Haszowanie

IIUWr. II rok informatyki. *Opracował: Krzysztof Loryś*

1 Wstęp

Haszowanie jest jedną z metod realizacji słowników. Poznaliśmy już m.in. drzewa czerwono-czarne, drzewa AVL, czy B-drzewa - struktury, które umożliwiały wykonywanie operacji słownikowych w czasie proporcjonalnym z logarytmu z wielkości słownika.

Gdy uniwersum jest małe (powiedzmy n elementowe), możemy wykorzystać n elementowe tablice bitowe (i-ty element takiej tablicy jest równy 1 wtedy i tylko wtedy gdy i-ty element uniwersum należy do zbioru; zakładamy przy tym, że umiemy efektywnie numerować elementy uniwersum). Przy takim sposobie pamiętania słownika czas wykonania operacji słownikowych jest stały.

2 Metoda funkcji haszujących

Do pamiętania elementów podzbioru wykorzystywana jest tablica $T[0..m-1]$. Zwykle m jest proporcjonalne do maksymalnej liczności słownika; wielkość uniwersum nie ma tu większego znaczenia. Metoda wykorzystuje funkcję (tzw. *funkcję haszującą*) $h : U \to \{0, ..., m-1\}$, określającą miejsce pamiętania elementów U w T.

2.1 Funkcje haszujące

Dobra funkcja haszująca powinna spełniać następujący warunek:

$$\text{(dfh)} \qquad \forall_{j=0,\dots,m-1} \sum_{k:h(k)=j} P(k) = \frac{1}{m},$$

gdzie $P(k) =$ jest prawdopodobieństwem tego, że $k \in U$ będzie parametrem którejś z operacji słownikowych. W praktyce warunek ten jest zwykle niesprawdzalny, gdyż nie znamy P. Ponadto, jeśli metoda ma być efektywna, funkcja haszująca powinna być szybkoobliczalna.

Nie polecam wymyślania własnych funkcji haszujących (przynajmniej na początku). Lepiej skorzystać ze sprawdzonych funkcji.

Przykłady funkcji haszujących.

- $h(k) = k \bmod m$

  Uwaga. Należy wykazać się ostrożnością w wyborze m. Nie zaleca się m postaci 2^p, gdyż wówczas $h(k)$ jest równe ostatnim p bitom klucza k, a te często mają nierównomierny rozkład. Z tego samego powodu nie zaleca się brać jako m potęg liczby 10. Zwykle dobrymi wartościami m są liczby pierwsze niezbyt bliskie potęgom liczby 2.

- $h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$, gdzie A jest ustaloną liczbą z przedziału $(0, 1)$.

  Uwaga. Teraz wartość m nie ma takiego znaczenia jak poprzednio i zwykle bierze się m równe potędze liczby 2 (ze względu na łatwość mnożenia). Wybór A jest bardziej zależny od cech danych, jednak zwykle A równe $(\sqrt{5}-1)/2 \approx 0.6180339887$ jest dobre.

2.2 Metody pamiętania elementów

Ponieważ wielkość tablicy T jest z reguły znacznie mniejsza od wielkości uniwersum, dość często zetkniemy się z sytuacją, gdy chcemy w T zapamiętać y, taki że $h(y) = h(x)$ dla pewnego x aktualnie pamiętanego w T. Sytuację taką nazywamy *kolizją*. Sposoby rozwiązywania kolizji zależą istotnie od tego w jaki sposób pamiętamy elementy w tablicy.

Rozważymy dwa sposoby pamiętania elementów.

2.2.1 Listy elementów

i-ty element tablicy zawiera wskaźnik na początek listy tych elementów x słownika, dla których $h(x) = i$.

Jeśli założymy, że koszt obliczenia wartości funkcji haszującej jest stały, to koszt Insert i Delete też jest stały, a koszt Search(k) jest zależny od długości listy $T[k]$.

Fakt 1 *Przy założeniu (dfh) średni koszt operacji* Search *wynosi* $\Theta(1+\frac{n}{m})$, *gdzie* n*-liczba elementów* U *pamiętanych w* T.

