



»Лекционен курс

»Интелигентни системи



Търсене в
конкурентни
пространства



Игри

- » Проблеми, свързани с планиране в конкурентни среди
 - > Едно пространство, където съществуват агенти, които планират един срещу друг
 - > Среди, в които целите на агентите са конфликтни
- » Решаването на проблеми в такива среди се нарича „състезателно търсене“
 - > Често също „Игри“
- » Математическата теория на игрите
 - > Дял от икономическите науки
 - > Разглежда всички мулти-агентни среди като игра, където влиянието на всеки агент върху останалите е съществено
 - > Независимо от това дали агентите са кооперативни или конкурентни



Игри

» В ИИ

- > Игрите имат по-специално естество
- > Обикновено разглеждаме детерминирани **игри с нулеви суми** с пълна информация
 - + Играят се от двама играчи, които се редуват да правят ходове
 - + Ползите в края на играта са еднакво големи и противоположни
 - Шах: ако един играч победи (+1), другият трябва безусловно да загуби (0)

» Означения:

- > **MAX** – играчът, който прави първия ход
- > **MIN** – вторият играч

» Игра с нулева сума: винаги с еднакви печалби

- > Напр., шах: $0+1$, $1+0$, $0.5+0.5$

» Много от игрите, където физическите качества на състезателите са решаващи обикновено не се разглеждат

- > За сега само футболът прави изключение



Комплексност

- » Игрите принадлежат към **трудните за решаване проблеми**
- » Шах
 - > Среден фактор на разклонение: 35
 - > Обикновено необходими ходове: 50 на играч
 - > Дърво на търсене: 35^{100} или 10^{154} възли
 - > Граф на търсене: 10^{40}
- » Игрите стимулират способността за вземане на някакво решение, дори да не е възможно изчисляване на оптимално (както в реалния свят)
- » Игрите наказват неефективност
 - > Напр., използването на A^* при шах е много по-неефективно в сравнение с търсене на решение без времеви ограничения



Формална дефиниция на игра

» Състои се от следните компоненти:

- > S_0 : начално състояние
- > $To-Move(s)$: играчът, който е на ход в едно състояние s
- > $Actions(s)$: множеството на допустимите ходове в едно състояние s
- > $Result(s,a)$: модел на прехода - резултатът от един ход
- > $Is-Terminal(s)$: терминален тест, който е **true**, когато е край на играта, **false** – не е край).
Състояния, в които играта е приключила, се наричат терминални състояния
- > $Utility(s,p)$: функция на ползите за играч p , когато s е терминално състояние

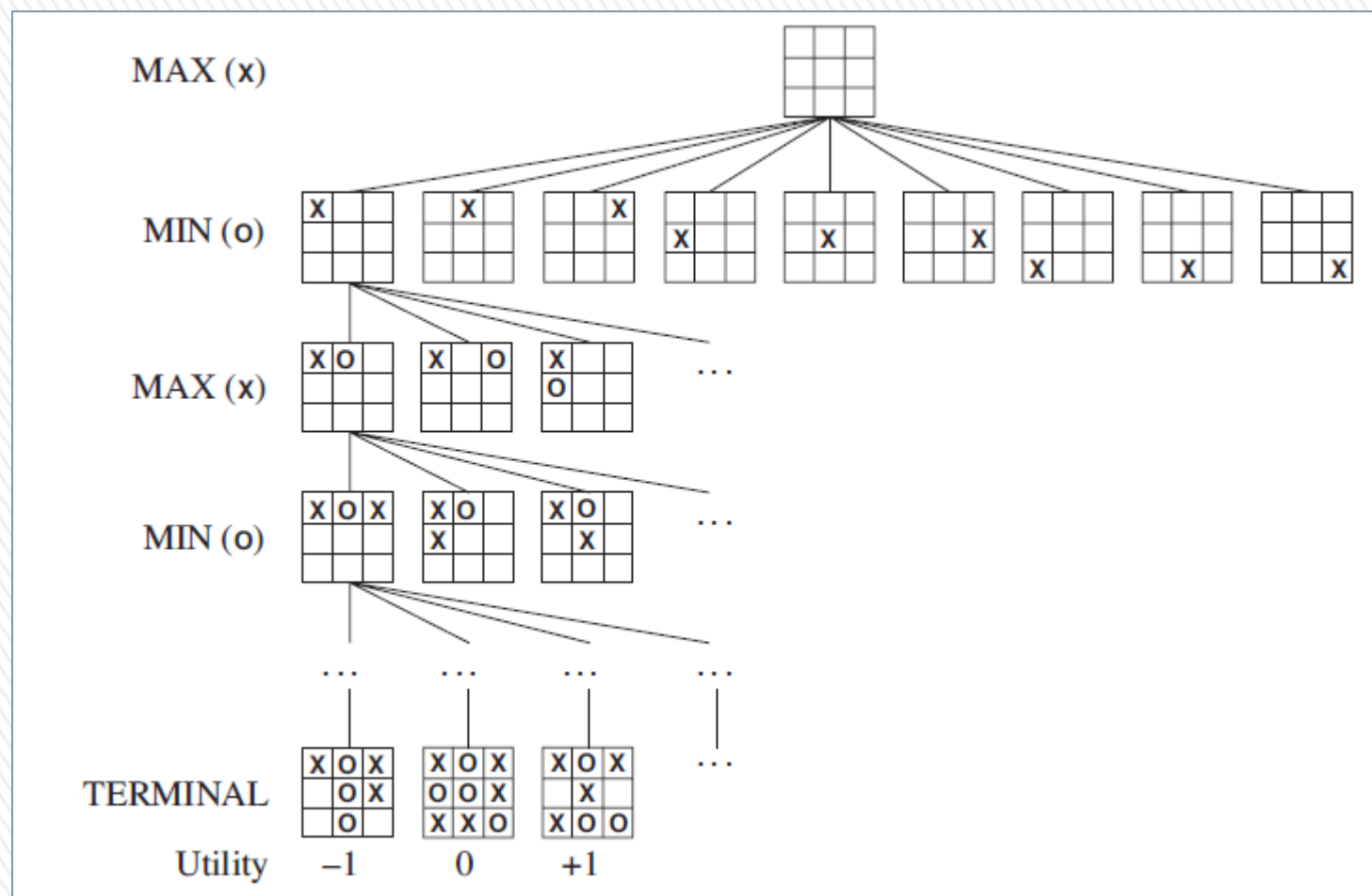


Дърво на играта

- » Възли
 - > Състояния на играта
- » Ребра
 - > Ходове
- » Дефинира се от:
 - > Началното състояние
 - > Функциите:
 - + Actions
 - + Result



Пример: морски шах



Оценка на комплексността на примера

- » Сравнително малко пространство на състоянията
 - > $362\,880 < 9!$
- » Сравнение с обикновения шах
 - > 10^{40} възела
- » Пространството се представя като игрово дърво



Характеристика на пространството

- » При обикновените проблеми за търсене оптималното решение е последователност от оператори, която води до едно целево състояние
- » В игровите пространства, когато **MAX** подготвя оптимална стратегия, трябва да се съобразява с това, че **MIN** има също „какво да каже“
- » Аналогия с търсене в И-ИЛИ-дърво
 - > **MAX** изпълнява ролята на ИЛИ
 - > **MIN** изпълнява ролята на И



Оптимални решения в игрите

» Оптимално решение

- > Последователност от действия, водеща до целево състояние, при което има печалба
- > И при което Min може да влияе

» Max трябва да намери възможна стратегия, която:

- > Определя хода му в началното състояние
- > Определя ходовете в състояния, получени като резултат от отделните противникови ходове на Min



Оптимални решения в игрите

- » За намиране на оптимална стратегия основна роля има намирането на **Minimax-стойност** за всеки възел
 - > Означаваме я като $\text{Minimax}(n)$
- » Minimax на един възел
 - > Ползата за **MAX**, да се намира в кореспондиращото състояние, приемайки че двата играча играят оптимално от тук до края на играта
 - > Minimax стойността в едно крайно състояние е точно **печалбата** (за MAX)
- » Minimax търсенето е търсене в дълбочина
 - > Във всеки момент се разглеждат възлите по дължината на един път в дървото на търсене
 - > При възможност за избор **MAX** предпочита ход към състояние с **максимална** стойност
 - > **MIN** предпочита ход с **минимална** стойност

