UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Finančna matematika – 1. stopnja

Anamari Oštarijaš, Tina Ražić

KATZOVA SREDIŠČNOST IN GOOGLOV PAGERANK

Poročilo projekta pri predmetu Finančni praktikum

Ljubljana, 2019

Kazalo

Slike

1. Opis projekta

Kompleksna omrežja lahko analiziramo z uporabo različnih kvantitetnih merjenj, imenujemo jih tudi mere središčnosti, ki intuitivno zajamejo pomembnost določenih vozlišč.

V projektu sva implementirali Googlov PageRank in Katzovo središčnost z uporabo potenčne metode. Na različnih grafih (tudi socialnih omrežjih) sva analizirali in primerjali, kako merjenji razvrstita vozlišča po pomembnosti.

2. Katzova središčnost

2.1. **Matematično ozadje.** Katzova središčnost izmeri vpliv igralca v omrežju tako, da upošteva direktne sosede igralca in vse druge igralce, ki so posredno povezani s tem igralcem preko njegovih direktnih sosedov.

Naj bo naše omrežje graf z n vozlišči oziroma spletnimi stranmi. Naš graf predstavimo z matriko sosednosti A, torej element matrike a_{ij} ima vrednost 1, če je vozlišče i povezano z vozliščem j, in 0, če nista povezana. Katzovo središčnost x_i vozlišča i lahko zapišemo kot

$$x_i = \alpha \sum_k a_{i,k} x_k + \beta,$$

kjer sta α in β konstanti. Konstanta β je dana začetna središčnost vsakega vozlišča. Zapišimo zvezo v matrični formi:

$$x = \alpha A x + \beta,$$

kjer je β sedaj vektor, katerega komponente so enake dani konstanti. Sledi, da lahko vektor x izračunamo kot

$$x = (I - \alpha A)^{-1}\beta = \sum_{t=0}^{\infty} (\alpha A)^t \beta.$$

Vidimo, da je Katzova središčnost vozlišča določena z uteženimi potmi do ostalih vozlišč. Vsaka povezava v grafu dobi utež α in z α^t izračunamo težo povezave vozlišča z drugim vozliščem, pri čemer je t število povezav med njima. Potence matrike A nam povejo, če je vozlišče povezano s drugimi indirektnimi vozlišči preko sosedov. Na primer, če je v matriki A^3 element $a_{2,5} = 1$, pomeni, da sta vozlišče 2 in vozlišče 5 povezana s tremi povezavami preko sosedov prve stopnje in sosedov druge stopnje.

Pri izbiri α moramo upoštevati omejitev

$$0 < \alpha < \frac{1}{|\lambda_{max}|},$$

saj pri $\alpha = \frac{1}{|\lambda_{max}|}$ algoritem divergira. Za majhen α poti, ki imajo več kot eno povezavo do izbranega vozlišča, prispevajo zelo malo k središčnosti vozlišča. Na primer pri $\alpha = 0$ izračun mere središčnosti sploh ne pride do izraza in imajo vsa vozlišča mero središčnosti β. Pri večjem α pa se vpliv β koeficienta manjša.

2.2. **Algoritem za Katzovo središčnost.** Algoritem sva zapisali v programu Python.

```
1 def katz(G, max_num_of_steps, tolerance, alpha, beta, vector=None):
2     '''Za dan graf vrne Katzovo srediscnost vozlisc'''
3     start = timer()
4     A = nx.adjacency_matrix(G)
5     diff = 1000
6     k = 0
7     ones = np.ones((A.shape[1], 1))
```

```
    if vector is None:
        vector = ones
    r = beta * vector
    while diff > tolerance and k < max_num_of_steps:
        #potencna metoda
        r, q = alpha * A.dot(r) + beta * ones, r
        diff = np.linalg.norm(q-r, ord=1)
        k += 1
    r = r / np.linalg.norm(r, ord = 2)
    print('proces koncan po {} iteracijah'.format(k))
    end = timer()
    # timing of the process
    print('trajanje procesa: {} sekund'.format(end-start))
    return r
```

3. Googlov PageRank

Nemogoče je definirati splošno mero pomembnosti, ki bi bila sprejemljiva za vse uporabnike iskalnika. Google uporablja pagerank za mero kakovosti spletnih strani. Temelji na predpostavki, da število linkov (povezav) do in iz strani daje informacijo o pomembnosti strani. Ta algoritem popravi pomanjkljivost Katzove središčnosti, ki pravi, da je spletna stran pomembna, če je povezana s pomembno spletno stranjo. V resnici to ne drži, zato Googlov PageRank enakomerno porazdeli pomembnost spletne strani med vse povezave.

