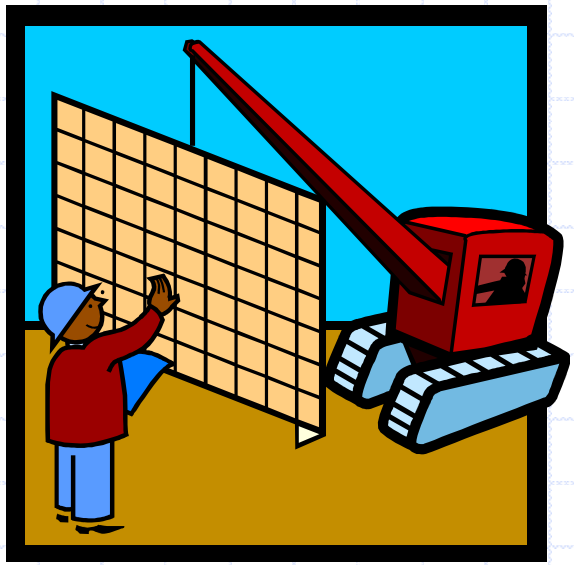
Λίστες



Έστω μια συλλογή S από n στοιχεία που έχουν αποθηκευθεί με κάποια γραμμική σειρά, έτσι που να αναφερόμαστε στο στα στοιχεία της S σαν πρώτο, δεύτερο, τρίτο κοκ. Μια τέτοια συλλογή γενικά αναφέρεται σαν **λίστα** ή **ακολουθία**. Μπορούμε να αναφερόμαστε μοναδικά σε κάθε στοιχείο e της S χρησιμοποιώντας έναν ακέραιο στο διάστημα [0,n-1] που είναι ίσος με το πλήθος των στοιχείων της S που προηγούνται του e. Ο **δείκτης** ενός στοιχείου e της S είναι το πλήθος των στοιχείων της S που προηγούνται του e. Επομένως, το πρώτο στοιχείο της S έχει δείκτη 0 και το τελευταίο n-1. Επίσης, αν ένα στοιχείο της S έχει δείκτη i, το προηγούμενο στοιχείο του (αν υπάρχει) έχει δείκτη i-1, και το επόμενό του (αν υπάρχει) έχει δείκτη i+1. Αυτή η έννοια του δείκτη σχετίζεται με αυτήν της κατάταξης ενός στοιχείου σε μια λίστα, που συνήθως ορίζεται ένα παραπάνω από τον δείκτη του• επομένως το πρώτο στοιχείο έχει κατάταξη 1 , το δεύτερο 2, κοκ.

Ο ΑΤΔ λίστα

- Ο ΑΤΔ πίνακα επεκτείνει την έννοια του πίνακα με την αποθήκευση μιας ακολουθίας αντικειμένων
- Μπορεί να γίνει προσπέλαση σε ένα αντικείμενο, μπορεί να γίνει εισαγωγή ή διαγραφή, προσδιορίζοντας τη θέση του (πλήθος στοιχείων που προηγούνται)
- Βγαίνει εξαίρεση αν δοθεί μη επιτρεπτή θέση (πχ., αρνητικός)

- Βασικές μέθοδοι:
  - get(integer i): επιστρέφει το στοιχείο στη θέση i χωρίς διαγραφή του
  - set(integer i, object o): αντικατάσταση του στοιχείου στη θέση i με το o και επιστροφή του παλαιού στοιχείου
  - add(integer i, object o): εισαγωγή ενός νέου στοιχείου o στη θέση i
  - remove(integer i): διαγράφει και επιστρέφει το στοιχείο στη θέση i
- Επιπλέον μέθοδοι:
  - size()
  - isEmpty()

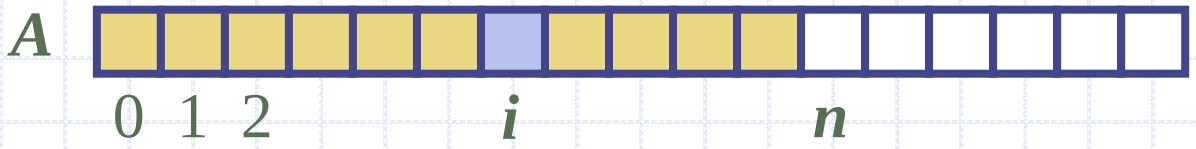
Τελεστής	Έξοδος	S
add(0,7)	-	(7)
add(0,4)	-	(4,7)
get(1)	7	(4,7)
add(2,2)	-	(4,7,2)
get(3)	“error”	(4,7,2)
remove(1)	7	(4,2)
add(1,5)	-	(4,5,2)
add(1,3)	-	(4,3,5,2)
add(4,9)	-	(4,3,5,2,9)
get(2)	5	(4,3,5,2,9)
set(3,8)	2	(4,3,5,8,9)

