Kőrösi Gábor

Algoritmusok és adatszerkezetek a gyakorlatban

Jelen tananyag a Szegedi Tudományegyetemen készült az Európai Unió támogatásával.

Projekt azonosító: EFOP-3.4.3-16-2016-00014

Szegedi Tudományegyetem
Cím: 6720 Szeged, Dugonics tér 13.
www.u-szeged.hu
www.szechenyi2020.hu

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

Mohó algoritmusok

Összefoglaló

A mohó algoritmus egy újabb részproblémákra bontáson alapuló módszer, melynek lényege, hogy minden lépésben egy lokálisan legjobb döntést hoz, de figyelmen kívül hagyja ennek a döntésnek a hatását a későbbi eseményekre. A mohó algoritmusok azoknál az optimalizálási feladatoknál használhatóak, ahol FELÜLRŐL LEFELÉ építkezve, minden lépésben meg tudjuk mondani a probléma megoldásához szükséges részproblémák közül melyik az az egy, aminek optimális megoldásából megépíthető a nagy probléma optimális megoldása. Mohó módon, csak azt az egy részfeladatot fogjuk megoldani, és az összes többit figyelmen kívül hagyjuk. Ezzel az algoritmussal számtalan feladatot oldhatunk meg, nagyon gyors, ám azt is ki kell hangsúlyoznunk, hogy nem minden probléma oldható meg mohó algoritmussal, hiszen a mohó választás nem mindig ad optimális megoldást.

Lecke fejezetei:

- A mohó algoritmus működése – Olvasó (5 perc)
- Pénzváltás probléma – Olvasó (5 perc)
- Levélfeladás – Olvasó (5 perc)
- Darabolható hátizsák probléma – Olvasó (15 perc)
- Huffman-kódolás – Olvasó (15 perc)
- Gyakorló feladatok – Gyakorlati (45 perc)

Téma típusa: Gyakorlati
Olvasási és gyakorlási idő: 90 perc

A mohó algoritmus működése (5 perc)

A mohó stratégia elemei:

1. A probléma optimális szerkezetének meghatározása.
2. Rekurzív megoldás kifejlesztése.
3. Annak bizonyítása, hogy minden rekurzív lépésben az egyik optimális választás a mohó választás. Tehát mindig biztonságos a mohó választás.
4. Annak igazolása, hogy a mohó választás olyan részproblémákat eredményez, amelyek közül legfeljebb az egyik nem üres.
5. A mohó stratégiát megvalósító rekurzív algoritmus kifejlesztése.
6. A rekurzív algoritmus átalakítása iteratív algoritmussá.
7.

Jellemzők:

1. A mohó algoritmus mindig az adott lépésben optimálisnak látszó választást teszi.
2. Csak egy részproblémát kell vizsgálni az optimális megoldáshoz.
3. Minden részproblémát felűről-lefelé haladó módon meg tudunk oldani.

Pénzváltás probléma (5 perc)

Egy nyugdíjas elmegy a postára, hogy felvegye 79.845 Ft-os nyugdíját. Hogyan tudja a postai pénztáros kifizetni neki ezt az összeget úgy, hogy a kifizetéshez a lehető legkevesebb darab pénzérmét/papírt használja?
Az alábbi címletek állnak rendelkezésre: 20.000, 10.000, 5.000, 2.000, 1.000, 500, 200, 100, 50, 20, 10 és 5 Ft-os. Tegyük fel azt is, hogy minden címletből korlátlan mennyiség van a postán.
Az nem tekinthető megoldásnak, ha a nyugdíjasnak vissza kell adnia (például a pénztáros ad 80.000 Ft-ot 4 db 20.000 Ft-os címlettel, majd a nyugdíjas visszaad 155 Ft-ot).

Mohó megoldás: Vegyünk a legnagyobb címletű pénzjegyből, amennyi szükséges, majd a maradék összeget fizessük ki a nála kisebb pénzjegyekkel!

