10. cvičení

Datové struktury I, 13. 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.

Řešení
Projdeme celý řetězec, a přesouváme dozadu (pokud můžeme – může se stát, že přesunutím by se prvek buď přesunul před svůj hash). Bez poznámek musíme spočítat všechny hashe, s čísly stačí koukat na čísla. (Ještě lépe to umí Robin Hood – česky též „zbojnické“ – hashování, které vzdálenosti používá i u insertu.)

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: *Základní věta algebry říká, že každý polynom stupně d má nejvýše d kořenů – zkuste tam nějaký najít.*

Řešení
Zajímá nás pravděpodobnost, že pro $x \neq y \in \mathbb{Z}_p^d$ máme $h_a(x) = h_a(y)$. To je ekvivalentní tomu, že $h_a(x) - h_a(y) = 0$, a to můžeme přepsat z definice na $\sum_{i=0}^{d-1}(x_{i+1} - y_{i+1})a^i$, což je polynom v proměnné a se stupněm nejvýše $d-1$. Ze základní věty algebry pak máme, že tento polynom má nejvýše $d-1$ kořenů, a právě tyto kořeny určují funkce h_a, ve kterých se x a y zahashují na téže místo. Celkem tedy máme pravděpodobnost kolize x a y $\frac{d-1}{p}$, a tedy máme $(d-1)$-univerzalitu.

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.

Řešení
Rabin-Karp s Rolling hashem, čas: máme m času na projití řetězce, v každém kroku uděláme konstantní úpravu hashe a zkontrolujeme, že není stejný. Pokud je hash stejný, zkontrolujeme celý string. Protože máme hashování do nějakého $\mathbb{Z}_p$, pravděpodobnost kolize je d/p, a tedy celkem máme ve střední hodnotě asi $(k + md/p)$ kolizí. Pokud ale zvolíme $p > m \cdot d$ (nebo obecně stačí $p \in \Omega(m \cdot d)$), máme konstantně mnoho falešných kolizí ve střední hodnotě.

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.

Řešení
Pro přetečení je potřeba, aby se nám na nějaký index trefilo alespoň ℓ přičtení jedničky. Začneme s pravděpodobností, že se tam trefilo *právě* ℓ přičtení: $\binom{nk}{\ell} \cdot (\frac{1}{m})^\ell \cdot (1 - \frac{1}{m})^{nk-\ell}$. Tedy pravděpodobnost, že máme alespoň ℓ přičtení je přesně $1 - \sum_{i=1}^{\ell-1} \binom{nk}{i} \cdot (\frac{1}{m})^i \cdot (1 - \frac{1}{m})^{nk-i}$, ale můžeme ji shora odhadnout jako $\binom{nk}{\ell} \cdot (\frac{1}{m})^\ell$, protože aspoň ℓ věcí se musí zahashovat, a . Díky odhadu $\binom{nk}{\ell} \leq (kne/\ell)^\ell$ můžeme odhadnout celou pravděpodobnost jako $(\frac{nke}{\ell m})^\ell$, a z přednášky víme, že optimální volba m je cca $kn/\ln 2$, a tedy máme pravděpodobnost cca $(e \ln 2/\ell)^\ell$, a na odhad přes všechna počítadla použijeme union bound.

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].

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

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 $\mathbb{E}[C_x]$.

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.])

Řešení
První část:

a) $\mathbb{E}_h[C] = \sum_{x \neq y \in S} \Pr[h(x) = h(y)] = \binom{n}{2} \frac{1}{n} = \frac{n-1}{2}$

b) $C = \sum_{i=1}^{n} \binom{b_i}{2} \rightsquigarrow 2C = \sum_{i=1}^{n} (b_i^2 - b_i) = \sum_{i=1}^{n} (b_i^2) - n \rightsquigarrow \sum_{i=1}^{n} b_i^2 = 2C + n.$

c) $\mathbb{E}[b_i^2] = 2\mathbb{E}[C] + n = 2n - 1$

d) $\Pr[X \geq 4n] \leq \frac{2n-1}{4n} \leq \frac{1}{2}$, a tedy ve střední hodnotě budeme potřebovat 2 pokusy, než najdeme vhodnou funkci.

Druhá část:

a) $\mathbb{E}[C_x] \leq \frac{b_i}{\alpha b_i^2} = \frac{1}{\alpha b_i}$.

b) Označíme $C' = \sum_{x \in B_i} C_x$, pak $\Pr[C' \geq 1] \leq \sum_{x \in B_i} \Pr[C_x \geq 1] \leq \sum_{x \in B_i} \frac{1}{\alpha b_i} = \frac{1}{\alpha}$.

c) Ve střední hodnotě tedy potřebujeme $\alpha = 2$ pokusy pro nalezení vhodné funkce.

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).

Řešení
Chceme spočítat $k \mod p, p = 2^s - 1$. Stačí vzít `i = (k & p) + (k >> s)`, a vrátit `if i >= p then i-p else i`. Tohle předpokládá $k \leq 2^{2s} - 1$, jinak bude potřeba počítat rekurzivně.
Proč? Rozdělíme $k = x \cdot 2^s + r \equiv_{2^s-1} x + r$, kde r získáme bitmaskou, a k jsme bitshiftem vydělili 2^s.

Užitečné definice

Definice (c-univerzální systém fcí)**.** Systém $\mathcal{H}$ funkcí $h : \mathcal{U} \to [m]$ je c-univerzální pro $c > 0$, pokud pro všechna $x \neq y$ platí $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq \frac{c}{m}$.
Systém $\mathcal{H}$ je univerzální, pokud je c-univerzální pro nějaké $c > 0$.