Εφαρμογές

- Άμεσες Εφαρμογές
  - Ταξινομημένη συλλογή αντικειμένων
- Έμεσες Εφαρμογές
  - Βοηθητική δομή δεδομένων για αλγορίθμους
  - Στοιχείο άλλων δομών δεδομένων

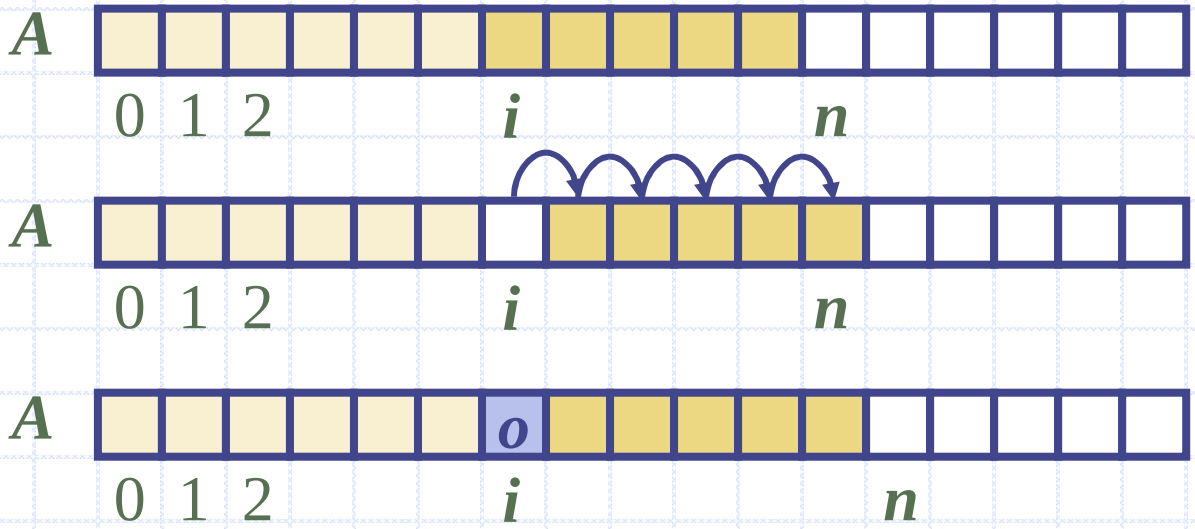
Υλοποίηση που βασίζεται σε πίνακες

- Χρήση ενός πίνακα A μεγέθους N
- Μια μεταβλητή n καταγράφει το μέγεθος της λίστας (πλήθος αποθηκευμένων στοιχείων)
- Η πράξη $get(i)$ υλοποιείται σε χρόνο $O(1)$ επιστρέφοντας το $A[i]$
- Η πράξη $set(i,o)$ υλοποιείται σε χρόνο $O(1)$ εκτελώντας τα $t = A[i]$, $A[i] = o$, και επιστρέφει t.



