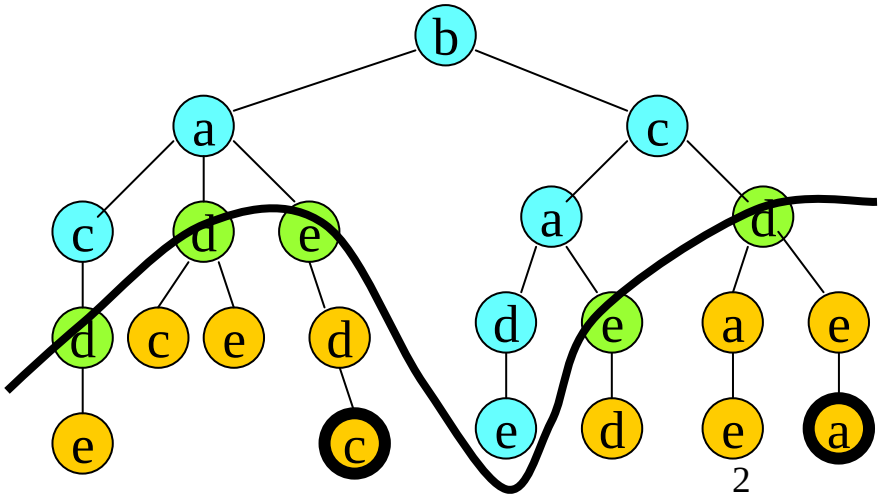
INF 3/4130
18. oktober 2007

- Dagens forelesning: Kapittel 23 i hovedboka
  - 23.5: Avslutte Branch and Bound
  - 23.6: Trær og strategier for spill med to spillere
- Oblig 2 har ligget ute en stund. Frist 26 oktober.
- Ikke vanlig forelesning neste uke, på grunn av ”dagen@ifi”:
  - Kl 14.15: Foredrag, Store aud: *Algoritmeperler i søk - teori og praksis*
    Kathrine Hammervold og Øystein Haug Olsen (FAST)
  - Kl 15.15: Gjesteforelesning spesielt for vårt kurs, Lille aud: *Sjakkalgoritmer*
    Rune Djurhuus: Internasjonal stormester, sjakkspalte i Aftenposten, nå FAST
  - Programmet for ”dagen@ifi”: http://dagen.at.ifi.uio.no/
- Uka etter det igjen:
  - Dino Karabeg starter forelesninger om NP-kompletthet, uavgjørbarehet etc.

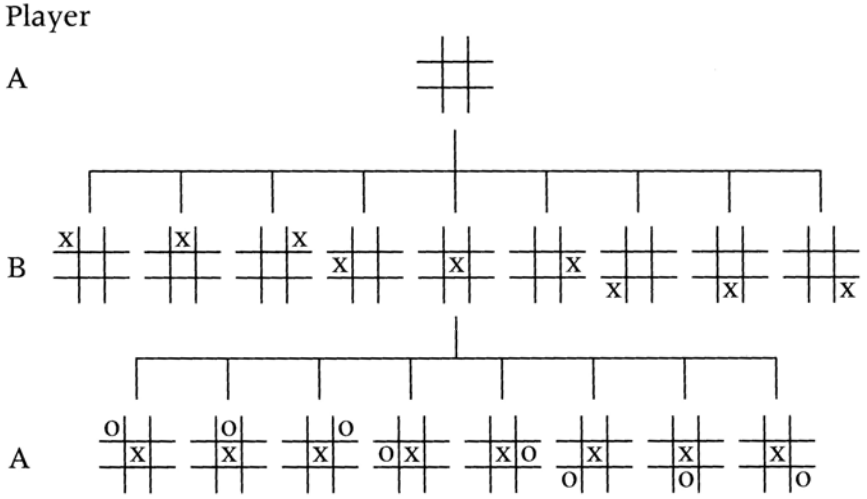
Kap 23.4: ”Least Cost Branch and Bound”

