4. gyakorlat

Téma:

Láncolt ábrázolás, egyszerű lista, egyirányú, fejelemes lista. Keresés, beszúrás, törlés rendezett egyszerű listába, beszúrás változata fejelemes lista esetén, egyszerű lista megfordítása, egyszerű lista szétfűzése, prímszita listával.

Bevezetés

Ha van idő, kezdhetjük az alábbi konkrét példával: tömbben megvalósított lista. A tömb elemei két mezőből állnak, egy szöveg és egy egész értékből. Az egész érték egy tömbindex, mely egy láncba fűzi a tömb elemeit.

Rajzoljuk fel közösen, lánc formájában, milyen listát tartalmaz a tömb? A tömbben a szabad helyeket is egy lista tartja nyilván, ennek első elemére mutat SZH. Rajzoljuk le közösen, mi lenne az állapot a „narancs" beszúrása, majd az eper törlése után.

1	málna	0	L=7
2		0	SZH=4
3	banán	8	
4		5	
5		2	
6	körte	1	
7	alma	3	
8	eper	6	

1. ábra: láncolt lista tömbös megvalósítása.

Láncolt listák

Egy vagy két irányúak lehetnek.
Összehasonlítás a tömbbel:

- Előny: a rendezett beszúrás/törlés nem igényel elemmozgatást. Persze a beszúrás/törlés helyének megtalálása rendezett esetben O(n).
- Hátrány: nem indexelhető konstans műveletigénnyel, csak O(n)-nel!

Egyirányú lista

Listaelem típusa (jegyzetből):

E1
$+key : \mathcal{T}$ $\ldots$ // satellite data may come here $+next$: **E1***
$+\mathbf{E1}()\ \{\ next := \oslash\ \}$

2. ábra Egy listaelem felépítése

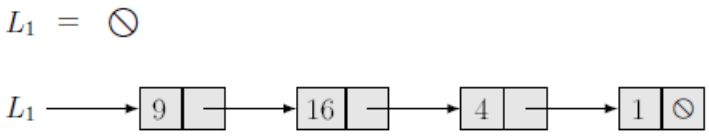
Bejárásához pointereket használunk: p,q: E1*

Elem adattagjainak elérése: p→key, p→next (Helyes még (*p).key, (*p).next)

Ha új listaelemet szeretnénk létrehozni: p = new E1,
feleslegessé vált listaelem felszabadítása: delete p (utána már nem hivatkozhatunk rá!)

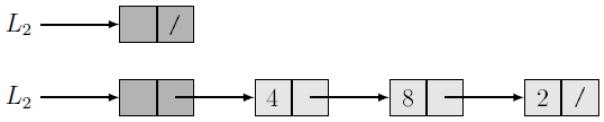
Egyirányú listák fajtái (jegyzetből)

1. **Egyszerű, egyirányú láncolt lista (S1L):** a legegyszerűbb forma, az első elemre egy pointer mutat. Ha még nincs eleme a listának, ez a pointer 0 (Null, Nil) értékű.



3. ábra: egyszerű láncolt lista

2. **Fejelemes egyirányú láncolt lista (H1L):** Gyakori trükk, hogy egy valódi adatot nem tároló elemet helyezünk el a lista elejére. Célja: a lista elején (vagy az üres listával) végzett műveletek megkönnyítése, mert így a lista elejére mutató pointerünk soha nem 0 értékű, továbbá az első elem előtt is van egy listaelem. (Léteznek végelemes listák is.)



4. ábra Fejelemes egyirányú láncolt lista

Megemlíthető a ciklikus egyirányú lista is, de a feladatokban nem fogjuk használni.

Kétirányú listákkal majd a következő gyakorlaton foglalkozunk.

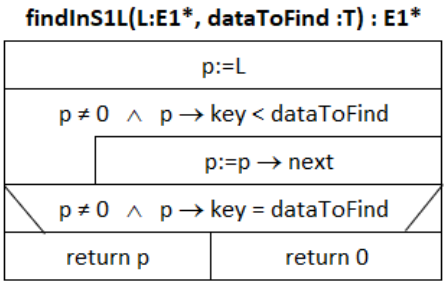
Feladatok

S1L kezelésének bemutatása az alábbi feladaton keresztül:
egy halmazt ábrázolunk egyszerű listával (egy adott kulcs csak egyszer fordulhat elő). Típus műveletek:

- egy elem benne van-e a halmazban: **keresés** kulcs alapján,
- egy elem betevése a halmazba (ha még nincs benne): **beszúrás,**
- egy elem kivevése a halmazból (ha benne van): **törlés.**

Gondoljuk meg a **rendezetlen** és **rendezett** ábrázolás közötti különbséget! Legyen most a listánk növekedően rendezett! Készítsük el a keresés, beszúrás, törlés algoritmusát:

Keresés:



Sikertelen keresés eredménye: 0 pointert adunk vissza, így nem kell a logikai változó, amit a lineáris keresés tételénél tanultak.

