Balanserte søketrær

- AVL-trær (Adelson-Velskii og Landis, 1962)
- Splay-trær (Sleator og Tarjan, 1985)

AVL-trær

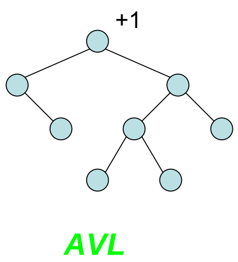
Et binært tre er et AVL-tre hvis følgende holder:

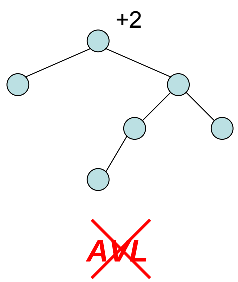
1. forskjellen i høyde mellom det høyre og det venstre deltreet er maksimalt 1, og
2. det høyre og det venstre deltreet også er AVL-trær.

AVL-trær

Et binært tre er et AVL-tre hvis følgende holder:

1. forskjellen i høyde mellom det høyre og det venstre deltreet er maksimalt 1, og
2. det høyre og det venstre deltreet også er AVL-trær.





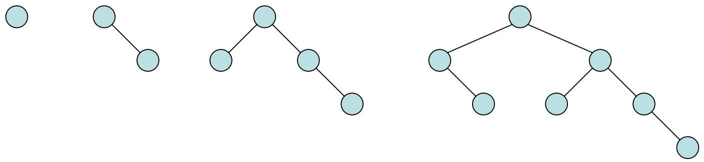
AVL-trær – høyde

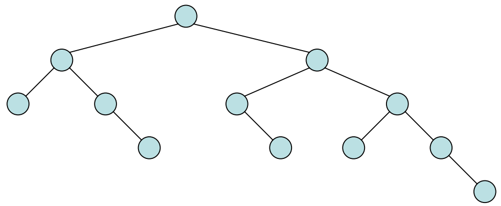
- Komplette binære trær har lav høyde ($O(\log_2 n)$), og de fleste operasjoner på trær har kjøretid proporsjonalt med høyden av treet.
- Siden høyre og venstre deltrær i et AVL-tre ikke kan avvike i høyde med mer enn 1, er AVL-trær opplagt relativt nær komplette.
- AVL-trær er derfor relativt effektive.

Men akkurat hvor effektive (dvs. høye) er AVL trær?

- Et komplett binært tre med n noder har $\log_2(n + 1)$ nivåer.
- AVL-trær med n noder har maksimalt $1.44 \cdot \log_2(n + 2)$ nivåer.

AVL-trær – høyde





worst case-trær

Beviset går mest elegant i gjennom ved å definere høyde som antall nivåer.

AVL-trær – høyde

Teorem. *Et AVL-tre på n noder har høyde* (*antall nivåer*) *maksimalt* $1.44042 \cdot \log_2(n + 2)$.

Bevis. La G_h være antall noder i et worst case-AVL-tre av høyde h,
$G_h = G_{h-1} + G_{h-2} + 1$, $\quad G_0 = 0, \quad G_1 = 1.$
Fibonacci-tallene er definert som
$F_h = F_{h-1} + F_{h-2}$, $\quad F_0 = 0, \quad F_1 = 1.$

h	0	1	2	3	4	5	6	7	7
F_h	0	1	1	2	3	5	8	13	…
G_h	0	1	2	4	7	12	20	33	…

Vi ser at $G_h = F_{h+2} - 1$.

$$F_h = \left[\frac{\phi^h}{\sqrt{5}}\right]_{\text{NINT}}$$, funksjonen $[x]_{\text{NINT}}$ er avrunding til nærmeste heltall.

$$\phi = \frac{(1+\sqrt{5})}{2}$$, ϕ er det såkalte gyldne snitt (1.61803…).

