Algorytm obejścia drzewa poszukiwań i zadanie o hetmanach szachowych

Alexander Denisjuk

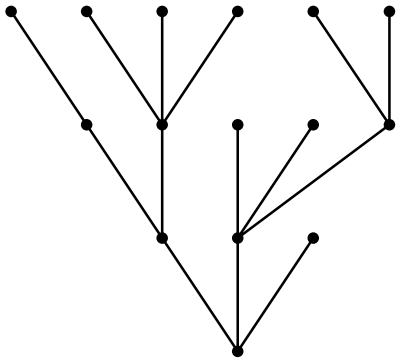
Prywatna Wyższa Szkoła Zawodowa w Giżycku

2005 rok

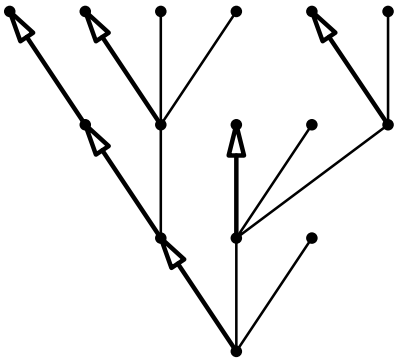
denisjuk@pwsz.net

1. Obejście drzewa poszukiwań (szukanie z powracaniem, backtraking)

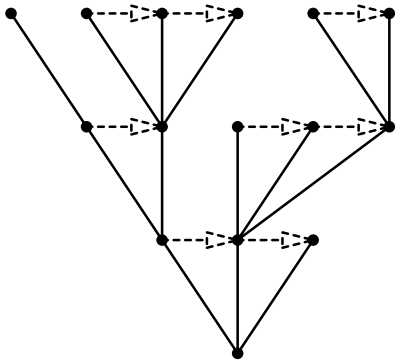
Definicje 1. *Drzewem poszukiwań,* o n wierzchołkach nazywa się graf pusty, jeżeli $n = 0$, oraz, przy $n > 1$, graf, mający wierzchołek W, nazywany *korzeniem*, oraz skończoną (być może pustą) uporządkowaną ilość *poddrzew* poszukiwań $t_1, \ldots, t_k$ o $n_1, \ldots, n_k$ wierzchołkach odpowiednio. Przy czym $n_1 + \ldots + n_k + 1 = n$, (v. rysunek 1). Wierzchołek, który nie ma poddrzew, nazywa się *liściem*.



Rysunek 1. Drzewo poszukiwań



Rysunek 2. Polecenie `Up`

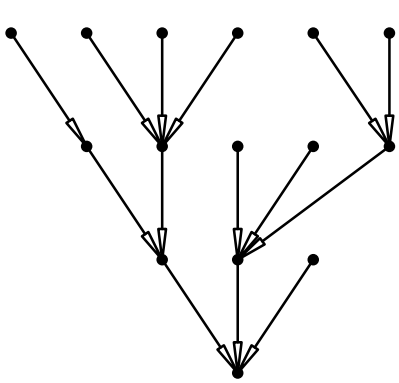


Rysunek 3. Polecenie `Right`

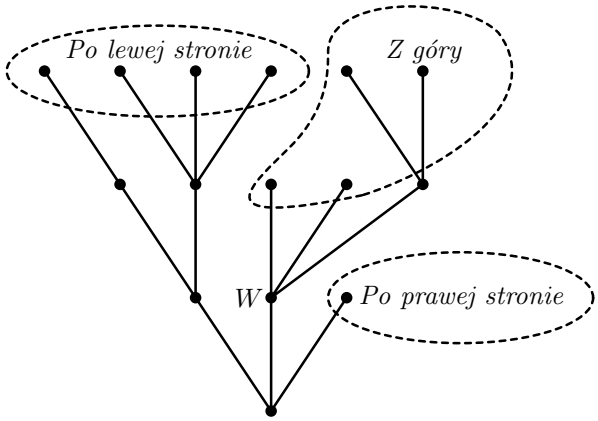
Zadanie 2. Dane jest drzewo poszukiwań. Wydrukować wszystkie jego liście dokładnie jeden raz.

