



## 12. hét: ládarendezés, hash táblák

 Czirkos Zoltán, Frey Balázs ·  2023.11.16.  
Ládarendezés. Hash táblák építése.

Felkészülés a laborra:

- A [rendezésekről szóló előadás](#) áttekintése.

### Tartalom

- [Ládarendezés egész számokra](#)
- [Műszaki menedzsereknek](#)
- [Informatikusoknak: Hash tábla: vödrös hash](#)
- [Automatikus tesztek](#)
- [Hash tábla: a hash függvény cseréje](#)

### 1. Ládarendezés egész számokra

Ez az algoritmus szerepelt előadáson is. Vigyázz: nem az a kérdés, hogy ki tudod-e onnan másolni a kódot, hanem az, hogy meg tudod-e írni magad!

A ládarendezés (leszámláló rendezés) nem hasonlítja össze az egyes elemeket egymással, hanem nagyságuk szerint csoportosítja őket.

Lássuk a legegyszerűbb esetet, rendezzünk egy listát egész számokkal!



Ebben a listában 0 és 9 között vannak számok. Fogunk egy másik listát, amelyben leszámoljuk, hogy melyikből hány darab szerepel:

0	1	2	3	4	5	6	7	8	9
1	0	0	0	3	1	1	0	2	1

Ebből az információból egy új lista állítható elő, amelyik rendezett lesz. Nem kell hozzá mást tenni, mint 1 db 0-st, 3 db 4-est, 1 db 5-öst, 1 db 6-ost, 2 db 8-ast és 1 db 9-est tenni bele:



Írj programot, amely generál egy 0...99 véletlenszámokból álló, 100 elemű listát, majd a fenti algoritmussal rendezi azt! Ellenőrizd az eredményt „szemrevételezéssel”! (Jobb előbb tesztelni olyan kicsi bemeneten, amire a futási eredmény könnyen ellenőrizhető.) Ha működik, próbáld ki nagyobb listára is, akár 10000 vagy 100000 eleműre, és ellenőrizd az eredményt! Ehhez érdemes egy **rendezett\_e()** függvényt írni.

Fogd az előző programod, és a ládarendezést tedd át egy függvénybe! Vegye át a függvény a szokásos módon a rendezendő listát paraméterként.

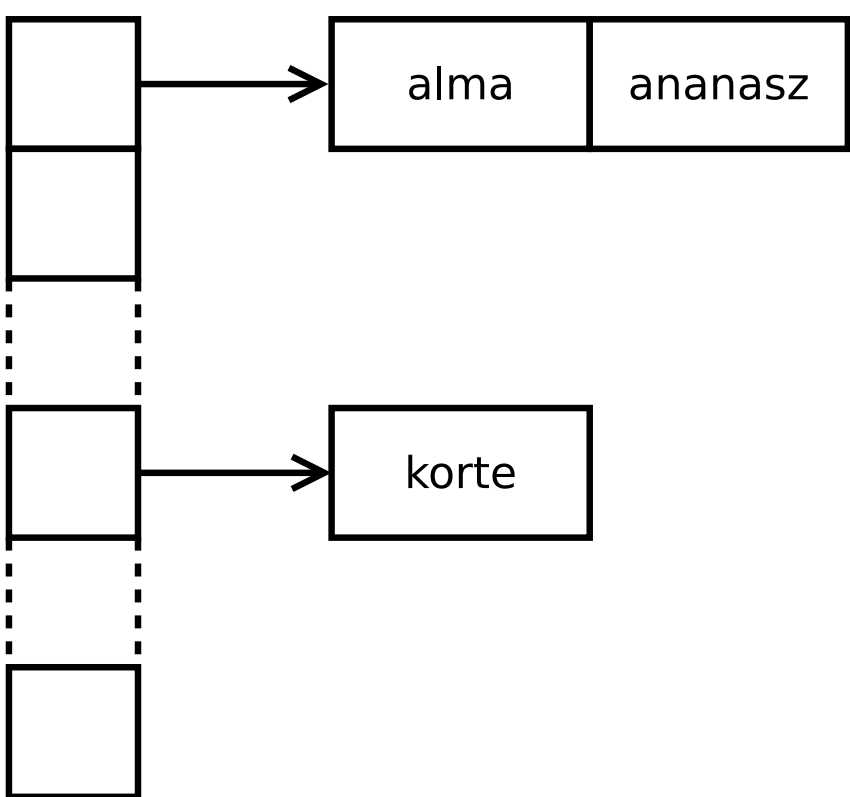
### 2. Műszaki menedzsereknek

Az ezután következő feladatok Informatikusoknak szólnak, mert [Algoritmusok és gráfok](#) tárgy anyagát illusztrálják.

Ha készen vagy a rendezős feladattal, dolgozz a házi feladatodon, vagy gyakorold a fájlkezelést ill. a bináris fák használatát a példatárból!

