Filtrovanie

kuko

3.11.2020

Vybrané partie z dátových štruktúr

Filtrovanie

Problém

- máme množinu S
  - dát je veľa
  - prístup je pomalý (disk, sieť)
- vedeli by sme reprezentovať S približne s malou pamäťou?
  - povolíme falošne pozitívne výsledky

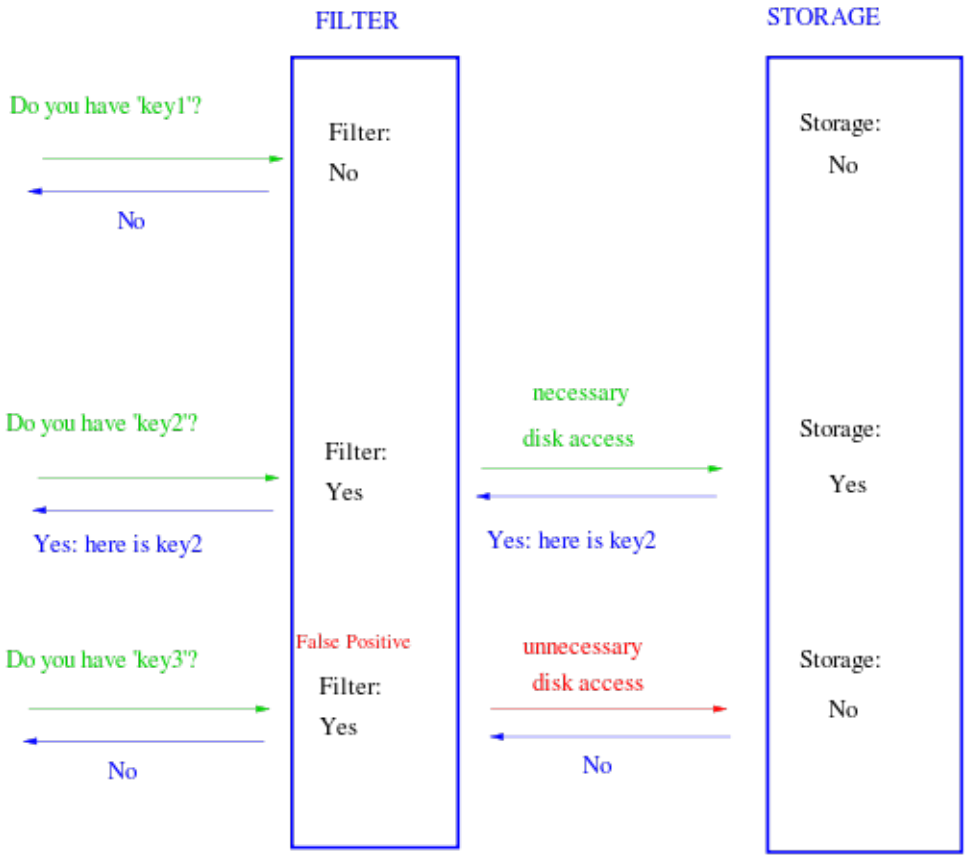
Problém

- máme množinu S
  - dát je veľa
  - prístup je pomalý (disk, sieť)
- vedeli by sme reprezentovať S približne s malou pamäťou?
  - povolíme falošne pozitívne výsledky

Problém

- máme množinu S
  - dát je veľa
  - prístup je pomalý (disk, sieť)
- vedeli by sme reprezentovať S približne s malou pamäťou?
  - povolíme falošne pozitívne výsledky

Filtrovanie



Akamai CDN

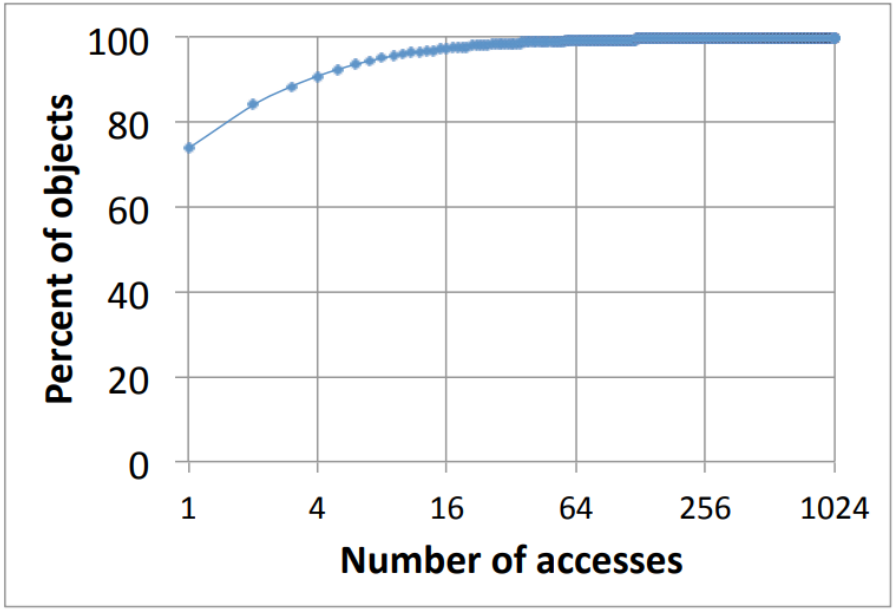
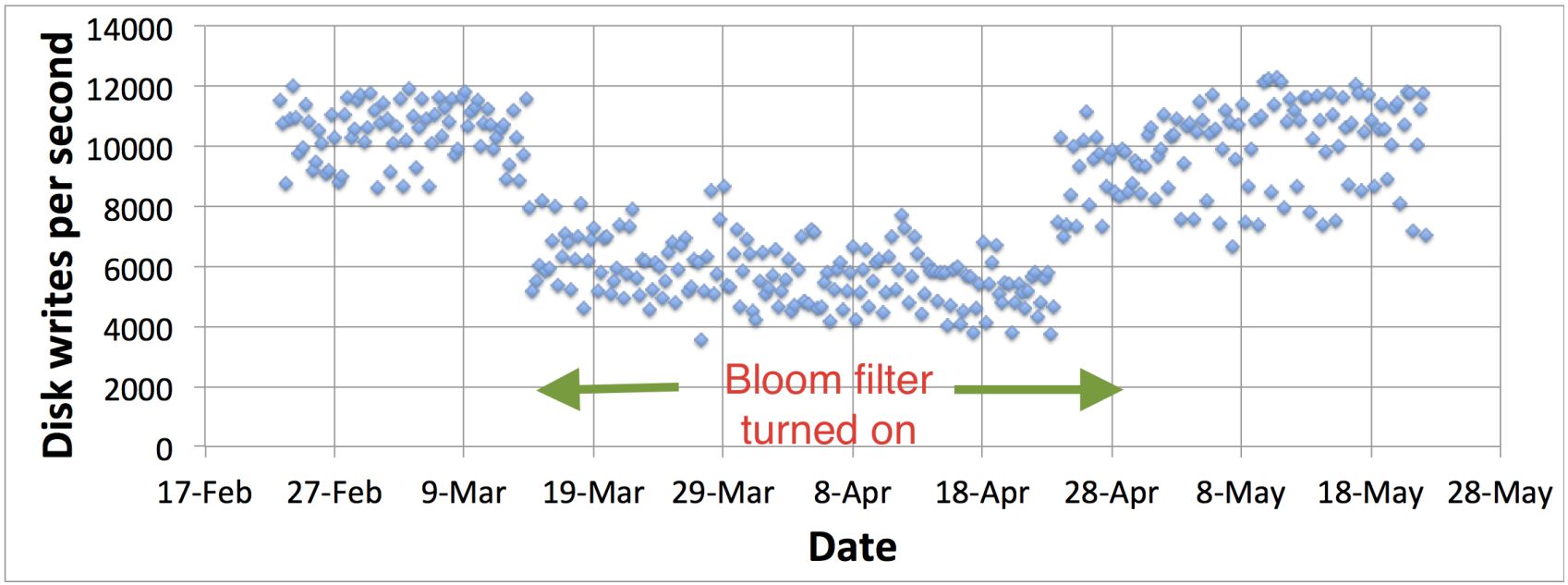


Figure 5: On a typical CDN server cluster serving web traffic over two days, 74% of the roughly 400 million objects in cache were accessed only once and 90% were accessed less than four times.

Akamai CDN


Disk writes per second
14000
12000
10000
8000
6000
4000
2000
0
17-Feb
27-Feb
9-Mar
19-Mar
29-Mar
8-Apr
18-Apr
28-Apr
8-May
18-May
28-May
Date
Bloom filter
turned on

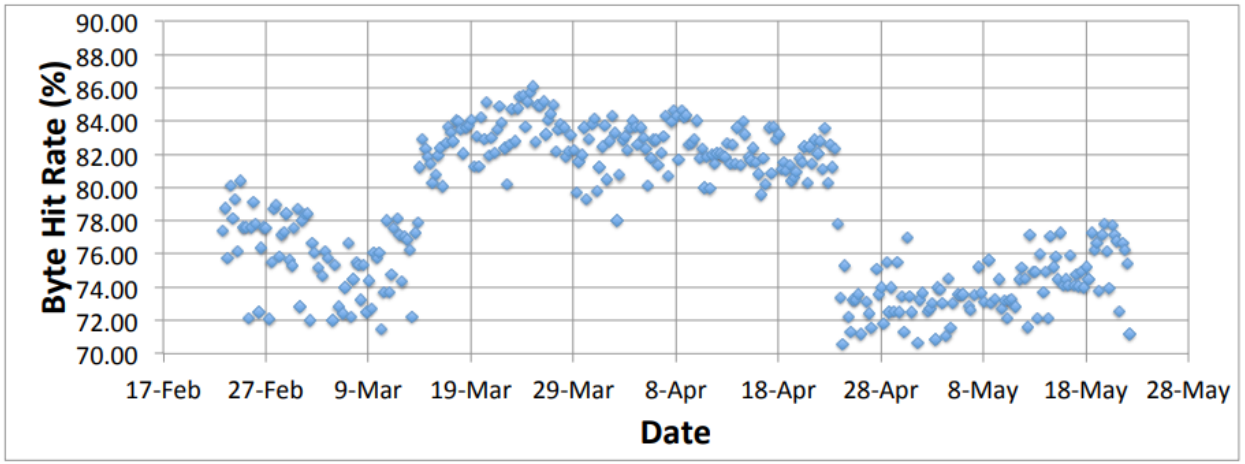


Figure 6: Byte hit rates increased when cache filtering was turned on between March 14th and April 24th because not caching objects that are accessed only once leaves more disk space to store more popular objects.

Motivácia

- milión aplikácií, napr.:
  - Akamai cacheuje iba web objekty, ktoré boli vyžiadané aspoň $2\times$
  - BigTable, Postgresql, HBase, Cassandra – odfiltrujú dotazy na neexistujúce riadky/stĺpce
  - Google Chrome – škodlivé URL
  - Bitcoin, synchronizácia peňaženiek
  - SPIN model checker
  - Medium
  - ...

Motivácia

- milión aplikácií, napr.:
  - Akamai cacheuje iba web objekty, ktoré boli vyžiadané aspoň $2\times$
  - BigTable, Postgresql, HBase, Cassandra – odfiltrujú dotazy na neexistujúce riadky/stĺpce
  - Google Chrome – škodlivé URL
  - Bitcoin, synchronizácia peňaženiek
  - SPIN model checker
  - Medium
  - ...

Motivácia

- milión aplikácií, napr.:
  - Akamai cacheuje iba web objekty, ktoré boli vyžiadané aspoň $2\times$
  - BigTable, Postgresql, HBase, Cassandra – odfiltrujú dotazy na neexistujúce riadky/stĺpce
  - Google Chrome – škodlivé URL
  - Bitcoin, synchronizácia peňaženiek
  - SPIN model checker
  - Medium
  - ...

Motivácia

- milión aplikácií, napr.:
  - Akamai cacheuje iba web objekty, ktoré boli vyžiadané aspoň $2\times$
  - BigTable, Postgresql, HBase, Cassandra – odfiltrujú dotazy na neexistujúce riadky/stĺpce
  - Google Chrome – škodlivé URL
  - Bitcoin, synchronizácia peňaženiek
  - SPIN model checker
  - Medium
  - ...

Motivácia

- milión aplikácií, napr.:
  - Akamai cacheuje iba web objekty, ktoré boli vyžiadané aspoň $2\times$
  - BigTable, Postgresql, HBase, Cassandra – odfiltrujú dotazy na neexistujúce riadky/stĺpce
  - Google Chrome – škodlivé URL
  - Bitcoin, synchronizácia peňaženiek
  - SPIN model checker
  - Medium
  - ...

Motivácia

- milión aplikácií, napr.:
  - Akamai cacheuje iba web objekty, ktoré boli vyžiadané aspoň $2\times$
  - BigTable, Postgresql, HBase, Cassandra – odfiltrujú dotazy na neexistujúce riadky/stĺpce
  - Google Chrome – škodlivé URL
  - Bitcoin, synchronizácia peňaženiek
  - SPIN model checker
  - Medium
  - ...

