Алгоритмы поиска пути

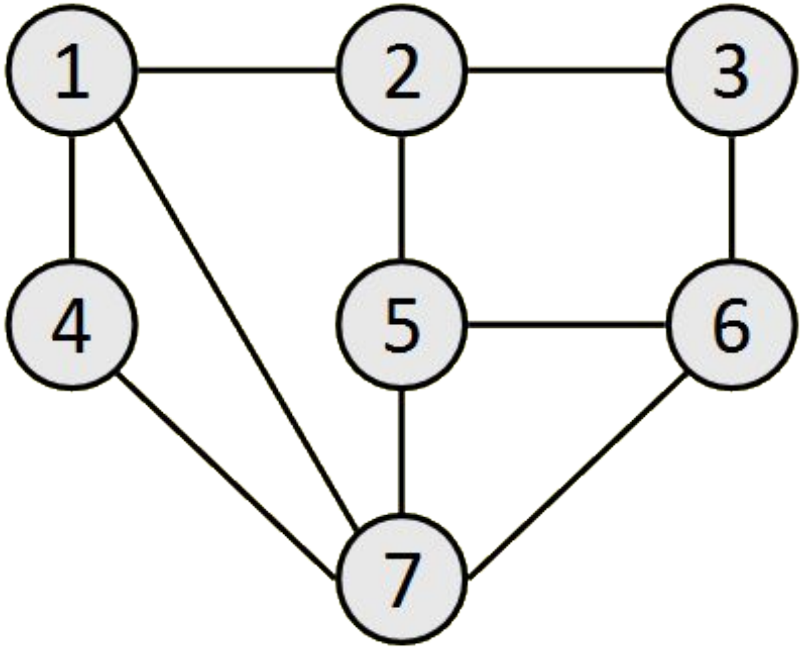
От поиска в ширину до A*

Графы: основы

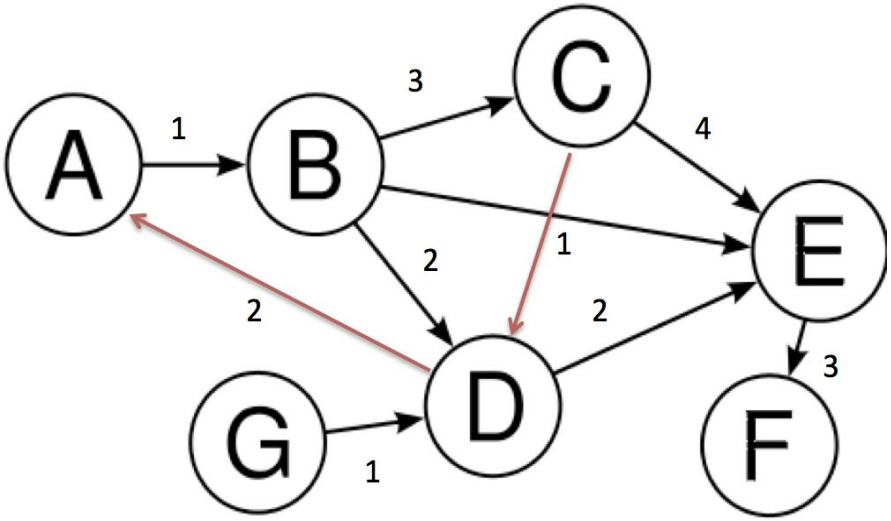
- Граф – множество вершин и ребер.
  Ребра соединяют между собой вершины.
- Графы бывают разные:
  1. Неориентированные и ориентированные
  2. Взвешенные и невзвешенные.

Графы: примеры

Неориентированный невзвешенный

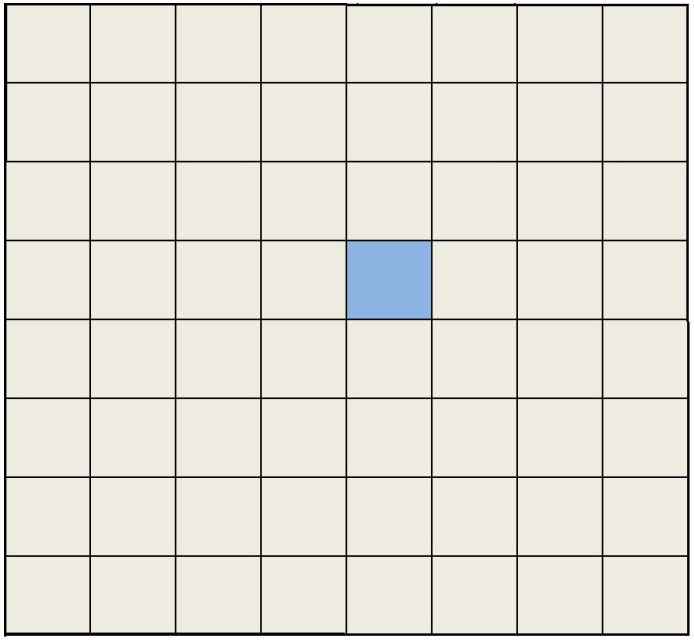


Ориентированный взвешенный



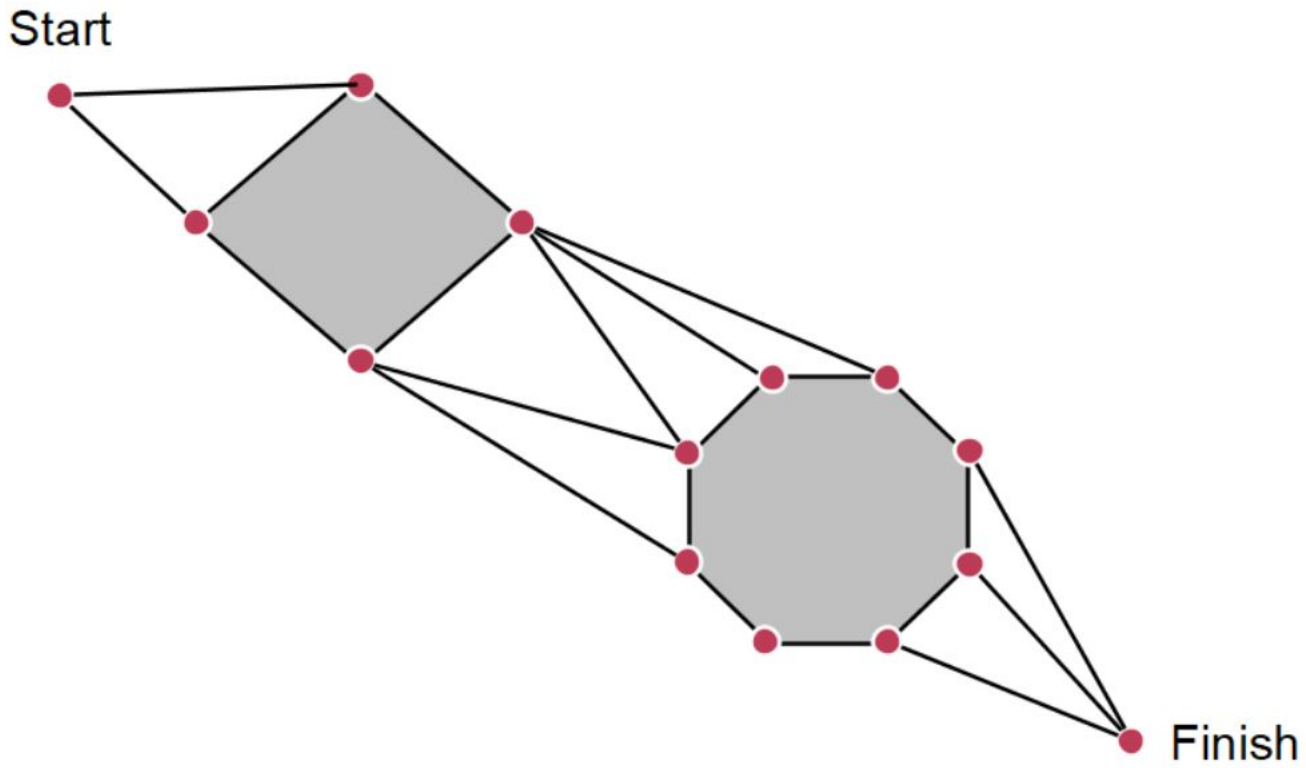
Графы в играх

Тайловая сетка(tile map, grid)



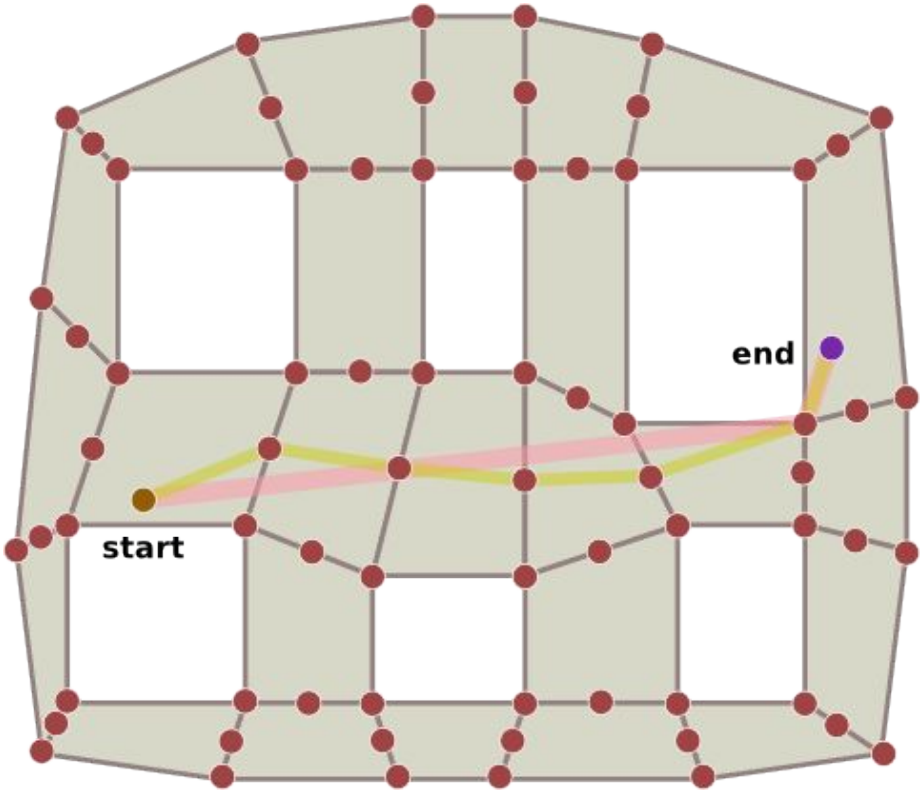
Графы в играх

Полигональная карта(polygonal map)



