ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
K24: Προγραμματισμός Συστήματος – Εαρινό Εξάμηνο 2013
1η Προγραμματιστική Εργασία
Ημερομηνία Ανακοίνωσης: 7/3/2013
Ημερομηνία Υποβολής: 28/3/2013

Εισαγωγή στην Εργασία:
Ο στόχος αυτής της εργασίας είναι να εξοικειωθείτε με τον προγραμματισμό στην γλώσσα C (ή C++) και στην δημιουργία δομών δεδομένων.
Καλείστε να υλοποιήσετε ένα πρόγραμμα που θα συγκρίνει δύο αρχεία και θα τυπώνει στην κονσόλα τον αριθμό κάθε σελίδας στην οποία διαφέρουν, χρησιμοποιώντας δέντρα κατακερματισμού (hash trees) υποστηριζόμενα από κρυπτογραφικές συναρτήσεις κατακερματισμού (cryptographic hash functions).

Διαδικαστικά:
- Το πρόγραμμά σας θα πρέπει να τρέχει στα μηχανήματα Linux/Unix της σχολής.
- Υπεύθυνοι για την άσκηση αυτή (ερωτήσεις, αξιολόγηση, βαθμολόγηση, κτλ) είναι όλοι οι βοηθοί. Ονόματα και email θα βρείτε στην ιστοσελίδα του μαθήματος.
- Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε τη λίστα του μαθήματος και το URL: http://cgi.di.uoa.gr/~mema/courses/k24/k24.html. Εγγραφείτε στην ηλεκτρονική λίστα (mailman) του μαθήματος και παρακολουθείτε ερωτήσεις/απαντήσεις/διευκρινήσεις που δίνονται σχετικά με την άσκηση (σημείωση: η η-λίστα αυτή δεν έχει καμία σχέση με την hard-copy λίστα που κυκλοφόρησε στα πρώτα μαθήματα και στην οποία θα έπρεπε να γράψετε το όνομά σας και το Unix user-id σας). **Η παρακολούθηση της ηλεκτρονικής λίστας είναι υποχρεωτική.**

Τι πρέπει να παραδοθεί:
1. Όλη η δουλειά σας σε ένα tar-file που να περιέχει όλα τα source files, header files, makefile. ΠΡΟΣΟΧΗ: φροντίστε να τροποποιήσετε τα δικαιώματα αυτού του αρχείου πριν την υποβολή, π.χ. με chmod 755 OnomaEponymoProject1.tar (περισσότερα στη σελίδα του μαθήματος).
2. Μια σύντομη περιγραφή (2-3 σελίδες) για τις επιλογές που κάνατε στο σχεδιασμό της άσκησης, σε μορφή PDF.
3. Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο θα πρέπει να αναφερθεί και στον πηγαίο κώδικά σας αλλά και στην παραπάνω αναφορά.

Υπόβαθρο

Οι κρυπτογραφικές συναρτήσεις κατακερματισμού (cryptographic hash functions ή απλώς, hash functions) παράγουν σαν έξοδο μια σύνοψη (digest) από ένα μήνυμα (message) που τους δίνεται στην είσοδο. Το μήνυμα εισόδου είναι μία συμβολοσειρά (char sequence) αυθαίρετου μήκους, δηλαδή, δεν υπάρχει κανένας περιορισμός ως προς το μέγεθος ή/και το περιεχόμενο του μηνύματος. Όμως, η σύνοψη που παράγουν σαν έξοδο είναι πάντα μία συμβολοσειρά σταθερού μεγέθους (π.χ. 16 bytes). Βέβαια, λόγω του περιορισμένου μεγέθους των συνόψεων, από την μία, και του αυθαίρετου μεγέθους των μηνυμάτων, από την άλλη, είναι αδύνατον να δημιουργηθεί μία ένα-προς-ένα αντιστοιχία ανάμεσα στο σύνολο τιμών των μηνυμάτων (το οποίο είναι αριθμησίμως άπειρο) και στο σύνολο τιμών των συνόψεων (το οποίο είναι πεπερασμένο). Συνεπώς, είναι πιθανό δύο διαφορετικά μηνύματα εισόδου να παράξουν την ίδια ακριβώς σύνοψη. Προφανώς, για να θεωρηθεί μία συνάρτηση κατακερματισμού αξιόπιστη (θέλουμε να μην συμβαίνει "ποτέ" το προαναφερθέν σενάριο) θα πρέπει μαθηματικά να έχει αποδειχτεί ότι αυτή η πιθανότητα είναι πολύ μικρή. Αυτή αποτελεί μόνο μία από τις ποιότητες που θέλουμε να μας παρέχει μία τέτοια συνάρτηση οι οποίες, συγκεντρωτικά, έχουν ως εξής:

- Δεδομένου μίας σύνοψης και μόνο αυτής, είναι αδύνατον να παράγουμε ένα μήνυμα που να έχει αυτή τη σύνοψη, δηλαδή να "αντιστρέψουμε" την διαδικασία παραγωγής. Χάρη σε αυτό το χαρακτηριστικό, οι συναρτήσεις αυτές είναι γνωστές και ως "συναρτήσεις μονής διαδρομής" (one-way functions). Προσοχή, όταν χρησιμοποιούμε τον όρο "αδύνατον" εννοούμε ότι είναι υπολογιστικά πολύ δύσκολο. Η ιδιότητα αυτή ονομάζεται pre-image resistance.
- Δεδομένου ενός μηνύματος, το οποίο προφανώς σημαίνει ότι ξέρουμε και την τιμή κατακερματισμού του, είναι αδύνατον να βρούμε ένα άλλο διαφορετικό μήνυμα με την ίδια σύνοψη. Η ιδιότητα αυτή ονομάζεται second pre-image resistance.
- Είναι αδύνατον να βρούμε δύο διαφορετικά μηνύματα με την ίδια σύνοψη. Η ιδιότητα αυτή ονομάζεται collision resistance.

