ВикипедиЯ

A*

Материал из Википедии — свободной энциклопедии

О радиоисточнике в центре галактики Млечный Путь см. Стрелец A.*

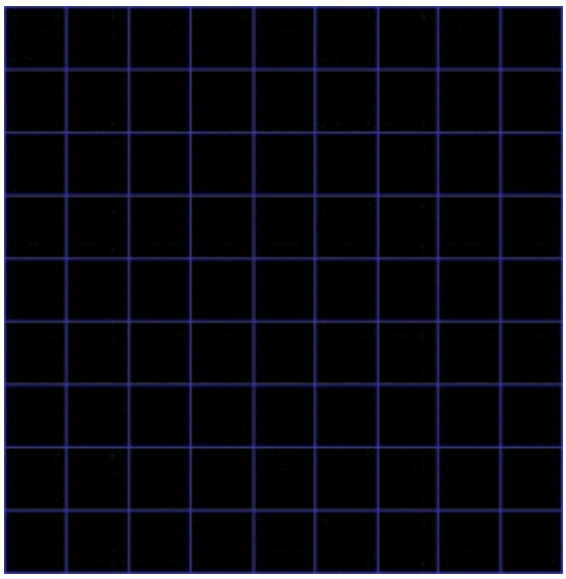
Поиск A* (произносится «А звезда» или «А стар», от англ. *A star*) — в информатике и математике, алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной).

Порядок обхода вершин определяется **эвристической функцией** «расстояние + стоимость» (обычно обозначаемой как $f(x)$). Эта функция — сумма двух других: функции стоимости достижения рассматриваемой вершины (x) из начальной (обычно обозначается как $g(x)$ и может быть как эвристической, так и нет), и функции эвристической оценки расстояния от рассматриваемой вершины к конечной (обозначается как $h(x)$).

Функция $h(x)$ должна быть **допустимой эвристической оценкой**, то есть не должна переоценивать расстояния к целевой вершине. Например, для задачи маршрутизации $h(x)$ может представлять собой расстояние до цели по прямой линии, так как это физически наименьшее возможное расстояние между двумя точками.

Этот алгоритм был впервые описан в 1968 году Питером Хартом, Нильсом Нильсоном и Бертрамом Рафаэлем. Это по сути было расширение алгоритма Дейкстры, созданного в 1959 году. Новый алгоритм достигал более высокой производительности (по времени) с помощью эвристики. В их работе он упоминается как «алгоритм A». Но так как он вычисляет лучший маршрут для заданной эвристики, он был назван A*.

Обобщением для него является двунаправленный эвристический алгоритм поиска.



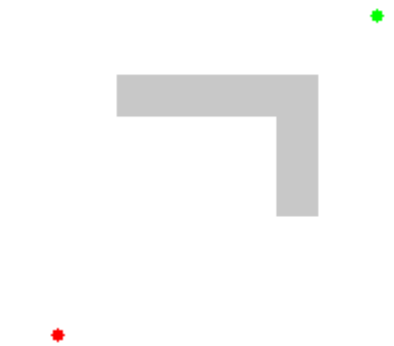
Содержание

История

В 1964 году Нильс Нильсон изобрёл эвристический подход к увеличению скорости алгоритма Дейкстры. Этот алгоритм был назван *A1*. В 1967 году Бертрам Рафаэль сделал значительные улучшения по этому алгоритму, но ему не удалось достичь оптимальности. Он назвал этот алгоритм *A2*. Тогда в 1968 году Питер Э. Харт представил аргументы, которые доказывали, что *A2* был оптимальным при использовании последовательной эвристики лишь с незначительными изменениями. В его доказательство алгоритма также включён раздел, который показывал, что новый алгоритм *A2* был возможно лучшим алгоритмом, учитывая условия. Таким образом, он обозначил новый алгоритм в синтаксисе звёздочкой, он начинается на *A* и включал в себя все возможные номера версий.

