Tabulky s rozptýlenými položkami, vyhledávání v tabulkách

Z FAV wiki

To samé jako PPA2 14 a 15,

jen doplňky/opakování/upřesnění:

Vnitřní zřetězení - synonyma (položky s kolizemi) se ukládají do volných míst

Vnější zřetězení - synonyma se ukládají do seznamů pod klíči

Obsah

Otevřené rozptýlení - hash funkce

Kolize:

1) nedefinované chování při kolizi - nutno dodefinovat

2) lineární posun - při kolizi vkládáme na první volné místo lineárně (na indexech 1,2,3,4 od původní polohy

3) kvadratický posun - při kolizi hledáme první volné místo na indexech podle kvadratické funkce (vzdálenosti 1, 4, 9, 16...)

4) opakovaný hash - hashovací fci použijeme víckrát pokud dojde ke kolizi

Uzavřené rozptýlení - přímé adresování

metody 2) a 3) nejsou vhodné, vzniká shlukování (clustering), proto je dobré použít způsob 4) nebo lepší hash funkci

Tabulky s uzavřeným rozptýlením a vnějším zřetězením

= kombinace obou

- adresujeme přímo, při kolizi si ale do tabulky na konec uložíme, kam budeme vnitžně zřetězovat. Hash funkce tedy může být při kolizi (třeba náhodná), nebo tabulku rozdělíme na 2 části, jednu pro vyhledávání klíčů, a druhou pro ukládání synonym. ta už může být dynamicky alokovaná.

Metoda rozptýlených indexů

To co bylo v předchozím případě v tabulce vyjmeme, a vytvoříme vedle. Uděláme si mapu synonym (synonyma v seznamu pro každý klíč) a tyto pak ukazují na index do pole.

Vyhledávání

Má samozřejmě smysl jen pokud nemáme orákulum (= hashovací fci). Pokud orákulum máme, není třeba hledat protože ideální orákulum vždy ví, kam se podívat pro správná data (The all-knowing Oracle is never surprised. How can she be, she knows everything. -- The Matrix: Revolution), reálné ví aspoň kde hledat.

Jelikož je při přímém adresování orákulum pouhý index do tabulky (mno, to moc orákulum není, orákulum vždy ví, kam jít, aby nebyla kolize), alespoň nám napoví odkud hledat.

Sekvenční:

Hledáme klíč lineárně, pokud na něj narazíme, vrátíme hodnotu

+ snadná implementace

+ tabulka nemusí být uspořádaná

- pomalé hledání

Binární

Jako sekvenční, ale klíče musí být seřazeny, a nad nimi vytvoříme BVS, tedy hledáme vždy v půlce intervalu

+ rychlejší (O(log n))

- nutnost udržovat tabulku seřazenou

Fibonacciho

Jako binární, ale nerozdělujeme v polovině, ale podle Fibonacciho posloupnosti (tedy ne 1:1 -> 1:(1:1) -> 1:((1:1): 1) ale (1:1) -> (1: (1:2)) - > (1:((2:3):2))

Prvky vybíráme jako dvojice, první z dvojice je ve Fibonacciho posloupnosti na pozici odpovídající úrovni vnožení stromu (výška větve, ve které hledáme)

Fibonacciho ppst: a1 = 1, a2 = 1, ai = ai-1 + ai - 2 , tedy 1,1,2,3,5,8,13,21,34,55, ...

Podle sekundárního klíče

Jen pouze pokud sekundární klíč existuje v datech, např seznam jmen, jméno je primární (není, ale pro vysvětlení řekněme, že je) a rok je sekundární. Vyhledáme jména v sekundárním klíči a v nich pak hledáme jméno podle primárního klíče.

Sekundární klíče máme v seznamu, který obsahuje seznam primárních klíčů zabulky pro každý sekundární klíč