Λόγω των παραπάνω ιδιοτήτων τους χρησιμοποιούνται σε μία πληθώρα από εφαρμογές όπως η επαλήθευση κωδικών, η εξακρίβωση για το αν ένα μήνυμα έχει υποστεί αλλαγές κατά την μετάδοσή του (ηθελημένα ή μη), η γέννεση ψευδό-τυχαίων αριθμών (pseudorandom number generators) κ.α. Παραδείγματα τέτοιων συναρτήσεων αποτελούν οι MD5 και SHA-1.

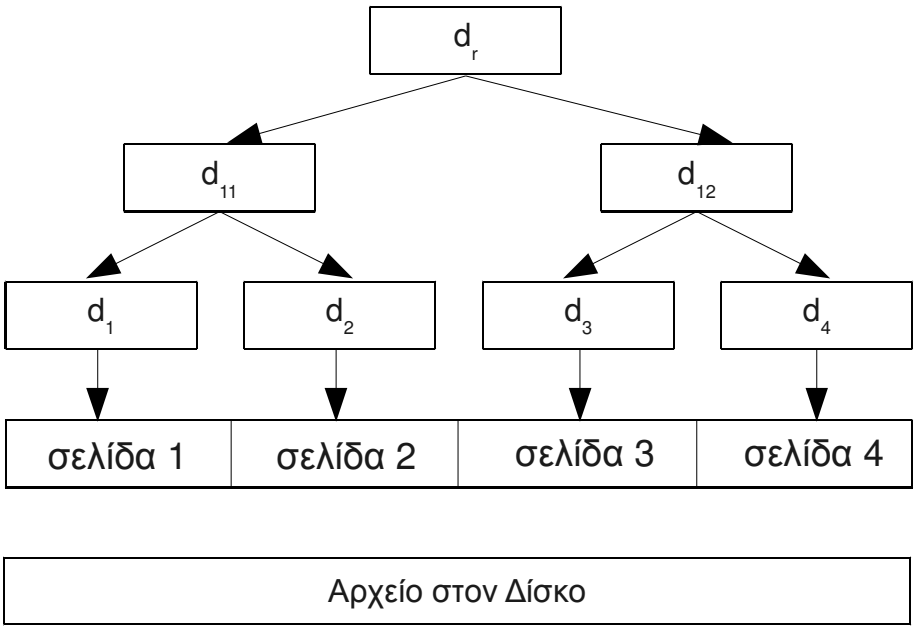
Στο πλαίσιο αυτής της εργασίας, η κρυπτογραφική συνάρτηση κατακερματισμού που θα χρησιμοποιήσουμε είναι η MD5. Η υλοποίηση της υπάρχει διαθέσιμη στην ιστοσελίδα του μαθήματος.

Δέντρα κατακερματισμού

Τα δέντρα κατακερματισμού είναι δεντρικές δομές δεδομένων των οποίων οι κόμβοι

περιέχουν πληροφορία, μέσω συνόψεων, για την κατάσταση ενός μεγαλύτερου (σε όγκο) δεδομένου, πχ. ενός αρχείου στο δίσκο. Εφευρέθηκαν το 1979 από τον Ralph Merkle και για αυτό πολλές φορές αναφερόμαστε σε αυτά και ως δέντρα Merkle (Merkle trees).

Στο παρακάτω σχήμα (Σχήμα 1) απεικονίζεται ένα παράδειγμα για το πώς μπορούμε να χρησιμοποιήσουμε ένα δέντρο Merkle με σκοπό να οργανώσουμε την πληροφορία που περιέχει ένα αρχείο που βρίσκεται στον δίσκο.



Σχήμα 1: Παράδειγμα δέντρου κατακερματισμού για ένα αρχείο που βρίσκεται στο δίσκο και έχει μέγεθος 4 σελίδες.

Στο κάτω μέρος του σχήματος αναπαρίσταται το αρχείο σαν ένα συνεχόμενο κομμάτι από bytes το οποίο, στο αμέσως επόμενο επίπεδο, χωρίζεται σε τμήματα ίσου μήκους τα οποία ονομάζονται σελίδες. Θεωρούμε ότι το μέγεθος της σελίδας είναι γνωστό (π.χ. 4KB) και σταθερό. Σε περίπτωση που το μέγεθος του αρχείου δεν είναι ακέραιο πολλαπλάσιο του μεγέθους της σελίδας (διόλου απίθανο σενάριο), θεωρούμε ότι στο υπολειπόμενο περιεχόμενο της τελευταίας σελίδας υπάρχουν μηδενικά bytes (zero padding).

