

IDATT2101 ALG DAT

Kompleksitet

Asymptotisk notasjon

$4n^2 + 3n + 1 \approx n^2$ ved $n \rightarrow \infty$

Denne brukes til grenser for kjøretid:

O – øvre, Ω – nedre, Θ – begge/lik.

Kompleksitet i praksis

$O(n)$, $O(n^2)$ osv. oppnås ofte med nestede løkker som teller $0 \rightarrow n$. Med *if*-break inni loop kan vi få ulik O og Ω .

Rekursjon

Krav til rekursjon

1. En basis som kan løses trivielt.
2. Problem som kan deles opp i mindre biter til man når basistilfellet.

Rekursiv kompleksitet

$$T(n) = \begin{cases} 1 & \text{når } n = 1 \\ aT\left(\frac{n}{b}\right) + cn^k & \text{når } n > 1 \end{cases}$$

$b^k \blacksquare a$	$T(n) \in \Theta(\blacksquare)$
$<$	$n^{\log_b a}$
$=$	$n^k * \log n$
$>$	n^k

a : antall rekursive kall, b : brøkdel av datasett brukt i ett kall, cn^k : kompleksitet.

NB: cn^k regnes ut med vanlig metode.

Rekursjon vs. Iterativ løsning

Hvis man enkelt kan løse problemet iterativt bør man det med tanke på overhead.

Sortering

Beste mulige kompleksitet er $O(n)$ siden alle elementer må sjekkes minst en gang.

Innsettingssortering – $O(n^2)$, $\Omega(n)$

Start med ett kort på hånda, og sett inn ett og ett kort på riktig sted.

+ Små og allerede sorterte datasett.

– Dårlig på store datasett.

Boblesortering – $O(n^2)$

Gå gjennom tabellen $n - 1$ ganger og bytt plass på naboer i feil rekkefølge. Store tall synker, små tall bobler opp.

+ Veldig enkel å implementere.

– Lite brukt: innsetting er like enkel/bedre.

Velgesortering – $O(n^2)$

Velger det største tallet og setter dette på riktig plass. Gjenta til man er ferdig.

+ Bytter plass på færre tall enn boble.

Kvadratiske sorteringsalgoritmer

Alle sorteringsalgoritmer som bytter naboer med hverandre, får kompleksitet $\Omega(n^2)$.

Shellsort – $O(n^2)$

Forbedring av innsetting. Starter med avstand $\frac{n}{2}$ og går ned til 1 (nabo).

+ Merkbart bedre enn innsetting.

– Fortsatt $O(n^2)$ som kompleksitet.

Flettesortering – $O(n * \log n)$

Deler tabellen i to rekursivt og fletter løsningene sammen til en.

+ Bedre enn kvadratiske algoritmer.

– Krever en ekstra tabell (plass).

Quicksort – $O(n \log n)$ i snitt pga. pivot

Deler data i to med en pivot-verdi. Alle tall større/mindre på hver sin side. Rekursiv.

+ Raskest kjente for generelle data.

– Små datasett. Dårlig plassert pivot.

Dual Pivot Quicksort

Forbedring av quicksort som er rundt 10-20% raskere. Deler data i tre, i stedet for to, og oppnår færre sammenlikninger og rekursive kall.

Tellesortering – $\Theta(n + k)$

Teller opp forekomster og skriver riktig antall i stigende rekkefølge til en ny tabell.

+ Veldig effektiv når tallene ligger tett.

– Krever heltall i $0 \dots k$. Ekstra hjelpetabell.

Lister, kø og stack

Enkeltlenket liste

Noder med verdi og referanse til neste.

Eksempel: $A: 1 \rightarrow B: 7 \rightarrow C: 2 \rightarrow D: 4$

Dobbeltlenket liste

Samme som 1, men og referanse til forrige.

Eksempel: $A: 1 \leftrightarrow B: 7 \leftrightarrow C: 2 \leftrightarrow D: 4$

Kø (bruker tabell/enkeltlenket liste)

Uttak alltid foran, innsetting alltid bak.

Innsetting: $O(1)$. **Uttak:** $O_{\text{tabell}}(n)$, $O_{\text{liste}}(1)$

Stack (bruker tabell/enkeltlenket liste)

Uttak alltid foran, innsetting alltid foran.

Innsetting: $O(1)$. **Uttak:** $O(1)$.

Trær

En graf som har ingen sykler. Ikke-lineær datastruktur.

Rot: Noden som ikke har en forelder.

