Конспект лекции по A*

Постановка задачи — ищем кратчайшие пути на реальной карте. Например, движение юнита в компьютерной игре (по сетке), маршрут автомобиля по карте дорог. Рассматриваются графы с неотрицательными весами рёбер.

Можно использовать алгоритмы BFS или Дейкстры, но они не используют информацию о том, какие перемещения скорее приведут нас к цели. Лучше не просматривать маршруты, ведущие в противоположном направлении от цели. BFS/Дейстра не учитывают это, они ищут кратчайшие маршруты во всех возможных направлениях, в результате просматривают слишком много вершин графа.

Давайте искать путь, пытаясь на каждом шаге выбирать наилучшее ребро (которое ведёт нас ближе к цели). Получим жадный алгоритм Best-First-Search, он не находит кратчайший путь (пример с вертикальной стенкой).

В этом алгоритме мы выбираем вершину, которая нам кажется наилучшей. Давайте введём оценочную функцию h(x) — оценка расстояния от вершины x до конечной вершины. Также отметим, что наш алгоритм ищет путь в конкретную вершину, а не во все остальные вершины.

Пусть g(x) — расстояние от начальной вершины до вершины x (на самом деле, не кратчайшее, а какое-то построенное). Эти величины релаксируются стандартным образом: g(v) = min(g(v), g(u) + wt(u, v)) при рассмотрении ребра u → v.

Тогда мы можем оценить длину пути, проходящего через вершину x, как g(x) + h(x). Будем считать это перспективной оценкой пути и будем рассматривать вершины в порядке неубывания эвристической функции g(x) + h(x). Получаем следующий алгоритм, напоминающий Дейкстру и DFS.

```
bool A*(start, goal):
    U = ∅ // Множество рассмотренных вершин
    Q = ∅ // Множество открытых вершин — очередь с приоритетом
    Q.push(start)
    g[start] = 0
    f[start] = g[start] + h(start)
    while Q.size() != 0
        current = Q.top() // Вершина с минимальным значением f
        if current == goal
            return true // нашли путь до нужной вершины
        Q.remove(current)
        U.push(current)
        for v : смежные с current вершины
            v_score = g[current] + wt(current, v) // wt(current, v) — вес ребра
            if v_score < g[v]
                parent[v] = current
                g[v] = v_score
                f[v] = g[v] + h(v)
                if v ∉ Q
                  Q.push(v)
    return false
```

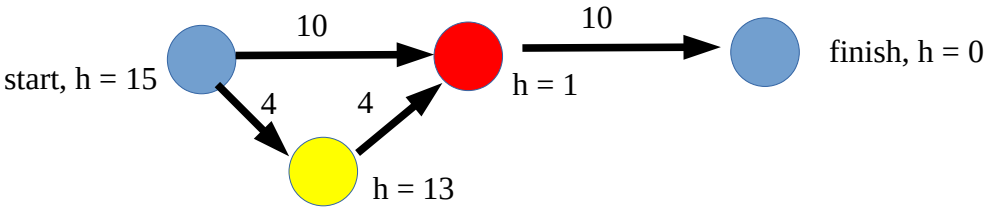
Требование к оценочной функции — допустимость. Оценочная функция не должна **завышать** расстояние до цели, она может только занижать расстояние. То есть h(x) ≤ h'(x), где h'(x) — правильное расстояние от данной вершины до конечной вершины.

Примеры оценочной функции: евклидово расстояние, манхэттенское расстояние, расстояние Чебышёва, *метрика коня*.

Утверждение. Алгоритм A* находит правильный результат, если оценочная функция допустима.

