

ADT Tabulka

Tabulka

- Datová struktura, která umožňuje vkládat a později vybírat informace podle identifikačního klíče. Mohou být:
 - pevně definované (LUT Look Up Table)
 - s proměnným počtem položek

Konvence:

k – klíč, kterým identifikujeme položku

A_k – adresní klíč tj. „adresa položky“ (ve většině případů je to index)

Hledání v tabulkách – parametry:

S – délka hledání položky (počet položek, které je nutno prozkoumat)

T – průměrná délka hledání (m je počet přístupů do tabulky)

$$T = \frac{1}{m} \sum_{i=1}^m S_i$$

A – průměrná délka prohledávání za předpokladu rovnoměrného přístupu (n je počet obsazených položek tabulky)

$$A = \frac{1}{n} \sum_{i=1}^n S_i$$

P – tzv. plnění tabulky (podíl obsazených položek)

$$P = \frac{n}{p}$$

kde p je velikost tabulky

Rozdělení tabulek podle způsobu organizace

- tabulky s přímým přístupem
- obyčejné vyhledávací tabulky
- tabulky se sekvenčním přístupem
- tabulky s rozptýlenými položkami

Tabulky s přímým přístupem

- $k \rightarrow Ak$ je prosté zobrazení, každá položka tabulky má své místo jednoznačně určené hodnotou Ak přímo odvozenou z k
- Optimální implementace tabulky je pomocí pole – indexy jsou přímo klíče v tabulce

$$S=T=A=1$$

- Výhody:
 - rychlý přístup
 - jednoduchá implementace
- Nevýhody:
 - Velikost tabulky je daná rozsahem klíče, pro praktické účely bývá většinou neúnosná
 - Řídké pole – nerovnoměrný počet klíčů vzhledem k rozsahu tabulky.
- Příklady:
 - Telefonní síť – klíčem je telefonní číslo uživatele
 - Telefonní seznam – klíčem je jméno

Vyhledávací tabulky

- Vyhledává se podle hodnoty klíče
- Pořadí položek může být:
 - definované (uspořádané)
 - náhodné
- Strategie vyhledávání:
 1. sekvenční
 2. binární
 3. Fibonacciho
 4. kombinované
- Výhoda:
 - plnění vyhledávacích tabulek může být až 100%
- Nevýhoda
 - časově náročné vyhledávání (může být až lineární)

Sekvenční vyhledávání

- Položky v tabulce mohou být neuspořádané
- Princip spočívá v postupném porovnávání klíčů položek s hledaným klíčem až do nalezení shody, popř. nalezení konce tabulky

$$A = \frac{1}{2}(n + 1)$$

- **Výhoda:** snadná implementace a časově nenáročná modifikace tabulky (implementace jako pole (popř. jako seznam), přidává se na konec pole, díry po odebrání prvků se vyplňují posledním prvkem – v případě implementace polem)

Binární vyhledávání

- lze použít v případě, že jsou položky tabulky seřazeny podle hodnoty klíče

Princip: porovnat hledaný klíč s klíčem uprostřed tabulky, pokud není shoda, hledat v levé popř. v pravé polovině tabulky (v závislosti na hodnotě klíče)

$$A \approx \log(p)$$

Fibonacciho vyhledávání

Princip: stejný jako u binárního vyhledávání, testované prvky však nevolíme uprostřed, ale v poměru Fibonacciho čísel

Složitost: stejná jako u binárního vyhledávání, ale prvky na začátku jsou nalezeny rychleji

- Pro efektivní hledání se snažíme, aby tabulka měla $F_n - 1$ prvků, kde F_n je určité Fibonacciho číslo

Fibonacciho posloupnost:

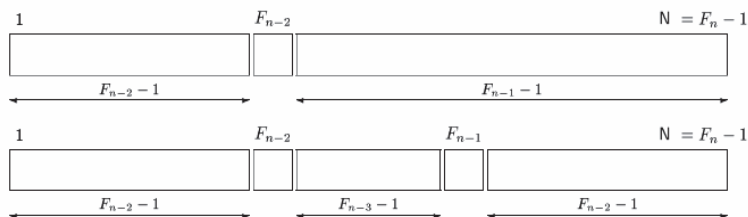
$$F_1 = 0, \quad F_2 = 1,$$

$$F_n = F_{n-1} + F_{n-2} \quad \forall n \in \mathbf{N}, n > 2$$

Prvky Fibonacciho posloupnosti:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Př. Nalezení prvku na pozici 26 v tabulce o 54 prvcích.



