Реализация задачи 11

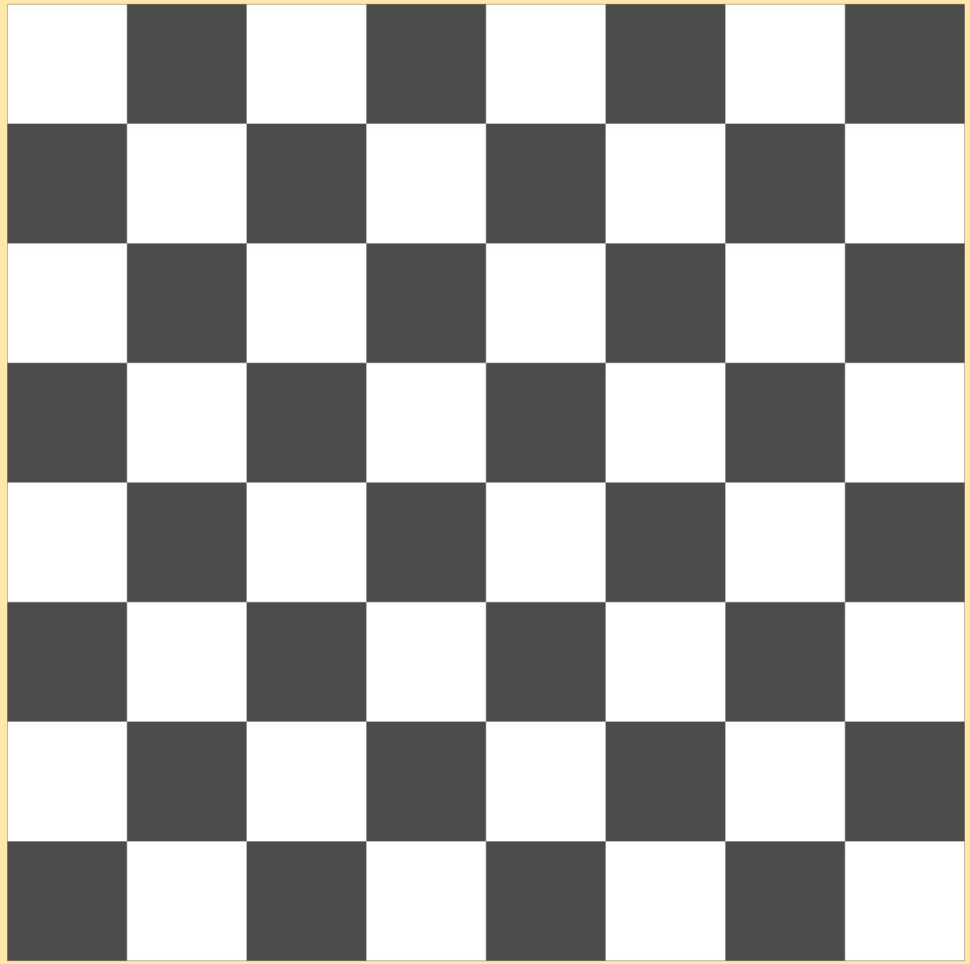
«Интеллектуальные Информационные Системы»

Низамов Максим

СГУ им. Чернышевского

Направление подготовки: (09.03.03) — Прикладная информатика

г. Саратов 2021г.



Заданное выражение

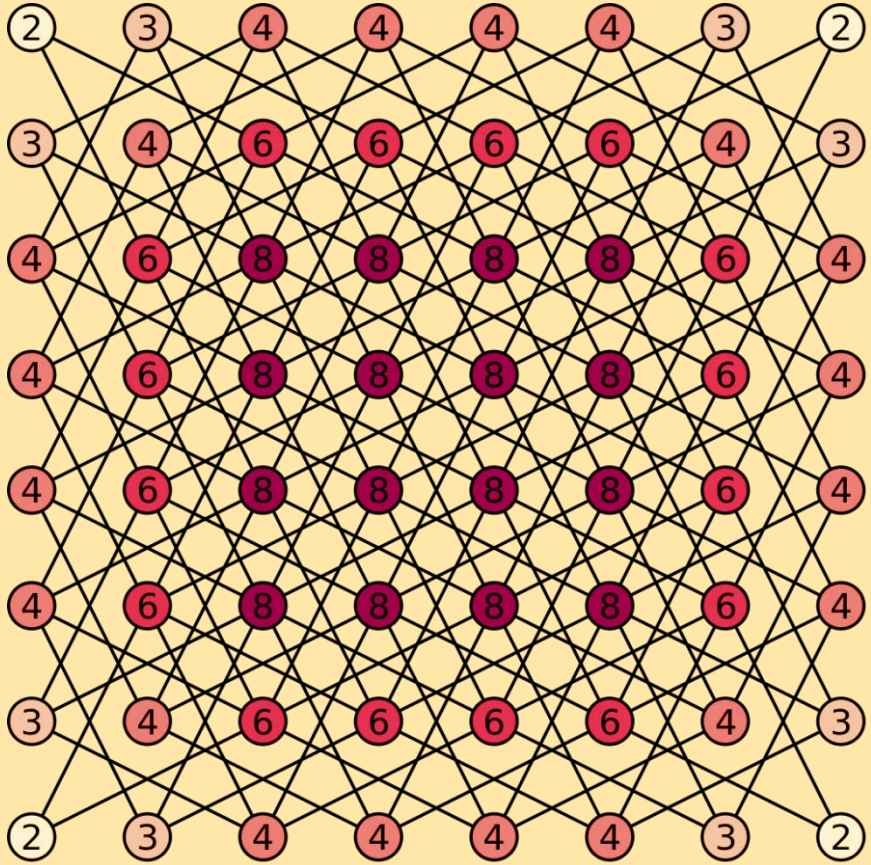
Реализовать алгоритм решения задачи о поиске последовательности перемещений коня на шахматной доске размера $m \times n$ (например, 4×4 или 4×5) из заданной начальной клетки (нижняя левая клетка) в нее же, при этом надо побывать хотя бы по одному разу на всех остальных клетках доски

