Στις αρχικές καθολικές καταστάσεις, όλες οι διεργασίες είναι σε κάποια από τις αρχικές τους καταστάσεις.

Γεγονότα σε συστήματα μεταβίβασης μηνύματος

Δύο είδη γεγονότων για κάθε διεργασία:

- **Γεγονός αποστολής μηνύματος (deliver event)**: μετακίνηση ενός μηνύματος από τον outbuf του αποστολέα στον inbuf του παραλήπτη (del(k,j,m): αποστολή του μηνύματος m από την p_k στην p_j)

- **Γεγονός υπολογισμού (computational event)**: αλλαγή της τρέχουσας κατάστασης μιας διεργασίας εφαρμόζοντας τη συνάρτηση μετάβασης στην τρέχουσα κατάσταση πρόσβασής της (comp(i): η p_i εκτελεί ένα βήμα)

Ασύγχρονες Εκτελέσεις

Ένα *τμήμα εκτέλεσης* (execution fragment) είναι μια ακολουθία (πεπερασμένη ή άπειρη) από καθολικές καταστάσεις και γεγονότα που εναλλάσσονται: C_0, e_1, C_1, e_2, C_2, …, όπου κάθε e_i είναι γεγονός και κάθε C_i είναι καθολική κατάσταση, $i \geq 0$.

Για κάθε τριάδα (C_{i-1}, e_i, C_i), η C_i είναι ίδια με την C_{i-1} αλλά:

- Αν e_i = del(k, j, m): ένα μήνυμα μεταφέρεται από κάποιον από τους πίνακες outbuf της p_k σε κάποιον από τους πίνακες inbuf της p_j.
- Αν e_i = comp(j): εκτελείται ένα βήμα από την p_j.

Μια *εκτέλεση* (execution) είναι ένα τμήμα εκτέλεσης που όμως ξεκινά από μια αρχική καθολική κατάσταση.

Σύγχρονες Εκτελέσεις

Η ακολουθία από αλλεπάλληλες καθολικές καταστάσεις και γεγονότα πρέπει να μπορεί να χωριστεί σε σύγχρονους γύρους εκτέλεσης σε κάθε έναν από τους οποίους όλοι οι κόμβοι εκτελούν ένα βήμα.

Ένας γύρος αποτελείται από την παράδοση όλων των εν αναμονή μηνυμάτων και από την εκτέλεση ενός βήματος από κάθε διεργασία (με οποιαδήποτε σειρά).

Χρονοδιάγραμμα γεγονότων

Το *χρονοδιάγραμμα* μιας εκτέλεσης είναι η ακολουθία από γεγονότα e_1, e_2, … της εκτέλεσης.

Επιτρεπτές Εκτελέσεις (admissible executions)

- **Ιδιότητες Σιγουριάς (safety properties):** πρέπει να ισχύουν για κάθε πεπερασμένο πρόθεμα μιας εκτέλεσης (υποδεικνύουν πως τίποτα κακό δεν έχει συμβεί μέχρι την τρέχουσα χρονική στιγμή)

- **Ιδιότητες ενεργοποίησης (liveness propoerties):** πρέπει να ισχύουν μία ή περισσότερες φορές μετά από κάποιο σημείο της εκτέλεσης (υποδεικνύουν ότι κάτι καλό θα πρέπει να συμβεί κάποια χρονική στιγμή).

Οι εκτελέσεις πρέπει να αποτελούνται από «σωστές ενέργειες». Οι «σωστές ενέργειες» ορίζονται βάσει των ιδιοτήτων σιγουριάς και ενεργοποίησης.

Δικαιοσύνη

- Σε μια άπειρου μήκους εκτέλεση, σε κάθε διεργασία πρέπει να δίνεται η δυνατότητα να εκτελέσει ένα ή περισσότερα βήματα άπειρο αριθμό φορών. Κάθε φορά που αυτό συμβαίνει, η διεργασία είτε μπορεί να εκτελέσει ένα ακόμη βήμα (αν ο αλγόριθμός της το επιτρέπει) ή όχι στην οποία περίπτωση δεν λαμβάνει χώρα καμία ενέργεια.
- Σε μια πεπερασμένη ακολουθία δεν θα πρέπει να υπάρχουν διεργασίες οι οποίες να μπορούν να εκτελέσουν και άλλα βήματα.

