Διάλεξη 17 - Δυαδική Αναζήτηση και Ταξινόμηση

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

Θανάσης Αυγερινός

Ανακοινώσεις / Διευκρινήσεις

- :)

Την προηγούμενη φορά

- Δεδομένα Εισόδου #2
  - Συναρτήσεις χειρισμού stdin (π.χ., scanf)
  - Αρχεία (Files) και ρεύματα δεδομένων (data streams)
  - Συναρτήσεις χειρισμού αρχείων
  - Παραδείγματα

Σήμερα

- Αλγόριθμοι Αναζήτησης (Search Algorithms)
  - Γραμμική Αναζήτηση (Linear Search)
  - Δυαδική Αναζήτηση (Binary Search)
- Αλγόριθμοι Ταξινόμησης (Sorting Algorithms)
- Παραδείγματα

Ένας Γρίφος

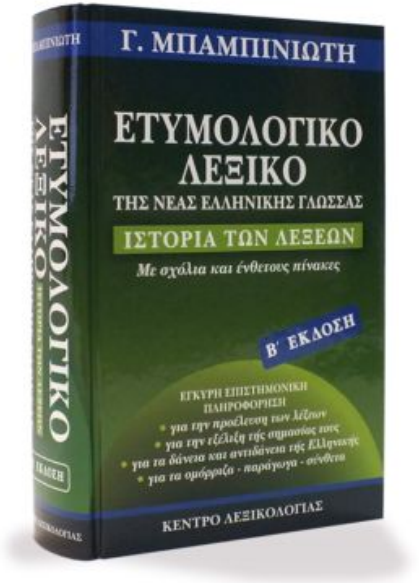
Έχω έναν αριθμό στο νου μου στο διάστημα $[1, 10^9]$

Θέλετε να τον βρείτε. Μπορείτε να με ρωτήσετε ότι ερώτηση θέλετε και εγώ απαντάω ναι ή όχι.

Τι θα με ρωτήσετε; Πόσες προσπάθειες χρειάζεστε για να βρείτε τον αριθμό μου;

Αναζήτηση Λήμματος σε Λεξικό

Αναζητώ την ετυμολογία μιας λέξης,
δεν διατρέχω όλο το λεξικό μέχρι να την βρω



Guess Who

Στόχος: βρες ποιον χαρακτήρα έχω επιλέξει με τις λιγότερες ερωτήσεις

Θέλω μια συνάρτηση που να δέχεται έναν πίνακα 100 ακεραίων και έναν ακέραιο και να γυρνάει την θέση του στοιχείου αν το βρήκε ή -1. Πως;

```
int find(int haystack[100], int needle) {
  int i;
  for(i = 0; i < 100; i++) {
      if (haystack[i] == needle) {
          return i;
      }
  }
  return -1;
}
```

Γραμμική / Σειριακή Αναζήτηση (Linear / Serial Search)

Η Γραμμική Αναζήτηση (Linear Search) είναι ένας αλγόριθμος αναζήτησης ενός στοιχείου σε μια ακολουθία στοιχείων.

Ο αλγόριθμος ξεκινά με το πρώτο στοιχείο της ακολουθίας και σε κάθε βήμα του συγκρίνει το στοιχείο μας με αυτό της ακολουθίας. Αν είναι ίσα, σταματάει, αλλιώς συνεχίζει στο επόμενο σειριακά μέχρι να φτάσει στο τελευταίο στοιχείο.

