ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

Μείωση και κυριαρχία

Κωνσταντίνος Γιαννουτάκης

Επίκουρος Καθηγητής

ΣΥΝΟΨΗ ΔΙΑΛΕΞΗΣ

- Μείωση και Κυριαρχία

ΜΕΙΩΣΗ ΚΑΙ ΚΥΡΙΑΡΧΙΑ

ΜΕΙΩΣΗ ΚΑΙ ΚΥΡΙΑΡΧΙΑ (DECREASE AND CONQUER)

Η τεχνική της **μείωσης και κυριαρχίας** (decrease and conquer) *βασίζεται στην εκμετάλλευση της σχέσης που υφίσταται μεταξύ της λύσης μιας δεδομένης έκφανσης ενός προβλήματος και της λύσης μιας μικρότερης έκφανσης του ίδιου προβλήματος*

– Υλοποιήσεις από **πάνω προς τα κάτω** (top-down): **αναδρομικές υλοποιήσεις**
– Υλοποιήσεις από **κάτω προς τα πάνω** (bottom-up): **επαναληπτικές υλοποιήσεις**

ΜΕΙΩΣΗ ΚΑΙ ΚΥΡΙΑΡΧΙΑ (DECREASE AND CONQUER)

Τρεις **παραλλαγές** της τεχνικής μείωσης και κυριαρχίας

- μείωση κατά μία **σταθερά**
- μείωση κατά έναν **σταθερό παράγοντα**
- μείωση **μεταβλητού μεγέθους**

ΜΕΙΩΣΗ ΚΑΤΑ ΜΙΑ ΣΤΑΘΕΡΑ

- Η παραλλαγή της **μείωσης κατά μία σταθερά** (decrease by a constant) **ελαττώνει κατά την ίδια σταθερά το μέγεθος του προβλήματος σε κάθε επανάληψη του αλγορίθμου**
- Τυπικά η σταθερά αυτή ισούται με 1, αν και πιο σπάνια εμφανίζονται περιπτώσεις μεγαλύτερων μειώσεων

ΤΑΞΙΝΟΜΗΣΗ ΜΕ ΕΙΣΑΓΩΓΗ (INSERTION SORT)

Ένας από τους απλούστερους αλγόριθμους ταξινόμησης με την εξής λειτουργία:

- ξεκινάμε από τη δεύτερη θέση και αν το στοιχείο είναι μικρότερο από το πρώτο, τότε τα αντιμεταθέτουμε, αλλιώς δεν κάνουμε τίποτα
- έπειτα ξεκινάμε από την τρίτη θέση και το συγκρίνουμε με το δεύτερο στοιχείο και αν είναι μικρότερο, τότε το αντιμεταθέτουμε και συνεχίζουμε την σύγκριση με το πρώτο στοιχείο
- και συνεχίζει με αυτόν τον τρόπο μέχρι να ταξινομηθεί ολόκληρος ο πίνακας

ΤΑΞΙΝΟΜΗΣΗ ΜΕ ΕΙΣΑΓΩΓΗ (INSERTION SORT)

23	42	10	100	8	44	12
23	42	10	100	8	44	12
23	42	10	100	8	44	12
23	10	42	100	8	44	12
10	23	42	100	8	44	12
10	23	42	100	8	44	12
10	23	42	8	100	44	12
10	23	8	42	100	44	12

10	8	23	42	100	44	12
8	10	23	42	100	44	12
8	10	23	42	44	100	12
8	10	23	42	44	100	12
8	10	23	42	44	12	100
8	10	23	42	12	44	100
8	10	23	12	42	44	100
8	10	12	23	42	44	100

ΚΩΔΙΚΑΣ C

```c
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];

        j = i - 1;
        /* Μετακίνησε τα στοιχεία του arr[0..i-1], που είναι
          μεγαλύτερα από το key, μια θέση προς τα δεξιά */
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Πολυπλοκότητα

ΚΩΔΙΚΑΣ C

