Teorie informace a kódování

L3: dokonalé hašování

Jan Konecny

4. listopadu 2021

First fit decreasing

A.C. Yao, R.E. Tarjan
Storing a sparse table.
Communications of the ACM. 1979 Nov 1; 22(11):606-11.

- Ukládání statických tabulek o n položkách, každá z nich je číslo mezi 0 a $N - 1$.

Pro jednoduchost budeme uvažovat

- číslo N je druhá mocnina nějakého celého čísla, t.j. $N = m^2$ pro nějaké $m \in \mathbb{Z}$.
- že $n \geq \max\{2, m\}$.

Algoritmus *first-fit-decreasing*

- seřadíme řádky dle počtů nenulových prvků v klesajícím pořadí
- hladovým (greedy) způsobem přiřazujeme posuny.

Příklad 1

$$A = \begin{pmatrix} 0 & \star & 0 & 0 & \star \\ \star & \star & 0 & \star & 0 \\ \star & 0 & \star & 0 & 0 \\ 0 & 0 & 0 & \star & 0 \\ \star & 0 & 0 & 0 & \star \end{pmatrix}$$

$$\begin{array}{ccccccccccc} & 0 & \star & 0 & 0 & \star & & & & & & r(1) = 1 \\ \star & \star & 0 & \star & 0 & & & & & & & r(2) = 0 \\ & & & & \star & 0 & \star & 0 & 0 & & & r(3) = 4 \\ & & & & & 0 & 0 & 0 & \star & 0 & & r(4) = 5 \\ & & & & & & & \star & 0 & 0 & 0 \quad \star & r(5) = 7 \\ \\ 5 & 6 & 1 & 8 & 10 & 4 & 12 & 20 & 18 & 0 & 0 \quad 24 & \end{array}$$

Krok 1:

for $i \leftarrow 1$ *to* m **do**
  count(i) $\leftarrow 0$;
  list(i) $\leftarrow \emptyset$
end
for *each nonzero position* (i, j) *in* A **do**
  count(i) $\leftarrow$ count(i)+1;
  list(i) $\leftarrow$ list(i) $\cup\{i\}$
end

Po kroku 1:

- list(i) je seznam nenulových sloupců v řádku i
- count(i) je jejich počet.

Krok 2:

for $l \leftarrow 0$ *to* n **do**
 | bucket(l) $\leftarrow \emptyset$
end
for $i \leftarrow 0$ *to* m **do**
 | bucket(count(i)) $\leftarrow$ bucket(count(i)) $\cup\{i\}$
end

Krok 2 je radix sort řádků podle jejich počtu nenulových prvků.

Krok 3:

```
for k ← 0 to n + m do
    entry(k) ← false
end
for l ← n downto 0 do
    for each i in bulcket(l) do
        r(i) ← 0 ;
        check for overlap: for each j in list(i) do
            if entry(r(i) + j) then
                r(i) ← r(i) + 1;
                goto check overlap
            end
        end
        for each j in list(i) do
            entry(r(i) + j) ← true
        end
    end
end
```

Pokud je rozdělení nenulových prvků v řádcích A „přiměřeně rovnoměrné“, pak *first-fit-decreasing* účinně komprimuje A.

Toto empirické pozorování můžeme ověřit analýzou metody.
Intuitivně:

- řádky s pouze několika nenulovými prvky neblokují příliš mnoho možných hodnot posunutí pro jiné řádky,
- problémy způsobují pouze řádky s mnoha nenulovými prvky.

Abychom tento jev kvantifikovali, definujeme

- $n(l)$, pro $l \geq 0$ jako celkový počet nenulových prvků v řádcích s více než l nenulovými prvky.

Nálsedující věta ukazuje, že pokud $\frac{n(l)}{n}$ klesá dostatečně rychle s rostoucím l, *first-fit-decreasing* funguje dobře.

Věta 2

Uvažujme že pole A splňuje podmínku harmonického rozpadu:

$$\text{pro libovolné } l,\ n(l) \leq n/(l+1).$$

Pak každý posun $r(i)$ vypočítaný pro A metodou first fit splňuje $0 \leq r(i) \leq n$.

Věta 2

Uvažujme že pole A splňuje podmínku harmonického rozpadu:

$$\text{pro libovolné } l,\ n(l) \leq n/(l+1).$$

Pak každý posun $r(i)$ vypočítaný pro A metodou first fit splňuje $0 \leq r(i) \leq n$.

