1. Backtracking Notes

Here are all the solutions to the Backtracking Algorithms problems so far:

#backtracking #dfs #bruteforce #combinations #permutations

Introduction

Backtracking

Backtracking is a powerful problem-solving strategy that allows us to systematically explore all possible solutions to a problem by following a brute-force approach. It is particularly useful when dealing with problems that have multiple valid solutions, unlike dynamic programming, which primarily focuses on optimization problems. In this note, we will delve into the concept of backtracking and its application through a simple example.

Backtracking Overview

1. **Brute-Force Approach**: Backtracking employs a brute-force approach, which means that it explores all possible solutions to a given problem and selects the desired solutions.
2. **Multiple Solutions**: Backtracking is employed when a problem has multiple valid solutions, and we aim to find and enumerate all of these solutions.
3. **State Space Tree**: Backtracking problems can be visualized as a state space tree, also known as a solution tree. This tree represents all possible states and decisions leading to a solution.

Example: Arranging Students in Chairs

To illustrate the backtracking strategy, let's consider a simple example: arranging three students (two boys and one girl) in three chairs. We want to find all possible arrangements.

Solution Tree Generation

1. **Initial State**: Start with the initial state, indicating that we haven't made any decisions yet.
2. **Exploring Possibilities**: Begin by considering each student for the first chair.
    - First Chair: Boy 1
        - Second Chair: Boy 2
            - Third Chair: Girl 1
            - *(Solution 1)*
        - *(Solution 2)*
    - Second Chair: Girl 1
        - Third Chair: Boy 2
            - *(Solution 3)*
        - *(Solution 4)*
    - Third Chair: Boy 2
        - Second Chair: Girl 1
            - *(Solution 5)*
            - *(Solution 6)*

Finding All Possible Arrangements

By systematically exploring all possibilities in the state space tree, we find all six possible arrangements of the students. Each path from the root of the tree to a leaf node represents a valid arrangement. We have successfully solved the problem using backtracking, as it allowed us to enumerate all solutions.

Introducing Constraints

Backtracking also handles constraints effectively. For example, if we impose the constraint that the girl should not sit in the middle chair, we can modify our approach.

Constraint Implementation

1. **Initial State**: Start with the initial state.
2. **Exploring Possibilities**: Begin by considering each student for the first chair.
    - First Chair: Boy 1
        - Second Chair: Boy 2
            - *(Solution 1)*
        - Second Chair: Girl 1 *(Constraint Violated)*

- Second Chair: Girl 1 *(Constraint Violated)*
- Third Chair: Boy 2
  - Second Chair: Girl 1
    - *(Solution 2)*

By applying the bounding function (in this case, the constraint), we can efficiently eliminate invalid solutions and significantly reduce the search space.

Conclusion

Backtracking is a versatile problem-solving strategy that explores all possible solutions to a problem. It utilizes state space trees to visualize the search process, making it an effective approach for scenarios with multiple valid solutions and constraints. This note has provided an introductory understanding of backtracking, setting the stage for further exploration of more complex problems that can be solved using this strategy.

Coding Solutions

1. Permutations I

#Medium

Given an array `nums` of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

Input: `nums = [1,2,3]`
Output: `[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]`

Example 2:

Input: `nums = [0,1]`
Output: `[[0,1],[1,0]]`

Example 3:

Input: `nums = [1]`
Output: `[[1]]`

Constraints:

- `1 <= nums.length <= 6`
- `-10 <= nums[i] <= 10`
- All the integers of `nums` are **unique**.

