Artificial Intelligence
Chapter-6

Adversarial Search

By : J.Razmara
Azarbaijan University

جستجوی رقابتی

- محیط چند عاملی Multi agent Environment
  - تلاش برای رسیدن به هدف در میان عاملهای دیگر با اعمال مخالف
- مثال: تئوری بازیها
  - بازیهای دو یاچند نفره

جستجوی رقابتی

- دلیل بررسی بازیها
  - ماهیت جذاب و سرگرم کننده بازیها
  - سادگی مدلسازی بازیها در فضای وضعیتها و مجموعه ای از عملیات
  - معیارهای عملی و واضح برای ارزیابی میزان موفقیت AI
  - مطالعه و بررسی عاملهای رقیب و دشمن
  - وجود بازیهای سخت و جالب با فضای حالتهای وسیع
    - بازی شطرنج با درخت جستجو شامل $۳۵^{۱۰۰}$ گره

تئوری بازیها

- بازیهای کامل Perfect games
  - بازیهای دو نفره
  - بازیکنان به نوبت بازی می کنند.
  - قانون مجموع صفر (Zero-sum) : اشتباه یکی به نفع دیگری است.
  - هر بازیکن اطلاعات کامل در مورد محیط و حالت بازی دارد و هیچ اطلاعی از دید بازیکن مخفی نیست.(محیط دسترس پذیر)
  - عناصر شانس (استفاده از تاس، ...) دخالت ندارند.
  - حرکات بطور واضح و مشخص تعریف شده اند.
  - هر حرکتی نتیجه مشخصی دارد.(محیط قطعی)
  - مثال: شطرنج، Tic-Tac-Toe، ...
- بازیهای غیر کامل Imperfect games
  - مثال: تخته نرد، فال ورق، ...

تئوری بازیها

- یک بازی با اجزای زیر قابل تعریف است:
  - حالت شروع شامل وضعیت اولیه به همراه بازیکنی که باید بازی را شروع کند.
  - یک تابع (Succesor function) که لیستی شامل زوجهای (*move*, *state*) متشکل از حرکات قانونی قابل اجرا و نتیجه هر حرکت را ارائه می دهد.
  - تابع تست هدف که خاتمه بازی را شناسائی می کند.
  - یک تابع کمکی (Utility/Objective function) که امتیازات دو طرف بازی را ارزیابی کند. (مثل Zero-Sum)

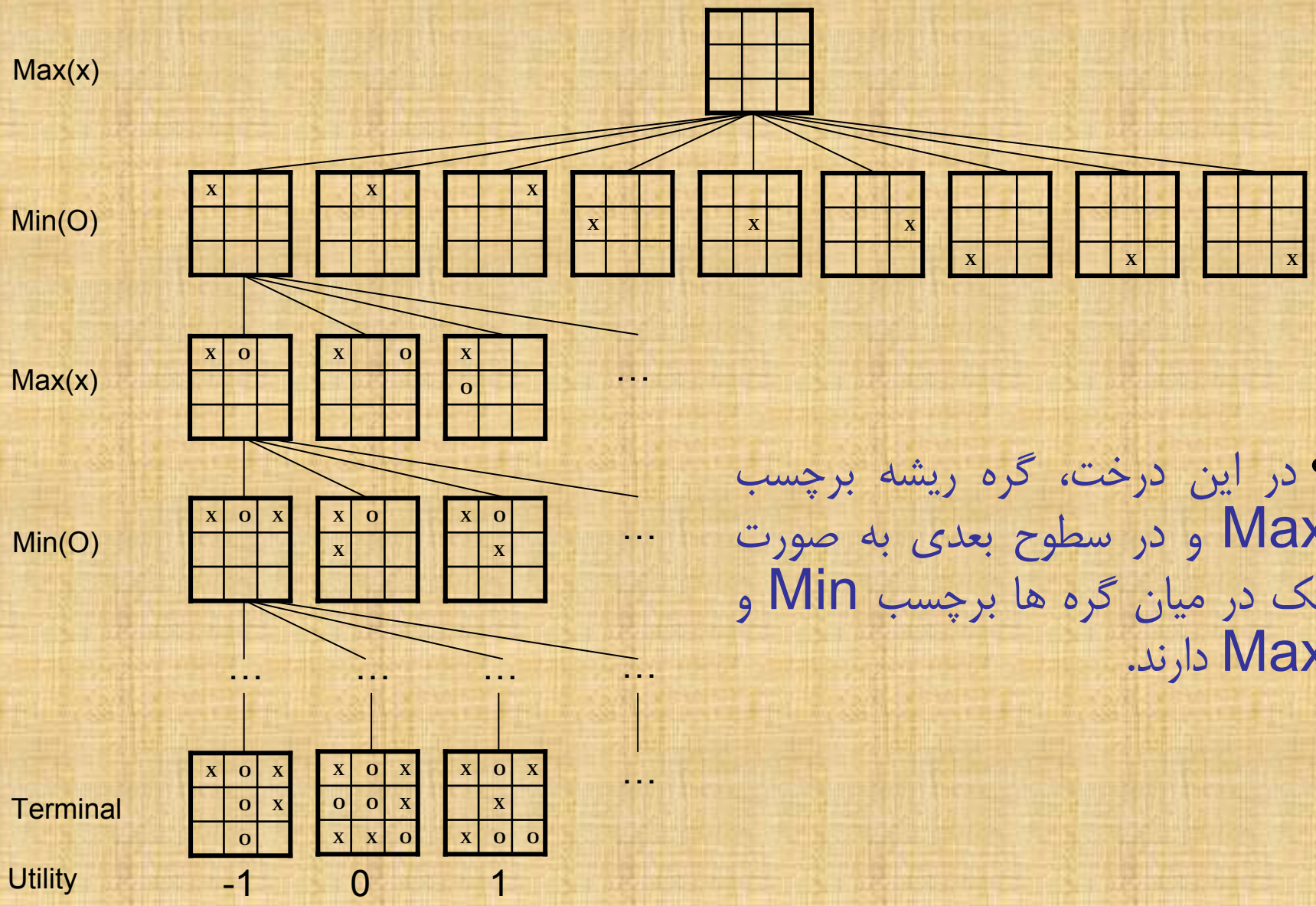
تئوری بازیها

- برای انجام بازی:
  - تمام حرکات قانونی ممکن را لیست کن.
  - حالت بعدی حاصل از انجام هر یک از حرکات فوق را مشخص کن.
  - هر یک از حالتهای بعدی ممکن را ارزیابی کرده و بهترین حرکت را انتخاب کن.
  - حرکت انتخاب شده را انجام بده.
  - صبر کن حریف حرکتش را انجام دهد، مراحل بالا را دوباره تکرار کن.

درخت بازی

- استفاده از درخت برای نمایش فضای مسئله برای بازیها اغلب مناسب است.
  - گره ریشه شامل حالت شروع بازی می باشد.
  - برای هر گره شامل وضعیت جاری، باید تصمیمی برای انتخاب بهترین حرکت بعدی اتخاذ شود.
  - هر حرکت قانونی توسط یک شاخه از درخت نشان داده می شود.
  - با استفاده از یک تابع ارزیابی، یک وضعیت از بازی ارزش گذاری می شود.
  - گره های برگ، وضعیت های نهائی بازی را نشان می دهند که در اینجا می تواند یکی از مقادیر برد، مساوی و یا باخت باشد.

درخت بازی Tic-Tac-Toe



- در این درخت، گره ریشه برچسب Max و در سطوح بعدی به صورت یک در میان گره ها برچسب Min و Max دارند.

تابع ارزیابی

- به منظور ارزیابی میزان خوب بودن یـک گـره، تـابع ارزیـابی استفاده می شود. (تابع هیوریستیک)
- با استفاده از قانون مجموع صفر (Zero-Sum) می توان از یک تابع برای هر دو بازیکن به منظور ارزیابی موقعیت بازی بهره برد:
  - $F(n)>0$: موقعیت n برای من خوب و برای حریف بد است.
  - $F(n)<0$: موقعیت n برای من بد و برای حریف خوب است.
  - $F(n)=0$: موقعیت n یک وضعیت خنثی است.
  - $F(n)>>0$: برد من
  - $F(n)<<0$: برد حریف

تابع ارزیابی

- تابع ارزیابی برای بازی Tic-Tac-Toe:

f(n)=[Number of 3-lengths open for me] –
[Number of 3-lengths open for you]

- تابع ارزیابی در بازی شطرنج اغلب به صورت مجموع وزن دار از امتیازات موقعیت مهره ها بیان می شود:

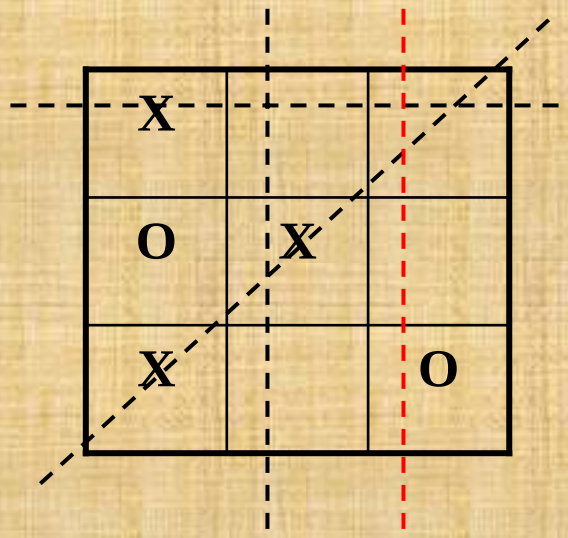
$$f(n) = w_1 * feat_1(n) + w_2 * feat_2(n) + \ldots + w_k * feat_k(n)$$

امتیازات موقعیت مهره ها مانند تعداد مهره ها، نحوه چیدن مهره ها، تعداد خانه های تحت پوشش و ... می باشد.

