ADATSZERKEZETEK ÉS ALGORITMUSOK

Python nyelven

Rendezések.

A rendező algoritmusok bonyolultsága. Radix rendezés. Kupacrendezés (HeapSort). Kupacrendezés bonyolultsága. Útvonalak hosszúsága, a Huffman algoritmus.

Teszt

- Adja meg egy adott elem listába beszúrásának algoritmusát
- Építsen BST fát a következő elemekből

12, 35, 3, 4, 89, 12, 56, 20

Radix rendezés

- Legtermészetesebb rendezés, ha sok névből álló, vagy sokjegyű számokból álló listát kell rendezni.
- Pl. decimális számokra
- Először az utolsó számjegy szerint csoportokba szétválogatom, és a csoportokat újra összevonom
- Utána az utolsó előtti számjegy szerint szétválogatom stb.

A radix rendezés bonyolultsága

Ha A1, A2…An az n elemet tartalmazó lista.

d az alap, vagyis a válogatás során kialakítandó csoportok száma (d=10 decimális számoknál, d=26 betűknél stb.)

s a számjegyek száma, karakterek száma stb., ennyi menetben kell végezni a szétválogatást

- Összehasonlítások száma<=d*s*n
- d független n értékétől, de s függ tőle:
- log n<=s<=n →O(n*log(n))
- Tárigénye: d*n, ezt dinamikus helyfoglalással 2*n-ig lehet csökkenteni.

Kupac adatszerkezet definíciója

- Kupac
  - Olyan (bináris) fa, amely teljesíti a kupactulajdonságot.
- Kupactulajdonság
  - Ha B A gyereke, akkor a (maximális) kupactulajdonság: kulcs(A) ≥ kulcs(B)

Vagyis

Minden elem kisebb kell legyen mint a szülő

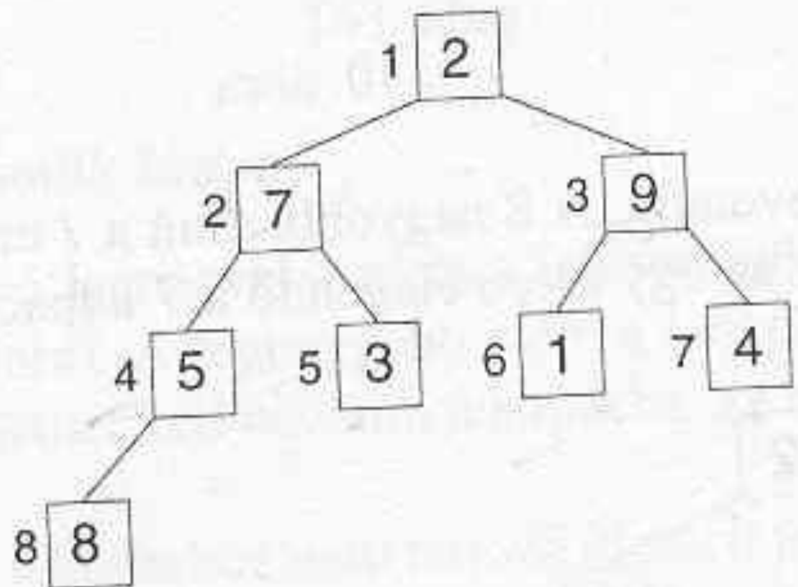
Rendezések - Kupac rendezés

- Kupac:
  - Olyan tömb, melynek elemeit egy bináris fa csomópontjaiként képzeljük,
  - K(1) a fa gyökere,
  - ahol a j indexű szülő gyerekei a 2*j és 2*j+1 indexű elemek (K(2*j) és K(2*j+1)
  - K([j/2])>=K(j) $\forall$ 1<=[j/2]<j<=n-re

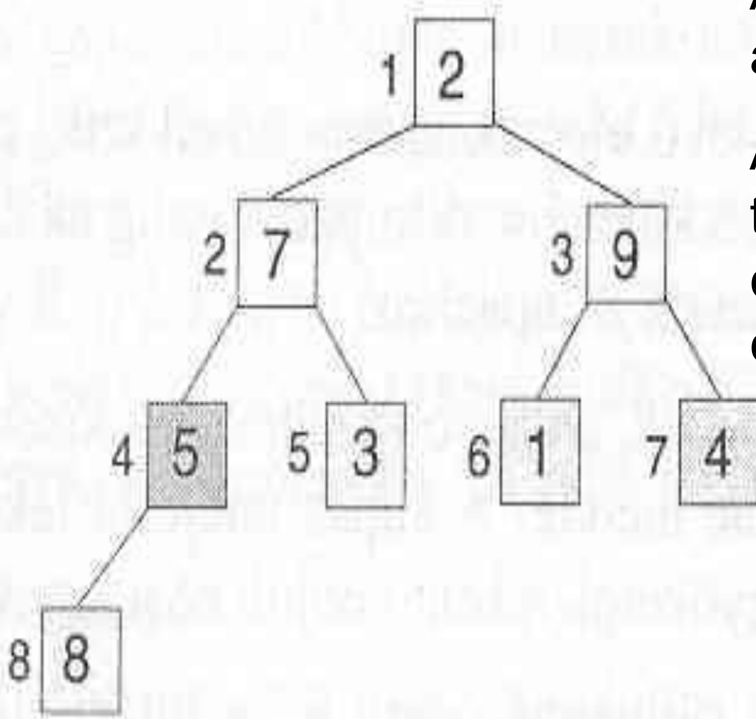
1. Az algoritmus első fázisa

j	1	2	3	4	5	6	7	8
	2	7	9	5	3	1	4	8

Kiindulás:



