UNIVERSITETI I PRISHTINËS

FAKULTETI I INXHINIERISË ELEKTRIKE DHE KOMPJUTERIKE

Nën-vargu me shumën maksimale

Lënda: Algoritmet e avancuara

Mentori:

Prof. Asoc. Dr. Kadri Sylejmani

Ass. Admir Kadriu

Studentet:

Arianit Lubishtani

Blend Arifaj

Tabela e përmbajtjeve

1. Hyrje

Në këtë dokument është përshkruar zgjidhja e problemit “Nen-vargu me shumen maksimale” duke e shfrytëzur teknikën “Përcaj dhe sundo”. Zgjidhja e problemit është bërë në gjuhën programuese Python, duke e përdorur editorin PyCharm.

Në fillim është përshkruar zgjidhja e këti problemi në gjuhen natyrale (në formë të pseudo-kodit), ku pastaj është kthyer në gjuhen programuese Python. Pas arritjes së zgjidhjes së problemit është caktuar kompleksiteti kohor dhe ai hapsinor i këti algoritmi, duke shfrytëzur notacionet Big O dhe Big Omega.

2. Përshkrimi i problemit

Përmes teknikes “Përçaj dhe sundo”, të dizajnohet algoritmi qe gjene shumen e nen-vargut qe ka shumen më të madhe nëse secili anëtar i tije mblidhet me tjetrin. Të tregohet kompleksitetit i algoritmit te dizajnuar përmes notacionit Big O ose Big Theta, si në aspektin kohor, ashtu edhe në aspektin hapësinor.

Shembull:

Input : [-1 -4 7 -3 -2 1 3 -8]

Output : 6 // [-1 -4 7 -3 -2 1 3 -8]

3. Zgjidhja e problemit

Duke e përdorur teknikën “Përcaj dhe sundo” do të bëjmë ndarjen e vargut në vargje me të vogla, respektivisht, më nga një anëtar. Pasi të jetë bërë ndarja atëhere do të kërkohet vlera maksimale e anës se majtë, ajo e anës se djatht si dhe vlera e shumës maksimale në mes të atyre dy vargjeve.

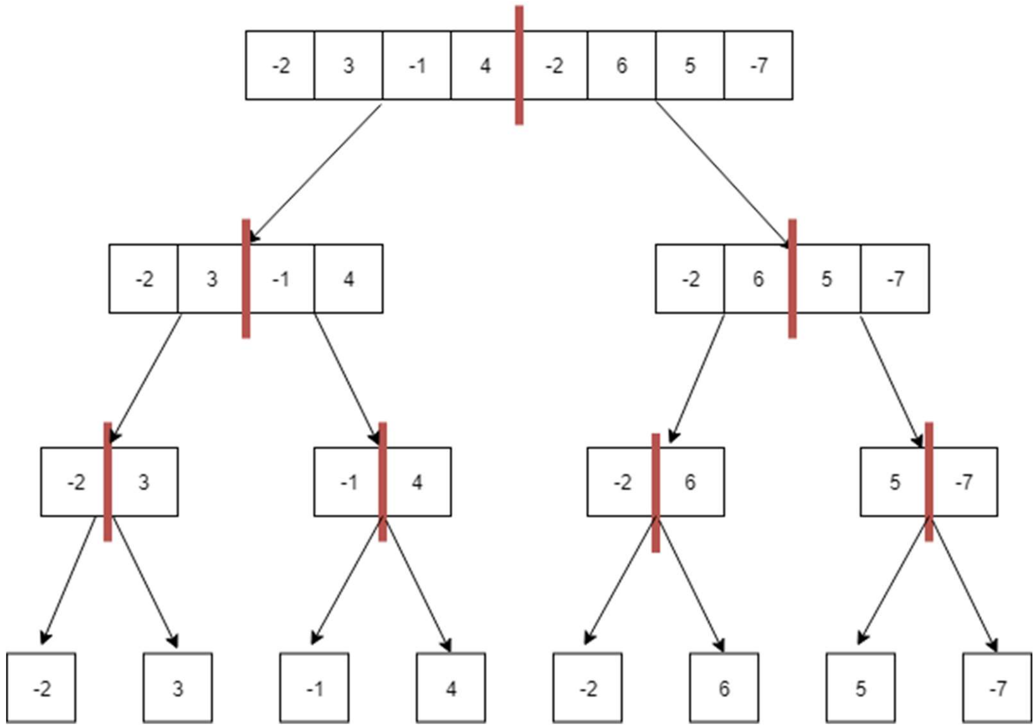

Figura 1: Ndarja e vargut në vargje më nga një element

Në Figura 1 është paraqitur ndarja e vargut në nënvargje me nga vetëm një element. Pasi që është coptuar vargu, atëhere e bëjmë llogaritjen e vlerës maksimale në mes të vleres maksimale të anëtarit në anën e majt, vlerës maksimale të anëtarit në anën e djatht si dhe shumës maksimale në mes të këtyre vargjeve. Kjo llogaritje është paraqitur në Figura 2.

