Εισαγωγή

Μοντέλο

Βασικοί Αλγόριθμοι Γράφων

ΕΛΛΑΔΑ
ΚΟΙΝΟΤΙΚΟ ΠΛΑΙΣΙΟ ΣΤΗΡΙΞΗΣ
2008
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

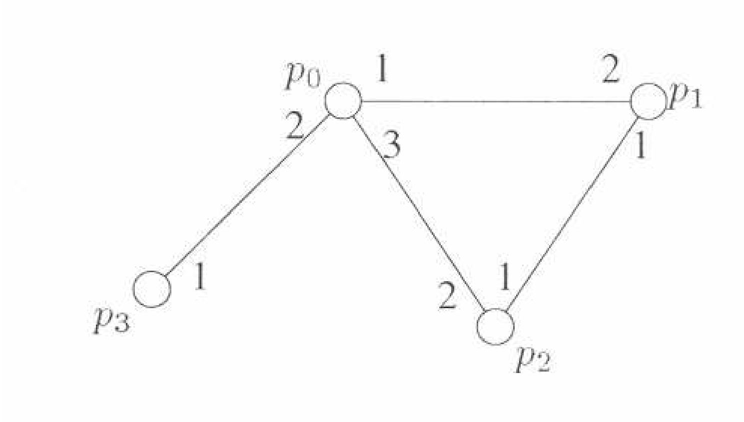
- **Μερικώς Σύγχρονο Μοντέλο:** υπάρχουν κάποιοι περιορισμοί στην σχετικό χρονισμό των γεγονότων, αλλά δεν είναι τόσο αυστηροί ώστε η εκτέλεση να εξελίσσεται σε γύρους (όπως στο σύγχρονο μοντέλο). Π.χ., μπορούν να υπάρχουν άνω και κάτω φράγματα για το χρόνο εκτέλεσης κάθε εντολής.

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 p_0 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 παραλαβής μηνύματος (deliver event)**: μετακίνηση ενός μηνύματος από τον outbuf του αποστολέα στον inbuf του παραλήπτη (del(k,j,m): το μήνυμα m που έχει σταλεί από την p_k φθάνει στην p_j και τοποθετείται στους inbuf της για επεξεργασία).

- **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 ⟨M⟩ is in transit from p_r to all its children in the spanning tree.

Code for p_r:
upon receiving no message:                    // first computation event by p_r
      terminate

Code for p_i, 0 ≤ i ≤ n − 1, i ≠ r:
upon receiving ⟨M⟩ from parent:
      send ⟨M⟩ to all children
      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 $t-1 \geq 1$: 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.

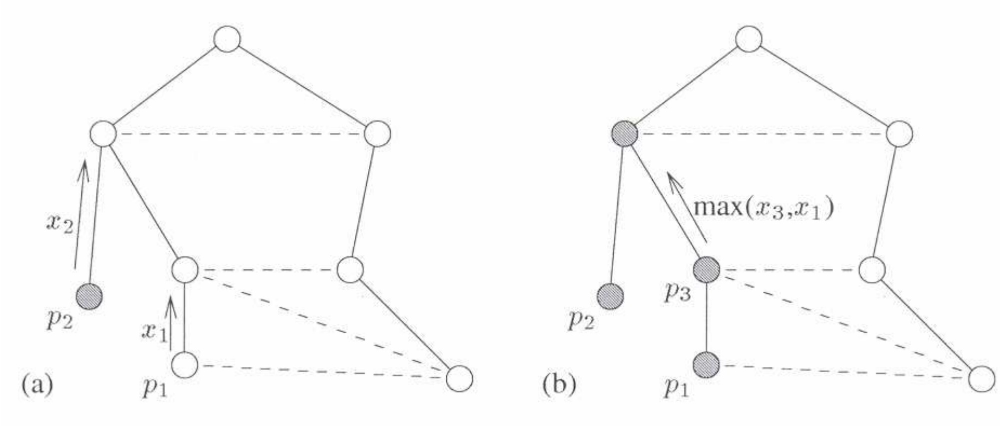
Ας υποθέσουμε (επαγωγικά) ότι ο ισχυρισμός ισχύει για $t-1 \geq 1$.

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.

Convergecast (συλλογή σε έναν κόμβο πληροφοριών που αποστέλλονται από τους υπόλοιπους κόμβους) δεδομένου ενός δένδρου επικάλυψης με ρίζα

- Συλλογή στη ρίζα πληροφοριών που αποστέλλονται από τους υπόλοιπους κόμβους.
- Ο κόμβος i ξεκινά με μια μεταβλητή x_i, στην οποία είναι αποθηκευμένη η πληροφορία του.
- Ζητείται να αποσταλεί η μέγιστη από τις τιμές των μεταβλητών αυτών στη ρίζα.



2 βήματα στην εκτέλεση του Convergcast αλγορίθμου

Θεώρημα: Υπάρχει ασύγχρονος convergecast αλγόριθμος 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 Μ 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 <already> σε όλες τις άλλες διεργασίες από τις οποίες έλαβε το Μ (για $1^η$ φορά).

- Η p_i στέλνει το Μ σε όλους τους γείτονές της και όταν λάβει απάντηση από αυτούς τερματίζει.

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

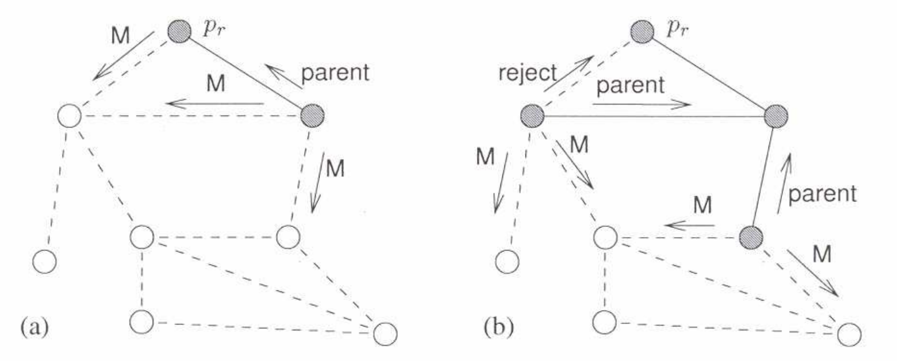
Initially *parent* $= \perp$, *children* $= \emptyset$, and *other* $= \emptyset$.

upon receiving no message:
    if $p_i = p_r$ and *parent* $= \perp$ then   // root has not yet sent ⟨M
        send ⟨M⟩ to all neighbors
        parent $:= p_i$

upon receiving ⟨M⟩ from neighbor p_j:
    if *parent* $= \perp$ then   // p_i has not received ⟨M⟩ befo
        parent $:= p_j$
        send ⟨parent⟩ to p_j
        send ⟨M⟩ to all neighbors except p_j
    else send ⟨already⟩ to p_j

upon receiving ⟨parent⟩ from neighbor p_j:
    add p_j to *children*
    if *children* $\cup$ *other* contains all neighbors except *parent* then
        terminate

upon receiving ⟨already⟩ from neighbor p_j:
    add p_j to *other*
    if *children* $\cup$ *other* contains all neighbors except *parent* then
        terminate

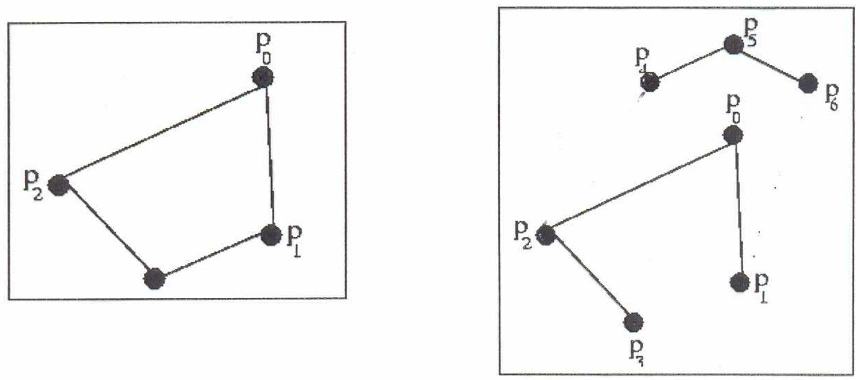


2 βήματα στην εκτέλεση του F-Spanning Tree

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;

Σύγχρονα Συστήματα

Ένα *κατευθυνόμενο δένδρο επικάλυψης* ενός κατευθυνόμενου γράφου G = (V,E) είναι ένα δένδρο στο οποίο ένας διακριτός κόμβος είναι ρίζα, το σύνολο ακμών είναι $\subseteq$ E και όλες οι ακμές κατευθύνονται από γονικούς κόμβους προς θυγατρικούς κόμβους.

Ένα κατευθυνόμενο δένδρο με ρίζα μια διεργασία i_0 ενός γράφου-διεργασιών G είναι *BFS* (Breath-First-Search) αν κάθε κόμβος σε απόσταση d από την i_0 στον G βρίσκεται σε βάθος d στο δένδρο.

Πως μπορούμε να σχεδιάσουμε ένα σύγχρονο αλγόριθμο που να υπολογίζει ένα BFS δένδρο του G δεδομένου ότι ο G είναι κατευθυνόμενος γράφος;

Πώς θα γίνει broadcast σε ένα κατευθυνόμενο γράφο;

Πως θα ενημερώνουν οι διάφοροι κόμβοι τους γονικούς τους κόμβους ότι είναι παιδιά τους;

Ποια είναι η πολυπλοκότητα του αλγορίθμου σε αυτή την περίπτωση;

Πως θα μάθει η i_0 ότι η κατασκευή του δένδρου έχει τελειώσει;

Ασύχρονα Συστήματα - 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 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 πατρικό του κόμβο στο δένδρο επικάλυψης. 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 Μ για πρώτη φορά, στέλνει ένα μήνυμα τύπου <already> στον αποστολέα του μηνύματος και τον διαγράφει από το unexplored. Αν το unexplored είναι κενό, ο p_i στέλνει ένα μήνυμα τύπου <parent> στον πατέρα του.
- Όταν ένας κόμβος λάβει μήνυμα τύπου <parent> ή <already>, στέλνει το μήνυμα σε έναν από τους ακόμη ανεξερεύνητους γείτονές του. Αν έχει λάβει το Μ ή μήνυμα τύπου <parent> ή <already> 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

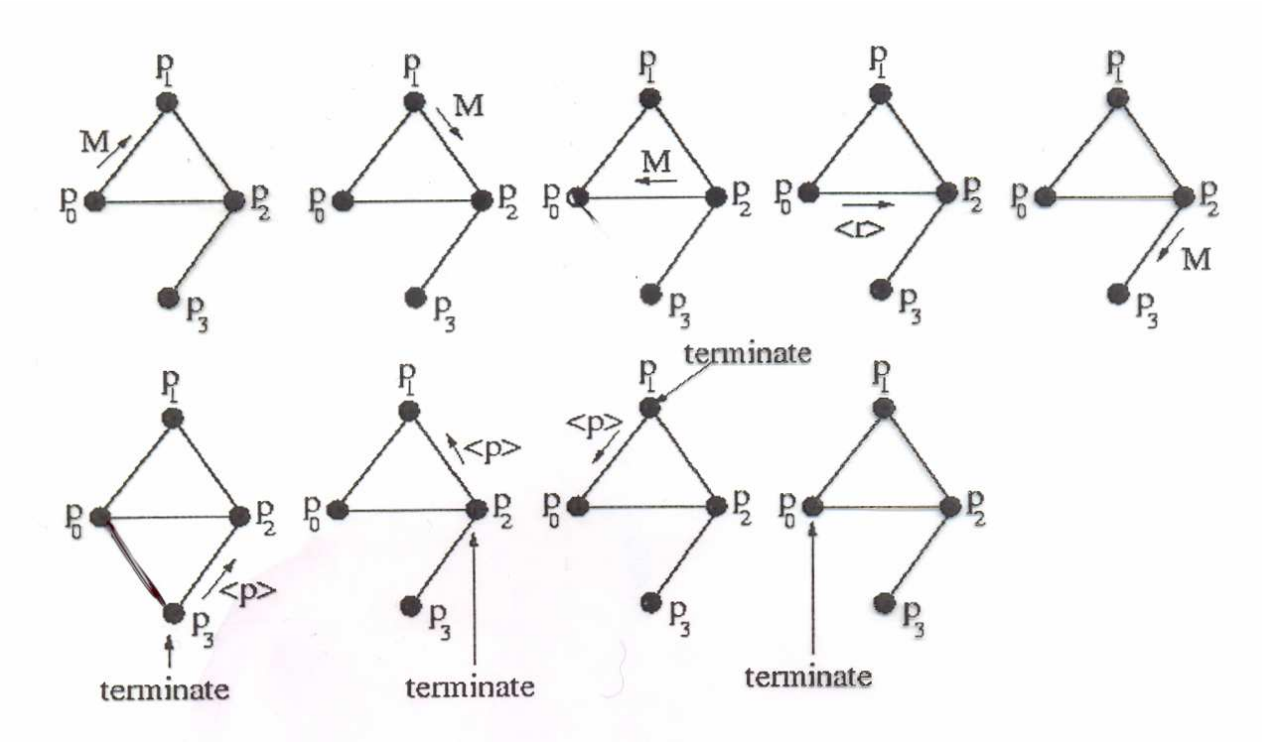
upon receiving no message:
    if $p_i = p_r$ and $parent = \perp$ then // root wakes up
    $parent := p_i$
    explore()

