# Trädsökning och datorschack

Johnny Bigert

Nada, KTH

**johnny@kth.se**

# Datorschack

- Hobby: schackdatorn Rainman

  http://www.nada.kth.se/~johnny/chess_comp.html
- Påbörjad i oktober 2002
- Spelar på free internet chess server (FICS, www.freechess.org) under namnet RainmanC
- Har spelat 2500 matcher (030228)
- FICS-rating version 060: ca 1875

# Datorschack

Lite kuriosa:

- 5000 rader C++
- Kommande version 073 innehåller
  - Bitbräden, alfa-betasökning, transpositionstabell, MTD(f)-sökning, enkel quiescencesökning, enhanced transposition cutoff, SEE-algoritm, hashtabell för bondestruktur, öppningsbok, slutspelsdatabas, dragupprepningsbehandling, Winboardinterface, tidskontroll m.m.
- Internmöte v073 mot v060: +26 =2 -2

# Översikt

- Allmänt
  - Minimax-sökning
  - Alfa-beta-sökning
  - Transpositioner
- Dragsortering
  - Killer move
  - Historik
  - Intern iterat. fördj.
  - Quiescence search
- Öppningsbok, slutspelstabeller
- Söktekniker
  - Iterativ fördjupning
  - Aspirationssökning
  - NegaScout
  - MTD(f)
  - Utökad transp.besk
- Heuristiker
  - Nulldrag
  - Futility
  - Selective extensions

# Minimaxsökning

(eng: minimaxing, negamaxing)

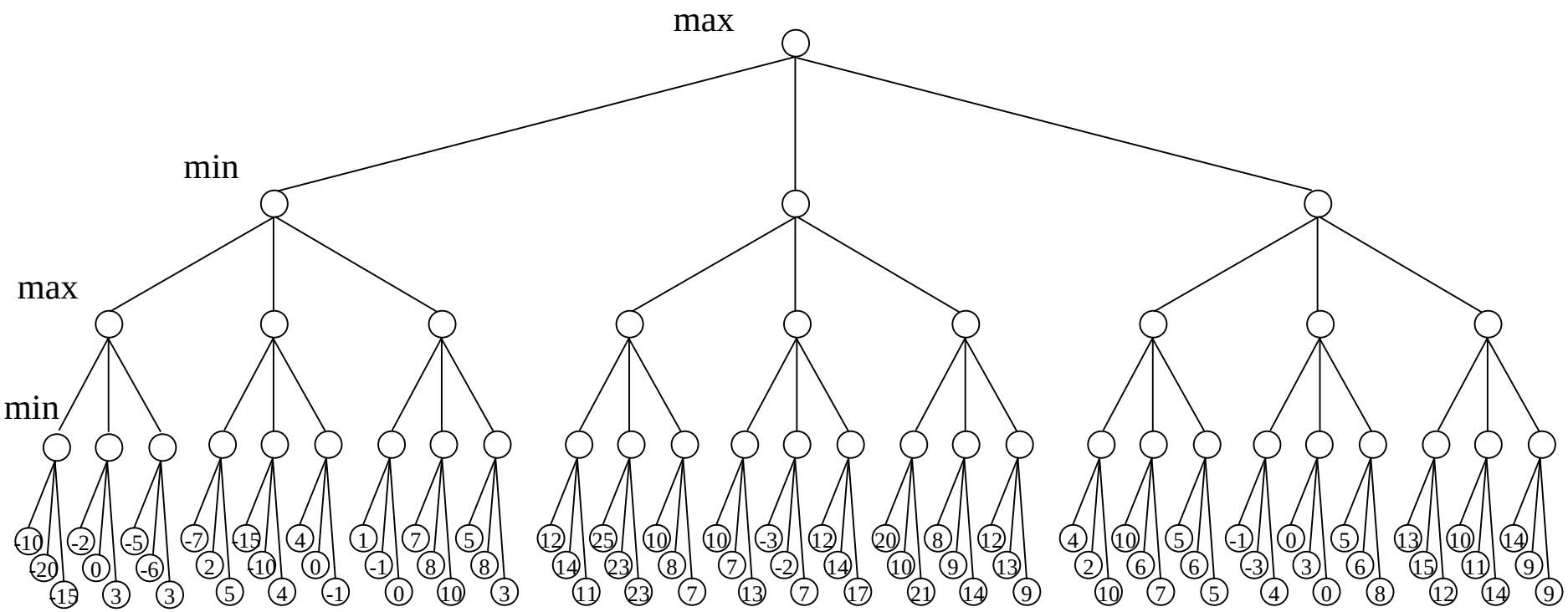
- Schack: det som är bra för dig är dåligt för din motståndare
- Du vill maximera din poäng, din motståndare vill minimera din poäng
- Varannat drag ditt,
  varannat din motståndares
- Maximera varannat, minimera varannat