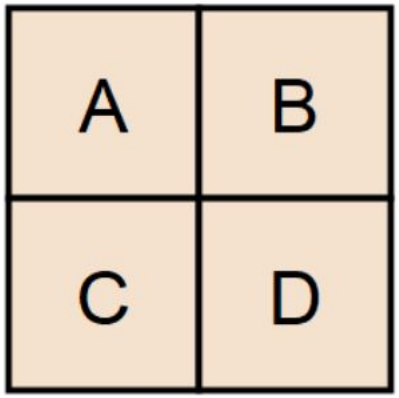
Графы в играх

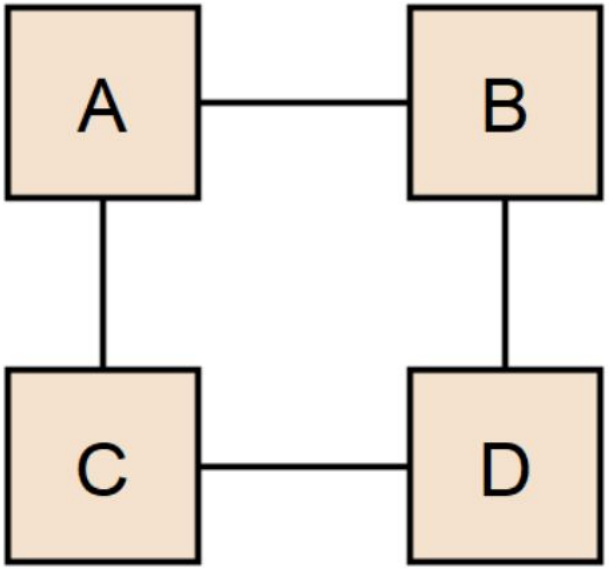
Навигационная сетка(navigation mesh)



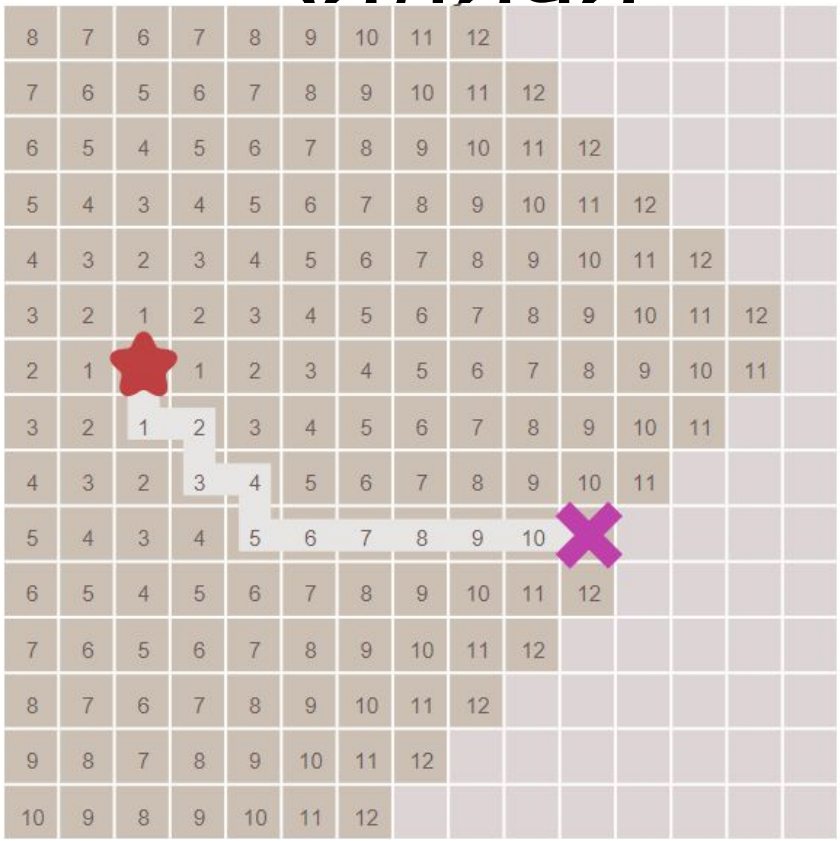
Графы в играх

Почему тайловая сетка это граф?





Поиск кратчайшего пути: задача



Есть вершина начала пути и вершина конца пути. Нужно найти кратчайший путь от начала до конца.

Поиск кратчайшего пути: общие принципы

- Разбиваем клетки на два типа: посещенные и непосещенные.
- Постепенно посещаем клетки.
- Изначально только стартовая клетка посещена.

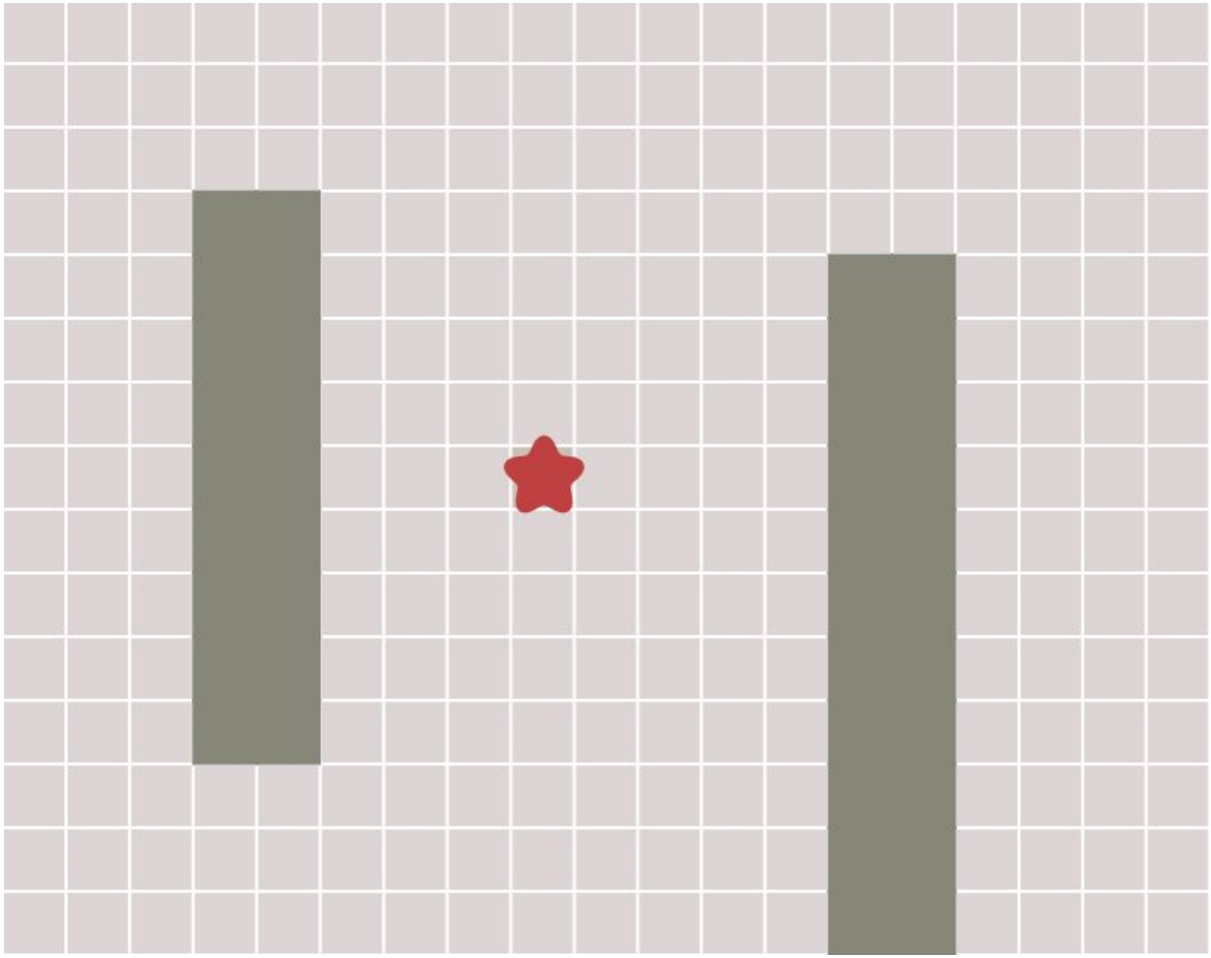
Поиск кратчайшего пути: обзор

- Поиск в ширину(breadth-first search)
- Алгоритм Дейкстры(Dijkstra's algorithm)
- Поиск первого наилучшего(best-first search)
- A*(A star)

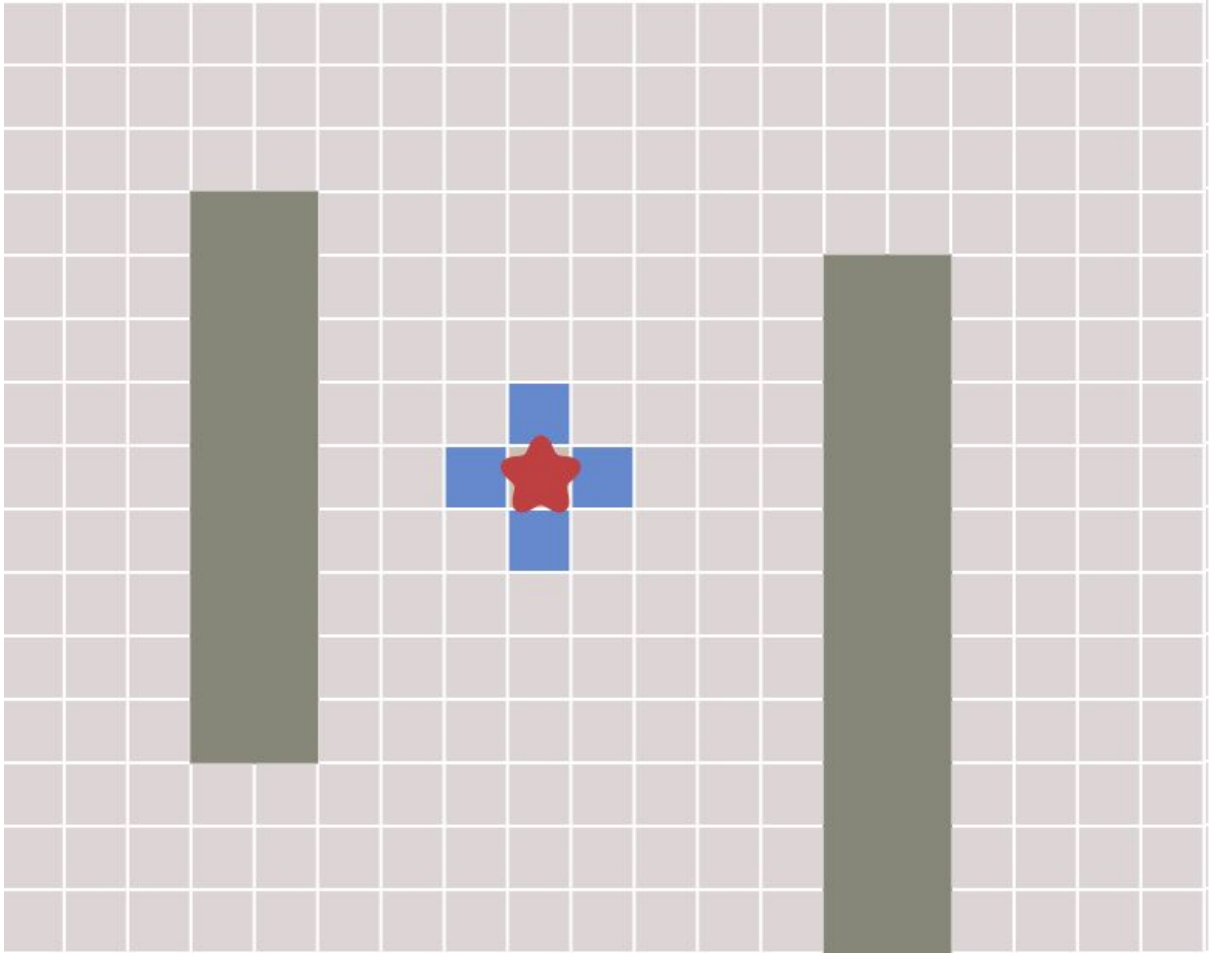
Поиск в ширину: идея

- Равномерно во все стороны расширяется радиус обхода.
- Посещенные вершины хранятся в очереди(queue).
- Заканчиваем, когда очередь пуста(изначально в очереди находится стартовая клетка).

