52. N-Queens II

It’s similar to N-QueenI and we will use dfs to iterate all possible combination and use self.result to record it. TC is O(n**n)

```
class Solution:
    def totalNQueens(self, n: int) -> int:
        cols = set()
        dia1 = set()
        dia2 = set()
        self.res = 0
        def dfs(i):
            if i > n:
                return
            if i == n:
                self.res += 1
            for j in range(0, n):
                if j not in cols and i + j not in dia1 and i - j not in dia2:
                    cols.add(j)
                    dia1.add(i + j)
                    dia2.add(i - j)
                    dfs(i + 1)
                    cols.remove(j)
                    dia1.remove(i + j)
                    dia2.remove(i - j)
        dfs(0)
        return self.res
```

1207. Unique Number of Occurrences

```
from collections import Counter
class Solution:
    def uniqueOccurrences(self, arr: List[int]) -> bool:
        counter = Counter(arr)
        vals = counter.values()
        return len(vals) == len(set(vals))
```

1208. Get Equal Substrings Within Budget

```
class Solution:
    def equalSubstring(self, s: str, t: str, maxCost: int) -> int:
        arr = []
        length = len(s)
        for i in range(length):
            arr.append(abs(ord(s[i]) - ord(t[i])))
        l, r, cur_sum, max_len = 0, 0, 0, 0
        while r < length:
            while r < length and cur_sum <= maxCost:
                cur_sum += arr[r]
                if cur_sum > maxCost:
                    break
                else:
                    r += 1
            max_len = max(max_len, r - l)
            while l <= r and cur_sum > maxCost:
                cur_sum -= arr[l]
                l += 1
            r += 1
        return max_len
```

1209. Remove All Adjacent Duplicates in String II

```
class Solution:
    def removeDuplicates(self, s: str, k: int) -> str:
        table = [i * k for i in 'abcdefghijklmnopqrstuvwxyz']
        while True:
            mark = s
            for w in table:
                s = s.replace(w, '')
            if mark == s:
                return s
```

1210. Minimum Moves to Reach Target with Rotations

```
class Solution:
    def minimumMoves(self, grid: List[List[int]]) -> int:
        n = len(grid)
        count = 0
        cur = [(0, 0, 0, 1, 1)]
        visited = set()
        visited.add((0, 0, 0, 1))
        while cur:
            next_ite = []
            for x1, y1, x2, y2, dire in cur:
                if dire == 1:
                    if x1 == n - 1 and y1 == n - 2 and x2 == n - 1 and y2 == n - 1:
                        return count
                    if y2 + 1 < n and grid[x1][y2 + 1] == 0 and (x2, y2, x1, y2 + 1) not in visited:
                        next_ite.append((x2, y2, x1, y2 + 1, 1))
                        visited.add((x2, y2, x1, y2 + 1, 1))
                    if x1 + 1 < n and grid[x1 + 1][y1] == 0 and grid[x2 + 1][y2] == 0:
                        if (x1, y1, x1 + 1, y1) not in visited:
```

```
                    visited.add((x1, y1, x1 + 1, y1))
                    next_ite.append((x1, y1, x1 + 1, y1, 0))

                if (x1 + 1, y1, x2 + 1, y2) not in visited:
                    visited.add((x1 + 1, y1, x2 + 1, y2))
                    next_ite.append((x1 + 1, y1, x2 + 1, y2, 1))
        else:
            if x2 + 1 < n and grid[x2 + 1][y2] == 0 and (x2, y2, x2 + 1, y2) not in visited:
                visited.add((x2, y2, x2 + 1, y2))
                next_ite.append((x2, y2, x2 + 1, y2, 0))
            if y1 + 1 < n and grid[x1][y1 + 1] == 0 and grid[x2][y2 + 1] == 0:
                if (x1, y1, x1, y1 + 1) not in visited:
                    visited.add((x1, y1, x1, y1 + 1))
                    next_ite.append((x1, y1, x1, y1 + 1, 1))

                if (x1, y1 + 1, x2, y2 + 1) not in visited:
                    visited.add((x1, y1 + 1, x2, y2 + 1))
                    next_ite.append((x1, y1 + 1, x2, y2 + 1, 0))
    cur = next_ite
    count += 1
return -1
```