3.1. **Matematično ozadje.** Naj bodo vse spletne strani urejene s števili od 1 do n in naj bo i neka spletna stran. Potem O_i določa množico strani, s katerimi je i povezana, tako da i vsebuje link do strani v množici O_i (*outlink*). Število outlinkov označimo z $N_i = \|O_i\|$. Množica *inlinkov*, označena z I_i, je množica strani, ki imajo outlink do i (strani v I_i vsebujejo linke do i). Splošno, več ko ima stran i inlinkov, pomembnejša je. S takim sistemom bi bilo preprosto manipulirati (če bi nekdo želel, da njegovo spletno stran vidi čim več ljudi, bi ustvaril veliko število nepomembnih spletnih strani, ki bi vsebovale linke do njegove spletne strani). Da bi tako manipulacijo preprečili, definiramo rang vozlišča i tako, da če ima visoko rangirana stran j outlink do i, to doda pomembnosti i na sledeč način: rang strani i je utežena vsota rangov strani, ki imajo outlink do i. Obteženost je taka, da je rang strani j razdeljen enakomerno med njenimi outlinki. Z enačbo:

$$r_i = \sum_{j \in I_1} \frac{r_j}{N_j}.$$

Ta definicija je rekurzivna, zato pageranki ne morejo biti izračunani direktno. Uporabimo iteracijo. Najprej ugibamo začetni rangni vektor r^0. Potem iteriramo:

$$r_i^{(k+1)} = \sum_{j \in I_1} \frac{r_j^{(k)}}{N_j}.$$

Raje zapišimo problem z matrikami. Naj bo Q kvadratna matrika dimenzije n. Definiramo:

$$Q_{ij} = \begin{cases} 1/N_j & \text{, če obstaja link od j do i} \\ 0 & \text{, sicer} \end{cases}$$

Torej ima vrstica i neničelne elemente na mestih inlinkov i. Podobno ima stolpec j neničelne elemente enake N_j na mestih outlinkov j, vsota vseh elementov v stolpcu je enaka 1.

Prejšnjo enačbo sedaj lahko zapišemo v matrični obliki:

$$\lambda r = Qr, \qquad \lambda = 1.$$

Tako dobimo, da je r lastni vektor matrike Q z lastno vrednostjo $\lambda = 1$. Sedaj preprosto vidimo, da je iteracija ekvivalentna

$$r^{(k+1)} = Qr^{(k)}, \qquad k = 0, 1, \ldots,$$

kar je potenčna metoda za izračun lastnega vektorja, ki pripada lastni vrednosti 1.

Trenutno še ni jasno, da je pagerank dobro definiran, saj ne vemo, če obstaja lastna vrednost enaka 1. Pomagamo si s teorijo Markovske verige.

Obstaja interpretacija pageranka z naključnimi sprehodi. Predpostavimo, da slučajni uporabnik na spletni strani izbere naslednjo stran med outlinki z enako verjetnostjo. Markovska veriga je slučajni proces, v katerem je naslednje stanje določeno le s trenutnim stanjem, proces nima spomina. Prehodna matrika Markovske verige je Q^T.

Slučajni uporabnik nikoli ne sme obtičati. Z drugimi besedami, naš model slučajnega sprehoda ne sme imeti strani brez outlinkov (taka stran bi imela ničelni stolpec v Q). Torej je model prilagojen tako, da so ničelni stolpci nadomeščeni s konstantnimi vrednostmi na vseh mestih. To pomeni, da je enaka verjetnost, da pridemo na katero koli drugo spletno stran.