Bevis.
(forts)

For å fjerne avhengigheten av NINT-funksjonen, kan vi skrive

$$F_h = \left[\frac{\phi^h}{\sqrt{5}}\right]_{\text{NINT}} > \frac{\phi^h}{\sqrt{5}} - 1 \text{ , slik at } G_h > \frac{\phi^{h+2}}{\sqrt{5}} - 2 \text{ .}$$

Ta nå et vilkårlig AVL-tre. La n være antall noder og h høyden. Da har vi

$$n \geq G_h > \frac{\phi^{h+2}}{\sqrt{5}} - 2$$

$$n + 2 > \frac{\phi^{h+2}}{\sqrt{5}}$$

$$\log_\phi(n+2) > \log_\phi\left(\frac{\phi^{h+2}}{\sqrt{5}}\right)$$

$$\log_\phi(n+2) > h + 2 - \log_\phi\sqrt{5}$$

$$\frac{\log_2(n+2)}{\log_2\phi} > h + 2 - \log_\phi\sqrt{5}$$

$$h < 1.44042 \cdot \log_2(n+2) - 0.32772 \text{ .}$$

AVL-trær – høyde

- Høyden av et komplett binærtre gir en nedre grense for høyden h av et AVL-tre med n noder.
- Teoremet gir en øvre grense.

$$\log_2(n + 1) \leq h < 1.44042 \cdot \log_2(n + 2) - 0.32772$$

Definerer vi høyden som antall nivåer, er høyden av et komplett binærtre $\log_2(n + 1)$.

AVL-trær – operasjoner

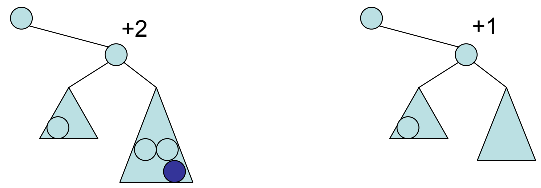
- Insert
- Delete
- Find
- …

} Kan føre til at balansekravet brytes – rebalansering kreves.

AVL-trær – operasjoner

- Insert
- Delete
- Find
- …

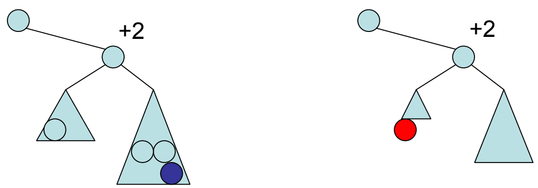
} Kan føre til at balansekravet brytes – rebalansering kreves.



AVL-trær – operasjoner

- Insert
- Delete
- Find
- …

} Kan føre til at balansekravet brytes – rebalansering kreves.



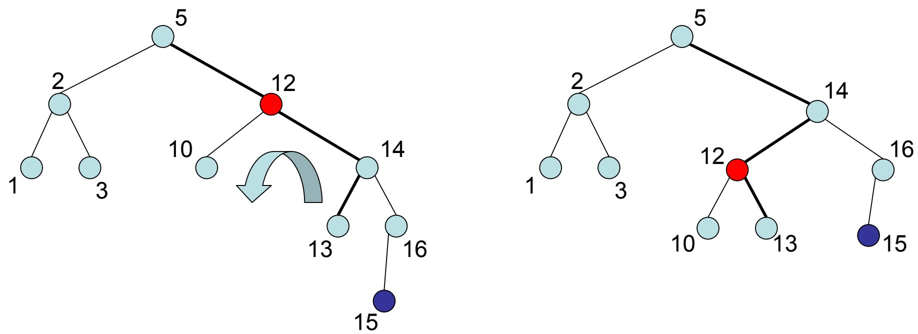
AVL-trær – insert

- La *x* være den laveste noden (lengst ned i treet) hvor ubalanse (høydeforskjell 2) oppstår.
- Det er fire mulige tilfeller (to og to er symmetriske venstre-høyre) avhengig av om innsettingen skjedde:

1. i venstre deltre av venstre barn av *x*, (VV)
2. i høyre deltre av venstre barn av *x*, (VH)
3. i venstre deltre av høyre barn av *x*, (HV)
4. i høyre deltre av høyre barn av *x*, (HH)

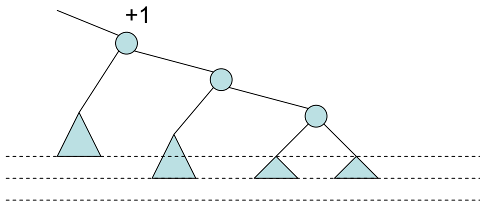
I tilfellene 1 og 4 rettes ubalansen med en ”enkel rotasjon”,
I tilfellene 2 og 3 rettes ubalansen med en ”dobbel rotasjon”.

