

Αλγόριθμοι και Πολυπλοκότητα

7ο εξάμηνο
Σ.Η.Μ.Μ.Υ. & Σ.Ε.Μ.Φ.Ε.

<http://www.corelab.ece.ntua.gr/courses/>

2η εβδομάδα: *Ουρές Προτεραιότητας, Σωροί, Πράξεις
Συνόλων, Λεξικά, Union-Find*

Διδάσκοντες:
Στάθης Ζάχος - Άρης Παγουρτζής

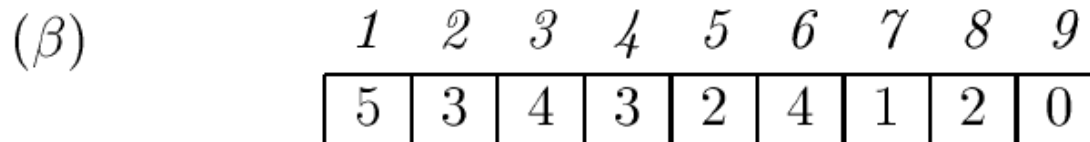
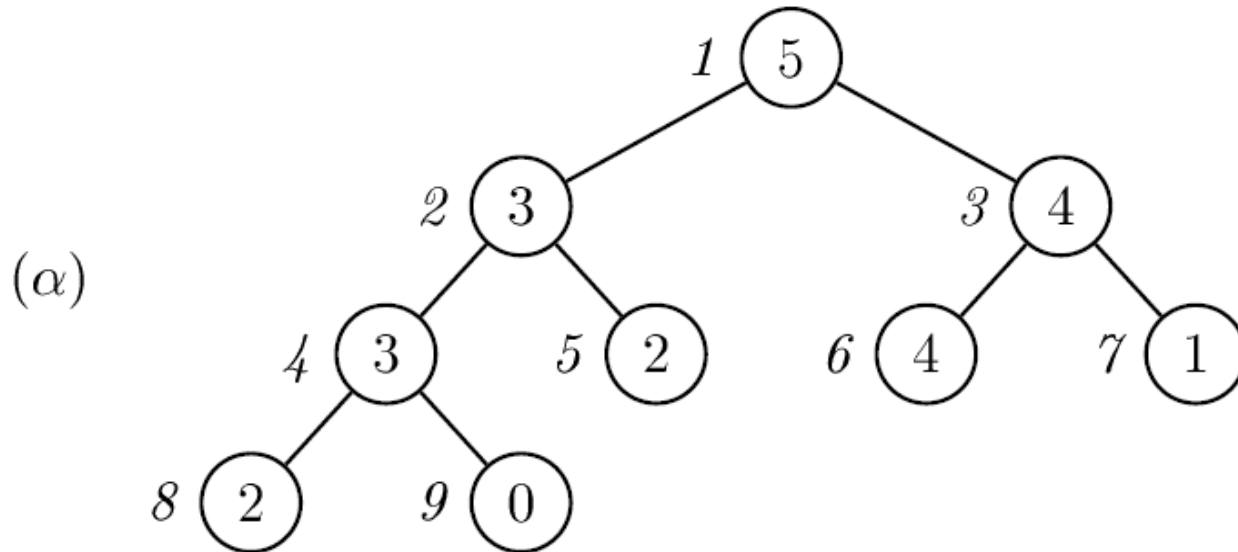
Ουρές Προτεραιότητας

- Έστω ότι μας δίνονται κάποια στοιχεία για τα οποία ισχύει μια ολική σχέση διάταξης ($<$).
- Μια δομή δεδομένων η οποία μας επιτρέπει να κάνουμε με αποδοτικό τρόπο εισαγωγή ενός καινούργιου στοιχείου και εξαγωγή (δηλαδή εύρεση και διαγραφή) του μεγαλύτερου στοιχείου, λέγεται ουρά με προτεραιότητα (priority queue).

Σωρός

- Σωρός (heap tree) είναι ένα πλήρες δυαδικό δέντρο στο οποίο:
 - η τιμή της ρίζας είναι μεγαλύτερη ή ίση (ή μικρότερη ή ίση) από τις τιμές των υπολοίπων κόμβων και,
 - αυτό ισχύει αναδρομικά για την ρίζα κάθε υποδέντρου.

