ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Μονότονος Καθαρισμός ενός Μολυσμένου Πλήρους m-αδικού Δέντρου με τη Χρήση Ελάχιστου Αριθμού Ερευνητών σε Λίγες Κινήσεις

Τελώνης Δημήτριος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
Μάρκου Ευριπίδης
Επίκουρος Καθηγητής

Λαμία, 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Μονότονος Καθαρισμός ενός Μολυσμένου Πλήρους m-αδικού Δέντρου με τη Χρήση Ελάχιστου Αριθμού Ερευνητών σε Λίγες Κινήσεις

Τελώνης Δημήτριος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
Μάρκου Ευριπίδης
Επίκουρος Καθηγητής

Λαμία, 2015

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [1], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί* ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. *Δέχομαι ότι η αυτολεξεί παράθεση χωρίς εισαγωγικά, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια.*

4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: 2 Νοεμβρίου 2015

Ο Δηλών

(Υπογραφή)

[1] Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

Μονότονος Καθαρισμός ενός Μολυσμένου Πλήρους m-αδικού Δέντρου με τη Χρήση Ελάχιστου Αριθμού Ερευνητών σε Λίγες Κινήσεις

Τελώνης Δημήτριος

Τριμελής Επιτροπή:

Μάρκου Ευριπίδης, Επίκουρος Καθηγητής

Αδάμ Μαρία, Επίκουρος Καθηγητής

Πλαγιανάκος Βασίλειος, Αναπληρωτής Καθηγητής

Ευχαριστίες

Θα ήθελα πάνω από όλα να ευχαριστήσω θερμά τον επίκουρο καθηγητή του τμήματος κ. Μάρκου Ευριπίδη, για την ανάθεση της συγκεκριμένης πτυχιακής εργασίας, για τη στήριξη και τη διαρκή καθοδήγηση που μου παρείχε, καθώς επίσης και για τις γνώσεις που μοιράστηκε μαζί μου.

Θα ήθελα επίσης να ευχαριστήσω και τα μέλη της επιτροπής, την επίκουρο καθηγητή κ. Αδάμ Μαρία και τον αναπληρωτή καθηγητή κ. Πλαγιανάκο Βασίλειο για τα εποικοδομητικά τους σχόλια.

Περίληψη

Ο υπολογισμός του ελάχιστου αριθμού πρακτόρων (ερευνητών) που απαιτούνται για να καθαρίσουν ένα μολυσμένο δίκτυο είναι ένα διάσημο πρόβλημα στη Θεωρητική Πληροφορική. Η επίλυση αυτού του προβλήματος, που συνήθως ονομάζεται Graph Searching, σε ένα δίκτυο έχει σημαντικές επιπτώσεις στον υπολογισμό ενός σημαντικού χαρακτηριστικού του δικτύου, το οποίο ονομάζεται Pathwidth.

Πιο αναλυτικά, στο πρόβλημα Graph Searching δίνεται ένα γράφημα το οποίο είναι μολυσμένο και ζητείται ο ελάχιστος αριθμός πρακτόρων που μπορούν να καθαρίσουν το γράφημα. Ανάλογα με τις επιτρεπόμενες στρατηγικές κίνησης των πρακτόρων και τους επιτρεπόμενους τρόπους καθαρισμού των ακμών του δικτύου, έχουν εμφανιστεί αρκετές παραλλαγές του προβλήματος. Η επίλυση του αντίστοιχου προβλήματος απόφασης σε οποιαδήποτε παραλλαγή για γενική τοπολογία δικτύου, έχει αποδειχθεί ότι είναι υπολογιστικά δύσκολη (NP-hard).

Το πρόβλημα απόφασης που αντιστοιχεί στον υπολογισμό του εύρους μονοπατιών (Pathwidth) ενός γενικού δικτύου έχει επίσης αποδειχθεί ότι είναι υπολογιστικά δύσκολο (NP-hard). Επιπλέον, δεν έχει βρεθεί γρήγορος αλγόριθμος (πολυωνυμικού χρόνου ως προς το μέγεθος του γραφήματος) που να υπολογίζει μια καλή προσέγγιση για το Pathwidth του γραφήματος. Είναι γνωστό όμως πως η βέλτιστη λύση του προβλήματος Graph Searching σε ένα γράφημα θα μας έδινε αμέσως μια καλή προσέγγιση για το Pathwidth του γραφήματος. Δυστυχώς, στις τοπολογίες γραφημάτων που έχουν μελετηθεί μέχρι τώρα, για τις οποίες ο υπολογισμός του Pathwidth είναι υπολογιστικά δύσκολος, όλες οι παραλλαγές του Graph Searching εκτός από μία είναι επίσης υπολογιστικά δύσκολες. Αυτή η σχετικά νέα παραλλαγή του Graph Searching, ονομάζεται Exclusive Graph Searching και είναι το αντικείμενο μελέτης αυτής της εργασίας.

Ειδικότερα, ασχολούμαστε με την επίλυση του Exclusive Graph Searching σε τοπολογίες γεμάτων m-αδικών δέντρων χρησιμοποιώντας μονότονες στρατηγικές. Ενώ είναι γνωστός ένας αλγόριθμος πολυωνυμικού χρόνου που λύνει το πρόβλημα σε γενικά δέντρα, η υπολογιστική του πολυπλοκότητα είναι αρκετά μεγάλη (αν και πολυωνυμική) και ο ίδιος ο αλγόριθμος είναι αρκετά πολύπλοκος. Εδώ δίνουμε αρκετά πιο απλούς, βέλτιστους αλγόριθμους για γεμάτα δυαδικά και m-αδικά δέντρα που καθαρίζουν σε γραμμικό χρόνο το δέντρο χρησιμοποιώντας τον ελάχιστο αριθμό ερευνητών ο οποίος υπολογίζεται σε σταθερό χρόνο. Αποδεικνύουμε αναλυτικά την ορθότητα και την πολυπλοκότητα κάθε αλγόριθμου καθώς και ότι ο αριθμός των ερευνητών που χρησιμοποιεί κάθε αλγόριθμος είναι ο ελάχιστος.

Η μελέτη και επίλυση του προβλήματος Graph Searching, εκτός από την εφαρμογή στον

υπολογισμό του Pathwidth, έχει επίσης σημαντικές εφαρμογές στα μοντέλα προσομείωσης για την κατάσβεση μιας πυρκαγιάς και γενικά στα μοντέλα προσομείωσης για την εξάλειψη μόλυνσης σε ένα δίκτυο.

Abstract

The computation of the minimum number of agents (searchers) needed to clean a contaminated network is a popular problem in Theoretical Computer Science. The solution of this problem, known as Graph Searching, in a network would help the computation of Pathwidth, which is an important property of the network.

The instance of the Graph Searching problem consists of a contaminated graph and the goal is to compute the minimum number of agents needed to clean the graph. Depending on the movement strategy of the agents and how the edges of the network can be cleaned, a few variations of the problem have appeared in the literature. The solution of the corresponding decision problem in any such variation for arbitrary networks has been proved to be computationally hard (NP-hard).

The corresponding decision problem of the computation of Pathwidth of an arbitrary network has been also proved to be computationally hard (NP-hard). Moreover, no efficient algorithm (i.e., polynomial-time with respect to the size of the graph) for computing a good approximation of Pathwidth has been found. However, it is known that an optimal solution for Graph Searching in a graph would immediately lead to a good approximation for the Pathwidth of this graph. Unfortunately, in all graph topologies studied so far, for which the computation of Pathwidth is NP-hard, all variations of Graph Searching except one, are also NP-hard. We study here this, relatively new, variation of Graph Searching which is called Exclusive Graph Searching.

More specifically, we study the solvability of Exclusive Graph Searching in special tree topologies such as full binary trees and full m-ary trees, using monotone strategies. While a polynomial-time optimal algorithm for the problem in arbitrary trees is known, its computational complexity is high (although polynomial) and the algorithm itself is rather technical and complex. We present here simpler optimal algorithms for full binary and m-ary trees wich decontaminate the network in linear time using a minimum number of agents. The optimal number of agents used by the algorithms can be calculated within a constant time. We rigorously prove the correctness and computational complexity of each algorithm. We also prove that the number of agents used by each algorithm is minimum.

The study and solution of the Graph Searching problem, apart from its application to Pathwidth computation, has also important applications to simulation models for distinguishing a fire and contamination prevention in networks.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Εισαγωγικές Έννοιες

Στο *Graph Searching* [8, 23], μια ομάδα ερευνητών έχει ως στόχο τον καθαρισμό ενός μολυσμένου δικτύου. Ένας ισοδύναμος ορισμός για το ίδιο πρόβλημα δόθηκε από τους Seymour και Thomas [26], σύμφωνα με τον οποίο μια ομάδα ερευνητών στοχεύει στη σύλληψη ενός αόρατου φυγά ο οποίος είναι κρυμμένος μέσα σε ένα δίκτυο και κινείται με άπειρη ταχύτητα. Το παιχνίδι των Seymour και Thomas είναι γνωστό στη βιβλιογραφία ως *helicopter cops and robber*. Εμείς από εδώ και πέρα θα στηριχθούμε στον πρώτο ορισμό, θεωρώντας ότι ένα μολυσμένο δίκτυο αναπαρίσταται από ένα γράφημα G.

Πολλές παραλλαγές έχουν μελετηθεί, οι οποίες διαφέρουν μεταξύ τους σχετικά με τις κινήσεις που επιτρέπονται στους ερευνητές, τους τρόπους με τους οποίους καθαρίζονται οι ακμές του γραφήματος και τους περιορισμούς που επιβάλλονται στις στρατηγικές της έρευνας [12]. Σε κάθε παραλλαγή, το κύριο πρόβλημα συνίσταται στον υπολογισμό του ελάχιστου αριθμού των ερευνητών που απαιτούνται για να καθαριστεί το G ο οποίος ονομάζεται *search number* του G.

Το *Graph Searching* εμφανίζεται για πρώτη φορά το 1967 από τον Breisch, ο οποίος προσπάθησε να μοντελοποιήσει τη διάσωση ενός χαμένου σπηλαιολόγου από μια ομάδα ερευνητών σε ένα πολύπλοκο δίκτυο από σκοτεινές σπηλιές [8]. Λίγα χρόνια αργότερα, το 1976, διατυπώνεται από τον Parsons η πρώτη μαθηματική έκφραση του *Graph Searching*, το οποίο πλέον καθιερώνεται ως ένα παιχνίδι για τον καθαρισμό μολυσμένων δικτύων [23].

Στο *edge Graph Searching*, όπως ορίστηκε από τον Parsons, ένας ερευνητής μπορεί να τοποθετείται σε κόμβους του γραφήματος, να απομακρύνεται από αυτούς ή να ολισθαίνει κατά μήκος αυτών. Κάθε ακμή του γραφήματος θεωρείται καθαρή όταν ένας ερευνητής τη διασχίσει. Μια καθαρή ακμή μολύνεται ξανά όταν υπάρχει μονοπάτι, χωρίς ερευνητή, από αυτή προς μία άλλη μολυσμένη ακμή.

Η πρώτη παραλλαγή του *Graph Searching* έρχεται από τους Κυρούση και Παπαδημητρίου οι οποίοι όρισαν το *node Graph Searching*. Στο *node Graph Searching*, οι μόνες κινήσεις που επιτρέπονται στους ερευνητές είναι η τοποθέτησή τους σε κόμβους του γραφήματος και η απομάκρυνσή τους από αυτούς. Μια ακμή λογίζεται ως καθαρή μόνο όταν δύο ερευνητές βρεθούν ταυτόχρονα στα άκρα της [17]. Στην παραλλαγή αυτή, δεν είναι

δυνατόν να καθαρίσει ένα μονοπάτι με έναν ερευνητή, δεδομένου ότι κάθε φορά που αυτός απομακρύνεται από έναν κόμβο, όλο το μονοπάτι μολύνεται ξανά.

Στη συνέχεια, οι Bienstock και Seymour όρισαν μια άλλη παραλλαγή, το *mixed Graph Searching* [6]. Σε αυτή την παραλλαγή οι επιτρεπόμενες κινήσεις είναι ίδιες με αυτές του *edge Graph Searching* όμως μια ακμή καθαρίζει είτε όταν ένας ερευνητής τη διασχίσει είτε όταν τα δύο της άκρα καταλαμβάνονται ταυτόχρονα από ερευνητή. Να σημειώθεί ότι, η *edge* στρατηγική που περιγράφεται παραπάνω αποτελεί επίσης μια *mixed* στρατηγική.

Πρόσφατα, οι Blin κ.α. εισήγαγαν μια νέα παραλλαγή, το *exclusive Graph Searching*, η οποία φαίνεται να είναι πολύ διαφορετική από τις προηγούμενες [7]. Σε αυτή την παραλλαγή, ένας ερευνητής μπορεί μόνο να ολισθαίνει κατά μήκος των ακμών, ενώ οι τρόποι με τους οποίους καθαρίζει μια ακμή είναι ίδιοι με αυτούς της *mixed* στρατηγικής. Η διαφορά αυτής της παραλλαγής εντοπίζεται σε 2 σημεία. Κάθε κόμβος θα πρέπει να καταλαμβάνεται από το πολύ έναν ερευνητή ενώ ένας ερευνητής δεν μπορεί να μεταπηδήσει από έναν κόμβο σε έναν μη γειτονικό.

Σε κάθε μια από τις *edge*, *node*, *mixed* και *exclusive* στρατηγικές αντιστοιχεί ένα *search number* το οποίο ονομάζεται *edge*, *node*, *mixed* και *exclusive search number* και συμβολίζεται με *es(G)*, *ns(G)*, *s(G)* και *xs(G)* αντίστοιχα, για κάθε γράφημα G.

Σε κάθε μία από της παραπάνω παραλλαγές μπορούμε να εισάγουμε διάφορους περιορισμούς. Για παράδειγμα, ορισμένες φορές θα θέλαμε μια ακμή από τη στιγμή που καθαρίζει να μην μολύνεται ποτέ ξανά. Κάθε στρατηγική που υπακούει σε αυτόν τον περιορισμό ονομάζεται *monotone* (ιδιότητα *monotonicity*) [6, 18]. Επιπλέον, σε ορισμένες περιπτώσεις θέλουμε να απαγορεύεται η μεταπήδηση ενός ερευνητή από έναν κόμβο σε έναν μη γειτονικό του. Μια τέτοια στρατηγική ονομάζεται *internal* (ιδιότητα *internality*) [2]. Τέλος, μια στρατηγική ονομάζεται *connected* αν και μόνο αν το καθαρό μέρος του γραφήματος παραμένει συνεκτικό (ιδιότητα *connectivity*) [2, 3, 4, 9, 10, 11, 13, 14, 22].

1.2 Σχετική Δουλειά

Όπως είδαμε νωρίτερα, το κύριο ερώτημα που τίθεται σε κάθε μια από τις παραλλαγές του *Graph Searching* είναι το ακόλουθο: Δοθέντος ενός γραφήματος G, ποιο είναι το *search number* του G; Ποιος είναι, δηλαδή, ο ελάχιστος αριθμός από ερευνητές που απαιτείται για τον καθαρισμό του γραφήματος; Έχει αποδειχθεί ότι το πρόβλημα απόφασης που αντιστοιχεί στο προηγούμενο ερώτημα, το οποίο αποτελεί το πρόβλημα βελτιστοποίησης, είναι *NP-complete* για κάθε μία από τις *edge*, *node* και *mixed* στρατηγικές [20, 17, 6]. Έχει επίσης αποδειχθεί ότι τα *edge*, *node* και *mixed Graph Searching* δεν διαφέρουν περισσότερο από μία σταθερά, αφού για κάθε γράφημα G ισχύει $ns(G)\text{-}1 \leq es(G) \leq ns(G)+1$ [17] και $s(G) \leq ns(G) \leq s(G)+1$ [6]. Κάθε μια από τις *edge*, *node* και *mixed* στρατηγικές είναι *monotone*, δηλαδή για κάθε γράφημα G ισχύει $es(G) = mes(G)$ [18], $ns(G) = mns(G)$ [17] και $s(G) = ms(G)$ [6].

Αποδεικνύεται ότι το *connected Graph Searching* δεν είναι *monotone* στη γενική περίπτωση [27]. Αυτό σημαίνει ότι υπάρχουν γραφήματα G για τα οποία ισχύει $cs(G) < mcs(G)$. Αντίθετα, αποδεικνύεται ότι το *connected Graph Searching* είναι *monotone* για κάθε δέντρο T, δηλαδή $cs(T) = mcs(T)$.

Στο *exclusive Graph Searching* τα πράγματα φαίνεται να διαφέρουν αρκετά σε σχέση με τις προηγούμενες παραλλαγές (*edge*, *node* και *mixed Graph Searching*). Παρατηρείται ότι το *exclusive search number* ενός γραφήματος μπορεί να διαφέρει εκθετικά σε σχέση με τις τιμές των κλασσικών *search numbers* (*edge*, *node* και *mixed search number*). Για παράδειγμα, σε έναν αστέρα ισχύει *xs(G) = n-2*, όπου n ο αριθμός των κόμβων του, ενώ *es(G) = ns(G) = s(G) = 2* (δες κεφάλαιο 2). Επίσης, σε ένα δέντρο το *exclusive search number* μπορεί να είναι γραμμικό ως προς τον αριθμό των κόμβων του, ενώ όλα τα κλασσικά *search numbers* στα δέντρα είναι *O(logn)* [24]. Αποδεικνύεται ότι το *exclusive Graph Searching* δεν είναι *monotone*, δηλαδή υπάρχουν γραφήματα G για τα οποία ισχύει *xs(G) < mxs(G)* [7]. Έχει επίσης αποδειχθεί ότι το *exclusive Graph Searching* στα δέντρα δεν είναι *monotone* [7]. Αποδεικνύεται ακόμη ότι υπάρχουν γραφήματα G και H για τα οποία ισχύει *xs(G) < xs(H)* και το H είναι υπογράφημα του G [7]. Έχει όμως αποδειχθεί ότι για κάθε δέντρο T και κάθε υποδέντρο T' του T, ισχύει *xs(T) ≥ xs(T')* [7]. Γνωρίζουμε επίσης ότι, για κάθε συνεκτικό γράφημα G με μέγιστο βαθμό Δ ισχύει *ns(G)-1 ≤ xs(G) ≤ (Δ-1) · ns(G)* [7]. Επιπλέον, αν Δ≤3 τότε *ns(G)-1 ≤ xs(G) ≤ ns(G)* [7]. Ακόμη, για κάθε δέντρο T με μέγιστο βαθμό Δ≥2 ισχύει *xs(T) > Δ-2* [7]. Έχει επίσης δοθεί ένας πολυωνυμικού χρόνου αλγόριθμος ο οποίος δοθέντος ενός δέντρου T υπολογίζει το *exclusive search number* του T [7], καθώς επίσης και μια πολυωνυμικού χρόνου *exclusive* στρατηγική η οποία χρησιμοποιεί *xs(T)* ερευνητές για τον καθαρισμό του T [7].

1.3 Αποτελέσματα

Αρχικά, προτείνουμε έναν γραμμικού χρόνου αλγόριθμο ο οποίος δέχεται ως είσοδο ένα πλήρες δυαδικό δέντρο T_1 και το καθαρίζει χρησιμοποιώντας μια *exclusive monotone* στρατηγική, και δίνουμε έναν σταθερού χρόνου αλγόριθμο για τον υπολογισμό του *xms(T_1)*. Αποδεικνύουμε ότι ο αριθμός των ερευνητών που χρησιμοποιεί ο αλγόριθμος είναι $\lceil \frac{n}{3} \rceil$, όπου n ο αριθμός των κόμβων του δέντρου, και ότι ο αριθμός αυτός είναι ο ελάχιστος. Θα αποδείξουμε επίσης ότι οι συνολικές κινήσεις που απαιτούνται από τους ερευνητές για να καθαριστεί το δέντρο είναι $2 \cdot \lfloor \frac{n}{3} \rfloor$. Ακόμα, δείχνουμε ότι για ένα πλήρες δυαδικό δέντρο άρτιου βαθμού αρκούν $log(n+1) - 1$ βήματα για τον καθαρισμό του, ενώ για ένα πλήρες δυαδικό δέντρο περιττου βαθμού αρκούν $log(n+1)$ βήματα.

Έπειτα, δίνουμε έναν γραμμικού χρόνου αλγόριθμου ο οποίος δέχεται ως είσοδο ένα πλήρες m-αδικό δέντρο T_2 και το καθαρίζει χρησιμοποιώντας μια *exclusive monotone* στρατηγική, καθώς επίσης και έναν σταθερού χρόνου αλγόριθμο ο οποίος υπολογίζει το *xms(T_2)*. Αποδεικνύουμε ότι ο αριθμός των ερευνητών που χρησιμοποιεί ο αλγόριθμος είναι $\lceil \frac{(m-1)n}{m+1} \rceil$ και ότι ο αριθμός αυτός είναι ο ελάχιστος. Αποδεικνύουμε ακόμη ότι οι συνολικές κινήσεις που απαιτούνται, σύμφωνα με τον αλγόριθμο, από τους ερευνητές για να καθαριστεί το δέντρο είναι $2 \cdot \lfloor \frac{n}{m+1} \rfloor$. Τέλος, δείχνουμε ότι για ένα πλήρες m-αδικό δέντρο άρτιου ύψους αρκούν $\lfloor logn \rfloor$ βήματα για τον καθαρισμό του, ενώ για ένα πλήρες m-αδικό δέντρο περιττού ύψους αρκούν $\lceil logn \rceil$ βήματα.

