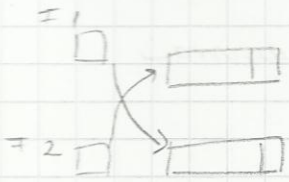
algol.
EA.

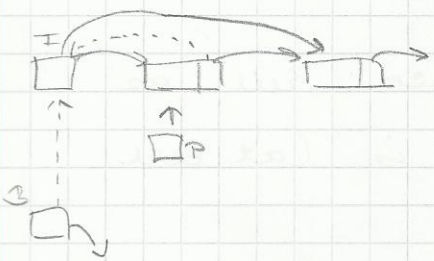


LISTA RENDEZÉSE:

< A eddi rendezések arra épültek, hk. az adatok közvetlen elérésűek. Itt nem alkalmazható. >

B

F → rendezetlen üres listát azonosít



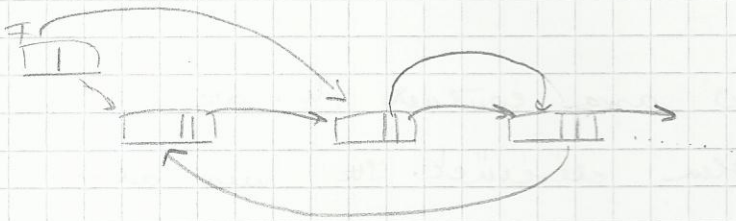
A B által azonosított listába lefűzzük az elemet, ill. rendezett legyen.

A listafűzésnél nem adatot adunk meg, hanem a mutatót.

listásbeillesztéses rendezés

cikluslánccá vista az utolsó elem arra mutat, amire a fej.

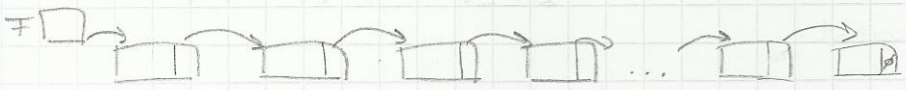
Sorrend rendszer definiálható, akárny mutatót adtunk meg a listában



Hátránysságai:

- a lista végén végrehajtandó művelethez végig kell menni
- elem törlése esetén ismerni kell az őt megelőző elem címét is (át kell írni a mutatóját.)
- nem tudunk adott elemet törölni, csak utána
- csak adott elem után tudunk beszúrni, elé nem. (ez lesz jó!)

Változtatás:

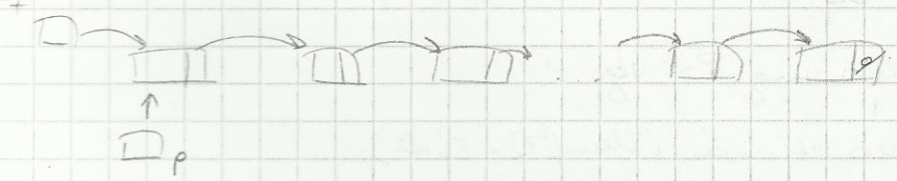


Lefoglalunk egyet több elemet, mint amennyire szükség van: STRÁZSÁS KERESÉS ⇒ STRÁZSÁS LISTA.

Ha az utolsó elemre keresünk, csak a strázselemet, de akkor a keresés és a beszúrás során 2x kell végigmenni.

Nem közvetlen elérésű, ⇒ TEGYÜK közvetlen elérésűvé a sorozatokat.

Legyen ez egy mutató, a vége mutató.



Utána azt kell nézni, hogy P = V vagy sem.
Ha igen, akkor nem található benne,
ha nem, akkor van benne.

A sorozás lista inicializált állapotban 1 elemet tartalmaz (a többi lista üres)

⇒ ilyenkor a sorozás lista üres.

