10. cvičení

Datové struktury I, 9. 12. 2024 https://iuuk.mff.cuni.cz/~chmel/2425/ds1/

Úloha 1 (Delete v lineárním přidávání bez náhrobků)
Na přednášce jste viděli, jak implementovat operaci delete pro hashování s lineárním přidáváním pomocí náhrobků. Navrhněte, jak toto implementovat bez náhrobků.
Máte dvě možnosti jak to dělat: buď si u každého prvku nebudete pamatovat vůbec nic, nebo si u každého prvku budete pamatovat, jak daleko je od svého hashe.

Úloha 2 (Rolling hash je d-univerzální)
Pro prvočíslo p a délku vektoru d definujeme třídu hashovacích funkcí $\mathcal{R} = \{h_a \colon a \in \mathbb{Z}_p\}$, kde $h_a(x) = \sum_{i=0}^{d-1} x_{i+1} a^i$, a všechno počítáme modulo p. (Bereme $x \in \mathbb{Z}_p^d$, indexujeme od jedničky.)
Dokažte, že tato třída hashovacích funkcí je $(d-1)$-univerzální.
Hint: Zdůkladni větu algebry říkající, že každý polynom stupně d má nejvýše d kořenů – zkuste tam nějakým nějak nalézt.

Úloha 3 (Vyhledávání jehly v textu)
Vymyslete algoritmus na nalezení všech výskytů podřetězce x délky n v textu T délky m pomocí hashování, který běží v průměrném čase (tj. ve střední hodnotě) $\mathcal{O}(n + m + k \cdot n)$, kde k je počet výskytů x v T.

Úloha 4 (Přetečení v počítacím Bloomově filtru)
Uvažme počítací Bloomův filtr s maximální hodnotou jednoho počítadla ℓ. Tento Bloomův filtr bude mít m políček s počítadly, a budeme používat k zcela náhodných hashovacích funkcí. Určete pravděpodobnost toho, že nám tento Bloomův filtr přeteče (tedy budeme mít aspoň jedno počítadlo, které se dostalo na hodnotu ℓ). Pro jednoduchost můžete začít nejprve s $k = 1$, a potom zobecnit pro libovolné k.

Úloha 5 (FKS (Fredman, Komlós, Szemerédi))
Ukážeme si konstrukci (statické) hashovací tabulky pro podmnožinu S velikosti n univerza $\mathcal{U}$, která nemá žádné kolize. Možná jste narazili na konstrukci, která potřebovala $\Omega(n^2)$ paměti (přesněji paměťových buněk). My to zvládneme s lineárním počtem paměťových buněk (za předpokladu, že můžeme mít zcela náhodnou hashovací funkci, tu dokážeme sestrojit a samplovat v konstantním čase a že si ji dokážeme pamatovat v konstantním prostoru)[1].
Proces stavby tabulky bude probíhat následovně: budeme stavět dvě úrovně. V první úrovni si zcela náhodnou hashovací funkcí f rozdělíme prvky S do kyblíků $B_1, \ldots, B_n$ (a označíme si $b_i := |B_i|$). V druhé úrovni pak postavíme bezkolizní tabulku pomocí konstrukce, kdy máme pro kyblík B_i tabulku velikosti $2b_i^2$ pro b_i prvků, a zkoušíme náhodně volit vhodnou hashovací funkci, dokud nemáme žádné kolize.

První úroveň: V konstantním čase si vybereme náhodnou hashovací funkci $f : \mathcal{U} \to [n]$, a tou rozdělíme S do kyblíků. Toto opakujeme, dokud neplatí, že $\sum_{i=1}^{n} b_i^2 \leq \beta n$ pro $\beta = 4$.
Chceme ukázat, že tento krok budeme ve střední hodnotě opakovat nejvýše dvakrát. Označme jako C počet kolizí.

a) Určete $\mathbb{E}[C]$.

b) Určete C v závislosti na b_i.

c) Na základě předchozích dvou hodnot určete $\mathbb{E}[\sum_{i=1}^{n} b_i^2]$.

d) Aplikujte Markovovu nerovnost na náhodnou veličinu $X = \sum_{i=1}^{n} b_i^2$ s vhodnou hodnotou, abychom dostali požadovaný výsledek. (Taky se bude hodit střední hodnota geometrického rozdělení.)

Druhá úroveň: Ve druhé úrovni pro každé $i \in [n]$ volíme v i-tém kyblíku univerzální hashovací funkci $g_i : \mathcal{U} \to [\alpha b_i^2]$ pro $\alpha = 2$. Toto opakujeme, dokud není prostá pro prvky v kyblíku B_i.
Označme jako C_x počet kolizí klíče $x \in B_i$ na druhé úrovni.

a) Shora odhadněte $\mathbb{E}[C_x]$.

[1] Totéž jde udělat s rozumně univerzální funkcí, tohle je jenom pro jednoduchost.

