



دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

جستجوی خصمانه

«هوش مصنوعی: یک رهیافت نوین»، فصل ۵

ارائه‌دهنده: سیده فاطمه موسوی

نیم‌سال اول ۱۴۰۰-۱۳۹۹

رئوس مطالب

- محیط‌های چندعاملی
- بازی‌ها و مسائل جستجو
- معرفی دو الگوریتم معروف بازی‌ها
 - الگوریتم بیشینه کمینه
 - الگوریتم آلفا-بتا
- تصمیمات بی‌درنگ ناکامل
- بازی‌های دارای عنصر شانس

محیط‌های چندعاملی

- در محیط‌های چندعاملی هر عامل باید فعالیت سایرعامل‌ها و تأثیر آنها بر روند کار خود را در نظر بگیرد.
- رفتارهای غیر قابل پیش‌بینی عامل‌های دیگر می‌تواند باعث بروز **مقتضیات** بسیاری در فرآیند حل مسأله شود.
- رقابتی ← محیط‌های خصمانه (بازی)
 - اهداف عامل‌ها با هم در تضاد هستند
 - هر عامل سعی می‌کند کارایی خود را افزایش دهد
- همکار
 - عامل‌ها اهداف مشترکی دارند.
 - عملی که یک عامل انجام می‌دهد باعث افزایش سودمندی دیگر عامل‌ها می‌شود.

- در هوش مصنوعی، “بازی‌ها” نوع خاصی از مسائل به شمار می‌روند.
- فرضیات اصلی در مورد بازی‌های هوش مصنوعی
 - دو نفره
 - نوبتی
 - هر عامل به نوبت عمل انجام می‌دهد
- مجموع صفر Zero-Sum
- اهداف عامل متناقض است مجموع مقادیر سودمندی در پایان بازی صفر یا مقدار ثابتی است.
- قطعی
- با اطلاعات کامل Perfect information
- کاملاً مشاهده‌پذیر

بازی به عنوان نوعی از مسأله جستجو

- حالت اولیه s_0 : موقعیت اولیه بازی را مشخص می کند.
- $Player(s)$: در وضعیت s نوبت کدام بازیکن است.
- $Actions(s)$: مجموعه اعمال قانونی در وضعیت s را برمی گرداند.
- $Result(s,a)$: مدل انتقال، نتیجه یک حرکت را تعریف می کند.
- $Terminal-Test(s)$: آزمون پایانی هنگامی که بازی تمام شده باشد درست و در غیر این صورت غلط برمی گرداند.
- $Utility(s,p)$: مقدار سودمندی بازیکن p در **حالت پایانی** s چقدر است.
- بازی مجموع صفر (مجموع ثابت): مجموع مقدار سودمندی تمام بازیکنان در حالت پایانی s برابر با صفر یا یک مقدار ثابت است.

بازی به عنوان نوعی از مسأله جستجو ...

- درخت بازی = حالت اولیه + تابع اقدامات + تابع نتیجه
- درختی که گره‌ها در آن وضعیت‌های بازی هستند و یال‌ها حرکات
- چون رقیب غیر قابل پیش‌بینی است باید یک حرکت **برای هر پاسخ ممکن** از طرف رقیب مشخص نمود
- مانند جستجوی AND-OR
- در ادامه فرض می‌کنیم
- دو بازیکن MAX و MIN داریم که شروع‌کننده‌ی بازی MAX است.
- هدف ما یافتن بهترین عملی است که MAX می‌تواند انجام دهد تا بیشترین سودمندی را به دست آورد.

استراتژی کمینه بیشینه (MINIMAX)

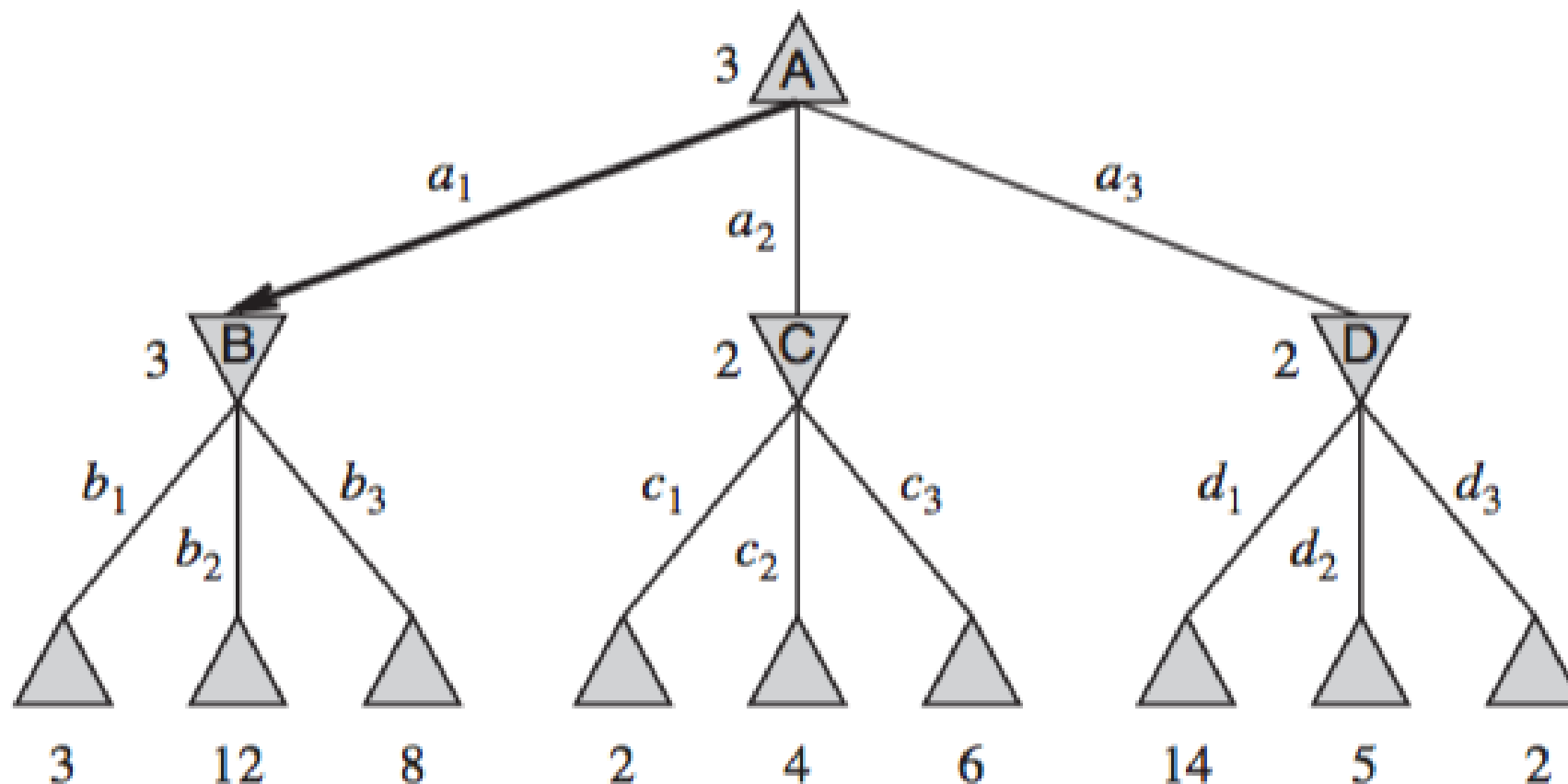
- با داشتن درخت بازی، استراتژی بهینه را می توان با در نظر گرفتن مقدار **minimax** گره ها تعیین نمود.
- تابع $\text{MINIMAX}(s)$ بهترین نتیجه ی بودن در وضعیت S را مشخص می کند. با فرض آن که هر دو بازیکن از گره شروع تا پایان بهینه بازی کنند.
- MAX به دنبال بیشینه کردن مقدار سودمندی خود و MIN به دنبال کمینه کردن مقدار سودمندی حریف است.

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

کمینه‌بیشینه (مثال ۱)

MAX

MIN

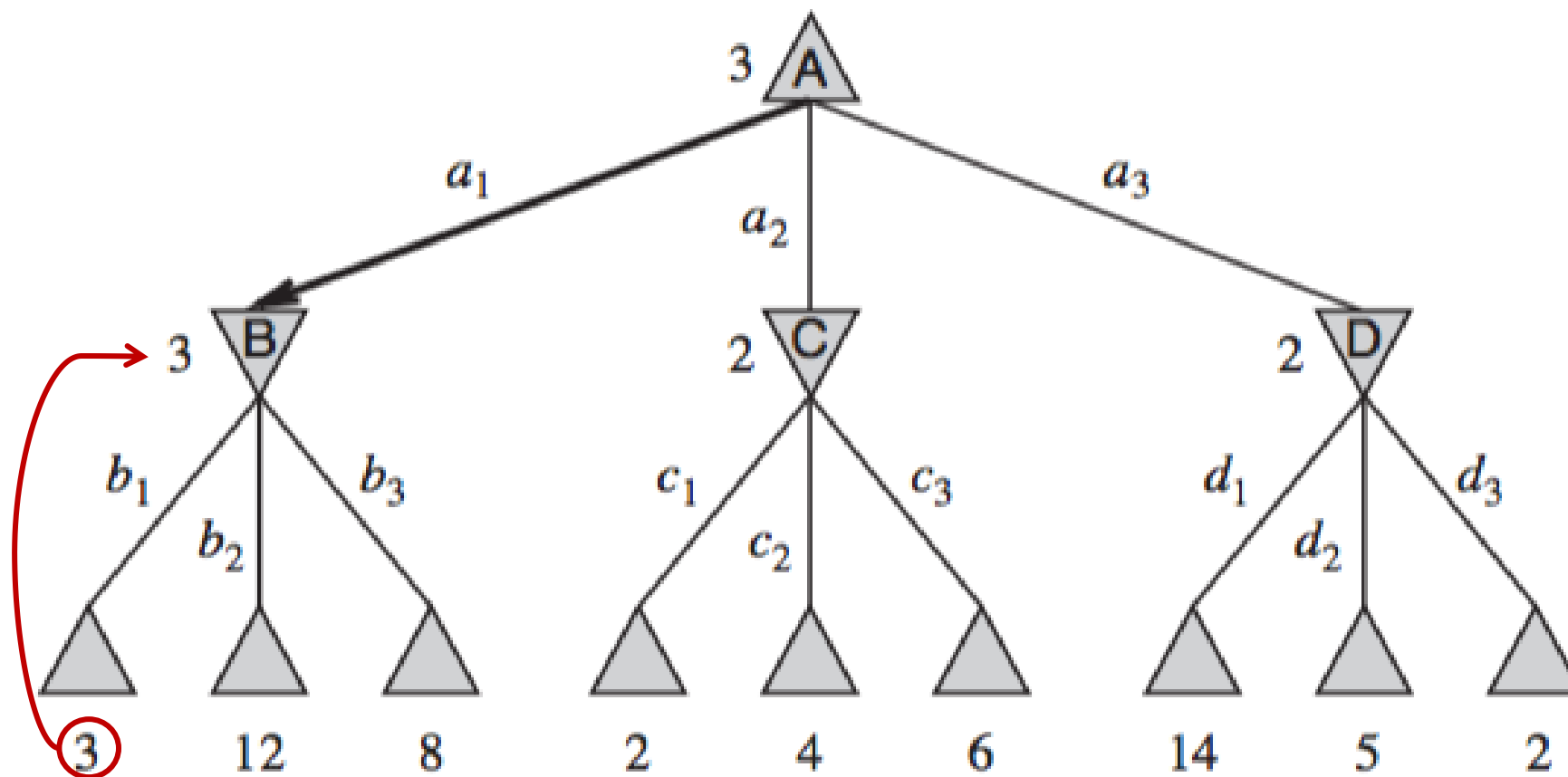


MINIMAX بدترین نتیجه برای MAX را بیشینه می‌کند چون فکر می‌کند همیشه MIN می‌خواهد بهینه عمل کند.

کمینه‌بیشینه (مثال ۱)

MAX

MIN

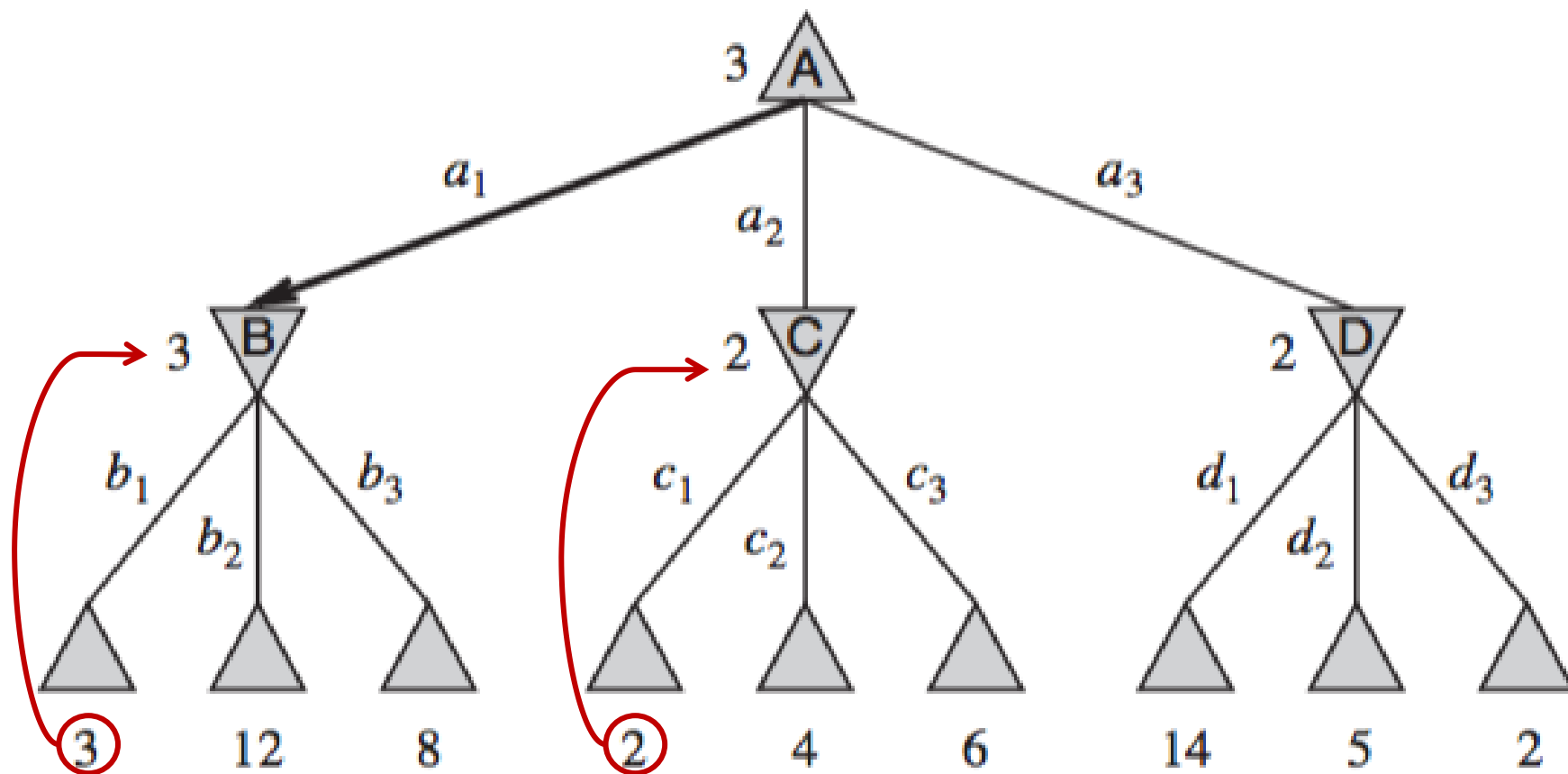


MINIMAX بدترین نتیجه برای MAX را بیشینه می‌کند چون فکر می‌کند همیشه MIN می‌خواهد بهینه عمل کند.

کمینه‌بیشینه (مثال ۱)

MAX

MIN

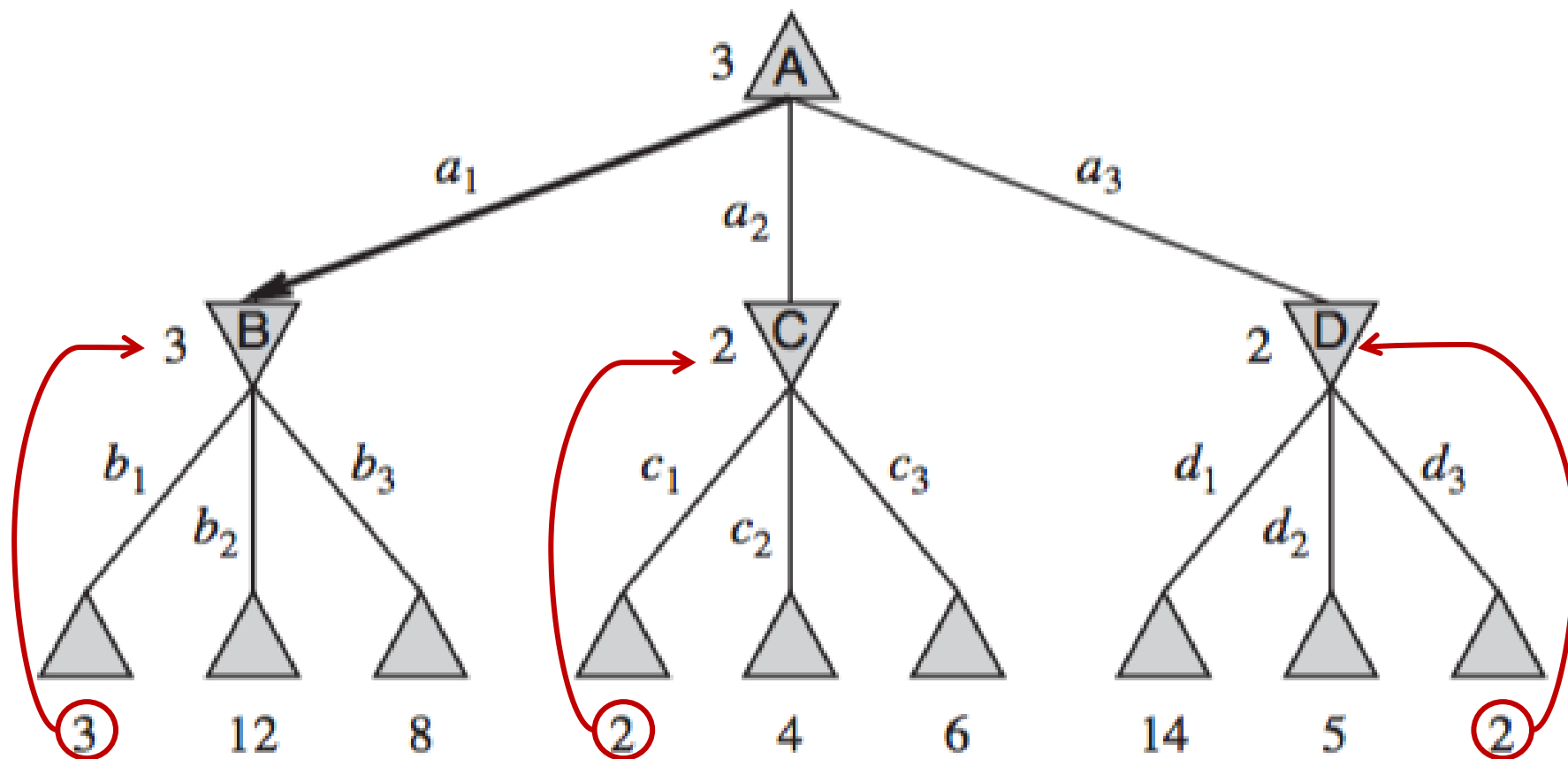


MINIMAX بدترین نتیجه برای MAX را بیشینه می‌کند چون فکر می‌کند همیشه MIN می‌خواهد بهینه عمل کند.

کمینه‌بیشینه (مثال ۱)

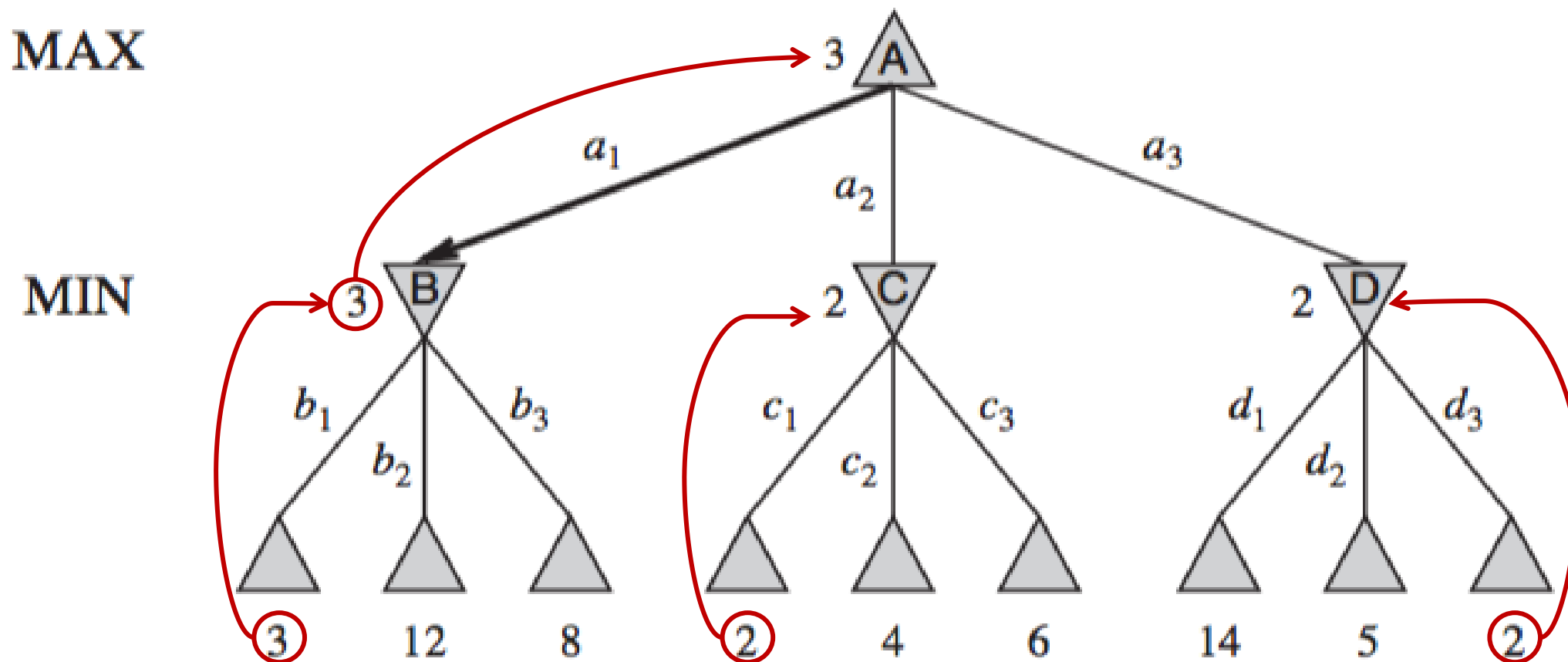
MAX

MIN



MINIMAX بدترین نتیجه برای MAX را بیشینه می‌کند چون فکر می‌کند همیشه MIN می‌خواهد بهینه عمل کند.

کمینه‌بیشینه (مثال ۱)



MINIMAX بدترین نتیجه برای MAX را بیشینه می‌کند چون فکر می‌کند همیشه MIN می‌خواهد بهینه عمل کند.

function MINIMAX-DECISION(*state*) **returns** *an action*
 return $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(state, a))$

function MAX-VALUE(*state*) **returns** *a utility value*
 if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
 $v \leftarrow -\infty$
 for each *a* **in** ACTIONS(*state*) **do**
 $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$
 return *v*

function MIN-VALUE(*state*) **returns** *a utility value*
 if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
 $v \leftarrow \infty$
 for each *a* **in** ACTIONS(*state*) **do**
 $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$
 return *v*

الگوریتم کمینه بیشینه

function MINIMAX-DECISION(*state*) **returns** an action

return $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(state, a))$

function MAX-VALUE(*state*) **returns** a utility value

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

for each *a* **in** ACTIONS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$

return *v*

function MIN-VALUE(*state*) **returns** a utility value

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow \infty$

for each *a* **in** ACTIONS(*state*) **do**

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$

return *v*

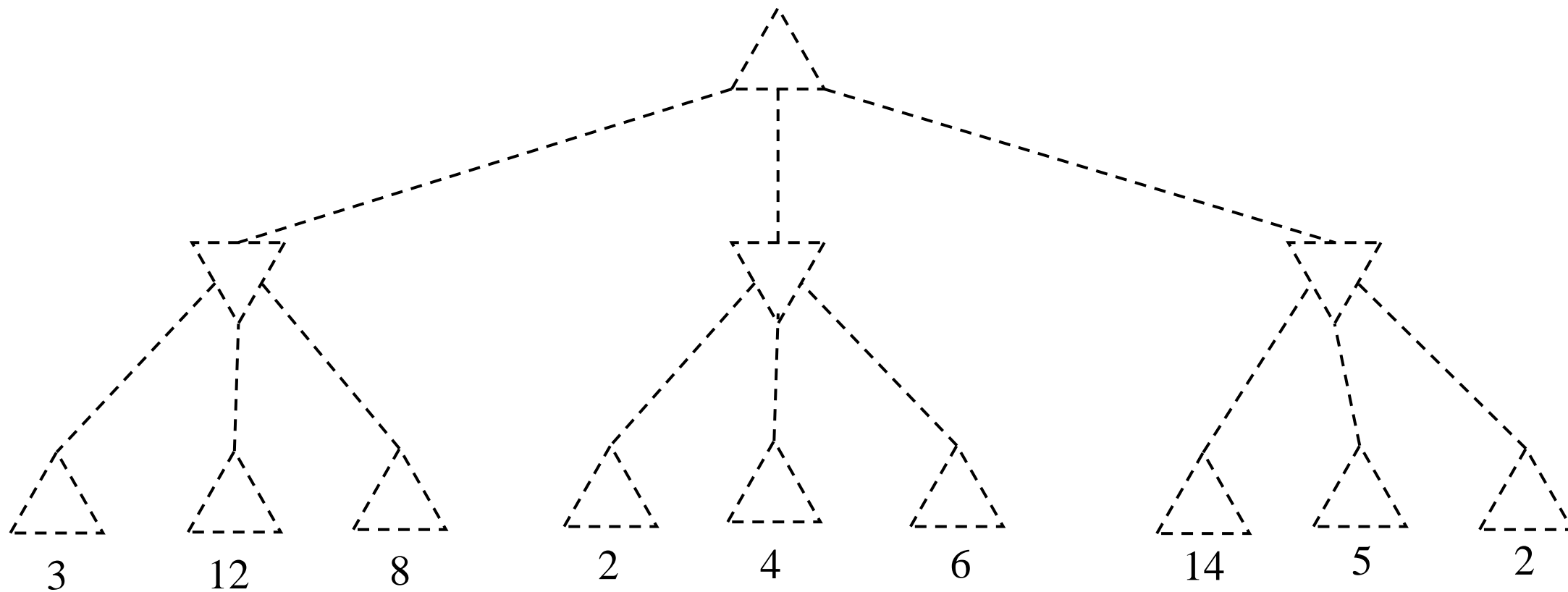
می توان با این قطعه
کد جایگزین نمود

$v \leftarrow \text{MAX-VALUE}(state)$

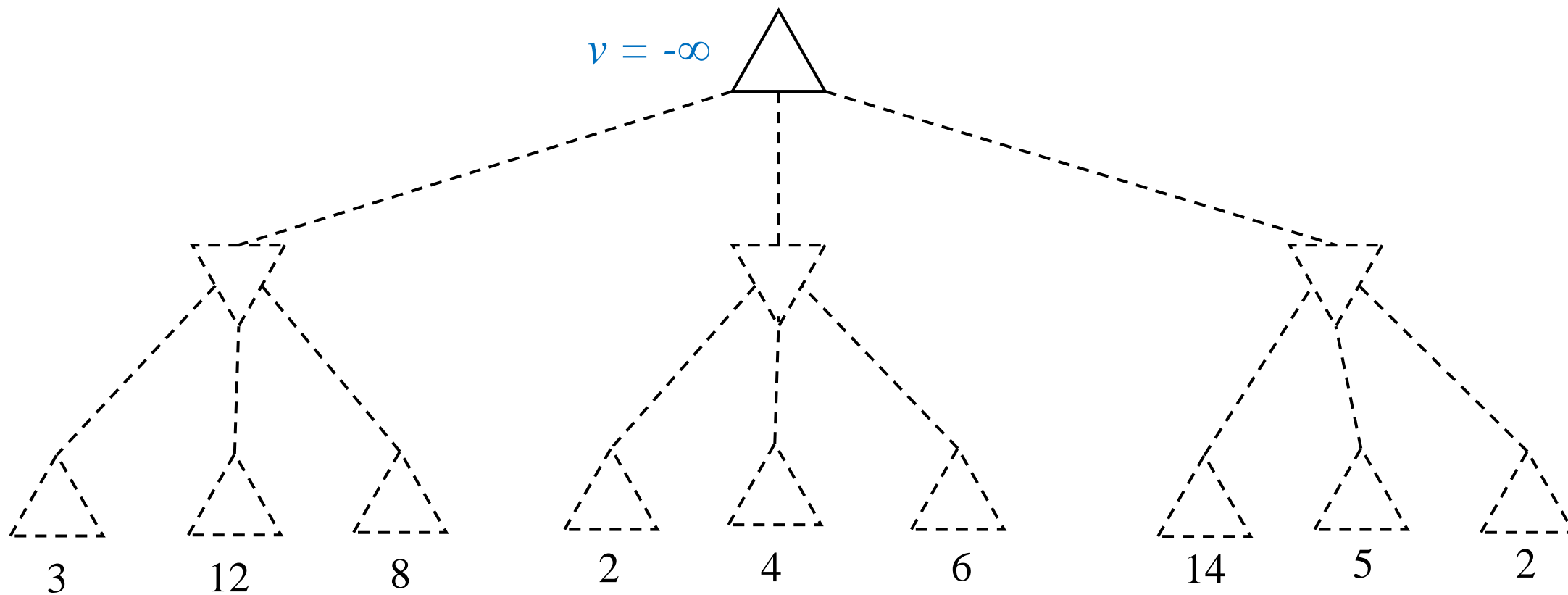
return the action in successors(*state*) with value *v*

در روند اجرای الگوریتم هر جا نود
MAX را دیدیم علامت $-\infty$ و هر جا
نود MIN را دیدیم علامت $+\infty$ قرار
می دهیم.

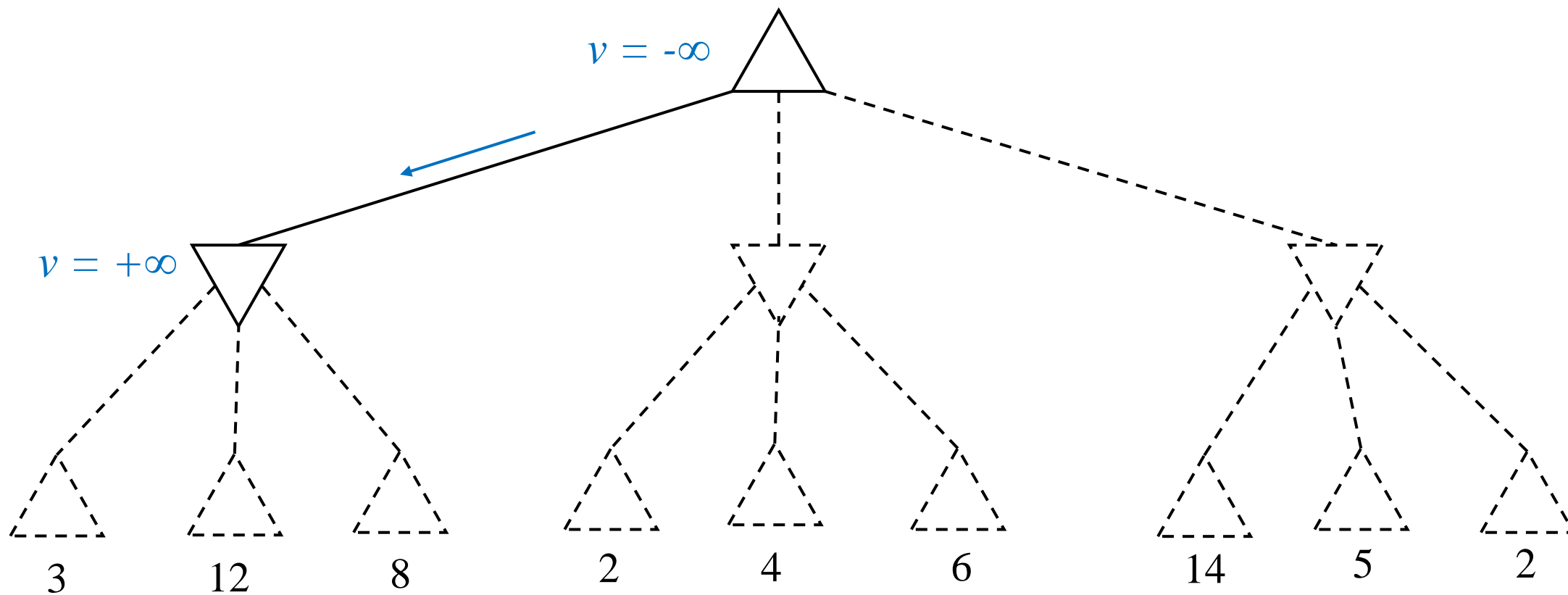
کمینہ پیشینہ (مثال ۱)



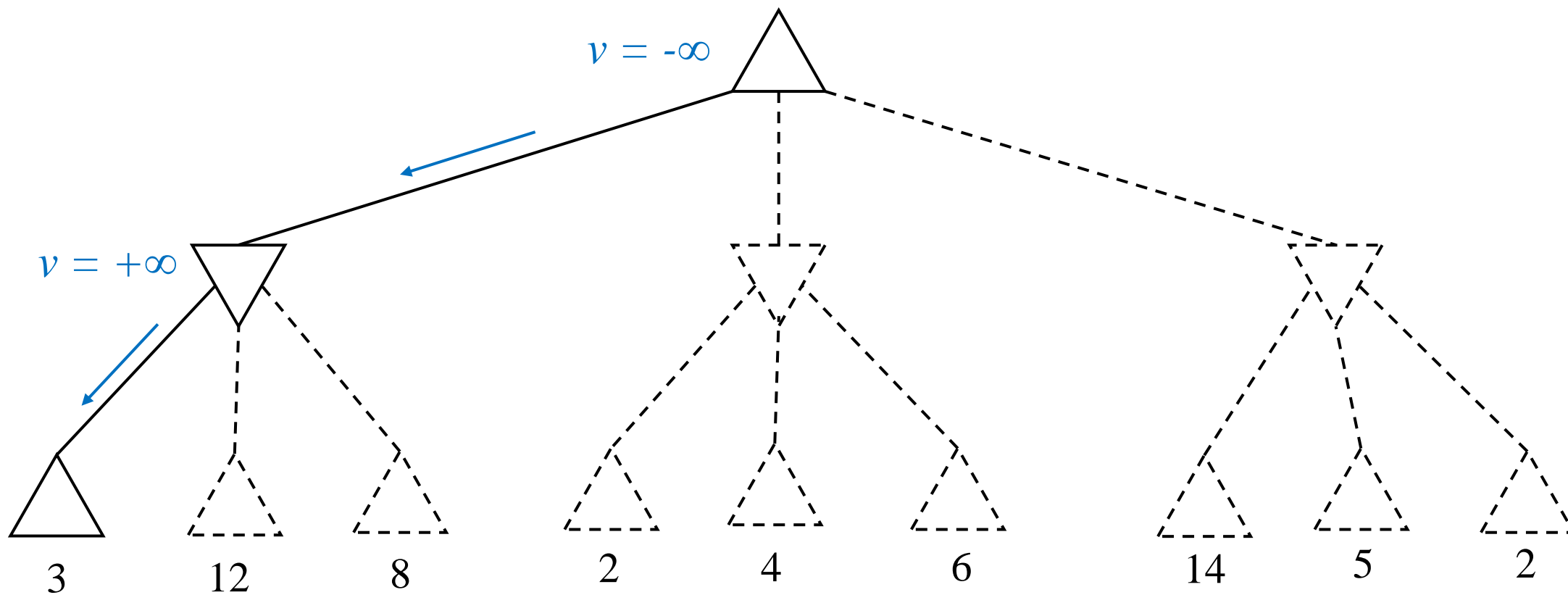
کمینه‌بیشینه (مثال ۱)



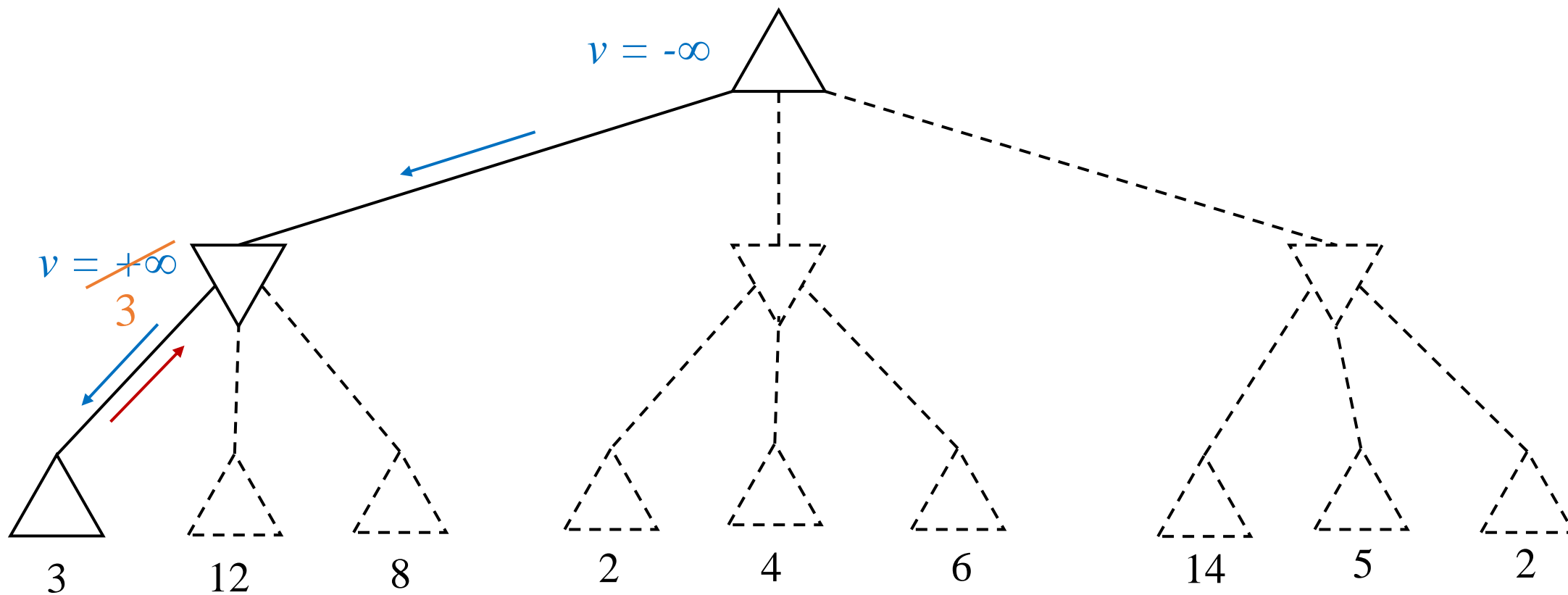
کمینه‌بیشینه (مثال ۱)



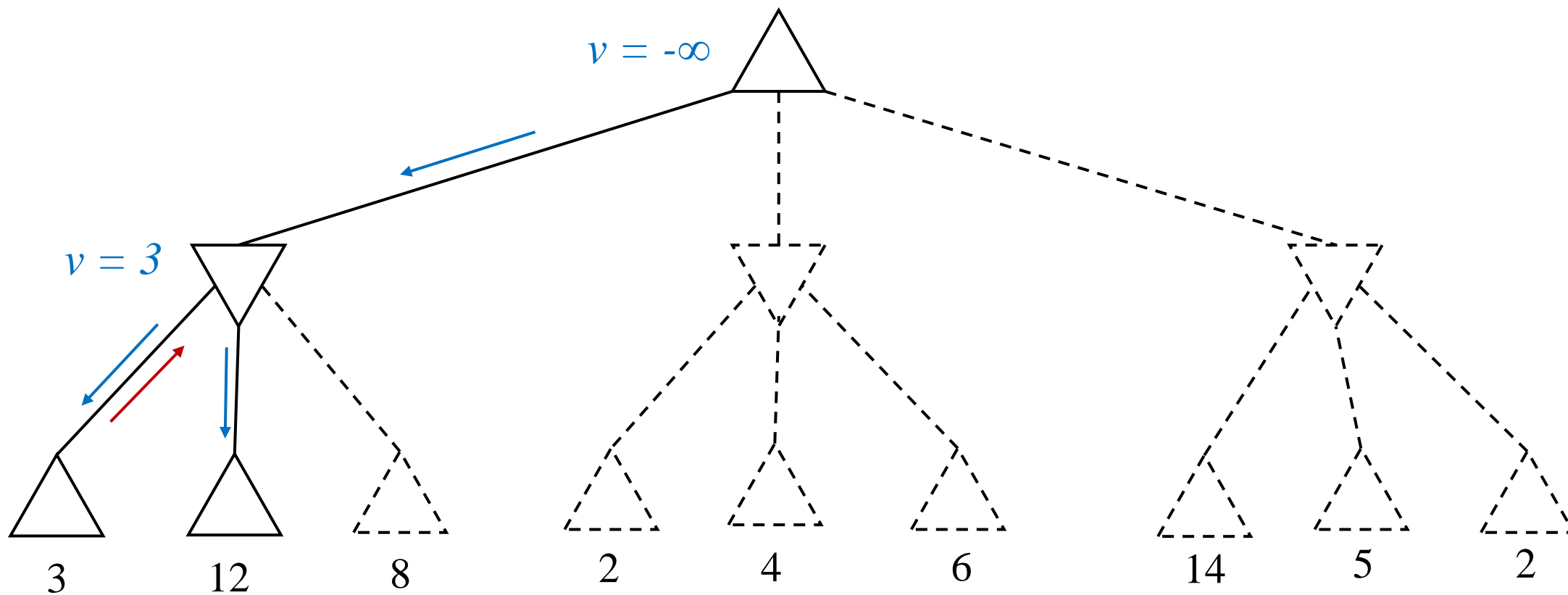
کمینه‌بیشینه (مثال ۱)



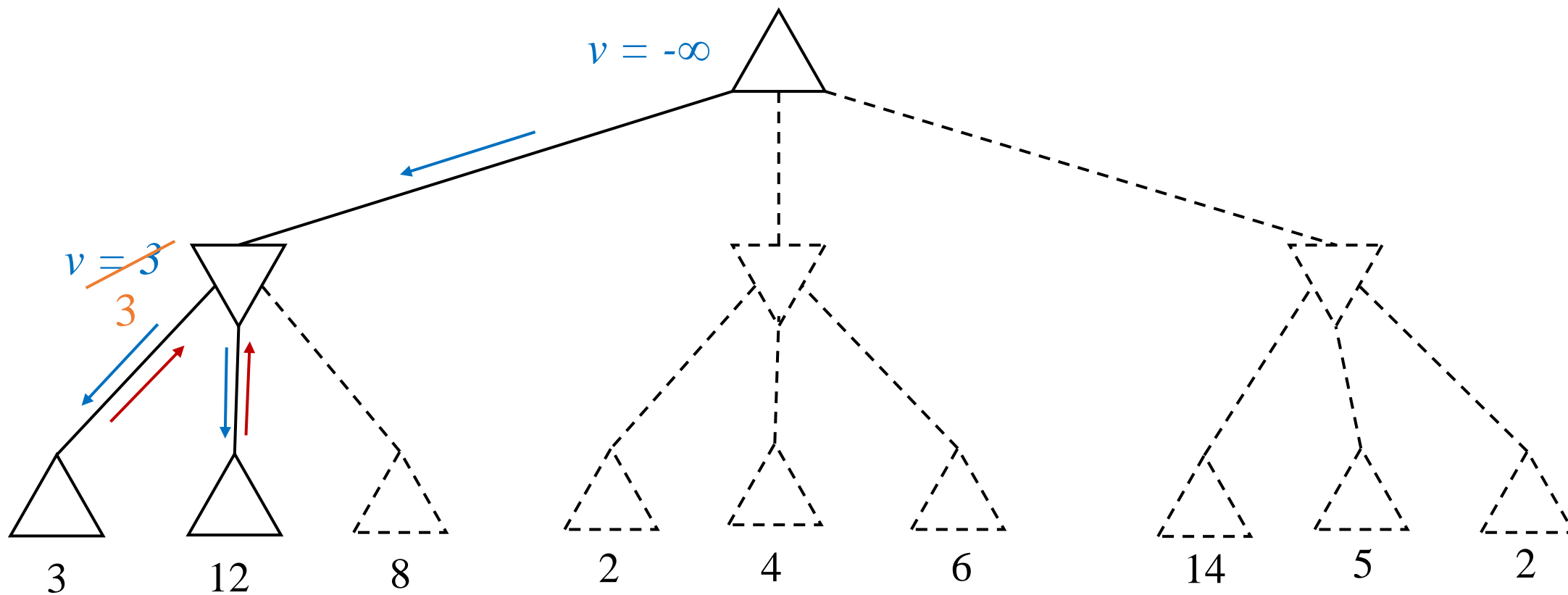
کمینه‌بیشینه (مثال ۱)



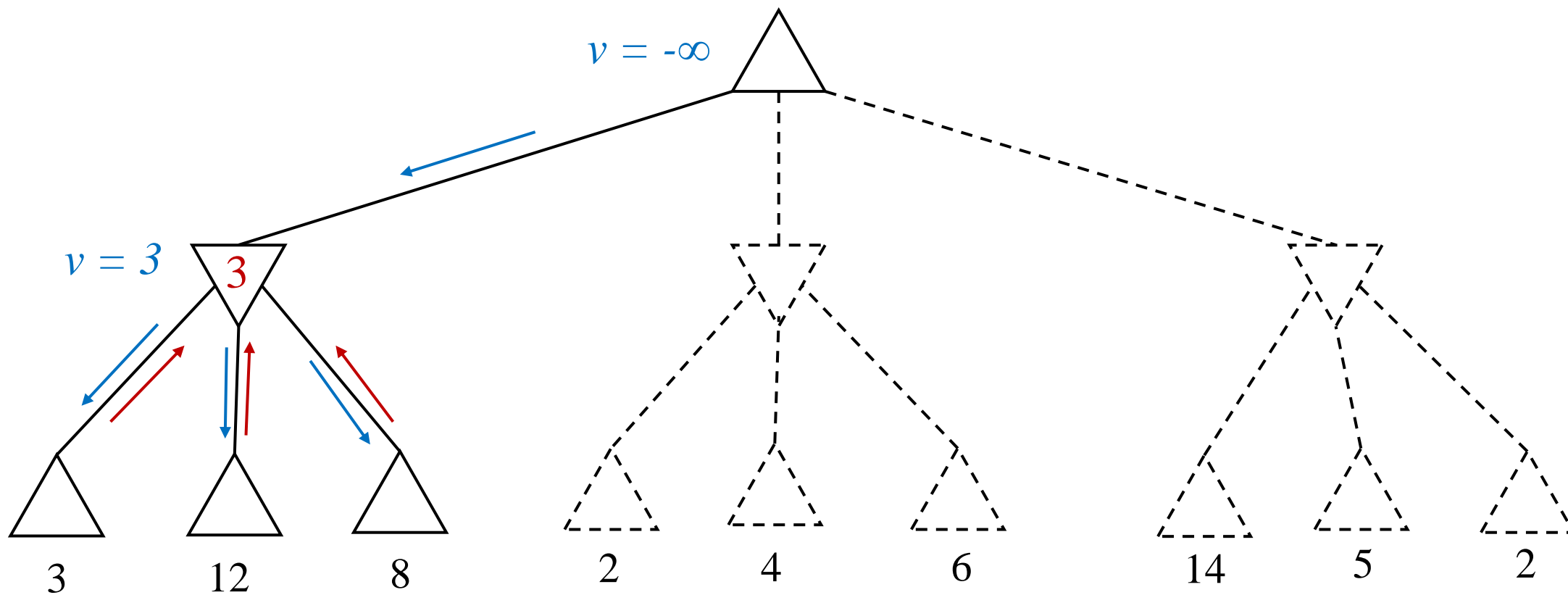
کمینه‌بیشینه (مثال ۱)