تابع ارزیابی

- مثال: تابع ارزیابی برای بازی Tic-Tac-Toe:

f(n)=[Number of 3-lengths open for me] –
[Number of 3-lengths open for you]



3 – 1 = 2

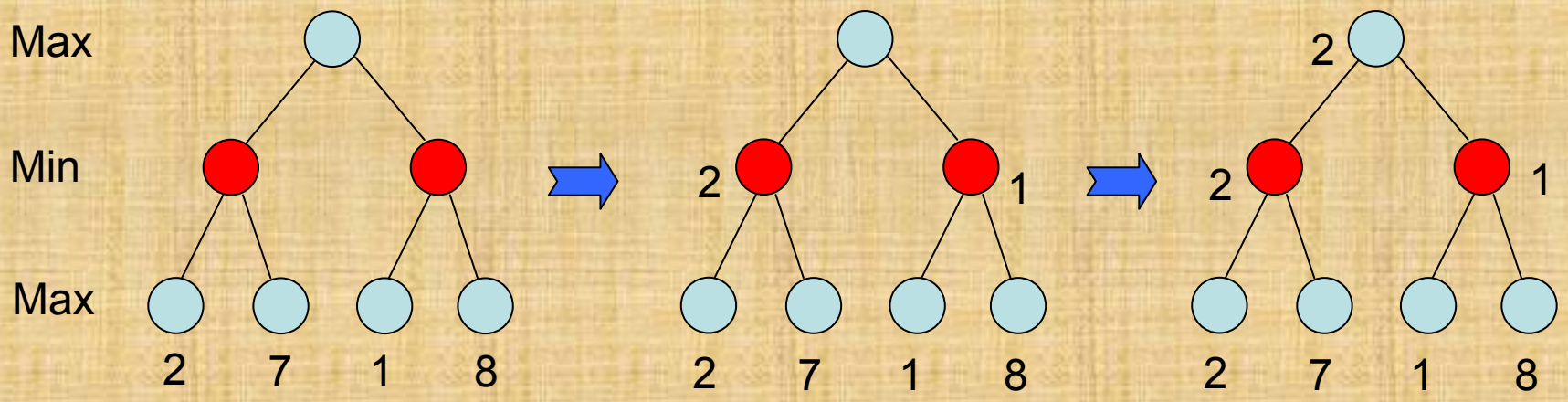
قانون Minimax

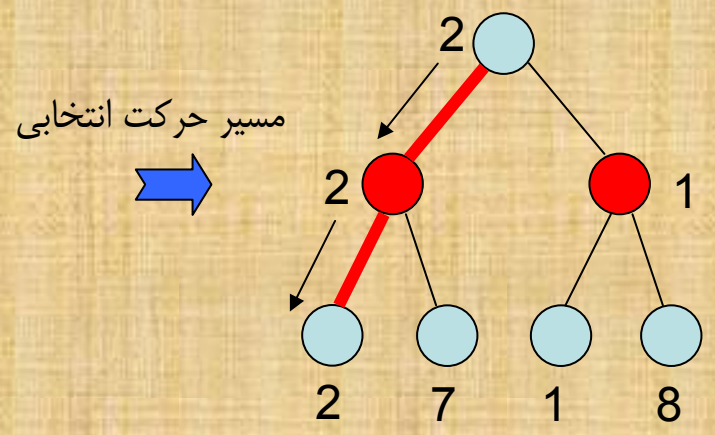
- قانون Minimax : هدف از جستجو در درخت بازی، تعیین حرکتی برای بازیکن Max است طوریکه امتیاز او را (بدون توجه به حرکات Min) ماکزیمم کند.
- بازیکن Max همواره فرض می کند که بازیکن Min حرکتی را انجام می دهد که امتیازش را ماکزیمم کند و امتیاز Max را مینیمم کند.
- امتیاز هر گره از طریق امتیاز فرزندانش مشخص می شود:
  - مقدار گره Max برابر ماکزیمم مقدار فرزندانش است.
  - مقدار گره Min برابر مینیمم مقدار فرزندانش است.

الگوریتم Minimax

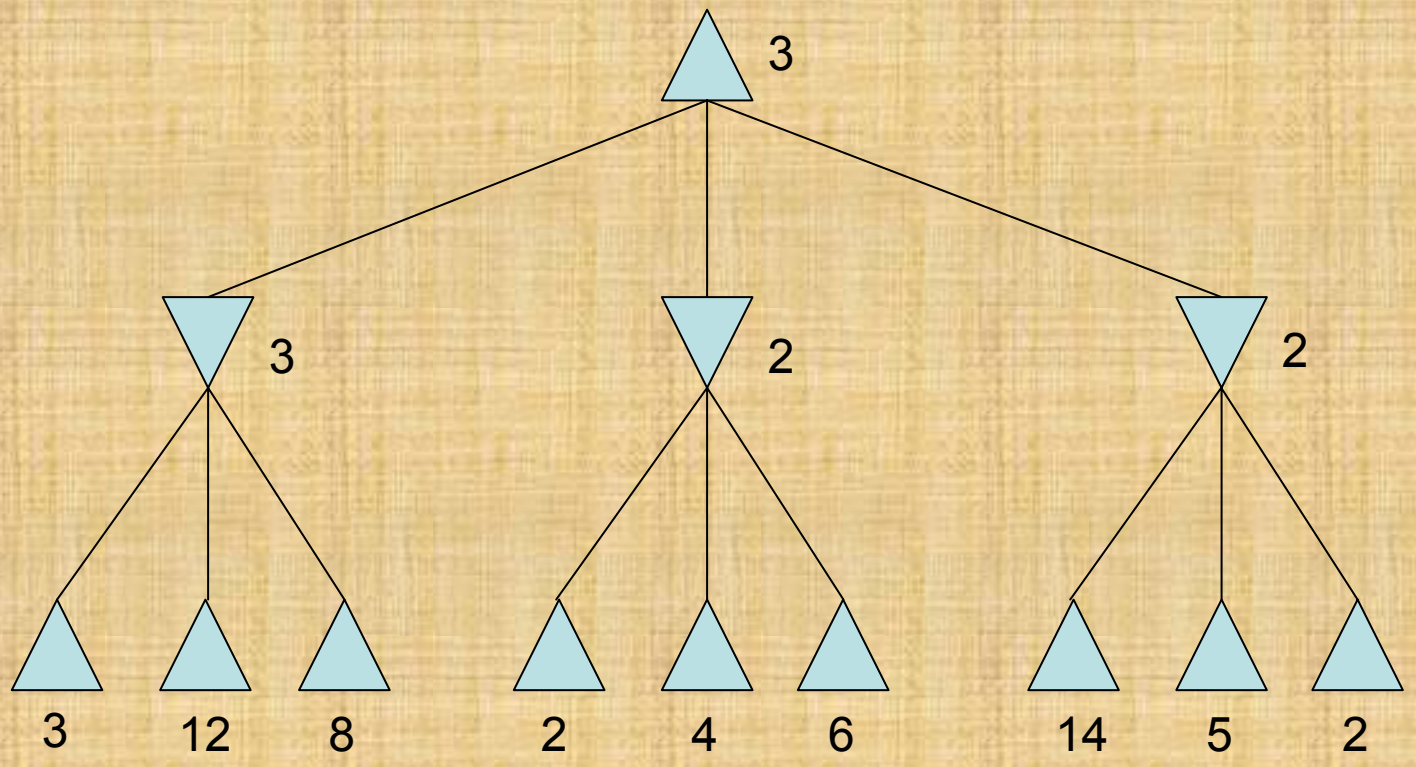
- گره شروع بازی را با برچسب Max ایجاد کن.
- گره های سطوح بعدی را تا یک حد معین بسط بده.
- تابع ارزیابی را به ازای هر گره برگ محاسبه کن.
- برای گره های غیر برگ تا رسیدن به گره ریشه، با استفاده از قانون Minimax یک مقدار انتساب بده:
  - مقدار گره Max برابر ماکزیمم مقدار فرزندانش است.
  - مقدار گره Min برابر مینیمم مقدار فرزندانش است.
- حرکتی را از گره جاری انتخاب کن که به بهترین گره برگ با ماکزیمم مقدار برسد.