Описание алгоритма

A* пошагово просматривает все пути, ведущие от начальной вершины в конечную, пока не найдёт минимальный. Как и все информированные алгоритмы поиска, он просматривает сначала те маршруты, которые «кажутся» ведущими к цели. От жадного алгоритма, который тоже является алгоритмом поиска по первому лучшему совпадению, его отличает то, что при выборе вершины он учитывает, помимо прочего, *весь* пройденный до неё путь. Составляющая *g(x)* — это стоимость пути от начальной вершины, а не от предыдущей, как в жадном алгоритме.



Пустые кружки в узлах принадлежат *открытому списку*, красные/зелёные относятся к *закрытому списку*.

В начале работы просматриваются узлы, смежные с начальным; выбирается тот из них, который имеет минимальное значение *f(x)*, после чего этот узел раскрывается. На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа — множеством частных решений, — которое размещается в очереди с приоритетом. Приоритет пути определяется по значению $f(x) = g(x) + h(x)$. Алгоритм продолжает свою работу до тех пор, пока значение *f(x)* целевой вершины не окажется меньшим, чем любое значение в очереди, либо пока всё дерево не будет просмотрено. Из множества решений выбирается решение с наименьшей стоимостью.

Чем меньше эвристика *h(x)*, тем больше приоритет, поэтому для реализации очереди можно использовать сортирующие деревья.

Множество просмотренных вершин хранится в `closed`, а требующие рассмотрения пути — в очереди с приоритетом `open`. Приоритет пути вычисляется с помощью функции *f(x)* внутри реализации очереди с приоритетом.

Псевдокод:

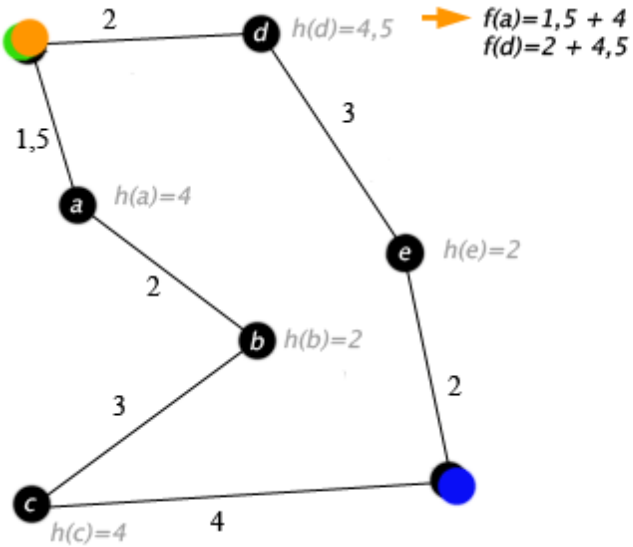
```
function A*(start, goal, f)
    % множество уже пройденных вершин
    var closed := the empty set
    % множество частных решений
    var open := make_queue(f)
    enqueue(open, path(start))
    while open is not empty
```

```
        var p := remove_first(open)
        var x := the last node of p
        if x in closed
            continue
        if x = goal
            return p
        add(closed, x)
        % добавляем смежные вершины
        foreach y in successors(x)
            enqueue(open, add_to_path(p, y))
    return failure
```

Множество уже пройденных вершин можно опустить (при этом алгоритм превратится в поиск по дереву решений), если известно, что решение существует, либо если `successors` умеет отсекать циклы.

Пример

Примером алгоритма A* в действии, где узлы — это города, связанные дорогами и H (x) является самым коротким расстоянием до целевой точки:



Ключ: зелёный — начало, синий — цель, оранжевый — посещённые узлы.

Свойства

Как и алгоритм поиска в ширину, A* является **полным** в том смысле, что он всегда находит решение, если таковое существует.