2. Az algoritmus első fázisa
Az elemeket egyesével bevonjuk a kupacba

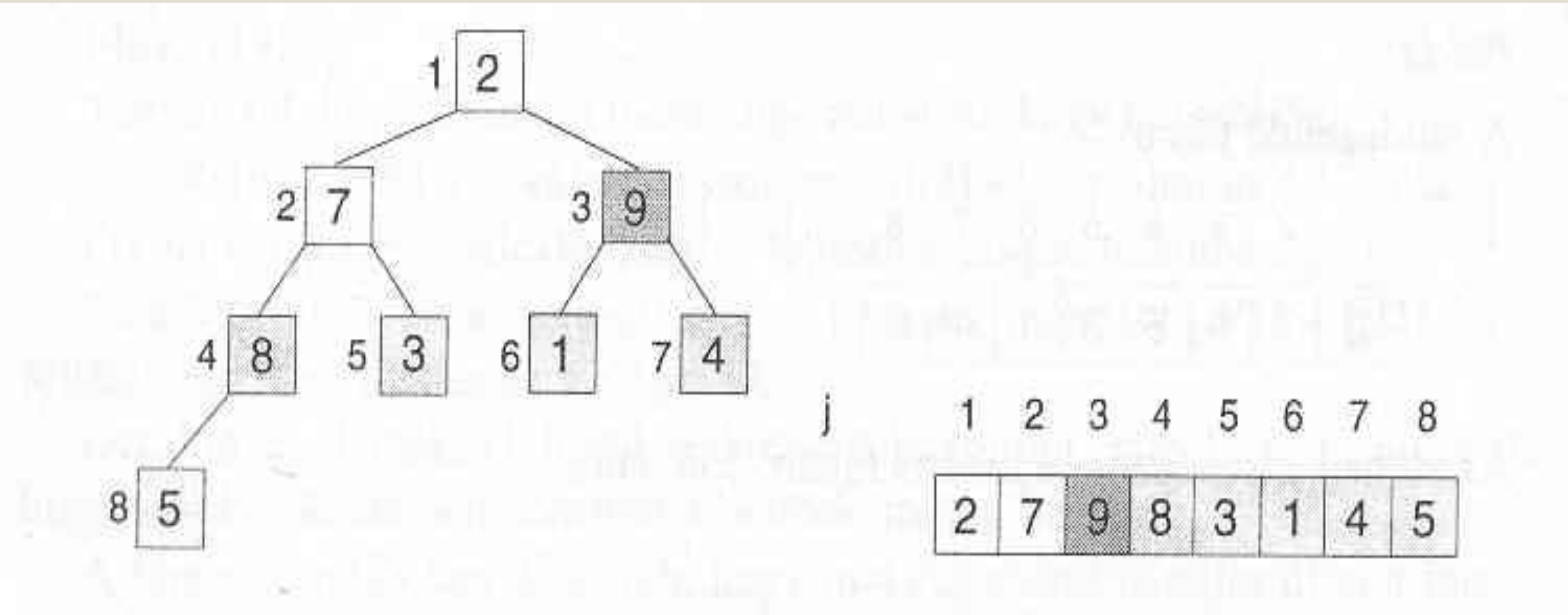


A levélelemekre automatikusan teljesül a kupac tulajdonság

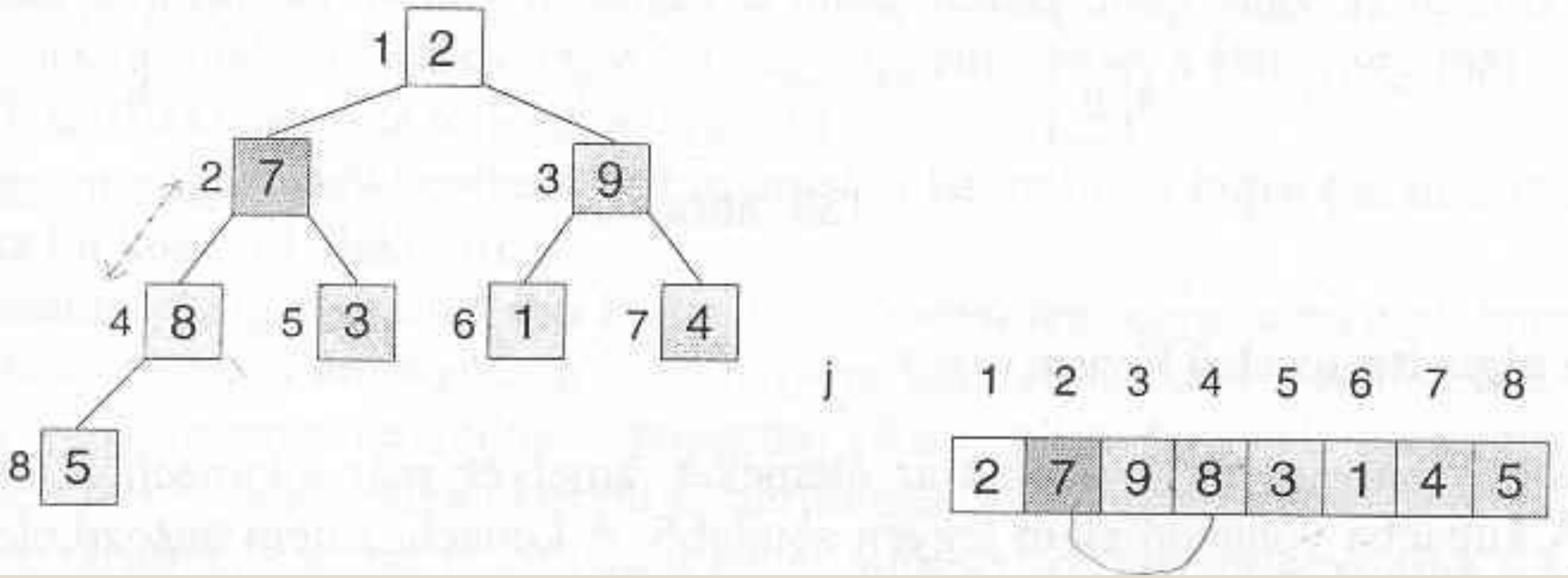
Az m=j/2 esetre kell biztosítani a kupac tulajdonság teljesülését az elemek cseréjével, úgy, hogy mindig csak az m csúcs gyökerű részfán dolgozunk.

