Κεφάλαιο 4

Παράλληλοι Αλγόριθμοι Κοινής Μνήμης

Στον παράλληλο υπολογισμό, όπως και στον ακολουθιακό, έχουν αναπτυχθεί γενικές τεχνικές σχεδιασμού αλγορίθμων, οι οποίες βρίσκουν εφαρμογή στην ανάπτυξη παράλληλων αλγορίθμων για μια ποικιλία προβλημάτων. Κάποιες από τις τεχνικές αυτές, είναι αποτέλεσμα προσαρμογής στον παράλληλο υπολογισμό τεχνικών, οι οποίες χρησιμοποιούνται στον ακολουθιακό υπολογισμό, όπως, για παράδειγμα, η τεχνική *"διαίρει και βασίλευε" (divide and conquer),* ενώ κάποιες άλλες είναι καθαρά, παράλληλες τεχνικές, με την έννοια ότι αναπτύχθηκαν προκειμένου να αντιμετωπισθούν ειδικά προβλήματα του παράλληλου υπολογισμού, τα οποία δεν εμφανίζονται κατά το σχεδιασμό ακολουθιακών αλγορίθμων. Ενδεικτικά αναφέρουμε το πρόβλημα, το οποίο είναι γνωστό με το όνομα *"διάσπαση της συμμετρίας" (symmetry breaking)*, του οποίου η λύση ανεξαρτητοποιεί κάποια τμήματα του συνολικού υπολογισμού, με σκοπό τα τμήματα αυτά να μπορούν να εκτελεσθούν παράλληλα.

Στην παράγραφο 4.1 του κεφαλαίου αυτού παρουσιάζονται βασικές τεχνικές σχεδιασμού παράλληλων αλγορίθμων, μέσω της εφαρμογής τους σε συγκεκριμένα προβλήματα. Στην παράγραφο 4.2 δίνονται παράλληλοι αλγόριθμοι ταξινόμησης (sorting), ενώ στην παράγραφο 4.3 παρουσιάζονται παράλληλοι αλγόριθμοι για προβλήματα γράφων.

4.1 Βασικές Τεχνικές Σχεδιασμού Παράλληλων Αλγορίθμων

Στην παράγραφο αυτή, παρουσιάζονται βασικές τεχνικές σχεδιασμού παράλληλων αλγορίθμων, μέσω της εφαρμογής τους σε συγκεκριμένα προβλήματα. Τα προβλήματα αυτά, όχι μόνον έχουν ενδιαφέρον να μελετηθούν ως ανεξάρτητα προβλήματα, αλλά, συχνά, εμφανίζονται και ως υποπροβλήματα σε πιο σύνθετους υπολογισμούς. Συγκεκριμένα, θα μελετηθεί *η τεχνική των ιζοσυγισμένων δένδρων (balanced trees)* και η εφαρμογή της στον παράλληλο υπολογισμό αθροισμάτων προθεμάτων (prefix sum computation), η τεχνική *του διπλασιασμού δεικτών (pointer doubling)* και η εφαρμογή της στην εύρεση της τάξης κάθε στοιχείου μιας λίστας (list ranking), η τεχνική της *διαδρομής του Euler (Euler tour technique)* και η εφαρμογή της στον υπολογισμό απλών συναρτήσεων σε δένδρα, και τέλος, η τεχνική της *διαμέρισης (partitioning)* και η εφαρμογή της στο πρόβλημα της *συγχώνευσης (merging)*.

4.1.1 Δομή Ισοζυγισμένου Δένδρου

Η δενδρική δομή αποτελεί μία βασική δομή στον παράλληλο υπολογισμό. Το δένδρο εμφανίζεται, όχι μόνον ως δομή δεδομένων σε διάφορους αλγόριθμους, αλλά και ως δομή του υπολογισμού κατά την εκτέλεση ενός αλγορίθμου, όπως και ως δομή λειτουργίας των επεξεργαστών στην είσοδο του προβλήματος, κατά τη διάρκεια της εκτέλεσης του αλγορίθμου. Για παράδειγμα, ο υπολογισμός του αλγορίθμου *SUM* του Κεφαλαίου 3, βασίζεται σε ένα ισοζυγισμένο δένδρο, του οποίου τα φύλλα αντιστοιχούν στους n αριθμούς του διανύσματος A, ενώ οι εσωτερικοί κόμβοι αντιπροσωπεύουν προσθέσεις μεταξύ αριθμών. Ο αλγόριθμος *SUM* αποτελεί ένα παράδειγμα μιας γενικής τεχνικής, σύμφωνα με την οποία κτίζεται ένα ισοζυγισμένο δένδρο στα στοιχεία εισόδου, το οποίο αποτελεί τη βασική δομή του αλγορίθμου. Το δένδρο αυτό διαπερνιέται μία ή περισσότερες φορές, από τα φύλλα προς τη ρίζα, ή/και αντίστροφα. Σε κάθε εσωτερικό κόμβο v του δένδρου, συνήθως αποθηκεύεται πληροφορία, η οποία έχει σχέση με τα στοιχεία εισόδου, τα οποία αντιστοιχούν στα φύλλα του υποδένδρου με ρίζα τον κόμβο v. Η τεχνική διασαφηνίζεται στη συνέχεια μέσω της εφαρμογής της στον αναδρομικό αλγόριθμο υπολογισμού των αθροισμάτων προθεμάτων των στοιχείων ενός συνόλων X.

4.1.1.1 Αναδρομικός Υπολογισμός Αθροισμάτων Προθεμάτων

Έστω $*$ μία δυαδική προσεταιριστική πράξη, η οποία ορίζεται σε ένα σύνολο X. Δοθέντος ενός διανύσματος $A[1..n]$, n στοιχείων, τα οποία ανήκουν στο σύνολο X, το πρόβλημα του *υπολογισμού των αθροισμάτων προθεμάτων των στοιχείων* του A, ζητεί να υπολογισθούν οι επόμενες n ποσότητες:

$$S[i] = A[1] * A[2] * \ldots * A[i], \quad i = 1, \ldots, n$$

Ένας προφανής ακολουθιακός αλγόριθμος για τον υπολογισμό των αθροισμάτων προθεμάτων είναι ο παρακάτω.

Αν *S[1]:=A[1],* τότε υπολόγισε το *S[i+1], i=1,…,n-1,* ως εξής: *S[i+1]:=S[i]*A[i+1].*

Ο αλγόριθμος αυτός, ο οποίος χρειάζεται *O(n)* χρόνο για να ολοκληρωθεί, είναι συμφυώς (inherently) ακολουθιακός, και επομένως πολύ δύσκολα μπορεί να μετατραπεί σε έναν γρήγορο και αποδοτικό παράλληλο αλγόριθμο.

Ο αλγόριθμος, ο οποίος ακολουθεί, είναι ένας παράλληλος αναδρομικός αλγόριθμος υπολογισμού προθεμάτων, ο οποίος βασίζεται στη δομή ενός ισοζυγισμένου δυαδικού δένδρου. Ο αλγόριθμος, όπως και όλοι οι αλγόριθμοι του κεφαλαίου αυτού, παρουσιάζονται σύμφωνα με το Ε-Χ πλαίσιο παρουσίασης παράλληλων αλγορίθμων.

