Algorytmy przeszukiwania w głąb (DFS) do rozwiązania problemu n-hetmanów

Joanna Kołodziejczyk

1 Problem do rozwiązania

Celem laboratorium jest implementacja algorytmów: w głąb (depthfirst search) w celu znalezienia rozwiązania dla układanki n-hetmanów.

2 Algorytm DFS

Algorytm przeszukiwania w głąb (Depth First Search - DFS) przeszukuje po kolei wszystkie węzły w drzewie(grafie) przeszukując gałęziami do najgłębszego poziomu. Innymi słowy można go zaimplementować jako stos (kolejka LIFO).

Procedurę rozwiązywania n hetmanów z pomocą DFS można opisać następującymi krokami:

1. Utwórz pustą kolejkę tablic/wektorów na listę stanów do przeszukania
2. Dodaj do listy stan początkowy
3. W pętli wykonuj dopóki lista stanów do przeszukania nie jest pusta:
   (a) Pobierz z **końca** listy stan (ostatnio dodany stan)
   (b) Sprawdź warunek osiągnięcia celu. Jeżeli został osiągnięty możesz przerwać procedurę i wyświetlić pierwsze znalezione rozwiązanie (opcjonalnie można kontynuować do znalezienia wszystkich rozwiązań). W przeciwnym wypadku kontynuuj.
   (c) Wygeneruj wszystkich potomków aktualnego stanu i dodaj ich na końcu listy stanów do przeszukiwania.

2.1 Algorytm DFS w rekurencji

Algorytm DFS można zaimplementować jako procedurę rekurencyjną. `http://en.wikipedia.org/wiki/Depth-first_search`

3 Wymagania do implementacji

Język implementacji jest dowolny.

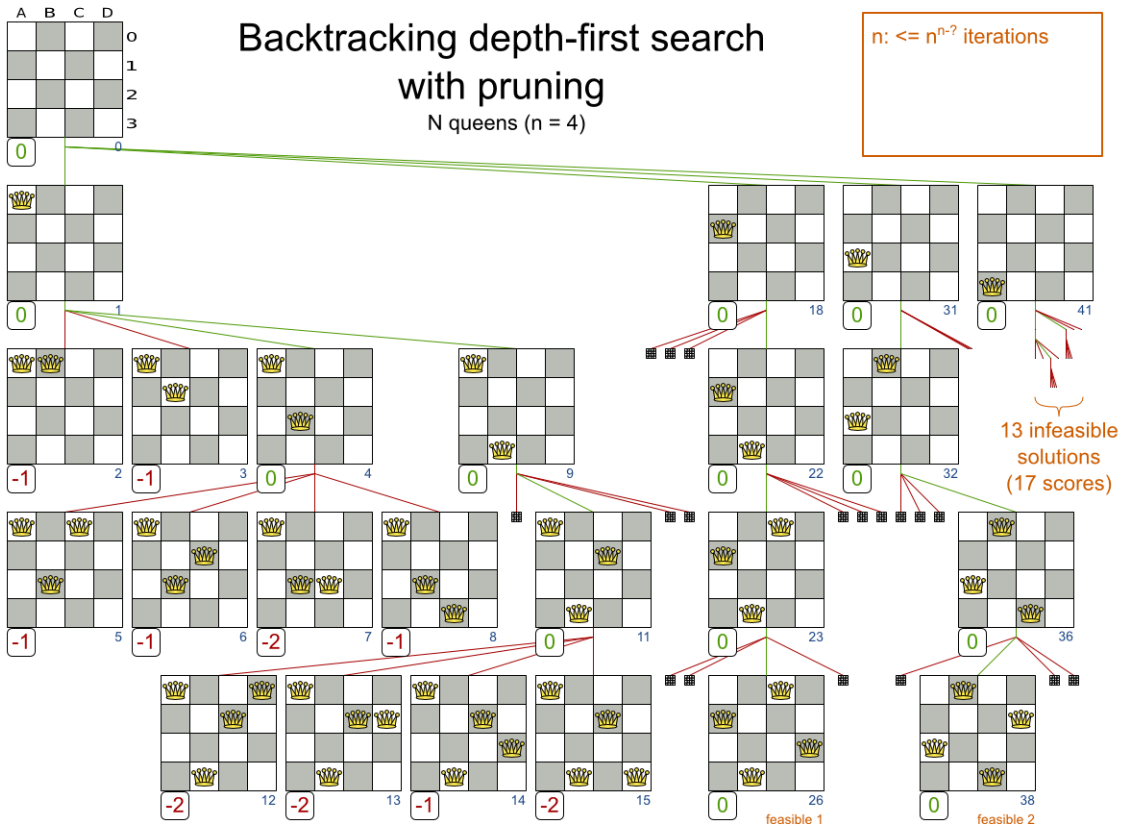
Celem implementacji jest porównanie względnej efektywności strategii z poprzedniego zadania czyli BFS i obecnej DFS do rozwiązania problemu n-hetmanów. Aby osiągnąć ten cel, strategia DFS powinna zostać zaimplementowane przy użyciu tego samego kodu bazowego. Wyszukiwanie w DFS powinno być realizowane przy użyciu stosu zewnętrznego (alternatywą dla stosu jest podejście rekurencyjne - które też używa stos w sposób ukryty).

