Logikai programozás

6.

LISTÁK RENDEZÉSE

Buborék rendezés:

```
buborekRendezes(Relacio, Lista, RendezettLista) :-
        csere(Relacio, Lista, UjLista), !,
        buborekRendezes(Relacio, UjLista, RendezettLista).
buborekRendezes(_, RendezettLista, RendezettLista).

csere(Relacio, [A,B|Lista], [B,A|Lista]) :-
                        ellenorzes(Relacio, B, A).
csere(Relacio, [A|Lista], [A|UjLista]) :- csere(Relacio,
                                        Lista, UjLista).
```

LISTÁK RENDEZÉSE

folyt.:

```
ellenorzes(Relacio, A, B) :-
                Cel =.. [Relacio,A,B], Cel.
```

Teszt:

```
?- buborekRendezes(<, [5,3,7,5,2,8,4,3,6], Lista).
```

```
?-  buborekRendezes(@<, [tej, víz, sör, bor], Lista).
```

LISTÁK RENDEZÉSE

Buborék rendezés probléma:

Tegyük fel, hogy a 100. és 101. elemet kellett megcserélni. Ekkor 99-szer fölöslegesen hasonlítottunk, hiszen a lista eleje már rendezett. (Korábban már rendeztük.)

Emiatt nagyon nagy a redundancia.

Javítás: a cserével végigmegyünk a listán, és a rendezést csak akkor folytatjuk, ha a lista még nem rendezett.

LISTÁK RENDEZÉSE

Buborék rendezés (hatékonyabb):

```
buborekRendezes2(Relacio, Lista, RendezettLista) :-
        csere(Relacio, Lista, UjLista),
        Lista \= UjLista,!,
        buborekRendezes2(Relacio, UjLista, RendezettLista).
buborekRendezes2(_, RendezettLista, RendezettLista).

csere(Relacio, [A,B|Lista], [B,A|Lista]) :-
                                ellenorzes(Relacio, B, A).
csere(Relacio, [A|Lista], [A|UjLista]) :-
                                csere(Relacio, Lista, UjLista).
csere(_,[],[]).
```

LISTÁK RENDEZÉSE

Quick sort:

```
quicksort(_, [], []).
quicksort(Relacio, [Fej|Torzs], RendezettLista) :-
        split(Relacio, Fej, Torzs, Bal, Jobb),
        quicksort(Relacio, Bal, RendezettBal),
        quicksort(Relacio, Jobb, RendezettJobb),
        append(RendezettBal, [Fej|RendezettJobb],
                                        RendezettLista).
```

LISTÁK RENDEZÉSE

Quick sort (folyt.):

```
split(_, _, [], [], []).
split(Relacio, VagoErtek, [Fej|Torzs], [Fej|Bal], Jobb) :-
        ellenorzes(Relacio, Fej, VagoErtek), !,
        split(Relacio, VagoErtek, Torzs, Bal, Jobb).
split(Relacio, VagoErtek, [Fej|Torzs], Bal, [Fej|Jobb]) :-
        split(Relacio, VagoErtek, Torzs, Bal, Jobb).
```

LISTÁK RENDEZÉSE

Quick sort:

Teszt:

```
?- quicksort(<, [5,3,7,5,2,8,4,3,6], Lista).
?- quicksort(@<, [tej, víz, sör, bor], Lista).
```

REKURZÍV ADATSZERKEZETEK

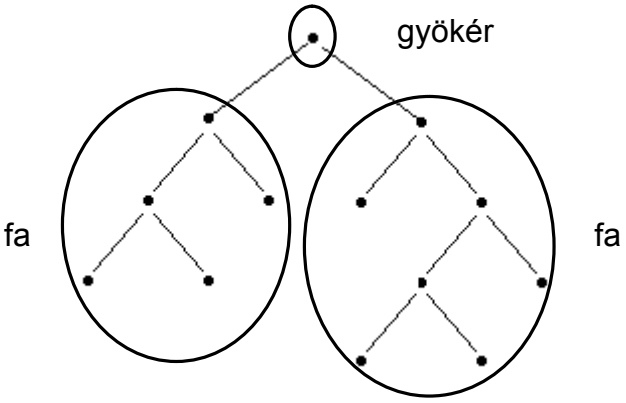
1. Lista

   lista : elem + lista

   leállás: üres lista

   szemléltetés: út egy gráfban

2. Fa



FA STRUKTÚRA

Fa megadása:

fa(gyökér, fa, fa) + üres

Az, hogy a fa definíciójában hol van az elem (gyökér), és hol vannak a részfák, lényegtelen, csak következetesnek kell lenni.

Ez milyen fa?

BINÁRIS FA

Bináris fa bejárása:

– Preorder

gyökér – bal – jobb

– Inorder

bal – gyökér – jobb

– Postorder

bal – jobb – gyökér

BINÁRIS FA

```
kiir(ures).
kiir(fa(Elem,Bal,Jobb)) :- writeln(Elem), kiir(Bal), kiir(Jobb).
```

Ez milyen kiírás?

