Algorytm α-β i MTD(f)

dr inż. Piotr Beling

Uniwersytet Łódzki

2020 (aktualizacja: 2021)

http://pbeling.w8.pl

Algorytm α-β – idea

- α-β to metoda obcinania gałęzi w drzewach minimaksowych.
- Załóżmy, że w pozycji P ruch należy do gracza *Max*
- i wiadomo, że wartość jednego z następników P wynosi α;
- zatem wartość P to co najmniej α
- i do jej wyznaczenia nie potrzebujemy znać dokładnych wartość następników o wartościach nie większych od α.
- Analogicznie, jeżeli w pozycji P ruch należy do gracza *Min*, to nie potrzebujemy znać dokładnych wartość następników P nie mniejszych od dotychczas znalezionego minimum β.
- W ten sposób otrzymujemy i następnie, idąc w głąb drzewa poszukiwań, zawężamy przedział (zwany też „oknem") wartości (α, β).
- Przeszukiwanie węzła można przerwać gdy dowiedziemy, że jego ostateczna wartość nie zmieści się w oknie (α, β) i nie przejdzie na jego drugą stronę.
- Przedział (α, β) jest zawsze obustronnie otwarty.

Algorytm α-β – idea

- α-β to metoda obcinania gałęzi w drzewach minimaksowych.
- Załóżmy, że w pozycji P ruch należy do gracza *Max*
- i wiadomo, że wartość jednego z następników P wynosi α;
- zatem wartość P to co najmniej α
- i do jej wyznaczenia nie potrzebujemy znać dokładnych wartość następników o wartościach nie większych od α.
- Analogicznie, jeżeli w pozycji P ruch należy do gracza *Min*, to nie potrzebujemy znać dokładnych wartość następników P nie mniejszych od dotychczas znalezionego minimum β.
- W ten sposób otrzymujemy i następnie, idąc w głąb drzewa poszukiwań, zawężamy przedział (zwany też „oknem") wartości (α, β).
- Przeszukiwanie węzła można przerwać gdy dowiedziemy, że jego ostateczna wartość nie zmieści się w oknie (α, β) i nie przejdzie na jego drugą stronę.
- Przedział (α, β) jest zawsze obustronnie otwarty.

Algorytm α-β – idea

- α-β to metoda obcinania gałęzi w drzewach minimaksowych.
- Załóżmy, że w pozycji P ruch należy do gracza *Max*
- i wiadomo, że wartość jednego z następników P wynosi α;
- zatem wartość P to co najmniej α
- i do jej wyznaczenia nie potrzebujemy znać dokładnych wartość następników o wartościach nie większych od α.
- Analogicznie, jeżeli w pozycji P ruch należy do gracza *Min*, to nie potrzebujemy znać dokładnych wartość następników P nie mniejszych od dotychczas znalezionego minimum β.
- W ten sposób otrzymujemy i następnie, idąc w głąb drzewa poszukiwań, zawężamy przedział (zwany też „oknem") wartości (α, β).
- Przeszukiwanie węzła można przerwać gdy dowiedziemy, że jego ostateczna wartość nie zmieści się w oknie (α, β) i nie przejdzie na jego drugą stronę.
- Przedział (α, β) jest zawsze obustronnie otwarty.

Algorytm α-β – idea

- α-β to metoda obcinania gałęzi w drzewach minimaksowych.
- Załóżmy, że w pozycji P ruch należy do gracza *Max*
- i wiadomo, że wartość jednego z następników P wynosi α;
- zatem wartość P to co najmniej α
- i do jej wyznaczenia nie potrzebujemy znać dokładnych wartość następników o wartościach nie większych od α.
- Analogicznie, jeżeli w pozycji P ruch należy do gracza *Min*, to nie potrzebujemy znać dokładnych wartość następników P nie mniejszych od dotychczas znalezionego minimum β.
- W ten sposób otrzymujemy i następnie, idąc w głąb drzewa poszukiwań, zawężamy przedział (zwany też „oknem") wartości (α, β).
- Przeszukiwanie węzła można przerwać gdy dowiedziemy, że jego ostateczna wartość nie zmieści się w oknie (α, β) i nie przejdzie na jego drugą stronę.
- Przedział (α, β) jest zawsze obustronnie otwarty.

Algorytm α-β – idea

- α-β to metoda obcinania gałęzi w drzewach minimaksowych.
- Załóżmy, że w pozycji P ruch należy do gracza *Max*
- i wiadomo, że wartość jednego z następników P wynosi α;
- zatem wartość P to co najmniej α
- i do jej wyznaczenia nie potrzebujemy znać dokładnych wartość następników o wartościach nie większych od α.
- Analogicznie, jeżeli w pozycji P ruch należy do gracza *Min*, to nie potrzebujemy znać dokładnych wartość następników P nie mniejszych od dotychczas znalezionego minimum β.
- W ten sposób otrzymujemy i następnie, idąc w głąb drzewa poszukiwań, zawężamy przedział (zwany też „oknem") wartości (α, β).
- Przeszukiwanie węzła można przerwać gdy dowiedziemy, że jego ostateczna wartość nie zmieści się w oknie (α, β) i nie przejdzie na jego drugą stronę.
- Przedział (α, β) jest zawsze obustronnie otwarty.

Algorytm α-β – idea

- α-β to metoda obcinania gałęzi w drzewach minimaksowych.
- Załóżmy, że w pozycji P ruch należy do gracza *Max*
- i wiadomo, że wartość jednego z następników P wynosi α;
- zatem wartość P to co najmniej α
- i do jej wyznaczenia nie potrzebujemy znać dokładnych wartość następników o wartościach nie większych od α.
- Analogicznie, jeżeli w pozycji P ruch należy do gracza *Min*, to nie potrzebujemy znać dokładnych wartość następników P nie mniejszych od dotychczas znalezionego minimum β.
- W ten sposób otrzymujemy i następnie, idąc w głąb drzewa poszukiwań, zawężamy przedział (zwany też „oknem") wartości (α, β).
- Przeszukiwanie węzła można przerwać gdy dowiedziemy, że jego ostateczna wartość nie zmieści się w oknie (α, β) i nie przejdzie na jego drugą stronę.
- Przedział (α, β) jest zawsze obustronnie otwarty.

Algorytm α-β – idea

- α-β to metoda obcinania gałęzi w drzewach minimaksowych.
- Załóżmy, że w pozycji P ruch należy do gracza *Max*
- i wiadomo, że wartość jednego z następników P wynosi α;
- zatem wartość P to co najmniej α
- i do jej wyznaczenia nie potrzebujemy znać dokładnych wartość następników o wartościach nie większych od α.
- Analogicznie, jeżeli w pozycji P ruch należy do gracza *Min*, to nie potrzebujemy znać dokładnych wartość następników P nie mniejszych od dotychczas znalezionego minimum β.
- W ten sposób otrzymujemy i następnie, idąc w głąb drzewa poszukiwań, zawężamy przedział (zwany też „oknem") wartości (α, β).
- Przeszukiwanie węzła można przerwać gdy dowiedziemy, że jego ostateczna wartość nie zmieści się w oknie (α, β) i nie przejdzie na jego drugą stronę.
- Przedział (α, β) jest zawsze obustronnie otwarty.

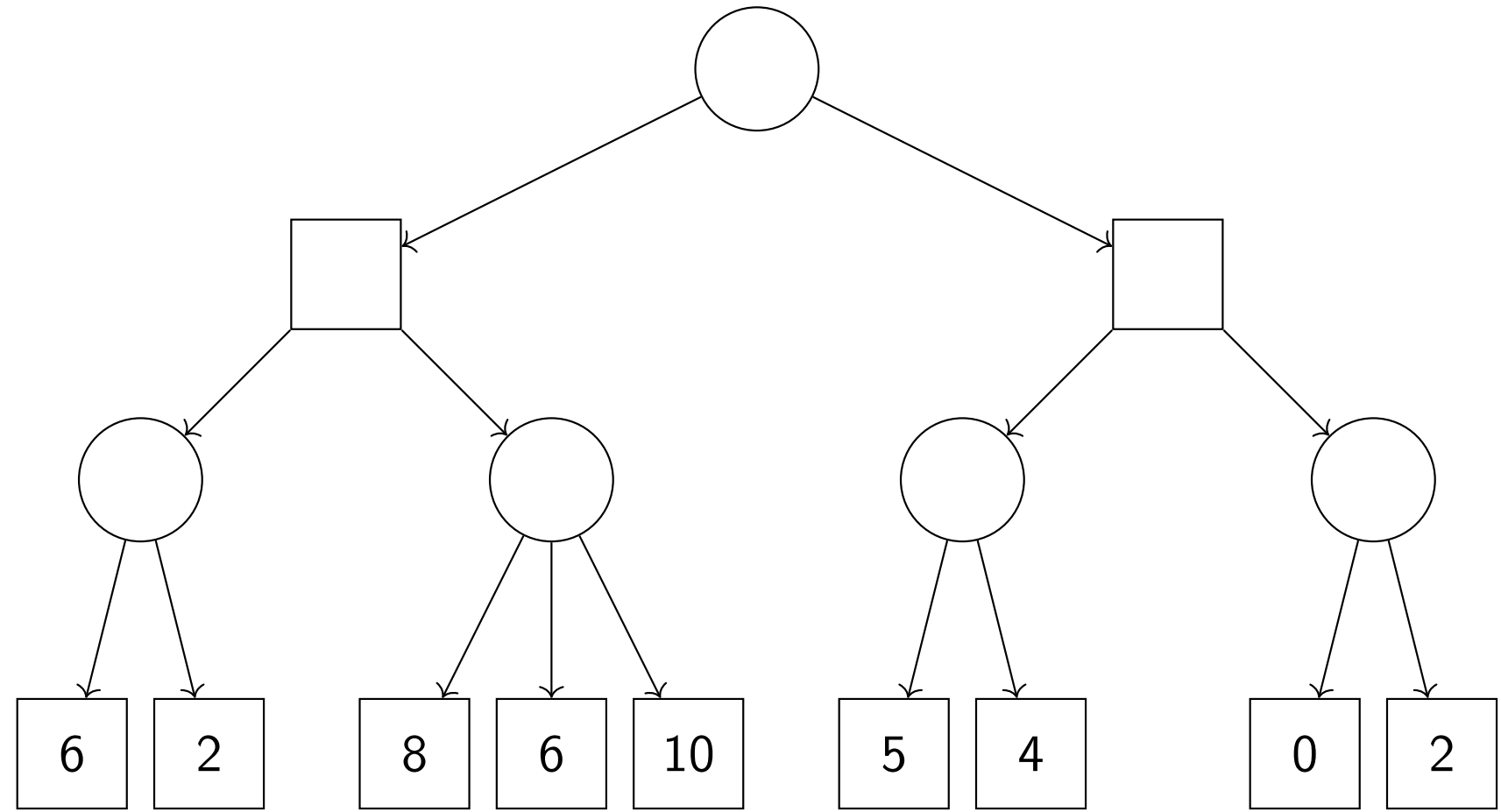
Algorytm α-β – idea

- α-β to metoda obcinania gałęzi w drzewach minimaksowych.
- Załóżmy, że w pozycji P ruch należy do gracza *Max*
- i wiadomo, że wartość jednego z następników P wynosi α;
- zatem wartość P to co najmniej α
- i do jej wyznaczenia nie potrzebujemy znać dokładnych wartość następników o wartościach nie większych od α.
- Analogicznie, jeżeli w pozycji P ruch należy do gracza *Min*, to nie potrzebujemy znać dokładnych wartość następników P nie mniejszych od dotychczas znalezionego minimum β.
- W ten sposób otrzymujemy i następnie, idąc w głąb drzewa poszukiwań, zawężamy przedział (zwany też „oknem") wartości (α, β).
- Przeszukiwanie węzła można przerwać gdy dowiedziemy, że jego ostateczna wartość nie zmieści się w oknie (α, β) i nie przejdzie na jego drugą stronę.
- Przedział (α, β) jest zawsze obustronnie otwarty.

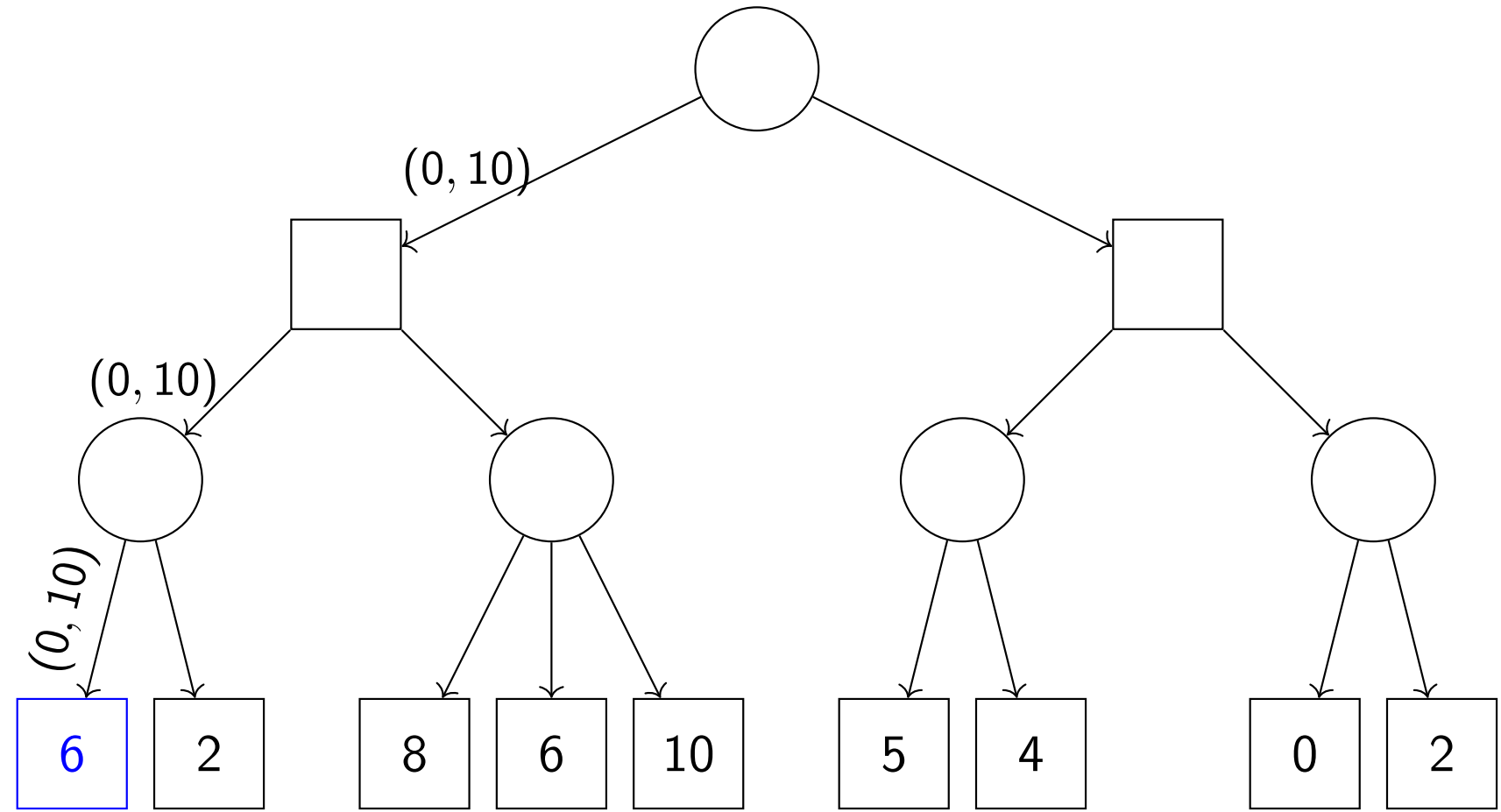
Algorytm α-β – idea

- α-β to metoda obcinania gałęzi w drzewach minimaksowych.
- Załóżmy, że w pozycji P ruch należy do gracza *Max*
- i wiadomo, że wartość jednego z następników P wynosi α;
- zatem wartość P to co najmniej α
- i do jej wyznaczenia nie potrzebujemy znać dokładnych wartość następników o wartościach nie większych od α.
- Analogicznie, jeżeli w pozycji P ruch należy do gracza *Min*, to nie potrzebujemy znać dokładnych wartość następników P nie mniejszych od dotychczas znalezionego minimum β.
- W ten sposób otrzymujemy i następnie, idąc w głąb drzewa poszukiwań, zawężamy przedział (zwany też „oknem") wartości (α, β).
- Przeszukiwanie węzła można przerwać gdy dowiedziemy, że jego ostateczna wartość nie zmieści się w oknie (α, β) i nie przejdzie na jego drugą stronę.
- Przedział (α, β) jest zawsze obustronnie otwarty.

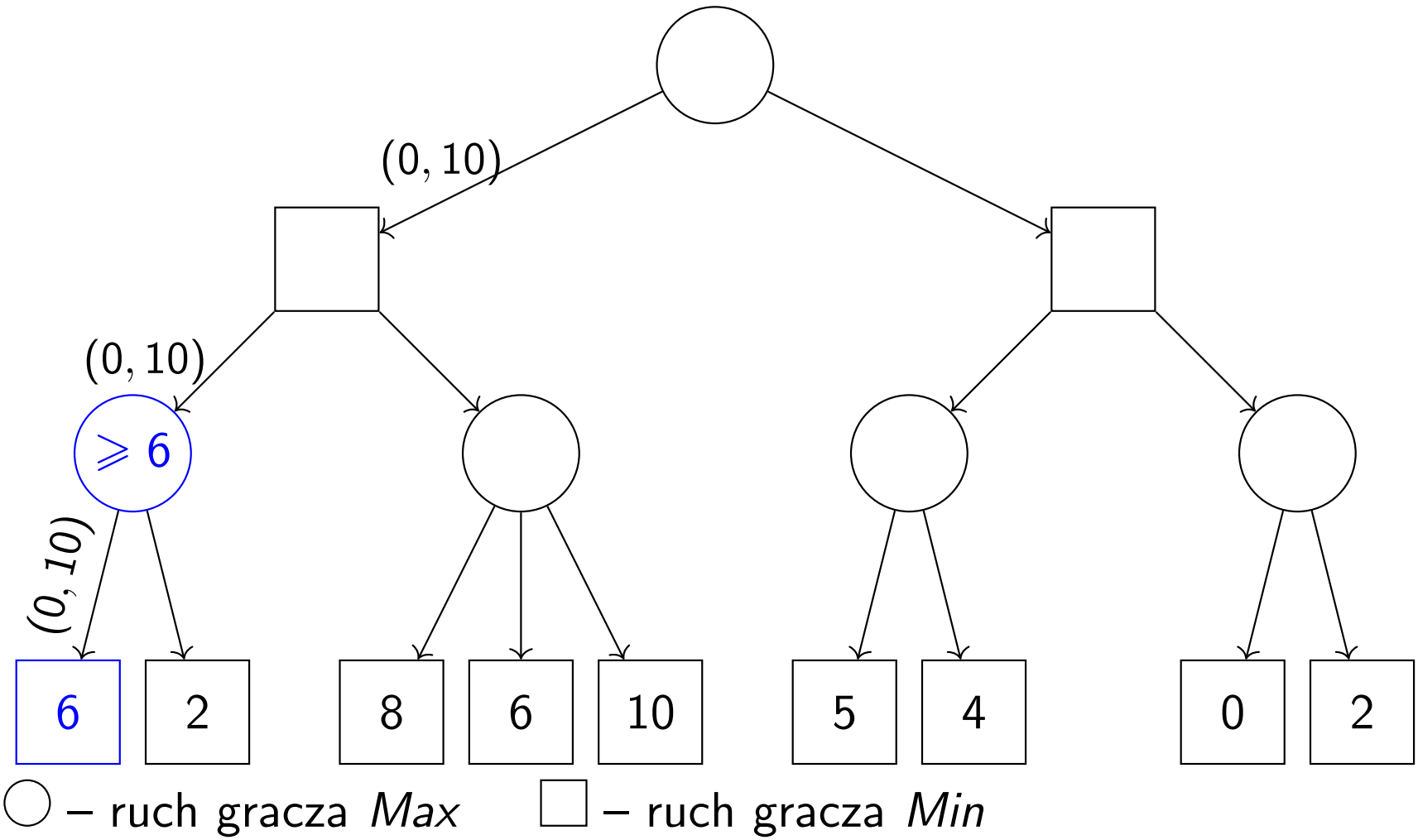
Przykład: Prześledźmy działanie, wywołanego z oknem $(0, 10)$ algorytmu α-β na następującym grafie gry o możliwych wypłatach od 0 do 10 oraz sumie wypłat 10:



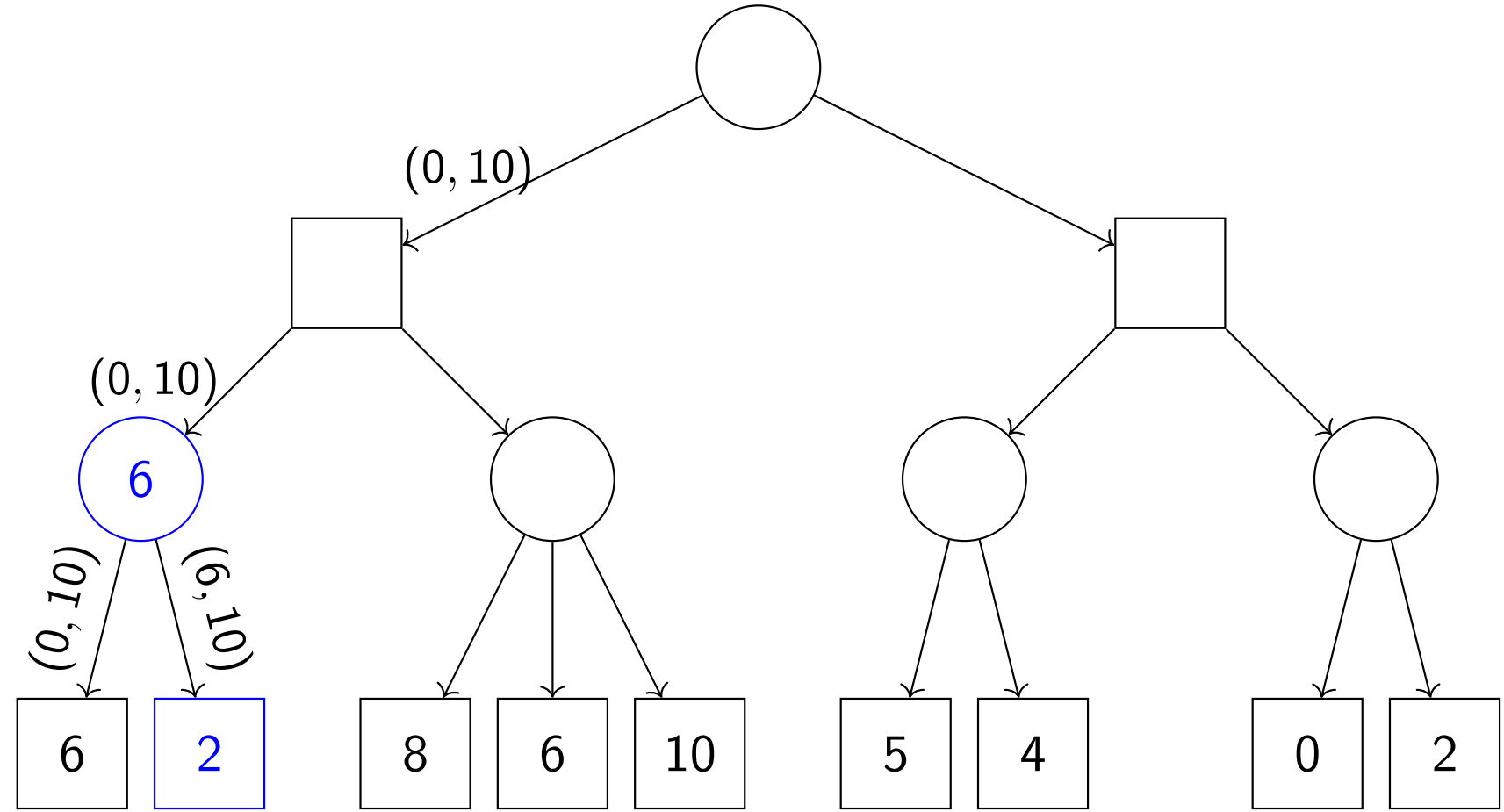
Przy użyciu początkowego okna $(0, 10)$, następują rekurencyjne wywołania dla kolejnych, pierwszych (od lewej) następników, aż do osiągnięcia pozycji końcowej o wartości 6.