Code

```python
def permute(nums):
  result = []
  #base case
  if len(nums) == 1:
    return [nums[:]]
  for _ in nums:
    n = nums.pop(0)
    perms = permute(nums)
    for perm in perms:
      perm.append(n)
    result.extend(perms)
    nums.append(n)
  return result
if __name__ == "__main__":
  nums = [1,2,3]
  print(permute(nums))
```

Explanation

```
nums = [1,2,3]
print(permute([1,2,3]))
def permute([1,2,3]):
        result=[]
        len([1,2,3])!=1
        for _ in [1,2,3]:
                #iter 1
                n = [1,2,3].pop(1) ➡️ n = 1 nums = [2,3]
                perms = permute([2,3])
                # recursion 1
                        result=[]
                        len([2,3])!=1 #ignore
                        for _ in [2,3]:
                                # iter 1
                                n = [2,3].pop(2) ➡️ n = 2, nums = [3]
                                perms = permute([3])
                                        #recursion (1,1)
                                        result=[]
                                        len([3])==1:
                                                return [[3]] ➡️ perms = [[3]] # recursion ends
                                for [3] in [[3]]:
                                        [3].append(2)➡️ [3,2]
                                        ➡️ perms = [[3,2]]
                                [].extend([[3,2]])➡️ result = [[3,2]]
                                [3].append(2) ➡️ nums = [3,2]

                                # iter 2
                                n = [3,2].pop(3) ➡️ n = 3, nums=[2]
                                perms = permute([2])
                                # recursion (1,2)
                                        result = []
                                        len([2])==1:
                                                return [[2]] ➡️ perms = [[2]] # recursion ends
                                for [2] in [[2]]:
                                        [2].append(3) ➡️ [2,3]
                                        ➡️ perms = [[2,3]]
                                [[3,2]].extend([[2,3]]) ➡️ result = [[3,2],[2,3]]
                                [2].append(3)➡️ nums = [2,3]
                                # for loop ends
                        return [[3,2],[2,3]] ➡️ perms = [[3,2],[2,3]] # recursion 1 ends
        for [3,2] and [2,3] in [[3,2],[2,3]]:
                [3,2].append(1) ➡️ [3,2,1] # iter 1
                [2,3].append(1) ➡️ [2,3,1] # iter 2
                ➡️ perms = [[3,2,1],[2,3,1]]
        [].extend([[3,2,1],[2,3,1]]) ➡️ result = [[3,2,1],[2,3,1]]
        [2,3].append(1) ➡️ nums = [2,3,1]

# iter 2
        n = [2,3,1].pop(2) ➡️ n = 2, nums ➡️ [3,1]
        perms = permute([3,1])
                # recursion 2
                result = []
                len([3,1])!=1 # ignore
                for _ in [3,1]:
                        # iter 1
                        n = [3,1].pop(3) ➡️ n = 3 ➡️ nums = [1]
                        perms = permute([3])
                        # recursion 2_1
                                result = []
                                len([1])==1:
                                        return [[1]] ➡️ perms = [[1]] # recursion ends
                        for [1] in [[1]]:
                                [1].append(3) ➡️ [1,3]
                                ➡️ perms = [[1,3]]
                        [].extend([[1,3]]) ➡️ result = [[1,3]]
                        [1].append(3) ➡️ nums = [1,3]
                        # iter 2
                        n = [1,3].pop(1) ➡️ n = 1, nums = [3]
                        perms = permute([3])
                        # recursion 2_2
                                result = []
                                len([3]) == 1:
                                        return [[3]] ➡️ perms = [[3]] # recursion ends
                        for [3] in [[3]]:
                                [3].append(1)
                                ➡️ perms = [[3,1]]
```

```
                [[1,3]].extend([[3,1]]) ➡️ result = [[1,3],[3,1]]
                        [3].append(1) ➡️ nums = [3,1]
                        # for loop ends
        for [1,3] and [3,1] in [[1,3],[3,1]]:
                [1,3].append(2) ➡️ [1,3,2]
                [3,1].append(2) ➡️ [3,1,2]
                ➡️ perms = [[1,3,2],[3,1,2]]
        [[3,2,1],[2,3,1]].extend([[1,3,2],[3,1,2]])
        ➡️ result = [[3,2,1],[2,3,1],[1,3,2],[3,1,2]]
        [3,1].append(2) ➡️ nums = [3,1,2]

# iter 3
        n = [3,1,2].pop(2) ➡️ n = 3, nums = [1,2]
        perms = permute([1,2])
        # recursion 3
                result = []
                len([1,2]) != 1  # Continue
                for _ in [1,2]:
                    # Iteration 1
                    n = [1,2].pop(1)  ➡️ n = 1, nums = [2]
                    perms = permute([2])
                    # Recursion (3,1)
                    result = []
                    len([2]) == 1:
                        return [[2]]  # Recursion ends
                    for [2] in [[2]]:
                        [2].append(1)  ➡️ [2,1]
                        ➡️ perms = [[2,1]]
                    [].extend([[2,1]])  ➡️ result = [[2,1]]
                    [2].append(1)  ➡️ nums = [2,1]
                    # Iteration 2
                    n = [2,1].pop(2)  ➡️ n = 2, nums = [1]
                    perms = permute([1])
                    # Recursion (3,2)
                    result = []
                    len([1]) == 1:
                        return [[1]]  ➡️ Recursion ends
                    for [1] in [[1]]:
                        [1].append(2)  ➡️ [1,2]
                        ➡️ perms = [[1,2]]
                    [[2,1]].extend([[1,2]])  ➡️ result = [[2,1],[1,2]]
                    [1].append(2)  ➡️ nums = [1,2]
                    # For loop ends
                return [[2,1],[1,2]]  ➡️ perms = [[2,1],[1,2]]  # Recursion 3 ends
        for [2,1] and [1,2] in [[2,1],[1,2]]:
                [2,1].append(3) ➡️ [2,1,3] #iter 1
                [1,2].append(3) ➡️ [1,2,3] #iter 2
                ➡️ perms [[2,1,3],[1,2,3]]
        [[3,2,1],[2,3,1],[1,3,2],[3,1,2]].extend([[2,1,3],[1,2,3]])
        ➡️ result = [[3,2,1],[2,3,1],[1,3,2],[3,1,2],[2,1,3],[1,2,3]]
        [1,2].append(3) ➡️ nums = [1,2,3]
        # for loop ends
                return [[3,2,1],[2,3,1],[1,3,2],[3,1,2],[2,1,3],[1,2,3]]
```