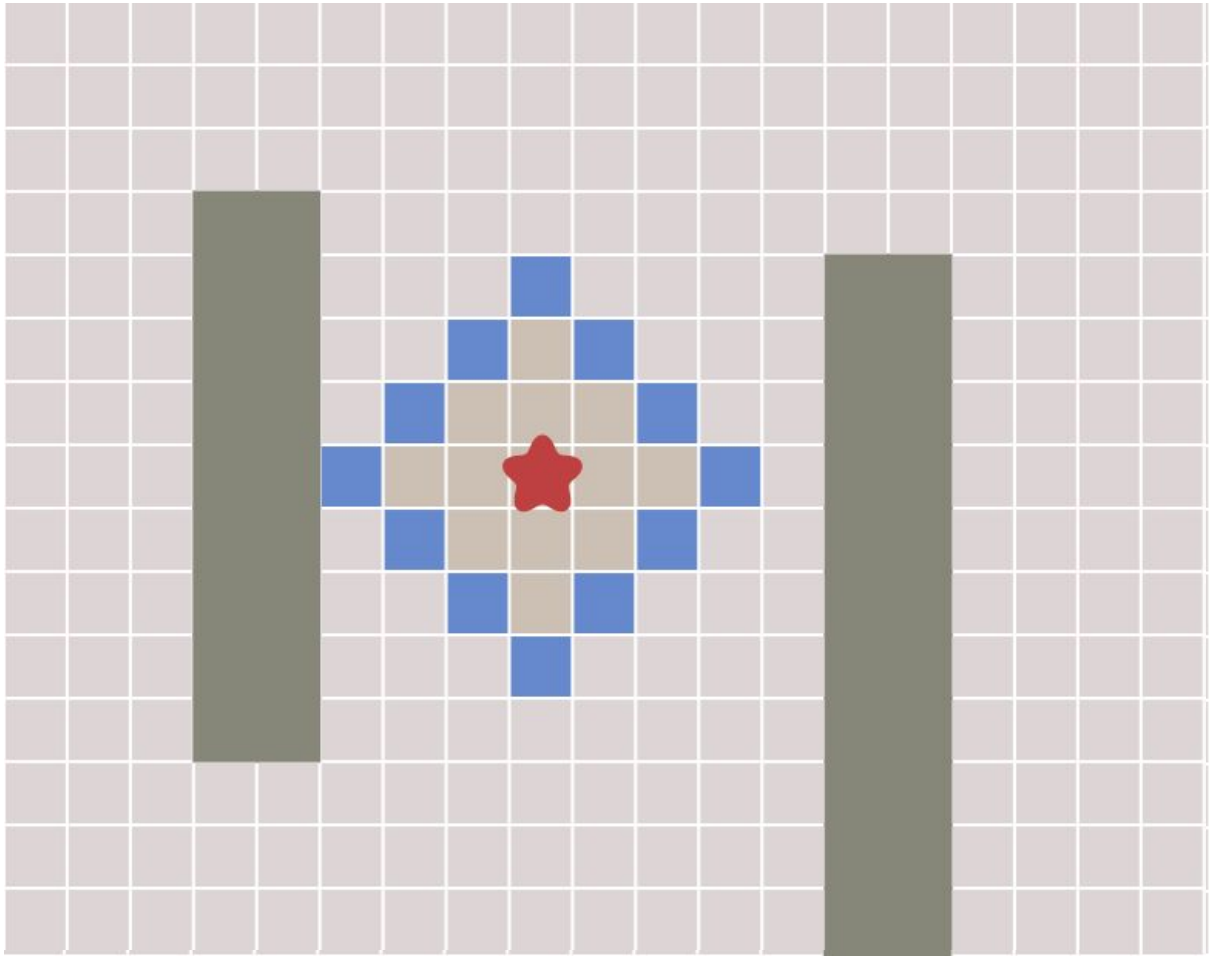
Поиск в ширину: демо



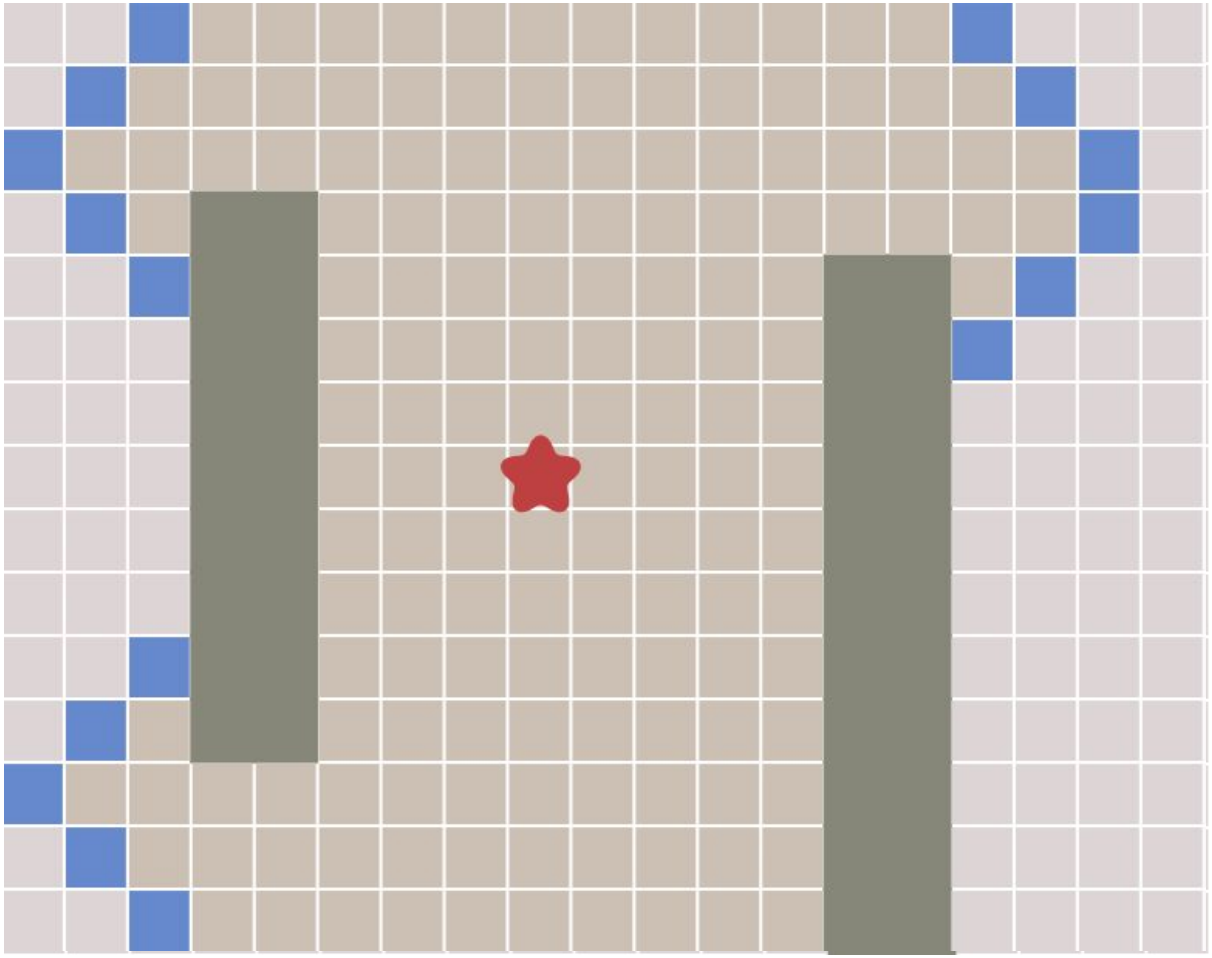
Поиск в ширину: демо



Поиск в ширину: демо



Поиск в ширину: демо



Поиск в ширину: код

Простейший вариант(инициализация)