Επιτρεπτές Εκτελέσεις (admissible executions)

- Στα ασύγχρονα συστήματα, μια εκτέλεση είναι επιτρεπτή (admissible) αν είναι δίκαιη και ισχύει πως οι διεργασίες έχουν παραλάβει και επεξεργαστεί όλα τα μηνύματα που έχουν αποσταλεί (δεν υπάρχουν μηνύματα στους πίνακες inbuf και outbuf στο τέλος της εκτέλεσης).

Μετρικά Πολυπλοκότητας – Συστήματα Μεταβίβασης Μηνύματος

Πολυπλοκότητα επικοινωνίας (message complexity)

Μέγιστος αριθμός μηνυμάτων που αποστέλλονται κατά τη διάρκεια της εκτέλεσης.

Χρονική Πολυπλοκότητα (time complexity)

Μέγιστος χρόνος που απαιτείται μέχρι να επιτευχθεί τερματισμός στην εκτέλεση.

Υποθέτουμε ότι ο χρόνος που σπαταλιέται μεταξύ της αποστολής και της παραλαβής ενός μηνύματος είναι το πολύ μια χρονική μονάδα. Με άλλα λόγια, εφαρμόζουμε κανονικοποίηση κάθε εκτέλεσης έτσι ώστε η μεγαλύτερη καθυστέρηση μηνύματος να είναι μια χρονική μονάδα. Η αυθαίρετη ανάμιξη των βημάτων των διεργασιών εξακολουθεί να είναι δυνατή.

Παράδειγμα

Εκπομπή (broadcast) πολλαπλών αποδεκτών δεδομένου ενός δένδρου επικάλυψης με ρίζα (rooted spanning tree)

Ένας κόμβος p_r θέλει να στείλει μήνυμα M σε όλους τους άλλους κόμβους.

Περιγραφή:

- Ο p_r αρχικά στέλνει το M στα παιδιά του
- Όταν ένας επεξεργαστής λαμβάνει το M από τον γονικό κόμβο, το στέλνει στα παιδιά του και τερματίζει.

