TDDI16 – Föreläsning 5

Hashning och hashtabeller

Filip Strömbäck

LINKÖPING UNIVERSITY

Planering

Vecka	Fö	Lab
35	Komplexitet, Linjära strukturer	----
36	Träd, AVL-träd	1---
37	Hashning	1---
38	Grafer och kortaste vägen	12--
39	Fler grafalgoritmer	-23-
40	Sortering	--3-
41	Mer sortering, beräkningsbarhet	--34
42	Tentaförberedelse	---4

LINKÖPING UNIVERSITY

Från föreläsning 3

	`vector`	`set`, `map`	Önsketänkande
Insättning	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$	$\mathcal{O}(1)$
Uppslagning	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$	$\mathcal{O}(1)$
Stavningskontroll	$\mathcal{O}(n^2)$	$\mathcal{O}(n\log(n))$	$\mathcal{O}(n)$

Kan vi implementera vårt önskade fall?

Förenkling

Antag att nycklarna är heltal mellan 0 och 9

Förenkling

Antag att nycklarna är heltal mellan 0 och 9

Vi använder nyckeln som index i ett array!

0	1	2	3	4	5	6	7	8	9

Sätt in: 2, c

Förenkling

Antag att nycklarna är heltal mellan 0 och 9

Vi använder nyckeln som index i ett array!

0	1	2	3	4	5	6	7	8	9
		c							

Sätt in: 2, c
Sätt in: 8, d

Förenkling

Antag att nycklarna är heltal mellan 0 och 9

Vi använder nyckeln som index i ett array!

0	1	2	3	4	5	6	7	8	9
		c						d	

Sätt in: 2, c
Sätt in: 8, d

Om vi inte vet något intervall?

Antag att nycklarna är heltal

Vi använder nyckeln **modulo längden** som index!

0	1	2	3	4	5	6	7	8	9

Sätt in: 8, d

Om vi inte vet något intervall?

Antag att nycklarna är heltal

Vi använder nyckeln **modulo längden** som index!

0	1	2	3	4	5	6	7	8	9
								d	

Sätt in: 8, d
Sätt in: 54, e

Om vi inte vet något intervall?

Antag att nycklarna är heltal

Vi använder nyckeln **modulo längden** som index!

0	1	2	3	4	5	6	7	8	9
				e				d	

Sätt in: 8, d

Sätt in: 54, e

Hämta: 54 $\Rightarrow$ e

Om vi inte vet något intervall?

Antag att nycklarna är heltal

Vi använder nyckeln **modulo längden** som index!

0	1	2	3	4	5	6	7	8	9
				e				d	

Sätt in: 8, d

Sätt in: 54, e

Hämta: 54 $\Rightarrow$ e

Hämta: 28 $\Rightarrow$ d **?**

Om vi inte vet något intervall?

Vi använder nyckeln **modulo längden** som index! Lagra även nyckeln.

0	1	2	3	4	5	6	7	8	9

Sätt in: 8, d

Om vi inte vet något intervall?

Vi använder nyckeln **modulo längden** som index! Lagra även nyckeln.

0	1	2	3	4	5	6	7	8	9
								8	
								d	

Sätt in: 8, d

Sätt in: 54, e

Om vi inte vet något intervall?

Vi använder nyckeln **modulo längden** som index! Lagra även nyckeln.

0	1	2	3	4	5	6	7	8	9
				54				8	
				e				d	

Sätt in: 8, d

Sätt in: 54, e

Hämta: 28 $\Rightarrow$ tomt

Om vi inte vet något intervall?

Vi använder nyckeln **modulo längden** som index! Lagra även nyckeln.

0	1	2	3	4	5	6	7	8	9
				54				8	
				e				d	
								?	

Sätt in: 8, d

Sätt in: 54, e

Hämta: 28 $\Rightarrow$ tomt

Sätt in: 38: f

LINKÖPING UNIVERSITY

Separat länkning

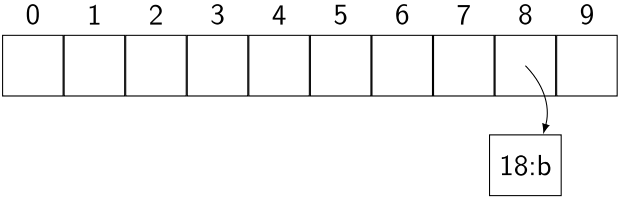
Idé: Använd länkade listor för att lagra fler element på samma index.

