Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wykład 10.

Słowniki i haszowanie

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Słowniki

Słownik (inaczej: tablica skojarzeniowa, tablica asocjacyjna, tablica indeksowana wartością klucza, odwzorowanie, mapa) to typ abstrakcyjny z następującymi operacjami:

- JestElementem - sprawdzenie, czy dany element występuje w słowniku, czasem "wyszukaj element o danym kluczu",
- Dodaj - dopisz element do słownika,
- Usuń - usuń element ze słownika,
- Wybierz - wyszukaj k-ty co do wielkości element słownika,
- Sortuj - udostępnij wszystkie elementy w sposób uporządkowany,
- Połącz - połącz dwa słowniki w jeden.

Najczęściej wykonywanymi operacjami na słowniku są wyszukiwania.
Zatem efektywność operacji wyszukiwania decyduje o przydatności danej realizacji słownika.

Metody wyszukiwania elementu zależą od sposobu organizacji słownika.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Słowniki

Elementy występujące w słowniku mogą się składać:
- z samego klucza,
- z klucza i skojarzonych z nim danych.

Od typu elementu zależy sposób działania operacji JestElementem:

1. Gdy element ma tylko klucz, następuje tylko sprawdzenie, czy element występuje w słowniku,
2. Gdy element posiada klucz i skojarzone z nim dane, następuje zwrócenie wartości pola "dane".

Operacje zmieniające stan słownika, czyli:
- dodawanie elementu,
- usuwanie elementu

są wykonywane relatywnie rzadko (mają sens "budowania słownika"), więc ich efektywność nie jest szczególnie istotna (nie ma praktycznego znaczenia).

Najczęściej takie operacje są wykonywane w trybie off-line.

Tylko w specyficznych sytuacjach budowanie słownika odbywa się w czasie normalnej pracy (on-line).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Słowniki

Do przechowywania elementów słownika można stosować następujące struktury danych :

- ciąg nieuporządkowany (tablica, plik lub lista o losowej kolejności elementów),
- ciąg uporządkowany (tablica, plik, lista, drzewo),
- tablica z haszowaniem ("tablica haszowana").

Operacje wyszukiwania w ciągach nieuporządkowanych odbywają się metodą "pełnego przeglądu" (wyszukiwanie liniowe) o złożoności O(N).

Dla ciągów uporządkowanych można zastosować:

- wyszukiwanie liniowe, umożliwiające szybsze (w porównaniu z ciągami nieuporządkowanymi) wykrycie braku elementu (gdy trafimy na element większy – kończymy przeszukiwanie ciągu z wynikiem negatywnym).

- wyszukiwanie binarne (z sukcesywnym podziałem obszaru przeszukiwania - dobór pozycji elementów <a,b> (o wartościach: klucz(a), klucz(b)) w zależności od wartości kluczX).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Słowniki z wyszukiwaniem binarnym

Idea wyszukiwania binarnego:

```
// wyszukujemy tylko wówczas, gdy
// kluczX jest w przedziale < klucz(a), klucz(b) >
// w przeciwnym przypadku: brak elementu w ciągu

// oznaczenia: a – pozycja 1. elementu, b – pozycja ostatniego elementu
boolean koniec=false;
do
    c=(a+b) div 2;   // "połowienie przedziału"
    if ( kluczX==klucz(c) ) { koniec=true; }
      else if ( kluczX > klucz(c) ) { a = c+1; }
               else { b = c; }
while (!koniec) && (a<b);
if (koniec) { ..... }   // znaleziono element na pozycji c
  else { .... }         // nie znaleziono elementu
```

Złożoność obliczeniowa wyszukiwania binarnego wynosi O(log(N)).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

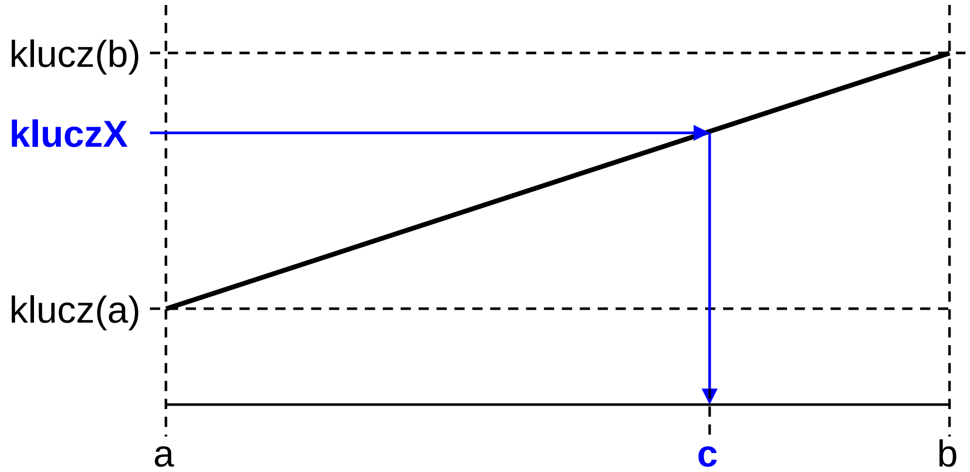
Słowniki z równomiernym rozkładem wartości kluczy