Η διαδικασία παραγωγής ενός δέντρου Merkle ξεκινά με την είσοδο των σελίδων του αρχείου στην κρυπτογραφική συνάρτηση κατακερματισμού της επιλογής μας. Αυτό έχει ως αποτέλεσμα να παραχθεί μία σύνοψη για κάθε σελίδα του αρχείου. Στο παράδειγμα του παραπάνω σχήματος (Σχήμα 1), η είσοδος "σελίδα 1" στην συνάρτηση κατακερματισμού παράγει την έξοδο (σύνοψη-digest) "d_1" (ομοίως και για τις υπόλοιπες σελίδες του αρχείου). Συνεπώς, στο επίπεδο των φύλλων του δέντρου κατακερματισμού, έχουμε τις συνόψεις όλων των σελίδων του αρχείου. Η παραγωγή των συνόψεων των παραπάνω επιπέδων (τα οποία ονομάζονται επίπεδα μετά-δεδομένων ή metadata levels) γίνεται μέσω της εισαγωγής των συνόψεων των παρακάτω επιπέδων

στην συνάρτηση κατακερματισμού. Δηλαδή, αντί να χρησιμοποιούμε τις σελίδες του αρχείου σαν είσοδο στην συνάρτηση κατακερματισμού, χρησιμοποιούμε τις συνόψεις τους. Για παράδειγμα, η σύνοψη "d_{11}" παράγεται όταν δώσουμε σαν είσοδο στην συνάρτηση κατακερματισμού την παράθεση των συνόψεων "d_1" και "d_2". Η παράθεση (concatenation) δύο συμβολοσειρών (όπως είναι οι συνόψεις "d_1" και "d_2") x και y συμβολίζεται ως x||y ή xy και είναι ουσιαστικά μία νέα συμβολοσειρά η οποία προκύπτει από την x ακολουθούμενη από την y. Για παράδειγμα, έστω x η συμβολοσειρά "asdf" και y η συμβολοσειρά "qwerty", τότε x||y είναι η συμβολοσειρά "asdfqwerty". Επιστρέφοντας στο παράδειγμά μας, όπως παράγεται η "d_{11}", με αντίστοιχο τρόπο, παράγεται και η "d_{12}", δηλαδή, δίνωντας σαν είσοδο στην συνάρτηση κατακερματισμού την συμβολοσειρά "$d_3||d_4$". Η διαδικασία αυτή επαναλαμβάνεται μέχρι την ρίζα του δέντρου, η σύνοψη της οποίας είναι η "d_r" (root digest).

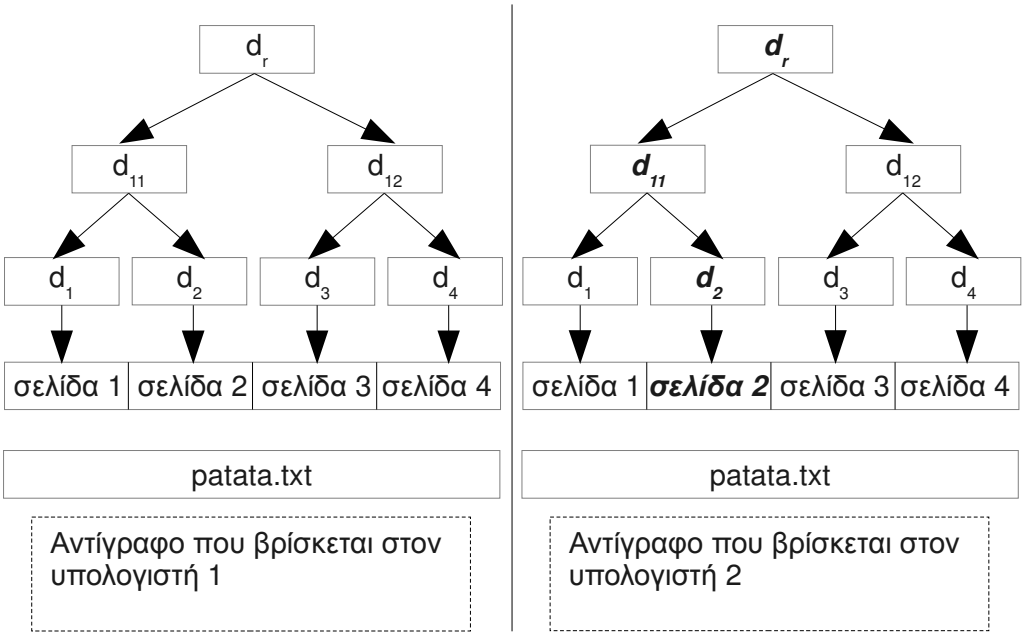
Στο παραπάνω παράδειγμα υποθέσαμε ότι ο παράγοντας διακλάδωσης του δέντρου (fan-out) είναι ίσος με 2, δηλαδή το δέντρο είναι δυαδικό (κάθε κόμβος έχει το πολύ δύο κόμβους παιδιά). Όμως, αυτό δεν σημαίνει ότι δεν μπορούμε να κατασκευάσουμε δέντρα κατακερματισμού με μεγαλύτερους παράγοντες διακλάδωσης.

Τα δέντρα κατακερματισμού χρησιμοποιούνται, κατά κόρον, όταν θέλουμε να συγκρίνουμε αποδοτικά, και στην πιο απλή περίπτωση, τα περιεχόμενα δύο αντιγράφων του ίδιου αρχείου. Ενδεχομένως σε αυτό το σημείο να σας δημιουργείτε η εξής απορία: Πως γίνεται δύο αντίγραφα να διαφέρουν; Για λόγους που δεν θα αναφέρουμε, μιας και είναι εκτός του πεδίου αυτής της εργασίας, δεχτείτε ότι κάτι τέτοιο μπορεί να συμβεί. Όταν λέμε ότι θέλουμε να συγκρίνουμε αποδοτικά τα δύο αρχεία, εννοούμε ότι, ιδεατά τουλάχιστον, θέλουμε να επικεντρωθούμε μόνο στα σημεία στα οποία διαφέρουν, όχι σε ολόκληρο το περιεχόμενό τους.