0	1	2	3	4	5	6	7	8	9

Sätt in 18:b

Separat länkning

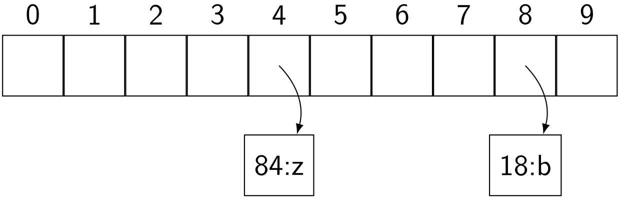
Idé: Använd länkade listor för att lagra fler element på samma index.



Sätt in 18:b, 84:z

Separat länkning

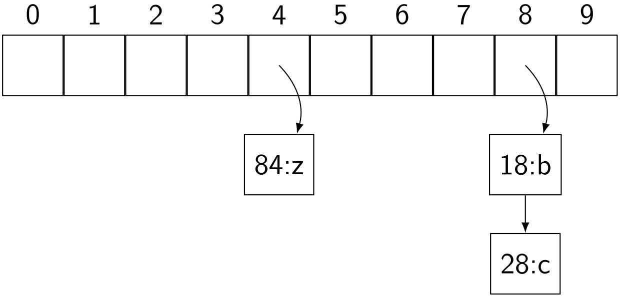
Idé: Använd länkade listor för att lagra fler element på samma index.



Sätt in 18:b, 84:z, 28:c

Separat länkning

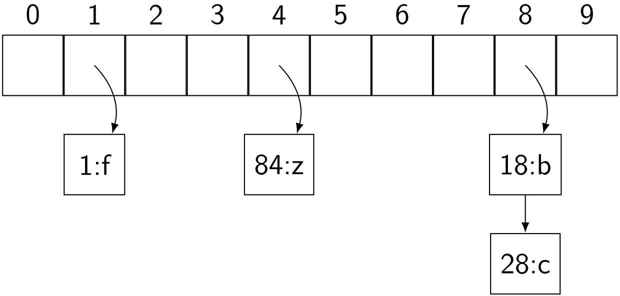
Idé: Använd länkade listor för att lagra fler element på samma index.



Sätt in 18:b, 84:z, 28:c, 1:f

Separat länkning

Idé: Använd länkade listor för att lagra fler element på samma index.



Sätt in 18:b, 84:z, 28:c, 1:f

Hur effektivt är detta?

- Uppslagning
  1. Hitta rätt plats i arrayen
  2. Hitta elementet i listan
- Insättning
  1. Hitta rätt plats i arrayen
  2. Se om elementet redan finns
  3. Sätt in i listan

Hur effektivt är detta?

- Uppslagning
    1. Hitta rätt plats i arrayen $\mathcal{O}(1)$
    2. Hitta elementet i listan $\mathcal{O}(c)$
- Insättning
    1. Hitta rätt plats i arrayen $\mathcal{O}(1)$
    2. Se om elementet redan finns $\mathcal{O}(c)$
    3. Sätt in i listan $\mathcal{O}(1)$

$c =$ antal element i en kedja (antal kollisioner)

Hur effektivt är detta?

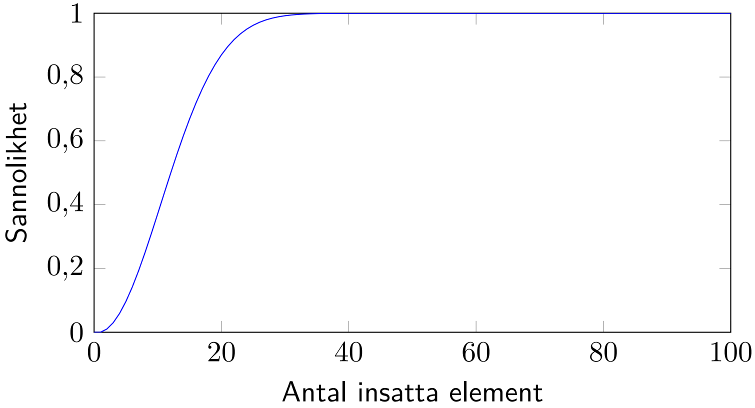
- Uppslagning
  1. Hitta rätt plats i arrayen $\mathcal{O}(1)$
  2. Hitta elementet i listan $\mathcal{O}(c)$
- Insättning
  1. Hitta rätt plats i arrayen $\mathcal{O}(1)$
  2. Se om elementet redan finns $\mathcal{O}(c)$
  3. Sätt in i listan $\mathcal{O}(1)$

