Algorytmy z nawrotami

Łukasz Konieczny | L4 | Lab 9

1. Omówienie teoretyczne

Algorytmy z nawrotami służą do generowania wszystkich rozwiązań danego problemu, poprzez próbowanie wszystkich możliwości, oraz rezygnację i cofanie się gdy stwierdzi, że dana możliwość na pewno nie prowadzi do rozwiązania.

Metoda ta potrafi być znacząco szybsza od wyczerpującego wyszukiwania rozwiązań, ponieważ odcinając jedno z nich potencjalnie odcina także wiele innych.

Przykładami algorytmów z nawrotami są algorytmy rozwiązujące problem komiwojażera, problem skoczka szachowego czy problem n-hetmanów.

Ich zaletą jest znajdowanie dokładnego rozwiązania, większa efektywność, a także uwzględnianie wszelkich ograniczeń problemu, które zmniejszają zakres wyszukiwania.

Nie są one jednak pozbawione wad, nie można zastosować ich we wszystkich problemach, bazują na rekurencji a w niektórych przypadkach nie zapewniają uzasadniającej ich wykorzystanie przewagi w efektywności względem alg. wyczerpujących.

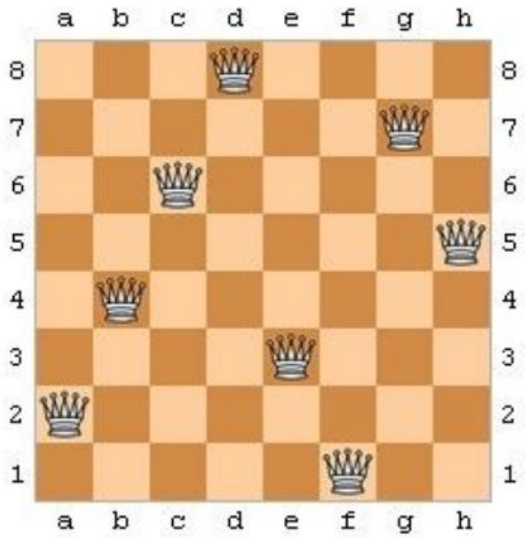
2. Problem n-hetmanów

Problem n-hetmanów polega na znalezieniu takiego ustawienia n hetmanów na szachownicy o wymiarach n x n, aby żaden z nich nie szachował innego.

Oznacza to, że żadna z figur nie może stać w wierszu, kolumnie, ani na przekątnej innej.

Problem ten możemy rozwiązać przy pomocy algorytmów z nawracaniem, w następujący sposób:

- Ustawiamy pierwszego hetmana w pierwszej kolumnie pierwszego wiersza
- Szukamy w wierszu drugim miejsca, które nie jest szachowane przez poprzedniego hetmana i wstawiamy tam hetmana nr. 2
- Postępujemy podobnie z kolejnymi figurami, szukając w odpowiednim wierszu miejsca nie szachowanego przez pozostałe figury. Jeśli takie nie istnieje, cofamy się o jeden wiersz w górę i przesuwamy znajdującą się tam figurę na następne wolne miejsce. Jeśli nie da się tego zrobić, cofamy się wyżej, itd.
- W momencie w którym ustawimy ostatniego hetmana, problem jest rozwiązany. Możemy zakończyć algorytm, lub rozpocząć go od nowa dla pierwszego hetmana przesuniętego o jedno pole w prawo, aż do momentu w którym nie będziemy w stanie dalej go przesunąć. Otrzymamy wtedy wszystkie możliwe rozwiązania problemu



Przykład rozwiązania problemu dla 8 hetmanów

Moja implementacja w C++ opiera się na macierzy reprezentującej szachownicę, w której program zaznacza wszystkie pola szachowane przez hetmanów. Przed umieszczeniem danego hetmana sprawdza, czy znajduje się on na szachowanym polu, a jeśli nie, wstawia go na nie i zaznacza wszystkie szachowane przez niego pola.

Przy pomocy pętli „while" umieszcza na planszy wszystkie figury po kolei, lub odrzuca rozwiązanie jeśli przy nawrotach dotrze do granicy szachownicy.

Aby wyznaczyć więcej niż jedno rozwiązanie, główna pętla jest obudowana kolejną, która po wyznaczeniu rozwiązania resetuje planszę, a następnie ustawia pierwszego hetmana o jedno pole dalej niż wcześniej, zmuszając program do wyznaczenia nowego ustawienia figur. Algorytm kończy pracę, gdy pierwszy hetman przemierzy cały wiersz.

```
while (row < n) {
  result = place(board, n, row, startCol);
  if (result != -1) {
    // Save the queen's position
    pos[row] = result;
    // Move to the next row
    row++;
    // Start from the beginning of the board
    startCol = 0;
  } else {
    // Go back a row
    row--;

    // Start placing the queen in the previous row one column further
    startCol = pos[row] + 1;

    if (startCol == n || row < 0) {
      possible = false;
      break;
    }

    // Reset the board
    for (int x = 0; x < n; x++) {
      for (int y = 0; y < n; y++) {
        board[x][y] = 0;
      }
    }

    // Place back previous queens
    for (int x = 0; x < row; x++) {
      place(board, n, x, pos[x]);
    }
  }
}
```

Główna pętla programu, odpowiadająca za wyznaczenie pojedynczego rozwiązania

```
int place(bool** board, int n, int row, int startCol) {
  int placedPos = -1;
  for (int i = startCol; i < n; i++) {
    if (board[row][i] == 1) continue;

    placedPos = i;

    // Mark horizontally
    for (int j = 0; j < n; j++) {
      board[row][j] = 1;
    }

    // Mark vertically
    for (int j = 0; j < n; j++) {
      board[j][i] = 1;
    }

    // Mark diagonally
    for (int j = 1; j < n; j++) {
      if (row + j < n) {
        if (i - j >= 0) board[row + j][i - j] = 1;
        if (i + j < n) board[row + j][i + j] = 1;
      }
      if (row - j >= 0) {
        if (i - j >= 0) board[row - j][i - j] = 1;
        if (i + j < n) board[row - j][i + j] = 1;
      }
    }

    break;
  }

  return placedPos;
}
```

Funkcja umieszczająca hetmana na planszy i zaznaczająca szachowane przez niego pola

```cpp
int main() {
  cout << "Program do rozwiazywania problemu n-hetmanow" << endl;
  cout << "Lukasz Konieczny | LK4 | LAB 9" << endl << endl;
  cout << "Podaj liczbe n: ";

  int n;
  cin >> n;

  if(n < 1){
    cout << "Niepoprane n!";
    exit(-1);
  }

  cout << endl;
  queens(n);
  return 0;
}
```

Kod sterujący

W sprawozdaniu pominięty został kod odpowiadający za resetowanie planszy i umieszczanie pierwszego hetmana o jedną kolumnę dalej.

3. Wnioski

Podczas zajęć zapoznaliśmy się z ideą algorytmów z nawrotami, ich zaletami oraz wadami. Nauczyliśmy się także rozwiązywać problem n-hetmanów, oraz zaimplementowaliśmy w języku C++ algorytm robiący to za nas.

4. Źródła

https://pl.wikipedia.org/wiki/Algorytm_z_nawrotami

https://pl.wikipedia.org/wiki/Problem_o%C5%9Bmiu_hetman%C3%B3w

Materiały z wykładów – Zbigniew Kokosiński