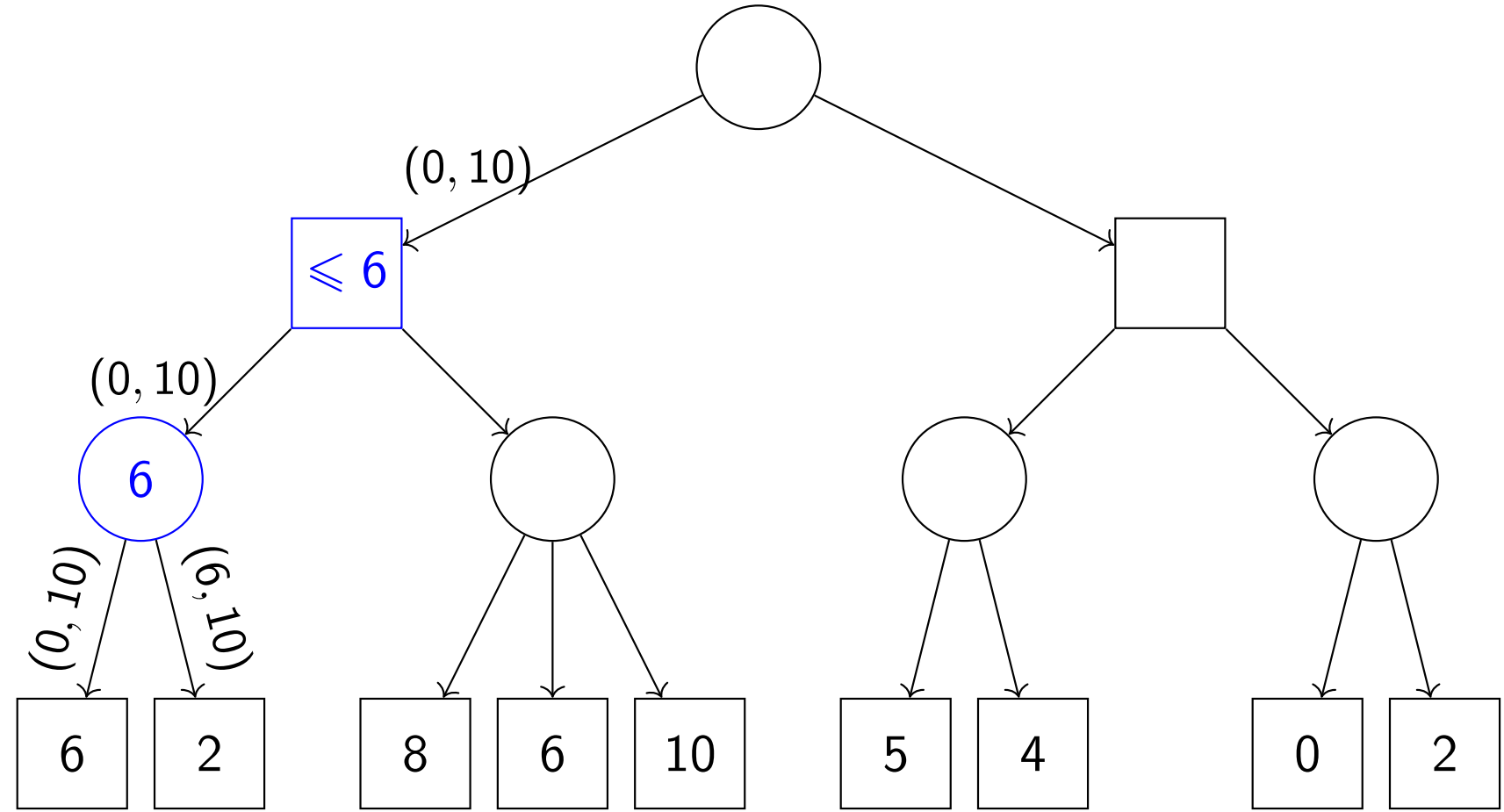
6 staje się bieżącą wartością poprzednika osiągniętej pozycji końcowej. Poprzednik ten jest typu MAX, więc jego ostateczna wartość wyniesie 6 lub więcej. Nie można odciąć więc jego drugiego następnika, bo wartość $\geqslant 6$ może się zmieścić w oknie $(0, 10)$, tj. $[6, \infty) \cap (0, 10) \neq \emptyset$.



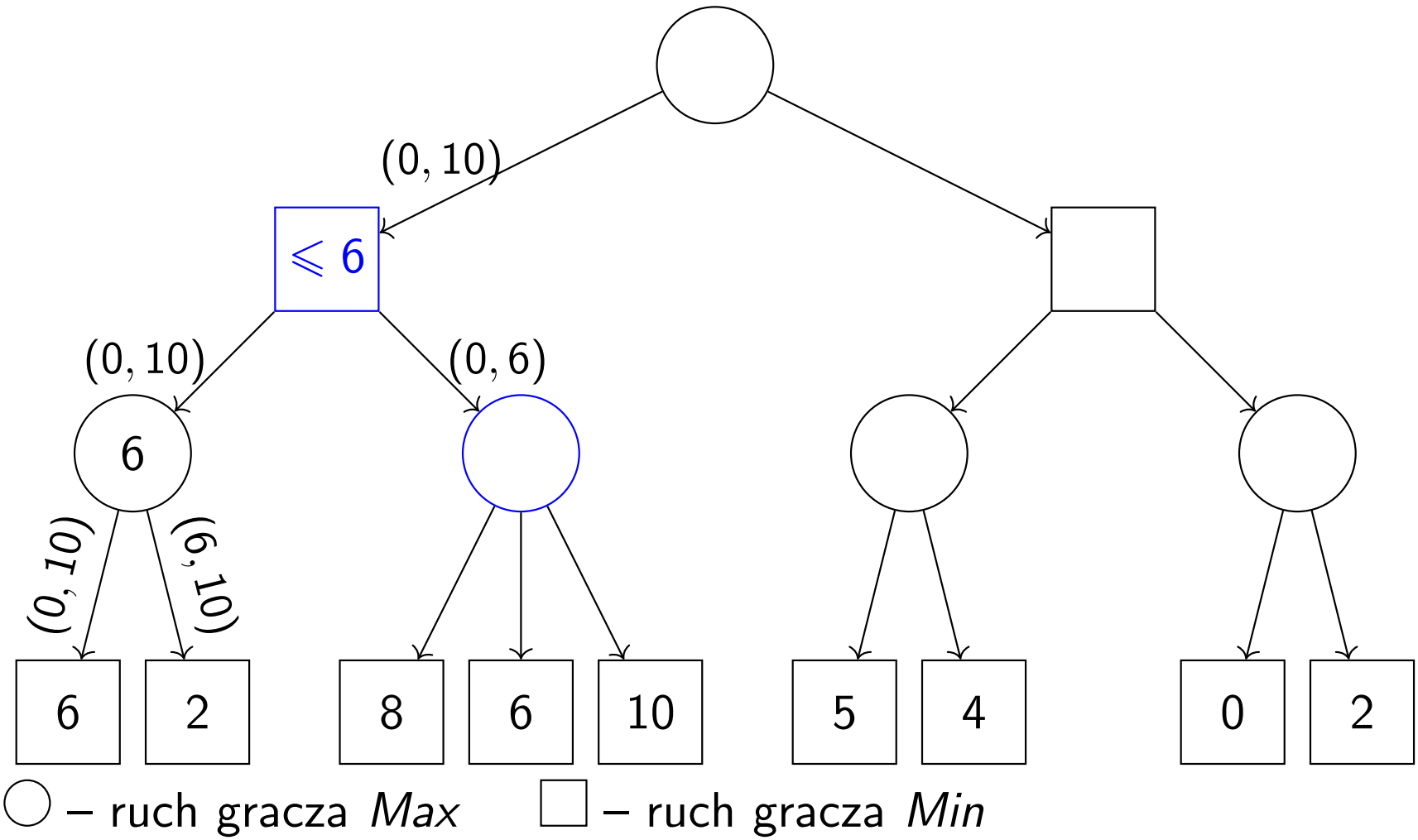
Dla drugiego następnika algorytm wywołuje się w węższym oknie $(6, \infty) \cap (0, 10) = (6, 10)$, bo wartości $\leqslant 6$ nie mogłyby poprawić bieżącego maksimum 6, nie są więc interesujące. 2 nie poprawia bieżącego maksimum. Nie ma już więcej następników, więc 6 jest ostateczną wartością pozycji.



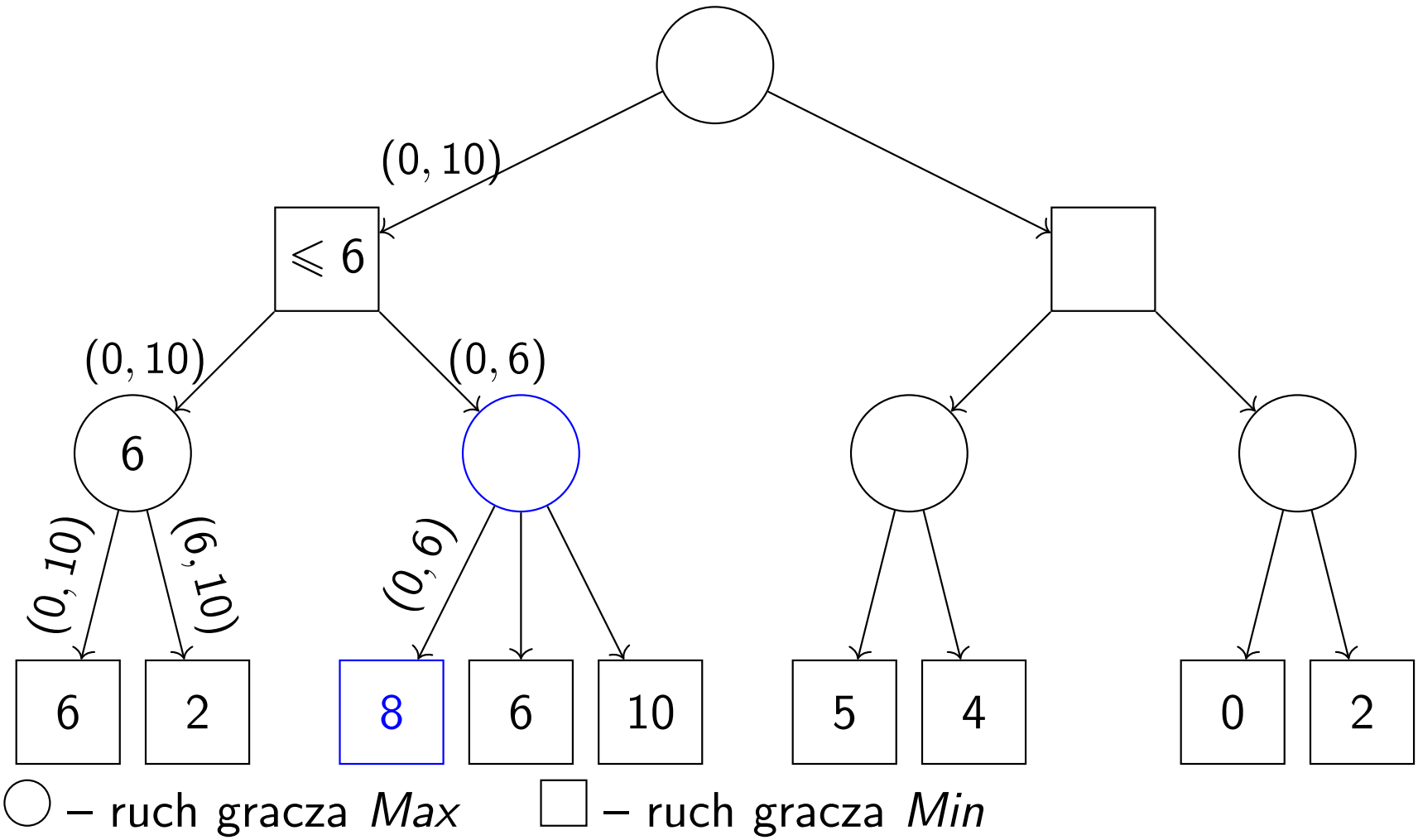
6 staje się bieżącą wartością poprzednika zbadanej pozycji. Poprzednik ten jest typu MIN, więc jego ostateczna wartość wyniesie 6 lub mniej. Nie można odciąć więc jego drugiego następnika, bo wartość $\leqslant 6$ może się zmieścić w oknie $(0, 10)$, tj. $(-\infty, 6] \cap (0, 10) \neq \emptyset$.



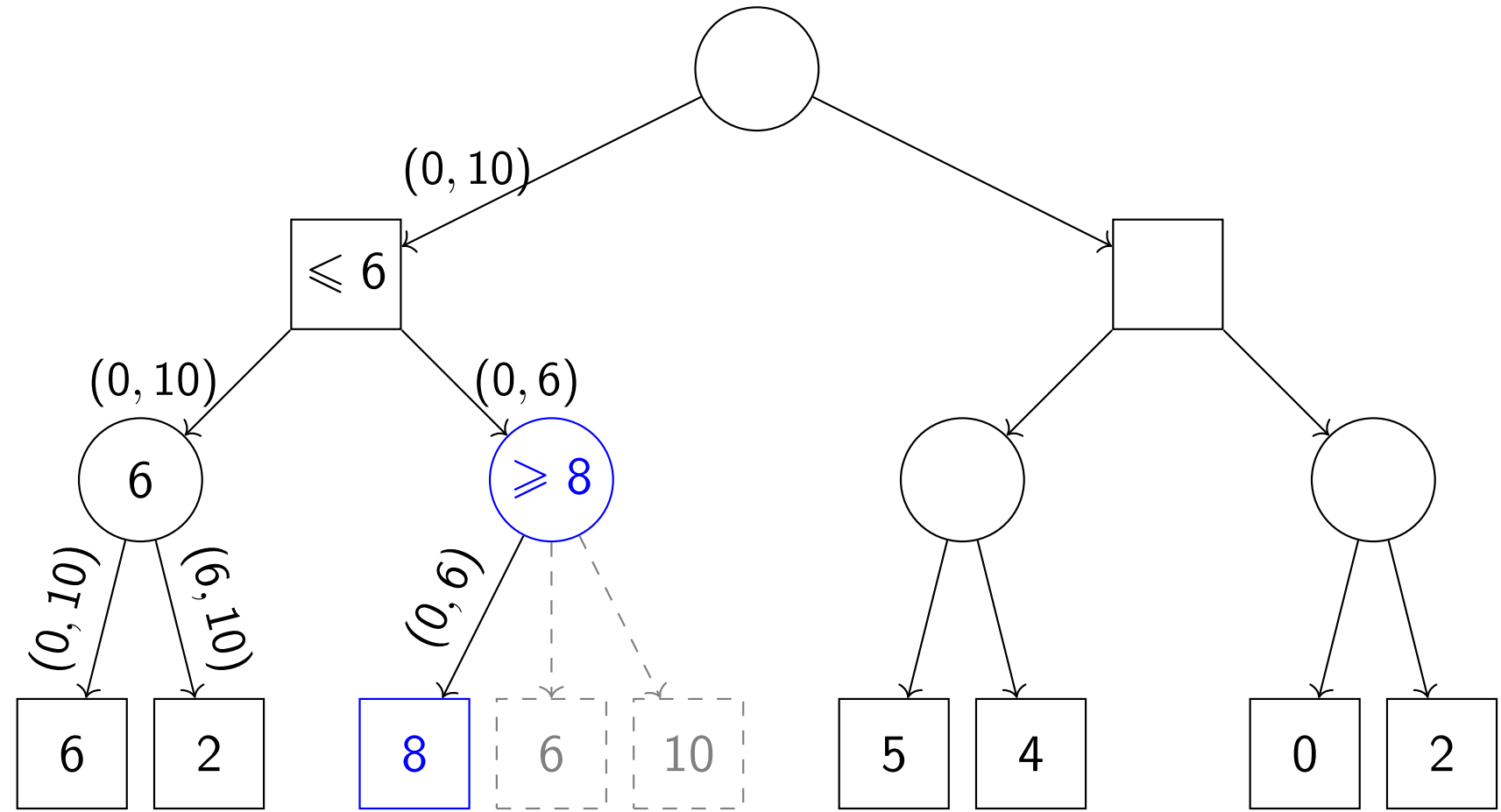
Dla drugiego następnika algorytm wywołuje się w oknie $(-\infty, 6) \cap (0, 10) = (0, 6)$, bo wartości $\geqslant 6$ nie mogłyby poprawić bieżącego minimum 6, nie są więc interesujące.



Pierwszy następnik badany jest w tym samym oknie co jego poprzednik, czyli $(0, 6)$. Jego wartość 8 staje się bieżącym maksimum poprzednika, którego ostateczna wartość wyniesie zatem $\geqslant 8$.

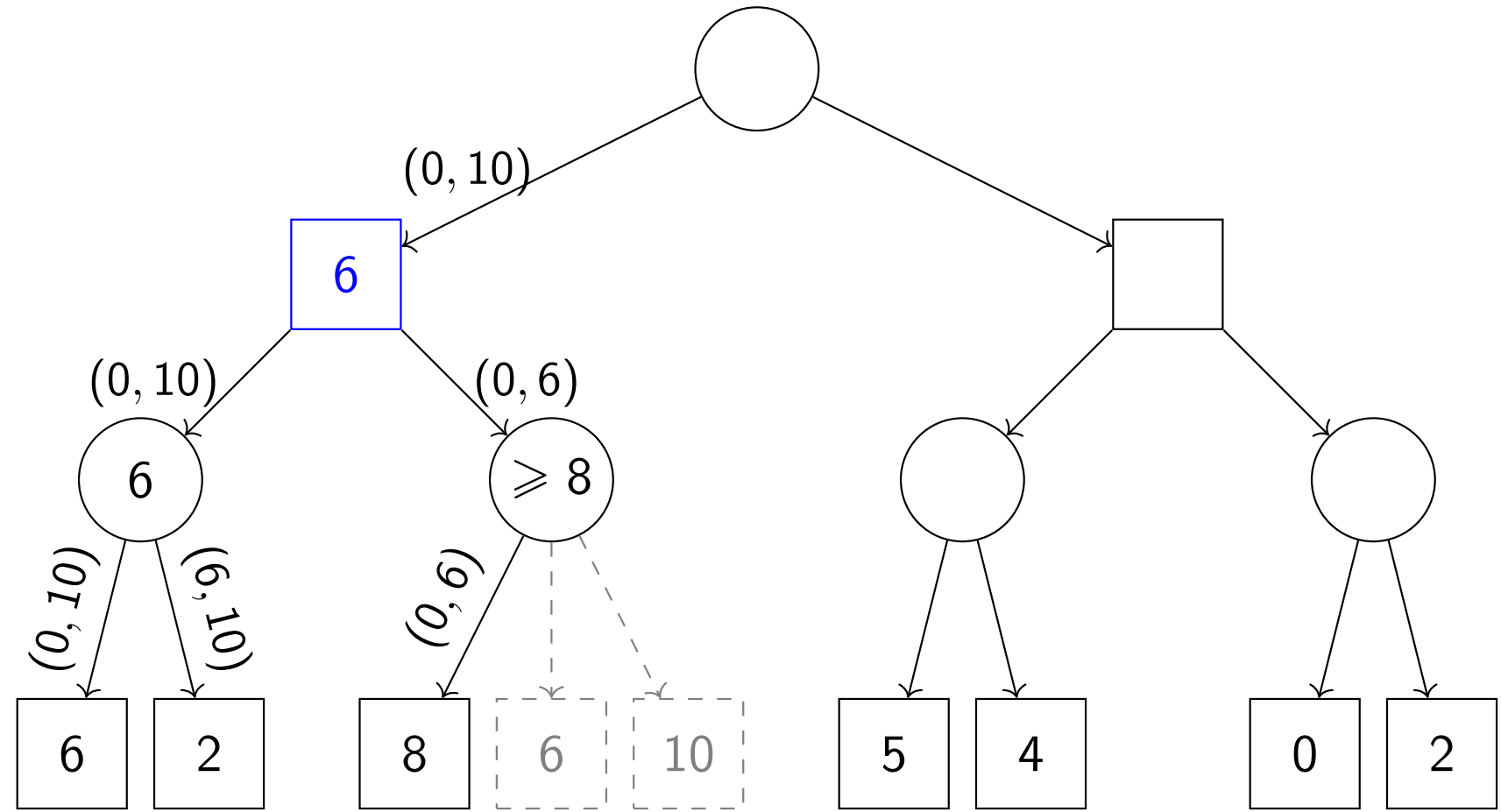


Ta ostateczna wartość $\geqslant 8$ nie zmieści się w oknie $(0, 6)$, tj. $[8, \infty) \cap (0, 6) = \emptyset$. Następuje więc odcięcie kolejnych następników. Nie są one w ogóle badane (ich wartości nie wpłyną na wartość korzenia). Bieżące maksimum, czyli 8, staje się ostateczną wartością (zwróconą z) zbadanego węzła.

