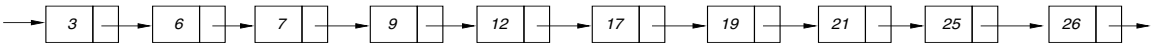
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Μια σημείωση από τον Α. Δελή για το άρθρο: W. Pugh, Skip Lists: A Probabilistic Alternative to Balanced Trees, *Comms of the ACM*, 33(6), June 1990, 668-676.

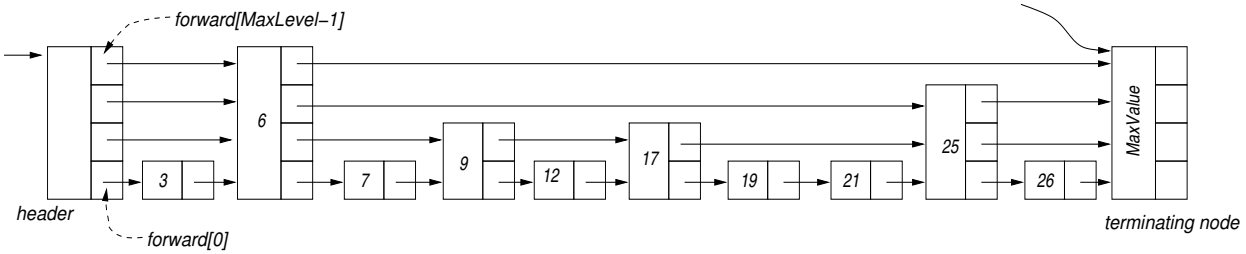
Εισαγωγη στη δομή δεδομενων Skip-List

Η Skip-List μοιάζει αρκετά με μια απλή ταξινομημένη συνδεδεμένη λίστα όπως εκείνη του Σχήματος 1. Ο κάθε



Σχήμα 1: Ταξινομημένη συνδεδεμένη λίστα

κόμβος της skip-list ομως μπορεί να περιέχει ένα ή περισσότερους δείκτες (forward pointers) προς κάποιους από τους επόμενους κόμβους της λίστας. Το Σχήμα 2 αναπαριστά μια τυχαία skip-list που περιέχει τα ίδια κλειδιά με εκεινα της απλής λίστας του Σχήματος 1. Επειδή αυτή η νέα δομή ουσιαστικά περιέχει συνδεδεμένες λίστες με επιπλέον δείκτες που παρακάμπτουν ενδιάμεσους κόμβους, ονομάστηκε Skip-List . Ο κάθε κόμβος έχει πάντα *MaxLevel* forward pointers οι οποίοι όμως δεν χρησιμοποιούνται όλοι για να δείξουν σε επόμενους κόμβους. Ο αριθμός των forward pointers που δείχνουν σε επόμενους κόμβους αποφασίζεται τυχαία σύμφωνα με τον αλγόριθμο εισαγωγής που παρουσιάζεται στη συνέχεια. Στο προγραμμά σας αυτό θα γίνεται με την χρήση των συναρτήσεων *srand(seed)/rand()*. Οι μόνες εξαιρέσεις είναι ο πρώτος κόμβος (header) και ο τελευταίος. Ο header έχει πάντα *MaxLevel* pointers που είτε δείχνουν στον τελευταίο κόμβο είτε σε κάποιο ενδιάμεσο κόμβο. Οι forward pointers του τελευταίου κόμβου είναι όλοι NULL.



Σχήμα 2: Skip-List

Βασικές έννοιες της Skip-List

- Ο κάθε κόμβος της skip-list περιέχει υποχρεωτικά ένα πεδίο κλειδί που τον χαρακτηρίζει μοναδικά, ένα δείκτη προς τα δεδομένα που θέλουμε να αποθηκεύσουμε καθώς επίσης και *MaxLevel* forward pointers. Το *MaxLevel* είναι μια σταθερά που εμείς ορίζουμε. Στο Σχήμα 2 αναπαριστούμε τους κόμβους χωρίς τα αποθηκευμένα δεδομένα τους. Επίσης, το Σχήμα 2 αναπαριστά σε κάθε κόμβο **μόνο** τους forward pointers που χρησιμοποιούνται για να δείξουν σε επόμενους κόμβους.
- Ο **header** της skip-list είναι ένας κόμβος που περιέχει *MaxLevel* forward pointers που αντιστοιχούν στα επίπεδα από μηδέν έως *MaxLevel-1* .

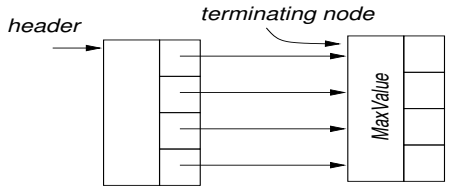
- Όταν εισάγουμε ένα νέο κόμβο στη λίστα, επιλέγουμε τυχαία το επίπεδο στο οποίο θα εισαχθεί ανεξάρτητα από τα δεδομένα που περιέχει. Ο κόμβος που βρίσκεται στο επίπεδο i περιέχει i forward pointers που δείχνουν προς επόμενους κόμβους της λίστας. Οι υπόλοιποι *MaxLevel-i* pointers είτε δείχνουν στον τελευταίο κόμβο είτε σε NULL (μιας και οι εν λόγω pointers δεν επισκεπτονται ποτέ).

