Uniwersytet Zielonogórski
Instytut Sterowania i Systemów Informatycznych

Algorytmy i struktury danych

Laboratorium 6

Techniki wyszukiwania danych – haszowanie

1 Cel ćwiczenia

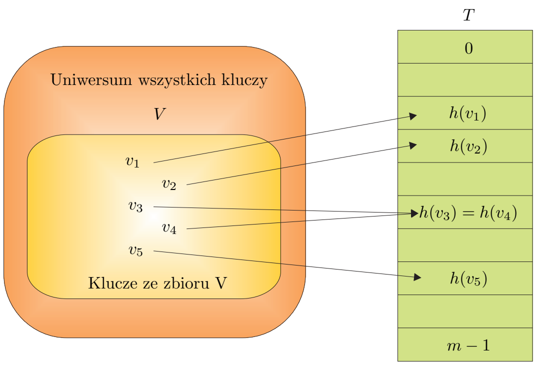
Ćwiczenie ma na celu zapoznanie studentów z technikami wyszukiwania danych.

2 Haszowanie

W klasycznych metodach wyszukiwanie polega na porównywaniu wyszukiwanego klucza z elementami przeszukiwanego zbioru. W haszowaniu próbuje się uzyskać bezpośrednie odniesienie do elementów w tablicy za pomocą operacji arytmetycznych przekształcających klucz w odpowiedni adres tablicy. Inaczej mówiąc, idea haszowania (rys. 1) polega na odnalezieniu takiej funkcji $h(\cdot)$, która na podstawie danej opisanej przez pewien klucz v, wskazywałaby indeks i w tablicy T pod którym ma być przechowana dana

$$T(i) = h(v),$$

- T - tablica przechowująca dane,
- x - dana opisana przez pewien klucz v,
- h - funkcja haszująca.



Rys. 1: Funkcja haszująca h przyporządkowuje komórki w tablicy z haszowaniem kluczom z uniwersum V. Klucze v_3 i v_4 kolidują ze sobą ponieważ odpowiada im ta sama komórka w tablicy

Największą zaletą takiego sposobu zapamiętywania danych jest maksymalne uproszczenie procesu poszukiwań. Znając funkcję $h(\cdot)$, można automatycznie określić indeks komórki tablicy w której zapisana jest dana. Oprócz przedstawionej zalety istnieje jedna podstawowa wada omawianej metody polegająca na trudności w odnalezieniu dobrej funkcji $h(\cdot)$.

3 Pożądane właściwości funkcji haszującej

Istnieje wiele potencjalnych funkcji haszujących, które na podstawie wartości danego klucza v wyznaczają pewien indeks tablicy. Dobra funkcja haszująca powinna posiadać dwie następujące własności:

- Funkcja haszująca powinna powodować równomiernie obciążanie pamięci, czyli różnym wartościom klucza v powinny odpowiadać odmienne indeksy tablicy T, aby nie powodować konfliktu dostępu.
- Funkcja haszujaca powinna być możliwie jak najprostsza.

Parametry, które maja wpływ na stopień złożoności funkcji $h(\cdot)$ to: długość tablicy, w której zamierzamy składować rekordy danych oraz wartość klucza v.

4 Znane funkcje haszujące

W zaprezentowanych przykładach będziemy zakładać, ze klucze v są ciągami znaków, które można łączyć ze sobą i dość dowolnie interpretować jako liczby całkowite. Każdy znak alfabetu będziemy kodować w przykładach przy pomocy pięciu bitów:

A=00001	B=00010	C=00011	D=00100	E=00101	F=00110	G=00111
H=01000	I=01001	J=01010	K=01011	L=01100	M=01101	N=01110
O=01111	P=10000	Q=10001	R=10010	S=10011	T=10100	U=10101
V=10110	W=10111	X=11000	Y=11001	Z=11010		

Kodowanie jest wykonywane w celu zmiany danej x na klucz v, który można poddać działaniu funkcji haszującej. Celem haszowania jest możliwie jak najbardziej losowe rozrzucenie rekordów po tablicy wielkości M. Problem polega na tym, że nie jest możliwe uzyskanie w pełni losowego rozrzutu elementów dysponując danymi wejściowymi, które z założenia nie są losowe. Musimy zatem ową losowość w jakiś sposób zasymulować. Istnieje grupa prostych funkcji arytmetycznych (modulo, mnożenie, dzielenie), które dość dobrze nadają się do tego celu.

