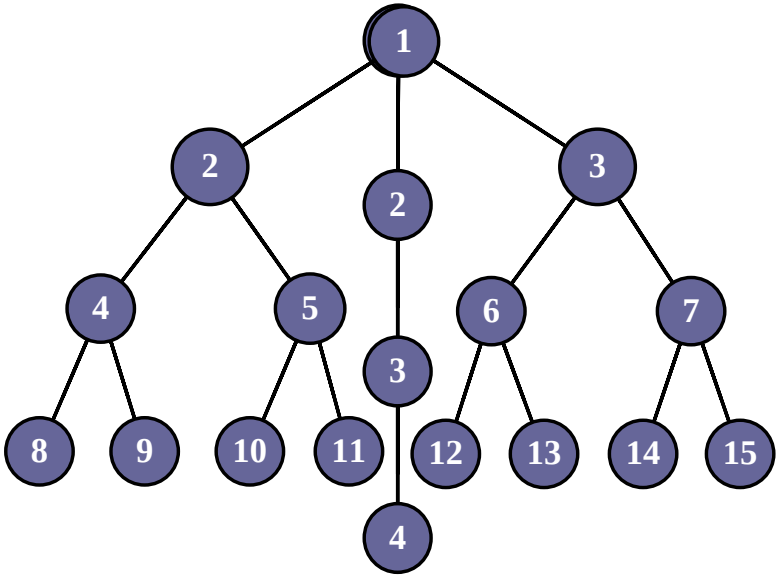
Δομές Δεδομένων & Αλγόριθμοι

Σωροί

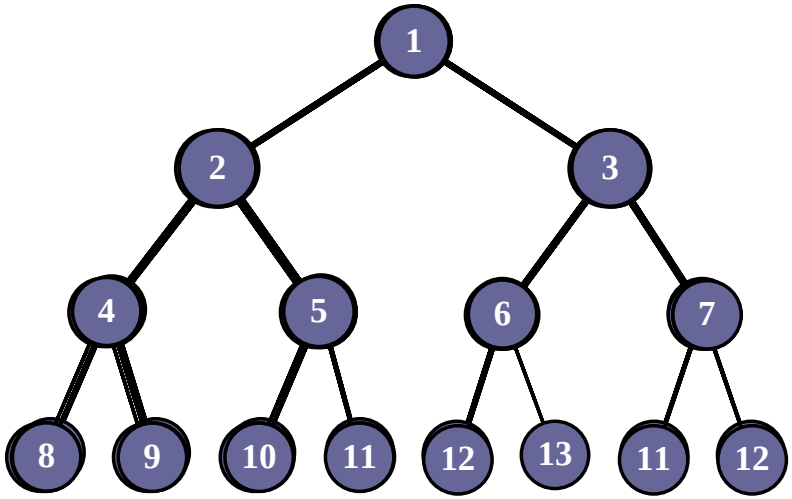
Δυαδικά Δέντρα

- $n(h)$: #κορυφών σε ΔΔ ύψους h.
  $h+1 \le n(h) \le 2^{h+1} - 1$
  - $h+1$ επίπεδα, ≥ 1 κορ. / επίπ.
  - $\le 2^i$ κορυφές στο επίπεδο i .
    $1 + 2 + \ldots + 2^h = 2^{h+1} - 1$
- $h(n)$: ύψος ΔΔ με n κορυφές:
  $\log_2(n+1) - 1 \le h(n) \le n - 1$
- **Γεμάτο** (full):
  - Κάθε κορυφή είτε φύλλο είτε 2 παιδιά.
- **Πλήρες** (complete) :
  - Γεμάτο και όλα τα επίπεδα συμπληρωμένα.
  - $n = 2^{h+1} - 1$



Σχεδόν Πλήρες

- Όλα τα επίπεδα συμπληρωμένα εκτός ίσως από τελευταίο που πληρώνεται από αριστερά προς τα δεξιά.
- $n(h)$: #κορυφών για ύψος h:
  $2^h \le n(h) \le 2^{h+1} - 1$
  - Πλήρες(h) : $2^{h+1} - 1$
  - Πλήρες($h-1$) + 1 : $(2^h - 1) + 1 = 2^h$.
- $h(n)$: ύψος για n κορυφές:
  $\log_2(n+1) - 1 \le h(n) \le \log_2 n$
- Ύψος : $h(n) = \lfloor \log_2 n \rfloor$
- #φύλλων $= \lceil n / 2 \rceil$

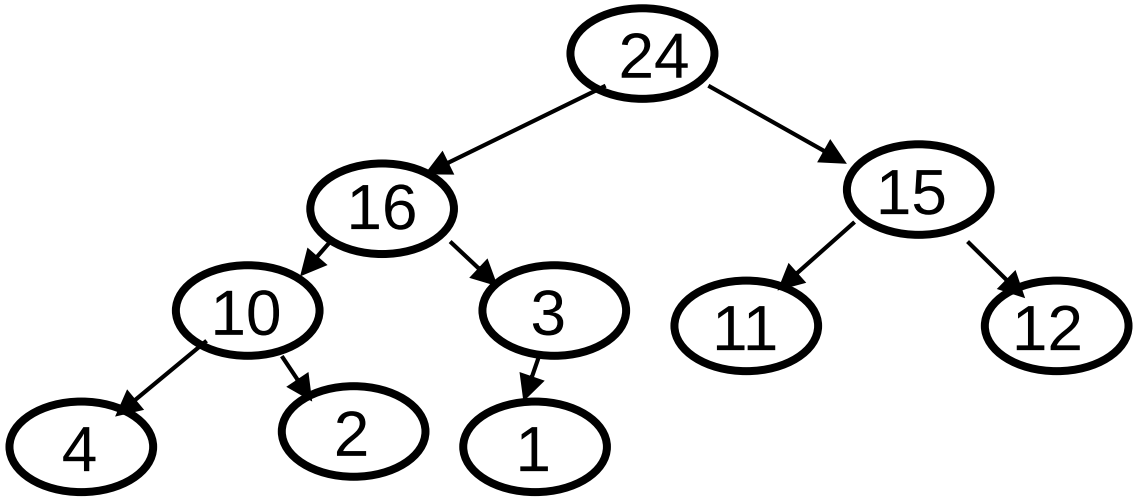


Ορισμοί

- Ένα **δέντρο μεγίστων (δένδρο ελαχίστων)** είναι ένα δένδρο, όπου η τιμή κάθε κόμβου είναι μεγαλύτερη (μικρότερη) ή ίση με των τιμών των παιδιών του

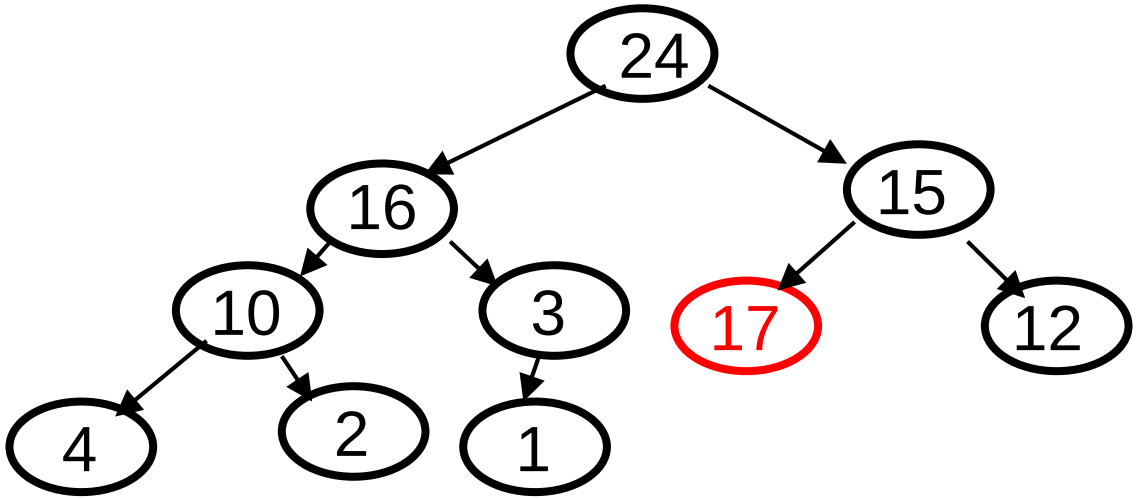
- Ένας **σωρός μεγίστων (σωρός ελαχίστων)** είναι ένα δέντρο μεγίστων (ελαχίστων) το οποίο είναι επίσης συμπληρωμένο δένδρο.
  - Έστω σωρός με αναπαράσταση θέσης κόμβων : η θέση της ρίζας είναι 1, του αριστερού παιδιού 2, του δεξιού παιδιού 3 κ.τ.λ.
  - Αφού ο σωρός είναι συμπληρωμένο δυαδικό δέντρο, ένας σωρός με n στοιχεία έχει ύψος άνω όριο log2(n+1). Συνεπώς αν μπορούμε να εισάγουμε και να διαγράψουμε σε χρόνο O*(ύψος)* τότε αυτές οι πράξεις έχουν πολυπλοκότητα O*(logn)*

Παράδειγμα Ι



Αυτό είναι σωρός

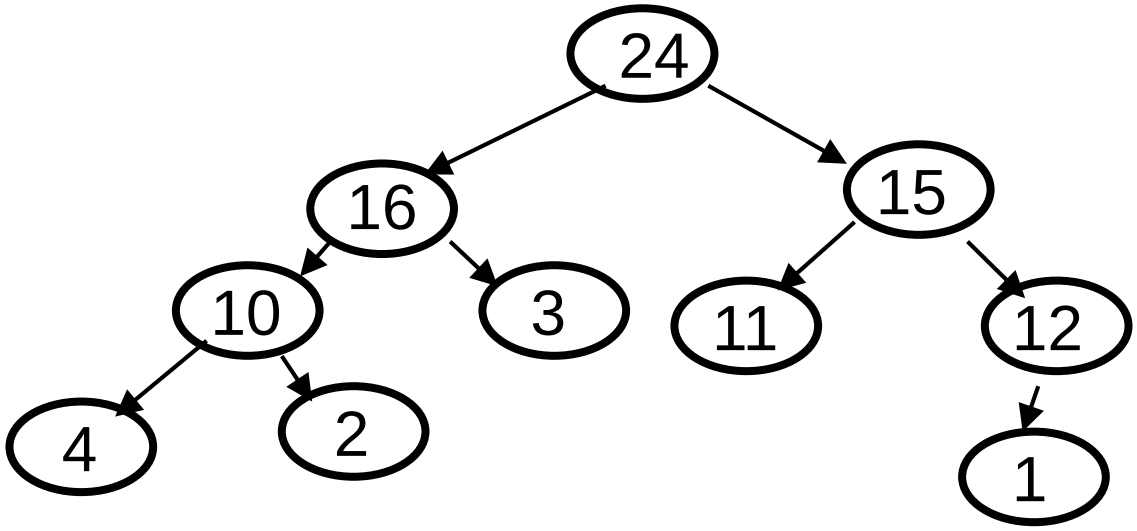
Παράδειγμα ΙΙ



Δεν είναι σωρός, γιατί δεν ικανοποιείται η ιδιότητα του σωρού

(17 > 15)

Παράδειγμα ΙΙΙ

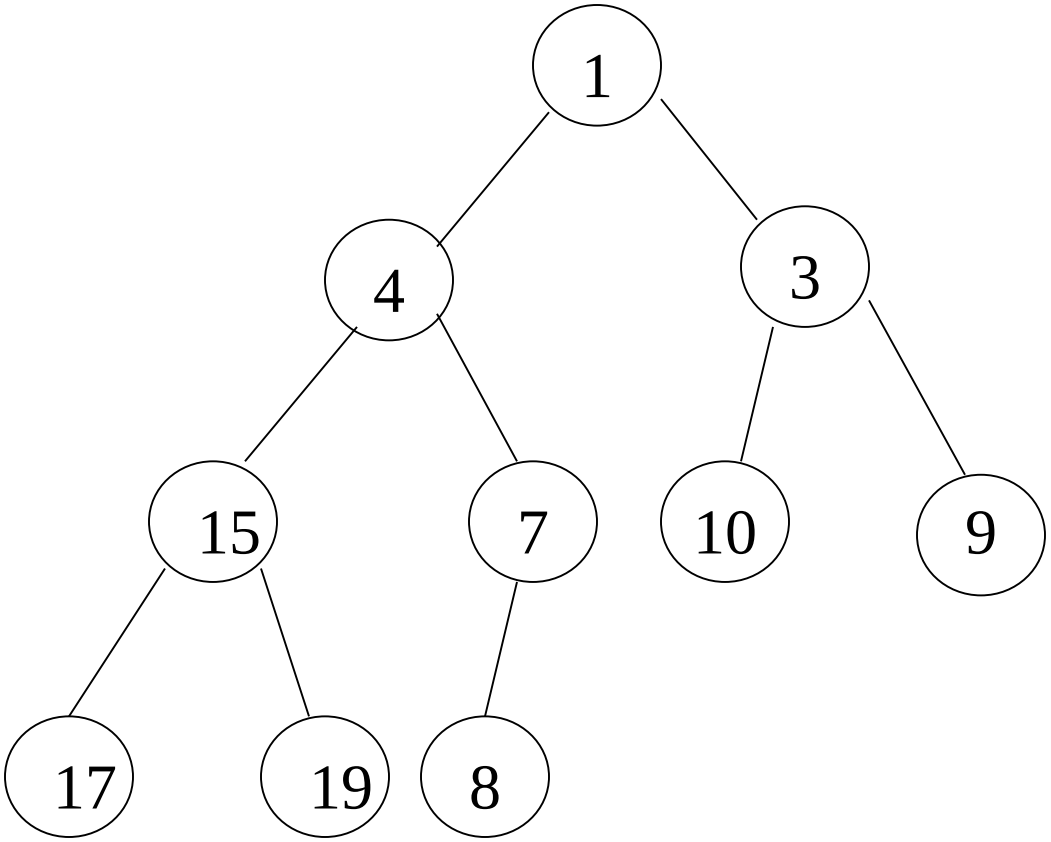


Δεν είναι σωρός, γιατί ο κόμβος με το 1 δεν είναι σε σωστή θέση.

Αναπαράσταση

-Με πίνακα
-Με δυναμική δέσμευση μνήμης

Αναπαράσταση με Πίνακα



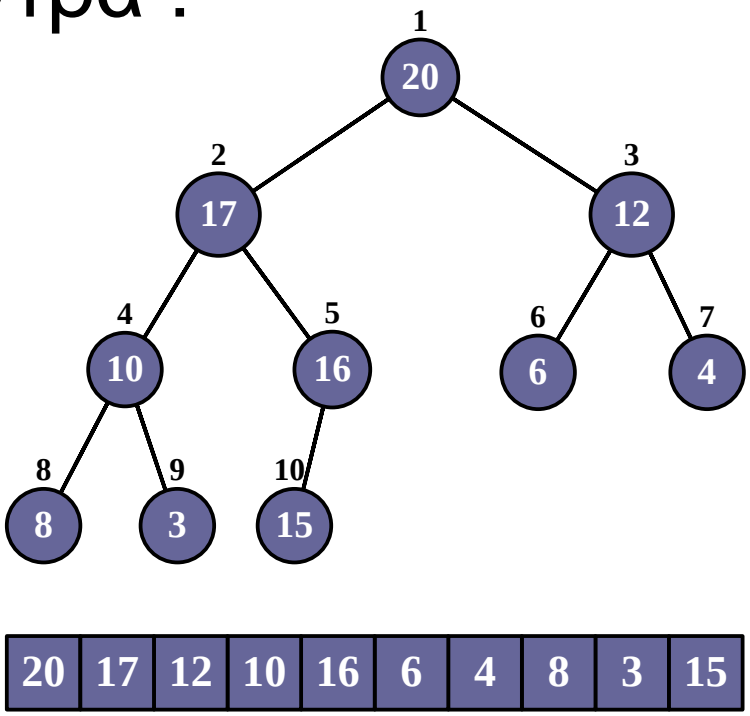
1	4	3	15	7	10	9	17	19	8

Αναπαράσταση με Πίνακα

- Για τον κόμβο που είναι στη θέση A[i],
  - Αριστερό παιδί στη θέση A[2i]
  - Δεξιό παιδί στη θέση A[2i+1]
  - Γονέας στη θέση A[$\lfloor i/2 \rfloor$]
- Και τα τρία τα βρίσκουμε σε χρόνο O(1)

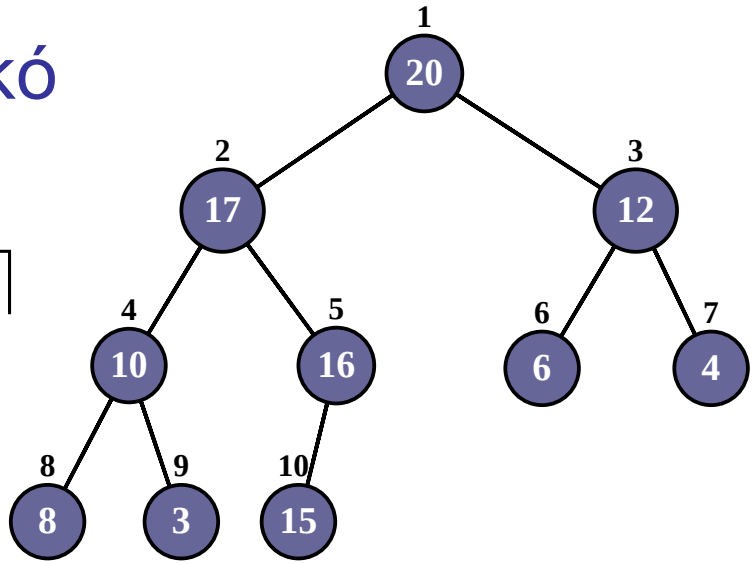
Αναπαράσταση

- **Δείκτες** σε παιδιά, πατέρα (δυναμική).
- **Σχεδόν πλήρη** δυαδικά δέντρα :
  - **Πίνακας** (στατική).
  - Αρίθμηση αριστερά → δεξιά και πάνω → κάτω.
  - **Ρίζα** : **Π[1]**
  - **Π[*i*]** : πατέρας **Π[*i* /2]**
    αριστερό παιδί **Π[2*i*]**
    δεξιό παιδί **Π[2*i*+1]**



Σωρός (heap)

- Δέντρο **μέγιστου** (ελάχιστου):
  Τιμές στις κορυφές και τιμή *κάθε* κορυφής ≥ (≤) τιμές παιδιών της.
- **Σωρός** : σχεδόν πλήρες δυαδικό δέντρο μέγιστου (ελάχιστου).
  - Ύψος $\Theta(\log n)$, #φύλλων = $\lceil n/2 \rceil$
- Πίνακας **A[ ]** ιδιότ. **σωρού** :
  $\forall i$ A[i] ≥ A[2i], A[2i+1].
- **Μέγιστο** : ρίζα
  Ελάχιστο : κάποιο φύλλο

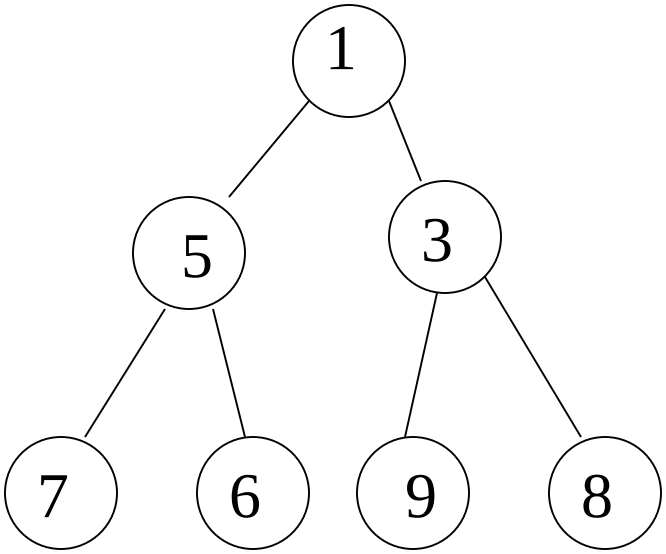


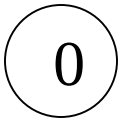
20	17	12	10	16	6	4	8	3	15

Εισαγωγή σε σωρό Ελαχίστων

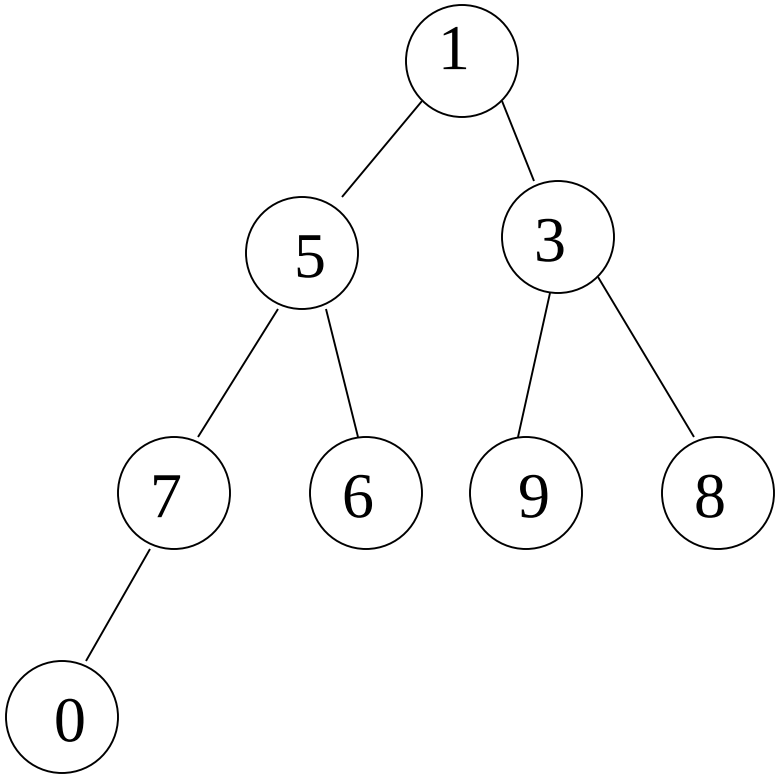
- Έστω ότι έχουμε ένα σωρό και θέλουμε να εισάγουμε ένα νέο στοιχείο.
- Μέθοδος
  - Βάζουμε το νέο στοιχείο στο τέλος του σωρού.
  - Πρέπει να βεβαιωθούμε ότι ικανοποιείται η ιδιότητα του σωρού.

