Oblikovanje i analiza algoritama - Walsh–Hadamard transformacija

The *Hadamard matrices* $H_0, H_1, H_2, \ldots$ are defined as follows:

- H_0 is the 1×1 matrix $[1]$
- For $k > 0$, H_k is the $2^k \times 2^k$ matrix

$$H_k = \left[\begin{array}{c|c} H_{k-1} & H_{k-1} \\ \hline H_{k-1} & -H_{k-1} \end{array}\right]$$

Show that if v is a column vector of length $n = 2^k$, then the matrix-vector product $H_k v$ can be calculated using $O(n \log n)$ operations. Assume that all the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time.

1.Iterativna varijanta ili Brza Walsh–Hadamard transformacija i opis algoritma

Uzmimo vektor v= $[v_1, v_2, v_3, v_4, \ldots, v_n]$ dimenzije n = 2^m

NAP: $s\ v^{(1)}\ i\ v^{(2)}\ su\ oznacene\ polovice\ od\ nekog\ vektora\ v$

OPIS KORAKA

Iterativna verzija ide odozdo:

-računa $H = \begin{pmatrix} v_1 + v_2 \\ v_1 - v_2 \end{pmatrix}$ za $\forall v_1$ neparan i njegov sljedb. v_2

Takvih H-ova ima $\frac{n}{2}$ i svaki izračuna 1 zbr. i 1 oduz.

-zatim se računa na dubini 2:

$\begin{pmatrix} w_1 + w_3 \\ w_2 + w_4 \\ w_1 - w_3 \\ w_2 - w_4 \end{pmatrix}$ gdje su w_i dobiveni iz prethodne dubine

Takvih H-ova ima $\frac{n}{4}$ i svaki izračuna 2 zbr. i 2 oduz.

.....

Na dubini m:

Postoji samo jedan H i on računa $\frac{n}{2}$ zbr. i $\frac{n}{2}$ oduz.

KORACI

$$H_m(v) = \begin{pmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{pmatrix} \begin{pmatrix} v^{(1)} \\ v^{(2)} \end{pmatrix}$$

Dubina m :
Dim(v)= 2^m

$$H_1(v^{(1)} =: q) = \begin{pmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{pmatrix} \begin{pmatrix} q^{(1)} \\ q^{(2)} \end{pmatrix} \qquad H_1(v^{(2)}) =..$$

Dubina m-1 :
Dim(v)= 2^{m-1}

.
.
.

$$H_2(v) = \begin{pmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{pmatrix} \begin{pmatrix} v^{(1)} \\ v^{(2)} \end{pmatrix} = \begin{pmatrix} H_1(v^{(1)}) + H_1(v^{(2)}) \\ H_1(v^{(1)}) - H_1(v^{(2)}) \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \end{pmatrix}$$

Dubina 2 :
Dim(v)=4

$$H_1(v^{(1)}) = \begin{pmatrix} v_1 + v_2 \\ v_1 - v_2 \end{pmatrix} \qquad H_1(v^{(2)})..$$

Dubina 1 :
Dim(v)=2

```python
def pomnoziFast(a) -> None:
    h = 1
    T=0
    while h < len(a):
        for i in range(0, len(a), h * 2):
            for j in range(i, i + h):
                x = a[j]
                y = a[j + h]
                a[j] = x + y

                a[j + h] = x - y
                T=T+2
        h *= 2
    return a,T
```

Kako se to veže na naš konkretni algoritam?

h nam odabire dubinu na kojoj se nalazimo. $h = 1$ zapravo glumi dubinu 1 u koracima. Kad je $h \neq 1$, i će zapravo glumiti pojedinu matricu u na dubini h, a j pojedini redak te matrice i. Algoritam tako radi identične operacije kao kod rekurzivne varijante.

(T je u algoritmu pomoćna varijabla koja broji korake)

2.Rekurzivna varijanta i Opis algoritma

U prvom koraku ,prog. mora napraviti još n dodatnih operacija (n/2 zbrajanja i n/2 oduzimanja) uz samo slanje 2 poziva rekurzije

Svaki od 2 iduća koraka mora napraviti n/2 operacija (n/4 zbrajanja i n/4 oduzimanja)

...

Dolazimo do zapisa jednadžbe, jedini neočiti dio je $+n$, jer se za dobivne vrijednosti iz poziva rek. ,na trenutnoj razini mora obaviti jos par zbrajanja i oduzimanja kojih po dubini uvijek ima n.

