Efektívna implementácia komprimovaných bitových polí

Bc. Andrej Korman
Školiteľ: Mgr. Jakub Kováč, PhD.

9. júna 2022

Bitové pole

- $access(i)$ – bit na i-tej pozícii
- $rank_b(i)$ – počet jednotiek(núl) od začiatku po i-tu pozíciu
- $select_b(i)$ – pozíciu i-tej jednotky(nuly)

Motivácia pre náš záujem

Úsporné dátové štruktúry

- dátové štruktúry kladúce dôraz na úsporu pamäte

\+ vyhnutie sa pomalým diskovým operáciám

– rýchlosť samotných metód

– zložitejšia implementácia

Ciele práce

- preskúmať existujúce implementácie komprimovaného bitového poľa
- navrhnúť vylepšenia aktuálnych riešení
- implementovať a experimentálne otestovať navrhované zmeny

Myšlienka RRR - Raman et al. (2007)

0 1 1 0 1 1 1 0 1 1 0 1 0 1 1 0 1 0 0 0

Myšlienka RRR

0 1 1 0 1 1 1 0 1 1 0 1 0 1 1 0 1 0 0 0

0 1 1 0	**1 1 1 0**	**1 1 0 1**	**0 1 1 0**	**1 0 0 0**

Myšlienka RRR

0 1 1 0 1 1 1 0 1 1 0 1 0 1 1 0 1 0 0 0

0 1 1 0	**1 1 1 0**	**1 1 0 1**	**0 1 1 0**	**1 0 0 0**

kódujeme ako (#jednotiek, poradie)

Myšlienka RRR

0 1 1 0 1 1 1 0 1 1 0 1 0 1 1 0 1 0 0 0

0 1 1 0	**1 1 1 0**	**1 1 0 1**	**0 1 1 0**	**1 0 0 0**

kódujeme ako (#jednotiek, poradie)

k = 0:	**k = 1**:	**k = 2**:	**k = 3**:	**k = 4**:
1. 0000	1. 0001	1. 0011	1. 0111	1. 1111
	2. 0010	2. 0101	2. 1011	
	3. 0100	3. 0110	3. 1101	
	4. 1000	4. 1001	4. 1110	
		5. 1010		
		6. 1100		

Myšlienka RRR

0 1 1 0 1 1 1 0 1 1 0 1 0 1 1 0 1 0 0 0

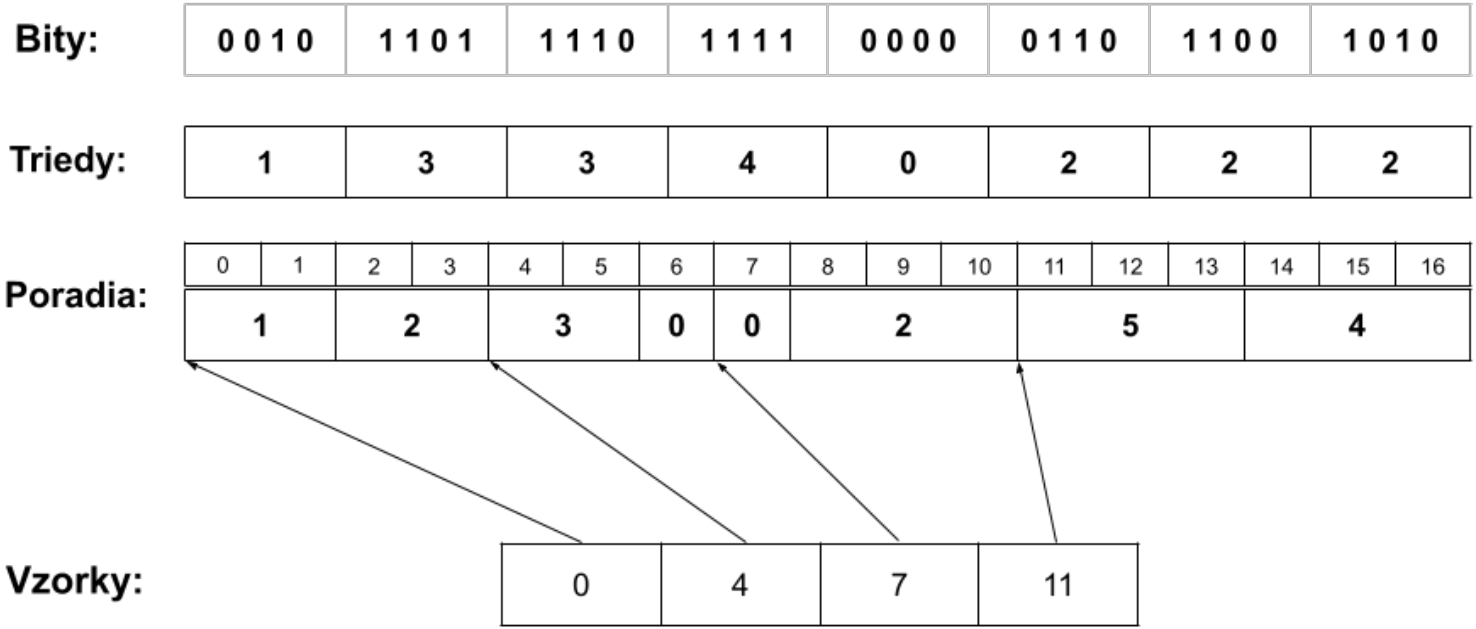
0 1 1 0	**1 1 1 0**	**1 1 0 1**	**0 1 1 0**	**1 0 0 0**

kódujeme ako (#jednotiek, poradie)