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

Поиск в ширину: код

Простейший вариант(алгоритм)

```
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: код

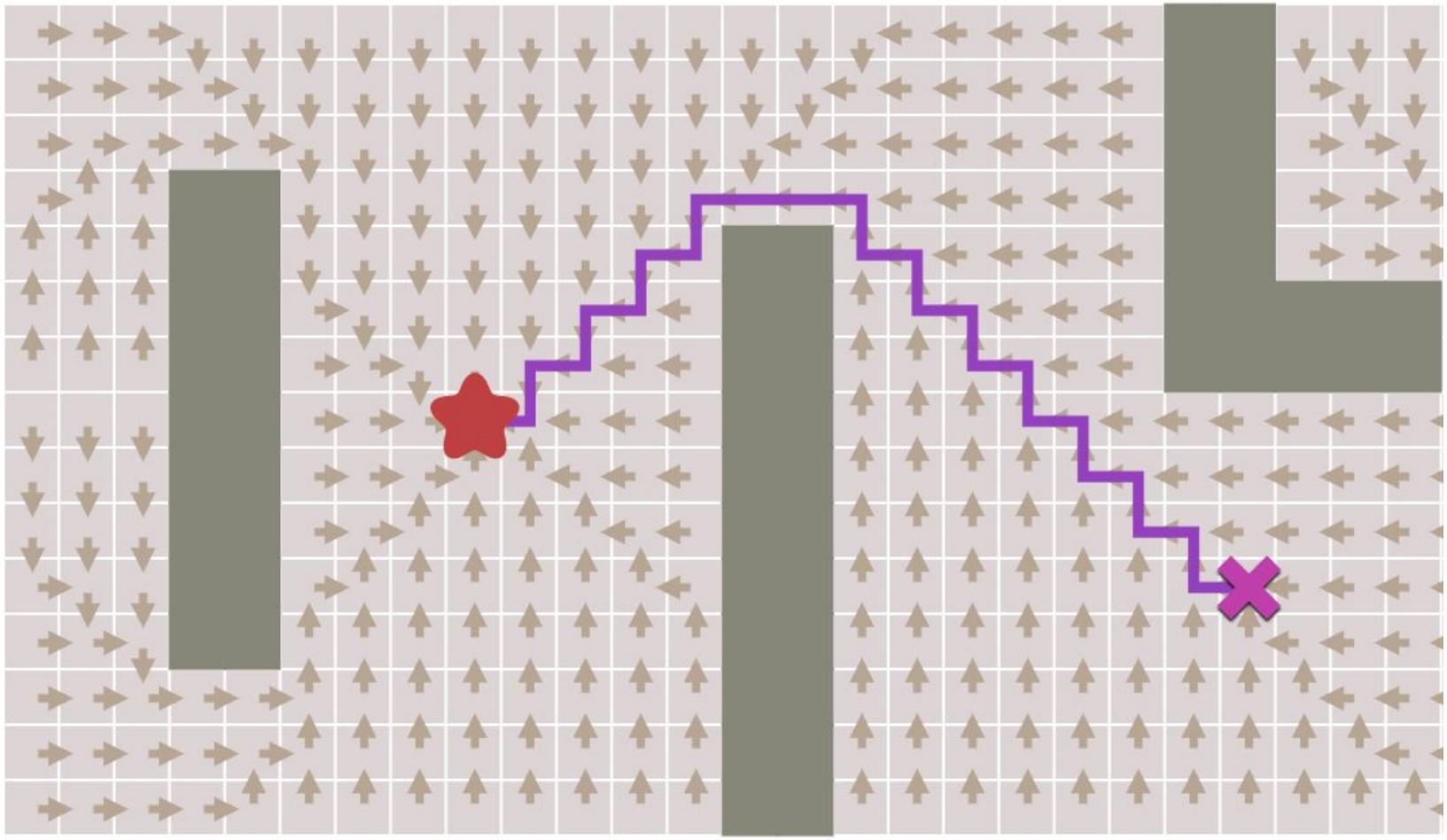
Чтобы узнать, откуда пришли(инициализация)

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

Поиск в ширину: код

Чтобы узнать, откуда пришли (алгоритм)

```
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: код

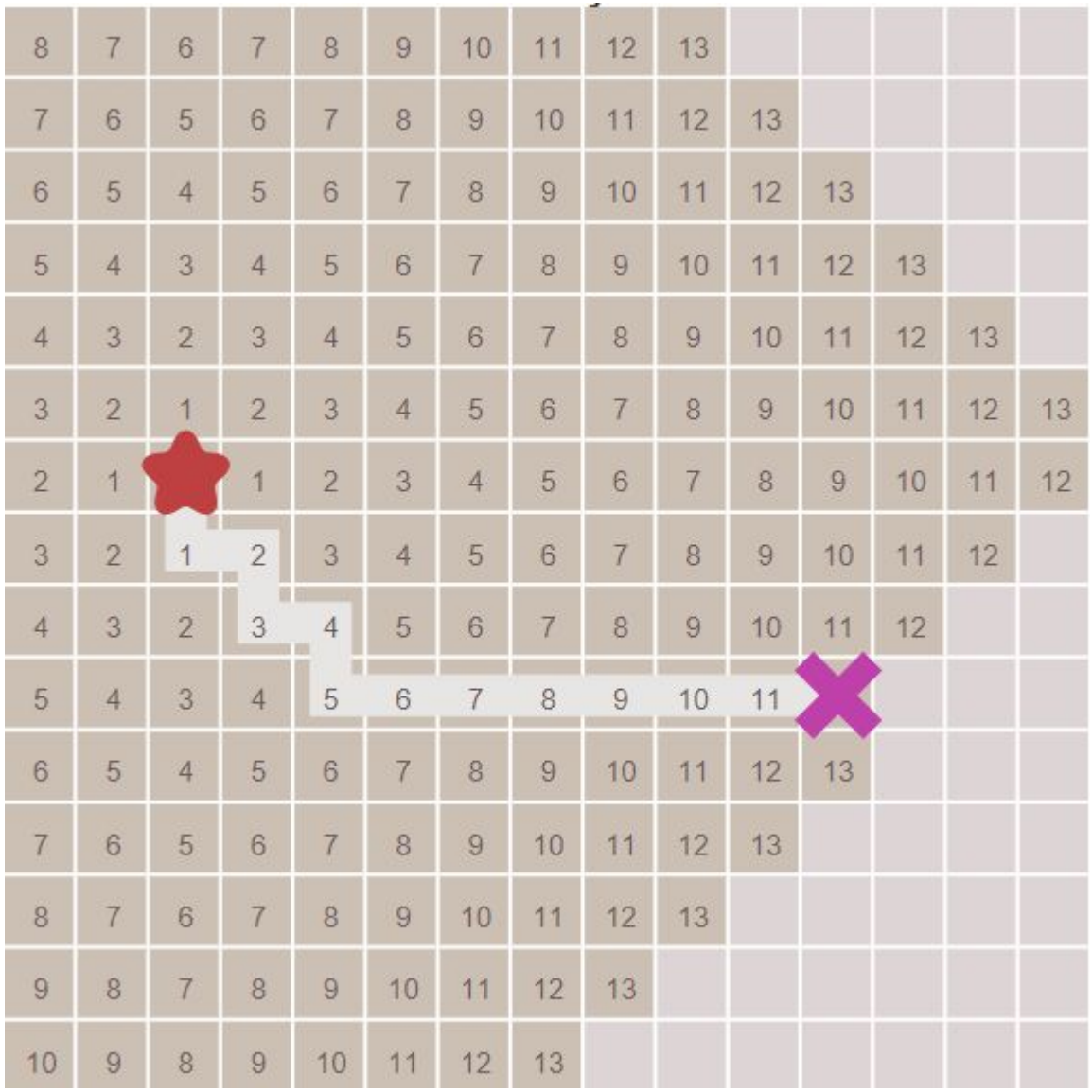
Чтобы узнать кол-во шагов (инициализация)

```
frontier = Queue()
frontier.put(start)
distance = {}
distance[start] = 0
```

Поиск в ширину: код

Чтобы узнать кол-во шагов(алгоритм)

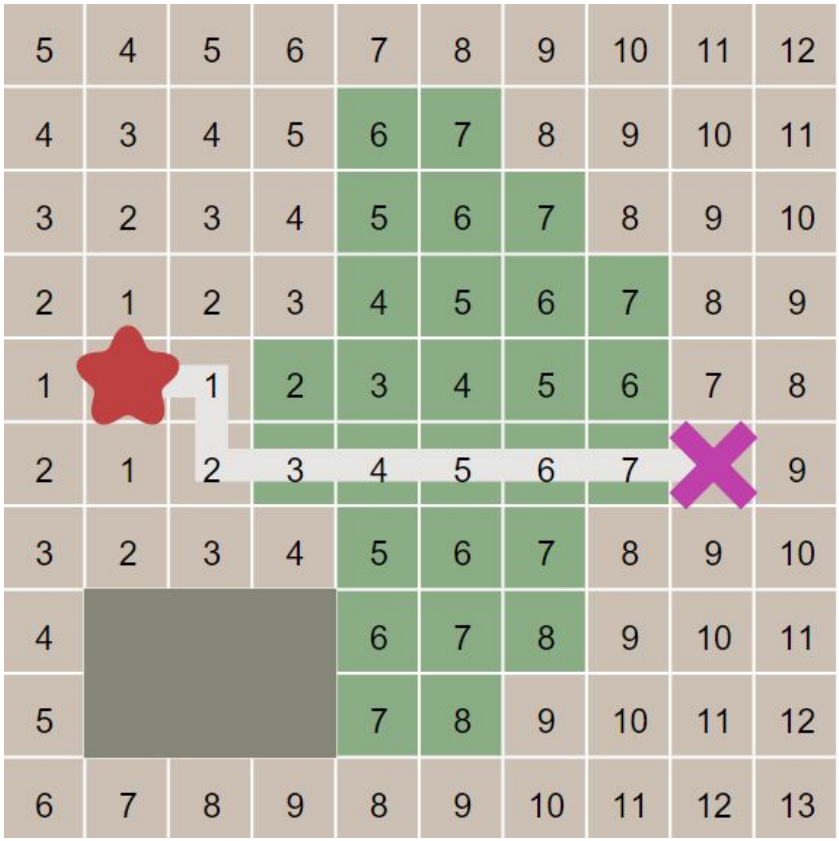
```
while not frontier.empty():
    current = frontier.get()
    for next in graph.neighbors(current):
        if next not in distance:
            frontier.put(next)
            distance[next] = distance[current] + 1
```

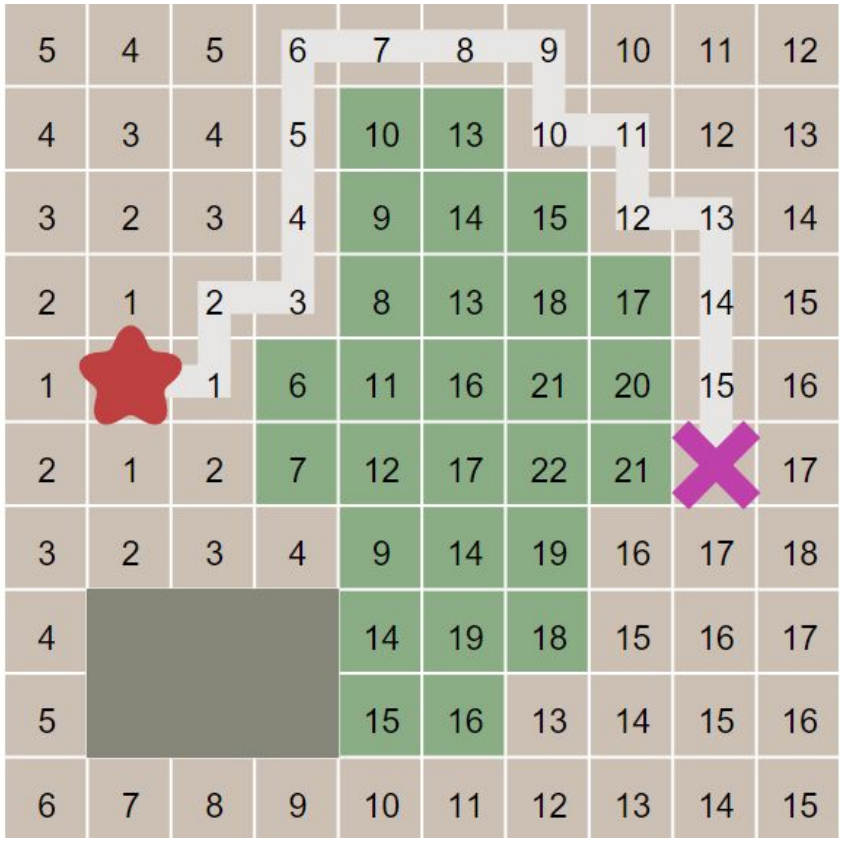

8 7 6 7 8 9 10 11 12 13
7 6 5 6 7 8 9 10 11 12 13
6 5 4 5 6 7 8 9 10 11 12 13
5 4 3 4 5 6 7 8 9 10 11 12 13
4 3 2 3 4 5 6 7 8 9 10 11 12 13
3 2 1 2 3 4 5 6 7 8 9 10 11 12 13
2 1 1 2 3 4 5 6 7 8 9 10 11 12
3 2 1 2 3 4 5 6 7 8 9 10 11 12
4 3 2 3 4 5 6 7 8 9 10 11 12
5 4 3 4 5 6 7 8 9 10 11
6 5 4 5 6 7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