A pénztáros az alábbi címleteket fogja adni:
3 db 20.000 Ft-os = 60.000 Ft
1 db 10.000 Ft-os = 10.000 Ft
1 db 5.000 Ft-os = 5.000 Ft
2 db 2.000 Ft-os = 4.000 Ft
1 db 500 Ft-os = 500 Ft
1 db 200 Ft-os = 200 Ft
1 db 100 Ft-os = 100Ft
2 db 20 Ft-os = 40 Ft
1 db 5 Ft-os = 5 Ft 79.845 Ft
Ez összesen 13 db papírpénzt, illetve pénzérmét jelent.

```
def findMin(V, ermek):

    n = len(ermek)
    ans = []                            # megoldashalmaz
    i = n - 1
    while(i >= 0):                      # vonjuk ki a aktualis erteket a pezoszeggbol, amig tudjuk
        while (V >= ermek[i]):
            V -= ermek[i]
            ans.append(ermek[i])        # adjuk hozza a kivont erteket a megoldashalmazhoz
        i -= 1
    return ans

ermek = [ 5, 10, 20, 50,100,200,500,1000,2000,5000,10000,20000]
megold = findMin(79845,ermek)           # teszteljuk a programot
for i in range(len(megold)):
            print(megold[i], end = " ")
```

Futási eredmény: 20000 20000 20000 10000 5000 2000 2000 500 200 100 20 20 5

Levélfeladás *(5 perc)*

Nyugdíjasunk most egy levelet szeretne feladni, amely 1400 Ft-ba kerül. Hogyan lehet ezt megtenni, ha a lehető legkevesebb számú bélyeget szeretnénk a borítékra tenni.

Rendelkezésre álló bélyegcímletek:
3.500 Ft
1.000 Ft
700 Ft
340 Ft
210 Ft
100 Ft
10 Ft

Ha a mohó algoritmust használjuk oly módon, hogy mindig a lehető legnagyobb címletű bélyeget ragasztjuk fel, akkor az alábbi megoldást kapjuk:
0 db 3.500 Ft-os = 0 Ft
1 db 1.000 Ft-os = 1.000 Ft
0 db 700 Ft-os = 0 Ft
1 db 340 Ft-os = 340 Ft
0 db 210 Ft-os = 0 Ft

0 db 100 Ft-os = 0 Ft
6 db 10 Ft-os = 60 Ft
1.400 Ft-os
Az algoritmus szerint 8 db bélyegre van szükségünk.

Könnyen látható, hogy ennél a feladatnál a mohó algoritmus által adott megoldás most nem optimális. Az optimális megoldás 2 db 700 Ft-os bélyeg választása lenne.

Darabolható hátizsák probléma *(15 perc)*

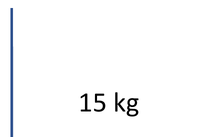
Egy adott hátizsákba tárgyakat szeretnénk pakolni. Adott n darab tárgy, ahol minden tárgynak van egy fontossági értéke (e[i]), és egy súlya (s[i]). Minden tárgyból 1 db áll rendelkezésre és a tárgyakat tetszőleges méretű darabokra vághatjuk. A hátizsákba maximum összesen S súlyt pakolhatunk. Az s[i] és S értékek egészek.
Szeretnénk úgy választani tárgyakat, hogy az összfontosság (profit) maximális legyen. Tehát feladatunk, hogy olyan tárgyakat válasszunk, amelyekre az összsúly nem haladja meg S-t azt, az összfontosság értéke pedig maximális.

Ez valójában egy maximalizálási probléma**, amelyre belátható, hogy a mohó algoritmus mindig optimális megoldást ad.**

n=7

m=15

Tárgy	1	2	3	4	5	6	7
Érték	10	5	15	7	6	18	3
Súly	2	3	5	7	1	4	1



Első lépésben meg kell határozni, hogy mi kerül a táskába. Hogyan kezdjünk ehhez?

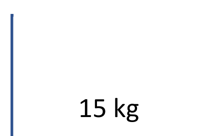
Tárgy	1	2	3	4	5	6	7
Érték	10	5	15	7	6	18	3
Súly	2	3	5	7	1	4	1
Bele							

Egy jó megoldás, ha kiszámítjuk a profit/súly hányadosát, melyet felhasználunk a megfelelő elemek kiválasztásához.