Rozwiązanie. Założmy, że obejście drzewa wykonuje Robot, znajdujący się na początku w korzeniu. Użyjmy następujących poleceń:

1. `Up` — przejście po pierwszej z wychodzących krawędzi (rysunek 2).
2. `Right` — przejście do sąsiedniego po prawej stronie wierzchołka (rysunek 3).
3. `Down` — zejście o jeden poziom w dól (rysunek 4).
4. `IsUp` — ma wartość **prawda**, jeśli istnieje wierzchołek z góry i **fałsz**, jeśli z góry nie ma wierzchołka.
5. `IsRight` — ma wartość **prawda**, jeśli istnieje wierzchołek z prawej strony i **fałsz**, jeśli z prawej strony nie ma wierzchołka.
6. `IsDown` — ma wartość **prawda**, jeśli istnieje wierzchołek z dołu i **fałsz**, jeśli z dołu nie ma wierzchołka.



Rysunek 4. Polecenie `Down`



Rysunek 5. Klasy liści

Niech Robot będzie w pewnym wierzchołku W. Podzielmy wszystkie liście na klasy (rysunek 5):

(1.) *Po lewej stronie* — ścieżka z korzenia do liścia skręca w lewo przed wierzchołkiem W.
(2.) *Po prawej stronie* — ścieżka z korzenia do liścia skręca w prawo przed wierzchołkiem W.
(3.) *Z góry* — ścieżka z korzenia do liścia idzie przez wierzchołek W.

W szczególności, jeżeli W jest liściem, to on należy do klasy *„z góry"*. Dla korzenia klasy *„po lewej stronie"* oraz *„po prawej stronie"* są puste. Natomiast klasa *„z góry"* zgadza się ze zbiorem wszystkich liści.

Zostanie potrzebna procedura `UpWork`, algorytm 1.

Dane: Wydrukowane są wszystkie liście *po lewej stronie*
Wyniki: Wydrukowane są wszystkie liście *po lewej stronie* i *z góry*
{*Niezmiennik: Wydrukowane są wszystkie liście* po lewej stronie}
while `IsUp` **do**
  `Up`
end while
{*Wydrukowane są wszystkie liście* po lewej stronie, *Robot jest w liściu*}
`Work`

Algorytm 1. Procedura `UpWork`

Program główny dany jest w algorytmie 2

Dane: Robot jest w korzeniu, liście nie wydrukowane
Wyniki: Robot jest w korzeniu, liście są wydrukowane
{*Wydrukowane są wszystkie liście* po lewej stronie}
`UpWork`
{*Niezmiennik: Wydrukowane są wszystkie liście* po lewej stronie *i* z góry}
while `IsDown` **do**
  if `IsRight` **then**
    `Right`
    {*Wydrukowane są wszystkie liście* po lewej stronie}
    `UpWork`
  else {*Nie ma z prawej, jest z dołu*}
    `Down`
  end if
end while
{*Robot jest w korzeniu, więc wydrukowane są wszystkie liście*}

Algorytm 2. Wydrukowanie wszystkich liści drzewa poszukiwań

Zadanie 3. Wydrukować *wszystkie* wierzchołki drzewa poszukiwań dokładnie jeden raz.

Rozwiązanie. Niech Robot będzie w pewnym wierzchołku W. Podzielmy wszystkie wierzchołki na klasy (rysunek 6):