W zastosowaniu do ciągów o równomiernym (lub zbliżonym do równomiernego) rozkładzie wartości kluczy wyszukiwanie binarne można jeszcze przyspieszyć, stosując podział interpolacyjny przedziału < a, b >, w proporcjach wynikających z wartości klucza: klucz(a), klucz(b) w pozycjach a oraz b.

(...)

c = a + (kluczX – klucz(a)) / (klucz(b) – klucz(a)) * (b-a);

(...)



Uzyskuje się wówczas jeszcze lepszą złożoność obliczeniową: O(log(log(N))).

Analogicznie, dla innych (znanych) rozkładów wartości kluczy, można zastosować inne zasady podziału.

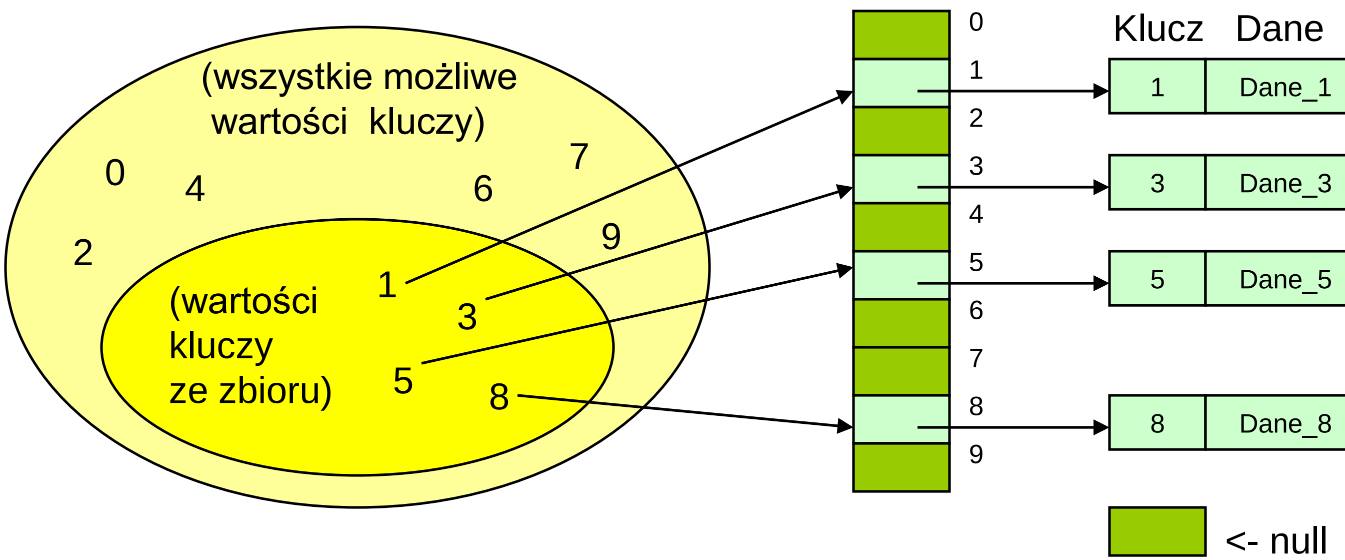
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Słowniki z użyciem struktur o efektywnym, bezpośrednim dostępie

Organizacja słownika z użyciem tablicy z adresowaniem bezpośrednim

Tablice z adresowaniem bezpośrednim mają zastosowanie wówczas, gdy:
- zbiór wartości kluczy elementów ciągu (zbioru) jest nieduży (zawiera się w przedziale <0, MAX-1>, co praktycznie oznacza możliwość utworzenia tablicy o rozmiarze MAX),
- wartości kluczy są unikatowe (nie powtarzają się w zbiorze).

Uzyskuje się stałą złożoność obliczeniową (O(1) – także złożoność pesymistyczną!) kosztem (niewykorzystanej) pamięci.



Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Słowniki z użyciem struktur o efektywnym, bezpośrednim dostępie

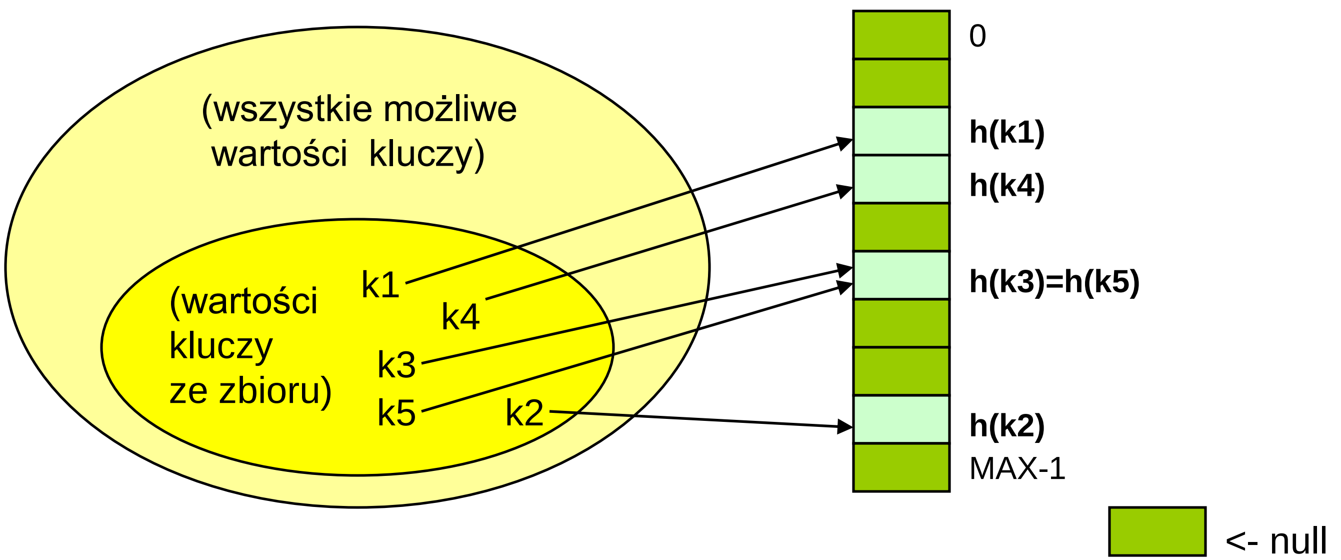
Organizacja słownika z użyciem tablicy z haszowaniem ("tablicy haszowanej")

Tablica z haszowaniem jest uogólnieniem tablicy z adresowaniem bezpośrednim. Stosuje się ją, gdy zbiór dopuszczalnych wartości kluczy jest zbyt liczny (MAX), żeby zastosować adresowanie bezpośrednie poprzez wartość klucza lub gdy zbiór wartości kluczy należących do zbioru dynamicznego jest mały w stosunku do MAX.

Idea haszowania polega na:

- przyporządkowaniu do każdej wartości klucza k (mogącej wystąpić w zbiorze) określonej wartości funkcji haszującej h(k),

- użyciu wartości h(k) do uzyskania dostępu do danych (średnio) w czasie stałym (złożoność obliczeniowa O(1)) do elementu o danej wartości klucza k.



Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

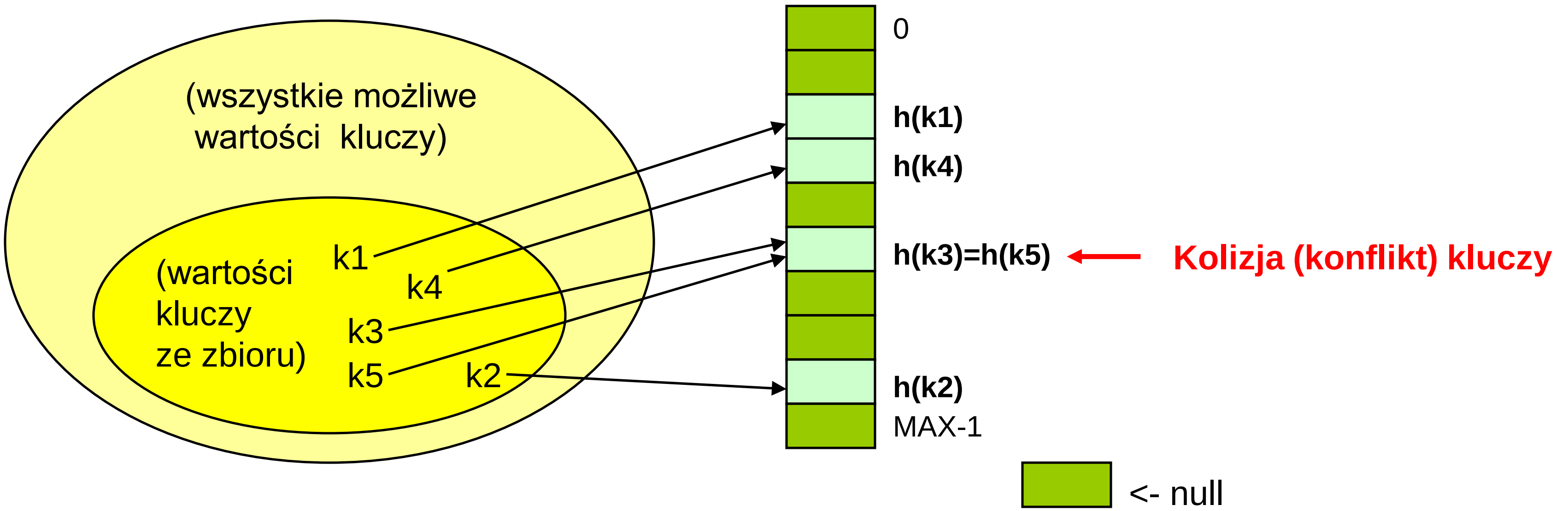
Słowniki z użyciem struktur o efektywnym, bezpośrednim dostępie

Organizacja słownika z użyciem tablicy z haszowaniem ("tablicy haszowanej")

Tablica z haszowaniem jest uogólnieniem tablicy z adresowaniem bezpośrednim. Stosuje się ją, gdy zbiór dopuszczalnych wartości kluczy jest zbyt liczny (MAX), żeby zastosować adresowanie bezpośrednie poprzez wartość klucza lub gdy zbiór wartości kluczy należących do zbioru dynamicznego jest mały w stosunku do MAX.

Idea haszowania polega na:

- przyporządkowaniu do każdej wartości klucza k (mogącej wystąpić w zbiorze) określonej wartości funkcji haszującej h(k),
- użyciu wartości h(k) do uzyskania dostępu do danych (średnio) w czasie stałym (złożoność obliczeniowa O(1)) do elementu o danej wartości klucza k.



Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

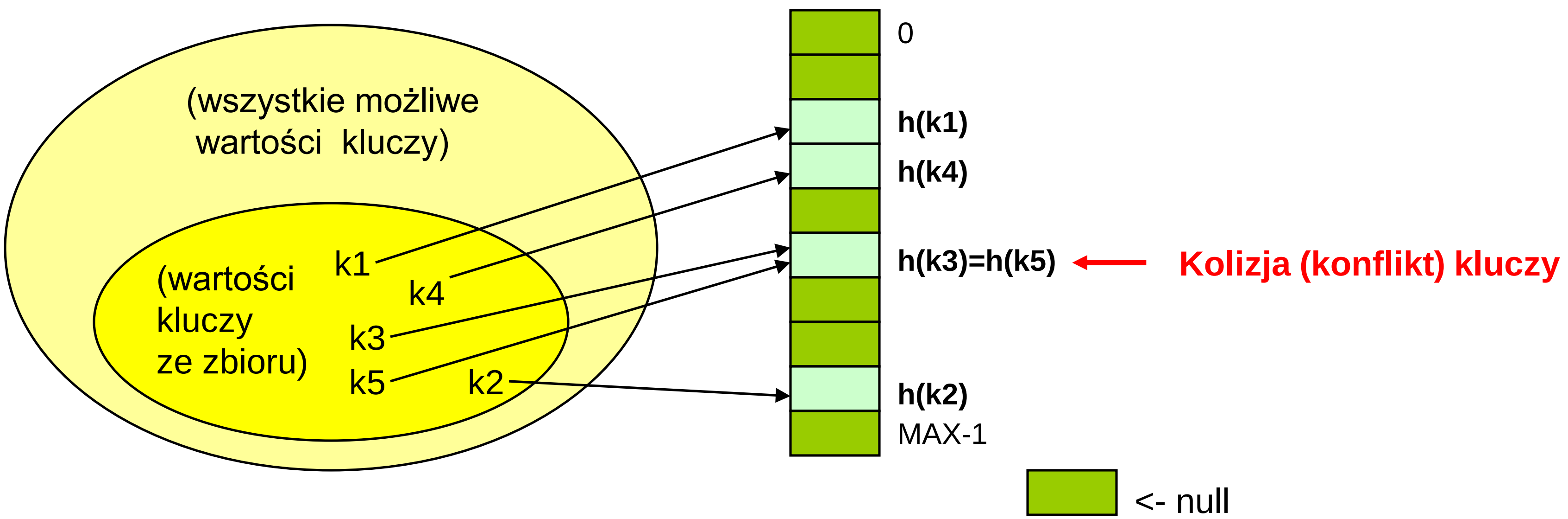
Słowniki z użyciem struktur o efektywnym, bezpośrednim dostępie