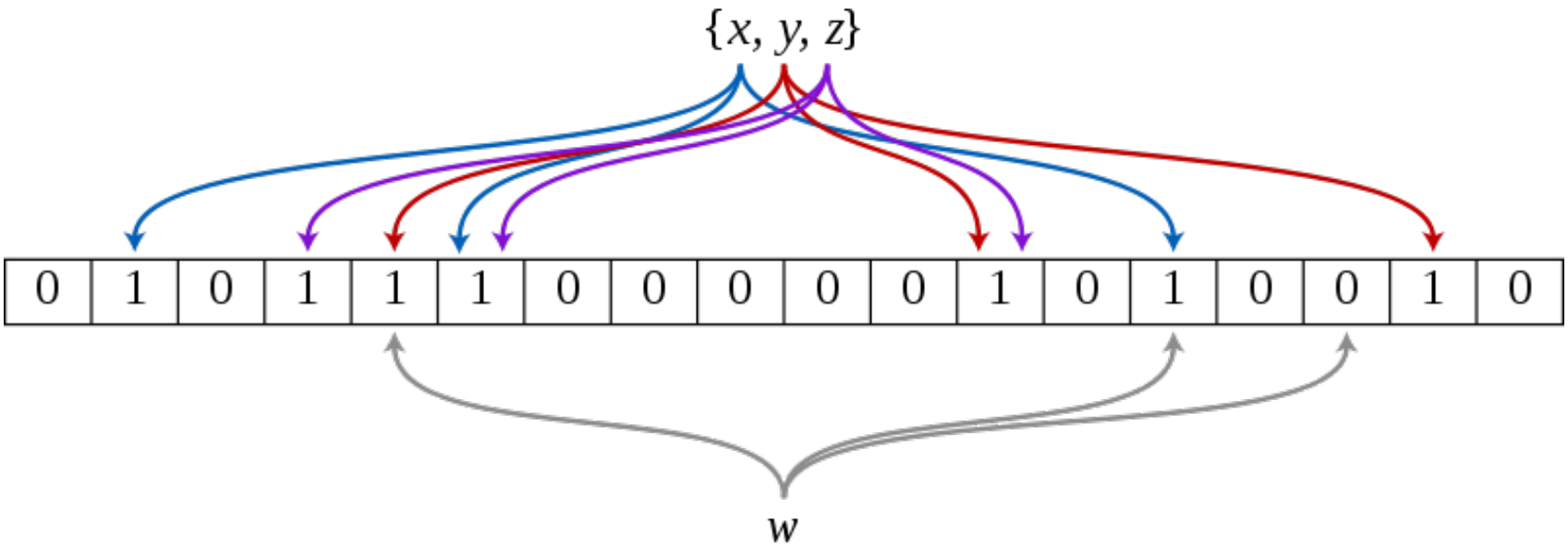
Motivácia

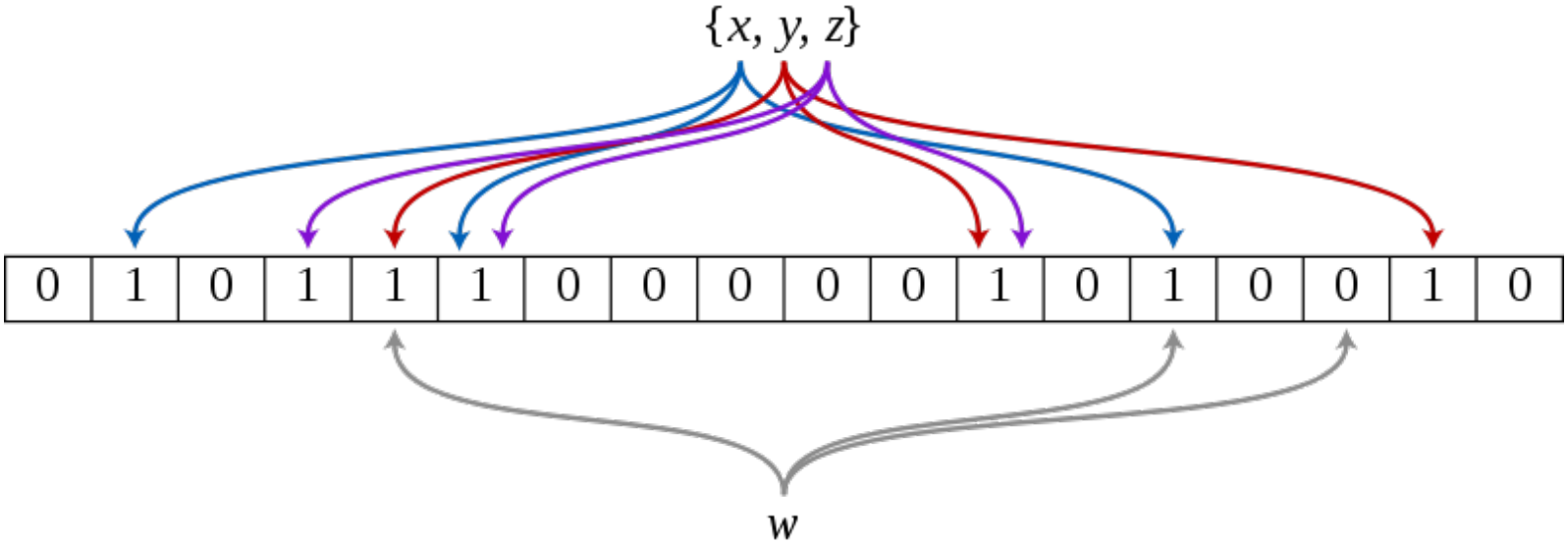
- milión aplikácií, napr.:
  - Akamai cacheuje iba web objekty, ktoré boli vyžiadané aspoň $2\times$
  - BigTable, Postgresql, HBase, Cassandra – odfiltrujú dotazy na neexistujúce riadky/stĺpce
  - Google Chrome – škodlivé URL
  - Bitcoin, synchronizácia peňaženiek
  - SPIN model checker
  - Medium
  - ...

Motivácia

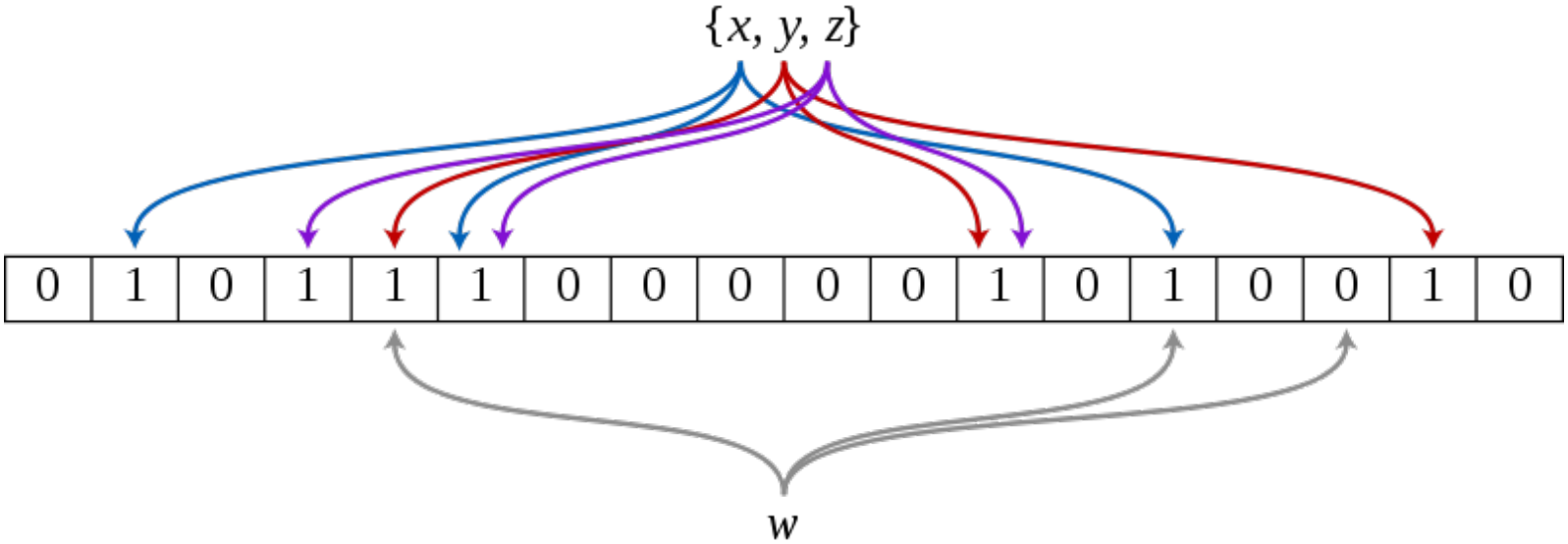
- milión aplikácií, napr.:
  - Akamai cacheuje iba web objekty, ktoré boli vyžiadané aspoň 2×
  - BigTable, Postgresql, HBase, Cassandra – odfiltrujú dotazy na neexistujúce riadky/stĺpce
  - Google Chrome – škodlivé URL
  - Bitcoin, synchronizácia peňaženiek
  - SPIN model checker
  - Medium
  - ...

Bloomove filtre

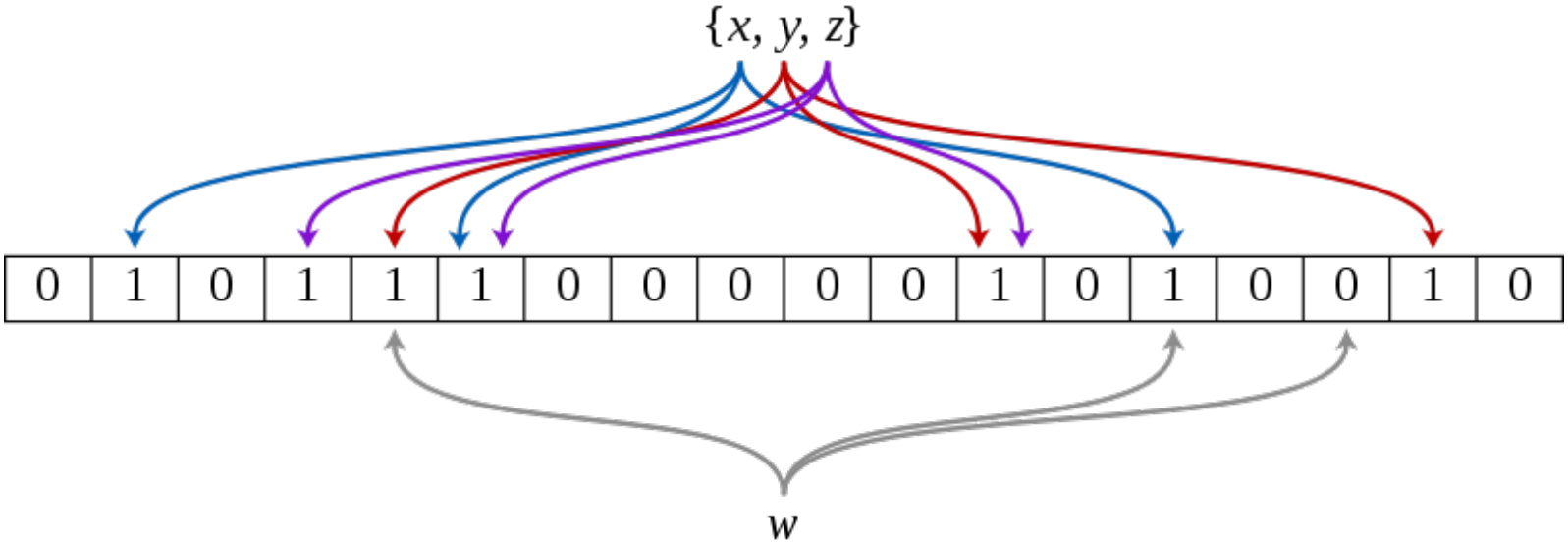

{x, y, z}
0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0
w



- $S = \{x_1, \ldots, x_n\}$
- m bitov
- k nezávislých hešovacích fn. $h_1, \ldots, h_k : \mathscr{U} \to [m]$
- insert(x): nastav bity $B[h_i(x)] = 1$ $(\forall i)$
- query(y): skontroluj, či $\forall i : B[h_i(y)] = 1$

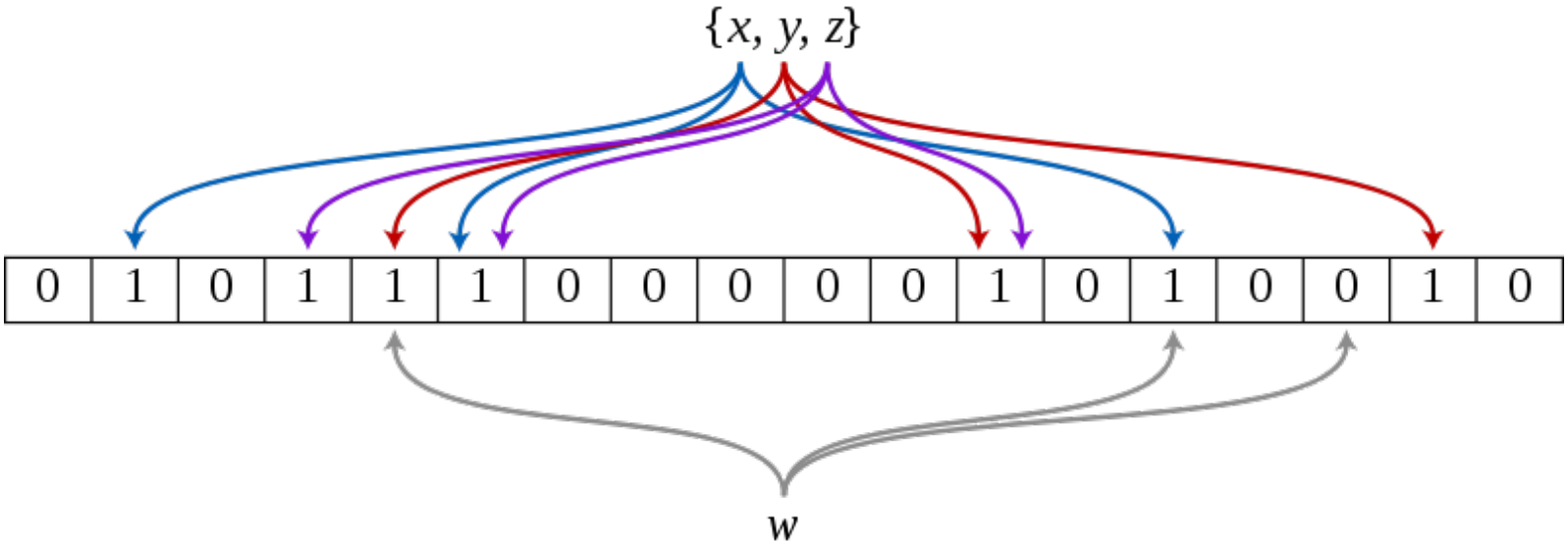


- $S = \{x_1, \ldots, x_n\}$
- m bitov
- k nezávislých hešovacích fn. $h_1, \ldots, h_k : \mathscr{U} \to [m]$
- insert(x): nastav bity $B[h_i(x)] = 1$ $(\forall i)$
- query(y): skontroluj, či $\forall i : B[h_i(y)] = 1$



