Föreläsning 3

ADT Map/Dictionary, hashtabeller

TDDC91,TDDE22,725G97: DALG

Utskriftsversion av föreläsning i *Datastrukturer och algoritmer* 7 september 2018

Magnus Nielsen, IDA, Linköpings universitet

3.1

1 ADT Map/Dictionary

1.1 Definitioner

ADT Map

- Domän: mängder av poster/par (*nyckel*, *värde*) Mängderna är partiella funktioner som avbildar nycklar på värden!
- Typiska operationer:
  - size() - storlek (antalet par i mängden)
  - isEmpty() - är mängden tom
  - get(k) - hämta informationen associerad med nyckeln$k$ eller **null** om någon sådan nyckel inte finns
  - put(k, v) - lägg till (k, v) till mängden och returnera **null** om k är ny; ersätt annars värdet med v och returnera det gamla värdet
  - remove(k) - ta bort post (k, v) och returnera v; returnera **null** om mängden inte har någon sådan post

3.2

ADT Map

- Exempel:
  - Kursdatabas: (kod, namn)
  - Associativt minne: (adress, värde)
  - Gles matris: ((rad, kolumn), värde)
  - Lunchmeny: (dag, rätt)
- Statisk Map: inga uppdateringar tillåtna
- Dynamisk Map: uppdateringar *är* tillåtna

3.3

ADT Dictionary

- Domän: mängder av par (*nyckel*, *värde*) Mängderna är relationer mellan nycklar och värden!
- Typiska operationer:
  - size() - antalet par i mängden
  - isEmpty() - kolla om mängden är tom
  - find(k) - returnera någon post med nyckel k eller **null** om inget sådant par finns
  - findAll(k) - returnera en itererbar samling av alla poster med nyckel k
  - insert(k, v) - lägg till (k, v) och returnera den nya posten
  - remove(k, v) - ta bort och returnera paret (k, v); returnera **null** om det inte finns något sådant par
  - entries() - returnera itererbar samling av alla poster

3.4

ADT Dictionary

- Exempel:
  - Svensk-engelskt lexikon . . . , (jakt, yacht), (jakt, hunting), . . .
  - Telefonkatalog (flera nummer tillåtna)
  - Relation mellan liuid och avklarade kurser
  - Lunchmeny (med flera val): (dag, rätt)
- Statisk Dictionary: inga uppdateringar tillåtna
- Dynamisk Dictionary: uppdateringar *är* tillåtna

3.5

1.2 Implementation

Implementation: Map, Dictionary

- Tabell/array: sekvens av minnesområden av lika storlek
  - Oordnad: ingen särskild ordning mellan $T[i]$ och $T[i+1]$
  - Ordnad: . . . men här gäller $T[i] < T[i+1]$
- Länkad lista
  - Oordnad
  - Ordnad
- Hashning
- (Binära) sökträd (föreläsning 4)

3.6

Tabellrepresentation av Dictionary

Oordnad tabell:
find genom *linjärsökning*

- misslyckad uppslagning: n jämförelser $\Rightarrow O(n)$ tid
- lyckad uppslagning, värsta fallet: n jämförelser $\Rightarrow O(n)$ tid
- lyckad uppslagning, medelfallet med likformig fördelning av förfrågningar: $\frac{1}{n}(1+2+\ldots+n) = \frac{n+1}{2}$ jämförelser $\Rightarrow O(n)$ tid

3.7

Tabellrepresentation av Dictionary

Ordnad tabell (nycklarna är linjärt ordnade):
find genom *binärsökning*

- uppslagning: $O(\log n)$ tid
- . . . uppdateringar är dyra!!

3.8

Kan vi hitta på något bättre?

