Tablice z haszowaniem

Przykład

W dalszym ciągu operujemy na parze (klucz, wartość).

Załóżmy, że mamy tablicę *T*, w której:

- Indeks jest kluczem - i
- W tablicy pod tym indeksem jest wartość, czyli *T*[*i*] zawiera wartość skojarzoną z *i*.

Mamy zatem zapewniony bezpośredni dostęp do wartości poprzez klucz, ale … uwaga: równowaga *czasu* i *pamięci.*

Tablice z haszowaniem

Haszowanie

Metoda, w której pary (klucz, wartość) w tablicach wskazywane są na podstawie operacji arytmetycznych przekształcających klucze w indeksy tablicy.

Pozwala ograniczyć do rozsądnej ilości potrzebny czas i pamięć. W algorytmach haszowania można zyskać czas kosztem pamięci (i na odwrót) dostosowując odpowiednio parametry, nie modyfikując kodu.

Tablice z haszowaniem

Algorytmy wyszukiwania oparte na haszowaniu składają się generalnie z dwóch kroków:

1. Obliczenie wartości *funkcji haszującej*, która przekształca klucz wyszukiwania na skrót wyznaczający indeks tablicy (zakłada się/ dąży się to sytuacji, że różne klucze odpowiadają różnym indeksom, ale nie zawsze tak jest).
2. Rozwiązywanie ew. kolizji
   a) Haszowanie metodą łańcuchową
   b) Haszowanie z wykorzystaniem próbkowania liniowego

Tablice z haszowaniem

Funkcje haszujące

Zakładając, że istnieje tablica mieszcząca *M* par (klucz, wartość), potrzebna jest *funkcja haszująca*, która potrafi przekształcić dowolny klucz na indeks tej tablicy, czyli [0, *M*-1].

Szukamy zatem funkcji, która byłaby łatwa do obliczenia i zapewniałaby równomierny rozkład kluczy. Czyli dla każdego klucza wystąpienie dowolnej liczby całkowitej z przedziału 0 do *M*-1 powinno być tak samo prawdopodobne.

Tablice z haszowaniem

Funkcje haszujące

Funkcja haszująca zależy od typu klucza, czyli dla każdego używanego typu klucza potrzebna jest inna funkcja haszująca, przykładowo:

- Numer albumu czy też PESEL można zwykle użyć bezpośrednio
- Nazwisko należałoby zamienić najpierw na liczbę
- Inne typy złożone, składające się z kilku pól, np. adres pocztowy – trzeba je w pewien sposób połączyć

Tablice z haszowaniem

Funkcje haszujące

Dodatnie liczby całkowite

Najbardziej popularny sposób (zwany *haszowaniem modularnym*):

Wybieramy rozmiar tablicy *M* – liczba pierwsza, a następnie dla dowolnej liczby dodatniej całkowitej k obliczamy resztę z dzielenia *k* przez *M* (w Javie: *k* % *M*).

Funkcja jest skuteczna w rozłożeniu kluczy równomiernie pomiędzy 0 a *M*-1.

Jeśli *M* nie byłaby liczbą pierwszą – niemożliwym może stać się równomierny podział wartości.

Tablice z haszowaniem

Funkcje haszujące

Liczby zmiennoprzecinkowe

Jeśli kluczami są liczby rzeczywiste z przedziału od 0 do 1, można pomnożyć je przez *M* i zaokrąglić do najbliższej liczby całkowitej, aby uzyskać indeks z przedziału od 0 do *M*-1.

Wadą jest to, iż nadaje większą wagę znaczącym bitom klucza, te mniej znaczącą nie liczą się. Obejściem może być wykorzystanie haszowania modularnego na binarnej reprezentacji klucza (zastosowano w Javie).

Tablice z haszowaniem

Funkcje haszujące

Łańcuchy znaków

Wykorzystujemy fakt, iż łańcuch znaków traktowany może być jako duża liczba całkowita.

Przykład obliczenia wartości funkcji haszującej dla łańcucha *s* typu String:

```
int h = 0;
for (int i = 0; i < s.length(); i++)
    h = (R * h + s.charAt(i)) % M;
```

charAt(i) – zwraca wartość typu char, czyli 16-bitową liczbę całkowitą.

R – mała liczba pierwsza (w Javie jest to 31).

Tablice z haszowaniem

Funkcje haszujące

Klucze złożone

Jeśli klucz obejmuje kilka pól całkowitoliczbowych, zwykle możemy je połączyć podobnie jak przy łańcuchach znaków.