# Minimaxing

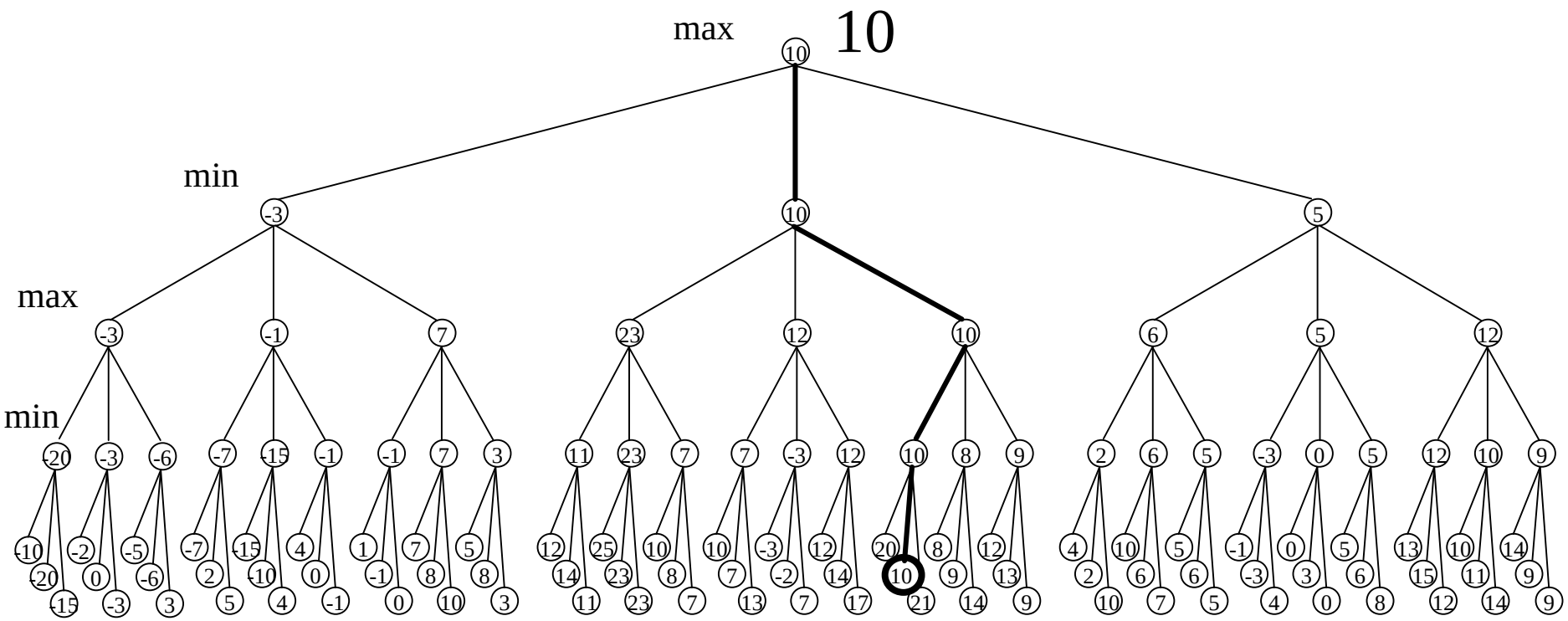
Trädsökning:

- Varje nod är ett bräde
- Varje kant är ett drag
- Utvärdering sker i löven
- Maximera varannat djup,
  minimera varannat djup

# Minimaxsökning



# Minimaxsökning



# Minimaxsökning

- Antag 50000 positioner/sekund
- Förgreningsfaktor i schack: ca 30

| Djup (halvdrag) | Positioner | Tid |
|---:|---:|---:|
