

IN2010 halvveis recap

Jakob Hansen

6. oktober 2020

Hva vi skal snakke om i dag

- ▶ Recap av pensum til nå! :)

Uke 35

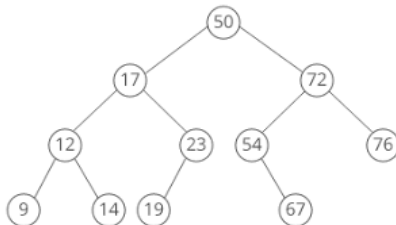
- ▶ Big O
 - ▶ Analyserer kjøretid for et program.
 - ▶ Abstraherer bort småforskjeller, ser på den generelle veksten.
 - ▶ Gjør det lettere å snakke om kjøretid og sammenlikne programmer
- ▶ Binærsøk
 - ▶ Algoritme som finner indeksen til et element i et sortert array veldig raskt.
 - ▶ Se på midten, hvis elementet man ser etter er mindre, gå til venstre halvdel, hvis det er større, gå til høyre halvdel. Repeat.
 - ▶ $O(\log(n))$



Uke 35

▶ Trær

- ▶ Lagrer verdier med en key som plasserer mindre keys til venstre, større til høyre
- ▶ Innsetting: Reis nedover treet til du finner en nullpeker. Reis til venstre dersom key er mindre, høyre hvis større.
- ▶ Sletting: 3 caser: 0-2 barn.
 - ▶ 0 barn: bare fjern pekeren fra forelder
 - ▶ 1 barn: Erstatt noden med barnet
 - ▶ 2 barn: Reduser til case 0/1 ved å erstatte noden med neste node i inorder traversal.



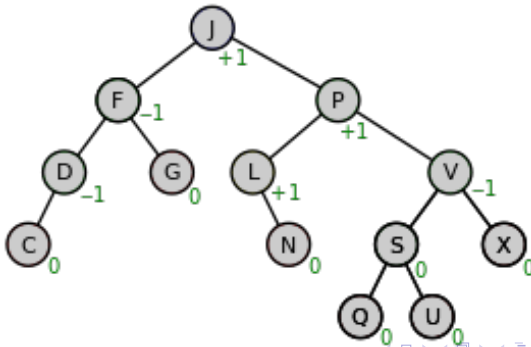
Uke 36

- ▶ Balanserte søketrær
 - ▶ Balanserer seg selv, for å få en garantert kortere høyde
 - ▶ Da kan vi gjøre innsetting, sletting osv i $O(\log(n))$ tid worst case
 - ▶ Retter opp i treet gjennom rotasjoner
- ▶ AVL trær
- ▶ Rødsvarte trær

Uke 36

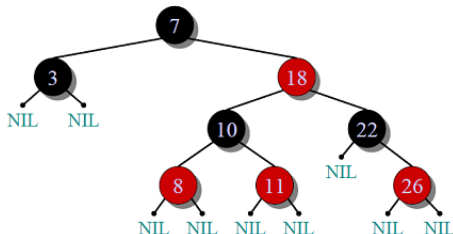
► AVL trær

- Høyde: $\max(\text{høyre.høyde}, \text{venstre.høyde}) + 1$
- Balanse: $\text{høyre.høyde} - \text{venstre.høyde}$
- Balanseverdien b må være $-1 \leq b \leq 1$
- Innsetting: Sett inn som vanlig, oppdater høyde og balanse på tilbakeveien i rekursjonsstacken. Hvis balanseverdien ikke er riktig, gjør rotasjoner i retningen av ubalansen for å fikse det!
- Sletting: Slett som vanlig. Gjør det samme som innsetting på tilbakeveien!



Uke 36

- ▶ Rødsvarte trær
 - ▶ Ganske abstrakt implementasjon
 - ▶ Regler:
 - ▶ Alle noder er røde eller svarte
 - ▶ Hvis en node er rød, så er barna svarte
 - ▶ Det må være like mange svarte noder på stien fra rot til alle nullpekere
 - ▶ Ekstra: nullpekere er sorte, roten er sort
- ▶ Rødsvarte trær gjør færre rotasjoner enn AVL, men er litt mindre balansert

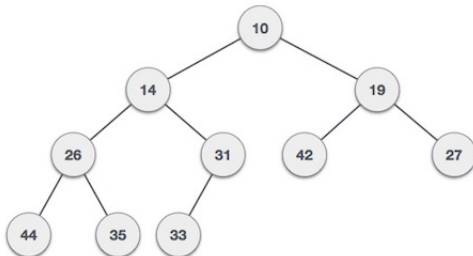


Uke 37

- ▶ Prioritetskø
 - ▶ abstrakt datatype
 - ▶ Har metoder som removeMin, insert, size, osv...

