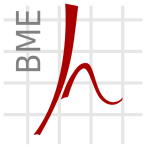
Bináris fák. Függvénymutatók

A programozás alapjai I.



Hálózati Rendszerek és Szolgáltatások Tanszék
Farkas Balázs, Fiala Péter, Vitéz András, Zsóka Zoltán

2020. november 23.

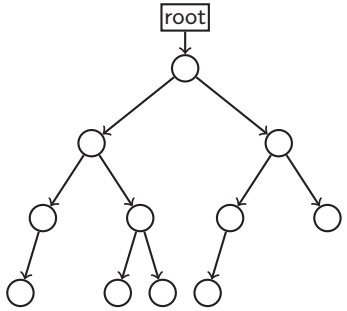
Tartalom

1. fejezet

Bináris fák

Fák

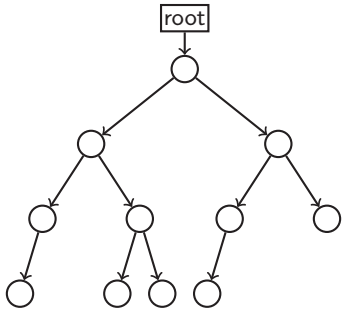
$K = 1$ (láncolt lista)



$K = 2$ (bináris fa)

- Körmentes gráf
- Minden csomópontba egy él fut be
- K-ágú fa: minden csomópontból legfeljebb K él fut ki

Bináris fák

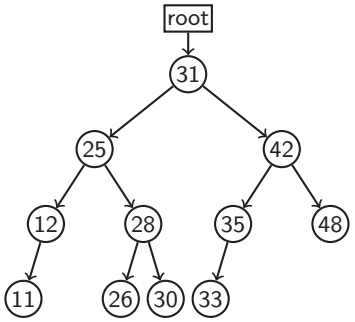


- A bináris fa adatszerkezetének deklarációja

```
typedef struct tree {
  int data;
  struct tree *left, *right;
} tree_elem, *tree_ptr;
```

- Szokványos egyben deklarálni a mutató típust is

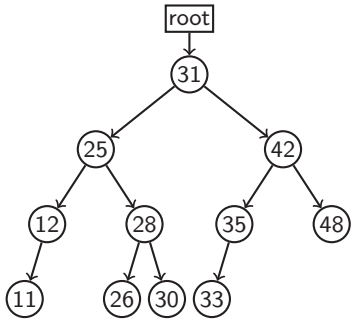
Bináris rendezőfa



31 25 28 42 12 35 11 48 30 33 26

- Elem bal oldali részfájában csak nála kisebb elemek vannak
- Elem jobb oldali részfájában csak nála nagyobb elemek vannak
- A fa struktúrája az elemek érkezési sorrendjétől függ!

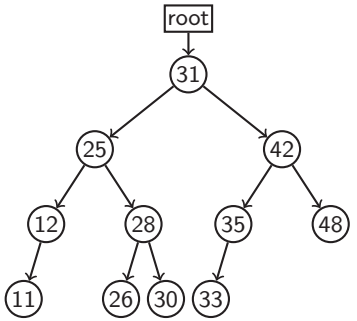
Elem megkeresése a fában




```
tree_ptr find(tree_ptr root,
              int data)
{
  while (root != NULL &&
    data != root->data)
  {
    if (data < root->data)
      root = root->left;
    else
      root = root->right;
  }
  return root;
}
```

- Ez még nem rekurzió!
- d mély fában max. d lépés alatt megvan az eredmény
- Kiegyensúlyozott fában n elem közül $\approx \log_2 n$ lépés!