(2, 3)	**(3, 4)**	**(3, 3)**	**(2, 3)**	**(1, 4)**

k = 0:	**k = 1**:	**k = 2**:	**k = 3**:	**k = 4**:
1. 0000	1. 0001	1. 0011	1. 0111	1. 1111
	2. 0010	2. 0101	2. 1011	
	3. 0100	3. 0110	3. 1101	
	4. 1000	4. 1001	4. 1110	
		5. 1010		
		6. 1100		

RRR - reprezentácia



Úspornosť RRR

Dekódovanie bloku

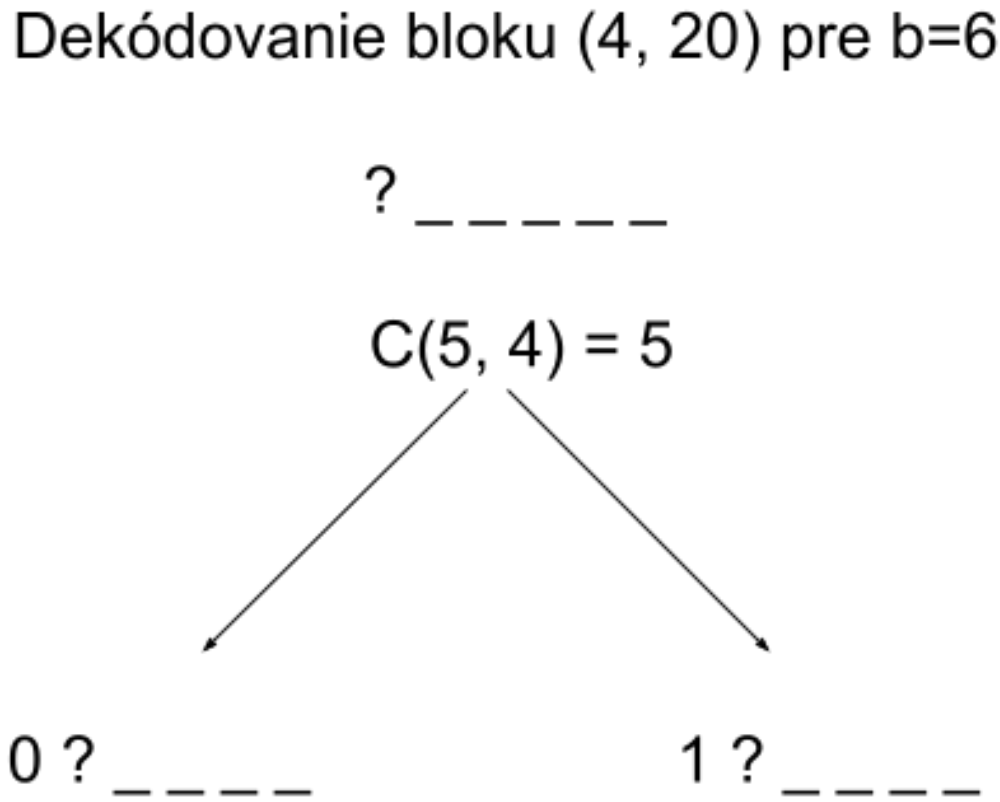
Úloha

Premeniť dvojicu (t, p) na pôvodný bitový reťazec.

Dekódovanie bloku - tabuľka

	trieda	poradie	blok
0	0	0	0000
1	1	0	0001
2	1	1	0010
3	1	2	0100
4	1	3	1000
5	2	0	0011
6	2	1	0101
7	2	2	0110
8	2	3	1001
9	2	4	1010
10	2	5	1100
11	3	0	0111
12	3	1	1011

Dekódovanie bloku - bit po bite (Navarro and Providel, 2012)



Nová dekódovacia metóda.

RRR - nové usporiadanie

- tabuľka pre blok dĺžky 6 a triedu 2:

Poradie	Blok	(t_1, t_2)
0	000 011	
1	000 101	$(0, 2)$
2	000 110	
3	001 001	
4	001 010	
5	001 100	$(1, 1)$
6	010 001	
7	010 010	

Poradie	Blok	(t_1, t_2)
8	010 100	
9	100 001	$(1, 1)$
10	100 010	
11	100 100	
12	011 000	
13	101 000	$(2, 0)$
14	110 000	

RRR - nové usporiadanie

- tabuľka pre blok dĺžky 6 a triedu 2:

Poradie	Blok	(t_1, t_2)
0	000 011	
1	000 101	$(0, 2)$
2	000 110	
3	001 001	
4	001 010	
5	001 100	$(1, 1)$
6	010 001	
7	010 010	

Poradie	Blok	(t_1, t_2)
8	010 100	
9	100 001	$(1, 1)$
10	100 010	
11	100 100	
12	011 000	
13	101 000	$(2, 0)$
14	110 000	

(t_1, t_2)	(0,2)	(1,1)	(2,0)
počet	3	9	3
prefix	3	12	15

Metóda hybridného kódovania.

Hybridné kódovanie - motivácia

Varianty hybridného kódovania

Dvojstranné hybridné kódovanie - 7(3):

trieda: 0 1 2 3 4 5 6 7
mapovanie: 0 1 3 3 3 3 3 2

Jednostranné hybridné kódovanie - 7(3):

trieda: 0 1 2 3 4 5 6 7
mapovanie: 0 1 2 3 3 3 3 3

RRR - hybridné riešenie

0 1 1 0 1 1 1 0 1 1 0 1 0 0 1 0 1 1 1 1

0 1 1 0	**1 1 1 0**	**1 1 0 1**	**0 0 1 0**	**1 1 1 1**

jednostranný variant - 4(3)

2, 3	**3, 1110**	**3, 1101**	**1, 2**	**3, 1111**

k = 0:	**k = 1**:	**k = 2**:	**k = 3**:	**k = 4**:
1. 0000	1. 0001	1. 0011	1. 0111	1. 1111
	2. 0010	2. 0101	2. 1011	
	3. 0100	3. 0110	3. 1101	
	4. 1000	4. 1001	4. 1110	
		5. 1010		
		6. 1100		