Organizacja słownika z użyciem tablicy z haszowaniem ("tablicy haszowanej")

Kolizja kluczy jest ceną płaconą za duże oszczędności pamięci.
Istnieją dwa główne zagadnienia związane z haszowaniem:

1. Dobór funkcji haszującej h (która musi być deterministyczna), zapewniającej uniknięcie lub co najmniej ograniczenie (zminimalizowanie?) liczby kolizji.

2. Opracowanie metody rozwiązywania kolizji.



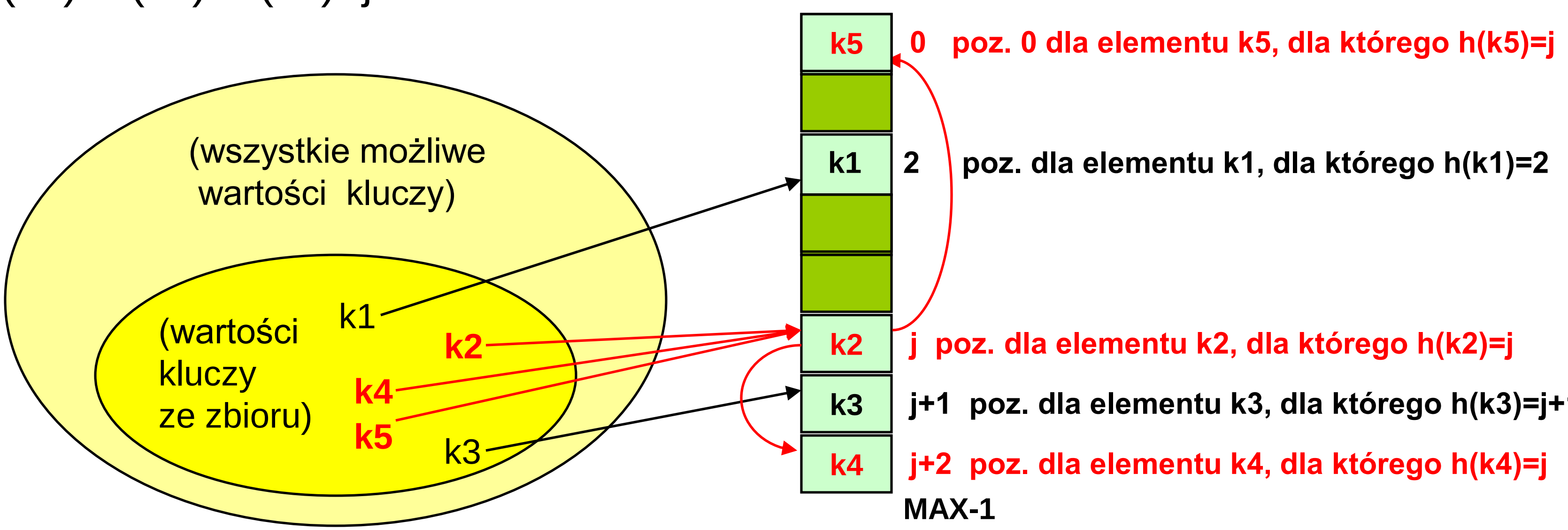
Prosty sposób rozwiązania kolizji kluczy: tablica statyczna

Algorytm zapisu elementu X o wartości klucza k do tablicy z haszowaniem:

if (pozycja tablicy h(k) jest wolna) zapisz element X
  else **{**
    znajdź pierwszą wolną pozycją (od pozycji h(k), modulo rozmiar tablicy),
    zapisz element na wolnej pozycji
  }

Jest to tzw. rozwiązywanie kolizji poprzez adresowanie otwarte.

