SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

BIOINFORMATIKA

Računanje optimalnog praga za q-gram filtre

Antun Flaš, Manuela Kajkara, Marija Kaselj

Zagreb, siječanj, 2016.

Sadržaj

1. Uvod

Pretraživanje teksta bitan je dio informacijsko komunikacijske infrastrukture. Do danas je najpopularnije pretraživanje teksta uspoređivanjem kontinuiranog niza znakova, dok je pretraživanje nepotpunog niza znakova zanemareno. Prve rezultate iz tog području dali su Stefan Burkhardt i Juha Kärkäinen na čijem radu se i temelji ovaj seminar.

U nastavku je dan kratak pregled teorije vezan uz q-grame i algoritam izračuna optimalnog praga za q-gram filtre uz jednostavan primjer, te opis vlastite implementacije navedenog.

2. Osnovni pojmovi

Za dani uzorak P, niz znakova T i udaljenost između dva niza k potrebno je pronaći sve podnizove znakova S koji zadovoljavaju navedene uvjete. Kako bi se taj proces ubrzao radi se filtriranje. Filtriranje je algoritam koji provjerava tekst prema uvjetima filtriranja te potom odbacuje višak teksta ostavljajući samo moguća slaganja kako bi se provjerili algoritmom uspoređivanja teksta. Mnogi filtri koriste q-grame, podnizove duljine q.

Sličnost između niza znakova definiramo pragom t koji predstavlja minimalni broj q-grama koje mogući podniz S niza T treba imati s uzorkom P.

Kako bi se filtriranje ubrzalo i kako bi se moglo koristiti mnogo različitih oblika uzoraka (eng. *shapes*) koriste se nepotpuni q-grami.

2.1. Nepotpuni q-gram

Skup Q definira uzorak filtriranja. Sastoji se od brojeva koji predstavljaju indekse na kojima provjeravamo postojanje uzorka u podnizu.

Uz definirani skup Q i poziciju i:

$$Q = \{i + j \mid j \in Q\}$$

$$Q_i = \{i_1, i_2, \dots, i_q\}, \qquad i = i_1 < i_2 < \cdots < i_q$$

te niz znakova:

$$S = s_1 s_2 \dots s_m$$

Za $1 \leq i \leq m - span(Q) + 1$, Q-gram na poziciji *i* u nizu S je $S[Q_i] = s_{i1} s_{i2} \dots s_{iq}$ gdje je $q = |q|$, $s = span(Q)$.

Q-gram definiramo i kao (q, s)-gram.

Primjer:

Za $Q = \{0, 1, 4, 6\}$ uzorak je # # _ _ #_#, q=4, s=7 te se radi o (4,7)-gramu.

2.2. Prag

Prag t predstavlja Q-gram sličnost sa uzorkom P.

$$t_Q(m,k) = \min_{M\subseteq[1,M],|M|=m-k} |\{i \in [1, m - span(Q) + 1] \mid Q_i \subseteq M\}| \qquad (*)$$

Previsoki prag može rezultirati gubitcima, no s višim pragom se povećava i učinkovitost filtriranja. Iz tih razloga veoma je bitno postići optimalan prag.

2.2.1. Optimalan prag

Optimalan prag je najveći prag kod kojeg ne dolazi do gubljenja podataka odnosno najmanja Q-gram sličnost bilo koja dva niza znakova duljine m i udaljenosti k.

Moguće ga je izračunati iscrpnom pretragom svih kombinacija k razlika koristeći (*) no to je vrlo skupa operacija za velike vrijednosti m i k. Postupak se bitno ubrza korištenjem dinamičkog programiranja.

2.2.1.1. Rekurzivan izračun optimalnog praga

Prema [1], definiramo:

Za *I* kao skup cjelobrojnih numeričkih vrijednosti vrijedi sljedeća notacija:

$I \oplus 1$ kao $\{i + j \mid i \in I\}$, isto tako $I \ominus 1$ kao $\{i - j \mid i \in I\}$,

$[cond]$ je 1 ako je *cond* istinit odnosno 0 ako je *cond* neistinit.

Vrijedi:

Za $s \leq i \leq m$, $0 \leq j \leq k$ i $M \subseteq [1, s-1]$, $|M| \geq s - 1 - j$ definiramo optimalan prag kao:

$$t_Q(s-1, j, M) = 0$$

$$t_Q(i,j,M) = min\begin{cases} t_Q(i-1, j-[s-1 \notin M], (M \cup \{0\} \setminus \{s-1\})\oplus 1) + [Q \subseteq (M\{0\})] \\ t_Q(i-1, j-[s-1 \notin M], (M \cup \{0\} \setminus \{s-1\})\oplus 1) \end{cases}$$

Skup M sadrži indekse podudaranja poslijednjih $s-1$ pozicije.

Dokaz:

Ako je $i < s, s = span(Q)$, $t_Q(i,*,*)$ je uvijek 0 jer niz znakova koji je kraći od duljine q grama ne može ga niti sadržavati.

Vrijednost $t_Q(i-1, j^*, M^*)$ se može izračunati iz $t_Q(i-1, j^*, M^*)$ uzimajući u obzir sljedeće mogućnosti:

1. $j^* = j$ ili $j^* = j - 1$ ovisno o tome da li postoji podudaranje na poziciji *i* (da li skup M sadrži $s-1$).
2. Prag $t_Q(i, j, M)$ je ili jednak $t_Q(i-1, j^*, M^*)$ ako ne postoji nova razlika, ili $t_Q(i-1, j^*, M^*)$+1 ako postoji nova razlika.
3. Skup M* se mora podudarat sa skupom M na s-2 pozicije $M^* \cap \{2, \dots, s-2\} = (M \cap \{1, \dots, s-2\}) \oplus 1$.

Uvjeti kombinirani s minimizacijom rezultiraju ispravnim izračunom optimalnog praga.

2.2.1.2. Izračun optimalnog praga dinamičkim programiranjem

Izračun započinjemo računanjem $t_Q(i, j, M)$ za sve pogreške j, $j \leq k$, za sve postojeće skupove M koji se podudaraju na zadnjih $s-1$ pozicija te za $i \leq m$. Od svih dobivenih vrijednost za $t_Q(m, k, *)$ biramo najmanji t_Q.