Κεφάλαιο 2

Το Μοντέλο

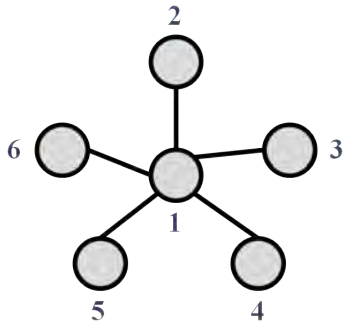
Το μοντέλο που θα χρησιμοποιήσουμε στην εργασία είναι το *exclusive Graph Searching*. Σε μια *exclusive* στρατηγική $k \leq n$ ερευνητές τοποθετούνται σε k διαφορετικούς κόμβους ενός γραφήματος G και πραγματοποιούν μια ακολουθία κινήσεων, όπου n το πλήθος των κόμβων του G. Μια κίνηση μπορεί να θεωρηθεί η ολίσθηση ενός ερευνητή από το ένα άκρο u μιας ακμής $e = \{u, v\}$ στο άλλο άκρο v. Όμως, στο μοντέλο αυτό δεν επιτρέπεται η ταυτόχρονη συνύπαρξη 2 ή περισσότερων ερευνητών σε έναν κόμβο (*exclusivity*) και επομένως για να θεωρηθεί έγκυρη η προηγούμενη κίνηση θα πρέπει στον κόμβο v να μην βρίσκεται άλλος ερευνητής. Αυτή είναι και η πρώτη διαφορά του μοντέλου αυτού σε σχέση με τις κλασσικές παραλλαγές (*edge*, *node* και *mixed* παραλλαγές). Μια άλλη διαφορά συνίσταται στο ότι απαγορεύεται η μεταπήδηση ενός ερευνητή από έναν κόμβο u σε έναν κόμβο v, αν ο v δεν είναι γειτονικός (*internality*). Με βάση αυτόν τον περιορισμό, η δυσκολία εντοπίζεται όταν ένας ερευνητής πρέπει να πάει από έναν κόμβο u σε έναν μη γειτονικό κόμβο v και όλα τα μονοπάτια από τον u στον v περιλαμβάνουν έναν κόμβο ο οποίος καταλαμβάνεται από ερευνητή.

Οι ακμές του γραφήματος G θεωρούνται αρχικά μολυσμένες. Μια ακμή $e = \{u, v\}$ καθαρίζεται με 2 τρόπους: είτε όταν ένας ερευνητής διασχίσει την e κατά μήκος, είτε όταν τα άκρα u και v της e καταλαμβάνονται ταυτόχρονα από ερευνητή. Μια ακμή μολύνεται ξανά αν υπάρχει μονοπάτι χωρίς ερευνητές, από την ακμή αυτή σε μια άλλη μολυσμένη ακμή. Η στρατηγική είναι νικηφόρα εάν έχει σαν αποτέλεσμα όλες οι ακμές του γραφήματος G να είναι ταυτόχρονα καθαρές.

Στο μοντέλο μας εισάγουμε επίσης και την ιδιότητα της μονοτονίας (*monotonicity*) και έτσι η στρατηγική που θα ακολουθήσουμε εκτός από *exclusive* θα είναι και *monotone*. Αυτό σημαίνει ότι κάθε ακμή που καθαρίζουμε δεν επιτρέπεται να μολυνθεί ξανά και επομένως το σύνολο των καθαρών ακμών δεν μειώνεται κατά τη διάρκεια του καθαρισμού.

Αναφέραμε στο κεφάλαιο 1 ότι το *exclusive Graph Searching* φαίνεται να διαφέρει αρκετά σε σχέση με τις κλασσικές *edge*, *node* και *mixed* παραλλαγές. Ας θεωρήσουμε έναν αστέρα S με n κόμβους (Σχήμα 2.1). Σε μια από τις κλασικές παραλλαγές (*edge*, *node* ή *mixed*) αρκούν 2 ερευνητές για να καθαρίσουν τον αστέρα (δηλαδή $es(S) = ns(S) = s(S) = 2$). Ένας ερευνητής ο οποίος θα τοποθετηθεί στον κεντρικό κόμβο και ένας δεύτερος ερευνητής ο οποίος είτε θα πηδάει από το ένα φύλλο στο άλλο, είτε θα ολισθαίνει από το ένα

φύλλο στο άλλο περνώντας πρώτα από το κεντρικό κόμβο. Από την άλλη, σε μια *exclusive* παραλλαγή χρειάζονται n-2 ερευνητές (δηλαδή *xs(S) = n-2*) οι οποίοι τοποθετούνται αρχικά σε n-2 φύλλα. Αν v το φύλλο που δεν έχει ερευνητή τότε ένας από τους ερευνητές ολισθαίνει προς τον κεντρικό κόμβο και από εκεί στο v.



Σχήμα 2.1

Ένας από τους λόγους δημιουργίας του *exclusive Graph Searching* σχετίζεται με το *pathwidth* [25, 16, 1]. Έχει αποδειχθεί ότι το *pathwidth* συνδέεται άμεσα με το *node search number* με τη σχέση *mns(G) = pw(G)+1* [5], για κάθε γράφημα G. Επιπλέον, τα *edge* και *mixed search numbers* δεν διαφέρουν περισσότερο από μία σταθερά από το *node search number* [17, 6], και επομένως ούτε και από το *pathwidth*. Το *pathwidth* από την άλλη, έχει αποδειχθεί ότι είναι *NP-hard* για αρκετές κλάσεις γραφημάτων [15, 21]. Οπότε, θα θέλαμε για κάποιες από αυτές τις κλάσεις το *search number* να υπολογίζεται σε πολυωνυμικό χρόνο αφού τότε θα είχαμε και έναν πολυωνυμικού χρόνου αλγόριθμο και για το *pathwidth*. Όμως, σε καμία από τις *edge*, *node* και *mixed* παραλλαγές δεν έχουν βρεθεί κλάσεις γραφημάτων στις οποίες η πολυπλοκότητα του *pathwidth* και του *search number* να διαφέρουν. Η πρώτη παραλλαγή του *Graph Searching* για την οποία η πολυπλοκότητα του *pathwidth* διαφέρει από αυτή του *search number* είναι το *exclusive Graph Searching*. Ειδικότερα, έχει αποδειχθεί ότι υπάρχουν κλάσεις γραφημάτων για τις οποίες το *pathwidth* είναι *NP-complete* και το *exclusive search number* πολυωνυμικό και αντίστροφα [19]. Όμως, δεν υπάρχει ακόμα σχέση που να συνδέει το *pathwidth* με το *exclusive search number*.

Κεφάλαιο 3

Καθαρισμός Πλήρους Δυαδικού Δέντρου

Στο κεφάλαιο αυτό θα ασχοληθούμε με τον καθαρισμό ενός πλήρους δυαδικού δέντρου. Συγκεκριμένα, θα δούμε τη διαδικασία με την οποία θα καθαρίσουμε ένα πλήρες δυαδικό δέντρο με τη χρήση μιας exclusive μονότονης στρατηγικής, και θα αναπτύξουμε έναν αλγόριθμο. Έπειτα, θα υπολογίσουμε τον αριθμό ερευνητών που χρησιμοποιεί ο αλγόριθμος για τον καθαρισμό του δέντρου, το συνολικό αριθμό μεταθέσεων των ερευνητών κατά τη διάρκεια του καθαρισμού, καθώς επίσης και τα βήματα που απαιτούνται για τον καθαρισμό του δέντρου σύμφωνα πάντα με τον αλγόριθμο.

3.1 Περιγραφή Αλγορίθμου

3.1.1 Ο Αλγόριθμος

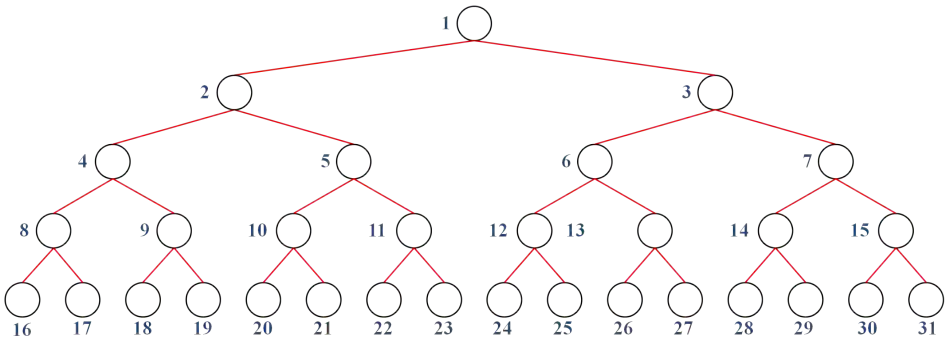
Η διαδικασία του καθαρισμού έχει ως εξής: Ανά 2 επίπεδα, ξεκινώντας από τα φύλλα και μέχρι να φτάσουμε στη ρίζα, τοποθετούμε έναν ερευνητή σε κάθε κόμβο του επιπέδου που είναι αριστερό παιδί (σε περίπτωση που το δέντρο έχει άρτιο ύψος, τοποθετούμε έναν ερευνητή και στη ρίζα του δέντρου). Η διαδικασία του καθαρισμού ολοκληρώνεται εφόσον κάθε ερευνητής εκτελέσει 2 κινήσεις. Κατά την πρώτη κίνηση κάθε ερευνητής μεταφέρεται στον πατέρα του κόμβου του ενώ προτεραιότητα έχει ο ερευνητής που τοποθετήθηκε νωρίτερα. Μόλις οι ερευνητές ολοκληρώσουν την πρώτη τους κίνηση, περνάμε στη δεύτερη και τελευταία κίνηση στην οποία κάθε ερευνητής μεταφέρεται στο δεξί παιδί του κόμβου του ενώ η προτεραιότητα εδώ ανήκει στον ερευνητή που κινήθηκε αργότερα από τους υπόλοιπους στο προηγούμενο βήμα. Ένας απλός αλγόριθμος που εκτελεί την παραπάνω διαδικασία είναι ο ακόλουθος:

1. **Για κάθε** επίπεδο s **από** height **μέχρι** 0 **με βήμα** -2
2. **Για κάθε** κόμβο n του επιπέδου s
3. **Αν** ο κόμβος n είναι αριστερό παιδί ή ρίζα **τότε**
4. Βάλε ερευνητή στον κόμβο n
5. **Αν** ο κόμβος n είναι αριστερό παιδί **τότε**
6. Κίνηση ερευνητή στον πατέρα του κόμβου n

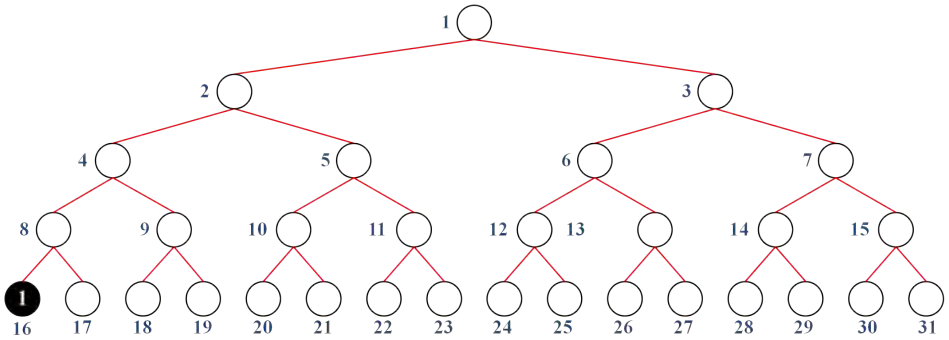
7. **Για κάθε** επίπεδο s **από** 1-(height ***mod*** 2) **μέχρι** height-1 **με βήμα** 2
8. **Για κάθε** κόμβο n του επιπέδου s
9. Κίνηση ερευνητή στο δεξί παιδί του κόμβου n

3.1.2 Παράδειγμα Εκτέλεσης Αλγορίθμου

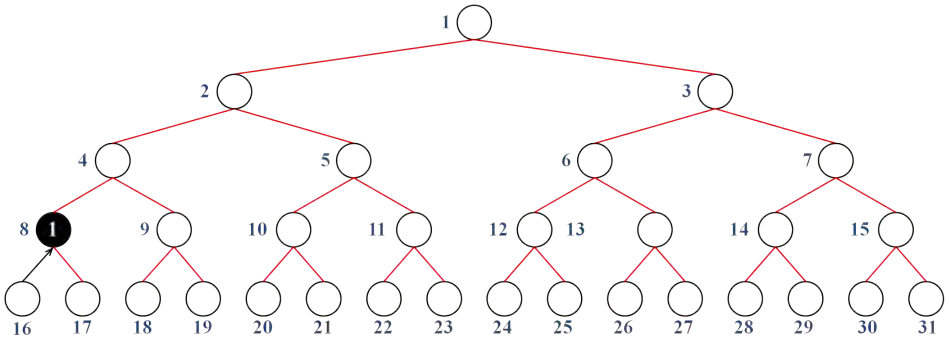
- Ας υποθέσουμε ότι το δέντρο που θέλουμε να καθαρίσουμε έχει ύψος 4. Αυτό σημαίνει ότι θα αποτελείται από 31 κόμβους, στους οποίους αρχικά δεν υπάρχει τοποθετημένος ερευνητής.



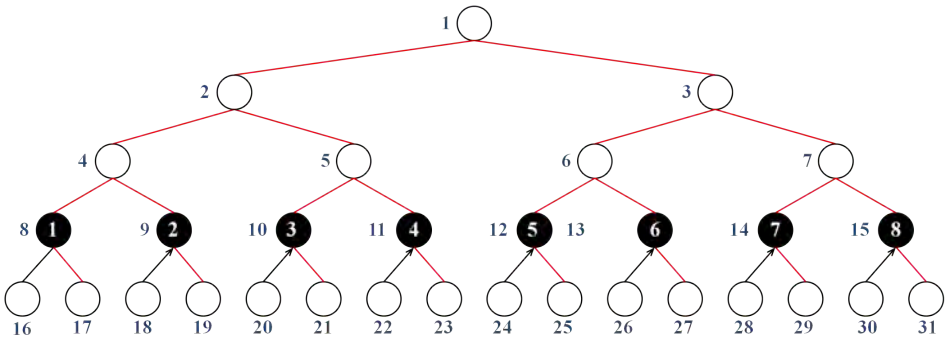
- Ο καθαρισμός ξεκινάει από το επίπεδο 4 του δέντρου. Ο πρώτος ερευνητής τοποθετείται στον κόμβο 16 του δέντρου.



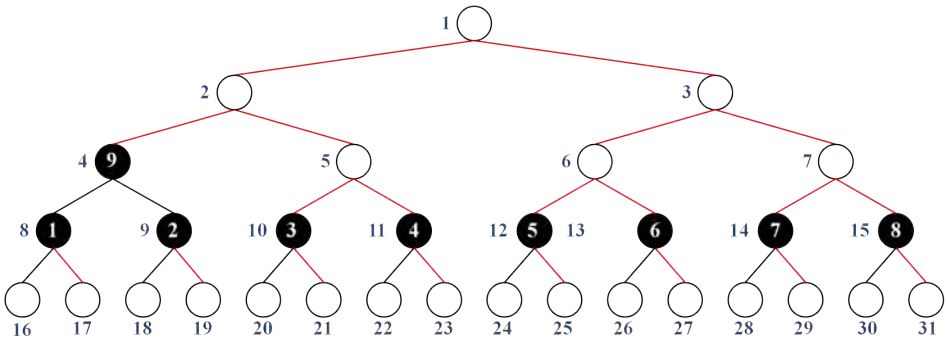
- Ο ερευνητής μετακινείται από τον κόμβο 16 στον κόμβο 8.



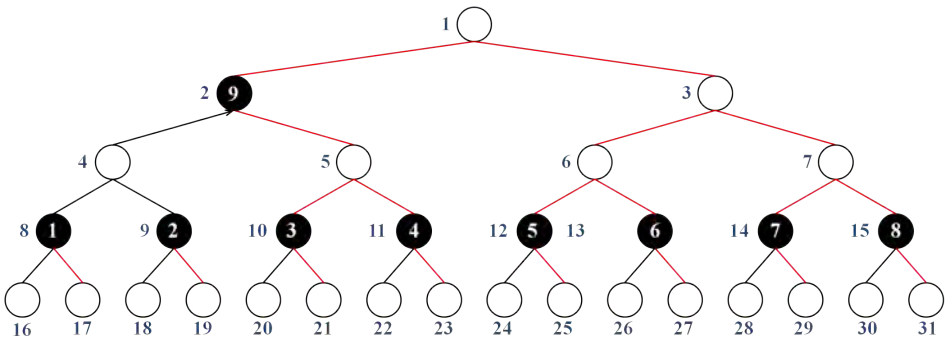
- Κάθε ερευνητής που τοποθετείται σε κόμβο του τέταρτου επιπέδου διασχίζει την ακμή που συνδέει τον κόμβο με τον πατέρα του.



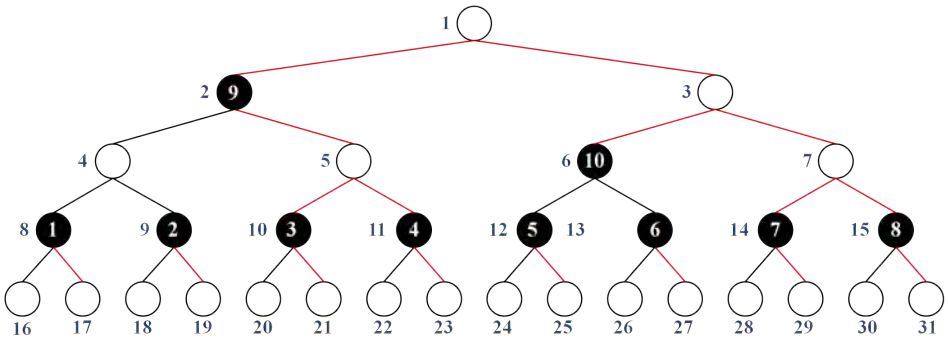
- Νέο επίπεδο κίνησης είναι το επίπεδο 2. Ένας ακόμη ερευνητής τοποθετείται στον κόμβο 4.



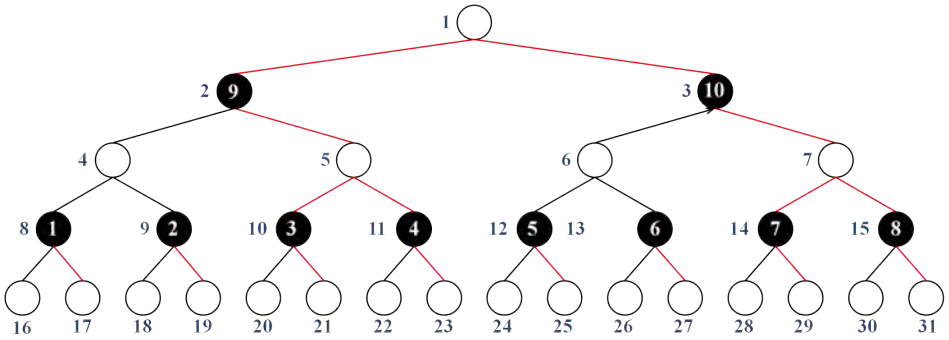
- Ο ερευνητής μετακινείται από τον κόμβο 4 στον κόμβο 2.



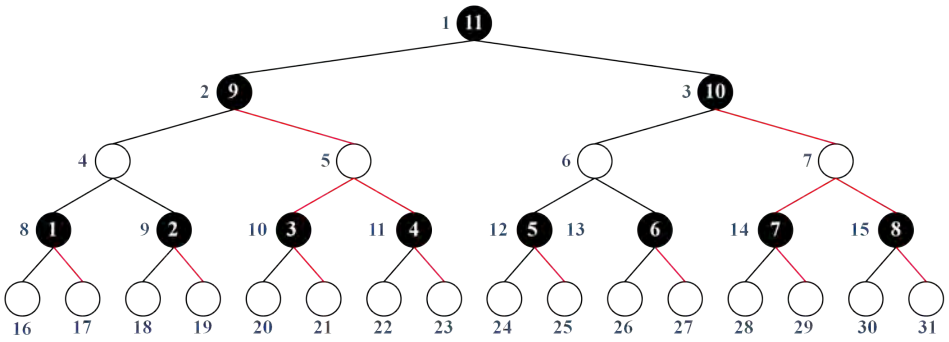
- Επόμενος κόμβος στον οποίο τοποθετείται ερευνητής είναι ο κόμβος 6.