Vyhledání:

1		
2		
3		
...		
①	21	
③	26	
②	34	
...		
54		

Hledání podle sekundárního klíče

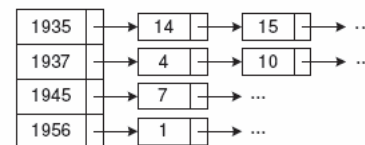
- Invertovaný soubor (indexovaný soubor)
 - jedná se o tabulku seřazenou podle sekundárního klíče, data tvoří primární klíč (nebo přímo index v původní tabulce)

1935	Fiala Jan
1935	Fiala Martin
1937	Bláha Josef
1945	Bláha Jan
1956	Fiala Petr

- invertovaný seznam
 - invertovaný soubor implementovaný jako zřetězený seznam

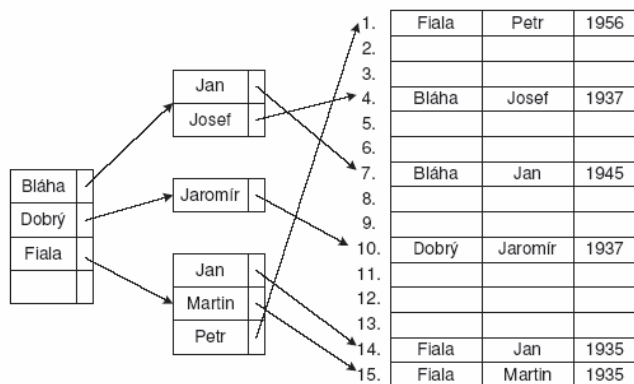


nebo



Vícerozměrné vyhledávání

- vyhledávání podle více klíčů realizujeme vícenásobným přístupem



tabulky s rozptýlenými položkami

- používají se v případě, že rozsah klíče $N \gg$ rozsah tabulky
- pro určení pozice v tabulce, na kterou máme uložit položku s klíčem k , používáme rozptylovací funkce (hash-funkci)

$$A_k = h(k),$$

která klíči k jednoznačně přiřazuje klíč A_k

- může se stát, že pro různé položky $k_1 \neq k_2$ platí, že $h(k_1) = h(k_2)$ – tzv. synonymické položky (dochází ke kolizi)

Při návrhu a implementaci hash-tabulky je nutné vzít v úvahu:

- jak definovat rozptylovací funkci
- jak řešit ukládání synonymických položek

Požadavky na rozptylovací funkci:

1. Pro každé k je jednoznačně definovaná (a v přijatelném čase vyčíslitelná)
2. Vytváří minimální počet kolizí (minimum synonym)
3. Pravděpodobnostní rozdělení $A_k=h(k)$ na intervalu $<0, p-1>$ je rovnoměrné - lze využít pseudonáhodné funkce (*randomizační funkce*)

Realizace $h(k)$:

hash funkci $i=h(k)$ je možné realizovat následujícími způsoby:

i je částí k

i je částí operace nad k

i je zbytkem po dělení rozsahem tabulky p

i je zbytkem po dělení N , N je nejbližší menší prvočíslo než hodnota p

i je dán váhovým součtem částí

$$i = \sum_{i=1}^r a_i x^i$$

kde a_i jsou váhy jednotlivých částí k^i klíče k

Tabulky s otevřeným rozptýlením

Každá pozice tabulky je potenciálně přístupná položce s libovolným klíčem, snadno vznikají shluky položek. Při kolizi se hodnota A_k přepočítá.

- rozptylovací funkci volíme například

$$h = h \bmod p$$

kde p je prvočíslo

- vhodné pro tabulky s velkým rozsahem klíčů, ale malým počtem položek

Podle způsobu řešení kolizí rozeznáváme čtyři podtypy:

1. Tabulky s otevřeným rozptýlením a *nedefinovaným způsobem* ukládání synonymických položek

- nutno dodefinovat způsob přepočítání A_k při kolizi

Příklad:

32	0	
	1	
	2	???
38	3	
	4	
	5	

Kolize: prvky 32 a 38 mají stejnou hodnotu rozptylové funkce

2. Tabulky s otevřeným rozptýlením a ukládáním synonymických položek s *konstantním krokem* s (s vícenásobnou hashovací funkcí)

- na klíč k aplikujeme funkci $h_0(k)$, pokud dojde ke kolizi, vypočteme znovu A_k podle $h_1(k)$ atd. až do té doby, než najdeme v tabulce volné místo

$$\begin{aligned}h_0(k) &= h(k) \\h_1(k) &= (h(k) + s) \bmod p \\&\vdots \\h_i(k) &= (h(k) + is) \bmod p\end{aligned}$$

kde p je rozsah tabulky a s je přirozené číslo nesoudělné s p (nesoudělnost zaručí možnost dostat se na každou pozici tabulky)

