2η Διάλεξη στις Δομές Δεδομένων

Γιάννης Λιβιεράτος

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Μαθηματικών

12 Μαρτίου 2024

Περιεχόμενα

Γενική Περιγραφή

- Σύνολο θέσεων μνήμης.
- Κάθε θέση μνήμης έχει συγκεκριμένη χωρητικότητά (πχ 1 byte)
- Κάθε θέση μνήμης έχει μία διεύθυνση.
- Random Access Memory: Ο χρόνος για προσπέλαση μιας θέσης μνήμης είναι ίδιος για όλες τις θέσεις.

Μεταβλητές

Σε ένα πρόγραμμα, πριν ή ταυτόχρονα με την αρχικοποίηση μιας μεταβλητής, δηλώνεται ο τύπος της:

$$int\ x,\ float\ *p,\ x = \text{“}I\ am\ a\ string\text{”}$$

Ο Η/Υ δεσμεύει κατάλληλο χώρο (θέσεις μνήμης) αναλόγως τον τύπο της κάθε μεταβλητής: π.χ. 1 byte για char, 4 για int κτλ. Θυμάται επίσης την διεύθυνση της θέσης μνήμης από την οποία ξεκινάει η δέσμευση του χώρου.

Σε γλώσσες όπως η *C*, μπορεί να γίνει και απευθείας δέσμευση θέσεων μνήμης ώστε να χρησιμοποιηθούν από το πρόγραμμα:

$$(int\ *)malloc(sizeof(int) * 5);$$

Όλοι οι τύποι μεταβλητών δεσμεύουν σταθερό, δηλαδή $O(1)$, πλήθος θέσεων μνήμης.
Απλοποιητική παραδοχή: Θα θεωρούμε ότι όλοι οι τύποι μεταβλητών δεσμεύουν 1 θέση μνήμης.

Περιεχόμενα

Γενικά

- Βασικά χαρακτηριστικά:
  - Στατική δομή,
  - Δέσμευση συνεχόμενων θέσεων μνήμης,
  - Συνήθως αποθηκεύουν δεδομένα ίδιου τύπου.
- Βασικές λειτουργίες:
  - Αρχικοποίηση/δημιουργία:

$$array = [],\ char[76]\ array,\ A = zeros(100)$$

  - Προσπέλαση/ανάκτηση στοιχείου στην i-οστή θέση:

$$A[i]$$

  - Καταχώρηση/ενημέρωση στοιχείου στην i-οστή θέση:

$$A[i] = 73$$

Πόροι

Πίνακας $A = 1 \times n$

- Χώρος: $O(n)$.
- Χρόνος:
  - Διάσχιση: $O(n)$.
  - Αρχικοποίηση: $O(n)$.
  - Εύρεση μήκους ($length(A)$): $O(n)$, εκτός αν έχει ήδη δηλωθεί.
  - Προσπέλαση/καταχώρηση στοιχείων στην θέση i: $O(1)$.

Περιεχόμενα

Ταξινόμηση ήδη ταξινομημένων πινάκων

Θα δούμε αλγορίθμους ταξινόμησης πολυπλοκότητας $O(n^2)$ και $O(n \log n)$. Θα δείξουμε ότι υπό συνθήκες, το $O(n \log n)$ είναι αυστηρό (tight bound).

Ταξινόμηση ήδη ταξινομημένων πινάκων

Θα δούμε αλγορίθμους ταξινόμησης πολυπλοκότητας $O(n^2)$ και $O(n \log n)$. Θα δείξουμε ότι υπό συνθήκες, το $O(n \log n)$ είναι αυστηρό (tight bound).

Έστω πίνακες ακέραιων A και B, με τα στοιχεία τους σε φθίνουσα σειρά. Να γραφεί αλγόριθμος που δημιουργεί πίνακα C με τα στοιχεία των A και B σε φθίνουσα σειρά, χρονικής πολυπλοκότητας $O(\mathbf{length}(A) + \mathbf{length}(B))$.