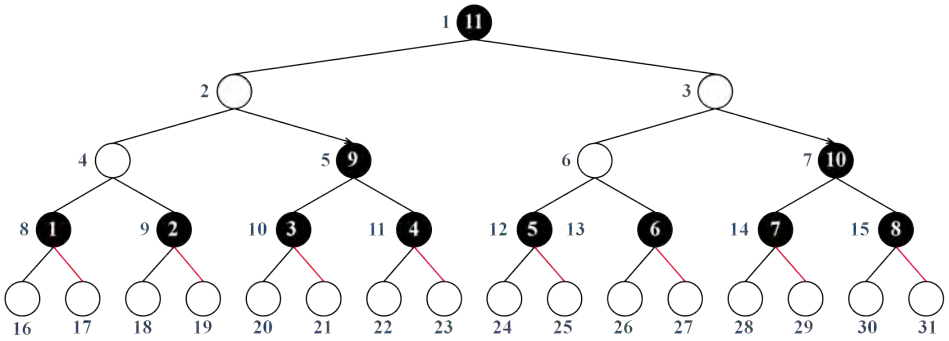
- Ο ερευνητής διασχίζει την ακμή που συνδέει τον κόμβο 6 με τον κόμβο 3.



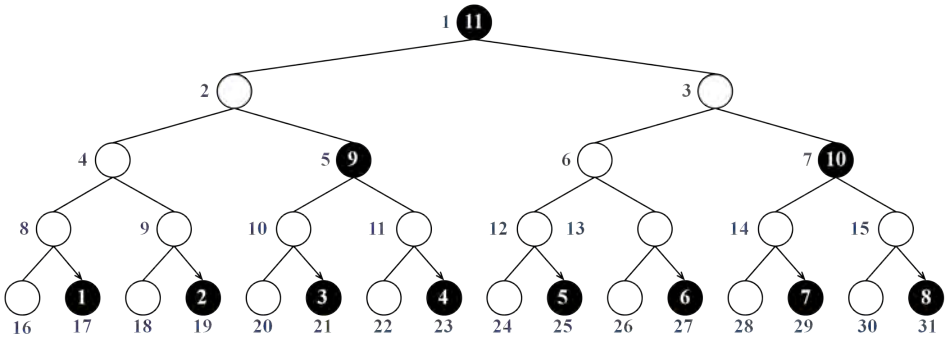
- Ένας επιπλέον ερευνητής τοποθετείται στη ρίζα του δέντρου και παραμένει ακίνητος.



- Επόμενο επίπεδο κίνησης είναι το επίπεδο 1. Κάθε ερευνητής αυτού του επιπέδου, διασχίζει την ακμή που ενώνει τον κόμβο του με το δεξί του παιδί.



- Τελευταίο επίπεδο κίνησης είναι το επίπεδο 3. Κάθε ερευνητής αυτού του επιπέδου, διασχίζει την ακμή που ενώνει τον κόμβο του με το δεξί του παιδί.



3.2 Υλοποίηση Αλγορίθμου

Στην ενότητα αυτή υλοποιούμε τον αλγόριθμο της παραγράφου 3.1.1, θεωρώντας όμως ότι σε κάθε βήμα μπορεί να κινηθεί μόνο ένας ερευνητής.

Αλγόριθμος 1 *Καθαρισμός πλήρους δυαδικού δέντρου*

Είσοδος: *πλήθος κόμβων n*
Έξοδος: *-*

1: $h \leftarrow log_2(n+1) - 1$
2: ***for*** *(*$i = h$*, *$i = i - 2$*, while* $i \geq 0$*)* ***do***
3: *lmost* $\leftarrow 2^i$
4: ***for*** *(*$u = lmost$*, *$u = u + 2$*, while* $u < 2 * lmost$*)* ***do***
5: *ένας νέος ερευνητής* s_u *τοποθετείται στον κόμβο u*
6: *{u, 2*u}* $\leftarrow$ *καθαρή*
7: *{u, 2*u+1}* $\leftarrow$ *καθαρή*
8: ***if*** *(ο κόμβος u δεν είναι ρίζα)* ***then***
9: *ο ερευνητής* s_u *πηγαίνει από τον κόμβο u στον κόμβο u/2*
10: *{u, u/2}* $\leftarrow$ *καθαρή*
11: ***end if***
12: ***end for***
13: ***end for***
14: ***for*** *(*$i = 1 - (h \textbf{mod} 2)$*, *$i = i + 2$*, while* $i < h$*)* ***do***
15: *lmost* $\leftarrow 2^i$
16: ***for*** *(*$u = lmost$*, *$u++$*, while* $u < 2 * lmost$*)* ***do***
17: *ο ερευνητής* s_u *πηγαίνει από τον κόμβο u στον κόμβο 2*u+1*
18: *{u, 2*u+1}* $\leftarrow$ *καθαρή*
19: ***end for***
20: ***end for***

Ανά 2 επίπεδα, από τα φύλλα προς τη ρίζα, επιλέγουμε έναν-έναν τους κόμβους του αντίστοιχου επιπέδου που ειναι αριστερά παιδιά. Σε κάθε κόμβο που επιλέγουμε τοποθετούμε 1 ερευνητή και έτσι οι ακμές που συνδέουν τον κόμβο αυτόν με τα παιδιά του θα καθαρίσουν, αφού στα άκρα τους θα βρίσκονται ερευνητές. Στη συνέχεια ο ερευνητής αυτός διασχίζει και επομένως καθαρίζει την ακμή που συνδέει τον κόμβο του με τον πατερά του (εξαιρείται ο ερευνητής που έχει τοποθετηθεί στη ρίζα).

Αφού ολοκληρωθει αυτή η διαδικασία, ακολουθούμε αντίστροφη πορεία στο δέντρο και ανά 2 επίπεδα, ξεκινώντας από το επίπεδο στο οποίο βρίσκεται ο ερευνητής που κινήθηκε τελευταίος και μέχρι τα φύλλα, επιλέγουμε έναν-έναν τους κόμβους κάθε επιπέδου. Ο ερευνητής που βρίσκεται στον αντίστοιχο κόμβο διασχίζει και καθαρίζει την ακμή που συνδέει τον κόμβο του με το δεξί παιδί.

Οι ακόλουθες συναρτήσεις επιστρέφουν το ύψος ενός πλήρους m-αδικού δέντρου (το οποίο υπολογίζουμε στη γραμμή 1 του αλγορίθμου, δίνοντας στην παράμετρο m της συνάρτησης την τιμή 2) και τη δύναμη a^b (a, b: φυσικοί αριθμοί), και ολοκληρώνουν την εκτέλεσή τους μετά από *Θ(*$log_m n$*)* και *Θ(b)* βήματα αντίστοιχα:

Συνάρτηση 1 *Υπολογισμός ύψους πλήρους m-αδικού δέντρου*

Είσοδος: *κάτω βαθμός m, πλήθος κόμβων n*
Έξοδος: *το ύψος του δέντρου*

```
1: function height(m,n)
2:     while (n > 1) do
3:         height ← height+1
4:         n ← n div 2
5:     end while
6:     return height
7: end function
```

Συνάρτηση 2 *Υπολογισμός του a^b, όπου a και b φυσικοί αριθμοί*

Είσοδος: *2 φυσικοί αριθμοί a, b*
Έξοδος: a^b

```
1: function power(a,b)
2:     power ← 1
3:     for (i = 0, i++, while i < b) do
4:         power ← a*power
5:     end for
6:     return power
7: end function
```

3.3 Ανάλυση Αλγορίθμου

Λήμμα 3.3.1. *(Ορθότητα αλγορίθμου) Ο αλγόριθμός μας (***Αλγόριθμος 1***) καθαρίζει ένα πλήρες δυαδικό δέντρο n κόμβων.*

Απόδειξη.

- Ας θεωρήσουμε ότι οι ερευνητές έχουν τοποθετηθεί στο δέντρο βάσει της περιγραφής που κάναμε στην παράγραφο 3.1.1. Στον αλγόριθμό μας η διδικασία του καθαρισμού ολοκληρώνεται αφού κάθε ερευνητής συμπληρώσει 2 κινήσεις. Στην πρώτη κίνηση μετακινούμε όλους τους ερευνητές κάθε επιπέδου, ξεκινώντας από το χαμηλότερο επίπεδο, στον πατέρα του κόμβου τους. Ας δούμε όμως αν αυτή η κίνηση αυτή μπορεί να πραγματοποιηθεί χωρίς να έχουμε επαναμόλυνση κάποιας ακμής. Οι ερευνητές που βρίσκονται στο επίπεδο των φύλλων (έστω h) έχουν μόνο 1 προσπίπτουσα μολυσμένη ακμή στον κόμβο τους, γεγονός που τους επιτρέπει να μεταβούν στον πατέρα του που βρίσκεται στο επίπεδο h-1 καθαρίζοντας την αντίστοιχη ακμή. Έτσι, επειδή οι ερευνητές του επιπέδου h καταλαμβάνουν πλέον όλους τους κόμβους του επιπέδου h-1, οι μολυσμένες ακμές που ενώνουν τους κόμβους του επιπέδου h-1 με τους κόμβους του επιπέδου h-2 στους οποίους υπάρχει ερευνητής καθαρίζουν αυτόματα. Βάσει αυτής της ανάλυσης, οι ερευνητές του επόμενου επιπέδου κίνησης (επίπεδο h-2) έχουν 1 προσπίπτουσα ακμή στον κόμβο τους και άρα μπορούν να κινηθούν, στη συνέχεια οι ερευνητές του επιπέδου h-4 μπορούν και αυτοί να κινηθούν κ.ο.κ.. Όμως, σε κάθε επίπεδο η κίνηση των ερευνητών δεν εξαρτάται από την κίνηση άλλων ερευνητών του επιπέδου και επομένως κάθε ερευνητής που τοποθετείται σε κόμβο μπορεί να κινηθεί αμέσως, ενώ αν το δέντρο είναι άρτιου ύψους ο ερευνητής που θα τοποθετηθεί στη ρίζα παραμένει ακίνητος αφού δεν θα υπάρχει στον κόμβο του μολυσμένη ακμή για να καθαρίσει.

- Στη δεύτερη κίνηση, όλοι οι ερευνητές που εκτέλεσαν την πρώτη τους κίνηση μεταφέρονται στο δεξί παιδί του κόμβου τους. Ας δούμε τώρα αν αυτή η κίνηση μπορεί να πραγματοποιηθεί χωρίς να μολυνθούν ξανά ακμές που έχουμε ήδη καθαρίσει. Το πρώτο επίπεδο για τη δεύτερη κίνηση των ερευνητών είναι το επίπεδο 1-(h***mod***2) στο οποίο βρίσκεται ο ερευνητής που ολοκλήρωσε τελευταίος την πρώτη του κίνηση. Στο επίπεδο αυτό, εξαιτίας της πρώτης κίνησης των ερευνητών, προσπίπτει σε κάθε κόμβο 1 μόνο μολυσμένη ακμή την οποία κάθε ερευνητής μπορεί να καθαρίσει. Έτσι, θα κινηθούν στο δεξί παιδί του κόμβου τους, δηλαδή στο επίπεδο 1-(h***mod***2)+1 και αυτόματα καθαρίζουν και οι ακμές που συνδέουν τους κόμβους του επιπέδου 1-(h***mod***2)+1 στους οποίους βρίσκονται πλέον οι ερευνητές με τους κόμβους του επιπέδου 1-(h***mod***2)+2. Εξαιτίας αυτής της κίνησης μπορούν πλέον να κινηθούν οι ερευνητές του επιπέδου 1-(h***mod***2)+2 στο επίπεδο 1-(h***mod***2)+3, έπειτα οι ερευνητές του επιπέδου 1-(h***mod***2)+4 στο επίπεδο 1-(h***mod***2)+5 κ.ο.κ. αφού κάθε φορά στον αντίστοιχο κόμβο θα προσπίπτει 1 μολυσμένη ακμή.

- Στη γραμμή 1 του αλγορίθμου υπολογίζουμε το ύψος του δέντρου, εφόσον γνωρίζουμε τον αριθμό των κόμβων του.

- Στη γραμμή 2 επιλέγουμε εναλλάξ τα επίπεδα, από το επίπεδο των φύλλων (επίπεδο h) μέχρι τη ρίζα.
- Έπειτα στη γραμμή 4, για κάθε επίπεδο που επιλέγουμε από τη γραμμή 2, πέρνουμε έναν-έναν τους κόμβους του επιπέδου που είναι αριστερά παιδία, ξεκινώντας από τον αριστερότερο κόμβο του επιπέδου.
- Στις γραμμές 5-10 τοποθετούμε 1 ερευνητή στον κόμβο και βάσει της παραπάνω ανάλυσης οι ακμές που συνδέουν τον κόμβο με το αριστερό και το δεξί του παιδί καθαρίζουν αυτόματα. Αν ο ερευνητής δεν έχει τοποθετηθεί στη ρίζα μετατίθεται στον πατέρα του κόμβου του.
- Στη γραμμή 14 επιλέγουμε εναλλάξ τα επίπεδα, από το επίπεδο 1-(hmod2) μέχρι 1 επίπεδο πριν τα φύλλα.
- Στις γραμμές 16-18 επιλέγουμε από αριστερά προς τα δεξιά όλους τους κόμβους του επιπέδου και μετακινούμε κάθε ερευνητή στο δεξί παιδί του κόμβου του καθαρίζοντας έτσι την αντίστοιχη ακμή. □

Λήμμα 3.3.2. *(Πολυπλοκότητα αλγορίθμου) Ο αλγόριθμός μας (***Αλγόριθμος 1***) απαιτει Θ(n) βήματα για να ολοκληρώσει την εκτέλεσή του, όπου n το πλήθος των κόμβων του δέντρου.*

Απόδειξη. Ο αλγόριθμος επιλέγει εναλλάξ τα επίπεδα και ελέγχει τους μισούς κόμβους κάθε επιπέδου τοποθετώντας και μετακινώντας τους ερευνητές σε 1 βήμα. Στη συνέχεια επιλέγει τα υπόλοιπα επίπεδα και ελέγχει όλους τους κόμβους κάθε επιπέδου μετακινώντας κατά 1 βήμα τους ερευνητές. Άρα, σε κάθε έναν από τους κόμβους που ελέγχονται γίνεται ένας σταθερός αριθμός από πράξεις. Επομένως, η πολυπλοκότητα του αλγορίθμου εξαρτάται από τον αριθμό των κόμβων του δέντρου και ισούται με $\Theta(n)$. Ας δούμε όμως μια τυπικότερη απόδειξη, μετρώντας το πλήθος εκτελέσεων κάθε γραμμής του αλγορίθμου.

- Η γραμμή 1 εκτελείται 1 φορά με κόστος $\Theta(log_2 n)$ που προκύπτει από τη συνάρτηση υπολογισμού του ύψους του δέντρου.
- Οι γραμμές 2 και 3 εκτελούνται $\lfloor \frac{h}{2} \rfloor + 1 = \lfloor \frac{log_2(n+1)-1}{2} \rfloor + 1$ φορές, όμως η γραμμή 3 έχει κόστος το οποίο δεν ξεπερνά την τιμή του ύψους του δέντρου.
- Το πλήθος εκτελέσεων των γραμμών 4-10 προκύπτει από τον τύπο,

$$\sum \{\tfrac{1}{2} \cdot (2^{i+1} - 1 - (2^i - 1))\} = \sum \{\tfrac{1}{2} \cdot (2^{i+1} - 2^i)\} = \sum 2^{i-1}$$

για κάθε τιμή που δίνεται στο i από τη γραμμή 2. Επομένως, οι συνολικές εκτελέσεις των γραμμών 4-10 υπολογίζονται από τον τύπο:

$$2^{h-1} + 2^{h-3} + 2^{h-5} + ... = \tfrac{2^h}{2} + \tfrac{2^h}{2^3} + \tfrac{2^h}{2^5} + ... = 2^h \cdot (\tfrac{1}{2} + \tfrac{1}{2^3} + \tfrac{1}{2^5} + ...) =$$

$$2^h \cdot \textstyle\sum_{i=0}^{\lceil \frac{h}{2} \rceil} \frac{1}{2^{2i+1}} = \frac{n+1}{2} \cdot \sum_{i=0}^{\lceil \frac{log_2(n+1)-1}{2} \rceil} \frac{1}{2^{2i+1}}$$

Οι γραμμές 9 και 10 μπορεί να εκτελεστούν 1 φορά λιγότερο από ότι οι γραμμές 4-8 και αυτό συμβαίνει αν στη ρίζα έχει τοποθετηθεί ερευνητής.

- Στη συνέχεια, εκτελούνται $\lceil \frac{h}{2} \rceil = \lceil \frac{log_2(n+1)-1}{2} \rceil$ φορές οι γραμμές 14 και 15, όμως η γραμμή 15 έχει κόστος το οποίο και πάλι δεν ξεπερνά την τιμή του ύψους του δέντρου.
- Τέλος, ο αριθμός εκτελέσεων των γραμμών 16-18 προκύπτει από τον τύπο,

$$\sum \{2 \cdot 2^i - 2^i\} = \sum 2^i$$

για κάθε τιμή που παίρνει το i από τη γραμμή 20. Δηλαδή, ο συνολικός αριθμός εκτελέσεων των γραμμών 16-18 υπολογίζεται από τον τύπο:

$$2^{h-(h-1+h\%2)} + 2^{h-(h-3+h\%2)} + ... = 2^h \cdot (\frac{1}{2^{h-1+h\%2}} + \frac{1}{2^{h-3+h\%2}} + ...) =$$

$$2^h \cdot \sum_{i=0}^{\lceil \frac{h}{2} \rceil - 1} \frac{1}{2^{2i+1}} = \frac{n+1}{2} \cdot \sum_{i=0}^{\lceil \frac{log_2(n+1)-1}{2} \rceil - 1} \frac{1}{2^{2i+1}}$$

- Στον επόμενο πίνακα συγκεντρώνουμε το κόστος και το πλήθος εκτελέσεων κάθε γραμμής του αλγορίθμου:

Γραμμή (i)	Κόστος (c_i)	Πλήθος εκτελέσεων
1	$\Theta(log_2 n)$	1
2	c_2	$\lfloor \frac{log_2(n+1)-1}{2} \rfloor + 1$
3	$\Theta(log_2 n)$	$\lfloor \frac{log_2(n+1)-1}{2} \rfloor + 1$
4-10	c_i	$\leq \frac{n+1}{2} \cdot \sum_{i=0}^{\lceil \frac{log_2(n+1)-1}{2} \rceil} \frac{1}{2^{2i+1}}$
14	c_{14}	$\lceil \frac{log_2(n+1)-1}{2} \rceil$
15	$\Theta(log_2 n)$	$\lceil \frac{log_2(n+1)-1}{2} \rceil$
16-18	c_i	$\frac{n+1}{2} \cdot \sum_{i=0}^{\lceil \frac{log_2(n+1)-1}{2} \rceil - 1} \frac{1}{2^{2i+1}}$

Όπως είναι φανερό όλα τα αθροίσματα του πίνακα συγκλίνουν. Επομένως, όπως είδαμε και παραπάνω, η πολυπλοκότητα του αλγορίθμου είναι $\Theta(n)$. □

3.4 Αριθμός Ερευνητών

Λήμμα 3.4.1. *(Άνω φράγμα) Ο αλγόριθμος μας (**Αλγόριθμος 1**) χρειάζεται $\lceil \frac{n}{3} \rceil$ ερευνητές για τον καθαρισμό ενός πλήρους δυαδικού δέντρου n κόμβων.*

Απόδειξη. Ας δούμε τι γίνεται στο δέντρο προσθέτοντας σιγά-σιγά επίπεδα. Θα διακρίνουμε 2 περιπτώσεις:

1. Αν το ύψος του δέντρου είναι άρτιος αριθμός

- Ας υποθέσουμε ότι το αρχικό μας δέντρο είναι ύψους 0. Τότε θα υπάρχει μόνο 1 κόμβος, οπότε χρειαζόμαστε 1 ερευνητή ο οποίος θα προστατεύει τον κόμβο από τη μόλυνση.
- Κάθε αύξηση του ύψους έτσι ώστε το δέντρο να παραμένει άρτιου ύψους θα έχει σαν αποτέλεσμα την εισαγωγή 2 ακόμα επιπέδων ($e_{height-1}$ και e_{height}) τα οποία θα αποτελούνται από n_{height} κόμβους. Έτσι:

$$n_{height} = n_{e_{height-1}} + n_{e_{height}} \Leftrightarrow n_{height} = n_{e_{height-1}} + 2n_{e_{height-1}} \Leftrightarrow$$

$$n_{e_{height-1}} = \frac{1}{3} n_{height}$$

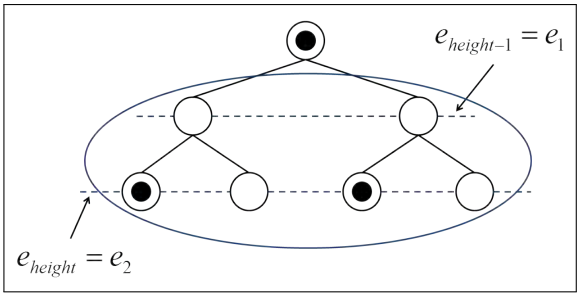
Οπότε, ο αριθμός τον κόμβων στο επίπεδο e_{height} είναι:

$$n_{height} = n_{e_{height-1}} + n_{e_{height}} \Leftrightarrow n_{height} = \frac{n_{height}}{3} + n_{e_{height}} \Leftrightarrow$$

$$n_{e_{height}} = \frac{2}{3} n_{height}$$

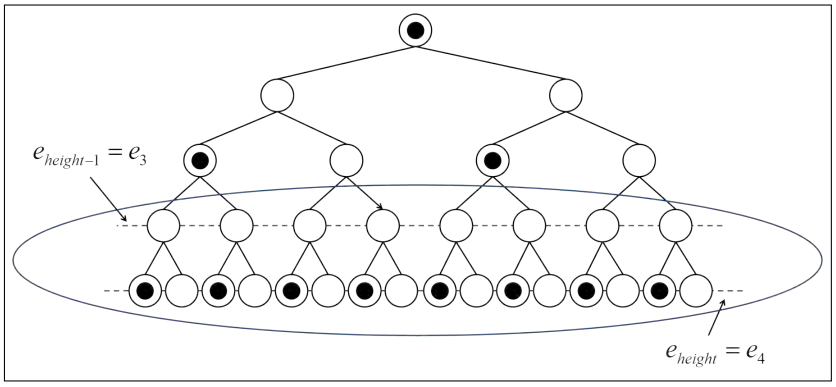
Δηλαδή το επίπεδο $e_{height-1}$ αποτελείται από $\frac{1}{3}n_{height}$ κόμβους και το επίπεδο e_{height} από $\frac{2}{3}n_{height}$ κόμβους. Στο επίπεδο e_{height} τοποθετούνται ερευνητές σε ένα μόνο από τα παιδιά, ενώ στο επίπεδο $e_{height-1}$ δεν τοποθετούνται ερευνητές. Αυτό σημαίνει ότι ο επιπλέον αριθμός ερευνητών που τοποθετούνται στο δέντρο σε κάθε αύξηση του ύψους ισούται με τον αριθμό των κόμβων που αντιστοιχούν στο επίπεδο $e_{height-1}$, δηλαδή $\frac{1}{3}n_{height}$.

