Geir Berge

En datastruktur for ordbøker for naturlige språk

1. Innledning

Det arbeidet som skal refereres her hadde som mål å konstruere en tjenlig datastruktur for konstruksjon av ordbøker for naturlig språk på små datamaskiner. Arbeidet ble gjort for noen år tilbake av George M. Gillow og undertegnede, og vi anvendte en PDP 11/20. I dag kan derfor våre resultater ha interesse i forbindelse med mikromaskinanvendelser på samme område.

1.1 Designkrav

- Systemet skal hurtig og enkelt kunne laste en ordbok. Hurtig skal her forsås slik at tid for lasting, t_l, i størst mulig grad burde være en linjær funksjon av grunnlagstekstens størrelse, dvs. antall tokens, N_{token}.
- Ingen preprosessering av grunnlagsteksten skal være nødvendig (f.eks. sortering).
- Egenskaper ved naturlige språk skal i størst mulig grad formulere resterende krav. (Dette er gjort i tabellen på neste side).

En datastruktur for ordbøker for naturlige språk
Geir Berge
Proceedings of NODALIDA 1979, pages 47-64

EGENSKAP	→	DESIGNKRAV
Stor variasjon i ordlengde		Variabel lengde record format
En liten del av typene utgjør en stor del av teksten		Systemet skal ta spesielt hensyn til høyfrekvens-typer

⇓

(Hovedparten av høyfrekvenstypene vil dekkes av den første delen av tekstgrunnlaget.)

og

(Å øke tekstgrunnlaget for en ordbok vil i hovedsak gi opphav til lavfrekvens-typer)

Typer vil svært sjelden bli strøket fra ordboken	"Garbage collection" trenger ikke være inkludert i systemet

Behovet for oppslag av enkeltord vil være nær proposjonalt med de enkelte typers forekomstfrekvens i tekstbasen, og vi vil gjerne at systemet gir priviligert, hurtig aksess til slike typer.

Til slutt skal nevnes at systemet også bør gi gode muligheter for listing av større eller mindre deler av ordboken i alfabetisk orden.

Vi fant ikke at noen standard filstrukturer med tilhørende aksessmetoder oppfylte våre designkrav, og vi konstruerte derfor systemet som blir beskrevet i det følgende.

2. Løsningsforslag

Forslaget er implementert med types og tokens i form av grafiske ord, men det er intet til hinder for at det kan fungere på andre nivå. Videre konstruerte vi et enkelt-aksess system, men alle programdeler er utviklet med enkel utvidelse til multiaksess system for øye.

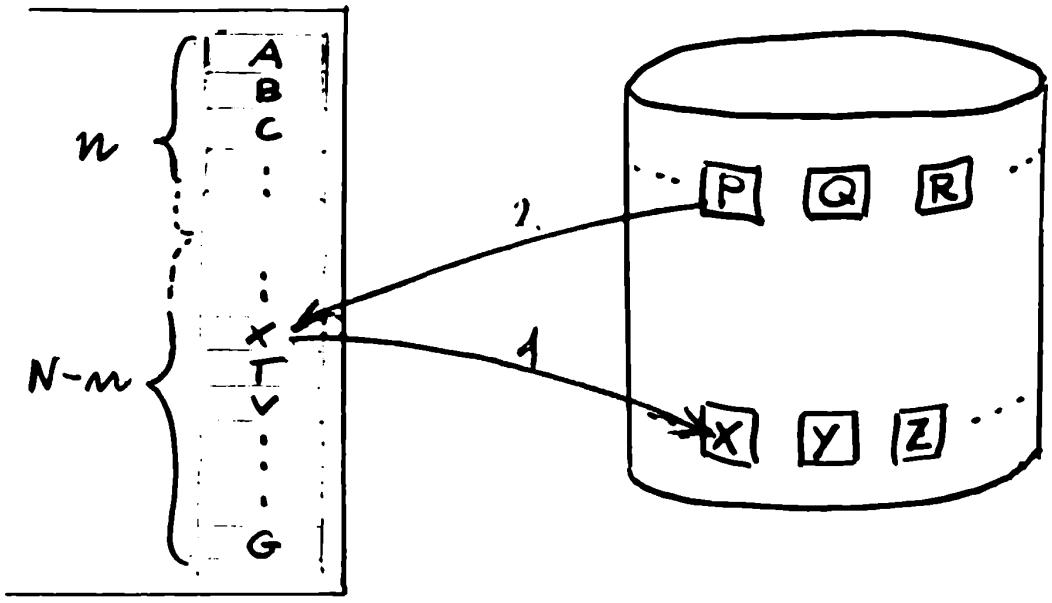
2.1 Paging systemet

Hele ordboken tenkes delt i et antall sider (pages). I hurtiglageret avsettes et område med plass til N slike. Hvert delområde med plass til en side kaller vi en nisje (slot). n av de N (n < N) nisjene inneholder residente sider, de resterende N-n sider som kan rulles ut på ytre lager (platelager) for å bli erstattet av andre. Dette skjer når systemet vil aksessere en side som ikke er i noen av de N nisjene i hurtiglageret. Se fig. 1.