Tárgy	1	2	3	4	5	6	7
Érték	10	5	15	7	6	18	3
Súly	2	3	5	7	1	4	1
Arány	**5**	**1.6**	**3**	**1**	**6**	**4.5**	**3**
Bele							

Most nézzük meg, hogy hogyan tudjuk ezt megoldani MOHÓ algoritmussal. Első lépésben kiválasztjuk a legnagyobb profithányadosú elemet, és megnézzük, hogy mekkora súlykapacitásunk maradt még.

Tárgy	1	2	3	4	5	6	7
Érték	10	5	15	7	6	18	3
Súly	2	3	5	7	1	4	1
Arány	**5**	**1.6**	**3**	**1**	**6**	**4.5**	**3**
Bele					1		

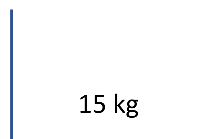


n=7 15-1=14

m=15

Ezután kiválasztjuk a második és harmadik legjobbat.

Tárgy	1	2	3	4	5	6	7
Érték	10	5	15	7	6	18	3
Súly	2	3	5	7	1	4	1
Arány	**5**	**1.6**	**3**	**1**	**6**	**4.5**	**3**
Bele	1				1	1	

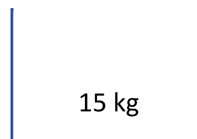


n=7 15-1-2-4=8

m=15

Majd a negyediket és az ötödiket (fontos, hogy mindvégig a hányadost figyeljük).

Tárgy	1	2	3	4	5	6	7
Érték	10	5	15	7	6	18	3
Súly	2	3	5	7	1	4	1
Arány	**5**	**1.6**	**3**	**1**	**6**	**4.5**	**3**
Bele	1		1		1	1	1

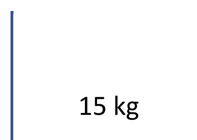


n=7 15-1-2-4-5-3=2

m=15

A következő lépésben viszont már nem tudjuk az egész „elemet” bepakolni, ezért annak csak egy részét tesszük bele.

Tárgy	1	2	3	4	5	6	7
Érték	10	5	15	7	6	18	3
Súly	2	3	5	7	1	4	1
Arány	**5**	**1.6**	**3**	**1**	**6**	**4.5**	**3**
Bele	1	2/3	1		1	1	1



n=7 15-1-2-4-5-3-(3*2/3)=0

m=15

Az utolsó elemet pedig nem akarjuk felhasználni.

Tárgy	1	2	3	4	5	6	7
Érték	10	5	15	7	6	18	3
Súly	2	3	5	7	1	4	1
Arány	**5**	**1.6**	**3**	**1**	**6**	**4.5**	**3**
Bele	1	2/3	1	0	1	1	1

Következő lépés, a profit/súly kalkulációja és az algoritmus igazolása.

Súly= {X_1+2/3*X_2+ X_3+0* X_4+ X_5+ X_6+ X_7} = 2+3*2/3+5+0*7+1+4+1 = 15kg
Profit= {X_1+2/3*X_2+ X_3+0* X_4+ X_5+ X_6+ X_7} = 10+5*2/3+15+0*7+6+18+3 = 15kg

Nézzük meg ezt forráskóddal is!

```
def hatizsak(suly,arany,m,megold):
    if (m==0):
        return megold
    max_ = arany.argmax()
    arany[max_] = 0
    if(m - suly[max_] > 0):
        megold[max_] = 1
        m= m - suly[max_]
    else:
        megold[max_] = m/suly[max_]
        m= m - (suly[max_]*m/suly[max_])
    return hatizsak(suly,arany,m,megold)

hatizsak(suly,arany,m,megold)

suly = np.array([2,3,5,7,1,4,1])
arany = np.array([5,1.6,3,1,        6,4.5,    3])
m = 15
megold = np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0])

print('A megoldas:')
hatizsak(suly,arany,m,megold)
```

```
A megoldas:
[1.,0.66666667,1.,0.,1.,1.,1.]
```

