Sprawozdanie 02: Programowanie z nawrotami na przykładzie "problemu hetmanów"

Dmytro Sakharskyi - L02 - 31259 03.04.2025

Metody Programowania

Informatyka 1 rok. 2 semestr

Maritime University of Szczecin

Wstęp

W niniejszym ćwiczeniu skupiamy się na technice algorytmicznej zwanej programowaniem z nawrotami, która służy do rozwiązywania problemów kombinatorycznych. Metoda ta polega na systematycznym przeszukiwaniu przestrzeni możliwych rozwiązań – algorytm próbuje różnych konfiguracji i cofa się, gdy dalsze próby nie mają sensu ze względu na narzucone ograniczenia.

Jednym z klasycznych przykładów problemów rozwiązywanych tą techniką jest problem N hetmanów, polegający na rozmieszczeniu N hetmanów na szachownicy o wymiarach N×N w taki sposób, aby żaden nie atakował innego – czyli, żeby nie znajdowali się w tych samych wierszach, kolumnach ani przekątnych.

Dzięki rozszerzeniu problemu również o wieże, możliwe było porównanie złożoności i czasu działania algorytmu w zależności od typu figury.

Zadanie

Kod 1.

```cpp
bool BezpDama(const vector<vector<int>>& board, int row, int col, int N) {
    for (int i = 0; i < col; i++)
        if (board[row][i]) return false;
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j]) return false;
    for (int i = row, j = col; i < N && j >= 0; i++, j--)
        if (board[i][j]) return false;
    return true;
}
```

Funkcja BezpDama sprawdza, czy dana pozycja na planszy (określona przez row i col) jest bezpieczna dla hetmana. Weryfikuje ona trzy kierunki ataku:

- Lewo w wierszu (czy w danym wierszu po lewej stronie nie ma innej damy),
- Lewa górna przekątna,
- Lewa dolna przekątna.

Jeśli na żadnym z tych pól nie ma innego hetmana – funkcja zwraca true, czyli pozycja jest bezpieczna.

Kod 2.

```cpp
bool BezpWierz(const vector<vector<int>>& board, int row, int col, int N) {
    for (int i = 0; i < col; i++)
        if (board[row][i]) return false;
    for (int i = 0; i < row; i++)
        if (board[i][col]) return false;
    return true;
}
```

BezpWierz sprawdza bezpieczeństwo pozycji dla wieży, czyli:

- Czy w tym samym wierszu po lewej nie ma innej wieży,
- Oraz czy w tej samej kolumnie powyżej nie ma innej wieży.

Wieża nie atakuje po przekątnych, więc sprawdzenie jest prostsze niż w przypadku hetmana.

Kod 3.

```cpp
bool solve(vector<vector<int>>& board, int col, int N, int count, bool isQueen) {
    if (count == 0) return true;
    if (col >= N) return false;

    for (int i = 0; i < N; i++) {
        if ((isQueen && BezpDama(board, i, col, N)) ||
            (!isQueen && BezpWierz(board, i, col, N))) {
            board[i][col] = 1;
            printTablica(board);
            if (solve(board, col + 1, N, count - 1, isQueen))
                return true;
            board[i][col] = 0;
        }
    }
    return solve(board, col + 1, N, count, isQueen);
}
```

Ta funkcja to serce całego algorytmu. Próbujemy rekursywnie ustawić kolejne figury (hetmany lub wieże) na planszy:

- col wskazuje kolumnę, w której aktualnie próbujemy postawić figurę,

- count to liczba figur, które jeszcze trzeba umieścić,
- isQueen decyduje, której funkcji bezpieczeństwa użyć (BezpDama lub BezpWierz).

Jeśli uda się ustawić wszystkie figury (count == 0), zwracane jest true. W przeciwnym razie, algorytm cofa się (czyli następuje backtracking) i próbuje kolejnych możliwych pozycji.