Σωρός (παράδειγμα)



Σωρός: κατασκευή (insert)

```
procedure insert(var a:array; n:integer);  
var item:integer; k:integer;  
begin  
  item:=a[n]; k:=n div 2; (* εύρεση του γονέα *)  
  while ((k>0) and (a[k]<item)) do  
    begin a[n]:=a[k]; n:=k; k:=k div 2; end;  
  a[n]:=item  
end
```

```
procedure ConstructHeapInsert(var a:array; n:integer);  
var i:integer;  
begin  
  for i:=2 to n do insert(a,i)  
end
```

Σωρός: κατασκευή (combine)

```
procedure combine(var a:array; i,n:integer);
begin
  while 2i<=n do    (* node i has at least one child *)
  begin
    l:= 2i  (* left child *)
    r:= 2i+1 (* right child *)
    largest_child:=l;
    if r<=n and a[r]>a[l] then largest_child:=r;
    (* largest_child: position of child with larger value *)
    if a[i]<a[largest_child] then
    begin
      swap(a[i], a[largest_child]);
      i:=largest_child
    end
    else i:=n div 2 + 1  (*exit while *)
  end
end
```

Σωρός: κατασκευή (combine) (συν.)

```
procedure CostructHeapCombine(var a:array; n:integer);  
var i:integer;  
begin  
    for i:=n div 2 downto 1 do  
        combine(a,i,n)  
end
```

Σωρός: Ταξινόμηση

```
procedure HeapSort(var a:array; n:integer);  
var i:integer;  
begin  
    ConstructHeap(a,n); (* Αλγόριθμος 1 ή  
                        Αλγόριθμος 2 *)  
    for i:=n downto 2 do  
        begin  
            swap(a[1],a[i]); combine(a,1,i-1)  
        end  
    end  
end
```


Πολυπλοκότητα Ταξινόμησης

- Κάθε δέντρο απόφασης για ταξινόμηση n στοιχείων, έχει πολυπλοκότητα (ύψος) $\Omega(n \log n)$.
- Η ταξινόμηση με συγκρίσεις n στοιχείων, έχει χρονική πολυπλοκότητα $\Theta(n \log n)$.

Πράξεις Συνόλων (I)

- Member(a, S): Ελέγχει αν το στοιχείο a ανήκει στο σύνολο S και αν αυτό ισχύει επιστρέφει true αλλιώς false. Search(a, S): Επιστρέφει ένα δείκτη στο στοιχείο a αν $a \in S$ αλλιώς επιστρέφει nil.
- Insert(a, S): Εισάγει το στοιχείο a στο σύνολο S και επιστρέφει το σύνολο $S \cup \{a\}$.
- Delete(a, S): Διαγράφει το στοιχείο a από το σύνολο S και επιστρέφει το σύνολο $S \setminus \{a\}$.

Πράξεις Συνόλων (II)

- Union(S_1, S_2): Επιστρέφει το σύνολο $S_1 \cup S_2$. Υποθέτουμε ότι τα σύνολα S_1, S_2 είναι ξένα (disjoint) για να αποφύγουμε τον έλεγχο για διπλά στοιχεία.
- Find(a): Επιστρέφει το όνομα του συνόλου στο οποίο ανήκει το a . Υποθέτουμε ότι έχουμε διαμέριση σε ξένα σύνολα, άρα το στοιχείο a , θα ανήκει σε ένα ακριβώς σύνολο.
- Split(a, S): Χωρίζει το σύνολο S σε δύο σύνολα S_1, S_2 τέτοια ώστε:

$$S_1 = \{b \mid b \leq a \wedge b \in S\} \text{ και } S_2 = \{b \mid b > a \wedge b \in S\}$$

Εδώ υποθέτουμε ότι το σύνολο S είναι ένα σύνολο του οποίου τα στοιχεία έχουν μια γραμμική διάταξη (\leq).

Πράξεις Συνόλων (III)

- Max(S), Min(S): Επιστρέφει το μεγαλύτερο ή το μικρότερο των στοιχείων του S . Υποθέτουμε πάλι γραμμική διάταξη των στοιχείων του S .
- Successor(a, S): Επιστρέφει το μικρότερο από τα στοιχεία του S που είναι μεγαλύτερο του a (επόμενο στοιχείο). Predecessor(a, S): Ομοίως επιστρέφει το μεγαλύτερο από τα στοιχεία του S που είναι μικρότερο του a (προηγούμενο στοιχείο).