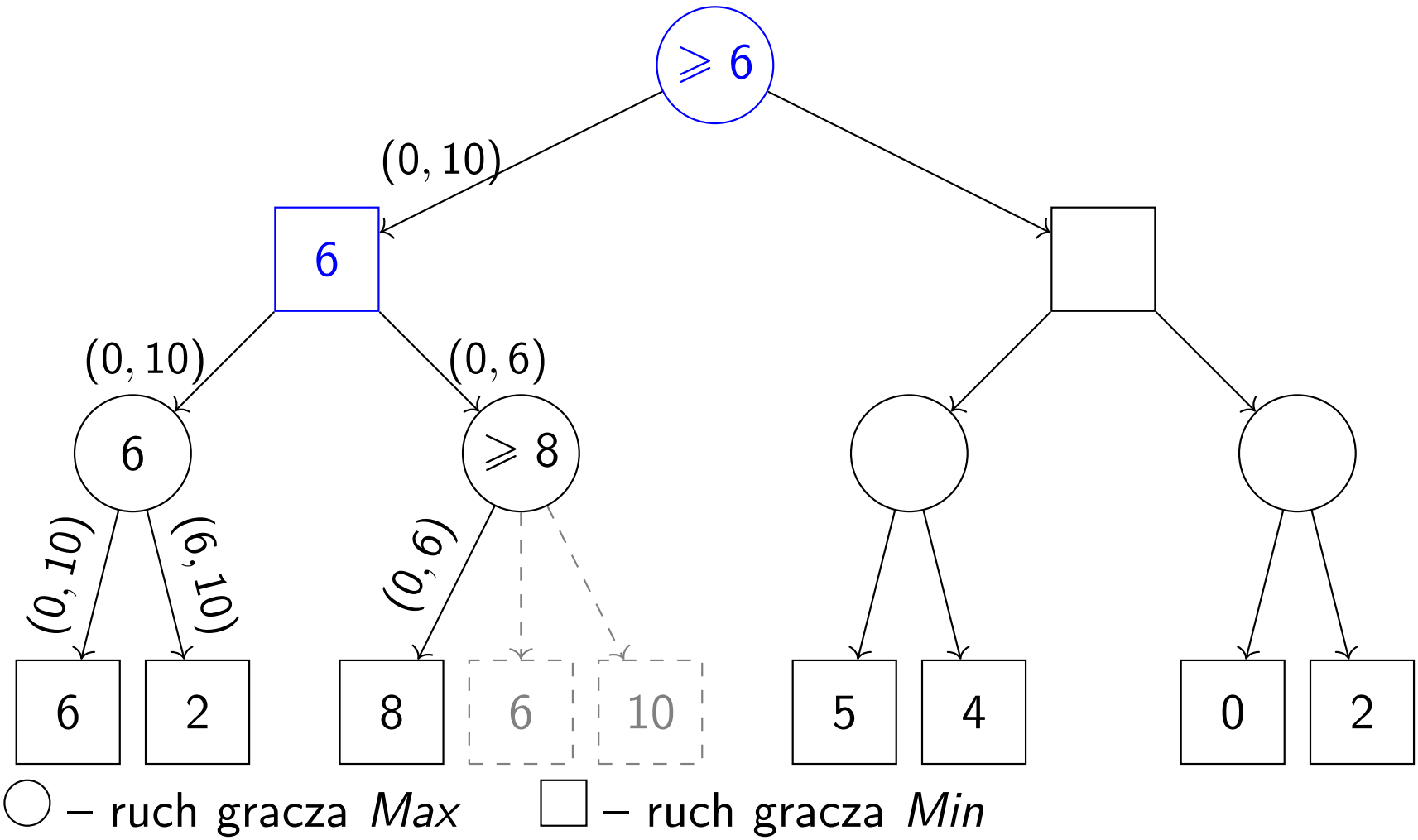


○ – ruch gracza *Max* □ – ruch gracza *Min*

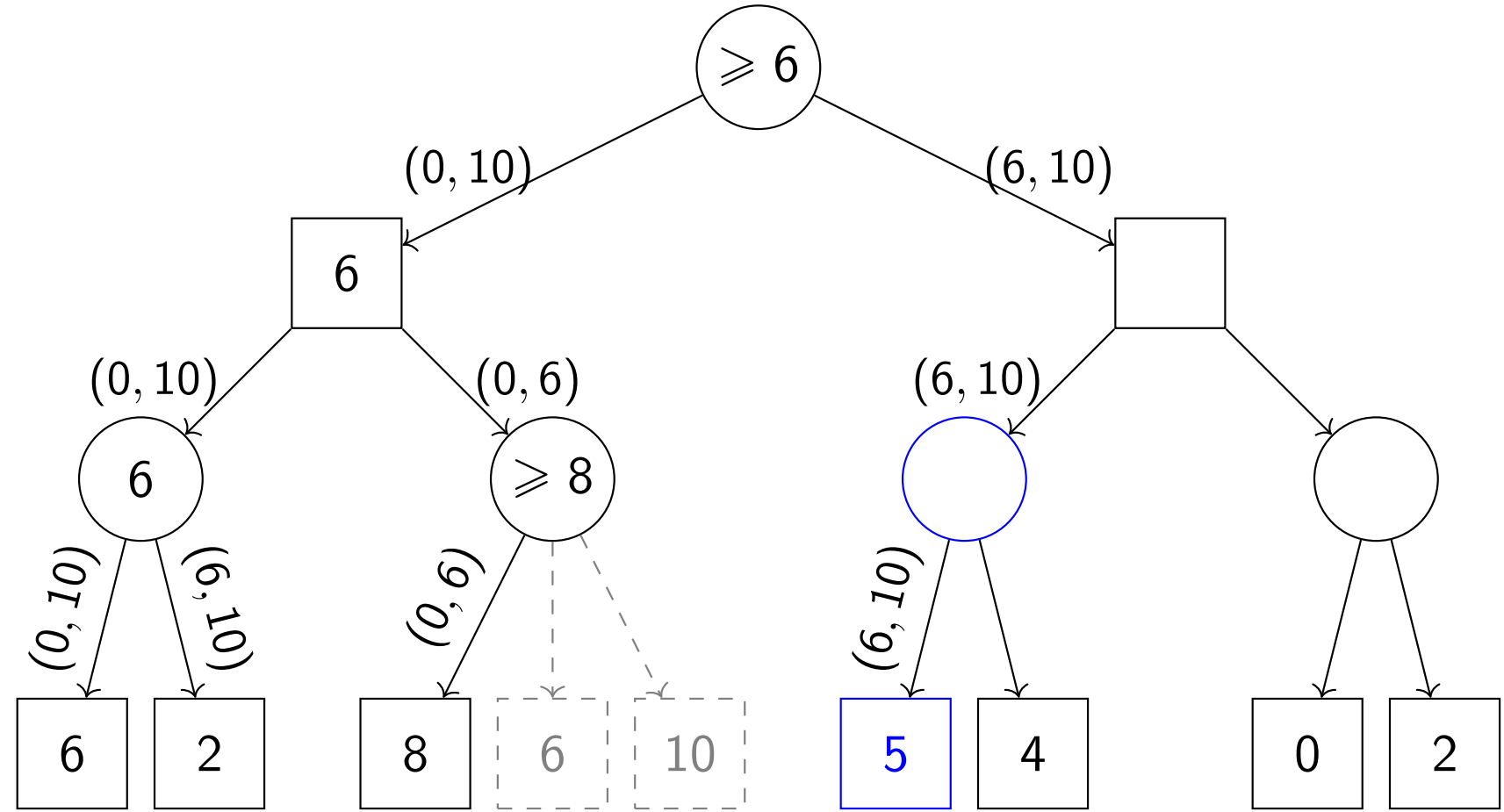
Ta 8 nie poprawia bieżącego minimum 6 lewego następnika korzenia. 6 staje się zatem jego ostateczną wartością, bo nie ma on więcej dzieci.



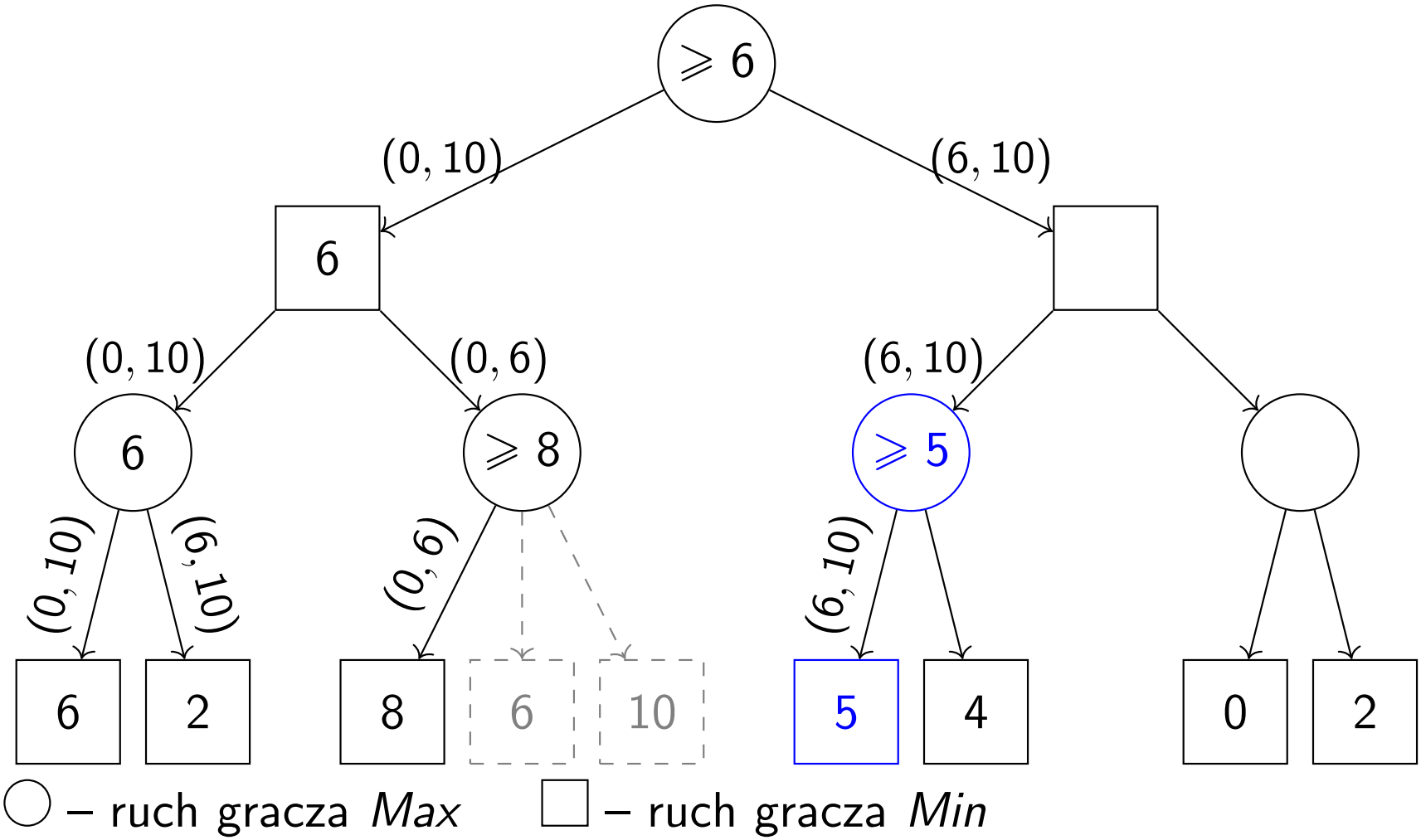
6 staje się bieżącą wartością korzenia. Korzeń jest typu MAX, więc jego ostateczna wartość wyniesie 6 lub więcej. Nie można odciąć jego prawego następnika, bo wartość $\geqslant 6$ może się zmieścić w oknie $(0, 10)$, tj. $[6, \infty) \cap (0, 10) \neq \emptyset$.



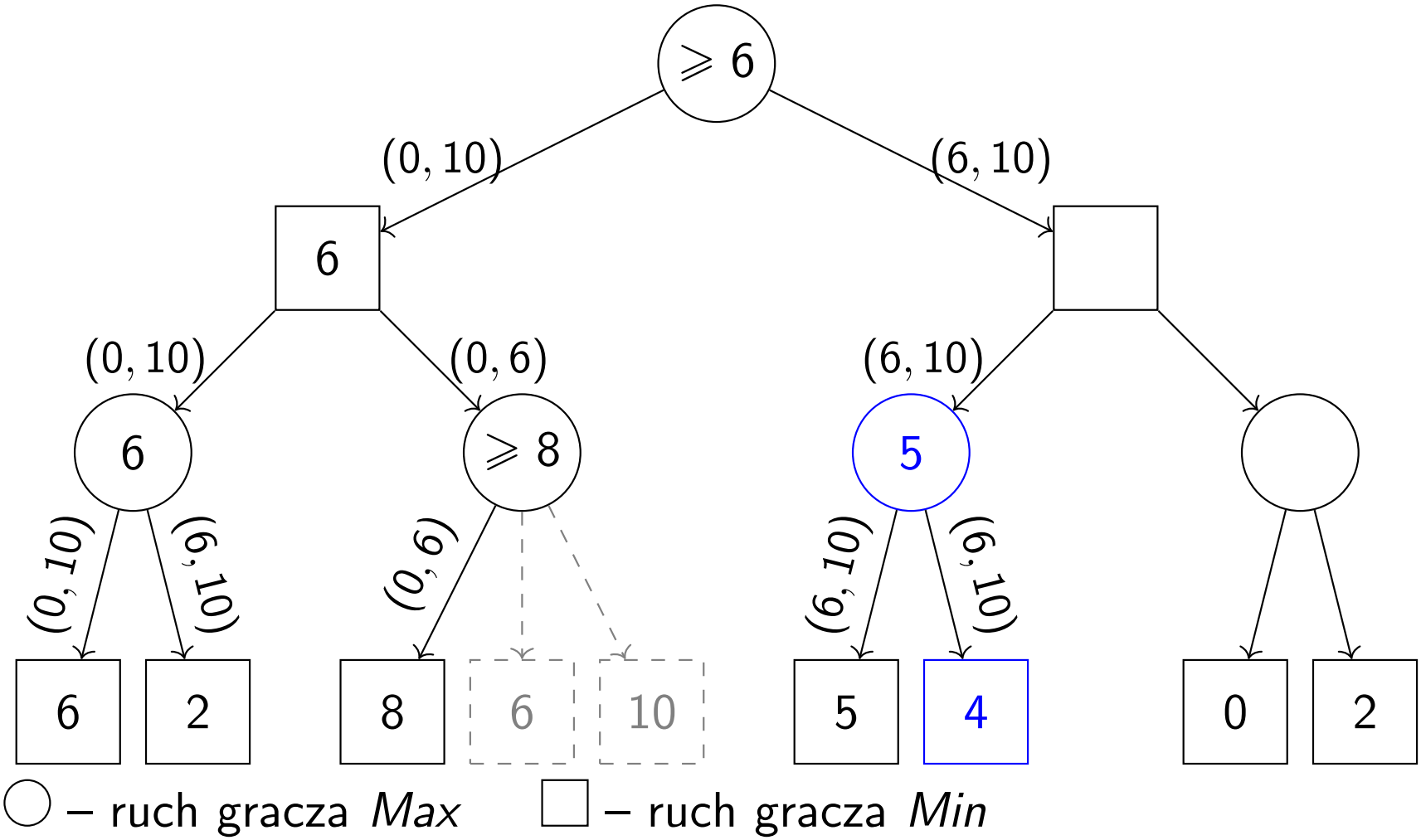
Dla prawego następnika korzenia algorytm wywołuje się w oknie $(6, \infty) \cap (0, 10) = (6, 10)$, bo wartości $\leqslant 6$ nie mogłyby poprawić bieżącego maksimum 6, nie są więc interesujące. Dalej następują rekurencyjne wywołania dla kolejnych, pierwszych (od lewej) następników, aż do osiągnięcia pozycji końcowej o wartości 5.



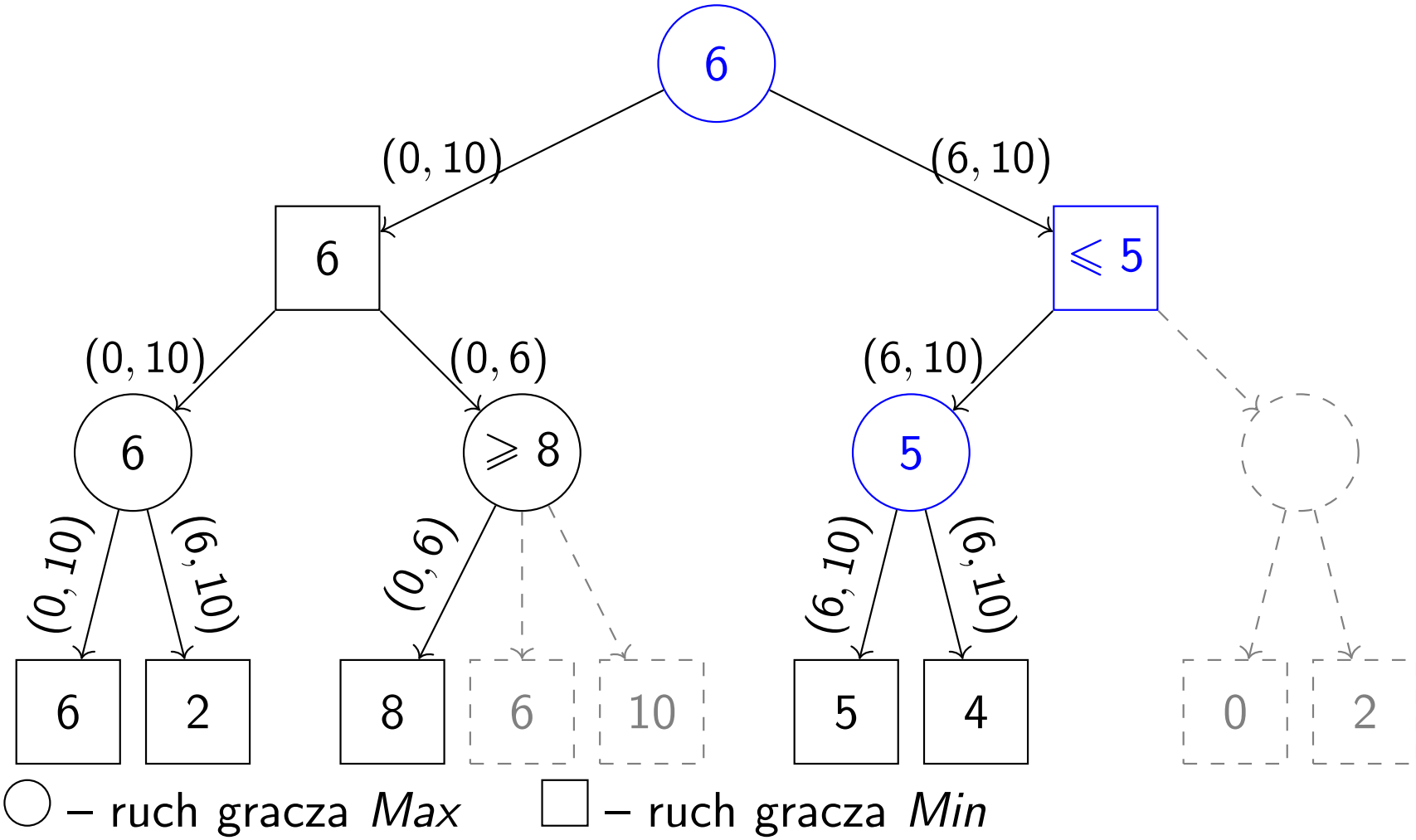
5 staje się bieżącą wartością poprzednika osiągniętej pozycji końcowej. Poprzednik ten jest typu MAX, więc jego ostateczna wartość wyniesie 5 lub więcej. Nie można odciąć więc jego drugiego następnika, bo wartość $\geqslant 5$ może się zmieścić w oknie $(6, 10)$, tj. $[5, \infty) \cap (6, 10) \neq \emptyset$.



Dla drugiego następnika algorytm wywołuje się w oknie $(5, \infty) \cap (6, 10) = (6, 10)$, bo wartości $\leqslant 6$ nie mieszczą się w przekazanym z góry oknie, nie są więc interesujące. 4 nie poprawia bieżącego maksimum. Nie ma już więcej następników, więc 5 jest ostateczną wartością pozycji.



5 staje się bieżącą wartością prawego następnika korzenia. Następnik ten jest typu MIN, więc jego ostateczna wartość wyniesie $\leqslant 5$ i nie zmieści się w oknie $(6, 10)$, tj. $(-\infty, 5] \cap (6, 10) = \emptyset$. Następuje odcięcie i zwrócenie wartości 5 do korzenia, która jednak nie poprawia jego bieżącego (i ostatecznego) maksimum 6.



