PROGRAMOZÁS

Adatszerkezetek csoportosítása,
vermek,
sorok.
Speciális algoritmusok veremmel

Python nyelven

Adatszerkezet

Az adatszerkezet az adatelemek egy olyan véges halmaza, amelyben az adatelemek között **szerkezeti összefüggések** vannak.

Az **összekapcsolódás módja** határozza meg az elemek egymáshoz való viszonyát, illetve azokat a műveleteket, amelyeket rajta végezhetünk.

(Adatelem az a legkisebb adategység, amelyre hivatkozni lehet.)

Matematikai **adatszerkezet**

<A,R>

rendezett pár, ahol

- A: adatelemek halmaza
- R: az A halmazon értelmezett valamilyen reláció

- A matematikai adatszerkezetek az R reláció bonyolultsága alapján csoportosíthatók

Absztrakt társzerkezetek

- A matematikai adatszerkezetek *elméleti adatszerkezetek* –le kell őket képezni a tárolókra
- címezhető tárolókra (lemez, memória)
- Nem címezhetőkre, ha csak sorosan szekvenciálisan járjuk be

Az adatszerkezetek osztályozása **Szerkezet** szerinti csoportosítás

I. Homogén adatszerkezetek

I. **Homogén adatszerkezetek**

(Azonos típusú adatelemekből épül fel)

- I.A Struktúra nélküli adatszerkezet
  - Az egyes adatelemek között nincs kapcsolat. Nem beszélhetünk az elemek sorrendjéről.

Az adatszerkezetek osztályozása Szerkezet szerinti csoportosítás

I. Homogén adatszerkezetek

- I. B Asszociatív adatszerkezet
  - Az adatelemek között lényegi kapcsolat nincs. Ez az adatszerkezet valamilyen közös tulajdonság alapján összeállított halmaz, amelyekből részismérvek alapján részhalmazokat választhatunk ki. Az adatszerkezet elemei tartalmuk alapján címezhetők. pl.: tömb, ritka mátrixok, táblák.
    - Közvetlen elérésű
    - Véletlen elérésű

Az adatszerkezetek osztályozása Szerkezet szerinti csoportosítás

I. Homogén adatszerkezetek

- I. C Szekvenciális adatszerkezetek
  - Szekvenciális adatszerkezetben az egyes adatelemek egymás után helyezkednek el. Az adatok között **egy-egy jellegű** a kapcsolat: minden adatelem csak egy helyről érhető el és az adott elemről csak egy másik látható. Pl. egyszerű lista

Az adatszerkezetek osztályozása Szerkezet szerinti csoportosítás

I. Homogén adatszerkezetek

- I.D Hierarchikus adatszerkezetek
  - A hierarchikus adatszerkezet olyan <A,R> rendezett pár, amelynél van egy kitüntetett elem a gyökérelem úgy, hogy
  - A gyökér nem lehet végpont
    Bármely gyökértől különböző elem egyszer és csak egyszer lehet végpont
    Bármely gyökértől különböző elem a gyökértől elérhető
  - A hierarchikus adatszerkezeteknél az adatelemek között **egy-sok jellegű** kapcsolat áll fenn. Minden adatelem csak egy helyről érhető el, de egy adott elemből tetszés szerinti számú adatelem látható.
    Pl. fa, összetett lista

Az adatszerkezetek osztályozása Szerkezet szerinti csoportosítás

I. Homogén adatszerkezetek

- I.E Hálós adatszerkezetek
  - A hálós adatszerkezet olyan <A,R> rendezett pár, amelynél az R relációra semmilyen kikötés nincs.
  - A hálós adatszerkezetek adatelemei között a kapcsolat **sok-sok jellegű**: bármelyik adatelemre több helyről is eljuthatunk, és bármelyik adatelemtől elvileg több irányban is mehetünk tovább.
  - Pl. gráf, irányított gráf

II. Heterogén adatszerkezetek

- Különböző típusú adatokból épül fel.
- pl.: Record

Memóriában történő helyfoglalás szerinti csoportosítás

- I. Statikus adatszerkezetek
  - Véges számú adatelemből épül fel és ezek csak értéküket változtathatják hosszukat nem
  - pl.: Tömb, rekord, halmaz

Memóriában történő helyfoglalás szerinti csoportosítás

- II. Dinamikus adatszerkezetek
  - Adatelemek száma tetszőleges, az adatelemek száma változhat.
  - pl.: Lista, Fa, Gráf