Если эвристическая функция *h* допустима, то есть никогда *не переоценивает* действительную минимальную стоимость достижения цели, то A* сам является допустимым (или **оптимальным**), также при условии, что мы не отсекаем пройденные вершины. Если же мы это делаем, то для оптимальности алгоритма требуется, чтобы *h(x)* была ещё и **монотонной**, или **преемственной** эвристикой. Свойство монотонности означает, что если существуют пути **A—B—C** и **A—C** (не обязательно через **B**), то оценка стоимости пути от **A** до **C** должна быть меньше либо равна сумме оценок путей **A—B** и **B—C**. (Монотонность также известна как неравенство треугольника: одна сторона треугольника не может быть длиннее, чем сумма двух других сторон.) Математически, для всех путей *x*, *y* (где *y* — потомок *x*) выполняется:

$$g(x) + h(x) \leq g(y) + h(y).$$

A* также **оптимально эффективен** для заданной эвристики *h*. Это значит, что любой другой алгоритм исследует *не меньше* узлов, чем A* (за исключением случаев, когда существует несколько частных решений с одинаковой эвристикой, точно соответствующей стоимости оптимального пути).

В то время как A* оптимален для «случайно» заданных графов, нет гарантии, что он сделает свою работу лучше, чем более простые, но и более информированные относительно проблемной области алгоритмы. Например, в неком лабиринте может потребоваться сначала идти по направлению *от* выхода, и только потом повернуть назад. В этом случае обследование вначале тех вершин, которые расположены ближе к выходу (по прямой дистанции), будет потерей времени.

Особые случаи

В общем говоря, поиск в глубину и поиск в ширину являются двумя частными случаями алгоритма A*. Для поиска в глубину возьмём глобальную переменную-счётчик **C**, инициализировав её неким большим значением. Всякий раз при раскрытии вершины будем присваивать значение счётчика смежным вершинам, уменьшая его на единицу после каждого присваивания. Таким образом, чем раньше будет открыта вершина, тем большее значение *h(x)* она получит, а значит, будет просмотрена в последнюю очередь. Если положить *h(x) = 0* для всех вершин, то мы получим ещё один специальный случай — алгоритм Дейкстры.

Тонкости реализации

Существует несколько особенностей реализации и приёмов, которые могут значительно повлиять на эффективность алгоритма. Первое, на что не мешает обратить внимание — это то, как очередь с приоритетом обрабатывает связи между вершинами. Если вершины добавляются в неё так, что очередь работает по принципу LIFO, то в случае вершин с одинаковой оценкой A* «пойдёт» в глубину. Если же при добавлении вершин реализуется принцип FIFO, то для вершин с одинаковой оценкой алгоритм, напротив, будет реализовывать поиск в ширину. В некоторых случаях это обстоятельство может оказывать существенное значение на производительность.

В случае, если по окончании работы от алгоритма требуется маршрут, вместе с каждой вершиной обычно хранят ссылку на родительский узел. Эти ссылки позволяют реконструировать оптимальный маршрут. Если так, тогда важно, чтобы одна и та же вершина не встречалась в очереди дважды (имея при этом свой маршрут и свою оценку стоимости). Обычно для решения этой проблемы при добавлении вершины проверяют, нет ли записи о ней в очереди. Если она есть, то запись обновляют так, чтобы она соответствовала минимальной стоимости из двух. Для поиска вершины в сортирующем дереве многие стандартные алгоритмы требуют времени O(n). Если усовершенствовать дерево с помощью хеш-таблицы, то можно уменьшить это время.

Почему A* допустим и оптимален

Алгоритм A* и допустим, и обходит при этом минимальное количество вершин, благодаря тому, что он работает с *«оптимистичной»* оценкой пути через вершину. Оптимистичной в том смысле, что, если он пойдёт через эту вершину, у алгоритма «есть шанс», что реальная стоимость результата будет равна этой оценке, но никак не меньше. Но, поскольку A* является информированным алгоритмом, такое равенство может быть вполне возможным.

Когда A* завершает поиск, он, согласно определению, нашёл путь, истинная стоимость которого меньше, чем *оценка* стоимости любого пути через любой открытый узел. Но поскольку эти оценки являются оптимистичными, соответствующие узлы можно без сомнений отбросить. Иначе говоря, A* никогда не упустит возможности минимизировать длину пути, и потому является допустимым.

Предположим теперь, что некий алгоритм **B** вернул в качестве результата путь, длина которого больше оценки стоимости пути через некоторую вершину. На основании эвристической информации, для алгоритма **B** нельзя исключить возможность, что этот путь имел и меньшую реальную длину, чем результат. Соответственно, пока алгоритм **B** просмотрел меньше вершин, чем A*, он не будет допустимым. Итак, A* проходит наименьшее количество вершин графа среди допустимых алгоритмов, использующих такую же точную (или менее точную) эвристику.