S tem namenom definiramo vektorje:

$$d_j = \begin{cases} 1 & , \text{če } N_j = 0 \\ 0 & , \text{sicer} \end{cases}$$

za $j = 1, .., n$ in

$$e = [1 \ldots 1]^T \in \mathbb{R}^n.$$

Prilagojena matrika je definirana s

$$P = Q + \frac{1}{n} e d^T.$$

S to prilagoditvijo je matrika P desna stohastična matrika; vsi elementi so nenegativni in elementi vsakega stolpca se seštejejo v 1. Desna stohastična matrika P zadošča $e^T P = e^T$.

Želimo definirati pagerank vektor kot enoličen lastni vektor matrike P z lastno vrednostjo 1:

$$Pr = r.$$

Lastni vektor prehodne matrike je stacionarna verjetnostna porazdelitev Markovske verige. Element na mestu i (r_i) je verjetnost, da po velikem številu korakov slučajni uporabnik pristane na strani i. Za zagotovitev enoličnosti mora biti matrika ireducibilna (slučajni uporabnik se lahko v nekem delu grafa zagozdi, v tem primeru ima matrika več lastnih vrednosti enakih 1).

Enoličnost največje lastne vrednosti ireducibilne, pozitivne, desne stohastične matrike je zagotovljena s *Perron-Frobeniusovim izrekom*, največja singularna vrednost bo enaka 1, pripadajoč lastni vektor je pozitiven in je edini lastni vektor, ki je nenegativen.

Zaradi velikosti spleta je matrika linkov P reducibilna, torej pagerank lastni vektor ni dobro definiran. Da si zagotovimo ireducibilnost, umetno dodamo linke iz vsake spletne strani do vseh drugih. To lahko storimo, če vzamemo konveksno kombinacijo P in matrike ranga 1:

$$A = \alpha P + (1 - \alpha)\frac{1}{n}ee^T,$$

za nek α, ki zadošča $0 \leq \alpha \leq 1$. Matrika A je desna stohastična. Razlaga naključnega sprehoda dodatnega rang-1 izraza je, da bo uporabnik na vsakem časovnem koraku skočil na naključno stran z verjetnostjo $1 - \alpha$. Za konvergenco numeričnega algoritma za lastno vrednost je pomembno, da vemo, kako so s prilagoditvijo ranga-1 spremenjene lastne vrednosti.

Izrek: Naj bodo lastne vrednosti desne stohastične matrike P enake $1, \lambda_2, \lambda_3, \ldots, \lambda_n$. Potem so lastne vrednosti $A = \alpha P + (1 - \alpha)\frac{1}{n}ee^T$ enake $1, \alpha\lambda_2, \alpha\lambda_3, \ldots, \alpha\lambda_n$.

Ta izrek nam pove, da tudi če ima P več lastnih vrednosti enakih 1, kar je po navadi res, bo druga največja lastna vrednost matrike A enaka α.

3.2. **Potenčna metoda.** Želimo rešiti problem lastne vrednosti:

$$Ar = r,$$

kjer je r nomaliziran $\|r\|_1 = 1$. Iskani lastni vektor označimo s t_1. Prepostavimo, da imamo dan začetni približek $r(0)$.

Algoritem:

$$\text{za k} = 1, 2, \ldots \text{ do konvergence}$$
$$q^{(k)} = Ar^{(k-1)}$$
$$r^{(k)} = q^{(k)}/\|q^{(k)}\|_1$$

Konvergenca je odvisna od porazdelitve lastnih vrednosti. Če je druga največja lastna vrednost blizu 1, bo iteracija zelo počasna. To na srečo ne velja za Google matriko. Vektor normaliziramo, da ne bi z iteracijami postali preveliki ali premajhni, posledično nereprezentativni s števili s plavajočo vejico. To v resnici ni potrebno, saj se v primeru desnih stohastičnih matrik temu izognemo.