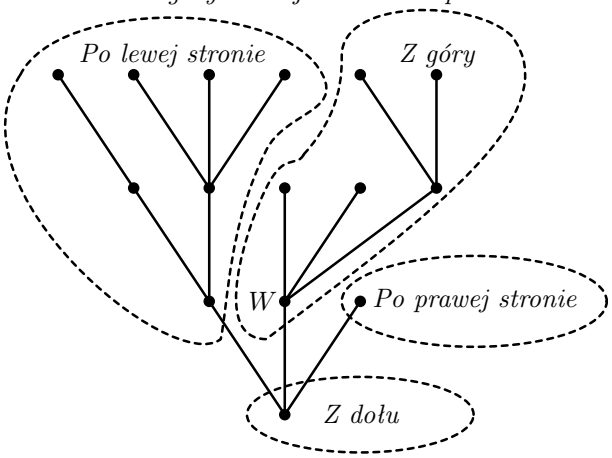
(1.) *Z dołu* — ścieżka z korzenia do wierzchołka W przechodzi przez dany wierzchołek.
(2.) *Po lewej stronie* — ścieżka z korzenia do wierzchołka skręca w lewo przed wierzchołkiem W.
(3.) *Po prawej stronie* — ścieżka z korzenia do wierzchołka skręca w prawo przed wierzchołkiem W.
(4.) *Z góry* — ścieżka z korzenia do wierzchołka idzie przez wierzchołek W.

Modyfikacja procedury pomocniczej (cf. 1) dana jest w algorytmie 3.
Program główny podobny jest do algorytmu 2 i podany jest w algorytmie 4.

Uwaga 4. Ilość działań w algorytmie jest $O(n)$, gdzie n jest ilością wierzchołków.

Zadanie 5. Na szachownicy $n \times n$ ustawić n hetmanów w taki sposób, żeby żadne dwa hetmany nie atakowały się. Wydrukować wszystkie ustawienia.

Rozwiązanie. Na każdej z n poziomnic powinien znajdować się jeden hetman. Nazwiemy k-stanem dowolne ustawienie k hetmanów na dolnych k poziomnicach. Określmy *drzewo stanów* (v. rysunek 7). Korzeniem będzie *stan pusty.* Z każdego k-stanu wychodzi n krawędzi do $k+1$ stanów, odpowiadającym umieszczeniu hetmana numer $k+1$ na poziomnicy $k+1$. Uporządkujmy $(k+1)$-stany według numeru *pionownicy,* w której został umieszczony $(k+1)$-ty



Rysunek 6. Klasy wierzchołków

Dane: Wydrukowane są wszystkie wierzchołki *po lewej stronie* i *z dołu*
Wyniki: Wydrukowane są wszystkie wierzchołki *po lewej stronie*, *z dołu* i *z góry*
{*Niezmiennik: Wydrukowane są wszystkie wierzchołki* po lewej stronie *i* z dołu}
while `IsUp` **do**
`Work`
`Up`
end while
{*Wydrukowane są wszystkie wierzchołki* po lewej stronie *i* z dołu, *Robot jest w liściu*}
`Work`
{*Wydrukowane są wszystkie wierzchołki* po lewej stronie, z dołu *i* z góry}

Algorytm 3. Modyfikowana procedura `UpWork`

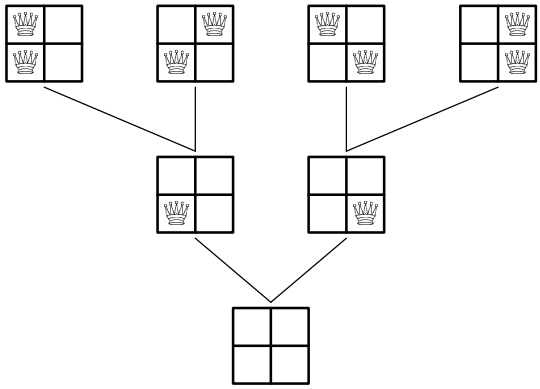
Dane: Robot jest w korzeniu, wierzchołki nie wydrukowane
Wyniki: Robot jest w korzeniu, wierzchołki są wydrukowane
{*Wydrukowane są wszystkie wierzchołki* po lewej stronie *i* z dołu}
`UpWork`
{*Niezmiennik: Wydrukowane są wszystkie wierzchołki* po lewej stronie, z dołu *i* z góry}
while `IsDown` **do**
if `IsRight` **then**
`Right`
{*Wydrukowane są wszystkie wierzchołki* po lewej stronie *i* z dołu}
`UpWork`
else {*Nie ma z prawej, jest z dołu*}
`Down`
end if
end while
{*Robot jest w korzeniu, więc wydrukowane są wszystkie wierzchołki*}