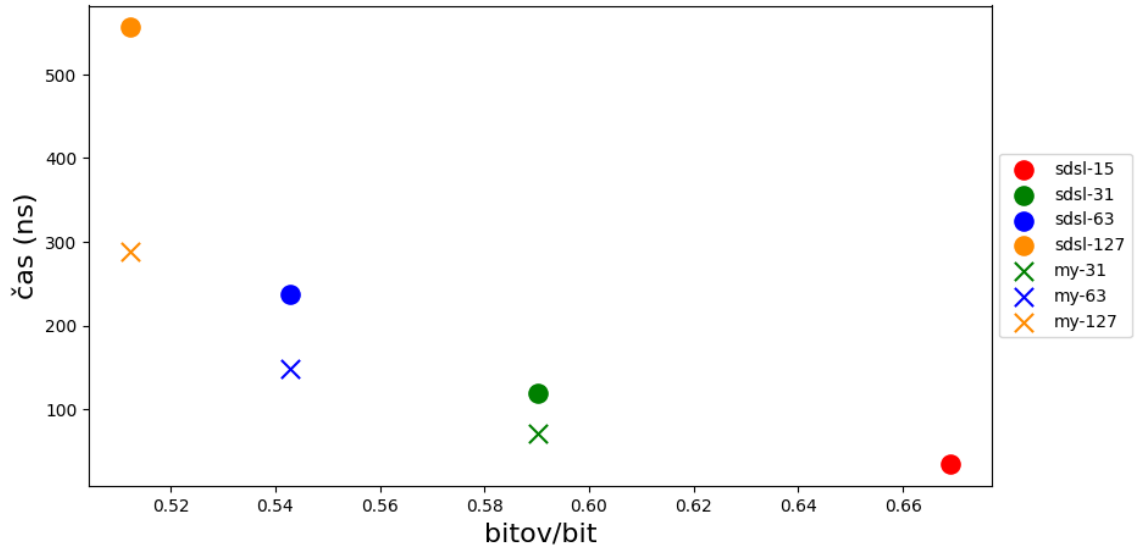
Implementácia a experimentálne výsledky.

Použité testy

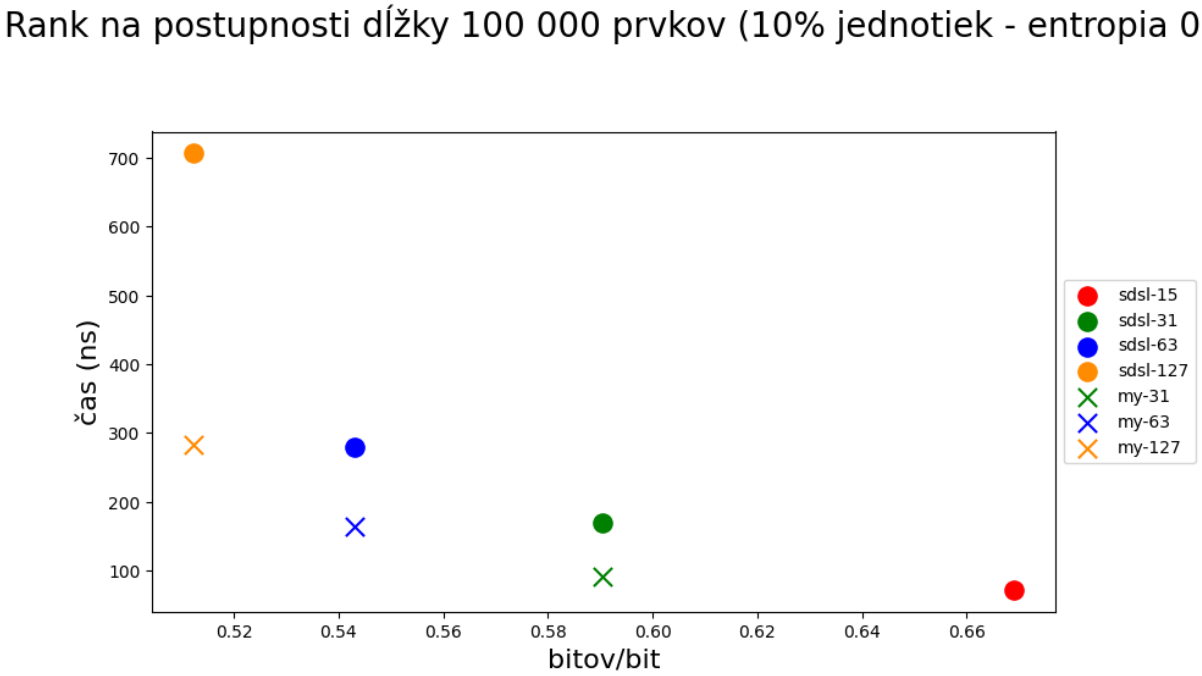
- Mikromerania dekódovania (Google Benchmark)
- Merania na náhodných dátach (Google Benchmark)
- Merania na reálnych dátach (SDSL, Gog et al. (2014))

Bitové pole - náhodné dáta

Access na postupnosti dĺžky 100 000 prvkov (10% jednotiek - entropia 0.47)



