Hashing

Svarny Petr

Katedra logiky FF UK

2. května 2021

Overview

Rozptylování (Hashing)

- Výpočet funkce
- Řešení kolizí
- Srůstající hashování
- Dynamické hashování

Python a hashing

Rozptylování (Hashing)

Definice: Hashovací funkce

Hašovací funkce je funkce, která převádí vstupní data na nějaké specificky dané hodnoty. Pro nás zejména převod klíče na adresu, tj. $H : K \rightarrow A$

Pokud se vytvořená čísla shodují pro dva vstupy, dochází k tzv. **kolizi**.

Hashovací funkce, ve které nedochází ke kolizi se nazývá **perfektní**.

Např. modulo může sloužit jako jednoduchá hashovací funkce.

Hlavní užití pro hašovací tabulky, šifrování nebo otisky/signatury (např. rodná čísla od roku 1954 mají kontrolní hodnotu mod $11 = 0$)

Převzato z kurzu na =ČVUT=.

Různá vyhledávání

Mám tabulku s klíči a obsahem.

- Porovnávání klíčů, "asociativní" (sekvenční, BVS), $\Omega(logn)$
- Indexování klíčem (přímý přístup), klíč je i adresa, ale rozsah klíčů je rozsah indexů, $\Theta(1)$
- Rozptylování, tj. výpočet adresy z hodnoty klíče, $\Theta(1)$

Rozptylování je kompromisem rychlosti a spotřeby paměti.

Rozptylování

Použito pokud množina použitých klíčů $<<$ univerzum všech možných klíčů.

- Konstantní čas pro vyhledání a vložení.
- Čas provádění odpovídá délce klíče.
- Nevhodné pro řazení a výběr podmnožin.

1. Výpočet rozptylovací funkce $h(k)$.
2. Řešení kolizí.

Výpočet rozptylovací funkce

- Výpočetně co nejjednodušší,
- Aproximuje náhodnou funkci,
- Využije rovnoměrně adresní prostor,
- **Generuje minimum kolizí**.

Příklady:

- **multiplikativní**: $h(k, M) = zaokrouhli(k \cdot M)$, M velikost tabulky, $k \in \mathbb{R}$
- **modulární**: $h(k, M) = k \mod M$, $k \in \mathbb{Z}$

Univerzální hashování

- Místo jedné funkce $h(k)$ použijeme konečnou množinu H mapujících U do tabulky s m indexy $(\{0, 1, \ldots, m-1\})$,
- Vybrat jednu $h \in H$ náhodně,
- Množina H je univerzální, pokud pro různé $x, y \in U : h(x) = h(y)$ přesně v $\frac{|H|}{m}$ případech,
- Tedy pravděpodobnost kolize je $\frac{1}{m}$

Kolize

Např. $11 \mod 10 = 1 \mod 10$. Co s tím?

Zřetězené rozptylování (Chaining)

Vytváříme lineární seznamy synonym.
$h(k) = k \mod 3$
$k = \{1, 5, 21, 10, 7\}$

- 0 : (21, -)
- 1 : (7, $10), (10, $1), (1,-)
- 2 : (5, -)

Kolidující prvky se vkládají na začátek.

Zřetězené rozptylování (Chaining)

Podle toho naimplementovat hledání, vkládání a odstranění.
n počet prvků, m velikost tabulky, $m < n$
$\alpha = \frac{n}{m}$ plnění tabulky

- Vkládání: $O(1)$
- Hledání: $O(n)$, v průměru ale $O(1 + \alpha)$
- Odstranění: $O(n)$, v průměru ale $O(1 + \alpha)$

Zřetězené rozptylování (Chaining)

Podle toho naimplementovat hledání, vkládání a odstranění.
n počet prvků, m velikost tabulky, $m < n$
$\alpha = \frac{n}{m}$ plnění tabulky

$m \in < n/5; n/10 >$, sekvenční hledání se ještě vyplatí a neplýtváme nepoužitými ukazateli.

Shrnutí

Řešení kolizí pomocí řetězení kolidujících prvků, kde nemusíme znát n dopředu, ale potřebuje dynamické přidělování paměti, paměť na ukazatele a na tabulku.