Løv-node: Noder som ikke har noen barn.

Grad: Antall barn en node har.

Dybde: Antall kanter fra noden til rota.

Høyde: Antall kanter fra laveste løv-node.

Binærtrær

Et tre som har maks to barn per forelder.

Perfekt binærtre: Alle noder har to barn, og løvnodene ligger på samme nivå.

Komplett binærtre: Perfekt, men kan mangle løvnoder nederst til høyre.

Fullt binærtre: Alle noder har 0/2 barn.

Innsetting: Sett inn som rot hvis en ikke eksisterer, ellers sammenlikn nøkkelverdier.

Ny verdi er: mindre \rightarrow venstre, større \rightarrow høyre.

Søking: Samme søk som innsetting.

Sletting: Har noden ett barn flyttes denne opp. Har noden to barn erstatter vi noden med den minste noden i høyre subtre. Har noden ingen barn kan vi slette den fritt.

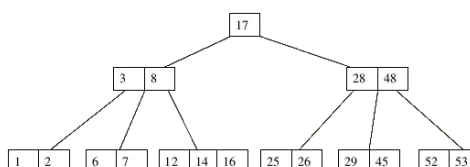
Kompleksitet: Alle handlingene $O(h)$.

B-trær

Datastruktur for effektiv bruk av disker. Mye brukt i databasesystemer. Et tre der man kan ha flere verdier i hver node.

Krav til b-tre av orden m : Noder har maks m barn, alle indre noder har minst $\lceil \frac{m}{2} \rceil$ barn. En

indre node med k barn har minst $k - 1$ nøkler. Alle løvnoder er i samme dybde.



Heap og prioritetskø

Et komplett binærtre hvor hver node inneholder en nøkkelverdi. Delvis ordning gjør heap veldig rask til å hente max/min.

Heapegenskapen

I en max/min-heap har alle noder nøkkelverdi større/mindre eller lik begge barnenodenes. Rota er størst/minst.

Prioritetskø

Datastruktur der man plukker ut basert på et elements prioritet. Ikke når det ble til. Veldig raskt å plukke ut viktigste element.

Heap som tabell

$$i_{\text{foreldrenode}} = \left\lfloor \frac{i_{\text{node}} - 1}{2} \right\rfloor$$

$$i_{\text{venstre barn}} = 2i_{\text{node}} + 1$$

$$i_{\text{høyre barn}} = 2i_{\text{node}} + 2$$

Heapsort – $O(n \log n)$, $\Omega(n)$

Lager en heap av usortert data og plukker ut det største/minste elementet. Fortsetter til alle elementene er plukket ut.

+ Like god som flettesort uten ekstra tabell.

Hashtabeller

Datastruktur hvor man kan bruke tekst, objekter eller uegnede tall som nøkkel.

Lastfaktor: Andel plass brukt i tabell. $\alpha = \frac{n}{m}$

m : Størrelse på tabell, n : Antall elementer.

Dårlig lastfaktor: $\alpha \rightarrow 0$, Full tabell: $\alpha \rightarrow 1$.

Hashfunksjon

En funksjon som tar ønsket nøkkel inn, og gir verdier i $0..m$ ut. Jevnt fordelt.

Beregningen må skje i $O(1)$ tid.

Hash med restdivisjon: $h(k) = k \bmod m$

Multiplikasjon: $h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$

Kollisjoner

To poster med ulik nøkkel kan få samme hash. Tabellposisjoner kan lagre ett element, så kollisjonen må håndteres.

Kollisjonshåndtering

Lenkede lister: Tabellelementene er hoder på lenkede lister som kan holde elementer.

Åpen adressering: Hvis ønsket plass er tatt, prob etter andre posisjoner. Tre måter:

Lineær: Gå 1 bort fram til ledig.

+ Neste ledige plass er enkel.

– Lange kjeder med kollisjoner.

Kvadratisk: Gå step^2 bort til ledig.

+ Naboverdier får ikke samme kjede.

– Like verdier har alltid samme kjede.

Dobbel hashing: Steglengde er avhengig av en andre hashfunksjon.

+ Like verdier ikke alltid samme kjede.

– Krever en ekstra hashfunksjon.

Uvektede grafer

Spor: En vei som ikke repeterer kanter.

Sti: En vei som ikke repeterer hjørner/noder.

Selvløkke: Kant som fra/til samme node.

Rundtur: Vei med start/slutt i samme node.

Enkel vei: Vei med kun unike noder.

Implementasjon

[illegible]