Załóżmy, że mamy typ `Date`, obejmujący trzy pola `day` (dwie cyfry), `month` (dwie cyfry) i `year` (cztery cyfry). Wartość obliczamy jako:

```
int h = (((day * R + month) % M) * R + year) % M
```

Uzyskana wartość to liczba z całkowita z przedziału od 0 do *M*-1.

Tablice z haszowaniem

Funkcje haszujące

Implementacja w Javie

Wszystkie klasy w Javie dziedziczą metodę `int hashCode()`, która zwraca liczbę 32 bitową.

- Wymóg: jeśli `x.equals(y)`, to `(x.hashCode()==y.hashCode())`.
- Pożądane: !`x.equals(y)`, to `(x.hashCode()!=y.hashCode())`.
- Domyślnie zwraca adres maszynowy obiektu x.
- Dla typów `Integer, Double, String, File, URL, Date`, …, Java udostępnia implementacje metody `hashCode()`

Tablice z haszowaniem

Funkcje haszujące

Implementacja w Javie

Jak przekształcić wartość funkcji `hashCode()` na indeks tablicy z przedziału od 0 do *M*-1?

Łączymy wartość funkcji `hashCode()` z haszowaniem modularnym:

```
private int hash(Key key) {
   return (key.hashCode() & 0x7fffffff) % M;
}
```

Tablice z haszowaniem

Funkcje haszujące

Implementacja w Javie

`hashCode()` definiowana przez użytkownika

Oczekuje się, że funkcja `hashCode()` rozkłada równomiernie pomiędzy możliwe 32-bitowe wartości. Oznacza to, że dla dowolnego obiektu `x` można zapisać `x.hashCode()` oczekiwać z równym prawdopodobieństwem jednej z 2^{32} możliwych 32-bitowych wartości.

Tablice z haszowaniem

Funkcje haszujące

Implementacja w Javie

`hashCode()` definiowana przez użytkownika

Trzy podstawowe wymagania przy implementacji dobrej funkcji haszującej dla określonego typu danych:

- *Determinizm* – równe klucze muszą dawać taką samą wartość funkcji haszującej.
- *Działanie wydajne*.
- *Równomiernie rozdzielanie kluczy*.

Tablice z haszowaniem

Funkcje haszujące

Założenie o równomiernym haszowaniu

Funkcje haszujące równomiernie i niezależnie rozdzielają klucze między całkowitoliczbowe wartości z przedziału od 0 do *M*-1.

Tablice z haszowaniem

Haszowanie metodą łańcuchową (*ang. separate chaining*)

Drugim krokiem algorytmów haszowania jest rozwiązywanie kolizji, czyli sytuacji, w której dla dwóch lub więcej kluczy funkcja haszująca wskazuje na ten sam indeks w tablicy.