Kako bi se ubrzao izračun optimalnog praga vrijednosti $t_Q(i, *, *)$ su pohranjene u polju. Za svaku moguću pogrešku *j* između 0 i *k* sadrži $t_Q(i, j, M)$ za sve skupove M čiji broj pogrešaka je između 0 i *j*. Prema tome veličina tog polja je:

$$\sum_{a=0}^{k} \binom{s-1}{a} (k - a + 1)$$

Ažuriranje polja s vrijednostima $t_Q(i-1, *, *)$ na vrijednosti za $t_Q(i, *, *)$ potrebno je formulu za izračun pozivati za svaki element polja. Rezultat se dobiva u konstantnom vremenu jer su moguće samo dvije opcije čije su vrijednosti već izračunate i pohranjene u polju.

2.2.2. Skup Q

Prema [1], učinkovito generiranje novih skupova Q postže se na sljedeći način. Prvo je potrebno podijeliti (*q-1,s*) uzorke s pozitivnim pragom u skupine kojima je jedina razlika element na predzadnjoj poziciji ($q-2$).

$Q_1 = \{i_1, \dots, i_{q-1}\}$ i $Q_2 = \{j_1, \dots, j_{q-1}\}$ pripadaju istom skupu ako
$i_k = j_k, \forall\, k \in \{1, \dots, q-3, q-2\}$.

Novi skup je $Q = Q_1 \cup Q_2 = \{i_1, \dots, i_{q-2}, j_{q-2}, i_{q-1}\}$. Veličine *q* je jer sadrži sve elemente iz skupa Q_1 i jedan dodatan element iz Q_2.

3. Implementacija

Za zadane vrijednosti m, k, q i s izračun optimalnog praga temelji se na dinamičkom programiranju i strukturi za pohranu međurezultata.

Prag se izračunava za generirane (q, s)-uzorke te se od svih izračunatih pragova uzima maksimalni.

Pri samom izračunu praga za određeni uzorak predstavljen setom Q potrebno je ažurirati strukturu $m - s$ puta za svaku duljinu od s do m. Tako se prag za duljinu i računa pomoću vrijednosti pragova dobivenih za duljinu $i - 1$, potom se vrijednosti za duljinu $i + 1$ računaju pomoću vrijednosti za i, povećavajući tako i sve do m.

Izračun se vrši prema formuli za izračun praga uz čitanje svih vrijednosti iz pomoćne strukture čime nema potrebe za rekurzijom.

U nastavku je prikazan pseudokod opisanog postupka.

```
result = 0
shapes = generate(q,s)
for shape in shapes
      shapeThreshold = MAX
      for i in range (s, m)
            for key in thresholds
                  for position in thresholds[key]
                        M, j = convert(M, position)
                        threshold = findThreshold(i, j, M, shape)
                        thresholds[key][position] = threshold
                        shapeThreshold = min(threshold, shapeThreshold)
                  end for
            end for
      end for
      result = max(shapeThreshold, result)
end for
```

Pseudokod 1 – Izračun optimalnog praga dinamičkim programiranjem

3.1. Struktura pohrane međurezultata

Prilikom izračuna praga za mjeru udaljenosti k, potrebno je spremati vrijednosti izračuna za sve vrijednosti greške do uključivo k te odgovarajuće vrijednosti skupa M koji predstavlja podudaranja na zadnjih $s - 1$ pozicija. Za najveći mogući broj pogrešaka k, broj mogućih vrijednosti skupa M jest:

$$\sum_{j=0}^{k} \binom{s-1}{j}$$

Odnosno, na zadnjih $s-1$ pozicija može biti 0 pogrešaka do $s-1$ pogreška. Skupovi M za k pogrešaka u sebi sadrže moguće skupove pridružene pogreškama $j < k$.

Ako uzmemo primjer s=4, k=2 dobivamo sljedeće vrijednosti skupa M po svim mogućim vrijednostima pogreške:

Tabela 1 – Sve moguće vrijednosti skupa M za j pogrešaka

Broj pogrešaka *j*	**Skup *M***	**Binarni zapis skupa *M***
0	{1, 2, 3}	111
1	{1, 2, 3}	111
	{1, 2}	110
	{1, 3}	101
	{2, 3}	011
2	{1, 2, 3}	111
	{1,2}	110
	{1, 3}	101
	{2, 3}	011
	{1}	100
	{2}	010
	{3}	001

Postupak računanja optimalnog praga za zadanu duljinu niza m uključuje ažuriranje ovih vrijednosti za svaku duljinu od s do m. Tako se prag za duljinu i računa pomoću vrijednosti pragova dobivenih za duljinu $i-1$, potom se vrijednosti za duljinu $i+1$ računaju pomoću vrijednosti za i, povećavajući tako i sve do m.
Potrebno je sve mogućnosti prikazane u tablici spremiti u odgovarajuću strukturu kako bi se lako moglo doći do vrijednosti za zadani broj pogrešaka j i skup M.
Kao odabranu strukturu koristili smo mapu koja kao ključ sadrži binarnu reprezentaciju skupa M, a kao vrijednost polje izračunatih pragova za svaki broj pogrešaka j za koji je skup M validan. Pri tome je na prvom mjestu u polju zapisan odgovarajući broj mogućih pogrešaka j što predstavlja i duljinu polja. Ostali elementi polja predstavljaju pragove za broj pogrešaka j poštivajući zahtjev da prva vrijednost odgovara broju pogrešaka k te svaka sljedeća vrijednosti za broj pogrešaka koji je za 1 manji od prethodnog. Ovo je bitno kako bi se jednostavno moglo doći do pozicije u polju koja odgovara vrijednosti za broj pogrešaka j, a računa se na sljedeći način:

$$1 + k - (\, s - 1 - \sum_{i=0}^{s-1} \mathbf{1}\{binary(M)[i]\})$$

Gdje je $\mathbf{1}\{P\}$ indikatorska funkcija čija je vrijednost jednaka 1 ako je $P \equiv \mathrm{T}$, a $binary(M)$ funkcija koja skup M pretvara u binarni zapis.

Struktura za gore navedeni primjer prema tome izgleda ovako:

111: [3, $threshold(2)$, $threshold(1)$, $threshold(0)$]
110: [2, $threshold(2)$, $threshold(1)$]
101: [2, $threshold(2)$, $threshold(1)$]
011: [2, $threshold(2)$, $threshold(1)$]
100: [1, $threshold(2)$]
010: [1, $threshold(2)$]
001: [1, $threshold(2)$]

Izraz $threshold(j)$ predstavlja vrijednost praga za broj pogrešaka j i skup M kojem je pridružen.