| 2 | 900 | 0.018 s |
| 3 | 27,000 | 0.54 s |
| 4 | 810,000 | 16.2 s |
| 5 | 24,300,000 | 8 min |
| 6 | 729,000,000 | 4 h |
| 7 | 21,870,000,000 | 5 dagar |

# Minimaxsökning

- Snabbschack: ca 5 min per person och parti dvs 5-10 s per drag
- Vi hinner inte ens söka klart djup 4!

Slutsats:

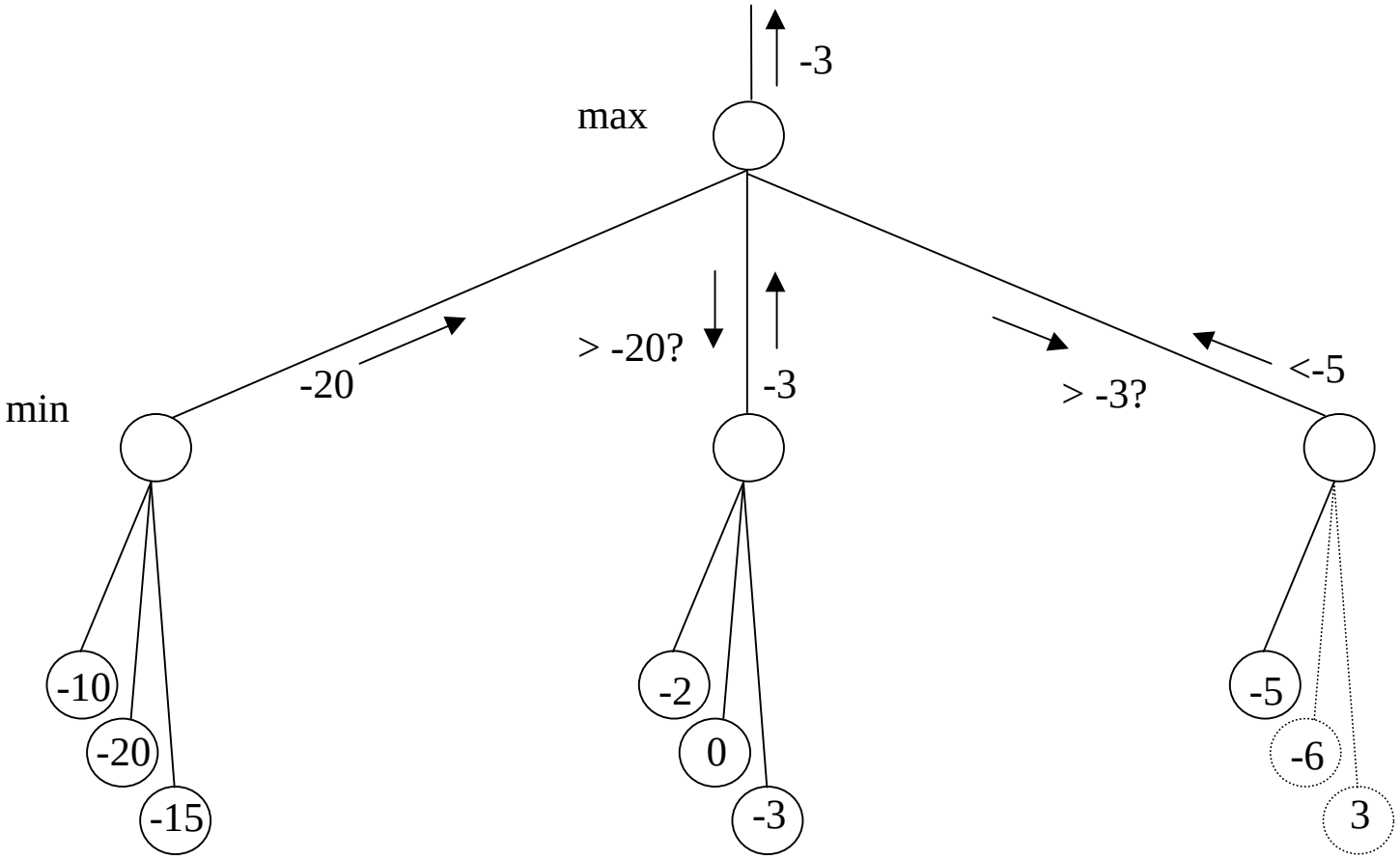
- Brute force ogörligt, träden växer för snabbt