- Έτσι, σύμφωνα με την προηγούμενη διαπίστωση, αν αυξήσουμε κατά 2 το ύψος (Σχήμα 3.1), θα προστεθούν n_2 νέοι κόμβοι εκ των οποίων $\frac{1}{3}n_2$ κόμβοι θα βρίσκονται στο επίπεδο e_1 και $\frac{2}{3}n_2$ στο επίπεδο e_2, ενώ ο αριθμός των επιπλέον ερευνητών που απαιτούνται είναι $\frac{1}{3}n_2$.



Σχήμα 3.1

- Όμοια, αν αυξήσουμε ξανά κατά 2 το ύψος (Σχήμα 3.2), δηλαδή το ύψος του δέντρου θα είναι πλέον 4, θα προστεθούν n_4 καινούριοι κόμβοι από τους οποίους $\frac{1}{3}n_4$ κόμβοι θα βρίσκονται στο επίπεδο e_3 και $\frac{2}{3}n_4$ στο επίπεδο e_4, ενώ θα χρειαστούν ακόμα $\frac{1}{3}n_4$ νέοι ερευνητές.



Σχήμα 3.2

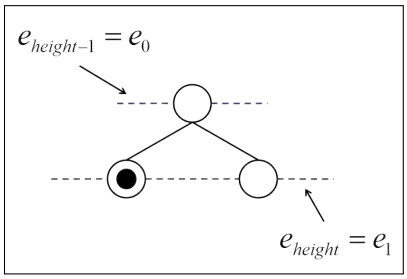
- Συνεχίζουμε την ίδια διαδικασία, αυξάνοντας συνεχώς το ύψος κατά 2. Έτσι, όταν το ύψος γίνει ίσο με k θα προστεθούν ακόμα n_k καινούριοι κόμβοι από τους οποίους $\frac{1}{3}n_k$ κόμβοι θα βρίσκονται στο επίπεδο e_{k-1} και $\frac{2}{3}n_k$ κόμβοι στο επίπεδο e_k, ενώ θα χρειαστούν ακόμα $\frac{1}{3}n_k$ νέοι ερευνητές.

Επομένως, ο αριθμός των ερευνητών που χρειαζόμαστε συνολικά όταν το ύψος του δέντρου είναι άρτιος αριθμός είναι:

$$1 + \tfrac{1}{3}n_2 + \tfrac{1}{3}n_4 + ... + \tfrac{1}{3}n_k = 1 + \tfrac{n_2+n_4+...+n_k}{3} = 1 + \tfrac{n-1}{3} = \tfrac{n+2}{3} = \lceil \tfrac{n}{3} \rceil$$

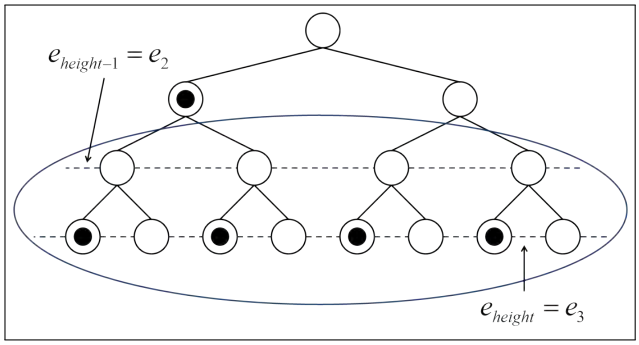
2. Αν το ύψος του δέντρου είναι περιττός αριθμός

   - Ας συνεχίσουμε την ίδια συλλογιστική πορεία, στην περίπτωση που το ύψος του δέντρου είναι περιττός αριθμός. Έτσι, αν ξεκινήσουμε με ένα δέντρο ύψους 1 (Σχήμα 3.3), θα έχουμε n_1 κόμβους, $\frac{1}{3}n_1$ από τους οποίους θα βρίσκονται στο επίπεδο e_0 ενώ $\frac{2}{3}n_1$ κόμβοι θα βρίσκονται στο επίπεδο e_1. Ο αριθμός των επιπλέον ερευνητών που θα χρειαστούν είναι $\frac{1}{3}n_2$.



Σχήμα 3.3

- Συνεχίζουμε αυξάνοντας και πάλι το ύψος του δέντρου κατά 2 (Σχήμα 3.4). Έτσι, το ύψος τώρα θα είναι 3. Άρα, θα έχουμε ακόμα n_3 καινούριους κόμβους από



Σχήμα 3.4

τους οποίους οι $\frac{1}{3}n_3$ θα βρίσκονται στο επίπεδο e_2 και οι $\frac{2}{3}n_3$ στο επίπεδο e_3, ενώ θα χρειαστούν ακόμα $\frac{1}{3}n_4$ νέοι ερευνητές.

- Τέλος, αν το ύψος γίνει k, θα προστεθούν ακόμα n_k νέοι κόμβοι εκ των οποίων $\frac{1}{3}n_k$ κόμβοι θα βρίσκονται στο επίπεδο e_{k-1} και $\frac{2}{3}n_k$ κόμβοι θα βρίσκονται στο επίπεδο e_k, ενώ τώρα θα χρειαστούν ακόμα $\frac{1}{3}n_k$ νέοι ερευνητές.

Άρα, ο αριθμός των ερευνητών που θα χρειαστούν συνολικά όταν το ύψος του δέντρου είναι περιττός αριθμός είναι:

$$\frac{1}{3}n_1 + \frac{1}{3}n_3 + ... + \frac{1}{3}n_k = \frac{n_1+n_3+...+n_k}{3} = \frac{n}{3}$$

Επομένως, συνδυάσουμε τις περιπτώσεις 1 και 2 καταλήγουμε στο συμπέρασμα ότι ένα πλήρες δυαδικό δέντρο μπορεί να καθαριστεί με $\lceil \frac{n}{3} \rceil$ ερευνητές, με τη χρήση μιας exclusive μονότονης στρατηγικής. □

Λήμμα 3.4.2. *(Κάτω φράγμα) Δεν υπάρχει αλγόριθμος ο οποίος να καθαρίζει ένα πλήρες δυαδικό δέντρο n κόμβων με λιγότερους από $\lceil \frac{n}{3} \rceil$, με τη χρήση μιας exclusive μονότονης στρατηγικής.*

Απόδειξη. Ξεκινώντας από τα φύλλα, χωρίζουμε το δέντρο (A) σε πλήρη δυαδικά υποδέντρα $A_1, A_2, ..., A_k$ ύψους 1 έτσι ώστε,

$$A_1 \cap A_2 \cap ... \cap A_k = 0 \text{ και}$$

$$A_1 \cup A_2 \cup ... \cup A_k = A \;||\; A_1 \cup A_2 \cup ... \cup A_k = A - \{root\}$$

Έστω k ο αριθμός των υποδέντρων που δημιουργούνται. Σύμφωνα με την περιγραφή που δώσαμε στην παράγραφο 3.1.1 παρατηρούμε ότι ο αλγόριθμός μας χρησιμοποιεί $k+1$ ερευνητές για δέντρο άρτιου ύψους και k ερευνητές για δέντρο περιττού ύψους.

Ας ξεκινήσουμε τον καθαρισμό του δέντρο χρησιμοποιώντας έναν ερευνητή, έστω s_1. Έστω ότι αυτός φτάνει για πρώτη φορά στη ρίζα ενός υποδέντρου μέσω της ακμής $\{u_1, u\}$. Τότε, η ακμή $\{u_1, u\}$ λόγω της ολίσθησης καθαρίζει, ενώ ο s_1 βρίσκεται πλέον στη ρίζα u αυτού του υποδέντρου. Όμως στον u προσπίπτουν τώρα 2 μολυσμένες ακμές, έστω $\{u, u_2\}$,

$\{u, u_3\}$ με αποτέλεσμα ο s_1 να μη μπορεί να απομακρυνθεί από τον κόμβο u, αφού αν το κάνει θα υπάρξει επαναμόλυνση. Επομένως χρειάζεται ακόμα 1 ερευνητής. Έστω s_2 ο ερευνητής αυτός ο οποίος στέκεται σε κάποιον από τους κόμβους u_2 ή u_3 (έστω στον u_2). Τότε, εκτός της $\{u_1, u\}$ που είναι ήδη καθαρή, καθαρή θα είναι και η $\{u, u_2\}$ λόγω της ταυτόχρονης ύπαρξης ενός ερευνητή και στα δυο της άκρα. Πλέον, στον u προσπίπτει μία μολυσμένη ακμή, η $\{u, u_3\}$, μέσω της οποίας ο s_1 μπορεί να απομακρυνθεί από τον u χωρίς να υπάρξει κίνδυνος επαναμόλυνσης.Όπως βλέπουμε οι ερευνητές s_1 και s_2 βρίσκονται σε κόμβο γειτονικό του{u}, ενώ δύο ερευνητές δεν επιτρέπεται να βρεθούν στον ίδιο κόμβο ταυτόχρονα. Επομένως, όλοι αυτοί οι ερευνητές θα εισέρχονται για πρώτη φορά σε έναν νέο κόμβο από την ίδια ακμή.

Ας θεωρήσουμε τώρα, $\{v_1, v\}$ την ακμή μέσω της οποίας ένας από τους παραπάνω ερευνητές εισέρχεται για πρώτη φορά στη ρίζα v ενός άλλου υποδέντρου. Έστω λοιπόν ότι ο ερευνητής s_1 κινήθηκε κατά μήκος της $\{v_1, v\}$ καθρίζοντάς την. Πλέον, στον κόμβο v προσπίπτουν 2 μολυσμένες ακμές, έστω $\{v, v_2\}$, $\{v, v_3\}$. Όπως αναφέραμε και πιο πάνω, σε κάποιο γειτονικό κόμβο του v, εκτός του v_1, θα πρέπει να βρίσκεται 1 ερευνητής. Όμως αυτός ο ερευνητής δεν μπορεί να είναι ο s_2, καθώς εισέρχεται για πρώτη φορά στον κόμβο v μέσω της $\{v_1, v\}$. Έστω s_3 ο ερευνητής που στέκεται στον κόμβο v_2. Έτσι η ακμή $\{v, v_2\}$ είναι καθαρή και ο s_1 μπορεί να απομακρυνθεί από τον v χωρίς να υπάρξει επαναμόλυνση. Όμοια με πριν, οι s_1 και s_3 εισέρχονται για πρώτη φορά σε έναν νέο κόμβο από την ίδια ακμή. Όμως και οι ερευνητές s_1 και s_2, εισέρχονται για πρώτη φορά σε έναν κόμβο από την ίδια ακμή που εισέρχεται στον κόμβο αυτό και ο s_1. Δηλαδή όλοι οι ερευνητές που υπάρχουν στο δέντρο εισέρχονται για πρώτη φορά σε έναν νέο κόμβο από την ίδια ακμή.

Όπως βλέπουμε, εκτός από τον αρχικό ερευνητή με τον οποίο ξεκινήσαμε να καθαρίζουμε το δέντρο, για κάθε υποδέντρο (εκτός από το υποδέντρο ρίζα του οποίου είναι η ρίζα του δέντρου) εισάγεται 1 ακόμη ερευνητής στο δέντρο. Για το υποδέντρο ρίζα του οποίου είναι η ρίζα του δέντρου δεν χρειάζεται να εισαχθεί κάποιος ερευνητής στο δέντρο διότι 1 από τους ήδη υπάρχοντες ερευνητές θα μπορούσε να εισέλθει μέσω μιας ακμής $\{w_1, w\}$ καθαρίζοντάς την. Τότε στον κόμβο w θα πρσπίπτει 1 μόνο μολυσμένη ακμή, οπότε ο ερευνητής του κόμβου w μπορεί να φύγει από το συγκεκριμένο κόμβο μέσω αυτής της ακμής. Επομένως χρειαζόμαστε τουλάχιστον $k+1$ ερευνητές για δέντρο άρτιου ύψους και k ερευνητές για δέντρο περιττού ύψους. □

Συμπέρασμα των λημμάτων 3.4.1 και 3.4.2, αφού άνω και κάτω φράγμα συμπίπτουν, είναι το ακόλουθο λήμμα:

Λήμμα 3.4.3. *Ο αλγόριθμος μας (***Αλγόριθμος 1***) χρησιμοποιεί των ελάχιστο αριθμό ερευνητών και επομένως είναι βέλτιστος.*

Χρησιμοποιώντας το λήμμα 3.4.1 προκύπτει ο παρακάτω αλγόριθμος, ο οποίος υπολογίζει το search number που απαιτείται για τον καθαρισμό ενός πλήρους δυαδικού δέντρου n κόμβων με τη χρήση μιας exclusive μονότονης στρατηγικής:

Αλγόριθμος 2 *Υπολογισμός xms(T) για πλήρη δυαδικά δέντρα*

Είσοδος: *πλήθος κόμβων n*
Έξοδος: *xms(T)*

1: *xmsn* ← *(n+2)***div***3*
2: ***return*** *xmsn*

Λήμμα 3.4.4. *Ο αλγόριθμος υπολογισμού του xms(T) (***Αλγόριθμος 2***) απαιτεί σταθερό αριθμό βημάτων.*

3.5 Βασικοί Υπολογισμοί

3.5.1 Αριθμός Συνολικών Κινήσεων

Με τον όρο συνολικός αριθμός κινήσεων, αναφερόμαστε στο συνολικό αριθμό μεταθέσων που γίνονται από τους ερευνητές κατά τη διάρκεια του καθαρισμού. Αρκεί να γνωρίζουμε ότι κάθε μετάθεση ενός ερευνητή από έναν κόμβο σε έναν άλλο λογίζεται ως 1 κίνηση.

Λήμμα 3.5.1. *Σύμφωνα με τον αλγόριθμό μας (***Αλγόριθμος 1***), ένα πλήρες δυαδικό δέντρο n κόμβων καθαρίζεται με* $2 \cdot \lfloor \frac{n}{3} \rfloor$ *κινήσεις.*

Απόδειξη. Σύμφωνα με τον αλγόριθμο μας, απαιτούνται 2 κινήσεις από κάθε ερευνητή. Ο μόνος ερευνητής που παραμένει ακίνητος κατά τη διάρκεια του καθαρισμού είναι αυτός που έχει τοποθετηθεί στη ρίζα, αλλά για να συμβεί αυτό θα πρέπει το δέντρου να είναι άρτιου ύψους. Οπότε:

- Αν το ύψος του δέντρου είναι άρτιος αριθμός, ο συνολικός αριθμός κινήσεων ισούται με το διπλάσιο του αριθμού των ερευνητών αν από το άθροισμα αυτό αφαιρέσουμε τις 2 κινήσεις του ακίνητου ερευνητή. Άρα, σύμφωνα και με το λήμμα 3.4.1, αφού ο αριθμός των ερευνητών είναι $\lceil \frac{n}{3} \rceil$, τότε ο συνολικός αριθμός κινήσεων των ερευνητών είναι ίσος με $2 \cdot \lceil \frac{n}{3} \rceil - 2 = 2 \cdot (\lceil \frac{n}{3} \rceil - 1) = 2 \cdot \lfloor \frac{n}{3} \rfloor$
- Αν το ύψος του δέντρου είναι περιττός αριθμός, τότε ο αριθμός κινήσεων ισούται με το διπλάσιο του αριθμού των ερευνητών. Επομένως, λαμβάνοντας υπόψη το λήμμα 3.4.1, οι συνολικές μεταθέσεις που απαιτούνται από τους ερευνητές είναι $2 \cdot \lceil \frac{n}{3} \rceil = 2 \cdot \frac{n}{3} = 2 \cdot \lfloor \frac{n}{3} \rfloor$

Άρα, ο συνολικός αριθμός κινήσεων των ερευνητών ισούται με $2 \cdot \lfloor \frac{n}{3} \rfloor$. □

3.5.2 Αριθμός Βημάτων

Κάποιες από τις κινήσεις του λήμματος 3.5.1 μπορούν να γίνουν ταυτόχρονα και συνεπώς να μετρηθούν σε 1 βήμα.

Λήμμα 3.5.2. *Σύμφωνα με τον αλγόριθμό μας (***Αλγόριθμος 1***), για τον καθαρισμό ενός πλήρους δυαδικού δέντρου n κόμβων, αρκούν:*

- *log(n+1)-1 βήματα, αν το ύψος του δέντρου είναι άρτιος αριθμός και*
- *log(n+1) βήματα, αν το ύψος του δέντρου είναι περιττός αριθμός.*

Απόδειξη. Όπως είδαμε κατά την απόδειξη ορθότητας του αλγόριθμου η κίνηση ενός ερευνητή από ένα επίπεδο δεν εξαρτάται από την κίνηση άλλου ερευνητή του επιπέδου. Οπότε, θεωρώντας ότι όλοι οι ερευνητές ενός επιπέδου κινούνται ταυτόχρονα θα έχουμε:

1. Αν το ύψος του δέντρου είναι άρτιος αριθμός

   Για την πρώτη κίνηση των ερευνητών θα χρειαστεί 1 βήμα για την κίνηση των ερευνητών από το επίπεδο των φύλλων (επίπεδοh) στο επίπεδο $h-1$, 1 βήμα για την κίνηση των ερευνητών από $h-2$ στο επίπεδο $h-3$ κ.ο.κ.. Οπότε, για την πρώτη κίνηση των ερευνητών, δεδομένου ότι ο ερευνητής στη ρίζα παραμένει ακίνητος, θα χρειαστούν $\frac{h}{2}$ βήματα. Για τη δεύτερη κίνηση των ερευνητών θα χρειαστεί 1 βήμα για την κίνηση των ερευνητών από το επίπεδο 1 στο επίπεδο 2, 1 βήμα για την κίνηση των ερευνητών από 3 στο επίπεδο 4 κ.ο.κ.. Επομένως, για τη δεύτερη κίνηση των ερευνητών απαιτούνται επίσης $\frac{h}{2}$ βήματα. Άρα, συνολικά για την πρώτη και τη δεύτερη κίνηση των ερευνητών χρειάζονται,

   $$\tfrac{h}{2} + \tfrac{h}{2} = h = log_2(n+1) - 1 \quad \text{βήματα}$$

2. Αν το ύψος του δέντρου είναι περιττός αριθμός

   Για την πρώτη κίνηση των ερευνητών θα χρειαστεί 1 βήμα για την κίνηση των ερευνητών από το επίπεδο h στο επίπεδο $h-1$, 1 βήμα για την κίνηση των ερευνητών από $h-2$ στο επίπεδο $h-3$ κ.ο.κ.. Οπότε, για την πρώτη κίνηση των ερευνητών απαιτούνται $\frac{h+1}{2}$ βήματα. Για τη δεύτερη κίνηση των ερευνητών θα χρειαστεί 1 βήμα για την κίνηση των ερευνητών από το επίπεδο 0 στο επίπεδο 1, 1 βήμα για την κίνηση των ερευνητών από 2 στο επίπεδο 3 κ.ο.κ.. Επομένως, για τη δεύτερη κίνηση των ερευνητών απαιτούνται $\frac{h+1}{2}$ βήματα. Άρα, συνολικά για την πρώτη και τη δεύτερη κίνηση των ερευνητών χρειάζονται,

   $$\tfrac{h+1}{2} + \tfrac{+1}{2} = h + 1 = log_2(n+1) \quad \text{βήματα}$$

□

Κεφάλαιο 4

Καθαρισμός Πλήρους m-αδικού Δέντρου

Στο κεφάλαιο ασχολούμαστε με τον καθαρισμό ενός πλήρους m-αδικού δέντρου. Θα αναλύσουμε τη διαδικασία καθαρισμού ενός πλήρους m-αδικού δέντρου χρησιμοποιώντας μια exclusive μονότονη στρατηγική, δίνοντας ταυτόχρονα και τον αντίστοιχο αλγόριθμο. Στη συνέχεια, θα υπολογίσουμε τον αριθμό ερευνητών που χρησιμοποιείται από τον αλγόριθμο έτσι ώστε να καθαριστεί το δέντρο, το συνολικό αριθμό μεταθέσεων των ερευνητών κατά τη διάρκεια του καθαρισμού, καθώς επίσης και τα βήματα που απαιτούνται για τον καθαρισμό του δέντρου σύμφωνα με τον αλγόριθμο.

