دانشگاه علم و صنعت
دانشکده مهندسی کامپیوتر

جستجوی رقابتی

«هوش مصنوعی: رهیافتی نوین»، فصل ۵

مدرس: آرش عبدی هجراندوست

نیم‌سال دوم ۱۴۰۱-۱۴۰۲

رئوس مطالب

- محیط‌های چندعاملی
  - بازی‌ها و مسائل جستجو
- معرفی دو الگوریتم معروف بازی‌ها
  - الگوریتم بیشینه کمینه
  - الگوریتم آلفا-بتا

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 رقابتی (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.

بازی

- 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
    - کاملا مشاهده‌پذیر

بازی به‌عنوان نوعی از مسأله جستجو

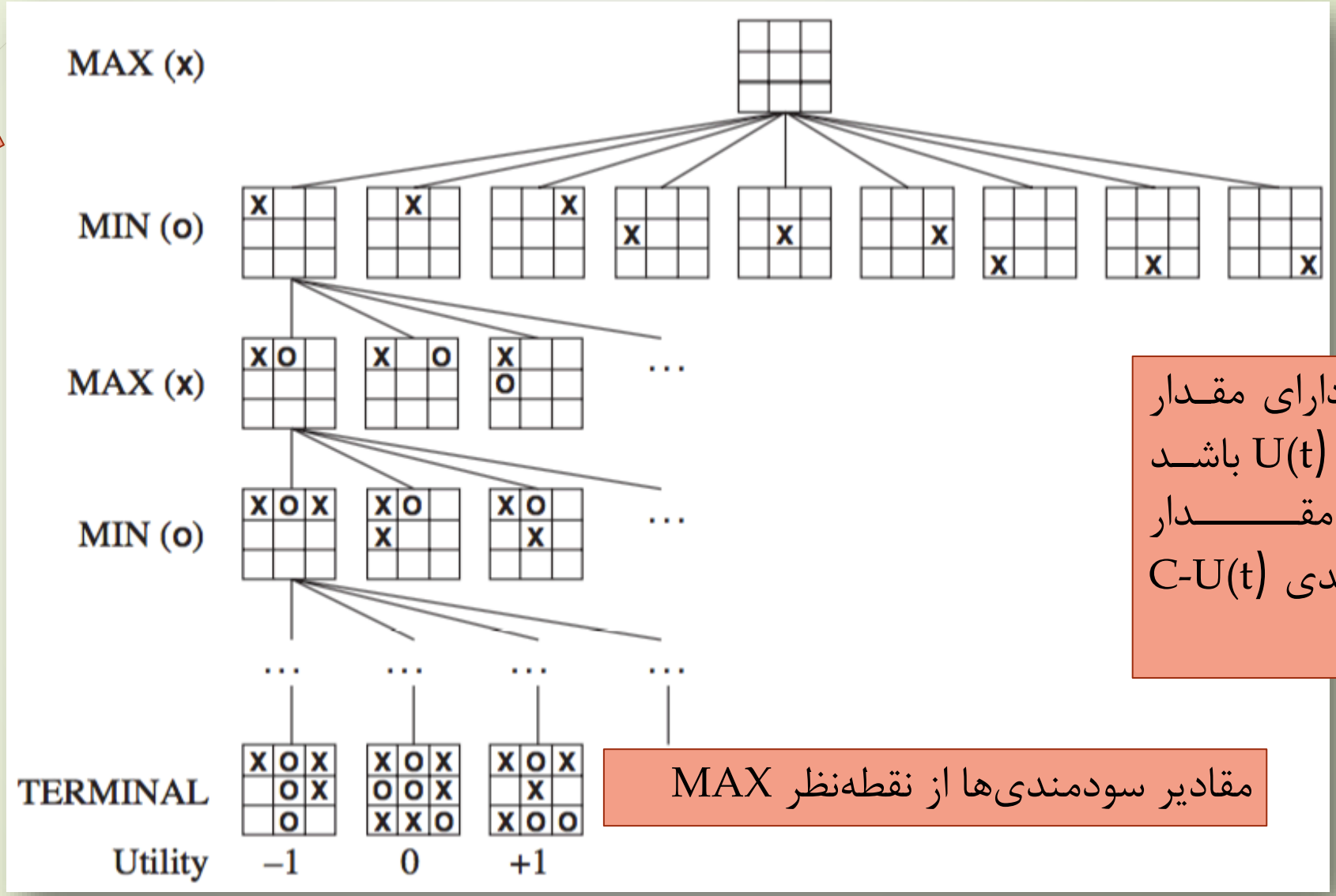
- حالت اولیه S_0: موقعیت اولیه بازی را مشخص می‌کند.
- Player(s): در وضعیت s نوبت کدام بازیکن است.
- Actions(s): مجموعه اعمال قانونی در وضعیت s را برمی‌گرداند.
- Result(s,a): مدل انتقال، نتیجه یک حرکت را تعریف می‌کند.
- Terminal-Test(s): آزمون پایانی هنگامی که بازی تمام شده باشد درست و در غیر این‌صورت غلط برمی‌گرداند.
- Utlity(s,p): مقدار سودمندی بازیکن p در حالت پایانی S چقدر است.
  - بازی مجموع صفر (مجموع ثابت): مجموع مقدار سومندی تمام بازیکنان در حالت پایانی S برابر با صفر یا یک مقدار ثابت است.

بازی به‌عنوان 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
- 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 به‌دست آورد.

درخت بازی (دوز tic-tac-toe)

دو بازیکن MAX و MIN که الان MAX به دنبال بهترین حرکت خود است



اگر MAX دارای مقدار سودمندی U(t) باشد Min دارای مقدار سودمندی C-U(t) است.

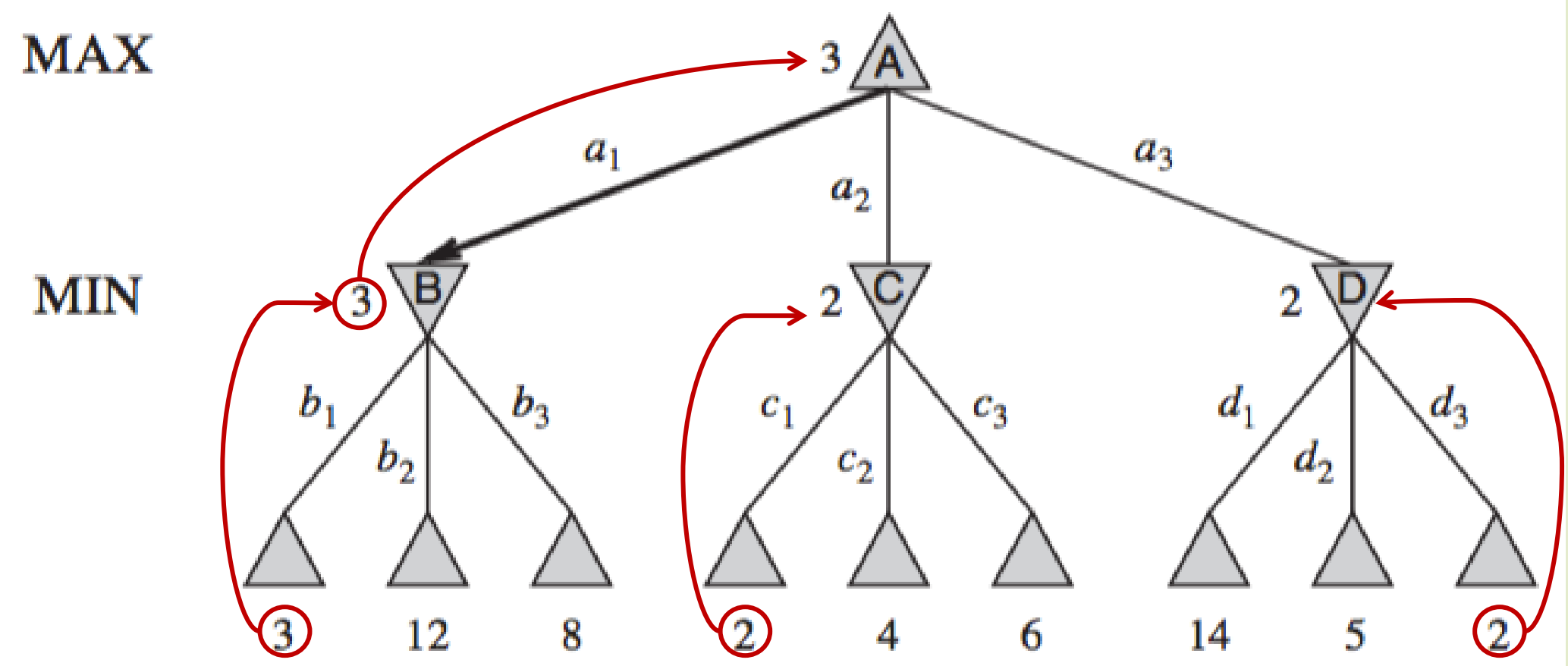
مقادیر سودمندی‌ها از نقطه‌نظر MAX

استراتژی کمینه‌بیشینه (MINIMAX)

- با داشتن درخت بازی، استراتژی بهینه را می‌توان با در نظر گرفتن مقدار minimax گره‌ها تعیین نمود.
  - تابع MINIMAX(s) بهترین نتیجه‌ی بودن در وضعیت s را مشخص می‌کند. با فرض آن‌که هر دو بازیکن از گره شروع تا پایان بهینه بازی کنند.
  - MAX به‌دنبال بیشینه کردن مقدار سودمندی خود و MIN به‌دنبال کمینه کردن مقدار سودمندی حریف است.

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

کمینه‌بیشینه (مثال ۱)



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

الگوریتم کمینه‌بیشینه

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

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

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

در روند اجرای الگوریتم هر جا نود MAX را دیدیم علامت ∞- و هرجا نود MIN را دیدیم علامت ∞+ قرار می‌دهیم.

ویژگی‌های الگوریتم بیشینه‌کمینه

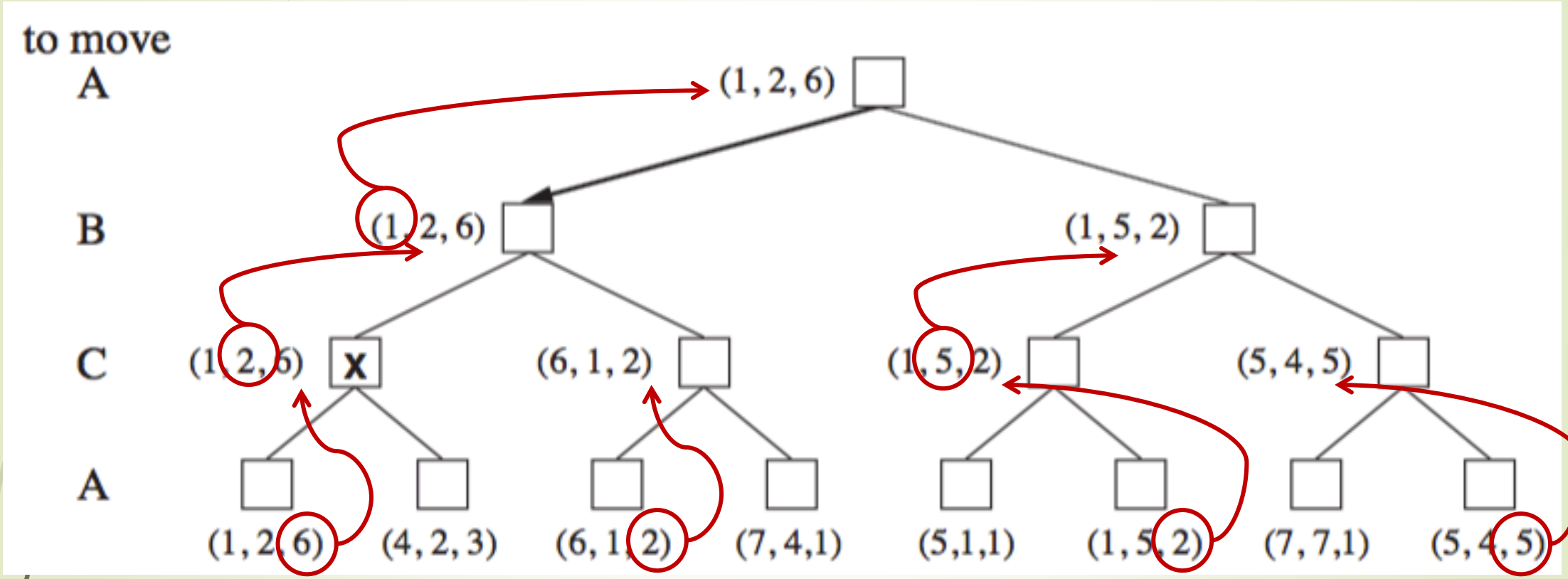
به‌صورت عمقی پیاده‌سازی شده است.

- کامل بودن؟ بله
  - هنگامی‌که 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 به‌طور بهینه بازی نکند چه پیش خواهد آمد؟
- پیچیدگی زمانی؟ نمایی $O(b^m)$
  - راه حل دقیق برای بازی‌های واقعی کاملا غیر قابل دسترس است.
  - برای مثال در شطرنج $b \approx 35, m \approx 100$
- پیچیدگی فضایی؟ خطی $O(bm)$

بسط ایده‌ی بیشینه‌کمینه به بازی‌های چند نفره

- مقادیر اختصاص داده شده به هر گره را با یک بردار سودمندی به اندازه تعداد بازیکنان جایگزین می‌کنیم.
  - اگر سه بازیکن A، B و C داشته باشیم بردار سودمندی برابر با $<v_A, v_B, v_C>$ 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 <۱۰۰، ۱۰۰> اتفاق می‌افتد بازیکن‌ها برای رسیدن به این وضعیت به‌طور خودکار همکاری می‌کنند.

بازی‌های چند نفره – مثال



هرس آلفا–بتا

- مشکل الگوریتم بیشینه‌کمینه این است که تمام گره‌های درخت را بررسی می‌کند.
  - تعداد حرکات نمایی
  - 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 بیشینه‌کمینه استاندارد اعمال می‌شود، همان جواب الگوریتم بیشینه‌کمینه را برمی‌گرداند با این تفاوت که در این روش، شاخه‌هایی که در تصمیم‌گیری نهایی تأثیری ندارند، هرس می‌شود.

حقایق هرس آلفا–بتا

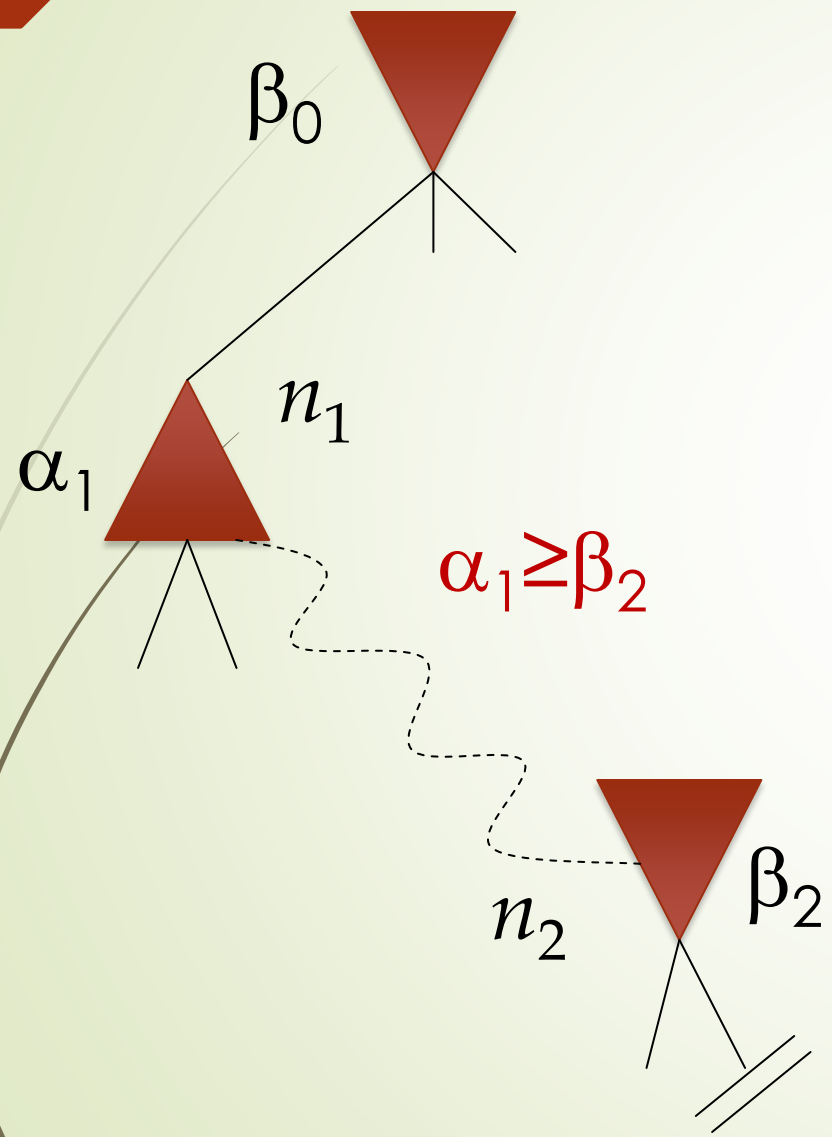
آلفا: کف یا مینیمم بازه سود ممکن

بتا: سقف یا ماکزیمم بازه سود ممکن

دو حقیقت زیر را می‌توان 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 = $+\infty$