Ταξινόμηση ήδη ταξινομημένων πινάκων

Θα δούμε αλγορίθμους ταξινόμησης πολυπλοκότητας $O(n^2)$ και $O(n \log n)$. Θα δείξουμε ότι υπό συνθήκες, το $O(n \log n)$ είναι αυστηρό (tight bound).

Έστω πίνακες ακέραιων A και B, με τα στοιχεία τους σε φθίνουσα σειρά. Να γραφεί αλγόριθμος που δημιουργεί πίνακα C με τα στοιχεία των A και B σε φθίνουσα σειρά, χρονικής πολυπλοκότητας $O(\mathbf{length}(A) + \mathbf{length}(B))$.

Με βάση τα παραπάνω, ξέρουμε ότι υπάρχει αλγόριθμος χρονικής πολυπλοκότητας $O((\mathbf{length}(A) + \mathbf{length}(B))^2)$ (ποιος;).

◇ Αν A, B δεν είναι ταξινομημένοι:

COMBINE(A, B) # *int* $A[n],\ B[m]$

1: $C = []$
2: **for** $i = 1, \ldots, n$ **do**
3: $\quad C[i] = A[i]$
4: **end for**
5: **for** $j = 1, \ldots, m$ **do**
6: $\quad C[n + j] = B[j]$
7: **end for**
8: **return** SORT(C) # Κάποια γνωστή $O(n \log n)$ ταξινόμηση

◇ Αν A, B δεν είναι ταξινομημένοι:
COMBINE(A, B) # *int* $A[n],\ B[m]$

1: $C = []$
2: **for** $i = 1, \ldots, n$ **do**
3: $\quad C[i] = A[i]$
4: **end for**
5: **for** $j = 1, \ldots, m$ **do**
6: $\quad C[n + j] = B[j]$
7: **end for**
8: **return** SORT(C) # Κάποια γνωστή $O(n \log n)$ ταξινόμηση

$$O(n) + O(m) + O((n + m) \log(n + m)) = O(((n + m) \log(n + m)).$$

◇ Αν A, B δεν είναι ταξινομημένοι:

COMBINE(A, B) # *int* $A[n],\ B[m]$

1: $C = []$
2: **for** $i = 1, \ldots, n$ **do**
3: $\quad C[i] = A[i]$
4: **end for**
5: **for** $j = 1, \ldots, m$ **do**
6: $\quad C[n + j] = B[j]$
7: **end for**
8: **return** SORT(C) # Κάποια γνωστή $O(n \log n)$ ταξινόμηση

$$O(n) + O(m) + O((n + m) \log(n + m)) = O(((n + m) \log(n + m)).$$

$$\begin{aligned} & c_1 n + c_2 m + c_3 (n + m) \log(n + m) \leq^{(c_4 = \max\{c_1, c_2, c_3\})} \\ & c_4((n + m) + (n + m) \log(n + m)) \leq^{(c = 2c_4)} \\ & c(n + m) \log(n + m). \end{aligned}$$

Παράδειγμα

A= | 8 | 5 | 5 | 2 |

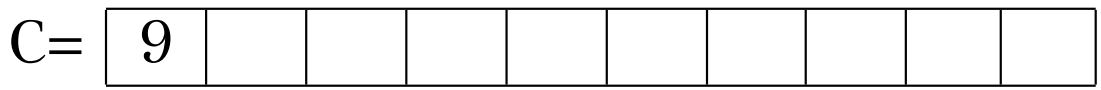
B= | 9 | 8 | 5 | 4 | 1 | 1 |

΄Ελεγχος:

C= | | | | | | | | | | |

Παράδειγμα

Έλεγχος: $A[i] \leq B[j]$



Παράδειγμα

A= | 8 | 5 | 5 | 2 |

B= | 9 | 8 | 5 | 4 | 1 | 1 |

Έλεγχος: $A[i] \leq B[j]$

C= | 9 | 8 | | | | | | | | |

Παράδειγμα

A=

8	5	5	2

B=

9	8	5	4	1	1

Έλεγχος: $A[i] \leq B[j]$

C=

9	8	8							

Παράδειγμα

A= | 8 | 5 | 5 | 2 |

B= | 9 | 8 | 5 | 4 | 1 | 1 |

Έλεγχος: $A[i] \leq B[j]$

C= | 9 | 8 | 8 | 5 | | | | | | |

Παράδειγμα

A= | 8 | 5 | 5 | 2 |

B= | 9 | 8 | 5 | 4 | 1 | 1 |

Έλεγχος: $A[i] \leq B[j]$

C= | 9 | 8 | 8 | 5 | 5 | | | | | |

Παράδειγμα

A= | 8 | 5 | 5 | 2 |

B= | 9 | 8 | 5 | 4 | 1 | 1 |

Έλεγχος: $A[i] \leq B[j]$

C= | 9 | 8 | 8 | 5 | 5 | 5 | | | | |

Παράδειγμα

A= | 8 | 5 | 5 | 2 |

B= | 9 | 8 | 5 | 4 | 1 | 1 |

Έλεγχος: $A[i] \leq B[j]$

C= | 9 | 8 | 8 | 5 | 5 | 5 | 4 | | | |

Παράδειγμα

A= | 8 | 5 | 5 | 2 |

B= | 9 | 8 | 5 | 4 | 1 | 1 |

Έλεγχος: $A[i] \leq B[j]$

C= | 9 | 8 | 8 | 5 | 5 | 5 | 4 | 2 | | |

Παράδειγμα

A= | 8 | 5 | 5 | 2 |

B= | 9 | 8 | 5 | 4 | 1 | 1 |

΄Ελεγχος:

C= | 9 | 8 | 8 | 5 | 5 | 5 | 4 | 2 | 1 | 1 |

CombineSorted(A, B) # *int* $A[n],\ B[m]$ σε φθίνουσα σειρά

```
1:  C = [],  i = j = 1
2:  for k = 1, ..., n + m do
3:      if i = n + 1 then
4:          C[k : n + m] = B[j : m]
5:      else if j = m + 1 then
6:          C[k : n + m] = A[i : n]
7:      else if A[i] ≤ B[j] then
8:          C[k] = B[j],  j+ = 1
9:      else
10:         C[k] = A[i],  i+ = 1
11:     end if
12: end for
13: return  C
```

CombineSorted(A, B) # *int* $A[n],\ B[m]$ σε φθίνουσα σειρά

```
1:  C = [],  i = j = 1
2:  for k = 1, ..., n + m do
3:      if i = n + 1 then
4:          C[k : n + m] = B[j : m]
5:      else if j = m + 1 then
6:          C[k : n + m] = A[i : n]
7:      else if A[i] ≤ B[j] then
8:          C[k] = B[j],  j+ = 1
9:      else
10:         C[k] = A[i],  i+ = 1
11:     end if
12: end for
13: return  C
```

$O(n + m)$ με pre-process $O(n \log n + m \log m)$
Πότε μας συμφέρει το pre-process?

Ταξινόμηση φραγμένου πίνακα

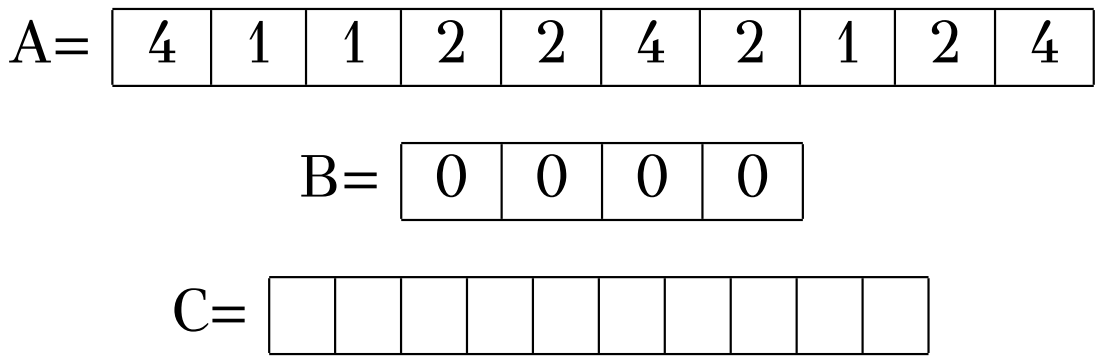
Έστω πίνακας μη-αρνητικών ακεραίων μήκους n.