For å avgjøre hvilken side som skal rulles ut fra hurtig-lageret, er det til hver side knyttet en "historikkvariabel" hvis verdi avspeiler sidens bruksfrekvens. Den av de N-n sidene som har lavest verdi for denne variabelen rulles ut. På denne måten oppnås at de sidene som oftest refereres har minst sannsynlighet for å bli rullet ut.

Av og til er det ønskelig å "låse" en side temporært fra å bli rullet ut etter regelen over, og dette kan gjøres fra systemprogrammet.

Det kan også nevnes at bare dersom en side er forandret fra den ble hentet inn til hurtiglageret blir den virkelig overført til ytre lager ved utrulling, en detalj som øker systemets effektivitet.



P forespørres og rulles inn,
X ut.
X er den side blant de N-n som har lavest verdi for 'historikk'.

Fig. 1

2.2 Generelt om systemet

Recordformat for elementene i ordboken fremgår i fig. 2. De enkelte betegnelsene

SCT: Neste elements seksjons-(side) nr.
DISPL: Adresse innen side for ditto.
SHC: Peker til neste logiske element innen denne side, hvis satt:
$\sqrt{}$: Frekvens for dette element (type)
LGTH: Lengde av datafeltet.

Ordboken har altså form av en listestruktur, og denne bygges i sin helhet under lasteprosessen. For hver ny token gjennomsøkes listen for å avgjøre om det er en ny type eller ikke. I førstnevnte tilfelle settes denne inn i listen, ellers økes bare frekvens for angjeldende type med 1. Se fig. 3.

Av fig. 3 fremgår det hvordan SHC'ene kan øke hastigheten i søkeprosessen vesentlig; v.h.a. disse går vi aldri mer enn en gang gjennom hver side.