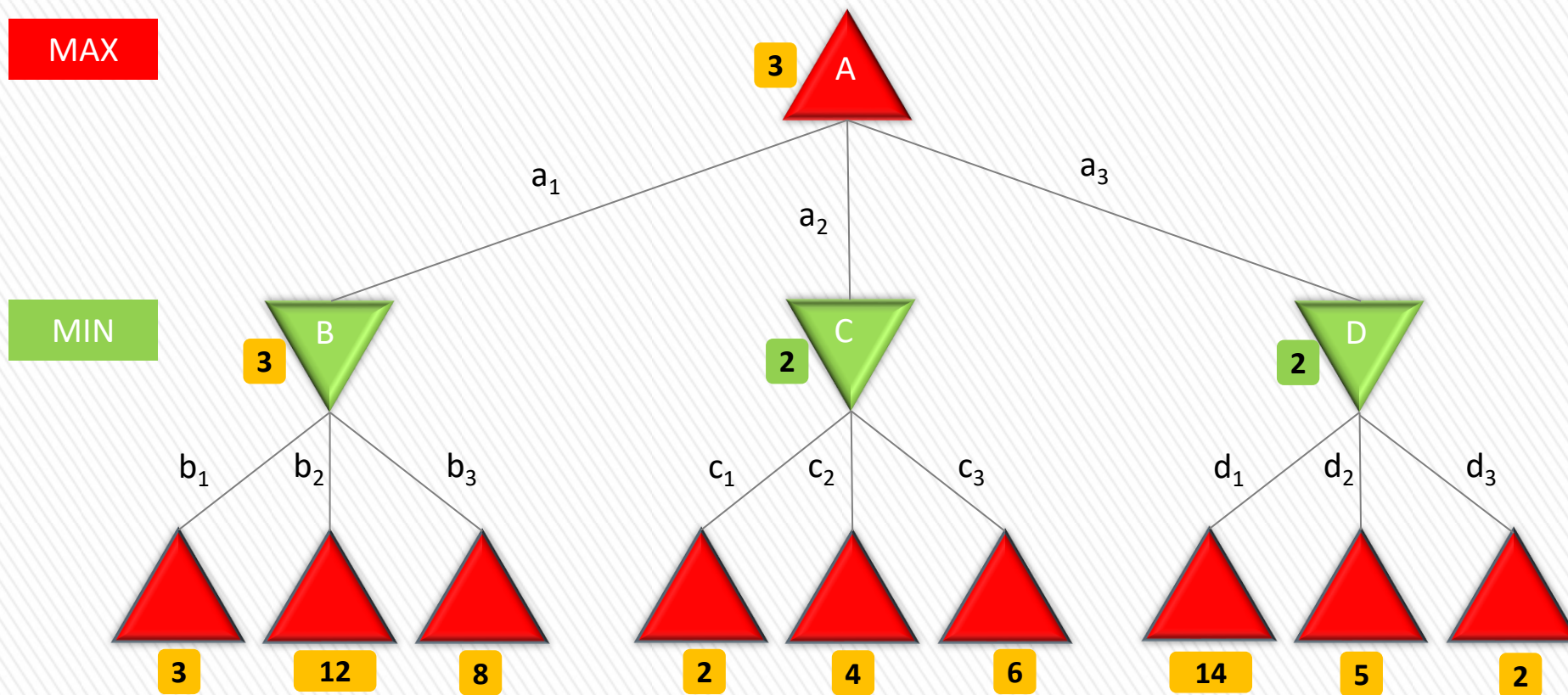


Minimax стойности на възлите

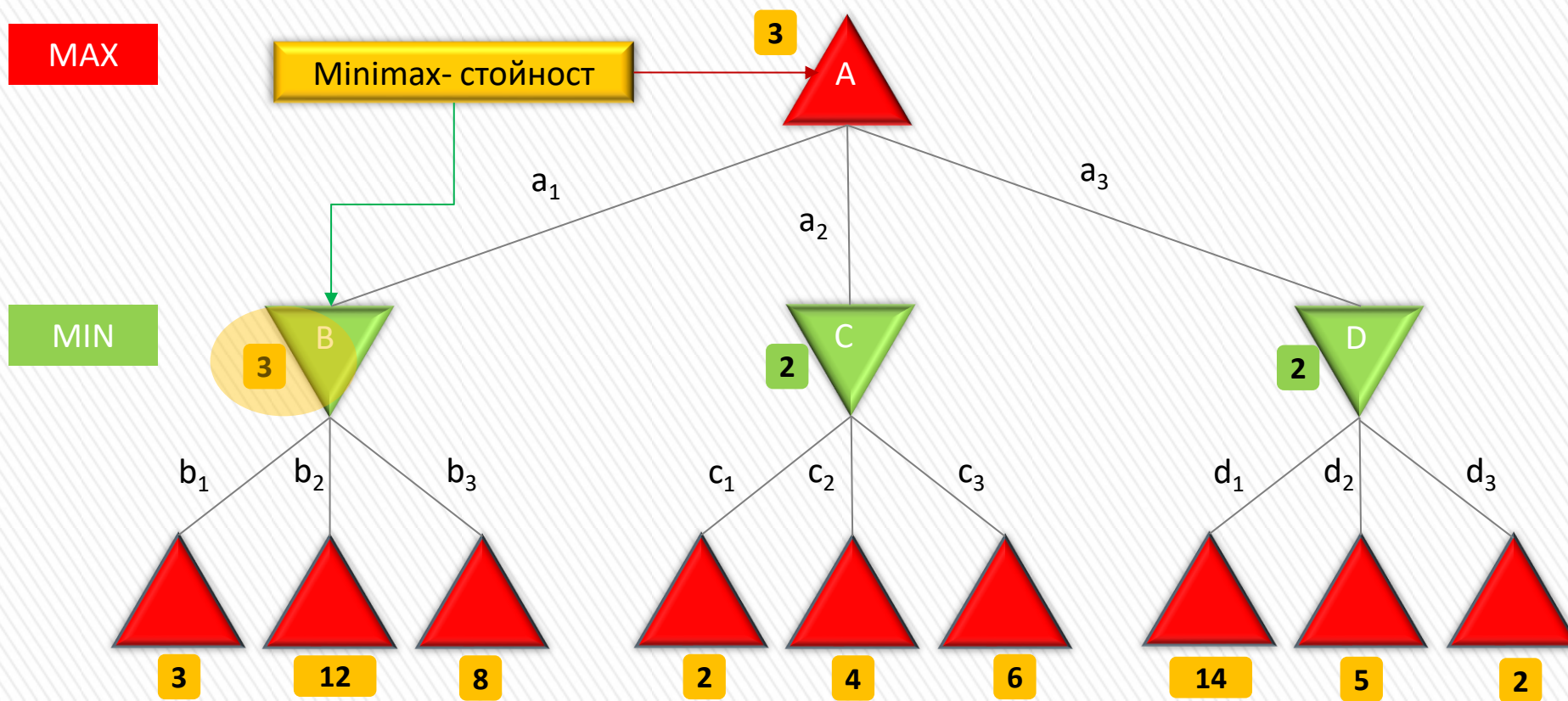
Minimax(s) =

- ✓ $Utility(s)$, когато $Terminal-Test(s)=true$;
- ✓ $\max_{a \in Actions(s)} Minimax(Result(s,a))$, когато $Player(s)=Max$;
- ✓ $\min_{a \in Action(s)} Minimax(Result(s,a))$, когато $Player(s)=Min$.

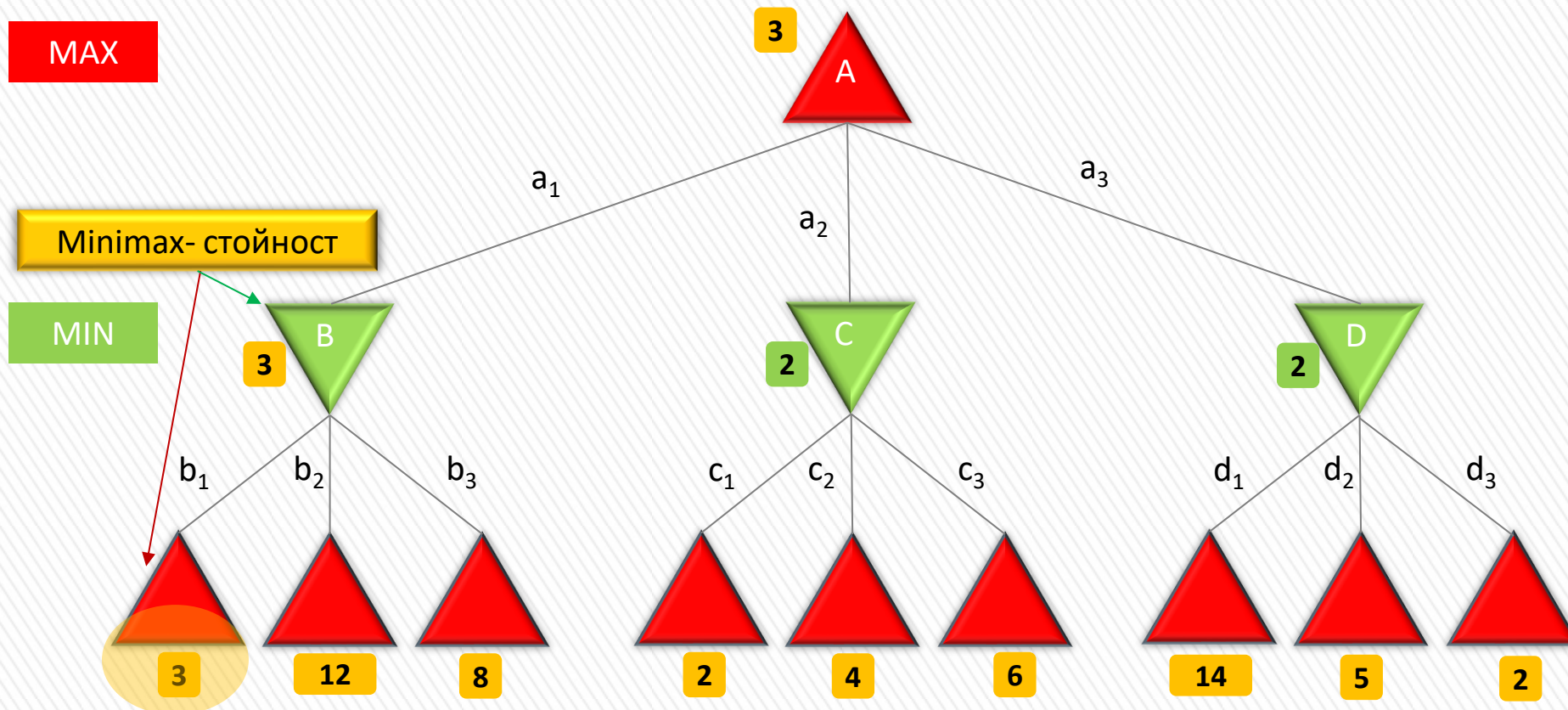
Елементарен пример: само два хода



Опростен пример



Опростен пример



Minimax алгоритъм

- » Сега сме в състояние, когато изчисляването на Minimax(s) можем да превърнем в алгоритъм за търсене, който намира най-добрия ход за MAX, като изпробва всички действия и избере това, чието резултатно състояние има най-висока Minimax стойност
- » Взема **Minimax- решения** от актуалното състояние
 - > Използва просто **рекурсивно изчисляване** на minimax-стойностите на наследниците и имплементира директно дефинираните уравнение
- » Рекурсията продължава до **листатата** на дървото



Minimax алгоритъм: псевдокод

```
function Minimax-Search(game,state) returns an action  
    player  $\leftarrow$  game.To-Move(state)  
    value, move  $\leftarrow$  MAX-Value(game,state)  
    return move
```

- За изчисляване на Minimax решения
- Връща действие, отговарящо на възможно най-добрия ход
- Т.е. ход, водещ към резултат с най-голяма печалба, при положение, че противникът играе така, че да минимизира печалбата
- За да изчислят стойността на едно състояние, функциите Max-Value и Min-Value преглеждат цялото дърво до листата

Minimax алгоритъм: псевдокод

```
function Max-Value(game, state) returns a (utility, move) pair
  if game.Is-Terminal(state) then return game.Utility(state, player), null
   $v, move \leftarrow -\infty, \text{null}$ 
  for each a in game.Actions(state) do
     $v2, a2 \leftarrow \text{Min-Value}(\text{game}, \text{game.Result}(\text{state}, a))$ 
    if  $v2 > v$  then
       $v, move \leftarrow v2, a$ 
  return  $v, move$ 
```

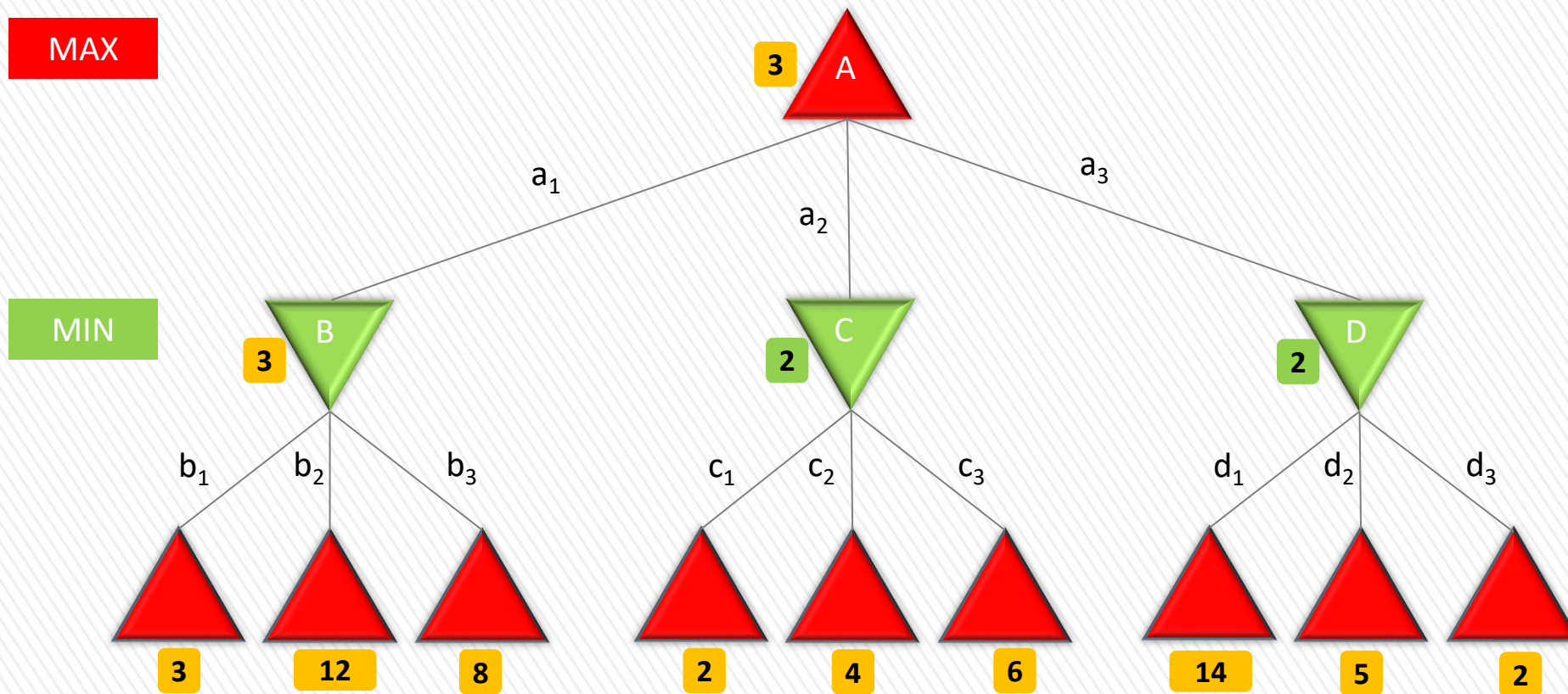
- За изчисляване на Minimax решения
- Връща действие, отговарящо на възможно най-добрия ход
- Т.е. ход, водещ към резултат с най-голяма печалба, при положение, че противникът играе така, че да минимизира печалбата
- За да изчислят стойността на едно състояние, функциите Max-Value и Min-Value преглеждат цялото дърво до листата

