

# Hashing, del II

**Diskret Matematik og Algoritmer**  
**Københavns Universitet, efterår 2021**



# Hvad sker der?

## Litteratur

- CLRS 11.0, 11.1, 11.2, 11.3 (dog ikke 11.3.3), 11.4 (dog ikke Analysis of open-address hashing). Vi bruger i denne uge notation defineret i CLRS sektion 3.2, sektionerne “Floors and Ceilings” samt “Modular arithmetic” (side 54)
- CLRS kapitel 17.0, 17.1, 17.2
- Note: Cuckoo hashing for undergraduates

## Mål for ugen

- Kendskab til hashtabeller (chaining, linear probing, cuckoo hashing) og deres egenskaber
- Introduktion til amortiseret analyse

## Plan for ugen

- Mandag: Hashtabeller med hængte lister
- Tirsdag: Open addressing, cuckoo hashing
- Fredag: Introduktion til amortiseret analyse

# Motiverende case, udvidet

- Data: En liste af  $t \approx 10^{12}$  Facebook venskaber,  $> 10^9$  brugere.  
[ (Rasmus, Anna) , (Rasmus, Thore) , (Thore, Tony) , (Anna, Thore) , ... ]
- Vi er nødt til at gemme data mere pladseffektivt!

# Teknik 1: Signaturer

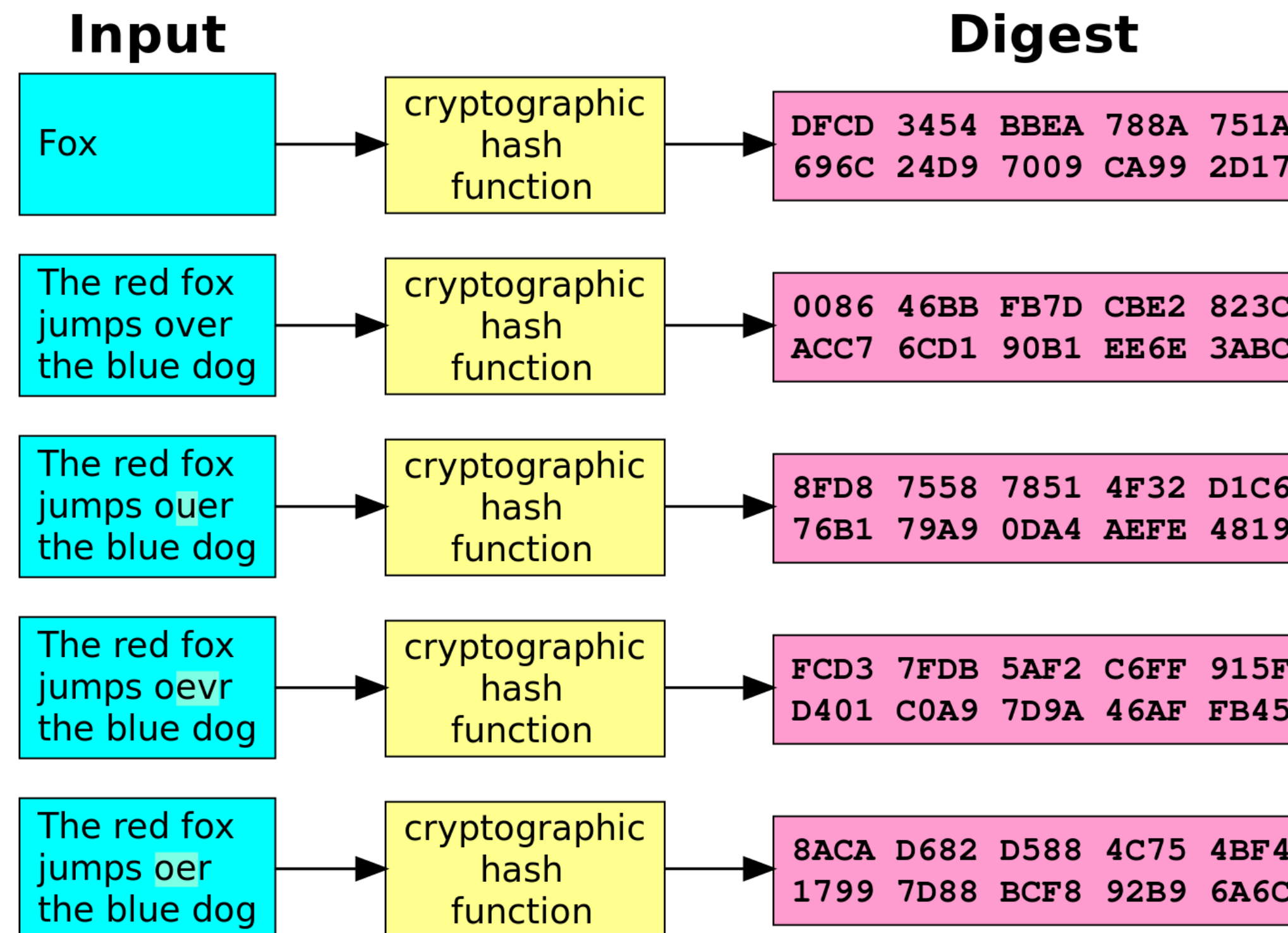
- **Idé:** Hvis vi har en hashfunktion  $h$ , der ikke har nogen kollisioner på nøglemængden  $K$ , så kan hver nøgle  $k_i$  erstattes af  $h(k_i)$ .
- **Eksperiment:** Kan I vælge unikke signaturer mellem 0 og 999? Gå ind på [denne Google form](#) og indtast jeres svar.
- Analyse af antal kollisioner:



- $X = \sum_{i=1}^n \sum_{j=1}^n X_{ij}$  hvor  $X_{ij} = 1$  hvis  $i \neq j$  og  $h(k_i) = h(k_j)$ , og  $X_{ij} = 0$  ellers
- $E[X] = \sum_{i=1}^n \sum_{j=1}^n E[X_{ij}] \leq \sum_{i=1}^n \sum_{j=1}^n 1/m = n^2/m$
- $\Pr[X \geq 1] \leq n^2/m$  (Markov's ulighed)

# Signaturer er overalt!

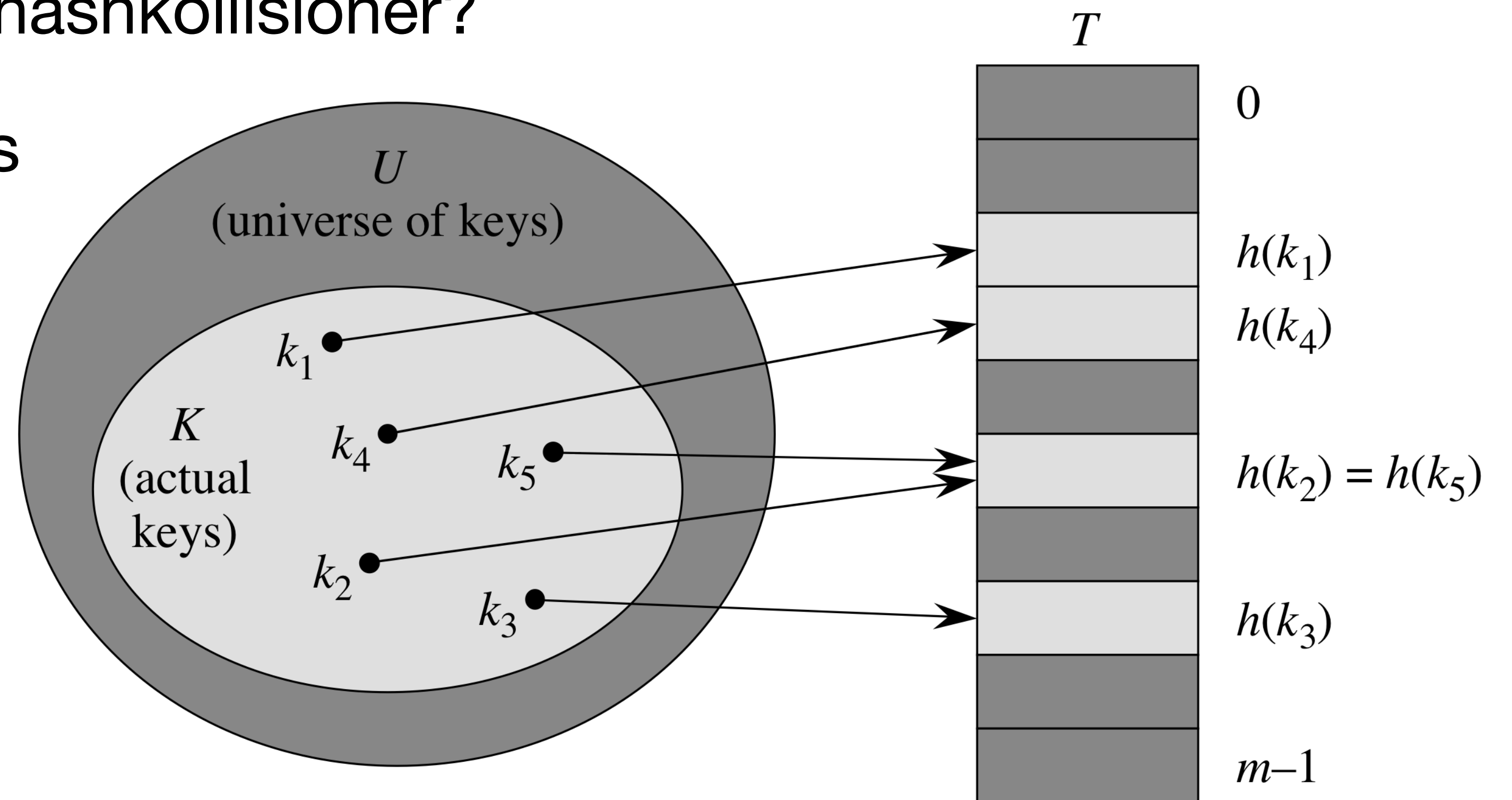
- Ofte bruges *kryptografiske* hashfunktioner til at lave unikke signaturer (digests).