Bitové pole - náhodné dáta

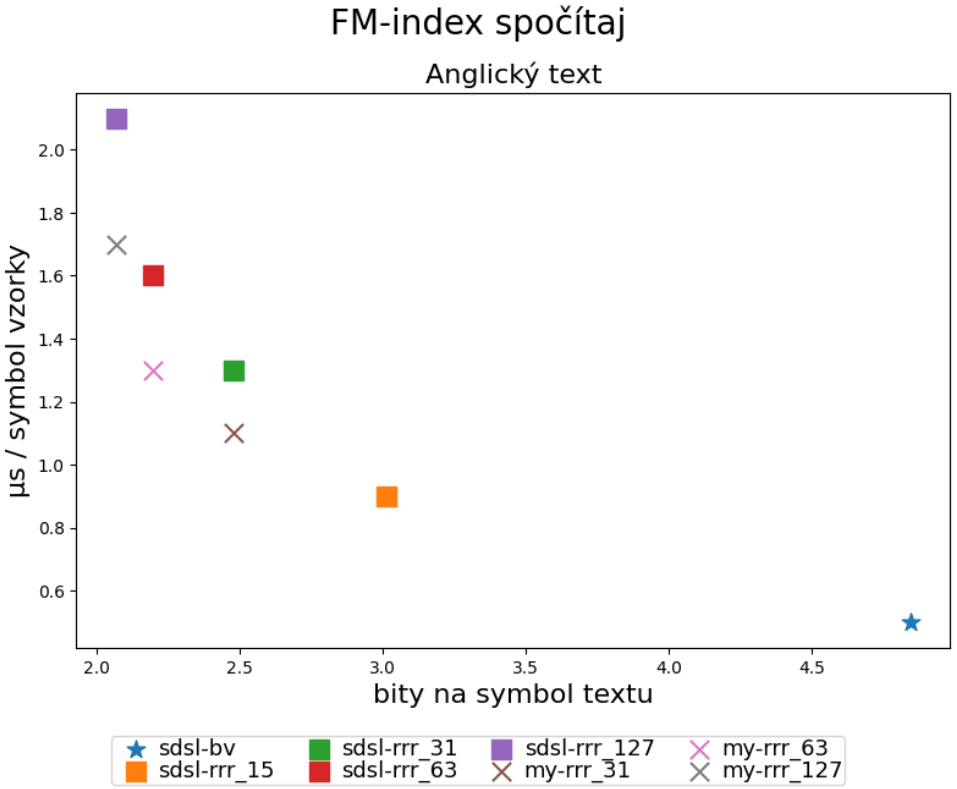


FM-index

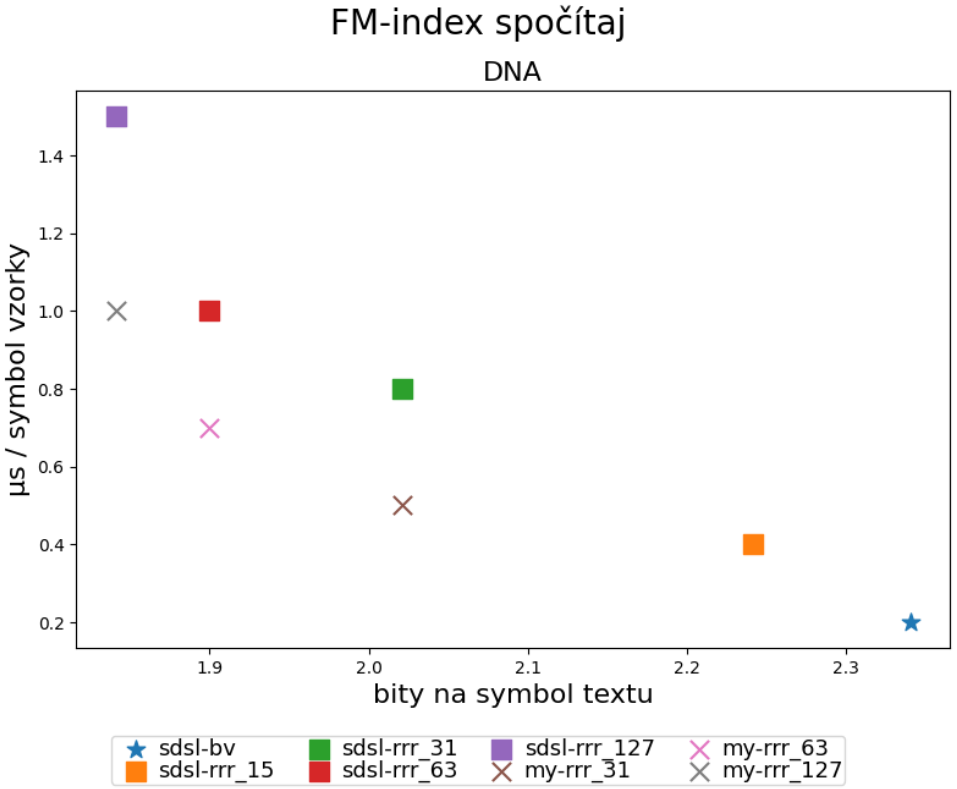
Podporované metódy nad textom T

- $spočítaj(P)$ – počet výskytov P v T
- $lokalizuj(P)$ – výskyty P v T
- $extrahuj(i, j)$ – podreťazec T od i-tej po j-tu pozíciu