Příklad: Tabulka s

- 8 pozicemi
- položkami s klíči DAVID, HANA, DANA, HELENA, EMIL, EVA, BOŽENA
- hashovací funkcí

$$h(k) = \text{pořadí 1. písmene v abecedě}$$

S = 1		
0	HELENA	2
1	BOŽENA	1
2		
3	DAVID	1
4	DANA	2
5	EMIL	2
6	EVA	3
7	HANA	1
st. složitost = 12/7		

S = 3		
0		
1	BOŽENA	1
2	HELENA	2
3	DAVID	1
4	EMIL	1
5	EVA	4
6	DANA	2
7	HANA	1
st. složitost = 12/7		

3. Tabulky s otevřeným rozptýlením a ukládáním synonym s *lineární vícenásobnou ukládací funkcí*

$$h_0(k) = h(k)$$

$$h_i(k) = (h(k) + a \cdot i + b) \bmod p$$

4. Tabulky s otevřeným rozptýlením a ukládáním synonym s *kvadratickou vícenásobnou ukládací funkcí*

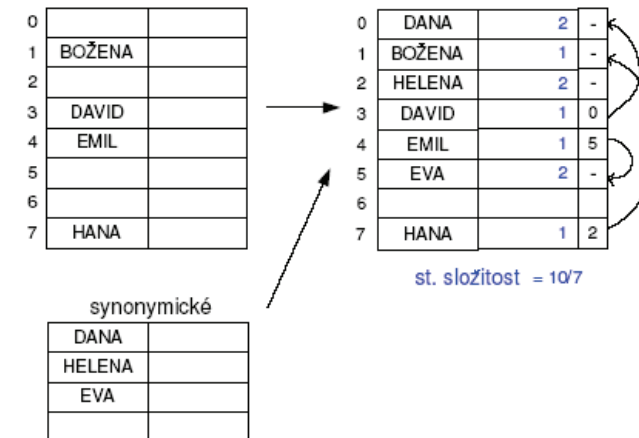
$$h_0(k) = h(k)$$

$$h_i(k) = \left(h(k) + (-1)^i \cdot \left\lceil \frac{i}{2} \right\rceil^2 \right) \bmod p$$

kde $\lceil x \rceil$ značí zaokrouhlení čísla x „nahoru“. U zjišťování zbytku po dělení (*mod*) je třeba dávat pozor na záporné argumenty — držet se definice!

Tabulky s otevřeným rozptýlením a vnitřním zřetězením

Ve fázi zařazování nových položek do tabulky si při výskytu kolize ponecháme synonymické položky vedle. Po zařazení všech položek vřadíme tato dočasně odložená synonyma na zbylá volná místa v tabulce (podle předem zvoleného systému) a příslušná synonyma zřetězíme do tzv. řetězu synonym



Přesun synonym do tabulky:

1. od začátku
 2. od konce
 3. s vícenásobnou hash-funkcí (to je výhodné, protože nám to „nerozbourá“ rovnoměrné rozložení prvků)
- tento typ tabulky je vhodný, pokud tabulku na začátku jednou vytvoříme a potom opakovaně používáme
 - problém je přidání položky na první místo v řetězci synonym, pokud toto místo již jiná položka obsadila.

Tabulky s uzavřeným rozptýlením a vnějším zřetězením

- přinášejí zlepšení poslední metody: rozdělíme tabulku na dvě části
 1. *primární část* — obsahuje pouze položky, které nejsou synonyma
 2. *sekundární část (zóna zřetězení, přeplnění, atd.)* — položky, které jsou synonymy k položkám v primární části

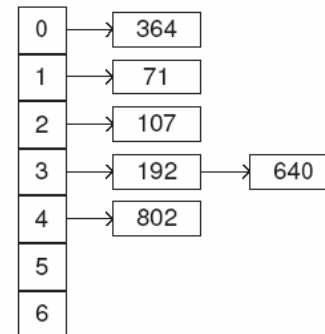
0			-
1	BOŽENA		
2			
prim. část 3	DAVID		8
4	EMIL		10
5			
6			
7	HANA		9
8	DANA		11
sek. část 9	HELENA		-
10	EVA		-
11	DIANA		-

zóna zřetězení

- problémem je vypuštění prvního prvku v řetězci — řeší se zavedením *děr*

Metoda rozptýlených indexů

- tabulku chápeme jako vektor seznamů synonymických položek (řetězců)



- zařazování nových prvků:
 1. na konec řetězu
 2. uspořádaně — složitější implementace, ale přináší jisté urychlení
- velice oblíbená metoda (100% plnění, jednoduchá implementace, ...)