Τα δέντρα κατακερματισμού μας προσφέρουν μία πολύ καλή προσέγγιση του παραπάνω ιδεατού σεναρίου. Ας υποθέσουμε λοιπόν πως έχουμε δύο αντίγραφα του ίδιου αρχείου. Όμως, για κάποιο μυστήριο λόγο (δεν μας ενδιαφέρει), τα περιεχόμενα μίας εκ των σελίδων τους είναι διαφορετικά. Το σενάριο αυτό αναπαρίσταται στο Σχήμα 2, όπου η σελίδα της οποίας τα περιεχόμενα διαφέρουν ανάμεσα στις δύο εκδόσεις είναι η "σελίδα 2".

Αφού τα περιεχόμενα της δεύτερης σελίδας διαφέρουν ανάμεσα στις δύο εκδόσεις του αρχείου τότε, λόγω της $2^{ης}$ ιδιότητας των συναρτήσεων κατακερματισμού, όταν αυτές δοθούν σαν είσοδο στην συνάρτηση κατακερματισμού θα παράξουν διαφορετικές συνόψεις ("d_2" και "$\boldsymbol{d_2}$" αντιστοίχως). Όμως, για να υπολογιστεί η σύνοψη του πρώτου κόμβου του δεύτερου επιπέδου μετά-δεδομένων (το "d_{11}" δηλαδή) χρησιμοποιούνται οι συνόψεις "d_1" και "d_2". Αφού όμως η σύνοψη του δεύτερου κόμβου φύλλου είναι διαφορετική, τότε αυτό σημαίνει ότι και σύνοψη "d_{11}" θα είναι διαφορετική σε κάθε περίπτωση ("d_{11}" στην πρώτη και "$\boldsymbol{d_{11}}$" στην δεύτερη). Ομοίως, το ίδιο συμβαίνει και στην σύνοψη του κόμβου ρίζα. Ο σκοπός αυτής της περιγραφής είναι να αναδείξει την εξής ιδιότητα που έχουν τα δέντρα κατακερματισμού: ακόμα και η πιο μικρή αλλαγή

στα υπό οργάνωση δεδομένα θα διαδοθεί μέχρι και την ρίζα του δέντρου. Αυτό σημαίνει ότι η σύνοψη του κόμβου ρίζα αρκεί, με μεγάλη πιθανότητα και δεδομένων των υποθέσεων που έχουμε κάνει για τις συναρτήσεις κατακερματισμού, για να εξακριβωθεί αν το περιεχόμενο του αρχείου έχει αλλοιωθεί ή όχι.



Σχήμα 2: Οργάνωση δύο αντιτύπων του ίδιου αρχείου τα οποία διαφέρουν στα περιεχόμενα μίας εκ των σελίδων τους.

Σημειώνουμε πως η ιδιότητα αυτή ισχύει και για κάθε υπό-δέντρο ενός δέντρου Merkle. Δηλαδή, η ρίζα κάθε δέντρου (ή υπό-δέντρου) Merkle πάντα καθορίζει μοναδικά (uniquely identifies) το υπό οργάνωση περιεχόμενο.

Αναλυτική περιγραφή

Αναπτύξτε την εφαρμογή filesync, η οποία συγκρίνει δύο αρχεία και τυπώνει στην κονσόλα τις σελίδες (pages) στις οποίες διαφέρουν, χρησιμοποιώντας δέντρα κατακερματισμού (hash trees) υποστηριζόμενα από κρυπτογραφικές συναρτήσεις κατακερματισμού (cryptographic hash functions)

Η σύνταξη της γραμμής εντολής είναι:

filesync <file 1 name> <file 2 name> <page-size> <tree fan-out>

Όπου:

file 1 name	Το όνομα του πρώτου αρχείου
file 2 name	Το όνομα του δεύτερου αρχείου
page-size	Το μέγεθος της σελίδας
tree fan-out	Το πλήθος των κόμβων-παιδιών στο δέντρο