- Fra kapittel 10: Branch and Bound med prioritet
  - ”Live-node”-mengden er prioriteskø
    - med en passelig heuristikk som prioritet (hvor ”lovende” er noden)
- Vi lager oss en variant over denne for følgende problem:
  - Det finnes diverse målnoder, og de har alle en verdi
  - Vi skal finne den målnoden med minst verdi
- Eksempel: Travelling Salesmann-problemt (TS), med veilengde oppgitt for alle par av byer. Finn den korteste rundturen.
  - Vi kan bygge ut alle mulige veier ut fra en startnode
  - Spesielt her: Alle målnoder vil ligge på samme nivå på bunnen av treet
- Vi forlanger en heuristikk slik at:
  - h(n) ikke er større enn verdien av noen målnode i subtreet til n Mange slike er mulige for TS (Oblig2).
- Vi kjører så prioritert branch and bound ut fra denne heuristikken
  - Og holder en variabel M med den beste (minste) løsningen vi har sett.
  - Når den noden vi får ut av prioriteskøen er større enn M er vi i mål

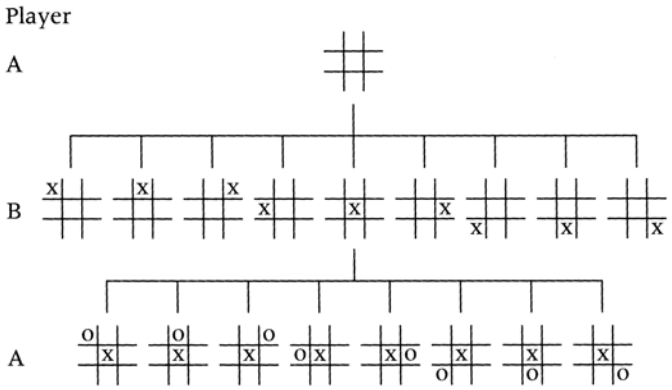


Kapittel 23.5: Spill, spilltrær og strategier

- Vi kunne lage et tre av avgjørelser for f.eks. ”15-spillet” eller leting etter en beste rundtur for en ”travelling salesman”, og prøve å finne veien til en løsningsnode eller en beste vei.
  - Men da er det bare én ”spiller”, og man er ute etter en ”nøytral” løsning
- Om det er to spillere som spiller mot hverandre blir saken annerledes. Det den ene ser som en bra situasjon ser den andre som dårlig.
  - Trærne av mulige spill blir ofte kjempestore. For sjakk estimert til 10^{100} noder.
  - Men selv for et enkelt spill som ”tick-tack-toe” kan ved rett fram utlegging for 9! = 362 880 noder.
- Starten av et tre for tic-tac-toe blir slik:
- Spilleren som starter er A
  - Vi skal også gjøre alle vurderinger ut fra A synspunkt (inntil videre)



Treet blir fort stort

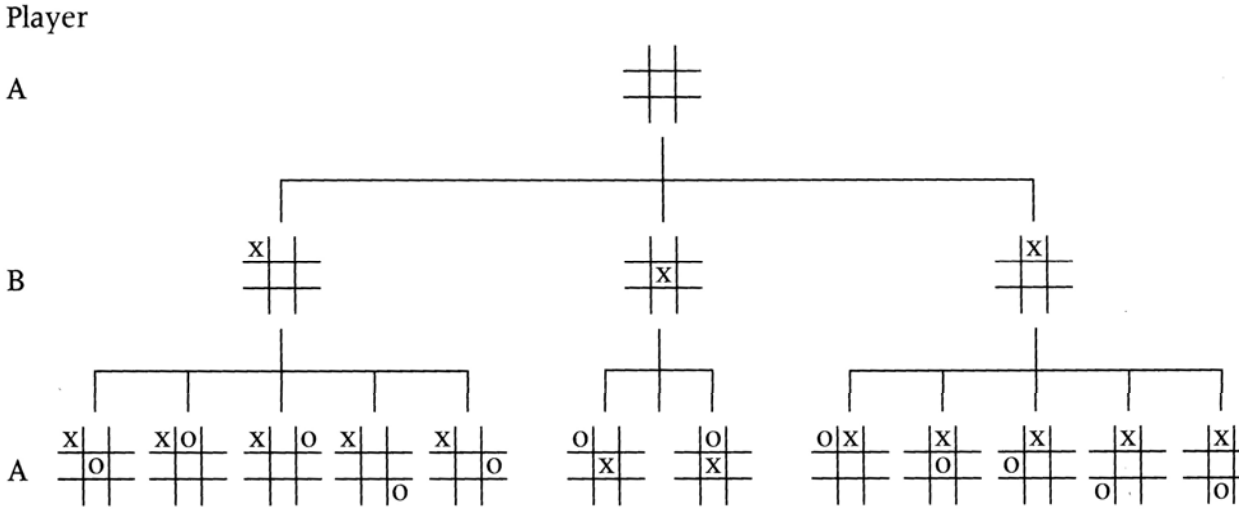


9 noder	1 node
9*8 noder	9 noder
9*8*7 noder	72 **forskjellige** noder
9*8*7*6 noder	252 **forskjellige** noder
9*8*7*6*5 noder	756 **forskjellige** noder
.....	1260 **forskjellige** noder
9*8*7*6*5*4*3*2*1 = 362 880 noder	.....
	126 **forskjellige** noder

