Välkommen till en översikt av...

Data Integrity

(In the context of networked file systems)

Björn Lalin (d00-bla)

Gör exjobb på IBM Research Lab, Schweiz

Vad det kommer handla om

- Motivering
- Modell och sammanhang
- Kända tekniker
  - “Vanliga” hashfunktioner
  - Inkrementella hashfunktioner
  - Ackumulatorer
  - Hashträd
  - Entropi
- Översikt & Sammanfattning
- Vad jag tänkte göra
- Ciao, frohe Weihnachten och gutes neues Jahr !!

Motivering

Snabbt ökande mängd av känslig data.

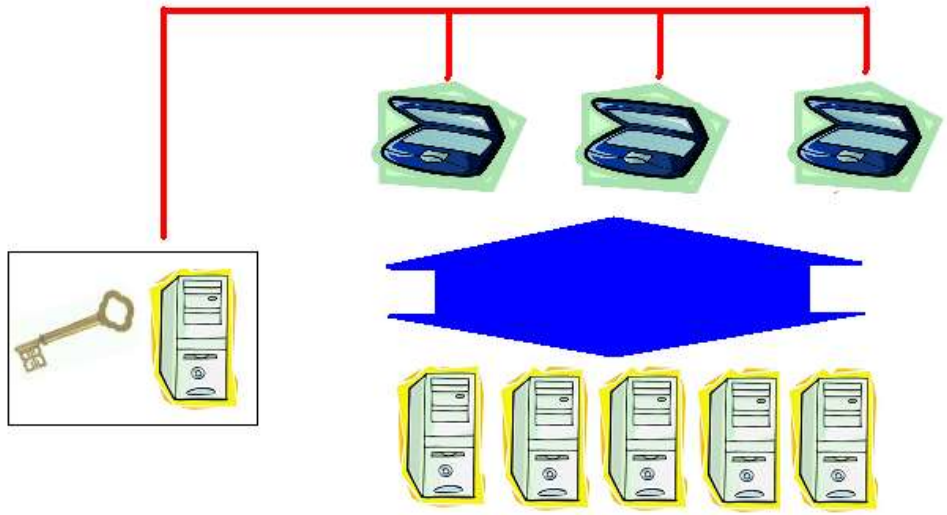
Behov av att hantera data på säkert sätt.

(Kher & Kim, Univ. of Minnesota, 2005)

Modell

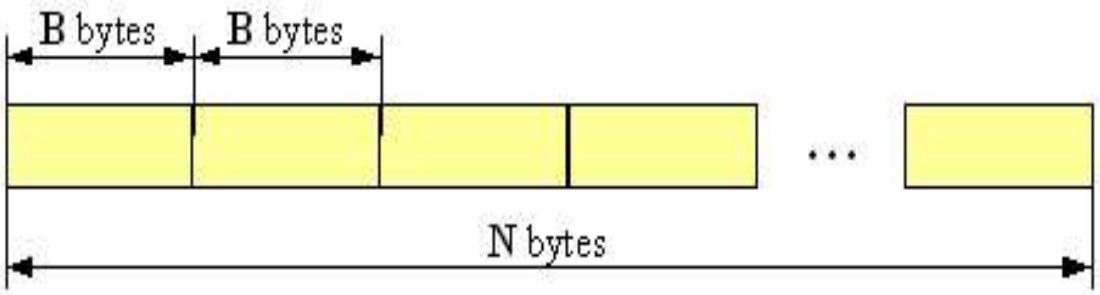
- Filservrar ej betrodda
- nycklar, etc. på betrodd server
- På betrodd server vill vi ha
  - Lite kommunikation
  - Lite minnesåtgång

(många använder samma autentiseringsserver, har eventuellt lite beräkningskraft och/eller lagringsutrymme)