Minimax алгоритъм: псевдокод

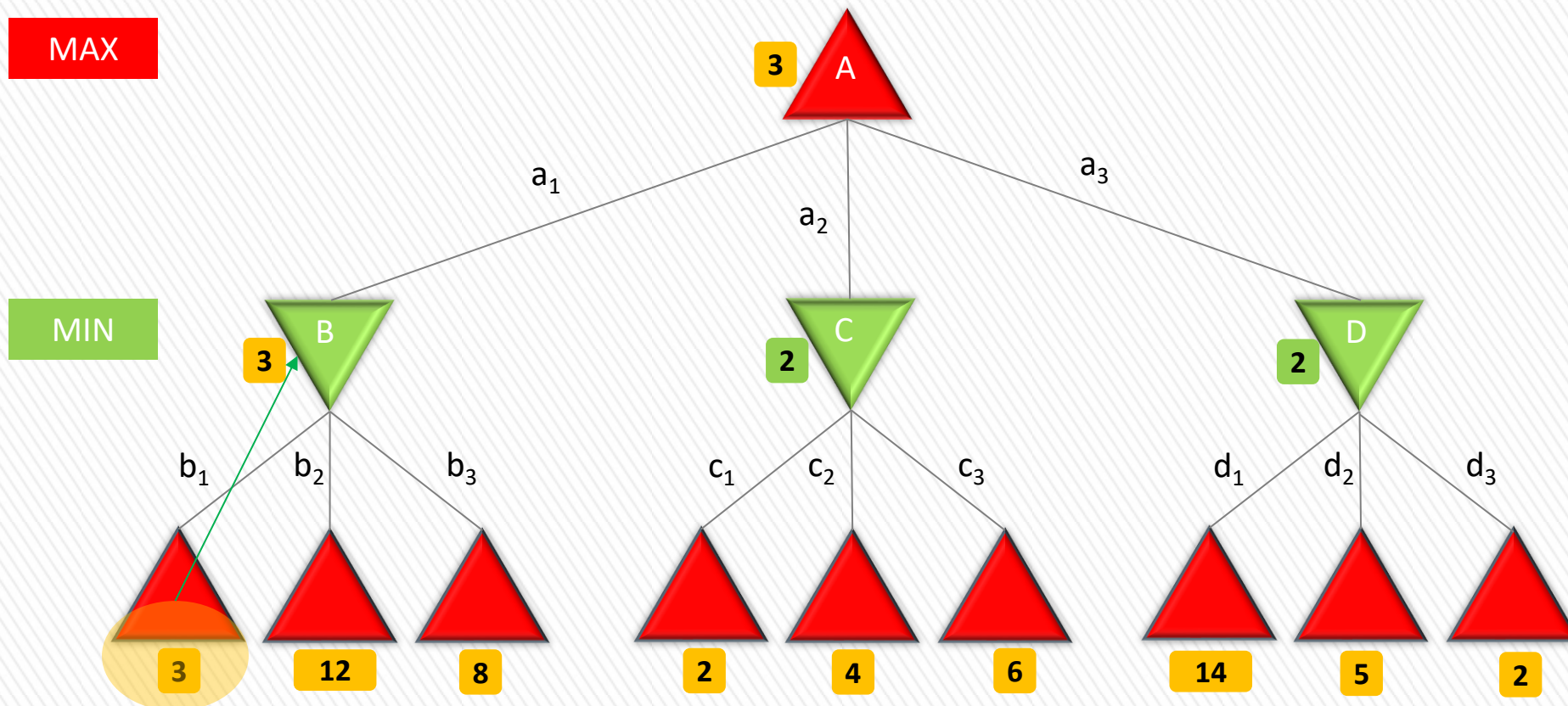
```
function Max-Value(game, state) returns a (utility, move) pair
  if game.Is-Terminal(state) then return game.Utility(state, player), null
   $v, move \leftarrow +\infty, null$ 
  for each a in game.Actions(state) do
     $v2, a2 \leftarrow \text{Max-Value}(\text{game}, \text{game.Result}(\text{state}, a))$ 
    if  $v2 < v$  then
       $v, move \leftarrow v2, a$ 
  return  $v, move$ 
```

- За изчисляване на Minimax решения
- Връща действие, отговарящо на възможно най-добрия ход
- Т.е. ход, водещ към резултат с най-голяма печалба, при положение, че противникът играе така, че да минимизира печалбата
- За да изчислят стойността на едно състояние, функциите Max-Value и Min-Value преглеждат цялото дърво до листата