Η χρονική πολυπλοκότητα του αλγορίθμου είναι O(n) ενώ χρειάζεται σταθερό χώρο μνήμης O(1)

Μπορούμε να βρούμε ένα στοιχείο πιο γρήγορα από O(n);

Ταξινόμηση

Μια ακολουθία στοιχείων a_i λέγεται ταξινομημένη (sorted) ως προς κάποιο τελεστή σύγκρισης $\leq_\alpha$, αν και μόνο αν $\forall\, i \leq j.\ a_i \leq_\alpha a_j$

π.χ., η ακολουθία 1, 7, 8, 10, 19 είναι ταξινομημένη ως προς έναν τελεστή $\leq_\alpha$ εάν είναι ο τελεστή σύγκρισης ακεραίων: $a_i \leq_\alpha a_j \Leftrightarrow a_i \leq a_j$ (αύξουσα)

π.χ., η ακολουθία 19, 10, 8, 7, 1 είναι ταξινομημένη ως προς έναν τελεστή $\leq_\varphi$ που ορίζεται ως εξής: $a_i \leq_\varphi a_j \Leftrightarrow -a_i \leq -a_j \Leftrightarrow a_j \leq a_i$ (φθίνουσα)

Δυαδική Αναζήτηση (Binary Search)

Η Δυαδική Αναζήτηση (Binary Search) είναι ένας αλγόριθμος αναζήτησης που μας επιτρέπει να βρούμε αν ένα στοιχείο βρίσκεται σε μια ακολουθία ταξινομημένων στοιχείων.

Σε κάθε βήμα του αλγορίθμου, η ακολουθία χωρίζεται σε δύο τμήματα και συγκρίνουμε το στοιχείο μας με το μεσαίο στοιχείο της ακολουθίας. Ο αλγόριθμος συνεχίζει σε ένα από τα δύο τμήματα μέχρι να βρεθεί το στοιχείο ή να δείξουμε ότι το στοιχείο δεν μπορεί να βρίσκεται σε αυτό το τμήμα.

Θέλω μια συνάρτηση που να δέχεται έναν πίνακα n ταξινομημένων ακεραίων σε αύξουσα σειρά και έναν ακέραιο που ψάχνουμε και να γυρνάει αν υπάρχει στον πίνακα ή όχι. Πως;

Δυαδική Αναζήτηση (Binary Search)

```c
int binary_search(int elem, int *array, int n) {
  int mid, low = 0, high = n - 1;
  while (low <= high) {
    mid = (low + high) / 2;
    if (array[mid] == elem)
      return 1;
    else if (array[mid] < elem)
      low = mid + 1;
    else
      high = mid - 1;
  }
  return 0;
}
```

Παράδειγμα

Δυαδική Αναζήτηση (Binary Search)

```c
int binary_search(int elem, int *array, int n) {
  int mid, low = 0, high = n - 1;
  while (low <= high) {
    mid = (low + high) / 2;
    if (array[mid] == elem)
      return 1;
    else if (array[mid] < elem)
      low = mid + 1;
    else
      high = mid - 1;
  }
  return 0;
}
```

Αυτός ο αλγόριθμος έχει σφάλμα, μπορούμε να το βρούμε και να το διορθώσουμε;

Δυαδική Αναζήτηση (Binary Search)

```c
int binary_search(int elem, int *array, int n) {
  int mid, low = 0, high = n - 1;
  while (low <= high) {
    mid = low + (high - low) / 2;
    if (array[mid] == elem)
      return 1;
    else if (array[mid] < elem)
      low = mid + 1;
    else
      high = mid - 1;
  }
  return 0;
}
```

Ο αλγόριθμος θεωρείται ιδιαίτερα δύσκολος, ώστε να υλοποιηθεί σωστά

Δυαδική Αναζήτηση (Binary Search)

```c
int binary_search(int elem, int *array, int n) {
  int mid, low = 0, high = n - 1;
  while (low <= high) {
    mid = low + (high - low) / 2;
    if (array[mid] == elem)
      return 1;
    else if (array[mid] < elem)
      low = mid + 1;
    else
      high = mid - 1;
  }
  return 0;
}
```

Τι πολυπλοκότητα έχει αυτός ο αλγόριθμος;

Χρόνος: O(logn)
Χώρος: O(1)

Υλοποιημένη στην συνάρτηση `bsearch` της stdlib.h

Αλγόριθμοι Ταξινόμησης (Sorting Algorithms)

