Árásir á hakkatöflur

Páll Melsted

Iðnaðarverkfræði-, Vélaverkfræði- og Tölvunarfræðideild

Háskóli Íslands

Hvað er hakkatafla

Gagnagrindin á bak við Dictionary, Map,...

Java: HashMap

Python: dict(), {}

Ruby: Hash

C#: Dictionary

Objective-C: NSDictionary

C++11: unordered_map

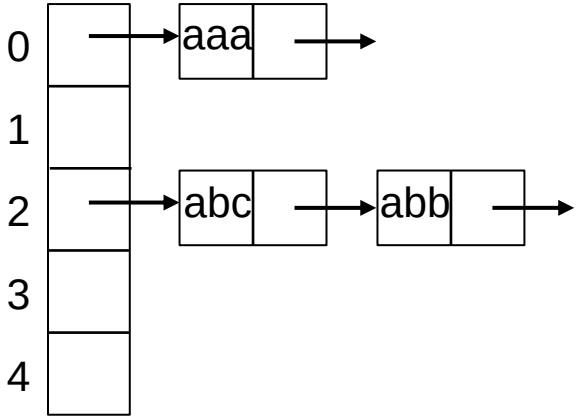
Hakkafall tekur inn streng og skilar heiltölu

Hakkataflan er fylki af tengdum listum, hakkafallið segir til í hvaða lista á að leita

h(“abc”) = 42, geymdur í hólfi 42 mod 5 = 2

h(“abb”) = 12, fer líka í hólf 2

h(“aaa”) = 30, geymt í hólfi 0



Af hverju hakkatöflur

- Hraðvirkari en tré, sérstaklega fyrir strengi
- Einfaldari minnisaðgangur
- Tré þurfa samanburð , “aab” < “abc”
  - skoðar allan strenginn
  - strengir geta geymt hakkagildið

Fræði

- Ef inntakið er vel dreift eða hakkafallið gott

  Fastur fjöldi aðgerða, O(1)

  fyrir put, get, delete

- Fjöldi hólfa jafn fjölda staka, að meðaltali 1 stak í hverjum lista.

“In theory, there is no difference between theory and practice. But in practice there is.”

Ef?

- **Ef** hakkafallið er gott er stórt ef
  - Hvað þýðir gott?
  - Hver ákveður hvað er nógu gott?

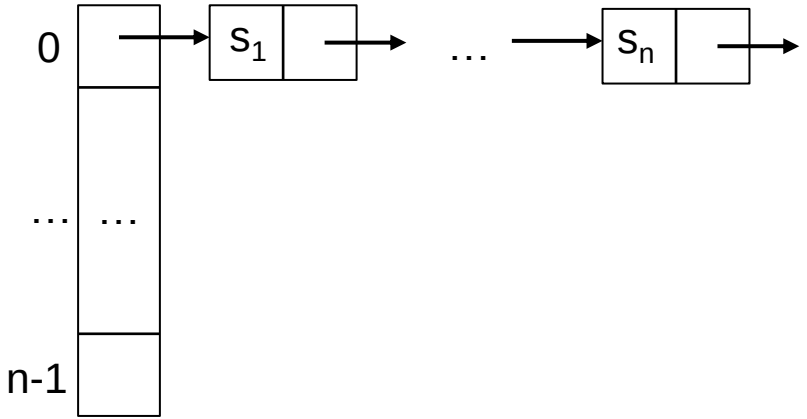
- Það eru til góðar fræðilegar skilgreiningar
  - enginn notar þær

Árás á reiknirit

- Einföld uppsetning, við vs. netþjónn
- Netþjónn leysir vandamál
  - t.d. tekur við HTTP fyrirspurn
- Við veljum inntak til að pína netþjóninn eins mikið og við getum
  - viljum hámarka CPU tíma sem netþjónn eyðir án þess að reyna mikið á okkur

Árás á hakkatöflu

- Veljum versta mögulega tilfelli.
- Finnum n lykla sem hafa allir sama hakkagildi
- Tími per lykil O(n)
- Heildartími $O(n^2)$
- Vinna fyrir okkur O(n)



Hvernig árás?

- Finnum netþjón sem keyrir kerfi, t.d. PHP
- Kíkjum á kóðann https://github.com/php/php-src
- Allar $_POST breytur eru geymdar í hakkatöflu
  - Finnum marga lykla sem allir hakkast í sama gildið
  - Finnum hvaða hakkafall er notað í kóðanum

Hakkaföll

- PHP4, ASP.NET nota DJBX33X
- Java (eldri útgáfur) nota DJBX31A,
- PHP5 svipað og Java

- Tiltölulega einfalt að ráðast á þessi föll

DJBX31A í Java

hashCode

```
public int hashCode()
```

Returns a hash code for this string. The hash code for a `String` object is computed as

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

```
hash = 0;
for (int i=0; i < s.length; i++) {
   hash = 31*hash + s[i];
}
```

DJBX31A

hashCode

```
public int hashCode()
```

Returns a hash code for this string. The hash code for a `String` object is computed as

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

‘A’=65,’B’=66,’a’=97

h(“BB”)=31*66+66 = 2112

h(“Aa”)=31*65+97 = 2112

$h(\text{“BBBB”}) = 31^2 * h(\text{“BB”}) + h(\text{“BB”})$

$h(\text{“BBAa”}) = 31^2 * h(\text{“BB”}) + h(\text{“Aa”})$

BBBB, BBAa, AaBB,AaAa hafa sama hakkagildi.