Uwaga. $\alpha = \frac{n}{m}$ nazywane jest *współczynnikiem wypełnienia* tablicy haszującej.

Wniosek 1 *Gdy* $n = O(m)$, *to średni koszt operacji* Search *(a więc także wszystkich operacji słownikowych) jest* $\Theta(1)$.

2.2.2 Adresowanie otwarte

Teraz elementy słownika pamiętamy bezpośrednio w elementach tablicy T. Likwidujemy w ten spoób istotny mankament poprzedniej metody: nie tracimy miejsca na pamiętanie wskaźników. Powstaje jednak nowe niekorzystne zjawisko - przepełnienie się tablicy T. Często nie potrafimy z góry określić wielkości słownika i dlatego rozpoczynamy z tablicą umiarkowanych rozmiarów. Gdy okazuje się ona za mała (jest tak nie tylko wtedy gdy w słowniku chcemy umieścić $(m+1)$-szy element, lecz już wtedy gdy duży stopień wypełnienia tablicy powoduje, że operacje słownikowe są kosztowne), powiększamy ją, zmieniamy funkcję haszującą (tak by przyjmowała wartości z nowego zakresu) i na nowo obliczamy miejsca umieszczenia wszystkich elementów słownika.

Usuwanie kolizji

Używamy funkcji haszującej

$$h : U \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}.$$

Najpierw próbujemy umieścić element k na pozycji $h(k, 0)$. Jeśli pozycja ta jest zajęta, próbujemy $h(k, 1)$, jeśli ta także jest zajęta sprawdzamy pozycję $h(k, 2)$, itd...

Funkcja h powinna spełniać następujący warunek:

$$\text{(per)} \qquad \forall_{k\in U} \langle h(k, 0), \ldots, h(k, m-1)\rangle \text{ jest permutacją zbioru } \{0, 1, \ldots, m-1\}.$$

To gwarantuje nam, że nie znajdziemy miejsca na umieszczenie danego elementu dopiero wtedy, gdy tablica jest całkowicie zapełniona.

Przykłady:

- *Metoda liniowa*:
$$h(k,i) = (h'(k) + i) \bmod m,$$
gdzie $h' : U \to \{0,..,m-1\}$ jest pomocniczą funkcją haszującą (np. takie jak opisano powyżej).

- *Metoda kwadratowa*:
$$h(k,i) = (h'(k) + c_1 i + c_2 * i^2) \bmod m,$$
gdzie h' - jak poprzednio.
UWAGA: $c_1, c_2 \neq 0$. Stałe c_1, c_2 oraz m powinny być tak dobrane by zachodził warunek (per).

- *Podwójne haszowanie*:
$$h(k,i) = (h_1(k) + ih_2(k)) \bmod m,$$
gdzie h_1, h_2 - pomocnicze funkcje haszujące.
UWAGA: Dla każdego $k \in U$, $h_2(k)$ powinno być względnie pierwsze z m.

W praktyce najlepsze rezultaty daje podwójne haszowanie. W metodzie liniowej (i w mniejszym stopniu w metodzie kwadratowej) występuje negatywne zjawisko tworzenia się *zlepków* (tj. zwartych obszarów tablicy T, zajętych przez elementy U), które znacznie obniża efektywność metody.

Podczas wykonywania operacji DELETE w miejscu usuwanego elementu w tablicy T należy wpisać znacznik świadczący o tym, że to miejsce było już kiedyś zajęte.

Analiza kosztów

Dla uproszczenia analizy stosujemy poniższe (nieco wyidealizowane) założenie:

(dper) ciąg $\langle h(k,0), \ldots, h(k,m-1)\rangle$ jest z równym prawdopodobieństwem dowolną permutacją zbioru $\{0, \ldots, m-1\}$.

