Информированные методы поиска на графах

Информированный поиск (эвристический поиск) – стратегия поиска решений в пространстве состояний, в которой используется дополнительная информация (эвристика), о направлении к искомой вершине графа или о минимальном расстоянии до искомой вершины. Информированные методы поиска обычно обеспечивают более эффективный поиск по сравнению с неинформированными методами.

Эвристическая функция на каждом шаге поиска оценивает нижнюю границу расстояния до конечной вершины, чтобы принять решение о том, из какой вершины следует продолжать движение.

Эвристическая функция должна быть *допустимой*. Т.е. являться нижней оценкой расстояния до цели, или даже меньше, чем минимальная оценка.

Эвристическая функция должна быть *монотонно невозрастающей* вдоль любого исследуемого пути. Т.е. при движении к цели каждая следующая вершина должна быть не дальше к целевой вершине, чем предыдущая.

Если $h_1(n)$, $h_2(n)$ – допустимые эвристические функции, и для любого узла n верно неравенство $h_1(n) \geq h_2(n)$, то h_1 является более информированной эвристикой, или доминирует над h_2.

Алгоритм A*

Авторы – Питер Харт, Нильс Нильсон и Бертрам Рафаэль. (1968 год). Алгоритм находит кратчайший путь от начальной вершины a до конечной вершины z. Алгоритм является модификацией алгоритма Дейкстры. Модификация заключается в том, что из фронта волны для раскрытия выбирается не самая близкая к началу вершина (согласно алгоритму Дейкстры), а вершина, лежащая на самом коротком пути к цели. Самый короткий путь оценивается суммой уже пройдённого расстояния $g(v)$ от a до v и оценкой расстояния от v до целевой вершины z с помощью эвристической функции $h(v)$.

Суть алгоритма заключается в вычислении для каждой вершины v двух величин по пути следования. Первая величина – известное расстояние $g(v)$, пройдённое от исходной вершины a до текущей вершины v. Вторая величина – оценка расстояния $h(v)$ от текущей вершины v до конечной вершины z. Эти величины нужны для выбора направления дальнейшего движения по критерию их минимальной суммы.

На каждом шаге алгоритма фронт волны расширяется путём замены фронтальной вершины v, имеющей минимальное значение $f(v) = g(v) + h(v)$, смежными ей вершинами, которые алгоритм ещё не посетил. Заменённая вершина приобретает статус *посещённой*. Алгоритм останавливается, когда целевая вершина z приобретает статус посещённой. При этом она будет содержать расстояние от a до неё $g(z)$.

Постановка задачи: Дан взвешенный ориентированный граф $G(V,E)$, содержащий вершины без дуг отрицательного веса на некоторой

поверхности. Найти кратчайший путь от вершины a графа G до вершины z этого графа.

Дано:

Пусть граф $G(V,E)$ задан списком узлов Node, хранящих имя вершины v и список смежных вершин u с весами рёбер $w_{v,u}$. Обозначим:

$w_{v,u}$ – вес ребра, соединяющего вершины v и u;

a – начальная вершина;

z – конечная вершина;

$g(v)$ – функция пройдённого расстояния от вершины a до вершины v;

$h(v)$ – допустимая монотонная функция оценки расстояния от вершины v до конечной вершины z. Пусть эта функция возвращает расстояние по прямой линии от вершины v до конечной вершины z;

U – список посещённых вершин;

F – очередь вершин фронта волны с приоритетом значений $f(v)$;

p_v – кратчайший путь из a в v в виде списка вершин;

max – максимальное число (целое беззнаковое или вещественное).