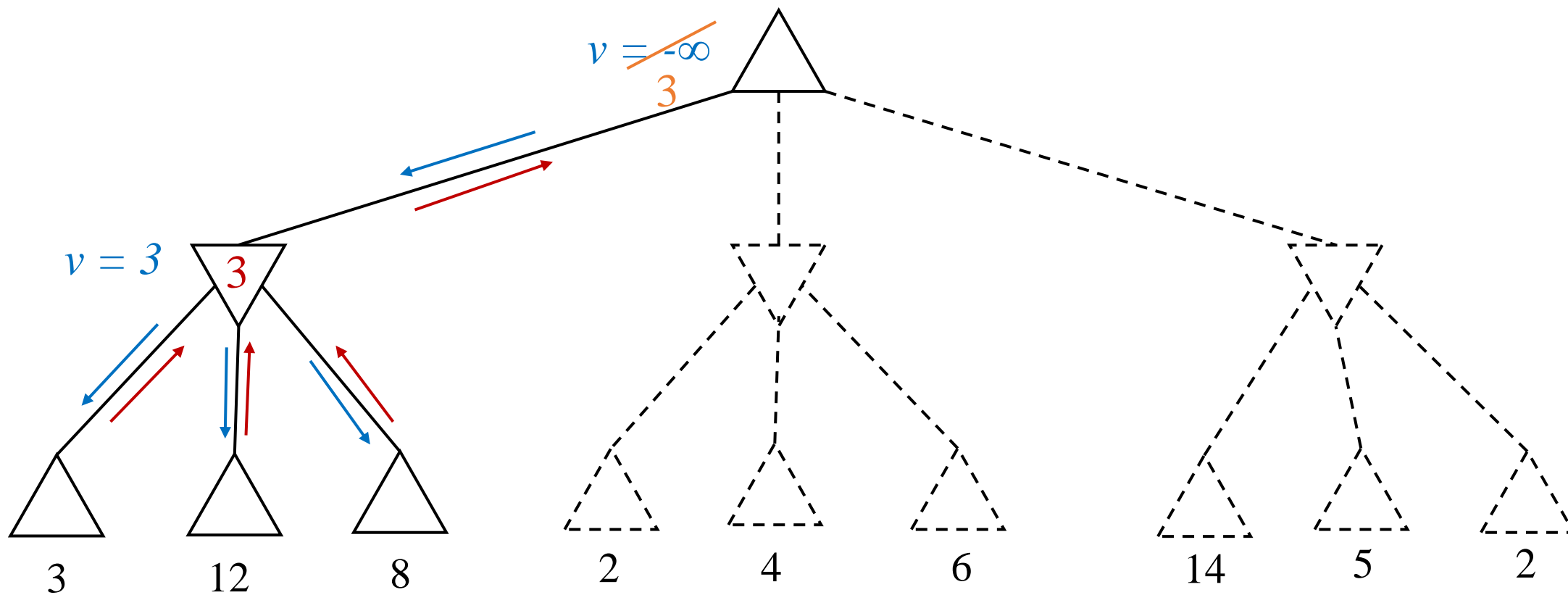
کمینه‌بیشینه (مثال ۱)



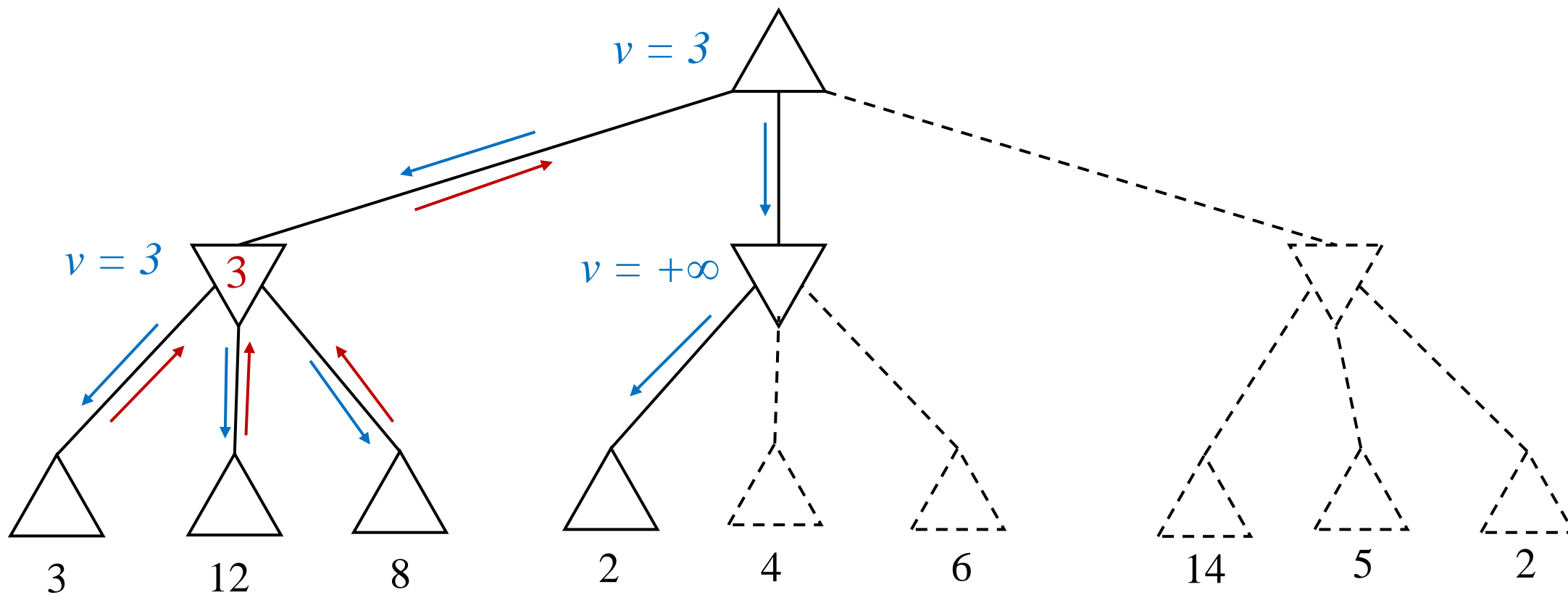
کمینه‌بیشینه (مثال ۱)



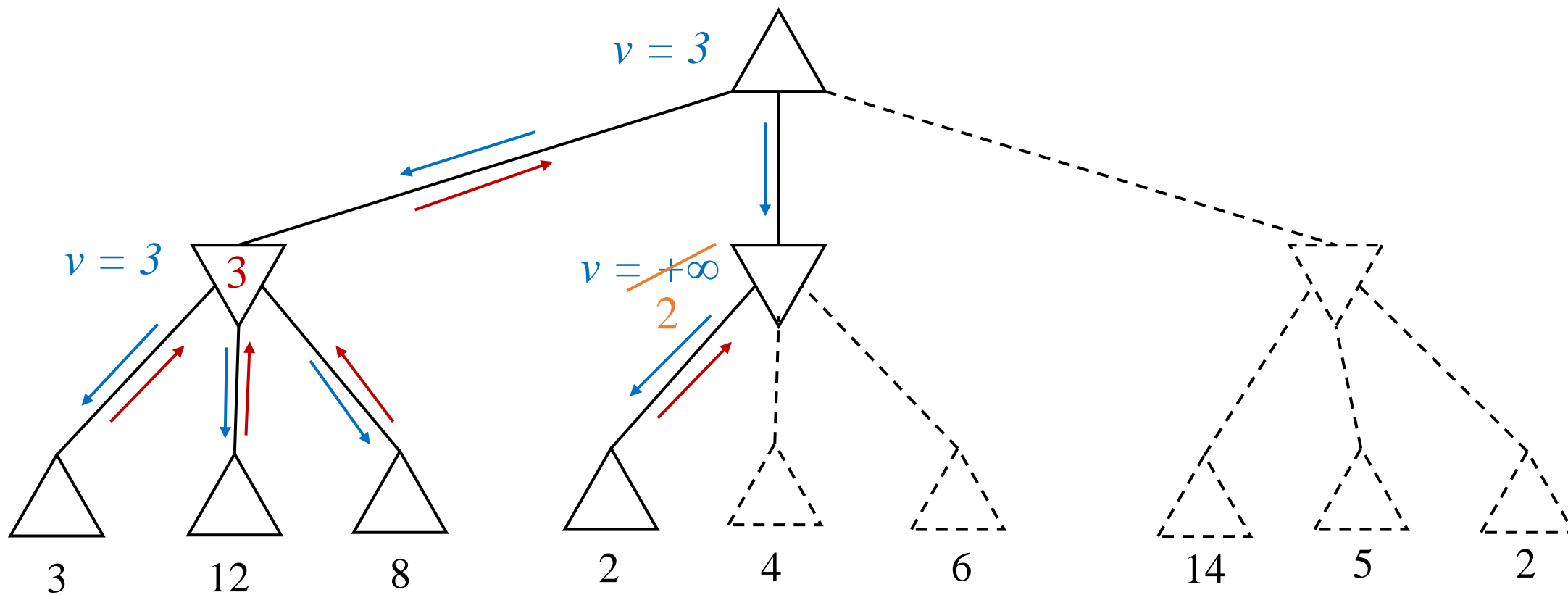
کمینه‌بیشینه (مثال ۱)



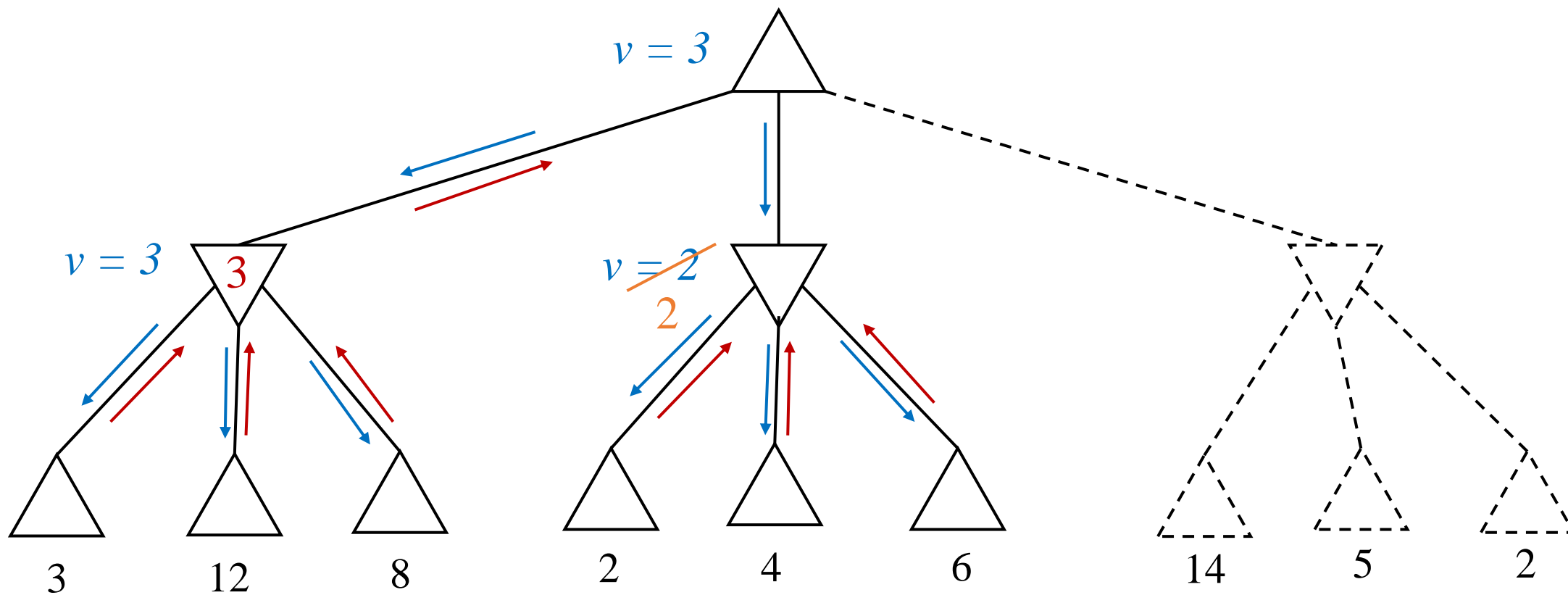
کمینه‌بیشینه (مثال ۱)



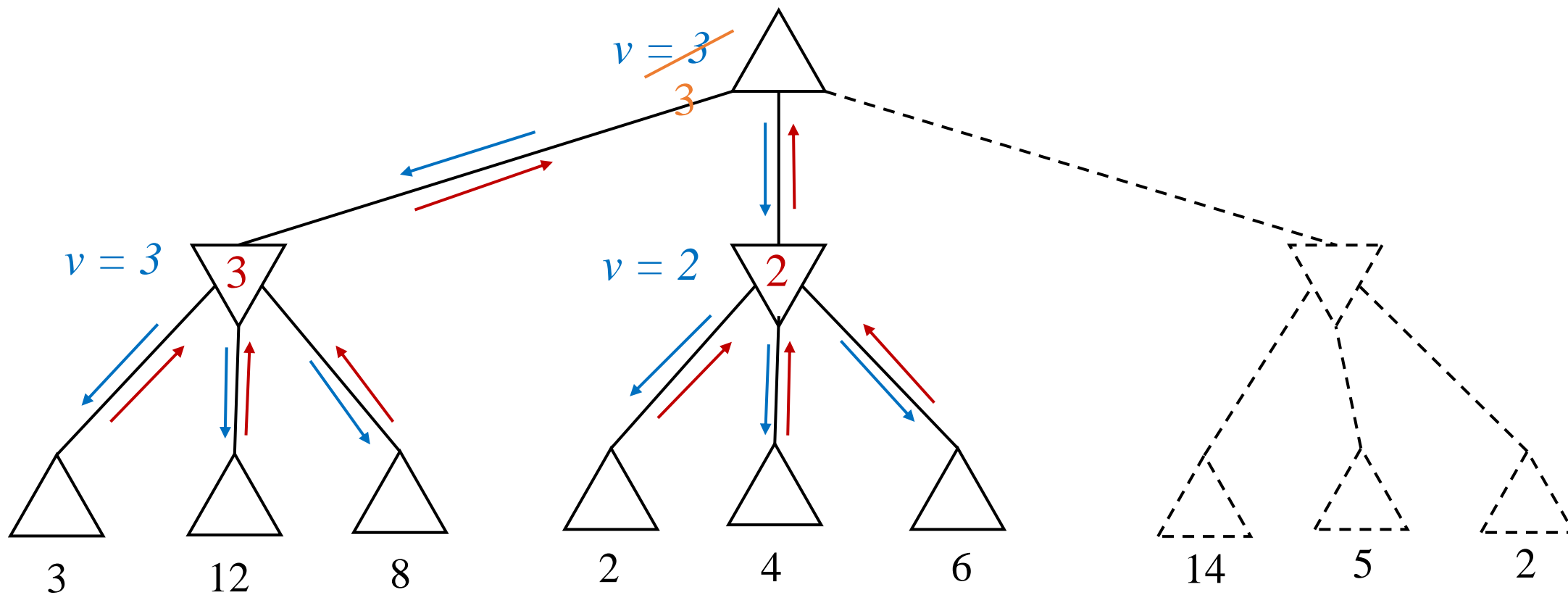
کمینه‌بیشینه (مثال ۱)



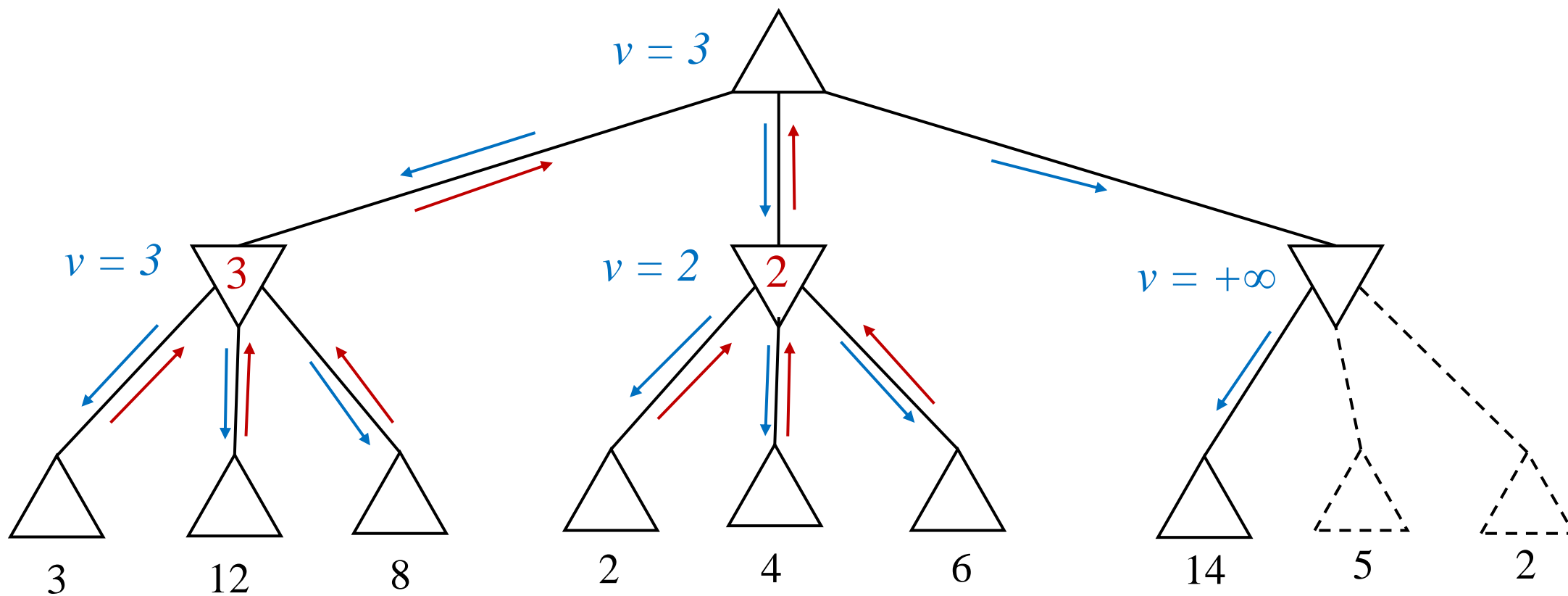
کمینه‌بیشینه (مثال ۱)



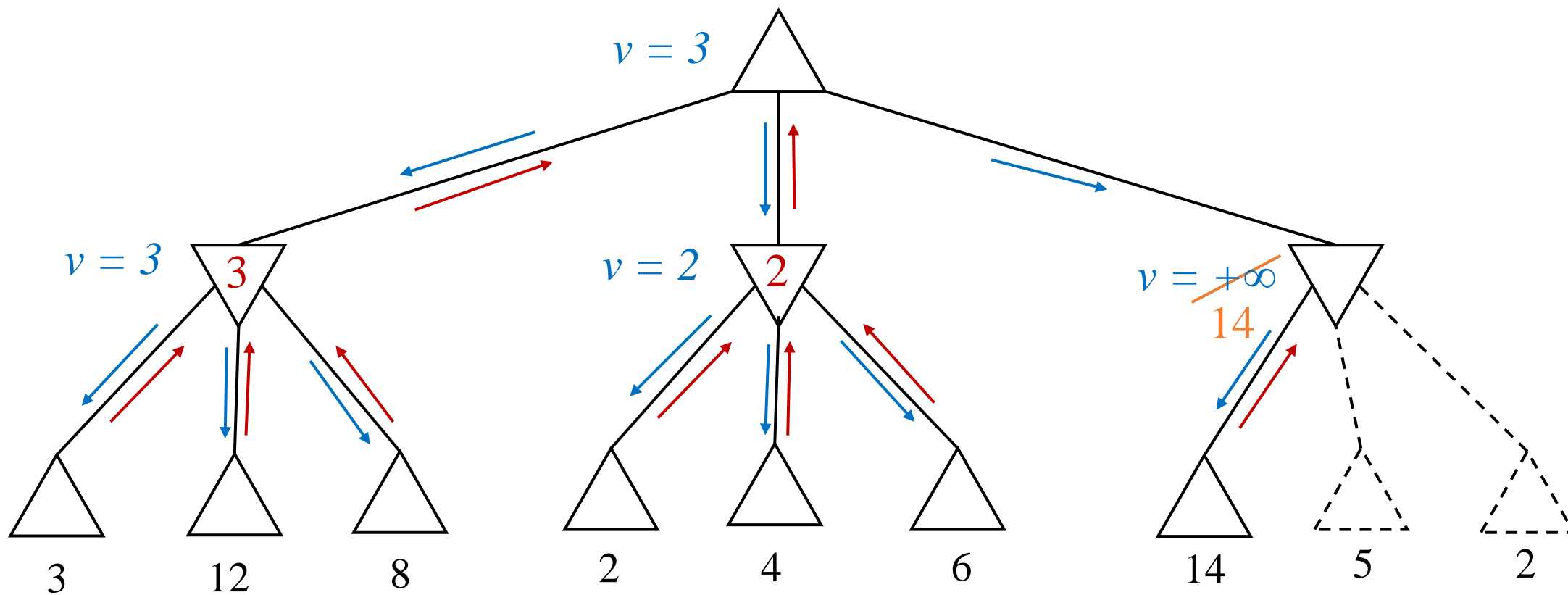
کمینه‌بیشینه (مثال ۱)



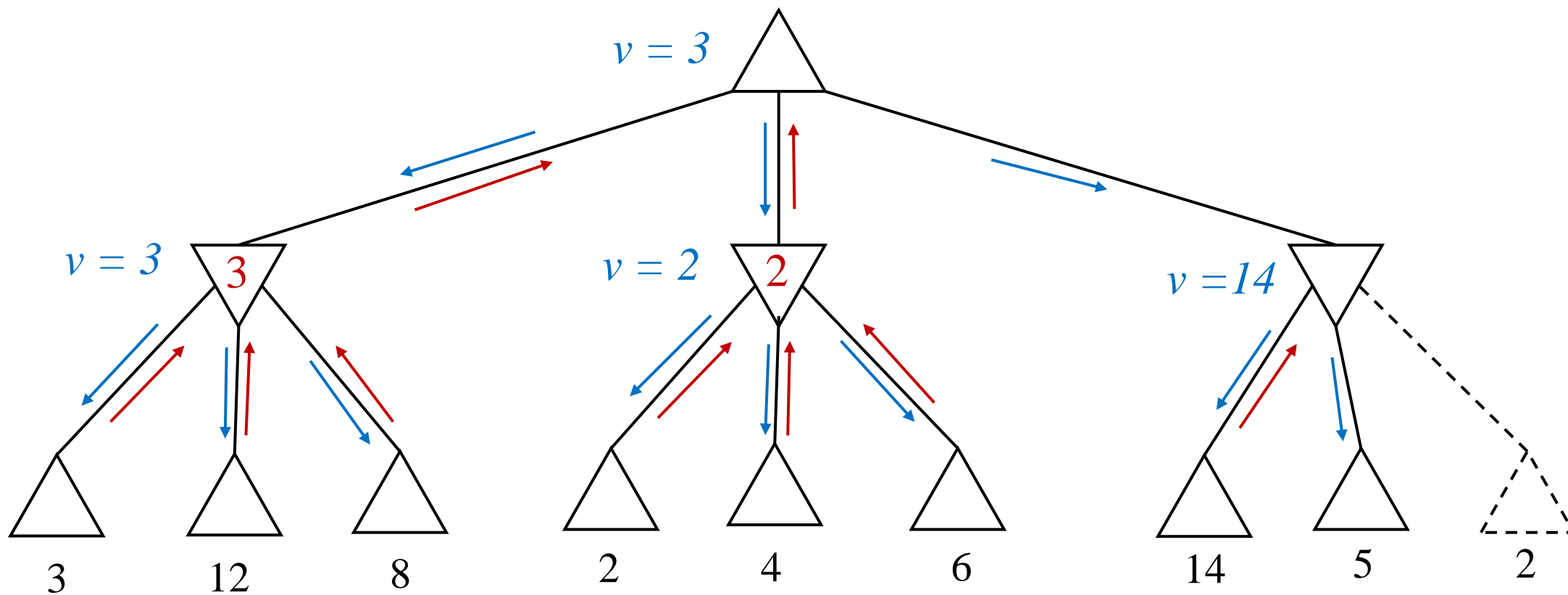
کمینه‌بیشینه (مثال ۱)



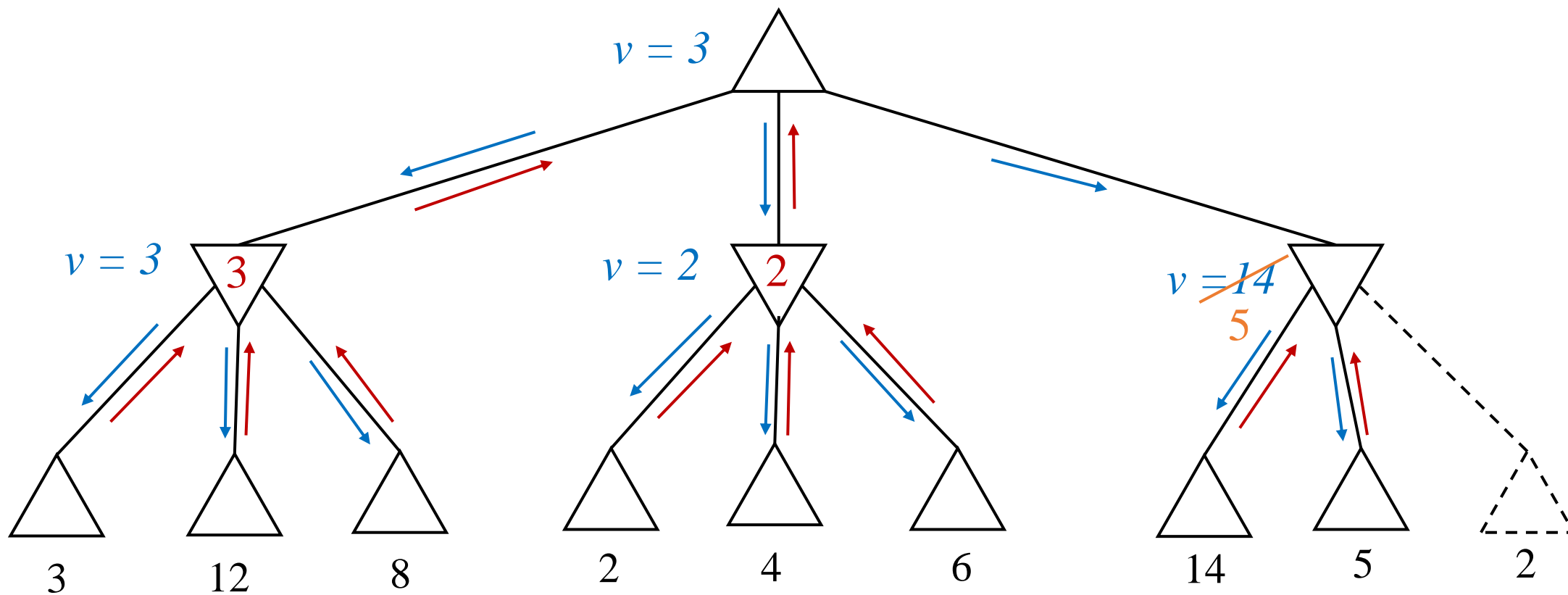
کمینه‌بیشینه (مثال ۱)



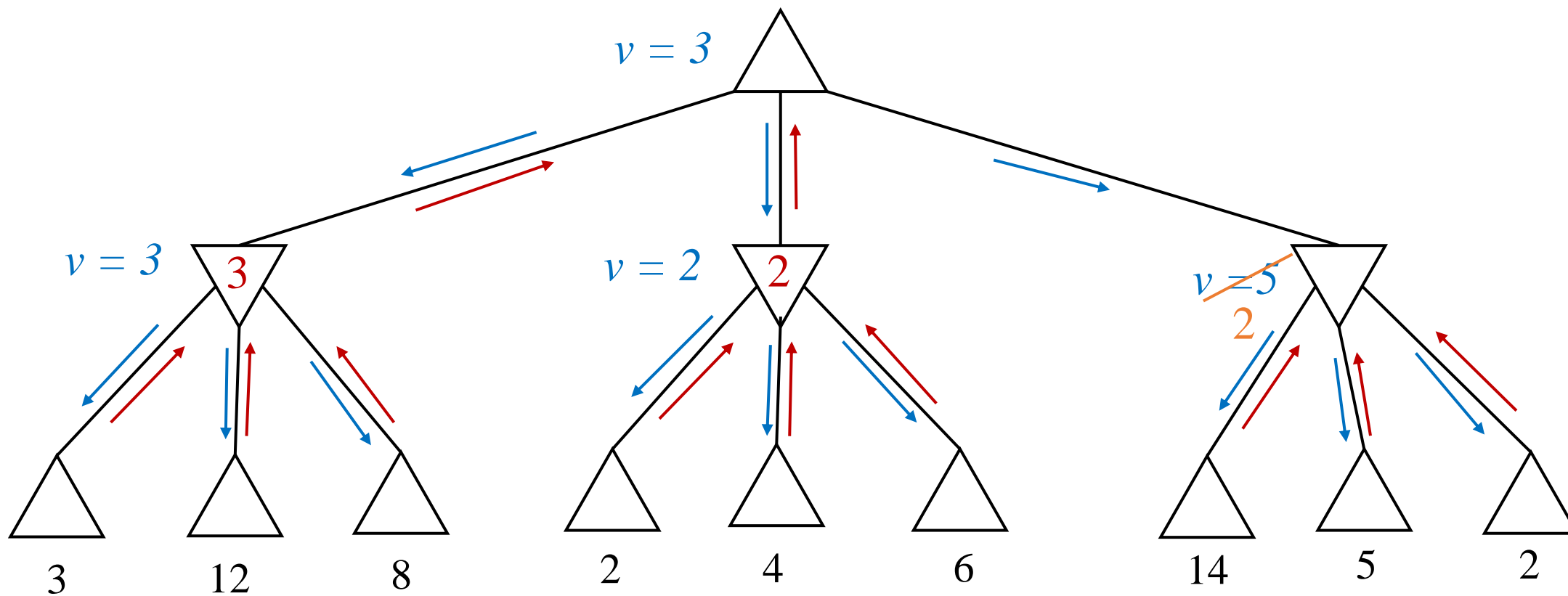
کمینه‌بیشینه (مثال ۱)



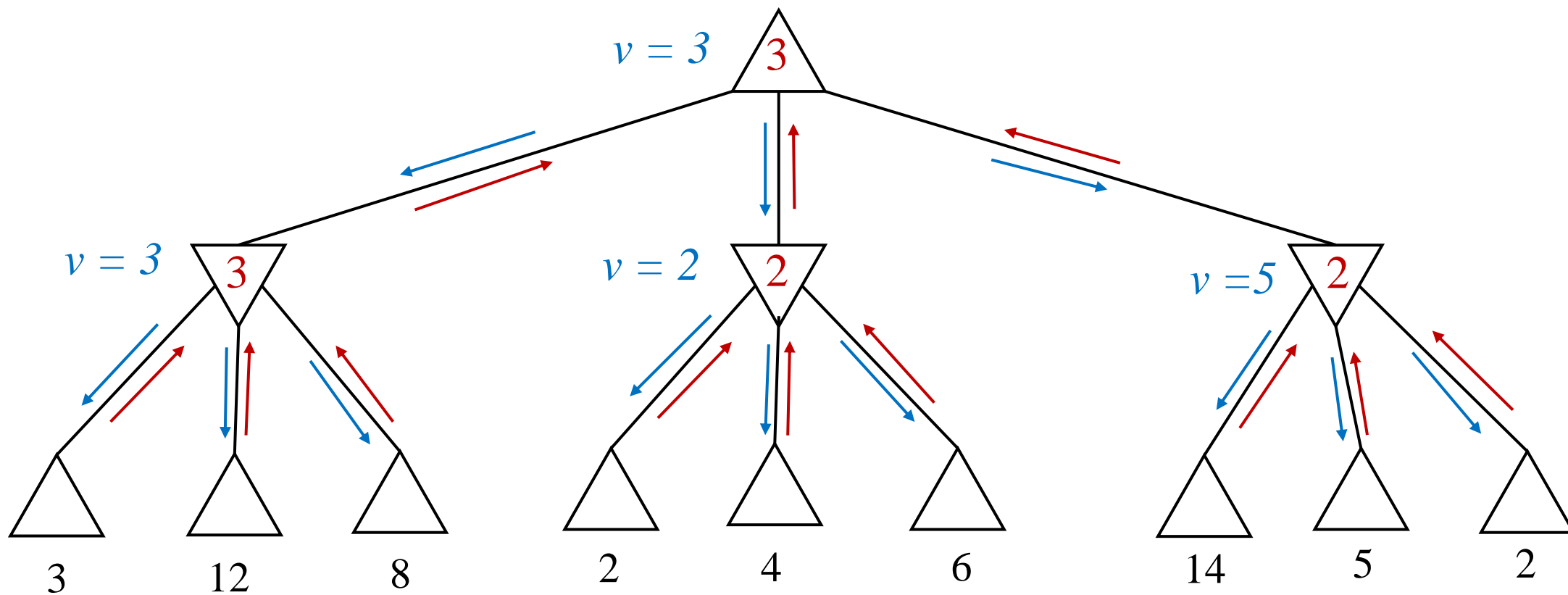
کمینه‌بیشینه (مثال ۱)



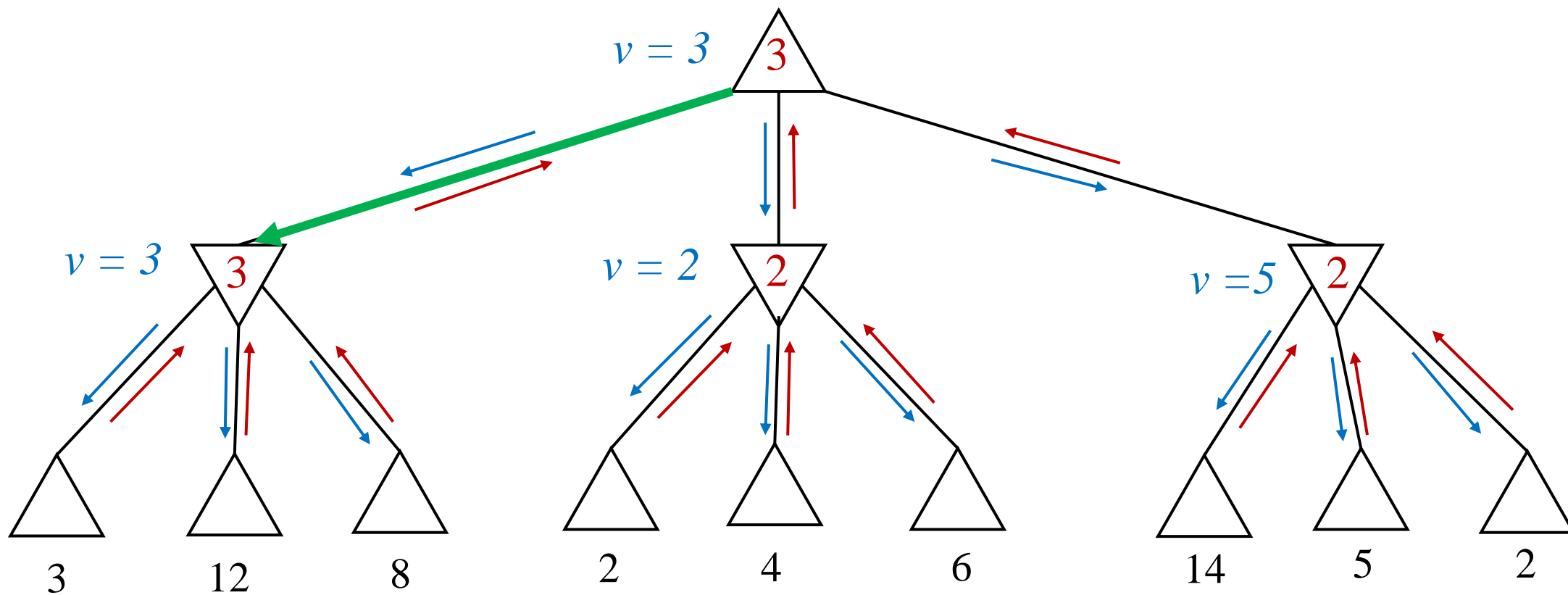
کمینه‌بیشینه (مثال ۱)



کمینه‌بیشینه (مثال ۱)



کمینه‌بیشینه (مثال ۱)



ویژگی‌های الگوریتم پیشینه کمینه

به صورت عمقی پیاده‌سازی شده است.

- کامل بودن؟ بله
- هنگامی که درخت متناهی باشد و حافظه به اندازه کافی موجود باشد.
- بهینه بودن؟ بله
- بهترین حرکت در مقابل یک بازیکن حرفه‌ای را انتخاب می‌کند.
- اگر حریف به طور بهینه بازی نکند چه پیش خواهد آمد؟
- پیچیدگی زمانی؟ $O(b^m)$
- راه حل دقیق برای بازی‌های واقعی کاملاً غیر قابل دسترس است.
- برای مثال در شطرنج $b \approx 35, m \approx 100$
- پیچیدگی فضایی؟ خطی $O(bm)$

بسط ایده‌ی بیشینه کمینه به بازی‌های چند نفره

- مقادیر اختصاص داده شده به هر گره را با یک بردار سودمندی با سائز تعداد بازیکنان جایگزین می‌کنیم.
- اگر سه بازیکن A ، B و C داشته باشیم بردار سودمندی برابر با $\langle V_A, V_B, V_C \rangle$ خواهد بود.
- برای حالات پایانی، این بردار حاوی مقادیر سودمندی آن حالت از نظر هر بازیکن است.
- بردار سودمندی برای هر گره برابر با برداری است که دارای مقدار سودمندی بیشتر برای بازیکنی باشد که در آن گره دارای حق انتخاب است.
- در بازی‌های چند نفره ممکن است بین بازیکن‌ها **اتحاد** و یا **همکاری** بوجود آید.
- اتحاد: حمله دو بازیکن ضعیف به بازیکن قوی‌تر
- همکاری: اگر ۱۰۰ بیشترین سودمندی ممکن باشد که تنها در یک حالت پایانه $\langle ۱۰۰, ۱۰۰ \rangle$ اتفاق می‌افتد بازیکن‌ها برای رسیدن به این وضعیت به‌طور خودکار همکاری می‌کنند.

بازی‌های چند نفره – مثال

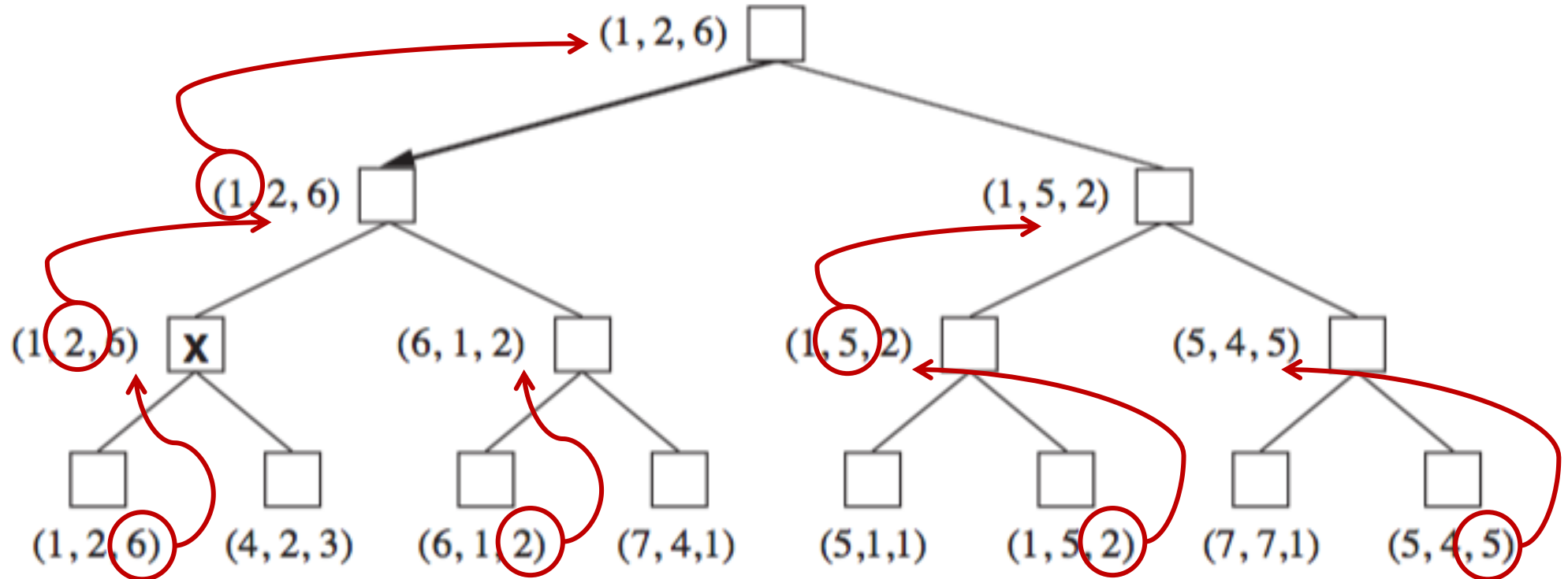
to move

A

B

C

A



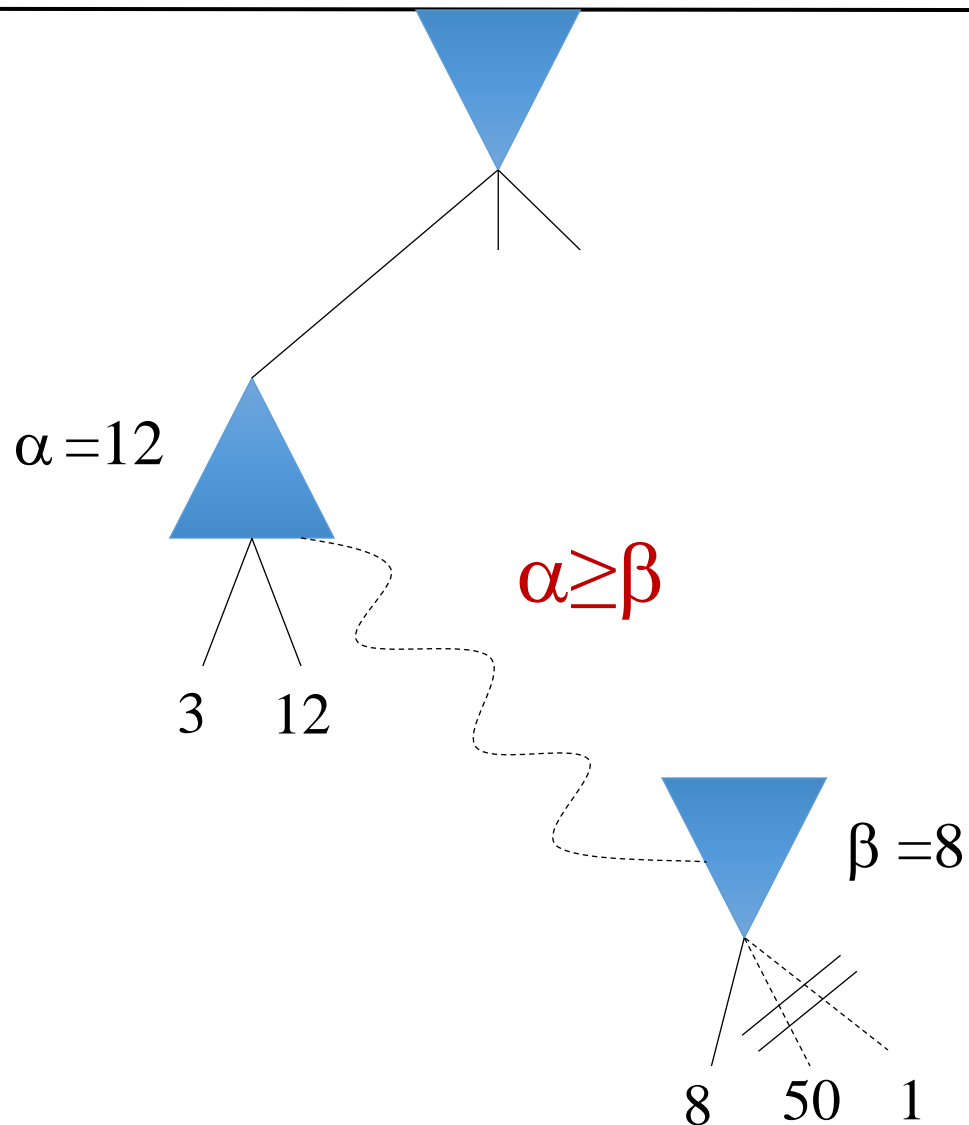
- مشکل الگوریتم بیشینه کمینه این است که تمام گره‌های درخت را بررسی می‌کند.
- این تعداد برحسب تعداد حرکات نمایی است در نتیجه زمان زیادی می‌برد.
- به هیچ طریقی نمی‌توان رابطه‌ی نمایی را از بین برد اما می‌توان با هرس تعداد حالات بررسی را تقریباً به نصف کاهش داد.
- ایده هرس کردن
- عدم بررسی برخی شاخه‌ها و افزایش سرعت در تصمیم‌گیری
- هرس آلفا-بتا که به یک درخت بیشینه کمینه استاندارد اعمال می‌شود، همان جواب الگوریتم بیشینه کمینه را برمی‌گرداند با این تفاوت که در این روش، شاخه‌هایی که در تصمیم‌گیری نهایی تأثیری ندارند، هرس می‌شود.

حقایق هرس آلفا-بتا

دو حقیقت زیر را می‌توان به راحتی از روش MINIMAX متوجه شد.

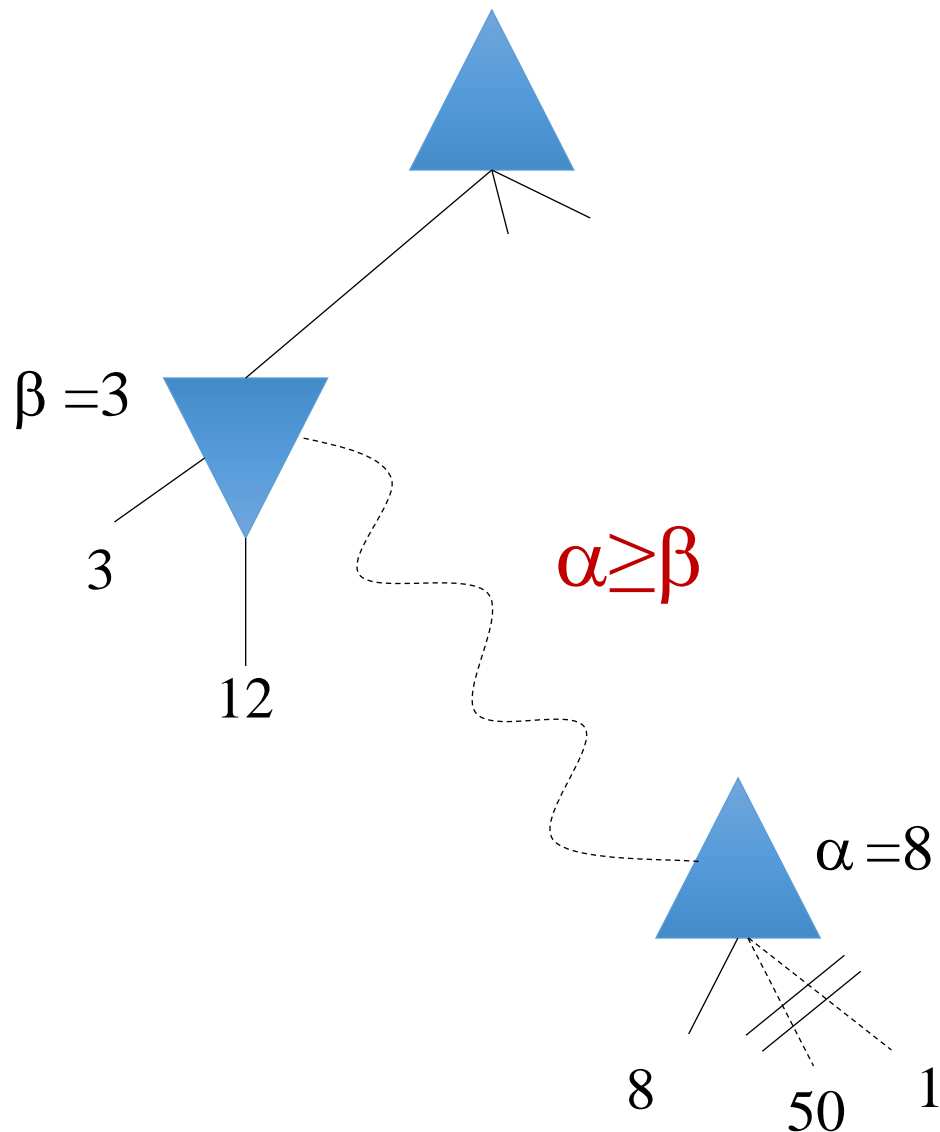
- مقادیر آلفای گره‌های MAX هیچ‌گاه کاهش نمی‌یابند.
- به گره‌های MAX مقادیر موقت آلفا نسبت داده می‌شود که این مقادیر با دیدن هر یک از فرزندانش همیشه باید در حال افزایش باشد نه کاهش.
- مقدار اولیه آلفا $= -\infty$
- مقادیر بتای گره‌های MIN هیچ‌گاه افزایش نمی‌یابند.
- به گره‌های MIN مقادیر موقت بتا نسبت داده می‌شود که این مقادیر با دیدن هر یک از فرزندانش همیشه باید در حال کاهش باشد نه افزایش.
- مقدار اولیه بتا $= +\infty$

قانون اول – هرس آلفا



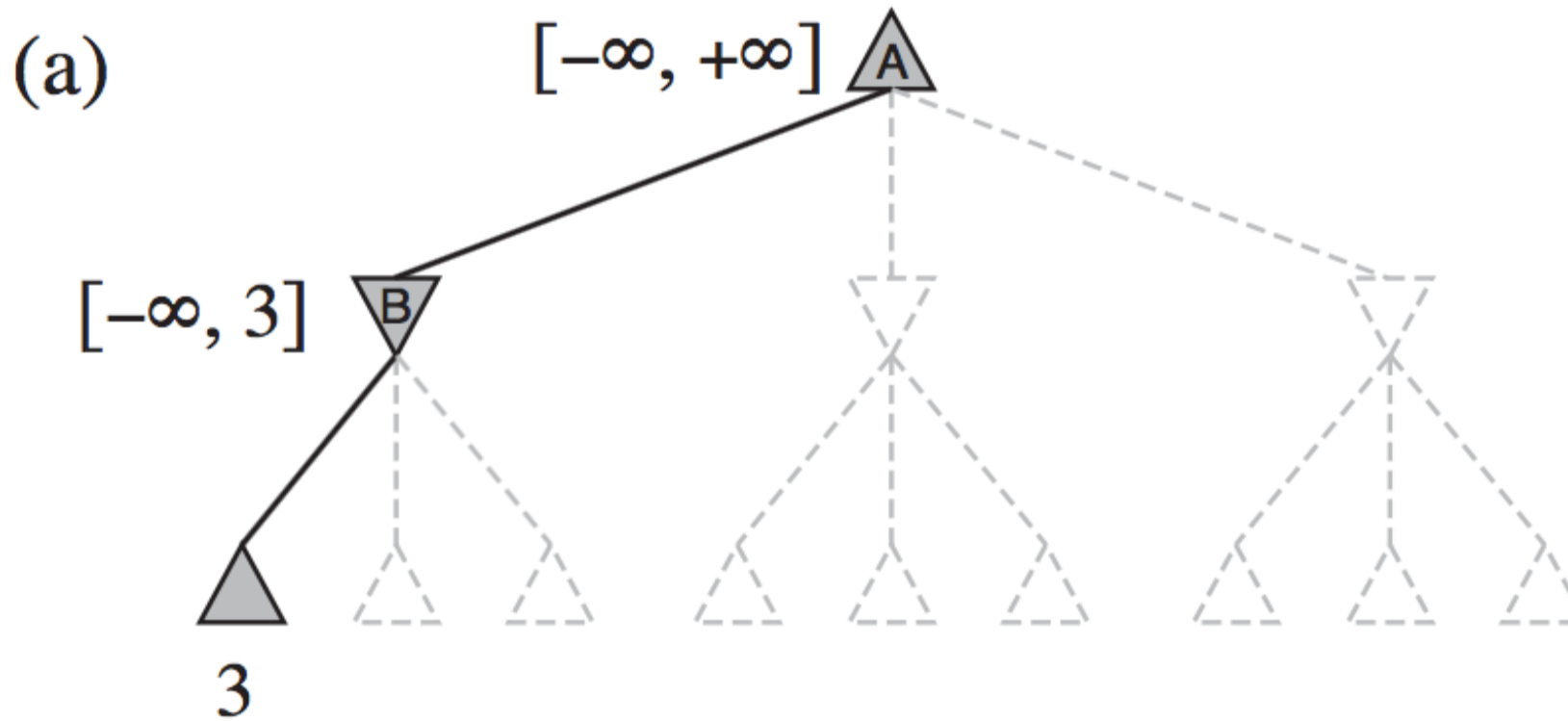
- عمل جستجو تحت گره MIN که مقدار بتای آن کوچکتر یا مساوی مقدار آلفای هر گره MAX اجداد آن گره است، قطع می‌شود.
- در این حالت، مقداری که تا به حال به گرهی MIN نسبت داده شده به عنوان مقدار نهایی بتای آن گره انتخاب خواهد شد.