j	1	2	3	4	5	6	7	8
	2	7	9	5	3	1	4	8

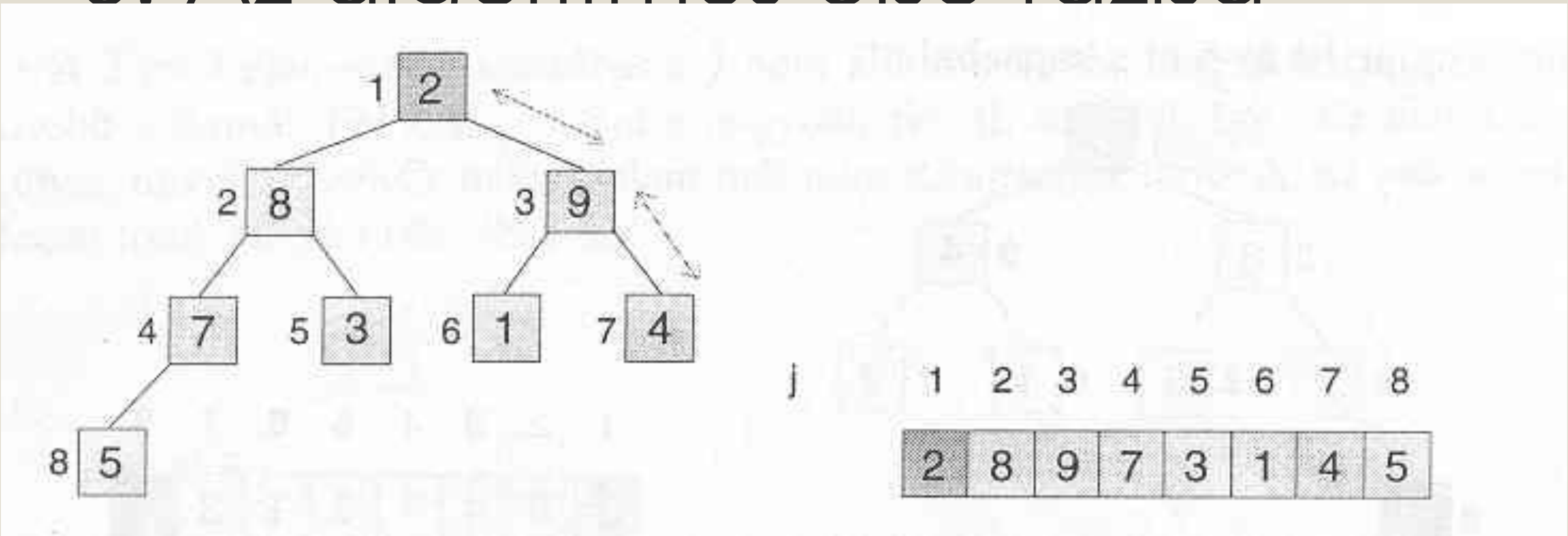
3. Az algoritmus első fázisa



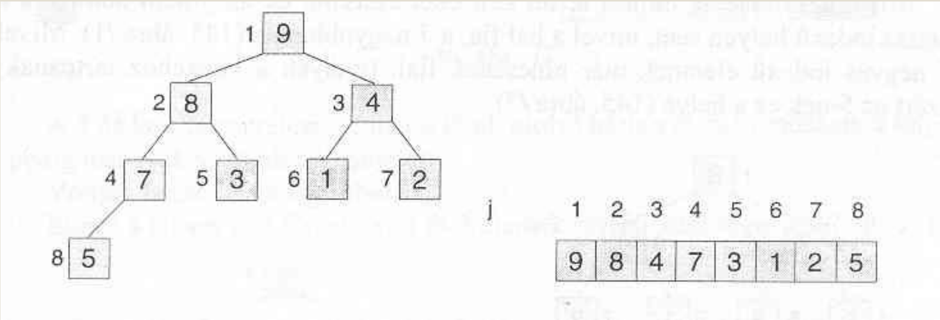
4. Az algoritmus első fázisa



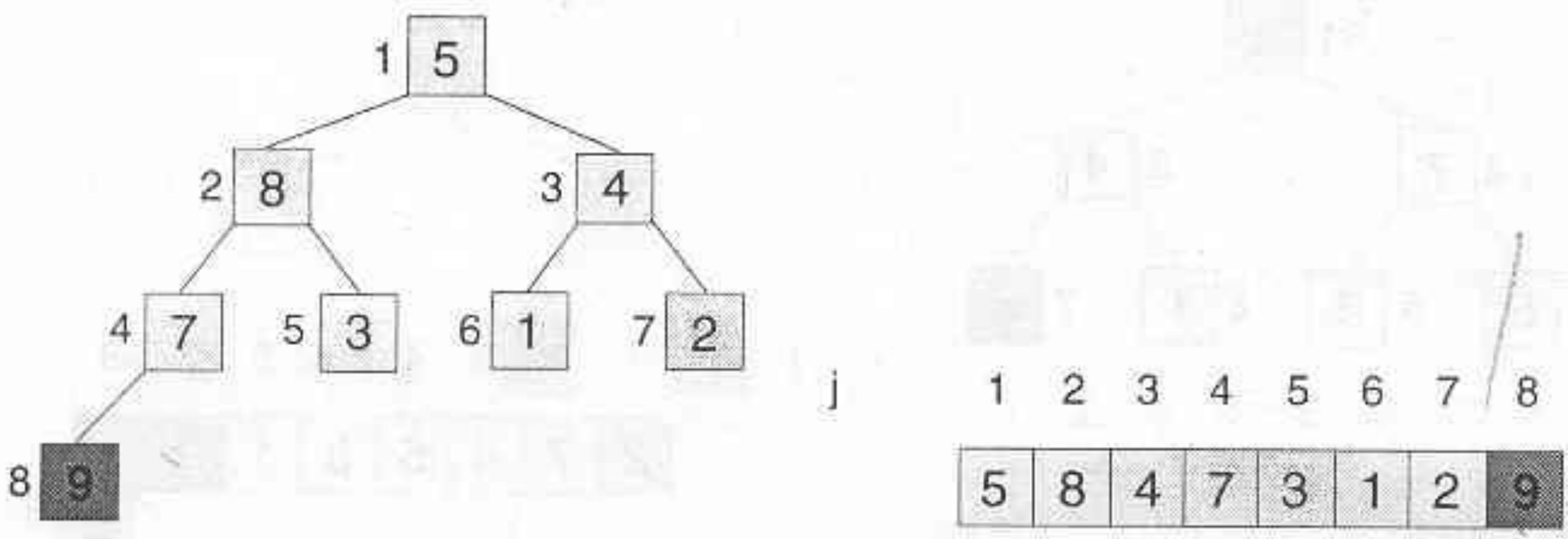
5. Az algoritmus első fázisa



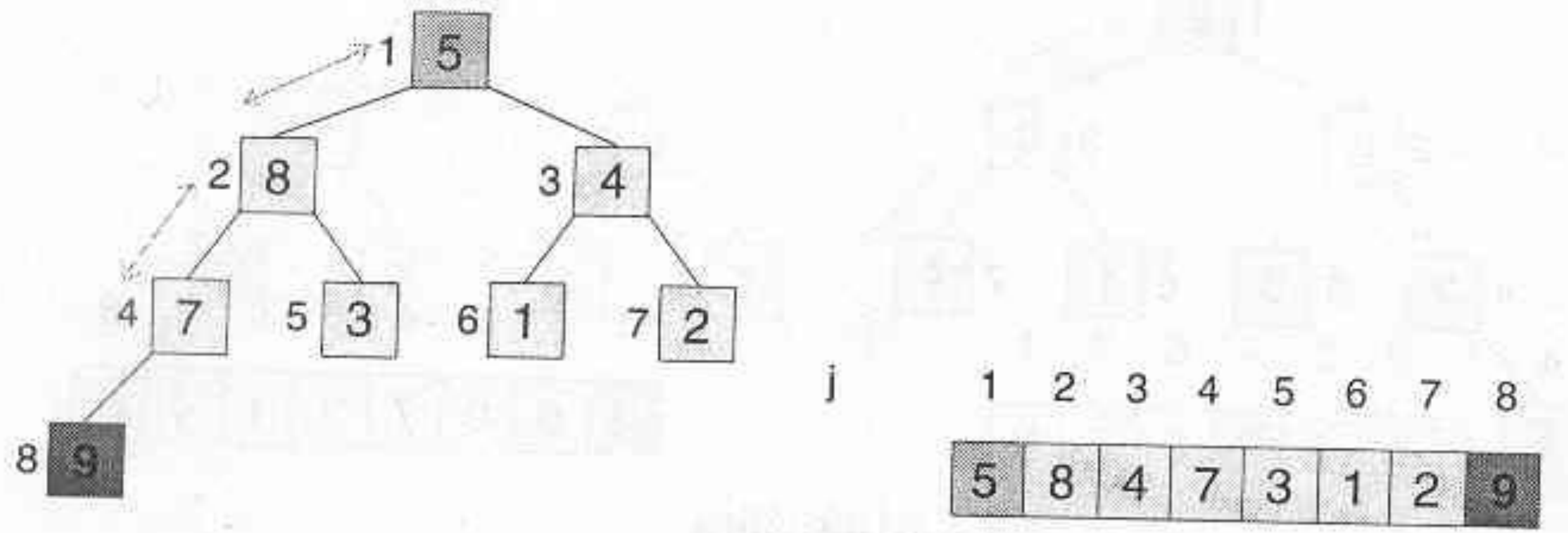
6. Az algoritmus első fázisa
A kupac elkészült