# Alfa-beta-sökning

Observationer:

- När vi maximerar har vi en undre gräns för det rätta värdet
- När vi minimerar får vi övre gränser
- Om inget överlapp, omöjligt att hitta bättre värde

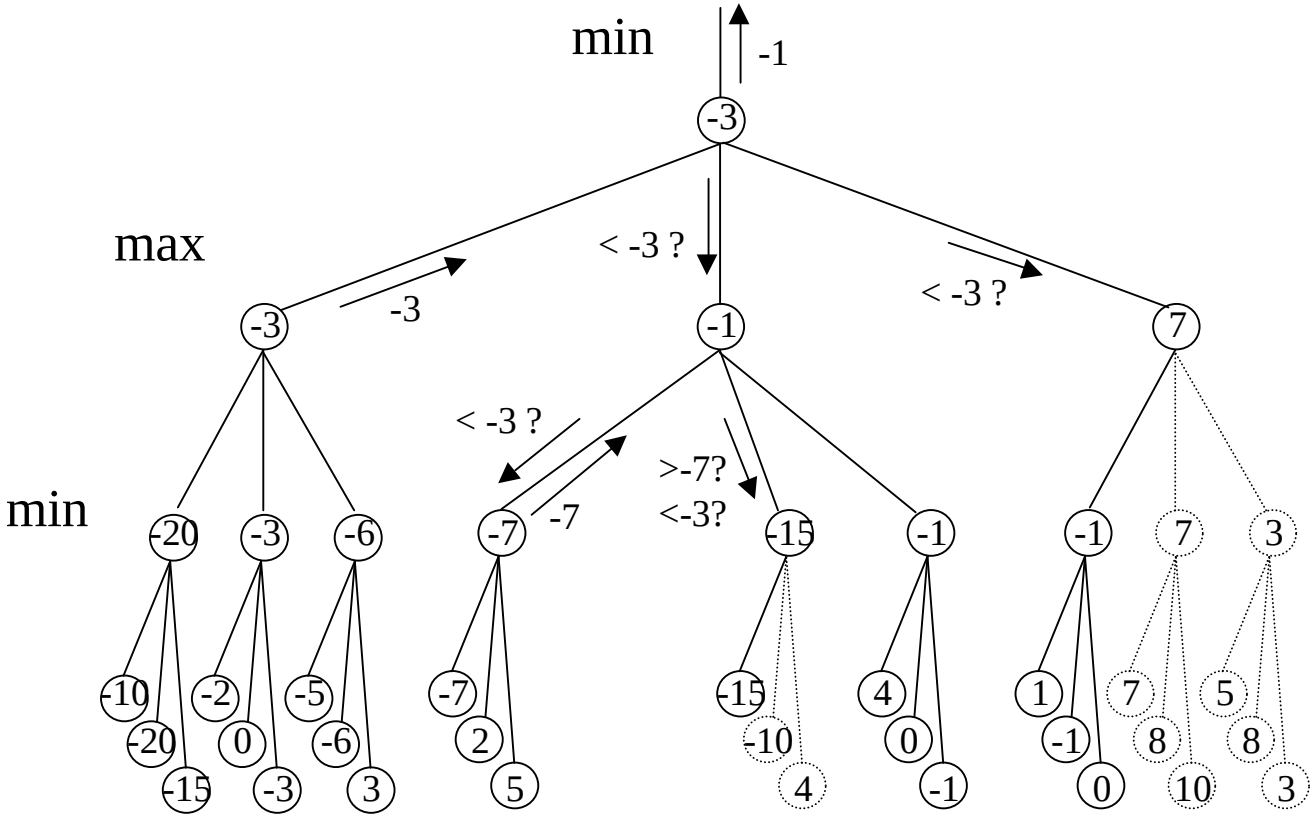
# Alfa-beta-sökning



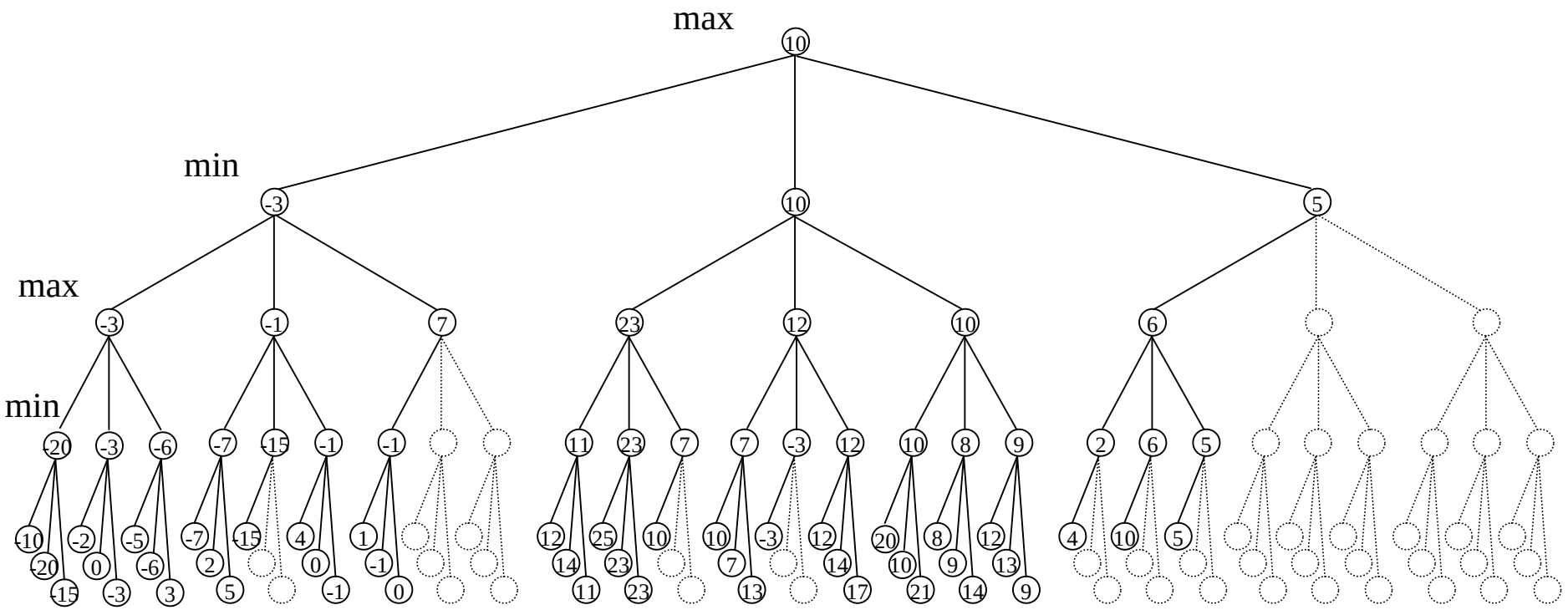
# Alfa-beta-sökning

- Vi kallar den undre gränsen för alfa
- Vi kallar den övre gränsen för beta
- Algoritmen heter **alfa-beta-sökning** (eng: alpha-beta-pruning)
- De grenar vi inte besökar kallas beskurna (eng: pruned)

# Alfa-beta-sökning



# Alfa-beta-sökning




Besökta: 45, beskurna: 36

```
function AlphaBeta(node, alpha, beta, depth)
    if depth == 0 then g = evaluate(node);
    else if node == MAXNODE then
        g = -INFINITY;
        a = alpha;
        c = firstchild(node);
        while (g < beta) and (c != NOCHILD) do
            g = max(g, AlphaBeta(c, a, beta, depth - 1));
            a = max(a, g);
            c = nextbrother(c);
    else /* node is a MINNODE */
        g = +INFINITY;
        b = beta;
        c = firstchild(node);
        while (g > alpha) and (c != NOCHILD) do
            g = min(g, AlphaBeta(c, alpha, b, depth - 1));
            b = min(b, g);
            c = nextbrother(c);
    return g;
```