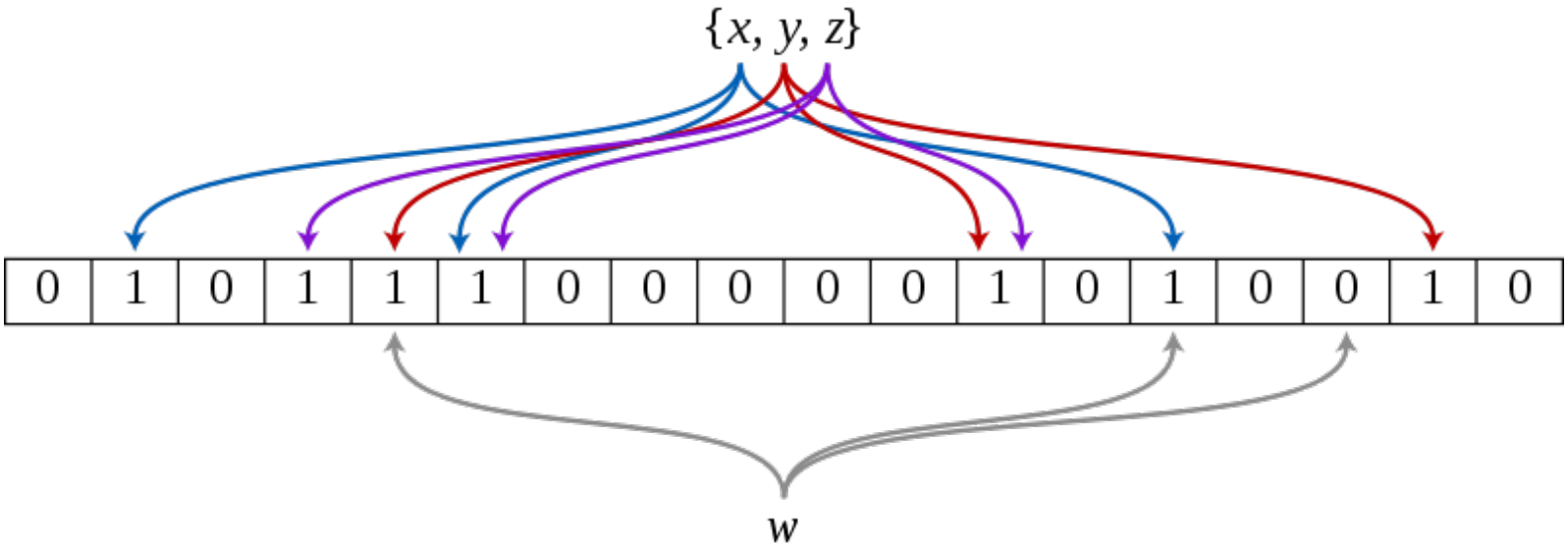
- $S = \{x_1, \ldots, x_n\}$
- m bitov
- k nezávislých hešovacích fn. $h_1, \ldots, h_k : \mathscr{U} \to [m]$
- insert(x): nastav bity $B[h_i(x)] = 1$ $(\forall i)$
- query(y): skontroluj, či $\forall i : B[h_i(y)] = 1$

Analýza



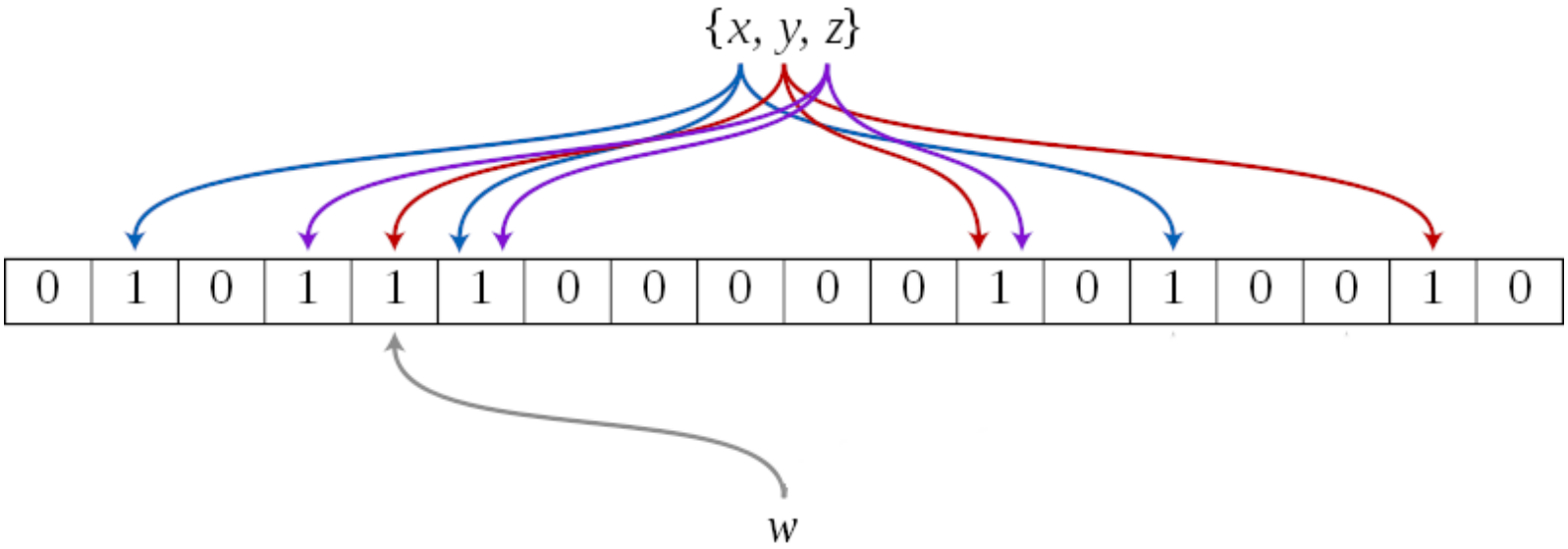
- predpokladajme úplne náhodné h_i
- aká je $\delta = \Pr[FP]$?
- pre dané m, n, aké k máme zvoliť pre min δ

Analýza



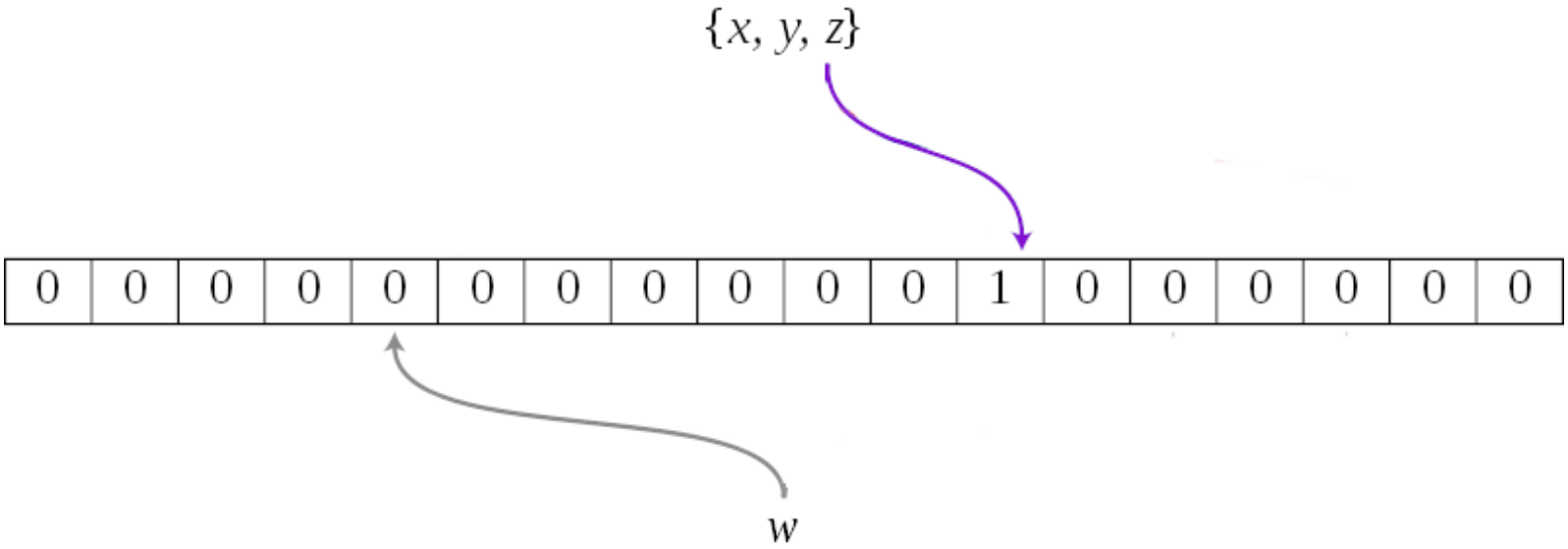
aká je pp., že k náhodných políčok budú 1, ak predtým zahešujeme n prvkov k-krát?

Analýza



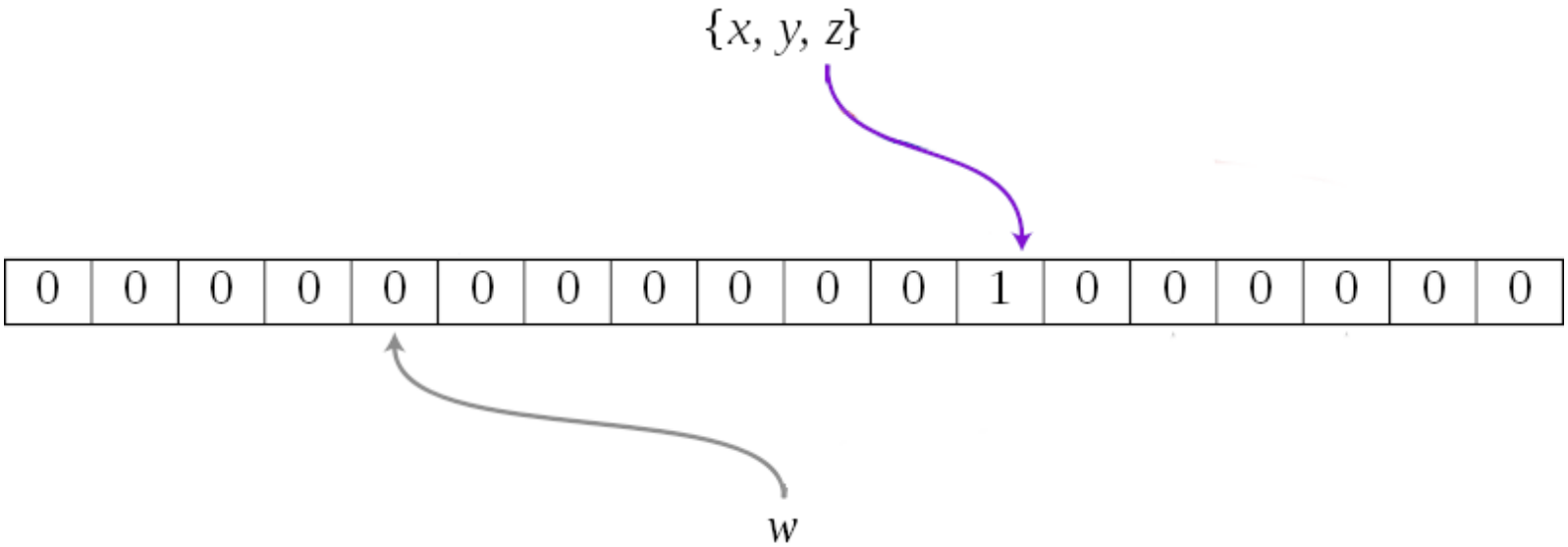
aká je pp., že 1 náhodné políčko bude 1, ak predtým zahešujeme n prvkov k-krát?

Analýza



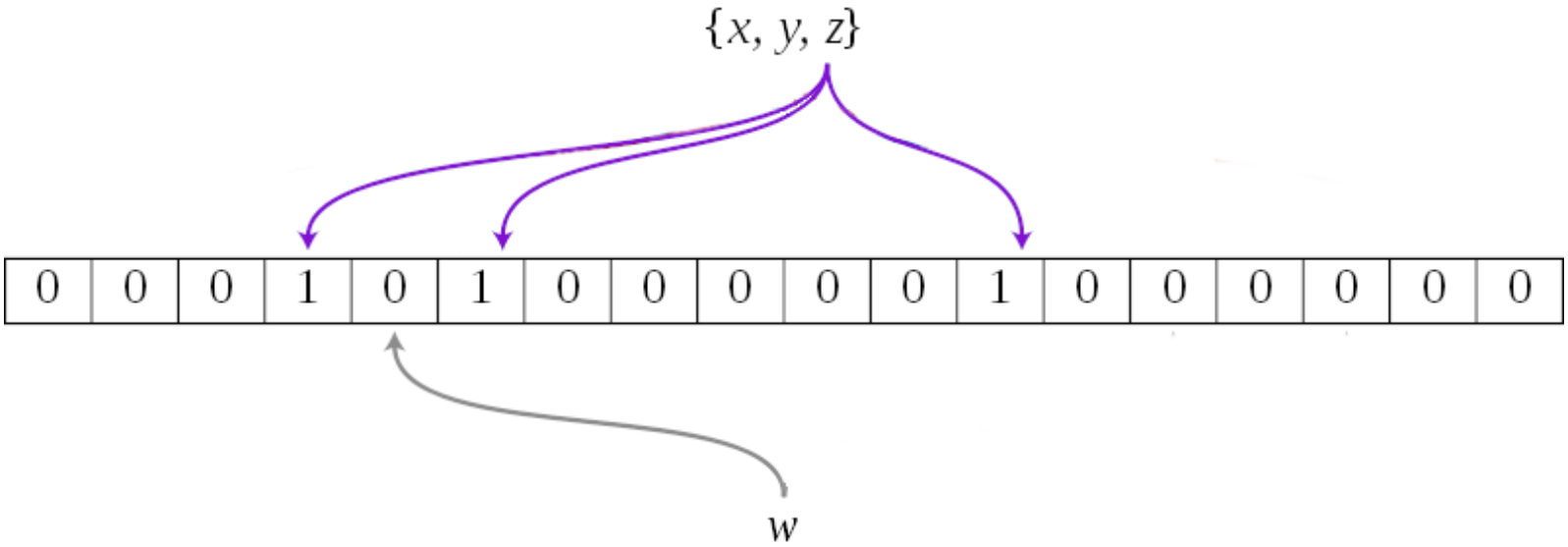
aká je pp., že 1 náhodné políčko bude 1, ak predtým zahešujeme 1 prvok jedenkrát?

