ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

ALGORITMUSOK ÉS ADATSZERKEZETEK

Python 4.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FÉLÉVES TEMATIKA

Oktatási hét	Témakör - Gyakorlat
1	Bevezetés a Python programozási nyelvbe, alapvető szintaxis és kifejezések, változók, típusok és alapvető műveletek, alapvető be- és kimenet, feltételes elágazások (if, elif, else)
2	Ciklusok (for, while), listák és egyszerű iterációk
3	Függvények és modulok, függvények definiálása és hívása, paraméterek és visszatérési értékek, beépített modulok használata, fájlkezelés, kivételkezelés
4	Algoritmusok és optimalizálási stratégiák rekurziós példán keresztül (brute force, dinamikus programozás, mohó algoritmusok, visszalépéses keresés)
5	Adatszerkezetek I. - Lista, tömb, sor, verem, listaműveletek, queue és stack implementációja statikusan (tömb) és dinamikusan (lista), gyakorlati feladatok
6	Adatszerkezetek II. – Halmaz és szótár, halmazműveletek és szótárműveletek, gyakorlati feladatok
7	**Zárthelyi dolgozat 1.**
8	Adatszerkezetek III. – Láncolt lista implementálása
9	Adatszerkezetek IV. – Bináris keresőfa implementálása
10	Gráfok alapjai, reprezentációik, gráfalgoritmusok implementálása
11	*Rektori szünet*
12	***Féléves beadandó feladat leadása és bemutatása***
13	**Zárthelyi dolgozat 2.**
14	**Félév zárása, javító/pótló zárthelyi dolgozat**

Algoritmusok és optimalizálási stratégiák

brute force, dinamikus programozás

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BRUTE FORCE STRATÉGIA

- Egyszerű, nem túl hatékony megoldási stratégia, amely a problémát az összes lehetséges megoldás kipróbálásával oldja meg.
  - *minden lehetséges megoldást végigpróbál, majd kiválasztja a megfelelőt vagy az optimálisat*
- **Előnyök:**
  - Egyszerű implementáció, és garantáltan megtalálja a megoldást, ha létezik.
- **Hátrányok:**
  - Nagyon lassú lehet *(hosszú futási idő),* különösen nagy bemenetek esetén, mivel az összes lehetséges kombinációt végig kell próbálni.

MINTAFELADAT – BRUTE FORCE ALGORITMUSRA HÁTIZSÁKPROBLÉMA

Hátizsák – Knapsack probléma

- Adott egy hátizsák, amelynek van egy maximális kapacitása.
- Van egy sor tárgy, mindegyiknek van egy súlya és egy értéke.

A cél az, hogy a hátizsákot úgy töltsük meg ezekkel a tárgyakkal,
hogy a hátizsákba kerülő tárgyak összsúlya ne haladja meg a hátizsák kapacitását,
és az összérték maximális legyen.

- A brute force megközelítésben az összes lehetséges kombinációt megvizsgáljuk, hogy megtaláljuk azt, amelyik maximális értéket ad.

BRUTEFORCE ALGORITMUS (REKURZÍV)

python_03_01_megoldás.py

```python
# Brutális erő algoritmus a hátizsákproblémára (n:targyak szama)
def hatizsak_bruteforce(sulyok, ertekek, kapacitas, n):  4 usages
    # Alapeset: ha nincs több tárgy vagy a kapacitás elérte a 0-t
    if n == 0 or kapacitas == 0:
        return 0, []
    # Ha a tárgy súlya nagyobb, mint a fennmaradó kapacitás, nem vehetjük fel
    if sulyok[n - 1] > kapacitas:
        return hatizsak_bruteforce(sulyok, ertekek, kapacitas, n - 1)
    else:
        # Két esetet hasonlítunk össze: a tárgyat felvesszük vagy nem
        val1, elemek1 = hatizsak_bruteforce(sulyok, ertekek, kapacitas - sulyok[n - 1], n - 1)
        val2, elemek2 = hatizsak_bruteforce(sulyok, ertekek, kapacitas, n - 1)
        # Mindkét rekurzív hívás eredményét tároljuk, és a nagyobb értéket választjuk
        if val1 + ertekek[n-1] > val2:
            # Ha a tárgyat felvesszük, akkor az indexét hozzáadjuk a listához
            elemek1.append(n-1)
            return val1 + ertekek[n-1], elemek1
        else:
            return val2, elemek2
```