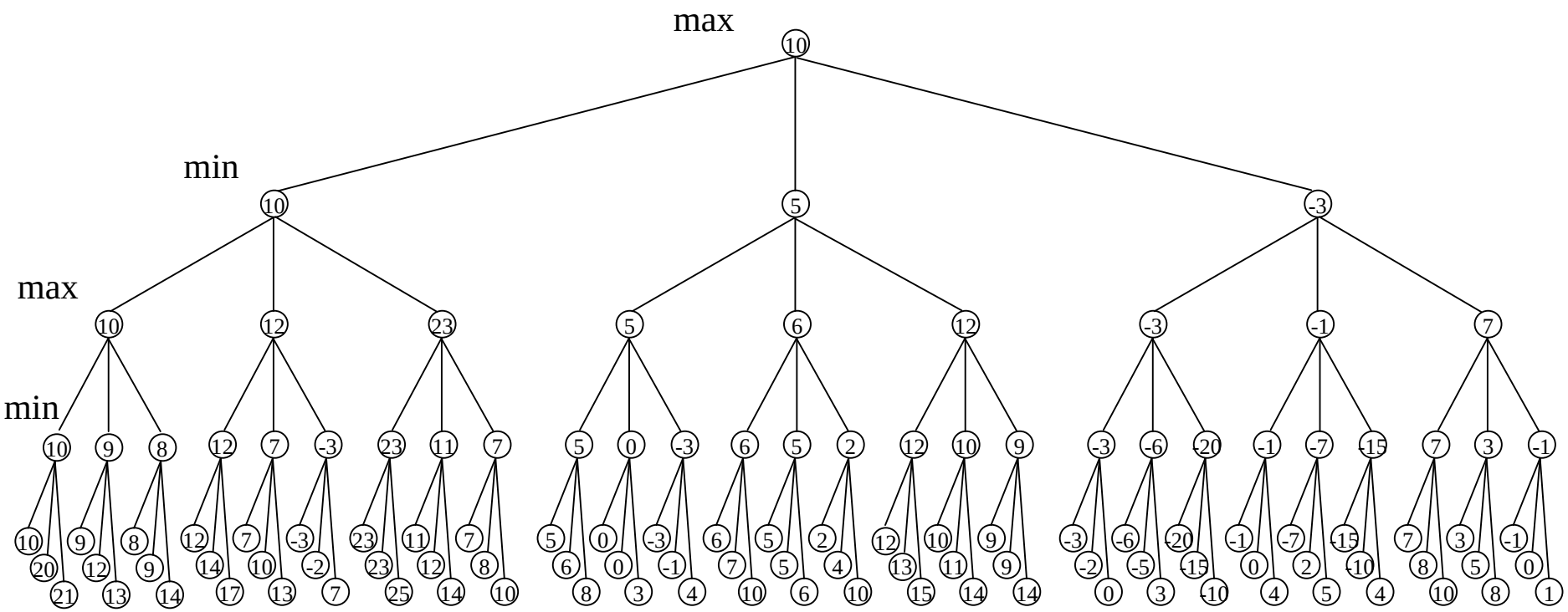
# Dragordning

- (eng: move ordering)
- Den mest kritiska delen av alfa-beta-algoritmen är ordningen på dragen
- Bäst drag först ger tidiga beskärningar
- Ditt bästa drag maximerar din poäng
- Motståndarens drag minimerar din poäng

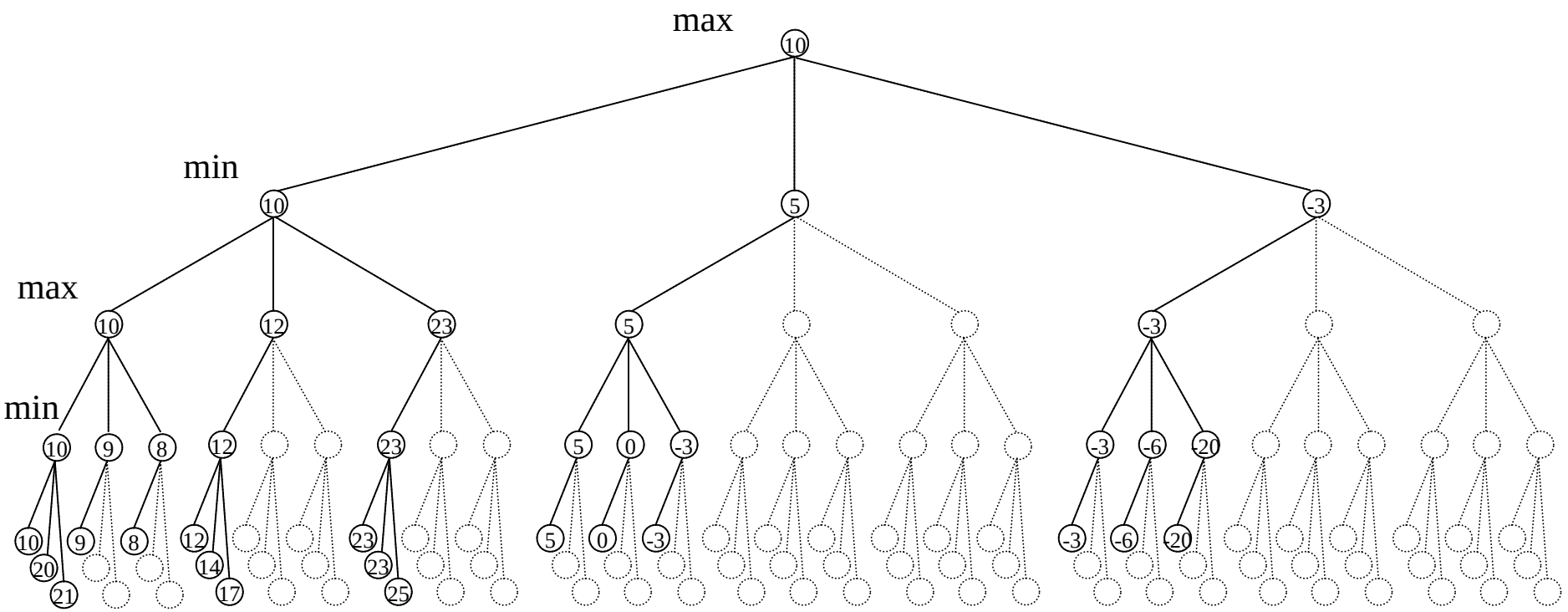
# Dragordning

- Förgreningsfaktor $w$
- Träddjup $d$
- Besökta noder:
  - värsta fallet: $w^d$
  - perfekt dragordning: $w^{ceil(d/2)} + w^{floor(d/2)}$
- Vi kan alltså söka dubbelt så djupt!