A mohó algoritmussal megoldott hátizsák feladat hibája, hogy nem darabolható esetekben nem az optimális eredményt kapjuk meg. Nézzük meg ezt is forráskóddal.

```
def hatizsak(suly,arany,m,megold,n):
    if (m==0) or (n==0):
        return megold
    max_ = arany.argmax()
    arany[max_] = 0
    if(m - suly[max_] > 0):
        megold[max_] = 1
        m= m - suly[max_]
    else:
        megold[max_] = 0
    return hatizsak(suly,arany,m,megold,n-1)

suly = np.array([2,3,5,7,1,4,1])
arany = np.array([5,1.6,3,1,6,4.5,3])
m = 15 #taska maxiumum tomege
n = len(suly)  #elemek szama
megold = np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0])

print('A megoldas:')
hatizsak(suly,arany,m,megold,n)
```

```
A megoldas:
[1., 0., 1., 0., 1., 1., 1.]
```

Jól látható, hogy ennél van jobb megoldás is, azonban a mohó algoritmusunk ezt figyelmen kívül hagyta, mivel az adott pillanatban mindig az optimálisnak tűnő megoldás választása erre „kényszeríti”.

Huffman-kódolás *(15 perc)*

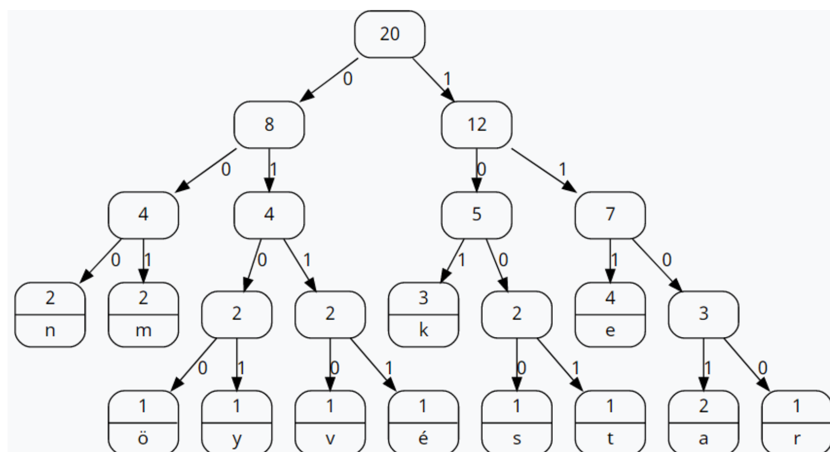
A mohó algoritmusok egy másik példája a Huffman-kódolás. Ennek az eljárásnak a célja, hogy egy szöveges fájlt rövidebb bitkódsorozattal helyettesítse (veszteség nélkül tömörítse).

Például a tömörítendő szöveg: „A könyvek néma mesterek."

Ehhez készítünk egy gyakorisági táblázatot:

Karakter	ö	y	v	é	s	t	r	a	n	m	k	e
Gyakoriság	1	1	1	1	1	1	1	2	2	2	3	4

A gyakorisági táblából úgy alakítjuk ki a kódoló bináris fát, hogy mindig a két legritkábban előforduló gyakorisági értéket kapcsoljuk össze, a két összekapcsolt elem helyére pedig betesszük a két összekapcsolt gyakoriság összegét.



Az így kialakult bináris fa ágaihoz akkor rendelünk nullát, ha a baloldali gyerekre mutat, valamint egyet, ha a jobboldali gyerekre mutat. Ez alapján az egyes karakterekhez változó hosszúságú, 8 bitnél rövidebb kódot tudunk rendelni. A gyakrabban előforduló karakterek kódja rövidebb lesz, mint a ritkábban előfordulóké. Az eredményben a karakterek helyett a hozzájuk rendelt kód jelenik meg.

