

Введение в ИИ на примере языка Prolog

Алгоритмы поиска путей

<https://github.com/Inscriptor/IntroductionToAI/tree/master/pdf>

Федеральное государственное автономное образовательное учреждение высшего образования
«Новосибирский национальный исследовательский государственный университет»

2 декабря 2019 г.

Источники

1. Introduction to the A* Algorithm.
(<https://www.redblobgames.com/pathfinding/a-star/introduction.html>)
2. Введение в алгоритм A* (<https://habr.com/ru/post/331192/>)

Источники

1. Поиск в ширину.
2. Поиск в равномерной стоимости (алгоритм Дейкстры).
3. Алгоритм A^* .

Поиск в ширину

Поиск в ширину выполняет исследование равномерно во всех направлениях.

1. Выбираем и удаляем точку из границы.
2. Помечаем точку как посещённую, чтобы знать, что не нужно обрабатывать её повторно.
3. Расширяем границу, глядя на её соседей. Всех соседей, которых мы ещё не видели, добавляем к границе.

Поиск в ширину

Код на Python.

```
frontier = Queue()
frontier.put(start)
visited = {}
visited[start] = True

while not frontier.empty():
    current = frontier.get()
    for next in graph.neighbors(current):
        if next not in visited:
            frontier.put(next)
            visited[next] = True
```

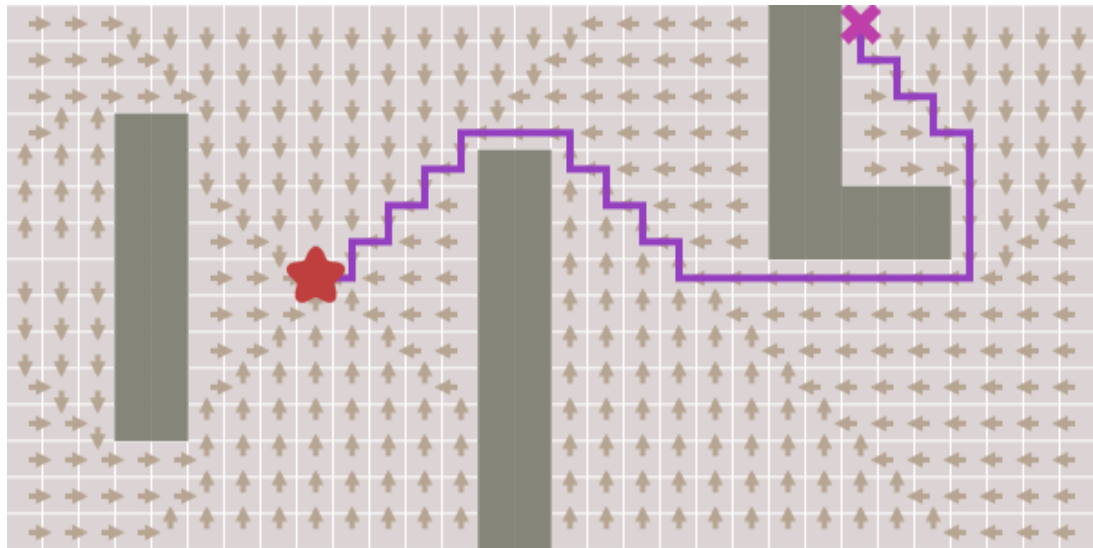
Поиск в ширину

Если мы хотим найти путь, то для каждой вершины надо запомнить, откуда мы в нее пришли.



```
frontier = Queue()
frontier.put(start)
came_from = {}
came_from[start] = None


while not frontier.empty():
    current = frontier.get()
    for next in graph.neighbors(current):
        if next not in came_from:
            frontier.put(next)
            came_from[next] = current
```

Поиск в ширину



Ранний выход

8	7	6	7	8	9	10	11	12	13	14	15	16	17	18
7	6	5	6	7	8	9	10	11	12	13	14	15	16	17
6	5	4	5	6	7	8	9	10	11	12	13	14	15	16
5	4	3	4	5	6	7	8	9	10	11	12	13	14	15
4	3	2	3	4	5						13	14	15	16
3	2	1	2	3	4						16	17	16	15
2	1		1	2	3						15	16	17	16
3	2	1	2	3	4						14	15	16	17
4	3	2	3	4	5						13	14	15	16
5	4	3	4	5	6						12	13	14	15
6	5	4	5	6	7						11	12		14
7	6	5	6	7	8	9	10	11	12	13	14	15	16	17
8	7	6	7	8	9	10	11	12	13	14	15	16	17	18
9	8	7	8	9	10	11	12	13	14	15	16	17	18	19
10	9	8	9	10	11	12	13	14	15	16	17	18	19	20

