ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

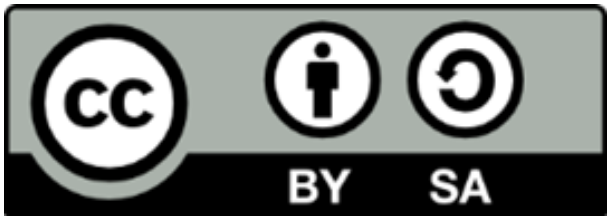
Ενότητα 6α: Αναζήτηση

Μαρία Σατρατζέμη

Τμήμα Εφαρμοσμένης Πληροφορικής

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

1. **Σειριακή αναζήτηση**
2. **Δυαδική Αναζήτηση**

Παραδοχή

Στη συνέχεια των διαφανειών (διαλέξεων) η ασυμπτωτική έκφραση (συμβολισμός *O*, *Ω*, *Θ*) του χρόνου εκτέλεσης ενός αλγόριθμου θα αποδίδεται με τον όρο (χρονική) πολυπλοκότητα ενός αλγόριθμου.

- **Πολυπλοκότητα Χειρότερης περίπτωσης:**

  T(n) = μέγιστος χρόνος εκτέλεσης του αλγορίθμου για οποιαδήποτε είσοδο μεγέθους n.

- **Πολυπλοκότητα Μέσης περίπτωσης:**

  T(n) = αναμενόμενος χρόνος εκτέλεσης αλγορίθμου επί όλων των εισόδων μεγέθους n.

Αναζήτηση

Το πρόβλημα της αποτελεσματικής αναζήτησης είναι σημαντικό καθώς εμφανίζεται σε πλήθος θεωρητικών και πρακτικών προβλημάτων.

Στην ενότητα αυτή συνοψίζονται δεδομένα αποτελέσματα αλλά παρουσιάζονται κατά αναλυτικότερο τρόπο.

Σειριακή Αναζήτηση

"Από πού να ξεκινήσω, παρακαλώ Μεγαλειότατε;" ρώτησε,
"Ξεκίνα από την αρχή," είπε ο Βασιλιάς με σοβαρότητα,
"και συνέχισε μέχρι να φτάσεις στο τέλος: τότε σταμάτα."

Lewis Caroll, Alice's Adventures in Wonderland

Σειριακή Αναζήτηση

- Η απλούστερη μέθοδος αναζήτησης είναι η **σειριακή** (*sequential*) ή αλλιώς **γραμμική** (*linear search*).
- Η επόμενη διαδικασία Sequential1 υποθέτει ότι
  - αναζητείται η τιμή *key* στον πίνακα *A* που περιέχει *n* αταξινόμητα στοιχεία, και
  - επιστρέφει τη θέση του κλειδιού στον πίνακα ή την τιμή -1 αν το κλειδί δεν υπάρχει (περίπτωση ανεπιτυχούς αναζήτησης).

Αλγόριθμος σειριακής αναζήτησης

Algorithm Sequential1(key)

1. $i = 1$
2. **while** ($i \leq n$) do
3. **if** ($A[i] == key$) then **return** i
4. **else** $i = i + 1$
5. **return** -1

Βελτίωση με κόμβο φρουρό

- Με την **τεχνική του κόμβου φρουρού** (*sentinel*), η μέθοδος θα υλοποιηθεί θεωρώντας μια ακόμη θέση στον πίνακα: $A[n + 1] \leftarrow key$
- Αν και πρακτικά η διαδικασία θα βελτιωθεί, σε θεωρητικό επίπεδο σε κάθε περίπτωση θα ισχύουν οι επόμενες προτάσεις υποθέτοντας ότι η πιθανότητα αναζήτησης του κλειδιού *A[i]* είναι $P_i = 1/n$, για $1 \leq i \leq n$ *(δηλαδή η πιθανότητα το κλειδί να βρίσκεται σε οποιαδήποτε θέση είναι ίδια)*

Πρόταση 1. *Η επιτυχής αναζήτηση σε πίνακα με n αταξινόμητα κλειδιά έχει πολυπλοκότητα Θ*(1), *Θ(n) και Θ(n) στην καλύτερη, στη χειρότερη και στη μέση περίπτωση, αντίστοιχα.*

Απόδειξη.