1. **Suma modulo 2**

   $$h(v_1, v_2, \ldots, v_n,) = v_1 \oplus v_2 \oplus \ldots \oplus v_n$$

   gdzie $h(\cdot)$ - suma modulo 2.

   Przykład: Dla $T_{\max} = 37$ oznaczającego maksymalną liczbę elementów tablicy, oraz $h(KOT) = (010110111110100)_2$ otrzymujemy:

   $$(01011)_2 \oplus (01111)_2 \oplus (10100)_2 = (10000)_2 = 16$$

   Zalety:

   - Wartość funkcji $h(\cdot)$ jest łatwa do obliczenia.
   - Suma modulo 2, w przeciwieństwie do iloczynu i sumy logicznej, nie powiększa (jak suma logiczna) lub pomniejsza (jak iloczyn) swoich argumentów.

   Wady:

   - Permutacja tych samych liter daje w efekcie ten sam wynik. Rozwiązaniem tego problemu może być systematyczne przesuwanie cykliczne reprezentacji bitowej: pierwszy znak o jeden bit w prawo, drugi o dwa bity w prawo, etc.

   Przykład: Bez przesuwania $h(KTO) = 16$ i jednoczenie $h(TOK) = 16$, z przesunięciem $h(KTO) = 16$ $(10101)_2 \oplus (00101)_2 \oplus (11101)_2 = (01101)_2 = 13$ natomiast $h(TOK) = 16$ $(01010)_2 \oplus (11011)_2 \oplus (01101)_2 = (11100)_2 = 28$.

2. **Wyznaczania reszty z dzielenia całkowitego $T_{\max}$**

   $$h(v) = v \mathrm{mod} T_{\max}$$

   gdzie mod oznacza operację wyznaczania reszty z dzielenia całkowitego.

   Przykład: Dla $T_{\max} = 37$ oraz $h(KOT) = (010110111110100)_2 \mathrm{mod} 37 = 35$

   Zalety:

   - Wartość funkcji $h(\cdot)$ jest łatwa do obliczenia.

   Wady:

- Otrzymana wartość zależy bardziej od $T_{\max}$, niż od klucza. Gdy $T_{\max}$ jest parzyste, wszystkie otrzymane indeksy danych o kluczach parzystych będą parzyste, ponadto dla pewnych dzielników wiele danych otrzyma te same indeksy. Można temu zaradzić poprzez wybór $T_{\max}$ jako liczby pierwszej, ale często mamy do czynienia z akumulacja elementów w pewnym obszarze tablicy
- W przypadku dużych liczb binarnych nie mieszczących się w reprezentacji wewnętrznej komputera, obliczanie modulo nie jest możliwe przez zwykłe dzielenie arytmetyczne.

3. **Mnożenie** $T_{\max}$

$$h(v) = \lfloor((v \cdot \Theta)\%1)T_{\max}\rfloor, \quad \text{gdzie} \quad 0 < \Theta < 1$$

gdzie $\lfloor\cdot\rfloor$ - część całkowita. Powyższą formułę należy odczytać następująco: klucz jest mnożony przez pewna liczbę Θ z przedziału $(0, 1)$. Z wyniku bierzemy część ułamkową, mnożymy przez $T_{\max}$ i ze wszystkiego liczymy część całkowitą. W literaturze dotyczącej metody haszowania wykazano, że dla poniżej przedstawionych wartości Θ otrzymujemy w miarę równomierne obciążenie pamięci.

