算法竞赛专题解析--

搜索进阶(4)--A*搜索

罗勇军 2020.3.17

本系列是这本书的扩展资料：《算法竞赛入门到进阶》（京东，当当） 清华大学出版社
本文 web 地址（同步）：https://blog.csdn.net/weixin_43914593
https://www.cnblogs.com/luoyj/
PDF 下载地址：https://github.com/luoyongjun999/code 其中的补充资料
如有建议，请联系：（1）QQ 群，567554289；（2）作者 QQ，15512356

《算法竞赛入门到进阶》的第 4 章“搜索技术”，讲解了递归、BFS、DFS 的原理，以及双向广搜、A*算法、剪枝、迭代加深搜索、IDA*的经典例题，适合入门搜索算法。

本文分几篇专题介绍搜索扩展内容、讲解更多习题，便于读者深入掌握搜索技术。

第 1 篇：搜索基础。

第 2 篇：剪枝。

第 3 篇：广搜进阶。

第 4 篇：A*搜索。

本文是第 4 篇。

1.1 A*搜索算法

A*搜索算法（A* Search Algorithm）可以高效率解决一类最短路径问题：给定一个确定起点、一个确定终点（或者可以预测的终点），求起点到终点的最短路径。

A*算法的核心是一个估价函数 $f = g + h$。它的效率取决于函数 h 的设计。

最短路问题的算法很多，例如双向广搜的效率也较高，而 A*算法比双向广搜效率更高。另外，从本文的例题（K 短路等）可以看出，A*算法可以解决更复杂的问题。

1.1.1 最短路径算法

有多种应用场景的最短路径问题，它们对应了不同的算法。下表总结了一些经典算法，除了贪心最优搜索之外，其他的**都是最优性算法**，即得到的解是最短路径。

问题	边权	算法	时间复杂度
一个起点，一个终点	非负数；无边权（或边权为 1）	A*	$< O((E+V)logV)$
		双向广搜	$< O((E+V)logV)$
		贪心最优搜索	$< O(E+V)$
一个起点到其他所有点	无边权（或边权为 1）	BFS	$O(E+V)$
	非负数	Dijkstra(堆优化优先队列)	$O((E+V)logV)$
	允许有负数	SPFA	$< O(EV)$
所有点对之间	允许有负数	Floyd-Warshall	$O(V^3)$

1.1.2 A*搜索算法详解

A*算法的技术可以概况为：A*算法 = 贪心最优搜索 + BFS + 优先队列。

在图问题中，“Dijkstra + 优先队列”就是“BFS + 优先队列”，此时也可以概况为：“A*算法 = 贪心最优搜索 + Dijkstra + 优先队列”。

下面以图的最短路径问题为例，推理出 A*算法的原理。

注意，除了图这种应用场合，A*算法还能在更多场合下得到应用。

1.贪心最优搜索

贪心最优搜索（Greedy Best First Search[①]）是一种启发式搜索，效率很高，但是得到的解不一定是最优的。

算法的基本思路就是贪心：从起点出发，在它的邻居结点中选择下一个结点时，挑那个到终点最近的结点。当然，实际上不可能提前知道结点到终点的距离，更不用说挑选出最近的邻居点了。所以，只能采用估计的方法，例如在网格图中，根据曼哈顿距离来估算邻居结点到终点的距离。

如何编程？仍然用“BFS + 优先队列”，不过，进入优先队列的，不是从起点 s 到当前点 k 的距离，而是从当前点 k 到终点 t 的距离。

很明显，贪心最优搜索避开了大量结点，只挑那些“好”结点，速度极快，但是显然得到的路径不一定最优。

在无障碍的网格图中，贪心最优搜索算法的结果是最优解。因为用于估算的曼哈顿距离就是实际存在的最短路，所以每次找到的下一个结点，显然是最优的。

在有障碍的网格图中，根据曼哈顿距离选下一跳结点，路线会一直走到碰壁，然后再绕路，最后得到的不一定是最短路径。

贪心搜索的算法思想是：“**只看终点，不管起点**”。走一步看一步，不回头重新选择，走错了也不改正。而且，用曼哈顿距离这种简单的估算，也不能提前绕开障碍。

2. Dijkstra（BFS）

用优先队列实现的 Dijkstra（BFS）[②]，能比较高效地求得一个起点到所有其他点的最短路径。Dijkstra 算法有 BFS 的通病：下一步的搜索是盲目的，没有方向感。即使给定了终点，Dijkstra 也需把几乎所有的点和边放进优先队列进行处理，直到终点从优先队列弹出为止。所以它适合用来求一个起点到所有其他结点的最优路径，而不是只求到一个终点的路径。