Altså, man kan tjene masse på å kjenne igjen like noder, og ikke gjøre undersøkelsen om igjen.

Det er ikke så greit ved dybde først

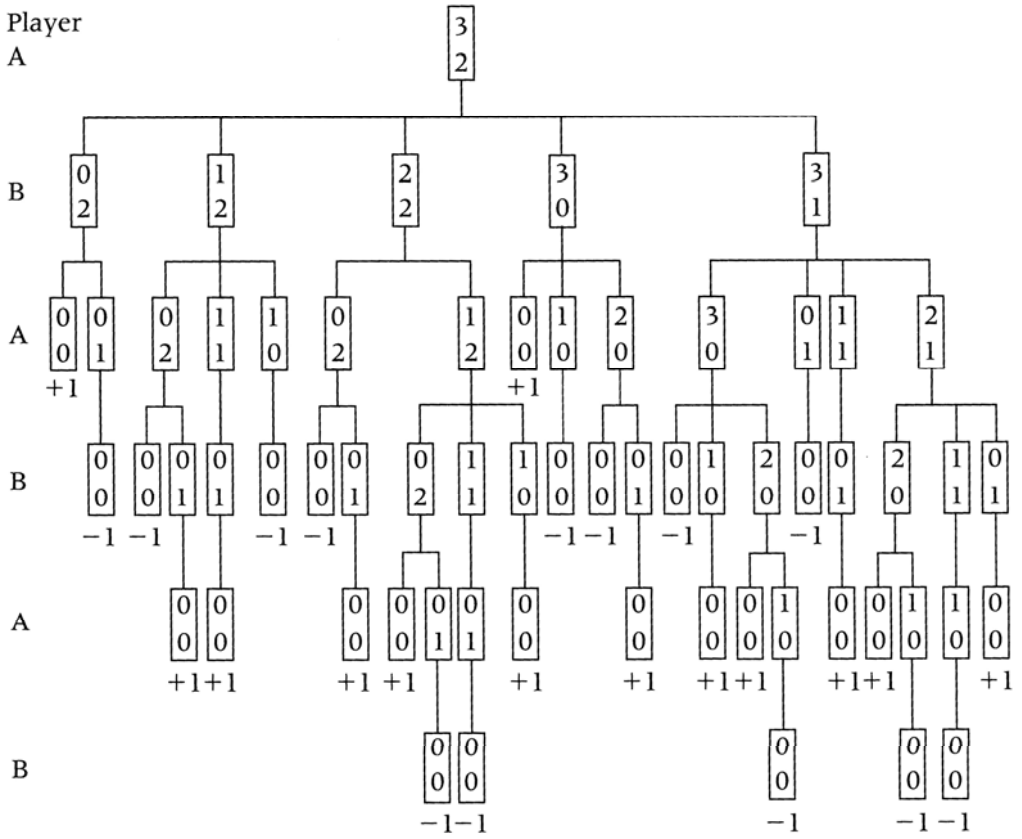
Ta bort symmetrier etc.