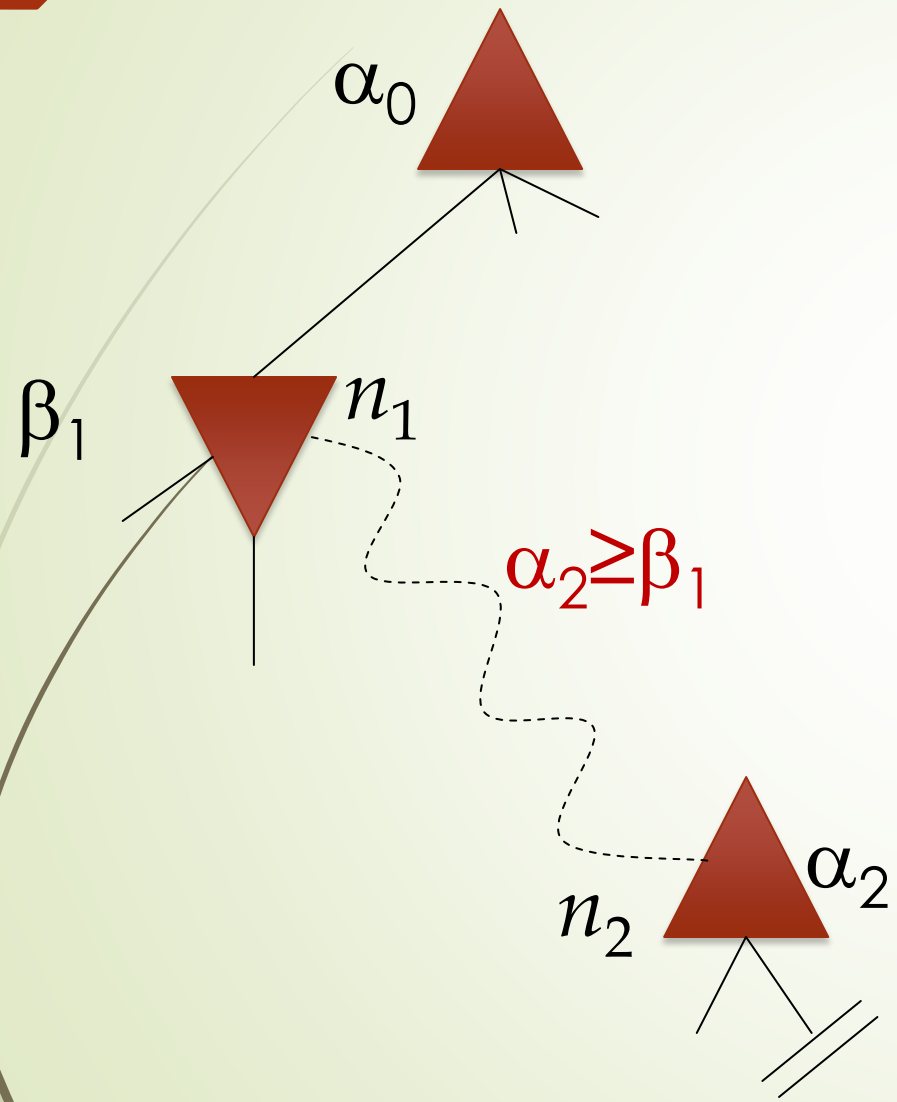
قانون اول– هرس آلفا



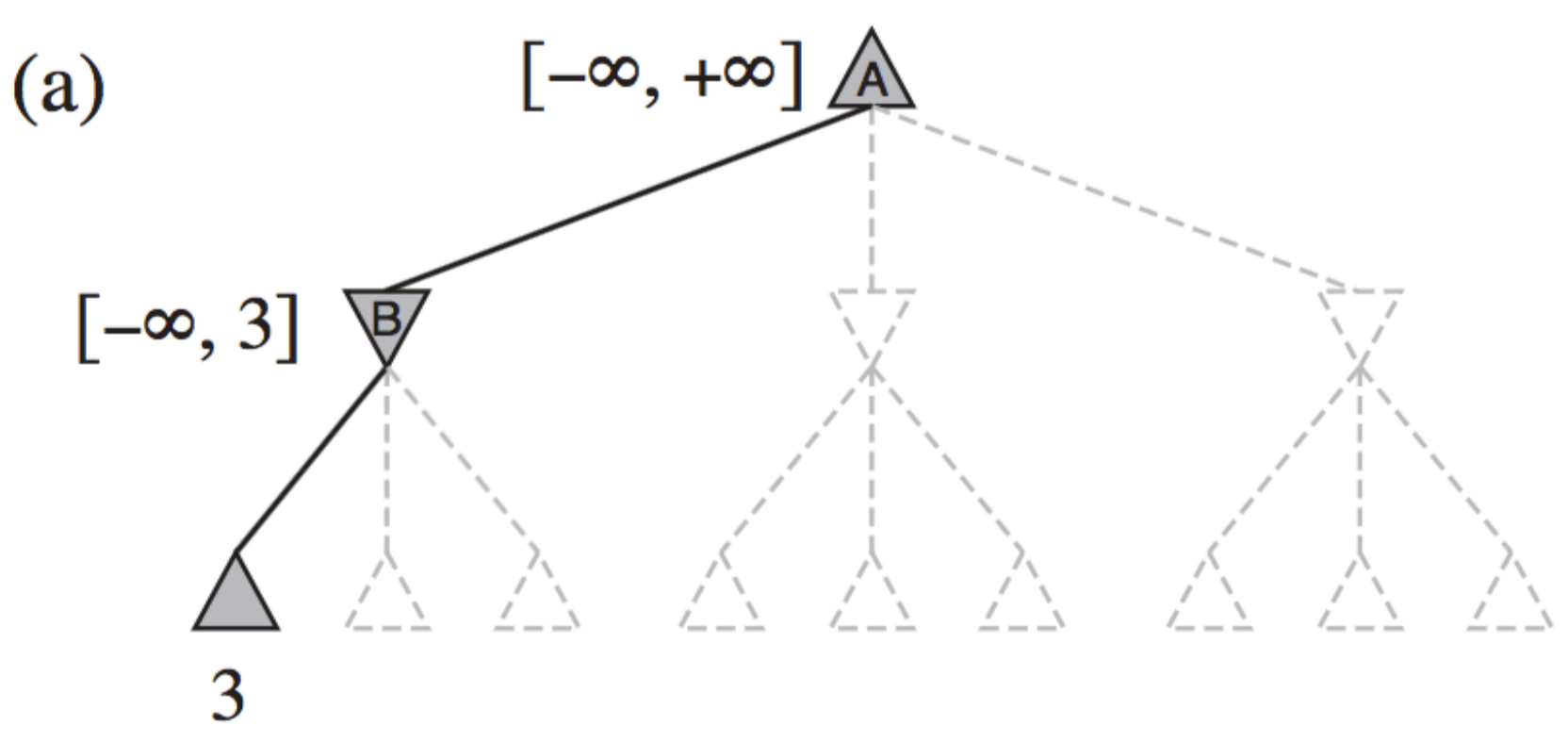
- عمل جستجو تحت گره MIN که مقدار بتای آن کوچک‌تر یا مساوی مقدار آلفای هر گره MAX اجداد آن گره است، قطع می‌شود.
- در این حالت، مقداری که تا به‌حال به گره‌ی MIN نسبت داده شده به‌عنوان مقدار نهایی بتای آن گره انتخاب خواهد شد.
- بررسی دو حالت:
  - اگر پدر مستقیم گره MIN دارای مقدار $\alpha_1 \geq \beta_2$ باشد
  - اگر یکی از پدربزرگ‌های گره MIN دارای مقدار $\alpha_1 \geq \beta_2$ باشد
    - در هر حالت ادامه جستجو بی فایده است. جد بزرگ، توجهی به نتایج این گره MIN نخواهد داشت، حتی اگر در خوش شانسانه ترین(!) حالت، مقدار این گره باعث بروزرسانی تمام گره های والد تا رسیدن به جد بزرگ شود.

قانون دوم– هرس بتا

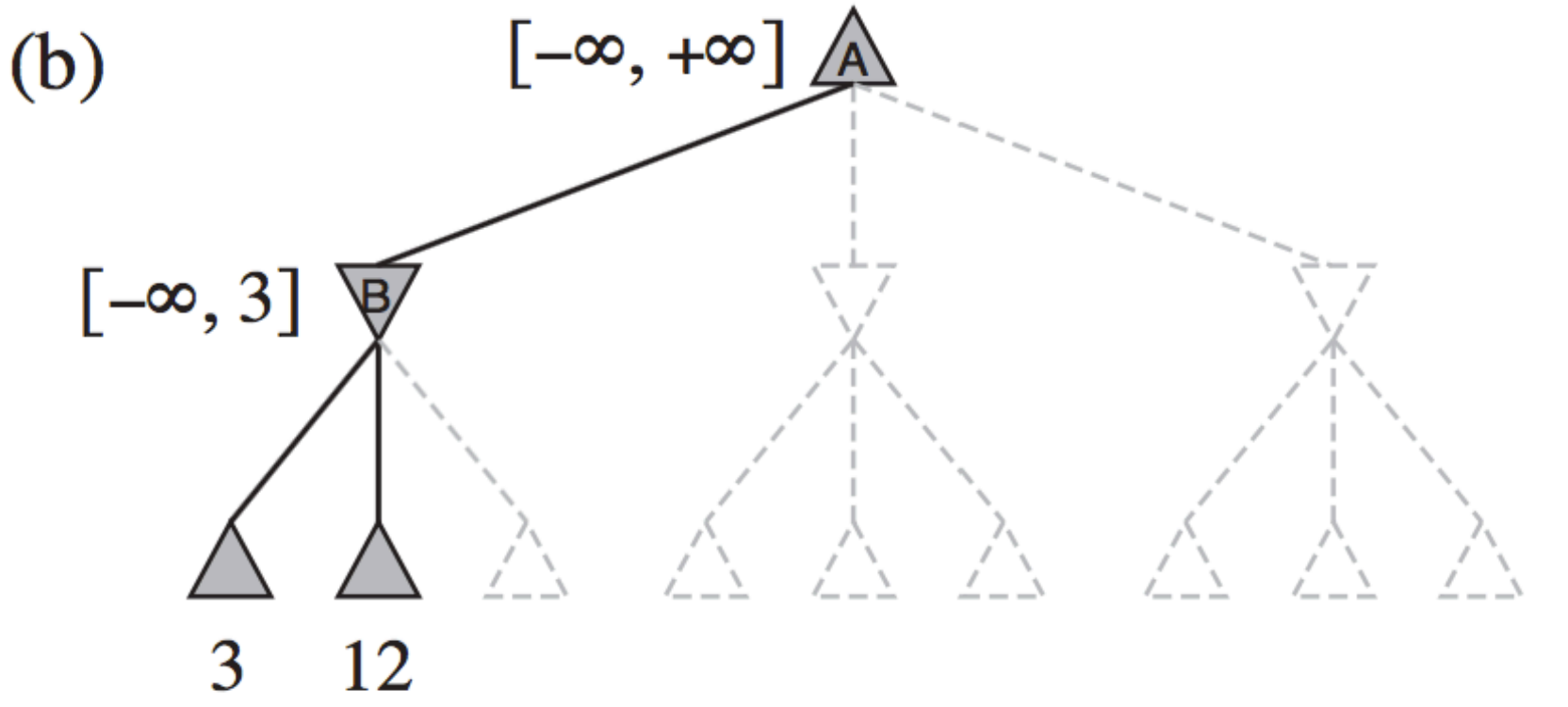
- عمل جستجو تحت گره MAX که مقدار آلفای آن بزرگ‌تر یا مساوی مقدار بتای هر گره MIN اجداد آن گره است، قطع می‌شود.
- در این حالت، مقداری که تا به‌حال به گره‌ی MAX نسبت داده شده به‌عنوان مقدار نهایی آلفای آن گره انتخاب خواهد شد.



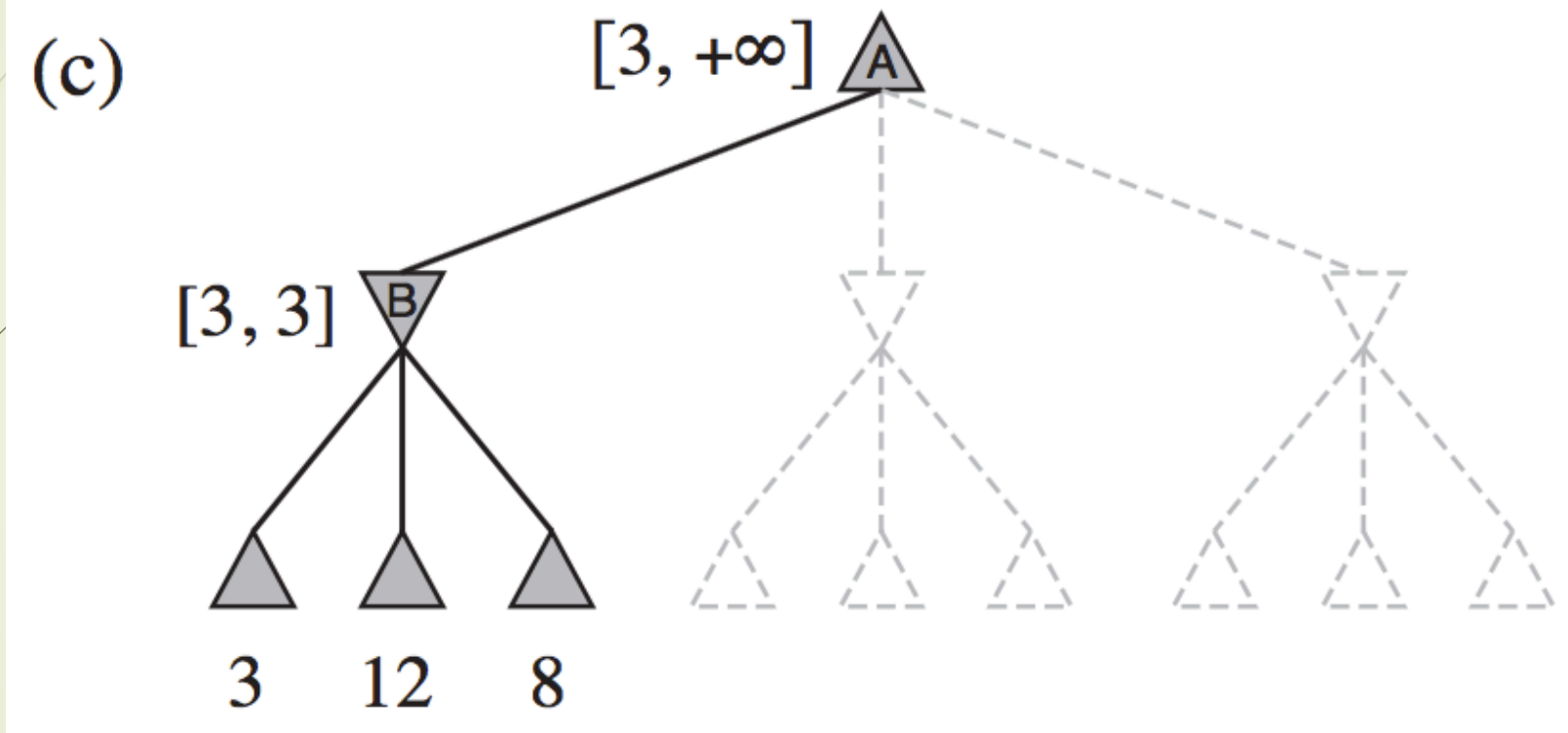
آلفا– بتا (مثال ۱)



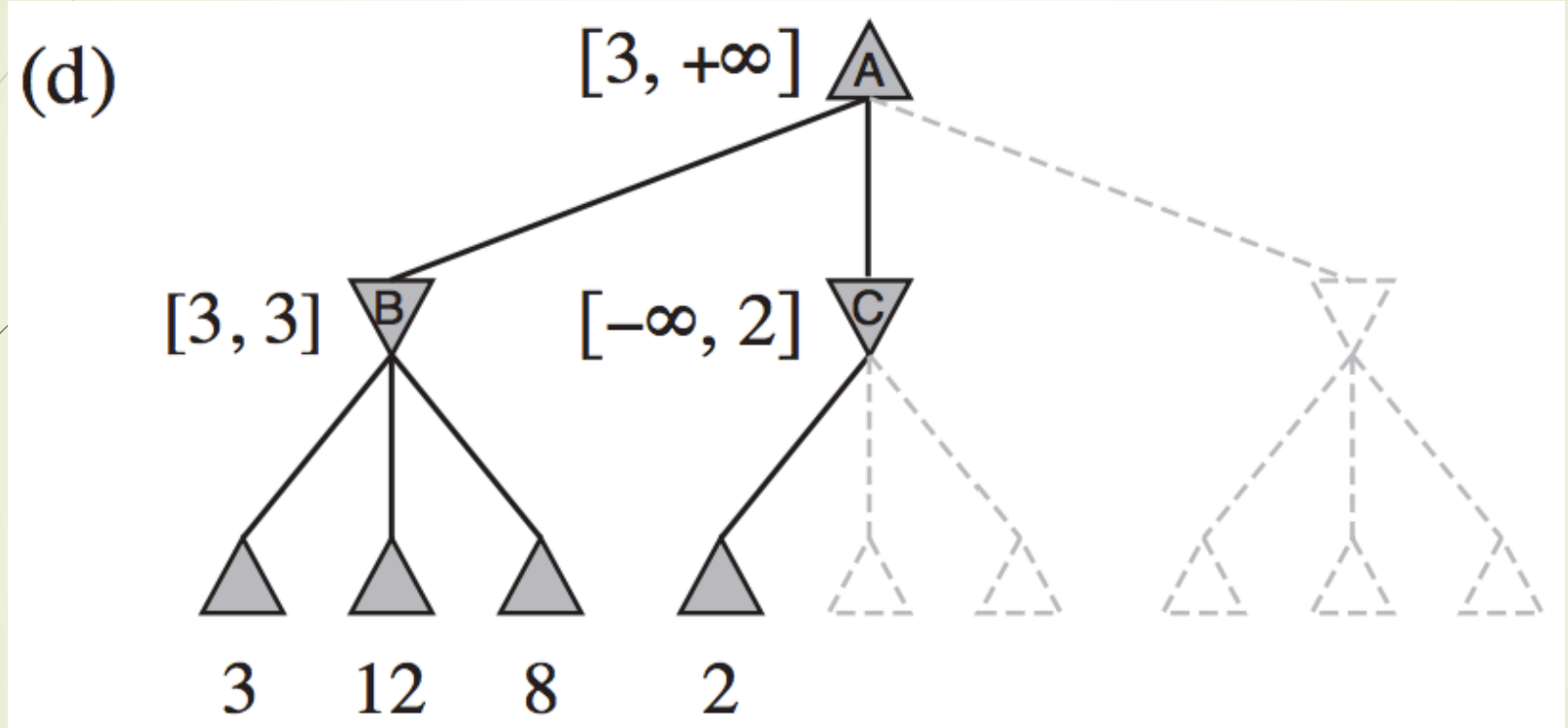
آلفا– بتا (مثال ۱)



