Programozás 2

4.

KERESÉS

Keresések

- A probléma általános megfogalmazása: adott egy N elemű sorozat, keressük meg azt az elemet (határozzuk meg a helyét a sorozatban), mely megfelel egy megadott tulajdonságnak.
- Ha több ilyen van, akkor a keresőalgoritmusok általában az első ilyen elemet találják meg.
- A konkrét megvalósításoknál mi egy adott értéket keresünk.

Lineáris keresés

- Más néven: szekvenciális keresés.
- Elölről kezdve sorra vizsgáljuk a sorozat elemeit.
- A keresés sikertelenségét jelzi, hogy a sorozat végére érve nem találtuk meg a keresett adatot.

a, Rendezetlen sorozatban

```
algoritmus Szekvenciális_keresés
 változó I, Hely:egész
 változó Adat: ElemTípus
 változó Talált: logikai
   I:=1
   amíg (I<=N) és (A[I]<>Adat) ismétel
     I:=I+1
   avége
   Talált := I<=N
   ha Talált akkor
     Hely:=I
   hvége
algoritmus vége
```

N elemű sorozat esetén minimum 1, maximum N, átlagosan (N+1)/2 ciklusvégrehajtás után találja meg a keresett elemet; vagy N összehasonlítás után derül ki, hogy nincs meg a keresett elem.

b, Rendezett sorozatban

```
algoritmus Rendezett_Szekvenciális_keresés
 változó I, Hely:egész
 változó Adat: ElemTípus
 változó Talált: logikai
   I:=1
   amíg (I<=N) és (A[I]<Adat) ismétel
     I:=I+1
   avége
   Talált := (I<=N) és (A[I]=Adat)
   ha Talált akkor
     Hely:=I
   hvége
algoritmus vége
```

Az előzőhöz képest növeli a hatékonyságot, hogy akkor is leáll a keresés,
ha nagyobb elemre lépünk, mint a keresett érték.
Viszont hátrány, hogy előfeltétel a rendezettség.

c, Keresési gyakoriság szerint

- Rendezhető a sorozat keresési gyakoriság szerint is (*konyhapolc elv*).
- Ekkor az **a***,* algoritmus átlagosan kevesebb, mint (N+1)/2 ciklusvégrehajtás után találja meg a keresett elemet.
- Gyakoriság szerint rendezhetünk:
  - Létrehozunk egy párhuzamos számlálótömböt, és időnként ez alapján csökkenő sorrendbe rendezzük az adattömböt.
  - Önszerveződő struktúra: valahányszor sikeresen megtaláltunk egy elemet, az eredetileg előtte lévőket eggyel hátrébb léptetjük, majd azt a sorozat legelejére helyezzük.

d, Ütközős (strázsás) keresés

- A kereső ciklus összetett feltételét egyszerűsítjük le.
- A sorozat végére felvesszük a keresett adatot, így felesleges vizsgálni, hogy túlhaladtunk-e a sorozaton (I<=N), hiszen mindenképpen megtaláljuk a keresett elemet.
- Főképp nagy elemszámú sorozatoknál hatéko-nyabb az egyszerű lineáris keresésnél, mivel a ciklusfeltétel kiértékelése lényegesen rövidebb lesz.

```
algoritmus Strázsás_Szekvenciális_keresés
 változó I, Hely:egész
 változó Adat: ElemTípus
 változó Talált: logikai
   A[N+1]:=Adat
   I:=1
   amíg A[I]<>Adat ismétel
     I:=I+1
   avége
   Talált := I<=N
   ha Talált akkor
     Hely:=I
   hvége
algoritmus vége
```