Αλγόριθμος 4.1

RECURSIVE_PREFIX_SUM(A,n,*)

Είσοδος: Ένα διάνυσμα *A[1…n] n* στοιχείων, $n=2^k$.

Έξοδος: Τα αθροίσματα προθεμάτων $S[i] = A[1] * A[2] * \ldots * A[i], \quad i = 1, \ldots, n,$ των στοιχείων του *A.*

begin

1. **if** $(n = 1)$ **then**
   begin
   $S[1] := A[1]$
   exit
   end
2. **for** $1 \le i \le n/2$ **pardo**
   $B[i] := A[2i-1] * A[2i]$
3. Υπολόγισε με αναδρομικό τρόπο τα αθροίσματα προθεμάτων
   $Q[i] = B[1] * B[2] * \ldots * B[i], \quad i = 1, \ldots, n/2,$ των στοιχείων του *B*
4. **for** $1 \le i \le n/2$ **pardo**
   if $(i = 1)$ **then**
   $S[1] := A[1]$
   else if $(i \bmod 2 = 1)$ **then**
   $S[i] := Q[(i-1)/2] * A[i]$
   else /* ο *i* είναι άρτιος αριθμός */
   $S[i] := Q[i/2]$

end

Ο τρόπος, με τον οποίο ο αλγόριθμος εκτελεί τον υπολογισμό, βασίζεται σε ένα ισοζυγισμένο δυαδικό δένδρο, του οποίου τα φύλλα αντιστοιχούν στα στοιχεία του *A* . Αν το μέγεθος του διανύσματος *A* είναι $n=2^k$, τότε ο αλγόριθμος απαιτεί *2k+1* χρονικές μονάδες. Κατά την διάρκεια των πρώτων *k* χρονικών μονάδων, ο υπολογισμός κατευθύνεται από τα φύλλα προς τη ρίζα του δένδρου, ενώ κατά την διάρκεια των τελευταίων *k* χρονικών μονάδων, το δένδρο διαπερνιέται από τη ρίζα προς τα φύλλα, και ταυτόχρονα υπολογίζονται οι ποσότητες προθεμάτων, με τη βοήθεια δεδομένων τα οποία έχουν παραχθεί κατά την διάρκεια των *k* πρώτων χρονικών μονάδων.

Παράδειγμα 4.1 Στη συνέχεια θεωρούμε ότι, το μέγεθος του διανύσματος *A* είναι $n=2^3$, και εξηγούμε τον τρόπο εκτέλεσης του αλγορίθμου σε αυτήν την περίπτωση. Κατά τη διάρκεια της πρώτης χρονικής μονάδας, υπολογίζονται τα τέσσερα στοιχεία του διανύσματος *B*, δηλαδή οι λειτουργίες της πρώτης χρονικής μονάδας είναι οι εξής (βήμα 2 του αλγορίθμου):

Χρονική Μονάδα 1:

B[1]:=A[1] *A[2] B[2]:=A[3] *A[4] B[3]:=A[5]*A[6] B[4]=A[7]*A[8]

Κατά τη διάρκεια της δεύτερης και τρίτης χρονικής μονάδας, μέσω της αναδρομικής κλήσης του αλγορίθμου, εκτελούνται οι εξής λειτουργίες (βήμα 3 του αλγορίθμου):

Χρονική Μονάδα 2:

*B'[1]:=B[1] *B[2] B'[2]:=B[3] *B[4]*

Χρονική Μονάδα 3:

*B''[1]:=B'[1] *B'[2]*

Έτσι, κατά την διάρκεια της τέταρτης χρονικής μονάδας υπολογίζεται το άθροισμα προθεμάτων *Q''[1]* του διανύσματος *B''[1]* (βήμα 1 του αλγορίθμου):

Χρονική Μονάδα 4:

Q''[1]:=B''[1]

Κατά την αντίστροφη πορεία, και στις χρονικές μονάδες *5, 6* και *7,* υπολογίζονται τα αθροίσματα προθεμάτων των διανυσμάτων *B', B* και *A* αντίστοιχα (βήμα 4 του αλγορίθμου):

Χρονική Μονάδα 5:

*Q'[1]:=B'[1] Q'[2]:=B'[1]*B'[2]*

Χρονική Μονάδα 6:

*Q[1]:=B[1] Q[2]:=B[1]*B[2] Q[3]:=B[1]*B[2]*B[3] Q[4]:=B[1]*B[2]*B[3]*B[4]*

Χρονική Μονάδα 7:

*S[1]:=A[1] S[2]:=A[1]*A[2] S[3]:=A[1]*A[2]*A[3] S[4]:=A[1]*A[2]*A[3]*A[4]*
*S[5]:=A[1]*A[2]*A[3]*A[4]*A[5] S[6]:=A[1]*A[2]*A[3]*A[4]*A[5]*A[6]*
*S[6]:=A[1]*A[2]*A[3]*A[4]*A[5]*A[6]*A[7]*
*S[7]:=A[1]*A[2]*A[3]*A[4]*A[5]*A[6]*A[7] *A[8]*

■

Ο χρόνος, ο οποίος απαιτείται για την εκτέλεση του αλγορίθμου, καθώς επίσης και το έργο το οποίο εκτελείται σύμφωνα με το Ε-Χ πλαίσιο παρουσίασης παράλληλων αλγορίθμων, απορρέουν από το Λήμμα, το οποίο ακολουθεί.

Λήμμα 4.1 *Ο αλγόριθμος* ***RECURSIVE_PREFIX_SUM*** *υπολογίζει σωστά τα αθροίσματα προθεμάτων ενός διανύσματος A, n στοιχείων, σε χρόνο T(n)=O(logn)*[1]*, εκτελώντας συνολικά W(n)=O(n) λειτουργίες.*

Απόδειξη. Η ορθότητα του αλγορίθμου θα αποδειχθεί με επαγωγή στο *k*, όπου $2^k=n$ είναι το μέγεθος του διανύσματος *A*.

Η βάση της επαγωγής, *k=0* ή *n=1*, αντιμετωπίζεται σωστά από το βήμα 1 του αλγορίθμου.

