Algoritmusok és adatszerkezetek 2 gyakorlati anyag

– készült Berend Gábor gyakorlati anyaga alapján –

Vetráb Mercedes
2024

1. gyakorlat – Bináris keresőfák

FOGALMAK

FA: Olyan körmentes, összefüggő gráf,
BINÁRIS: melynek minden csúcsából maximum 2 gyermek származik,
KERESŐ: és teljesül rá, hogy:

- a bal oldali fiú (ha létezik), mindig kisebb az apjánál
- a jobb oldali fiú (ha létezik), mindig nagyobb az apjánál

KERESÉS:

- **Legkisebb elem:** a gyökértől végig balra haladva
- **Legnagyobb elem:** a gyökértől végig jobbra haladva
- **Rákövetkező elem (tehát a nála egyel nagyobb):** x gyökerű fában x rákövetkezője
  - ha x-nek van **jobb részfája**, akkor a részfa **legkisebb eleme**
  - ha x-nek nincs jobb részfája, kezdjünk el felfele haladni, egészen addig, amíg olyan **szülőt** találunk ami **bal fiú**, ekkor a bal fiú szülője a keresett elem
  - ha x-nek nem volt jobb részfája és nem találtunk olyan szülőt ami bal fiú, akkor **nincs rákövetkező** elem
- **Megelőző elem (tehát a nála egyel kisebb):** egy x gyökerű fában x megelőzője
  - ha x-nek van **bal részfája**, akkor a részfa **legnagyobb eleme**
  - ha x-nek nincs bal részfája, kezdjünk el felfele haladni, egészen addig, amíg olyan **szülőt** találunk ami **jobb fiú**, ekkor a jobb fiú szülője a keresett elem
  - ha x-nek nem volt bal részfája és nem találtunk olyan szülőt ami jobb fiú, akkor **nincs megelőző** elem

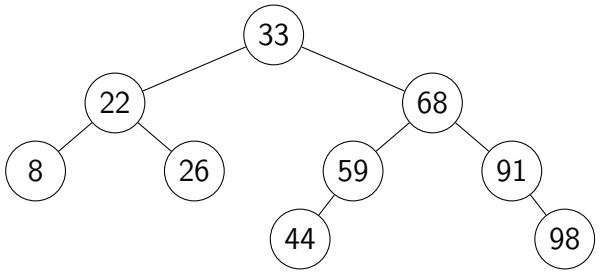
BESZÚRÁS: x elem beszúrásakor induljunk el a gyökérből, ha x $<$ gyökér reláció igaz/hamis haladjunk tovább balra/jobbra egészen addig, míg NULL csúcshoz nem érünk.
Optimális fa készítéséhez rendezzük az elemeket **növekvő sorrendbe**. Szúrjuk be az így kapott **lista középső elemét** (kettő ilyen esetén a kisebbet). **Rekurzívan** a középső beszúrásának módszerével szúrjuk be a középsőnél kisebb elemeket, majd a középsőnél nagyobbakat.

TÖRLÉS: x törlése a fából

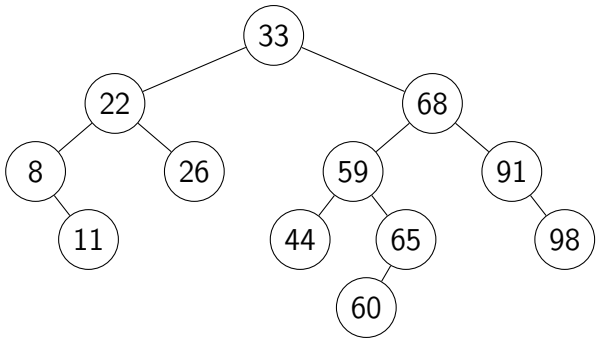
- ha x-nek **nincs gyereke**, akkor x apjának az x-re vonatkozó mutatóját **NULL**-ra állítjuk
- ha x-nek pontosan **1 gyereke van**, akkor x gyereke kerüljön **x helyére**
- ha x-nek **2 gyereke van**, akkor x-et a **megelőzőjével** helyettesítjük

FELADATOK

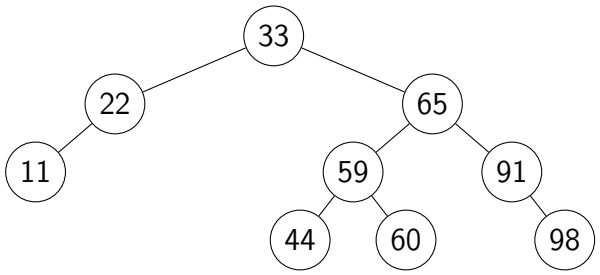
Vegyük az alábbi bináris keresőfát!



a) Keressük meg a fában a 11, 89, 44, 90 kulcsokat!
b) Keressük meg a 11, 59 megelőzőjét/rákövetkezőjét?
c) Szúrjuk be a fába a 11, 65, 60 kulcsokat!



d) Töröljük a beszúrások után előálló fából a 26, 8, 68 kulcsokat!



Tegyük fel, hogy egy bináris keresőfában a 15-ös elemet keressük. Lehetséges keresési sorozat-e az alábbi: 20, 9, 12, 8, 15?

Nem, ugyanis 15 nagyobb, mint a 12, ezért a 8-as a 12 jobb fia kellett, hogy legyen, de ez ellentmondás, hiszen a 12 jobb részfájában nem lehetnek nála kisebb elemek. Szintén helytálló az az észrevétel, hogy a 9-es kulcs jobboldali részfájában nem lenne szabad 9-nél kisebb kulcsot találjunk.

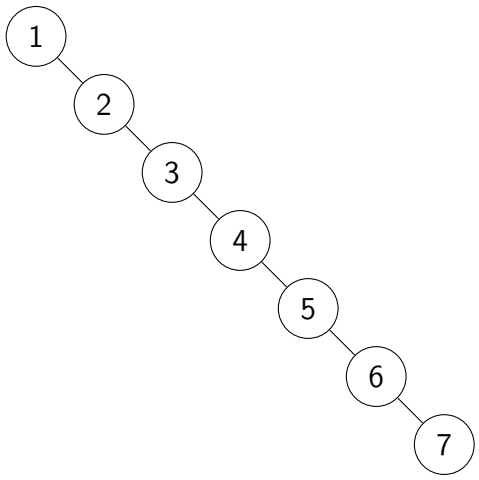
Írassuk ki a fa elemeit!

```
Kulcs szerint növekvő sorrendben:
     void inorder(x) {
       if(x!=nil) {
         inorder(x.bal);
         print(x.kulcs);
         inorder(x.jobb);
       }
     }
```
Végeredmény: 11, 22, 33, 44, 59, 60, 65, 91, 98

```
Postorder sorrendben:
     void postorder(x) {
       if(x!=nil) {
         postorder(x.bal);
         postorder(x.jobb);
         print(x.kulcs);
       }
     }
```
Végeredmény: 11, 22, 44, 60, 59, 98, 91, 65, 33

```
Preorder sorrendben:
     void preorder(x) {
       if(x!=nil) {
         print(x.kulcs);
         preorder(x.bal);
         preorder(x.jobb);
       }
     }
```
Végeredmény: 33, 22, 11, 65, 59, 44, 60, 91, 98

Szúrjuk be egy üres bináris keresőfába a következő elemeket a megadott sorrendben: 1,2,3,4,5,6,7.



2. gyakorlat – AVL-fák és B-fák

FOGALMAK

A bináris keresőfák nagy hátránya volt, hogyha szerencsétlen sorrendben szúrjuk be a csúcsokat, akkor a későbbiekben a fában történő **keresés átlagos ideje aránytalanul megnövekszik**. Ennek kiküszöbölésére, minden beszúrás után próbáljuk meg arra kényszeríteni a fát, hogy végig **teljes- vagy majdnem teljes-bináris fa maradjon**. Hogyan érhetjük el, hogy egy fa (más szóval mondva) kiegyensúlyozott maradjon?

1. módszer - AVL-fa: egy olyan kiegyensúlyozott bináris keresőfa, melyben minden csúcs **kiegészítő információja** megmondja, hogy **milyen magas a belőle induló részfa**. Ahhoz, hogy kiegyensúlyozott legyen teljesülnie kell minden x csúcsra, hogy x jobb és bal oldali részfamagasságainak különbsége abszolútértékben nem nagyobb, mint 1, azaz **|x.bal.kiegeszito-x.jobb.kiegeszito|<= 1**. Ezt a tulajdonságot a beszúrás és törlés művelettel meg tudjuk sérteni, így forgatással javítanunk kell. A forgatás megváltoztatja a fa szerkezetét, miközben megőrzi a bináris keresőfa tulajdonságot (adott csúcstól balra csak kisebb, jobbra pedig csak nagyobb leszármazottak vannak).
X gyökerű részfa **magassága**: $max(x.bal.kiegeszito, x.jobb.kiegeszito) + 1$
X gyökerű részfa **egyensúlyi faktora**: $x.bal.kiegeszito - x.jobb.kiegeszito$

JOBBRA FORGATÁS (x-gyökér, y-bal fiú): y lesz az új gyökér; y kisebb volt, mint x, ezért x lesz y új jobb fia; y korábbi jobb fia nagyobb mint y, de kisebb, mint x, ezért x bal fiának kötjük be. A forgatás után az **új kiegészítő információk** az alábbiak lesznek:
$y.kiegeszito = x.kiegeszito - 1$
$x.kiegeszito = max(x.bal.kiegeszito, regiY.jobb.kiegeszito) + 1$

BALRA FORGATÁS (x-gyökér, y-jobb fiú): y lesz az új gyökér; y nagyobb volt, mint x, ezért x lesz y új bal fia; y korábbi bal fia kisebb mint y, de nagyobb, mint x, ezért x jobb fiának kötjük be. A forgatás után az **új kiegészítő információk** az alábbiak lesznek:
$y.kiegeszito = x.kiegeszito - 1$
$x.kiegeszito = max(x.jobb.kiegeszito, regiY.bal.kiegeszito) + 1$

KERESÉS, BESZÚRÁS és TÖRLÉS: A három művelet a korábban tanultak analógiáján működik, azonban fontos, hogy az **egyensúlyi helyzet** a BESZÚRÁS és a TÖRLÉS műveletek hatására felborulhat, így minden ilyen művelet után **frissítenünk kell a kiegészítő információkat, majd ellenőriznünk** kell, hogy továbbra is fennáll-e az egyensúlyi tulajdonság. Abban az esetben, ha valamelyik csúcsban sérül a kiegyensúlyozottság szabálya, akkor **csírájában folytjuk el** a bajt.
A sérülés gyökere legyen x, a nagyobb magasságú (nehezebb) fia pedig y:

- ha, x bal nehéz és y jobb nehéz, akkor **cikk-cakk** forgatást csinálunk (y balra majd x jobbra forgatása)
- ha, x jobb nehéz és y bal nehéz, akkor **cikk-cakk** forgatást csinálunk (y jobbra majd x balra forgatása)
- egyébként x-et a **könnyebb oldal felé** forgatjuk

Ezután **frissítjük** a kiegészítő információkat.

H magas, kiegyensúlyozott fában lévő csúcsok minimális és maximális száma:

H	MIN	MAX
1	1	2^0
2	2	$2^0 + 2^1$
h	$min(h-1) + min(h-2) + 1$	$\sum_{i=1}^{h} 2^{i-1}$

2. módszer - B-fa: Nem forgatásokkal érjük el a kiegyensúlyozottságot, hanem azzal, hogy **egy csúcsban egyszerre több elemet is tárolunk** (tehát NEM BINÁRIS!). A csúcsban tárolt kulcsok mindig a $<$ **reláció** szerint rendezettek.

A szakirodalomban a B-fáknál kétféle definícióval is találkozhatunk. Egy B-fa definíciójában bevezetett t változó mindig korlátozást jelöl. Ezt a változót kétféleképpen is megadhatjuk:

- ha t-rangú B-fáról beszélünk, akkor t az egy csúcsban tárolható kulcsok számára ad korlátokat. Ekkor egy belső csúcs kulcsainak száma nagyobb-egyenlő kell legyen t-nél és kisebb-egyenlő kell legyen $2t$-nél
- ha t-rendű B-fáról beszélünk, akkor t az egy csúcsból leszármazott gyermekek számára ad korlátokat. Ekkor egy belső csúcs kulcsainak száma nagyobb-egyenlő kell legyen $t-1$-nél és kisebb-egyenlő kell legyen $2t-1$-nél.

Hogy lehet, hogy mindkét definíció ugyan olyan effektív fát hoz létre? A válasz abban rejlik, hogy a B-fáknál mindig igaz lesz, hogy bármely nem levél csúcsnak maximum egyel több leszármazottja lehet a benne tárolt kulcsok számához képest. Ebből adódóan, ha a gyermekek számát korlátozzuk, abból pontosan meg tudjuk állapítani a kulcsokra vonatkozó korlátokat.

A gyakorlaton **m-rendű B-fákról** fogunk beszélni és bevezetjük a $\mathbf{t}=\lceil \mathbf{m/2} \rceil$ (t=ceil(m/2)) egyenlősséget. A B-fákra az alábbi tulajdonságoknak kell teljesülnie:

- A fa **gyökeré csúsára** igaz, hogy $0 \leq kulcs \leq m-1$
- Minden **gyökértől különböző csúcsra** igaz, hogy $t-1 \leq kulcs \leq m-1$
- Minden nemlevél csúcsnak maximum egyel több leszármazottja lehet a benne tárolt kulcsok számához képest
- Minden levélnek azonos a mélysége, tehát minden $x \in F$ levélpontra $d(x) = h(F)$

Megjegyzés: A t érték a programozó által megadott konstans. Egy x csúcs **rangja egyenlő az x-ben tárolt kulcsok számával**. Az x csúcsban levő **kulcsok meghatározzák**, x **gyermekeinek intervallumait**.

KERESÉS: nagyon hasonlóan működik, mint a korábban nézett bináris fáknál. A lényeges különbség, hogy egy-egy csúcsba belépve balról jobbra meg kell vizsgálnunk az elemeket. Tehát elindulunk egy adott csúcstól (nevezzük ezt x-nek) és elkezdünk keresni egy k elemet. Balról jobbra elkezdjük vizsgálni x levél kulcsait. Ha olyan elemet találunk, ami $\geq k$, vagy elérjük az adott csúcs legjobboldalibb elemét, akkor megállunk és jön egy eldöntendő kérdés. Ha az az elem, aminél megálltunk a keresett elem volt, akkor boldogok vagyunk és visszatérünk vele. Ha az az elem, aminél megálltunk egy levél csúcsban volt, akkor szomorúak vagyunk és visszatérünk azzal, hogy nem találtuk meg k-t. Ha az előző kettő közül egyik se volt igaz, akkor pedig folytassuk a keresést a jobb fiúban ha nagyobbat keresünk, mint a jelenleg talált elem, vagy a bal fiúban, ha kisebbett.

```
KERESÉS(x, k) {
  i=0
  while i < length(x.kulcsok) and k >= x.kulcsok[i] {
     i = i+1
  }
  if (k = x.kulcsok[i]) { return (x,i) } // az x csúcs i-edik kulcsát kerestük
  if (x.level) { return nil }    // a B-fa nem tartalmazza k-t
  if (x.kulcsok[i] < k) {  // a megfelelő ágban keresünk tovább
     return  B-FÁBANKERES(x.kulcsok[i].jobb_fiu, k)
  } else {
     return  B-FÁBANKERES(x.kulcsok[i].bal_fiu, k)
  }
}
```

BESZÚRÁS: Az első lépés, hogy bináris keresőfához hasonlóan, **keressük meg a beszúrandó csúcs helyét**, majd szúrjuk be. Ezután ellenőrizzük, hogy az új fa, nem séri-e a B-fa tulajdonságait. **Ha a beszúrás után sérülnek a B-fa tulajdonságok, akkor módosított csúcson hajtsunk végre „szétvágás" műveletet, majd ellenőrizzük a tulajdonságokat tovább a gyökér felé.**
Alapvetően a javítás, azaz a szétvágás művelet során tovább akarjuk terebélyesíteni a fát. Miért? Mert új elemet szúrtunk be, ha pedig új elem van, akkor nő a fa szerkezete, ha pedig nő, akkor terebélyesedik a fa. Alapvetően széltében és mélységében tudnánk növelni, azonban az kulcsok száma és így az egy csúcsból induló gyerekek száma is le van korlátozva, ahogy a B-fa tulajdonságainál megbeszéltük. **Mivel széltében korlátozva vagyunk, ezért az a lehetőségünk maradt, hogy mélységében dolgozzunk a javítás során a szétvágás művelettel.**

SZÉTVÁG: 2t-1 méretű csúcsba szúrtunk be, mikor sértettük a tulajdonságot, így a **beszúrás után** biztos, hogy, **2t** méretű csúcsot kapunk. Ebben az esetben úgy kell új csúcsokat kialakítani, hogy **figyelünk az intervallum tulajdonságára** a fának, tehát arra, hogy ami balra ágazik le, az kisebb, ami jobbra az pedig nagyobb legyen. **Vágjuk el** az új $2t$ méretű csúcsunkat **a "középső„ elem mentén 3 részre** úgy, hogy a középső elem maradjon egyedül, a tőle jobbra lévők továbbra is a jobb oldalon és a tőle balra lévők bal oldalon legyenek. Így kapunk egy 1 méretű, egy t méretű és egy $t-1$ méretű csúcsot. A középső, **1 méretű csúcsot felküldjük az ősbe** és fiaiként bekötjük a korábban tőle jobbra és balra lévő csoportokat. Ezzel megtartottuk az intervallum tulajdonságot.

TÖRLÉS: Keressük meg a törlendő csúcsot, majd töröljük. **Ha nem levélben lévő kulcsot töröltünk, akkor helyettesítjük a megelőzőjével.** Ezután a tényleges törlés helyétől (törlendő vagy az ő megelőzője) ellenőrizzük egészen a gyökérig, hogy az új fa, nem sérti-e a B-fa tulajdonságait.
Egy kulcs törlése akkor okoz sérülést, ha egy $t-1$ méretű csúcsból **t-2 méretű** lesz. A sérülés javítása során tömöríteni akarjuk a fát, erre pedig mind széltébe, mind magasságában van lehetőségünk. Ebben az esetben **a sérült csúcson hajtsunk végre "javít„ műveletet, majd ellenőrizzük a tulajdonságokat tovább a gyökér felé.**

JAVÍT: A javításnak két lehetséges formája van, melyek közül ha elvégezhető, akkor **a kölcsönzést preferáljuk**:

- **Szomszédtól kölcsönzünk** (alap esetben a bal szomszédtól, egyébként a jobbtól): ha a szomszédnak van felesleges kulcsa (tehát ha a szomszéd mérete >**t-1**), akkor tudunk kölcsönözni. A kölcsönzés menete: a bal/jobb oldali szomszédtól kölcsönvesszük a legjobboldalibb/legbaloldalibb kulcsot és **felküldjük az apja helyére, az apját pedig levisszük a feltöltendő csúcsba**. Így megőrizhetjük az intervallum tulajdonságot.
- **Összeolvasztunk** (alap esetben a bal szomszéddal, egyébként a jobbal): ilyenkor nem tudtunk kölcsönvenni, tehát a szomszédaink biztos, hogy $t-1$ méretűek, a sérült csúcs pedig **t-2** méretű. Olvasszuk össze a sérült csúcsot a bal/jobb szomszédjával. Ha csak ennyit csinálnánk, akkor sérülne az intervallum tulajdonság, ezért olvasszuk be a sérült csúcs és a szomszéd **közös ősét is**. Az így kapott csúcs pontosan **(t-2)+(t-1)+1=2t-2** méretű lesz, miközben megőriztük az intervallum tulajdonságot.

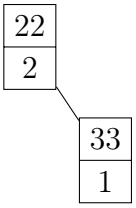
Egy n kulcsú B-fa magassága legalább: $1 + log_t((n+1)/2)$
Egy t-rendű B-fa i-edik (i $>$ 1) szintjén a csúcsok és az értékek minimális száma:

szint	csúcs	kulcs
i	$2(t+1)^{i-2}$	$2t(t+1)^{i-2}$

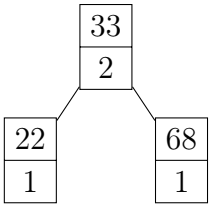
FELADATOK

Szúrjuk be egy kezdetben üres AVL fába a 22, 33, 68, 98, 91, 44, 11, 8, 26, 59, 89, 92 kulcsokat.