Описание алгоритма

1. Снабдим каждую вершину v меткой, содержащей значение $g(v)$ и путь к ней p_v. Исходная вершина a имеет метку 0 и путь $p_v = ()$. Остальные вершины имеют метку бесконечности, которая при программировании может быть представлена максимальным в разрядной сетке числом max.
2. Помещаем исходную вершину a в очередь вершин фронта F со приоритетом $f(v) = 0 + h(a)$.
3. Вырезаем из очереди F первую вершину v, т.е. вершину, имеющую минимальное значение $f(v)$, и помещаем её в список U. Если в список U была помещена конечная вершина z, то останавливаем алгоритм, иначе – переходим к п.4.
4. Устанавливаем меткам вершин $u \in F$, смежных v, минимальное значение $g(u)$:

$g(u) > g(v) + w_{v,u}$ $\quad g(u) = g(v) + w_{v,u}$,

и помещаем их в очередь F с приоритетом $f(u) = g(u) + h(u)$.

Устанавливаем меткам непосещённых вершин $u \notin U$, смежных v, значение $g(u) = g(v) + w_{v,u}$ и помещаем их в очередь F с приоритетом $f(u) = g(u) + h(u)$. Переходим к п.3.

Пример. Найдём минимальный путь из вершины a в вершину e.

Шаг 1. Выполняем п.1 и 2 алгоритма: расставим метки расстояний на исходном графе (рис.1). Они красного цвета. Помещаем исходную вершину в очередь вершин фронта F со значением $f(a) = 19$. Будем обрамлять очередь вершин F круглыми скобками с указанием значения $f(v)$ через дефис. В очереди вершины расположим по мере увеличения значения $f(v)$. Фронт волны – голубой.

F=(a-19);
U=();
g(a)=0; $p_a = (a)$;
g(b)=max; $p_b = ()$;
g(c)=max; $p_c = ()$;
g(d)=max; $p_d = ()$;
g(e)=max; $p_e = ()$;
g(f)=max; $p_f = ()$;

Кроме этого зелёным цветом сразу расставим значения эвристики $f(v)$ для всех вершин. Хотя это действие можно было выполнять по мере прохождения по графу.

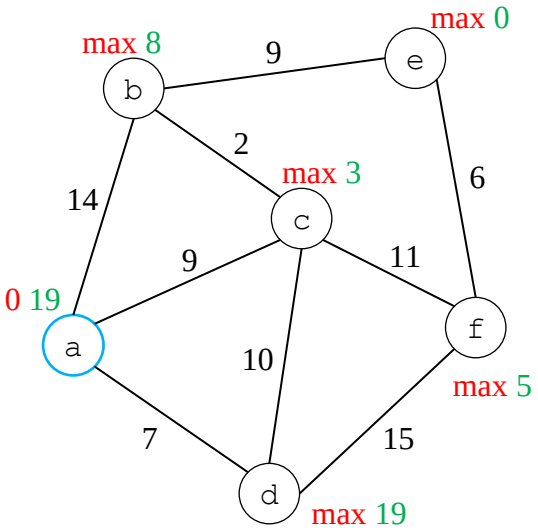


Рис.1. Исходный размеченный граф

Шаг 2. Выполняем п.3 алгоритма – вырезаем из очереди F единственную вершину a и помещаем её в список U. Цвет посещённых вершин – оранжевый (рис.2).

F=();
U=(a);

Выполняем п.4 алгоритма:

g(b)=14; $p_b = (a, b)$;
F=(b-22);
g(c)=9; $p_c = (a, c)$;
F=(c-12, b-22);
g(d)=7; $p_d = (a, d)$;
F=(c-12, b-22, d-26);

Переходим к п.3 алгоритма.