Fgv-ként kell írni, mert kell, h. mikor marad hely ⇒ STRÁSA

Függvény SListatelt(Fej, Vég : Mutatótíp) : Logikai;
↑ ha tömött ⇒ IGAZ

Algoritmus

Ha elfoglal (fej) akkor

Vég := fej; SListatelt := igaz;

különben

SListatelt := hamis;

Ha vége;

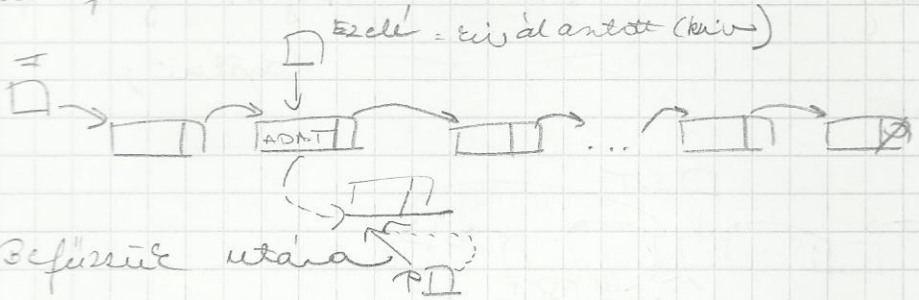
Ivége

```
Eljárás Slistafeldolg (fej*, vég: mutató típ);
  Változó P: mutató típus;
  Algoritmus
    P:= fej;
    Ciklus míg P<>vég
⇒     feldolgozó eljárás (Elem[P].Érték);
⇒     P:= Elem[P].köv;
    c vége;
elj vége;

Függvény SlistaKeresés (fej, vég, hely: mutatótípus; adat:
                        értéktípus): logikai;
  Változó P: mutatótípus;
  Algoritmus
    Elem[vég].Érték := Adat;
    P:= fej;
    Ciklus míg Elem[P].Érték <> Adat
    P:= Elem[P].köv;
    c vége;
    hely:=P;
    Slistakeresés:= (P<>vég);
c vége;

Nem működne a P, ehelyett helyettesítse a
HELY-et is használási.
```

Listafűzelemelé



Befűzése utána

P

Nincsen kell! Csak nem kellenes

Függvény ListafűzElemelé (Eszelé, vég : mutató típ,
adat : Érték típ) : logikai

változó P: mutató típ

Algoritmus

Ha Ufoglal(P) akkor

Elem[P] := Elem[Eszelé];

Elem[Eszelé], Érték := Adat;

Elem[Eszelé].köv := P;

Ha Vég = Eszelé akkor

Vég := P.
↳ újonan beszúrt elem helye.

Ü vége;

ListafűzElemelé := Igaz;

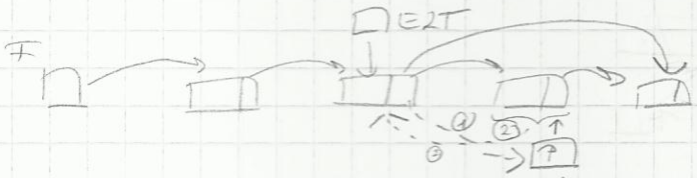
különben

ListafűzElemelé := Hamis;

Ü vége;

F vége;

Függvény Slistatöröllelem (Ezt, Vég : mutató típ):
logikai;



Változó P: mutató típ; (lokális)

Algoritmus

Ha ezt <> vég akkor

P: = Elem [Ezt]. KÖV;

Elem [Ezt] : = Elem [P];

Ha Vég = P akkor Vég: = Ezt ;

Felszabadít (P);

Slistatöröllelem : = Igaz;

Különben

listatöröllelem : = Hamis;

Hvége;

Fvége;

A felszabadítás: a szabad lista bővítése az elején

lefoglal: szabad lista első elemének törlése.

Ez pont úgy működik, mint a verem.
↓
szabad lista.

Nincs annyira kerete közé szorítva.

Ez a verem ún. szétszórt reprezentációja

A verem így nem más, mint az elbuktott lista.

Így lehet elvégezni a veremkezelő műveleteket.

Veremüresít, veremüres, veremteli, verembe, veremből.