Konačni rezultat, nakon što je struktura ažurirana za duljinu m, pronalazi se tako da se za svaku vrijednost ključa pogleda izračunata vrijednost praga za točno k pogrešaka i od svih vrijednosti uzme se minimalna.

3.2. Generiranje uzoraka

Za zadanu duljinu uzorka q i raspon uzorka s, potrebno je generirati moguće vrijednosti (q, s)-uzorka za izračun praga. Uzorci su predstavljeni skupom Q.

3.2.1. Generiranje svih kombinacija uzoraka

Uzorci su izračunati variranjem svih mogućih kombinacija binarnog niza duljine $s - 2$ za zadane vrijednosti q i s. Krajnje pozicije uzorka nije potrebno uključiti u izračun kombinacija jer su one uvijek uključene u skup. Od svih kombinacija uzimaju se samo one čiji je broj jedinica jednak $q - 2$ te se indeksi tih jedinica u generiranom uzorku pomaknutih za 1 dodaju u skup. Konačan skup Q dobije se unijom dobivenog skupa sa pozicijama 0 i $s - 1$.

Primjerice, za s=4 i q=3, moguće su dvije kombinacije:

$\#\# - \#$ Q = {0, 1, 3}

$\# - \#\#$ Q = {0, 2, 3}

3.2.2. Generiranje (q, s)-uzoraka pomoću $(q - 1, s)$-uzoraka

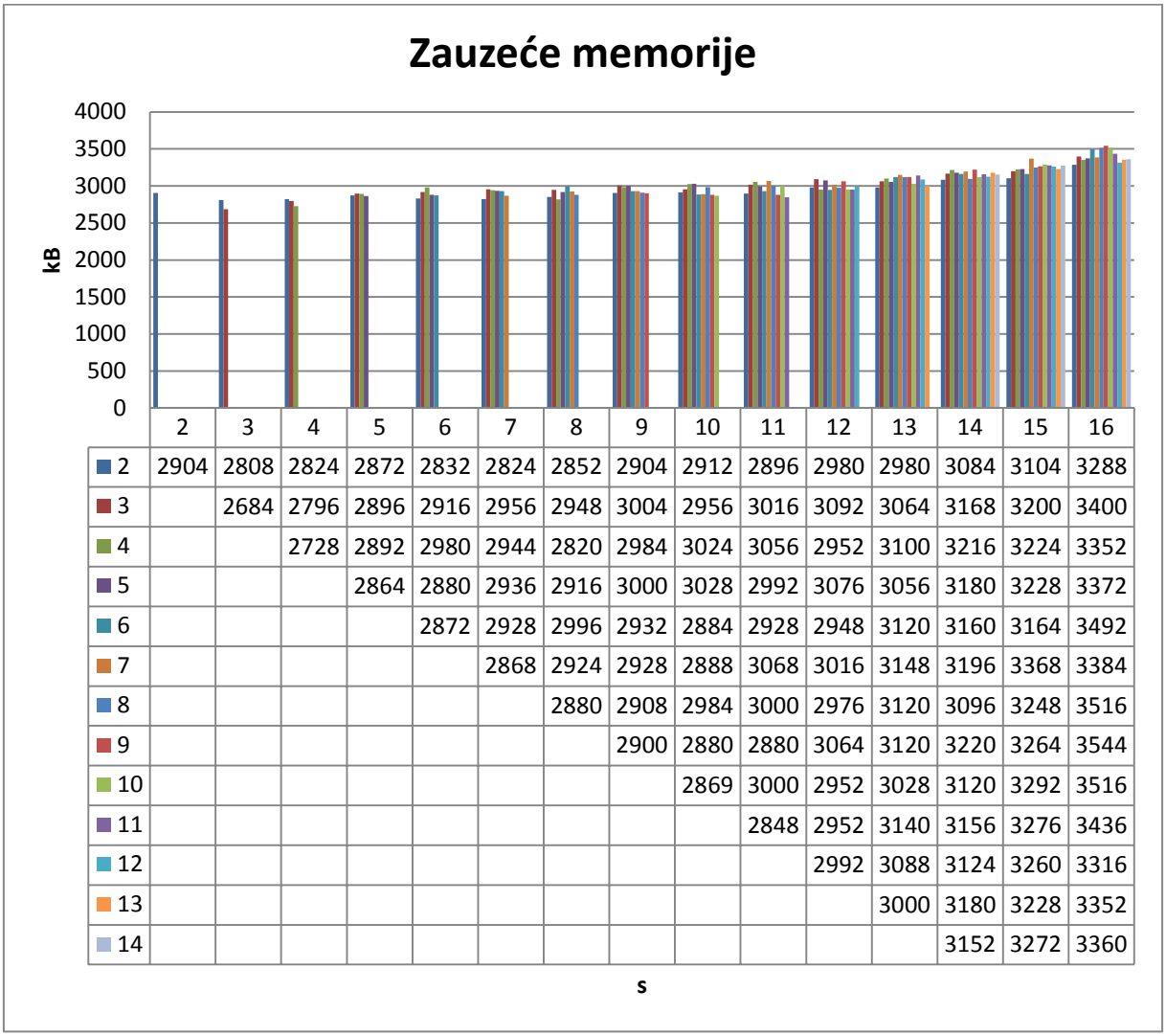
Izračun (q, s)-uzoraka pomoću $(q - 1, s)$-uzoraka rekurzivan je postupak koji se poziva umanjujući q sve dok vrijednost ne postane 3. Za q=3 uzorci se računaju generiranjem svih mogućih kombinacija. Za sve veće vrijednosti (q, s)-uzorci se generiraju kombinacijom $(q - 1, s)$-uzoraka iz prethodnog koraka za koje je izračunata pozitivna vrijednost praga.

Ti uzorci se odvoje u grupe na način da jednoj grupi pripadaju uzorci koji se razlikuju samo na $(q-1)-2$ poziciji. Unijom svakog para uzorka iz grupe dobiju se (q,s)-uzorci jer se svaki par grupe razlikuje samo na jednoj poziciji.

4. Testiranje

Testiranje je provedeno za broj razlika $k = 4$, duljinu niza $m = 50$ te za $s \in [2,14]$. Zbog značajnog usporenja izvođenja programa povećanjem q i s, prikazani su rezultati mjerenja za $q \in [2,16]$.