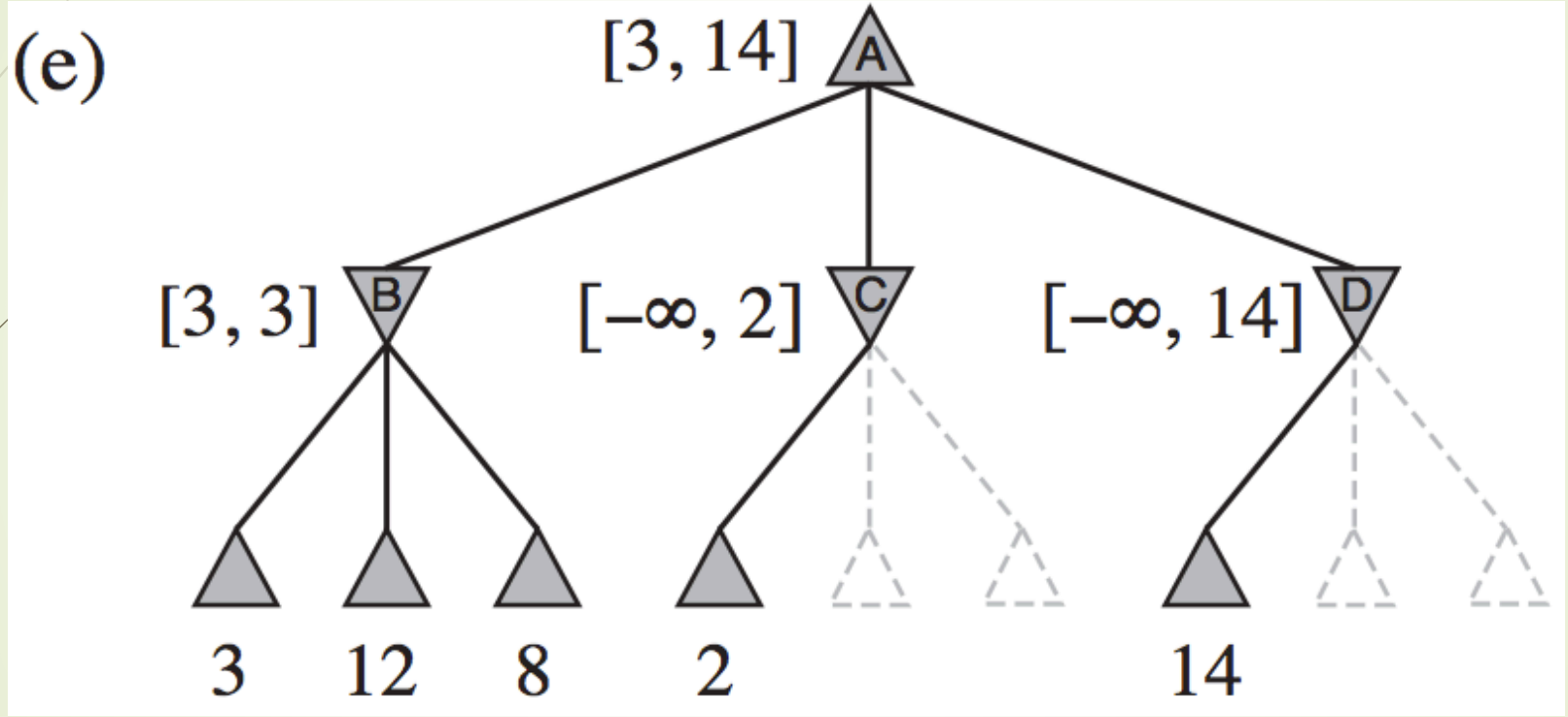
آلفا– بتا (مثال ۱)



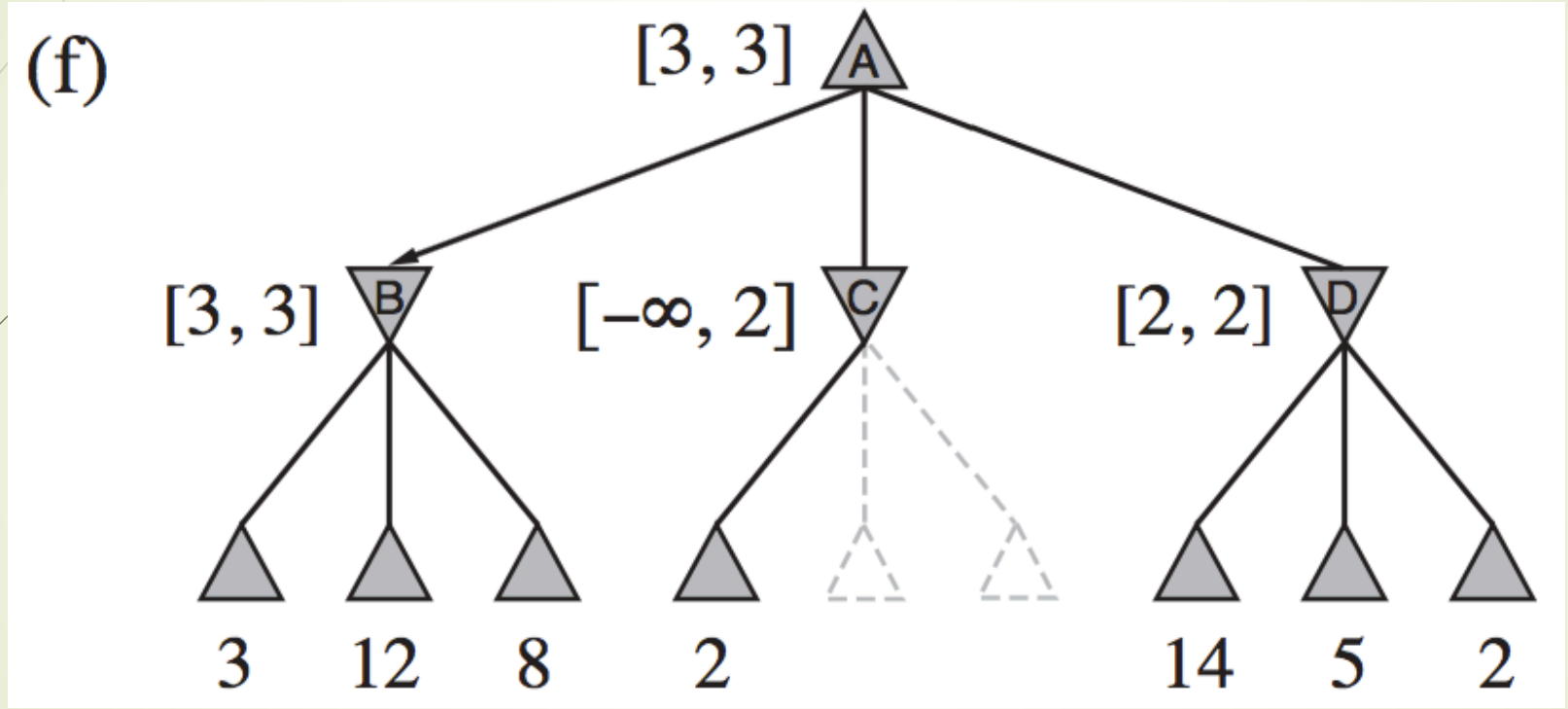
آلفا– بتا (مثال ۱)



آلفا– بتا (مثال ۱)



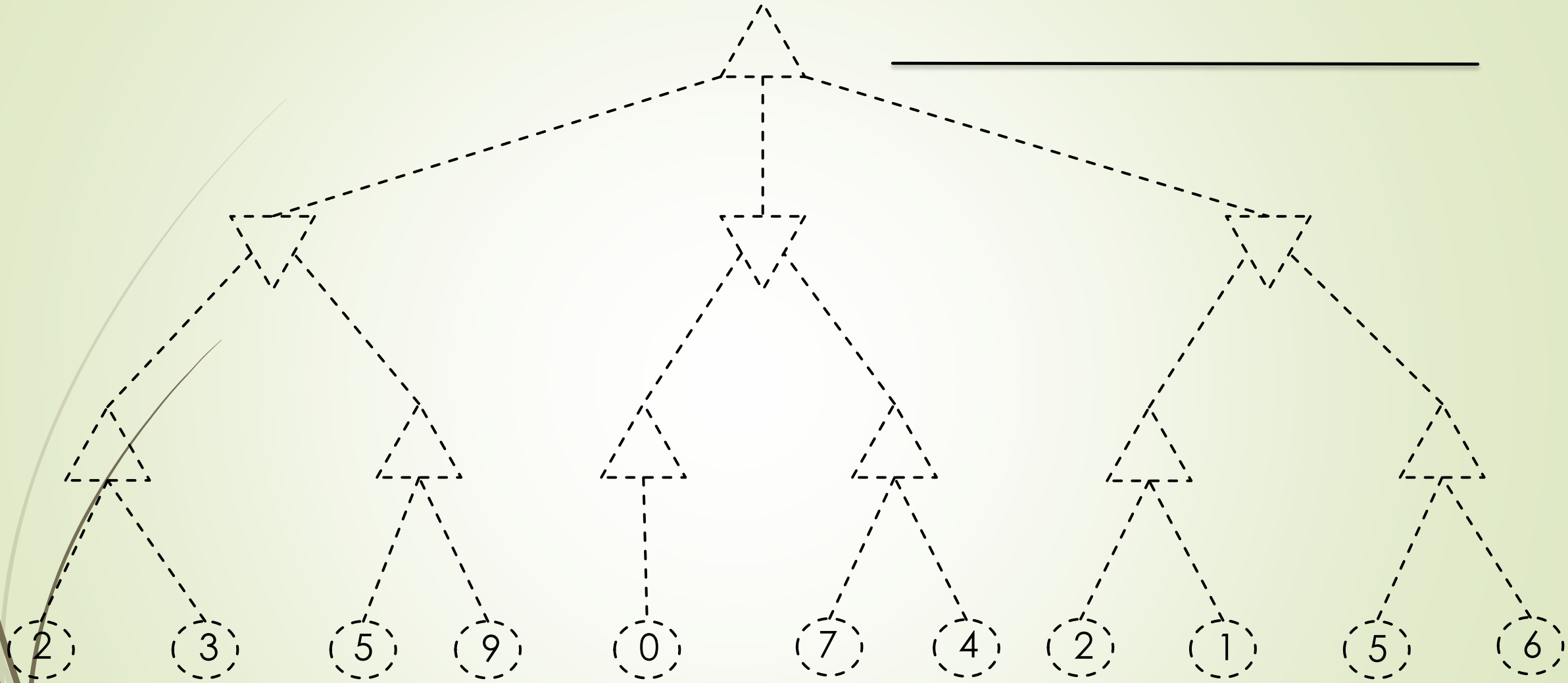
آلفا– بتا (مثال ۱)



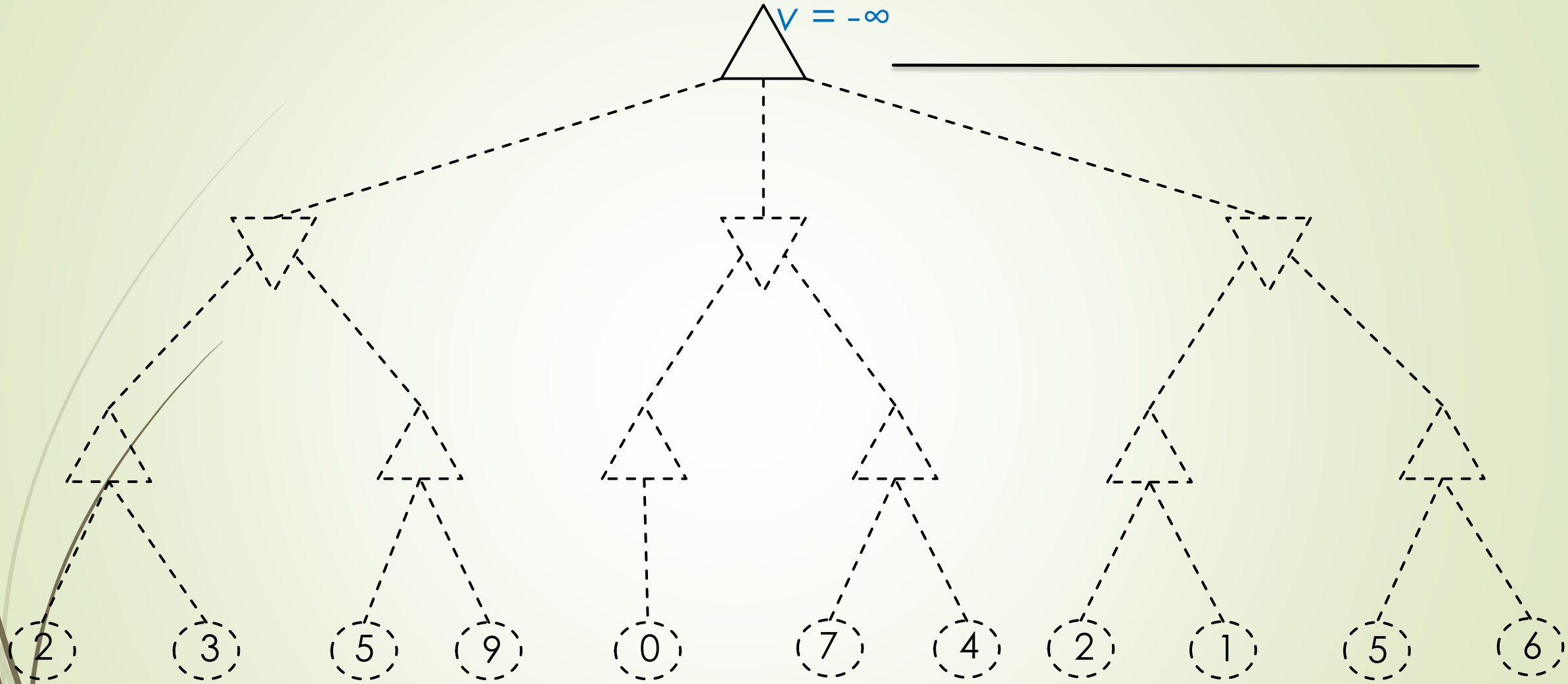
الگوریتم آلفا–بتا

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

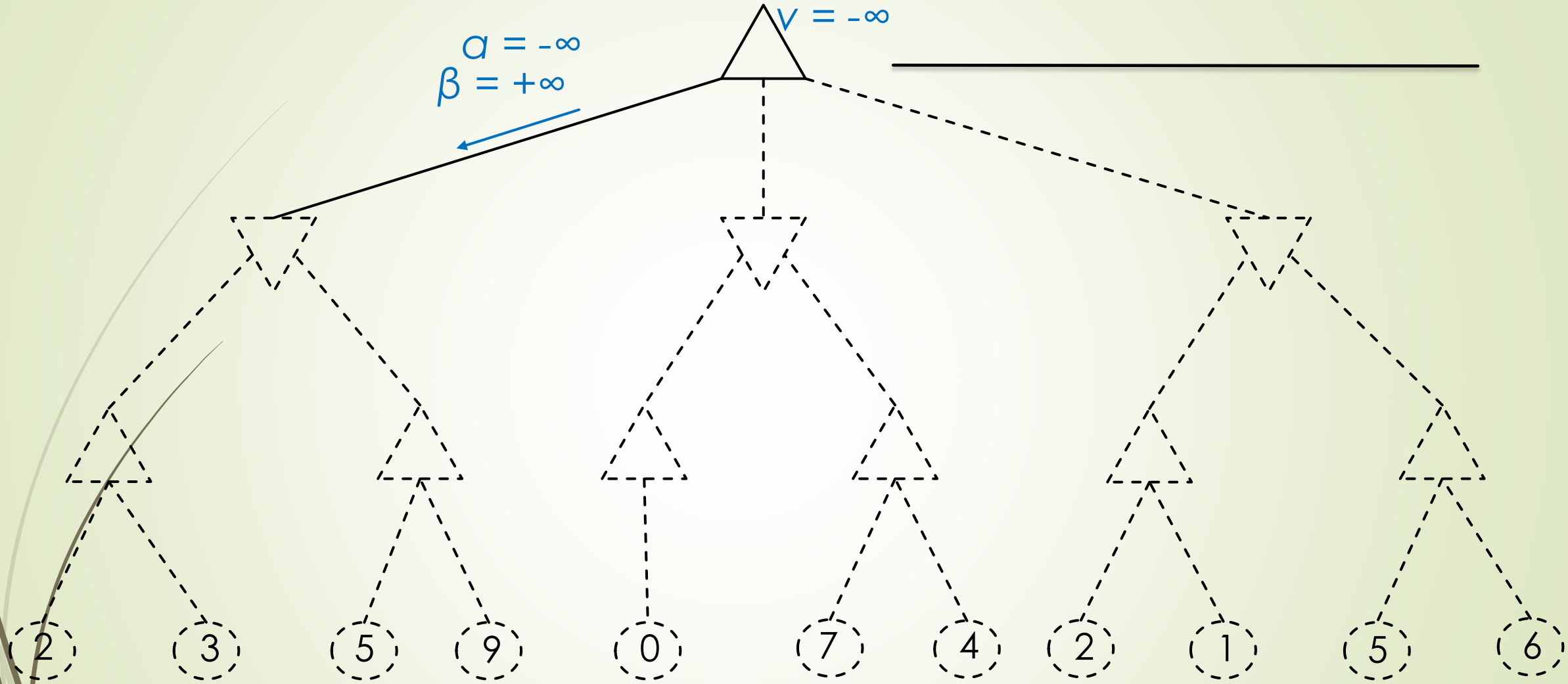
آلفا– بتا (مثال ۲)



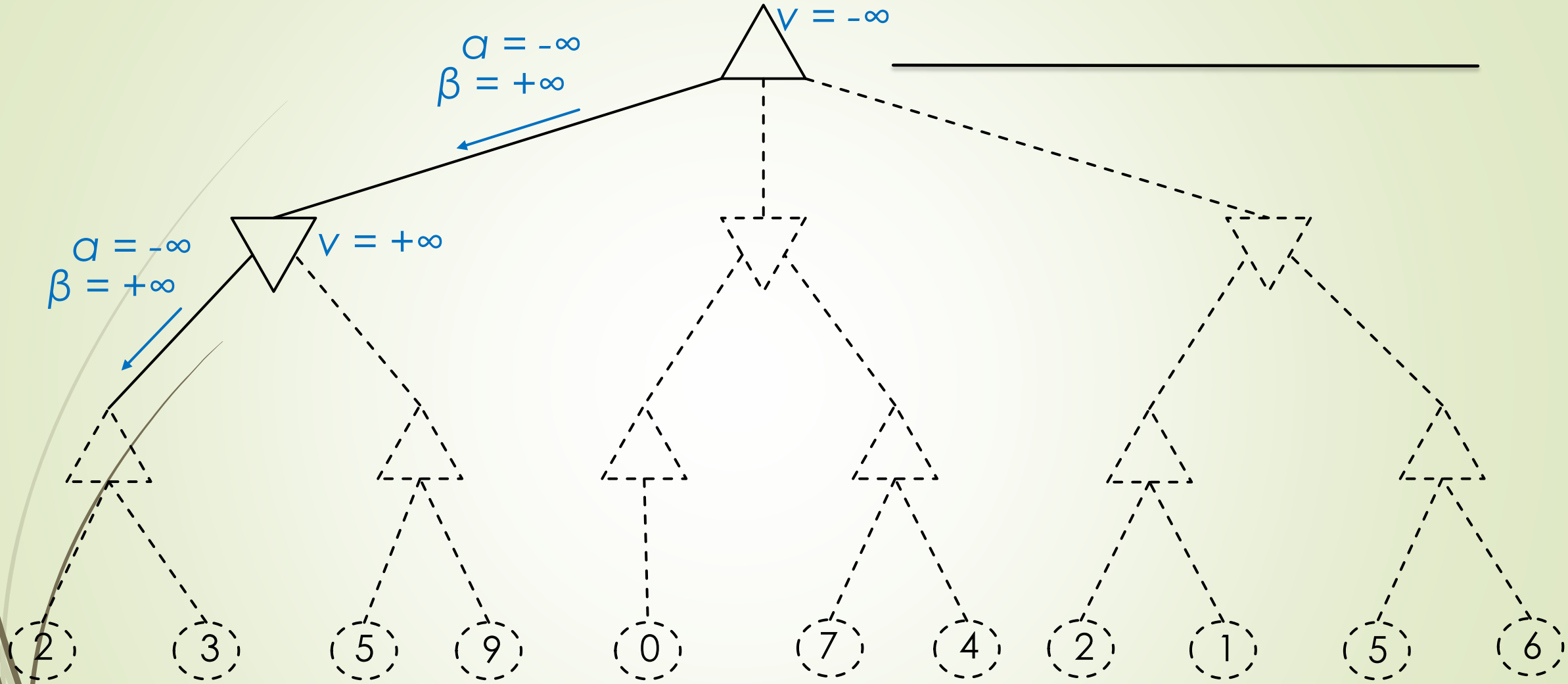
آلفا– بتا (مثال ۲)