Оценка сложности

Временна́я сложность алгоритма A* зависит от эвристики. В худшем случае, число вершин, исследуемых алгоритмом, растёт экспоненциально по сравнению с длиной оптимального пути, но сложность становится полиномиальной, когда эвристика удовлетворяет следующему условию:

$$|h(x) - h^*(x)| \leq O(\log h^*(x));$$

где h* — оптимальная эвристика, то есть точная оценка расстояния из вершины x к цели. Другими словами, ошибка *h(x)* не должна расти быстрее, чем логарифм от оптимальной эвристики.

Но ещё бо́льшую проблему, чем временна́я сложность, представляют собой потребляемые алгоритмом ресурсы памяти. В худшем случае ему приходится помнить экспоненциальное количество узлов. Для борьбы с этим было предложено несколько вариаций алгоритма, таких как алгоритм A* с итеративным углублением (iterative deeping A*, IDA*), A* с ограничением памяти (memory-bounded A*, MA*), упрощённый MA* (simplified MA*, SMA*) и рекурсивный поиск по первому наилучшему совпадению (recursive best-first search, RBFS).

Литература

- *Лорьер Ж.-Л.* Системы искусственного интеллекта / Пер. с фр. и ред. В. Л. Стефанюка. — М.: Мир, 1991. — С. 238—244. — 20 000 экз, экз. — ISBN 5-03-001408-X.
- *Рассел С. Дж., Норвиг, П.* Искусственный интеллект: современный подход = Artificial Intelligence: A Modern Approach / Пер. с англ. и ред. К. А. Птицына. — 2-е изд.. — М.: Вильямс, 2006. — С. 157—162. — 3 000 экз, экз. — ISBN 5-8459-0887-6.
- *Нильсон Н.* Искусственный интеллект: методы поиска решений = Problem-solving Methods in Artificial Intelligence / Пер. с англ. В. Л. Стефанюка; под ред. С. В. Фомина. — М.: Мир, 1973. — С. 70 — 80.
- *Dechter, R., Pearl, J.* Generalized best-first search strategies and the optimality of A* (http://portal.acm.org/citation.cfm?id=3830&coll=portal&dl=ACM) // Journal of the ACM. — 1985. — Т. 32, № 3. — С. 505 — 536.
- *Hart P. E., Nilsson, N. J., Raphael, B.* A Formal Basis for the Heuristic Determination of Minimum Cost Paths // IEEE Transactions on Systems Science and Cybernetics SSC4. — 1968. — № 2. — С. 100 — 107.
- *Hart P. E., Nilsson, N. J., Raphael, B.* Correction to «A Formal Basis for the Heuristic Determination of Minimum Cost Paths» // SIGART Newsletter. — 1972. — Т. 37. — С. 28 — 29.
- *Pearl J.* Heuristics: Intelligent Search Strategies for Computer Problem Solving. — Addison-

Wesley, 1984. — ISBN 0-201-05594-5.

Ссылки

- *Патрик Лестер.* Алгоритм A* для новичков (http://www.policyalmanac.org/games/aStarTutorial_rus.htm) (недоступная ссылка) (26 июля 2004). Дата обращения 12 декабря 2015. Архивировано (https://www.webcitation.org/66fdfaHVe?url=http://www.policyalmanac.org/games/aStarTutorial_rus.htm) 4 апреля 2012 года.
- A* построение пути со сглаживанием и реалистичными поворотами (http://www.gamasutra.com/view/feature/3096/toward_more_realistic_pathfinding.php) — Автор Marco Pinter
- Amit’s Pages (http://theory.stanford.edu/~amitp/GameProgramming/) — О поиске пути и A*
- *Cupper.* Принцип работы алгоритма поиска пути Астар (A*) (http://www.gamedev.ru/articles/?id=70121). GameDev.ru (17 января 2004 года). Дата обращения 22 июля 2009.
