Alsó korlát rendezési algoritmusokra

Minden olyan rendezési algoritmusnak a futását, amely elempárok egymással való összehasonlítása alapján működik leírja egy bináris döntési fa.

Az algoritmus által a legrosszabb esetben szükséges összahasonlítások számát a neki megfelelő döntési fa magassága adja meg.

Tétel Egy n darab elemet rendező döntési fa magassága $\Omega(nlogn)$.

Bizonyítás A fa leveleinek száma a lehetséges sorrendek száma, azaz $n!$. Ha a fa magassága h, akkor $2^{h-1} \geq n!$ kell teljesüljön, következésképpen $h-1 \geq \lg(n!) \geq lg((n/e)^n) = n\lg n - n\lg e = \Omega(n\lg n)$.

Következmény Minden olyan rendezési algoritmusra, amely csak az elempárok összehasonlítása alapján működik $T_{lr}(n) = \Omega(n\lg n)$.

Számláló rendezés

Amennyiben a rendezendő elemek által felvehető értékek halmazának számossága kicsi, akkor megadható lineáris időigényű algoritmus. A bemenet a rendezendő elemek egy n méretű A tömbben eltárolva, a kimenet ezen számok egy B tömbben rendezetten. Továbbá feltesszük, hogy a rendezendő elemek mindegyike az $\{1,\ldots,k\}$ halmazba esik. $C[i]$ tartalmazza azt az információt, hogy a legfeljebb i nagyságú elemből hány darab van.

```
Szamlalorend(A,B,k)
  for i=1 to k
    C[i]:=0
  for j=1 to n
    C[A[j]]:=C[A[j]]+1
  for i=2 to k
    C[i]:=C[i]+C[i-1]
  for j=n downto 1
     {B[C[A[j]]]:=A[j]
      C[A[j]]:=C[A[j]]-1}
```

Megjegyzés: Az algoritmus ötlete nem csak akkor használható, ha az elemek az $\{1,\ldots,k\}$ halmazba esnek. Általánosabb esetben meg kell keresni a minimális elemet *min*-t, és $C[i]$ a legfeljebb $i-min+1$ nagyságú elemeket tartalmazza.

Definíció Egy rendezést stabilnak nevezünk, ha az azonos értékű elemek ugyanabban a sorrendben jelennek meg a kimeneti tömbben, mint ahogy a bemeneti tömbben szerepeltek.

A leszámláló rendezés stabil.

Futási idő: $\Theta(n+k)$

Példa

```
A=[3,1,3,1,2,1,1,2,3]
C=[4,2,3]
C=[4,6,9]
B=[0,0,0,0,0,0,0,0,3]
C=[4,6,8]
B=[0,0,0,0,0,2,0,0,3]
C=[4,5,8]
B=[0,0,0,1,0,2,0,0,3]
C=[3,5,8]
```

Számjegyes vagy radix rendezés

Példa

```
329 720 720 329
457 436 329 436
657 457 436 457
839 657 839 657
436 329 457 720
720 839 657 839
```

Bemenet: H k dimenziós vektorok halmaza, ahol az i-edik komponens minden i-re egy olyan elemtípusból kerül ki, amelyen adott egy rendezési reláció.

Kimenet: H elemei a lexikografikus rendezés alapján rendezve.

Definíció Egy $X = (x_1, \ldots, x_k)$ vektor megelőzi az $Y = (y_i, \ldots, y_k)$ vektort a lexikografikus rendezésben, ha $\exists i$, amelyre teljesül, hogy $\forall j < i \; x_j = y_j$ és $x_i < y_i$.

Elvi algoritmus

```
for i:=d downto 1
   H Stabil rendezése a kulcs i-edik jegye szerint
```

Helyesség: A ciklusmag végrehajtása után a H halmaz rendezett lesz arra a kulcsra nézve, amely az eredeti kulcs [i..d] jegyeit (karaktereit) tartalmazza. Bizonyítás i-szinti indukcióval.

Futási idő Ha a jegyek szerinti rendezés lineáris idejű (pl számláló), akkor a futási idő $O(dn)$.

