一步步改造 BFS: 从盲目搜索到启发式搜索

问题求解（三）第 2 周 Open Topic

黄文睿　221180115

南京大学

主要内容

1.1 回顾 BFS: 它可以解决的问题

- BFS 可以解决的是**无权图**中的最短路问题。
- 为了和下文保持一致，只考虑 s 和 t 两点之间的最短路问题，记 s 到 u 的最短路为 $g(u)$。
- 算法流程就不讲了。

1.1 回顾 BFS: 它可以解决的问题

- BFS 可以解决的是**无权图**中的最短路问题。
- 为了和下文保持一致，只考虑 s 和 t 两点之间的最短路问题，记 s 到 u 的最短路为 $g(u)$。
- 算法流程就不讲了。
- 为什么它是正确的呢？

1.1(cont'd) BFS 正确性证明

任意非负权图上的寻找 s 到 t 的最短路的算法

维护一个优先队列（关键字为 $g(u)$），每次取出关键字最小的顶点，更新其邻点的 g 值，若有更新则把邻点入队。第一次 t 出队时，$g(t)$ 的值就是 s 到 t 的距离。

1.1(cont'd) BFS 正确性证明

任意非负权图上的寻找 s 到 t 的最短路的算法

维护一个优先队列（关键字为 $g(u)$），每次取出关键字最小的顶点，更新其邻点的 g 值，若有更新则把邻点入队。第一次 t 出队时，$g(t)$ 的值就是 s 到 t 的距离。

证明.

- 每个点 u 在出队后，$g(u)$ 不再改变（之后队中点都 $\geq g(u)$）。
- t 第一次出队之前，所有可能更新 $g(t)$ 的 t 的邻点都被考虑过了，优先队列保证该记录是最小的，还未考虑过的不可能更新 t。

□

1.1(cont'd) BFS 正确性证明

- BFS 中队列的特征？

1.1(cont'd) BFS 正确性证明

- BFS 中队列的特征？
- 非减！它就是一个优先队列！
- BFS 队列的性质（任一时刻至多只有两种值 v 和 $v+1$）保证了它的非减。

1.1(cont'd) BFS 正确性证明

- BFS 中队列的特征？
- 非减！它就是一个优先队列！
- BFS 队列的性质（任一时刻至多只有两种值 v 和 $v+1$）保证了它的非减。
- 那自然是正确的。

1.1(cont'd) BFS 正确性证明

- BFS 中队列的特征？
- 非减！它就是一个优先队列！
- BFS 队列的性质（任一时刻至多只有两种值 v 和 $v+1$）保证了它的非减。
- 那自然是正确的。
- 应用：编辑距离问题，寻路问题等。

1.2 第一次扩展：0/1 BFS

修改一下问题

若图中有两种边，一种权为 1，一种权为 0，求最短路?

1.2 第一次扩展：0/1 BFS

修改一下问题

若图中有两种边，一种权为 1，一种权为 0，求最短路?

也修改一下算法! 在 BFS 更新 $g(u)$ 时:

- 如果从 1 边更新，把 u 放到队尾;
- 如果从 0 边更新，把 u 放到队首。

1.2 第一次扩展：0/1 BFS

修改一下问题

若图中有两种边，一种权为 1，一种权为 0，求最短路?

也修改一下算法! 在 BFS 更新 $g(u)$ 时:

- 如果从 1 边更新，把 u 放到队尾;
- 如果从 0 边更新，把 u 放到队首。

这样的队列同样非减、至多只有两值，正确性是保证了的。

1.2(cont'd) 0/1 BFS 算法流程

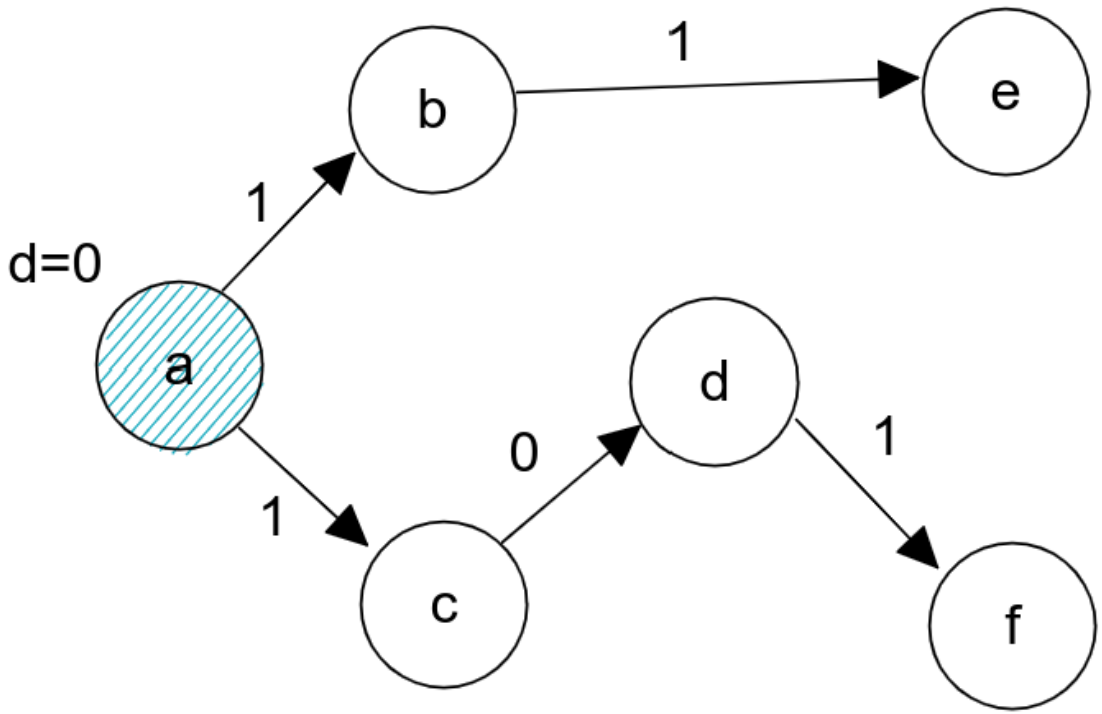

图: 0/1 BFS 算法流程（一）

1.2(cont'd) 0/1 BFS 算法流程

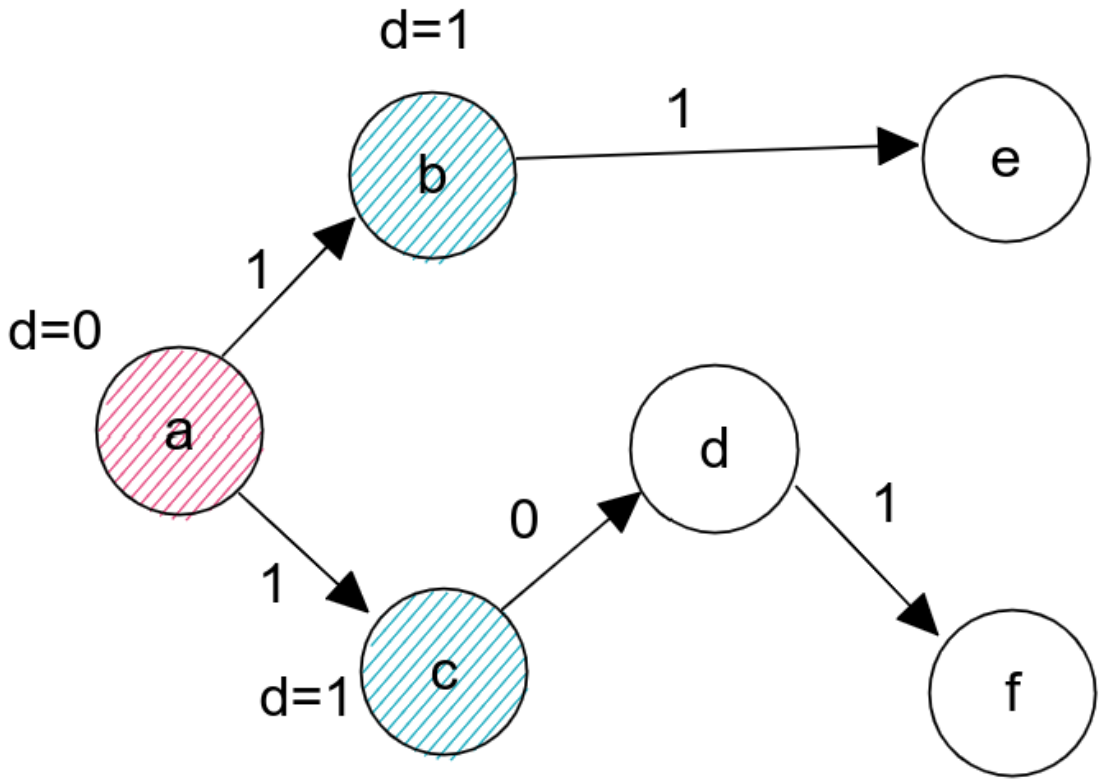

图: 0/1 BFS 算法流程（二）

1.2(cont'd) 0/1 BFS 算法流程

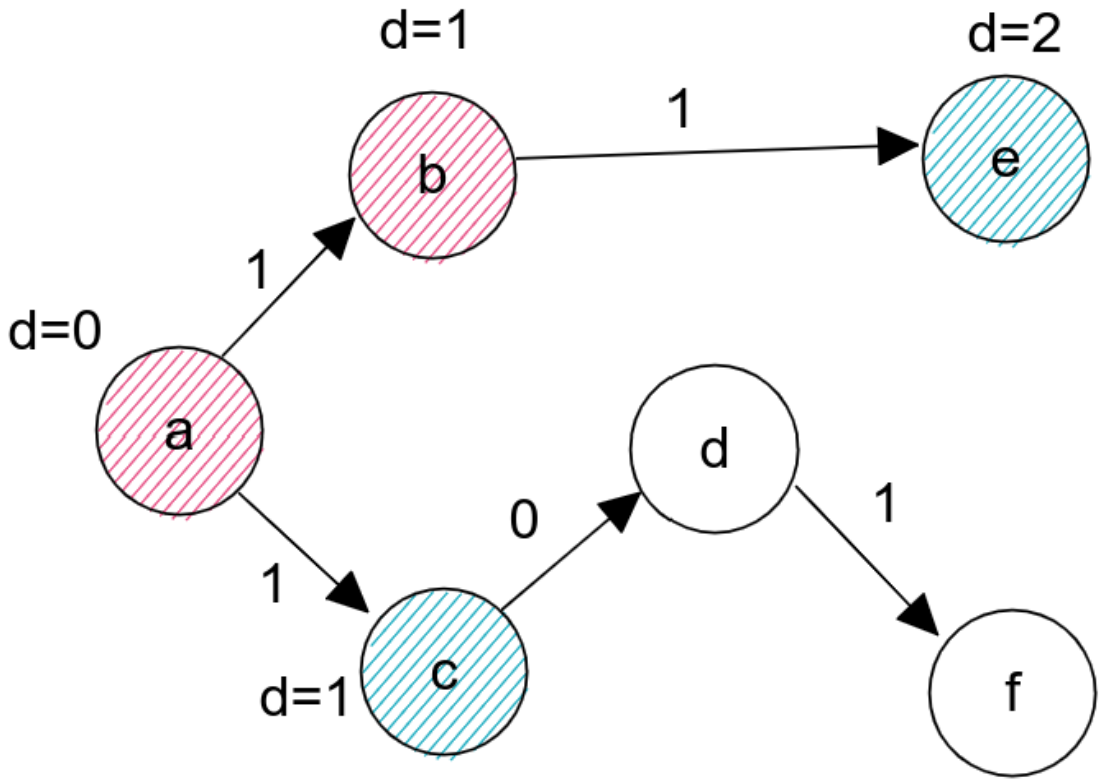

图: 0/1 BFS 算法流程（三）

1.2(cont'd) 0/1 BFS 算法流程

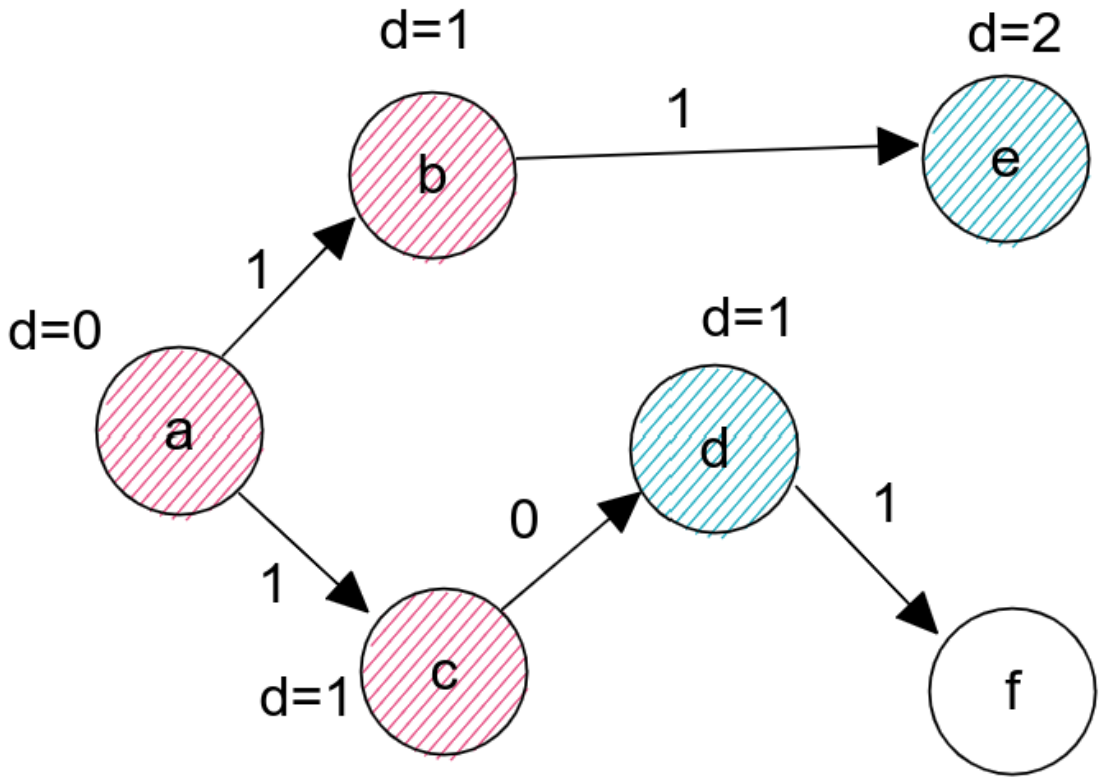