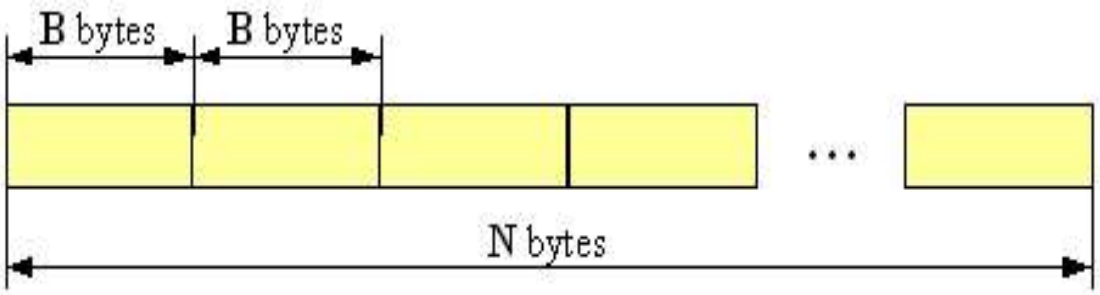
Modell av data på lagringsmedia

Vår bild av en typisk fil av storlek **N** bytes:



Modell av data på lagringsmedia

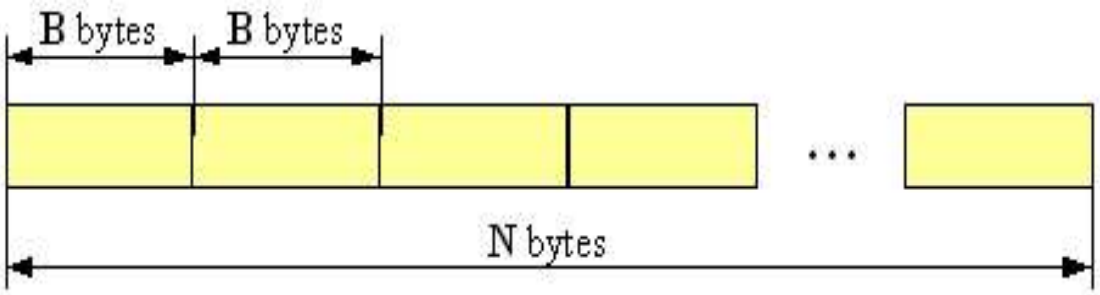
Vår bild av en typisk fil av storlek **N** bytes:



READ(address) WRITE(addr, data)

Modell av data på lagringsmedia

Vår bild av en typisk fil av storlek **N** bytes:



READ(address) WRITE(addr, data)

Invasive respektive **Non-invasive**

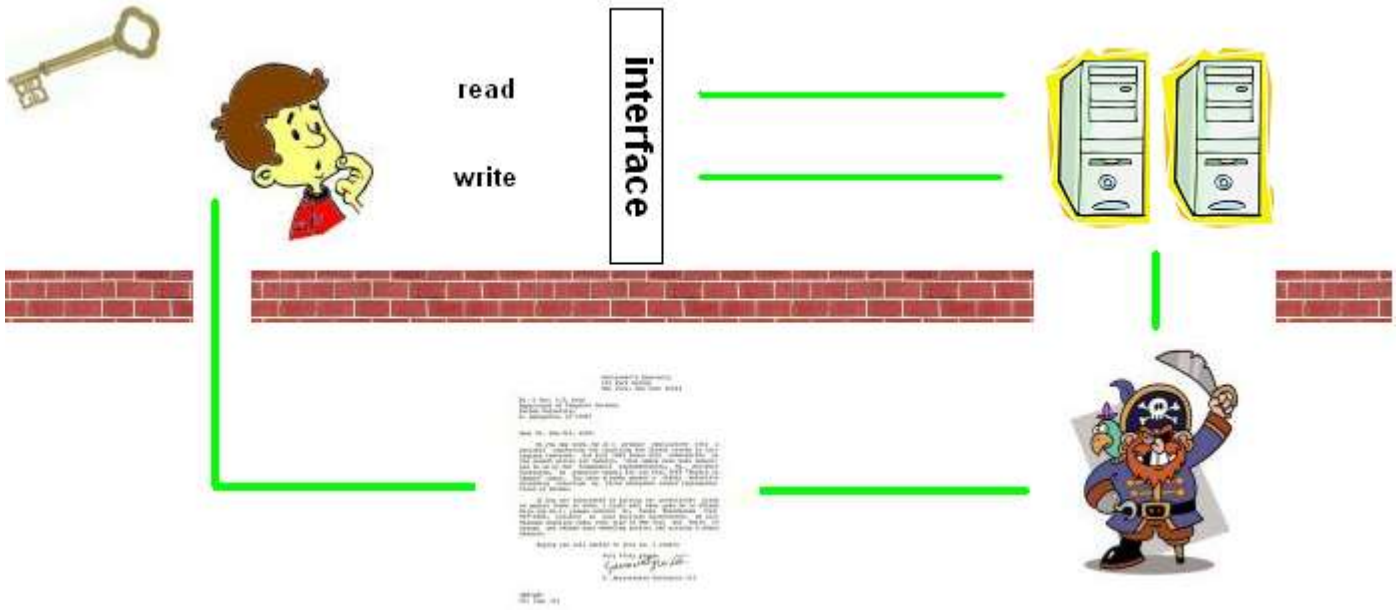
Online respektive **Offline**

Definition av Integritet

Vad är det vi vill göra?