Továbbá

- Lehet **rekurzív**, melynek definíciója önmagára való hivatkozást tartalmaz. **Nem lineáris**, ha több ilyen hivatkozás van.
- file-ok: nem rekurzív
- láncolt lista: rekurzív lineáris
- összetett lista, fa, gráf: rekurzív, nem lineáris

Összetett adattípusok

Tömb

Füzér

Lista
- Verem
- Sor

Fa, Bináris fa

Kupac, halmaz

Gráf

Hasító tábla (dictionary)

Műveletek adatszerkezetekkel

Az adatszerkezetek meghatározzák, a rajtuk végezhető műveleteket

Lehetnek *konstrukciós* vagy *szelekciós* műveletek

Műveletek adatszerkezetekkel

Tömb: Műveletek, csak az egyedi összetevőkön végezhetők (kivétel a paraméterátadás)

Rekord: A mezőkön, a mező típusának megfelelő műveletek végezhetők

Műveletek adatszerkezetekkel

Halmaz:

- Csak a teljes halmazon végezhetők műveletek:
  - létrehozás: (feltétellel, felsorolással)
  - in: az adott elem benne van-e a halmazban
  - +: unió
  - *: metszet
  - -: különbség összehasonlítás

Műveletek adatszerkezetekkel

- **Láncolt lista:**
  - beszúrások
  - törlések
  - bejárás,
  - feldolgozás (pozícionálás, érték kiolvasása)
  - jelzés: üres-e, van-e következő elem

Műveletek adatszerkezetekkel

- **Verem, sor:**
  - elem be
  - elem ki
  - üres-e
- **Összetett lista**
  - Cons
  - Append
  - List

Műveletek adatszerkezetekkel

- **fa:**

elem hozzáadása (levél, gyökér)

törlés (levél, 1 leágazású csúcs, 2 leágazású csúcs)

bejárás (preorder, postorder, inorder)

tesztek (levél-e, van-e adott oldali részfája)...

Műveletek adatszerkezetekkel

- Mindegyiken elvégezhető: üres szerkezet inicializálása
- Adatelemeken végezhető műveletek (ahol ez engedélyezett)
  - csere - egy adatelem értékének megváltoztatása
  - rendezés
  - keresés - egy adott értékű vagy adott feltételt kielégítő adatelemek helyének megkeresése
  - bejárás - valamennyi adatelem egyszeri elérése, feldolgozás

Matematikai modell: Címezhető absztrakt társzerkezetek

- 1. Vektor

Elemei direkt elérhetők

Kezelése egyszerű, megfelel az egydimenziós tömbének

Ha a matematikai adatszerkezet és a társzerkezet rokon, akkor jó

Asszociatív és szekvenciális , konstans méretű, kevésbé átszervezendő adatszerkezet

2. Lista (sorozat)-Nem címezhető, hanem szekvenciális

- Elemtípusa bármi, akár lista is
- $[a_1, a_2,\ldots a_{i-1}][a_i, a_{i+1},\ldots a_n]$
- Fejrész, farok-rész