8	7	6	7	8	9	10	11	12	13	14				
7	6	5	6	7	8	9	10	11	12	13	14			
6	5	4	5	6	7	8	9	10	11	12	13	14		
5	4	3	4	5	6	7	8	9	10	11	12	13	14	
4	3	2	3	4	5							13	14	
3	2	1	2	3	4							14		
2	1		1	2	3									
3	2	1	2	3	4									
4	3	2	3	4	5									
5	4	3	4	5	6									
6	5	4	5	6	7									
7	6	5	6	7	8	9	10	11	12	13				
8	7	6	7	8	9	10	11	12	13	14				
9	8	7	8	9	10	11	12	13	14					
10	9	8	9	10	11	12	13	14						

Ранний выход

Мы должны просто прекратить расширять границу, как только нашли цель.

```
frontier = Queue()
frontier.put(start)
came_from = {}
came_from[start] = None



while not frontier.empty():
    current = frontier.get()

    if current == goal:
        break

    for next in graph.neighbors(current):
        if next not in came_from:
            frontier.put(next)
            came_from[next] = current
```

Поиск с равномерной стоимостью

Количество шагов

5	4	5	6	7	8	9	10	11	12
4	3	4	5	6	7	8	9	10	11
3	2	3	4	5	6	7	8	9	10
2	1	2	3	4	5	6	7	8	9
1		1	2	3	4	5	6	7	8
2	1	2	3	4	5	6	7		9
3	2	3	4	5	6	7	8	9	10
4				6	7	8	9	10	11
5				7	8	9	10	11	12
6	7	8	9	8	9	10	11	12	13

Расстояние

5	4	5	6	7	8	9	10	11	12
4	3	4	5	10	13	10	11	12	13
3	2	3	4	9	14	15	12	13	14
2	1	2	3	8	13	18	17	14	15
1		1	6	11	16	21	20	15	16
2	1	2	7	12	17	22	21		17
3	2	3	4	9	14	19	16	17	18
4				14	19	18	15	16	17
5				15	16	13	14	15	16
6	7	8	9	10	11	12	13	14	15

Поиск с равномерной стоимостью

Здесь нужна очередь с приоритетами и функция стоимости.

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
cost_so_far = {}
came_from[start] = None
cost_so_far[start] = 0

while not frontier.empty():
    current = frontier.get()

    if current == goal:
        break

    for next in graph.neighbors(current):
        new_cost = cost_so_far[current] + graph.cost(current, next)
        if next not in cost_so_far or new_cost < cost_so_far[next]:
            cost_so_far[next] = new_cost
            priority = new_cost
            frontier.put(next, priority)
            came_from[next] = current
```

Алгоритм A*

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
cost_so_far = {}
came_from[start] = None
cost_so_far[start] = 0

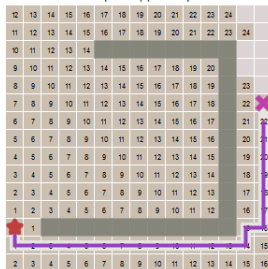
while not frontier.empty():
    current = frontier.get()

    if current == goal:
        break

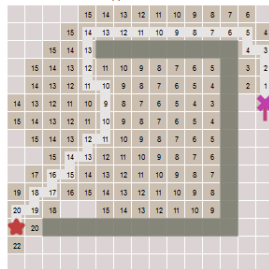
    for next in graph.neighbors(current):
        new_cost = cost_so_far[current] + graph.cost(current, next)
        if next not in cost_so_far or new_cost < cost_so_far[next]:
            cost_so_far[next] = new_cost
            priority = new_cost + heuristic(goal, next)
            frontier.put(next, priority)
            came_from[next] = current
```

Алгоритм A*

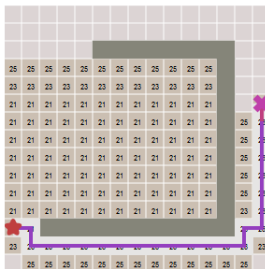
Алгоритм Дейкстры



Жадный поиск



Поиск A*



Дополнительно

1. Реализация алгоритма A*
(<https://www.redblobgames.com/pathfinding/a-star/implementation.html>)
2. Перевод на хабре (<https://habr.com/en/post/331220/>)
3. Оптимизации поиска путей
(<https://www.redblobgames.com/pathfinding/grids/algorithms.html>)
4. Цикл статей по поиску путей и практическому применению теории игр
(<http://theory.stanford.edu/~amitp/GameProgramming/>)