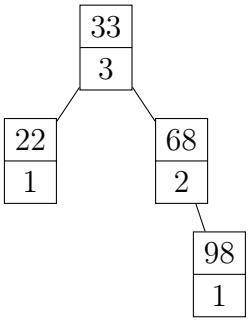
(a) 22



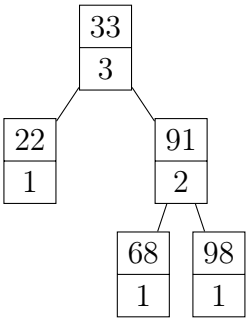
(b) 33 beszúr



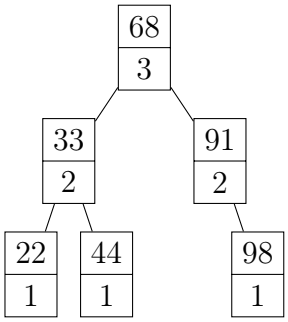
(c) 68 beszúr és forgat



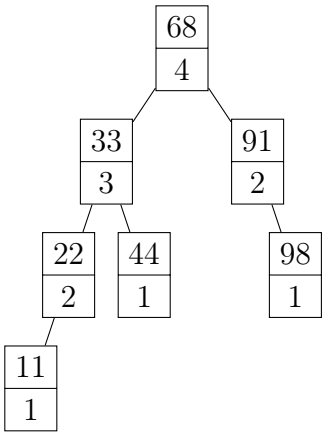
(d) 98 beszúr



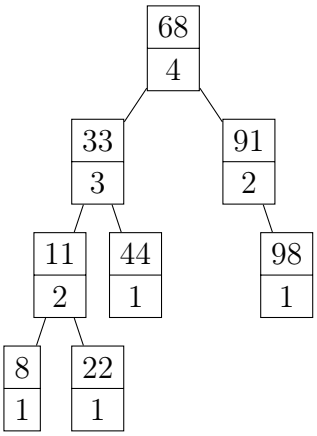
(e) 91 beszúr és cikk-cakk forgat



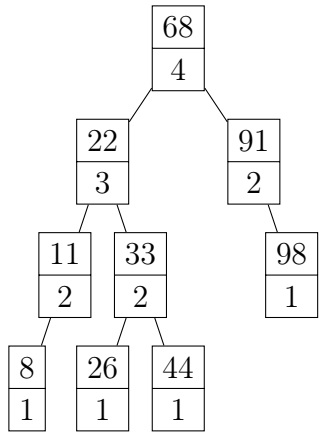
(f) 44 beszúr és cikk-cakk forgat



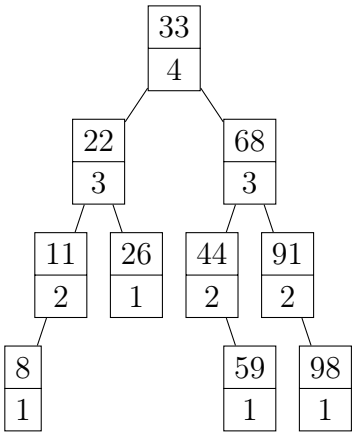
(g) 11 beszúr



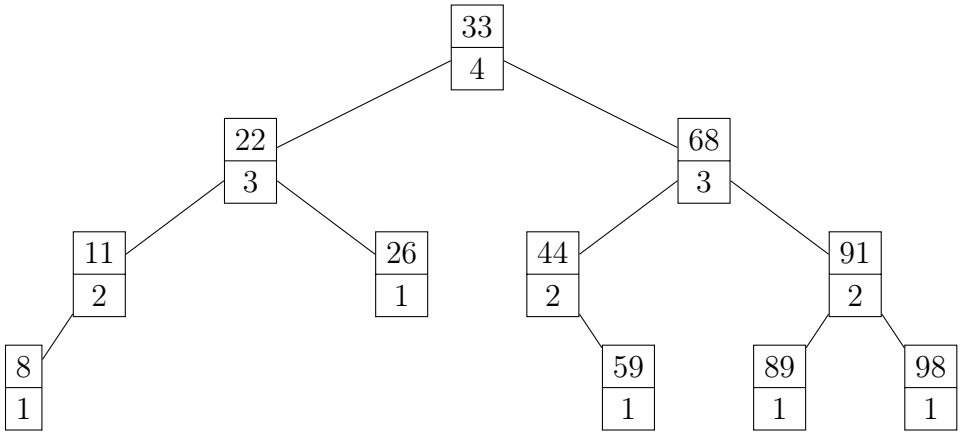
(h) 8 beszúr és forgat



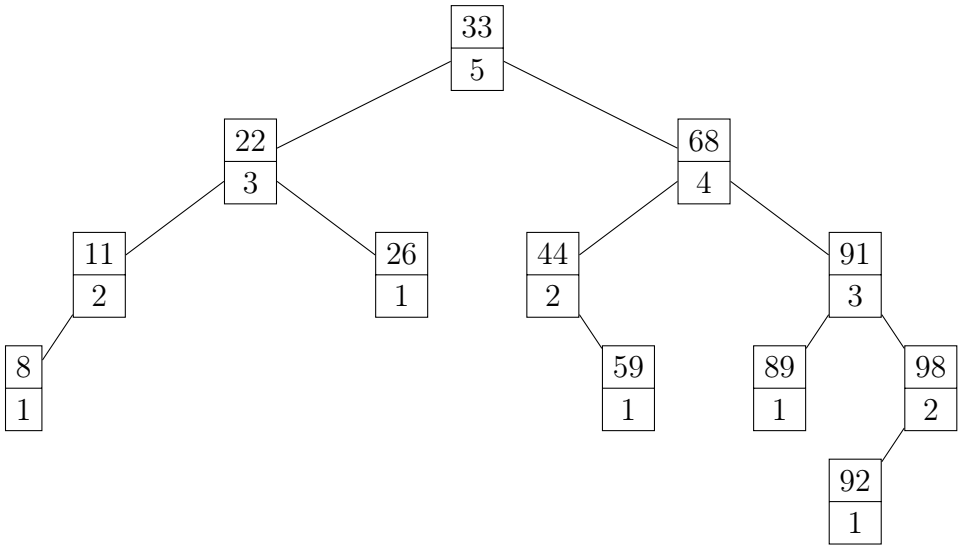
(i) 26 beszúr és cikk-cakk forgat



(j) 59 beszúr és cikk-cakk forgat

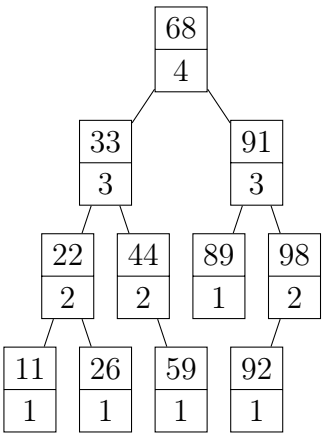


(k) 89 beszúr

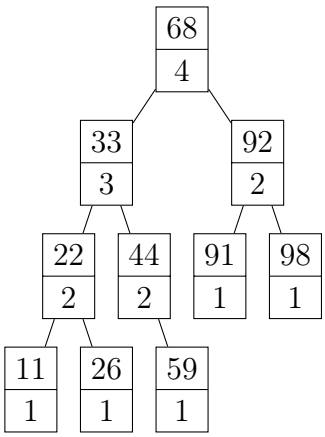


(l) 92 beszúr

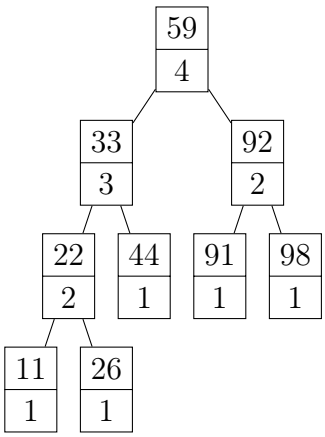
Töröljük az előzőleg kapott fából a 8, 89, 68 kulcsokat! A törléseket követően tudunk-e úgy törölni az AVL-fából, hogy ne legyen szükség forgatásos helyreállításra?



(m) 8 törlése után



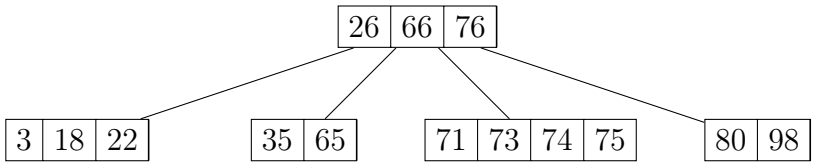
(n) 89 törlése után



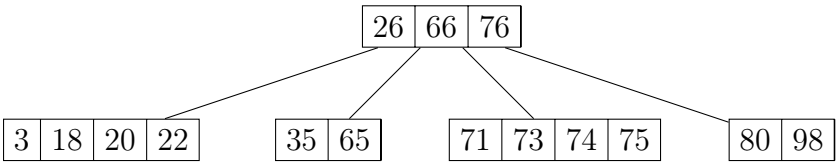
(o) 68 törlése után

Vegyük az alábbi $m = 5$ rendű B-fát, és szúrjuk be egymás után a 20, 25, 72, 27, 28, 29, 30 kulcsokat!

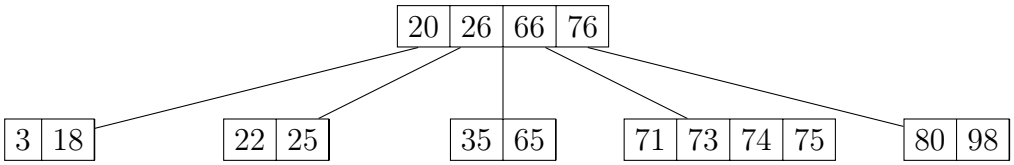
Ebben az esetben az egy csúcsban tárolható kulcsok korlátja $2 \leq kulcs \leq 4$.



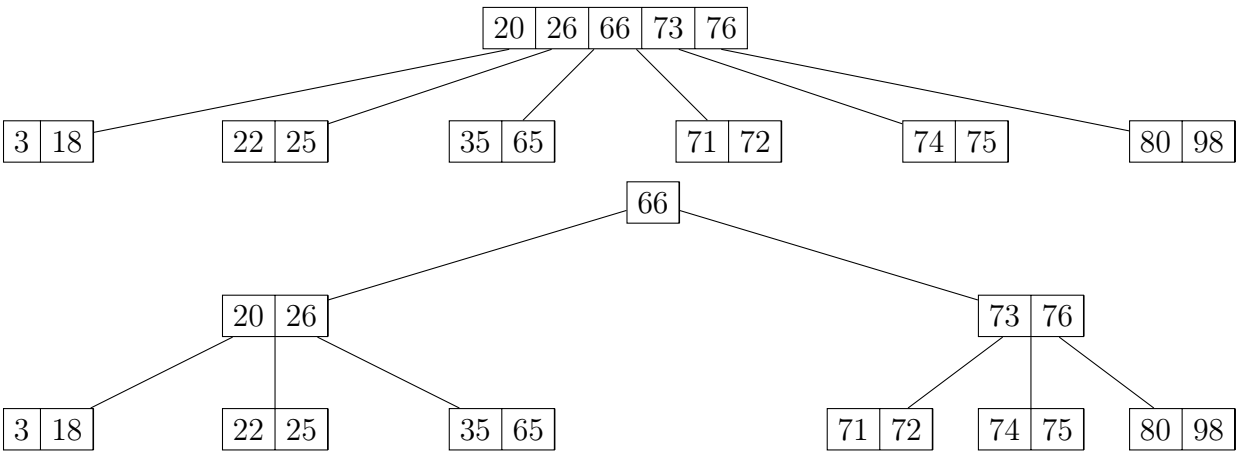
20 beszúrása után:



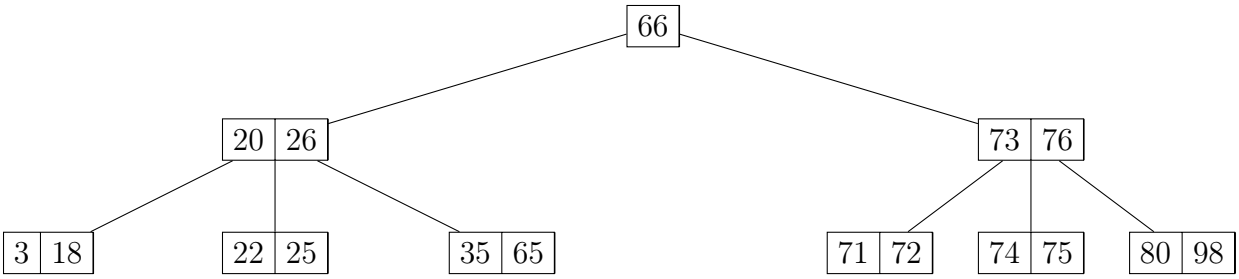
25 beszúrása után:



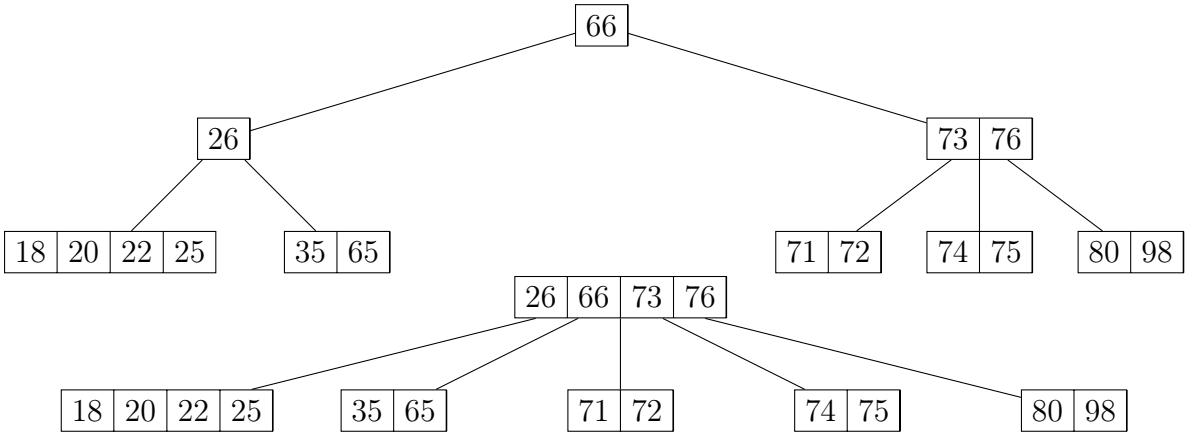
72 beszúrása után:



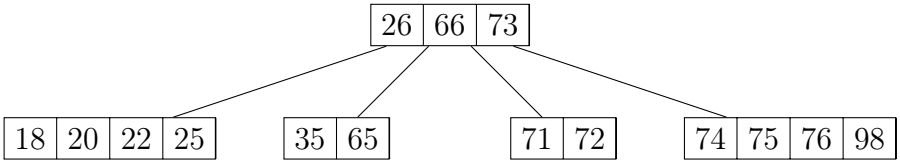
Töröljük az alábbi $m = 5$ rendű B-fából a 3, 80, 35, 76 kulcsokat! Ebben az esetben az egy csúcsban tárolható kulcsok korlátja $2 \leq kulcs \leq 4$.



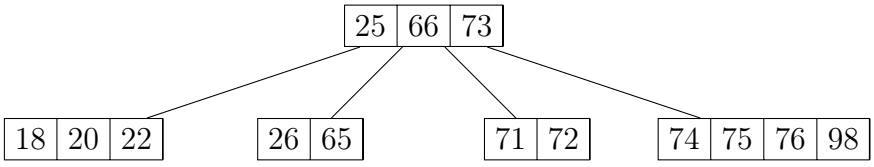
3 törlése után:



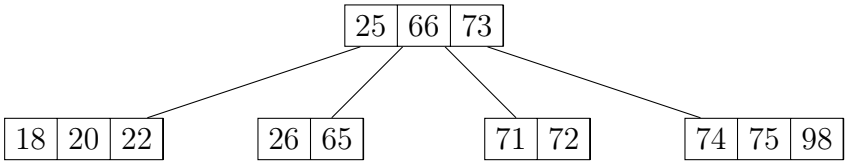
80 törlése után:



35 törlése után:



76 törlése után:



3. gyakorlat – Piros-fekete fa beszúrás

FOGALMAK

Piros-fekete fa: Olyan **bináris keresőfa**, melynek minden csúcsa egy **kiegészítő információt** tartalmaz, ami nem más, mint a csúcs színe (**piros vagy fekete**). A piros-fekete fa színezésének korlátozásával biztosítható, hogy **a fa leghosszabb levélig vezető útja maximum kétszer olyan nagy, mint a legrövidebb ilyen út**, így kiegyensúlyozott marad. Ez nagyon hasonló ahhoz, mint amikor az AVL-fáknál maximum 2 szintnyi különbségről beszéltünk a levelek között. Az AVL-fához képest itt kisebb az átlagos műveletigény beszúrás esetén (AVL-fánál a helyreállítást forgatássokkal végeztük. Itt a fa tulajdonságait színezéssel is helyre tudjuk állítani, ami a forgatáshoz képest, csak egy bit átállítását jelenti).

A piros-fekete fákra az alábbi tulajdonságok teljesülnek:

- Minden csúcs színe piros vagy fekete
- A gyökér csúcs színe fekete
- Minden NULL csúcs színe fekete
- Minden piros csúcsnak mindkét gyereke fekete kell legyen
- Bármely csúcsból kiindulva, minden levélig vezető úton ugyanannyi a fekete csúcs

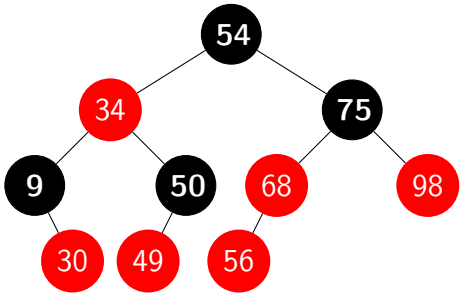
KERESÉS: Bináris keresőfához hasonlóan.

BESZÚRÁS: A bináris keresőfákhoz hasonlóan először keressük meg az új csúcs helyét, majd szúrjuk be. Beszúrés után először az **új csúcs színét állítsuk pirosra**. A beszúrás művelet módosítja a fa szerkezetét, így előfordulhat, hogy a beszúrás után az új fa sérti a piros-fekete fa tulajdonságok valamelyikét. Ezért minden beszúrás után **ellenőrizzük** a beszúrás helyétől **egészen a gyökérig**, hogy kell-e korrigálnunk a fán, és ha igen, akkor hajtsuk végre a „JAVÍTÁS" műveletet.
Melyik tulajdonság sérülhetett?
a) **A gyökér színe fekete**: ha legelőször szúrtunk be csúcsot gyökérelemnek.
b) **Minden piros csúcsnak mindkét gyereke fekete**: ha piros csúcs alá szúrtunk be.
JAVÍTÁS:
Legyen a beszúrt csúcs x:
a) Ha x **nagybátyja** (x apjának testvére, azaz $x.apa.apa$ másik fia) **piros**: **Színezzük újra** a csúcsokat. Így x maradjon piros, $x.apa$ legyen fekete, $x.apa.apa$ legyen piros, x nagybátyja legyen fekete. Majd folytassuk rekurzívan az ellenőrzést x nagyapjától.
b) Ha x **nagybátyja fekete**, **x.apa bal fiú** valamint **x jobb fiú**: Hajtsunk végre **balra forgatás**t $x.apa$ körül. Majd folytassuk rekurzívan az ellenőrzést x régi apjától.
(Megjegyzés: ennek az inverze, ha x nagybátyja fekete, x.apa jobb fiú és x bal fiú, ekkor balra forgatás helyett jobbra forgatást csinálunk).
c) Ha x **nagybátyja fekete**, **x.apa bal fiú** valamint **x bal fiú**: Először **színezzünk át**. Így **x.apa** legyen **fekete**, **x.apa.apa** legyen **piros**. Ezután hajtsunk végre **jobbra forgatást** $x.apa.apa$ körül. Majd folytassuk rekurzívan az ellenőrzést x régi apjától.
(Megjegyzés: ennek az inverze, ha x nagybátyja fekete, x.apa jobb fiú és x jobb fiú, ekkor jobbra forgatás helyett balra forgatást csinálunk.)
Bármelyik javítás végrehajtása után, folytonosan ellenőrizzük a piros-fekete tulajdonságok teljesülését egészen a gyökérig, és ha valahol újra sérül, ott ismét javítsunk.