Dijkstra 的算法思想是：“**只看起点，不管终点**”。等遍历得差不多了，总会碰到终点的。

3. A*算法的原理和复杂度

A*算法是贪心最优搜索和 Dijkstra 的结合，“**既看起点，又看终点**”。它比 Dijkstra 快，因为它不像 Dijkstra 一样盲目；它不仅有贪心搜索的预测能力，而且能得到最优解。

它是如何结合这两个算法的？

设起点是 s，终点是 t，算法走到当前位置 i 点，把 s-t 的路径分为两部分：s-i-t。

（1）s-i 的路径，由 Dijkstra 保证最优性；

（2）i-t 的路径，由贪心搜索进行预测，选择 i 的下一个结点；

（3）当走到 i 碰壁时，i 将被丢弃，并回退到上一层重新选择新的点 j，j 仍由 Dijkstra 保证最优性。

以上思路可以用一个估价函数来具体操作：

$$f(i) = g(i) + h(i)$$

f(i)是对 i 点的评估，g(i)是从 s 到 k 的代价，h(i)是从 k 到 t 的代价。

---

① https://www.hackerearth.com/zh/practice/notes/a-search-algorithm/

② 在“BFS+优先队列”一节中，指出“dijkstra+优先队列”=“BFS+优先队列”。

若 g = 0，则 f = h，A*就退化为贪心搜索；

若 h = 0，则 f = g，A*就退化为 Dijkstra。

A*每次根据最小的 f(i)来选择下一个点。 g(i)是已经走过的路径，是已知的；h(i)是预测未走过的路径；所以 f(i)的性能取决于 h(i)的计算。

A*算法的复杂度，在最差情况下的上界是 Dijkstra，或者 BFS+队列，一般情况下会更优。

4. A*算法的最优性

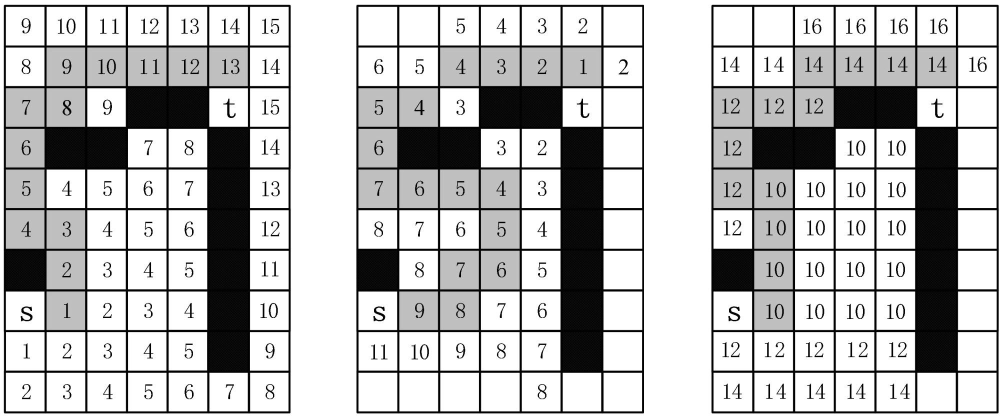
A*算法的解是最优的吗？答案是确定的，它的解和 Dijkstra 的解一样，是最短路径。

当 k 到达终点 t 时，有 h(t) = 0，那么 f(t) = g(t) + h(t) = g(t)，而 g(t)是通过 Dijkstra 求得的最优解，所以在终点 t 这个位置，A*算法的解是最优的。

总结：A*算法通过 Dijkstra 获得最优性结果；通过贪心最优搜索预测扩展方向，减少搜索的结点数量。

5. 三种算法对比

下面这张图[①]，准确地说明了三种算法的区别。图中起点是 s，终点是 t，黑格是障碍。图中精心设置了障碍的位置，以演示三种算法是如何绕过障碍的。



(1) Dijkstra　　(2) Greedy Best First　　(3) A*Search

图 1 三个算法对比

三个算法都基于"**BFS+优先队列**"。有数字的格子是搜索过的结点，并进入优先队列处理。阴影格是最后得到的一条完整路径。格子中的数字是距离，按曼哈顿距离计算。Dijkstra（BFS）算法遍历的格子最多，贪心最优搜索算法遍历的格子最少。