- Anvendelser: Dataintegritet, filsystemer, blockchain, ...

# Teknik 2: Open addressing

- **Idé:** Gem nøglerne direkte i tabellen  $T$  (undgå pointere)
- **Problem:** Hvordan håndteres hashkollisioner?
- **Svar:** Definér en probesekvens  $h(k,0), h(k,1), h(k,2), \dots$  og indsæt på den første ledige plads.





# Analyse af open addressing

- **Vigtigste varianter:**
  - Linear probing,  $h(k, i) = (h(k) + i) \bmod m$ ,
  - Double hashing,  $h(k, i) = (h_1(k) + i h_2(k)) \bmod m, \dots$
- CLRS 11.4 analyserer en variant af open addressing (ikke pensum)
- I skal vide:  $O(1)$  tid per operation så længe  $\alpha = n/m$  ikke er for tæt på 1
  - Når  $n$  nærmer sig  $m$ : Nødt til at genopbygge hashtabellen med større  $m$
  - Mere om det sidst i forelæsningen



# Hashing med begrænset søgetid

- **Idé:** Begræns probelængden til 2. Nøglen  $k$  er altid enten i  $h_1(k)$  eller  $h_2(k)$ .

[illegible]

# Gøgen



# Cuckoo hashing

- **Idé:** Begræns probelængden til 2. Nøglen  $k$  er altid enten i  $h_1(k)$  eller  $h_2(k)$ .

0	1	2	3	4	5	6	7	8	9	10	11	12

- Det er ok at sparke nøgler ud af deres “rede”!  
(De har jo også et andet sted, de godt kan være)



# Cuckoo hashing pseudokode og simulering

```
procedure insert( $x$ )  
  if  $T[h_1(x)] = x$  or  $T[h_2(x)] = x$  then return;  
  pos  $\leftarrow h_1(x)$ ;  
  loop  $n$  times {  
    if  $T[\text{pos}] = \text{NULL}$  then {  $T[\text{pos}] \leftarrow x$ ; return };  
     $x \leftrightarrow T[\text{pos}]$ ;  
    if pos =  $h_1(x)$  then pos  $\leftarrow h_2(x)$  else pos  $\leftarrow h_1(x)$ ;  
  }  
  rehash(); insert( $x$ )  
end
```

- Simulering, lavet af Martin Aumüller.  
(En variant hvor tabellen  $T$  er todelt, og hashfunktionerne har hver sin del.)

# Øvelse (5 min)

- Hvor mange skridt tager følgende for cuckoo hashing i *værste fald*?
  - Search
  - Delete
  - Insert
- Konstruér gerne et “værste eksempel” i Aumüllers simulation!