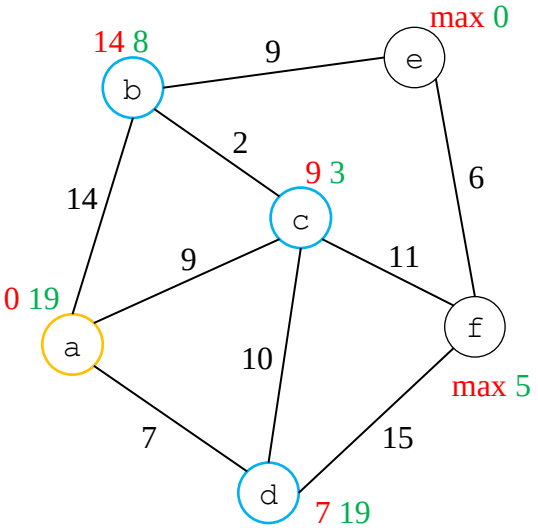


Рис.2.

Шаг 3. Выполняем п.3 алгоритма (рис.3) – вырезаем из очереди F первую вершину, т.е. вершину c, имеющую минимальное значение $f(c) = 12$, и помещаем её в список U:

F=(b-22, d-26);
U=(a, c);

Выполняем п.4 алгоритма:

g(b)=11; $p_b = (a, c, b)$;
F=(b-19, d-26);

Метку вершины d не обновляем, так как текущее значение g(d)=7 меньше, чем путь к этой вершине через вершину c.

Добавляем в очередь F новую вершину f и помечаем её голубым цветом (рис.3):

g(f)=20; $p_f = (a, c, f)$;
F=(b-19, f-25, d-26);

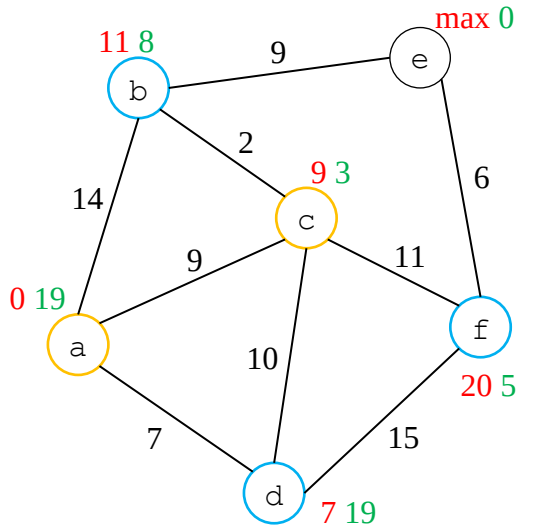


Рис.3.

Переходим к п.3 алгоритма.

Шаг 4. Выполняем п.3 алгоритма (рис.4) – вырезаем из очереди F первую вершину, т.е. вершину b, имеющую минимальное значение $f(b) = 19$, и помещаем её в список U:

```
F=(f-25, d-26);
U=(a, c, b);
```

Выполняем п.4 алгоритма (рис.4):

```
g(e)=20;  p_e = (a,c,b,e);
F=(e-20, f-25, d-26);
```

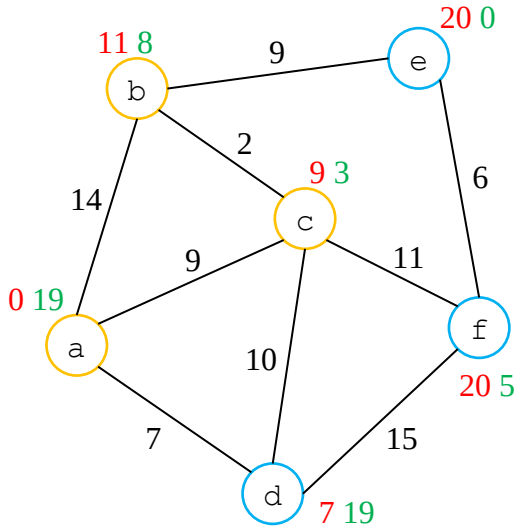


Рис.4.

Мы добрались до искомой вершины e по пути $p_e = (a,c,b,e)$. Скорее всего этот путь самый короткий. Но во фронте могут быть вершины v, у которых `g(v)<f(e)`. Поэтому нам надо либо найти и раскрыть такие вершины, чтобы выяснить какой путь короче, либо лучше всего продолжать алгоритм до тех пор, пока конечная вершина e не будет помещена в список U. Изберём второй путь и вновь перейдём к п.3 алгоритма.