Προ-επεξεργασία $O(n)$: $M = \max A$, $m = \min A$.

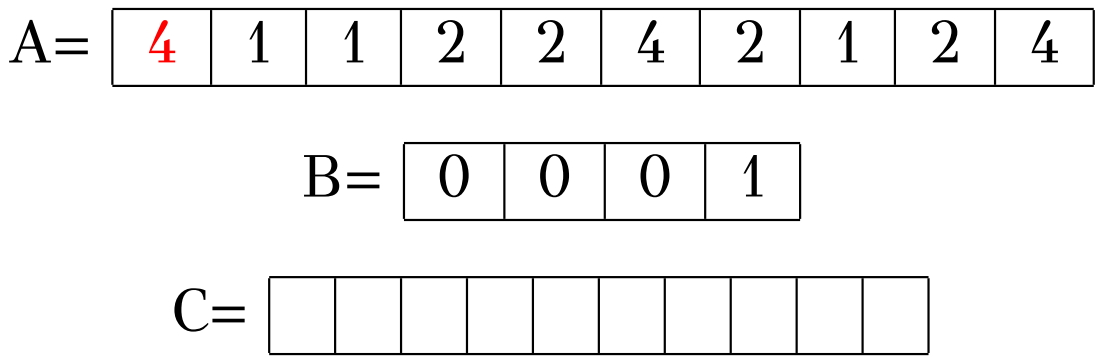
Στόχος: Ταξινόμηση στοιχείων σε αύξουσα σειρά, σε $O(n + M - m$ χρόνο.

Υπόδειξη: Να μην γίνει χρήση "συγκριτικού αλγορίθμου", δηλαδή αλγορίθμου που συγκρίνει τα στοιχεία του A μεταξύ τους.