图: 0/1 BFS 算法流程（四）

1.2(cont'd) 0/1 BFS 算法流程

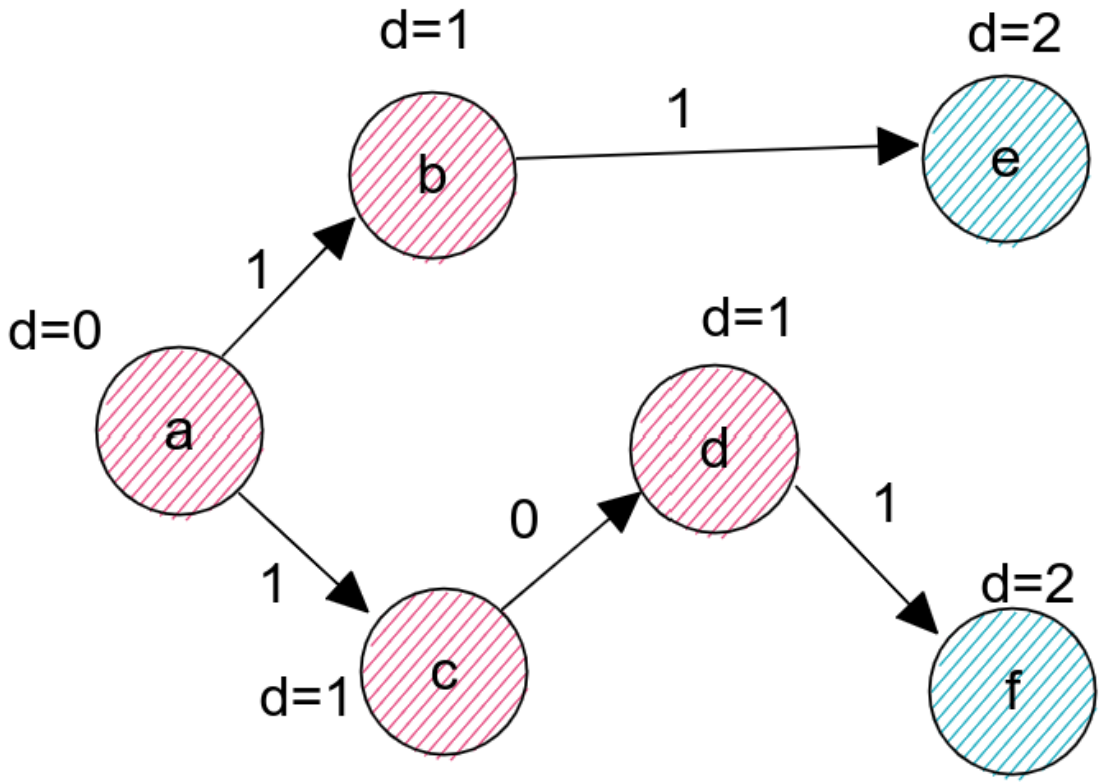

图: 0/1 BFS 算法流程（五）

1.2(cont'd) 0/1 BFS 算法流程

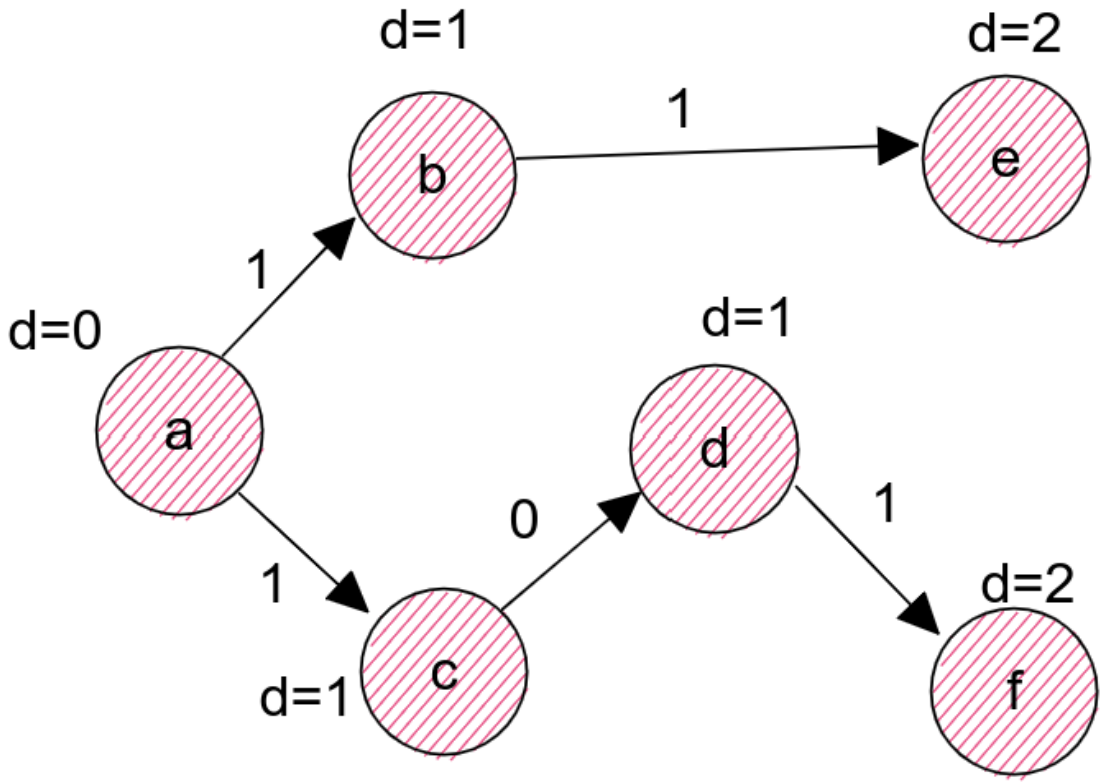

图: 0/1 BFS 算法流程（六）

1.2(cont'd) 0/1 BFS 的应用

0 边不是必须的，但是很方便。

e.g. 在地铁方案计算（考虑站数，即无权图）中，引入零边，把换乘站拆成多个点，之间用 0 边连接，就可以用 0/1 BFS 了。

思考题

用普通的 BFS 真的不能处理有 0/1 边的图吗?

1.3 两种不同的搜索问题