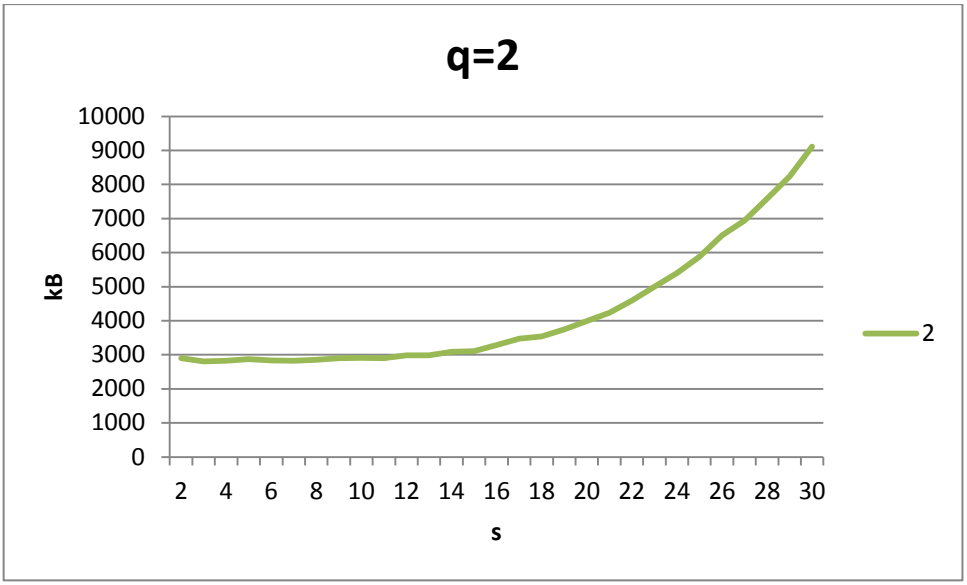
4.1. Zauzeće memorije



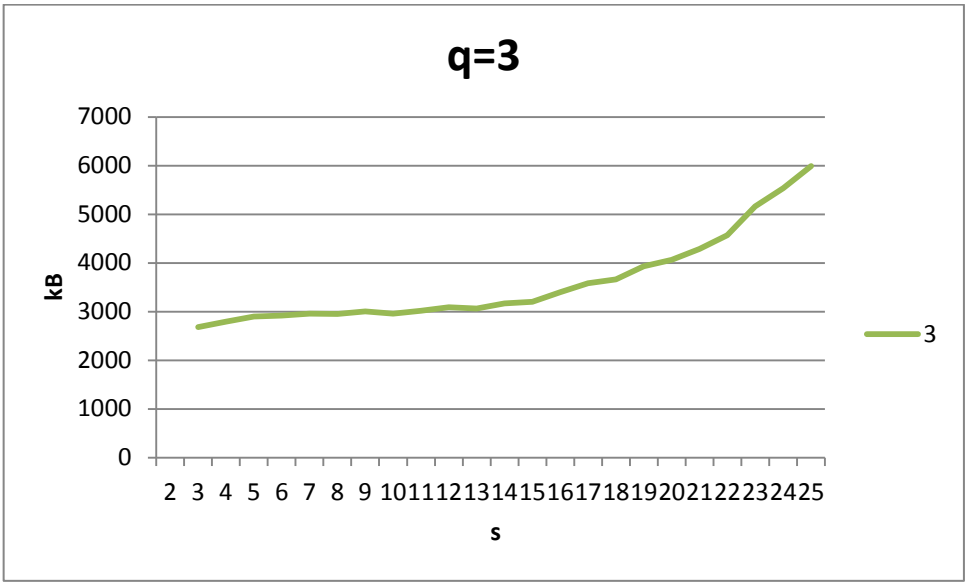
s	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	2904	2808	2824	2872	2832	2824	2852	2904	2912	2896	2980	2980	3084	3104	3288
3		2684	2796	2896	2916	2956	2948	3004	2956	3016	3092	3064	3168	3200	3400
4			2728	2892	2980	2944	2820	2984	3024	3056	2952	3100	3216	3224	3352
5				2864	2880	2936	2916	3000	3028	2992	3076	3056	3180	3228	3372
6					2872	2928	2996	2932	2884	2928	2948	3120	3160	3164	3492
7						2868	2924	2928	2888	3068	3016	3148	3196	3368	3384
8							2880	2908	2984	3000	2976	3120	3096	3248	3516
9								2900	2880	2880	3064	3120	3220	3264	3544
10									2869	3000	2952	3028	3120	3292	3516
11										2848	2952	3140	3156	3276	3436
12											2992	3088	3124	3260	3316
13												3000	3180	3228	3352
14													3152	3272	3360

Grafikon 1 - Mjerenje zauzeća memorije

Iz Grafikona 1 vidljivo je postojanje rasta zauzete memorije. Taj rast bolje je uočljiv u grafikonima 2 i 3.