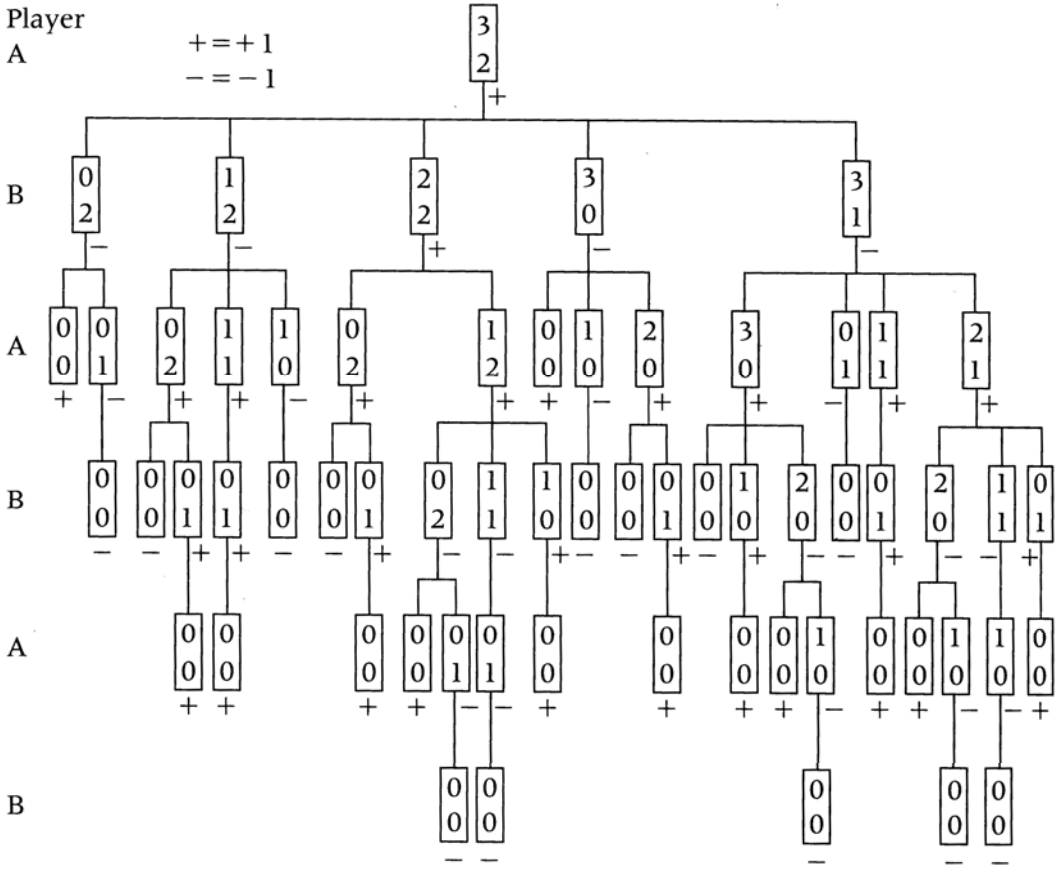
- Man kan også spare mye på lokal optimalisering:
  - Ta bare med de barna som er essensielt forskjellige
  - Altså her, ta bort løsninger som er symmetriske av hverandre
  - Men dette gir ikke globalt veldig god avskjæring

Fystikk-spillet Nim

- Vi ser bare på ”null-sum-spill”. Altså, når den ene spilleren vinner noe så taper den andre tilsvarende.
- Enkleste variant: Det er to utfall, A vinner (+1) og B taper (-1), eller omvendt
- En del spill har også ”uavgjort”, for eksempel tic-tac-toe.
- Spillet Nim:
  - Start med to bunker med fyrstikker.
  - Antall: m og n
  - En spiller kan ta så mange man vil fra én bunke
  - Den som tar siste fyrstikken har tapt.
- Her blir det aldri uavgjort
- Om vi starter med m=3 og n=2 er hele spilltreet tegnet ut her:
- Verdi for A satt på hver slutt-node