Przykład: zakładamy, że elementy były wstawiane w kolejności: k1, k2, k3, k4, k5, h(k2)=h(k4)=h(k5)=j



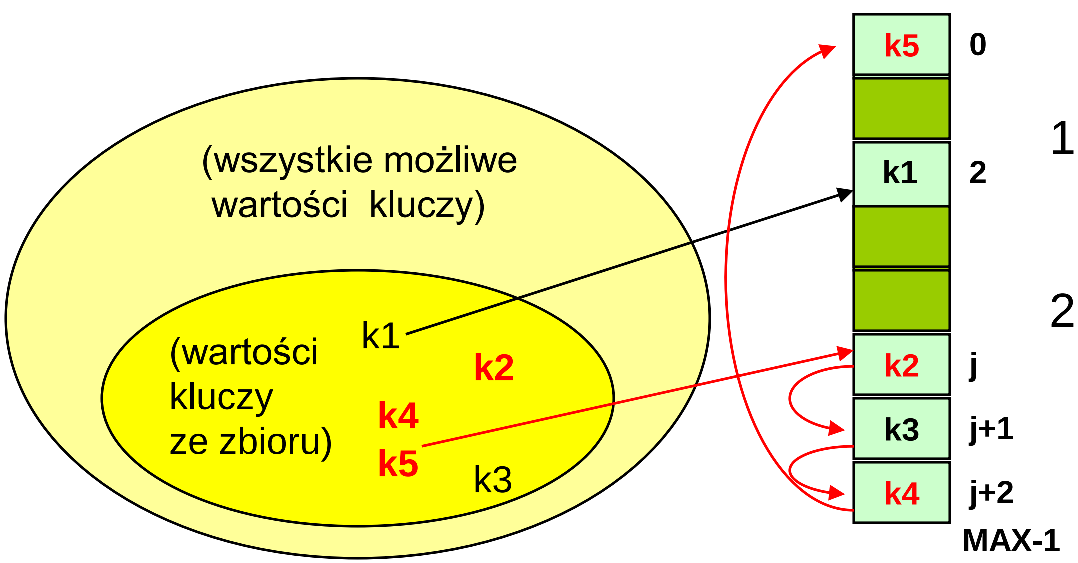
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Prosty sposób rozwiązania kolizji kluczy: tablica statyczna

Wyszukanie elementu o wartości klucza k w tablicy z haszowaniem:

// zakładamy, że w tablicy zawsze istnieje jedna wolna pozycja – dla uproszczenia
if (na pozycji h(k) jest szukany element) { koniec, element znaleziono }
else **{**
szukaj elementu na kolejnych pozycjach tablicy, modulo rozmiar tablicy;
if (zakończenie wyszukiwania na wolnej pozycji) { koniec, nie znaleziono }
else { koniec, element znalaziono }
}

Wyszukiwanie może być czasochłonne.



Przykłady:

1. Szukamy k1 na poz. 2, gdyż h(k1)=2, znajdujemy k1.

2. Szukamy k5: h(k5)=j, ale na poz. j nie ma k5, na j+1 i j+2 też nie ma k5, następną poz. jest poz. 0; tu znajdujemy k5; koniec, element znaleziono.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

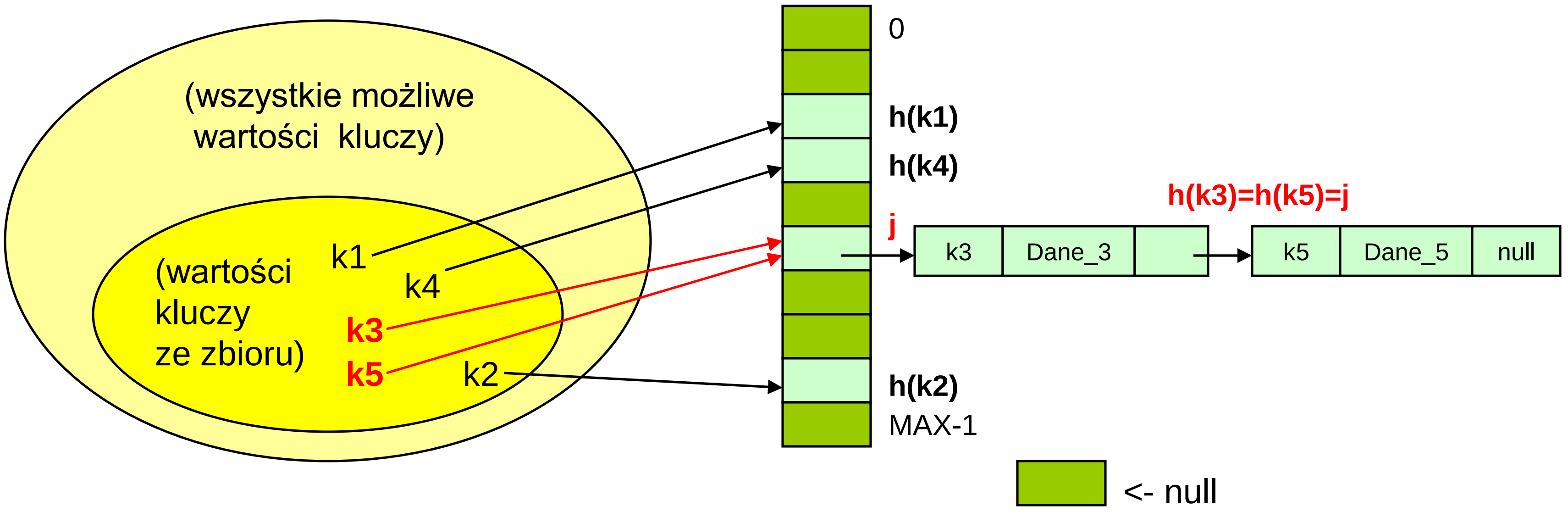
Inna (uznawana za najprostszą) technika rozwiązywanie kolizji kluczy: metoda łańcuchowa (tablica list)

