ΗΥ240 : Δομές Δεδομένων

Φροντιστήριο Προγραμματιστικής Εργασίας

2ο και 3ο Μέρος

Εισαγωγή

- Στο 2ο μέρος της εργασίας θα πρέπει να γίνουν τροποποιήσεις στο πρόγραμμα που προέκυψε κατά την υλοποίηση του 1ου μέρους.
- Το πρόγραμμα που θα προκύψει θα προσομοιώνει και πάλι τη λειτουργία ενός συστήματος δημοσιοποίησης/συνδρομής.

Διαφορές (1/3)

- Πλέον δεν υπάρχει ταξινομημένη διπλά συνδεδεμένη λίστα πληροφοριών!
  - Αντικαταστάται με ένα ταξινομημένο, διπλά συνδεδεμένο, δυαδικό δένδρο (binary search tree) - ΔΕΝΔΡΟ ΠΛΗΡΟΦΟΡΙΩΝ

Διαφορές (2/3)

- Εισαγωγή ενός καινούργιου γεγονότος
  - Γεγονός Prune

Διαφορές (3/3)

- Αντικαταστάται η λίστα Συνδομητών!
  - Την θέση της θα πάρει ένας:
    - πίνακας κατακερματισμού και
    - φυλλοπροσανατολισμένων δυαδικών δένδρων των οποίων τα φύλλα συνενώνονται με απλά συνδεδεμένη λίστα - ΔΕΝΔΡΑ ΚΑΤΑΝΑΛΩΣΗΣ

Struct Group

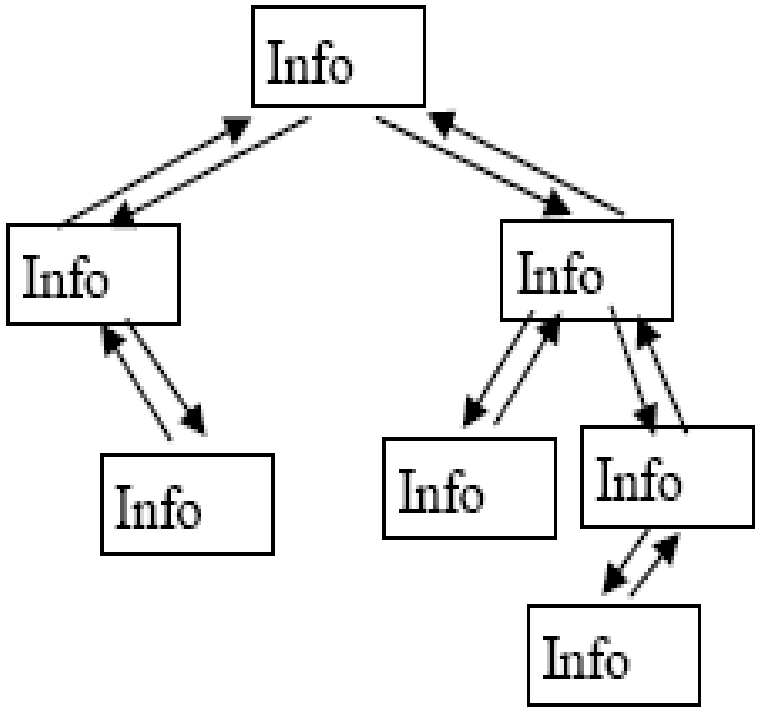
```
struct Group {                          // Εγγραφή πίνακα ομάδων
        int gId;                        // Αναγνωριστικό ομάδας
        struct Subscription *gsub;      // Δείκτης στο πρώτο στοιχείο λίστας συνδρομών
        struct Info *gr;                // Δείκτης στη ρίζα του δένδρου πληροφοριών
};
```

Struct Subscription (όπως στο 1ο μέρος της προγραμματιστικής άσκησης)

```
struct Subscription {                  // Εγγραφή λίστας συνδρομών
        int sId;                       // Αναγνωριστικό Συνδρομητή
        struct Subscription *snext;    // Επόμενη εγγραφή λίστας συνδρομών
};
```

Struct Info

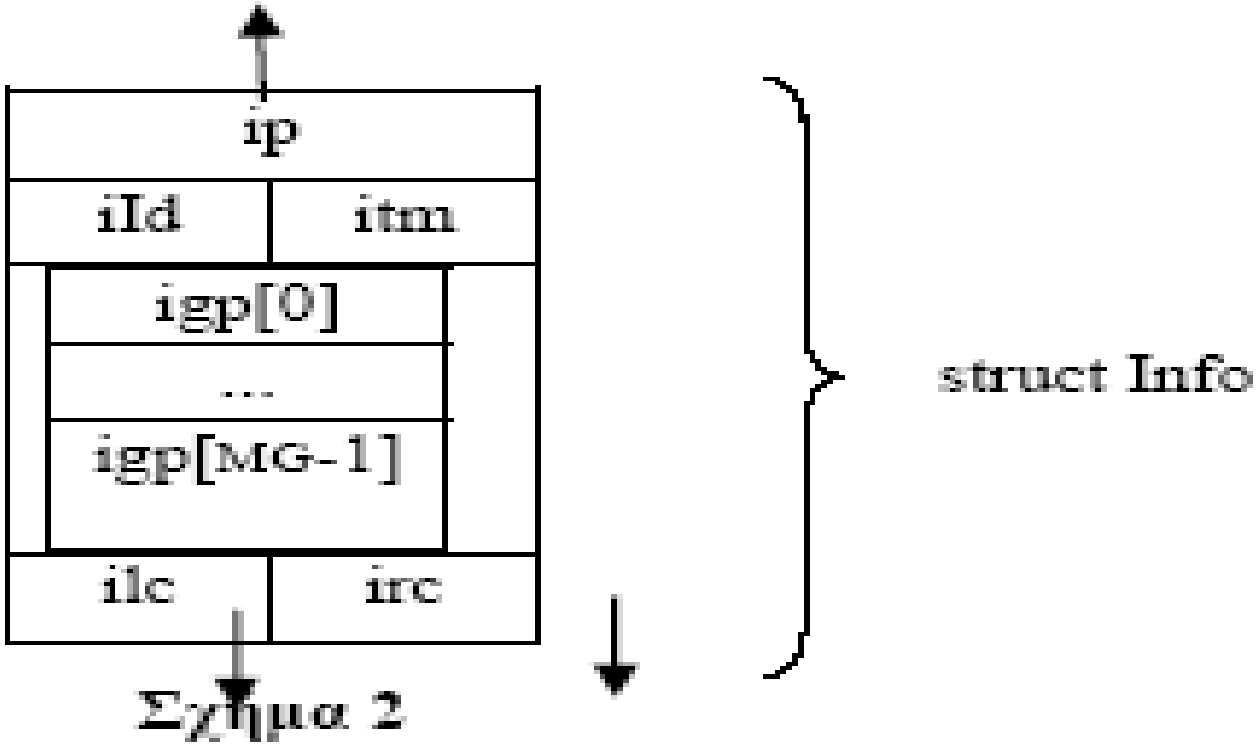
- Η ταξινομημένη διπλά συνδεδεμένη λίστα πληροφοριών κάθε ομάδας θα πρέπει να αντικατασταθεί με ένα **ταξινομημένο, διπλά-συνδεδεμένο δυαδικό δένδρο** (doubly-linked binary search tree).
- Το δένδρο πληροφοριών κάθε ομάδας είναι **ταξινομημένο ως προς το αναγνωριστικό** των εγγραφών που περιέχει.



Struct Info

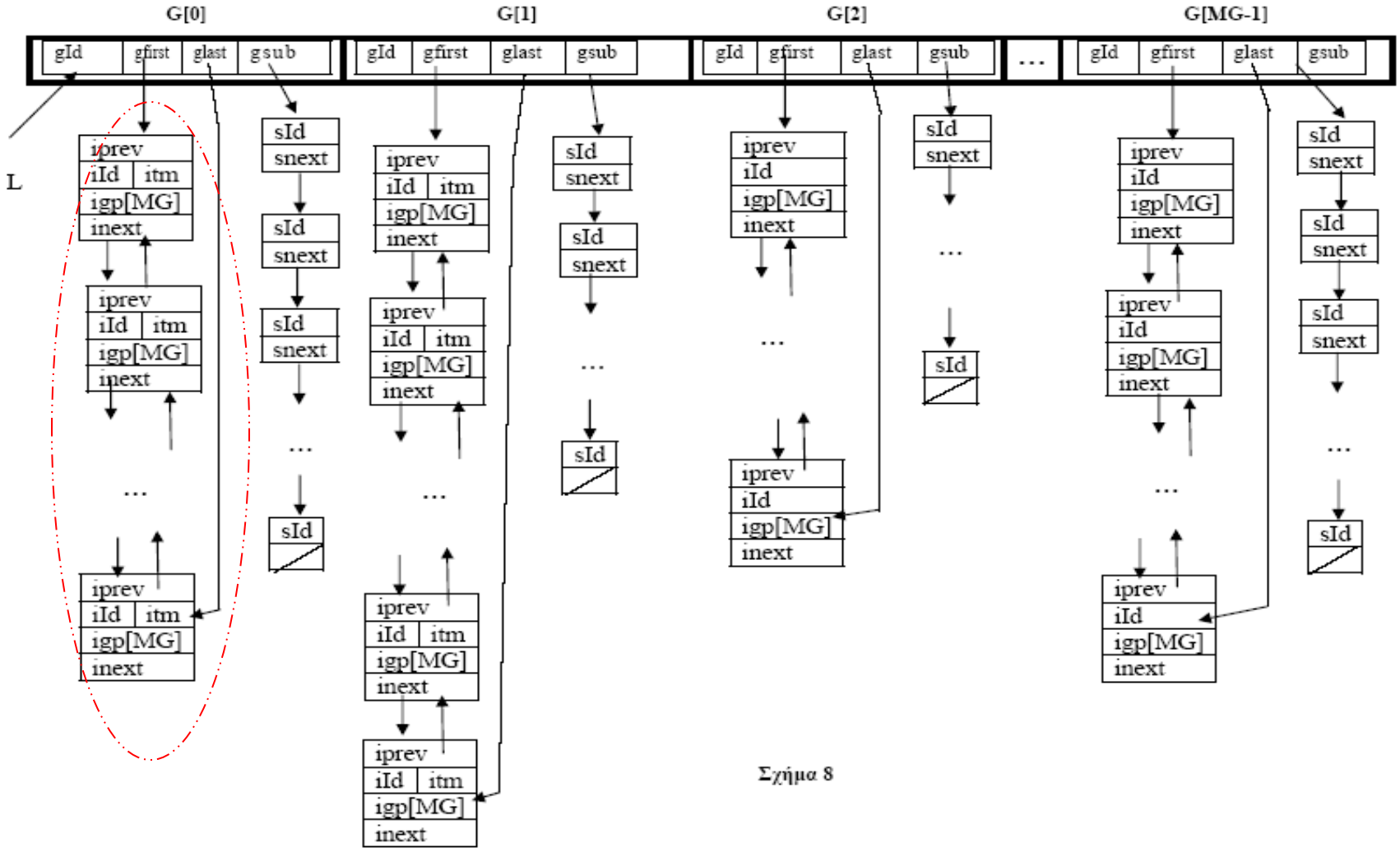
```
struct Info {                   // Εγγραφή δένδρου πληροφοριών
        int iId;                // Αναγνωριστικό πληροφορίας
        int itm;                // Χρονοσφραγίδα δημοσίευσης πληροφορίας
        int igp[MG];            // Αυτό πρέπει να είναι ο πίνακας igp MG θέσεων
        struct Info *ilc;       // Δείκτης στον αριστερό θυγατρικό κόμβο
        struct Info *irc;       // Δείκτης στο δεξιό θυγατρικό κόμβο
        struct Info *ip;        // Δείκτης στον πατρικό κόμβο
};
```

