Tablice hashowe

Tablica hashowa jest zwykłą tablicą indeksowaną liczbami, np. od 0 do $N-1$, do której dołączona jest **funkcja hashowa** wyliczająca pozycję w tablicy na podstawie wartości klucza. Jeśli oznaczymy zbiór wszystkich kluczy przez **K**, to funkcja hashowa $\mathrm{h}(k)$ jest postaci:

$$\mathrm{h} : \mathbf{K} \longmapsto 0..N-1$$

Zauważmy najpierw, że w pewnym sensie same klucze mogą służyć jako indeksy do tablicy. Weźmy, na przykład, zestaw bitów tworzących kody wszystkich znaków (liter lub cyfr) wchodzących w skład kluczu. Taki kod bitowy jest liczbą, i będzie różny dla różnych kluczy, tworzy zatem poprawną funkcję hashową.

Własności funkcji hashowych

Bitowe kodowanie kluczy ma poważną wadę — daje wielkie liczby, które wymuszają użycie ogromnych tablic hashowych. W praktyce tak wielkie tablice nie są zapewne potrzebne, więc można stosować uproszczone funkcje hashowe. Jest to wskazane z drugiego punktu widzenia: tablice hashowe mają być niezwykle szybką strukturą danych, wiec obliczanie funkcji hashowej powinno być bardzo prostą operacją arytmetyczną, stałoprzecinkową, operującą na małych liczbach.

Dla wielu prostych przypadków wystarczające jest obliczenie dowolnie zdefiniowanej wartości funkcji hashowej dla klucza, być może o nieco zbyt dużej wartości, i sprowadzenie go do przedziału wyznaczonego wielkością tablicy N, na przykład za pomocą wzoru:

$$\mathrm{h}(k) = \mathrm{kod}(k)\ \mathrm{MOD}\ N$$

To jednak nie wystarcza. Idealna funkcja hashowa powinna być **różnowartościowa**, czyli różnym kluczom przyporządkowywać różne wartości. Zapewnienie tego jest już trudniejsze, dążmy więc do tego, żeby funkcja hashowa generowała wartości jak najbardziej zróżnicowane, prawie przypadkowe. Jednak i w tym wypadku można nie ustrzec się niezamierzonych regularności.

Przykład: notatnik adresowy o pojemności 26 pozycji.

Dlatego też do konstrukcji takiej funkcji wprowadza się elementy zwiększające rozrzut wartości i przypadkowość. Na przykład, jako wielkość tablicy N, przez którą dzieli się wstępny kod klucza, przyjmuje się na ogół liczby pierwsze. Niezależnie jednak od tych podejmowanych środków zmierzających do zróżnicowania wartości funkcji hashowej, konieczne jest sprawdzenie, czy funkcja jednak nie posiada jakiejś regularności.

Funkcje hashowe — użycie wybranych cyfr

Jeśli klucze są liczbami całkowitymi, to wygodną funkcją hashową jest wybranie pewnej grupy cyfr klucza i potraktowanie tak uzyskanej liczby jako wartości funkcji. Na przykład, jeśli chcemy przechowywać dane o pewnej grupie ludzi według ich numerów PESEL, to ostatnie 3 cyfry dadzą, być może, dość przypadkowy i równomierny rozkład wartości:

$$\mathrm{h}(d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9 d_{10} d_{11}) = d_9 d_{10} d_{11}$$

Liczba wybranych cyfr jest związana z rozmiarem tablicy, dla 3 cyfr tablica będzie miała tysiąc elementów. Gdyby zatem liczba elementów do przechowywania była większa wtedy musielibyśmy wybrać więcej cyfr z numeru.

Zwłaszcza w takich kluczach jak numer PESEL istnieją regularności, które trzeba eliminować.