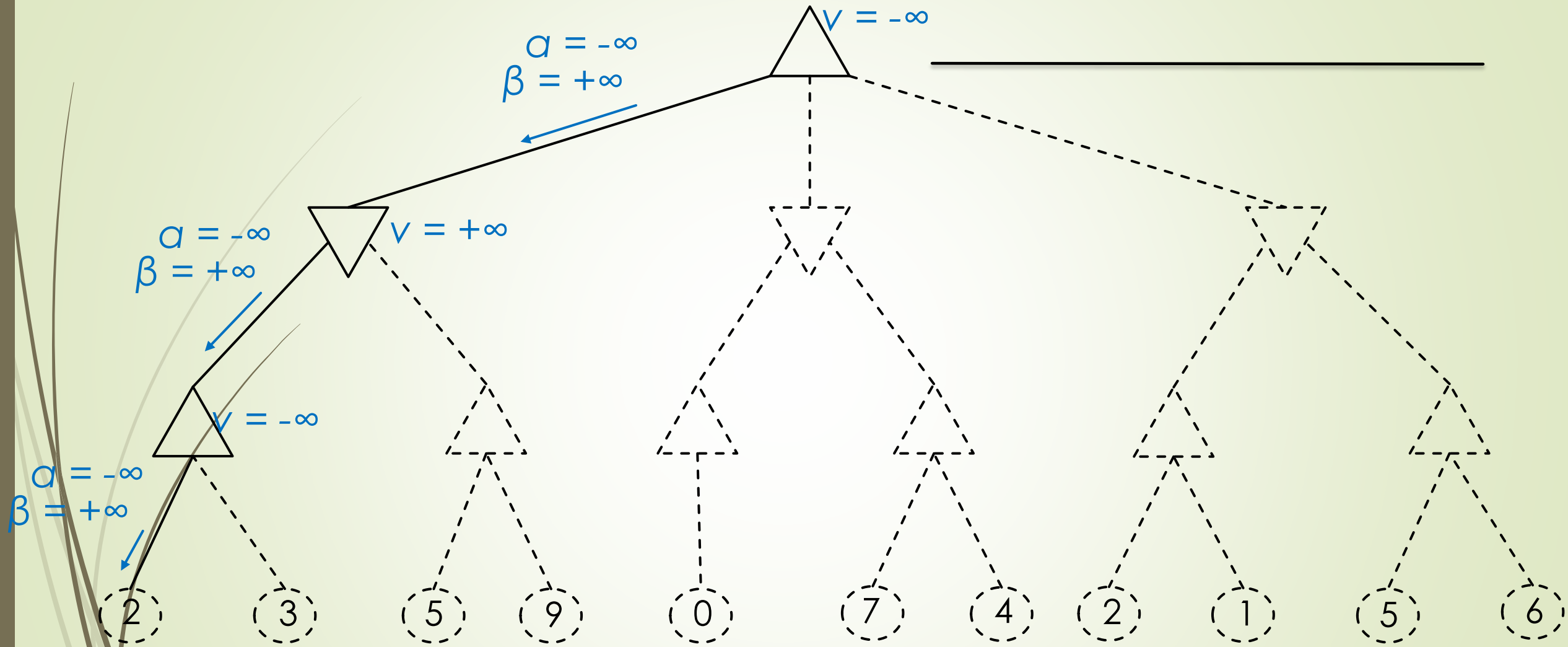
آلفا– بتا (مثال ۲)



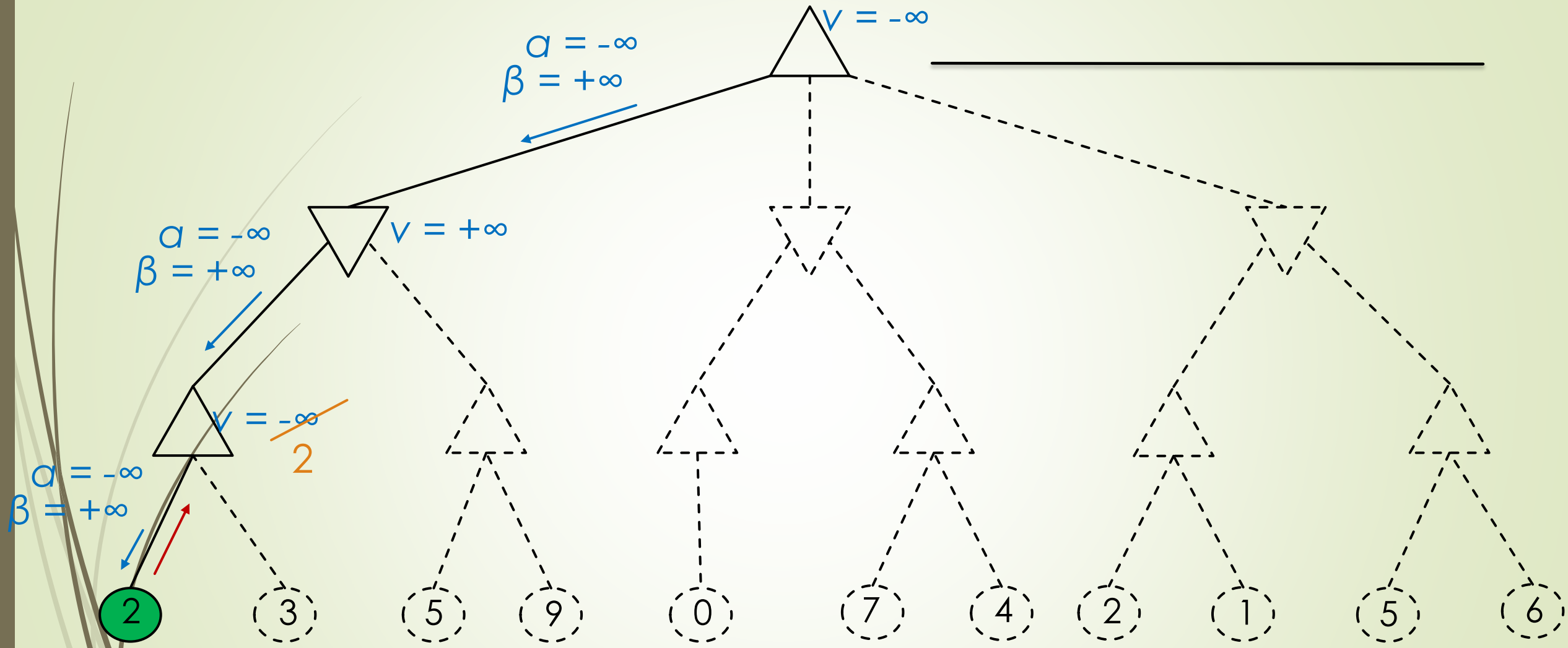
آلفا– بتا (مثال ۲)



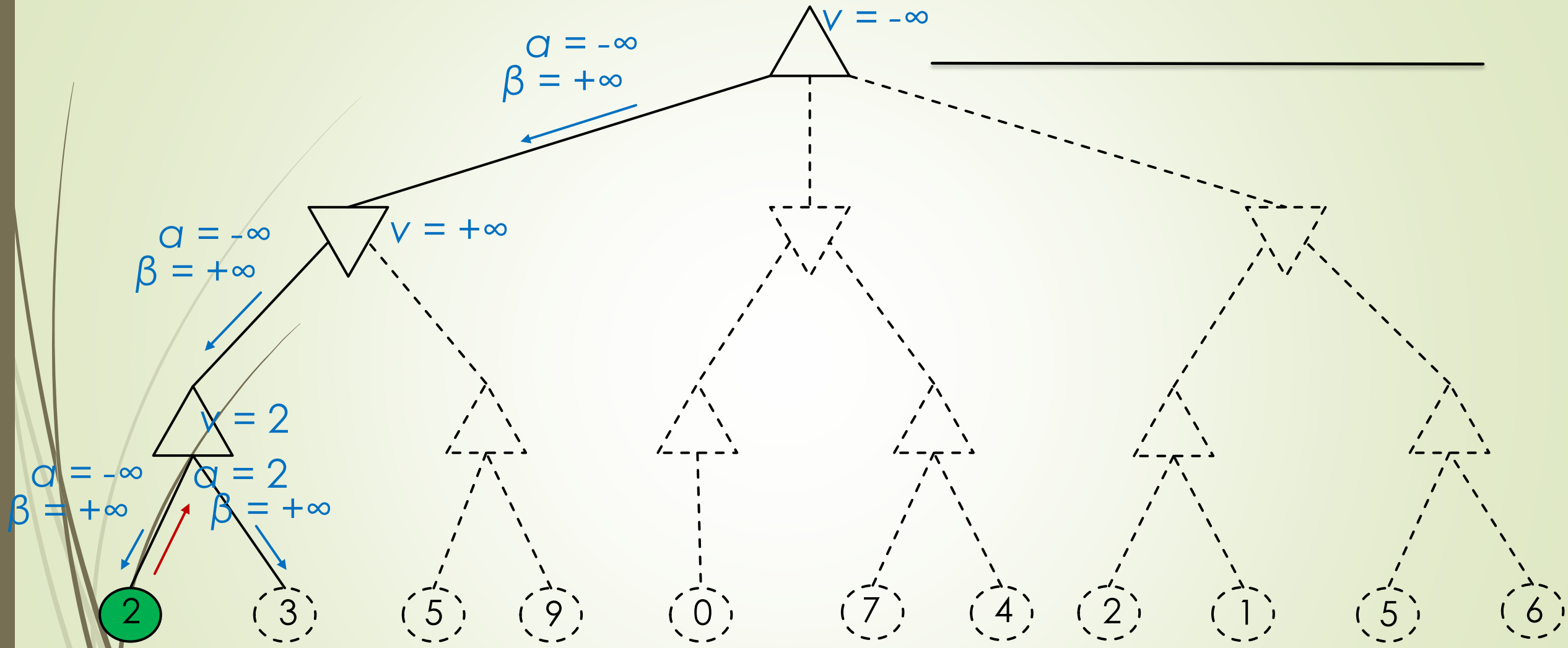
آلفا– بتا (مثال ۲)



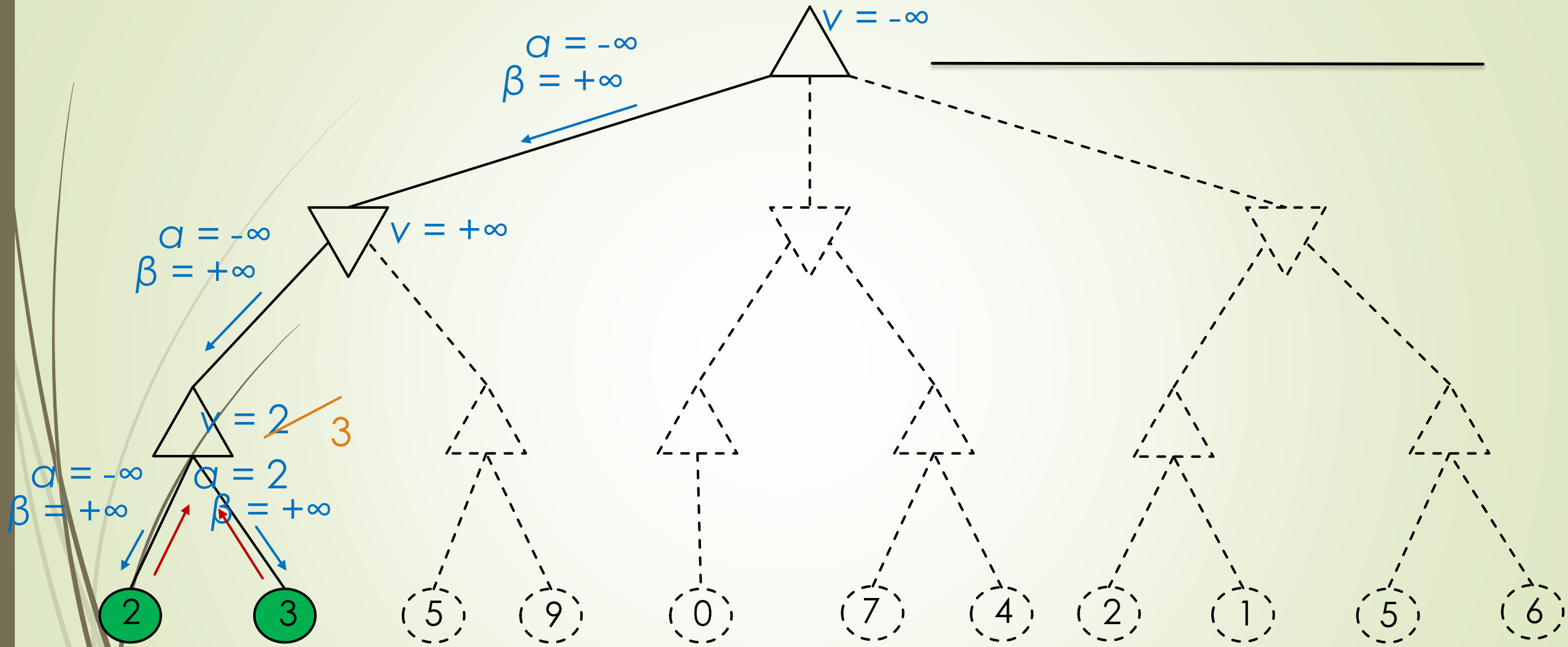
آلفا– بتا (مثال ۲)



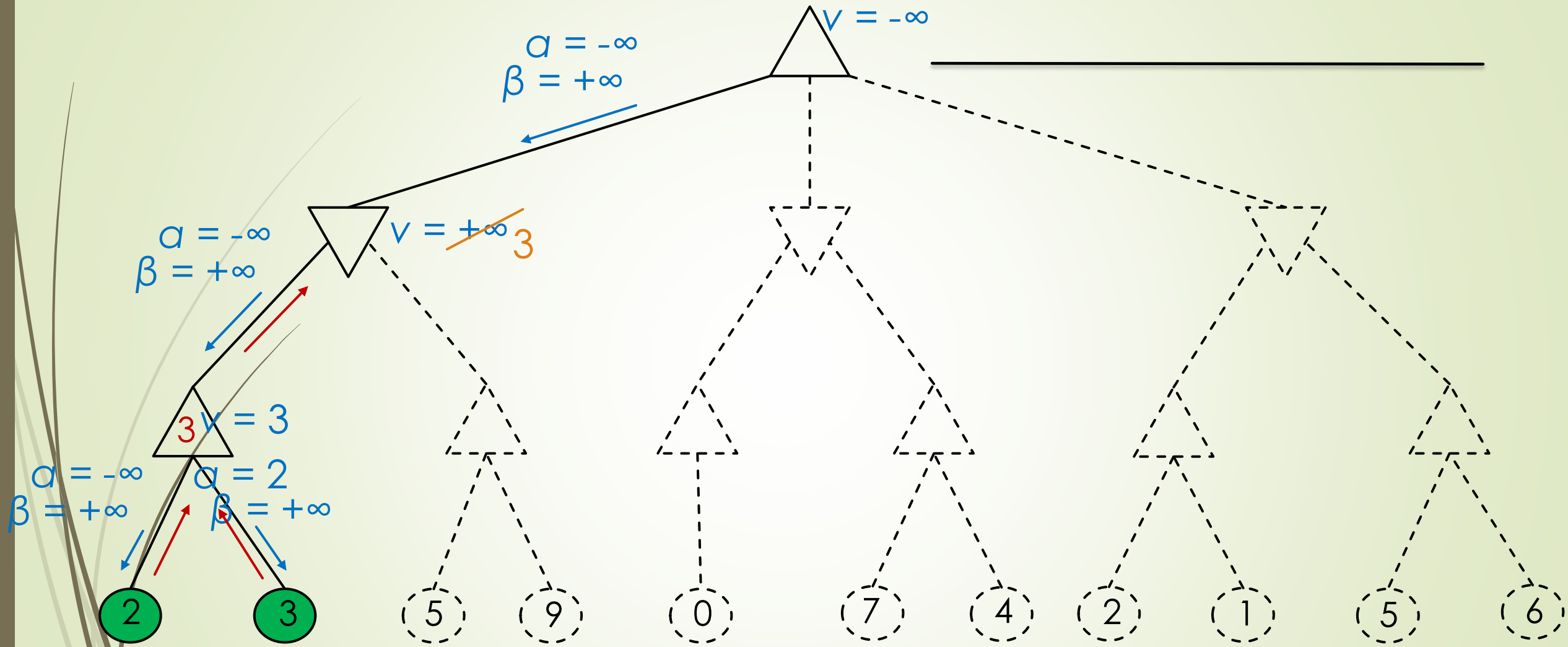
آلفا– بتا (مثال ۲)