Struct Info



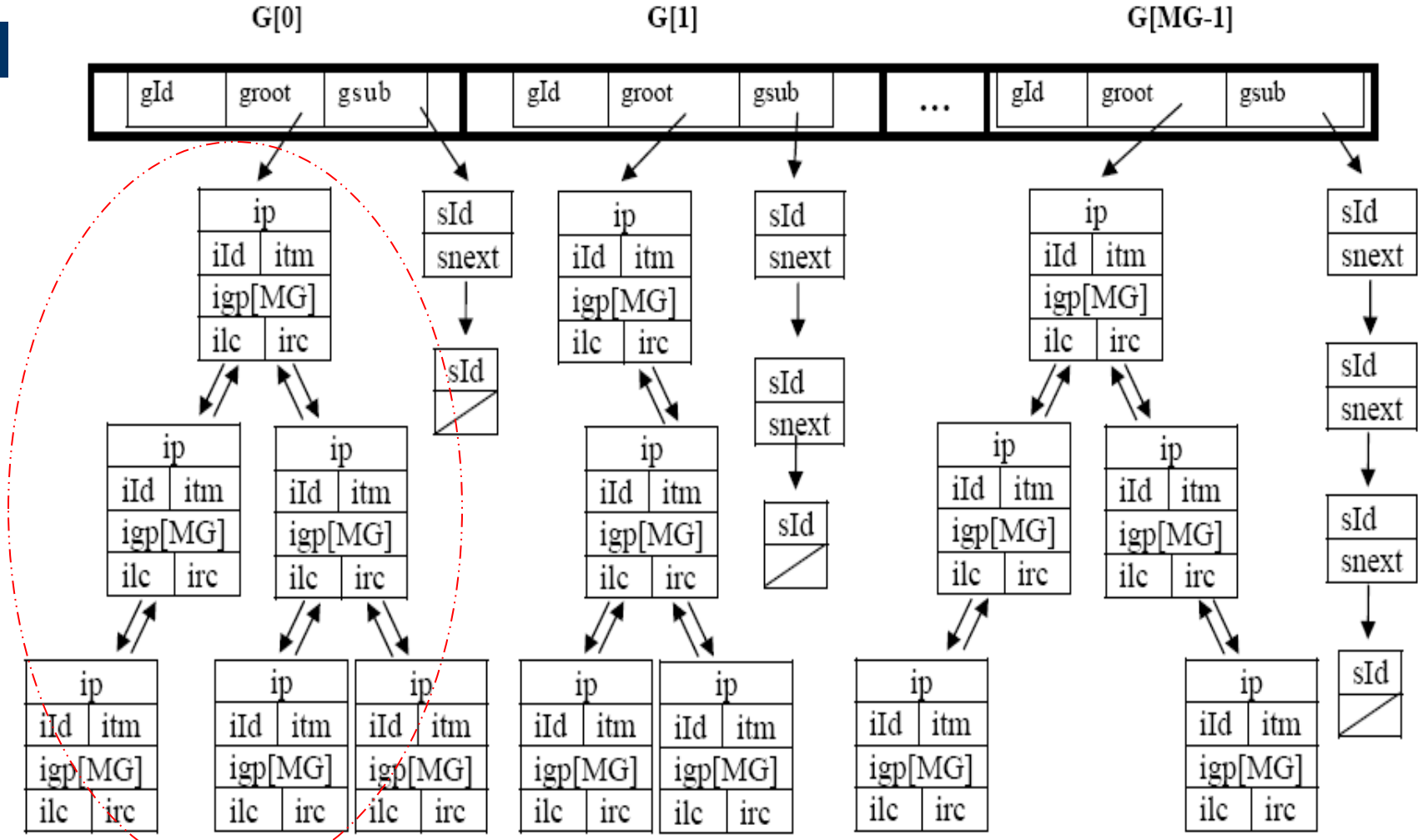
Σχήμα 2

1ο Μέρος Εργασίας (Struct Info)



Σχήμα 8

2ο Μέρος Εργασίας (Struct Info)



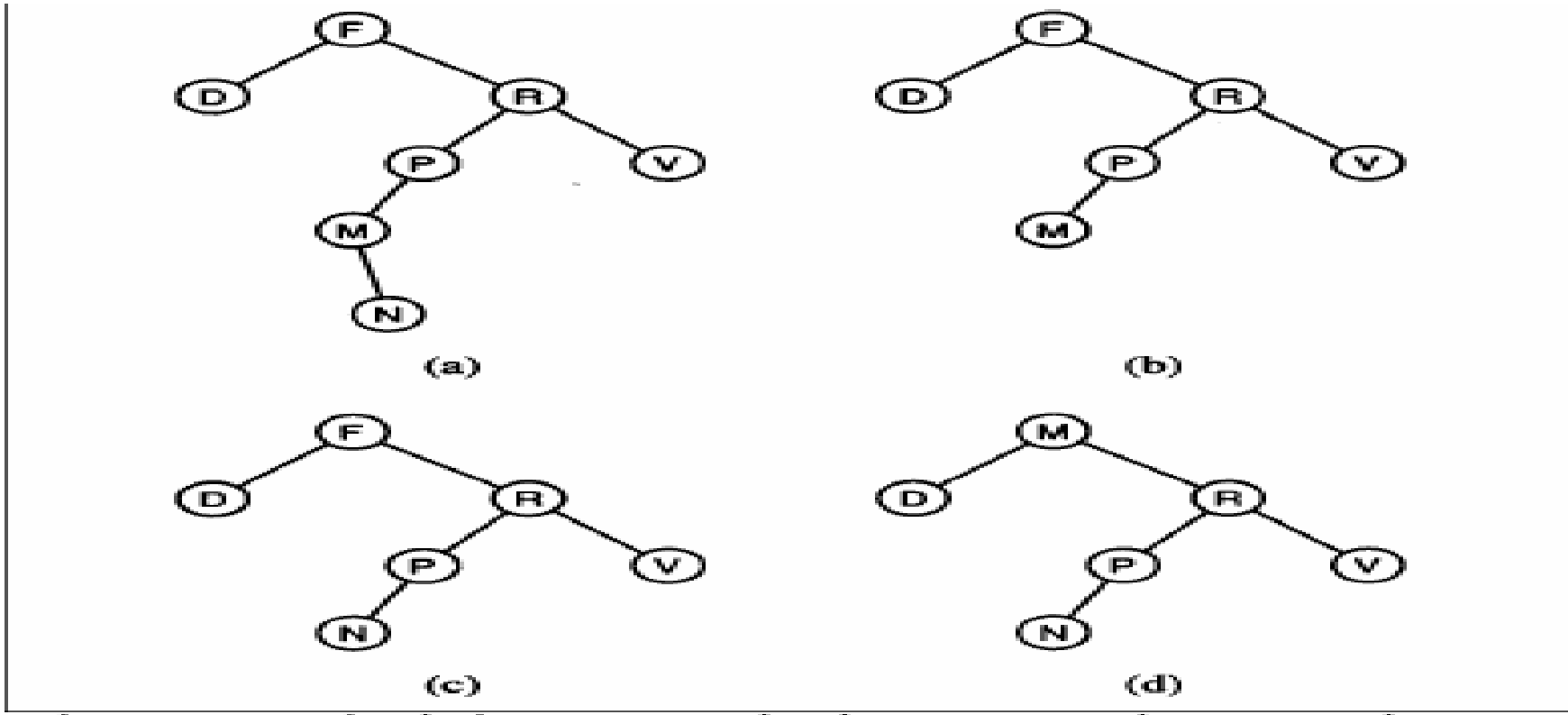
Γεγονός Prune

- Περιοδικά, ο εξυπηρέτης αποστέλλει κάποιες από τις πληροφορίες που περιέχονται στις ομάδες (πχ για λόγους χώρου) στους συνδρομητές κάθε ομάδας (γεγονός Prune).
- Για κάθε ομάδα G[k], 0 ≤ k ≤ MG-1, διαγράφονται από το δένδρο πληροφοριών της ομάδας όλες οι πληροφορίες που έχουν χρονοσφραγίδα μικρότερη από ή ίση με μια δοθείσα χρονοσφραγίδα που συσχετίζεται με το συγκεκριμένο γεγονός τύπου Prune και παρέχεται στο αρχείο εισόδου.
- Π.χ. Prune 10

Γεγονός Prune

- Για κάθε συνδρομητή S που είναι εγγεγραμμένος στην ομάδα k, όλες αυτές οι πληροφορίες (μία προς μία) εισάγονται στο δένδρο κατανάλωσης του S που αντιστοιχεί στην ομάδα k.

Γεγονός Prune (διαγραφή σε δυαδικό ταξινομημένο δένδρο)

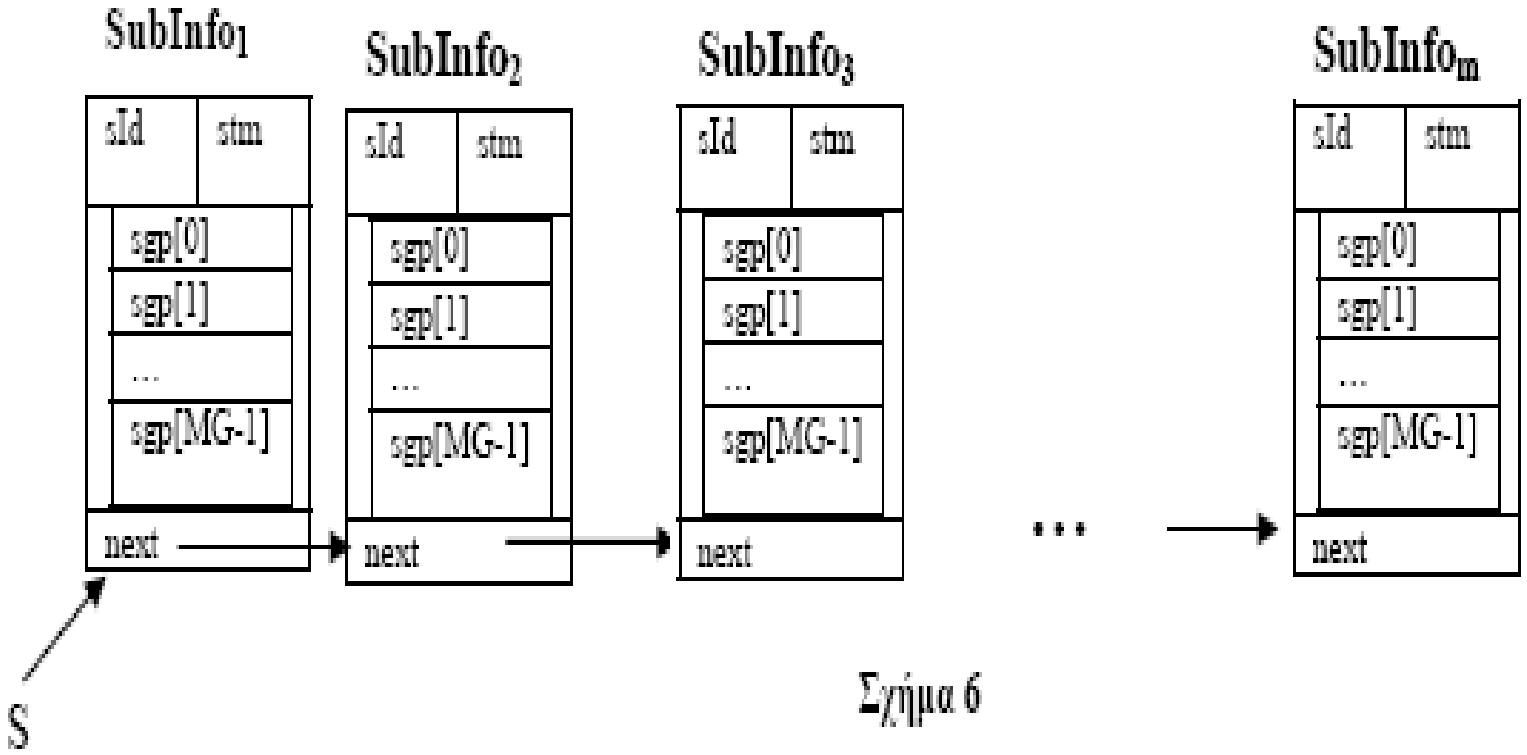


α) Αρχικό Δένδρο, β) Διαγραφή Ν από αρχικό δένδρο, γ) Διαγραφή Μ από αρχικό δένδρο, δ) Διαγραφή F από αρχικό δένδρο

Δενδρο Πληροφοριών

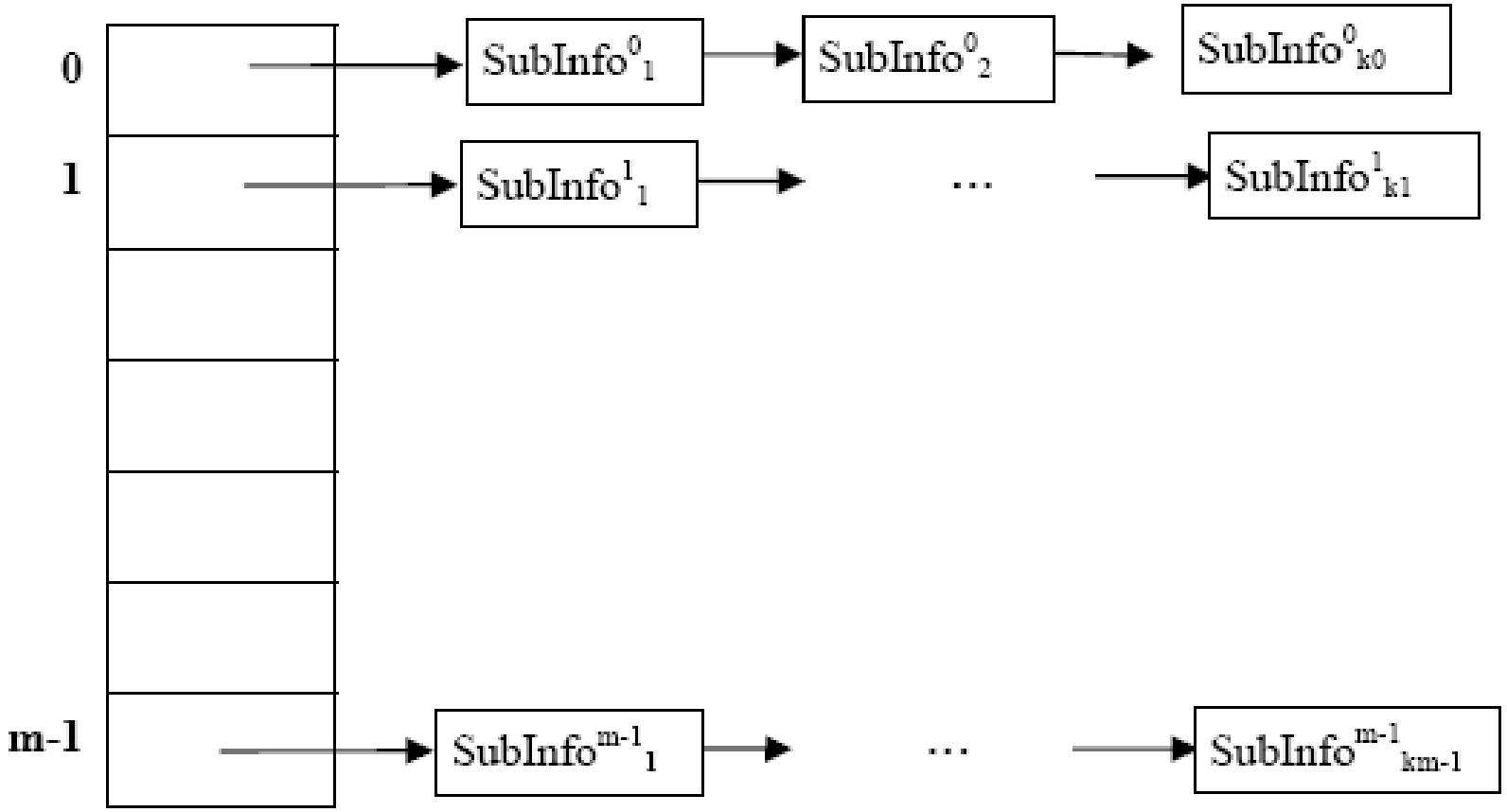
- Όπως καταλαβαίνετε για το Δένδρο Πληροφοριών χρειάζεται να υλοποιήσετε τις εξής συναρτήσεις:
  - BST_Insert,
  - BST_Delete και
  - BST_LookUp

Struct SubInfo (1ο Μέρος)



Σχήμα 6

Struct SubInfo (2ο Μέρος)



Struct SubInfo

- Ζητείται να υλοποιήσετε την τεχνική κατακερματισμού των **ταξινομημένων ξεχωριστών αλυσίδων.** Κάθε αλυσίδα διατηρείται **ταξινομημένη ως προς τα αναγνωριστικά** των συνδρομητών.