Testy

Zostały przeprowadzone testy, w których liczba figur była równa rozmiarowi szachownicy. Oznaczało to, że dla planszy o wymiarach 5×5 należało ustawić 5 figur. Poniżej przedstawiono wyniki przeprowadzonych eksperymentów.

Czas dla Hetmanów

Test dla 1

```
Czas wykonania: 64 ms
Rozwiazanie znalezione:
X
```

Test dla 2

```
Czas wykonania: 246 ms
Brak rozwiazania
```

Test dla 3

```
Czas wykonania: 1133 ms
Brak rozwiazania
```

Test dla 4

```
Czas wykonania: 926 ms
Rozwiazanie znalezione:
. . X .
X . . .
. . . X
. X . .
```

Test dla 5

```
Czas wykonania: 301 ms
Rozwiazanie znalezione:
X . . . .
. . . X .
. X . . .
. . . . X
. . X . .
```

Test dla 6

```
Czas wykonania: 16136 ms
Rozwiazanie znalezione:
. . . X . .
X . . . . .
. . . . X .
. X . . . .
. . . . . X
. . X . . .
```

Test dla 7

```
Czas wykonania: 704 ms
Rozwiazanie znalezione:
X . . . . . .
. . . . X . .
. X . . . . .
. . . . . X .
. . X . . . .
. . . . . . X
. . . X . . .
```

Test dla 8

```
Czas wykonania: 133283 ms
Rozwiazanie znalezione:
X . . . . . . .
. . . . . . X .
. . . . X . . .
. . . . . . . X
. X . . . . . .
. . . X . . . .
. . . . . X . .
. . X . . . . .
```

Po ustawieniu **9 figur i więcej**, dalsze eksperymenty nie mogły zostać przeprowadzone z powodu ograniczonej mocy obliczeniowej komputera – czas obliczeń przekraczał **5 minut**.

Czas dla Wież

Test dla 1

```
Czas wykonania: 65 ms
Rozwiazanie znalezione:
X
```

Test dla 2

```
Czas wykonania: 120 ms
Rozwiazanie znalezione:
X .
. X
```

Test dla 3

```
Czas wykonania: 189 ms
Rozwiazanie znalezione:
X . .
. X .
. . X
```

Test dla 4

```
Czas wykonania: 240 ms
Rozwiazanie znalezione:
X . . .
. X . .
. . X .
. . . X
```

Test dla 5

```
Czas wykonania: 304 ms
Rozwiazanie znalezione:
X . . . .
. X . . .
. . X . .
. . . X .
. . . . X
```

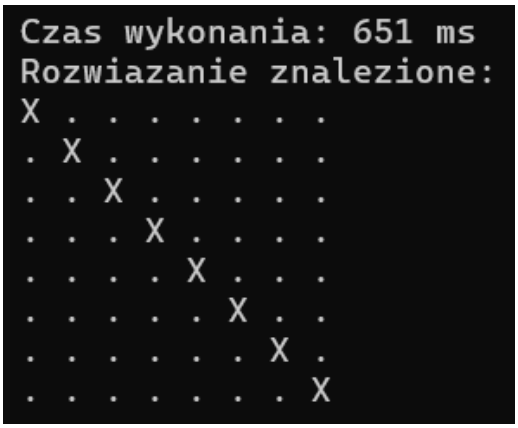
Test dla 6

```
Czas wykonania: 369 ms
Rozwiazanie znalezione:
X . . . . .
. X . . . .
. . X . . .
. . . X . .
. . . . X .
. . . . . X
```