Εισάγουμε το 0

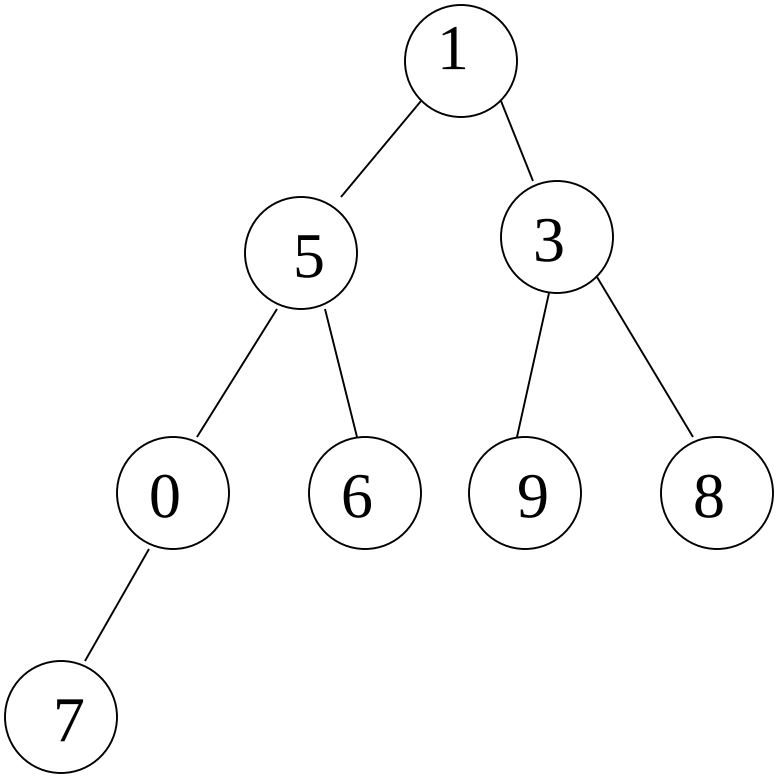




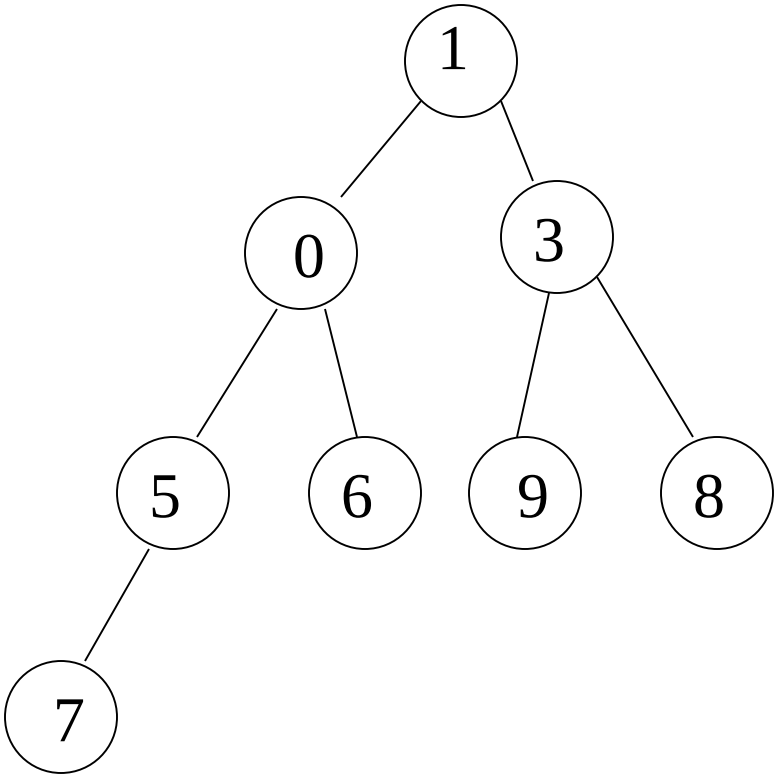
Τοποθετούμε το 0 στην τελευταία θέση του σωρού.



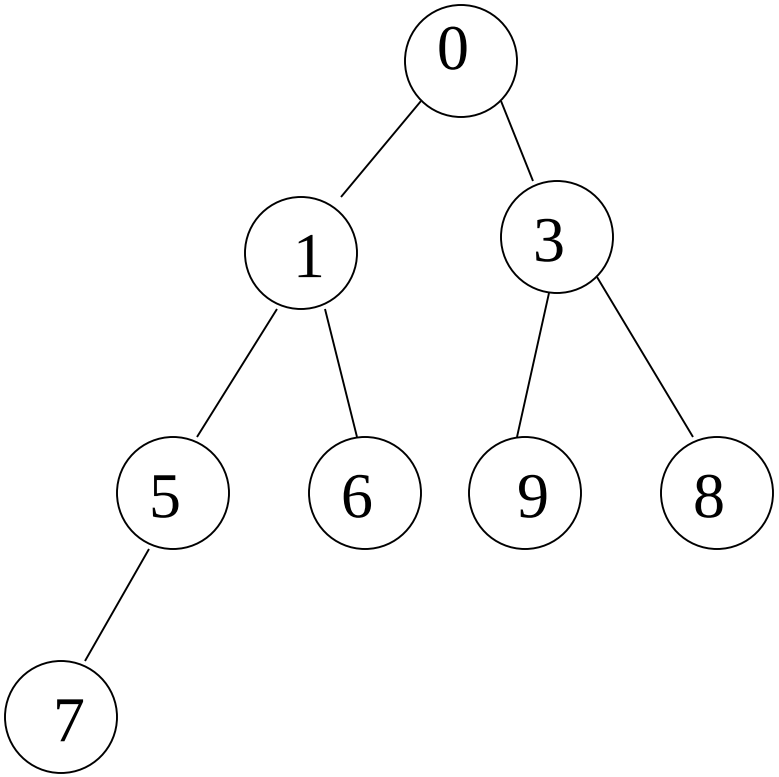
Ανταλλαγή 7 ←→ 0



Ανταλλαγή 5 ←→ 0



Ανταλλαγή 1 ←→ 0



Κόστος Εισαγωγής

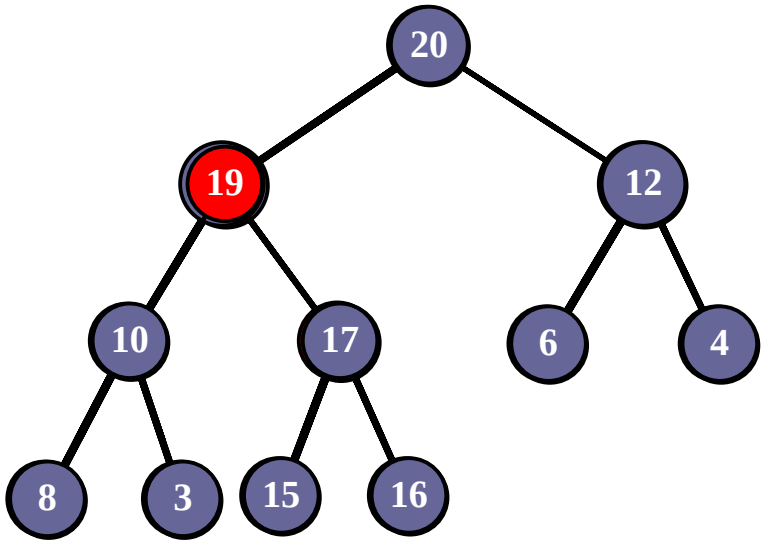
- Για την αναδιάρθρωση του σωρού δεν απαιτείται πάντα να φτάσουμε μέχρι τη ρίζα.

- Απαιτούμενο κόστος: O(*log* n) αφού το ύψος του σωρού που έχει n στοιχεία είναι O(*log* n)

Εισαγωγή: Αποκατάσταση Προς-τα-Πάνω

- insert(*k*) :
  - Εισαγωγή στο τέλος.
  - Ενόσω όχι σωρός, A[*i*] ↔ A[*i* / 2]

```
insert(int k) {
   A[++hs] = k;
   i = hs; p = i / 2;
   while ((i > 1) && (A[p] < A[i]))
   {     swap(A[p], A[i]);
         i = p; p = i / 2; }  }
```



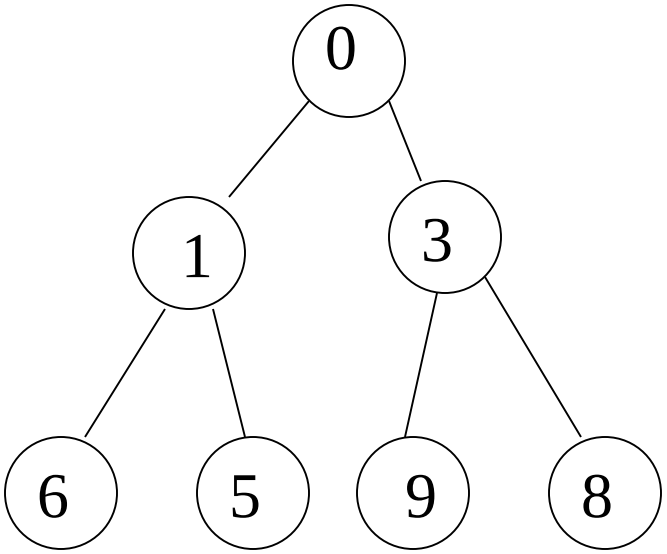
- Χρόνος για insert() : O(ύψος) = O(log n)
- Αύξηση προτεραιότητας : εισαγωγή (αποκατ. προς-τα-πάνω).
  Μείωση προτεραιότητας : διαγραφή (αποκατ. προς-τα-κάτω).

Διαγραφή της Κορυφής

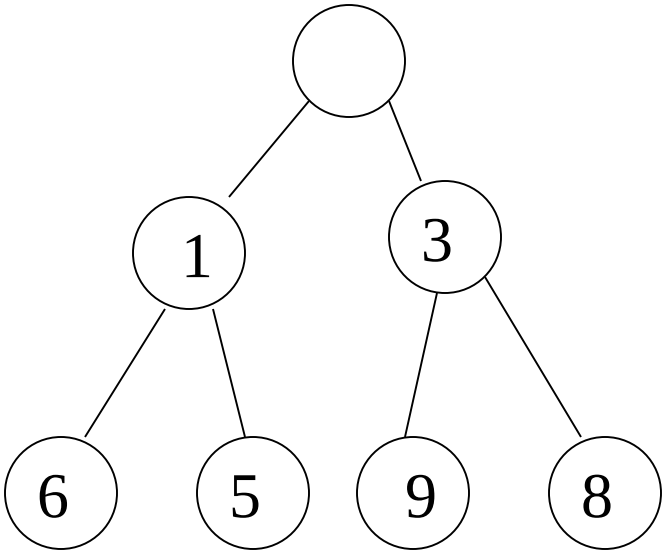
Ακολουθούμε την ακόλουθη μέθοδο:

- Διαγράφουμε το στοιχείο στην κορυφή και το αντικαθιστούμε με το τελευταίο στοιχείο του σωρού.
- Ακολουθεί αναδιάρθρωση του σωρού εάν απαιτείται.

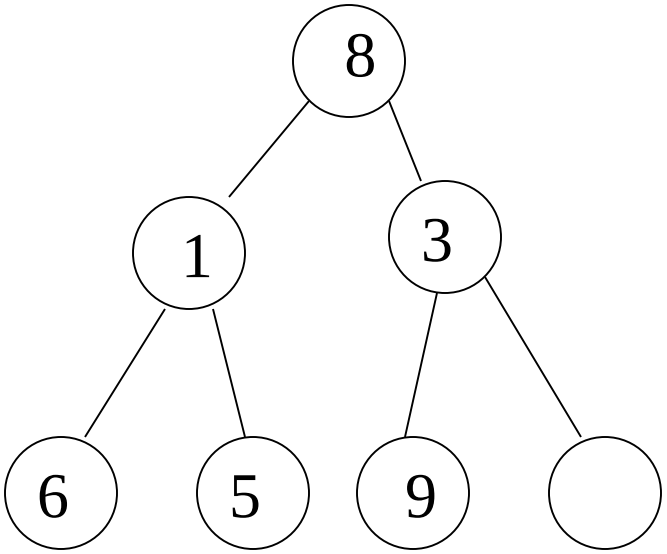
Διαγραφή του 0



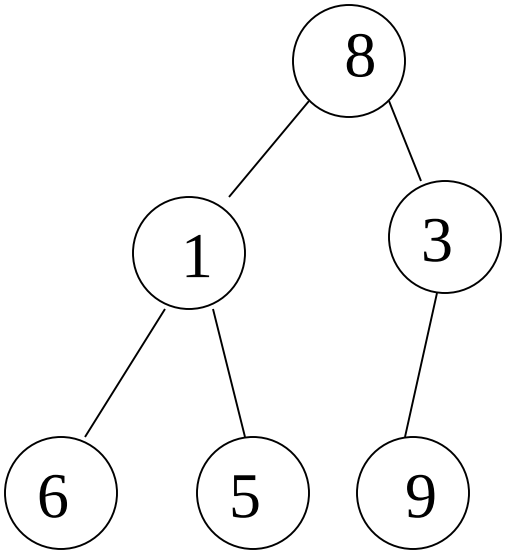
Διαγραφή του 0



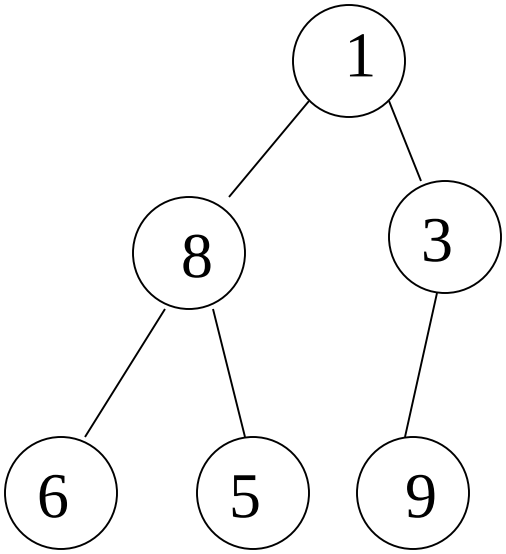
Αντικατάσταση με το τελευταίο στοιχείο (8)



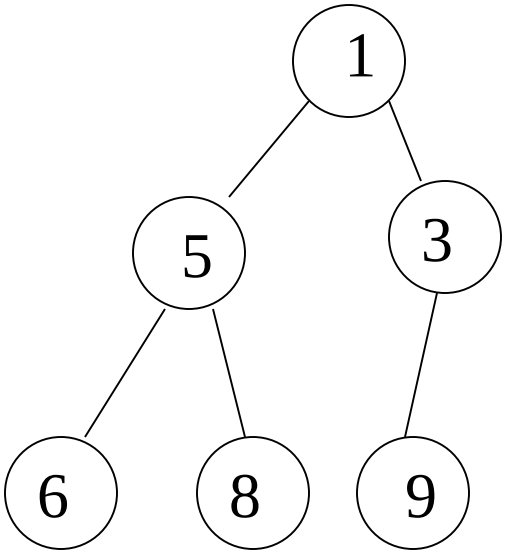
Διαγραφή του τελευταίου κόμβου που είναι άδειος.



Ανταλλαγή 8←→1



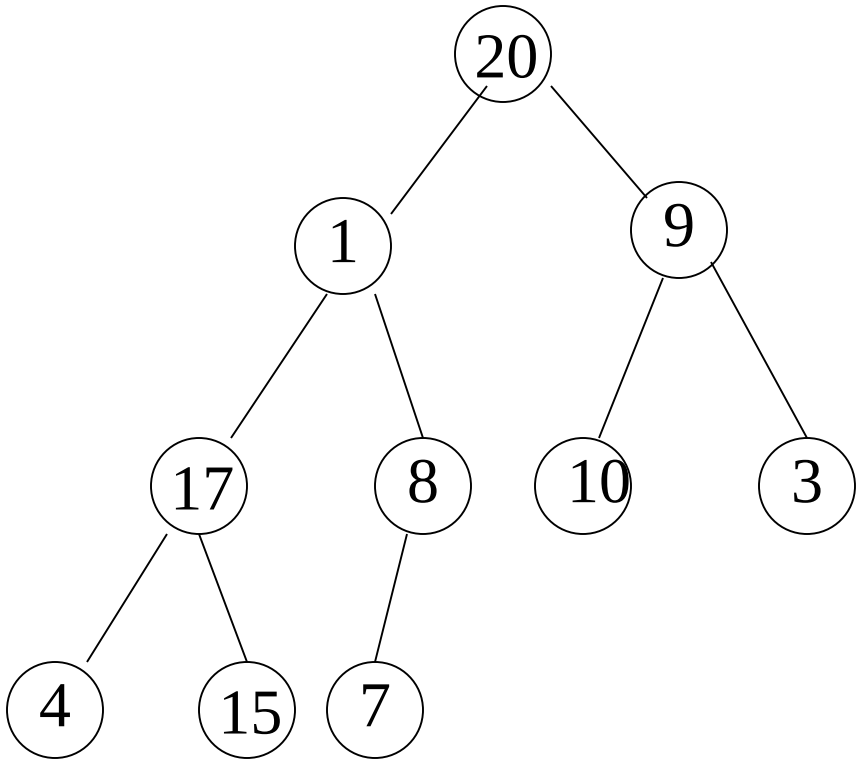
Ανταλλαγή 8←→5

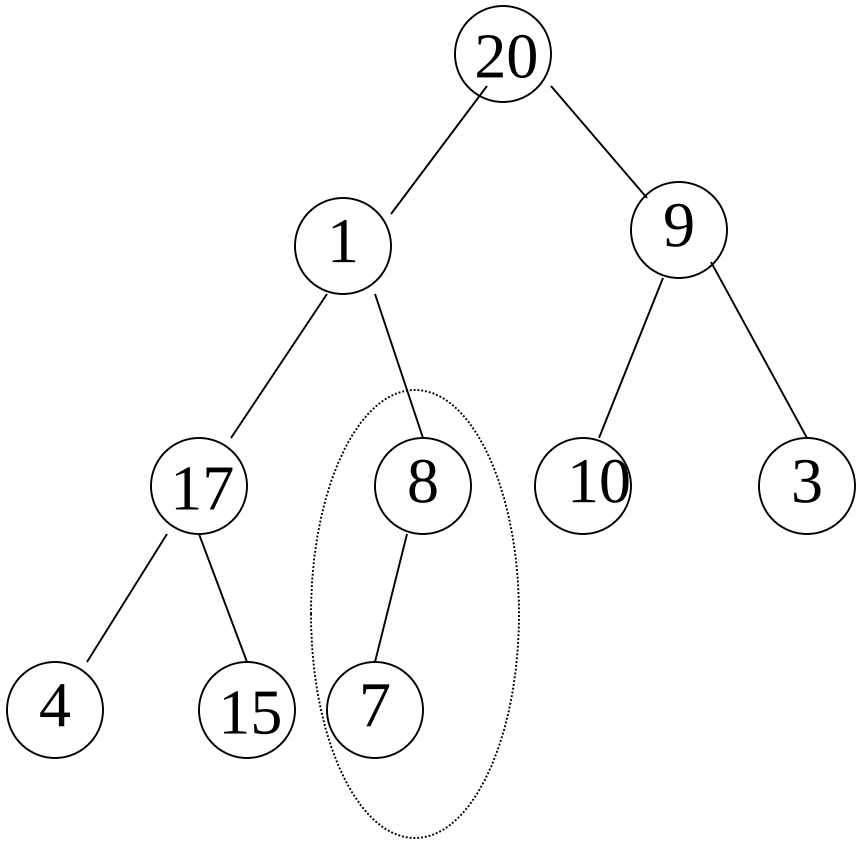


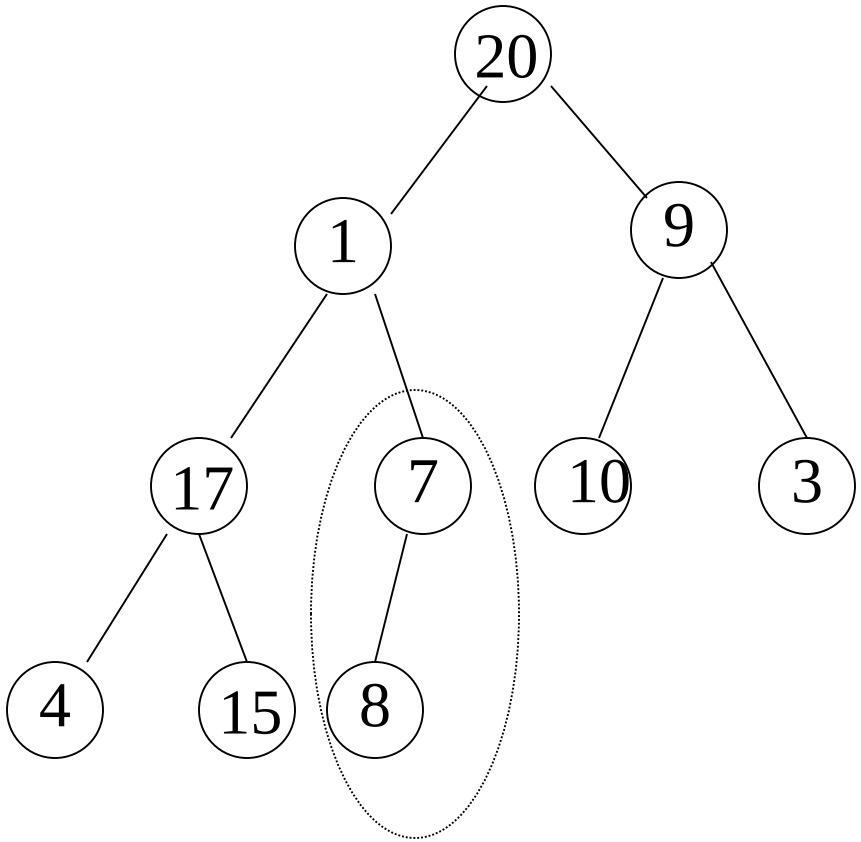
Είναι σωρός ;;;