4.1 Ιδιότητες Κόμβων

Ας ξεκινήσουμε αποδεικνύοντας διάφορες ιδιότητες που ισχύουν για τους κόμβους ενός πλήρους m-αδικού δέντρου, τις οποίες θα χρησιμοποιήσουμε στον αλγόριθμο που θα αναπτύξουμε παρακάτω.

Λήμμα 4.1.1. *(Λήμμα κόμβων) Έστω ενα πλήρες m-αδικό δέντρο όπου node είναι η θέση ενός κόμβου του εκτός της ρίζας.*

*i) Αν (node-1)**mod**m = 0 τότε ο κόμβος node είναι τελευταίο παιδί.*

*ii) Αν (node-1)**mod**m = n, όπου n ένας φυσικός αριθμός, τότε ο κόμβος node είναι n-οστό παιδί.*

Απόδειξη. i) Ας παρατηρήσουμε το σχήμα 4.1. Ο πρώτος κόμβος που είναι τελευταίο παιδί και για τον οποίο ισχύει η συνθήκη, είναι ο κόμβος m+1 αφού:

$$(\text{node-1})\textbf{mod}\text{m} = [(\text{m+1})\text{-1}]\textbf{mod}\text{m} = 0$$

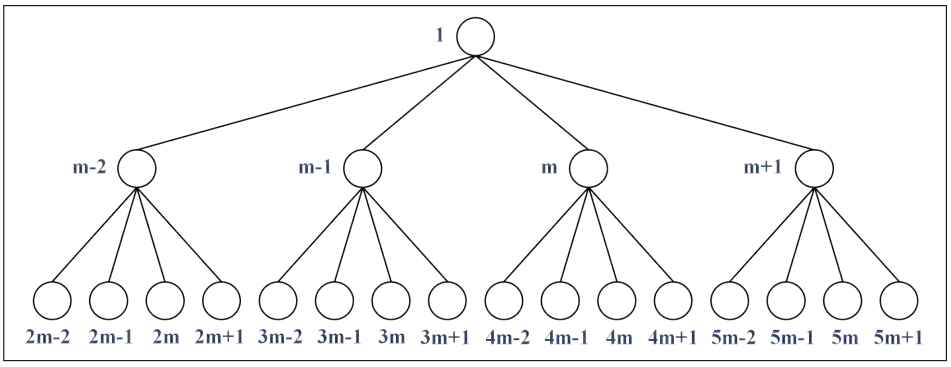
Κάθε άλλος κόμβος τελευταίο παιδί θα βρίσκεται k·m θέσεις μετά τον κόμβο αυτό, όπου k ένας φυσικός αριθμός. Δηλαδή θα βρίκεται στη θέση:

$$\text{m+1+k}\cdot\text{m} = (\text{k+1})\cdot\text{m+1}$$

Τότε η συνθήκη (node-1)**mod**m = 0 θα είναι αληθής αφού:

$$(\text{node-1})\ \textbf{\textit{mod}}\ \text{m} = [(\text{k+1})\cdot\text{m+1-1}]\ \textbf{\textit{mod}}\ \text{m} = [(\text{k+1})\cdot\text{m}]\ \textbf{\textit{mod}}\ \text{m} = 0$$

Επομένως, η συνθήκη ισχύει για κάθε κόμβο που είναι τελευταίο παιδί.



Σχήμα 4.1

Ας δούμε όμως τι γίνεται με τους υπόλοιπους κόμβους. Προφανώς, επειδή οι θέσεις m, m-1, m-2,...,m-(m-2) έχουν τιμή μικρότερη ή ίση του m θα κάνουν ψευδή τη συνθήκη (node-1)***mod***m = 0. Και επειδή όλοι οι υπόλοιποι αντίστοιχοι κόμβοι βρίσκονται k·m θέσεις μέτα θα αφήνουν και αυτοί υπόλοιπο στη συνθήκη. Επομένως, η συνθήκη (node-1)***mod***m = 0 δεν ισχύει για κόμβους που δεν είναι τελευταία παιδιά.

ii) Ας παρατηρήσουμε και πάλι το σχήμα 4.1. Βλέπουμε ότι η θέση m-(m-2) αντιστοιχεί στον κόμβο 2, η θέση m-(m-3) αντιστοιχεί στον κόμβο 3 κ.ο.κ.. Επομένως:

- m-(m-2) = 2 → θέση πρώτου παιδιού
- m-(m-3) = 3 → θέση δεύτερου παιδιού

  .
  .
  .

- m → θέση προτελευταίου παιδιού

Αν μειώσουμε κατά 1 την τιμή των παραπάνω θέσεων τότε οι νέες τιμές των θέσεων είναι:

- m-(m-2)-1 = 1
- m-(m-3)-1 = 2

  .
  .
  .

- m-1

Παρατηρούμε ότι οι τιμές αυτές συμβαδίζουν με τον αριθμό του παιδιού στην αντίστοιχη θέση. Έστω τώρα n μία από τις προηγούμενες τιμές. Επειδή η τιμή του n θα είναι πάντα μικρότερη του m η συνθήκη θα γίνεται:

$$(\text{node}-1)\ \textbf{\textit{mod}}\ \text{m} = \text{n}\ \textbf{\textit{mod}}\ \text{m} = \text{n}$$

Το αντίστοιχο παιδί ενός άλλου κόμβου θα βρίσκεται στη θέση $n+1+k\cdot m$, οπότε τώρα το αποτέλεσμα της συνθήκης θα είναι το ακόλουθο:

$$(\text{node}-1)\ \textbf{\textit{mod}}\ \text{m} = [(\text{n}+1+\text{k}\cdot\text{m})-1]\ \textbf{\textit{mod}}\ \text{m} = \text{n}\ \textbf{\textit{mod}}\ \text{m} = \text{n}$$

Βλέπουμε και πάλι ότι το αποτέλεσμα της συνθήκης μας δίνει τον αριθμό του παιδιού για τη συγκεκριμένη θέση. Επομένως, η συνθήκη ισχύει. □

Λήμμα 4.1.2. *(Λήμμα προτελευταίου παιδιού) Έστω ενα πλήρες m-αδικό δέντρο όπου node είναι η θέση ενός κόμβου του. Αν ισχύει η συνθήκη node* ***mod*** *m = 0 τότε ο κόμβος node είναι προτελευταίο παιδί.*

Απόδειξη. Ας δούμε και πάλι το σχήμα 4.1. Ο πρώτος κόμβος που είναι προτελευταίο παιδί και για τον οποίο ισχύει η συνθήκη, είναι ο κόμβος m αφού:

$$\text{node}\ \textbf{\textit{mod}}\ \text{m} = \text{m}\ \textbf{\textit{mod}}\ \text{m} = 0$$

Κάθε άλλος κόμβος που είναι προτελευταίο παιδί θα βρίσκεται $k\cdot m$ θέσεις μετά τον κόμβο αυτό. Δηλαδή θα βρίκεται στη θέση:

$$\text{m}+\text{k}\cdot\text{m} = (\text{k}+1)\cdot\text{m}.$$

Τότε θα ισχύει η συνθήκη node***mod***m = 0 αφού:

$$\text{node}\ \textbf{\textit{mod}}\ \text{m} = [(\text{k}+1)\cdot\text{m}]\ \textbf{\textit{mod}}\ \text{m} = 0$$

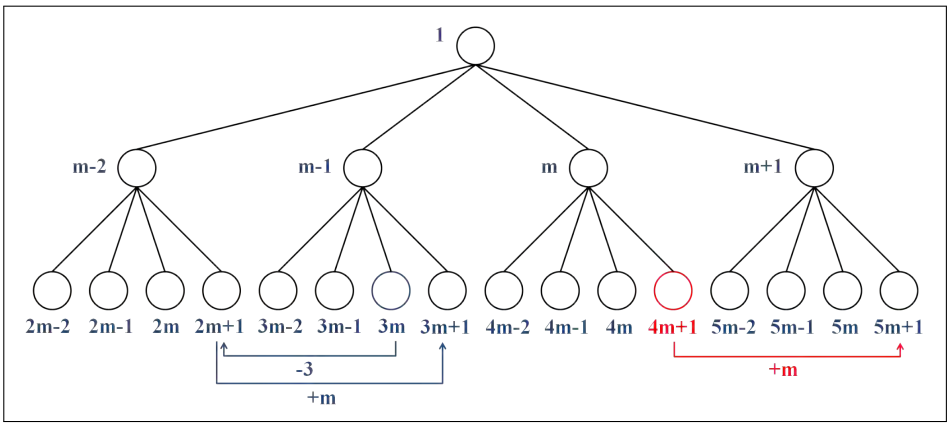
Επομένως, η συνθήκη ισχύει για κάθε κόμβο που είναι προτελευταίο παιδί.

Ας δούμε τώρα τι συμβαίνει με τους υπόλοιπους κόμβους του δέντρου. Όπως εύκολα γίνεται κατανοητό, οι θέσεις m-1, m-2,...,m-(m-2) έχουν τιμή μικρότερη του m, ενώ η θέση m+1 έχει τιμή μεγαλύτερη του m αλλά μικρότερη του $2\cdot m$, αφού $m>=2$. Επομένως, για τις θέσεις αυτές η συνθήκη node***mod***m = 0 θα είναι ψευδής, εφόσον θα μένει υπόλοιπο. Και επειδή όλοι οι υπόλοιποι αντίστοιχοι κόμβοι βρίσκονται $k\cdot m$ θέσεις μέτα θα αφήνουν και αυτοί υπόλοιπο στη συνθήκη. Επομένως, η συνθήκη node***mod***m = 0 δεν ισχύει για κόμβους που δεν είναι προτελευταία παιδιά. □

Λήμμα 4.1.3. *Έστω ενα πλήρες m-αδικό δέντρο όπου node είναι η θέση ενός κόμβου του. Η θέση του επόμενου κατά σειρά κόμβου που είναι τελευταίο παιδί, υπολογίζεται από τη συνάρτηση:*

$$f(node) = node+m-((node-1)\ \textbf{\textit{mod}}\ m)$$

Απόδειξη. Ας υποθέσουμε ότι η θέση που παίρνουμε αντιστοιχεί σε τελευταίο παιδί (π.χ. ο κόμβος 4m+1 του σχήματος 4.2). Τότε, το μόνο που έχουμε να κάνουμε είναι να προσθέσουμε m θέσεις στη θέση αυτή έτσι ώστε να μεταβούμε στο επόμενο τελευταίο παιδί. Ας δούμε τώρα τι κάνουμε αν η θέση που θα πάρουμε δεν είναι τελευταίο παιδί (π.χ. ο κόμβος 3m του σχήματος 4.2). Τότε θα πρέπει να πάμε στο προηγούμενο τελευταίο παιδί, άρα θα πρέπει να αφαιρέσουμε τον αριθμό του παιδιού της αντίστοιχης θέσης, και από εκεί να προσθέσουμε m θέσεις έτσι ώστε να πάμε στο επόμενο τελευταίο παιδί.



Σχήμα 4.2

Έτσι, σύμφωνα και με το λήμμα 4.1.1, η νέα θέση υπολογίζεται από τον τύπο:

$$f(node) = node+m-((node-1)\textbf{\textit{mod}}m)$$

□

4.2 Περιγραφή Αλγορίθμου

4.2.1 Ο Αλγόριθμος

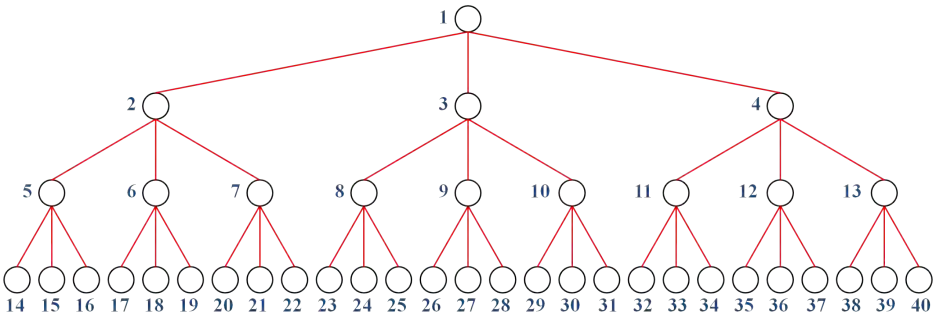
Ο αλγόριθμος που προτείνουμε στη συνέχεια βασίζεται στη λογική που ακολουθήσαμε για τα πλήρη δυαδικά δέντρα στην παράγραφο 3.1.1 και αφορά τα πλήρη m-αδικά δέντρα. Συγκεκριμένα, οι ερευνητές τοποθετούνται στο δέντρο ανά 2 επίπεδα ξεκινώντας από το επίπεδο των φύλλων, μόνο που τώρα η τοποθέτησή τους γίνεται σε κόμβους του αντίστοιχου επιπέδου που δεν είναι τελευταία παιδιά. Σε περίπτωση που το δέντρο είναι άρτιου ύψους, τοποθετούμε έναν ερευνητή και στη ρίζα του δέντρου. Επίπλέον, μόνο ένας ερευνητής από κάθε ομάδα κόμβων που ανήκουν στον ίδιο πατέρα αναλαμβάνει να κινηθεί. Στον αλγόριθμό μας το ρόλο αυτό αναλαμβάνει ο ερευνητής που βρίσκεται σε κόμβο ο οποίος είναι πρώτο παιδί. Οι υπόλοιποι ερευνητές, όπως επίσης και ο ερευνητής που τοποθετείται στη ρίζα, παραμένουν ακίνητοι κατά τη διάρκεια του καθαρισμού είναι όμως απαραίτητοι για τον καθαρισμό του δέντρου. Ο καθαρισμός και εδώ ολοκληρώνεται αφού κάθε ένας από

τους κινούμενους ερευνητές πραγματοποιήσει 2 κινήσεις, μία από το πρώτο παιδί στον πατέρα και μία από τον πατέρα στο τελευταίο παιδί, ενώ η δεύτερη κίνηση των ερευνητών προϋποθέτει την ολοκλήρωση της πρώτης από όλους του κινούμενους ερευνητές. Τέλος, η πρώτη κίνηση ενός ερευνητή που βρίσκεται σε ανώτερο επίπεδο απαιτεί την ολοκλήρωση της πρώτης κίνησης των κινούμενων ερευνητών παρακάτω επιπέδων, ενώ η δεύτερη κίνηση ενός ερευνητή που βρίσκεται σε κατώτερο επίπεδο προϋποθέτει την ολοκλήρωση της δεύτερης κίνησης των κινούμενων ερευνητών παραπάνω επιπέδων. Η προηγούμενη ανάλυση ισχύει μόνο για την περίπτωση που ο βαθμός του δέντρου είναι μεγαλύτερος του 1. Αν ο βαθμός του δέντρου είναι 1, τότε ουσιαστικά θα έχουμε να καθαρίσουμε μία τοπολογία γραμμής. Όπως είναι γνωστό, για τον καθαρισμό μιας γραμμικής τοπολογίας αρκεί ένας ερευνητής ο οποίος θα διασχίσει το γράφο από το ένα άκρο του στο άλλο. Ένας απλός αλγόριθμος που εκτελεί την παραπάνω διαδικασία είναι ο ακόλουθος:

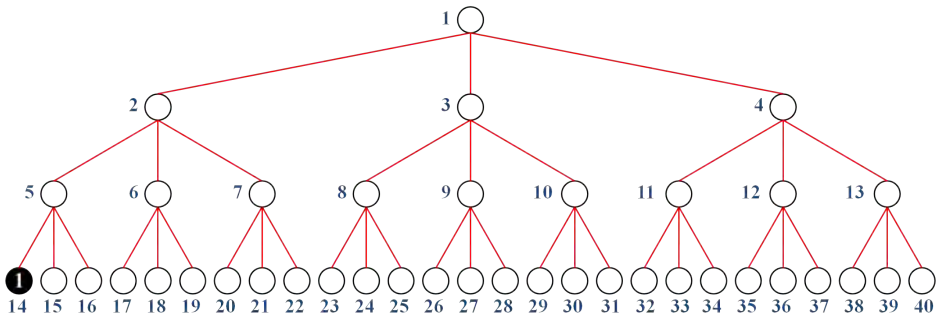
1. **Αν** ο κάτω βαθμός του δέντρου είναι 1 **τότε**
2. Καθαρισμός γραμμικής τοπολογίας
3. **Αλλιώς**
4. **Για κάθε** επίπεδο s **από** height **μέχρι** 0 **με βήμα** -2
5. **Για κάθε** κόμβο n του επιπέδου s
6. **Αν** ο κόμβος n δεν είναι τελευταίο παιδί **τότε**
7. Βάλε ερευνητή στον κόμβο n
8. **Αν** ο κόμβος n είναι πρώτο παιδί **τότε**
9. Κίνηση ερευνητή στον πατέρα του κόμβου n
10. **Για κάθε** επίπεδο s **από** 1-(height ***mod*** 2) **μέχρι** height-1 **με βήμα** 2
11. **Για κάθε** κόμβο n του επιπέδου s
12. Κίνηση ερευνητή στο τελευταίο παιδί του κόμβου n

4.2.2 Παράδειγμα Εκτέλεσης Αλγορίθμου

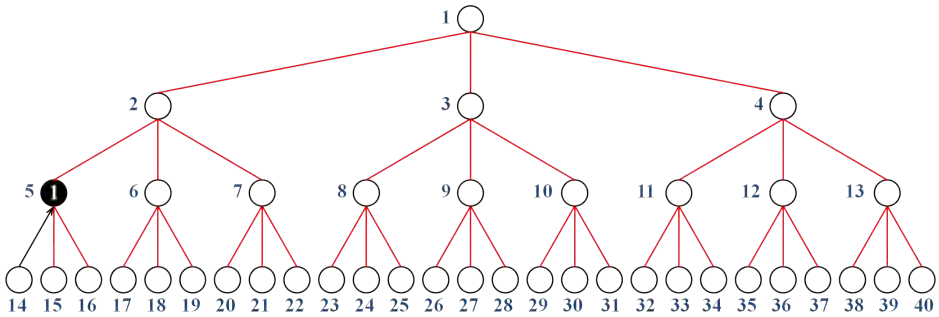
- Ας θεωρήσουμε ότι θέλουμε να καθαρίσουμε ένα δέντρο ύψους και βαθμού 3. Έτσι, το δέντρο θα αποτελείται από 40 κόμβους.



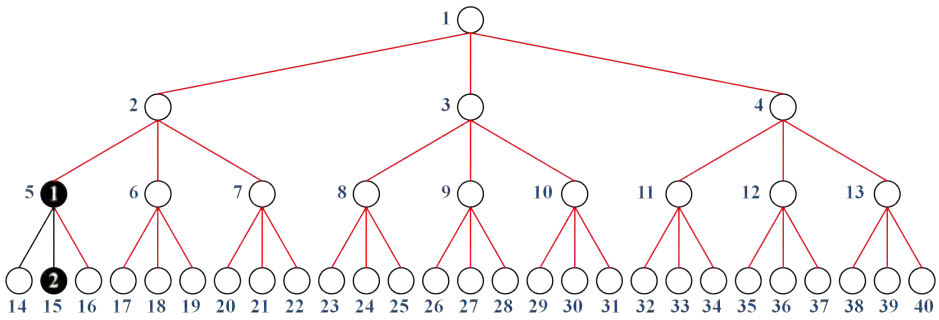
- Η διαδικασία του καθαρισμού ξεκινά. Πρώτο επίπεδο κίνησης είναι το επίπεδο 3. Ένας ερευνητής τοποθετείται στον κόμβο 14 του δέντρου.



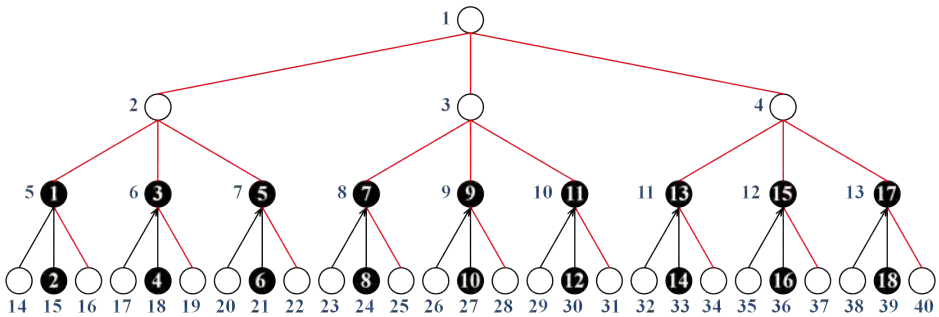
- Ο ερευνητής μετακινείται από τον κόμβο 14 στον κόμβο 5.



