12. Hashování

V předchozích kapitolách jsme se zabývali tříděním prvků a zjistili jsme, že v obecném případě (když smíme prvky pouze porovnávat a prohazovat) nelze třídit rychleji než v čase $\Theta(n \log n)$. Zároveň jsme došli k tomu, že pokud víme o prvcích něco více (např. jejich rozsah), tak jsme v některých případech schopni třídit i s lineární časovou složitostí.

V praxi nicméně většinou netřídíme samotná čísla, ale spíše nějaké položky, např. určené uspořádanou dvojicí (klíč, hodnota), kde klíče jsou unikátní a existuje na nich lineární uspořádání. Ukázali jsme si, jak takovouto množinu udržovat (aby se v ní dalo rychle vyhledávat, zařazovat i mazat) ve vyhledávacích stromech. Dokázali jsme, že pokud budeme udržovat strom dostatečně vyvážený, tak budou jednotlivé operace (*Find*, *Insert* a *Delete*) trvat $\Theta(\log n)$.

Nešlo by to ale rychleji? Odpověď je „jak kdy“. Podívejme se na další možnosti. Nejdříve si ale připomeňme základní pojmy, které budeme používat.

Klíče budou prvky universa, což je množina $\{0, \ldots, U-1\}$, kterou budeme označovat $[U]$. Tuto množinu budeme chtít udržovat v nějaké šikovné datové struktuře, abychom v ní mohli rychle vyhledávat, zařazovat i mazat. Dále n bude značit (maximální) počet prvků, které si budeme v jednu chvíli pamatovat.

1. Pole *indexované od* 0 *do* $U-1$

Pokud bude velikost universa dostatečně malá, tak si můžeme položky pamatovat v poli o velikosti universa. Operace *Find*, *Insert* i *Delete* pracují v čase $\mathcal{O}(1)$, ale platíme za to paměťovou složitostí $\Theta(U)$. Když uvážíme, že si budeme pamatovat mnohem méně prvků, než je velikost universa, tak je to velké plýtvání.

2. Číslicový strom

Zvolíme vhodný základ soustavy b, klíč zapíšeme v soustavě o základu b a zápis uložíme do stromu. Tímto nám vznikne „b-ární strom“, tedy strom, jehož každý vrchol má až b synů.

Na první hladině se větvíme podle první číslice, na druhé podle druhé, atd. Na poslední hladině bude tedy n listů. Hloubka tohoto stromu je $\lceil \log_b U \rceil$.

Operace *Find* bude trvat $O(\log_b U)$ (najití jednoho klíče odpovídá projití jedné cesty ve stromě od kořene do listu). Stejný čas budou trvat i operace *Insert* a *Delete*. (Při vkládání zakládáme vrcholy, které odpovídají našemu klíči a které ještě nejsou založené. Při mazání smažeme list a následně vrcholy, ze kterých nic nevede. Pro tuto operaci je výhodné si ve vrcholech udržovat jejich stupně.) Strom zabere prostor velikosti $O(nb \log_b U)$.

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

Příklad: Představme si, že klíče budou IP-adresy, tedy 32-bitová čísla. Jak zvolit b, aby byla časová i paměťová složitost přijatelná? Zkusme rozdělit IP-adresu na 4 kusy, b bude tedy 256. Strom bude mít hloubku 4. Čas bude v pořádku ($\log_b U = 4$),

ale požadavky na prostor jsou dost vysoké($nb \log_b U = n \cdot 256 \cdot 4 = 1024$). Výhodnější bude rozdělit IP-adresu na 8 částí, b bude tedy 16, hloubka stromu 8 a nároky na čas i paměť jsou přijatelné, neboť $\log_b U = 8$ a $nb \log_b U = n \cdot 16 \cdot 8 = 128$.

4. Hashování

Mějme m přihrádek a v každé z nich spojový seznam. Dále h bude hashovací funkce, která libovolnému prvku z universa přiřadí právě jednu přihrádku.

Kdyby h fungovala „rovnoměrně“ (každému prvku by přiřadila jinou přihrádku, tedy každý spojový seznam by měl jen jeden prvek), tak bychom uměli vyhledávat, přidávat i mazat prvky v konstantním čase. To je ovšem samozřejmě příliš krásné na to, aby to byla pravda.

Problém je, že pokud bude m výrazně menší než U (což chceme, jinak se jedná o normální (obrovské) pole a funkce je identita – tento případ jsme již vyřešili), tak určitě budou existovat prvky, kterým funkce přiřadí stejnou hodnotu. Takovému případu se říká *kolize*. Prvky se pak zařadí do spojového seznamu. Kdyby byla naopak funkce hodně nešikovná a namapovala by nám všechny prvky do jedné přihrádky, tak budeme vždy muset projít (v nejhorším případě celý) dlouhý spojový seznam.