Wszystkie elementy, którym odpowiada ta sama pozycja "j" w tablicy (czyli dla których funkcja haszująca daje wartość j), zostają umieszczone w jednej liście, do której referencja znajduje się w komórce "j" tablicy.

Złożoność obliczeniowa dla operacji wstawiania jest stała: O(1), jeśli wstawia się element na początku listy.

Operacja wyszukania elementu wymaga przeszukiwania listy, więc jest zależna od długości listy (złożoność obliczeniowa, w wariancie pesymistycznym, liniowa: O(N)).

Usuwanie elementu także ma taką samą złożoność (o ile lista jest podwójnie wiązana), ponieważ wymaga wyszukania elementu.



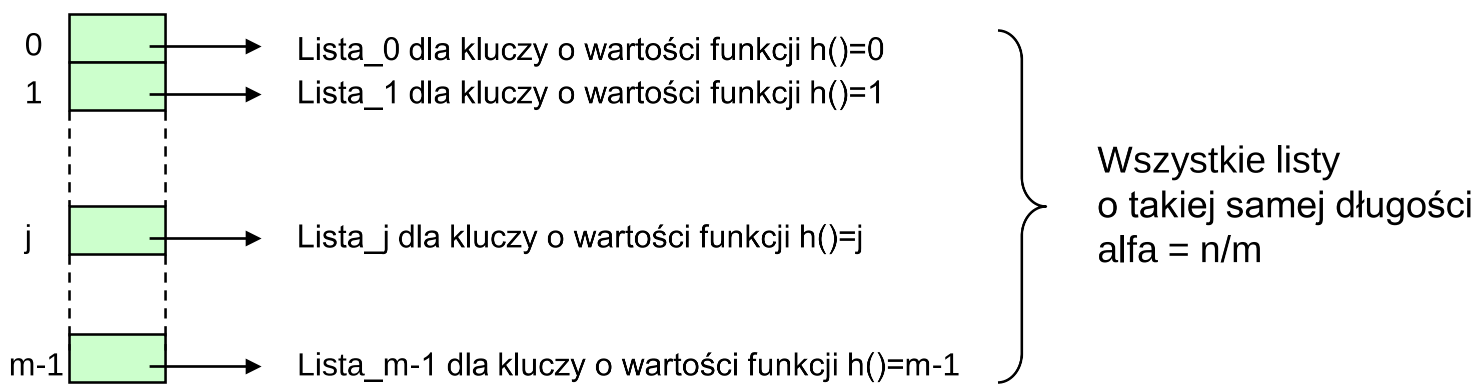
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Analiza haszowania metodą łańcuchową

Dla tablicy o m pozycjach, w której znajduje się n elementów zbioru, współczynnik jej zapełnienia alfa= n/m. Jest to średnia długość listy elementów (łańcucha).

Przypadek najgorszy: klucze wszystkich elementów zbioru zostały odwzorowane na jeden element tablicy (tworząc jedną listę n-elementową). Złożoność: O(N).

Przypadek najlepszy: klucze wszystkich elementów zbioru zostały odwzorowane równomiernie (tworząc m list n/m-elementowych). Jeśli funkcja haszująca zapewnia taki równomierny rozkład (nazywany prostym równomiernym haszowaniem), a liczba pozycji w tablicy jest proporcjonalna do n, to złożoność obliczeniowa operacji wstawiania, wyszukiwania i usuwania elementu jest stała, czyli O(1).



Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Dobór funkcji haszującej

O skuteczności haszowania decyduje dobór funkcji haszującej.