Twierdzenie 1 *Przy założeniu (dper) i $\alpha = \frac{n}{m} < 1$ oczekiwana liczba prób w poszukiwaniu zakończonym fiaskiem jest $\leq \frac{1}{1-\alpha}$.*

PRZYKŁAD. Załóżmy, że utworzyliśmy słownik i teraz wykonujemy wiele operacji SEARCH. Jeśli tablica jest zajęta w 50%, to średnia liczba prób przy poszukiwaniu zakończonym niepowodzeniem jest nie większa od 2; gdy tablica zajęta jest w 90%, to liczba ta jest nie większa od 10.

Wniosek 2 *Przy powyższych założeniach umieszczenie elementu w tablicy haszującej wymaga średnio $\leq \frac{1}{1-\alpha}$ prób.*

Twierdzenie 2 *Przy założeniu (dper) i $\alpha = \frac{n}{m} < 1$ oczekiwana liczba prób w poszukiwaniu zakończonym sukcesem jest $\leq \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$.*

PRZYKŁAD. Gdy tablica wypełniona jest w 90%, poszukiwanie zakończone sukcesem wymaga średnio nie więcej niż 2.56 prób.

UWAGA. W praktyce, pomimo niespełnienia założenia (dper), koszt operacji słownikowych jest zbliżony do kosztu wynikającego z powyższych twierdzeń.

3 Haszowanie uniwersalne

Oczywiście dla każdej funkcji haszującej istnieją dane, które powodują, że czas wykonywania operacji słownikowych jest duży (np. może się zdarzyć, że dla wszystkich elementów umieszczanych w słowniku wartość funkcji haszującej będzie ta sama). Aby uniezależnić się od takich danych wprowadzamy, podobnie jak w algorytmie *Quicksort*, randomizację: zamiast korzystać z ustalonej funkcji haszującej, losujemy ją na początku działania programu z pewnej rodziny funkcji.

Definicja 1 *Niech H będzie rodziną funkcji haszujących z U w $\{0,\ldots,m-1\}$. Rodzinę H nazywamy* uniwersalmą, *jeśli* $\forall_{x,y\in U;\ x\neq y}$:

$$|\{h \in H: \ \ h(x) = h(y)\}| \leq \frac{|H|}{m}$$

Twierdzenie 3 *Niech H będzie uniwersalną rodziną funkcji haszujących. Dla dowolnego zbioru $n \leq m$ kluczy, liczba kolizji w jakich bierze udział ustalony (ale dowolny) klucz x jest w średnim przypadku mniejsza od 1.*

3.1 Przykłady rodzin uniwersalnych

3.2 R1

Niech m będzie liczbą pierwszą oraz $|U| < m^{r+1}$. Dla każdego $0 \leq a < m^{r+1}$ definiujemy funkcję h_a:

$$h_a(x) = \sum_{i=0}^{r} a_i x_i \bmod m,$$

gdzie $\langle a_0, a_1, \ldots, a_r\rangle$ i $\langle x_0, x_1, \ldots, x_r\rangle$ są reprezentacjami odpowiednio liczb a i x w systemie m-arnym.

Twierdzenie 4 *Rodzina $H = \{h_a : 0 \leq a < m^{r+1}\}$ jest rodziną uniwersalną.*

3.3 R2

Niech p będzie liczbą pierwszą większą niż uniwersum kluczy i niech m będzie wielkością tablicy haszującej (a więc także ograniczeniem na wielkość słowników).

Dla dowolnych $a \in Z_p^*$ i $b \in Z_p$ definiujemy

$$h_{ab}(x) = ((ax+b) \bmod p) \bmod m$$

Fakt 2 *Zbiór funkcji*

$$H_{pm} = \{h_{ab} : a \in Z_p^* \text{ i } b \in Z_p\}$$

jest rodziną uniwersalną.

Dowód: Niech $h'_{ab}(x) = (ax+b) \bmod p$. Niech k i l będą różnymi kluczami i niech $s = h'(k)$ oraz $t = h'(l)$.