upon receiving ⟨M⟩ from p_j:
    if $parent = \perp$ then // p_i has not received ⟨M⟩ before
        $parent := p_j$
        remove p_j from *unexplored*
        explore()
    else
        send ⟨already⟩ to p_j // already in tree
        remove p_j from *unexplored*
upon receiving ⟨already⟩ from p_j:
    explore()

upon receiving ⟨parent⟩ from p_j:
    add p_j to *children*
    explore()

procedure explore():
    if $unexplored \neq \emptyset$ then
        let p_k be a processor in *unexplored*
        remove p_k from *unexplored*
        send ⟨M⟩ to p_k
    else
        if $parent \neq p_i$ then send ⟨parent⟩ to *parent*
        terminate // DFS subtree rooted at p_i has been built

Παράδειγμα Εκτέλεσης του DFS-ST Αλγορίθμου



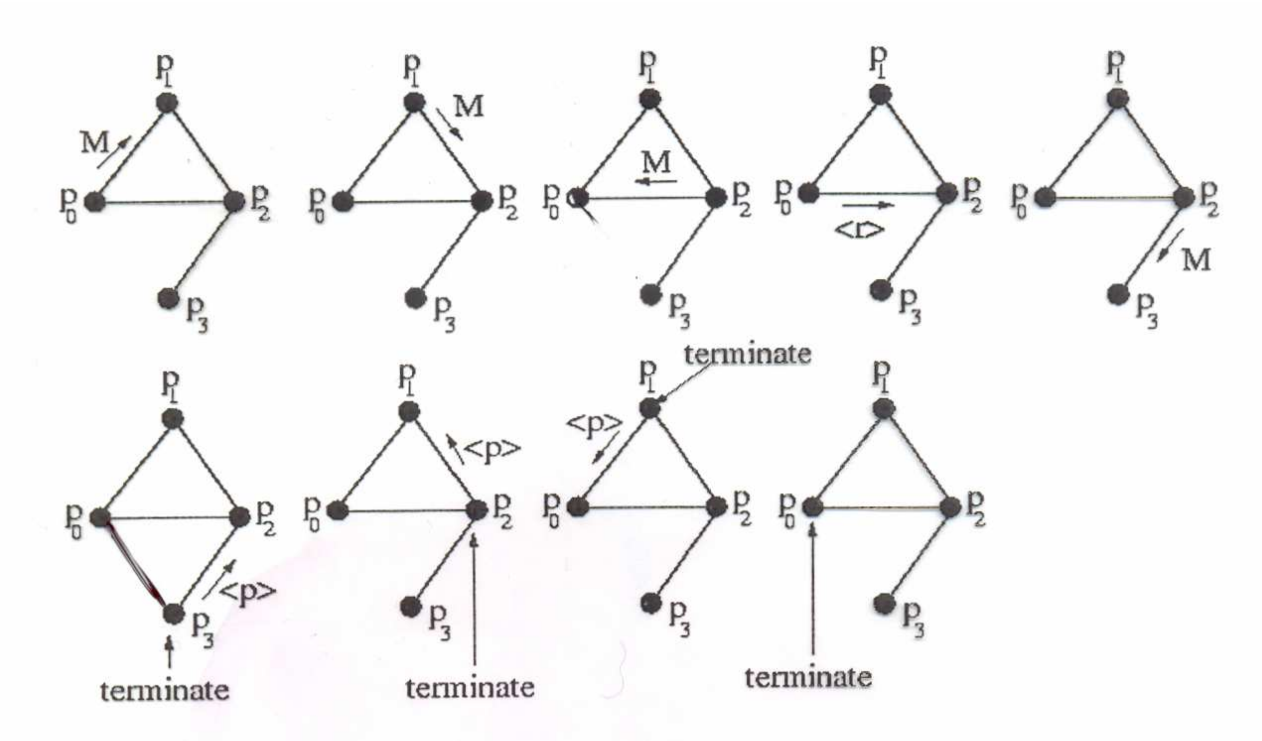
unexplored$_0$ = {2} parent$_0$ = nil children$_0$ = {}

unexplored$_1$ = {0,2} parent$_1$ = nil children$_1$ = {}

unexplored$_2$ = {0,1,3} parent$_2$ = nil children$_2$ = {}

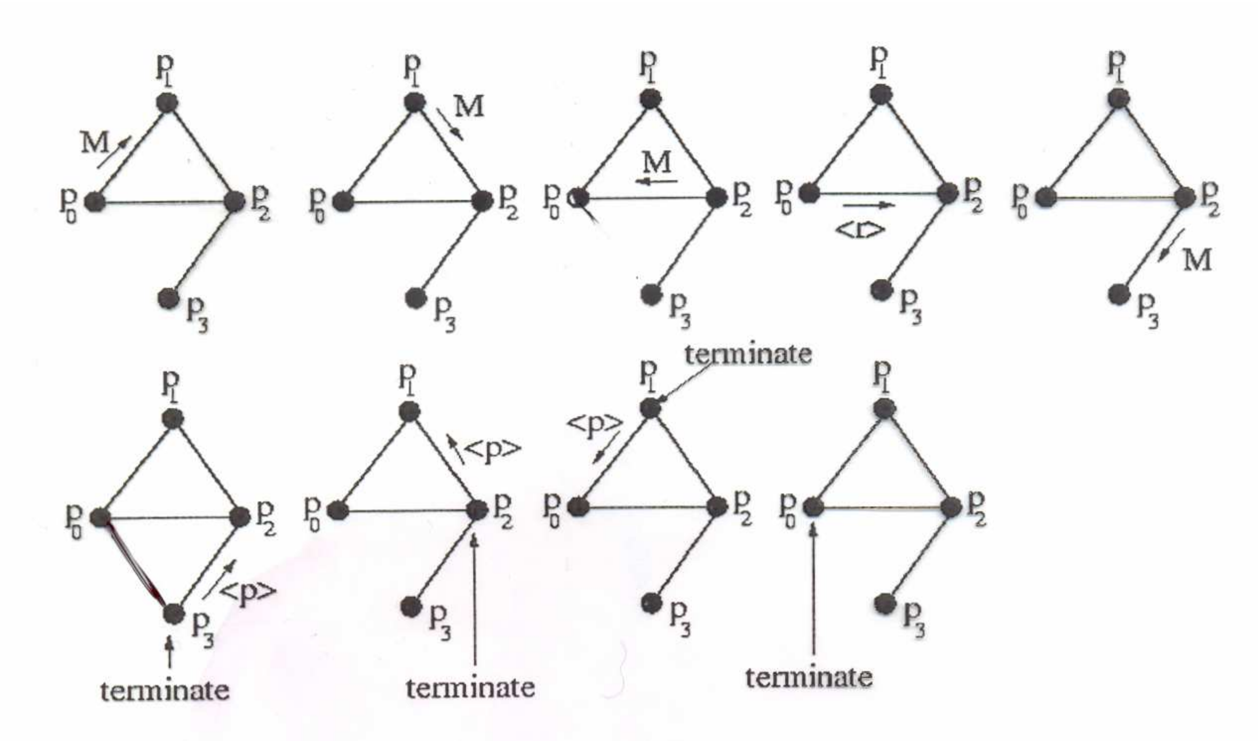
unexplored$_3$ = {2} parent$_3$ = nil children$_3$ = {}

Παράδειγμα Εκτέλεσης του DFS-ST Αλγορίθμου



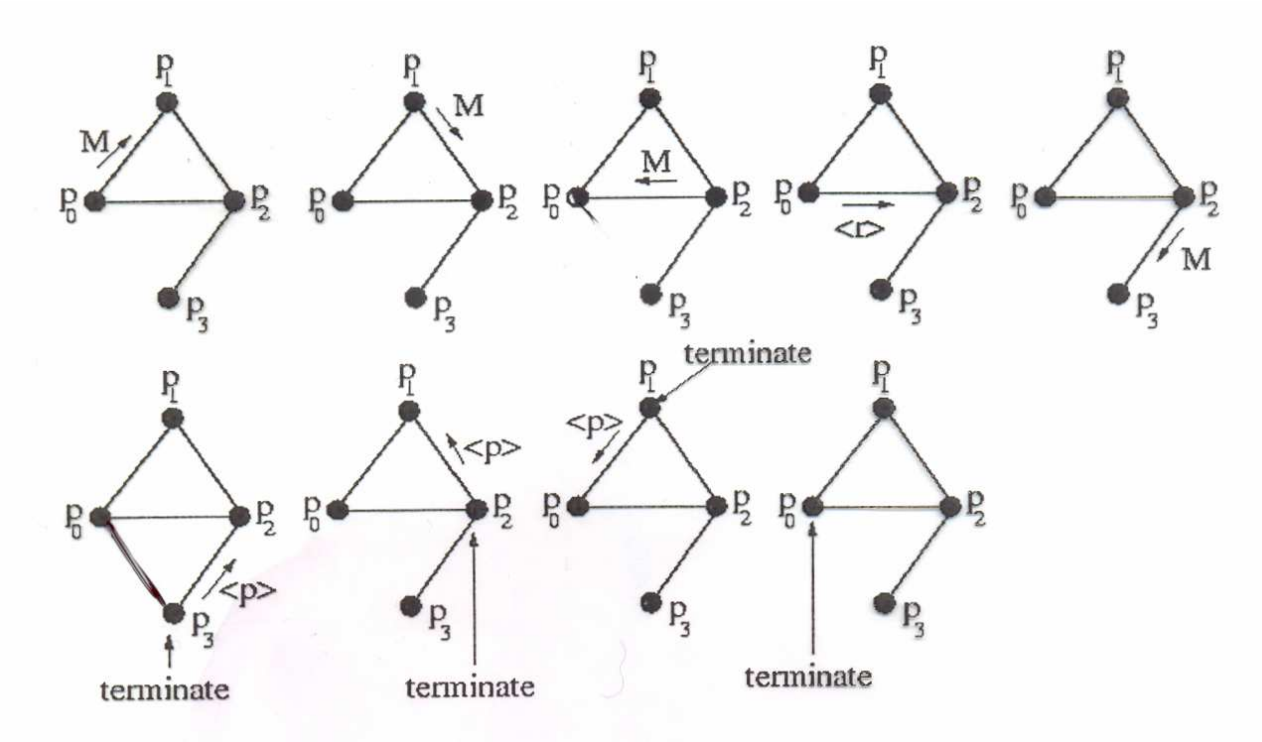
$unexplored_0 = \{2\}$	$parent_0 = 0$	$children_0 = \{\}$
$unexplored_1 = \{2\}$	$parent_1 = 0$	$children_1 = \{\}$
$unexplored_2 = \{0,1,3\}$	$parent_2 = nil$	$children_2 = \{\}$
$unexplored_3 = \{2\}$	$parent_3 = nil$	$children_3 = \{\}$