Inorder:

```
kiir(ures).
kiir(fa(Elem,Bal,Jobb)) :- kiir(Bal), writeln(Elem), kiir(Jobb).
```

BINÁRIS FA

Példa:

```
csalad(fa('Kati', fa('Barna', fa('Anna',ures,ures),
                              fa('Bela',ures,ures)    ),
                  fa('Pali', fa('Marta',fa('Laci',ures,ures),
                                        ures),
                             fa('Zoli', fa('Robi',ures,ures),
                                        ures) ) ) ).
```

```
indit :- csalad(Fa), kiir(Fa).
```

Mit ír ki inorder bejárással?

BINÁRIS FA

Bináris rendezés

ügyes faépítés + megfelelő bejárás

Építés:
A gyökértől balra az összes elem kisebb, jobbra pedig az összes elem nagyobb, mint a gyökérelem (vagy összetett elemeknél ugyanez vonatkozik a kulcsokra).

Bejárás: inorder

BINÁRIS FA

Faépítés:

```
epit(Regifa, ):- read(Elem),Elem\==end_of_file,
                 beszur(Elem,Regifa,

epit(Fa,Fa).
```

BINÁRIS FA

```
beszur(Elem, ures, fa(Elem,ures,ures) ) :- !.

beszur(Elem, fa(Gyoker,Bal,Jobb), fa(Gyoker,Ujbal,Jobb) ) :-
                                    Elem @< Gyoker,!,
                                    beszur(Elem,Bal,Ujbal).

beszur(Elem, fa(Gyoker,Bal,Jobb), fa(Gyoker,Bal,Ujjobb)) :-
                                    beszur(Elem,Jobb,Ujjobb).
```

BINÁRIS FA

```
start :- write('Beolvasando file neve:  '),
         read(Be), see(Be),  be(Fa), seen,
         write('Eredmeny file neve: '),
         read(Ered), tell(Ered), ki(Fa,'fileba'), told,
         writeln('Rendezve:'), ki(Fa,'kepernyore').

be(Fa):- epit(ures,Fa).
```

BINÁRIS FA

```
ki(ures,_).
ki(fa(Elem,Bal,Jobb), Tipus) :-
                    ki(Bal,Tipus), ir(Elem,Tipus), ki(Jobb,Tipus).

ir(Elem, 'fileba') :- writeq(Elem), writeln('.').
ir(Elem, 'kepernyore') :- writeln(Elem).
```

HATÉKONYSÁG

Buborék rendezés:

Legrosszabb eset:

```
?- buborekRendezes2(>, [10,9,8,7,6,5,4,3,2,1], List).
```

Az algoritmus először a 10-et viszi a lista végére, majd a 9-et, stb.
Minden esetben végig kell mennünk a listán (n elem), és n-1 összehasonlítást végezni.
Így n*(n-1) összehasonlítást végzünk.
Vagyis a rendezés komplexitása: $O(n^2)$.

(Bizonyítható, hogy az első változaté $O(n^3)$)

HATÉKONYSÁG

Split:

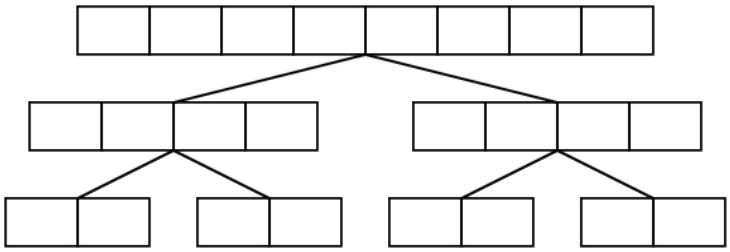
n elemű lista esetén n-1 hasonlítás kell.
Így az algoritmus komplexitása: O(n).

Quick sort:

Azon múlik, hogy melyik elem mentén vágtuk szét a listát.

HATÉKONYSÁG

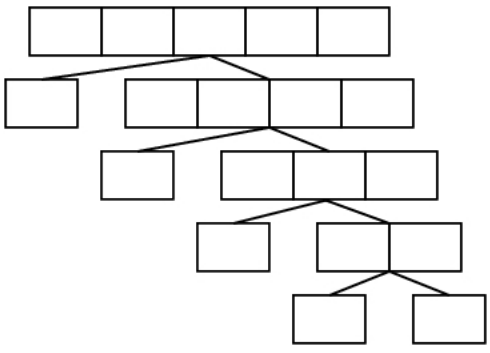
Quick sort:



Kiegyensúlyozott vágás

HATÉKONYSÁG

Quick sort:



Teljesen kiegyensúlyozatlan vágás

(Ha pl. az input lista már rendezett, és mindig a fej alapján vágunk.)

HATÉKONYSÁG

Quick sort:

Komplexitás:
Kiegyensúlyozott lista esetén O(n*log(n)).
Teljesen kiegyensúlyozatlan lista esetén $O(n^2)$.

Bináris rendezés:

Feljavítható O(n*log(n)) komplexitásra.

ELTÁNCOLT ALGORITMUSOK ☺

https://www.youtube.com/watch?v=lyZQPjUT5B4

https://www.youtube.com/watch?v=ywWBy6J5gz8

https://www.facebook.com/AlgoRythmics