4. tétel

Adatszerkezetek és algoritmusok vizsga

Frissült: 2013. január 28.

Listák, szekvenciális adatszerkezetek

A szekvenciális adatszerkezet olyan $\langle A, R\rangle$ rendezett pár amelynél az $R \in (A \times A)$ reláció tranzitív lezártja teljes rendezési reláció.

Az $R \in (A \times A)$ reláció tranzitív lezártja az a reláció, mely tranzitív, tartalmazza R-et, és a lehető legkevesebb elemet tartalmazza.

Megadása.

1. $R' = R \cup (R \circ R)$
2. Ha $R \neq R'$, akkor folytatás 1.-nél, különben $R' = R_T$, a tranzitív lezárt.

Szekvenciális adatszerkezetben az egyes adatelemek egymás után helyezkednek el, van egy *logikai* sorrendjük.

Az adatok között egy-egy jellegű a kapcsolat: minden adatelem csak egy helyről érhető el és az adott elemtől csak egy másik látható.

Két kitüntetett elem: az *első* és az *utolsó*.

Ez egy *homogén* adatszerkezet, azaz azonos típusú véges adatelemek sorozata.

Jelölése : $L = (a_1, a_2, ...a_n)$ Ha $n = 0$, akkor $L = ()$ az üres lista.

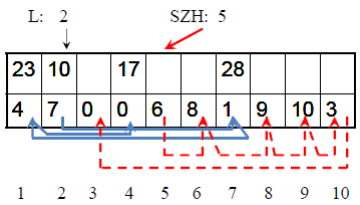
A láncolt lista olyan adatszerkezet, amelynek minden eleme tartalmaz egy (vagy több) mutatót (hivatkozást) egy másik, ugyanolyan típusú adatelemre.

A lánc első elemének a címét a lista *feje* tartalmazza. Ez nem tartalmaz információs részt.

A lánc végét az jelzi, hogy az utolsó elemben a rákövetkező elem mutatója *üres*.

Statikus lista

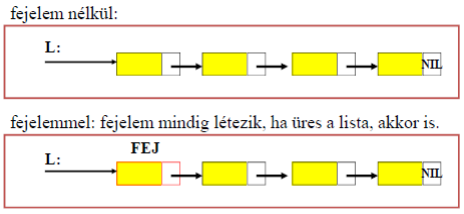
Tömbben, a tömbelemek érték-index párok, a logikai sorrendet az indexek mutatják, a szabad helyek is listában.



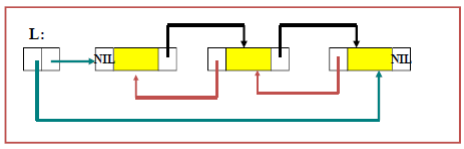
Viszont a beszúrandó adatok száma nem ismerhető előre. Nem akarunk feleslegesen helyet foglalni. Ezért jobb lehet a dinamikus lista.

Dinamikus lista

Egyirányú láncolt lista szemléltetése:



Mindig van egy *aktuális elemre mutató* is.
Kétirányú láncolt listánál vissza mutatók is vannak:



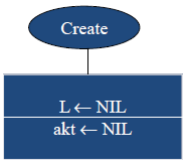
A lista egy eleme áll egy *adatrészből* és egy *mutató részből.*

Műveletek

A lista típus komponensei: *L* - első elem mutatója, *akt* - aktuális elem mutatója.

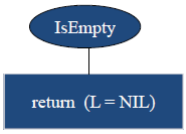
Létrehozás

Egy üres listát ad vissza. Struktogramja:



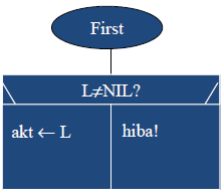
Üres lista lekérdezése

Logikai értéket ad vissza



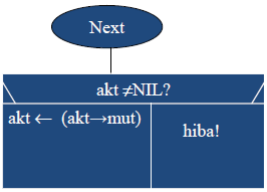
Első elemre állítás

Üres lista esetén hibát dob



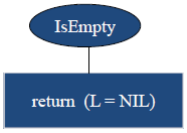
Következő elemre állítás

A lista utolsó elemére kiadott *Next* hatása *akt=NIL* lesz



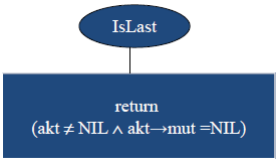
Lista végének lekérdezése

Logikai értéket ad vissza.



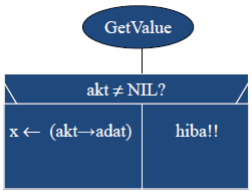
Az utolsó elemen vagyok?

Logikai értéket ad vissza.



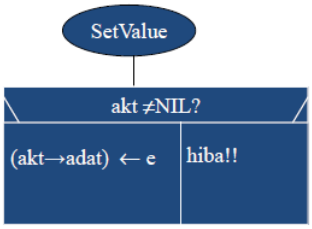
Aktuális elem értéke

Aktuális elem értékét x-ben adja.