Pseudokod algorytmu fail-soft α-β

```
1  fun AlfaBeta(s, α, β):
2      if ev(s) ∈ [0, 1]: return ev(s)
3      if ev(s) = Max:
4          res ← 0
5          foreach m ∈ N(s):
6              resnew ← AlfaBeta(m, max(α, res), β)
7              if resnew ⩾ β: return resnew //odcięcie
8              if resnew > res: res ← resnew
9      else: //ev(s) = Min:
10         res ← 1
11         foreach m ∈ N(s):
12             resnew ← AlfaBeta(m, α, min(β, res))
13             if resnew ⩽ α: return resnew //odcięcie
14             if resnew < res: res ← resnew
15     return res
```

Twierdzenie o wynikach algorytmu α-β [Knuth1975]

Niech $(\mathbb{S}, s_I, N, \mathrm{ev})$ – gra, $s \in \mathbb{S}$ – stan gry, $0 \leqslant \alpha < \beta \leqslant 1$.
Oznaczmy ponadto $ab = \mathrm{AlfaBeta}(s, \alpha, \beta)$ oraz $m = \mathrm{MinMax}(s)$.
Wtedy możemy wyróżnić trzy sytuacje:

$$\alpha \geqslant ab \quad \Longrightarrow \quad ab \geqslant m,$$

$$\alpha < ab < \beta \quad \Longrightarrow \quad ab = m,$$

$$\beta \leqslant ab \quad \Longrightarrow \quad ab \leqslant m.$$

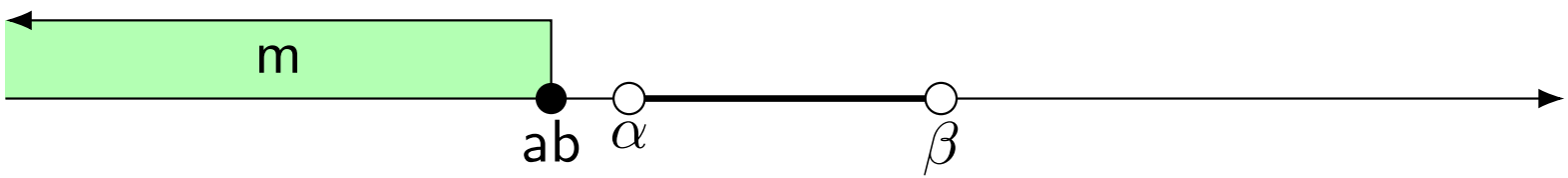
Twierdzenie o wynikach algorytmu α-β [Knuth1975]

Niech $(\mathbb{S}, s_I, N, \mathrm{ev})$ – gra, $s \in \mathbb{S}$ – stan gry, $0 \leqslant \alpha < \beta \leqslant 1$.
Oznaczmy ponadto $ab = \mathrm{AlfaBeta}(s, \alpha, \beta)$ oraz $m = \mathrm{MinMax}(s)$.
Wtedy możemy wyróżnić trzy sytuacje:

$$\alpha \geqslant ab \quad \Longrightarrow \quad ab \geqslant m,$$

$$\alpha < ab < \beta \quad \Longrightarrow \quad ab = m,$$

$$\beta \leqslant ab \quad \Longrightarrow \quad ab \leqslant m.$$



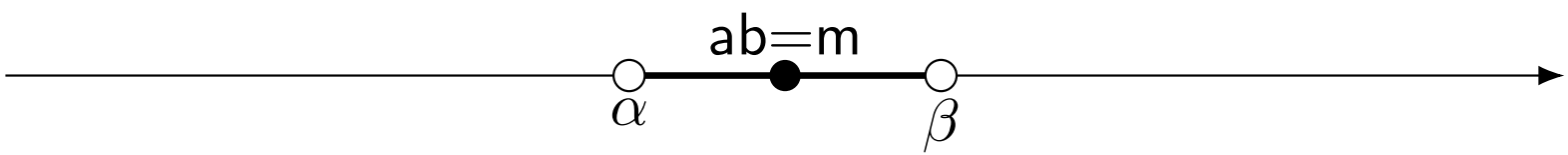
Twierdzenie o wynikach algorytmu α-β [Knuth1975]

Niech $(\mathbb{S}, s_I, N, \mathrm{ev})$ – gra, $s \in \mathbb{S}$ – stan gry, $0 \leqslant \alpha < \beta \leqslant 1$.
Oznaczmy ponadto $ab = \mathrm{AlfaBeta}(s, \alpha, \beta)$ oraz $m = \mathrm{MinMax}(s)$.
Wtedy możemy wyróżnić trzy sytuacje:

$$\alpha \geqslant ab \quad \Longrightarrow \quad ab \geqslant m,$$

$$\alpha < ab < \beta \quad \Longrightarrow \quad ab = m,$$

$$\beta \leqslant ab \quad \Longrightarrow \quad ab \leqslant m.$$



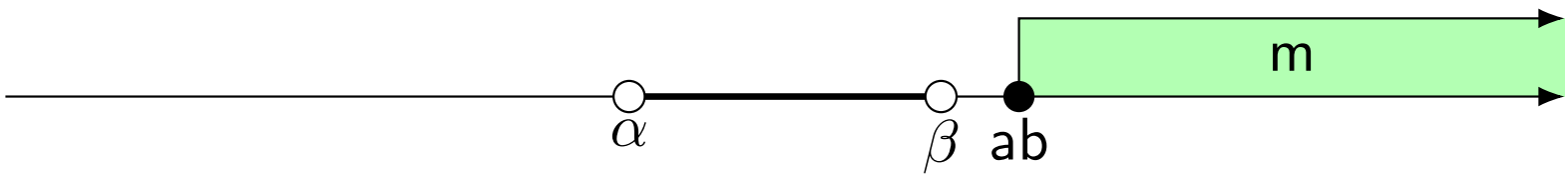
Twierdzenie o wynikach algorytmu α-β [Knuth1975]

Niech $(\mathbb{S}, s_I, N, \mathrm{ev})$ – gra, $s \in \mathbb{S}$ – stan gry, $0 \leqslant \alpha < \beta \leqslant 1$.
Oznaczmy ponadto $ab = \mathrm{AlfaBeta}(s, \alpha, \beta)$ oraz $m = \mathrm{MinMax}(s)$.
Wtedy możemy wyróżnić trzy sytuacje:

$$\alpha \geqslant ab \quad \Longrightarrow \quad ab \geqslant m,$$

$$\alpha < ab < \beta \quad \Longrightarrow \quad ab = m,$$

$$\beta \leqslant ab \quad \Longrightarrow \quad ab \leqslant m.$$



Twierdzenie o wynikach algorytmu α-β [Knuth1975]

Niech $(\mathbb{S}, s_I, N, \mathrm{ev})$ – gra, $s \in \mathbb{S}$ – stan gry, $0 \leqslant \alpha < \beta \leqslant 1$.
Oznaczmy ponadto $ab = \mathrm{AlfaBeta}(s, \alpha, \beta)$ oraz $m = \mathrm{MinMax}(s)$.
Wtedy możemy wyróżnić trzy sytuacje:

$$\alpha \geqslant ab \quad \Longrightarrow \quad ab \geqslant m,$$

$$\alpha < ab < \beta \quad \Longrightarrow \quad ab = m,$$

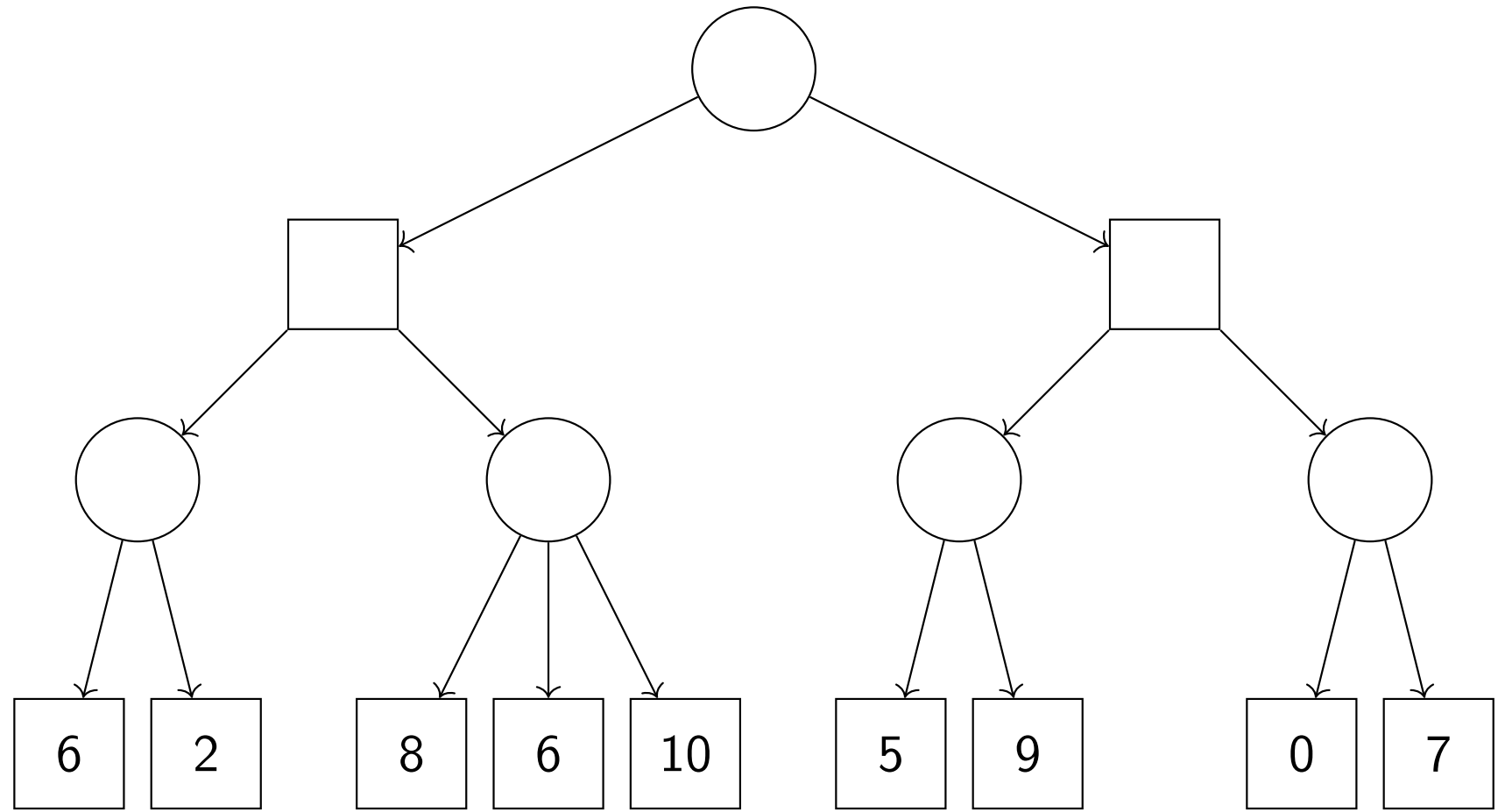
$$\beta \leqslant ab \quad \Longrightarrow \quad ab \leqslant m.$$

Wniosek:

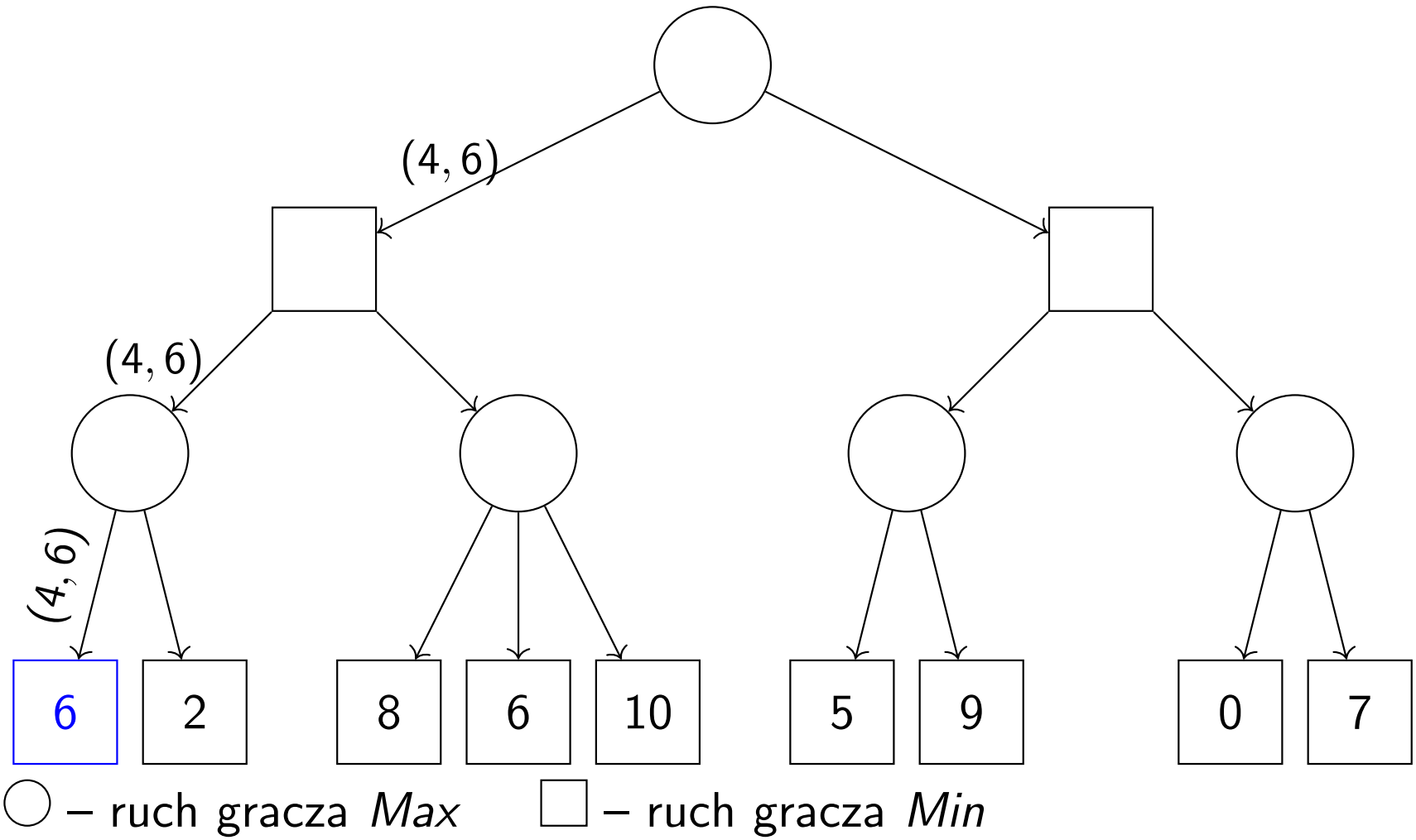
$$\forall s \in \mathbb{S}: \qquad \mathrm{AlfaBeta}(s, 0, 1) = \mathrm{MinMax}(s),$$

co określa możliwość bezpośredniego wykorzystania α-β
do znalezienia wartości minimaksowej stanu gry.

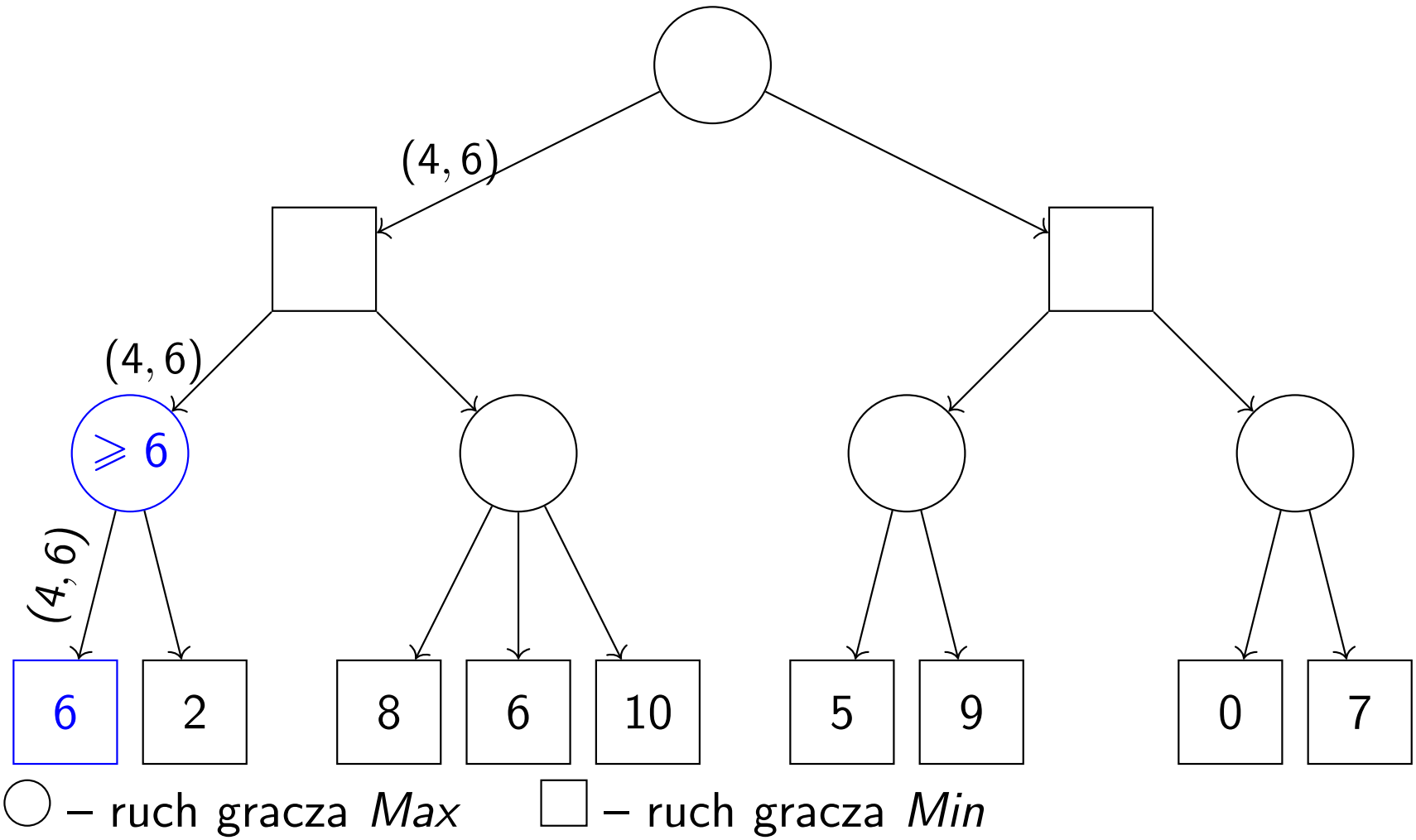
Przykład: Prześledźmy działanie algorytmu α-β wywołanego z oknem $(4, 6)$ na następującym grafie gry:



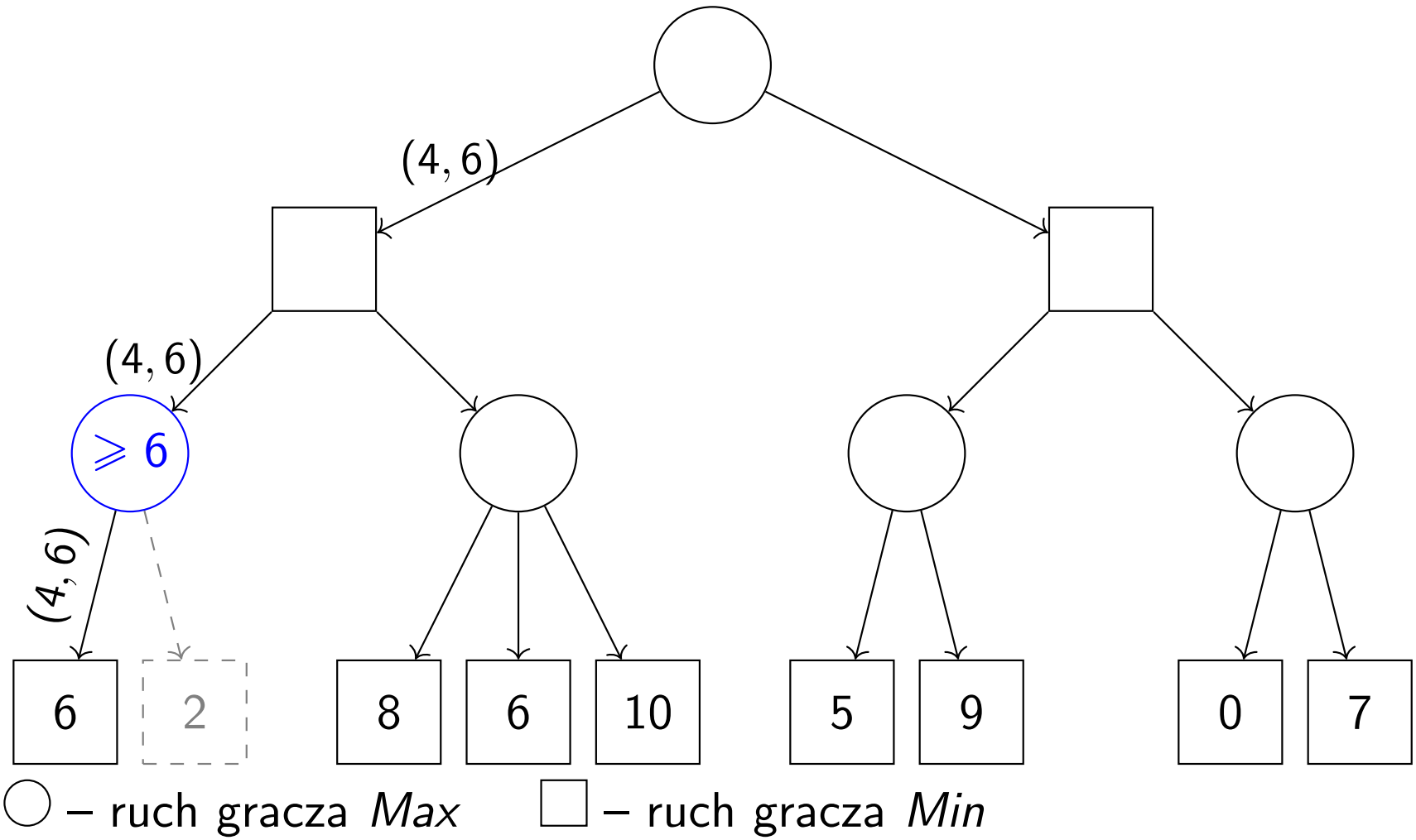
Przy użyciu początkowego okna $(4, 6)$, następują rekurencyjne wywołania dla kolejnych, pierwszych (od lewej) następników, aż do osiągnięcia pozycji końcowej o wartości 6.



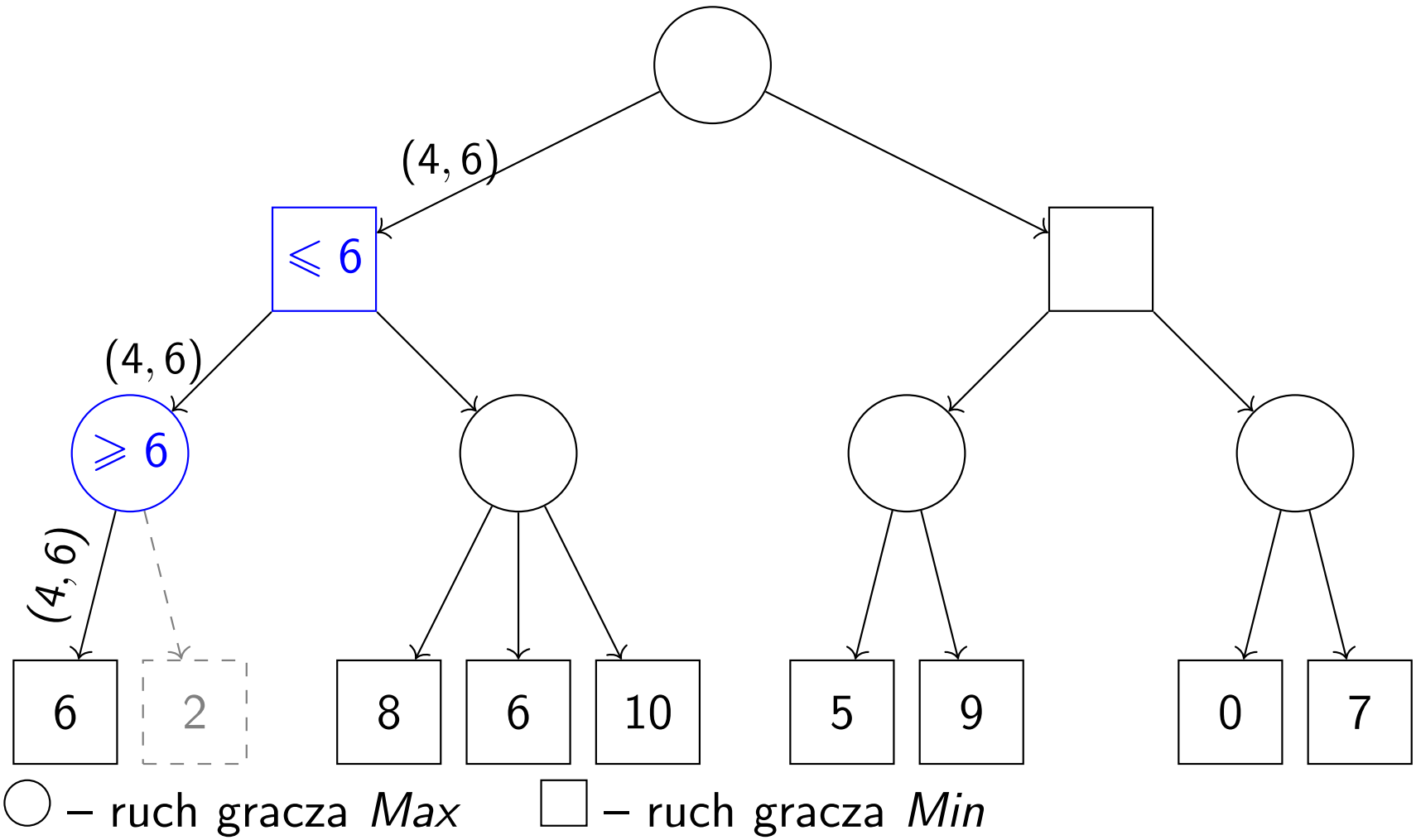
6 staje się bieżącą wartością poprzednika osiągniętej pozycji końcowej. Poprzednik ten jest typu MAX, więc jego ostateczna wartość wyniesie 6 lub więcej.