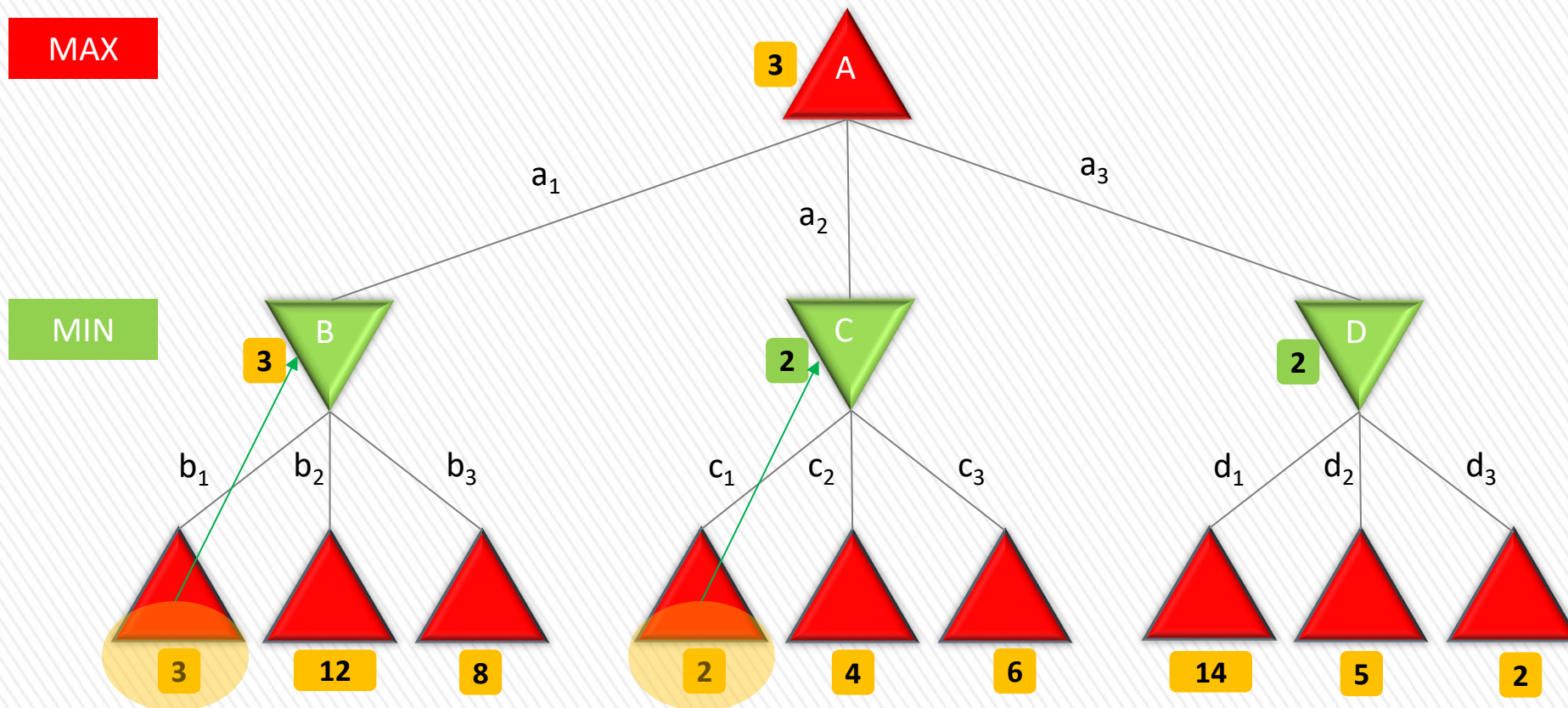
Minimax алгоритъм за примера



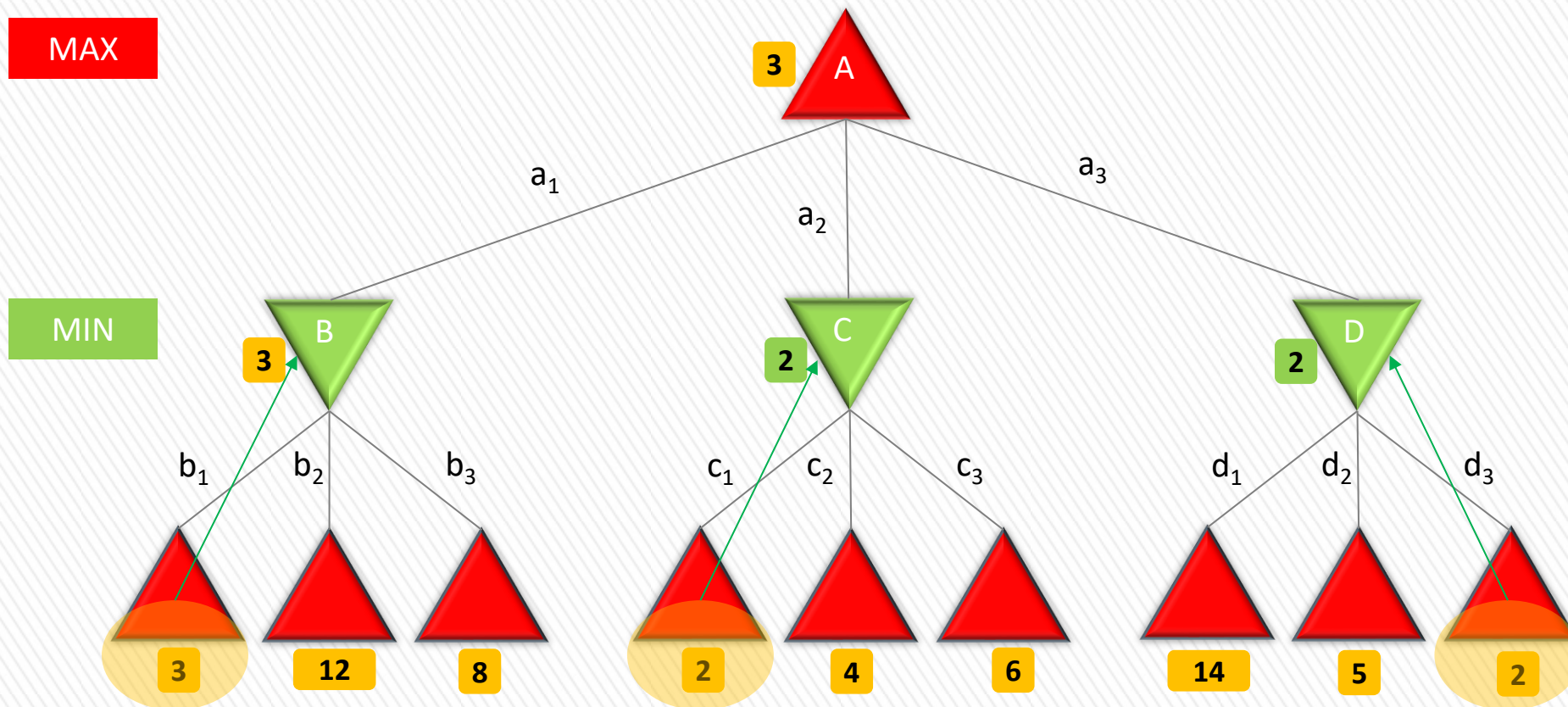
Minimax алгоритъм за примера



Minimax алгоритъм за примера



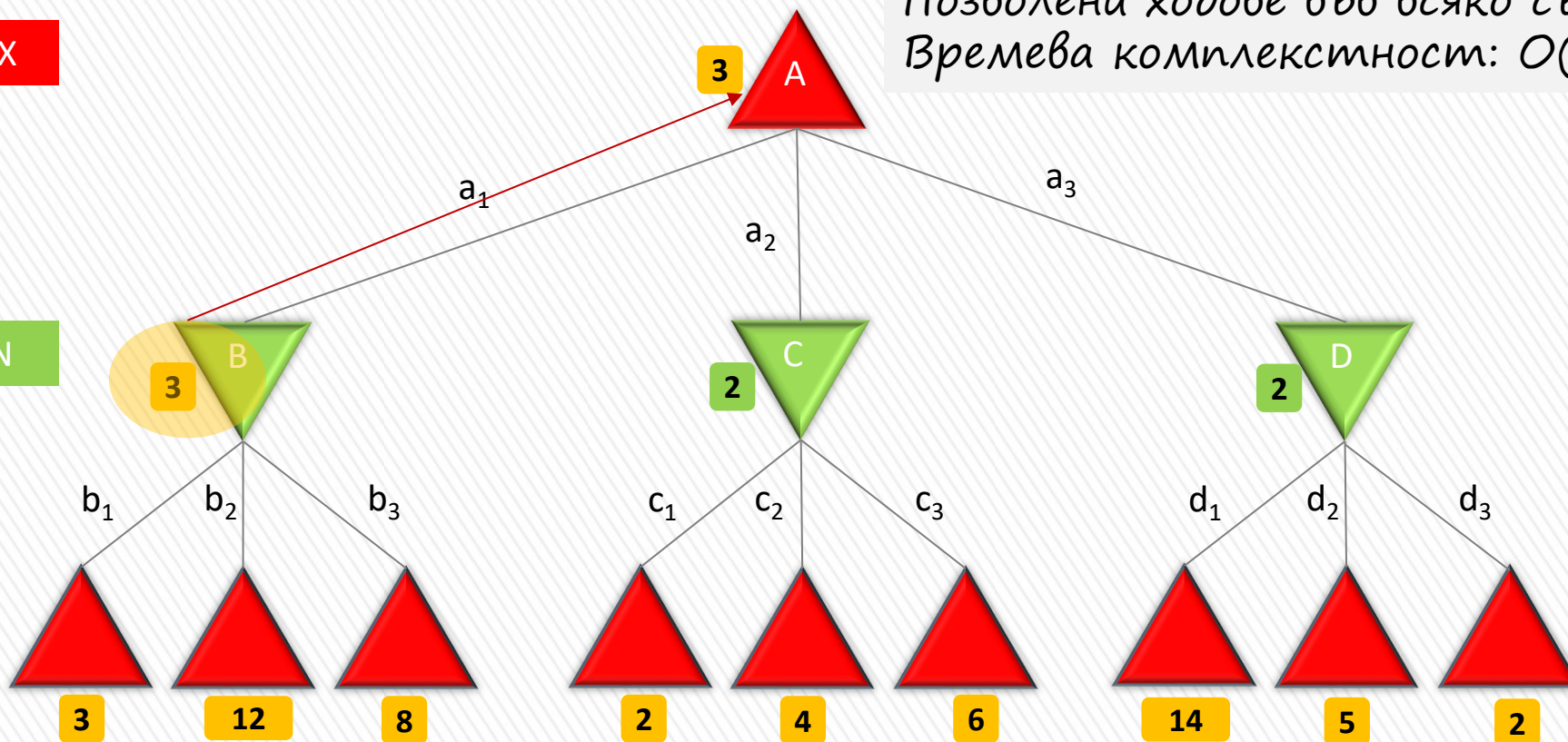
Minimax алгоритъм за примера



Minimax алгоритъм за примера

MAX

MIN



Максимална дълбочина: m
Позволен ходове във всяко състояние: b
Времева сложност: $O(b^m)$

α - β подрязване

» Проблем на Minimax търсенето

- > Броят на състоянията, които трябва да бъдат оценени **нараства експоненциално** с дълбочината на дървото
- > Сериозна комплексност

» Този ефект не може да бъде премахнат

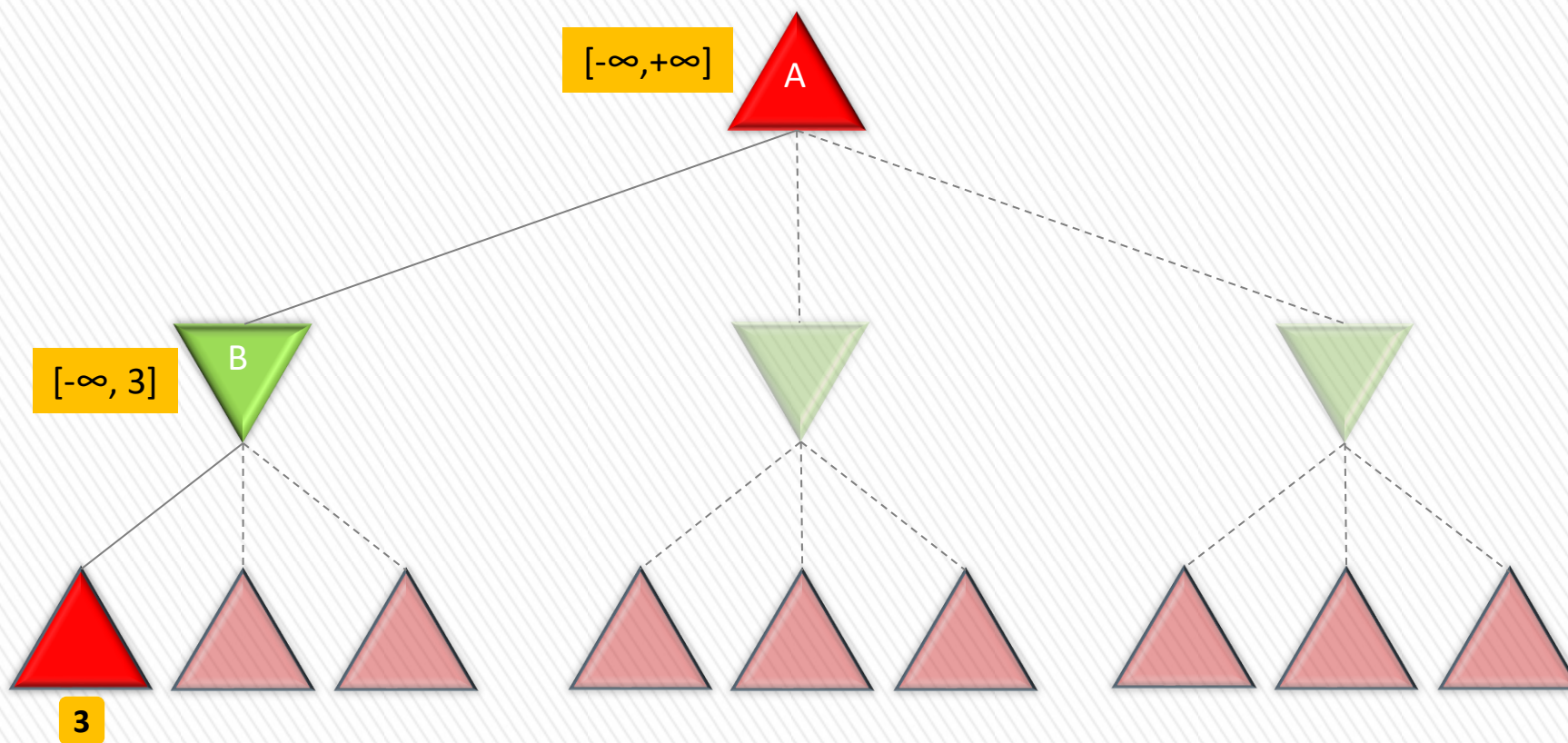
- > Възможно е обаче да се намери коректно решение, без да е необходимо разглеждането на всеки възел от пространството

» Използва се концепцията за подрязване

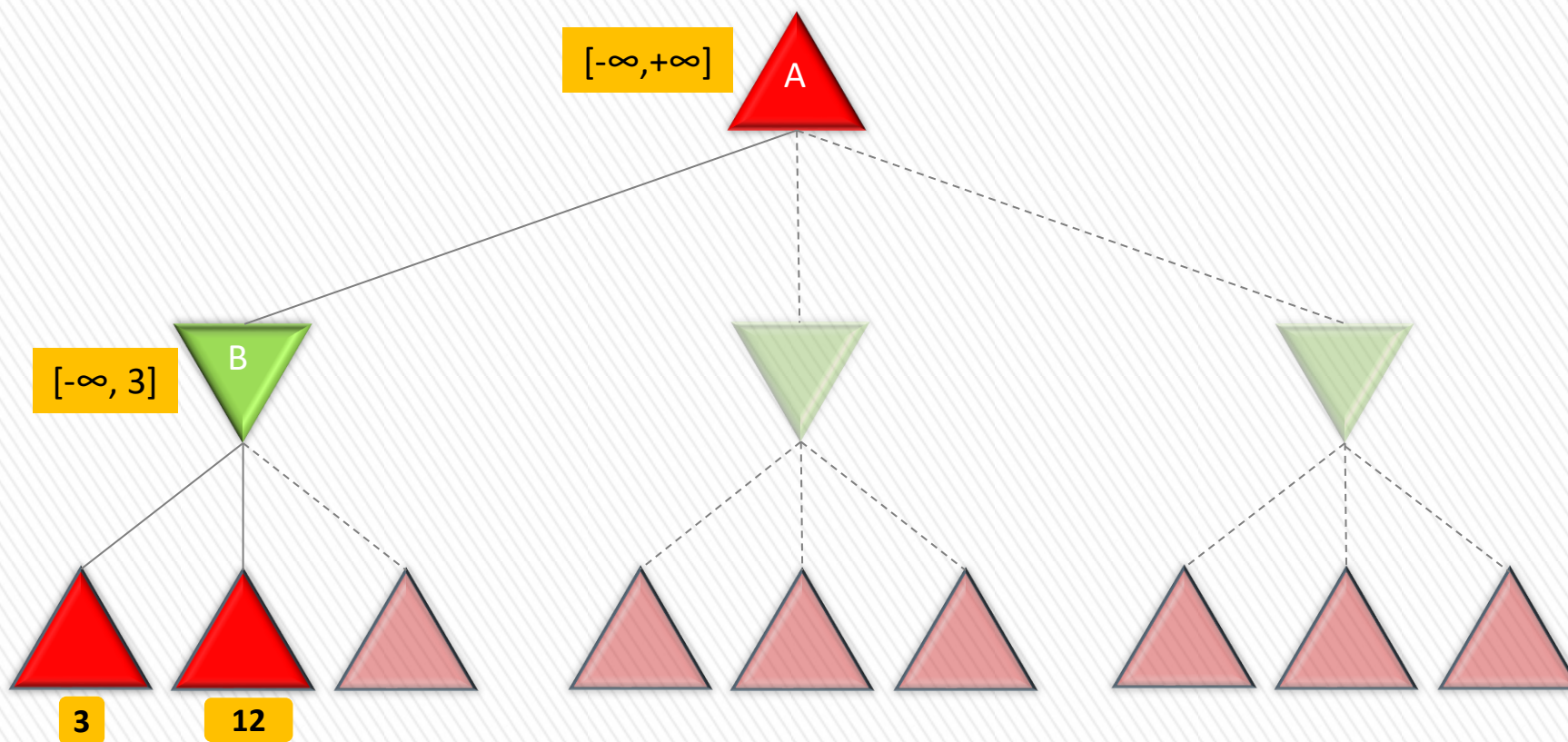
- > Съответната техника се нарича α - β подрязване
- > Подрязват се разклонения, които вероятно няма да повлияят на крайното решение