Record Format

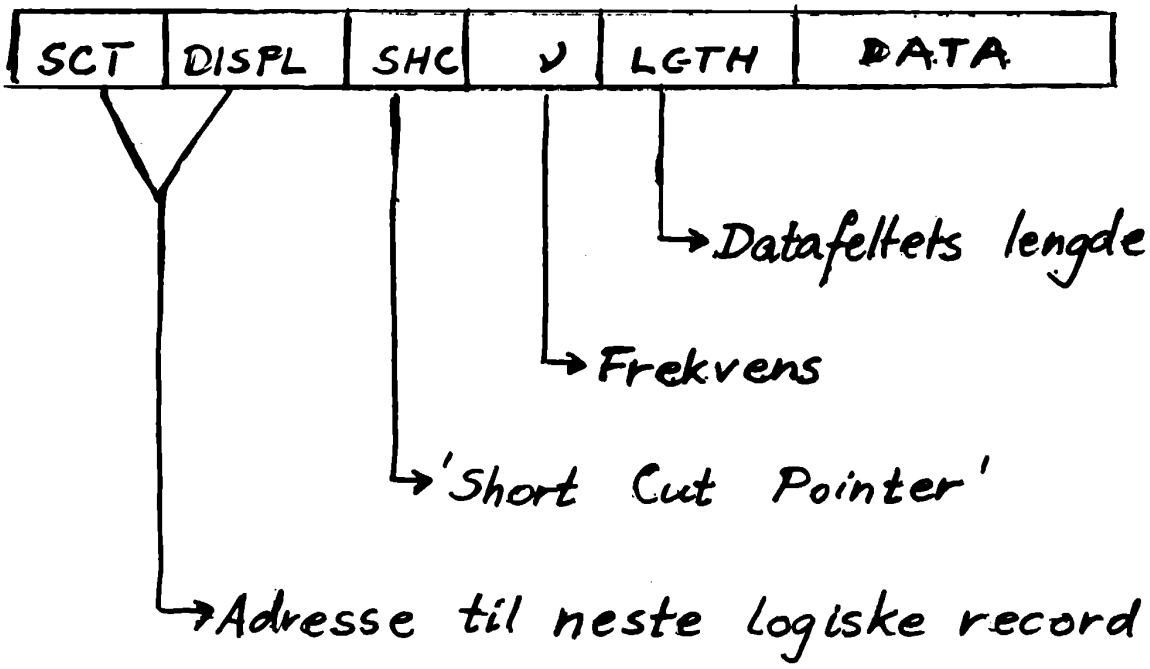


Fig. 2

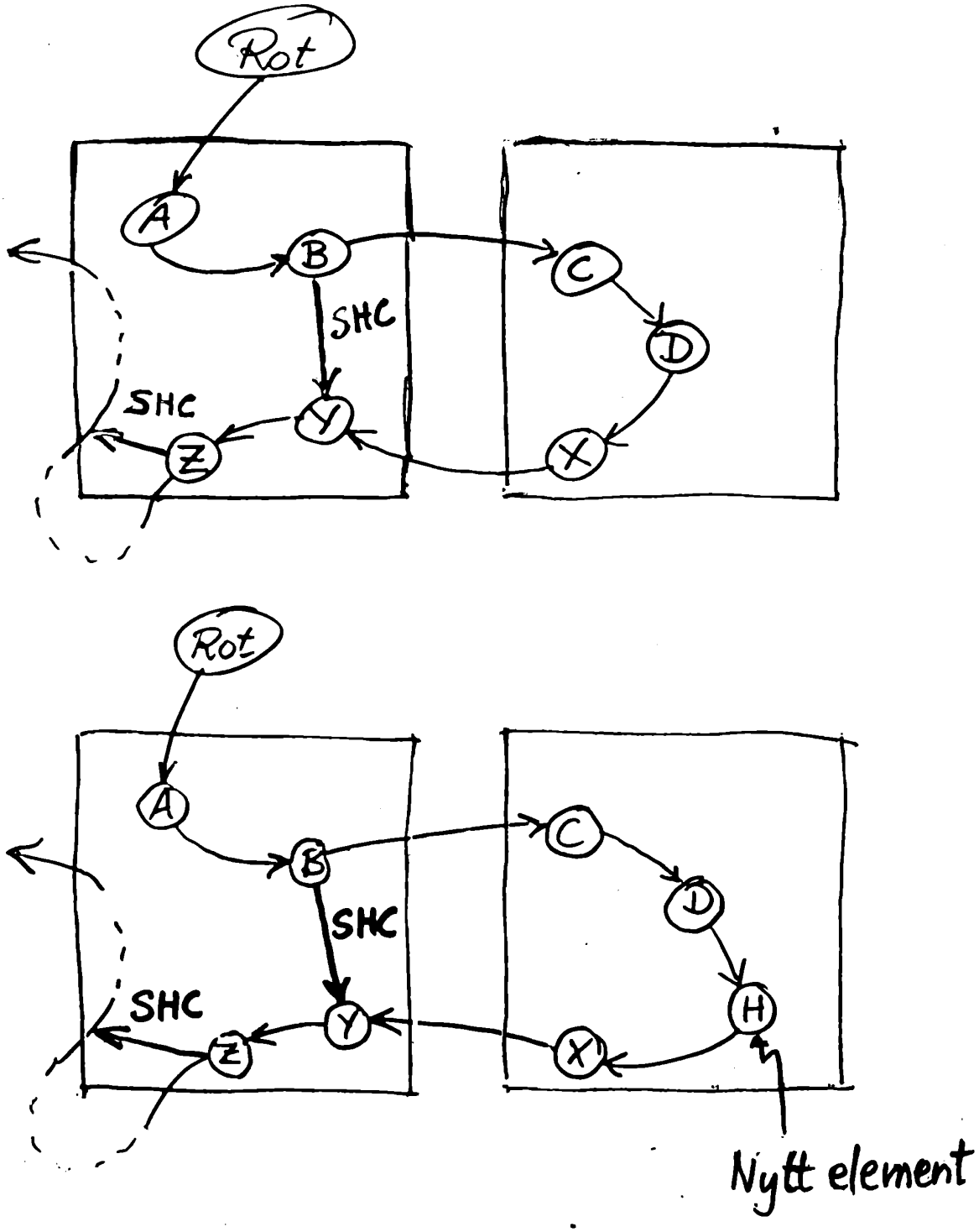


Fig. 3. Listebygging

2.3 Laste-strategi

Sidene kan under lasteprosessen være i ulike "tilstander":

- Nyeste side er den som sist er tatt i bruk.
- Backup siden inneholder logisk forgjenger til det elementet vi prøver å plassere.
- Nåværende side inneholder logisk etterfølger til samme.

