10. cvičení

Datové struktury I, 4. 12. 2023 https://iuuk.mff.cuni.cz/~chmel/2324/ds1/

Úloha 1 (4-nezávislost tabulkového hashování)
Ukažte, že tabulkové hashování není 4-nezávislé.
Hint:·*Zkuste najít nějakou čtveřici vstupů takovýchˌ že hashe prvních tří jednoznačně určují hash toho čtvrtého.*

Úloha 2 (Rehashujeme)
Jednoduchá implementace rehashe u kukačkového hashování je, že si všechny hodnoty vložíme do pomocného pole, a potom je po jednom insertujeme. Vymyslete implementaci rehashe, která pomocné pole nepotřebuje. (Pozor na to, že během rehashe můžeme znovu začít s rehashem.)

Úloha 3 (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.
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.])

Úloha 4 (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.

Bonusy

Úloha 5 (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).

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

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 $\Pr_h[h(x) = h(y)] \leq \frac{c}{m}$.
Systém $\mathcal{H}$ je univerzální, pokud je c-univerzální pro nějaké $c > 0$.

Definice (Tabulkové hashování)**.** Představme si, že chceme zahashovat n-bitové řetízky do m-bitových řetízků, kde $n = k \cdot \ell$. Řetízek $x \in \{0,1\}^n$ pak rozložíme do k částí délky ℓ, které značíme x_i. Můžeme tedy psát $x = x^1 x^2 \dots x^k$. Pak generování naší hashovací funkce $h : \{0,1\}^n \to \{0,1\}^m$ vypadá tak, že vybereme uniformně náhodně k funkcí $T_i : \{0,1\}^\ell \to \{0,1\}^m$ (tyto reprezentujeme tabulkou, proto tabulkové hashování). Vyhodnocujeme pak $h(x) = \bigoplus_{i=1}^{k} T_i(x^i) = T_1(x^1) \oplus T_2(x^2) \oplus \dots \oplus T_k(x^k)$, kde $\oplus$ značí XOR (po jednotlivých bitech).

Definice (Kukačkové hashování)**.** V každém okamžiku máme dvě hashovací funkce $f, g : \mathcal{U} \to [m]$ volené uniformně náhodně z nějakého systému hashovacích funkcí a jedno pole velikosti m. Naším cílem je, že každý prvek x, který je zahashovaný, se vyskytuje v jednom ze dvou „hnízd“ $f(x)$ nebo $g(x)$.
Lookup se podívá na tato dvě místa, a podle výsledku buď řekne, zda se tam prvek nachází, nebo ne.
Insert probíhá následovně: pokud je jedno z hnízd $f(x)$ nebo $g(x)$ volné, usadíme x do volného místa. Jinak vybereme jedno z plných míst (řekněme $f(x)$), x do něj vložíme, a vyjmeme prvek x_1, který byl v tomto hnízdě původně uložený. Teď musíme uložit x_1, a to vložíme do toho hnízda $f(x_1), g(x_1)$, ze kterého jsme jej *ne*vyjmuli – takže jej dáme do druhého hnízda než bylo $f(x)$. Takhle můžeme nějakou dobu pokračovat, dokud nenajdeme prázdné místečko, nebo dokud nedojde k tomu, že už takhle přesouváme prvky příliš dlouho (řekněme $6 \log(m)$ nebo $6 \log(n)$, kde m je počet hnízd/přihrádek a n je počet uskladňovaných prvků). Potom se na tento pokus o vložení vykašleme, a začneme znovu s tím, že si vygenerujeme nové funkce f a g, a všechny prvky v našem poli přehashujeme.

Tvrzení (Union bound)**.** Pro jevy A_1, A_2 platí, že $P[A_1 \cup A_2] \leq P[A_1] + P[A_2]$.

Věta (Markovova nerovnost)**.** Buď X nezáporná náhodná veličina. Pak $\forall \varepsilon > 0$ platí $P[X \geq \varepsilon] \leq \frac{\mathbb{E}[X]}{\varepsilon}$.
Ekvivalentně pro jakékoliv $d \geq 1$, $P[X \geq d \cdot \mathbb{E}[X]] \leq \frac{1}{d}$.