Solution for Problem Set 8

201300035 方盛俊

Problem 1

1. {1} {2} {3} {4} {5} {6} {7} {8} {9} {10} {11} {12} {13} {14} {15} {16}
2. {1, 2} {3, 4} {5, 6} {7, 8} {9, 10} {11, 12} {13, 14} {15, 16}
3. {1, 2, 3, 4} {5, 6, 7, 8} {9, 10, 11, 12} {13, 14, 15, 16}
4. {1, 2, 3, 4, 5, 6, 7, 8} {9, 10, 11, 12} {13, 14, 15, 16}
5. {1, 2, 3, 4, 5, 6, 7, 8} {9, 10, 11, 12, 13, 14, 15, 16}
6. {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}

所以两次 `Find` 操作的结果都是 `x1`.

Problem 2

我们给定这一系列操作为:

1. n 次 $\mathrm{MakeSet}(x_i)$ 操作创建 $x_i, i = 1, 2, \cdots, n$
2. 先进行 $n/2$ 次 $\mathrm{Union}(x_i, x_{i+1}), i = 1, 3, 5, \cdots$
3. 再进行 $n/4$ 次 $\mathrm{Union}(x_i, x_{i+2}), i = 1, 5, 9, \cdots$
4. 以此类推, 进行 $n/2^k$ 次 $\mathrm{Union}(x_i, x_{i+k}), i = 1, 1 + 2^k, \cdots \wedge k = 1, 2, \cdots, \log n$
5. 因此总共进行了 $m' = n/2 + n/4 + \cdots + 1 \leqslant n$ 次 Union 操作, 并且我们最后得到且仅得到一棵树, 设其高度为 h.
6. 我们取该树高度为 1 的点 x' (即最底部的点), 进行 $m - m'$ 次 $\mathrm{Find}(x')$, 我们可知 $T(\mathrm{Find}(x')) = \Omega(h)$, 且我们又知道 $m - m' \geqslant m - n \geqslant m - \frac{1}{2}m = \frac{1}{2}m$, 因此我们知道, 即使仅仅考虑所有 $\mathrm{Find}(x')$ 的时间复杂度, 也已经有下界 $\frac{1}{2}m\Omega(h) = \Omega(mh)$ 了, 即 $T(m, n) = \Omega(mh)$.

所以, 对于这一系列操作, 我们只需要证明树高度 $h = \Omega(\log n)$, 即可证明 $T(m, n) = \Omega(m \log n)$.

我们知道, 对于两棵等高的树, 经过 Union 操作之后, 新生成的树是原来的树的高度加一.

对于 2 到 5 的每一步操作, 树的高度都会增加一, 而 2 到 5 总共执行了 $k = \log n$ 次, 因此最后树的高度为 $h = \log n + 1$.

于是我们就证明了, 总的时间复杂度为 $T(m, n) = \Omega(m \log n)$

Problem 3

(a)

当只存在一棵树, 且树呈链状时, 有着最坏的运行时间.

此时 $T(n) = \sum_{i=1}^{n} i = \frac{1}{2} n (n + 1) = \Theta(n^2)$

(b)

Find 操作会访问被查找的节点及它的所有父节点, 随后把路径上所有节点的父节点都改为根节点.

假定一次 Find 操作涉及到根节点 r, 父节点恰为 r 的节点 y, 除此以外的其他节点 x_i, 路径长度为 l, 我们分析一次 Find 操作中所有的开销: (1) 对 x_i 的访问和父节点指针赋值, 单独对 x_i 开销为 c_1, 一条路径上的总开销为 $(l - 2)c_1$; (2) 对 y 和 r 的访问, 开销为 c_2, 一个常数.

我们设其中一棵树的节点数为 m, 由于路径压缩, 对于这棵树上的任意节点, 我们都有开销 (1) 对应的次数不会超过 1 次, 因为经过一次压缩之后, x_i 类型节点就变为了 y 类型节点, 不会再有 (1) 的开销.

因此, 我们可得, 对于所有的树来说, 他们的节点总数是 n, 因此 (1) 的开销不会超过 $c_1 n$; 一共进行了 n 次 Find, 因此 (2) 的开销为 $c_2 n$.

最后的总时间复杂度为 $T(n) \leqslant c_1 n + c_2 n$, 因此 $T(n) = O(n)$.

Problem 4

我没办法将这道题完整地想出来, 只能给一些我大体的可能的想法. 大致如下:

为了让 `NewString(c)`, `Reverse(S)`, `Lookup(S, k)` 操作在最坏情况下花费 $O(1)$ 的时间, 并且结构体总的空间复杂度为 $O(n)$, 那么我就一定要初始化一个大小为 $O(n)$ 的大数组, 然后将所有的字符串都放在这同一个大数组里.

我们先定义这个结构体为 `strings_pool`, 具体定义如下:

```
struct strings_pool {

    List<string> {

        // 双向链表, 用于前后查询
        pointer prev;
        pointer next;

        // 字符串区间包括 begin, 不包括 end
        char *begin;    // 字符串区间的开始位置
        char *end;      // 字符串区间的结束位置
        char *head;     // 字符串的第一个字符的位置

        // 是否逆转
        bool reversed;

        // 字符串的长度, 函数式表达
        function len = () => { return this.end - this.begin; };

    } string_pointers;

    char array[n];      // 一个足够大的数组, 用于存放具体的字符串数据
}
```

其中, 如果 n 是变动的话, 那就在总量超过 $\frac{3}{4}n$ 的时候, n 扩容为原来的两倍, 并更新对应指针.