Műveletek

Létesítés | Megszüntet | Üres | Üres-e | Elejére

Végére | Tovább | Kiolvas | Módosít | Bővít

töröl | kapcsol

Műveletek

- Létesítés

$[][] \Rightarrow [a_1, a_2,\ldots a_{i-1}][a_i, a_{i+1},\ldots a_n]$

- Megszűntet

$[a_1, a_2,\ldots a_{i-1}][a_i, a_{i+1},\ldots a_n] \Rightarrow [][]$

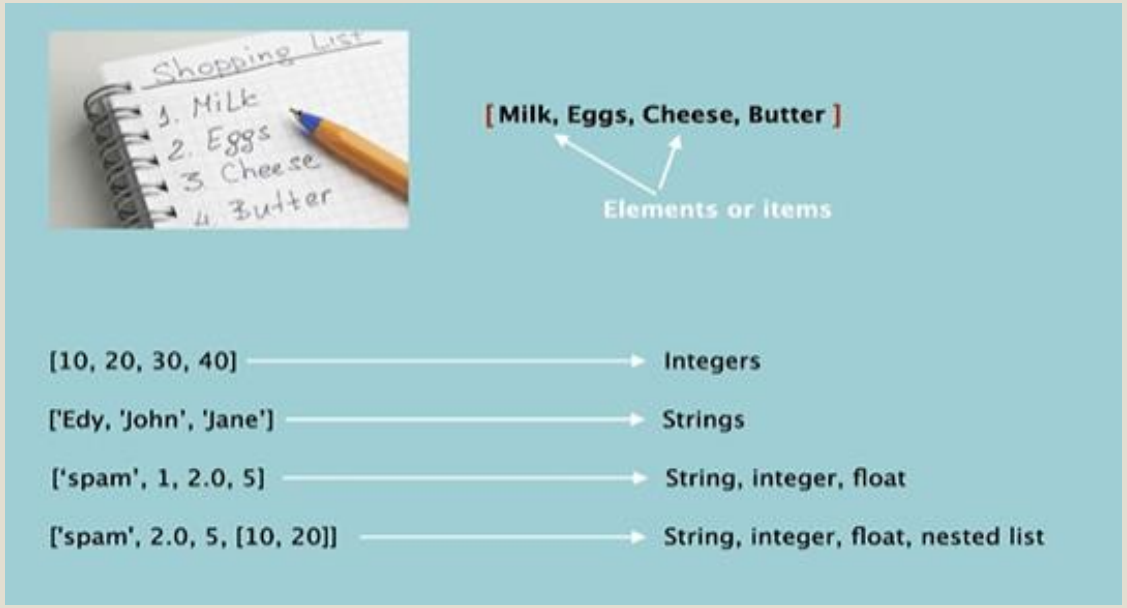
Adatszerkezetek – részletes tárgyalása

- Tömb Pytonban

```
from array import *
# 1. Create an array and traverse.
my_array = array('i',[1,2,3,4,5])
for i in my_array:
    print(i)
# 2. Access individual elements
through indexes
print("Step 2")
print(my_array[3])
```

Adatszerkezetek – részletes tárgyalása

- Lista Pytonban



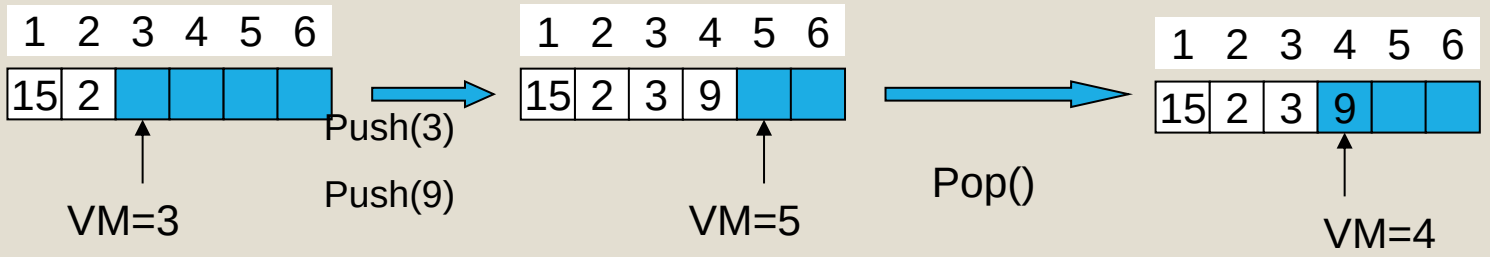

```
shoppingList = ['Milk', 'Cheese',
'Butter']

for i in range(len(shoppingList)):
    shoppingList[i] =
shoppingList[i]+"+"
    # print(shoppingList[i])
empty = []
```

Adatszerkezetek – részletes tárgyalása

- Verem és sor adatszerkezet
  - Fogalma
  - Műveletei
  - Implementációja
  - Érdekes algoritmusok vermekkel, sorokkal

Verem statikus megvalósítása tömb segítségével



- Jellemző adatok:
  - Veremmutató – stack pointer- az első üres elemre mutat
  - Veremméret – capacity
- Jellemző műveletek
  - Pop – (veremből)egy elem kivétele a veremből
  - Push – (verembe)egy elem verembe helyezése
  - Top – (tető) a következő kivehető adat olvasása
  - Empty – (üres) van-e adat a veremben
  - Full - (tele) tele van-e a verem