Edényrendezés

Tegyük fel, hogy a rendezendő $H = \{a_1, \ldots, a_n\}$ halmaz elemei a $[0,1)$ intervallumba eső valós számok. Vegyünk m db vödröt, $V[0], \ldots, V[m-1]$ és osszuk szét a rendezendő halmaz elemeit a vödrökbe úgy, hogy az a_i elem az $\lfloor m \cdot a_i \rfloor$ sorszámú vödörbe kerüljön. Majd rendezzük az egy vödörbe került elemeket, és a vödrök sorszáma szerinti növekvő sorrendben fűzzük össze a rendezett részsorozatokat (vödröket).

Edényrendezés

For i=1 to n
    Beszúrjuk a_i-t a $B[\lfloor na_i \rfloor]$ listába
For i=0 to m-1
    Rendezzük a $B[i]$ listát beszúró rendezéssel
Összefűzzük a $B[0], B[1], \ldots, B[m-1]$ listákat.

Helyesség: Ha két elem ugyanabba a vödörbe kerül, akkor a beszúró rendezés miatt lesz megfelelő a sorrendjük, egyébként pedig a vödrök összefűzésének sorrendje miatt.

Példa

Legyen $A = \{0.12, 0.21, 0.68, 0.26, 0.72, 0.94, 0.78, 0.17, 0.23, 0.39\}$, és legyen $m = 10$.

Ekkor a vödrök tartalma $B[1] = \{0.12, 0.17\}$, $B[2] = \{0.21, 0.26, 0.23\}$, $B[3] = \{0.39\}$, $B[6] = \{0.68\}$, $B[7] = \{0.72, 0.78\}$, $B[9] = \{0.94\}$.

Futási idő: Tegyük fel, hogy $m = n$, ekkor:

- legjobb eset $\Theta(n)$,
- legrosszabb eset $\Theta(n^2)$,

- átlagos eset $\Theta(n)$.

Kiválasztási probléma

Bemenet: Azonos típusú (különböző) elemek $H = \{a_1, \ldots, a_n\}$ halmaza, amelyeken értelmezve van egy $\leq$ lineáris rendezési reláció és egy i $(1 \leq i \leq n)$ index.

Kimenet: Olyan $x \in H$, hogy $|\{y : y \in H \wedge y \leq x\}| = i$.

A kimenetben szereplő x-et a H rendezett minta i-edik (legkisebb) elemének nevezzük. A középső elemet a rendezett minta mediánjának hívjuk. Pontosabban, az $i = \lfloor (n+1)/2 \rfloor$ -edik elem az alsó, az $i = \lceil (n+1)/2 \rceil$ -edik elem a felső medián.

Minimum és maximum egyidejű választása

A feladat adott n méretű tömb elemeiből kiválasztani a maximális és minimális elemet.

```
MaxMin(A)
if (n%2=0) then
    {k:=2
    if (A[1]<A[2]) then
       {mini:=1
        maxi:=2}
    else
       {mini:=2
        maxi:=1}}
else
    {k:=1
     mini:=1
     maxi:=1}

while(k<n)
    {if A[k+1]<A[k+2] then
        {if A[k+1]<A[mini] then mini:=k+1
         if A[k+2]>A[maxi] then maxi:=k+2}
    else
        {if A[k+2]<A[mini] then mini:=k+2
        if A[k+1]>A[maxi] then maxi:=k+1}
    k:=k+2}
```

Futási idő Az algoritmus legfeljebb $\lfloor 3n/2 \rfloor$ összehasonlítást végez.

Kiválasztás ismételt felosztással

Az algoritmus használja a gyorsrendezésnél definiált Feloszt(A,p,r) eljárást, amely átrendezi az A tömb A[p] és A[r] közé eső szakaszát úgy, hogy a visszaadott q értékre $A[p, \ldots, q]$ minden eleme kisebb vagy egyenlő $A[q+1, \ldots, r]$ minden eleménél. Az alábbi algoritmus a Hoare felosztást használja, ahol nem garantált, hogy $A[p, \ldots, q-1]$ minden eleme kisebb vagy egyenlő, mint $A[q]$.

