Union-Find

主要内容

- 等价类的动态变化
- 并查集

Dynamic Equivalence Relations - 动态等价关系

- 等价关系
  - 自反性、对称性、传递性（离散数学）
  - 等价类形成一种划分(partition)
- 动态等价关系
  - 在计算过程中变化
  - **IS** instruction: 判断两个元素是否在同一个等价类中
  - **MAKE** instruction: 合并等价类——通过关联两个不同元素，影响接下来的**IS**操作
  - starting as equality relation

Implementation: How to Measure

用处理一系列（m个）在一个有n个元素的集合S上**MAKE** and/or **IS**操作所使用的原子操作的数量来评估复杂度

Union-Find based Implementation (一些运用并查集的例子)

生成迷宫

- 在一个矩阵（或方格棋盘）中，一开始全填满"墙"，设定一个入口(inlet)，一个出口(outlet)
- 随机选某个位置推倒"墙"
- 如果位置i和j在同一个等价类(equivalence class)里，那么选另外一堵"墙"来推倒
- 否则把i和j放入同一个等价类
- 在入口和出口在同一个等价类时，迷宫构建完成

并查集实现

- 随机删除一个墙并union两个格子
- 循环直到find发现inlet和outlet在同一个等价类里

最小生成树——Kruskal算法

贪心(Greedy)

- 选一条边

        - 具有最小的权重
        - 不在现有的树中
    - 评估这条边
        - 不会导致回路的出现

关键问题

- 怎么判断不会产生回路

并查集实现

- Find查看顶点u和v是不是在同一个等价类里
- 如果不是，就不会产生回路，加入这条边然后union两个顶点

Black Pixels

- Maximum black pixel component
    - let α be the size of the component
- Color one pixel black
    - How α changes?
    - How to choose the pixel, to accelerate the change in α

- Find看两个black pixel是否在通过一个等价类里
- How the union will increase α

Jigsaw Puzzle

- 拼板
- 从“one player”到“two players”
    - ……

（PPT上没有解答）

并查集不同的实现方式

- 矩阵（relation matrix）
    - 空间$\Theta(n)$，最坏时间$\Omega(mn)$（主要花在MAKE的行拷贝）
- 数组（for 等价类 ID）
    - 空间$\Theta(n)$，最坏时间$\Omega(mn)$（主要花在MAKE的查找和替换）
- Forest of rooted trees

Union-Find ADT

- create

- find
- makeSet
- union

具体实现请百度或参考数据结构课程，建议自行实现一个

Union-Find Program

- 一个并查集程序是m(程序的size/输入的size)条union/find指令组成的指令序列
- The measure: number of accesses to the parent (用看parent次数来评估复杂度)
  - assignments: for union operations
  - lookups: for find operations

以上两种操作均称为link operation

Worst-Case Analysis for Union-Find Program

- 假设每次查看(lookup)/赋值(assignment)的cost为$O(1)$
- 每个makeSet或union一次赋值(assignment)，每个find做d+1词查看(lookup)（d指结点深度）

Union序列是一条长$n-1$的链，同时也是有最大高度的树的情况，也就是最坏情况，单支树

Find(1)需要n次lookup

需要的操作次数：$n+(n-1)+(m-n+1)n=\Theta(n)$

Weighted-Union: For Short Trees

- Weighted union (wUnion)
  - 总是让有更少结点的树成为子树，从而降低树的高度

cost for the program: $n+3(n-1)+2(m-n+1)$

Upper Bound of Tree Height

- 通过wUnion建的树高度不会超过$\lfloor \log k \rfloor$
- 归纳法证明：
  - base case: k = 1, 高0，满足
  - 归纳假设：$h_1 \le \lfloor \log k_1 \rfloor, h_2 \le \lfloor \log k_2 \rfloor$
  - $h=\max(h_1, h_2+1), k=k_1+k_2$
  - 如果$h=h_1$，则$h_1 \le \lfloor \log k_1 \rfloor \le \lfloor \log k \rfloor$
  - 如果$h=h_2+1$，注意$k_2 \le k/2$，从而$h_2+1 \le \lfloor \log k_2 \rfloor + 1 \le \lfloor \log k \rfloor$

Upper Bound for Union-Find Program

- 一个size为m的、在一个有n个元素的集合上进行的并查集程序，如果使用wUnion和普通find，那么最坏情况下要用$O(n+m\log n)$次link操作
- 证明：
  - n个元素，最多只能做n-1次wUnion，建一棵树高度最多为$\lfloor \log n \rfloor$
  - 从而，每次find的cost最多为$\lfloor \log n \rfloor + 1$
  - 每次union的cost是$O(1)$，分析上界考虑每次find最大值，m次wUnion/find操作序列的cost等同于m次find操作，即$m(\lfloor \log n \rfloor + 1) \in O(n+m\log n)$
  - 有算法可以达到$\Omega(n+(m-n)\log n)$步完成

Path Compression - find 优化

做法：把路径上的点的parent全都连到root上

Challenges for the Analysis - 分析方法

- cFind的cost普通find的2倍——find & assign
- cFind will traverse shorter paths

Analysis: the Basic Idea

- cFind贵
- 但是只有有限次cFind
- 所以可以用**平摊分析**

Co-Strength of wUnion and cFind

$O((n+m)\log^*(n))$

- link operations for a Union-Find program of length m on a set of n elements in the worse case.
- Implemented with wUnion and cFind

What's $\log^*(n)$?

$H(0)=1$

$H(i)=2^{H(i-1)}$

Then, $\log^*(n)$ for $j \geq 1$ is defined as: $\log^*(j)=\min\{k|H(k)>j\}$

特征

- 值域增长快
- 定义域增长慢

Definitions with a Union-Find Program P

- **Forest F**: P程序中union指令序列形成的森林，这里假设

- 使用wUnion
- P中的find操作全部忽略

- **任意树中结点v的高度**：根在v的子树的高度
- **Rank of v**：v在F中的高度

Constraints on Ranks in F

Lemmas 引理们

- rank为$r(r \geq 0)$的结点数上限是$n/2^r$
  - wUnion建的树最高$\lfloor \log n \rfloor$，稍微变形就有高为r的子树至少有2^r个结点，又有之前结点v的高度为以v为根的子树的高度，从而有这样的结点数量小于等于$n/2^r$
  - 根的rank为r的子树互不相交
- 最多有$\lfloor \log n \rfloor$个不同的rank
  - 集合S中共n个元素，换言之，建成的树F有n个结点

(wiki 上有证明，可以对比看)

Increasing Sequence of Ranks

- 在一棵树F中，从叶节点一路到根节点，所有rank都是严格递增的
- 当cFind改变了parent，新parent是旧parent的parent（就是cFind操作的功能），即前者rank必定大于后者

A Function Growing Extremely Slowly

$$\forall p \geq 0, \lim_{n \to \infty} \frac{\log^*(n)}{\log^{(p)}(n)} = 0$$

说明一件事情：log-star函数增长极度缓慢（正如前面**特征**所说），所以考虑分组(Grouping)。

往后这部分证明比较难，不考察，节约时间起见，在此暂不整理。

Grouping Nodes by Ranks

“怎么用这个log*呢，大家不用细究这个东西，只要有直观感受就行”

- rank在一定范围内，log*都差不多
- 分rank组

Very Few Groups

Amortized Cost of Union-Find

- Operations

- makeSet
- union, at most $n-1$

One Execution of cFind(w_0)

Amortizing Scheme for wUnion-cFind

“大家这边自己理解一下就好了”

“如果有兴趣，大家可以看一下”

“blablabla，大概就是这样一个算法”

“不是重点要掌握的一个内容”