Επαγωγική υπόθεση: Ο αλγόριθμος υπολογίζει σωστά τα αθροίσματα προθεμάτων διανυσμάτων με 2^k στοιχεία, όπου *k>0.*

Επαγωγικό βήμα: Θα δείξουμε ότι, ο αλγόριθμος υπολογίζει σωστά τα αθροίσματα προθεμάτων διανυσμάτων με $2^{k+1}=n$ στοιχεία. Από την επαγωγική υπόθεση, το διάνυσμα *Q*, το οποίο υπολογίζεται στο βήμα 3, αποτελείται από τα αθροίσματα προθεμάτων των στοιχείων του διανύσματος *B*, το οποίο υπολογίζεται στο βήμα 2. Συγκεκριμένα, *Q[i]=B[1]*B[2]*…*B[i]* για *i = 1, … ,n/2*, και επειδή *B[i] = A[2i-1]*A[2i] για i = 1, …,n/2,* έχουμε:

$$Q[i] = A[1]*A[2]*\ldots*A[2i-1]*A[2i], \quad i = 1, \ldots, n/2$$

Επομένως, *Q[i]=S[2i]* για *i=1, … ,n/2.* Έτσι, όταν ο *i* είναι άρτιος αριθμός στο σύνολο *{1,2,…,n},* ο αλγόριθμος υπολογίζει σωστά τις ποσότητες:

S[i]=Q[i/2], για *i=1, … ,n.*

Διαφορετικά, όταν ο *i* είναι περιττός αριθμός, έχουμε τις εξής περιπτώσεις:

(1) *i=1*, οπότε *S[1]=A[1].*

[1] Όλοι οι λογάριθμοι, οι οποίοι εμφανίζονται στο παρόν κεφάλαιο, είναι με βάση το *2,* εκτός αν δηλώνεται διαφορετικά.

(2) $i \neq 1$, οπότε αν $i=2j+1$, τότε $S[i]=S[2j+1]=S[2j]*A[2j+1]$ και συνεπώς, $S[i]=Q[(i-1)/2]*A[i]$.

Παρατηρούμε ότι, ο αλγόριθμος αντιμετωπίζει σωστά και τις δύο περιπτώσεις και συμπεραίνουμε ότι, αυτός υπολογίζει σωστά τα αθροίσματα προθεμάτων των στοιχείων του διανύσματος *A*.

Προκειμένου να υπολογίσουμε τις απαιτήσεις του αλγορίθμου σε χρόνο και αριθμό λειτουργιών, παρατηρούμε ότι, το βήμα 1 χρειάζεται σταθερό χρόνο και σταθερό αριθμό λειτουργιών, ενώ τα βήματα 2 και 4 απαιτούν σταθερό παράλληλο χρόνο και $O(n)$ λειτουργίες. Επειδή ο αλγόριθμος καλείται αναδρομικά στο βήμα 3 σε είσοδο μεγέθους $n/2$, ο συνολικός χρόνος $T(n)$ ο οποίος απαιτείται, και το έργο $W(n)$ το οποίο εκτελείται, δίνονται από τις ακόλουθες αναδρομικές σχέσεις:

$$T(n) = T(n/2) + c$$

$$W(n) = W(n/2) + c'n$$

όπου c και c' είναι σταθερές. Λύνοντας τις αναδρομικές αυτές σχέσεις έχουμε ότι, $T(n)=\log n$ και $W(n)=O(n)$, και συνεπώς, ο αλγόριθμος ***RECURSIVE_PREFIX_SUM*** είναι βέλτιστος. ■

4.1.1.2 Μη Αναδρομικός Υπολογισμός Αθροισμάτων Προθεμάτων

Στην παράγραφο αυτή δίνεται ένας απλός μη αναδρομικός παράλληλος αλγόριθμος υπολογισμού αθροισμάτων προθεμάτων, ο οποίος απαιτεί $O(\log n)$ χρόνο και εκτελεί $O(n \log n)$ λειτουργίες και επομένως, σε αντίθεση με τον αναδρομικό αλγόριθμο της προηγούμενης παραγράφου, δεν είναι βέλτιστος.

Δοθέντος ενός διανύσματος $A[1..n]$, n στοιχείων, ο αλγόριθμος εκτελεί $\log n$ επαναλήψεις (βήμα 2 του αλγορίθμου). Στην επανάληψη k, $1 \leq k \leq \log n$, υπολογίζονται τα αθροίσματα προθεμάτων $S[2^{k-1}+1]$ έως $S[2^k]$.

Αλγόριθμος 4.2

NON_RECURSIVE_PREFIX_COMPUTATION(A,n,*)

Είσοδος: Ένα διάνυσμα $A[1\ldots n]$ n στοιχείων, $n=2^k$.

Έξοδος: Τα αθροίσματα προθεμάτων $S[i] = A[1] * A[2] * \ldots * A[i]$, $i = 1, \ldots, n$, των στοιχείων του *A*.

begin
1. *S[1] := A[1]*
2. **for** *k = 1 to log n* **do**
   for $1 \leq i \leq n$ **pardo**
   jump = 2^{k-1}
   if $i \geq jump + 1$ **then**
   $A[i] = A[i - jump] * A[i]$
   if $jump < i \leq 2\ jump$ **then**
   S[i] := A[i]

end

Παράδειγμα 4.1 Στη συνέχεια διεκρινίζεται ο τρόπος εκτέλεσης του αλγορίθμου στην περίπτωση του διανύσματος *A[1..n], n=2*^{3} και όταν δυαδική προσεταιριστική πράξη είναι η πρόσθεση.

A[1]	*A[2]*	*A[3]*	*A[4]*	*A[5]*	*A[6]*	*A[7]*	*A[8]*
2	*4*	*3*	*2*	*1*	*3*	*4*	*2*

Μετά την ολοκλήρωση της πρώτης επανάληψης *(k=1)* του βήματος 2, τα περιεχόμενα των διανυσμάτων *A* και *S* διαμορφώνονται ως εξής:

jump=1

A[1]	*A[2]*	*A[3]*	*A[4]*	*A[5]*	*A[6]*	*A[7]*	*A[8]*
2	*6*	*7*	*5*	*3*	*4*	*7*	*6*

S[1]	*S[2]*
2	*6*

Μετά την ολοκλήρωση της δεύτερης επανάληψης *(k=2)* του βήματος 2, τα περιεχόμενα των διανυσμάτων *A* και *S* διαμορφώνονται ως εξής:

jump=2

A[1]	*A[2]*	*A[3]*	*A[4]*	*A[5]*	*A[6]*	*A[7]*	*A[8]*
2	*6*	*9*	*11*	*10*	*9*	*10*	*10*

S[1]	*S[2]*	*S[3]*	*S[4]*
2	*6*	*9*	*11*

Τέλος, μετά την ολοκλήρωση της τρίτης επανάληψης *(k=3)* του βήματος 2, τα περιεχόμενα των διανυσμάτων *A* και *S* διαμορφώνονται ως εξής:

jump=4

A[1]	*A[2]*	*A[3]*	*A[4]*	*A[5]*	*A[6]*	*A[7]*	*A[8]*
2	*6*	*9*	*11*	*12*	*15*	*19*	*21*

S[1]	*S[2]*	*S[3]*	*S[4]*	*S[5]*	*S[6]*	*S[7]*	*S[8]*
2	*6*	*9*	*11*	*12*	*15*	*19*	*21*

■

Λήμμα 4.2 *Ο αλγόριθμος* ***NON_RECURSIVE_PREFIX_SUM*** *υπολογίζει σωστά τα αθροίσματα προθεμάτων ενός διανύσματος Α, n στοιχείων, σε χρόνο T(n)=O(logn), εκτελώντας συνολικά W(n)=O(nlogn) λειτουργίες.*

Απόδειξη. Η απόδειξη της ορθότητας του αλγορίθμου είναι προφανής, αφού είναι εύκολο να παρατηρήσουμε ότι στην επανάληψη k, $1 \leq k \leq logn$, του βήματος 2 του αλγορίθμου, υπολογίζονται σωστά τα αθροίσματα προθεμάτων $S[2^{k-1}+1]$ έως $S[2^k]$.

