Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

**Лабораторна робота №6**

з дисципліни «Основи штучного інтелекту»

на тему:

*«***Неінформативний пошук***»*

Варіант -13

Виконав:

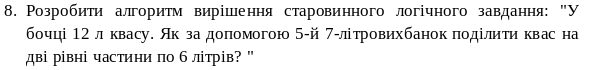
студент групи ІС-72

Кривохижа Роман Андрійович

Київ - 2020

**Мета роботи**: дослідити методи неінформативного пошуку та способи їх застосування на прикладі мови CLIPS.

**Умова**:



**Варіант:**



**GitHub:** <https://github.com/kryvokhyzha/Courses/tree/master/AI>

**Лістинг програми**:

* task.pl

(deffacts ai-lab-6

(state 12 12 7 0 5 0)

)

(deffunction move-rule

(?max-1 ?current-1 ?max-2 ?current-2)

(if (<= ?current-1 (- ?max-2 ?current-2)) then

(create$ 0 (+ ?current-2 ?current-1))

else

(create$ (- ?current-1 (- ?max-2 ?current-2)) ?max-2))

)

(defrule move-1-to-2

(state ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3)

=>

(bind ?move-info (move-rule ?max-1 ?current-1 ?max-2 ?current-2))

(if (not (any-factp ((?f state)) (eq ?f:implied (create$ ?max-1 (nth$ 1 ?move-info) ?max-2 (nth$ 2 ?move-info) ?max-3 ?current-3))))

then

(assert (state ?max-1 (nth$ 1 ?move-info) ?max-2 (nth$ 2 ?move-info) ?max-3 ?current-3))

(assert (move-link ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3 >> ?max-1 (nth$ 1 ?move-info) ?max-2 (nth$ 2 ?move-info) ?max-3 ?current-3))

)

)

(defrule move-1-to-3

(state ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3)

=>

(bind ?move-info (move-rule ?max-1 ?current-1 ?max-3 ?current-3))

(if (not (any-factp ((?f state)) (eq ?f:implied (create$ ?max-1 (nth$ 1 ?move-info) ?max-2 ?current-2 ?max-3 (nth$ 2 ?move-info)))))

then

(assert (state ?max-1 (nth$ 1 ?move-info) ?max-2 ?current-2 ?max-3 (nth$ 2 ?move-info)))

(assert (move-link ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3 >> ?max-1 (nth$ 1 ?move-info) ?max-2 ?current-2 ?max-3 (nth$ 2 ?move-info)))

)

)

(defrule move-2-to-1

(state ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3)

=>

(bind ?move-info (move-rule ?max-2 ?current-2 ?max-1 ?current-1))

(if (not (any-factp ((?f state)) (eq ?f:implied (create$ ?max-1 (nth$ 2 ?move-info) ?max-2 (nth$ 1 ?move-info) ?max-3 ?current-3))))

then

(assert (state ?max-1 (nth$ 2 ?move-info) ?max-2 (nth$ 1 ?move-info) ?max-3 ?current-3))

(assert (move-link ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3 >> ?max-1 (nth$ 2 ?move-info) ?max-2 (nth$ 1 ?move-info) ?max-3 ?current-3))

)

)

(defrule move-2-to-3

(state ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3)

=>

(bind ?move-info (move-rule ?max-2 ?current-2 ?max-3 ?current-3))

(if (not (any-factp ((?f state)) (eq ?f:implied (create$ ?max-1 ?current-1 ?max-2 (nth$ 1 ?move-info) ?max-3 (nth$ 2 ?move-info)))))

then

(assert (state ?max-1 ?current-1 ?max-2 (nth$ 1 ?move-info) ?max-3 (nth$ 2 ?move-info)))

(assert (move-link ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3 >> ?max-1 ?current-1 ?max-2 (nth$ 1 ?move-info) ?max-3 (nth$ 2 ?move-info)))

)

)

(defrule move-3-to-1

(state ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3)

=>

(bind ?move-info (move-rule ?max-3 ?current-3 ?max-1 ?current-1))

(if (not (any-factp ((?f state)) (eq ?f:implied (create$ ?max-1 (nth$ 2 ?move-info) ?max-2 ?current-2 ?max-3 (nth$ 1 ?move-info)))))

then

(assert (state ?max-1 (nth$ 2 ?move-info) ?max-2 ?current-2 ?max-3 (nth$ 1 ?move-info)))

(assert (move-link ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3 >> ?max-1 (nth$ 2 ?move-info) ?max-2 ?current-2 ?max-3 (nth$ 1 ?move-info)))

)

)

(defrule move-3-to-2

(state ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3)

=>

(bind ?move-info (move-rule ?max-3 ?current-3 ?max-2 ?current-2))

