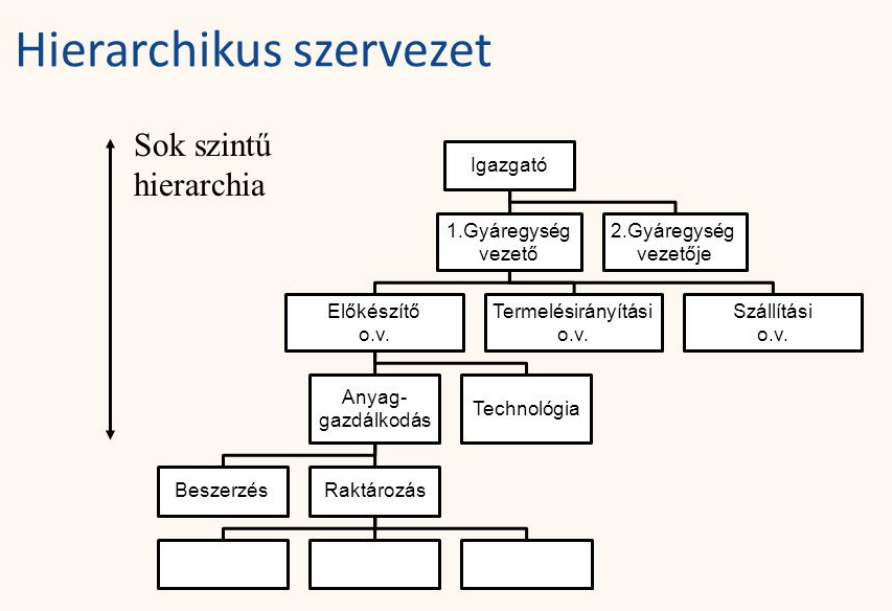
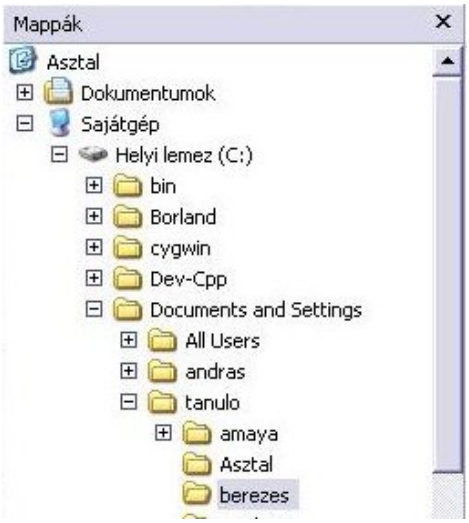
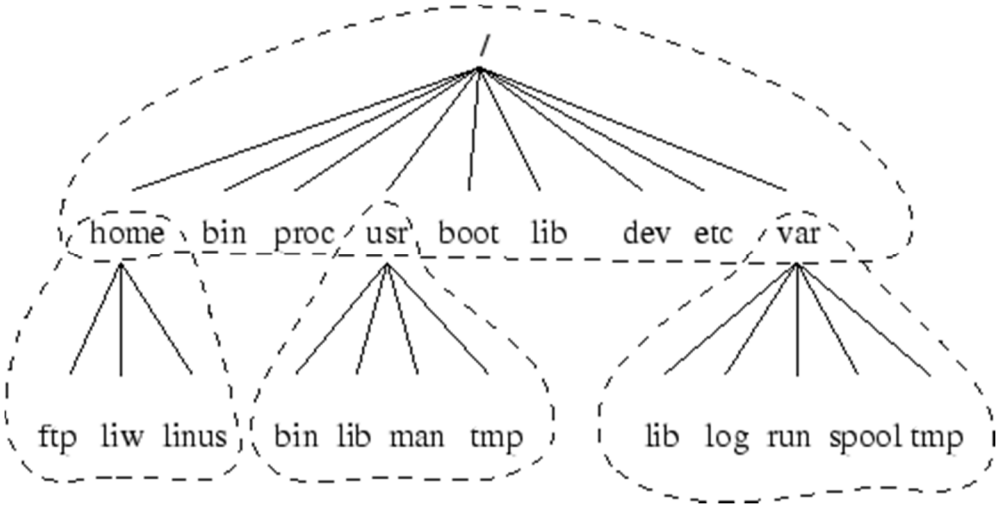
Fa

