Twee verschillende husseltabellen

Datastructuren, UvA. *16 april 2008*
0440949, Andreas van Cranenburgh

1 Inleiding

In dit verslag wordt bekeken of een ketting-husseltabel significant beter presteert dan een aftastende husseltabel, door twee zelfgemaakte implementaties te vergelijken.

2 Implementatie

2.1 Ketting-husseltabel

De ketting-husseltabel handelt husselbotsingen af door elementen met dezelfde husselcode in een gelinkde lijst te plaatsen. Telkens wanneer tijdens het toevoegen van een element reeds een ander element in de tabel aanwezig is op de gewenste plek, dan wordt dit element de nieuwe kop van deze lijst.

Het verwijderen van elementen is hetzelfde als het verwijderen van een element uit een gelinkde lijst: de voorganger van het element word gelinkt aan het opvolgende element, waarbij de referentie naar het te verwijderen element verdwijnt.

Als de tabel voor 75% gevuld is wordt de vergrotingsroutine aangeroepen. De groote wordt verdubbeld, plus één, in de hoop dat hier een priemgetal uit zal komen. Vervolgens worden alle elementen opnieuw in de tabel geplaatst.

2.2 Aftastende husseltabel

De aftastende husseltabel handelt botsingen af door het eerstvolgende vrije vakje op te zoeken, zolang hier nog ruimte voor is. Als er geen vrije ruimte is na de elementen met dezelfde husselcode en voor elementen met een andere, dan wordt de husseltabel vergroot ('gegroeid'), waarna er zeker een vrije plek zal zijn. Het aftasten vormt een routine die door alle operaties wordt aangeroepen.

Het verwijderen is ingewikkelder. Veel implementaties markeren verwijderde elementen slechts als 'beschikbaar.' Dit heeft echter het grote nadeel dat na verloop van tijd de tabel vol kan komen te staan met lege elementen.

Deze implementatie maakt gebruik van het algoritme van Knuth (1998), om clusters van elementen te comprimeren nadat er een element verwijderd is. Alle elementen na het zojuist verwijderde element worden opgeschoven, totdat er een vrij slot bereikt wordt.

Net als bij de ketting-husseltabel wordt ook deze tabel vergroot, maar dan al wanneer 50% van de capaciteit gebruikt is.

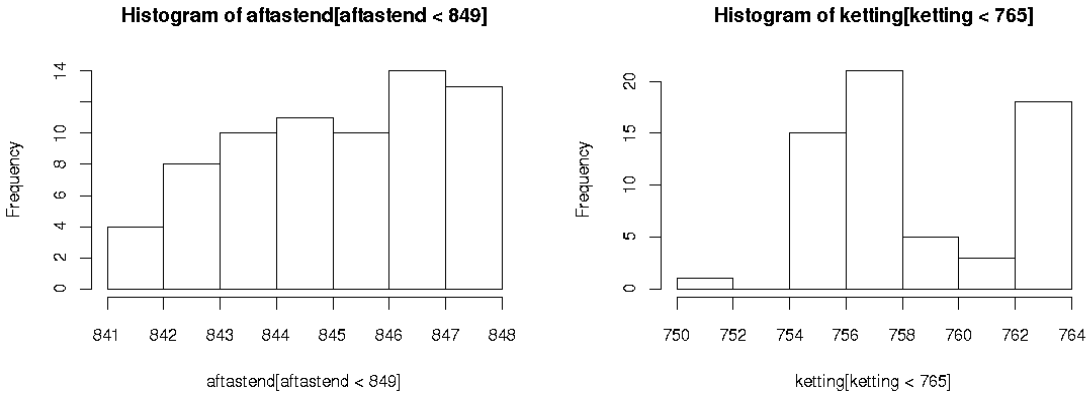
3 Prestatiemeting

Voor de presatiemeting zijn de twee geïmplementeerde husseltabellen vergeleken met een beschikbare referentie-implementatie: `java.lang.HashMap` [1] (een ketting-husseltabel). Vervolgens worden de drie husseltabellen als experiment gevuld met 100000 getallen, welk steeds dezelfde reeks semi-willekeurige getallen zijn. Dan wordt het vorige getal opgevraagd, en als laatste wordt met 30% kans het element daarvoor verwijderd. Dit experiment wordt 100 maal herhaald, telkens met lege tabellen, en met steeds dezelfde *seed* voor de willekeurige getallen.

De tijdsmeting gebeurd met `System.currentTimeMillis()`. Deze geeft helaas niet de werkelijke processortijd weer, maar heeft wel het voordeel dat deze binnen Java is aan te roepen. Om optimalisaties uit schakelen is `java` aangeroepen met de optie `-Xint`, hierdoor wordt alle gecode geinterpreteerd, en nooit gecompileerd.

De uitvoer van het programma is dusdanig dat deze meetgegevens direct als geconcateneerde vectors naar het statistisch programma 'R' kunnen worden gevoerd.

[1] Er is gekozen voor `HashMap` omdat deze, in tegenstelling tot `Hashtable`, geen synchronisatiebeloftes doet, en daardoor dus een eerlijkere vergelijking biedt



Figuur 1: De histogrammen van de resultaten. Uitschieters boven het derde kwartiel zijn weggelaten.

4 Resultaten

Als nulhypothese stellen we dat de ketting-husseltabel even snel zal zijn als de aftastende husseltabel, met een significantieniveau van $\alpha = 0.5$.

In het eerste experiment vergelijken we de prestaties van alle drie fundamentele operaties van de husseltabel, viz. `get, put` en `remove`. Een element wordt toegevoegd, opgevraagd en weer verwijderd.

De honderd resultaten van de drie husseltabellen zijn als volgt samen te vatten:

```
> summary(referentie)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  811.0   819.8   822.0   830.2   823.0  1302.0
> summary(ketting)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  751.0   757.0   763.0   771.5   765.0  1371.0
> summary(aftastend)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  841.0   845.0   847.0   868.2   849.0  1462.0
```

Vervolgens kan met een statistische toets bepaald worden of de gemiddelden significant afwijken:

```
> wilcox.test(ketting, aftastend)

        Wilcoxon rank sum test with continuity correction

data:  ketting and aftastend
W = 187, p-value < 2.2e-16
```

Aangezien de p-waarde ruimschoots kleiner is dan $\alpha = 0.05$, moet de nulhypothese verworpen worden, de twee implementaties hebben dus wel degelijk een verschillende gemiddelde snelheid.

5 Conclusie

De hier geïmplementeerde ketting-husseltabel is sneller gebleken dan zowel een aftastende husseltabel, alsook de referentie implementatie van de Javabibliotheek. De aftastende husseltabel is gemiddeld langzamer dan ketting-husseltabellen.

Dit is intuïtief te verklaren doordat het afhandelen van botsingen bij de aftastende husseltabel verdere bostingen kan veroorzaken, omdat plaatsen worden ingenomen die voor andere husselwaarden bestemd zijn. De ketting-husseltabel laat in vergelijking daarmee meer ruimte over, en zal dus minder last hebben van botsingen.

6 Referenties

De broncode: `https://unstable.nl/andreas/ai/ds/opdr4/`, opstarten als `java -Xint Assignment4`

Donald Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching, Second Edition. Addison-Wesley, 1998. ISBN 0-201-89685-0. Section 6.4: Hashing, pp.513558.