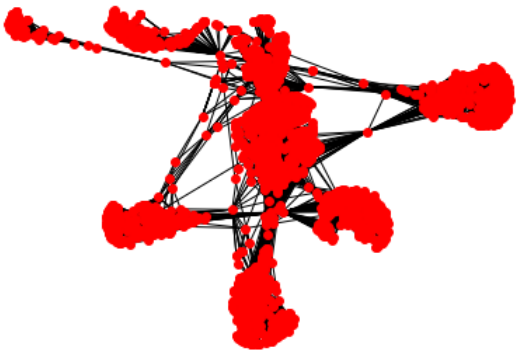
3.3. **Algoritem za Googlov PageRank.** Algoritem sva napisali v programu Python.

```
1 def Qmatrix(matrika):
2     '''za dano matriko sosednosti grafa vrne desno stohasticno matriko
     Q, kjer so nekateri stolpci se vedno nicelni'''
3     #tabela vsot stolpcev matrike
4     sums = np.sum(matrix, axis=0)
5     #vsak stolpec matrike delimo z njegovo vsoto, ce je vsota 0, vrne 0
      namesto nan
6     return np.nan_to_num(matrix/sums)
```

```
7
8  def dvector(matrika):
9      '''za dano matriko sosednosti vrne transponiran vektor d, kjer d[j]
       = 1 ; ce #(outlinkov od j) = 0 in d[j] = 0 ; sicer'''
10     Q = Qmatrix(matrix)              #izracunamo matriko Q
11     sums = np.sum(matrix, axis=0)    #tabela vsot stolpcev matrike
12     d_bool = (sums == 0)             #bool tabela, True ce je vsota
       stolpca enaka 0 in False sicer
13     d = d_bool*1                     #False v 0 in True v 1
14     return d
15
16 def evector(n):
17     '''vrne vektor v iz samih enk velikosti n, kjer n = st. stolpcev v
       matriki'''
18     return np.ones((n,1))
19
20 def Pmatrix(matrix):
21     '''vrne desno stohasticno matriko P'''
22     n = matrix.shape[1]
23     Q = Qmatrix(matrix)
24     e = evector(n)
25     d = dvector(matrix)
26     return Q + 1/n*e.dot(d)
27
28 def Amatrix(matrix, alpha):
29     '''vrne ireducibilno desno stohasticno matriko A, kjer je 0 <=
       alpha <= 1'''
30     n = matrix.shape[1]
31     e = evector(n)
32     P = Pmatrix(matrix)
33     return alpha*P + (1-alpha)*1/n*e.dot(e.transpose())
34
35 def pagerank2(matrix, max_num_of_steps, tolerance, alpha):
36     '''vrne pagerank vektor matrike sosednosti grafa, 0 <= alpha <= 1
       '''
37     start = timer()
38     #desno stohasticna matrika dane matrike
39     A = Amatrix(matrix, alpha)
40     #ugibanje pagerank vektorja s prvim stolpcem matrike Q
41     r = A[:,0]
42     diff = 1000
43     k = 0
44     while diff > tolerance and k < max_num_of_steps:
45         r, q = A.dot(r), r           #potencna metoda
46         #razlika med prejsnjim in novim vekorjem r
47         diff = np.linalg.norm(q-r, ord=1)
48         k+=1
49     print('proces koncan po {} iteracijah'.format(k))
50     end = timer()
51     print('trajanje procesa: {} sekund'.format(end-start))
52     return r
```

4. Analiza algoritmov

Analizirali sva učinkovitost algoritma na podlagi večih parametrov. Najbolj naju je zanimalo, kako izbira parametera α, toleranca in število korakov vplivajo na hitrost algoritma in natančnost rezultata.
Uporabljali sva *Facebook graf*, ki vsebuje 4039 vozlišč in 88234 povezav.