Fig. 4 anskueliggjør dette.

Strategi for oppfylling av sidene, dvs. for plassering av nye typer var som vist under fig. 4.

For å gi en indikasjon på hvordan dette virker, henviser vi til fig. 5. Reglene anvendt for de nye elementene vil her være:

Regel 1 for C,
" 2 " D,
" 3 " M,
" 4 " B,

Hensikten med lastegrensen (se preferanselisten) som bare anvendes når en side er i tilstand "nyeste", er å øke sjansene for at regel 1 kan anvendes og derved for lange, ubrutte tråder innen en side. Dette medfører mindre inn-ut rulling og mindre overhead forbundet med short-cut pekerne.

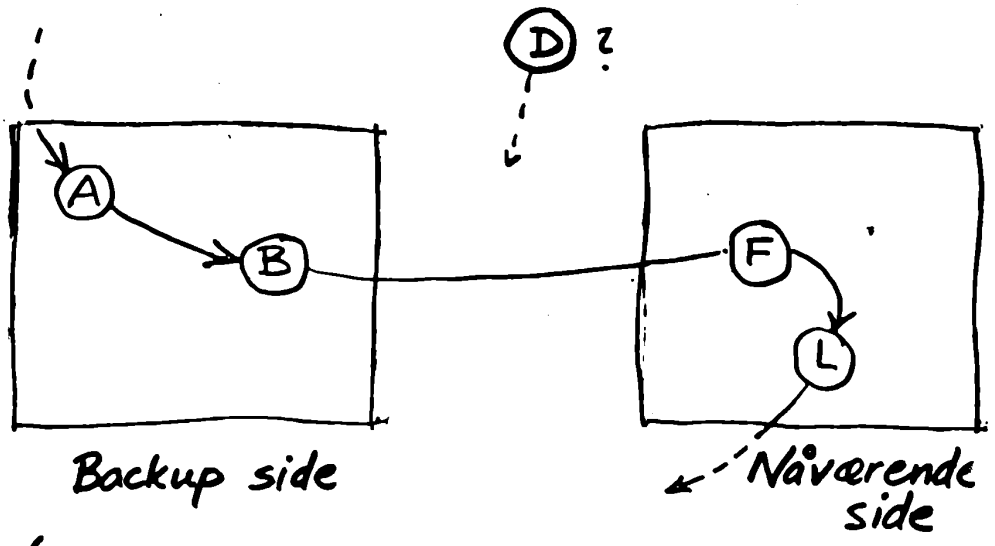


Fig. 4. Sidetilstander relativt (D).

Preferanseliste for plassering av element:

1. Backup, hvis plass
2. Nåværende, hvis plass
3. Nyeste, hvis "lastegrensen" ikke er nådd
4. Ta i bruk en ny nyeste

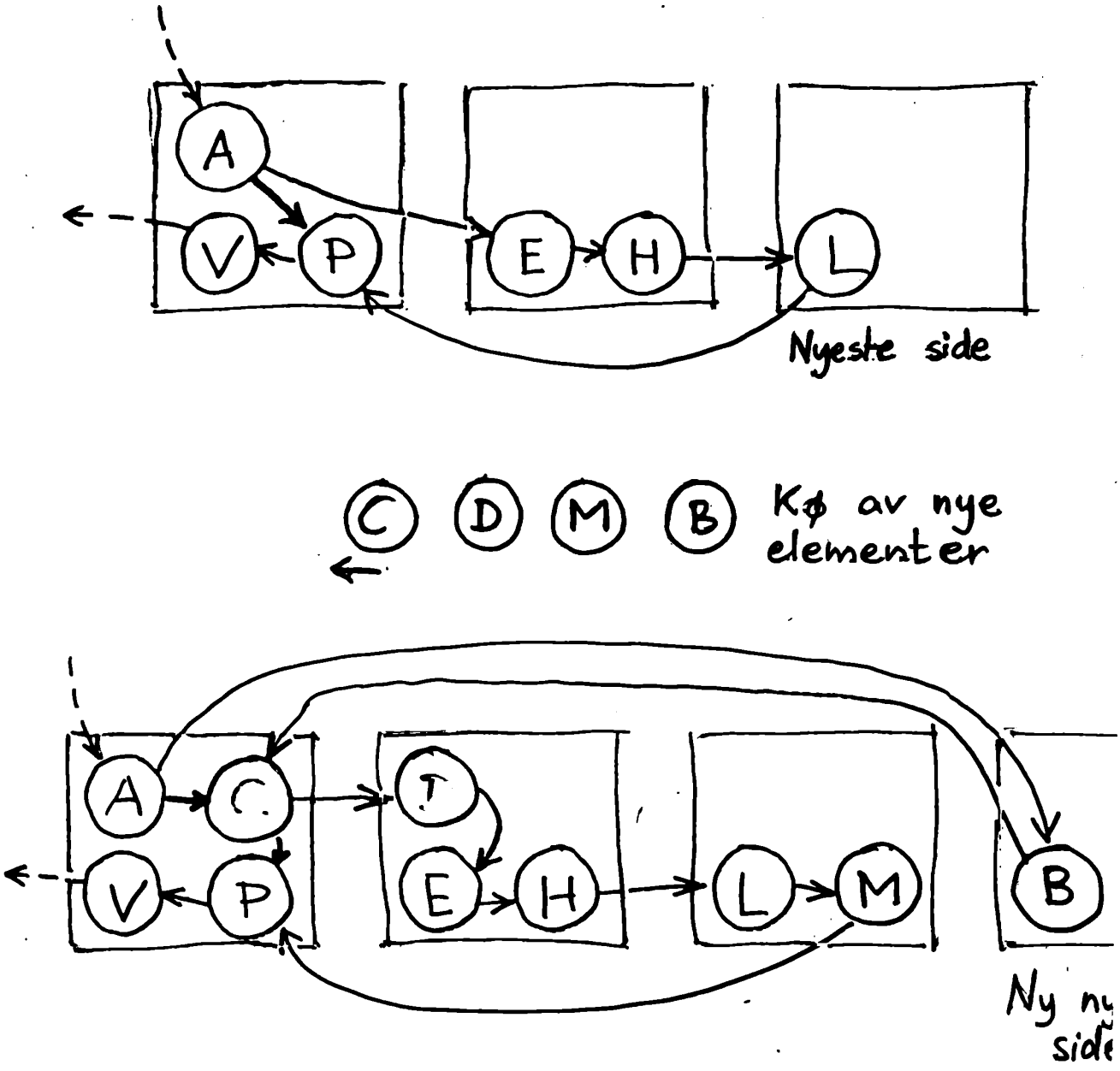


Fig. 5. Innsetting av nye elementer.
(Lastegrense = 0.5)

Dette vil imidlertid komme i konflikt med vårt ønske om å plassere flest mulig høyfrekvenstyper i de første sidene (spesielt i de residente), både for å få rask aksess til disse typene, og fordi de første sidene i stor grad bør virke som en indeks for resten av ordboken. For å oppnå det siste bør disse sidene fylles med så mange element som mulig spredd best mulig alfabetisk.

Vi valgte derfor å la fyllingsgraden variere som funksjon av sidenummer som antydet i fig. 6.

