

Union-Find

Dynamic Equivalence Relation

Example

- Minimum Spanning Tree 的 Kruskal 算法：选择权最小的边，若**不成环**则加入结果
- Maze generation：生成随机迷宫
- Black Pixels：最大黑色像素的 component
- Jigsaw Puzzle

Equivalence

满足自反，对称，传递的二元关系

In [mathematics](#), an **equivalence relation** is a [binary relation](#) that is [reflexive](#), [symmetric](#) and [transitive](#). The relation "is equal to" is the canonical example of an equivalence relation, where for any objects a , b , and c :

- $a = a$ (reflexive property),
- if $a = b$ then $b = a$ (symmetric property), and
- if $a = b$ and $b = c$ then $a = c$ (transitive property).

相互等价的所有元素构成一个等价类，所有等价类构成全局的划分

The equivalence class of an element a is denoted $[a]$ and is defined as the set

$$[a] = \{x \in X | a \sim x\}$$

参见

- [Equivalence relation](#)
- [Equivalence class](#)
- [Partition of a set](#)

Dynamic equivalence relation

等价关系动态变化，支持 IS, MAKE 操作

- IS: 两个元素是否属于同一等价类
- MAKE: 合并两个不同的等价类

Union-Find 的输入规模定义: n 个元素, m 个 IS 或者 MAKE 的操作序列

基于 Union-Find 的问题

- The maze problem
 - Randomly delete a wall and **union** two cells
 - Loop until you **find** the inlet and outlet are in one equivalent class
- The Kruskal algorithm
 - **Find** whether u and v are in the same equivalent class
 - If not, add the edge and **union** the two nodes
- The black pixels problem
 - **Find** black pixel groups
 - How the **union** of black pixel groups increases α

BF implementation

- Matrix (relation matrix): 空间复杂度 $\Theta(n^2)$, 时间复杂度 $\Omega(mn)$
- Array (for equivalence class ID): 空间复杂度 $\Theta(n)$, 时间复杂度 $\Omega(mn)$

需要更高效的实现方式

Forest of rooted trees

- A collection of disjoint sets, supporting Union and Find operations
- Not necessary to traverse all the elements in one set

Disjoint Set

Rooted tree

rooted tree: 每个结点存放集合中的一个元素, 每个结点有 parent 域指向其父结点, 所有元素存放在若干颗树中, 每棵树对应一个等价类, 根节点即是其代表元

Union-Find program: 在 n 次创建结点操作后 m 条 union/find 的指令序列

定义 Union-Find program 作为 Union-Find 的时间复杂度的输入, 代价的度量为计算访问 parent 域的次数

Straightforward Union-Find

基于 rooted tree 的朴素 find 和 union 操作

- find: 反复访问 parent 直至根, 判断两个元素的根是否相同
- union: 将一棵树的根的 parent 指向另一颗树的根

最坏情况由 union 操作产生了长度为 $n - 1$ 的链, 每次 find 操作代价均为 $O(n)$

最坏情况时间复杂度为 $\Theta(mn)$

Weighted Union + Straightforward Find

weighted union: 将结点数较少的树作为子树

引理 10.2 当采用 weighted union 操作时, 包含 k 个结点的树, 高度至多为 $\lfloor \log k \rfloor$

上述引理可采用对结点数归纳证明

- base case: $k = 0, h = 0$
- inductive hypothesis: $h_1 \leq \lfloor \log k_1 \rfloor, h_2 \leq \lfloor \log k_2 \rfloor$
- inductive step: $h = \max(h_1, h_2 + 1), k = k_1 + k_2$
 - if $h = h_1, h \leq \lfloor \log k_1 \rfloor \leq \lfloor \log k \rfloor$
 - if $h = h_2 + 1$, note: $k_2 \leq k/2$ so, $h_2 + 1 \leq \lfloor \log k_2 \rfloor + 1 \leq \lfloor \log k \rfloor$

定理 10.4 采用 weighted union 和 find, 对于 n 个元素的并查集和 m 条指令的 program, 最坏情况时间复杂度为 $\Theta(n + m \log n)$

显然, 最多进行 $n - 1$ 次 union, 树高最多为 $\lfloor \log n \rfloor$, 故每次 find 的代价最多为 $\lfloor \log n \rfloor + 1$, find 的代价不超过 $O(m \log n)$, 初始化的代价为 $O(n)$

Weighted Union + Path-compressing Find

Path compression find: 查找一个结点的根时, 将其路径上所有结点的 parent 修改为根

显然一次 c-find 的代价是普通 find 的两倍, 但是之后的 c-find 代价会显著降低 (常数时间)

c-find 为 expensive operation, 但是只会做有限次 c-find

分析思路: **amortized analysis**

超指数函数 $H(n)$

$$H(n) = \begin{cases} 1, & \text{for } n = 0 \\ 2^{H(n-1)}, & \text{for } n > 0 \end{cases}$$

根据超指数函数可定义 \log^*

$$\log^*(j) = \min\{k | H(k) \geq j\}$$

超指数函数增长非常快，相对地 \log^* 增长很慢，可以验证对于任意 $k > 0$

$$\lim_{n \rightarrow \infty} \frac{\log^*(n)}{\log^{(k)} n} = 0$$

结论：

定理 10.5 采用 weighted union 和 c-find，对于 n 个元素的并查集和 m 条指令的 program，最坏时间复杂度为 $O((n + m) \log^*(n))$

由于 \log^* 的增长速度非常慢，可以近似认为性能为 $O(n + m)$

详细证明可见下列参考文献

- [Efficiency of a Good But Not Linear Set Union Algorithm. Tarjan.pdf](#)
- [Set Merging Algorithms | SIAM Journal on Computing | Vol. 2, No. 4 | Society for Industrial and Applied Mathematics](#)