（1）Dijkstra（BFS）算法。格子中的数字，是从起点 s 到这个格子的最短距离。算法搜索格子时，把这些格子到起点的距离送入优先队列，当弹出时，就得到了 s 到这些格子的最短路径。最后，当终点 t 从优先队列弹出时，即得到 s 到 t 的最短距离 14。

（2）贪心最优搜索。格子中的数字，是从这个格子到终点 t 的曼哈顿距离。读者可以仔细分析它的工作过程，这里简单说明如下：从 s 沿最小曼哈顿距离，一直走到碰壁处的 2；2 从优先队列弹出后，剩下最小的是 3；3 弹出后，剩下最小的是 4；......；持续这个过程，那些看起来更近但是最终碰壁的结点被逐个弹走，直到拐过障碍，最后到达 t。得到的路径不是最优路径。

（3）A*搜索。例如格子 i 中的数字，是“s 到 i 的最短路 + i 到 t 的曼哈顿距离”。算法在扩展格子的过程中，标记数字的格子都会进入优先队列。在图示中，先弹出所有标记为 10 的格子，再弹出标记为 12 的格子，直到最后弹出终点 t。最后得到的 s-t 最短路径也是 14。

如何打印出完整的一条路径？三个算法都基于 BFS，而 BFS 记录路径是非常简单的：在结点 u 扩展邻居点 v 的时候，在 v 上记录它的前驱结点 u，即可以从 v 回溯到 u；到达目的后，

① 强烈推荐动图演示：https://www.redblobgames.com/pathfinding/a-star/introduction.html

从终点逐步回溯到起点，就得到了路径。在 Dijkstra 算法中，每次从优先队列中弹出的，都是得到了最短路径的结点，从它们扩展出来的邻居结点，也会继续形成最短路径，所以能根据前驱和后继结点的关系，方便地打印出一条完整的最短路径[①]。A*算法用 Dijkstra 算法来确定前驱后继的关系，也一样可以打印出一条最短路径。贪心最优搜索的路径打印最简单，就是普通 BFS 的路径打印。

6. 函数 h 的设计

在二维平面上，有 3 种方法可以近似计算 h。下面的(i.x, i.y)是 i 点的坐标，(t.x, t.y)是终点 t 的坐标。

（1）曼哈顿距离。应用场景：只能在四个方向（上，下，左，右）移动。

h(i) = abs (i.x – t.x) + abs (i.y – t.y)

（2）对角线距离。应用场景：可以在八个方向上移动，例如国际象棋的国王的移动。

h(i) = max {abs(i.x – t.x), abs(i.y – t.y)}

（3）欧几里得距离。应用场景：可以向任何方向移动。

h(i) = sqrt ($(i.x – t.x)^2$ + $(i.y – t.y)^2$)

非平面问题，需要设计合适的 h 函数，后面的例题中有一些比较复杂的 h 函数。

设计 h 时注意以下基本规则：

（1）g 和 h 应该用同样的计算方法。例如 h 是曼哈顿距离，g 也应该是曼哈顿距离。如果计算方法不同，f = g + h 就没有意义了。

（2）根据应用情况正确选择 h。各个结点的 h 值，应该能正确反映它们到终点的距离远近。例如下一跳结点有 2 个选项：A(280, 319)、B(300, 300)，如果用曼哈顿距离应该选 A，用欧氏距离应该选 B。如果只能走四个方向（需要按曼哈顿距离计算路径），用欧式距离计算就会出错。

（3）**h 应该优于实际存在的所有路径**。前面的例子中，h(i)小于等于 i-t 的所有可能路径长度，也就是说，最后得到的实际路径，长度大于 h(i)。这个规则可以用下面两点讨论来说明。

1）h(i)比 i-t 的实际存在的最优路径长。假设这条实际的最优路径是 path，由于程序是根据 h(i)来扩展下一个结点的，所以很可能会放弃 path，而选择另一条非最优的路径，这会造成错误。

2）h(i)比 i-t 的所有实际存在的路径都短。此时 i-t 上并不存在一条长度为 h(i)的路径，如果程序根据 h(i)来扩展下一结点，最后肯定会碰壁；但是不要紧，程序会利用 BFS 的队列操作弹走这些错误的点，退回到合适的结点，从而扩展出实际的路径，所以仍能保证正确性。