Ja, med hjälp av *hashtabeller*

- Idé: givet en tabell $T[0,\ldots,max]$ att lagra element i . . . *. . . hitta ett lämpligt tabellindex* för varje element
- Hitta en funktion h sådan att $h(key) \in [0,\ldots,max]$ och (idealt) sådan att $k_1 \neq k_2 \Rightarrow h(k_1) \neq h(k_2)$
- Lagra varje nyckel-värdepar (k,v) i $T[h(k)]$

Kursbokens terminologi: $h(k) = g(hc(k))$ hash code $hc : Nycklar \rightarrow Heltal$; kompressionsfunktion $g : Heltal \rightarrow [0,\ldots,max]$

3.9

2 Hashtabeller

Hashtabell

- I praktiken ger inte hashfunktioner unika värden (de är inte *injektiva*)
- Vi behöver kollisionshantering

. . . och

- Vi behöver hitta en bra hashfunktion

3.10

2.1 Kollisionshantering

Kollisionshantering

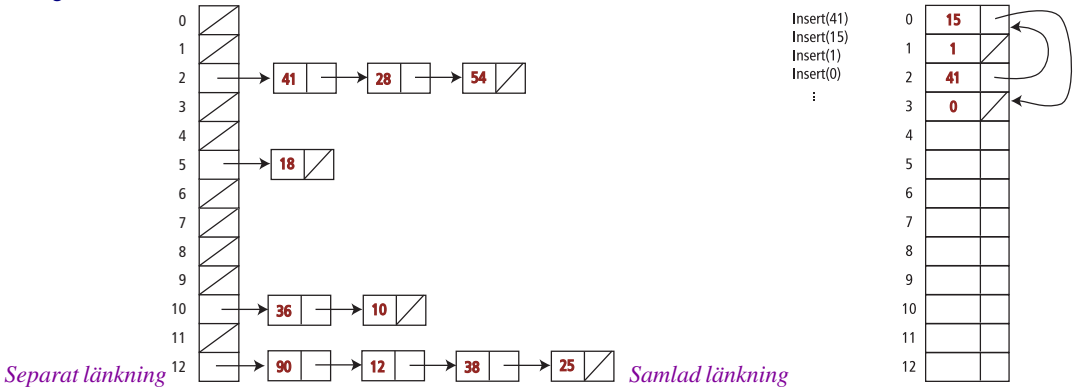
Två principer för att hantera kollisioner:

- *Länkning*: håll krockande data i länkade listor
  - *Separat länkning*: ha de länkade listorna utanför tabellen
  - *Samlad länkning*: lagra alla data *i* tabellen
- *Öppen adressering*: lagra alla data *i* tabellen *och* låt någon algoritm bestämma vilket index som ska användas vid en kollision

[Eng: Separate Chaining, Coalesced Chaining, Open Addressing]

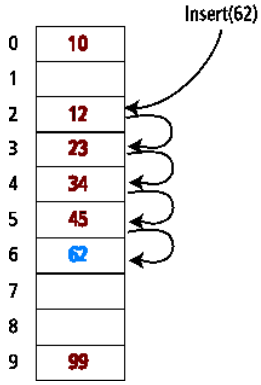
3.11

Länkning



3.12

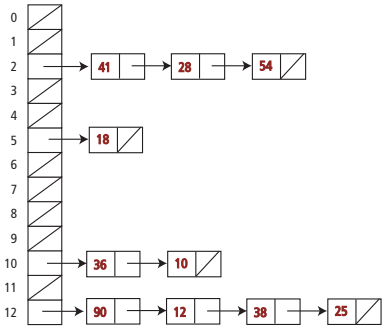
Öppen adressering



3.13

Exempel: hashning med separat länkning

- Hashtabell med storlek 13
- Hashfunktion h med $h(k) = k \bmod 13$
- Lagra 10 heltalsnycklar: 54, 10, 18, 25, 28, 41, 38, 36, 12, 90



3.14

Separat länkning: find

Givet: nyckel k, hashtabell T, hashfunktion h

- beräkna $h(k)$
- leta efter k i listan $T[h(k)]$ pekar ut

Notation: sondering= en access i den länkade listan

- 1 sondering för att komma åt listhuvudet (om icke-tomt)
- 1+1 sondering för att komma åt innehållet i första listelementet
- 1+2 sondering för att komma åt innehållet i andra listelementet
- ...

En sondering (att följa en pekare) tar konstant tid. Hur många avpekningar P behövs för att hämta en post i hashtabellen?