قانون دوم – هرس بتا

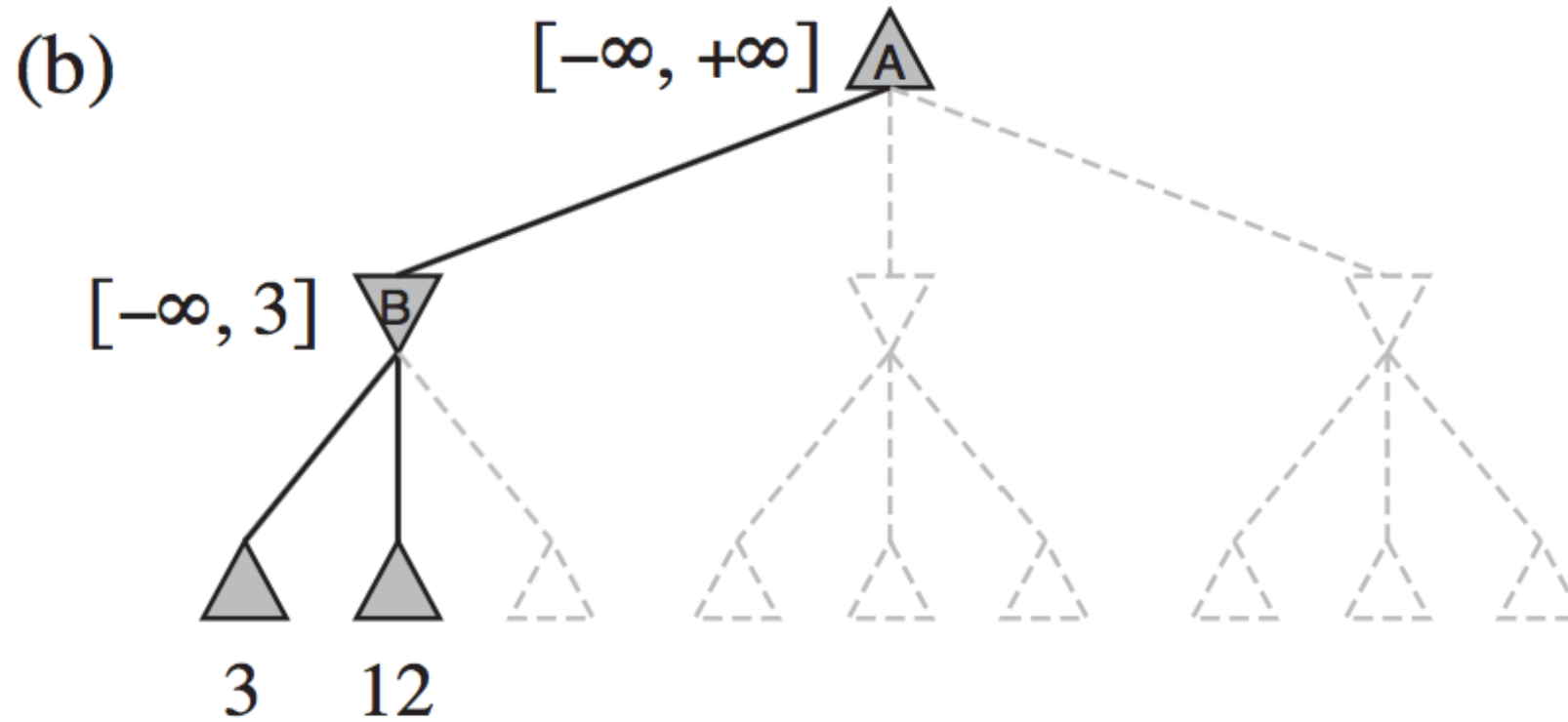


- عمل جستجو تحت گره MAX که مقدار آلفای آن بزرگتر یا مساوی مقدار بتای هر گره MIN اجداد آن گره است، قطع می‌شود.
- در این حالت، مقداری که تا به حال به گرهی MAX نسبت داده شده به عنوان مقدار نهایی آلفای آن گره انتخاب خواهد شد.

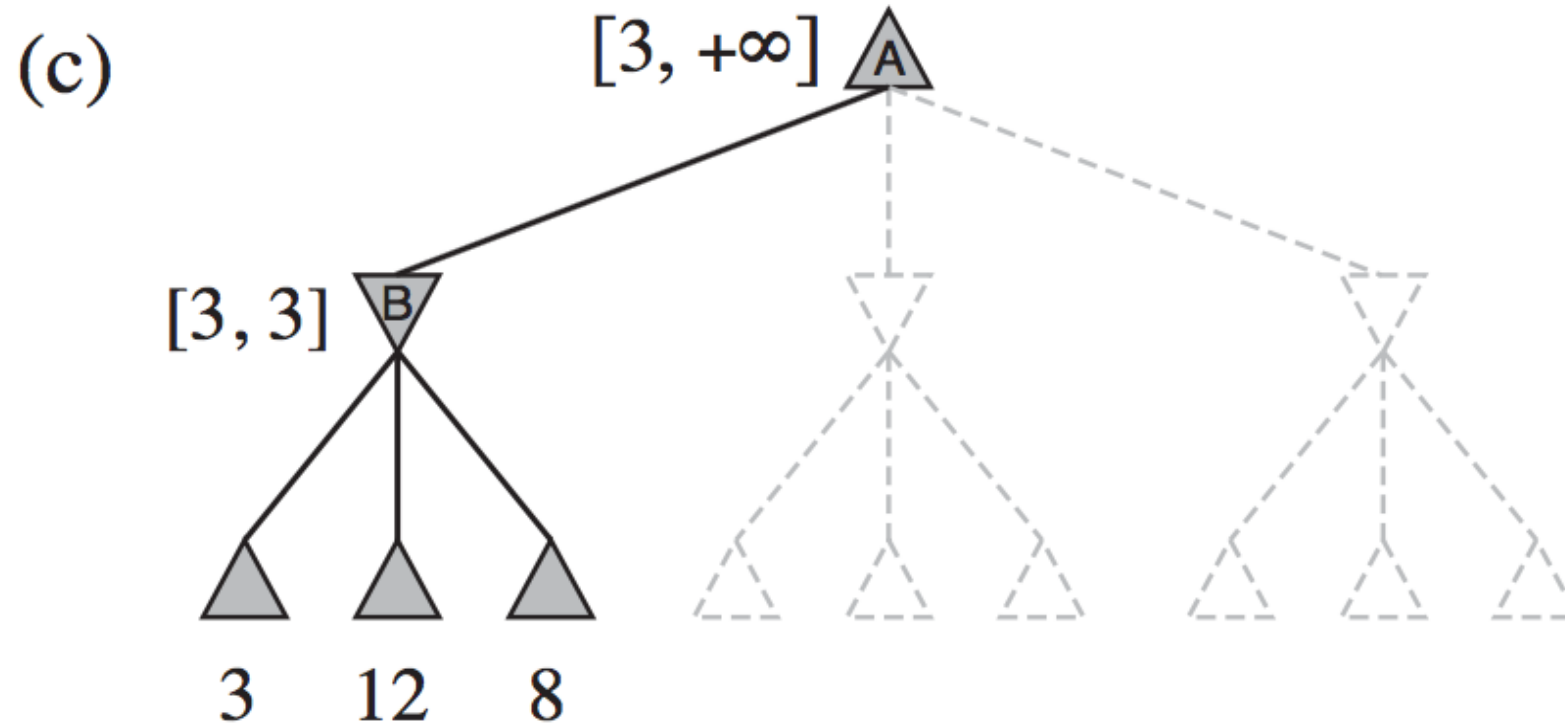
آلفا-بتا (مثال ۱)



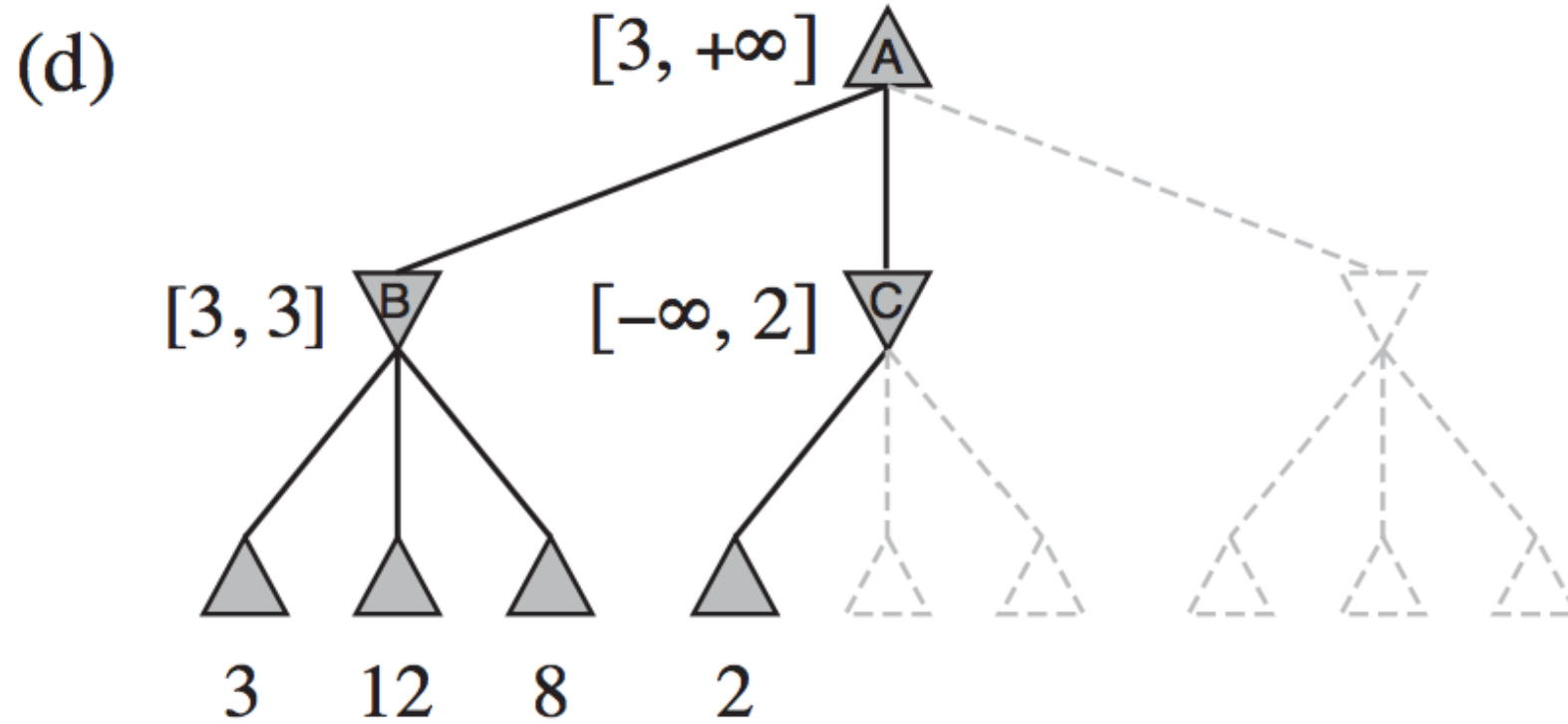
آلفا-بتا (مثال ۱)



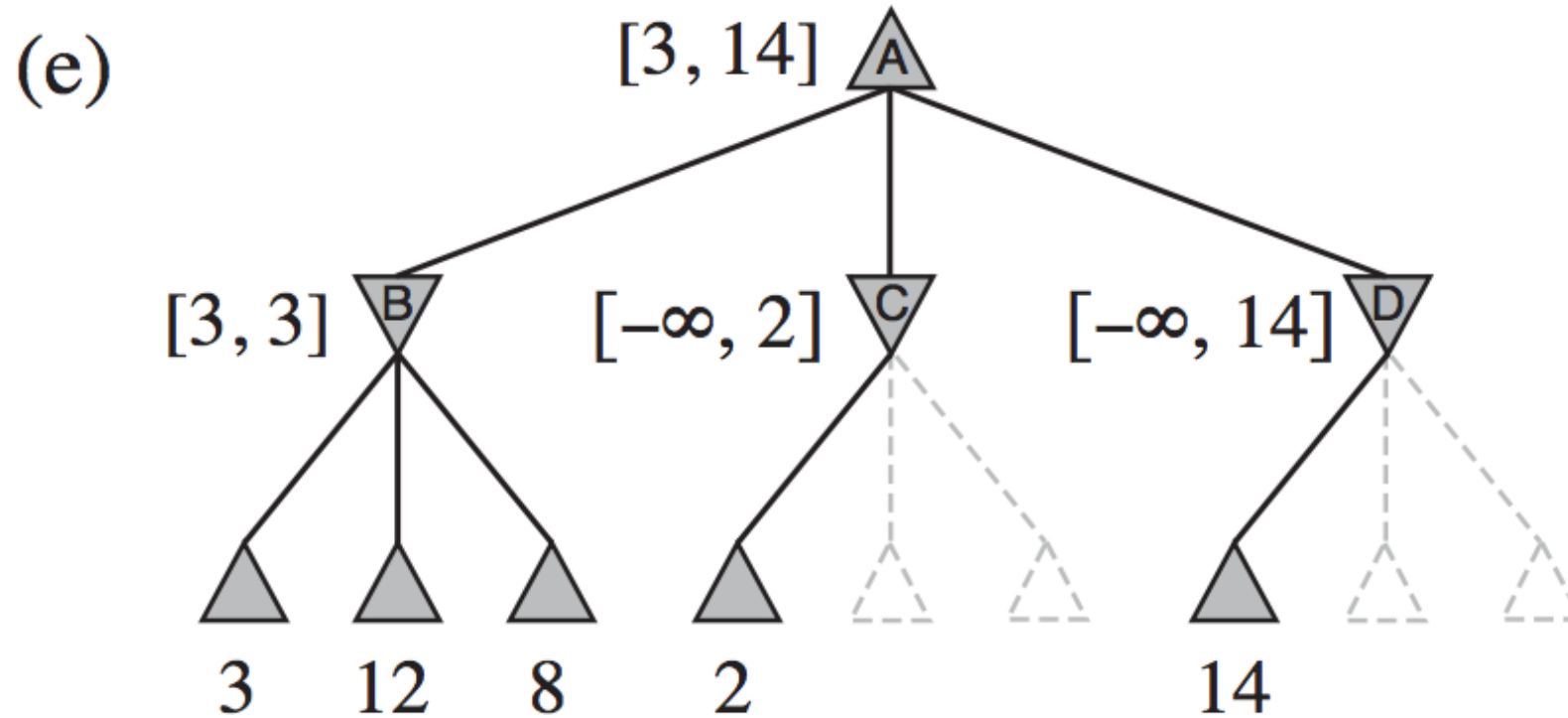
آلفا-بتا (مثال ۱)



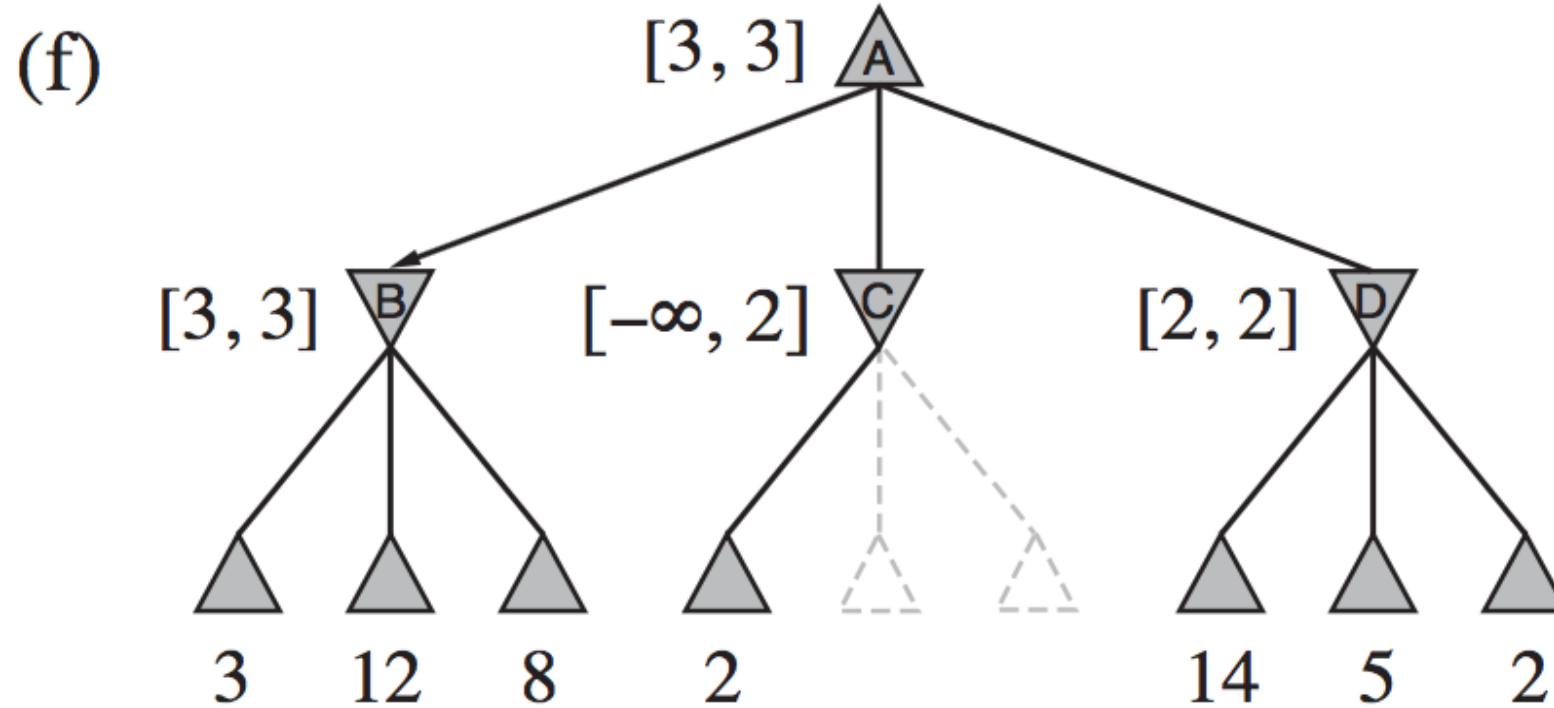
آلفا-بتا (مثال ۱)



آلفا-بتا (مثال ۱)



آلفا-بتا (مثال ۱)



function ALPHA-BETA-SEARCH($state$) **returns** an action
 $v \leftarrow \text{MAX-VALUE}(state, -\infty, +\infty)$
return the *action* in $\text{ACTIONS}(state)$ with value v

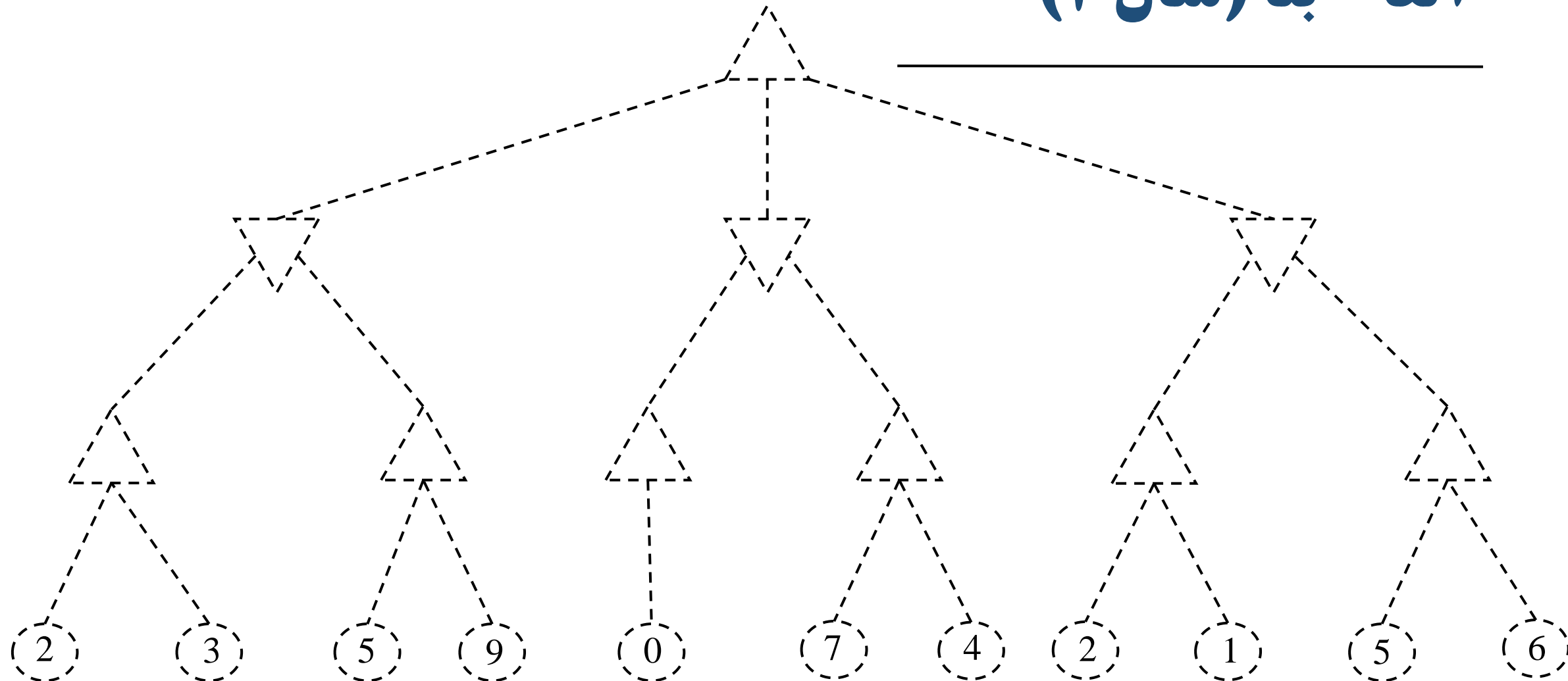
function MAX-VALUE($state, \alpha, \beta$) **returns** a *utility value*
if $\text{TERMINAL-TEST}(state)$ **then return** $\text{UTILITY}(state)$
 $v \leftarrow -\infty$
for each a **in** $\text{ACTIONS}(state)$ **do**
 $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$
 if $v \geq \beta$ **then return** v
 $\alpha \leftarrow \text{MAX}(\alpha, v)$
return v

function MIN-VALUE($state, \alpha, \beta$) **returns** a *utility value*
if $\text{TERMINAL-TEST}(state)$ **then return** $\text{UTILITY}(state)$
 $v \leftarrow +\infty$
for each a **in** $\text{ACTIONS}(state)$ **do**
 $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$
 if $v \leq \alpha$ **then return** v
 $\beta \leftarrow \text{MIN}(\beta, v)$
return v

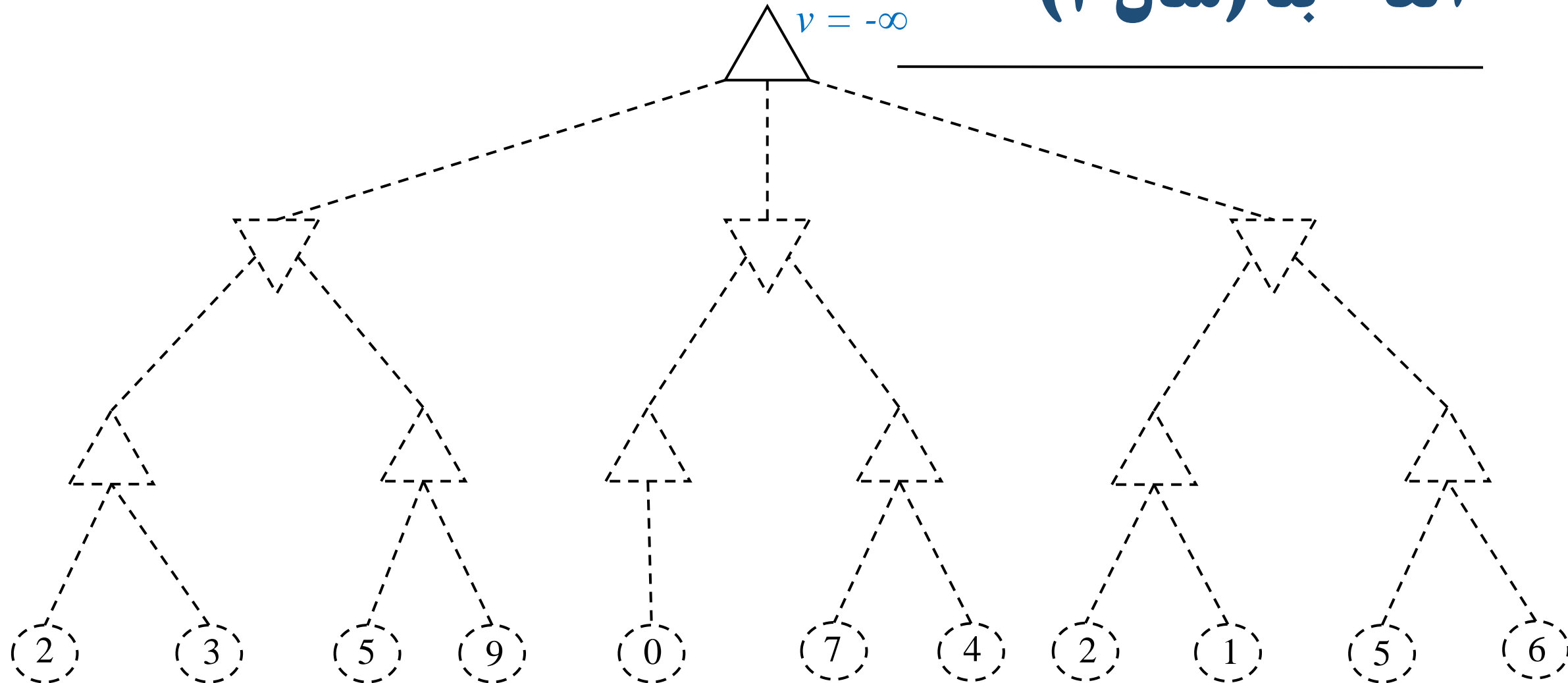
استفاده از الگوریتم آلفا-بتا

- **آلفا:** سودمندی بهترین (بیشترین) انتخابی که تا کنون در گره‌های مسیر مربوط به MAX پیدا شده است.
- **بتا:** سودمندی بهترین (کم‌ترین) انتخابی که تا کنون در گره‌های مسیر مربوط به MIN پیدا شده است
- به‌روزرسانی آلفا و بتا در طول فرایند جستجو و هرس کردن براساس آن‌ها
- **هرس بتا:** برای یک نود MAX زمانی که مقدار یافت‌شده برای آن v بیشتر از مقدار بتای فعلی بود ($v \geq \beta$)، زیرشاخه‌هایش هرس می‌شوند.
- **هرس آلفا:** برای یک نود MIN زمانی که مقدار یافت‌شده برای آن v کمتر از مقدار آلفای فعلی بود ($v \leq \alpha$)، زیرشاخه‌هایش هرس می‌شوند.

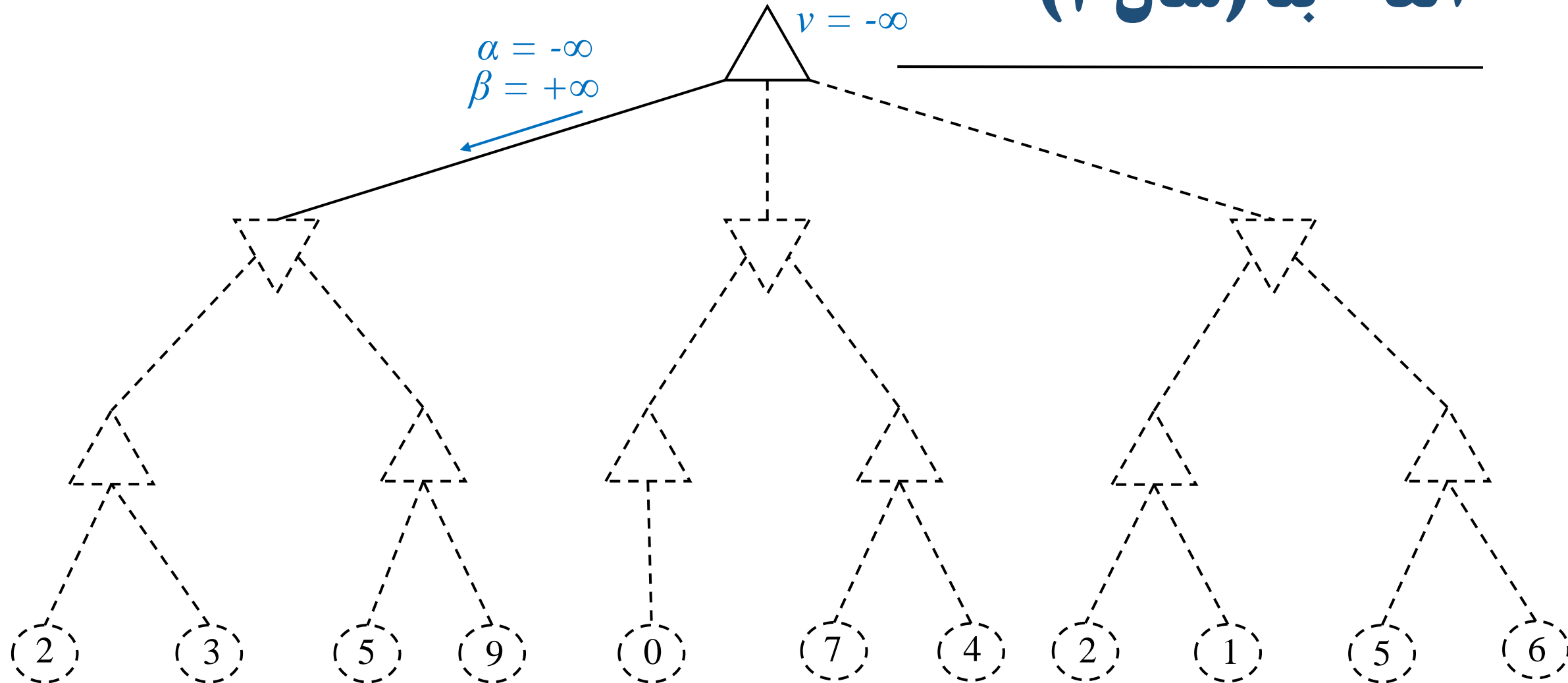
آلفا-بتا (مثال ۲)



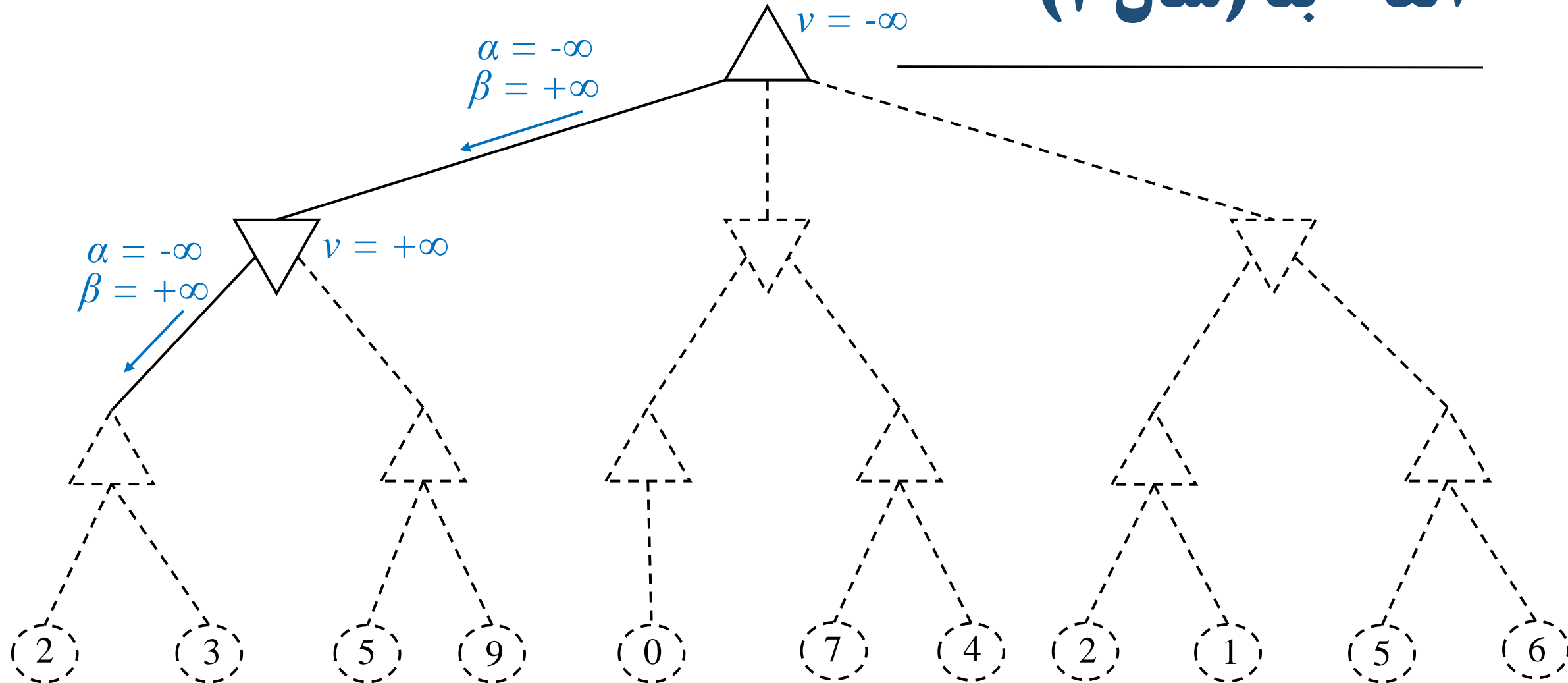
آلفا-بتا (مثال ۲)



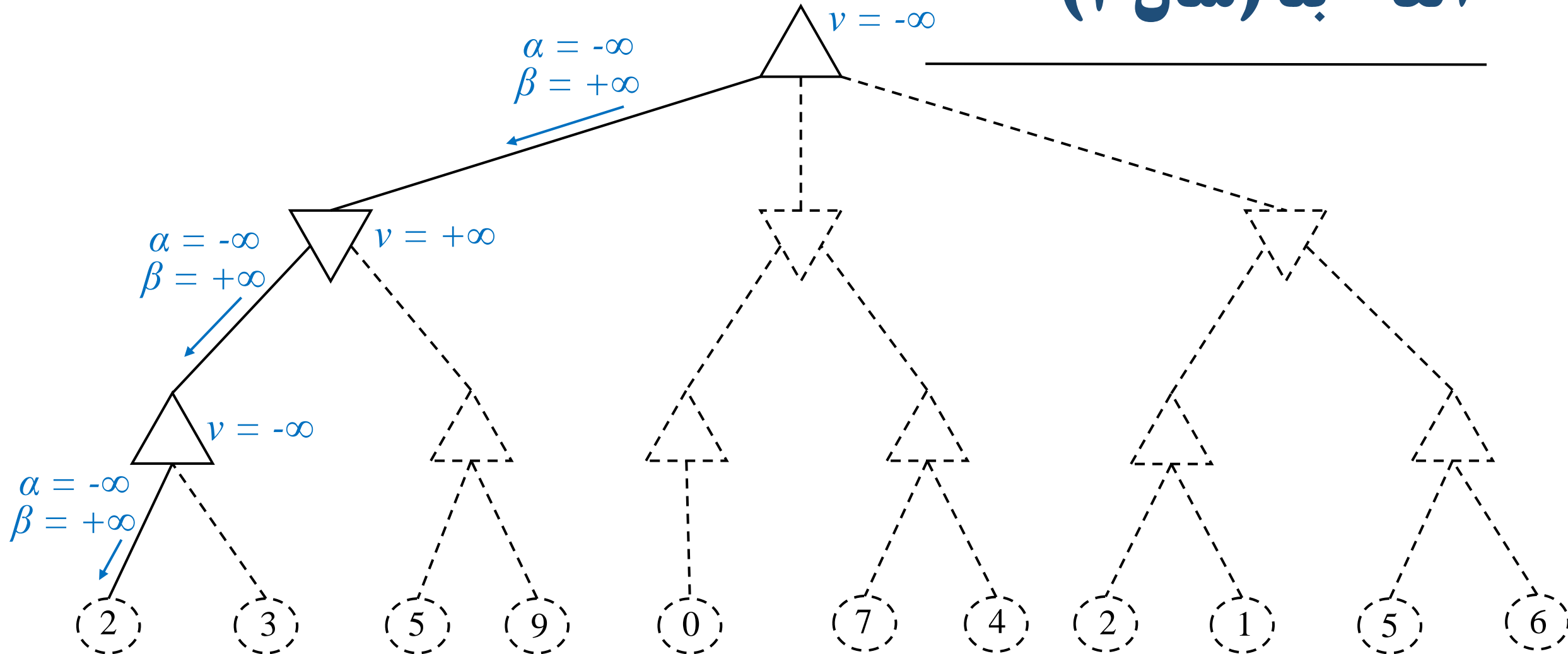
آلفا-بتا (مثال ۲)



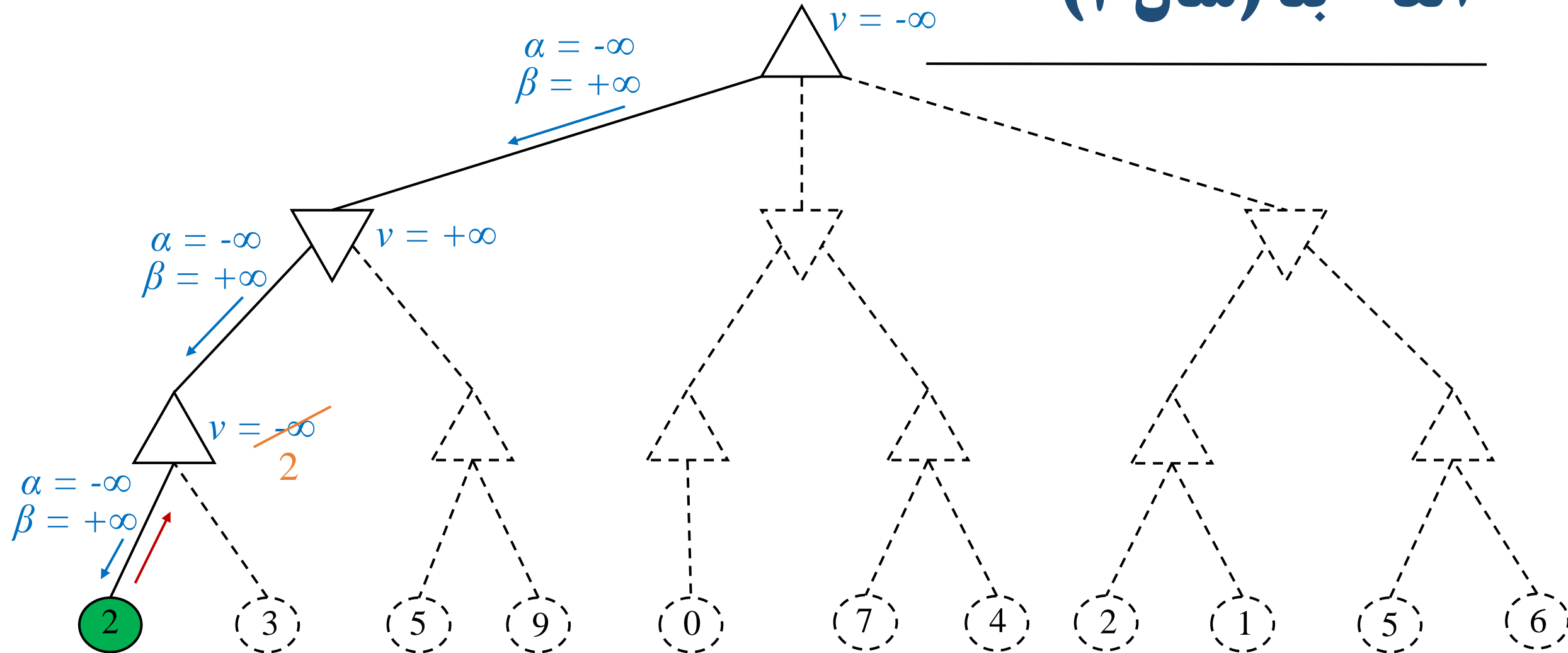
آلفا-بتا (مثال ۲)



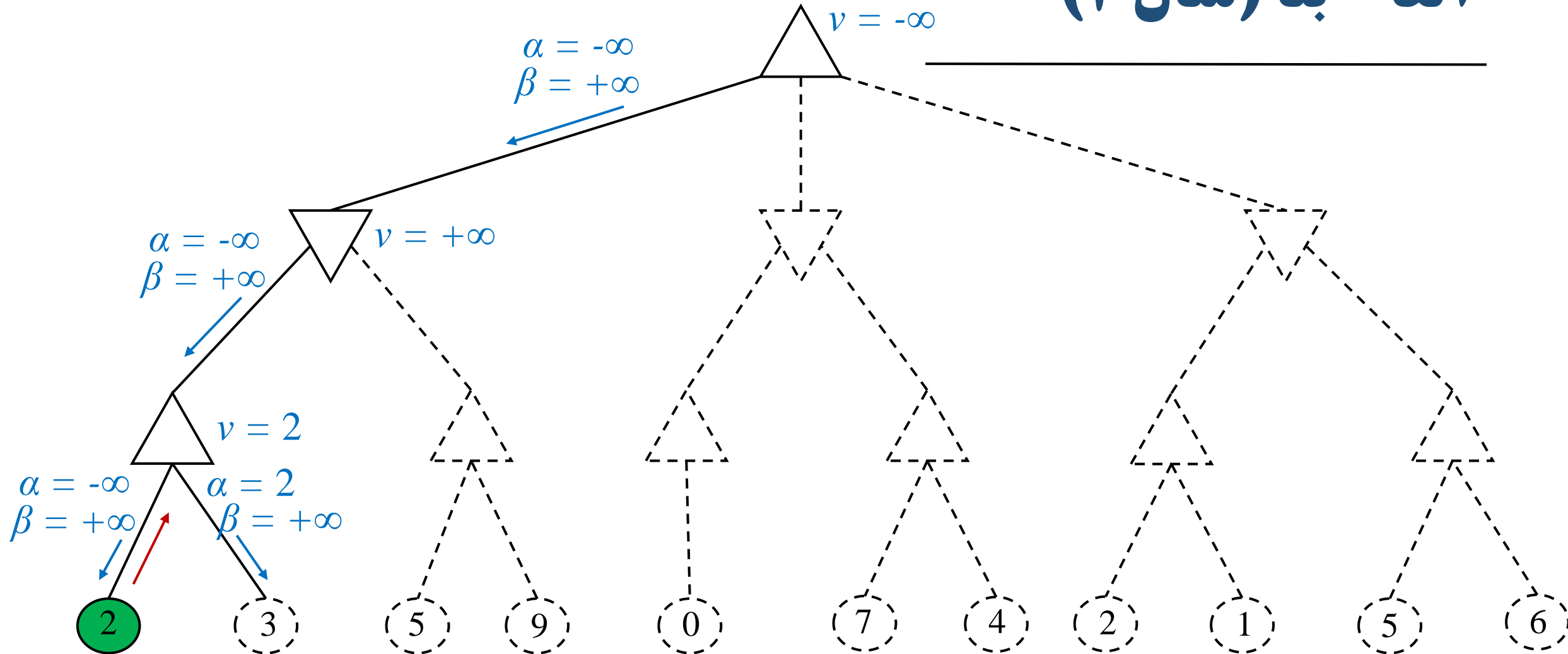
آلفا-بتا (مثال ۲)



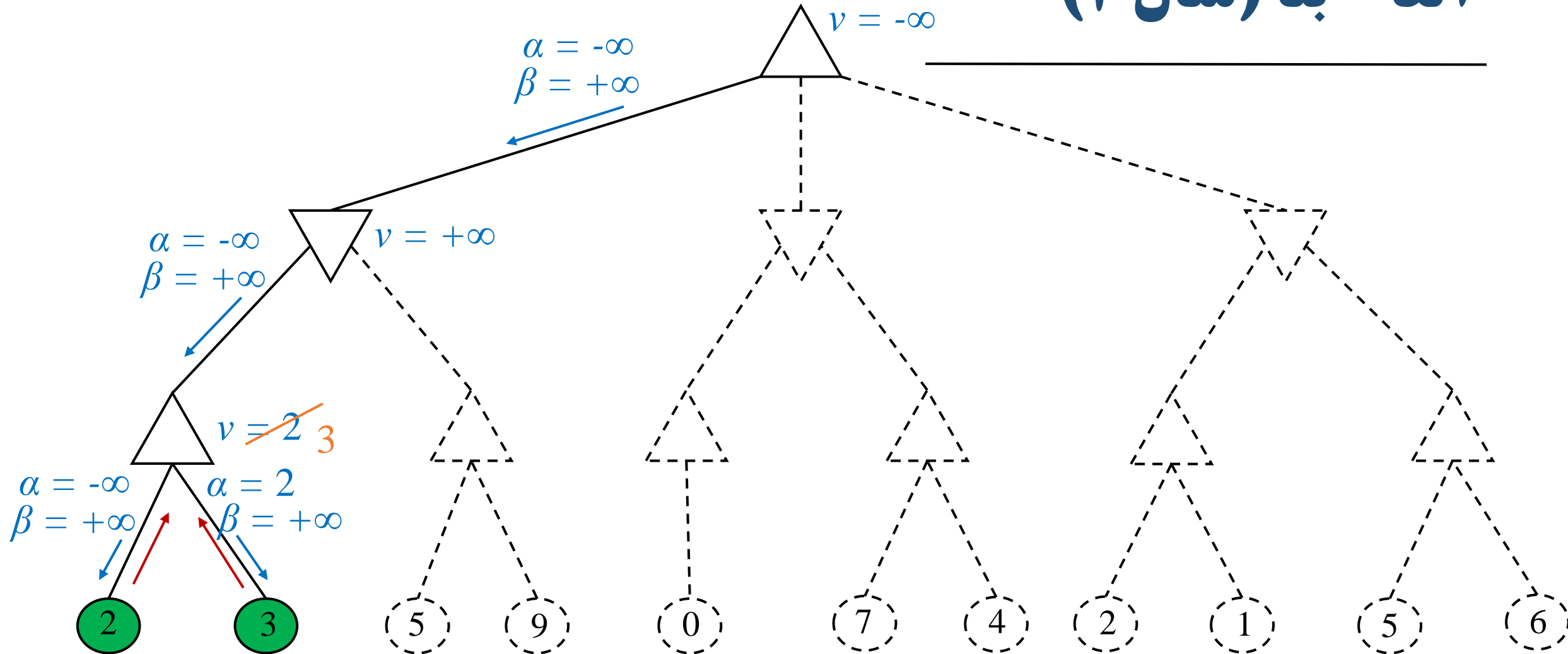
آلڦا-بتا (مثال ۲)