Поиск в ширину: use cases

- Отметить все достижимые вершины из данной вершины
- Найти пути и расстояния до всех вершин из данной вершины(просмотреть, что находится рядом с героем/монстром)

Поиск в ширину: ограничения



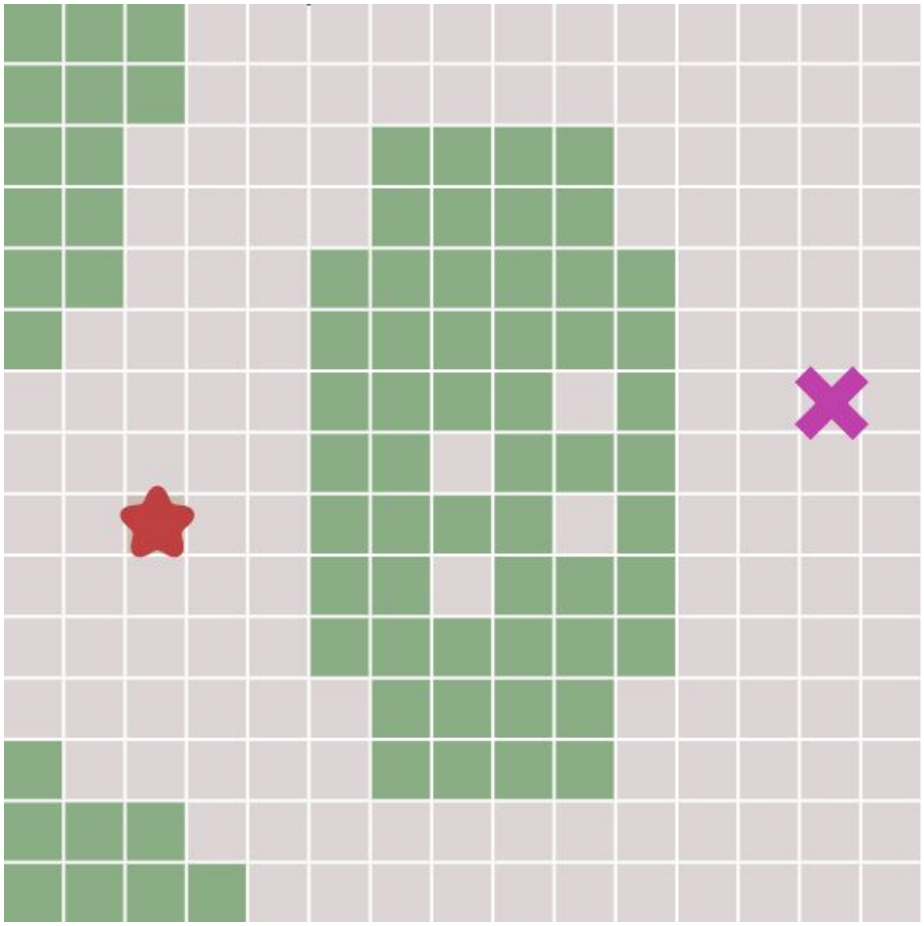


Алгоритм Дейкстры: идея

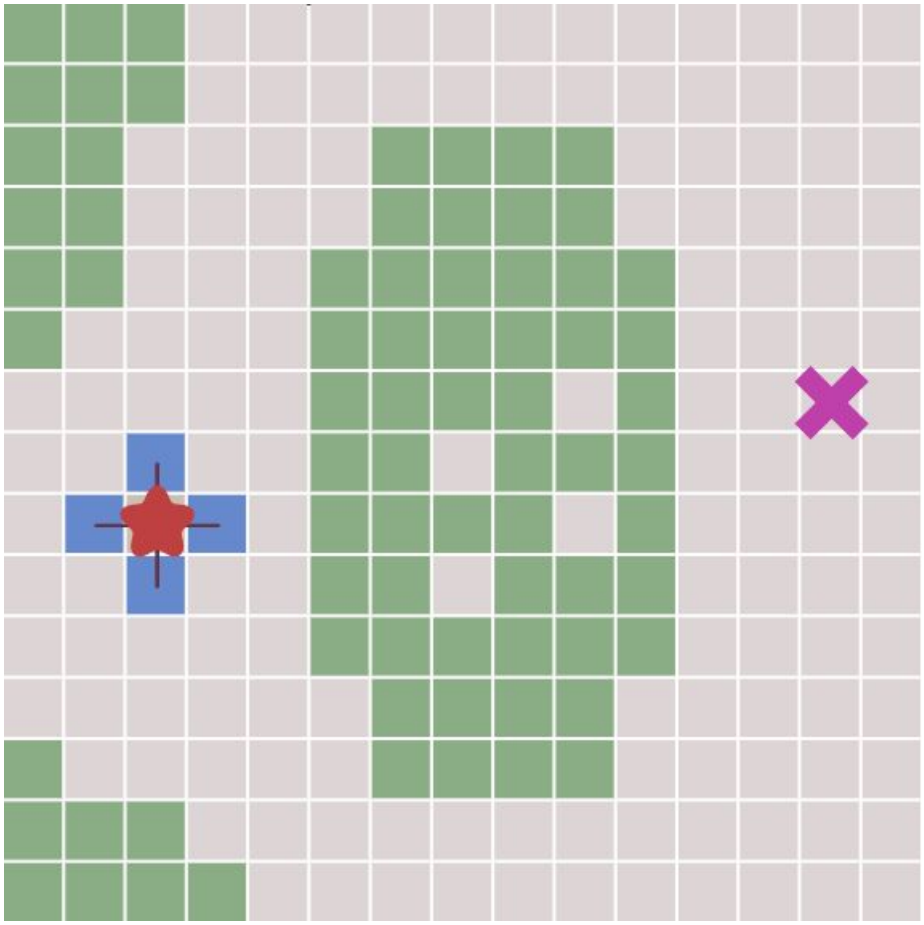
- Исследуем вершины не равномерно, а ориентируясь на расстояние до начала поиска

- Посещенные вершины хранятся в очереди с приоритетом(min priority queue) – чем меньше расстояние до вершины, тем больше ее приоритет. Т. е. тем раньше эта вершина будет исследована.