Tehát a kódtáblánk:
e-111
k-101
m-001
n-000
a-1101
r-1100
t-1001
s-1000
é-0111
v-0110
y-0101
ö-0100

A kódolt szöveg:
1101 101 0100 000 0101 0110 111 000 0111 001 1101 001 111 1000 1001 111 1100 111 101
(A könyvek néma mesterek)

Megjegyzés: Attól függően, hogy a bináris fa felépítésénél egy adott lépésben melyik elem kerül balra és melyik jobbra, különböző eredményt kaphatunk, de ez nem befolyásolja a kapott kód hatékonyságát, illetve a megoldás helyességét.

Az eredeti szöveg mérete 20 byte volt (20 karakter)

A tömörítés utáni méret
111-3bit *4
101-3bit *3
001-3bit *2
000-3bit *2
1101-4bit *2
1100-4bit*1
1001-4bit *1
1000-4bit *1
0111-4bit *1
0110-4bit *1
0101-4bit *1
0100-4bit *1_____
69bit azaz 8,62 byte

Gyakorló feladatok *(45 perc)*

1. Nézzünk meg egy példát, amelyben a munkavégzés sorrendjét próbáljuk meghatározni, ha tudjuk, hogy egyes feladatoknak más-más az átadási határideje! Oldjuk meg ezt a feladatot mohó algoritmussal!

Munka	1	2	3	4	5
Nyereség	20	15	10	5	1
Határidő	2	2	1	3	3

Először meghatározzuk a lépések (részek) számát, melyek a következők: 0__1__2__3, tehát a „munkát” legkésőbb három lépésen belül be kell fejezni. Így a feladatunk valójában az, hogy kiválasszuk, hogy az öt munka közül melyik hármat akarjuk elvégezni (max profit).

Az érthetőbb megfogalmazás miatt jelöljék a számok az időt. Pl.: a bolt 9 órakor nyit, és egyes vevőknek 1, 2, 3 órán belül kell valami.

Mohó algoritmusunkban először válasszuk ki a legnagyobb profitú műveletet!

Munka	1	2	3	4	5
Nyereség	20	15	10	5	1
Határidő	2	2	1	3	3
Megoldás	1				

0 9.00 óra		1 10.00 óra	1. munka (20)	2 11.00 óra		3 12.00 óra

Majd a másodikat.

Munka	1	2	3	4	5

Nyereség	20	15	10	5	1
Határidő	2	2	1	3	3
Megoldás	1	1			

0 9.00 óra	 2. munka (15)	1 10.00 óra	 1. munka (20)	2 11.00 óra		3 12.00 óra

A harmadikat nem választhatjuk, hiszen annak határideje már lejárt, és emiatt csak a negyedik munkát tudjuk időben befejezni.

0 9.00 óra	 2. munka (15)	1 10.00 óra	 1. munka (20)	2 11.00 óra	 4. munka (5)	3 12.00 óra

Munka	1	2	3	4	5
Nyereség	20	15	10	5	1
Határidő	2	2	1	3	3
Megoldás	1	1	0	1	0

Számoljuk ki a profitot:
Profit= $\{M_1+M_2+ 0^* M_3+ M_4+ 0^*M_5\}$ = 20+15+0*10+5+0*1=40

Nézzük meg ezt forráskóddal is!

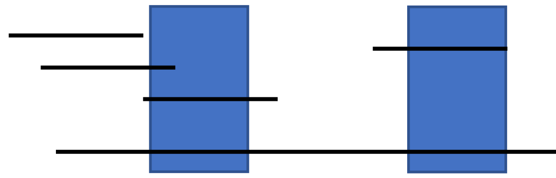
```
profit = np.array([20,15,10,5,1])
hatarido=np.array([2,2,1,3,3])
megold = [0,0,0,0,0]
nap = [0,0,0]

def munkautem(profit,hatarido,nap,megold):
    print(nap)
    if np.count_nonzero(nap)==len(nap):            # ha elertuk az utolso napot, akkor lepjunk ki
        return nap
    max_ = profit.argmax()                         # melyik a legnagyobb megszerezheto profit
    for i in range(hatarido[max_]-1,-1,-1):        # nezzuk meg, hogy van-e meg idonk erre a munkara
        print(i)
        if (nap[i]==0):
            nap[i]=profit[max_]
            profit[max_]=0
            return munkautem(profit,hatarido,nap,megold)
    else:
        profit[max_]=0                            # ha nincs akkor dobjuk ki, es nezzuk a kovetkezot
        return munkautem(profit,hatarido,nap,megold)


munkautem(profit,hatarido,nap,megold)
```