Elvi algoritmus

Kiválaszt(A,p,r,i)

If p=r

```
    Then return A[p]
q:=Feloszt(A,p,r)
k:=q-p+1
if i ≤ k
    Return Kiválaszt(A,p,q,i)
else
    Return Kiválaszt(A,q+1,r,i-k)
```

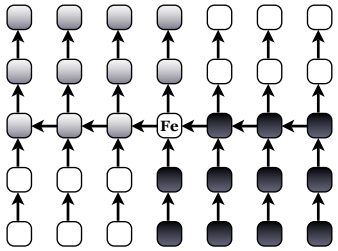
Futási idő: Legjobb eset: $O(n)$. Legrosszabb eset: $O(n^2)$. Átlagos eset: $O(n)$.

Kiválasztás lineáris időben

Linkiválaszt algoritmus

- 1. Osszuk a bemeneti tömb n elemét $\lfloor n/5 \rfloor$ 5 elemű csoportra, a maradék elemekből alkossunk egy további csoportot.
- 2. Minden csoportnak keressük meg a mediánját úgy, hogy az 5 elemű csoportokat (és az esetleges egyetlen kisebb elemszámú csoportot) rendezzük.
- 3. A Linkiválaszt algoritmus rekurzív hívásával határozzuk meg a kapott mediánokból álló halmaz mediánját.
- 4. A Feloszt eljárás alapján osszuk fel az így kiválasztott elem körül a bemeneti tömböt, legyen k az elemnél nem nagyobb tőle különböző elemek száma, $n-k-1$ az elemnél nagyobb elemek száma.
- 5. A Linkiválaszt algoritmus rekurzív hívásával határozzuk meg a baloldali részben az i-edik elemet, ha $i \leq k$, illetve a jobboldali részben az $i-k-1$-edik elemet, ha $i > k+1$. Az $i = k+1$ esetben a felosztó elem a keresett elem.

Elemzés



1. ábra.

Az ábrán a sötét elemek kisebbek a felosztó elemnél, a világosak nagyobbak, így a rekurzív hívás során legfeljebb $7n/10+6$ elemre kell végrehajtani az algoritmust. Tehát elegendően nagy n esetén a legrosszabb eset korlátra fennáll a következő rekurzív összefüggés:

$$T_{lr}(n) \leq T_{lr}(\lceil n/5 \rceil) + T_{lr}(7n/10+6) + O(n).$$

Teljes indukcióval igazolható, hogy $T_{lr}(n) = O(n)$.

Megvalósítás: Az 5-ös csoportokat úgy választjuk ki, hogy a tömb első, második, harmadik, negyedik és ötödik ötödéből is veszünk egy-egy elemet, így az ötösöket tudjuk úgy rendezni, hogy a mediánok a tömb közepső részére kerüljenek. Ezáltal a Linkiválaszt algoritmus helyben megvalósítható csak a kiindulási tömb felhasználásával.

Kiválasztás kupaccal

Az algoritmus során a for ciklus j értékkel való végrehajtása után a kupac az első j elemből az i legkisebbet tartalmazza.

```
Kupacvalaszt(A,i)
 Kupacmeret(A):=i
 for j= [i/2] to 1
    MAXIMUM KUPACOL(A,j)
 // az elso i elembol egy kupac
 for j=i+1 to n
    {if A[1]>A[j] then
        {Csere(A[1],A[j])
        MAXIMUM KUPACOL(A,1)}}
return A[1]
```

Futási idő: $O(i)+(n-i)O(\log i)$

Példa

```
A=[2,5,7,3,6,4,8], k=3
A=[2,5,7|3,6,4,8]
A=[7,5,2|3,6,4,8]
A=[3,5,2|7,6,4,8]
A=[5,3,2|7,6,4,8]
A=[4,3,2|7,6,5,8]
```