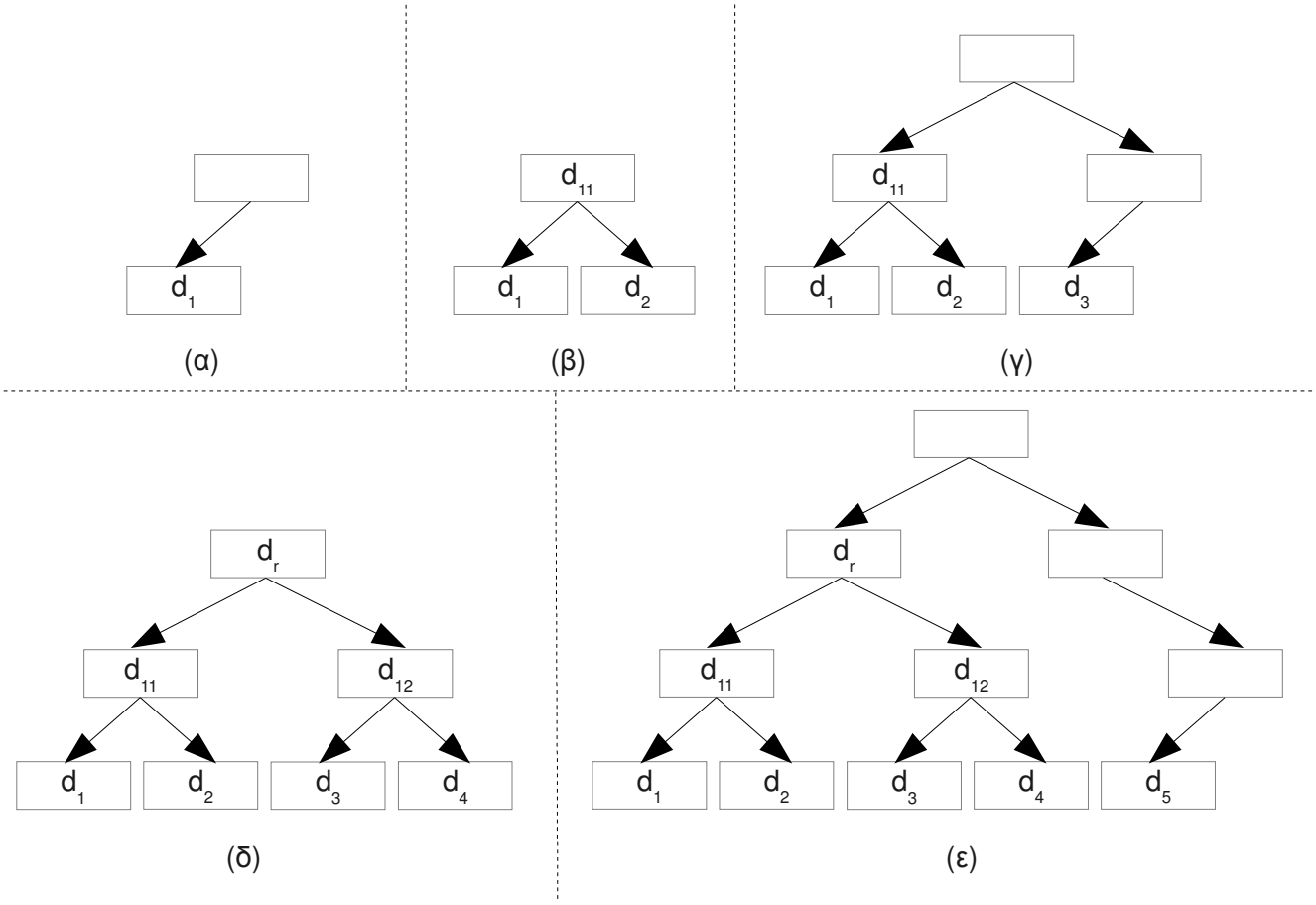
Η εφαρμογή θα διαβάσει το κάθε αρχείο σελίδα-σελίδα (σύμφωνα με το <page-size>) και θα χτίσει δυναμικά ένα ισοσταθμισμένο (balanced) δέντρο κατακερματισμού με σταθερό αριθμό παιδιών ίσο με <tree fan-out>. Τα φύλλα αυτού του δέντρου καταγράφουν (έχουν ετικέτα, label) την σύνοψη της αντίστοιχης σελίδας του αρχείου, ενώ οι εσωτερικοί κόμβοι έχουν σαν ετικέτα την σύνοψη των συνόψεων των κόμβων-παιδιών τους.
Στη συνέχεια θα συγκρίνει τα δύο δέντρα και θα τυπώσει στην κονσόλα είτε μήνυμα ισότητας (αν αυτά είναι ίσα) είτε τον αριθμό της κάθε σελίδας στην οποία τα δύο αρχεία διαφέρουν.
Για την ανάγνωση των αρχείων θα χρειαστείτε τις συναρτήσεις fopen/fread/fclose. Περισσότερες πληροφορίες για αυτές μπορείτε να βρείτε στα man-pages του συστήματος (π.χ. man fread) αλλά και στο πρόγραμμα-παράδειγμα που παρατίθεται στο τέλος.

Περιγραφή αλγόριθμου δημιουργίας δέντρου

Τα δέντρα που θα διατηρεί η εφαρμογή filesync για κάθε ένα από τα αρχεία που δέχεται σαν είσοδο θα κατασκευάζονται εντελώς δυναμικά. Το πρόγραμμα σας θα διαβάζει τα περιεχόμενα κάθε αρχείου σελίδα-σελίδα και για κάθε μία ανάγνωση θα πραγματοποιεί μία πράξη εισαγωγής στο δέντρο. Με αυτόν τον τρόπο, και ενώ διαβάζουμε τις σελίδες του αρχείου από τον δίσκο, το μέγεθος του δέντρου θα μεγαλώνει σταδιακά. **Προσοχή,** οποιαδήποτε παραδοχή "στατικού" χαρακτήρα αποκλίνει από το ζητούμενο της άσκησης. Για παράδειγμα, το μέγεθος κάθε αρχείου είναι γνωστό εκ των προτέρων (άρα ξέρουμε πόσες σελίδες έχει κάθε αρχείο), συνεπώς, είναι πολύ εύκολο να προκατασκευάσουμε το δέντρο, δηλαδή να δεσμεύσουμε μνήμη για τους κόμβους που θα έχει κάθε επίπεδο εξ αρχής και έπειτα να ακολουθήσει ο υπολογισμός των συνόψεων των κόμβων του. **Τέτοιου είδους πρακτικές δεν θα γίνουν αποδεκτές (θα μηδενιστούν) στα πλαίσια αυτής της εργασίας.**
Το υπόλοιπο αυτής της παραγράφου είναι αφιερωμένο στην περιγραφή ενός δυναμικού αλγορίθμου δημιουργίας ενός δέντρου κατακερματισμού. Ευθύς εξ αρχής διευκρινίζουμε ότι ο παρακάτω αλγόριθμος αποτελεί μόνο μία πρόταση εκ μέρους μας αναφορικά με τον τρόπο που μπορεί ένα τέτοιο δέντρο να μεγαλώνει δυναμικά. Σε καμία περίπτωση δεν είστε υποχρεωμένοι να τον ακολουθήσετε κατά γράμμα, μπορείτε να εισάγετε, και μάλιστα σας προτρέπουμε, τις όποιες δικές σας παραλλαγές ή βελτιστοποιήσεις, αρκεί να μην θίγεται η δυναμική φύση του δέντρου.

