Algorytmy dla Problemów Trudnych Obliczeniowo.
Rozwiązanie zadania: **Potęga hetmanów**
Informatyka 2015/16

Autor: Zbigniew Królikowski

14 lipca 2016

Prowadzący: dr hab. inż. Piotr Faliszewski

1 Uwagi

Moja filozofia zapisu: **hetmany bijące nie zmieniają miejsca tylko znikają na rzecz zbitego** - taki obraz przyjąłem gdyż łatwiej było go rozpisać, mniej dynamiki. Często zapisuję bicie jako redukcję. Nie zmienia to w żaden sposób istoty algorytmu tylko jego opis, który mógłby być mylący bez tej notki.

2 Reprezentacja problemu

Plansza została przedstawiona jako **graf nieskierowany**, w którym każdy węzeł połączony jest z innymi co najwyżej 8 krawędziami.

2.1 Zapis wartości hetmanów

W celach uproszczenia obliczeń wartości są reprezentowane jako wykładniki liczby 2.

2.2 Krótki słownik moich pojęć - w razie wątpliwości

- osie - pion, poziom i skosy
- hetman widoczny dla hetmana *(zbijalny bezpośrednio, ale bez patrzenia na wykładnik)*

3 Wstępne obserwacje - ogranicznie maksymalnej wartości hetmana

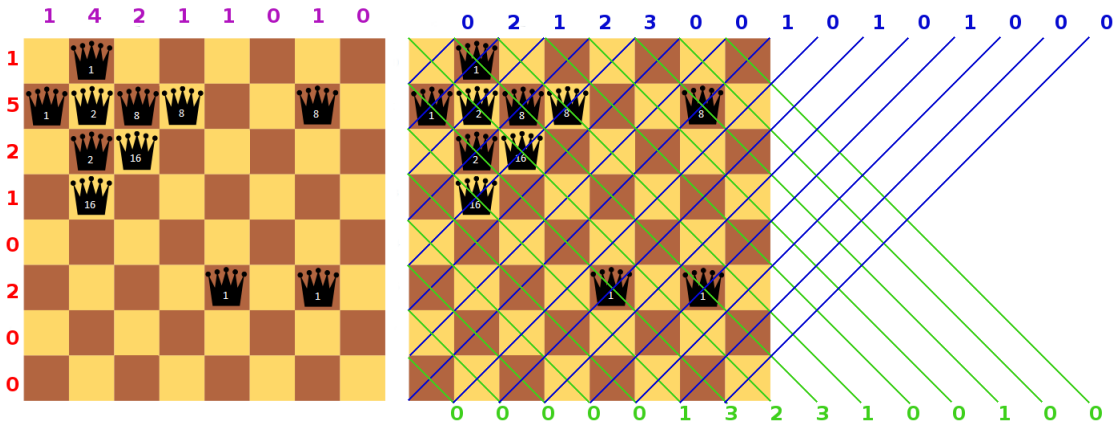
Na wartość końcową hetmana da się nałożyć pewne ograniczenie:

Końcowa wartość hetmana h nie może być wyższa od jego aktualnej wartości $V(h) * 2^n$ gdzie n - liczba hetmanów w pionie, poziomie i na skosach (osiach).

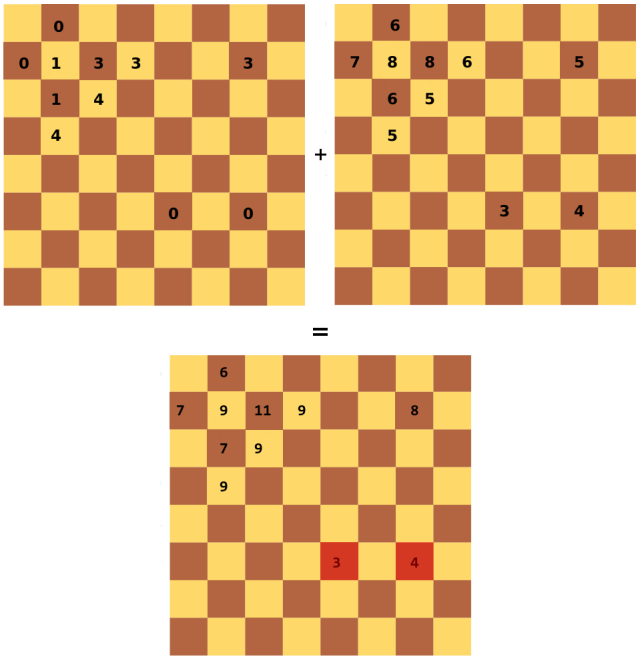
Jest to łatwo udowodnić - każde bicie skierowane przeciwko hetmanowi podnosi jego wartość dwukrotnie. Bić może być *w najlepszym wypadku* tyle ile hetmanów będących na tych samych osiach co dany hetman.

Jak zademonstrowano na powyższym przykładzie możemy **wykluczyć** pewne kombinacje(o wielkości K) hetmanów z bycia końcowymi. Warunkiem wykluczenia jest niemożliwość uzyskania przez tą kombinację wartości będącej sumą wartości wszystkich hetmanów na planszy. Jest to pewne uproszczenie problemu, część postaci rozwiązania.

Bezpośrednio nieużyteczne - liczba tych kombinacji wynosi $\binom{N}{K}$ gdzie N to liczba hetmanów na planszy a K to ilość hetmanów końcowych.