جستجوی Minimax





جستجوی Minimax



Minimax الگوریتم

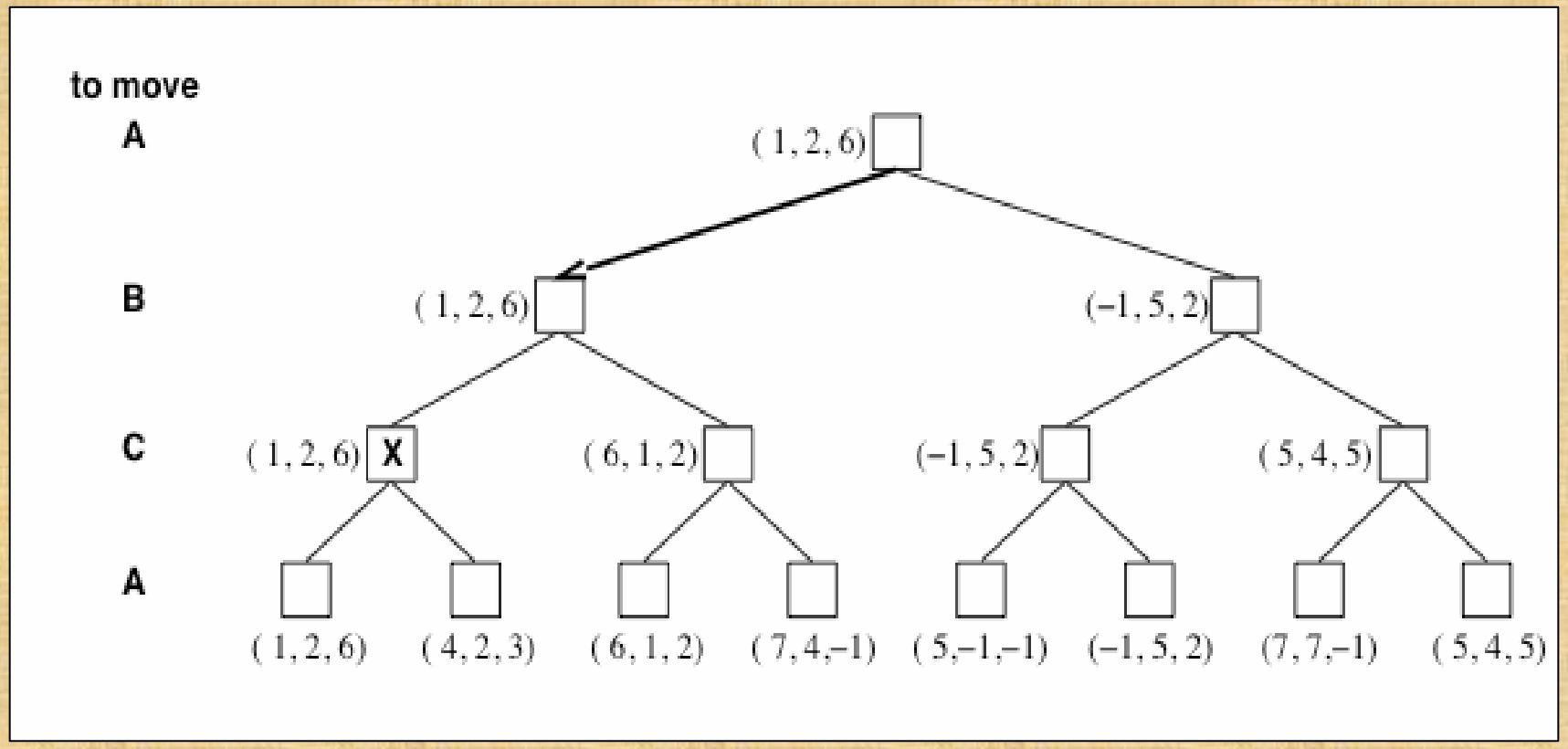
```
function MINIMAX-DECISION (state) returns an action
   inputs: state, current state in game
   v ←MAX-VALUE (state)
   return the action in SUCCESSORS (state) with value v

function MAX-VALUE (state) returns a utility value
   if TERMINAL-TEST (state) then return UTILITY (state)
   v ← ∞
   for a, s in SUCCESSORS (state) do
      v ← MAX (v, MIN-VALUE (s))
   return v

function MIN-VALUE (state) returns a utility value
   if TERMINAL-TEST (state) then return UTILITY (state)
   v ← -∞
   for a, s in SUCCESSORS (state) do
      v ← MIN (v, MAX-VALUE (s))
   return v
```

جستجوی Minimax

- بازیهای چند نفره:
  - استفاده از یک بردار برای نمایش وضعیت بازی
  - بازیکنان بطور رسمی یا غیر رسمی با همدیگر در پیشبرد بازی همکاری می کنند.

جستجوی Minimax

- مثال: درخت Minimax برای بازی سه نفره

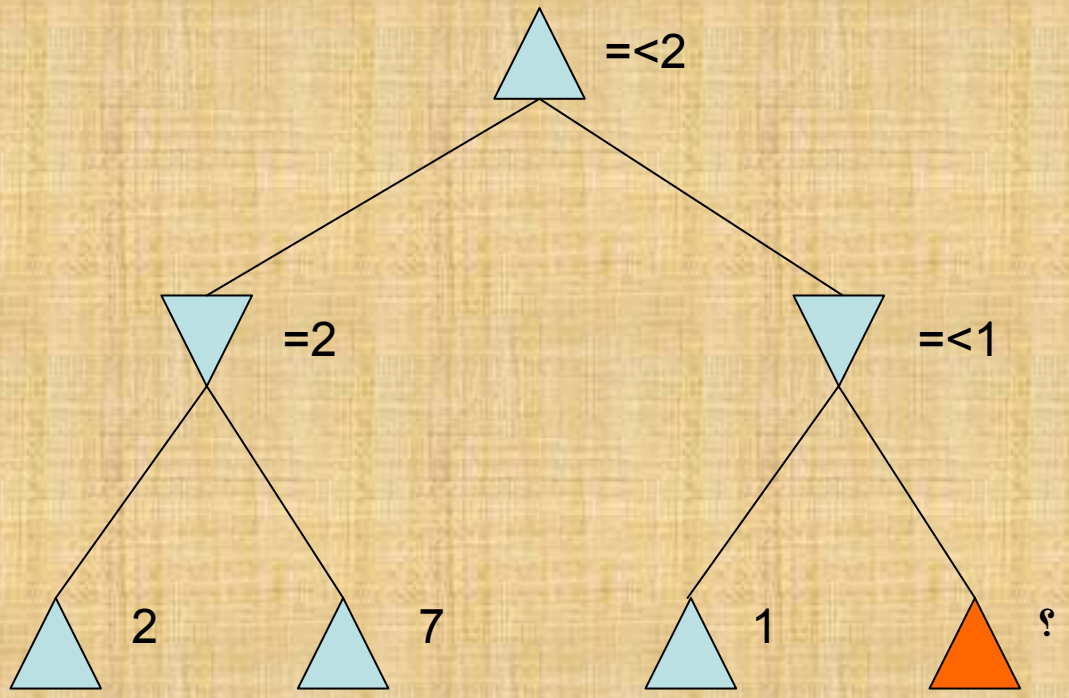


Alpha-Beta Pruning

- در جستجوی Minimax تعداد گره های درخت جستجو به صورت نمائی افزایش می یابد.
- برای حل مشکل، راه حل پیشنهادی عدم بسط گره هایی است که مقدار آنها تاثیری در تصمیم نهائی ندارد.
- برای این منظور، کارآیی الگوریتم Minimax را می توان با هرس Alpha-Beta افزایش داد.