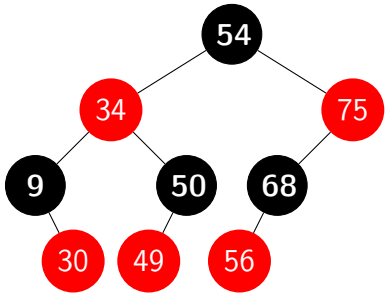
Megjegyzés: a **b) eset javítása, mindig a c) esetbe forgatja át a részfát**, így a javítás gépi implementációjánál nem kell két külön if ág a két esetnek.

FELADATOK

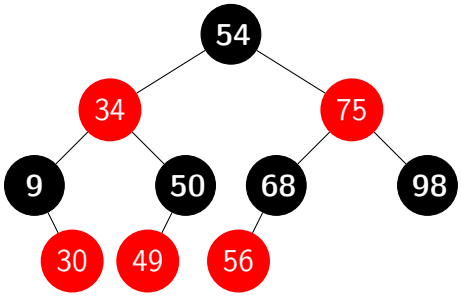
Az alábbi fák közül melyekre teljesül a piros-fekete fákkal kapcsolatos összes elvárás?



(a) Nem piros-fekete fa (piros gyerek, piros szülő)

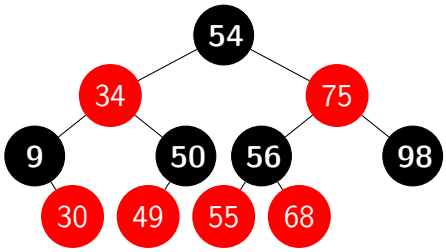


(b) Nem piros-fekete fa (levélig érintett fekete csúcsok száma)

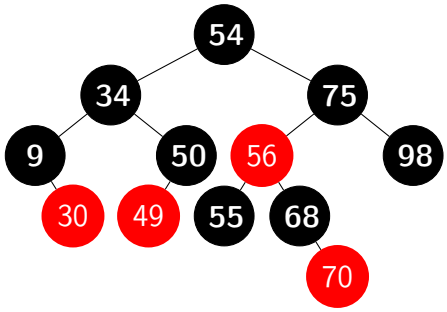


(c) Piros-fekete fa

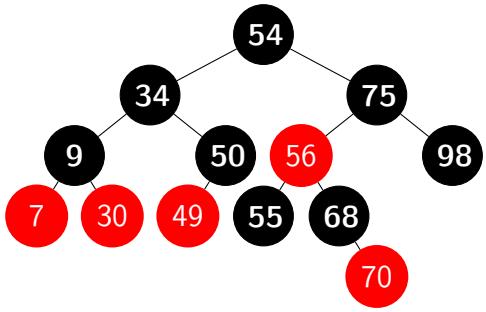
Szúrjuk be az 55, 70, 7, 5, 69, 73 kulcsokat az előző feladat (c) jelű piros-fekete fájába.



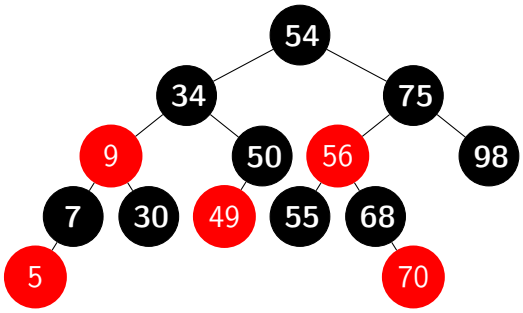
(a) fekete nagybácsi → forgatás



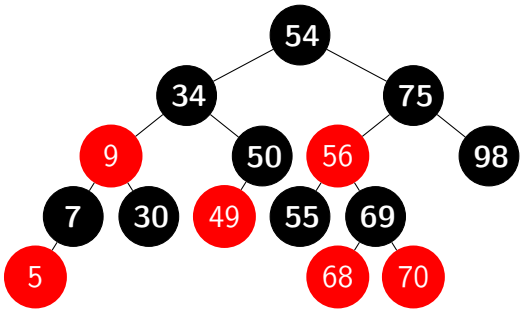
(b) piros nagybácsi → átszínezés(ek)



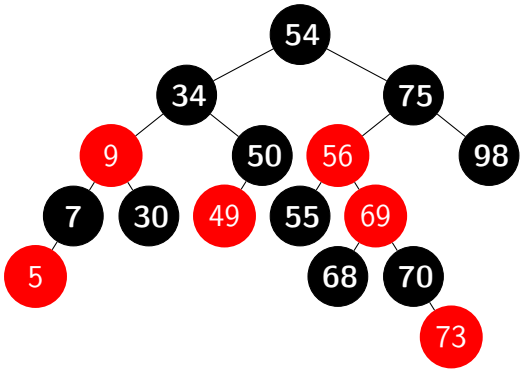
(c) fekete ős → nincs teendő



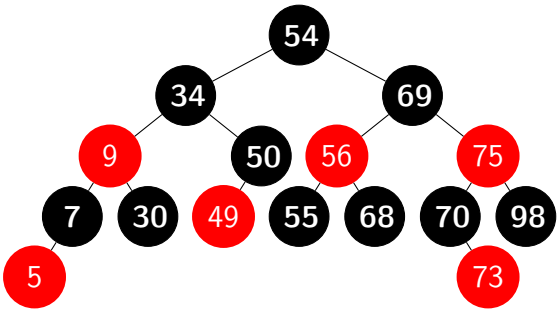
(d) piros nagybácsi → átszínezés



(e) fekete nagybácsi és zikk-zakk → 2 forgatás



(f) piros nagybácsi → kaszkád színezés



(g) fekete nagybácsi és zikk-zakk → 2 forgatás

4. gyakorlat – Piros-fekete fa törlés

FOGALMAK

TÖRLÉS: A bináris keresőfákhoz hasonlóan töröljük a csúcsot (ha kell, helyettesítsünk a megelőzőjével). A törlés művelet módosítja a fa szerkezetét, így előfordulhat, hogy a törlés után az új fa sérti a piros-fekete fa tulajdonságok valamelyikét. Ha a törölt csúcs piros volt, akkor a piros-fekete fa tulajdonságok nem sérülnek. Ha viszont fekete volt, akkor a törlés helyétől egyészen a gyökérig ellenőrizzük, hogy kell-e korrigálnunk a fán, és ha igen, akkor hajtsuk végre a „JAVÍTÁS" műveletet.
Melyik tulajdonság sérülhetett?
a) A gyökér színe fekete: ha a törölt elem gyökér volt, egyetlen piros gyerekkel, így piros csúccsal helyettesítjük.
b) Minden piros csúcsnak mindkét gyereke fekete: ha a törölt csúcs szülője piros volt és egy piros csúccsal helyettesítettük őt.
c) Bármely csúcsból bármelyik levélig vezető úton ugyanannyi fekete csúcs van: egy fekete csúcs törlése után eggyel kevesebb fekete marad az úton.

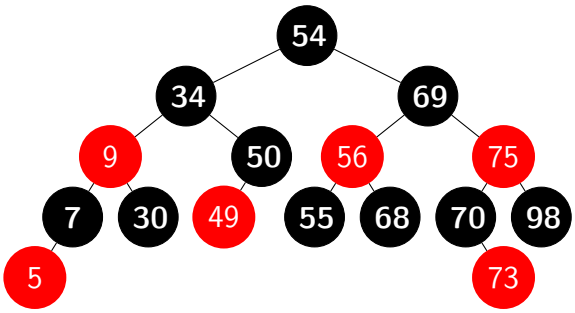
JAVÍTÁS: Legyen a törölt csúcs x:
a) Ha x bal fiú és x testvére ($x.apa.jobb$) piros: Legyen $x.apa.jobb$ színe fekete és $x.apa$ színe piros. Ezután forgassunk balra $x.apa$ körül. Az x mutatója nem változik.
(Inverz: x jobb fiú-x.apa.bal piros: balra forgatás helyett jobbra forgatást csinálunk.)
b) Ha x testvére fekete, és a testvér mindkét fia fekete: Ekkor x testvére legyen piros, x apja legyen fekete. Az x mutatóját állítsuk, legyen az új x az $x.apa$.
c) Ha x bal fiú, x testvére fekete, és a testvér bal fia piros, jobb fia fekete: Először $x.apa.jobb$ színe legyen piros és $x.apa.jobb.bal$ színe legyen fekete. Ezután $x.apa.jobb$ körül hajtsunk végre jobbra forgatást. Az x mutatója nem változik.
(Inverz: x jobb fiú, testvér F, testvér.jobb P testvér.bal F: jobb helyett balra forgatunk)
d) Ha x bal fiú, x testvére fekete, és a testvér jobb fia piros: Ekkor állítsuk $x.apa.jobb$ színét az $x.apa$ színére, az $x.apa$ színét feketére és $x.apa.jobb.jobb$ színét feketére. Ezután $x.apa$ körül forgassunk balra. Az x mutatóját állítsuk, legyen az új x a fa gyökere.
(Inverz: x jobb fiú, x testvére F, testvér bal fia P: bal helyett jobbra forgatunk)
A javítás lokális végrehajtása után addig ismételjük a JAVÍTÁS műveletet, amíg el nem érünk a gyökérig és az aktuális x csúcs fekete. Ha az aktuális x csúcs piros vagy gyökér elem, akkor megállunk és feketére színezzük az aktuális csúcsot.

Megjegyzés: az a) eset javítása, mindig a b/c/d esetek valamelyikébe forgatja át a részfát, valamint a c) eset javítása, mindig a d) esetre módosítja a részfát, így a javítás gépi implementációjánál nem kell két külön if ág minden esetnek.

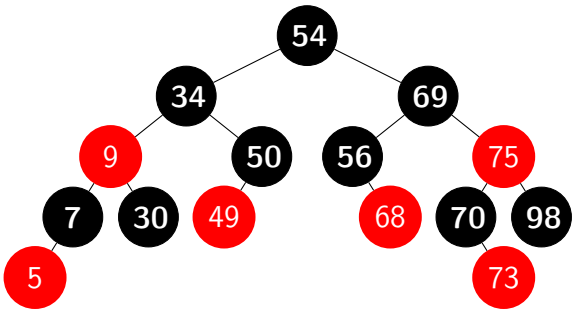
B-fa és piros-fekete fák kapcsolata: A **2/3/4-fa egy t=2-rendű B-fa**, így egy-egy csúcsnak 2 vagy 3 vagy 4 gyereke lehet.
A 2/3/4-fából alakítsunk piros-fekete fává: A **2 kulcsú** csúcsokból alakítsunk ki **apa fiú kapcsolatot**, úgy, hogy az **apát feketére** valamint a **fiút pirosra** színezzük. A **3 kulcsú** csúcsokból csináljuk egy bináris csúcsszerkezetet úgy, hogy a **középső fekete** elem lesz a **bal és a jobb oldali piros elemek** apja. Az **1 kulcsú** csúcs legyen **fekete**.
Piros-fekete fát alakítsunk 2/3/4-fává: A piros-fekete fa **fekete csúcsait olvasszuk össze a piros leszármazottakkal** új 2/3/4-fa csúcsokká. Az összeolvasztásokon kívül eső összeköttetések továbbra is maradjanak meg.

FELADATOK

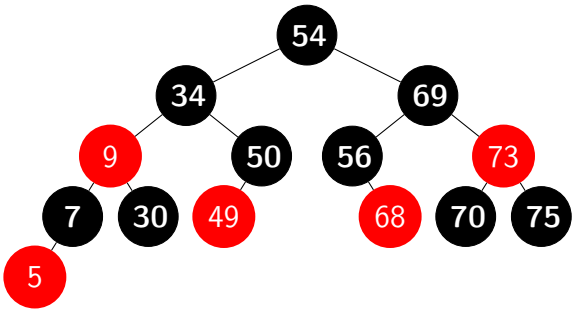
Az alábbi piros-fekete fából töröljük a 55, 98, 30, 50, 49, 7, 73, 34 kulcsokat.



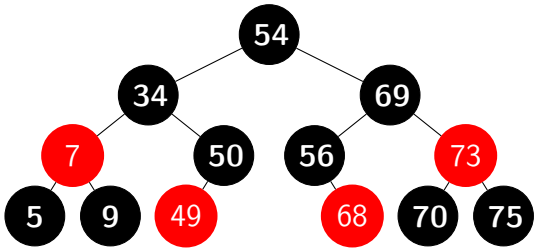
(a) eredeti fa



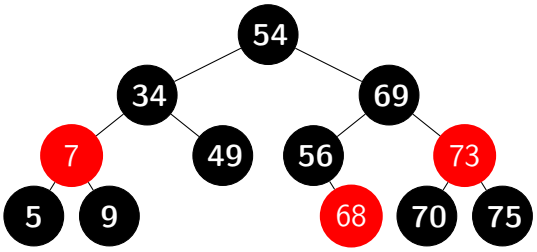
(b) fekete testvér+unokaöccsek → színezés



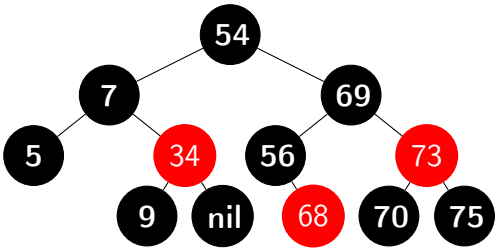
(c) fekete testvér, piros unokaöccs → forgatás



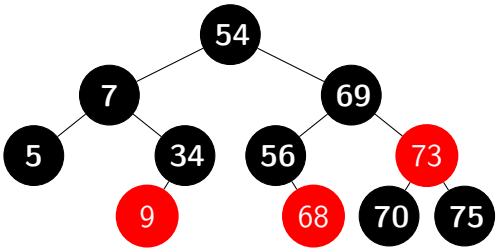
(d) fekete testvér, piros unokaöccs → forgatás



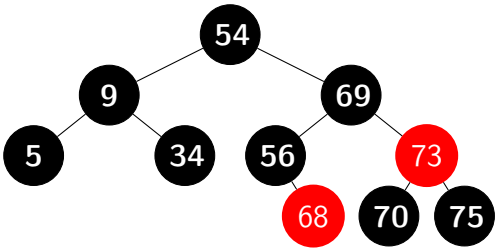
(e) egyedüli gyereke piros → nincs teendő



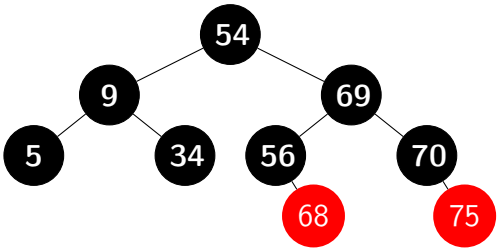
(f) piros testvér → segédforgatás



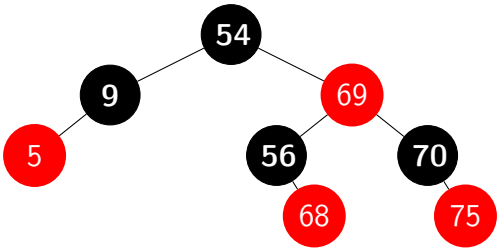
(g) fekete testvér+unokaöccsek → színezés



(h) fekete testvér+piros unokaöccs → forgatás

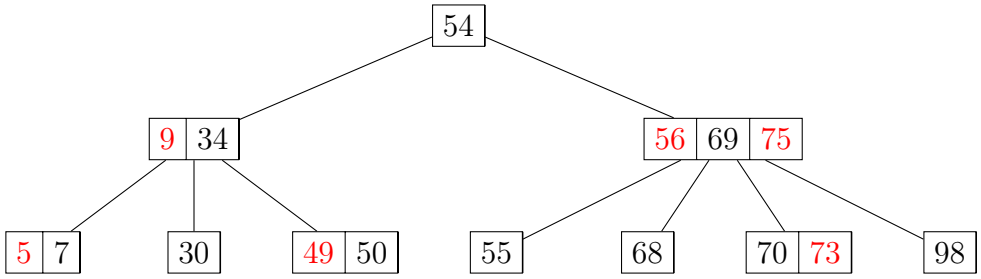


(i) fekete testvér+unokaöccsek → átszínezés

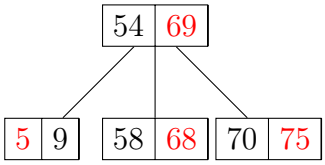


(j) fekete testvér+unokaöccsek → átszínezés

Adjuk meg az előző feladat kezdeti piros-fekete fájával ekvivalens 2-3-4 fát!



Mi lenne a törlések végrehajtását követően előálló 2-3-4 fa?



5. gyakorlat – Kibővített keresőfák

FOGALMAK

Nézzünk olyan bináris keresőfákat, amelyek egy-egy speciális feladatra kifejezetten jók. Ezek a bináris keresőfák a feladatuk tökéletes ellátása érdekében egy-egy **speciális kiegészítő információt** fognak tartalmazni. Fontos, hogy **önmagukban még nem kiegyensúlyozottak**. Ha a való életben használni szeretnénk, akkor célszerű kombinálni őket például egy AVL-fával, hogy kiegyensúlyozottak legyenek.

Intervallum-fa: Olyan bináris keresőfa, amelyben a **csúcsok értékei intervallumok**, a csúcsok **kiegészítő információi** pedig az adott gyökerű részfában található **maximális felső végpont**. Az intervallum-fákban hatékonyabban kereshetünk **átfedő intervallumokat**. Más kurzusokon tanulhattok ütemezési feladatokról, amelyekhez jó eszköz lehet egy intervallum-fa.

KERESÉS:
Átfedő intervallum keresése x gyökerű részfában. Mindig az első találattal tér vissza.

```
ÁtfedőKeres(x, i) {
  while (x != nil) {
    if (i.also <= x.felso és i.felso >= x.also) {
       return x        // átfedést találtunk
    }
    if (x.bal != nil és x.bal.kiegeszito >= i.also) {
      x = x.bal        // ha x-nek van bal fia és
                       // x bal fiának kiegészítő információja nagyobb egyenlő,
                       // mint a keresett intervallum alsó korlátja,
                       // akkor folytassuk balra a keresést
    } else {
      x = x.jobb       // folytassuk jobbra a keresést
    }
  }
  return nil           // nem találtunk átfedő intervallumot
}
```

AZ INTERVALLUM-FÁKBAN MINDIG MEGTALÁLJUK AZ ÁTFEDŐ INTERVALLUMOT, HA VAN! Miért? Tegyük fel, hogy jobbra lett volna átfedés, de balra mentünk ezért nem találtuk meg és nézzük meg milyen lépéseket követnénk végig.

BESZÚRÁS: Először keressük meg, a beszúrandó csúcs helyét. A keresés során mindig az intervallumok **bal végpontját** tekintjük kulcsnak (ha a bal végpont alapján nem tudunk dönteni, döntsünk a jobb alapján). Ezután a beszúrt csúcstól egészen a gyökérig **frissítsük** az elemek kiegészítő információját.

TÖRLÉS: Az x elem törlésekor x-et helyettesítjük a **megelőzőjével**. Ezután **frissítjük** a módosított elmeket egészen a gyökérig.

Rendezettminta-fa: olyan bináris keresőfa, amelynek minden csúcsában plusz információt is tárolunk. Az x gyökerű részfa, x-hez csatolt plusz információja az x gyökerű részfa mérete (hány elem van a részfában).
A rendezettminta-fában hatékonyabban megtalálhatjuk a $<$ reláció szerinti rendezés i-edik elemet, valamint hatékonyan megmondható egy elem rangja (azaz, hogy hanyadik legkisebb elem a $<$ reláció szerinti rendezett sorban).

KERESÉS:
Adott, i rangú kulcs keresésének pszeudokódja. Tartsunk fent egy r segédváltozót. Az r értéke minden körben legyen egyenlő a bal oldali részfa kegészítő infója +1-el. A +1-re azért van szükségünk, mert az aktuális x-et is hozzászámoljuk a kereséshez. Ha r nagyobb, mint a keresett elem sorszáma, tehát az x még a keresett elem után van a sorban, akkor menjünk balra. Ha r kisebb, mint a keresett elem sorszáma, akkor menjünk jobbra, mert biztos, hogy még x-nél is feljebb van a rendezésben. Egyébként ha r pont egyenlő i-vel, akkor megtaláltuk a keresett elemet.

```
RangKeres(x, i) {
  r = x.bal.kiegeszito + 1 //x rangjától indulunk

  if (i < r) {
      RangKeres(x.bal, i)
  } elseif (i > r) {
      RangKeres(x.jobb, i - r)
     //Megjegyzés: tudjuk, hogy balra r-1 elem van, és x az r. elem, tehát
     //tőle jobbra olyan elemek vannak, aminek a sorszáma nagyobb, mint r.
     //Emiatt az i. elem megtalálásához a jobb részfában már csak az
     //(i-r)-edik legkisebb elemet kell megkeresnünk
  } else { return x }
}
```

Adott, x kulcs rangjának meghatározása: x-től a gyökérig lépkedve azon y gyökerű részfák gyökérelemeinek rangjait összegezzük, melyekre $x.kulcs >= y.kulcs$, tehát meg kell számolnunk hány olyan csúcs van, ami kisebb x-nél.

