Ουρές Προτεραιότητας

ΑΤΔ Ουρά Προτεραιότητας

- Μια ουρά προτεραιότητας αποθηκεύει μια συλλογή από καταχωρήσεις
- Κάθε καταχώρηση είναι ένα ζεύγος (key, value)
- Βασικές μεθοδοι του ΑΤΔ Ουρά Προτεραιότητας
  - insert(k, x)
    εισάγει μια καταχώρηση με κλειδί k και τιμή x
  - removeMin()
    διαγράφει και επιστρέφει την καταχώρηση με την μικρότερη τιμή

- Επιπλέον μέθοδοι
  - min()
    επιστρέφει, αλλά δεν διαγράφει, μια καταχώρηση με το μικρότερο κλειδί
  - size(), isEmpty()
- Εφαρμογές:
  - Standby επιβάτες
  - Πλειστηριασμοί
  - Αγορά μετοχών

Σχέσεις Ολικής Διάταξης

- Τα κλειδιά σε μια ουρά προτεραιότητας μπορεί να είναι οποιαδήποτε αντικείμενα στα οποία ορίζεται μια διάταξη
- Δύο διακριτές καταχωρήσεις μπορεί να έχουν το ίδιο κλειδί

- Μαθηματική έννοια της σχέσης ολικής διάταξης $\leq$
  - Αντανακλαστική ιδιότητα: $x \leq x$
  - Αντισυμμετρική ιδιότητα: $x \leq y \wedge y \leq x \Rightarrow x = y$
  - Μεταβατική ιδιότητα: $x \leq y \wedge y \leq z \Rightarrow x \leq z$

Καταχώρηση ΑΤΔ

- Μια καταχώρηση σε μια ουρά προτεραιότητας είναι απλά ένα ζεύγος κλειδί-τιμή
- Οι ουρές προτεραιότητας αποθηκεύουν καταχωρήσεις που υποστηρίζουν την αποτελεσματική εισαγωγή και διαγραφή με βάση τα κλειδιά
- Μέθοδοι:
  - getKey: επιστρέφει το κλειδί της καταχώρησης
  - getValue: επιστρέφει την τιμή της καταχώρησης

- Σαν Java διεπαφή:

```
/**
 * Interface for a key-value
 * pair entry
 **/
public interface Entry<K,V>
  {
  public K getKey();
  public V getValue();
}
```

ΑΤΔ Τελεστή Συγκρισης

- Ένας τελεστής σύγκρισης ενθυλακώνει την πράξη της σύγκρισης δυο αντικειμένων σύμφωνα με δοθείσα σχέση ολικής διάταξης
- Μια πρωτογενής ουρά προτεραιότητας χρησιμοποιεί ένα βοηθητικό τελεστή σύγκρισης
- Ο τελεστής σύγκρισης είναι εξωτερικός στα κλειδιά σύγκρισης
- Όταν η ουρά προτεραιότητας πρέπει να συγκρίνει δύο κλειδιά, χρησιμοποιεί τον τελεστή σύγκρισης

- Πρωταρχική μέθοδος ΑΤΔ τελεστή σύγκρισης
- compare(x, y): επιστρέφει έναν ακέραιο i τέτοιο ώστε
  - $i < 0$ αν $a < b$,
  - $i = 0$ αν $a = b$
  - $i > 0$ αν $a > b$
  - Λάθος αν τα a και b δεν μπορούν να συγκριθούν.

Παράδειγμα Τελεστή Σύγκρισης

- Λεξικογραφική σύγκριση σημείων 2-D:

```
/** Τελεστής σύγκρισης για σημεία 2D με
    την τυπική λεξικογραφική διάταξη */
public class Lexicographic implements
    Comparator {
  int xa, ya, xb, yb;
  public int compare(Object a, Object b)
   throws ClassCastException {
    xa = ((Point2D) a).getX();
    ya = ((Point2D) a).getY();
    xb = ((Point2D) b).getX();
    yb = ((Point2D) b).getY();
    if (xa != xb)
        return (xb - xa);
    else
        return (yb - ya);
  }
}
```

- Αντικείμενα σημεία :

```
/** Class representing a point in the
    plane with integer coordinates */
public class Point2D {
  protected int xc, yc; // coordinates
  public Point2D(int x, int y) {
    xc = x;
    yc = y;
  }
  public int getX() {
        return xc;
  }
  public int getY() {
        return yc;
  }
}
```