Το βήμα 2 αποτελείται από *logn* επαναλήψεις και κάθε επανάληψη k, $1 \leq k \leq logn$, απαιτεί *σταθερό χρόνο.* Επίσης, κάθε επανάληψη k, $1 \leq k \leq logn$, απαιτεί $n-2^{k-1}$ λειτουργίες για την εκτέλεση της πρώτης εντολής if και 2^{k-1} λειτουργίες για την εκτέλεση της δεύτερης εντολής if. Αθροίζοντας το χρόνο και τον αριθμό των λειτουργιών όλων των επαναλήψεων, προκύπτει ότι ο αλγόριθμος ***NON_RECURSIVE_PREFIX_SUM*** απαιτεί συνολικά *O(logn)* χρόνο και εκτελεί *O(nlogn)* λειτουργίες. ■

4.1.2 Η Τεχνική του Διπλασιασμού Δεικτών

Μία από τις πλέον χρήσιμες τεχνικές σχεδιασμού παράλληλων αλγορίθμων είναι η τεχνική του *διπλασιασμού δεικτών (pointer doubling).* Η τεχνική εφαρμόζεται σε συνδεσμικές λίστες, αλλά και σε δενδρικές δομές. Έστω για παράδειγμα ότι, δοθείσης μιας συνδεσμικής λίστας *n* κόμβων, ή ενός δένδρου *n* κόμβων, ζητείται να αποκτήσει κάθε κόμβος της λίστας ή του δένδρου την ετικέτα του τε-

λευταίου κόμβου στη λίστα ή της ρίζας του δένδρου, αντίστοιχα. Η τεχνική του διπλασιασμού δεικτών χρησιμοποιείται για την επίλυση του προβλήματος, ως εξής: σε κάθε βήμα του υπολογισμού, κάθε κόμβος της λίστας (ή του δένδρου), παράλληλα, αντικαθιστά τον δείκτη του με τον δείκτη του επόμενου κόμβου στη λίστα (ή του πατέρα του στο δένδρο, αντίστοιχα). Είναι φανερό ότι σε $\lceil logn \rceil$ το πολύ βήματα, κάθε κόμβος της λίστας (ή του δένδρου) "δείχνει" στον ίδιο κόμβο, δηλαδή στον τελευταίο κόμβο της λίστας (ή στη ρίζα του δένδρου, αντίστοιχα).

Η τεχνική διασαφηνίζεται περισσότερο στη συνέχεια, μέσω της εφαρμογής της στον αλγόριθμο υπολογισμού της *τάξης (rank)* κάθε κόμβου *k* μιας συνδεσμικής λίστας *L (list ranking),* ήτοι της απόστασης του κόμβου *k* από τον τελευταίο κόμβο της *L*.

4.1.2.1 Εύρεση της Τάξης Στοιχείου Λίστας

Έστω *L* μια συνδεσμική λίστα *n* κόμβων, των οποίων οι θέσεις στην *L* δίνονται μέσω ενός διανύσματος *NEXT*. Για κάθε κόμβο *i,* $1 \le i \le n$, το στοιχείο *NEXT[i]* περιέχει έναν δείκτη στον κόμβο, ο οποίος ακολουθεί τον κόμβο *i* στη λίστα *L.* Υποθέτουμε ότι, αν *i* είναι ο τελευταίος κόμβος της *L,* τότε *NEXT[i]=0.* Ζητείται να προσδιορισθεί, για κάθε κόμβο *i,* $1 \le i \le n$, η τάξη του *i* στη λίστα (list ranking), ήτοι η απόσταση *DIST[i]* του κόμβου *i* από τον τελευταίο κόμβο της λίστας.

Ενώ είναι εύκολο να σχεδιάσουμε έναν ακολουθιακό αλγόριθμο, ο οποίος θα απαιτεί γραμμικό χρόνο *(O(n))* στο μέγεθος της λίστας *L*, ο σχεδιασμός ενός βέλτιστου παράλληλου αλγορίθμου είναι πολύ πιο σύνθετο αλλά και ενδιαφέρον πρόβλημα. Στη συνέχεια, δίνουμε έναν αλγόριθμο, ο οποίος δεν είναι βέλτιστος.

Αλγόριθμος 4.3

LIST_RANKING(L)

Είσοδος: Μία συνδεσμική λίστα *L,* $n=2^k$ στοιχείων, τέτοια ώστε: (1) η διαδοχή των κόμβων της περιέχεται σε ένα διάνυσμα *NEXT[1…n],* δηλαδή, για κάθε κόμβο *i*, το *NEXT[i]* περιέχει έναν δείκτη στον επόμενο κόμβο του *i* και (2) αν *i* είναι ο τελευταίος κόμβος της *L*, τότε *NEXT[i]=0.*

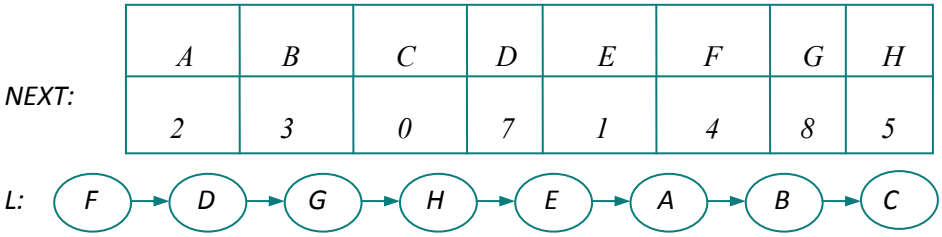
Έξοδος: Για κάθε κόμβο *i,* η απόσταση *DIST[i]* του κόμβου *i* από τον τελευταίο κόμβο της λίστας.

begin

1. **for** $1 \le i \le n$ **pardo**
   M[i] := NEXT[i]
   if *(M[i] = 0)* **then** *DIST[i] := 0*
   else *DIST[i] := 1*

```
2. for 1 ≤ h ≤ logn  do
        for 1 ≤ i ≤ n  pardo
            if  ( M[i] != 0 ) then
                begin
                    DIST[i] := DIST[i] + DIST[M[i]]
                    M[i] := M[M[i]]
                end
end
```

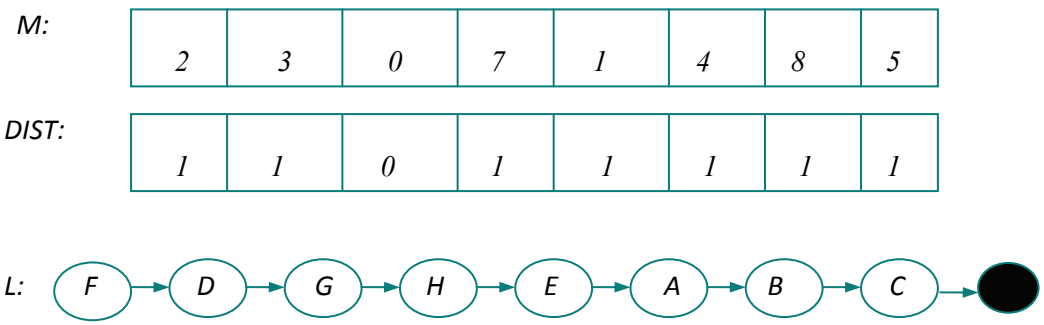
Παράδειγμα 4.3 Έστω ότι δίνεται η λίστα *L* του Σχήματος 4.1.