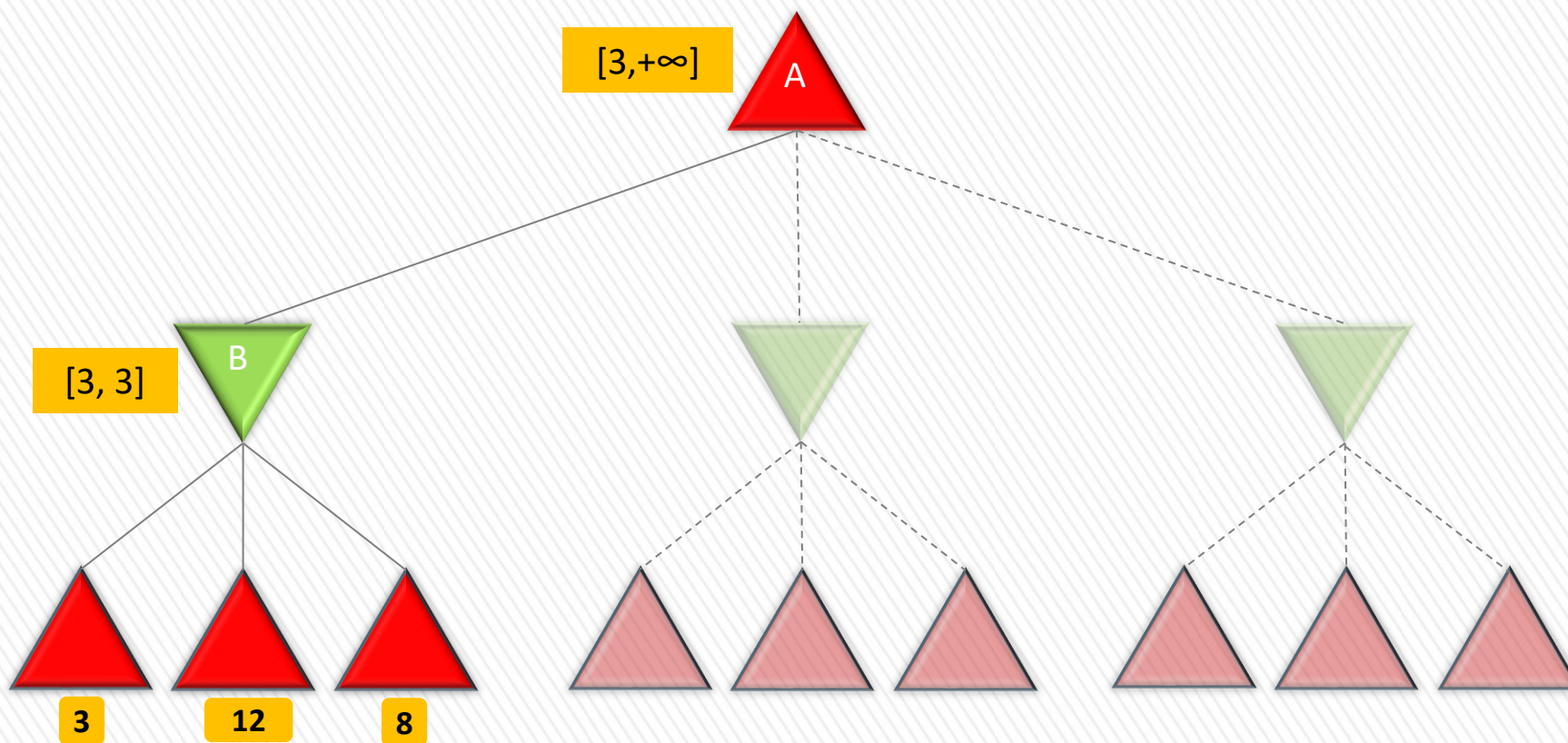
α - β подрязване за примера



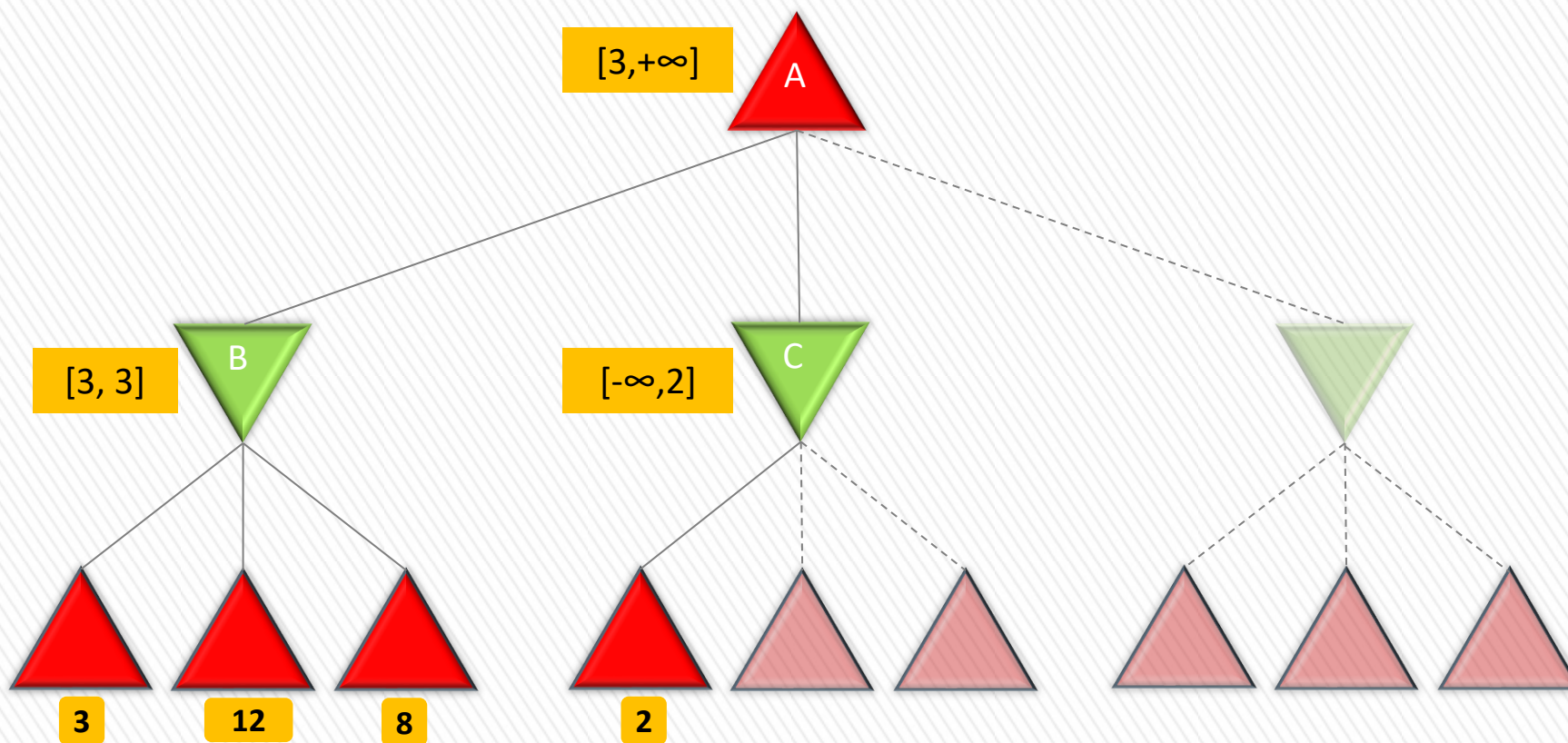
α - β подрязване за примера



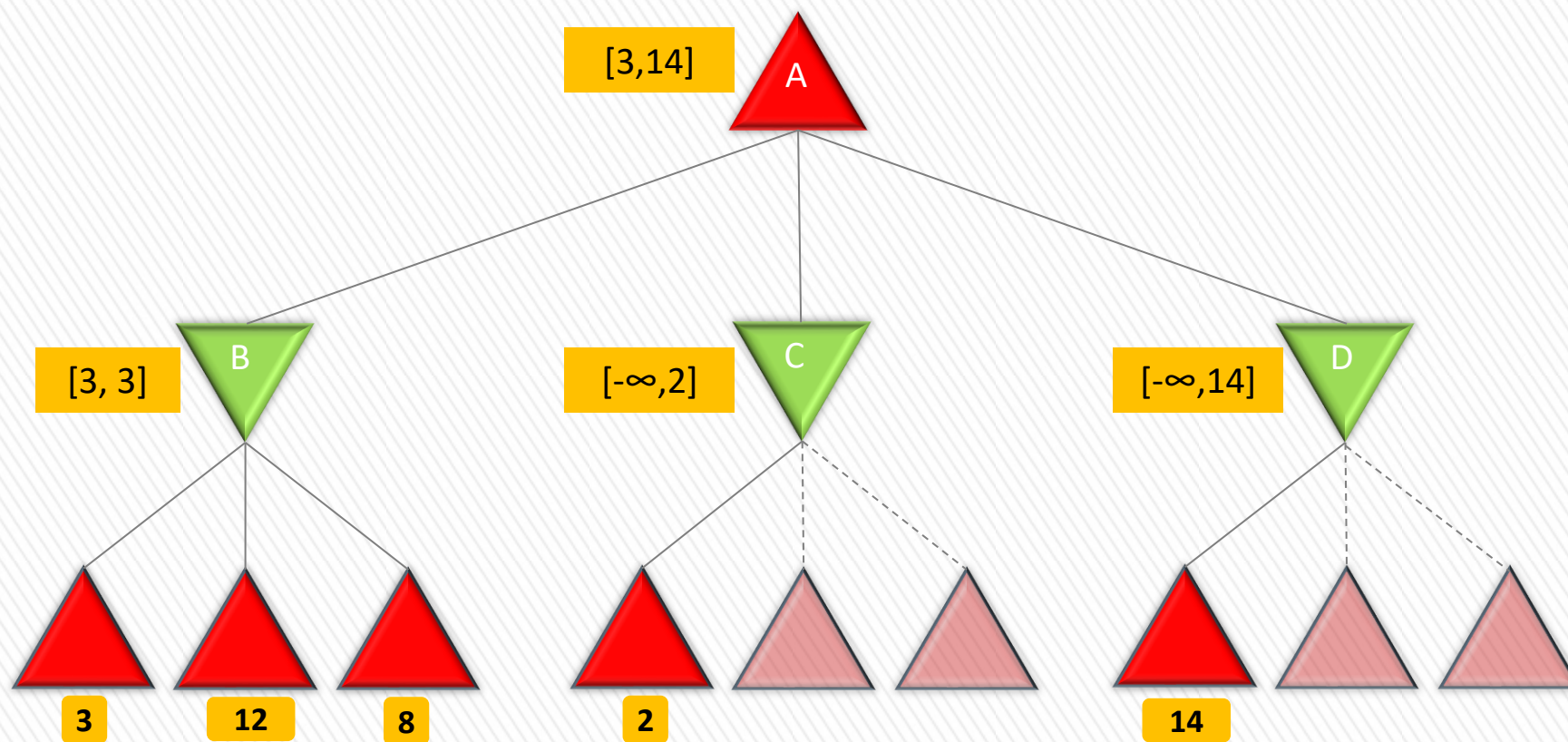
α - β подрязване за примера



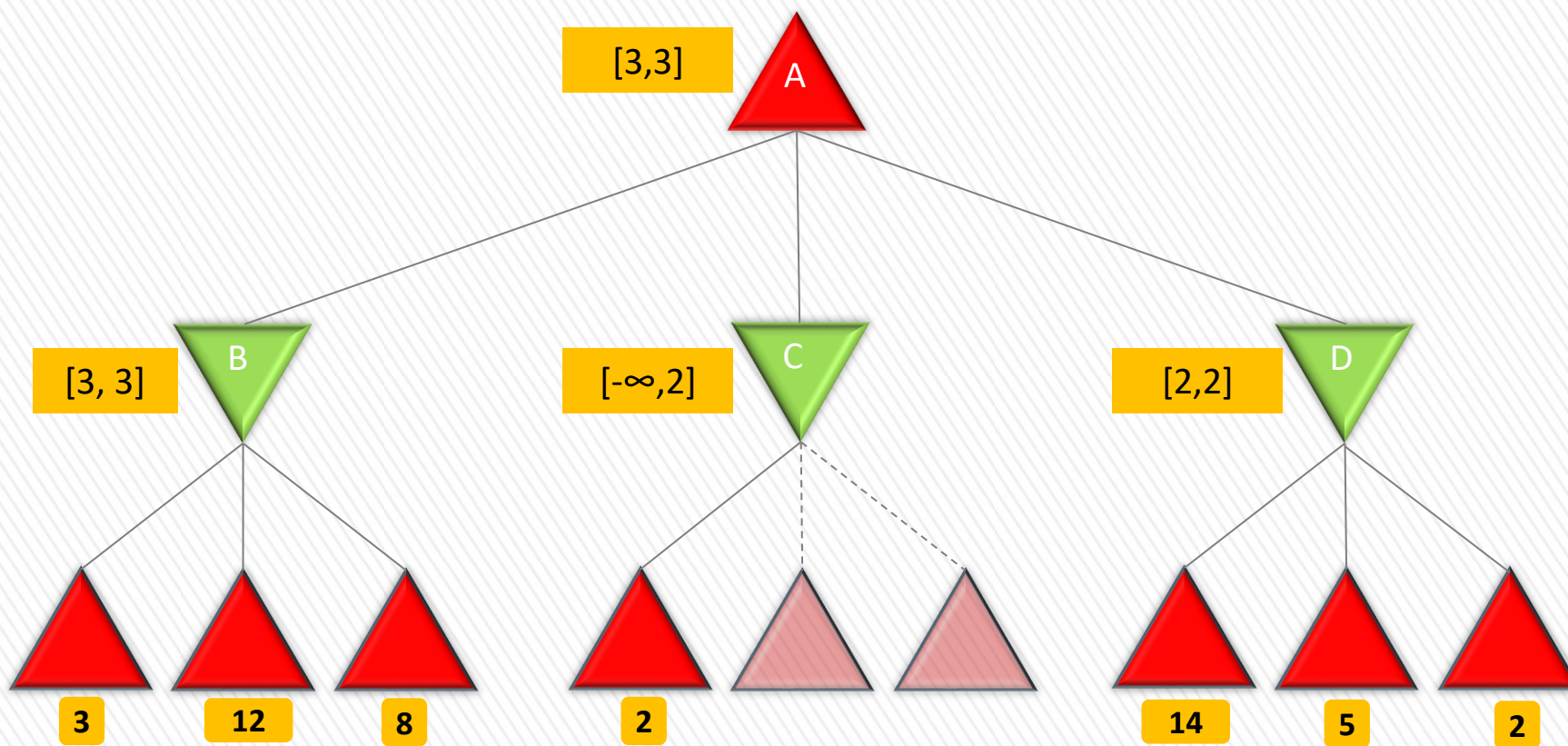
α - β подрязване за примера



α - β подрязване за примера



α - β подрязване за примера



Коментар

$$\begin{aligned}\text{» Minimax}(\text{root}) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\ &= \max(3, \min(2,x,y), 2) \\ &= \max(3, z, 2), \text{ където } z = \min(2,x,y) \leq 2 \\ &= 3\end{aligned}$$

- » Т.е. стойността на корена и съответно minimax-решението е независимо от стойностите на подрязаните листа x и y



α - β подрязване

- » Може да се прилага върху **дървета с произволна дълбочина**
- » Възможно е да се подрязват не само отделни листа, но и цели поддървета
- » Общ принцип
 - > Разглеждаме един възел **n** някъде в дървото, така че играчът има избор да направи ход към този възел
 - > Ако играчът има по-добър избор **m** в стоящ по-високо възел или в някоя точка на избор още по-горе, тогава **n** никога няма да бъде достигнат в действителната игра
 - > Щом като сме изследвали **n** достатъчно (като разглеждаме някои от неговите наследници) за да стигнем до това заключение, можем да го подрежем



α - β подрязване: общ принцип

Играч

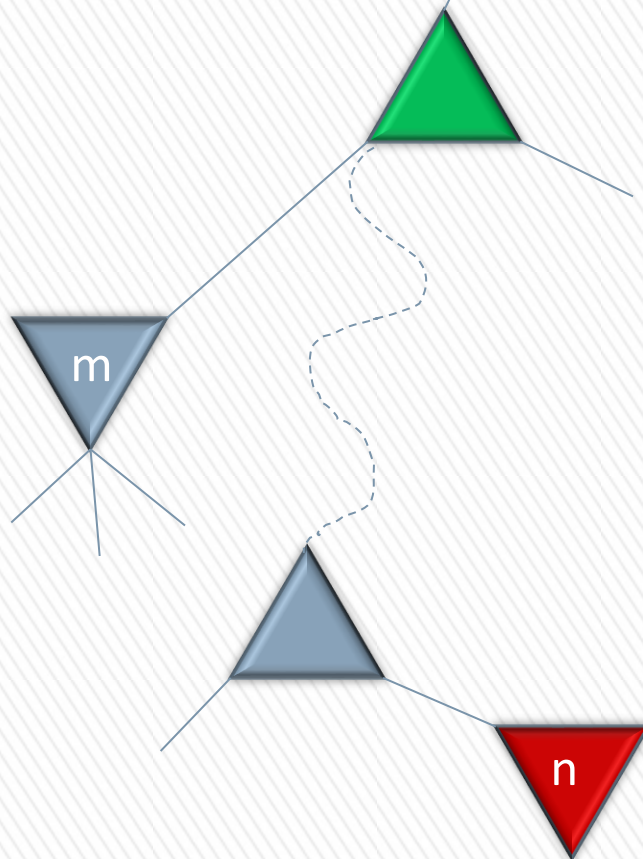
Противник

•

•

Играч

Противник



α - β подрязване

- » Както споменахме minimax търсенето е търсене първо в дълбочина
