Hashtabellen

Classic Computer Science Algorithms

Stijn Lievens **Pieter-Jan Maenhaut** **Koen Mertens** **Lieven Smits**

AJ 2021–2022

HO GENT

Hashtabellen

HO
GENT

Inhoud

HO GENT

Inleiding

We hebben al kennis gemaakt met een array en gelinkte lijst als gegevensstructuur.

- Element opzoeken in een gelinkte lijst: lineaire tijd.
- Andere basisfuncties in een constante tijd: op voorwaarde dat men reeds de positie heeft bepaald en de referentie hiernaar kent
- Stel: we wensen een digitaal woordenboek te implementeren. Dan moet het opzoeken efficiënt gebeuren.
  - Voor dergelijke toepassingen bieden *hashtabellen* een oplossing.
  - Voordeel: typisch hebben we in een hashtabel een constante tijd voor: het opzoeken, toevoegen en het verwijderen van elementen
  - Nadeel: een hashtabel is echter *niet* in staat om de elementen gesorteerd terug te geven, wat bv. bij een binaire zoekboom (zie later) wel het geval is.

HO GENT

Inleiding

Stel: een verzameling waarbij de sleutelwaarden de natuurlijke getallen tussen 0 en 999 zijn. In dit geval kan men eenvoudig weg een array a van grootte 1000 alloceren, en we spreken af dat

$$a[i] = i$$

enkel en alleen als het getal i tot de verzameling hoort. We gebruiken een bijzonder waarde (bv. “null” of –1) om aan te geven dat i niet tot de verzameling behoort $\implies$ de drie basisbewerkingen, nl. toevoegen, opzoeken en verwijderen kunnen in constante tijd uitgevoerd worden. Het gebruik van een array op deze manier noemen we *directe adressering*.

HO GENT

Inleiding

Voorbeeld: de verzameling natuurlijke getallen tussen 0 en 9.
Dan komt de array

Index	0	1	2	3	4	5	6	7	8	9
Sleutel	-1	1	-1	3	-1	-1	-1	7	8	9

overeen met de verzameling $\{1, 3, 7, 8, 9\}$

HO GENT

Specificatie

In veel gevallen zijn de objecten die we willen opslaan echter geen natuurlijke getallen.

- Stel strings van 2 karakters lang bvb. “aa” t.e.m. “zz”.
- Construeer afbeelding die tweeletterige string afbeeldt op uniek natuurlijk getal $0 - 675$. vb $f(a) = 0$, $f(z) = 25$
- Daarna interpreteren we deze twee cijfers in basis 26. De index voor het woord “je” is dan gelijk aan

$$9 \times 26^1 + 4 \times 26^0 = 238.$$

Deelverzamelingen van alle tweeletterige strings kunnen m.a.w. eenvoudig geïmplementeerd worden m.b.v. een array van lengte $26 \times 26 = 676$. Opnieuw moet afgesproken worden welke speciale waarde wordt gebruikt om aan te geven dat een element niet tot de verzameling behoort.

HO GENT

Specificatie

Zelfde tactiek voor het woord “hashtabel” :

$$7 \times 26^8 + 0 \times 26^7 + 18 \times 26^6 + 7 \times 26^5 + 19 \times 26^4 + 0 \times 26^3 + 1 \times 26^2 + 4 \times 26^1 + 11 \times 26^0 = 1467\,441\,788\,967 \approx 1.46 \times 10^{12}.$$

- Veel te grote getallen!
- Meeste elementen toch “null” want meeste combinaties van letters zijn geen geldige woorden
- Oplossing: Kies grootte N en bereken HASHCODE
- Transformeer hashcodes naar bereik tussen 0 en $N - 1$ door gebruik HASHFUNCTIE. Vb. modulo N, i.e. men neemt de rest na deling door N.

HO GENT

Specificatie

Formeel uitgedrukt, waarbij w een woord voorstelt, vinden we

$$h(w) = w.\text{hashCode}(\,)\ \pmod{N}.$$

Deze hashfunctie zorgt er inderdaad voor dat we voor elk woord (of object) w een positie bepalen in de array.

HO GENT

Botsingen

Aangezien er meer hashcodes zijn dan posities in de tabel is het niet uitgesloten dat een aantal woorden op dezelfde positie moet opgeslagen worden. De opbouw van de *hashtabel* zal afhangen van de manier waarop wordt omgegaan met deze *botsingen*.
Algemeen kunnen we 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 (*collisions*) te verwerken.

HO GENT

Types hashing

Er zijn verschillende oplossingsmethodes voor het opvangen van de *collisions*. We bespreken twee methodes.

1. *gesloten hashing*: zoek in de tabel zelf naar een goede positie om een element, dat botst met een ander element uit de tabel, op te slaan.
2. *open hashing*: zoek buiten de tabel naar een plaats om de overlappende elementen op te slaan.