Informellt: “Ge en garanti för att data som läses är identisk med den data som skrevs allra senast.”

Motståndare / Möjliga attacker



Förstöra data

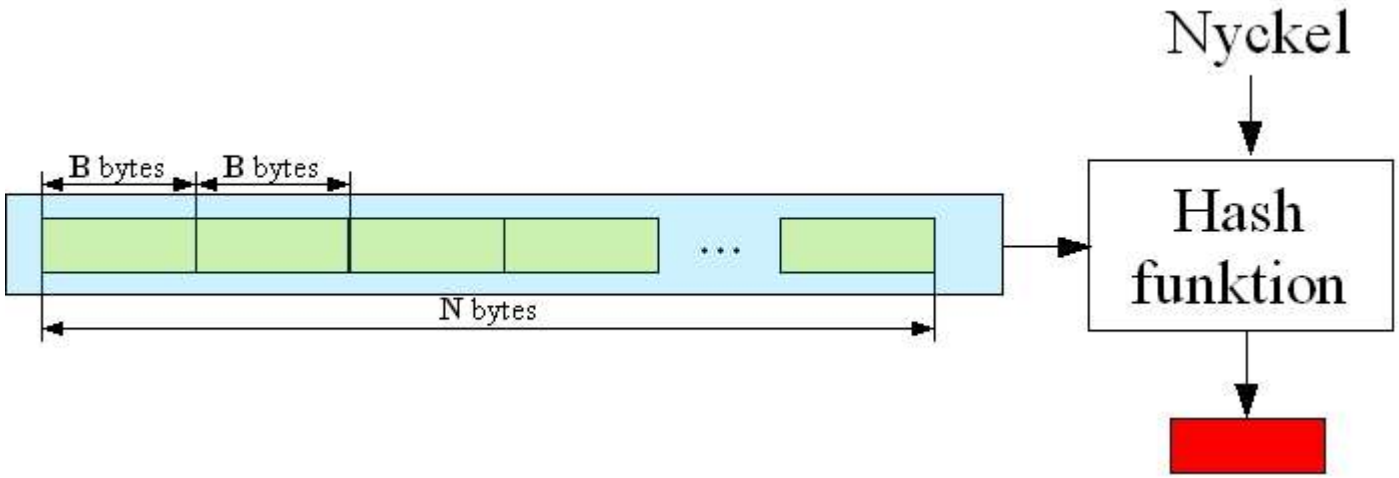
I den här modellen antar vi att en aktiv motståndare har både läs- och skrivmöjligheter till data. Vi kan inte göra något mot detta.

Modifiera data

Returnera gammal data (s k replay attack)

Ett enkelt (men kanske inte så bra) sätt

Beräkna en hash/fingerprint för varje fil.
(t.ex SHA-256 → 32 bytes)



Var lagrar vi “fingeravtrycket”?

- Tillsammans med data på icke betrodd server?
- På vår betrodda server?

Ett enkelt (men kanske inte så bra) sätt

Fördelar:	Enkelt Ett unikt värde för hela filen (tar lite plats)
Nackdelar:	Långsamt att skriva: O(N) Långsamt att läsa(verifiera): O(N)
Exempel:	Tripwire och liknande system (separat, inte i filsystemet) Många filsystem som implementerar integritet gör det så här.

Men det här då...?

Om vi lagrar ett “fingeravtryck” för varje block istället? (Där vi kan variera “block” till att vara allt mellan en byte och hela filen).

- Större block → snabbare verifiering, snabbare uppdatering
- Mindre block → Mer lagringsutrymme och last på betrodd server

<u>Fördelar / Nackdelar:</u>

Kan balansera kostnader mellan minne/belastning och prestanda

En (lite) bättre variant?