PYTHON_03_01.PY

Feladat:

- Egészítsük ki a kódot a kommentek alapján és próbáljuk ki a hátizsákos függvényünket.

```
20
21  # Fő program, ahol megadjuk a bemeneti adatokat
22  ertekek = # adjunk meg értékeket [] zárójelben tetszőleges számban
23  sulyok = # ugyanannyi súlyt állítsunk be, minden értékhz tartozzon egy súly []
24  # definiáljuk a kapacitas értékét
25  n = len(ertekek)
26  max_ertek, kivasztott_elemek = hatizsak_bruteforce(sulyok, ertekek, kapacitas, n)
27  # irjuk ki az elérhető legnagyobb érték
28  # irjuk ki a  kiválasztott elemek indexeit:
```

BRUTE FORCE ÉS AZ IDŐ

Időkomplexitás:

- A brute force megoldás minden lehetséges tárgykombinációt végigpróbál.
- Ha n tárgy van, akkor a lehetséges kombinációk száma 2^n.
- Tehát a futási idő exponenciális: $O(2^n)$.

Miért lassú…?

- Mivel minden tárgy esetében két lehetőségünk van (felvesszük vagy nem), az összes lehetséges kombináció száma exponenciálisan nő a tárgyak számával.
- Nagy számú tárgy esetén ez nagyon hosszú futási időt eredményez.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

GYAKORLÓFELADAT – BRUTEFORCE

DOLGOZZATOK PÁROKBAN!

- Az összes lehetséges részhalmaz megtalálása egy lista elemeiből brute force módszerrel
- Feladat:
  - Írjatok egy rekurzív függvényt, amely megtalálja egy adott lista összes lehetséges részhalmazát.
  - Például, ha a lista [1, 2, 3],
  - akkor a részhalmazok: [ ] [1] [2] [3] [1, 2] [1, 3] [2, 3] [1, 2, 3]

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

AZ ALGORITMUS MŰKÖDÉSE

- A függvény rekurzívan működik, azaz önmagát hívja meg amíg el nem ér egy alapesetet.
- **Alap eset:**
  - Ha a bemeneti lista üres, akkor az egyetlen részhalmaza az üres halmaz.
- **Rekurzív eset:**
  - ***Első elem kiválasztása:*** Kiválasztjuk a lista első elemét.
  - ***Maradék részhalmazok:*** Rekurzív hívással meghatározzuk a lista többi elemének összes részhalmazát.
  - ***Kombinációk létrehozása:*** Az előző lépésben kapott részhalmazokhoz hozzáadjuk az első elemet, így újabb részhalmazokat kapunk.
  - ***Összes részhalmaz visszaadása:*** Összevonjuk az eredeti részhalmazokat és az új, az első elemmel kiegészített részhalmazokat, majd az így kapott teljes listát adjuk vissza.

python_03_02_megoldás.py python_03_02.py

PYTHON_03_02.PY

```python
def reszhalmazok(lista):  1 usage
    # Alapeset: üres lista esetén az egyetlen részhalmaz az üres halmaz
    # if üres listavizsgálat:
        # üres listával tér vissza
    else:
        # A lista első elemét kiválasztjuk
        # elso = ?
        # A maradék lista összes részhalmaza
        # maradek_reszhalmazok = ? rekurziv hívás
        # Létrehozzuk az összes lehetséges részhalmazt
        kombinaciok = []
        for halmaz in maradek_reszhalmazok:
            kombinaciok.append(halmaz)
            kombinaciok.append([elso] + halmaz)
        return sorted(kombinaciok, key=len) # részhalmaz hossza alapján rendez

# Példa meghívás:
print(reszhalmazok([1, 2, 3]))
```