- » Името си този метод получава от следните два параметъра:
 - > α
 - + Стойността на най-добрия избор (т.е. с най-високата стойност), който сме получили досега за MAX във всяка точка за избор по дължината на пътя
 - > β
 - + Стойността на най-добрия избор (т.е. с най-ниската стойност), който сме получили досега за MIN във всяка точка за избор по дължината на пътя
- » По време на изпълнението алгоритъмът актуализира стойностите на α и β
- » Подрязва останалите разклонения в един възел, когато за стойността на актуалния възел е известно, че е по-лоша от актуалните α и β стойности за MAX съотв. MIN



α - β -Search алгоритъм: псевдокод

```
function Alpha-Beta-Search(game,state) returns an action  
    player  $\leftarrow$  game.To-Move(state)  
    value, move  $\leftarrow$  MAX-Value(game,state,  $-\infty$ ,  $+\infty$  )  
    return move
```

- Същите процедури, като при Minimax алгоритъма
- С изключение на редовете, които управляват α и β

α - β -Search алгоритъм: псевдокод

```
function Max-Value(game, state,  $\alpha$ ,  $\beta$ ) returns a (utility, move) pair
  if game.Is-Terminal(state) then return game.Utility(state, player), null
   $v \leftarrow -\infty$ 
  for each a in game.Actions(state) do
     $v_2, a_2 \leftarrow \text{Min-Value}(\text{game}, \text{game.Result}(\text{state}, a), \alpha, \beta)$ 
    if  $v_2 > v$  then
       $v, \text{move} \leftarrow v_2, a$ 
       $\alpha \leftarrow \text{Max}(\alpha, v)$ 
    if  $v \geq \beta$  then return  $v, \text{move}$ 
  return  $v, \text{move}$ 
```