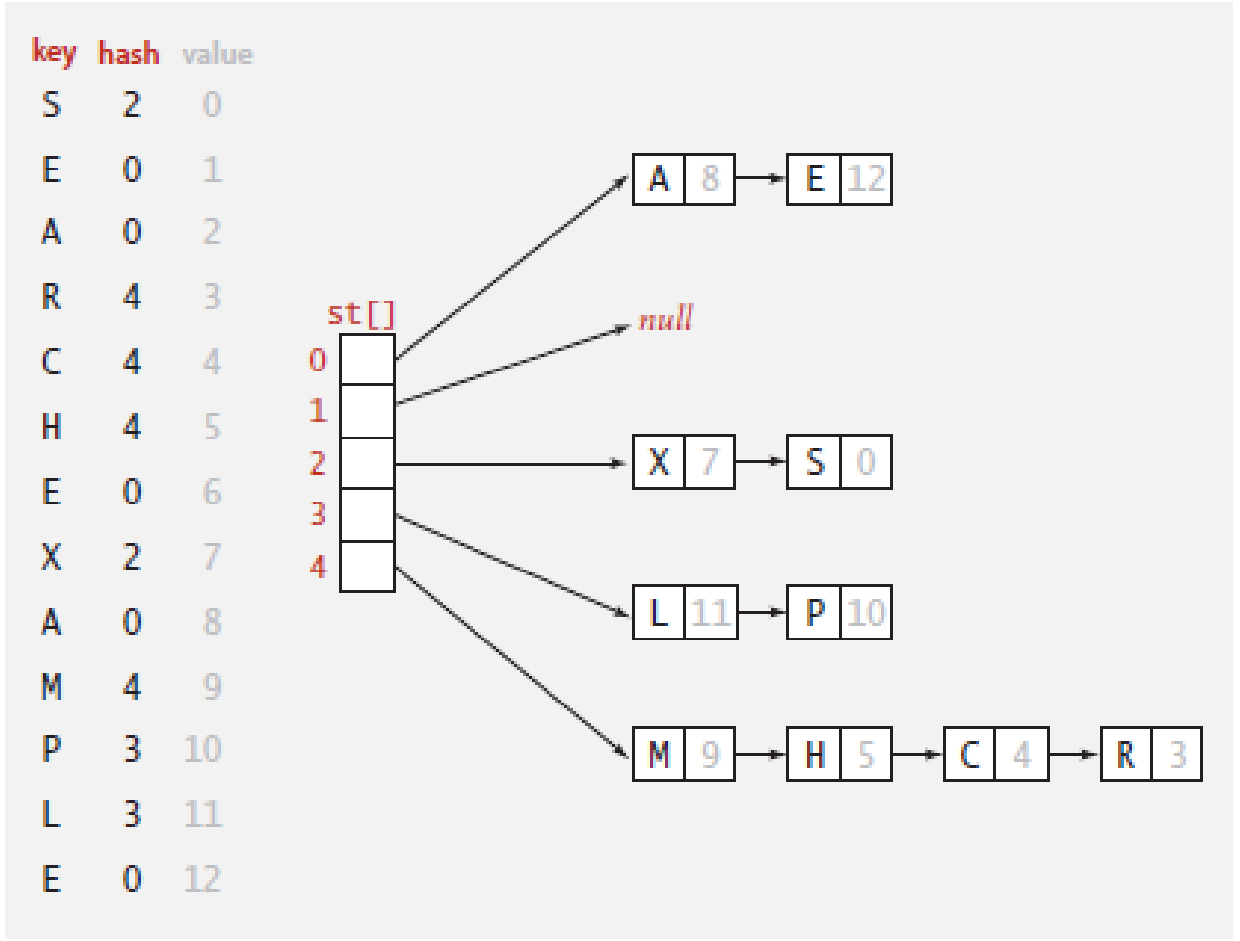
Tablice z haszowaniem

Haszowanie metodą łańcuchową

Proste rozwiązanie:

- Utrzymujemy tablicę *M* elementową (indeksy: 0 ... *M*-1) do przechowania *N* par (klucz, wartość).
- Dla każdego indeksu *i* utrzymujemy listę powiązaną par, dla których funkcja haszująca zwraca indeks *i*.

Jeżeli dla danego klucza znajdziemy indeks pozostanie do przeszukania tylko lista (łańcuch) przypisana do tego indeksu.



[SW-2018]

Tablice z haszowaniem

Haszowanie metodą łańcuchową

Implementacja

```
public class SeparateChainingHashST <Key, Value>
{
   private int M = 97;
   private Node[] st = new Node[M];

   private static class Node {
      private Object key;
      private Object val;
      private Node next;
   }

   private int hash(Key key) {
      return (key.hashCode() & 0x7fffffff) % M;
   }

   public Value get(Key key) {
   }

   public void put(Key key, Value val) {
   }
}
```

Tablice z haszowaniem

Haszowanie metodą łańcuchową

Implementacja

```
public class SeparateChainingHashST <Key, Value>
{
   …
   public Value get(Key key) {
      int i = hash(key);
      for (Node x = st[i]; x != null; x = x.next)
         if (key.equals(x.key))
            return (Value) x.val;
      return null;
   }

   public void put(Key key, Value val) {
      int i = hash(key);
      for (Node x = st[i]; x != null; x = x.next)
         if (key.equals(x.key)) {
            x.val = val;
            return;
         }
      st[i] = new Node(key, val, st[i]);
   }
   …
}
```

Tablice z haszowaniem

Haszowanie metodą łańcuchową

Implementacja

```
public class SeparateChainingHashST <Key, Value>
{
   …
   public Value delete(Key key) {
        …
   }



   public int size() {
        …
   }

}
```

Tablice z haszowaniem

Haszowanie metodą łańcuchową

Jakie wybrać M?

- Jeżeli *M* za duże ⇒ zbyt wiele pustych łańcuchów.
- Jeżeli *M* za małe ⇒ łańcuchy są zbyt długie.
- Typowy wybór: $M \sim N / 4$.