AVL-trær – insert (enkel rotasjon)

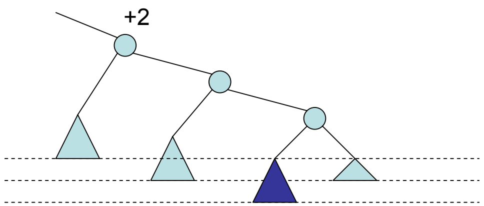


(HH)

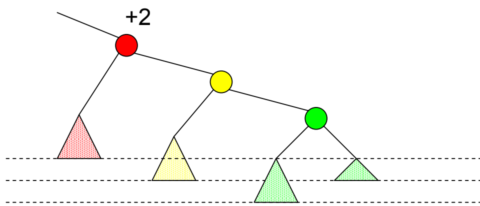
AVL-trær – insert (enkel rotasjon)



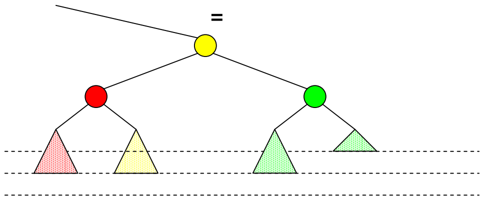
AVL-trær – insert (enkel rotasjon)



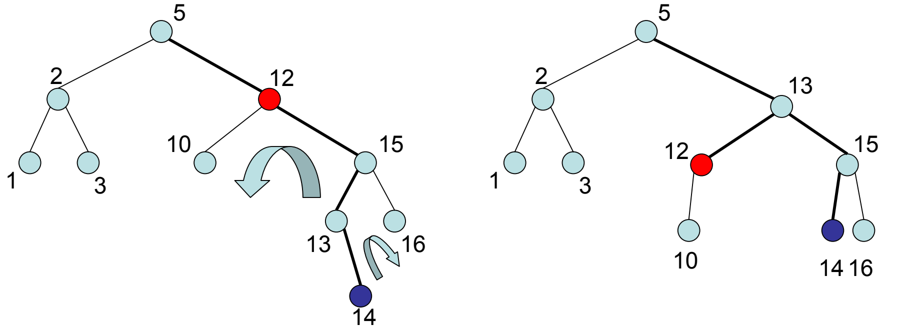
AVL-trær – insert (enkel rotasjon)



AVL-trær – insert (enkel rotasjon)

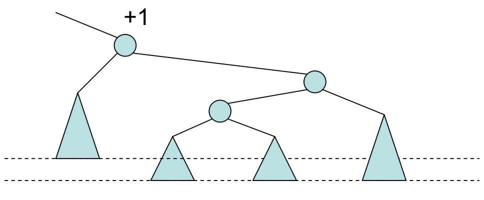


AVL-trær – insert (dobbel rotasjon)

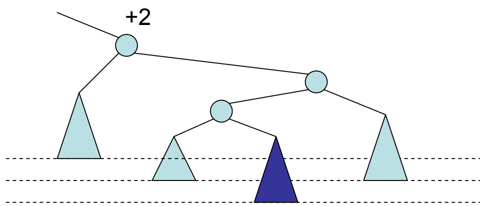


(HV)

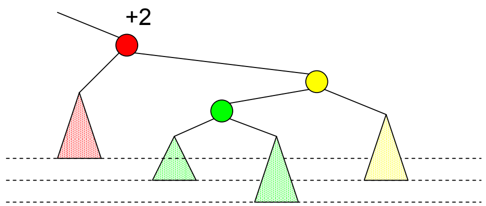
AVL-trær – insert (dobbel rotasjon)



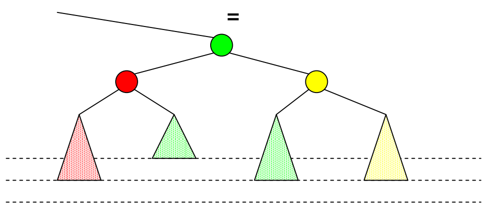
AVL-trær – insert (dobbel rotasjon)



AVL-trær – insert (dobbel rotasjon)



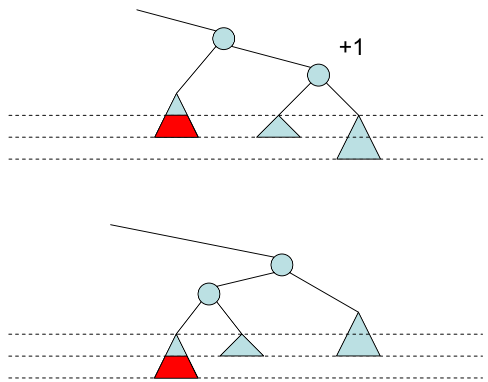
AVL-trær – insert (dobbel rotasjon)