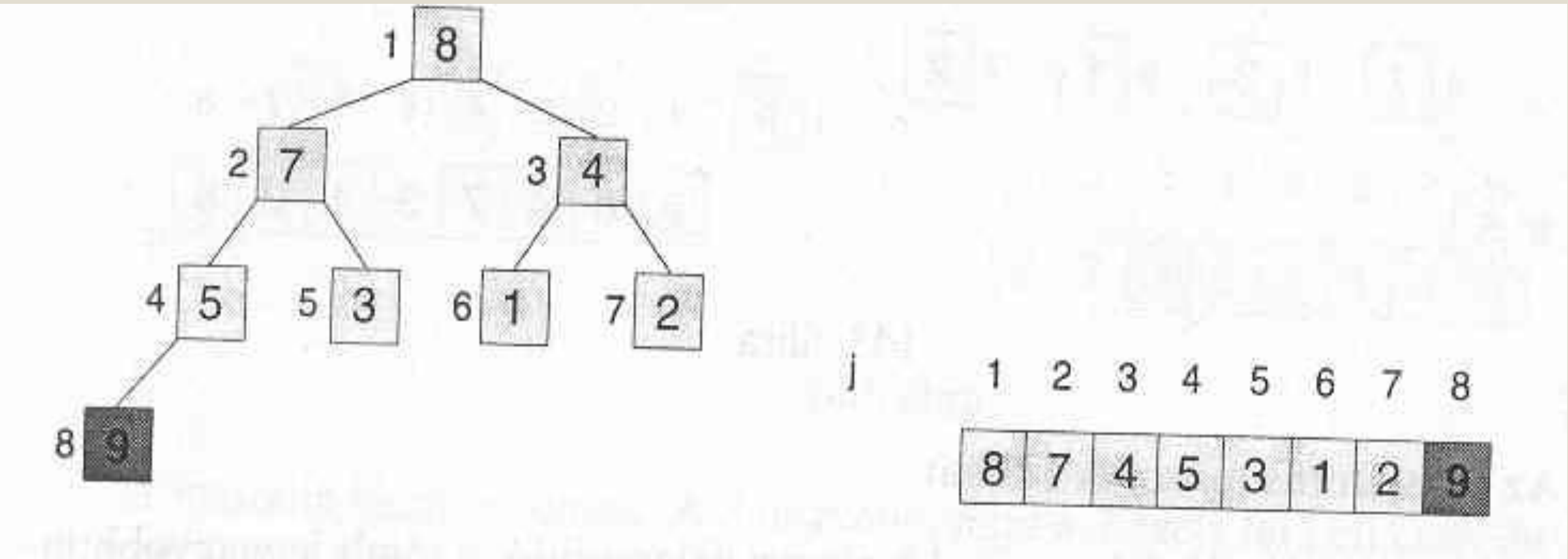
1. Az algoritmus második fázisa



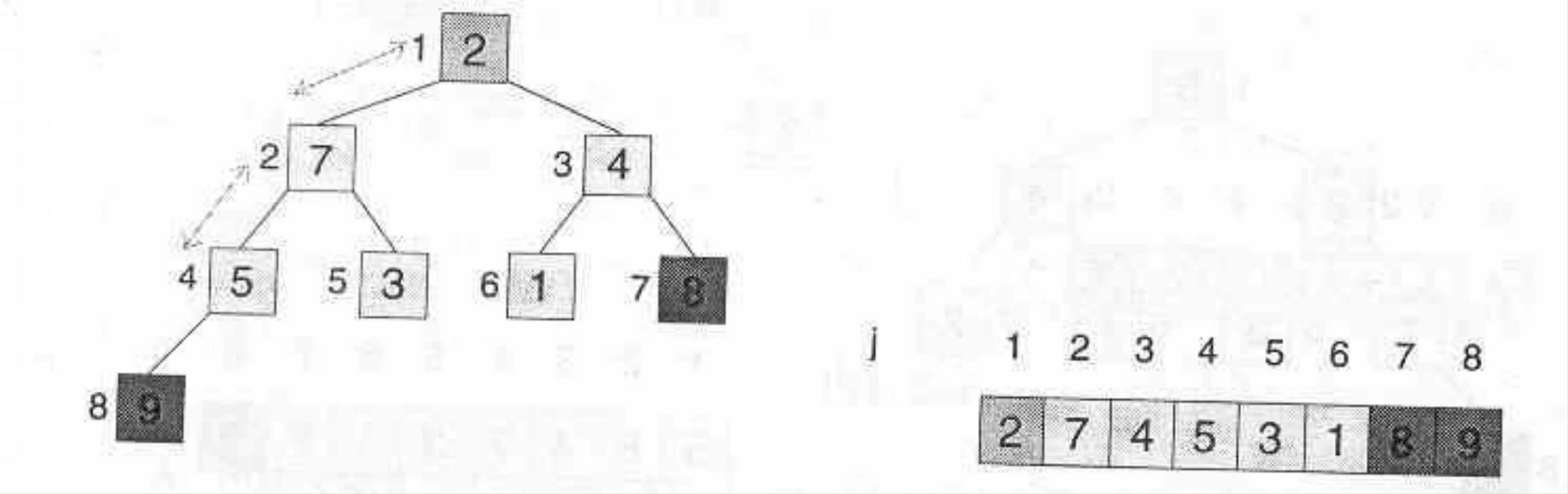
2. Az algoritmus második fázisa



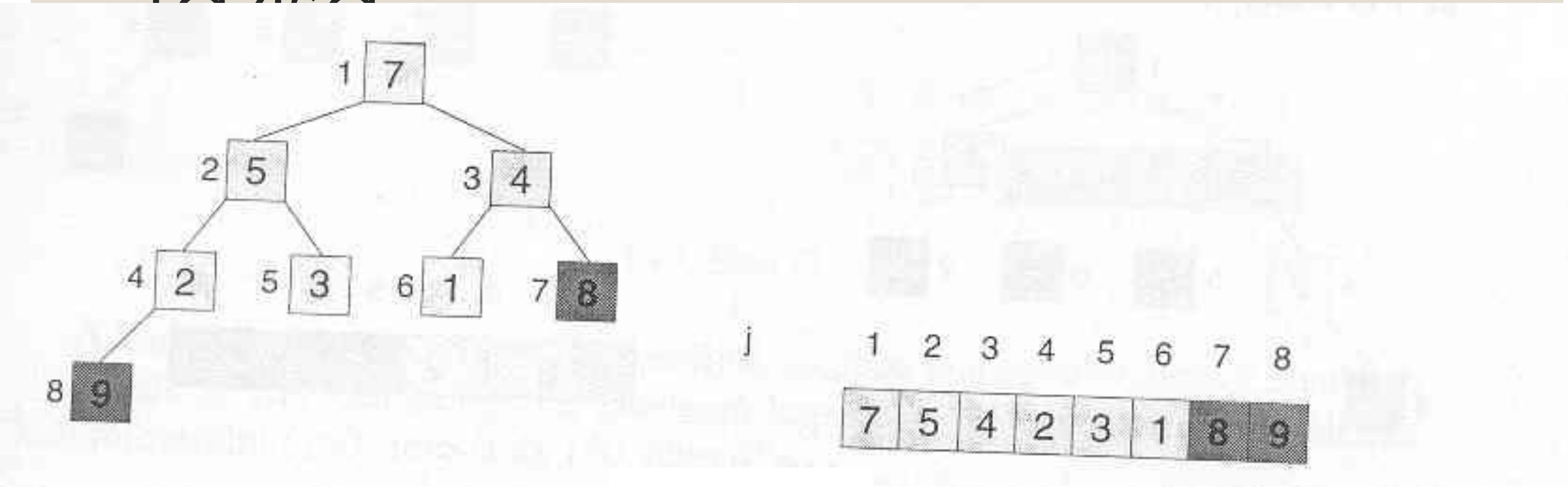
3. Az algoritmus második fázisa



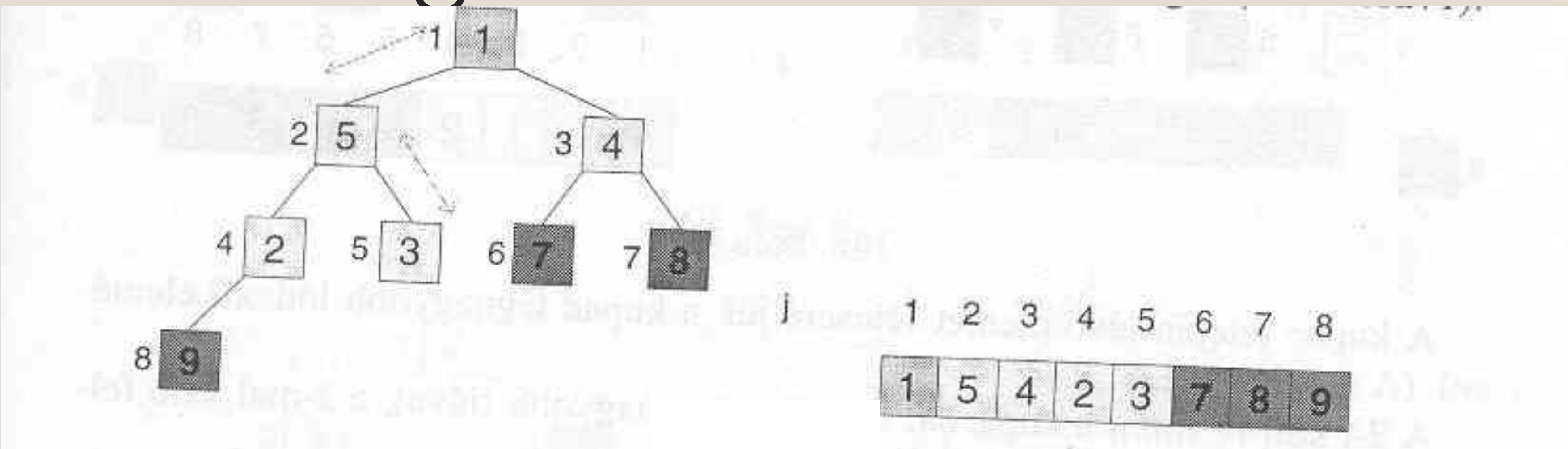
Az algoritmus második fázisa

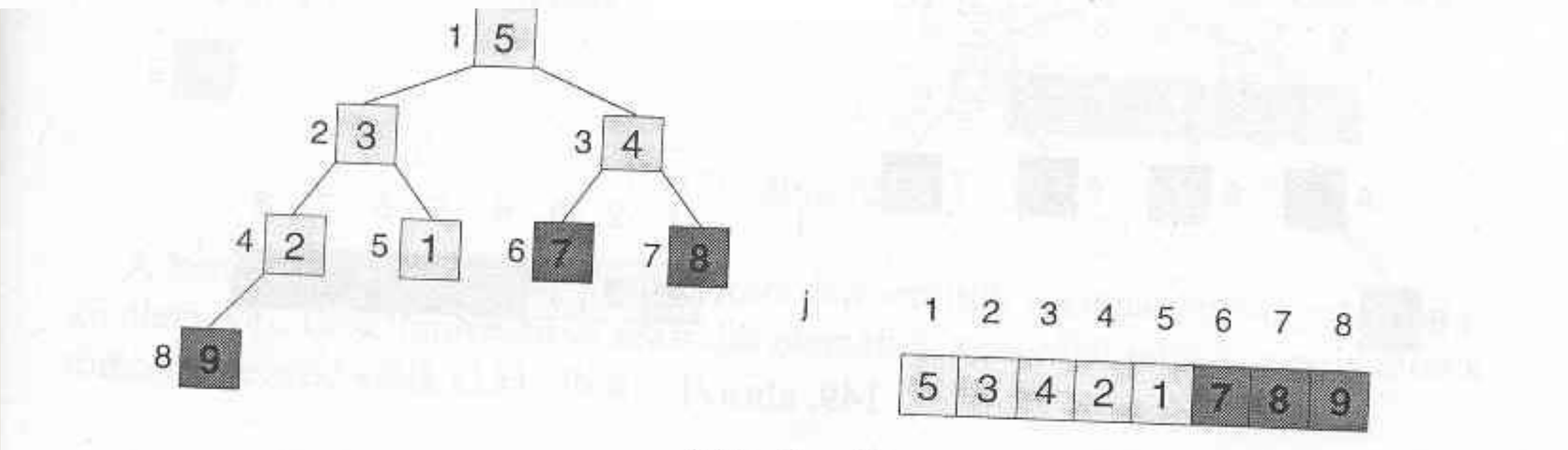


4. Az algoritmus második fázisa

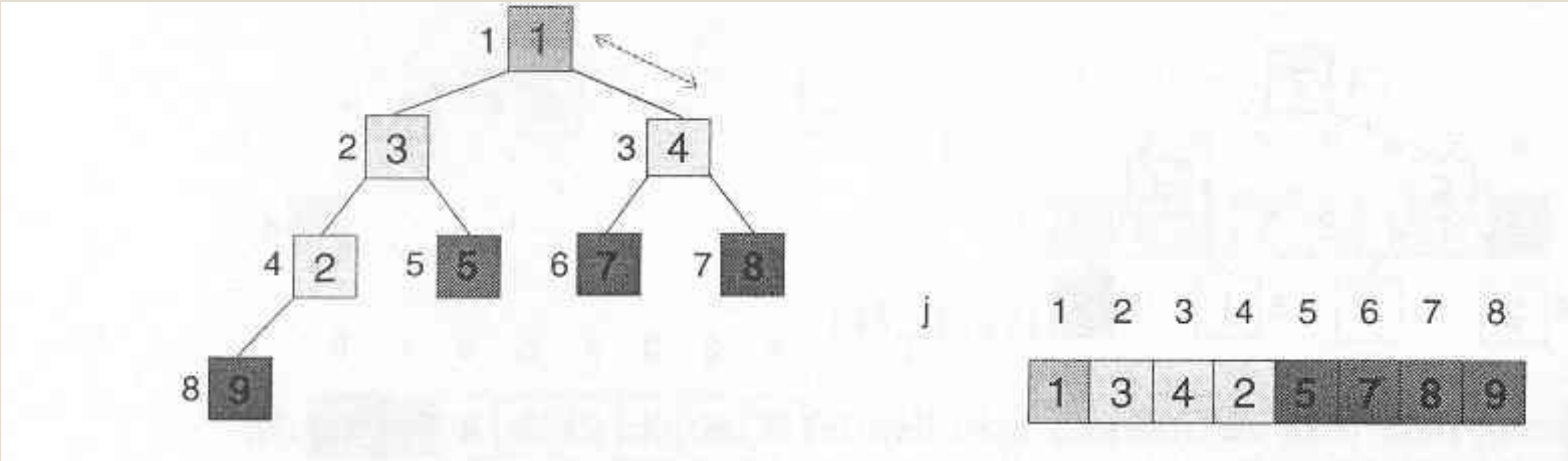


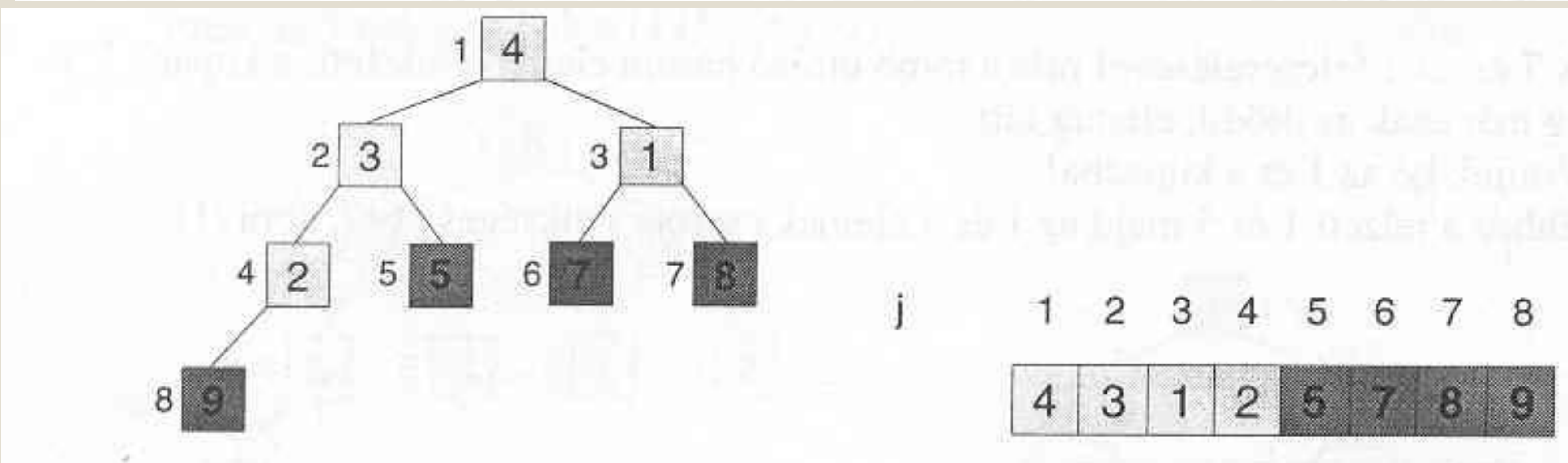
5. Az algoritmus második fázisa



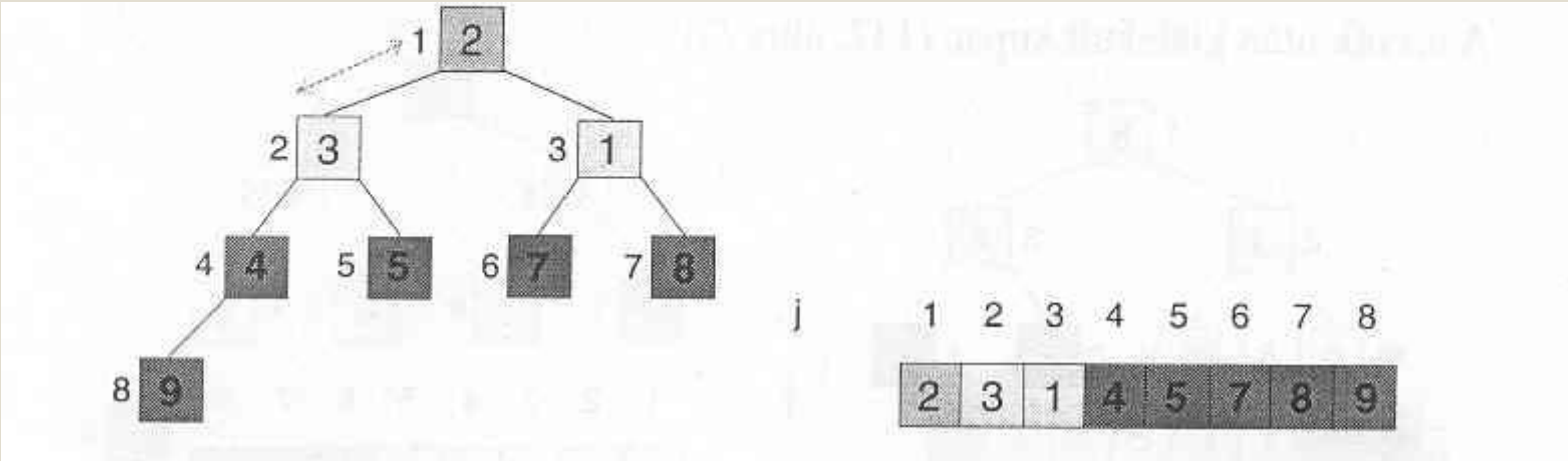


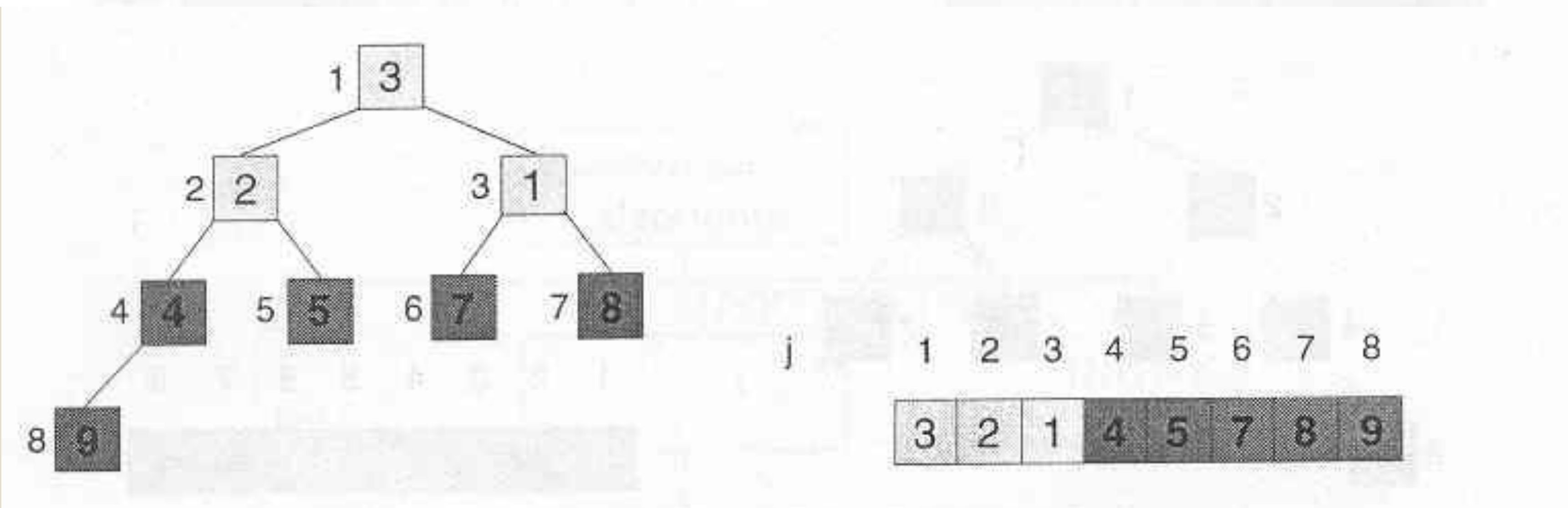
6. Az algoritmus második fázisa



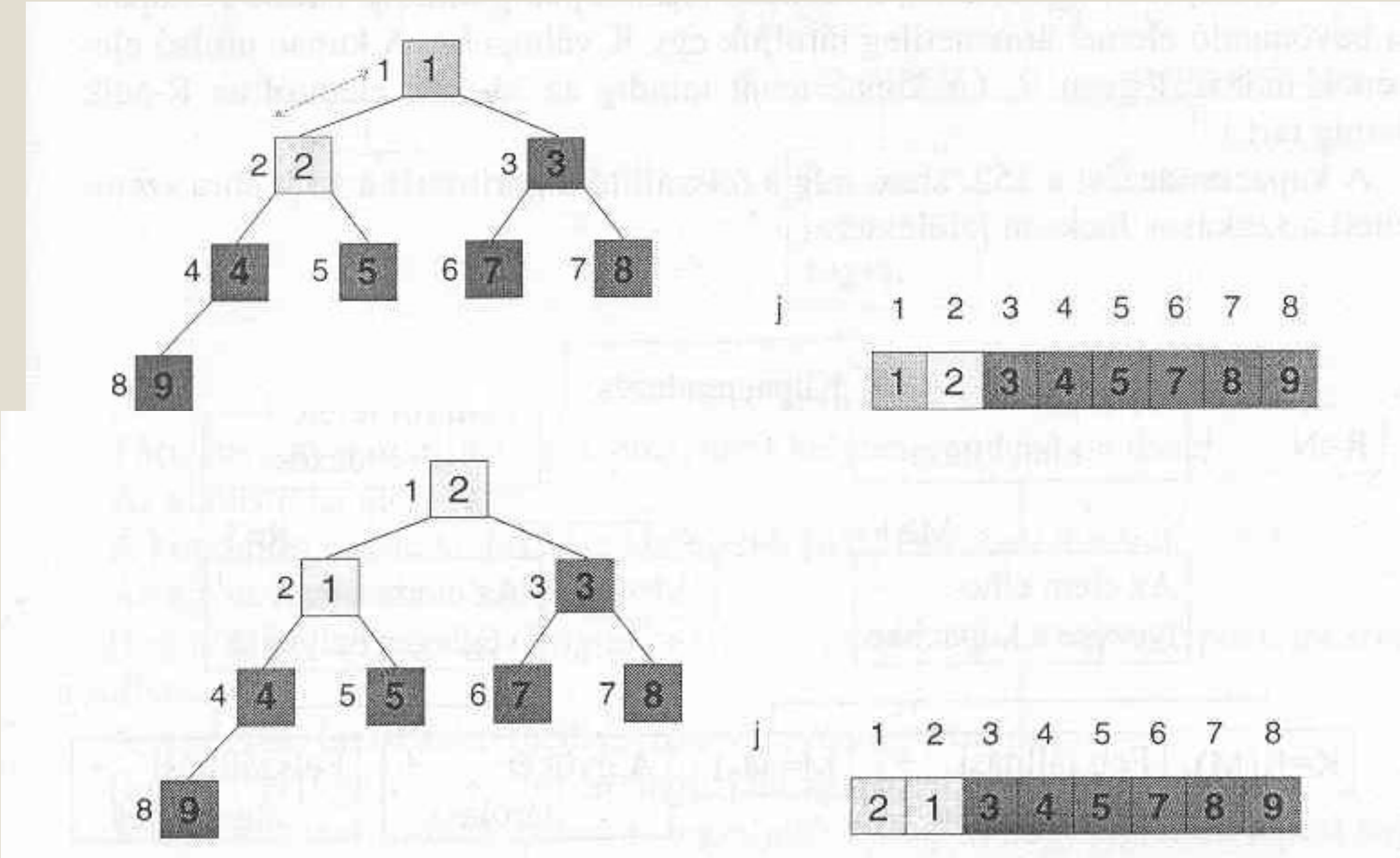


7. Az algoritmus második fázisa