Struct SubInfo

```
struct SubInfo {                    // Εγγραφή πίνακα κατακερματισμού συνδρομητών
        int sId;                    // Αναγνωριστικό συνδρομητή
        int stim;                   // Χρονοσφραγίδα δημιουργίας συνδρομητή
        struct TreeInfo *tgp[MG];   // Πίνακας tgp MG θέσεων
        struct TreeInfo *sgp[MG];   // Πίνακας sgp MG θέσεων
        struct SubInfo *snext;      // Δείκτης στο επόμενο στοιχείο της αλυσίδας του πίνακα κατακερματισμού συνδρομητών
};
```

Struct SubInfo-> tgp[MG]

- MG δείκτες έναν για κάθε ομάδα.
- Ο δείκτης tgp[k] δείχνει στη ρίζα ενός δένδρου που αποθηκεύει πληροφορίες δημοσιευμένες στην ομάδα G[k].
- Ο tgp[k] έχει την τιμή 1αν ο συνδρομητής δεν είναι εγγεγραμμένος στην ομάδα με αναγνωριστικό k.
- Τόσα δένδρα κατανάλωσης όσες και οι ομάδες για τις οποίες ενδιαφέρεται ο συνδρομητής.

Struct SubInfo-> sgp[MG]

- MG δείκτες έναν για κάθε ομάδα.
- Ο δείκτης sgp[k] δείχνει σε ένα από τα στοιχεία της λίστας των φύλλων του δένδρου κατανάλωσης tgp[k] του συνδρομητή.
- Αν ο συνδρομητής δεν είναι εγγεγραμμένος στην ομάδα με αναγνωριστικό k, ο δείκτης sgp[k] του συνδρομητή έχει την τιμή 1.

Struct SubInfo -> snext

- Ο δείκτης snext δείχνει στο επόμενο στοιχείο της αλυσίδας στην οποία ανήκει η εγγραφή που αφορά τον συνδρομητή στον πίνακα κατακερματισμού των συνδρομητών.

Δένδρα Κατανάλωσης

- Κάθε ένα από τα δένδρα κατανάλωσης ενός συνδρομητή είναι ένα ταξινομημένο , διπλά συνδεδεμένο, φυλλοπροσανατολισμένο δυαδικό δένδρο του οποίου τα φύλλα έχουν διασυνδεθεί ώστε να σχηματίσουν μια απλά συνδεδεμένη (ταξινομημένη ως προς τη χρονοσφραγίδα) λίστα.
- Όλα τα κλειδιά που εισάγονται στο δένδρο βρίσκονται πάντα αποθηκευμένα στα φύλλα, από αριστερά προς τα δεξιά κατά μη φθίνουσα τιμή κλειδιού.
- Στους εσωτερικούς κόμβους αποθηκεύονται κατάλληλα κλειδιά που καθιστούν εφικτή την ορθή διεκπεραίωση λειτουργιών LookUp() στο δένδρο.

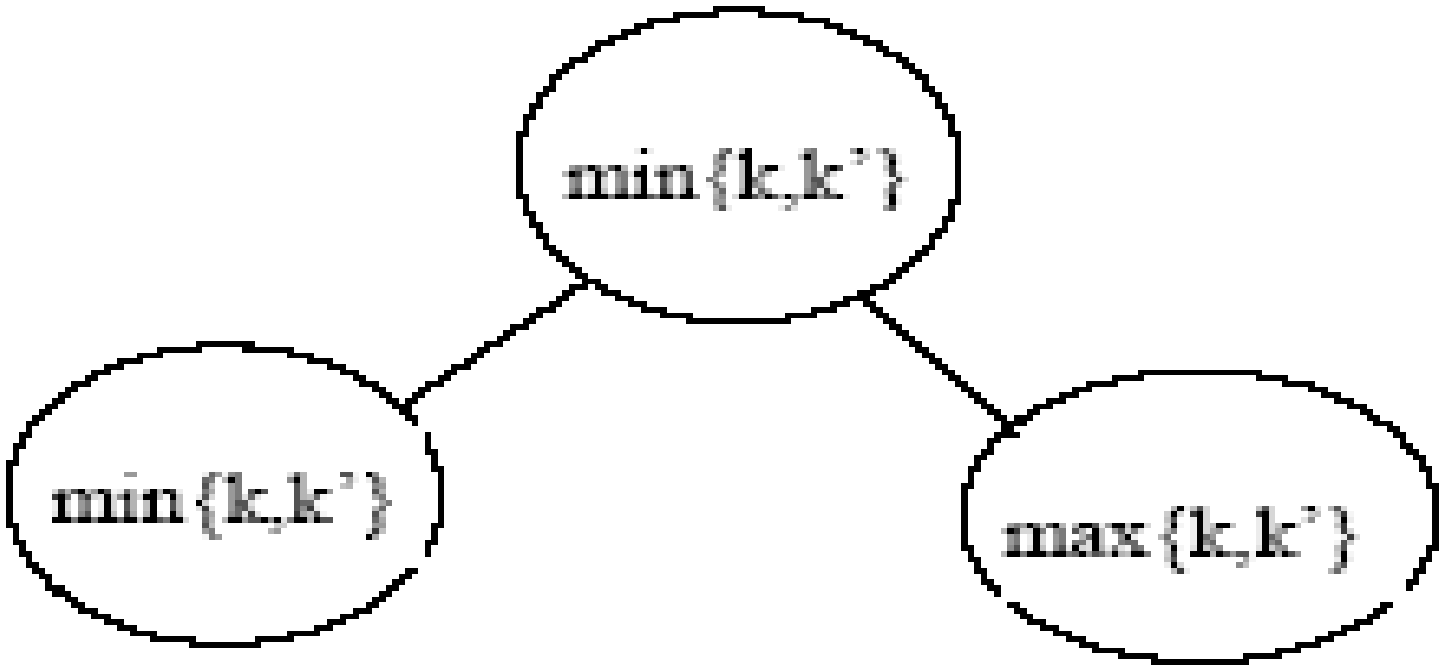
Δένδρα Κατανάλωσης - Αναζήτηση

- Για την αναζήτηση ενός κλειδιού k, ακολουθείται ο κλασικός αλγόριθμος αναζήτησης του k σε ένα ταξινομημένο δυαδικό δένδρο.
- Οι αναζητήσεις κλειδιών είναι επιτυχημένες μόνο αν βρίσκουν το ζητούμενο κλειδί σε έναν από τους κόμβους φύλλα του δένδρου.

Δένδρα Κατανάλωσης - Εισαγωγή Νέου Κλειδιού

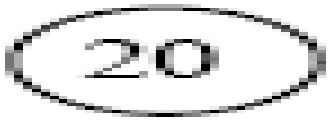
- Για την εισαγωγή ενός νέου κλειδιού k, ακολουθείται ο κλασικός αλγόριθμος αναζήτησης του k σε ένα ταξινομημένο δυαδικό δένδρο μέχρι να προσεγγιστεί κάποιος κόμβος φύλλο.
- Έστω k’ το κλειδί του φύλλου αυτού. Τότε, η εισαγωγή γίνεται αντικαθιστώντας το φύλλο με κλειδί k’ με την τριάδα κόμβων που φαίνεται στη συνέχεια.

Δένδρα Κατανάλωσης - Εισαγωγή Νέου Κλειδιού



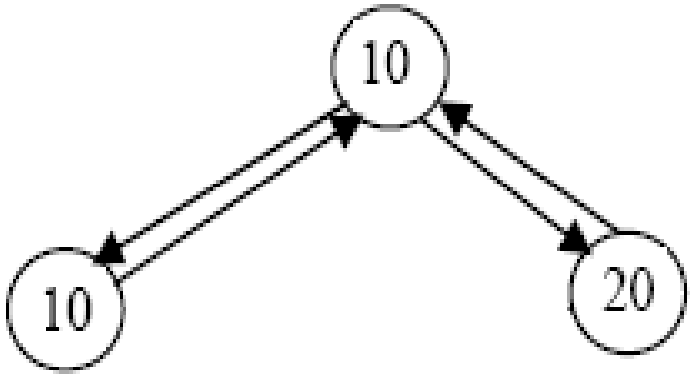
Δένδρα Κατανάλωσης – Παράδειγμα

- Αρχικά



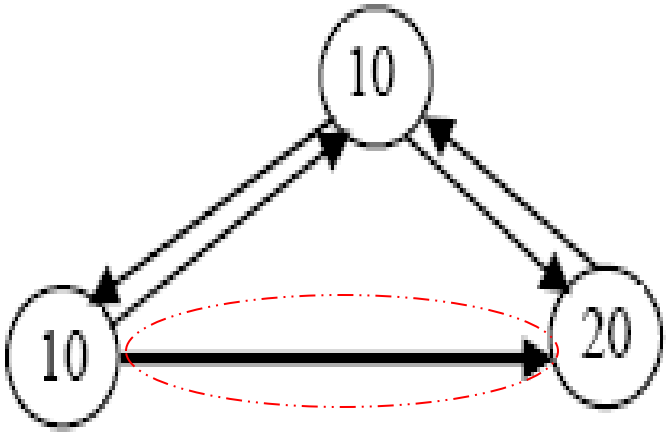
Δένδρα Κατανάλωσης – Παράδειγμα

- Insert(10)



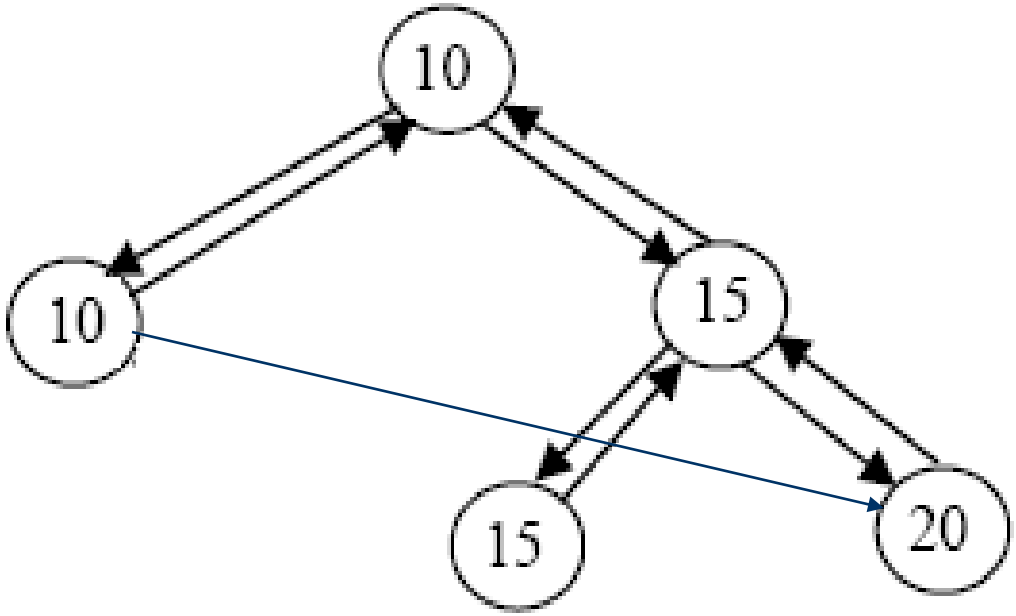
Δένδρα Κατανάλωσης – Παράδειγμα

- Διασύνδεση ταξινομημένης και απλά συνδεδεμένης λίστας φύλλων του δέντρου κατανάλωσης.



