Inhoud

[3.1 Hashtabellen 2](#_Toc86697255)

[3.2 Verwerken van de overlappingen 3](#_Toc86697256)

[3.2.1 Gesloten hashing 3](#_Toc86697257)

[3.2.2 Open hashing 4](#_Toc86697258)

[3.3 Keuze van hashcode en hashfunctie 6](#_Toc86697259)

Hoofdstuk 3 Hashtabellen

# 3.1 Hashtabellen

Voorbeeld: we willen een verzameling implementeren waarbij de sleutelwaarde de natuurlijke getallen tussen 0 en 999 zijn. We maken gebruik van een eenvoudige array a van grootte 1000. We spreken af dat a[i] = i enkel en alleen als het getal i tot de verzameling hoort. We gebruiken null of -1 om aan te geven dat i niet tot de verzameling hoort.

We hebben 3 basis functionaliteiten, toevoegen, opzoeken en verwijderen. Het gebruik van een array op deze manier hier direct adressering.

Stel dat we objecten willen opslaan dat strings zijn van precies twee karakters lang. (aa tot zz). Dit zijn geen natuurlijke getallen. We kunnen wel een afbeelding construeren waarbij iedere twee letterige string afbeeld als een uniek natuurlijk getal tussen 0 en 675. A=0 b=2 z=25,…. Daarna interpreteren we deze twee cijfers in basis 26. De index voor het woord “je” is dan gelijk aan 9\*26^1 + 4\*26^0.

Dit kunnen we ook implementeren voor woorden van willekeurige lengte, het woord “hashtabel” ziet er als volgt uit:

We zien onmiddellijk dat de getallen die gegeneerd worden veel te groot zijn. We beperken ons tot een array van een bepaalde vooraf gekozen en doenbare grootte N. om de positie van een woord in deze tabel te bepalen, berekenen we eerst zijn getalwaarde. We gaan ervan uit dat de objecten een methode beschikken om hun hashcode te berekenen.

Om die hashcodes dan te transformeren naar het bereik tussen 0 en N-1 gebruikt men een hashfunctie. Een zeer gebruikte hashfunctie is werken met modulo N, men neemt de rest na de deling door N. Waarbij w een woord voorstelt, vinden we:

H(w) = w.hashcode() (mod N).

Deze functie zorgt ervoor dat we voor elk woord w een positie bepalen in de array.

We kunnen stellen dat hashing kan opgesplitst worden in twee luiken:

1. De keuze van een hashfunctie h die alle mogelijke items afbeeldt op een positie uit de hashtabel
2. Het selecteren van een methode om de waarden die overlappen of botsen te verwerken.

# 3.2 Verwerken van de overlappingen

Er zijn verschillende methodes voor het opvangen van botsingen (collisions). We bespreken er twee.

* Gesloten hashing
  + Er wordt in de tabel zelf gezocht naar een goede positie om een element, dat botst met een ander element op te slaan.
* Open hashing:
  + Er wordt buiten de tabel gezocht naar een plaats om de overlappende elementen op te slaan

## 3.2.1 Gesloten hashing

In dit geval worden alle woorden in de array zelf opgeslagen. Er doet zich een botsing voor wanneer de positie die correspondeert met een bepaald woord reeds ingenomen is. Het woord w kan dan niet op die positie opgeslagen worden zonder informatie te verliezen.

Als alternatief kan er gezocht worden naar de eerst volgende vrije positie in de tabel, waarbij de tabel als lineaire structuur wordt opgevat. Dit heet lineaire peiling.

We bouwen een hashtabel van lengte N=10. In de tabel moeten een aantal gehele getallen worden opgeslagen. Als bijhorende hashcode kiezen we voor het getal zelf. Als functie kiezen we:

H(w) = w (mod N)

We voegen een voor een de elementen toe. Indien een positie al ingenomen is, zoeken we naar de eerste vrije positie om het nieuwe element op te slaan. Afbeelding met tafel

Automatisch gegenereerde beschrijving

Men kan in een hashtabel met lineaire peiling ook eenvoudig zoeken. Men berekent de waarde van de hashfunctie voor het gekozen element.

* Wanneer die positie leeg is, komt het element niet voor.wanneer deze positie bezet is, vergelijkt men of het element op die positie gelijk is aan het te zoeken element.
* Als deze 2 gelijk zijn met elkaar stopt het programma.
* Wanneer deze elementen echter verschillen bekijkt het programma de volgende positie. Dit proces wordt herhaald tot dat het gezochte element gevonden wordt. Of tot dat men een lege positie tegen komt. Dit wilt zeggen dat het element niet in de hashtabel zit. In dat geval moet men ook stoppen

Het efficiënt opzoeken van een willekeurig element in een dergelijke tabel is afhankelijk van het aantal lege posities in de tabel. Als het aantal elementen in een tabel toeneemt, hebben deze bij lineaire peiling de neiging om zich te groepering. Dit heet clustering.