1. 图比较小，容易得知图的全貌，知道阶 n、边数 m、图的结构特征等，往往可以搜索整个图得到时间复杂度用 n 和 m 描述的算法，比如 BFS、DFS 的时间、空间复杂度都是 $O(n+m)$。
   e.g. 常见的寻路问题等。

1.3 两种不同的搜索问题

1. 图比较小，容易得知图的全貌，知道阶 n、边数 m、图的结构特征等，往往可以搜索整个图得到时间复杂度用 n 和 m 描述的算法，比如 BFS、DFS 的时间、空间复杂度都是 $O(n+m)$。
   e.g. 常见的寻路问题等。
2. 图很大（甚至无限大），难以得到图的全貌。当只考虑两点 s 和 t 之间的最短路时，可以用分支因子 b 和深度 d 来描述。BFS 的时间、空间复杂度都是 $O(b^d)$; DFS 在限定只搜 d 层就返回时，时间是 $O(b^d)$ 但空间是 $O(d)$ 的。
   e.g. n 皇后问题，巨型迷宫搜索，竖式填写问题。

$$\begin{array}{rr} & 43\#9865\#045 \\ + & 8468\#6633 \\ \hline & 44445509678 \end{array}$$

图: 竖式填写问题

1.4 BFS 的空间问题

BFS 的时间、空间都是 $O(b^d)$，都是较高的指数型增长。

[1] picture from https://yey.world/2020/03/06/COMP90054-02/

1.4 BFS 的空间问题

BFS 的时间、空间都是 $O(b^d)$，都是较高的指数型增长。
实际上，空间往往比时间更难以忍受。
取 $b = 10$，计算机速度 1 Knodes/s，内存消耗 1 KB/node: [1]

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

图: BFS 的时空消耗

$d = 12$ 时，时间是 13 days，空间是 1 PB!