- 2n bæti
- 2^n mögulegar samsetningar með sama hakkagildi
- Einum of auðvelt.

DJBX33X

- Notar 33 í stað 31 fyrir margföldun
- Notar XOR í stað +

```
hash = 5381
for(int i=0; i < s.length; i++) {
   hash = (33*hash)^s[i];
}
```

Erfiðari árás

- Jafnvel þótt “BB” og “Aa” fái sama hakkagildi þýðir ekki að “BBBB” og “AaBB” fái sama hakkagildi.

- Notum “meet in the middle” árás
  - Skiptum inntakinu í tvennt, fyrri hluta, x og síðustu 3 stafi s.
  - lokaniðurstaðan ákvarðast af h(x) og s.
  - Ef við viljum fá niðurstöðuna 0 og höfum s (t.d. “AAA”) þá getum við reiknað okkur aftur á bak

Afturábak

h_i er hakkagildið eftir i skref

$$h_i = (33*h_{i-1}) \text{ ^ } s_i$$

h_i ^ s_i = $33*h_{i-1}$ // s_i^s_i = 0, styttist út

1041204193 *(h_i ^ s_i) = h_{i-1} //margföldunarandhverfa mod 2^{32}

Uppflettitafla

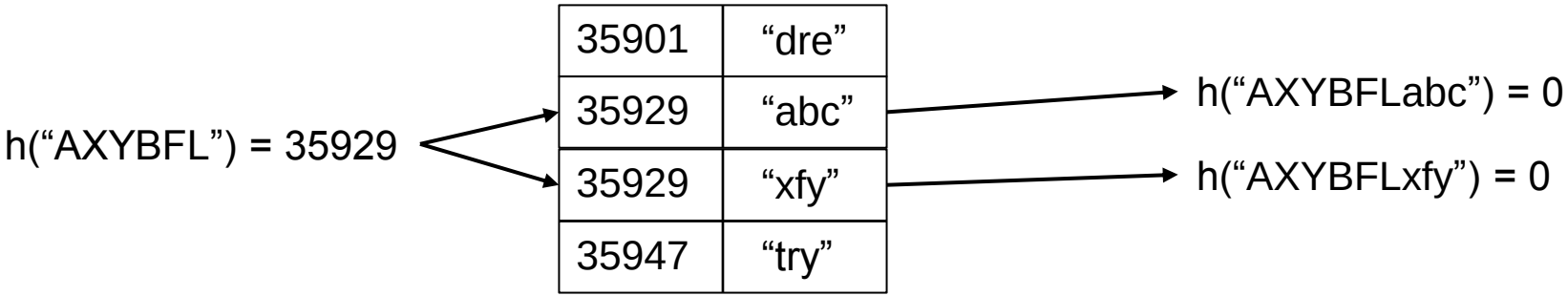
- Búum til uppflettitöflu fyrir alla 3 bæta strengi
  - setjum h_n=0 og reiknum aftur á bak 3 stafi
  - útkoman (35929,”abc”) segir okkur að ef h(x)=35929 þá er h(x+”abc”)=0
  - Veljum x af handahófi og vonum að útkoman sé í töflunni

Meet in the middle

- 3 bæta uppflettitafla, 2^{24} gildi af 2^{32} mögulegum útkomum, 32 milljón stök
- Líkurnar á að finna eitthvað í töflunni eru ca 1/256

- Getum fundið marga lykla sem hakkast í gildið 0

Meet in the middle



Fjöldi lykla

- Default stilling á php
  - 8 Mb takmark á POST
  - 10 bæti per lykil, getum sent fullt af lyklum
  - 8 Mb inntak → 288 mín CPU tími
  - 1 Mb inntak → 4 mín CPU tími
  - 500 K inntak → 1 mín CPU tími
- Þurfum ekki að senda mjög mikið til að halda netþjóninum uppteknum.

Lausnir?

- Hætta að nota hakkatöflur?
- Nota hakkatré, skipta út tengdum listum fyrir tré ef þeir verða langir (flókin útfærsla)
- Betri hakkaföll?
- Takmarka stærð á POST eða inntaki
- Takmarka keyrslutíma per fyrirspurn

Slembin hakkaföll

- Hakkatöflur eru út um allt, betra að skipta út hakkaföllunum
- Slembin hakkaföll nota handahófskenndar tölur

```
MurmurHash3_x86_32(const void * key,
                   int len,
                   uint32_t seed,
                   void *out);
```

- Veljum seed af handahófi í byrjun, erfitt að ráðast á.

Slembin hakkaföll

- Ekki töfralausn
  - MurmurHash3 hefur verið brotið
  - vel valið inntak lætur seed styttast út
- Besta lausnin í dag
  - SipHash
  - Örlítið hægara en MurmurHash3
  - Mun erfiðara að brjóta

Staðan í dag

- Java 7u40 skipti yfir í MurmurHash3, þarf auka stillingar við startup á java
- .NET notar Marvin32, enginn kóði til en virðist hægt að brjóta
- Ruby skipti yfir í SipHash
- PHP notar enn DJBX33X

Heimadæmi

Finnið alla staði í kóðanum ykkar þar sem hakkaföll eru notuð fyrir gögn frá notendum.

Ímyndið ykkur það versta sem gæti gerst.