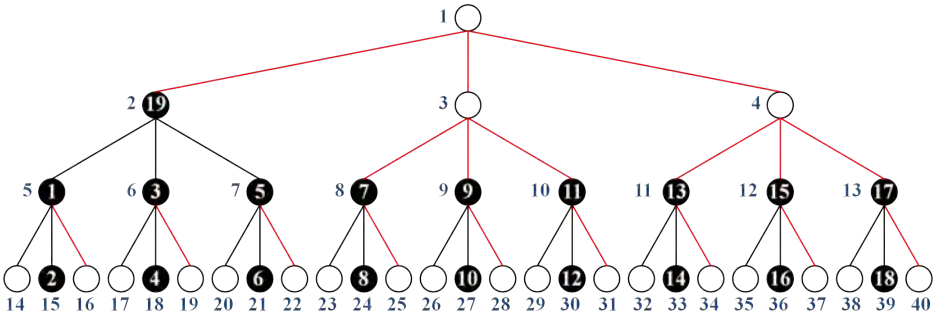
- Ένας ακόμα ερευνητής τοποθετείται στον κόμβο 15 του δέντρου αλλά παραμένει ακίνητος.



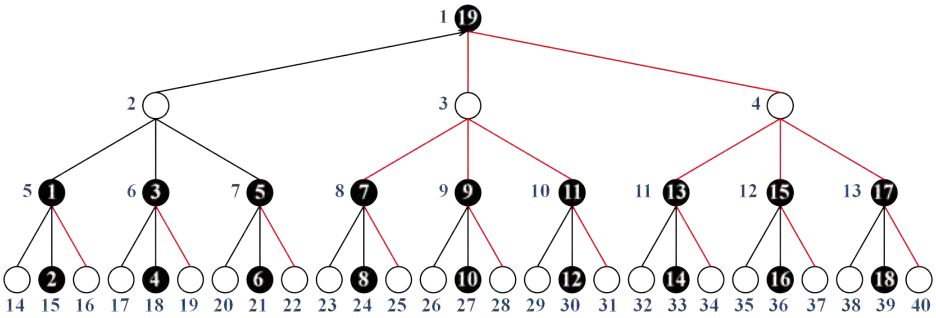
- Κάθε ερευνητής που τοποθετείται σε κόμβο του τρίτου επιπέδου που είναι πρώτο παιδί μετακινείται στον πατέρα αυτού του κόμβου, ενώ κάθε ερευνητής που τοποθετείται σε έναν από τους υπόλοιπους κόμβους του επιπέδου παραμένει ακίνητος.



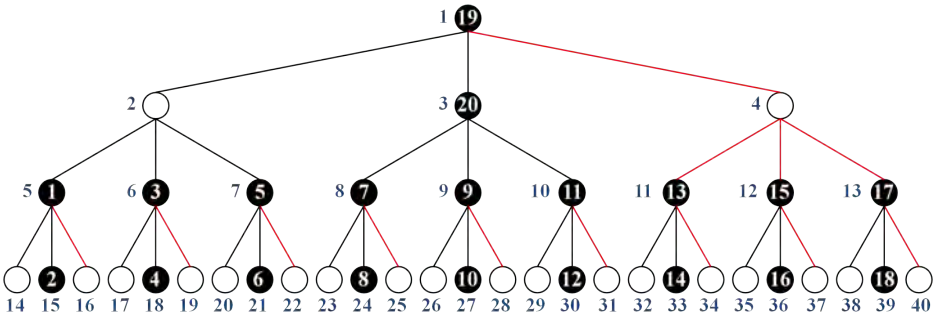
- Επόμενο επίπεδο κίνησης είναι το επίπεδο 1. Ένας ερευνητής τοποθετείται στον κόμβο 2.



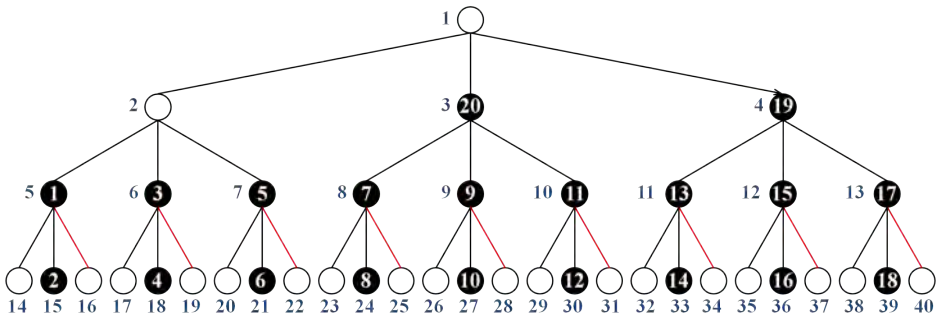
- Ο ερευνητής μετακινείται από τον κόμβο 2 στη ρίζα του δέντρου.



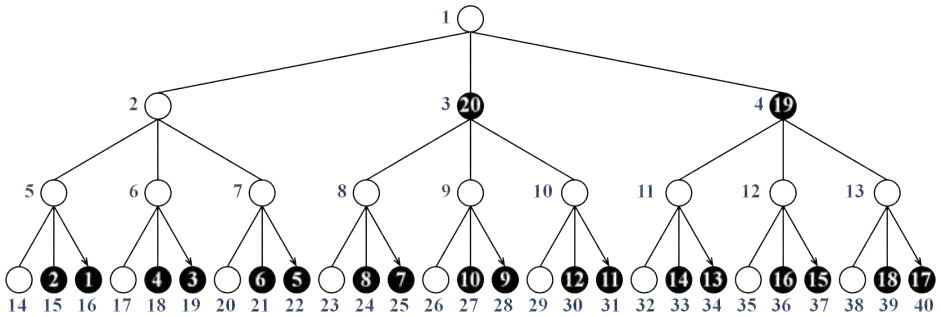
- Ένας επιπλέον ερευνητής τοποθετείται στον κόμβο 3 του δέντρου, παραμένει όμως ακίνητος.



- Σειρά έχει ο ερευνητής που βρίσκεται στη ρίζα. Ο ερευνητής διασχίζει την ακμή που ενώνει τη ρίζα με το τελευταίο της παιδί.



- Τελευταίο επίπεδο κίνησης είναι το επίπεδο 2. Κάθε ερευνητής αυτού του επιπέδου, διασχίζει την ακμή που ενώνει τον κόμβο του με το τελευταίο του παιδί.



4.3 Υλοποίηση Αλγορίθμου

Στην ενότητα αυτή υλοποιούμε τον αλγόριθμο της παραγράφου 4.2.1. Αν και σε κάποια βήματα θα μπορούσαν να κινηθούν ταυτόχρονα περισσότεροι από ένας ερευνητές, εμείς επιλέγουμε να αναπτύξουμε τον αλγόριθμο θεωρώντας ότι μόνο ένας ερευνητής μπορεί να κινηθεί σε κάθε βήμα.

Αλγόριθμος 3 *Καθαρισμός πλήρους m-αδικού δέντρου*

Είσοδος: *κάτω βαθμός m, πλήθος κόμβων n*
Έξοδος: *-*

```
1:  if (m=1) then
2:      καθαρισμός γραμμικής τοπολογίας
3:  else
4:      height ← log_m n
5:      for (i = h, i = i − 2, while i ≥ 0) do
6:          lmost ← (m^i − 1)/(m − 1) + 1
7:          rmost ← (m^{i+1} − 1)/(m − 1)
8:          v ← (m^{i−1} − 1)/(m − 1) + 1
9:          for (u = lmost, u + +, while u ≤ rmost) do
10:             if (ο u δεν είναι τελευταίο παιδί or ο u είναι ρίζα) then
11:                 ένας νέος ερευνητής s_u τοποθετείται στον κόμβο u
```

12: *{u, παιδιά(u)} ← καθαρές*
13: ***if** (ο κόμβος u είναι πρώτο παιδί) **then***
14: *ο ερευνητής s_u πηγαίνει από τον κόμβο u στον κόμβο v*
15: *{u, v} ← καθαρή*
16: *v ← v+1*
17: ***end if***
18: ***end if***
19: ***end for***
20: ***end for***
21: ***for** ($i = 1 - (h \bmod 2)$, $i = i + 2$, *while* $i < h$) **do***
22: *lmost* $\leftarrow (m^i - 1)/(m - 1) + 1$
23: *rmost* $\leftarrow (m^{i+1} - 1)/(m - 1)$
24: *v ← rmost*
25: ***for** ($u = lmost$, $u++$, *while* $u \leq rmost$) **do***
26: *v ← ο επόμενος κόμβος του u που είναι τελευταίο παιδί*
27: *ο ερευνητής s_u πηγαίνει από τον κόμβο u στον κόμβο v*
28: *{u, v} ← καθαρή*
29: ***end for***
30: ***end for***
31: ***end if***

Ανά 2 επίπεδα, από τα φύλλα προς τη ρίζα, επιλέγουμε έναν-έναν όλους τους κόμβους του αντίστοιχου επιπέδου. Αν ο κόμβος που επιλέχθηκε δεν είναι τελευταίο παιδί τοποθετούμε σε αυτόν έναν ερευνητή, καθαρίζοντας έτσι τις ακμές που συνδέουν τον κόμβο αυτόν με τα παιδιά του αφού στα άκρα τους θα υπάρχουν ερευνητές, και αν ο κόμβος αυτός είναι πρώτο παιδί ο ερευνητής που βρίσκεται σε αυτόν διασχίζει και καθαρίζει την ακμή που τον συνδέει με τον πατέρα του.

Μόλις η προηγούμενη διαδικασία ολοκληρωθεί, ανά 2 επίπεδα ξεκινώντας από το επίπεδο στο οποίο βρίσκεται ο ερευνητής που κινήθηκε τελευταίος και μέχρι τα φύλλα, επιλέγουμε έναν-έναν όλους τους κόμβους κάθε επιπέδου. Ο ερευνητής που βρίσκεται στον αντίστοιχο κόμβο διασχίζει και καθαρίζει την ακμή που συνδέει τον κόμβο του με το τελευταίο του παιδί.

4.4 Ανάλυση Αλγορίθμου

Λήμμα 4.4.1. *(Ορθότητα αλγορίθμου) Ο αλγόριθμός μας (**Αλγόριθμος 3**) καθαρίζει ένα πλήρες m-αδικό δέντρο n κόμβων.*

Απόδειξη.

- Ας υποθέσουμε ότι οι ερευνητές έχουν τοποθετηθεί στο δέντρο σύμφωνα με την περιγραφή που κάναμε στην παράγραφο 4.2.1. Στον αλγόριθμο μας η διδικασία του καθαρισμού ολοκληρώνεται εφόσον κάθε ερευνητής που βρίσκεται σε κόμβο που είναι πρώτο παιδί εκτελέσει 2 κινήσεις. Στην πρώτη κίνηση, στην οποία προτεραιότητα

έχει ο ερευνητής χαμηλότερου επιπέδου, κάθε ερευνητής που βρίσκεται σε κόμβο που είναι πρώτο παιδί μετακινείται στον πατέρα. Ας δούμε όμως αν αυτή η κίνηση αυτή μπορεί να πραγματοποιηθεί χωρίς να μολυνθεί ξανά μία καθαρή ακμή. Οι ερευνητές που βρίσκονται στο επίπεδο των φύλλων (έστω h) έχουν μόνο 1 προσπίπτουσα μολυσμένη ακμή στον κόμβο τους, γεγονός που επιτρέπει σε έναν από κάθε ομάδα ερευνητών που βρίσκονται σε κόμβους του ίδιου πατέρα, να μεταβεί στον πατέρα του κόμβου που βρίσκεται στο επίπεδο h-1 καθαρίζοντας την αντίστοιχη ακμή. Έτσι καθαρίζουν αυτόματα και οι ακμές που συνδέουν τους κόμβους του επιπέδου h-1 με τους κόμβους του επιπέδου h-2 στους οποίους υπάρχει ερευνητής, καθώς επίσης και οι ακμές που συνδέουν τους κόμβους του επιπέδου h-1 με εκείνα τα παιδιά του επιπέδου h στα οποία βρίσκονται ερευνητές. Βάσει αυτής της ανάλυσης, οι ερευνητές του επόμενου επιπέδου κίνησης (επίπεδο h-2) έχουν 1 προσπίπτουσα ακμή στον κόμβο τους και άρα ένας από τους ερευνητές κάθε ομάδας κόμβων μπορεί να κινηθεί, στη συνέχεια ένας από τους ερευνητές κάθε ομάδας κόμβων του επιπέδου h-4 μπορεί και αυτός να κινηθεί κ.ο.κ.. Όμως, σε κάθε επίπεδο η κίνηση των ερευνητών δεν εξαρτάται από την κίνηση άλλων ερευνητών του επιπέδου και επομένως κάθε ερευνητής που τοποθετείται σε κόμβο που είναι πρώτο παιδί μπορεί να κινηθεί αμέσως, ενώ αν το δέντρο είναι άρτιου ύψους ο ερευνητής που θα τοποθετηθεί στη ρίζα παραμένει ακίνητος αφού δεν θα υπάρχει στον κόμβο του μολυσμένη ακμή για να καθαρίσει.

- Στη δεύτερη κίνηση, όλοι οι ερευνητές που εκτέλεσαν την πρώτη τους κίνηση μεταφέρονται στο τελευταίο παιδί του κόμβου τους. Το πρώτο επίπεδο για τη δεύτερη κίνηση των ερευνητών είναι το επίπεδο 1-(h***mod***2) στο οποίο βρίσκεται ο ερευνητής που ολοκλήρωσε τελευταίος την πρώτη του κίνηση. Στο επίπεδο αυτό, εξαιτίας της πρώτης κίνησης των ερευνητών, προσπίπτει σε κάθε κόμβο 1 μόνο μολυσμένη ακμή την οποία κάθε ένας από τους ερευνητές μπορεί να καθαρίσει. Έτσι, θα μεταβούν στο τελευταίο παιδί του κόμβου τους, δηλαδή στο επίπεδο 1-(h***mod***2)+1 και αυτόματα καθαρίζουν και οι ακμές που συνδέουν τους κόμβους του επιπέδου 1-(h***mod***2)+1 στους οποίους βρίσκονται πλέον οι ερευνητές με τους κόμβους του επιπέδου 1-(h***mod***2)+2. Εξαιτίας αυτής της κίνησης μπορούν πλέον να κινηθούν οι ερευνητές του επιπέδου 1-(h***mod***2)+2 στο επίπεδο 1-(h***mod***2)+3, έπειτα οι ερευνητές του επιπέδου 1-(h***mod***2)+4 στο επίπεδο 1-(h***mod***2)+5 κ.ο.κ. αφού κάθε φορά στον αντίστοιχο κόμβο θα προσπίπτει 1 μολυσμένη ακμή.

- Οι γραμμές 1 και 2 εκτελούνται μόνο όταν ο κάτω βαθμός είναι 1, οπότε καθαρίζουμε μία γραμμική τοπολογία.

- Στη γραμμή 4 υπολογίζουμε το ύψος του δέντρου, γνωρίζοντας το πλήθος των κόμβων του.

- Στη γραμμή 5 επιλέγουμε εναλλάξ τα επίπεδα, από το επίπεδο των φύλλων (επίπεδο h) μέχρι τη ρίζα.

- Στη γραμμή 9, πέρνουμε έναν-έναν τους κόμβους του επιπέδου, ξεκινώντας από τον αριστερότερο κόμβο του επιπέδου.

- Στις γραμμές 10-16, αν ο κόμβος που επιλέξαμε δεν είναι τελευταίο παιδί τοποθετούμε 1 ερευνητή σε αυτόν (καθαρίζοντας έτσι τις ακμές που συνδέουν τον κόμβο με τα παιδία του) και αν ο κόμβος είναι πρώτο παιδί ο ερευνητής διασχίζει και καθαρίζει την ακμή που συνδέει τον κόμβο του με τον πατέρα του.

- Στη γραμμή 21 επιλέγουμε εναλλάξ τα επίπεδα, από το επίπεδο 1-(h*mod*2) μέχρι 1 επίπεδο πριν τα φύλλα.

- Στις γραμμές 25-28 επιλέγουμε από αριστερά προς τα δεξιά όλους τους κόμβους του επιπέδου και μετακινούμε κάθε ερευνητή στο τελευταίο παιδί του κόμβου του καθαρίζοντας έτσι την αντίστοιχη ακμή. □

Λήμμα 4.4.2. *(Πολυπλοκότητα αλγορίθμου) Ο αλγόριθμός μας (***Αλγόριθμος 3***) ολοκληρώνει την εκτέλεσή του μετά από Θ(n) βήματα, όπου n το πλήθος των κόμβων του δέντρου.*

Απόδειξη. Ο αλγόριθμος επιλέγει εναλλάξ τα επίπεδα και ελέγχει όλους τους κόμβους κάθε επιπέδου, τοποθετώντας και μετακινώντας τους ερευνητές σε 1 βήμα. Στη συνέχεια επιλέγει τα υπόλοιπα επίπεδα και ελέγχει όλους τους κόμβους κάθε επιπέδου μετακινώντας κατά 1 βήμα τους ερευνητές. Άρα, σε κάθε έναν από τους κόμβους που ελέγχονται γίνεται ένας σταθερός αριθμός από πράξεις. Επομένως, η πολυπλοκότητα του αλγορίθμου εξαρτάται από τον αριθμό των κόμβων του δέντρου και επομένως ισούται με $\Theta(n)$. Μια τυπικότερη απόδειξη, υπολογίζοντας το πλήθος εκτελέσεων κάθε γραμμής του αλγορίθμου, είναι η ακόλουθη:

- Η γραμμή 1 του αλγορίθμου εκτελείται 1 φορά. Αν η συνθήκη είναι αληθής θα εκτελεστεί η γραμμή 2 το κόστος της οποίας, όπως είναι γνωστό, είναι $\Theta(n)$ και προκύπτει από τον καθαρισμό μιας γραμμικής τοπολογίας. Αν η συνθήκη της γραμμής 1 είναι ψευδής, τότε θα εκτελεστούν οι γραμμές 4-30.

- Η γραμμή 4 εκτελείται 1 φορά με κόστος $\Theta(log_m n)$.

- Οι γραμμές 5-8 εκτελούνται $\lfloor \frac{h}{2} \rfloor + 1 = \lfloor \frac{\lfloor log_m n \rfloor}{2} \rfloor + 1$ φορές, όμως οι γραμμές 6-8 έχουν κόστος το οποίο δεν ξεπερνάει την τιμή του ύψους του δέντρου.