$c =$ antal element i en kedja (antal kollisioner)

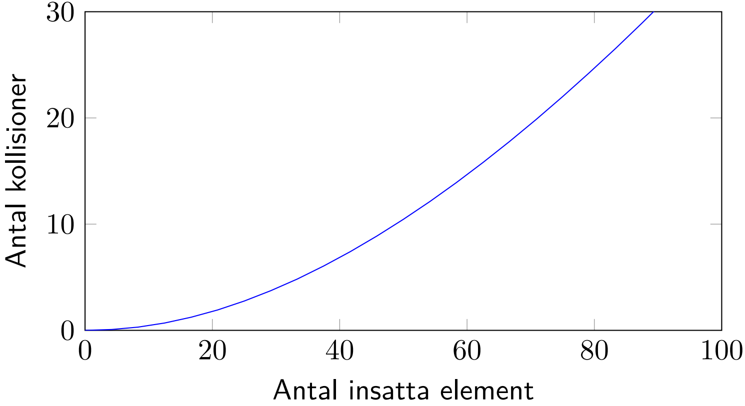
Om elementen är utspridda: $c \approx \frac{n}{m}$ $m =$ arraystorlek

Alltså: $c \approx 1$

Hur ofta förekommer kollisioner? (tabellstorlek 100)



Hur många kollisioner blir det? (tabellstorlek 100)



Linjär sondering

	0	1	2	3	4	5	6	7	8	9
key:										
value:										

Sätt in: 7:a

Linjär sondering

	0	1	2	3	4	5	6	7	8	9
key:								7		
value:								a		

Sätt in: 7:a, 22:b

Linjär sondering

	0	1	2	3	4	5	6	7	8	9
key:			22					7		
value:			b					a		

Sätt in: 7:a, 22:b, 17:c

Linjär sondering

	0	1	2	3	4	5	6	7	8	9
key:			22					7	17	
value:			b					a	c	

Sätt in: 7:a, 22:b, 17:c, 8:d

Linjär sondering

	0	1	2	3	4	5	6	7	8	9
key:			22					7	17	8
value:			b					a	c	d

Sätt in: 7:a, 22:b, 17:c, 8:d, 37:e

Linjär sondering

	0	1	2	3	4	5	6	7	8	9
key:	37		22					7	17	8
value:	e		b					a	c	d

Sätt in: 7:a, 22:b, 17:c, 8:d, 37:e

Andra sonderingsalternativ

- Linjär sondering: $v + i$
- Annan steglängd: $v + 3i$
- Kvadratisk sondering: $v + i^2$

Hastabell i Lua – en blandning

	0	1	2	3	4	5	6	7	8	9
key:										
value:										
next:										

Sätt in: 7:a

Hastabell i Lua – en blandning

	0	1	2	3	4	5	6	7	8	9
key:								7		
value:								a		
next:								-		

Sätt in: 7:a, 22:b

Hastabell i Lua – en blandning

	0	1	2	3	4	5	6	7	8	9
key:			22					7		
value:			b					a		
next:			-					-		

Sätt in: 7:a, 22:b, 17:c

Hastabell i Lua – en blandning

	0	1	2	3	4	5	6	7	8	9
key:			22					7	17	
value:			b					a	c	
next:			-					8	-	

Sätt in: 7:a, 22:b, 17:c
Hitta: 17, 28

Hastabell i Lua – en blandning

	0	1	2	3	4	5	6	7	8	9
key:			22					7	17	
value:			b					a	c	
next:			-					8	-	

Sätt in: 7:a, 22:b, 17:c, 8:d
Hitta: 17, 28

Hastabell i Lua – en blandning

	0	1	2	3	4	5	6	7	8	9
key:			22		17			7		
value:			b		c			a		
next:			-		-			4		

Sätt in: 7:a, 22:b, 17:c, 8:d
Hitta: 17, 28

Hastabell i Lua – en blandning

	0	1	2	3	4	5	6	7	8	9
key:			22		17			7	8	
value:			b		c			a	d	
next:			-		-			4	-	

Sätt in: 7:a, 22:b, 17:c, 8:d, 37:e
Hitta: 17, 28

Hastabell i Lua – en blandning

	0	1	2	3	4	5	6	7	8	9
key:			22		17	37		7	8	
value:			b		c	e		a	d	
next:			-		5	-		4	-	

