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

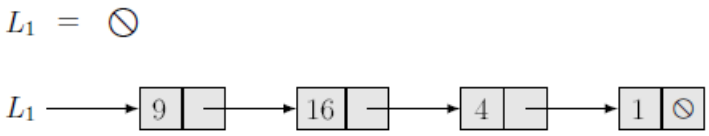
Bejárásához pointereket használunk: p,q: E1*

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!)

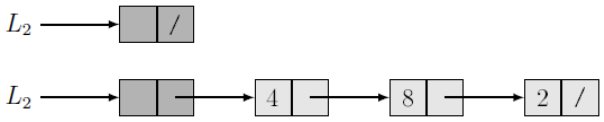
Egyirányú listák fajtái (jegyzetből)

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.

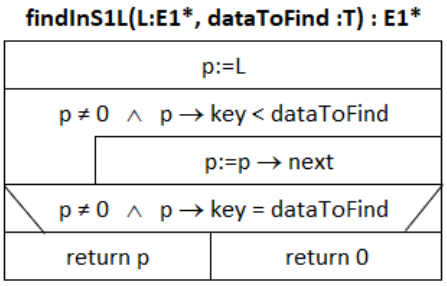
Feladatok

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.

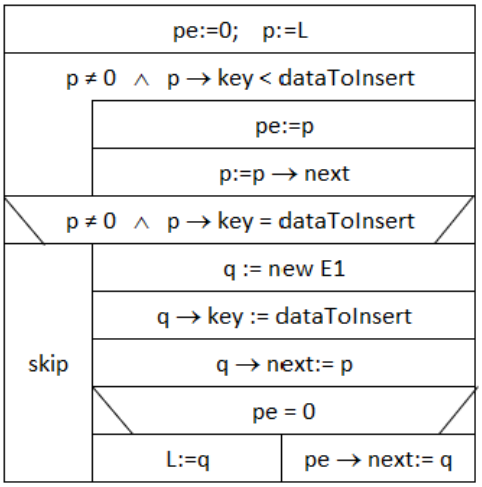
Beszúrás:

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!

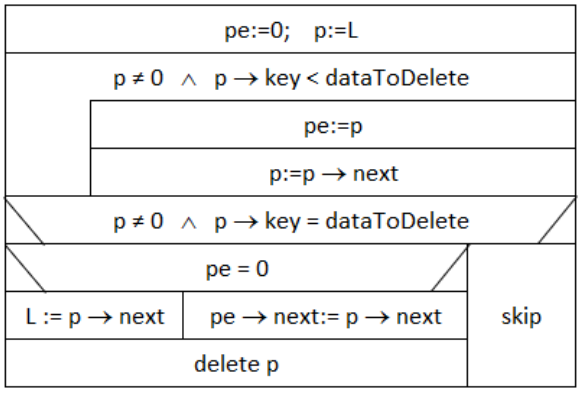
Ha már van ilyen kulcsú elemünk, nem történik semmi.

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.

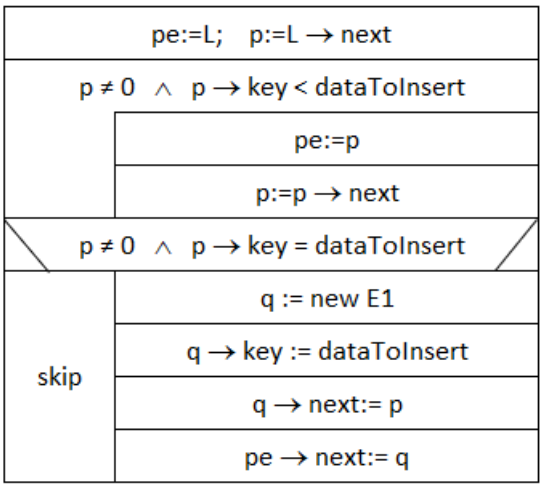
A delete művelet a memóriaszivárgás elkerülése miatt fontos!

Nézzük meg fejelemes listára is (H1L) a műveleteket!

Keresés: csak az indulás különbözik, a fejelemben nincs kulcs, így p:= L→next az induló lépés.

Beszúrás: indulásnál pe a fejelemre mutathat, hiszen az első elem előtt ott van mindig a fejelem, p pedig hasonlóan az előbbi algoritmushoz a lista első elemére mutat. Mivel pe nem lehet 0, nincs szükség az elágazásra a befűzésnél.

insertIntoH1L(L: E1*, dataToInsert: T)

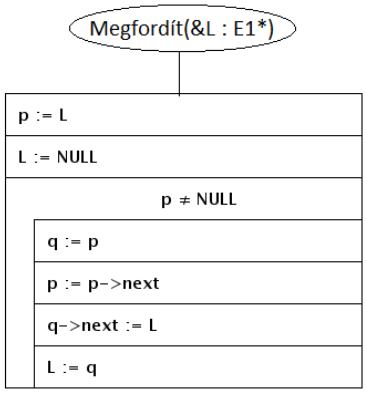


Fejelemes lista esetén a fejelem címét megadó paraméter nem cím szerint adódik át, hiszen a fejelem nem változtatja meg a címét, így az arra mutató pointer biztosan nem fog változni.