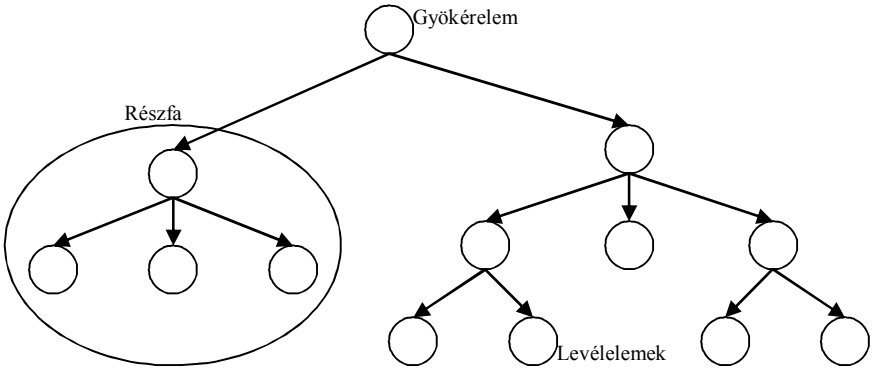
A fa egy hierarchikus adatszerkezet, amelyben egy elemnek akár több rákövetkezője, de legfeljebb egy megelőzője lehet.





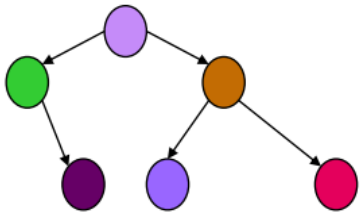


- Minden elemeknek több rákövetkezője (gyereke, **leszármazottja**) lehet.
- A **gyökérelem** a fa azon eleme, amelynek nincs megelőzője (szülője). Az üres fa kivételével minden fának van egy gyökéreleme. A gyökérelemen kívül az összes elemnek pontosan egy megelőzője van.
- Minden elem pontosan **egy úton** érhető el a gyökérelemtől.
- **Út**: az adatelemek olyan sorozata, ahol az egymást követő elemek egymásnak rákövetkezői (leszármazottjai). Az út hosszán az adott útban található élek számát értjük.
- **Levél**: a fa azon elemei, amelyeknek nincs rákövetkezőjük.
- **Belső csúcs**: a fa nem levél elemei.
- A fákkal kapcsolatban beszélhetünk **szint**ekről, egy **elem szintje** megegyezik a gyökérelemtől vett távolságával. A nulladik szinten a gyökérelem van, az első szinten a gyökérelem rákövetkezői, stb.
- A maximális szintszámot a **fa magasságának vagy mélységének** nevezzük.
- Minden közbenső elem egy részfa gyökereként tekinthető, így a fa részfákra bontható. Tehát a fát lehet **rekurzívan is definiálni**, azaz a fa áll egy gyökérelemből és a leszármazottai részfáiból.

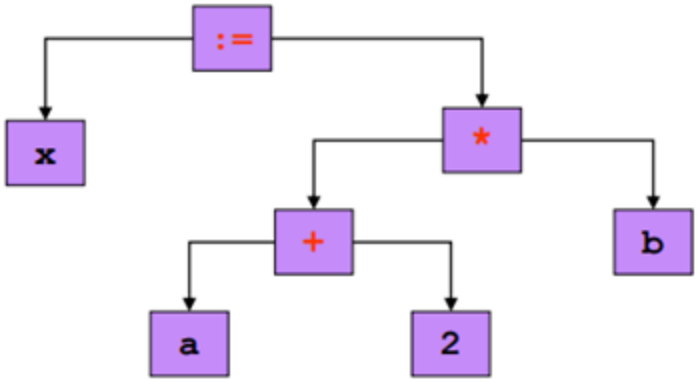


Bináris fa

Olyan fa, ahol minden csúcsnak legfeljebb két leszármazottja lehet.