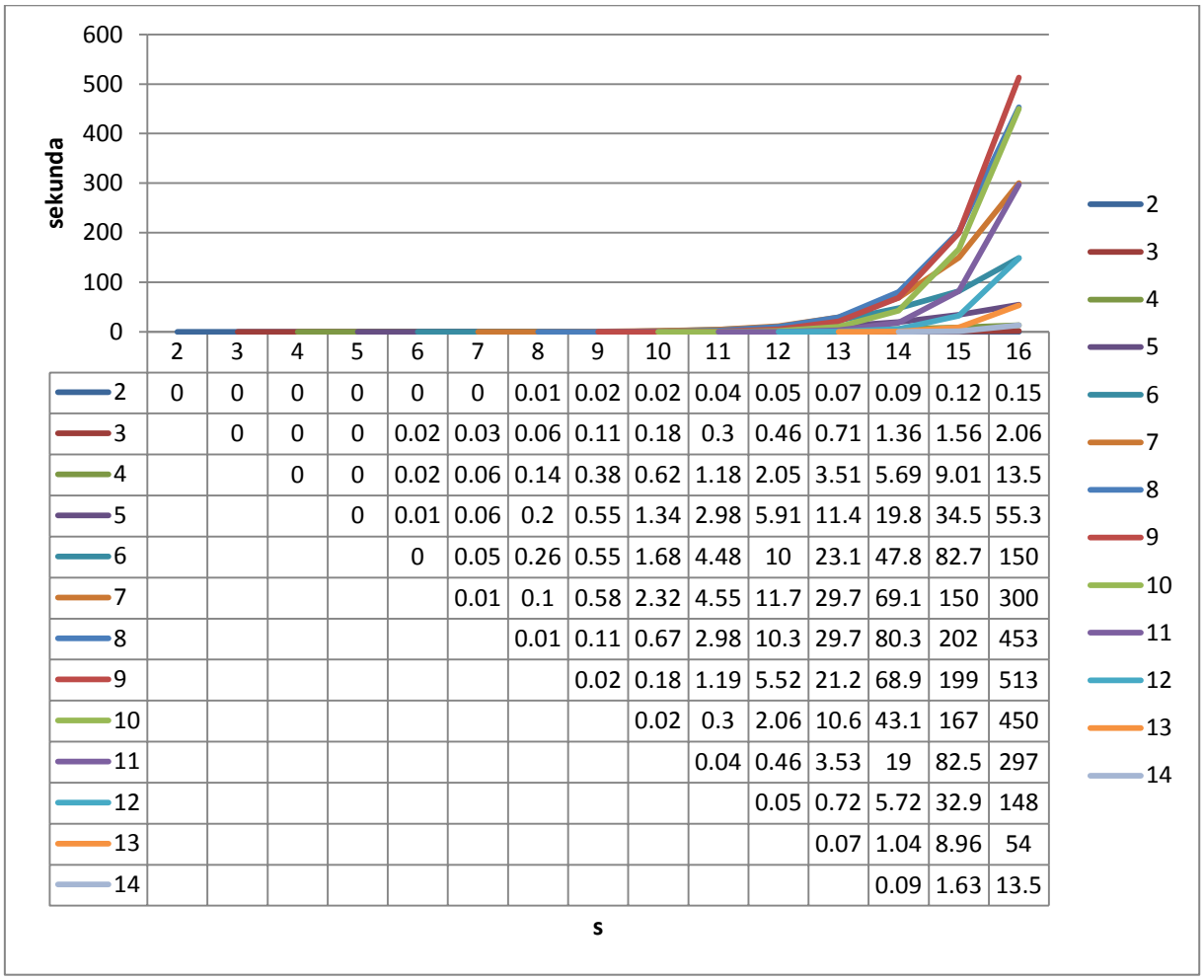


Grafikon 2 – Prikaz zauzeća memorije za duljinu uzorka q = 2 i raspon uzoraka s € [2, 30]



Grafikon 3 - Prikaz zauzeća memorije za duljinu uzorka q = 3 i raspon uzoraka s € [3, 25]

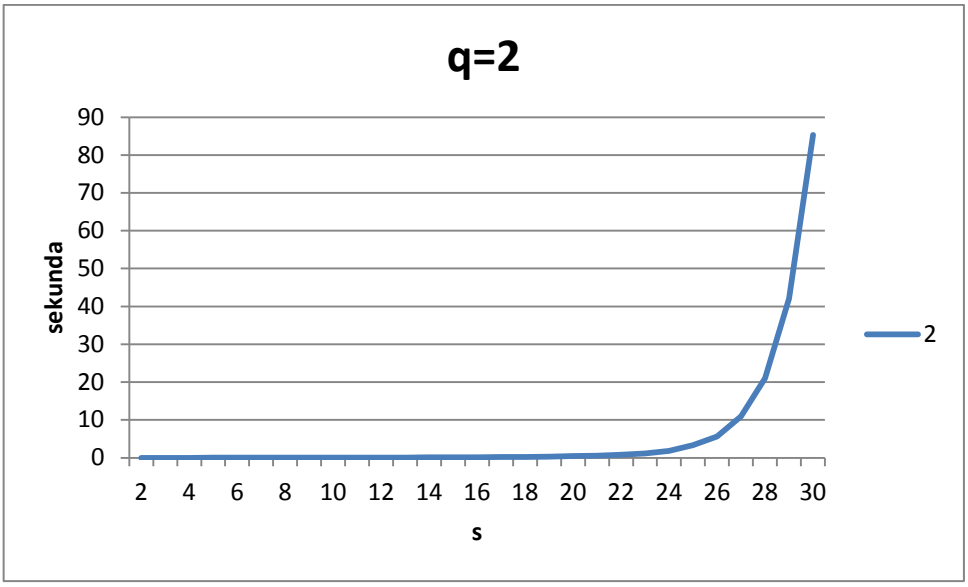
4.2. Vrijeme izvođenja



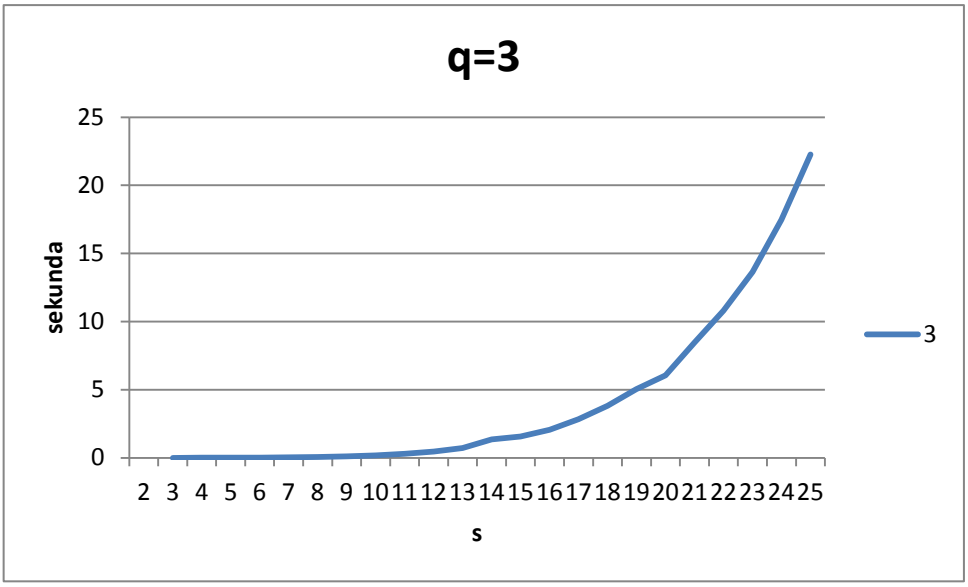
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	0	0	0	0	0	0	0.01	0.02	0.02	0.04	0.05	0.07	0.09	0.12	0.15
3		0	0	0	0.02	0.03	0.06	0.11	0.18	0.3	0.46	0.71	1.36	1.56	2.06
4			0	0	0.02	0.06	0.14	0.38	0.62	1.18	2.05	3.51	5.69	9.01	13.5
5				0	0.01	0.06	0.2	0.55	1.34	2.98	5.91	11.4	19.8	34.5	55.3
6					0	0.05	0.26	0.55	1.68	4.48	10	23.1	47.8	82.7	150
7						0.01	0.1	0.58	2.32	4.55	11.7	29.7	69.1	150	300
8							0.01	0.11	0.67	2.98	10.3	29.7	80.3	202	453
9								0.02	0.18	1.19	5.52	21.2	68.9	199	513
10									0.02	0.3	2.06	10.6	43.1	167	450
11										0.04	0.46	3.53	19	82.5	297
12											0.05	0.72	5.72	32.9	148
13												0.07	1.04	8.96	54
14													0.09	1.63	13.5