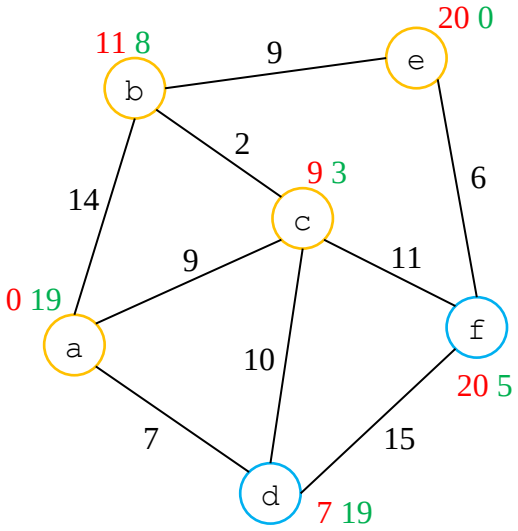


Рис.5.

Шаг 5. Выполняем п.3 алгоритма (рис.5) – вырезаем из очереди F первую вершину, т.е. вершину e, имеющую минимальное значение $f(e) = 20$, и помещаем её в список U:

```
F=(f-25, d-26);
U=(a, c, b, e);
```

Останавливаем алгоритм, так как искомая вершина e была помешена в список посещённых вершин U. Имеем самый короткий путь $p_e = (a,c,b,e)$, и его длину `g(e)=20`.

Примечания к алгоритму A*

1. Пути p_u в каждую вершину u можно не сохранять. Достаточно для каждой вершины u сохранять предыдущую вершину v. Тогда всегда можно восстановить путь, двигаясь от конечной вершины назад – к начальной.

2. Если очередь F имеет несколько перспективных вершин с одинаковой оценкой $f(v)$, то выбор порядка раскрытия таких вершин будет влиять на ход поиска. Если такие вершины раскрывать от самой первой, помещённой в очередь, то поиск будет охватывать возможные препятствия со всех сторон, и только потом устремляться к цели, напоминая по поведению поиск «сначала вширь». Если такие вершины раскрывать от самой последней помещённой в очередь, то поиск будет узконаправленным, т.е. вначале устремляться к цели

и если она не будет достигнута, то совершать обход возможных препятствий, напоминая по поведению поиск «сначала вглубь».

Сравнительный анализ алгоритмов поиска оптимальных путей

Поиск в ширину выполняет обход графа равномерно во всех направлениях, как круги на воде от брошенного камня. Рёбра графа имеют одинаковый вес/расстояние.

Алгоритм Дейкстры вместо равномерного обхода графа во всех возможных направлениях отдаёт предпочтение путям с низкой стоимостью, огибая препятствия. Мы можем задать уменьшенные затраты, чтобы алгоритм двигался по дорогам, повышенную стоимость, чтобы он избегал лесов и врагов, и многое другое. Когда стоимость движения по рёбрам графа может быть разной, мы используем алгоритм Дейкстры вместо поиска в ширину.

Алгоритм A* – это модификация алгоритма Дейкстры, оптимизированная для единственной конечной вершины. Алгоритм A* отдаёт приоритет более коротким путям, ведущим к цели.

В алгоритме Дейкстры используется только расстояние от начала. В жадном поиске используется только оценка расстояния до цели. В алгоритме A* используется сумма этих расстояний.

Поэтому алгоритм Дейкстры хорош в поиске кратчайшего пути(ей), но он тратит время на исследование всех направлений, даже бесперспективных.

Жадный поиск эффективен, исследует перспективные направления, но может не найти кратчайший путь, когда требуется двигаться в сторону от цели (как в лабиринте).