- Η καλύτερη και η χειρότερη περίπτωση συμβαίνουν όταν η αναζήτηση τερματίζεται με την εξέταση της πρώτης και της τελευταίας θέσης του πίνακα, αντίστοιχα. Αρα *Θ*(1) και *Θ*(*n*) αντίστοιχα καθώς θα κάνει 1 σύγκριση ή *n*.
- Για τη μέση περίπτωση θεωρώντας ότι η πιθανότητα αναζήτησης του κλειδιού *A[i]* είναι $P_i = 1/n$, για $1 \le i \le n$, άρα η εντολή {3} μπορεί να εκτελεστεί 1 ή 2 ή 3 ή …*n* φορές αν το κλειδί βρίσκεται αντίστοιχα στην 1η ή στη 2η … ή στην *n*-οστή θέση και όλες οι θέσεις να βρεθεί το κλειδί είναι με ίση πιθανότητα *1/n* :

$$\frac{1+2+\ldots+n}{n} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n(n+1)}{2n} = \frac{n+1}{2} \in \Theta(n)$$

από όπου προκύπτει η αλήθεια της πρότασης.

Πρόταση 2. *Η ανεπιτυχής αναζήτηση σε πίνακα με n αταξινόμητα κλειδιά έχει πολυπλοκότητα Θ(n) στην καλύτερη, στη χειρότερη και στη μέση περίπτωση.*

Απόδειξη.

- Σε κάθε περίπτωση θα σαρωθεί ολόκληρος ο πίνακας και άρα η πρόταση ισχύει.
- Αν υποθέσουμε ότι ο πίνακας περιέχει *n ταξινομημένα* στοιχεία, τότε για την επιτυχή αναζήτηση ισχύει η ανωτέρω πρόταση αλλά για να επιταχύνουμε την ανεπιτυχή αναζήτηση πρέπει να μετατρέψουμε τη διαδικασία Sequential1 ως εξής.

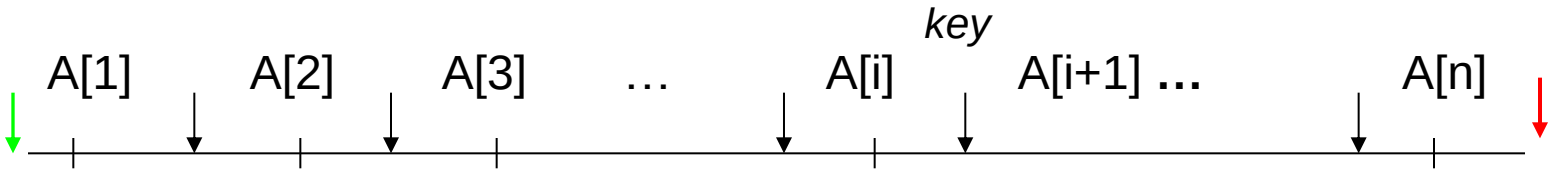
Αλγόριθμος σειριακής αναζήτησης για ταξινομημένους πίνακες

Algorithm Sequential2(key)

1. $i = 1$
2. **while** ($i \leq n$) do
3. **if** ($A[i] < key$) then $i = i + 1$
4. **else** if ($A[i] == key$) then **return** i
5. **else return** -1

Πρόταση 3. *Η ανεπιτυχής αναζήτηση σε πίνακα με n ταξινομημένα κλειδιά έχει πολυπλοκότητα Θ(1), Θ(n) και Θ(n) στην καλύτερη, στη χειρότερη και στη μέση περίπτωση, αντίστοιχα.*

Απόδειξη. Όταν αναζητούμε ένα μη υπαρκτό κλειδί, τότε αυτό μπορεί να ανήκει σε ένα από *n* + 1 μεσοδιαστήματα που (σχηματικά) δημιουργούνται από τις τιμές των στοιχείων του πίνακα επάνω στην ευθεία των ακεραίων.