上面第（3）点最重要，应用 A*算法时应特别注意。

1.1.3 A*算法例子

A*算法的主要难点是设计合适的 h 函数，而编码很容易。例如图问题中，Dijkstra 或 BFS 使用 g 函数，A*使用 f = g + h 函数，那么编码时只要用 f 代替 g 即可。读者可以尝试把图论的最短路径题目改成用 A*算法实现，例如 poj 2243[②]。

下面给出 2 个复杂一点的例题。

1. K 短路

问题描述[③]：给定一个图，定义起点 s 和终点 t，以及数字 k；求 s 到 t 的第 k 短的路。允许环路。相同长度的不同路径，也被认为是完全不同的。

K 短路问题是 A*算法应用的经典例子，几乎完全套用了 A*算法的估价函数。

① 如果需要打印出最短路径，参考《算法竞赛入门到进阶》“10.9 最短路”，给出了路径打印的代码。
② http://poj.org/problem?id=2243
③ http://poj.org/problem?id=2449

下面分别用暴力法和 A*算法求解。

（1）**暴力法：BFS + 优先队列**

用 BFS 搜索所有的路径，用优先队列让路径按从短到长的顺序输出。

“BFS+优先队列”求最短路，在“BFS+优先队列”这一节中曾讲解过。其原理是当再次扩展到某个点 i 时，如果这次的新路径比上次到达 i 的路径更短，就替代它；优点队列可以让结点按路径长短先后输出，从而保证最优性。队列的元素是一个二元组(i, dist(s-i))，即结点 i 和路径 s-i 的长度。

BFS 求所有路径，就是最简单的“BFS + 优先队列”，再次扩展邻居 i 时，计算它到 s 的距离，然后直接进队列，并不与上次 i 进队列的情况进行比较。一个结点 i 会进入优先队列很多次，因为可以从它的多个邻居分别走过来；每一次代表了一个从 s 到 i 的路径。优先队列可以让这些路径按 dist 从短到长的顺序输出，i 从优先队列中第 x 次弹出，就是 s 到 i 的第 x 个短路。对于终点 t，统计它出队列的次数，第 K 次时停止，这就是第 K 短路。

在 K 短路问题中，路径有可能形成环路。有的题目允许环路，有的不允许。如果允许环路，那么想在环路上绕多少圈都可以，环路上的结点反复进入队列，K 可以无限大。

在最短路算法中并不需要判断环路，因为更新操作有去掉环路的隐含作用。

复杂度：因暴力法需要生成几乎所有的路径，而路径数量是指数增长的，所以暴力法的复杂度非常高。

下面用一个简单的图例解释 BFS 暴力搜索所有路径的过程。

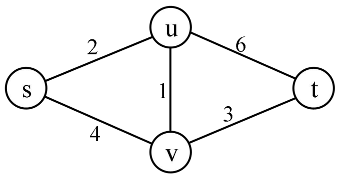


图 2 一个简单图

下面的表格给出了算法的步骤。结点后面的下标表示从 s 到这个结点的路径长度，例如 u_2，就是二元组(u, 2)，即结点 u，以及 s-u 的路径长度 2。步骤中没有列出环路。

步骤	出队	邻居进队	优先队列	新得到的路径	输出队头的路径
1		s	$\{s_0\}$		
2	s_0	u, v	$\{u_2, v_4\}$	$s\text{-}u_2$　　$s\text{-}v_4$	
3	u_2	v, t	$\{v_3, v_4, t_8\}$	$s\text{-}u_2\text{-}v_3$　　$s\text{-}u_2\text{-}t_8$	$s\text{-}u_2$
4	v_3	t	$\{v_4, t_8, t_6\}$	$s\text{-}u_2\text{-}v_3\text{-}t_6$	$s\text{-}u_2\text{-}v_3$
5	v_4	u, t	$\{t_8, t_6, u_5, t_7\}$	$s\text{-}v_4\text{-}u_5$　　$s\text{-}v_4\text{-}t_7$	$s\text{-}v_4$
6	u_5	t	$\{t_8, t_6, t_7, t_{11}\}$	$s\text{-}v_4\text{-}u_5\text{-}t_{11}$	$s\text{-}v_4\text{-}u_5$
7	t_6		$\{t_8, t_7, t_{11}\}$		$s\text{-}u_2\text{-}v_3\text{-}t_6$
8	t_7	u	$\{t_8, t_{11}, u_{13}\}$	$s\text{-}v_4\text{-}t_7\text{-}u_{13}$	$s\text{-}v_4\text{-}t_7$
9	t_8	v	$\{t_{11}, u_{13}, v_{11}\}$	$s\text{-}u_2\text{-}t_8\text{-}v_{11}$	$s\text{-}u_2\text{-}t_8$
10	t_{11}		$\{u_{13}, v_{11}\}$		$s\text{-}v_4\text{-}u_5\text{-}t_{11}$
11	v_{11}		$\{u_{13}\}$		$s\text{-}u_2\text{-}t_8\text{-}v_{11}$
12	u_{13}		$\{\}$		$s\text{-}v_4\text{-}t_7\text{-}u_{13}$