Alpha-Beta Pruning

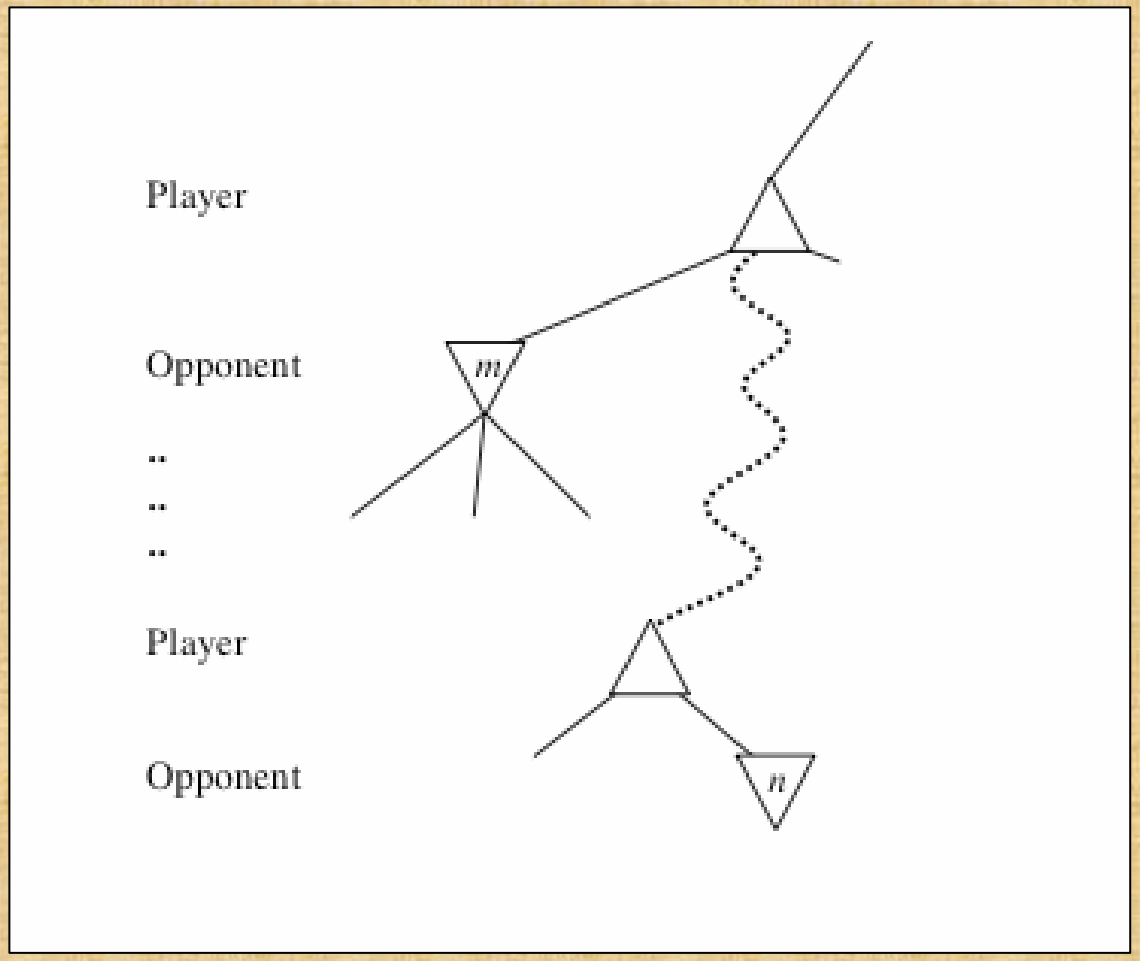
- در این روش، گره هایی که نیازی به مقدار آنها وجود ندارد را محاسبه نمی کنیم.



Alpha-Beta Pruning

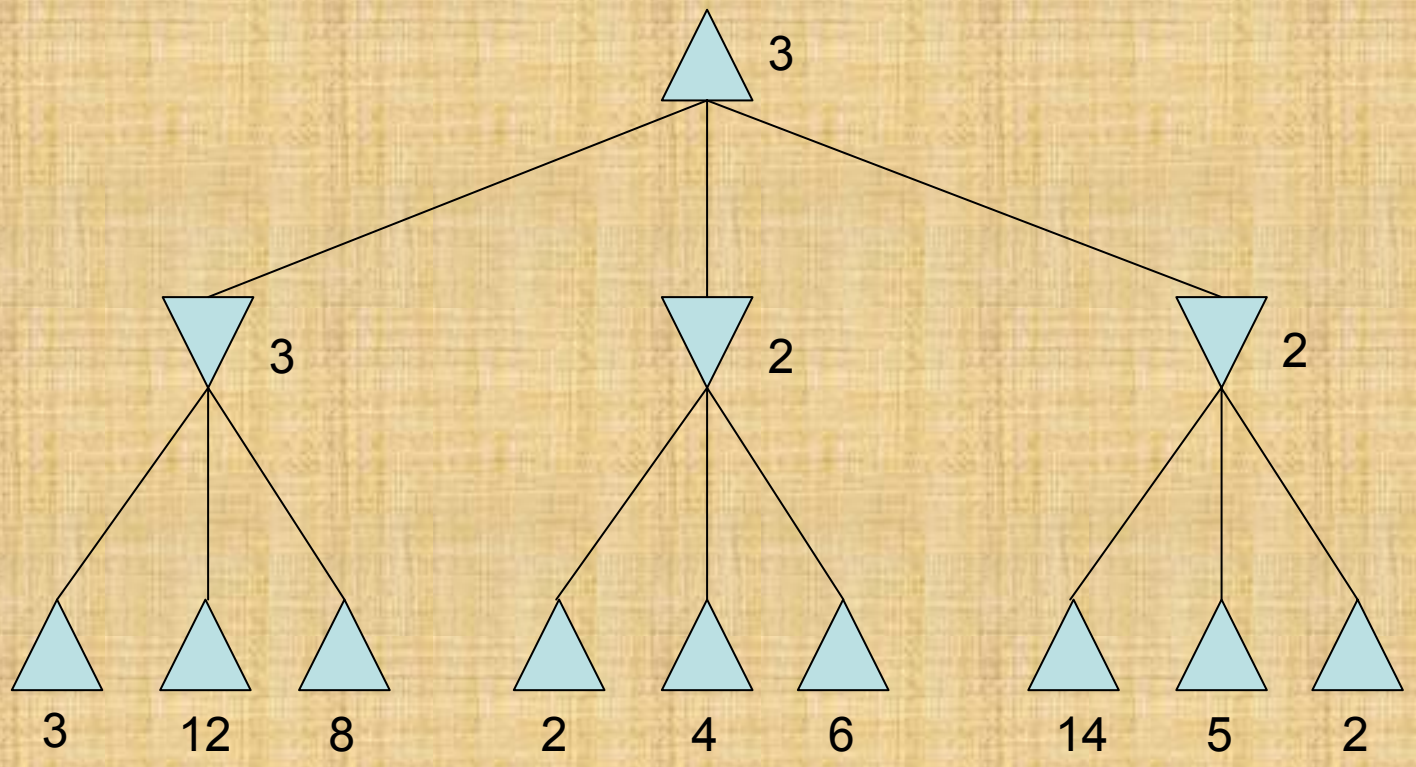
- الگوریتم :
  - درخت را به صورت اول عمق پیمایش کن
  - برای هر گره داخلی در ابتدا $\alpha = -\infty$ و $\beta = +\infty$
  - در هر گره n با برچسب Max، مقدار $\alpha(n)$ برابر ماکزیمم مقدار فرزندانش است.
  - در هر گره n با برچسب Min، مقدار $\beta(n)$ برابر مینیمم مقدار فرزندانش است.
  - مقدار α به عنوان حد پائین و مقدار β به عنوان حد بالا بکار می رود.
  - هرس در صورتی انجام می گیرد که $\alpha(n) >= \beta(n)$ شود، در اینصورت جستجو در بین فرزندان گره n متوقف می شود.

Alpha-Beta Pruning

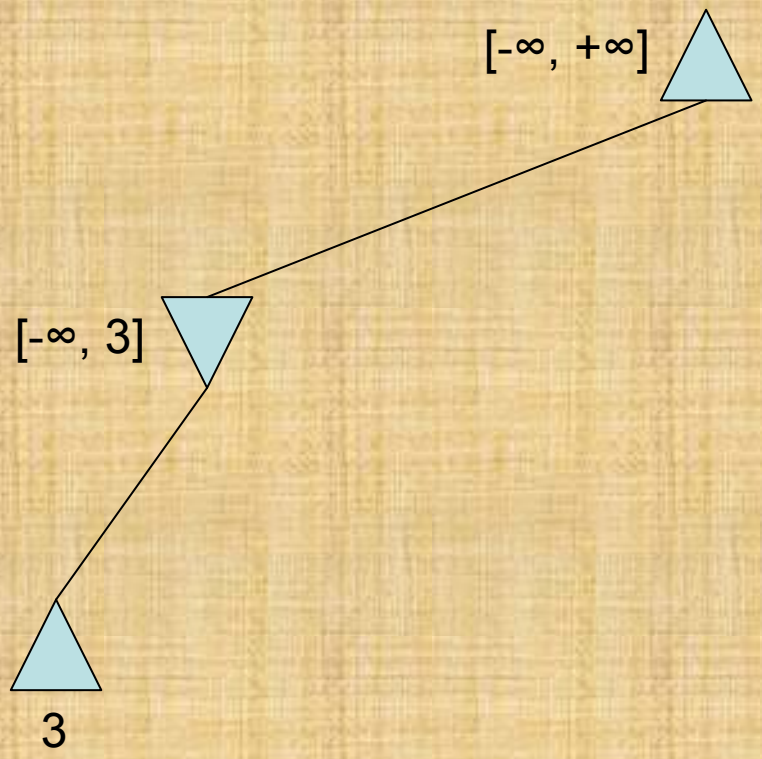


- در حالت کلی در صورتیکه وضعیت m از دید بازیکن بهتر از وضعیت n باشد هرگز سراغ مسیر n نرفته و به عبارتی گره n هرس می شود.