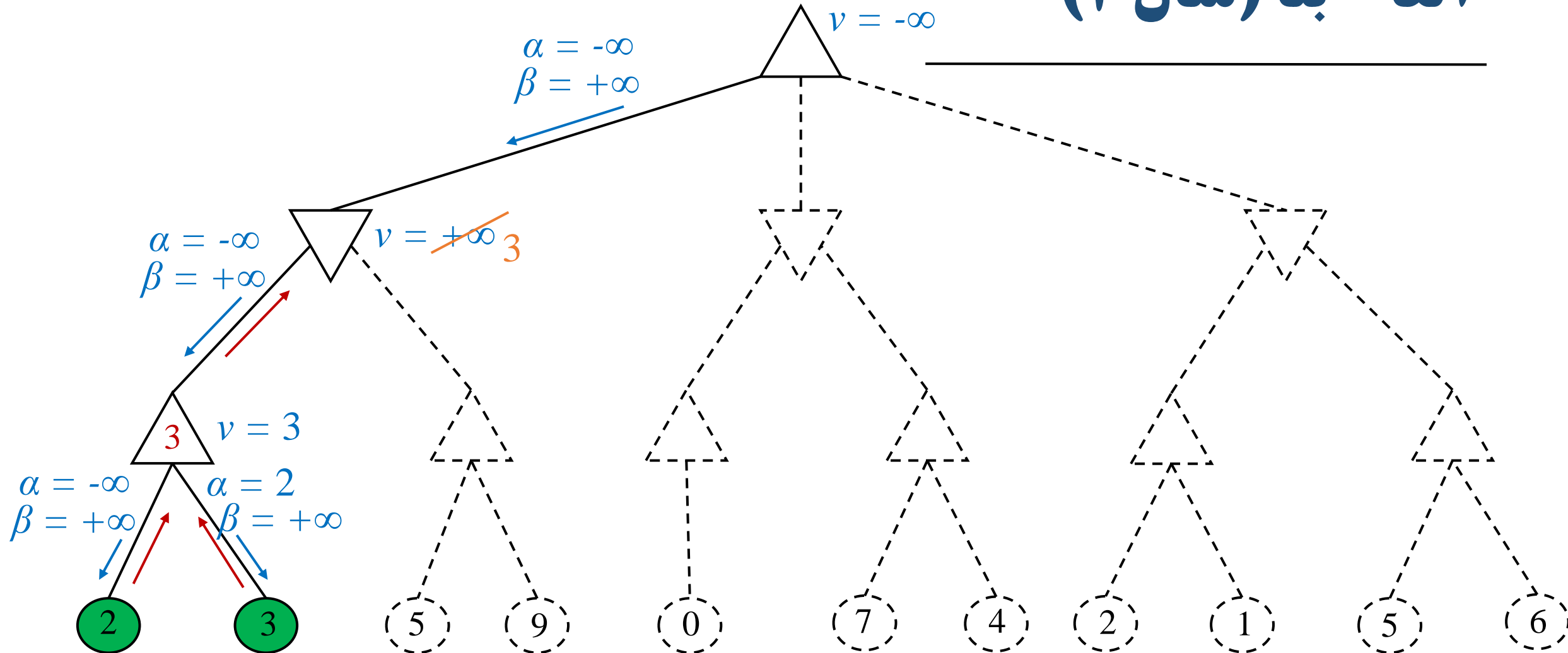
آلفا-بتا (مثال ۲)



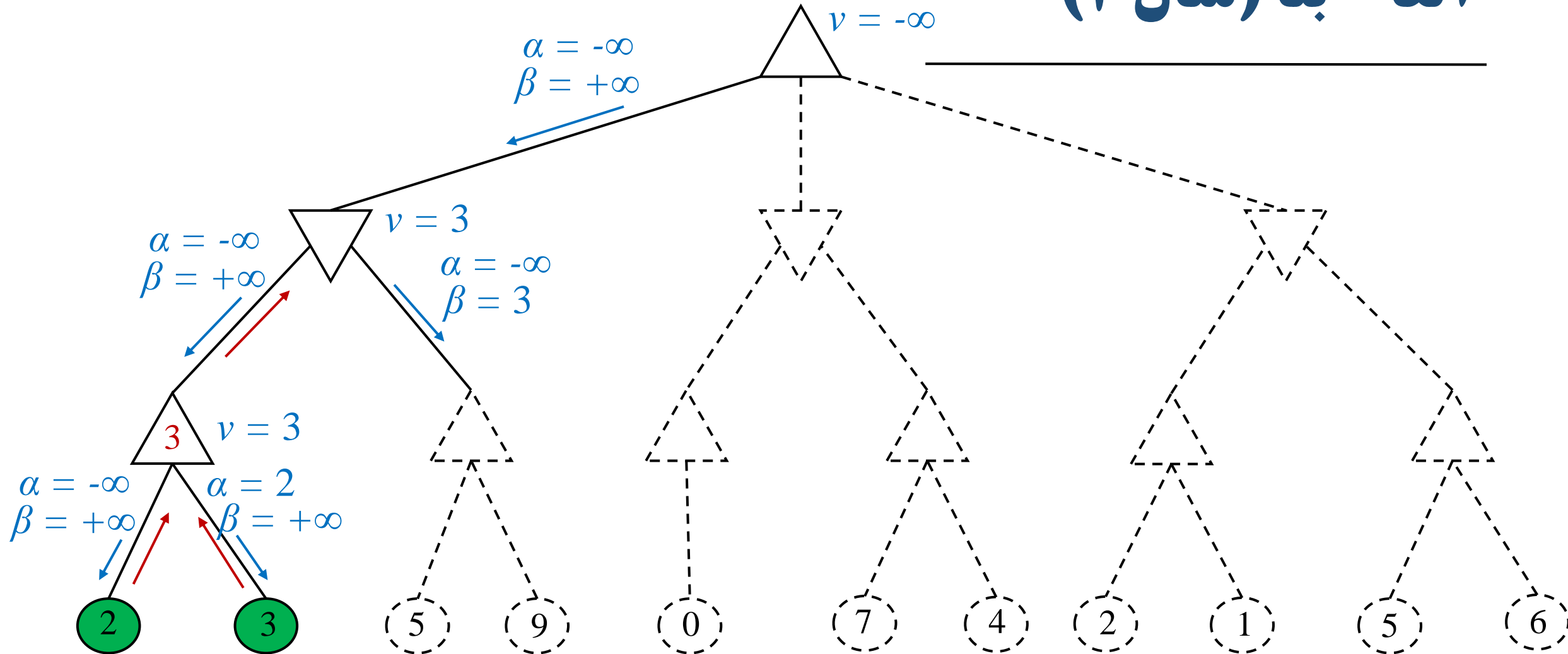
آلفا-بتا (مثال ۲)



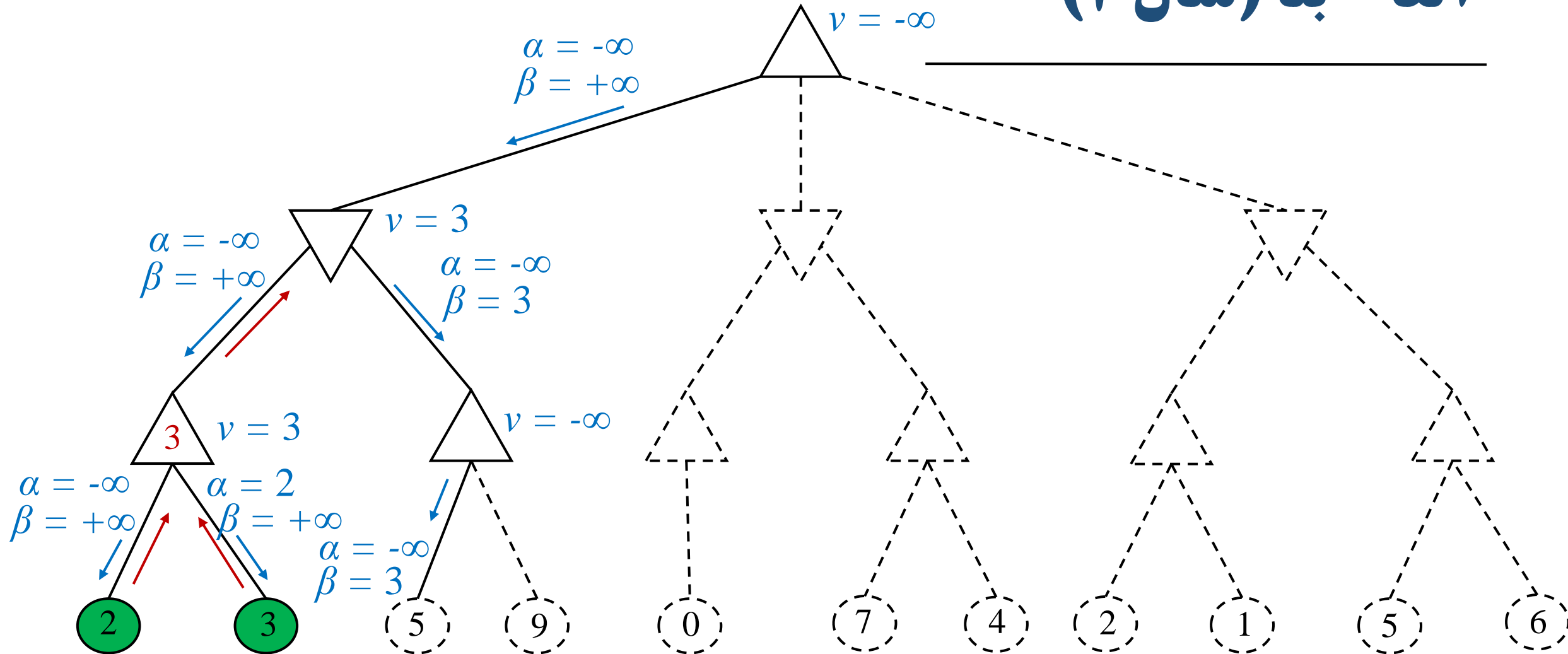
آلفا-بتا (مثال ۲)



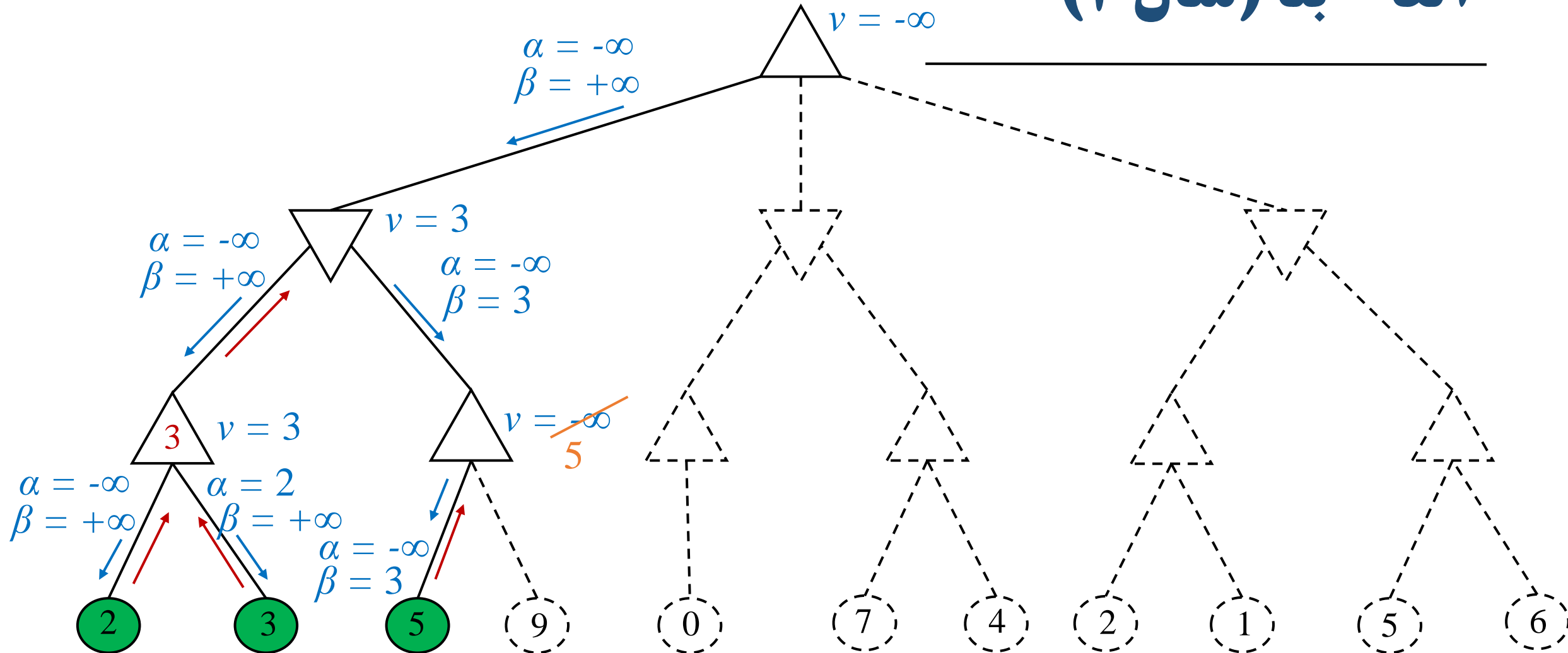
آلفا-بتا (مثال ۲)



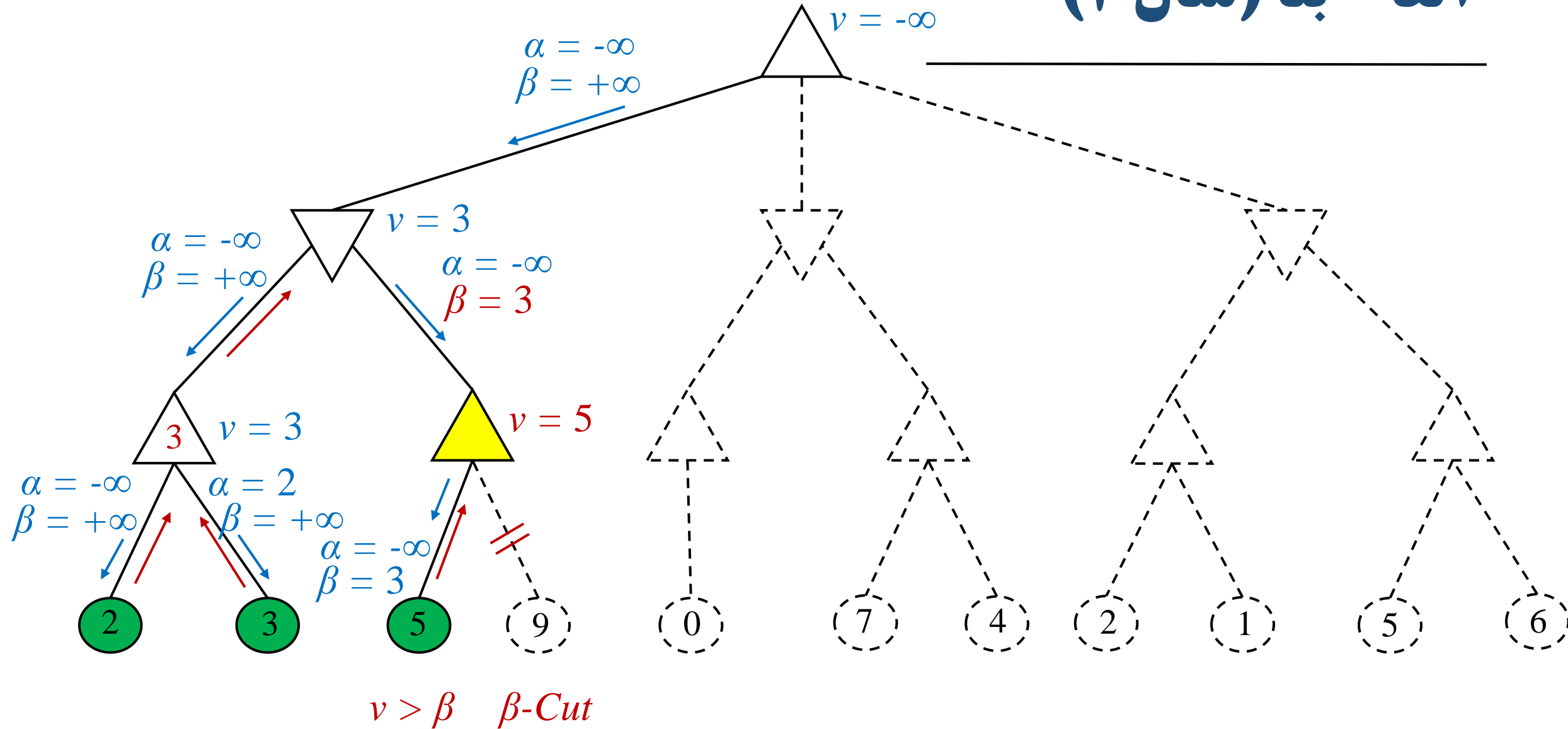
آلفا-بتا (مثال ۲)



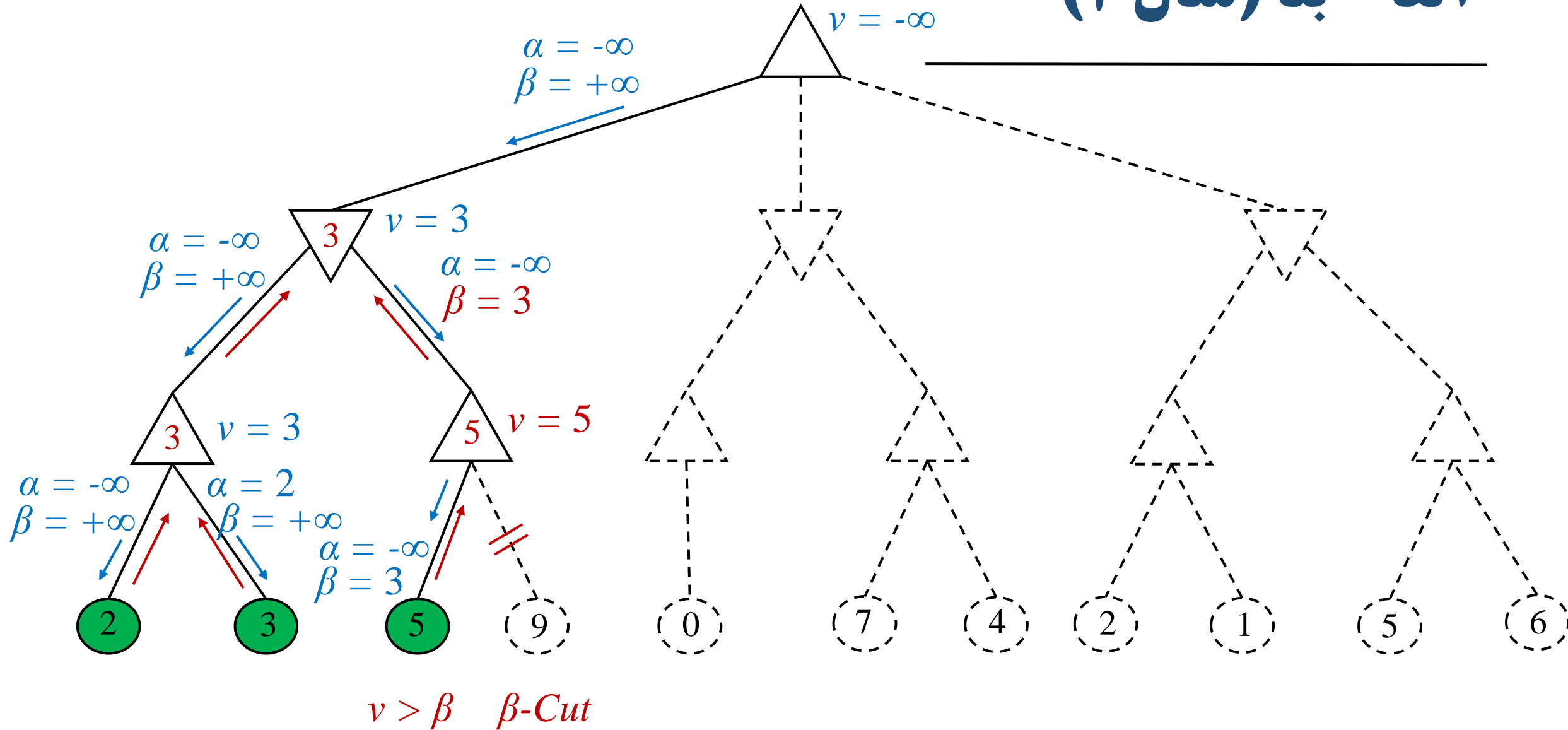
آلفا-بتا (مثال ۲)



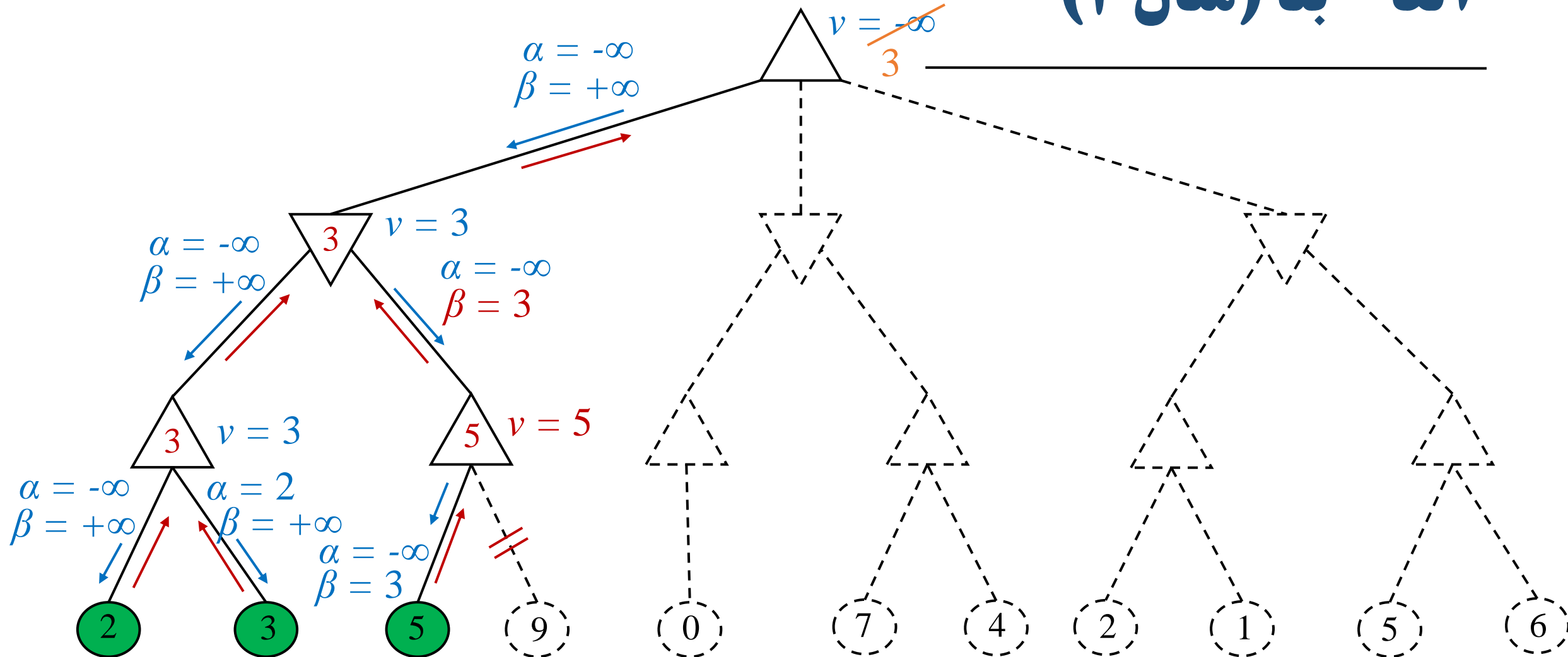
آلڦا-بتا (مثال ۲)



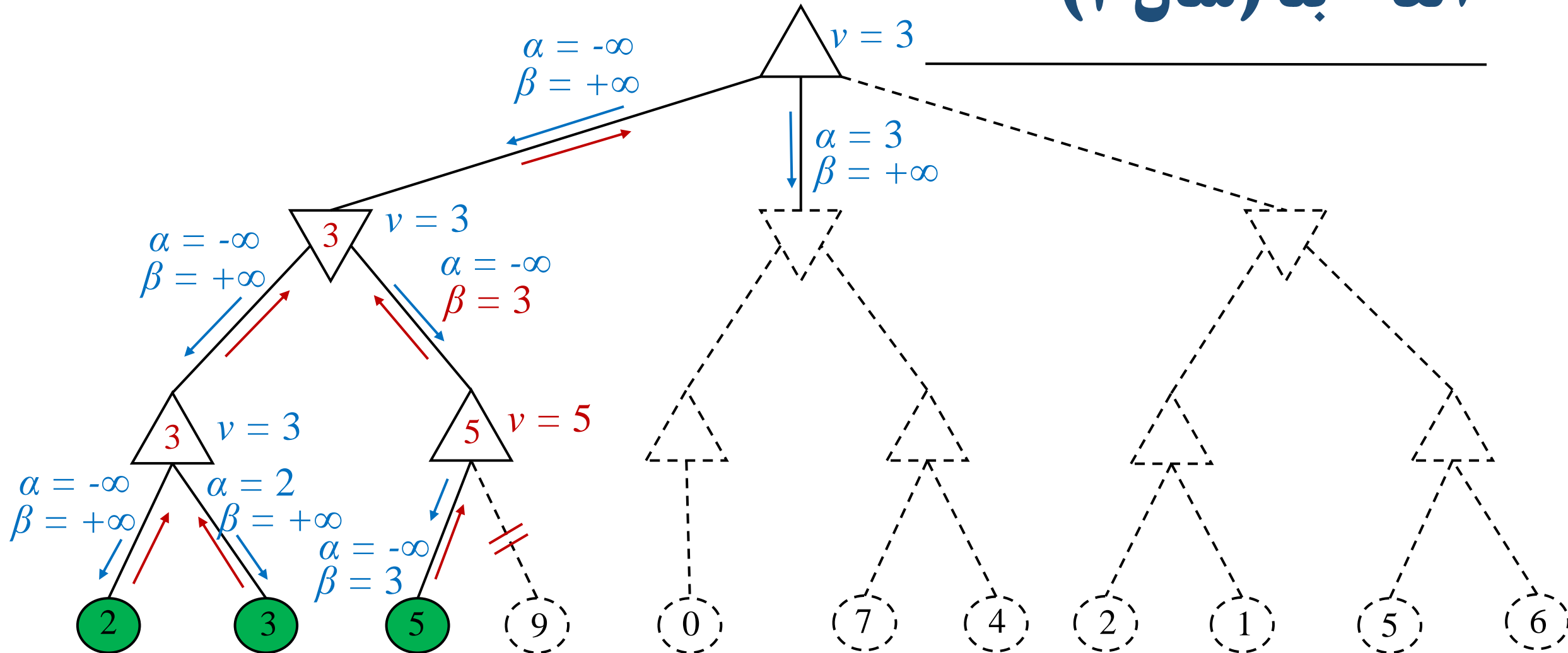
آلفا-بتا (مثال ۲)



آلفا-بتا (مثال ۲)

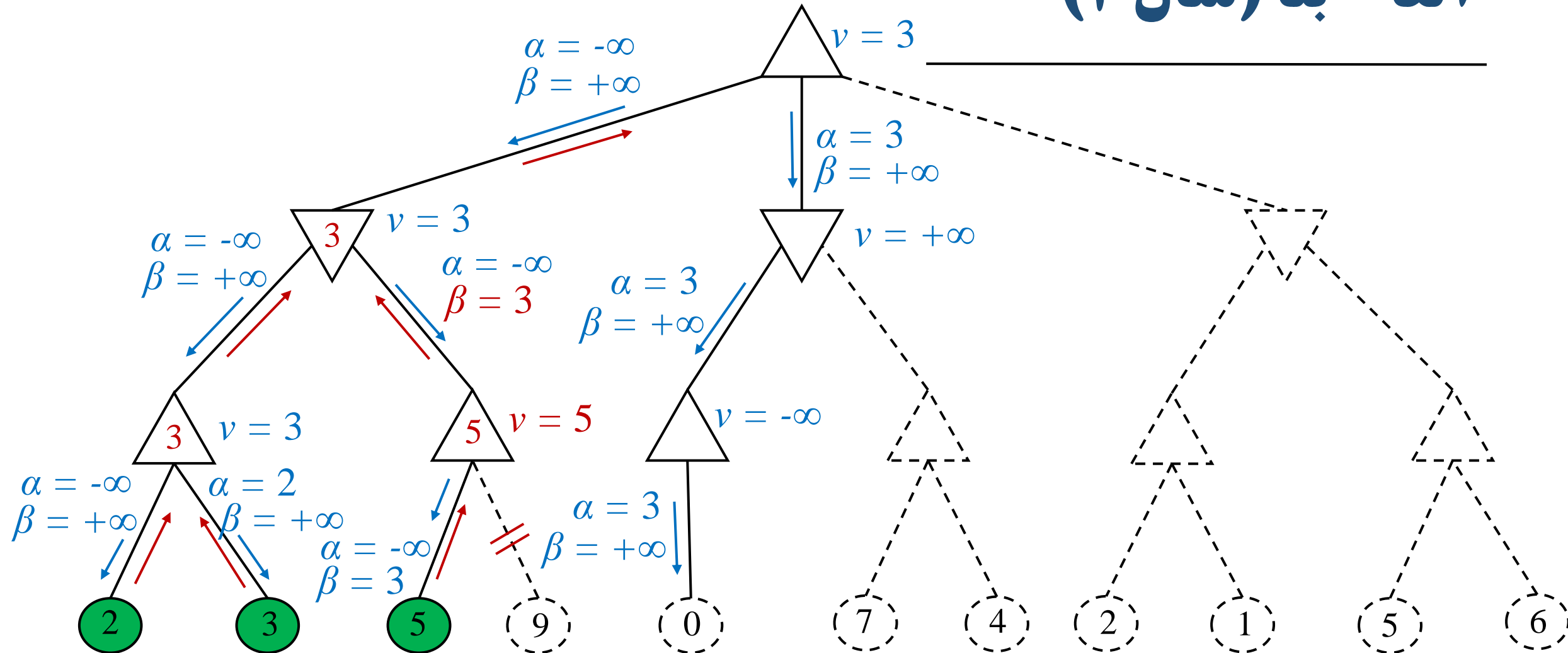


آلفا-بتا (مثال ۲)

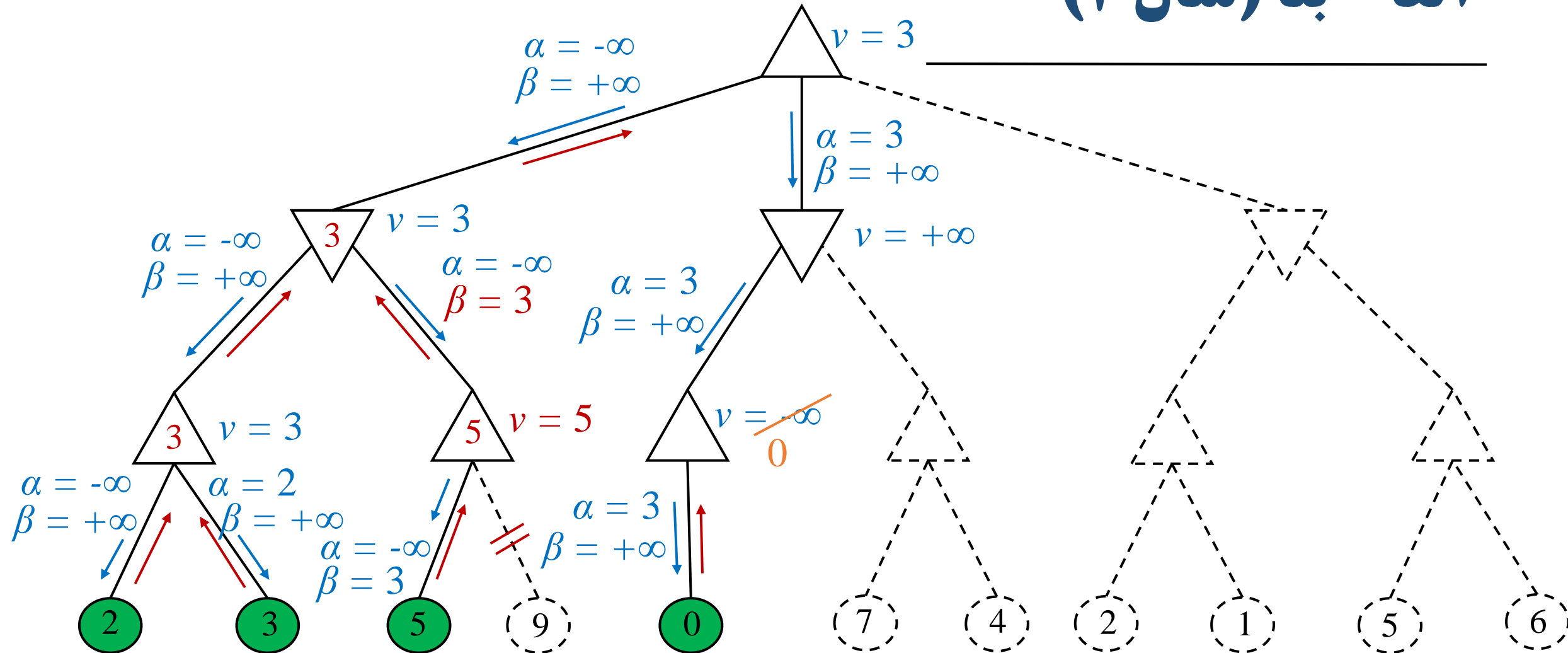


$v > \beta$ β -Cut

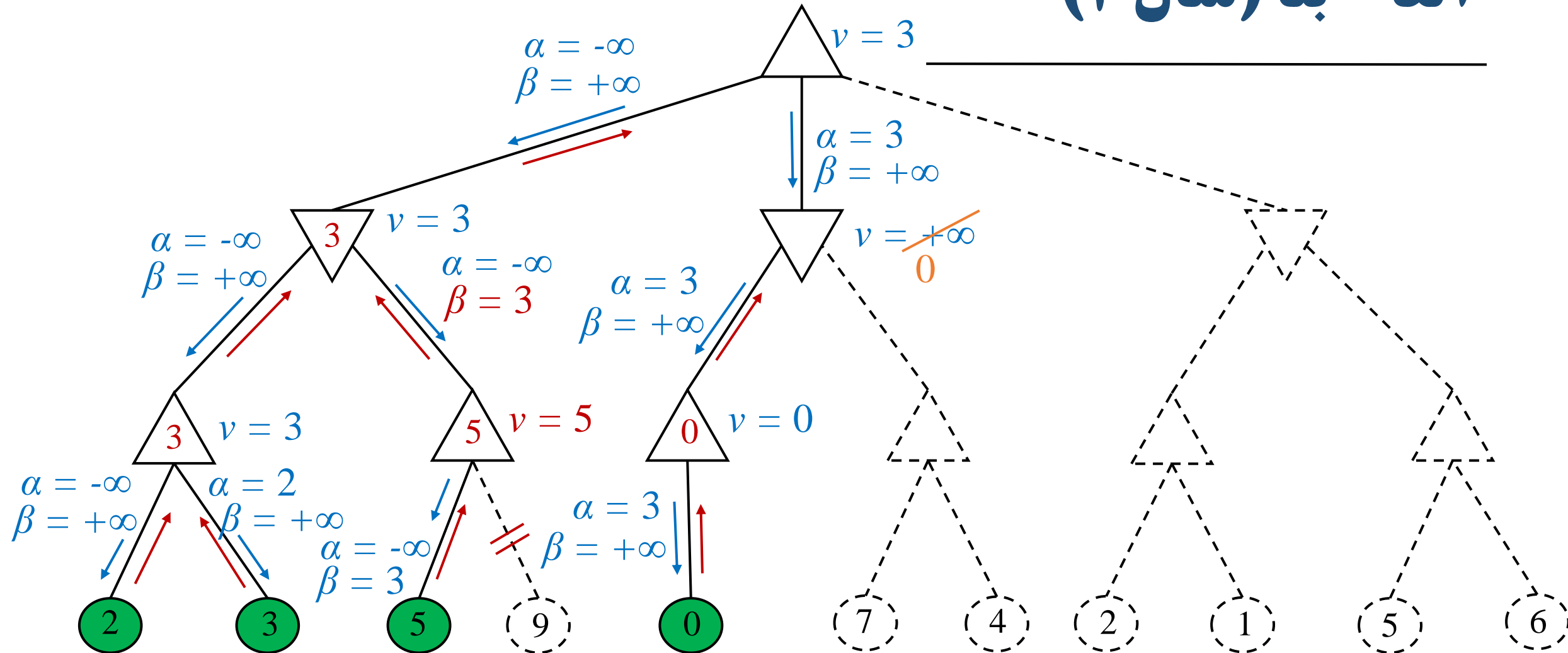
آلڦا-بتا (مثال ۲)


$$v > \beta \quad \beta\text{-Cut}$$

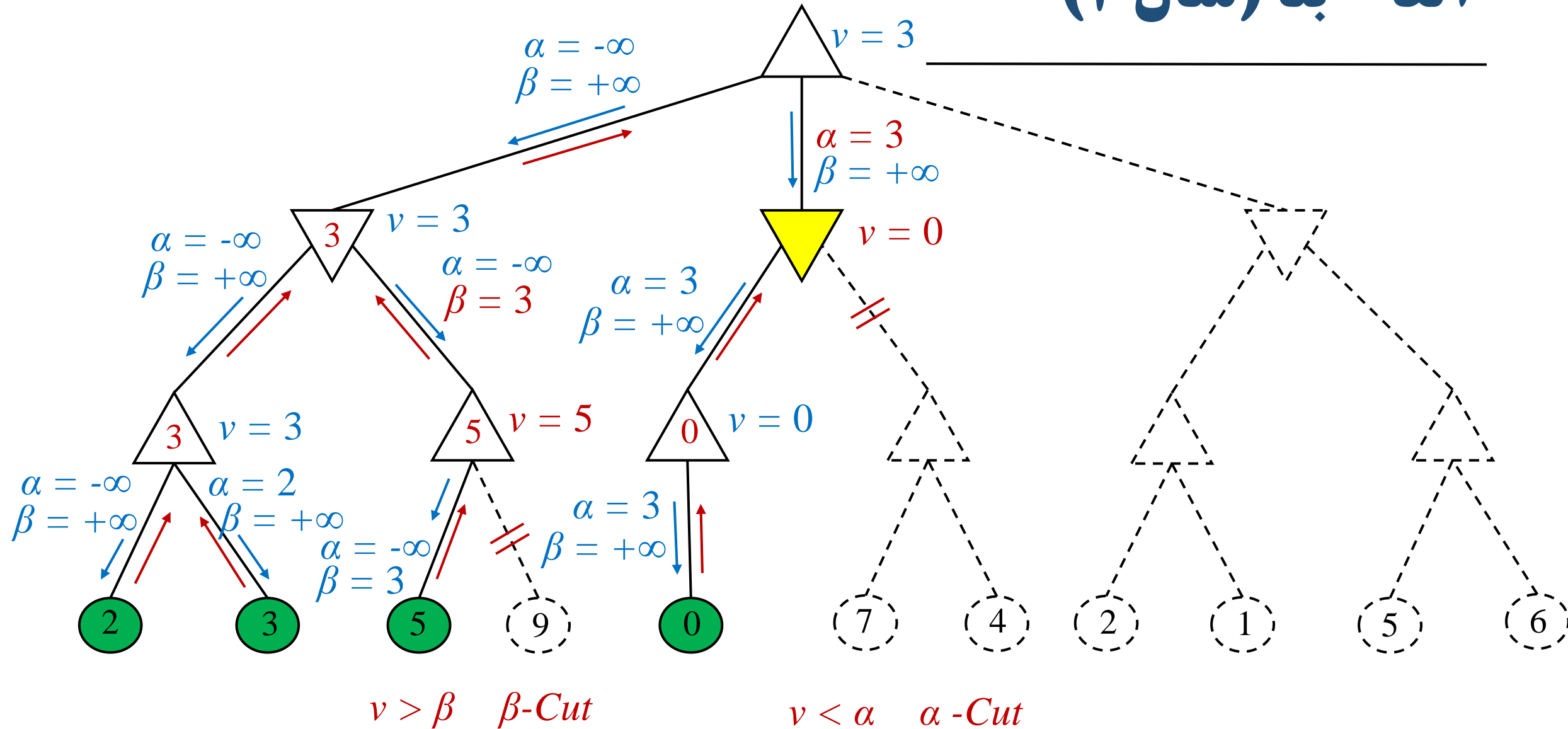
آلڦا-بتا (مثال ۲)


$$v > \beta \quad \beta\text{-Cut}$$

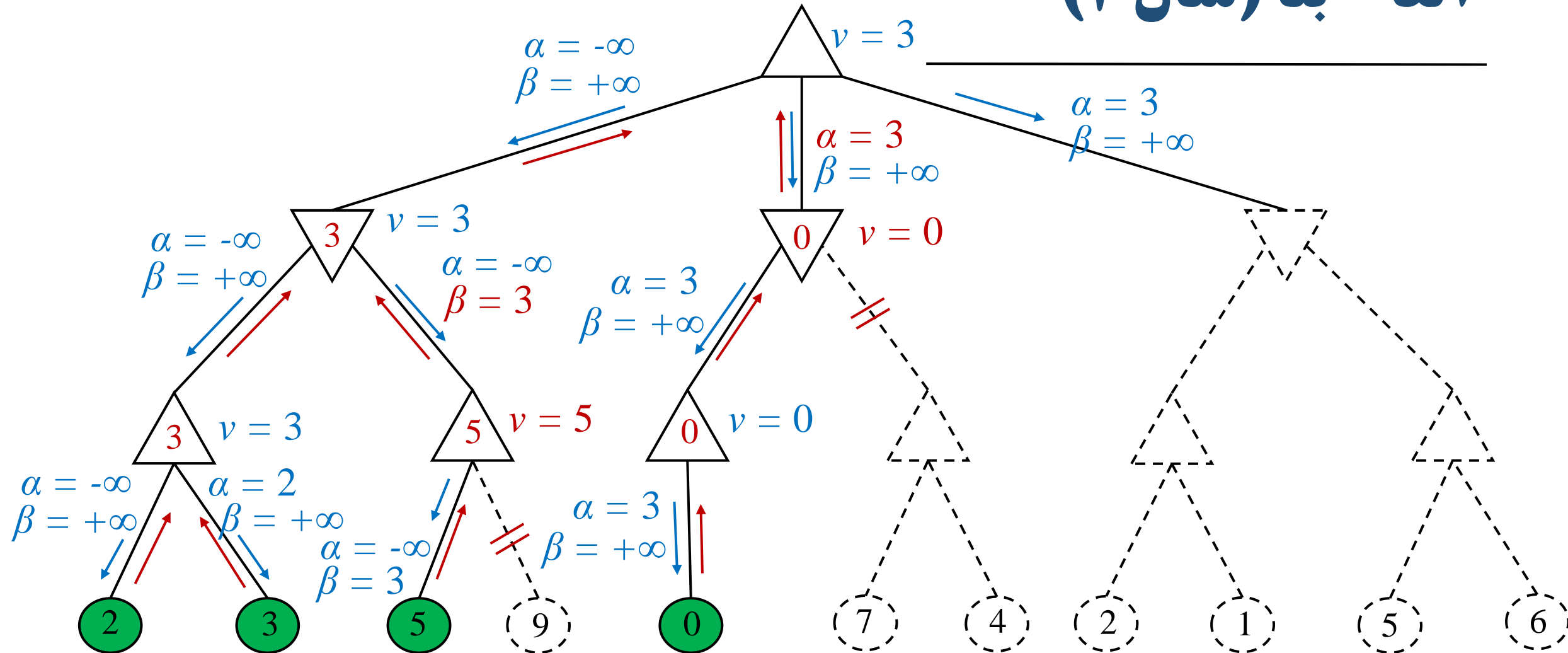
آلڦا-بتا (مثال ۲)


$$v > \beta \quad \beta\text{-Cut}$$

آلڦا-بتا (مثال ۲)



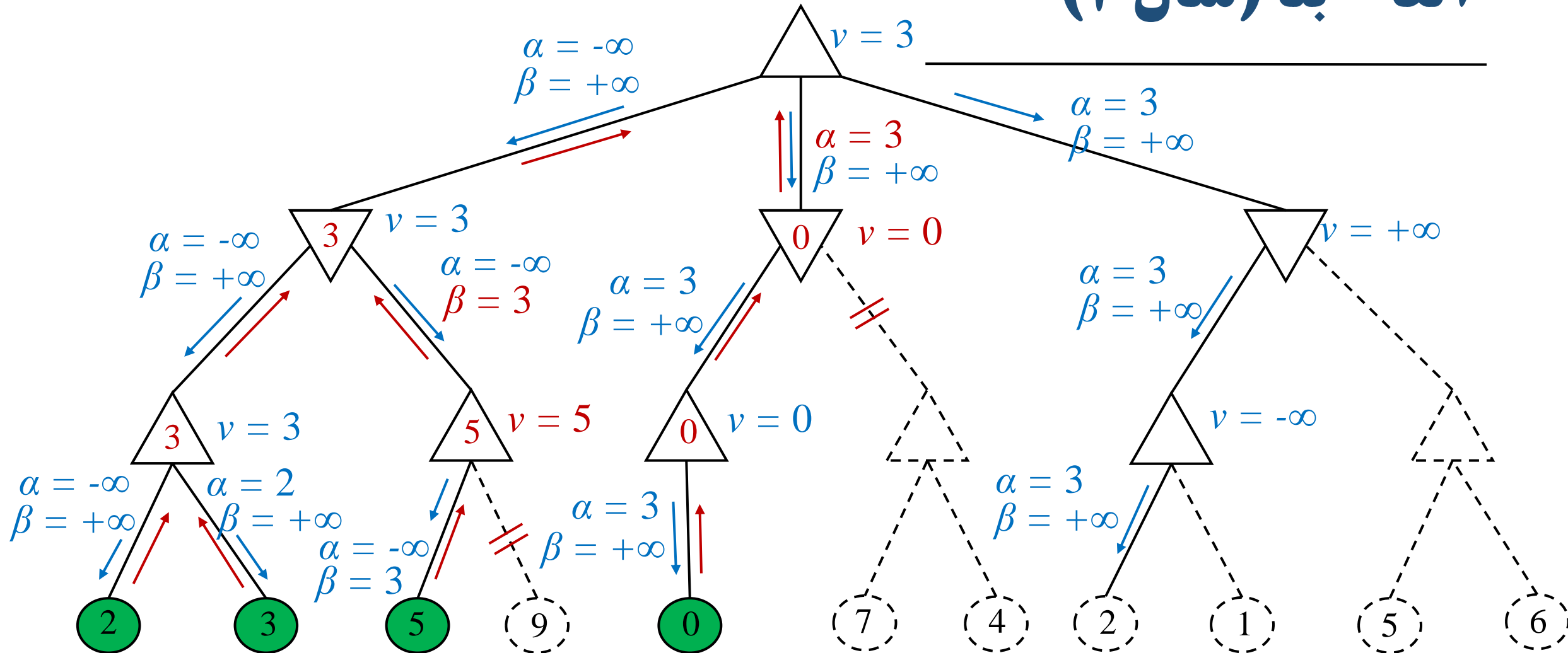
آلفا-بتا (مثال ۲)



$v > \beta$ β -Cut

$v < \alpha$ α -Cut

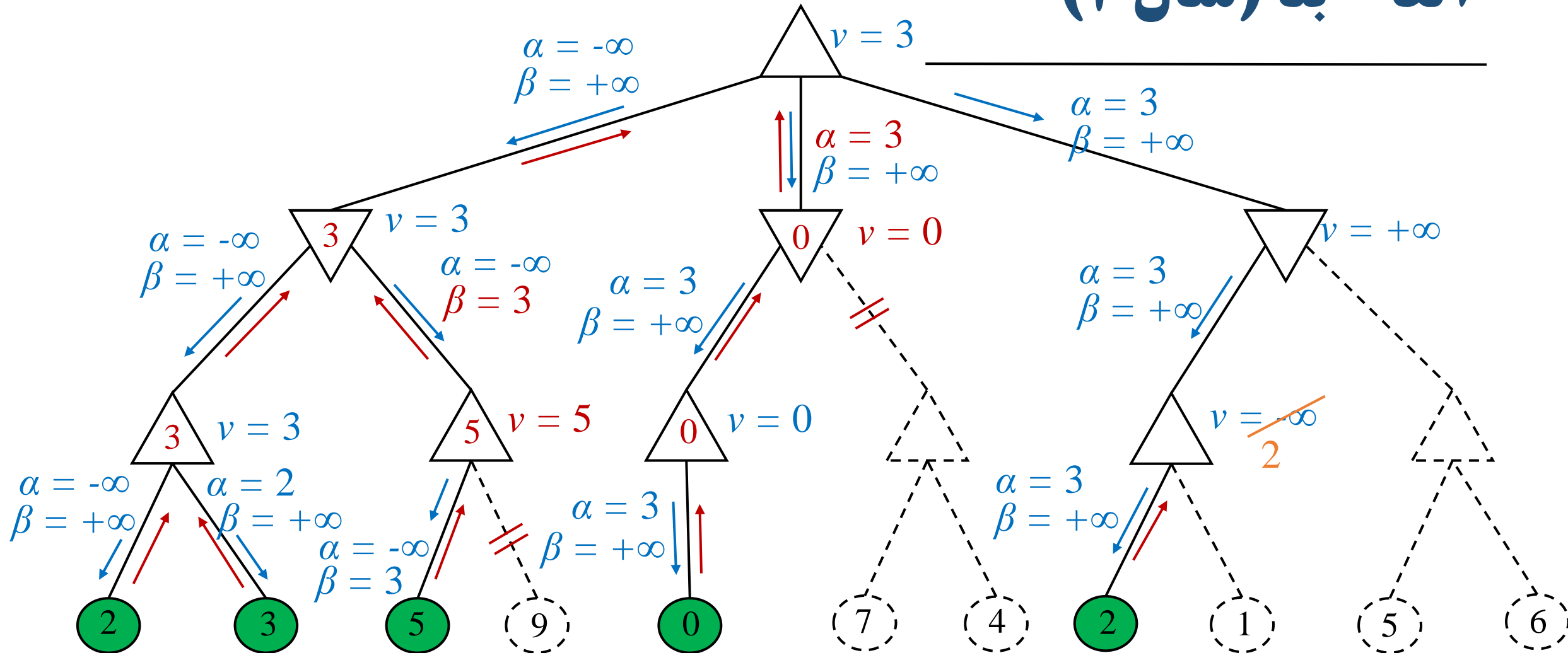
آلفا-بتا (مثال ۲)