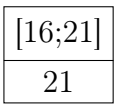
```
RangMeghatároz(x) {
  r = 0 AND y = x
  while (y != nil) {              //amíg el nem hagytuk a gyökeret
      if (x.kulcs >= y.kulcs) {        //ha x kulcsa, azaz értéke nagyobb!
          r = r + y.bal.kiegeszito + 1  //eddig megtalált + bal gyerekek + én
      }
      y = y.apa
  }
  return r
}
```

BESZÚRÁS: Az x elem beszúrásához először keressük meg x helyét, majd szúrjuk be. Ezután x-től egészen a gyökérig **frissítsük** az elemek kiegészítő információját.

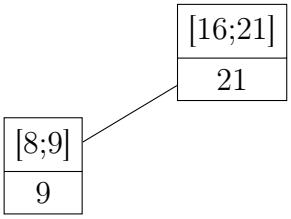
TÖRLÉS: Az x elem törlésekor x-et helyettesítjük a megelőzőjével. Ezután **frissítjük** a módosított elmeket egészen a gyökérig.

FELADATOK

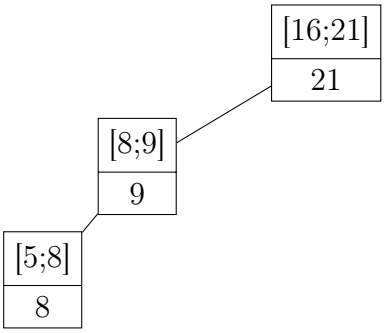
Szúrjuk be az alábbi intervallumokat egy kezdetben üres intervallum-fába:
$[16; 21], [8; 9], [5; 8], [25; 30], [15; 23], [17; 19], [26; 26], [0; 3], [6; 10]$.



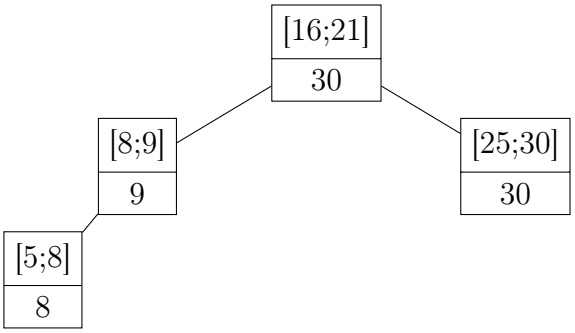
(a) 16;21 beszúrása



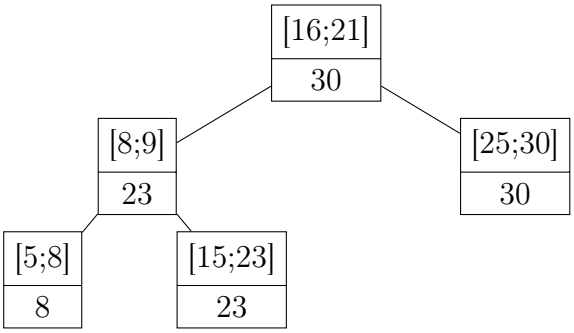
(b) 8;9 beszúrása



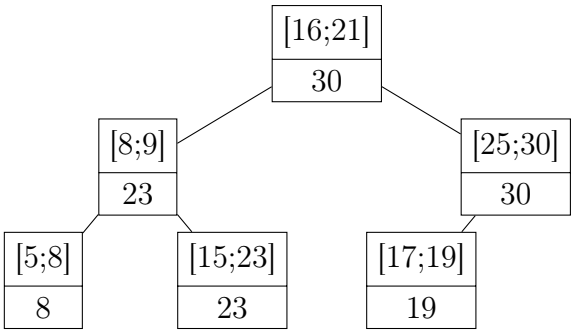
(c) 5;8 beszúrása



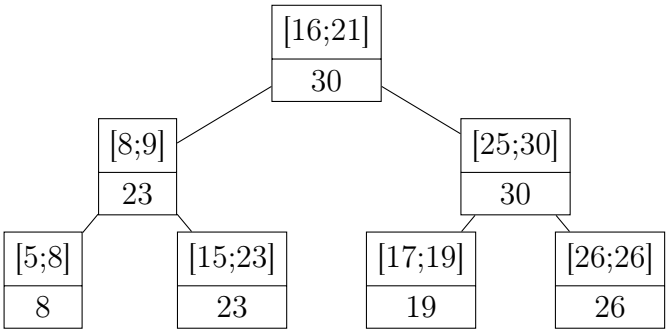
(d) 25;30 beszúrása



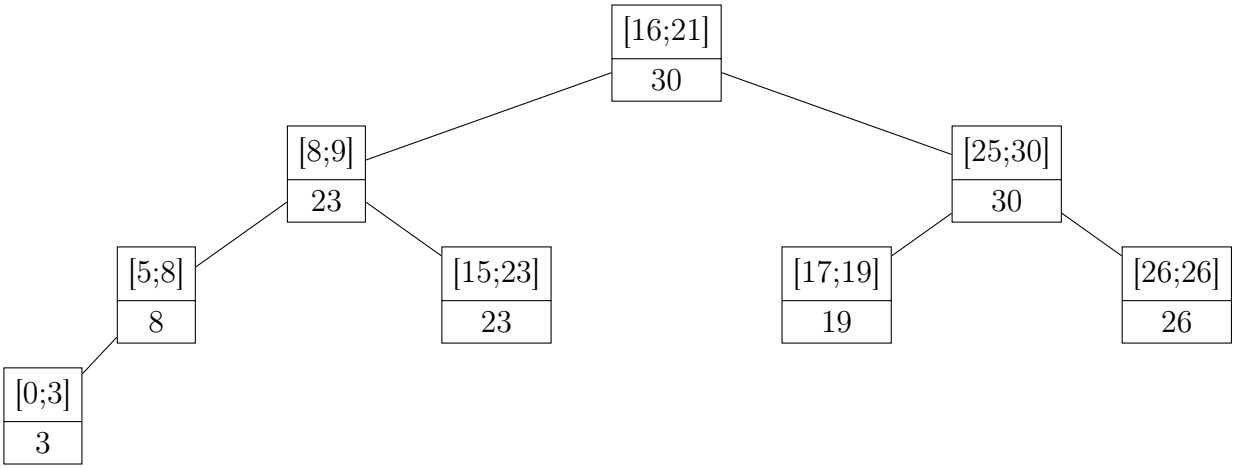
(e) 15;23 beszúrása



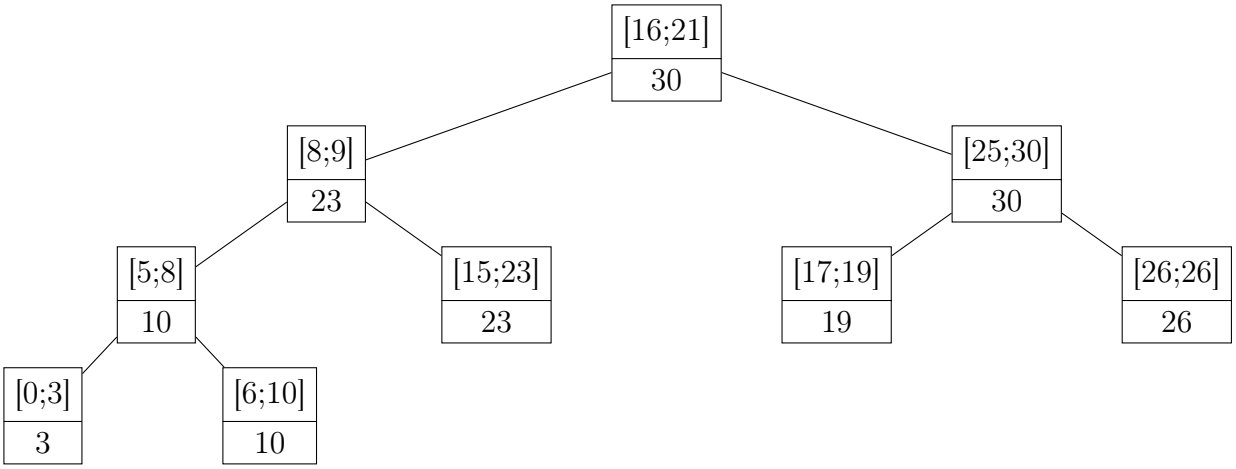
(f) 17;19 beszúrása



(g) 26;26 beszúrása



(h) 0;3 beszúrása



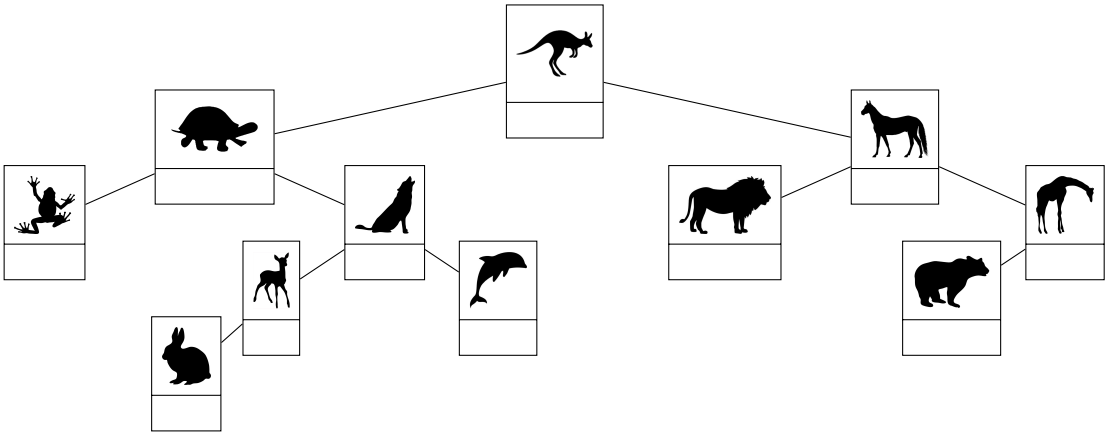
(i) 6;10 beszúrása

Keressünk átfedő intervallumot a [22;25] és a [11;14] intervallumokhoz!

ÁtfedőKeres([22;25]): $[16;21] \rightarrow [8;9] \rightarrow [\mathbf{15};\mathbf{23}] \rightarrow$ ☺

ÁtfedőKeres([11;14]): $[16;21] \rightarrow [8;9] \rightarrow [15;23] \rightarrow$ ☹

Tekintsük az alábbi bináris keresőfát rendezettminta-faként!



a) Határozzuk meg a fában lévő kulcsok $<$ reláció szerinti rendezését!

b) Töltsük ki a rendezettminta-fából hiányzó kiegészítő információkat!
Milyen fabejárással lehetne kitölteni a fából hiányzó, rendezettminta-fák által használt kiegészítő információkat? (postorder)
Megjegyzés: a valóságban persze nem "utólag", fabejárást használva határozzuk meg a kiegészítőinformációkat, hanem a műveletek végrehajtása során aktualizáljuk azokat!

c) A kiegészítő információkra támaszkodva adjuk meg a $<$ rendezés szerinti
- 6 rangú elemet

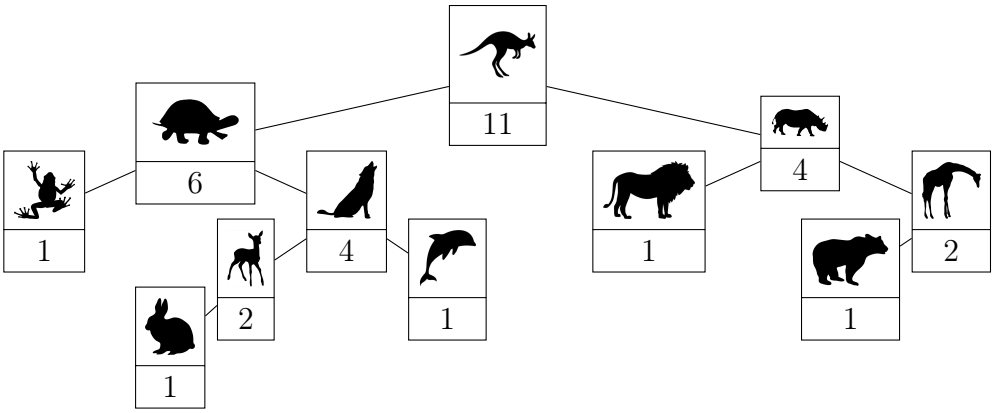
`RangKeres(`🦅`, 6)`
`RangKeres(`🐢`, 6)`
`RangKeres(`🐓`, 4)`
`RangKeres(`🦩`, 1)`

- 9 rangú elemet

`RangKeres(`🦅`, 9)`
`RangKeres(`🐎`, 2)`

d) A kiegészítő információk alapján mi lesz 🐓 rangja?
Megjegyzés: $r_x($🐓$)$ az 🐓 szimbólum rangjára vonatkozó aktuális ismereteinket jelöli abban a pillanatban, amikor az algoritmus az x jelű csúcs feldolgozásánál tart.
$r_{🐓}($🐓$) = 1 + 2$
$r_{🐢}($🐓$) = r_{🐓}($🐓$) + 1 + 1$
$r_{🦅}($🐓$) = r_{🐢}($🐓$) + 0 = 5$

e) Hajstuk végre a BESZÚR(🦏), illetve a TÖRÖL(🐎) műveleteket, amennyiben tudjuk, hogy a 🦏 $<$ 🐎, illetve a 🦏 $>$ 🦁 relációk teljesülnek!



6. gyakorlat – Bináris és binomiális kupacok

FOGALMAK

Kupactulajdonság: Azt mondjuk, hogy egy fa rendelkezik a minimum (maximum) kupactulajdonsággal, ha minden p csúcsának minden q fiára igaz, hogy $q = NULL$ VAGY $p.kulcs < q.kulcs (p.kulcs > q.kulcs)$. Tehát minimum(maximum)-kupac esetén a kupac minden gyökértől különböző **elemének értéke legalább(legfeljebb) akkora, mint a szülőjének értéke**. Így a **minimum(maximum)-kupac legkisebb(legnagyobb) eleme a gyökér**, és egy adott csúcs alatti részfa minden elemének értéke nem kisebb(nagyobb), mint az adott csúcsban lévő elem értéke.

Bináris kupac: A (bináris) kupac adatszerkezet úgyis szemlélhető, mint egy **majdnem teljes bináris fa** (tehát olyan fa, amelyben **minden nem levél csúcsnak két gyereke van**) egy tömbben ábrázolva. A fa minden csúcsa megfelel a tömb egy elemének, mely a csúcs értékét tárolja. **A nevezetes elemek így könnyen indexelhetőek.** A fa gyökere a tömb első eleme, és ha i a fa egy adott csúcsának tömbbeli indexe, akkor az ősének Szülő(i), bal oldali gyerekének Bal(i), és jobb oldali gyerekének Jobb(i) indexe egyszerűen kiszámítható: Szülő(i) $= i/2$, Bal(i) $= 2i$, Jobb(i) $= 2i + 1$.
A bináris kupacnak két fajtája van: a maximum-kupac és a minimum-kupac. Mindkét fajta kupacban a csúcsok értékei kielégítik a maximum/minimum kupactulajdonságot.

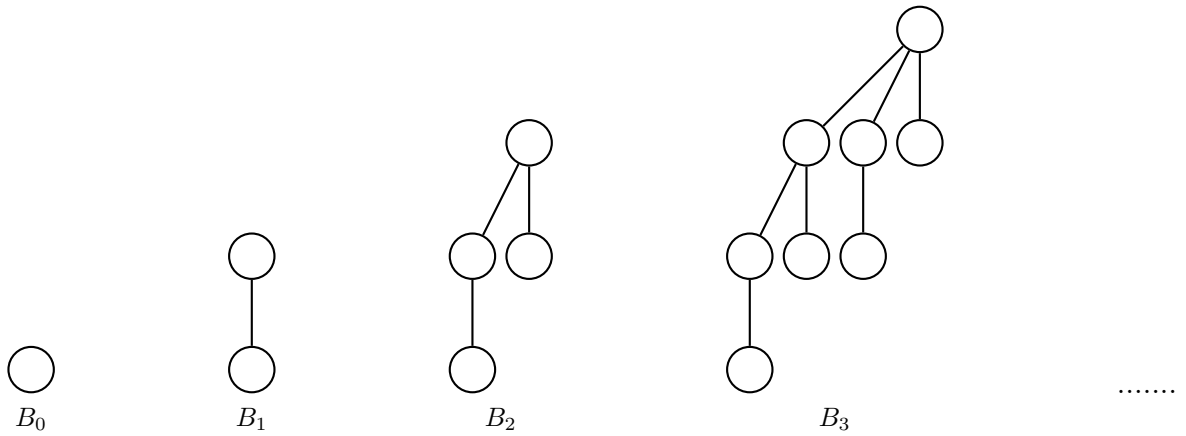
```
(művelet B maximum-kupacra, mely kiveszi a maximumot)
 SORBOL() {
    MAX = B[1]
    B[1] = B[utolso]
    kupacmeret[B] -= 1
     MAX-KUPACOL(B, 1)
    return MAX
 }

(B maximum-kupac javítása)
 MAX-KUPACOL(B, i) {     //ahol i a fa egy csúcsának tömbbeli indexe
    l = Bal(i)  //B[2i]
    r = Jobb(i) //B[2i+1]
    MAX = MAX(i, l, r)
    if(MAX != i){
        B[MAX] felcserélése B[i]-vel
    }
     MAX-KUPACOL(B, MAX)  //az új/aktuális maximum felé megyünk
 }
//tehát ellenőrizzük, hogy az adott pontban a maximum érték a szülőben van-e,
ha nem, akkor helyezzük át oda, majd hívjuk meg a MAX-KUPACOL eljárást//
```

```
(B maximum-kupac bővítése)
 SORBA(B, kulcs) {
    kupacmeret[B] += 1
    B[kupacmeret[B]] = -INF
     KUPACBAN-KULCSOT-NOVEL(B, kupacmeret[B], kulcs)
    return MAX
 }
//tehát adjunk hozzá egy -végtelen kulcsot a tömb végéhez,
majd írjukát azt a kívánt értékre//

(B maximum-kupac kulcsértékének növelése)
 KUPACBAN-KULCSOT-NOVEL(B, i, kulcsertek) {
    B[i] = kulcsertek
    while(i > Szülő(i)){
        B[Szülő(i)] felcserélése B[i]-vel
        i = Szülő(i)
    }
 }
//tehát írjuk át a kulcsot az új értékre, és a gyökérig haladva ellenőrizzük,
hogy az aktuális kulcs nagyobb-e, mint a szülő,
ha igen, akkor cseréljük fel őket//
```

Binomiális fa: Az előbb említett tömbös reprezentáció helyett használhatunk **fás reprezentációkat** is. A B_k binomiális fa egy **rekurzív módon definiált rendezett fa**. A B_k binomiális fa két összekapcsolt B_{k-1} binomiális fából áll, ahol az egyik fa gyökércsúcsa a másik pedig a fa gyökércsúcsának legbaloldalibb gyereke.



A B_k binomiális fa tulajdonságai:

- 2^k csúcsa van
- magassága k
- az i-edik mélységben pontosan $\binom{k}{i}$ csúcs van
- a gyökér fokszáma k, ami nagyobb, mint bármely másik csúcs fokszáma; továbbá, ha a gyökércsúcs gyerekeit balról jobbra haladva megszámozzuk $k-1, k-2, ..., 0$-val, akkor az i. gyerek a B_i részfa gyökércsúcsa (így fokszáma pontosan i)

Binomiális kupac: Egy H binomiális kupac, binomiális fák olyan halmaza, amely kielégíti a binomiális-kupac tulajdonságokat az alábbiakkal:

- H minden binomiális fája rendelkezik a minimum(maximum)-kupac tulajdonsággal
- egy csúcs kulcsa nagyobb(kisebb) vagy egyenlő, mint a szülőjének a kulcsa. Ekkor azt mondjuk, hogy ezek a fák min(max)-kupac-rendezett fák
- H-ban nincsenek azonos fokszámmal rendelkező binomiális fák

Minimum(maximum) keresése: keressük meg a binomiális fák csúcsai között a minimumot(maximumot).