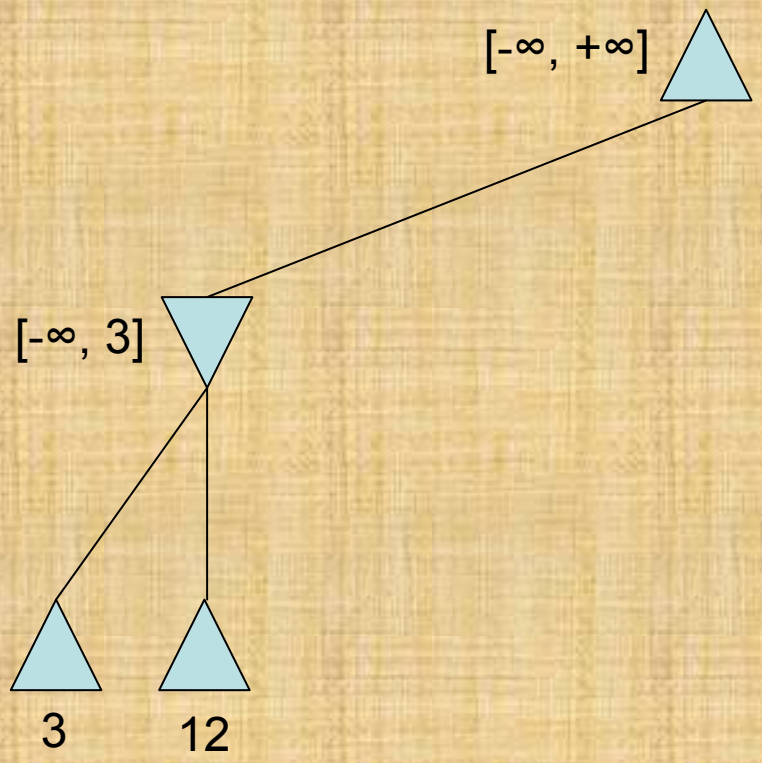
Alpha-Beta Pruning



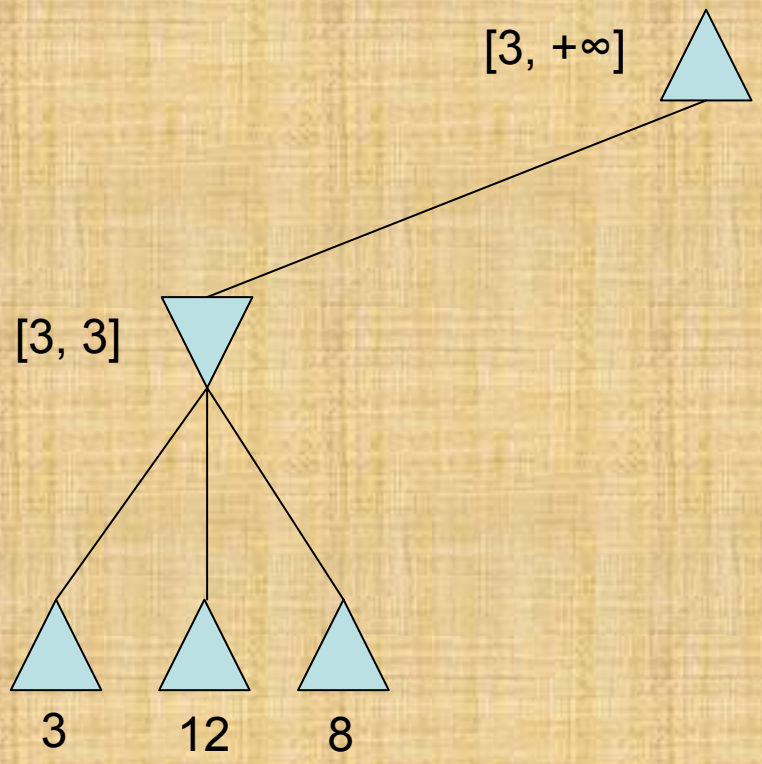
Alpha-Beta Pruning



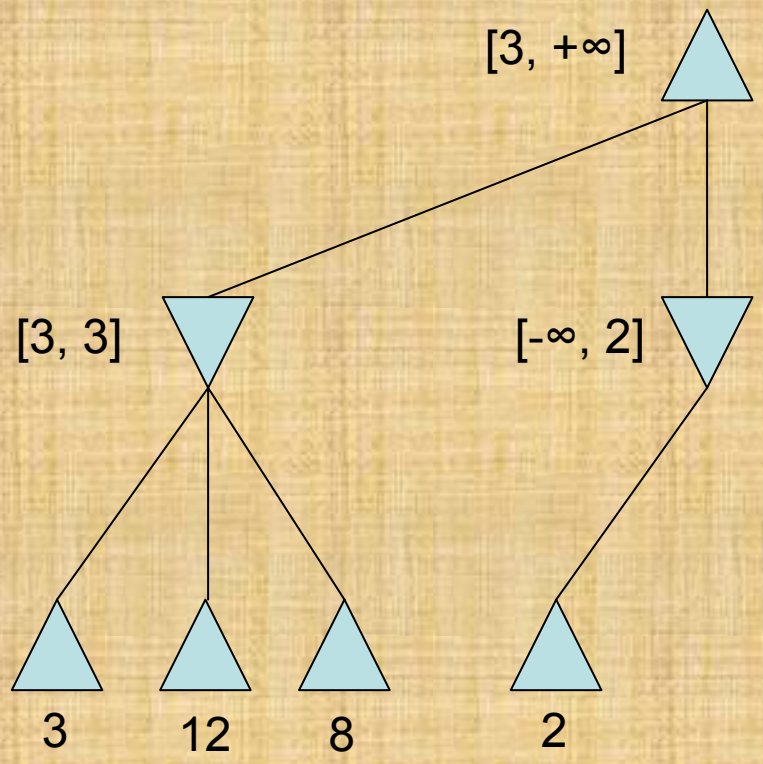
Alpha-Beta Pruning



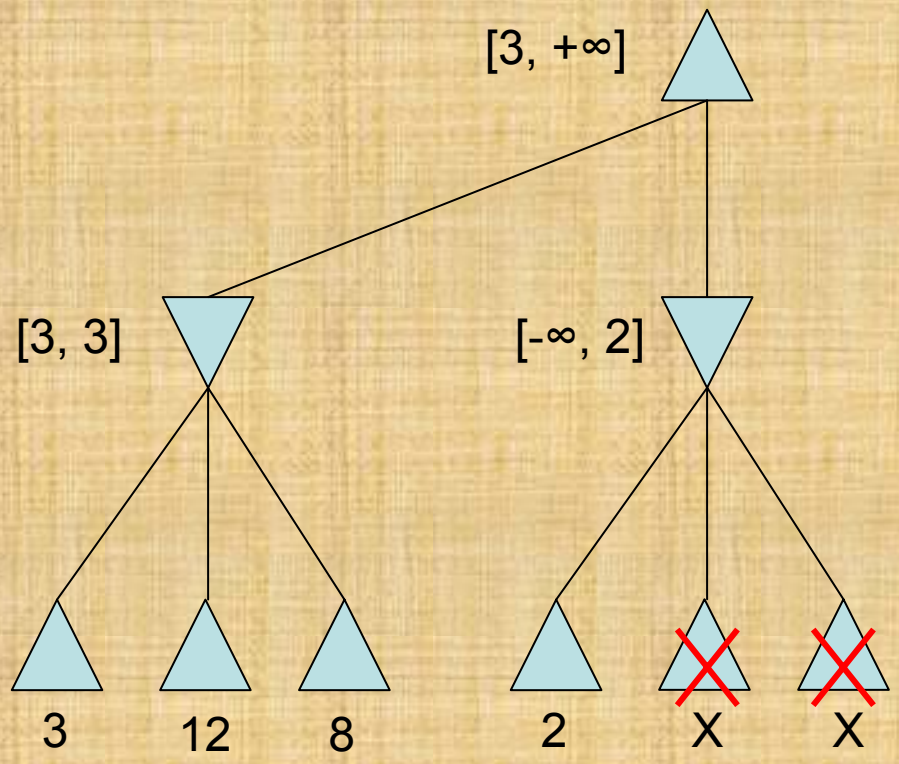
Alpha-Beta Pruning



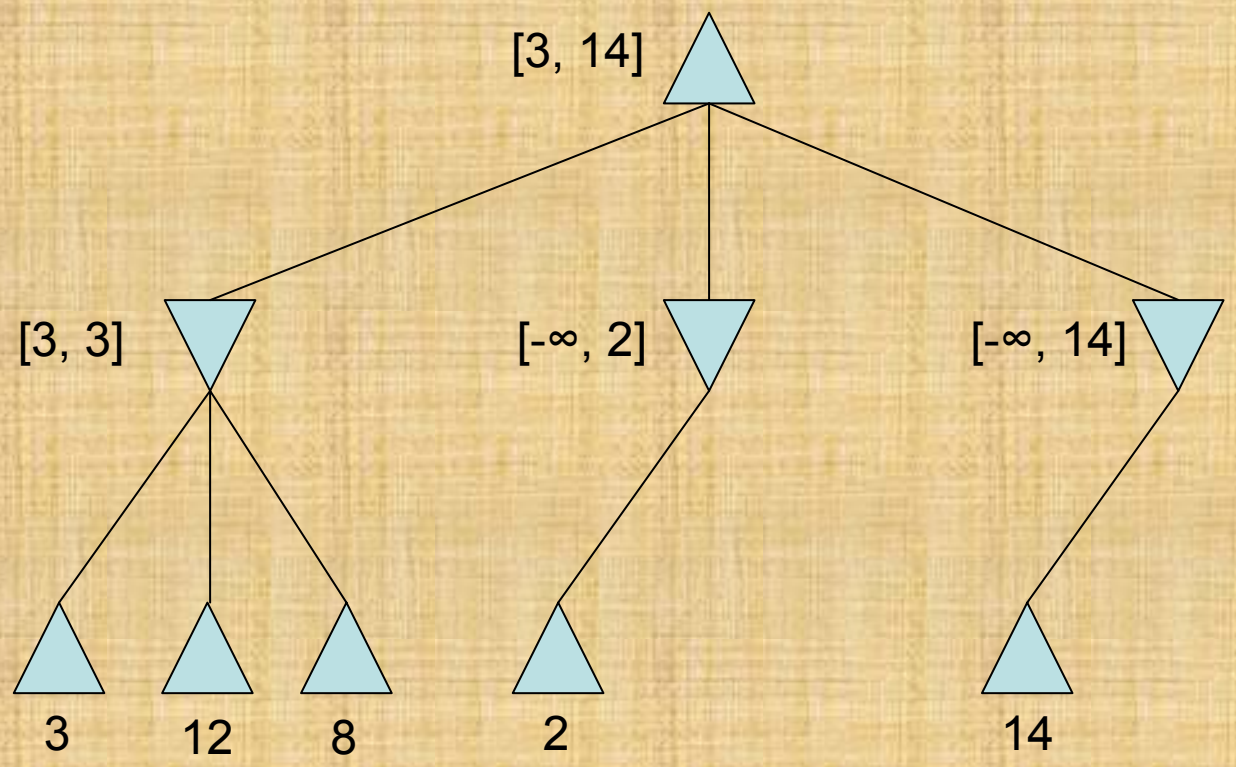
Alpha-Beta Pruning



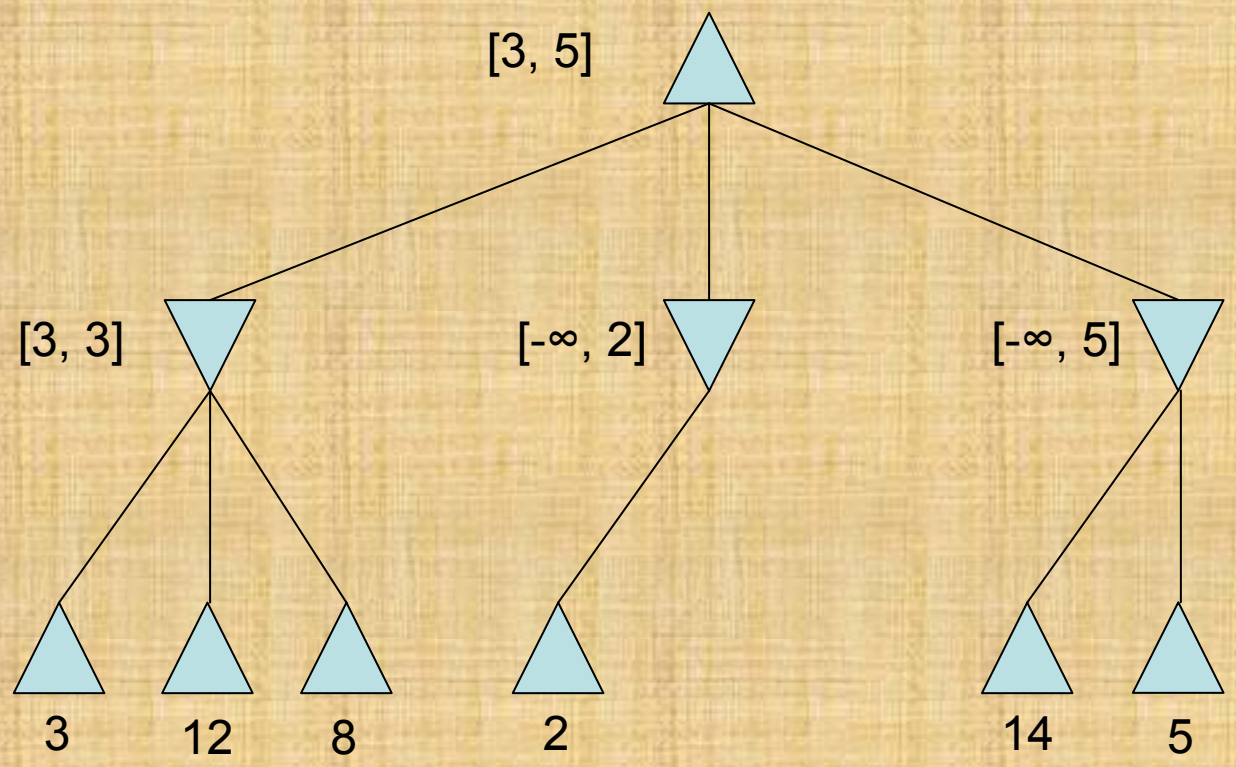
Alpha-Beta Pruning



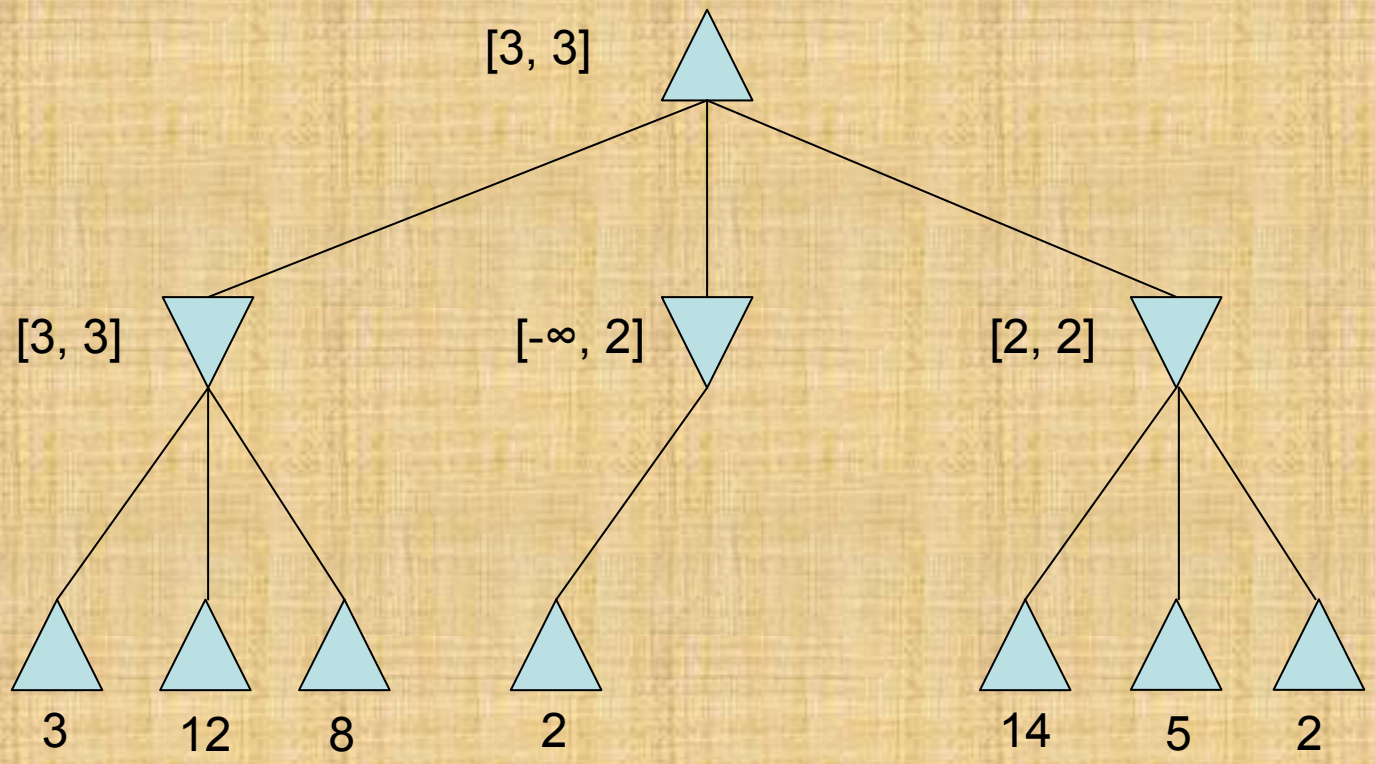
Alpha-Beta Pruning



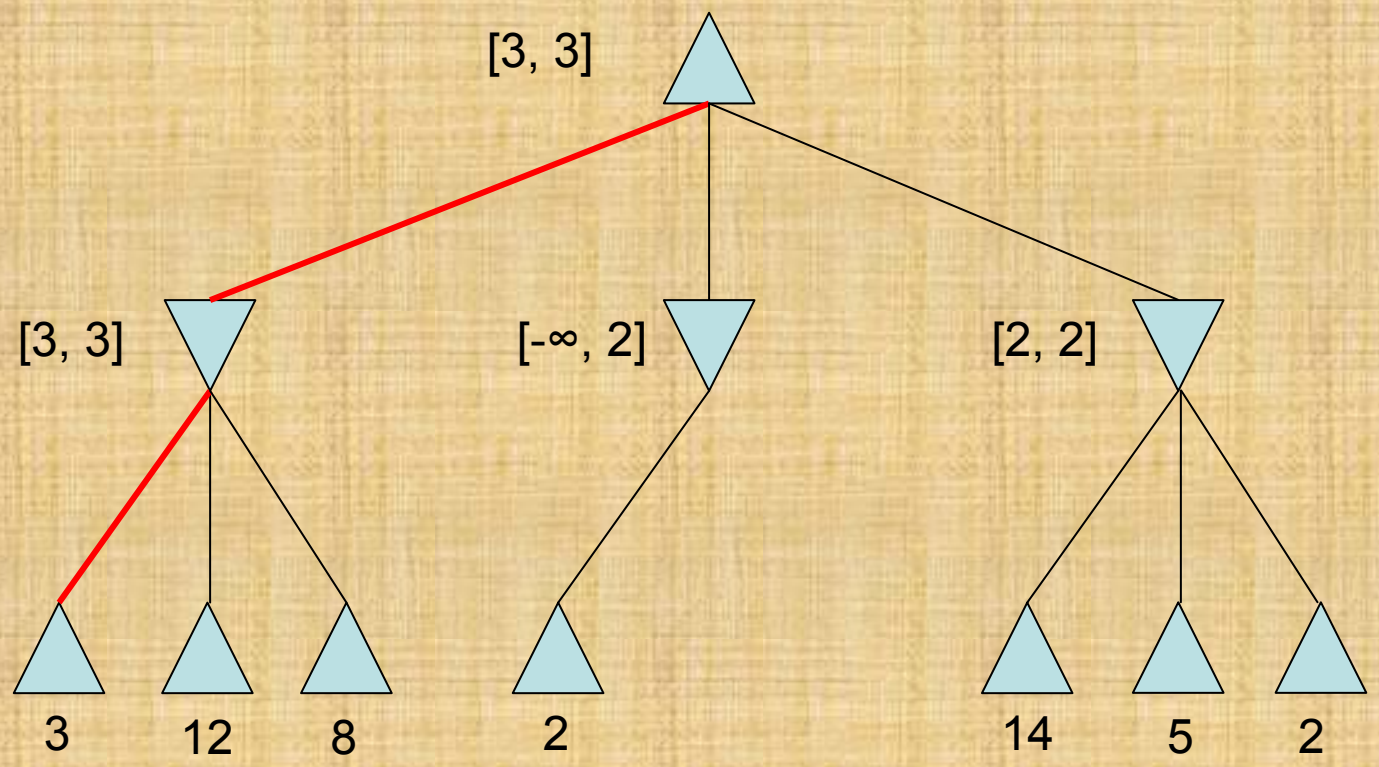
Alpha-Beta Pruning



Alpha-Beta Pruning



Alpha-Beta Pruning



Alpha-Beta Pruning

```
function ALPHA-BETA-SEARCH (state) returns an action
    inputs: state, current state in game
    v ← MAX-VALUE (state, -∞, +∞)
    return the action in SUCCESSORS (state) with value v

function MAX-VALUE (state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for max along the path to state
            β, the value of the best alternative for min along the path to state
    if TERMINAL-TEST (state) then return UTILITY (state)
    v ← -∞
    for a, s in SUCCESSORS (state) do
        v ← MAX (v, MIN-VALUE (s, α, β))
        if v >= β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE (state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for max along the path to state