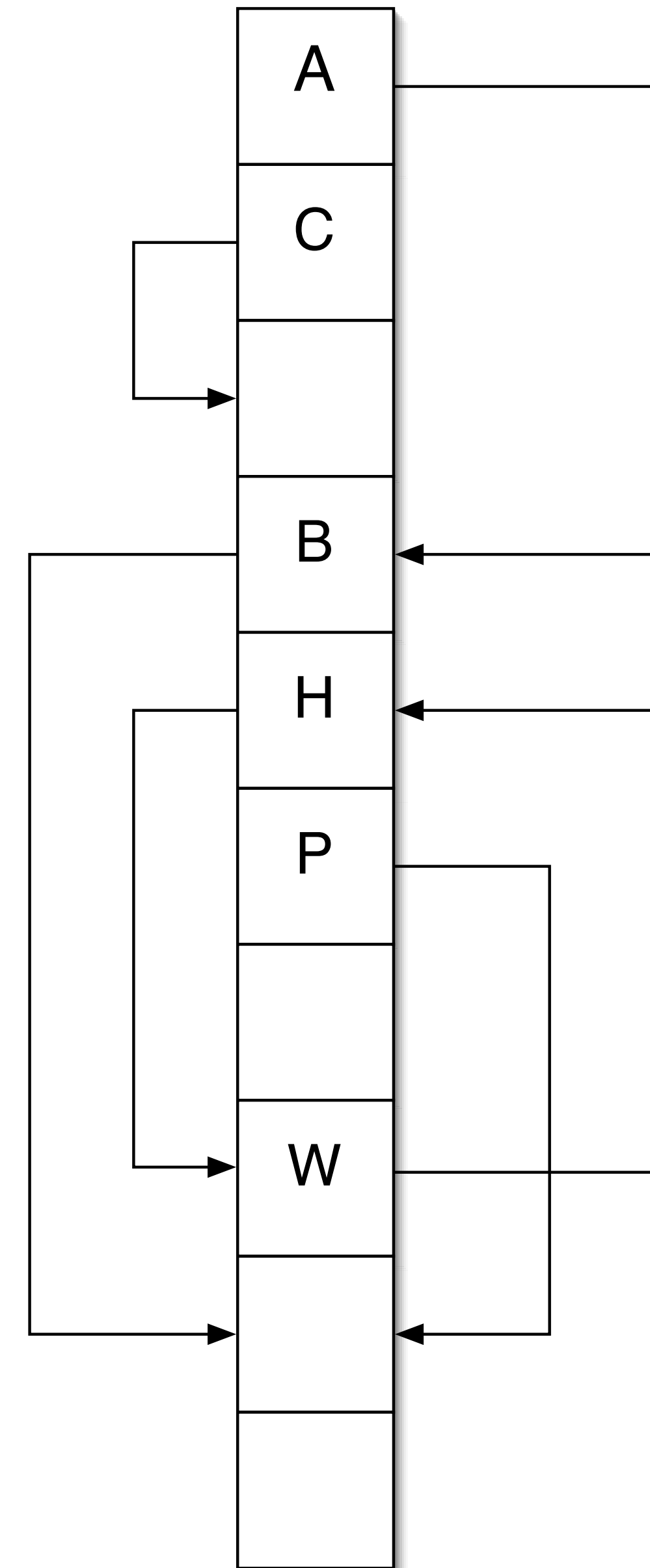
- Denne animation, lavet af Thore Husfeldt, viser at det sjældent går så galt!

# Analyse af cuckoo hashing indsættelse

- **Invariant:**  $n/m < 1/2$  (tabellen er mindre end halvt fyldt)
- **Intuition:**
  - Så længe vi ikke vender tilbage til en nøgle, vi tidligere har flyttet, er der mindst 50% chance for at finde en tom plads i hver iteration.
  - Under denne antagelse: Første iteration sker med sandsynlighed 1, anden iteration sker med sandsynlighed højst  $1/2$ , ...,  $i$ te iteration sker med sandsynlighed  $< 2^{1-i}$ .
  - Forventet søgetid (uden rehashing):  $\sum_{i=1}^n 2^{1-i} < 2$ .
- Vi vender tilbage til omkostningen ved rehashing.

# Cuckoo grafen

- Noten “Cuckoo hashing for undergraduates” indeholder en mere udførlig analyse af cuckoo hashing (ikke pensum).
- Analysen ser på “cuckoo grafen” (eksempel til højre), der repræsenterer hvordan nøgler kan bevæge sig i tabellen.
- Senere i kurset skal vi se meget mere på grafer.