Algorytm 4. Wydrukowanie wszystkich wierzchołków drzewa poszukiwań

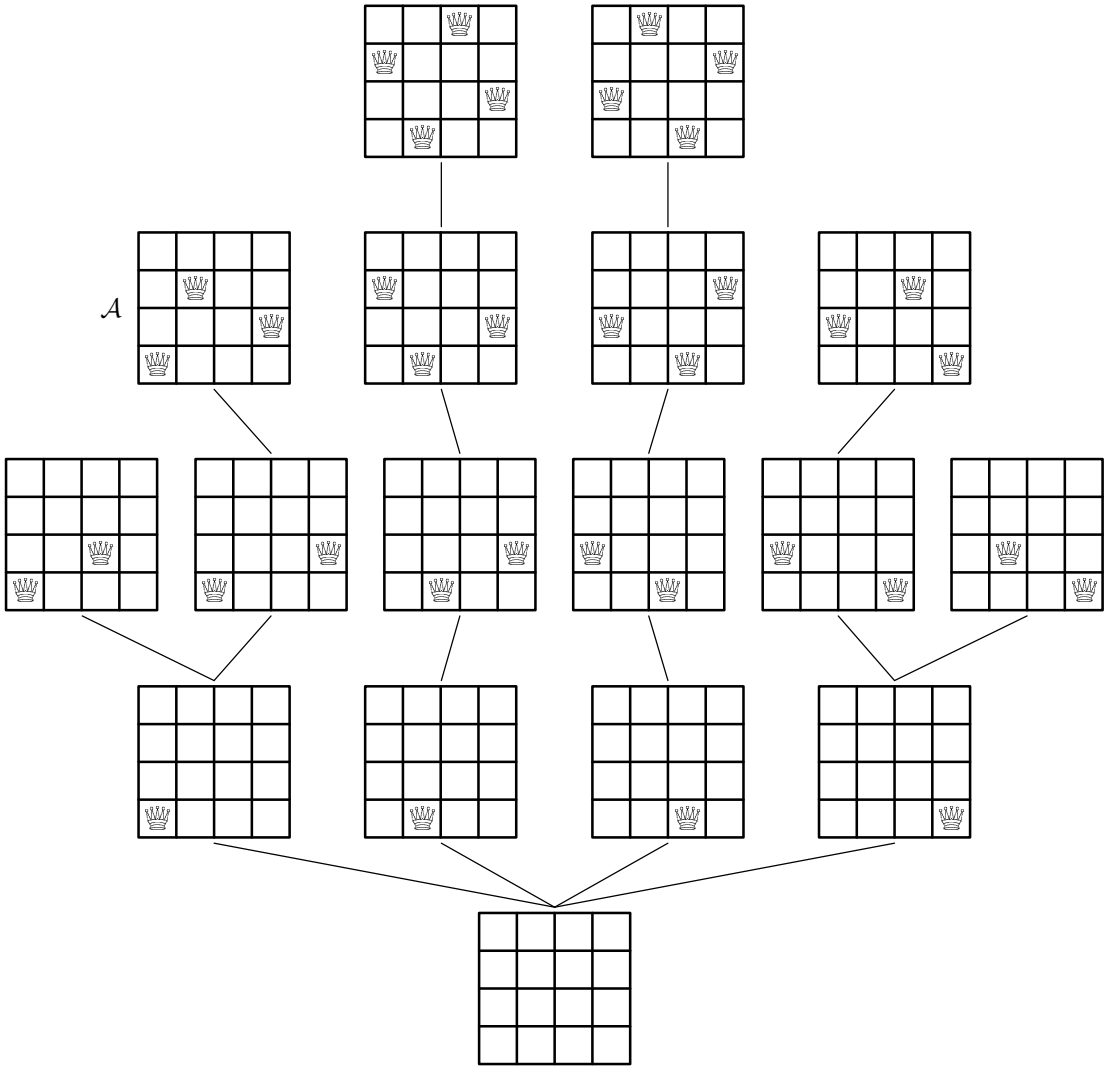
hetman. Otrzymany graf będzie drzewem poszukiwań i można zastosować program obejścia drzewa 2, wybierając do druku tylko te liście, w których hetmany się nie atakują.