- Η **καλύτερη** περίπτωση όταν το αναζητούμενο *key* είναι μικρότερο από το πρώτο στοιχείο του πίνακα και έχουμε *Θ*(1).
- Η **χειρότερη** περίπτωση όταν το αναζητούμενο *key* είναι μεγαλύτερο από το τελευταίο στοιχείο και έχουμε Θ(*n*).

- Για τη **μέση περίπτωση** θα πρέπει να εξετάσουμε *n* + 1 περιπτώσεις και να λάβουμε το μέσο όρο τους.
  - Αν το *key* είναι μικρότερο από το πρώτο στοιχείο, τότε αρκεί μία σύγκριση για να τερματισθεί η διαδικασία.
  - Αν το *key* είναι μεγαλύτερο από το *i*-οστό και μικρότερο από το (*i* + 1)-οστό στοιχείο (για i < *n),* τότε αρκούν *i* + 1 συγκρίσεις.

A[i]<key< A[i+1]	1	2	3	…	*i*	*i+1*	…	*n-1*	*n*
συγκρίσεις	2	3	4	…	*i*+1	*i*+2	…	*n*	*n*

  - Αν το *key είναι* μεγαλύτερο από το *n*-οστό στοιχείο, τότε αρκούν *n* συγκρίσεις. Συνεπώς ισχύει:

$$\frac{1+(2+\ldots+n)+n}{n}=\frac{(1+2+\ldots+n)+n}{n}=\frac{n(n+1)/2}{n}+\frac{n}{n}$$

$$=\frac{n(n+1)}{2n}+\frac{n}{n}=\frac{n+1}{2}+1=\frac{n+3}{2}\in\Theta(n)$$

από όπου προκύπτει η αλήθεια της πρότασης

Δυαδική Αναζήτηση

- Κλασικό παράδειγμα των αλγορίθμων της οικογενείας Διαίρει και Βασίλευε είναι η **δυαδική αναζήτηση** (binary search). Όπως γνωρίζουμε η αναζήτηση αυτή εφαρμόζεται σε πίνακες που περιέχουν ταξινομημένα στοιχεία.
- Υπενθυμίζοντας ότι η αποτελεσματικότητα των επαναληπτικών μεθόδων είναι καλύτερη των αναδρομικών (το κόστος κλήσης κάθε υποπρογράμματος δεν είναι αμελητέο), στη συνέχεια παρουσιάζουμε την *επαναληπτική διαδικασία* **Binary_Iterate** και την *αναδρομική διαδικασία* **Binary_Rec** που δείχνουν την ίδια θεωρητική συμπεριφορά.

Επαναληπτικός αλγόριθμος δυαδικής αναζήτησης

Algorithm *Binary_Iterate(key)*

1. *bottom* = 1; *top* = *n*;
2. **while** *(bottom* $\leq$ *top)* do
3. $middle = \left\lfloor \frac{top + bottom}{2} \right\rfloor$
4. **if** (*A[middle]* == *key)* then return *middle*
5. **else if** (*A[middle]* > *key)* then *top* = *middle* - 1
6. **else** *bottom* = *middle* + 1
7. **return** -1

Αναδρομικός αλγόριθμος δυαδικής αναζήτησης

Algorithm Binary_Rec(key, left, right)

1. **if** (*left* > *right)* then return -1;
2. $middle = \left\lfloor \frac{left + right}{2} \right\rfloor$
3. **if** (*A*[*middle*] == *key)* then **return** *middle*
4. **else if** (*A*[*middle*] > *key)* then
5. Binary_Rec(*key*, *left*, *middle* - 1)
6. **else** Binary_Rec(*key*, *middle* + 1, *right*)

Μπορούμε να περιγράψουμε τη λογική των προηγούμενων διαδικασιών ως εξής.