# Problemer fejlet ind under gulvtæppet

- Det I har set virker *kun* hvis størrelsen  $m$  af hashtabellen er stor nok i forhold til antallet  $n$  af nøgler!
- Dyrt at øge  $m$  (rehash):  
Kræver at hashtabellen bygges op på ny, tid  $O(n + m)$
- **Idé:** Sørg for at øge  $m$  tilstrækkeligt (f.eks. fordoble), således at der er længe til næste rehash.



# Analyse af rehash, del 1

- Start med en tom hashtabel af størrelse (eksempelvis)  $m = 10$ .
- Vi fordobler  $m$  når tabellen er mere end halvt fyldt,  $n > m/2$ :
  - Når  $n = 6$  fordobles størrelsen til  $m = 20$
  - Når  $n = 11$  fordobles størrelsen til  $m = 40$
  - Når  $n = 21$  fordobles størrelsen til  $m = 80$
  - Når  $n = 41$  fordobles størrelsen til  $m = 160$
  - ...

# Analyse af rehash, del 2

- **Invariant:**  $2n \leq m < 4n$
- Forventet omkostning ved ét rehash er  $O(n + m)$ , hvilket er  $O(m)$
- Total omkostning ved rehash efter at  $n$  nøgler er blevet indsat (uden store-O):

$$m + m/2 + m/4 + \dots < 2m < 8n$$

- Det er  $O(1)$  per indsættelse!

# Ekstra: Kattis opgaver


 **Kattis**  
PROBLEMS CONTESTS RANKLISTS JOBS (3) HELP

Search Kattis

## Cuckoo Hashing

One of the most fundamental data structure problems is the dictionary problem: given a set  $D$  of words you want to be able to quickly determine if any given query string  $q$  is present in the dictionary  $D$  or not. Hashing is a well-known solution for the problem. The idea is to create a function  $h : \Sigma^* \rightarrow [0..n - 1]$  from all strings to the integer range  $0, 1, \dots, n - 1$ , i.e. you describe a fast deterministic program which takes a string as input and outputs an integer between  $0$  and  $n - 1$ . Next you allocate an empty hash table  $T$  of size  $n$  and for each word  $w$  in  $D$ , you set  $T[h(w)] = w$ . Thus, given a query string  $q$ , you only need to calculate  $h(q)$  and see if  $T[h(q)]$




 **Kattis**  
PROBLEMS CONTESTS RANKLISTS JOBS (3) HELP

Search Kattis

## Enlarging Hash Tables

When implementing an open addressing hash table, we can use many different types of collision resolution strategies. One such strategy is quadratic probing. If we use quadratic probing, it's very important that the size of the hash table is a prime number. If it is a prime number and the table is less than half full, then we can prove that the quadratic probing algorithm will always be able to find an empty slot. That's necessary because finding an empty slot is the way the algorithm knows when to stop probing. However, if the table size is not prime, then even if the table is much less than half full, the probing algorithm may probe the same slots repeatedly and not find any

 **Kattis**  
PROBLEMS CONTESTS RANKLISTS JOBS (3) HELP

Search Kattis

## Hash

Little Mirko is studying the hash function which associates numerical values to words. The function is defined recursively in the following way:

- $f(\text{empty word}) = 0$
- $f(\text{word} + \text{letter}) = ((f(\text{word}) \cdot 33) \oplus \text{ord}(\text{letter})) \% \text{MOD}$

The function is defined for words that consist of only lowercase letters of the English alphabet.  $\oplus$  stands for the bitwise exclusive or operator (i.e.  $0110 \oplus 1010 = 1100$ ),  $\text{ord}(\text{letter})$  stands for the ordinal number of the letter in the alphabet ( $\text{ord}(\text{a}) = 1$ ,  $\text{ord}(\text{z}) = 26$ ) and  $A \% B$  stands for the remainder of the number  $A$  when performing integer division with the number  $B$ . MOD will be an integer of the form  $2^M$ .

# Opsummering

- Open addressing giver mere effektive ordbøger når nøglerne kan gemmes direkte i hashtabellen
  - *Linear probing* er simpelt, og effektivt på moderne computere
  - *Cuckoo hashing* sikrer at søgning altid sker med 2 opslag i tabellen
- Hashtabeller skal skaleres når  $n$  vokser — det er dyrt men skal gøres sjældent
- Flere detaljer i CLRS 11.4 og noten *Cuckoo Hashing for Undergraduates*
- **Fredag:** Vi begynder at se på *amortiseret analyse* af datastrukturer