9. cvičení

Datové struktury I, 2. 12. 2024 https://iuuk.mff.cuni.cz/~chmel/2425/ds1/

Úloha 1 (Špatná verze kukačky)
Proč je následující implementace insertu pro kukačkové hashování problematická? (Implementaci a podmínky pro rehashování pro tento příklad meteme pod koberec.)

```
for i=1 to n
  if T[h1(x)] je prázdné
    T[h1(x)] = x
    return
  swap(T[h1(x)], x)
  if T[h2(x)] je prázdné
    T[h2(x)] = x
    return
  swap(T[h2(x)], x)
```

Řešení
Protože bychom měli vždycky prohazovat hashovací funkci, ale prvek, se kterým jsme x vyměnili, mohl na tohle místo být zahashovaný, takže ho potom zase vyhodíme, a takhle se budeme cyklit dokola.

Úloha 2 (4-nezávislost tabulkového hashování)
Ukažte, že tabulkové hashování není 4-nezávislé (pokud používáme aspoň dvě tabulky).
Hint: *Zkuste najít nějakou čtveřici vstupů takových, že hashe prvních tří jednoznačně určují hash toho čtvrtého.*

Řešení
Mějme a, b, c, d takové, že $a = XXa^3 \dots a^k, b = XYa^3 \dots a^k, c = YXa^3 \dots a^k, d = YYa^3 \dots a^k$. Pak $h(a) \oplus h(b) \oplus h(c) \oplus h(d) = 0^\ell$, neboť hashe a^i vyXORujeme vždycky čtyřikrát, a hashe X, Y v každém bloku vyXORujeme právě dvakrát. Speciálně to tedy znamená, že $\Pr[h(a) = \alpha \wedge h(b) = \beta \wedge h(c) = \gamma \wedge h(d) = \alpha \oplus \beta \oplus \gamma] = \Pr[h(a) = \alpha \wedge h(b) = \beta \wedge h(c) = \gamma] = \frac{1}{m^3}$.

Věta. Tabulkové hashování je 3-nezávislé.

Úloha 3 (Tuhle větu si dokážeme)
Dokažte předcházející větu s následujícím postupem. Mějme $a, b, c \in \mathbb{Z}_2^\ell, x \neq y \neq z \neq x \in \mathbb{Z}_2^w$, a používejme tabulkové hashování s d částmi. Pak chceme ukázat, že $\Pr_{h \in \mathcal{H}}[h(x) = a \wedge h(y) = b \wedge h(z) = c] \leq \frac{1}{m^3}$.

a) Prvně si uvědomme, že pokud máme jen jednu část, a tedy jednu tabulku, tvrzení je triviální.

   Dále mějme alespoň dvě části. Protože x, y, z jsou různé, musí se (po dvou) lišit alespoň v jedné části.

b) Začneme s případem, kdy existuje část i, že x^i, y^i, z^i jsou všechny různé. Mějme jakkoliv zvolené ostatní tabulky, kromě tabulky T_i. S jakou pravděpodobností můžeme zvolit funkci pro tabulku T_i tak, že $h(x) = a, h(y) = b, h(z) = c$?

c) Jinak existují (BÚNO) části i, j takové, že $z^i = x^i \neq y^i$ a $y^j = x^j \neq z^j$. Potom máme následující soustavu rovnic, kde v_x, v_y, v_z jsou vyXORované výsledky z ostatních tabulek:

$$T_i[x^i] \oplus T_j[x^j] \oplus v_x = a$$
$$T_i[y^i] \oplus T_j[y^j] \oplus v_y = b$$
$$T_i[z^i] \oplus T_j[z^j] \oplus v_z = c$$

   Opět si představme, že v_x, v_y, v_z už známe. S jakou pravděpodobností budou náhodně volené tabulky T_i, T_j splňovat tuto soustavu rovnic?

d) Uvědomte si, že toto stačí.

Řešení a) Máme jednu tabulku, takže máme uniformně náhodnou funkci z $\{0,1\}^\ell$ do $\{0,1\}^w$, a ta je automaticky nezávislá.

b) Máme zafixované všechny hodnoty, a víme, že $h(x)$ musí být a, tedy speciálně $T_i(x^i) = a \oplus \bigoplus_{j=1 j\neq i}^{k} T_j(x^j)$, a to je dáno jednoznačně. Pro y, z platí totéž analogicky, a tedy máme pro volbu $T_i(x^i), T_i(y^i), T_i(z^i)$ právě jednu možnost z celkem $2^{3w} = m^3$ možností, a tedy v tomto případě máme 3-nezávislost.

c) Uvědomme si, že máme vlastně jen tři hodnoty, protože $z^i = x^i$ a $y^j = x^j$, pak naše soustava rovnic je

$$T_i[x^i] \oplus T_j[x^j] \oplus v_x = a$$
$$T_i[y^i] \oplus T_j[x^j] \oplus v_y = b$$
$$T_i[x^i] \oplus T_j[z^j] \oplus v_z = c$$

Kolik má tato soustava řešení? Hodnoty v_x, v_y, v_z předpokládáme, že už jsou zafixované, tedy naše soustava rovnic se zjednoduší, zároveň označíme $W = T_i[x^i], X = T_i[y^i], Y = T_j[x^j], Z = T_j[z^j]$, a máme pak

$$W \oplus Y = a \oplus v_x$$
$$X \oplus Y = b \oplus v_y$$
$$W \oplus Z = c \oplus v_z$$

Tady vidíme, že když zvolíme libovolné Z, pak W, Y, X jsou jednoznačně určené. Tedy celkem máme m možných řešení této soustavy rovnic, a máme m^4 způsobů, jak W, X, Y, Z vybrat (jakýmkoliv způsobem, i když rovnice neplatí). Tedy náhodnými volbami tuto rovnost splníme s pravděpodobností $m/m^4 = 1/m^3$.

d) Přesně tak: v každém případě tedy máme pravděpodobnost toho, že se hodnoty trefí právě $1/m^3$, což je přesně to, co po nás chce definice nezávislosti.

Úloha 4 (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.)

Řešení
Stačí projít pole T podle indexů, a když najdeme prvek ve špatné buňce, tak ho odstraníme, a znovu vložíme.

Užitečné definice

Definice (k-nezávislý systém fcí)**.** Systém $\mathcal{H}$ funkcí $h : \mathcal{U} \to [m]$ je (k, c)-nezávislý pro nějaká $k \geq 1, c > 0$, pokud $\Pr_{h\in\mathcal{H}}[h(x_1) = a_1 \wedge \ldots \wedge h(x_k) = a_k] \leq \frac{c}{m^k}$ pro libovolná $x_1, \ldots, x_k$ různá, $a_1, \ldots, a_k$ ne nutně různá. Systém $\mathcal{H}$ je k-nezávislý, pokud je (k, c)-nezávislý pro nějakou nezávislou konstantu c.

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 $x = x^1 x^2 \ldots x^k$. 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 $h(x) = \bigoplus_{i=1}^{k} T_i(x^i) = T_1(x^1) \oplus T_2(x^2) \oplus \cdots \oplus T_k(x^k)$, 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.