In een dergelijke tabel kan je ook niet zomaar een element verwijderen. Dit zou een lege plaats creëren die mogelijks deel uitmaakt van een sequentie van posities die nodig is om een element te bereiken.

Een mogelijke oplossing is om een vlag te plaatsen die bijhoudt dat het element verwijderd is. Als we dit toepassen bij het verwijderen van de waarde 10, zal de 0-de positie niet effectief leeg zijn. Hierdoor leiden we af dat 100 op een volgende positie is opgeslagen, wat in dit geval zo is.

## 3.2.2 Open hashing

In dit geval wordt er buiten de tabel gezocht naar een goede positie om elementen op te slaan. Een mogelijke oplossing is om alle elementen die op dezelfde positie moeten staan in de hashtabel op te slaan in een gelinkte lijst.

Wanneer die lijsten worden gebruikt om de overlappingen op te vangen bestaat een hashtabel uit een array van lengte N en op elke positie van de array wordt een referentie opgeslagen naar de gelinkte lijst waarin alle waarden worden opgeslagen die afgebeeld worden op die positie.

**Stappen bij implementatie van open hashing**

Veronderstel dat we een woordenboek willen opbouwen waarbij we met een groot aantal woorden w een definitie willen associëren

Afbeelding met tekst, klok

Automatisch gegenereerde beschrijving

.

We kunnen de basisfuncties voor een hashtabel als volgt beschrijven:

* hashFunctie(w):
  + bereken de hashcode van het woord w
  + bereken de bijhorende positie
  + geef de berekende positie terug
* voegToe(w,definitie):
  + bereken de juiste positie voor het woord w met de hashfunctie
  + maak een nieuwe knoop aan met drie velden:
    - veld voor het woord
    - veld voor de definitie
    - veld volgende
  + voeg de knoop toe aan de gelinkte lijst die start op de gevonden positie. De plaats van de knoop hangt af van de implementatie, zo kan de knoop altijd vooraan toegevoegd worden of kan er voor gekozen worden om de lijst gesorteerd te houden
* zoekOp(w):
  + bereken de juiste positie van het woord w met de hashfunctie
  + zoek in de gelinkte lijst waar de gevonden positie naar refereert, naar de knoop met het woord w
  + geef de definitie van die knoop terug of geef aan dat het gegeven woord niet tot de hashtabel behoort
* verwijder(w):
  + bereken de juiste positie van het woord w met de hashfunctie
  + zoek in de gelinkte lijst, waarde gevonden positie naar refereert, naar de knoop met woord w
  + verwijder de gevonden knoop uit de ketting
  + geef de verwijderde definitie terug

# 3.3 Keuze van hashcode en hashfunctie

Een goede combinatie van hashcode en hashfunctie moet aan twee voorwaarde voldoen:

* het moet snel te berekenen zijn
* zo groot mogelijke spreiding van elementen over verschillende posities

Dit lijkt voor de hand liggend maar het gebeurt in de praktijk heel vaak dat er tegen de voorwaarden gezondigd wordt.

Veronderstel dat we gegevens over studenten aan HOGENT willen opslaan in een hashtabel. Als sleutelwaarde gebruiken we de studentennummer. We gebruiken de nummers onmiddellijk ook als hashcode. Het nummer 1234-2019 correspondeert met een student die zich voor het eerst inschreef in 2019. Het nummer 1234-2020 voor een student van het jaar 2020. Op deze manier kunnen er 10000 studenten een nummer worden toegekend.

We willen de gegevens van de laatste 10 jaar bij te houden van ongeveer 50000 studenten. Wanneer de grote van de hashtabel gelijk is aan 10000, zou er bij ideale verdeling, elk lineair gelinkte lijst ongeveer 5 elementen bevatten.

Echter door de gekozen tabelgrootte bestaat de hashfunctie uit het werken met modulo 10000 en krijgen we:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Dit betekent dat alle student die gestart zijn in hetzelfde jaar op dezelfde positie terechtkomen in de tabel. Als al deze overlappingen bijgehouden zouden worden in een gelinkte lijst, zou deze enorm groot worden.

Dit kan vermeden worden door niet met modulo 10000 te werken maar met een priemgetal. Het liefst een priemgetal dichtbij de gewenste grootte. Hoe groter, hoe meer verschillende beeldwaarden de hashfunctie zal hebben.

Het priemgetal dichtbij 10000 is 10007, wanneer we modulo 10007 doen, zijn er 10007 verschillende antwoorden mogelijk.

Afbeelding met tekst

Automatisch gegenereerde beschrijving