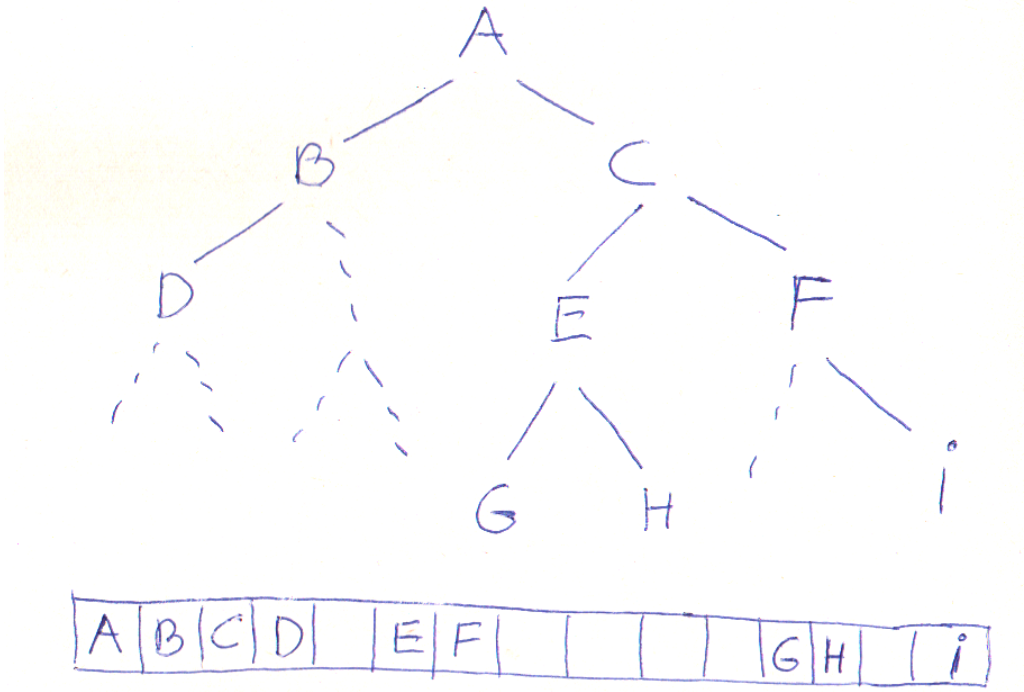


x:=(a+2)*b



Tömbös ábrázolás

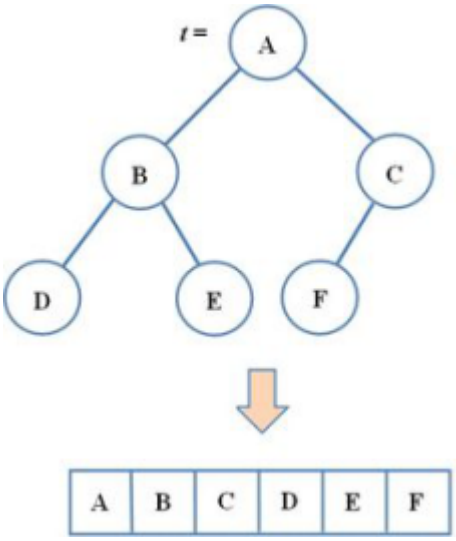
Az elemeket szintfolytonosan elhelyezzük egy tömbbe



Ha az 1-es indextől helyezzük el a fát, akkor a szülő-gyerek és a gyerek-szülő kapcsolatok az alábbi egyszerű képletek által kezelhetők maradnak:

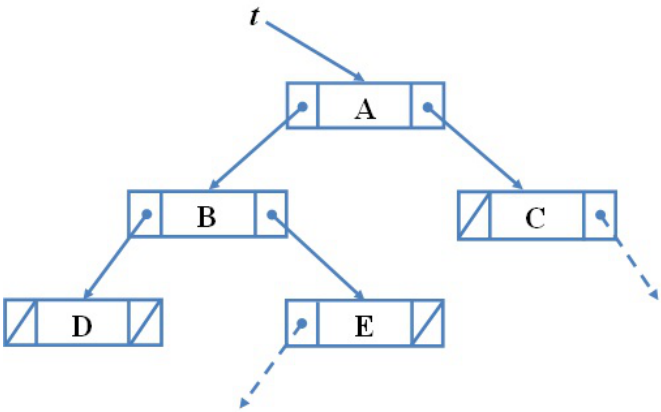
- bal(i) = 2*i
- jobb((i) = 2*i + 1
- szülő(i) = i/2 (egészrész)