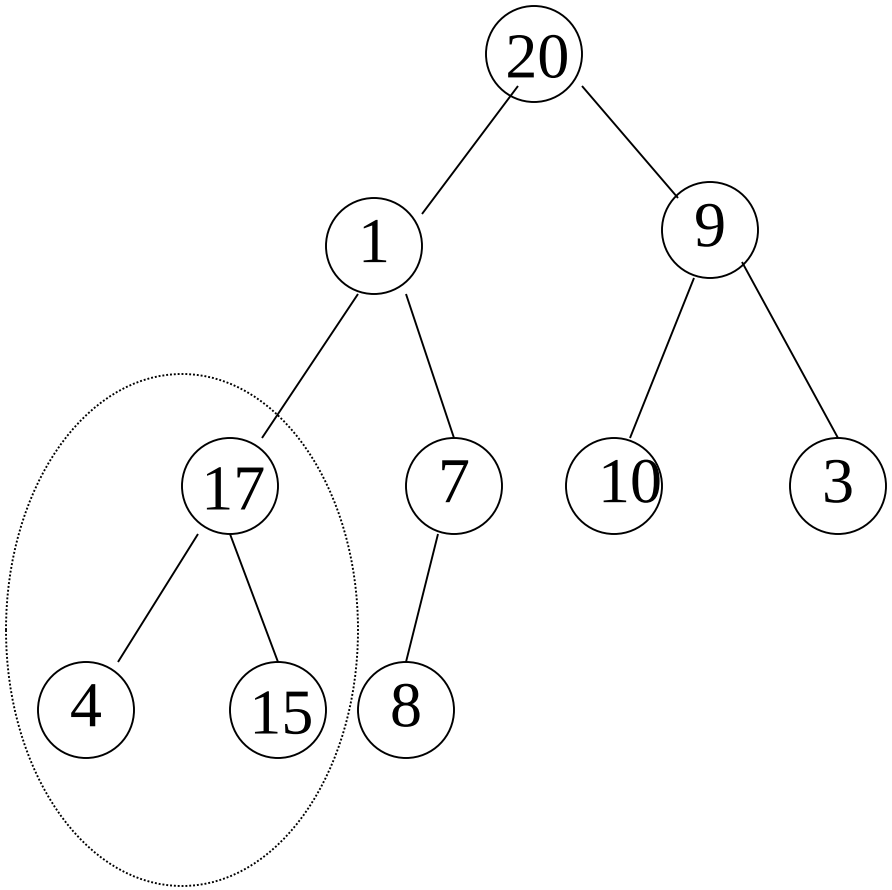
Κατασκευή Σωρού

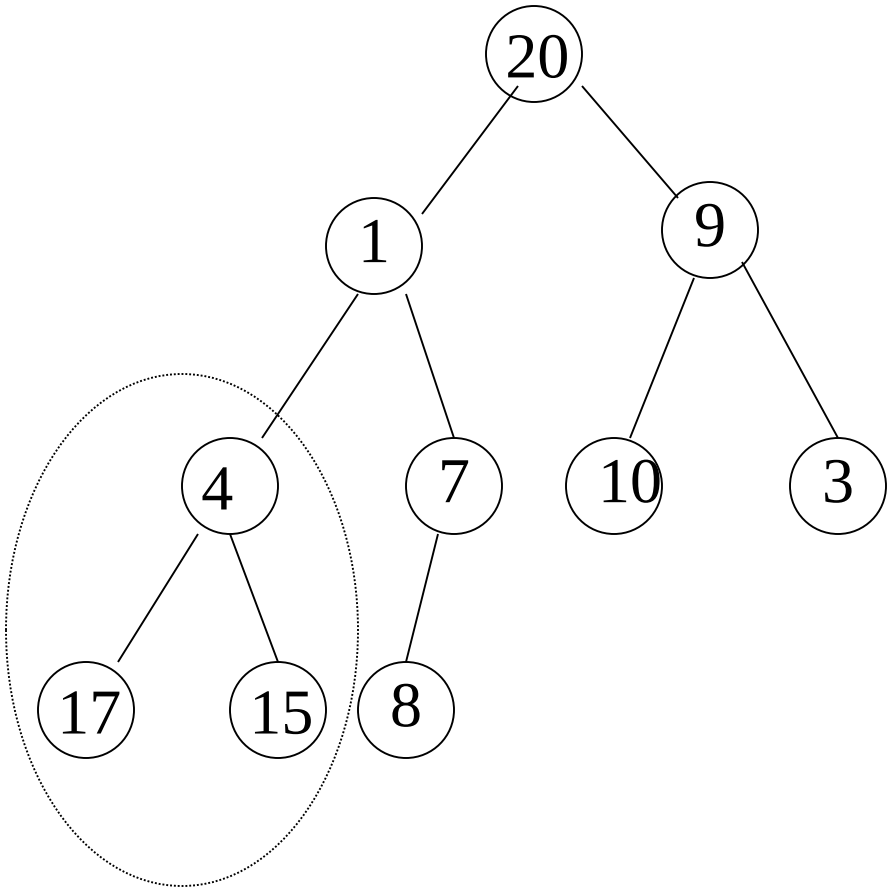
- Πολλές φορές έχουμε ένα σύνολο στοιχείων τα οποία ικανοποιούν τις δομικές απαιτήσεις του σωρού (έχουμε δηλαδή ένα σχεδόν πλήρες δυαδικό δένδρο).
- Πως θα κατασκευάσουμε σωρό ;

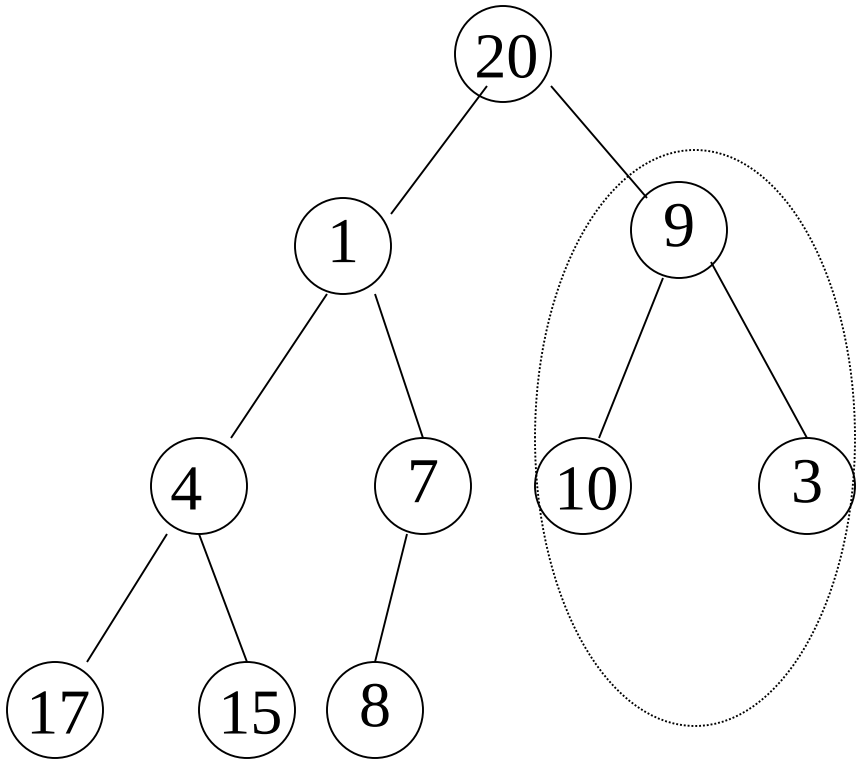

20
1
9
17
8
10
3
4
15
7

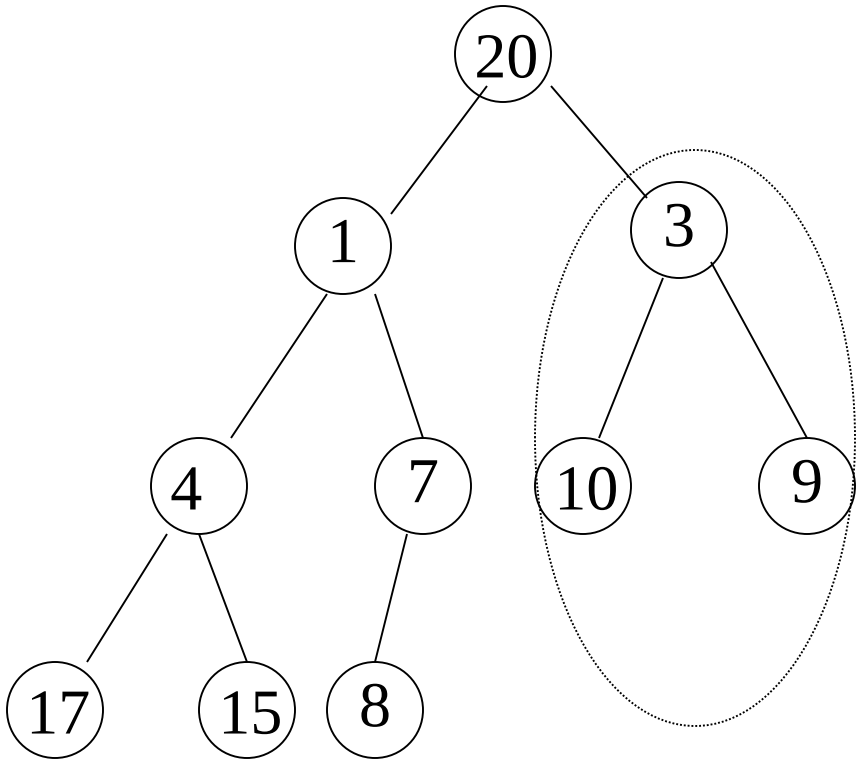

20
1
9
17
8
10
3
4
15
7

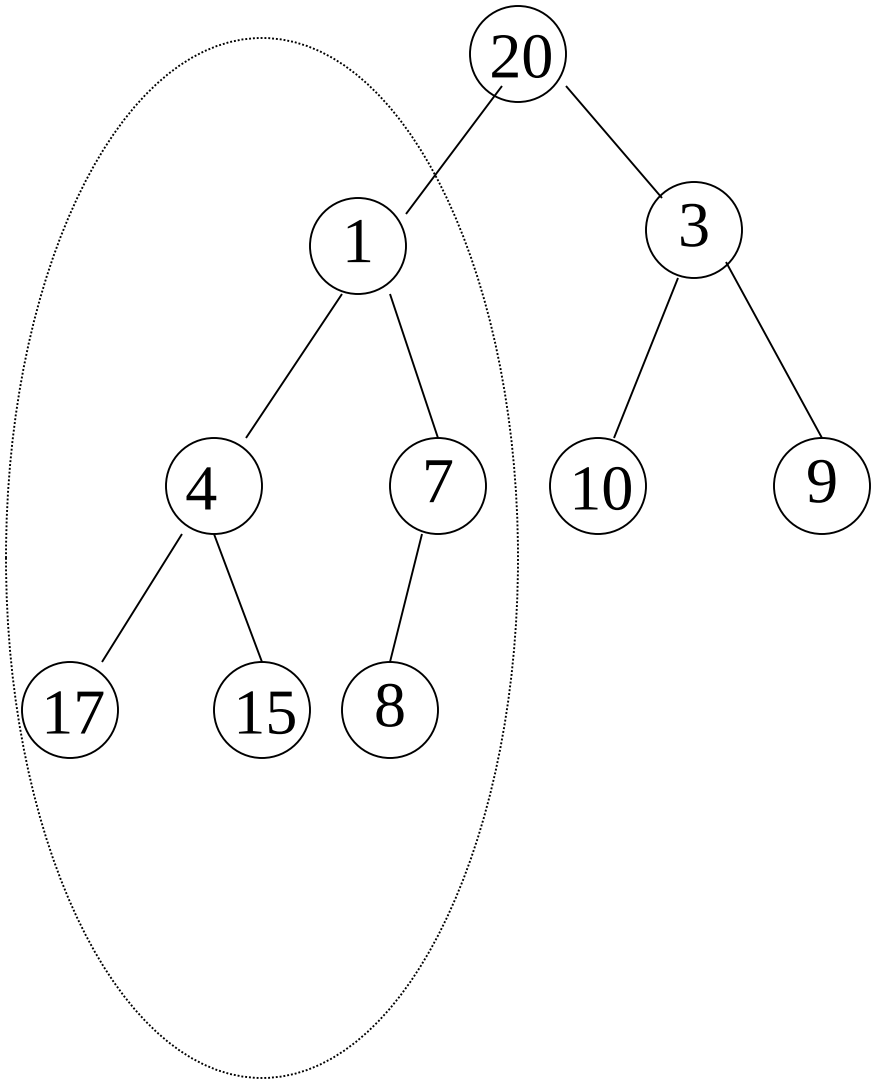

20
1
9
17
7
10
3
4
15
8

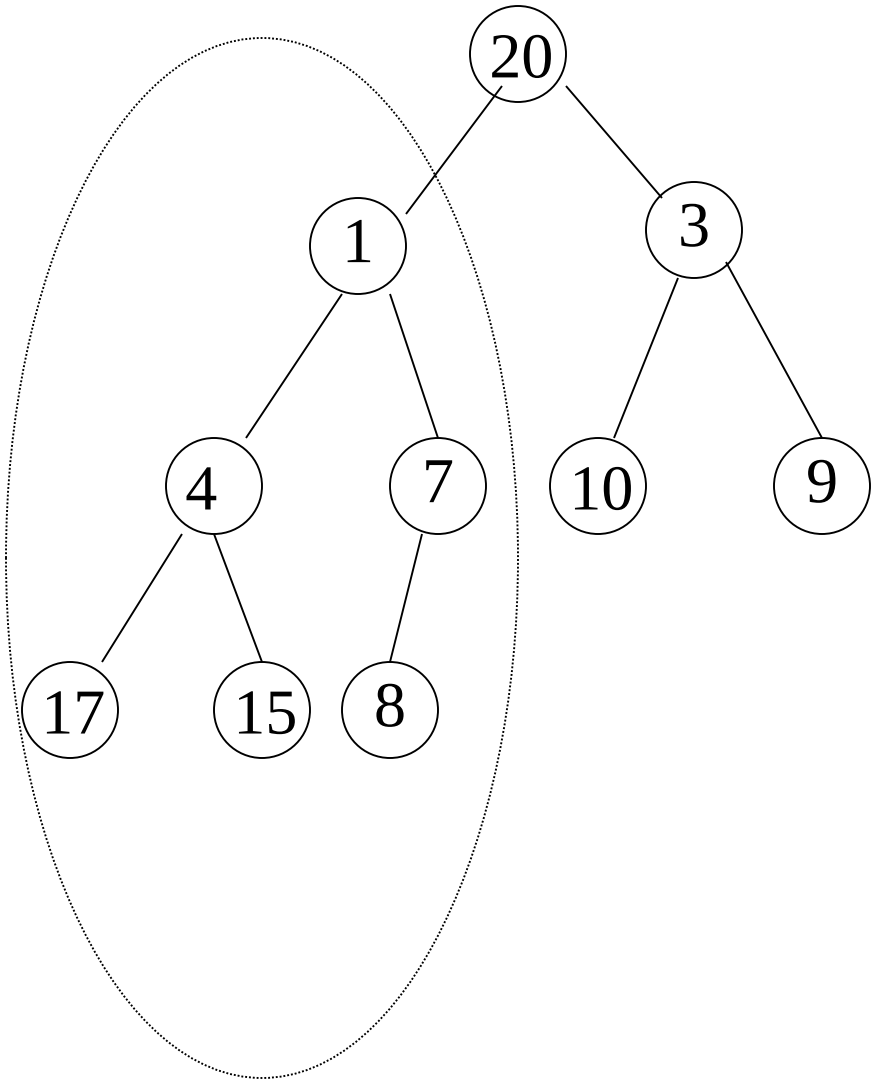

20
1
9
17
7
10
3
4
15
8

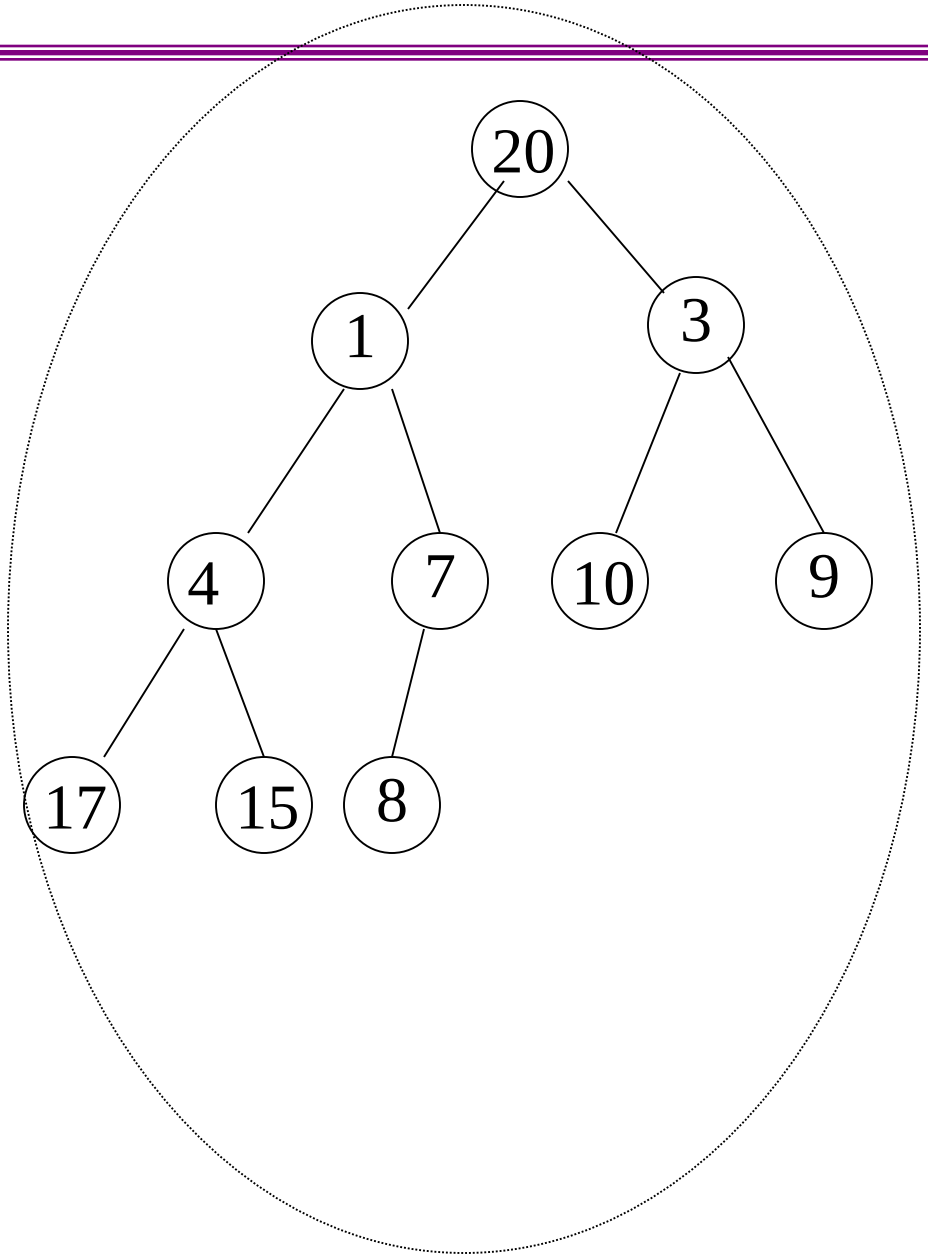

20
1
9
4
7
10
3
17
15
8

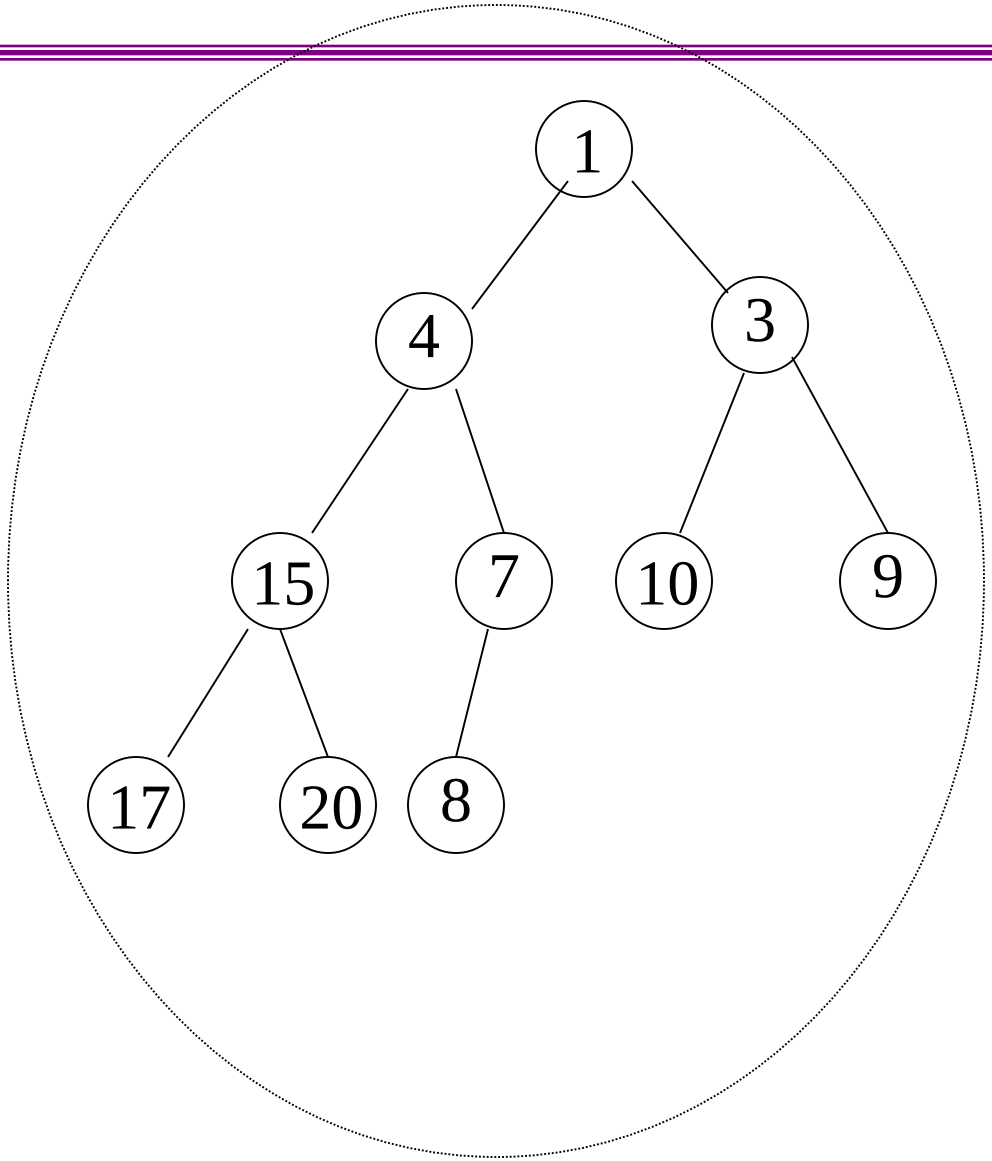

20
1
9
4
7
10
3
17
15
8


20
1
3
4
7
10
9
17
15
8


20
1
3
4
7
10
9
17
15
8


20
1
3
4
7
10
9
17
15
8


20
1
3
4
7
10
9
17
15
8


1
4
3
15
7
10
9
17
20
8

Κόστος Κατασκευής

Απλή εκτίμηση:

Αφού στη χειρότερη περίπτωση θα φτάσουμε μέχρι τη ρίζα και τα μισά στοιχεία πρέπει να μετακινηθούν, το συνολικό κόστος είναι

$n/2 * C * \log n = O(n \log n)$

Κόστος Κατασκευής

Καλύτερη εκτίμηση

- Κατά την κατασκευή έχουμε το πολύ
  - $n/2^2$ στοιχεία που μετακινούνται 1 θέση κάτω
  - $n/2^3$ στοιχεία που μετακινούνται 2 θέσεις κάτω
  - $n/2^4$ στοιχεία που μετακινούνται 3 θέσεις κάτω
  - … …
- $t(n) = O(1*n/2^2 + 2*n/2^3 + 3*n/2^4 + \ldots)$
- $=O((n/2)(1/2 + 2/2^2 + 3/2^3 + 4/2^4 + \ldots)$
- $=O(n)$

Απόδοση Σωρού

- Χώρος : $\Theta(1)$ (in-place)
- Χρόνοι :
  - createHeap : $\Theta(n)$
  - insert, deleteMax : $O(\log n)$
  - max, size, isEmpty : $\Theta(1)$
- Εξαιρετικά εύκολη υλοποίηση!
- Συμπέρασμα:
  - Γρήγορη και ευρύτατα χρησιμοποιούμενη ουρά προτεραιότητας.