[1] picture from https://yey.world/2020/03/06/COMP90054-02/

1.4(cont'd) 迭代加深搜索

迭代加深搜索 (Iterative deepening DFS) 是一种特殊的 DFS 算法，第一次只搜 $d \leq 1$ 的顶点，第二次 $d \leq 2$，依次类推，直到找到目标结点。

1.4(cont'd) 迭代加深搜索

迭代加深搜索 (Iterative deepening DFS) 是一种特殊的 DFS 算法，第一次只搜 $d \leq 1$ 的顶点，第二次 $d \leq 2$，依次类推，直到找到目标结点。

- (正确性) 每次有意义的只是在“返回层”，而在“返回层”之前的搜索都是重复的，事实上等价于 BFS。

1.4(cont'd) 迭代加深搜索

迭代加深搜索 (Iterative deepening DFS) 是一种特殊的 DFS 算法，第一次只搜 $d \leq 1$ 的顶点，第二次 $d \leq 2$，依次类推，直到找到目标结点。

- (正确性) 每次有意义的只是在“返回层”，而在“返回层”之前的搜索都是重复的，事实上等价于 BFS。
- (时间复杂度) 看起来重复搜索了了很多：第 1 层搜了 d 次，第 2 层搜了 $d-1$ 次，依次类推，第 d 层搜了 1 次，有

$$T = O(b^0 \cdot d + b^1 \cdot (d-1) + \cdots + b^{d-1} \cdot 1) = O(b^d), b > 1.$$

事实上只有常数倍代价。

1.4(cont'd) 迭代加深搜索

迭代加深搜索 (Iterative deepening DFS) 是一种特殊的 DFS 算法，第一次只搜 $d \leq 1$ 的顶点，第二次 $d \leq 2$，依次类推，直到找到目标结点。

- (正确性) 每次有意义的只是在“返回层”，而在“返回层”之前的搜索都是重复的，事实上等价于 BFS。
- (时间复杂度) 看起来重复搜索了了很多：第 1 层搜了 d 次，第 2 层搜了 $d-1$ 次，依次类推，第 d 层搜了 1 次，有

$$T = O(b^0 \cdot d + b^1 \cdot (d-1) + \cdots + b^{d-1} \cdot 1) = O(b^d), b > 1.$$