2. Oldjuk meg a hátizsák feladatot, úgy, hogy csak egész egységeket lehessen belerakni (pl.: a 2/3 mobiltelefonnak nincs semmi értelme).
3. Egy kábelhálózat különböző csatornáin N különböző filmet játszanak. Ismerjük mindegyik film kezdési és befejezési idejét. Egyszerre csak egy filmet tudunk nézni. Határozd meg, hogy maximum hány filmet nézhetünk végig, és melyeket!
4. Egy rendezvényre N darab vendég érkezik. Ismerjük mindegyik vendég érkezési és távozási idejét. A résztvevőket fényképeken szeretnénk megörökíteni. A fényképészt K perces időintervallumokra fizetjük. Határozd meg, hogy minimum hány intervallumra kell fizetni!

Segítség: A megoldás a lehetséges időpontok halmaza legkisebb, adott tulajdonsággal rendelkező részhalmazának kiválasztása. Akkor fényképezzünk, amikor feltétlenül szükséges! Ez azt jelenti, hogy amikor elmenne az első ember, aki még nem volt rajta egy fényképen sem, akkor kezdődik egy fényképezési intervallum.



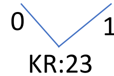
5. Mi a Huffman-kódja az alábbi ábécének:

Karakter	E	I	K	R	T	V
Gyakoriság	10	43	11	12	15	9

V	E	K	R	T	I
9	10	11	12	15	43

VE:19

V	E	K	R	T	I
9	10	11	12	15	43
	K	R	T	VE	I
	11	12	15	19	43



KR:23

V	E	K	R	T	I
9	10	11	12	15	43
	K	R	T	VE	I
	11	12	15	19	43
		T	VE	KR	I
		15	19	23	43

TVE:34

V	E	K	R	T	I
9	10	11	12	15	43
	K	R	T	VE	I
	11	12	15	19	43
		T	VE	KR	I
		15	19	23	43
			KR	TVE	I
			23	34	43

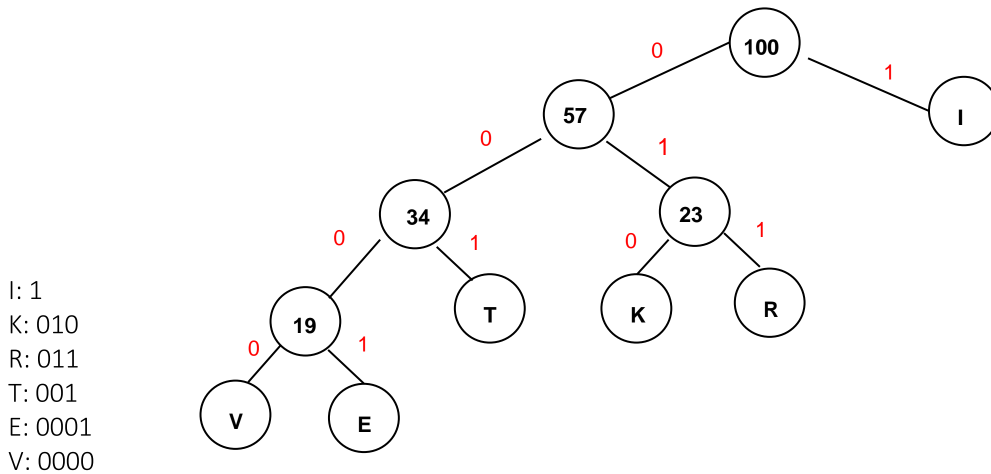
KRTVE:66

V	E	K	R	T	I
9	10	11	12	15	43
	K	R	T	VE	I

	11	12	15	19	43
		T	VE	KR	I
		15	19	23	43
			KR	TVE	I
			23	34	43
				I	KRTVE
				43	57

0 1

IKRTVE:100



I: 1
K: 010
R: 011
T: 001
E: 0001
V: 0000

Írj egy programot a példa megoldására!