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 ⟨M⟩ from parent:
4:      send ⟨M⟩ to all children
5:      terminate
```

Κατάσταση μιας διεργασίας p_i, $i \in \{0, \ldots, n-1\}$

- μια μεταβλητή $parent_i$
- μια μεταβλητή $children_i$
- μια μεταβλητή $terminated_i$
- οι πίνακες inbuf και outbuf της διεργασίας

Αρχική κατάσταση

- Όλες οι terminated μεταβλητές είναι false.
- Όλοι οι πίνακες inbuf είναι άδειοι για όλες τις διεργασίες.
- Οι πίνακες outbuf είναι άδειοι για όλες τις διεργασίες εκτός από την p_r, αλλά $outbuf_r[j]$ περιέχει το M για όλα τα $j \in children_r$.

Πολυπλοκότητες

Πολυπλοκότητα Επικοινωνίας;

Χρονική Πολυπλοκότητα;

Χρονική Πολυπλοκότητα

Σύγχρονο Μοντέλο

Λήμμα: Κάθε διεργασία σε απόσταση t από την p_r στο δένδρο επικάλυψης λαμβάνει το μήνυμα στον γύρο t.

Απόδειξη: Με επαγωγή στην απόσταση t μιας διεργασίας από την p_r.

t = 1. Κάθε παιδί της p_r λαμβάνει το μήνυμα από την p_r στον πρώτο γύρο.

Ας υποθέσουμε ότι ο ισχυρισμός ισχύει για t-1 ≥ 1: Κάθε διεργασία σε απόσταση t-1 από την p_r λαμβάνει το M στον γύρο t-1.

Έστω ότι η p είναι μια διεργασία σε απόσταση t από την p_r. Έστω ότι p' είναι η γονική διεργασία στο δένδρο επικάλυψης. Τότε, η p' είναι σε απόσταση t-1 από την p_r. Από την επαγωγική υπόθεση, η p' λαμβάνει το M στο γύρο t-1. Άρα, η p λαμβάνει από την p' το M στον επόμενο γύρο.

Ασύγχρονο Μοντέλο

Λήμμα: Κάθε διεργασία σε απόσταση t από την p_r στο δένδρο επικάλυψης λαμβάνει το μήνυμα M μέχρι τη χρονική στιγμή t.

Απόδειξη:

t=1. Όλα τα παιδιά της p_r λαμβάνουν το M μέχρι το τέλος της χρονικής στιγμής 1.

Ας υποθέσουμε (επαγωγικά) ότι ο ισχυρισμός ισχύει για $t\text{-}1 \geq 1$.

Έστω ότι p είναι μια διεργασία σε απόσταση t από την p_r. Έστω ότι p' είναι η γονική διεργασία στο δένδρο επικάλυψης. Τότε, η p' είναι σε απόσταση t-1 από την p_r. Από την επαγωγική υπόθεση, η p' λαμβάνει το M μέχρι τη χρονική στιγμή t-1. Άρα, η p λαμβάνει από την p' το M μέχρι τη χρονική στιγμή t.

Θεώρημα: Υπάρχει ασύγχρονος αλγόριθμος εκπομπής πολλαπλών αποδεκτών με πολυπλοκότητα επικοινωνίας n-1 και χρονική πολυπλοκότητα d, όταν είναι γνωστό ένα δένδρο επικάλυψης του γράφου των διεργασιών με βάθος d.

Δημιουργία ενός δένδρου επικάλυψης

Πως θα μπορούσε να τροποποιηθεί ο flooding ώστε να υπολογίζεται ένα δένδρο επικάλυψης;

Αλγόριθμος F-Spanning Tree

- Η p_r στέλνει το μήνυμά της σε όλους τους γείτονες της

- Όταν μια διεργασία p_i λαμβάνει το M για πρώτη φορά από ενδεχόμενα περισσότερες από μία διεργασία, διαλέγει μια από αυτές 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 <reject> σε όλες τις άλλες διεργασίες από τις οποίες έλαβε το M (για $1^η$ φορά).

- Η p_i στέλνει το M 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, $0 \leq i \leq n-1$.

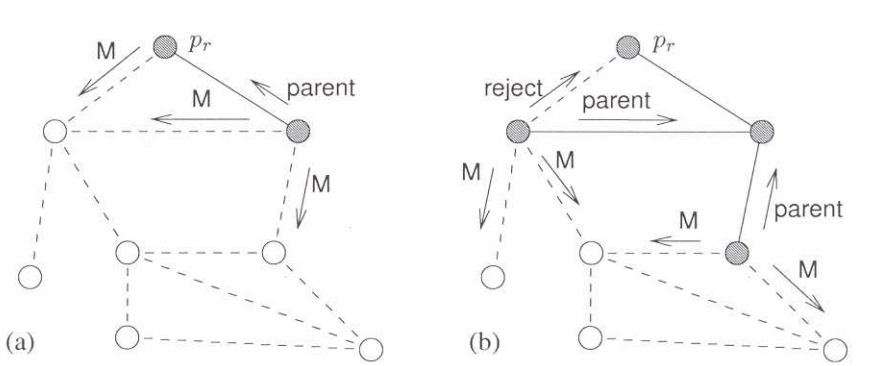
Initially $parent = \perp$, $children = \emptyset$, and $other = \emptyset$.