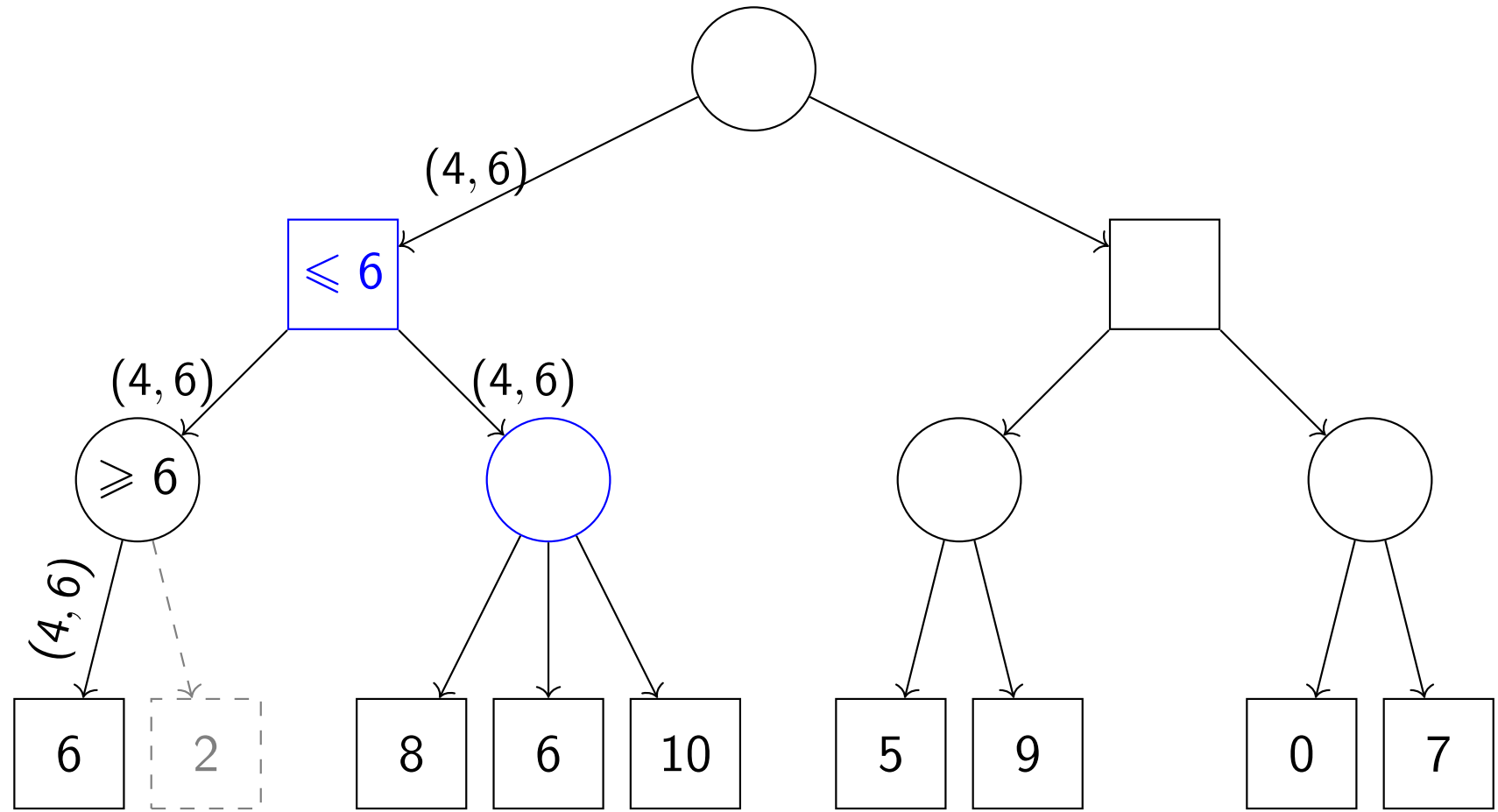
Wartość $\geqslant 6$ nie zmieścić w oknie $(4, 6)$, tj. $[6, \infty) \cap (4, 6) = \emptyset$.
6 przekroczy okno z prawej strony. Drugi następnik nie będzie więc badany, zostanie odcięty.



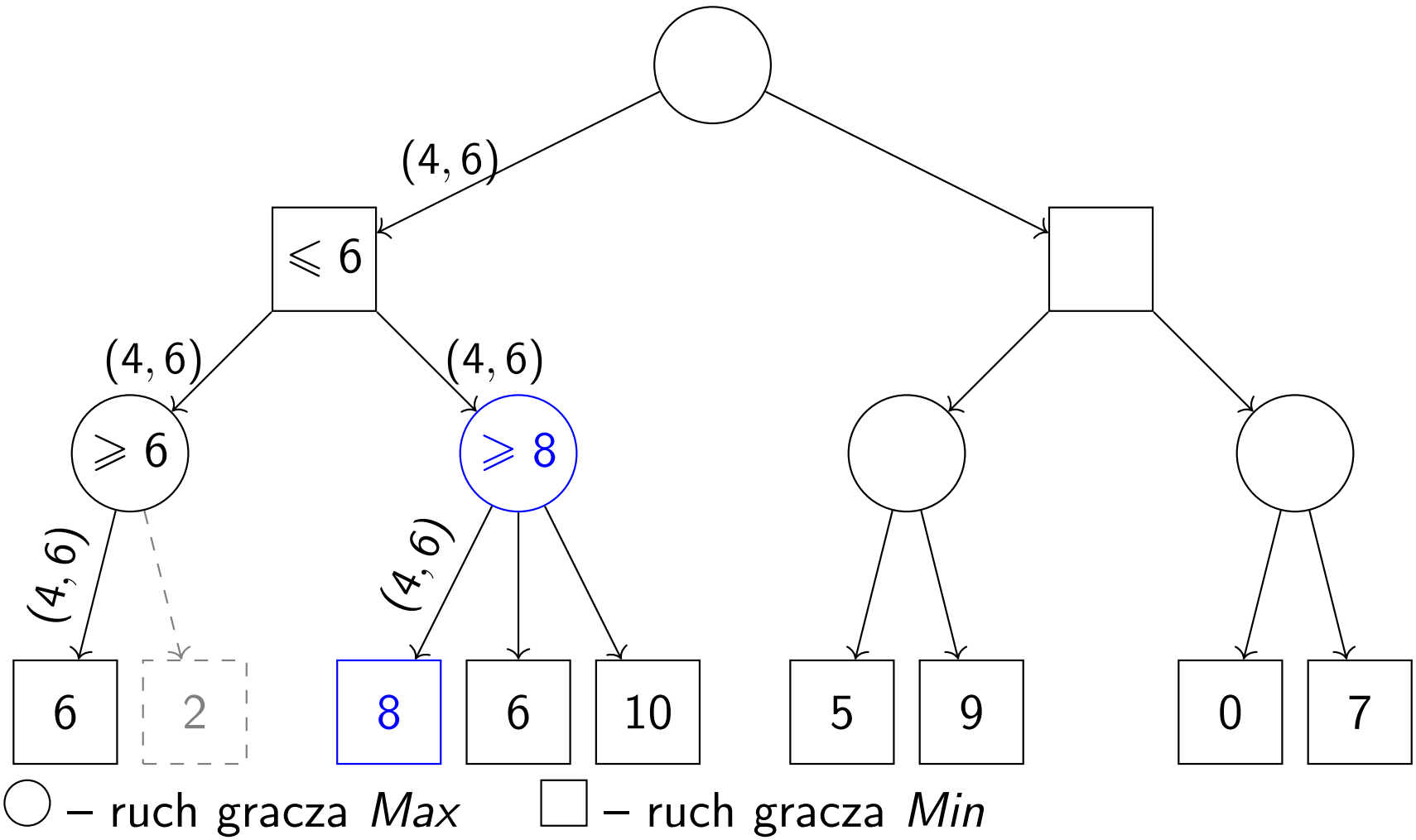
6 staje się bieżącą wartością poprzednika zbadanej pozycji. Poprzednik ten jest typu MIN, więc jego ostateczna wartość wyniesie 6 lub mniej. Ponieważ $(-\infty, 6] \cap (4, 6) \neq \emptyset$, to nie można odciąć drugiego następnika.



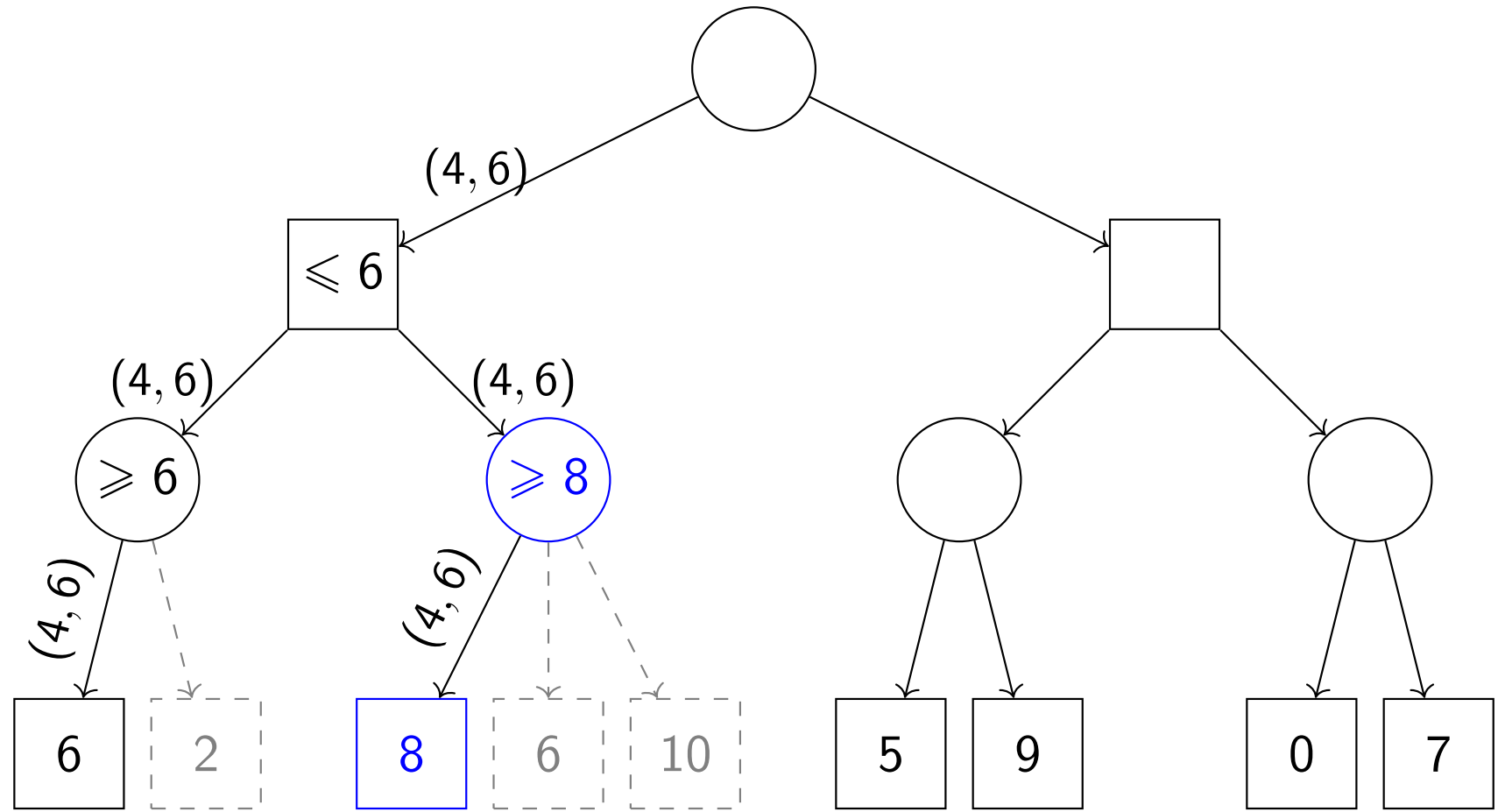
Dla drugiego następnika algorytm wywołuje się w oknie $(-\infty, 6) \cap (4, 6) = (4, 6)$.



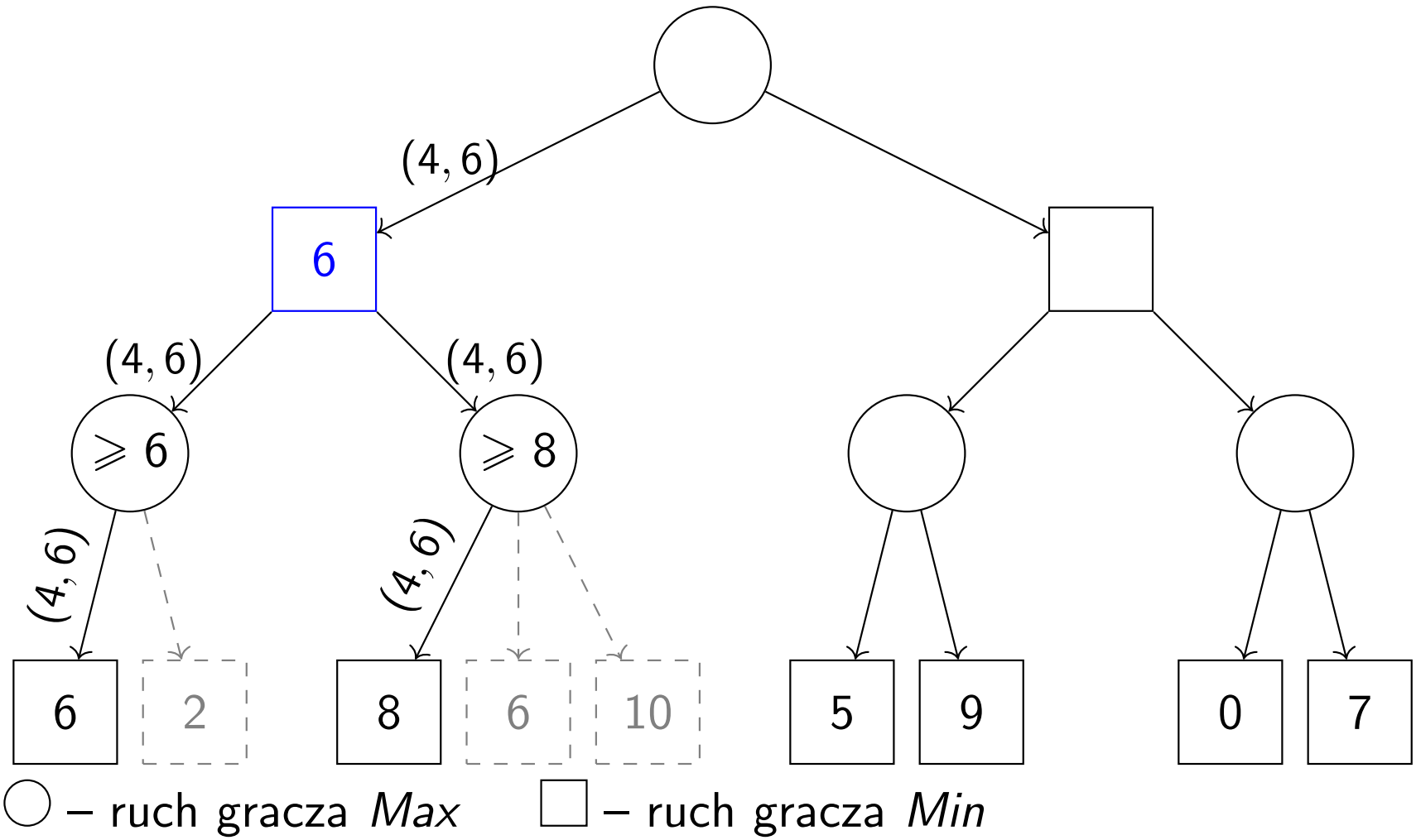
I w tym samym oknie $(4, 6)$ wywołuje się dla jego następnika o wartość 8. 8 staje się jego bieżącym maksimum, zaś jego ostateczna wartość wyniesie $\geqslant 8$.



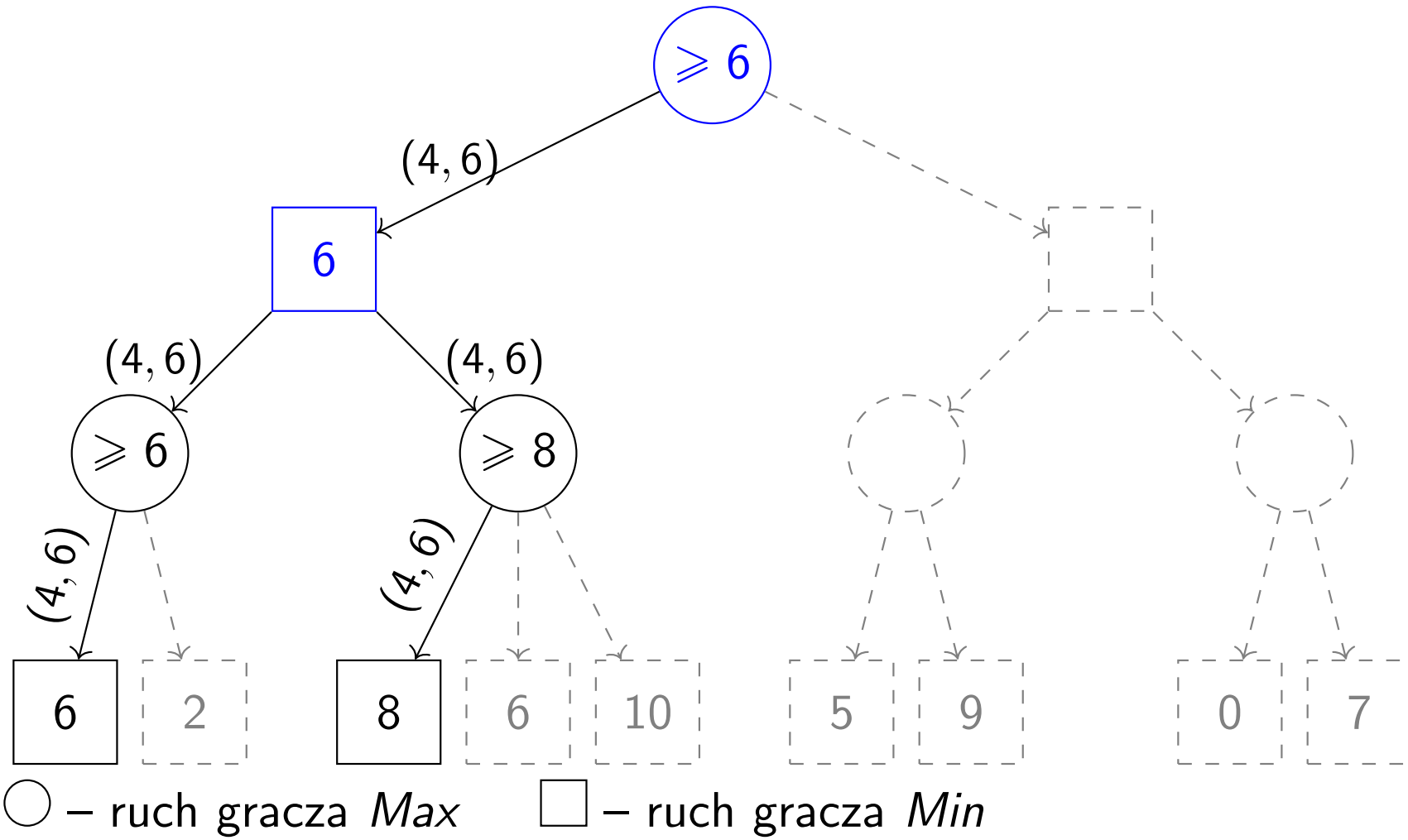
Ta ostateczna wartość $\geqslant 8$ nie zmieści się w oknie $(4, 6)$, tj. $[8, \infty) \cap (4, 6) = \emptyset$. 8 przekroczy okno z prawej strony. Następuje więc odcięcie kolejnych następników. Bieżące maksimum 8 staje się ostateczną wartością (zwróconą z) zbadanego węzła.



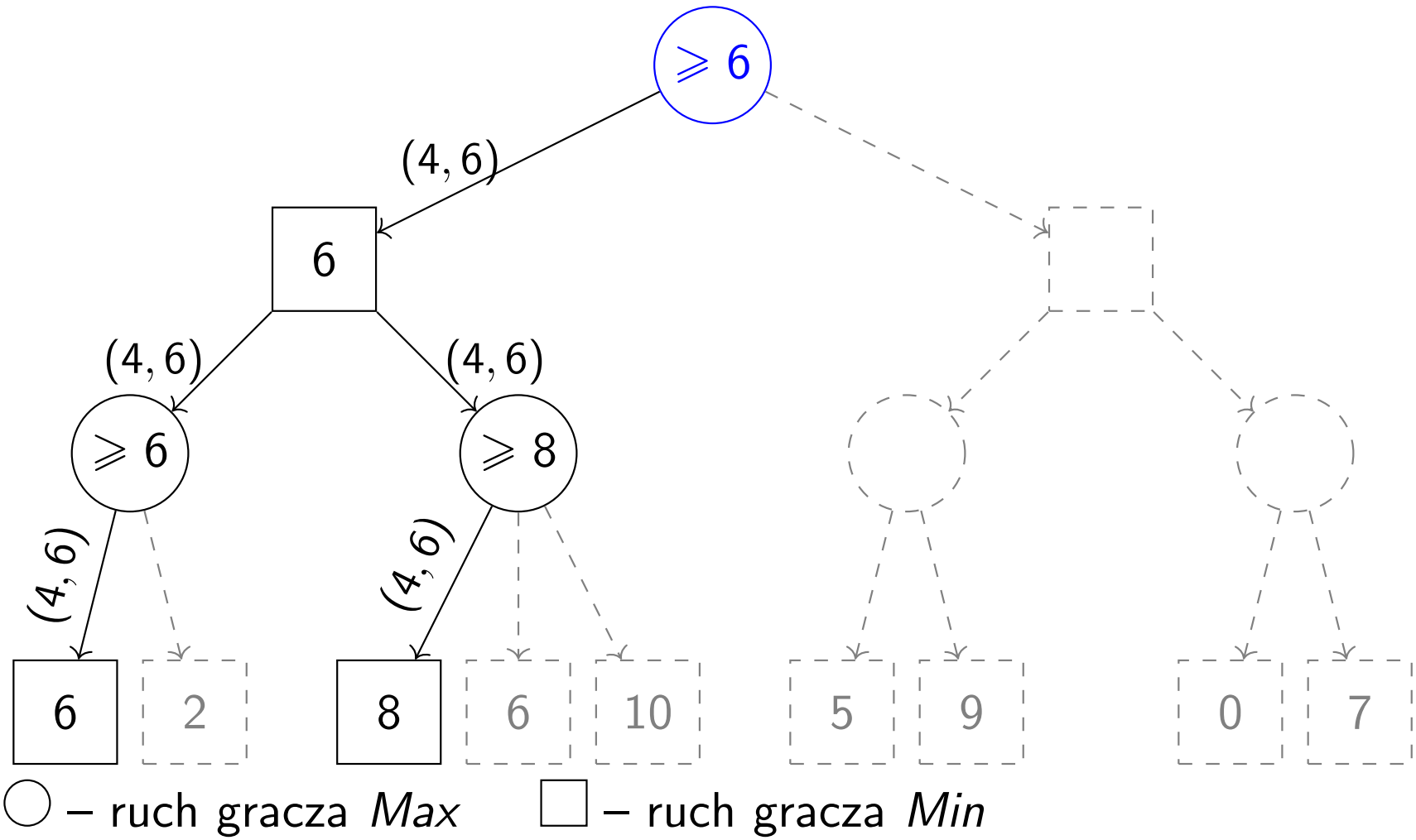
Ta 8 nie poprawia bieżącego minimum 6 lewego następnika korzenia. 6 staje się zatem jego ostateczną wartością, bo nie ma on więcej dzieci.