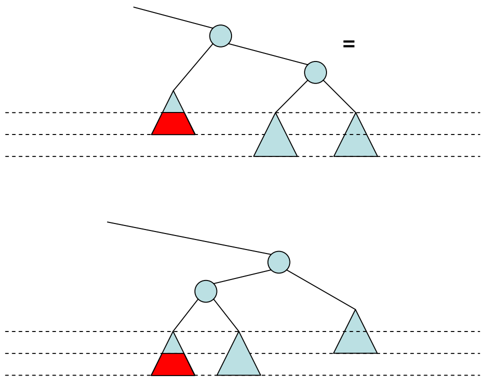
AVL-trær – insert

- Etter rotasjonen (enkel eller dobbel) er høyden av det berørte deltreet den samme som før insert.

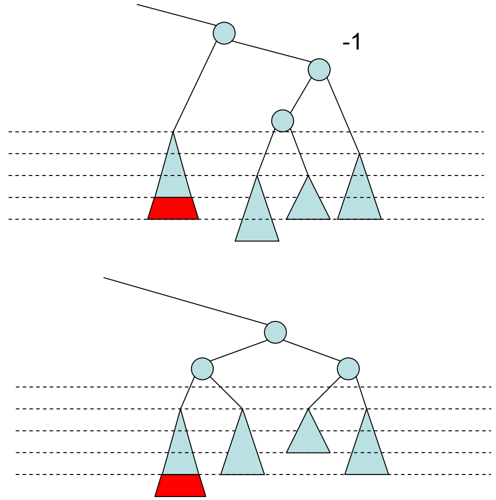
AVL-trær – delete (enkel rotasjon)



AVL-trær – delete (enkel rotasjon)



AVL-trær – delete (dobbel rotasjon)



AVL-trær – delete

- Når vi rebalanserer et deltre etter sletting, kan deltreet bli for lavt, slik at det blir ubalanse høyere opp i treet.
- Hele veien opp til roten av treet (log n nivåer) må vi sjekke for ubalanse, og evt. rebalansere.

AVL-trær – effektivitet

- Et AVL-tre med n noder har høyde h (*antall* nivåer) mellom $\log_2(n+1)$ og $1.44 \cdot \log_2(n+2)$
- Operasjonene *insert*, *delete*, *find* har kjøretid $O(h)$.

Demo

http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm

Splay-trær

Splay-trær er binære trær, med spesielle oppdaterings- og aksesseringsregler som gjør at treet over tid er effektivt å søke i.

- Det er ikke noe eksplisitt balansekrav, men treet tilpasser seg operasjonene som gjøres. Nylig aksesserte noder flyttes høyt opp i treet.
- En sekvens av m operasjoner på initielt tomt tre tar tid $O(m \log_2 n)$.
  Vi sier at amortisert kostnad per operasjon er $O(\log_2 n)$.

  Litt svakere enn å garantere at hver operasjon tar tid $O(\log_2 n)$, men over tid like bra. Ingen sekvens av operasjoner på treet er dårlig.

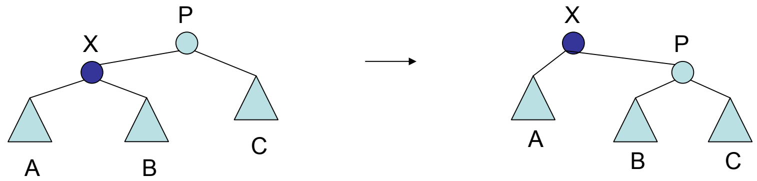
Splay-trær

- I vanlige binære trær kan vi ha lange sekvenser av operasjoner som alle tar $O(n)$ IKKE BRA!
- Med splay-trær har vi fortsatt worst case tid $O(n)$ for operasjonene, men en garanti for at en vilkårlig sekvens av m operasjoner ikke tar mer tid enn $O(m \log_2 n)$ BRA!
- Innimellom kan vi godta at noen operasjoner på Splay-treet tar tid $O(n)$
  Men for ikke ofte.
  Og vi må ha gjort nok billige operasjoner i forkant til at det veier opp.