Σχήμα 4.1 *Συνδεσμική λίστα*

Στη συνέχεια, εξηγούμε τον τρόπο εκτέλεσης του αλγορίθμου με είσοδο τη λίστα του Σχήματος 4.1. Κατά την διάρκεια της πρώτης και της δεύτερης χρονικής μονάδας, ενημερώνονται τα διανύσματα *M* και *DIST* (βήμα 1 του αλγορίθμου). `Ετσι, τα περιεχόμενα των διανυσμάτων *M* και *DIST*, καθώς και η μορφή της λίστας *L*, εάν θεωρήσουμε ότι, η διαδοχή των κόμβων της δίνεται από το διάνυσμα *M[1…n],* μετά το τέλος των χρονικών μονάδων *1* και *2*, έχουν ως εξής:

Χρονική Μονάδα 1 και 2:



Κατά τη διάρκεια των *6* χρονικών μονάδων του βήματος 2 του αλγορίθμου, με τη βοήθεια της τεχνικής του διπλασιασμού δεικτών, ενημερώνονται τα διανύσματα *M* και *DIST* κατάλληλα, και μετά το τέλος της χρονικής μονάδας *8*, κάθε κόμβος *i* γνωρίζει την απόσταση του *DIST[i]* από τον τελευταίο κόμβο της λίστας. Τα περιεχόμενα των διανυσμάτων *M* και *DIST*, καθώς και η μορφή της λίστας *L*, εάν θεωρήσουμε ότι, η διαδοχή των κόμβων της δίνεται από το διάνυσμα *M[1…n],* μετά το τέλος των χρονικών μονάδων *3* και *4, 5* και *6, 7* και *8*, έχουν αντίστοιχα, ως εξής:

Χρονική Μονάδα 3 και 4:

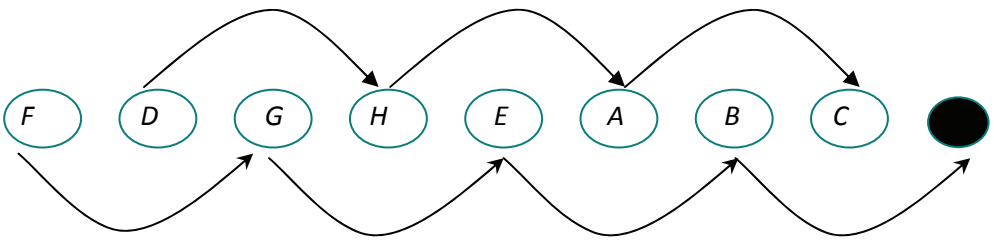
M:

3	0	0	8	2	7	5	1

DIST:

2	1	0	2	2	2	2	2

L:



Χρονική Μονάδα 5 και 6:

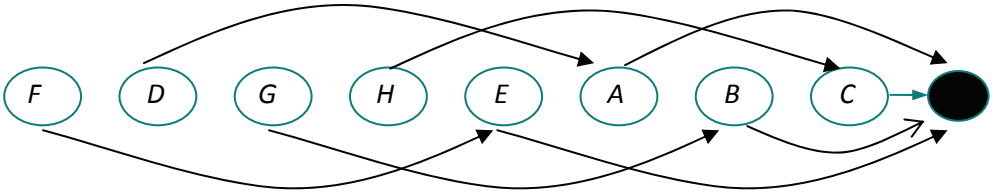
M:

0	0	0	1	0	5	2	3

DIST:

2	1	0	4	3	4	4	4

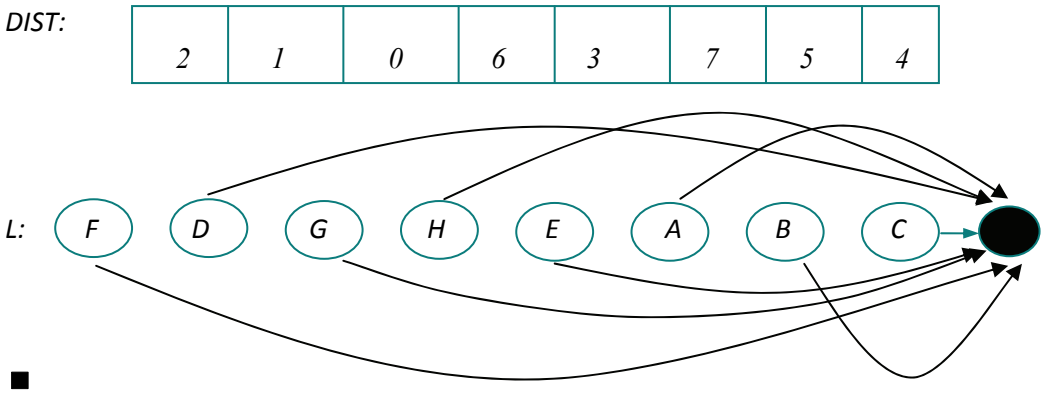
L:



Χρονική Μονάδα 7 και 8:

M:

0	0	0	0	0	0	0	0



■

Παρατηρούμε ότι, στο τέλος της επανάληψης *h* του βήματος 2 του αλγορίθμου 4.2, για κάθε *h=1,…, logn*, η εξής πρόταση αληθεύει:

Για κάθε i=1,…,n, όταν M[i]≠ 0, τότε το στοιχείο DIST[i] περιέχει την απόσταση του i από το M[i], ενώ όταν M[i]=0, τότε το DIST[i] περιέχει την απόσταση του i από το τέλος της λίστας.

Το γεγονός ότι, η προηγούμενη πρόταση ισχύει στο τέλος κάθε επανάληψης του βήματος 2 του αλγορίθμου, εξασφαλίζει το ότι στο τέλος της τελευταίας επανάληψης, για κάθε *i,* το στοιχείο *DIST[i]* θα περιέχει την απόσταση του *i* από το τέλος της λίστας, και επομένως, το ότι ο αλγόριθμος υπολογίζει σωστά την τάξη κάθε στοιχείου της λίστας.

Ο χρόνος, ο οποίος απαιτείται για την εκτέλεση του αλγορίθμου, είναι *O(logn).* Προκειμένου να υπολογίσουμε το έργο, το οποίο εκτελείται, σύμφωνα με το Ε-Χ πλαίσιο παρουσίασης παράλληλων αλγορίθμων, αρκεί να παρατηρήσουμε ότι, κατά τη διάρκεια των χρονικών μονάδων *1* και *2* (βήμα 1 του αλγορίθμου) εκτελούνται *O(n)* λειτουργίες, ενώ σε κάθε μία από τις *logn* επαναλήψεις του βήματος 2 εκτελούνται επίσης *O(n)* λειτουργίες. Επομένως, συνολικά εκτελούνται *O(n logn)* λειτουργίες. Συνεπώς, μπορούμε να συνοψίσουμε ότι το ακόλουθο Λήμμα ισχύει.