Αρχικοποίηση της Skip-List

Ο κάθε κόμβος της skip-list μπορει να ειναι είναι ένα *struct* πιθανώς με την μορφη:
struct node {
int key;
*record *ptr;*
struct node forward[MaxLevel];*
}

Η skip-list αρχικοποιείται με την εντολή του κέλυφους του προγράμματος "initialize MaxNumOfPointers MaxValue" . Ο αριθμός MaxNumOfPointers καθορίζει την τιμή του *MaxLevel* και η MaxValue μια μέγιστη τιμή την οποία δεν θα πάρουν **ποτέ** τα κλειδιά της λίστας.

Αρχικά η λίστα περιέχει μόνο δύο κόμβους: τον header και ένα κόμβο τερματισμού (terminating node). Ο κόμβος του header χρησιμοποιείται μόνο για να έχουμε πρόσβαση στους forward pointers . Δεν μας ενδιαφέρει ούτε ασχολούμαστε ποτέ με την τιμή του κλειδιού του header . Ο κόμβος τερματισμού περιέχει ένα πεδίο κλειδί του οποίου η τιμή είναι ίση με MaxValue . Οπως αναφεραμε, η MaxValue είναι μια τιμή μεγαλύτερη από οποιαδήποτε τιμή μπορούν να πάρουν τα κλειδιά της skip-list την οποία ορίζουμε με την βοήθεια της εντολής initialize. Οι forward pointers του κόμβου τερματισμού είναι όλοι NULL. Αρχικά, όλοι οι forward pointers του header δείχνουν στον κόμβο τερματισμού οπως δειχνει το Σχημα 3.



Σχήμα 3: Αδεια Skip-list

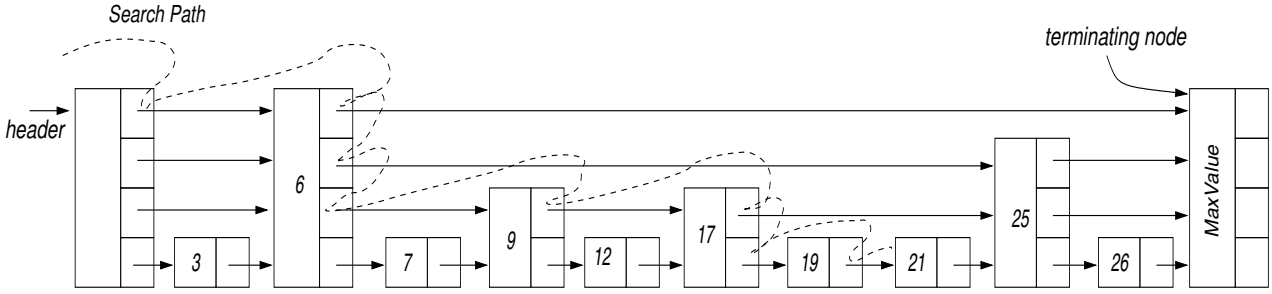
Αλγόριθμος αναζήτησης

Ο αλγόριθμος αναζήτησης επιστρέφει τα περιεχόμενα του κόμβου ο οποίος χαρακτηρίζεται από το επιθυμητό πεδίο κλειδί (ή τίποτα στην περίπτωση που το ζητούμενο κλειδί δεν υπάρχει στη λίστα). Ο αλγόριθμος αναζήτησης αναπαριστάται στον Αλγοριθμο 1. Η αναζήτηση ξεκινά από το επίπεδο *MaxLevel-1* της λίστας. Όταν δεν βρεθεί ο ζητούμενος κόμβος σε αυτό το επίπεδο, τότε η αναζήτηση προχωρά στο αμέσως χαμηλότερο επίπεδο μέχρι να φτάσουμε στο επίπεδο μηδέν. Η μεταβλητή 'x' που εμφανίζεται στους παρακάτω αλγορίθμους αποτελεί ένα δείκτη στη δομή (struct) που περιγράφει ένα κόμβο.

Στο Σχημα 4 βλέπετε την πορεία για την ανεύρεση του κόμβου που το κλειδί του είναι το 21.