SORBA(B, kulcs): a kapott kulcs-ból készítsünk egy 0 fokszámú binomiális fát, majd ebből egy egy elemű binomiális kupacot, legyen ez K. Végül EGYESÍT(B,K) kupacokra.

KULCS-MODOSIT(B, kulcsindex, ertek): Módosítsuk a kulcsot az új értékre, majd a gyökérig haladva ellenőrizzük, hogy az aktuális kulcs nagyobb-e(kisebb-e), mint a szülő, ha igen, akkor cseréljük fel őket.

SORBOL(): Keressük meg a kupac minimumát (maximumát). Ezután a minimumot (maximumot) tartalmazó kupac gyökerének (tehát a minimum(maximum) elemnek) fiait alkotó binomiális fákból készítsünk új kupacot, legyen ez K. Töröljük a kezdeti kupacból a minimumot (maximumot) tartalmazó fát, legyen ez H. Végül EGYESÍT(H,K) kupacokra.

EGYESÍT(H1, H2):
Első lépésként fésüljük össze H1 és H2 kupacot, azaz fokszámok szerint növekvő sorrendben rendezzük be kupacok fáit és alkossunk belőle új kupacot. Legyen ez a K kupac.
Ezután megszüntetjük az azonos fokszámú fákat az alábbi módon:
$x = K.fej$
while(van következő fa)

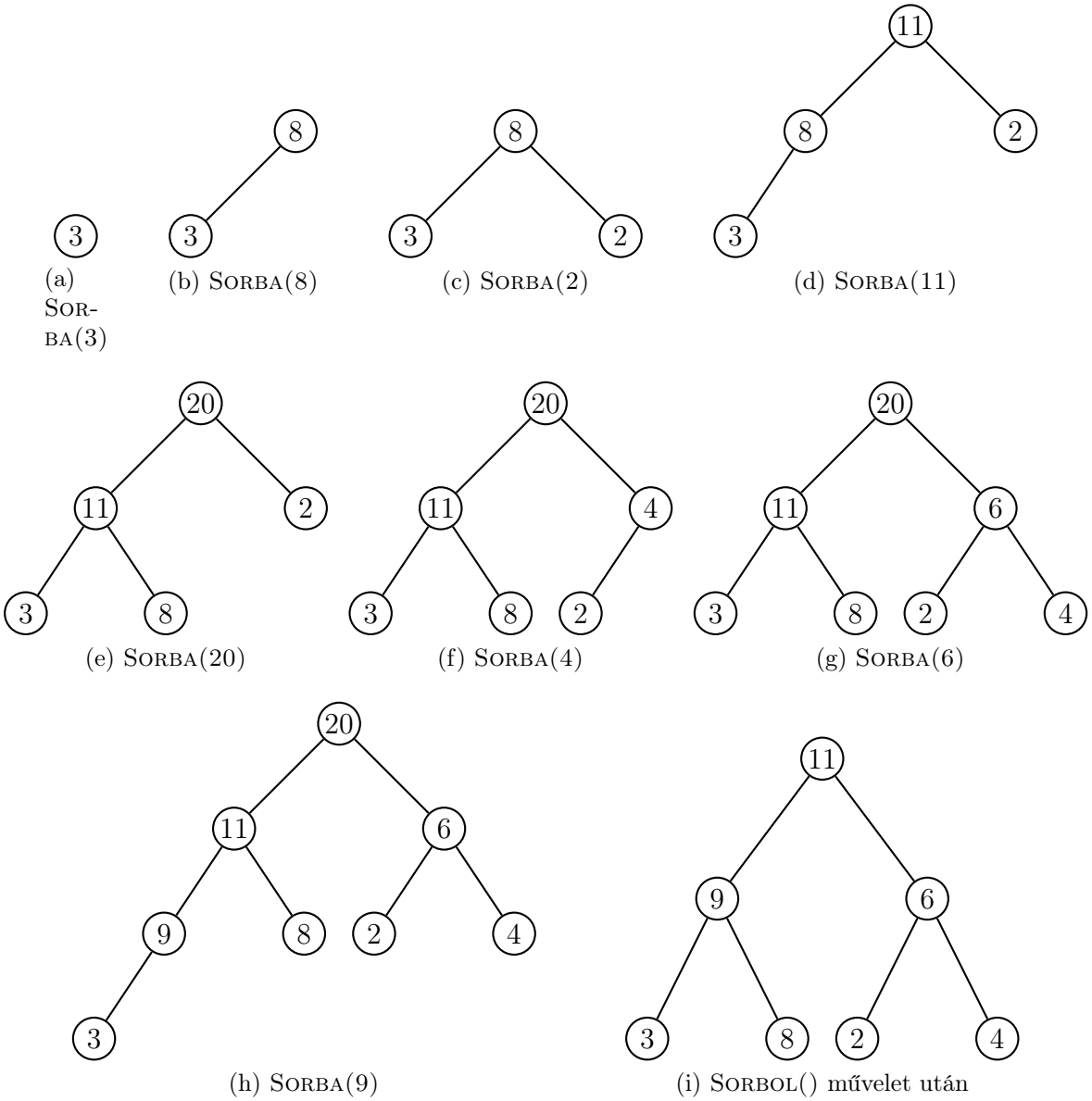
a) ha $x.fokszam \neq x.kovetkezo.fokszam$, akkor $x = x.kovetkezo$

b) ha $x.fokszam == x.kovetkezo.fokszam == x.kovetkezo.kovetkezo.fokszam$, akkor $x = x.kovetkezo$

c) ha $x.fokszam == x.kovetkezo.fokszam \neq x.kovetkezo.kovetkezo.fokszam$, akkor kössük be x és $x.kovetkezo$ csúcsokat úgy, hogy a nagyobb gyökerű csúcs új fia legyen a kisebb gyökerű csúcs, ezután $x = ujonnankialakitottfa$

FELADATOK

Hajtsuk végre a SORBA() műveletet egy üres maximum kupacon a következő elemekkel: 3, 8, 2, 11, 20, 4, 6, 9. Mi lesz a SORBOL() eredménye?

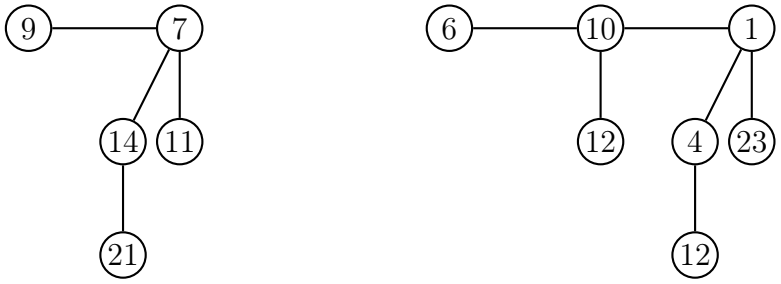


(a) SORBA(3) (b) SORBA(8) (c) SORBA(2) (d) SORBA(11)

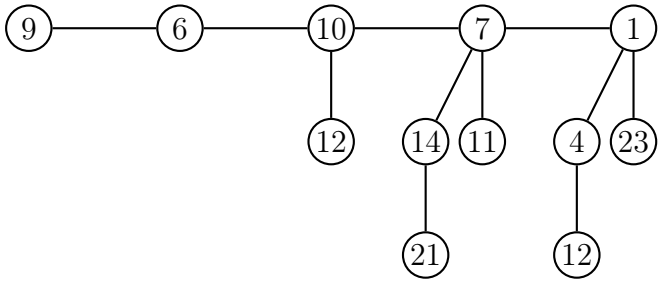
(e) SORBA(20) (f) SORBA(4) (g) SORBA(6)

(h) SORBA(9) (i) SORBOL() művelet után

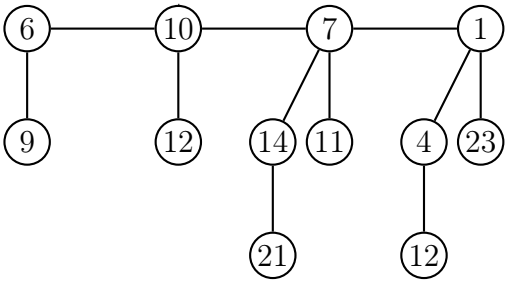
Egyesítsük az alábbi két minimális binomiális kupacot.



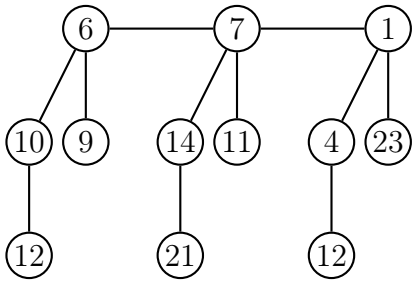
(a) (b)



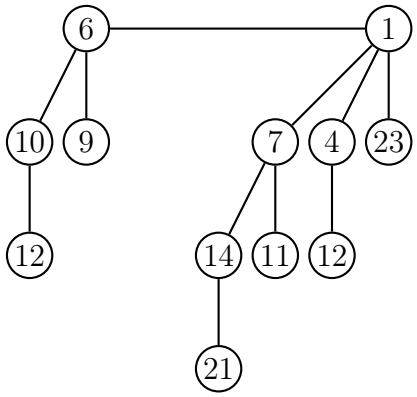
(c) Az első összefésülő lépés után előálló kupac



(d)

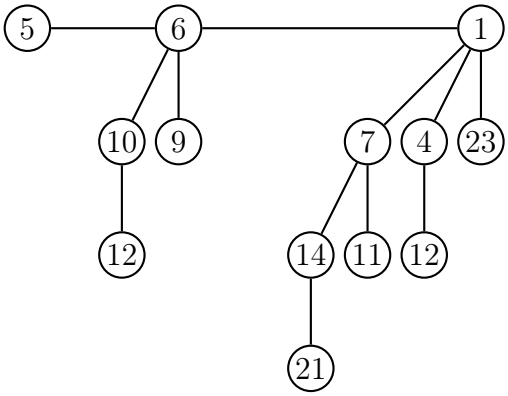


(e)

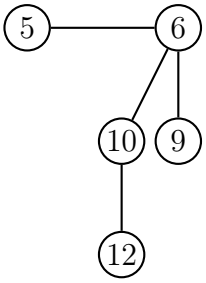


(f) Az első összefésülő lépés után előálló kupac

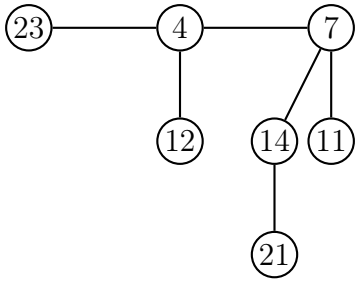
Az egyesített kupacra végezzük el a SORBA(5), SORBOL() műveleteket, végül módosítsuk a 14-es kulcsot 3-ra, illetve a 7-es kulcsot 2-re.



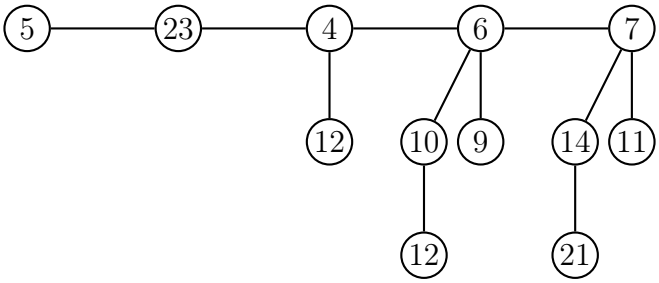
(g) SORBA(5)



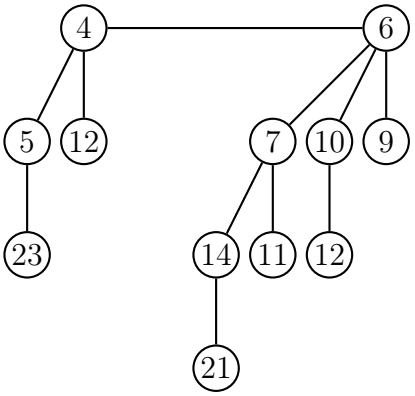
(h) SORBOL() módisítatlan része



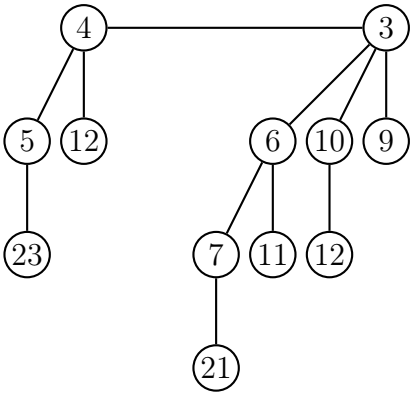
(i) SORBOL() minimum kulcs kupacainak új kupaca



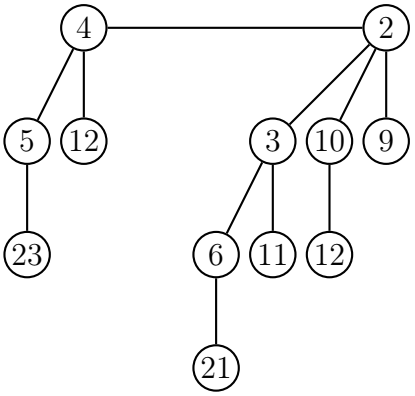
(j) SORBOL() kupacainak összefűzése



(k) EGYESIT



(l) $14 \rightarrow 3$ után



(m) $7 \rightarrow 2$ után

7. gyakorlat – Fibonacci kupacok és Amortizált költségelemzés

FOGALMAK

Fibonacci-kupac: Elméleti szempontból a Fibonacci-kupacok különösen jól alkalmazhatók, ha a SORBOL és a TÖRÖL műveleteket kevesebbszer kell végrehajtani, mint a többi műveletet. A Fibonacci-kupacok fő alkalmazásait azok a gyors algoritmusok adják, amelyek olyan problémák megoldására szolgálnak, mint a minimális feszítőfák számítása és az egy csúcsból induló legrövidebb utak megkeresése.
A binomiális kupacokhoz hasonlóan, a Fibonacci-kupac min-kupac-rendezett fák gyűjteménye, azonban a Fibonacci-kupacban levő fák nem feltétlenül binomiális fák. Egy másik eltérés, hogy a Fibonacci-kupacokban levő fáknak bár van gyökérelemük, de a fák a kupacban méretük alapján nem rendezettek. A fák sorrendje a gyökérlistában tetszőleges. A Fibonacci-kupac fáinak gyökércsúcsai bal és jobb pointerekkel vannak összekapcsolva, ezt a ciklikus kétirányú listát a Fibonacci-kupac gyökérlistájának nevezzük. Egy H kupac min[H] pointere a gyökérlista minimális kulcsú csúcsára mutat.
A Fibonacci-kupacokban a kétszeresen láncolt listák alkalmazásának két előnye is van. Az első az, hogy egy kétszeresen láncolt listából O(1) idő alatt lehet elemet törölni. A második pedig az, hogy két ilyen listát egy kétszeresen láncolt listába konkatenálni (azaz egymáshoz kapcsolni) szintén O(1) időben lehet. Egy adott Fibonacci-kupac a min[H] pointerrel címezhető meg. Ha egy H Fibonacci-kupac üres, akkor min[H] = nil.
A csúcsokban eltárolunk egy logikai [x] mezőt, ami azt jelzi, hogy az adott csúcs elvesztette-e gyerekét azóta, mióta az aktuális helyén áll. Az újonnan létrehozott csúcsok [x] mezője hamis. Egy csúcs x mezője akkor válik igazzá, amikor a csúcs elveszíti egy gyerekét. Egy csúcs x mezője akkor válik hamissá, amikor a csúcsot elmozgatjuk eddigi helyéről. Gyökércsúcs x mezője MINDIG hamis. Az x logikai érték karbantartásával felhasználásával elérhetjük, hogy ne darabolódjanak szét túlságosan a kupacban lévő fák.

Minimum keresése: adjuk vissza a min[H]-ban eltárolt elemet, így a költség O(1).

SORBA(B, kulcs): a kapott kulcsból készítsünk egy 0 fokszámú (B0) binomiális fát, majd ebből egy egy elemű binomiális kupacot, legyen ez K. Végül EGYESÍT(B,K) kupacokra.

EGYESÍT(H1, H2): Fűzzük össze a két kupac gyökérlistáját, majd aktualizáljuk a min[H] mutatót.
Figyeljünk rá, hogy egyesítés során az eredeti kupacokban lévő elemek megjelöltsége változatlan marad, tehát ne felejtsük el azokat is átvinni! Gépi megvalósításnál a H2 kupacot a H1 kupac minimumának bal oldalára szoktuk felfűzni.

KULCSOT-CSOKKENT(B, kulcs, ertek): Módosítsuk a kulcsot az új értékre, majd ellenőrizzük, hogy az új kulcs kisebb-e, mint a szülő, ha igen, akkor megsértettük a minimum kupac tulajdonságot, úgyhogy, vigyük fel a gyökérbe (a fiaival együtt). Ha felvittük, akkor az elem régi szülőjét, ha eddig nem volt megjelölve jelöljük meg, ha már meg volt jelölve, akkor vigyük fel a gyökérbe a szülőt is (a fiaival együtt) és szedjük le róla a megjelölést. Végül állítsuk be min[B] minimális elemre mutató pointert.

SORBOL(): Keressük meg a kupac minimumát. Ezután a minimumot tartalmazó kupac gyökerének (tehát a minimum elemnek) fiaiból készítsünk új kupacot, legyen ez K. Töröljük a kezdeti H kupacból a minimumot tartalmazó fát. Idáig ugyan azt csináltuk, mint amit a binomiális kupacoknál. Ezután EGYESÍT(H,K) kupacokra. Végül hajtsunk végre *karbantartást* a kupacunkon az alábbi módon:

while(van egyforma fokszámú fa a kupacban){
    A while ciklus során végig számon tartjuk, hogy az aktuális kupacban hány darabot láttunk eddig az adott fokszámú fákból.
    A kupac elemein balról jobbra haladva, mikor találunk két azonos fokszámú fát, akkor a kisebb gyökérértékű fa gyökere alá kössük be a másik fát.
    Ha meg volt jelölve a nagyobb gyökérértékű fa gyökere,
    akkor szedjük le a jelölést, mert mozgattuk.
}

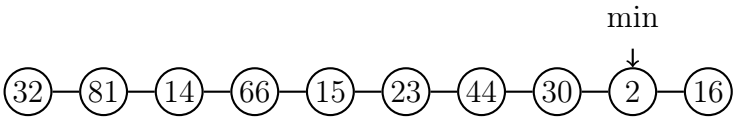
A karbantartás után állítsuk be a min[H] minimális elemre mutató pointert.

TÖRÖL (B, kulcs): Keressük meg a törlendő elemet. A törlendő elem értékét állítsuk át -INF (-végtelen) értékre. A törlendő elemet vigyük fel a gyökérbe (a fiaival együtt). A törlendő elem régi szülőjét, ha eddig nem volt megjelölve, akkor jelöljük meg (tehát az x logikai értéket állítsuk igazra), ha már meg volt jelölve (tehát x igaz volt), akkor vigyük fel a gyökérbe (a fiaival együtt) és szedjük le róla a megjelölést. Állítsuk be min[B] minimális elemre mutató pointert. Utolsó lépésben végrehajtjuk a SORBOL() műveletet.

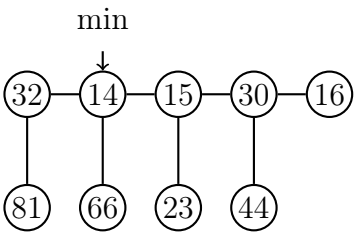
Amortizált költségelemzés: A legrosszabb eset alapú költségelemzésnél előfordulhat, hogy a becslésünk túlságosan is pesszimista lesz, ha a nagyobb költségű műveletek valójában ritkábban fordulnak elő az algoritmus végrehajtása során. Ennek kiküszöbölésére használjuk az amortizált költségelemzést, ami az egyes műveletek költségére ad felső korlátot a legrosszabb esetre. Amortizált költségelemzésre több módszer is van: összesítéses-, könyvelési- és potenciálmódszer.