Vanlig hashfunktion:

- H(x) = y
- H envägsfunktion (svårt finna x givet y)
- H kollisions-fri(resistent)
- |H(x)| << |x|

Inkrementell hashfunktion:

Alla egenskaper som en vanlig hashfunktion. Dessutom, om vi uppdaterar x → x' så kan vi beräkna H(x') till en kostnad proportionell mot diff(x, x')

Inkrementella hashfunktioner

Två operationer HASH och UPDATE istället för en. Fil **M**, består av **b** block

HASH:

UPDATE:

(Bellare, Goldreich, Goldwasser)

Inkrementella hashfunktioner

Två operationer HASH och UPDATE istället för en. Fil **M**, består av **b** block

HASH:

$$h = H(M) = \prod g_i^{M[i]} \pmod P,$$

g_i ett slumpvis element i gruppen
M[i] block "i" av filen **M**

UPDATE:

$$h' = U(h,M,m',j) = h * g_j^{-M[i]} * g_j^{m'}$$

(Bellare, Goldreich, Goldwasser)

Inkrementella hashfunktioner

Fördelar: O(1) för uppdateringar, dvs. skrivning
Tar lite plats, O(1) minne

Nackdelar: Långsamt att läsa(verifiera): O(N)

Kommentarer: Visserligen O(1) för uppdateringar, men detta innefattar ett konstant antal exponentieringar, som är relativt dyra.

Finns andra typer av inkrementella hashfunktioner med billigare primitiver.

Ackumulator

Ackumulator (eng. Cryptographic Accumulator)

en funktion $F : X \times Y \rightarrow X$, där

- $f(f(x_1,y_1),y_2) = f(f(x_1,y_2),y_1)$.
- Svår att invertera

$x = e(x,y) = x^y \bmod N$

(Benaloh, de Mare)

Poäng: O(n) att uppdatera.
O(1) att verifiera.

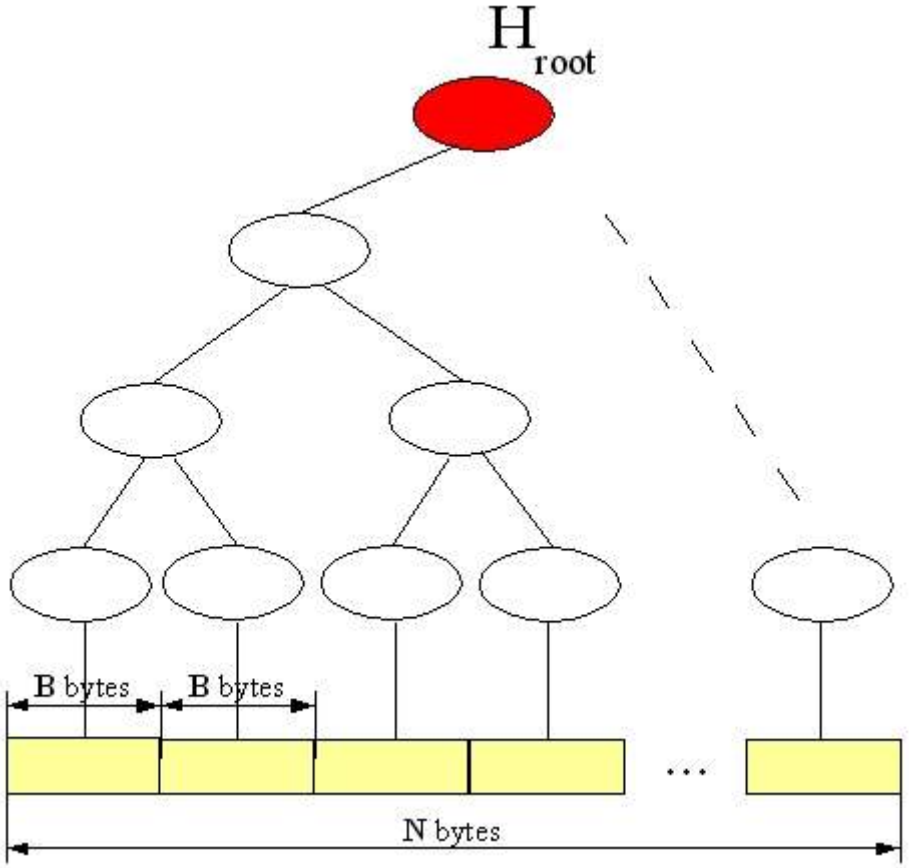
(”Christmas tree lecture...”)