Bejárás – inorder




```
void inorder(tree_ptr root)
{
  if (root == NULL)
    return;
  inorder(root->left);
  printf("%d ", root->data);
  inorder(root->right);
}
```

11 12 25 26 28 30 31 33 35 42 48

- inorder bejárás
  1. bal részfa
  2. gyökérelem
  3. jobb részfa

Ebben a sorrendben nagyság szerinti sorrendben dolgozzuk fel az elemeket

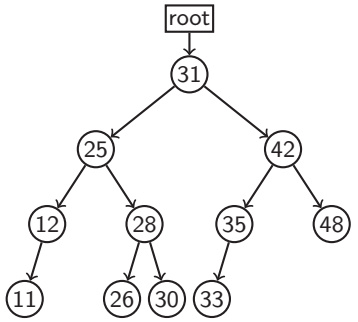
Bejárás – inorder

Másként is szervezhetjük a bejárást

```
1  void inorder(tree_ptr root)
2  {
3    if (root->left != NULL)
4      inorder(root->left);
5    printf("%d ", root->data);
6    if (root->right != NULL)
7      inorder(root->right);
8  }
```

Ebben az esetben a hívó függvénynek kell vizsgálnia a `root != NULL` feltételt

Bejárás – preorder




```
1  void preorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    printf("%d ", root->data);
6    preorder(root->left);
7    preorder(root->right);
8  }
```

```
31 25 12 11 28 26 30 42 35 33 48
```

- preorder bejárás
  1. gyökérelem
  2. bal részfa
  3. jobb részfa

Ebben a sorrendben kimentve majd visszaolvasva az elemeket, a fa struktúrája visszaállítható.

Faépítés

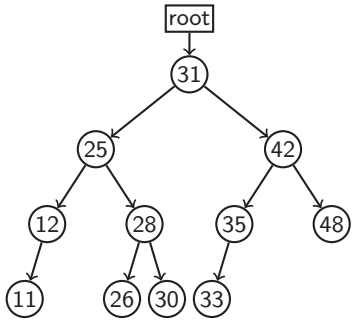
Új elem beillesztése a fába

```
1  tree_ptr insert(tree_ptr root, int data)
2  {
3    if (root == NULL) {
4      root = (tree_ptr)calloc(1, sizeof(tree_elem));
5      root->data = data;
6    }
7    else if (data < root->data)
8      root->left = insert(root->left, data);
9    else
10     root->right = insert(root->right, data);
11   return root;
12 }
```

A függvény használata

```
1  tree_ptr root = NULL;
2  root = insert(root, 2);
3  root = insert(root, 8);
4  ...
```


Bejárás – posztorder




```
1  void postorder(tree_ptr root)
2  {
3    if (root == NULL)
4      return;
5    postorder(root->left);
6    postorder(root->right);
7    printf("%d ", root->data);
8  }
```

```
11 12 26 30 28 25 33 35 48 42 31
```

- posztorder bejárás
  1. bal részfa
  2. jobb részfa
  3. gyökérelem

Ebben a sorrendben először a levélelemeket dolgozzuk fel → alkalmazás: pl. fa lebontása

Fa lebontása posztorder bejárással

```c
void delete(tree_ptr root)
{
  if (root == NULL) /* üres fa leállási feltétel */
    return;
  delete(root->left);    /* postorder bejárás */
  delete(root->right);
  free(root);
}
```

Egy teljes programrész (memóriaszivárgás nélkül)

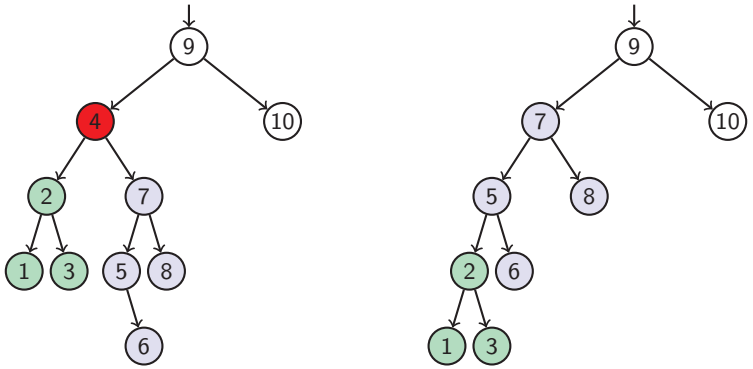
```c
tree_ptr root = NULL;
root = insert(root, 2);
root = insert(root, 8);
...
delete(root);
root = NULL;
```