Zadania:

1. n jest zadawanym parametrem algorytmu. Reprezentacja szachownicy - wektor, jak w BFS.
2. Zaimplementować funkcję generujące potomka (jak w BFS). Funkcja przyjmuje na wejściu stan z liczbą hetmanów x a na wyjściu zwraca listę wszystkich stanów z liczbą hetmanów $x + 1$.
3. Zaimplementować algorytmy DFS zgodnie z instrukcją z punktu 2.
4. Program po wykonaniu powinien wyświetlać:
   - stan, który został wskazany przez program jako rozwiązanie, czyli współrzędne hetmanów na planszy (możliwe jest użycie wizualizacji),
   - statystyki:
     - liczbę stanów wygenerowanych (funkcją generującą potomka),
     - liczbę stanów sprawdzonych ((dla których wykonany został test osiągnięcia celu),
     - względny czas wykonania.
5. Wykonać eksperymenty obliczeniowe dla n zmieniającego się od 4 do osiągalnej liczby np 12.
6. Zademonstrować wyniki eksperymentu w postaci wykresu (ewentualnie listy) zmienność statystyk w zależności od liczby hetmanów. Na osi odciętych powinny być przyrastające wartości n a na osi rzędnych wartości statystyk.
7. Porównać algorytmy DFS i BFS, wykorzystując podane statystyki. (Wszystko powyżej do oceny 4.0)
8. Implementacja podejścia inteligentnego w metodzie BFS i DFS oraz porównanie z podejściem podstawowym. (Dodatkowe zadanie na ocenę 5.0)

4 Inteligentna funkcja generująca potomka

Oba algorytmy BFS i DFS można przyspieszyć zmieniając funkcję generującą potomka. W funkcji tej dostawia się hetmana tylko w dozwolonym (nie atakowanym) miejscu. Innymi słowy dodana zostanie wiedza z dziedziny problemu do strategii.



Rysunek 1: Wizualizacja podejścia z wykorzystaniem kodowania wektorem i poprawionej funkcji generowania potomka

1

1. Funkcja generująca potomka: Zwróć tylko poprawne tablice/wektory (bez ataków: różne wiersze, różne kolumny i przekątne) dodając 1 hetmana w polu nie atakowanym przez dotychczas wstawionych hetmanów.

2. Test osiągnięcia celu: Zwróć TRUE iff 'n' hetmanów.

4.1 Przekazanie programu

- Kod z rozwiązaniem proszę podpiąć w Teams. Bardzo proszę nazwać plik źródłowy nazwisko.imię.???.