Алгоритм A* использует и подлинное расстояние от начала, и оценённое расстояние до цели. Он взял наилучшие качества у алгоритма Дейкстры и жадного поиска.

Сравните эти алгоритмы поиска минимального пути от клетки с звёздочкой до клетки с крестиком (рис.6).

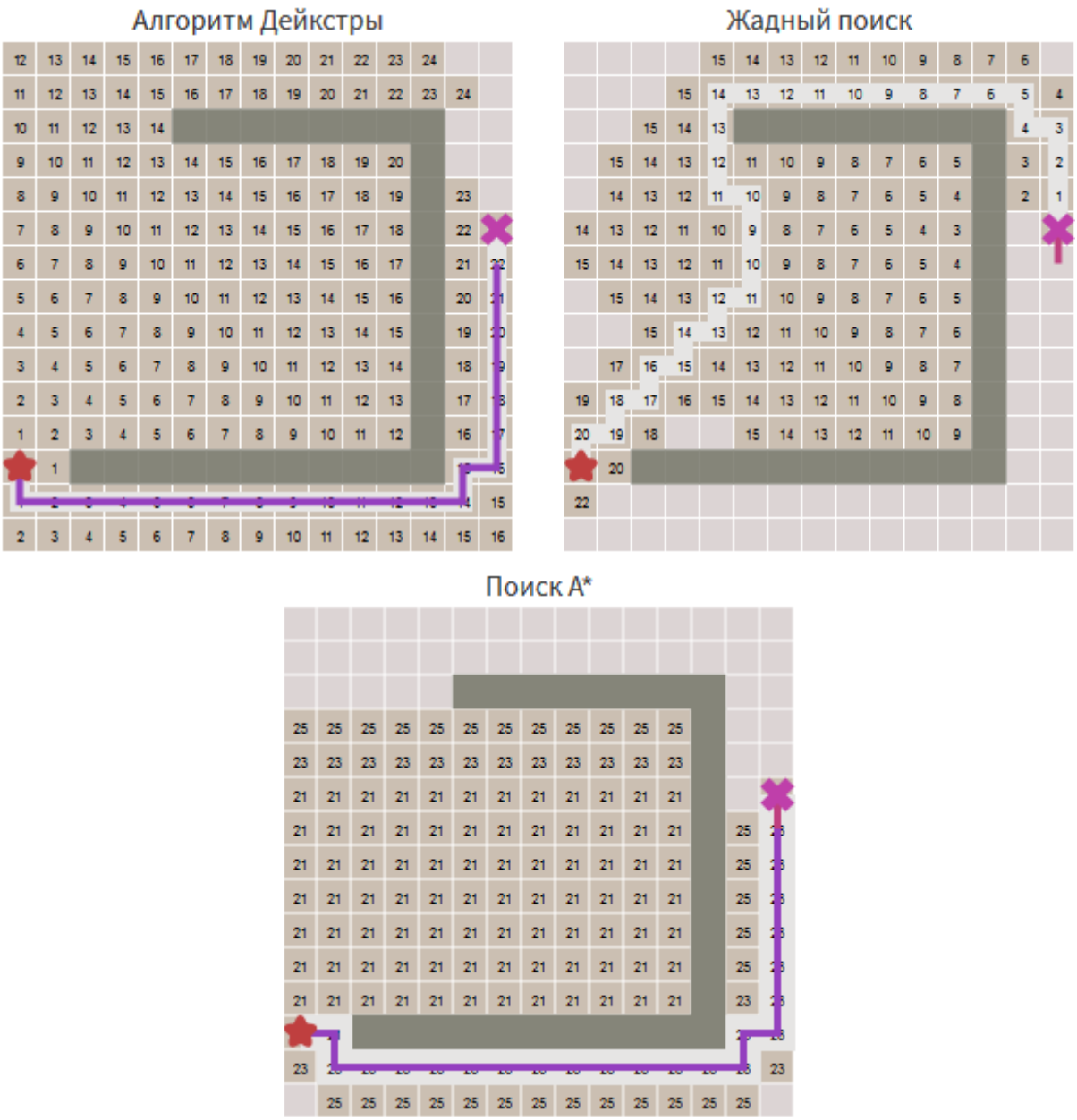


Рис. 6.