1. Bubblesort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quicksort

Γράψτε μια συνάρτηση swap που ανταλλάζει δύο ακεραίους

```c
#include <stdio.h>
int main() {
  int a = 100, b = 200;
  printf("%d %d\n", a, b);
  swap( ... );
  printf("%d %d\n", a, b);
  return 0;
}
```

Γράψτε μια συνάρτηση swap που ανταλλάζει δύο ακεραίους

```c
#include <stdio.h>

void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main() {
  int a = 100, b = 200;
  printf("%d %d\n", a, b);
  swap(&a, &b);
  printf("%d %d\n", a, b);
  return 0;
}
```

Ταξινόμηση Επιλογής (Selection Sort)

```
void selection_sort(int n, int *x) {
  int i, j, min;
  for (i = 1 ; i <= n - 1 ; i++) {
    min = i - 1;
    for (j = i ; j <= n - 1 ; j++)
      if (x[j] < x[min])
        min = j;
    swap(&x[i-1], &x[min]);
  }
}
```

Ταξινόμηση Επιλογής (Selection Sort)

```
void selection_sort(int n, int *x) {
  int i, j, min;
  for (i = 1 ; i <= n - 1 ; i++) {
    min = i - 1;
    for (j = i ; j <= n - 1 ; j++)
      if (x[j] < x[min])
        min = j;
    swap(&x[i-1], &x[min]);
  }
}
```

Χρόνος: $O(n^2)$
Χώρος: $O(1)$

Ταξινόμηση Εισαγωγής (Insertion Sort)

```
void insertion_sort(int n, int *x) {
  int i, j;
  for (i = 1 ; i <= n - 1 ; i++) {
    j = i - 1;
    while (j >= 0 && x[j] > x[j+1]) {
      swap(&x[j], &x[j+1]);
      j--;
    }
  }
}
```

Ταξινόμηση Εισαγωγής (Insertion Sort)

```
void insertion_sort(int n, int *x) {
  int i, j;
  for (i = 1 ; i <= n - 1 ; i++) {
    j = i - 1;
    while (j >= 0 && x[j] > x[j+1]) {
      swap(&x[j], &x[j+1]);
      j--;
    }
  }
}
```

Χρόνος: $O(n^2)$
Χώρος: $O(1)$

Ταξινόμηση Φυσαλίδας (Bubblesort)

```
void bubblesort(int n, int *x) {
  int i, j;
  for (i = 1 ; i <= n - 1 ; i++)
    for (j = n - 1 ; j >= i ; j--)
      if (x[j-1] > x[j])
        swap(&x[j-1], &x[j]);
}
```

Ταξινόμηση Φυσαλίδας (Bubblesort)

```
void bubblesort(int n, int *x) {
  int i, j;
  for (i = 1 ; i <= n - 1 ; i++)
    for (j = n - 1 ; j >= i ; j--)
      if (x[j-1] > x[j])
        swap(&x[j-1], &x[j]);
}
```

Χρόνος: $O(n^2)$
Χώρος: $O(1)$

Ταξινόμηση Συγχώνευσης (Merge Sort)

Η ταξινόμηση συγχώνευσης (merge sort) είναι ένας αλγόριθμος divide and conquer (διαίρει και βασίλευε) που έχει θεωρητικά την καλύτερη πολυπλοκότητα. Ο αλγόριθμος έχει δύο βήματα:

1. Χώρισε τον πίνακα σε δύο υποπίνακες
    a. κάλεσε ταξινόμηση συγχώνευσης στους υποπίνακες
2. Συγχώνευσε τα στοιχεία των δύο ταξινομημένων υποπινάκων

Ταξινόμηση Συγχώνευσης (Merge Sort)

```
void merge_sort(int *array, int left, int right) {
    if (left < right) {
        int middle = left + (right - left) / 2;
        merge_sort(array, left, middle);
        merge_sort(array, middle + 1, right);
        merge(array, left, middle, right);
    }
}
```

Ταξινόμηση Συγχώνευσης (Merge Sort)

```
void merge_sort(int *array, int left, int right) {
    if (left < right) {
        int middle = left + (right - left) / 2;
        merge_sort(array, left, middle);
        merge_sort(array, middle + 1, right);
        merge(array, left, middle, right);
    }
}
```