Otevřené rozptylování (Open-address hashing)

Posloupnost do pole pokud známe (odhad) n.
Podle tvaru hashovací funkce se při kolizi:

- lineární prohledávání (linear probing)
- dvojí rozptylování (double hashing)

Otevřené rozptylování (Open-address hashing)

Volíme modulo podle velikosti pole.
$h(k) = k \mod 5$
$k = \{1, 5, 21, 10, 7\}$
Kolize pro 21:

0	5
1	1
2	
3	
4	

Linear prohledávání

Volíme modulo podle velikosti pole.
$h(k) = k \mod 5$
$k = \{1, 5, 21, 10, 7\}$
Kolize pro 21, tak umísti na pozici $k + 1 \mod 5$

0	5
1	1
2	21
3	
4	

Linear prohledávání

Volíme modulo podle velikosti pole.
$h(k) = k \mod 5$
$k = \{1, 5, 21, 10, 7\}$
Kolize pro 10, tak umísti na pozici $k + 3 \mod 5$ neboť

0	5 (blokuje)
1	1 (blokuje)
2	21 (blokuje)
3	10
4	

Linear prohledávání

Volíme modulo podle velikosti pole.

$h(k) = k \mod 5$

$k = \{1, 5, 21, 10, 7\}$

Kde skončí 7 a proč?

0	5
1	1
2	21
3	10
4	

Linear prohledávání

Volíme modulo podle velikosti pole.

$h(k) = k \mod 5$

$k = \{1, 5, 21, 10, 7\}$

Výsledek i s tím, nakolik se posunuly:

0	5 ($i = 0$)
1	1 ($i = 0$)
2	21 ($i = 1$)
3	10 ($i = 3$)
4	7 ($i = 2$)

Šlo by to nějak udělat lépe, tj. menší/méně posunů, resp. shluků?

Linear prohledávání

Volíme modulo podle velikosti pole.
$h(k) = k \mod 5$
$k = \{1, 5, 21, 10, 7\}$
Při kolizi: $(h(k) + i * d) \mod 5$, d je nesoudělné s m (tedy pokryjeme celou tabulku), např. $d = 3$ pro nás.

0	5 ($i = 0$)
1	1 ($i = 0$)
2	7 ($i = 0$)
3	10 ($i = 1$)
4	21 ($i = 1$)

Viz také =visualgo.net=
Nebo vizualizace =jina vizualizace=

Dvojí rozptylování (Double hashing)

Kolize pro 21:
$h(21) = [21 \mod 5 + 1 \cdot (1 + (21 \mod 4))] \mod 5 =$
$[1 + (1 + 1)] \mod 5 = 3$

0	5
1	1
2	
3	21
4	

Dvojí rozptylování (Double hashing)

Kolize pro 10:

$h(10) = [10 \mod 5 + 1 \cdot (1 + (10 \mod 4))] \mod 5 =$
$[0 + (1 + 2)] \mod 5 = 3$

$h(10) = [10 \mod 5 + 2 \cdot (1 + (10 \mod 4))] \mod 5 =$
$[0 + 2 \cdot (1 + 2)] \mod 5 = 1$

$h(10) = [10 \mod 5 + 3 \cdot (1 + (10 \mod 4))] \mod 5 =$
$[0 + 3 \cdot (1 + 2)] \mod 5 = 4$

0	5
1	1
2	
3	21
4	10

Dvojí rozptylování (Double hashing)

Volíme modulo podle velikosti pole.
$h(k) = [h_1(k) + i \cdot h_2(k)] \mod m$
$h_1(k) = k \mod m$
$h_2(k) = 1 + (k \mod m')$ offset
m je prvočíslo potom m' je menší číslo (např. 13 a 12).
m je mocnina 2, potom m' je liché číslo.
$k = \{1, 5, 21, 10, 7\}$

0	5, $i = 0$
1	1, $i = 0$
2	7, $i = 0$
3	21, $i = 1$
4	10, $i = 3$

Otevřené rozptylování

Shrnuti

Zápis do pole a řešení kolizí pomocí umístění jinam - lineárně nebo pomocí druhotného rozptylování. Je třeba znát alespoň odhad počtu prvků. Druhotné rozptylování funguje efektivněji při vyšším plnění ($\alpha > 3/4$) a tedy stačí menší tabulka.