Log (n) arbete för uppdatering,
Log(n) arbete för verifiering,

Vi använder en trädstruktur!

Hashträd (Ralph Merkle)

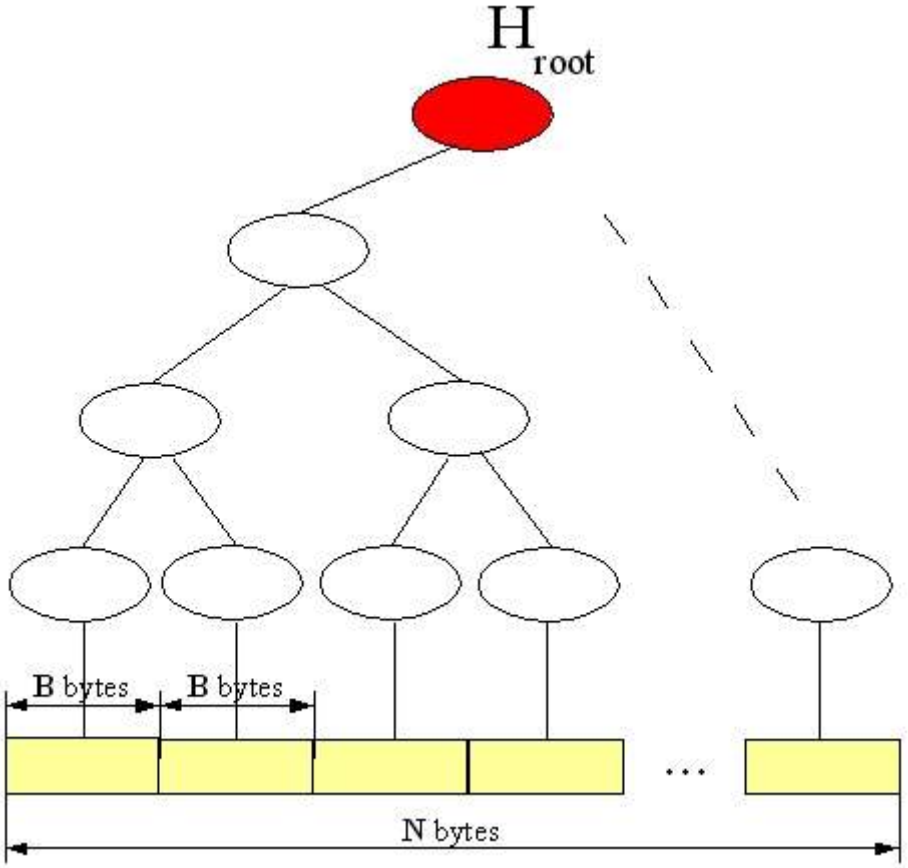
- $H_{<i,j>} = H(H_{<i, (i+j-1)/2>}, H_{<(i+j+1)/2,j>})$ om $i \neq j$
- $H_{<i,j>} = H(B_i)$ om $i = j$
- $H_{root} = H_{<0,n-1>}$, n = #block i filen



Hashträd (Ralph Merkle)

- $H_{<i,j>} = H(H_{<i, (i+j-1)/2>}, H_{<(i+j+1)/2,j>})$ om $i \neq j$
- $H_{<i,j>} = H(B_i)$ om $i = j$
- $H_{root} = H_{<0,n-1>}$, n = #block i filen

Vi lagrar bara H_{root} på en säker plats (dvs på en betrodd server). Resten av trädet lagrar vi tillsammans med data på filserver.