Analýza

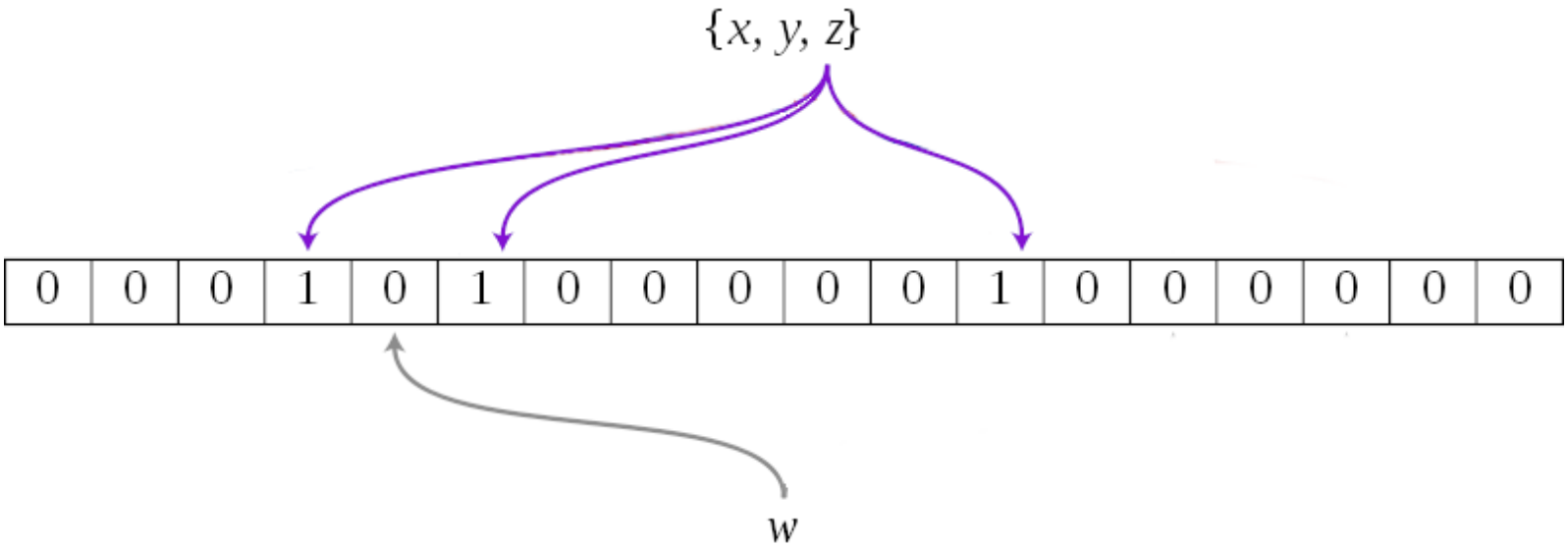


- $\Pr[B[i] = 1] = 1/m$
- $\Pr[B[i] = 0] = 1 - 1/m$

Analýza

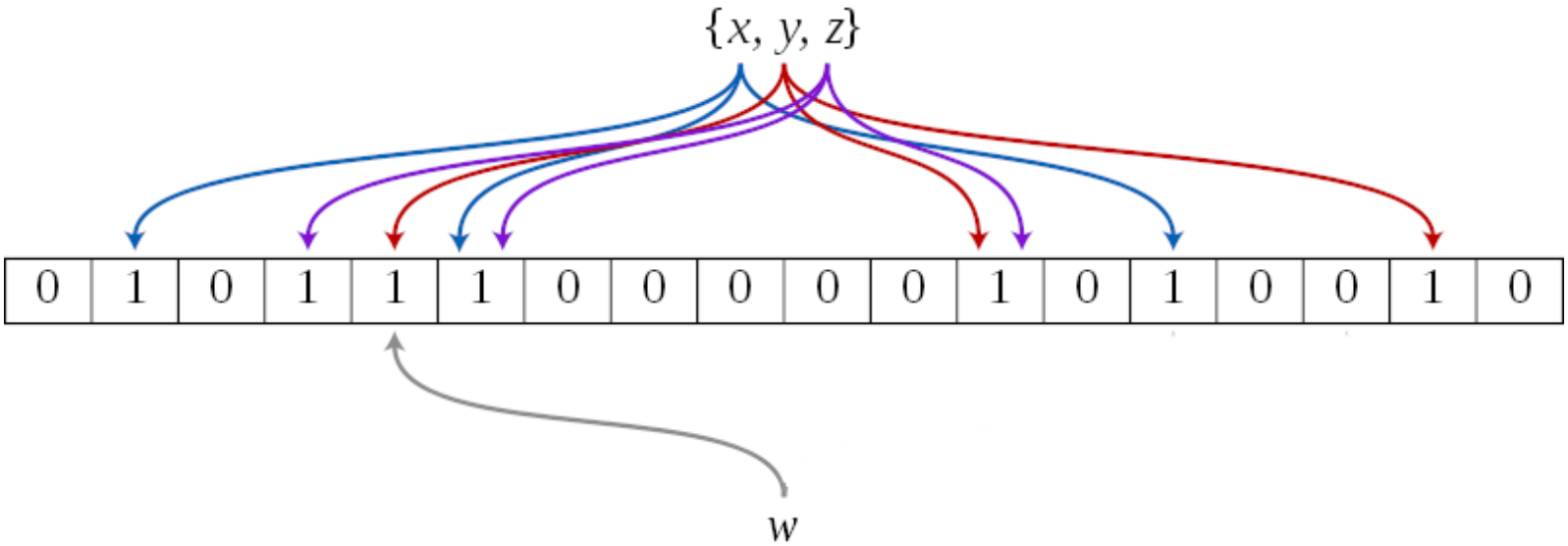


Analýza



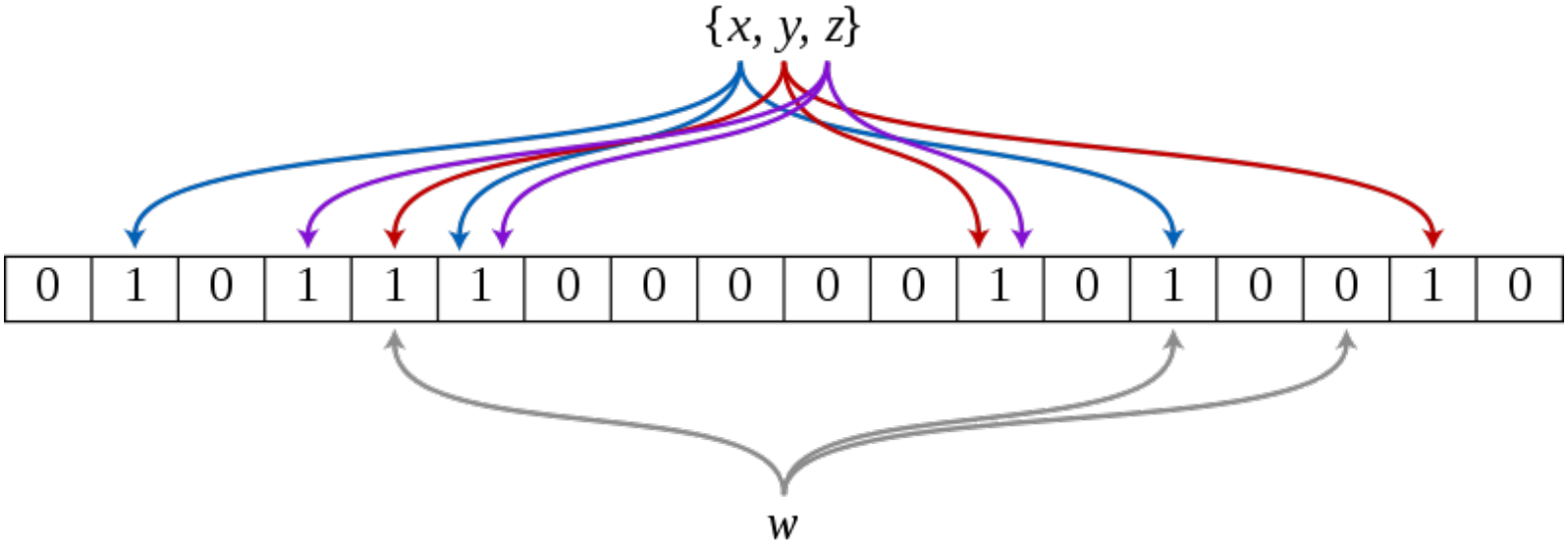
- $\Pr[B[i] = 0] = (1 - 1/m)^k$

Analýza



- aká je pp., že 1 náhodné políčko bude 1, ak predtým označíme $k \times n$ náhodných políčok?
- $p = \Pr[B[i] = 0] = (1 - 1/m)^{nk}$

Analýza



- aká je pp., že k náhodných políčok budú 1, ak predtým označíme $k \times n$ náhodných políčok?
- $\delta = \Pr[FP] = (1 - p)^k$

Analýza

- $p = \Pr[B_i = 0] = (1 - 1/m)^{kn} = \left((1 - 1/m)^m\right)^{kn/m} \approx e^{-kn/m}$
- $\delta = \Pr[FP] = (1 - p)^k = \exp(k \ln(1 - p)) = \exp(k \ln(1 - e^{-kn/m}))$
- ak zvolíme fixné m/n =#bitov na jeden prvok

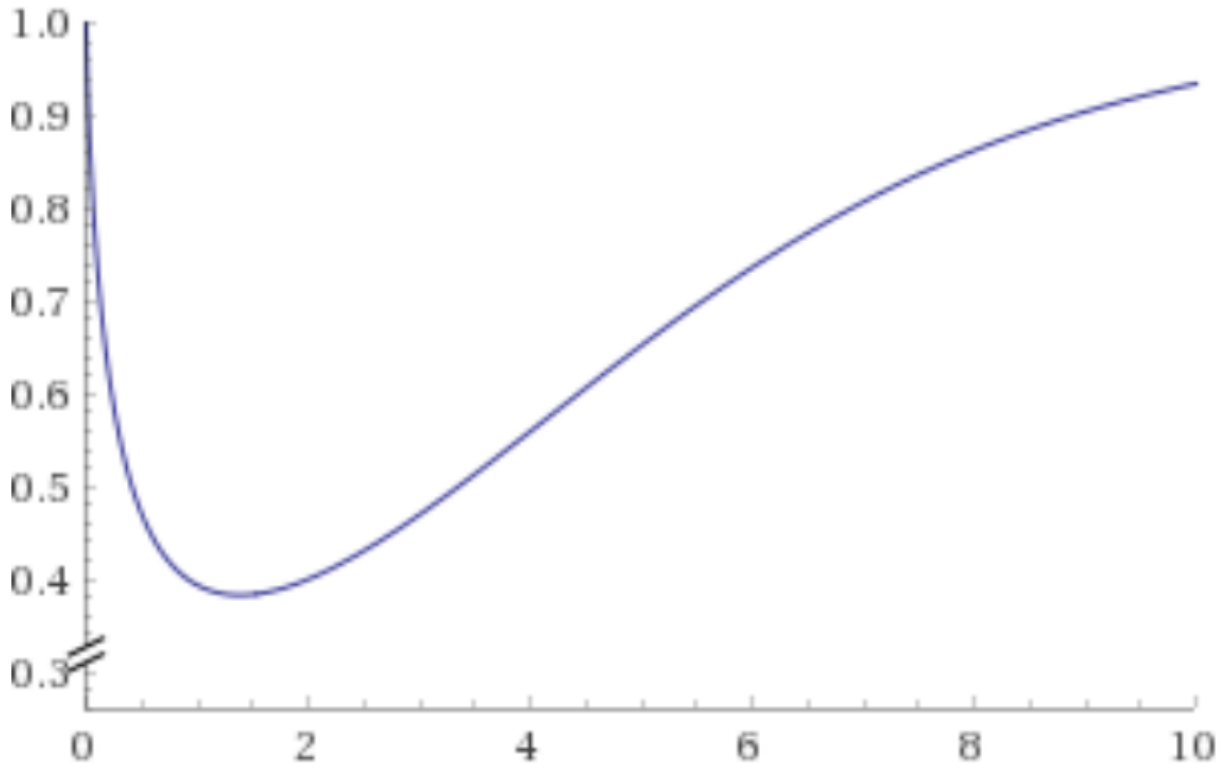
Analýza

- $p = \Pr[B_i = 0] = (1 - 1/m)^{kn} = \left((1 - 1/m)^m\right)^{kn/m} \approx e^{-kn/m}$
- $\delta = \Pr[FP] = (1 - p)^k = \exp(k \ln(1 - p)) = \exp(k \ln(1 - e^{-kn/m}))$
- ak zvolíme fixné m/n =#bitov na jeden prvok

Analýza

- $p = \Pr[B_i = 0] = (1 - 1/m)^{kn} = \left((1 - 1/m)^m\right)^{kn/m} \approx e^{-kn/m}$
- $\delta = \Pr[FP] = (1 - p)^k = \exp(k \ln(1 - p)) = \exp(k \ln(1 - e^{-kn/m}))$
- ak zvolíme fixné $m/n =$#bitov na jeden prvok