Algorithm 1 $Search(header, searchKey)$

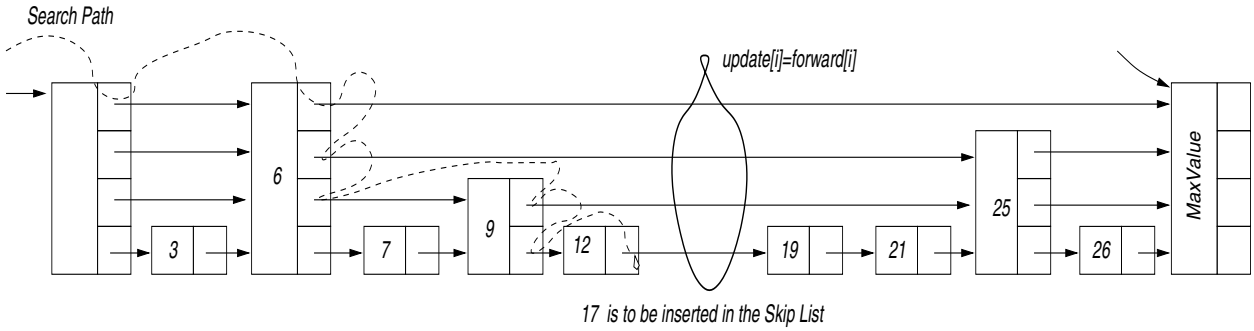
```
1: x := header
2: for i = MaxLevel − 1 downto 0 do
3:    while x → forward[i] → key < searchKey do
4:       x := x → forward[i]
5:    end while
6: end for
7: x := x → forward[0]
8: if x → key = searchKey then
9:    return x → value
10: else
11:    return failure
12: end if
```



Σχήμα 4: Μονοπατι για επερωτηση 'βρες το κομβο με κλειδι 21' στην skip-list

Αλγόριθμος Εισαγωγής

Ο αλγόριθμος εισαγωγής βρίσκει την σωστή τοποθεσία που πρεπει να εισαχθεί ενα κλειδί (και τα σχετικά δεδομένα) και εισάγει ένα νέο κόμβο με αυτό το κλειδί αν δεν υπάρχει ήδη στην skip-list. . Η εισαγωγή κλειδιών (και αντιστοίχων δεδομένων) δίνεται από τον Αλγόριθμο 2. Ένα παράδειγμα εισαγωγής εγγραφής με κλειδί 17 δίνεται στο Σχήμα 5. Προφανώς, όταν η εισαγωγή της εγγραφής με κλειδί 17 ολοκληρωθεί και αν ο



Σχήμα 5: Παράδειγμα εισαγωγής κόμβου με το κλειδι 17

κόμβος τυγχάνει να αποκτήσει δύο forwarding pointers τότε η skip-list γίνεται όπως εκείνη του Σχήματος 4.

Το επίπεδο στο οποίο γίνεται μια εισαγωγή καθορίζεται χρησιμοποιώντας τη γεννήτρια τυχαίων αριθμών *randomLevel()* (που θα την υλοποιησετε με την βοηθεια των κλησεων srand(seed)/rand()). Να σημειωθεί ότι η

Algorithm 2 $Insert(header, searchKey, newValue)$

```
1: local update[0..MaxLevel-1]
2: x := header
3: for i = MaxLevel − 1 downto 0 do
4:    while x → forward[i] → key < searchKey do
5:       x := x → forward[i]
6:    end while
7:    update[i] := x
8: end for
9: x := x → forward[0]
10: if x → key = searchKey then
11:    x → value := newValue
12: else
13:    lvl := randomLevel()
14:    x := makeNode(lvl, searchKey, value)
15:    for i = 0 to lvl do
16:       x → forward[i] := update[i] → forward[i]
17:       update[i] → forward[i] := x
18:    end for
19: end if
```

γεννήτρια τυχαίων αριθμών πρέπει να παράγει αριθμούς στο διάστημα από *0* έως και *MaxLevel-1* . Ο νέος κόμβος που δημιουργείται με τη συνάρτηση *makeNode(lvl, searchKey, value)* , περιέχει το κλειδί *searchKey*, τα δεδομένα value και τοποθετείται στο επίπεδο *lvl* . Όλοι οι forward pointers του νέου κόμβου δείχνουν στον τερματικό κόμβο αμέσως μετά την ολοκλήρωση τη συνάρτησης *makeNode()*. Στη συνέχεια του αλγορίθμου εισαγωγής, το for loop είναι υπεύθυνο για το 'ταίριασμα' του νέου κόμβου μέσα στη λίστα. Να σημειωθεί ότι η τοπική μεταβλητή 'update' είναι ένας μονοδιάστατος πίνακας που περιέχει MaxLevel δείκτες σε δομές (struct) τύπου κόμβου.

Αλγόριθμος Διαγραφής

Ο αλγόριθμος διαγραφής διαγράφει τα δεδομένα που αντιστοιχούν σε ένα κλειδί καθώς και το ίδιο το κλειδί και δίνεται στο Αλγόριθμο 3. Η συνάρτηση *free()* απελευθερώνει τη μνήμη που δεσμεύτηκε από τη συνάρτηση

Algorithm 3 $Delete(header, searchKey)$

```
1:  local update[0..MaxLevel-1]
2:  x := header
3:  for i = MaxLevel − 1 downto 0 do
4:     while x → forward[i] → key < searchKey do
5:        x := x → forward[i]
6:     end while
7:     update[i] := x
8:  end for
9:  x := x → forward[0]
10: if x → key = searchKey then
11:    for i = 0 to MaxLevel − 1 do
12:       if update[i] → forward[i] ≠ x then
13:          break
14:       end if
15:       update[i] → forward[i] := x → forward[i]
16:    end for
17:    free(x)
18: end if
```

makeNode().