Tablice z haszowaniem

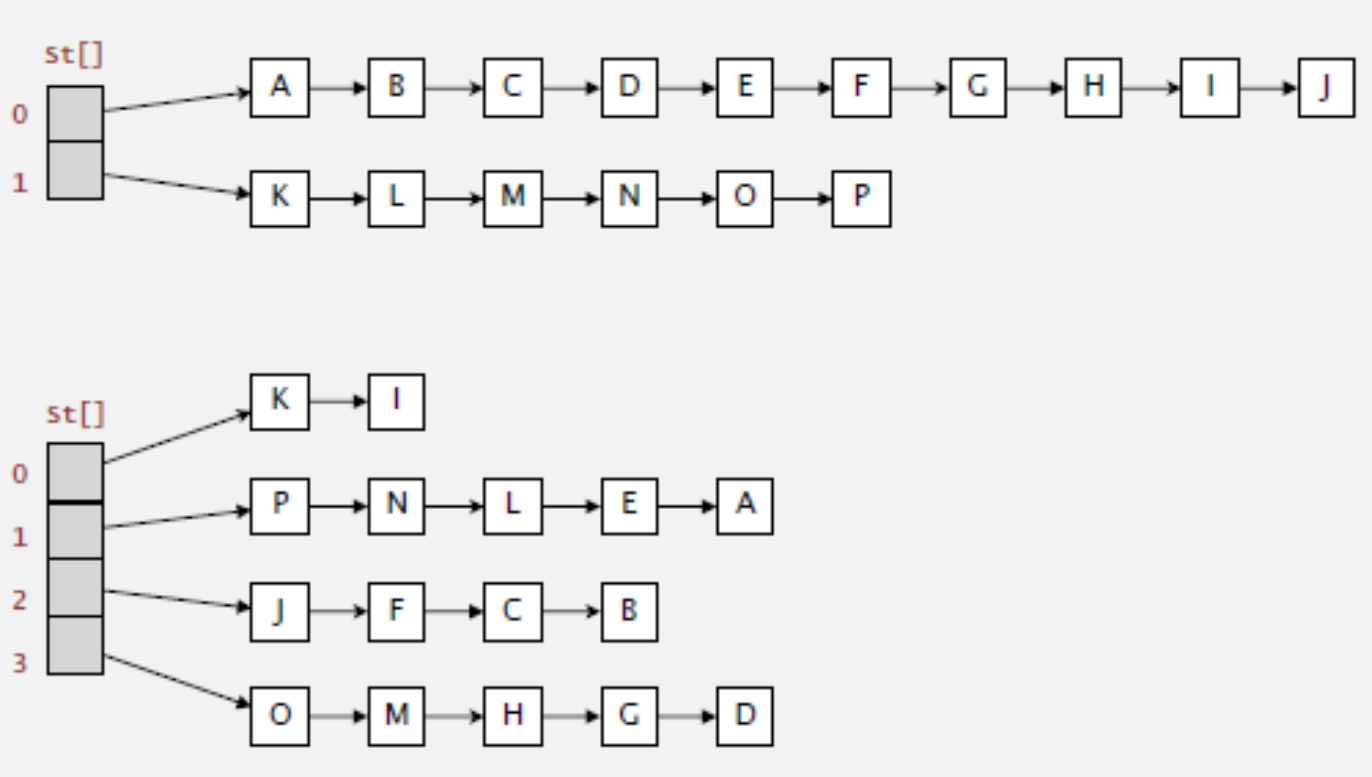
Haszowanie metodą łańcuchową

Celem jest aby średnia długość listy N/M była stała.

Zmiana wielkości tablicy:

- Podwajamy wielkość tablicy M gdy $N/M \geq 8$.
- Zmniejszamy dwukrotnie wielkość tablicy M gdy $N/M \leq 2$.
- Uwaga: wymagane jest re-haszowanie wszystkich kluczy.

przed



po

[SW-2018]

Tablice z haszowaniem

Implementacja	WORST CASE Wyszukanie	WORST CASE Wstawienie	WORST CASE Usuwanie	AVG CASE Wyszukanie	AVG CASE Wstawienie	AVG CASE Usuwanie	Porównanie kluczy
Przeszukiwanie sekwencyjne w tablicy symboli (lista powiązana nieuporządkowana)	*N*	*N*	*N*	*N/2*	*N*	*N/2*	`equals()`
Przeszukiwanie binarne w tablicy symboli (tablica uporządkowana)	log *N*	*N*	*N*	log *N*	*N/2*	*N/2*	`compareTo()`
Drzewa wyszukiwań binarnych	*N*	*N*	*N*	1.39 log *N*	1.39 log *N*	$\sqrt{N}$	`compareTo()`
Drzewa 2-3 Czerwono-czarne BST	2 log *N*	2 log *N*	2 log *N*	1.00 log *N*	1.00 log *N*	1.00 log *N*	`compareTo()`
Haszowanie metodą łańcuchową	*N*	*N*	*N*	3-5	3-5	3-5	

Tablice z haszowaniem

Haszowanie metodą liniowego próbkowania (*ang. linear probing*)

- Utrzymujemy tablicę *M* elementową (indeksy: 0 ... *M*-1) do przechowania *N* par (klucz, wartość), gdzie $M > N$.
- Dla każdego indeksu *i* utrzymujemy listę powiązaną par (klucz, wartość), dla których funkcja haszująca zwraca indeks *i*.
- Zakładamy, że puste miejsca w tablicy wykorzystamy przy rozwiązywaniu kolizji, czyli jeśli stwierdzimy kolizję, szukamy następnego indeksu (dokonujemy jego zwiększenia).