$v > \beta$ β -Cut

$v < \alpha$ α -Cut

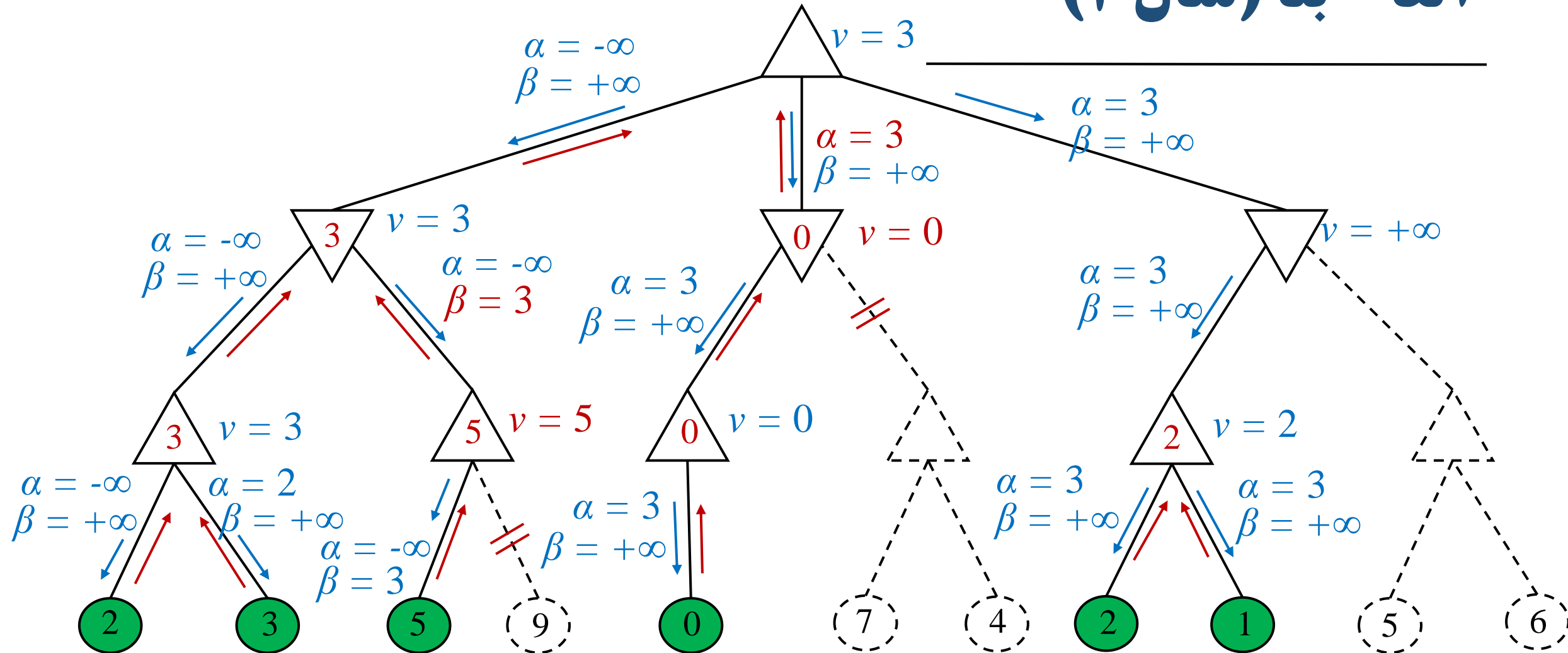
آلفا-بتا (مثال ۲)



$v > \beta$ β -Cut

$v < \alpha$ α -Cut

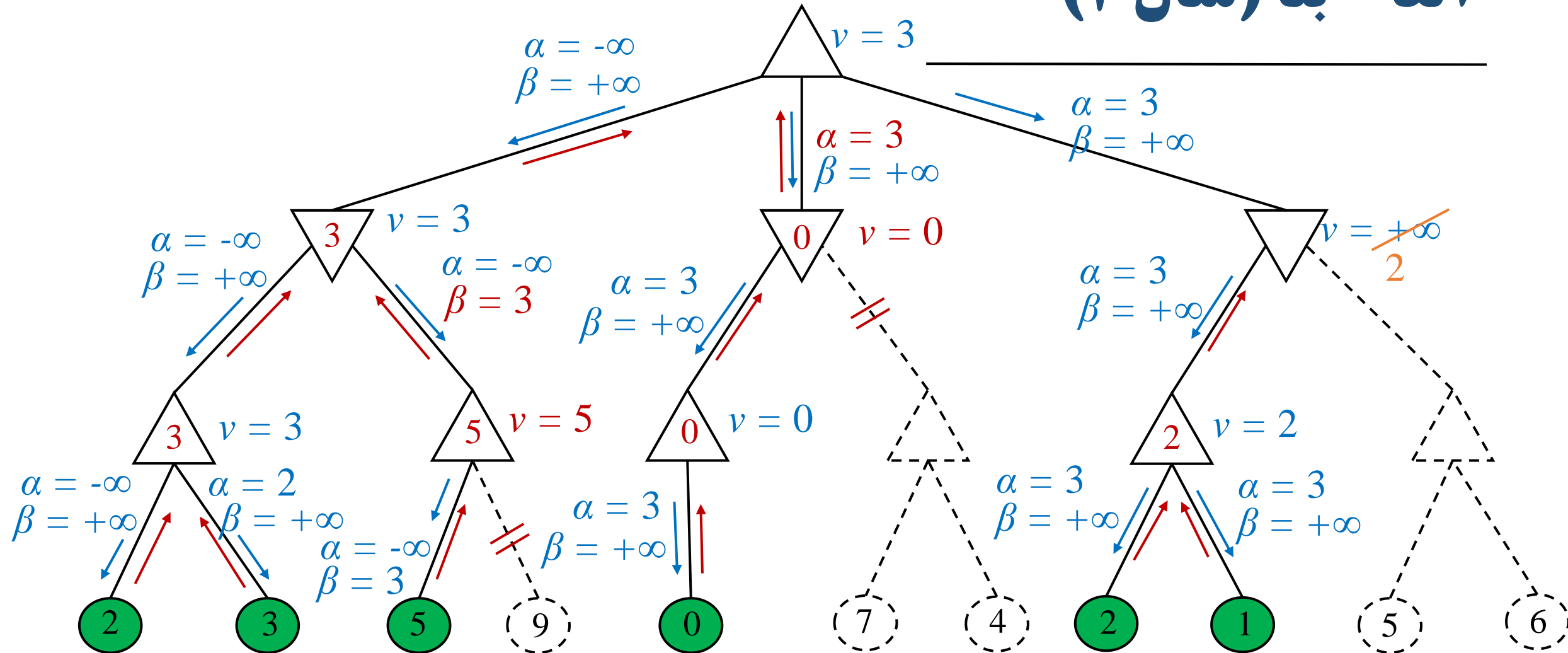
آلفا-بتا (مثال ۲)



$v > \beta$ β -Cut

$v < \alpha$ α -Cut

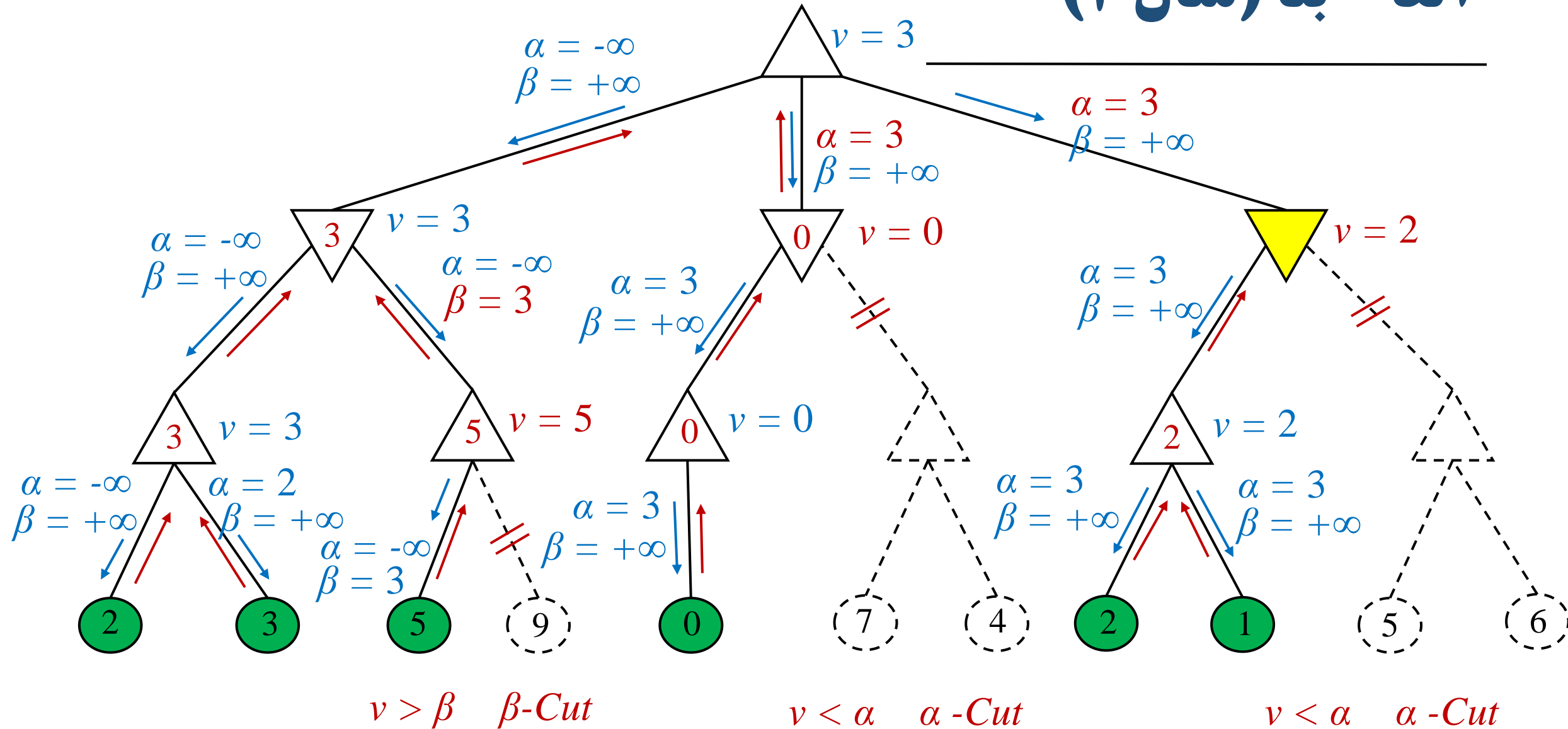
آلفا-بتا (مثال ۲)



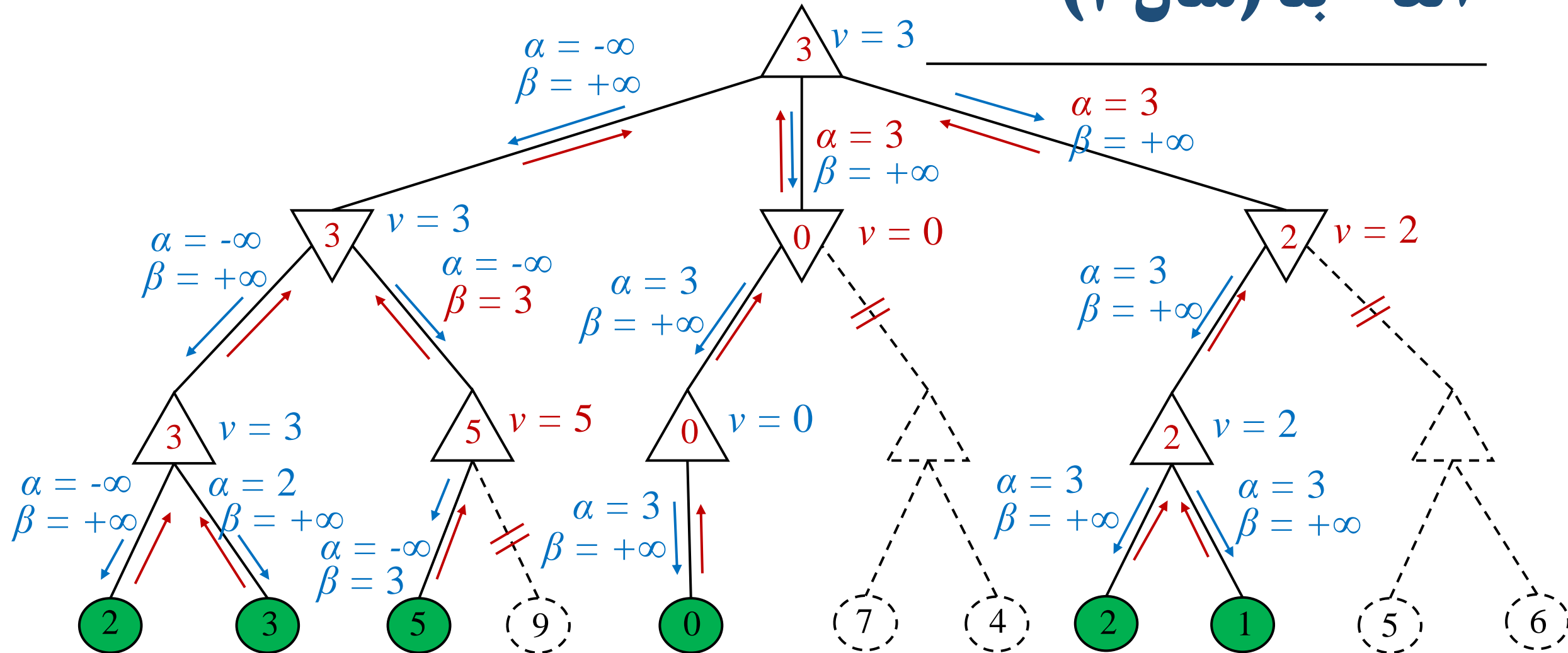
$v > \beta$ β -Cut

$v < \alpha$ α -Cut

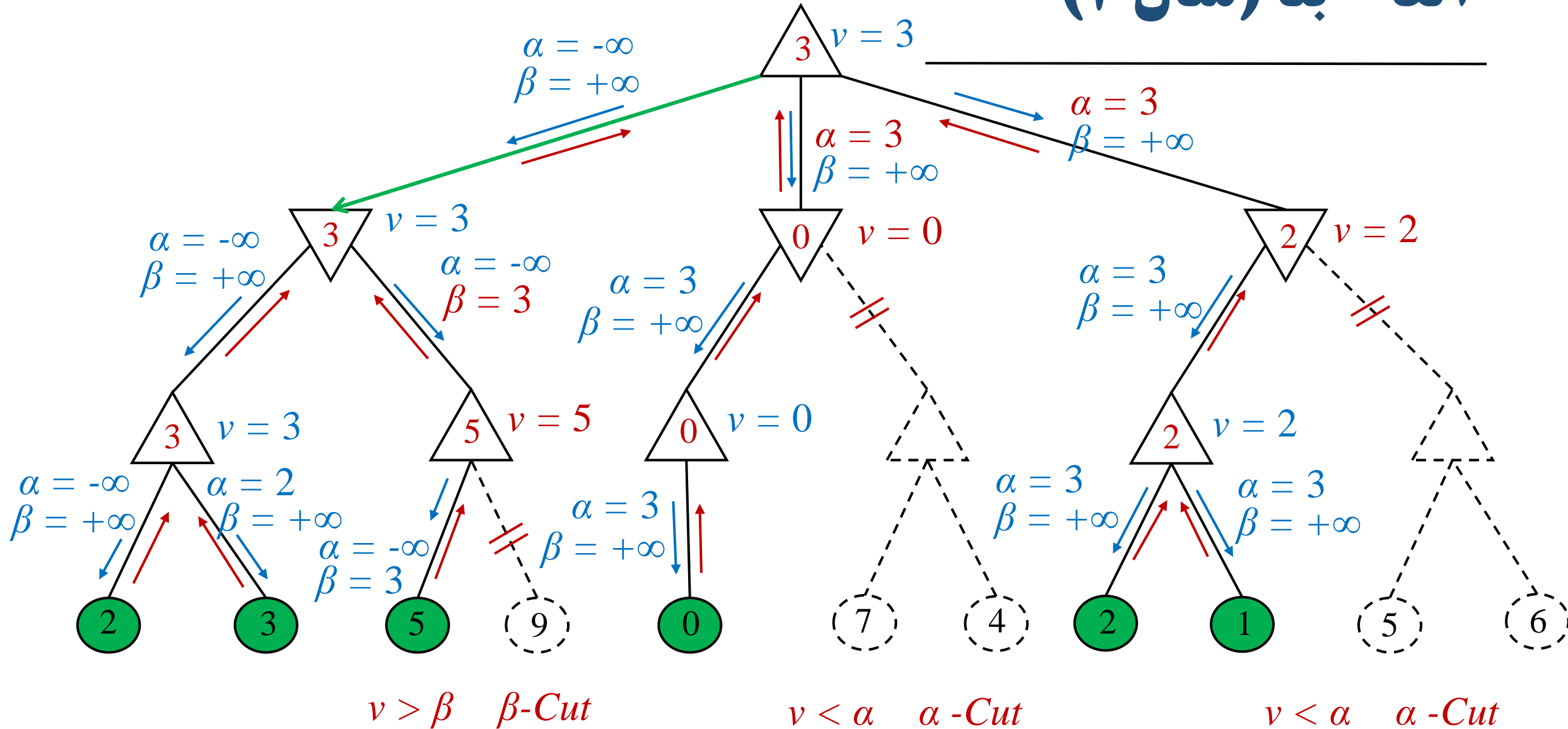
آلڦا-بتا (مثال ۲)



آلفا-بتا (مثال ۲)



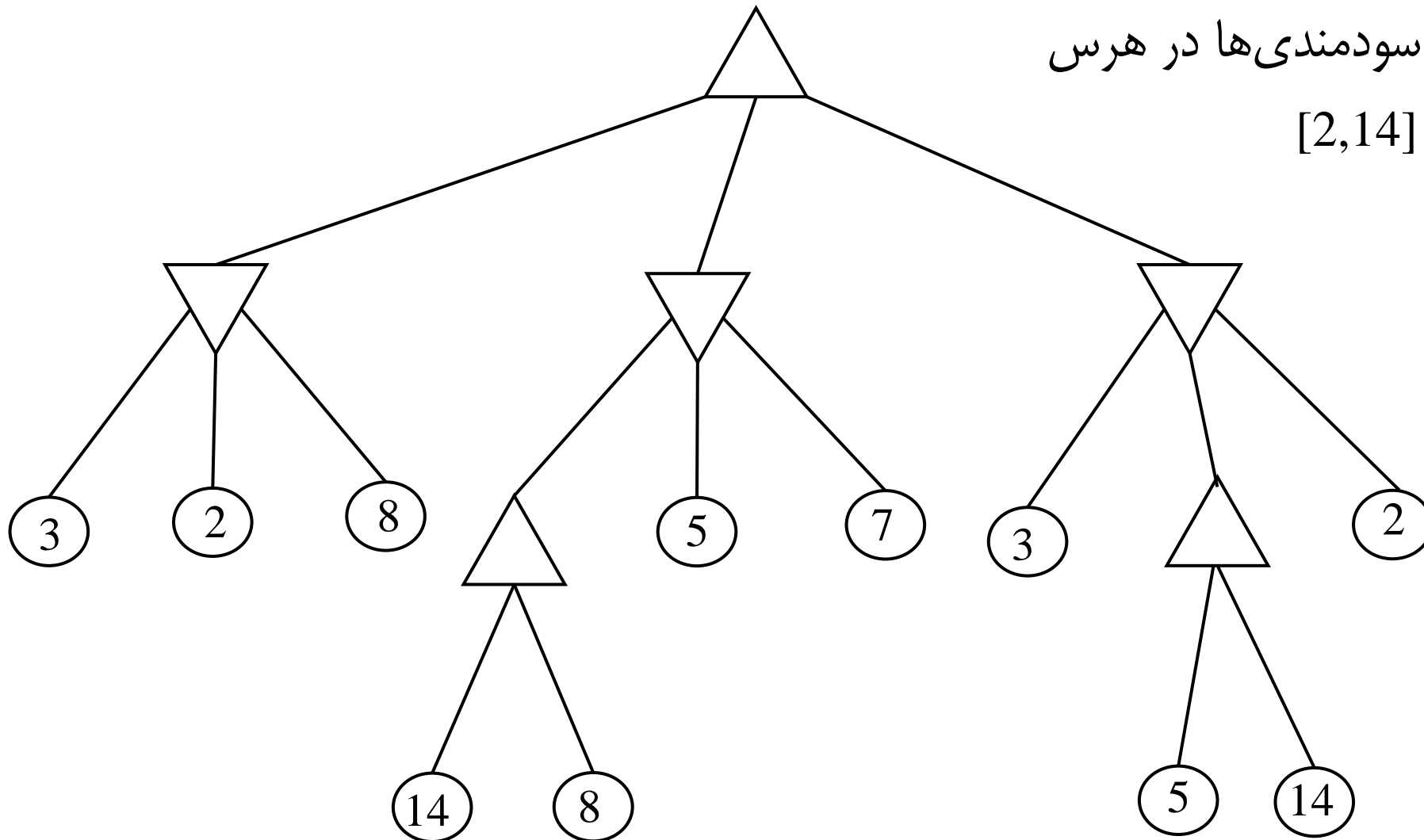
آلفا-بتا (مثال ۲)



ویژگی‌های الگوریتم آلفا-بتا

- تاثیر دانستن بازه سودمندی‌ها در هرس

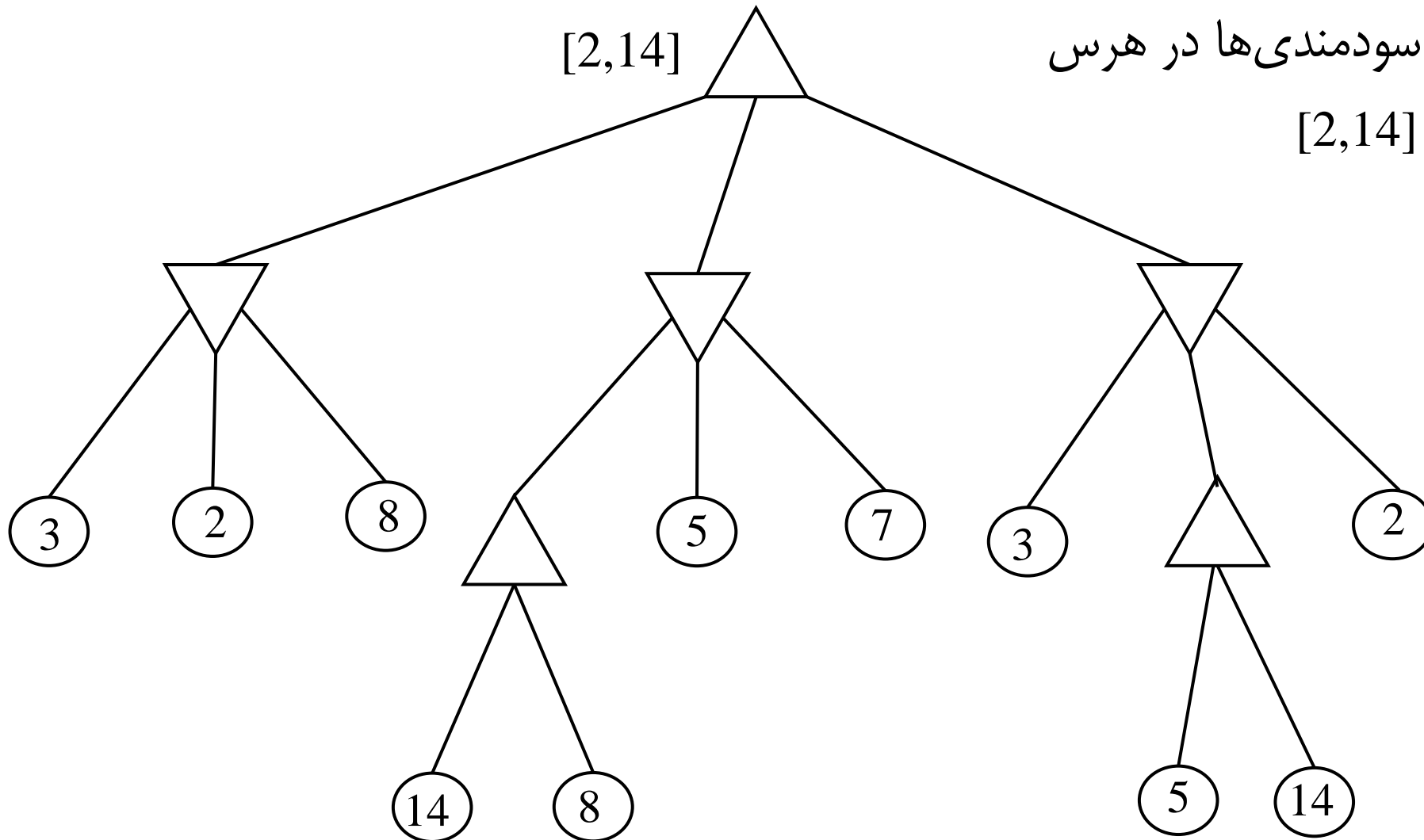
- برای مثال بازه $[2, 14]$



ویژگی‌های الگوریتم آلفا-بتا

- تاثیر دانستن بازه سودمندی‌ها در هرس

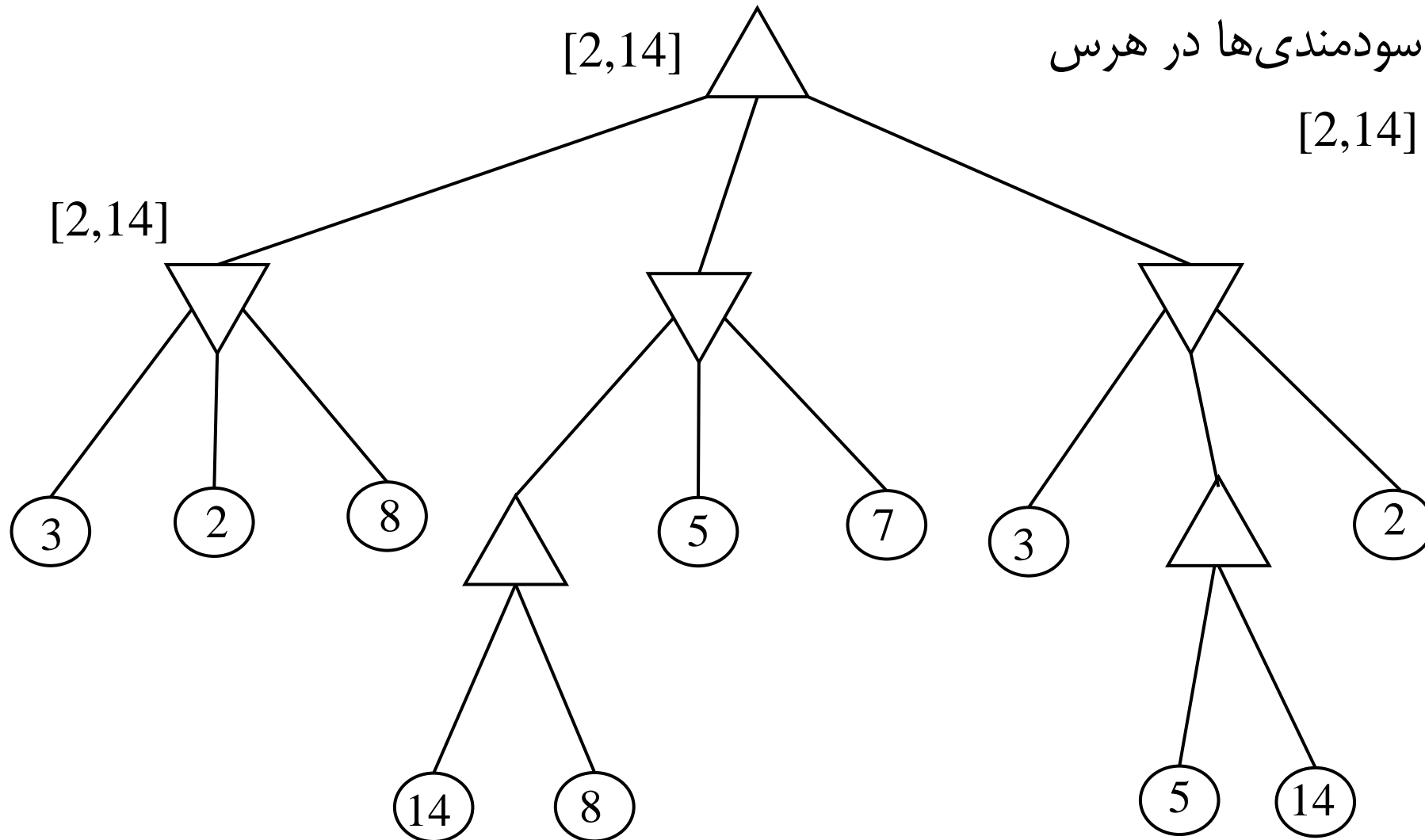
- برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

- تاثیر دانستن بازه سودمندی‌ها در هرس

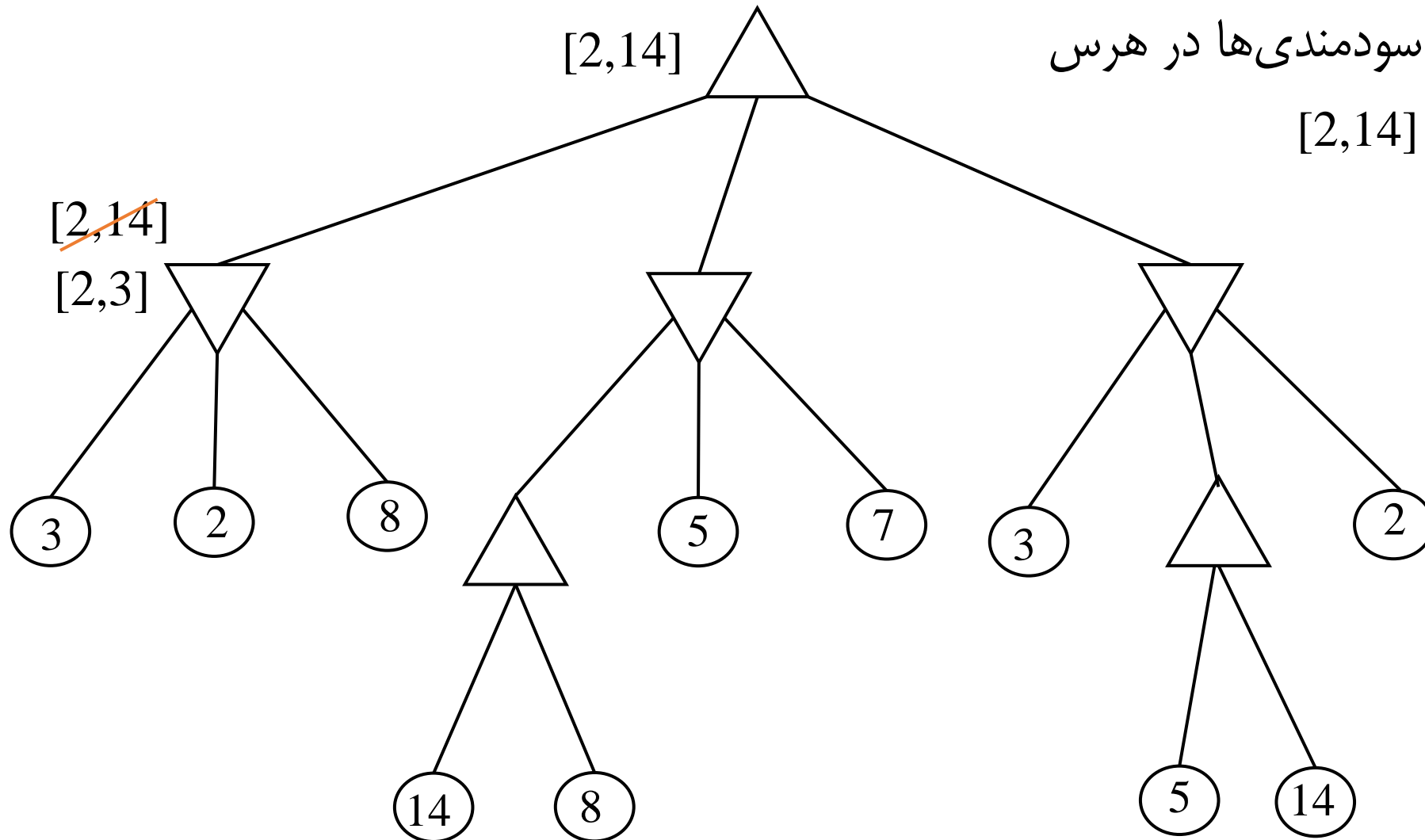
- برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

- تاثیر دانستن بازه سودمندی‌ها در هرس

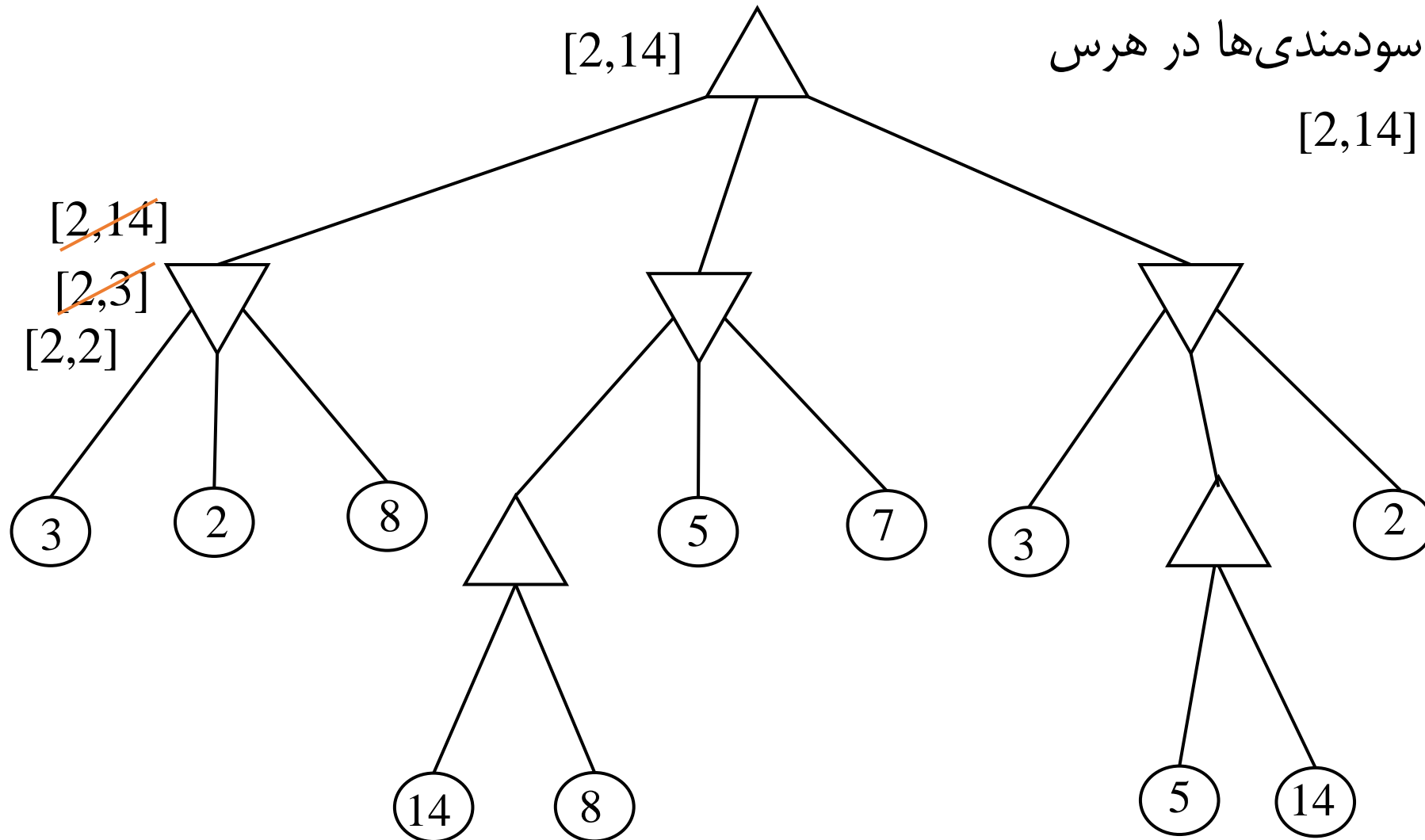
- برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

- تاثیر دانستن بازه سودمندی‌ها در هرس

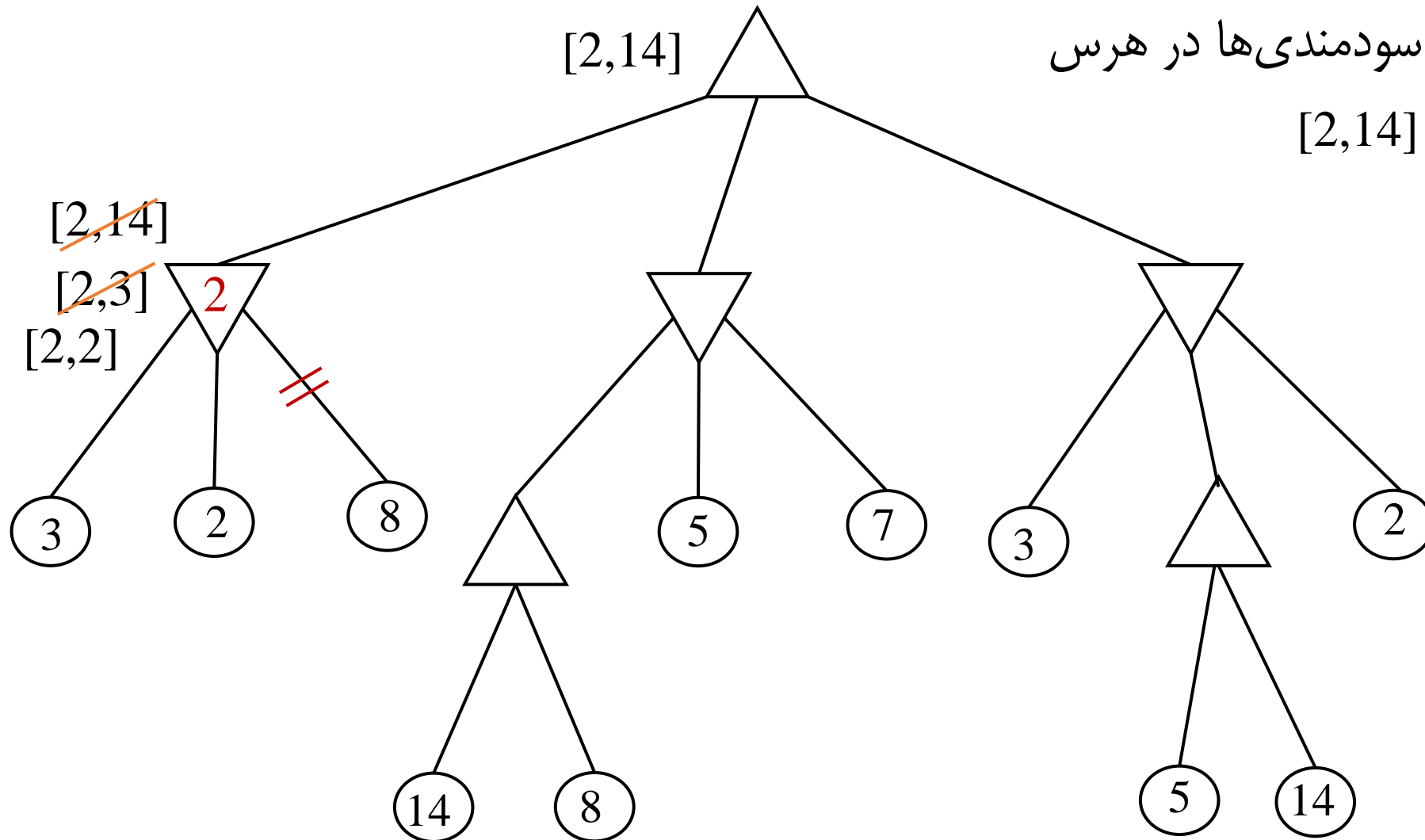
- برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

- تاثیر دانستن بازه سودمندی‌ها در هرس

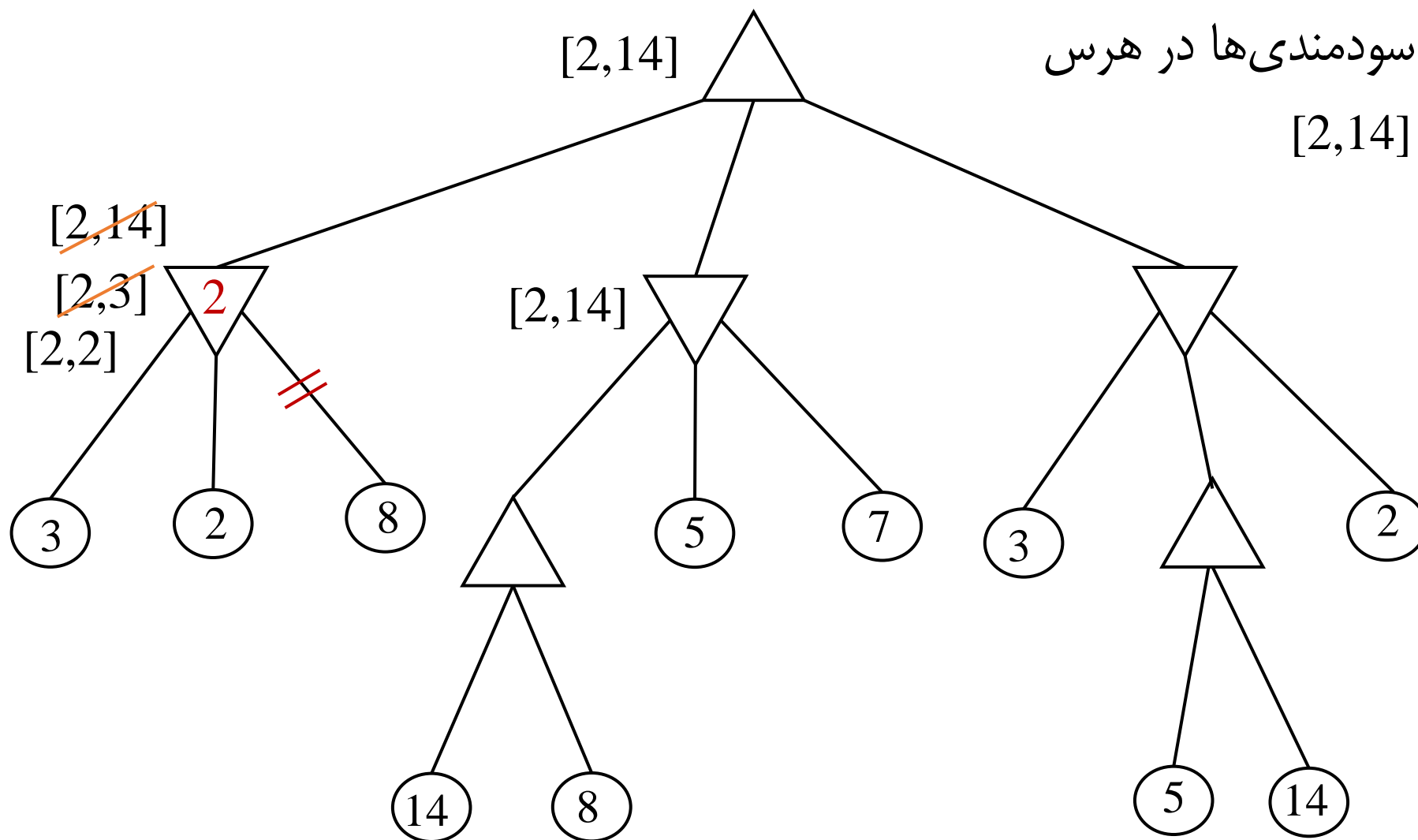
- برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

- تاثیر دانستن بازه سودمندی‌ها در هرس

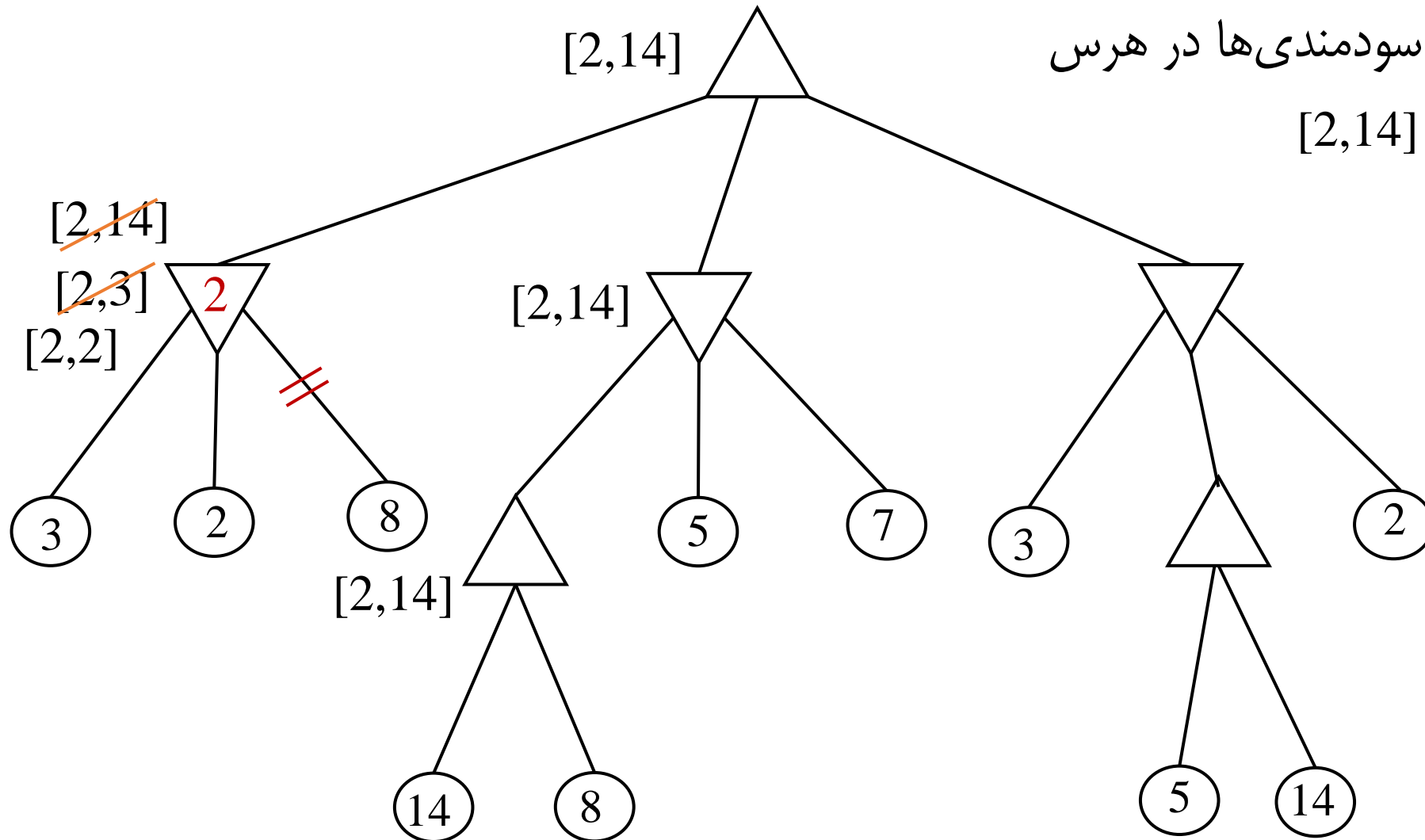
- برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