```
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
```

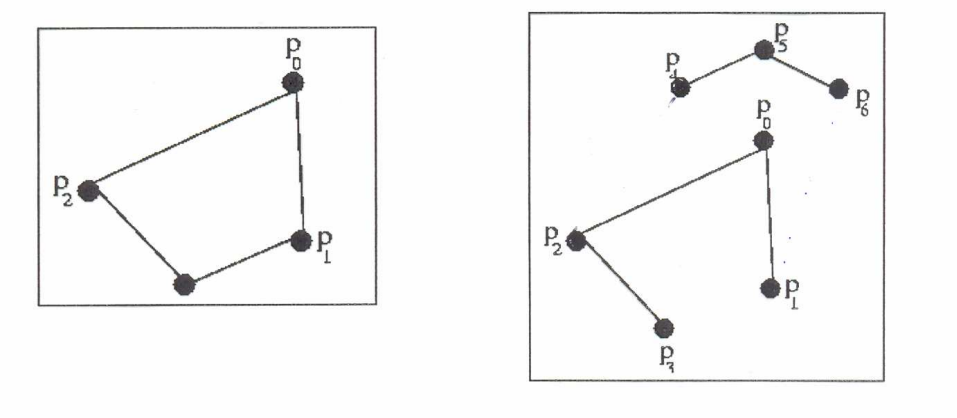


2 βήματα στην εκτέλεση του F-Spanning Tree

Ορθότητα

Γιατί δεν υπάρχει κύκλος;

Γιατί υπάρχει μονοπάτι από τη ρίζα προς κάθε άλλο κόμβο;



Τι συμβαίνει όταν το σύστημα είναι σύγχρονο;

Τι συμβαίνει όταν το σύστημα είναι ασύγχρονο;

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

Άτυπη Περιγραφή

- Κάθε κόμβος διατηρεί ένα σύνολο unexplored από «ανεξερεύνητους» γειτονικούς κόμβους και ένα σύνολο των κόμβων που θα αποτελέσουν παιδιά του στο δένδρο επικάλυψης που θα υπολογιστεί.
- Η ρίζα αρχικά στέλνει το M σε έναν από τους γείτονές της τον οποίο διαγράφει από το σύνολο unexplored.
- Όταν ένας κόμβος p_i λάβει το M για πρώτη φορά από κάποιο κόμβο p_j, ο p_i σημειώνει τον p_j ως τον πατρικό του κόμβο στο δένδρο επικάλυψης και του στέλνει ένα μήνυμα τύπου <parent>. Στη συνέχεια, επιλέγει έναν από ανεξερεύνητους γείτονες του (δηλαδή έναν από τους κόμβους του unexplored) και του προωθεί το μήνυμα. Αν ο p_i δεν λαμβάνει το M για πρώτη φορά, στέλνει ένα μήνυμα τύπου <reject> στον αποστολέα του μηνύματος και τον διαγράφει από το unexplored.
- Όταν ένας κόμβος λάβει μήνυμα τύπου <parent> ή <reject>, στέλνει το μήνυμα σε έναν από ακόμη ανεξερεύνητους γείτονές του. Αν έχει λάβει το M ή μήνυμα τύπου <parent> ή <reject> από όλους τους γείτονές του, ο κόμβος τερματίζει.

Δημιουργία ενός DFS Δένδρου Επικάλυψης δεδομένου του κόμβου ρίζα

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

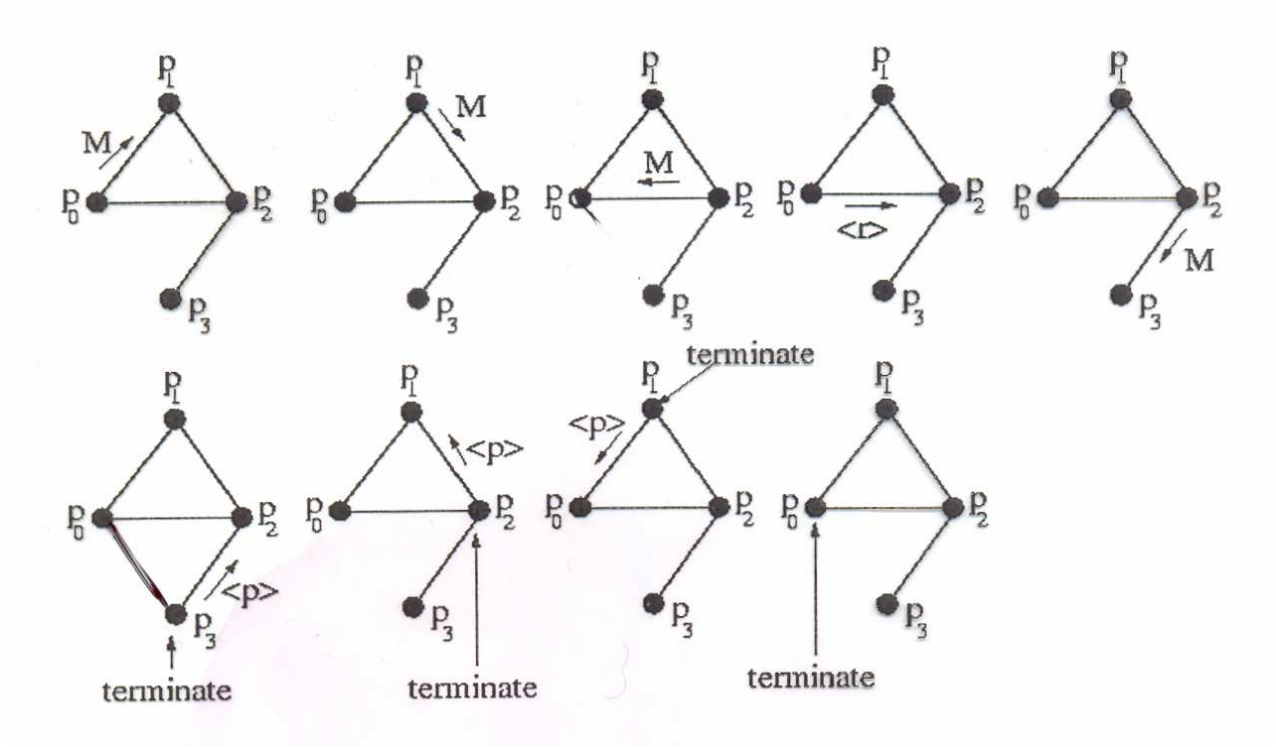
```
1:  upon receiving no message:
2:      if p_i = p_r and parent = ⊥ then                    // root wakes up
3:      parent := p_i
4:      explore()