Sätt in: 7:a, 22:b, 17:c, 8:d, 37:e

Hitta: 17, 28

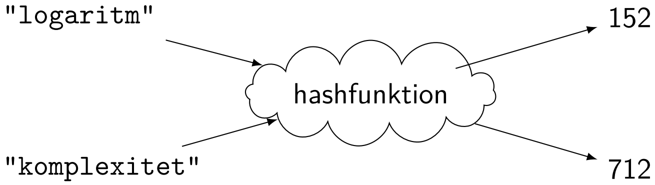
Hastabell i Lua – en blandning

	0	1	2	3	4	5	6	7	8	9
key:			22		17	37		7	8	
value:			b		c	e		a	d	
next:			-		5	-		4	-	

Sätt in: 7:a, 22:b, 17:c, 8:d, 37:e
Hitta: 17, 28, 37, 44

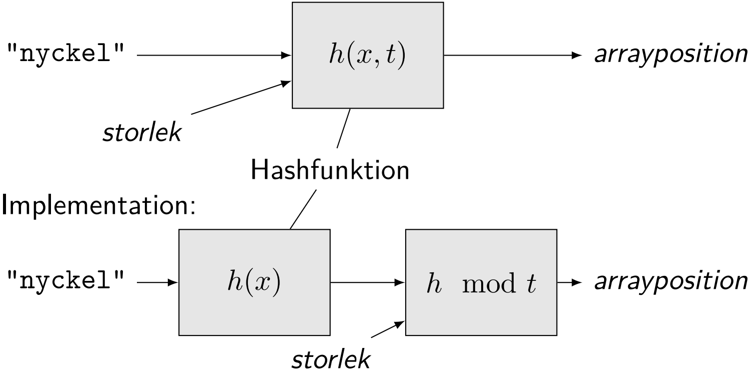
LINKÖPING UNIVERSITY

Hashfunktioner – idé



Terminologi

Literatur:



Egenskaper hos en hashfunktion

För två nycklar, x och y, samt en hashfunktion $h(k)$ gäller:

$$\begin{aligned} x = y &\implies h(x) = h(y) \\ h(x) \neq h(y) &\implies x \neq y \end{aligned}$$

Notera dock:

$$\begin{aligned} x \neq y &\not\Longrightarrow h(x) \neq h(y) \\ h(x) = h(y) &\not\Longrightarrow x = y \end{aligned}$$

En bra hashfunktion

- Snabb
  Vi vill implementera en effektiv datastruktur
- Bra fördelning av nycklar
  Vi vill minska kollisioner i hashtabellen, därmed vill vi att så få tänkbara nycklar ska resultera i samma hashvärde!

Det är ofta en bra idé att använda en hashfunktion även för heltal, trots att vi hade kunnat använda dem direkt!

Exempel

```
struct data {
  int a;
  int b;
};

size_t hash(const data &v) {
  size_t result = v.a;
  result ^= v.b << 16;
  return result;
}
```

Mål: Vi tar indatan och försöker sprida ut den så mycket som möjligt i utdatan

Exempel – djb2

```
size_t hash(const std::string &v) {
  size_t result = 5381;
  for (char c : v) {
    // result = result * 33 + c;
    result = ((result << 5) + result) + c;
  }
  return result;
}
```

Mål: Vi tar indatan och försöker sprida ut den så mycket som möjligt i utdatan

Kryptografiska hashfunktioner

Ytterligare mål:

- Om $z = h(x)$ är känd ska x vara svår att beräkna ($h^{-1}(z)$ är dyr)
- Om $z = h(x)$ är känd ska det vara svårt att hitta ett $y \neq x$ så att $z = h(x) = h(y)$

De har ofta större utdata än vad som behövs för hashtabeller, och är ofta långsammare.