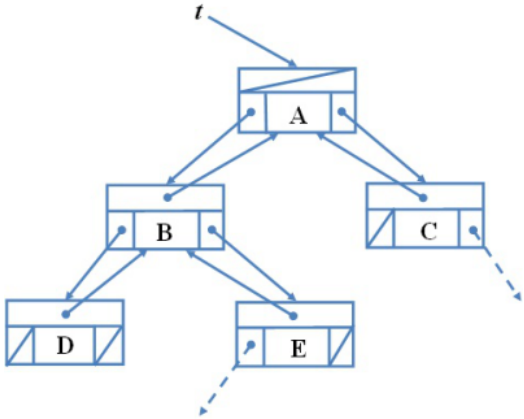
A hátránya, hogy tele lesz a tömb lyukakkal, azaz a nem létező csúcsoknak is foglalunk helyet. Ezt az ábrázolást kizárólag a majdnem teljes, balra tömörített bináris fáknál használjuk.(lásd prioritásos sor-kupac) A majdnem teljes balra tömörített bináris fa sajátossága az, hogy ha az elemeit szintfolytonosan érintjük, akkor egyetlen hiányzó csúcsot sem fedezünk fel, mert – egy bizonyos határpontig a levelek szintjén – minden csúcs szerepel a fában. Ha a csúcsokat szintfolytonos sorrendben egy tömbbe helyezzük, akkor hiánymentes kitöltéshez jutunk.



Láncolt ábrázolás

A bináris fa láncolt ábrázolásában mutatók/indexek valósítják meg a rákövetkezési és megelőzési relációkat.





<u>Mutatókkal megvalósítva:</u>

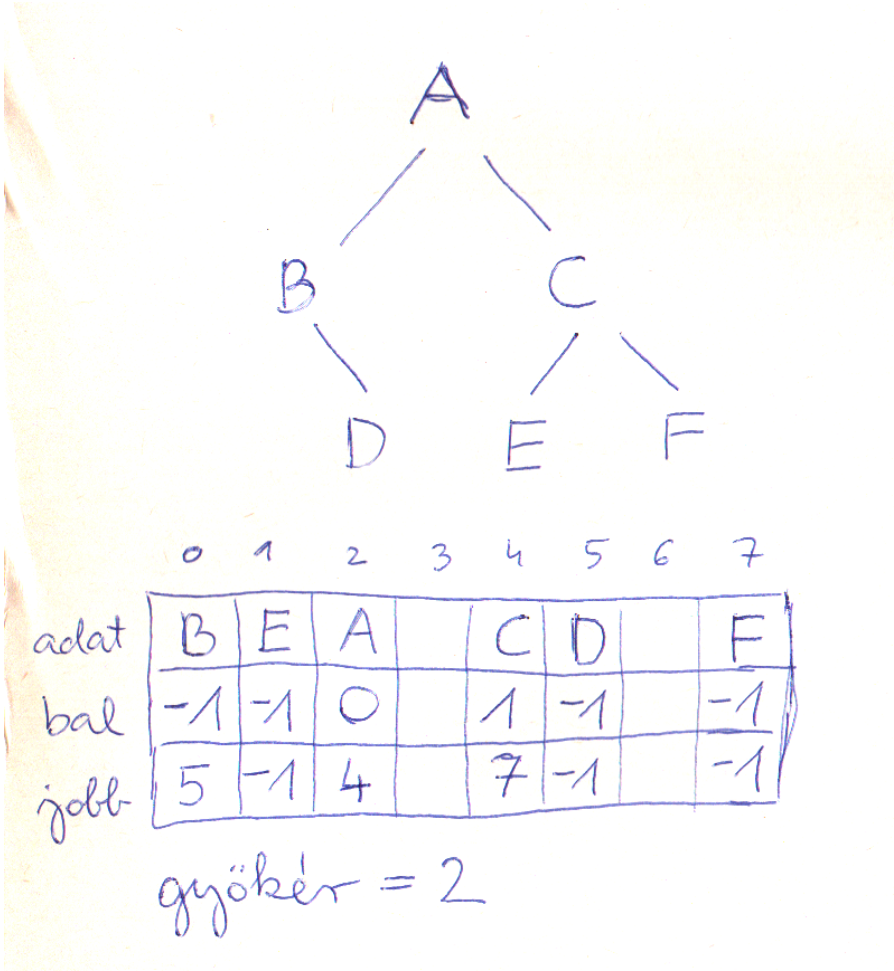
C++:

```
struct node{
    int adat;
    node *bal, *jobb;

    node(int adat){
        this->adat = adat;
        this->jobb = NULL;
        this->bal = NULL;
    }
};
```