Ταξινόμηση Ουράς Προτεραιότητας

- Μπορούμε να χρησιμοποιήσουμε μια ουρά προτεραιότητας για ταξινόμηση ενός συνόλου στοιχείων με δυνατότητα σύγκρισης
  1. Εισαγωγή στοιχείων ένα-ένα με μια σειρά πράξεων insert
  2. Διαγραφή των στοιχείων της ταξινομημένης διάταξης με μια σειρά πράξεων removeMin
- Ο χρόνος τρεξίματος αυτής της μεθόδου ταξινόμησης εξαρτάται από την υλοποίηση της ουράς προτεραιότητας

Algorithm *PQ-Sort*(*S*, *C*)

Input sequence *S*, comparator *C* for the elements of *S*

Output sequence *S* sorted in increasing order according to *C*

P ← priority queue with comparator *C*

while ¬*S.isEmpty* ()

e ← *S.removeFirst* ()

P.insert (*e*, ∅)

while ¬*P.isEmpty*()

e ← *P.removeMin*().*getKey*()

S.addLast(*e*)

Ο Αφηρημένος τύπος της Ουράς Προτεραιότητας

- size(): επιστρέφει το πλήθος των καταχωρήσεων της P.
- isEmpty(): ελέγχει αν η P είναι κενή
- min(): επιστρέφει (αλλά δεν αφαιρεί από την P την καταχώρηση με το μικρότερο κλειδί ή μια συνθήκη λάθους αν η P είναι κενή
- insert(k, x): εισάγει στην P το κλειδί k με τιμή x και επιστρέφει την καταχώρηση που το περιέχει. Αν το k δεν έχει επιτρεπόμενη τιμή επιστρέφει συνθήκη λάθους
- removeMin(): Αφαιρεί από την P και επιστρέφει την καταχώρηση με το μικρότερο κλειδί. Αν η P είναι κενή συμβαίνει συνθήκη λάθους

Παράδειγμα

Πράξη	Έξοδος	Ουρά Προτεραιότητας
insert(5, A)	e_1[=(5,A)]	{(5,A)}
insert(9, C)	e_2[=(9,C)]	{(5,A), (9,C)}
insert(3, B)	e_3[=(3,B)]	{(3,B), (5,A), (9,C)}
insert(7, D)	e_4[=(7,D)]	{(3,B), (5,A), (7,D), (9,C)}
min()	e_3	{(3,B), (5,A), (7,D), (9,C)}
removeMin()	e_3	{(5,A), (7,D), (9,C)}
size()	3	{(5,A), (7,D), (9,C)}
removeMin()	e_1	{(7,D), (9,C)}
removeMin()	e_4	{(9,C)}
removeMin()	e_2	{}

Ταξινόμηση με Ουρά προτεραιότητας

Μια σημαντική εφαρμογή των ουρών προτεραιότητας είναι η ταξινόμηση μιας λίστας S. Ο αλγόριθμος είναι σχετικά απλός και ακολουθεί δύο φάσεις:

1. Στην πρώτη φάση θέτουμε τα στοιχεία της S σε μια αρχικά άδεια ουρά προτεραιότητας εκτελώντας n εισαγωγές, μια για κάθε στοιχείο.
2. Στη δεύτερη φάση εξάγουμε τα στοιχεία από την P σε αύξουσα σειρά με μια σειρά από removeMin πράξεις και τα θέτουμε πίσω στην S.

```
Αλγόριθμος PriorityQueueSort(S, P)
Είσοδος: μια λίστα S με n στοιχεία, και μια ουρά προτεραιότητας P
Έξοδος: η λίστα S ταξινομημένη
while !S.isEmpty() do
  e←S.removeFirst()
  P.insert(e, ∅)
  while !P.isEmpty() do
  e ← P.removeMin().getkey()
  S.addLast(e)
```

Ουρά προτεραιότητας που βασίζεται σε ακολουθία

- Υλοποίηση με μη ταξινομημένη λίστα

- Απόδοση:
  - insert απαιτεί χρόνο $O(1)$ αφού μπορούμε να προσθέσουμε το στοιχείο στην αρχή ή το τέλος της ακολουθίας
  - removeMin και min απαιτεί χρόνο $O(n)$ αφού πρέπει να σαρωθεί όλη η ακολουθία για να βρεθεί το μικρότερο κλειδί