从第二列的“出队”可以看到，共产生 10 个路径，按从短到长的顺序排队输出。从起点 s 到终点 t 共有 4 条路径，t 在第 7、8、9、10 步出队的时候，输出了第 1、第 2、第 3、第 4 路径。表格中也列出了 s 到每个结点的多个路径和它们的长度，例如 s-u 有 3 个路径，s-v 有 3 个路径。

（2）**A*算法求 K 短路**

从暴力法可以知道：

1）从优先队列弹出的顺序，是按这些结点到 s 的距离排序的。

2）一个结点 i 从优先队列第 x 次弹出，就是 s-i 的第 x 短路；终点 t 从队列中第 K 次弹出，就是 s-t 的第 K 短路。

如何优化暴力法？是否可以套用 A*算法？

联想前面讲解 A*算法求最短路的例子，A*算法的估价函数 $f(i) = g(i) + h(i)$，g 是从起点 s 到 i 的距离，h 是 i 到终点 t 的最短距离（例子中是曼哈顿距离）。

那么在 K 短路问题中，可以设计几乎一样的估价函数。g(i)仍然是起点 s 到 i 的距离；而 h(i)，只是把曼哈顿距离改为从 i 到 t 的最短距离。这个最短距离如何求？用 Dijkstra 算法，以终点 t 为起点，求所有结点到 t 的最短距离即可。

编程非常简单。仍用暴力法的“BFS+优先队列”，但是在优先队列中，用于计算的不再是 g(i)，而是 f(i)。当终点 t 第 K 次弹出队列时，就是第 K 短路。

根据前面对 A*算法原理的解释，求 K 短路的过程将得到很大优化。虽然在最差情况下，算法复杂度的上界仍是暴力法的复杂度，但优化是很明显的。

2. poj 1945

Power Hungry Cows　http://poj.org/problem?id=1945

题目描述：两个变量 a、b，初始值 a = 1，b = 0。每一步可以执行一次 a×2，b×2，a+b，|a-b| 之一的操作，并把结果再存回 a 或者 b。问最快多少步能得到一个整数 P，1 <= P <= 20,000。

例如 P = 31，需要 6 步：

	a	b
初始值：	1	0
a×2，存到 b	1	2
b×2：	1	4
b×2：	1	8
b×2：	1	16
b×2：	1	32
b-a：	1	31

样例输入：

31

样例输出：

6

题解：

（1）BFS+剪枝

这一题是典型的 BFS。从{a, b}可以转移到 8 种情况，即{2a, a}、{2a, b}、{2b,a}、{2b,b}等等。把每种{a, b}看成一个状态，那么 1 个状态可以转移到 8 个状态。编码时，再加上去重和剪枝。此题 P 不是太大，“BFS+剪枝”可行。

（2）A*算法

如何设计估价函数 $f(i) = g(i) + h(i)$ ？

g(i)是从初始态到 i 状态的步数。h(i)是从 i 状态到终点的预期步数，它应该小于实际的步数。如何设计呢？容易观察到，{a, b}中的较大数，一直乘以 2 递增，是最快的。例如样例中的 31，在起点状态，$2^5 > 31$，经 5 步可以超过目标值，所以 h = 5。

3. 一些习题

下面的题目有多种实现方法，尝试用 A*算法来做。

洛谷 P1379 八数码难题，https://www.luogu.com.cn/problem/P1379。八数码有多种解法，A*也是经典解法之一。

洛谷 2324 骑士精神，https://www.luogu.com.cn/problem/P2324
洛谷 P2901 Cow Jogging https://www.luogu.com.cn/problem/P2901，K 短路。