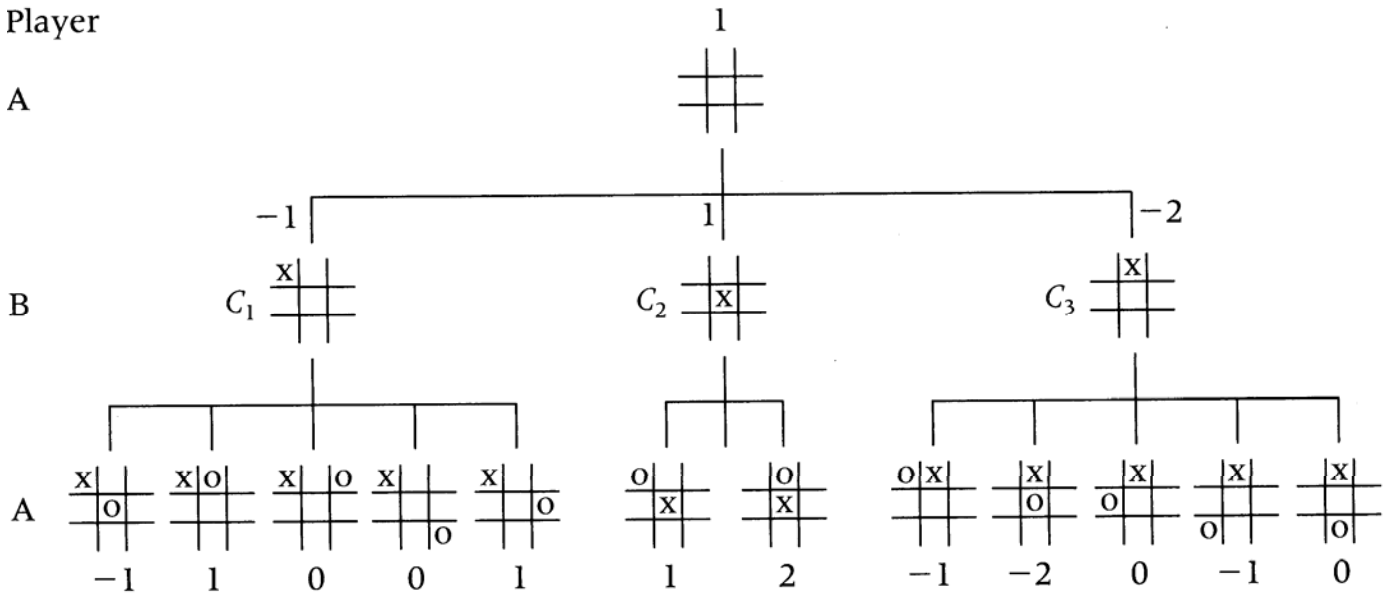


Hvordan finne strategi for å vinne, sett fra A?



- Vi angir hele tiden verdien av en node sett fra A
- A vil gjøre optimale trekk sett fra seg: Vil velge til node med størst verdi
- A må regne med at B gjør optimale trekk sett fra seg.
- Siden verdiene noteres sett fra A, vil da B trekke til den minste subnoden
- For å få vite verdien av en node må vi derfor vite verdiene i alle subnodene
- Dette kan vi gjøre ved dybde først gjennomgang, og å beregne verdiene postfiks.

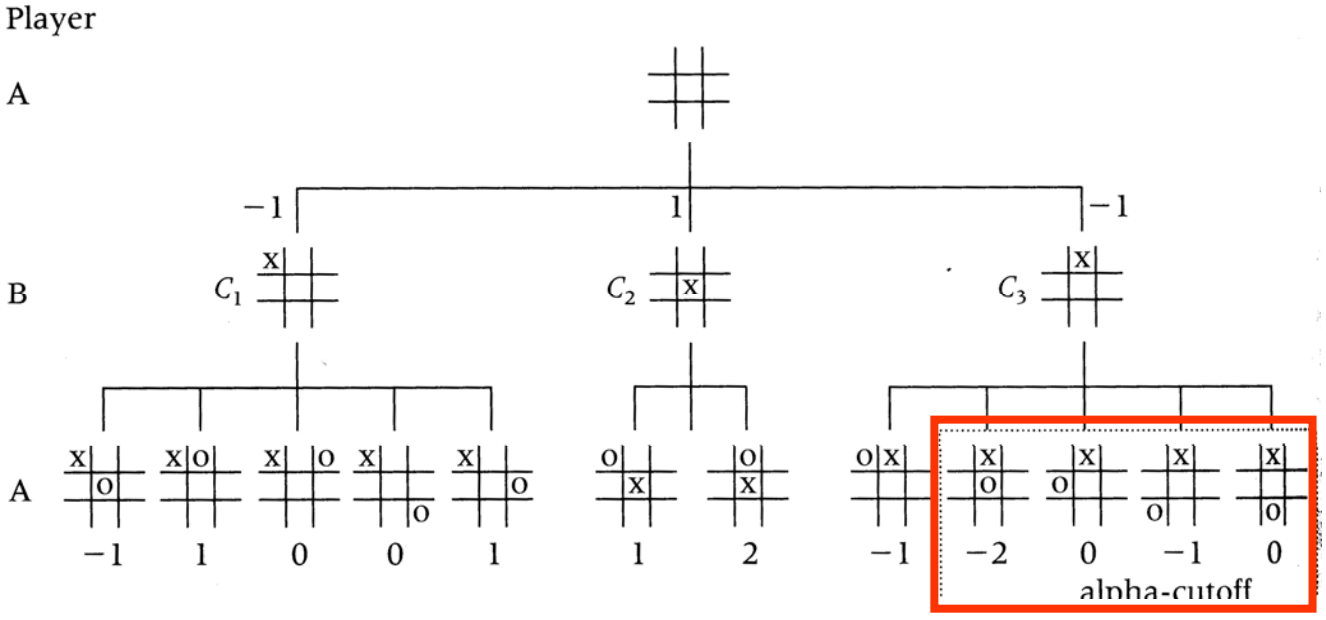
Oftest kan man ikke se gjennom hele spillet