```c
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];

        j = i - 1;
        /* Μετακίνησε τα στοιχεία του arr[0..i-1], που είναι
          μεγαλύτερα από το key, μια θέση προς τα δεξιά */
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

n-1 επαναλήψεις

i επαναλήψεις

$$\sum_{i=1}^{n-1}\sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1}(i) = 1 + 2 + \cdots + n - 1$$

$$= \frac{n(n-1)}{2}$$

Άρα πολυπλοκότητα $\boldsymbol{\mathcal{O}(n^2)}$

Καλύτερη περίπτωση

ΜΕΙΩΣΗ ΚΑΤΑ ΣΤΑΘΕΡΟ ΠΑΡΑΓΟΝΤΑ

- Η παραλλαγή της μείωσης **κατά ένα σταθερό παράγοντα** (decrease by a constant factor) **ελαττώνει κατά τον ίδιο σταθερό παράγοντα του μεγέθους του προβλήματος σε κάθε επανάληψη του αλγορίθμου**
- Η μείωση κατά έναν παράγοντα διάφορο του 2 είναι εξαιρετικά σπάνια
- Οι τεχνικές αυτές αποφέρουν αλγόριθμους **λογαριθμικής τάξης**

ΔΥΑΔΙΚΗ ΑΝΑΖΗΤΗΣΗ (BINARY SEARCH)

Ένας από τους γρήγορους αλγόριθμους αναζήτησης ενός κλειδιού σε ταξινομημένη λίστα:

- Συγκρίνουμε την τιμή του κλειδιού με την τιμή του μεσαίου στοιχείου της λίστας
- Εάν είναι ίσα, ο αλγόριθμος τερματίζει
- Διαφορετικά, επαναλαμβάνεται αναδρομικά η ίδια διαδικασία για το πρώτο μισό της λίστας (εάν η τιμή του κλειδιού είναι μικρότερη) ή για το δεύτερο μισό της λίστας (εάν η τιμή του κλειδιού είναι μεγαλύτερη)

ΔΥΑΔΙΚΗ ΑΝΑΖΗΤΗΣΗ

Στοιχείο προς αναζήτηση
K=70

1	3	4	8	9	13	15	20	24	38	40	55	67	68	70	79

K>20

1	3	4	8	9	13	15	20	24	38	40	55	67	68	70	79

K>55

1	3	4	8	9	13	15	20	24	38	40	55	67	68	70	79

K>68

1	3	4	8	9	13	15	20	24	38	40	55	67	68	70	79

K=70

ΚΩΔΙΚΑΣ C

```c
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;    // Ισοδύναμο με (l+r)/2
        // Εάν το στοιχείο βρίσκεται στη μέση
        if (arr[mid] == x)
            return mid;

        // Εάν το στοιχείο είναι μικρότερο από το mid
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        // Διαφορετικά, το στοιχείο μπορεί να βρίσκεται στον δεξιό υποπίνακα
        return binarySearch(arr, mid + 1, r, x);
    }

    // Εάν δεν υπάρχει το στοιχείο στον πίνακα
    return -1;
}
```

Πολυπλοκότητα

ΚΩΔΙΚΑΣ C

```c
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;    // Ισοδύναμο με (l+r)/2
        // Εάν το στοιχείο βρίσκεται στη μέση
        if (arr[mid] == x)
            return mid;

        // Εάν το στοιχείο είναι μικρότερο από το mid
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        // Διαφορετικά, το στοιχείο μπορεί να βρίσκεται στον δεξιό υποπίνακα
        return binarySearch(arr, mid + 1, r, x);
    }

    // Εάν δεν υπάρχει το στοιχείο στον πίνακα
    return -1;
}
```

1 σύγκριση

1 σύγκριση

1 σύγκριση

Χρόνος εκτέλεσης

$$T(n) = \begin{cases} \mathcal{O}(1), & \alpha\nu\ n = 1 \\ T\left(\frac{n}{2}\right) + 3, & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$

Άρα πολυπλοκότητα $\boldsymbol{\mathcal{O}}(\mathbf{lg}\ \boldsymbol{n})$

Καλύτερη περίπτωση

ΜΕΤΑΒΛΗΤΗ ΜΕΙΩΣΗ ΜΕΓΕΘΟΥΣ

Στην παραλλαγή της **μείωσης μεταβλητού μεγέθους** (variable size decrease), **το πρότυπο μείωσης ποικίλλει από την μία επανάληψη του αλγορίθμου στην άλλη**

ΑΝΑΖΗΤΗΣΗ ΠΑΡΕΜΒΟΛΗΣ (INTERPOLATION SEARCH)

Η αναζήτηση Παρεμβολής είναι μια βελτίωση της Binary Search, όταν ο πίνακας είναι ταξινομημένος και οι τιμές κατανέμονται ομοιόμορφα. Η Binary Search πηγαίνει πρώτα στο μεσαίο στοιχείο και ελέγχει. Η Αναζήτηση Παρεμβολής μπορεί να πάει σε διαφορετικά σημεία του πίνακα ανάλογα με την τιμή του στοιχείου αναζήτησης.