Εισαγωγή

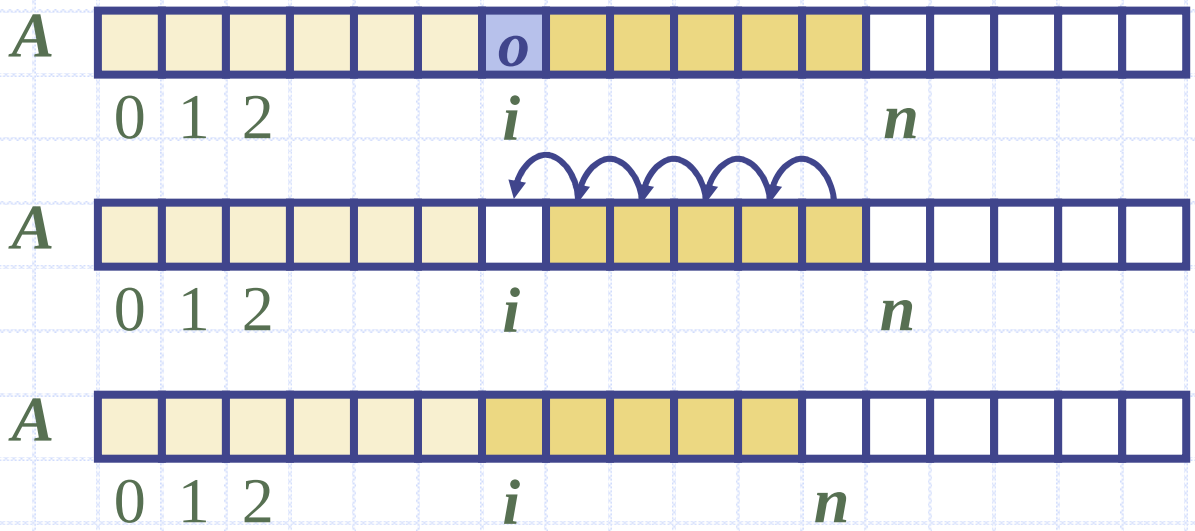
- Για την πράξη $add(i, o)$, πρέπει να δημιουργήσουμε χώρο για το νέο στοιχείο ολισθαίνοντας τα $n - i$ στοιχεία $A[i], \ldots, A[n-1]$
- Στη χειρότερη περίπτωση ($i = 0$),αυτό απαιτεί χρόνο $O(n)$



```
Algorithm add(i,e)
        for j=n-1, n-2,…i do
                A[j+1]←A[j] {δημιουργία χώρου
                        για το νέο στοιχείο}
                A[i] ← e
        n ←n+1
```

Διαγραφή στοιχείου

- Για την πράξη ***remove***(i), πρέπει να γεμίσουμε το κενό που αφήνει το στοιχείο που διαγράφουμε με ολίσθηση προς τα πίσω των $n - i - 1$ στοιχείων $A[i + 1], \ldots, A[n - 1]$
- Στη χειρότερη περίπωση $(i = 0)$, απαιτεί χρόνο $O(n)$



```
Algorithm remove(i)
    e←A[i]  {το e είναι προσωρινή μεταβλητή}
    for j ←i,i+1,…,n-2 do
        A[j] ←A[j+1] {συμπλήρωση για το
                      στοιχείο που διαγράφεται}
    n ←n-1
return e
```

Απόδοση

- Στην υλοποίηση μιας λίστας πίνακα με πίνακα:
  - Ο χώρος που απαιτεί η δομή δεδομένων είναι $O(n)$
  - Οι ***size***, ***isEmpty***, ***get*** και ***set*** τρέχουν σε χρόνο $O(1)$
  - ***add*** και ***remove*** τρέχουν σε χρόνο $O(n)$ στη χειρότερη περίπτωση
- Αν χρησιμοποιήσουμε τον πίνακα κατά κυκλικό τρόπο, οι πράξεις ***add***$(0, x)$ και ***remove***$(0, x)$ τρέχουν σε χρόνο $O(1)$
- Για μια πράξη ***add***, όταν ο πίνακας είναι γεμάτος, αντί για εξαίρεση, μπορούμε να αντικαταστήσουμε τον πίνακα με έναν μεγαλύτερο

Απόδοση της υλοποίησης με πίνακα

Μέθοδος	Χρόνος
size()	O(1)
isEmpty	O(1)
get(i)	O(1)
set(i,e)	O(1)
add(i,e)	O(n)
remove(i)	O(n)

Πίνακας λίστα με δυνατότητα επέκτασης

- Σε μια πράξη add(o) (χωρίς δείκτη), προσθέτουμε πάντα στο τέλος
- Όταν ο πίνακας είναι γεμάτος, αντικαθιστούμε τον πίνακα με έναν μεγαλύτερο
- Πόσο πιό μεγάλος πρέπει να είναι ο νέος πίνακας?
  - Αυξητική: αύξηση του μεγέθους του πίνακα κατά μια σταθερά c
  - Στρατηγική διπλασιασμού: διπλασιασμός του μεγέθους

Algorithm *add*(*o*)
  if $t = S.length - 1$ **then**
    $A \leftarrow$ **new array of size …**
    for $i \leftarrow 0$ **to** $n-1$ **do**
      $A[i] \leftarrow S[i]$
    $S \leftarrow A$
  $n \leftarrow n + 1$
  $S[n-1] \leftarrow o$