6 staje się bieżącą wartością korzenia. Korzeń jest typu MAX, więc jego ostateczna wartość wyniesie 6 lub więcej. Ta ostateczna wartość nie zmieści się w oknie $(4, 6)$, tj. $[6, \infty) \cap (4, 6) = \emptyset$; przekroczy to okno z prawej strony. Następuje więc odcięcie prawego następnika korzenia.



Ostatecznie więc algorytm α-β wywołany w oknie $(4, 6)$ zwróci $ab = 6$. Ponieważ $\beta = 6 \leqslant 6 = ab$, to zgodnie z twierdzeniem o wynikach algorytmu α-β, wartość minimaksowa m korzenia spełnia $ab = 6 \leqslant m$, tj. wynosi 6 lub więcej.



Złożoność czasowa algorytmu α-β [Knuth1975]

Złożoność czasowa algorytmu α-β zależy od kolejności badania następników poszczególnych stanów.

Gdy każda pozycja ma b następników, zaś głębokość poszukiwań to d, liczba odwiedzonych pozycji wynosi:

- $O\left(b^{0,5d}\right)$ – optymistycznie (gdy najlepsze dla idącego ruchy sprawdzane są jako pierwsze);
- $O\left(b^{d}\right)$ – pesymistycznie (brak odcięć; złożoność taka sama jak algorytmu Min-Max);
- $O\left(b^{0,75d}\right)$ (dla $b < 10$) albo $O\left(b^{0,76d}\right)$ (dla $10 \leqslant b \leqslant 30$) – oczekiwana, dla ruchów wykonanych w losowej kolejności (rozkład jednostajny).

Złożoność czasowa algorytmu α-β [Knuth1975]

Złożoność czasowa algorytmu α-β zależy od kolejności badania następników poszczególnych stanów.

Gdy każda pozycja ma b następników, zaś głębokość poszukiwań to d, liczba odwiedzonych pozycji wynosi:

- $O\left(b^{0,5d}\right)$ – optymistycznie (gdy najlepsze dla idącego ruchy sprawdzane są jako pierwsze);
- $O\left(b^{d}\right)$ – pesymistycznie (brak odcięć; złożoność taka sama jak algorytmu Min-Max);
- $O\left(b^{0,75d}\right)$ (dla $b < 10$) albo $O\left(b^{0,76d}\right)$ (dla $10 \leqslant b \leqslant 30$) – oczekiwana, dla ruchów wykonanych w losowej kolejności (rozkład jednostajny).

Złożoność czasowa algorytmu α-β [Knuth1975]

Złożoność czasowa algorytmu α-β zależy od kolejności badania następników poszczególnych stanów.

Gdy każda pozycja ma b następników, zaś głębokość poszukiwań to d, liczba odwiedzonych pozycji wynosi:

- $O\left(b^{0,5d}\right)$ – optymistycznie (gdy najlepsze dla idącego ruchy sprawdzane są jako pierwsze);
- $O\left(b^{d}\right)$ – pesymistycznie (brak odcięć; złożoność taka sama jak algorytmu Min-Max);
- $O\left(b^{0,75d}\right)$ (dla $b < 10$) albo $O\left(b^{0,76d}\right)$ (dla $10 \leqslant b \leqslant 30$) – oczekiwana, dla ruchów wykonanych w losowej kolejności (rozkład jednostajny).

Złożoność czasowa algorytmu α-β [Knuth1975]

Złożoność czasowa algorytmu α-β zależy od kolejności badania następników poszczególnych stanów.

Gdy każda pozycja ma b następników, zaś głębokość poszukiwań to d, liczba odwiedzonych pozycji wynosi:

- $O\left(b^{0,5d}\right)$ – optymistycznie (gdy najlepsze dla idącego ruchy sprawdzane są jako pierwsze);
- $O\left(b^{d}\right)$ – pesymistycznie (brak odcięć; złożoność taka sama jak algorytmu Min-Max);
- $O\left(b^{0,75d}\right)$ (dla $b < 10$) albo $O\left(b^{0,76d}\right)$ (dla $10 \leqslant b \leqslant 30$) – oczekiwana, dla ruchów wykonanych w losowej kolejności (rozkład jednostajny).

Złożoność czasowa algorytmu α-β [Knuth1975]

Złożoność czasowa algorytmu α-β zależy od kolejności badania następników poszczególnych stanów.

Gdy każda pozycja ma b następników, zaś głębokość poszukiwań to d, liczba odwiedzonych pozycji wynosi:

- $O\left(b^{0,5d}\right)$ – optymistycznie (gdy najlepsze dla idącego ruchy sprawdzane są jako pierwsze);
- $O\left(b^{d}\right)$ – pesymistycznie (brak odcięć; złożoność taka sama jak algorytmu Min-Max);
- $O\left(b^{0,75d}\right)$ (dla $b < 10$) albo $O\left(b^{0,76d}\right)$ (dla $10 \leqslant b \leqslant 30$) – oczekiwana, dla ruchów wykonanych w losowej kolejności (rozkład jednostajny).

Przyspieszanie poprzez uporządkowanie ruchów

- Wniosek: warto uporządkować ruchy tak, by zwiększyć szansę na wczesne odwiedzenie dobrych następników;
- w tym celu można użyć statycznych, heurystycznych ocen ruchów;
- zazwyczaj oparte są one o wiedzę dziedzinową na temat gry;
- istnieją też bardziej ogólne oceny, jak np. *heurystyka ruchów morderców* (ang. killer heuristic) czy *heurystyka historyczna* (ang. history heuristic), opierające się na założeniu, że ruchy które okazały się dobre (np. spowodowały odcięcie) w pewnych pozycjach, prawdopodobnie będą też dobre w innych, podobnych.
- Ponadto, w pierwszej kolejności warto przeszukiwać gałęzie w których jest mniej węzłów, by zwiększyć szansę na odcięcie bardziej rozgałęzionych części grafu lub by przeszukiwać je w możliwie najmniejszym oknie (α, β).

Przyspieszanie poprzez uporządkowanie ruchów

- Wniosek: warto uporządkować ruchy tak, by zwiększyć szansę na wczesne odwiedzenie dobrych następników;
- w tym celu można użyć statycznych, heurystycznych ocen ruchów;
- zazwyczaj oparte są one o wiedzę dziedzinową na temat gry;
- istnieją też bardziej ogólne oceny, jak np. *heurystyka ruchów morderców* (ang. killer heuristic) czy *heurystyka historyczna* (ang. history heuristic), opierające się na założeniu, że ruchy które okazały się dobre (np. spowodowały odcięcie) w pewnych pozycjach, prawdopodobnie będą też dobre w innych, podobnych.
- Ponadto, w pierwszej kolejności warto przeszukiwać gałęzie w których jest mniej węzłów, by zwiększyć szansę na odcięcie bardziej rozgałęzionych części grafu lub by przeszukiwać je w możliwie najmniejszym oknie (α, β).

Przyspieszanie poprzez uporządkowanie ruchów

- Wniosek: warto uporządkować ruchy tak, by zwiększyć szansę na wczesne odwiedzenie dobrych następników;
- w tym celu można użyć statycznych, heurystycznych ocen ruchów;
- zazwyczaj oparte są one o wiedzę dziedzinową na temat gry;
- istnieją też bardziej ogólne oceny, jak np. *heurystyka ruchów morderców* (ang. killer heuristic) czy *heurystyka historyczna* (ang. history heuristic), opierające się na założeniu, że ruchy które okazały się dobre (np. spowodowały odcięcie) w pewnych pozycjach, prawdopodobnie będą też dobre w innych, podobnych.
- Ponadto, w pierwszej kolejności warto przeszukiwać gałęzie w których jest mniej węzłów, by zwiększyć szansę na odcięcie bardziej rozgałęzionych części grafu lub by przeszukiwać je w możliwie najmniejszym oknie (α, β).

Przyspieszanie poprzez uporządkowanie ruchów

- Wniosek: warto uporządkować ruchy tak, by zwiększyć szansę na wczesne odwiedzenie dobrych następników;
- w tym celu można użyć statycznych, heurystycznych ocen ruchów;
- zazwyczaj oparte są one o wiedzę dziedzinową na temat gry;
- istnieją też bardziej ogólne oceny, jak np. *heurystyka ruchów morderców* (ang. killer heuristic) czy *heurystyka historyczna* (ang. history heuristic), opierające się na założeniu, że ruchy które okazały się dobre (np. spowodowały odcięcie) w pewnych pozycjach, prawdopodobnie będą też dobre w innych, podobnych.
- Ponadto, w pierwszej kolejności warto przeszukiwać gałęzie w których jest mniej węzłów, by zwiększyć szansę na odcięcie bardziej rozgałęzionych części grafu lub by przeszukiwać je w możliwie najmniejszym oknie (α, β).

Przyspieszanie poprzez uporządkowanie ruchów

- Wniosek: warto uporządkować ruchy tak, by zwiększyć szansę na wczesne odwiedzenie dobrych następników;
- w tym celu można użyć statycznych, heurystycznych ocen ruchów;
- zazwyczaj oparte są one o wiedzę dziedzinową na temat gry;
- istnieją też bardziej ogólne oceny, jak np. *heurystyka ruchów morderców* (ang. killer heuristic) czy *heurystyka historyczna* (ang. history heuristic), opierające się na założeniu, że ruchy które okazały się dobre (np. spowodowały odcięcie) w pewnych pozycjach, prawdopodobnie będą też dobre w innych, podobnych.
- Ponadto, w pierwszej kolejności warto przeszukiwać gałęzie w których jest mniej węzłów, by zwiększyć szansę na odcięcie bardziej rozgałęzionych części grafu lub by przeszukiwać je w możliwie najmniejszym oknie (α, β).

α-β z tablicą transpozycji

- Jak wynika z twierdzenia o wynikach α-β, algorytm ten nie zawsze znajduje dokładną wartość minimaksową, czasem jedynie jej ograniczenie z góry lub z dołu;
- dlatego w tablicy transpozycji (TT), oprócz obliczonej liczby, należy zapamiętać czy jest to wartość: dokładna, jej ograniczenie z góry, czy jej ograniczenie z dołu.
- Uniknięcie głębszego przeszukiwania grafu gry jest możliwe gdy w TT znajduje się dokładna wartość minimaksowa badanej pozycji lub gdy z TT wynika, że wartość minimaksowa badanej pozycji znajduje się poza oknem (α, β);
- w przeciwnym razie wciąż można na podstawie TT spróbować zawęzić otrzymane okno poszukiwań (α, β).
- Przykład: jeśli z TT wiemy, że wartość minimaksowa badanej pozycji jest większa lub równa x, to:
  - jeśli $x \geqslant \beta$, to można wykonać odcięcie (zwrócić x);
  - jeśli $\beta > x > \alpha$, to można zawęzić okno do (x, β).

α-β z tablicą transpozycji

- Jak wynika z twierdzenia o wynikach α-β, algorytm ten nie zawsze znajduje dokładną wartość minimaksową, czasem jedynie jej ograniczenie z góry lub z dołu;
- dlatego w tablicy transpozycji (TT), oprócz obliczonej liczby, należy zapamiętać czy jest to wartość: dokładna, jej ograniczenie z góry, czy jej ograniczenie z dołu.
- Uniknięcie głębszego przeszukiwania grafu gry jest możliwe gdy w TT znajduje się dokładna wartość minimaksowa badanej pozycji lub gdy z TT wynika, że wartość minimaksowa badanej pozycji znajduje się poza oknem (α, β);
- w przeciwnym razie wciąż można na podstawie TT spróbować zawęzić otrzymane okno poszukiwań (α, β).
- Przykład: jeśli z TT wiemy, że wartość minimaksowa badanej pozycji jest większa lub równa x, to:
  - jeśli $x \geqslant \beta$, to można wykonać odcięcie (zwrócić x);
  - jeśli $\beta > x > \alpha$, to można zawęzić okno do (x, β).

α-β z tablicą transpozycji

- Jak wynika z twierdzenia o wynikach α-β, algorytm ten nie zawsze znajduje dokładną wartość minimaksową, czasem jedynie jej ograniczenie z góry lub z dołu;
- dlatego w tablicy transpozycji (TT), oprócz obliczonej liczby, należy zapamiętać czy jest to wartość: dokładna, jej ograniczenie z góry, czy jej ograniczenie z dołu.
- Uniknięcie głębszego przeszukiwania grafu gry jest możliwe gdy w TT znajduje się dokładna wartość minimaksowa badanej pozycji lub gdy z TT wynika, że wartość minimaksowa badanej pozycji znajduje się poza oknem (α, β);
- w przeciwnym razie wciąż można na podstawie TT spróbować zawęzić otrzymane okno poszukiwań (α, β).
- Przykład: jeśli z TT wiemy, że wartość minimaksowa badanej pozycji jest większa lub równa x, to:
  - jeśli $x \geqslant \beta$, to można wykonać odcięcie (zwrócić x);
  - jeśli $\beta > x > \alpha$, to można zawęzić okno do (x, β).

α-β z tablicą transpozycji

- Jak wynika z twierdzenia o wynikach α-β, algorytm ten nie zawsze znajduje dokładną wartość minimaksową, czasem jedynie jej ograniczenie z góry lub z dołu;
- dlatego w tablicy transpozycji (TT), oprócz obliczonej liczby, należy zapamiętać czy jest to wartość: dokładna, jej ograniczenie z góry, czy jej ograniczenie z dołu.
- Uniknięcie głębszego przeszukiwania grafu gry jest możliwe gdy w TT znajduje się dokładna wartość minimaksowa badanej pozycji lub gdy z TT wynika, że wartość minimaksowa badanej pozycji znajduje się poza oknem (α, β);
- w przeciwnym razie wciąż można na podstawie TT spróbować zawęzić otrzymane okno poszukiwań (α, β).
- Przykład: jeśli z TT wiemy, że wartość minimaksowa badanej pozycji jest większa lub równa x, to:
  - jeśli $x \geqslant \beta$, to można wykonać odcięcie (zwrócić x);
  - jeśli $\beta > x > \alpha$, to można zawęzić okno do (x, β).

α-β z tablicą transpozycji

- Jak wynika z twierdzenia o wynikach α-β, algorytm ten nie zawsze znajduje dokładną wartość minimaksową, czasem jedynie jej ograniczenie z góry lub z dołu;
- dlatego w tablicy transpozycji (TT), oprócz obliczonej liczby, należy zapamiętać czy jest to wartość: dokładna, jej ograniczenie z góry, czy jej ograniczenie z dołu.
- Uniknięcie głębszego przeszukiwania grafu gry jest możliwe gdy w TT znajduje się dokładna wartość minimaksowa badanej pozycji lub gdy z TT wynika, że wartość minimaksowa badanej pozycji znajduje się poza oknem (α, β);
- w przeciwnym razie wciąż można na podstawie TT spróbować zawęzić otrzymane okno poszukiwań (α, β).
- Przykład: jeśli z TT wiemy, że wartość minimaksowa badanej pozycji jest większa lub równa x, to:
  - jeśli $x \geqslant \beta$, to można wykonać odcięcie (zwrócić x);
  - jeśli $\beta > x > \alpha$, to można zawęzić okno do (x, β).

α-β z tablicą transpozycji

- Jak wynika z twierdzenia o wynikach α-β, algorytm ten nie zawsze znajduje dokładną wartość minimaksową, czasem jedynie jej ograniczenie z góry lub z dołu;
- dlatego w tablicy transpozycji (TT), oprócz obliczonej liczby, należy zapamiętać czy jest to wartość: dokładna, jej ograniczenie z góry, czy jej ograniczenie z dołu.
- Uniknięcie głębszego przeszukiwania grafu gry jest możliwe gdy w TT znajduje się dokładna wartość minimaksowa badanej pozycji lub gdy z TT wynika, że wartość minimaksowa badanej pozycji znajduje się poza oknem (α, β);
- w przeciwnym razie wciąż można na podstawie TT spróbować zawęzić otrzymane okno poszukiwań (α, β).
- Przykład: jeśli z TT wiemy, że wartość minimaksowa badanej pozycji jest większa lub równa x, to:
  - jeśli $x \geqslant \beta$, to można wykonać odcięcie (zwrócić x);
  - jeśli $\beta > x > \alpha$, to można zawęzić okno do (x, β).

α-β z tablicą transpozycji

- Jak wynika z twierdzenia o wynikach α-β, algorytm ten nie zawsze znajduje dokładną wartość minimaksową, czasem jedynie jej ograniczenie z góry lub z dołu;
- dlatego w tablicy transpozycji (TT), oprócz obliczonej liczby, należy zapamiętać czy jest to wartość: dokładna, jej ograniczenie z góry, czy jej ograniczenie z dołu.
- Uniknięcie głębszego przeszukiwania grafu gry jest możliwe gdy w TT znajduje się dokładna wartość minimaksowa badanej pozycji lub gdy z TT wynika, że wartość minimaksowa badanej pozycji znajduje się poza oknem (α, β);
- w przeciwnym razie wciąż można na podstawie TT spróbować zawęzić otrzymane okno poszukiwań (α, β).
- Przykład: jeśli z TT wiemy, że wartość minimaksowa badanej pozycji jest większa lub równa x, to:
  - jeśli $x \geqslant \beta$, to można wykonać odcięcie (zwrócić x);
  - jeśli $\beta > x > \alpha$, to można zawęzić okno do (x, β).

MTD(f) i rodzina algorytmów MTD [Plaat1996]

- *Memory-enhanced Test (MT)* to używające tablicy transpozycji wywołanie fail-soft α-β w zerowym oknie, tj. dla takich wartości α, β, że nie istnieje wartość wypłaty a taka, że $\alpha < a < \beta$ (różnica między α i β równa jest najmniejszej możliwej, niepodzielnej jednostce wypłaty);
- zgodnie z twierdzeniem o wynikach algorytmu α-β, takie wywołanie pozwala określić nowe, dolne albo górne ograniczenie szukanej wartości minimaksowej m, tj. albo $m \geqslant ab \geqslant \beta$, albo $m \leqslant ab < \beta$, gdzie ab to wynik wywołania;
- algorytmy z rodziny MTD (Memory-enhanced Test Driver) używają serii takich wywołań do stopniowego zawężania przedziału w którym mieści się szukana wartość minimaksowa;
- przy czym zawartość tablicy transpozycji nie jest czyszczona pomiędzy kolejnymi wywołaniami.
- MTD(f) jest przedstawicielem rodziny MTD, który jako β dla kolejnego wywołania używa wyniku poprzedniego.

MTD(f) i rodzina algorytmów MTD [Plaat1996]

- *Memory-enhanced Test (MT)* to używające tablicy transpozycji wywołanie fail-soft α-β w zerowym oknie, tj. dla takich wartości α, β, że nie istnieje wartość wypłaty a taka, że $\alpha < a < \beta$ (różnica między α i β równa jest najmniejszej możliwej, niepodzielnej jednostce wypłaty);
- zgodnie z twierdzeniem o wynikach algorytmu α-β, takie wywołanie pozwala określić nowe, dolne albo górne ograniczenie szukanej wartości minimaksowej m, tj. albo $m \geqslant ab \geqslant \beta$, albo $m \leqslant ab < \beta$, gdzie ab to wynik wywołania;
- algorytmy z rodziny MTD (Memory-enhanced Test Driver) używają serii takich wywołań do stopniowego zawężania przedziału w którym mieści się szukana wartość minimaksowa;
- przy czym zawartość tablicy transpozycji nie jest czyszczona pomiędzy kolejnymi wywołaniami.
- MTD(f) jest przedstawicielem rodziny MTD, który jako β dla kolejnego wywołania używa wyniku poprzedniego.

MTD(f) i rodzina algorytmów MTD [Plaat1996]

- *Memory-enhanced Test (MT)* to używające tablicy transpozycji wywołanie fail-soft α-β w zerowym oknie, tj. dla takich wartości α, β, że nie istnieje wartość wypłaty a taka, że $\alpha < a < \beta$ (różnica między α i β równa jest najmniejszej możliwej, niepodzielnej jednostce wypłaty);
- zgodnie z twierdzeniem o wynikach algorytmu α-β, takie wywołanie pozwala określić nowe, dolne albo górne ograniczenie szukanej wartości minimaksowej m, tj. albo $m \geqslant ab \geqslant \beta$, albo $m \leqslant ab < \beta$, gdzie ab to wynik wywołania;
- algorytmy z rodziny MTD (Memory-enhanced Test Driver) używają serii takich wywołań do stopniowego zawężania przedziału w którym mieści się szukana wartość minimaksowa;
- przy czym zawartość tablicy transpozycji nie jest czyszczona pomiędzy kolejnymi wywołaniami.
- MTD(f) jest przedstawicielem rodziny MTD, który jako β dla kolejnego wywołania używa wyniku poprzedniego.

MTD(f) i rodzina algorytmów MTD [Plaat1996]

- *Memory-enhanced Test (MT)* to używające tablicy transpozycji wywołanie fail-soft α-β w zerowym oknie, tj. dla takich wartości α, β, że nie istnieje wartość wypłaty a taka, że $\alpha < a < \beta$ (różnica między α i β równa jest najmniejszej możliwej, niepodzielnej jednostce wypłaty);
- zgodnie z twierdzeniem o wynikach algorytmu α-β, takie wywołanie pozwala określić nowe, dolne albo górne ograniczenie szukanej wartości minimaksowej m, tj. albo $m \geqslant ab \geqslant \beta$, albo $m \leqslant ab < \beta$, gdzie ab to wynik wywołania;
- algorytmy z rodziny MTD (Memory-enhanced Test Driver) używają serii takich wywołań do stopniowego zawężania przedziału w którym mieści się szukana wartość minimaksowa;
- przy czym zawartość tablicy transpozycji nie jest czyszczona pomiędzy kolejnymi wywołaniami.
- MTD(f) jest przedstawicielem rodziny MTD, który jako β dla kolejnego wywołania używa wyniku poprzedniego.