Εφαρμογές Σωρού

- Ταξινόμηση με σωρό (HeapSort)
- Εκτέλεση εργασιών με προτεραιότητες
- Κωδικοποίηση Huffman

Heap-Sort

- **Αρχικοποίηση** : δημιουργία σωρού με n στοιχεία.
  - constructHeap() : χρόνος $\Theta(n)$.
- **Εξαγωγή μέγιστου** και τοποθέτηση στο τέλος ($n-1$ φορές).
  - deleteMax() : χρόνος $\Theta(\log n)$.
- **Χρόνος** : $\Theta(n) + n\,\Theta(\log n) = \Theta(n \log n)$.

```
hs = n;
constructHeap(n);
   for (i = n; i > 1; i--) {
      swap(A[1], A[i]); hs--;
       combine(1);  }
```

- Χρονική Πολυπλοκότητα Ταξινόμησης: $O(n \log n)$.

Heapsort

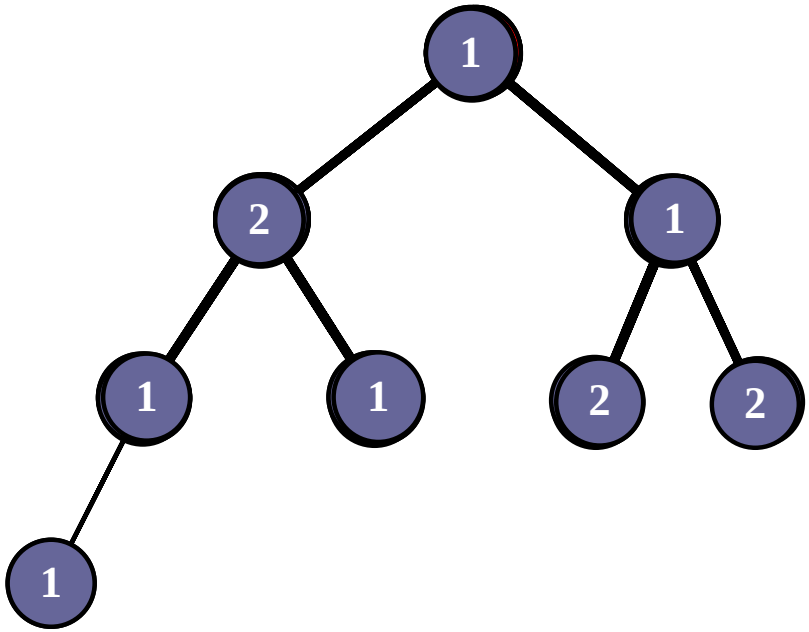
```
Heapsort(A)
{
      BuildHeap(A);
      for (i = length(A) downto 2)
      {
            Swap(A[1], A[i]);
            heap_size(A) -= 1;
            Heapify(A, 1);
      }
}
```

Ανάλυση Heapsort

- **`BuildHeap()`** απαιτεί O(*n*) κόστος
- Κάθε μία από τις *n*-1 κλήσεις της **`Heapify()`** απαιτεί O(log *n*) χρόνο
- Συνολικά έχουμε για την **`HeapSort()`**
  = O(*n*) + (*n* - 1) O(log *n*)
  = O(*n*) + O(*n* log *n*)
  = O(*n* log *n*)

Heap-Sort : Παράδειγμα

```
constructHeap(n);
for (i = n; i > 1; i--)  {
       swap(A[1], A[i]); hs--;
       combine(1);   }
```



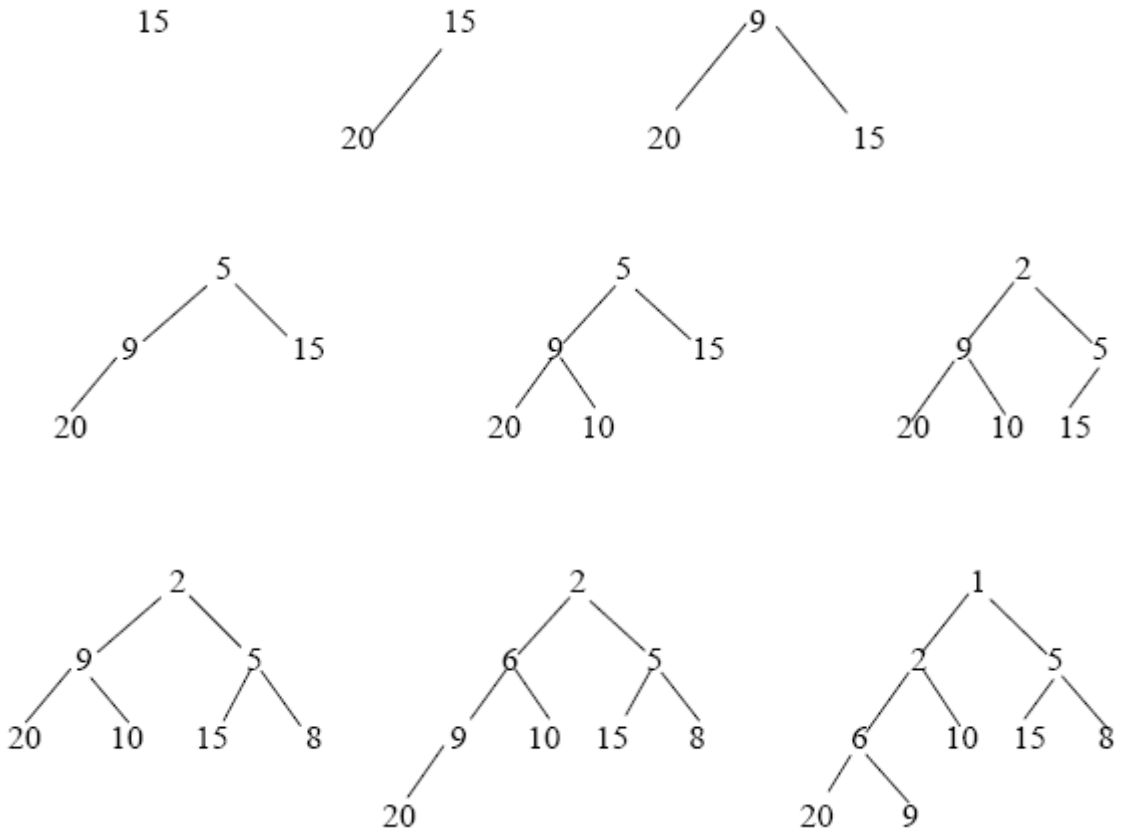
1	2	3	4	5	6	7	8

Παράδειγμα

Ξεκινώντας με ένα άδειο σωρό να εφαρμόσετε διαδοχικά εισαγωγή των στοιχείων 15, 20, 9, 5, 10, 2, 8, 6, 1, δείχνοντας το αποτέλεσμα της κάθε μιας από τις εισαγωγές.

Λύση

Ξεκινώντας με ένα άδειο ελαχίστων σωρό να εφαρμόσετε διαδοχικά εισαγωγή των στοιχείων 15, 20, 9, 5, 10, 2, 8, 6, 1, δείχνοντας το αποτέλεσμα της κάθε μιας από τις εισαγωγές.



Κωδικοποίηση Huffman

Βασικά Σημεία:

- Ο κύριος στόχος ενός κωδικοποιητή είναι η αντιστοίχιση μικρών κωδικών σε συχνά εμφανιζόμενα σύμβολα και μεγάλων κωδικών σε σπάνια εμφανιζόμενα σύμβολα.
- Ο χρόνος κωδικοποίησης και αποκωδικοποίησης είναι σημαντικός. Μερικές φορές προτιμούμε να έχουμε μικρότερο λόγο συμπίεσης προκειμένου να κερδίσουμε σε χρόνο (π.χ. WinZIP).

Κωδικοποίηση Huffman

Έστω τα σύμβολα A,B,C,D με τους εξής κωδικούς:

Code(‘A’) = 0
Code(‘B’) = 000
Code(‘C’) = 11
Code(‘D’) = 1

DDDAAA

DCB

CDAAA

DDDB

Ο κωδικός 111000 σε ποια σειρά χαρακτήρων αντιστοιχεί;

Κωδικοποίηση Huffman

Βασική προϋπόθεση:

Μετά τη φάση της κωδικοποίησης κανένας κωδικός δεν πρέπει να αποτελεί πρόθεμα (prefix) άλλου κωδικού.

Κωδικοποίηση Huffman

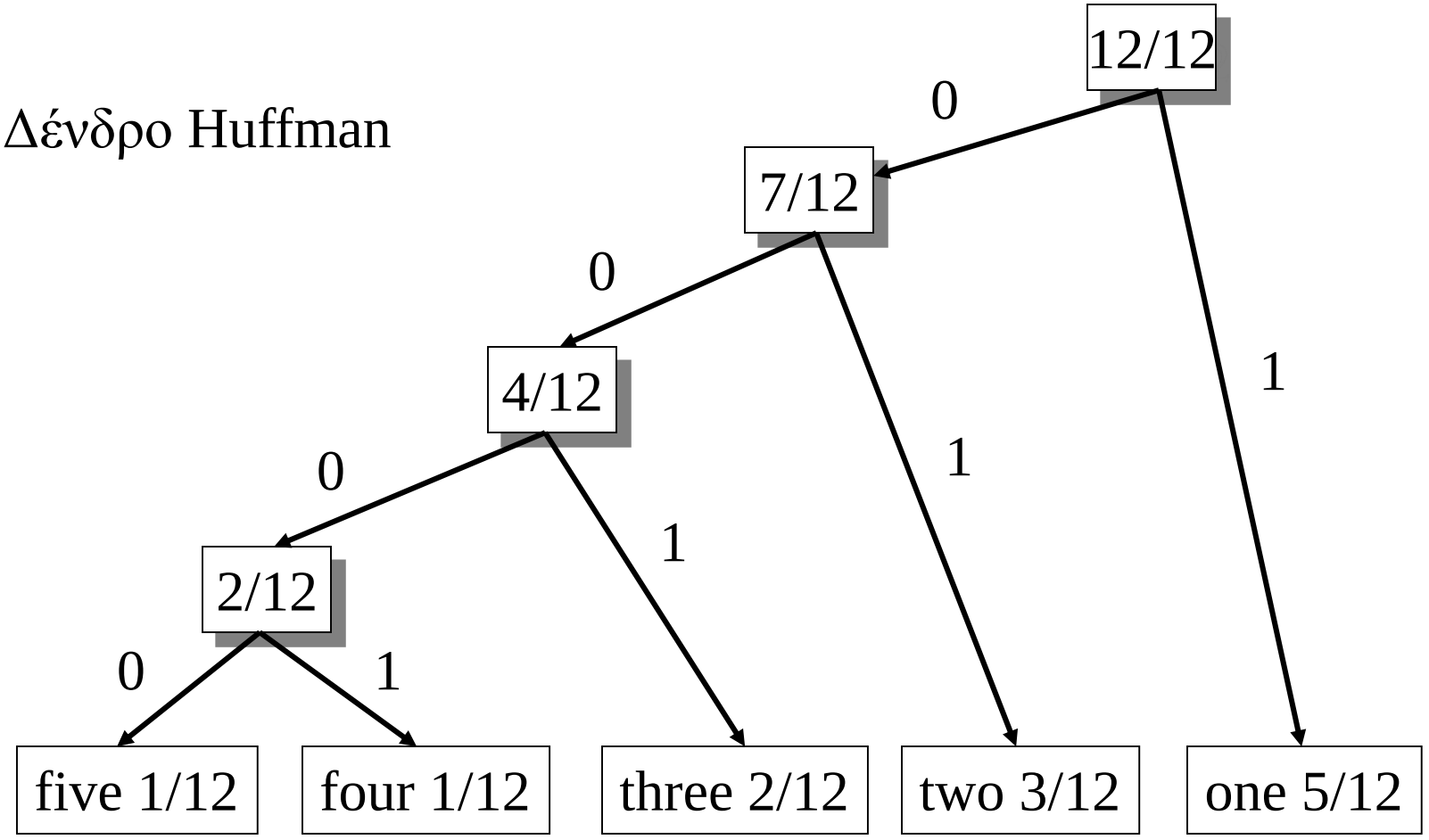
Έστω το ακόλουθο κείμενο:
one two three one two one one one two three four five

one:	5/12
two:	3/12
three:	2/12
four:	1/12
five:	1/12

Συχνότητες εμφάνισης λέξεων

Κωδικοποίηση Huffman

Δένδρο Huffman



Κωδικοποίηση Huffman

Μετά την κωδικοποίηση προκύπτουν οι εξής κωδικοί:

- five: 0000
- four: 0001
- three: 001
- two: 01
- one: 1

Τι παρατηρούμε;

Κωδικοποίηση Huffman

Τι συμπίεση επιτυγχάνουμε για το παράδειγμα;

one two three one two one one one two three four five

Απαιτούνται 42*8 = 336 bits για το αρχικό κείμενο (χωρίς τους κενούς χαρακτήρες)

Απαιτούνται 25 bits για το συμπιεσμένο κείμενο

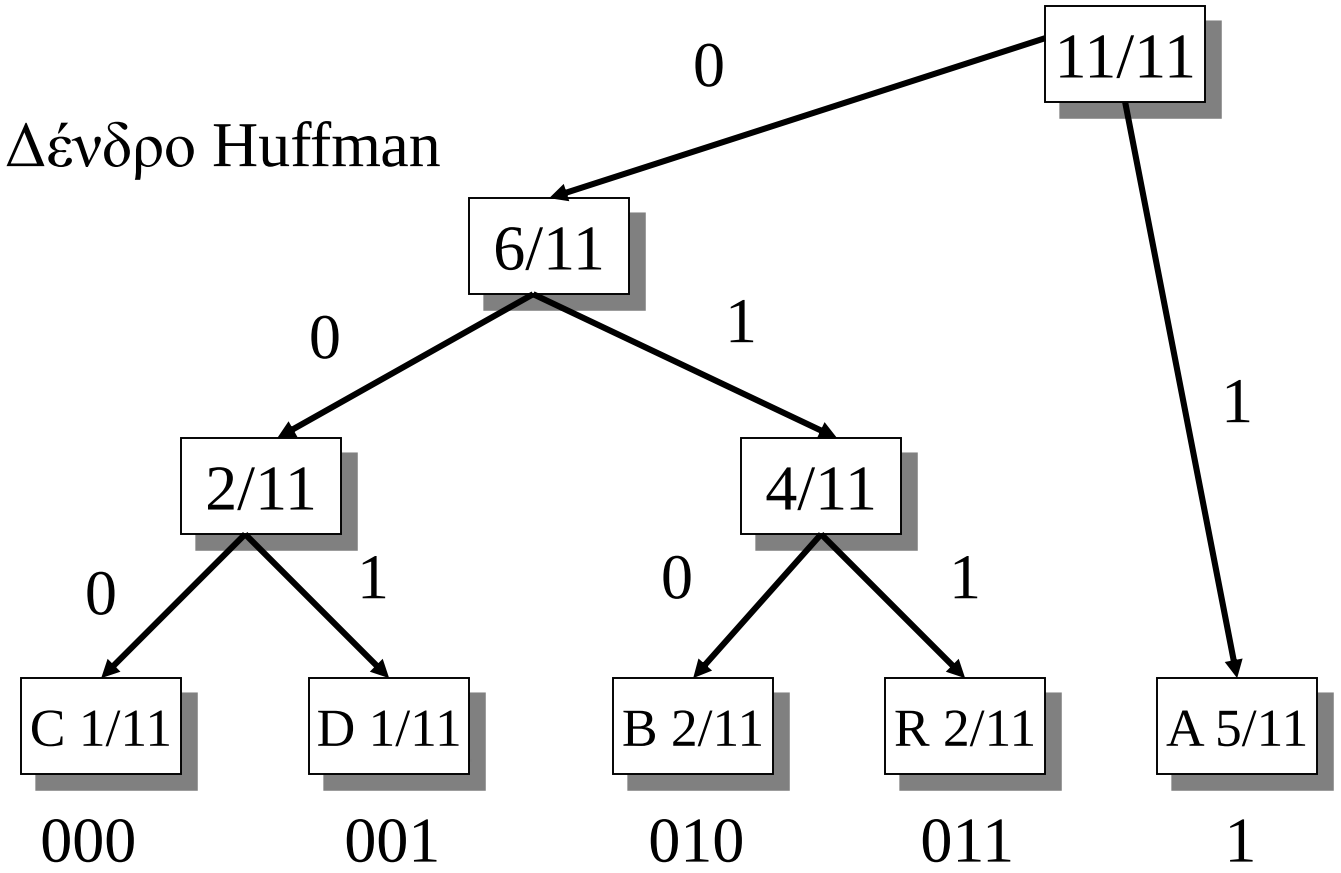
Κωδικοποίηση Huffman

Έστω το ακόλουθο κείμενο

ABRACADABRA

A	5/11
B	2/11
C	1/11
D	1/11
R	2/11

Κωδικοποίηση Huffman



Κωδικοποίηση Huffman

130.000.000 μετρήσεις σε τακτά διαστήματα.
Κάθε μέτρηση προσεγγίζεται με τον "πλησιέστερο" αριθμό από ένα σύνολο Γ.
Έστω ότι το Γ είχε τέσσερα μόνο σύμβολα, A, B, C, D

- A 70.000.000 φορές
- B 3.000.000 φορές
- C 20.000.000 φορές
- D 37.000.000 φορές

```
Algorithm Huffman-coding(f: array of integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
   { i= deletemin(H); j=deletemin(H);
     left(next) = i; right(next) = j;
     f(next) = f(i) + f(j);
     insert(H, next);
     next++;
   }
}
```

Για ευκολία στους υπολογισμούς μας θα υποθέσουμε ότι:

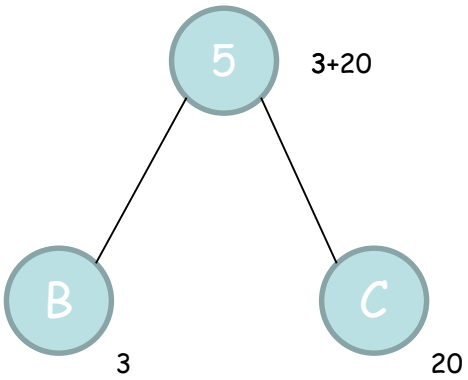
- A=70
- B=3
- C=20
- D=37

B=3
C=20
D=37
A=70

```
Algorithm Huffman-coding(f: array of
integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
  { i= deletemin(H); j=deletemin(H);
    left(next) = i; right(next) = j;
    f(next) = f(i) + f(j);
    insert(H, next);
    next++;
  }
}
```

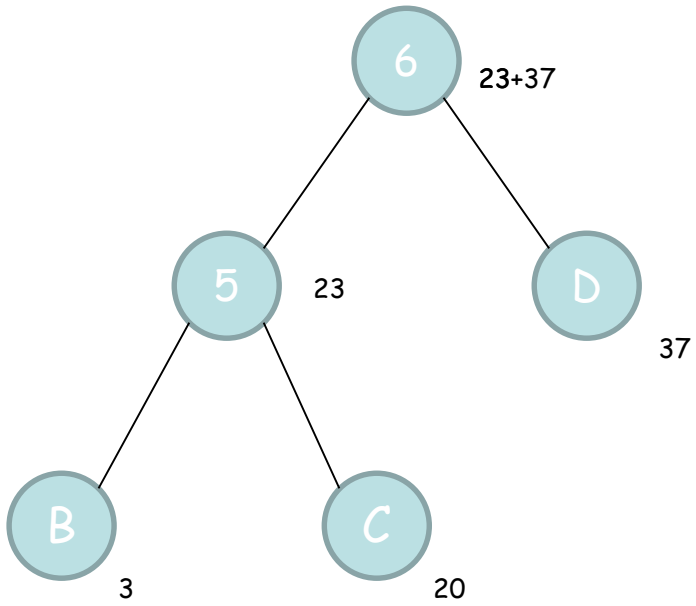
B=3
C=20
D=37
A=70

```
Algorithm Huffman-coding(f: array of integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
  { i= deletemin(H); j=deletemin(H);
    left(next) = i; right(next) = j;
    f(next) = f(i) + f(j);
    insert(H, next);
    next++;
  }
}
```



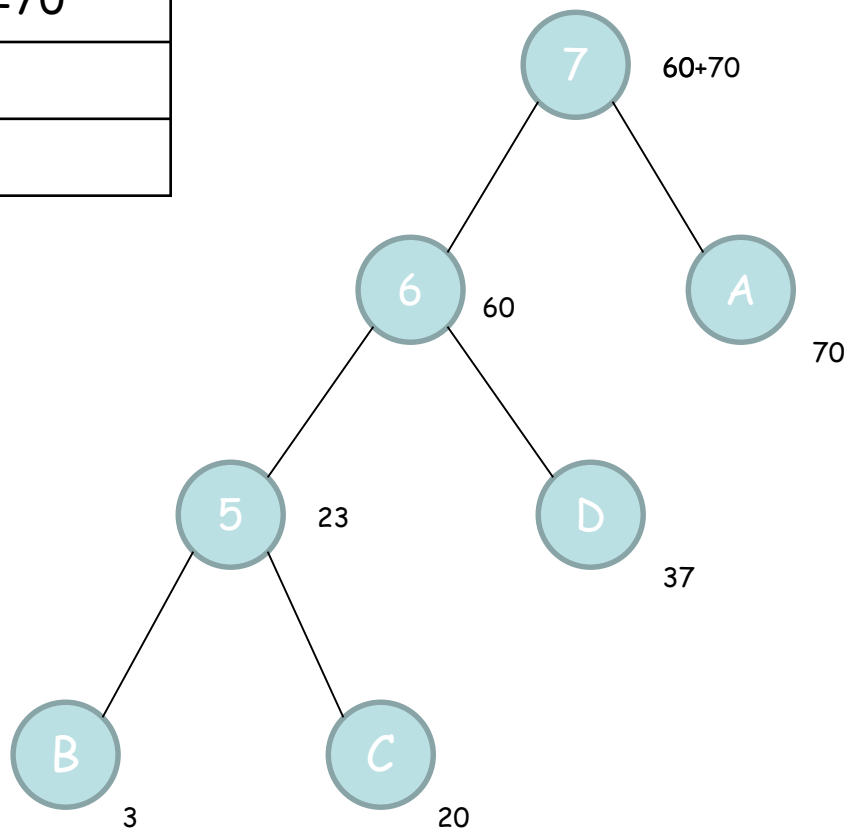
```
Algorithm Huffman-coding(f: array of
integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
  { i= deletemin(H); j=deletemin(H);
    left(next) = i; right(next) = j;
    f(next) = f(i) + f(j);
    insert(H, next);
    next++;
  }
}
```