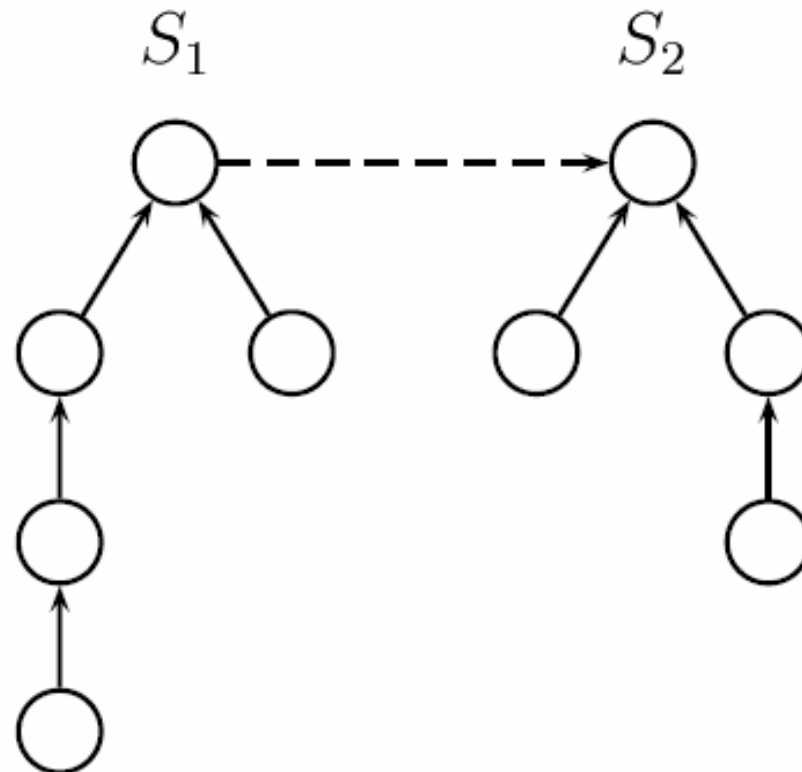
Δομή Λεξικού (Dictionary)

Ας υποθέσουμε ότι έχουμε ένα σύνολο S και θέλουμε να εκτελούνται γρήγορα οι λειτουργίες της εισαγωγής καινούργιου στοιχείου, διαγραφής παλαιού στοιχείου και ελέγχου για την ύπαρξη κάποιου στοιχείου στο S .

Θέλουμε δηλαδή να εκτελούνται γρήγορα οι διαδικασίες Insert, Delete και Member.

Αυτή τη γενικευμένη δομή δεδομένων την ονομάζουμε λεξικό (Dictionary).

Δομή Union-Find



Union - Find

```
function Union (i,j:integer(*set*)):integer;(*set*)  
begin  
    Parent[i]:=j; Union:=j  
end
```

```
function Find (i:integer(*element*)):integer;(*set*)  
begin  
    while Parent[i]>0 do i := Parent[i];  
    Find := i  
end
```

Union (με Balancing)

```
function Union (i,j:integer(*set*)): integer(*set*)  
var x:integer(*αρνητικός αριθμός στοιχείων του νέου συνόλου *)  
begin  
  x:=parent[i]+parent[j];  
  if |parent[i]| < |parent[j]| then  
    begin parent[i]:=j; parent[j]:=x; Union:=j end  
  else begin parent[j]:=i; parent[i]:=x; Union:=i end  
end
```


Βελτιωμένη Διαδικασία Εύρεσης (Path Compression)

```
function Union (i,j:integer(*set*)): integer(*set*)  
var x:integer(*αρνητικός αριθμός στοιχείων του νέου συνόλου *)  
begin  
  x:=parent[i]+parent[j];  
  if |parent[i]| < |parent[j]| then  
    begin parent[i]:=j; parent[j]:=x; Union:=j end  
  else begin parent[j]:=i; parent[i]:=x; Union:=i end  
end
```