Για την περιγραφή του αλγορίθμου θα θεωρήσουμε ότι ο παράγοντας διακλάδωσης του δέντρου έχει την τιμή 2. Το Σχήμα 3 παρέχει μία οπτικοποίηση της διαδικασίας που θα περιγράψουμε. Αρχικά, δημιουργούμε ένα δέντρο το οποίο είναι απολύτως κενό, δηλαδή, δεν περιέχει απολύτως κανένα κόμβο. Στην συνέχεια, εισερχόμαστε σε έναν

βρόγχο στον οποίο διαβάζουμε όλες τις σελίδες του αρχείου από τον δίσκο. Θα περιγράψουμε την διαδικασία εισαγωγής 5 σελίδων. Διαβάζουμε την πρώτη σελίδα και την δίνουμε σαν είσοδο στην συνάρτηση κατακερματισμού. Λαμβάνουμε την σύνοψή της και την δίνουμε σαν όρισμα στην συνάρτηση εισαγωγής του δέντρου κατακερματισμού. Δημιουργούμε έναν νέο κόμβο φύλλο ο οποίος θα φιλοξενεί την σύνοψη της σελίδας που μόλις διαβάσαμε. Όμως, το δέντρο χρειάζεται και έναν κόμβο ρίζας, τον οποίο δεν τον έχει. Για το λόγο αυτό, λοιπόν, δημιουργούμε έναν νέο κόμβο, ο οποίος όμως δεν έχει περιεχόμενο. Αυτός ο νέος κόμβος πρέπει να δείξει στο φύλλο το οποίο μόλις πριν δημιουργήσαμε. Το αποτέλεσμα αυτής της διαδικασίας φαίνεται στο Σχήμα 3 (α). Όταν ένας κόμβος έχει κενό περιεχόμενο εννοούμε ότι δεν έχει υπολογιστεί (ακόμα) η σύνοψή του.



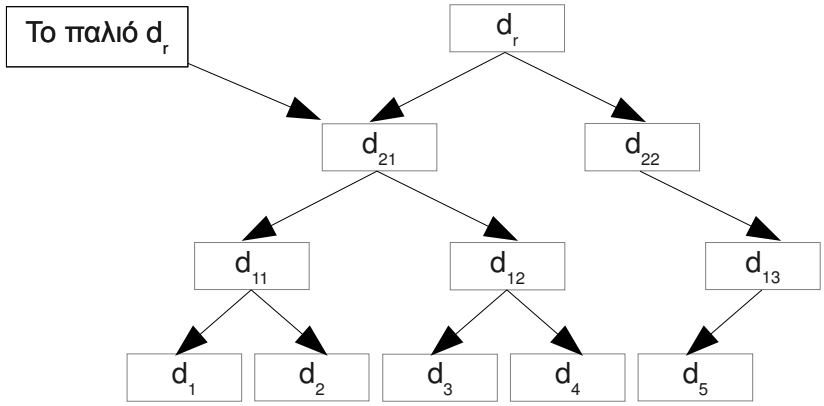
Σχήμα 3: Παράδειγμα δυναμικής επέκτασης δέντρου κατακερματισμού

Στην συνέχεια, διαβάζουμε την δεύτερη σελίδα από το αρχείο, υπολογίζουμε την σύνοψή της και την δίνουμε σαν είσοδο στην συνάρτηση εισαγωγής. Τώρα, πρέπει να βαδίσουμε το δέντρο και να βρούμε το σημείο στο οποίο πρέπει να εισάγουμε την σύνοψη της δεύτερης σελίδας. Ξεκινάμε από τον κόμβο ρίζα (στον οποίο πάντα θεωρούμε πως έχουμε έναν δείκτη) και ο σκοπός μας είναι να εισάγουμε αυτήν την νέα