            β, the value of the best alternative for min along the path to state
    if TERMINAL-TEST (state) then return UTILITY (state)
    v ← +∞
    for a, s in SUCCESSORS (state) do
        v ← MIN (v, MAX-VALUE (s, α, β))
        if v <= α then return v
        β ← MAX(β, v)
    return v
```

Alpha-Beta Pruning

- ویژگیهای هرس آلفا – بتا:
  - در بهترین حالت $O(b^{d/2})$ گره مورد بررسی قرار می گیرد.
  - بهترین حالت زمانی رخ می دهد که بهترین حرکت هر بازیکن، سمت چپ ترین حرکت در درخت بازی باشد. به عبارت دیگر در گره های Max فرزند سمت چپ بزرگترین مقدار و در گره های Min فرزند سمت چپ کمترین مقدار را دارا باشد.
  - فاکتور انشعاب مؤثر به جای b برابر با جذر b خواهد بود.

تصمیم گیری ناقص و بلادرنگ

- معایب روشهای بررسی شده:
  - روش Minimax تمامی گره های فضای جستجو را تولید کرده و بررسی می کند.
  - روش Alpha-Beta اگر چه بخشی از گره های فضای جستجو را هرس می کند ولی هنوز لازم است مسیرهای منتهی به حالتهای پایانی را برای درخت جستجو طی کند. لازمه اینکار صرف زمان زیاد است.
- شانون (۱۹۵۰) برای پیاده سازی برنامه بازی شطرنج پیشنهاد کرد:

برای کاهش زمان جستجو، تابع ارزیابی هیوریستیک بر روی گره ها اعمال شود تا گره های میانی درخت سریعتر به گره های نهائی راه یابد.

