Description | Editorial | Solutions (1.2K) | Submissions

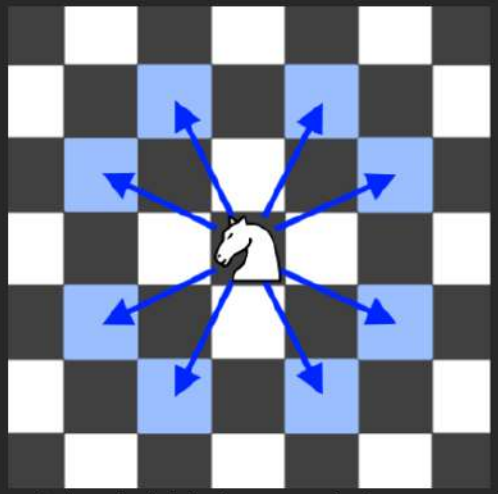
688. Knight Probability in Chessboard

Medium 3.3K 420

Companies

On an `n x n` chessboard, a knight starts at the cell `(row, column)` and attempts to make exactly `k` moves. The rows and columns are **0-indexed**, so the top-left cell is `(0, 0)`, and the bottom-right cell is `(n - 1, n - 1)`.

A chess knight has eight possible moves it can make, as illustrated below. Each move is two cells in a cardinal direction, then one cell in an orthogonal direction.



Each time the knight is to move, it chooses one of eight possible moves uniformly at random (even if the piece would go off the chessboard) and moves there.

The knight continues moving until it has made exactly `k` moves or has moved off the chessboard.

Return *the probability that the knight remains on the board after it has stopped moving.*

Java | Auto

```java
public class Solution {
    double[][][] dp;
    int[] xmove = {-2, -2, -1, -1, 1, 1, 2, 2};
    int[] ymove = {-1, 1, -2, 2, -2, 2, -1, 1};

    public double knightProbability(int n, int k, int row, int col) {
        dp = new double[n][n][k + 1];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                Arrays.fill(dp[i][j], -1.0);
            }
        }
        double favourableOutcome = solve(row, col, n, k);
        double totalOutcome = Math.pow(8, k);
        return favourableOutcome / totalOutcome;
    }

    private double solve(int row, int col, int n, int k) {
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

n =
3

k =
2

Console | Run | Submit

Description | Editorial | Solutions (7.9K) | Submissions

56. Merge Intervals

Medium 19.9K 674

Companies

Given an array of `intervals` where `intervals[i] = [start_i, end_i]`, merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input.*

Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

Example 2:

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

Constraints:

- `1 <= intervals.length <= 10^4`
- `intervals[i].length == 2`
- `0 <= start <= end <= 10^4`

Java | Auto

```java
class Solution {
    public int[][] merge(int[][] intervals) {
        if (intervals == null || intervals.length == 0) {
            return new int[0][];
        }

        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));

        List<int[]> merged = new ArrayList<>();
        int[] mergedInterval = intervals[0];

        for (int i = 1; i < intervals.length; i++) {
            int[] interval = intervals[i];

            if (interval[0] <= mergedInterval[1]) {
                mergedInterval[1] = Math.max(mergedInterval[1], interval[1]);
            } else {
                merged.add(mergedInterval);
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

```
intervals =
[[1,3],[2,6],[8,10],[15,18]]
```

Output

```
[[1,6],[8,10],[15,18]]
```

Console | Run | Submit