Slika 1. Facebook graf

4.1. **Katzova središčnost.**

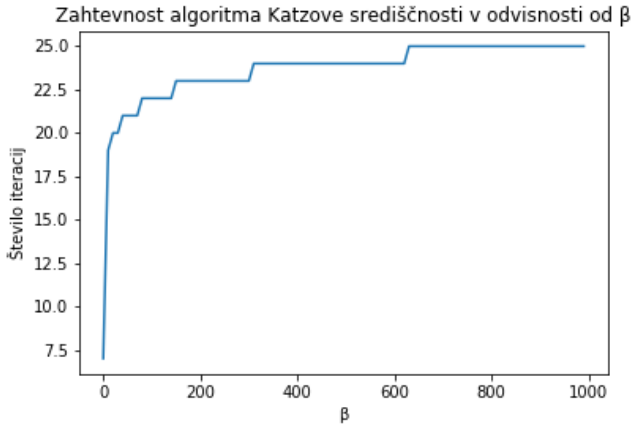
4.1.1. *Izbira parametra β.* Pri izračunu Katzove središčnosti uporabimo dve dani konstanti α in β. Dana konstanta β je začetna središčnost za vsa vozlišča, zato je vektor s konstantami β naš začetni vektor pri potenčni metodi. Pogledali sva si primera, ko je $\beta = 1$ in $\beta = 50$, pri fiksnem $\alpha = 0.003$. Spodaj bomo videli, da je to dobra izbira za α.

```
b1 = katz(G_fb, 10000000, 0.01, 0.003, 1)
b1
-----------------------------------------------------------------------
proces koncan po 15 iteracijah
trajanje procesa: 0.4371460449999631 sekund
array([[0.02708885],
       [0.01358665],
       [0.01327441],
       ...,
       [0.01291167],
       [0.01299226],
       [0.01318997]])
```

```
b2 = katz(G_fb, 10000000, 0.01, 0.003, 1000)
b2
-----------------------------------------------------------------------
proces koncan po 24 iteracijah
trajanje procesa: 0.4746733580000182 sekund
array([[0.0270888 ],
       [0.01358663],
       [0.01327439],
```

```
        ...,
        [0.01291165],
        [0.01299224],
        [0.01318995]])
```

Opazimo, da parameter β ne vpliva na rezultat algoritma, samo število iteracij se poveča. Poglejmo si graf zahtevnosti algoritma Katzove središčnosti v odvisnosti od β.



Slika 2. Graf števila iteracij pri danem β izračunan na Facebook grafu.

Vidimo, da število iteracij naraste iz 15 na 25, medtem ko beta povečamo za 1000. Tudi trajanje algoritma se vseskozi giblje okoli 0.5 sekunde, torej parameter β res nima znatnega vpliva na algoritem.

4.1.2. *Izbira parametra α.* Pri izbiri prevelikega α lahko algoritem divergira. Vemo, da mora biti α manjša od inverzne vrednosti največje lastne vrednosti. V našem primeru je največja lastna vrednost 162.37394233563828, torej za α velja omejitev $\alpha < 1/162.37 \doteq 0.00616$. Poglejmo kaj se zgodi, če je $\alpha = 0.0062$.

```
r1 = katz(G_fb, 10000000, 0.01, 0.0062, 1)
rank(r1)
```

```
proces koncan po 104436 iteracijah
trajanje procesa: 213.18206373500004 sekund
RuntimeWarning: invalid value encountered in subtract
array([[    0],
       [2683],
       [2684],
       ...,
       [1352],
       [2018],
       [4038]], dtype=int64)
```

Naš proces se ustavi po dobrih 3 minutah in pojavi se napaka. Naj bo sedaj $\alpha = 0.0061$.

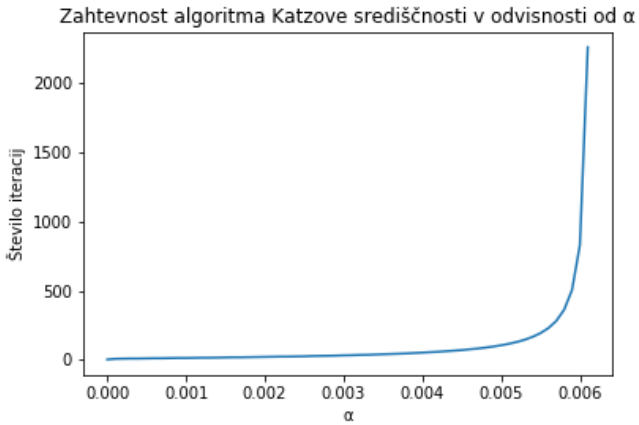
```
r2 = katz(G_fb, 100000, 0.01, 0.0061, 1)
rank(r2)
-------------------------------------------------------------------------------
proces koncan po 1056 iteracijah
trajanje procesa: 1.1106130180000946 sekund
array([[3404],
       [1125],
       [ 577],
       ...,
       [  17],
       [  91],
       [ 361]], dtype=int64)
```