Spostrzeżenia:

- $s \neq t$.
- Każda z funkcji h'_{ab} przekształca (k,l) na inną parę (s,t), taką, że $(s = h_{ab}(k), t = h_{ab}(l))$.
- Ponieważ funkcji h'_{ab} jest tyle co par (s,t) z $s \neq t$, więc każda para (t,s) z $t \neq s$ jest obrazem pary (k,l) dla pewnej funkcji h'_{ab}.
  $a \neq 0$ więc funkcji h'_{ab} jest $p(p-1)$. Jeśli $s = (h'_{ab}(k)$ i $h'_{ab}(l))$, to $s \neq t$, więc takich par (s,t) też jest $p(p-1)$.
- Istnieje więc bijekcja między funkcjami h'_{ab} a parami wartości (s,t), na które może być odwzorowana para kluczy (k,l).
- Zatem prawdopodobieństwo kolizji kluczy (k,l) przy haszowaniu jest równe prawdopodobieństwu wylosowania pary (s,t) spośród wszystkich par z $s \neq t$, dla których $t \equiv s \bmod m$.
- Dla dowolnego (ustalonego s) liczba takich t, które przystają do s modulo m i są różne od s jest równa $\lfloor p/m \rfloor - 1 \leq (p+m-1)/m - 1 = (p-1)/m$.
- Ponieważ różnych t dla danego s jest $p-1$ więc prawdopodbieństwo, że s i t kolidują jest $\leq 1/m$.

□

4 Schematy urnowe

Haszowanie w oczywisty sposób odpowiada klasycznemu zagadnieniu z rachunku prawdopodobieństwa: mamy m kul, które wrzucamy losowo, niezależnie i w sposób jednostajny do n urn. Przykłady pytań, które nas interesują:

- jakiej liczby kul możemy spodziewać się w najbardziej popularnej urnie (tj. takiej, do której trafi najwięcej kul)?
- ile kul możemy wrzucić do urn, by z dużym prawdopodobieństwem do każdej urny trafiła co najwyżej jedna kula?

Pierwsze z pytań jest pytaniem o pesymistyczny czas operacji wyszukiwania klucza w słowniku, w którym klucze nawlekane są na listy.

Twierdzenie 5 *Gdy m kul wrzucamy losowo, niezależnie i w sposób jednostajny do n urn, to dla odpowiednio dużego n, z prawdopodobieństwem co najmniej $1 - 1/n$ liczba kul w najbardziej popularnej urnie nie przekracza $\frac{3 \ln n}{\ln \ln n}$*

Ponadto można pokazać, że z dużym prawdopodobieństwem liczba ta jest osiągana.

- n kul do n urn - średnia i maksymalna liczba kul w urnie
- m kul do n urn - jeśli $m \leq \sqrt{n}$ to z ppb'stwem $> 1/2$ nie ma kolizji

Jak zmniejszyć maksymalną liczbę kul w urnie: dla każdej kuli wybieramy losowo dwie urny ostatecznie kula ląduje w tej urnie, w której było mniej elementów

Twierdzenie 6 *Jeśli m kul wrzucamy do n urn w sposób zbalansowany, to z dużym prawdopodobieństwem maksymalna liczba kul w urnie jest $\Theta(\ln \ln n / \ln 2)$.* [1]

5 Słownik statyczny

Ustalony zbiór n kluczy chcemy zapamiętać tak, by:

- struktura zajmowała n komórek pamięci,
- (oczekiwany) czas konstrukcji struktury był wielomianowy względem n,
- czas wykonywania instrukcji *find* był stały.

Nie chcemy wykonywać operacji *insert* i *delete*.

IDEA:

- stosujemy haszowanie dwupoziomowe,
- na pierwszym poziomie funkcja haszująca rozrzuca klucze do kubełków tak, by $\sum_{i=0}^{n-1} n_i^2 = O(n)$, gdzie n_i - liczba kluczy wrzuconych do kubełka i,
- na drugim poziomie haszujemy niezależnie klucze w każdym kubełku używając tablicy o rozmiarze n_i^2; haszowanie to jest bezkolizyjne,
- funkcje haszujące są brane losowo z uniwersalnej rodziny funkcji haszujących.