  事实上只有常数倍代价。
- (空间复杂度) 和 DFS 一样，$O(d)$。

1.4(cont'd) IDDFS 算法流程

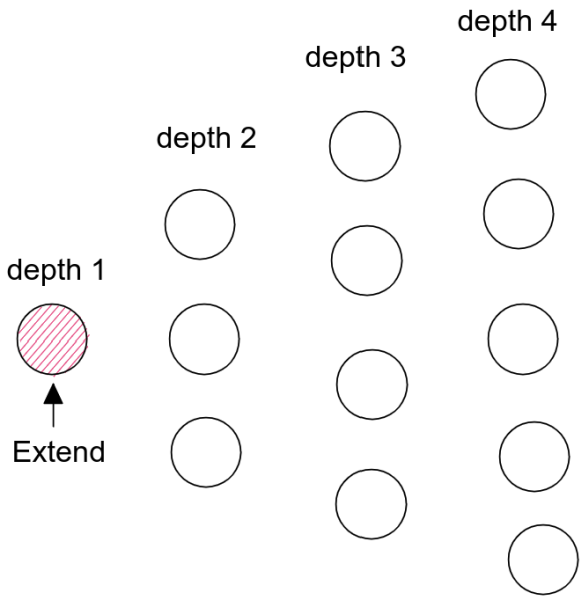

图: IDDFS 算法流程（一）

1.4(cont'd) IDDFS 算法流程

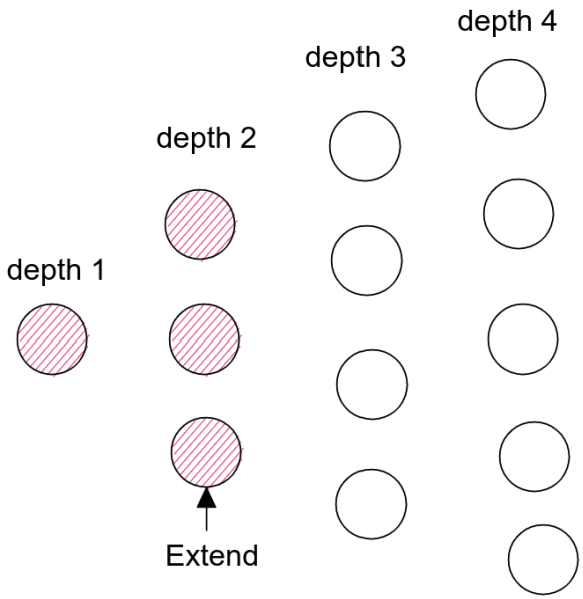

图: IDDFS 算法流程（二）

1.4(cont'd) IDDFS 算法流程

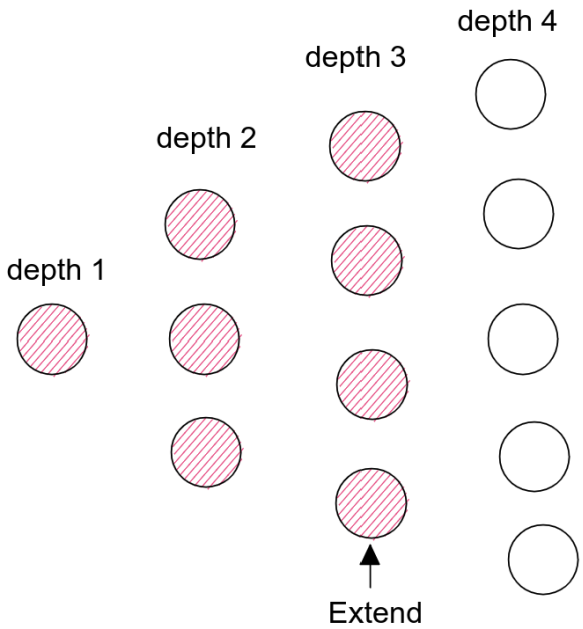

图: IDDFS 算法流程（三）

1.4(cont'd) IDDFS 算法流程

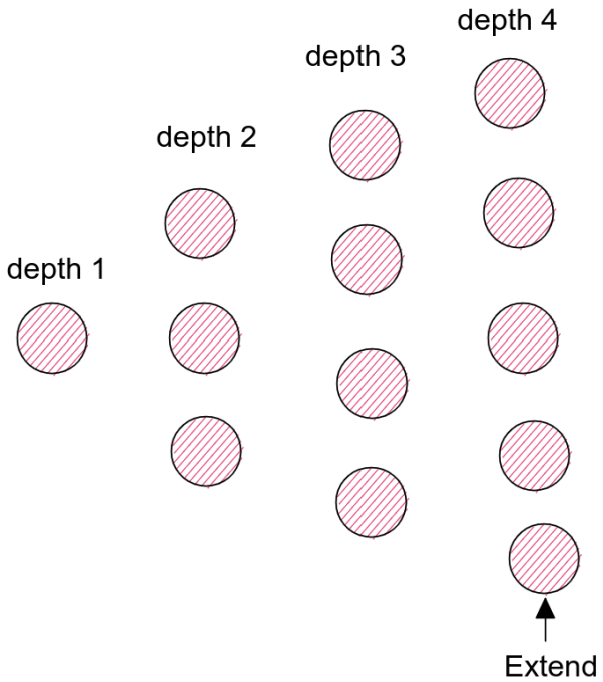

图: IDDFS 算法流程（四）

1.5 一致代价搜索 (Dijkstra 算法)

- 当边权全是 1 或者有 0 有 1 时，BFS 已经可以完美解决了。

1.5 一致代价搜索 (Dijkstra 算法)

- 当边权全是 1 或者有 0 有 1 时，BFS 已经可以完美解决了。
- 如果边权可以是任意非负数，就要回到之前的定理了。

任意非负权图上的寻找 s 到 t 的最短路的算法

维护一个优先队列（关键字为 $g(u)$），每次取出关键字最小的顶点，更新其邻点的 g 值，若有更新则把邻点入队。第一次 t 出队时，$g(t)$ 的值就是 s 到 t 的距离。

1.5 一致代价搜索 (Dijkstra 算法)

- 当边权全是 1 或者有 0 有 1 时，BFS 已经可以完美解决了。
- 如果边权可以是任意非负数，就要回到之前的定理了。

任意非负权图上的寻找 s 到 t 的最短路的算法

维护一个优先队列（关键字为 $g(u)$），每次取出关键字最小的顶点，更新其邻点的 g 值，若有更新则把邻点入队。第一次 t 出队时，$g(t)$ 的值就是 s 到 t 的距离。

用数据结构维护优先队列则可应用在一般图上。这就是一致代价搜索 (Dijkstra 算法)。

- （时间复杂度）用线性表维护优先队列 $O(n^2+m)$，用于稠密图；用二叉堆维护优先队列 $O((n+m)\log m)$，用于稀疏图。
- （空间复杂度）$O(n+m)$。