Bináris keresés

- Más néven: felezéses keresés.
- Csak **rendezett** sorozaton valósítható meg.
- Algoritmus:
  - Meghatározzuk a középső elemet, majd megvizsgáljuk ez-e a keresett.
  - Ha nem, akkor ha a keresett kisebb a középsőnél, akkor a sorozat alsó, egyébként a felső részébe folytatjuk a keresést.
- A keresés két esetben fejeződhet be:
  - megtaláltuk a keresett adatot;
  - a részsorozat alsó indexe (E) nagyobb a felső (U), nincs benne a sorozatban a keresett adat.

```
algoritmus Bináris_keresés
 változó Alsó, Felső, Közép, Hely:egész
 változó Adat: ElemTípus
 változó Talált: logikai
 Alsó:=1; Felső:=N; Közép:=(Alsó+Felső)/2
 amíg (Alsó<=Felső) és (A[Közép]<>Adat) ismétel
   ha Adat<A[Közép] akkor
     Felső:=Közép-1
   különben
     Alsó:=Közép+1
   hvége
     Közép:=(Alsó+Felső)/2
 avége
 Talált := Alsó<=Felső
 ha Talált akkor
   Hely:=Közép
 hvége
```

- Szokásos logaritmikus keresésnek is nevezni, mivel 2^k elemű sorozatot k lépésben vizsgálhatunk végig.
- Tehát egy n elemű sorozatnál átlagosan $log_2 n$ hasonlítás után találja meg az elemet, vagy derül ki, hogy nincs ilyen.
- Ez egymillió elem esetén kb. 20 lépést jelent, tehát igen látványos a különbség a lineáris módszerhez képest.
- Probléma lehet a rendezettség fenntartása mellett az, hogy nem minden adatszerkezetnél címezhető meg a középső elem (pl. láncolt lista).

Bár nem olyan hatékony, de kínálja magát a rekurzív megoldás:

```
eljárás BinKeres(Alsó, Felső: egész)
  változó Közép, Hely:egész
  ha Alsó<=Felső akkor
    Közép:=(Alsó+Felső)/2
    elágazás
      amikor Adat<A[Közép]:
        BinKeres(Alsó,Közép-1)
      amikor Adat>A[Közép]:
        BinKeres(Közép+1,Felső)
      különben
        Talált:=Igaz
        Hely:=Közép
    evége
  különben
    Talalt:=Hamis
  hvége
eljárás vége
```

ABSZTRAKT ADATSZERKEZETEK

Verem

- A verem (angolul: *stack*) homogén adatelemek olyan sorozata, amelyen két művelet értelmezett:
  - új elem elhelyezése a verem tetejére (*push*)
  - elem kivétele a verem tetejéről (*pop*).
- Működése LIFO (Last In First Out) elvű.

Sor

- A sor (angolul: *queue*) homogén adatelemek olyan sorozata, amelyen két művelet értelmezett:
  - új elem elhelyezése a sor végére (*put*)
  - elem kivétele a sor elejéről (*get*).
- Működése FIFO (First In First Out) elvű. Mindazon feladatok megoldására alkalmas, ahol sorban állást kell megvalósítanunk.

Láncolt lista

- Gazdaságos memóriafoglalás, egyszerű karbantartási műveletek (törlés, beszúrás).
- Szekvenciális adatszerkezet.
- *Listafej*: a lista első elemére mutat.
- *Végjel (null)*: speciális mutató érték, az utolsó elem mutatórészében állva jelzi a lánc végét.

Egy listaelem szerkezete

```
típus ListaElem: rekord(
        Érték: ÉrtékTípus;
        Köv: MutatóTípus
//mely a ListaElem-re képes mutatni
                  )
```

Strázsás (ütközős) lista

- A lista végére egy üres listaelemet (veg) láncolunk.
- Ezzel és más megoldásokkal a következő kényelmetlenségeket küszöböljük ki:
  - a lista végén végzett műveletekhez nem kell végigfutni a listán;
  - egy elem törlésekor elég a törlendő elem címét ismerni, nincs szükség a megelőző elemre;
  - egy adott elem elé tudunk beszúrni;
  - beszúrás, törlés hasonló a lista elején, közepén, végén.

