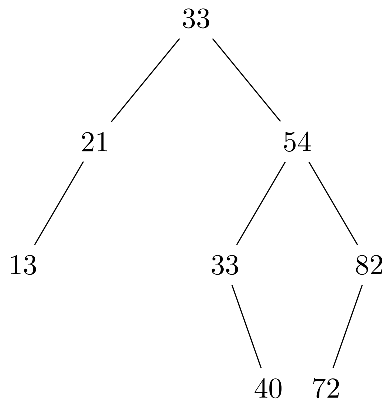
# 并行计算作业 2 & 3

傅申 PB20000051

**2: Ex6.3.**

假设总是编号较小的处理器写入变量, 则二叉树为:




- $P_0$ 能成功写入 $root$, 因此树根为 33.
  - $P_1$ 能成功写入 $LC_0$, 因此 31 为根的左节点. $P_0$ 的左子树中还有 [13].
    - $P_2$ 写入 $LC_1$, 因此 13 为 31 的左节点.
  - $P_3$ 能成功写入 $RC_0$, 因此 54 为根的右节点. $P_0$ 的右子树里还有 [82, 33, 40, 72].
    - $P_5$ 能成功写入 $LC_3$, 因此 33 为 54 的左节点. $P_3$ 的左子树里还有 [40].
      - $P_6$ 写入 $RC_5$, 因此 40 为 33 的右节点.
    - $P_4$ 能成功写入 $RC_3$, 因此 82 为 54 的右节点. $P_3$ 的右子树里还有 [72].
      - $P_7$ 写入 $LC_4$, 因此 72 为 82 的左节点.

**2: 二叉树并行转有序数组.**

采用类似顺序统计量的做法. 算法见下一页, 各个部分的功能为:

(1) 初始化, $O(1)$.
  (1.1) 初始化 size 数组 `s`, 其中叶节点初始化为 1, 否则为 0;
  (1.2) 初始化 rank 数组 `r`, `r[root] = -2` 是为了使 (4) 能从根节点开始.
  (1.3) `s[n + 1] = 0` 是为了方便 (2.1) 中的赋值;
  (1.4) `r[n + 1]` 是冗余量, 方便 (3.2.2) 中的赋值.
(2) 从下到上计算各个节点的 size `s[i]`, 直到根节点的 size 更新, $O(\log n)$.
  (2.1) 若节点的 size 为 0 且孩子节点的 size 已更新, 则更新节点 size.
(3) 从上到下计算各个节点的 rank `r[i]`, $O(\log n)$.
  (3.1) 初始化 `updated`, 它是该轮迭代中是否有 `r[i]` 被更新的标记;
  (3.2) 如果 `r[i] = -2`, 则它的父节点在上一轮迭代中被更新;
    (3.2.1) 计算并更新 `r[i]`;
    (3.2.2) 标记其子节点为 -2 并更新 `updated`, 其中 `r[n + 1]` 作为冗余量永远不会被处理, 因此不用考虑没有子节点的情况.
(4) 按照 `r[i]` 的值将排序好的数组写入 `B` 中, $O(1)$.

总时间为 $O(\log n)$. 因为存在同时读写的情况, 所以模型为 PRAM-CRCW.

```
begin
  (1) for each processor i par-do
      (1.1) if LC[i] = RC[i] = n + 1 then s[i] = 1 else s[i] = 0 end if
      (1.2) if i = root then r[i] = -2 else r[i] = -1 end if
      (1.3) s[n + 1] = 0
      (1.4) r[n + 1] = 0
      end for
  (2) repeat for each processor i do
      (2.1) if s[i] = 0
                and (LC[i] = n + 1 or s[LC[i]] ≠ 0)
                and (RC[i] = n + 1 or s[RC[i]] ≠ 0) then
              s[i] = 1 + s[LC[i]] + s[RC[i]]
            end if
      until s[root] ≠ 0
  (3) repeat for each processor i do
      (3.1) updated = false
      (3.2) if r[i] = -2 then
            (3.2.1) if i = root then
                      r[i] = s[LC[i]]
                    else if i = LC[f[i]] then
                      r[i] = r[f[i]] - s[i] + s[RC[i]]
                    else
                      r[i] = r[f[i]] + s[LC[i]] + 1
                    end if
            (3.2.2) r[LC[i]] = -2
                    r[RC[i]] = -2
                    updated = true
            end if
      until updated = false
  (4) for each processor i par-do
        B[r[i]] = A[i]
      end for
end
```

**3: Activity 11.**

```
begin
  for each processor i < n par-do
    if A[i] = 0 and A[i + 1] = 1 then
      index = i + 1
    end if
  end for
end
```

**3: 7.3.**

(1) 总共 $O(\log m + n)$:

  (1) 显然时间是 $O(1)$ 的;

  (2) 求 $rank$ 可以使用二分查找, $O(\log n)$;

  (3) 将数据拷贝存储, 时间为 $O(\log m + n)$.

(2) 算法执行得到: $rank(b_4 : A) = 3$, $rank(b_8 : A) = 6$, $rank(b_{12} : A) = 10$. 划分得:

| $B$ | 3, 4, 5, 6 | 8, 10, 12, 13 | 14, 15, 20, 21 | 22, 26, 29, 31 |
|---|---|---|---|---|
| $A$ | 0, 1, 2 | 7, 9, 11 | 16, 17, 18, 19 | 23, 24, 25, 27, 28, 30, 33, 34 |
| 归并 | 0, 1, 2, 3, 4, 5, 6 | 7, 8, 9, 10, 11, 12, 13 | 14, 15, 16, 17, 18, 19, 20, 21 | 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33, 34 |

最后归并得到 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33, 34).

**3: 7.6.**

(1) $W(n) = O(n)$
   (1) $O(n)$
   (2) $\sum_{h=1}^{\log n} O\left(\frac{n}{2^h}\right) = O(n)$
   (3) $\sum_{h=0}^{\log n} O\left(\frac{n}{2^h}\right) = O(n)$

(2) 如下, 其中计算的是前缀"和"
   (1) 初始化, $B$ 如下:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

   (2) 正向遍历, $B$ 如下:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 3 | 7 | 11 | 15 | | | | |
| 10 | 26 | | | | | | |
| 36 | | | | | | | |

   (3) 反向遍历, $C$ 如下:

| 1 | 3 | 6 | 10 | 15 | 21 | 28 | 36 |
|---|---|---|---|---|---|---|---|
| 3 | 10 | 21 | 36 | | | | |
| 10 | 36 | | | | | | |
| 36 | | | | | | | |

   第一行 $C(0)$ 就是前缀和.