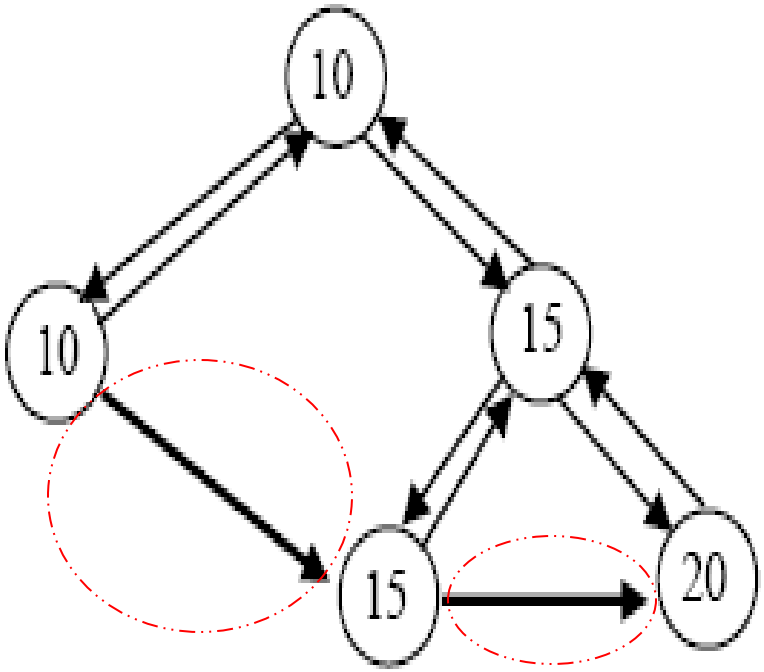
Δένδρα Κατανάλωσης – Παράδειγμα

- Insert(15)



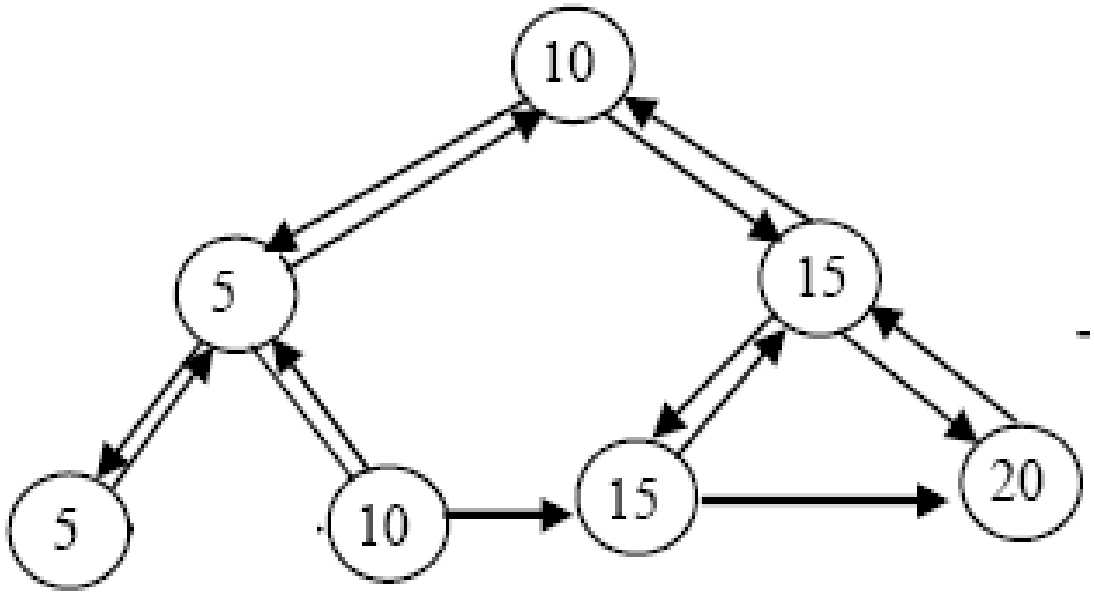
Δένδρα Κατανάλωσης – Παράδειγμα

- Διασύνδεση ταξινομημένης και απλά συνδεδεμένης λίστας φύλλων του δέντρου κατανάλωσης.



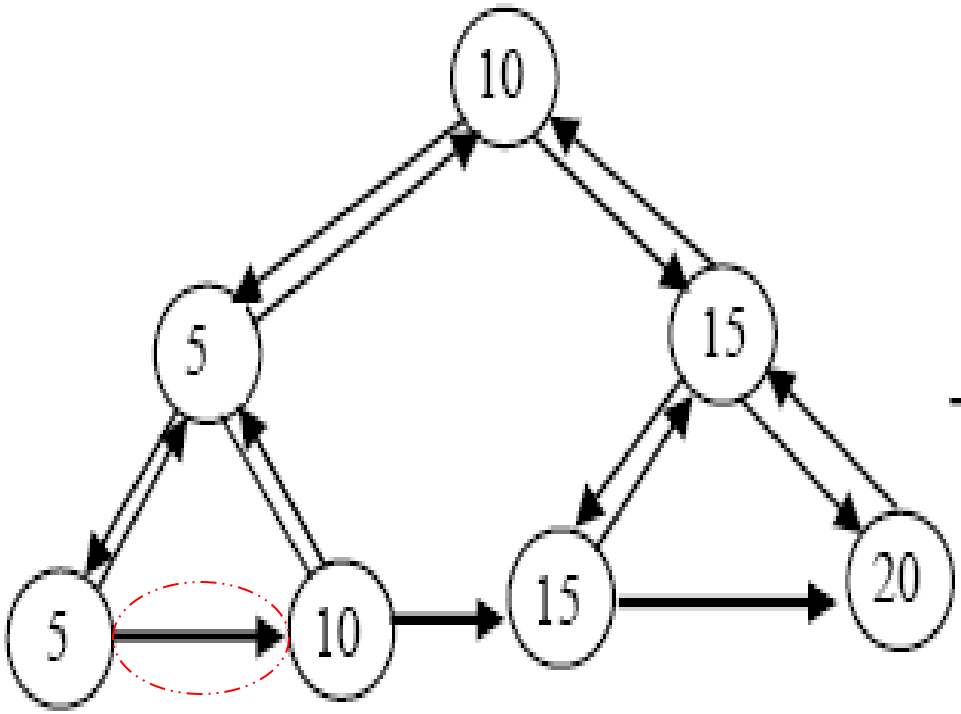
Δένδρα Κατανάλωσης – Παράδειγμα

- Insert(5)

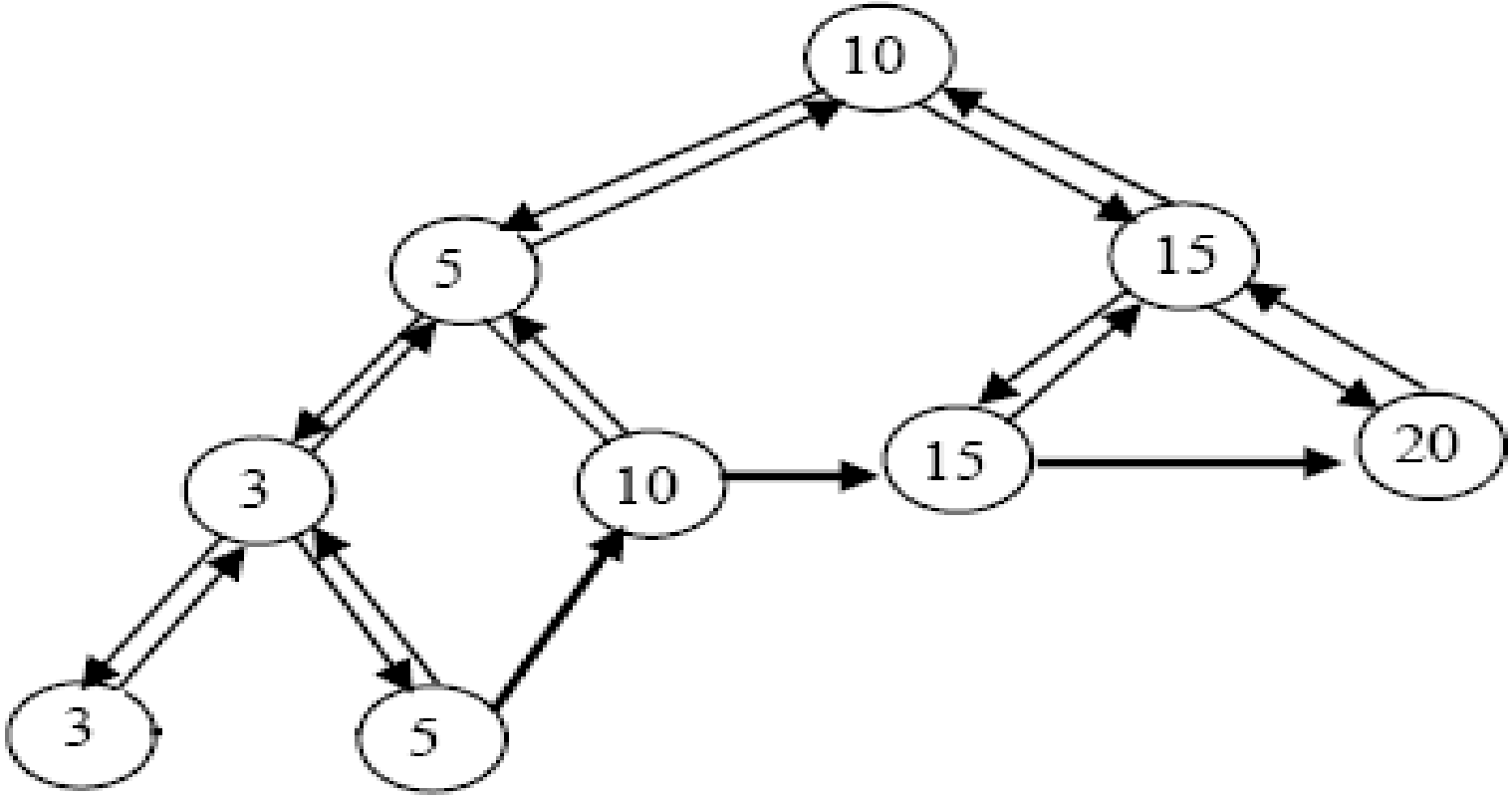


Δένδρα Κατανάλωσης – Παράδειγμα

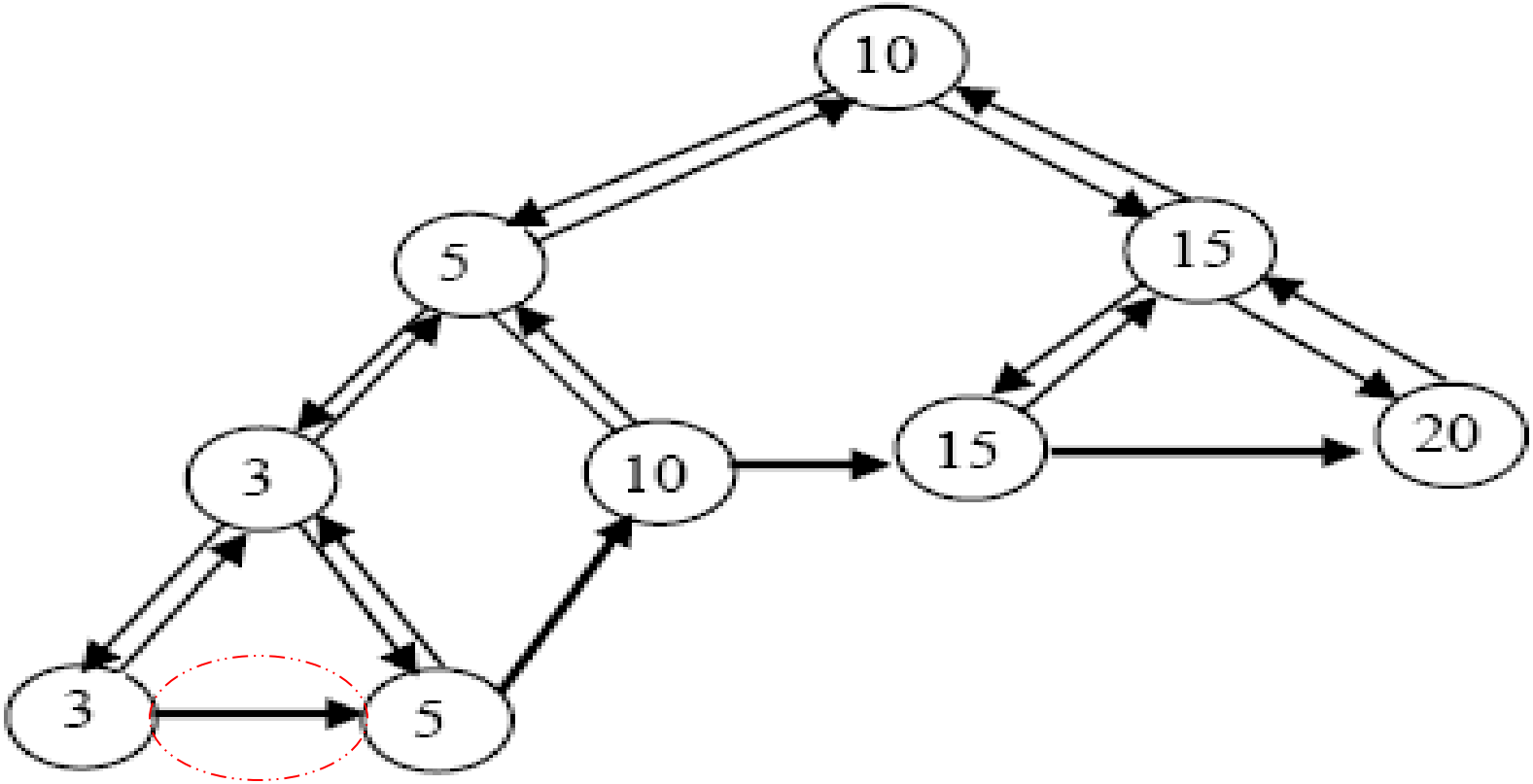
- Διασύνδεση ταξινομημένης και απλά συνδεδεμένης λίστας φύλλων του δέντρου κατανάλωσης.