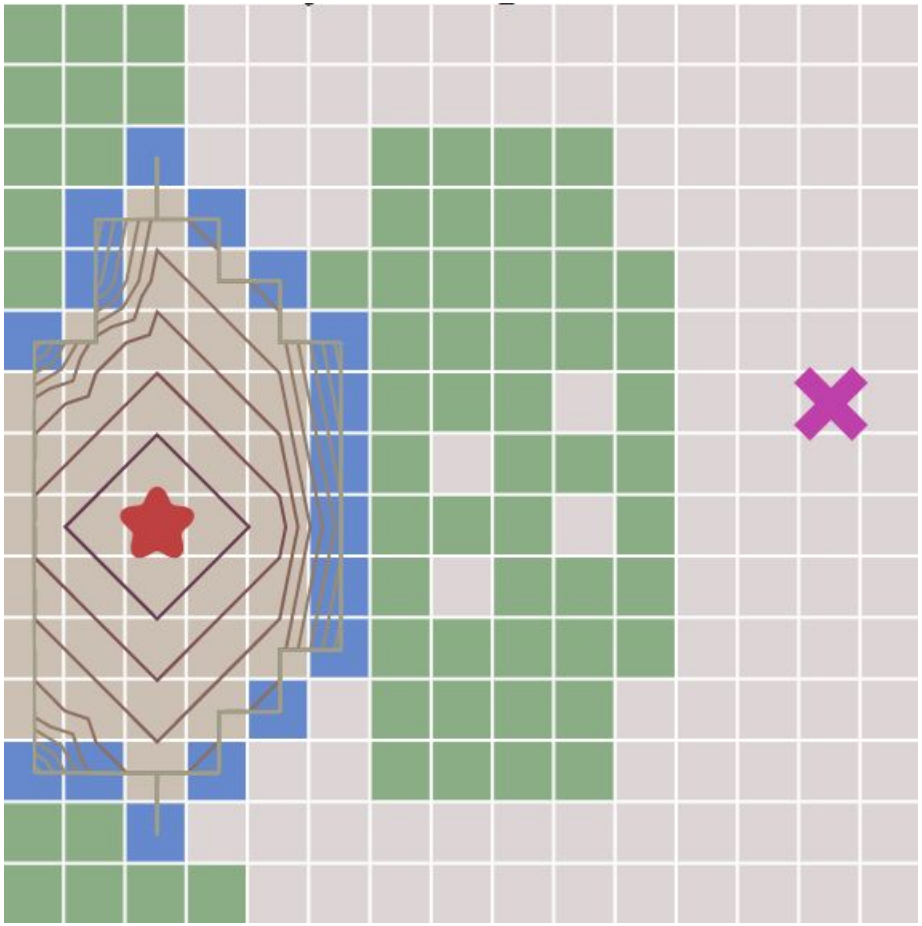
Алгоритм Дейкстры: демо



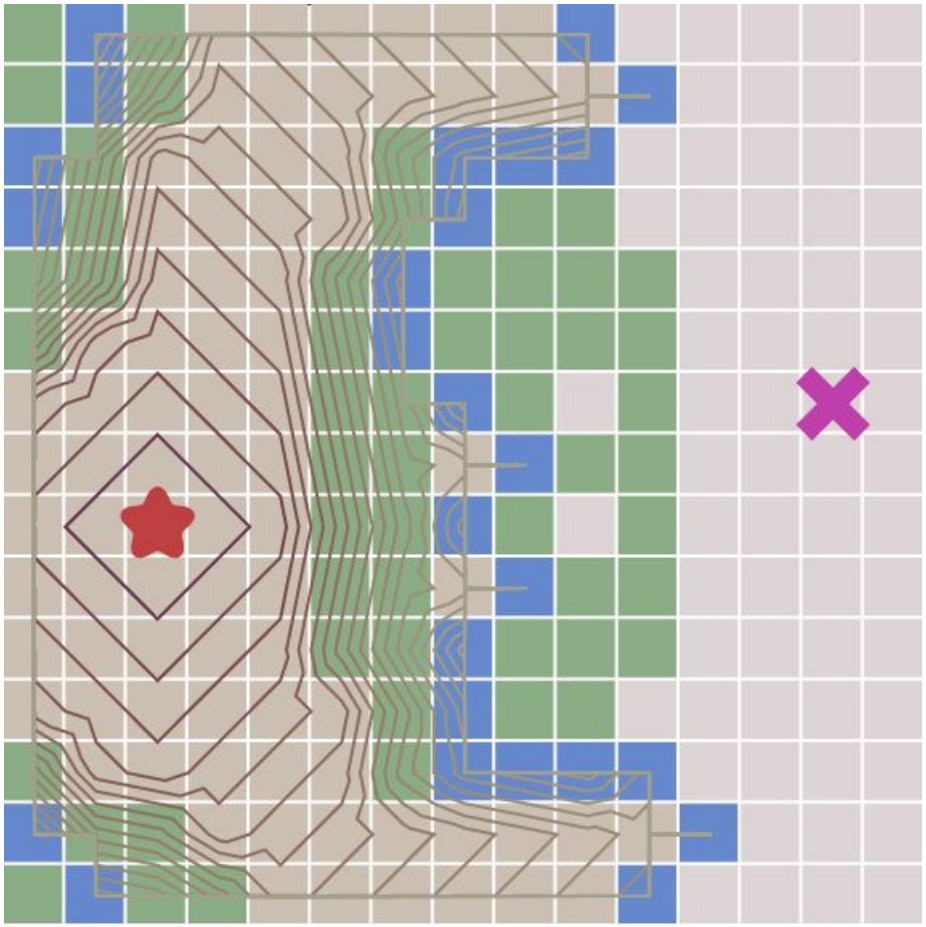
Алгоритм Дейкстры: демо



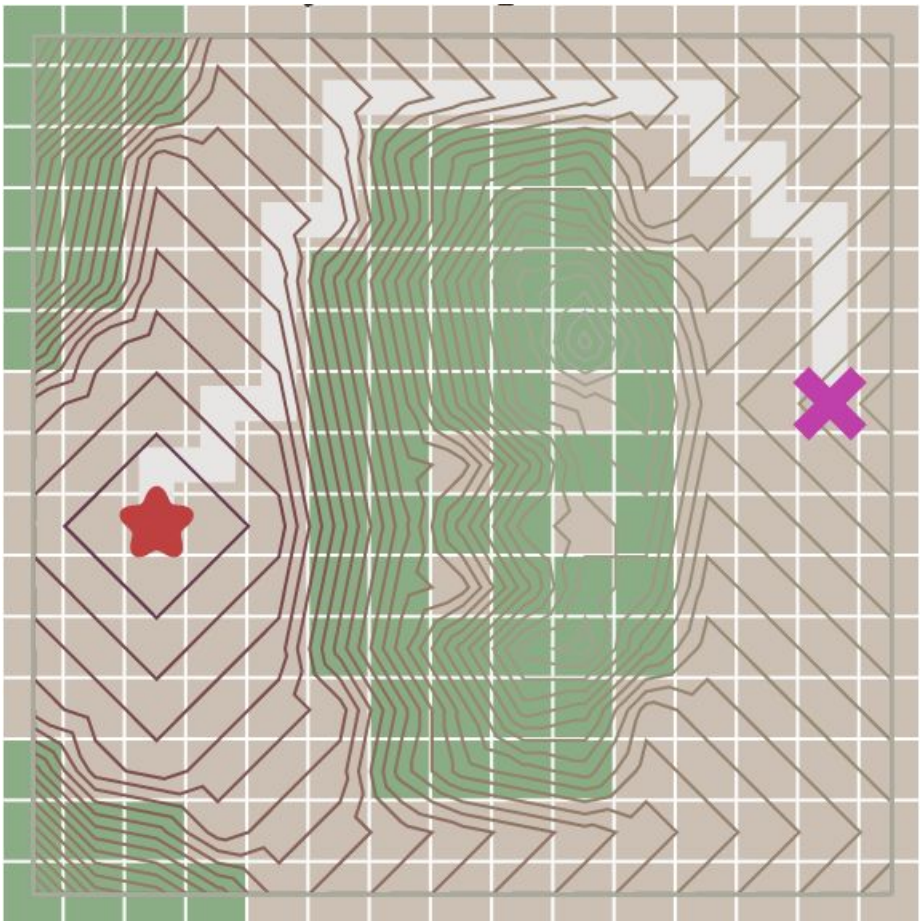
Алгоритм Дейкстры: демо



Алгоритм Дейкстры: демо



Алгоритм Дейкстры: демо



Алгоритм Дейкстры: код

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

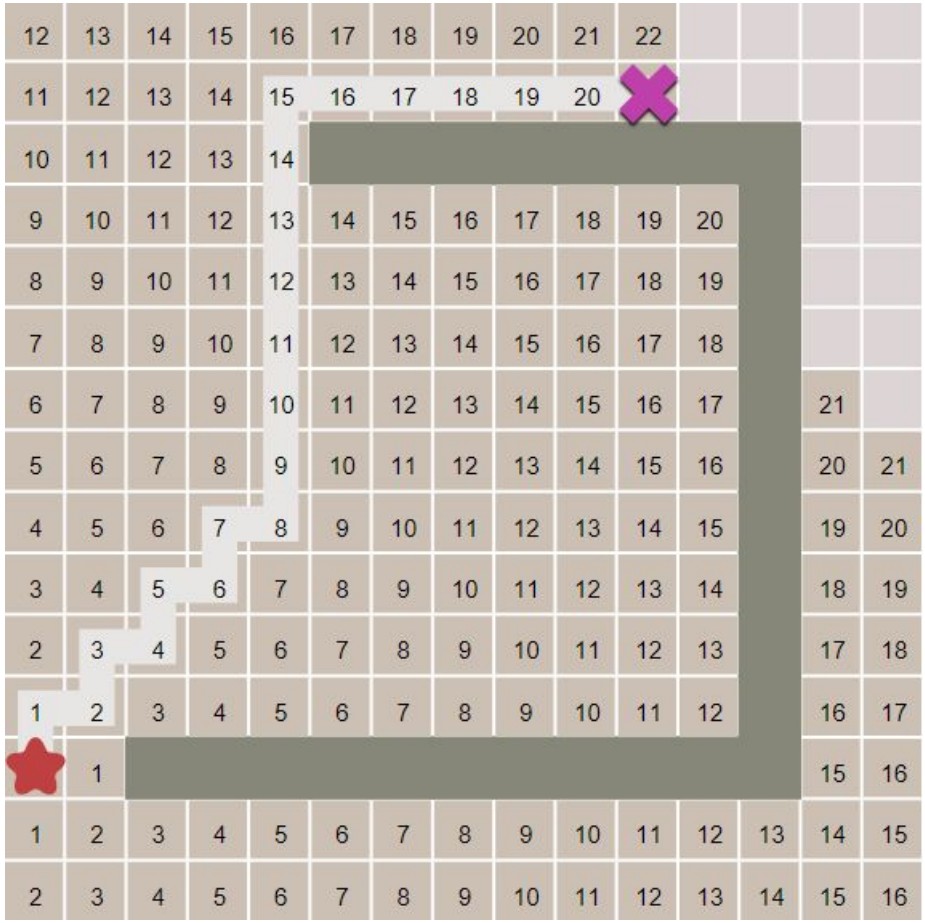
Алгоритм Дейкстры: код

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

    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
            frontier.put(next, new_cost)
            came_from[next] = current
```

Алгоритм Дейкстры: use cases

- Найти кратчайший путь от одной вершины до многих других вершин во взвешенном графе
- Когда нет знания об общей структуре графа. Т. е. мы обладаем лишь локальной информацией о графе (вблизи каждой клетки)

Алгоритм Дейкстры : ограничения

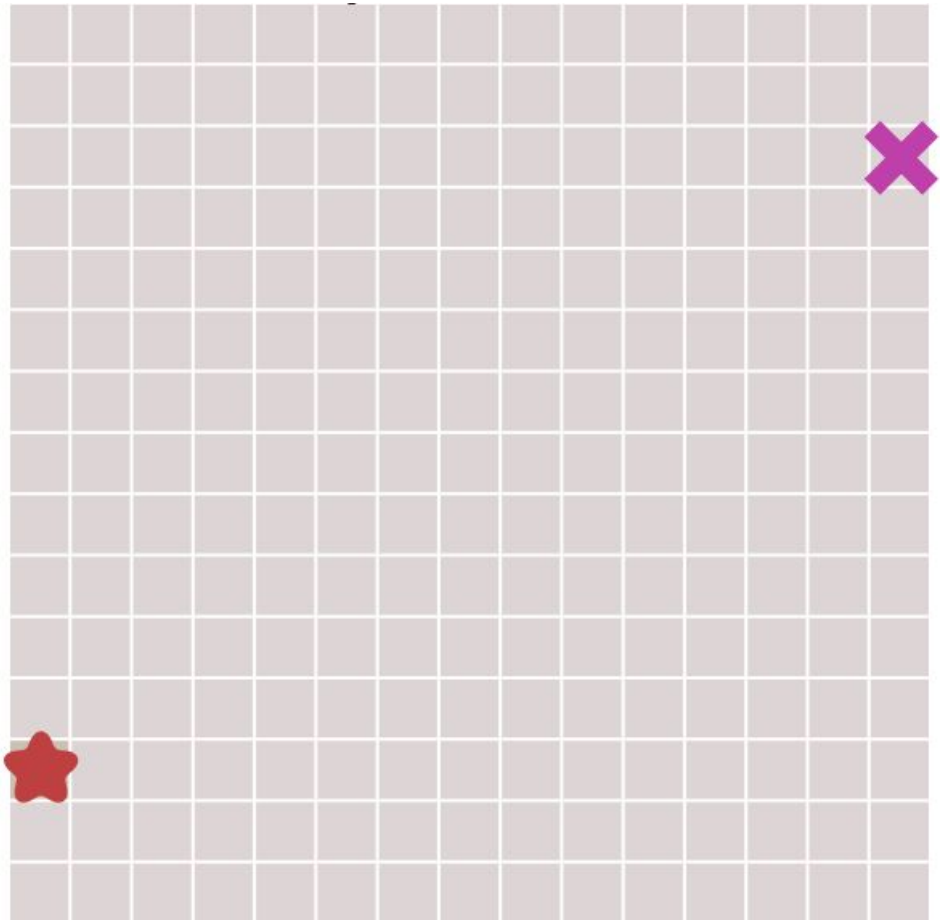


Если нужно найти путь до единственной вершины, исследуется слишком много клеток

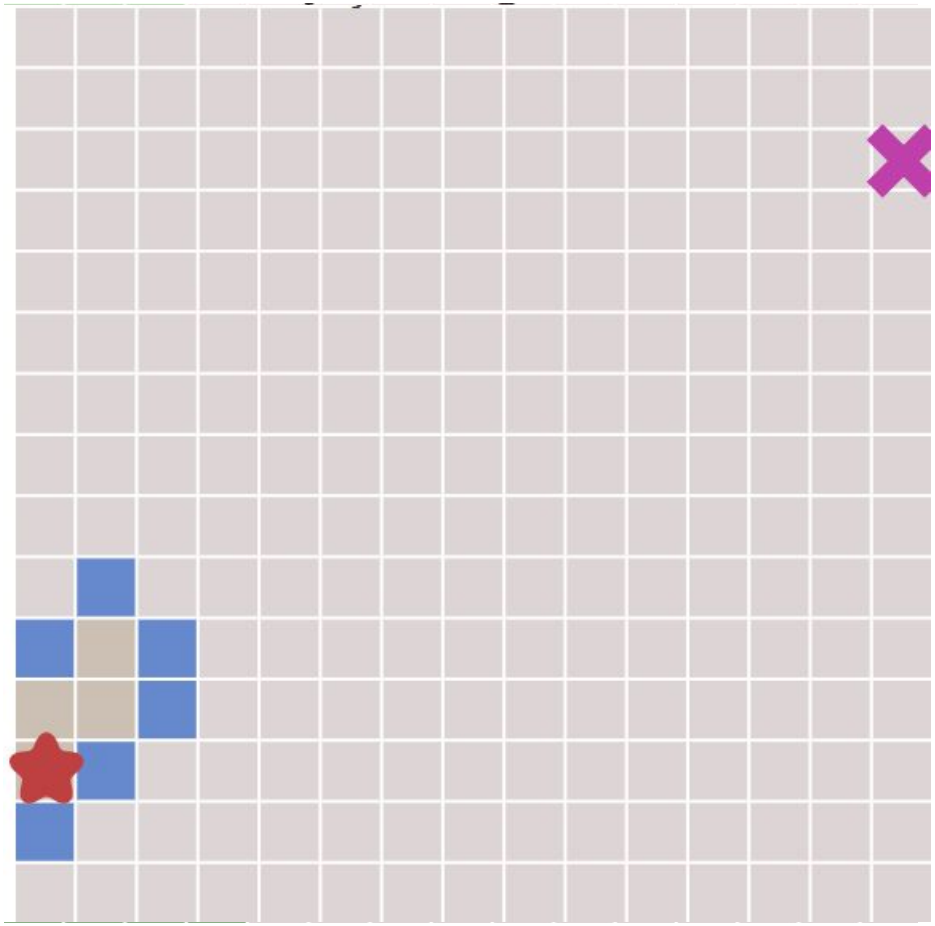
Поиск первого наилучшего: идея

- Исследуем вершины, ориентируясь на расстояние до цели
- Используем эвристику(heuristic)

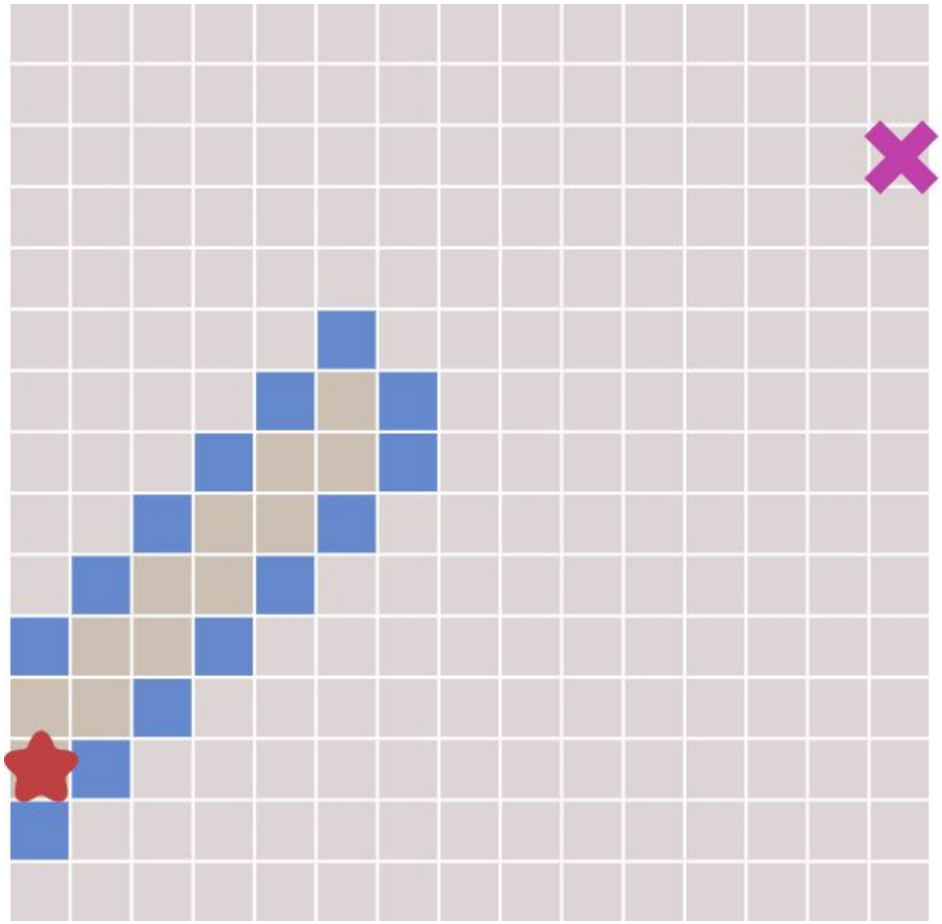
Поиск первого наилучшего: демо