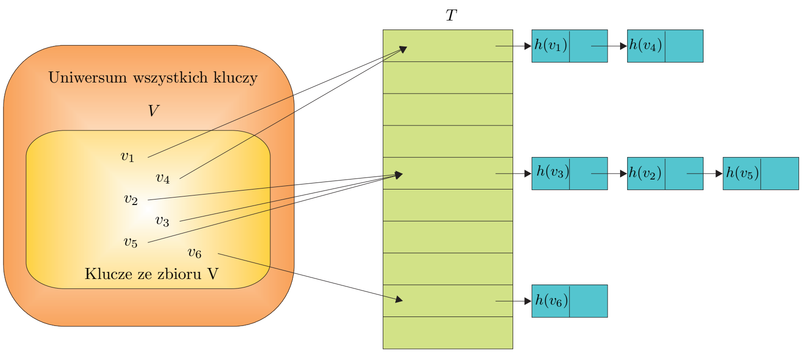
$$\Theta = \frac{1}{2}\sqrt{5} - \frac{1}{2}, \quad \Theta = -\frac{1}{2}\sqrt{5} + \frac{3}{2}.$$

Przykład: Dla $\Theta = 0.61800339887$ oraz $T_{\max} = 30$ i klucza $v = KNT = 1732$ otrzymamy $h(KNT) = 23$.

5 Metody obsługi konfliktów

1. **Metoda łańcuchowa**

W metodzie łańcuchowej wszystkie elementy, którym w wyniku haszowania odpowiada ta sama pozycja w tablicy T, zostają umieszczone na jednej oddzielnej liście (rys. 2). Na pozycji m w



Rys. 2: Rozwiązanie konfliktu metodą łańcuchową

tablicy T zapamiętywany jest jedynie wskaźnik do początku listy danych, dla których funkcja $h(\cdot)$ daje taką samą wartość m. Przeszukiwanie będzie działać w sposób następujący: szukany element x poddany działaniu funkcji haszującej $h(x)$ zwraca pewien indeks m. W przypadku, gdy komórka tablicy $T[m]$ zawiera NULL, możemy być pewni, że szukanego elementu nie odnaleźliśmy. W sytuacji gdy komórka tablicy zawiera listę to musimy przeszukać każdy jej element. Niestety podczas obsługi konfliktów przy pomocy przedstawionej metody korzysta się z list. Z tego powodu rozwiązanie to można uznać za nieco sztuczne, jednakże parametry "czasowe" tej metody są korzystne.

2. **Tablice**

Idea tej metody polega na podziale tablicy T na dwie części: strefę podstawowa i strefę przepełnienia. Do strefy przepełnienia elementy trafiają w momencie wystąpienia konfliktu podczas zapisu danej pod indeks znajdujący się w części podstawowej. Strefa ta wypełniana jest liniowo wraz z napływem nowych elementów powodujących konflikty. Efekt wypełnienia tablicy przedstawiony jest na rys. 3.
Rekordy E i F zostały zapamiętane w momencie stwierdzenia konfliktu. Użycie przedstawionej metody powinno być poprzedzone szczególnie starannym obliczeniem rozmiarów tablicy przepełnienia w celu zagwarantowania odpowiedniej jej wielkości mogącej pomieścić wszystkie elementy wywołujące konflikt.

0	1	2	3	4	5	6	7	8	9
G	A	C	B		D	E	F		
Strefa podstawowa						Strefa przepełnienia			

Rys. 3: Podział tablicy na część podstawową i przepełnienia umożliwiającą obsługę konfliktów

3. **Próbkowanie liniowe**

Jak zauważyliśmy wcześniej, konflikty dostępu są w metodzie haszowania nieuchronne. Nie istnieje bowiem idealna funkcja $h(\cdot)$, która rozmieściłaby równomiernie wszystkie $T_{\max}$ elementów po całej tablicy T. Idea próbkowania liniowego jest przedstawiona na rys. 4.

