11. cvičení

Datové struktury I, 13. 12. 2022 https://iuuk.mff.cuni.cz/~chmel/2223/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 [„zbojnické“] hashování, které vzdálenosti používá i u insertu.)

Úloha 2 (FKS (Fredman, Komlós, Szemerédi))
Ukážeme si konstrukci (statické) hashovací tabulky pro podmnožinu S univerza $\mathcal{U}$, která nemá žádné kolize. Na přednášce jste měli 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 z přednášky.
Začneme s první úrovní. 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.

1. Určete $\mathbb{E}[C]$.
2. Určete C v závislosti na b_i.
3. Na základě předchozích dvou hodnot určete $\mathbb{E}[\sum_{i=1}^{n} b_i^2]$.
4. 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.)

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.

1. Shora odhadněte $\mathbb{E}[C_x]$.
2. Použijte Markovovu nerovnost a union bound, abyste shora odhadli pravděpodobnost existence prvku aspoň s jednou kolizí.
3. Kolikrát budeme muset proces opakovat? ([Sem si vložte si svou oblíbenou poznámku o střední hodnotě geometrického rozdělení.])

Řešení
První část:

1. $\mathbb{E}_h[C] = \sum_{x \neq y \in S} \Pr[h(x) = h(y)] = \binom{n}{2} \frac{1}{n} = \frac{n-1}{2}$
2. $C = \sum_{i=1}^{n} \binom{b_i}{2} \rightsquigarrow 2C = \sum_{i=1}^{n} (b_i^2 - b_i) = \sum_{i=1}^{n} (b_i^2) - n \rightsquigarrow \sum_{i=1}^{n} b_i^2 = 2C + n.$
3. $\mathbb{E}[b_i^2] = 2\mathbb{E}[C] = 2n - 1$
4. $\Pr[X \geq 4n] \leq \frac{2n-1}{4n} \leq \frac{1}{2}$, a tedy ve střední hodnotě budeme potřebovat 2 pokusy, než najdeme vhodnou funkci.

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

Druhá část:

1. $\mathbb{E}[C_x] \leq \frac{b_i}{\alpha b_i^2} = \frac{1}{\alpha b_i}$.
2. Označíme $C' = \sum_{x \in B_i} C_x$, pak $\Pr[C' \geq 1] \leq \sum_{x \in B_i} \Pr[C_x \geq 1] \leq \sum_{x \in B_i} \frac{1}{\alpha b_i} = \frac{1}{\alpha}$.
3. Ve střední hodnotě tedy potřebujeme $\alpha = 2$ pokusy pro nalezení vhodné funkce.

Bonusové úlohy

Úloha 3 (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 které 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.