MTD(f) i rodzina algorytmów MTD [Plaat1996]

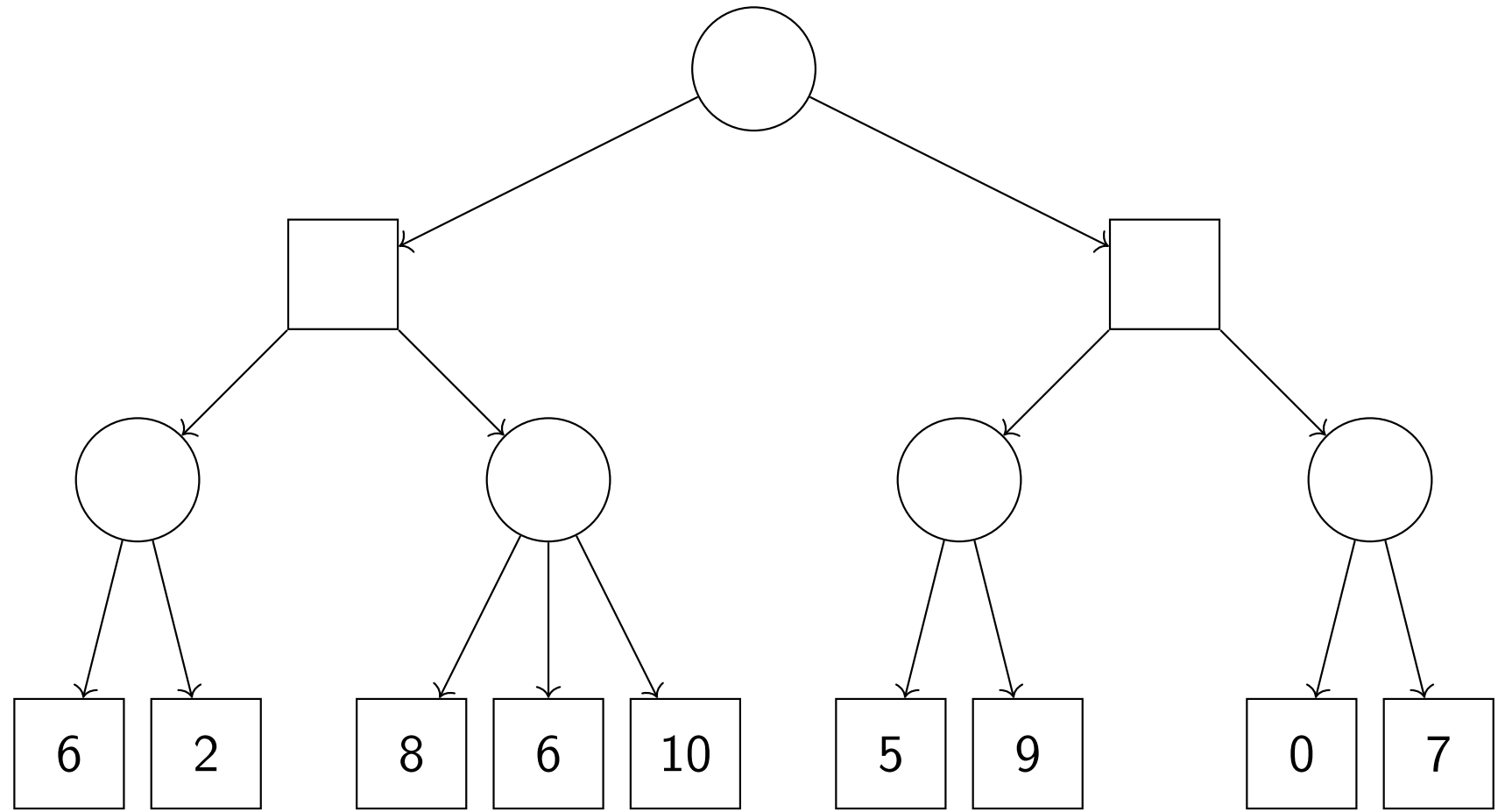
- *Memory-enhanced Test (MT)* to używające tablicy transpozycji wywołanie fail-soft α-β w zerowym oknie, tj. dla takich wartości α, β, że nie istnieje wartość wypłaty a taka, że $\alpha < a < \beta$ (różnica między α i β równa jest najmniejszej możliwej, niepodzielnej jednostce wypłaty);
- zgodnie z twierdzeniem o wynikach algorytmu α-β, takie wywołanie pozwala określić nowe, dolne albo górne ograniczenie szukanej wartości minimaksowej m, tj. albo $m \geqslant ab \geqslant \beta$, albo $m \leqslant ab < \beta$, gdzie ab to wynik wywołania;
- algorytmy z rodziny MTD (Memory-enhanced Test Driver) używają serii takich wywołań do stopniowego zawężania przedziału w którym mieści się szukana wartość minimaksowa;
- przy czym zawartość tablicy transpozycji nie jest czyszczona pomiędzy kolejnymi wywołaniami.
- MTD(f) jest przedstawicielem rodziny MTD, który jako β dla kolejnego wywołania używa wyniku poprzedniego.

Pseudokod algorytmu MTD(f)

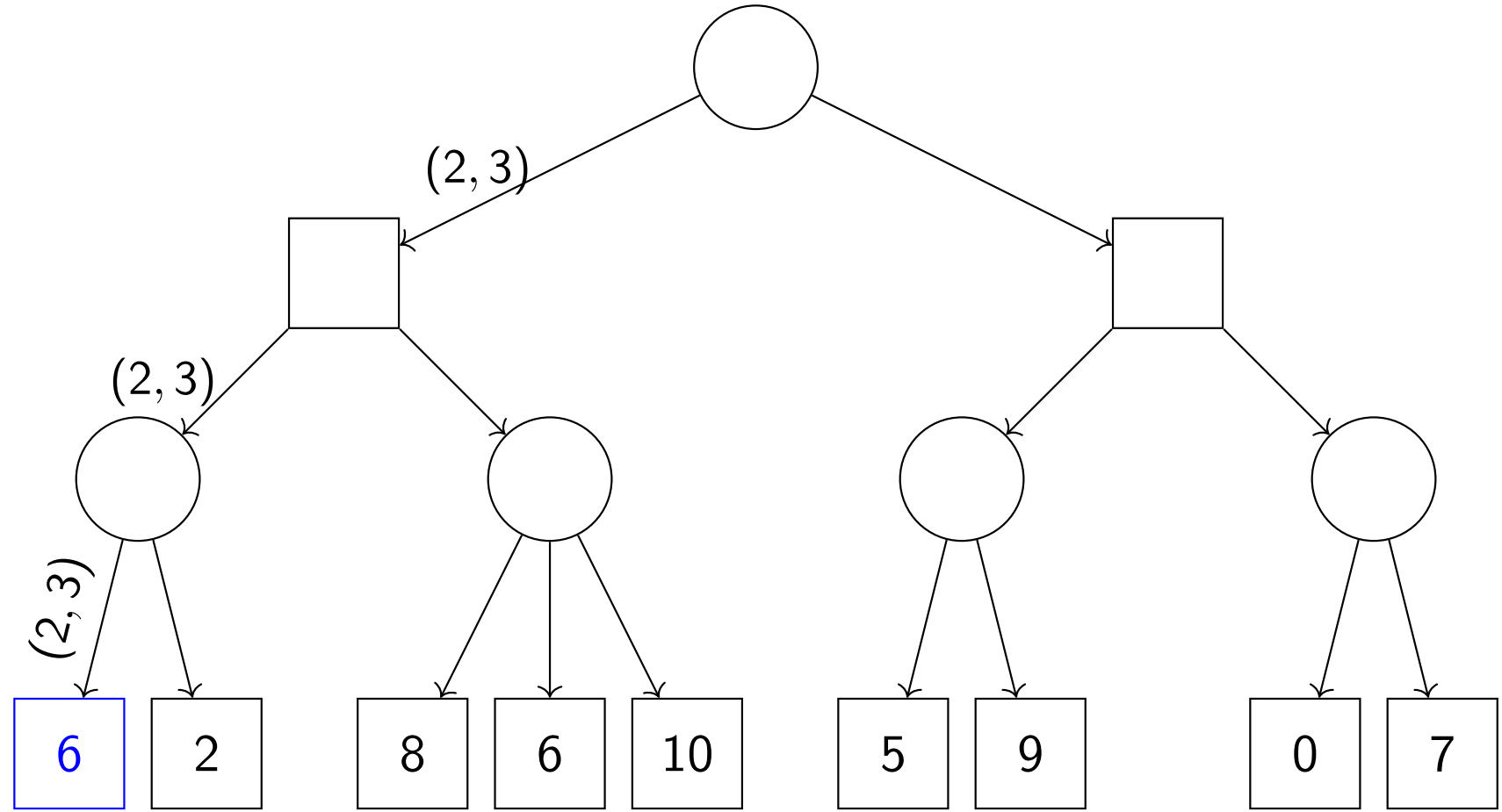
Poniższy algorytm zwraca wartość minimaksową pozycji s. *firstbeta* to wartość β dla pierwszego wywołania *AlfaBeta*. δ to najmniejsza, niepodzielna jednostka wypłaty.

```
1  fun MTDF(s, firstbeta = 0.5):
2      lowerBound ← 0 // dolne ograniczenie MinMax(s)
3      upperBound ← 1 // górne ograniczenie MinMax(s)
4      ab ← firstbeta
5      while lowerBound < upperBound:
6          β ← max(ab, lowerBound + δ)
7          ab ← AlfaBeta(s, β − δ, β)
8          if ab < β:
9              upperBound ← ab
10         else:
11             lowerBound ← ab
12     return ab
```

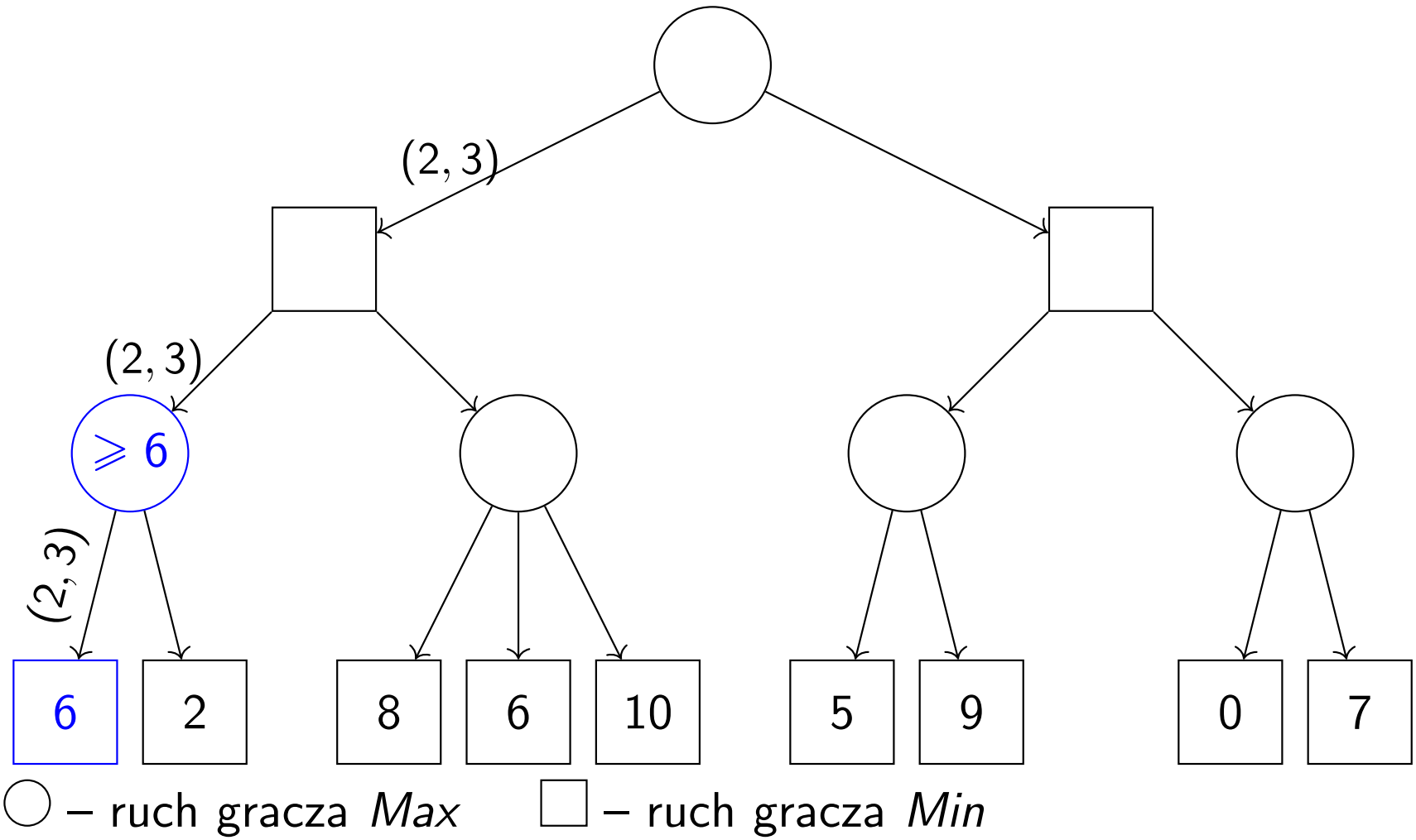
Przykład: Prześledźmy działanie, wywołanego z (zerowym) oknem $(2, 3)$ algorytmu α-β na następującym grafie gry:



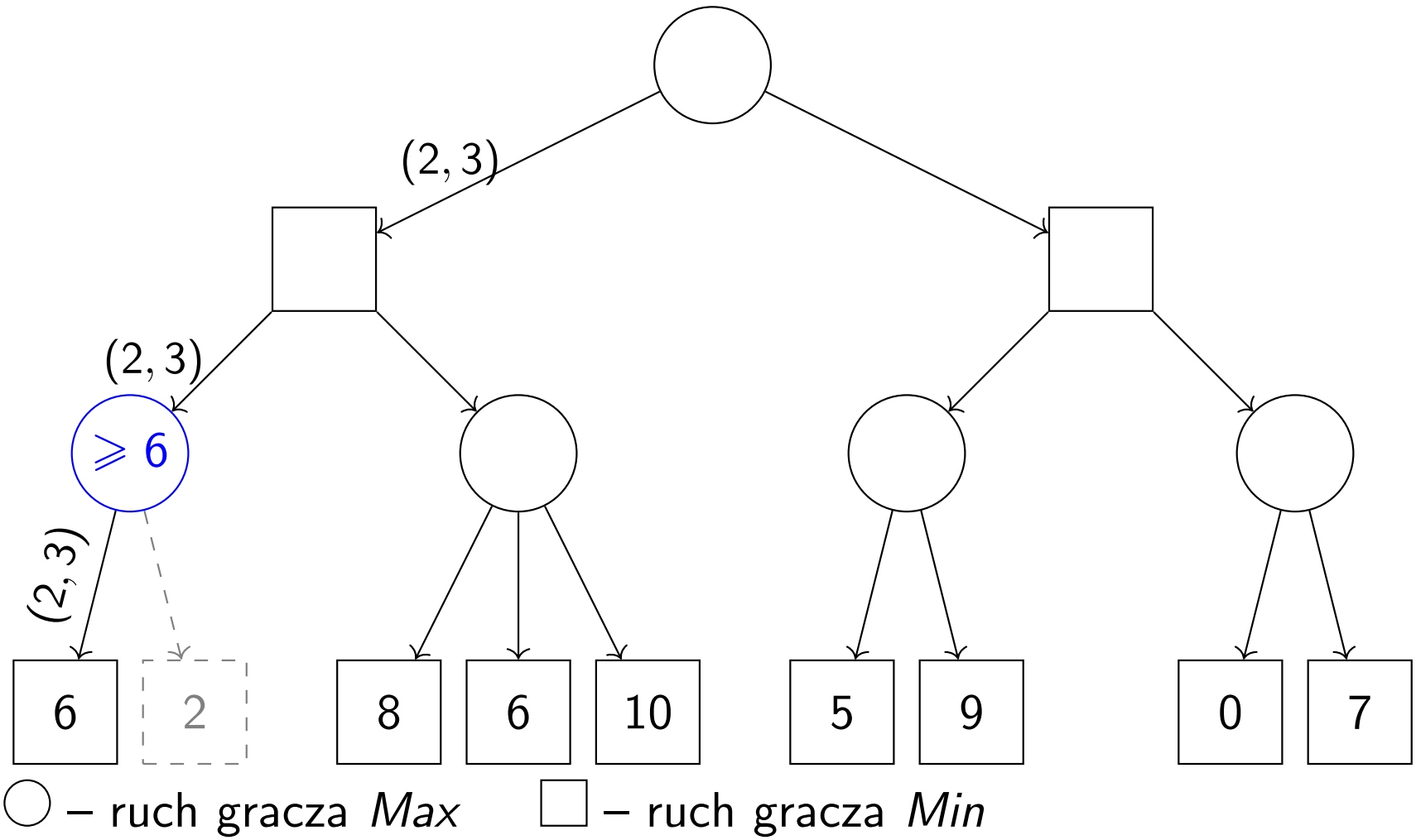
Przy użyciu początkowego okna $(2, 3)$, następują rekurencyjne wywołania dla kolejnych, pierwszych (od lewej) następników, aż do osiągnięcia pozycji końcowej o wartości 6.
Uwaga: Ponieważ okno $(2, 3)$ jest zerowe, to nie może być dalej zawężone i wszystkie wywołania będą w takim oknie.



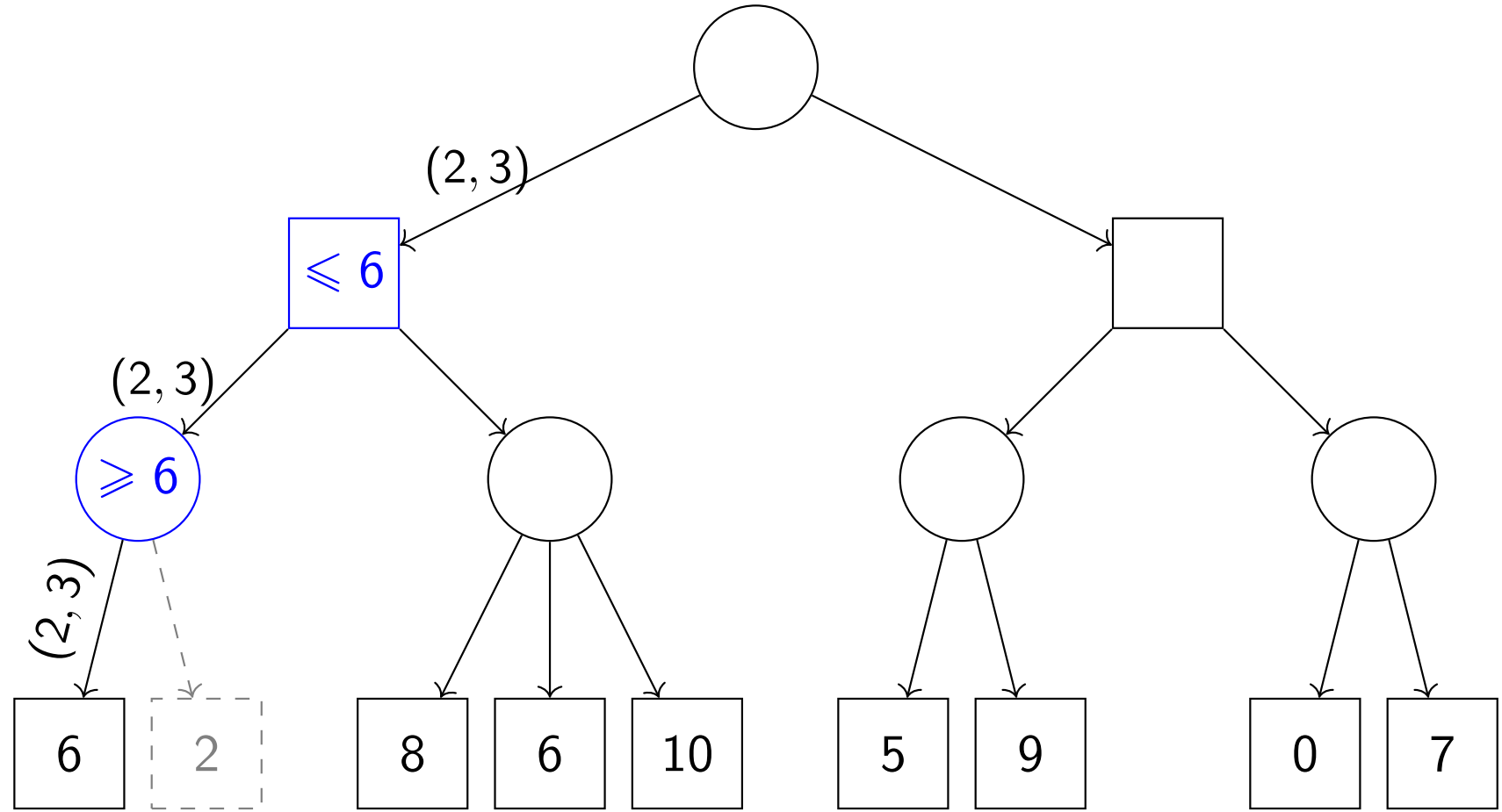
6 staje się bieżącą wartością poprzednika osiągniętej pozycji końcowej. Poprzednik ten jest typu MAX, więc jego ostateczna wartość wyniesie 6 lub więcej.