- تاثیر دانستن بازه سودمندی‌ها در هرس

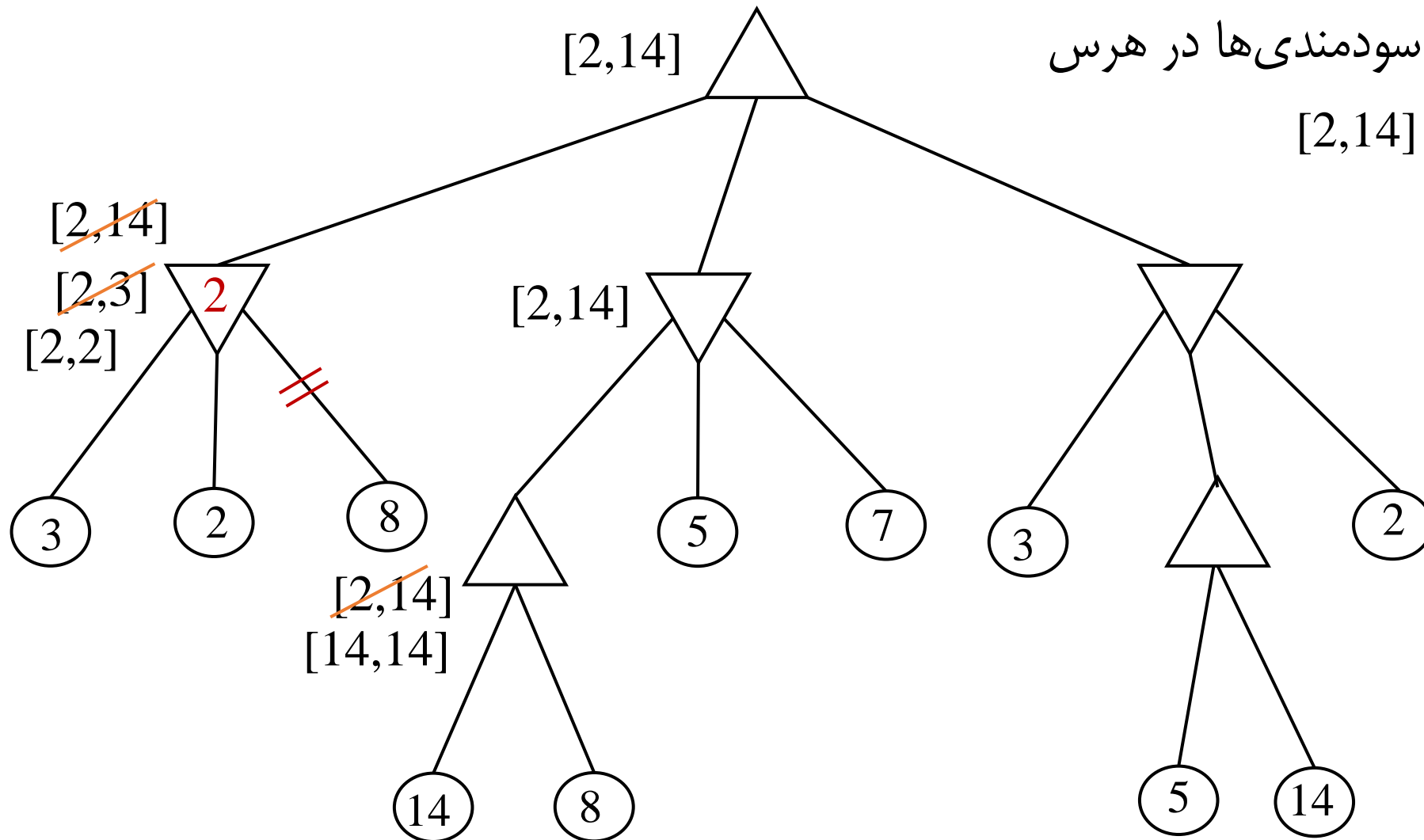
- برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

• تاثیر دانستن بازه سودمندی‌ها در هرس

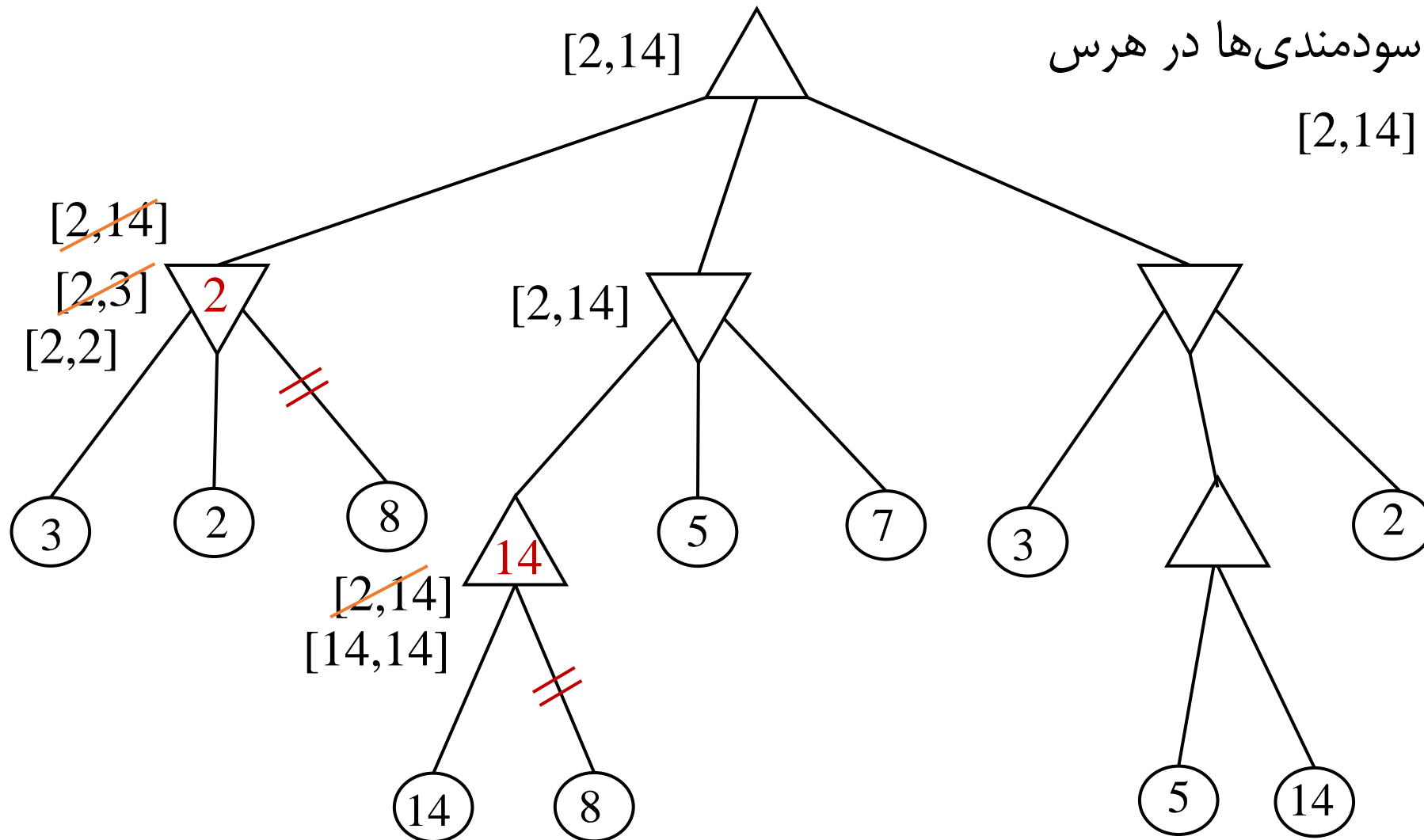
• برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

• تاثیر دانستن بازه سودمندی‌ها در هرس

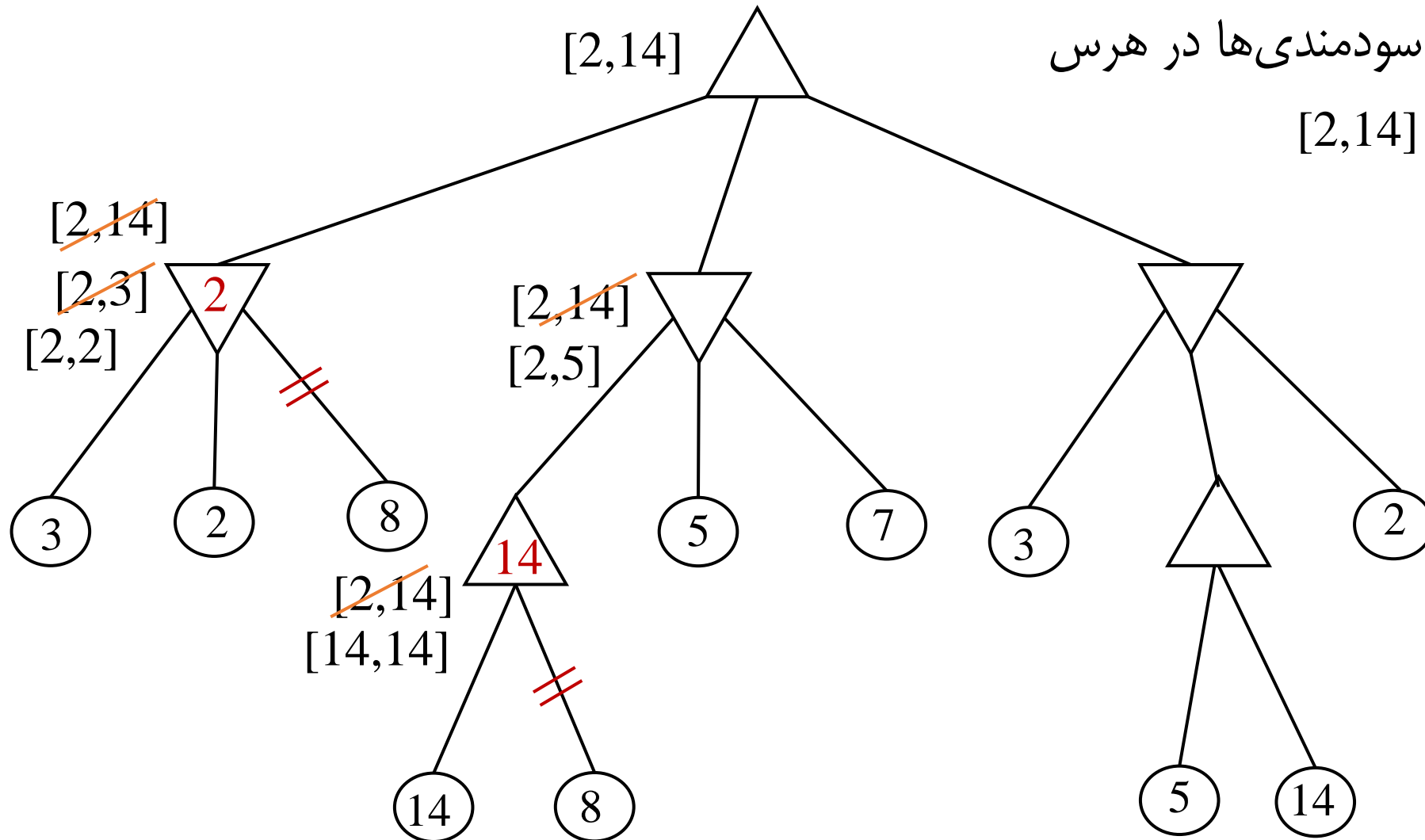
• برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

• تاثیر دانستن بازه سودمندی‌ها در هرس

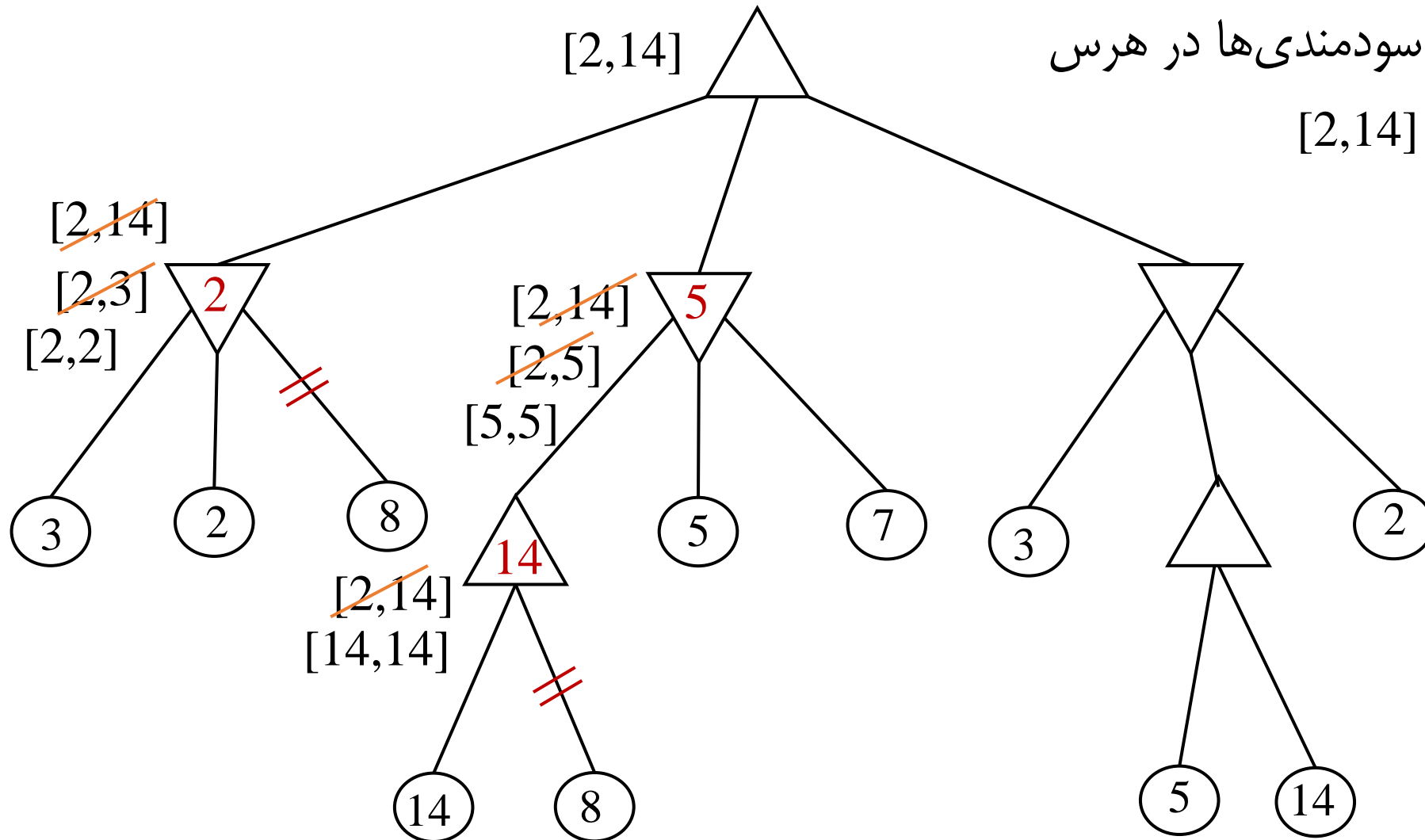
• برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

• تاثیر دانستن بازه سودمندی‌ها در هرس

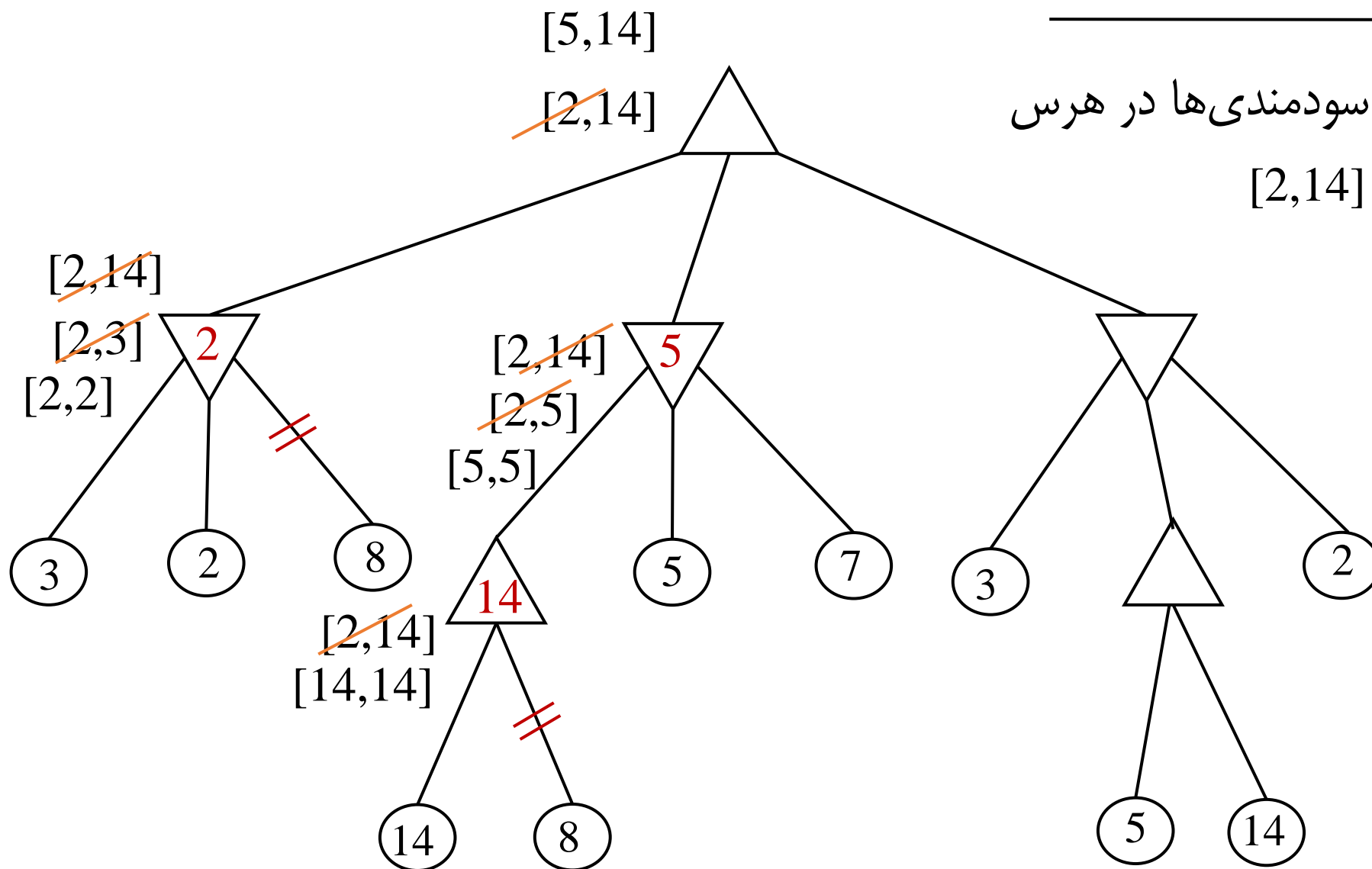
• برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

• تاثیر دانستن بازه سودمندی‌ها در هرس

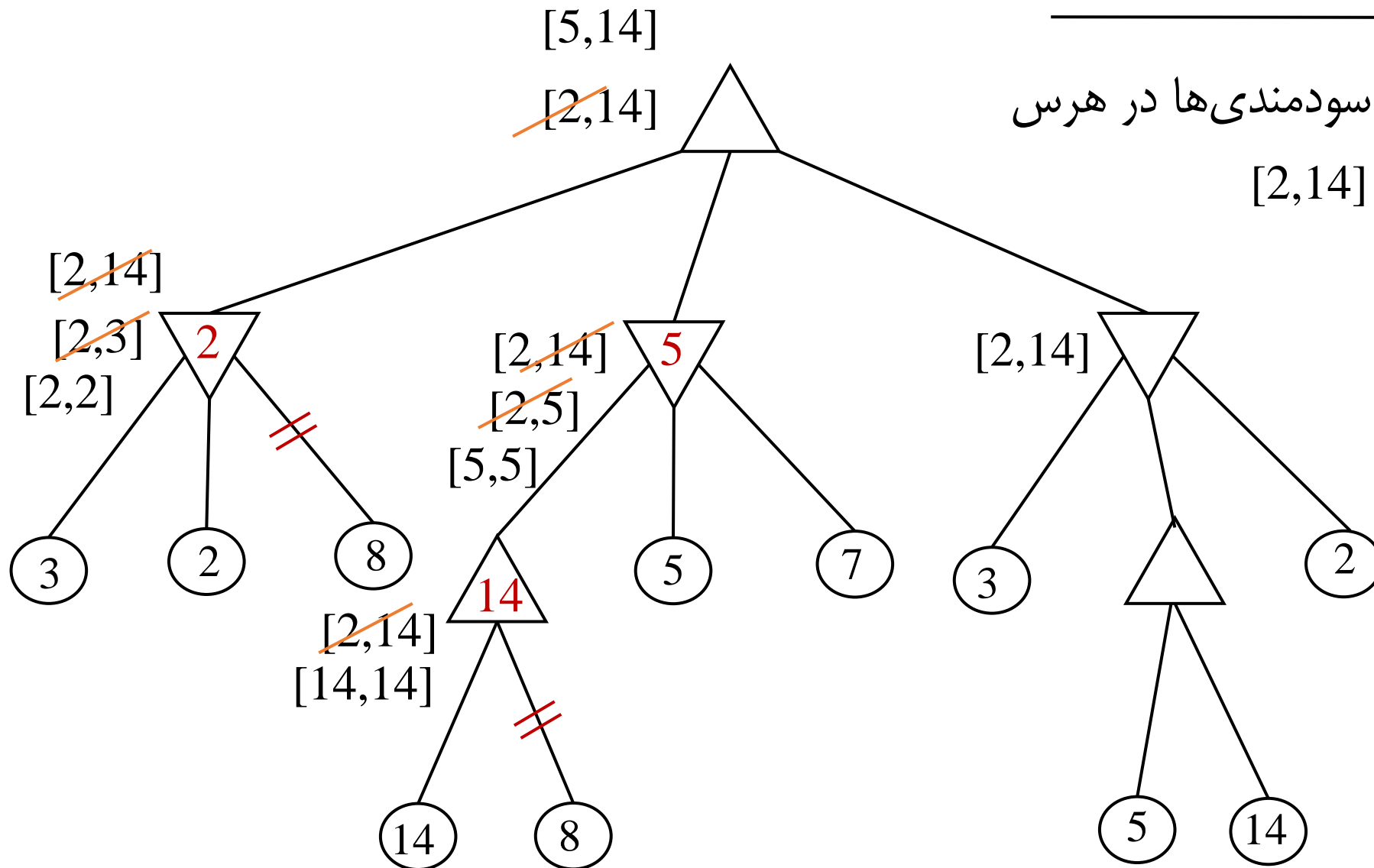
• برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

• تاثیر دانستن بازه سودمندی‌ها در هرس

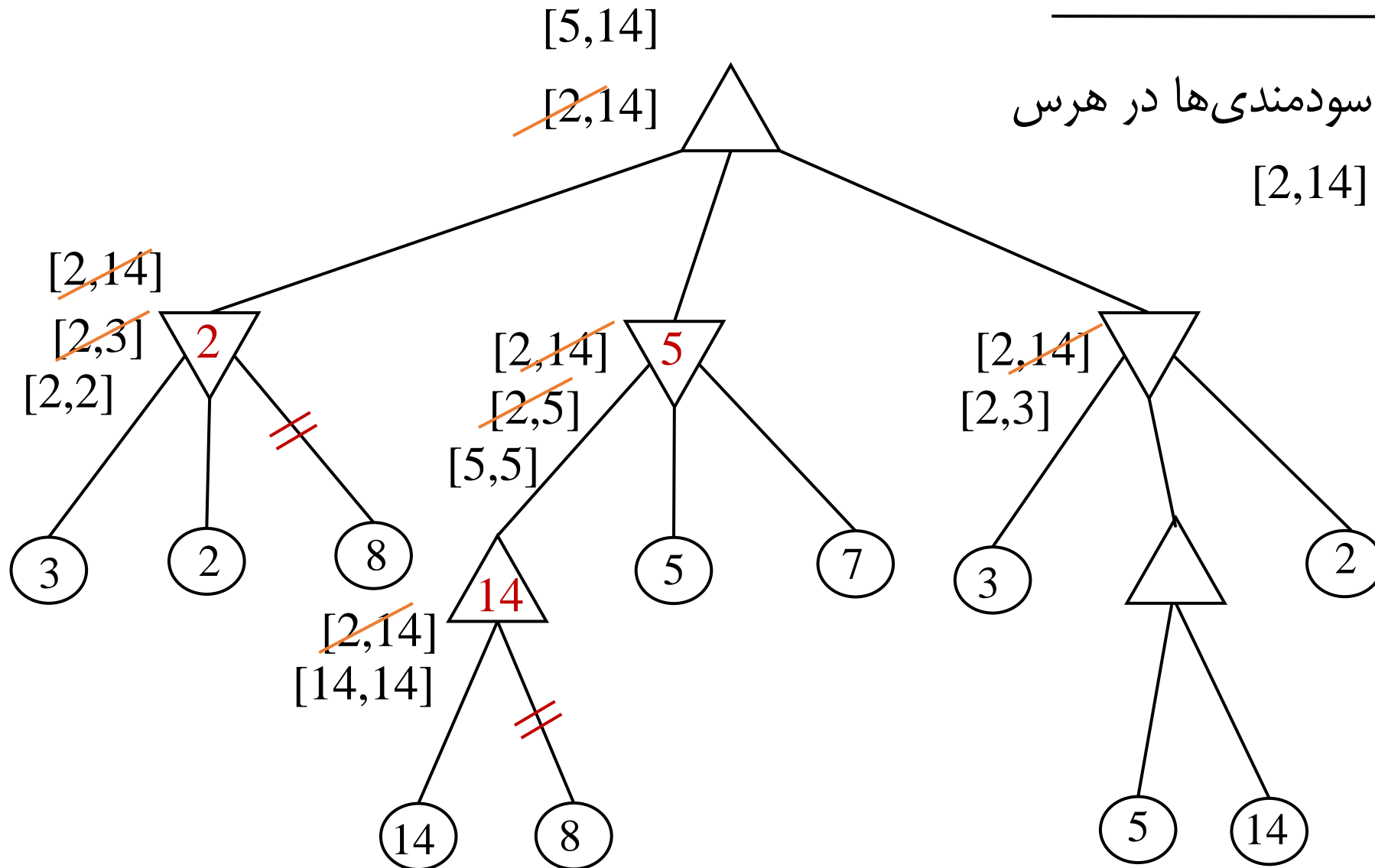
• برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

• تاثیر دانستن بازه سودمندی‌ها در هرس

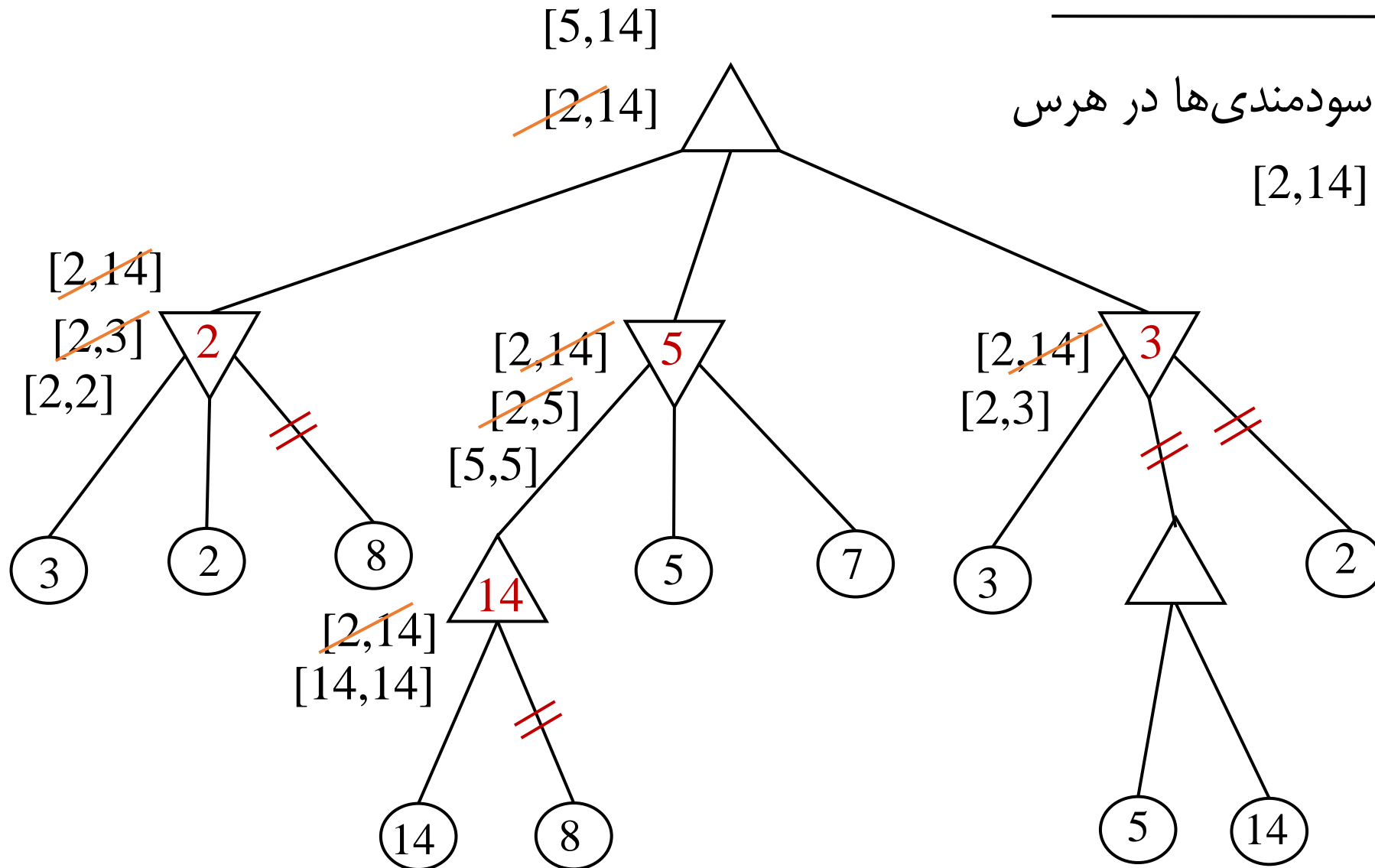
• برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

• تاثیر دانستن بازه سودمندی‌ها در هرس

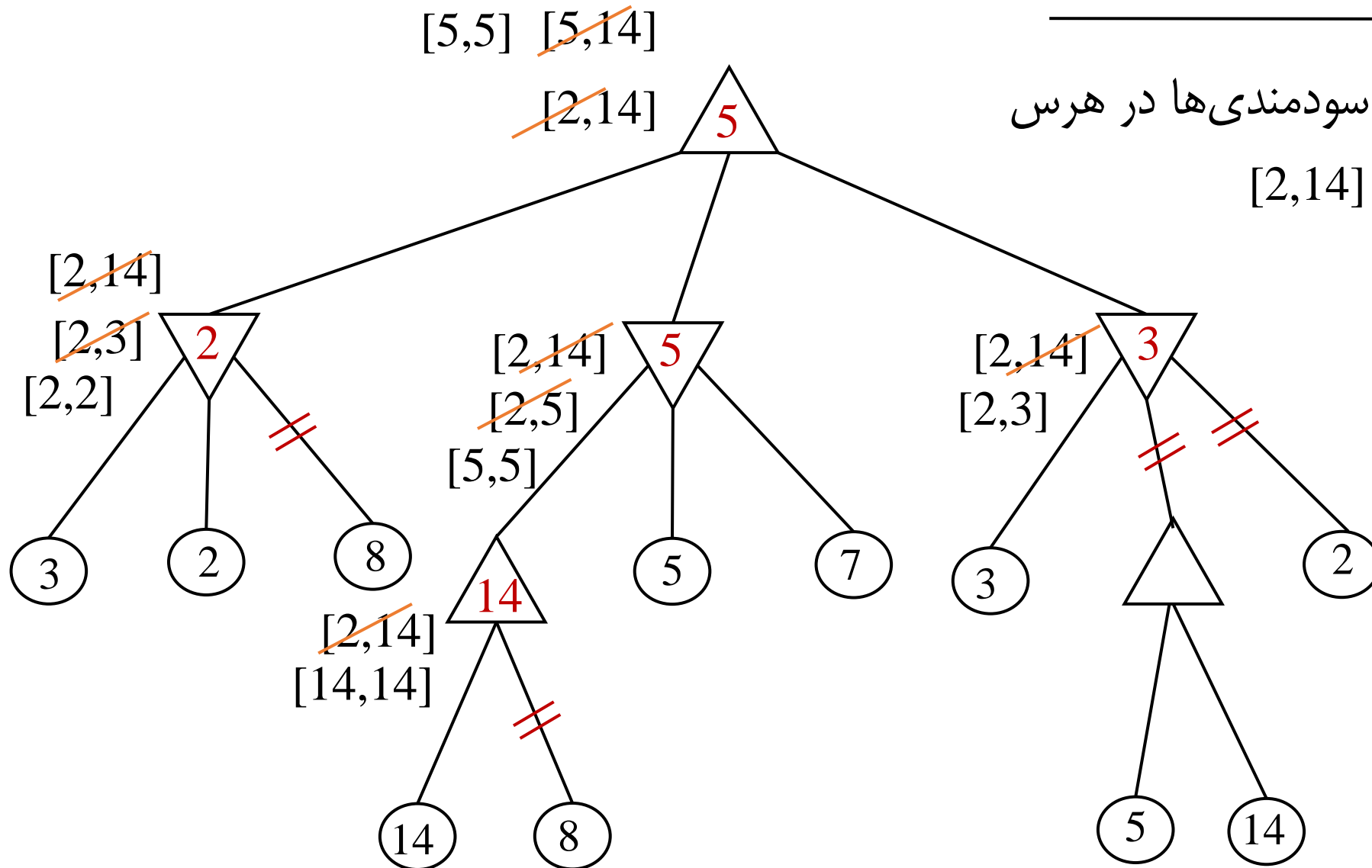
• برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

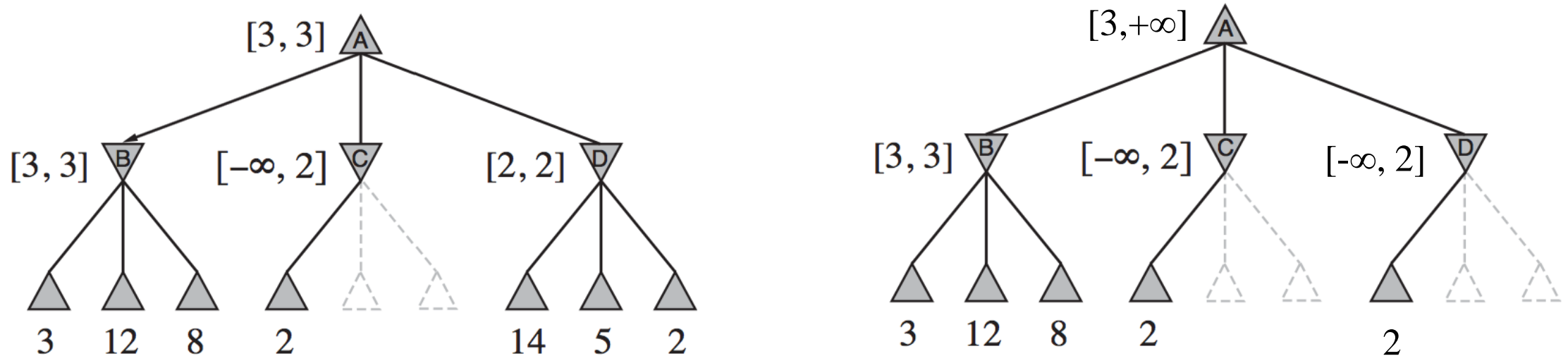
• تاثیر دانستن بازه سودمندی‌ها در هرس

• برای مثال بازه $[2,14]$



ویژگی‌های الگوریتم آلفا-بتا

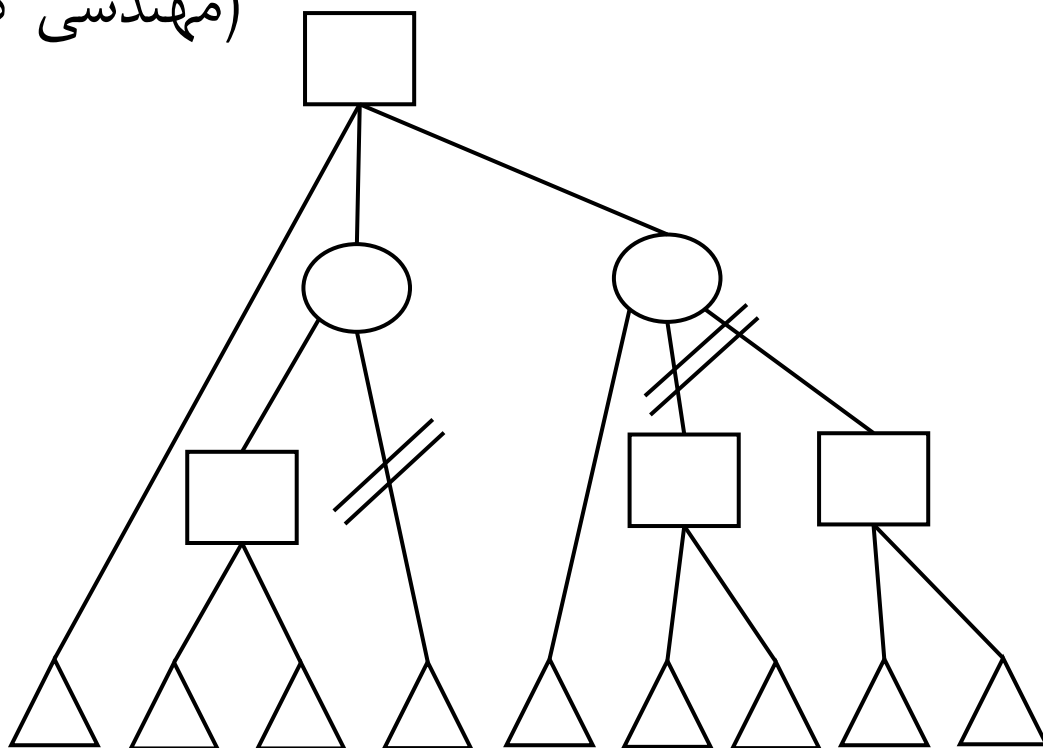
- ترتیب بررسی شاخه‌های درخت بر روی تعداد شاخه‌های هرس شده تاثیر می‌گذارد.
- برای بازی‌هایی که سودمندی گره‌ها از $-\infty$ تا $+\infty$ است بیشترین هرس زمانی اتفاق می‌افتد که
 - برای هر گره MAX، فرزند با بیشترین سودمندی آن در سمت چپ‌ترین شاخه باشد.
 - برای هر گره MIN، فرزند با کم‌ترین سودمندی آن در سمت چپ‌ترین شاخه باشد.



ویژگی‌های الگوریتم آلفا-بتا

- الگوریتم هرس آلفا-بتا در بهترین حالت می‌تواند حداکثر نصف شاخه‌های درخت را هرس کند.
- در این صورت پیچیدگی زمانی در بهترین حالت برابر است با $O(b^{m/2})$
- و اگر فاکتور انشعاب موثر برای MINIMAX برابر با b باشد برای آلفا-بتا برابر با \sqrt{b} است.
- سرعت الگوریتم هرس آلفا-بتا دو برابر الگوریتم بیشینه کمینه است. یعنی اگر در یک زمان معین، الگوریتم بیشینه کمینه تا عمق m از درخت را بررسی کند، الگوریتم هرس آلفا-بتا تا عمق $2m$ را بررسی خواهد کرد.
- اگر ترتیب بررسی شاخه‌ها و گره‌ها به‌طور تصادفی انتخاب شوند، تعداد کل گره‌هایی که بررسی می‌شوند، به‌طور متوسط برابر با $O(b^{3m/4})$ است.

در گراف مقابل مربع نشانه بازیکن Max، دایره نشانه بازیکن Min و مثلث نشانه حالت پایانی است. اگر مقادیر ارزیابی بتوانند در فاصله بسته $[0,10]$ باشند و با هرس آلفا-بتا فقط یال‌های علامت زده شده // حذف شوند، ترتیب گره‌های پایانی به ترتیب از چپ به راست در شکل کدام یک از گزینه‌های زیر خواهد بود؟



۱) ۱، ۴، ۱، ۰، ۱۰، ۹، ۳، ۵، ۲، ۸

۲) ۱۰، ۹، ۸، ۵، ۴، ۳، ۲، ۱، ۰

۳) ۹، ۵، ۳، ۱۰، ۸، ۰، ۱، ۲، ۴ ✓

۴) ۰، ۱، ۲، ۳، ۴، ۹، ۸، ۹، ۱۰

تصمیمات بی‌درنگ ناکامل

- الگوریتم هرس آلفا-بتا نیاز دارد که برخی از شاخه‌ها را تا برگ‌ها پیمایش کند.
- در عمل گره‌های پایانه عمق زیادی دارند و رسیدن به آن‌ها زمان زیادی را می‌برد. در حالی که انتظار داریم الگوریتم تصمیم‌گیری در زمان معقولی پایان پذیرد.
- برای سریع‌تر شدن تصمیم‌گیری می‌توانیم درخت جست‌وجو را تا برگ‌ها پیمایش نکنیم بلکه آن را تا عمق خاصی که عمق برش (cutoff) نامیده می‌شود، پیمایش کنیم و سپس از روی سودمندی گره‌های آن عمق که احتمالا پایانه نیستند و انجام الگوریتم Minimax یا هرس آلفا-بتا تصمیم‌گیری کنیم.
- چگونه می‌توان سودمندی گره‌های غیر پایانه را در عمق برش تعیین کرد؟
- تا چه عمقی از درخت بازی را بهتر است پیمایش کنیم؟

تعیین سودمندی گره‌های غیر پایانه

- تعریف تابع ارزیابی (EVAL): تابعی است که تخمینی از سودمندی مورد انتظار یک وضعیت غیر پایانه بازی را برمی‌گرداند.

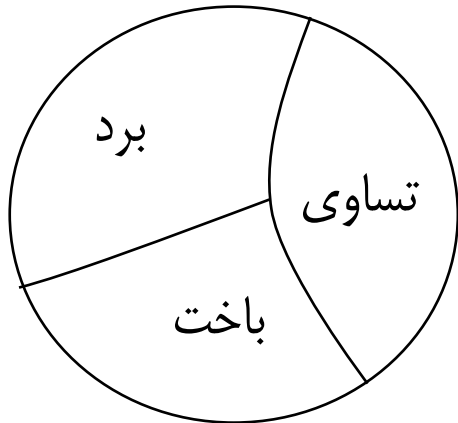
$$H\text{-MINIMAX}(s, d) =$$

$$\begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s, d) \\ \max_{a \in \text{Actions}(s)} H\text{-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} H\text{-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}$$

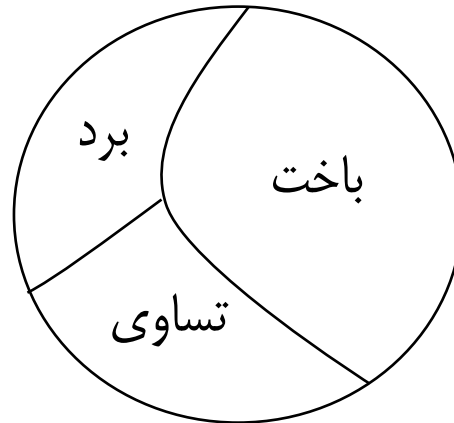
- ویژگی‌های تابع ارزیابی مطلوب:
- باید حالت‌های پایانه را متناسب با تابع سودمندی ارزش‌دهی کند.
- زمان لازم برای ارزیابی نباید زیاد شود.
- در مورد حالات غیرپایانی، تابع ارزیابی باید وابستگی زیادی به شانس‌های واقعی بُرد داشته باشد.

تعریف مناسب یک تابع ارزیابی

- بیشتر توابع ارزیاب به وسیله محاسبه خصوصیات گوناگون حالت، کار می کنند.
- برای مثال تعداد سربازان در بازی شطرنج
- این خصوصیات در مجموع، طبقات یا دسته های مشابهی از حالات را تعریف می کنند.
- حالتی که در یک دسته قرار می گیرند، دارای مقادیر یکسانی برای تمامی خصوصیات می باشند.
- هر دسته مفروض شامل حالاتی خواهد بود که برخی از آن ها به بُرد، برخی به تساوی و برخی دیگر به شکست منجر می شوند.

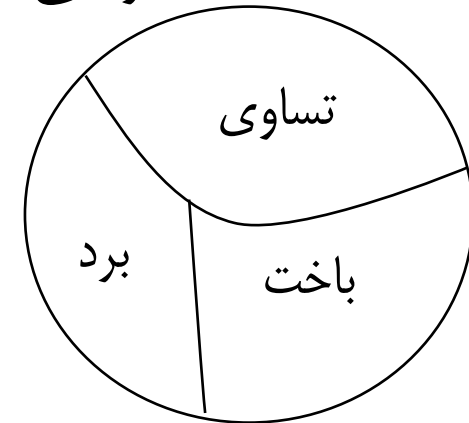


$$f_1 = v_1^1, \dots, f_n = v_n^1$$



$$f_1 = v_1^2, \dots, f_n = v_n^2$$

...



$$f_1 = v_1^m, \dots, f_n = v_n^m$$

تعریف مناسب یک تابع ارزیابی

- تابع ارزیاب قابلیت تشخیص این حالات را ندارد اما می‌تواند مقداری را که نشان‌گر نسبت حالات با هر خروجی است را تولید نماید.
- مثال: فرض کنید که براساس تجربه ما، از میان حالات یک دسته مشخص، ۷۲٪ از حالات به برد (۱)، ۲۰٪ به شکست (۰) و ۸٪ به تساوی (۱/۲) منجر شود:

$$(0.72 \times +1) + (0.20 \times 0) + (0.08 \times 1/2) = 0.76$$

- تابع ارزیابی نیازی ندارد ارزش گره‌های پایانی را نزدیک به مقدار واقعی سودمندی آن‌ها تخمین بزند، بلکه تنها کافی است گره‌ها را همانند تابع سودمندی مرتب کند، یعنی به موقعیت‌های بهتر، ارزش بیشتری دهد.
- در عمل این نوع تحلیل و بررسی به دسته‌ها و بنابراین تجربه‌های بسیار زیادی به منظور تخمین تمام احتمال‌های برد نیاز دارد.

تعریف مناسب یک تابع ارزیابی

- بنابراین، برای تخمین سودمندی یک وضعیت غیر پایانه در یک بازی، برای آن بازی یک سری ویژگی در نظر می‌گیریم و به هر کدام یک ارزش نسبت می‌دهیم بعد از آن با ترکیب ارزش‌های ویژگی‌ها، سودمندی یک وضعیت را تخمین می‌زنیم.
- مثال:

• f_i : تعداد هر نوع قطعه در صفحه

• w_i : ارزش آن قطعات (۱ برای پیاده، ۳ برای اسب یا فیل، ۵ برای رخ و ...)

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

- ارزش دو فیل از دو برابر ارزش یک فیل بیشتر است یا این که ارزش یک فیل در انتهای بازی بیشتر می‌شود. بنابراین بهتر است که یک تابع غیرخطی از ویژگی‌ها را به عنوان تابع ارزیابی استفاده کرد.

پایان دادن به جستجو

- در زمان مناسب جستجو را قطع و تابع هیوریستیک EVAL را فراخوانی می کند.
$$\text{if CUTOFF-TEST}(\text{state}, \text{depth}) \text{ then return EVAL}(\text{state})$$
- جستجوی عمقی با حداکثر عمق محدود:
- درخت جستجو را تا عمق مشخص d پیمایش کنیم.
- مشکل در تعیین عمق d : اگر d بزرگ باشد تصمیم دقیق تری می گیرد اما ممکن است از زمان قابل قبول فراتر رود. برعکس اگر d کوچک باشد الگوریتم سریع تصمیم می گیرد ولی از کل زمان مجاز خود استفاده نمی کند.
- استفاده از جستجوی عمقی تکراری (IDS):
- از این جستجو استفاده می کنیم و تا اتمام زمان مجاز جستجو را ادامه می دهیم.
- در این صورت تا جای ممکن عمق بیشتری از درخت را دیده ایم و تصمیم دقیق تری می گیریم.

پایان دادن به جستجو

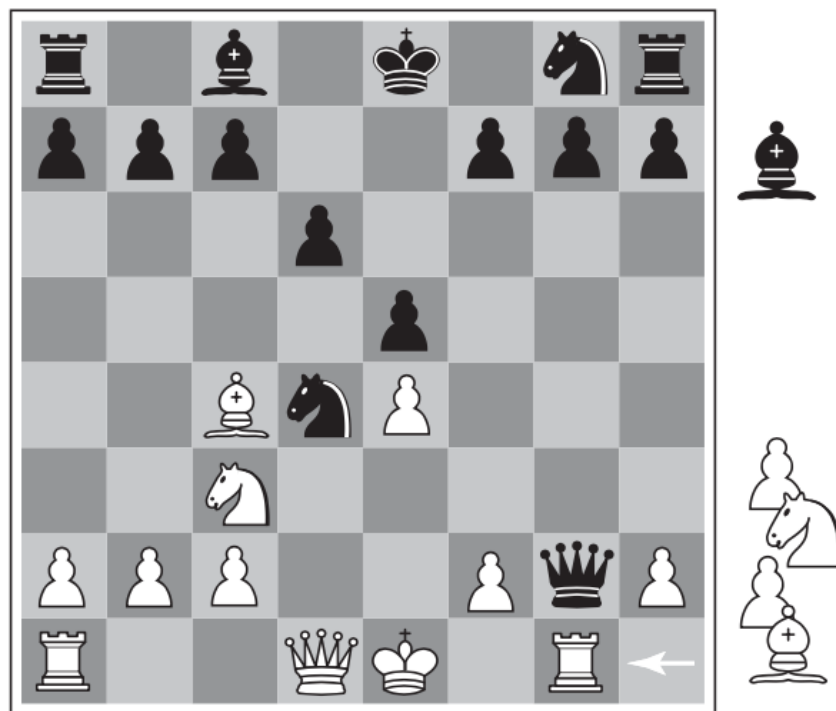
• هر دو روش گفته شده در اسلاید قبل به دلیل غیر دقیق بودن تابع ارزیابی ممکن است دچار مشکلاتی شوند:

۱- وضعیت غیر ساکن

۲- اثر افق

مشکل وضعیت‌های غیر ساکن

- **وضعیت ساکن (quiescent):** وضعیتی است که به احتمال زیاد در آینده نزدیک تغییر چندانی نمی‌کند. چه فرزندانش بررسی شوند یا نه ارزش تخمینی آن تغییر چندانی نمی‌کند و جانشین‌های آن تا چند مرحله جلوتر ارزشی نزدیک به ارزش آن داشته باشند.