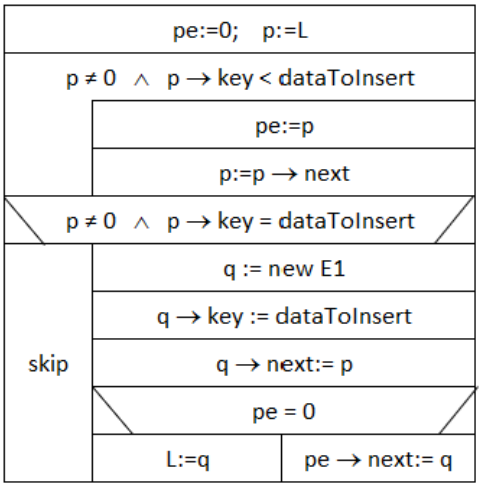
Beszúrás:

A kereséshez hasonló ciklussal indul, meg kell keresni a kulcs helyét a rendezett sorozatban. Rajzoljuk le az eseteket!

- Leggyakoribb, hogy a lista belsejébe szúrunk be, mely elemek címe kell a művelethez?
- Változik-e valami, ha a lista utolsó eleme után fűzzük be az új elemet?
- Mennyiben más a lista elejére történő befűzés?
- Hogyan kell üres listába befűzni?

A műveletekhez világos, hogy két elem címe kell: az az elem, ami elé fogunk befűzni (az első olyan, melynek kulcsa nagyobb a beszúrandó kulcsnál), valamint az előtte lévő elem címe. Ehhez két bejáró pointert használunk. Megoldható persze egy bejáró pointerrel is. Egyirányú listák esetén, ha a lista felépítését megváltoztatja az algoritmus, kevesebb hibával jár, ha mindig két bejáró pointert használunk. Próbáljuk a fenti négy eset közös elemeit megtalálni! Kiderül, hogy csak az a különbség, hogy az az elem, amelyik az aktuális elem előtt van, az nem mindig létezik, így egyedül ezt kell majd egy elágazással kezelni.

insertIntoS1L(&L: E1*, dataToInsert: T)



A két bejáró pointer: pe, és p.

A pe pointert 0-ról indítjuk; ez jelzi majd, hogy nincs még „előző” elem!

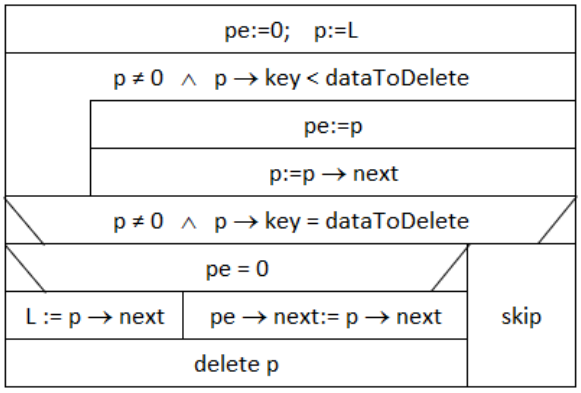
Ha már van ilyen kulcsú elemünk, nem történik semmi.

A beszúrásnál csak az a lépés kerül elágazásba, amikor beszúrt elem előtti elemnek a next pointerét módosítjuk, ugyanis, ha nincs előző elem, akkor L módosul!

Mivel a műveletnél L pointer módosulhat, így fontos, hogy az cím szerint átvett paraméter legyen!

Törlés:

deleteFromS1L(&L:E1*, dataToDelete: T)



Hasonlóan a beszúráshoz, itt is pe és p a két bejáró pointer.

Ha az adott kulcsú elem nem található meg, akkor nem történik semmi.

Itt is fontos, hogy L cím szerinti paraméter.

A delete művelet a memóriaszivárgás elkerülése miatt fontos!

HF: Nézzük meg fejelemes listára is (H1L) a műveleteket!