{5}=23
D=37
A=70

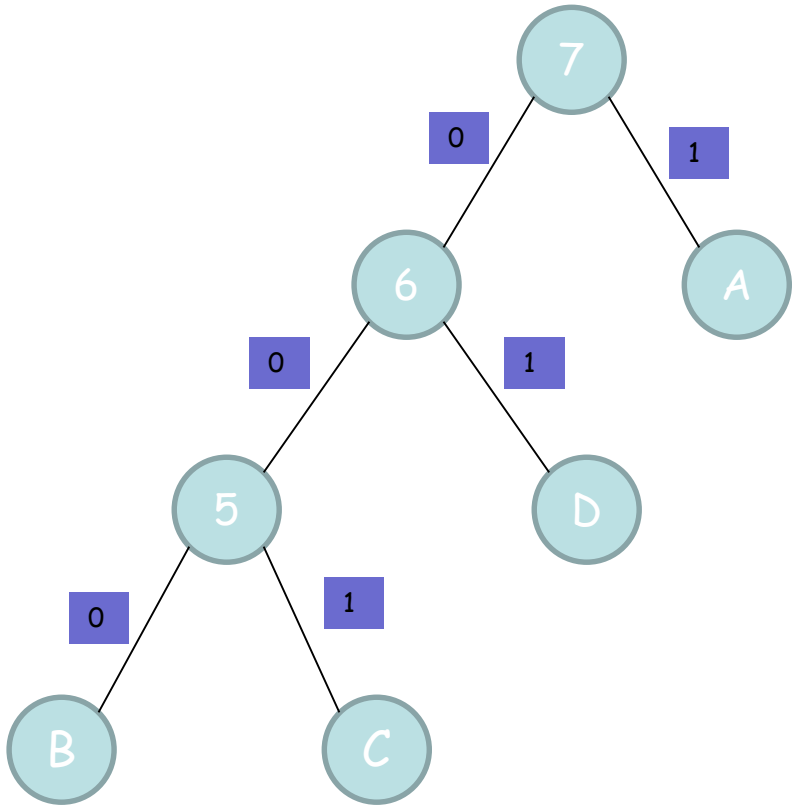


```
Algorithm Huffman-coding(f: array of
integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
  { i= deletemin(H); j=deletemin(H);
    left(next) = i; right(next) = j;
    f(next) = f(i) + f(j);
    insert(H, next);
    next++;
  }
}
```

{6}=60
A=70



Με '0' στα αριστερά και '1' στα δεξιά θα έχουμε




```
Algorithm Huffman-coding(f: array of
integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
   { i= deletemin(H); j=deletemin(H);
     left(next) = i; right(next) = j;
     f(next) = f(i) + f(j);
     insert(H, next);
     next++;
   }
}
```

Άρα

- A=1
- B=000
- C=001
- D=01

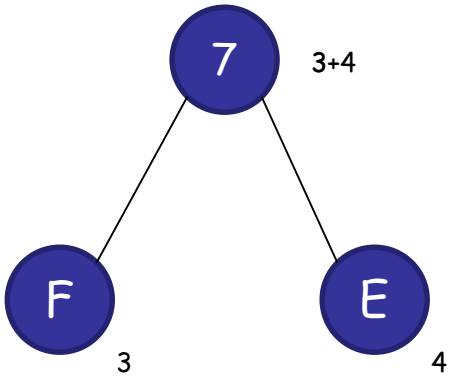
Κωδικοποίηση Huffman

A=40
B=30
C=6
D=5
E=4
F=3

```
Algorithm Huffman-coding(f: array of integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
   { i= deletemin(H); j=deletemin(H);
     left(next) = i; right(next) = j;
     f(next) = f(i) + f(j);
     insert(H, next);
     next++;
   }
}
```

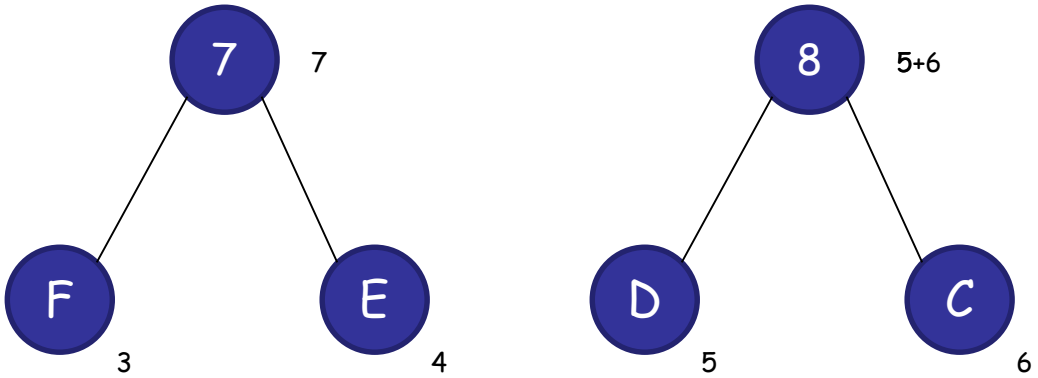
A=40
B=30
C=6
D=5
E=4
F=3

```
Algorithm Huffman-coding(f: array of integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
   { i= deletemin(H); j=deletemin(H);
     left(next) = i; right(next) = j;
     f(next) = f(i) + f(j);
     insert(H, next);
     next++;
   }
}
```



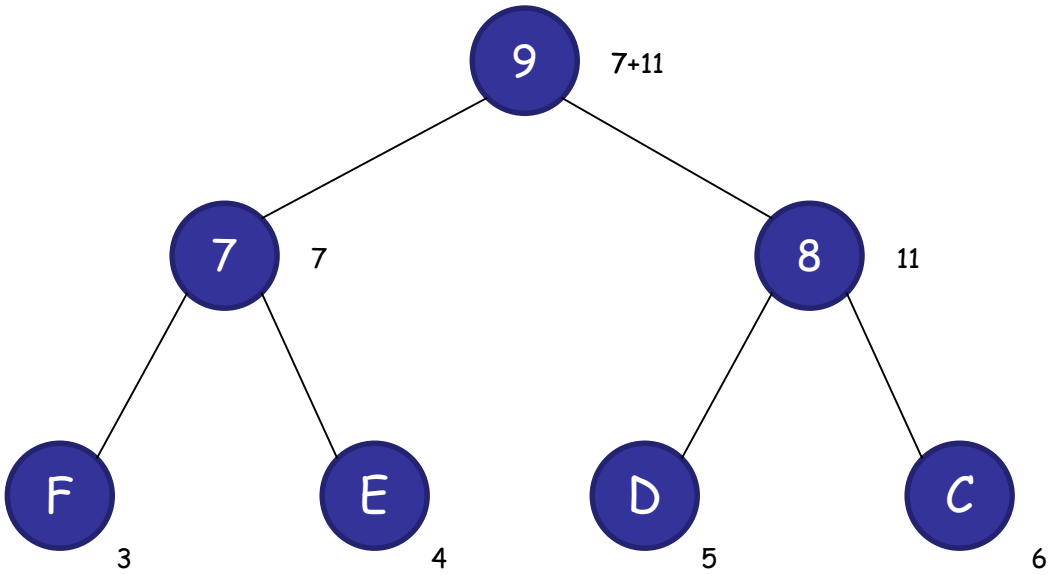
A=40
B=30
{7}=7
C=6
D=5

```
Algorithm Huffman-coding(f: array of integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
   { i= deletemin(H); j=deletemin(H);
     left(next) = i; right(next) = j;
     f(next) = f(i) + f(j);
     insert(H, next);
     next++;
   }
}
```



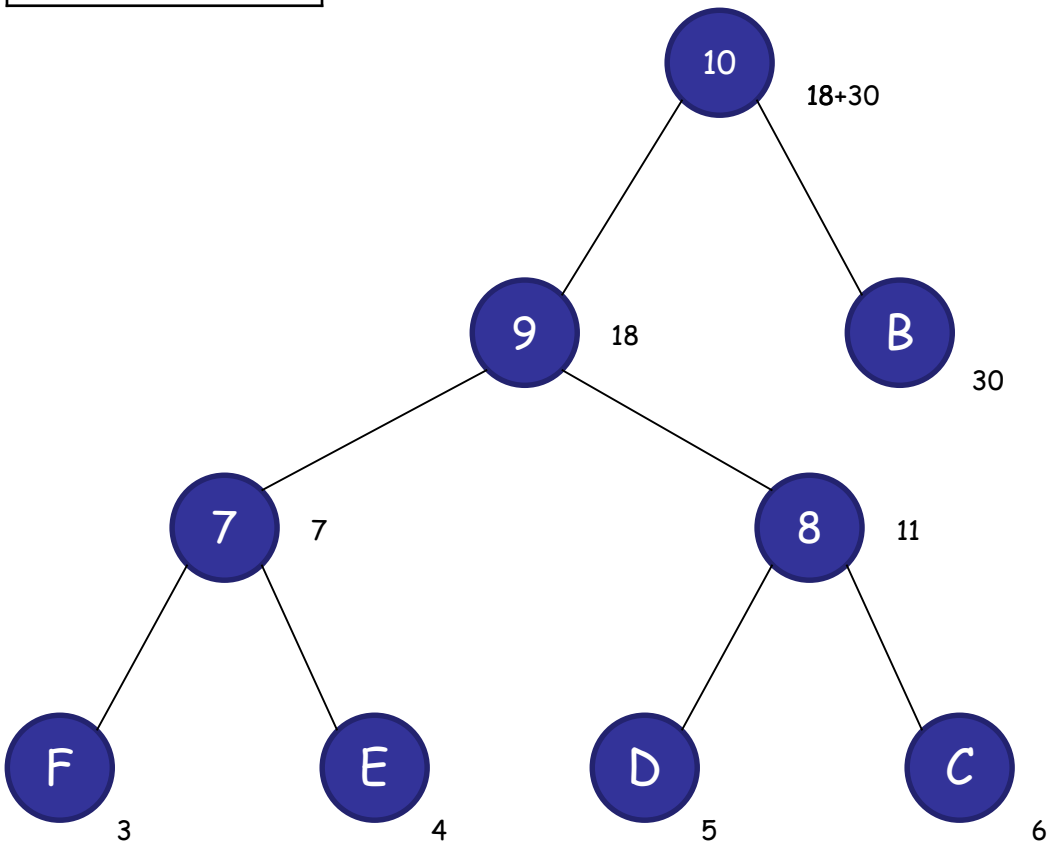
A=40
B=30
{8}=11
{7}=7

```
Algorithm Huffman-coding(f: array of integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
   { i= deletemin(H); j=deletemin(H);
     left(next) = i; right(next) = j;
     f(next) = f(i) + f(j);
     insert(H, next);
     next++;
   }
}
```



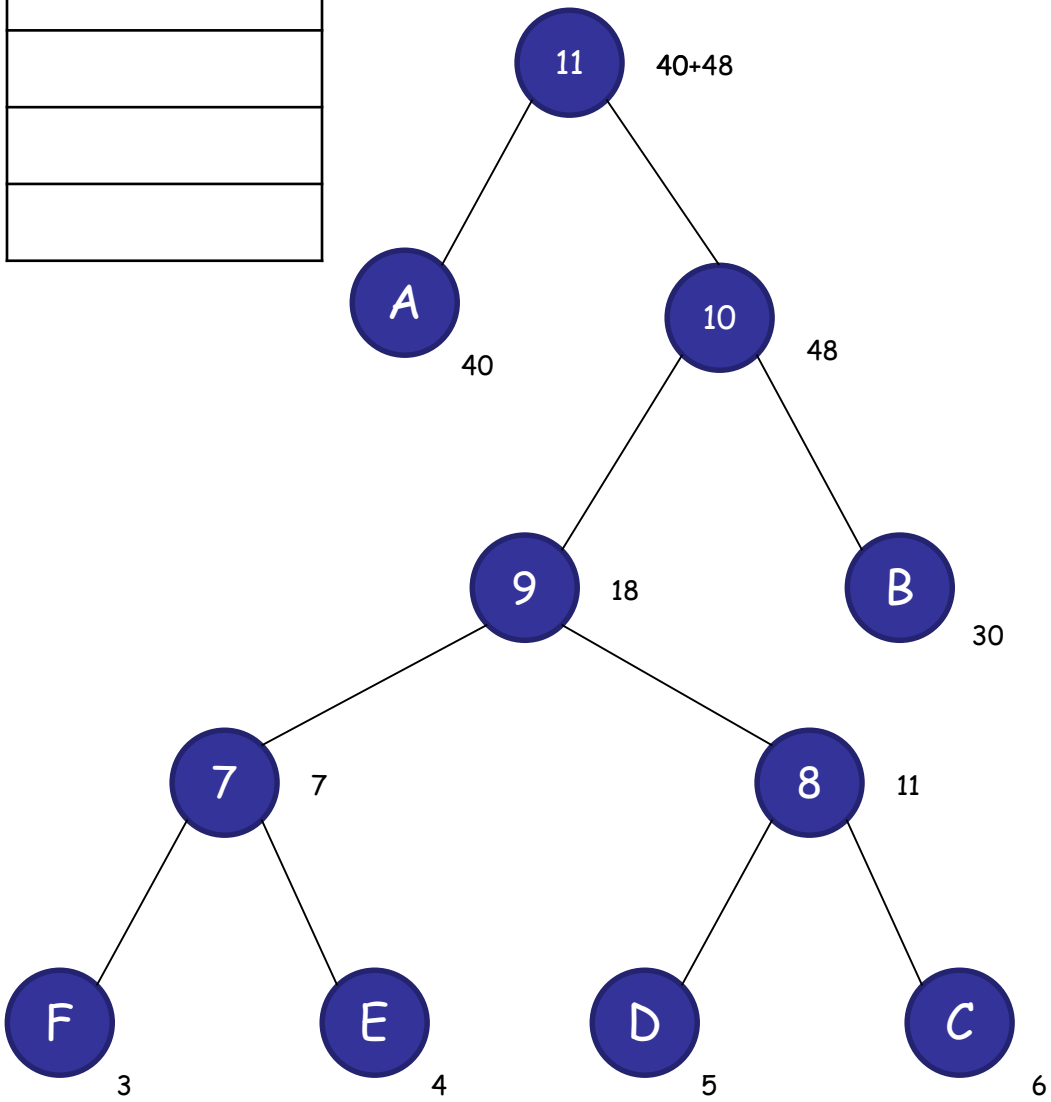
A=40
B=30
{9}=18

```
Algorithm Huffman-coding(f: array of integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
   { i= deletemin(H); j=deletemin(H);
     left(next) = i; right(next) = j;
     f(next) = f(i) + f(j);
     insert(H, next);
     next++;
   }
}
```

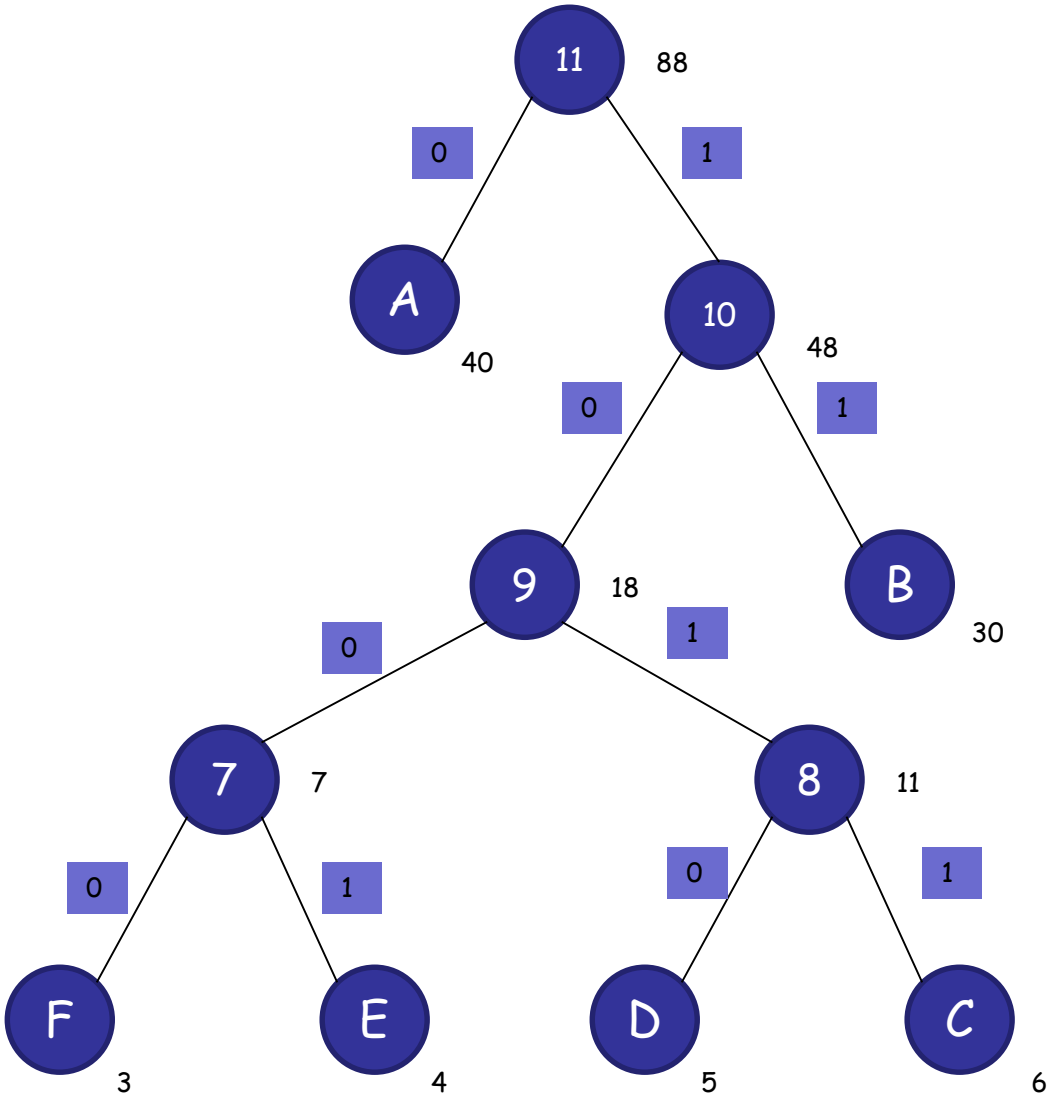


{10}=48
A=40

```
Algorithm Huffman-coding(f: array of integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
   { i= deletemin(H); j=deletemin(H);
     left(next) = i; right(next) = j;
     f(next) = f(i) + f(j);
     insert(H, next);
     next++;
   }
}
```



Με '0' στα αριστερά και '1' στα δεξιά θα έχουμε

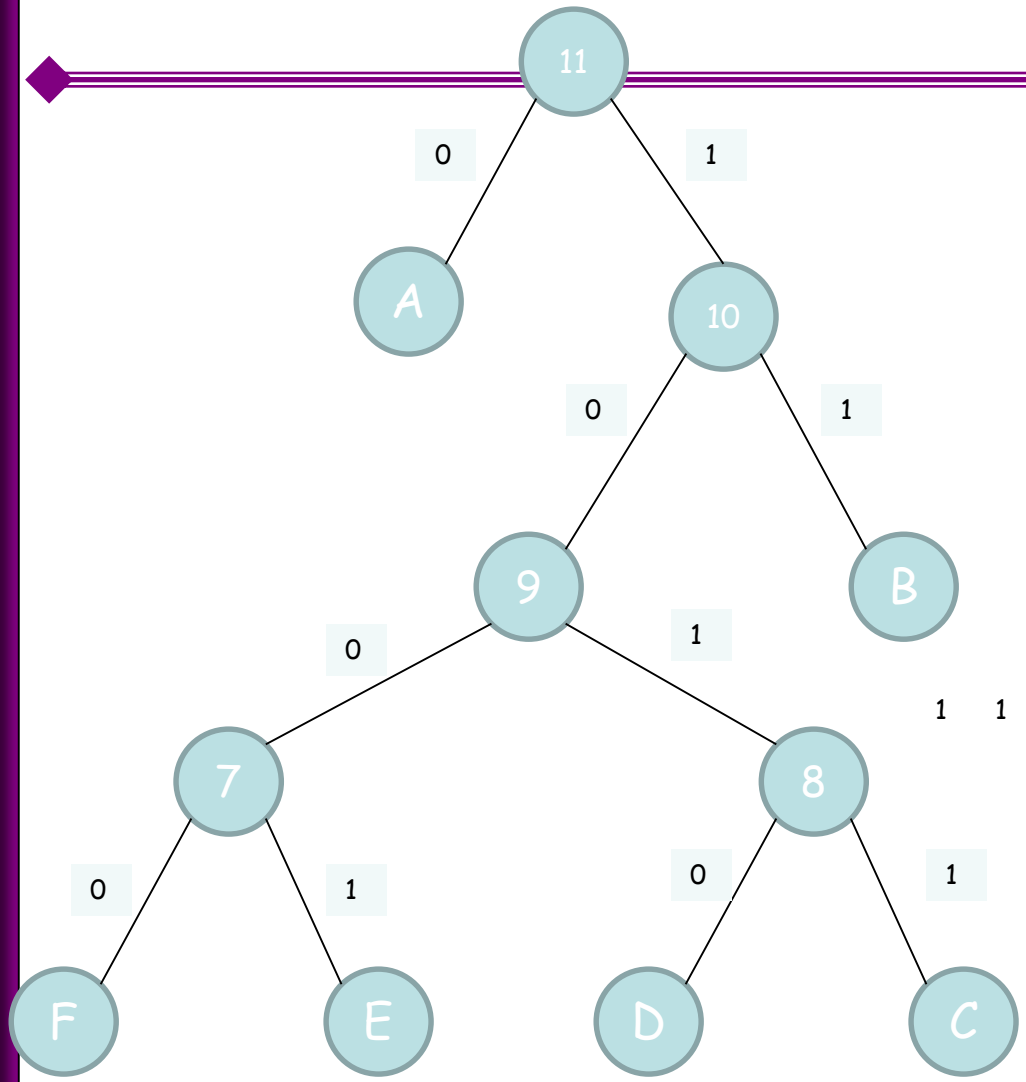



```
Algorithm Huffman-coding(f: array of
integer)
{ H: heap με κλειδί το f;
  for i = 1 to n do insert (H, i);
  next = n + 1;
  while |H| >= 2 do
   { i= deletemin(H); j=deletemin(H);
     left(next) = i; right(next) = j;
     f(next) = f(i) + f(j);
     insert(H, next);
     next++;
   }
}
```