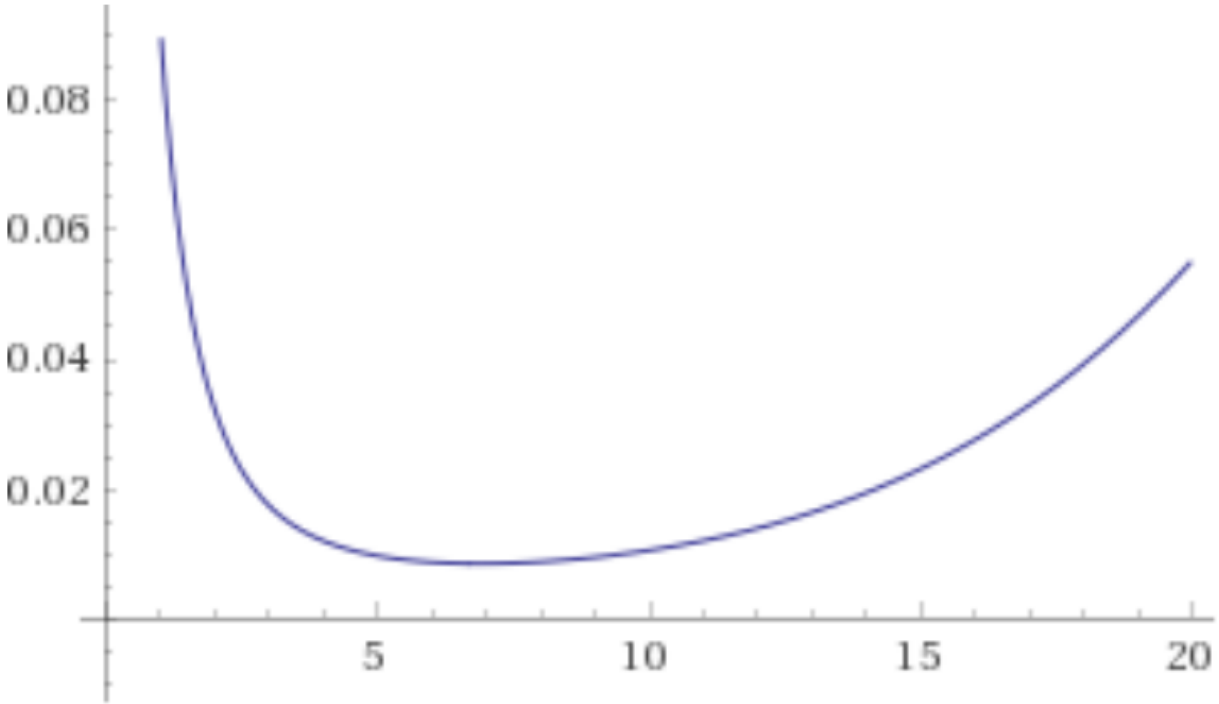
Analýza



pre $m/n = 2$: $y = \exp(x \ln(1 - e^{-x/2}))$

Analýza

Plot:



pre $m/n = 10$: $y = \exp(x \ln(1 - e^{-x/10}))$

Analýza

Input interpretation:

minimize

function	$\exp(x \log(1 - e^{-x/10}))$
domain	$0 \le x \le 20$

$\log(x)$ is the natural logarithm

Enlarge | Customize | Plain Text

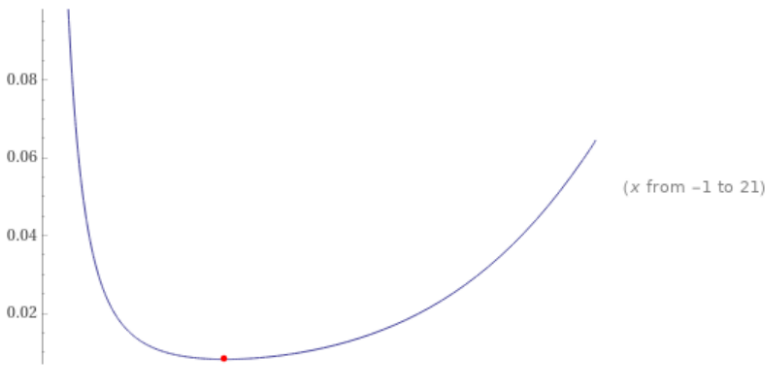
Global minima:

(no global minima found)

Local minimum:

$\min\{\exp(x \log(1 - e^{-x/10})) \mid 0 \le x \le 20\} \approx 0.00819255$ at $x \approx 6.93147$

Plot:



Analýza

- $p = e^{-kn/m} \longrightarrow k = -\frac{m}{n} \ln p$
- $\delta = \exp(k \ln(1-p)) =$

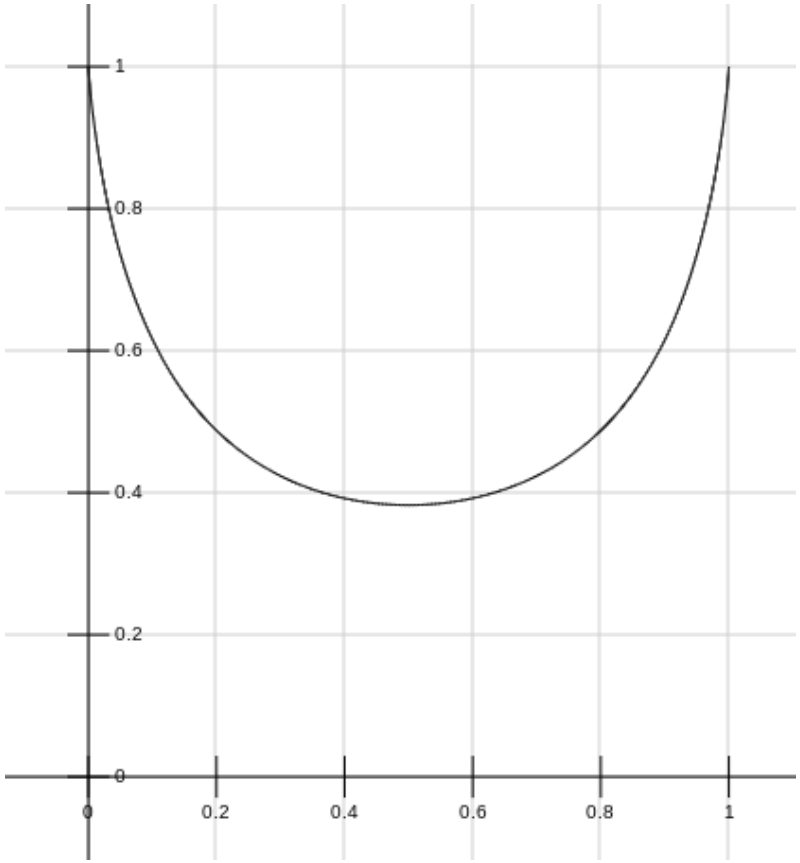
$$\exp\left(-\frac{m}{n} \cdot \ln p \cdot \ln(1-p)\right)$$

Analýza

- $p = e^{-kn/m} \longrightarrow k = -\frac{m}{n} \ln p$
- $\delta = \exp(k \ln(1-p)) =$

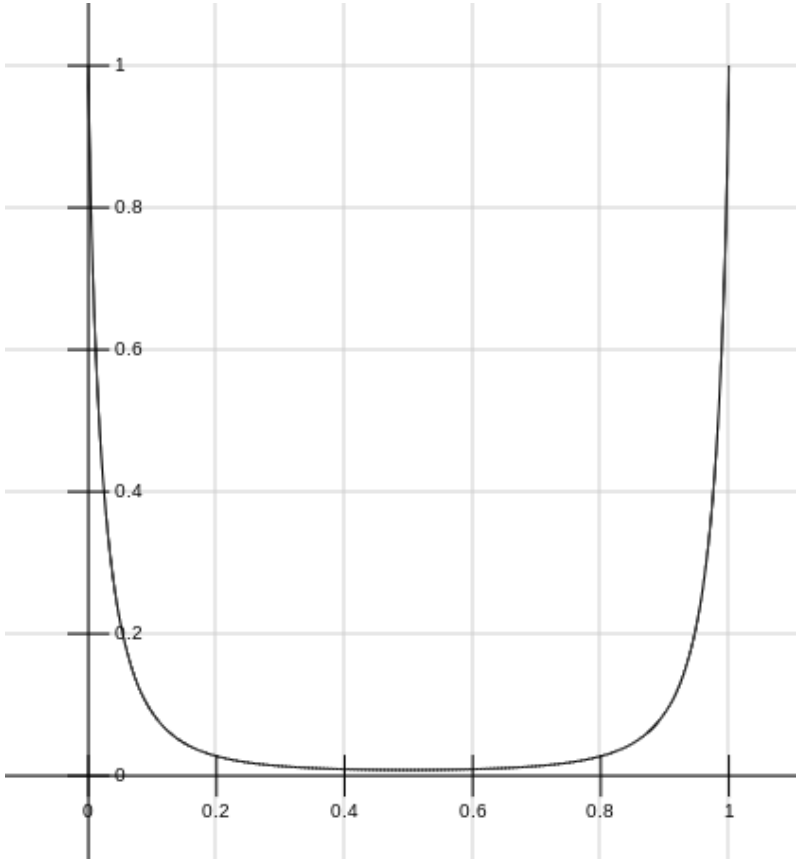
$$\exp\left(-\frac{m}{n} \cdot \ln p \cdot \ln(1-p)\right)$$

Analýza



pre $m/n = 2$: $y = \exp(-2 \ln x \ln(1-x))$

Analýza



pre $m/n = 10$: $y = \exp(-10 \ln x \ln(1-x))$

Analýza

- opt $p = 1/2 = e^{-kn/m}$, $\longrightarrow$ $k = \ln 2 \cdot (m/n) \approx 0.693(m/n)$
- pre takto zvolené k je $\delta = 1/2^k \approx 0.619^{m/n}$
- resp. $m/n \approx 1.445 \lg(1/\delta)$
- napr. ak chceme $\delta = 1\%$, zvolíme $m/n \approx 9.6$, $k = 6$ alebo 7

Analýza

- opt $p = 1/2 = e^{-kn/m}$, $\longrightarrow$ $k = \ln 2 \cdot (m/n) \approx 0.693(m/n)$
- pre takto zvolené k je $\delta = 1/2^k \approx 0.619^{m/n}$
- resp. $m/n \approx 1.445 \lg(1/\delta)$
- napr. ak chceme $\delta = 1\%$, zvolíme $m/n \approx 9.6$, $k = 6$ alebo 7

Analýza

- opt $p = 1/2 = e^{-kn/m}$, $\longrightarrow$ $k = \ln 2 \cdot (m/n) \approx 0.693(m/n)$
- pre takto zvolené k je $\delta = 1/2^k \approx 0.619^{m/n}$
- resp. $m/n \approx 1.445 \lg(1/\delta)$
- napr. ak chceme $\delta = 1\%$, zvolíme $m/n \approx 9.6$, $k = 6$ alebo 7

Analýza

- opt $p = 1/2 = e^{-kn/m}$, $\longrightarrow$ $k = \ln 2 \cdot (m/n) \approx 0.693(m/n)$
- pre takto zvolené k je $\delta = 1/2^k \approx 0.619^{m/n}$
- resp. $m/n \approx 1.445 \lg(1/\delta)$
- napr. ak chceme $\delta = 1\%$, zvolíme $m/n \approx 9.6$, $k = 6$ alebo 7

Analýza

% FP		*k = #hešovacích funkcií*									
		1	2	3	4	5	6	7	8	9	10
m/n = #bitov na jeden prvok	1	**63.21**	74.76	85.80	92.87	96.68	98.52	99.36	99.73	99.89	99.95
	2	**39.35**	39.96	46.89	55.90	65.16	73.61	80.68	86.25	90.43	93.46
	3	28.35	**23.68**	25.26	29.41	35.11	41.79	48.97	56.21	63.15	69.54
	4	22.12	**15.48**	**14.69**	15.97	18.49	21.98	26.28	31.25	36.70	42.46
	5	18.13	10.87	**9.18**	9.20	10.09	11.64	13.78	16.46	19.67	23.36
	6	15.35	8.04	**6.09**	**5.61**	**5.78**	6.38	7.34	8.65	10.31	12.33
	7	13.31	6.18	**4.23**	**3.59**	**3.47**	3.64	4.03	4.63	5.44	6.46
	8	11.75	4.89	3.06	**2.40**	**2.17**	**2.16**	2.29	2.55	2.92	3.42
	9	10.52	3.97	2.28	**1.66**	**1.41**	**1.33**	1.35	1.45	1.61	1.84
	10	9.52	3.29	1.74	**1.18**	**0.94**	**0.84**	**0.82**	0.85	0.91	1.02
	11	8.69	2.76	1.36	0.86	**0.65**	**0.55**	**0.51**	0.51	0.53	0.58
	12	8.00	2.36	1.08	0.65	0.46	**0.37**	**0.33**	**0.31**	0.32	0.33
	13	7.40	2.03	0.88	0.49	0.33	0.26	**0.22**	**0.20**	**0.19**	0.20
	14	6.89	1.77	0.72	0.38	0.24	0.18	**0.15**	**0.13**	**0.12**	0.12
	15	6.45	1.56	0.60	0.30	0.18	0.13	0.10	**0.09**	**0.08**	**0.07**
	16	6.06	1.38	0.50	0.24	0.14	0.09	0.07	**0.06**	**0.05**	**0.05**

Zhrnutie

Existuje DŠ, ktorej stačí $\approx 1.445 \lg(1/\delta)$ bitu na prvok

- $p = (1 - 1/m)^{kn} \approx e^{-kn/m}$, optimálne $p \approx 1/2$
- FPR $\delta = (1-p)^k$
- napríklad stačí 10 bitov/prvok a 5 heš. fn. pre FPR$< 1\%$

Dolný odhad

- pre BF nám vyšlo $m/n \approx 1.445 \lg(1/\delta)$
- v skutočnosti zhruba toľko bitov potrebujeme:
- porovnajme počet n-prvkových množín vs. počet m-bitových stringov

$$\underbrace{2^m}_{\text{počet } m\text{-bit. stringov}} \underbrace{\binom{n+\delta(\mathscr{U}-n)}{n}}_{\text{každý reprezentuje najviac toľkoto množín}} \geq \underbrace{\binom{\mathscr{U}}{n}}_{\text{počet } n\text{-prvkových množín}}$$

$$m \geq \lg \frac{\binom{\mathscr{U}}{n}}{\binom{n+\delta(\mathscr{U}-n)}{n}} \approx \lg \frac{\binom{\mathscr{U}}{n}}{\binom{\delta\mathscr{U}}{n}} \geq \lg \delta^{-n} = n \lg(1/\delta)$$

