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

```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;
}
```

Ten fragment kodu odpowiada za sprawdzenie, czy hetman może zostać postawiony na danym polu bez atakowania innych hetmanów.

```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;
}
```

Ten fragment kodu odpowiada za sprawdzenie, czy wieża może zostać postawiony na danym polu bez atakowania innych wież.

```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);
}
```

Funkcja rekurencyjna odpowiedzialna za próby ustawienia kolejnych figur.

```
Wybierz fig (Q - hetman, R - wieza): R
Podaj rozmiar szachownicy (N): 10
Podaj liczbe figur do umieszczenia: 8
X . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
```

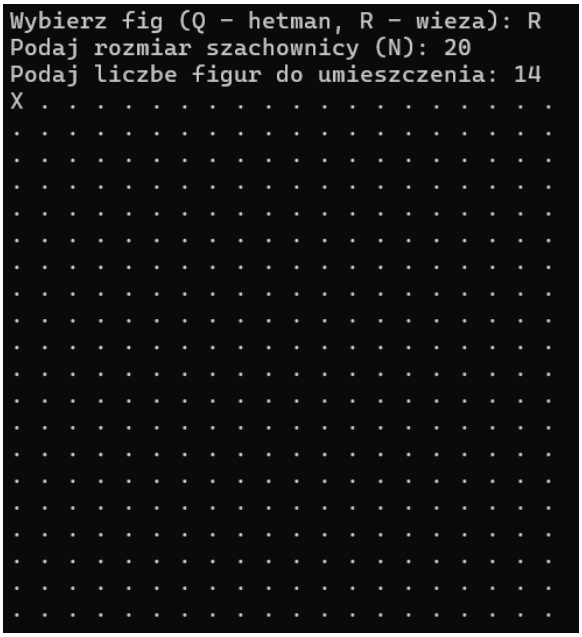
```
Czas wykonania: 559 ms
Rozwiazanie znalezione:
X . . . . . . . . .
. X . . . . . . . .
. . X . . . . . . .
. . . X . . . . . .
. . . . X . . . . .
. . . . . X . . . .
. . . . . . X . . .
. . . . . . . X . .
. . . . . . . . . .
. . . . . . . . . .
```

```
Wybierz fig (Q - hetman, R - wieza): Q
Podaj rozmiar szachownicy (N): 10
Podaj liczbe figur do umieszczenia: 8
X . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```

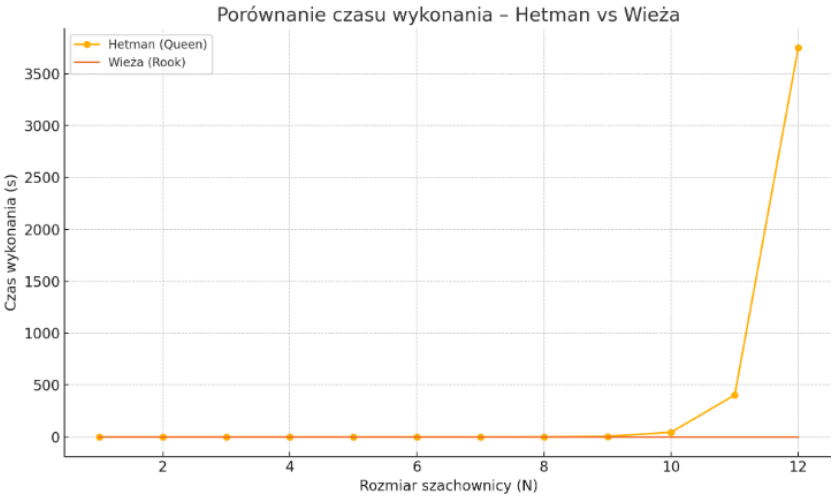
```
Czas wykonania: 1549 ms
Rozwiazanie znalezione:
X . . . . . .
. . . X . . . .
. X . . . . . .
. . . . . X . .
. . X . . . . .
. . . . . .
. . . . . X . .
. . . . X . . .
. . . . . .
. . . . X . . .
```

```
Wybierz fig (Q - hetman, R - wieza): Q
Podaj rozmiar szachownicy (N): 20
Podaj liczbe figur do umieszczenia: 14
X . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```


Czas wykonania: 1471 ms
Rozwiazanie znalezione:


Wybierz fig (Q - hetman, R - wieza): R
Podaj rozmiar szachownicy (N): 20
Podaj liczbe figur do umieszczenia: 14


Czas wykonania: 1431 ms
Rozwiazanie znalezione:



(symulacja od chataGPT - porównanie czasu wykonania)

Wnioski

Na podstawie przeprowadzonych pomiarów oraz obserwacji działania algorytmu można wyciągnąć następujące wnioski:

Wieże układają się średnio około 3 razy szybciej niż hetmany, jednak dotyczy to głównie sytuacji, w których przestrzeń na planszy jest ograniczona. W przypadkach, gdy na planszy jest dużo wolnego miejsca w stosunku do liczby figur, różnica w czasie wykonania między hetmanami a wieżami staje się minimalna.

Czas wykonania zależy silnie od dwóch czynników: ilości dostępnego miejsca (rozmiar planszy) oraz liczby figur do rozmieszczenia. Im mniej miejsca i więcej figur, tym więcej czasu potrzebuje algorytm na znalezienie poprawnej konfiguracji — lub w ogóle nie znajdzie żadnego rozwiązania.

W niektórych przypadkach czas wykonania może gwałtownie rosnąć – szczególnie gdy plansza jest ciasna i możliwości ustawienia figur są ograniczone. Jest to typowe zachowanie dla algorytmów z nawrotami, które w takich sytuacjach muszą przeanalizować ogromną liczbę możliwych konfiguracji.

Dzięki symulacji przeprowadzonej przez *ChatGPT* możliwe było również zobrazowanie zależności między liczbą figur a czasem działania programu. Wygenerowany wykres pokazał, jak wraz ze wzrostem N rośnie czas potrzebny na znalezienie rozwiązania, co dobrze ilustruje rosnącą złożoność problemu.

Dzięki realizacji tego ćwiczenia mogłem nie tylko lepiej zrozumieć ideę programowania z nawrotami, ale również w praktyce przekonać się, jak złożoność obliczeniowa wpływa na działanie algorytmu. Dodatkowo rozszerzenie problemu o inną figurę – wieżę – pozwoliło spojrzeć na problem z szerszej perspektywy i porównać efektywność różnych wariantów. Ćwiczenie to było wartościowym doświadczeniem zarówno pod kątem algorytmicznym, jak i praktycznym – w zakresie optymalizacji, analizy oraz wizualizacji działania programu.