Egyszerű házi feladatok

. . . senki nem ellenőrzi ☺

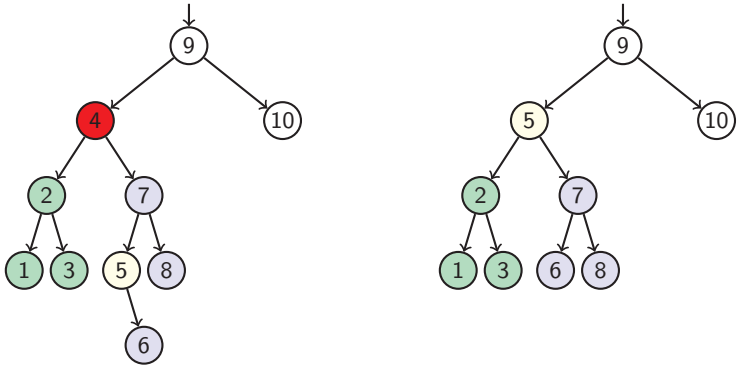
- Írj max. 10 soros rekurzív függvényt, amely
  - megállapítja, hogy milyen mély a fa
  - kiszámolja a faelemek összegét / szorzatát / átlagát
- Írj max. 10 soros iteratív függvényt, amely
  - kiszámolja a faelemek minimumát / maximumát
  - visszaadja a legkisebb / legnagyobb adatot tartalmazó faelem címét

Elem törlése bináris rendezőfából – után



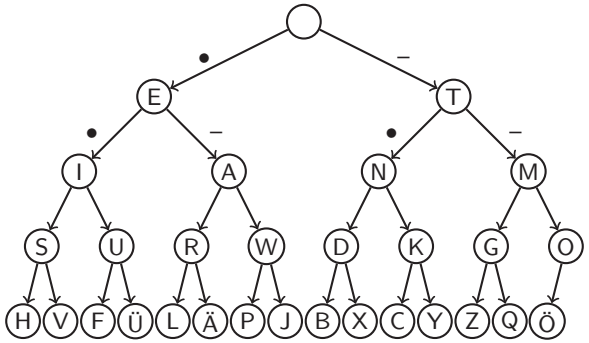
- Jobb részfát felvisszük a törlendő elem helyére
- Bal részfát beillesztjük a jobb részfa legkisebb eleme alá
- Kiegyensúlyozottság romlik!

Elem törlése bináris rendezőfából – okosan



- Jobb részfa legkisebb elemét felvisszük a törlendő helyére
- A felvitt elemnek csak jobb oldali részfája lehetett, ezt gond nélkül feljebbvisszük.

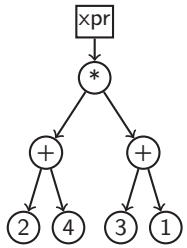
Morse dekódoló fa



SOSOS: ●●● − − − ●●● − − − ●●●

A válasz: ●●●● − − − ● ●●●● − − − ● ●●●● − − − ●

Matematikai kifejezések kiértékelése



- Matematikai kifejezések tárolása fában
- Levél → konstans
- Elágazás → kétoperandusú operátor
- Példában $(2+4)*(3+1)$

```
1  int eval(tree_ptr xpr)
2  {
3    char c = xpr->data;
4    if (isdigit(c))      /* leállási feltétel */
5      return c - '0';
6    if (c == '+')
7      return eval(xpr->left) + eval(xpr->right);
8    if (c == '*')
9      return eval(xpr->left) * eval(xpr->right);
10 }
```


Függvény kiértékelése

Vezessük be az x változót is levélelemként:

```
1  double feval(tree_ptr xpr, double x)
2  {
3    char c = xpr->data;
4    if (isdigit(c))
5      return c - '0';
6    if (c == 'x')
7      return x;
8    if (c == '+')
9      return feval(xpr->left, x) + feval(xpr->right, x);
10   if (c == '*')
11     return feval(xpr->left, x) * feval(xpr->right, x);
12 }
```


Függvény deriváltjának kiértékelése

Deriváljuk a függvényt:

- $c' = 0$
- $x' = 1$
- $(f+g)' = f' + g'$
- $(f \cdot g)' = f' \cdot g + f \cdot g'$