További láncoltlista változatok

- rendezett láncolt lista: nem utólag rendezzük a listát, hanem az új elem felvitele a rendezettség megtartása mellett történik;
- ciklikusan láncolt lista: az utolsó elem mutatója az első elemre mutat;
- két irányban láncolt lista: egy listaelemben két mutató;
- többszörösen láncolt lista: több láncolat mentén is bejárható, azaz több szempont szerint is rendezett, egy elemben több mutató.

Fa

- A fa az adatok hierarchikus kapcsolatának ábrázolására alkalmas adatszerkezet (pl. háttértárolók könyvtárszerkezete).
- Rendelkezik egy kitüntetett kezdőponttal, és e kezdőpontból kiindulva minden adatelemhez tetszőleges számú új elem kapcsolódhat.
- A kapcsolódó elemek a fa más részeihez nem kapcsolódhatnak.

Elnevezések

- *gyökér*: a kezdőelem;
- *csomópontok*: adatelemek;
- *élek*: egymással közvetlen kapcsolatban lévő csomópontokat kötik össze, irányuk mindig a gyökértől a távolabbi csomópont felé mutat;
- *levél*: azon csomópontok, amelyekből nem vezet tovább él;
- *út*: egy csomópontból egy másikba vezető élsorozat;
- *szülő*: ha A csomópontból él mutat B csomópontba, akkor A szülője B-nek;
- *gyerek*: ha A csomópontból él mutat B csomópontba, akkor B gyereke A-nak;
- *testvér*: egy szülőhöz tartozó csomópontok.

Bináris fa

- *Bináris fa*: minden csomópontnak legfeljebb két gyereke van.
- *Szigorúan bináris fa*: a levélelemek kivételé-vel minden csomópontnak pontosan két gyereke van.

A bináris fa megvalósítása

- Tárbeli megvalósítása a láncolt listában alkalmazott adatszerkezet bővítésével: minden elemnek két rákövetkezője lehet, ezért két mutatót alkalmazunk a csomópontokban, az egyik a baloldali, a másik a jobboldali részfa gyökerére mutat.
- Ha valamely mutató értéke a VégJel (null), akkor ebben az irányban nincs folytatása a fának.
- A levélelemek mindkét mutatója VégJel.

Egy csomópont szerkezete

```
típus BinFaElem: rekord(
      Érték: ÉrtékTípus;
    Bal, Jobb: MutatóTípus
  //mely a BinFaElem-re képes mutatni
             )
```

Minden fához tartozik egy változó: `Gyökér`, mely a fa kezdőpontjára, a gyökérelemre mutat.
Ez a mutató fogja azonosítani a fát.

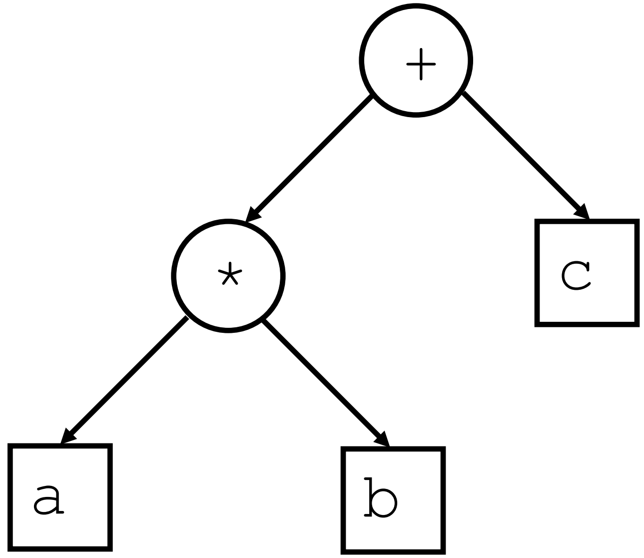
Bináris fa bejárása

Bejárásnak nevezzük azt a folyamatot, amikor a fa minden elemét pontosan egyszer érintve feldolgozzuk. Erre a gyakorlatban három, a rekurzív definíciókra támaszkodó módszer terjedt el:

• *preorder* bejárás:
  – gyökérelem feldolgozása;
  – baloldali részfa preorder bejárása;
  – jobboldali részfa preorder bejárása;

• *inorder* bejárás:
  – baloldali részfa inorder bejárása;
  – gyökérelem feldolgozása;
  – jobboldali részfa inorder bejárása;

• *postorder* bejárás:
  – baloldali részfa postorder bejárása;
  – jobboldali részfa postorder bejárása;
  – gyökérelem feldolgozása;



Példa: Algebrai kifejezések (pl. a*b+c) ábrázolása, bejárása.

Az ilyen típusú felírásban a fa levelei az operandusokat, a többi csomópont pedig az operátorokat tartalmazza.
A háromféle bejárás szerint feldolgozva az elemeket, az algebrai kifejezések ismert formáit kapjuk:

1. preorder bejárással a prefix alakot: +*abc;
2. inorder bejárással az infix alakot: a*b+c;
3. postorder bejárással a postfix alakot: ab*c+;

A bejáró rekurzív algoritmusok (Elem[P]-vel jelöljük a P által mutatott csomópontot).

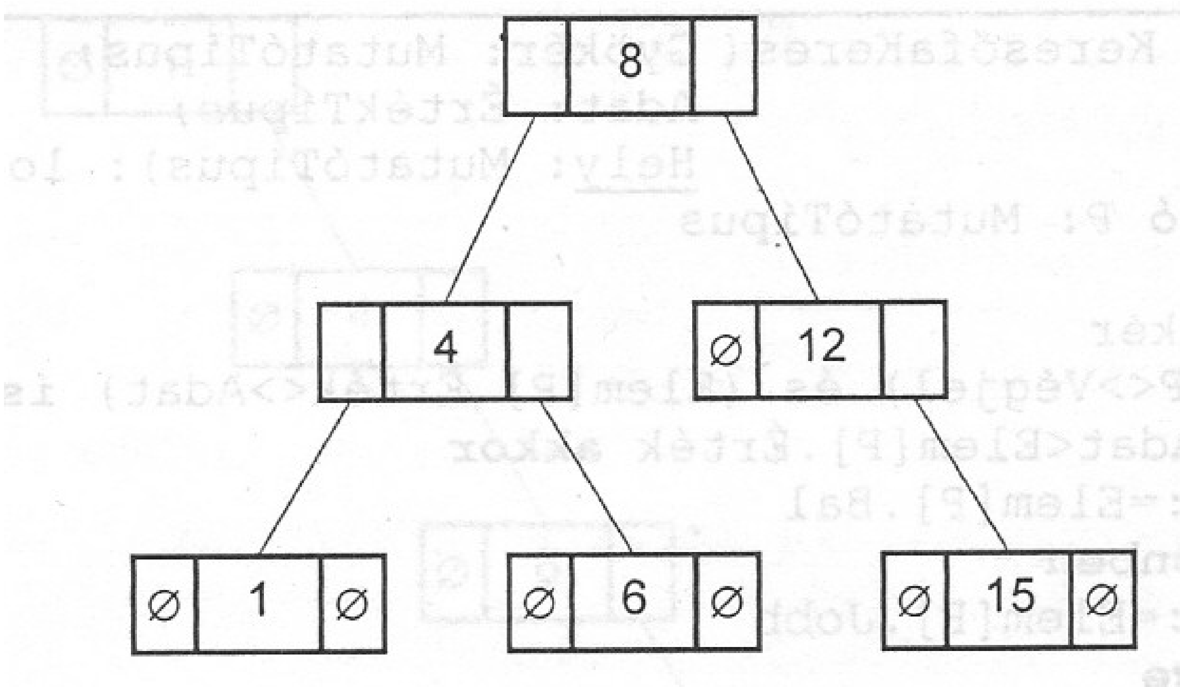
```
eljárás BinFaPreorder(P: MutatóTípus)
  ha P<>VégJel akkor
    Elem[P].Érték feldolgozása
    BinFaPreorder(Elem[P].Bal)
    BinFaPreorder(Elem[P].Jobb)
  hvége
eljárás vége
```

```
eljárás BinFaInorder(P: MutatóTípus)
    ha P<>VégJel akkor
        BinFaInorder(Elem[P].Bal)
        Elem[P].Érték feldolgozása
        BinFaInorder(Elem[P].Jobb)
    hvége
eljárás vége

eljárás BinFaPostorder(P: MutatóTípus)
    ha P<>VégJel akkor
        BinFaPostorder(Elem[P].Bal)
        BinFaPostorder(Elem[P].Jobb)
        Elem[P].Érték feldolgozása
    hvége
eljárás vége
```