Παράδειγμα Εκτέλεσης του DFS-ST Αλγορίθμου



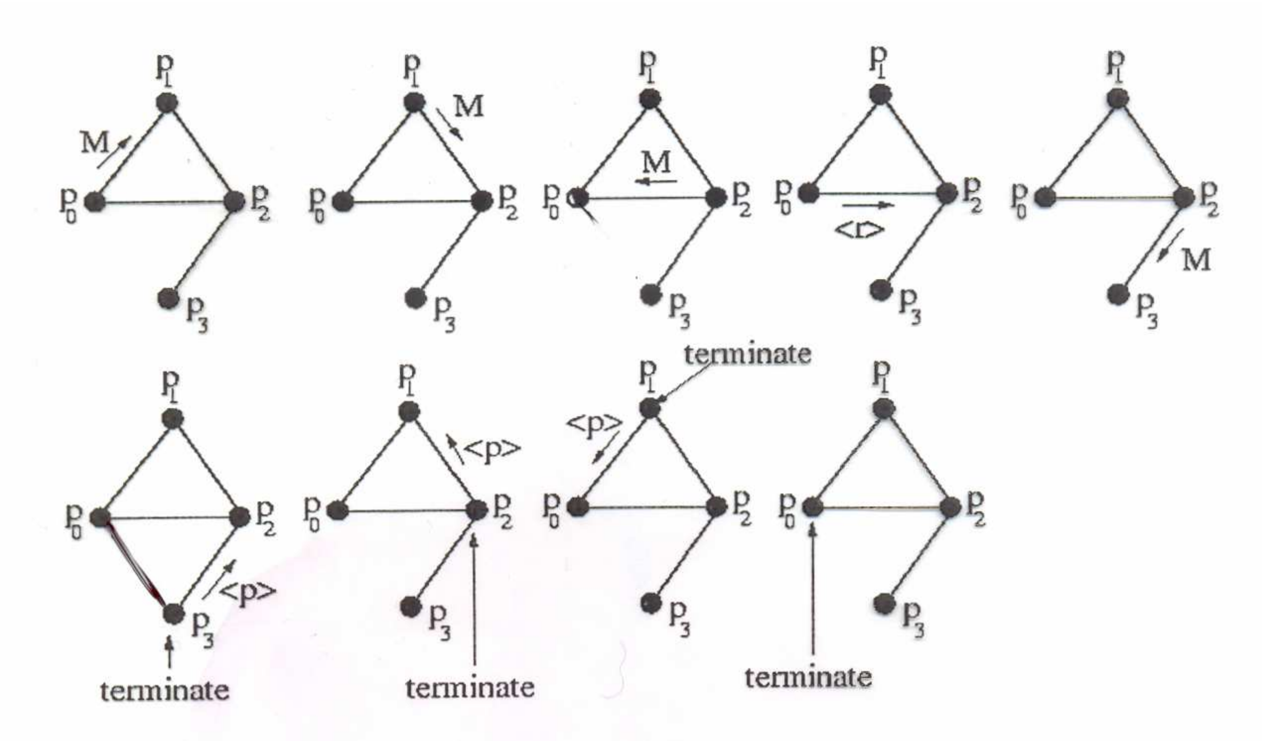
$unexplored_0 = \{2\}$	$parent_0 = 0$	$children_0 = \{\}$
$unexplored_1 = \{\}$	$parent_1 = 0$	$children_1 = \{\}$
$unexplored_2 = \{0,3\}$	$parent_2 = 1$	$children_2 = \{\}$
$unexplored_3 = \{2\}$	$parent_3 = nil$	$children_3 = \{\}$

Παράδειγμα Εκτέλεσης του DFS-ST Αλγορίθμου



unexplored$_0$ = {} parent$_0$ = 0 children$_0$ = {}
unexplored$_1$ = {} parent$_1$ = 0 children$_1$ = {}
unexplored$_2$ = {3} parent$_2$ = 1 children$_2$ = {}
unexplored$_3$ = {2} parent$_3$ = nil children$_3$ = {}

Παράδειγμα Εκτέλεσης του DFS-ST Αλγορίθμου



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

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 (ή μήπως < 2m?).

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 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) και m το αναγνωριστικό του p_m

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

```
Initially parent = ⊥, leader = −1, children = ∅, unexplored = all neighbors of p_i

upon receiving no message:
    if parent = ⊥ then                                  // wake up spontaneously
        leader := id
        parent := p_i
        explore()

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

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

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

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

Δημιουργία ενός DFS Δένδρου Επικάλυψης χωρίς να υπάρχει καθορισμένος κόμβος ρίζα

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.

Σύγχρονα Συστήματα – Υπολογισμός δένδρου επικάλυψης

Εφαρμογές

Εκπομπή με πολλούς αποδέκτες;

Υπολογισμός της διαμέτρου;

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

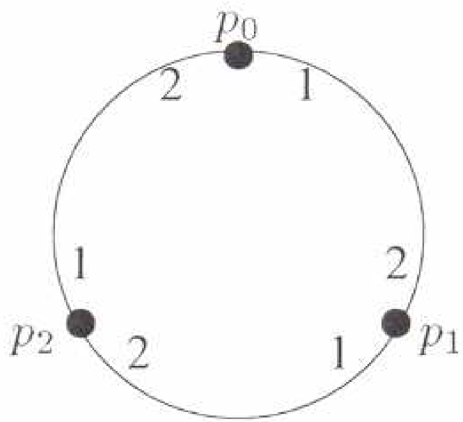
Φορμαλιστικά:

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.

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.

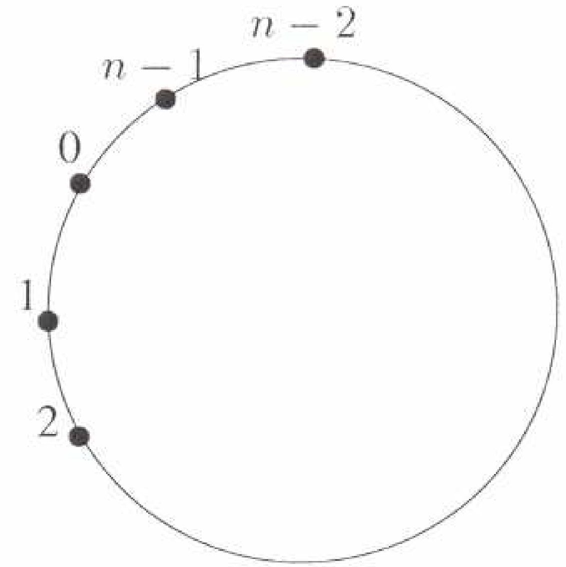
Ασύγχρονοι Δακτύλιοι

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, $0 \le i < n$.

```
Initially, asleep = true

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

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

upon receiving ⟨reply,j,k⟩ from left (resp., right):
    if j ≠ id then send ⟨reply,j,k⟩ to right (resp., left)    // forward the reply
    else                                                   // reply is for own probe
        if already received ⟨reply,j,k⟩ from right (resp., left) then
            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:

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

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

Σύγχρονοι Δακτύλιοι

Η μη-λήψη ενός μηνύματος αποτελεί τώρα χρήσιμη πληροφορία. *Βοηθάει αυτό;*

Ένας σύγχρονος αλγόριθμος εκλογής αρχηγού με πολυπλοκότητα επικοινωνίας O(n)

- Οι διεργασίες ξεκινούν τον αλγόριθμο είτε αυτογενώς (αυθόρμητα) σε κάποιο αυθαίρετο γύρο ή με τη λήψη ενός μηνύματος.
- Κάθε διεργασία που ξεκινά αυτογενώς θεωρείται ενεργή και στέλνει το μήνυμά της σε ένα γρήγορο μήνυμα (fast message) το οποίο ταξιδεύει με ταχύτητα 1 ακμή/γύρο.
- Μια διεργασία που ξεκινά τον αλγόριθμο με τη λήψη ενός μηνύματος λειτουργεί μόνο ως αναμεταδότης μηνυμάτων (relay).
- Ένα γρήγορο μήνυμα γίνεται αργό όταν παραλαμβάνεται από την πρώτη ενεργή διεργασία. Ένα αργό μήνυμα, το οποίο προέρχεται από τη διεργασία i, καθυστερείται κατά 2^i-1 γύρους σε κάθε διεργασία που το παραλαμβάνει και στη συνέχεια μεταδίδεται στην επόμενη διεργασία προς τα αριστερά στο δακτύλιο.
- Μια διεργασία εκλέγεται αρχηγός, αν λάβει το δικό της μήνυμα.

Ένας σύγχρονος αλγόριθμος εκλογής αρχηγού με πολυπλοκότητα επικοινωνίας O(n)

Algorithm 6 Synchronous leader election: code for processor p_i, $0 \le i < n$.

Initially *waiting* is empty and *status* is asleep

let R be the set of messages received in this computation event
$S := \emptyset$ // the messages to be sent

if *status* = asleep then
    if R is empty then // woke up spontaneously
        status := participating
        $min := id$
        add $\langle id,1\rangle$ to S // first phase message
    else
        status := relay
        $min := \infty$

for each $\langle \overset{id}{m},\overset{phase}{h}\rangle$ in R do
    if $m < min$ then
        become not elected
        $min := m$
        if (*status* = relay) and ($h = 1$) then // m stays first phase
            add $\langle m,h\rangle$ to S
        else // m is/becomes second phase
            add $\langle m,2\rangle$ to *waiting* tagged with current round number
    elsif $m = id$ then become elected
// if $m > min$ then message is swallowed

for each $\langle m,2\rangle$ in *waiting* do
    if $\langle m,2\rangle$ was received $2^m - 1$ rounds ago then
        remove $\langle m\rangle$ from *waiting* and add to S

send S to left

Ένας σύγχρονος αλγόριθμος εκλογής αρχηγού με πολυπλοκότητα επικοινωνίας O(n)

Λήμμα 1: Μόνο η διεργασία με το μικρότερο αναγνωριστικό ανάμεσα στις ενεργές διεργασίες λαμβάνει το δικό της μήνυμα.

Για να υπολογιστεί ο αριθμός μηνυμάτων που αποστέλλονται, κατηγοριοποιούμε τα μηνύματα σε 3 κατηγορίες:

- **Κατηγορία 1**: Μηνύματα $1^{ης}$ φάσης
- **Κατηγορία 2**: Μηνύματα $2^{ης}$ φάσης που αποστέλλονται πριν το μήνυμα του μελλοντικού αρχηγού μπει στη $2^{η}$ φάση
- **Κατηγορία 3**: Μηνύματα $3^{ης}$ φάσης που αποστέλλονται αφού το μήνυμα του μελλοντικού αρχηγού μπει στη $2^{η}$ φάση

Λήμμα 2

Ο συνολικός αριθμός μηνυμάτων στην πρώτη κατηγορία είναι το πολύ n.

Απόδειξη: Το πολύ ένα μήνυμα $1^{ης}$ φάσης μεταδίδεται από κάθε διεργασία.

Ένας σύγχρονος αλγόριθμος εκλογής αρχηγού με πολυπλοκότητα επικοινωνίας O(n)

Έστω r ο πρώτος γύρος στον οποίο κάποια διεργασία ξεκινά την εκτέλεση του αλγορίθμου και έστω p_i μια από αυτές τις διεργασίες.

Λήμμα 3: Αν η p είναι σε απόσταση k δεξιά από την p_i, τότε το πρώτο μήνυμα 1^{ης} φάσης λαμβάνεται από την p το αργότερο ως τον γύρο r+k.

Λήμμα 4: Ο συνολικός αριθμός μηνυμάτων κατηγορίας 2 είναι το πολύ n.

Απόδειξη: Το μήνυμα του μελλοντικού αρχηγού μπαίνει στην 2^{η} φάση το πολύ μετά από n γύρους από όταν το πρώτο μήνυμα του αλγορίθμου αποστέλλεται.

Κάθε μήνυμα <i> αποστέλλεται το πολύ $n/2^i$ φορές.

<u>Χειρότερη Περίπτωση</u>: Όλες οι διεργασίες συμμετέχουν και τα αναγνωριστικά είναι όσο το δυνατό μικρότερα.

Ο αριθμός μηνυμάτων στην κατηγορία 2 είναι Sum_{i=1 to n-1} $n/2^i \leq$ n.

Ένας σύγχρονος αλγόριθμος εκλογής αρχηγού με πολυπλοκότητα επικοινωνίας O(n)

Λήμμα 5: Ο συνολικός αριθμός μηνυμάτων κατηγορίας 3 είναι το πολύ 2n.