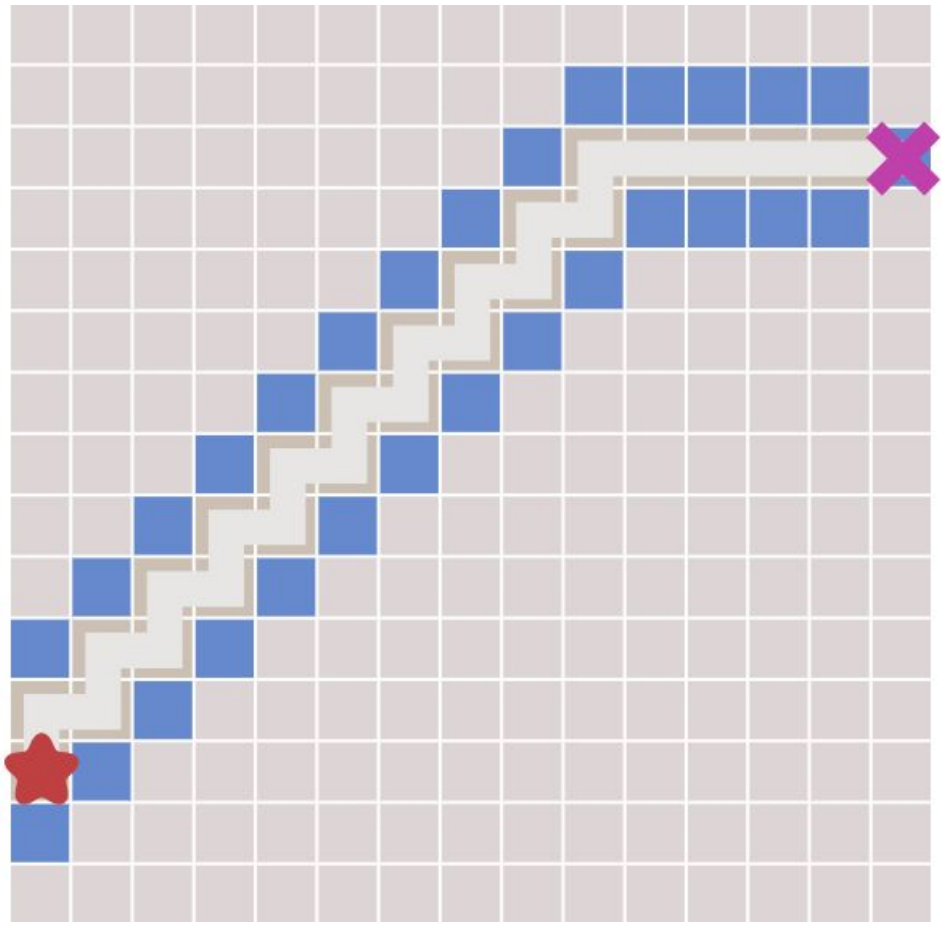
Поиск первого наилучшего: демо



Поиск первого наилучшего: демо



Поиск первого наилучшего: демо



Поиск первого наилучшего: код

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

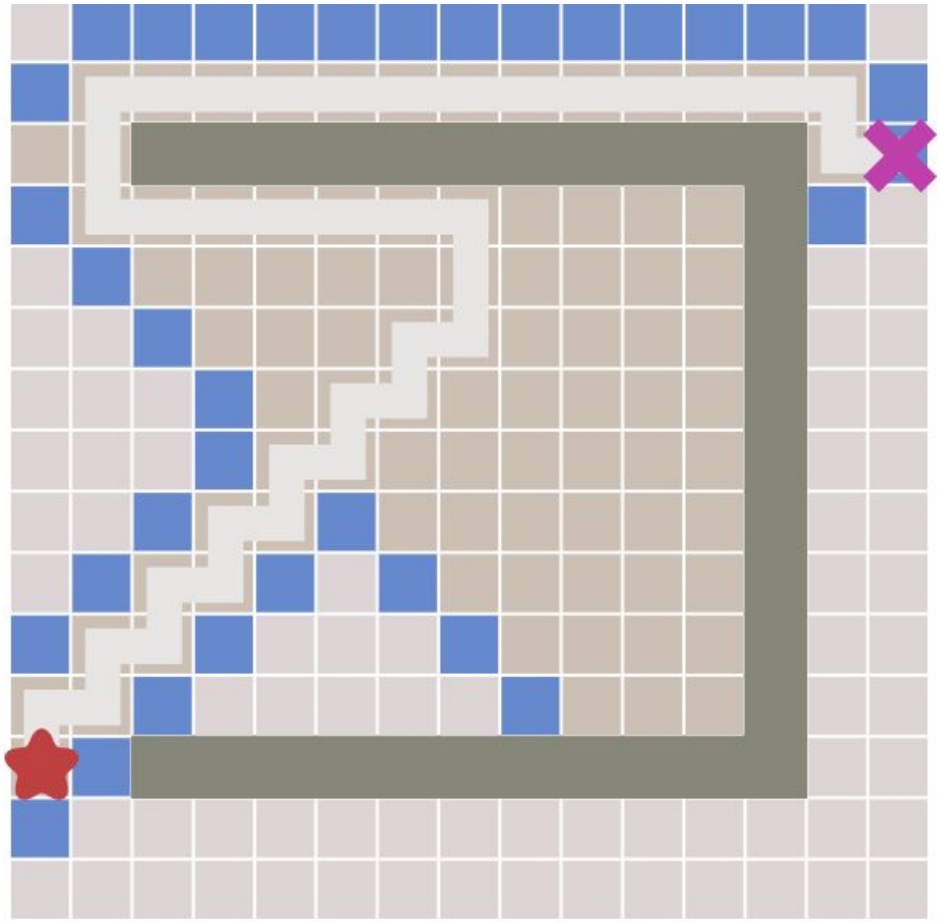
Поиск первого наилучшего: код

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

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

Поиск первого наилучшего: use cases

- Быстро найти кратчайший путь от одной вершины до другой, когда нет препятствий

Поиск первого наилучшего: ограничения

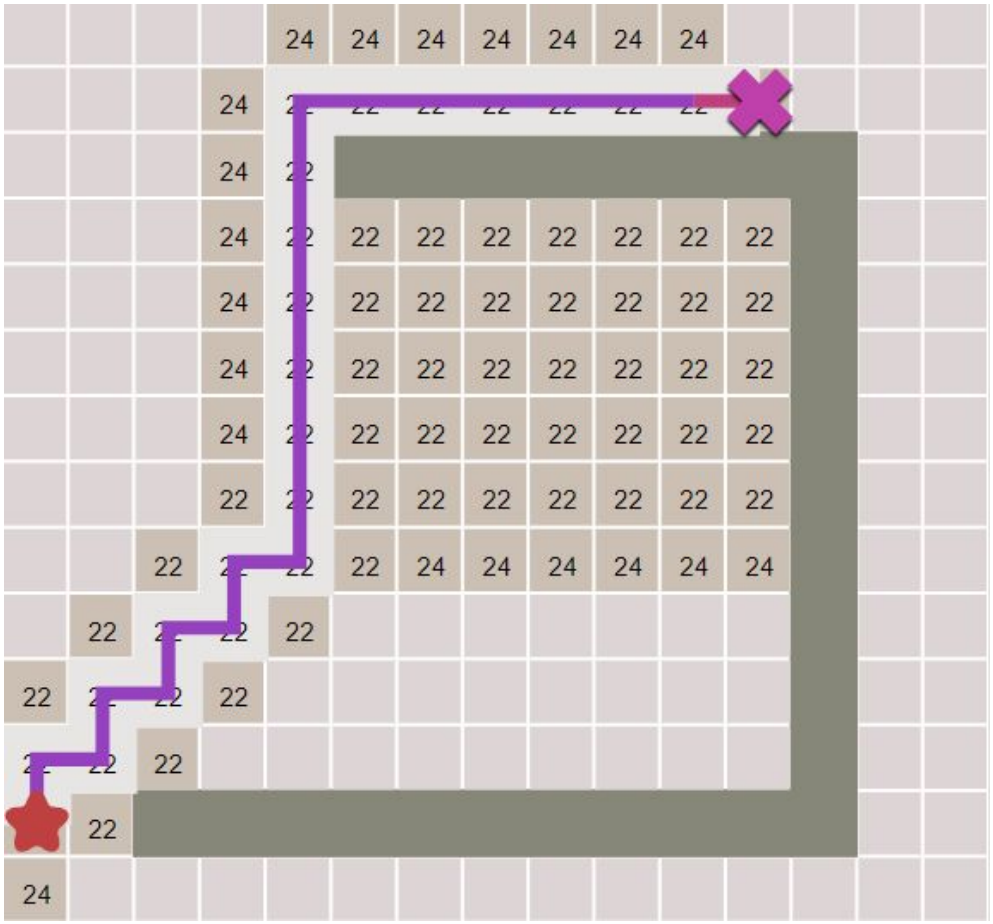


Кратчайший путь не найден

A*: идея

- Исследуем вершины не равномерно, а ориентируясь на расстояние до начала поиска…
- И на расстояние до цели. Т.е. используем эвристику.
- Сочетание алгоритма Дейкстры и поиска первого наилучшего.

A*: демо


24 24 24 24 24 24 24
24
24
24 22 22 22 22 22 22 22
24 22 22 22 22 22 22 22
24 22 22 22 22 22 22 22
24 22 22 22 22 22 22 22
22 22 22 22 22 22 22 22
22 22 22 24 24 24 24 24 24
22 22 22
22 22
22 22
22
24

A*: код

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

A*: код

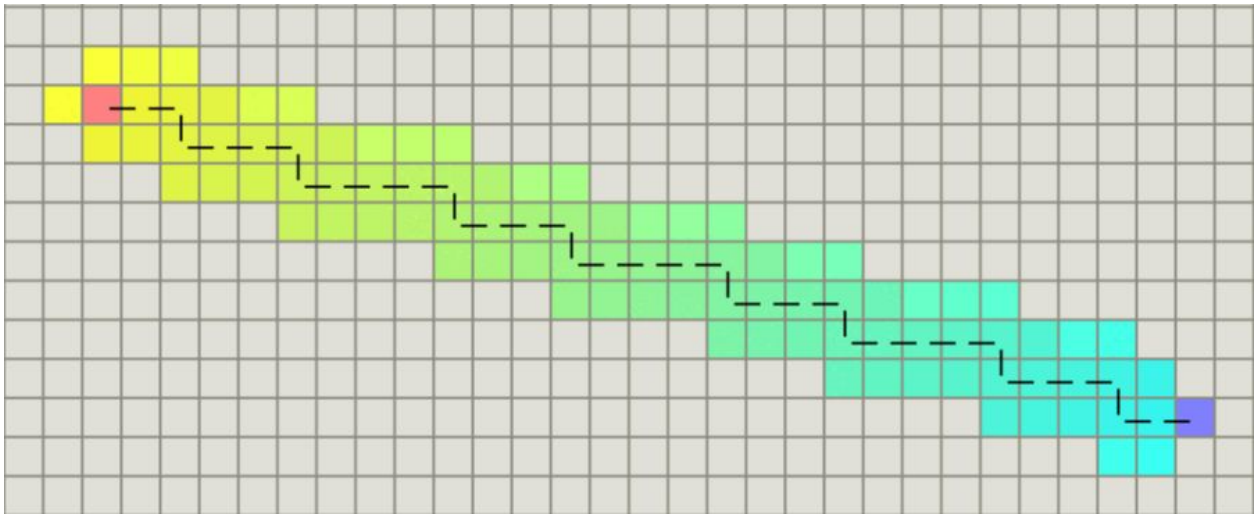
```
while not frontier.empty():
   current = frontier.get()