Porovnání otevřeného rozptylování

$\alpha = n/m$, kde m je velikost tabulky a n je počet prvků

Očekávaný počet testů (probes)

- Lineárně
  - Nalezen klíč $0.5(1 + 1/(1-\alpha))$
  - Nenalezen $0.5(1 + 1/(1-\alpha)^2)$
- Dvojitě
  - Nalezen klíč $(1/\alpha)\ln(1/(1-\alpha))$
  - Nenalezen $1/(1-\alpha)$

Porovnání otevřeného rozptylování

Lineárně

α	1/2	2/3	3/4	9/10
Search hit	1.5	2.0	3.0	5.5
Search miss	2.5	5.0	8.5	55.5

Dvojitě

α	1/2	2/3	3/4	9/10
Search hit	1.4	1.6	1.8	2.6
Search miss	2.0	3.0	4.0	10.0

Srůstající hashování (coalesced hashing)

Tabulka obsahuje nejen klíč, ale i ukazatel na další klíč v tabulce. Taktéž máme ukazatel na poslední volné místo. Každý klíč je v tabulce a je součástí nějakého seznamu synonym.

0	K1	-
1	K3	3
→2	-	-
3	K2	-

Jaký je rozdíl proti zřetězenému rozptylování?

LISCH (Late Insert Standard Coalesced Hashing)

$h(k) = k \mod 5$
$k = \{1, 5, 21, 10, 7\}$

0	-	-
1	-	-
2	-	-
3	-	-
$\rightarrow$4	-	-

Kolize zapíšeme na poslední volné místo v tabulce a na konec seznamu synonym. Posuneme ukazatel.

LISCH (Late Insert Standard Coalesced Hashing)

$h(k) = k \mod 5$
$k = \{1, 5, 21, 10, 7\}$

0	-	-
1	1	-
2	-	-
3	-	-
→4	-	-

LISCH (Late Insert Standard Coalesced Hashing)

$h(k) = k \mod 5$
$k = \{1, 5, 21, 10, 7\}$

0	5	-
1	1	-
2	-	-
3	-	-
→4	-	-

LISCH (Late Insert Standard Coalesced Hashing)

$h(k) = k \mod 5$
$k = \{1, 5, 21, 6, 10\}$

0	5	-
1	1	4
2	-	-
$\rightarrow$ 3	-	-
4	21	-

Kolize 21, zapíšeme na poslední volné místo v tabulce a na konec seznamu synonym. Posuneme ukazatel.

LISCH (Late Insert Standard Coalesced Hashing)

$h(k) = k \mod 5$
$k = \{1, 5, 21, 6, 10\}$

0	5	3
1	1	4
$\rightarrow$ 2	-	-
3	6	-
4	21	3

Podobně s 10.

LISCH (Late Insert Standard Coalesced Hashing)

$h(k) = k \mod 5$
$k = \{1, 5, 21, 6, 10\}$

0	5	2
1	1	4
2	10	-
3	6	-
4	21	3

Podobně s 10.

EISCH (Early Insert Standard Coalesced Hashing)

$h(k) = k \mod 5$
$k = \{1, 5, 21, 6, 10\}$

0	-	-
1	-	-
2	-	-
3	-	-
$\rightarrow$4	-	-

Jak bude asi fungovat EISCH?

EISCH (Early Insert Standard Coalesced Hashing)

$h(k) = k \mod 5$
$k = \{1, 5, 21, 6, 10\}$

0	5	-
1	1	4
2	-	-
$\rightarrow$3	-	-
4	21	-

Jak bude asi fungovat EISCH?

EISCH (Early Insert Standard Coalesced Hashing)

$h(k) = k \mod 5$
$k = \{1, 5, 21, 6, 10\}$

0	5	-
1	1	3
→2	-	-
3	6	4
4	21	-

Stejně jako LISCH, ale nové synonymum se řadí na začátek seznamu synonym.

EICH, LICH, VICH

Máte sklep? A mohla bych ho vidět?

EICH, LICH, VICH

$h(k) = k \mod 5$
$k = \{1, 5, 21, 6, 10\}$

0	-	-
1	-	-
2	-	-
3	-	-
4	-	-
5	-	-
$\rightarrow$ 6	-	-

Pomocná paměť, tzv. sklep. LICH a EICH jen plní od sklepa ale fungují jinak stejně. Zkuste si LICH...

EICH, LICH, VICH

$h(k) = k \mod 5$
$k = \{1, 5, 21, 6, 10\}$

0	5	4
1	1	6
2	-	-
$\rightarrow$ 3	-	-
4	10	-
5	6	-
6	21	5

Co je tedy výhoda?