Απόδειξη

Έστω p_i ο μελλοντικός αρχηγός και p_j μια άλλη ενεργή διαδικασία ($p_i < p_j$).

Το πολύ $n * 2^{idi}$ γύροι απαιτούνται από το μήνυμα $<id_i>$ για να επιστρέψει στον $p_i \rightarrow$ μηνύματα κατηγορίας 3 στέλνονται μόνο κατά τη διάρκεια $n*2^{idi}$ γύρων.

Το μήνυμα $<id_j>$ προωθείται το πολύ:

$$n*2^{idi} / 2^{idj} = n / 2^{idj-idi}$$

Ο συνολικός αριθμός μηνυμάτων που αποστέλλονται σε αυτή την κατηγορία είναι:

$$\text{Sum\_}\{j=0 \text{ to } n-1\}\ n/2^{idj-idi}.$$

Στη χειρότερη περίπτωση όλες οι διεργασίες είναι ενεργές και τα αναγνωριστικά είναι όσο το δυνατόν μικρότερα:

$$\text{Sum\_}\{k=0 \text{ to } n-1\}\ n/2^{k} \leq 2n.$$

Εκλογή Αρχηγού σε Γενικό Σύγχρονο Δίκτυο - Flooding

Αλγόριθμος FloodMax

Κώδικας για p_i

- Κάθε διεργασία p_i αποθηκεύει το μέγιστο UID που έχει δει μέχρι τώρα (αρχικά το UID της p_i).
- Σε κάθε γύρο, η p_i προωθεί αυτή τη μέγιστη τιμή σε όλους τους γείτονές της.
- Μετά από diam γύρους, αν η μέγιστη τιμή που είδε η p_i είναι το δικό της αναγνωριστικό, η p_i αποφασίζει ότι είναι ο αρχηγός. Διαφορετικά, τερματίζει σε κατάσταση μη-ισχύος.

Κατάσταση p_i:

id_i: αναγνωριστικό της διεργασίας p_i

max-id_i: μέγιστο αναγνωριστικό που η p_i έχει δει μέχρι τώρα, αρχικά id_i

$status_i \in$ {UNKNOWN, LEADER, NON-LEADER}, αρχικά UNKNOWN

$rounds_i$: ένας ακέραιος, αρχικά 0

Εκλογή Αρχηγού σε Γενικό Σύγχρονο Δίκτυο - Αλγόριθμος FloodMax

Αρχικά, το id_i περιέχεται σε όλους τους outbuf πίνακες της p_i, $\forall$ i.

Ενέργειες p_i σε κάθε γύρο

$rounds_i = rounds_i + 1$;

let U be the set of UIDs that arrive from neighboring processes;

$max\text{-}uid_i = \max(\{max\text{-}uid_i\} \cup U)$

if ($rounds_i$ == diam) then

    if ($max\text{-}uid_i = id_i$) then $status_i$ = LEADER;

    else $status_i$ = NON-LEADER;

if ($rounds_i$ < diam) then

    send $max\text{-}uid_i$ to all neighbors;

Εκλογή Αρχηγού σε Γενικό Σύγχρονο Δίκτυο - Αλγόριθμος FloodMax

Έστω i_{max} ο δείκτης της διεργασίας με το μέγιστο αναγνωριστικό και id_{max} το αναγνωριστικό αυτό.

Θεώρημα

Σε κάθε εκτέλεση του αλγορίθμου FloodMax, μέσα σε (το πολύ) diam γύρους, η διεργασία i_{max} τερματίζει σε κατάσταση ισχύος, ενώ όλες οι υπόλοιπες διεργασίες τερματίζουν σε κατάσταση μη-ισχύος.

Απόδειξη

Για κάθε $0 \leq r \leq$ diam και για κάθε διεργασία j, μετά από r γύρους, αν η απόσταση από της j από την i_{max} είναι το πολύ r, τότε $max\text{-}id_j = id_{max}$.

Για να αποδειχθεί το παραπάνω βοηθάει να αποδείξουμε ότι:

- Για κάθε r και j, μετά από r γύρους, $rounds_j = r$.
- Για κάθε r και j, μετά από r γύρους, $max\text{-}id_j \leq id_{max}$.

Εκλογή Αρχηγού σε Γενικό Σύγχρονο Δίκτυο - Αλγόριθμος FloodMax

Πολυπλοκότητα

Χρονική Πολυπλοκότητα;

Πολυπλοκότητα Επικοινωνίας;

Μειώνοντας την Πολυπλοκότητα Επικοινωνίας – Αλγόριθμος OptFloodMax

Πως θα μπορούσαμε να μειώσουμε την πολυπλοκότητα επικοινωνίας (χωρίς απαραίτητα να επιτυγχάνουμε βελτίωση της τάξης της);

Εκλογή Αρχηγού σε Γενικό Σύγχρονο Δίκτυο - Αλγόριθμος OptFloodMax

Η κατάσταση της p_i έχει μια ακόμη μεταβλητή, την new-info$_i$, αρχικά TRUE. Αρχικά, το id$_i$ περιέχεται σε όλους τους outbuf πίνακες της p_i, $\forall$ i.

Ενέργειες p_i σε κάθε γύρο

rounds$_i$ = rounds$_i$ + 1;
let U be the set of UIDs that arrive from neighboring processes
if (max(U) > max-id$_i$) then new-info$_i$ = TRUE;
else new-info$_i$ = FALSE;
max-uid$_i$ = max({max-uid$_i$} $\cup$ U)
if (rounds$_i$ == diam) then
    if (max-uid$_i$ = id$_i$) then status$_i$ = LEADER;
    else status$_i$ = NON-LEADER;
if (rounds$_i$ < diam AND new-info$_i$ == TRUE) then
    send max-uid$_i$ to all neighbors

Εκλογή Αρχηγού σε Γενικό Σύγχρονο Δίκτυο - Αλγόριθμος OptFloodMax

Θεώρημα

Σε κάθε εκτέλεση του αλγορίθμου OptFloodMax, μέσα σε (το πολύ) diam γύρους, η διεργασία i_{max} τερματίζει σε κατάσταση ισχύος, ενώ όλες οι υπόλοιπες διεργασίες τερματίζουν σε κατάσταση μη-ισχύος.

Απόδειξη – Βασικές Ιδέες

Λήμμα 1: Για κάθε r, $0 \leq r \leq$ diam, και κάθε διεργασία i,j με την j να είναι γειτονική διεργασία στην i, ισχύει ότι αν max-id_j < max-id_i τότε new-$info_i$ = TRUE.

Απόδειξη: Με επαγωγή στο r.

Βάση επαγωγής προφανής αφού όλες οι new-info μεταβλητές είναι αρχικά TRUE.

Επαγωγικό Βήμα:

Αν max-id_i αυξάνει στον γύρο r, από τον κώδικα, η new-$info_i$ γίνεται TRUE.

Αν max-id_i δεν αυξάνει στον γύρο r, η επαγωγική υπόθεση συνεπάγεται ότι είτε max-id_j έχει ήδη τουλάχιστον όσο μεγάλη τιμή έχει και το max-id_i ή διαφορετικά new-$info_i$ == TRUE αμέσως πριν ξεκινήσει ο γύρος r.

Εκλογή Αρχηγού σε Γενικό Σύγχρονο Δίκτυο - Αλγόριθμος OptFloodMax

Λήμμα: Για κάθε r, $0 \le r \le$ diam, μετά από r γύρους, οι τιμές των μεταβλητών id, max-id, status και rounds είναι ίδιες στις καταστάσεις που προκύπτουν κατά το τρέξιμο των δύο αλγορίθμων.

Απόδειξη: Με επαγωγή στο r.

Έστω i μια οποιαδήποτε διεργασία και j μια γειτονική της διεργασία.

Αν new-info_i == TRUE πριν το γύρο r, τότε η i στέλνει την ίδια πληροφορία στη j στο γύρο r και στους δύο αλγορίθμους.

Αν new-info_i == FALSE πριν το γύρο r, τότε η i δεν στέλνει τίποτα στην j στον γύρο r στον OptFloodMax, ενώ στέλνει max-id_i στον FloodMax. Στην περίπτωση αυτή όμως, το Λήμμα 1 εγγυάται ότι $\text{max-id}_j \geq \text{max-id}_i$ αμέσως πριν το γύρο r.

Άρα, οι ενέργειες της i έχουν την ίδια επίδραση και στους δύο αλγορίθμους.

Αφού αυτό ισχύει για όλα τα i,j, συνεπάγεται ότι οι τιμές max-id είναι ίδιες κατά τη διάρκεια εκτέλεσης των δύο αλγορίθμων.

Εύρεση Συντομότερων Μονοπατιών από μια πηγή σε Γενικά Σύγχρονα Δίκτυα

Ας θεωρήσουμε έναν ισχυρά συνδεδεμένο **κατευθυνόμενο** γράφο. Υποθέτουμε ότι με κάθε ακμή e = <i,j> έχει συσχετιστεί μια μη-αρνητική πραγματική τιμή που λέγεται *βάρος* και συμβολίζεται με weight(e) ή $weight_{i,j}$.

Το *βάρος ενός μονοπατιού* είναι το άθροισμα των βαρών των ακμών του.

Ένα *συντομότερο μονοπάτι* από κάποιο κόμβο i προς κάποιο άλλο κόμβο j είναι ένα μονοπάτι με ελάχιστο βάρος (ανάμεσα σε αυτά που ενώνουν τον i με τον j).

Πρόβλημα

Εύρεση των συντομότερων μονοπατιών από κάποιο διακεκριμένο κόμβο i_0 προς κάθε άλλο κόμβο του γράφου.

Υποθέτουμε ότι κάθε διεργασία γνωρίζει το βάρος όλων των ακμών που πρόσκεινται σε αυτή.

Υποθέτουμε ότι κάθε διεργασία γνωρίζει το n, τον αριθμό διεργασιών στο σύστημα.

Εύρεση Συντομότερων Μονοπατιών από μια πηγή σε Γενικά Σύγχρονα Δίκτυα

Απαιτείται κάθε διεργασία να γνωρίζει:

- τον πατέρα της, και
- την απόστασή της (συνολικό βάρος συντομότερου μονοπατιού) από τον κόμβο ρίζα στο δένδρο συντομότερων μονοπατιών που θα κατασκευαστεί.

Αν όλες οι ακμές είχαν το ίδιο βάρος το δένδρο συντομότερων μονοπατιών με ρίζα i_0 θα ήταν το ίδιο με το BFS δένδρο με ρίζα p_r.

Υποθέτουμε ότι διαφορετικές ακμές μπορεί να έχουν διαφορετικά βάρη.

Εύρεση Συντομότερων Μονοπατιών από μια πηγή σε Γενικά Σύγχρονα Δίκτυα

Αλγόριθμος SynchBellmanFord – Διεργασία i

Η i έχει μια μεταβλητή $dist_i$ στην οποία αποθηκεύει την απόσταση της από την i_0 που γνωρίζει μέχρι τον τρέχοντα γύρο ως συντομότερη. Αρχικά, $dist_r = 0$ και $dist_i = \infty$ για κάθε $i \neq r$.

Υπάρχει επίσης μια διεργασία $parent_i$, στην οποία αποθηκεύεται ένας κόμβος j που προηγείται του i στο μονοπάτι με βάρος $dist_i$. Αρχικά όλοι οι parent = nill.

Σε κάθε γύρο, η i στέλνει την $dist_i$ διαμέσου όλων των ακμών που εξέρχονται της i.

Η i ενημερώνει την $dist_i$ κάθε φορά που λαμβάνει μήνυμα από κάποιο εισερχόμενο γειτονικό κόμβο π.χ., j, εκτελώντας ένα βήμα ενημέρωσης (relaxation step) στο οποίο υπολογίζει το ελάχιστο μεταξύ της προηγούμενης τιμής της $dist_i$ και των τιμών $dist_j + weight_{j,i}$.

Κάθε φορά που η $dist_i$ αλλάζει, η $parent_i$ επίσης ενημερώνεται.

Μετά από n-1 γύρους, η $dist_i$ περιέχει τη συντομότερη απόσταση, ενώ η μεταβλητή $parent_i$ τον σωστό γονικό κόμβο στο δένδρο επικάλυψης συντομότερων μονοπατιών.