1)

0	1	2	3	4	5	6	7
	A	C	B		D		G

2)

0	1	2	3	4	5	6	7
	A	C	B	E	D		G

3)

0	1	2	3	4	5	6	7
	A	C	B	E	D	F	G

4)

0	1	2	3	4	5	6	7
H	A	C	B	E	D	F	G

Rys. 4: Obsługa konfliktów dostępu przy próbkowaniu liniowym: 1) bez konfliktu, 2) konflikt B z E, 3) konflikt A z F, 4) konflikt G z H

W momencie zapisu nowego rekordu do tablicy, w przypadku stwierdzenia konfliktu możemy spróbować zapisać element na pierwsze kolejne wolne miejsce. Ciekawe jest teoretyczne wyliczanie średniej ilości prób potrzebnej do odnalezienia danej x. W przypadku poszukiwania zakończonego sukcesem średnia ilość prób wynosi około:

$$\frac{1}{2} + \frac{1}{2}\frac{1}{(1-\alpha)}$$

gdzie α jest współczynnikiem zapełnienia tablicy T. Analogicznie wynik dla przeszukiwania zakończonego niepowodzeniem wynosi około:

$$\frac{1}{2} + \frac{1}{2}\frac{1}{(1-\alpha)^2}$$

Przykładowo dla tablicy zapełnionej w 2/3 swojej pojemności ($\alpha = 2/3$) liczby te wyniosą odpowiednio 2 i 5. W praktyce należy unikać szczelnego zapełnienia tablicy T, gdyż powyższe liczby staja się bardzo duże. Powyższy wzór został wyprowadzony przy założeniu funkcji $h(\cdot)$, która rozsiewa równomiernie elementy po dużej tablicy T. Powyższe zastrzeżenia są bardzo istotne, ponieważ podane wyżej rezultaty maja charakter statystyczny.

4. **Podwójne haszowanie**

W metodzie próbowania liniowego jesteśmy narażeni na liniowe grupowanie elementów, co zwiększa czas poszukiwania wolnego miejsca dla nowego elementu tablicy T. Jednocześnie zwiększa się czas przeszukiwania tablicy T. Istnieje łatwy sposób uniknięcia liniowego grupowania elementów polegający na podwójnym haszowaniu. W momencie napotkania konfliktu następuje próba dodatkowego rozrzucenia elementów przy pomocy drugiej, pomocniczej funkcji haszującej $h_2(\cdot)$. Dobór funkcji $h_2(\cdot)$ ma duży wpływ na jakość wstawiania i poszukiwania. Przede wszystkim funkcja $h_2(\cdot)$ powinna być różna od pierwotnej funkcji haszującej $h_1(\cdot)$. Ponadto, funkcja ta musi być prostą aby nie spowolnić procesu wstawiania i poszukiwania. Przykładem takiej prostej i jednocześnie skutecznej funkcji jest funkcja w postaci: $h_2(v) = 8 - (v \mod 8)$. Metoda podwójnego kluczowania jest interesująca ze względu na zysk w szybkości poszukiwania danych. średnia ilość prób zakończona sukcesem wynosi około:

$$\frac{\ln(\frac{1}{1-\alpha})}{\alpha}$$

gdzie α jest współczynnikiem zapełnienia tablicy T. Analogicznie wynik dla poszukiwania zakończonego niepowodzeniem wynosi około:

$$\frac{1}{1-\alpha}$$

Literatura

[1] T.H.Cormen i in. , *Wprowadzenie do algorytmów*, WNT, 2000

[2] A.Drozdek *Struktury danych w jezyku C*, WNT, 1996

[3] D.Horel *Rzecz o istocie informatyki. Algorytmika*, WNT, 1992

[4] R.Sedgewick *Algorytmy w C++*, Wydawnictwo RM, 1999

[5] P.Wróblewski, *Algorytmy, struktury danych i techniki programowania*, HELION, 1996