C#: Rekurzív definíció.

```
class binfa
{
    public int adat;
    public binfa bal, jobb;

    public binfa(int adat) { this.adat = adat; }
}
```

<u>Tömbben láncolva megvalósítva</u>




```
struct node{
      int adat,bal,jobb;
};

vector<node> fa;
int gyoker;
```

A bináris fa bejárásai

A fák jellegzetes algoritmusai a bejárások. A bináris fa rekurzív jellegéhez illeszkedően rekurzív módon könnyű definiálni a három féle bejárást, amelyek abban különböznek, hogy a gyökérelemet mikor érintjük a bejárás során.

Preorder bejárás

Ha a gyökérelemet először, a bal és jobb oldali részfa bejárásai előtt közvetlenül érintjük, akkor preorder bejárásról beszélünk.

```
void preorder(int i)
{
    if(i!=-1)
    {
       cout<<fa[i].adat<<endl;
       preorder(fa[i].bal);
       preorder(fa[i].jobb);
    }
}
```

```
void preorder(node* i)
{
    if(i!=NULL)
    {
        cout<<i->adat<<endl;
        preorder(i->bal);
        preorder(i->jobb);
    }
}
```

Inorder bejárás

Amikor a gyökérelemet a bal és jobb oldali részfa bejárásai között érjük el, akkor inorder bejárást valósítunk meg.

```
void inorder(int i)
{
    if(i!=-1)
    {
       inorder(fa[i].bal);
       cout<<fa[i].adat<<endl;
       inorder(fa[i].jobb);
    }
}
```

```
void inorder(node* i)
{
    if(i!=NULL)
    {
        inorder(i->bal);
        cout<<i->adat<<endl;
        inorder(i->jobb);
    }
}
```

Postorder bejárás

Ha a gyökérelemhez a bal és jobb oldali részfa bejárásai után jutunk el, akkor postorder bejárást valósítunk meg.

```
void postorder(int i)
{
    if(i!=-1)
    {
       postorder(fa[i].bal);
       postorder(fa[i].jobb);
       cout<<fa[i].adat<<endl;
    }
}
```

```
void postorder(node* i)
{
    if(i!=NULL)
    {
        postorder(i->bal);
        postorder(i->jobb);
        cout<<i->adat<<endl;
    }
}
```

Szintfolytonos bejárás

Érdekes feladat egy bináris fa szintfolytonos bejárásának a megvalósítása. A csúcsok szintfolytonos sorrendű elérése ugyanis – szemléletesen szólva – ortogonális arra az irányra, amelyet az élek meghatároznak. Szintfolytonos bejárást készíthetünk egy sor adatszerkezet alkalmazásával. Az algoritmust néhány szabály alapján ezután már könnyen kialakíthatjuk.

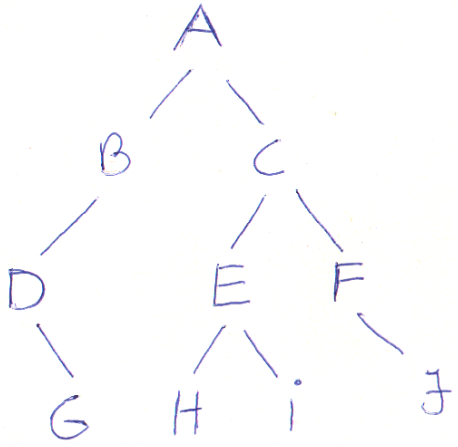
(1) A bejárás során a gyökérelem legyen az első érintett csúcs.
(2) Amint egy csúcsot elértünk, akkor a gyerekeit helyezzük el a sorban.
(3) A bejárás következő elemét vegyük a sorból.
(4) Az eljárás addig folytatódik, ameddig a sor ki nem ürül.