Drzewo poszukiwania, określone w poprzednim akapicie, ma n^n liści i $\frac{n^{n+1}-1}{n-1}$ wierzchołków. W taki sposób, można spodziewać się algorytmu o skomplikowalności $O(n^n)$. Można powiększyć efektywność algorytmu, zmniejszywszy drzewo. Zauważmy, że jeżeli w k-stanie dwa hetmany się atakują, to nie ma sensu kontynuować poszukiwanie po tej gałęzi. Nazwiemy k-stan *dopuszczalnym,* jeżeli hetmany się nie atakują. Rozważmy drzewo, złożone tylko z *dopuszczanych* stanów (v. rysunek 8).

Przedstawmy stan za pomocą zmiennej k, ilości ustawionych hetmanów, oraz tablicy P_i, $i = 1, \ldots n$, gdzie P_i jest numerem pionownicy hetmana numer i, ustanowionego w poziomnicy i. Na przykład, stanowi $\mathcal{A}$ na rysunku 8 odpowiada $k = 3$, $P = (1, 4, 2, *)$, w korzeniu $k = 0$. Przy $i > k$ wartość P_i nie jest istotną.

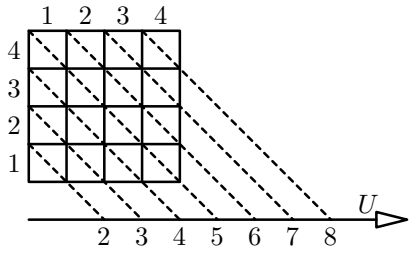


RYSUNEK 7. Drzewo stanów w zadaniu o hetmanach szachowych dla $n = 2$

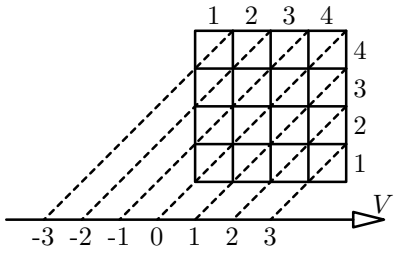


RYSUNEK 8. Drzewo stanów dopuszczalnych w zadaniu o hetmanach szachowych dla $n = 4$

Wprowadzimy trzy tablice pomocnicze H_i, $i = 1, \ldots, n$, U_i, $i = 2, \ldots 2n$ oraz V_j, $j = 1 - n, \ldots, n - 1$. H_i ma wartość **prawda**, jeżeli na i-ej pionownicy znajduje się hetman, oraz **fałsz**, jeżeli i-ta pionownica jest wolna. U_i ma wartość **prawda**, jeżeli na i-ej *prawej przekątnej* znajduje się hetman, oraz **fałsz**, jeżeli i-ta prawa przekątna jest wolna. Hetman, umieszczony na poziomnicy y i pionownicy x, znajduje się na prawej przekątnej $x+y$, v. rysunek 9. V_i ma wartość **prawda**, jeżeli na i-ej *lewej przekątnej* znajduje się hetman, odpowiednio **fałsz**, jeżeli i-ta lewa przekątna



Rysunek 9. Prawe przekątne



Rysunek 10. Lewe przekątne

jest wolna. Hetman, umieszczony na poziomnicy y i pionownicy x, znajduje się na lewej przekątnej $x-y$, v. rysunek 10.