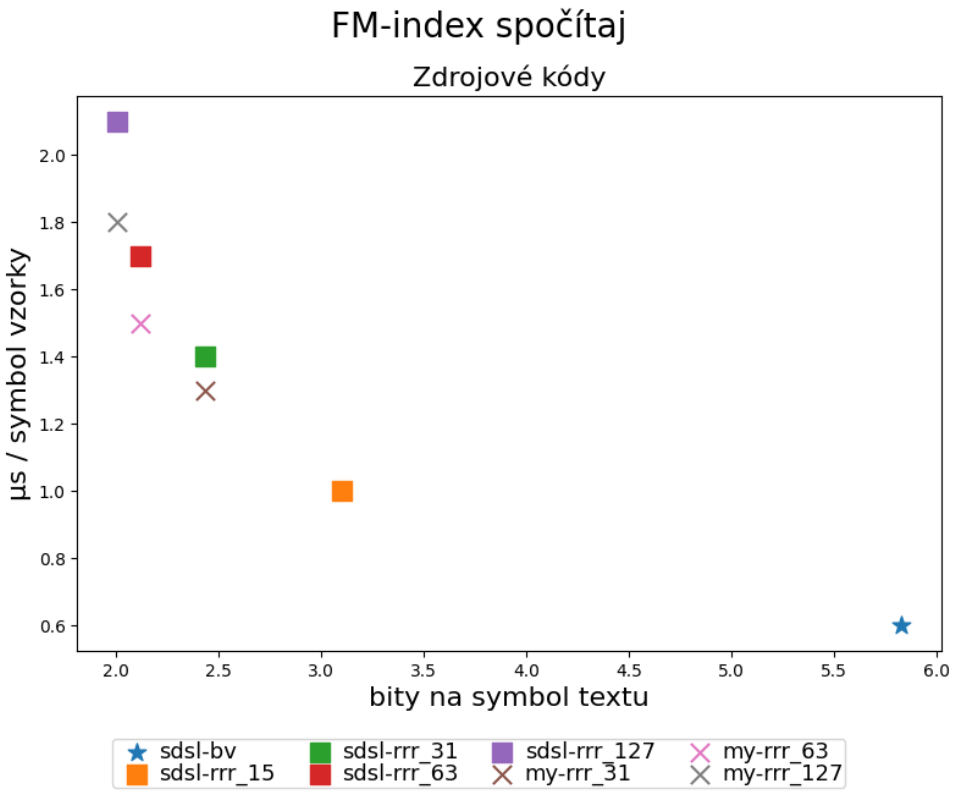
FM-index - počítanie výskytov



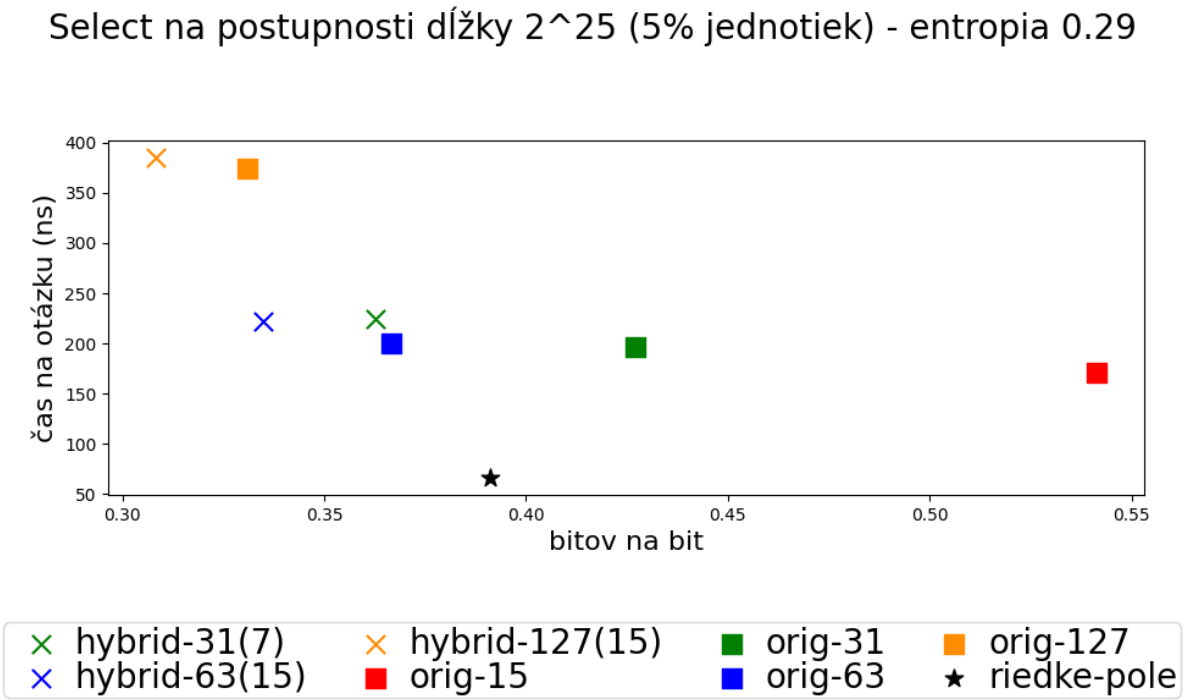
FM-index - počítanie výskytov



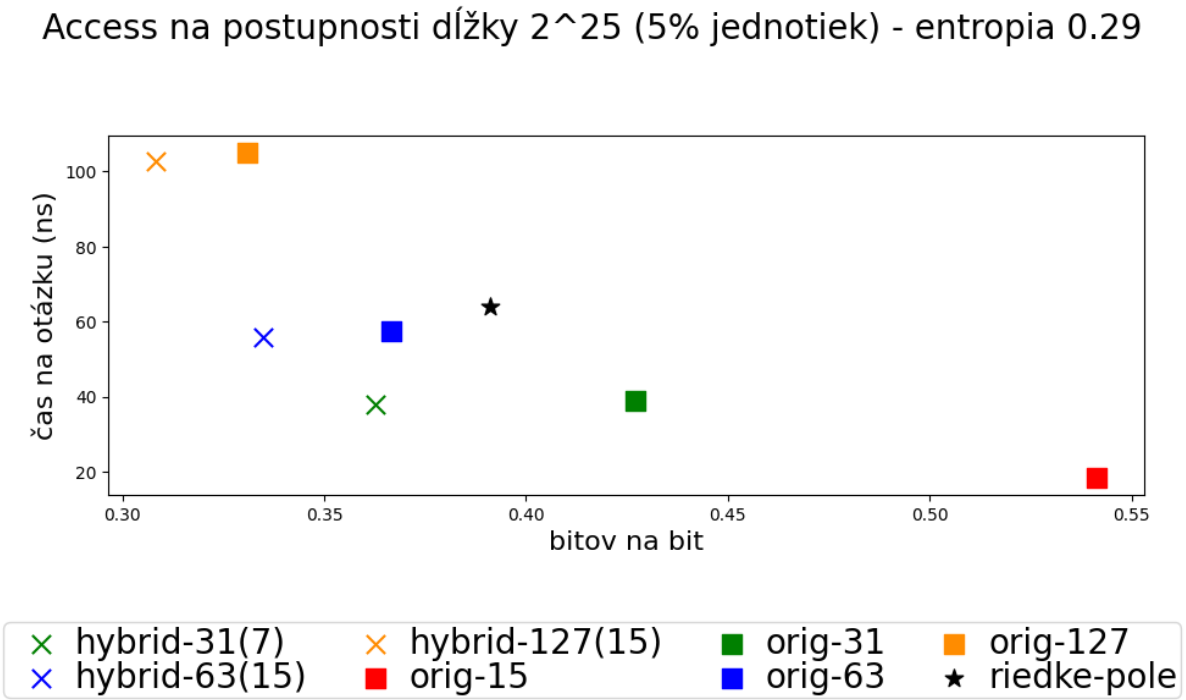
FM-index - počítanie výskytov



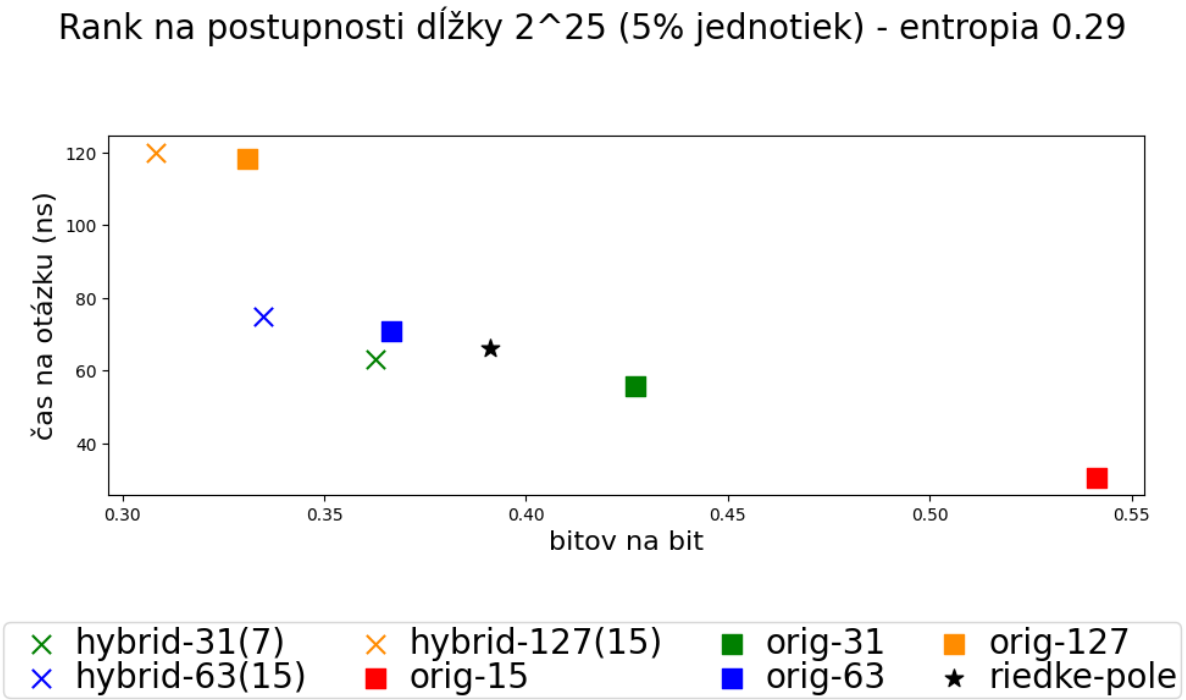