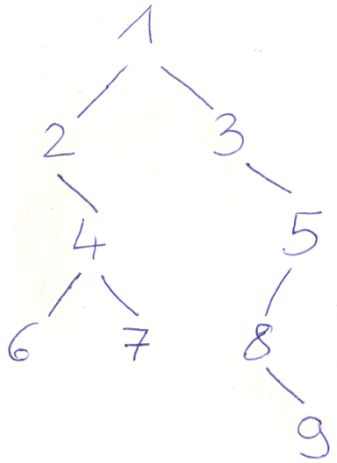
```
void szintfolytonos(node* i)
{
    if(i!=NULL)
    {
        queue<node*> q;
        q.push(i);

        while(!q.empty())
        {
            node* p = q.front();
            q.pop();
            cout<<p<<endl;

            if(p->bal!=NULL)
                q.push(p->bal);
            if(p->jobb!=NULL)
                q.push(p->jobb);
        }
    }
}
```

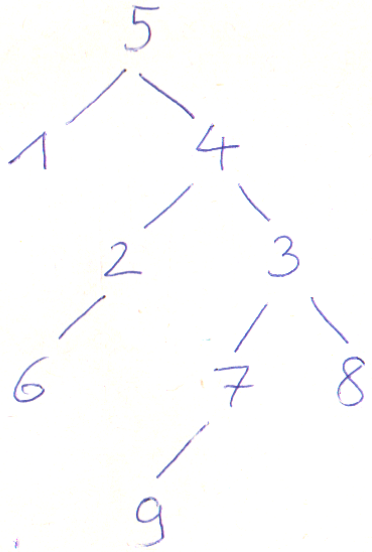
Mind a négy fabejárásra teljesül az, hogy műveletigényük a bináris fa csúcsainak számában lineáris, ugyanis a bináris fa minden csúcsát pontosan egyszer érintik.

Feladatok bináris fára

1. Írja le az alábbi bináris fák preorder, inorder, postorder és szintfolytonos bejárásinak csúcssorrendjét.







2. Az előző feladatbeli fákat tömbben láncolva szeretnénk ábrázolni. Megadtuk a fa pontjaiban lévő adatokat egy tömbben. Töltsd ki a bal és jobb indexek értékeit, hogy a fenti szerkezetű fákat megkapjuk.

	0	1	2	3	4	5	6	7	8	9
adat	A	B	C	D	E	F	G	H	i	J
bal										
jobb										

	0	1	2	3	4	5	6	7	8
adat	2	4	6	1	7	8	3	9	5
bal									
jobb									

3. Bináris fákat tömbben láncolva ábrázoltunk, megadtuk a fa pontjaiban lévő adatokat, a bal- és jobb gyerek indexeit egy tömbben. Rajzolja fel a fát!

	0	1	2	3	4	5	6	7	8	9
adat	B	A	E	C	D	F	I	G	H	J
bal	1	3	-1	6	7	-1	-1	-1	-1	-1
jobb	2	4	5	-1	8	9	-1	-1	-1	-1

gyökér = $\emptyset$

	0	1	2	3	4	5	6	7
adat	2	3	5	6	7	8	1	4
bal	3	4	6	-1	-1	-1	-1	0
jobb	-1	5	7	-1	-1	-1	-1	1

gyökér = ?

	0	1	2	3	4	5	6	7	8
adat	3	4	7	1	9	8	2	5	6
bal	-1	8	-1	6	-1	-1	-1	5	-1
jobb	7	2	-1	0	-1	4	1	-1	-1

gyökér = ?

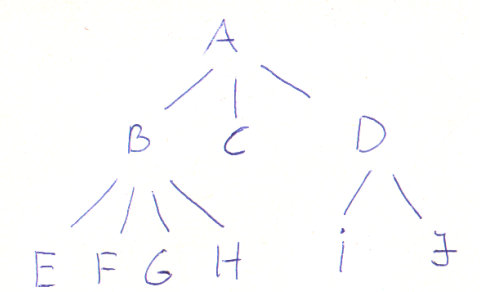
4. Készítsünk olyan algoritmust, amelyik megszámolja egy bináris fában a
   4.1. csúcsokat
   4.2. leveleket.
5. Határozzuk meg egy bináris fa mélységét, azaz a maximális szintszámot, amelyen levél található. (Az egyetlen pontból álló fára ez a szám 0, az üres fára pedig -1.)
6. Adjunk olyan eljárást, amelyik megszámolja egy bináris fa adott szintjén lévő
   6.1. csúcsokat
   6.2. leveleket.
7. Adjuk meg egy bináris fa csúcsaiban (belső pontjaiban, leveleiben) szereplő értékek maximumát.
8. Határozzuk meg azt a legkisebb magasságot, amelyben egy megadott bináris fa már tartalmaz levelet. (Üres fára ez az érték legyen -1, egy pontból álló fára pedig 0.)
9. Döntsük el egy bináris fáról, hogy piramis-e. (Minden belső csúcsban kisebb vagy egyenlő érték található, mint a gyerekeiben.)
10. Adott két bináris fa. Döntsük el, hogy azonos-e a szerkezetük.