Ταξινόμηση Συγχώνευσης (Merge Sort)

```
void merge(int *x, int l, int m, int r) {
    int i, j, k, n1 = m - l + 1, n2 = r - m;
    int left[n1], right[n2];
    for (i = 0; i < n1; i++) left[i] = x[l + i];
    for (j = 0; j < n2; j++) right[j] = x[m + 1 + j];
    i = 0; j = 0; k = l;
    while (i < n1 && j < n2) {
        if (left[i] <= right[j]) x[k++] = left[i++];
        else x[k++] = right[j++];
    }
    while (i < n1) x[k++] = left[i++];
    while (j < n2) x[k++] = right[j++];
}
```

Ταξινόμηση Συγχώνευσης (Merge Sort)

```
void merge(int *x, int l, int m, int r) {
    int i, j, k, n1 = m - l + 1, n2 = r - m;
    int left[n1], right[n2];
    for (i = 0; i < n1; i++) left[i] = x[l + i];
    for (j = 0; j < n2; j++) right[j] = x[m + 1 + j];
    i = 0; j = 0; k = l;
    while (i < n1 && j < n2) {
        if (left[i] <= right[j]) x[k++] = left[i++];
        else x[k++] = right[j++];
    }
    while (i < n1) x[k++] = left[i++];
    while (j < n2) x[k++] = right[j++];
}
```

Χρόνος: O(nlogn)
Χώρος: O(n)

Ταχυταξινόμηση (Quicksort)

Η ταξινόμηση ταχυταξινόμηση (quicksort) είναι ένας αλγόριθμος divide and conquer (διαίρει και βασίλευε) που είναι ιδιαίτερα δημοφιλής. Ο αλγόριθμος έχει τρία βήματα:

1. Διάλεξε (έστω τυχαία) το στοιχείο διαμέρισης του πίνακα (pivot element)
2. Διαμέρισε τον πίνακα σε δύο υποπίνακες - αριστερά έχει τα στοιχεία που είναι μικρότερα του pivot και δεξιά τα στοιχεία που είναι μεγαλύτερα
3. Τρέξε ταχυταξινόμηση για τους δύο υποπίνακες

Ταχυταξινόμηση (Quicksort)

```
void quicksort (int *x, int lower, int upper) {
   if (lower < upper) {
        int pivot = x[(lower + upper) / 2];
        int i, j;
        for (i = lower, j = upper; i <= j;) {
          while (x[i] < pivot) i++;
          while (x[j] > pivot) j--;
          if (i <= j) swap(&x[i++], &x[j--]);
        }
        quicksort(x, lower, j);
        quicksort(x, i, upper);
   }
 }
```

Ταχυταξινόμηση (Quicksort)

```
void quicksort (int *x, int lower, int upper) {
   if (lower < upper) {
        int pivot = x[(lower + upper) / 2];
        int i, j;
        for (i = lower, j = upper; i <= j;) {
          while (x[i] < pivot) i++;
          while (x[j] > pivot) j--;
          if (i <= j) swap(&x[i++], &x[j--]);
        }
        quicksort(x, lower, j);
        quicksort(x, i, upper);
   }
}
```

Χρόνος: $O(n^2)$ (worst case), O(nlogn) (average case)
Χώρος: O(n) (εδώ) - γίνεται και σε O(logn)

Υλοποιημένη στην συνάρτηση `qsort` της stdlib.h

Για την επόμενη φορά

Από τις διαφάνειες του κ. Σταματόπουλου καλύψαμε τις σελίδες 160-177

- Visualizing sorting algorithms [1], [2], [3]
- Visualizing binary search
- Binary Search και δυσκολίες υλοποίησης, πολλές δυσκολίες
- Sorting algorithm (περιέχει εκτενή λίστα με όλους τους αλγορίθμους)
- Quicksort analysis
- Divide and conquer

Ευχαριστώ και καλή μέρα εύχομαι!
Keep Coding ;)