FELADATOK

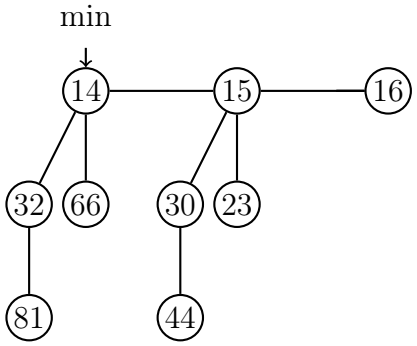
Szúrjuk be egy üres Fibonacci kupacba az alábbi elemeket: 16, 32, 81, 2, 14, 66, 15, 23, 44, 30.



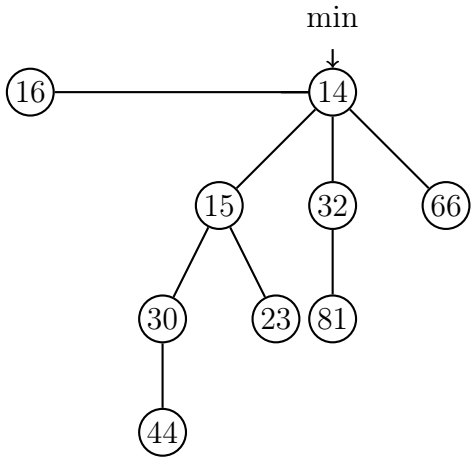
Hajtsunk végre két SORBÓL() műveletet!



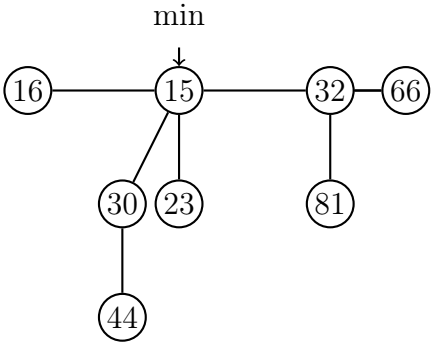
(a) SORBOL()



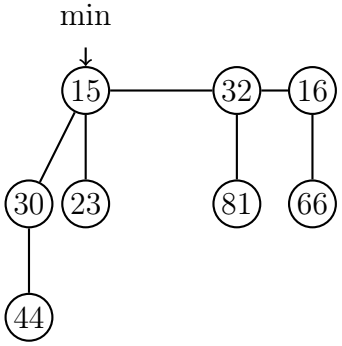
(a) SORBOL()



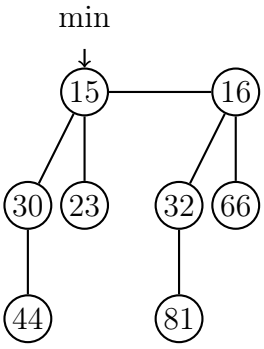
(a) SORBOL() végeredmény



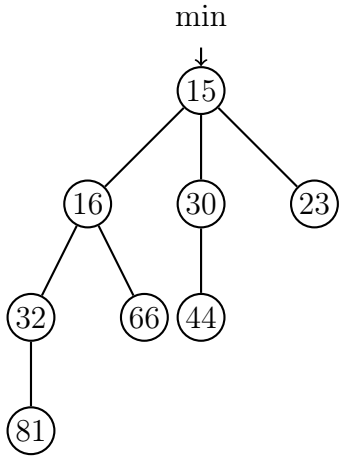
(b) SORBOL()



(b) SORBOL()

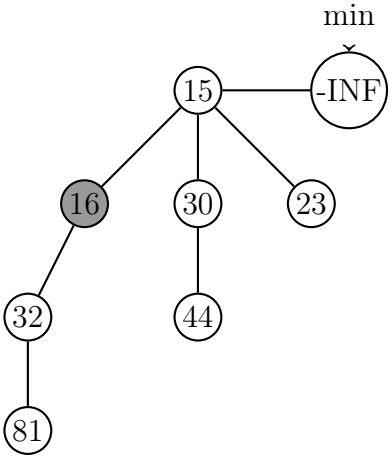


(b) SORBOL()

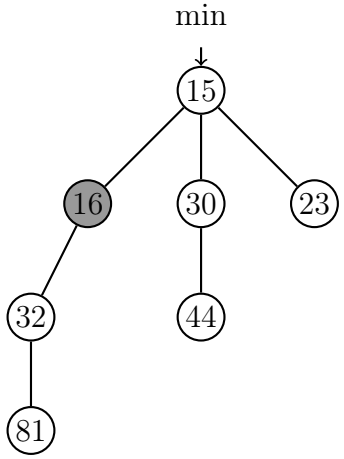


(b) SORBOL() végeredmény

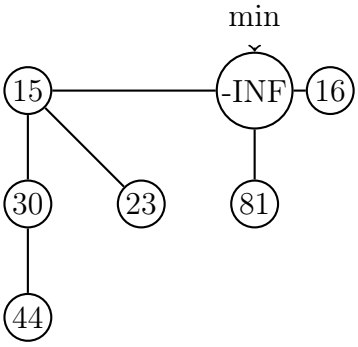
Hajtsuk végre az alábbi műveleteket: TÖRÖL(66), TÖRÖL(32), TÖRÖL(44)



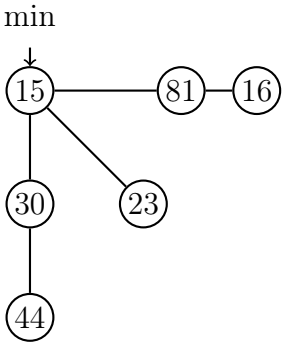
(c) TÖRÖL(66)



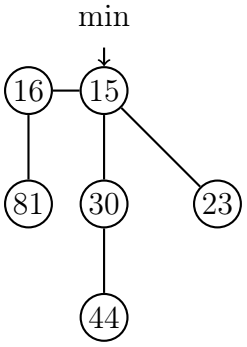
(c) TÖRÖL(66)



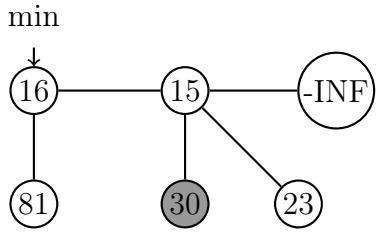
(d) TÖRÖL(32)



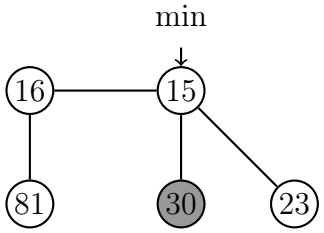
(d) TÖRÖL(32)



(d) TÖRÖL(32)

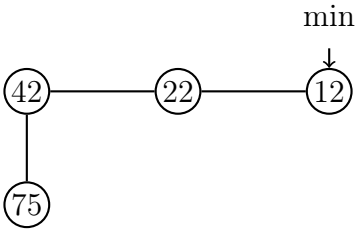


(e) Töröl(44)

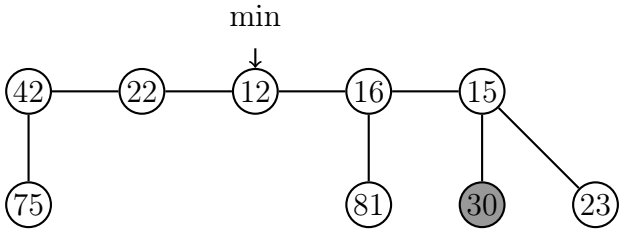


(e) Töröl(44)

A törlések után kapott kupacot egyesítsük az alábbi Fibonacci kupaccal.

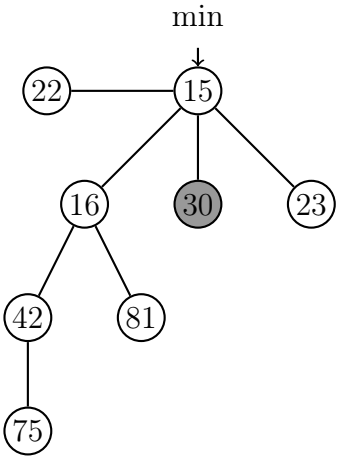


(a) H_2

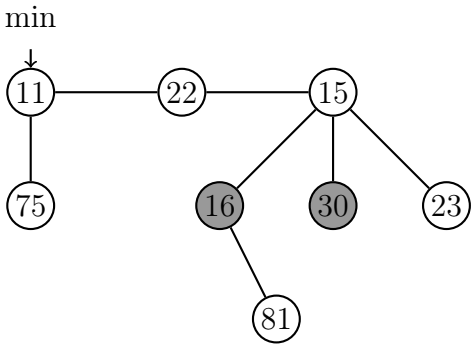


(b) Egyesít(H_1, H_2) eredménye

Végül hajtsunk végre egy Sorból() műveletet, majd módosítsuk a 42 kulcsot 11-re.



(c) Sorból() után



(d) KulcsCsökkent(42,11) után

Az utolsó művelet elvégzése után mennyi lesz a Fibonacci kupac potenciálfüggvényének értéke?

A H kupacra számított potenciálfüggvény jele $\Phi(H)$. A potenciálfüggvény egy adatszerkezet pillanyatnyi állapotához rendel egy potenciált. Ezeket a potenciálokat felhasználva a potenciálváltozást elemezve számíthatunk amortizációs költséget. A Fibonacci-kupacok esetén legyen $\Phi(H) = t(H) + 2m(H)$, ahol $t(h)$ a kupacot alkotó fákna a száma és $m(h)$, a megjelölt csúcsok száma.
Tehát a feladat megoldása: $\Phi(H) = t(H) + 2m(H) = 3 + 2 * 2 = 7$

Műveletek n hosszú sorozatát végezzük el egy adatszerkezeten. Az i-edik művelet költsége i, ha i éppen kettőhatvány, máskülönben 1. Mennyi az adatszerkezet műveletenkénti amortizációs költsége? Használjuk az összesítéses módszert.

$$c_i = \begin{cases} i, & \text{ha } i = 2^k \\ 1, & \text{különben.} \end{cases}$$

Elemezzünk kicsit a feladatot. Egy ilyen műveletsorozatnál igaz lesz, hogy minden kettőhatványadik művelet költsége nagyobb lesz (2,4,8,16,32...), azonban az összes többi művelet költsége 1. Itt érezzük, hogy a kevés nagy költség arányaiban annyira nem lassítja a rendszert. Lerosszabb esettel történő költségelemzés azt mondaná, hogy az n. műveletnek legrosszabb esetben n a költsége (ez a kettőhatványoknál igaz, mert ugye például 8 egy kettőhatvány és ilyenkor a művelet költsége is 8 lesz). Ha ilyen műveletből van n darab, akkor annak a legrosszabb esetbeli költsége $O(n^2)$.
Térjünk rá az összesítéses módszerre, ahol a feladatunk, hogy adjuk össze minden művelet költségét, és erre adjunk felső korlátot. Képlettel: $\sum_{i=1}^{n} c_i \leq T(n)$
Látjuk, hogy kétféle műveletünk van, úgyhogy próbáljunk meg matematikailag egy felső korlátot adni arra, hogy ha összesen n darab műveletet hajtunk végre, akkor ebből hány darab lesz az egyik és hány darab a másik típusból. Ha ez megvan, akkor a költség összegzésével kiszámolhatjuk a felső korlátot.
A $c_i = 1$ költségű műveletekből biztos, hogy kicsivel kevesebb, mint n darab lesz, mert ugye azt mondtuk, hogy az 1.-től az n.-ig minden kettőhatványadik sorszámú műveletet kivéve 1 költségűek a műveleteink. Ebből adódóan, ha azt mondjuk, hogy n darab $c_i = 1$ költségű műveletünk lesz, az jó felső körlát. Ekkor az összköltség $n * 1$, így megkapjuk $T(n) = n$ felső korlát.
Az $c_i = i$ költségű műveletből kevés lesz. Egész pontosan $\log_2 n$ darab. Szóval összegeznünk kell $\log_2 n$ darab kettőhatványt. Képlettel: $\sum_{j=0}^{\lfloor \log n \rfloor} 2^j$. Ezt át tudjuk alakítani a mértani sorok összegképletével, és a logaritmus hatvány azonossággal, így az eredmény $2n - 1$. Tehát jó felső korlát a $T(2n)$.
Még nem végeztünk mert a feladat eredeti célja az n darab műveletre adandó felső korlát. Most, hogy megvan a két típusra a felső korlátunk, elég csak összeadnunk őket, tehát $T(n) = n + 2n = 3n = O(n)$, mivel a konstans elhagyható. Ha n művelet amortizált költsége $O(n)$, akkor 1 műveleté $O(1)$.

Emlékeztetőül a mértani sorok összegképlete: $\sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1}$

Emlékeztetőül a logaritmus hatvány azonossága: $a^{\log_a b} = b$

8. gyakorlat – Geometriai algoritmusok

FOGALMAK

A geometriai algoritmusok alkalmazási területei többek között a számítógépes tervezés és grafika, a robotika és a statisztika. Egy geometriai algoritmus **bemenete általában egy geometriai objektumhalmaz** leírása, például pontok, szakaszok vagy egy poligon óramutató járásával ellentétes körüljárás szerint felsorolt csúcsai. A **kimenet** gyakran csak egy **válasz** az objektumokkal kapcsolatos kérdésre, de lehet egy új **alakzat** is, például egy ponthalmaz konvex burka. A geometriai algoritmusok számos gyakorlati probléma (pl. gépi tanulás) megoldása során felmerülnek. **A numerikus hibák (egy része) kiküszöbölhető a FORGÁSIRÁNY használatával.**

Konvex kombináció: A $P_3 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix}$ pontot $P_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$ és $P_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$ pontok konvex kombinációjának nevezzük, amennyiben $x_3 = (1-\alpha)x_1 + \alpha x_2$, valamint $y_3 = (1-\alpha)y_1 + \alpha y_2$ teljesül valamely $0 \leq \alpha \leq 1$-ra.

Szakasz: $\overline{P_1P_2}$ szakasz a P_1 és P_2 pontokból vett konvex kombinációik halmaza.

Keresztszorzat: A $\overline{P_0P_1}$ és $\overline{P_1P_2}$ **vektorokból készített mátrix determinánsa**. A keresztszorzat megadja a **vektorok forgásirányát**. Ha $P_1 \mathrm{x} P_2$ **pozitív**, akkor P_1 az óramutató járásával egyező forgásirányba esik P_2-höz képest, az origóból nézve, tehát P_2 **balra van**. Ha **negatív**, akkor P_1 az óramutató járásával ellentétes forgásirányba esik P_2-től, tehát P_2 **jobbra van**. Abban az esetben, ha a vektorok **„egymáson fekszenek"** és azonos, vagy ellentétes irányításúak a keresztszorzat **nulla**.
Kérdés: Vajon két egymást követő szakasz, $\overline{P_0P_1}$ és $\overline{P_1P_2}$ jobbra vagy balra fordul-e el egymáshoz képest a P_1 pontban?
Megoldás: Keresztszorzat segítségével a kérdést a szög kiszámolása nélkül is meg tudjuk válaszolni. Transzformáljuk a $\overline{P_0P_1}$ vektort és a P_2 pontot az origóba: P_0-ból, P_1-ből és P_2-ből is P_0-t kivonva origó központúvá tesszük a koordinátarendszerünket. Végül számítsuk ki, hogy az új P_2 merre helyezkedik el az eddigi $\overline{P_0P_1}$ szakaszhoz képest. **Ha az eredmény negatív, akkor jobbra fordul, ha pedig pozitív, akkor balra.**

```
FORGÁSIRÁNY(P0, P1, P2) {
   return (P1.x-P0.x)*(P2.y-P0.y) - (P2.x-P0.x)*(P1.y-P0.y)
}
```

Átfogó szakasz: A $\overline{P_1P_2}$ szakasz átfog egy egyenest, ha a P_1 pont az egyenes egyik oldalára, a P_2 pont pedig a másik oldalára esik. Határesetben P_1 vagy P_2 illeszkedik az egyenesre. Két szakasz akkor és csak akkor metszi egymást, ha a következő két feltétel valamelyike (vagy mindkettő) fennáll:

- Mindkét szakasz átfogja a másik egyenesét.
- Az egyik szakasz egyik végpontja illeszkedik a másik szakaszra. (Ez a feltétel felel meg a határesetnek.)

Kérdés: Vajon két szakasz metszi-e egymást?
Megoldás: mindkét szakaszra ellenőrizzük, hogy az átfogja-e a másik egyenesét.

```
MetszőSzakaszok(A, B, C, D) {
   d1 = Forgásirány(A, B, C)
   d2 = Forgásirány(A, B, D)
   d3 = Forgásirány(C, D, A)
   d4 = Forgásirány(C, D, B)
   return d1 * d2 < 0 és d3 * d4 < 0
}
```

Az itt feltüntetett pszeudokóddal csak „valódi" metszéseket találunk meg, a szakaszra illeszkedő végpontú szakaszt nem kezeltük le.

Metsző-szakaszpárok: Adott szakaszok (n elemű) S halmaza.
Kérdés: Vajon van-e köztük egymást metsző szakaszpár?
Az algoritmus használ egy **„söprésnek"** nevezett technikát, mely sok más geometriai algoritmusban is előfordul. A söprés során egy képzeletbeli függőleges söprő egyenes halad át a geometriai elemek adott halmazán, általában balról jobbra.
Megoldás: Rendezzük az S-beli szakaszok kezdő és végpontjait **x koordináta szerint** a növekvő sorrendbe. Abban az esetben, ha több pont x koordinátája **megegyezik**, tehát holtverseny van, a szakasz **kezdőpontokat** a szakasz végpontok elé soroljuk. Ha ezen belül is van még további holtverseny (ugyan az az x koordináta több szakaszkezdő és/vagy szakaszzáró pontnál), akkor pedig a **kisebb y-koordinátájú** pontok nagyobbak elé sorolásával döntjük el. Ezután inicializáljunk és tartsunk fent egy **kiegyensúlyozott keresőfát** (pl: AVL-fa, vagy piros-fekete fa), amiben a szakaszokat rendezett módon fogjuk tárolni, legyen ez T. A metszés eldöntéséhez két esetet vizsgálunk: a **belépő szakaszok metszik-e a** T fában lévő **megelőzőjüket**/rákövetkezőjüket, valamint a **kilépő szakaszok megelőzője és rákövetkezője metszi-e egymást**. A fában való elhelyezkedése a szakaszoknak leírja a térbeli elhelyezkedésüket is, tehát egy csúcs megelőzője a térben a közvetlenül alatta lévő szakasz lesz, a rákövetkezője pedig a térben közvetlenül felette lévő szakasz.

```
Van-e-Metsző-Szakaszpár(S) {
  L = S-beli szakaszok végpontjainak rendezett listája
  for p in L{
    if p egy s szakasz bal végpontja { //belépő szakasz
      Beszur(T,s) //azt a szakaszt, amely belép, betesszük az aktuális szakaszlistába
      if Megelozo(T,s) vagy Rakovetkezo(T,s) metszi s-et {
        return Igaz
      }
    }
    if p egy s szakasz jobb végpontja { //kilépő szakasz
      if Megelozo(T,s) metszi Rakovetkezo(T,s)-t {
        return Igaz
      }
      Torol(T,s) //azt a szakaszt, amely kilép, töröljük az aktuális szakaszlistából
    }
  }
  return Hamis
}
```

Megjegyzés: a kiegyensúlyozott keresőfába való beszúráskor, ahhoz, hogy eldöntsünk, hogy hova kell beszúrnunk, azt kell megvizsgálnunk, hogy az új szakasz kezdőpontja a már T fában lévő szakaszokhoz képest y koordináta szerint lejjebb vagy feljebb van. A fából való törléskor helyettesítsünk a megelőzőjével, és ha felborul az egyensúlyi tulajdonság, javítsuk a fát.

Konvex-burok: Q ponthalmaz konvex burka az a legkisebb P konvex poligon, amelyre Q minden pontja vagy P határán vagy a belsejében van. Q konvex burkát $CH(Q)$-val jelöljük.
Kérdés: Mi egy adott Q ponthalmaz konvex burka?
Megoldás: Két algoritmust is ismertetünk, mellyel meghatározható a konvex burok. Az egyik a Graham-féle pásztázás, a másik a Jarvis-menetelés. A konvex burok definíciójából adódóan tudni fogjuk, hogy $CH(Q)$ minden csúcsa a Q halmaz egy pontja. Mindkét algoritmus kihasználja ezt a tulajdonságot, és csak azt dönti el, hogy mely Q-beli csúcsokat tartsa meg a konvex burok csúcsaként, és mely Q-beli csúcsokat dobja el.