Důkaz.

- Pro libovolný řádek i, uvažujme výběr posunu $r(i)$.
- Uvažujme, že $r(i)$ obsahuje $l \geq 1$ nenulových prvků.
- Dle harmonického rozkladu je počet nenulových prvků nejvýše $\frac{n}{l}$.
- Každý takový nenulový prvek může blokovat nejvýše l výběrů pro $r(i)$.
- Dohromady může být blokováno nejvýše n výběrů, a tedy $0 \leq r(i) \leq n$. □

Význam harmonického rozpadu:

Pokud má A harmonický rozpad,

- musí být alespoň polovina nenulových prvků v A v řádcích pouze s jedním nenulovým prvkem.

- žádný řádek nesmí mít více než $\sqrt{n}$ nenulových prvků.

Pokud A nemá harmonický rozklad, pak *first-fit-decreasing* neudělá dobrou kompresi – aspoň ne v nejhorším případě.

A protože nás zajímá i nejhorší případ, meli bychom se podívat, jak se s ním srovnat.

Potřebujeme najít způsob, jak „uhladit“ rozložení nenulových prvků v řádcích A, než začneme počítat posuny řádků.

A to tak, že v A posuneme sloupce:

- každému sloupci přiřadíme posun $c(j)$ zobrazující každou pozici (i, j) do nové pozice $(i + c(j), j)$.
- to transformuje A do nového pole B s více řádky – konkrétně $\max_j(c_j) + m$ – a se stejným počtem sloupců.

Příklad 3

$$A = \begin{pmatrix} 0 & \star & 0 & 0 & \star \\ \star & \star & 0 & \star & 0 \\ \star & 0 & \star & 0 & 0 \\ 0 & \star & 0 & \star & 0 \\ \star & 0 & 0 & 0 & \star \end{pmatrix} \quad \Rightarrow \quad B = \begin{pmatrix} 0 & 0 & 0 & 0 & \star \\ \star & 0 & 0 & 0 & 0 \\ \star & 0 & \star & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \star & 0 & 0 & 0 & \star \\ 0 & \star & 0 & 0 & 0 \\ 0 & \star & 0 & 0 & 0 \\ 0 & 0 & 0 & \star & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \star & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$c(1) = 0, c(2) = 5, c(3) = 0, c(4) = 6, c(5) = 0$$

Potřebujeme kritérium pro výběr posunů sloupců.
Tím kritériem bude podmínka exponenciálního rozpadu definovaná následovně:
Nechť

- B_j je pole sestávající z prvních j posunutých sloupců z A.
- n_j je celkový počet nenulových prvků v B_j.
- $n_j(i)$ pro $i \geq 0$ je celkový počet nenulových prvků v B_j, které se objevují v řádcích B_j obsahující více než i nenulových prvků.

$$E_j\text{: Pro libovolné } i > 0,\ n_j(i) \leq n_j/2^{i(2-n_j/n)}$$

Vybíráme sloupcove posuny tak, aby splňovaly E_j pro všechna j s tím, že používáme *first-fit* následovně:

- Vypočítáme postupně – od 1 do m.
- Vybereme jako posun $c(j)$ pro sloupec j nejmenší hodnotu, t.ž. B_j splňuje E_j.

„Univerzální“ hašování

Problém:

- mod je docela pomalá operace

Dietzfelbinger M, Hagerup T, Katajainen J, Penttonen M.
A reliable randomized algorithm for the closest-pair problem.
Journal of Algorithms. 1997 Oct 1;25(1):19-51.

- Navržena technika multiply-shift

Multiply-shift

Předpokládáme

- $m = 2^M$.
- w je počet bitů ve slově.
- Hašovací funkce jsou parametrizovány lichými kladnými čísly $a < 2^w$ (které se vlezou do slova o w bitech)

$$h_a(x) = (a \cdot x \mod 2^w) \div 2^{w-M}$$

Neboli, zapsáno v jazyku C:

$$h_a(x) = \texttt{(size\_t) (a*x) >> (w-M)}$$

Příklad 4

$h_{111}(80) = \ldots$

$a \cdot x$:

$111 \cdot 80 = 8880 = 10001010110000_2$

$(a \cdot x) \mod 2^w$:

$10001\mathbf{0110000}_2 = 176$

pro $M = 4$:

$((a \cdot x) \mod 2^8) \div 2^{8-4}$:

$\mathbf{1011}0000_2 = 11$

- Až 4× rychlejší výpočet.
- Toto schema **ne**-splňuje univerzálnost.
- Splňuje volnější podmínku

$$\Pr\{h_a(x) = h_a(y)\} \leq \frac{2}{m}$$

- To je tzv. 2-univerzálnost.