Keresőfa

- Gyors keresési módszerekben illetve az adattömörítésben alkalmazzák.
- A keresőfa egy olyan bináris fa, amelynek minden csomó-pontjára igaz, hogy a benne tárolt érték:
  - nagyobb, mint a baloldali részfájában tárolt bármely érték;
  - kisebb, mint a jobboldali részfájában tárolt bármely érték.
- Ha az adatok ismétlődését is megengedjük, akkor a feltételek enyhíthetők kisebb vagy egyenlőre illetve nagyobb vagy egyenlőre.

Az ábrán megfigyelhető, hogy inorder bejárás szerint az eredmény egy rendezett számsorozat lesz.



Keresés a keresőfában

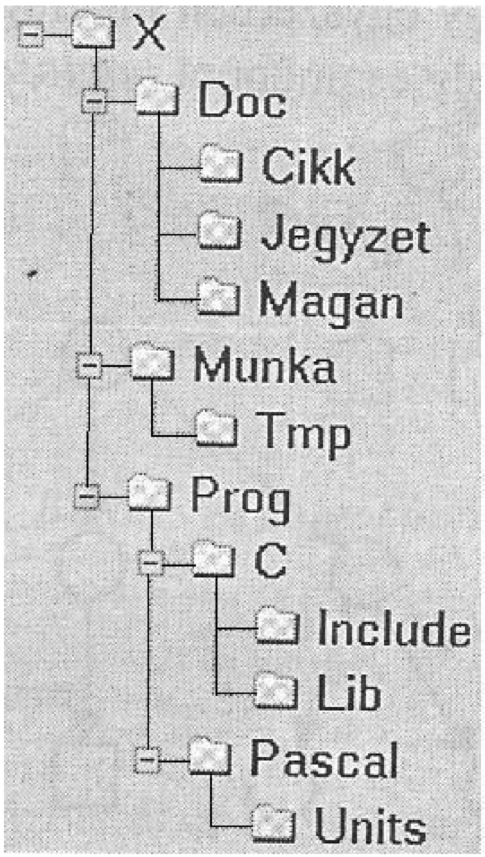
A keresés során a keresőfa definícióját (rendezettség) használjuk ki.