Grafikon 4 – Prikaz vremena izvođenja programa

Grafikon 4 pokazuje da vrijeme izvršavanje programa raste linearno do neke vrijednosti raspona s nakon čega počinje eksponencijalno rasti. U grafikonima 5 i 6 izdvojen je prikaz rasta za q=2 odnosno q=3.



Grafikon 5 – Prikaz vremena računanja optimalnog praga za duljinu uzorka q=2 i raspon uzoraka s € [2, 30]

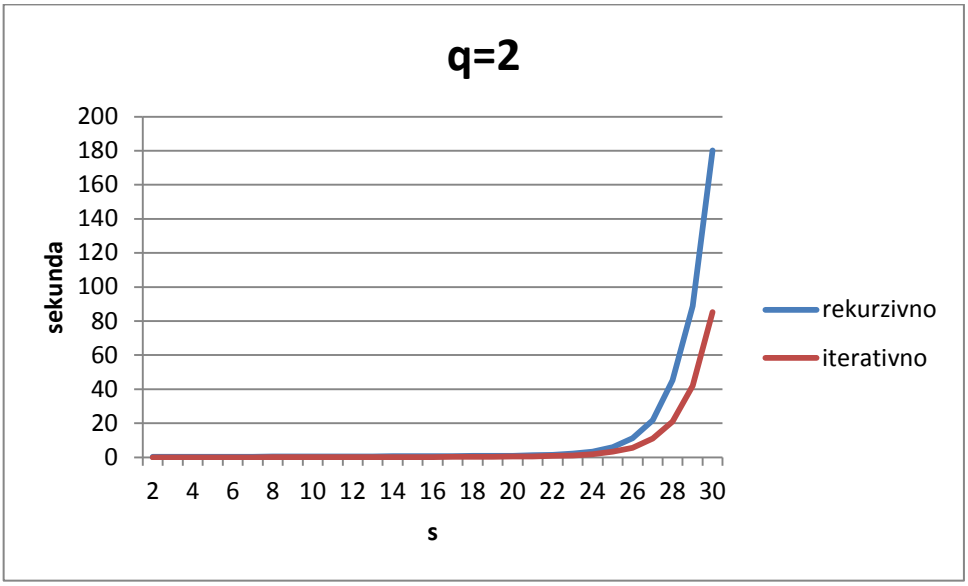


Grafikon 6 - Prikaz vremena računanja optimalnog praga za duljinu uzorka q=3 i raspon uzoraka s € [3, 25]

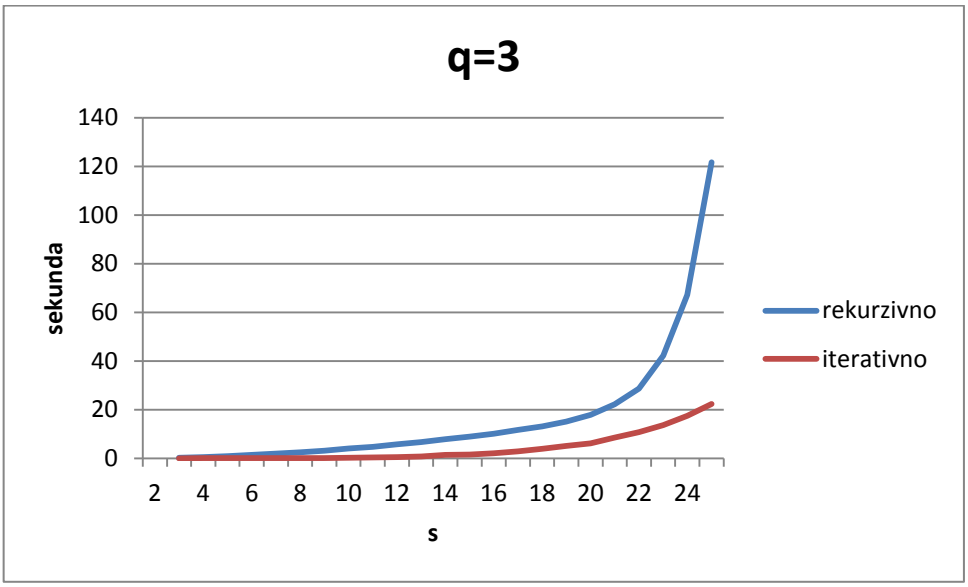
4.3. Usporedba rekurzivnog algoritma s algoritmom dinamičkog programiranja

Na sljedećim grafikonima (Grafikon 7 – Grafikon 14) uočljivo je veliko ubrzanje i nešto manje zauzeće memorije koje je postignuto promjenom algoritma iz rekurzivnog u algoritam dinamičkog programiranja.