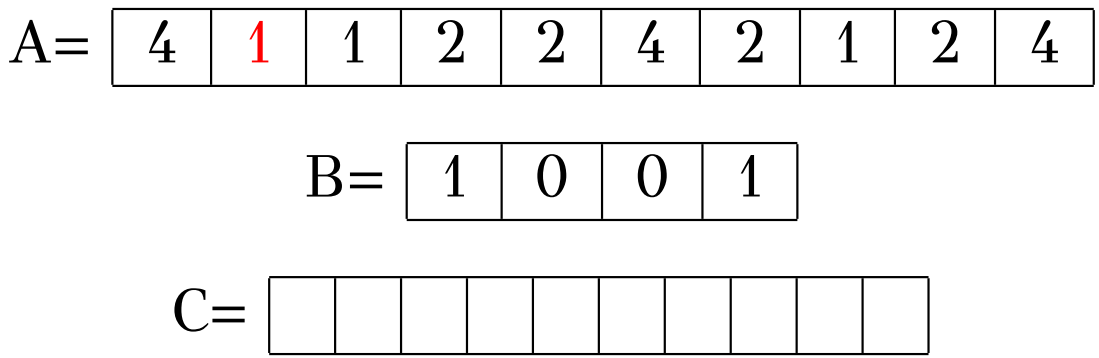
Παράδειγμα



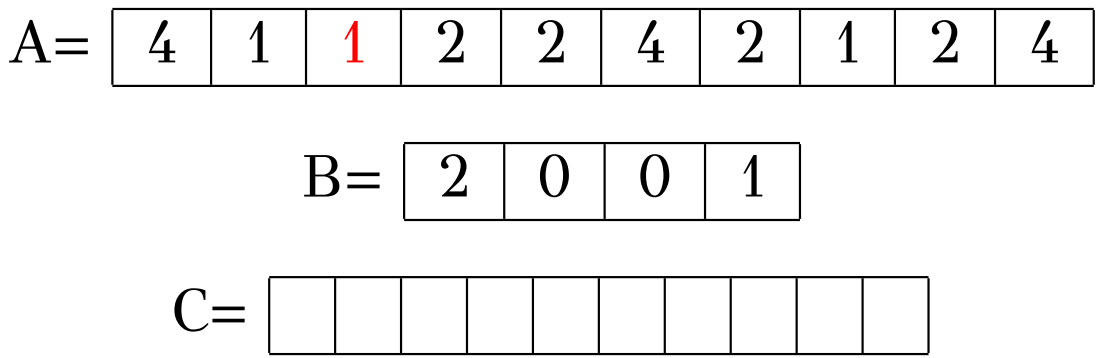
Παράδειγμα



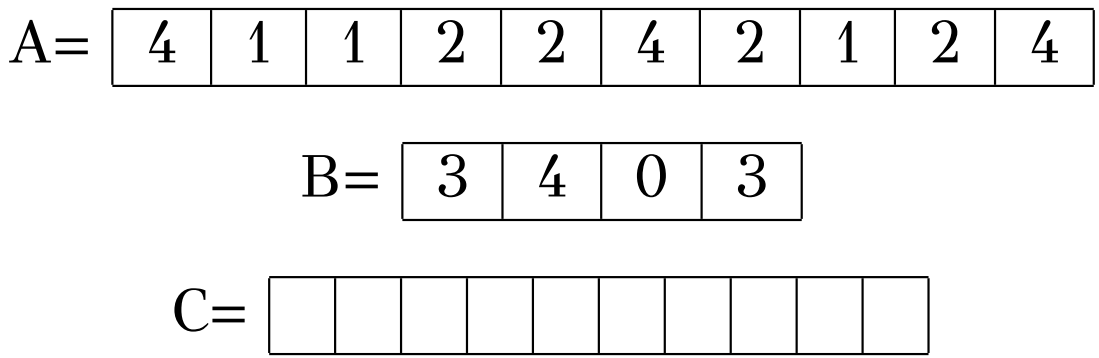
Παράδειγμα



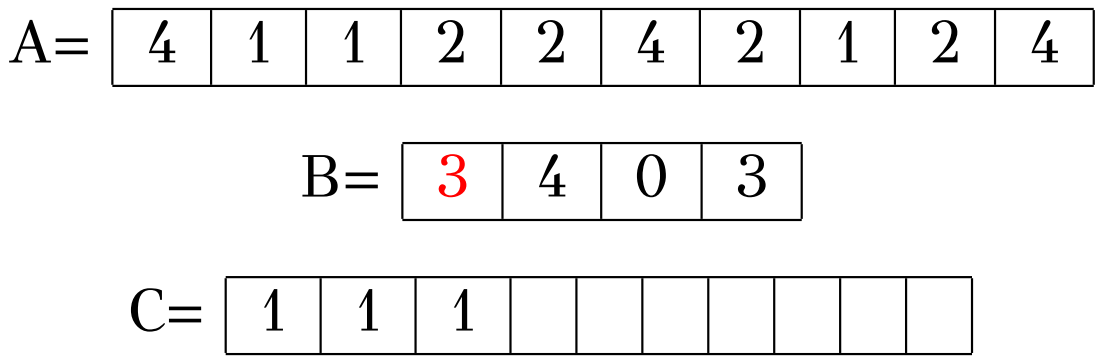
Παράδειγμα



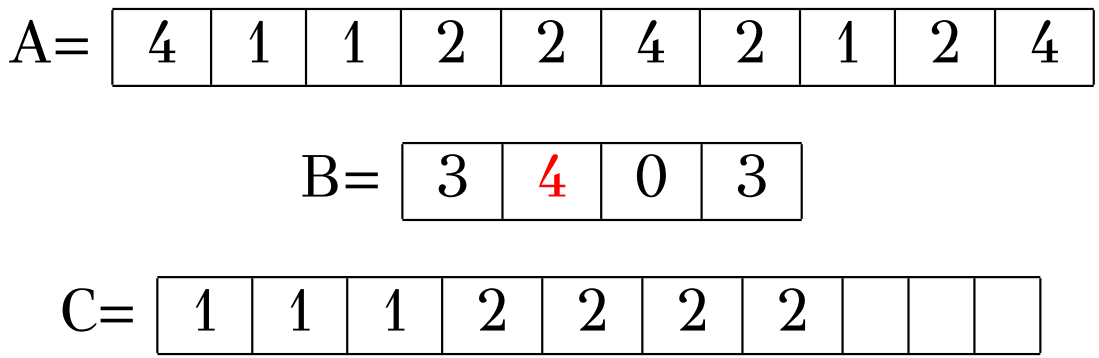
Παράδειγμα



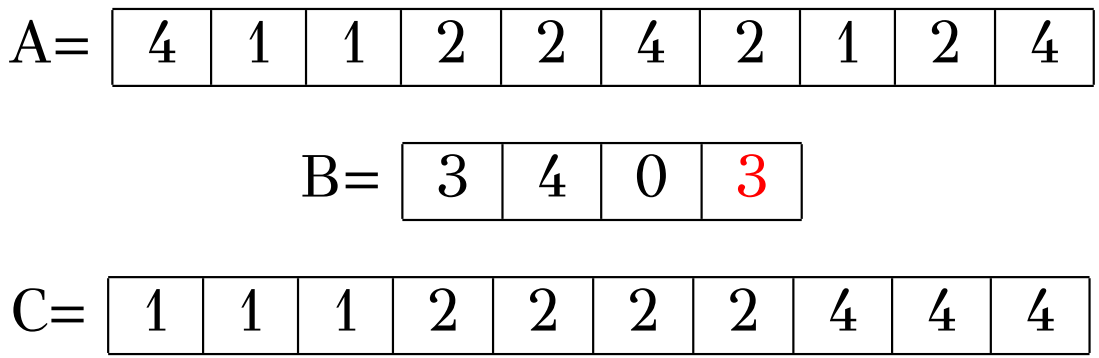
Παράδειγμα



Παράδειγμα



Παράδειγμα



BucketSort(A, M, m)

$int\ A[n],\ 0 \le m \le A[i] \le M,\ i = 1, \ldots, n$

1: $B = \text{zeros}[M - m + 1]$ # $B[i] = 0,\ i = 1, \ldots, M - m + 1$
2: **for** $i = 1, \ldots, n$ **do**
3: $\quad B[A[i] - m + 1] + +$ # $x + + \ \longrightarrow\ x + 1$
4: **end for**
5: $C = [], k = 1$
6: **for** $j = 1, \ldots, M - m + 1$ **do**
7: $\quad$ **for** $l = 1, \ldots, B[j]$ **do**
8: $\quad\quad C[k] = j + m - 1, k + +$
9: $\quad$ **end for**
10: **end for**
11: **return** C

BucketSort(A, M, m)

$int\ A[n],\ 0 \le m \le A[i] \le M,\ i = 1, \ldots, n$

1: $B = \text{zeros}[M - m + 1]$ # $B[i] = 0,\ i = 1, \ldots, M - m + 1$
2: **for** $i = 1, \ldots, n$ **do**
3: $\quad B[A[i] - m + 1] + +$ # $x + + \ \longrightarrow\ x + 1$
4: **end for**
5: $C = [], k = 1$
6: **for** $j = 1, \ldots, M - m + 1$ **do**
7: $\quad$ **for** $l = 1, \ldots, B[j]$ **do**
8: $\quad\quad C[k] = j + m - 1, k + +$
9: $\quad$ **end for**
10: **end for**
11: **return** C