Λήμμα 4.3 *Ο αλγόριθμος* ***LIST_RANKING*** *υπολογίζει σωστά την τάξη κάθε στοιχείου μιας συνδεσμικής λίστας L, n στοιχείων, σε χρόνο T(n)=O(logn), εκτελώντας συνολικά W(n)=O(n logn) λειτουργίες.*

4.1.3 Η Τεχνική της Διαδρομής του Euler

Ένα από τα πρώτα βήματα στο σχεδιασμό παράλληλων αλγορίθμων για την επίλυση πολλών προβλημάτων σε γράφους, είναι η κατασκευή ενός *εκτεινόμενου δένδρου (spanning tree)* και ο υπολογισμός απλών συναρτήσεων σε αυτό το

δένδρο, όπως, για παράδειγμα, ο υπολογισμός της *προδιαταγμένης και μεταδιαταγμένης αρίθμησης (preorder and postorder numbering)* των κόμβων του δένδρου, του *επιπέδου (level)* κάθε κόμβου στο δένδρο, και του *αριθμού των απογόνων* κάθε κόμβου στο δένδρο. Ο παράλληλος υπολογισμός των συναρτήσεων αυτών μπορεί να γίνει με τη χρήση της *τεχνικής της διαδρομής του Euler (Euler tour technique)* σε δένδρα, την οποία θα μελετήσουμε στην παράγραφο αυτήν, και η οποία ανάγει τον υπολογισμό των συναρτήσεων στο πρόβλημα της εύρεσης της *τάξης των στοιχείων μιας λίστας (list ranking).*

Δοθέντος ενός δένδρου χωρίς ρίζα (unrooted tree) *T=(V,E),* ο κατευθυνόμενος γράφος *T'=(V,E')* ο οποίος προκύπτει, αν κάθε ακμή *{u,v}* του δένδρου *T* αντικατασταθεί από δύο κατευθυνόμενες ακμές *(u,v)* και *(v,u),* έχει την εξής ιδιότητα: ο έσω βαθμός (indegree) κάθε κόμβου ισούται με τον έξω βαθμό του (outdegree). Η ιδιότητα αυτή συνεπάγεται ότι, ο γράφος *T'* είναι γράφος *Euler* ήτοι ότι, υπάρχει κάποιο κατευθυνόμενο κύκλωμα (directed circuit), το οποίο

i. περνάει από όλους τους κόμβους του *T'*,

ii. περνάει από κάθε ακμή του *T'* ακριβώς μία φορά, και

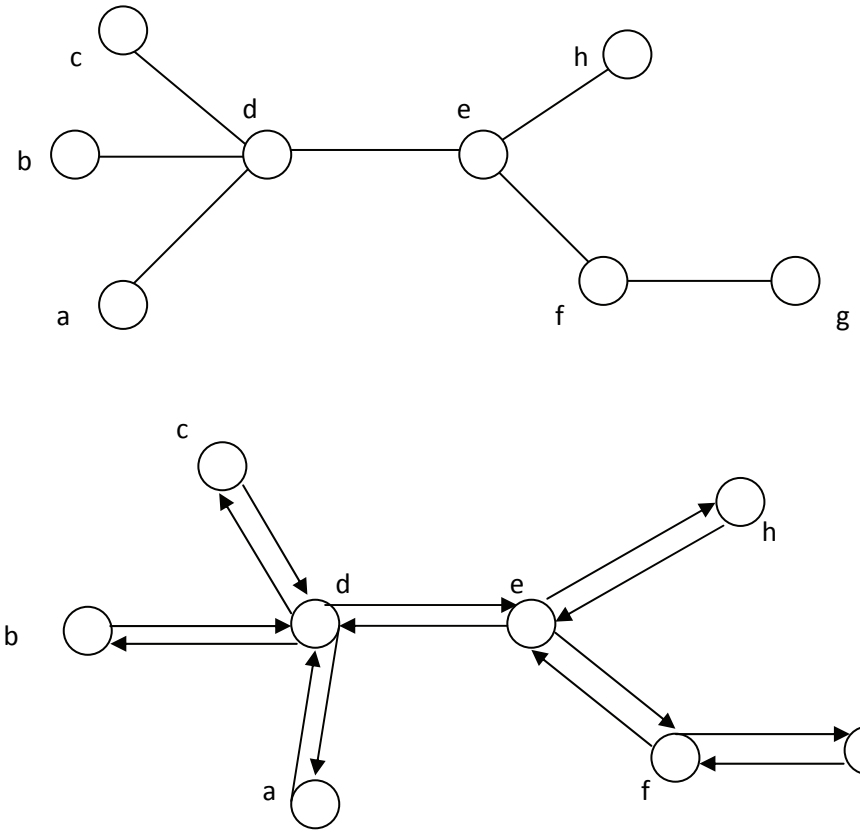
iii. καταλήγει στον κόμβο, από τον οποίο ξεκίνησε.

Το κύκλωμα αυτό, καλείται *κύκλωμα του Euler (Euler circuit)* του *T'*. Το κύκλωμα του Euler του κατευθυνόμενου γράφου *T'*, καλείται διαδρομή του Euler, του δένδρου *T*.

Στην παράγραφο αυτή θα δούμε πρώτα, έναν παράλληλο αλγόριθμο υπολογισμού της διαδρομής του Euler, και στη συνέχεια, θα δούμε το πώς η διαδρομή αυτή μπορεί να χρησιμοποιηθεί στο σχεδιασμό βέλτιστων παράλληλων αλγορίθμων για τον υπολογισμό διαφόρων συναρτήσεων σε δένδρα.

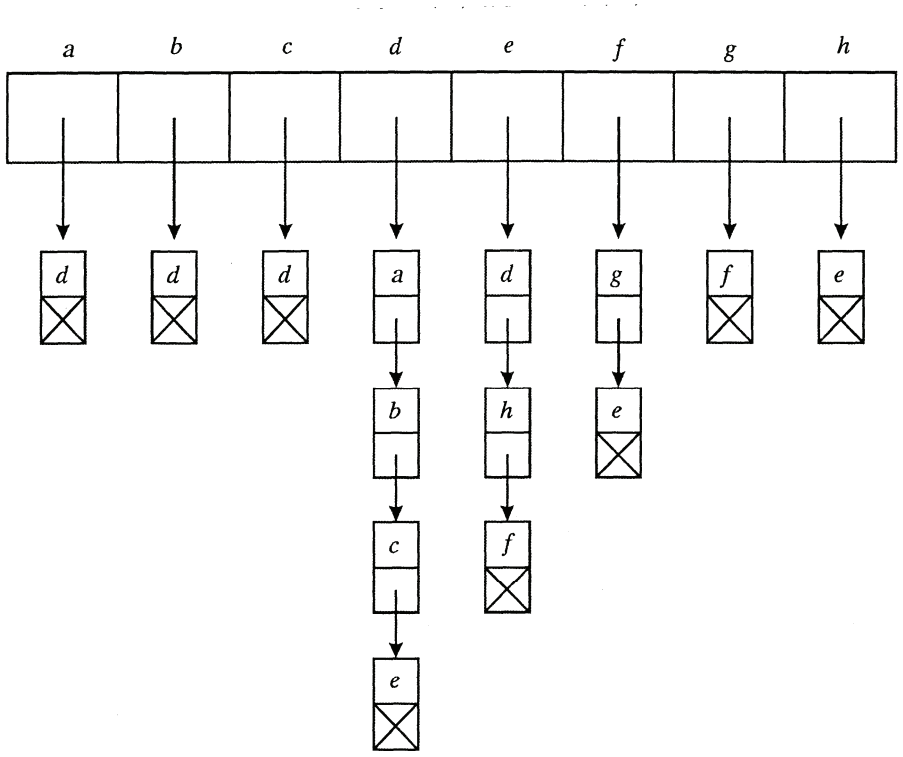
4.1.3.1 Υπολογισμός της Διαδρομής του Euler