Алгоритм

Заданное выражение является разновидностью задачи о ходе коня (Knight's tour), однако мы не связаны с ограничением один шаг на клетку

Простейшее решение нашей задачи сводится к представлению шахматного поля в качестве графа и поиск пути на нем

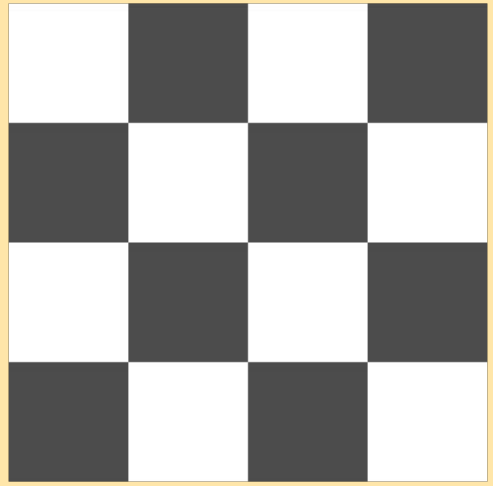
Так клетки – это вершины, а ребра – возможные пути



Алгоритм

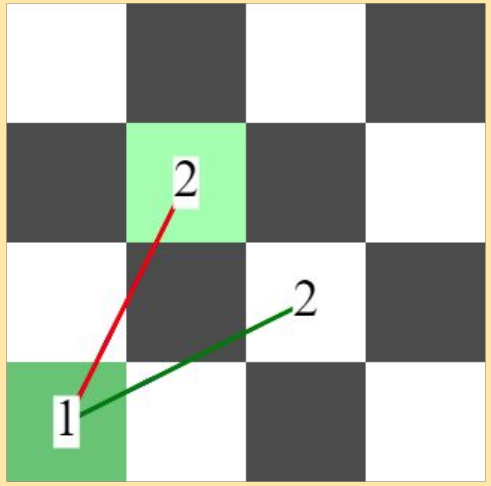
Мы можем выстроить алгоритм, основанный на обходе в глубину с некоторыми оптимизациями

К примеру, не возвращаться на предыдущую клетку в процессе поиска. Это позволит сократить количество зацикливаний



Алгоритм

При очередном выборе следующей клетки мы базируемся на информации о уже посещенных из возможных

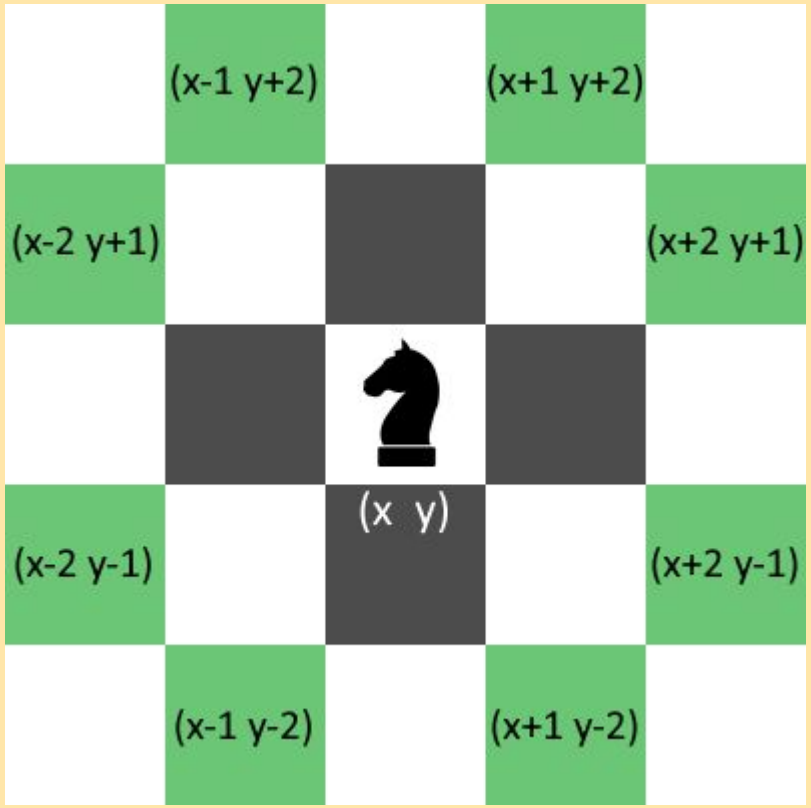


Алгоритм

Поскольку фигура коня может передвигаться по определенному правилу, будем вычислять доступные шаги посредством следующего списка разности:

((2 1) (1 2) (-1 2) (-2 1) (-2 -1) (-1 -2) (1 -2) (2 -1))

Координаты клеток в свою очередь будем хранить в формате (x y)



Реализация

Для начала определим несколько вспомогательных функций

Первой идет *numlist*, возвращающая список целых числе от 1 до *size*

Далее по списку функция *is-contains,* определяющая является ли *x* элементом верхнего уровня списка *l*

```lisp
(defun numlist (size)
    (cdr
        (reverse
            (let ((out nil))
                (dotimes (n (+ 1 size) out)
                (setq out (cons n out)))
            )
        )
    )
)
```

```lisp
(defun is-contains (x l)
    (dolist (el l nil)
        (when (equal el x) (return t))
    )
)
```

Реализация

Функция *concat*. Принимает на вход список *l* и возвращает соединение всех его внутренних элементов

```
1  (defun concat (l)
2      (let ((out '()))
3          (dolist (el (reverse l) out)
4              (setq out (append el out))
5          )
6      )
7  )
```

Функция foldr. Поведение соответствует аналогичной функции в я.п. Haskell. Рекурсивна со строки 5

```
1  (defun foldr (f tail l)
2      (if (null l) tail
3          (funcall f
4              (car l)
5              (foldr f tail (cdr l))
6          )
7      )
8  )
```

Реализация

Наконец начинаем описывать целевую функцию *knight*, принимающую размер поля в качестве аргументов

Имея размеры поля мы можем инициализировать список *board*, содержащий координаты всех клеток на поле. В этом нам поможет ранее описанная функция *numlist*

```
1  (defun knight (m n)
2      (setq board
3          (concat
4              (mapcar
5                  (lambda (i)
6                      (mapcar
7                          (lambda (j) (list i j))
8                          (numlist n)
9                      )
10                 )
11                 (numlist m)
12             )
13         )
14     )
```

Реализация

В локальной области видимости определим несколько контекстно-зависимых функций.

Функция *is-visited* возвращает *t* если искомая клетка *x* уже посещена из *p*

```
16      (defun is-visited (x p cb)
17          (cond
18              ((null (cdr cb)) nil)
19              ((and
20                  (equal x (car cb))
21                  (equal p (cadr cb))
22              ) t)
23              (t (is-visited x p (cdr cb)))
24          )
25      )
```

Реализация

Функция *next-step* принимает на вход текущую ветку перемещений

Возвращает все не посещенные точки, в которые возможно переместиться

```
73      (defun next-step (cb)
74          (setq p (car cb))
75          (setq s
76              (remove-if-not
77                  (lambda (x)
78                      (and
79                          (<= 1 (car x) m)
80                          (<= 1 (cadr x) n)
81                      )
82                  )
83
84                  (mapcar
85                      (lambda (x)
86                          (list
87                              (+ (car p) (car x))
88                              (+ (cadr p) (cadr x))
89                          )
90                      )
91                      '(  (2 1) (1 2) (-1 2) (-2 1)
92                          (-2 -1) (-1 -2) (1 -2) (2 -1
93                  )
94              )
95          )
96
97          (remove-if
98              (lambda (x)
99                  (is-visited x p cb)
100             )
101             s
102         )
103     )
```

Реализация

Далее идет функция *next-branches*, возвращающая слияние доступной ветки перемещения с текущей

Функция *is-branch-full* позволяет закончить обход шахматной доски, если все клетки из *board* находятся в текущей ветке *cb*

```
58      (defun next-branches (cb)
59          (mapcar
60              (lambda (x) (cons x cb))
61              (next-step cb)
62          )
63      )
64
65      (defun is-branch-full (cb)
66          (and
67              (equal (car cb) '(1 1))
68              (null
69                  (remove-if-not
70                      (lambda (x) (not (is-contains x cb)))
71                      board
72                  )
73              )
74          )
75      )
```

Реализация

Своеобразной точкой входа и последней локальной функцией является функция *move*. Она проводит рекурсивный проход по текущей ветке использую *foldr* для выбора направления
В 95 строке мы запускаем обход из левой нижней точки

```
77      (defun move (cb)
78          (setq nbs (next-branches cb))
79          (cond
80              ((is-branch-full cb) (reverse cb))
81              ((null nbs) '())
82              (t  (foldr
83                      (lambda (x a)
84                          (cond
85                              ((null a) (move x))
86                              (t a)
87                          )
88                      )
89                      nil
90                      nbs
91                  )
92              )
93          )
94      )
95      (move '((1 1)))
```

Заключение

После вызова целевой функции результатом работы нашей программы будет список, элементы которого являются координаты клеток – последовательность перемещений

Проверить выходные данные программы вы можете на странице https://lucors.ru/knight/