主要内容

2.1 盲目搜索和启发式搜索

之前提到的搜索，都是盲目搜索。

- 盲目搜索：仅使用最基本的信息，不知道搜索空间的特征，效率一般低下；
- 启发式搜索：预先知道了额外的一些信息——知道当前点 u 离目标 t 的差距。

2.1 盲目搜索和启发式搜索

之前提到的搜索，都是盲目搜索。

- 盲目搜索：仅使用最基本的信息，不知道搜索空间的特征，效率一般低下；
- 启发式搜索：预先知道了额外的一些信息——知道当前点 u 离目标 t 的差距。

A* 算法

在一致代价搜索 (Dijkstra) 基础上，优先队列中排序的关键词不仅仅是 $g(u)$（起点 s 到 u 的距离），改为了

$$f(u) = g(u) + h(u).$$

其中 $h(u)$ 是**估计**的从 u 到 t 还需要的代价。

2.2 A* 的最优性

关于最优性的疑问

按照 $f(u) = g(u) + h(u)$ 排序是否可以找到从 s 到 t 的最短路？

2.2 A* 的最优性

关于最优性的疑问

按照 $f(u) = g(u) + h(u)$ 排序是否可以找到从 s 到 t 的最短路？

记 $h(u)$ 是从 u 到 t 的**估计**代价，$h^*(u)$ 是从 u 到 t 的实际最小代价，我们断言，只要

$$\forall u, 0 \leq h(u) \leq h^*(u),$$

那么 A* 算法是满足最优性的。即，我们估计的 $h(u)$ 只能少估不能多估。

2.2 A* 的最优性

关于最优性的疑问

按照 $f(u) = g(u) + h(u)$ 排序是否可以找到从 s 到 t 的最短路？

记 $h(u)$ 是从 u 到 t 的**估计**代价，$h^*(u)$ 是从 u 到 t 的实际最小代价，我们断言，只要

$$\forall u, 0 \leq h(u) \leq h^*(u),$$

那么 A* 算法是满足最优性的。即，我们估计的 $h(u)$ 只能少估不能多估。
比如，在迷宫搜索问题中，选择 $h(u)$ 是 u 到 t 的曼哈顿距离是可行的。

2.2(cont'd) A* 最优性证明

定理

若

$$\forall u, 0 \leq h(u) \leq h^*(u),$$

则 A* 算法满足最优性。

证明.

显然 $0 \leq h(t) \leq h^*(t) = 0$，从而 $h(t) = 0$，故 $f(t) = g(t)$。所以选择 $f(t)$ 最小也就是选择 $g(t)$ 最小。
对于可达 t 的它的邻点 u，它要么还没入队，要么在队列中 t 的后面，有

$$f(t) \leq f(u) = g(u) + h(u) \leq g(u) + h^*(u),$$

而 $g(u) + h^*(u)$ 是从 u 到 t 的最小代价，不会更优。 □

2.3 A* 算法的分析与 IDA*

A* 算法的时空分析：

- A* 算法快在哪里？可以认为减少了分支因子 b，使得整个搜索树更窄。A* 算法本质来说，还是 Dijkstra 算法。
- 设减少后的分支因子为 b_*，时间、空间复杂度都是 $O(b_*^d)$。b_* 的好坏与 h 的构造有关。

2.3 A* 算法的分析与 IDA*

A* 算法的时空分析：

- A* 算法快在哪里？可以认为减少了分支因子 b，使得整个搜索树更窄。A* 算法本质来说，还是 Dijkstra 算法。
- 设减少后的分支因子为 b_*，时间、空间复杂度都是 $O(b_*^d)$。b_* 的好坏与 h 的构造有关。

思考题

Dijkstra 和 A* 可否类似于 IDDFS 一样用 DFS 替代？

2.3 A* 算法的分析与 IDA*

A* 算法的时空分析：

- A* 算法快在哪里？可以认为减少了分支因子 b，使得整个搜索树更窄。A* 算法本质来说，还是 Dijkstra 算法。
- 设减少后的分支因子为 b_*，时间、空间复杂度都是 $O(b_*^d)$。b_* 的好坏与 h 的构造有关。

思考题

Dijkstra 和 A* 可否类似于 IDDFS 一样用 DFS 替代？

可以！只需把深度改为 $f(u)$，每次只考虑 $f(u) \leq d$ 的结点，下次迭代选择在 $f(v) > d$ 的点中的 $f(u)$ 最小点。

参考资料

1. https://en.wikipedia.org/wiki/Breadth-first_search
2. https://yey.world/2020/03/06/COMP90054-02/
3. https://yey.world/2021/03/12/COMP90054-03/
4. https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search
5. https://hljmssjg.github.io/2021/11/02/关于搜索算法的复习/
6. https://en.wikipedia.org/wiki/Uniform-cost_search
7. https://en.wikipedia.org/wiki/A*_search_algorithm
8. https://en.wikipedia.org/wiki/Iterative_deepening_A*

谢谢大家!