13. Hashování

(zapsali Martin Franců a Bohdan Maslowski)

V předchozích kapitolách jsme ukázali, že ačkoliv v obecném případě nelze třídit rychleji než v čase $\Theta(n \log n)$, pro některé speciální druhy hodnot (malá celá čísla, řetězce apod.) existuje efektivnější lineární algoritmus. Nyní si podobným způsobem posvítíme na reprezentaci množin – navrhneme datovou strukturu, která bude umět s vhodnými typy prvků provádět základní množinové operace (*Insert*, *Delete*, *Find*) v *průměrně* konstantním čase (pomocí vyhledávacích stromů jsme to zatím uměli v čase $\mathcal{O}(\log n)$ na operaci).

Jaké máme možnosti? Začněme od těch nejjednodušších a označme si n velikost množiny a U velikost univerza, z nějž prvky množiny vybíráme.

1. Pole indexované od 0 **do** $U - 1$

Insert, *Delete* i *Find* pracují v čase $\mathcal{O}(1)$, ale platíme za to paměťovou složitostí $\Theta(U)$.

2. Číslicový strom (radix tree)

Zvolíme základ soustavy b, prvky universa zapisujeme jako k-tice číslic. Čili:

$$x \in U \mapsto [b]^k, k = \lceil \log_b U \rceil.$$

Tím nám vznikne strom s hloubkou k, na k-té úrovni se rozhodujeme podle k-té číslice.

Operace *Find* bude trvat $O(\log_b U)$, operace *Insert* a *Delete* pak $O(b \log_b U)$. Strom zabere prostor velikosti $O(nb \log_b U)$.

Čím bude základ b menší, tím lepší paměťová a horší časová složitost, a obráceně. Číslicový strom tedy vyžaduje pečlivé nastavení parametrů.

3. Písmenkový strom

Používá se pro řezězce, za základ soustavy b zvolíme abecedu.

$|x|$…délka řetězce

Find, Insert, Delete: $O(|x|)$

Paměť: $O(\sum_i |x|)$

4. Hashování

h … hashovací funkce $[U] \rightarrow [m]$

m … počet přihrádek, v každé přihrádce spojový seznam.

Perfektní hashovací funkce přiřadí do každé z m přihrádek jediný prvek z $[U]$.

Příklad: Jaká je pravděpodobnost, že 2 z k lidí mají narozeniny ve stejný den? Stačí 23 lidí, aby nastala kolize s pravděpodobností větší než jedna polovina. Aby nenastala, potřebovali bychom větší počet přihrádek: $p = cm^2$

Hashování se separovými řetezci

Insert, Delete i Find pracují v čase $O(|x|)$, paměťová složitost je $O(m+n)$

vložili jsme $x_1 \dots x_n \in [U]$
nyní pracujeme s $x \in [U]$
předpokládejme, že $h(x_i)$ a $h(x)$ jsou náhodné
$p := h(x)$
$N_p := \sharp : h(x_1) = h(x)$
$N_{i,p} := 0$ pokud $h(x_i) \neq p$
$N_{i,p} := 1$ pokud $h(x_i) = p$
$N_p = \sum_{i=1}^{n} N_{i,p}$
$EN_{i,p} = Pr[h(x_i) = p] = \frac{1}{m}$
$EN_p = n\frac{1}{m} = \frac{n}{m}$

Za předpokladu, že hash přiděluje náhodnou přihrádku, je v každé přihrádce $\frac{n}{m}$ prvků.

Druhá půlka

VĚTA: *Pokud jsme do hashovací tabulky vložili hodnoty $x_1, \dots, x_m$ tak, že h_{x_i} jsou všechny stejně pravděpodobné, pak následující Insert, Delete, Find pracují v čase $\mathcal{O}\left(\frac{n}{m}+1\right)$. (Za předpokladu, že výpočet h trvá $\mathcal{O}(1)$.)*

Insert: Spočítá si $h(x)$ a vloží jej do spojového seznamu v patřičné přihrádce.

Delete: Spočítá si $h(x)$ a najde jej v spojovém seznamu v patřičné přihrádce a odstraní jej.

Find: Spočítá si $h(x)$ a najde jej v spojovém seznamu v patřičné přihrádce.