- Ο αριθμός εκτελέσεων των γραμμών 9 και 10 υπολογίζεται από τον τύπο,

$$\sum \{rmost - lmost + 1\} = \sum \{\tfrac{m^{i+1}-1}{m-1} - (\tfrac{m^i-1}{m-1} + 1) + 1\} =$$

$$\sum \{\tfrac{m^{i+1}-m^i}{m-1}\} = \sum \{\tfrac{m^i \cdot (m-1)}{m-1}\} = \sum \{m^i\}$$

  για κάθε τιμή που δίνεται στο i από τη γραμμή 5. Οπότε, αν a είναι ένας πραγματικός αριθμός, το πλήθος εκτελέσεων των γραμμών αυτών προκύπτει από τη σχέση,

$$m^h + m^{h-2} + m^{h-4} + ... = m^h \cdot (1 + \tfrac{1}{m^2} + \tfrac{1}{m^4} + ...) =$$

$$m^h \cdot \sum_{i=0}^{\lfloor \frac{h}{2} \rfloor} \frac{1}{m^{2i}} = \frac{1}{a} \cdot n \cdot \sum_{i=0}^{\lfloor \frac{\lfloor log_m n \rfloor}{2} \rfloor} \frac{1}{m^{2i}}$$

- Οι γραμμές 11-16, εκτελούνται μόνο όταν ο κόμβος δεν είναι τελευταίο παιδί ή ρίζα, οπότε θα εκτελουστούν λιγότερες φορές από ότι οι γραμμές 9 και 10.
- Οι γραμμές 21-24 εκτελούνται $\lceil \frac{h}{2} \rceil = \lceil \frac{\lfloor log_m n \rfloor}{2} \rceil$ φορές, ενώ οι γραμμές 22 και 23 έχουν κόστος που και πάλι δεν ξεπερνάει την τιμή του ύψους του δέντρου.
- Το πλήθος εκτελέσεων των γραμμών 25-28 προκύπτει από τον τύπο,

$$\sum \{rmost - lmost + 1\} = \sum \{\frac{m^{i+1}-1}{m-1} - (\frac{m^i-1}{m-1} + 1) + 1\} =$$

$$\sum \frac{m^{i+1}-m^i}{m-1} = \sum m^i$$

για κάθε τιμή που παίρνει το i από τη γραμμή 21. Δηλαδή, το πλήθος εκτελέσεων των γραμμών αυτών υπολογίζεται από τη σχέση,

$$m^{1-h\%2} + m^{3-h\%2} + ... = m^{h-(h-1)-h\%2} + m^{h-(h-3)-h\%2} + ... =$$

$$m^h \cdot (\frac{1}{m^{h-1+h\%2}} + \frac{1}{m^{h-3+h\%2}} + ...) = m^h \cdot \sum_{i=0}^{\lceil \frac{h}{2} \rceil - 1} \frac{1}{m^{2i+1}} =$$

$$\frac{1}{a} \cdot n \cdot \sum_{i=0}^{\lceil \frac{\lfloor log_m n \rfloor}{2} \rceil - 1} \frac{1}{m^{2i+1}}$$

- Στον επόμενο πίνακα βλέπουμε το κόστος και το πλήθος εκτελέσεων κάθε γραμμής του αλγορίθμου:

Γραμμή i	Κόστος c_i	Πλήθος εκτελέσεων
1	c_1	1
2	$\Theta(n)$	1
4	$\Theta(log_m n)$	1
5	c_5	$\lfloor \frac{\lfloor log_m n \rfloor}{2} \rfloor + 1$
6-8	$\Theta(log_m n)$	$\lfloor \frac{\lfloor log_m n \rfloor}{2} \rfloor + 1$
9-16	c_i	$\leq \frac{1}{a} \cdot n \cdot \sum_{i=0}^{\lfloor \frac{\lfloor log_m n \rfloor}{2} \rfloor} \frac{1}{m^{2i}}$
21, 24	c_i	$\lceil \frac{\lfloor log_m n \rfloor}{2} \rceil$
22, 23	$\Theta(log_m n)$	$\lceil \frac{\lfloor log_m n \rfloor}{2} \rceil$
25-28	c_i	$\frac{1}{a} \cdot n \cdot \sum_{i=0}^{\lceil \frac{\lfloor log_m n \rfloor}{2} \rceil - 1} \frac{1}{m^{2i+1}}$

Σύμφωνα λοιπόν με τον παραπάνω πίνακα και επειδή όλα τα αθροίσματα συγκλίνουν, η πολυπλοκότητα του αλγορίθμου είναι *Θ(n)*. □

4.5 Αριθμός Ερευνητών

Λήμμα 4.5.1. *(Άνω φράγμα) Ο αλγόριθμός μας (**Αλγόριθμος 3**) απαιτεί* $\lceil \frac{(m-1)n}{m+1} \rceil$ *ερευνητές, για τον καθαρισμό ενός πλήρους m-αδικού δέντρου n κόμβων.*

Απόδειξη. Όπως και στην απόδειξη του λήμματος 3.4.1 έτσι κι εδώ δημιουργούμε το δέντρο προσθέτοντας 2 επίπεδα σε κάθε βήμα. Έτσι, διακρίνουμε 2 περιπτώσεις:

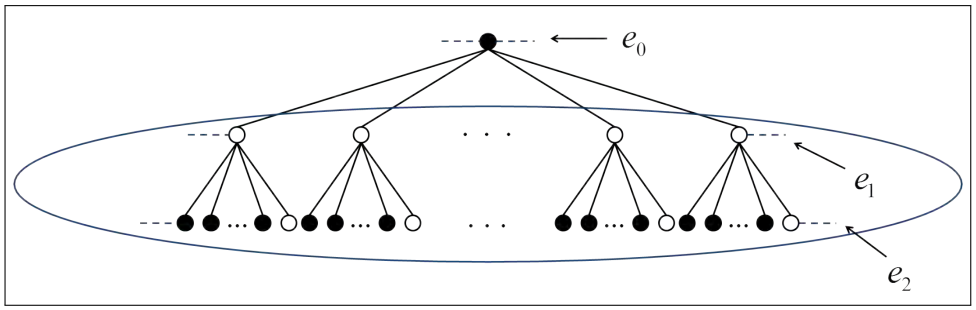
1. Αν το ύψος του δέντρου είναι άρτιος αριθμός

   - Αρχικά θα θεωρήσουμε ότι το ύψος του δέντρου είναι 0. Άρα θα έχουμε:

     m^0 κόμβους και m^0 ερευνητές

   - Έπειτα, θα αυξήσουμε κατά 2 το ύψος (Σχήμα 4.3), οπότε θα προστεθούν:

     $m^1 + m^2$ κόμβοι και $m^2 - m^1$ ερευνητές



Σχήμα 4.3

   - Αυξάνοντας και πάλι το ύψος του δέντρου κατά 2 (Σχήμα 4.4) θα προστεθούν ακόμη:

     $m^3 + m^4$ κόμβοι και $m^4 - m^3$ ερευνητές

   - Συνεχίζουμε τη διαδικασία αυτή, μέχρι το ύψος του να γίνει ίσο με h. Οπότε, θα προστεθούν επιπλέον:

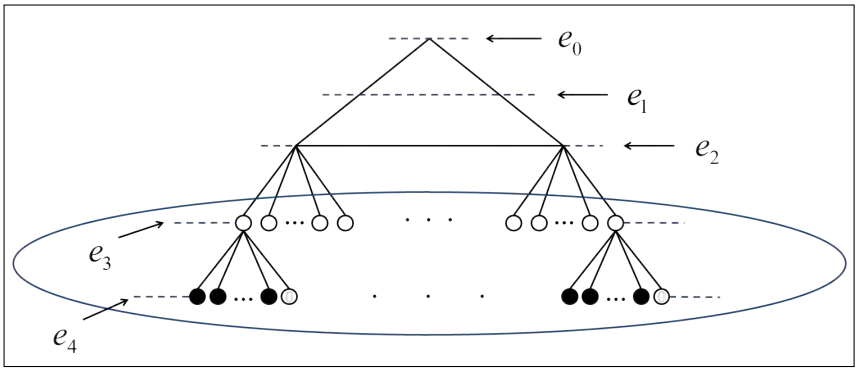
     $m^{h-1} + m^h$ κόμβοι και $m^h - m^{h-1}$ ερευνητές

   Επομένως, ο αριθμός των ερευνητών που θα χρειαστούμε συνολικά για ένα δέντρο άρτιου ύψους είναι:

$$m^0 + (m^2 - m^1) + (m^4 - m^3) + ... + (m^h - m^{h-1}) =$$

$$m^0 + m^2 + m^4 + ... + m^h - (m^1 + m^3 + ... + m^{h-1}) =$$

$$\sum_{i=0}^{h} m^i - (m^1 + m^3 + ... + m^{h-1}) - (m^1 + m^3 + ... + m^{h-1}) =$$



Σχήμα 4.4

$$\sum_{i=0}^{h} m^i - 2\sum_{i=0}^{\frac{h}{2}-1} m^{2i+1} = n - 2 \cdot m \cdot \sum_{i=0}^{\frac{h}{2}-1} m^{2i}$$

Γνωρίζουμε όμως ότι: $\sum_{i=0}^{a} b^i = \frac{(b^{a+1})-1}{b-1}$ και $\frac{m^{h+1}-1}{m-1} = n$. Οπότε συνεχίζοντας την παραπάνω σχέση έχουμε:

$$n - 2 \cdot m \cdot \sum_{i=0}^{\frac{h}{2}-1} m^{2i} = n - 2 \cdot m \cdot \frac{(m^2)^{\frac{h}{2}-1+1}-1}{m^2-1} =$$

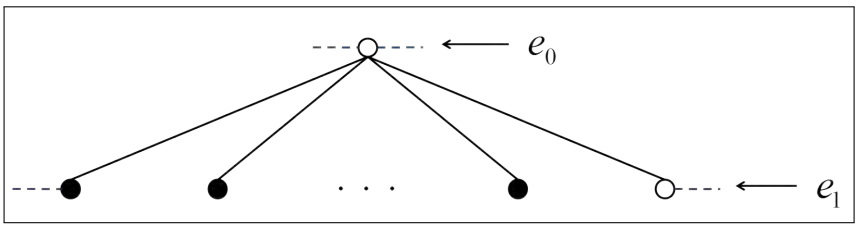
$$n - 2 \cdot \frac{m^{h+1}-1+1-m}{(m-1)(m+1)} = n - 2 \cdot (\frac{n}{m+1} - \frac{m-1}{(m-1)(m+1)}) = n - \frac{2n(m-1)-2(m-1)}{(m-1)(m+1)} =$$

$$n - \frac{2n-2}{m+1} = \frac{n(m+1)-2n+2}{m+1} = \frac{(m-1)n+2}{m+1}$$

2. Αν το ύψος του δέντρου είναι περιττός αριθμός

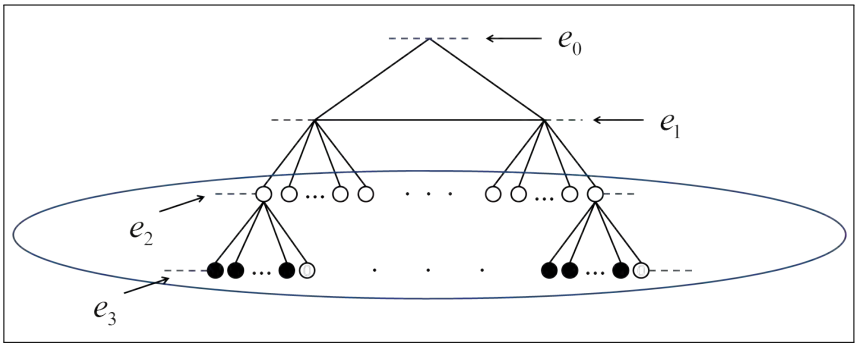
   - Ξεκινάμε υποθέτοντας ότι το δέντρο είναι ύψους 1 (Σχήμα 4.5). Τότε θα χρειαστούμε:

     $m^0 + m^1$ κόμβους και $m^1 - m^0$ ερευνητές



Σχήμα 4.5

   - Συνεχίζουμε αυξάνοντας κατά 2 το ύψος (Σχήμα 4.6), οπότε θα προστεθούν ακόμα:



Σχήμα 4.6

$$m^2 + m^3 \text{ κόμβοι} \qquad \text{και} \qquad m^3 - m^2 \text{ ερευνητές}$$

- Τέλος, όταν το ύψος του δέντρου γίνει h, θα προστεθούν επιπλέον:

$$m^{h-1} + m^h \text{ κόμβοι} \qquad \text{και} \qquad m^h - m^{h-1} \text{ ερευνητές}$$

Επομένως, αν το ύψος του δέντρου είναι περιττός αριθμός, τότε ο αριθμός των ερευνητών που θα χρειαστούμε είναι:

$$(m^1 - m^0) + (m^3 - m^2) + ... + (m^h - m^{h-1}) =$$

$$m^1 + m^3 + ... + m^h - (m^0 + m^2 + ... + m^{h-1}) =$$

$$\sum_{i=0}^{h} m^i - (m^0 + m^2 + ... + m^{h-1}) - (m^0 + m^2 + ... + m^{h-1}) =$$

$$\sum_{i=0}^{h} m^i - 2(m^0 + m^2 + ... + m^{h-1}) = \sum_{i=0}^{h} m^i - 2\sum_{i=0}^{\frac{h-1}{2}} m^{2i} =$$

$$n - 2 \cdot \sum_{i=0}^{\frac{h-1}{2}} m^{2i} = n - 2 \cdot \frac{(m^2)^{\frac{h-1}{2}+1} - 1}{m^2 - 1} = n - 2 \cdot \frac{m^{h+1} - 1}{(m-1)(m+1)} =$$

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

Οπότε, συνδυάζοντας τις περιπτώσεις 1 και 2, καταλήγουμε στο συμπέρασμα ότι για ένα πλήρες m-αδικό δέντρο αρκούν $\lceil \frac{(m-1)n}{m+1} \rceil$ ερευνητές για να το καθαρίσουν χρησιμοποιώντας μια exclusive μονότονη στρατηγική. □

Λήμμα 4.5.2. *(Κάτω φράγμα) Δεν υπάρχει αλγόριθμος ο οποίος να καθαρίζει ένα πλήρες m-αδικό δέντρο n κόμβων με λιγότερους από* $\lceil \frac{(m-1)n}{m+1} \rceil$*, με τη χρήση μιας exclusive μονότονης στρατηγικής.*

Απόδειξη. Ξεκινώντας από τα φύλλα, χωρίζουμε το δέντρο (A) σε πλήρη m-αδικά υποδέντρα $A_1, A_2, ..., A_k$ ύψους 1 έτσι ώστε,

$$A_1 \cap A_2 \cap ... \cap A_k = 0 \text{ και}$$

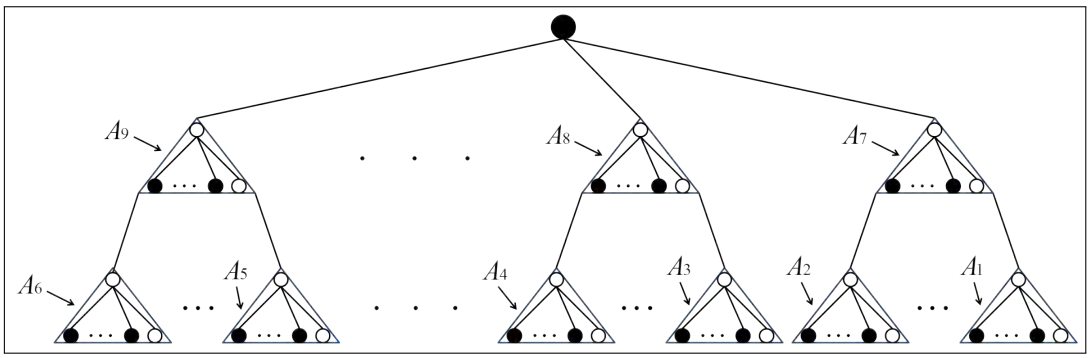
$$A_1 \cup A_2 \cup ... \cup A_k = A \,||\, A_1 \cup A_2 \cup ... \cup A_k = A - \{root\}$$

Έστω k ο αριθμός των υποδέντρων που δημιουργούνται. Βάσει της περιγραφής που δώσαμε στην παράγραφο 4.2.1 και παρατηρώντας το σχήμα 4.7, βλέπουμε ότι ο αριθμός των ερευνητών που χρησιμοποιεί ο αλγόριθμός μας για δέντρο άρτιου ύψους ισούται με,

$$k \cdot (m-1) + 1$$

ενώ για δέντρο περιττού ύψους ο αριθμός των ερευνητών που χρησιμοποιείται από το αλγόριθμό μας είναι ίσος με,

$$k \cdot (m-1)$$



Σχήμα 4.7

Ας ξεκινήσουμε τον καθαρισμό του δέντρο χρησιμοποιώντας έναν ερευνητή, έστω s_1. Έστω ότι αυτός φτάνει για πρώτη φορά στη ρίζα ενός υποδέντρου μέσω της ακμής $\{u_1, u\}$. Τότε, η ακμή $\{u_1, u\}$ λόγω της ολίσθησης καθαρίζει, ενώ ο s_1 βρίσκεται πλέον στη ρίζα u αυτού του υποδέντρου. Όμως στον u προσπίπτουν τώρα m μολυσμένες ακμές, έστω $\{u, u_2\}$, $\{u, u_3\}$,...,$\{u, u_{m+1}\}$, με αποτέλεσμα ο s_1 να μη μπορεί να απομακρυνθεί από τον κόμβο u, αφού αν το κάνει θα υπάρξει επαναμόλυνση. Επομένως χρειάζονται ακόμα $m-1$ ερευνητές. Έστω s_2, s_3,...,s_m οι ερευνητές αυτοί κάθε ένας από τους οποίους στέκεται σε κάποιον γειτονικό κόμβο του u, όχι όμως στον u_1. Έστω ότι οι ερευνητές στέκονται στους κόμβους u_2,...,u_m. Τότε, εκτός της $\{u_1, u\}$ που είναι ήδη καθαρή, καθαρές θα είναι και οι $\{u, u_2\}$, $\{u, u_3\}$,...,$\{u, u_m\}$, λόγω της ταυτόχρονης ύπαρξης ενός ερευνητή και στα δυο τους άκρα. Πλέον, στον u προσπίπτει μία μολυσμένη ακμή, η $\{u, u_{m+1}\}$, μέσω της οποίας ο s_1 μπορεί να απομακρυνθεί από τον u χωρίς να υπάρξει κίνδυνος επαναμόλυνσης.Όπως βλέπουμε οι ερευνητές s_1, s_2, s_3,...,s_m βρίσκονται σε κόμβο γειτονικό του{u}, ενώ δύο ερευνητές δεν επιτρέπεται να βρεθούν στον ίδιο κόμβο ταυτόχρονα. Επομένως, όλοι αυτοί οι ερευνητές θα εισέρχονται για πρώτη φορά σε έναν νέο κόμβο από την ίδια ακμή.

Ας θεωρήσουμε τώρα, $\{v_1, v\}$ την ακμή μέσω της οποίας ένας από τους παραπάνω ερευνητές εισέρχεται για πρώτη φορά στη ρίζα v ενός άλλου υποδέντρου. Έστω λοιπόν ότι ο

ερευνητής s_1 κινήθηκε κατά μήκος της $\{v_1, v\}$ καθρίζοντάς την. Πλέον, στον κόμβο v προσπίπτουν m μολυσμένες ακμές, έστω $\{v, v_2\}$, $\{v, v_3\}$,...,$\{v, v_{m+1}\}$. Όπως αναφέραμε και πιο πάνω, σε $m-1$ γειτονικούς κόμβους του v, εκτός του v_1, θα πρέπει να βρίσκεται 1 ερευνητής. Όμως αυτοί οι ερευνητές δεν μπορεί να είναι οι s_1, s_2,...,s_m, καθώς εισέρχονται για πρώτη φορά στον κόμβο v μέσω της $\{v_1, v\}$. Έστω s_{m+1}, s_{m+2},...,s_{2m-1} οι ερευνητές οι οποίοι στέκονται στους κόμβους v_2,...,v_m. Έτσι οι ακμές $\{v, v_2\}$, $\{v, v_3\}$,...,$\{v, v_{m+1}\}$ είναι καθαρές και ο s_1 μπορεί να απομακρυνθεί από τον v χωρίς να υπάρξει επαναμόλυνση. Όμοια με πριν, ο s_1 και οι s_{m+1}, s_{m+2},...,s_{2m-1} εισέρχονται για πρώτη φορά σε έναν νέο κόμβο από την ίδια ακμή. Όμως, και οι ερευνητές s_2,...,s_m, εισέρχονται για πρώτη φορά σε έναν κόμβο από την ίδια ακμή που εισέρχεται στον κόμβο αυτό και ο s_1. Δηλαδή όλοι οι ερευνητές που υπάρχουν στο δέντρο εισέρχονται για πρώτη φορά σε έναν νέο κόμβο από την ίδια ακμή.

Όπως γίνεται αντιληπτό, εκτός από τον αρχικό ερευνητή με τον οποίο ξεκινήσαμε να καθαρίζουμε το δέντρο, για κάθε υποδέντρο (εκτός από το υποδέντρο ρίζα του οποίου είναι η ρίζα του δέντρου) εισάγονται $m-1$ ερευνητές στο δέντρο. Όταν η ρίζα του δέντρου ανήκει σε κάποιο υποδέντρο (δηλαδή όταν το δέντρο είναι περιττού βαθμού), τότε για το συγκεκριμένο υποδέντρο θα πρέπει να εισαχθούν $m-2$ ερευνητές, διότι στη ρίζα του συγκεκριμένου υποδέντρου προσπίπτει μία ακμή λιγότερη. Άρα:

- Για δέντρα άρτιου βαθμού, δηλαδή για δέντρα των οποίων η ρίζα δεν ανήκει σε υποδέντρο χρειαζόμαστε τουλάχιστον,

$$k \cdot (m-1) + 1 \quad \text{ερευνητές.}$$

- Για δέντρα περιττού βαθμού, δηλαδή για δέντρα των οποίων η ρίζα ανήκει σε υποδέντρο χρειαζόμαστε τουλάχιστον,

$$(k-1) \cdot (m-1) + (m-2) + 1 =$$

$$k \cdot (m-1) \quad \text{ερευνητές.}$$

□

Το συμπέρασμα των λημμάτων 4.5.1 και 4.5.2, αφού το άνω φράγμα συμπίπτει με το κάτω φράγμα, συνοψίζεται στο ακόλουθο λήμμα:

Λήμμα 4.5.3. *Ο αλγόριθμός μας χρησιμοποιεί των ελάχιστο αριθμό ερευνητών και άρα είναι βέλτιστος.*

Χρησιμοποιώντας το αποτέλεσμα του λήμματος 4.5.1 παραθέτουμε τον παρακάτω αλγόριθμο, ο οποίος υπολογίζει το search number *xms(T)* που απαιτείται για τον καθαρισμό ενός πλήρους m-αδικού δέντρου n κόμβων με τη χρήση μιας exclusive μονότονης στρατηγικής:

Αλγόριθμος 4 *Υπολογισμός xms(T) για πλήρη m-αδικά δέντρα*

Είσοδος: *κάτω βαθμός m, πλήθος κόμβων n*

Έξοδος: *xms(T)*
1: *xmsn* ← *((m-1)*n+2)***div***(m+1)*
2: ***return*** *xmsn*

Λήμμα 4.5.4. *Ο αλγόριθμος υπολογισμού του xms(T) (***Αλγόριθμος 4***) ολοκληρώνει την εκτέλεσή του μετά από σταθερό αριθμό βημάτων.*

4.6 Βασικοί Υπολογισμοί

4.6.1 Αριθμός Συνολικών Κινήσεων

Λήμμα 4.6.1. *Σύμφωνα με τον αλγόριθμό μας (***Αλγόριθμος 3***), ένα πλήρες m-αδικό δέντρο n κόμβων καθαρίζεται με* $2 \cdot \lfloor \frac{n}{m+1} \rfloor$ *κινήσεις.*

Απόδειξη. Σύμφωνα με τον αλγόριθμο, για τον καθαρισμό του δέντρου απαιτούνται 2 κινήσεις από κάθε ερευνητή που βρίσκεται αρχικά τοποθετημένος σε κόμβο ο οποίος είναι πρώτο παιδί, ενώ αν στη ρίζα υπαρχει αρχικά τοποθετημένος ερευνητής, δηλαδή το ύψος του δέντρου (έστω h) είναι άρτιος αριθμός, τότε αυτός παραμένει ακίνητος κατά τη διάρκεια του καθαρισμού. Όμως, το πλήθος των κινούμενων ερευνητών σε ένα επίπεδο i ισούται με το πλήθος των κόμβων στο επίπεδο i-1. Οπότε:

1. Αν το ύψος του δέντρου είναι άρτιος αριθμός, τότε ο αριθμός των ερευνητών που κινούνται είναι:

$$m^{h-1} + m^{h-3} + ... + m^3 + m^1 = \sum_{i=0}^{\frac{h}{2}-1} m^{2i+1} =$$

$$m \cdot \frac{(m^2)^{\frac{h}{2}-1+1}-1}{m^2-1} = \frac{m^{h+1}-1+1-m}{(m-1)(m+1)} = \frac{n(m-1)-(m-1)}{(m-1)(m+1)} = \frac{n-1}{m+1}$$

   Και επειδή κάθε ερευνητής χρειάζεται 2 κινήσεις για τον καθαρισμό του δέντρου, τότε ο συνολικός αριθμός κινήσεων που απαιτείται ισούται με $2 \cdot \frac{n-1}{m+1}$

2. Αν το ύψος του δέντρου είναι περιττός αριθμός, τότε ο αριθμός των κινούμενων ερευνητών είναι:

$$m^{h-1} + m^{h-3} + ... + m^2 + m^0 = \sum_{i=0}^{\frac{h-1}{2}} m^{2i} =$$

$$\frac{(m^2)^{\frac{h-1}{2}+1}-1}{m^2-1} = \frac{m^{h+1}-1}{(m-1)(m+1)} = \frac{n}{m+1}$$

   Επομένως, ο συνολικός αριθμός κινήσεων που απαιτείται για τον καθαρισμό του δέντρου είναι ισούται με $2 \cdot \frac{n}{m+1}$.