Proces se konča po 1056 iteracijah in v dobri 1 sekundi, kar je skoraj stokrat manj kot pri $\alpha = 0.0062$. Poglejmo, če se z manjšim α, tudi število iteracij zmanjša. Najprej poskusimo z $\alpha = 0.003$.

```
r3 = katz(G_fb, 10000000, 0.01, 0.003, 1)
rank(r3)
-------------------------------------------------------------------------------
proces koncan po 15 iteracijah
trajanje procesa: 0.2149773610003649 sekund
array([[3977],
       [1403],
       [ 807],
       ...,
       [  43],
       [ 175],
       [ 598]], dtype=int64)
```

Opazimo ogromen napredek. Sedaj algoritem potrebuje samo 15 iteracij in približno 0.2 sekundi.

Poglejmo si graf zahtevnosti algoritma Katzove središčnosti v odvisnosti od α.



Slika 3. Graf števila iteracij pri danem α izračunan na Facebook grafu.

Vidimo, da število iteracij z večanjem konstante α zelo počasi narašča, ko pa se α približuje zgornji omejitvi, pa število iteracij eksponentno zraste. Za primerjavo

algoritma Katzove središčnosti z algoritmom PageRank bomo vzeli $\alpha = 0.003$, pri katerem algoritem potrebuje 15 iteracij in 0.4 sekunde ter $\beta = 1$.

4.2. **Googlov PageRank.**

4.2.1. *Izbira parametra α.* Vemo, da manjši ko je α, več podatkov o omrežju izgubimo. Torej želimo, da je ta parameter čim večji.

Poglejmo najprej kako velikost vpliva na čas izračuna pagerank vektorja.

```
M_fb = nx.adjacency_matrix(G_fb)
r_1 = pagerank2(M_fb,10000000,0.0000001, 0.98)
print(r_1)
```

```
proces koncan po 575 iteracijah
trajanje procesa: 11.7118154744594 sekund
[[ 4.79508113e-03]
 [ 2.17910466e-04]
 [ 1.52316417e-04]
 ...,
 [ 6.77736016e-05]
 [ 1.25649865e-04]
 [ 2.72346363e-04]]
```

Izračun je bil končan po 575 iteracijah in 11.71 sekundah.

```
r_2 = pagerank2(M_fb,10000000,0.0000001, 0.999)
print(r_2)
```

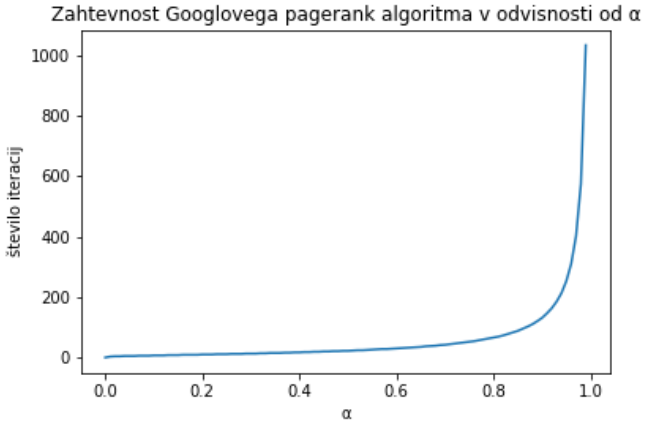
```
proces koncan po 4719 iteracijah
trajanje procesa: 61.73710875091092 sekund
[[ 2.37207123e-03]
 [ 1.15387662e-04]
 [ 6.99886732e-05]
 ...,
 [ 2.37766591e-05]
 [ 4.70770334e-05]
 [ 1.04002744e-04]]
```

S povečanjem α smo povročili, da se je čas izračuna povečal na 61.74 sekund, torej za več kot 5 kratnik prejšnjega.

```
r_4 = pagerank2(M_fb,10000000,0.0000001, 0.9999)
print(r_4)
```