- Υλοποίηση με μια ταξινομημένη λίστα

- Απόδοση:
  - insert απαιτεί χρόνο $O(n)$ αφού πρέπει να βρεθεί η θέση εισαγωγής
  - removeMin και min απαιτεί χρόνο $O(1)$, αφού το μικρότερο κλειδί είναι στην αρχή

Ταξινόμηση με επιλογή

- Η ταξινόμηση με επιλογή είναι παραλλαγή της της ταξινόμησης ουράς προεραιότητας όπου η ουρά προτεραιότητας υλοποιείται με μη ταξινομημένη ακολουθία
- Χρόνος τρεξίματος ταξινόμησης με επιλογή:
  1. Η εισαγωγή των στοιχείων στην ουρά προτεραιότητας με n insert πράξεις απαιτεί χρόνο $O(n)$
  2. Η διαγραφή των στοιχείων σε ταξινομημενη σειρά από την ουρά προτεραιότητας με n removeMin πράξεις απαιτεί χρόνο ανάλογο του

$$1 + 2 + \ldots + n$$

- Η ταξινόμηση με επιλογή τρέχει σε χρόνο $O(n^2)$

Παράδειγμα Ταξινόμησης με επιλογή

	Ακολουθία S	Ουρά προτεραιότητας P
Είσοδος:	(7,4,8,2,5,3,9)	()
Φάση 1		
(a)	(4,8,2,5,3,9)	(7)
(b)	(8,2,5,3,9)	(7,4)
..	.. ..	
(g)	()	(7,4,8,2,5,3,9)
Φάση 2		
(a)	(2)	(7,4,8,5,3,9)
(b)	(2,3)	(7,4,8,5,9)
(c)	(2,3,4)	(7,8,5,9)
(d)	(2,3,4,5)	(7,8,9)
(e)	(2,3,4,5,7)	(8,9)
(f)	(2,3,4,5,7,8)	(9)
(g)	(2,3,4,5,7,8,9)	()

Ταξινόμηση με εισαγωγή

- Η ταξινόμηση με εισαγωγή είναι παραλλαγή της της ταξινόμησης ουράς προεραιότητας όπου η ουρά προτεραιότητας υλοποιείται με ταξινομημένη ακολουθία
- Χρόνος τρεξίματος ταξινόμησης με εισαγωγή:
  1. Η εισαγωγή των στοιχείων στην ουρά προτεραιότητας με n insert πράξεις απαιτεί χρόνο ανάλογο του
  $$1 + 2 + \ldots + n$$
  2. Η διαγραφή των στοιχείων σε ταξινομημενη σειρά από την ουρά προτεραιότητας με n removeMin πράξεις απαιτεί χρόνο $O(n)$
- Η ταξινόμηση με εισαγωγή τρέχει σε χρόνο $O(n^2)$

Παράδειγμα ταξινόμησης με εισαγωγή

	Ακολουθία S	Ουρά προτεραιότητας P
Είσοδος:	(7,4,8,2,5,3,9)	()
Φάση 1		
(a)	(4,8,2,5,3,9)	(7)
(b)	(8,2,5,3,9)	(4,7)
(c)	(2,5,3,9)	(4,7,8)
(d)	(5,3,9)	(2,4,7,8)
(e)	(3,9)	(2,4,5,7,8)
(f)	(9)	(2,3,4,5,7,8)
(g)	()	(2,3,4,5,7,8,9)
Φάση 2		
(a)	(2)	(3,4,5,7,8,9)
(b)	(2,3)	(4,5,7,8,9)
..	..	..
(g)	(2,3,4,5,7,8,9)	()

Ταξινόμηση με εισαγωγή στην ίδια δομή

- Αντί να χρησιμοποίηθεί μια εξωτερική δομή δεδομένων, μπορούμε να υλοποιήσουμε την ταξινόμηση με επιλογή και με εισαγωγή στην ίδια δομή
- Ένα μέρος της ακολουθίας εισαγωγής χρησιμοποιείται σαν ουρά προτεραιότητας
- Για ταξινόμηση με εισαγωγή στην ίδια δομή
  - Κρατάμε ταξινομημένο το αρχικό τμήμα της ακολουθίας
  - Μπορούμε να χρησιμοποιήσουμε *ανταλλαγές αντί* για τροποποίηση της ακολουθίας