# Perfekt dragordning

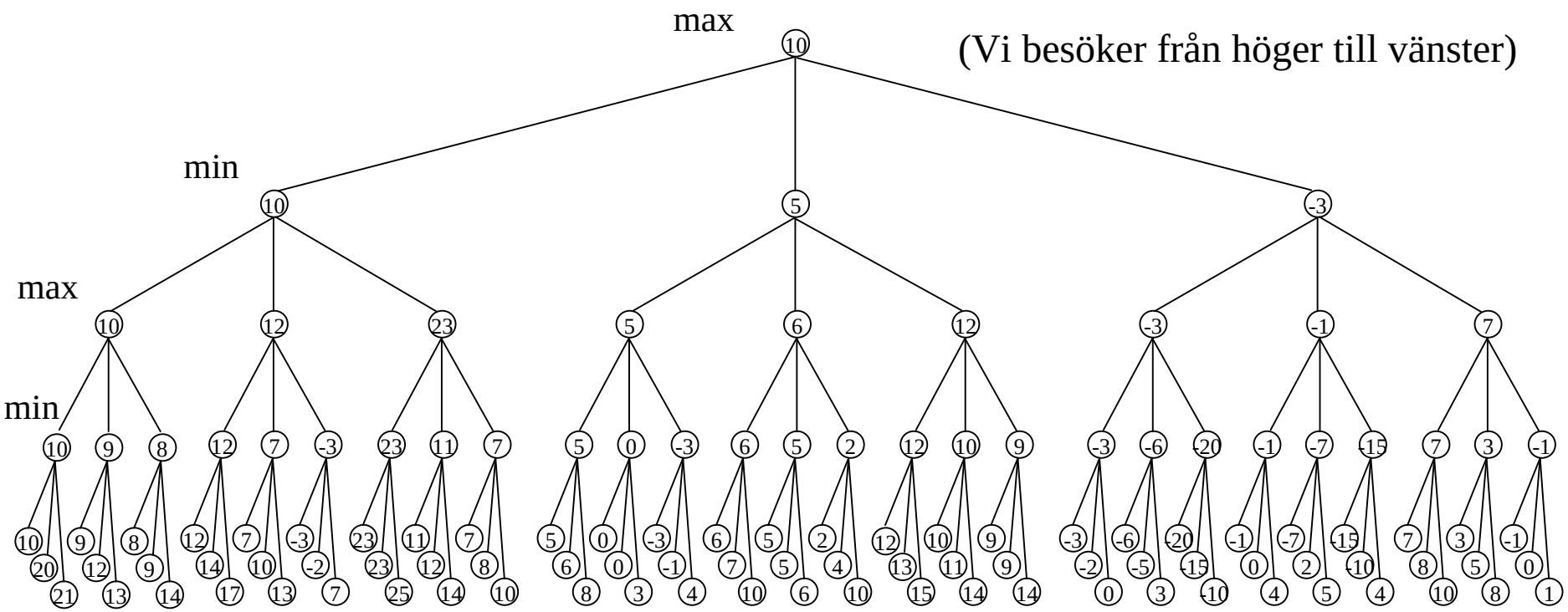


# Perfekt dragordning

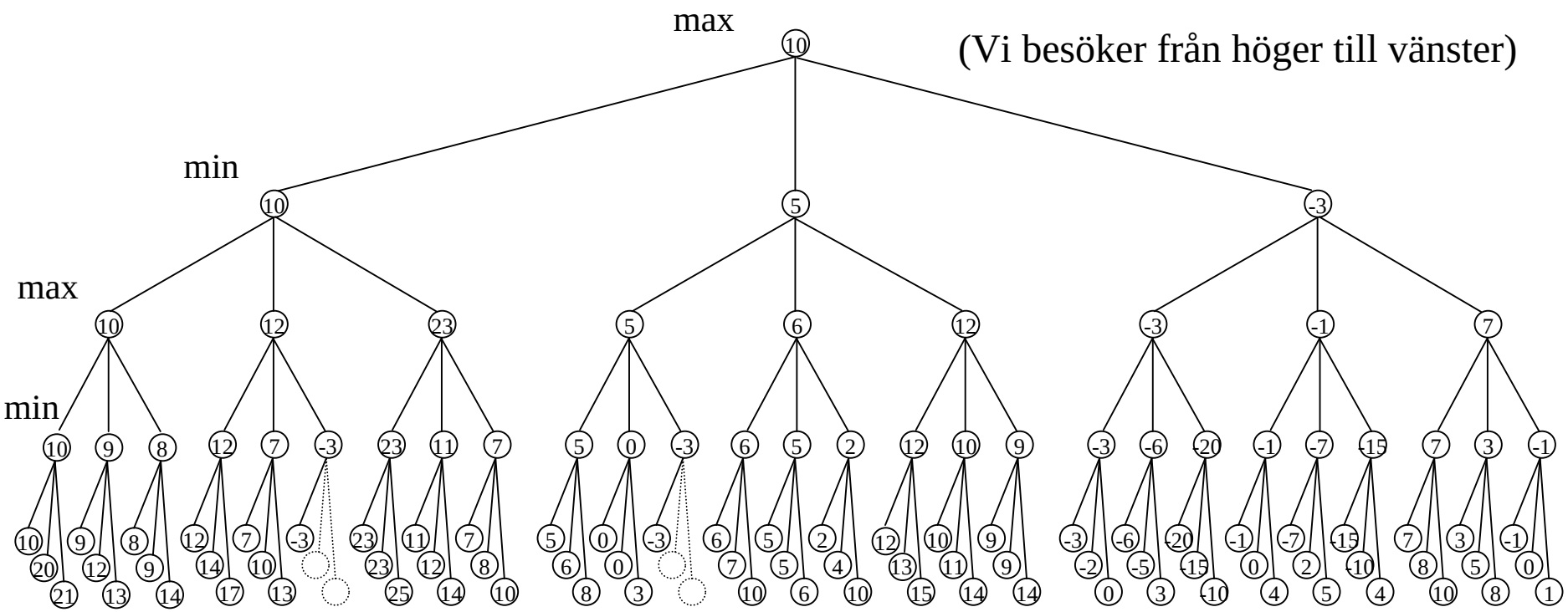



Besökta: 17, beskurna: 64

# Omvänt perfekt dragordning



# Omvänt perfekt dragordning




Besökta: 77, beskurna: 4

# Dragordning

- Med perfekt dragsortering behöver vi inte söka! Det bästa draget är ju det första.
- Man vet inte det bästa draget
- Heuristiker:
  - Grundläggande schackunskap
  - Killer move
  - Historik

# Dragordning

Intressanta drag:

- De som kan förändra poängen drastiskt
- Slå den pjäs som flyttades sist
- Övriga drag som slår en pjäs
- Bonde som blir till dam
- Schackar (problematiskt)

# Killer move

- Observation: ett drag som är bra på djup *d* i en gren är nog bra i en annan gren
- Ex: flytta damen om den är hotad
- Drag som orsakar avskärningar är bra
- Vi sparar ett eller flera drag per nivå
- Minskning av trädet: ca 30%

# Historik

- (eng: history heuristic)
- Skapa tabell med 64 x 64 heltal (från, till)
- Varje gång ett drag ger avskärning, räkna upp mosvarande räknare i tabellen
- Ger inbördes ordning mellan alla drag

# Transpositioner

- Observation: samma bräde kan uppkomma på olika sätt
- Ex: jag flyttar **kungen** och sen **tornet** eller jag flyttar **tornet** och sen **kungen**
- Om en position redan är besökt behöver den inte behandlas igen
- Vi skapar en tabell över besökta noder

# Transpositionstabell

- Istället för att spara besökta noder i trädform, spara den i en hashtabell
- Snabb uppslagning
- Varje bräde får ett hashvärde som beror på hur pjäserna står, vems tur det är etc.

# Transpositionstabell

Kollisioner:

- Efter ett tag blir tabellen full
- Många bräden hashas till samma element
- Probabilistiskt lås - ytterligare ett hashvärde per bräde

# Transpositionstabell

Sparad information:

- Hashlås (normalt 32 bitar)
- Sökdjup
- Undre gräns för värdet
- Övre gräns för värdet
- Bästa draget!

# Transpositionstabell

Om bräde finns i hash vid uppslagning

- Om djup = sparat djup, använd hashat värde:
  - Om upper < alpha, sluta sök
  - Om lower > beta, sluta sök