Hybridná implementácia - *select*



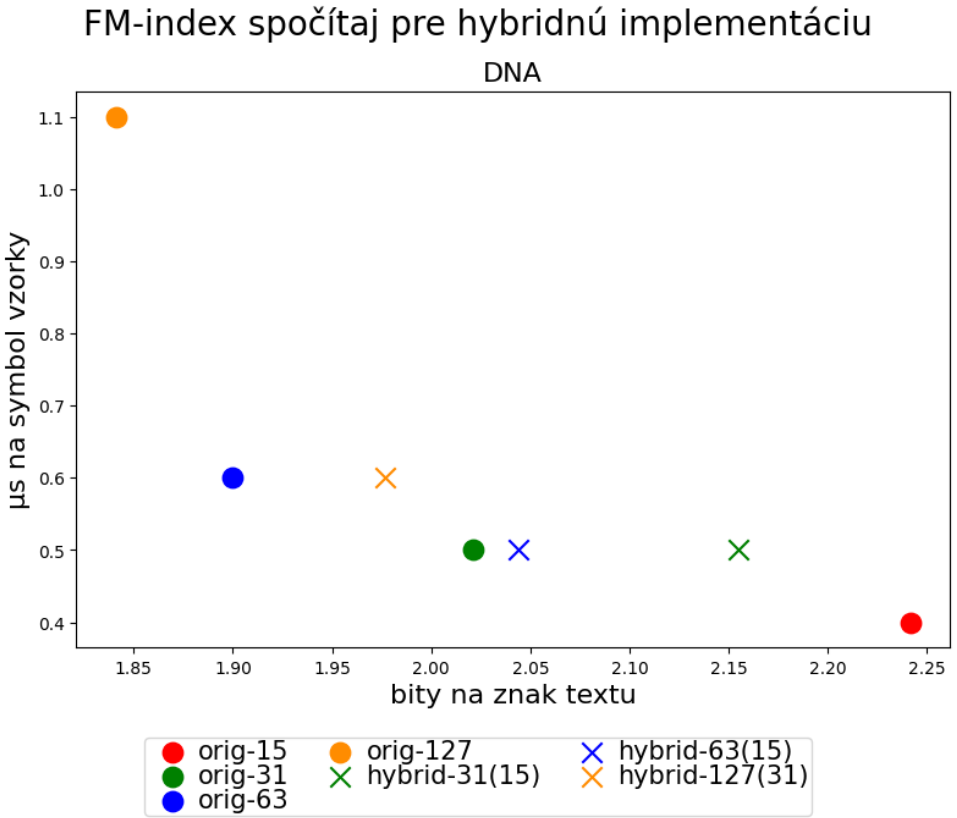
Hybridná implementácia - *access*



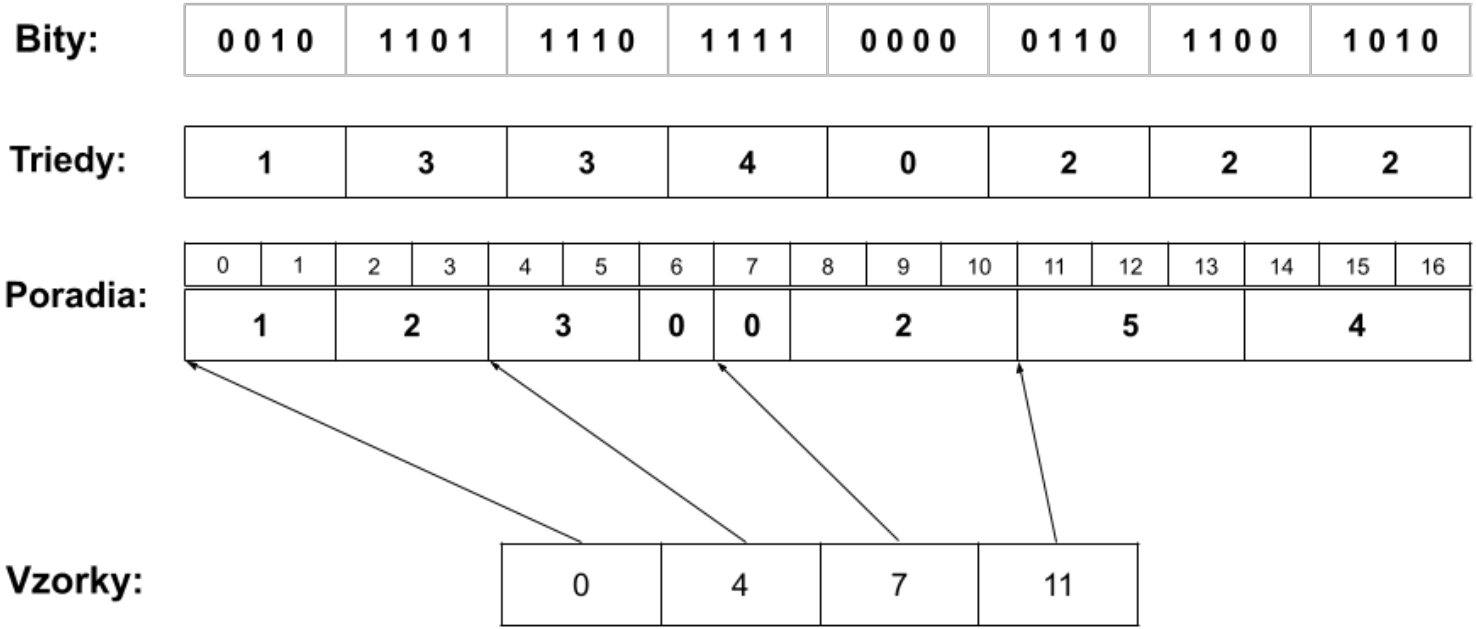
Hybridná implementácia - *rank*



Hybridná implementácia - obojstranná verzia



RRR - reprezentácia



Možnosti ďalšej práce

- adaptívna kombinácia klasického a hybridného riešenia
- porovnanie s inými implementáciami FM-indexu
- špeciálna verzia bloku so 4/8 podblokmi

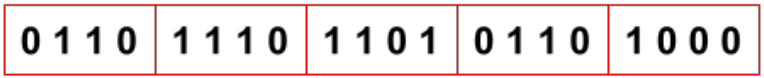
Ďakujem za pozornosť.