Verem műveletek megvalósítása

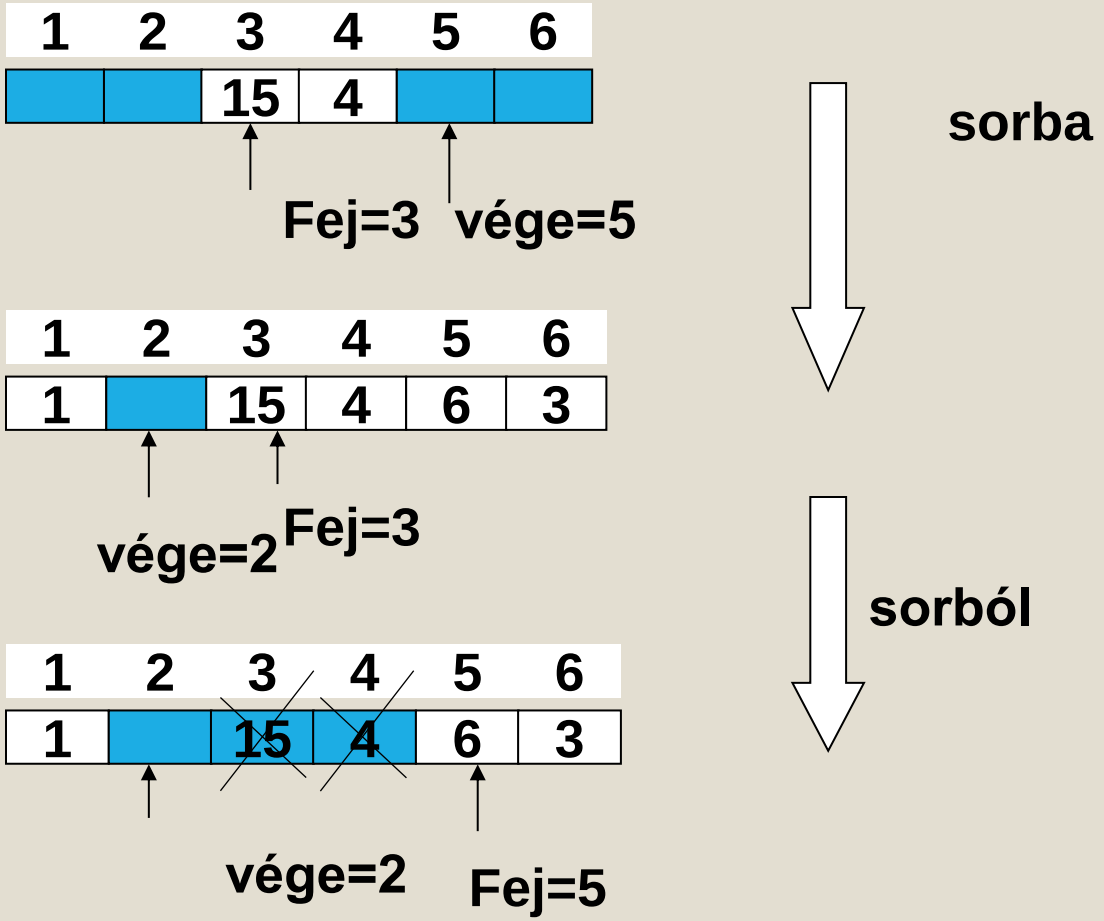
```
Logikai függvény Üres()
   ha veremmutató=0 akkor return igaz
   különben return hamis
Függvény vége
Eljárás verembe(mit)  // push()
   ha veremmutato>=veremmeret
   akkor hiba „túlcsordulás”
   ha veremmutató<veremméret akkor
      verem[veremmutato]=mit
      veremmutató=veremmutató+1;
   elágazás vége
Eljárás vége
```

```
Függvény veremből() //pop
   ha Üres() akkor hiba
   „alulcsordulás”
   ha nem Üres() akkor
     veremmutató=veremmutató-1;
     return verem[veremmutató]
   elágazás vége
Függvény vége
```

Sorok

- Jellemző adat
  - Sor feje : a kiolvasható adatra mutat
  - Sor vége: az első üres helyre mutat
  - Sor mérete //kapacitás
- Jellemző műveletek
  - Sorból
  - Sorba