Rysunek 1: Zliczenie ilości hetmanów na poszczególnych osiach.



Rysunek 2: *Lewo:* Wartość hetmanów w mojej notacji *Prawo:* maksymalna ilość bić skierowana ku konkretnemu hetmanowi. *Dół:* Górne ograniczenie na wartość końcową hetmanów. Na czerwono hetmany, które nie mogą być końcowe dla K=1.

3.0.1 Heurystyka

Możemy się zatem spodziewać, że pewne hetmany zostaną zredukowane "wcześniej niż później", a więc mamy nałożoną jakąś **kolejność przeszukiwania** przestrzeni rozwiązań. **Skoro mniejszy potencjał hetmana wiąże się z niemożnością bycia końcowym hetmanem to może powinniśmy ruszać się wcześniej tymi z mniejszym potencjałem?**

4 Algorytm

Algorytm przeszukuje przestrzeń rozwiązań za pomocą wywołań rekurencyjnych modyfikując za każdym razem planszę. Kolejność przeszukiwania jest heurystyką, która zostanie później opisana.

```
//Dane globalne
queens[] // lista hetmanow, kazdy zawiera powiazania do sasiadow
queenCount // obecna liczba hetmanow
targetCount // docelowa liczba hetmanow

bool recursiveCheck():
        if achievedTarget():
                return true
        else:
                sortQueens()
                for queen in board:
                        connections = queen.possibleConnections()
                        sortConnections()
                        for connection in connections:
                                move(queen, connection)
                                if recursiveCheck():
                                        return true
        // Nie znaleziono rozwiazania
                undoMove()
                return false

bool achievedTarget():
        return queenCount <= targetCount

void move(queen, target) // Logika ruchu
```

4.1 Sposób sortowania hetmanów

4.1.1 Najprostsze rozwiązanie

Hetmany porządkujemy w oparciu o wartość:

$$c_n$$

czyli ilość hetmanów "widocznych"*(zbijalnych bezpośrednio, ale bez patrzenia na wykładnik)* z tego pola przed podjęciem jakiegokolwiek ruchu

Algorytm okazał się dawać lepsze wyniki od rozwiązania *brute-force*, zwłaszcza biorąc pod uwagę, że porządek zbijania hetmanów jest statyczny.

4.1.2 Wersja wynikająca z analizy

W początkowym założeniu wybór hetmana, który w danej chwili się rusza miał opierać się na heurystyce **4.0.1** gdzie każdemu hetmanowi zostaje przyporządkowana wartość liczbowa v_n dana formułą:

$$v_n = p_n * C_p + c_n * C_c$$

gdzie:

- p_n - aktualny wykładnik hetmana
- c_n - maksymalna teoretyczna ilość bić na to pole
- C_p - stała, wyznaczana eksperymentalnie, domyślnie 1
- C_c - stała, wyznaczana eksperymentalnie, domyślnie 1

Hetmany miałyby być sortowane rosnąco wedle tej wartości a następnie w tym porządku ruszane.

4.1.3 Udoskonalona wersja

Okazało się, że **znacząco** lepsze wyniki daje dodatkowe uproszczenie : zamiast maksymalnej teoretycznej ilości bić na pole - początkowa liczba "widocznych"*(zbijalnych bezpośrednio, ale bez patrzenia na wykładnik)*.

W wyniku tego otrzymujemy formułę:

$$v_n = p_n * C_p + c_n * C_c$$

gdzie:

- p_n - aktualny wykładnik hetmana
- c_n - ilość hetmanów widocznych z tego pola
- C_p - stała, wyznaczana eksperymentalnie, domyślnie 1
- C_c - stała, wyznaczana eksperymentalnie, **domyślnie** 2

Bardzo ważny stała się waga C_c, która wyznaczona została przypadkowo, lecz daje bardzo dobre wyniki. Np. wartości 1.9 i 2.1 dają sporo wolniejszy algorytm. Nie udało mi się znaleźć przyczyny takiego stanu rzeczy.

4.2 Sposób sortowania ruchów dla danego hetmana

Zadaniem tego algorytmu jest porządkowanie połączeń wychodzących z danego hetmana. W końcowej wersji nie nałożyłem tutaj żadnego heurystycznego porządku - wszystkie testowane heurystyki okazały się nieużyteczne.

4.3 Podsumowanie

Najlepszy okazał się algorytm, który bazował na pierwotnej analizie - a raczej uproszonych wnioskach z niej, który został poddany dodatkowemu uproszczeniu poprzez "irracjonalneźmiany. Pokazuje to, że początkowa analiza była stanowczo za słaba jednak czasami eksperymenty z algorytmem przynoszą efekty.

Problem można poddać dużo dogłębniejszej analizie - próbowałem to robić nieformalnie ale mało które "teorie"dawały efekty w praktyce. Implementacja zdecydowanie zajmowała sporo czasu, choćby ze względu na wymóg arbitra zamieszczenia kodu w jednym pliku co utrudniało modyfikacje.

5 Dodatek: Generator scenariuszy

Dodatkowo wykonałem w języku Python generator który dla danej wielkości planszy, końcowej ilość hetmanów i docelowej ilości ruchów tworzy losowy scenariusz wraz z przykładowym rozwiązaniem. Okazał się on bardzo przydatny w testowaniu algorytmu.