Porovnanie s prácou Kaneta (2017)

- prístup rozdeľ a panuj
- prezentovaná možnosť kombinovať kódovania pre podbloky
- náhodné dáta, reálne dáta, FM-index
- prístupná implementácia
- 127 bitový blok
- hybridné kódovanie

- lineárny prístup
- náhodné dáta

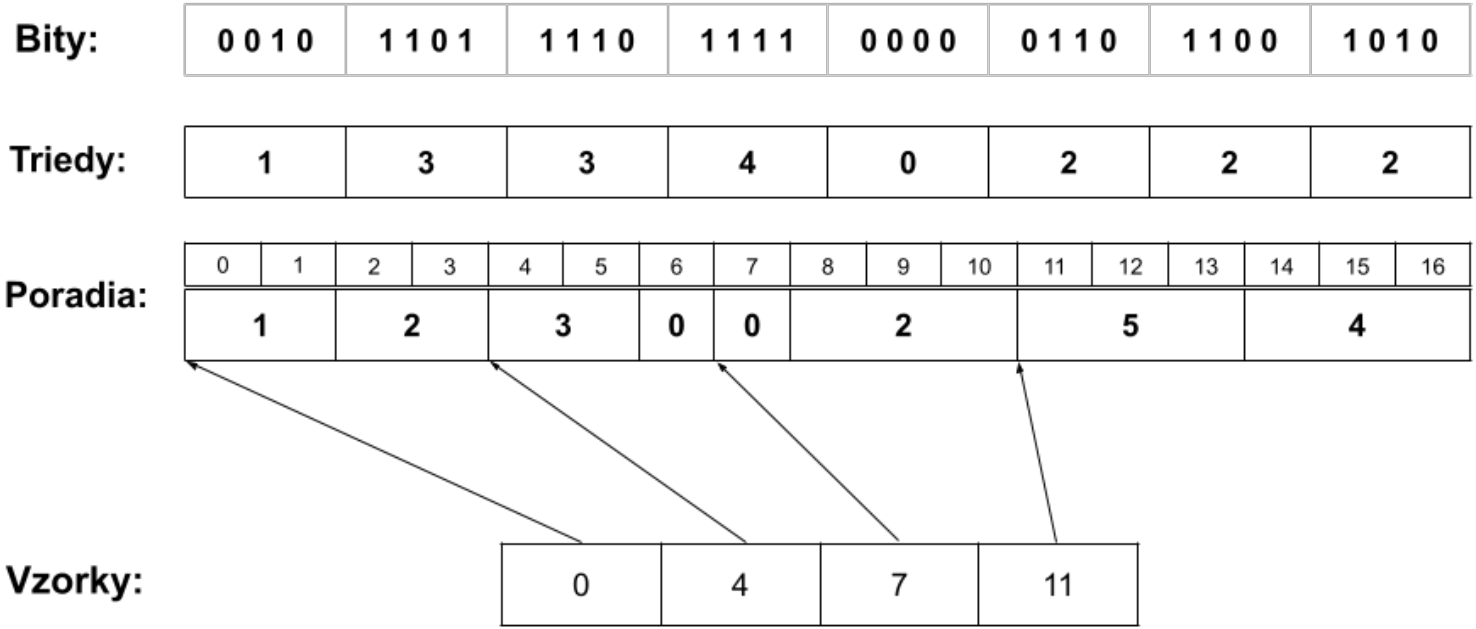
Porovnanie s prácou Kaneta (2017)



Odpovede na otázky oponenta

- Namiesto hybridnej reprezentácie by bolo možné namiesto binárnej reprezentácie počtu jednotiek použiť napr. huffmanovské kódovanie. Aké by boli výhody a nevýhody takéhoto prístupu?
- Veľkosť bloku má tvar $2^k - 1$, aby bolo možné reprezentovať počet jednotiek efektívne pomocou k-bitov. V prípade hybridného kódovania toto nie je dôležité. Nie je túto skutočnosť možné využiť a zvoliť inú, výhodnú, veľkosť bloku?

RRR - reprezentácia



Implementácia

Experimenty

github.com/Aj0SK/master-thesis

Verzia SDSL s našou implementáciou

github.com/Aj0SK/sdsl-lite

References I

Gog, S., T. Beller, A. Moffat, and M. Petri (2014). From theory to practice: Plug and play with succinct data structures. In *International Symposium on Experimental Algorithms*, pp. 326–337. Springer.

Kaneta, Y. (2017). Faster practical block compression for rank/select dictionaries. In *International Symposium on String Processing and Information Retrieval*, pp. 234–240. Springer.

Navarro, G. and E. Providel (2012). Fast, small, simple rank/select on bitmaps. In *International Symposium on Experimental Algorithms*, pp. 295–306. Springer.

Raman, R., V. Raman, and S. R. Satti (2007). Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms (TALG) 3*(4), 43–es.

Počítač 1

Parameter	**Hodnota**
Architecture	x86_64
CPU(s)	16
Thread(s) per core	2
Core(s) per socket	8
Socket(s):	1
Model name	AMD Ryzen 7 2700X Eight-Core Processor
L1d cache	256 KiB
L1i cache	512 KiB
L2 cache	4 MiB
L3 cache	16 MiB
Memory RAM	16 GB DDR4

Počítač 2

Parameter	**Hodnota**
Architecture	x86_64
CPU(s)	8
Thread(s) per core	2
Core(s) per socket	4
Socket(s):	1
Model name	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
L1d cache	128 KiB
L1i cache	128 KiB
L2 cache	1 MiB
L3 cache	8 MiB
Memory RAM	16 GB DDR3

Operácie rank a select

Definícia

Nech S je reťazec (konečná postupnosť) symbolov z abecedy Σ dĺžky n. Potom definujeme operácie `rank` a `select` takto:

- $rank_c(x) = |\{k; k \leq x : S[k] = c\}|$
- $select_c(x) = \min(\{k; k \in \{1, 2, \ldots, n\} : rank_c(k) = x\})$

Neformálne povedané:

- $rank_c(i)$ = počet symbolov c od začiatku po i-tu pozíciu
- $select_c(i)$ = pozícia i-teho symbolu c od začiatku

Operácie rank a select

- *prístup*(i) = symbol c na i-tej pozícii
- $rank_c(i)$ = počet symbolov c od začiatku po i-tu pozíciu
- $select_c(i)$ = pozícia i-teho symbolu c od začiatku

indexy	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
B	1	1	0	1	1	0	1	1	1	0	0	0	0
	1	0	0	0	1	0	0	0	1	0	1	1	0