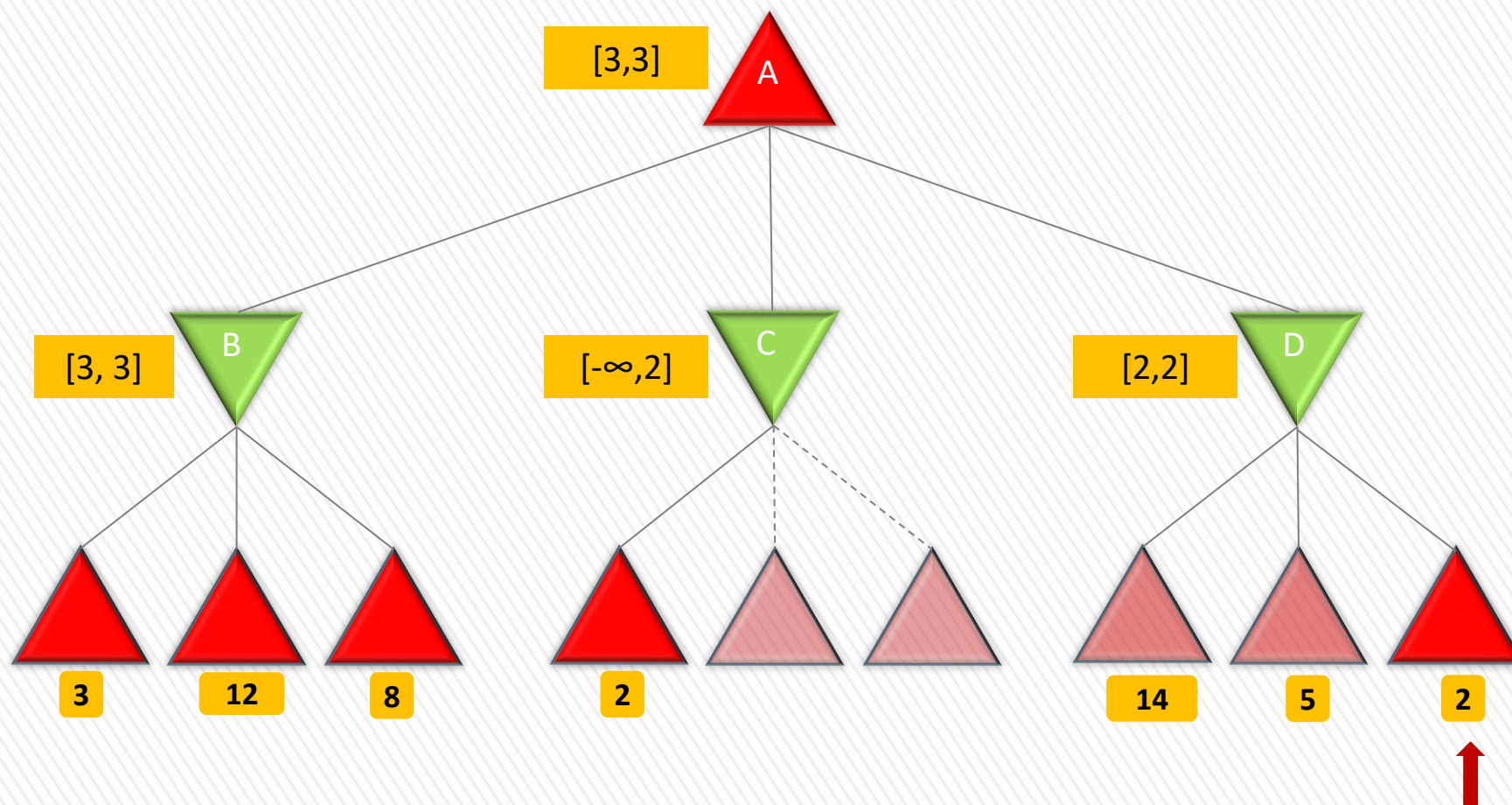
α - β -Search алгоритъм: псевдокод

```
function Min-Value(game, state,  $\alpha$ ,  $\beta$ ) returns a (utility, move) pair
  if game.Is-Terminal(state) then return game.Utility(state, player), null
   $v \leftarrow +\infty$ 
  for each a in game.Actions(state) do
     $v_2, a_2 \leftarrow \text{Max-Value}(\text{game}, \text{game.Result}(\text{state}, a), \alpha, \beta)$ 
    if  $v_2 < v$  then
       $v, \text{move} \leftarrow v_2, a$ 
       $\beta \leftarrow \text{Min}(\beta, v)$ 
  if  $v \leq \alpha$  then return v, move
  return v, move
```

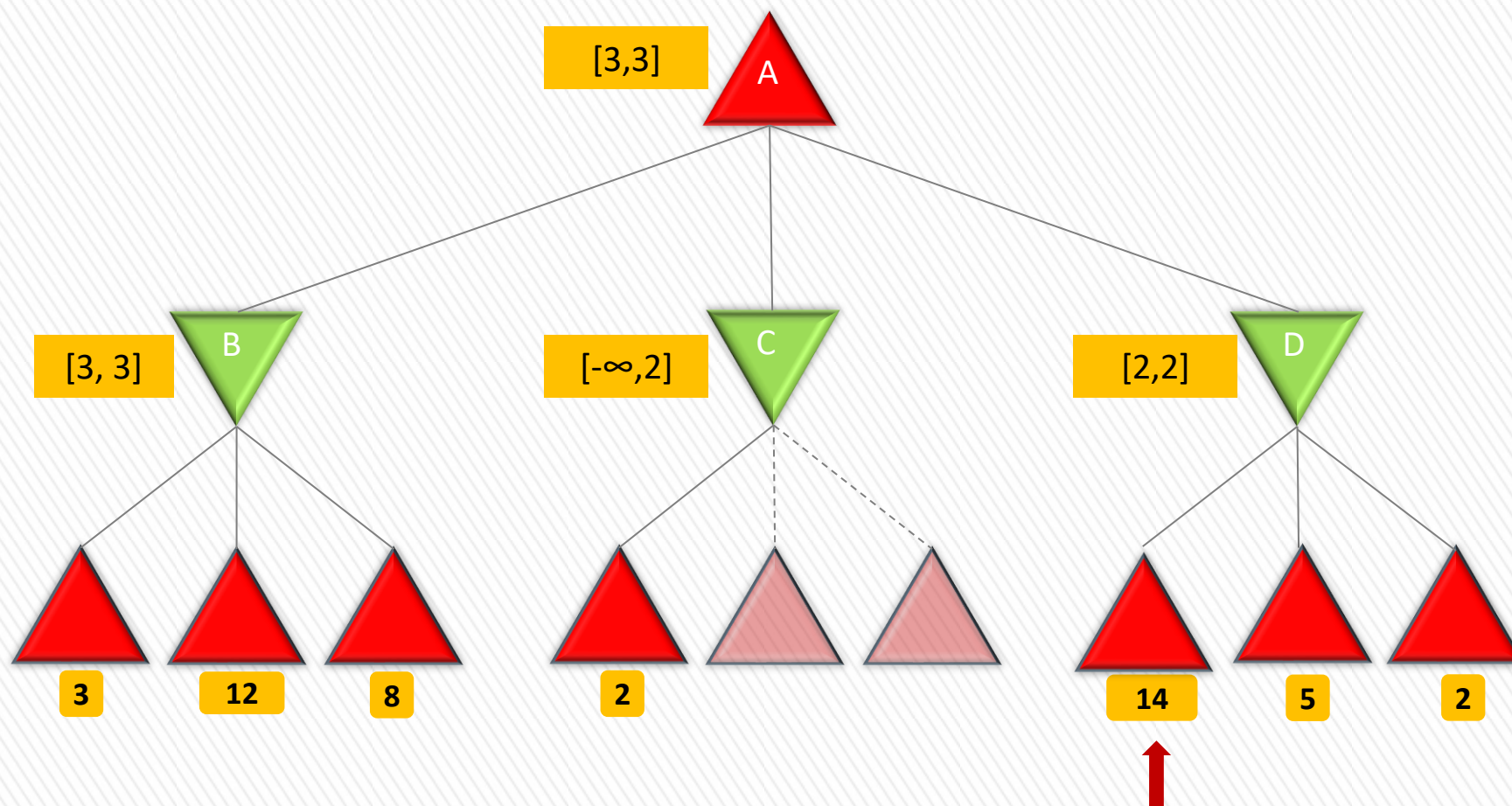

Последователност на ходовете

- » Ефективността на α – β подрязването в значителна степен зависи от последователността, в която се изследват състоянията
- » За примера: какво ще се получи ако първо генерираме третия наследник на D?
 - > Не можем да подрежем никой наследник на D понеже първо се разглеждат най-лошите наследници (от гледна точка на MIN)

α - β подрязване за примера



α - β подрязване за примера



Оценка

- » Когато това е възможно, тогава α – β подрязването трябва да оцени само $O(b^{m/2})$ възли за да намери най-добрия ход
 - > Не $O(b^m)$ като Minimax
- » Това означава, че ефективният фактор на разклонение става \sqrt{b} вместо b
 - > За шах: 6 вместо 35
- » Т.е. α – β подрязването за същото време може прегледа двойно по-дълбоко дърво в сравнение с Minimax

Оценка

- » Ако наследниците се оценяват в случайна последователност (вместо в най-добро търсене) общият брой на възлите е около $O(b^{3m/4})$ за умерен b
- » За шах:
 - > Съществува сравнително проста функция за сортиране
 - + Напр., (взема фигура, заплашва фигура, нападателни ходове, отбранителни ходове)
 - + Редукция с фактор 2, т.е. $O(b^{m/2})$ за най-добрия случай
 - > Ако въведем динамична схема за последователност (напр., първо ходове, които са били най-добри в миналото), тогава достигахме още по-близо до теоретичната граница

Непълни решения в реално време

- » Minimax алгоритъмът изгражда **пълното** игрово пространство
- » α - β -алгоритъмът позволява **подрязване** големи части от пространството
 - > Трябва обаче да претърси целия път до крайните състояния (поне на една част на пространството)
- » Достигането на необходимата дълбочина в много случаи е **невъзможно**
 - > Ходовете трябва да се направят в определено време (в шаха, няколко минути)

Решение

- » Търсенето се прекъсва предварително
- » Използва се **евристична оценъчна функция** за състоянията, която **трансформира** междинните (некрайните) възли в листа
- » Minimax или α - β се модифицират по два начина:
 - > Функцията на печалбата се заменя с евристичната функция **Eval**
 - > Тестът за край се заменя с **Cutoff-Test** (тест за прекъсване)
 - + Решава кога да се прилага Eval

Евристичен Minimax

H-Minimax(s, d) =

- Eval(s, MAX), когато Is-Cutoff(s, d)
- $\max_{s \in \text{Actions}(s)} \text{H-Minimax}(\text{Result}(s, a), d + 1)$, когато To-Move(s)=MAX
- $\min_{s \in \text{Action}(s)} \text{H-Minimax}(\text{Result}(s, a), d + 1)$, когато To-Move(s)=MIN

Оценъчни функции

» Оценъчна функция

- > Оценка на очакваната печалба от играта от една определена позиция (подобна на оценката на разходите до целта)
- > Пример: шах
 - + В течение на стотици години, шахматистите са разработили възможности за оценка на една позиция - понеже възможностите на хората да търсят са много по-ограничени от компютърните програми

» Производителността на една игрова програма зависи от качеството на оценъчната функция

- > С лоша функция един агент очаквано ще губи

Добри оценъчни функции

» Как могат да се създават добри оценъчни функции?

- > Оценъчната функция трябва да може да сортира крайните състояния, също както функцията на печалбата
 - + Състояния, представлящи победи, трябва да доставят по-добър резултат от тези, представлящи равни резултати, които от своя страна трябва да са по-добри от загубите
- > Изчисленията не трябва да продължават дълго
- > Оценъчната функция за състояния, различни от крайните, силно да корелират с действителните шансове за победа

Шансове за победа

» Понятието „шансове за победа“ поставя различни въпроси

> Шах не е „игра на късмета“

- + Със сигурност знаем актуалното състояние (също не се хвърлят зарове)
- + Въпреки това, когато търсенето трябва да прекъсне в некрайно състояние, алгоритъмът е несигурен в окончателните резултати на тези състояния
- + Тази несигурност е породени от ограничените изчислителни ресурси, а не от липсата на информация
- + Крайният резултат се „гадае“

Концепция

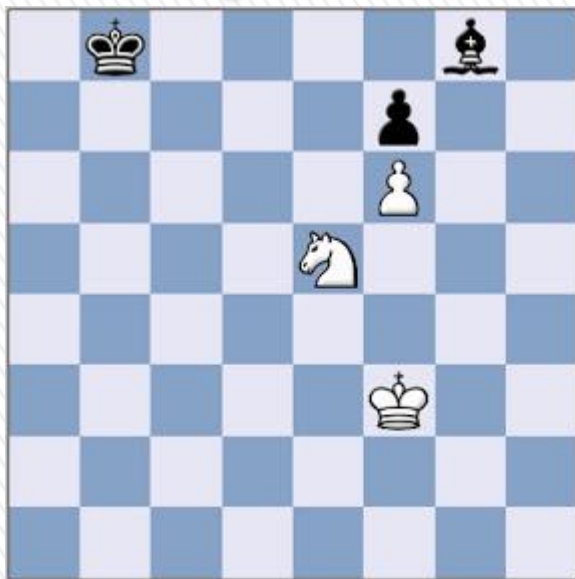
- » Да формулираме тази концепция по-конкретно
- » Повечето оценъчни функции изчисляват различните свойства на състоянието
 - > Напр., брой бели пешки, брой черни пешки, ...
- » **Оценъчната функция може да комбинира оценките** на свойствата за получаване на обща стойност