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-ST Αλγορίθμου



$unexplored_0 = \{2\}$ $parent_0 = nil$ $children_0 = \{\}$

$unexplored_1 = \{0,2\}$ $parent_1 = nil$ $children_1 = \{\}$

$unexplored_2 = \{0,1,2\}$ $parent_2 = nil$ $children_2 = \{\}$

$unexplored_3 = \{2\}$ $parent_3 = nil$ $children_3 = \{\}$

Ανάλυση

Ορθότητα

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

Πολυπλοκότητα Επικοινωνίας

Λήμμα: Η πολυπλοκότητα επικοινωνίας του DFS-ST 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 4m μηνύματα.

Ανάλυση

Χρονική Πολυπλοκότητα

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.

Δημιουργία ενός DFS Δένδρου Επικάλυψης χωρίς να υπάρχει καθορισμένος κόμβος ρίζα

Έστω:

- p_m: ο κόμβος με το μεγαλύτερο αναγνωριστικό ανάμεσα στους κόμβους που ξυπνούν αυθόρμητα (και όχι επειδή έλαβαν κάποιο μήνυμα)
- m: το αναγνωριστικό του p_m

Παρατηρήσεις χρήσιμες για την απόδειξη ορθότητας του αλγορίθμου:

- Κόμβοι που θα λάβουν μήνυμα τύπου <leader> με αναγνωριστικό m δεν θα απορρίψουν το μήνυμα αφού δεν υπάρχει περίπτωση να έχουν λάβει άλλο μήνυμα με μεγαλύτερο αναγνωριστικό.
- Ομοίως, μηνύματα τύπου <already> με αναγνωριστικό m δεν θα απορριφθούν λόγω λάθος αναγνωριστικού
- Επίσης, μηνύματα τύπου <parent> με αναγνωριστικό m δεν θα απορριφθούν λόγω λάθος αναγνωριστικού
- Τέλος, μηνύματα με αναγνωριστικό m δεν θα απορριφθούν επειδή κάποιος κόμβος μπορεί να έχει τερματίσει.

Algorithm 4 Spanning tree construction: code for processor p_i, $0 \leq i \leq n-1$.

Initially *parent* $= \perp$, *leader* $= -1$, *children* $= \emptyset$, *unexplored* $=$ all neighbors of p_i

```
1:  upon receiving no message:
2:      if parent = ⊥ then                          // wake up spontaneously
3:          leader := id
4:          parent := p_i
5:          explore()

6:  upon receiving ⟨leader,new-id⟩ from p_j:
7:      if leader < new-id then                     // switch to new tree
8:          leader := new-id
9:          parent := p_j
10:         children := ∅
11:         unexplored := all neighbors of p_i except p_j
12:         explore()
13:     else if leader = new-id then
14:         send ⟨already,leader⟩ to p_j            // already in same tree
                // otherwise, leader > new-id and the DFS for new-id is stalled

15: upon receiving ⟨already,new-id⟩ from p_j:
16:     if new-id = leader then explore()

17: upon receiving ⟨parent,new-id⟩ from p_j:
18:     if new-id = leader then                     // otherwise ignore message
19:         add p_j to children
20:         explore()

21: procedure explore():
22:     if unexplored ≠ ∅ then
23:         let p_k be a processor in unexplored
24:         remove p_k from unexplored
25:         send ⟨leader,leader⟩ to p_k
26:     else
27:         if parent ≠ p_i then send ⟨parent,leader⟩ to parent
28:         else terminate as root of spanning tree
```

Εκλογή Αρχηγού σε Δακτύλιο

Το Πρόβλημα Εκλογής Αρχηγού

- Κάθε διεργασία πρέπει να αποφασίσει αν είναι ο αρχηγός ή όχι.
- Μία μόνο από τις διεργασίες θα πρέπει να αποφασίσει πως είναι ο αρχηγός.