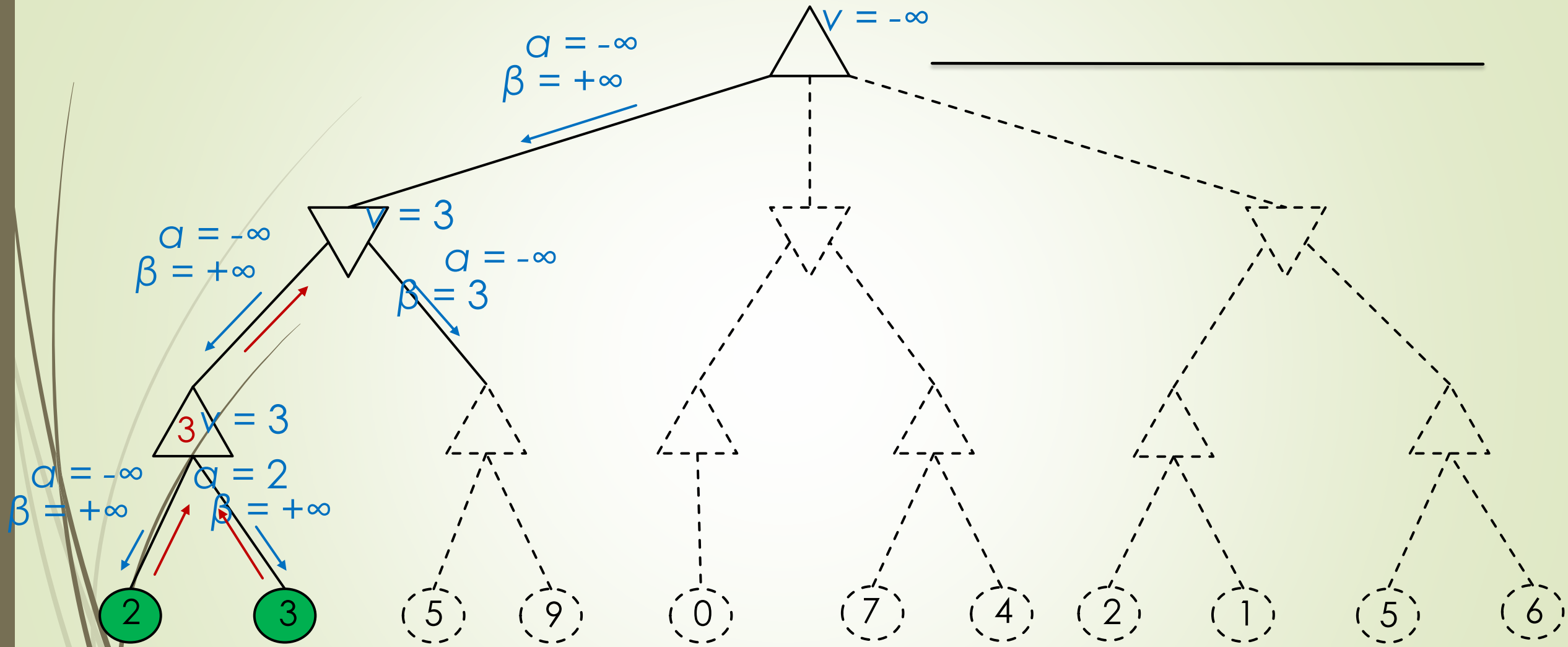
آلفا– بتا (مثال ۲)



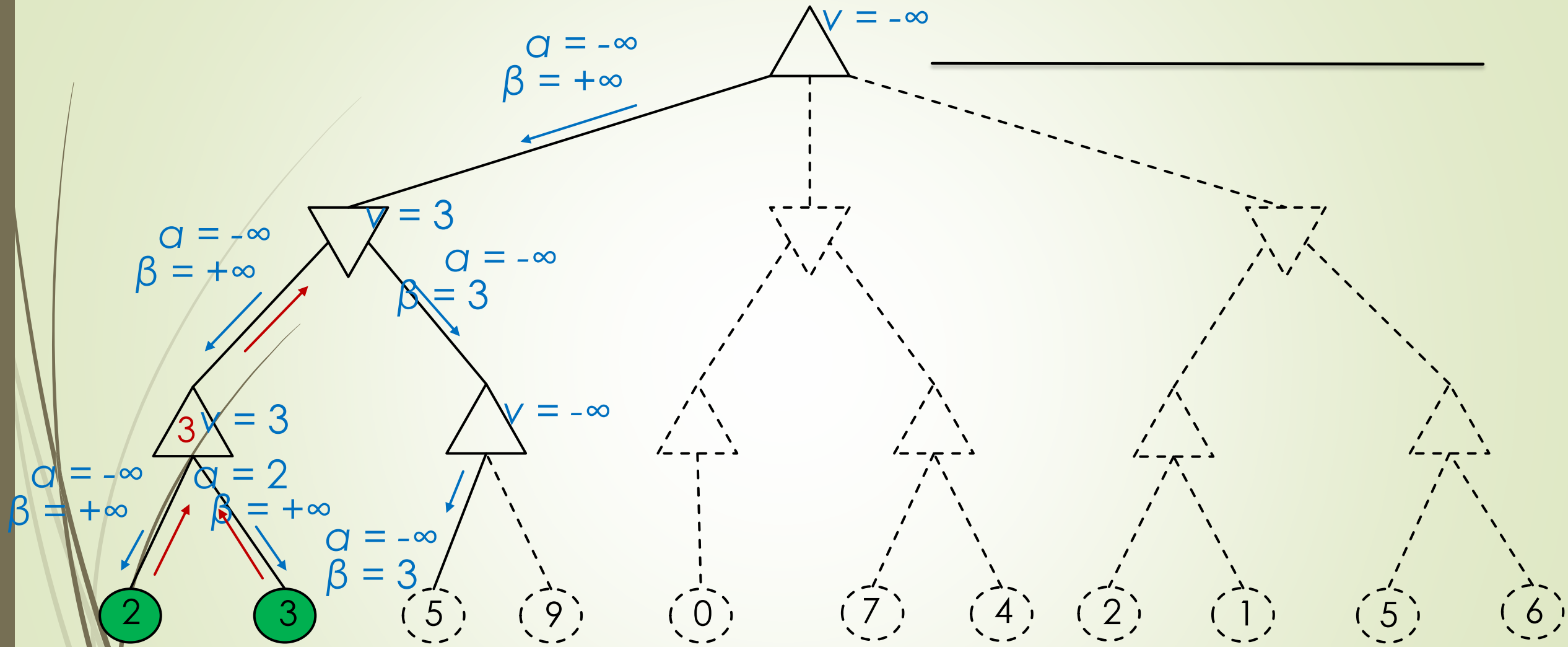
آلفا– بتا (مثال ۲)



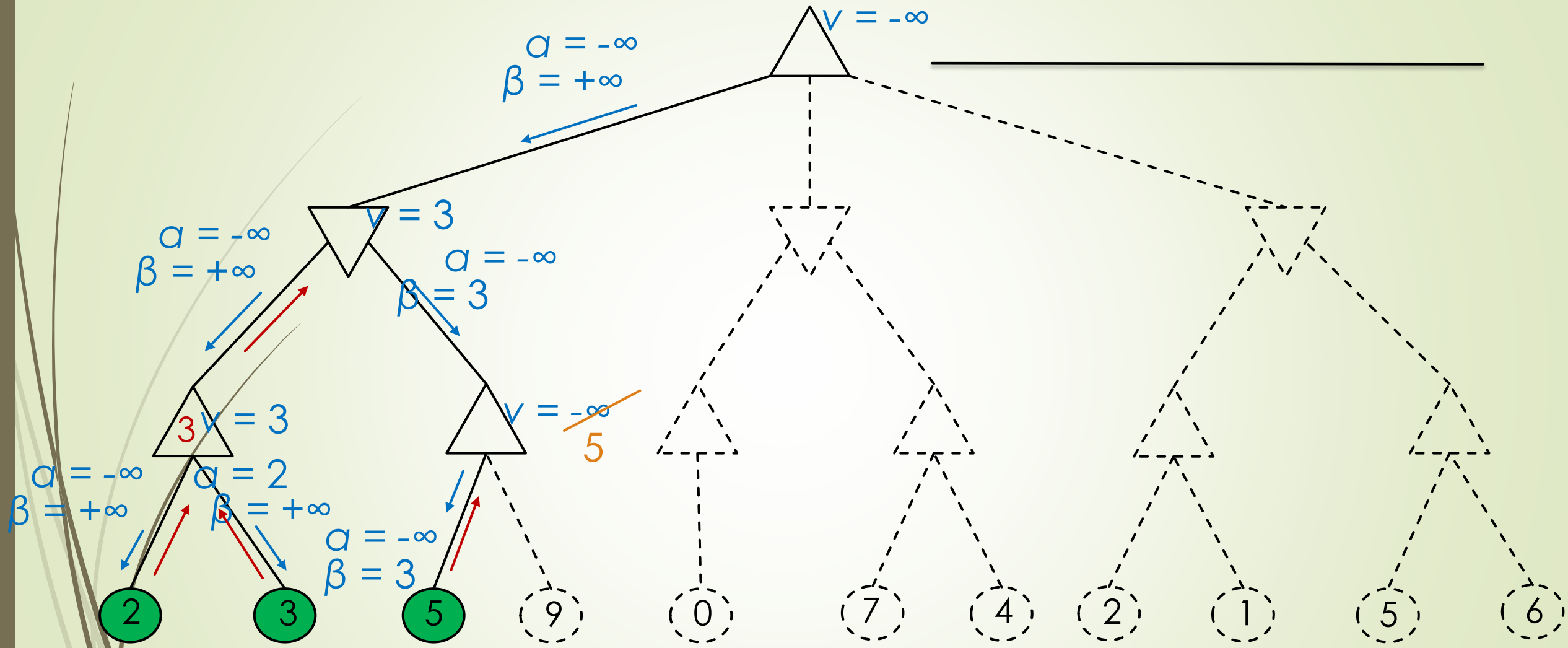
آلفا– بتا (مثال ۲)



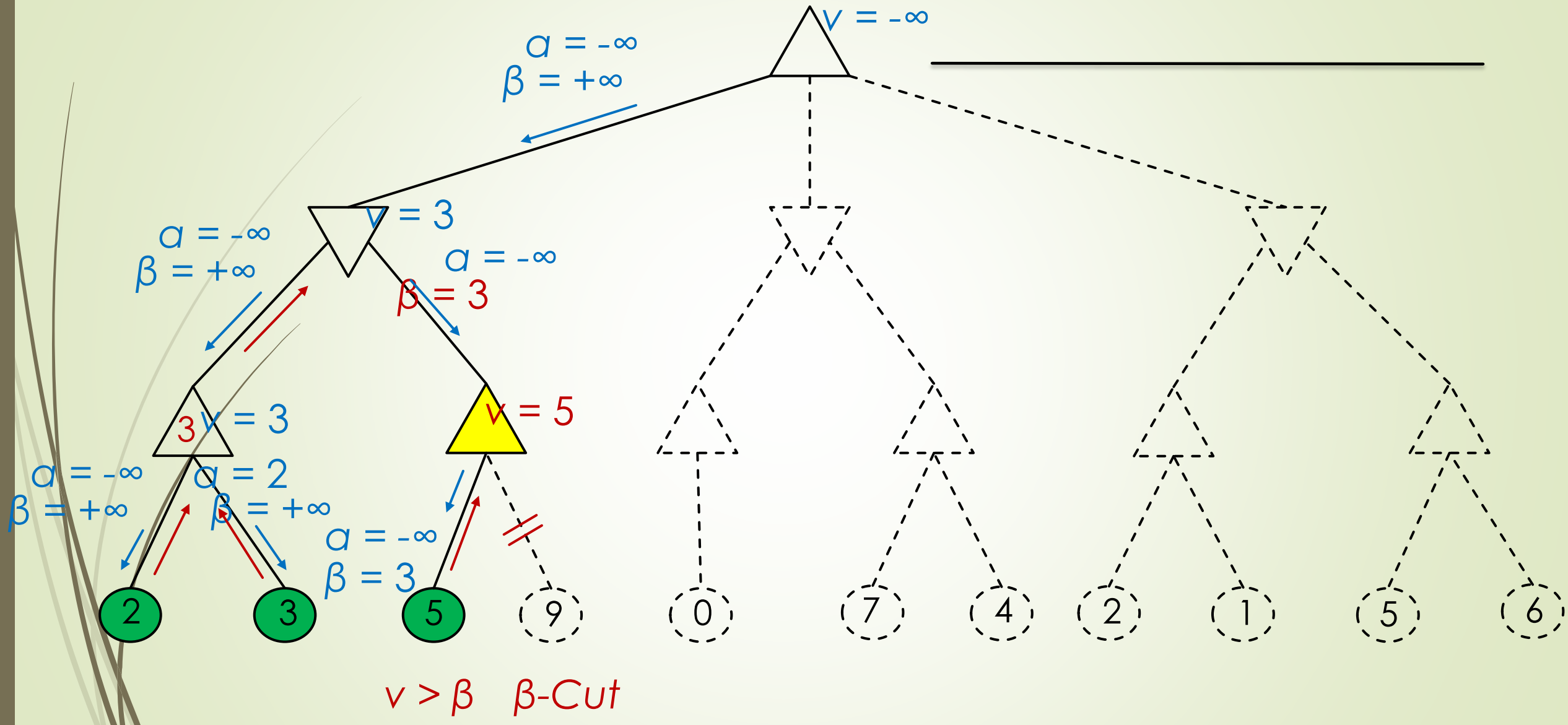
آلفا– بتا (مثال ۲)



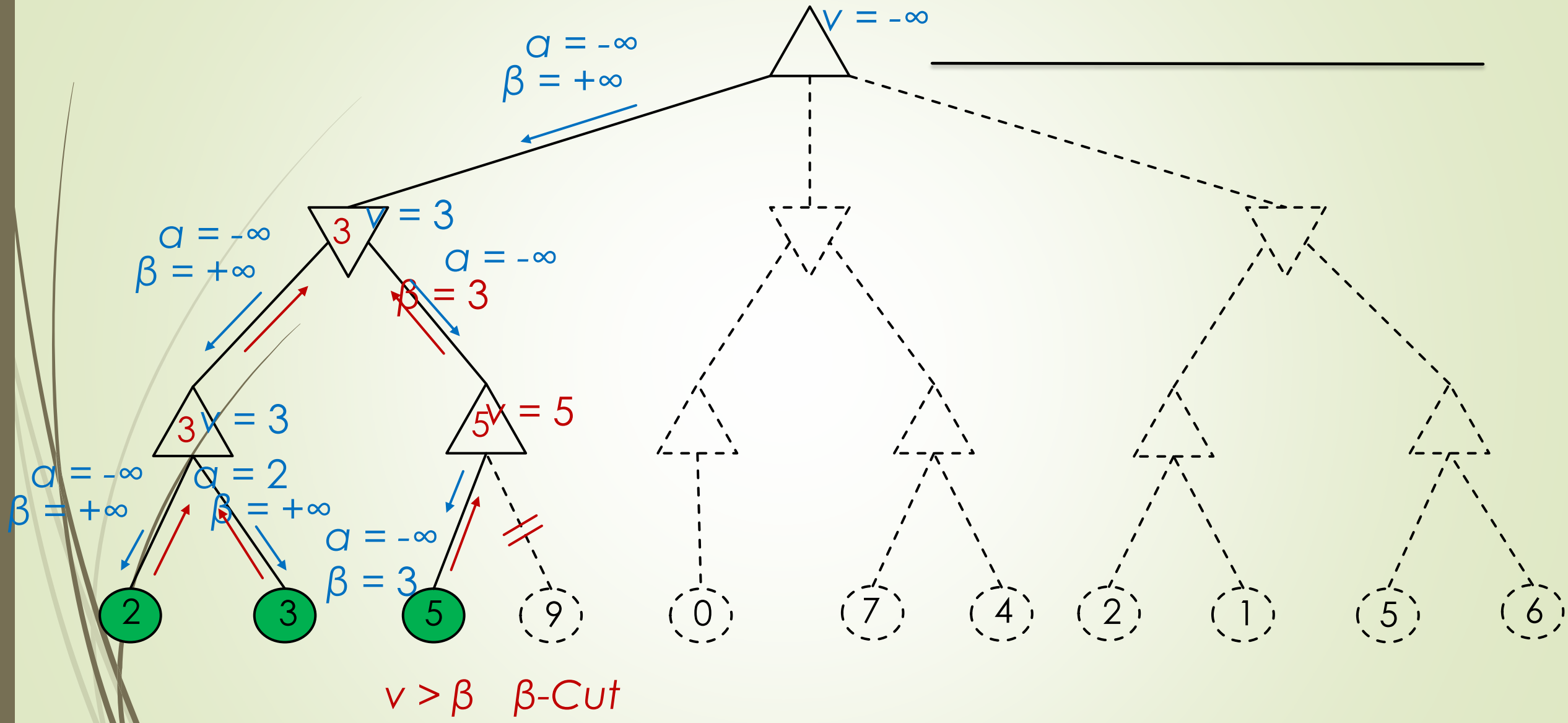
آلفا– بتا (مثال ۲)



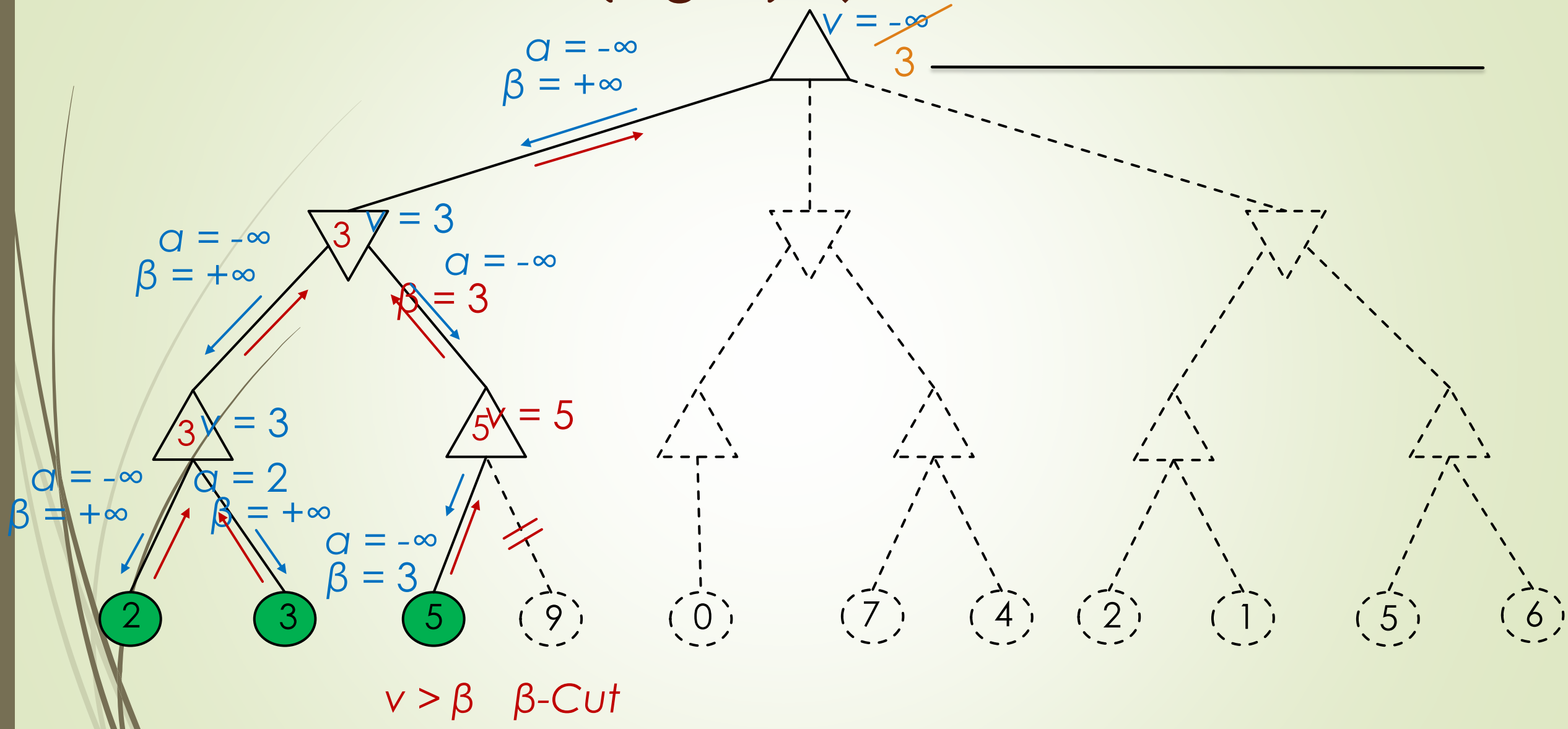
آلفا– بتا (مثال ۲)



