Union Find (część II)

$\sigma = op_1, \dots, op_k$

$op_i \in \{union, find\}$

$T(union) \in O(1)$

$T(find) \sim$ koszt ścieżki, którą przejdziemy
- koszt rozkładany między operacje i wierzchołki

Rozpatrzmy $\sigma' = \sigma \setminus \{find\}$

Z tego mamy drzewo, na podstawie niego określony rząd wierzchołków (wysokość poddrzewa)

Grupa rzędu: $f(i) = 2^{f(i-1)}$

$\log^* n = \min\{k \mid f(k) \geq n\}$

Rząd r należy do grupy $\log^* r$

r	$\log^* r$
0	0
1	0
2	1
3	2
4	2
5	3
⋮	
16	3

przypomnienie

Find płaci za koszt, dzieci kosztują i tam, gdzie zmienia (ojciec w innej grupie) się grupa rzędu $\rightarrow 2 + \log^* n = O(\log^* n)$

Koszt findów: $\underbrace{\text{koszt na operacjach}}_{O(n \log^* n)} + \underbrace{\text{koszt na wierzchołkach}}_{\sum_{g=0}^{\log^* n} \text{koszt na wierzchołkach w grupie } g}$

Koszt na wierzchołkach w grupie g:

Ile jest wierzch. o rzędach w grupie g?

Rzędy w grupie g: $\{f(g-1)+1 \,..\, f(g)\}$

$N(r) \leq \frac{n}{2^r}$ - # w o rzędzie r

W grupie g:

$$\sum_{r=f(g-1)+1}^{f(g)} N(r) \leq \sum_{r=f(g-1)+1}^{f(g)} \frac{n}{2^r} \leq \frac{n}{2^{f(g-1)+1}} \underbrace{\left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right)}_{\leq 2} \leq \frac{n}{2^{f(g-1)}} = \frac{n}{f(g)}$$

Ile razy możemy obciążyć wierzchołek?

Za każdym razem, kiedy obciążamy wierzchołek, jego nowy ojciec będzie miał wyższy rząd niż poprzedni.

Kiedy jego ojciec będzie w innej grupie, już nigdy go nie obciążymy

} dalej działa

Obserwacja

Wierzchołek o rzędzie r w grupie g nie może być obciążony więcej razy, niż grupa g ma rzędów $(f(g) - f(g-1))$

Zatem koszt na wierzchołkach w grupie g:

$$\frac{n}{f(g)} \cdot (f(g) - f(g-1)) \leq n$$

Zatem

$$\sum_{g=0}^{\log^* n} \text{koszt na wierzchołkach w grupie } g \leq (\log^* n) \cdot n$$

można lepiej oszacować

A co gdyby podwieszać wierzchołek pod jego dziadka, a nie korzeń?

Haszowanie

$f: U \to \underline{m}$ $\qquad \underline{m} = \{0, \dots, m-1\}$

$|U| \geq |\underline{m}| \Rightarrow$ konflikty

- i to wielokrotne $\to$ find $\Omega(n)$

1) f - losowa

- oczekiwana liczba kluczy w poli: 1
- ale nie to nas interesuje!
- maksymalna liczba kluczy w którymkolwiek poli?

$$\frac{\lg \lg n}{\lg \lg \lg n}$$

Przykłady funkcji haszujących:

$h(k) = n \bmod m \qquad m \neq 2^l \quad m \neq 10^l$

p - l. pierwsza niezbyt blisko potęgom 2-ki

$h(k) = \lfloor m - (k \cdot A - \lfloor k \cdot A \rfloor) \rfloor$

$A \in \mathbb{R}$ - stała, np. $\approx \frac{\sqrt{5}-1}{2}$

- hint: testować swoje funkcje haszujące (tak na oko)

Struktura słownika

$k \in S \rightsquigarrow h(k)$

- Listy elementów (metoda naiwna)

$H[k]$ - lista elementów e t., że $h(e) = k$

Co kiedy H jest za mała? (operacje stają się za drogie)

- tablicę H rozszerzamy dynamicznie
- koszt przehaszowania do nowej tablicy: stały zamortyzowany

Dobra funkcja haszująca powinna spełniać:

(abpHash) $\forall j \in [0..m-1]. \sum_{h(k)=j} P_r(k) = \frac{1}{m}$

↑ ppb, że k będzie argumentem operacji słownikowej

Fakt

Przy założeniu o (abpHash) średni czas operacji find dla haszowania z nalewaniem wynosi $\Theta(1 + \frac{n}{m})$, gdzie

n - l. elementów w słowniku
m - rozmiar tablicy H

(Struktura słownika)

- adresowanie otwarte

Klucze bezpośrednio w H, jeden w komórce

Rozwiązywanie kolizji

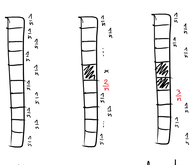
1) $h: U \times \underline{m} \to \underline{m}$

↑ numer próby
(m, bo za każdą próbą sprawdzamy w inne miejsce)

- metoda liniowa

$h'(k, i) = (h(k) + i) \bmod m$

- będą tworzyć się zajęte obszary



- komplikacja przy usuwaniu elementów (trzeba pamiętać, że było zajęte)

- metoda kwadratowa

$h'(k, i) = (h(k) + c_1 i + c_2 i^2) \bmod m$

Wymaganie: pojście każdego klucza:

$(\lambda x. c_1 x + c_2 x^2)[\underline{m}] = \underline{m}$

- też tworzą się zlepki, ale jest lepiej

↑ (takie działanie)

- podwójne haszowanie

$h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod$

Intro: dlaczego ♯ jest lepsze
; 2 fajne rzeczy