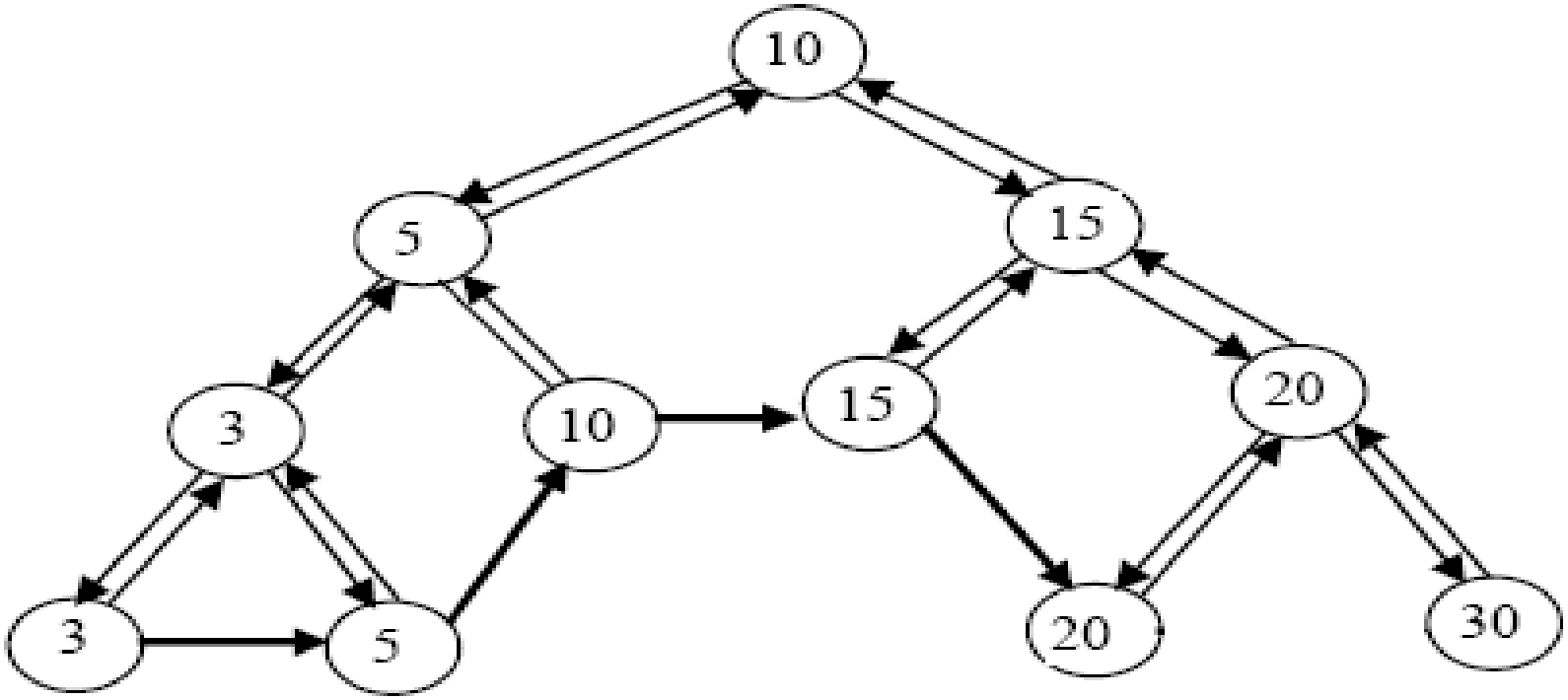
Δένδρα Κατανάλωσης – Παράδειγμα Insert(3)



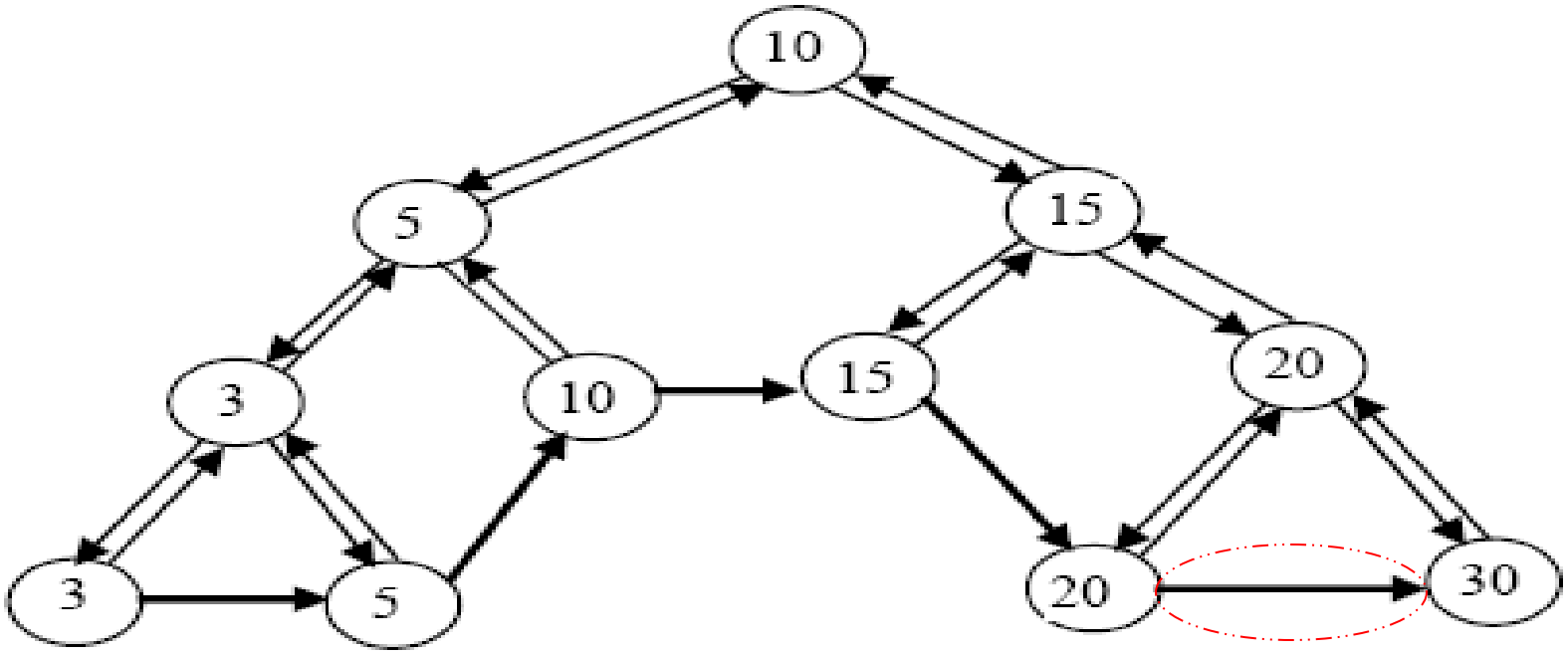
Δένδρα Κατανάλωσης – Παράδειγμα



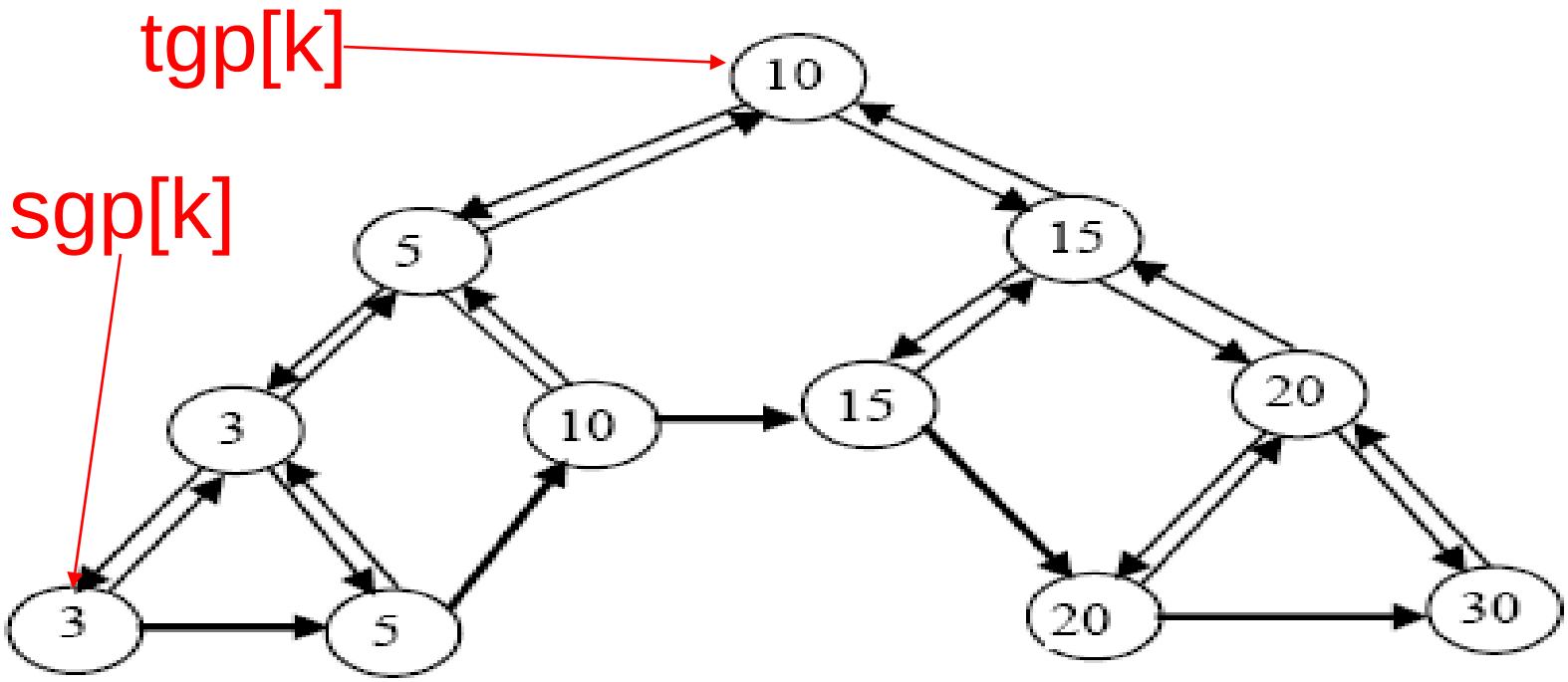
Δένδρα Κατανάλωσης – Παράδειγμα Insert(30)



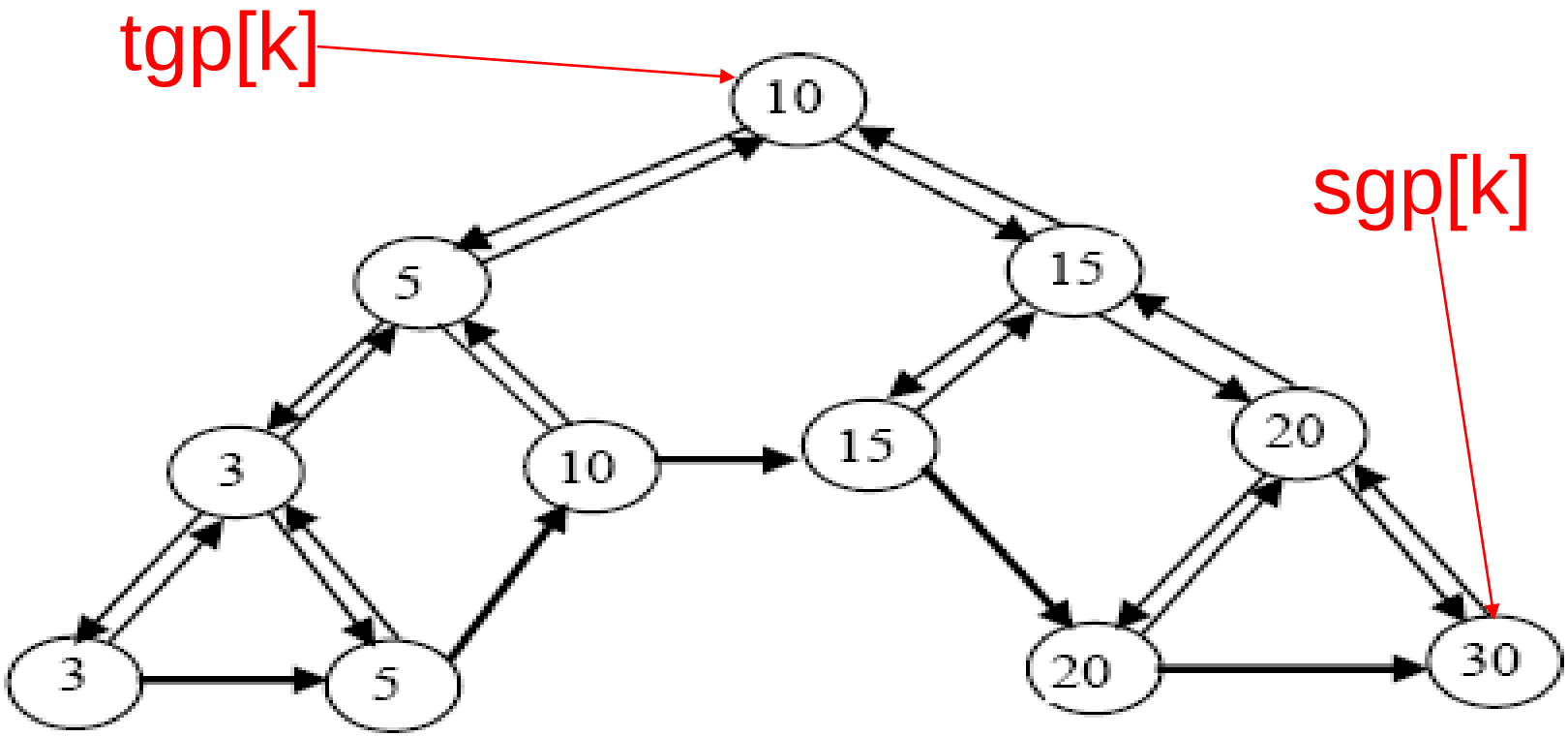
Δένδρα Κατανάλωσης – Παράδειγμα



Δένδρα Κατανάλωσης – Παράδειγμα



Δένδρα Κατανάλωσης – Παράδειγμα Consume



Δένδρα Κατανάλωσης

- Κάθε φυλλοπροσανατολισμένο δένδρο είναι γεμάτο.
- Όλα τα φύλλα του είναι συνδεδεμένα μεταξύ τους σχηματίζοντας μια ταξινομημένη (ως προς τη χρονοσφραγίδα κάθε εγγραφής) απλά συνδεδεμένη λίστα.
- Το πρώτο στοιχείο αυτής της λίστας είναι το αριστερότερο φύλλο του φυλλοπροσανατολισμένου δένδρου, ενώ το τελευταίο στοιχείο της λίστας είναι το δεξιότερο φύλλο του δένδρου.