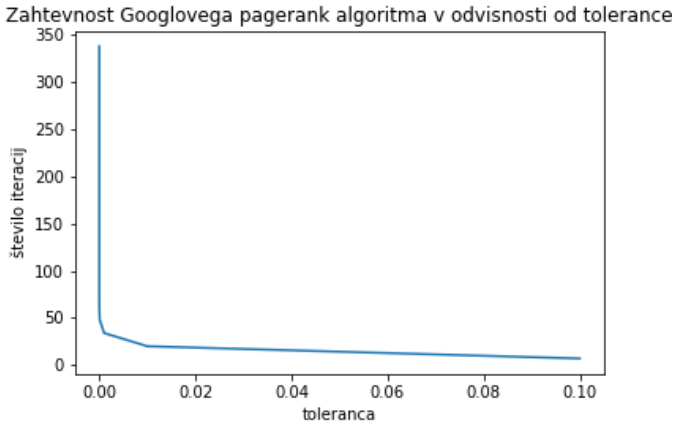
```
proces koncan po 8104 iteracijah
trajanje procesa: 105.30294071864228 sekund
[[ 1.99799361e-03]
 [ 9.77897213e-05]
 [ 5.77554403e-05]
 ...,
 [ 1.28541957e-05]
 [ 2.56615989e-05]
 [ 5.75079410e-05]]
```

Večji ko je α, več iteracij potrebujemo, da pridemo do željene tolerance. Razlika med r_1 in r_2 v drugi normi je 0.00718, med r_2 in r_4 pa 0.00372, torej lahko z višjim α pričakujemo bolj natančne rezultate.



Slika 4. Graf števila iteracij pri danem α izračunan na Facebook grafu.

4.2.2. *Izbira tolerance.* Očitno je, da višjo ko zastavimo toleranco, daljši bo postopek izračuna. Toleranco si izberemo sami in je odvisna od tega, kako natančen rezultat želimo. Tu sva za izračun sprotne tolerance v algoritmu izbrali prvo normo vektorja razlike med prejšnjim in trenutnim pagerank vektorjem ob določeni iteraciji, preprosto zaradi razloga, da je ta norma vedno večja ali enaka drugi normi vektorja.



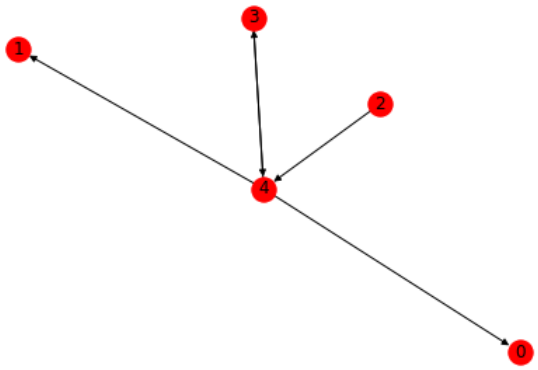
Slika 5. Graf števila iteracij pri $\alpha = 0.85$ glede na dano toleranco izračunan na Facebook grafu.

5. Primerjava algoritmov

Pri primerjavi algoritmov sva si pomagali z napisano funkcijo *rang*, ki nama vrne vektor rangiranih vrednosti, pri čemer ima vozlišče z najmanjšo središčnostjo najmanjši rang. Najprej sva najina algoritma preverili na majhnem grafu, potem pa še na večjih.

Za toleranco sva pri obeh algoritmih vzeli 0.000000001.

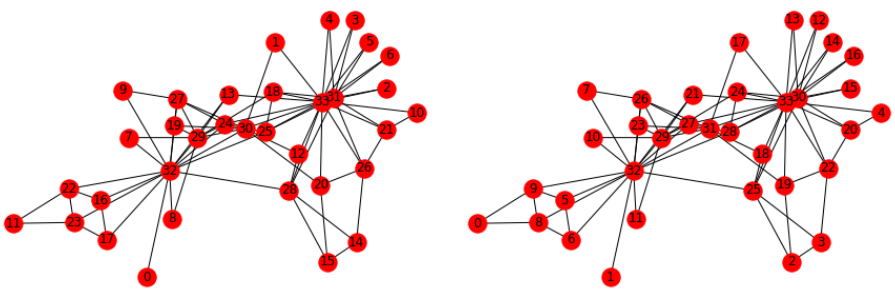
5.1. **Majhen graf.** Majhen graf T ima 5 vozlišč in 5 usmerjenih povezav. Najina algoritma na grafu T vrneta isto razvrstitev pomembnosti vozlišč, pri čemer Katz opravi 101 iteracij ($\alpha = 0.8$), pagerank pa 27 ($\alpha = 0.99$). α sva si izbrali tako, da je čim bolj ustrezala algoritmu na danem grafu, torej pri pageranku čim večja, pri Katzu pa ustrezno majhna.