Univerzální hashování vektorů

Hashujeme $\overline{x} = (x_0, \ldots, x_k)$, x_i jsou celá čísla každé má w bitů.

- Pokud $\mathcal{H}$ je univerzální, pak také třída hashovacích funkcí

$$h(\overline{x}) = \left(\sum_{i=0}^{k-1} h_i(x_i) \right) \mod m,$$

  kde $h_i \in \mathcal{H}$ jsou vybrány náhodně a nezávisle, je univerzální.

- Též možno použít multiply-shift:

$$h_{\overline{a}}(\overline{x}) = \left((\sum_{i=0}^{k-1} x_i \cdot a_i) \mod 2^{2w} \right) \div 2^{2w-M},$$

  kde $\overline{a} = (a_0, \ldots, a_{k-1})$ jsou náhodná lichá celá čísla na $2w$ bitech.

Univerzální hashování řetězců

Předpokládejme, že $\overline{x} = (x_0, \ldots, x_l)$, kde neznáme horní hranici pro l. Znaky řetězce $\overline{x}$ budeme uvažovat jako koeficienty polynomu modulo velké prvočíslo p.

$$h_a(\overline{x}) = h_{\text{int}} \left((\sum_{i=0}^{l} x_i \cdot a^{l-i}) \mod p \right)$$

kde $a \in \{1, \ldots, p-1\}$ je náhodné číslo a h_{int} je vybrána z univ. třídy hašovacích funcí na hašování celých čísel $\{0, \ldots, p-1\} \to \{0, \ldots, m-1\}$.

$h \leftarrow$ `INIT_VAL`;
for $i \leftarrow 0$ **to** $x.length$ **do**
 | $h \leftarrow ((h \cdot a) + x[i]) \mod p$
end
return h

V praxi můžeme $\bmod$ a p vypustit, a povolit přetečení integeru, protože to je v mnoha programovacích jazycích ekvivalentní:
$\bmod$ (Max-Int-Value $+ 1$)

Hodnoty pro inicializaci h v některých implementacích:

- djb2: INIT_VAL = 5381, a=33
- STLPort 4.6.2: INIT_VAL = 0, a=5
- `java.lang.String.hashCode()`: INIT_VAL = 0, a=31

Hash and displace

def `Construct(`B, y`)`**:**

input : S – set of keys; b – number of bins; $\varepsilon > 0$

$n \leftarrow |S| \cdot (1 + \varepsilon)$

if $b < (2 + \varepsilon) \cdot n$ **then**

return -1

end

repeat

Pick f at random in UTHF with range $\{0, \ldots, n-1\}$

Pick g at random in UTHF with range $\{0, \ldots, b-1\}$

for $i \leftarrow 0$ **to** $b - 1$ **do**

$B[i] \leftarrow \{s \in S \mid g(s) = i\}$

Find permutation P s.t. $|B[P[i]]| \geq |B[P[i+1]]|$ for $i = 0, \ldots, b-2$

$m = max\{i | B[P[i]] > 1\}$

end

until $|B[P[0]]|^2 + \cdots + |B[P[m]]|^2 \leq \frac{n}{1+\varepsilon}$ *and* f *is 1-1 on each* $B[i]$

for $i \leftarrow 0$ **to** $n-1$ **do**
  $T[i] \leftarrow$ `false` (marks unoccupied slots of the table);
end
for $i \leftarrow 0$ **to** m **do**
  repeat
    Pick $d \in 0, \ldots, n-1$ at random
  until $T[f(s)+d) \mod n]$ *is* `false` *for each* $s \in B[P[i]]$;
  $D[P[i]] \leftarrow d$;
  for each $s \in B[P[i]]$ **do**
    $T[f(s)+d) \mod n] \leftarrow$ `true`
  end
end
Set $E[1], \ldots, E[k]$ to distinct values where $T[E[i]] \leftarrow$ `false`;
for $i \leftarrow m+1$ **to** $m+k$ **do**
  Take (the unique) $s \in B[P[i]]$;
  $D[P[i]] \leftarrow (E[i-m] - f(s)) \mod n$
end
return(f, g, D, n)