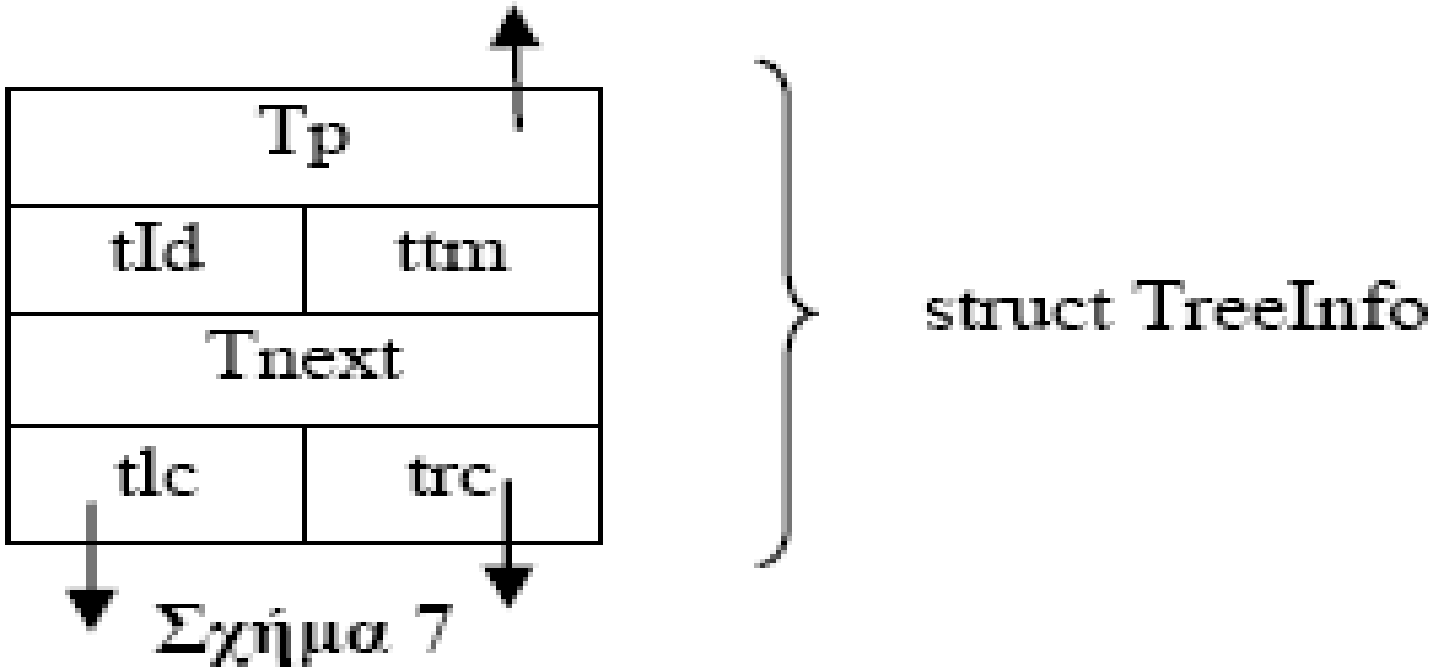
Δένδρα Κατανάλωσης

- Για τα Δένδρα Πληροφοριών χρειάζεται να υλοποιήσετε τις εξής συναρτήσεις:
  - LO_BST_Insert και
  - LO_BST_LookUp
  - Μια συνάρτηση η οποία θα παίρνει ως παράμετρο έναν δείκτη p σε ένα κόμβο φύλλο του δένδρου και θα επιστρέφει έναν δείκτη στο αμέσως προηγούμενο φύλλο του δένδρου (δηλαδή στο φύλλο του δένδρου με το μεγαλύτερο κλειδί μεταξύ εκείνων των φύλλων που έχουν μικρότερα κλειδιά από εκείνο του κόμβου p).

Struct TreeInfo

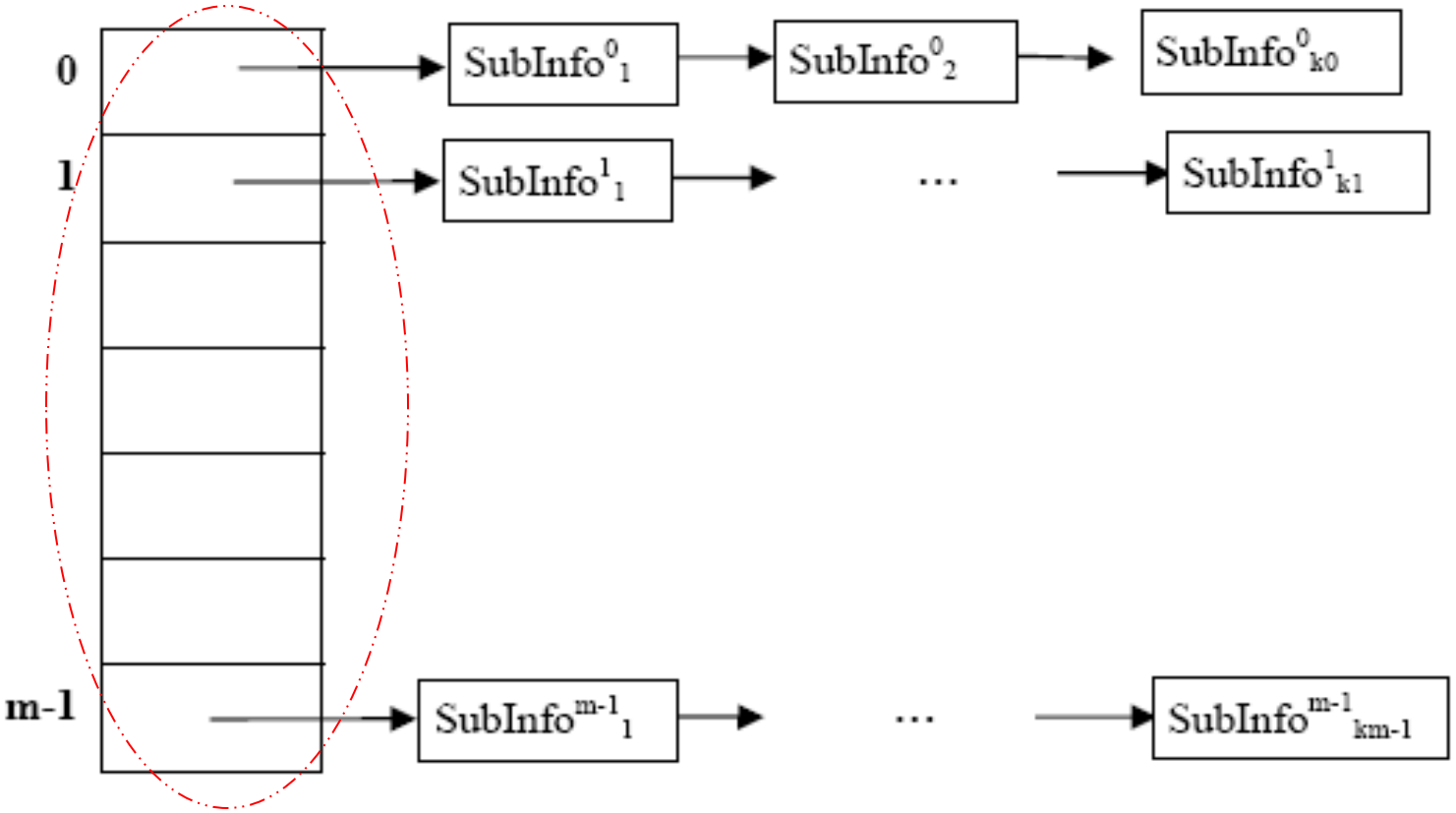
```
Struct TreeInfo {                  // Εγγραφή δένδρου κατανάλωσης
        int tId;                   // Αναγνωριστικό πληροφορίας
        int ttm;                   // Χρονοσφραγίδα δημοσίευσης πληροφορίας
        struct TreeInfo *tlc;      // Δείκτης στον αριστερό θυγατρικό κόμβο
        struct TreeInfo *trc;      // Δείκτης στον δεξιό θυγατρικό κόμβο
        struct TreeInfo *tp;       // Δείκτης στον πατρικό κόμβο
        struct TreeInfo *next;     // Δείκτης στο επόμενο φύλλο προς τα δεξιά (NULL αν η εγγραφή δεν αντιστοιχεί σε κόμβο φύλλο)
};
```

Struct TreeInfo



Σχήμα 7

Hash-Tables


0
1
m-1
SubInfo$^{0}_{1}$
SubInfo$^{0}_{2}$
SubInfo$^{0}_{k0}$
SubInfo$^{1}_{1}$
…
SubInfo$^{1}_{k1}$
SubInfo$^{m-1}_{1}$
…
SubInfo$^{m-1}_{km-1}$

Κατακερματισμός - Αρχές Λειτουργίας

- Έστω S <= U το προς αποθήκευση σύνολο κλειδιών.
- Ας υποθέσουμε πως ο χώρος των κλειδιών είναι το σύνολο των φυσικών αριθμών.
- Αποθηκεύουμε τα στοιχεία του συνόλου S σε έναν πίνακα m θέσεων, που ονομάζεται πίνακας κατακερματισμού και χρησιμοποιούμε μια συνάρτηση h η οποία απεικονίζει το σύνολο {0,…,|U|} στο σύνολο {0,…m}. Η συνάρτηση αυτή λέγεται συνάρτηση κατακερματισμού. Το στοιχείο με κλειδί k αποθηκεύεται στη θέση A[h[k]] του πίνακα.

Κατακερματισμός - Αρχές Λειτουργίας

- Πάντα υπάρχει ένα πάνω όριο m στο διαθέσιμο χώρο.
- Η σπατάλη σε μνήμη είναι μεγάλη όταν το σύνολο έχει λίγα στοιχεία ενώ ο χώρος U των κλειδιών είναι τεράστιος.

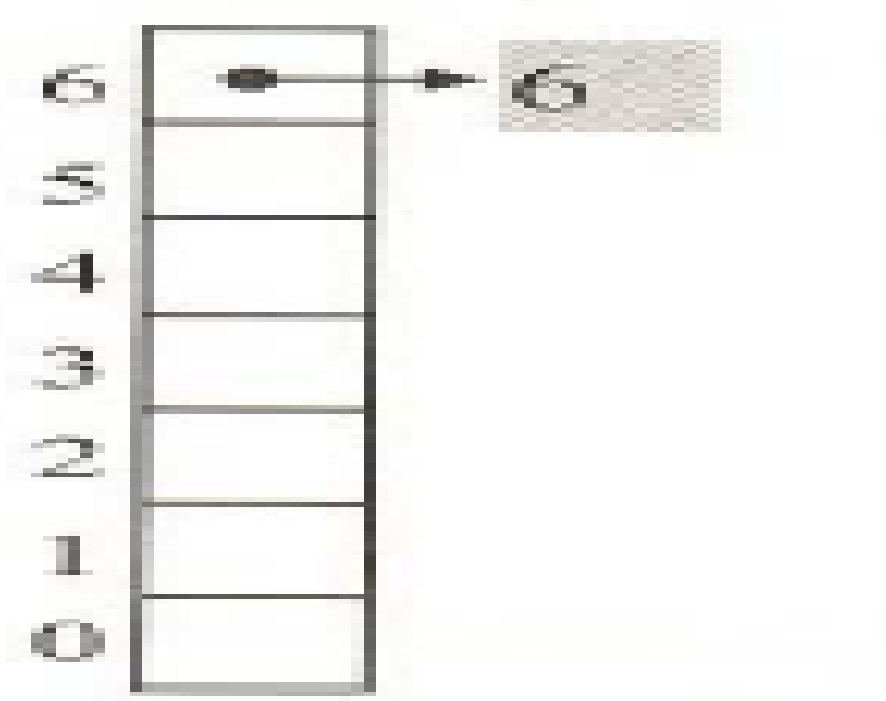
Κατακερματισμός

- Συγκρούσεις - Όταν για δύο κλειδιά Ki και Kj με Ki ≠ Kj ισχύει ότι h(Ki) = h(Kj) λέμε πως συμβαίνει σύγκρουση.
- Καλές Συναρτήσεις Κατακερματισμού - Κάνουν καλή διασκόρπιση των κλειδιών στον πίνακα.
- Παράδειγμα Συνάρτησης Κατακερματισμού - h(k) = k mod m