Υποθέτουμε ότι, το δένδρο *T=(V,E)* αναπαρίσταται μέσω μιας λίστας γειτνίασης (adjacency list). Μπορούμε να υποθέσουμε ότι, η ίδια λίστα γειτνίασης αναπαριστά και τον κατευθυνόμενο γράφο *T'=(V,E'),* που προκύπτει, αν θεωρήσουμε κάθε ακμή $\{u,v\} \in E$ ως δύο κατευθυνόμενες ακμές *(u,v)* και *(v,u).*



Σχήμα 4.2 *Το δένδρο Τ και ο κατευθυνόμενος γράφος Τ′ του Παραδείγματος 4.3*

Για κάθε κατευθυνόμενη ακμή *e* του *T′,* προσδιορίζουμε την επομένη ή διάδοχο ακμή της *e* στο κύκλωμα του Euler, ως εξής. Πρώτα, για κάθε κόμβο *v*, καθορίζουμε μία συγκεκριμένη διάταξη των κόμβων, που πρόσκεινται στον *v.* Οποιαδήποτε διάταξη και αν επιλεγεί, θα οδηγήσει σε μία διαδρομή του Euler. Ωστόσο, διαφορετικές διατάξεις δίνουν διαφορετικές διαδρομές του Euler. Για ευκολία στην παρουσίαση, επιλέγουμε τη διάταξη, η οποία προκύπτει από τις λίστες γειτνίασης των κόμβων. Έτσι, αν *d* είναι ο βαθμός ενός κόμβου *u,* και $v_0,v_1,\ldots,v_{d-1}$ είναι οι γείτονες του *u,* με τη σειρά που εμφανίζονται στη λίστα γειτνίασης του *u*, τότε η επομένη ακμή της (v_i,u) στο κύκλωμα του Euler είναι η ακμή $(u,v_{(i+1) \bmod d})$, $i=0,1,\ldots,d-1$.



Σχήμα 4.3 *Η λίστα γειτνίασης του δένδρου T και του κατευθυνόμενου γράφου T'*

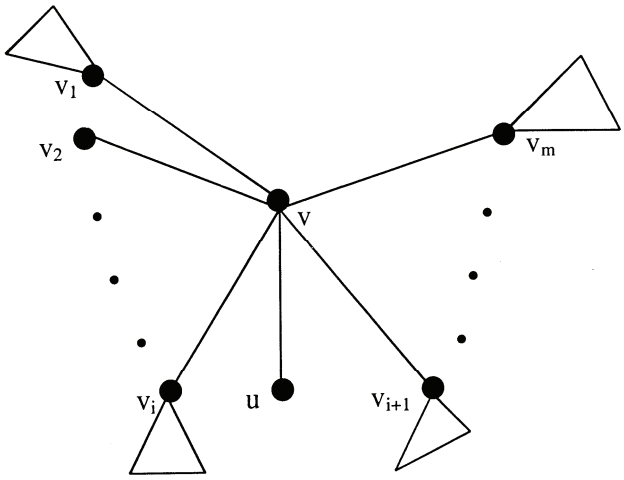
Παράδειγμα 4.4 Έστω ότι, δίνεται το δένδρο *T=(V,E)* του Σχήματος 4.2. Ο κατευθυνόμενος γράφος *T'=(V,E'),* ο οποίος προκύπτει από το δένδρο *T*, φαίνεται στο ίδιο σχήμα, ενώ η λίστα γειτνίασης του *T* (η οποία είναι ίδια με αυτήν του *T'*) δίνεται στο Σχήμα 4.3. Θα υπολογίσουμε τις επόμενες ακμές όλων των ακμών, που πρόσκεινται στον κόμβο e. Από τη λίστα γειτνίασης προκύπτει ότι, οι γείτονες κόμβοι του *e* είναι οι κόμβοι *d, h* και *f* με αυτήν τη συγκεκριμένη σειρά. Επομένως, έχουμε *u=e,* v_0*=d,* v_1*=h* και v_2*=f*. Η επόμενη ακμή της *(d,e)* είναι η *(e,h),* και η επόμενη ακμή της *(h,e)* είναι η *(e,f).* Συνεχίζοντας κατά τον ίδιο τρόπο και για τους άλλους κόμβους, βρίσκουμε ότι, οι επόμενες ακμές των ακμών του *T'* είναι:

Ακμή	Επόμενη Ακμή
(a,d)	*(d,b)*
(b,d)	*(d,c)*
(c,d)	*(d,e)*
(d,a)	*(a,d)*
(d,b)	*(b,d)*
(d,c)	*(c,d)*
(d,e)	*(e,h)*
(e,f)	*(f,g)*
(e,d)	*(d,a)*
(e,h)	*(h,e)*
(f,g)	*(g,f)*
(f,e)	*(e,d)*
(g,f)	*(f,e)*
(h,e)	*(e,f)*

Επομένως, η διαδρομή του Euler για το δένδρο *T* είναι η εξής: *(a,d),(d,b),(b,d), (d,c), (c,d), (d,e), (e,h), (h,e), (e,f), (f,g), (g,f), (f,e), (e,d), (d,a).* ■

Είναι εύκολο να παρατηρήσουμε ότι, η παραπάνω διαδικασία καθορίζει, για κάθε ακμή e στον κατευθυνόμενο γράφο *T'*, μία και μόνο διάδοχο ακμή *e'*. Επίσης, κάθε ακμή *e* του *T'* είναι διάδοχος μίας και μόνον άλλης ακμής *e'*. Εκείνο, το οποίο δεν είναι προφανές και θα πρέπει να δειχθεί, είναι ότι, από την παραπάνω διαδικασία προκύπτει ένα μόνον κύκλωμα, το οποίο περιέχει όλες τις ακμές του *T'*, και όχι ένα σύνολο από κυκλώματα, τα οποία δεν έχουν μεταξύ τους κοινές ακμές. Η απόδειξη της ιδιότητας αυτής γίνεται με επαγωγή στον αριθμό *n* των κόμβων του δένδρου *T* ως εξής: Όταν το δένδρο έχει δύο κόμβους, τότε η απόδειξη είναι προφανής. Θα δείξουμε ότι, η ιδιότητα ισχύει για ένα τυχαίο δένδρο *n* κόμβων, υποθέτοντας ότι, αυτή ισχύει για οποιοδήποτε δένδρο με λιγότερους από *n* κόμβους. Έστω T_1 ένα δένδρο με *n-1* κόμβους, στο οποίο εάν προσθέσουμε στον τυχαίο κόμβο *v* ένα φύλλο u, παίρνουμε ένα δένδρο T_2 με *n* κόμβους. Έστω

επίσης ότι, στον κόμβο *v* του T_1 πρόσκεινται *m* κόμβοι και εμφανίζονται στη λίστα γειτνιάσεων με τη σειρά $v_1, v_2, \ldots, v_m$ (Σχήμα 4.4).