Funkcje hashowe — zwijanie

Gdy liczby zawarte w kluczach albo powstające w procesie wyliczania funkcji hashowej są tak duże, że ich przetwarzanie jest kłopotliwe, można zastosować jedną z technik **zwijania**, to znaczy zmniejszania zakresu liczb. Na przykład:

$$\mathsf{h}(d_1 d_2 d_3 d_4 d_5) = d_1 + d_2 + d_3 + d_4 + d_5$$

Zakres wartości powyższej funkcji wynosi 46 wartości (0 do 45), więc ponieważ zwykle będzie to zbyt mało, możemy zastosować modyfikację powyższego wzoru. polegającą na dodawaniu cyfr grupami, np.:

$$\mathsf{h}(d_1 d_2 d_3 d_4 d_5) = d_1 + d_2 d_3 + d_4 d_5$$

lub

$$\mathsf{h}(d_1 d_2 d_3 d_4 d_5) = d_1 d_2 + d_3 d_4 d_5$$

co daje zakresy odpowiednio 207 i 1098.

Funkcje hashowe — kodowanie znaków

Gdy klucze są znakami lub napisami, można do funkcji hashowych brać kody znaków. Ponieważ jednak napisy są zwykle długie i liczba kodująca pełny napis byłaby bardzo duża, stosuje się zwykle wybieranie znaków, analogiczne do metody wybierania cyfr. Na przykład:

$$\mathsf{h}(z_1 z_2 z_3 z_4 \ldots) = 16384 * \mathsf{ORD}(z_1) + 128 * \mathsf{ORD}(z_2) + \mathsf{ORD}(z_3)$$

Powyższy wzór da równomierny rozrzut dla napisów, których przynajmniej pierwsze trzy znaki rozkładają się równomiernie. Jednak wiele zbiorów kluczy ma inny charakter, na przykład nazwiska, w których pierwszy znak jest dużą literą, a pozostałe znaki małymi. W dodatku nazwiska grupują się wokół pewnych typowych początków, np. „Gor”, „Kra”, „Sta”, a inne są rzadko spotykane, np. „Kwr”, „Mpt”, „Oai”, itd. Co gorsza, nawet wzięcie kodów tylko pierwszych trzech liter daje w powyższym wzorze duże wartości, przekraczające typowe wielkości tablic hashowych, jak również bezpieczny zakres operacji stałoprzecinkowych dla niektórych komputerów.

Funkcje hashowe — dzielenie

Poza przedstawionymi metodami wyboru kodu klucza można zastosować wspomnianą metodę dzielenia (całkowitoliczbowego) w celu sprowadzenia wartości funkcji hashowej do określonego przedziału.

$$\mathsf{h}(k) = \mathsf{kod}(k)\ \mathsf{MOD}\ N$$

```
CONST WielkTabl = 7;

TYPE Pozycja = 0..WielkTabl-1;
     T_Elem = RECORD
                k: INTEGER;
                dane: T_dane;
             END;
     T_TabH = ARRAY[Pozycja]
                OF T_Elem;

VAR tablica: T_TabH;
```

Pozycja	Zawartość
[0]	pusta
[1]	pusta
[2]	pusta
[3]	374, dane
[4]	pusta
[5]	pusta
[6]	pusta

Funkcje hashowe — kombinacje metod

$$\begin{aligned} \mathsf{h}(k) &= (\mathsf{kod}(z_{\text{pierwszy}}) * 676 + \mathsf{kod}(z_{\text{ostatni}}) * 26 + \mathsf{kod}(z_{\text{środkowy}}))\ \mathsf{MOD}\ 251 \\ \mathsf{kod}(z) &= \mathsf{ORD}(\text{mała-litera}(z)) - \mathsf{ORD}(\text{'a'}) \end{aligned}$$