Άρα

- A=0
- B=11
- C=1011
- D=1010
- E=1001
- F=1000

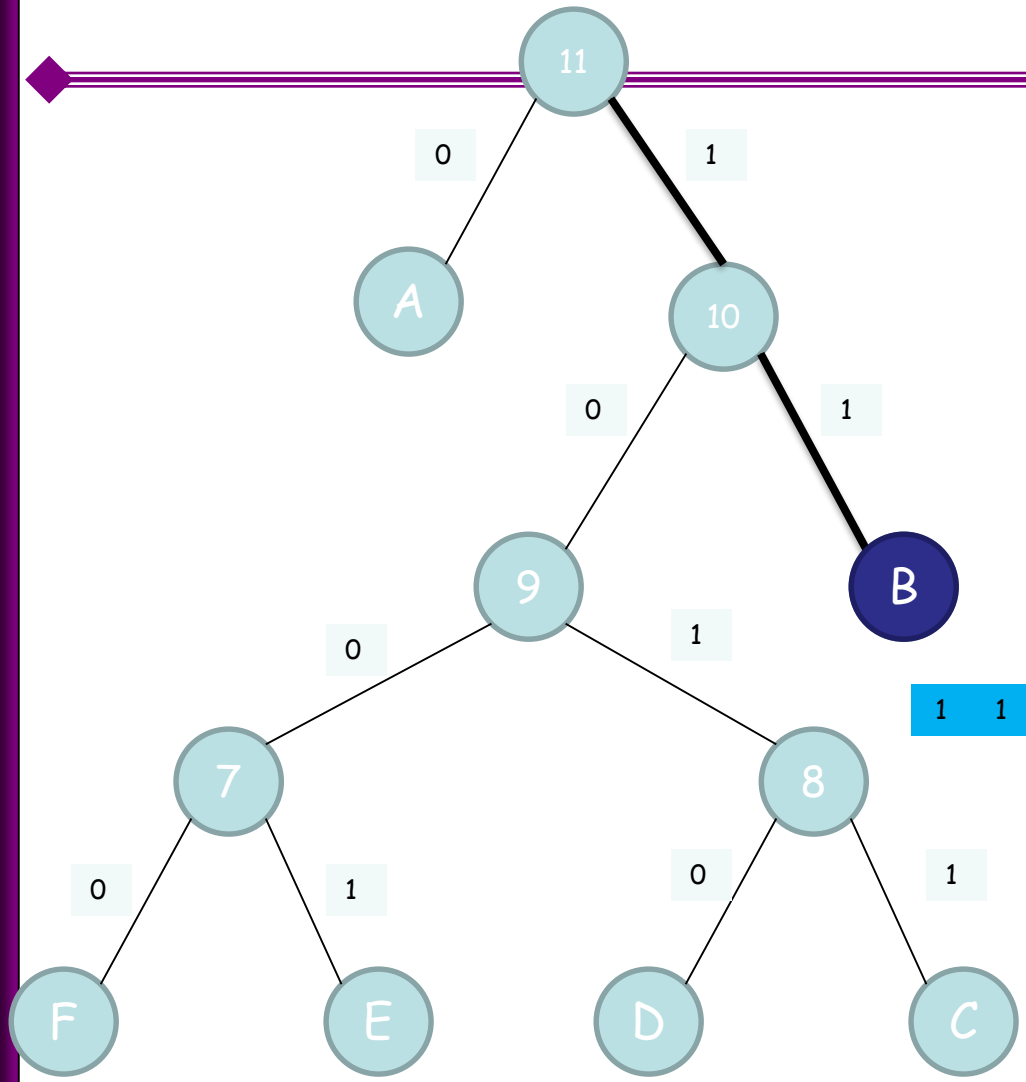
Αποκωδικοποίηση Huffman




```
Algorithm Huffman-decoder
{ i = 2n-1;
  while not eofdo
  { read(nextbit);
    if nextbit = 0 then i = left(i) else i =
    right(i);
    if left(i) = nil then
    { write(i); i=2n-1;}
  }
}
```

είσοδος προς αποκωδικοποίηση:

1 1 0 1 0 1 1 1 1 1 0 0 1 0 1 1

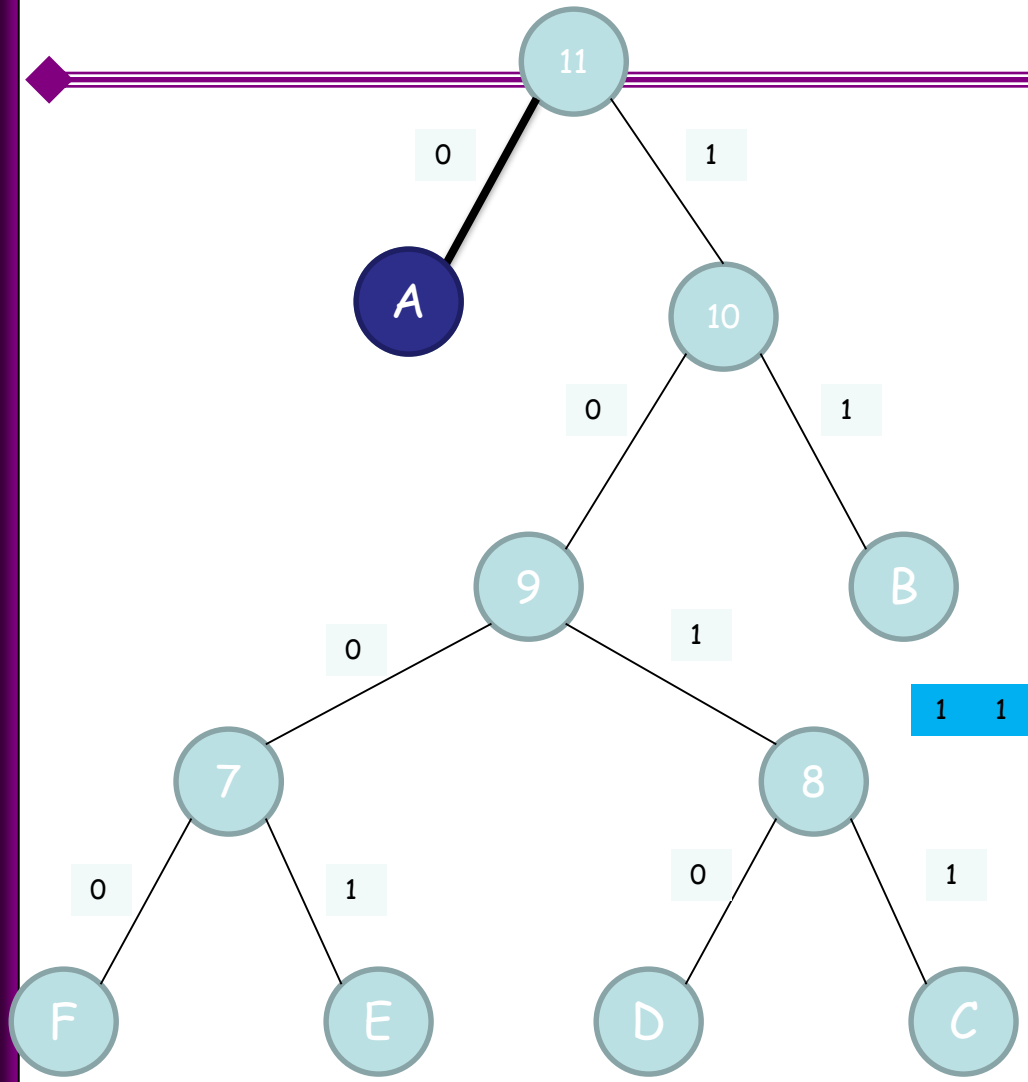



```
Algorithm Huffman-decoder
{ i = 2n-1;
  while not eofdo
  { read(nextbit);
    if nextbit = 0 then i = left(i) else i =
    right(i);
    if left(i) = nil then
    { write(i); i=2n-1;}
  }
}
```

είσοδος προς αποκωδικοποίηση:

1	1	0	1	0	1	1	1	1	1	0	0	1	0	1	1

B

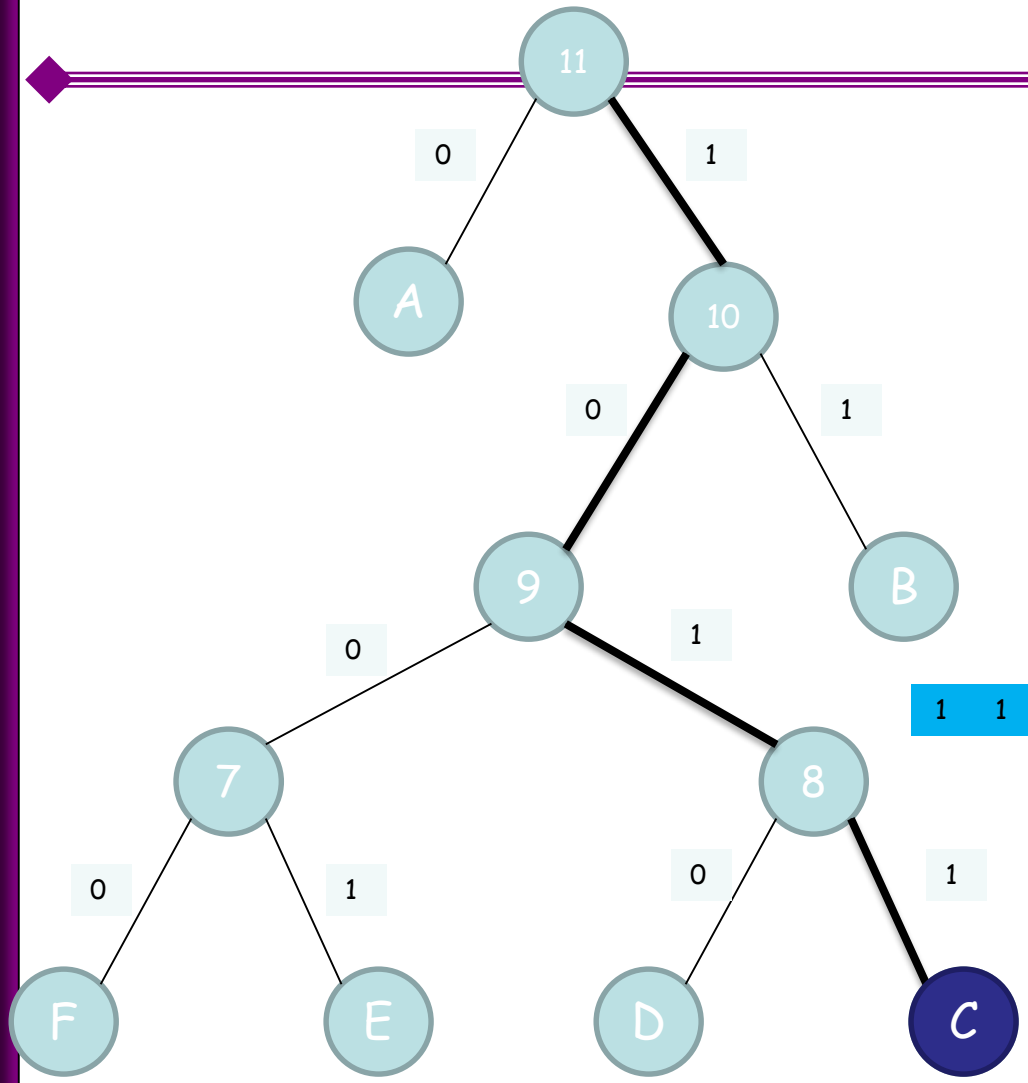



```
Algorithm Huffman-decoder
{ i = 2n-1;
  while not eofdo
  { read(nextbit);
    if nextbit = 0 then i = left(i) else i =
    right(i);
    if left(i) = nil then
    { write(i); i=2n-1;}
  }
}
```

είσοδος προς αποκωδικοποίηση:

1	1	0	1	0	1	1	1	1	1	0	0	1	0	1	1

B A

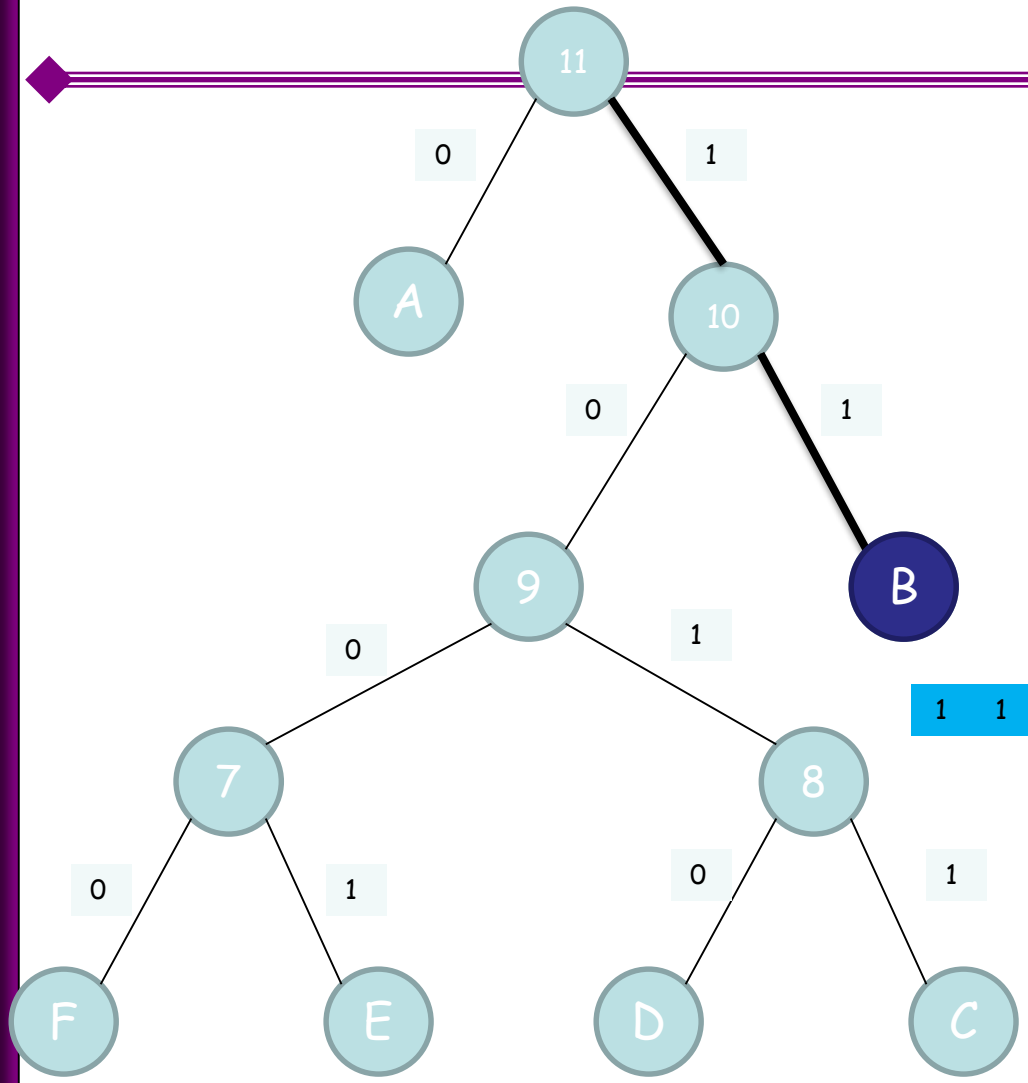



```
Algorithm Huffman-decoder
{ i = 2n-1;
  while not eofdo
  { read(nextbit);
    if nextbit = 0 then i = left(i) else i =
    right(i);
    if left(i) = nil then
    { write(i); i=2n-1;}
  }
}
```

είσοδος προς αποκωδικοποίηση:

1	1	0	1	0	1	1	1	1	1	0	0	1	0	1	1

B A C

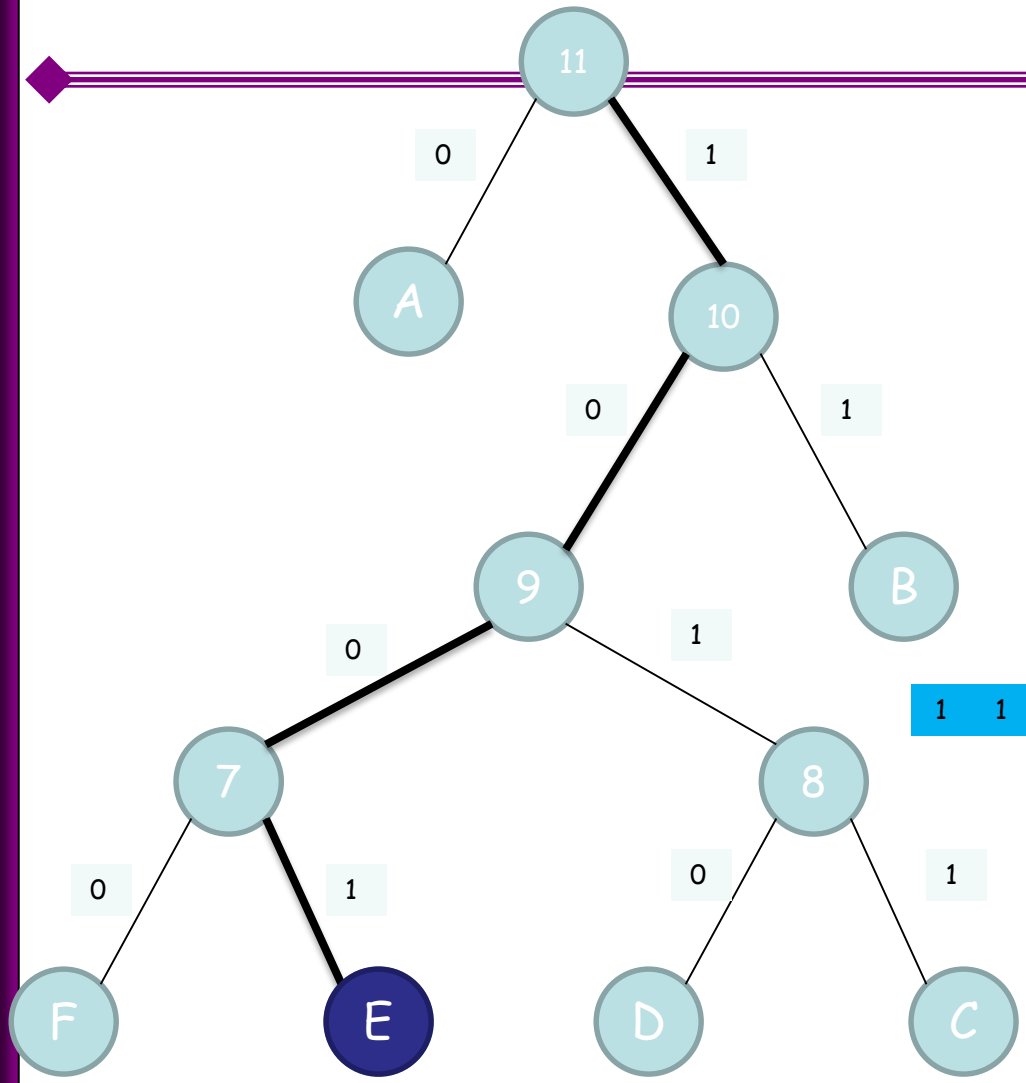



```
Algorithm Huffman-decoder
{ i = 2n-1;
  while not eofdo
  { read(nextbit);
    if nextbit = 0 then i = left(i) else i =
    right(i);
    if left(i) = nil then
    { write(i); i=2n-1;}
  }
}
```

είσοδος προς αποκωδικοποίηση:

1	1	0	1	0	1	1	1	1	1	0	0	1	0	1	1

B A C B

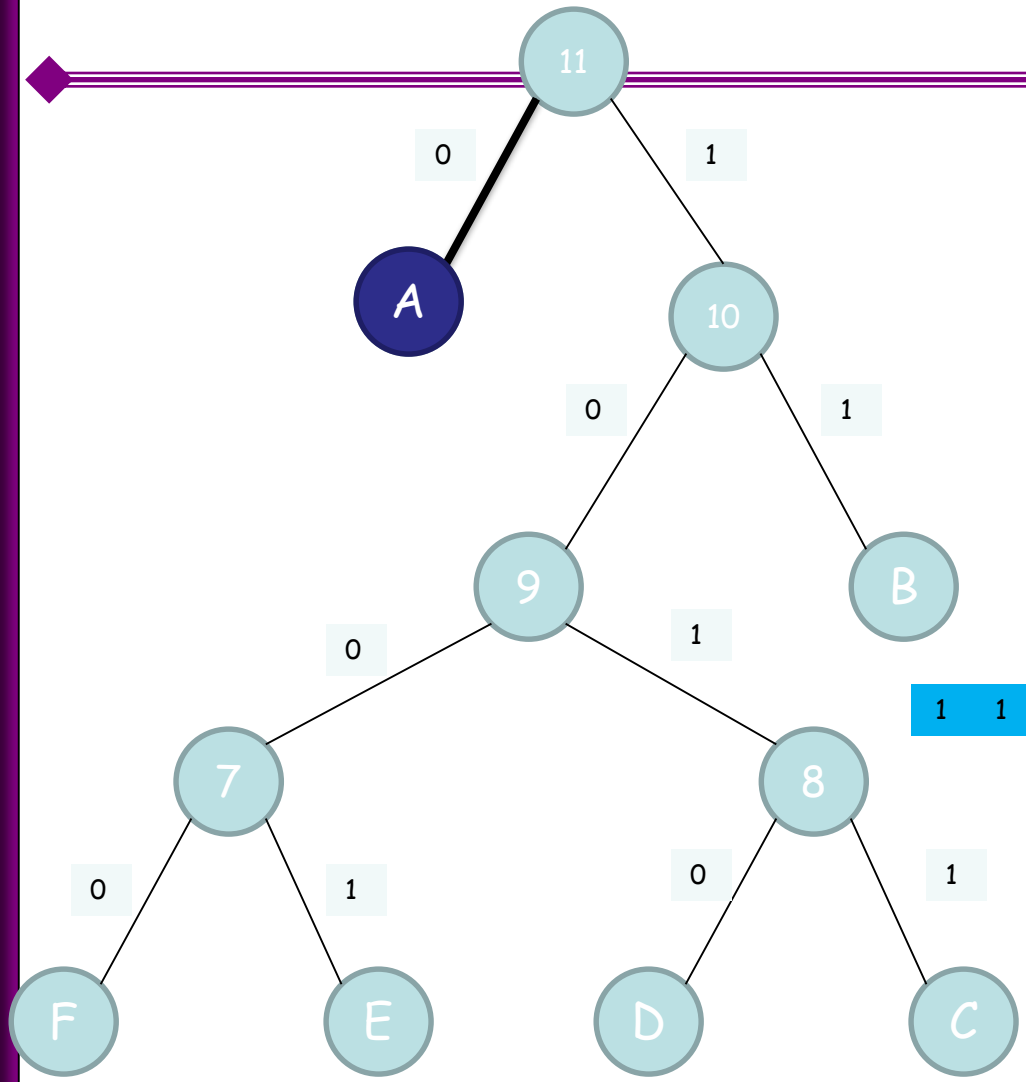



```
Algorithm Huffman-decoder
{ i = 2n-1;
  while not eofdo
  { read(nextbit);
    if nextbit = 0 then i = left(i) else i =
    right(i);
    if left(i) = nil then
    { write(i); i=2n-1;}
  }
}
```

είσοδος προς αποκωδικοποίηση:

1	1	0	1	0	1	1	1	1	1	0	0	1	0	1	1

B A C B E

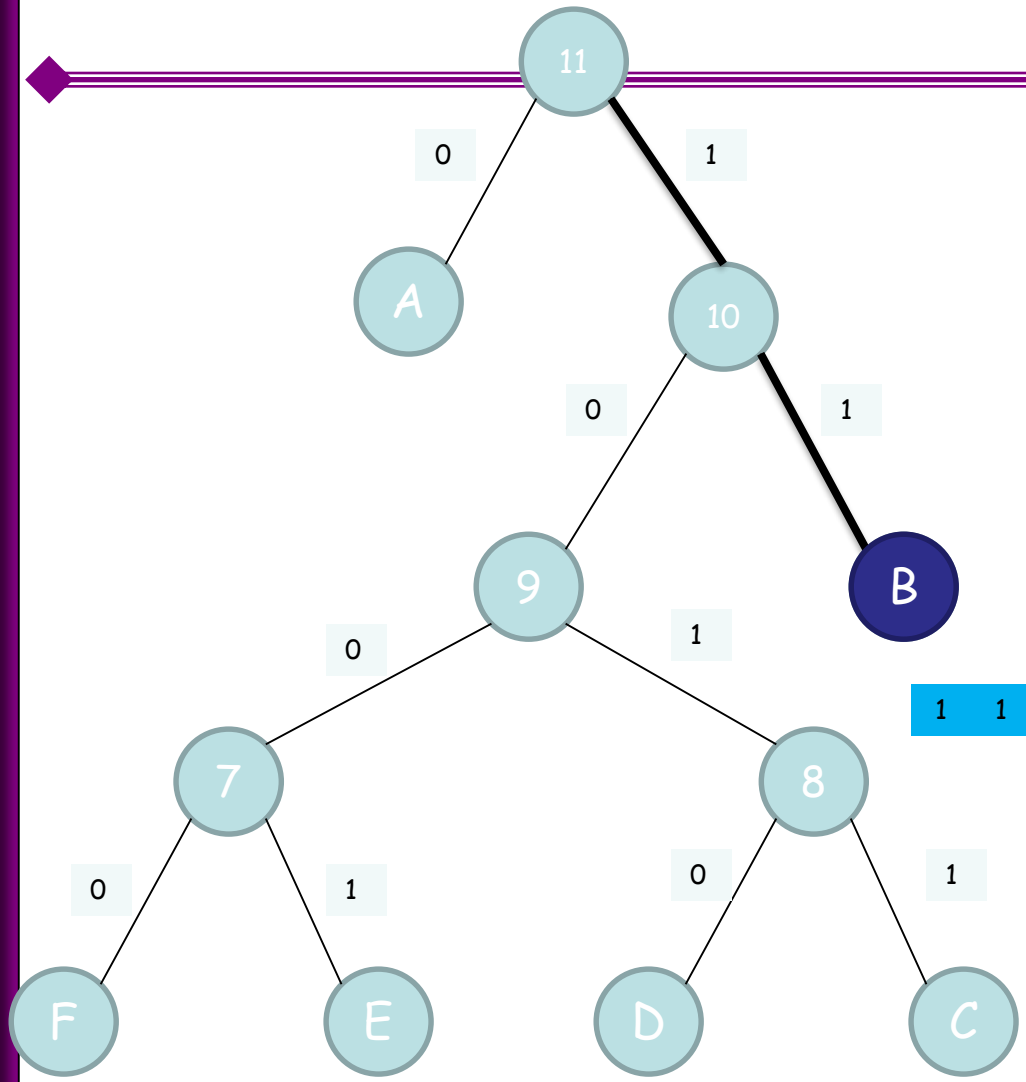



```
Algorithm Huffman-decoder
{ i = 2n-1;
  while not eofdo
  { read(nextbit);
    if nextbit = 0 then i = left(i) else i =
    right(i);
    if left(i) = nil then
    { write(i); i=2n-1;}
  }
}
```

είσοδος προς αποκωδικοποίηση:

1	1	0	1	0	1	1	1	1	1	0	0	1	0	1	1

B A C B E A




```
Algorithm Huffman-decoder
{ i = 2n-1;
  while not eofdo
  { read(nextbit);
    if nextbit = 0 then i = left(i) else i =
    right(i);
    if left(i) = nil then
    { write(i); i=2n-1;}
  }
}
```

είσοδος προς αποκωδικοποίηση:

1	1	0	1	0	1	1	1	1	1	0	0	1	0	1	1

B A C B E A B

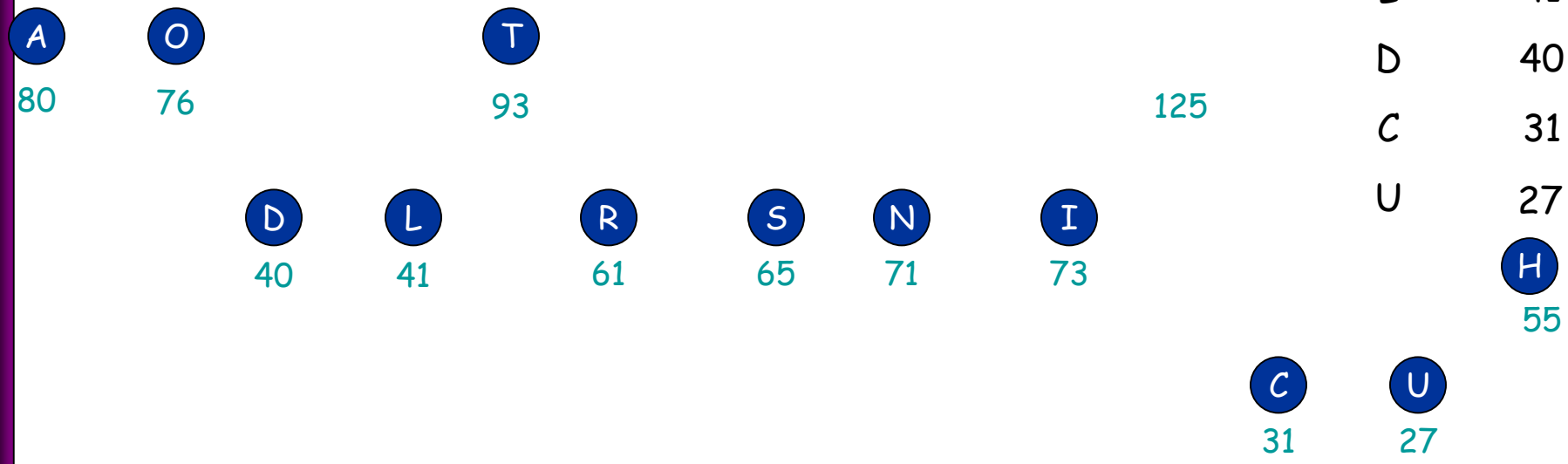
Κωδικοποίηση Huffman

- Σε ένα κείμενο, δίνεται η συχνότητα
- Εμφάνισης των χαρακτήρων

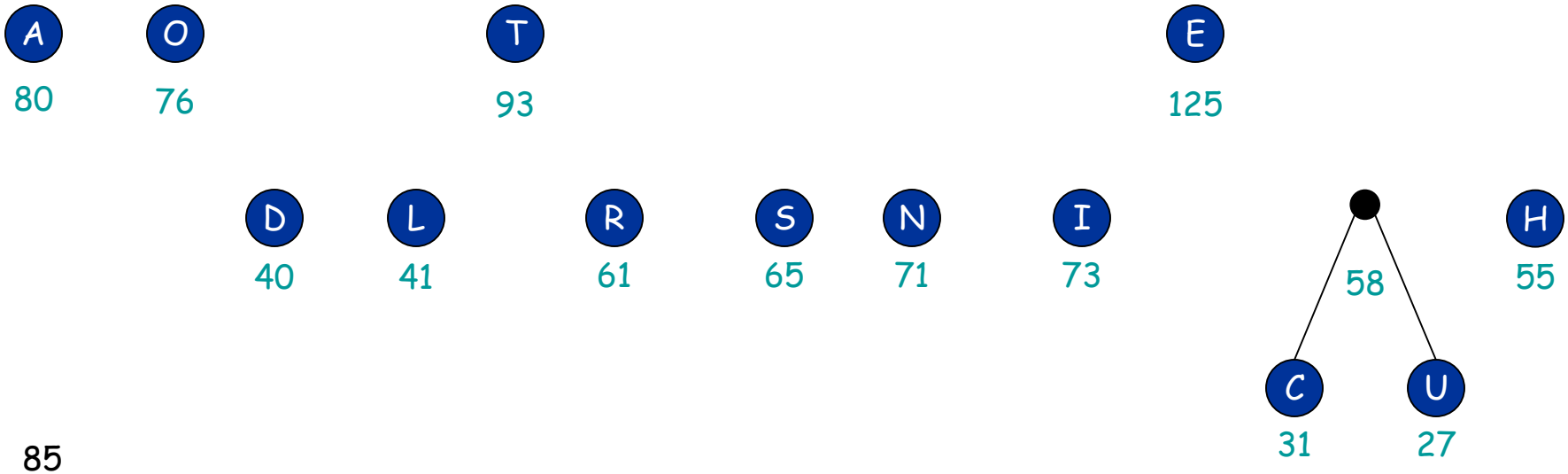
Char	Freq
E	125
T	93
A	80
O	76
I	72
N	71
S	65
R	61
H	55
L	41
D	40
C	31
U	27

Κωδικοποίηση Huffman

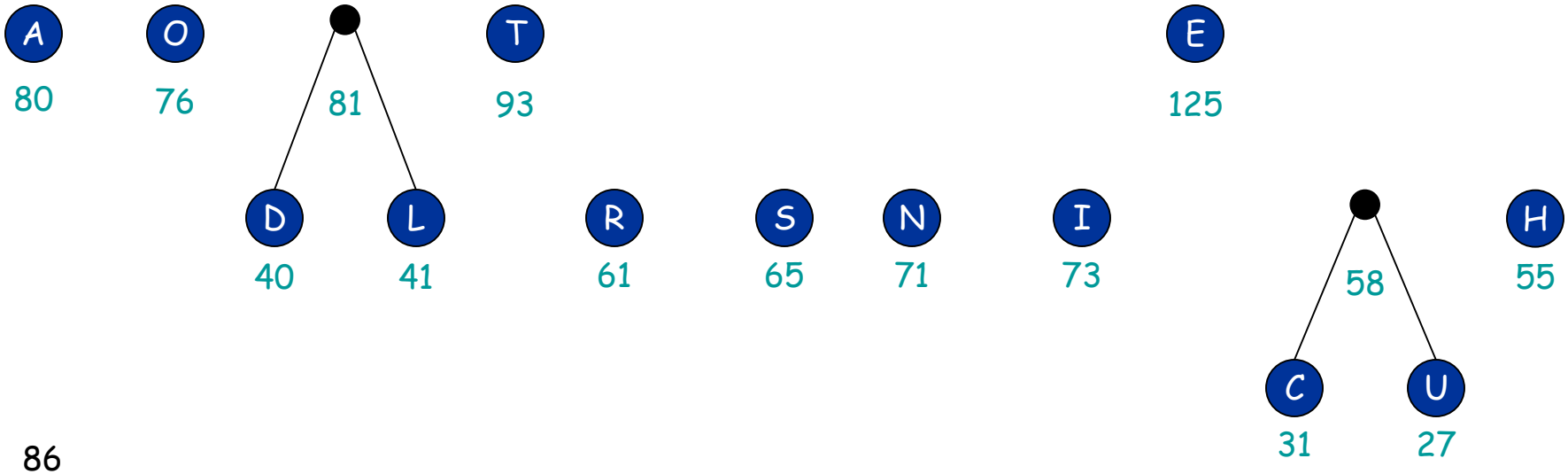
Char	Freq
E	125
T	93
A	80
O	76
I	72
N	71
S	65
R	61
H	55
L	41
D	40
C	31
U	27