Splay-trær

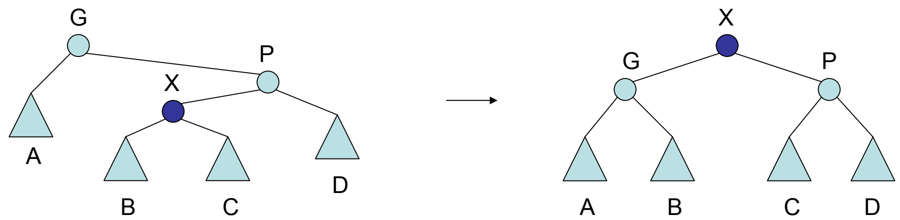
- Hvis en operasjon (f.eks find på en verdi) kan ta tid $O(n)$, så må opplagt denne operasjonen modifisere treet. Ellers kunne vi bare utført denne operasjonen m ganger og brukt tid $O(mn)$.
- Grunntanken bak splay-trær er at noder som aksesseres flyttes opp til roten (med AVL-aktige modifikasjoner).
- Noder som kan ligge langt nede i treet flyttes helt opp til roten, og drar da med seg andre noder oppover. Vi unngår å dytte andre noder like langt ned. Flyttingen har altså en balanserende effekt.
- Ofte slik at en nylig aksessert node aksesseres igjen.

Splay-trær – splaying



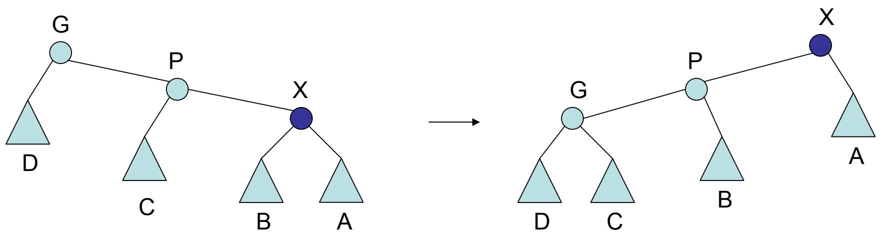
(zig)

Splay-trær – splaying



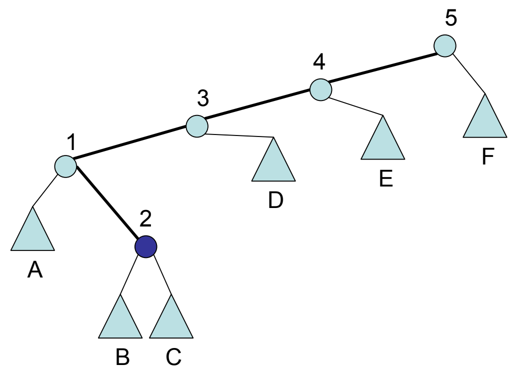
(zigzag)

Splay-trær – splaying

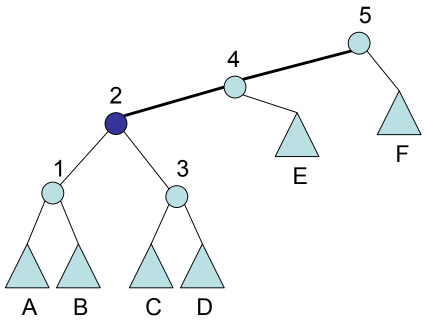


(zigzig)

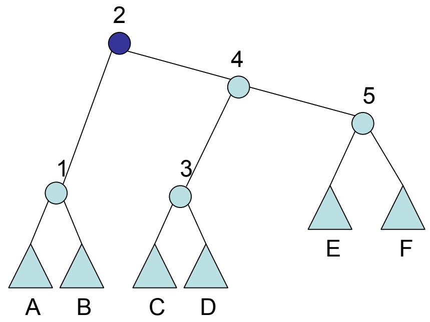
Splay-trær – eksempel



Splay-trær – eksempel



Splay-trær – eksempel



Splay-trær – operasjoner

- Insert
  - Søk nedover i treet til noden *x* som skal få nytt barn finnes.
  - Splay *x*. (*x* blir rot.)
  - Sett inn det nye elementet som ny rot, mellom *x* og *x* sitt venstre eller høyre barn (avhengig av om elementet som settes inn er større eller mindre).
- Find
  - Søk nedover i treet for å se om noden *x* med verdien finnes.
  - Splay x.
- Delete
  - Søk nedover i treet til noden *x* som slettes finnes.
  - Splay *x*. (*x* blir rot.)
  - Slett *x*. Treet deles i to.
  - Finn største node i venstre deltre *v*, og splay denne. (*v* blir rot i det venstre treet og har bare venstre barn.)
  - Koble det høyre deltreet på som *v* sitt høyre barn.

