Oppgave 1

La n være antall elementer i tabellen. Symboltabellen lagrer informasjon om deklarerte navn/symboler, inkludert informasjon om skop. Informasjon om skop behøves ved sletting av symboler. Anta at m er antall elementer som slettes i et gitt skop. Under skisseres noen mulige datastrukturer.

Array Vi setter inn symboler bakerst i et array og markerer når et nytt skop begynner (f.eks. ved at vi lagrer et `begin` når nytt skop entres).

Lenket liste Samme struktur som array. Vi setter inn først i lista.

B–tre Alle symboler ligger i et felles B–tre. Siden vi må holde orden på skopene, legger vi hvert objekt også inn i en liste som lenker sammen alle med samme skop. Når vi sletter elementene fra treet, kan vi beholde dem i de lenkede listene dersom vi har bruk for dem andre steder (som er tilfelle i en kompilator). Merk at hvert element vil settes inn i to strukturer: både et B–tre og en liste. Elementene må derfor ha pekere som understøtter begge disse strukturene.

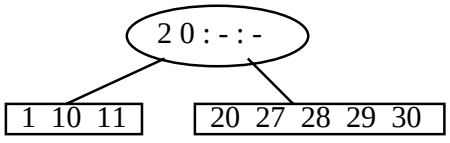
Hashtabell Vi kan bruke "separate chaining" kap. 5.3, med krav om listen som samler opp alle som hasher til samme verdi er sirkulær. I tillegg lenker vi objektene i en skop–liste som for B–trær. Strukturen er en variant av strukturen i fig. 3.25 s. 68 i boka. Med referanse til den figuren vil de horisontale listene lenke sammen elementer som hasher til samme verdi, mens de vertikale listene kjeder sammen symboler med samme skop.

	oppslag	innsetting	sletting
Array	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(m)$
Lenket liste	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(m)$
B–tre	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(m \log n)$
Hashtabell	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(m)$

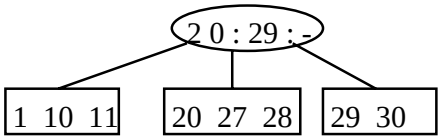
Angivelsene er grove, noen faktorer er ignorert fordi de vil være neglisjerbare.

Oppgave 2

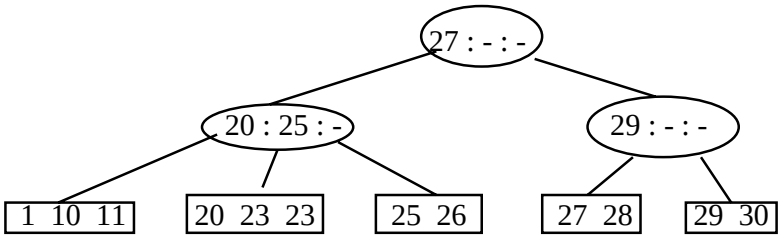
Treet under viser situasjonen når vi forsøker å sette inn 27:



Strategien i oppgaven tilsier at vi må splitte i to noder, som gir følgende tre:

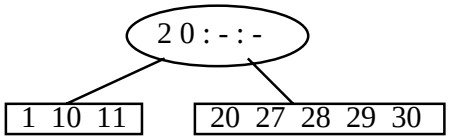


Det endelige treet blir:

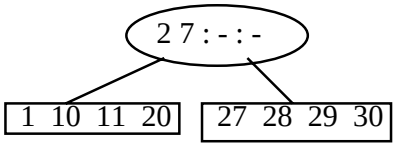


Oppgave 3

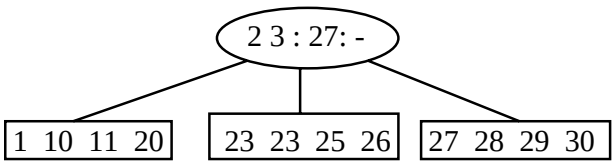
Strategien i MAW er: Hvis en node ikke har plass til et nytt element, undersøker vi om en av søsknene har plass. Vi splitter noder kun hvis alle søsknene er fulle. La oss igjen ta for oss situasjonene når vi forsøker å sette inn 27:



Vi avgir et element til den venstre nabonoden og oppdaterer foreldrenoden:

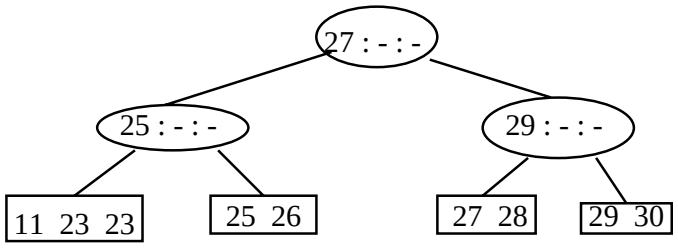


Det endelige treet blir:



Observer at ved et stort tre kan søsken på samme nivå ligge langt unna (i en helt annen gren av treet). Et kompromiss kan da være å bare undersøke om venstre og høyre nabonode ("nabosøsken") har plass.

Oppgave 4



Oppgave 5

MERK: Det som her gjøres er grove beregninger ut fra gjennomsnittstall. Selv om gjennomsnittet er slik som antydet KAN det f.eks. være forgreninger i treet som er spesielt tynt befolket, og som gjør at treet som helhet kunne få et nivå til. En mer ordentlig analyse ville være å gjøre beregningene over med minimalt og maksimalt antall verdier i alle nodene, og det litt overraskende (??) er da at høyden av treet sjelden vil variere mer enn EN, eller kanskje to. Derfor er det også rimelig nok å gjøre en slik gjennomsnittsbetrakning som vi gjør her.

```
Treet må har pekere til 1.000.000 data-recorder
Nivå 0:   1 node               (roten, ca. 100 barn)
Nivå 1:   1*100 = 100 noder    (hver node har ca. 100 barn)
Nivå 2:   100*100=10.000 noder (hver node har ca. 100 pekere til data-recorder)

Plass til data-recorder:
  Har 1.000.000 recorder som hver bruker 100 byte
  Totalt: 1.000.000*100 byte = 10^8 byte

Plass til treet:
  Tilsammen har vi 1 + 100 + 10.000 = 10.101 (ca. 10^4) noder
  Hver node har 2*128 pekere (hver på 4 byte) pluss 10 byte ekstra
  D.v.s. 2*128*4+10 = 1034 (ca 10^3) byte
  Totalt: 10^4*10^3=10^7 byte
```

Treet tar altså ca. 10% av plassen til data-recordene. Dermed kan vi gjerne ha flere B-trær ("indekser") over hver recordmengde (på forskjellige nøkler) uten at det betyr mye for plassen.

Oppgave 6

Vi går gjennom det opprinnelige B-treet i "infiks" rekkefølge, og kommer dermed til bladnodene i sortert rekkefølge. Vi tar verdiene herfra og stapper dem tett inn i nye bladnoder, og kan så kaste det gamle B-treet. De nye bladnodene må holdes i en sortert liste, og vi går så gjennom denne og bytter ut M og M bladnoder med en indre node (som får de M bladnodene til barn). Dette gjentas til listen har en node, som er roten.

Merk at vi her kan få noder helt til "høyre" i hvert nivå som ikke er forskriftsmessig fulle, men antall noder på hvert nivå er så lite som mulig ut fra den gitte M.