Tablice H, U i V pomogą sprawdzić dopuszczalność stanu. Na przykład, dla stanu $\mathcal{A}$ na rysunku 8 tablicy te równe są odpowiednio

$$H = (\textbf{prawda}, \textbf{prawda}, \textbf{fałsz}, \textbf{prawda}),$$
$$U = (\textbf{prawda}, \textbf{fałsz}, \textbf{fałsz}, \textbf{prawda}, \textbf{prawda}, \textbf{fałsz}, \textbf{fałsz}),$$
$$V = (\textbf{fałsz}, \textbf{fałsz}, \textbf{prawda}, \textbf{prawda}, \textbf{fałsz}, \textbf{prawda}, \textbf{fałsz}).$$

W korzeniu wszystkie wartości tych tablic równe są **fałsz**.

Do rozwiązania zadania zastosujemy algorytm 2. Główny program, łącznie z działaniami wstępnymi, przedstawiony jest w algorytmie 5.

Dane: $n > 0$
Wyniki: wydrukowane są wszystkie rozwiązania zadania o hetmanach
$k \leftarrow 0$
$U_2 \leftarrow$ **fałsz**, ... , $U_{2n} \leftarrow$ **fałsz**
$V_{1-n} \leftarrow$ **fałsz**, ... , $V_{n-1} \leftarrow$ **fałsz**
$H_1 \leftarrow$ **fałsz**, ... , $H_n \leftarrow$ **fałsz**
`algorytm 2`

Algorytm 5. Program główny do zadania o hetmanach

Do implementacja procedur `IsDown` oraz `Down` zauważmy, że poziom Robota na drzewie określony jest poprzez wartość zmiennej k (ilości umieszczonych hetmanów). W szczególności, w korzeniu $k = 0$. Implementacja procedur dana jest w algorytmach 6 i 7.

Wyniki: `IsDown` = **fałsz** $\iff$ Robot jest w korzeniu
if $k = 0$ **then**
`IsDown` $\leftarrow$ **fałsz**
else $\{k > 0\}$
`IsDown` $\leftarrow$ **prawda**
end if

Algorytm 6. Procedura `IsDown` do zadania o hetmanach

Dane: Robot jest w dopuszczalnym stanienie, nie w korzeniu ($k > 0$)
Wyniki: Robot jest w dopuszczalnym stanie o jeden poziom niżej
$x \leftarrow P_k$; $y \leftarrow k$
$H_x \leftarrow$ **fałsz**
$U_{x-y} \leftarrow$ **fałsz**
$V_{x+y} \leftarrow$ **fałsz**
$k \leftarrow k - 1$

Algorytm 7. Procedura `Down` do zadania o hetmanach

W procedurze `IsUp` trzeba dobrać pionownicę dla kolejnego hetmana. W szczególności, jeżeli $k = n$, i.e. wszystkie hetmany zostały umieszczone, `IsUp` powinno być **fałsz**. Jeżeli $k < n$, to trzeba dobrać taką wartość P_{k+1} między 1

Dane: Robot jest w dopuszczalnym k-stanie
Wyniki: `IsUp` = **prawda i** numer pionownicy dla kolejnego hetmana zapisany jest w `New` **lub** `IsUp` = **fałsz i** nie ma możliwości umieścić kolejnego hetmana

if $k = n$ **then**
  `IsUp` $\leftarrow$ **fałsz**
else $\{k < n\}$
  `IsUp` $\leftarrow$ **fałsz**
  $y \leftarrow k + 1$
  $x \leftarrow 1$
  $\{$*Niezmiennik:* `IsUp` = **fałsz i** *nie można umieścić hetmana w pinownicach* $1, \ldots, x-1$ **lub** `IsUp` = **prawda i** *można umieścić hetmana w pionownicy* $x\}$
  while $x \leqslant n$ **i** `IsUp` = **fałsz do**
    if H_x = **fałsz i** U_{x-y} = **fałsz i** V_{x+y} = **fałsz then**
      `IsUp` $\leftarrow$ **prawda**
    else $\{$*Pionownica lub jedna z przekątnych nie jest wolna*$\}$
      $x \leftarrow x + 1$
    end if
  end while
  if `IsUp` = **prawda then**
    `New` $\leftarrow x$
  end if
end if

Algorytm 8. Procedura `IsUp` do zadania o hetmanach

Dane: Robot jest w dopuszczalnym k-stanie **i** można umieścić kolejnego hetmana w pinownicy `New`
Wyniki: Robot jest w pierwszym dopuszczalnym $(k+1)$-stanie.