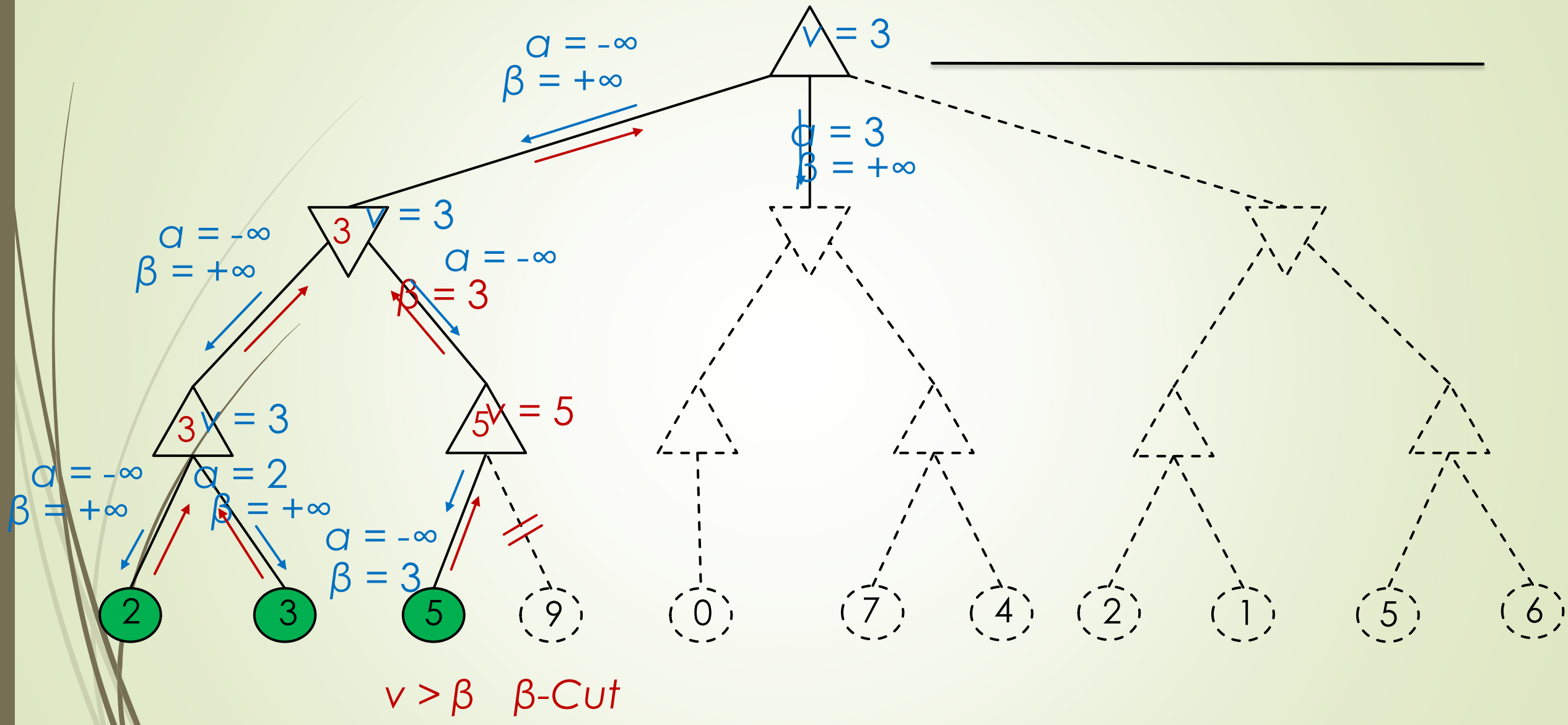
آلفا– بتا (مثال ۲)



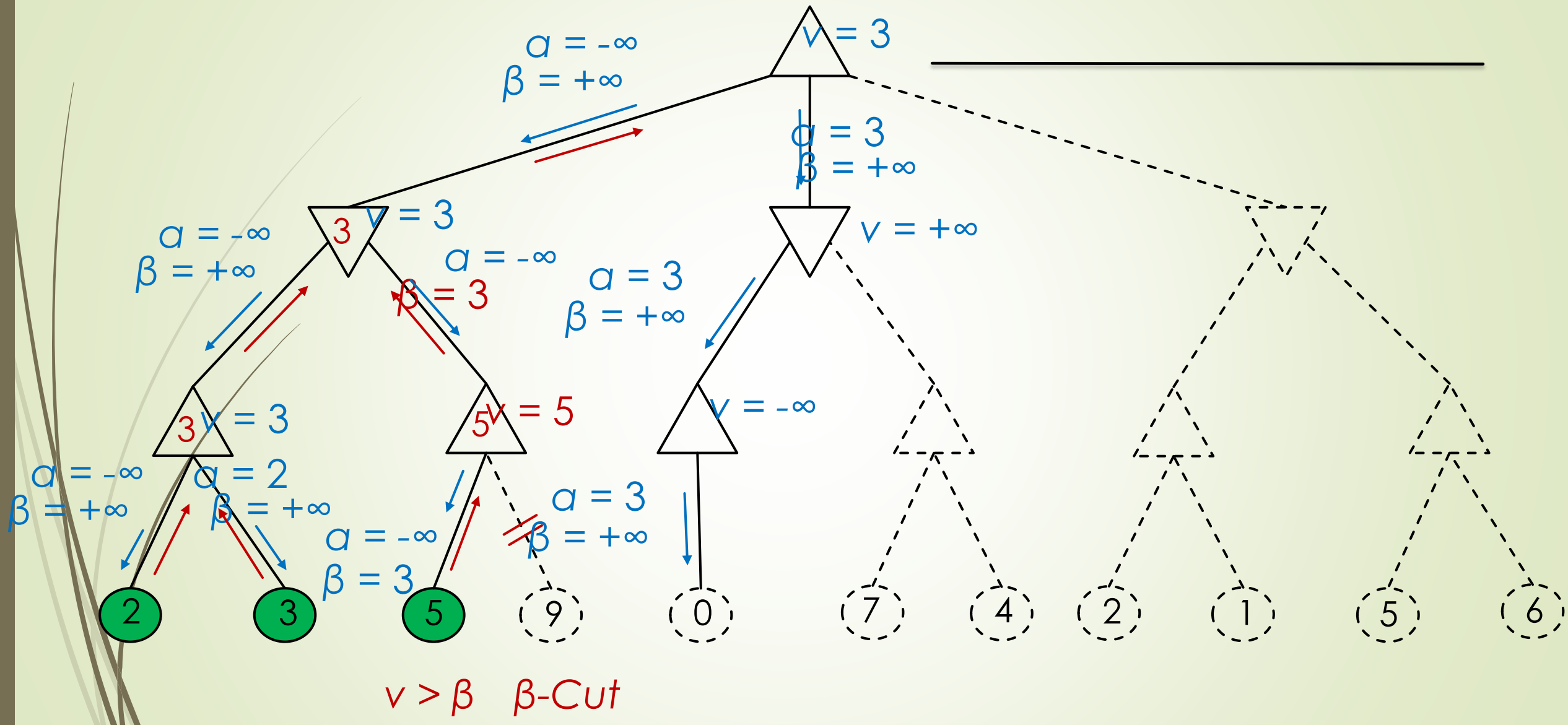
آلفا– بتا (مثال ۲)



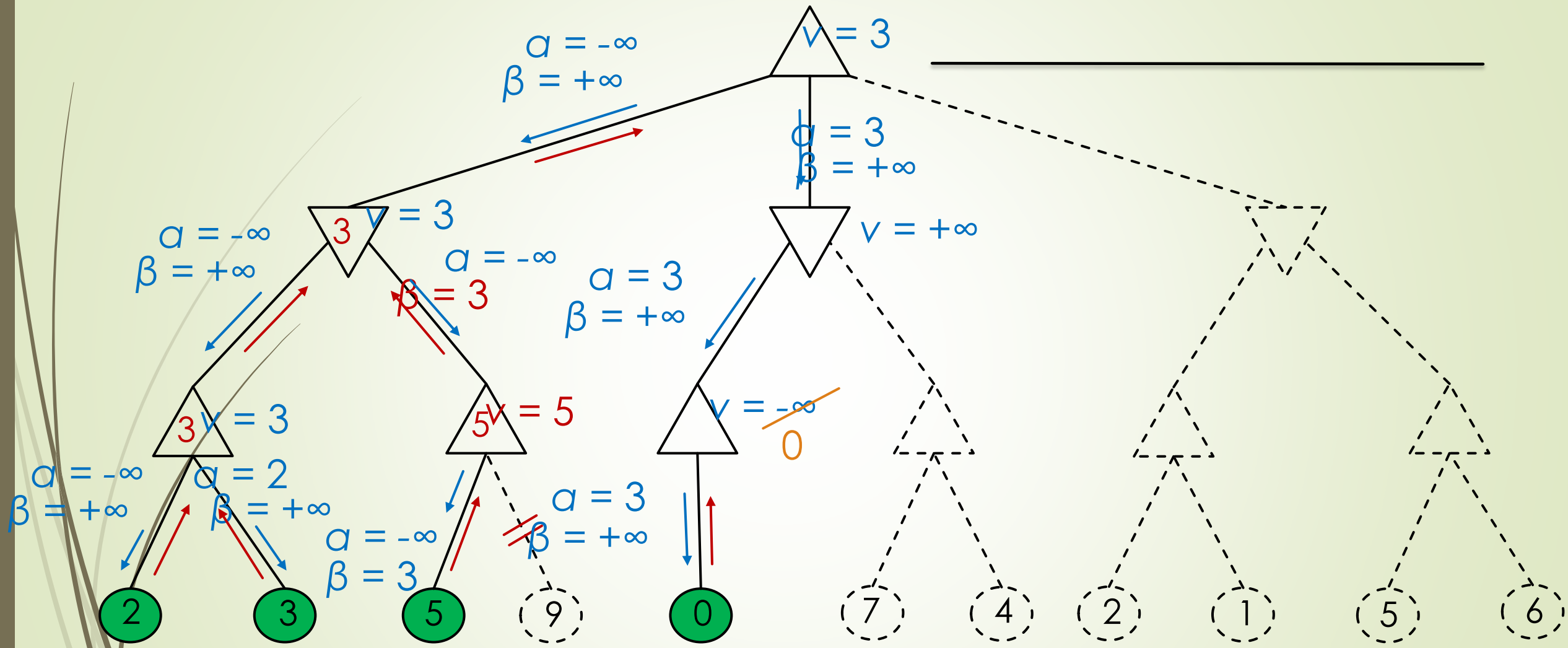
آلفا– بتا (مثال ۲)



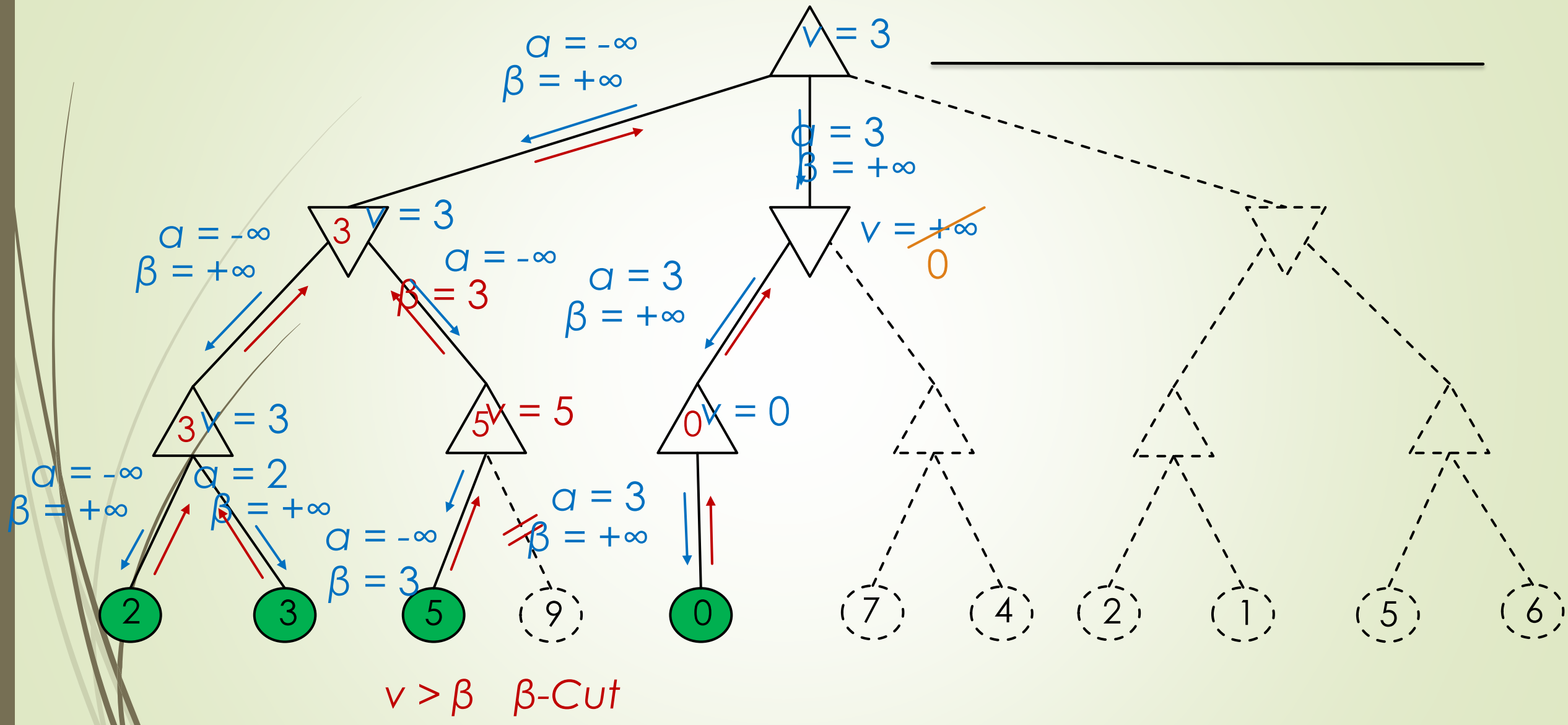
آلفا– بتا (مثال ۲)



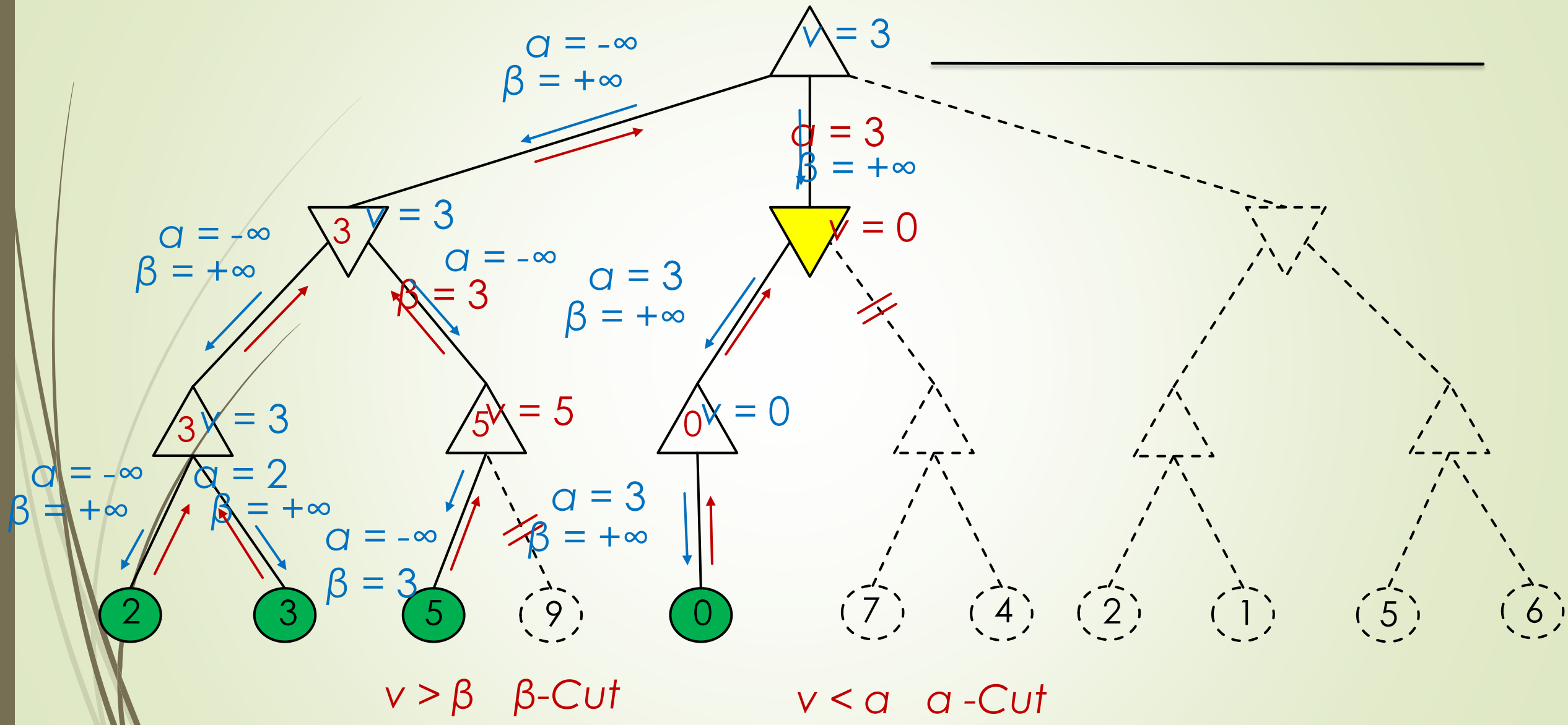
آلفا– بتا (مثال ۲)