(b) White to move

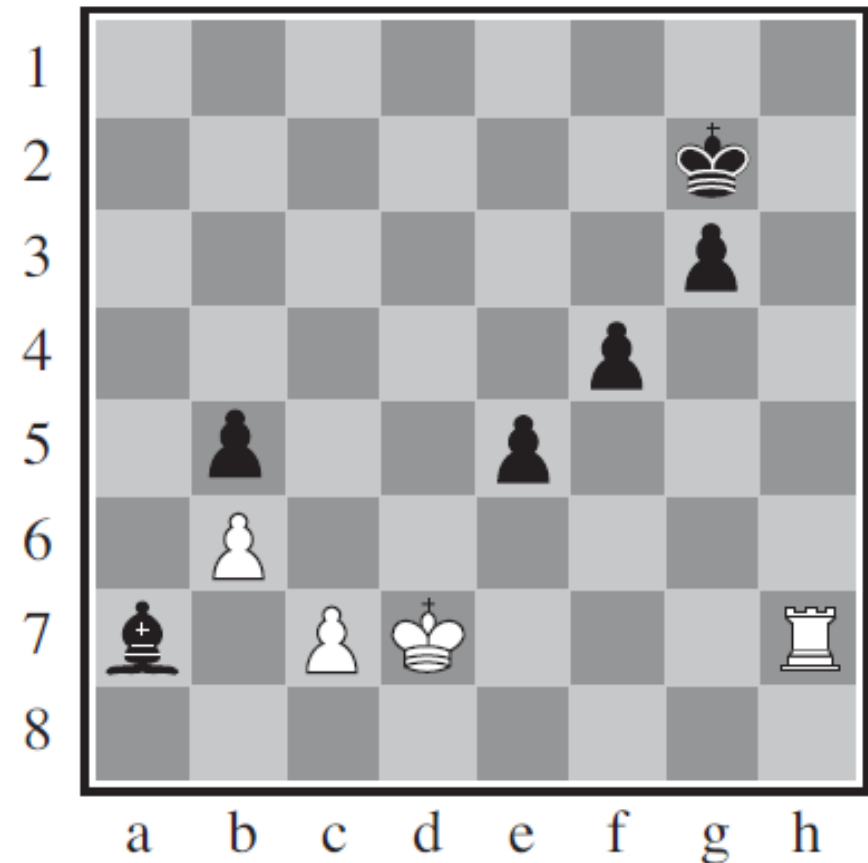
مشکل وضعیت‌های غیر ساکن

- وضعیت‌های غیر ساکن زمانی مشکل‌ساز هستند که یک گره در عمق برش، متناظر با یک وضعیت غیر ساکن باشد. در این حالت چون سودمندی این گره توسط تابع ارزیابی غیردقیق تخمین زده می‌شود، تصمیم‌گیری صورت گرفته نیز غیردقیق خواهد بود.
- برای حل این مشکل باید تابع ارزیابی را فقط به وضعیت‌های ساکن اعمال کرد. وضعیت‌هایی که ساکن نیستند را می‌توان بسط داد تا به وضعیت‌های ساکن برسیم و سپس آن وضعیت‌های ساکن را ارزیابی کرد. به این کار **جستجوی سکون (quiescence search)** گویند.

مشکل اثر افق (Horizon effect)

- وقتی به وجود می آید که برنامه با حرکتی از سوی رقیب مواجه شود که اثرات مخرب زیادی در پی دارد و قابل اجتناب نیست اما می توان آن را به تاخیر انداخت.

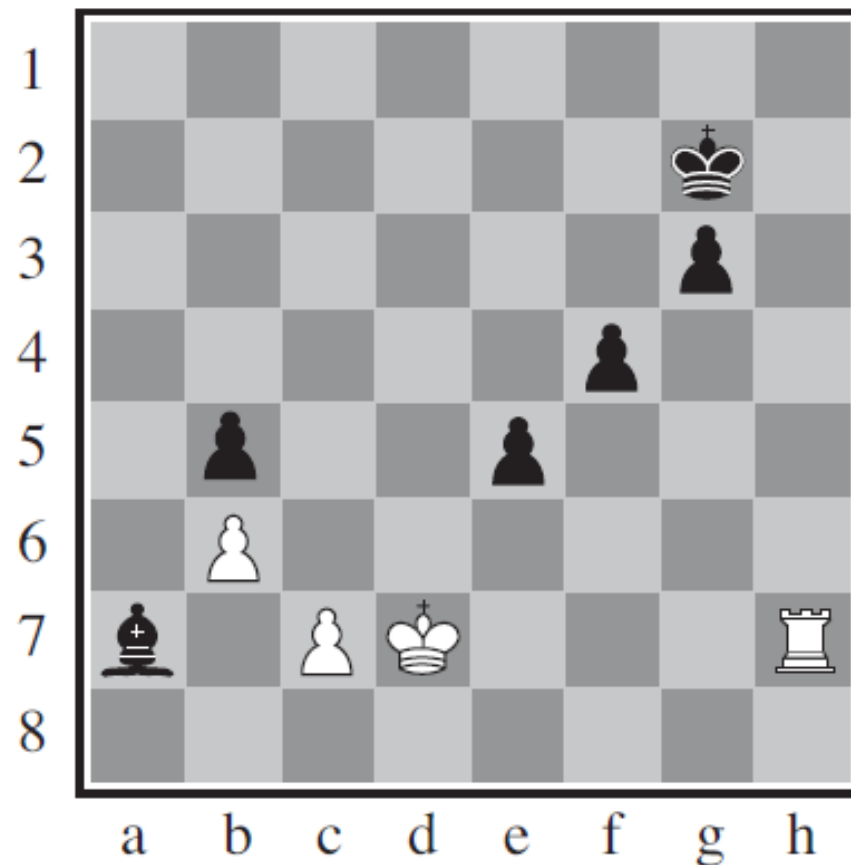
● سرباز	$f4 \rightarrow f5$
○ شاه	$d7 \rightarrow e7$
● فیل	$a7 \rightarrow b8$
○ قلعه	$h7 \rightarrow h8$
● سرباز	$g3 \rightarrow g4$
○ قلعه	$h8 \rightarrow b8$



مشکل اثر افق (Horizon effect)

- وقتی به وجود می آید که برنامه با حرکتی از سوی رقیب مواجه شود که اثرات مخرب زیادی در پی دارد و قابل اجتناب نیست اما می توان آن را به تاخیر انداخت.

● سرباز	$e5 \rightarrow e6$	● سرباز	$f4 \rightarrow f5$
○ شاه	$d7 \rightarrow e6$	○ شاه	$d7 \rightarrow e7$
● سرباز	$f4 \rightarrow f5$	● فیل	$a7 \rightarrow b8$
○ شاه	$e6 \rightarrow f5$	○ قلعه	$h7 \rightarrow h8$
● سرباز	$g3 \rightarrow g4$	● سرباز	$g3 \rightarrow g4$
○ شاه	$f5 \rightarrow g4$	○ قلعه	$h8 \rightarrow b8$
● فیل	$a7 \rightarrow b8$		
○ قلعه	$h7 \rightarrow g7$		
● فیل	$b8 \rightarrow c7$		
○ قلعه	$g7 \rightarrow c7$		



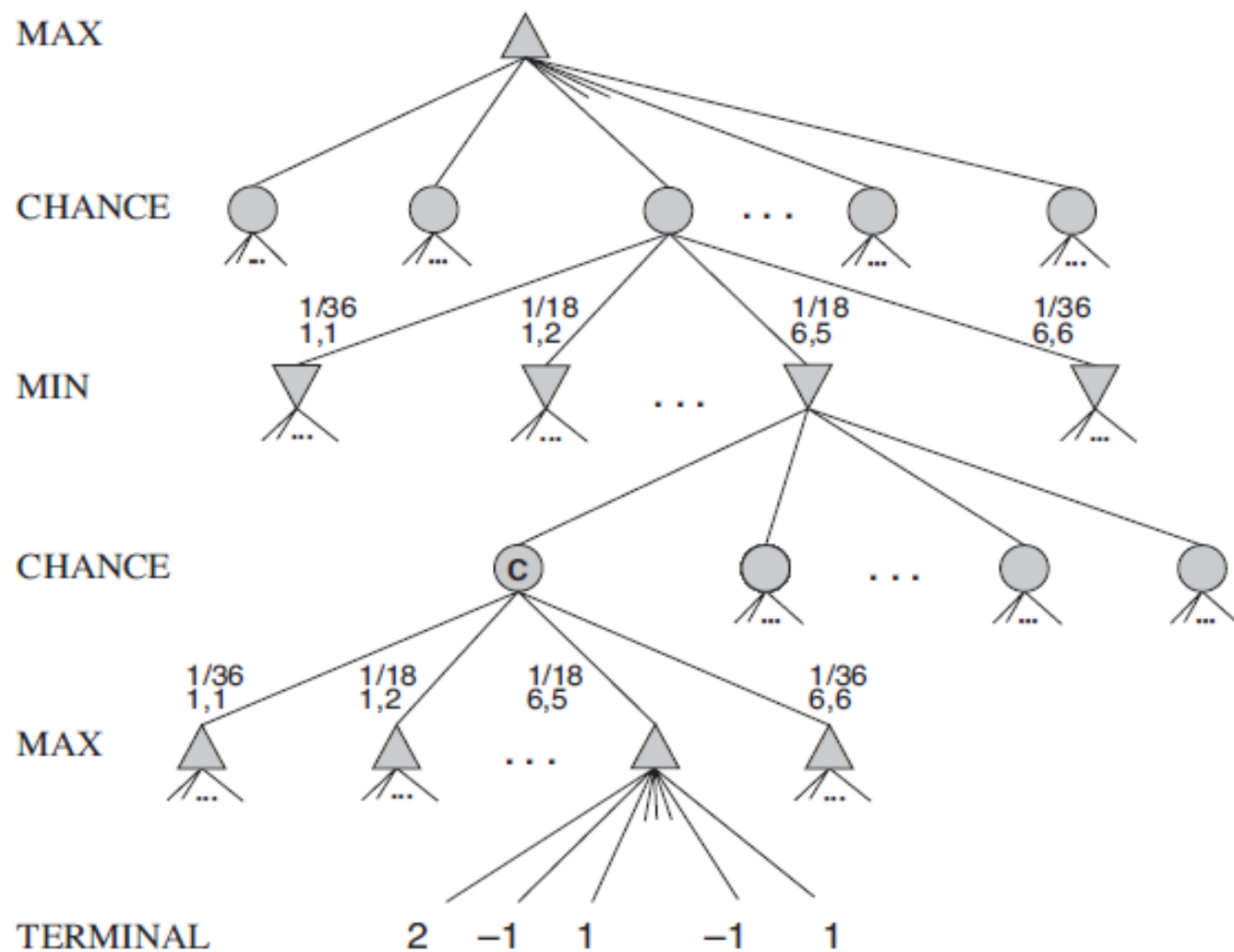
کم کردن مشکل اثر افق

- استفاده از سخت افزار قوی تر:
- در این صورت می توان جستجو را تا عمق بیشتری ادامه داد.
- استفاده از بسط های یکتا (Singular Extension):
- به حرکتی گفته می شود که از تمام حرکتهای ممکن دیگر در یک وضعیت خاص بهتر باشد. اگر بدانیم یکی از فرزندان یک گره از بقیه فرزنداناش بهتر است. فقط آن فرزند را بررسی می کنیم و بسط می دهیم.

هرس کردن پیش‌رو (Forward pruning)

- منظور از هرس کردن پیش‌رو این است که بعضی از حرکات‌ها در یک گره فوراً و بدون بررسی حذف می‌شوند. مثلاً می‌توان گره‌هایی که طبق تابع ارزیابی مناسب نیستند را دیگر ادامه نداد.
- یک روش انجام هرس پیش‌رو جست‌وجوی پرتو است یعنی در هر لایه به‌جای در نظر گرفتن تمام حرکات‌های ممکن فقط «پرتوی» از n بهترین حرکت را در نظر می‌گیریم.
- متأسفانه این روش بسیار خطرناک است زیرا هیچ تضمینی وجود ندارد که بهترین حرکت هرس نشود.
- هرس کردن پیش‌رو بهتر است برای گره‌های در عمق زیاد از درخت جست‌جو انجام شود. اگر این کار در نزدیکی ریشه انجام شود می‌تواند فاجعه‌انگیز باشد چون الگوریتم بسیاری از حرکات‌های بدیهی را از دست خواهد داد.

بازی های حاوی عنصر شانس

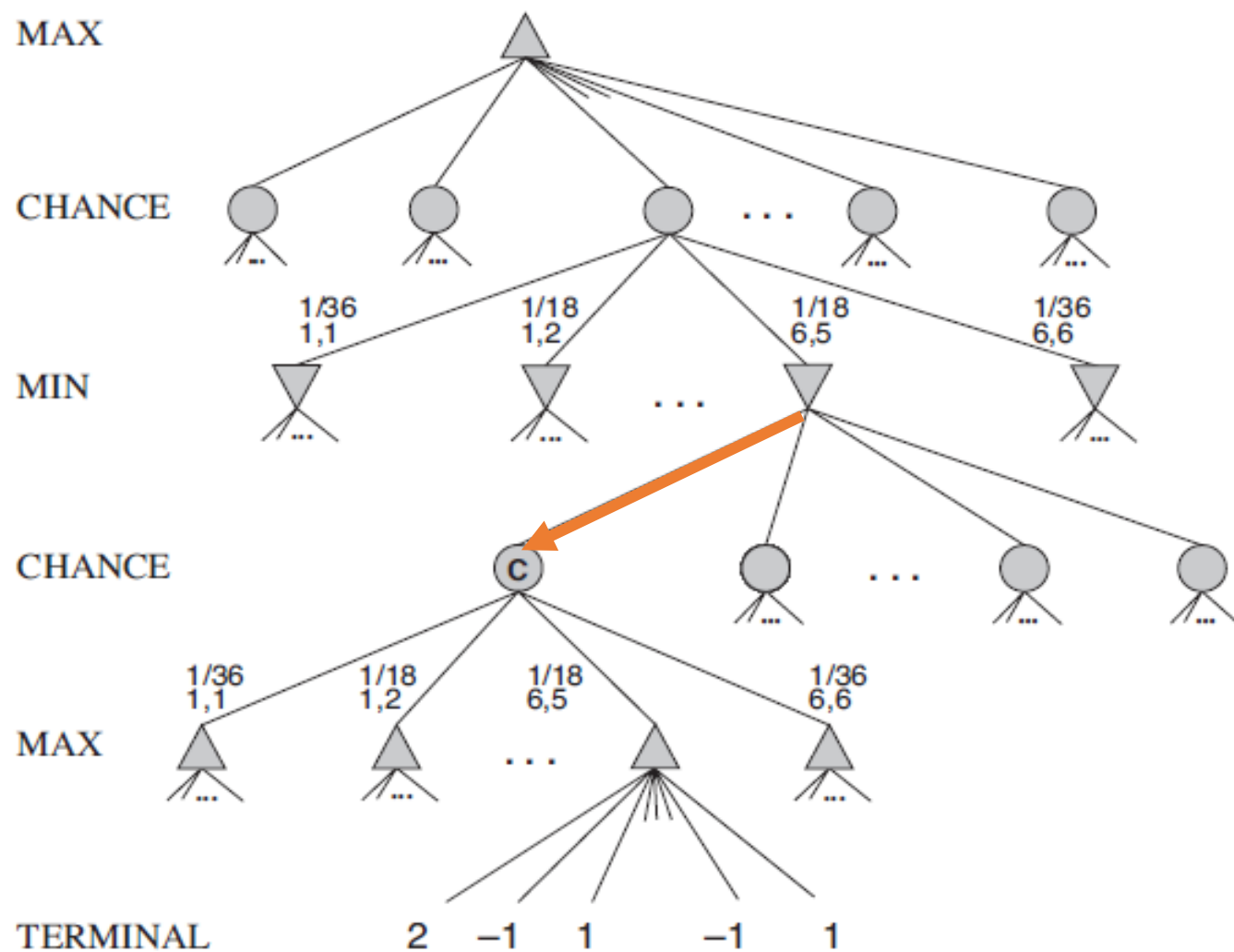


- در بسیاری از بازی ها یک عنصر شانس مانند تاس یا سکه وجود دارد که بازیکن ها به نوبت از آن استفاده می کنند. این عنصر شانس باعث می شود وضعیت های ممکن در آینده برای بازیکن ها قابل پیش بینی نباشد.

- در درخت مقابل بازیکن Max می خواهد تصمیم بگیرد کدام یک از اعمال ممکن را انجام دهد.

- توجه: گره های شانس در سطح یک مربوط به استفاده بازیکن Min از عنصر شانس است.

بازی های حاوی عنصر شانس



- در بسیاری از بازی ها یک عنصر شانس مانند تاس یا سکه وجود دارد که بازیکن ها به نوبت از آن استفاده می کنند. این عنصر شانس باعث می شود وضعیت های ممکن در آینده برای بازیکن ها قابل پیش بینی نباشد.

- در درخت مقابل بازیکن Max می خواهد تصمیم بگیرد کدام یک از اعمال ممکن را انجام دهد.

- توجه: گره های شانس در سطح یک مربوط به استفاده بازیکن Min از عنصر شانس است.

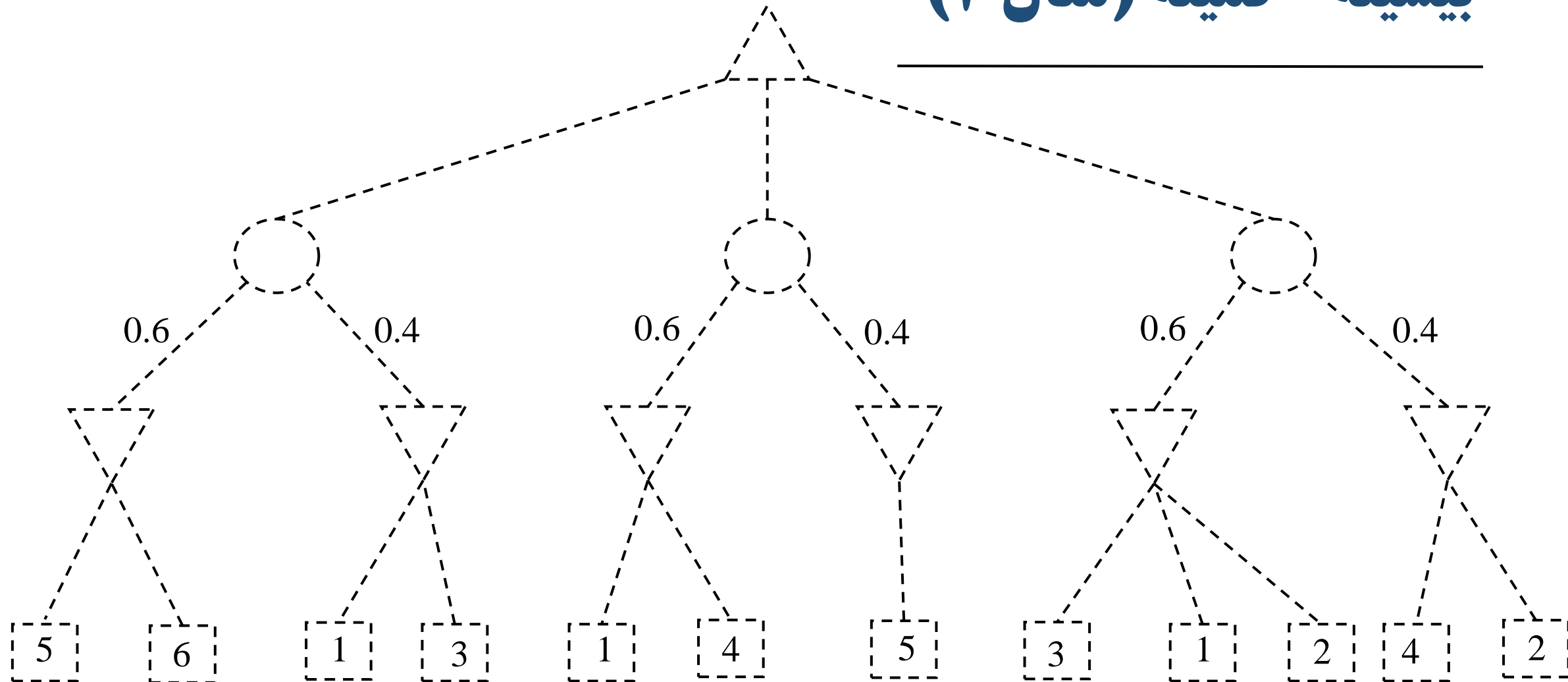
الگوریتم بیشینه کمینه برای بازی های حاوی عنصر شانس

- ما می خواهیم حرکتی را انتخاب کنیم که بهترین موقعیت را ایجاد کند اما MINMAX قطعی برای موقعیت ها نداریم. در عوض می توانیم مقدار مورد انتظار موقعیت را شناسایی کنیم که میانگین تمام نتایج ممکن مربوط به گره های شانس است.
- به این ترتیب می توان مقدار MIINMAX مربوط به بازی های قطعی را به یک مقدار MINMAX مورد انتظار برای بازی های با گره شانس تعمیم داد.

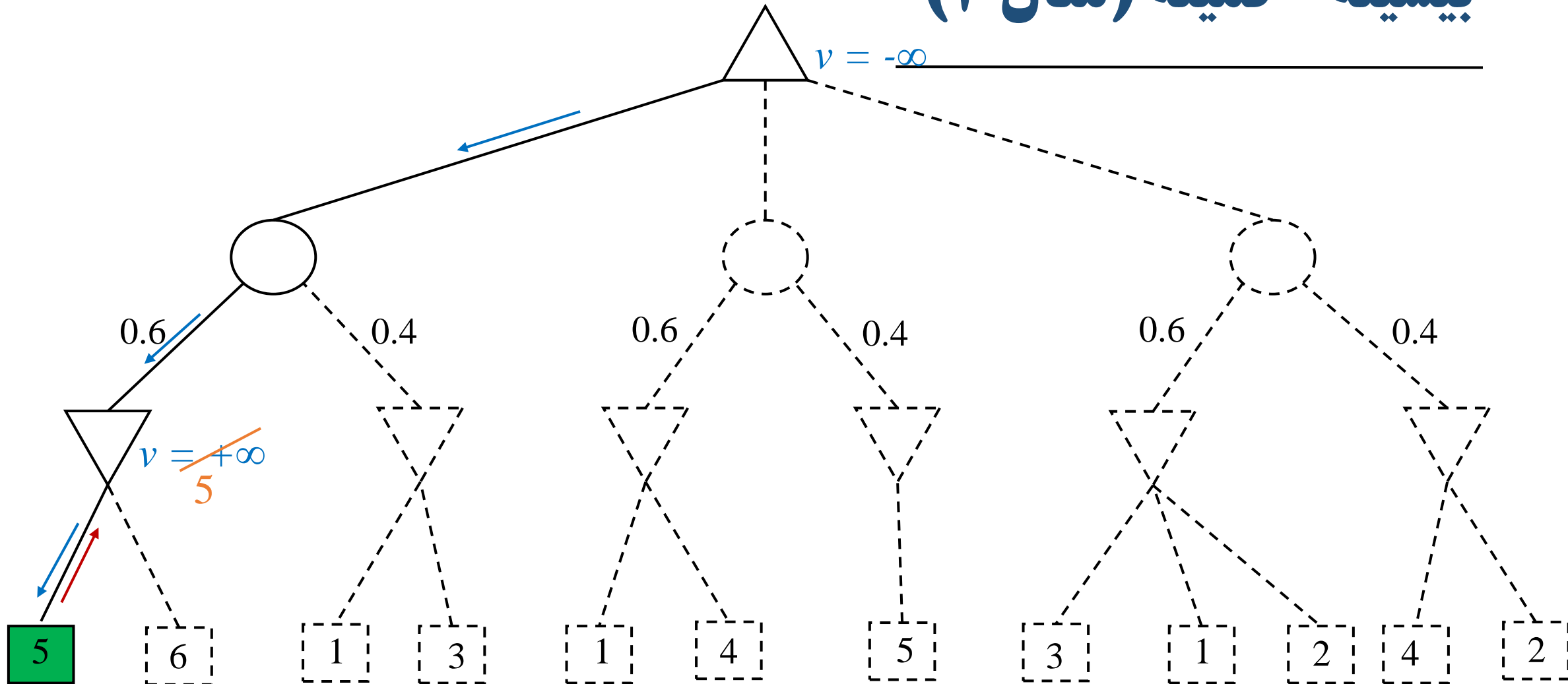
$$\text{EXPECTIMINIMAX}(s) =$$

$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \\ \sum_r P(r) \text{EXPECTIMINIMAX}(\text{RESULT}(s, r)) & \text{if } \text{PLAYER}(s) = \text{CHANCE} \end{cases}$$

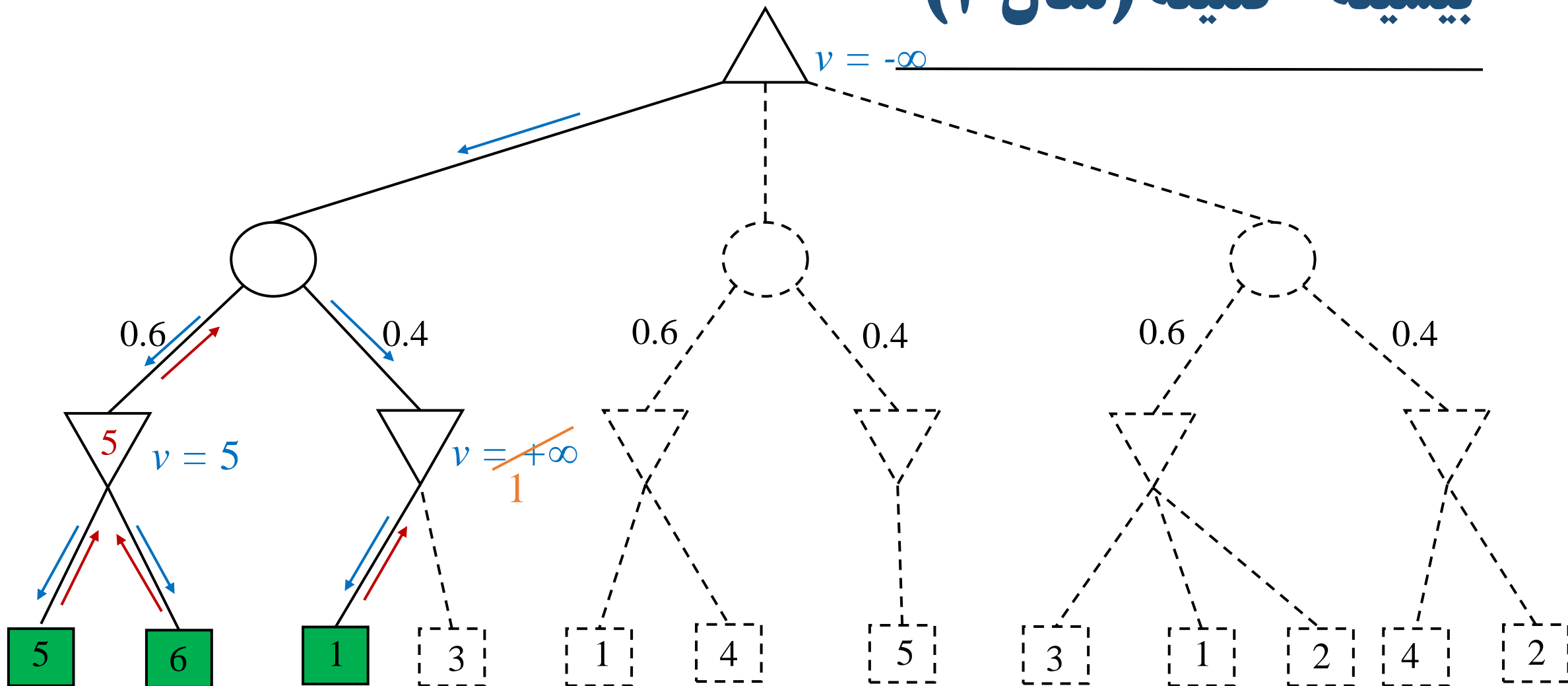
پیشینه – کمینه (مثال ۲)



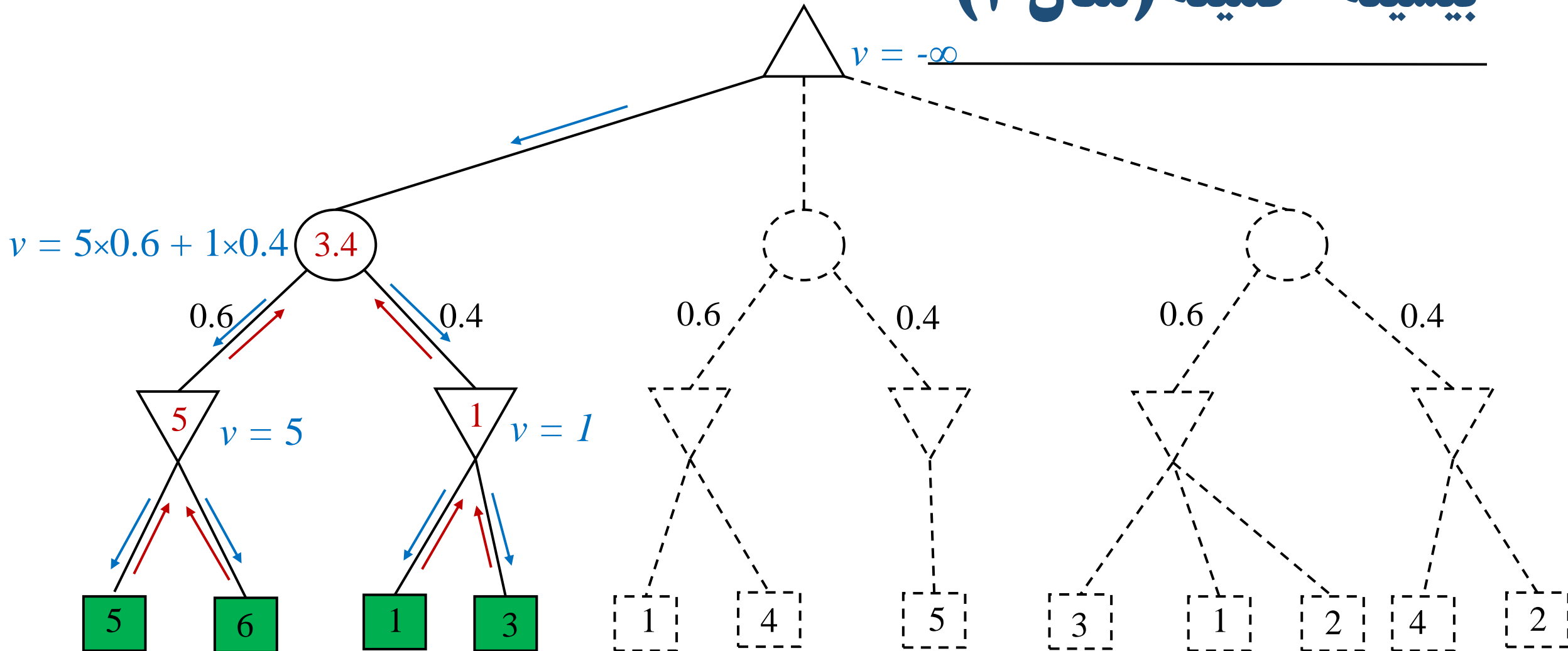
پیشینه-کمینه (مثال ۲)



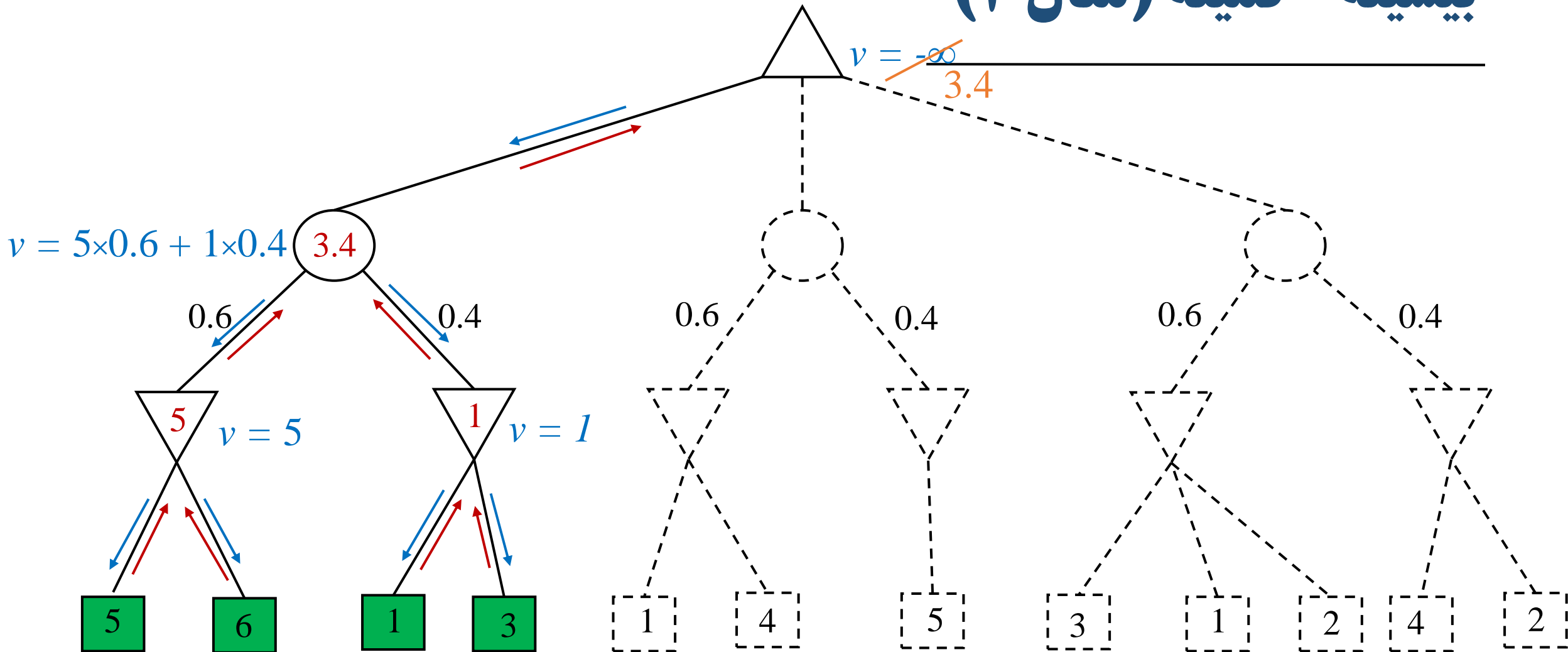
پیشینہ—کمینہ (مثال ۲)



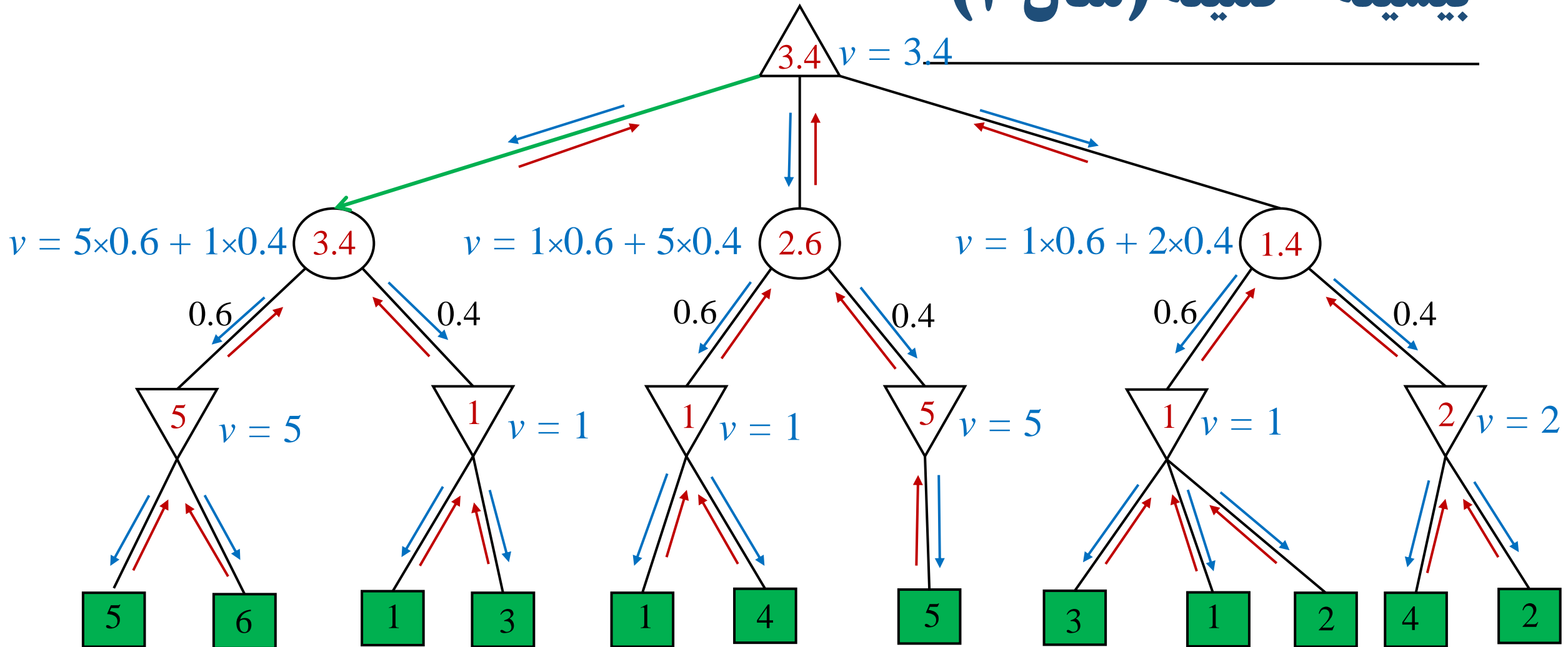
پیشینه-کمینه (مثال ۲)



پیشینہ—کمینہ (مثال ۲)



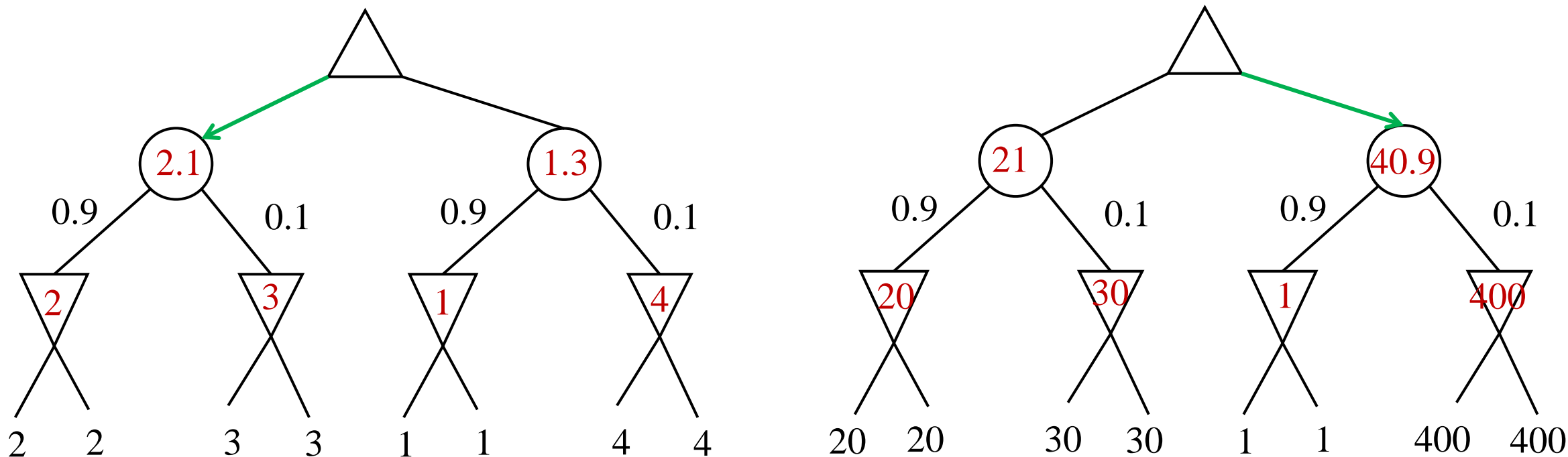
پیشینه - کمینه (مثال ۲)



الگوریتم بیشینه کمینه برای بازی های حاوی عنصر شانس

- همانند بازی های بدون عنصر شانس، اگر بررسی هر شاخه تا برگ ها زمان بر باشد، می توان درخت را تا عمق خاصی پیمایش کرد و از تابع ارزیابی برای ارزیابی تخمینی گره های غیر پایانه استفاده کرد.
- در بازی های فاقد عنصر شانس، لازم نبود تابع ارزیابی ارزش گره ها را نزدیک به مقدار واقعی سودمندی آن ها تخمین بزند، بلکه تنها کافی بود گره ها را به ترتیب ارجح بودنشان ارزش دهی و مرتب کند. اما در بازی های حاوی عنصر شانس، چنین شرطی کافی نیست.
- تابع ارزیابی در این بازی ها باید یک جابه جایی خطی مثبت از سودمندی واقعی گره ها باشد. یعنی مقدار X تبدیل به $ax+b$ ($a>0$) شود.
- مثال بعد نشان می دهد که تغییر با حفظ ترتیب در مقادیر برگ ها در بازی های حاوی عنصر شانس، بهترین حرکت را ممکن است عوض کند.

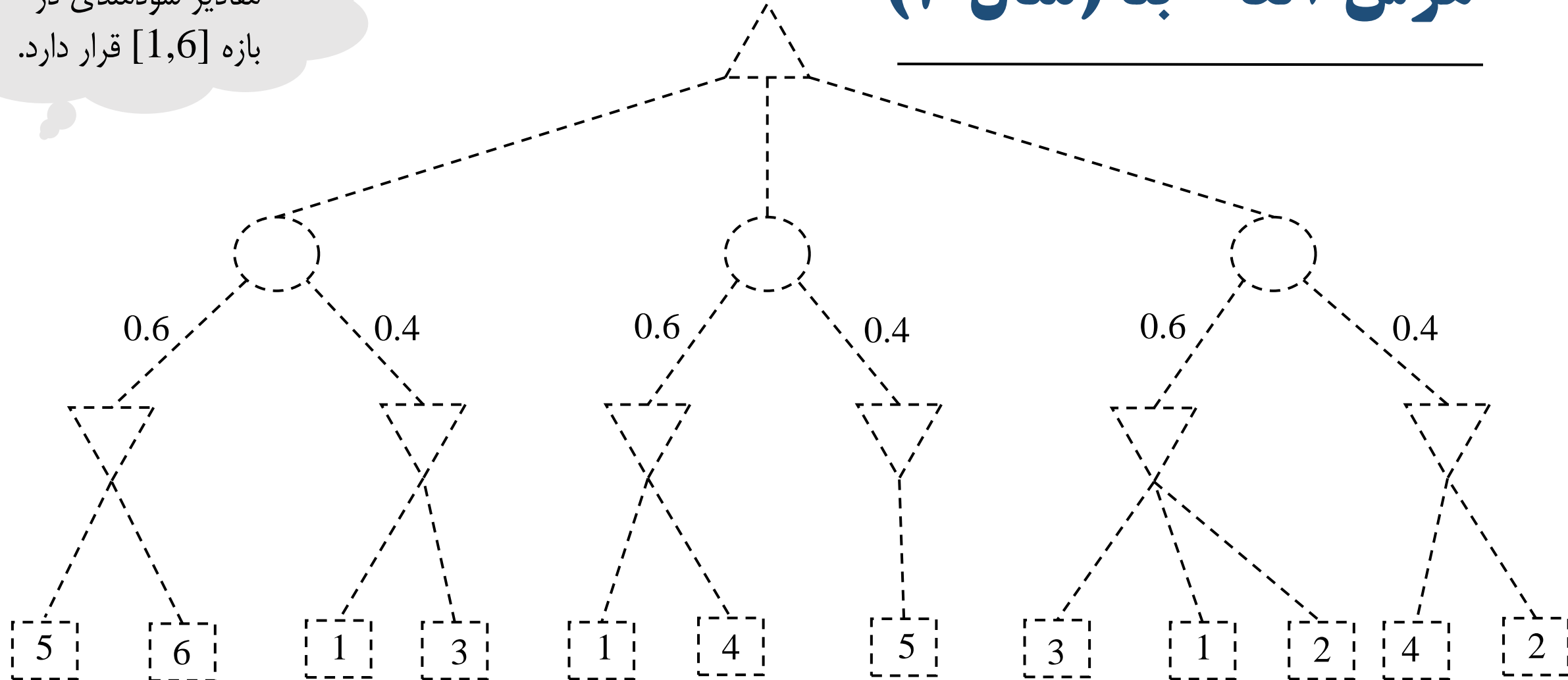
الگوریتم بیشینه کمینه برای بازی های حاوی عنصر شانس



- اگر عنصر شانس، n مقدار مختلف داشته باشد، پیچیدگی زمانی $O(b^m n^m)$ خواهد بود. در این صورت، حداکثر عمق قابل دسترسی (با توجه به محدودیت زمانی) کمتر از بازی های بدون عنصر شانس خواهد بود.

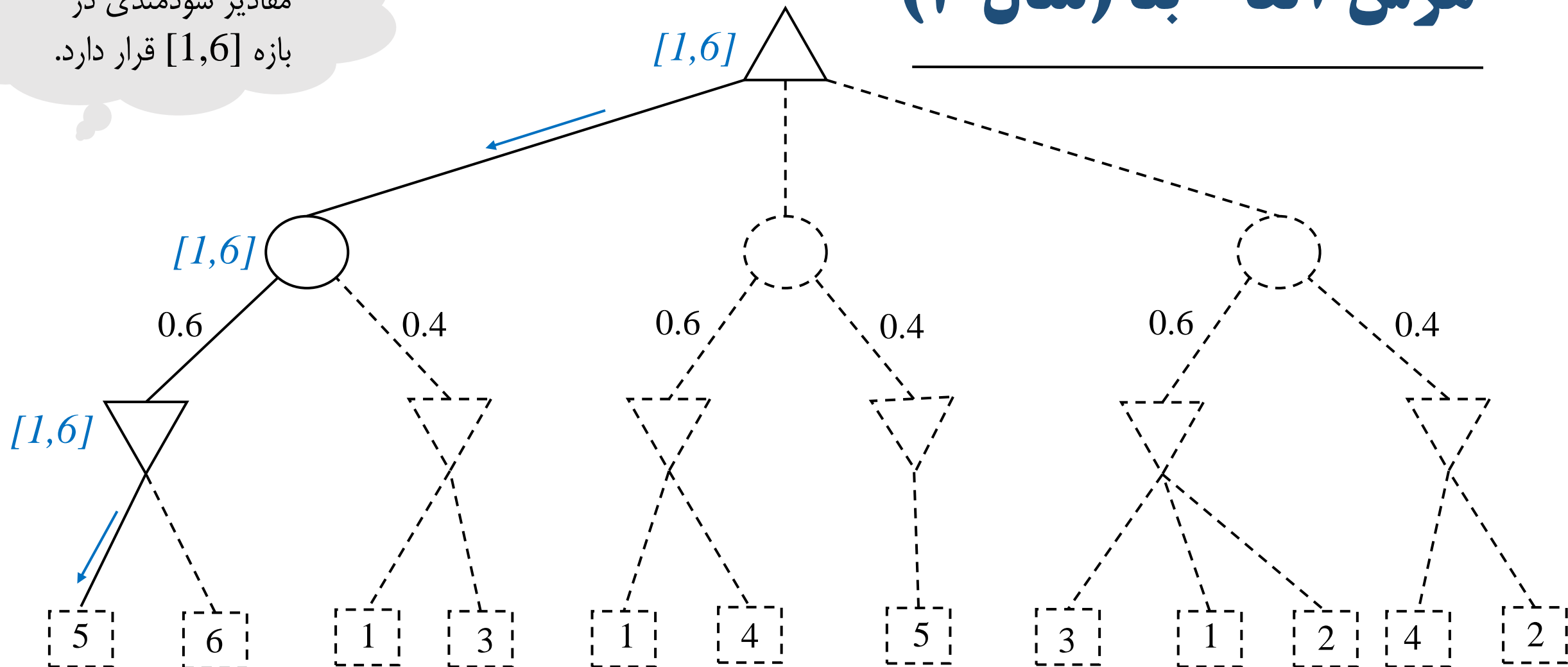
هرس آلفا-بتا (مثال ۳)

مقادیر سودمندی در
بازه $[1,6]$ قرار دارد.