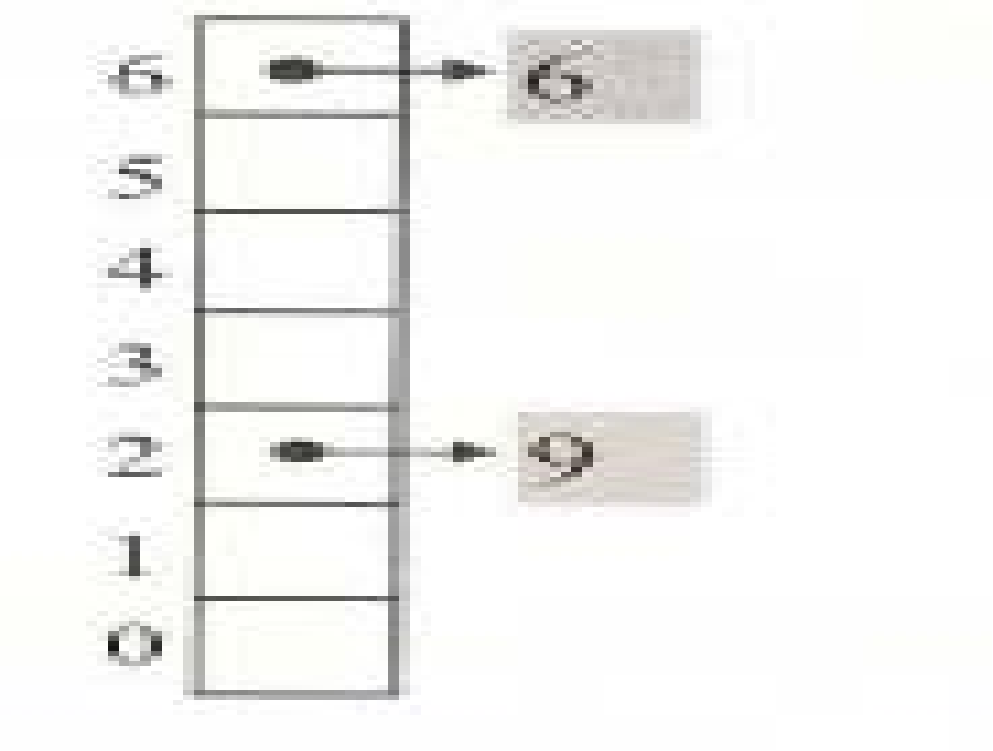
Κατακερματισμός - Μέθοδος ξεχωριστών αλυσίδων

- Η θέση A[j] του πίνακα κατακερματισμού δεν περιέχει ένα στοιχείο αλλά ένα δείκτη σε μια δυναμική δομή η οποία περιέχει κάθε στοιχείο με κλειδί K τέτοιο ώστε h(K) = j.
- Παράγειγμα →

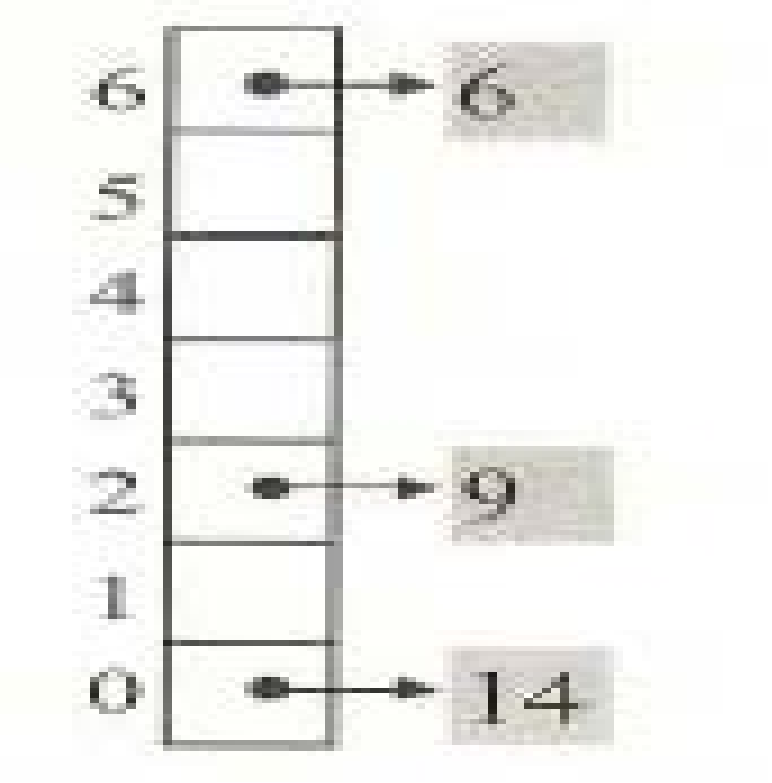
Μέθοδος ξεχωριστών αλυσίδων – Παράδειγμα h(k) = k mod 7



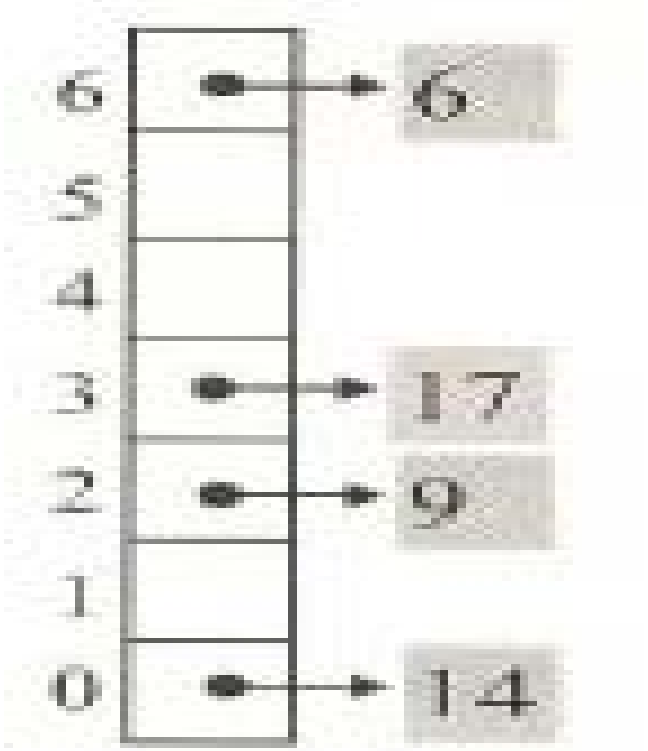
Μέθοδος ξεχωριστών αλυσίδων – Παράδειγμα h(k) = k mod 7



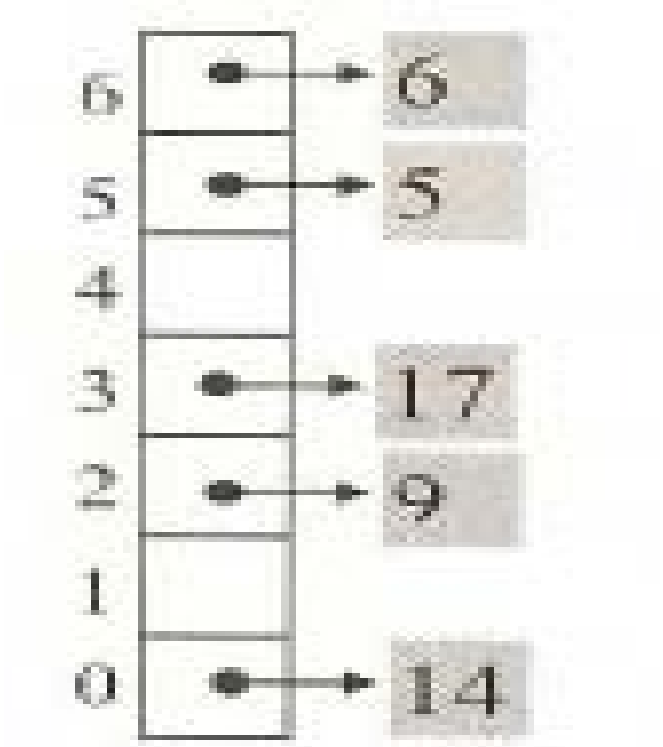
Μέθοδος ξεχωριστών αλυσίδων – Παράδειγμα h(k) = k mod 7



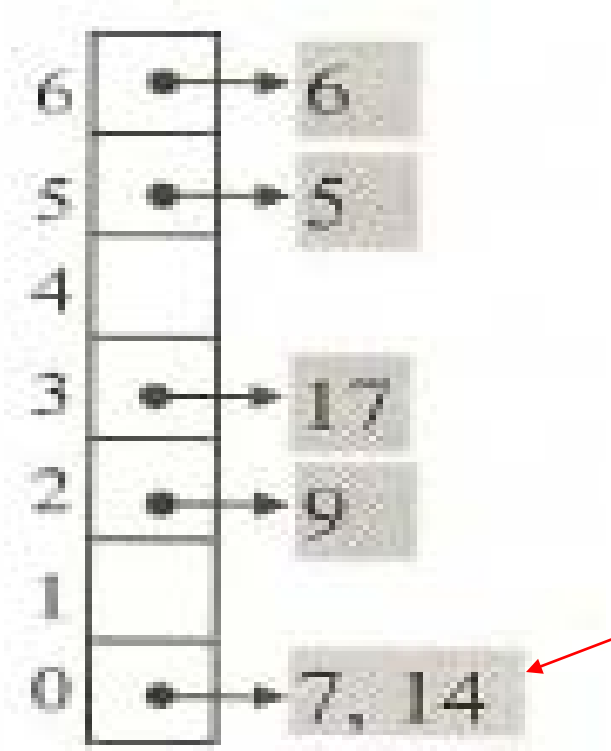
Μέθοδος ξεχωριστών αλυσίδων – Παράδειγμα h(k) = k mod 7



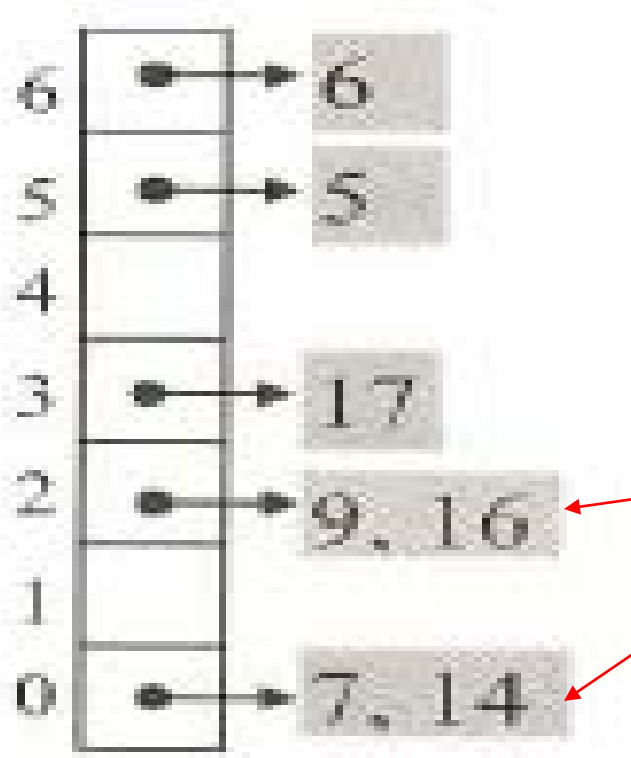
Μέθοδος ξεχωριστών αλυσίδων – Παράδειγμα h(k) = k mod 7



Μέθοδος ξεχωριστών αλυσίδων – Παράδειγμα h(k) = k mod 7

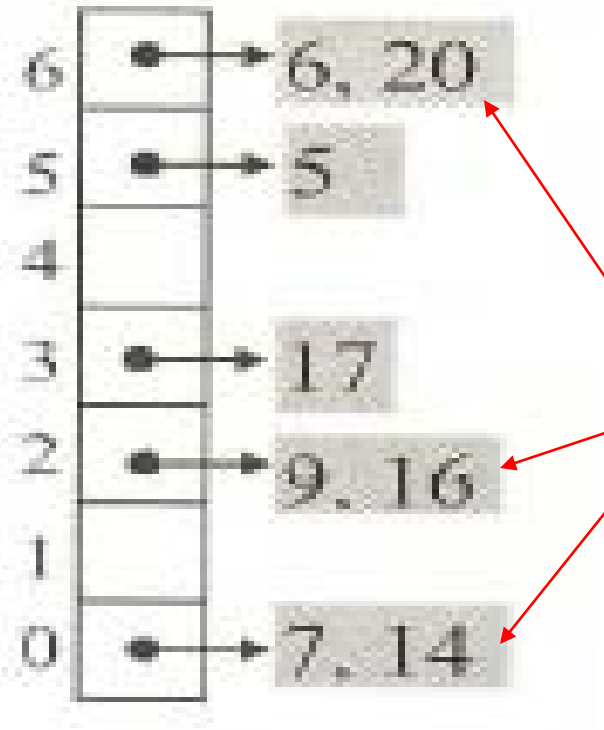


Μέθοδος ξεχωριστών αλυσίδων – Παράδειγμα h(k) = k mod 7



!!!Collision!!!