Aktuális elem módosítása

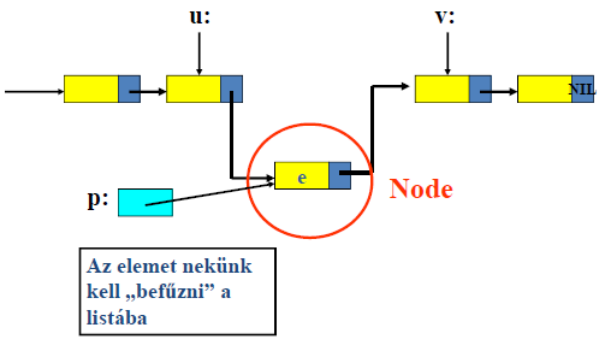
Aktuális elem módosítása e-re.



Listaelem beszúrása

Menete:

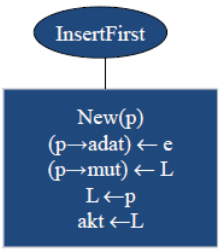
1. Deklarálás - Node típusú elemet fogunk létrehozni
2. Létrehozás - jelölés: new(p)
3. Befűzés



Többféle helyre lehet: elejére, aktuális elem elé, mögé, vagy a végére.

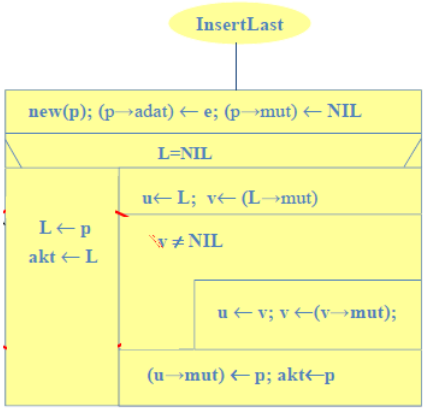
Beszúrás első elemként

Üres és nem üres listára is működik. Első elem lesz az aktuális.



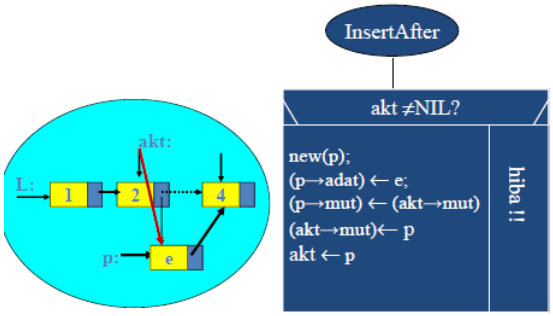
Beszúrás utolsó elemként

Itt az utolsó elem lesz az aktuális. Ez is működik üres listára is.

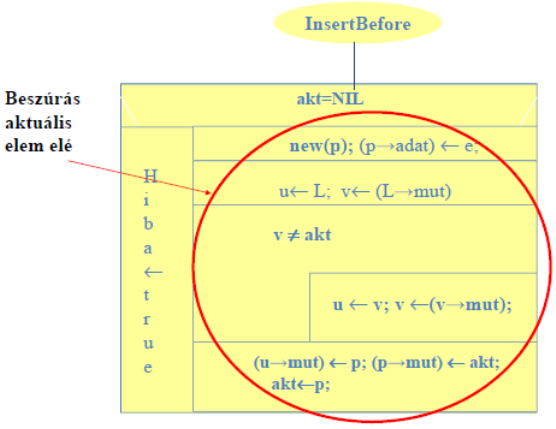


Aktuális elem után

Ha nincs aktuális elem, hiba, az aktuális elem ekkor a beszúrt lesz.



Aktuális elem elé

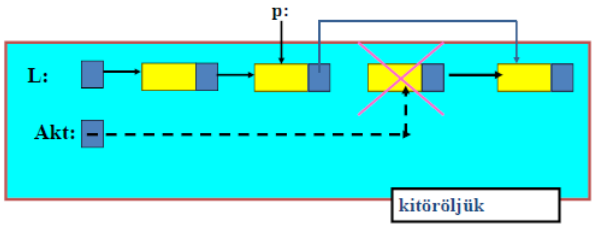


Törlés

Aktuális elem törlése

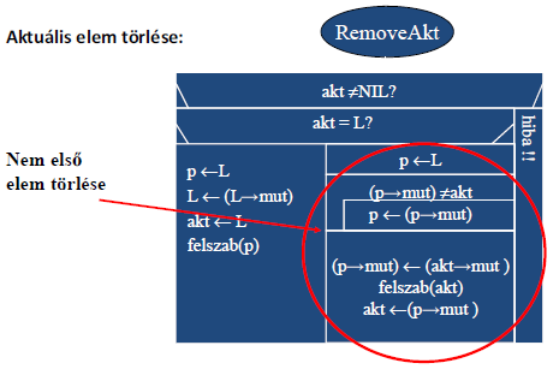
Menete:

1. Akutális elem megelőzőjét átláncoljuk
2. Kitörlés ("delete")



3. új aktuális: az utána lévő - őt is átláncoljuk

Aktuális elem törlésének struktogramja:



Egyéb műveletek

Lehet még pl. teljes lista törlése, listák összefűzése, elemszám. Egyirányú láncolt listánál nem elég hatékonyak az alábbi műveletek: last, remove, insertbefore, insertlast. Itt jobb a kétirányú láncolt lista, vagy egyéb más.

Rendezés

Egy példa a fenti műveletek alkalmazására a rendezés.

Adott az $A[1..n]$ egészeket tartalmazó tömb. Helyezzük el a *pozitív* elemeit rendezett módon egy listába!