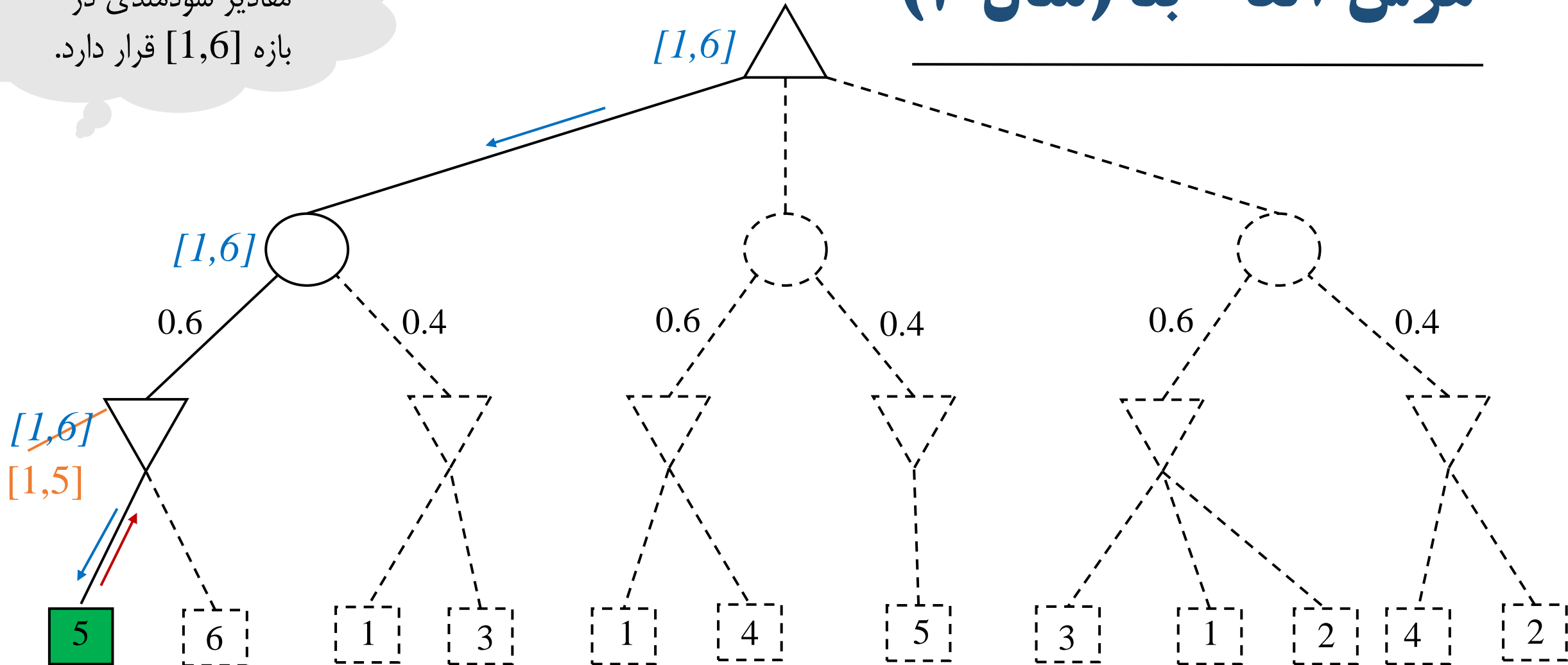
هرس آلفا-بتا (مثال ۳)

مقادیر سودمندی در بازه $[1,6]$ قرار دارد.



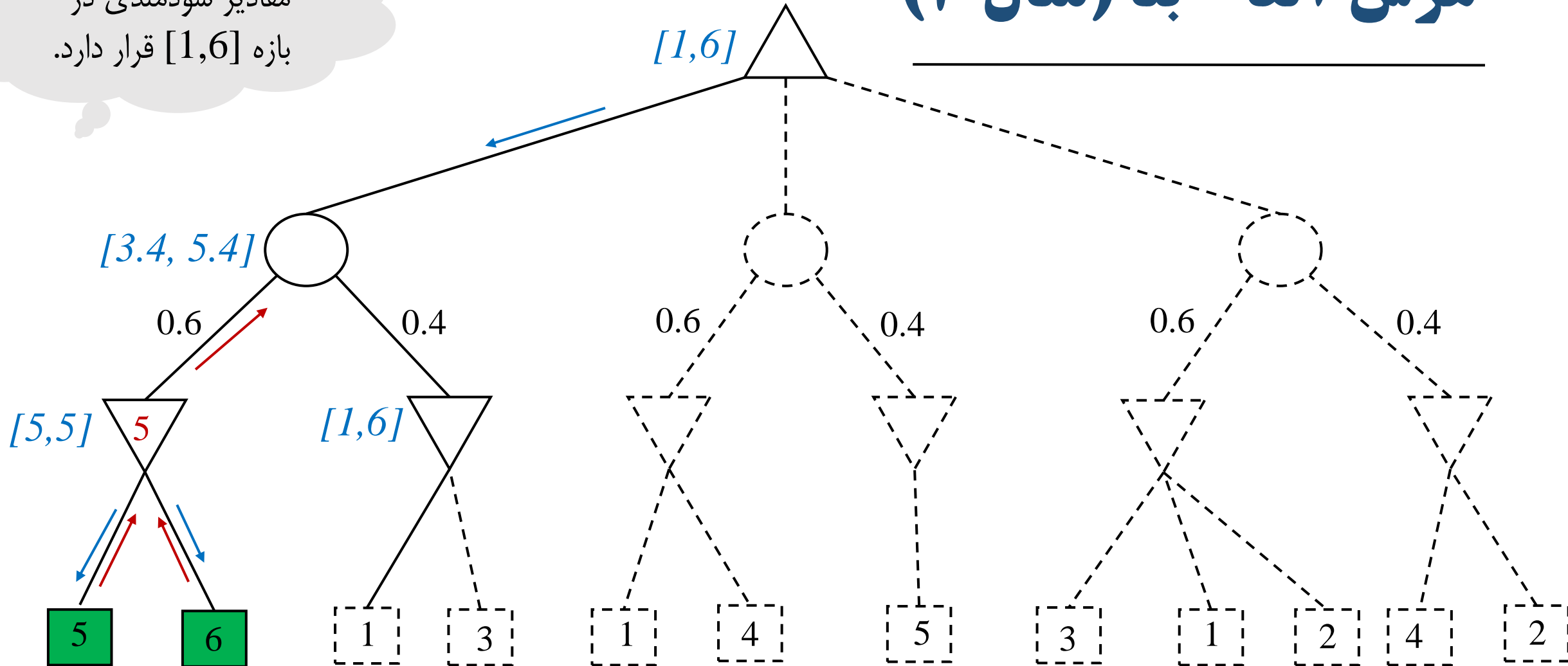
هرس آلفا-بتا (مثال ۳)

مقادیر سودمندی در
بازه $[1,6]$ قرار دارد.



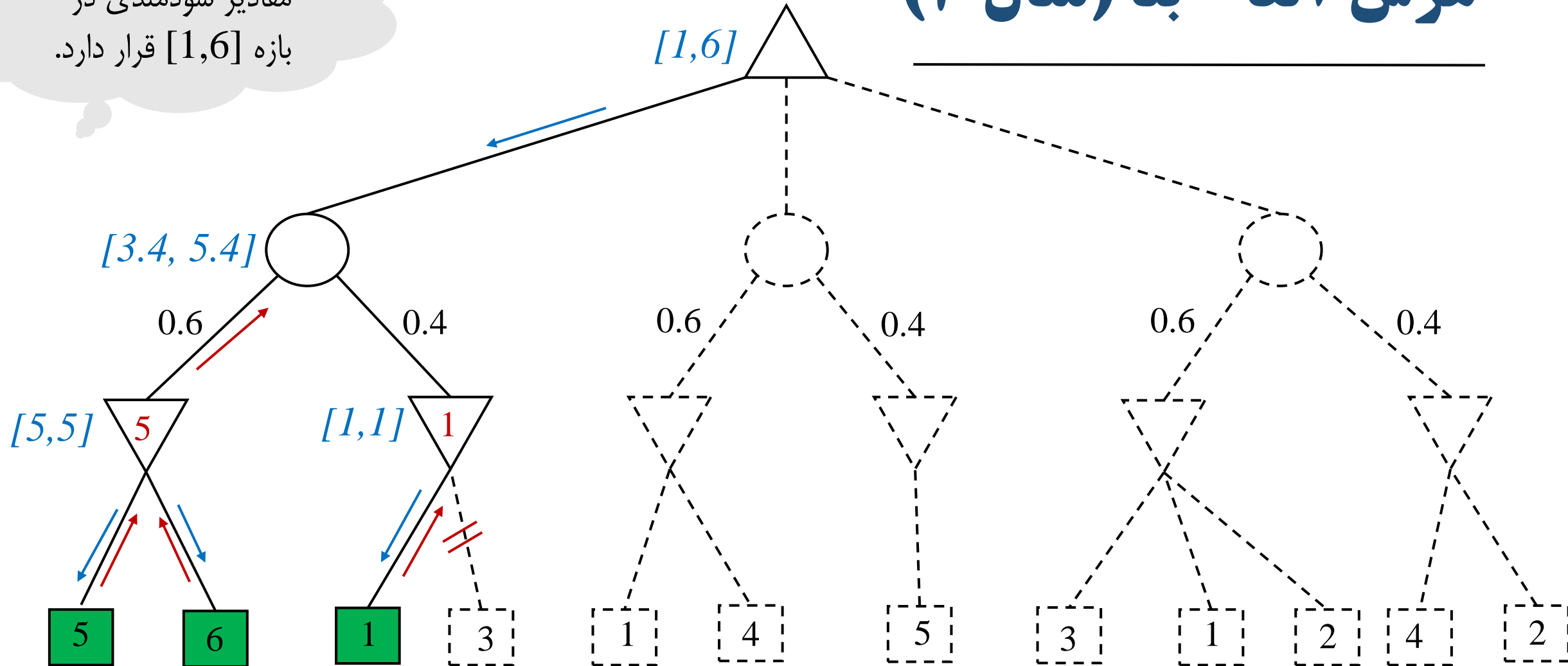
هرس آلفا-بتا (مثال ۳)

مقادیر سودمندی در
بازه $[1,6]$ قرار دارد.



هرس آلفا-بتا (مثال ۳)

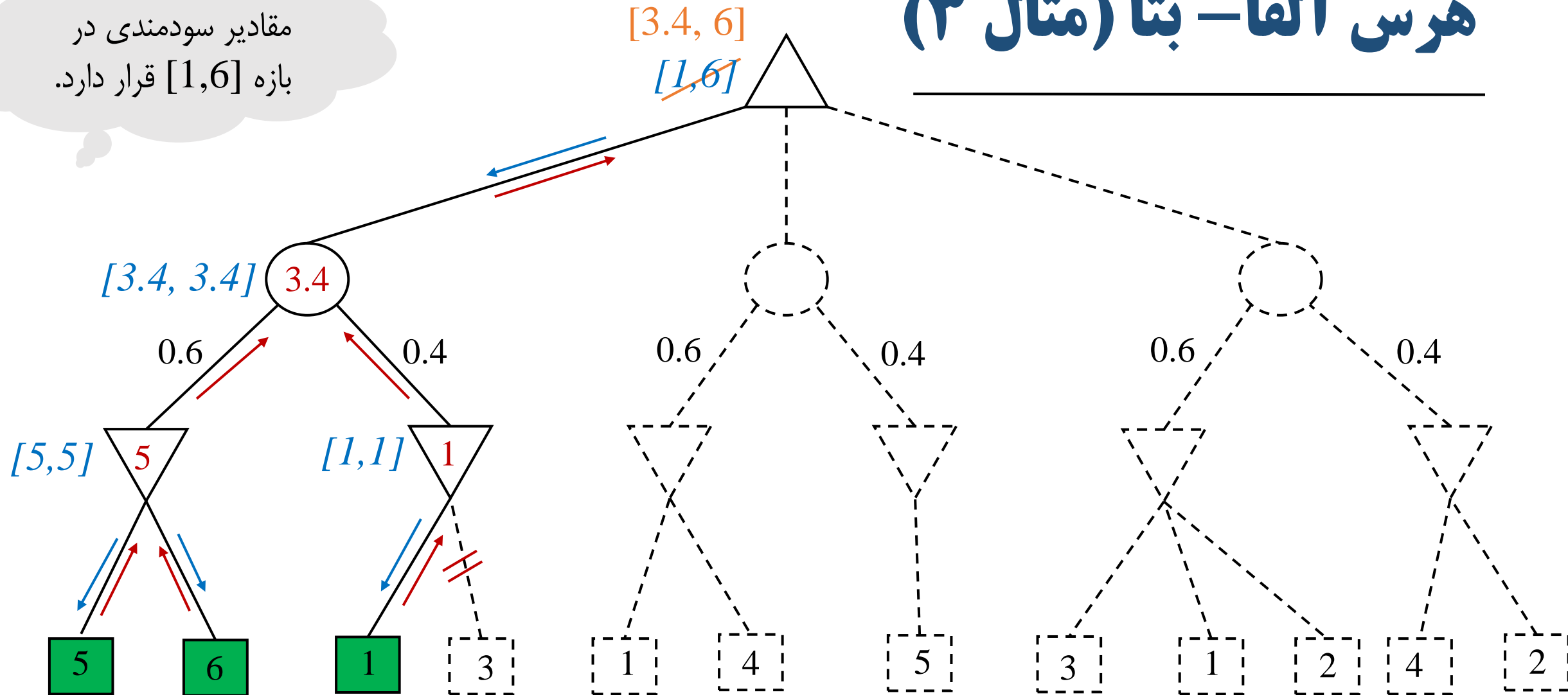
مقادیر سودمندی در
بازه $[1,6]$ قرار دارد.



هرس به علت بازه

هرس آلفا-بتا (مثال ۳)

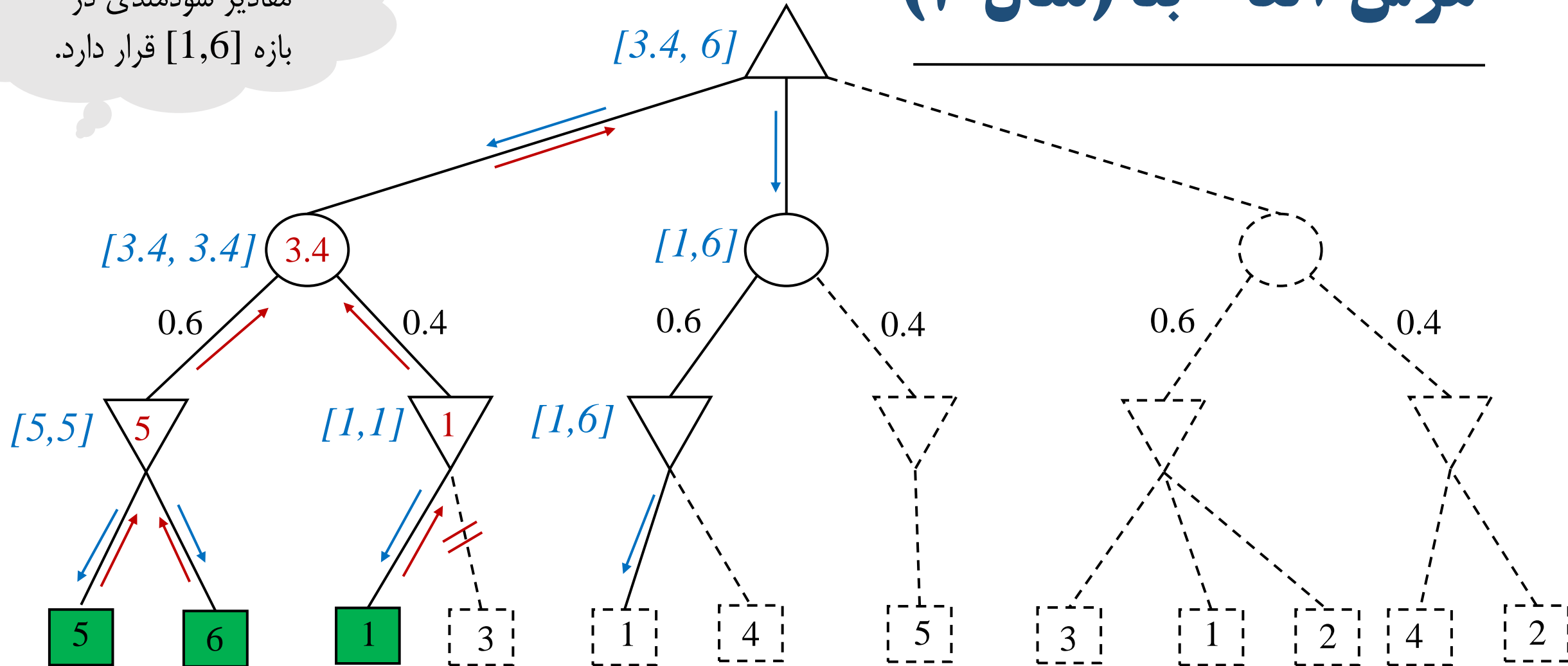
مقادیر سودمندی در
بازه $[1,6]$ قرار دارد.



هرس به علت بازه

هرس آلفا-بتا (مثال ۳)

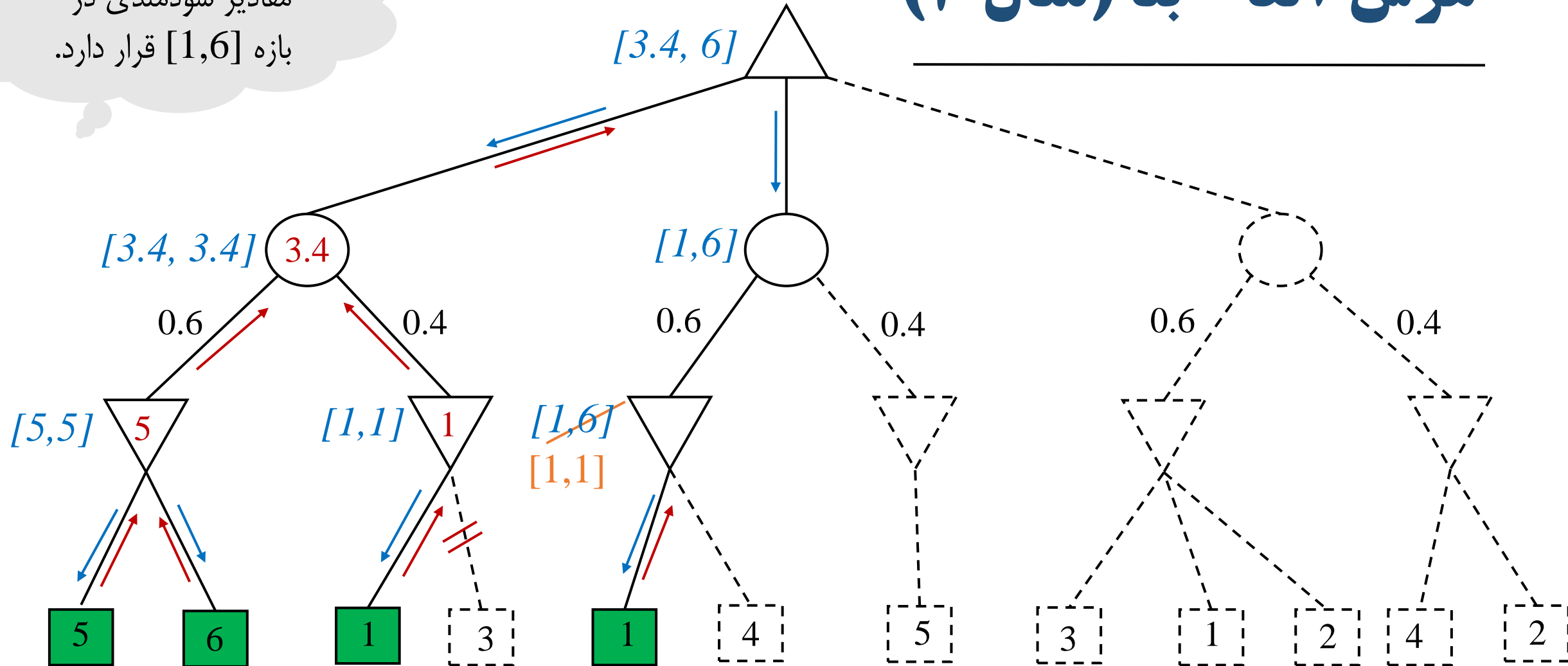
مقادیر سودمندی در بازه $[1,6]$ قرار دارد.



هرس به علت بازه

هرس آلفا-بتا (مثال ۳)

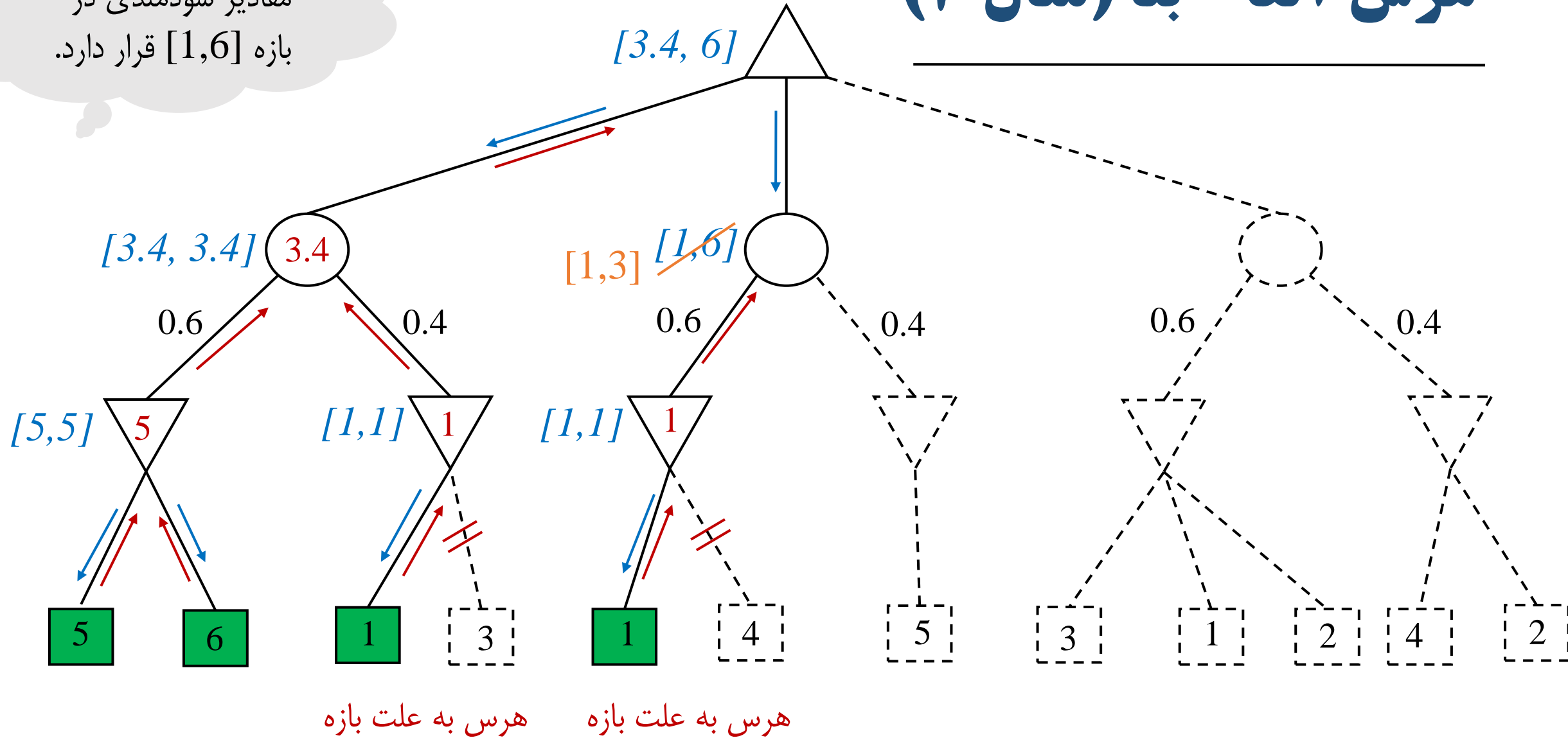
مقادیر سودمندی در
بازه $[1,6]$ قرار دارد.



هرس به علت بازه

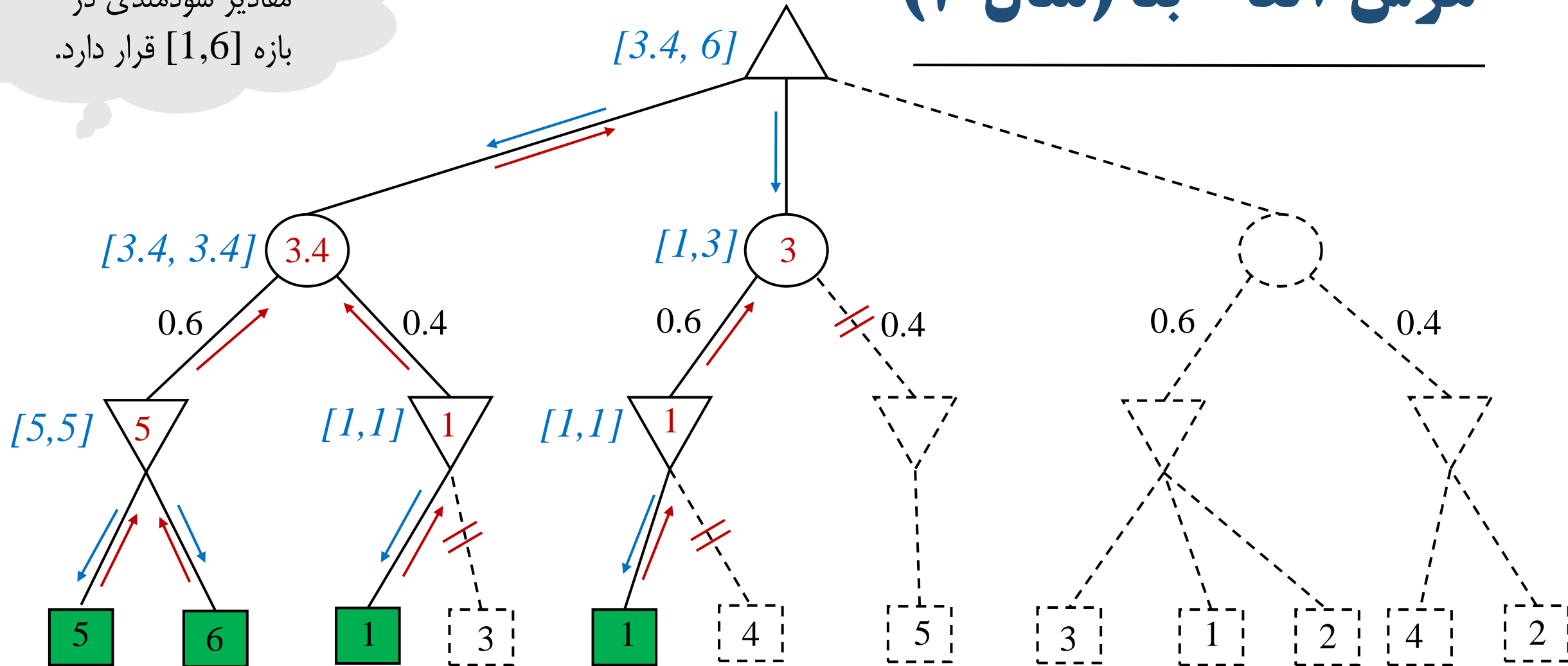
هرس آلفا-بتا (مثال ۳)

مقادیر سودمندی در
بازه $[1,6]$ قرار دارد.



هرس آلفا-بتا (مثال ۳)

مقادیر سودمندی در بازه $[1,6]$ قرار دارد.

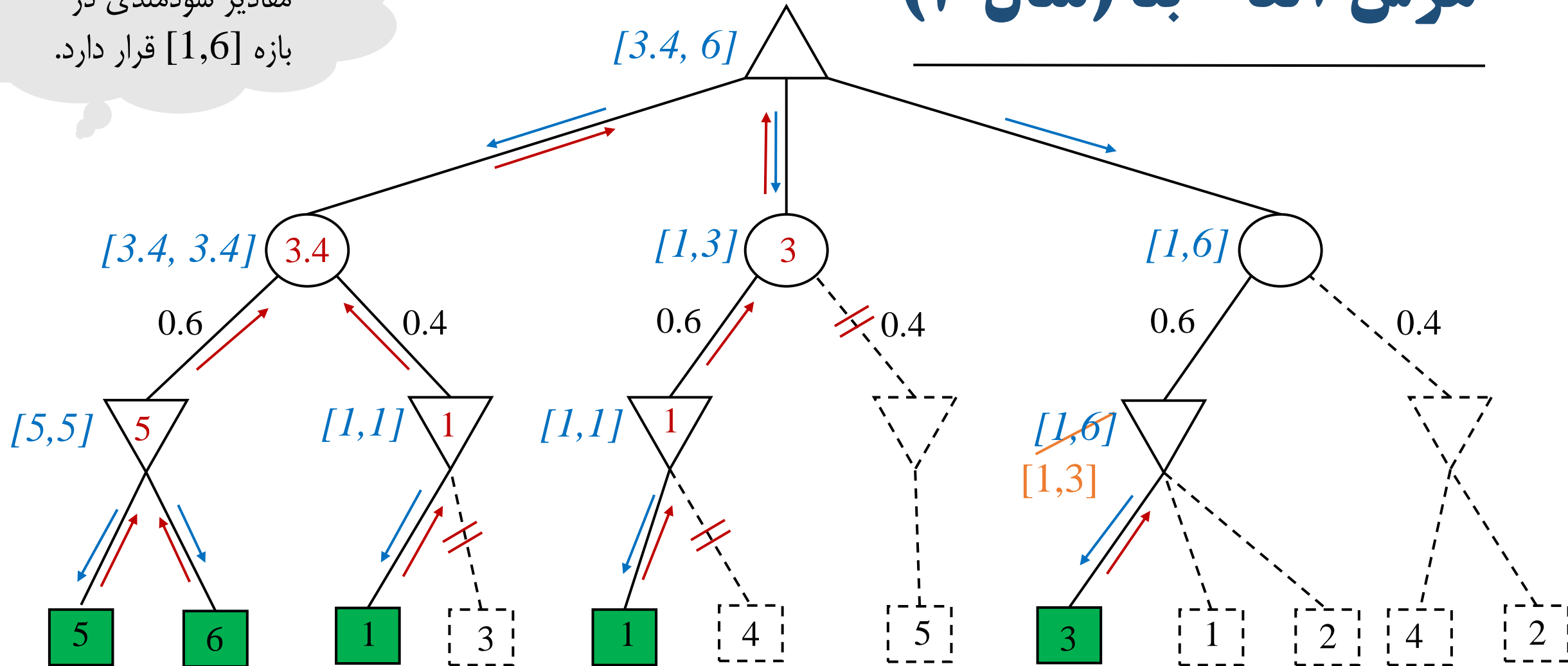


هرس شاخه شانس هرس به علت بازه هرس به علت بازه

اگر شاخه سمت راست حداکثر سودمندی ممکن (۶) را داشته باشد، مقدار مورد انتظار گره شانس از حداقل سودمندی گره max کمتر است.

هرس آلفا-بتا (مثال ۳)

مقادیر سودمندی در بازه $[1,6]$ قرار دارد.

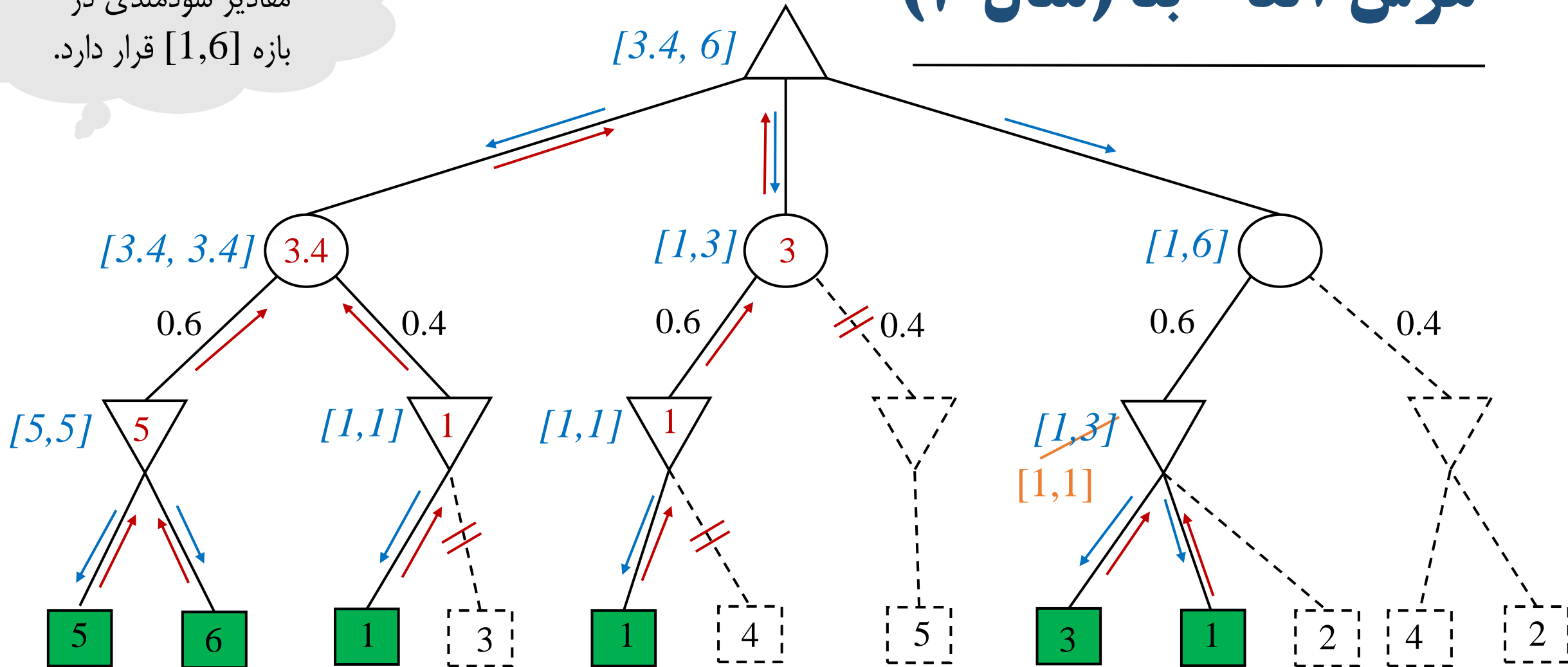


هرس شاخه شانس هرس به علت بازه هرس به علت بازه

اگر شاخه سمت راست حداکثر سودمندی ممکن (۶) را داشته باشد، مقدار مورد انتظار گره شانس از حداقل سودمندی گره max کمتر است.

هرس آلفا-بتا (مثال ۳)

مقادیر سودمندی در بازه $[1,6]$ قرار دارد.

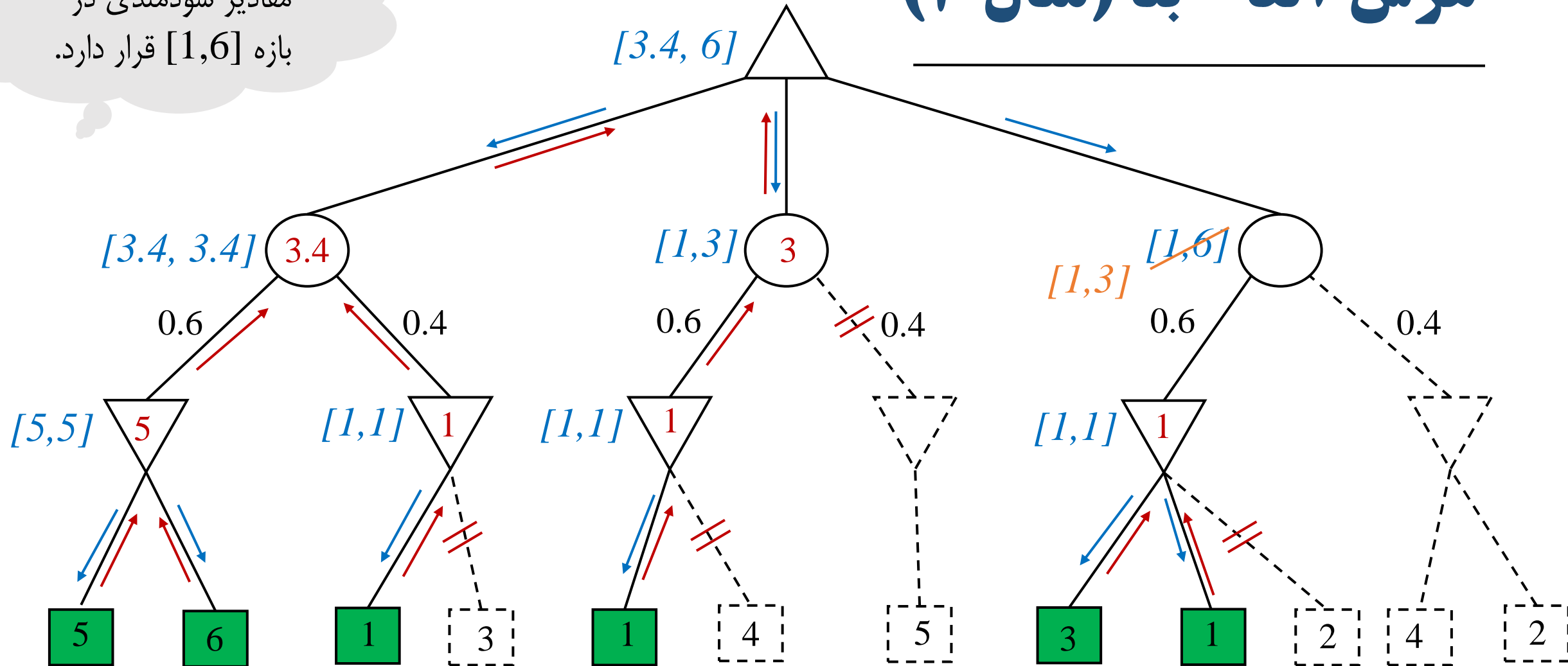


هرس شاخه شانس هرس به علت بازه هرس به علت بازه

اگر شاخه سمت راست حداکثر سودمندی ممکن (۶) را داشته باشد، مقدار مورد انتظار گره شانس از حداقل سودمندی گره max کمتر است.

هرس آلفا-بتا (مثال ۳)

مقادیر سودمندی در بازه $[1,6]$ قرار دارد.



هرس به علت بازه

هرس به علت بازه

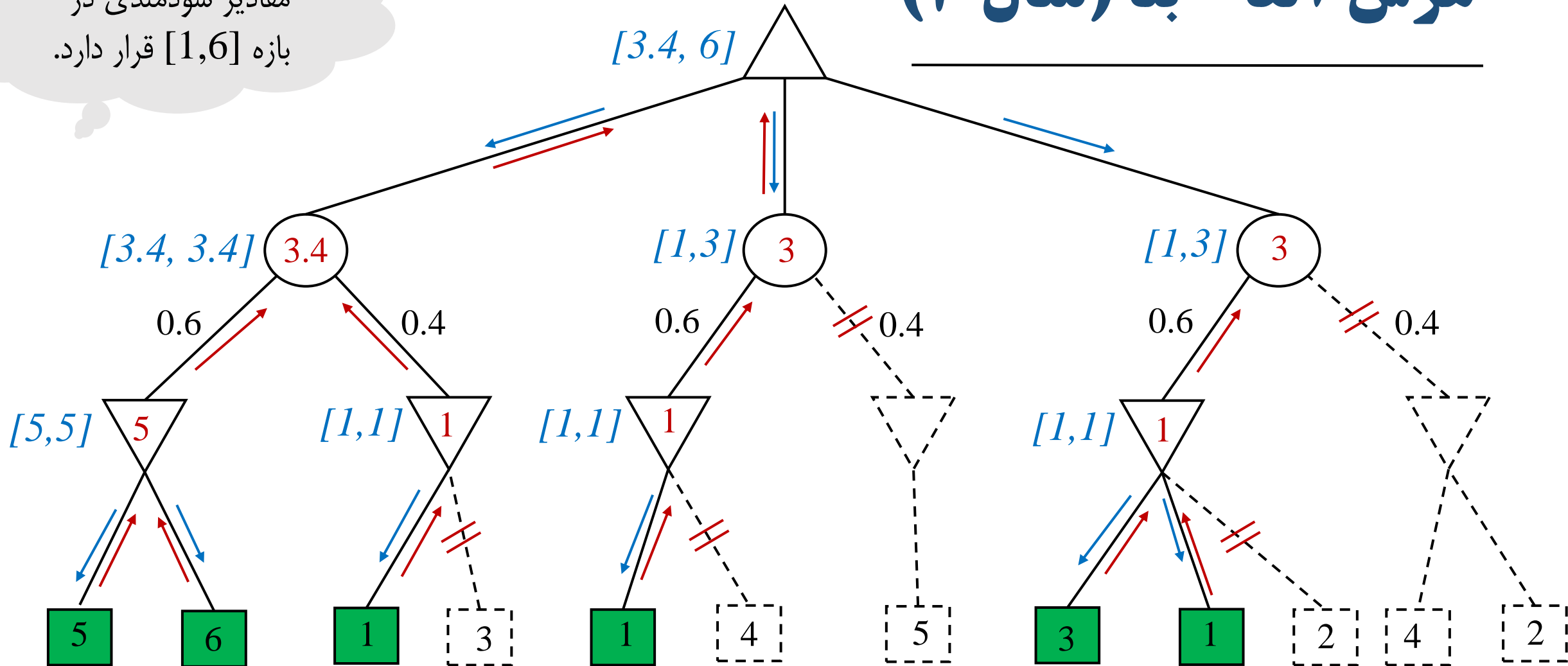
هرس شاخه شانس

هرس به علت بازه

اگر شاخه سمت راست حداکثر سودمندی ممکن (۶) را داشته باشد، مقدار مورد انتظار گره شانس از حداقل سودمندی گره max کمتر است.

هرس آلفا-بتا (مثال ۳)

مقادیر سودمندی در
بازه $[1,6]$ قرار دارد.



هرس به علت بازه

هرس به علت بازه

هرس شاخه شانس

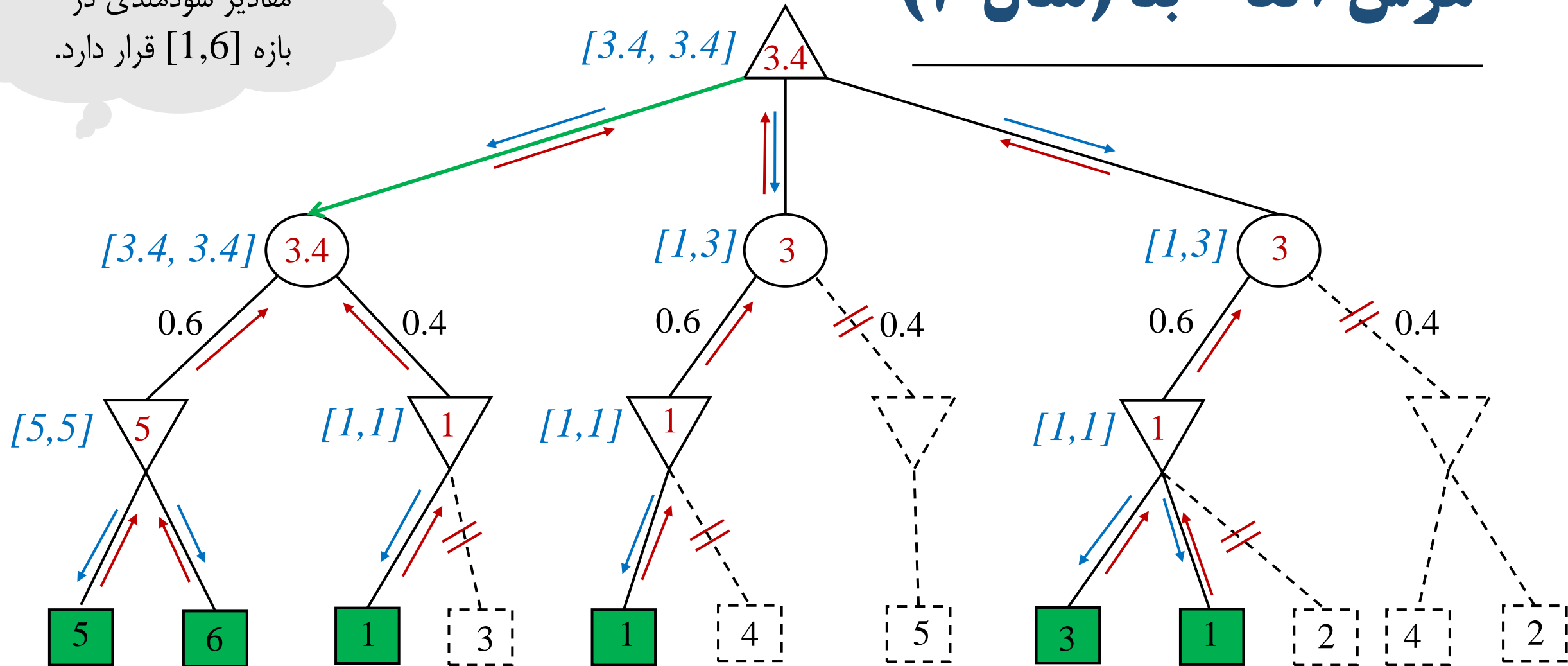
هرس به علت بازه

هرس شاخه شانس

اگر شاخه سمت راست حداکثر سودمندی ممکن (۶) را داشته باشد، مقدار مورد انتظار گره شانس از حداقل سودمندی گره max کمتر است.

هرس آلفا-بتا (مثال ۳)

مقادیر سودمندی در بازه $[1,6]$ قرار دارد.



هرس به علت بازه

هرس به علت بازه

هرس شاخه شانس

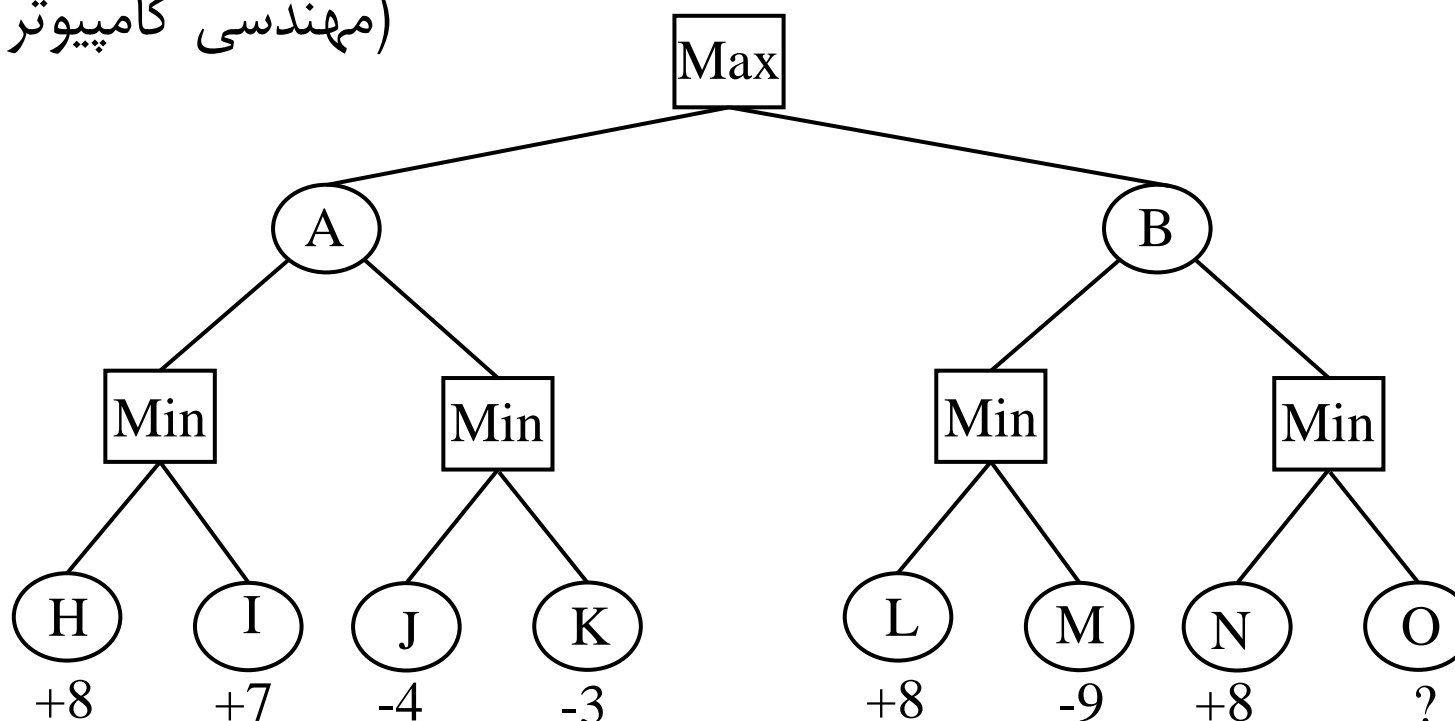
هرس به علت بازه

هرس شاخه شانس

اگر شاخه سمت راست حداکثر سودمندی ممکن (۶) را داشته باشد، مقدار مورد انتظار گره شانس از حداقل سودمندی گره max کمتر است.

در درخت زیر، دو گره A و B از نوع گره شانس و معادل عمل تصادفی انداختن یک سکه هستند. در صورتی که بدانیم مقدار کمینه و بیشینه تابع ارزیابی گره‌های برگ، به ترتیب برابر با -10 و $+10$ است. در روش هرس آلفا-بتا کدام یک از گره‌های این درخت هرس خواهند شد؟

(مهندسی کامپیوتر دولتی ۹۳)



- O (۱)
- N, O (۲) ✓
- M, N, O (۳)
- K, N, O (۴)