- $O(n + M - m)$. Πότε είναι προτιμότερος; $M - m << n$
- Στοιχεία σε φθίνουσα σειρά; Αρνητικοί ακέραιοι;

Περιεχόμενα

Γενικά

- Βασικά χαρακτηριστικά:
  - Στατική δομή,
  - Δέσμευση συνεχόμενων θέσεων μνήμης,
  - Συνήθως αποθηκεύουν δεδομένα ίδιου τύπου.
- Βασικές λειτουργίες:
  - Αρχικοποίηση/δημιουργία:

$$array = [\ ][\ ],\ char[76][39]\ array,\ A = zeros(100, 100)$$

  - Προσπέλαση/ανάκτηση στοιχείου στην θέση (i, j):

$$A[i, j]$$

  - Καταχώρηση/ενημέρωση στοιχείου στην θέση (i, j):

$$A[i, j] = 73$$

Πόροι

Πίνακας $A = n \times m$

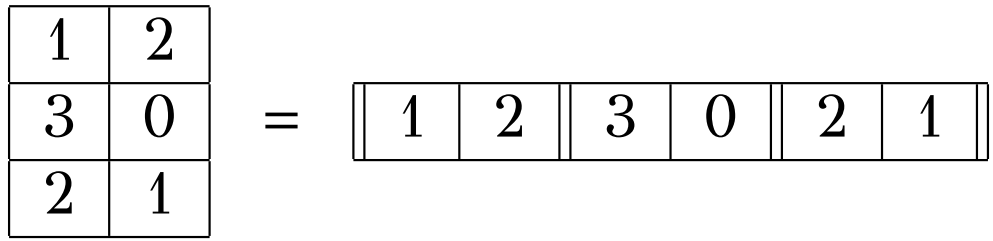
1	2
3	0
2	1

$=$

1	2	3	0	2	1

Πόροι

Πίνακας $A = n \times m$



- Χώρος: $O(nm)$.
- Χρόνος:
  - Διάσχιση: $O(nm)$.
  - Αρχικοποίηση: $O(nm)$.
  - Εύρεση διαστάσεων:
    $rows(A) = size(A, 1)$
    $cols(A) = size(A, 2)$
    $O(n + m)$, εκτός αν έχουν ήδη δηλωθεί.
  - Προσπέλαση/καταχώρηση στοιχείων στην θέση (i, j): $O(1)$.
- Αντίστοιχα για πίνακες μεγαλύτερων διαστάσεων.

Περιεχόμενα

Πολ/μός Πινάκων

$$A[n][m],\ B[m][q],\ AB[i,j] = \sum_{k=1}^{m} A[i][k]B[k][j]$$

$$\begin{array}{|c|c|c|} \hline A[1][1] & A[1][2] & A[1][3] \\ \hline A[2][1] & A[2][2] & A[2][3] \\ \hline \end{array} \cdot \begin{array}{|c|} \hline B[1][1] \\ \hline B[2][1] \\ \hline B[3][1] \\ \hline \end{array} =$$

$$\begin{array}{|c|} \hline \sum_{k=1}^{3} A[1][k]B[k][1] \\ \hline \sum_{k=1}^{3} A[2][k]B[k][1] \\ \hline \end{array}$$

Πολ/μός Πινάκων

MatrixMult(A, B) # *int* $A[n][m],\ B[p][q]$

```
1:  if m = q then
2:      C = [][]
3:      for i = 1, ..., n do
4:          for j = 1, ..., q do
5:              C[i][j] = 0
6:              for k = 1, ..., m do
7:                  C[i][j]+ = A[i][k] · B[k][i]
8:              end for
9:          end for
10:     end for
11:     return C
12: end if
```