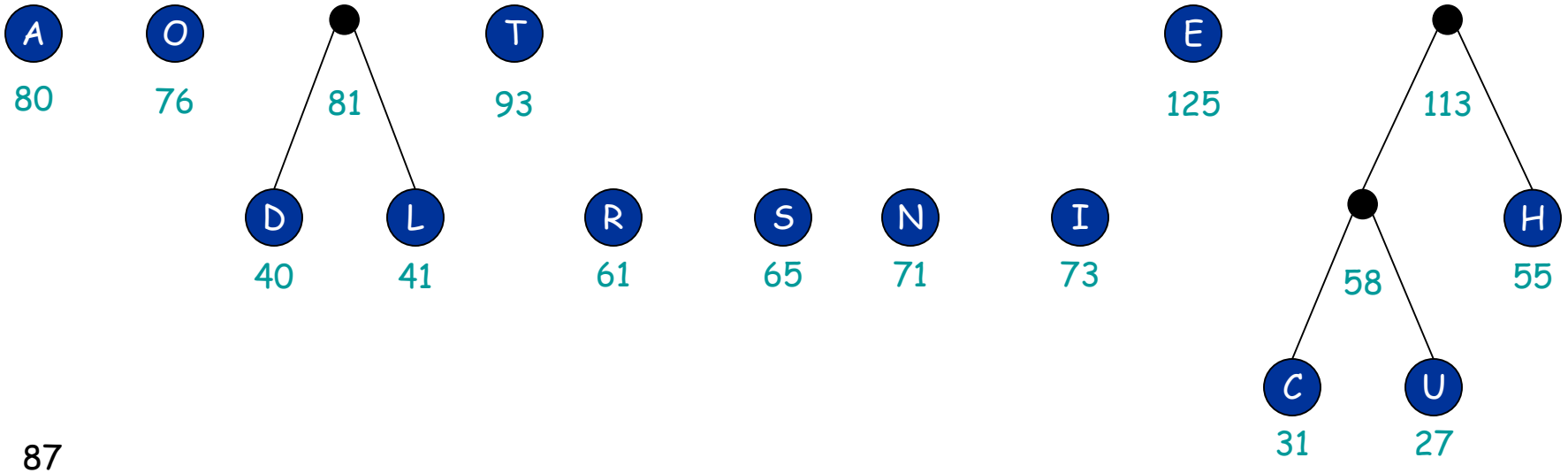
Κωδικοποίηση Huffman



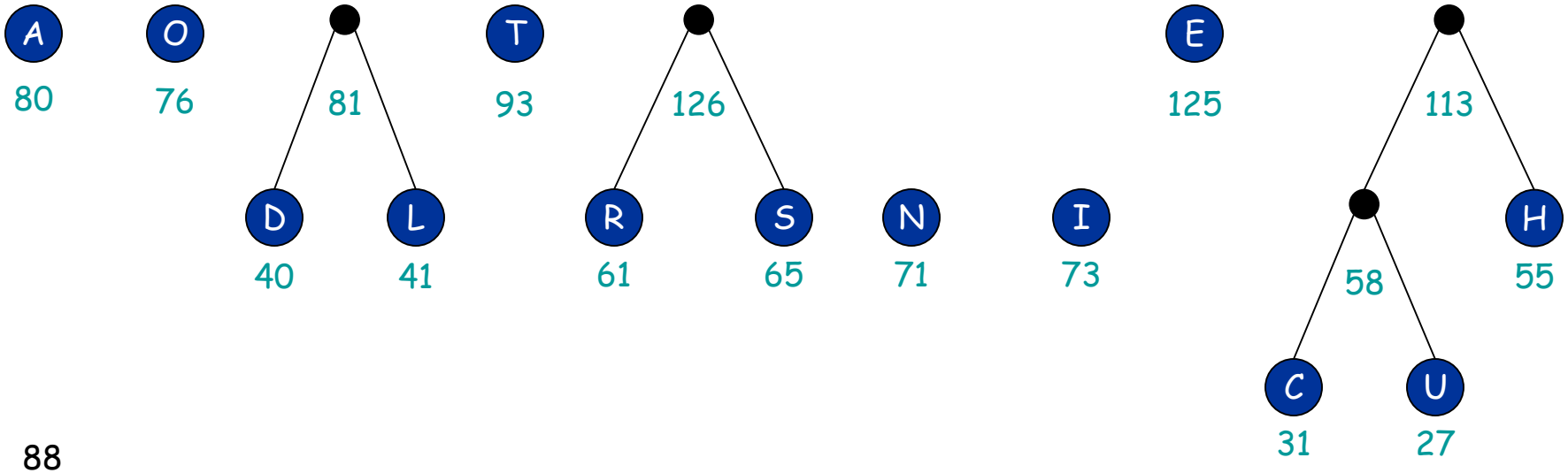
Κωδικοποίηση Huffman



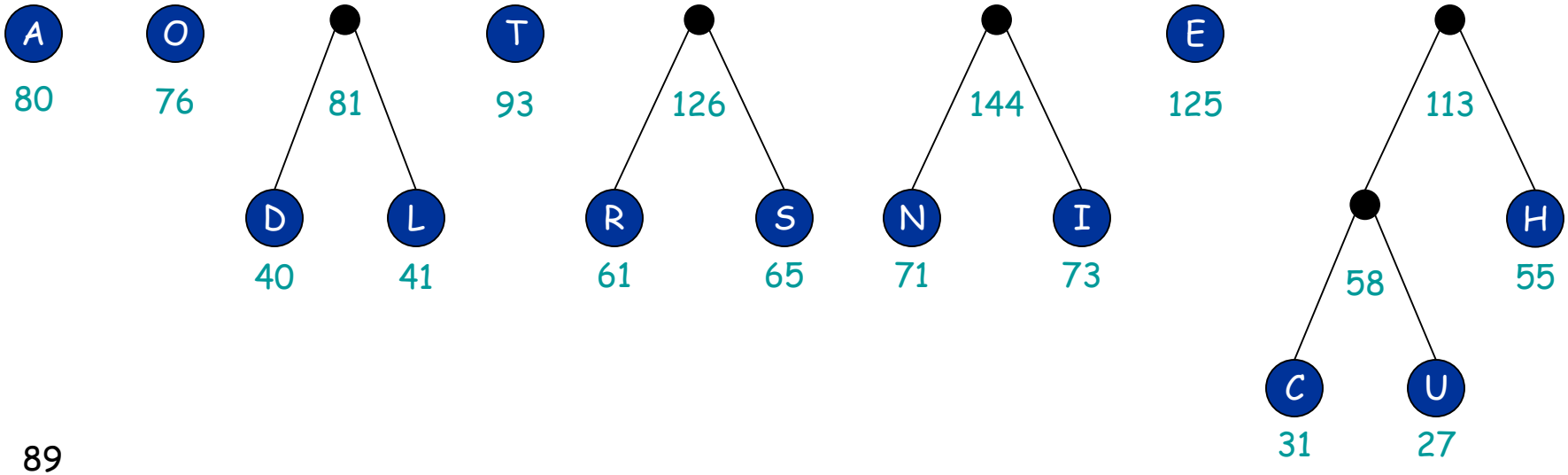
Κωδικοποίηση Huffman



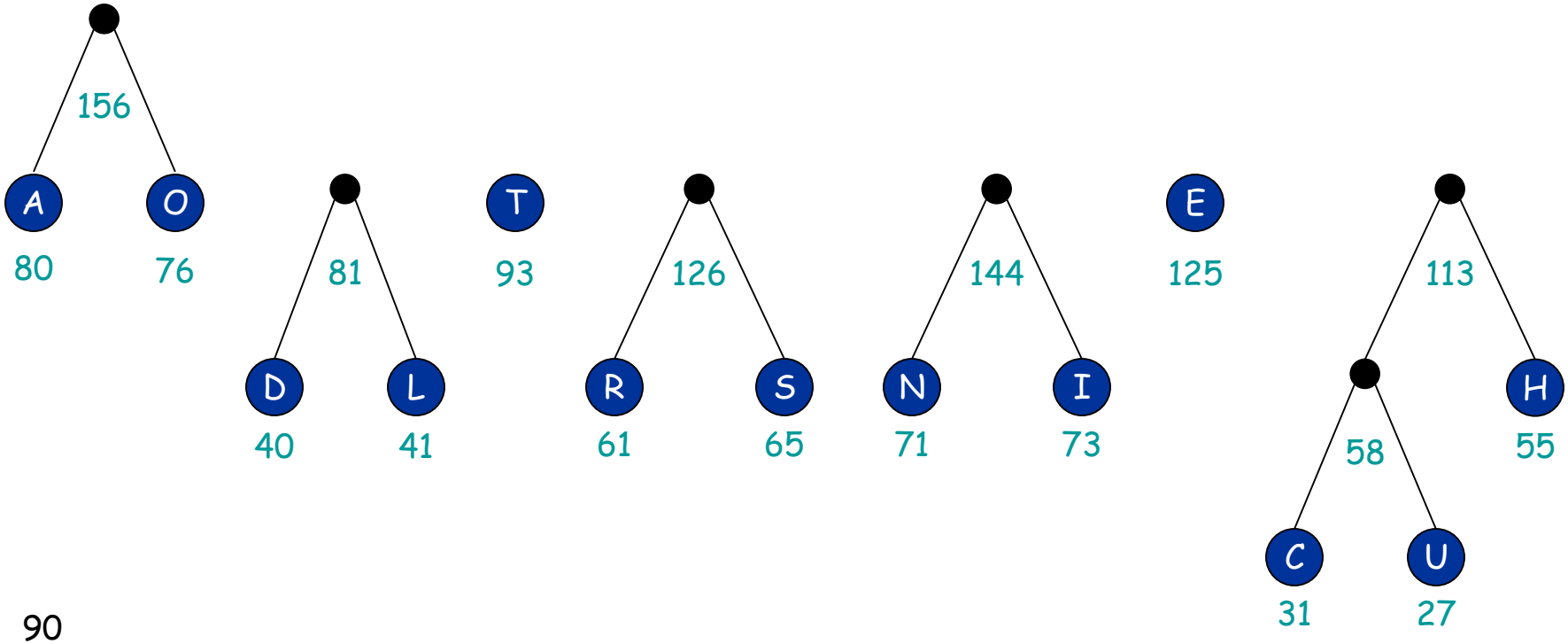
Κωδικοποίηση Huffman



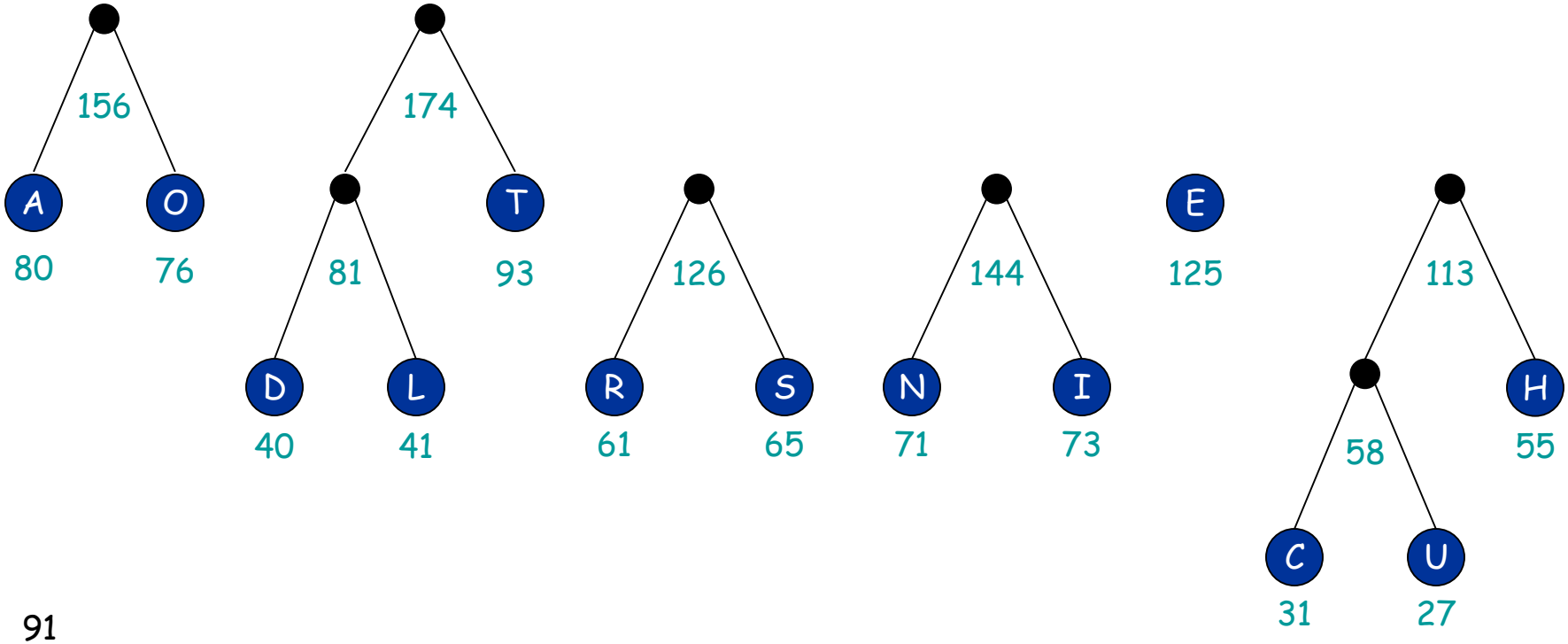
Κωδικοποίηση Huffman



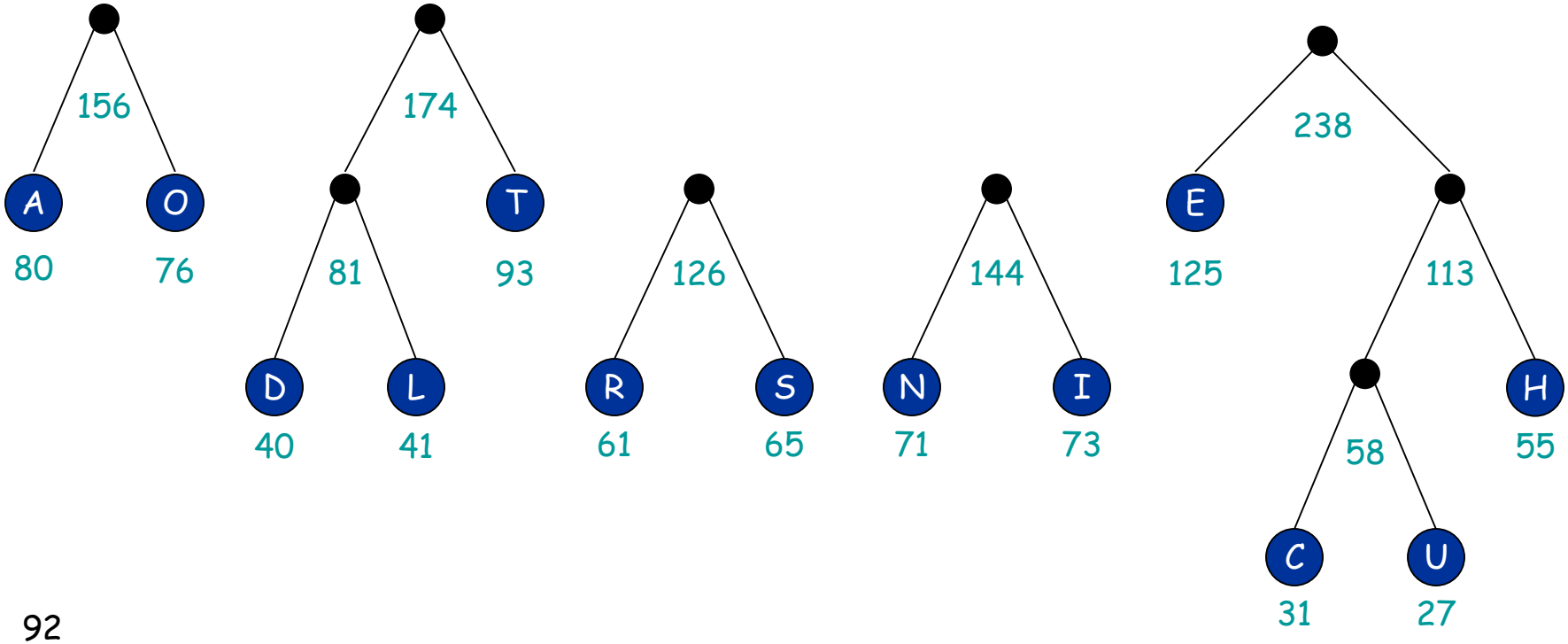
Κωδικοποίηση Huffman



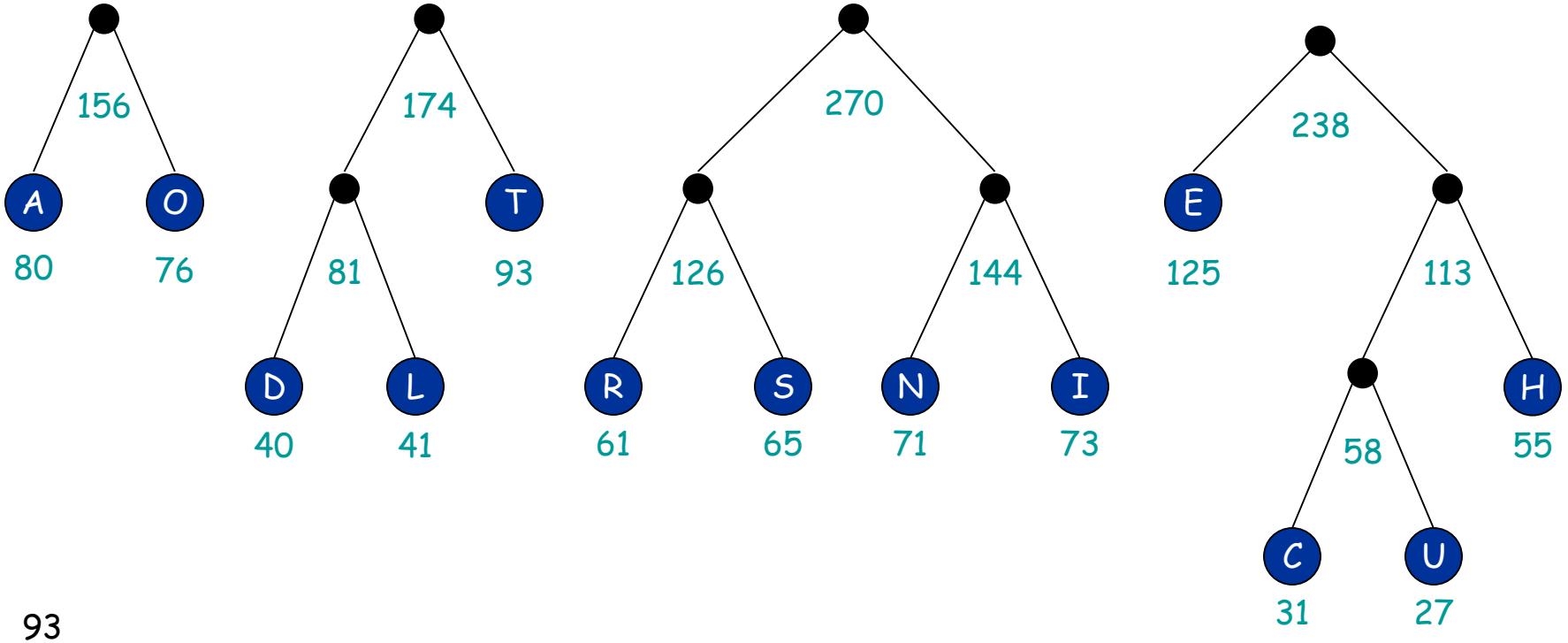
Κωδικοποίηση Huffman



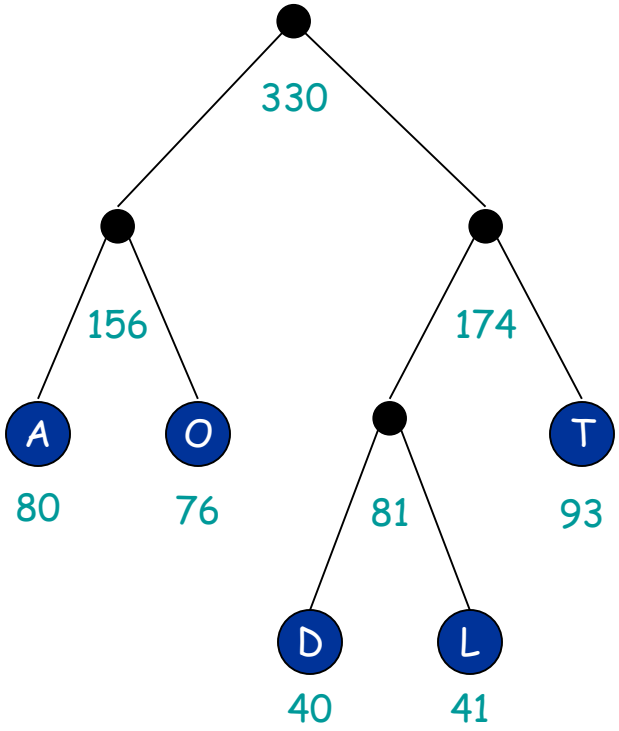
Κωδικοποίηση Huffman

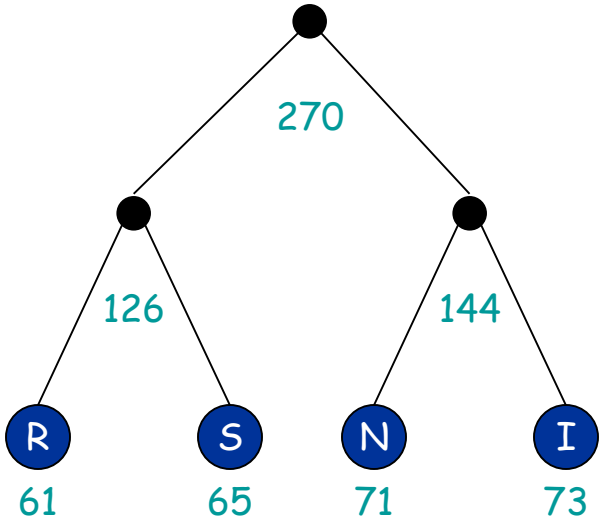


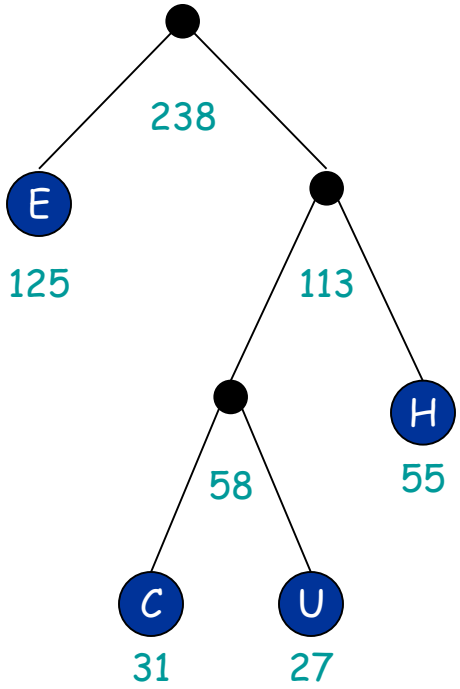
Κωδικοποίηση Huffman



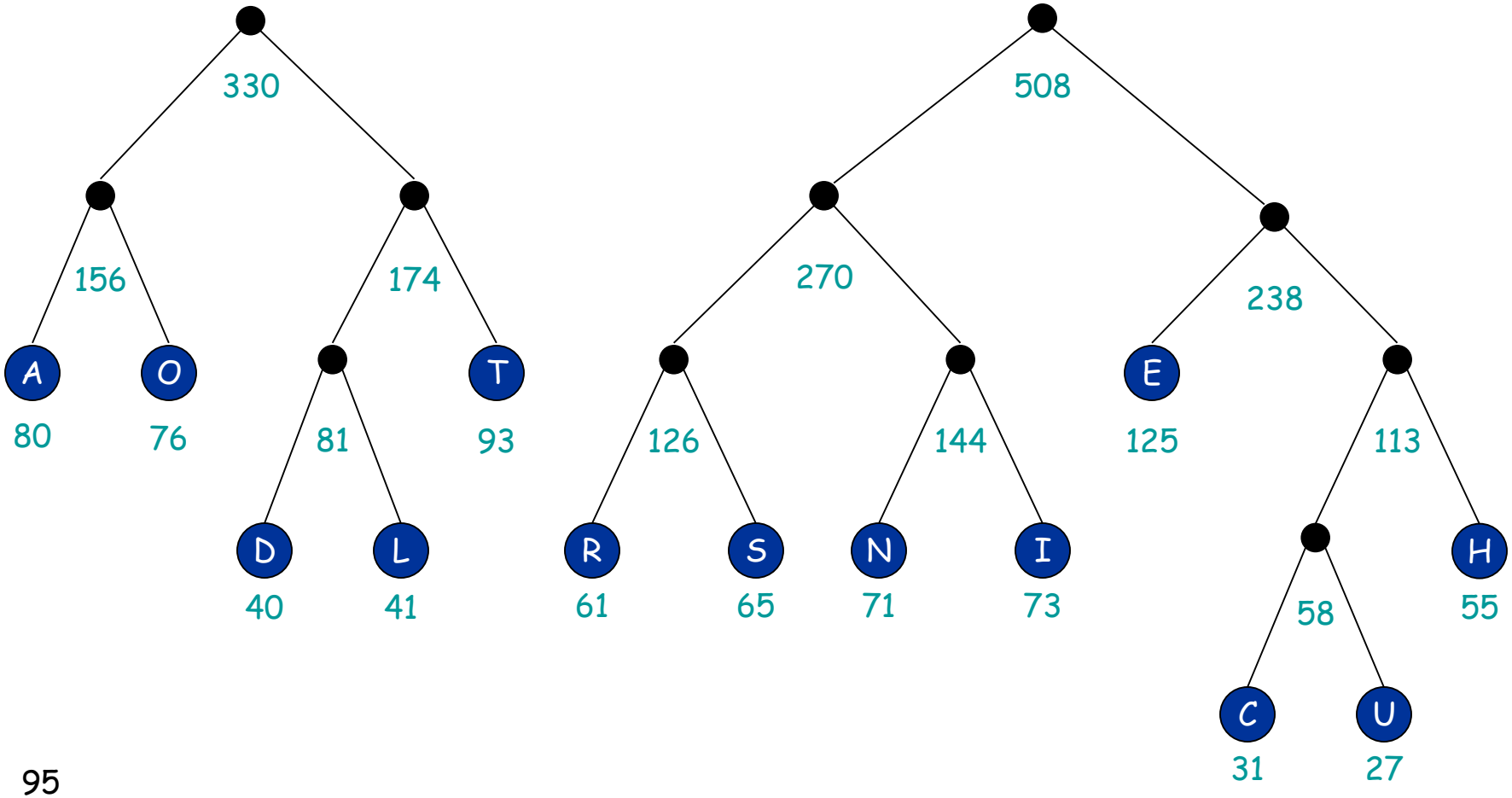
Κωδικοποίηση Huffman



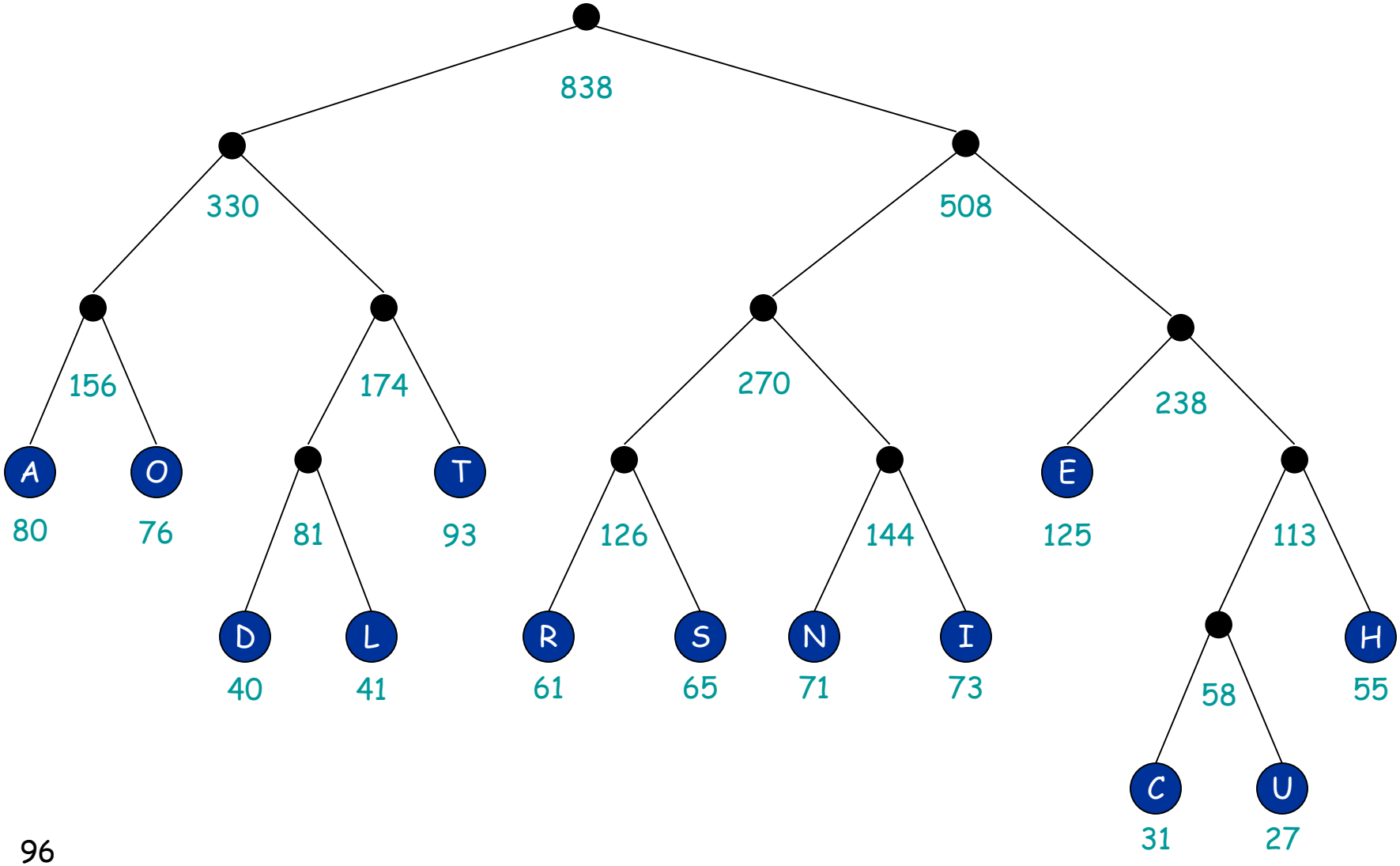




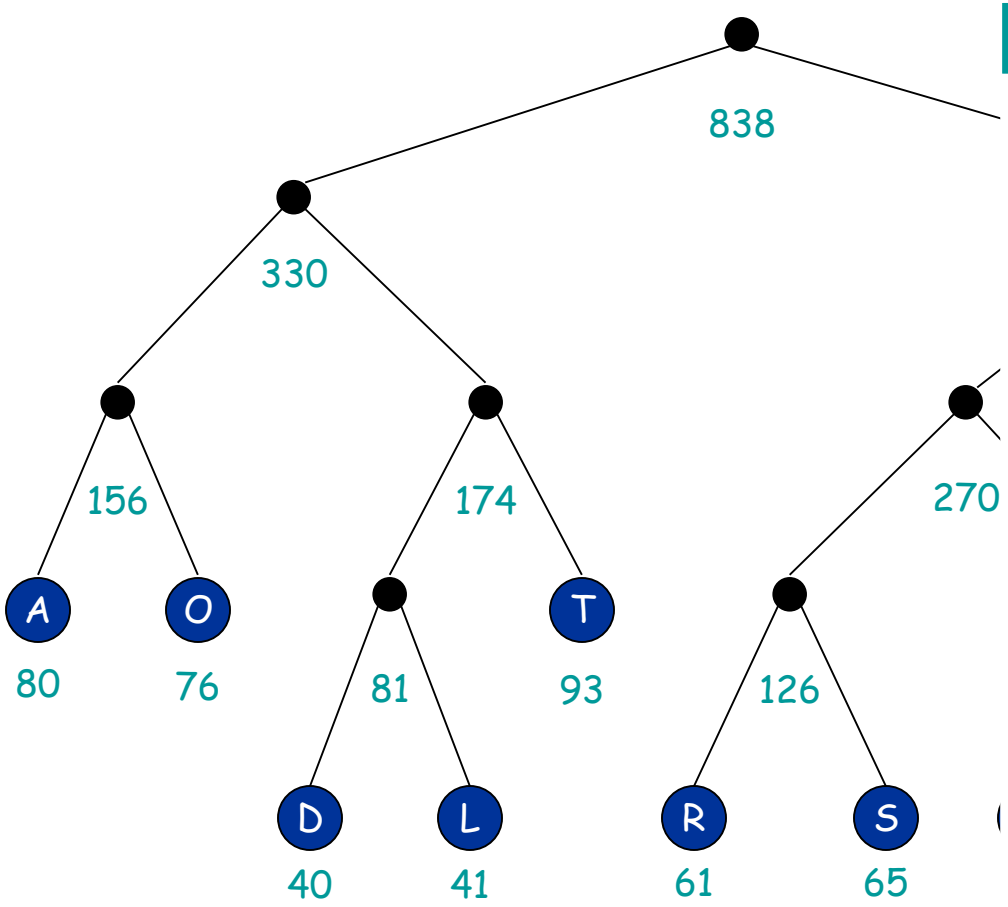
Κωδικοποίηση Huffman



Κωδικοποίηση Huffman



Κωδικοποίηση Huffman



Char	Freq	Fixed	Huff
E	125	0000	110
T	93	0001	011
A	80	0010	000
O	76	0011	001
I	73	0100	1011
N	71	0101	1010
S	65	0110	1001
R	61	0111	1000
H	55	1000	1111
L	41	1001	0101
D	40	1010	0100
C	31	1011	11100
U	27	1100	11101
Total	838	4.00	3.62