Bináris keresőfák

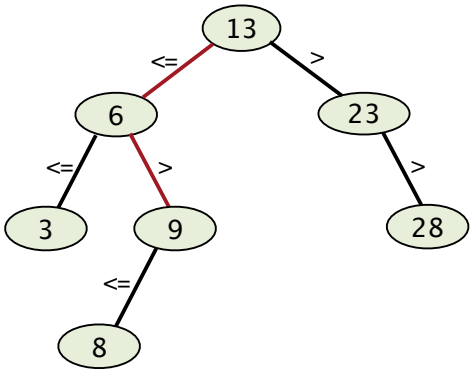
Az F = (M,R,Adat) absztrakt adatszerkezetet bináris keresőfának nevezzük, ha

- F bináris fa, $R=\{bal, jobb, apa\}$, $bal, jobb, apa : M \to M$,
- $Adat : M \to Elemtip$ és $Elemtip$-on értelmezett egy $\leq$ lineáris rendezési reláció,
- $(\forall x \in M)(\forall p \in F_{bal(x)})(\forall q \in F_{jobb(x)})(kulcs(p) \leq kulcs(x) \leq kulcs(q))$

```
InorderBejaras(F,M)
 if F!=Nil
 then {InorderBejaras(bal(F),M)
      M(F)
      InorderBejaras(jobb(F),M)}
```

Az InorderBejaras algoritmus a bináris keresőfa elemeit a kulcsok rendezés szerinti sorrendjében látogatja meg.

Adott kulcsú elem keresése



2. ábra.

```
KERES(F,k)
 if F=Nil or k=kulcs(F)
    then return F
 If k< kulcs(F)
    then return KERES(bal(F),k)
    else return KERES(jobb(F),k)


KERES2(F,k)
 while(F!=Nil or k!=kulcs(F))
   {if k<kulcs(F)
       then F:=bal(F)
       else F:=jobb(F)}
 return F
```

Futási idő: A fa magasságával arányos.

Minimális maximális elem keresése

```
FabanMinimum(F)
 while(bal(F)!=Nil)
      F:=bal(F)
 return F
```

```
FabanMaximum(F)
 while(jobb(F)!=Nil)
      F:=jobb(F)
 return F
```

Futási idő: A fa magasságával arányos.

Rákövetkező, megelőző elem keresése

```
FabanKovetkezo(p)
 if jobb(p)!=Nil
    then return FabanMinimum(jobb(p))
 q:=apa(p)
 while(q!=Nil and p=jobb(q))
    {p:=q
     q:=Apa(q)}
 return q


FabanMegelozo(p)
 if bal(p)!=Nil
    then return FabanMaximum(bal(p))
 q:=apa(p)
 while(q!=Nil and p=bal(q))
    {p:=q
     q:=Apa(q)}
 return q
```

Futási idő: A fa magasságával arányos.

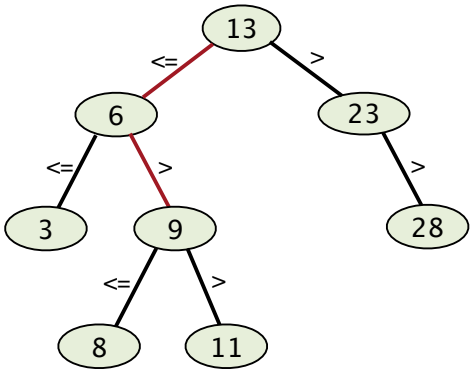
Beszúrás bináris keresőfába

A fát a gyökérpontja által adtuk meg.

```
Beszur(F,z)
  y:=Nil
  x:=F
  while(x!=Nil)
     {y:=x
     if kulcs(z)<kulcs(x)
        then x:=bal(x)
        else x:=jobb(x)}
  apa(z):=y
  if y=Nil
     then F:=z    //Üres volt a fa
     else {if kulcs(z)<kulcs(y)
          then bal(y):=z
          else jobb(y):=z}
```

Futási idő: A fa magasságával arányos.

Törlés bináris keresőfából



3. ábra.