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

```python
def pomnozi(H,v):
    if len(v)==1:
        return [v,0]

    p = len(H)//2              #nova dimenzija podmatrice
    G = H[0:p,0:p]             #nova podmatrica
    c = np.array_split(v, 2)   #podijeli vektor na dvije polovice

    y0,cost0= pomnozi(G,c[0])  #posalji podmatricu s prvom  polovicom vekt.
    y1,cost1= pomnozi(G,c[1])  #posalji podmatricu s drugom polovicom vekt.

    additionCost = len(y0)                       #cijena operacija u trenutnom koraku
    ukupanCost = cost0+cost1+2*additionCost +0.1    #ukupna cijena operacija koju zelim vratiti

    rez=np.concatenate((                          #rezultat je vektor oblika (y0+y1,y0-y1)
        np.add(y0 , y1),
        np.subtract(y0, y1)
    ))

    return rez,ukupanCost
```

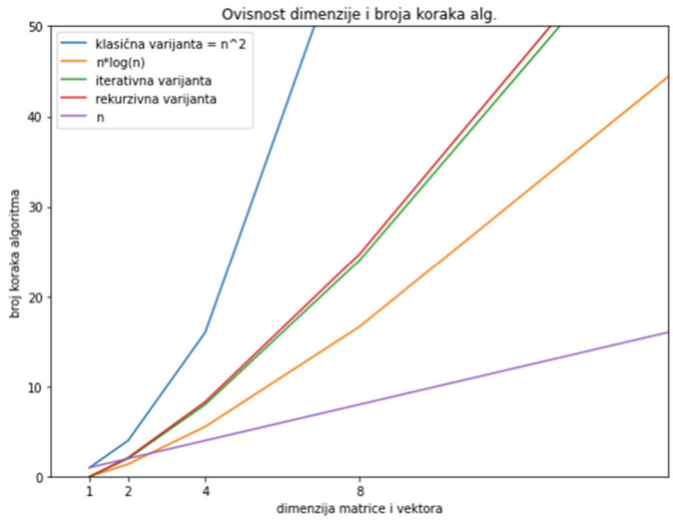
(additionCost je broj operacija po koraku,ukupanCost je cost od cijelog algoritma)
(dodan „+0.1“ kod costa zbog prikaza na grafu)

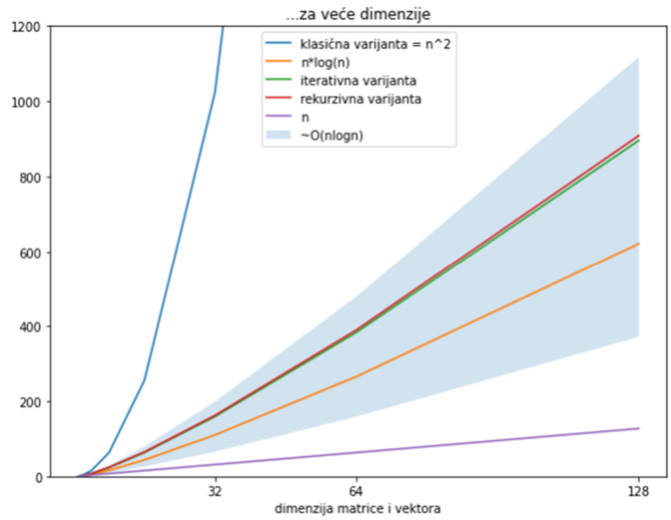
3.Klasična varijanta (dana samo za usporedbu) , očita složenost $O(n^2)$

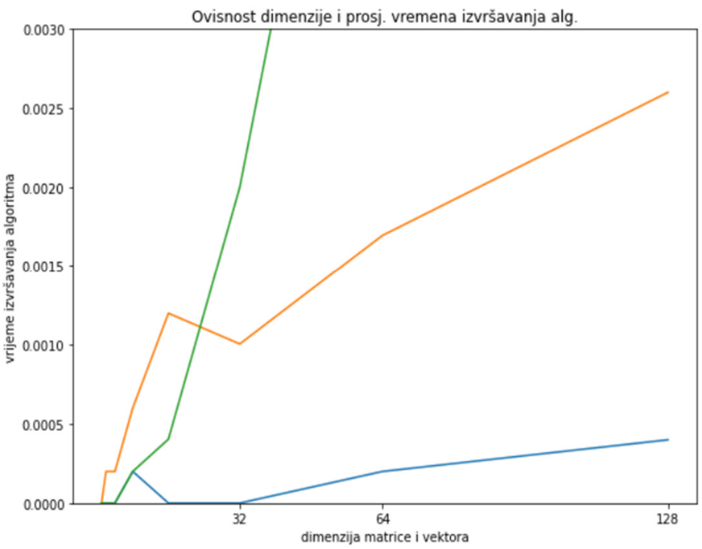
```python
def pomnoziKlasicno(H,v):
    N=len(H)
    res=np.zeros(N)

    for j in range(N):
        for i in range(N):
            res[j]= res[j] + H[j,i]*v[i]

    return res
```