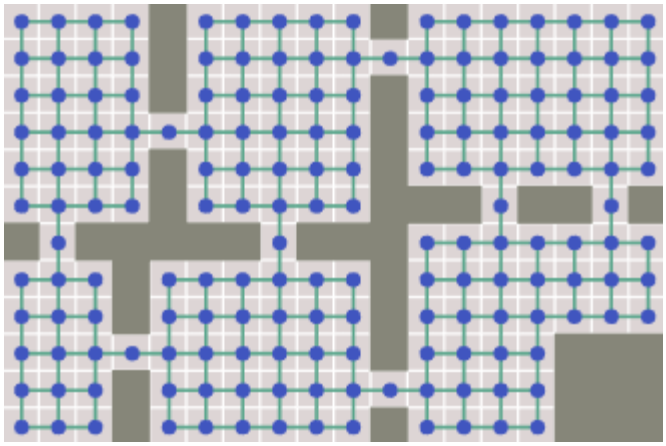
Алгоритм Дейкстры всегда находит оптимальный путь, но тратит на это много ресурсов, так как просматривает все возможные пути. На рисунке мало непосещённых клеток – всего 8.

Жадный поиск быстро находит путь, но не всегда он будет минимальным. Причиной этому являются возможные препятствия, которые надо обходить, двигаясь в сторону от цели. Этот алгоритм эффективен, так как просматривает только перспективные пути. На рисунке много непосещённых клеток – 62.

Алгоритм A* всегда находит оптимальный путь и тратит на это мало ресурсов, так как просматривает сначала перспективные пути. На рисунке много непосещённых клеток – 44, но меньше, чем при жадном поиске.

Рекомендации

1. Для нахождения путей из заданной вершины во все остальные подходит поиск в ширину или алгоритм Дейкстры. Поиск в ширину используется в случае, если стоимость движения по всем рёбрам одинакова. Алгоритм Дейкстры используйте в случае, если стоимость движения по рёбрам различная.
2. Если нужно найти пути к одной вершине, используйте жадный поиск или A*. В большинстве случаев стоит отдать предпочтение A*. Когда есть искушение использовать жадный поиск, то подумайте над применением A* с «недопустимой» эвристикой, т.е. эвристикой, переоценивающей расстояние до цели.
3. Для гарантированного нахождения оптимальных путей стоит использовать поиск в ширину или алгоритм Дейкстры.
4. Для гарантированного нахождения оптимального пути из одной вершину в другую, используйте алгоритм A*. Жадный алгоритм найдёт путь, но не гарантирует, что он оптимальный.
5. Если эвристика алгоритма A* никогда не больше истинного расстояния, то алгоритм A* гарантирует оптимальность решения. Если эвристика уменьшает истинное расстояние до цели, то алгоритм A* в своём поведении смещается в сторону поведения алгоритма Дейкстры, т.е. развивает не самые перспективные пути. Если для всех вершин $h(v)=0$, то A* превращается в алгоритм Дейкстры. Если эвристика преувеличивает истинное расстояние до цели, то алгоритм A* в своём поведении смещается в сторону поведения жадного алгоритма, т.е. раскрывает только те вершины, которые расположены ближе к цели по прямой линии, рассчитывая что препятствий на пути не будет.
6. Если используется сетка, то уменьшение размера графа помогает уменьшить вычислительную сложность поиска (рис.7).



