1. 顺序查找的时间复杂 O(n)，二分查找的时间复杂度为 O(logn)，将 n = 1000000 代入比值为 1000000 / $\log_2(1000000)$ = 50171.665944 倍

考虑平均比较次数

顺序查找：$\frac{n+1}{2}$

折半查找：$\frac{n+1}{\mathrm{n}}\log_2(n+1)-1$

带入 n = 1000000，结果为：26410.91 倍

差异分析：在时间复杂度表示中忽略了系数，因此相差几乎两倍

2. 运送士兵过河

(1) 方法，令两个小男孩为 a、b，过河士兵为 $s_1, s_2, ..., s_n$

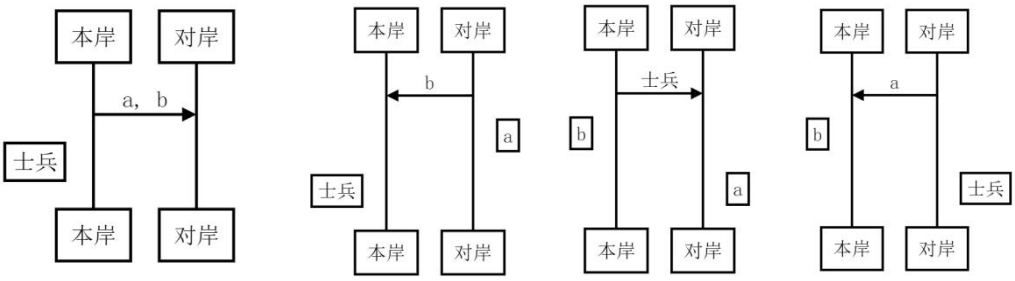
① 首先让小男孩 a、b 过河，选择任意一个小男孩留在对岸，这里不妨选择 a

② 然后 b 回到本岸下船

③ 让士兵 s_i(1 <= i <= n)过河到对岸，士兵 s_i 下船

④ 然后小男孩 a 坐船回到本岸

示意图如下：



(2) 综上，运送一个士兵需要四次岸到岸的横渡，于是对于 N 个士兵，需要 4N 次横渡

3. 遍历顺序

(1) 前序遍历 abdecf

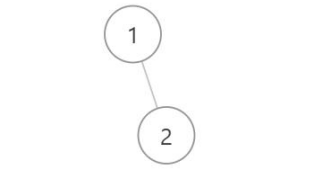
(2) 中序遍历 dbeacf

(3) 后序遍历 debfca

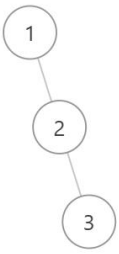
4. 构造 AVL 树

(1) 123456

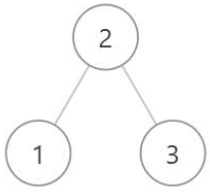
① 插入 1

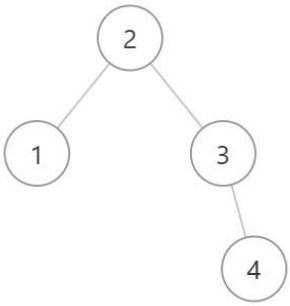


② 插入 2

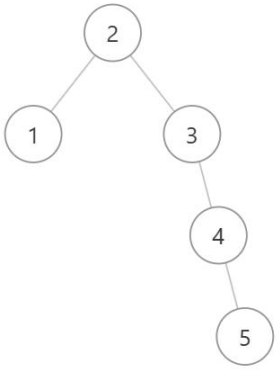


③ 插入 3 需要调整为

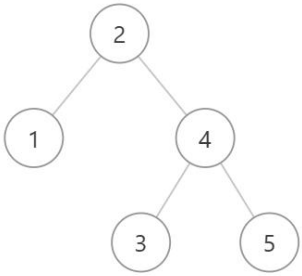