3.15

Separat länkning: misslyckad uppslagning

- n dataelement
- m platser i tabellen

Värsta fallet:

- alla dataelement har samma hashvärde: $P = 1 + n$

Medelfallet:

- hashvärden likformigt fördelade över m:
- medellängd α av lista: $\alpha = n/m$
- $P = 1 + \alpha$

3.16

Separat länkning: lyckad uppslagning

Medelfallet:

- access av $T[h(k)]$ (början av en lista L): 1
- traversera $L \Rightarrow k$ hittas efter: $|L|/2$
- förväntat $|L|$ svarar mot α, alltså: förväntat $P = \alpha/2 + 1$

3.17

Samlad länkning: behåll elementen i tabellen

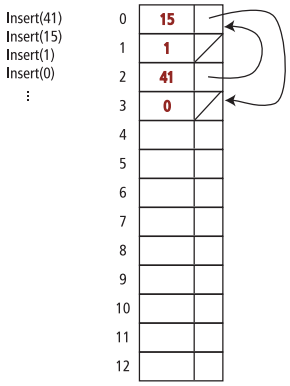
- Placera dataelementen i tabellen
- Utöka dem med pekare
- Lös kollisioner genom att använda första lediga plats

Kedjor kan innehålla nycklar med olika hashvärden. . . . . men alla nycklar med samma hashvärden dyker upp i samma kedja

\+ Bättre minnesanvändning - Tabellen kan bli full - Längre kollisionskedjor

3.18

Samlad länkning



3.19

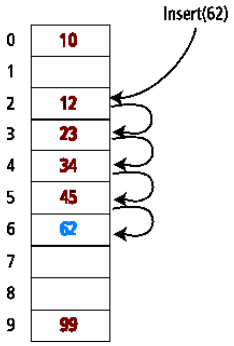
Öppen adressering

- Lagra alla element inuti tabellen
- Använd en fix algoritm för att hitta en ledig plats

Sekvensiell/linjär sondering

- önskvärt hashindex $j = h(k)$
- om konflikt uppstår gå till *nästa* lediga position
- om tabellen tar slut, gå till början av tabellen...

- Positioner i närheten av varandra fylls snabbt upp (*primärklustring*)
- Hur gör man remove(k)?



3.20

Öppen adressering — remove()

Elementet som ska tas bort kan vara del i en kollisionskedja – kan vi avgöra det?

Om det är del av en kedja kan vi inte bara ta bort elementet!

- Eftersom alla nycklar lagras, hasha om alla data som är kvar?
- Titta bland elementen efter, hasha om eller dra ihop när lämpligt, stanna vid första lediga position...?
- Ignorera – sätt in en markör "borttagen" (deleted) om nästa plats är icke-tom...

3.21

Dubbel hashning – eller vad göra vid kollision?

- Andra hashfunktion h_2 beräknar *inkrement* i fall av konflikter
- Inkrement utanför tabellen tas modulo $m = tableSize$

Linjär sondering är dubbel hashning med $h_2(k) = 1$ Krav på h_2:

- $h_2(k) \neq 0$ för alla k
- $h_2(k)$ har inga gemensamma delare med m för något $k \Rightarrow$ *alla* tabellpositioner kan nås

Ett vanligt val $h_2(k) = q - (k \bmod q)$ för $q < m$, q primtal (dvs, välj ett primtal mindre än tabellstorleken!)

3.22

2.2 Att välja hashfunktion

Vad är en bra hashfunktion?

Antag att k är ett naturligt tal.

Hashning bör ge en likformig fördelning av hashvärden, *men* detta beror på *distributionen av nycklar* i datat som ska hashas.

Exempel: Hashning av efternamn i en (svensk) grupp studenter

- hashfunktion: ASCII-värdet av sista bokstaven *dåligt val:* majoriteten av namn slutar med 'n'.

3.23

Hashning genom heltalsdivision

Låt m vara tabellstorleken

$$h(k) = k \bmod m$$

Undvik

- $m = 2^d$: hashning ger sista d bitarna i k
- $m = 10^d$: hashning ger d sista siffrorna

Man brukar föreslå primtal för m.

Se `http://burtleburtle.net/bob/hash/doobs.html` för vidare läsning.

3.24