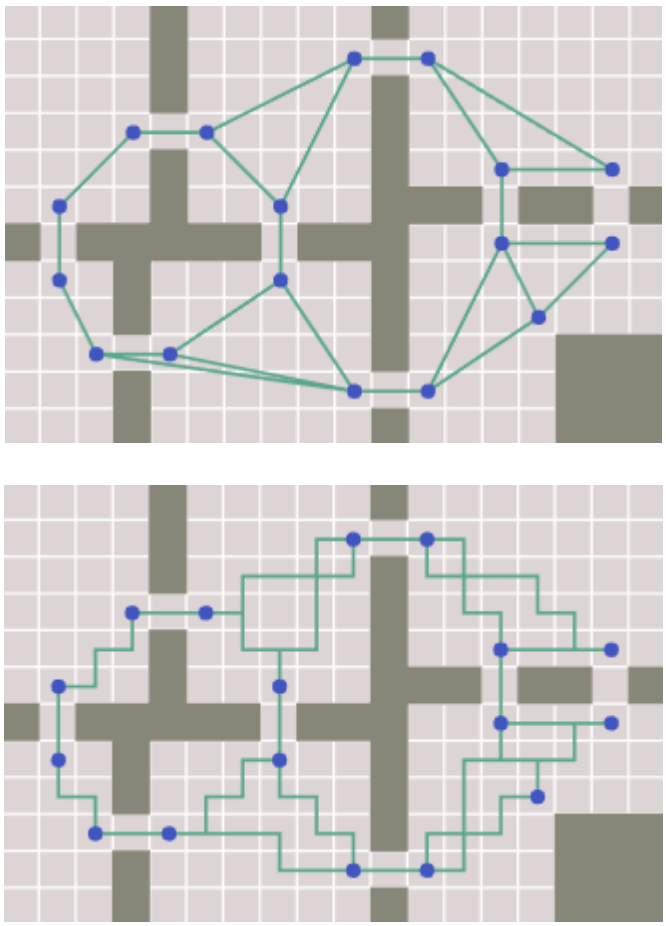


Рис.7

7. Если поиск проводится не на карте, а на абстрактном графе, то поиск допустимой эвристики усложняется.
8. Если $f(v)$ и $h(v)$ измеряются в разных величинах (например, $f(v)$ – это расстояние в километрах, а $h(v)$ – оценка времени пути в часах), то A* может выдать некорректный результат.
9. Поведение алгоритма сильно зависит от того, какая эвристика используется. В свою очередь, выбор эвристики зависит от постановки задачи. Часто A* используется для моделирования перемещения по поверхности, покрытой координатной сеткой. Возможные правила построения эвристик для плоской карты:

- Соседние ячейки принято классифицировать двояко: в смысле окрестности Мура и окрестности фон Неймана, отличающихся тем, что в окрестности фон Неймана соседними ячейками считаются только 4 ячейки, в окрестности Мура – все 8 ячеек. Возможны 6 соседних ячеек, когда поле гексагональное.
- Расстояние Германа Минковского или манхэттенское расстояние – минимальное количество ходов ладьи на шахматной доске (окрестность фон Неймана), количество кварталов, пройдённых в городе только по дорогам:

$h(v)=|v.x-goal.x|+|v.y-goal.y|$.

- Расстояние Пафнутия Чебышёва – минимальное количество ходов короля на шахматной доске (окрестность Мура). Ход по диагонали равен ходу по вертикали или горизонтали:

$h(v)=\max(|v.x-goal.x|,|v.y-goal.y|)$.

- Расстояние Евклида (перемещения не ограничены сеткой):
$h(v) = \sqrt{(v.x - goal.x)^2 + (v.y - goal.y)^2}$.
10. В игре пятнашки эвристикой может служить либо количество костяшек, лежащих не на своём месте, либо сумма расстояний Минковского всех костяшек от своих мест.

ЛР:
1. Разработать программу поиска пути на графе на основе алгоритма A*. Использовать расстояние Минковского.
2. Разработать программу поиска пути на графе на основе алгоритма A*. Использовать расстояние Чебышёва.