$k \leftarrow k + 1$
$x \leftarrow$ `New`
$y \leftarrow k$
$P_k \leftarrow x$
$H_x \leftarrow$ **prawda**
$U_{x-y} \leftarrow$ **prawda**
$V_{x+y} \leftarrow$ **prawda**

Algorytm 9. Procedura `Up` do zadania o hetmanach

a n, żeby odpowiednia pionownica i dwie przekątne były wolne. Jeżeli takiej wartości nie ma, `IsUp` powinno być **fałsz**. Jeżeli `IsUp` = **fałsz**, wartość P_{k+1} nie jest istotną. Implementacja procedury dana jest w algorytmie 8.

W procedurze `Up` trzeba umieścić kolejnego hetmana w pionownicy P_{k+1}, i. e., oznaczyć pionownicę i obydwie przekątne, jako zajęte i powiększyć poziom Robota w drzewie. Implementacja procedury dana jest w algorytmie 9.

Procedura `IsRight` jest podobna do procedury `IsUp`. Różnica polega na tym, że dobieramy pionownicę dla hetmana w tej samej poziomnicy. Implementacja procedury dana jest w algorytmie 10.

W procedurze `Right` trzeba „zdjąć" hetmana z pozycji (P_k, k) i umieścić jego w pozycji $(\texttt{New}, k)$. Implementacja — algorytm 11.

W procedurze `Work` trzeba sprawdzić, czy zostały umieszczone wszystkie hetmany i, jeżeli tak, to wydrukować rozwiązanie. Implementacja dana jest w algorytmie 12.

Dane: Robot jest w liściu.
Wyniki: Jeżeli liść odpowiada rozwiązaniu, to ono jest wydrukowane

if $k = n$ **then**
  Wydrukować P.
end if

Algorytm 12. Procedura `Work` do zadania o hetmanach

Dane: Robot jest w dopuszczalnym k-stanie
Wyniki: `IsRight` = **prawda i** nowy numer pionownicy dla hetmana k zapisany jest w `New` **lub**
`IsRight` = **fałsz i** nie ma możliwości umieścić hetmana k w tej samej poziomnicy

$y \leftarrow k$
$x \leftarrow P_k + 1$
{*Niezmiennik:* `IsRight` = **fałsz i** *nie można umieścić hetmana w pionownicach* $P_{k+1}, \ldots, x-1$ **lub**
`IsRight` = **prawda i** *można umieścić hetmana w pionownicy* x}
while $x \leqslant n$ **i** `IsRight` = **fałsz do**
if H_x = **fałsz i** U_{x-y} = **fałsz i** V_{x+y} = **fałsz then**
`IsRight` $\leftarrow$ **prawda**
else {*Pionownica lub jedna z przekątnych nie jest wolna*}
$x \leftarrow x + 1$
end if
end while
if `IsRight` = **prawda then**
`New` $\leftarrow x$
end if

Algorytm 10. Procedura `IsRight` do zadania o hetmanach

Dane: Robot jest w dopuszczalnym k-stanie **i** można umieścić hetmana w pionownicy `New`
Wyniki: Robot jest w następnym dopuszczalnym k-stanie.

$x \leftarrow P_k$
$y \leftarrow k$
$H_x \leftarrow$ **fałsz**
$U_{x-y} \leftarrow$ **fałsz**
$V_{x+y} \leftarrow$ **fałsz**
$x \leftarrow$ `New`
$P_k \leftarrow x$
$H_x \leftarrow$ **prawda**
$U_{x-y} \leftarrow$ **prawda**
$V_{x+y} \leftarrow$ **prawda**

Algorytm 11. Procedura `Right` do zadania o hetmanach