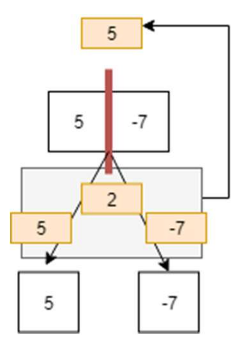

Figura 2: Kthimi i vlerës maksimale

Pra, vargu i copëtur (me vetëm një anëtar) e kthen vleren e vetme të anëtarit, pasi që vlera maksimale është po ajo vlerë. Shuma maksimale në mes të anëtareve është shuma maksimale në anën e djatht me atë në anën e majt, pra në rastin tonë shuma maksimale e anës së majtë është 5 kurse ajo e anës djathtë është -7. Nga rrjedh që shuma maksimale në mes të anëtarëve është 2. Pasi që i kemi gjetur të tri vlerat e kërkuara, e gjejmë vleren maksimale në mes të ketyre, dhe ky funksion e kthen këtë vlerë maksimale deri kur nuk ka vargje tjera të copëtuara.

Në Figura 3 është paraqitur komplet shembulli, deri në gjetjen e shumës maksimale në varg.

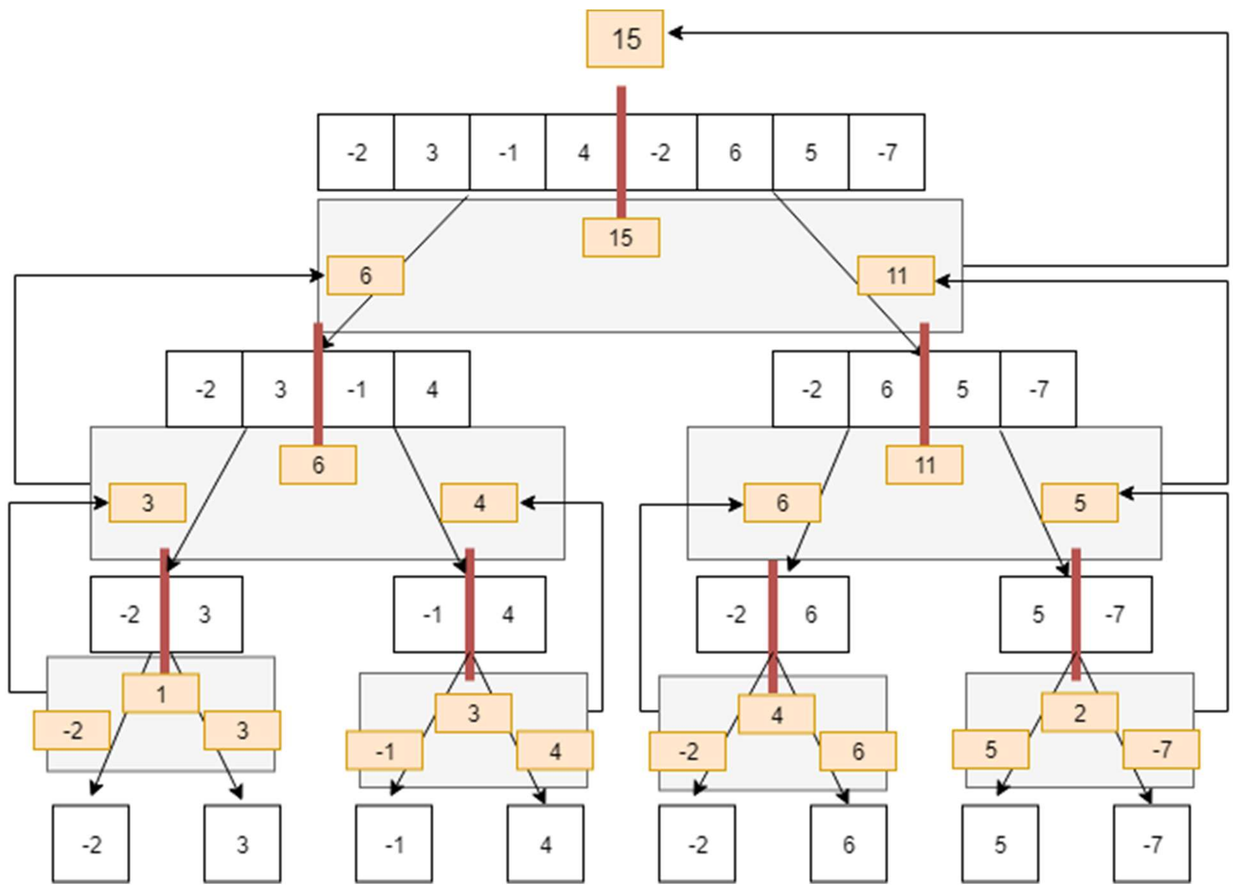

Figura 3: Llogaritja e shumës maksimale në varg

3.1. Pseudo-kodi i algoritmit

Nga pjesa e sipërme mundëm të verejme se si bëhet caktimi i shumës maksimale në një varg. Nga kjo do të caktojm pseudo-kodin e algoritmit, i cili do të bëjë caktimin e kësaj vlerë për qfardo vargu të caktuar. Në Figura 4 është paraqitur ky pseudo-kod.

```
vargu = {elementet}
fillimi <- 0
gjatesia <- len(vargu)
ShumaMaksimaleNeVarg(vargu,fillimi,gjatesia):
if fillimi == gjatesia:
return vargu[1]
mesi = (fillimi+gjatesia)//2
return max(ShumaMaksimaleNeVarg(fillimi,mesi),ShumaMaksimaleNeVarg(
mesi+1,gjtesia),ShumaMaksimaleNdermjet(fillimi,mesi,gjatesia))

ShumaMaksimaleNdermjet(vargu,fillimi,mesi,gjatesia):
temp <- 0
shumaMajtas <- infinit
for i in range(mesi,fillimi-1,-1):
temp <- temp + vargu[i]
if temp > shumaMajtas:
shumaMajtas = temp

temp <- 0
shumaDjathtas <- infinit
for i in range(mesi+1,gjatesia+1,1):
temp = temp + vargu[i]
if temp> shumaDjathtas:
shumaDjathtas = temp

return shumaMajtas + shumaDjathtas
```

Figura 4: Pseudo-kodi i algoritmit

3.2. Implementimi i algoritmit ne Python