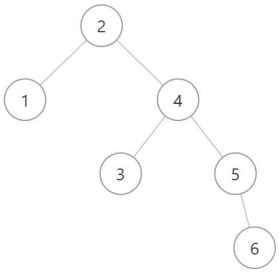


④ 插入 4

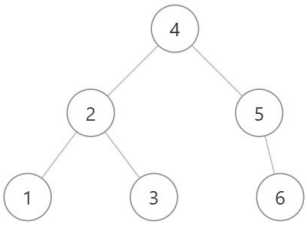


⑤ 插入 5 需要调整为



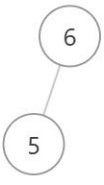


⑥ 插入 6 需要调整为



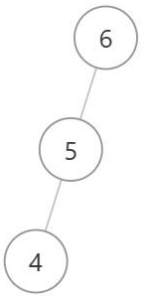
(2) 654321

① 插入 6

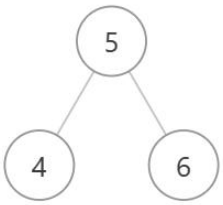


② 插入 5

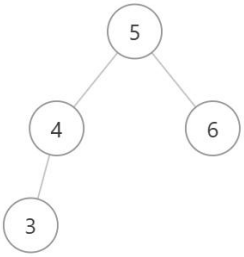
③ 插入 4



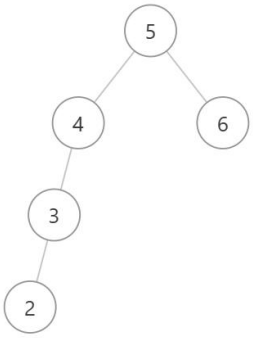
需要调整为



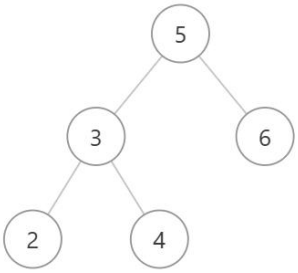
④ 插入 3



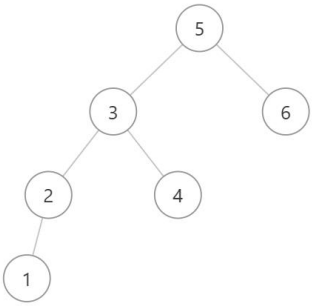
⑤ 插入 2



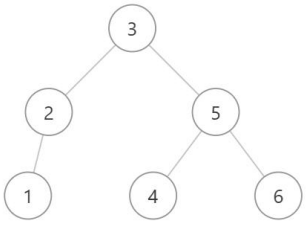
需要调整为



⑥ 插入 1



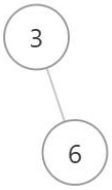
需要调整为

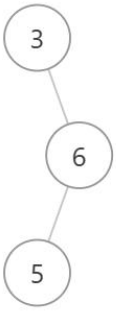


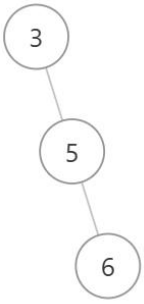
(3) 365124

① 插入 3

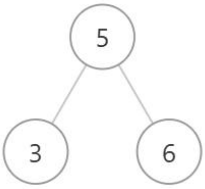
② 插入 6



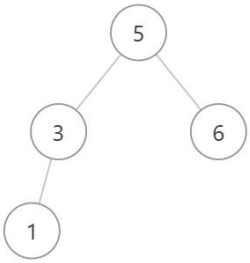




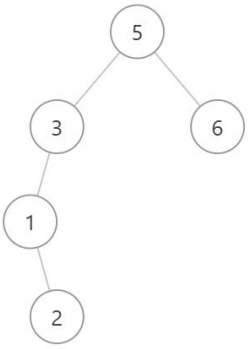