11. Adott egy kifejezésfa. Írjuk ki a kifejezés teljesen zárójelezett alakját. (Az operandusok nincsenek zárójelben.)
12. Adott egy kifejezés lengyel formája egy szekvenciális input sorozatban. Építsük fel a kifejezésfát.
13. Adott egy teljes bináris fa szintfolytonos elhelyezése egy tömbben. Építsük fel a fát.
14. Barkóba játék. Egy fájlba tároljuk a kérdések fáját. A program a fájlból betölti a fát és fa kérdéseit feltéve megpróbálja kitalálni a felhasználó által kigondolt feladványt.
15. Egy bináris fában csupa különböző érték található. Adott a fa preorder és inorder bejárása egy-egy tömbben. Építsük fel a fát.

```
int pre[] = {15,7,3,10,5,1,8,9,20,12};
int in[] = {10,3,7,1,5,8,15,20,12,9};
```

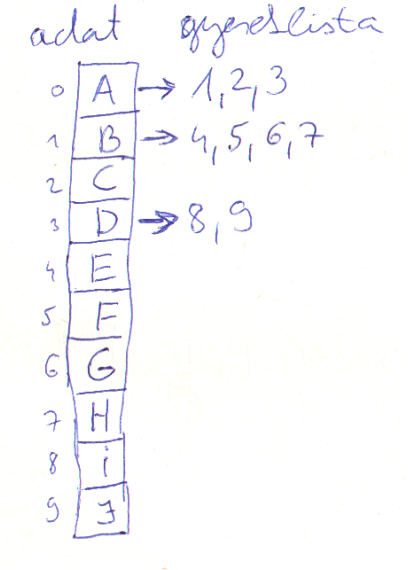
Nem bináris fák

Olyan fa, ahol minden csúcsnak tetszőleges számú leszármazottja lehet. Kizárólag láncolt ábrázolást használunk, a leszármazottakra való hivatkozásokat listákban tárolhatjuk.



Tömbben láncolva megvalósítva

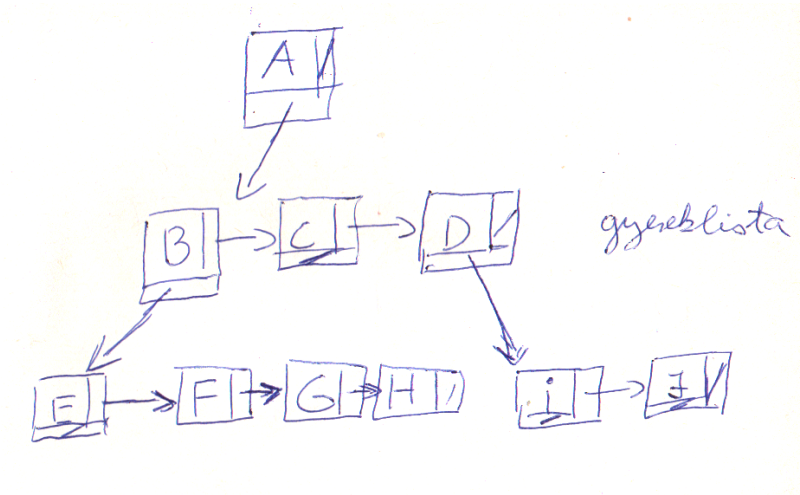
Egy tömbben tároljuk a csúcsok adatait és a leszármazottak indexeinek a listáját (vektorát).




```
struct node
{
    string adat;
    vector<int> gyerek;
};

vector<node> fa;
int gyoker;
```

Mutatókkal megvalósítva:

Egy tömbben tároljuk a csúcsok adatait és a leszármazottakra mutatók listáját (vektorát).