- Για παράδειγμα, εάν η τιμή του κλειδιού είναι πιο κοντά στο τελευταίο στοιχείο του πίνακα, τότε η αναζήτηση παρεμβολής θα εκκινήσει από τα τελευταία στοιχεία του πίνακα.

ΑΝΑΖΗΤΗΣΗ ΠΑΡΕΜΒΟΛΗΣ (INTERPOLATION SEARCH)

- Έστω ότι ο πίνακας T είναι ταξινομημένος σε αύξουσα διάταξη.
- Μοιάζει με την δυαδική αναζήτηση, αλλά αντί να χωρίζει τον πίνακα στα δύο, χρησιμοποιεί τον παρακάτω **τύπο για να βρει το νέο σημείο διαχωρισμού**

$$\boldsymbol{pos = l + \frac{h - l}{T[h] - T[l]}(x - T[l])}$$

- Δουλεύει καλά όταν τα στοιχεία του πίνακα είναι ομοιόμορφα κατανεμημένα γιατί πλησιάζει πλησιέστερα στο σημείο προς αναζήτηση από ότι η δυαδική αναζήτηση.

ΑΝΑΖΗΤΗΣΗ ΠΑΡΕΜΒΟΛΗΣ (INTERPOLATION SEARCH)

Βήμα 1: Μέσα σε έναν βρόχο, υπολογίστε την τιμή “pos” χρησιμοποιώντας τον προηγούμενο τύπο.

Βήμα 2: Εάν υπάρχει ταίριασμα του στοιχείου αναζήτησης και του στοιχείου pos στο πίνακα, επιστρέψτε το index του στοιχείου του πίνακα και έξοδος.

Βήμα 3: Εάν το στοιχείο αναζήτησης είναι μικρότερο από το στοιχείο του πίνακα arr[pos], υπολογίστε πάλι με βάση τον τύπο την τιμή “pos” για τον αριστερό υπο-πίνακα. Διαφορετικά, Υπολογίστε το ίδιο για το δεξιό υπο-πίνακα.

Βήμα 4: Επαναλάβετε μέχρι να μην υπάρχει ή να υπάρχει ταίριασμα.

ΑΝΑΖΗΤΗΣΗ ΠΑΡΕΜΒΟΛΗΣ

Στοιχείο προς αναζήτηση
K=18

l														h
10	12	13	16	18	19	20	21	22	23	24	33	35	42	47

l	h	Arr[l]	Arr[h]	pos
0	14	10	47	3

				l										h
10	12	13	16	18	19	20	21	22	23	24	33	35	42	47

l	h	Arr[l]	Arr[h]	pos
4	14	18	47	4

$$pos = l + \frac{h - l}{T[h] - T[l]}(x - T[l])$$

Στοιχείο προς αναζήτηση
K=70

l															h
1	3	4	8	9	13	15	20	24	38	40	55	67	68	70	79

l	h	Arr[l]	Arr[h]	pos
0	15	1	79	13

														l	h
1	3	4	8	9	13	15	20	24	38	40	55	67	68	70	79

l	h	Arr[l]	Arr[h]	pos
14	15	70	79	14

ΚΩΔΙΚΑΣ C

```c
int interpolation_search(int T[], int n, int a) {
    int l = 0, h = (n - 1);
    while (l <= h && a >= T[l] && a <= T[h]) {
        if (l == h) {
            if (T[l] == a) return l;
            return -1;
        }
        int pos = l + (((h - l) / (T[h] - T[l])) * (a - T[l])); /* εύρεση σημείο διαχωρισμού */
        if (T[pos] == a) /* Έλεγχος αν το στοιχείο είναι στην μέση του τμήματος αναζήτησης */
            return pos;
        else if (T[pos] < a) /* Αν το στοιχείο είναι μεγαλύτερο, δεξιό κομμάτι */
            l = pos + 1;
        else /* Αν το στοιχείο είναι μικρότερο, αγνόησε το δεξιό κομμάτι */
            h = pos - 1;
    }
    return -1;
}
```

ΠΟΛΥΠΛΟΚΟΤΗΤΑ

- Καλύτερη περίπτωση

$$\mathcal{O}(1)$$

- Χειρότερη περίπτωση

$$\mathcal{O}(n)$$

- Μέση περίπτωση

$$\mathcal{O}(\lg \lg n)$$

Κωνσταντίνος Γιαννουτάκης

Επίκ. Καθηγητής

kgiannou@uom.edu.gr

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