- Annars
  - Prova sparat drag först!

# Transpositionstabell

- Bräden i hashen: ca 1/4 av trädets storlek
- Hashade drag ger väldigt bra gissning på vilket drag som är bäst

# Utökad transpositionsbeskärning

- (eng: enhanced transposition cutoff)
- Vänta med att utföra dragen (söka djupt i trädet)
- Slå istället upp alla drag i hashtabellen
- Hittar vi **ett** som ger beskärning så kan vi sluta söka!

# Iterativ fördjupning

- Dragen i hashen är ofta bra gissning
- Vi vet inte hur mycket tid sökning på djup *n* kommer att ta
- När tiden är slut måste vi ha ett drag
- Vi söker djup 1, 2, ..., *n*!

# Iterativ fördjupning

- Onödigt arbete? Djup *n* kan ta 30 ggr längre tid än djup *n-1*
- Det går oftast snabbare att söka 1, 2, ..., *n* än att söka *n-1*, *n*!
- Hashen är fylld med användbar information

# Iterativ fördjupning

- Finns inget drag i hashen har vi ingen gissning
- Vi kan då tillämpa intern iterativ fördjupning
- Vi söker ett par nivåer för att fylla hashen med nyttig information
- Försumbart arbete om vi får en tidig beskärning

# Aspirationssökning

- Poängen i ett parti ändras oftast inte så mycket
- Med snävare ingångsvärden i alfa-beta-sökningen går sökningen snabbare
- Vi gissar på föregående värde och lägger på en bråkdel av en bonde
- Alfa-beta med fönster (a, ß) = (*v*-30, v+30)

# Aspirationssökning

Alfa-beta med fönster (a, ß) = (*v*-30, v+30)

- Får vi ett värde *w* = *v*-30 så vet vi att det rätta värdet *x* är = *w*
- Får vi ett värde *w* = *v*+30 så vet vi att det rätta värdet *x* är = *w*
- Vi måste göra en omsökning för att hitta det rätta värdet, a < *x* < ß

# Nollfönster

- En ambitiös gissning är ett s.k. nollfönster (a, ß) = ($v$-1, $v$)
- Vi kan aldrig träffa det rätta värdet, men med flera sökningar kan vi ta reda på värdet
- Vi får bara information om huruvida värdet är < $v$ eller = $v$

# NegaScout

- Idé: Vi söker bara noggrant i det första barnet i varje nod
- De övriga får frågan:
  Är ditt värde < *v* eller = *v* ?
- Är värdet mindre än *v* är allt bra
- Är värdet större måste vi söka om