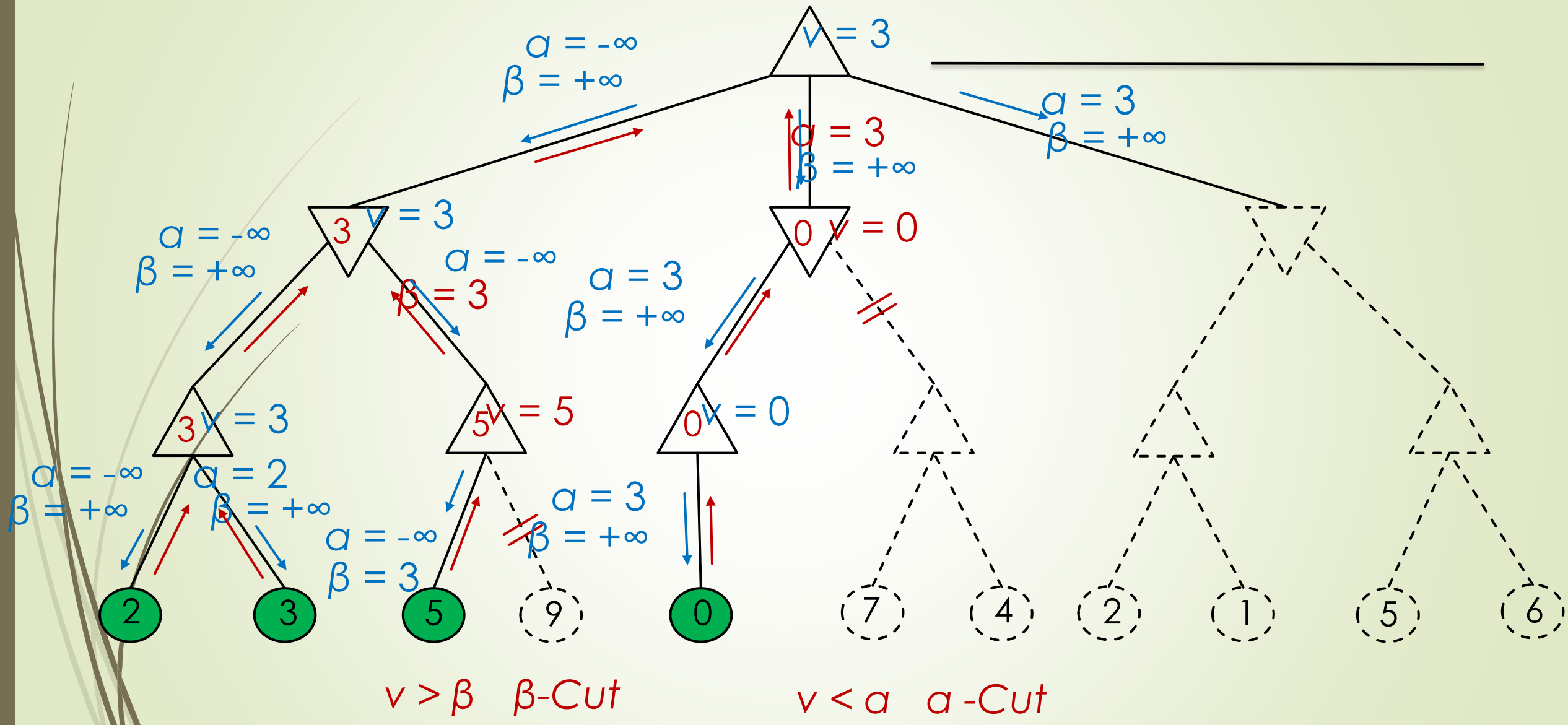
آلفا– بتا (مثال ۲)



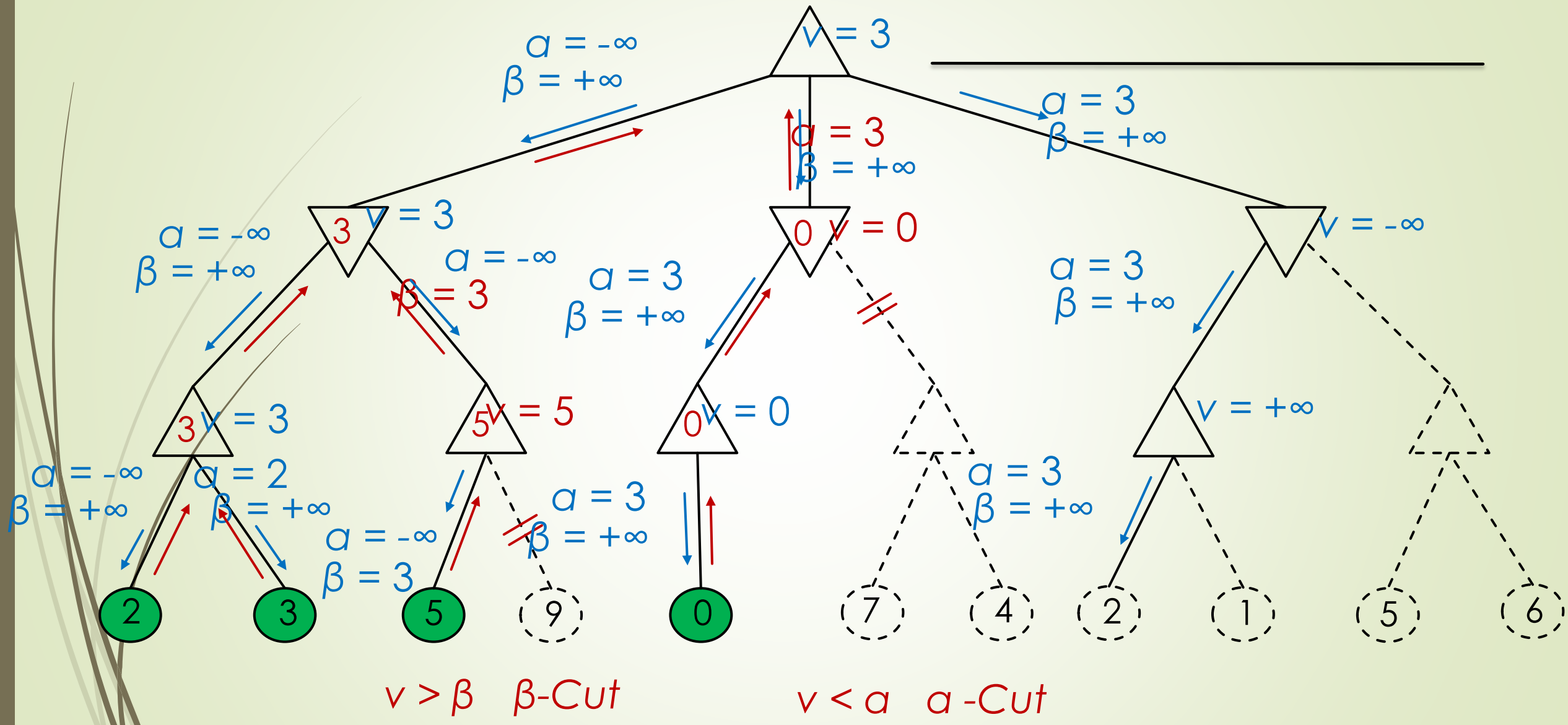
آلفا– بتا (مثال ۲)



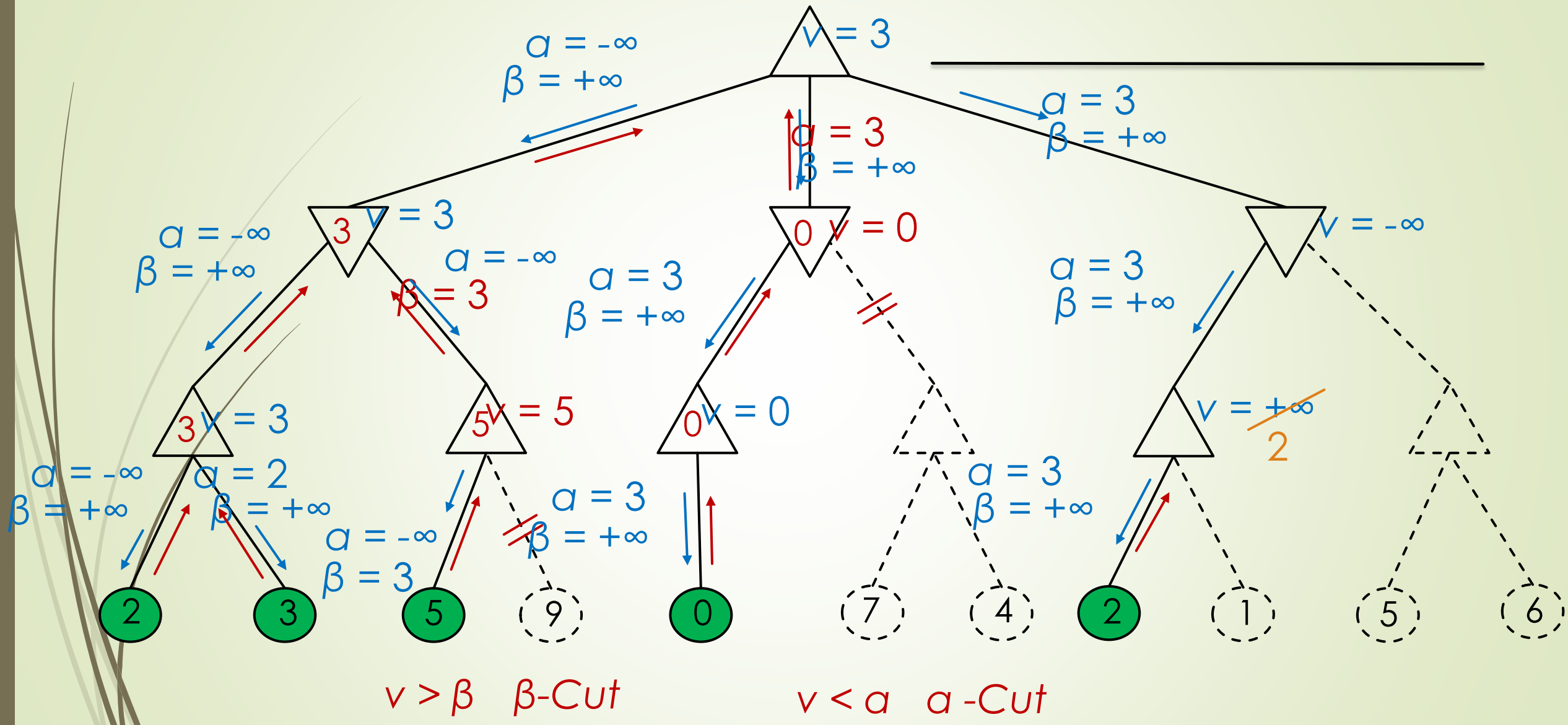
آلفا– بتا (مثال ۲)



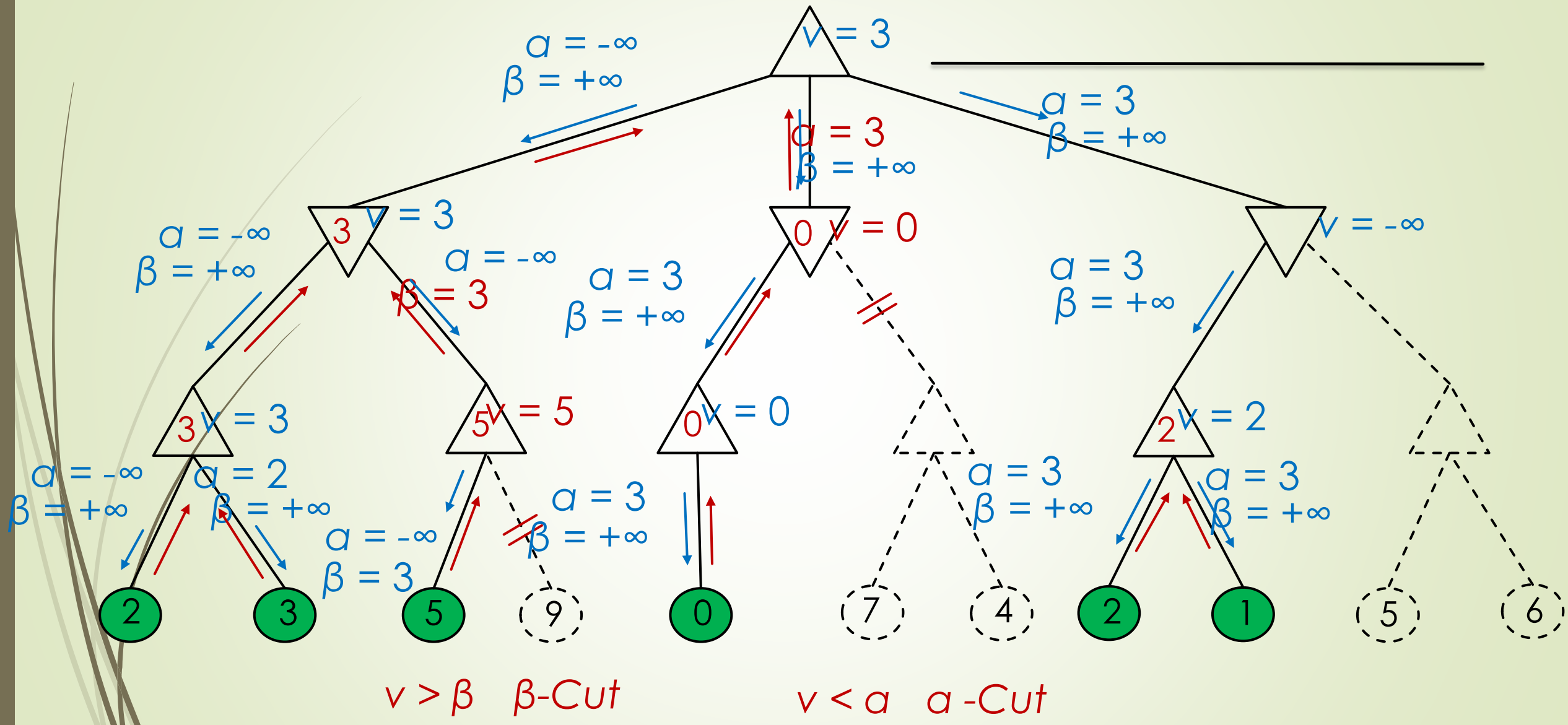
آلفا– بتا (مثال ۲)



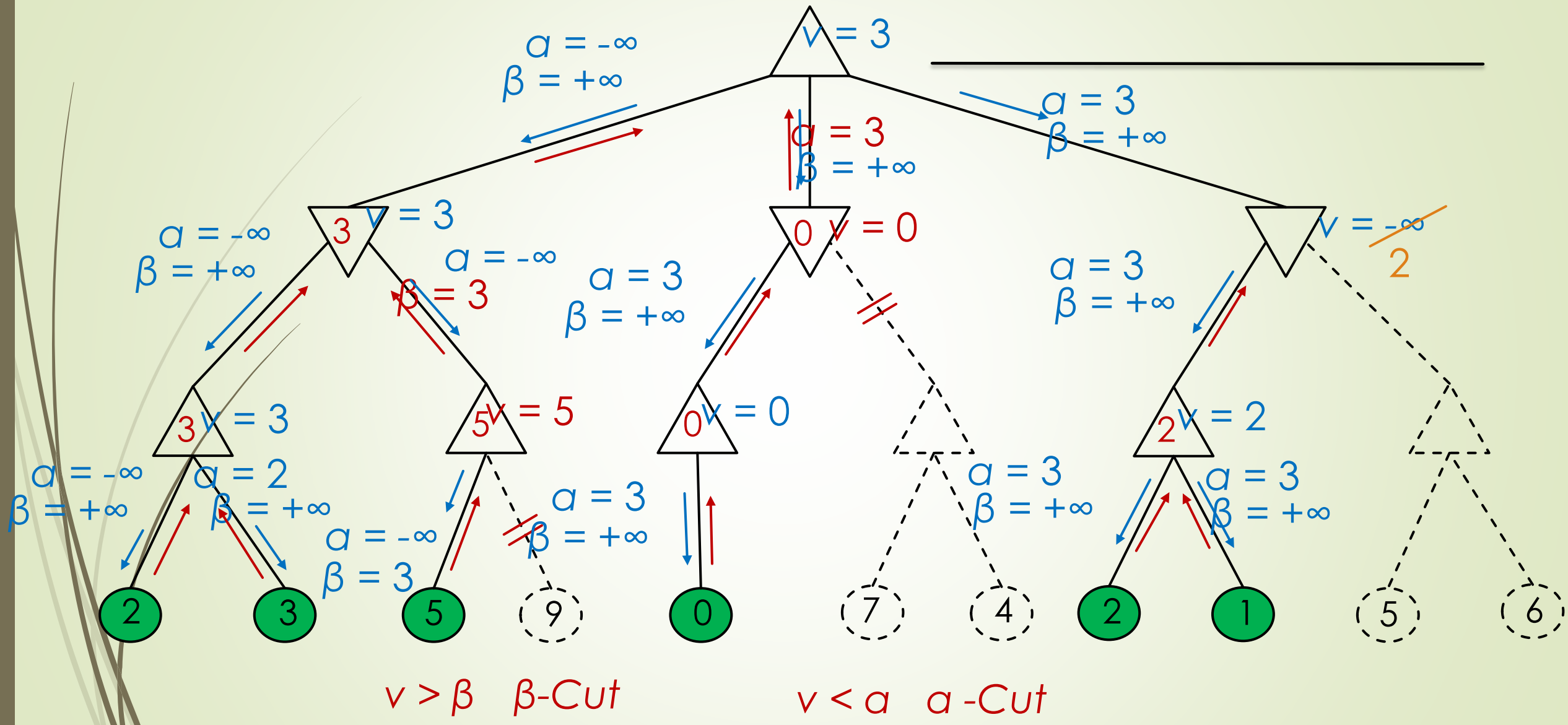
آلفا– بتا (مثال ۲)