Πολ/μός Πινάκων

MatrixMult(A, B) # *int* $A[n][m],\ B[p][q]$

```
1:  if m = q then
2:      C = [][]
3:      for i = 1, ..., n do
4:          for j = 1, ..., q do
5:              C[i][j] = 0
6:              for k = 1, ..., m do
7:                  C[i][j]+ = A[i][k] · B[k][i]
8:              end for
9:          end for
10:     end for
11:     return C
12: end if
```

Πολυπλοκότητα: $O(nmq)$ (Βελτίωση: Fast-Fourier Transform)

Αραιοί πίνακες

$A[n][n]$, με $|\{(i,j) \mid A[i][j] \neq 0\}| \ << \ |\{(i,j) \mid A[i][j] = 0\}|$

★ Καταναλώνουμε n^2 θέσεις μνήμης.

Αραιοί πίνακες

$A[n][n]$, με $|\{(i,j) \mid A[i][j] \neq 0\}| \ << \ |\{(i,j) \mid A[i][j] = 0\}|$

★ Καταναλώνουμε n^2 θέσεις μνήμης.

Έστω $\{(i,j) \mid A[i][j] \neq 0\} = \{(i_k, j_k) \mid k = 1, \ldots, m\}$, με $m << n$
(πχ $m = n^2/2$ ή n ή $\log n$)

Αραιοί πίνακες

$A[n][n]$, με $|\{(i,j) \mid A[i][j] \neq 0\}| \ << \ |\{(i,j) \mid A[i][j] = 0\}|$

★ Καταναλώνουμε n^2 θέσεις μνήμης.

Έστω $\{(i,j) \mid A[i][j] \neq 0\} = \{(i_k, j_k) \mid k = 1, \ldots, m\}$, με $m << n$
(πχ $m = n^2/2$ ή n ή $\log n$)

$$NONZERO = [A[i_1][j_1] \cdots A[i_m][j_m]]$$
$$ROWS = [i_1 \cdots i_m]$$
$$COLS = [j_1 \cdots j_m]$$

Αραιοί πίνακες

$A[n][n]$, με $|\{(i,j) \mid A[i][j] \neq 0\}| \ << \ |\{(i,j) \mid A[i][j] = 0\}|$

★ Καταναλώνουμε n^2 θέσεις μνήμης.

Έστω $\{(i,j) \mid A[i][j] \neq 0\} = \{(i_k, j_k) \mid k = 1, \ldots, m\}$, με $m << n$
(πχ $m = n^2/2$ ή n ή $\log n$)

$$
\begin{aligned}
NONZERO &= [A[i_1][j_1] \cdots A[i_m][j_m]] \\
ROWS &= [i_1 \cdots i_m] \\
COLS &= [j_1 \cdots j_m]
\end{aligned}
$$

Πώς γίνεται ενημέρωση νέας τιμής x στην θέση $A[i][j]$;

Αραιοί πίνακες

$A[n][n]$, με $|\{(i,j) \mid A[i][j] \neq 0\}| \ << \ |\{(i,j) \mid A[i][j] = 0\}|$

★ Καταναλώνουμε n^2 θέσεις μνήμης.

Έστω $\{(i,j) \mid A[i][j] \neq 0\} = \{(i_k, j_k) \mid k = 1, \ldots, m\}$, με $m << n$
(πχ $m = n^2/2$ ή n ή $\log n$)

$$NONZERO = [A[i_1][j_1] \cdots A[i_m][j_m]]$$
$$ROWS = [i_1 \cdots i_m]$$
$$COLS = [j_1 \cdots j_m]$$

Πώς γίνεται ενημέρωση νέας τιμής x στην θέση $A[i][j]$;

- Αν $A[i][j], x \neq 0$, απλή αντικατάσταση στο $NONZERO$.
- Αν $A[i][j] \neq 0$, $x = 0$, διαγραφή $A[i][j], i, j$ από τους πίνακες.
- Αν $A[i][j] = 0$, $x \neq 0$; Αλγόριθμος;