# NegaScout

- Algoritmen kallas NegaScout
- Den är en förbättring av huvudvariantssökning (principal variation search, PVS)
- PVS söker noggrant på den mest troliga dragutvecklingen och mindre noggrant på andra
- NegaScout dominerar Alfa-beta-algoritmen, dvs Alfa-beta beskär inget som NegaScout besöker

```
function NegaScout(node, alpha, beta, depth)
    a = alpha;
    b = beta;
    if depth == 0 then g = evaluate(node);
    else if node == MAXNODE then
        c = firstchild(node);
        while (a < beta) and (c != NOCHILD) do
            t = NegaScout(c, a, b, depth - 1);
            if (t > a) and (t < beta) and (c != FIRST) and (depth > 1)
               a = NegaScout(c, t, beta);
            a = max(a, t);
            b = a + 1;
            c = nextbrother(c);
        return a;
    else /* node is a MINNODE */
        c = firstchild(node);
        while (b < alpha) and (c != NOCHILD) do
            t = NegaScout(c, a, b, depth - 1);
            if (t > alfa) and (t < b) and (c != FIRST) and (depth > 1)
               b = NegaScout(c, alfa, t);
            b = min(b, t);
            a = b - 1;
            c = nextbrother(c);
        return b;
```

# MTD(*f*)

- MTD(*f*) är en sökalgoritm som bara använder nollfönster
- Den anropar Alfa-beta-sökning upprepat
- Beroende på om returvärdet slår i alfa eller beta så stegar vi ett steg nedåt resp. uppåt
- Resultatet kan bli hundratals små men väldigt snabba sökningar

# MTD(*f*)

- Varför tar man då inte större steg?
- Jag har gjort lite experiment med steglängder och binärsökning
- Med nollfönster som ligger intill tidigare sökta nollfönster blir varje ny sökning liten
- Längre steg ger stor ökning i tid
- Intressant, men osäkert om det lönar sig

```
function MTDf(root, f, depth)
    g := f;
    upper := +INFINITY;
    lower := -INFINITY;
    repeat
        if (g == lower)
            beta = g + 1;
        else
            beta = g;
        g := AlphaBeta(root, beta - 1, beta, depth);
        if (g < beta)
            upper = g;
        else
            lower = g;
    until (lower >= upper)
    return g;
```

# Horizonteffekten

- (eng: horizon effect)
- Om man söker till djup *n* kan man ha otur och sluta mitt i en slagväxling
- Det kan då se ut som om man ligger en pjäs under, fast detta åtgärdas i nästa drag
- Man kan även få för sig att ens hopplöst förlorade dam går att rädda, det är bara att offra tornet och sen löparen och sen en bonde
- På så sätt så slipper man se att damen går förlorad
- Detta kallas för horizonteffekten

# Stabil sökning och horizonteffekten

- (eng: quiescense search)
- Man bör därför bara utvärdera stabila positioner (eng: quiet)
- Exempel på instabila positioner:
  - Slagväxlingar
  - Kungen i schack
  - Bonde till sista raden
- Man fortsätter söka tills positionen är stabil och utvärderar först då

# Selektiv utvidgning

- (eng: selective extensions)
- En människa spelar inte schack på samma sätt som en dator
  - En människa söker djupt på lovande positioner
  - En dator provar alla drag till djup *n*
- För att förbättra datorns spel kan man utvidga lovande stigar

# Kostnad och bråkdelsdjup

- (eng: fractional depth)
- Ett sätt att utvidga en stig är att tilldela en kostnad/belöning till egenskaper som uppkommer utmed stigen
- Ex:
  - Drag med bara ett motdrag, sök +0.5 djupare
  - Hot mot tung pjäs, sök +0.25 djupare
  - Bonde på sjunde raden, sök +0.5 djupare
- Summan kan ge ytterligare djup

# Nolldrag

- (eng: null move heuristic)
- För att minska sökträdets storlek kan man ta till potentiellt farliga heuristiker
- Idé: Om motståndaren får göra tre drag på rad och ändå inte leder, sluta sök (eller sök -2.0 mindre djup etc.)
- Farligt: man kan missa djupa kombinationer
- Farligt: det finns situationer där det är bra att inte vara vid draget (Zugzwang)

# Hopplöshet

- (eng: futility pruning)
- Om följande kriterier är uppfyllda:
  - Situationen är stabil och nästa drag slår inte en pjäs
  - Motståndaren ligger minst en lätt pjäs efter
- Isåfall, utvärdera inte nästa drag
- Man spar stora delar av sökdjup *n*
- Razoring - ungefär: om motståndaren ligger en dam efter och bara två drag till horizonten, sluta sök
- Mycket farligt, många besparingar

# Öppningsbok

- (eng: opening book)
- Datorn har dålig positionell känsla
- Svårt att starta spelet på ett vettigt sätt
- Lösning: öppningsbok med acceptabla drag
- Att ladda ned: ECO - vanligaste öppningarna

# Slutspelstabeller

- (eng: end game table bases, EGTB)
- Slutspelet har få pjäser och mycket positionell karaktär
- Man har spelat samtliga spel med upp till fem pjäser (några med sex) och sparat till disk!
- Begränsad användbarhet (6 pjäser = många terabyte)

# Slut

- Moderna schackdatorer innehåller det mesta av det vi berört och mycket därtill
- Ladda ner och prova Rainman:

  **`http://www.nada.kth.se/~johnny/chess_comp.html`**