Dolný odhad

- pre BF nám vyšlo $m/n \approx 1.445 \lg(1/\delta)$
- v skutočnosti zhruba toľko bitov potrebujeme:
- porovnajme počet n-prvkových množín vs. počet m-bitových stringov

$$\underbrace{2^m}_{\text{počet } m\text{-bit. stringov}} \underbrace{\binom{n+\delta(\mathscr{U}-n)}{n}}_{\text{každý reprezentuje najviac toľkoto množín}} \geq \underbrace{\binom{\mathscr{U}}{n}}_{\text{počet } n\text{-prvkových množín}}$$

$$m \geq \lg \frac{\binom{\mathscr{U}}{n}}{\binom{n+\delta(\mathscr{U}-n)}{n}} \approx \lg \frac{\binom{\mathscr{U}}{n}}{\binom{\delta\mathscr{U}}{n}} \geq \lg \delta^{-n} = n \lg(1/\delta)$$

Dolný odhad

- pre BF nám vyšlo $m/n \approx 1.445 \lg(1/\delta)$
- v skutočnosti zhruba toľko bitov potrebujeme:
- porovnajme počet n-prvkových množín vs. počet m-bitových stringov

$$\underbrace{2^m}_{\text{počet } m\text{-bit. stringov}} \qquad \underbrace{\binom{n+\delta(\mathscr{U}-n)}{n}}_{\text{každý reprezentuje najviac toľkoto množín}} \qquad \geq \qquad \underbrace{\binom{\mathscr{U}}{n}}_{\text{počet } n\text{-prvkových množín}}$$

$$m \geq \lg \frac{\binom{\mathscr{U}}{n}}{\binom{n+\delta(\mathscr{U}-n)}{n}} \approx \lg \frac{\binom{\mathscr{U}}{n}}{\binom{\delta\mathscr{U}}{n}} \geq \lg \delta^{-n} = n \lg(1/\delta)$$

Dolný odhad

- pre BF nám vyšlo $m/n \approx 1.445 \lg(1/\delta)$
- v skutočnosti zhruba toľko bitov potrebujeme:
- porovnajme počet n-prvkových množín vs. počet m-bitových stringov

$$\underbrace{2^m}_{\text{počet } m\text{-bit. stringov}} \qquad \underbrace{\binom{n+\delta(\mathscr{U}-n)}{n}}_{\text{každý reprezentuje najviac toľkoto množín}} \qquad \geq \qquad \underbrace{\binom{\mathscr{U}}{n}}_{\text{počet } n\text{-prvkových množín}}$$

$$m \geq \lg \frac{\binom{\mathscr{U}}{n}}{\binom{n+\delta(\mathscr{U}-n)}{n}} \approx \lg \frac{\binom{\mathscr{U}}{n}}{\binom{\delta\mathscr{U}}{n}} \geq \lg \delta^{-n} = n \lg(1/\delta)$$

Odhad počtu prvkov

- pre daný Bloom filter – vedeli by sme odhadnúť počet prvkov v ňom?

Odhad počtu prvkov

- pre daný Bloom filter – vedeli by sme odhadnúť počet prvkov v ňom?
- $\Pr[B_i = 0] = p = e^{-kn/m}$
- teda $E[\#_0] = me^{-kn/m}$
- odtiaľ odhad počtu prvkov: $n^* = m/k \cdot \ln(m/\#_0)$

Odhad počtu prvkov

- pre daný Bloom filter – vedeli by sme odhadnúť počet prvkov v ňom?
- $\Pr[B_i = 0] = p = e^{-kn/m}$
- teda $E[\#_0] = me^{-kn/m}$
- odtiaľ odhad počtu prvkov: $n^* = m/k \cdot \ln(m/\#_0)$

Odhad počtu prvkov

- pre daný Bloom filter – vedeli by sme odhadnúť počet prvkov v ňom?
- $\Pr[B_i = 0] = p = e^{-kn/m}$
- teda $E[\#_0] = me^{-kn/m}$
- odtiaľ odhad počtu prvkov: $n^* = m/k \cdot \ln(m/\#_0)$

Nevýhody

- BF sa nedá zväčšiť – vybudovaný pre konkrétne n, δ
- vedeli by sme pridať operáciu delete?
- nevýhodné pre cache (k náhodných prístupov)

Podielové filtre

- myšlienka: $h : \mathscr{U} \to [2^p]$, budeme exaktne reprezentovať multimnožinu $h(S)$

$$\{x_1, x_2, \ldots, x_n\} \quad \mapsto^h \quad \{h(x_1), h(x_2), \ldots, h(x_n)\}$$

- myšlienka: $h : \mathscr{U} \to [2^p]$, budeme exaktne reprezentovať multimnožinu $h(S)$

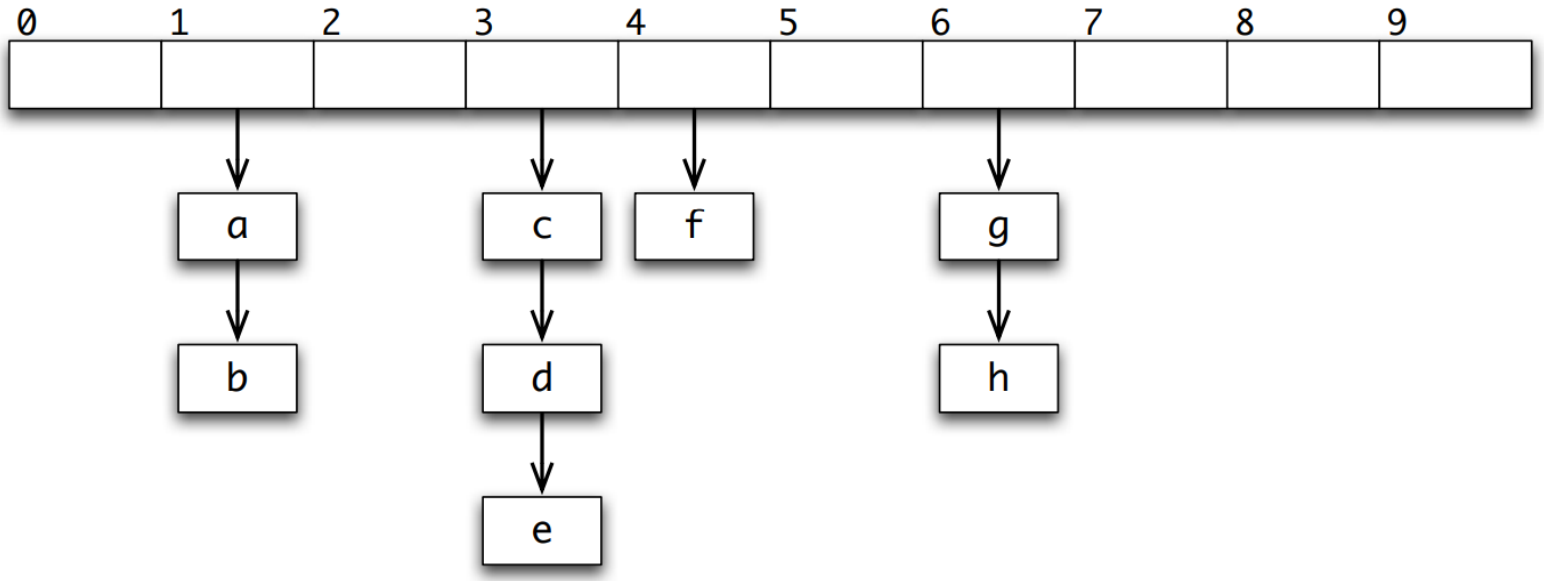
$$\{x_1, x_2, \ldots, x_n\} \quad \mapsto^h \quad \{h(x_1), h(x_2), \ldots, h(x_n)\}$$

FRP je pp. kolízie $h(x) = h(x')$, pričom $h : \mathscr{U} \to [2^p]$

$$\delta = 1 - (1 - 1/2^p)^n \approx 1 - e^{-n/2^p} \leq n/2^p$$

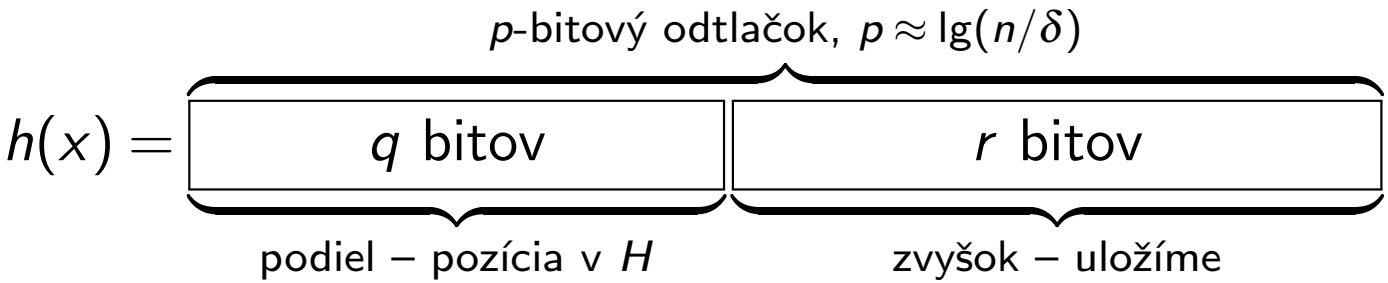
$\implies p \approx \lg(n/\delta)$ pre FPR δ

- zoznam $h(x_1), h(x_2), \ldots, h(x_n)$
  - hľadanie $O(n)$ ☹
  - pamäť $\lg(n/\delta) = \lg n + \lg(1/\delta)$ bitov/prvok ☹