Αλγόριθμος SynchBellmanFord

Ορθότητα

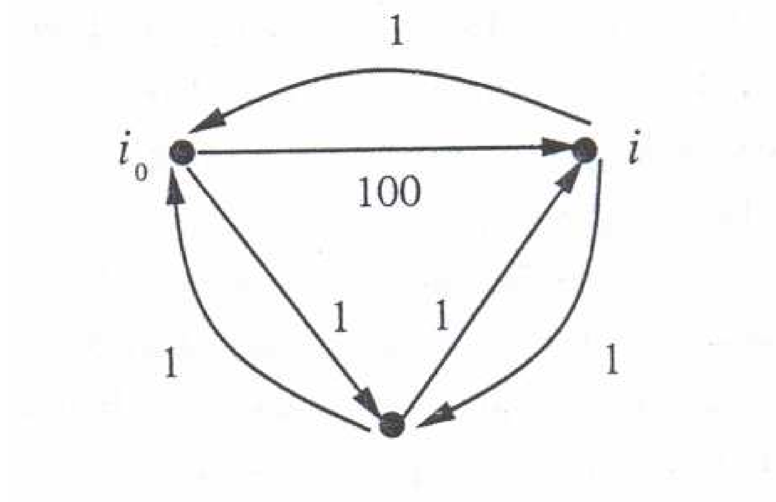
Δεν είναι δύσκολο να αποδειχθεί ότι σε κάθε γύρο r:

Για κάθε διεργασία i, οι μεταβλητές $dist_i$ και $parent_i$ έχουν τιμές που αντιστοιχούν στην απόσταση και στον γονικό (εισερχόμενο) κόμβο ενός συντομότερου μονοπατιού από την i_0 στην i που αποτελείται από το πολύ r ακμές.

Πολυπλοκότητα

Αριθμός Μηνυμάτων; (n-1) γύροι

Χρονική Πολυπλοκότητα; (n-1)*|E|



Εύρεση Ελάχιστου Δένδρου Επικάλυψης σε Σύγχρονο Δίκτυο

Ένα *δάσος επικάλυψης* (spanning forest) ενός **μη-κατευθυνόμενου** γράφου G = (V,E) (με βάρη στις ακμές) είναι ένα δάσος (δηλαδή, ένας γράφος που είναι κυκλικός αλλά όχι απαραίτητα συνδεδεμένος) που αποτελείται από ακμές του G και περιέχει όλους τους κόμβους του G.

Ένα *δένδρο επικάλυψης* ενός γράφου G είναι ένα συνδεδεμένο δάσος επικάλυψης του G.

Το *βάρος* ενός υπογράφου G' του G είναι το άθροισμα των βαρών των ακμών του G'.

Πρόβλημα

- Εύρεση ενός δένδρου επικάλυψης ελάχιστου βάρους.
  - Κάθε διεργασία πρέπει να αποφασίσει ποιες ακμές πρόσκεινται σε αυτή στο δένδρο επικάλυψης ελάχιστου βάρους και ποιες όχι.

Εύρεση Ελάχιστου Δένδρου Επικάλυψης - Βασική Θεωρία

Βασικές Ιδέες

- Εκκίνηση με το προφανές δάσος επικάλυψης στο οποίο κάθε κόμβος αποτελεί συνιστώσα (το δάσος δεν έχει καθόλου ακμές).
- Επαναληπτικά, συγχώνευσε συνιστώσες μέχρι να δημιουργηθεί μια και μοναδική συνιστώσα (που θα αποτελεί το δένδρο επικάλυψης).
- Η συγχώνευση θα πρέπει να γίνει προσεκτικά ώστε το δένδρο που θα προκύψει να έχει ελάχιστο βάρος.

Λήμμα 1: Έστω $G = (V,E)$ ένας γράφος με βάρη στις ακμές και έστω $\{(V_i, E_i): 1 \leq i \leq k\}$ ένα δάσος επικάλυψης για τον G, όπου $k > 1$. Για μια αυθαίρετη τιμή του i, $1 \leq i \leq k$, έστω e μια ακμή με το μικρότερο βάρος ανάμεσα στις ακμές του συνόλου

$$\{e': \text{η } e' \text{ έχει ακριβώς ένα άκρο στο } V_i\}.$$

Τότε, υπάρχει ένα δένδρο επικάλυψης για τον G που περιέχει τις ακμές $\cup_j E_j$ και την e, και το οποίο είναι ελάχιστου βάρους μεταξύ όλων των δένδρων επικάλυψης του G που περιέχουν τις ακμές $\cup_j E_j$.

Εύρεση Ελάχιστου Δένδρου Επικάλυψης - Βασική Θεωρία

Απόδειξη Λήμματος 1

Με εις άτοπο απαγωγή. Έστω ότι υπάρχει δένδρο επικάλυψης T που περιέχει τις ακμές στο $\cup_j E_j$, δεν περιέχει την e και έχει < βάρος από οποιοδήποτε άλλο δένδρο επικάλυψης που περιέχει το $\cup_j E_j$ και την e.

Έστω T' ο γράφος που προκύπτει αν εισάγουμε την e στο T. Προφανώς ο T' περιέχει έναν κύκλο που περιέχει μια άλλη ακμή e' η οποία (όπως η e) εξέρχεται του V_i.

Από τον τρόπο με τον οποίο η e επιλέγεται, weight(e) $\leq$ weight(e').

Έστω τώρα ο γράφος T'' που προκύπτει αν η e' διαγραφεί από τον T'.

Ο κύκλος καταστρέφεται, οπότε ο T'' είναι δένδρο επικάλυψης του G, περιέχει τις $\cup_j E_j$ και την e και το βάρος του δεν είναι μεγαλύτερο από του T. Άτοπο.

Εύρεση Ελάχιστου Δένδρου Επικάλυψης - Βασική Θεωρία

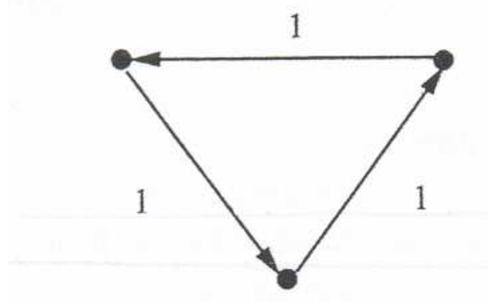
Γενική Στρατηγική για επίλυση του προβλήματος MST

Ξεκινώντας με το προφανές δάσος επικάλυψης στο οποίο δεν υπάρχουν ακμές, επαναληπτικά εκτελούνται τα εξής:

- Επιλέγεται μια αυθαίρετη συνιστώσα C του δάσους και μια αυθαίρετη ακμή e που εξέρχεται της C με ελάχιστο βάρος (ανάμεσα στις ακμές που εξέρχονται της C).
- Μια νέα συνιστώσα δημιουργείται συγχωνεύοντας τη C με τη συνιστώσα στην άλλη πλευρά της e. Η νέα συνιστώσα προφανώς περιέχει την e.
- Ο αλγόριθμος τερματίζει όταν το δάσος αποτελείται από μια μόνο συνιστώσα.

Ποια είναι η παράλληλη έκδοση αυτού του αλγορίθμου;

Γιατί ο αλγόριθμος αυτός αποτυγχάνει στην παράλληλη έκδοσή του;



Εύρεση Ελάχιστου Δένδρου Επικάλυψης - Βασική Θεωρία

Λήμμα 2: Αν όλες οι ακμές ενός γράφου G έχουν μοναδικά βάρη, τότε υπάρχει ακριβώς ένα MST για τον G.

Απόδειξη: Αντίστοιχη αυτής του Λήμματος 1.

Αν υπάρχουν δύο διαφορετικά δένδρα ελάχιστου βάρους T και T' μπορούμε να προσθέσουμε μια ακμή σε ένα από αυτά και να αφαιρέσουμε μια άλλη, δημιουργώντας ένα δένδρο με μικρότερο βάρος από το αρχικό. Άτοπο.

Αλγόριθμος SynchGHS

- Οι συνιστώσες χτίζονται σε εποχές.
- Για κάθε k, οι συνιστώσες της εποχής k αποτελούν ένα δάσος επικάλυψης.
- Κάθε συνιστώσα εποχής k του δάσους, είναι ένα δένδρο-υπογράφος του MST.
- Κάθε συνιστώσα εποχής k έχει τουλάχιστον 2^k κόμβους.
- Ένας κόμβος κάθε συνιστώσας, σε κάθε εποχή, είναι αρχηγός.
- Οι διεργασίες εκτελούν O(n) γύρους σε κάθε εποχή.

Οι n συνιστώσες της εποχής 0 αποτελούνται από έναν κόμβο η κάθε μια (και δεν έχουν καθόλου ακμές).

Ας υποθέσουμε επαγωγικά ότι έχουν καθοριστεί οι συνιστώσες της εποχής k (καθώς και οι αρχηγοί τους), $k \geq 0$. Έστω ότι κάθε διεργασία γνωρίζει το id του αρχηγού της συνιστώσας στην οποία ανήκει. Αυτό το id χρησιμοποιείται ως αναγνωριστικό ολόκληρης της συνιστώσας.

Κάθε διεργασία γνωρίζει επίσης ποιες από τις ακμές που πρόσκεινται σε αυτή βρίσκονται στο δένδρο επικάλυψης της συνιστώσας.

Σχηματισμός των συνιστωσών της εποχής k+1:

1. Κάθε συνιστώσα C της εποχής k ψάχνει (μέσω του δένδρου επικάλυψης της) για μια ακμή e τ.ω. η e εξέρχεται της C και έχει το ελάχιστο βάρος ανάμεσα στις ακμές που εξέρχονται της C. *Πως μπορεί να γίνει αυτό;*

2. Όταν όλες οι συνιστώσες της εποχής k έχουν βρει τις εξερχόμενες ακμές τους με το ελάχιστο βάρος, οι συνιστώσες συγχωνεύονται μέσω αυτών των ακμών προκειμένου να σχηματιστούν οι συνιστώσες της εποχής k+1.

3. Ο αρχηγός κάθε συνιστώσας της εποχής k επικοινωνεί με τη διεργασία της συνιστώσας που πρόσκειται στην ακμή με ελάχιστο βάρος που επιλέχθηκε από τη συνιστώσα, έτσι ώστε αυτή να μαρκάρει την ακμή αυτή ως ακμή του δένδρου. Η διεργασία στην άλλη πλευρά της ακμής εκτελεί την ίδια ενέργεια.

Επιλογή αρχηγού για κάθε συνιστώσα της εποχής k+1:

Μπορεί να αποδειχθεί ότι:

Για κάθε ομάδα από συνιστώσες της εποχής k που συνενώνονται για να δημιουργήσουν μια συνιστώσα της εποχής k+1, υπάρχει μια μοναδική ακμή e που είναι κοινή ακμή ελάχιστου βάρους για δύο από τις συνιστώσες της εποχής k.

- Ο νέος αρχηγός είναι η διεργασία με το μεγαλύτερο αναγνωριστικό από τις δύο διεργασίες που πρόσκεινται στην e.
- Το id του νέου αρχηγού προωθείται σε όλες τις διεργασίες της νέας συνιστώσας εποχής k+1 που δημιουργείται. *Πως γίνεται αυτό;*

Τερματισμός:

- Μετά από έναν συγκεκριμένο αριθμό εποχών, το δάσος επικάλυψης αποτελείται από μόνο μια συνιστώσα.
- Οποιαδήποτε προσπάθεια εύρεσης εξερχόμενης ακμής θα αποτυγχάνει.
- Όταν ο αρχηγός πληροφορείται το γεγονός αυτό, εκπέμπει μήνυμα τερματισμού του αλγορίθμου.

Αλγόριθμος SynchGHS – Ορθότητα

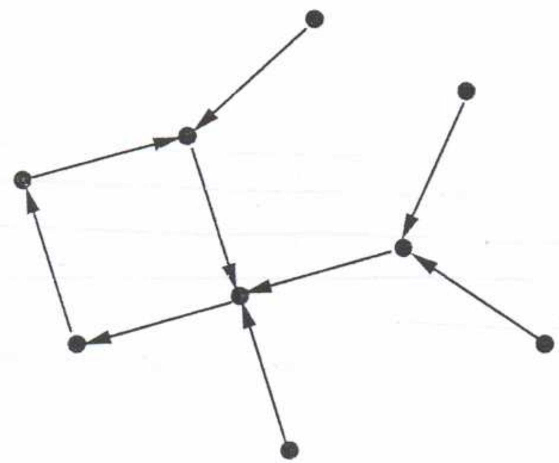
Σε κάθε ομάδα από συνιστώσες επιπέδου k που συνδυάζονται, υπάρχει μια και μοναδική ακμή που είναι κοινή εξερχόμενη ακμή ελάχιστου βάρους για δύο συνιστώσες.