Tato metoda má dva háčky:

1. vysoká koliznost
2. musíme dopředu vědět, kolik prvků budeme vkládat

Nafukovací hashovací tabulka

na počátku: $n = 0; m = 4$
v každém okamžiku budeme udržovat poměr $\frac{n}{m} \in \langle \frac{1}{2}, 1)$
tedy: (pokud nastane)

$$n = 4 \Rightarrow m := 8$$

$$n = 8 \Rightarrow m := 16$$

Při každém zvětšování tabulky musíme zvětšit obor hodnot hashovací funkce a všechny prvky již v tabulce uložené znovu "zhashovat".

Nafukování trvá $\mathcal{O}(n)$, ale mezi dvěma nafouknutími je minimálně $\frac{n}{2}$ Insertů.

Pokud tedy každý Insert přispěje konstantou, tak to máme zadarmo!

PROBLÉM: Jak udělat funkci Delete? Jednoduché zmenšování tabulky při poměru $\frac{n}{m} = \frac{1}{2}$ není dostačující, neboť by se mohlo stát, že bychom nafukovali a zfukovali hned po sobě.

Verze s Insert a Delete

Udržujeme $\frac{n}{m} \in \langle\frac{1}{4}, 1)$

při zfukování tabulku půlíme, při nafukování zdvojnásobujeme

Po přehashování je poměr $\frac{n}{m} = \frac{1}{2} \Rightarrow$mezi dvěma přehashováními nastane minimálně $\frac{m}{2}$ Insertů či $\frac{m}{4}$ Deletů, tedy alespoň $\frac{m}{4}$ operací.

Zároveň zabereme vždy lineárně prostoru vzhledem k n.

VĚTA: *Nafukovací hashování (se separovanými řetězci) za předpokladu rovnoměrné pravděpodobnosti pracuje v průměrném čase $\mathcal{O}(1)$ na operaci a v prostoru $\mathcal{O}(n)$ (kde n je počet přítomných prvků).*

Kde přijít k rovnoměrné hashovací funkci?

Dědečkovo vyprávění:

a) $h(x) = x\ mod(m)$ -rychlá, snadná kolize, sudé x vždy v sudých přihrádkách. Na náhodných číslech funguje v pohodě.

POZOROVÁNÍ: $\alpha \in (0, 1)$ iracionální;

$x \in N$ pak $\alpha x\ mod(1)$ se chová "náhodně".

b) $h(x) = \lfloor m(\alpha x\ mod(1))\rfloor$

$$\frac{A}{W} \sim \alpha \Rightarrow h(x) = \lfloor m(Ax\ mod(W))\rfloor$$

Jako velmi výhodné se jeví používat $\alpha = \frac{\sqrt{5}-1}{2}$ - převrácená hodnota zlatého řezu.

c) funkce k hashování řetězců

$$h(\emptyset) = 0$$

$$h(x_1, \ldots, x_n) = c(h(x_1, \ldots, x_{n-1}) + x_n)$$

$$\Rightarrow h(x_1, \ldots, x_n) = x_n + cx_{n-1} + c^2 x_{n-2} + \ldots$$

(polynom)

d)Pokud x je k-tice (celých) čísel. Pak můžeme jako hashovací funkci použít skalární součín s náhodně vybraným prvkem z vektorového prostoru Z_p^k (nad konečným tělesem):

$$h(x) = \overrightarrow{a} * \overrightarrow{x} mod(p)$$

Pravděpodobnost přes všechny volby $\overrightarrow{a}$, že x, y se zahashují stejně je pak:

$$p[h(x) = h(y)] = p[\overrightarrow{a} * \overrightarrow{x} = \overrightarrow{a} * \overrightarrow{y}] = p[\overrightarrow{a} * (\overrightarrow{x} - \overrightarrow{y}) = 0)]$$

Pokud platí $p[\overrightarrow{a} * (\overrightarrow{x} - \overrightarrow{y} = 0)]$, pak $\overrightarrow{a}$ je prvkem ortogonálního doplňku k $(x - y)$, ten má $dim(W^{\perp}) = (k - 1)$

$$p = \frac{W^{\perp}}{Z_p^k} = \frac{p^{k-1}}{p^k} = \frac{1}{p} = \frac{1}{m}$$

kde m je počet přihrádek. Což je dostačující.