Implementimi është bërë duke e shfrytëzur paradigmën e orientuar në objekte, ky është krijuar një klasë më emrin DetyraEPare e cila e ka konstruktorin ku bëhet inicializimi i variablave të nevojshme. Gjithashtu kjo klasë ka edhe dy metodat me anë të te cilave është realizuar algoritmi. Në Figura 5 është paraqitur klasa DetyraEPare.

```python
class DetyraEPare:
    def __init__(self,_array):
        self.array = array
        self.vleraMaksimale = self.shumaMaksimaleDC(0,len(self.array)-1)
        print("Shuma maksimale qe eshte gjendur ne ",self.array," eshte : ",
              self.vleraMaksimale)

    def shumaMaksimaleDC(self,_from,_to):
        if _from == _to:
            return self.array[1]
        _middle = (_from+_to)//2
        return max(self.shumaMaksimaleDC(_from,_middle),
                   self.shumaMaksimaleDC(_middle+1,_to),
                   self.shumaMaksimaleNdermjet(_from,_middle,_to))
    def shumaMaksimaleNdermjet(self,_from,_middle,_to):
        tempCount = 0
        shumaMajtas = -100000000
        for i in range(_middle,_from-1,-1):
            tempCount = tempCount + self.array[i]
            if(tempCount > shumaMajtas):
                shumaMajtas = tempCount

        tempCount = 0
        shumaDjathtas = -100000000
        for i in range(_middle+1,_to+1):
            tempCount = tempCount + self.array[i]
            if tempCount > shumaDjathtas:
                shumaDjathtas = tempCount
        return shumaMajtas+shumaDjathtas
```

Figura 5: Implementimi i klasës në Python

3.3. Testimi i Algoritmit

Në Figura 6 është paraqitur testimi i algoritmit, ku ne hyrje është vendosur vargu me vlerat -2, 3, -1, 4, -2, 6, 5 dhe -7.

```python
vargu = [-2, 3, -1, 4, -2, 6, 5,-7]
obj = DetyraEPare(vargu)
```

```
Shuma maksimale qe eshte gjendur ne vargun  [-2, 3, -1, 4, -2, 6, 5, -7]  eshte :  15
```