Η διεργασία αρχηγός μπορεί να είναι υπεύθυνη για το συγχρονισμό μελλοντικών δραστηριοτήτων στο σύστημα:

- επανα-δημιουργία του αναγνωριστικού
- επαναφορά από αδιέξοδο
- να αποτελέσει τη διεργασία-ρίζα στη δημιουργία ενός δένδρου επικάλυψης

Το Πρόβλημα Εκλογής Αρχηγού

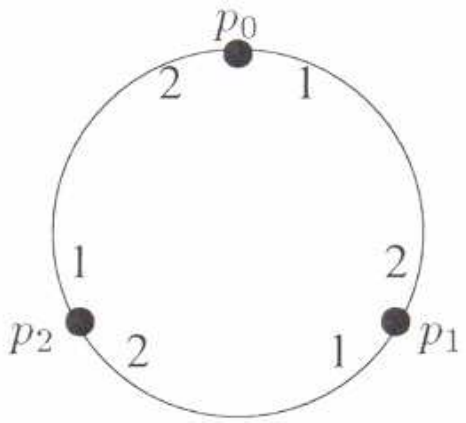
Φορμαλιστικά:

Υπάρχουν δύο είδη τερματικών καταστάσεων για μια διεργασία, η τερματική κατάσταση στην οποία η διεργασία έχει εκλεγεί αρχηγός (*κατάσταση ισχύος*) και η τερματική κατάσταση που η διεργασία δεν είναι ο αρχηγός (*κατάσταση μη-ισχύος*).

Επιτρεπτές Εκτελέσεις:

- Κάθε διεργασία τελικά εισέρχεται σε μια τερματική κατάσταση, ισχύος ή μη.
- Μόνο μια διεργασία, ο αρχηγός, μπαίνει σε τερματική κατάσταση ισχύος.

Δακτύλιοι



- Κάθε διεργασία γνωρίζει ποιος είναι ο αριστερός και ποιος ο δεξιός της γείτονας:
  - 1: αριστερά ή σύμφωνα με τους δείκτες του ρολογιού
  - 2: δεξιά ή αντίθετα με τους δείκτες του ρολογιού

- n: αριθμός διεργασιών στο σύστημα

Δακτύλιοι

Οι διεργασίες συνήθως έχουν μοναδικά αναγνωριστικά (τα οποία μπορούν να χρησιμοποιηθούν κατά το σχεδιασμό αλγορίθμων).

Αναγνωριστικό: αυθαίρετος ακέραιος (πιθανόν διαφορετικό από τα 0, .., n-1). Κάθε διεργασία p_i έχει μια μεταβλητή id_i που αποθηκεύει το αναγνωριστικό της (η μεταβλητή id_i είναι μέρος της κατάστασης κάθε διεργασίας)

Ανώνυμοι Δακτύλιοι

Οι διεργασίες δεν έχουν μοναδικά αναγνωριστικά.

Εκλογή Αρχηγού σε Ανώνυμους Δακτύλιους

Κάθε διεργασία στο σύστημα έχει την ίδια μηχανή καταστάσεων.

Αδυναμία Επίλυσης Προβλήματος σε Σύγχρονους Δακτύλιους

Διαισθητικά:

Η συμμετρία δεν μπορεί να καταστραφεί χωρίς την ύπαρξη κάποιας ασυμμετρίας των διεργασιών στην αρχή (π.χ., διαφορετικά αναγνωριστικά).

- A: Αλγόριθμος
- R: Δακτύλιος

Εκλογή Αρχηγού σε Ανώνυμους Δακτύλιους

Λήμμα: Σε κάθε γύρο k μιας εκτέλεσης του A στον R, οι καταστάσεις όλων των διεργασίων είναι ίδιες στο τέλος του γύρου k.

Απόδειξη: Με επαγωγή στο k.

k =0. Όλες οι διεργασίες έχουν την ίδια αρχική κατάσταση.

Υποθέτουμε ότι ο ισχυρισμός ισχύει επαγωγικά για $k\text{-}1 \geq 0$.