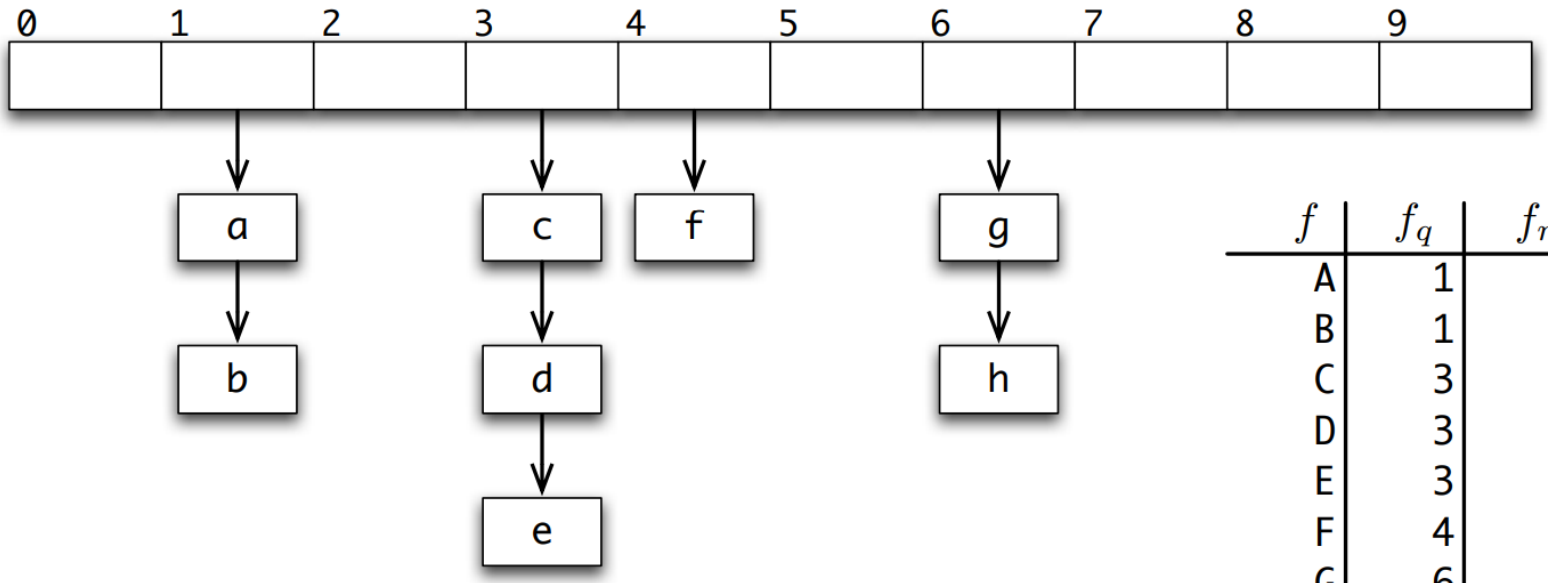


- hešovanie
  - hľadanie $O(1)$ ☺
  - pamäť $64 + \lg(n/\delta)$ bitov/prvok ☹

Podielové hešovanie

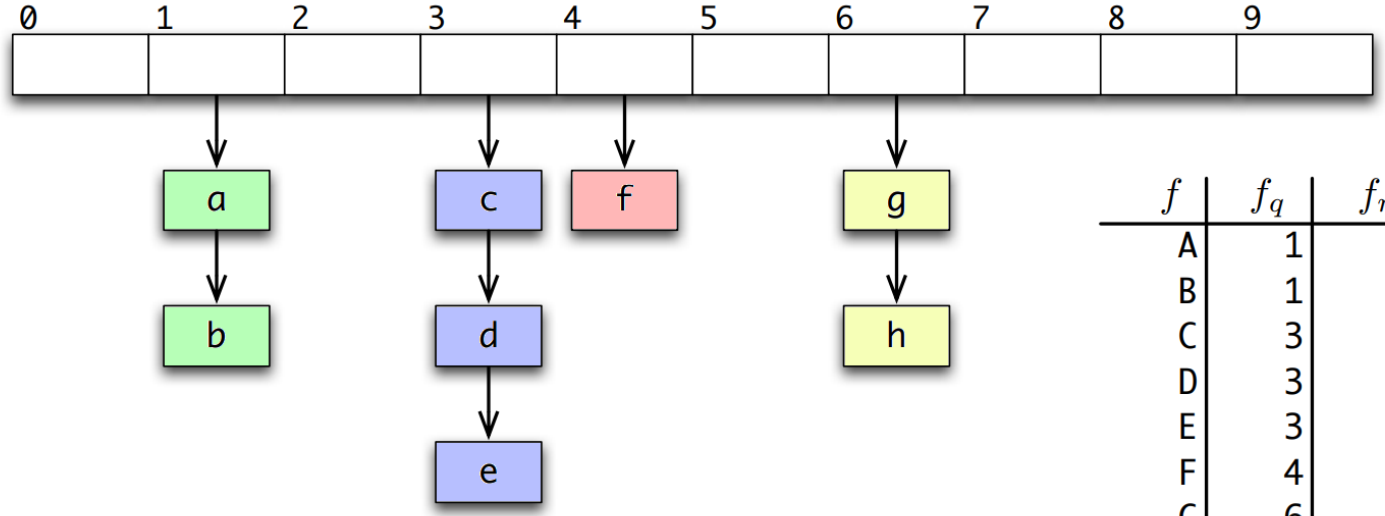


Podielové hešovanie

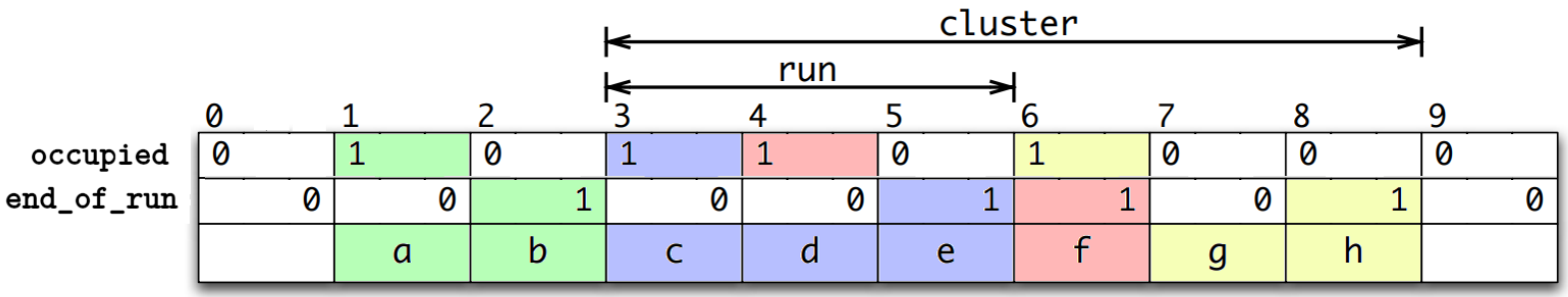


f	f_q	f_r
A	1	a
B	1	b
C	3	c
D	3	d
E	3	e
F	4	f
G	6	g
H	6	h

- hešovanie so zreťazením; $2^q = \Theta(n)$ políčok, r-bitové zvyšky uložíme
  - pamäť $64 + \lg(1/\delta)$ bitov/prvok ☹
  - vieme zrekonštruovať celý heš $\Longrightarrow$ vieme mergovať, vieme postupne zväčšovať heštabuľku



f	f_q	f_r
A	1	a
B	1	b
C	3	c
D	3	d
E	3	e
F	4	f
G	6	g
H	6	h

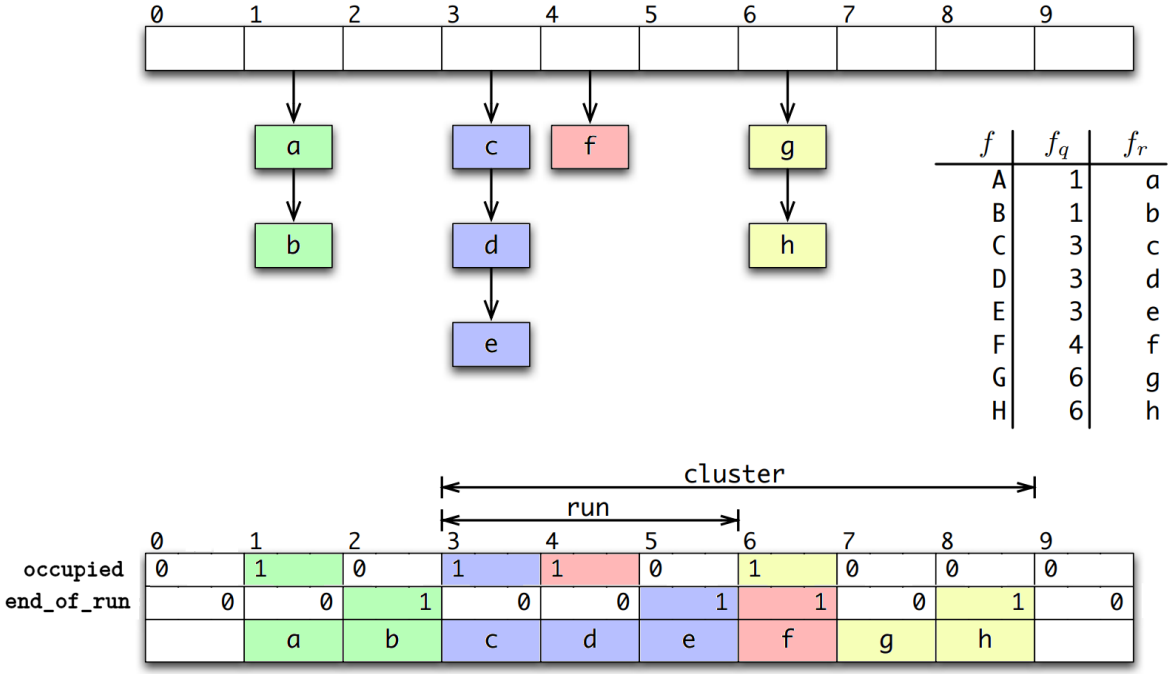


- otvorené adresovanie s lineárnym sondovaním
  - bitvektory pre „začiatky“ a konce
  - vieme zrekonštruovať všetky heše, mergovať, dynamicky zväčšovať

Algorithm 1 Algorithm for determining whether x may have been inserted into a simple rank-and-select-based quotient filter.

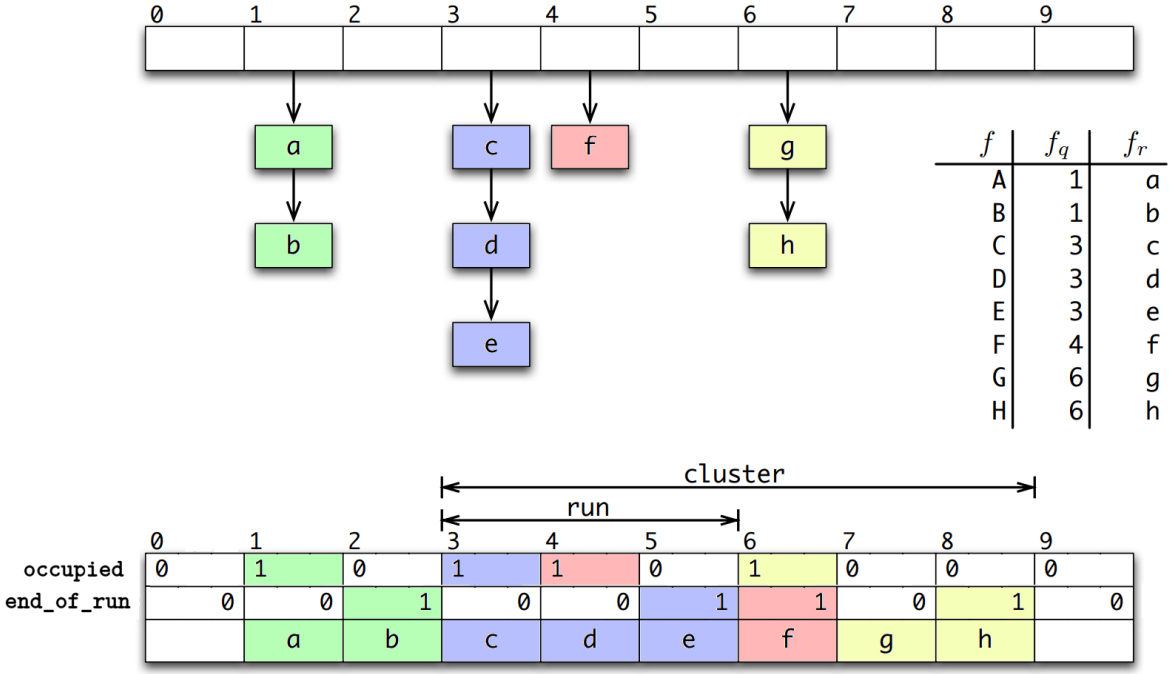
```
 1: function MAY_CONTAIN(Q, x)
 2:     b ← h_0(x)
 3:     if Q.occupieds[b] = 0 then
 4:         return 0
 5:     t ← RANK(Q.occupieds, b)
 6:     ℓ ← SELECT(Q.runends, t)
 7:     v ← h_1(x)
 8:     repeat
 9:         if Q.remainders[ℓ] = v then
10:             return 1
11:         ℓ ← ℓ − 1
12:     until ℓ < b or Q.runends[ℓ] = 1
13:     return false
```

```
 9: function INSERT(Q, x)
10:     r ← RANK(Q.occupieds, h_0(x))
11:     s ← SELECT(Q.runends, r)
12:     if h_0(x) > s then
13:         Q.remainders[h_0(x)] ← h_1(x)
14:         Q.runends[h_0(x)] ← 1
15:     else
16:         s ← s + 1
17:         n ← FIND_FIRST_UNUSED_SLOT(Q, s)
18:         while n > s do
19:             Q.remainders[n] ← Q.remainders[n − 1]
20:             Q.runends[n] ← Q.runends[n − 1]
21:             n ← n − 1
22:         Q.remainders[s] ← h_1(x)
23:         if Q.occupieds[h_0(x)] = 1 then
24:             Q.runends[s − 1] ← 0
25:         Q.runends[s] ← 1
26:     Q.occupieds[h_0(x)] ← 1
27:     return
```



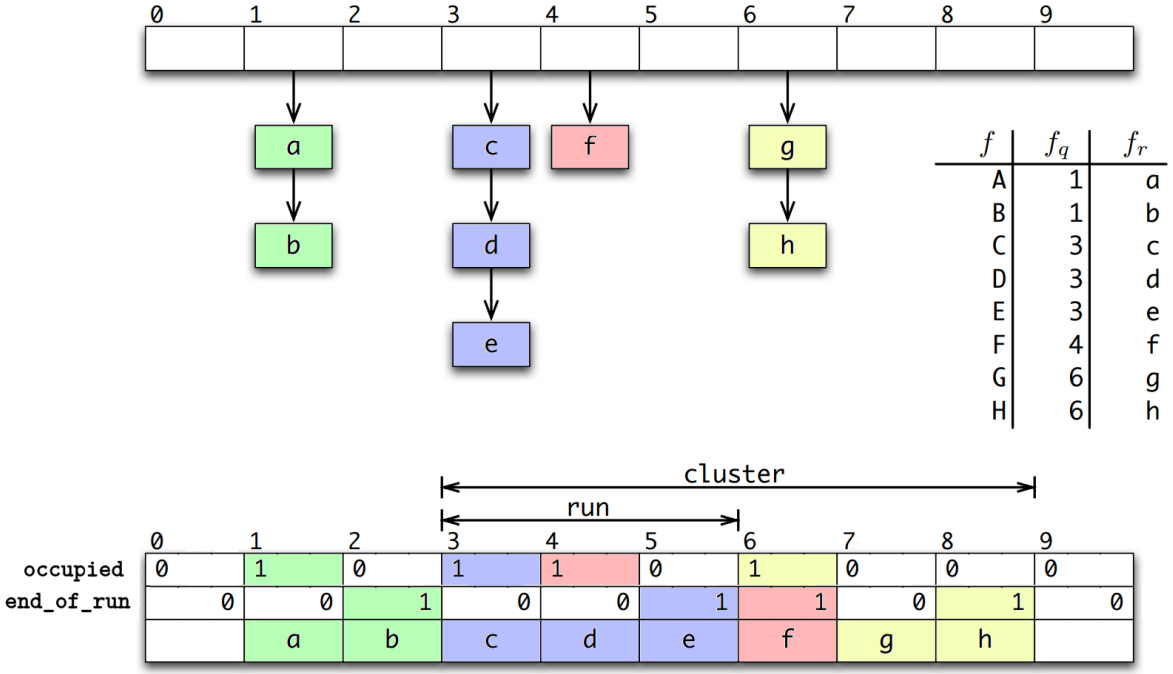
ako nájsť koniec runu?

- v lineárnom čase? ☹
- predpočítať rank & select – $O(1)$ čas
- predpočítať offsety – $(1+\varepsilon)(10+\lg(1/\delta))$ bitov :-/
- bloky po 64: $(1+\varepsilon)(2.125+\lg(1/\delta))$ bitov/prvok ☺



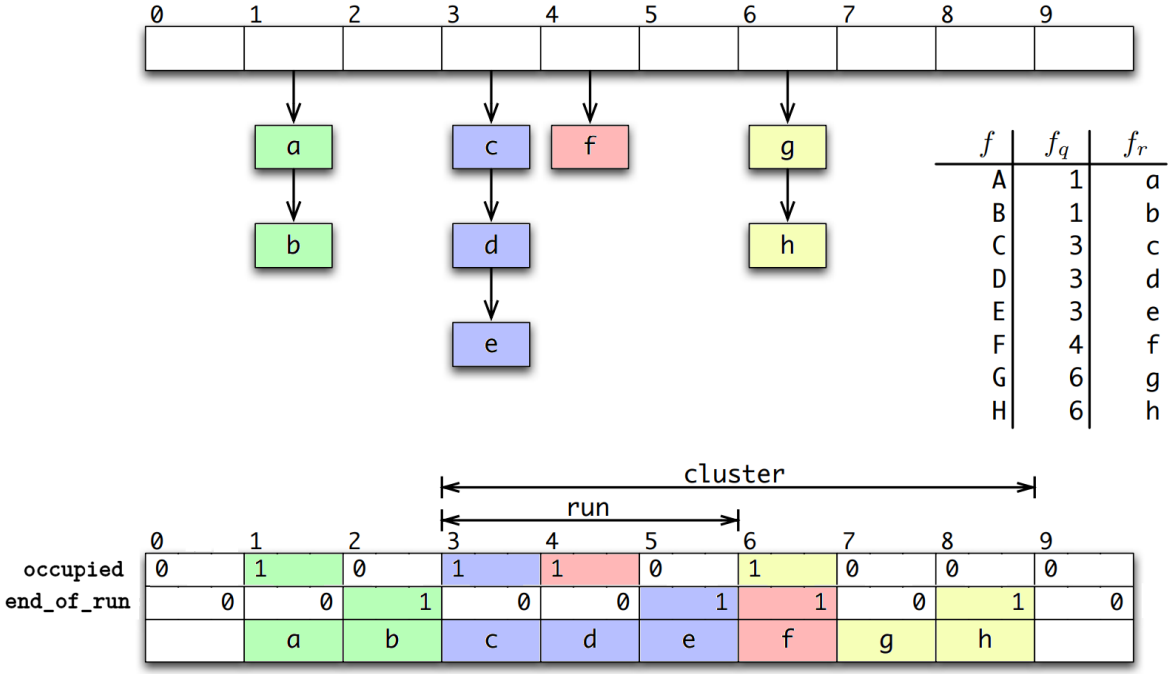
ako nájsť koniec runu?