Lemat 1 (Nierówność Markowa) *Dla każdej zmiennej losowej X i dla każdego $t > 0$:*

$$Pr[|X| \geq t] \leq \frac{E[|X|]}{t}.$$

Fakt 3 *Z prawdopodobieństwem co najmniej 1/2 funkcja wybrana losowo z rodziny uniwersalnej bezkonfliktowo umieszcza $n = \sqrt{m}$ kluczy w tablicy m elementowej.*

UZASADNIENIE: W zbiorze n kluczy jest $< n^2/2$ par kluczy. Każda para koliduje z ppb $\leq 1/m$. Stąd oczekiwana liczba kolizji podczas wstawiania n kluczy jest mniejsza niż $n^2/m < 1/2$. Stosujemy lemat Markowa dla $t = 1$. □

Lemat 2 *Jeśli do umieszczenia n kluczy w tablicy n elementowej użyjemy funkcji losowo wybranej z rodziny uniwersalnej, to z prawdopodobieństwem co najmniej $1/2$ zachodzi:*

$$\sum_{j=0}^{n-1} n_j^2 < 4n,$$

gdzie n_j oznacza liczbę kluczy umieszczonych w j-tym kubełku.

UZASADNIENIE:

Najpierw pokazujemy, że wartość oczekiwana sumy $\sum_{j=0}^{n-1} n_j^2$ jest mniejsza od $2n$, potem stosujemy nierówność Markowa dla $t = 4n$.

Chcemy obliczyć $E[\sum_{j=0}^{n-1} n_j^2]$. Umiemy policzyć:

- $E[\sum_{j=0}^{n-1} n_j]$. Ta suma jest równa n - liczbie wszystkich kluczy w słowniku.
- $E[\sum_{j=0}^{n-1} \binom{n_j}{2}]$. Ta suma jest równa liczbie wszystkich kolizji. Ponieważ każda para kluczy koliduje z ppb'stwem nie większym od $1/n$ (tu n jest także wielkością tablicy), więc oczekiwana liczba kolizji jest nie większa od $\frac{n(n-1)}{2} \cdot \frac{1}{n} = \frac{n-1}{2}$.

Ale

$$E\left[\sum_{j=0}^{n-1} \binom{n_j}{2}\right] = E\left[\sum_{j=0}^{n-1} \frac{n_j^2 - n_j}{2}\right],$$

więc

$$E\left[\sum_{j=0}^{n-1} n_j^2\right] = 2E\left[\sum_{j=0}^{n-1} \binom{n_j}{2}\right] + E\left[\sum_{j=0}^{n-1} n_j\right] \leq 2\frac{n-1}{2} + n < 2n.$$

□

Reasumując:

- **Pamięć**. Potrzebujemy (dla dobrze wylosowanych funkcji):
  - nie więcej niż $4n$ komórek na tablice wtórne,
  - trzy komórki na parametry funkcji pierwotnej (jedną na p, jedną na a, jedną na b),
  - dodatkowo na każdą tablicę wtórną 3 komórki: jedną na rozmiar tablicy; dwie na parametry funkcji; czyli w sumie $3n$ komórek.
- **Czas tworzenia struktury.**
  - Oczekiwana liczba losowań funkcji pierwotnej jest nie większa od 2. Sprawdzenie, czy wylosowana funkcja jest dobra wymaga obliczenia jej wartości dla wszystkich n kluczy. To daje się zrobić w czasie $O(n)$.
  - Oczekiwana liczba losowań wtórnej funkcji losowej (dla j-tego kubełka) jest nie większa od 2. Czas sprawdzenia, czy jest dobra jest $O(n_j)$. Stąd oczekiwany czas związany z losowaniem wszystkich funkcji wtórnych jest $O(n)$.
- **Czas instrukcji *find*.** Jest stały - wystarczy bowiem wyliczyć wartości dwóch funkcji haszujących.

6 Extendible arrays

7 Appendix

7.1 Dowód Twierdzenia 5

7.2 Dowód Twierdzenia ??

Prawdopodobieństwo, że podczas rozrzucania żadna z k nie trafi do urny

Literatura

[1] Y. Azar, A.Z. Broder, A.R. Karlin, E. Upfal, Balanced Allocations, *SIAM J. Comput* , 29(1999), 180–200.