Sor statikus megvalósítása tömb segítségével



Sorműveletek megvalósítása

```
Eljárás sorba(mit)
  ha Vége=Fej akkor hiba „túlcsordul”
  különben Sor[vége]=mit
  ha
  Vége=Méret-1 akkor Vége=0 egyébként Vége=Vége+1
  Elágazás vége
Eljárás vége

Függvény sorból
  X=S[Fej]
  ha Üres akkor hiba „alulcsordulás”
  Különben ha Fej=Méret-1 akkor Fej=0 különben Fej=Fej+1
  return X
Üres sor kezelése
  ha Fej=Vége akkor Fej=Méret-1 Vége=0
  Elágazás vége
Függvény vége
```

```
class Stack:
    def __init__(self):
        self.list = []
    def __str__(self):
        values = self.list.reverse()
        values = [str(x) for x in self.list]
        return '\n'.join(values)
    def isEmpty(self):
        if self.list == []:
            return True
        else:
            return False
    def push(self, value):
        self.list.append(value)
        return "The element has been
successfully inserted"
    def pop(self):
            if self.isEmpty():
                return "There is not any element
in the stack"
            else:
                return self.list.pop()
    def peek(self):
            if self.isEmpty():
                return "There is not any element
in the stack"
            else:
                return self.list[len(self.list)-
1]
    def delete(self):
            self.list = None
```

Klasszikus feladatok veremmel

- Matematikai kifejezések kiértékelése verem segítségével – lengyel forma létrehozása és kiértékelése

1. Lengyel forma létrehozása verem segítségével(prefix, infix, **postfix** alak)
   - Postfix alak jelentése: A két operandust követi az operátor
2. Kiértékelés verem segítségével

Lengyel forma létrehozásának szabályai

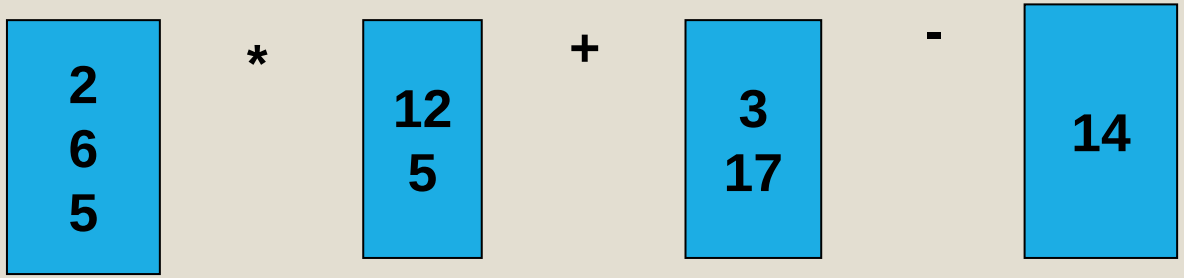
- A lengyelforma lényege, hogy nincsenek benne zárójelek és prioritási sorrendek, verem segítségével balról jobbra egyszerűen kiértékelhető.
- Az összetevőket szóköz választja el, a műveleti jel az operandusokat követi.
- Prioritási sorrend
  - 0 nem műveleti jelek
  - 1 (
  - 2 + -
  - 3 * /
  - 4)
- Szabály (az input és az output is string)

5+6*2-3 lengyelformája 5 6 2 * + 3 –

1. Az input sztring balról jobbra karakterenként kerül feldolgozásra.
2. A számok változatlan formában átkerülnek az outputra szóközzel elválasztva
3. A (bekerül a verembe
4. A) esetén a legelső (-ig kiürítjük a vermet, és a zárójeleken kívüli jelek az output sztringbe kerülnek
5. Egyéb műveleti jelnél a veremből kivesszük az összes, a jelenlegi jelnél nem kisebb prioritású műveletet, és az otput sztringbe írjuk, a jelenlegi műveleti jel a verembe kerül
6. Ha az input sztring végére értünk a verem tartalmát az outputhoz hozzáírjuk

Kiértékelés verem segítségével

5+6*2-3 lengyelformája 5 6 2 * + 3 –



A veremben lévő utolsó szám: végeredmény

ÖNÁLLÓ FELADAT

Készítse el a veremhez hasonlóan a sor adatszerkezet implementációját, és tesztelje.

Összefoglalás

- Adatszerkezetek csoportosítása
  - Szerkezet
  - Hozzáférés (statikus, nem statikus)
  - Programozhatóság(rekurzív, nem rekurzív)
- Sor adatszerkezet implementációja
  - Érdekes problémák sorokkal
- Verem adatszerkezet implementációja
  - Érdekes problémák veremmel

KÖSZÖNÖM A FIGYELMET!