$$\begin{aligned} \mathsf{h}('Stasiak') &= (\mathsf{kod}('S') * 676 + \mathsf{kod}('k') * 26 + \mathsf{kod}('s'))\ \mathsf{MOD}\ 251 = 147 \\ \mathsf{h}('Stoklosa') &= (\mathsf{kod}('S') * 676 + \mathsf{kod}('a') * 26 + \mathsf{kod}('k'))\ \mathsf{MOD}\ 251 = 130 \\ \mathsf{h}('Stachowicz') &= (\mathsf{kod}('S') * 676 + \mathsf{kod}('z') * 26 + \mathsf{kod}('h'))\ \mathsf{MOD}\ 251 = 24 \\ \mathsf{h}('Starczewski') &= (\mathsf{kod}('S') * 676 + \mathsf{kod}('i') * 26 + \mathsf{kod}('z'))\ \mathsf{MOD}\ 251 = 102 \\ \mathsf{h}('Starczynski') &= (\mathsf{kod}('S') * 676 + \mathsf{kod}('i') * 26 + \mathsf{kod}('z'))\ \mathsf{MOD}\ 251 = 102 \end{aligned}$$

Funkcje hashowe — kolizje

Kolizja występuje, gdy dwa różne klucze mają tę samą wartość funkcji hashowej. Nawet dla bardzo dobrych funkcji hashowych trudno jest zwykle udowodnić, że funkcja nigdy nie spowoduje kolizji. Zatem każda implementacja tablic hashowych powinna zawierać w sobie jakąś metodę postępowania w razie wystąpienia kolizji, która pozwoliłaby jakoś przechować oba kolidujące klucze, i oczywiście odnaleźć obydwa w razie potrzeby.

Rozwiązywanie kolizji — metody wolnych pozycji

Prostą techniką w chwili wykrycia kolizji dla klucza na pozycji i w tablicy jest próba umieszczenia go na pozycji $i+1$, albo na pierwszej wolnej pozycji za i. Oczywiście w trakcie wyszukiwania elementu trzeba również, w razie gdyby na pozycji i nie było poszukiwanego elementu, ale był inny, przeszukać fragment tablicy za i. Dopiero po dojściu do pierwszej wolnej pozycji bez znalezienia poszukiwanego elementu możemy mieć pewność, że nie ma go w tablicy.

Pozycja	Zawartość
[0]	pusta
[1]	911
[2]	pusta
[3]	374
[4]	pusta
[5]	pusta
[6]	1091

Pozycja	Zawartość
[0]	pusta
[1]	911
[2]	pusta
[3]	374 $\bullet$
[4]	220 $\hookleftarrow$
[5]	pusta
[6]	1091

Pozycja	Zawartość
[0]	pusta
[1]	911 $\bullet$ $\prec$ 624
[2]	449 $\hookleftarrow$ $\prec$ 624
[3]	374 $\bullet$ $\prec$ 624
[4]	220 $\hookleftarrow$ $\prec$ 624
[5]	pusta $\prec$ 624
[6]	1091

Ten podstawowy wariant techniki ma wadę, określaną mianem „sklejania" (ang. *clustering*). Oczywiście powoduje to, że wyszukiwanie, ani wstawianie elementów do tablicy hashowej, nie jest już natychmiastowe.

Jeżeli łańcuchy „wyrastające" z dwóch lub więcej pozycji tablicy o dużej liczbie kolizji sklejają się ze sobą, to będą już rosnąć wspólnie, wzajemnie się przeplatając. Taki wydłużony łańcuch jest w tablicy hashowej dużym „celem", w który mogą „trafiać" następne klucze, powodując jeszcze szybsze rośnięcie łańcucha. Ulepszone metody wolnych pozycji po wykryciu kolizji nie wykorzystują bezpośrednio następnej wolnej pozycji w tablicy, lecz pozycję oddaloną o pewną odległość, np. kwadratowo rosnącą, albo wyznaczoną przez pomocniczą funkcję hashową.

Inną wadą metod wolnych pozycji jest fakt, że przy usuwaniu elementu z tablicy jego pozycja nie może być po prostu zwalniana, lecz musi być zaznaczona jako poprzednio wykorzystana. W przeciwnym wypadku jakiś inny element może stać się niedostępny przy przeszukiwaniu, ponieważ gdy elementu nie ma na swojej właściwej pozycji, przeszukiwanie trwa tylko do końca łańcucha, tzn. do wystąpienia pierwszej wolnej pozycji.