Feladat:

- Egészítsük ki a kódot a leírtak alapján!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

DINAMIKUS PROGRAMOZÁS

- Ez egy optimalizációs technika, amelyet olyan problémák megoldására használunk, amelyek több kisebb, átfedő alproblémára bonthatók.
- A dinamikus programozás két kulcseleme:
  - **Memoizáció:**
    - Az alproblémák megoldásait elmentjük, hogy a későbbiekben újra felhasználhassuk őket, ezzel elkerülve az ismétlődő számításokat.
  - **Tabulation:**
    - Az alproblémákat iteratív módon oldjuk meg és tároljuk őket egy táblában, általában egy listában vagy mátrixban.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FIBONACCI SZÁMÍTÁS DINAMIKUS PROGRAMOZÁSSAL

- **Rekurzív Fibonacci-probléma:**
  - A rekurzív Fibonacci-algoritmus exponenciális időben fut, mivel sokszor számolja ki ugyanazokat az alproblémákat.

- **Optimalizálás memoizációval:**
  - A memoizáció során egy tárolót használunk a már kiszámított Fibonacci-értékek tárolására, így minden alproblémát csak egyszer számolunk ki.

REKURZÍV FÜGGVÉNY

python_03_04_megoldás.py | python_03_03_megoldás.py | p

```
1  def fibonacci_rek(n):  3 usages
2      if n == 0:
3          return 0
4      elif n == 1:
5          return 1
6      else:
7          return fibonacci_rek(n-1) + fibonacci_rek(n-2)
```

DINAMIKUS FÜGGVÉNY – LISTÁS MEGOLDÁS

python_03_04_megoldás.py | python_03_03_megoldás.py | python_03_03.py

```
9   def fibonacci_dinamikus(n):  1 usage
10      if n <= 1:
11          return [0]
12      else:
13          fib_numbers = [1, 1]
14          for i in range(2, n):
15              fib_numbers.append(fib_numbers[i-1] + fib_numbers[i-2])
16          return fib_numbers
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

DINAMIKUS FÜGGVÉNY – MEMOIZÁCIÓS MEGOLDÁS

python_03_04_megoldás.py | python_03_03_megoldás.py | python_03_03.py | proba

```
18  def fibonacci_dinamikus2(n, memo={}):  4 usages
19      if n in memo: # Ha az érték már megvan a memo-ban, akkor azt visszaadjuk
20          return memo[n]
21      if n <= 1:
22          return n
23      memo[n] = fibonacci_dinamikus2(n-1, memo) + fibonacci_dinamikus2(n-2, memo)
24      print(memo,end=", ") # memo felépítése
25      return memo[n]
```

PYTHON_03_03.PY

Feladat:

- Egészítsük ki a kódot (komment alapján) és értelmezzük a meglévő kódsorokat!

```
28  n = 10
29  print(f"dinamikus2: {fibonacci_dinamikus2(n)}")
30  print(f"dinamikus: {fibonacci_dinamikus(n)}")
31  # ciklus segítségével írjuk ki a rekurziv függvénnyel a számsor elemeit
```

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

GYAKORLÁS - FAKTORIÁLIS SZÁMÍTÁS PYTHON_03_04.PY

Feladat (párokban dolgozzatok)

- Faktoriális számítás pl 5!=1*2*3*4*5=120:
  - rekurzív módon
  - brute force startégiával
  - dinamikus programozási elveket követve
- A program jelenítse meg mindhárom elven n szám faktoriálisát.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

FELADAT PYTHON_03_05.PY

Feladat:

- Nézzünk utána a hátizsák probléma megoldásának dinamikus programozással
- Írjuk meg a kódot és lássuk el kommentekkel, melyek megmagyarázzak az algoritmus működését.
- Mutassuk be az adattárolást, az alproblémák tárolásának szemszögéből.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KÖSZÖNÖM

A MEGTISZTELŐ FIGYELMET!

Módné Takács Judit *modne.t.judit@amk.uni-obuda.hu*