Splay trær – amortisert tid

Amortisert kjøretid for operasjoner på et splay-tre er $O(\log_2 n)$.
(Altså: *m* operasjoner tar tid $O(m \log_2 n)$, uansett hvilke *m* operasjoner det er.)

Vi bruker den såkalte potensialmetoden.

Tenk på det som en bankkonto du kan sette inn penger på og ta penger ut av.

- Gjør vi en billig operasjon, øker vi potensialet tilsvarende, slik at vi har penger på konto til å gjøre dyre operasjoner seinere.
- Gjør vi en dyr operasjon, må vi ta penger fra kontoen.
- Overtrekker vi kontoen, har vi tapt. (Da er det en sekvens av *m* operasjoner som bruker for lang tid.)

Splay trær – amortisert tid

Potensialfunksjonen vi skal bruke for et tre T er

$$\Phi(T) = \sum_{i \in T} \log_2 S(i)$$

$$S(i) = \text{antall etterkommere av i, medregnet i.}$$

For å forenkle litt skriver vi

$$R(i) = \log_2 S(i)$$

Slik at

$$\Phi(T) = \sum_{i \in T} R(i).$$

Splay trær – amortisert tid

Lemma. *Hvis $a + b \le c$ og a og b er positive heltall, så er* $\log_2 a + \log_2 b \le 2 \log_2 c - 2$.

Bevis.

$$\sqrt{ab} \le \frac{a+b}{2} \quad \text{, følger av gjennomsnittsteoremet}$$

$$\sqrt{ab} \le \frac{c}{2}$$

$$ab \le \frac{c^2}{4}$$

$$\log_2(ab) \le \log_2\left(\frac{c^2}{4}\right)$$

$$\log_2 a + \log_2 b \le 2\log_2 c - 2$$

$$(a-b)^2 \ge 0$$

$$a^2 - 2ab + b^2 \ge 0$$

$$a^2 + 2ab + b^2 \ge 4ab$$

$$(a+b)^2 \ge 4ab$$

$$\frac{(a+b)^2}{4} \ge ab$$

$$\frac{(a+b)}{2} \ge \sqrt{ab}$$

Splay trær – amortisert tid

Teorem. *Amortisert tid for å splaye en node X i et tre med rot T er maksimalt* $3(R(T)-R(X)) + 1 = O(\log_2 n)$

Bevis. Hvis X er rotnoden, er det ingen rotasjoner, aksessering tar tid 1, og teoremet holder.

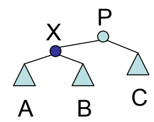
Vi antar derfor at det gjøres minst en rotasjon når vi splayer.
La $R_i(X)$ og $S_i(X)$ være verdiene før splay-steget, og
la $R_f(x)$ og $S_f(X)$ være verdiene etter steget.

Vi ser på operasjonene (zig, zigzag og zigzig) hver for seg.

Bevis.
(forts.)

Tidsforbruk: 1 (for den ene rotasjonen)

Endring i potensial: $R_f(X) + R_f(P) – (R_i(X) + R_i(P))$



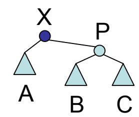
$$AT_{zig} = 1 + R_f(X) + R_f(P) - (R_i(X) + R_i(P))$$

$$\Downarrow S_i(P) \ge S_f(P)$$

$$AT_{zig} \le 1 + R_f(X) - R_i(X)$$

$$\Downarrow S_f(X) \ge S_i(X)$$

$$AT_{zig} \le 1 + 3(R_f(X) - R_i(X))$$

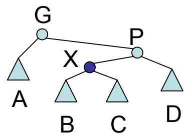


(zig)

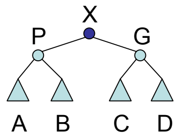
Bevis.
(forts.)