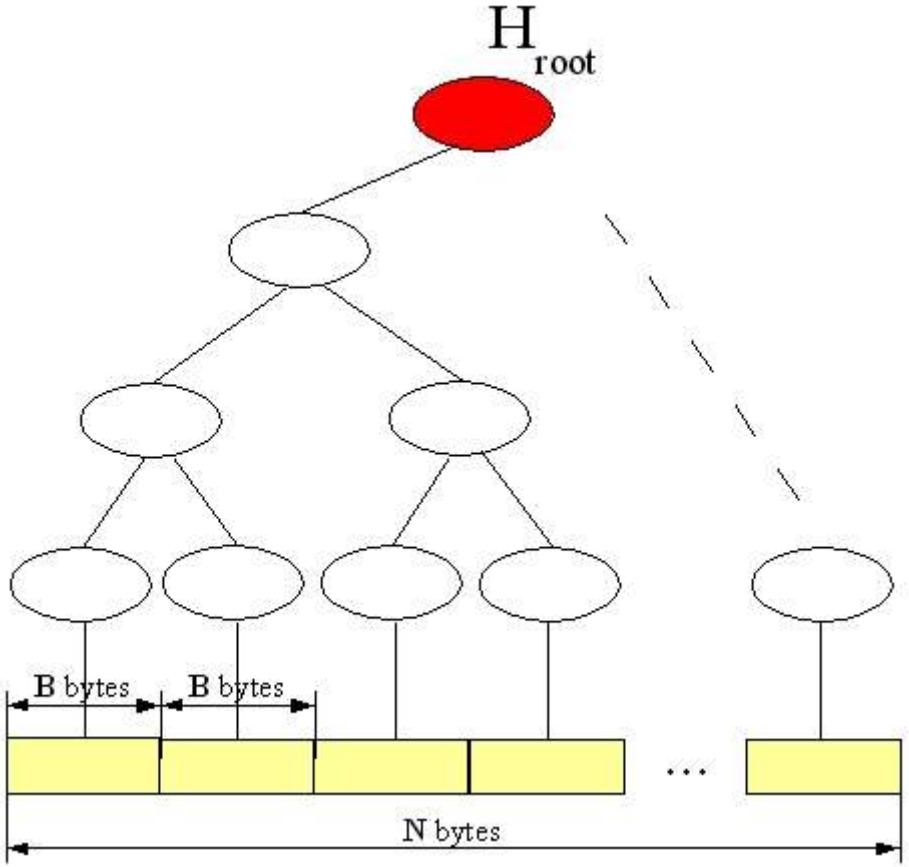
Hashträd (Ralph Merkle)

- $H_{<i,j>} = H(H_{<i, (i+j-1)/2>}, H_{<(i+j+1)/2,j>})$ om $i \neq j$
- $H_{<i,j>} = H(B_i)$ om $i = j$
- $H_{root} = H_{<0,n-1>}$, n = #block i filen

Vi lagrar bara H_{root} på en säker plats (dvs på en betrodd server). Resten av trädet lagrar vi tillsammans med data på filserver.

<u>Verifiera</u>:
För att verifiera block B_1, beräkna hela stigen upp till root.

<u>Uppdatera</u>:
För att uppdatera måste hela stigen från uppdaterade noden beräknas på nytt.



Hashträd (Ralph Merkle)

Fördelar:
verifiera: O(log N)
skriva: O(log N)
lagrar bara H_{root} i trusted storage -> lite belastning

Nackdelar:
Om filens storlek ökar måste trädet byggas om.
(Gör t.ex trädet lite extra stort så sannolikheten för “ombyggnad” är liten)

Kombination: Hashträd och inkrementell hash

Kontext: Många skrivningar mellan varje verifiering
Många accesser till samma adress

Idé: Ha ett hashträd till att börja med. Första gången vi läser data från fil så verifierar vi och flyttar till en annan datastruktur byggd med inkrementell hashing.

Exempel på användning: Trusted computing. Verifiera minnet som använts, men endast vid "kritiska operationer".

(Clarke, Suh, Gassend, Sudan, van Dijk, Devadas)

Utnyttja struktur i data (entropi)

Kontext: Inte skriva på obetrodd lagringsplats (non-invasive).
Minimera betrodd lagringsplats (tillståndsinfo).

Förslag: Använd strukturen hos data för integritet. (Oprea, Reiter, Yang)

content(adress, data) = C_tweak(data,adress,key)

isRand(adress, data) → yes / no

Nackdelar: Inget skydd mot replay attacks.
Vad om datan inte har någon struktur (slumpmässig data)?

Snabb sammanfattning

Fingerprint av hela filen	O(N)	O(N)	
Fingerprint / block	O(1) – O(N)	O(1) – O(N)	Tradeoff minne/prestanda
Inkrementell hashfunktion	O(N)	O(1)	
Hashträd	(log N)	O(log N)	
Ackumulator	O(1)	O(N)	Dyr exponentierings-Funktion

GOD JUL & GOTT NYTT ÅR