Απόδειξη

Κατευθυνόμενος γράφος συνιστωσών G’: οι κόμβοι του G’ είναι οι συνιστώσες επιπέδου k που συνδυάζονται για να δημιουργήσουν μια συνιστώσα επιπέδου k+1 και οι ακμές του G’ είναι οι εξερχόμενες ακμές ελάχιστου βάρους των συνιστωσών του επιπέδου k μέσω των οποίων γίνεται η συγχώνευση.

Στον G’ κάθε κόμβος έχει μια εξερχόμενη ακμή και ο μη-κατευθυνόμενος γράφος που αντιστοιχεί στον G’ είναι συνδεδεμένος.

Αποδεικνύεται ότι ένας τέτοιος γράφος περιέχει ακριβώς έναν κύκλο.



Αλγόριθμος SynchGHS – Ορθότητα

Απόδειξη (συνέχεια)

Εξ αιτίας του τρόπου δημιουργίας του G', διαδοχικές ακμές στον κύκλο έχουν φθίνοντα βάρη

$\Rightarrow$ το μήκος του κύκλου δεν μπορεί να είναι > 2

$\Rightarrow$ το μήκος του κύκλου $= 2$

$\Rightarrow$ πρόκειται για μια ακμή που έχει επιλεγεί ως η εξερχόμενη ακμή ελάχιστου βάρους από δύο συνιστώσες

Γιατί είναι σημαντικό το σύστημα να είναι σύγχρονο;

Θα ήταν σωστός ο αλγόριθμος αν το σύστημα ήταν ασύγχρονο;

Αλγόριθμος SynchGHS

Πολυπλοκότητα

Πόσες εποχές θα εκτελεστούν συνολικά;

Πόσους γύροι εκτελούνται σε κάθε εποχή;

Ποια είναι η χρονική πολυπλοκότητα του αλγορίθμου;

Πόσα μηνύματα αποστέλλονται σε κάθε εποχή;

Ποια είναι η πολυπλοκότητα επικοινωνίας του αλγορίθμου;

Αλγόριθμος SynchGHS

Ο αλγόριθμος υποθέτει ότι τα βάρη των ακμών είναι όλα διαφορετικά.

Πως θα μπορούσαμε να επιλύσουμε το πρόβλημα χωρίς αυτή την υπόθεση;

Μπορούμε να ξεχωρίσουμε με κάποιο τρόπο διαφορετικές ακμές που έχουν το ίδιο βάρος;

Ασύγχρονα Συστήματα

Εκλογή Αρχηγού σε Γενικό Ασύγχρονο Σύστημα

Το σύστημα μοντελοποιείται ως μη-κατευθυνόμενος, συνδεδεμένος γράφος.

Ο αλγόριθμος FloodMax δεν λειτουργεί ως έχει στο ασύγχρονο μοντέλο γιατί στο μοντέλο αυτό δεν υπάρχει η έννοια του γύρου.

Πως θα μπορούσαμε να προσομοιώσουμε ένα γύρο υπολογισμού του αλγορίθμου στο ασύγχρονο μοντέλο;

Εκλογή Αρχηγού σε Γενικό Ασύγχρονο Δίκτυο

Κάθε διεργασία που στέλνει ένα μήνυμα στο γύρο r προσθέτει μια ετικέτα με την τιμή του r στο μήνυμα που αποστέλλει.

Ο παραλήπτης περιμένει μέχρι να λάβει μηνύματα με ετικέτα r από όλους τους γείτονες του πριν εκτελέσει το βήμα του στον γύρο r (round r transition).

Προσομοιώνοντας με αυτό τον τρόπο diam γύρους, ο αλγόριθμος τερματίζει και το αποτέλεσμα είναι σωστό.

Μπορούμε με κάποιο τρόπο να προσομοιώσουμε τον OptFloodMax (δηλαδή τη βελτιστοποιημένη έκδοση του FloodMax) σε ασύγχρονο σύστημα; Ποιο είναι το πρόβλημα που προκύπτει;

Εκλογή αρχηγού σε ασύγχρονο σύστημα δοθέντος του δένδρου επικάλυψης του γράφου χωρίς καθορισμένη ρίζα

Έστω ότι κάθε κόμβος ξέρει τους γειτονικούς του κόμβους στο δένδρο αλλά όχι ποιος από αυτούς είναι ο πατρικός του κόμβος.

Πως μπορούμε να εκλέξουμε αρχηγό σε ένα τέτοιο σύστημα;

Αλγόριθμος STtoLeader

Κάθε κόμβος φύλλο ξεκινά έναν αλγόριθμο convergecast στέλνοντας ένα μήνυμα τύπου elect στο μοναδικό γειτονικό του κόμβο.

Κάθε κόμβος v που λαμβάνει μηνύματα τύπου elect από όλους τους γείτονές του εκτός από έναν αποστέλλει ένα μήνυμα τύπου elect στον γειτονικό κόμβο που απομένει (από όπου ο v δεν έλαβε μήνυμα).

Στο τέλος υπάρχουν δύο ενδεχόμενα:

1. Είτε κάποια συγκεκριμένη διεργασία v λαμβάνει μηνύματα τύπου elect από όλους τους γειτονικούς της κόμβους πριν προλάβει να αποστείλει το δικό της μήνυμα τύπου elect, ή
2. μηνύματα τύπου elect αποστέλλονται και προς τις δύο κατευθύνσεις μιας συγκεκριμένης ακμής e του δένδρου.

Στην περίπτωση 1 η v αποφασίζει πως είναι αρχηγός. Στην περίπτωση 2 η διεργασία με το μεγαλύτερο id που πρόσκειται στην e εκλέγεται αρχηγός.

Πολυπλοκότητες;

Υπολογισμός ενός Breadth-First Search Δένδρου Επικάλυψης

Θεωρούμε ότι ο γράφος είναι συνδεδεμένος, μη-κατευθυνόμενος, και υπάρχει ένας διακριτός κόμβος p_r που αποτελεί ρίζα (ή πηγή).

Κάθε ακμή e = (i,j) έχει ένα βάρος weight(e) ή weight(i,j) που είναι ένας μη-αρνητικός πραγματικός αριθμός γνωστός και στις δύο διεργασίες που πρόσκεινται στην e.

Πως μπορούμε να τροποποιήσουμε τον αλγόριθμο Flooding ώστε αυτός να κατασκευάζει BFS δένδρο επικάλυψης;

1η Λύση: Αλγόριθμος AsynchBFS

Κώδικας για τη διεργασία p_i

Αρχικά, parent = nill;
    dist = 0 αν $p_i = p_r$ και dist = ∞ αν $p_i \neq p_r$;

```
upon receiving no message:
    if (p_i == p_r) and (parent == nill) then
        send <0> to all neighbors;
        parent = p_i;

upon receiving <M> from neighnor p_j:
    if (M+1 < dist) then
        dist = M+1;
        parent = p_j;
        send <dist> to all neighbors except p_j;
```

Αλγόριθμος AsynchBFS - Κώδικας για τη διεργασία p_i

Θεώρημα: Σε κάθε εκτέλεση του AsynchBFS, το σύστημα τελικά σταθεροποιείται σε μια καθολική κατάσταση στην οποία οι μεταβλητές parent αναπαριστούν ένα BFS δένδρο επικάλυψης.

Απόδειξη (σύνοψη):

Αποδεικνύεται ότι σε κάθε προσβάσιμη καθολική κατάσταση:

1. Για κάθε διεργασία $p_i \neq p_r$, $dist_i$ είναι το μήκος ενός μονοπατιού μ από την p_r στην p_i στο G στο οποίο ο κόμβος που προηγείται του p_i είναι ο $parent_i$.
2. Για κάθε μήνυμα m που βρίσκεται στους inbuf μιας διεργασίας p_i, m είναι το μήκος ενός μονοπατιού μ από την p_r στην p_i. Το ίδιο ισχύει και για τα μηνύματα που βρίσκονται σε κάθε outbuf πίνακα που περιέχει μηνύματα που έχουν αποσταλεί από την p_i.

Αποδεικνύεται επίσης ότι σε κάθε προσβάσιμη καθολική κατάσταση, για κάθε ζεύγος γειτόνων i,j, είτε $dist_j \leq dist_i+1$, το μήνυμα $<dist_i>$ βρίσκεται είτε σε κάποιον από τους outbuf πίνακες της p_i ή σε κάποιον από τους inbuf πίνακες της p_j.

Αλγόριθμος AsynchBFS - Κώδικας για τη διεργασία p_i

Πολυπλοκότητες; Αριθμός μηνυμάτων: O(n*m), # βημάτων O(diam)

Τερματισμός;

Πως μπορώ να χρησιμοποιήσω μηχανισμό με acknowledgments για να επιτύχω τερματισμό;

- Για κάθε μήνυμα αποστέλλεται acknowledgement.
- Κάθε φορά που η p_i λαμβάνει μήνυμα από κάποια γειτονική p_j το οποίο προκαλεί αλλαγή στη μεταβλητή dist και οδηγεί στην αποστολή μηνυμάτων διόρθωσης της απόστασης στις γειτονικές διεργασίες, περιμένει acknowledgments από όλες αυτές τις γειτονικές της διεργασίες πριν στείλει το δικό της acknowledgement στην p_j.
- Απαιτείται κάποιου είδους bookkeeping για να είναι δυνατό να διαχωριστούν διαφορετικά acknowledgements από την ίδια διεργασία.

2η Λύση: Αλγόριθμος LayeredBFS

Το BFS δένδρο επικάλυψης κατασκευάζεται σε επίπεδα.

Κάθε επίπεδο k αποτελείται από τους κόμβους του δένδρου σε βάθος k.

Τα επίπεδο κατασκευάζονται σε φάσεις, ένα επίπεδο σε κάθε φάση και όλες οι φάσεις συγχρονίζονται από τη p_r.

Πρώτη Φάση

Η p_r στέλνει μηνύματα τύπου search σε όλες τις γειτονικές διεργασίες και περιμένει acks από αυτές.

Μια διεργασία που λαμβάνει μήνυμα τύπου search στη φάση 1 αποστέλλει θετικό ack.

Στο τέλος της 1ης φάσης, όλες οι διεργασίες στο επίπεδο 1 καθορίζουν τον πατέρα τους και η p_r γνωρίζει τις θυγατρικές της διεργασίες.

Ας υποθέσουμε επαγωγικά ότι k φάσεις έχουν ήδη εκτελεστεί και k επίπεδα έχουν κατασκευαστεί: κάθε διεργασία σε βάθος το πολύ k γνωρίζει τον πατέρα της και κάθε κόμβος σε βάθος το πολύ k-1 γνωρίζει τα παιδιά του, ενώ η p_r γνωρίζει ότι η φάση k έχει ολοκληρωθεί.

$2^η$ Λύση: Αλγόριθμος LayeredBFS - συνέχεια

Φάση (k+1): Κατασκευή του (k+1)-οστού επιπέδου

Η p_r αποστέλλει μηνύματα τύπου newphase κατά μήκος των ακμών του δένδρου που έχει κατασκευαστεί μέχρι τώρα.

Τα μηνύματα αυτά απευθύνονται προς τις διεργασίες βάθους k. Όταν μια διεργασία βάθους k λαμβάνει μήνυμα τύπου newphase, στέλνει μήνυμα τύπου search σε όλους τους γειτονικούς της κόμβους εκτός από τον πατέρα της και περιμένει acks.

Όταν μια διεργασία $p_j \neq p_r$ λάβει από μια διεργασία p_i για πρώτη φορά μήνυμα τύπου search, σημειώνει την p_i ως πατέρα της και της αποστέλλει θετικό ack. Για τα επόμενα μηνύματα τύπου search που η p_j θα λάβει, θα αποστείλει στον αποστολέα αρνητικό ack.