Αποδεικνύουμε τον ισχυρισμό για k:

- Στον γύρο k-1, όλες οι διεργασίες στέλνουν το ίδιο μήνυμα m_r στον δεξιό γείτονά τους και τον ίδιο μήνυμα m_l στον αριστερό γείτονά τους.
- Στον γύρο k, όλες οι διεργασίες λαμβάνουν τα ίδια μηνύματα.
- Δεδομένου ότι όλες εκτελούν τον ίδιο αλγόριθμο, βρίσκονται όλες στην ίδια κατάσταση στο τέλος του γύρου k.

Ισχύει το αρνητικό αυτό αποτέλεσμα για ασύγχρονους δακτύλιους;

Ισχύει το αρνητικό αυτό αποτέλεσμα αν το n είναι άγνωστο;

Ένας τέτοιος αλγόριθμος θα λειτουργούσε σωστά και σε σύγχρονους δακτύλιους. Άτοπο.

Ένας τέτοιος αλγόριθμος μπορεί να θεωρηθεί ως σωστός αλγόριθμος που επιλύει το πρόβλημα όταν το n είναι γνωστό (απλά η γνώση αυτή του n δεν χρησιμοποιείται). Άτοπο.

Ασύγχρονοι Δακτύλιοι

Ένας αλγόριθμος που προκαλεί την αποστολή $O(n^2)$ μηνυμάτων

Ενέργειες κάθε διεργασίας p:

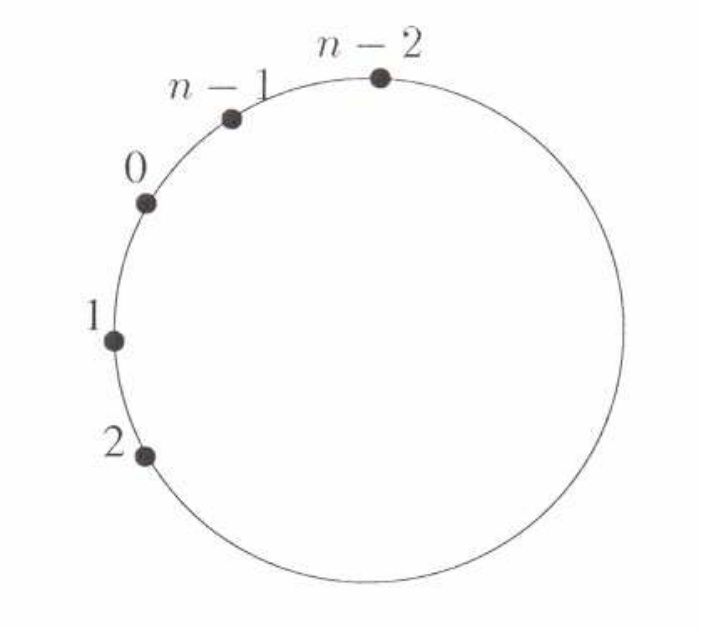
- Αποστολή του αναγνωριστικού του στα αριστερά.
- Όταν η p λάβει ένα αναγνωριστικό (από δεξιά) κάνει τα εξής:
  - αν είναι μεγαλύτερο από το δικό της, το προωθεί προς τα αριστερά
  - αν είναι μικρότερο από το δικό της, το αγνοεί (και δεν το προωθεί)
  - αν είναι ίσο με το δικό της, αποφασίζει πως αυτή είναι ο αρχηγός στο σύστημα και στέλνει ένα μήνυμα τερματισμού προς τα αριστερά
- Όταν η p λάβει μήνυμα τερματισμού, το προωθεί προς τα αριστερά και εισέρχεται σε τερματική κατάσταση μη-ισχύος.

Ασύγχρονοι Δακτύλιοι

Ένας αλγόριθμος που προκαλεί την αποστολή $O(n^2)$ μηνυμάτων

Πολυπλοκότητα Επικοινωνίας

Καμιά διεργασία δεν στέλνει περισσότερα από n μηνύματα.
Υπάρχει εκτέλεση στην οποία αποστέλλονται $\Theta(n^2)$ μηνύματα;



Ένας Αλγόριθμος που προκαλεί την αποστολή O(nlogn) μηνυμάτων

k-γειτονιά μιας διεργασία p_i: το σύνολο των διεργασιών που βρίσκονται σε απόσταση το πολύ k από την p_i στο δακτύλιο (είτε προς τα αριστερά ή προς τα δεξιά).