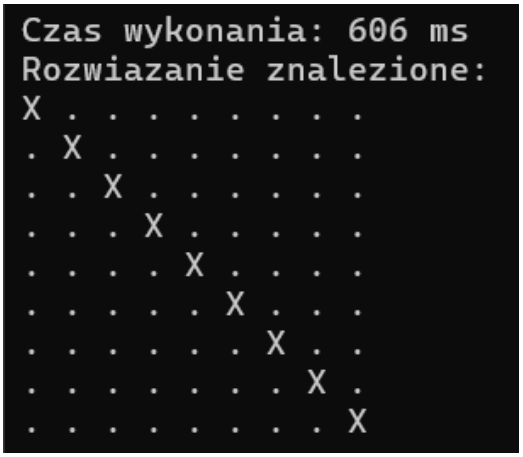
Test dla 7

```
Czas wykonania: 489 ms
Rozwiazanie znalezione:
X . . . . . .
. X . . . . .
. . X . . . .
. . . X . . .
. . . . X . .
. . . . . X .
. . . . . . X
```

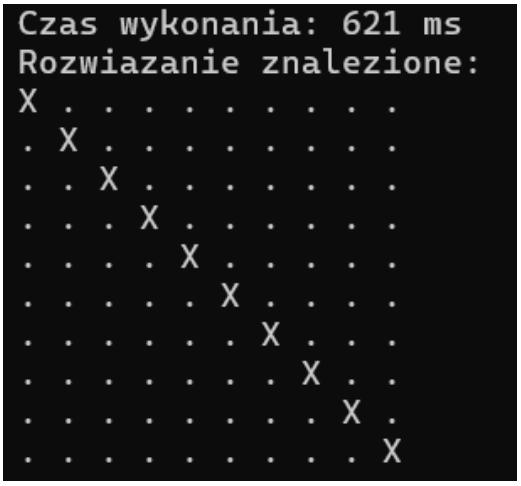
Test dla 8



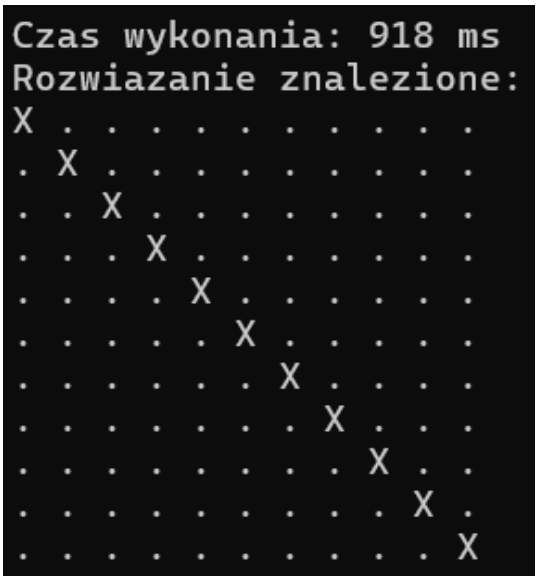
Test dla 9



Test dla 10



Test dla 11



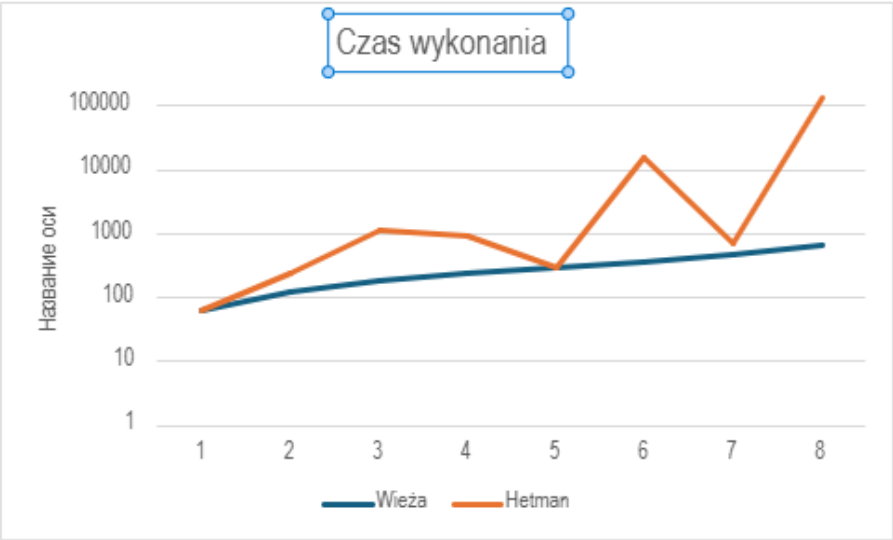
Test dla 12

Możemy zauważyć, że wzrost czasu wykonania jest stabilny i w przybliżeniu równomierny, ponieważ figura jest ustawiana wzdłuż jednej linii w sposób uporządkowany.