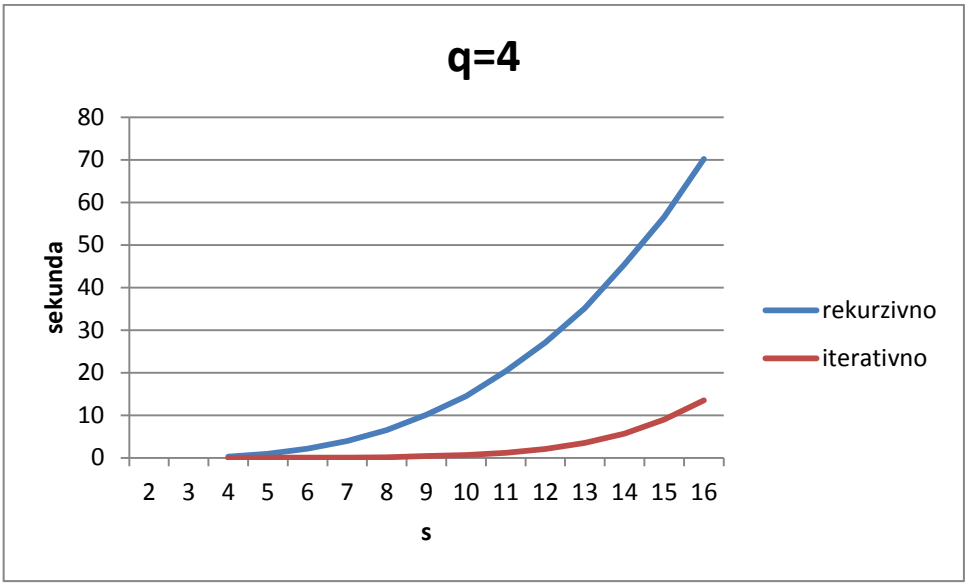
4.3.1. Vrijeme izvođenja



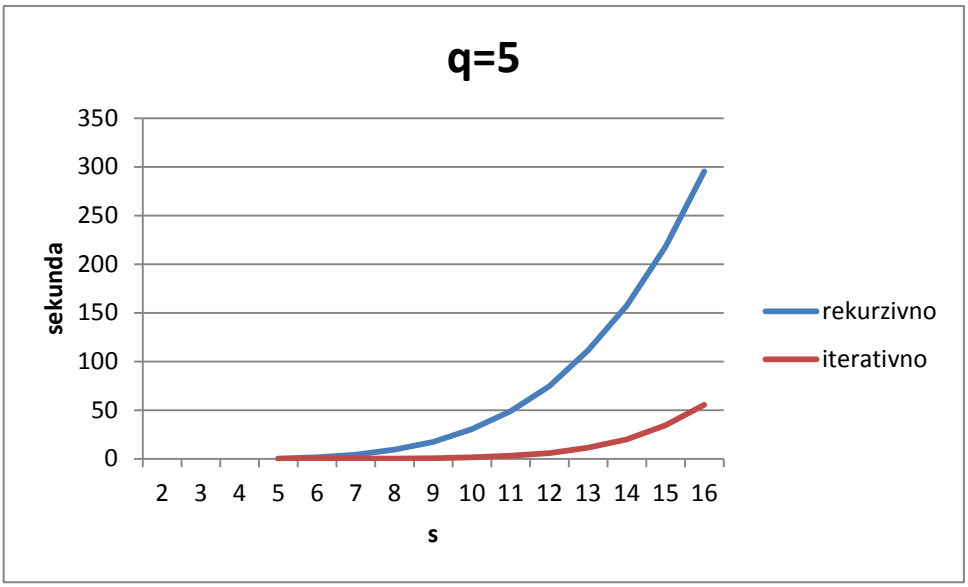
Grafikon 7 - Usporedba dinamičkog i rekurzivnog algoritma za vrijeme izvođenja uz q=2.



Grafikon 8 - Usporedba dinamičkog i rekurzivnog algoritma za vrijeme izvođenja uz q=3.

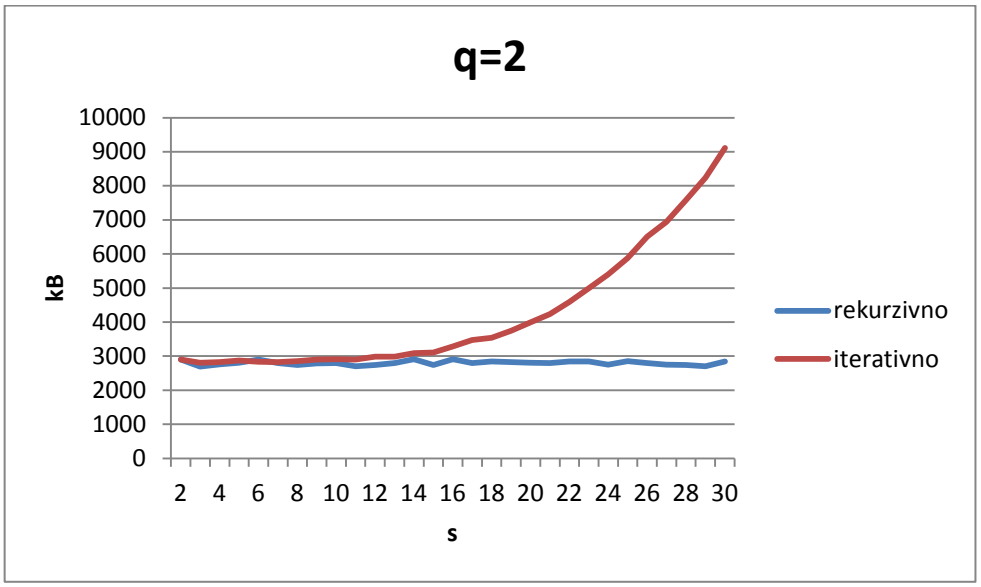


Grafikon 9 - Usporedba dinamičkog i rekurzivnog algoritma za vrijeme izvođenja uz q=4.

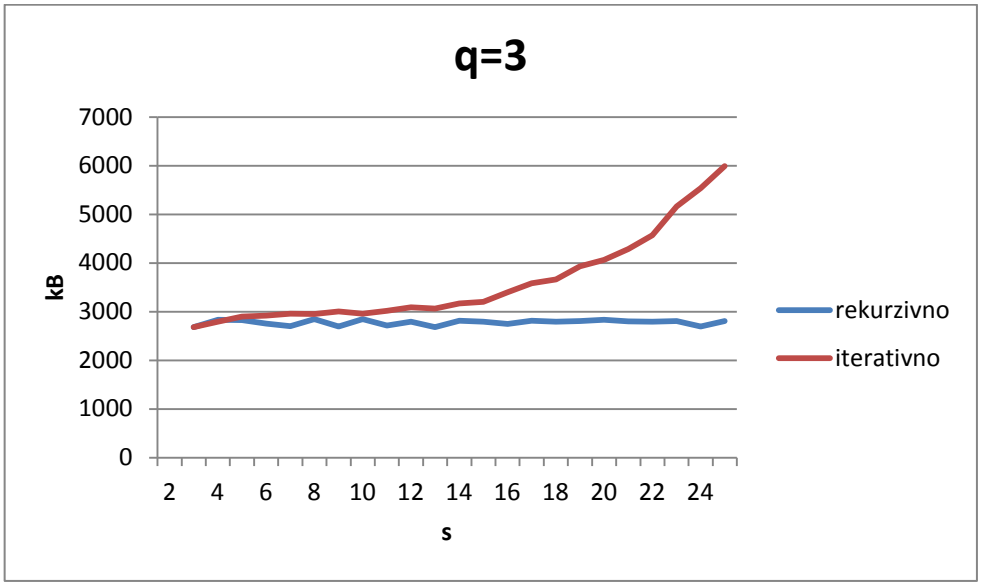


Grafikon 10 - Usporedba dinamičkog i rekurzivnog algoritma za vrijeme izvođenja uz q=5.