Jak se dá tedy kolizím bránit? Nejvýhodnější by bylo mít dobrou hashovací funkci, která by mapovala prvky pokud možno rovnoměrně. Ale někdy stačí i průměrně dobrá funkce a dostatečně náhodné klíče.

Vsuvka: 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: $m \approx n^2$.

Praktické hashovací funkce

První příklad

Tato hashovací funkce $h : [U] \to [m]$ je definovaná předpisem: $h(x) = x \bmod m$. Pro dostatečně náhodné hodnoty bude fungovat, ale v mnoha případech může být nevhodná. Například je zřejmé, že pro sudé m zachovává paritu. V případě že sudých, resp. lichých klíčů bude mnohem více, vznikne mnoho zbytečných kolizí. Pro takovouto hashovací funkci je nejlepší zvolit za m nějaké prvočíslo.

Druhý příklad

Zvolme si iracionální α z intervalu $(0, 1)$. Funkce bude definovaná následovně:

$$h(x) = \lfloor m(x \cdot \alpha \bmod 1) \rfloor.$$

Tato funkce by měla na x záviset velmi málo, ale nebudeme si to dokazovat.

Implementace: Reálná čísla se samozřejmě implementují dosti náročně (resp. to vůbec nejde), takže v praxi se tento postup aproximuje celočíselně. Místo reálné α si zvolme $A = \lfloor \alpha \cdot 2^w \rfloor$, kde 2^w je šířka slova (w je např. počet bitů integeru). Potom místo výrazu $(x \cdot \alpha \bmod 1)$, který je z intervalu $(0, 1)$, budeme počítat s výrazem $(x \cdot A \bmod 2^w)$. Potom naše funkce vyjde následovně:

$$h(x) = \left\lfloor \frac{m(x \cdot A \bmod 2^w)}{2^w} \right\rfloor$$

Zbývá jen otázka, jak volit α? Osvěčená hodnota je $1/\tau$, což je přibližně 0,618. (τ je hodnota zlatého řezu.) Takto zvolené α dobře rozbíjí aritmetické posloupnosti, ale i toto je pouze představa a nebudeme si to dokazovat.

Hashování řetězců

Vzhledem k tomu, že hashování nijak speciálně nepotřebuje, aby byly prvky čísla, tak se mohou hashovat i řetězce. Ukažme si jeden příklad funkce pro hashování řetězců. Prázdnému řetězci přiřaďme nulu: $h(\varepsilon) = 0$. Řetězci znaků $x_1, \ldots, x_n$ přiřaďme hodnotu:

$$h(x_1, \ldots, x_n) = (h(x_1, \ldots, x_{n-1}) \cdot + x_n) \bmod m.$$

A je zde nějaká konstanta. Aby se využily všechny přihrádky, tak je vhodné zvlit A tak, aby $A^{\mathbb{E}[\textit{délka řetězce}]} \gg m$. Tato funkce využívá podobných vlastností jako lineární konkurenční generátor náhodných čísel, který pomocí vhodných konstant A a B a šířky slova w vyrábí nové náhodné číslo x' z minulého čísla x jako

$$x' = (x \cdot A + B) \bmod 2^w.$$

Věta Pokud jsou $h(x_1), \ldots, h(x_n)$ rovnoměrně náhodné, pak

$$\forall i : \mathbb{E}[\textit{počet prvků v i-té přihrádce}] = \frac{n}{m}.$$

Důkaz: Počet prvků v i-té přihrádce si označme N_i. Definujme si indikátor P_{ij}, který je roven 1, pokud x_j padlo do i-té přihrádky, jinak je roven 0. Vidíme, že počet prvků v i-té přihrádce je součet P_{ij} přes všechna j.

$$N_i = \sum_j P_{i,j}.$$

Dále využijeme mimo jiné větu o linearitě střední hodnoty.

$$\mathbb{E}[P_{ij}] = \frac{1}{m} \cdot 1 + \left(1 - \frac{1}{m}\right) \cdot 0 = \frac{1}{m}.$$

$$\mathbb{E}[N_i] = \mathbb{E}\left[\sum_j P_{i,j}\right] = \sum_j \mathbb{E}[P_{i,j}] = \frac{n}{m}.$$

♡

Důsledek: Pokud bude n přibližně stejně velké jako m, tak v každé přihrádce bude průměrně jeden prvek.