具体来说, `string_pointers` 应该是一个存储了一系列 S 指针的双向链表.

其中字符串区间用于保证我们移动一个字符串只需要移动一部分, 例如说, 如果我有 字符串 `[ | a, b, c, d]`, 我要把它向后移动两格, 就变成 `[c, d, | a, b]`, 其中竖线所在位置即 `head` 第一个字符位置.

我们每 `NewString` 一次, 就在 `array` 和 `string_pointers` 链表初始化和加入对应内容.

然后我们通过链表操作保证 `string_pointers` 里的顺序和 `array` 里存放的字符串顺序一致.

对于每一次 `Concat(S,T)`, 我们把二者中 `S.len()` 或 `T.len()` 小的那一个字符串拼接到长度大的字符串中 (当然, 要通过 `reversed` 属性进一步确定如何拼接). 这个过程中, 我们依次把冲突的字符串往相应的位置移动. 这时的 `Concat(S, T)` 均摊复杂度就很难分析了, 也许已经超过了 $O(\log n)$, 但是也许加入一定冗余之后, 可以解决时间复杂度的问题.

通过这种方式实现, 除了 `Concat` 操作之外, 其他操作时间复杂度均为 `O(1)`, 并且空间复杂度也能达到 $O(n)$.

Problem 5

我们先证明: 无回路连通图 G 的任意两点 u, v 有且仅有一条通路.

假设 P_1 和 P_2 是 u, v 的两条不同的通路, 不妨取出一条边 $e = (x, y) \in P_1$ 但 $e \notin P_2$. 则我们知道 $G - \{e\}$ 形成的新图仍然包含着通路 P_2, 且 $x - u - P_2 - v - y$ 是一条 x 到 y 的通路. 那么对于原来的图 G 来说, $x - u - P_2 - v - y$ 通路加上边 e, 便成了一个回路, 与无回路图的原设定矛盾. 因此任意两点 u, v 有且仅有一条通路.

我们再证明: 含 n 个顶点的无回路连通图 G 的边数为 $n - 1$.

对于 G 的任意一条边 $e = (x, y)$ 来说, e 是连接 x, y 的唯一一条通路, 那么将 e 去除, x, y 便不再连通, 因此 x, y 分别所在的两个新图 G_x, G_y 也是不连通的. 并且因为没有添加新边, 还有去除一条边不连通图个数最多加一, 所以易知 G_x 和 G_y 分别也是无回路连通图.

我们用数学归纳法证明含 n 个顶点的无回路连通图 G 的边数为 $n - 1$.

Basis: 对于只有一个顶点的无回路连通图, 即 $n = 1$, 易知其没有边, 满足边数为 $n - 1$.

I.H.: 假设一个无回路连通图有 n_i 个顶点, 并且 $n_i < n$ 时, 有其边数为 $n_i - 1$.

I.S.:

对于一个 n 个顶点的无回路连通图 G, 选取任意一条边为 $e = (x, y)$, 我们将 e 去掉, 便产生了两个新的无回路连通图 G_x, G_y.

我们知道, G_x, G_y 的顶点个数 $n_x, n_y \geqslant 1$, 并且 $n = n_x + n_y$, 那么 $n_x, n_y < n$.

并且由 I.H. 可知 G_x, G_y 的边总数分别为 $n_x - 1$ 和 $n_y - 1$.

那么加上边 e 的 G 的边总数为 $(n_x - 1) + (n_y - 1) + 1 = n - 1$.

归纳成立.

我们最后证明: 无回路连通图至少有两个 1 度顶点.

因为 G 中顶点的总度数等于边数的两倍 (易证, 图 G 中每去除 1 条边, 总度数便减 2, 直到边数为 0 之后, 度数也变为 0 了), 因此 G 的总度数为 $2n - 2$.

因为 G 是连通图, 所以每个点的度数至少都为 1.

用反证法, 假设至多只有一个 1 度顶点, 其他 $n - 1$ 的顶点的度数均大于等于 2, 那么 G 的总度数为 $1 + 2(n - 1) = 2n - 1 > 2n - 2$, 矛盾.

因此无回路连通图至少有两个 1 度顶点.

Problem 6

令 $A = BB^T = (a_{ij})$, 则 $a_{ij} = \sum_{k=1}^{|E|} b_{ik} b_{jk}$

可以看出 a_{ij} 数学表达式的意义是, 对于顶点 i 和顶点 j 来说 ($i \neq j$), 遍历每一条边 e, 有两种情况, 一种是没有相连, 则赋值 0; 一种是相连, 则一个是入边一个是出边, 则值为 -1. 最后求和.

于是 a_{ij} 的意义是,

当 $i \neq j$ 时, a_{ij} 是连接顶点 i 和顶点 j 的边总数的负数;

当 $i = j$ 时, a_{ii} 是顶点 i 连接到的边的总数.