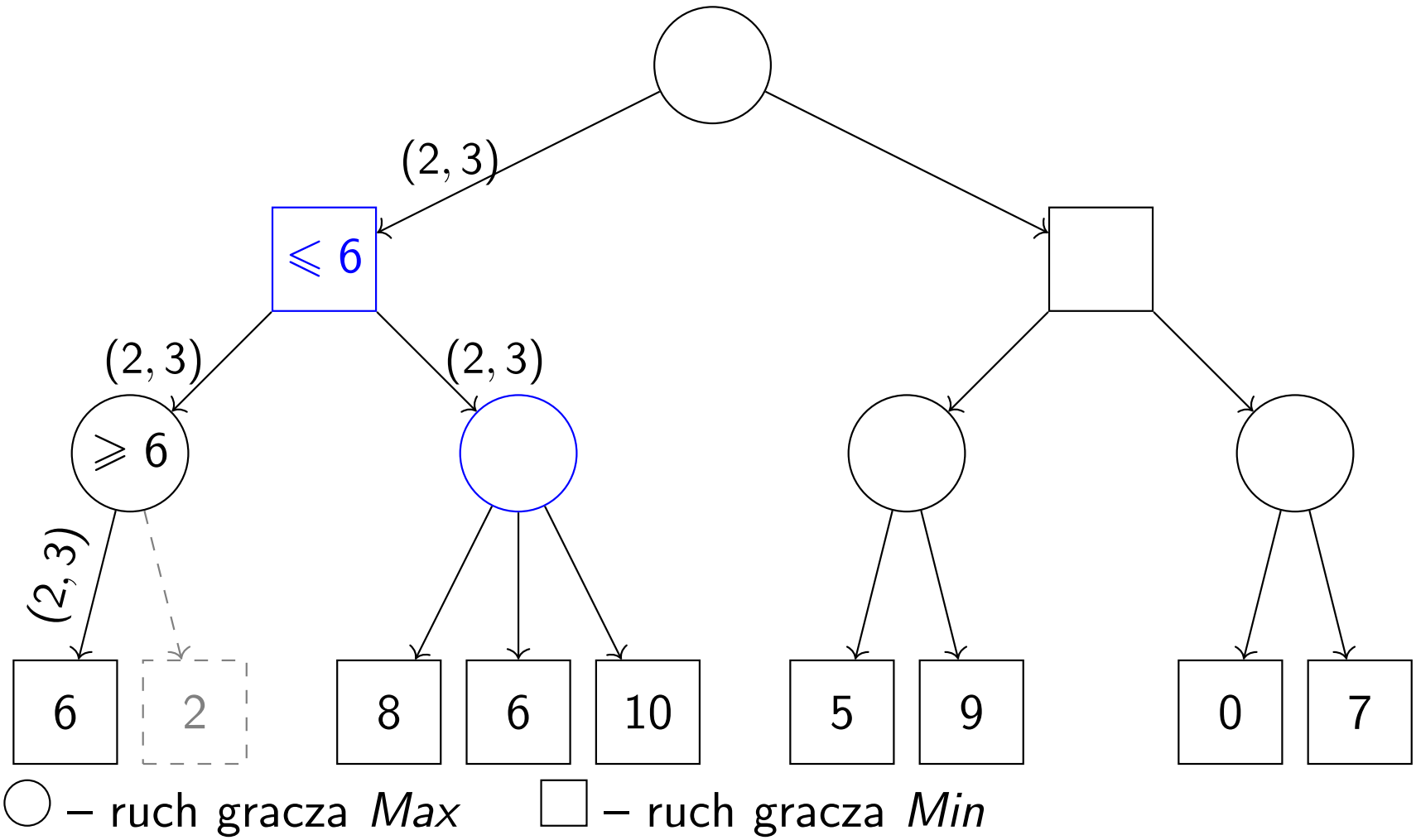
Wartość $\geqslant 6$ nie zmieścić w oknie $(2, 3)$, tj. $[6, \infty) \cap (2, 3) = \emptyset$.
6 przekroczy okno z prawej strony. Drugi następnik nie będzie więc badany, zostanie odcięty.



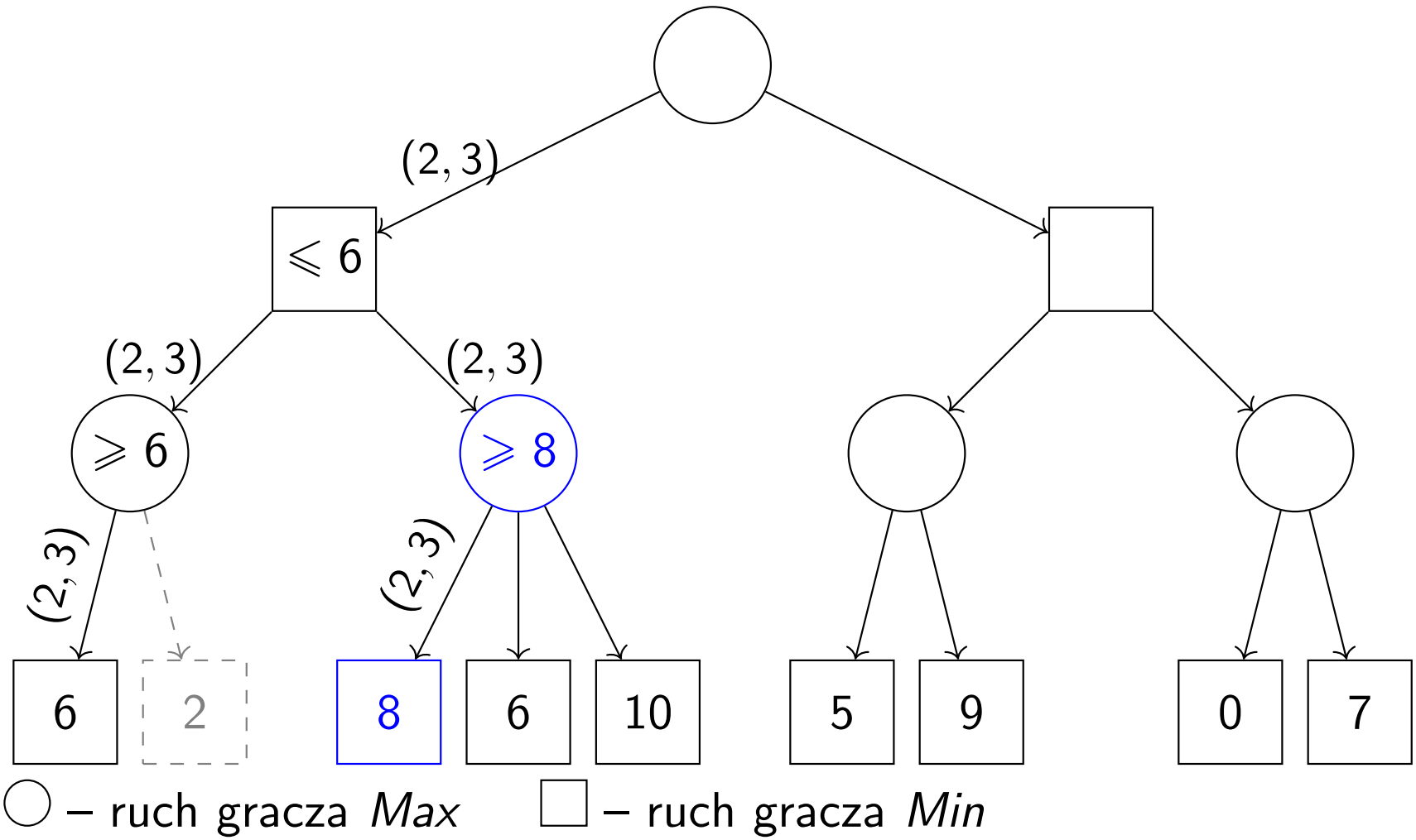
6 staje się bieżącą wartością poprzednika zbadanej pozycji. Poprzednik ten jest typu MIN, więc jego ostateczna wartość wyniesie 6 lub mniej. Nie wiadomo, po której stronie okna znajdzie się ta ostateczna wartość. $(-\infty, 6] \cap (2, 3) \neq \emptyset$. Nie można więc odciąć drugiego następnika.



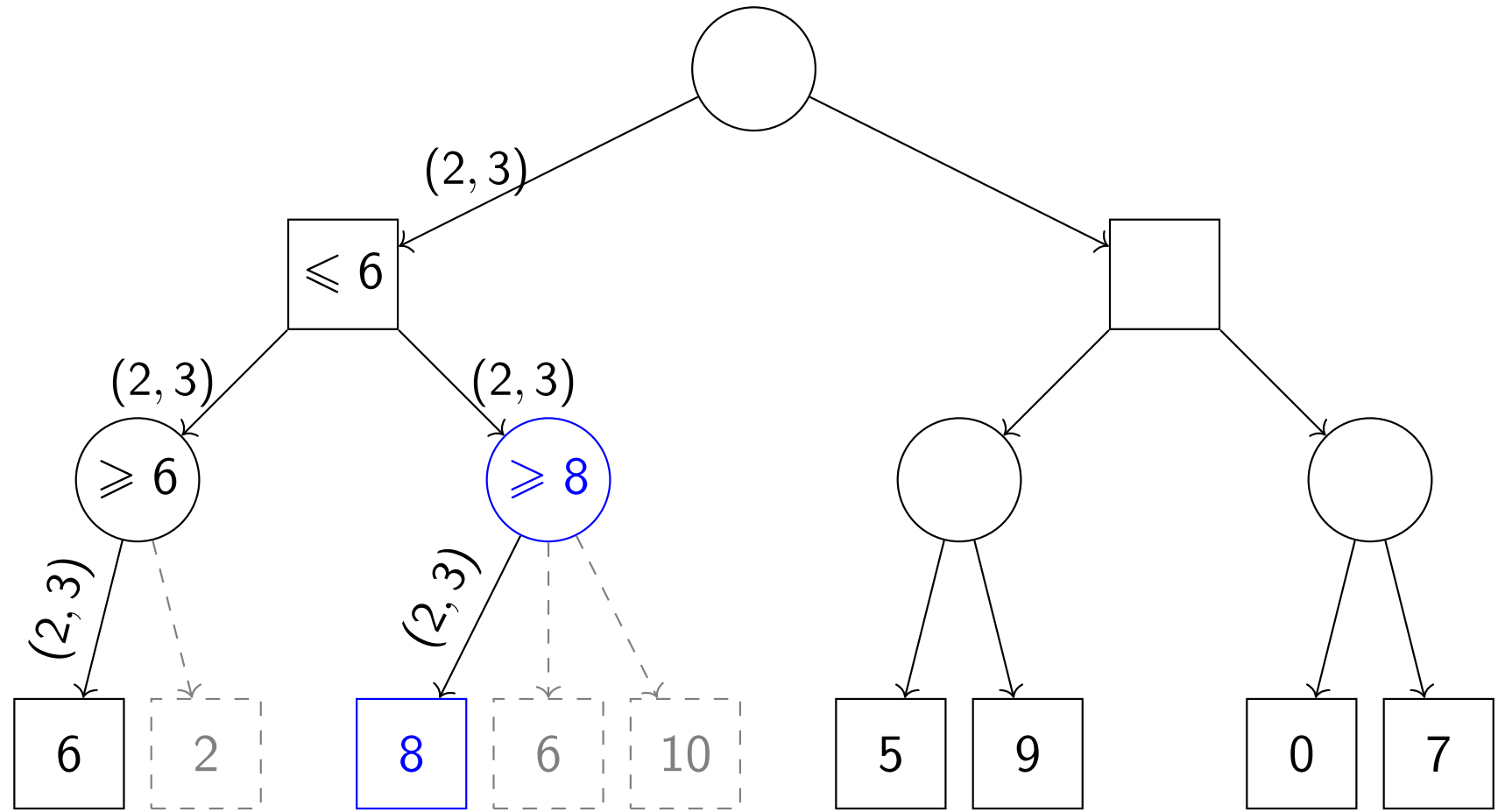
Dla drugiego następnika algorytm wywołuje się w oknie $(-\infty, 6) \cap (2, 3) = (2, 3)$.



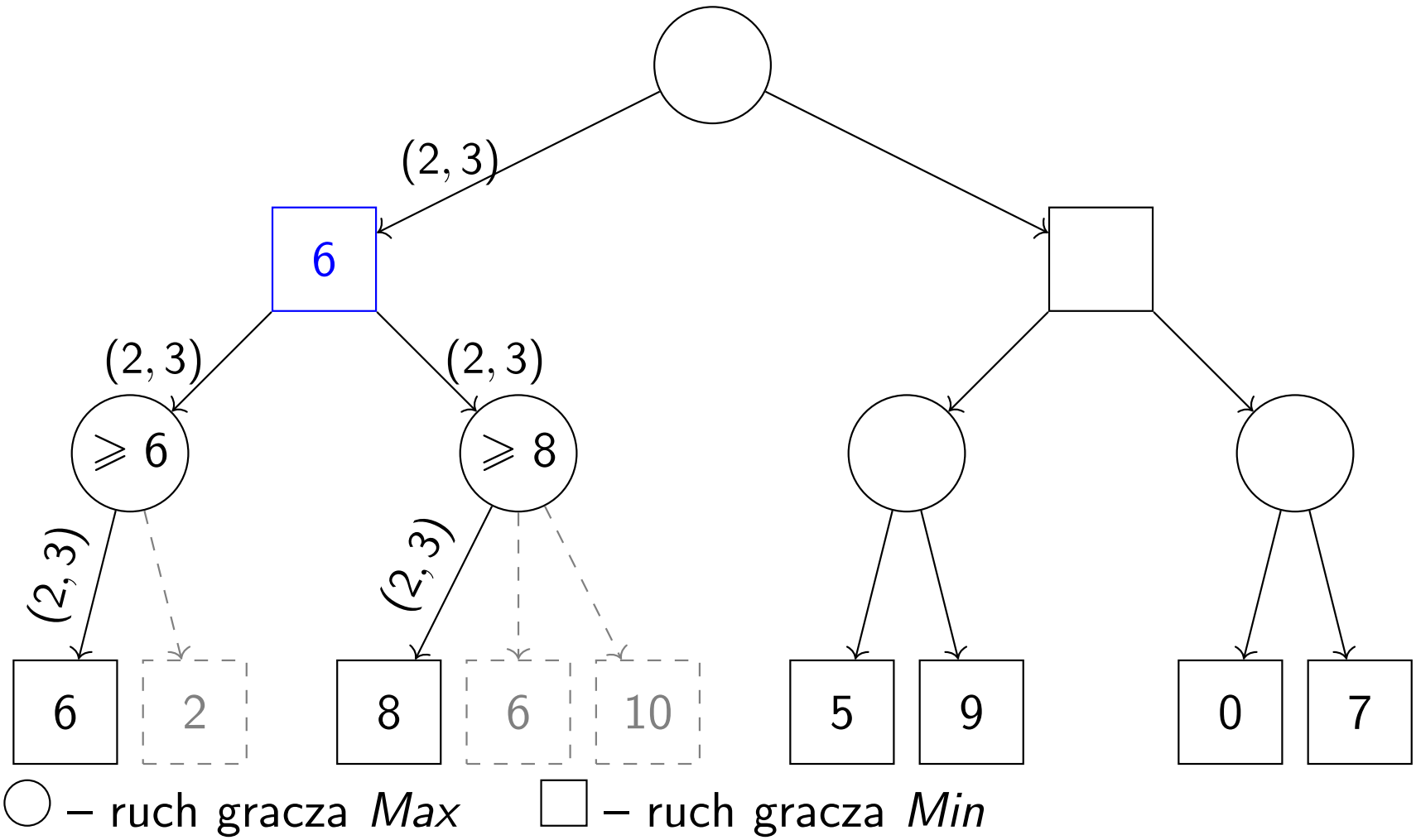
I w tym samym oknie $(2, 3)$ wywołuje się dla jego następnika o wartość 8. 8 staje się jego bieżącym maksimum, zaś jego ostateczna wartość wyniesie $\geqslant 8$.



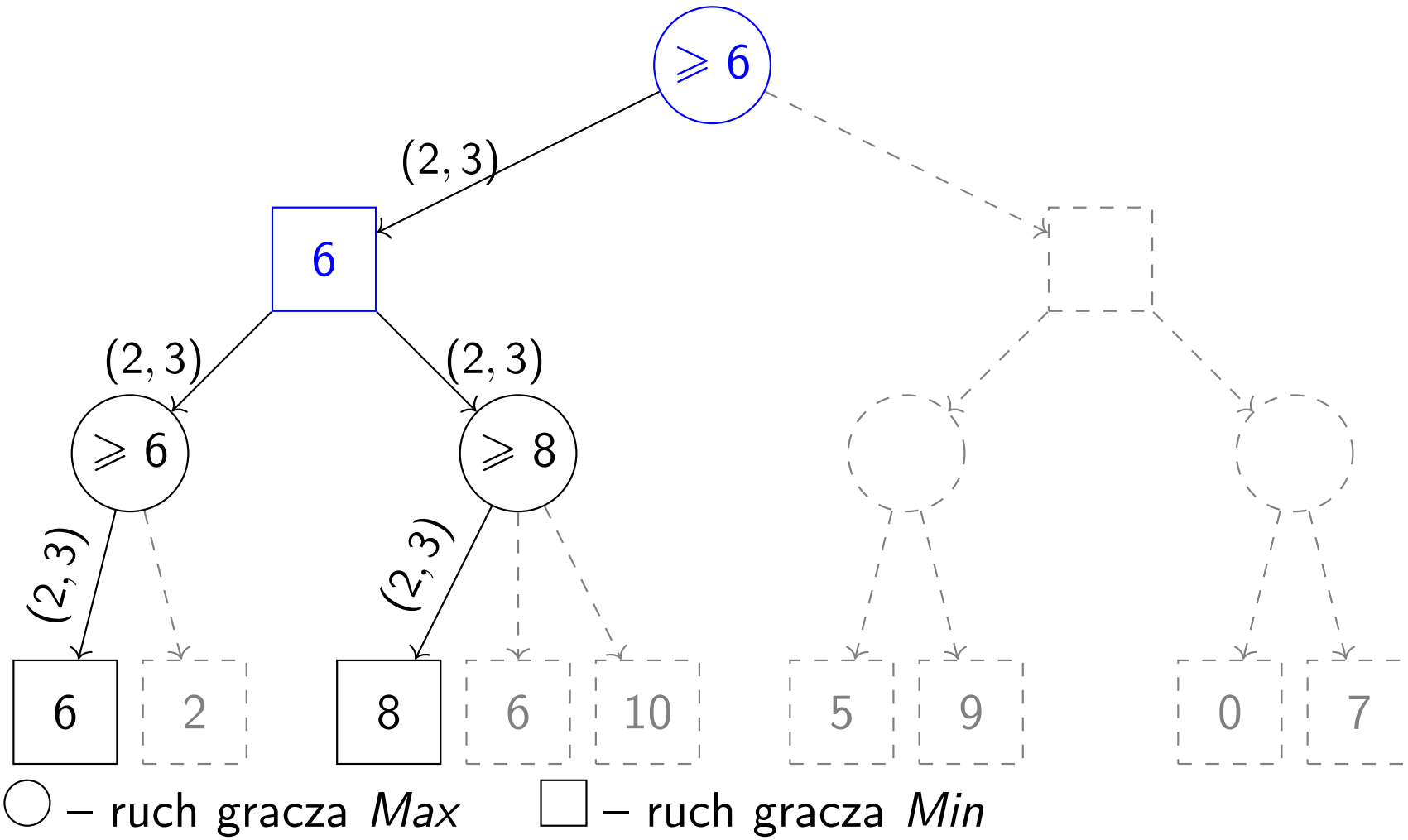
Ta ostateczna wartość $\geqslant 8$ nie zmieści się w oknie $(2, 3)$, tj. $[8, \infty) \cap (2, 3) = \emptyset$. 8 przekroczy okno z prawej strony. Następuje więc odcięcie kolejnych następników. Bieżące maksimum 8 staje się ostateczną wartością (zwróconą z) zbadanego węzła.



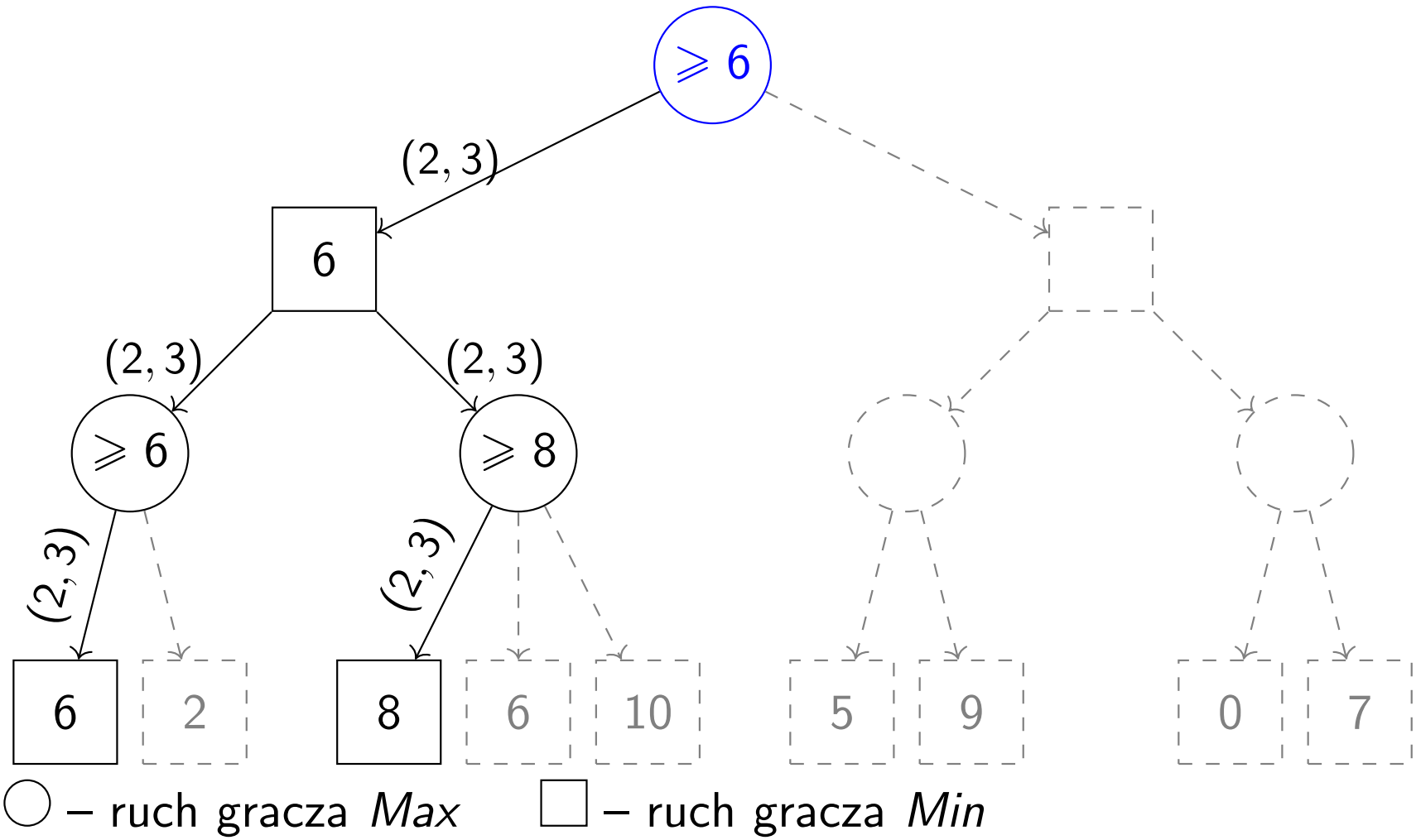
Ta 8 nie poprawia bieżącego minimum 6 lewego następnika korzenia. 6 staje się zatem jego ostateczną wartością, bo nie ma on więcej dzieci.



6 staje się bieżącą wartością korzenia. Korzeń jest typu MAX, więc jego ostateczna wartość wyniesie 6 lub więcej. Ta ostateczna wartość nie zmieści się w oknie $(2, 3)$, tj. $[6, \infty) \cap (2, 3) = \emptyset$; przekroczy to okno z prawej strony. Następuje więc odcięcie prawego następnika korzenia.



Ostatecznie więc algorytm α-β wywołany w oknie $(2, 3)$ zwróci 6. Ponieważ $3 \leqslant 6$, to zgodnie z twierdzeniem o wynikach algorytmu α-β, wartość minimaksowa m korzenia spełnia $6 \leqslant m$, tj. wynosi 6 lub więcej.



Bibliografia

- **Donald E. Knuth, Ronald W. Moore** *An Analysis of Alpha-Beta Pruning*, Artificial Intelligence 6/1975
- **A. Plaat, J. Schaeffer, W. Pijls, A. de Bruin** *Best-first fixed-depth minimax algorithms*, Artificial Intelligence, 87(1–2):255–293, 1996
- **P. Beling** *Szybkie algorytmy decyzyjne w grach z doskonałą informacją i ich wykorzystanie w grach jej pozbawionych*, rozprawa doktorska, Uniwersytet Łódzki, 2016