Figura 6: Testimi i algoritmit

4. Kompleksiteti i algoritmit

4.1. Kompleksiteti kohor

Për ta përcaktuar kompleksitetin kohore në menyrë më të sakt, duket të identifikojmë operacionin primar apo ndryshe operacionin i cili përseritet më se shpeshti. Në rastin tonë kemi një funksion i cili e therret vetveten si dhe nje funksion tjeter, dhe e cakton vleren maksimale në mes të rezultateve, pra:

T(n) = T(n/2) + T(n/2) + C(n) = 2T(n/2) + n

Me T(n/2) kemi shenuar kompleksitetin qe do te kemi pasi qe funksioni e therret vetveten, kurse me C(n) kemi kompleksitetin e funksioni ShumaMaksimaleNdermjet. Funksioni ShumaMaksimaleNdermjet ka një kompleksitet Big Theta(2n+c)=Big Theta(n), pasi që e bënë mbledhjen e secilit anëtar dhe krahasimin, dhe me raste edhe kemi vendosje të vlerës.

Nëse vazhdojmë të zgjedhim këtë barazim, do të kemi:

$T(n) = 4T(n/4) + 2n = ... = 2^k T(n/2^k) + kn$

$T(1) = 0 => 2^k = n => k = \log_2 n$

$T(n) = \log_2 n + n\log_2 n = n\log_2 n$

Duke u bazuar në tri rastet e mundëshme[1]:

$$\lim_{n\to\infty}\frac{T(n)}{G(n)} = \begin{cases} 0 - implikon\ \text{që}\ T(n)\ e\ ka\ \text{një}\ ngrite\ \text{më të}\ vogel\ se\ G(n) \\ c - implikon\ \text{që}\ T(n) e\ ka\ \text{një}\ ngritje\ \text{të}\ \text{njejtë}\ me\ G(n) \\ \infty - implikon\ \text{që}\ T(n)\ e\ ka\ \text{një}\ ngritje\ \text{më të}\ madhe\ se\ G(n) \end{cases}$$

për rastin tonë kemi $T(n) = n\log_2 n$, kurse C(n)= nlogn.

$$\lim_{n\to\infty}\frac{T(n)}{G(n)} = \lim_{n\to\infty}\frac{\text{nlog2n}}{\text{nlogn}} = \lim_{n\to\infty}\frac{\text{log2n}}{\text{logn}} = \lim_{n\to\infty}\frac{\frac{logn}{log2}}{\text{logn}} = \lim_{n\to\infty}\frac{logn}{\text{logn log2}} = \lim_{n\to\infty}\frac{1}{\text{log2}} = 3$$

Pasi që $\lim_{n\to\infty}\frac{T(n)}{G(n)}$=3, pra me një konstante, mund të themi që kompleksiteti kohore i këti algoritmi ështe Bit Theta(nlogn)

4.2. Kompleksiteti hapsinor

Për ta përcaktuar kompleksitetin hapsinore duhet të shohim variablat shtesë të cilat i kemi përdorur për ta bërë zgjidhjen e problemit (duke mos i llogaritur inicializimet e klasës).

Kompleksiteti hapsinor i algoritmit është i barabartë me shumën e kompleksitetit hapsinor të te dy funksioneve (ShumaMaksimaleDC dhe ShumaMaksimaleNdermjet). Te funksioni ShumaMaksimale DC shohim që kemi përdorur vetëm një varibël shtesë, _middle. Pra, kompleksiteti i këtij funksioni është Big O(1). Te funksioni ShumaMaksialeNdermjet kemi përdorur 4 variabla shtesë (tempCount, shumaMajtas, shumaDjathtas dhe i). Më këtë konstatojm se kompleksiteti hapsinor i këtij funksioni është Big O(4) = Big O(1).

Kompleksiteti hapsinor i algoritmit është Big O(1), pra është konstant.

5. Konkluzion

Përdorimi i teknikës “Përcaj dhe sundo” për të gjetur shumën maksimale në një varg ka një kompleksitet konstant hapsinor, dhe një kompleksitet linear-logaritmik kohore, që e përmirëson kompleksitetin kohor me zgjidhje “tradicionale” me dy unaza (kompleksiteti kohore kuadratik).

6. Shtesat

7. Referencat

[1] A. Levitin. *Introduction to “The Design and Analysis of Algorithms” 3rd Edition.* Page: 57.