HO GENT

Gesloten hashing

- In het geval van GESLOTEN HASHING 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, of anders gezegd $h(w)$ is reeds bezet.
- Het woord w kan dan niet op die positie opgeslagen worden zonder informatie te verliezen
- Alternatieve positie om het woord op te slaan kan gekozen worden voor de eerstvolgende vrije positie in de tabel, waarbij de tabel als een circulaire structuur wordt opgevat

  $\Longrightarrow$ LINEAIRE PEILING.

HO GENT

Lineaire peiling: voorbeeld

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

$$h(w) = w \pmod{N}.$$

We voegen één voor één de elementen 10, 15, 29, 100, 115 en 129 toe. Indien een positie reeds ingenomen is, zoeken we naar de eerste vrije positie om het nieuwe element op te slaan. Hierbij beschouwen we de tabel als een circulaire structuur: de eerste positie volgt op de laatste positie.

HO GENT

Lineaire peiling: voorbeeld

t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	opmerkingen:
10										waarde 10 toegevoegd op positie 0
10					15					waarde 15 toegevoegd op positie 5
10					15				29	waarde 29 toegevoegd op de positie 9
10	100				15				29	100 schuift door naar eerste vrije positie
10	100				15	115			29	115 schuift door naar eerste vrije positie
10	100	129			15	115			29	129 schuift door naar eerste vrije positie

HO GENT

Lineaire peiling: zoeken

Ook zoeken gaat eenvoudig:

- Bereken hashfunctie voor element
- Indien leeg: element niet aanwezig
- Indien bezet: ofwel element gevonden
- Indien bezet: element zoeken op volgende posities
  - tot gevonden
  - of lege positie
  - of terug bij beginpositie

HO GENT

Lineaire peiling: zoeken voorbeeld

Voorbeeld: zoek de waarden hieronder in de voorbeeldtabel

- 10
- 115
- 149

HO GENT

Lineaire peiling

Efficiëntie van zoeken is afhankelijk van het aantal lege posities in de tabel.

- Bij toename aantal elementen $\Longrightarrow$ groeperen : *primaire clustering* genoemd
- Wat als we elementen zomaar verwijderen? $\Longrightarrow$ kan gat creëren in sequentie vb. verwijder waarde 10
- Daarom plaatsen we een *vlag* om te duiden dat het element verwijderd is.

HO GENT

Open hashing

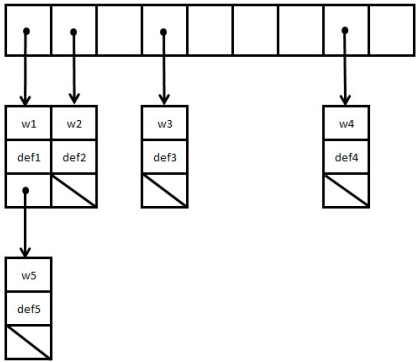
In het geval van OPEN HASHING wordt er buiten de tabel gezocht naar een goede positie om elementen op te slaan.

- mogelijke oplossing : sla alle elementen op dezelfde positie in de hashtabel op in een gelinkte lijst.
- In praktijk bestaat zo'n hashtabel uit een array van lengte N en op elke positie van de array wordt een referentie opgeslagen naar de gelinkte lijst met alle waarden

HO GENT

Open hashing

Voorbeeld van de *map* interface voor de definitie van woorden. Om te kunnen achterhalen welke knoop van de gelinkte lijst bij welk woord hoort, zal in de knopen van de gelinkte lijst naast de definitie ook het woord zelf moeten bijgehouden worden.



HO GENT

Implementatie van open hashing

Veronderstel dat we een woordenboek willen opbouwen waarbij we met een groot aantal woorden w een definitie willen associëren. In het geval van open hashing zijn basisfuncties voor een hashtabel als volgt:

- *hashFunctie*(w) voert achtereenvolgens uit:
  - bereken de hashcode van het woord w;
  - bereken de bijhorende positie;
  - geef de berekende positie terug.
- *voegToe*(w, *definitie*) voert achtereenvolgens uit:
  - bereken de juiste positie voor het woord w met de hashfunctie;
  - maak een nieuwe knoop aan met drie velden:
    - een veld voor het woord,
    - een veld voor de definitie,
    - een veld volgende;
  - voeg de knoop toe aan de gelinkte lijst die start op de gevonden positie.

HO GENT

Implementatie van open hashing

- *zoekOp*(*w*) werkt als volgt:
  - 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*) werkt als volgt:
  - 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*;
  - verwijder de gevonden knoop uit de ketting;
  - geef de verwijderde definitie terug.

HO GENT

Keuze van hashcode en hashfunctie

Om een efficiënte hashtabel op te bouwen is het niet alleen belangrijk dat de overlappingen goed worden opgevangen maar is het vooral belangrijk een efficiënte hashcode en hashfunctie te kiezen.
Een goede combinatie van hashcode en hashfunctie moet aan twee voorwaarden voldoen:

1. snel te berekenen
2. zo groot mogelijke spreiding van de elementen over de verschillende posities

Dit lijkt voor de hand liggend maar het gebeurt in de praktijk heel vaak dat er tegen de tweede voorwaarde wordt gezondigd. Dit leidt dan tot een teleurstellende performantie.