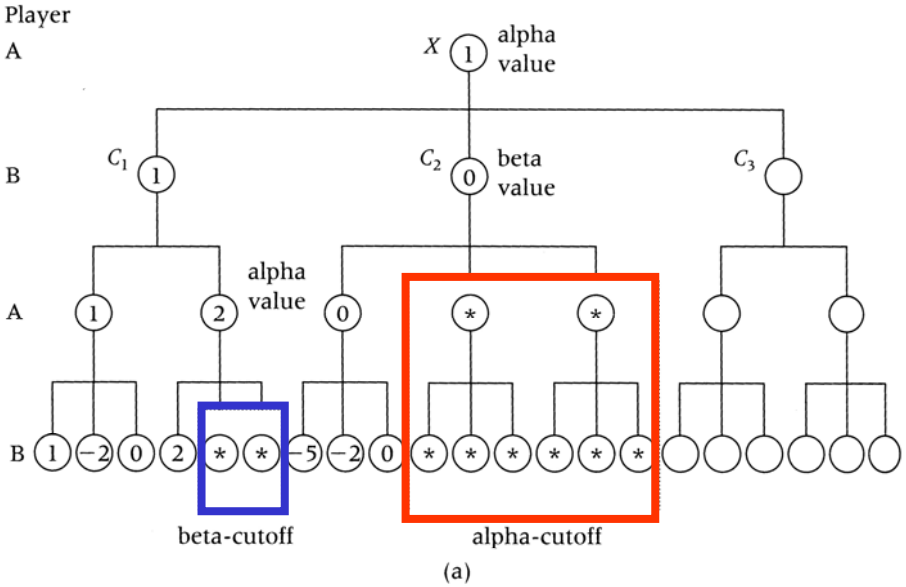
- Man må da f.eks. gå en viss dybde, og her komme med et estimat for hvor god situasjonen er, sett fra A
- Her går vi til dybde 3.
- Heuristikk her: Antall ”vinnende” linjer for A minus det tilsvarende for B.
- A vil derfor her trekke til C_2, som ser bra ut
- Men denne heuristikken er ikke lur når vi har gjort noen trekk. Tar ikke hensyn til at å vinne er bedre enn alle heuristikker. Gir derfor vinst-node verdi $+\infty = 9$
- Da får vi en god strategi for A.
- Dog: Om begge spiller perfekt, så blir det uavgjort i tic-tac-toe.
- Hvor dypt kan man gå i sjakk? Og hvordan lage god heuristikk?? Kommer neste torsdag!!