③ 插入 5　　调整为



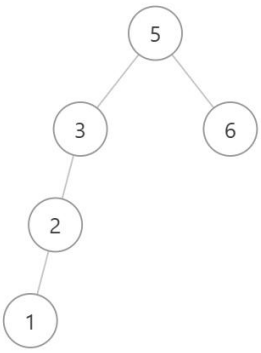
再调整为

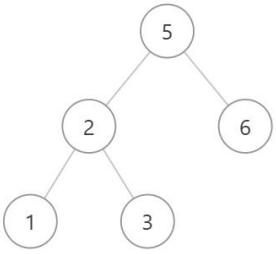


④ 插入 1

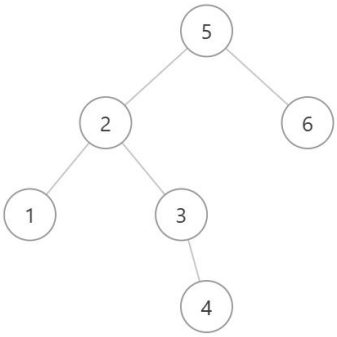


⑤ 插入 2　　调整为

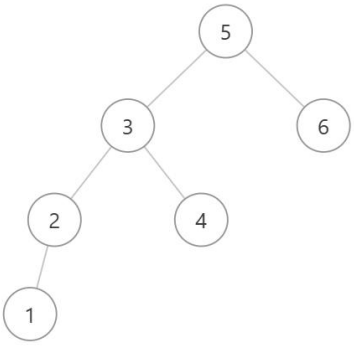


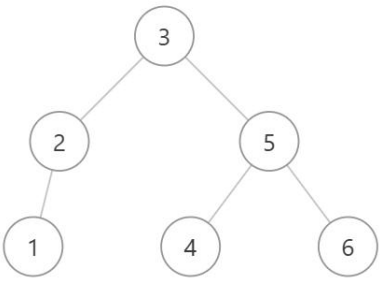


再调整为



⑥ 插入 4 调整为

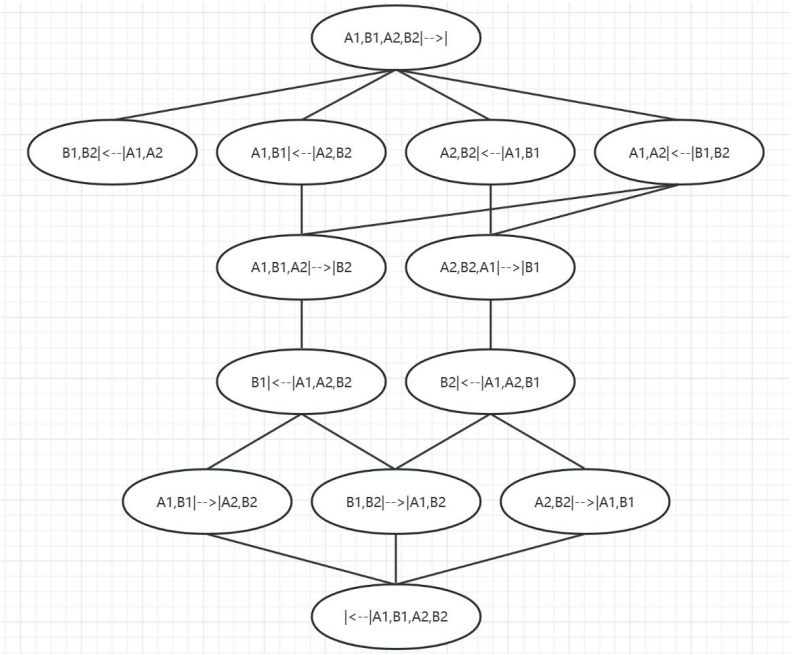




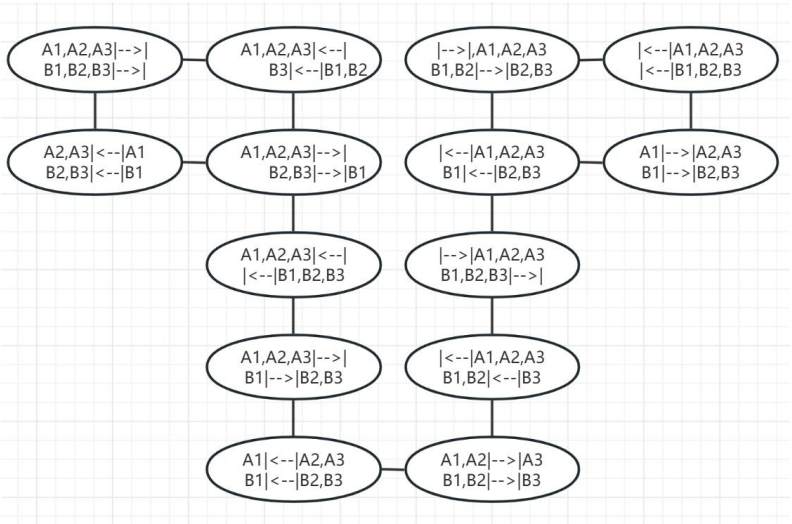
再调整为

5. 爱吃醋的丈夫

(1) n = 2，不妨令丈夫为 A_i，妻子为 B_i，共需要 5 次渡河状态图如下：

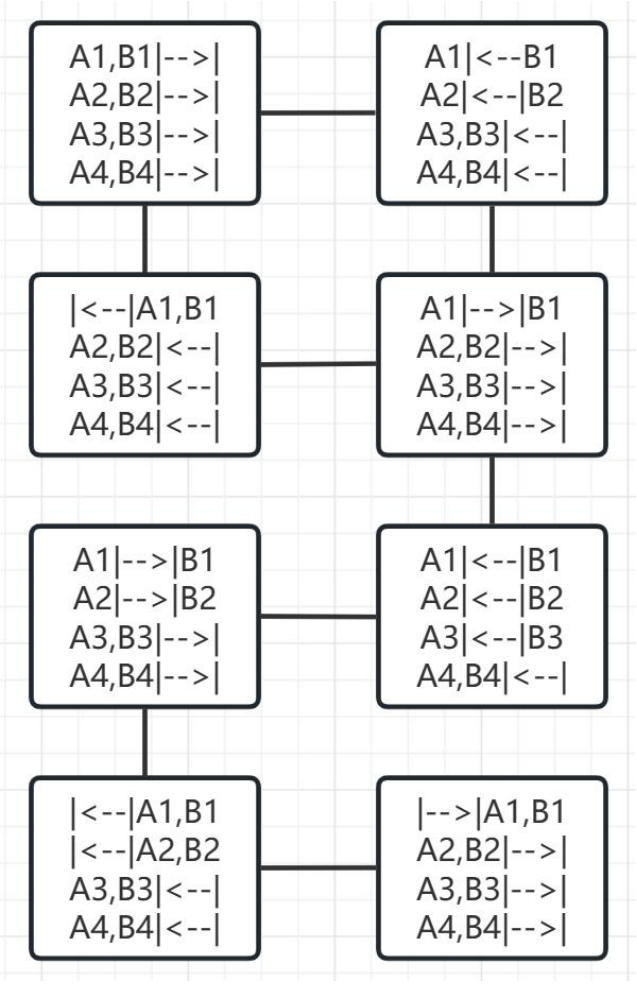


(2) n = 3，不妨令丈夫为 A_i，妻子为 B_i，共需要 11 次渡河状态图如下：



(1) n = 4,该问题 n≥4 时无解。在上一小问 3 对夫妻的基础上，从 1 个或多个额外的夫妇开始，将不会产生新的状态，并且在六次过河之后，将到达 n-1 对夫妇和船在初始河岸，有一对夫妇在另一岸的状态，从该状态唯一允许的变化是将该夫妇运送到另一岸而回到之前的状态。

状态图如下：



7.

(1) 算法思路

① 从矩阵的右上角即 matrix[0][n-1]开始搜索，每一轮搜索中当前位置为 matrix[i][j]

② 若 matrix[i][j] == target，说明搜索成功

③ 若 matrix[i][j] < target，说明该 target 大于该行左侧的所有数字，忽略，则执行 i <- i + 1，到下一行