تصمیم گیری ناقص و بلادرنگ

- برای بهبود دو روش Minimax و Alpha-Beta دو تابع زیر جایگزین می شود:
  - تابع EVAL جایگزین تابع UTILITY می شود. این تابع تخمینی از وضعیت جاری طرفین بازی را مشابه توابع هیوریستیک که پیش از این بررسی شد ارائه می دهد.
  - تابع cutoff-test جایگزین تابع terminal-test می شود. این تابع تصمیم می گیرد کی تابع EVAL اعمال شود.

تابع ارزیابی

- تفاوت با توابع هیوریستیک:
  - توابع هیوریستیک تخمینی از فاصله جاری تا گره هدف را مشخص می کنند.
  - تابع ارزیابی تخمینی از میزان سودمندی وضعیت جاری را مشخص می کند.
- تابع ارزیابی خصوصیات مختلفی از وضعیت جاری بازی را مورد بررسی و محاسبه قرار می دهند. برای این منظور اغلب مقادیر و امتیازات مختلفی برای هر خصوصیت تعریف و محاسبه شده و در نهایت باهم ترکیب می شوند.
- تابع ارزیابی توانائی تشخیص حالات بعدی را ندارد و تنها می تواند مقداری محاسبه کند که میزان سودمندی آن وضعیت است.

تابع ارزیابی

- مثال: در بازی شطرنج تابع EVAL به صورت زیر محاسبه می شود:

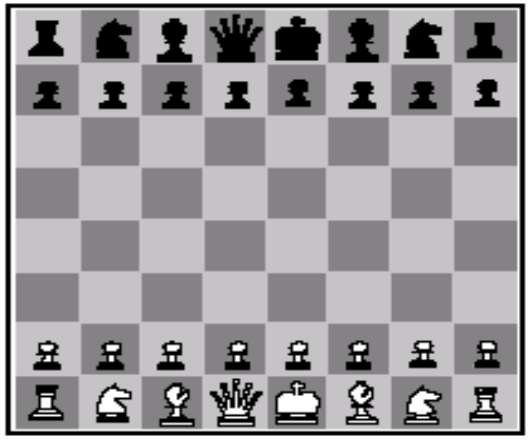
$$EVAL(n) = w_1 * f_1(n) + w_2 * f_2(n) + \ldots + w_k * f_k(n)$$

که در آن :

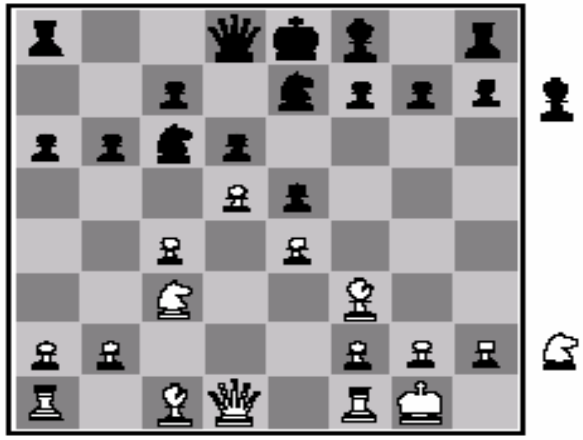
f_i : تعداد هر نوع قطعه در صفحه بازی

w_i : امتیاز آن قطعات (1 برای مهره پیاده، 3 برای مهره اسب و فیل، 5 برای مهره رخ، ...)