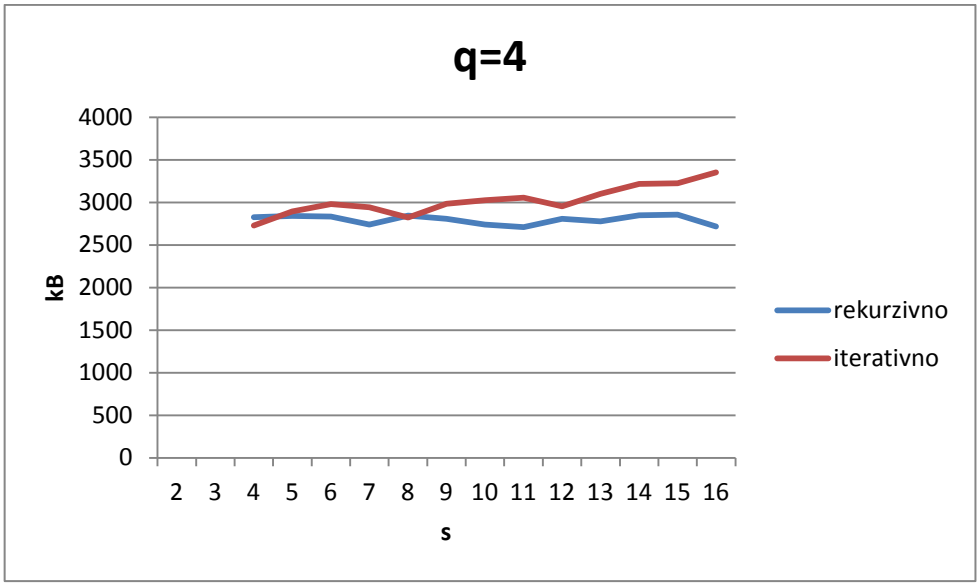
4.3.2. Zauzeće memorije



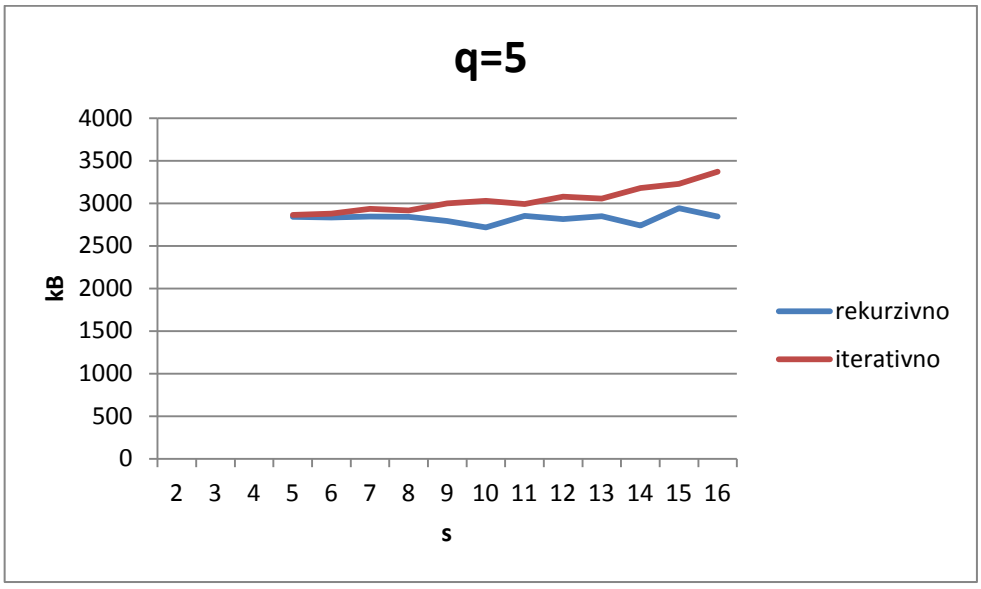
Grafikon 11 - Usporedba dinamičkog i rekurzivnog algoritma za vrijeme izvođenja uz q=2.



Grafikon 12 - Usporedba dinamičkog i rekurzivnog algoritma za vrijeme izvođenja uz q=3.



Grafikon 13 - Usporedba dinamičkog i rekurzivnog algoritma za vrijeme izvođenja uz q=4.



Grafikon 14 - Usporedba dinamičkog i rekurzivnog algoritma za vrijeme izvođenja uz q=5.

5. Zaključak

Rad je dao kratak uvid u izračun optimalnog praga za Q-gram filtre. Testiranje je potvrdilo očekivanja, izračun dinamičkim programiranjem mnogostruko je brži od rekurzivnog.

Generiranje skupa uzoraka pomoću prethodnih skupova s pozitivnim pragom rezultiralo je iako točnim, dosta sporijim izračunima za nevelike vrijednosti raspona s i duljine uzorka q. S porastom raspona i duljine uzorka q očekuje se veće ubrzanje s navedenim generiranjem.

Ovo interesantno područje još nije pretjerano istraženo te se autori nadaju da će ovaj rad pobuditi zanimanje čitatelja.

6. Literatura

[1] Burkhardt, Stefan i Kärkkäinen, Juha. *Better Filtering with Gapped q-Grams*. (2003): 1001-1020 Preuzeto: 16.10.2015., Poveznica: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.5942&rep=rep1&type=pdf

7. Sažetak

Kako bi se ubrzao proces pretraživanja teksta prvo se obavlja filtriranje, a jedno od mogućih je Q-gram filtriranje. Q-gram je podniz duljine q. Optimalan prag je najveći prag kod kojeg ne dolazi do gubitka podataka. Računanje optimalnog praga q-gram algoritma moguće je postići na dva načina, rekurzivnim izračunom te dinamičkim programiranjem. Implementacija oba načina napravljena je u C++ programu. Testiranjem je utvrđeno da oscilacije u zauzeću memorije postoje, no nisu pretjerano izražene, dok se vrijeme izvođenja vidno povećalo za velike broj uzoraka koje je potrebno ispitati s povećanjem raspona s.