- Έστω ότι αναζητούμε το ακέραιο κλειδί *key* σε ένα πίνακα ***A***[1 ... *n*] με ταξινομημένους ακεραίους αριθμούς.
- Συγκρίνουμε το κλειδί *key* με το περιεχόμενο της μεσαίας θέσης του πίνακα *A,* που είναι η θέση *middle.*
- Στο σημείο αυτό τρία ενδεχόμενα μπορεί να συμβούν:
  - τα δύο στοιχεία είναι ίσα, οπότε ο σκοπός μας επιτεύχθηκε,
  - το *key* είναι μικρότερο από το *A*[*middle*], οπότε είμαστε βέβαιοι ότι το *key* αποκλείεται να βρίσκεται στον υποπίνακα *A*[*middle* ... *top*]. Έτσι συνεχίζουμε στον υποπίνακα A[*bottom* ... *middle* - 1] εξετάζοντας το μεσαίο στοιχείο του,
  - το *key* είναι μεγαλύτερο του *A*[*middle*], οπότε το *key* σαφώς δεν βρίσκεται στον υποπίνακα *A*[*bottom* ... *middle*). Έτσι συνεχίζουμε και πάλι εξετάζοντας το μεσαίο στοιχείο του υποπίνακα *A*[*middle* + 1 ... *top*]

Πρόταση 4. *Η πολυπλοκότητα της δυαδικής αναζήτησης στη χειρότερη περίπτωση είναι λογαριθμική.*

Απόδειξη. Κάθε φορά που η σύγκριση του *key* με το μεσαίο στοιχείο του πίνακα δεν καταλήγει σε ισότητα, η σύγκριση επαναλαμβάνεται σε υποπίνακα μισού μεγέθους σε σχέση με το μέγεθος του αρχικού.

Η αναδρομική σχέση που εκφράζει το χειρότερο κόστος του αλγορίθμου ο οποίος διαιρεί σε 3 μέρη το πρόβλημα μπορεί να σχηματισθεί ως εξής, αν θεωρήσουμε με $g(n)$ το κόστος για να διαιρέσουμε το πρόβλημα σε 3 μέρη:

$T(n)$= $g(n)$ + το κόστος για να λύσουμε το 1ο μέρος + το κόστος για να λύσουμε το 2ο μέρος + το κόστος για να λύσουμε το 3ο μέρος

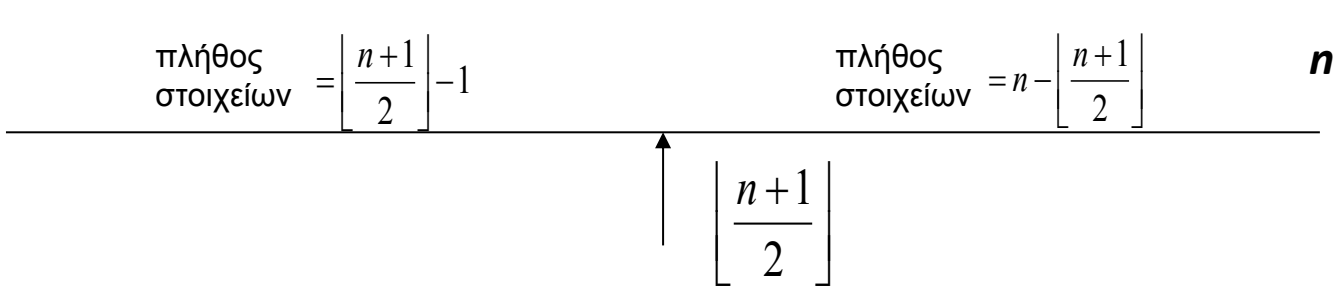
Καθώς $g(n)$=1 (σύγκριση με το μεσαίο στοιχείο) και τα 3 μέρη της εργασίας είναι ξένα μεταξύ τους, και το 2ο μέρος δεν έχει επιπλέον κόστος (περίπτωση το key ίσο με το μεσαίο στοιχείο) τότε έχουμε την εξής αναδρομική σχέση