σύνοψη στο "κάτω-δεξιά" άκρο του δέντρου. Δηλαδή, είναι σαν να κάνουμε προσάρτηση σε ένα αρχείο, οι νέες συνόψεις μπαίνουν πάντα στο τέλος του επιπέδου των κόμβων φύλλων. Εξετάζουμε τους δείκτες του κόμβου ρίζα και παρατηρούμε τα εξής: 1) το επίπεδο που βρίσκεται αμέσως μετά είναι το επίπεδο των κόμβων φύλλων και 2) υπάρχει ένας κενός δείκτης ο οποίος μπορεί να φιλοξενήσει την νέα αυτή σύνοψη. Δημιουργούμε έναν νέο κόμβο φύλλο, θέτουμε την σύνοψή του και τον συνδέουμε με τον δεξί δείκτη του κόμβου του παραπάνω επιπέδου, του κόμβου ρίζα δηλαδή. Όμως, αυτή τη στιγμή, όλοι οι δείκτες του κόμβου ρίζα είναι κατειλημμένοι, συνεπώς μπορούμε να πάμε και να υπολογίσουμε την σύνοψή του. Ακολουθούμε την διαδικασία υπολογισμού που περιγράψαμε προηγουμένως της οποίας το αποτέλεσμα το θέτουμε σαν περιεχόμενο του κόμβου ρίζα. Η διαδικασία αυτή έχει ως αποτέλεσμα το Σχήμα 3 (β). Συνεχίζουμε με την εισαγωγή της τρίτης σύνοψης (η οποία προέκυψε από την ανάγνωση και τον κατακερματισμό της τρίτης σελίδας). Ξεκινάμε από τον κόμβο ρίζα και παρατηρούμε τα εξής: 1) το επίπεδο που βρίσκεται αμέσως μετά είναι το επίπεδο των κόμβων φύλλων, και 2) δεν υπάρχει κανένας κενός δείκτης. Συνεπώς, πρέπει να μεγαλώσουμε το δέντρο. Δημιουργούμε έναν νέο κόμβο φύλλο ο οποίος θα φιλοξενήσει την σύνοψη της τρίτης σελίδας και έναν επιπλέον εσωτερικό κόμβο ο οποίος θα δείχνει σε αυτόν τον νέο κόμβο φύλλο. Τα περιεχόμενα αυτού του εσωτερικού κόμβου είναι κενά. Τώρα όμως πρέπει να δημιουργήσουμε μία νέα ρίζα για το δέντρο. Δημιουργούμε έναν ακόμα κόμβο λοιπόν ο οποίος θα παίξει τον ρόλο της νέας ρίζας. Συνδέουμε τον αριστερό του δείκτη με την παλιά ρίζα και τον δεξί του δείκτη με τον εσωτερικό κόμβο που μόλις δημιουργήσαμε. Το αποτέλεσμα αυτής της διαδικασίας φαίνεται στο Σχήμα 3 (γ). Συνεχίζουμε με την σύνοψη της τέταρτης σελίδας. Με αφετηρία τον κόμβο ρίζα, ακολουθούμε το δεξί μονοπάτι (αφού είναι διαθέσιμο). Όταν μεταβούμε στο αμέσως επόμενο επίπεδο κάνουμε, ξανά, τις εξής παρατηρήσεις: 1) το επίπεδο που βρίσκεται αμέσως μετά είναι το επίπεδο των κόμβων φύλλων και 2) υπάρχει ένας κενός δείκτης ο οποίος μπορεί να φιλοξενήσει την νέα αυτή σύνοψη. Συνεπώς, όπως και πριν, δημιουργούμε έναν νέο κόμβο φύλλο, θέτουμε το περιεχόμενό του. Επειδή, όπως και πριν άλλωστε, ο πατρικός κόμβος έχει γεμίσει, είμαστε σε θέση πλέον να υπολογίσουμε την σύνοψή του. Όμως, σε αυτό το σημείο, τα παιδιά του κόμβου ρίζα έχουν τις συνόψεις τους υπολογισμένες, το οποίο σημαίνει ότι μπορούμε πλέον να υπολογίσουμε και την σύνοψη του κόμβου ρίζα (που μέχρι τώρα ήταν κενή). Το αποτέλεσμα αυτής της διαδικασίας φαίνεται στο Σχήμα 3 (δ). Η εισαγωγή της πέμπτης σύνοψης θα οδηγήσει πάλι σε διάσπαση του δέντρου αφού είναι πλήρες. Το αποτέλεσμά της φαίνεται στο Σχήμα 3 (ε). Παρακάτω παραθέτουμε μία βήμα προς βήμα περιγραφή του παραπάνω αλγορίθμου:

Είναι πιθανό το πλήθος των σελίδων του αρχείου να μην είναι ακέραιο πολλαπλάσιο του παράγοντα διακλάδωσης (κάτι που είναι διαφορετικό από το μέγεθος του αρχείου να είναι ακέραιο πολλαπλάσιο του μεγέθους της σελίδας που είχε αναφερθεί σε προηγούμενη παράγραφο). Συνεπώς, και αν υποθέσουμε για παράδειγμα ότι το μέγεθος

του αρχείου εισόδου είναι 5 σελίδες (και ο παράγοντας διακλάδωσης είναι 2), στο τέλος της ανάγνωσης του αρχείου θα καταλήξουμε με την κατάσταση που αναπαρίσταται στο Σχήμα 3 (ε). Δηλαδή, οι συνόψεις κάποιων κόμβων δεν θα έχουν υπολογιστεί, κάτι που πρέπει να αντιμετωπιστεί. Αυτό το αναλαμβάνει μία ειδική συνάρτηση η οποία θα πρέπει να κληθεί μετά το πέρας της ανάγνωσης του αρχείου εισόδου. Η συνάρτηση αυτή θα πρέπει να διατρέξει (προσοχή, αν υπάρχουν) τα σημεία του δέντρου στα οποία δεν έχει υπολογιστεί σύνοψη. Το αποτέλεσμα της εφαρμογής αυτής της συνάρτησης στο δέντρο που υπάρχει στο Σχήμα 3 (ε) φαίνεται στο Σχήμα 4.



Σχήμα 4: Τελική κατάσταση του δέντρου κατακερματισμού

Η τιμή της σύνοψης "d_{13}" προέκυψε από την εφαρμογή της κρυπτογραφικής συνάρτησης κατακερματισμού πάνω στην σύνοψη "d_5" (αντιστοίχως και η "d_{22}"). Γενικότερα, όταν ένας εσωτερικός κόμβος του δέντρου έχει λιγότερα παιδιά από αυτά που ορίζει ο παράγοντας διακλάδωσης, τότε η σύνοψή του υπολογίζεται από την παράθεση των συνόψεων των όσων κόμβων διευθυνσιοδοτεί.

Περιγραφή αλγόριθμου σύγκρισης

Ο αλγόριθμος απαιτεί από έναν δείκτη για κάθε δέντρο.
Αρχικά θέτει τον κάθε δείκτη να δείχνει στην ρίζα του αντίστοιχου δέντρου.
Αν είναι φύλλο,
    Συγκρίνει τις συνόψεις των τρεχόντων κόμβων.
    Αν διαφέρουν
        Τυπώνει τον αριθμό της σελίδας
Αλλιώς
    Συγκρίνει τις συνόψεις των τρεχόντων κόμβων.
    Αν είναι ίσες
        επιστρέφει.
    Αν διαφέρουν
        συνεχίζει αναδρομικά στο κάθε παιδί του τρέχοντα κόμβου.