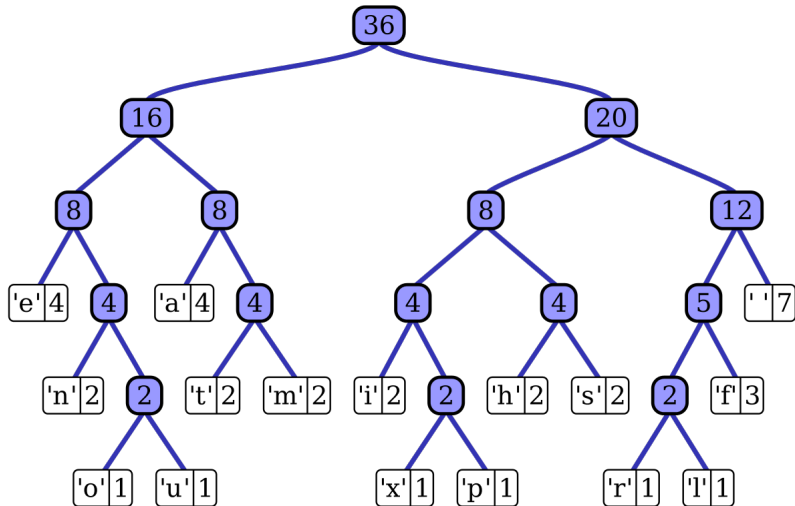
Uke 37

- ▶ Heap (binær heap)
 - ▶ Implementasjon av prioritetskø
 - ▶ MinHeap -> Barna til en node må ha større verdi
 - ▶ MaxHeap -> Barna til en node må ha mindre verdi
 - ▶ Da er alltid det minste/største elementet øverst
 - ▶ Innsetting: Sett inn nederst til venstre, boble opp om nødvendig.
 - ▶ Sletting: Erstatt roten med elementet nederst til venstre, boble ned om nødvendig.
 - ▶ Kompleksitet på innsetting og sletting: $O(\log(n))$



- ▶ Huffman koding
 - ▶ Mål: Gjøre “encodingen” av en streng så kort som mulig + uniquely decodable
 - ▶ Løsning: La bokstavene som fremkommer oftest ha kort bit representasjon.
 - ▶ Implementasjon: Prioritetskø med alle bokstavene der prioriteten er antall forekomster. Bygg et binært tre fra bunnen og opp.
 - ▶ Encodingen til en bokstav: Venstre er 0, høyre er 1, reis nedover treet fra roten for å finne encodingen.
 - ▶ Uniquely decodable: Ingen encodings er en prefiks av en annen encoding.

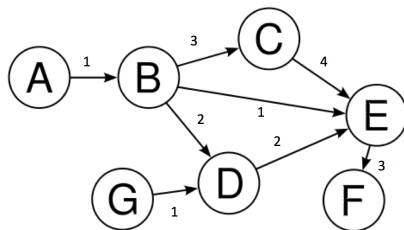
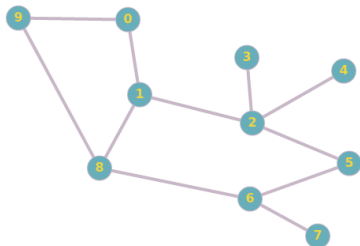
Uke 37



Uke 38

► Grafer

- Noder og kanter
- Rettet / urettet
- Vektet / uvektet
- Sammenhengende, komponenter



Uke 38

- ▶ Traversering
- ▶ DFS
 - ▶ Gå så dypt som mulig i en retning, så prøv en annen
 - ▶ Bruker en stack (ofte rekursjonsstacken)
- ▶ BFS
 - ▶ Ta alle noder 1 kant ut fra start, så 2 kanter, 3, 4, ...
 - ▶ Bruker en kø (FIFO)

- ▶ Korteste sti
 - ▶ Finne korteste sti (minimal vekt) mellom 2 noder (alternativt mellom alle noder)
 - ▶ Dijkstra, Bellman-Ford
- ▶ Minimale spenntreer
 - ▶ Minimalt spenntre = minimal sammenhengende subgraf med minimal sammenlagt kost.
 - ▶ Prims, Kruskals, Boruvka
- ▶ Baserer seg alle på en liknende ide: Legg noder/kanter i en prioritetskø, ta ut og gå over kanter. Får kompleksitet $O(|E| * \log(|V|))$ (Utenom Bellman-Ford)

- ▶ Dijkstra
 - ▶ Lag en prioritetskø der alle noder utenom start har prioritet ∞ , start har 0. Dette kalles “relax” verdien.
 - ▶ Loop: Ta ut minste element, se på alle kanter ut, hvis kantnoden er i køen, og relaxverdien til noden du tok ut + kosten av kanten er mindre enn relaxverdien til kanten, oppdater relaxverdien.
- ▶ Bellman-Ford
 - ▶ Prøv alle stier $|V| - 1$ kanter i lengde ut fra startnoden!
 - ▶ Ha en liknende prioritetskø som Dijkstra.
 - ▶ Gå over alle kanter i grafen $|V| - 1$ ganger, oppdater relax verdi.
- ▶ Bruk et predecessor array/instansvariabel for å finne den korteste stien

Uke 39

- ▶ Prims
 - ▶ Noe likt Dijkstra, start fra en node, bygg opp et tre ved å legge til den billigste kanten ut fra treet.
- ▶ Kruskals
 - ▶ Legg alltid til den billigste kanten!
 - ▶ Enkel tanke, verre å programmere
- ▶ Boruvka
 - ▶ Noe lik Kruskals, la hver node være en komponent.
 - ▶ Loop: Hver eneste komponent legger til en kant ut fra komponenten. Halverer antall komponenter i hver iterasjon.
 - ▶ Godt egnet for parallellisering, men har samme kompleksitet!