Rozwiązywanie kolizji — dołączanie elementów na zewnątrz tablicy

Inna grupa metod rozwiązywania kolizji polega na dołączaniu kluczy do pozycji tablicy w postaci list wskaźnikowych.

Pozycja	Zaw.
[0]	NIL
[1]	NIL
[2]	NIL
[3]	NIL
[4]	NIL
[5]	NIL
[6]	NIL

Pozycja	Zawart.
[0]	NIL
[1]	$\rightarrow$ 911
[2]	NIL
[3]	$\rightarrow$ 374
[4]	NIL
[5]	NIL
[6]	$\rightarrow$ 1091

Pozycja	Zawartość
[0]	NIL
[1]	$\rightarrow$ 911 $\rightarrow$ 449 $\rightarrow$ 624
[2]	NIL
[3]	$\rightarrow$ 374 $\rightarrow$ 220
[4]	NIL
[5]	NIL
[6]	$\rightarrow$ 1091

Metody dołączania elementów na zewnątrz tablicy są niezawodne, jednak przy dużej liczbie kolizji sprawność tablicy ulega degradacji podobnie jak przy innych metodach. Nadmiernie wypełniona tablica hashowa przestaje pełnić swoją funkcję.

Tablice hashowe jako abstrakcyjny typ danych

```
FUNCTION UtworzTabliceH( VAR T: TablicaH ) : BOOLEAN;
                                                 EXTERN;
(* Tworzy i inicjuje tablice hashowa w zmiennej T.   *)
(* Istniejaca wartosc T jest ignorowana.  Ta funkcja *)
(* musi byc wywolana przed pierwszym uzyciem tablicy.*)
(* Zwraca wartosc TRUE iff wynik operacji pomyslny.  *)

FUNCTION DodajDoTablicyH
         ( VAR T: TablicaH
         ; E: TypElementu
         ; FUNCTION klucz(e:TypElementu): TypKlucza
         ): BOOLEAN;
                                                 EXTERN;
(* Dodaje element E do tablicy hashowej T.           *)
(* Zwraca wartosc TRUE iff wynik operacji pomyslny.  *)
```



```
FUNCTION ZnajdzWTablicyH
         ( T: TablicaH
         ; K: TypKlucza
         ; VAR E: TypElementu
         ; FUNCTION klucz(e:TypElementu): TypKlucza
         ): BOOLEAN;
                                                 EXTERN;
(* Znajduje w tablicy hashowej T element o kluczu K  *)
(* i, jesli znaleziony, umieszcza element w zmien.E. *)
(* Zwraca wartosc TRUE iff wynik operacji pomyslny.  *)

FUNCTION UsunZTablicyH
         ( VAR T: TablicaH
         ; K: TypKlucza
         ; FUNCTION klucz(e:TypElementu): TypKlucza
         ): BOOLEAN;
                                                 EXTERN;
(* Usuwa z tablicy hashowej element o kluczu K.      *)
(* Zwraca wartosc TRUE iff wynik operacji pomyslny.  *)
```


Podsumowanie operacji na poznanych strukturach danych

Struktura:	Złożoność obliczeniowa operacji:			
	wyszukiwanie elementu	dodawanie elementu	usuwanie elementu	przeglądanie uporządkowane
tablice jednowymiarowe	O(N)	O(1)	O(1)	O(N log N)
tablice uporządkowane	O(log N)	O(N)	O(N)	O(N)
listy wskaźnikowe	O(N)	O(1)	O(1)	O(N log N)
drzewa binarne	O(N)	O(1)	O(1)	O(N log N)
drzewa uporządkowane	O(N)	O(N)	O(N)	O(N)
drzewa AVL	O(log N)	O(log N)	O(log N)	O(N)
B-drzewa	O(log N)	O(log N)	O(log N)	O(N)
stogi	O(N)	O(log N)	O(log N)	O(N log N)
drzewa *trie*	O(1)	O(1)	O(1)	O(N)
tablice hashowe	O(1)	O(1)	O(1)	O(N log N)