EICH, LICH, VICH

$h(k) = k \mod 5$
$k = \{1, 6, 11, 16, 21\}$

0	-	-
1	-	-
2	-	-
3	-	-
4	-	-
5	-	-
$\rightarrow$ 6	-	-

Variable ICH - za poslední prvek ve sklepě, jinak jako EISCH. Alias připojuje prvek na konec řetězce, pokud řetězec končí ve sklepě, jinak na místo, kde řetězec opustil sklep.

EICH, LICH, VICH

$h(k) = k \mod 5$
$k = \{1, 6, 11, 16, 21\}$

0	-	-
1	1	6
$\rightarrow 2$	-	-
3	21	4
4	16	-
5	11	3
6	6	5

Variable ICH - za poslední prvek ve sklepě, jinak jako EISCH. Alias připojuje prvek na konec řetězce, pokud řetězec končí ve sklepě, jinak na místo, kde řetězec opustil sklep.

ICH

Podle průměrného počtu navštívených klíčů (probes) je bez sklepa nejlepší EISCH.

Se sklepem je počet navštívených klíčů obecně menší a nejméně pro VICH.

Velikostí sklepa je možno zvýšit účinnost rozptylování. Poměr $\frac{M}{M'}$, kde M je počet adresovatelných míst v tabulce a M' je celkový počet míst v tabulce by je doporučen na 0.86.

Dynamické hashování

Inkrementální
Rozšiřitelné (Extendible)

Inkrementální dynamické hashování

Prosté: Tabulku zvětšíme a přehashujeme.
Postupné: Alokujeme novou tabulku kam zapisujeme nové prvky. Ve staré jen hledáme a rušíme. Při každé operaci přepíšeme prvek ze staré do nové tabulky a když je prázdná, tak ji zrušíme.

Rozšiřitelné dynamické hashování

Hash vrací binární číslo, např. $h(k) = 11100$.

Hashovací tabulka má adresář co zohledňuje určitou délku hashe (prefix) a ty odkazují na záznamy (kýble, buckets). Když kýbl přeteče, tak se začne přeorganizovávat.

Viz take =emory.edu=

Rozšiřitelné dynamické hashování

$h(k) = bin(k)$
$k = \{1, 21, 6, 16, 11\}$ $h(k) = \{00001, 10101, 01011, 10000, 00110\}$
Začneme na nějaké délce prefixu, třebas 1:

0 Záznam A
- 00001

1 Záznam B
- -

Rozšiřitelné dynamické hashování

$h(k) = \{00001, 10101, 01011, 10000, 00110\}$
Začneme na nějaké délce prefixu (zvaná hloubka), třebas 1:

0 Záznam A
- 00001

1 Záznam B
- 10101

Rozšiřitelné dynamické hashování

$h(k) = \{00001, 10101, 01011, 10000, 00110\}$
Podstatná je velikost kýblů, mohou i držet veškeré záznamy.

0 Záznam A
  - 00001
  - 01011
1 Záznam B
  - 10101

Rozšiřitelné dynamické hashování

$h(k) = bin(k)$
$k = \{1, 21, 6, 16, 11\}$ $h(k) = \{00001, 10101, 01011, 10000, 00110\}$
Nebo můžeme rozšířit záznamy pokud už dosáhly kapacity:

00 Záznam A
- 00001

01 Záznam B
- 01011

10 Záznam C
- 10101

11 Záznam D
- -

Rozšiřitelné dynamické hashování

$h(k) = \{00001, 10101, 01011, 10000, 00110\}$
U záznamů se také sleduje lokální hloubka. Pokud dojde k přetečení a lokální = globální hloubka, je třeba zvětšit globální hloubku a vše předělat.

00 Záznam A (2)

- 00001

01 Záznam B (2)

- 01011

1 Záznam C (1)

- 10101

Rozšiřitelné dynamické hashování

$h(k) = \{00001, 10101, 01011, 10000, 00110\}$
U záznamů se také sleduje lokální hloubka. Pokud dojde k přetečení a lokální = globální hloubka, je třeba zvětšit globální hloubku a vše předělat.

00 Záznam A (2)
- 00001

01 Záznam B (2)
- 01011

10 Záznam C (2)
- 10101

11 Záznam D (2)
- 10000

Python a hashing

Python dict je hash-table (viz =dokumentace= nebo =blog=.)

Je to speciální metoda objektu a je ji možno předefinovat (viz =programviz=), jinak odpovídá číslu.

Tj. hash pro objekty jako str nebo tuple je definována kódu (kompozičně, viz =stackoverflow=).