$T(n)$=1 + max{T(*μέγεθος της αριστερής υπολίστας*), 0, T(*μέγεθος της δεξιάς υπολίστας*)}

$$T(n) = 1 + \max\left\{T\left(\left\lfloor (n+1)/2 \right\rfloor - 1\right), T\left(n - \left\lfloor (n+1)/2 \right\rfloor\right)\right\}$$

$T(n)=1+\max\{T(\textit{μέγεθος της αριστερής υπολίστας}),\ 0,$
$T(\textit{μέγεθος της δεξιάς υπολίστας})\}$ (1)

Στην πρώτη εξέταση δυαδικής αναζήτησης έχουμε



Μέγεθος της αριστερή υπολίστας $=\left\lfloor\frac{n+1}{2}\right\rfloor-1$

Μέγεθος της δεξιάς υπολίστας $=n-\left\lfloor\frac{n+1}{2}\right\rfloor$

(*το στοιχείο μεσαίο στοιχείο δεν είναι στοιχείο ούτε της αριστερής ούτε της δεξιάς υπολίστας*)

Αρα η (1) γίνεται:

$$T(n)=1+\max\{T(\lfloor(n+1)/2\rfloor-1),T(n-\lfloor(n+1)/2\rfloor)\}$$

Καθώς η δεξιά υπολίστα είναι πάντα τουλάχιστον τόσο μεγάλη όσο και η αριστερή, άρα:

$$\max\{T(\lfloor(n+1)/2\rfloor-1),T(n-\lfloor(n+1)/2\rfloor)\}=T(\lfloor(n+1)/2\rfloor-1)$$

$$T(0)=0$$

$$T(n)=1 \qquad \text{αν } key = A[middle]$$

$$T(n)=1+T(\lfloor (n+1)/2 \rfloor -1) \qquad \text{αν } key < A[middle]$$

$$T(n)=1+T(n-\lfloor (n+1)/2 \rfloor) \qquad \text{αν } key > A[middle]$$

$n = 2^k - 1$
$n + 1 = 2^k$
$\log_2(n+1) = \log_2 2^k$
$\log_2(n+1) = k \log_2 2$
$\log_2(n+1) = k$

Απλοποιούμε τη σχέση αυτή θεωρώντας τη χειρότερη περίπτωση (δηλαδή, αγνοούμε το δεύτερο σκέλος) και επίσης υποθέτουμε ότι n = 2^k - 1 (*n* περιττός) για κάποιο ακέραιο αριθμό *k*. Έτσι προκύπτει

$$n = 2^k - 1 \Rightarrow n+1 = 2^k \Rightarrow \left\lfloor \frac{n+1}{2} \right\rfloor = \left\lfloor \frac{2^k}{2} \right\rfloor = \lfloor 2^{k-1} \rfloor = 2^{k-1}$$

$$T(n)=1+T(\lfloor (n+1)/2 \rfloor -1) \Rightarrow T(2^k-1)=1+T(2^{k-1}-1)$$

$$n = 2^k - 1 \Rightarrow n+1 = 2^k \Rightarrow \left\lfloor \frac{n+1}{2} \right\rfloor = \left\lfloor \frac{2^k}{2} \right\rfloor = \lfloor 2^{k-1} \rfloor = 2^{k-1}$$

$$T(n) = 1 + T(\lfloor (n+1)/2 \rfloor - 1) \Rightarrow T(n) = 1 + T(2^{k-1} - 1)$$

$n = 2^k - 1$
$n + 1 = 2^k$
$\log_2(n+1) = \log_2 2^k$
$\log_2(n+1) = k \log_2 2$
$\log_2(n+1) = k$