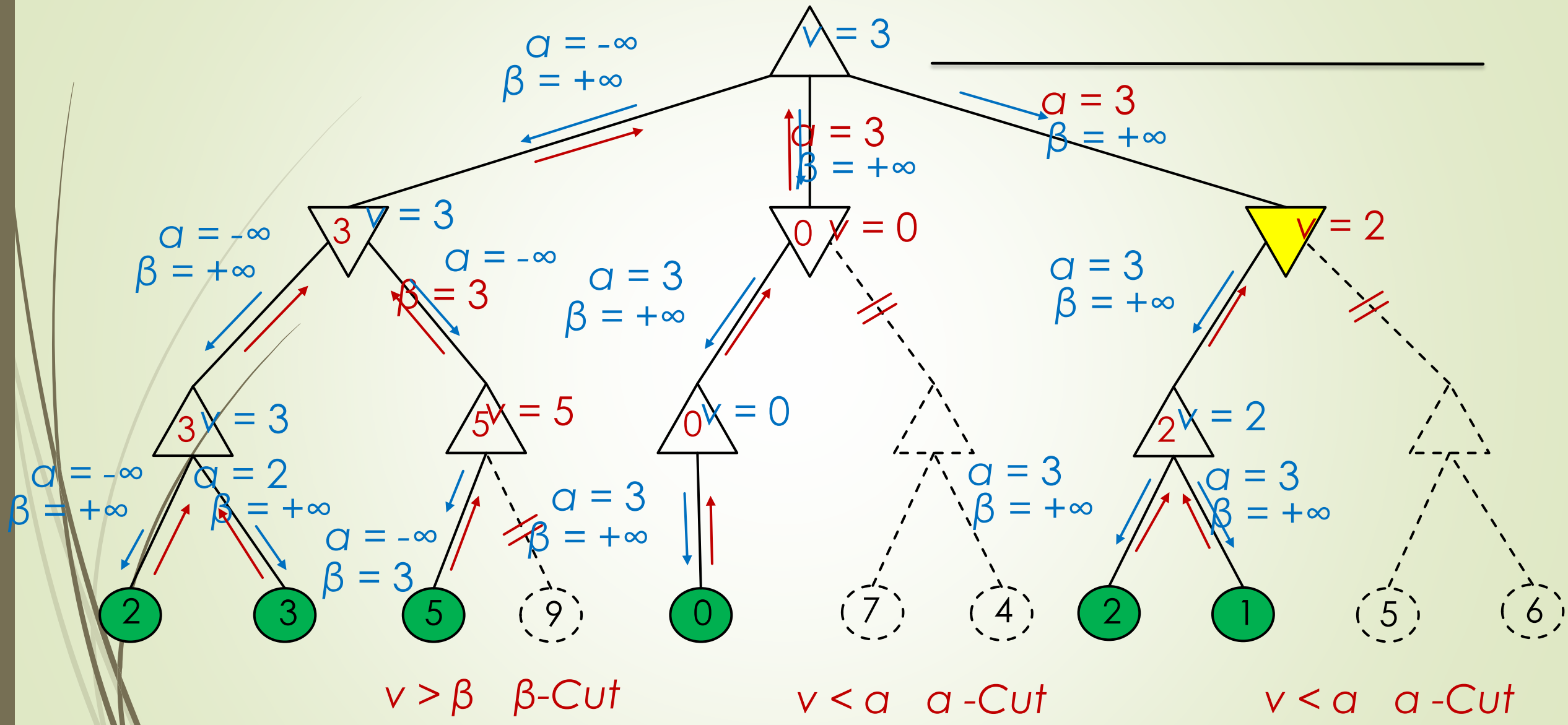
آلفا– بتا (مثال ۲)



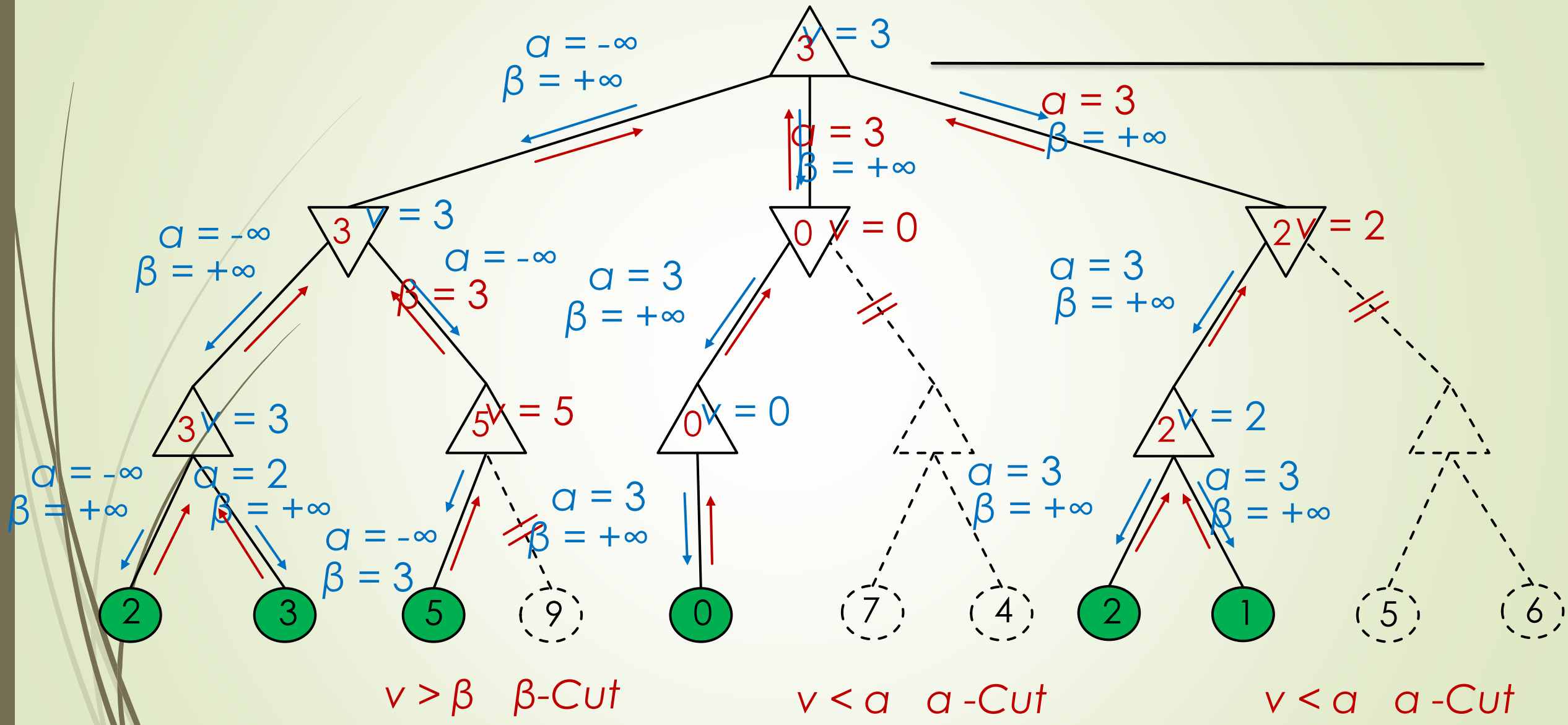
آلفا– بتا (مثال ۲)



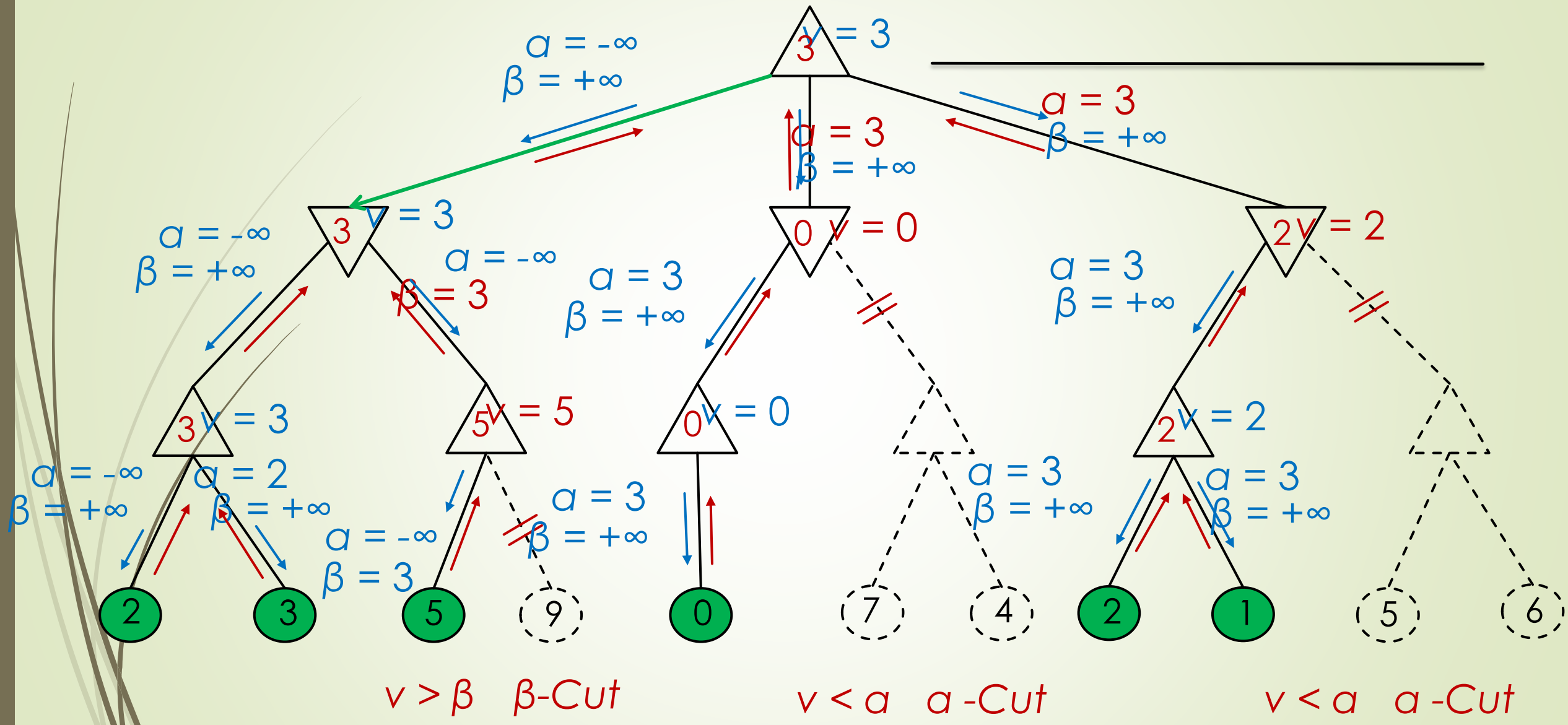
آلفا– بتا (مثال ۲)



آلفا– بتا (مثال ۲)

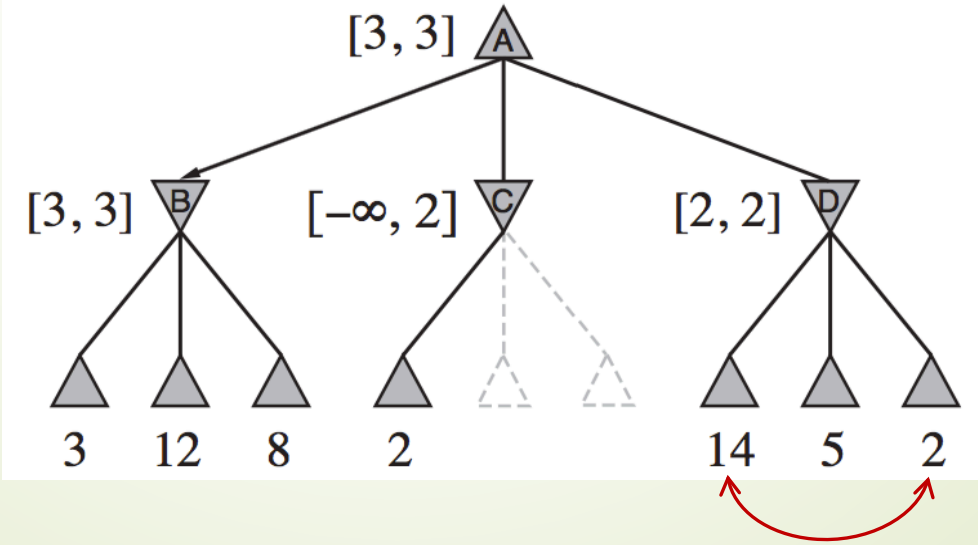


آلفا– بتا (مثال ۲)



ویژگی‌های الگوریتم آلفا–بتا

- ترتیب بررسی شاخه‌های درخت بر روی تعداد شاخه‌های هرس‌شده تاثیر می‌گذارد.
- برای بازی‌هایی که سودمندی گره‌ها از ∞- تا ∞+ 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.



تست

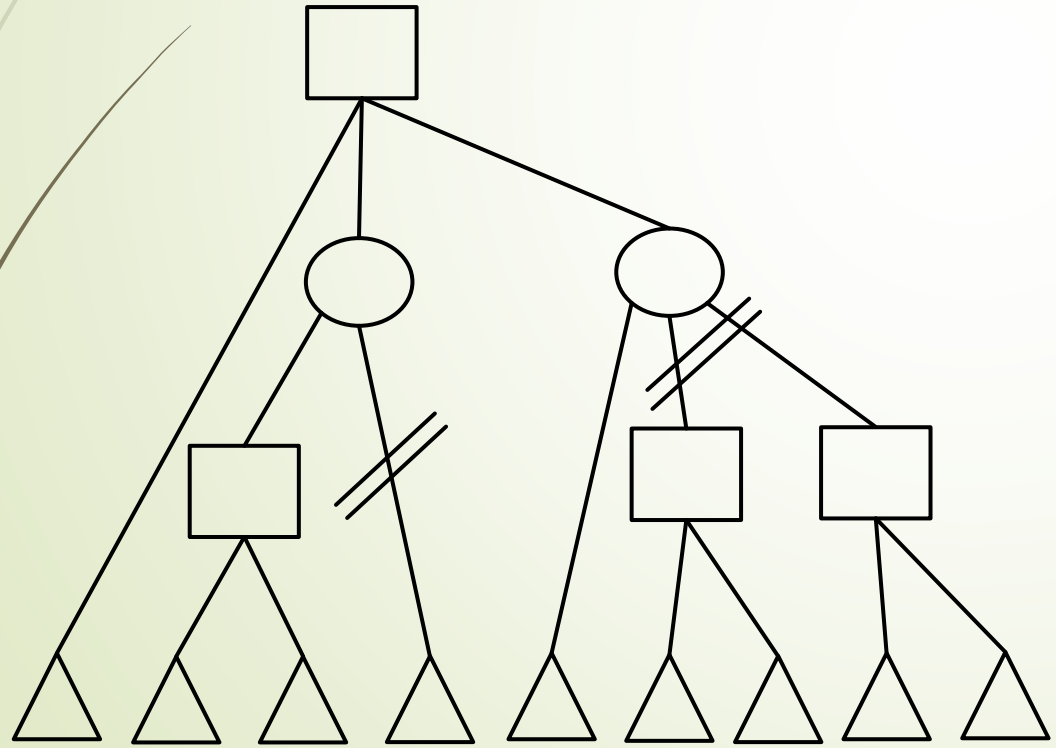
در گراف مقابل مربع نشانه بازیکن Max، دایره نشانه بازیکن Min و مثلث نشانه حالت پایانی است. اگر مقادیر ارزیابی بتوانند در فاصله بسته [0,10] باشند و با هرس آلفا-بتا فقط یال‌های علامت زده شده // حذف شوند، ترتیب گره‌های پایانی به‌ترتیب از چپ به راست در شکل کدام یک از گزینه‌های زیر خواهد بود؟

۱) ۴، ۱، ۰، ۱۰، ۹، ۳، ۵، ۲، ۸

۲) ۰، ۱، ۲، ۳، ۴، ۵، ۸، ۹، ۱۰

۳) ۴، ۲، ۱، ۰، ۸، ۱۰، ۳، ۵، ۹

۴) ۱۰، ۹، ۸، ۹، ۴، ۳، ۲، ۱، ۰



تست

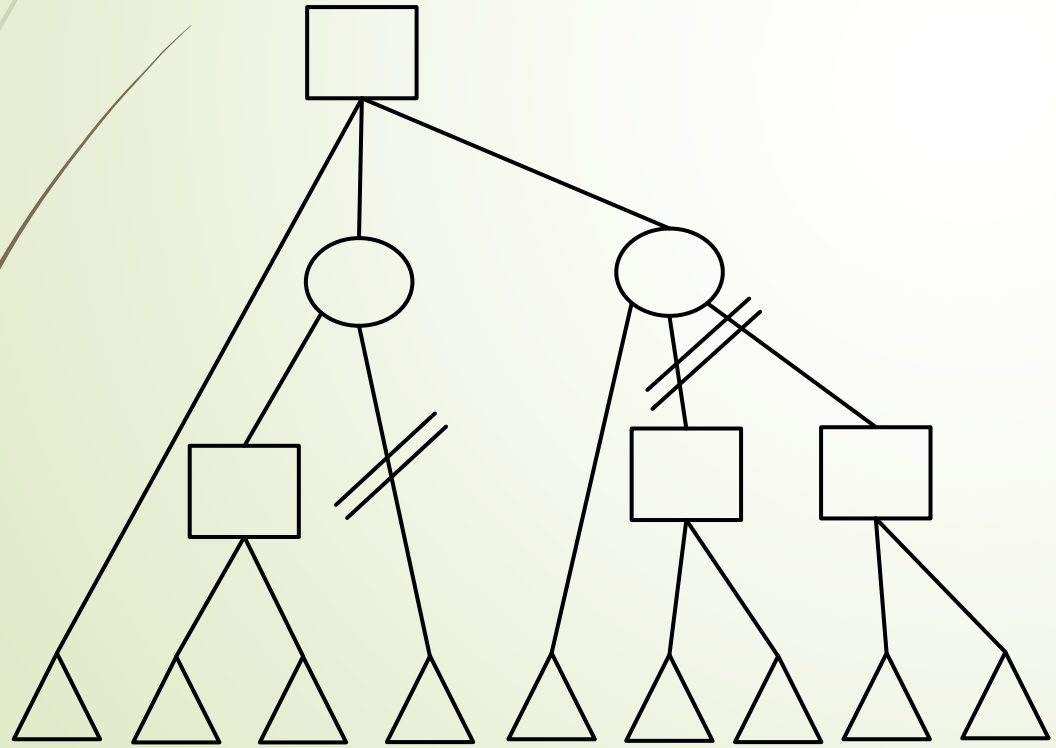
در گراف مقابل مربع نشانه بازیکن Max، دایره نشانه بازیکن Min و مثلث نشانه حالت پایانی است. اگر مقادیر ارزیابی بتوانند در فاصله بسته [0,10] باشند و با هرس آلفا-بتا فقط یال‌های علامت زده شده // حذف شوند، ترتیب گره‌های پایانی به‌ترتیب از چپ به راست در شکل کدام یک از گزینه‌های زیر خواهد بود؟

۱) ۴، ۱، ۰، ۱۰، ۹، ۳، ۵، ۲، ۸

۲) ۰، ۱، ۲، ۳، ۴، ۵، ۸، ۹، ۱۰

✓ ۳) ۴، ۲، ۱، ۰، ۸، ۱۰، ۳، ۵، ۹

۴) ۱۰، ۹، ۸، ۹، ۴، ۳، ۲، ۱، ۰



بازه ممکن [0,10] است. بنابراین در گزینه ۲ به محض رویت ۱۰ در چپ ترین برگ، الگوریتم خاتمه می یابد و زیر شاخه های دوم و سوم ریشه درخت نیز هرس می‌شوند