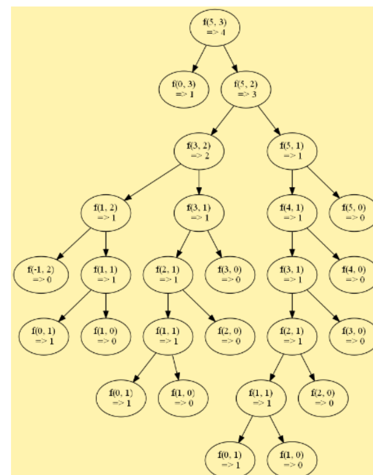
6. Módosítsd a feladatot úgy, hogy a vizualizálható legyen a pénzváltó algoritmus működése.

```
# Author: Bishal Sarang
from visualiser.visualiser import Visualiser as vs
@vs(ignore_args=["coins"], show_argument_name=False)

def f(coins, amount, n):
        if amount == 0:
            return 1
        if amount < 0:
            return 0
        if n <= 0 and amount >= 1:
            return 0
        include = f(coins=coins, amount=amount - coins[n - 1], n=n)
        exclude = f(coins=coins, amount=amount, n=n-1)
        return include + exclude

def main():
      amount = 5
      coins = [1, 2, 5]
      print(f(coins=coins, amount=amount, n=len(coins)))
      vs.make_animation("coin_change.gif", delay=3)

if __name__ == "__main__":
    main()
```

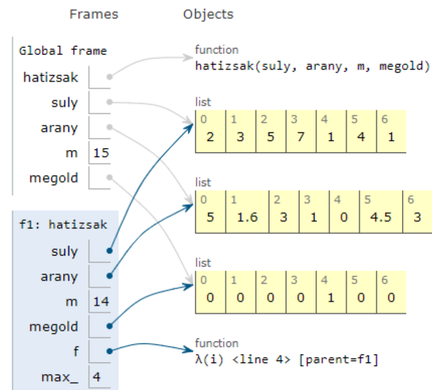


7. Vizualizáljuk a hátizsák pakolási feladatunkat a http://pythontutor.com/live.html oldal segítségével. Megjegyzés: az oldal nem tudja kezelni a „numpy” típusú adatokat, így kizárólag listákkal tudunk dolgozni.

```
def hatizsak(suly,arany,m,megold):
    if (m==0):
        return megold
    # max elem keresese a listaban
    f = lambda i: arany[i]
    max_=max(range(len(arany)), key=f)

    arany[max_] = 0
    if(m - suly[max_] > 0):
        megold[max_] = 1
        m= m - suly[max_]
    else:
        megold[max_] = m/suly[max_]
        m= m - (suly[max_]*m/suly[max_])
    return hatizsak(suly,arany,m,megold)

suly = [2,3,5,7,1,4,1]
arany= [5,1.6,3,1,6,4.5,3]
m= 15
megold=[0,0,0,0,0,0,0]
hatizsak(suly,arany,m,megold)
```



Ajánlott kitekintő anyag (155 perc)

Töredékes hátizsák feladat **(10 perc)** – Videó magyar nyelven

Mohó algoritmusok **(17 perc)** – Videó magyar nyelven

Rekurzív részproblémára bontási algoritmusok összefoglalása **(12 perc)** – Videó magyar nyelven

Tömörítési feladat **(17 perc)** – Videó magyar nyelven

Huffman kódolás **(24 perc)** – Videó magyar nyelven

Mohó algoritmusok **(20 perc)** – Gelle Kitti - link

Mohó algoritmus **(15 perc)** – Youtube link angol nyelven

Mohó algoritmusok **(40 perc)** – Zsakó László - link

SZÉCHENYI 2020

MAGYARORSZÁG KORMÁNYA

Európai Unió
Európai Szociális Alap

BEFEKTETÉS A JÖVŐBE