- v lineárnom čase? ☹
- predpočítať rank & select – $O(1)$ čas
- predpočítať offsety – $(1+\varepsilon)(10+\lg(1/\delta))$ bitov :-/
- bloky po 64: $(1+\varepsilon)(2.125+\lg(1/\delta))$ bitov/prvok ☺



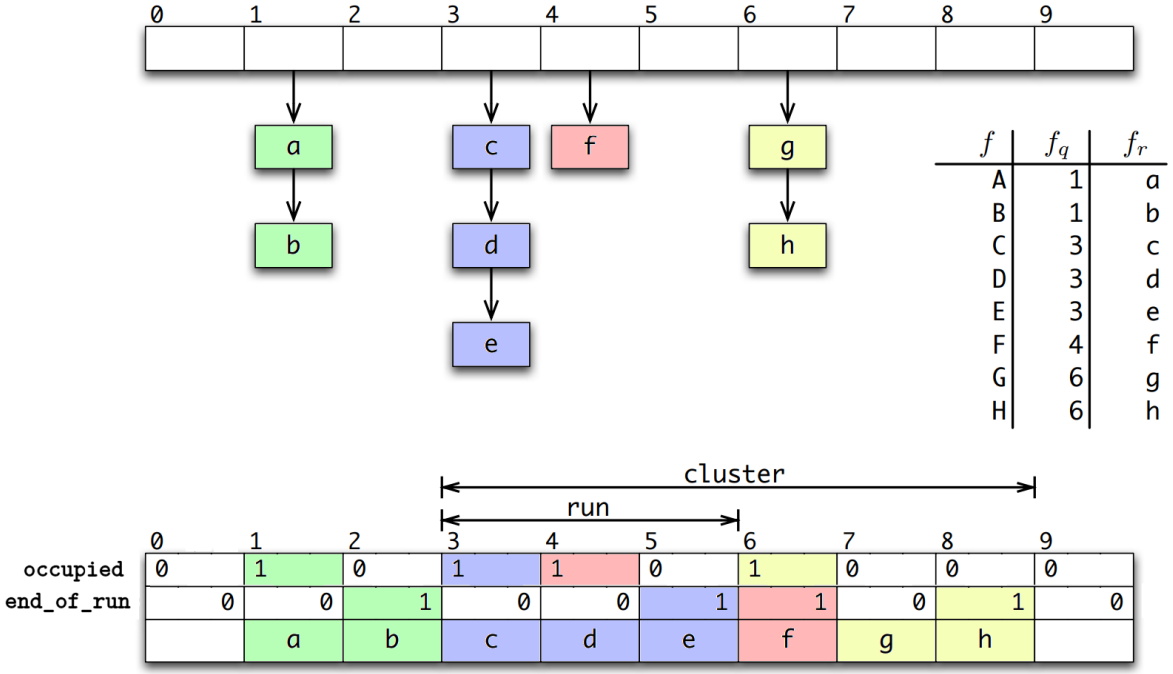
ako nájsť koniec runu?

- v lineárnom čase? ☹
- predpočítať rank & select – $O(1)$ čas
- predpočítať offsety – $(1+\varepsilon)(10+\lg(1/\delta))$ bitov :-/
- bloky po 64: $(1+\varepsilon)(2.125+\lg(1/\delta))$ bitov/prvok ☺



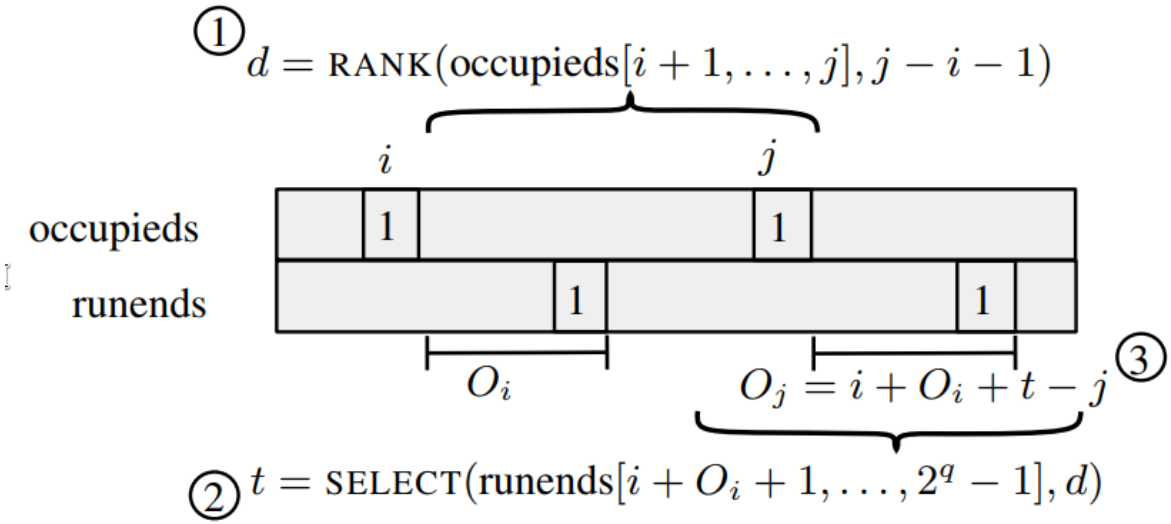
ako nájsť koniec runu?

- v lineárnom čase? ☹
- predpočítať rank & select – $O(1)$ čas
- predpočítať offsety – $(1+\varepsilon)(10+\lg(1/\delta))$ bitov :-/
- bloky po 64: $(1+\varepsilon)(2.125+\lg(1/\delta))$ bitov/prvok ☺



ako nájsť koniec runu?

- v lineárnom čase? ☹
- predpočítať rank & select – $O(1)$ čas
- predpočítať offsety – $(1+\varepsilon)(10+\lg(1/\delta))$ bitov :-/
- bloky po 64: $(1+\varepsilon)(2.125+\lg(1/\delta))$ bitov/prvok ☺


① $d = \text{RANK}(\text{occupieds}[i+1, \ldots, j], j-i-1)$
i
j
occupieds
runends
1
1
1
1
O_i
$O_j = i + O_i + t - j$
③
② $t = \text{SELECT}(\text{runends}[i + O_i + 1, \ldots, 2^q - 1], d)$

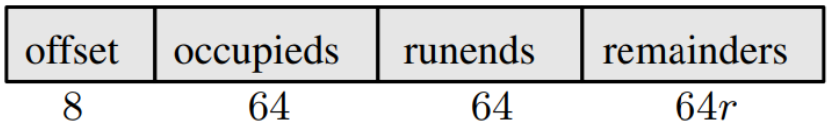


Figure 3: Layout of a rank-and-select-based-quotient-filter block. The size of each field is specified in bits.

- počet jednotiek v prvých i bitoch:

$$PopCount(v \,\&\, (2^i - 1))$$

- počet jednotiek v prvých i bitoch:

$$PopCount(v \,\&\, (2^i - 1))$$

- pozícia i-tej jednotky:

$$TZCnt(PDep(2^i, v))$$

PDEP — Parallel Bits Deposit

Description

PDEP uses a mask in the second source operand (the third operand) to transfer/scatter contiguous low order bits in the first source operand (the second operand) into the destination (the first operand). PDEP takes the low bits from the first source operand and deposit them in the destination operand at the corresponding bit locations that are set in the second source operand (mask). All other bits (bits not set in mask) in destination are set to zero.

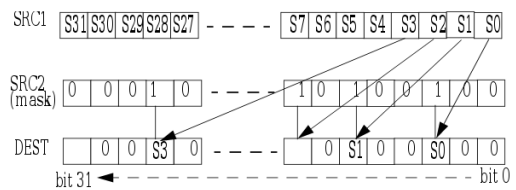


Figure 4-8. PDEP Example

Intel C/C++ Compiler Intrinsic Equivalent

```
PDEP: unsigned __int32 _pdep_u32(unsigned __int32 src, unsigned __int32 mask);
```

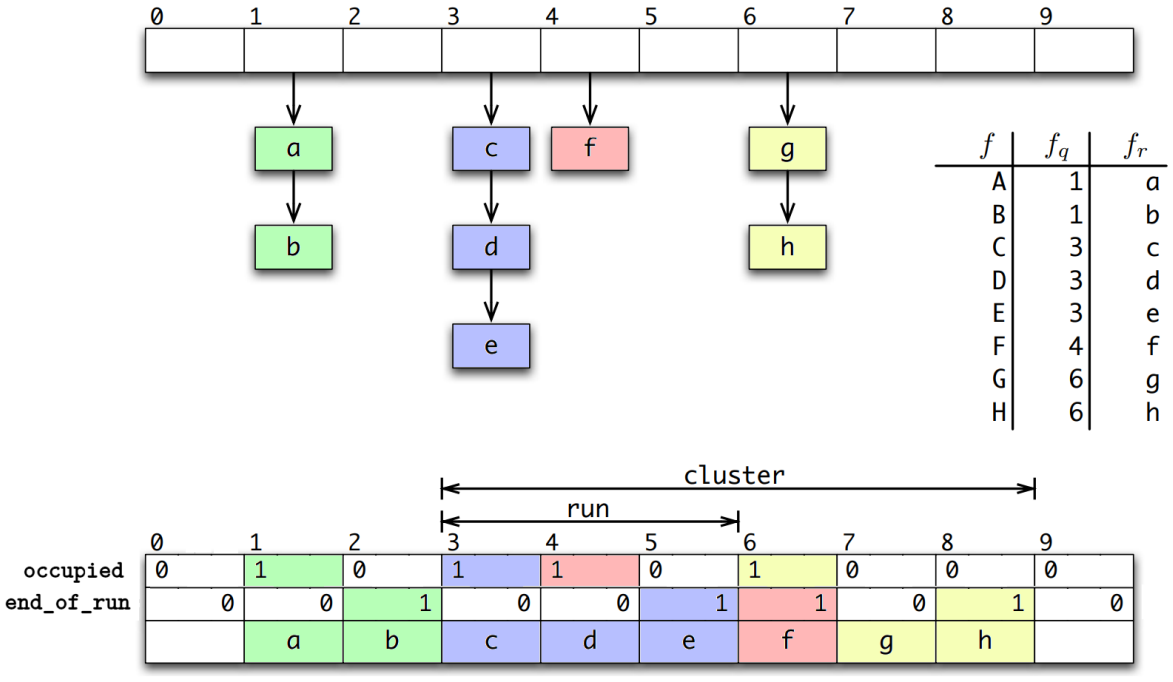
TZCNT — Count the Number of Trailing Zero Bits

Description

TZCNT counts the number of trailing least significant zero bits in source operand (second operand) and returns the result in destination operand (first operand). TZCNT is an extension of the BSF instruction. The key difference between TZCNT and BSF instruction is that TZCNT provides operand size as output when source operand is zero while in the case of BSF instruction, if source operand is zero, the content of destination operand are undefined. On processors that do not support TZCNT, the instruction byte encoding is executed as BSF.

Intel C/C++ Compiler Intrinsic Equivalent

```
TZCNT: unsigned __int32 _tzcnt_u32(unsigned __int32 src);
TZCNT: unsigned __int64 _tzcnt_u64(unsigned __int64 src);
```


0 1 2 3 4 5 6 7 8 9
a
b
c
d
e
f
g
h
f | f_q | f_r
A 1 a
B 1 b
C 3 c
D 3 d
E 3 e
F 4 f
G 6 g
H 6 h
cluster
run
occupied
end_of_run

- delete?

Count	Encoding	Rules
$C = 1$	x	none
$C = 2$	x, x	none
$C > 2$	$x, c_{\ell-1}, \ldots, c_0, x$	$x > 0$ $c_{\ell-1} < x$ $\forall i \ c_i \neq x$ $\forall i < \ell - 1 \ \ c_i \neq 0$
$C = 3$	$0, 0, 0$	$x = 0$
$C > 3$	$0, c_{\ell-1}, \ldots, c_0, 0, 0$	$x = 0$ $\forall i \ c_i \neq 0$

Table 3: Encodings for C occurrences of remainder x in the CQF.

Data Structure	QF	RSQF	CQF	CF	BF
Uniform random inserts per sec	11.12	12.06	11.19	14.25	2.84
Uniform successful lookups per sec	3.39	17.13	11.16	18.87	2.55
Uniform random lookups per sec	5.71	25.09	25.93	18.84	11.56
Bits per element	12.631	11.71	11.71	12.631	12.984

(a) In-memory uniform-random performance (in millions of operations per second).

Data Structure	CQF	CBF
Zipfian random inserts per sec	13.43	0.27
Zipfian successful lookups per sec	19.77	2.15
Uniform random lookups per sec	43.68	1.93
Bits per element	11.71	337.584

(b) In-memory Zipfian performance (in millions of operations per second).

Data Structure	RSQF	CQF	CF
Uniform random inserts per sec	69.05	68.30	42.20
Uniform successful lookups per sec	35.42	34.49	12.26
Uniform random lookups per sec	31.32	29.87	11.09
Bits per element	11.71	11.71	12.631

(c) On-SSD uniform-random performance (in thousands of operations per second).

Table 1: Summary of evaluation results. All the data structures are configured for 1/512 false-positive rate. We compare the quotient filter (QF) [5], rank-and-select quotient filter (RSQF), counting quotient filter (CQF), cuckoo filter (CF) [17], Bloom filter (BF) [26], and counting Bloom filter (CBF) [31].