Κάθε φορά που η p_r λαμβάνει μήνυμα τύπου search αποστέλλει στον αποστολέα πάντα αρνητικό ack.

Όταν μια διεργασία επιπέδου k έχει λάβει acks για όλα τα search μηνύματα που έχει αποστείλει, καθορίζει τα παιδιά της να είναι εκείνοι οι κόμβοι που της έστειλαν θετικό ack. Στη συνέχεια αποστέλλει ack τερματισμού στον πατέρα της.

Τα ack τερματισμού convergecast προς τη ρίζα. Περιέχουν ένα bit που υποδηλώνει αν υπήρχαν διεργασίες στο επίπεδο k+1. Η p_r τερματίζει τον αλγόριθμο αν αυτό το bit είναι 0.

$2^η$ Λύση: Αλγόριθμος LayeredBFS - συνέχεια

Θεώρημα

Ο αλγόριθμος LayeredBFS υπολογίζει ένα BFS δένδρο επικάλυψης.

Πολυπλοκότητες;

	Πολυπλοκότητα Επικοινωνίας	Χρονική Πολυπλοκότητα
AsynchBFS	O(m*n)	O(diam)
LayeredBFS	O(m +n*diam)	O(diam^2)

Εύρεση Συντομότερων Μονοπατιών από μια πηγή σε Γενικά Ασύγχρονα Δίκτυα

Δουλεύει ο αλγόριθμος SynchBellmanFord σε ασύγχρονα συστήματα;

Ποιο πρόβλημα καλείται να επιλύσει ο AsynchBellmanFord (δηλαδή η ασύγχρονη έκδοση του SynchBellmanFord);

Πώς θα επιλυθεί αυτό το πρόβλημα;

Θεώρημα

Σε κάθε εκτέλεση του αλγορίθμου AsynchBellmanFord, το σύστημα τελικά σταθεροποιείται σε μια κατάσταση στην οποία οι μεταβλητές parent αναπαριστούν ένα δένδρο συντομότερων μονοπατιών με ρίζα i_0 και στην οποία οι μεταβλητές dist περιέχουν τις σωστές αποστάσεις των κόμβων από την i_0.

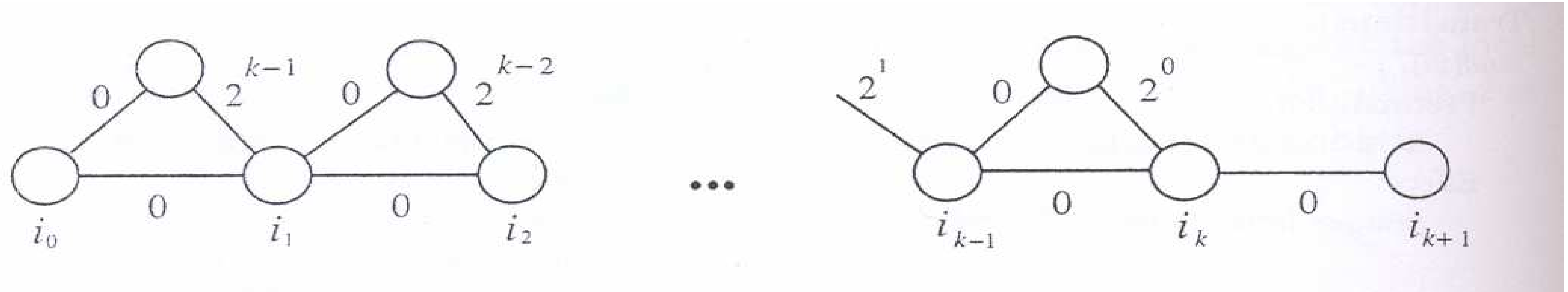
Πολυπλοκότητα;

Αλγόριθμος AsynchBellmanFord

Θεώρημα

Έστω ότι n είναι ένας οποιοδήποτε ζυγός αριθμός, $n \geq 4$. Υπάρχει ένας γράφος n κόμβων με βάρη στις ακμές, στον οποίο ο αλγόριθμος AsynchBellmanFord στέλνει τουλάχιστον $\Omega(c^n)$ μηνύματα για να σταθεροποιηθεί στη χειρότερη περίπτωση, όπου $c > 1$ είναι μια σταθερά.

Απόδειξη: Έστω $k = (n-2)/2$.



Υπάρχει εκτέλεση στην οποία η μεταβλητή dist της διεργασίας i_k παίρνει όλες τις τιμές στο σύνολο $\{2^k -1, 2^k -2, \ldots, 3,2,1,0\}$.

Αλγόριθμος AsynchBellmanFord

Απόδειξη (συνέχεια) - Κακή εκτέλεση

Έστω ότι τα μηνύματα ταξιδεύουν πολύ γρήγορα στα υψηλά μονοπάτια του γράφου.

1. Η πρώτη εκτίμηση του dist από την i_k θα είναι 2^k -1.
2. Το μήνυμα από την i_{k-1} φθάνει στην i_k κατά μήκος του χαμηλού μονοπατιού. Αυτό οδηγεί σε νέα εκτίμηση του dist της i_k σε 2^k-2.
3. Το μήνυμα από την i_{k-2} φθάνει στην i_{k-1} κατά μήκος του χαμηλού μονοπατιού, προκαλώντας το dist της i_{k-1} να αλλάξει από 2^k-2 σε 2^k -4.
4. Η i_{k-1} στέλνει τη νέα εκτίμηση του dist και προς τις δύο κατευθύνσεις.
5. Το υψηλό μονοπάτι είναι και πάλι γρήγορο $\Rightarrow$ η i_k υπολογίζει τη νέα τιμή του dist σε 2^k-3.

Το παραπάνω σενάριο επαναλαμβάνεται.

Χρονική Πολυπλοκότητα;

Υπολογισμός Δένδρου Επικάλυψης Ελάχιστου Βάρους σε Γενικό Ασύγχρονο Σύστημα

Ο γράφος G = (V,E) είναι μη-κατευθυνόμενος, συνδεδεμένος και υπάρχουν βάρη στις ακμές του.

Υποθέτουμε ότι όλα τα βάρη των ακμών είναι διαφορετικά.

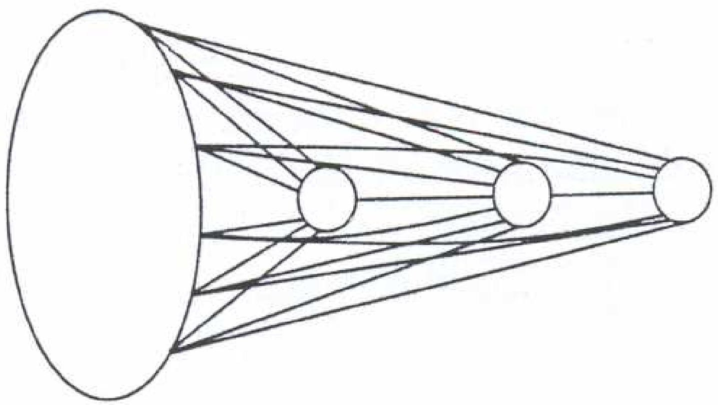
Κάθε διεργασία πρέπει να γνωρίζει κατά τον τερματισμό ένα σύνολο ακμών που είναι γειτονικές της στο MST που θα κατασκευαστεί.

Δυσκολίες στην υλοποίηση του GHS σε ασύγχρονο σύστημα

1. Αν η j γνωρίζει, για τη συνιστώσα στην οποία ανήκει, id διαφορετικό από εκείνο της συνιστώσας της i, πρέπει να είναι εγγυημένο ότι η j δεν ανήκει στην ίδια συνιστώσα με την i. Σε ασύγχρονα συστήματα θα μπορούσε η j να μην έχει ακόμη ενημερωθεί για το id της συνιστώσας στην οποία ανήκει (γιατί κάποια μηνύματα μπορεί να είναι πολύ αργά).

Δυσκολίες στην υλοποίηση του GHS σε ασύγχρονο σύστημα (συνέχεια)

2. O SynchBFS επιτυγχάνει καλή πολυπλοκότητα επικοινωνίας λόγω του συγχρονισμού που επιτυγχάνεται μεταξύ των εποχών κατασκευής. Κάθε συνιστώσα εποχής k αποτελείται από τουλάχιστον 2^k κόμβους και έτσι έχουμε το πολύ logn εποχές. Σε ασύγχρονα συστήματα μπορεί οι συνιστώσες να φτιάχνονται χωρίς καμία εγγύηση για τον αριθμό των κόμβων που ανήκουν σε αυτές. Έτσι, η πολυπλοκότητα επικοινωνίας του αλγορίθμου μπορεί να προκύψει να είναι πολύ μεγαλύτερη.



3. Στον AsynchGHS κάποιες συνιστώσες μπορεί να βρίσκονται σε διαφορετικές εποχές από κάποιες άλλες. Οι ταυτόχρονες αναζητήσεις ακμών ελάχιστου βάρους σε συνιστώσες διαφορετικών εποχών μπορεί να οδηγήσει σε μη-προβλέψιμα αποτελέσματα.

Σκιαγράφηση του AsynchGHS

Αρχικά, κάθε κόμβος είναι μια συνιστώσα. Κάθε συνιστώσα έχει έναν αρχηγό και ένα δένδρο επικάλυψης που είναι υπογράφος του MST (Minimum Spanning Tree).

Οι κόμβοι κάθε συνιστώσας συνεργάζονται για την εύρεση των ακμών ελάχιστου βάρους που εξέρχονται της συνιστώσας:

- ο αρχηγός εκκινεί μια broadcast
- κάθε κόμβος βρίσκει τη δική του εξερχόμενη ακμή ελάχιστου βάρους
- με μια convergecast ο αρχηγός ενημερώνεται για την ακμή ελάχιστου βάρους (MWOE) μεταξύ αυτών, η οποία και θα συμπεριληφθεί στο MST.

Ο αρχηγός στέλνει μήνυμα στις διεργασίες που πρόσκεινται στη MWOE και οι συνιστώσες συγχωνεύονται σε μια.

Η διαδικασία αυτή επαναλαμβάνεται μέχρι όλοι οι κόμβοι να ανήκουν στην ίδια συνιστώσα.

AsynchGHS – Δυσκολίες

1. *Πως μια διεργασία i γνωρίζει αν μια ακμή είναι εξερχόμενη;*

   Χρειάζεται κάποιου είδους συγχρονισμός για να είναι εγγυημένο ότι μια διεργασία j δεν θα απαντήσει στο μήνυμα της i πριν να είναι σίγουρη ότι έχει ενημερωθεί για το σε ποια συνιστώσα ανήκει!

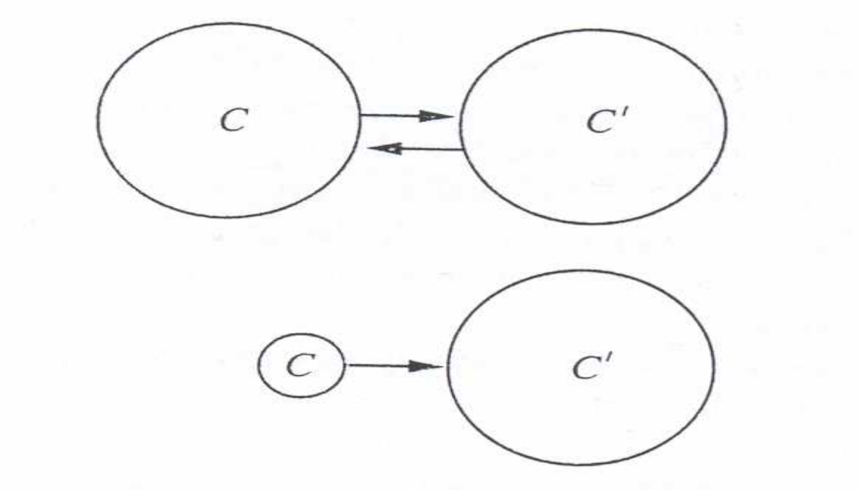
2. *Πως θα καταφέρουμε (δεδομένης της έλλειψης συγχρονισμού) ο αριθμός των εποχών να είναι O(logn);*

   - Θα συσχετίσουμε και πάλι τις συνιστώσες με εποχές.
   - Θα εγγυηθούμε ότι κάθε συνιστώσα εποχής k αποτελείται από τουλάχιστον 2^k κόμβους.
   - Μια συνιστώσα εποχής k+1 θα δημιουργηθεί με συγχώνευση δύο μόνο συνιστωσών εποχής k.