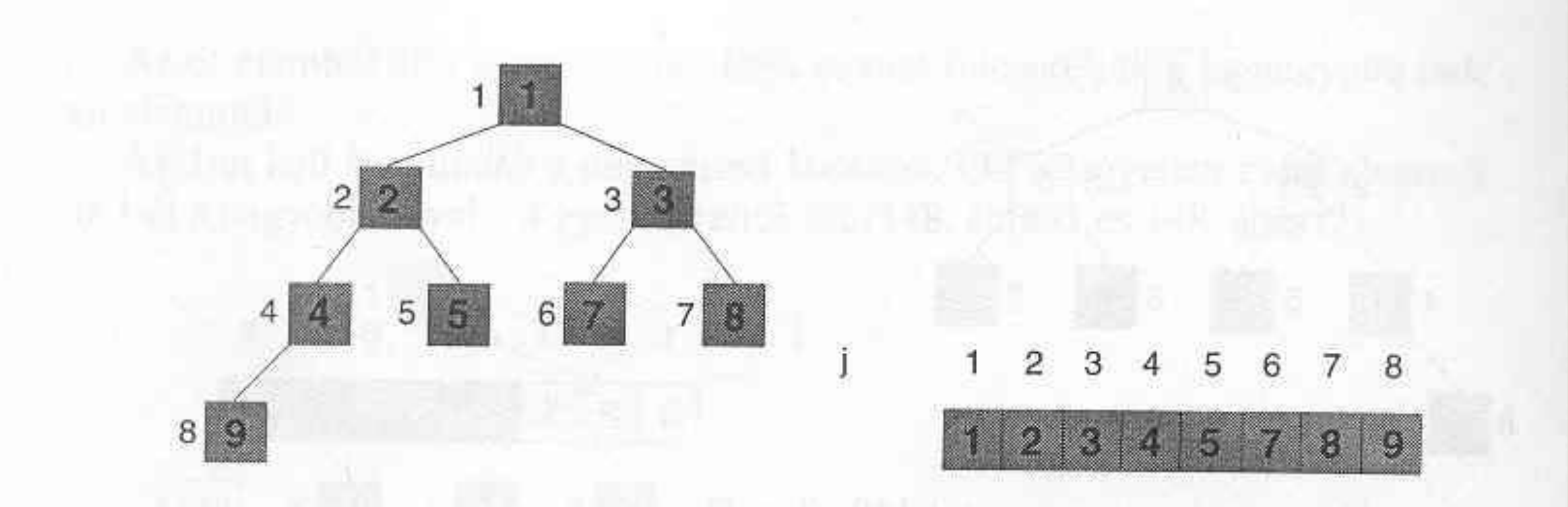




8. Az algoritmus második fázisa



9. Az algoritmus második fázisa



Végeredmény

Algoritmus I. Rendezés

```
Kupacrendezés(T)
     Kupacot épít(T)
     ciklus i=n-1-től 1-ig
       csere(T[0],T[i])
       m=m-1  //kupac méret csökkentése
       maximum-kupacol(T,0)
     ciklus vége
Eljárás vége
```

Algoritmus II. Kupac-építése

```
Kupacot épít(T ) // T tömb
 m=n-1  //utolsó index értékét
 kapja
 ciklus i=(n-1)/2-től 0-ig
     maximum-kupacol(T,i)
 ciklus vége
Eljárás vége
```

Algoritmus II. (rekurzív) Kupactulajdonság fenntartása

```
Maximum-kupacol(T,i)
 L=i*2+1  //bal
 R=i*2+2  //jobb
 ha L<=m és T[L]>T[i]   //m a kupacméret
 akkor legnagyobb=L
 különben legnagyobb=i
 ha r<=m és T[R]>T[legnagyobb]
 akkor legnagyobb=r
 ha legnagyobb!=i
     akkor csere(T[i], T[legnagyobb])
     Maximum-kupacol(T,legnagyobb)
 elágazás vége
Eljárás vége
```

Algoritmus II. Kupacrendezés algoritmusa

```
KupacRend(T/*tömb*/, int n /*hossz*/)
   // épít egy kupacot
      for (int i = n / 2; i >= 0; --i)
                    Sullyeszt(T, i, n);
// a kupac legnagyobb elemét kicseréli az utolsó elemmel
// majd kijavítja a kupacot, ami mostmár nem a teljes, hossz, hanem
  csak az "utolsó" előtti elemtől számít
      for (int i = n; i >= 0; --i)
           { csere(T[0], T[i]);
      Sullyeszt(T, 0, i - 1);
           }
}
```