Törlés: indulás hasonlóan, itt sem kell elágazás a kifűzésnél, mert pe nem lehet 0 értékű.

Egyszerű lista megfordítása:

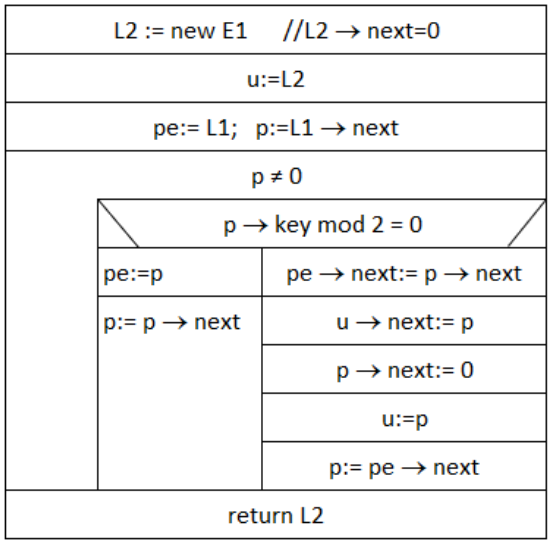
Trükk: Mintha egy verembe raknánk a lista elemeit, az eredmény listának mindig az elejére fűzzük be az eredeti lista aktuális elemét.



H1L szétfűzése két listába

Feladat: adott egy H1L, egész számokat tartalmaz, a fejelemre L1 mutat. Fűzzük ketté az elemeket: L1-ben maradjanak a páros elemek, egy új L2 H1L listába fűzzük át a páratlan elemeket. Az eredeti sorrendet tartsuk meg, azaz átfűzésnél mindig a lista végére kell majd fűzni. Az L1 listát egyszer lehet bejárni, L2-be a befűzés konstans műveletigényű legyen!

Szétfűz(L1:E1*): E1*



Első lépésként egy új H1L létrehozását mutatjuk be. Kihasználjuk, hogy E1 konstruktora az elem next pointerét 0-ra állítja!

u mindig L2 utolsó elemére fog mutatni, ez kezdetben a fejelem.

pointerek L1 bejárásához

Ha páratlan kulcsú elemmel találkozunk, a lépések:

p kifűzése

p befűzése L2 végére

mivel p az L2 utolsó eleme, 0-ra állítjuk a next pointerét

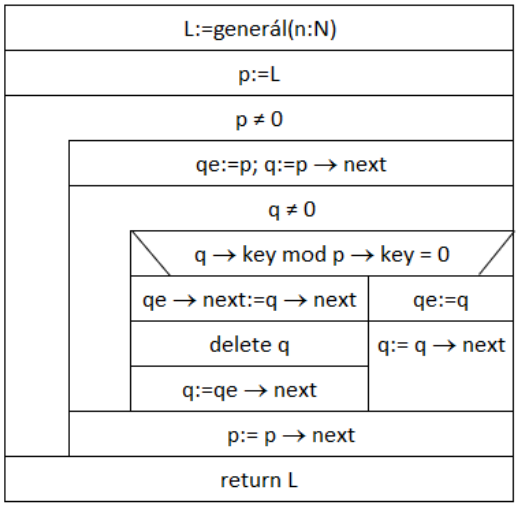
módosítjuk u-t

p-vel L1 következő elemére állunk

Primszita egyszerű listán

Készítsünk egy listát, mely 2-től n-ig tartalmazza a prímeket, az Eratoszthenészi szita algoritmus ötletét felhasználva. Töltsünk fel egy egyszerű listát 2..n természetes számokkal. Az első elem prím, azokat, melyek oszthatók ezzel az elemmel, töröljük a listából. A következő megmaradt szám megint prím. Többszöröseit ismét töröljük. A végén a prímek maradnak a listánkban.

Primszita(n:N) : E1*



Generálunk egy egyszerű listát 2..n természetes számokból.

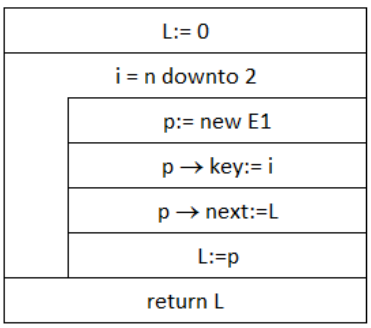
p mindig a következő, még a szitában lévő prímszámra mutat (elsőként a 2-re). Amíg ez nem 0 ...

q-val a p utáni elemeket fogjuk bejárni, és kifűzzük p → key többszöröseit

Ha p→key osztója q→key-nek, a q című elemet töröljük a láncból. Ehhez szükségünk van q elem előzőjére, ez lesz a qe pointerben.

ha p című elem többszöröseit töröltük, a következő elem megint prím lesz.

