Дисциплина: СОЗ (3ти курс ИС, зимен семестър 2020/2021)

ЗАДАНИЕ ЗА ДОМАШНА РАБОТА №1
Евристично търсене в пространство на състояния

Зорница Николаева Димитрова,
фак. № 71843

Описание на задачата и описание на използваните методи за нейното решаване в свободен формат:

1. Board.java

В класът Board дефинирам дъската 3x3 като едномерен масив(защото е по-оптимално) и позицията на празното поле представена с 0. Класът съдържа конструктори(копиращ конструктор, конструктор за общо ползване и default-ен конструктор), както и set и get методи за масива, като при полуване на null стойност сетъра сам генерира начално състояние, разбъркващи числата.

Функциите:

-boolean isGoal() – проверява дали текущото състояние е целевото

-Board moveLeft() – мести празната плочка наляво, като проверяваме дали това е възможно(indexOfZero != 0 && indexOfZero != 3 && indexOfZero != 6). На мястото на нулата слагаме числото, което се е намирало отляво, а на негово място записваме 0. След това обновяваме стойността на indexOfZero и връщаме новополученото състояние на дъската.

- Board moveRight(), Board moveUp(), Board moveDown() - работят аналогично на горната функция.

- String printBoard() - служи за изобразяването на едномерния масив като дъска.

- String toString() – връща информация за дъската(изобразява я с помощта на printBoard() и принтира индекса, на който се намира нулата)

2. State.java

Класът State съдържа дъска, предишния state(parent-а на текущия), евристичната функция (h(n)), цената на пътя(g(n)) и операторът (up, down, left, right), с който сме стигнали от parent-а до текущото състояние.

Класът съдържа конструктори(копиращ конструктор, конструктор за общо ползване и default-ен конструктор), както и set и get методи за член данните на класа.

- Функцията boolean isGoal() проверява дали текущото състояние на дъската е целевото.

- Имплементира интерфейса Comparable, като функцията int compareTo(State other) сравнява текущия state с дадения(other) спрямо сборовете от цена на пътя и евристична функция. Ако сборът е по-малък при текущия state връщаме -1, иначе 1.

- toString методът на класа връща информация за текущия state;

3. Heuristics.java

В класът Heuristics са двете евристични функции.

int manhattanDistance(Board board) и int numberOfUnorderedBlocks(Board board) като и при двете смятаме само за тези плочки, които не са си на местата и които са различни от 0.

4. SearchStrategies.java

Класът SearchStrategies имплементира метода за търсене с минимизиране на общата цена на пътя (A*) и метода на най-доброто спускане (best-first search).

- State aStar(State initial, Function<Board, Integer> heuristic)
    - като втори параметър приемаме унарна функция с един аргумент Board, която връща Integer. По този начин методът може да се тества, с която да е от двете евристичните функции.
    - Фактът, че обхождаме дърво и switch(lastMove) statement-а избягват цикличните пътища => няма нужда да запазваме посетените възли в структура visited.
    - PriorityQueue<State> frontier ни държи текущите възли. Първо добавяме initial state-а(началния).

    След това докато приоритетната ни опашка frontier не е празна:
    - с метода poll() приложен върху frontier взимаме нашия currentState, който е най-малкият елемент във frontier спрямо нашия компаратор (функцията int compareTo(State other) в класът State, сравняваща текущия state с дадения спрямо сборовете от цена на пътя и евристична функция. Ако сборът е по-малък при текущия state връщаме -1, иначе 1.)
    - проверяваме дали currentState е целта, ако да – приключваме, ако не:
        - взимаме currentBoard от currentState и взимаме последния си извършен ход, за да не минаваме през едни и същи места.
        - чрез switch(lastMove) избягваме цикличните пътища => ако последният ни ход е бил да се придвижим наляво, няма смисъл да ходим надясно (генерираме 3 наследника-чрез местене наляво, нагоре и надолу); ако е бил нагоре, няма смисъл да ходим надолу (генерираме 3 наследника-чрез местене наляво, надясно и нагоре) и т.н. В default ще влезе само когато имаме “start”(веднъж като започваме)-тогава генерираме и 4-те наследника.
        - след това проверяваме кои наследници са генерирани(различни от null) и ги добавяме във frontier като подаваме новогенерирания state, parent-а му – currentState, евристиката му, цената на пътя(цената на пътя на parent-а му - currentState + 1) и съответното преместване, с което сме стигнали до това състояние(оператор-left, right, up или down).
    - Опашката никога не трябва да остане празна!

- State bestFirstSearch(State initial, Function<Board, Integer> heuristic)

- за разлика от A* този метод не е оптимален. Тук гледаме само евристиката(без цената на пътя).

- Единствената разлика в кода на двете функции е, че при метода на най-доброто спускане при добавянето на наследниците променяме pathCost да е винаги 0, защото ни интересува само евристиката.

Резултати от изпълнението на създаденото приложение:

5. BoardTest.java

Класът BoardTest тества програмата. Съдържа функция void printState(State state), която рекурсивно извежда използваните оператори за достигане до целта. Чрез родителите се връщаме до start state-а.

В main функцията тествам всичко имплементирано.

Описание на реализацията с псевдокод:

1. Метода на най-доброто спускане (best-first search)

- Фронтът на търсенето (сортиран по отношение на стойностите на евристичната функция(h(n))) е списък от пътища (отначало е списък от един път, който включва само началното състояние).

- Ако фронтът има вида [p1, p2, … , pn], то:
  - Избира се p1. Ако p1 е довел до целта, край.
  - Ацикличните пътища p11, p12, … , p1k, които разширяват (продължават) p1, се добавят към фронта и новополученият фронт се сортира в нарастващ ред на оценките от h(n), в резултат на което придобива вида [q1, q2, … , qn+k-1].
  - На следващата стъпка се обработва пътят от фронта, който има най-добра евристична оценка, т.е. q1.

2. Метода за търсене с минимизиране на общата цена на пътя (A*)

- Фронтът на търсенето (сортиран по отношение на стойностите на f(n) = g(n) + h(n)) е списък от пътища (отначало е списък от един път, който включва само началното състояние).

- Ако фронтът има вида [p1, p2, … , pn], то:
  - Избира се p1. Ако p1 е довел до целта, край.
  - Ацикличните пътища p11, p12, … , p1k, които разширяват (продължават) p1, се добавят към фронта и новополученият фронт се сортира в нарастващ ред на оценките от f(n), в резултат на което придобива вида [q1, q2, … , qn+k-1].
  - На следващата стъпка се обработва пътят от фронта, който има най-добра оценка за f(n), т.е. q1.