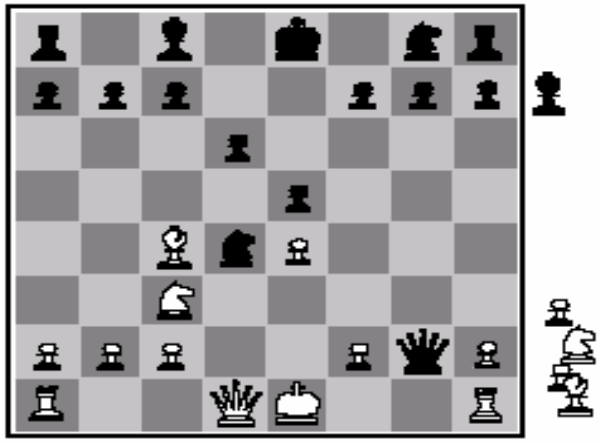
تابع ارزیابی



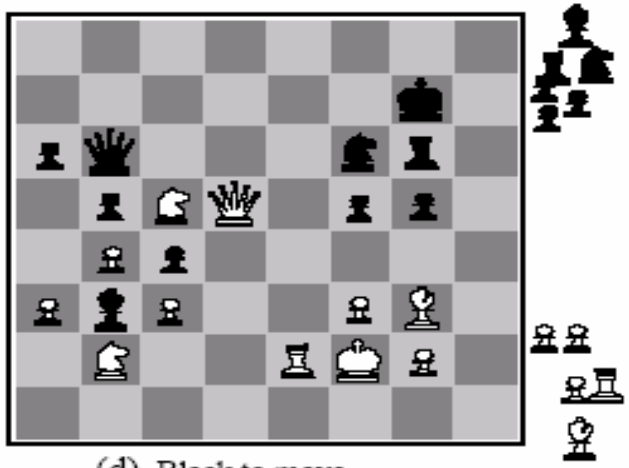
(a) White to move
Fairly even



(b) Black to move
White slightly better

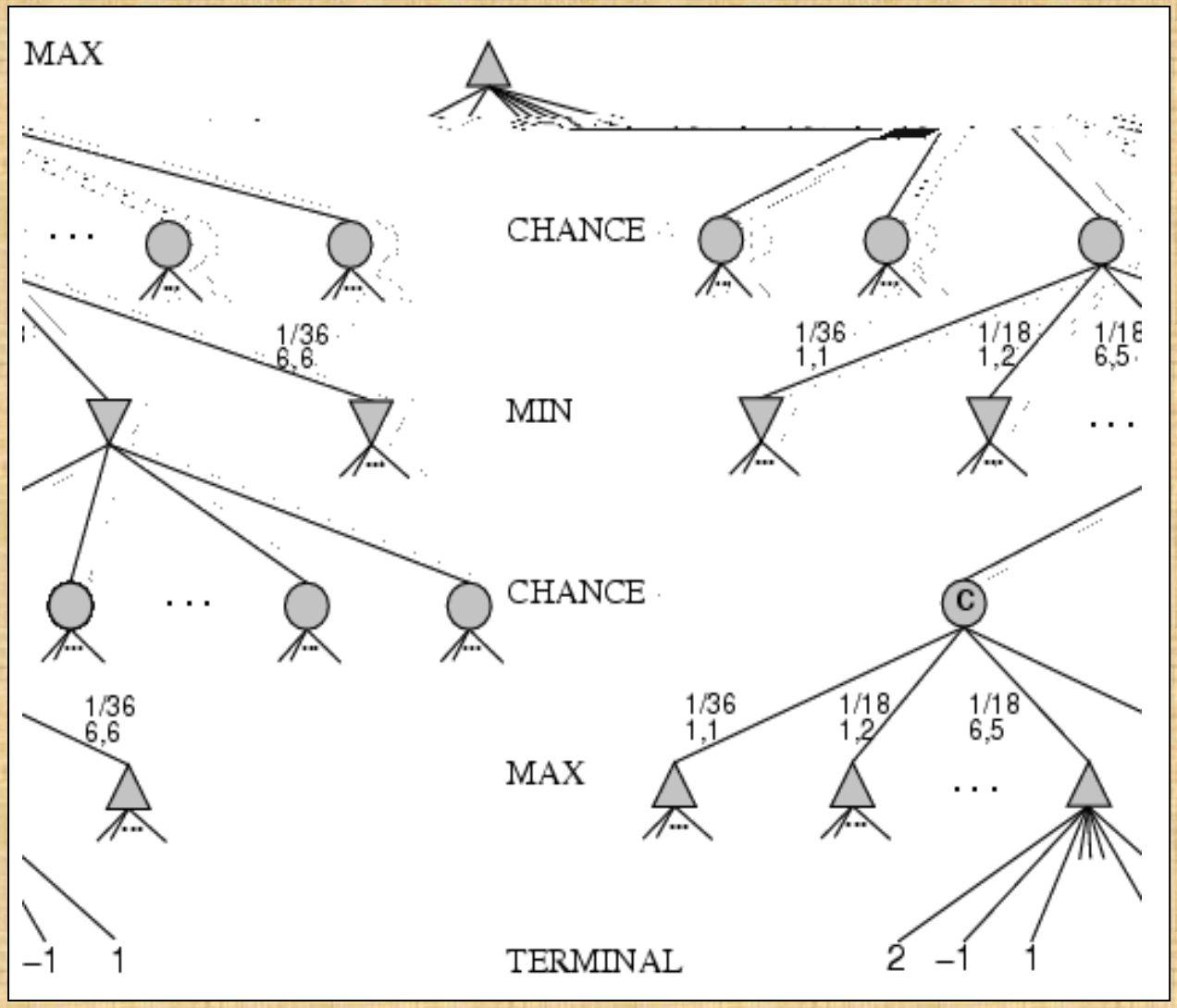


(c) White to move
Black winning



(d) Black to move
White about to lose

بازیهای همراه با عنصر شانس

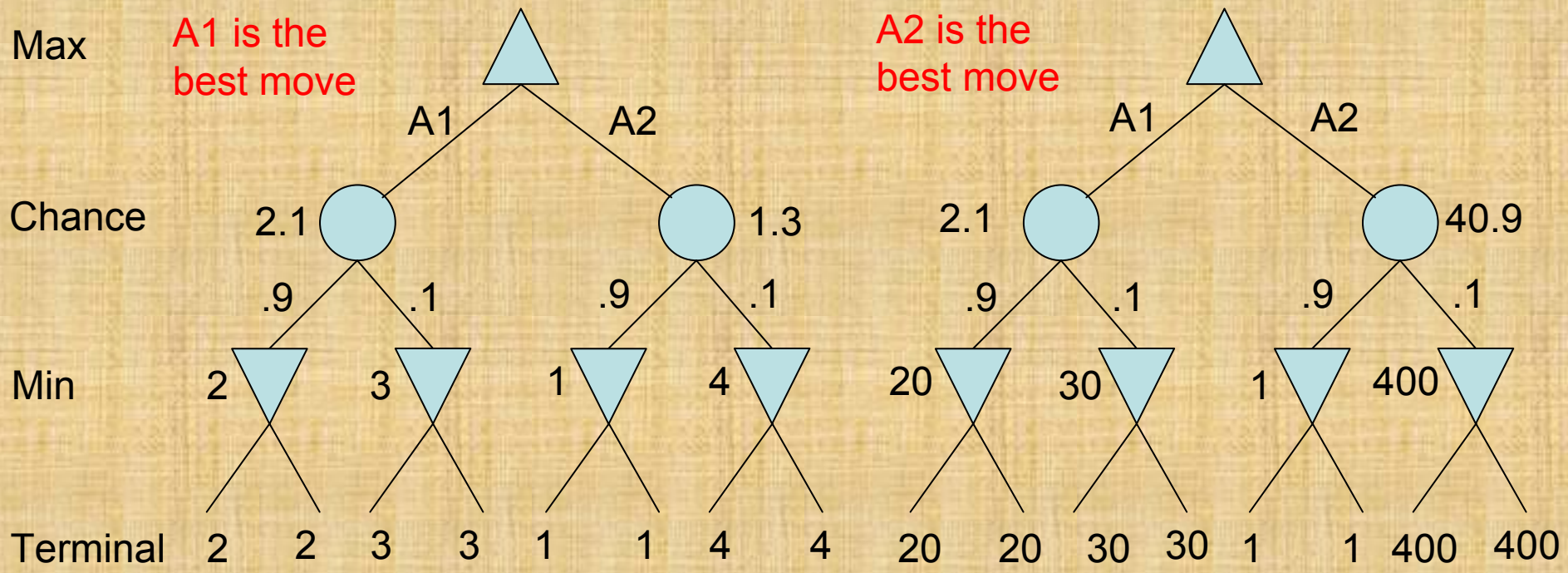


- در گره های شانس پرتاب تاس صورت می گیرد.
- هر گره Max و یا Min دارای ۲۱ فرزند شانسی می باشد.
- روش Minimax برای محاسبه مقدار گره های Min و Max استفاده می شود.

بازیهای همراه با عنصر شانس

- نحوه محاسبه مقدار یک گره در روش Minimax :

EXPECTMINIMAX(n)=

Utility(n)	if *n* is a terminal state
Max(s ϵ *Successors(n))* Expectiminimax(*s*)	if *n* is Max node
Min(s ϵ *Successors(n))* Expectiminimax(*s*)	if *n* is Min node
∑(s ϵ *Successors(n)) P(s)*.Expectiminimax(*s*)	if *n* is a chance node

بازیهای همراه با عنصر شانس