Graham-féle pásztázás: Az algoritmus egy S verem segítségével dolgozik. Az adott Q halmaz minden pontját beírjuk egyszer a verembe, majd azokat a pontokat, amelyek nem csúcsai $CH(Q)$-nak, előbb vagy utóbb kivesszük a veremből. Amikor az algoritmus véget ér, S pontosan $CH(Q)$ csúcsait tartalmazza felülről lefelé nézve az óramutató járásával ellenkező irányban.
A LEGFELSŐ függvény visszaadja az S verem legfelső pontját S megváltoztatása nélkül, valamint a LEGFELSŐ-ALATTI függvény visszaadja az S verem legfelső eleme alatt eggyel lévő pontot S megváltoztatása nélkül.

```
GRAHAM-PÁSZTÁZÁS(Q) {
  P0 = minimális y-koordinátájú Q-beli pont (több ilyen
  esetén válasszuk az x-koordináta szerint is minimálisat)
  P = POLARSZOGSZERINTRENDEZ(Q, P0)
  S = VERMETLETESIT()
  VEREMBE(P[0], S)
  VEREMBE(P[1], S)
  VEREMBE(P[2], S)
  for i=3 to P.length {
    while FORGASIRANY(LEGFELSOALATTI(S), LEGFELSO(S), P[I]) <= 0 {
      //nem balra fordult
      VEREMBOL(S)
    }
    VEREMBE(P[i], S)
  }
  return S
}
```

Polárszög szerinti rendezés: a korábban meghatározott P_0 ponton áthaladó, vízszintes (x-tengellyel párhuzamos) egyenessel bezárt előjeles szög alapján növekvő sorrendben (ha több mint egy pontnak ugyanaz a szöge, csak a P_0-tól legtávolabbit hagyjuk meg).
Belső ciklusok magyarázata: A for ciklussal végigjárjuk a polárszög szerint rendezett csúcsokat. A while ciklus távolítja el azokat a pontokat a veremből, amik nem csúcsai a konvex buroknak. A metódus alapja, hogy amikor a konvex burkot az óramutató

járásával ellentétes irányban járjuk be, akkor minden csúcsában balra kell fordulnunk. Ezért minden alkalommal, amikor a while ciklus olyan csúcsot talál a veremben, melynél nem fordulunk balra, kivesszük az illető csúcsot a veremből. (Azzal, hogy a "balra nem fordulást" vizsgáljuk a jobbra fordulás helyett, kizárjuk az egyenesszög lehetőségét is az eredményül kapott konvex burok csúcsainál. Nem akarunk egyenesszöget, mivel definíció szerint a konvex poligon csúcsa nem állhat elő a poligon többi csúcsának konvex kombinációjaként.) Miután minden olyan csúcsot kivettünk, amelyeknél $P[i]$ felé haladva nem fordulunk balra, betesszük $P[i]$-t a verembe.

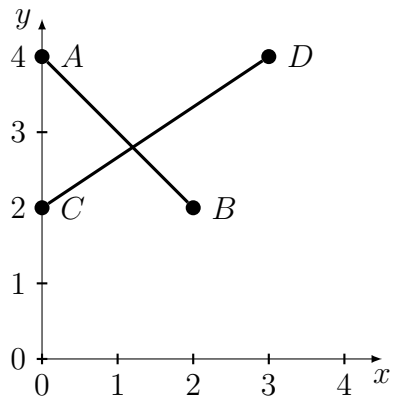
Jarvis-menetelés: Jarvis menetelése olyan, mintha a Q ponthalmazt egy feszesen tartott papírral akarnánk becsomagolni. Először is odaragasztjuk a papír végét a halmaz legalsó pontjához, (ugyanahhoz a $P[0]$ ponthoz, amellyel a Graham-féle pásztázást indítjuk). Ez a pont már a konvex burok egy csúcsát képezi. Innen jobbra húzva feszesen tartjuk a papírt, majd feljebb húzzuk, míg el nem érjük a leginkább balra eső pontot. Ennek a pontnak szintén a konvex burok csúcsának kell lennie. A papírt feszesen tartva addig haladunk tovább ily módon a csúcsok halmaza körül, míg vissza nem érünk a kiindulási $P[0]$ pontba.

```
JARVIS-MENETELÉS(Q) {
  P = minimális y-koordinátájú Q-beli pont (több ilyen
  esetén válasszuk az x-koordináta szerint is minimálisat)
  P0 = P
  R = ABCRENDEZETTPONTOK(Q)
  S = VERMETLETESIT()
  while (S.count() == 0 || P != P0) {
    referencia = R[P+1]     //abc rendezes szerint a P utani
    for (i=0 to R.length) {
        if(FORGASIRANY(P,REFERENCIA,R[I])<0) {
            //talaltunk egy csucsot akitol meginkabb balra vagyunk
            referencia = P[i]
        }
        if(FORGASIRANY(P,REFERENCIA,R[I])=0) {
            //ugyan arra az egyenesre esnek
            if(P-tol tavolabb van P[i], mint a referenciapont) {
                referencia = P[i]
            }
        }
    }
    VEREMBE(referencia, S)
    P = referencia
  }
  return S
}
```

Belső ciklusok magyarázata: Amíg vissza nem érünk a kezdőpontba, válasszuk ki a legutolsónak választott ponttól leginkább jobbra eső pontot (vagyis azt a pontot, amelytől minden további ponthoz balra fordulásra van szükség). Majd adjuk hozzá a kiválasztott csúcsot a konvex burokhoz.

FELADATOK

Döntsük el az $A = [0, 4], B = [2, 2]$, valamint a $C = [0, 2], D = [3, 4]$ végpontokkal adott szakaszokról, hogy metszik-e egymást?



I. $\overline{CD}$ átfogja-e $\overline{AB}$-t?

I/a) $\textsc{Forgásirány}(A, B, C) = \det\left(\begin{bmatrix} 2-0 & 0-0 \\ 2-4 & 2-4 \end{bmatrix}\right) = \det\left(\begin{bmatrix} 2 & 0 \\ -2 & -2 \end{bmatrix}\right) = -4 < 0$

$\Rightarrow \overrightarrow{AB}$ szakaszhoz képest a C csúcs jobbra fordulva érhető el

I/b) $\textsc{Forgásirány}(A, B, D) = \det\left(\begin{bmatrix} 2-0 & 3-0 \\ 2-4 & 4-4 \end{bmatrix}\right) = \det\left(\begin{bmatrix} 2 & 3 \\ -2 & 0 \end{bmatrix}\right) = 6 > 0$

$\Rightarrow \overrightarrow{AB}$ szakaszhoz képest a D csúcs balra fordulva érhető el

II. $\overline{AB}$ átfogja-e $\overline{CD}$-t?

II/c) $\textsc{Forgásirány}(C, D, A) = \det\left(\begin{bmatrix} 3-0 & 0-0 \\ 4-2 & 4-2 \end{bmatrix}\right) = \det\left(\begin{bmatrix} 3 & 0 \\ 2 & 2 \end{bmatrix}\right) = 6 > 0$

$\Rightarrow \overrightarrow{CD}$ szakaszhoz képest a A csúcs balra fordulva érhető el

II/d) $\textsc{Forgásirány}(C, D, B) = \det\left(\begin{bmatrix} 3-0 & 2-0 \\ 4-2 & 2-2 \end{bmatrix}\right) = \det\left(\begin{bmatrix} 3 & 2 \\ 2 & 0 \end{bmatrix}\right) = -4 < 0$

$\Rightarrow \overrightarrow{CD}$ szakaszhoz képest a B csúcs jobbra fordulva érhető el

I. és II. alapján kijelenthető, hogy az $\overline{AB}$ és $\overline{CD}$ szakaszok metszik egymást

Döntsük el az $A = [0, 4], B = [2, 2]$, valamint a $C = [1, 0], D = [3, 3]$ végpontokkal adott szakaszokról, hogy metszik-e egymást?

I. $\overline{AB}$ átfogja-e $\overline{CD}$-t?

I/a) $\textsc{Forgásirány}(C, D, A) = \det\left(\begin{bmatrix} 3-1 & 0-1 \\ 3-0 & 4-0 \end{bmatrix}\right) = \det\left(\begin{bmatrix} 2 & -1 \\ 3 & 4 \end{bmatrix}\right) = 8 + 3 > 0$

$\Rightarrow \overrightarrow{CD}$ szakaszhoz képest a A csúcs balra fordulva érhető el

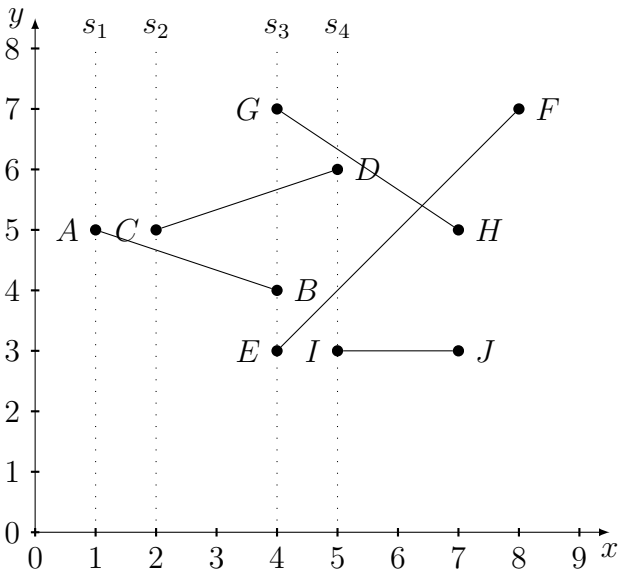
I/b) $\textsc{Forgásirány}(C, D, B) = \det\left(\begin{bmatrix} 3-1 & 2-1 \\ 3-0 & 2-0 \end{bmatrix}\right) = \det\left(\begin{bmatrix} 2 & 1 \\ 3 & 2 \end{bmatrix}\right) = 6 - 3 > 0$

$\Rightarrow \overrightarrow{CD}$ szakaszhoz képest a B csúcs balra fordulva érhető el

$\Rightarrow \overline{AB}$ nem fogja át a $\overline{CD}$-re illeszkedő egyenest, így $\overline{AB}$ nem is metszheti $\overline{CD}$-t.

Hatékony algoritmussal határozzuk meg, hogy az alábbi szakaszok között található-e egymást metsző szakaszpár!

$\overline{AB} = [(1,5),(4,4)]$ $\overline{CD} = [(2,5),(5,6)]$ $\overline{EF} = [(4,3),(8,7)]$ $\overline{GH} = [(4,7),(7,5)]$
$\overline{IJ} = [(5,3),(7,3)]$



Rendezzük a szakaszok pontjait x-koordinátájuk szerint. A holtversenyeknél a kezdőpontokat helyezzük előrébb a végpontoknál. Az esetleges további holtversenyeket a kisebb y-koordinátájú pontok nagyobbak elé sorolásával oldjuk föl.
Eredmény: $A, C, E, G, B, I, D, J, H, F$

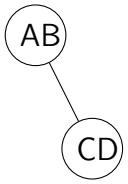
A szakaszokat tartalmazó kiegyensúlyozott (itt most AVL[1]) keresőfa állapotai a seprőegyenes (s_i) haladása szerint.

1. s_1 mentén
   (a) `Be(AB)`

   Metszi-e $\overline{AB}$ a fabeli megelőzőjét vagy rákövetkezőjét?

2. s_2 mentén
   (a) `Be(CD)`

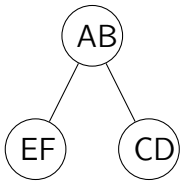
   

   Metszi-e $\overline{CD}$ a fabeli megelőzőjét vagy rákövetkezőjét?

3. s_3 mentén

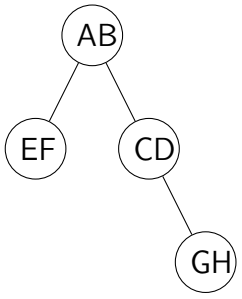
[1] Hf.:piros-fekete fával is végignézni

(a) `Be(EF)`



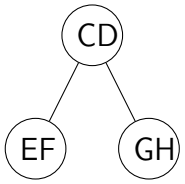
Metszi-e $\overline{EF}$ a fabeli megelőzőjét vagy rákövetkezőjét?

(b) `Be(GH)`



Metszi-e $\overline{GH}$ a fabeli megelőzőjét vagy rákövetkezőjét?

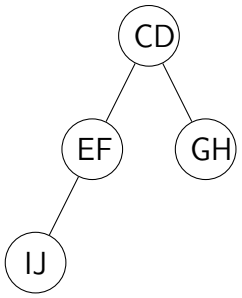
(c) `Ki(AB) - megelőzővel helyettesítés, majd avl-fa javítás`



Metszi-e egymást $\overline{AB}$ fabeli megelőzője és rákövetkezője?

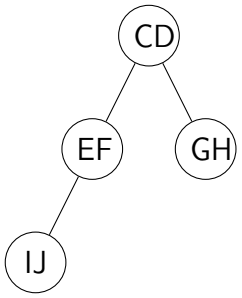
4. s_4 mentén

(a) `Be(IJ)`



Metszi-e $\overline{IJ}$ a fabeli megelőzőjét vagy rákövetkezőjét?

(a) `Ki(CD)`



Metszi-e egymást $\overline{CD}$ fabeli megelőzője és rákövetkezője? Metszést találtunk! Visszatér az algoritmus, a CD törlése már nem hajtódik végre!

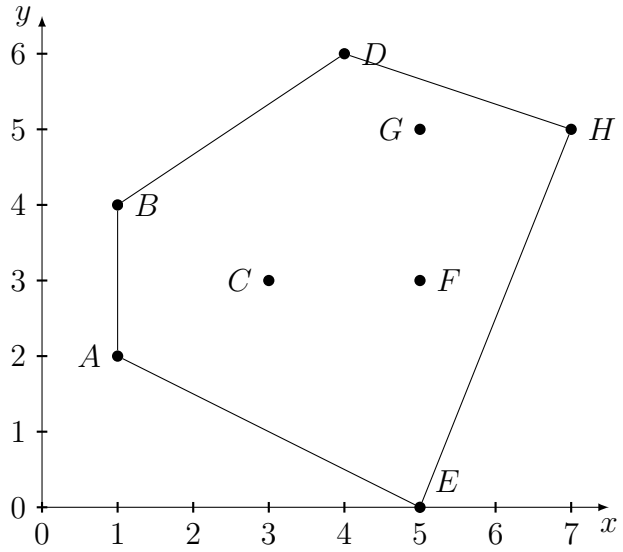
Határozzuk meg a (1,2), (1,4), (3,3), (4,6), (5,0), (5,3), (5,5), (7,5) pontok konvex burkát Graham-féle pásztázással, illetve Jarvis meneteléssel!

Graham-féle pásztázás

I. lépés: csúcsok polárszög szerinti rendezése: E, H, G, D, C, B, A.
II. lépés: a konvex burok csúcsait nyilvántartó verem fenntartása.

1. S_0=[E, H, G] (ekkor még nem kell forgásirányt számoljunk)
2. ForgásIrány(H,G,D)= -2, S_1=[E,H,D]
3. ForgásIrány(H,D,C) = 10, S_2=[E,H,D,C]
4. ForgásIrány(D,C,B) = -7, S_3=[E,H,D,B]
5. ForgásIrány(D,B,A) = 6, S_4=[E,H,D,B,A]

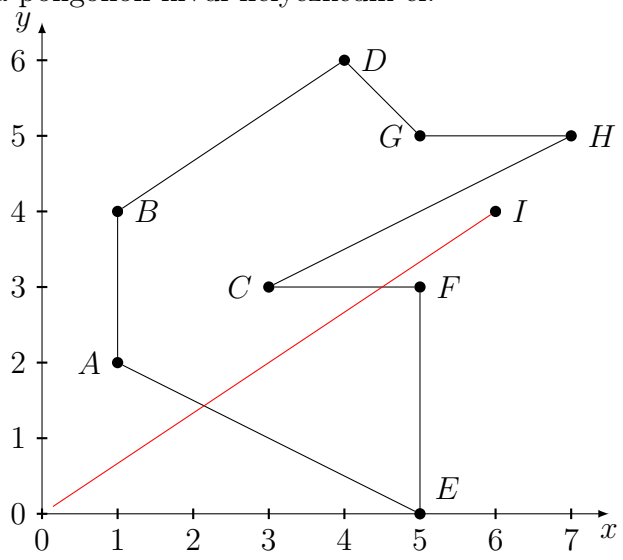
Jarvis menetelés



1. iteráció	2. iteráció	3. iteráció	4. iteráció	5. iteráció
FI(E,A,F)=12	FI(H,A,A)=0	**FI(D,A,E)=-22**	**FI(B,A,C)=-4**	FI(A,A,B)=0
FI(E,B,F)=12	**FI(H,B,A)=-12**	**FI(D,B,A)=-6**	FI(B,B,A)=0	FI(A,B,B)=0
FI(E,C,F)=6	FI(H,C,B)=8	FI(D,C,B)=7	FI(B,C,A)=4	**FI(A,C,B)=-4**
FI(E,D,F)=3	**FI(H,D,B)=-9**	FI(D,D,B)=0	FI(B,D,A)=6	FI(A,D,C)=5
FI(E,E,F)=0	FI(H,E,D)=17	FI(D,E,B)=20	FI(B,E,A)=8	**FI(A,E,C)=-8**
FI(E,F,F)=0	FI(H,F,D)=8	FI(D,F,B)=11	FI(B,F,A)=8	FI(A,F,E)=12
FI(E,G,F)=0	FI(H,G,D)=2	FI(D,G,B)=5	FI(B,G,A)=8	FI(A,G,E)=20
FI(E,H,G)=-6	FI(H,H,D)=0	FI(D,H,B)=9	FI(B,H,A)=12	FI(A,H,E)=24
$H \in CH(Q)$	$D \in CH(Q)$	$B \in CH(Q)$	$A \in CH(Q)$	$E \in CH(Q)$

Döntsük el az előző feladat ponthalmazához tartozó zárt nemmetsző poligonjához képest az $I = (6, 4)$ pont belül vagy kívül helyezkedik-e el!

Válasszunk egy garantáltan poligonon kívüli K pontot, és vizsgáljuk $\overline{IK}$-nak a poligon oldalaival való metszéspontjainak az m számát. Ha m páros, akkor biztos, hogy I is a poligonon kívül helyezkedik el.



9. gyakorlat – Mintaillesztés és moduláris hatványozás

FOGALMAK

Szövegszerkesztő programokban gyakori feladat megkeresni egy szövegben egy **minta összes előfordulását**. A szöveg rendszerint a szerkesztendő dokumentum, a keresendő minta pedig a felhasználó által megadott szó. A feladat megoldására szolgáló gyors algoritmusok jelentősen javíthatják a szövegszerkesztő programok hatékonyságát. Ezeket az ún. minta-illesztő algoritmusokat ennél jóval szélesebb körben használják, például, amikor egy bizonyos mintát keresnek **DNS-láncokban**, vagy akár a **természetes nyelvi beszédfeldolgozásnál** is.

Alapfogalmak:
Determinisztikus, véges állapotú automata: $M = (Q, q_0, A, \Sigma, \delta)$, ahol

- Q : állapotok véges, nemüres halmaza
- q_0 : kezdőállapot
- A : végállapotok halmaza
- Σ : input ábécé (betűk véges, nemüres halmaza)
- $\delta = Q \times \Sigma \to Q$: átmenetfüggvény

Egy automata **elfogad** egy stringet, ha az utolsó betű feldolgozása után **végállapotba kerülünk**. Ha nem végállapotban áll meg, akkor elutasítja. Kétféle módon is meg lehet adni egy automatát: gráffal vagy táblázatos módon.
Akkor **determinisztikus** egy automata, ha minden állapotra, minden lehetséges átmenet definiálva van.