Dobrze dobrana funkcja haszująca powinna spełniać (w przybliżeniu) założenie prostego równomiernego haszowania: losowo wybrany klucz jest z jednakowym prawdopodobieństwem odwzorowywany na każdą z *m* pozycji tablicy, niezależnie od tego, gdzie zostają odwzorowane inne klucze.

Nie jest to, w ogólnym przypadku, warunek możliwy do spełnienia, ponieważ najczęściej nie jest znany rozkład prawdopodobieństwa pojawiania się kluczy. Nie jest też gwarantowana niezależność pojawiania się kluczy.

W praktyce przy doborze funkcji haszujących stosuje się rozmaite metody heurystyczne, które w powiązaniu z wiedzą o zawartości słownika (charakterze informacji) są bardzo użyteczne. Dobrze dobrana funkcja haszująca powinna np. minimalizować szansę, że niewiele różniące się między sobą symbole będą odwzorowane na tę samą pozycję w tablicy z haszowaniem.

Typowe podejście polega na takim doborze wartości funkcji haszującej, aby były one maksymalnie niezależne od możliwych wzorców mogących występować w danych. Przykład takiego podejścia: haszowanie modularne, w którym wartość fukcji haszującej jest równa reszcie z dzielenia wartości klucza przez pewną ustaloną liczbę pierwszą (np. większą od liczby elementów w tablicy).

Można np. starać się, by "bliskim sobie" wartościom kluczy odpowiadały znacznie od siebie odległe wartości funkcji haszującej (co jest ważne np. w rozwiązywaniu kolizji matodą adresowania liniowego).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przykłady funkcji haszujących

1. Utożsamianie wartości kluczy z liczbami naturalnymi

Idea: Jeśli wartości kluczy nie są liczbami naturalnymi, należy je odwzorować w zbiór liczb naturalnych. Przykłady:

- kolejne znaki ciągu znaków (z podstawowego kodu ASCII, a więc kody do 127) można odwzorować w liczby i zapisać je w systemie liczbowym o podstawie 128 ("AS" => 65*128 + 83 = 8403, "as" => 97*128 + 115 = 12531).
- kolejne znaki ciągu znaków (z podstawowego kodu ASCII, a więc kody do 127) można odwzorować w liczby i zapisać je jako sumę: ("OSA" => 79+ 83 + 65 = 227, "ASO" => 65 + 83 + 79 = 227; mamy przykład kolizji wartości kluczy).

2. Haszowanie modularne

Idea: Dla wartości liczby naturalnej *k* odpowiadającej wartości klucza wyznaczamy wartość h(k) = k mod m (m – liczba określona na podstawie MAX - liczby pozycji w tablicy z haszowaniem; m>=MAX).

Należy unikać wartości m, które są potęgą 2. Najlepiej stosować liczby pierwsze niezbyt bliskie potęgom 2.

3. Haszowanie dwukrotne

Idea: Rozwiązywać lepiej kolizje, występujące w adresowaniu otwartym, wykorzystując losowość permutacji adresów w tablicy z haszowaniem. Funkcja haszująca ma ogólna postać: h(k,i) = (h1(k) + i*h2(k)) mod m. Kolejne pozycje oddalone są od początkowej o h2(k) mod m (czyli nie są to kolejne pozycje).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przykłady funkcji haszujących – c.d.

4. Haszowanie przez mnożenie

Idea: Wykorzystuje się (jako podstawę funkcji haszującej), część ułamkową wyrażenia k*A, gdzie A jest stałą z przedziału (0,1):

h(k) = część całkowita (m * część ułamkowa (k*A))

Zaletą tej metody jest mała zależność h(k) od m.

W literaturze można znaleźć różne przykłady "szczególnie dobrych" wartości A (np. Knuth podaje A = 0.6180339887...)

5. Haszowanie uniwersalne

Idea: Zastosować "losowość" w dobieraniu funkcji haszującej z rodziny funkcji haszujących (zbiorze funkcji haszujących, zwanej rodziną uniwersalną funkcji) o pewnych szczególnych własnościach, by zapewnić małe oczekiwane czasy działania operacji na tablicach z haszowaniem.

6. Haszowanie doskonałe

Idea: Zastosować dwupoziomowe haszowanie uniwersalne, by całkowicie uniknąć konfliktów i zapewnić stałą liczbę odwołań do tablicy z haszowaniem nawet dla przypadku pesymistycznego [Cormen, rozdział 11].

Haszowanie pozwala zbliżyć się do stałej średniej złożoności wszystkich operacji wykonywanych na słowniku (czyli do tego, co oferuje tablica z adresowaniem bezpośrednim).