Σύγκριση στρατηγικών

- Συγκρίνουμε την αυξητική στρατηγική και την στρατηγική διπλασιασμού αναλύοντας το συνολικό χρόνο $T(n)$ που απαιτείται για εκτέλεση μιας ακολουθίας n add(o) πράξεων
- Υποθέτουμε ότι ξεκινάμε με μια κενή στοίβα που αναπαρίσταται από ένα πίνακα μεγέθους 1
- Ονομάζουμε κλιμακούμενο χρόνο μιας πράξης προσθήκης το μέσο χρόνο που απαιτείται για μια ακολουθία πράξεων προσθήκης δηλ. $T(n)/n$

Ανάλυση Αυξητικής Στρατηγικής

- Αντικαθιστούμε τον πίνακα $k = n/c$ φορές
- Ο συνολικός χρόνος $T(n)$ μιας ακολουθίας από n πράξεις πρόσθεσης είναι ανάλογη του

$$n + c + 2c + 3c + 4c + \ldots + kc =$$

$$n + c(1 + 2 + 3 + \ldots + k) =$$

$$n + ck(k + 1)/2$$

- Εφόσον το c είναι μια σταθερά, το $T(n)$ είναι $O(n + k^2)$, δηλ., $O(n^2)$
- Ο κλιμακούμενος χρόνος μιας πράξης προσθήκης είναι $O(n)$

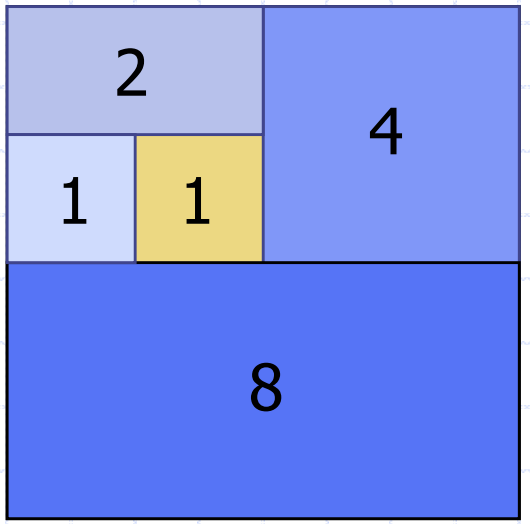
Ανάλυση στρατηγικής διπλασιασμού

- Αντικαθιστούμε τον πίνακα $k = \log_2 n$ φορές
- Ο συνολικός χρόνος $T(n)$ μιας ακολουθίας n add πράξεων προσθήκης είναι ανάλογος του

$$n + 1 + 2 + 4 + 8 + \ldots + 2^k =$$
$$n + 2^{k+1} - 1 =$$
$$3n - 1$$

- $T(n)$ είναι $O(n)$
- Ο κλιμακούμενος χρόνος μιας πράξης προσθήκης είναι $O(1)$

Γεωμετρική πρόοδος



Περιγράψτε την έξοδο της παρακάτω ακολουθίας πράξεων σε στοίβα: push(5), push(3), pop(), push(2), push(8), pop(), pop(0, push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(0, pop(0).

Περιγράψτε την έξοδο για την παρακάτω ακολουθία πράξεων σε μια ουρά: enqueue(5), enqueue(3), dequeue(), enqueue(2), enqueue(8), dequeue(), dequeue(), enqueue(9), enqueue(1), dequeue(), enqueue(7), enqueue(6), dequeue(),dequeue(), enqueue(4), dequeue(),dequeue().

Υποθέστε ότι έχετε μια στοίβα S που περιέχει n στοιχεία και μια ουρά Q που είναι αρχικά κενή. Περιγράψτε πως μπορείτε να χρησιμοποιήστε την Q για να εξετάσετε αν η S περιέχει κάποιο στοιχείο x, με τον επιπλέον περιορισμό ότι ο αλγόριθμός σας πρέπει να επιστρέφει τα στοιχεία στην S στην αρχική σειρά τους. Θα πρέπει να μην χρησιμοποιήσετε πίνακα ή συνδεδεμένη λίστα-μόνο τις S και Q και ένα σταθερό πλήθος από μεταβλητές αναφοράς.