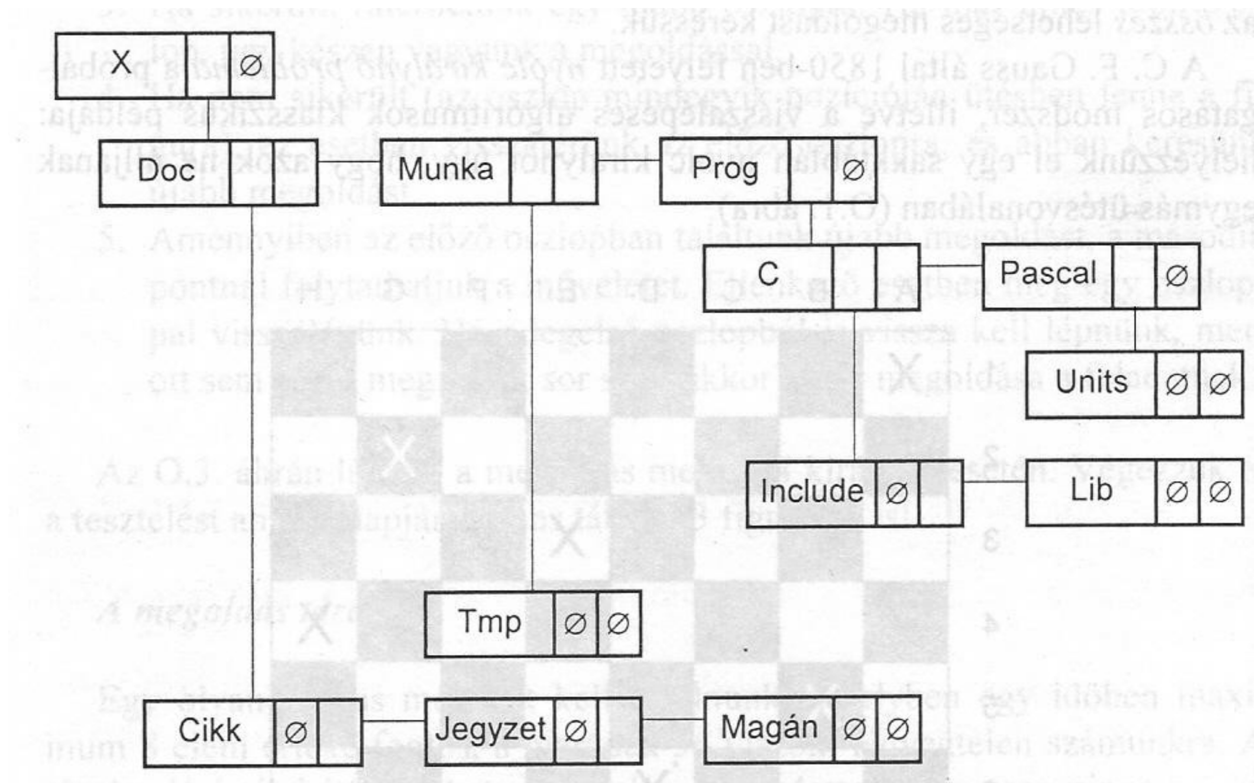
```
algoritmus KeresőfaKeres
 változó P: MutatóTípus;

 P:=Gyökér
 amíg(P<>Végjel) és (Elem[P].Érték<>Adat) ismétel
   ha Adat< Elem[P].Érték akkor
     P:=Elem[P].Bal
   különben
     P:=Elem[P].Jobb
   hvége
 avége
 Talált:=P<>Végjel
 ha Talált akkor
   Hely:=P
 hvége
algoritmus vége
```

Általános fák megvalósítása

- Egy elemnek tetszőleges számú gyereke (vagyis tetszőleges számú testvére lehet).
- Alkalmazhatunk láncolt listát: fűzzük listába a testvér csomópontokat, így a szülőtől csak egyetlen mutatónak kell megcímeznie ezt a listát. Természetesen a szülő maga is része egy hasonló listának, ami a testvéreivel kapcsolja össze.
- Minden csomópontban két mutatóra lesz szükség: az egyik mutat a legelső gyerekre, a másik a következő testvérre. Így az általános fák kezelését visszavezettük a bináris fákéra.

Az első ábrán látható könyvtárszerkezete ily módon ábrázoljuk a memóriában:





Felhasznált anyagok

- Dévényi Károly (SZTE): Programozás alapjai
- Simon Gyula (PE): A programozás alapjai
- Pohl László (BME): A programozás alapjai
- B. W. Kernighan - D. M. Ritchie: A C programozási nyelv