Algorytmy przeszukiwania niepoinformowane do rozwiązania problemu n-hetmanów

Joanna Kołodziejczyk

1 Cel laboratoriów

Celem laboratorium jest implementacja algorytmu wszerz (breadth-first search) i algorytmu w głąb (depth first search) w celu znalezienia rozwiązania dla zadania n-hetmanów.

2 Definicja problemu

Problem z n-hetmanów można sformułować następująco: „Biorąc pod uwagę szachownicę $n \times n$, znajdź sposób na umieszczenie n hetmanów na planszy tak, aby wzajemnie się nie atakowały". Nie istnieją rozwiązania dla $n = 2$ i $n = 3$.

Możliwe jest sformułowanie problemu n-hetmanów jako problem przeszukiwania (search). Cztery składowe problemu przeszukiwania:

1. Stan: szachownica z liczbą hetmanów z przedziału od 0 do n.
2. Stan początkowy: Pusta szachownica.
3. Generowanie stanów potomnych (manipulacja): z każdego stanu można otrzymać bezpośrednio stany potomne. W przypadku n-hetmanów przyjmujemy, że nowy potomek ma o 1 hetmana więcej niż jego rodzic. Hetman dostawiony do rodzica, nie może zajmować pozycji na szachownicy już zajętej.
4. Zakończenie (warunek stopu): Proces szukania rozwiązania zostanie zakończony, gdy na planszy znajduje się n hetmanów, którzy się nie atakują.

3 Struktura danych do reprezentacji problemu

W zadaniu trzeba użyć reprezentację wektora współrzędnych (wiersz, kolumna), na ostatnich zajęciach tzw. krotka, np. `board = [(0, 2), (1, 0), (2, 3), (3, 1)]`.

4 Algorytmy BFS i DFS

Algorytmy są podane w skrypcie na stronach 24 i 25. Kilka wskazówek do implementacji:

1. s_0 stan początkowy to pusta szachownica np. w języku Python [] - czyli szachownica bez hetmanów.

2. Warunek czy s jest stanem końcowym ma realizować sprawdzenie liczby hetmanów na szachownicy i porównanie z n oraz czy ich położenie nie generuje bić.

3. Wygeneruj zbiór stanów potomnych t dla s polega na dodaniu do listy reprezentującej stan s kolejnej krotki (pary współrzędnych) np. w przypadku $n = 4$ dla stanu $s = [(0, 0)]$ potomkowie to

$$t = \begin{Bmatrix} [(0,0),(0,1)], [(0,0),(0,2)], [(0,0),(0,3)], [(0,0),(1,0)] \ldots, \\ [(0,0),(3,0)], [(0,0),(3,1)], [(0,0),(3,2)], [(0,0),(3,3)] \end{Bmatrix}$$

4. Listę *Open* należy zaimplementować jako kolejkę dla BFS i stos dla DFS. W ten sposób określi się kolejność pobierania elementów do sprawdzenia i generowania potomków.

5. Ze względu na oczekiwany efekt (znalezienie układu będącego rozwiązaniem) wskaźniki na rodzica nie są konieczne, bo jako rozwiązanie nie trzeba podawać jaka była ścieżka (kolejność wstawiania hetmanów), która doprowadziła do rozwiązania. Można jednak wskaźniki zaimplementować jako element uogólniający algorytm.

Strategię BFS wizualizuje rysunek 1 [1].

5 Wymagania dla implementacji

Język implementacji dla zadania jest dowolny.

Celem implementacji jest porównanie względnej efektywności strategii BFS i DFS w rozwiązywaniu problemu n-hetmanów.

Zadania:

1. n jest zadawanym parametrem algorytmu.