Exempel: SHA-1, SHA-2, SHA-3, Blowfish, ...

Perfekt hashning

I vissa fall kan vi konstruera en hashfunktion så att

$$x = y \iff h(x) = h(y)$$

(och $h(x) < n$ för något lagom stort n)

Vi säger då att $h(x)$ är en *perfekt hashfunktion*. Det kan exempelvis utnyttjas för att implementera billiga jämförelser (datatypen *symbol* i LISP), eller för att förbättra prestandan hos en hashtabell.

Exempel: Hashning av en enum, hashning av heltal, ...

LINKÖPING UNIVERSITY

Hashtabell eller sökträd?

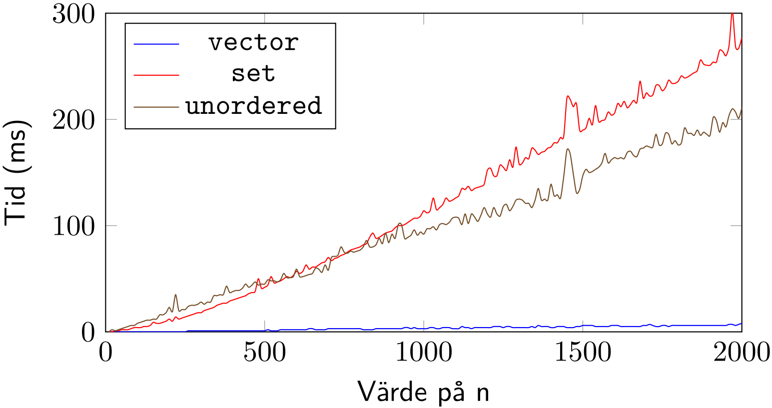
	Hashtabell	**Sökträd**
Krav på `Key`	==, `hash()`	<
Tidkomplexitet	$\mathcal{O}(1)$[1]	$\mathcal{O}(\log(n))$
Elementens ordning	odefinierad	sorterad

[1] amorterad tid, antar att en bra hashfunktion används

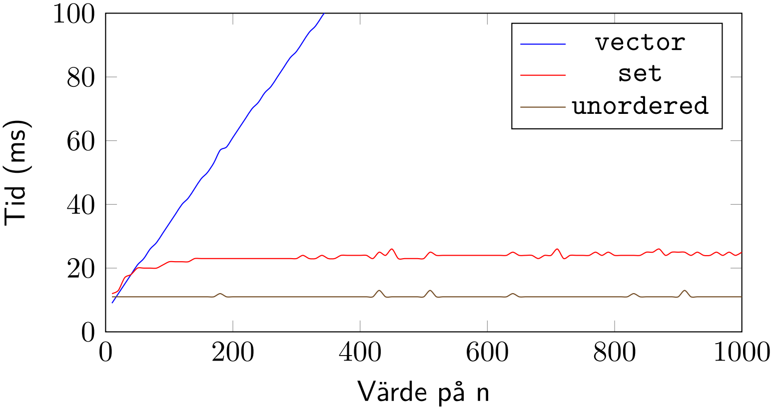
Testkörning (kompilerat med -O2)

```
$ time lookup_vector >/dev/null <thesis.txt
real    0m4.925s
user    0m4.900s
sys     0m0.025s
$ time lookup_set >/dev/null <thesis.txt
real    0m0.169s
user    0m0.145s
sys     0m0.025s
$ time lookup_uset >/dev/null <thesis.txt
real    0m0.154s
user    0m0.137s
sys     0m0.017s
```

Insättning, 1 000 körningar per indata



Uppslagning, 100 000 körningar per indata



Begränsningar

Vi har en mängd av heltal: N

Problem vi vill lösa:
Givet x, finns det något $y \in N$ så att $|y - x| \leq 10$?

Begränsningar

Vi har en mängd av heltal: N

Problem vi vill lösa:
Givet x, finns det något $y \in N$ så att $|y - x| \leq 10$?

Kan vi konstruera en hashfunktion $h(x)$ så att:
$|y - x| \leq 10 \Rightarrow h(x) = h(h)$?