Επισημάνσεις/Παραδοχές:

- Παρά το ότι το μέγεθος των αρχείων σας είναι γνωστό με το άνοιγμά τους, δεν θα κάνετε καμία παραδοχή για αυτό όσον αφορά το χτίσιμο του δέντρου. Το δέντρο δηλαδή θα χτιστεί δυναμικά (κόμβο-κόμβο) καθώς το πρόγραμμά σας θα διαβάζει με τη σειρά τις σελίδες του αρχείου.
- Τα αρχεία είναι ίδιου μεγέθους.
- Σας δίνεται η συνάρτηση κατακερματισμού md5 σε μορφή κώδικα για να την συμπεριλάβετε και ένα πρόγραμμα-παράδειγμα χρήσης παρακάτω.
- Σας δίνεται παρακάτω ένα πρόγραμμα-παράδειγμα χρήσης των συναρτήσεων για το άνοιγμα και διάβασμα αρχείων (fopen/fread/fclose).

Τι θα βαθμολογηθεί:

1. Η συμμόρφωση του κώδικά σας με τις προδιαγραφές της άσκησης.
2. Η οργάνωση και η αναγνωσιμότητα (μαζί με την ύπαρξη σχολίων) του κώδικα.
3. Η χρήση Makefile και η κομματιαστή σύμβολο-μετάφραση (separate compilation).
4. Η αναφορά που θα γράψετε και θα υποβάλετε μαζί με τον πηγαίο κώδικα σε μορφή PDF.

Άλλες σημαντικές παρατηρήσεις:

1. Οι εργασίες είναι **ατομικές.**
2. Όποιος υποβάλλει / δείχνει κώδικα που δεν έχει γραφτεί από την ίδια/τον ίδιο **μηδενίζεται** στο μάθημα.
3. Αν και αναμένεται να συζητήσετε με φίλους και συνεργάτες το πως θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, αντιγραφή κώδικα (οποιασδήποτε μορφής) είναι κάτι που **δεν επιτρέπεται** και δεν πρέπει να γίνει. Οποιοσδήποτε βρεθεί αναμεμειγμένος σε αντιγραφή κώδικα **απλά παίρνει μηδέν** στο μάθημα. Αυτό ισχύει για **όλους όσους εμπλέκονται**, ανεξάρτητα από το ποιος έδωσε/πήρε κλπ.
4. Το πρόγραμμά σας θα πρέπει να γραφτεί σε C (ή C++ **χωρίς όμως** STL extensions) και θα πρέπει να τρέχει σε Ubuntu-Linux ή Solaris, αλλιώς **δεν θα βαθμολογηθεί**.
5. Σε καμία περίπτωση τα MS-Windows **δεν είναι επιλογή** πλατφόρμας για την παρουσίαση αυτής της άσκησης.

Παράδειγμα ανάγνωσης αρχείου:

```
#include <stdio.h>  //for I/O functions
#include <string.h> //for memset
```

```
int main(int argc, char *argv[]) {
  FILE * fp;
  size_t read;
  char buffer[16];
  size_t buffsize = sizeof(buffer);
  unsigned int iPage = 0;

  fp = fopen(argv[1], "rb"); //open the file for binary input
  if( fp == NULL ) {         //if that failed...
    perror("Could not open file");
    return 1;
  }
  //loop through the file reading a page at a time
  do {
    read = fread(buffer, 1, sizeof(buffer), fp); //issue the read call
    if ( read > 0 ) {   //if return value is 0
      if ( read < buffsize ) { //if fewer bytes than requested were returned...
        //fill the remainder of the buffer with zeroes
        memset(buffer + read, 0, buffsize - read);
      }
      printf("read page %u with size %zu\n", iPage, read);
      iPage++;
    }
  } while (read == buffsize ); //end when a read returned fewer items
  if ( ferror(fp) )
    perror("An error occured during reading");

  fclose(fp); //close the file
  return 0;
}
```

Παράδειγμα χρήσης MD5:

```
#include <stdio.h>

#include <string.h>

#include "MD5.h"



//paradeigma paragogis MD5 hashes

//online applet at: http://www.md5.cz/



int main( void )

{

      MD5_CTX context;

      unsigned int message1_digest[4],message2_digest[4],concatenation_digest[4];
```

```
char message1[] = "asdf";
char message2[] = "qwerty";
//Ipologismos tis sinopsis (digest) tou minimatos 1
MD5Init( &context );
MD5Update( &context , message1 , strlen( message1 ) );
MD5Final( message1_digest , &context );
//Ektiposi tis sinopsis tou minimatos 1
printf("MD5(%s)=",message1);
MD5Print( message1_digest );
printf("\n");
//Ipologismos tis sinopsis (digest) tou minimatos 2
MD5Init( &context );
MD5Update( &context , message2 , strlen( message2 ) );
MD5Final( message2_digest , &context );
//Ektiposi tis sinopsis tou minimatos 1
printf("MD5(%s)=",message2);
MD5Print( message2_digest );
printf("\n");
//Ipologismos tis sinopsis (digest) tis parathesis ton
//sinopsewn ton minimatwn 1 kai 2, diladi:
//               MD5( message1_digest || message2_digest )
MD5Init( &context );
MD5Update( &context , (char *)message1_digest , 4 * sizeof( int ) );
MD5Update( &context , (char *)message2_digest , 4 * sizeof( int ) );
MD5Final( concatenation_digest , &context );
//Ektiposi tis sinopsis tis parathesis ton sinopsewn
printf("MD5(");
MD5Print( message1_digest );
```

```
      printf("||");

      MD5Print( message2_digest );

      printf(")=");

      MD5Print( concatenation_digest );

      printf("\n");

      return 0;

}
```