C++:

```
struct node
{
    string adat;
    vector<node*> gyerek;
};

node* gyoker;
```

C#:

```
class fa
{
     public string adat;
     public List<fa> gyerek = new List<fa>();
}
```

A fa bejárása:

```
void bejar(int i)
{
    cout<<fa[i].adat<<endl;
    for(int j=0; j<fa[i].gyerek.size(); ++j)
        bejar(fa[i].gyerek[j]);
}
```

```
void bejar(node* i)
{
    cout<<i->adat<<endl;
    for(int j=0; j<i->gyerek.size(); ++j)
        bejar(i->gyerek[j]);
}
```

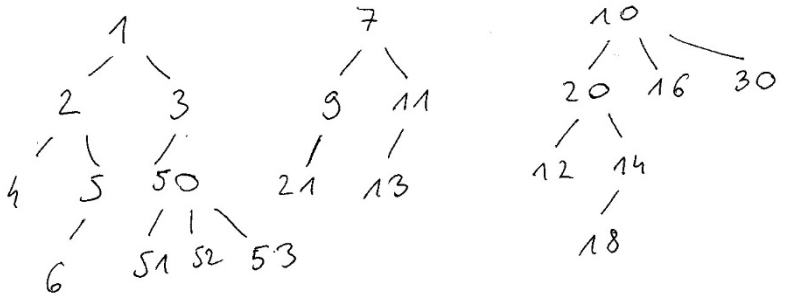
Alapfeladatok/tételek R-áris fákra, erdőkre:

A fák csúcsai különböző egész számokat tartalmaznak. Olvassa be a fák adatait az input.txt fájlból. A fájl első sorában a csúcsok n száma található, majd a következő sorban a csúcsokban lévő számok következnek (n db szám). Ez követő sorokban adjuk meg a fa felépítését. Minden sorban két szám szerepel az első a szülő a második a gyereke.

1) Határozza meg a fák számát. *int FakSzama()*
2) Válogassa ki a fák gyökereit. *void FakGyokerei()*
3) Írja meg egy fa mélységét meghatározó függvényt. *int Melyseg(int gyoker)*
4) Írja meg egy fa csúcsainak a számát meghatározó függvényt. *int Csucsszam(int gyoker)*
5) Írja meg egy fa leveleinek a számát meghatározó függvényt. *int Levszam(int gyoker)*
6) Írja meg egy fa leveleinek összegét meghatározó függvényt. *int Osszeg(int gyoker)*
7) Írja meg azt a függvényt, ami eldönti, hogy egy fa tartalmaz-e páros számot. *bool Eldontes(int gyoker)*
8) Írja meg azt a függvényt, amely kiválogatja egy fában található páros számokat. void *Kivalogatas(int gyoker)*
9) Írja meg azt a függvényt, amely megszámolja egy fában található páros számokat. *int Megszamolas(int gyoker)*
10) Írja meg azt a függvényt, amely megadja egy fában található legnagyobb számot. *int Max(int gyoker)*

Input:

```
22
1 2 3 4 5 6 7 9 11 21 13 10 20 16 30 12 14 18 50 51 52 53
1 2
1 3
2 4
2 5
3 50
5 6
50 51
50 52
50 53
7 9
7 11
9 21
11 13
10 20
10 16
10 30
20 12
20 14
14 18
```



```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#include <iostream>
#include <string>
using namespace std;


//Listázzuk ki a könyvtárfát egy adott ponttól kezdve.
//Annyi TAB-ot rakjunk a fájl/könyvtárnév elé, amilyen mélyen van a fában.

void reklist(string konyvtar,int szint)
{
    DIR* dir = opendir (konyvtar.c_str());
    if (dir!=NULL)
    {
       struct dirent *akt = readdir(dir);
       while(akt!=NULL)
       {
         string nev = akt->d_name;
         if(nev!="." && nev!="..")
         {
            for(int i=0; i<szint; ++i)
               cout<<'\t';
                cout<<nev<<endl;
            reklist(konyvtar+"\\"+nev,szint+1);
         }
             akt=readdir(dir);
       }
       closedir (dir);
     }
}



int main (int argc, char **argv)
{
    reklist("c:\\SZLG\\megoldasok\\adatbazis",0);
}
```