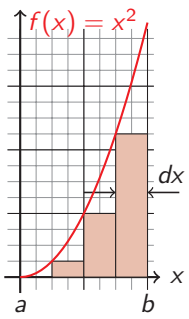
```
1  double deval(tree_ptr xpr, double x)
2  {
3    char c = xpr->data;
4    if (isdigit(c))      /* leállási feltétel */
5      return 0.0;
6    if (c == 'x')      /* leállási feltétel */
7      return 1.0;
8    if (c == '+')
9      return deval(xpr->left, x) + deval(xpr->right, x);
10   if (c == '*')
11     return deval(xpr->left, x) * feval(xpr->right, x) +
12       feval(xpr->left, x) * deval(xpr->right, x);
13 }
```

2. fejezet

Függvénymutatók

Bemelegítő feladat

Írjunk függvényt, amely az $f(x) = x^2$, ill. x^3 függvény görbéje alatti területet közelíti az $[a, b]$ intervallumon, n darab téglalappal




```c
double xsquare(double x)
{
  return x*x;
}
```

```c
double integral_x2(double a, double b,
                   unsigned n)
{
  double dx = (b-a)/n, sum=0.0, x;
  unsigned i;

  for (x=a, i=0; i<n; x+=dx, i++)
    sum += xsquare(x) * dx;

  return sum;
}
```


Motiváció

Határ a csillagos ég...

```c
double integral_x2(double a, double b, unsigned n);
double integral_x3(double a, double b, unsigned n);
double integral_sin(double a, double b, unsigned n);
double integral_sqrt(double a, double b, unsigned n);
```

- ha módosítjuk a számítást, minden ilyen függvényt át kell írnunk
- Helyette jó lenne a görbét megadó függvényt is átadnunk a számolónak

Függvénymutató

Függvénymutató

- a függvény is a memóriában van, ezért címe is képezhető → függvényre mutató pointernek is van értelme
- értékadásnál egy megfelelő típusú függvény azonosítóját kell megadnunk
  típus: paraméterek típusai + visszatérési érték típusa
- a mutatott függvényt a mutatón keresztül meghívhatjuk

```c
double (*fx)(double);
fx = sqrt;
printf("%f\n", fx(5.0));  /* sqrt(5.0) */
fx = sin;                 /* <math.h> */
printf("%f\n", fx(1.57)); /* sin(1.57) */
```

Görbe alatti - bármire

```
13 double integral(double (*fx)(double),
14                 double a, double b, unsigned n)
15 {
16   double dx=(b-a)/n, sum=0.0, x;
17   unsigned i;
18
19   for(x=a, i=0; i<n; x+=dx, i++)
20     sum += fx(x) * dx;
21
22   return sum;
23 }
24
25 int main(void)
26 {
27   printf("%f\n", integral(xsquare, 1.0, 5.0, 100));
28   printf("%f\n", integral(xcube, 1.0, 5.0, 100));
29   return 0;
30 }
```


Függvény mint függvény paramétere

```
1 double integral(/* double (*fx)(double) */
2                 double fx(double),
3                 double a, double b, unsigned n) {
4 ...
5 sum += fx(x) * dx;
6 ...
7 }
8
9 printf("%f\n", integral(xsquare, 1.0, 5.0, 100));
```

- A függvényt mutatóval adjuk át
  Egyszerűsítés: a függvény fejlécét is írhatjuk
- A mutatót a függvényhívás operátorral () használjuk
- Aktuális paraméterként csak a függvény azonosítóját kell megadnunk, hasonló formában, mint a tömb átadásánál

Függvénymutatók tömbje

- Függvénypointerekből álló tömböt is képezhetünk

```
1 double (*(ftrig[2]))(double) ={sin, cos};
2
3 for (i = 0; i < 2; ++i)
4   printf("%f\n", ftrig[i](3.14));
```

- Könnyebben érthető a típus, ha `typedef`-et használunk

```
1 typedef double (*fvptr)(double);
2 fvptr fhyp[]={sinh, cosh};
```

- Megjegyzések
  - A függvénymutatókkal nem végezhetünk mutatóaritmetikai műveleteket
  - A függvénymutatók read-only területre mutatnak, a függvények nem írhatók felül