5. előadás

eljárás l_veremürezd(v: veremtip);

TIPUS veremtip =
= mutatótip ;

v := végjel ;

e vége ;

Megvalósíthatjuk strázsás listával, de nem kell, mert a lista elején dolgozunk.

——— o ———

függvény l_veremüres(v: veremtip): logikai ;

l_veremüres := (v = végjel) ;

f vége ;

függvény l_veremtele (v: veremtip): logikai ;

l_veremtele := (szabad = végjel)

ha a szabadlista feje végjel, a szabadlista üres ⇒ nem lehet bővíteni !

f vége;

——— o ———

(v: → mert a lista feje megváltozik.)

függvény l_verembe (v: veremtip, adat: elemtip): logikai ;

l_verembe := listafűzelől (v, adat) ;

f vége;

növeli a [illegible]

——— o ———

függvény l_veremből (v: veremtip, adat: elemtip): logikai ;

ha (v ≠ végjel) akkor

adat := elem[v].érték ;

l_veremből := listatörölelől(v) ;

különben

l_veremből := hamis ;

f vége;

Sor

- Mindig a legelőször betett adatot olvassa ki leghamarabb.

- Mind a két végén kell műveleteket végezni
- El kell dönteni, hogy az elején bővítsünk vagy a végén s ezért kell hozzá → hosszadalmas lehet.

3 K ↘ K 3

Ez lesz a sor eleje.

Az elejéről kell kiolvasni, onnan kell felsza-badítani

l-sorüres
↓
függvény előkészíti az 1-es adatelemet kezelésére és a strázsás lista inicializálása.

függvény l-sorüres (s: sortip): logikai;

függvény sortip = rekord
s.e, s.v : mutató
r vége

l sorüres := Slistaüres (s.e, s.v);

függvény l-sorüres (s: sortip): logikai;
l - sorüres := s.e = s.v);

f vége;

függvény sortelé (s : sortip)

A sorban is akkor vagyunk benne, ha a listaelemek helyét tároljuk.

l_sortelé (szabad = végjel)

függvény l_sorbaradása (s : sortip, adat : elemtip) : logikai

l_sorba : s listafűz (s, v, adat)

függvény l_sorból (s : sortip, adat : elemtip) : logikai

ha s.e ≠ s.v akkor

adat := elem [s.e]. érték.

l_sorból := s_listatörölelem (s.e);

különben

l_sorból := hamis;

f vége.

A kétirányban láncolt lista ugyanúgy (minimumkasan) működik. lehet törölni az elejéről, meg a végéből, és nem is lehet szeret megfeleltetni. csak H-ek kellenek.

A sor kétkört reprezentációja ⇒ dinamikus

ha verem : a verem kétkört reprezentációja

• dinamikusan változik a mérete.

Helyenként a listakezelő algo-k is használhatók.

Csak akkor lépheti túl 1 helyen határait, ha már a dinamikus tárkezelés nem bírja.

összefuttatás külde ⇒ unió

⇓

meg lehet csinálni a metszetet, meg a különbséget is.

<u>Sor megvalósításai statikus esetekben</u>

• lefoglalunk egy egybefüggő memóriaterületet

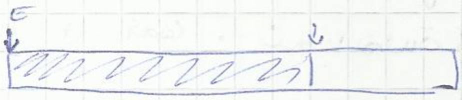
• van eleje- (olvashatunk) és vége- (írhatunk) mutatónk

• ha bővítjük, a V mutató halad a tárterület vége felé, ill. olvasáskor az E is halad a tárterület vége felé

• ha az eleje után elérjük a V mutatót (kiolvasáskor), ill. ha tele van a tárterület és nem tudunk már írni bele ⇒ EGYSZERŰ SOR

megtelik, ha a V-vel elértük a tárterület végét.

• megoldás lehet:



eleje mindig az első elemre mutat, kiolvasás után minden elemet 1 pozícióval előbbre visszük, és a V is 1-gyel előbbre ugrik.

LÉPTETŐSOR

kiküszöböli azt a hibát, hogy csak akkor telik meg, ha minden hely ki van ~~tárolva~~ használva.