Begränsningar

Vi har en mängd av heltal: N

Problem vi vill lösa:
Givet x, finns det något $y \in N$ så att $|y - x| \leq 10$?

Kan vi konstruera en hashfunktion $h(x)$ så att:
$|y - x| \leq 10 \Rightarrow h(x) = h(h)$?

Fungerar $h(x) = \lfloor \frac{x}{10} \rfloor$? $h(x) = \lfloor \frac{x}{20} \rfloor$?

LINKÖPING UNIVERSITY

Memoisering

```
int fibonacci(int x) {
  int result = x;
  if (x > 1)
    result = fibonacci(x - 1) + fibonacci(x - 2);

  return result;
}
```

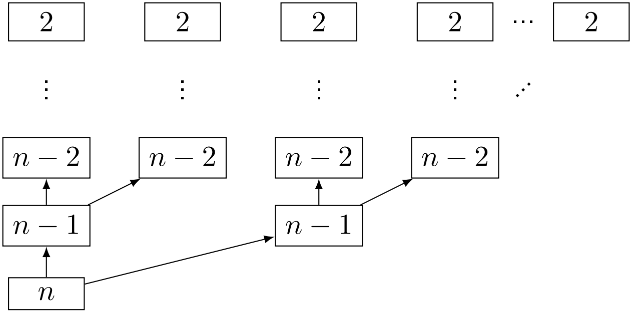
Komplexitet: ?

Tidskomplexitet: Förenklat

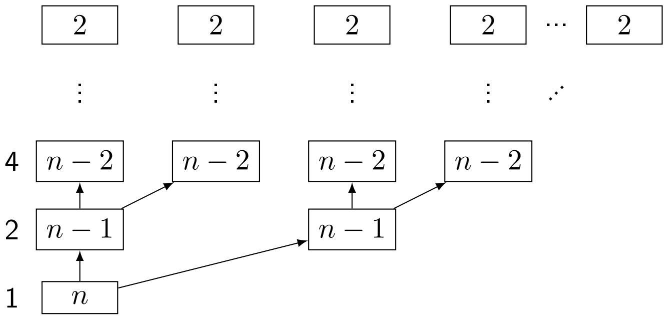
```
int fibonacci(int n) {
  int result = x;
  if (x > 1)
    result = fibonacci(x - 1) + fibonacci(x - 1);

  return result;
}
```

Detta är okej, vi gör algoritmen **långsammare** (och felaktig)! Vårt $\mathcal{O}$-uttryck kommer vara lite för högt, men det kommer fortfarande stämma.

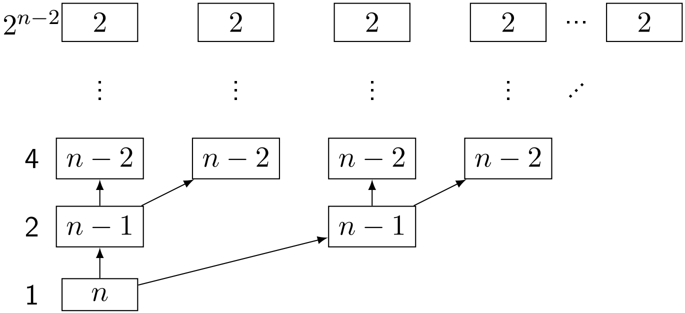
Tidskomplexitet: Analys



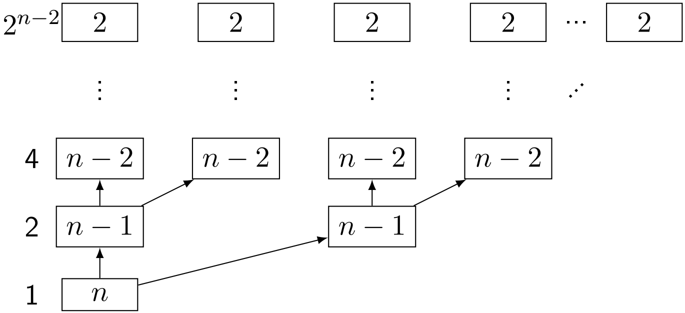
Tidskomplexitet: Analys



Tidskomplexitet: Analys



Tidskomplexitet: Analys



Totalt: $2^{n-1} - 1 \in \mathcal{O}(2^n)$ anrop

Memoisering

```
int fibonacci(int x) {
  int result = x;
  if (x > 1)
    result = fibonacci(x - 1) + fibonacci(x - 2);

  return result;
}
```