2. Wykorzystać reprezentację szachownicy jak opisano wyżej.

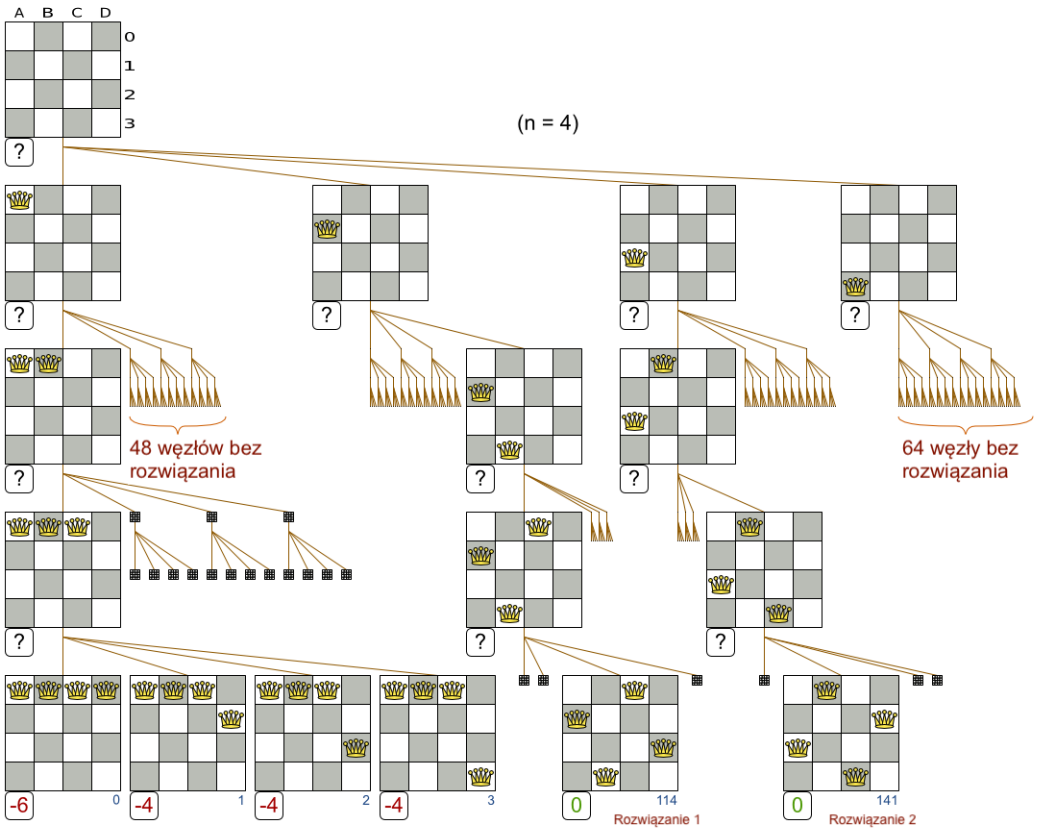
3. Zaimplementować funkcję generującą potomka. Funkcja przyjmuje na wejściu stan z liczbą hetmanów x a na wyjściu zwraca listę/zbiór wszystkich stanów z liczbą hetmanów $x + 1$.

4. Zaimplementować algorytmy BFS i DFS zgodnie z kodem ze skryptu i wskazówkami z punktu 4. Najlepiej by algorytm był parametrem i na tej podstawie zmieniał sposób obsługi listy Open.

5. Program po wykonaniu powinien wyświetlać:

   - Stan, który został wskazany przez strategię jako rozwiązanie: (uwaga: może być w formie „surowej"). Ewentualnie, jeżeli ktoś zaimplementuje wersję ze wskaźnikami, to podać oprócz stanu końcowego wszystkich jego poprzedników.

[1] `https://docs.drools.org/6.0.0.Final/optaplanner-docs/html/exactMethods.html`



Rysunek 1: Wizualizacja podejścia BFS dla $n = 4$

- Statystyki:
  - liczbę stanów w liście *Open*,
  - liczbę stanów sprawdzonych —lista *Closed*,
  - względny czas wykonania.
- Wykonać eksperymenty obliczeniowe dla n zmieniającego się od 4 do max, gdzie max jest znalezioną wartością, dla której czas wykonania jest znośny dla użytkownika.
- Zademonstrować wyniki eksperymentu w postaci trzech wykresów pokazujących zmienność statystyk w zależności od liczby hetmanów. Na osi odciętych powinny być przyrastające wartości n a na osi rzędnych wartości statystyk dla BFS i DFS.
- Skomentować krótko wyniki. Jakie są słabe punkty wykonanej implementacji?

6 Kryteria oceny

1. Zgodność z wymaganiami w zakresie algorytmu (generowanie potomka, obsługa FIFO, obsługa LIFO, rest osiągnięcia , wskaźniki na stan rodzicielski) (max 2pkt)
2. Wyniki poprawne i wyświetlanie wyników zgodnie z wymaganiami (max 0,25pkt))
3. Eksperymenty (max 0,5 pkt))
4. Wnioski z eksperymentów (max 0.25pkt)
5. Wypełniona samoocena i odpytanie z kodu będzie mnożnikiem (procentem w jakim autor rozumie swoje rozwiązanie).

7 Przekazanie programu

- Kod z rozwiązaniem proszę podpiąć w Teams. Bardzo proszę nazwać plik źródłowy nazwisko.imię.???.
- W przypadku wykonania paczki, proszę o format zip.
- Wyniki i wnioski do eksperymentów mogą być podane jako komentarz w kodzie.