Концепция

» В тяхната общност, свойствата дефинират различни **категории или еквивалентни класове** от състояния

- > Състоянията в една категория имат еднакви стойности за всички свойства
- > Общо, всяка категория съдържа:
 - + Определени състояния, водещи до победа
 - + Други, водещи към равен резултат
 - + Трети, означаващи загуба
- > Оценъчната функция не може да знае кои състояния какви са, но тя може да върне една единична стойност, която отразява пропорцията на състоянията към всеки резултат

Пример: шах



Всяка фигура има материална стойност



Други свойства като напр., „добра пешечна структура“, „сигурност на царя“ могат да имат също материална стойност

0.5

$$\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s) = \sum w_i f_i(s)$$

Брой фигури

Материална стойност

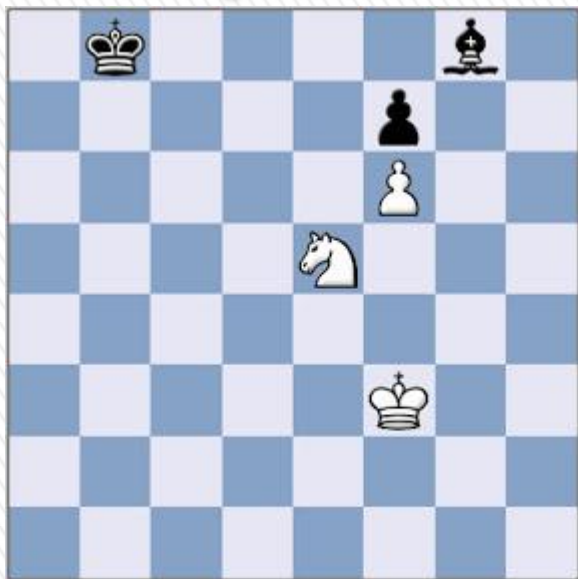
Състояние „2 пешки срещу една пешка“ (в съответствие с опита)

- 72% → победа (печалба 1)
- 20% → загуба (печалба 0)
- 8% → реми (печалба 0.5)

Оценка на състоянията в категорията е очаквана стойност:

$$(0.72 \times 1) + (0.20 \times 0) + (0.08 \times 0.5) = 0.76$$

Пример: шах



Свойствата и теглата не са част от правилата на шаха – изведени са от опита, събиран от човечеството в продължение на стотици години

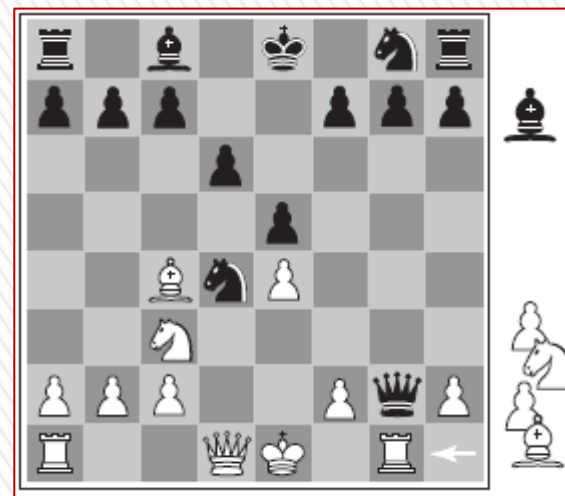
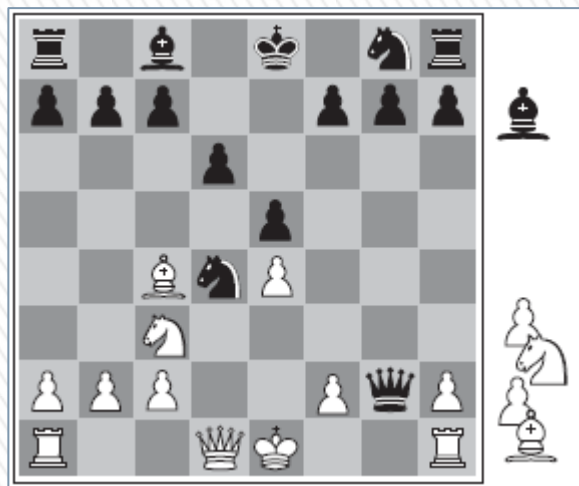
Прекъсване на търсенето

- » α - β търсенето се променя така, че да извиква функцията Eval, когато е целесъобразно да се прекъсне търсенето
 - > Замествахме условните оператори, съдържащи Is-Terminal
- » Вкарваме допълнителни разходи за поддържане на дълбочина, която се увеличава при всяко рекурсивно извикване
- » Обемът на търсенето може да се управлява най-просто, когато е твърдо зададена дълбочината
 - > Is-Cutoff(state, depth) връща true за всички стойности на depth, които са по-големи от пределната дълбочина (d)
 - > Също true за всички крайни състояния (както Is-Terminal)
 - > Дълбочината d трябва да се определи така, че ходът да може да бъде направен в определеното време
- » Тези прости допускания могат да водят до грешки
 - > Поради естеството на оценъчната функция това са само предположения

Пример за грешна оценка

Имаме две позиции с еднаква оценка:

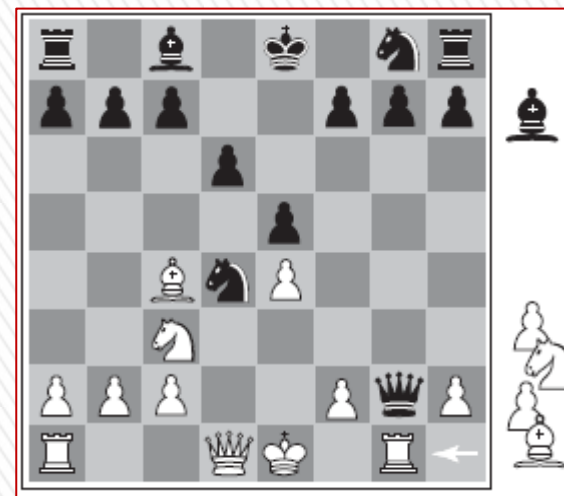
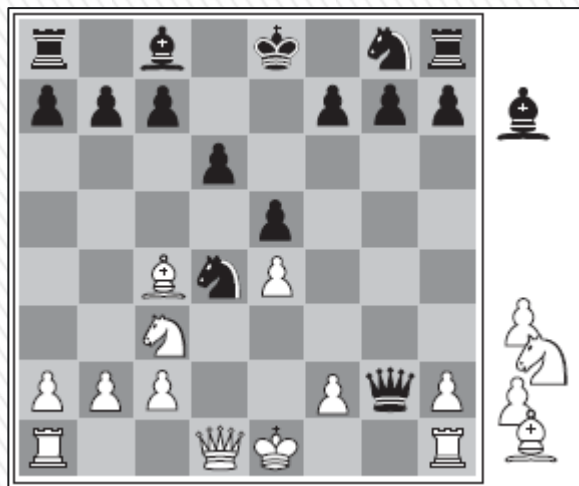
- Черни: кон и 2 пешки повече – в това състояние оценяваме евентуална победа за тях
- Белите са на ход



Пример за грешна оценка

Имаме две позиции с еднаква оценка:

- Черни: кон и 2 пешки повече – в това състояние оценяваме евентуална победа за тях
- Белите са на ход



В следващия ход белите
вземат дамата, без да има
компенсация за това

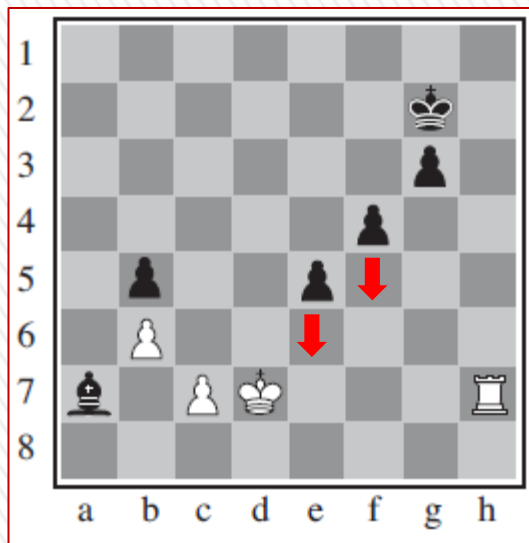
Подобрение

- » Очевидно, необходим **по-сложен тест за прекъсване**
- » Оценъчната функция трябва да бъде прилагана само в позиции, които са „**спокойни**“
 - > Т.е., в близко бъдеще не водят до големи промени
 - > В примера:
 - + Такива могат да бъдат удари по значими фигури
- » „**Неспокойните**“ позиции могат да бъдат разширявани докато станат спокойни
 - > Такова допълнително търсене се нарича „спокойно търсене“
 - > Понякога е ограничено в избора само на определени видове ходове (напр. удари по значими фигури), които бързо премахват несигурността на позицията

Ефект на хоризонта

- » Ефектът на хоризонта се появява, когато програмата се конфронтира с ход на противника, който причинява съществени щети и не може да бъде избегнат, посредством **тактики за временно забавяне**
- » Ефектът на хоризонта се елиминира по-трудно

Пример



- С черните на ход, черният офицер няма шансове за измъкване
- Обаче, черните могат да осуетят това събитие, като жертват 2 пешки
- Преместват събитието извън хоризонта
- Неминуемото вземане на офицера се случва „извън“ хоризонта, така че алгоритъмът оценява двете жертви като положителни ходове (вместо като грешни)

Предварително съкращаване

- » В предишните разсъждения – търсенето се прекратява на определена дълбочина
- » Възможно е също да се предприеме **предварително отрязване**
 - > Т.е. определени ходове във възлите се отрязват веднага без да се проследяват по-нататък
 - > Повечето играчи разглежда (проследяват) малко ходове от всяка позиция
- » Подход при предварителното съкращаване: „търсене в лъчи“
 - > Във всеки слой се разглежда само един „лъч“ от n най-добри ходове (относно оценъчната функция)
 - > Вместо да се разглеждат всички възможни ходове

Предварително съкращаване

- » Този подход обаче е опасен – няма гаранция, че можем да задраскаме най-добър ход
- » Напр., ProbCut алгоритъм – версия на α - β търсене с предварително съкращаване
 - > Използва статистически данни от предишен опит за да намали опасността за отрязване на най-добър ход
- » Когато се комбинират разгледаните в лекцията техники може да се разработи програма, която играе шах удивително добре

Търсене и преглед

- » За дебюти и ендшпили вместо търсене обикновено се използва преглед на предварително подготвена информация (напр., таблици)
 - > Позволяват много по-бързо да се вземат решение за ходовете
 - > Вместо да се търси в едно огромно пространство
- » За дебютите компютрите се опират на ноу-хау на хора-експерти
 - > Техните препоръки се записват в специални таблици

Търсене и преглед

- » Компютърните програми могат да използват също **статистически данни** от играни в миналото партии, които се съхраняват в БД
 - > За да установят най-често използвани дебюти, които са водели до победи
- » Обикновено след 10-тия ход се получава особена (характерна) позиция, която **трудно се идентифицира** с позната от миналото
 - > Тогава софтуерът превключва от преглед към търсене
- » В ендшпила съществуват **значително по-малко** възможни позиции и с това се увеличават възможностите за работа в режим на преглед
- » Тук обаче компютърът притежава ноу-хау
 - > Компютърният анализ на ендшпила превишава значително това, което е постигнато от хората



Благодаря за вниманието!