Algoritmus I. Kupac tulajdonság fenntartása

```
Sullyeszt(T /*kupac*/, int pont, int m)
 int apa = pont; int fiu;  temp =
 kupac[pont];
Ciklus amíg ((fiu = 2*apa + 1) < m )
 //a nulla indexelés miatt a 2*apa + 1 adja a bal gyereket
 //pelda: 8, 2, 5, 7, 6. a 0 indexelés szerinti 1. elem azaz a "2" bal gyermeke a "2*1
 + 1 = 3" indexű elem, azaz a "7".
//  jobb gyerek > bal gyerek, akkor átlép jobb gyerekre
      ha (T[fiu + 1] > T[fiu]) fiu++;// ha
 teljesül a kupac tulajdonság, akkor megáll
      ha (temp >= T[fiu]) break; // a gyereket
 feljebb hozza
      T[apa] = T[fiu]; apa = fiu;
 Ciklus vége // a javítandó elemet beteszi az utolsó emelt gyerek
 helyére
 T[apa] = temp;
Eljárás vége
```

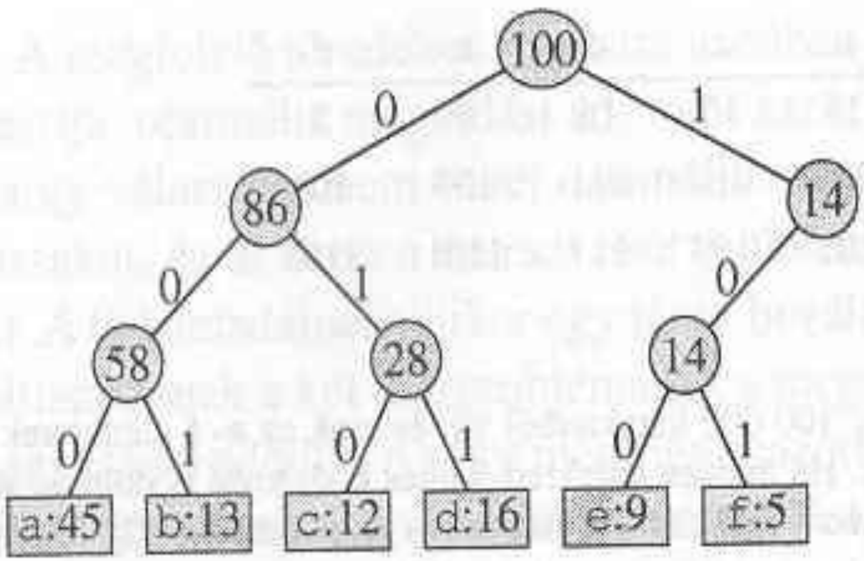
Rendezés jellemzői

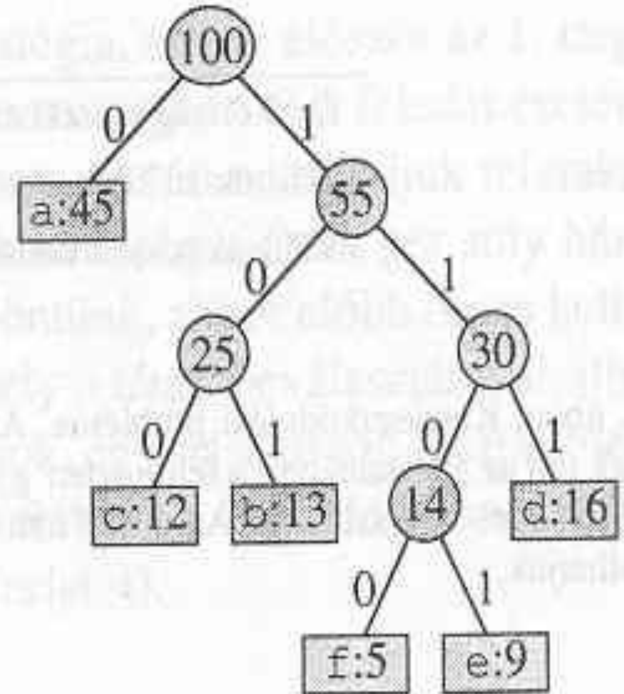
- Tárigény: n (helyben rendező algoritmus)
- Időbonyolultság:
  - Kupac kialakítása:
    - Legjobb eset: 0 mozgatás
    - Legrosszabb eset: n/2 mozgatás
  - Kupac lebontása:
    - log(n-1)+log(n-2)+….<n*log(n)
    - Összesen O(n*log(n)/2)

Huffman-kód

- Széles körben használt, nagyon hatékony módszer állományok tömörítésére
- Az állomány karaktereinek gyakoriságát alapul vevő, állandó vagy változó hosszúságú kód.
- Prefix kód: olyan kódszavakat tartalmaz, amelyekre igaz az, hogy egyik kód sem kezdőszelete a másiknak.
  - Pl.ha a = 0 akkor a többi karakter kódja csak 1-gyel kezdődhet
  - pl. b=10 akkor 10 –val több nem kezdődhet

Huffman-kódhoz tartozó

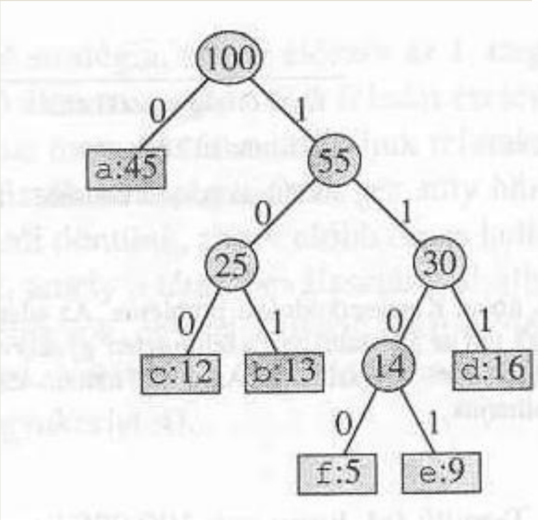




Azonos hosszúságú kódok, minden levélelem ugyanazon a szinten helyezkedik el.

Változó hosszúságú prefix kódot tartalmazó bináris fa ✓

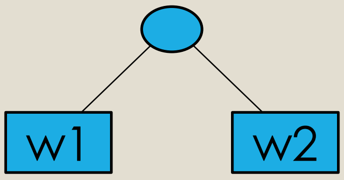
2-es Fa jellemzői:



- Def: közvetlen utódok száma 0 v 2.
- Állítások:
  - Külső csomópontok száma eggyel több mint a belső csomópontok száma
  - A kód hossza a levélelemek magassága
  - Ha az egyes csomópontokban a karakterek gyakorisága, vagy a bal és jobb gyerek súlyának összege a súly, akkor a legtömörebb kódolást a legkisebb súlyozott külső útvonalhosszúságú fával érhetjük el.

Huffmann-algoritmus

- Mohó algoritmus
- Tegyük fel, hogy adott az n darab súly (karakterek gyakorisága), w1,w2…wn növekvő sorrendben.
- T minimális súlyozott útvonal-hosszúságú fa előállítása: w1+w2 külső csomópontot helyettesítjük a részfával



Példa

- Tegyük fel, hogy az A, B, C, D, E, F, G, H karakterekhez az alábbi súlyok tartoznak:

A 22

B 5

C 11

D 19

E 2

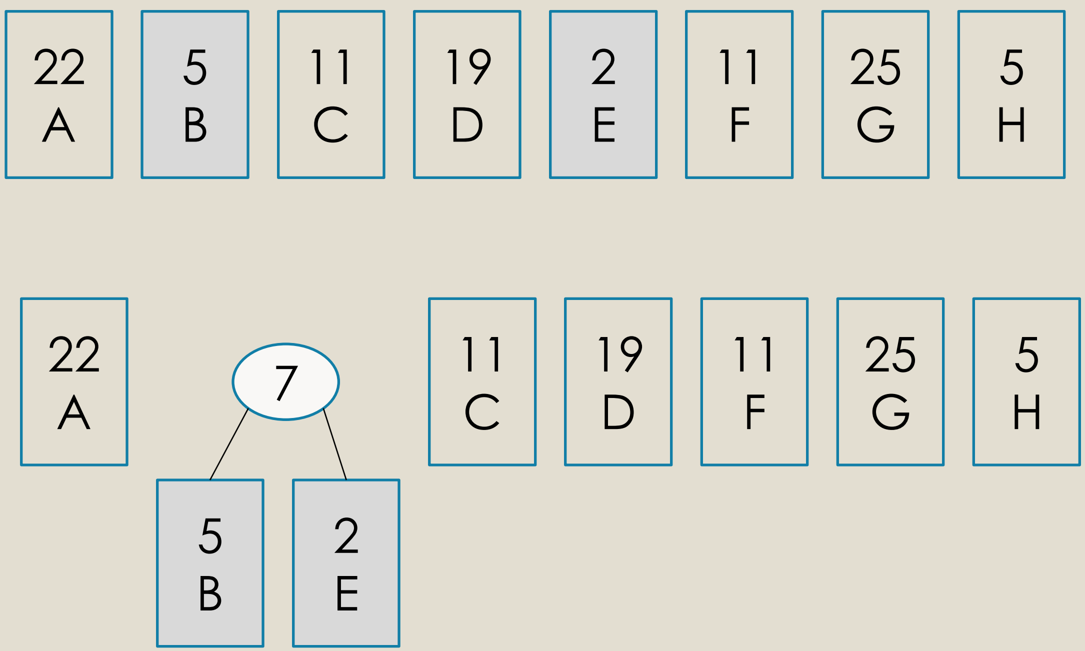
F 11

G 25

H 5

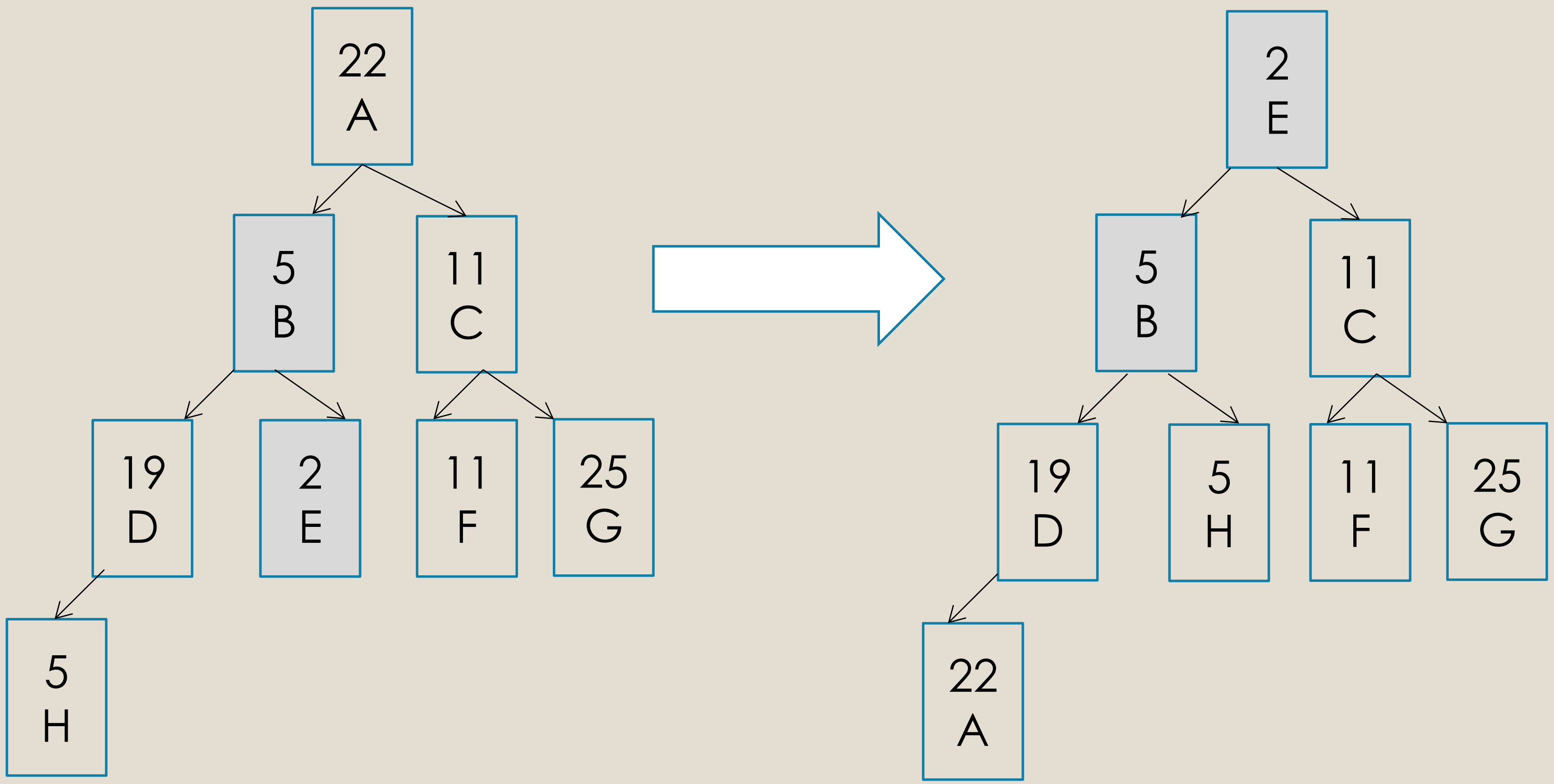
1.

2.

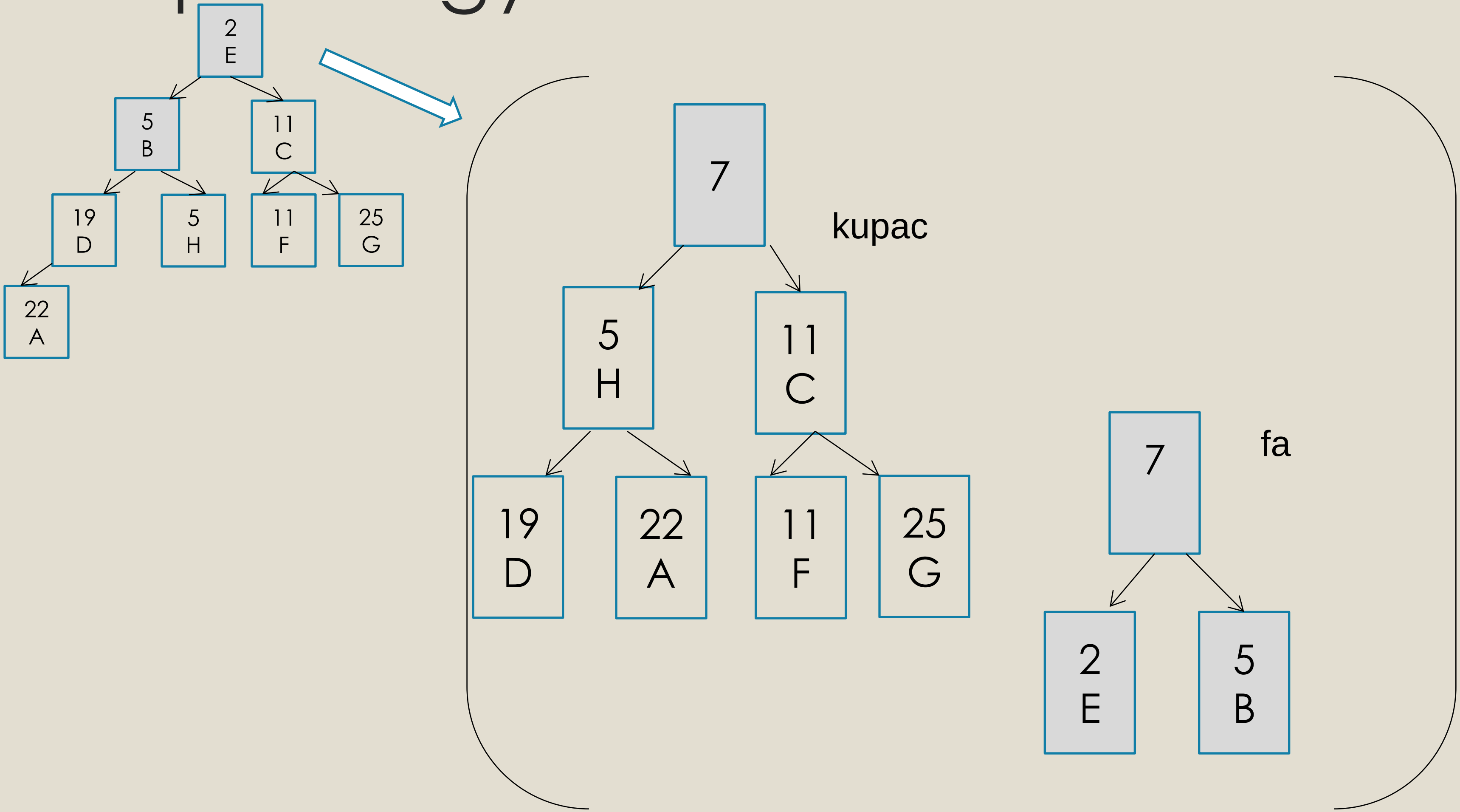


...

I. Fázis minimum-kupac kialakítása



II. Fázis fa-építése minimum-kupac fogyasztása



Algoritmus

```
Huffman(C)
Karaktergyakoriságok megállapítása
Q kupac létrehozása (minimum-kupacol
 eljárással)
Ciklus i=1 től n-1 ig
 új z csúcs létesítése
     bal[z]=x kivesz-min(Q)
     jobb[z]=y kivesz-min(Q)
     beszúr(Q, x gyakorisága+y gyakorisága)
Ciklus vége
Return kivesz-min(Q)
```

Köszönöm a figyelmet!