Komplexitet: $\mathcal{O}(2^n)$

Memoisering – Vad är komplexiteten?

```
unordered_map<int, int> table;
int fibonacci(int x) {
  auto i = table.find(x);
  if (i != table.end())
    return i->second;

  int result = x;
  if (x > 1)
    result = fibonacci(x - 1) + fibonacci(x - 2);

  table[x] = result; // Kom ihåg!
  return result;
}
```

LINKÖPING UNIVERSITY

Bloom filter

Antag: Vi har en disk med $10 \cdot 10^6$ filer. Vi vill snabbt svara på frågan: finns fil x på disken?

Bloom filter

Antag: Vi har en disk med $10 \cdot 10^6$ filer. Vi vill snabbt svara på frågan: finns fil x på disken?

Hashtabell: $20 \cdot 10 \cdot 10^6 \approx 200 \text{ GB}$, för stort...

Bloom filter

Antag: Vi har en disk med $10 \cdot 10^6$ filer. Vi vill snabbt svara på frågan: finns fil x på disken?

Hashtabell: $20 \cdot 10 \cdot 10^6 \approx 200\ \mathrm{GB}$, för stort...

Vi kan använda ett *Bloom filter* i stället!

Bloom filter

Idé: Vi ignorerar kollisioner som ett första filter!

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

Bloom filter

Idé: Vi ignorerar kollisioner som ett första filter!

0	1	2	3	4	5	6	7	8	9
0	0	0	1	0	0	0	0	0	1

Insättning:

$h(\texttt{liu.png}) = 3$

$h(\texttt{lecture1.pdf}) = 9$

Bloom filter

Idé: Vi ignorerar kollisioner som ett första filter!

0	1	2	3	4	5	6	7	8	9
0	0	0	1	0	0	0	0	0	1

Insättning:

$h(\texttt{liu.png}) = 3$

$h(\texttt{lecture1.pdf}) = 9$

Medlemstest:

$h(\texttt{lecture1.pdf}) = 9$ ✓

$h(\texttt{answers.pdf}) = 3$ ✓?

Bloom filter

Idé: Vi ignorerar kollisioner som ett första filter!

0	1	2	3	4	5	6	7	8	9
1	0	0	1	0	1	0	0	0	1

Insättning:

$h_1(\texttt{liu.png}) = 3$

$h_2(\texttt{liu.png}) = 5$

$h_1(\texttt{lecture1.pdf}) = 9$

$h_2(\texttt{lecture1.pdf}) = 0$

Medlemstest:

Bloom filter

Idé: Vi ignorerar kollisioner som ett första filter!

0	1	2	3	4	5	6	7	8	9
1	0	0	1	0	1	0	0	0	1

Insättning:

$h_1(\texttt{liu.png}) = 3$
$h_2(\texttt{liu.png}) = 5$

$h_1(\texttt{lecture1.pdf}) = 9$
$h_2(\texttt{lecture1.pdf}) = 0$

Medlemstest:

$h_1(\texttt{lecture1.pdf}) = 9$ ✓
$h_2(\texttt{lecture1.pdf}) = 0$ ✓

$h_1(\texttt{answers.pdf}) = 3$ ✓
$h_2(\texttt{answers.pdf}) = 7$ ✗

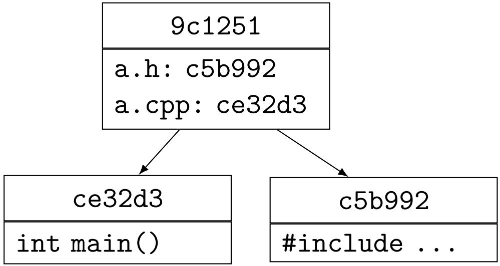
Merkle-träd (hashträd)

Idé: Vi använder en hash för att identifiera noder!