④ 若 matrix[i][j] > target，说明 target 小于该列上侧的所有数字，忽略，则执行 j <- j - 1，到左边一列

⑤ 重复上述步骤直到到达边界为止

(2) 时间复杂度分析

① 输入规模：矩阵行数 m、矩阵列数 n

② 基本操作：比较

③ 检查

1) 最好情况：target == matrix[0][n - 1]，时间复杂度为 O(1)

2) 最坏情况：i 增加 m 次，j 减少 n 次，时间复杂度为 O(m + n)

④ 时间复杂度为 O(m + n)

(3) 空间复杂度分析

① 使用常数个数的变量，空间复杂度为 O(1)

(4) 补充的函数

```
bool searchMatrix(vector<vector<int> >& matrix, int target){
    //TODO
    // 获得矩阵的维度
    int m = matrix.size();
    int n = matrix[0].size();
    // 从左上角开始
    int i = 0;
    int j = n - 1;
    // 保证不越界
    while (i < m && i >= 0 && j < n && j >= 0) {
        // 正好是要找的
        if (matrix[i][j] == target)
            return true;
        // 第i行的最大都比target小说明整行都比target小
        // 移动到下一行寻找
        else if (matrix[i][j] < target)
            ++i;
        // 当前比target更大，向左找更小的
        else
            --j;
    }
    // 找不到
    return false;
}
```

(5) 运行截图