Με αρχική $T(0)=0$. Εχουμε (με επαναλήψεις της αναδρομής):

$$\begin{aligned}
T(n) &= 1 + T(2^{k-1} - 1) = \\
&= 1 + (1 + T(2^{k-2} - 1)) = \\
&= 1 + (1 + (1 + T(2^{k-3} - 1))) = \\
&= .... \\
&= i + T(2^{k-i} - 1) = \\
&= ... \\
&= k + T(2^{k-k} - 1) = k + T(2^0 - 1) = \\
&= k + T(1-1) = k + T(0) = k + 0 = \log_2(n+1) = \Theta(\log_2 n)
\end{aligned}$$

Ετσι λοιπόν για $n = 2^k - 1$ προκύπτει ότι η πολυπλοκότητα της δυαδικής αναζήτησης είναι λογαριθμική $\Theta(\log_2 n)$.

$\log_2(n+1)=\Theta(\log_2 n)$ (δες άσκηση 2.3-6, απόδειξη με κανόνα του ορίου)

- Με την Πρόταση 4 αποδείξαμε ότι η πολυπλοκότητα της χειρότερης περίπτωσης της δυαδικής αναζήτησης είναι της τάξης $\Theta(\log_2(n))$.
- Αυτήν την απάντηση την αναμέναμε εφόσον ο αλγόριθμος υποδιπλασιάζει περίπου το μέγεθος του εναπομείναντα πίνακα σε κάθε επανάληψη, ο αριθμός των επαναλήψεων που απαιτούνται για την αναγωγή του αρχικού μεγέθους n στο τελικό 1, είναι περίπου $\log_2 n$.
- Η λογαριθμική συνάρτηση αυξάνει τόσο αργά που οι τιμές της εξακολουθούν να παραμένουν μικρές ακόμη και για πολύ μεγάλες τιμές του n. Για την περίπτωση με n =1000 τότε $\log_2(10^3+1)$=10 συγκρίσεις.

- Μια ανάλυση της μέσης περίπτωσης επίσης θα δείξει ότι ο μέσος συγκρίσεων είναι ελαφρά μόνο καλύτερος από την χειρότερη περίπτωση.
- Οι ακριβείς τύποι για το μέσον όρο των συγκρίσεων για επιτυχή και ανεπιτυχή αναζήτηση είναι $\log_2 n$ -1 και $\log_2(n+1)$ αντίστοιχα.
- Αν και η δυαδική αναζήτηση είναι ένας βέλτιστος αλγόριθμος αναζήτησης αν και περιοριζόμαστε στις επιτελούμενες πράξεις μόνο στις συγκρίσεις μεταξύ των κλειδιών, παρ' όλα αυτά υπάρχουν αλγόριθμοι που επιδεικνύουν καλύτερη αποδοτικότητα μέσης περίπτωσης όπως ο κατακερματισμός για παράδειγμα που δεν απαιτεί ο πίνακας να είναι ταξινομημένος.

- Ένα τελευταίο σχόλιο είναι ότι ο αλγόριθμος της δυαδικής αναζήτησης παρουσιάζεται ως αλγόριθμος της κατηγορίας διαίρει και βασίλευε.
- Ο αλγόριθμος της δυαδικής αναζήτησης είναι μια κατεξοχήν μη τυπική περίπτωση της κατηγορίας διαίρει και βασίλευε.
- Η τεχνική του διαίρει και βασίλευε διαιρεί ένα πρόβλημα σε μικρότερα υποπροβλήματα που το καθένα απαιτεί λύση.
- Αυτό δε συνάδει με τη δυαδική αναζήτηση όπου μόνο ένα από τα δυο προκύπτοντα υποπροβλήματα απαιτεί λύση.
- Αρα ο αλγόριθμος της δυαδικής αναζήτησης μπορεί να θεωρηθεί ως εκφυλισμένη περίπτωση της τεχνικής διαίρει και βασίλευε.

Τέλος Ενότητας