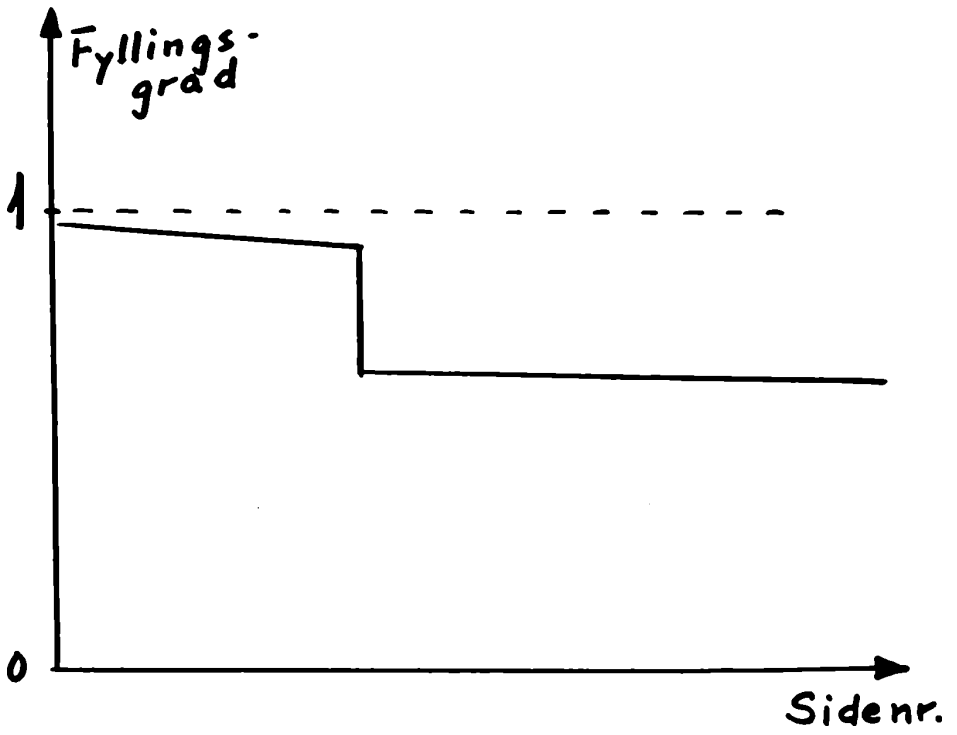


Fig. 6. Fyllingsgrad som funksjon av sidenummer

3. Eksperimenter og resultater

Som tekstbase ble Garman & Worse av A. Kielland benyttet. Denne inneholder omtrent 65.000 grafiske ord og gir opphav til en ordbok med ca. 10.500 ord.

Vi fant at vår teknikk virkelig ga mange høyfrekvensord i de første sidene, og at teknikken med lastegrense og plassering av typer i henhold til preferanselisten førte til lange uavbrudte tråder i de senere sidene.

En del forsøk ble gjort for å finne et "godt" parametersett for systemet. Her skal bare effekten av partiell fylling av nyeste side påpekes. Allerede med fyllingsgrad litt under en fikk vi resultater som vist i fig. 7, og svært lave fyllingsgrader ga ikke ytterligere forbedringer.

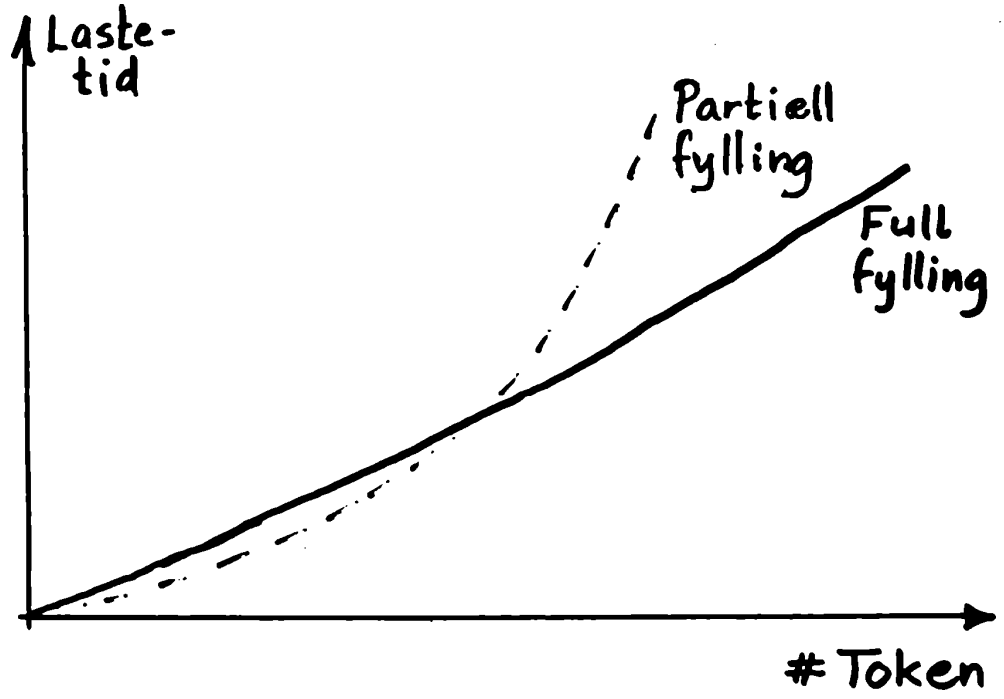


Fig. 7. Effekt av partiell fylling av "nyeste side"

Som vi ser av fig. 8 er vårt krav om en lastetid som er nær proposjonal med antall elementer i tekstbasen godt oppfylt. Fig. 9 viser en del andre data fra lasteprosessen, og vi ser at:

- TID/TOKEN vokser relativt lite og fremfor alt jevnt ettersom ordboken øker i omfang.
- PR/TOKEN som er sidereferanser pr. token forklarer mye av systemet "pene" adferd idet denne bare vokser fra 2,66 ved $M_{token} = 2 \cdot 10^4$ til 3.0 ved $M_{token} = 6{,}5 \cdot 10^4$ dvs. ved 11 %, mens ordboken i samme intervall øker fra ca. $4{,}1 \cdot 10^3$ types til ca. $10.5 \cdot 10^3$ types, altså med 156 %.
- Hvis vi ser på oppslagstid/token med full ordbok, var denne ca. 0.05 s., noe vi var vel tilfreds med, utstyret vi brukte tatt i betraktning. Brukes ordboken bare for oppslag, reduseres denne tiden vesentlig fordi vi slipper alle "skriveoperasjoner" knyttet til plassering av nye types. Senere tester indikerte et fall på omkring 60 % som bringer oppslagstiden ned på 0,02 s.

Alt i alt mener vi disse resultatene er oppmuntrende, og at en slik eller lignende design egner seg godt for konstruksjon og bruk av ordbøker på mindre datamaskiner som ikke tilbyr gode databasesystem eller tilsvarende avansert software.

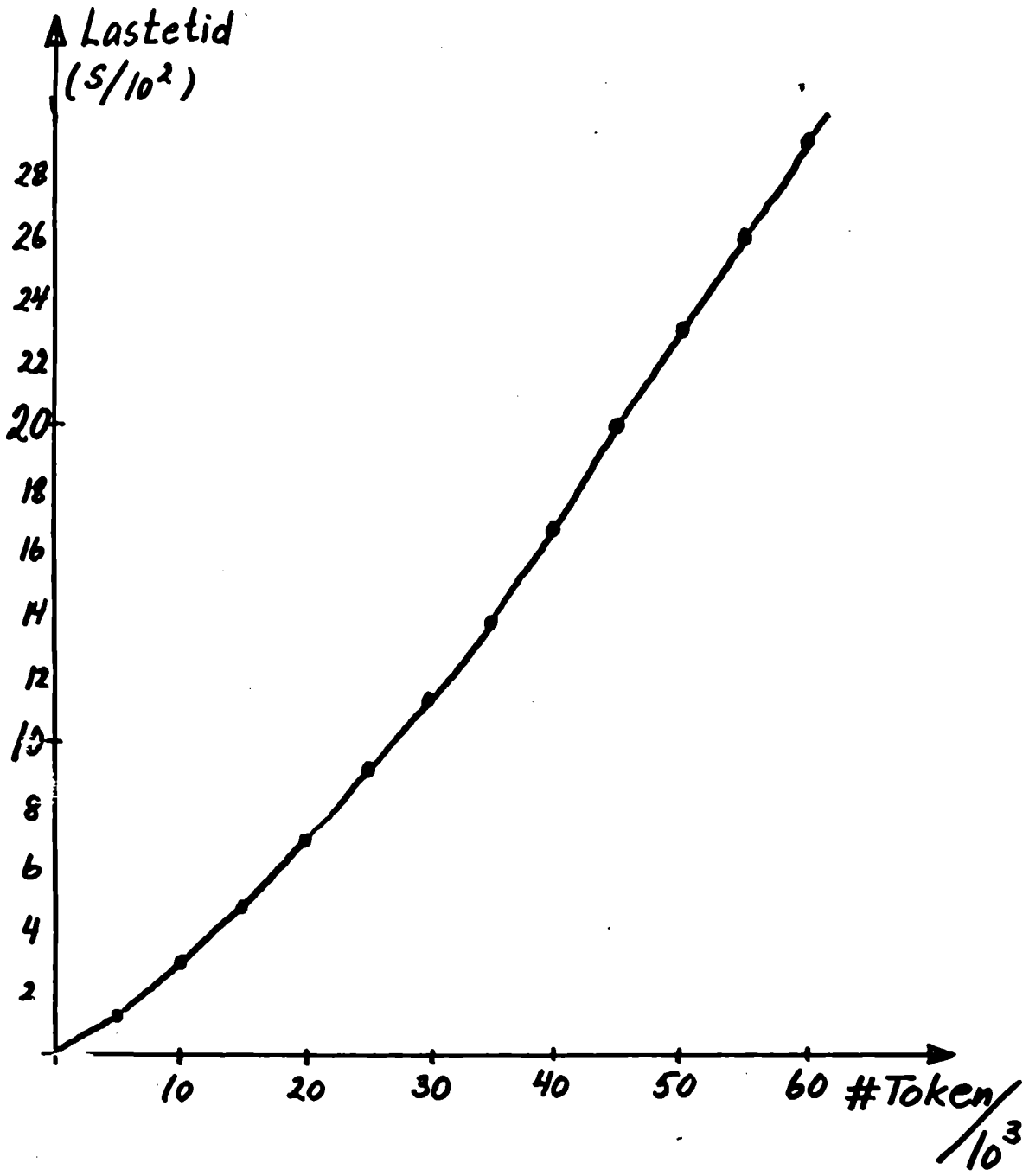


Fig. 8. Lastetid som funksjon av # token

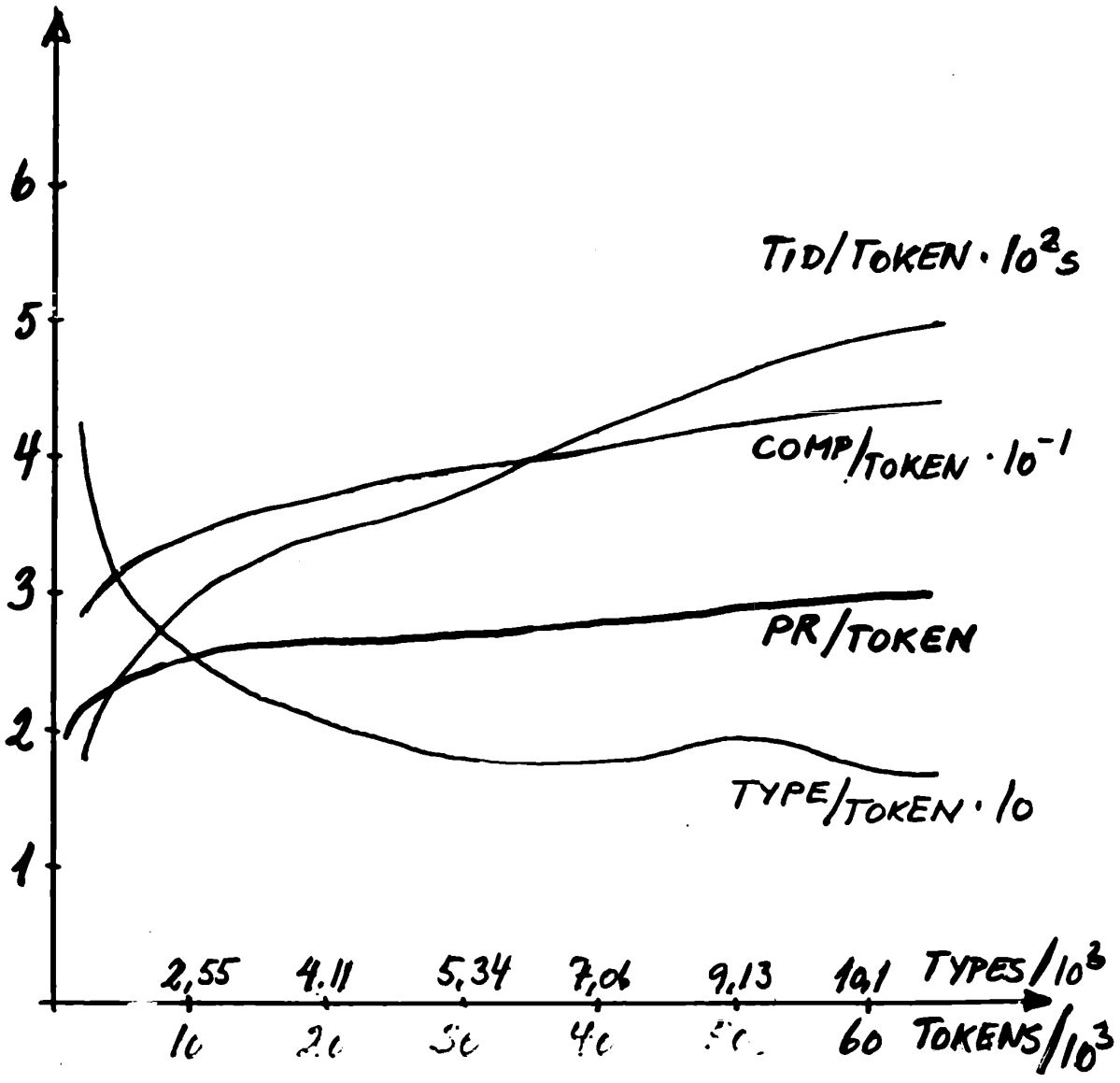


Fig. 9. Statistikk fra lasting av ordbok.