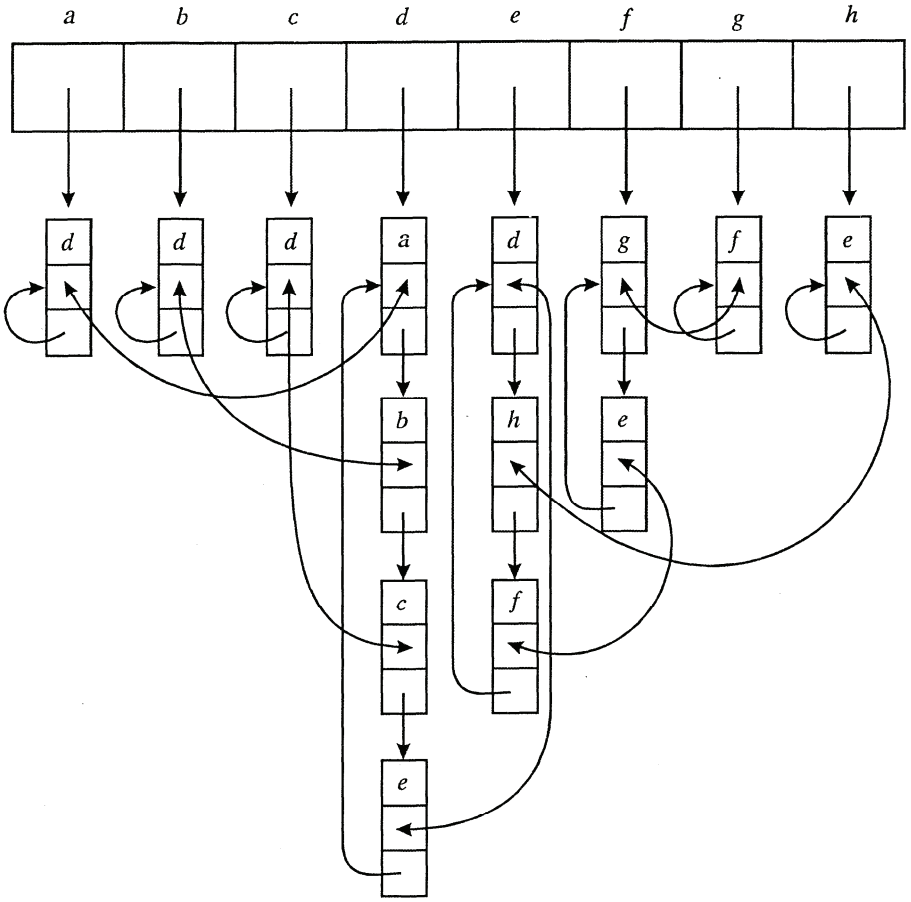
Σχήμα 4.4 *Ο κόμβος v του δένδρου T_1*

Τότε, από την επαγωγική υπόθεση, προκύπτει ότι, η εν λόγω διαδικασία καθορίζει στον κατευθυνόμενο γράφο T_1', ένα κύκλωμα του Euler, στο οποίο η επόμενη της ακμής (v_i,v) είναι η ακμή (v,v_{i+1}). Αν υποθέσουμε ότι, στη λίστα γειτνιάσεων του *v*, ο κόμβος *u* παρεμβάλλεται μεταξύ των κόμβων v_i και v_{i+1}, τότε, ορίζοντας την ακμή (v,u) ως την επόμενη της ακμής (v_i,v) και την ακμή (v,v_{i+1}) ως την επόμενη της ακμής (u,v), προκύπτει ένα και μόνον κύκλωμα το οποίο αποτελεί κύκλωμα του Euler στον κατευθυνόμενο γράφο T_2'. Έτσι, ολοκληρώνεται η επαγωγική απόδειξη.

Προκειμένου να σχεδιάσουμε έναν παράλληλο αλγόριθμο υπολογισμού της διαδρομής του Euler ενός δένδρου *T*, αλλάζουμε τον τρόπο αναπαράστασης του κατευθυνόμενου γράφου *T'*. Συγκεκριμένα, η νέα αναπαράσταση προκύπτει από την κανονική λίστα γειτνίασης ως εξής:

- Μετατρέπουμε τη μη κυκλική λίστα γειτνίασης κάθε κόμβου σε κυκλική.
- Για κάθε ακμή (v,u) του κατευθυνόμενου γράφου *T'*, από τον κόμβο ο οποίος αναπαριστά τον κόμβο *u* στη λίστα γειτνίασης του κόμβου *v*, προσθέτουμε ένα δείκτη στον κόμβο, ο οποίος αναπαριστά τον *v* στη λίστα γειτνίασης του *u*.

Στο Σχήμα 4.5 φαίνεται η τροποποιημένη λίστα γειτνίασης του κατευθυνόμενου γράφου *T'*, του Σχήματος 4.2.



Σχήμα 4.5 *Η τροποποιημένη λίστα γειτνίασης του δένδρου του Σχήματος 4.2*

Δοθείσης της προηγούμενης αναπαράστασης του *T'*, η επόμενη της ακμής *(v,u)* (η οποία αναπαρίσταται με την εμφάνιση του κόμβου *u* στη λίστα γειτνίασης του κόμβου *v*) υπολογίζεται σε *O(1)* χρόνο, χρησιμοποιώντας *O(1)* λειτουργίες, ως εξής. Ακολουθούμε το δείκτη εμφάνισης του *v* στη λίστα γειτνίασης του *u,* και κατόπιν πηγαίνουμε στον κόμβο, που ακολουθεί τον *v* στη λίστα γειτνίασης του *u*. Για παράδειγμα, προκειμένου να υπολογίσουμε την επόμενη της ακμής *(a,d)* στον κατευθυνόμενο γράφο του Παραδείγματος 4.3, ακολουθούμε τον δείκτη εμφάνισης του *a* στη λίστα γειτνίασης του *d,* και επειδή στη λίστα αυτήν, ο κόμβος που ακολουθεί τον κόμβο *a* είναι ο *b*, η ζητούμενη ακμή είναι η *(d,b).*

Επομένως, μπορούμε να ισχυρισθούμε ότι ισχύει το ακόλουθο Θεώρημα.