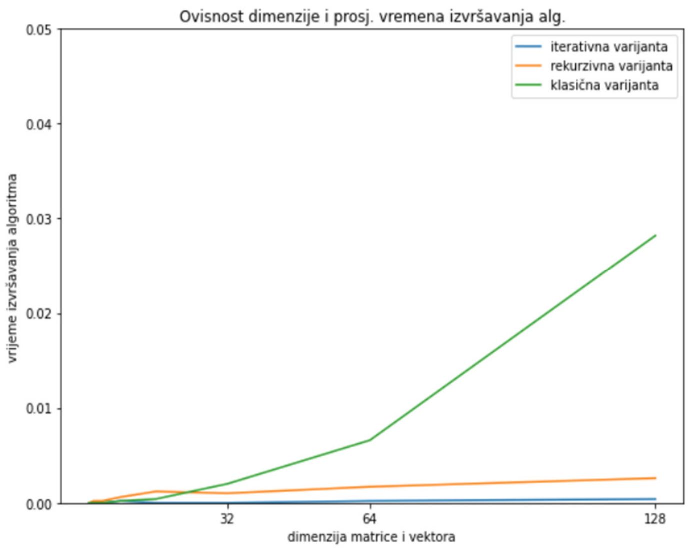
4.Testovi

NAP: broj koraka identičan za obje varijante, na grafu razlika namjerno napravljena









NAP: 5 iteracija je uzeto kod testiranja vremena,pa nije predvidljivo za male dimenzije,za veće je očito već da postoji drastična razlika u vremenu

NAP2: nisam u mogućnosti bio testirati veće dimenzije (od 128?) zbog velične registra za integere u pytonu, tada bi broj koraka bio još više očit, da teži prema nlogn

5.Dokazi složenosti

Dokaz složenosti iterativne varijante

$M := len(a) = 2^k$ za neki k

$N := M/2 = 2^{k-1}$

$H := \{1,2,4,8,\dots,2^{k-1} = N\}$

$I := \{0,2h,4h,8h,\dots\}$

$$S = \sum_{h\in H}\sum_{i\in I}\sum_{j=i}^{i+h} 1 =$$

$$= \sum_{h\in H}\sum_{i\in I} h =$$

Skup I smo preuredili, jer i ide po (+2h)

$$= \sum_{h\in H} h \sum_{i=0}^{\left\lfloor\frac{N}{2h}\right\rfloor} 1 =$$

Skup H smo preuredili, jer h ide po (*2)

$$= \sum_{h=0}^{log(N)} h\left\lfloor\frac{N}{2h}\right\rfloor = \sum_{h=0}^{log(N)} h\frac{N}{2h} = log(N)\frac{N}{2}$$

Najvece cijelo nam netreba zbog izbora N

$$\frac{1}{8}log(M)\mathrm{M} \le \frac{1}{4}log\left(\frac{M}{2}\right)M \le \frac{1}{4}log(M)\mathrm{M}$$

$(za\ x \ge 4)$

$$\rightarrow S \in O(n\log(\mathrm{n}))$$

Dokaz složenosti rekurzivne varijante

$T(n) = 2T(\frac{n}{2}) + n$

Rekurzija se grana na 2 dijela ,te izvodi n operacija po dubini

$T(2^k) = 2T(2^k) + 2^k, n = 2^k$

$$\begin{cases} -2t_k = -4t_{k-1} - 2\cdot 2^k \\ t_{k+1} = 2t_k + 2\cdot 2^k \end{cases}$$

$t_{k+1} - 4t_k + 4t_{k-1} = 0$

$x^2 - 4x + 4 = (x-2)^2 = 0$

$t_k = C_1 2^k + C_2 k 2^k$

$T(n) = C_1 n + C_2\log_2(n)\cdot n \in O(n\log(\mathrm{n}))$

Kraći dokaz:

Theorem 1 *The recurrence*

$$\begin{aligned} T(n) &= aT(n/b) + cn^k \\ T(1) &= c, \end{aligned}$$

where a, b, c, and k are all constants, solves to:

$$\begin{aligned} T(n) &\in \Theta(n^k) \text{ if } a < b^k \\ T(n) &\in \Theta(n^k \log n) \text{ if } a = b^k \\ T(n) &\in \Theta(n^{\log_b a}) \text{ if } a > b^k \end{aligned}$$

$$\begin{pmatrix} k = 1 \\ a = 2 \\ b = 2 \\ c = 1 \end{pmatrix} \rightarrow \text{MASTER TM.}$$

$T(n) \in n^k log(n) = \mathrm{n}\, log(n)$

Mateo Martinjak

Dodatni materijali:
https://en.wikipedia.org/wiki/Fast_Walsh%E2%80%93Hadamard_transform

U prilogu se još nalazi:

- code.py - svi korišteni algoritmi,pomoćne funkcije i biblioteke
- OAA_DZ.ipynb - Python Jupyter bilježnica u kojoj je pisan sav kod