Интуитивное соображение — не может быть пути короче, т. к. если такой путь проходит через вершину x, то его длина не меньше, чем f(x) = g(x) + h(x), а мы уже рассмотрели все пути с меньшим значением f(x) = g(x) + h(x). Проблема в рассуждении в том, что значение g(x) не является кратчайшим расстоянием до вершины x. Исправим доказательство, применим индукцию аналогично алгоритму Дейкстры. Пусть мы нашли какой-то путь до какой-то вершины длины g(finish), тогда f(finish) = g(finish). Пусть есть какой-то кратчайший путь u_0=start, u_1, u_2, …, u_k=finish. Вершина u_0 — рассмотрена (закрытая), значит, ребро $u_0 \to u_1$ отрелаксировано и $g(u_1)=d(u_1)$ (значение $g(u_1)$ есть настоящее кратчайшее растояние до u_1). Тогда $f(u_1)=g(u_1)+h(u_1) \le d(u_1) + h'(u_1)$ = длина пути от start до finish < f(finish). Тогда вершина finish не могла быть извлечена из очереди, если не была извлечена вершина u_1, значит, u_1 была закрыта. Далее по индукции для всех вершин на пути доказывается, что они были обработаны, и тогда расстояние до вершины finish было найдено и оно меньше

Поскольку d(x) не является реальным кратчайшим расстоянием даже для закрытых вершин, оно может обновляться в том числе для закрытых вершин. Приведём пример такого графа.



В этом графе сначала будет покрашена красная вершина со значением (g=10) + (h=1) = 11, потом будет добавлена жёлтая вершина со значением (g = 4) + (h=13) = 17, и после этого уменьшится значение g для красной вершины, она станет g = 8, после чего красная вершина срелаксируется повторно.

У этого графа странная оценочная функция, она немонотонная (не удовлетворяет неравенству треугольника). Дадим определение монотонной функции: h(x) — монотонна, если для ребра $u \to v$: $h(u) \le h(v) + wt(u, v)$, а h(goal) = 0.

Утверждение: монотонная функция является допустимой. Доказательство индукцией по количеству рёбер в кратчайшем пути от данной вершины до конечной вершины.

Утверждение — при движении по любому пути от начальной вершины значения f(x) = g(x) + h(x) неубывают. Здесь f(x) — наша оценка сколько уже прошли + сколько осталось.

Доказательство. Пусть мы пришли в вершину u, и затем прошли по ребру $u \to v$.

f(u) = g(u) + h(u)
f(v) = g(v) + h(v)
g(v) = g(u) + wt(u, v) (как мы пересчитали функцию, пройдя по ребру)
$h(u) \le wt(u, v) + h(v)$ (условие монотонности).

$f(v) = g(u) + wt(u, v) + h(v) \ge g(u) + h(u) = f(u)$

Следствие — если функция монотонная, то каждая вершина будет обработана не более одного раза (т. е. для обработанной вершины значение g(x) и f(x) не может быть уменьшено). Следует из монотонности функции при релаксации ребра. Если мы обработали какую-то вершину, то для всех остальных вершин значение f(x) не может быть меньше.

Связь A* с другими алгоритмами. Дейкстра (и BFS) — это A*, в котором h(x) = 0. То есть h(x) не даёт никакой информации и все соседние вершины одинаково ценны. Также можно получить DFS из A*. Пусть C — большая константа, положим для каждой открытой вершины значение h(x) = C, и будем уменьшать C на 1. Чем позже открыта вершина, тем меньше для неё эвристическая оценка. Эта функция недопустима, поэтому кратчайший путь не находится (а DFS и не ищет кратчайший путь).

A* - оптимальный алгоритм в смысле рассмотренных вершин. Если есть другой алгоритм с аналогичной оценочной функцией, то он обязан рассмотреть все те же вершины, что и алгоритм A*. Если он пропустил какую-то вершину, то, возможно, от пропустил и какой-то короткий путь.

Тонкости реализации — в каком порядке рассматриваются вершины с равным значением f? Если рассматривать в порядке FIFO, то это будет движение в ширину, если в порядке LIFO — то в глубину. Я советую LIFO — при равном значении f выбираем те вершины, у которых минимально значение h, в этом случае мы двигаемся далее по найденному пути, вместо того, чтобы открывать другой путь такой же длины. Пример — обход препятствия.

Алгоритм всегда заканчивает работу на конечном графе. На бесконечном графе алгоритм всегда заканчивает работу (находит путь), если веса всех рёбер не меньше заданного положительного ε и путь существует.