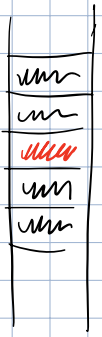
b) Použijte Markovovu nerovnost a union bound, abyste shora odhadli pravděpodobnost existence prvku s aspoň jednou kolizí.

c) Kolikrát budeme muset proces opakovat? ([Sem si vložte si svou oblíbenou poznámku o střední hodnotě geometrického rozdělení.])

Bonusové úlohy

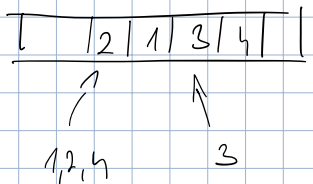
Úloha 6 (Něco „praktičtějšího“)
Při hashování často počítáme modulo, ale to se přeloží do strojového kódu jako operace `idiv`, která je hodně drahá (latence pro 64 bitová čísla může být, dle procesoru, klidně i řádově desítky cyklů). Existují ale speciální prvočísla, tzv. Mersennova, pro která to jde rychleji. Ta mají tvar $p = 2^s - 1$.
Zkuste naimplementovat modulo Mersennovo prvočíslo jen pomocí bitových operací (bitshift, bitwise and/or), sčítání, odčítání a porovnávání (obecně rychlých operací).

Úloha 1 (Delete v lineárním přidávání bez náhrobků)
Na přednášce jste viděli, jak implementovat operaci delete pro hashování s lineárním přidáváním pomocí náhrobků. Navrhněte, jak toto implementovat bez náhrobků.
Máte dvě možnosti jak to dělat: buď si u každého prvku nebudete pamatovat vůbec nic, nebo si u každého prvku budete pamatovat, jak daleko je od svého hashe.



„stačí si pamatovat max(„jak daleko od hashe jsem to posledl)

a nebo:



když smažu 1, můžu přesunout 4 na pozici bývalé 1. Pak se mi vytvořila díra na předchozí pozici 4 a vidím to zpátky.

Úloha 2 (Rolling hash je d-univerzální)
Pro prvočíslo p a délku vektoru d definujeme třídu hashovacích funkcí $\mathcal{R} = \{h_a : a \in \mathbb{Z}_p\}$, kde $h_a(x) = \sum_{i=0}^{d-1} x_{i+1} a^i$, a všechno počítáme modulo p. (Bereme $x \in \mathbb{Z}_p^d$, indexujeme od jedničky.)
Dokažte, že tato třída hashovacích funkcí je $(d-1)$-univerzální.
Hint: *Základní věta algebry říká, že každý polynom stupně d má nejvýše d kořenů – zkuste tam nějakým nějak najít.*

Potřebujeme $P(h(x) = h(y)) \leq \frac{d}{m}$ pro $x \neq y$

$x_1 + x_2 a + x_3 a^2 + \dots x_d a^{d-1}$
$y_1 + y_2 a + y_3 a^2 + \dots y_d a^{d-1}$
} vždy může mít nejvýš d kořenů

Uvažme $P\left[\sum_{i=0}^{d-1} (x_{i+1} - y_{i+1}) a^i = 0\right] = \frac{d-1}{p}$

tohle je ale znovu polynom stupně nejvýše d, absolutní člen je zafixovaný. Takže mám zbylých $d-1$ voleb kořenu a pro vektor $(x-y)$.

Úloha 3 (Vyhledávání jehly v textu)
Vymyslete algoritmus na nalezení všech výskytů podřetězce x délky n v textu T délky m pomocí hashování, který běží v průměrném čase (tj. ve střední hodnotě) $\mathcal{O}(n + m + k \cdot n)$, kde k je počet výskytů x v T.

ověření správnosti

$h(x_1 \_ x_j) = h(x_0 \_ x_{j-1}) \cdot d^{-1} - x_0 + x_j \cdot d^{j-1}$ -> takto mám velmi rychle porovnávat substringy

Kolik může vzniknout kolizí: $P(x/p)$ je šance kdice jednoho substringu.
Mám celkem $O(n)$ substringů. Takže celkem mám $O\left(\frac{n \cdot x}{p}\right)$ falešných negativ.
Pokud $p \geq x$, pak jich je $O(n)$.

Úloha 4 (Přetečení v počítacím Bloomově filtru)
Uvažme počítací Bloomův filtr s maximální hodnotou jednoho počítadla ℓ. Tento Bloomův filtr bude mít m políček s počítadly, a budeme používat k zcela náhodných hashovacích funkcí. Určete pravděpodobnost toho, že nám tento Bloomův filtr přeteče (tedy budeme mít aspoň jedno počítadlo, které se dostalo na hodnotu ℓ). Pro jednoduchost můžete začít nejprve s $k = 1$, a potom zobecnit pro libovolné k.