Περιγραφή Αλγορίθμου

- Λειτουργεί σε φάσεις.
- Στην k-οστή φάση, ένας επεξεργαστής προσπαθεί να γίνει ο προσωρινός αρχηγός της 2^k-γειτονιάς του.
- Μόνο οι επεξεργαστές που εκλέγονται αρχηγοί στην k-οστή φάση θα συνεχίσουν στην (k+1)-οστή φάση.

Ένας Αλγόριθμος που προκαλεί την αποστολή O(nlogn) μηνυμάτων

Περιγραφή k-οστής Φάσης

- Κάθε διεργασία p_i, που εκλέχθηκε προσωρινός αρχηγός στην (k-1)-οστή φάση, στέλνει μηνύματα τύπου <probe> με το αναγνωριστικό της σε όλους τους κόμβους στην 2^k-γειτονιά της.
- Μια διεργασία αγνοεί ένα μήνυμα τύπου <probe> (δεν το αναμεταδίδει), αν αυτό περιέχει ένα αναγνωριστικό που είναι μικρότερο από το δικό της.
- Όταν ένα μήνυμα τύπου <probe> φθάσει στην τελευταία διεργασία στη τρέχουσα γειτονιά, τότε αυτή η διεργασία στέλνει στην p_i ένα μήνυμα τύπου <reply>.
- Αν η p_i λάβει το <reply> και από τις δύο κατευθύνσεις, αποφασίζει πως είναι ο αρχηγός της 2^k-γειτονιάς της στη φάση k.
- Η διεργασία που θα λάβει το δικό της μήνυμα τύπου <probe>, τερματίζει σε κατάσταση ισχύος (στέλνοντας μήνυμα τερματισμού στις υπόλοιπες).

Algorithm 5 Asynchronous leader election: code for processor p_i, $0 \leq i < n$.

Initially, *asleep* = true

```
1:  upon receiving no message:
2:      if asleep then
3:          asleep := false
4:          send ⟨probe,id,0,1⟩ to left and right

5:  upon receiving ⟨probe,j,k,d⟩ from left (resp., right):
6:      if j = id then terminate as the leader
7:      if j > id and d < 2^k then                          // forward the message
8:          send ⟨probe,j,k,d + 1⟩ to right (resp., left)   // increment hop counter
9:      if j > id and d ≥ 2^k then                          // reply to the message
10:         send ⟨reply,j,k⟩ to left (resp., right)
                                          // if j < id, message is swallowed

11: upon receiving ⟨reply,j,k⟩ from left (resp., right):
12:     if j ≠ id then send ⟨reply,j,k⟩ to right (resp., left)   // forward the reply
13:     else                                                     // reply is for own probe
14:         if already received ⟨reply,j,k⟩ from right (resp., left) then
15:             send ⟨probe,id,k + 1,1⟩ to left and right !      // phase k winner
```

- Ένα μήνυμα τύπου <probe> περιέχει ένα αναγνωριστικό id, τον αριθμό της τρέχουσας φάσης k, και έναν μετρητή d (του μήκους του μονοπατιού που έχει ακολουθηθεί).
- Ένα μήνυμα τύπου <reply> περιέχει id και k.

Ανάλυση του Αλγορίθμου Εκλογής Αρχηγού που αποστέλλει O(nlogn) μηνύματα

Λήμμα: Για κάθε $k \geq 1$, ο αριθμός των επεξεργαστών που εκλέγονται αρχηγοί στη φάση k είναι το πολύ $n/(2^k+1)$.

Απόδειξη:

Δύο αρχηγοί της k-οστής φάσης θα πρέπει να έχουν ανάμεσά τους τουλάχιστον 2k διεργασίες.

Παρατηρήσεις

Υπάρχει μόνο ένας νικητής μετά από τουλάχιστον log(n-1) φάσεις.

Ο συνολικός αριθμός μηνυμάτων είναι:

$$5n + \text{Sum\_\{k=1\}\^{}\{\lfloor log(n-1) \rfloor+1\}} 4*2^k*n/(2^{k-1}+1) < 8n(logn+2) + 5n$$

Θεώρημα: Υπάρχει ασύγχρονος αλγόριθμος εκλογής αρχηγού του οποίου η πολυπλοκότητα επικοινωνίας είναι O(nlogn).