Tablice z haszowaniem

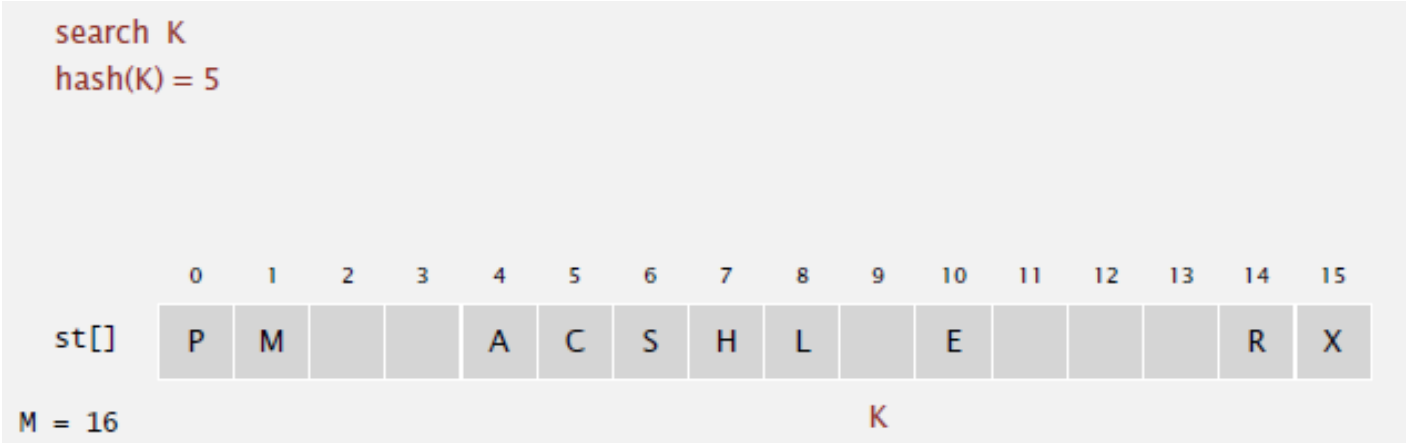
Haszowanie metodą liniowego próbkowania

Mamy trzy możliwości:

- Klucz jest równy kluczowi wyszukiwania – wyszukiwanie udane
- Pozycja jest pusta, czyli na pozycji o danym indeksie jest null – wyszukiwanie jest nieudane
- Klucz nie jest równy kluczowi wyszukiwania – należy sprawdzić następną pozycję.

Tablice z haszowaniem

Haszowanie metodą liniowego próbkowania



[SW-2018]

Tablice z haszowaniem

Haszowanie metodą liniowego próbkowania

Implementacja

```
public class LinearProbingHashST <Key, Value>
{
   private int M;
   private Value[] vals = (Value[]) new Object[M];
   private Key[] keys = (Key[]) new Object[M];

   private int hash(Key key) {
   }

   public Value get(Key key) {
   }

   public void put(Key key, Value val) {
   }
}
```

Tablice z haszowaniem

Haszowanie metodą liniowego próbkowania

Implementacja

```
public class LinearProbingHashST <Key, Value>
{
…
   public Value get(Key key) {
      for (int i = hash(key); keys[i] != null;
            i = (i+1) % M)
         if (key.equals(keys[i]))
            return vals[i];
      return null;
   }

   public void put(Key key, Value val) {
      int i;
      for (i = hash(key); keys[i] != null;
            i = (i+1) % M)
         if (keys[i].equals(key))
            break;
      keys[i] = key;
      vals[i] = val;   }
   …
}
```

Tablice z haszowaniem

Haszowanie metodą liniowego próbkowania

Celem jest aby średnia długość listy $N/M <= ½$.

Zmiana wielkości tablicy:

- Podwajamy wielkość tablicy M gdy $N/M \geq ½$.
- Zmniejszamy dwukrotnie wielkość tablicy M gdy $N/M \leq 1/8$.
- Uwaga: wymagane jest re-haszowanie wszystkich kluczy.