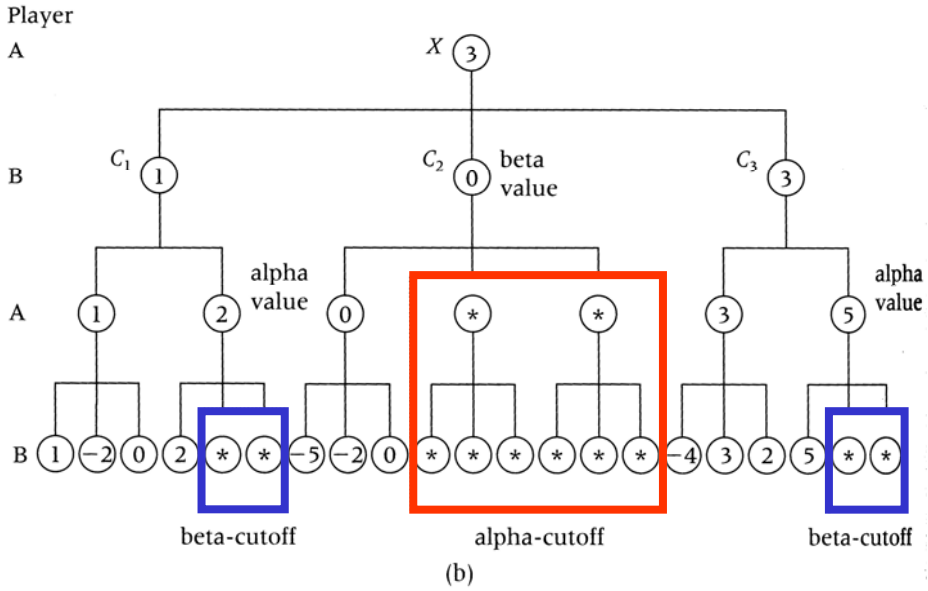
Alfa-beta-avskjæring

- Ved siden av å forsøke å redusere treet ved å se på symmetrier og/eller kjenne igjen noder som allerede er analysert, kan man gjøre ”alfa-beta-avskjæring”
- Dette er som følger:
  - Jeg, A, vurderer trekk ett etter ett fra en gitt A-situasjon A1
  - Jeg har allerede sett så gode trekk at jeg kan oppnå verdien U
  - Jeg ser så på neste potensielle trekk som fører til situasjonen B1, og ser at B herfra kan gjøre et trekk som fører til en god situasjon for B (men dårlig for A), som sett fra A har verdien V. Da kan ikke verdien i B1 bli større en V (B minimaliserer!)
  - Om da V<U så ”gidder” ikke A gå videre å studere om B skulle ha et enda bedre trekk. Det kan ikke gjøre A mer lysten på trekket til B1.



Eksempler på alfa-beta-avskjæring





- Avskjæring som gjøres ut fra A's betrakning kalles alfa-avskjæring
- Samme betrakning som gjøres fra B-tilstander, men med omvendt fortegn. Disse kalles beta-avskjæring
- Ved siden av ser vi eksempler på både alfa- og beta-avskjæring

- Når vi skal implementere alfa-beta-avskjæring kan det være greit å skifte synspunkt for hvert nivå.
  - Da skal det alltid maksimaliseres
  - Men man må skifte tegn på verdiene når man skifter nivå.
- En slik implementasjon står på neste foil

Alfa-beta-søk

```
real function ABNodeValue(
        X,   // Noden vi vil ha alfa/beta-verdien for. Barn: C1, C2, … , Ck
        numLev,   // Antall nivåer som står igjen
        parentVal)   // Alfa/beta-verdien fra forelder-noden (-LB fra foreldren)
// returverdi: Den endelige alfa/beta-verdien for denne noden
        real LB;  // Løpende LowerBound for alfa/beta-verdi for denne noden

        if <X er terminal-situasjon> then {return <verdien av denne sett fra X>;}
        else if numLev = 0 then {return <estimat av kvaliteten av denne, sett fra X>;}
        else {
           LB :=  - ABNodeValue(C1, NumLev-1, ∞);  // Heller -∞?  Se neste foil.
           for i :=  2 to k do {
               if  LB >= parentValue  then {return LB;}
               else { LB := max(LB, -ABNodeVal(Ci, Numlev-1, -LB) ); }
           }
         }
         return LB;
}
Startkall:      situasjonsKvalitet := ABNodeValue(rotnoden, 10, - ∞)
```

Lagt til etter forelesningen, trykkfeil m.m.

- Først: Det var altså noen enkle trykkfeil i programmet på side 741 i boka:
  - ”AB” manglet i navnet på prosedyren, der den blir kalt rekursivt
  - Det mangler en sluttparentes på enden av linja der **max** kalles

- Disse er rettet i skissen på forrige foil, men på forelesningen lurte vi også på om det skulle være -∞ som siste parameter (ParentValue) i linja midt i programmet.
  - Foreleseren (Stein K.) mener det må være -∞.
  - Grunnen er at ParentValue jo ikke blir negért (eller forandret på annen måte) før den brukes, og den brukes bare i testen ”if LB >= ParentValue then”... Her er altså ParentValue en fast verdi, mens LB vokser. Om ParentValue er +∞ vil jo testen aldri kunne slå til, mens den slår til med en gang om den er -∞. Og det er jo det siste som er vitsen.