3. *Πως θα ξεπεράσουμε την 3η δυσκολία;*

   Θα υπάρξει κάποιος συγχρονισμός ώστε να αποφευχθεί η «ανάμιξη» ταυτόχρονων αναζητήσεων για ακμές ελάχιστου βάρους από συνιστώσες που βρίσκονται σε διαφορετικές εποχές.

AsynchGHS – Πιο αναλυτικά



Ο AsynchGHS συνδυάζει συνιστώσες με δύο διαφορετικούς τρόπους:

merge: συμβαίνει μόνο μεταξύ δύο συνιστωσών C και C' τ.ω. epoch(C) = epoch(C') και οι C,C' έχουν την ίδια εξερχόμενη ακμή ελάχιστου βάρους.

Η νέα συνιστώσα είναι στην k+1 εποχή.

absorb: συμβαίνει μεταξύ δύο συνιστωσών C και C' τ.ω. epoch(C) < epoch(C') και η ακμή ελάχιστου βάρους του C οδηγεί σε κάποιο κόμβο του C'.

Η συνιστώσα που δημιουργείται θεωρείται μια επέκταση της C' (και όχι μια νέα συνιστώσα) και βρίσκεται στην ίδια εποχή όπως η C'.

AsynchGHS – Πιο αναλυτικά

Λήμμα *: Έστω ότι ξεκινάμε από μια αρχική καθολική κατάσταση όπου κάθε συνιστώσα αποτελείται από ένα μοναδικό κόμβο και είναι στην εποχή 0. Έστω ότι από την αρχική αυτή κατάσταση εφαρμόζουμε μια αυθαίρετη, επιτρεπτή, πεπερασμένη ακολουθία από merge και absorb λειτουργίες. Μετά από την εκτέλεση της ακολουθίας λειτουργιών αυτής, είτε υπάρχει μια μόνο συνιστώσα, ή κάποια merge ή κάποια absorb λειτουργία μπορεί να πραγματοποιηθεί (και είναι ενεργή).

Απόδειξη: Έστω ότι υπάρχουν περισσότερες από μια συνιστώσες μετά από την εκτέλεση κάποιας ακολουθίας merge και absorb λειτουργιών.

Έστω G’ ο κατευθυνόμενος γράφος συνιστωσών. Στον G’ κάθε κόμβος έχει μια εξερχόμενη ακμή και ο μη-κατευθυνόμενος γράφος που αντιστοιχεί στον G’ είναι συνδεδεμένος.

Στον G’ υπάρχει ένας κύκλος μήκους 2 $\Rightarrow$ υπάρχουν συνιστώσες C και C’ των οποίων οι MWOEs πρόσκεινται στους ίδιους κόμβους $\Rightarrow$ πρόκειται για την ίδια ακμή στον G

Αν epoch(C) = epoch(C’) $\Rightarrow$ merge. Διαφορετικά $\Rightarrow$ absorb.

AsynchGHS – Πιο αναλυτικά

Για κάθε συνιστώσα εποχής ≥ 1, υπάρχει μια ακμή που λέγεται βασική (core edge):

- μετά από merge η βασική ακμή της νέας συνιστώσας είναι η κοινή MWOE των δύο συνιστωσών που συγχωνεύτηκαν
- μετά από absorb η βασική ακμή της συνιστώσας είναι η βασική ακμή της συνιστώσας στην μεγαλύτερη εποχή

Για κάθε συνιστώσα, το ζεύγος <βάρος βασικής ακμής συνιστώσας, εποχή συνιστώσας> αποτελεί το όνομα της συνιστώσας.

Η διεργασία με το μεγαλύτερο id από τις διεργασίες που πρόσκεινται στη βασική ακμή είναι ο αρχηγός.

AsynchGHS – Πιο αναλυτικά

Πως μια διεργασία i καθορίζει αν μια διεργασία j με την οποία τη συνδέει μια ακμή είναι στην ίδια συνιστώσα ή όχι;

1. Αν το id της συνιστώσας της j είναι το ίδιο με της i τότε οι i,j είναι στην ίδια συνιστώσα.

Αν ids διαφορετικά:

2. Αν η εποχή στο id της συνιστώσας της j είναι ≥ εκείνου της i τότε η j είναι σε διαφορετική συνιστώσα από την i.
3. Αν η εποχή της j είναι μικρότερη από την εποχή της i, η j καθυστερεί την απάντησή της μέχρι η εποχή της να γίνει τουλάχιστον ίση με εκείνη της i.

AsynchGHS – Πιο αναλυτικά

Μπορεί αυτό να δημιουργήσει πρόβλημα στην πρόοδο του αλγορίθμου;

Επαναλαμβάνουμε το ίδιο επιχείρημα με εκείνο στην απόδειξη του Λήμματος *, αλλά με τον G’ να έχει κόμβους μόνο εκείνες τις συνιστώσες των οποίων η εποχή είναι η μικρότερη δυνατή.

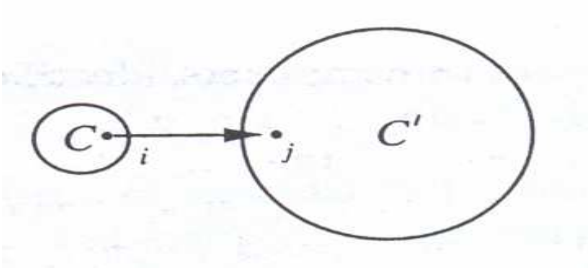
Αν κάποια MWOE ακμή μιας τέτοιας συνιστώσας οδηγεί σε συνιστώσα μεγαλύτερης εποχής $\Rightarrow$ absorb

Διαφορετικά, υπάρχει κύκλος μήκους 2. Άρα, δύο από τις συνιστώσες έχουν την ίδια MWOE $\Rightarrow$ merge

AsynchGHS – Πιο αναλυτικά

Πως θα ξεπεράσουμε την 3η δυσκολία;

Τι συμβαίνει όταν μια συνιστώσα C πρέπει να γίνει absorbed (gets absorbed) από μια άλλη συνιστώσα C’ μεγαλύτερης εποχής ενόσω η C’ ψάχνει για την εξερχόμενη ακμή της ελάχιστου βάρους;



1. Η j δεν έχει ακόμη καθορίσει την MWOE της. Τότε η C συμμετέχει στην αναζήτηση.
2. Η j έχει ήδη καθορίσει την MWOE της (έστω e). Τότε e $\neq$ (i,j) (αφού η e συνδέει την j με κάποια διεργασία σε συνιστώσα που βρίσκεται σε $\geq$ εποχή) $\Rightarrow$ weight(e) < weight(i,j).
3. Η e δεν μπορεί να πρόσκεινται σε κόμβο της C. *Γιατί;*
4. Καμία ακμή της C δεν μπορεί να έχει μικρότερο βάρος $\Rightarrow$ συγχώνευση σωστή!!!!

AsynchGHS – Τύποι Μηνυμάτων

initiate: εκπέμπεται από τον αρχηγό στην αρχή κάθε εποχής και έχει σαν αποτελέσμα οι διεργασίες που το λαμβάνουν να ξεκινήσουν την αναζήτηση των MWOEs τους

report: convergecasts πληροφορία σχετικά με MWOEs προς τον αρχηγό

test: μια διεργασία i στέλνει τέτοιο μήνυμα σε μια άλλη j για να διερευνήσει αν η j βρίσκεται στην ίδια συνιστώσα

accept και **reject**: απάντηση της j σε test μήνυμα (accept αν όχι στην ίδια συνιστώσα, reject διαφορετικά)

changeroot: αποστέλλεται από τον αρχηγό προς τη διεργασία (έστω j) που πρόσκειται στην MWOE της συνιστώσας. Μεταφέρει την πληροφορία ότι η διεργασία πρέπει να προσπαθήσει να συγχωνευτεί με τη συνιστώσα στην οποία ανήκει η διεργασία (έστω i) στο άλλο άκρο της MWOE της.

connect: αποστέλλεται από την j στην i ως ένδειξη προσπάθειας συγχώνευσης των δύο συνιστωσών. Η συγχώνευση γίνεται μόνο αν η i έχει επίσης λάβει changeroot μήνυμα και αν MWOE(i) είναι η εν λόγω ακμή.

AsynchGHS – Τύποι Μηνυμάτων

Κάθε διεργασία i κατηγοριοποιεί τις ακμές της:

- **branch**: ανήκουν ήδη στο MST
- **rejected**: δεν ανήκουν στο MST (γιατί οδηγούν σε κόμβους της ίδιας συνιστώσας)
- **basic**: όλες οι υπόλοιπες

➢ Μηνύματα τύπου test αποστέλλονται μόνο κατά μήκος των ακμών τύπου basic.

➢ Οι basic ακμές ελέγχονται μία προς μία σε αύξουσα διάταξη των βαρών τους.

➢ Όταν μηνύματα τύπου connect μεταδοθούν και προς τις δύο κατευθύνσεις μιας ακμής, πραγματοποιείται συγχώνευση ⇒ νέα βασική (core) ακμή, νέα εποχή, νέος αρχηγός.

➢ Ο νέος leader ξεκινά broadcast στέλνοντας μήνυμα τύπου initiate. Το μήνυμα αυτό ενημερώνει όλες τις διεργασίες της νέας συνιστώσας για το νέο id της συνιστώσας.

➢ Κατά την absorb διαμέσου μιας ακμής (i,j), η j γνωρίζει αν έχει ήδη βρει την MWOE της. Στην όποια περίπτωση η j ξεκινά εκπομπή initiate μηνύματος στους κόμβους της συνιστώσας με μικρότερη εποχή για να μάθουν αυτοί το νέο id.

AsynchGHS

Θεώρημα

Ο αλγόριθμος AsynchGHS επιλύει το πρόβλημα εύρεσης δένδρου επικάλυψης ελάχιστου βάρους σε αυθαίρετο, μη-κατευθυνόμενο γράφο με βάρη στις ακμές.

Απόδειξη

4 διαφορετικές αποδείξεις έχουν προταθεί.

Όλες είναι πολύ πολύπλοκες για να παρουσιαστούν αναλυτικά στην τάξη (ή ακόμη και σε βιβλία)!!!

Η δημιουργία μιας απλής, δομημένης (modular) απόδειξης για τον αλγόριθμο εξακολουθεί να είναι ανοικτό πρόβλημα!!!!

AsynchGHS – Πολυπλοκότητα

Πολυπλοκότητα Επικοινωνίας: O(m + nlogn)

- ➢ O(m): αριθμός μηνυμάτων test-reject
- ➢ Όλα τα υπόλοιπα μηνύματα χρεώνονται στην εύρεση της MWOE.

<u>Σε κάθε εποχή και για κάθε συνιστώσα C:</u>

- Για κάθε διεργασία της C υπάρχει μόνο ένα test-accept ζεύγος μηνυμάτων.
- Αποστέλλονται O(|C|) μηνύματα τύπου initiate-report.
- Ο αριθμός μηνυμάτων changeroot και connect επίσης φράσσονται από O(|C|).

Άρα, ο συνολικός αριθμός μηνυμάτων είναι:

Sum_{C} |C| =

Sum_{k: 0 ≤ k ≤logn} (Sum_{C: epoch(C) = k} |C|) =

Sum_{0}^{logn} n =

n logn

Χρονική Πολυπλοκότητα: O(nlogn)

Ένας απλούστερος αλγόριθμος

Αλγόριθμος SimpleMST

Λειτουργεί με τρόπο παρόμοιο με τον GHS. Ωστόσο:

Κάθε διεργασία διατηρεί μια μεταβλητή local-level, αρχικά 0.

Κάθε διεργασία i για την οποία (local-level == k) προσπαθεί να μη συμμετέχει στον αλγόριθμο για την εύρεση των MWOEs μέχρι όλες οι γειτονικές της διεργασίες να έχουν επίσης local-level == k.

Κάθε φορά που μια διεργασία μαθαίνει ότι ανήκει σε κάποια νέα συνιστώσα ενημερώνει το local-level.