Μέθοδος ξεχωριστών αλυσίδων – Παράδειγμα h(k) = k mod 7



Μέθοδος Διαχείρισης Συγκρούσεων των Ξεχωριστών Αλυσίδων

- HT_Insert(HashTable A, Key K)
- HT_LookUp(HashTable A, Key k) {

  pointer p; int pos;

  pos = h(K); // εύρεση του κλειδιού του στοιχείου e βάσει της συναρτήσεως κατακερματισμού

  p = A[pos];

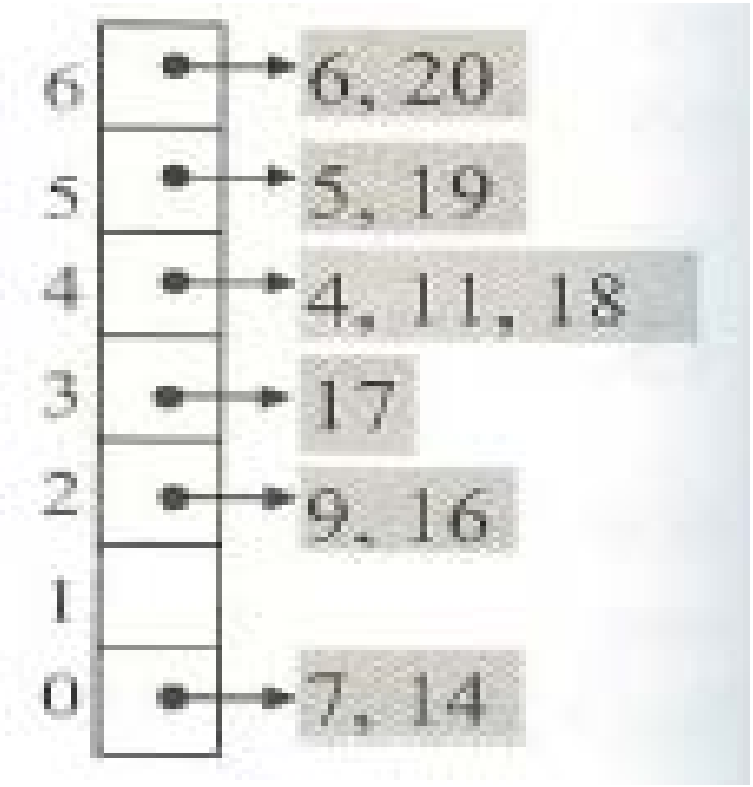
  // ο p είναι δείκτης στο πρώτο στοιχείο της αλυσίδας

  while (p!= NULL AND p->key != K) p = p->next;

  // διάσχιση της αλυσίδας μέχρι είτε να βρεθεί το κλειδί ή να φθάσουμε στο τέλος της

  return p;}
- HT_Delete(HashTable A, Key K)

Μέθοδος Διαχείρισης Συγκρούσεων των Ξεχωριστών Αλυσίδων



Καθολικές Κλάσεις Συναρτήσεων Κατακερματισμού – Probleeeeem

??????????????????????????????????

- Ένας κακόβουλος αντίπαλος που γνωρίζει τη συνάρτηση κατακερματισμού που χρησιμοποιεί ένας αλγόριθμος μπορεί να επιλέξει τα προς αποθήκευση κλειδιά έτσι ώστε να έχουν όλα την ίδια τιμή κατακερματισμού.

??????????????????????????????????

Καθολικές Κλάσεις Συναρτήσεων Κατακερματισμού

- Επιλέγουμε τη συνάρτηση κατακερματισμού με τυχαίο τρόπο, ανεξάρτητο από τα κλειδιά που πρόκειται να αποθηκευθούν τελικά στη δομή κατακερματισμού, από μια προσεχτικά σχεδιασμένη κλάση συναρτήσεων κατά την εκκίνηση της εκτέλεσης.
- Η προσέγγιση αυτή ονομάζεται καθολικός κατακερματισμός.
- Λόγω της τυχαιότητας, ο αλγόριθμος μπορεί να συμπεριφέρεται διαφορετικά σε κάθε εκτέλεση, ακόμη και για την ίδια είσοδο.
- Με τον τρόπο αυτό εξασφαλίζεται καλή αναμενόμενη επίδοση για οποιαδήποτε είσοδο

Δημιουργία μιας Καθολικής Κλάσης Συναρτήσεων Κατακερματισμού

- Επιλέγουμε έναν πρώτο αριθμό p αρκετά μεγάλο ώστε όλα τα δυνατά κλειδιά K να βρίσκονται στο διάστημα {0,.., p-1}.
- Για κάθε αριθμό a Ε {1,…,p-1} και b Ε {0,…,p-1} έστω
  - **$h_{a,b}(x) = ((ax+b) \bmod N) \bmod m$**
- Τότε, το σύνολο συναρτήσεων H = {$h_{a,b}$: $1 \le a < p$ και $0 \le b < p$} είναι μια καθολική κλάση συναρτήσεων κατακερματισμού.

Δημιουργία μιας Καθολικής Κλάσης Συναρτήσεων Κατακερματισμού

- Πρέπει να υλοποιήσετε μια συνάρτηση Universal_Hash_Function() η οποία θα υλοποιεί την καθολική κλάση συναρτήσεων προγραμματισμού.

Περιγραφή Γεγονότων Εισόδου

- **run <m> <p> <input-file>**
  - <m> είναι ένας πρώτος αριθμός που καθορίζει το μέγεθος του πίνακα κατακερματισμού των συνδρομητών,
  - <p> είναι ένας πρώτος αριθμός που θα χρησιμοποιήσετε για την σχεδίαση της καθολικής κλάσης των συναρτήσεων κατακερματισμού

Περιγραφή Γεγονότων Εισόδου

- **I <itm> <ild> <gld1> <gld2> … <gldk> -1**

- Γεγονός τύπου Insert_Info το οποίο έχει την ίδια μορφή με εκείνη του 1ου μέρους της προγραμματιστικής εργασίας.

Περιγραφή Γεγονότων Εισόδου

- **S <sTM> <sId> <gId1> <gId2> … <gIdm> -1**

- Γεγονός τύπου Subscriber_Registration το οποίο σηματοδοτεί την εγγραφή ενός νέου συνδρομητή με αναγνωριστικό <sId> στο σύστημα την χρονική στιγμή <sTM>, όπως και στο 1ο μέρος της εργασίας.

Περιγραφή Γεγονότων Εισόδου

- **R <tm>**

- Γεγονός τύπου Prune («κλάδεμα») το οποίο σηματοδοτεί την εκκίνηση ενέργειας αποστολής των πληροφοριών κάθε ομάδας που έχουν χρονοσφραγίδα μικρότερη της <tm> στους συνδρομητές που είναι εγγεγραμμένοι στην ομάδα αυτή

Περιγραφή Γεγονότων Εισόδου

- Για κάθε ομάδα G[k] πραγματοποιούνται οι εξής ενέργειες:
  - διάσχιση του δένδρου πληροφοριών της G[k] και για κάθε εγγραφή I η οποία έχει χρονοσφραγίδα μικρότερη από ή ίση με <tm>, η εγγραφή I διαγράφεται από το δένδρο
  - Για κάθε εγγραφή S της λίστας συνδρομών της G[k], η I εισάγεται στο δένδρο κατανάλωσης του S που αντιστοιχεί στην ομάδα G[k] (δηλαδή στο δένδρο κατανάλωσης που δεικτοδοτείται από τον δείκτη tgp[k] του S).

Περιγραφή Γεγονότων Εισόδου

R DONE

GROUPID = <gId1>, INFOLIST = <iId_1^1, iId_2^1, …, iId_{r1}^1>, SUBLIST = < sId_1^1, sId_2^1, …, sId_{n1}^1>

GROUPID = <gId2>, INFOLIST = <iId_1^2, iId_2^2, …, iId_{r2}^2>, SUBLIST = < sId_1^2, sId_2^2, …, sId_{n2}^2>

…

GROUPID = <gIdMG>, INFOLIST= <iId_1^{MG}, iId_2^{MG}, …, iId_{rMG}^{MG}>, SUBLIST= < sId_1^{MG}, sId_2^{MG},…, sId_{nMG}^{MG}>

SUBSCRIBERID = <sId1>, GROUPLIST =

<gId_1^1> , TREELIST= <tId_1^1(<gId_1^1>), tId_2^1(<gId_1^1>), …, tId_{r1}^1(<gId_1^1>)>,

…

<gId_{v1}^1>, TREELIST = <tId_1^1(<gId_{v1}^1>), tId_2^1(<gId_{v1}^1>), …, tId_{rv1}^1(<gId_{v1}^1>)>,

SUBSCRIBERID = <sId2>, GROUPLIST =

<gId_1^2>, TREELIST = < tId_1^2(<gId_1^2>), tId_2^2(<gId_1^2>), …, tId_{r2}^2(<gId_1^2>)>

…

<gId_{v2}^2>, TREELIST = < tId_1^2(<gId_{v2}^2>), tId_2^2(<gId_{v2}^2>), …, tId_{rv2}^2(<gId_{v2}^2>)>

…

SUBSCRIBERID = <sIdq>, GROUPLIST =

<gId_1^q> , TREELIST = < tId_1^q(<gId_1^q>), tId_2(<gId_1^q>), …, tId_{rq}^q(<gId_1^q>)>

…

<gId_{vq}^q>, TREELIST = < tId_1^q(<gId_{vq}^q>), tId_2^q(<gId_{vq}^q>), …, tId_{rvq}^q(<gId_{vq}^q>)>

Περιγραφή Γεγονότων Εισόδου

- **C <sId>**

- Κατά το γεγονός τύπου Consume, κάθε συνδρομητής καταναλώνει **τις νέες πληροφορίες** που περιέχονται στα δένδρα πληροφοριών που δεικτοδοτούνται από τους tgp δείκτες που περιέχει η εγγραφή του συνδρομητή στον πίνακα κατακερματισμού συνδρομητών.

Περιγραφή Γεγονότων Εισόδου

C <sId> DONE

GROUPID = <gId1>, TREELIST = <tId_1^1, tId_2^1, …, tId_{r1}^1>, NEWSGP = <tId1>

GROUPID = <gId2>, TREELIST = <tId_1^2, tId_2^2, …, tId_{r2}^2>, NEWSGP = <tId2>

…

GROUPID = <gIdk>, TREELIST = <tId_1^k, tId_2^k, …, tId_{rk}^k>, NEWSGP = <tIdk>

Περιγραφή Γεγονότων Εισόδου

- **D <sId>**

- Το γεγονός αυτό σηματοδοτεί την αποχώρηση του συνδρομητή με αναγνωριστικό <sId> από το σύστημα.

Περιγραφή Γεγονότων Εισόδου

- **P**

- Γεγονός τύπου Print που σηματοδοτεί την εκτύπωση των δομών δεδομένων του συστήματος.

Περιγραφή Γεγονότων Εισόδου

P DONE

GROUPID = <gId1>, INFOLIST = <iId_1^1, iId_2^1, …, iId_{r1}^1>, SUBLIST = < iId_1^1, iId_2^1, …, iId_{n1}^1>

GROUPID = <gId2>, INFOLIST = <iId_1^2, iId_2^2, …, iId_{r2}^2>, SUBLIST = < iId_1^2, iId_2^2, …, iId_{n2}^2>

…

GROUPID = <gIdMG>, INFOLIST= <iId_1^{MG}, iId_2^{MG}, …, iId_{rk}^{MG}>, SUBLIST= < iId_1^{MG}, iId_2^{MG},…, iId_{nk}^{MG}>

SUBSCRIBERLIST = <sId1, sId2, …, sIdm>

SUBSCRIBERID = <sId1>, GROUPLIST =

<gId_1^1> , TREELIST= <tId_1^1(<gId_1^1>), tId_2^1(<gId_1^1>), …, tId_{r1}^1(<gId_1^1>)>,

…

<gId_{v1}^1>, TREELIST = <tId_1^1(<gId_{v1}^1>), tId_2^1(<gId_{v1}^1>), …, tId_{rv1}^1(<gId_{v1}^1>)>,

SUBSCRIBERID = <sId2>, GROUPLIST =

<gId_1^2>, TREELIST = < tId_1^2(<gId_1^2>), tId_2^2(<gId_1^2>), …, tId_{r1}^2(<gId_1^2>)>

…

<gId_{v2}^2>, TREELIST = < tId_1^2(<gId_{v2}^2>), tId_2^2(<gId_{v2}^2>), …, tId_{rv2}^2(<gId_{v2}^2>)>

…

SUBSCRIBERID = <sIdq>, GROUPLIST =

<gId_1^q> , TREELIST = < tId_1^q(<gId_1^q>), tId_2^q(<gId_1^q>), …, tId_{rq}^q(<gId_1^q>)>

…

<gId_{vq}^q>, TREELIST = < tId_1^q(<gId_{vq}^q>), tId_2^q(<gId_{vq}^q>), …, tId_{rvq}^q(<gId_{vq}^q>)>

NO_GROUPS = <number of groups>, NO_SUBSCRIBERS = <number of subscribers>

Οδηγίες και Συμβουλές για την Ομαλή Διεκπεραίωση της Εργασίας

- Θα ήταν καλύτερο για εσάς να ακολουθήσετε την βηματική διεκπεπαίωση της εργασίας, όπως προτείνεται και από την εκφώνηση.
- Γενικότερα, δεν «πετάμε» το παλιό πρόγαμμα αλλά προσπαθούμε να το βελτιώσουμε και να το προσαρμόσουμε στις απαιτήσεις της νέας εργασίας.

Παραδοση Εργασιας

- Ημερομηνία Παράδοσης : Παρασκευή 18/12/2009
- Τρόπος Παράδοσης : πρόγραμμα submit