### 3. Informatikusoknak: Hash tábla: vödrös hash

Emlékezz vissza az [Algoritmusok és gráfok](#) tárgyban tanult hash táblákra! Azon belül is most konkrétan a vödrös hash-re. Ennek lényege az volt, hogy ütközések esetén az ütköző elemeket egy listába tesszük:



Implementálj egy ilyen hash táblát! Az egyszerűség és a szemléletesség kedvéért a megvalósítás működjön a következőképp:

- A hash tábla tároljon sztringeket. Tételezd fel, hogy a sztringekben csak ékezet nélküli, kisbetűs szavak vannak, például „alma”, „korte” és „barack”.
- A hash függvény legyen a szó első betűjének ábécébeli sorszáma: a=0, b=1, c=2 és így tovább. Emlékezz vissza a [karakterkódok kezelésére](#), ilyesmivel már találkoztál.
- Az ütközéseket könnyű elképzelni: minden ugyanolyan betűvel kezdődő szó ütközés lesz (pl. „alma”, „ananasz” és „avokado”). De ez nem baj, ennek kezelésére valók a vödrök.

A táblát halmazként fogjuk használni: be lehet tenni, ki lehet venni egy szót, és megnézni, hogy épp benne van-e a szó a táblában. Valósítsd meg az alábbi függvényeket:

- hash\_tabela\_letrehoz()**: létrehoz és visszaad egy hash táblát, ahol maga a táblázat létre van már hozva, és üres elemeket tartalmaz. (Vagyis egy olyan listát kell csinálnod, ami üres listákat tartalmaz. Hasonló lesz ez a [kétdimenziós listához](#), de itt a belsők kezdetben üresek.) Vajon hány elemű lesz a tábla, ha a fenti hash függvényt használsz?
- hash\_tabela\_betesz(tabela, szo)**: betesz egy szót a táblába. Ha már benne van, nem csinál semmit.
- hash\_tabela\_debug(tabela)**: kiírja a hash tábla tartalmát a kimenetre olyan formában, hogy az segítse a hibakeresést.
- hash\_tabela\_benne\_van(tabela, szo)**: megadja, hogy egy szó benne van-e a táblában. Ügyelj arra, hogy az ütközések miatt a vödrökben keresni kell majd.
- hash\_tabela\_kivesz(tabela, szo)**: kivesz egy szót a hash táblából. Ha nincs benne, nem csinál semmit.
- hash\_tabela\_listaz(tabela)**: kiírja a táblában tárolt összes szót ömlesztve.

Ha a kapott sztring alkalmatlan a hasheléshez (nem az „a...z” karakterek valamelyikével kezdődik), dobj kivételt! Ügyelj arra, hogy ne duplikáld a hash számító kódot, inkább írd egy függvényt hozzá!

Teszteld a kapott programod, ellenőrizd a helyes működését! Tegyé a hash tábládba azonos betűvel, és eltérő betűvel kezdődő szavakat is!

### 4. Automatikus tesztek

Az előző feladathoz kitaláltál egy műveletsort, amelyben megadott sorrendben kellett beszúrni, törölni, keresni elemeket a táblában. Például „alma betesz”, „barack betesz”, „alma betesz”, „barack kivesz” stb. Minden lépésnél adott volt, hogy milyen eredményt vársz.

Dolgozd át azt a teszt sorozatot egy automatikus tesztté! Használd ehhez a beépített **assert()** függvényt! Erre példát mutat az alábbi programocska:

```
def ltko(a, b):
    """Legnagyobb közös osztó, Euklidész algoritmusával."""
    while b != 0:
        t = b
        b = a%b
        a = t
    return a

def main():
    assert(ltko(30, 12) == 6)
    assert(ltko(12, 30) == 6)
    assert(ltko(35, 2) == 1)
    assert(ltko(3, 2) == 1)

main()
```

„Rontsd el” a programod, például módosítsd a keresőfüggvényed úgy, hogy mindig hamis értéket adjon! Próbáld ki így a tesztet!

### 5. Hash tábla: a hash függvény cseréje

Az előző feladatban használt hash függvény nem túl jó. Az mindig a szó első betűjét használja indexnek, viszont pl. sokkal-sokkal több „a” betűvel kezdődő szó van, mint ahány „x” betűvel kezdődő. Így aztán nem szór jól, nem ad nagyjából egyenletes eloszlást a táblázatban.

A Python viszont beépítve tartalmaz egy **hash()** nevű függvényt, amelyik viszont minden sztringhez egy jó nagy számot ad:

```
print(hash("alma"))      # 1808484321251193290
print(hash("barack"))    # 2498199152977029591
print(hash("dinnye"))    # -4940521116470653504
```

A kapott számok esetleg változhatnak is, ahányszor indítjuk a programot, és negatívak is lehetnek. Viszont egy futtatás közben mindig ugyanazok lesznek ugyanarra a sztringre.

Írd át úgy a programod, hogy ezt a hash függvényt használja! Ez már bármilyen sztringre jó, és így lehetővé válik az is, hogy a hash tábla méretét megválasszuk.

- Emlékezz vissza az [Algoritmusok és gráfok](#) tárgyból tanultakra! Hogyan képezzük le a hash függvény értékét a fix méretű táblázat indexeire?
- Próbáld ki a Python % operátort, hogy viselkedik, ha negatív számot kap osztandónak!
- Egészítsd ki a **hash\_tabela\_letrehoz()** függvényt egy méretet adó paraméterrel! Vagyis lehessen megadni a függvénynek, hogy mekkora a tábla.
- Módosítsd a többi függvényt, figyelembe véve a változtatásokat!

Teszteld az így kapott függvényeid működését!

