KOVANCI – POROČILO

Na vajah sem se naučila uporabljati rekurzijo na primeru dinamičnega programiranja. Naredila sem 2 nalogi v zvezi s kovanci v vrsti, in sicer optimalno vsoto kovancev in izpis vrednosti kovancev, pri katerih je dosežena optimalna vsota.

V sklopu kovanci v trikotniku pa sem naredila 3 naloge. Optimalna vsota v trikotniku, izpis vrednosti, kjer je ta vsota dosežena ter izpis poti z nizoma 'levo', 'desno'.

Kovanci v vrsti:

Uporabila sem rekurzivno idejo, kjer sem pogledala kaj se zgodi z vsoto če zadnji kovanec vzamemo ali ne.

*#Funkcija, ki vrne optimalno vsoto kovancev v vrsti***def** naj\_vsota1(K, i = **None**):  
 *'''optimalna vsota za prvih i kovancev'''* **if** i **is None**:  
 i = len(K)  
 **if** i == 0:  
 **return** 0  
 **if** i == 1:  
 **return** K[0]  
 **if** i == 2:  
 **return** max(K[0], K[1])  
 vzamemo = K[i-1] + naj\_vsota1(K, i-2)  
 nevzamemo = naj\_vsota1(K, i-1)  
 **return** max(vzamemo, nevzamemo)

Ideja je, da vsa števila pretvorimo v binarna (0 in 1). Recimo, da imamo tabelo [3,2,1,5,7] in število 10110 v binarnem zapisu. Tedaj je naša rešitev 3, 5, 7. Ker ne smemo jemati sosednjih elementov lahko to pogledamo tako, da se v binarnem zapisu ne smeta pojaviti dve zaporedni enki. Takrat lahko primer takoj prekinemo.

*#Funkcija, ki vrne vrednosti kovancev, pri katerih je dosežena optimalna vrednost***def** vrednosti\_opt\_vsota(tab):  
 *'''vrne vrednosti kovancev, pri katerih je dosezena optimalna vsota.  
 Pri tem ne smemo vzeti sosednjih dveh kovancev'''* opt\_vsota = 0  
 **for** i **in** range(2\*\*(len(tab))):  
 niz = bin(i)[2:] *#stevilo spremenimo v binarno in odrežemo '0b'* **if '11' not in** niz: *#preverimo da ne vzamemo dveh zaporednih stevil* **if** len(niz) < len(tab): *#nizu moramo na zacetek dodati ustrezno st. nicel* niz = (len(tab)-len(niz))\***'0'** + niz  
 tab\_nizov = list(niz)  
 nova\_vsota = 0  
 **for** i, el **in** enumerate(tab\_nizov):  
 **if** el == **'1'**: *#pristejemo element, ker je pri vsoti izbran* nova\_vsota += tab[i]  
 **if** nova\_vsota > opt\_vsota: *#ce smo nasli boljso vsoto, obstojeco najboljšo zamenjamo* opt\_vsota = nova\_vsota  
 max\_niz = niz *#zapomnimo si indekse izbranih kovancev* vrednosti\_izbranih\_kovancev = []  
 **for** i, el **in** enumerate(max\_niz):  
 **if** el == **'1'**: *#ce je kovanec v resitvi si zapomnimo njegovo vrednost* vrednosti\_izbranih\_kovancev += [tab[i]]  
 **return** vrednosti\_izbranih\_kovancev

Kovanci v trikotniku:

V nalogah sem uporabila idejo, da sem za vsak vzet kovanec preverila njegovega levega in desnega sina ter vzela tistega, ki je bil večji.

*#Funkcija za optimalno vsoto kovancev v trikotniku.***def** naj\_vsota(trikotnik, m, n):  
 *'''najvecja vsota kovancev v trikotniku'''* **for** i **in** range(m-1, -1, -1): *#pogledamo trikotnik od spodaj gor* **for** j **in** range(i + 1):  
 *#za vsak element pogledamo obe vrednosti in izpisemo maksimum* **if** (trikotnik[i + 1][j] > trikotnik[i + 1][j + 1]): *#ce je vrednost na levi strani vecja, jo pristejemo vsoti* trikotnik[i][j] += trikotnik[i + 1][j]  
 **else**:  
 trikotnik[i][j] += trikotnik[i+1][j+1]  
 **return** trikotnik[0][0]

Ideja: Trikotnik definiramo kot matriko. Primer, če ima trikotnik največ 5 nivojev, bo to 5x5 matrika.

trikotnik5 =[[3,0,0,0,0],  
 [1,4,0,0,0],  
 [2,4,6,0,0],  
 [8,9,9,3,0],  
 [1,18,3,1,3]]

Ko pogledamo cel trikotnik, vrnemo optimalno rešitev te matrike.

**def** vsota(trikotnik):  
 *'''vrne optimalno pot zapisano v podseznamih'''* pot = [[**' ' for** i **in** range(j+1)] **for** j **in** range(len(trikotnik)-1)]  
 **if not** trikotnik:  
 **return** 0  
 **if** len(trikotnik) == 1:  
 **return** trikotnik[0][0]  
 **for** i **in** range(len(trikotnik)-2,-1,-1):  
 **for** j **in** range(i+1):  
 trikotnik[i][j] = max(trikotnik[i][j]+trikotnik[i+1][j],trikotnik[i][j]+trikotnik[i+1][j+1])  
 **if** trikotnik[i+1][j] > trikotnik[i+1][j+1]:  
 pot[i][j] = **'levo'  
 else**:  
 pot[i][j] = **'desno'  
 return** pot

*#Funkcija za izpis optimalno pot, pri kateri je dosežena optimalna vsota. Funkcija izpiše niza 'levo', 'desno'.*

**def** optimalna\_pot (trikotnik):  
 *'''vrne opis optimalne poti z nizoma 'levo', 'desno' '''* pot = vsota(trikotnik)  
 tab = list()  
 j = 0  
 **for** i **in** range(len(pot)):  
 **if** pot[i][j] == **'levo'**: *#ce je optimalno vozlišče na levi, gremo na levo* tab.append(**'levo'**)  
 **if** pot[i][j]== **'desno'**: *#ce je optimalno vozlišče na desni, gremo na desno* j+=1  
 tab.append(**'desno'**)  
 **return** tab

*#Funkcija za izpis vrednosti kovancev v trikotniku, pri katerih je dosežena optimalna vsota.*  
  
**def** kateriKovanci (trikotnik):  
 *'''vrne vrednost kovancev, pri katerih je dosežena optimalna vsota'''* cel\_trikotnik = [[trikotnik[j][i] **for** i **in** range(j+1)] **for** j **in** range(len(trikotnik))] *#cel trikotnik* pot = vsota(trikotnik)  
 tab = list() *#v tabeli bomo shranjevali vrednosti* j = 0  
 tab = [cel\_trikotnik[0][0]]  
 **for** i **in** range(len(pot)):  
 **if** pot[i][j] == **'levo'**: *#ce se optimalen kovanec nahaja levo* tab.append(cel\_trikotnik[i+1][j]) *#ga dodamo v tabelo* **if** pot[i][j]== **'desno'**: *#ce se optimalen kovanec nahaja desno* j+=1  
 tab.append(cel\_trikotnik[i+1][j]) *#ga dodamo v tabelo* **return** tab