Tidsforbruk: 2 (dobbel rotasjonen)

Endring i potensial: $R_f(X) + R_f(P) + R_f(G) - (R_i(X) + R_i(P) + R_i(G))$



$$AT_{zigzag} = 2 + R_f(X) + R_f(P) + R_f(G) - (R_i(X) + R_i(P) + R_i(G))$$
$$\Downarrow S_f(X) = S_i(G)$$
$$AT_{zigzag} = 2 + R_f(P) + R_f(G) - (R_i(X) + R_i(P))$$
$$\Downarrow S_i(X) \le S_i(P)$$
$$AT_{zigzag} \le 2 + R_f(P) + R_f(G) - 2R_i(X)$$
$$\Downarrow S_f(P) + S_f(G) \le S_f(X) \ + \text{ Lemma}$$
$$AT_{zigzag} \le 2 + 2R_f(X) - 2 - 2R_i(X)$$
$$AT_{zigzag} \le 2(R_f(X) - R_i(X))$$
$$AT_{zigzag} \le 3(R_f(X) - R_i(X))$$

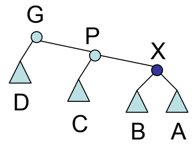


(zigzag)

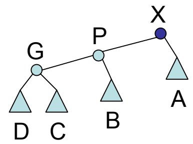
Bevis.
(forts.)

Tidsforbruk: 2 (dobbel rotasjonen)

Endring i potensial: $R_f(X) + R_f(P) + R_f(G) - (R_i(X) + R_i(P) + R_i(G))$



$$AT_{zigzig} = 2 + R_f(X) + R_f(P) + R_f(G) - (R_i(X) + R_i(P) + R_i(G))$$
$$\Downarrow S_f(X) = S_i(G)$$
$$AT_{zigzig} = 2 + R_f(P) + R_f(G) - (R_i(X) + R_i(P))$$
$$\Downarrow S_f(P) \le S_f(X)$$
$$AT_{zigzig} \le 2 + R_f(X) + R_f(G) - (R_i(X) + R_i(P))$$
$$\Downarrow S_i(X) \le S_i(P)$$
$$AT_{zigzig} \le 2 + R_f(X) + R_f(G) - 2R_i(X)$$
$$\Downarrow 2R_f(X) - R_i(X) - R_f(G) \ge 2$$
$$AT_{zigzig} \le 2R_f(X) - R_i(X) - R_f(G) + R_f(X) + R_f(G) - 2R_i(X)$$
$$AT_{zigzig} \le 3(R_f(X) - R_i(X))$$

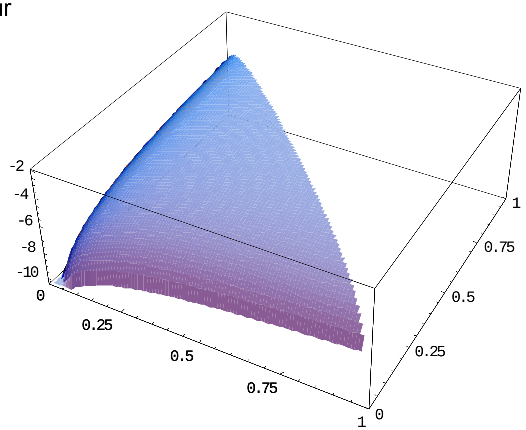


(zigzig)

En liten sidebemerkning for å vise
$2R_f(X)$-$R_i(X)$-$R_i(G) \ge 2$

log x + log y, x,y> 0 og x + y < 1 har maksimal verdi på -2 når x = y = 1/2, se figur

$$-1 \cdot (2R_f(X) - R_i(X) - R_f(G)) =$$
$$-2R_f(X) + R_i(X) + R_f(G) =$$
$$R_i(X) - R_f(X) + R_f(G) - R_f(X) =$$
$$\log\left(\frac{S_i(X)}{S_f(X)}\right) + \log\left(\frac{S_f(G)}{S_f(X)}\right)$$



Vi har altså

$$\log\left(\frac{S_i(X)}{S_f(X)}\right) + \log\left(\frac{S_f(G)}{S_f(X)}\right) \le -2 \ ,$$

ergo er $2R_f(X) - R_i(X) - R_f(G) \ge 2$.

Bevis.
(forts.)

Til slutt ser vi på en serie av *m* rotasjoner, bare den siste kan være zig:

$$3(R_2(X) - R_1(X)) +$$
$$3(R_3(X) - R_2(X)) +$$
$$3(R_4(X) - R_3(X)) +$$
$$\vdots$$
$$3(R_m(X) - R_{m-1}(X)) + 1 =$$
$$3(R_m(X) - R_1(X)) + 1$$

Siden X blir rot til slutt, er dette det samme som

$$3(R_f(T) - R_i(X)) + 1 =$$
$$3(\log_2 n - \log_2 S_i(X)) + 1 \le 3\log_2 n + 1 = O(\log n).$$