`ce32d3`	`c5b992`
`int main()`	`#include ...`

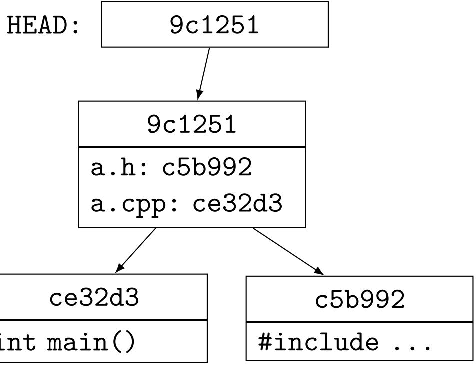
Merkle-träd (hashträd)

Idé: Vi använder en hash för att identifiera noder!



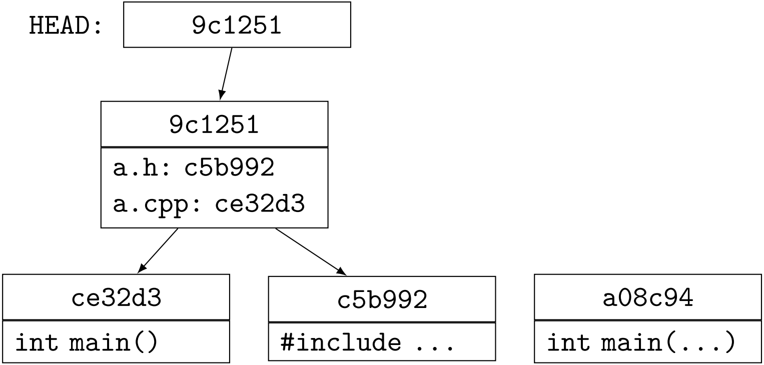
Merkle-träd (hashträd)

Idé: Vi använder en hash för att identifiera noder!



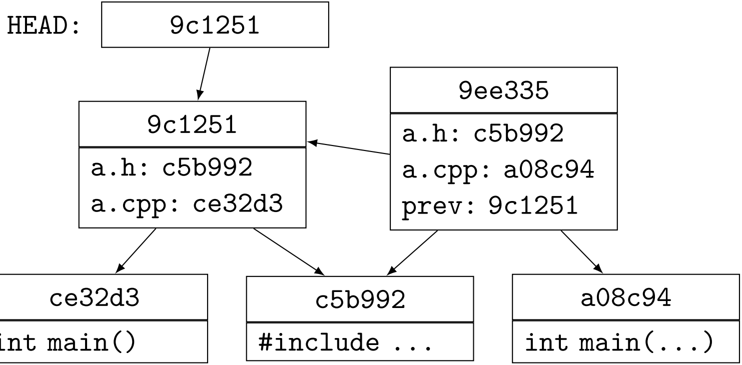
Merkle-träd (hashträd)

Idé: Vi använder en hash för att identifiera noder!



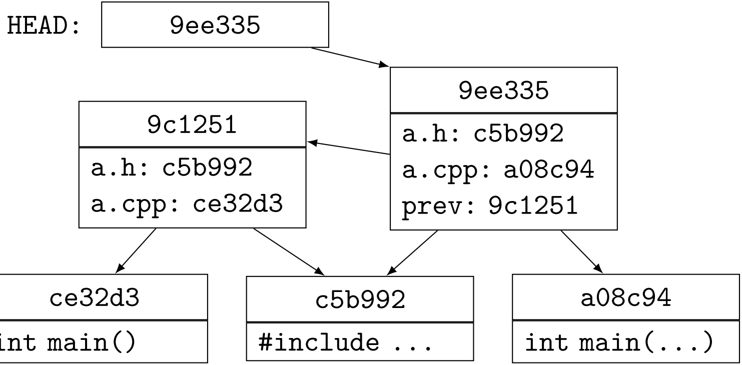
Merkle-träd (hashträd)

Idé: Vi använder en hash för att identifiera noder!



Merkle-träd (hashträd)

Idé: Vi använder en hash för att identifiera noder!



LINKÖPING UNIVERSITY

I kursen framöver

- Nästa föreläsning
  - Grafer och enkla grafalgoritmer
- Uppgifter i Kattis
  - `compoundwords` (enkel)
    Hitta sammansatta ord.
  - `setstack` (svårare)
    Tänk på tidsåtgången – hur illa kan det bli? Kan vi använda oss av perfekt hashning?

Filip Strömbäck

www.liu.se

LINKÖPING UNIVERSITY