Από τις περιπτώσεις 1 και 2 προκύπτει το συμπέρασμα ότι ο συνολικός αριθμός κινήσεων των ερευνητών ισούται με $2 \cdot \lfloor \frac{n}{m+1} \rfloor$ □

4.6.2 Αριθμός Βημάτων

Ομοίως με την απόδειξη του λήμματος 3.5.2 για τον αριθμό βημάτων σε ένα πλήρες δυαδικό δέντρο και γνωρίζοντας ότι σε καθε επίπεδο ενός πλήρους m-αδικού δέντρου κινούνται μόνο οι ερευνητές που βρίσκονται σε κόμβους που είναι πρώτα παιδιά, αποδεικνύεται και το παρακάτω λήμμα:

Λήμμα 4.6.2. *Σύμφωνα με τον αλγόριθμό μας (***Αλγόριθμος 3***), για τον καθαρισμό ενός πλήρους m-αδικού δέντρου n κόμβων, αρκούν:*

- $\lfloor log_m n \rfloor$ *βήματα, αν το ύψος του δέντρου είναι άρτιος αριθμός και*
- $\lceil log_m n \rceil$ *βήματα, αν το ύψος του δέντρου είναι περιττός αριθμός.*

Βιβλιογραφία

[1] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[2] Lali Barrière, Paola Flocchini, Fedor V Fomin, Pierre Fraigniaud, Nicolas Nisse, Nicola Santoro, and Dimitrios M Thilikos. Connected graph searching. *Information and Computation*, 219:1–16, 2012.

[3] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 200–209. ACM, 2002.

[4] Lali Barriere, Pierre Fraigniaud, Nicola Santoro, and Dimitrios M Thilikos. Searching is not jumping. In *Graph-Theoretic Concepts in Computer Science*, pages 34–45. Springer, 2003.

[5] Daniel Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science*, 5:33–49, 1991.

[6] Daniel Bienstock and Paul Seymour. Monotonicity in graph searching. *Journal of Algorithms*, 12(2):239–245, 1991.

[7] Lélia Blin, Janna Burman, and Nicolas Nisse. Exclusive graph searching. In *Algorithms–ESA 2013*, pages 181–192. Springer, 2013.

[8] Richard Breisch. An intuitive approach to speleotopology. *Southwestern cavers*, 6(5):72–78, 1967.

[9] Paola Flocchini, Miao Jim Huang, and Flaminia L Luccio. Contiguous search in the hypercube for capturing an intruder. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 62–62. IEEE, 2005.

[10] Paola Flocchini, Miao Jun Huang, and Flaminia L Luccio. Decontaminating chordal rings and tori using mobile agents. *International Journal of Foundations of Computer Science*, 18(03):547–563, 2007.

[11] Paola Flocchini, Miao Jun Huang, and Flaminia L Luccio. Decontamination of hypercubes by mobile agents. *Networks*, 52(3):167–178, 2008.

[12] Fedor V Fomin and Dimitrios M Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.

[13] Fedor V Fomin, Dimitrios M Thilikos, and Ioan Todinca. Connected graph searching in outerplanar graphs. *Electronic Notes in Discrete Mathematics*, 22:213–216, 2005.

[14] Pierre Fraigniaud and Nicolas Nisse. Connected treewidth and connected graph searching. In *LATIN 2006: Theoretical Informatics*, pages 479–490. Springer, 2006.

[15] Jens Gusted. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3):233–248, 1993.

[16] Toshinobu Kashiwabara and Toshio Fujisawa. Np-completeness of the problem of finding a minimum clique number interval graph containing a given graph as a subgraph. In *Proc. Symposium of Circuits and Systems*, 1979.

[17] Lefteris M Kirousis and Christos H Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47:205–218, 1986.

[18] Andrea S LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM (JACM)*, 40(2):224–245, 1993.

[19] Euripides Markou, Nicolas Nisse, and Stéphane Pérennes. Exclusive graph searching vs. pathwidth. 2014.

[20] Nimrod Megiddo, S Louis Hakimi, Michael R Garey, David S Johnson, and Christos H Papadimitriou. The complexity of searching a graph. *Journal of the ACM (JACM)*, 35(1):18–44, 1988.

[21] Rodica Mihai and Ioan Todinca. Pathwidth is np-hard for weighted trees. In *Frontiers in Algorithmics*, pages 181–195. Springer, 2009.

[22] Nicolas Nisse. Connected graph searching in chordal graphs. *Discrete Applied Mathematics*, 157(12):2603–2610, 2009.

[23] Torrence D Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Springer, 1978.

[24] Torrence D Parsons. The search number of a connected graph. In *Proc. 9th South-Eastern Conf. on Combinatorics, Graph Theory, and Computing*, pages 549–554, 1978.

[25] Neil Robertson and Paul D Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

[26] Paul D Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993.

[27] Boting Yang, Danny Dyer, and Brian Alspach. Sweeping graphs with large clique number. In *Algorithms and computation*, pages 908–920. Springer, 2005.

Παράρτημα Α΄

Υλοποίηση Αλγορίθμου σε C

Α΄.1 Ο Κώδικας

```
#include <stdio.h>
#include <stdlib.h>

typedef struct tree_node
{
   int searcher;
   int move;
}tree_node;

void cleaningFRT(tree_node *tree,int n,int m);
void lineTopology(tree_node *tree,int n);
int leftmostNode(int m,int level);
int rightmostNode(int m,int level);
void setSearcher(tree_node *tree,int m,int node);
int moveSearcher(tree_node *tree,int m,int node,int new_node);
int nextLastChild(int m,int node);
void clearEdges(tree_node *tree,int m,int h,int lev,int node);
void checkTree();
int heightOfTree(int n,int m);
int myPow(int a,int b);
tree_node *input(int *n,int *m);
void display(tree_node *tree,int n,int m);

int label_of_searcher=0,edges,clear_edges=0;

int main(int argc, char *argv[])
{
   int n,m;
   tree_node *tree;

   tree=input(&n,&m);
   cleaningFRT(tree,n,m);
```

```
33
34    system("PAUSE");
35    return 0;
36 }
37
38 void cleaningFRT(tree_node *tree,int n,int m)
39 {
40    int i,u,h,lmost,rmost,new_position;
41
42    edges=n-1;
43    if(m==1)
44       lineTopology(tree,n);
45    else
46    {
47       h=heightOfTree(n,m);
48       printf("\n******** Setting and First move ********\n\n");
49       for(i=h;i>=0;i=i-2)
50       {
51          lmost=leftmostNode(m,i);
52          rmost=rightmostNode(m,i);
53          new_position=leftmostNode(m,i-1);
54          for(u=lmost;u<=rmost;u++)
55          {
56             if((u-1)%m!=0 || u==1)
57             {
58                setSearcher(tree,m,u);
59                printf("- - - Set new searcher- - -\n\n");
60                display(tree,n,m);
61                clearEdges(tree,m,h,i,u);
62                printf("- - - Move searcher - - -\n\n");
63                if(moveSearcher(tree,m,u,new_position)==1)
64                {
65                   clearEdges(tree,m,h,i,new_position);
66                   new_position++;
67                }
68                display(tree,n,m);
69             }
70          }
71       }
72       printf("******** Second move of searchers ********\n\n");
73       for(i=1-(h%2);i<h;i=i+2)
74       {
75          lmost=leftmostNode(m,i);
76          rmost=rightmostNode(m,i);
77          new_position=rmost;
78          for(u=lmost;u<=rmost;u++)
79          {
80             new_position=nextLastChild(m,new_position);
81             moveSearcher(tree,m,u,new_position);
82             clearEdges(tree,m,h,i,new_position);
83          }
84       }
```

```
85          display(tree,n,m);
86       }
87       checkTree();
88    }
89
90    void lineTopology(tree_node *tree,int n)
91    {
92       int node;
93
94       node=leftmostNode(1,n-1);
95       setSearcher(tree,1,node);
96       tree[node].move=1;
97       clearEdges(tree,1,n-1,node-1,node);
98       while(node>1)
99       {
100         moveSearcher(tree,1,node,node-1);
101         clearEdges(tree,1,n-1,node-2,node-1);
102         node--;
103         tree[node].move=1;
104      }
105      tree[node].move=0;
106   }
107
108   int leftmostNode(int m,int level)
109   {
110      int leftmost_node;
111
112      if(m==1)
113         leftmost_node=level+1;
114      else
115         leftmost_node=(myPow(m,level)-1)/(m-1)+1;
116      return leftmost_node;
117   }
118
119   int rightmostNode(int m,int level)
120   {
121      int rightmost_node;
122
123      if(m==1)
124         rightmost_node=level+1;
125      else
126         rightmost_node=(myPow(m,level+1)-1)/(m-1);
127      return rightmost_node;
128   }
129
130   void setSearcher(tree_node *tree,int m,int node)
131   {
132      label_of_searcher++;
133      tree[node].searcher=label_of_searcher;
134      if(node==1)
135         tree[node].move=-1;
136      else if((node-1)%m==1)
```

```
137         tree[node].move=1;
138     else
139         tree[node].move=0;
140 }
141
142 int moveSearcher(tree_node *tree,int m,int node,int new_node)
143 {
144     if(tree[node].move==1)
145     {
146         tree[new_node].searcher=tree[node].searcher;
147         tree[new_node].move=2;
148         tree[node].searcher=0;
149         tree[node].move=0;
150         return 1;
151     }
152     else if(tree[node].move==2)
153     {
154         tree[new_node].searcher=tree[node].searcher;
155         tree[new_node].move=0;
156         tree[node].searcher=0;
157         tree[node].move=0;
158         return 2;
159     }
160     else
161         return 0;
162 }
163
164 int nextLastChild(int m,int node)
165 {
166     node=node+m-((node-1)%m);
167     return node;
168 }
169
170 void clearEdges(tree_node *tree,int m,int h,int lev,int node)
171 {
172     if(tree[node].move==1||tree[node].move==-1)
173     {
174         if(lev!=h)
175             clear_edges=clear_edges+m;
176     }
177     else if(tree[node].move==2)
178             clear_edges++;
179     else
180     {
181         if(lev==h-1||lev==h)
182             clear_edges++;
183         else
184             clear_edges=clear_edges+(m+1);
185     }
186 }
187
188 void checkTree()
```

```
189 {
190    if(clear_edges==edges)
191       printf("\n***** Tree is clear *****\n\n");
192    else
193       printf("\n***** Tree is contaminated *****\n\n");
194 }
195
196 int heightOfTree(int n,int m)
197 {
198    int div=n,height=0;
199
200    if(m==1)
201       height=n-1;
202    else
203    {
204       while(div>1)
205       {
206          height++;
207          div=div/m;
208       }
209    }
210    return height;
211 }
212
213 int myPow(int a,int b)
214 {
215    int pow=1,i;
216
217    for(i=0;i<b;i++)
218       pow=a*pow;
219    return pow;
220 }
221
222 tree_node *input(int *n,int *m)
223 {
224    int height,i;
225    tree_node *tree;
226
227    printf("Give m: ");
228    scanf("%d",m);
229    printf("Give height of tree:");
230    scanf("%d",&height);
231    *n=rightmostNode(*m,height);
232    tree=(tree_node *)malloc(sizeof(tree_node)*(*n+1));
233    for(i=0;i<=*n;i++)
234    {
235       tree[i].searcher=-1;
236       tree[i].move=-1;
237    }
238    return tree;
239 }
240
```

```
241 void display(tree_node *tree,int n,int m)
242 {
243    int lmost,rmost,h,i,u;
244
245    i=0;
246    lmost=leftmostNode(m,i);
247    rmost=rightmostNode(m,i);
248    h=heightOfTree(n,m);
249    while(i<=h)
250    {
251       printf("\nLevel %d\n",i);
252       printf("________\n\n");
253       for(u=lmost;u<=rmost;u++)
254          printf(" %d\t",tree[u].searcher);
255       printf("\n\n\n");
256       i++;
257       lmost=leftmostNode(m,i);
258       rmost=rightmostNode(m,i);
259    }
260 }
```

Α΄.2 Παράδειγμα Εκτέλεσης Κώδικα

```
Give m: 3
Give height of tree:2
********* Setting and First move *********
- - - Set new searcher- - -

Level 0
_____
 -1

Level 1
_____
 -1       -1       -1

Level 2
_____
 1        -1       -1       -1       -1       -1       -1       -1       -1

- - - Move searcher - - -

Level 0
_____
 -1

Level 1
_____
 1        -1       -1

Level 2
_____
 0        -1       -1       -1       -1       -1       -1       -1       -1
```

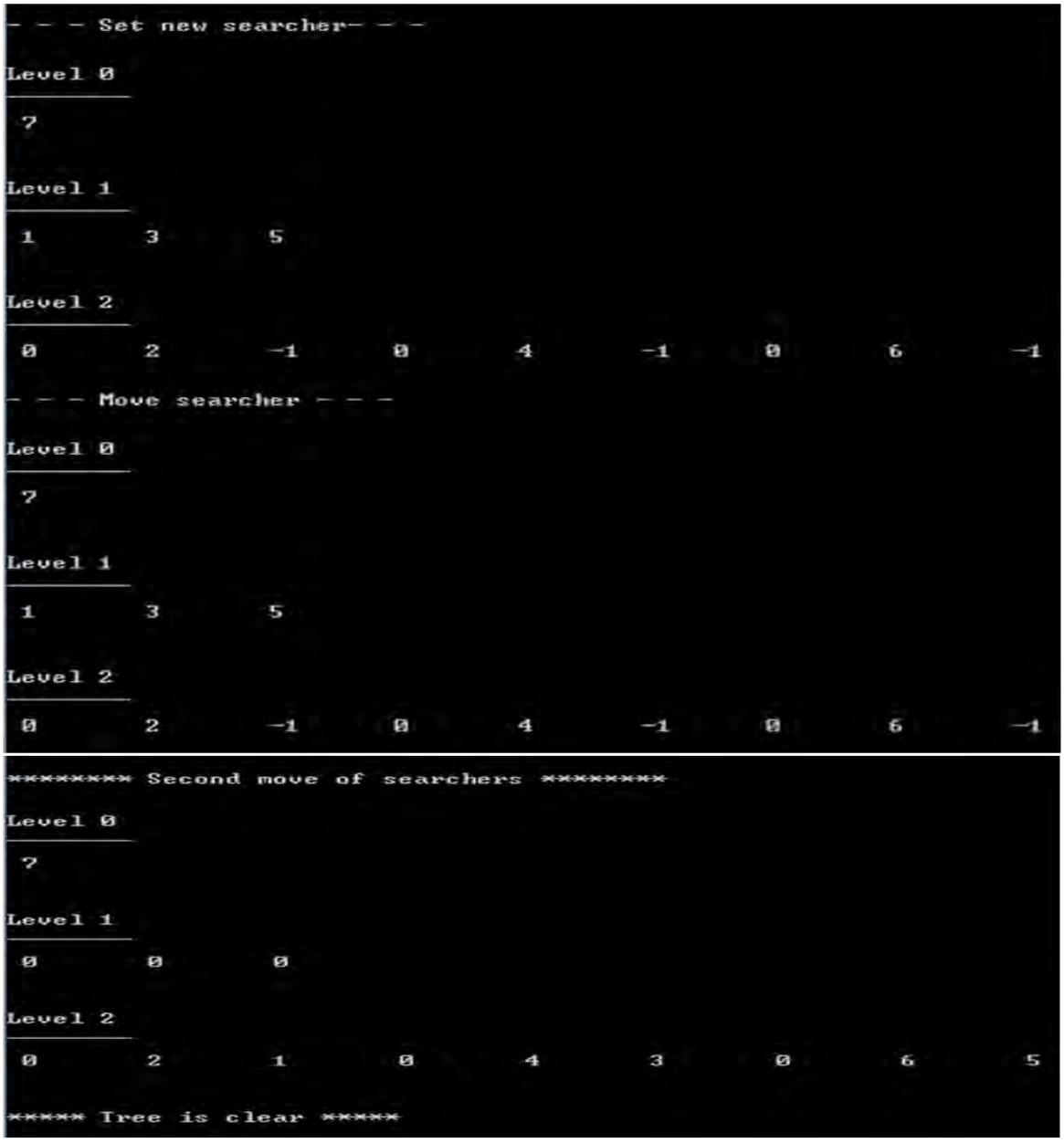
```
- - - Set new searcher- - -

Level 0
_____
 -1

Level 1
_____
 1        -1       -1

Level 2
_____
 0        2        -1       -1       -1       -1       -1       -1       -1

- - - Move searcher - - -

Level 0
_____
 -1

Level 1
_____
 1        -1       -1

Level 2
_____
 0        2        -1       -1       -1       -1       -1       -1       -1
```

```
- - - Set new searcher- - -
Level 0
-1
Level 1
1       -1      -1
Level 2
0       2       -1      3       -1      -1      -1      -1      -1
- - - Move searcher - - -
Level 0
-1
Level 1
1       3       -1
Level 2
0       2       -1      0       -1      -1      -1      -1      -1
```

```
- - - Set new searcher- - -
Level 0
-1
Level 1
1       3       -1
Level 2
0       2       -1      0       4       -1      -1      -1      -1
- - - Move searcher - - -
Level 0
-1
Level 1
1       3       -1
Level 2
0       2       -1      0       4       -1      -1      -1      -1
```

```
- - - Set new searcher- - -
Level 0
 -1
Level 1
 1        3        -1
Level 2
 0        2        -1       0        4        -1       5        -1       -1
- - - Move searcher - - -
Level 0
 -1
Level 1
 1        3        5
Level 2
 0        2        -1       0        4        -1       0        -1       -1
```

```
- - - Set new searcher- - -
Level 0
 -1
Level 1
 1        3        5
Level 2
 0        2        -1       0        4        -1       0        6        -1
- - - Move searcher - - -
Level 0
 -1
Level 1
 1        3        5
Level 2
 0        2        -1       0        4        -1       0        6        -1
```



- Αρχικά ζητείται ο κάτω βαθμός *m* και το ύψος του δέντρου, έτσι ώστε να δημιουργηθεί το δέντρο. Έστω λοιπόν *m=3* και *height=2*.

- Συμβολίζουμε με *Level i* το επίπεδο *i* του δέντρου, όπου *i=0, 1,. . .,height*.

- Σε κάθε επίπεδο i διακρίνουμε τόσες τιμές όσοι είναι και οι κόμβοι του επιπέδου i του δέντρου.

- Η τιμή -1 αντιστοιχεί σε κόμβο στον οποίο μέχρι εκείνη τη στιγμή δεν έχει βρεθεί ποτέ ερευνητής.

- Η τιμή 0 αντιστοιχεί σε κόμβο στον οποίο έχει βρεθεί ερευνητής αλλά έχει φύγει από αυτόν.
- Κάθε τιμή μεγαλύτερη του 0 αντιστοιχεί σε κόμβο όπου ήδη βρίσκεται ερευνητής.