Σ^*: a Σ ábécé betűiből képzett véges hosszú szavak halmaza, tehát az összes lehetséges szó, ami kirakható az ábécénkből. *Azaz az ábécé betűinek összes lehetséges kombinációja.*
ϵ: üres szó, ami mindig eleme Σ^*-nak
$|w|$: a w szó hossza (pl: $|aba| = 3$)
uw: az u és w szavak konkatenációja (pl: $w : aba, u : bba \to uw : bbaaba$)
$w \sqsubset u$: azaz w prefixe (kezdőszelete) u-nak, azaz u szó a w-vel kezdődik
$w \sqsupset u$: azaz w szuffixe (zárószelete) u-nak, azaz u szó a w-vel végződik
Megjegyzés: Az $\texttt{Y} \sqsupset \texttt{Y}$*, valamint az* $\epsilon \sqsupset \texttt{Y}$ *relációk triviálisan teljesülnek minden* `Y`*-ra.*
Szuffix függvény: $\sigma(x)$: $\Sigma^* \to \{0, ..., m\}$
$\sigma(x)$: x szó azon leghosszabb szuffixének hossza, ami egyben a P minta prefixe

Mintaillesztés véges automatával:
Az állapotátmeneteket határozzuk meg a szuffix függvény segítségével.
Mintaillesztő automata egységei (P minta esetén, ahol $|P| = m$):

- Σ ábécé, a P minta összes betűjét tartalmazza
- Állapotok halmaza: $\{0, ..., m\}$, ahol minden q_k állapotra igaz, hogy a P szót a k-adik karakterig sikeresen megtaláltuk
- Kezdőállapot: q_0 (még a P minta egyetlen karatkerére sem illesztettünk)
- Végállapot: q_m (a teljes P mintára sikeresen illesztettünk)
- **Átmenetfüggvény**: pl: q_5 állapotra és a betűre $\to \delta(q_5, a) = \sigma(P_{q_5}a)$, ahol P_q a minta q hosszú prefixe. **Előnye**, hogy egy sikertelen illesztéskor nem kell előről kezdeni a szó illesztését, hanem az első lehetséges folytatási ponttól megyünk tovább.

Knuth-Morris-Pratt algoritmus:
Az automatáktól eltérően δ **kiszámítása helyett egy** π **segédfüggvényt** használ.
$\pi : \{1, ..., m\} \to \{0, ..., m\}$ prefixfüggvény, ahol $\pi(q) = max(k : k < q, P_k \sqsupset P_q)$, tehát a q-adik indexhez azt a maximális k számot rendeli, amire igaz, hogy kissebb, mint q és igaz, hogy **a minta** k **hosszú prefixe egyben szuffixe a minta** q **hosszú prefixének**. Más szavakkal, P_{q-1} leghosszabb prefixe, ami valamely P_q prefixnek a szuffixe is.
Lényegében azt szeretnénk, hogyha a mintánk egy részét már sikeresen megtaláltuk, de tovább nem illeszkedik a minta, akkor ne kelljen elölről kezdenünk a keresést. Helyette ellenőrizzük le, hogy az eddig megtalált minta részletnek van-e olyan prefixe ami megyegyezik az eddig beolvasott szöveg szuffixével.

Rabin-Karp algoritmus:
Az input továbbra is egy S szöveg, amiben egy P mintát keresünk. Az alap kiindulási ötlet, hogy a Σ **ábécé feletti szavakra ne betűsorozatokként, hanem** $|\Sigma| = d$ **számrendszerű számokként tekintsünk**. Így a P mintánkat is kezelhetjük számként.
A P mintához rendelt T szám egyértelműen meghatározható, az alábbi képlettel:
$T(p_{1...m}) = p_m + d(p_{m-1} + d(p_{m-2} + ... + d(p_2 + dp_1)))$.
Az algoritmus lényege, hogy végigmegy az S inputon és minden indexen kivág a bementből egy $|P| = m$ hosszú résszót, és ha **a kivágott résszó hasított értéke megyezik** P **hasított értékével**, akkor az azt jelzi, hogy illeszkedést találtunk.
Tehát algoritmusnak az S szöveg minden i indexére ki kellene számítania a $T(s_{i...i+m-1})$ számot és a P minta akkor illeszkedik, ha $T(s_{i...i+m-1}) = T(p_{1...m})$. Alapvetően ez az érték gyorsan számolható az **„ablak" eltolás módszerével**, de mivel a gépi számábrázolás véges, így könnyen elérhetjük a számábrázolásunk korlátait. Erre a megoldást az adja, hogy alkalmazzunk egy hasítófüggvényt. Legyen ez $h(x) = x \mod q$.
Így az ablakolás számítható az alábbi módon:
$T(s_{i+1...i+m}) = (d \times (h(s_{i..i+m-1}) - h(d^{m-1}) \times s_i)) + s_{i+m}) \mod q$,
ahol d az aktuális számrendszer, valamint s_i a kilépő szám, s_{i+m} a belépő számjegy, $h(s_{i..i+m-1})$ az aktuláis indextől vett m hosszú részletre alkalmazott hasítófüggvény értéke.
Sajnos a hasítófüggvény használatával igaz lesz, hogy **nem csak akkor kapunk egyenlőséget az ablakoláskor, ha pontos illeszkedés van**, hanem akkor is, ha egy másik olyan számmal számoltunk, melyre ugyan azt adja a *mod* függvény. Emiatt, minden egyenlőségnél még le kell ellenőriznünk, hogy valóban egyezést találtunk-e vagy csak hamis riasztás volt.

Moduláris hatványozás: Az alapfeladat, hogy $a^b \mod n$ alakú kifejezések értékét határozzuk meg minél gyorsabban. Az algoritmus első lépése, hogy n-t váltsuk át 2-es számrendszerbe. Második lépésben menjünk végig n kettes számrendszer beli értékének karakterein és minden iterációban legyen a $d = d^2 \mod n$ értéket, majd ha ezen felül épp 1-es karakteren állunk, d új értéke legyen $d = d * a \mod n$.

```
ModularisHatvanyozas(a, b, n) {
  binaryB = b 2es szamrendszer beli erteke
  d = 1
  for i=1 to binaryB.length {
    d=d^2 mod n
    if(binaryB[i] == 1) {
        d=d*a mod n}}
```

```
return d}
```

Példa a Knuth-Morris-Pratt algoritmus π függvény segítségével történő mintaillesztésre:

FELADATOK

1. Adjunk meg egy véges állapotú determinisztikus automatát, ami a `P='aababa'` minta illesztését hajtja végre!

$|P| = 6$ és $\Sigma = \{a, b\}$

Állapotok halmaza: $q_0, ..., q_6$, ahol p_0 kezdőállapot és q_6 végállapot.

Minden állapothoz definiálnunk kell az átmenetet a Σ ábécé összes betűjéhez!

P prefixei: $\{\epsilon(P_0), a(P_1), aa(P_2), aab(P_3), aaba(P_4), aabab(P_5), aababa(P_6)\}$

A nem triviális átmeneteket a szuffix függvény segítségével meghatározhatjuk:

$\sigma(P_0b) = \sigma(b) = 0$, ahol a szuffixek: $\{b, \epsilon\}$

$\sigma(P_1b) = \sigma(ab) = 0$, ahol a szuffixek: $\{ab, b, \epsilon\}$

Magyarázat: q_1 állapotból keressük a nem triviális b átmenetet. A q_1-ig beolvasott P minta részlet a P_1 = 'a'. Fűzzük össze P_1-et és a keresett átmenetet, ekkor kapjuk 'ab'-t. Ennek a szuffixei közül keressük a leghosszabbat ami egyben P prefixe is.

$\sigma(P_2a) = \sigma(aaa) = 2$, ahol a szuffixek: $\{aaa, aa, a, \epsilon\}$

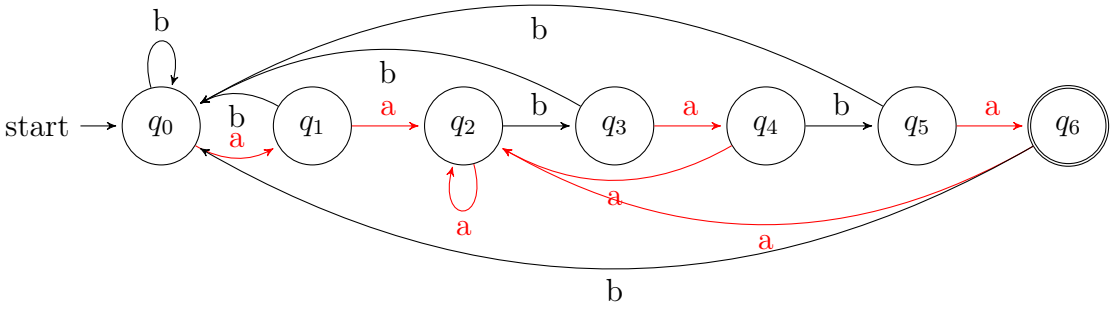
$\sigma(P_3b) = \sigma(aabb) = 0$, ahol a szuffixek: $\{aabb, abb, bb, b, \epsilon\}$

$\sigma(P_4a) = \sigma(aabaa) = 2$, ahol a szuffixek: $\{aabaa, abaa, baa, aa, a, \epsilon\}$

$\sigma(P_5b) = \sigma(aababb) = 0$, ahol a szuffixek: $\{aababb, ababb, babb, abb, bb, b, \epsilon\}$

$\sigma(P_6a) = \sigma(aababaa) = 2$, ahol a szuffixek: $\{aababaa, ababaa, babaa, abaa, baa, aa, a, \epsilon\}$

$\sigma(P_6b) = \sigma(aababab) = 0$, ahol a szuffixek: $\{aababab, ababab, babab, abab, bab, ab, b, \epsilon\}$



$\delta(q_0, a) = q_1 \quad \delta(q_0, b) = q_0$
$\delta(q_1, a) = q_2 \quad \delta(q_1, b) = q_0$
$\delta(q_2, a) = q_2 \quad \delta(q_2, b) = q_3$
$\delta(q_3, a) = q_4 \quad \delta(q_3, b) = q_0$
$\delta(q_4, a) = q_2 \quad \delta(q_4, b) = q_5$
$\delta(q_5, a) = q_6 \quad \delta(q_5, b) = q_0$
$\delta(q_6, a) = q_2 \quad \delta(q_6, b) = q_0$

vagy kicsit olvashatóbban

Q \ Σ	a	b
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_3
q_3	q_4	q_0
q_4	q_2	q_5
q_5	q_6	q_0
q_6	q_2	q_0

1. táblázat. `P` minta illesztését vizsgáló automata δ állapotátmenet-függvénye.

Milyen állapotokat érint az automata a `T='aaabaababa'` input feldolgozása során?

i	0	1	2	3	4	5	6	7	8	9	10
T[i]	–	a	a	a	b	a	a	b	a	b	a
q_i	q_0	q_1	q_2	q_2	q_3	q_4	q_2	q_3	q_4	q_5	$\boxed{q_6}$

2. Adjuk meg a Knuth-Morris-Pratt algoritmus által a `P='aababa'` mintához meghatározott $\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$ prefixfüggvényt!
P prefixei: $\{\epsilon, a, aa, aab, aaba, aabab, aababa\}$

$\pi[1] = 0$, mert $P_1 = a$ és P_0 prefixei $\{\epsilon\}$ valamint P 1 hosszú prefixének a szuffixei $\{\epsilon, a\}$, ezek között a leghosszabb egyező pedig ϵ.

$\pi[2] = 1$, mert $P_2 = aa$ és P_1 prefixei $\{\epsilon, a\}$ és P 2 hosszú prefixének a szuffixei $\{\epsilon, a, aa\}$, ezek között a leghosszabb egyező pedig a.

$\pi[3] = 0$, mert $P_3 = aab$ és P_2 prefixei $\{\epsilon, a, aa\}$ és P 3 hosszú prefixének a szuffixei $\{\epsilon, b, ab, aab\}$, ezek között a leghosszabb egyező pedig ϵ.

$\pi[4] = 1$, mert $P_4 = aaba$ és P_3 prefixei $\{\epsilon, a, aa, aab\}$ és P 4 hosszú prefixének a szuffixei $\{\epsilon, a, ba, aba, aaba\}$, ezek között a leghosszabb egyező pedig a. Vizuálisan

- $\underline{aab} = \texttt{P}_3 \not\supset \texttt{P}_4 = a\underline{aba}$
- $\underline{aa} = \texttt{P}_2 \not\supset \texttt{P}_4 = aa\underline{ba}$
- $\underline{a} = \texttt{P}_1 \sqsupset \texttt{P}_4 = aab\underline{a}$.

$\pi[5] = 0$, mert $P_5 = aabab$ és P_4 prefixei $\{\epsilon, a, aa, aab, aaba\}$ és P 5 hosszú prefixének a szuffixei $\{\epsilon, b, ab, bab, abab, aabab\}$, ezek között a leghosszabb egyező pedig ϵ.

$\pi[6] = 1$, mert $P_6 = aababa$ és P_5 prefixei $\{\epsilon, a, aa, aab, aaba, aabab\}$ és P 6 hosszú prefixének a szuffixei $\{\epsilon, a, ba, aba, baba, ababa, aababa\}$, ezek között a leghosszabb egyező pedig a.

i	1	2	3	4	5	6
P[i]	a	a	b	a	b	a
$\pi[i]$	0	1	0	1	0	1

Tehát $\pi = \{0, 1, 0, 1, 0, 1\}$, amit később mintaillesztéshez lehet használni.

2. A Knuth-Morris-Pratt algoritmussal keressük meg az `x='abababacaad'` szövegben a `P='ababac'` mintát.

1. iteráció (illeszkedést találtunk):
$x = abababacaad$
$P = a$
2. iteráció (illeszkedést találtunk):
$x = abababacaad$
$P = ab$

3. iteráció (illeszkedést találtunk):
$x = abababacaad$
$P = aba$

4. iteráció (illeszkedést találtunk):
$x = abababacaad$
$P = abab$

5. iteráció (illeszkedést találtunk):
$x = abababacaad$
$P = ababa$

6. iteráció (illeszkedési hiba):
$x = abababacaad$
$P = ababac$
Számoljuk ki az illeszkedési eltolást:
ababab szuffixei: $ababab, babab, abab, bab, ab, b, \epsilon$
ababa prefixei: $ababa, abab, aba, ab, a, \epsilon$
Leghoszabb prefix, ami egyben szuffix is: *abab*. Illesszük ehhez a szuffixhez a mintát a következő iterációtól.

7/a. iteráció (illeszkedést találtunk):
$x = abababacaad$
$P = \quad abab$

7/b. iteráció (illeszkedést találtunk):
$x = abababacaad$
$P = \quad ababa$

8. iteráció (végső illeszkedést találtunk):
$x = abababacaad$
$P = \quad ababac$

3. Rabin-Karp algoritmussal döntsük el, hogy a `T=3613203214` input kapcsán mely indexeiről kezdődhet a `P=321` mintára való illeszkedés a $h(x) = x \mod 11$ hasítófüggvény használata mellett?
Hint: $T(s_{i+1\ldots i+m}) = (d \times (h(s_{i..i+m-1}) - h(d^{m-1}) \times s_i)) + s_{i+m}) \mod q$
Tekintsük úgy, hogy $d = 10$, tehát a 10-es számrendszerben vagyunk.
A minta hossza, $m = 3$.
A P minta hasított értéke, $h(321) = 321 \mod 11 = 2$.
Kelleni fog nekünk a képletben $h(d^{m-1})$ értéke is, ami nem más, mint $h(10^{3-1}) = 100 \mod 11 = 1$.

Indítsuk el az ablakolást:

0. index: $h(361) = 361 \mod 11 = 9$
Mivel $9 \neq 2$, így nem találtunk illesztést.

1. index: $h(613) = (10 * (h(361) - h(10^2) * 3) + 3) \mod 11 = (10 * (9 - 1 * 3) + 3) \mod 11 = 8$
Mivel $8 \neq 2$, így nem találtunk illesztést.

2. index: $h(132) = (10 * (h(613) - h(10^2) * 6) + 2) \mod 11 = 10 * (8 - 1 * 6) + 2 \mod 11 = 0$

3. index: $h(320) = (10 * (h(132) - h(10^2) * 1) + 0) \mod 11 = 10 * (0 - 1 * 1) + 0 \mod 11 = 1$

4. index: $h(203) = (10 * (h(320) - h(10^2) * 3) + 3) \mod 11 = 10 * (1 - 1 * 3) + 3 \mod 11 = 5$

5. index: $h(032) = (10 * (h(203) - h(10^2) * 2) + 2) \mod 11 = 10 * (5 - 1 * 2) + 2 \mod 11 = 10$

6. index: $h(321) = (10 * (h(032) - h(10^2) * 0) + 1) \mod 11 = 10 * (10 - 1 * 0) + 1 \mod 11 = 2$
Egyezést találtunk a hasított értékeknél, ezért ellenőrizzük le, hogy van-e pontos egyezés P és az adott részlet között. Mivel $321 = 321$, így valós egyezést találtunk!

7. index: $h(214) = (10 * (h(321) - h(10^2) * 3) + 4) \mod 11 = 10 * (2 - 1 * 3) + 4 \mod 11 = 5$

Milyen karakternek kéne a `T=3613203214y` minta y pozícióján álljon, hogy a Rabin-Karp algoritmus tévesen megvizsgálja az illeszkedést a `P=321` mintára a $h(x) = x \mod 11$ hasítófüggvény használata mellett?
P hasított értéke, $h(321) = 2$.
Tehát a kérdés $h(14y)$ értéke. Itt olyan y-t kell keresni, ahol y belépő karakter esetében a képlet eredménye 2, mert ekkor fogunk vizsgálatot végrehajtani. Ezen felül tudjuk, hogy y egy karakter, tehát az értéke a $[0, 9]$ intervallumba kell essen.
A keresett összefüggés: $h(14y) = (10(h(214) - h(10^2) * 2) + y) \mod 11 = 10 * (5 - 1 * 2) + y \mod 11 = 2$.
Így
$10(5 - 2) + y \mod 11 = 2$
$(30 + y) \mod 11 = 2$
$35 \mod 11 = 2$
$y = 5$

4. Moduláris hatványozás segítségével határozzuk meg a $d = 7^{13} \mod 17$ kifejezés értékét?
$13 = 1101$

$d = 1$
$\xrightarrow{1/a} d = (1^2) \mod 17 = 1$
$\xrightarrow{1/b} d = (1 * 7) \mod 17 = 7$ —> *mert a bináris ábrázolás 1. eleme = 1*
$\xrightarrow{2/a} d = (7^2) \mod 17 = 15$
$\xrightarrow{2/b} d = (15 * 7) \mod 17 = 3$ —> *mert a bináris ábrázolás 2. eleme = 1*
$\xrightarrow{3/a} d = (3^2) \mod 17 = 9$
$\xrightarrow{4/a} d = (9^2) \mod 17 = 13$
$\xrightarrow{4/b} d = (13 * 7) \mod 17 = \boxed{6}$ —> *mert a bináris ábrázolás 4. eleme = 1*

10. gyakorlat – Korlátozás és szétválasztás módszere

FOGALMAK

FELADATOK

1. Vegyük az következő hozzárendelési feladatot[2]. Adott n munkás és n elvégzendő feladat. Minden munkás más-más költségen végez el egy feladatot. Rendeljük hozzá az elvégzendő feladatokat úgy a munkásokhoz, hogy azokat a legkisebb összköltséggel végezzék el. Például a

$$C = \begin{matrix} a & b & c & d & \\ \end{matrix} \begin{pmatrix} 6 & \underline{2} & 8 & 7 \\ 2 & 1 & 5 & \underline{4} \\ \underline{1} & 3 & 2 & 5 \\ 4 & 2 & \underline{1} & 3 \end{pmatrix} \begin{matrix} I. \\ II. \\ III. \\ IV. \end{matrix}$$

költségmátrix esetében az a, b, c, d feladatokat rendre a III., I., IV., valamint II. munkás végezze el $1 + 2 + 1 + 4 = 8$ összköltséggel, melyről belátható, hogy egy minimális hozzárendelést eredményez.

Fontos kikötés, hogy minden munkásnak **pontosan** egy feladatot kell elvégezzen, tehát pl. a c és a d feladatok nem kerülhetnek egyidejűleg kiosztásra a IV. munkás számára.

A B&B használata során a tényleges f (össz)költségfüggvényt optimista módon (alulról) becslő g függvényre van szükség. $g \leq f^*$ értékét egy részhozzárendeléshez határozzuk meg mohó módon, vagyis a még ki nem osztott munkákra vonatkozóan átmenetileg tegyük fel, hogy nem kell teljesüljön az egy munkás-egy feladat megkötés, vagyis a ki nem osztott munkák közül egy munkásra több feladat is kioszthtató.

Tehát pl. g kezdeti értéke $g = 1+1+1+3 = 6$ (minden oszlop minimumának összegét véve), azaz akárhogy is rendeljük hozzá a feladatokat a munkásokhoz, az összköltség legalább 6 lesz.

A B&B a feladatot leíró állapotteret a g függvény figyelembe vétele mellett járja be/szűri meg.

[2]A B&B módszeren túl Magyar módszerrel (aka. Kuhn–Munkres algoritmus) is megoldható a probléma $O(n^3)$ időben.