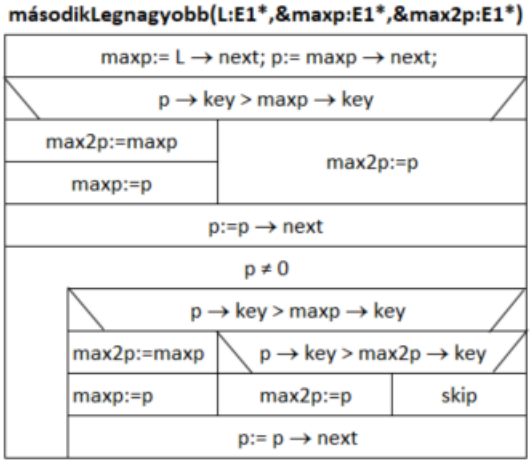
generál(n:N) : E1*



Egyszerű lista feltöltése 2..n természetes számokkal.
Trükk: a ciklust csökkenőleg futtatjuk, így mindig a lista elejére kell befűzni.

Házi feladatok:

1. Egy rendezetlen egyirányú fejelemes listában (H1L) keressük meg a 2. legnagyobb elemet, egyszeri bejárással! Feltehető, hogy a listánk legalább két eleme van (a fejelemen kívül).



Fontos az előfeltétel: a listának legalább két eleme van!

p a fejelem utáni második elemre mutat! (Ha esetleg üres a lista, ez elszállna!)

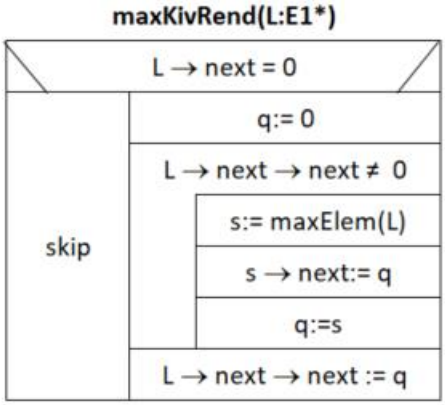
Elindulás az első két elem alapján:

maxp a nagyobbik, max2p a másik elemre mutat.

"Mellékesen" a legnagyobb elem címét is megkapjuk.

2. Rendezzünk egy H1L listát maximum kiválasztó rendezéssel. (Ügyeljünk, hogy a megoldó algoritmus üres listára is működjön!) Ezt mindenképp beszéljük meg a következő gyakorlaton, vizsgaanyag!

Megoldási ötlet: megkeressük és kifűzzük a lista legnagyobb elemét. A kifűzött elemekből egy egyszerű listát kezdünk építeni úgy, hogy mindig a lista elejére fűzzük be a rendezendő listából kifűzött elemet, nevezzük az így kapott listát segéd listának. Ez a segédlista nyilvánvalóan növekvően rendezett lesz. Amikor a rendezni kívánt listának már csak egy eleme maradt, hozzáláncoljuk a segédlistánkat az utolsónak megmaradt elemhez, és készen vagyunk.



Üres listát külön kezeljük a ciklusfeltétel miatt.

q mutat a rendezett elemekből álló egyszerű lista első elemére.

amíg legalább két elemet tartalmaz a lista...

megkeressük, és kifűzzük a lista legnagyobb kulcsú elemét

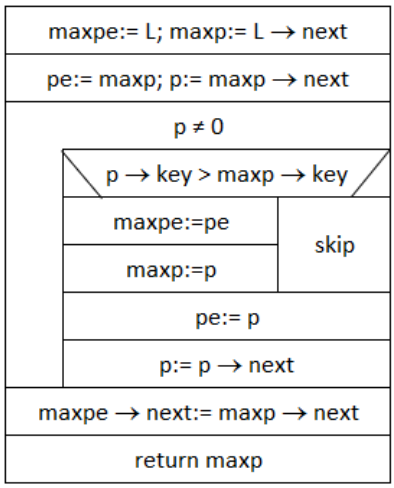
q című egyszerű lista elejére fűzzük a kapott elemet

Amikor L már csak egy elemet tartalmaz, a segéd listát befűzzük az első elem után.

Észrevehető, hogy egy elemű lista esetén a jobb ág gyorsan és helyesen lefut, így nem indokolt ennek külön esetként történő feldolgozása.

A legnagyobb elemet megkereső, és kifűző algoritmus:

maxElem(L:E1*): E1*



Megkeresi és kűzi a lista egyik legnagyobb elemét.

Előfeltétel, hogy legalább egy eleme legyen a listának.

maxp tartja nyilván a legnagyobb elemet, maxpe az előtte lévőt.

pe, és p pointerekkel járjuk be a listát.

Menet közben karban tartjuk maxpe és maxp pointereinket:

ha nagyobb kulcsot találunk, mint az eddigi legnagyobb,

maxpe-t pe-re és maxp-t p-re állítjuk.

Kifűzzük a maxp című elemet,

és visszatérünk a címével.

Készült az „Integrált kutatói utánpótlás-képzési program az informatika és számítás-tudomány diszciplináris területein” című EFOP 3.6.3-VEKOP-16-2017-00002 azonosítójú projekt támogatásával.