Slika 6. Razvrstitev pomembnosti vozlišč s funkcijo rang v majhnem grafu.

5.2. **Malo večji graf.** Za malo večji graf sva vzeli vgrajen graf *Zachary's Karate Club* v knjižnici *NetworkX*, ki ima 34 vozlišč in 78 povezav.
Pagerank je za doseg tolerance porabil 122 iteracij, 0.22 sekunde, Katz pa 398 iteracij in 0.051 sekunde. Razlog za hitrejši izračun vektorja s Katzom kljub večjemu številu iteracij je proces priprave vhodne matrike do Google matrike, na kateri potem izvajamo potenčno metodo.

Najina algoritma samo 7 vozlišč enako rangirata. Do glavne razlike pride, ko pri Googlovem PageRank algoritmu pomembnost vsakega vozlišča razdelimo enakomerno med izhodne povezave, pri Katzovem algoritmu pa celotno pomembnost prenesemo na povezana vozlišča. To je vidno s primerjavo ranga vozlišča, ki je na levem grafu označen z rangom 23 na desnem pa z 8. Googlov pagerank da temu vozlišču večjo rangiranost zaradi povezanosti z drugim najpomembnejšim vozliščem, t.j. 32, Katzova središčnost pa gleda splošno povezanost, zato je rangiranost med manjšimi.



Slika 7. Razvrstitev pomembnosti vozlišč v malo večjem grafu z Googlovim PageRank algoritmom (levo) in Katzovim algoritmom (desno).

Opazimo, da z obema algoritmoma na zunanjih vozliščih dobimo nižji rang, torej manj pomembna vozlišča, najpomembnejši 2 vozlišči sta pa enaki.

5.3. **Facebook graf.** Algoritma sva preizkusili tudi na Facebook grafu, na katerem sva oba že prej testirali. Pri enaki toleranci kot pri manjših grafih je Katz zaključil po 741 iteracijah ($\alpha = 0.006$), pagerank pa po 1033 ($\alpha = 0.99$) z ujemanjem ranga le na dveh vozliščih. Do malo večjega ujemanja lahko pridemo s popravljanjem parametra α, ampak so ta ujemanja še vedno na manj kot 10 vozliščih.

5.4. **Sklep.** Katzovo središčnost in Googlov pagerank je na primerih težko primerjati, saj delujeta na drugačen način. Katzova središčnost meri povezanost izbranega vozlišča z ostalimi, medtem ko Googlov pagerank meri pomembnost z obteženostjo sosednjih povezav.

Algorima se na manjših, manj povezanih grafih lahko ujemata na večih vozliščih, pri večjih grafih pa bo zaradi različnega pristopa prišlo do vse večjih razlik. Katz je bil na izbranih grafih večinsko hitrejši od pageranka, razlog je poleg dodatnega računanja Google matrike tudi v izbiri $\alpha = 0.99$, ki precej oteži čas računanja.

Obe metodi sta časovno in vsebinsko še dodatno prilagodljivi z določitvijo parametrov, zato bi splošno bi za neko iskano mero središčnosti izbrali algoritem, ki bolj ustreza našemu namenu.

Literatura

[1] L. Elden, *Matrix Methods in Data Mining and Pattern Recognition*, Linkoping, Sweden: Linkoping University, 2007.

[2] J. Leskovec in A. Krevlj, *SNAP Datasets: Large network*, 2014, [ogled 30.12.2018], dostopno na: `http://snap.stanford.edu/data`.

[3] A. Disney, *EigenCentrality and PageRank*, 2015, [ogled 2.1.2019], dostopno na: `https://cambridge-intelligence.com/eigencentrality-pagerank`