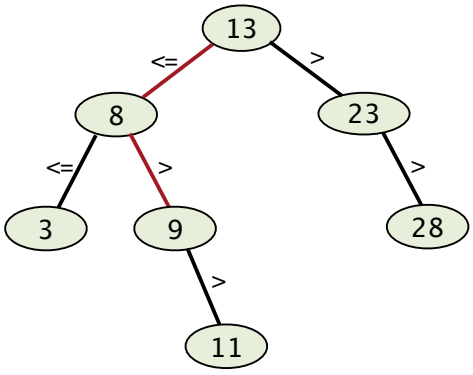
```
FabolTorol(F,z)
 If bal(z)=Nil or jobb(z)=Nil
    then y:=z
    else y:=FabanKovetkezo(z)
 If bal(y)!=Nil
     then x:=bal(y)
     else x:=jobb(y)
 If x!=Nil
     then apa(x):=apa(y)
 If apa(y)=Nil
    then F:=x
    else {if y=bal(apa(y))
         then bal(apa(y)):=x
         else jobb(apa(y)):=x}
 If y!=z
     then kulcs(z):=kulcs(y)
```

Keresőfák magassága

Ha az $1, 2, \ldots, n$ sorrendben szúrunk pontokat egy üres fába a magasság n lesz.
Az n csúcsból álló véletlen építésű bináris fák magassága $O(\log n)$.

Kiegyensúlyozott keresőfák

Mivel a keresőfa műveletek időigénye a fa magasságával arányos, ezért szeretnénk a fát úgy karbantartani, hogy



4. ábra.

a magassága mindig $O(\log n)$ legyen, ahol n a fa pontjainak a száma.

- AVL fák
- piros fekete fák

Sorozat adattípus

Értékhalmaz: E $Sor = [a_1, \ldots, a_n : a_i \in E, i = 1, \ldots, n,]$ Műveletek: S: Sor, x : E

- $\{Igaz\}$ Letesit(S) $\{S = []\}$
- $\{S = S\}$ Megszuntet(S) $\{Igaz\}$
- $\{S = S\}$ Uresit(S) $\{S = []\}$
- $\{S = [a_1, \ldots, a_n]\}$ Elemszam(S) $\{Elemszam = n\}$
- $\{S = [a_1, \ldots, a_n] \wedge 1 \leq i \leq n\}$ Kiolvas(S, i,x) $\{x = a_i \wedge S = Pre(S)\}$
- $\{S = [a_1, \ldots, a_i, \ldots, a_n] \wedge 1 \leq i \leq n\}$ Modosit(S, i,x) $\{S = [a_1, \ldots, x, \ldots, a_n]\}$
- $\{S = [a_1, \ldots, a_i, a_{i+1}, \ldots, a_n] \wedge 0 \leq i \leq n\}$ Bovit(S, i,x) $\{S = [a_1, \ldots, a_i, x, a_{i+1}, \ldots, a_n]\}$
- $\{S = [a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n] \wedge 1 \leq i \leq n\}$ Torol(S, i) $\{S = [a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n]\}$
- $\{S = S\}$ IterKezd(S, I) $\{\}$

- $\{I = I\}$ IterAd(I,x) $\{\}$
- $\{I = I\}$ IterVege(I) $\{\}$

Megvalósítás: kiegyensúlyozott bináris keresőfával.

Tároljuk az $S = \{a_1, \ldots, a_n\}$ sorozatot egy F bináris fában úgy, hogy F inorder bejárása az S sorozatot adja. Ekkor F egy bináris keresőfa lesz a sorozat indexeire nézve.

A fa minden p pontjában tároljuk még extra információként a bal-részfájában levő pontok számánál 1-el nagyobb értéket $s(p)$. Ekkor $s(p)$ a p pontra végrehajtott inorder bejárás során p sorszáma. Ezen $s(p)$ értékek alapján a fa gyökeréből indulva megtalálható minden i-re az i-edik elem.

Bővítés esetén ha a keresőút balra halad egy p ponttól, akkor $s(p)$ értékéhez egyet kell adni. Törlés esetén, ha a keresőút balra halad egy p ponttól, akkor $s(p)$ értékéből egyet le kell vonni.