HO GENT

Problematische hashcode

Veronderstel dat we hashcode voor een string definiëren als de som van de waarden van de verschillende letters. Bv. het woord “hashtabel” zou in dit geval de volgende waarde krijgen:

$$7 + 0 + 18 + 7 + 19 + 0 + 1 + 4 + 11 = 67.$$

Deze hashcode heeft twee problemen:

1. waarden zijn (te) klein vb. woorden met hoogstens 10 karakters lang geeft een maximumwaarde van $25 \times 10 = 250$. Zelfs als we een array alloceren met $10\,000$ posities dan zullen enkel de eerste 250 posities gebruikt kunnen worden.
2. Alle permutaties van dezelfde letters worden op dezelfde hashcode afgebeeld, zo zal het Engelstalige woord “hashtable” ook hashcode 67 hebben. Dit zorgt voor onnodig veel botsingen.

HO GENT

Problematische tabelgrootte

Ook de grootte van de hashtabel, i.e. de waarde van N kan een grote invloed hebben op de performantie van de hashtabel.

- Stel : alle HOGENT studenten in een hashtabel. Als sleutelwaarde gebruiken we een uniek studentennummer voor elke student. Veronderstel dat dit studentennummer als volgt is opgebouwd. Eerst komen er 4 cijfers (een volgnummer) en daarna komt het jaar waarin deze student zich voor het eerst inschreef. Volgens deze werkwijze kan er jaarlijks aan maximaal $10^4 = 10\,000$ studenten een nummer toegekend worden. We veronderstellen dat dit voldoende is aangezien er zich gemiddeld 5000 nieuwe studenten inschrijven
- We wensen gegevens bij te houden voor de laatste 10 jaar, dus van ongeveer $50\,000$ studenten, en we gebruiken hiervoor een hashtabel met open hashing. Wanneer de grootte van deze hashtabel gelijk aan $10\,000$ wordt gekozen dan zou, bij een ideale verdeling, elke lineair gelinkte lijst ongeveer 5 elementen bevatten.

HO GENT

Problematische tabelgrootte

Echter, door de gekozen tabelgrootte krijgen we het volgende:

$$h(1234\text{–}2019) = 2019$$
$$h(5489\text{–}2019) = 2019$$
$$h(0358\text{–}2019) = 2019$$
$$h(9786\text{–}2020) = 2020.$$

Dit betekent dat alle studenten die gestart zijn in hetzelfde academiejaar op dezelfde positie terechtkomen in de tabel. Al deze overlappingen worden bijgehouden in een gelinkte lijst, maar deze gelinkte lijst zal wel bijzonder lang worden $\implies$ 5000 studenten

- Hoe kunnen we dit oplossen?
- Waarom priemgetallen gebruiken?
- $10\,007$ is het priemgetal het dichtste gelegen bij $10\,000$

HO GENT

Problematische tabelgrootte

De hashfunctie wordt nu gegeven door

$$h : student \mapsto student.\text{hashcode} \pmod{10\,007}.$$

Deze hashfunctie zorgt ervoor dat de verschillende studenten beter verdeeld worden over alle posities van de tabel. Bijvoorbeeld:

$$h(1234\text{–}2019) = 3388$$
$$h(5489\text{–}2019) = 3624$$
$$h(0358\text{–}2019) = 9520$$
$$h(9786\text{–}2020) = 3567.$$

De studentengegevens worden op de corresponderende positie ingevuld in de tabel.

HO GENT

Oefening 1

Voeg alle (verschillende) letters uit het woord ‘DEMOCRATISCH’ toe aan een hashtabel. De grootte van de hashtabel wordt bepaald door $N = 5$, dus er zijn vijf plaatsen (buckets) in de hashtabel. In de hashtabel wordt er geen waarde opgeslagen corresponderend met de letters.
De te gebruiken hashcode is $11 \times k$ met k de positie van de letter in het alfabet.
Hoe evolueert de hashtabel?

HO GENT

Oefening 2

Beschouw het volgende probleem: gegeven een array a bestaande uit gehele getallen en een geheel getal t. Retourneer twee verschillende indices i en j zodanig dat $a_i + a_j = t$.

1. De meest voor de hand liggende oplossing is om alle koppels (i, j) met $i < j$ te overlopen en te verifiëren of $a_i + a_j = t$. Schrijf deze oplossing uit in pseudocode en analyseer de tijdscomplexiteit.
2. Op welke manier kan je gebruikmaken van een hashtabel om deze tijdscomplexiteit te verbeteren? Je mag er hierbij van uitgaan dat toevoegen aan en opzoeken in de hashtabel kan gebeuren in constante tijd. Vergelijk de hoeveelheid extra geheugen die deze oplossing nodig heeft met de oplossing van vorig puntje.

HO GENT