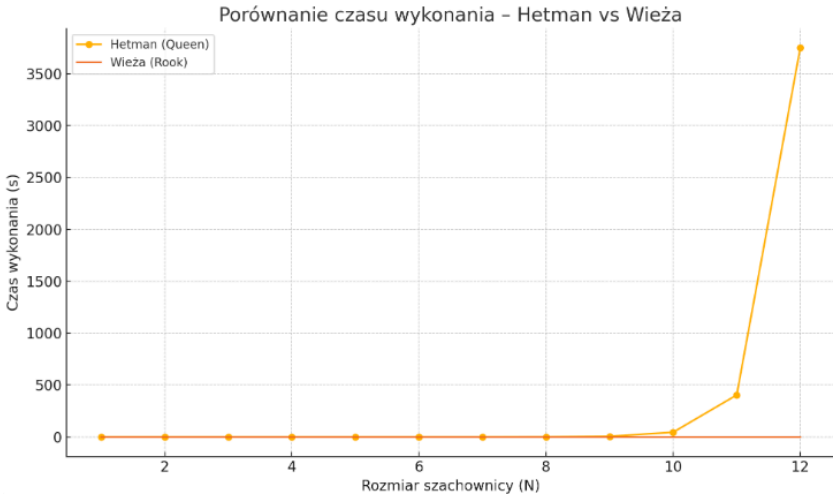
Porównanie czasów wykonania

Ilość figur	1	2	3	4	5	6	7	8
Wieża	65	120	189	240	304	369	489	651
Hetman	64	246	1133	926	301	16136	704	133283

Czas jest porównywany w milisekundach.



(symulacja w excelu)



(symulacja od chataGPT)

Wnioski

Na podstawie przeprowadzonych nowych testów można wyciągnąć następujące wnioski, z symulacji w excela:

- W przypadku hetmanów czas wykonania rośnie **gwałtownie i nieregularnie**, można powiedzieć, że **wzrost jest skokowy i zbliżony do wykładniczego**. Już przy N = 8 i większych dalsze testy stawały się bardzo czasochłonne, a czas obliczeń często przekraczał 5 minut, przez co część eksperymentów nie mogła zostać przeprowadzona.

- Dla wież sytuacja wygląda inaczej – **czas rośnie stopniowo i stabilnie**, przy zachowaniu mniej więcej stałego przyrostu. Dla tego typu figury algorytm wykonuje się znacznie szybciej nawet przy większych rozmiarach planszy.

- Nie udało się znaleźć jednej wyraźnej zależności czasowej między tymi dwiema figurami – mają zupełnie inną charakterystykę działania w ramach algorytmu nawrotowego. Wydajność zależy nie tylko od typu figury, ale również od układu przestrzeni i liczby dostępnych miejsc.

- Wcześniejsza symulacja wykonana przez ChatGPT była niedokładna – opierała się na przybliżonych danych. Dlatego była wykonana symulacja w programie Excel, która lepiej oddaje rzeczywiste wyniki działania programu. Na jej podstawie można zauważyć, że **tylko ręczne testy pozwalają dokładnie przeanalizować zachowanie algorytmu**.

- Dzięki realizacji tego ćwiczenia można nie tylko lepiej zrozumieć ideę programowania z nawrotami, ale również zauważyć, jak typ figury oraz ograniczenia przestrzenne wpływają na złożoność obliczeniową.