   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
```

A*: use cases

- Быстро найти кратчайший путь от одной вершины до другой, даже если есть препятствия

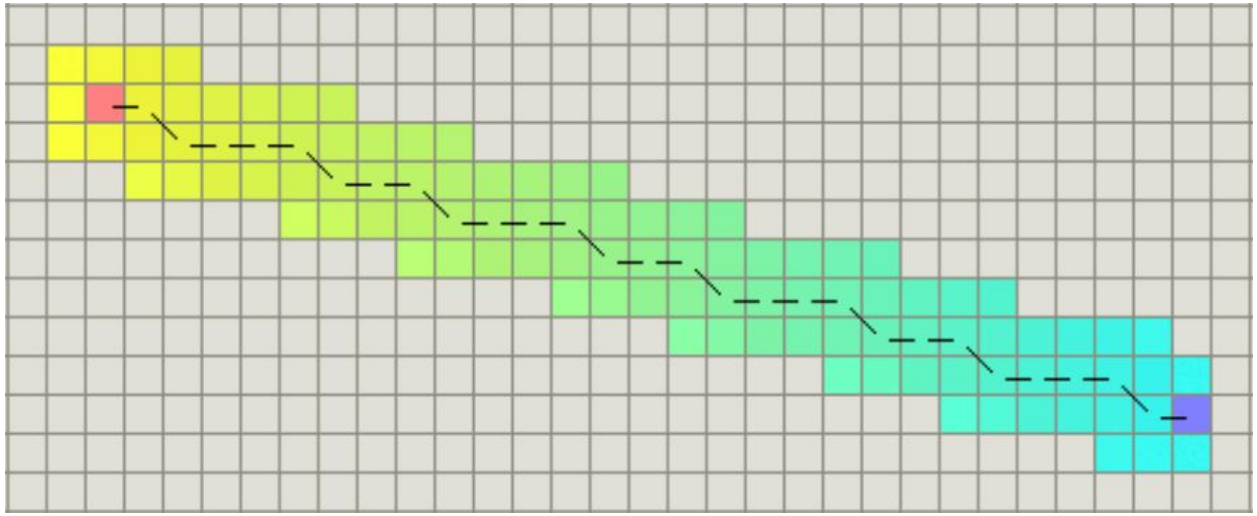
A*: эвристики

- Эвристики бывают разные. От выбора эвристики зависит корректность алгоритма и его быстрота.



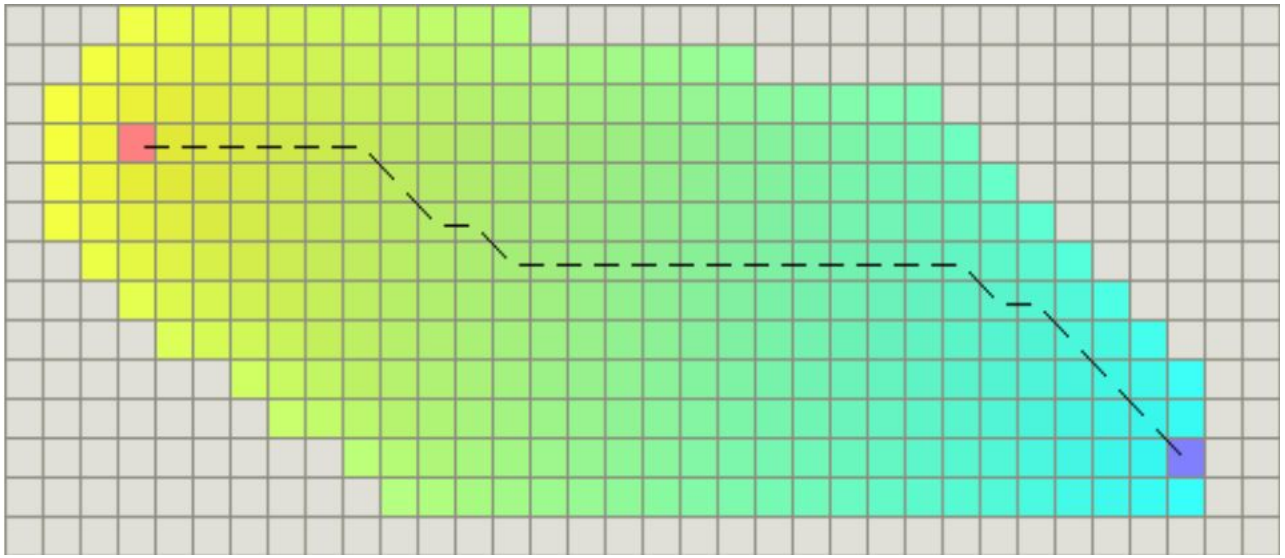
Манхэтонновское
расстояние

A*: эвристики



Диагональное
расстояние

A*: эвристики



Евклидово
расстояние

Спасибо!