(if (not (any-factp ((?f state)) (eq ?f:implied (create$ ?max-1 ?current-1 ?max-2 (nth$ 2 ?move-info) ?max-3 (nth$ 1 ?move-info)))))

then

(assert (state ?max-1 ?current-1 ?max-2 (nth$ 2 ?move-info) ?max-3 (nth$ 1 ?move-info)))

(assert (move-link ?max-1 ?current-1 ?max-2 ?current-2 ?max-3 ?current-3 >> ?max-1 ?current-1 ?max-2 (nth$ 2 ?move-info) ?max-3 (nth$ 1 ?move-info)))

)

)

(defrule direct-path

(move-link ?max-1-first ?current-1-first ?max-2-first ?current-2-first ?max-3-first ?current-3-first >> ?max-1-second ?current-1-second ?max-2-second ?current-2-second ?max-3-second ?current-3-second)

=>

(assert (path from ?max-1-first ?current-1-first ?max-2-first ?current-2-first ?max-3-first ?current-3-first to ?max-1-second ?current-1-second ?max-2-second ?current-2-second ?max-3-second ?current-3-second

(str-cat "( " ?max-1-first " " ?current-1-first ", " ?max-2-first " " ?current-2-first ", " ?max-3-first " " ?current-3-first " )

>> ( " ?max-1-second " " ?current-1-second ", " ?max-2-second " " ?current-2-second ", " ?max-3-second " " ?current-3-second " )")))

)

(defrule indirect-path

(path from ?max-1-first ?current-1-first ?max-2-first ?current-2-first ?max-3-first ?current-3-first to ?max-1-second ?current-1-second ?max-2-second ?current-2-second ?max-3-second ?current-3-second ?route)

(move-link ?max-1-second ?current-1-second ?max-2-second ?current-2-second ?max-3-second ?current-3-second >> ?max-1-third ?current-1-third ?max-2-third ?current-2-third ?max-3-third ?current-3-third)

=>

(assert (path from ?max-1-first ?current-1-first ?max-2-first ?current-2-first ?max-3-first ?current-3-first to ?max-1-third ?current-1-third ?max-2-third ?current-2-third ?max-3-third ?current-3-third

(str-cat ?route "

>> ( " ?max-1-third " " ?current-1-third ", " ?max-2-third " " ?current-2-third ", " ?max-3-third " " ?current-3-third " )")))

)

(defrule display-path

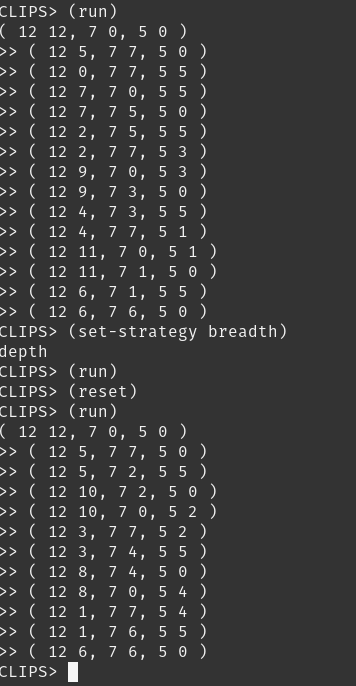
(path from 12 12 7 0 5 0 to 12 6 7 6 5 0 ?route)

=>

(printout t ?route crlf)

)

**Виконання програми**:



**Контрольні питання:**

BFS:

BFS(G, s)

let Q be queue.

Q.enqueue(s)

mark s as visited.

while (Q is not empty)

v = Q.dequeue()

for all neighbours w of v in Graph G

if w is not visited

Q.enqueue(w)

mark w as visited.

DFS:

DFS(G, s)

mark v as discovered

for all neighbours w of v in Graph G

if w is not discovered

call DFS(G, w)

1. Метод з обмеженням глибини:

Є еквівалентом пошуку в глибину, але не заглиблюється більше зазначеної глибини.

Метод з ітеративним поглибленням:

На кожній ітерації виконує пошук з обмеженням глибини (починаючи з глибини 1) та збільшує максимальну глибину на 1.

1. DFS:

* не оптимальний
* не повний:

Може занурутись в занадто глибоку гілку

BFS:

* оптимальний
* повний

DLS:

* не оптимальний

так само як і звичайний пошук в глибину

* не повний (повний якщо ціль знаходиться в межах зазначеної глибини)

так само як і звичайний пошук в глибину

IDDFS:

* оптимальний

Через оптимізації усуває проблеми пошуку в глибину

* повний

Через обмеження глибини гарантовано не заглибиться в занадто глибокі частини графу

**Висновок:** в даній лабораторній роботі мною було розроблено генерацію станів та переходів для відповідної задачі та два методи неінформативного пошуку: в глибину та в ширину. Метод пошуку в ширину є оптимальним та повним, тому знайдена відповідь є оптимальною.