2. Permutations II

#Medium

Given a collection of numbers, `nums`, that might contain duplicates, return *all possible unique permutations* ***in any order***.

Example 1:

Input: `nums = [1,1,2]`
Output:
`[[1,1,2],[1,2,1], [2,1,1]]`
Example 2:

Input: `nums = [1,2,3]`
Output: `[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]`

Constraints:

- `1 <= nums.length <= 8`

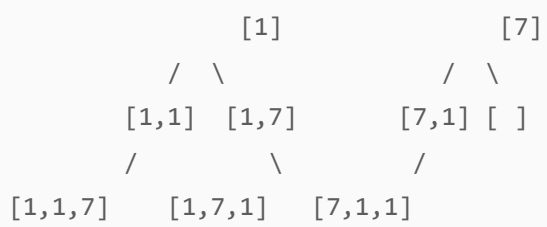
- `-10 <= nums[i] <= 10`

Code

```
def main(arr):
  res = []
  perm = []
  ans={}
  for i in arr:
      if i not in ans:
          ans[i]=1
      else:
          ans[i]+=1
  def dfs():
      if len(arr)==len(perm):
          res.append(perm[:])
          return
      for n in ans:
          if ans[n]>0:
              perm.append(n)
              ans[n]-=1
              dfs()
              ans[n]+=1
              perm.pop()
      return res
  res = dfs()
  return res

print(main([1,1,7]))
```

Explanation




```
def main([1, 1, 7]):
        res = []
        perm = []
        ans = {}
        for i in [1, 1, 7]:
                1 not in {}: --> True
                        ans[1]=1 >> {1: 1}
                else:
                        ans[1]+=1 >> {1: 2}
                7 not in {1: 2}: --> True
                        ans[7]=1 >> {1: 2, 7: 1}
        def dfs():
                if len([1, 1, 7])==len([]): --> False
                for n in {1: 2, 7: 1}:
                        if ans[1]>0: --> True
                                [].append(1) >> [1]
                                ans[1]-=1 >> {1: 1, 7: 1}
#                               🟢Recursive block starts 🟢
                                dfs()
                if len([1, 1, 7])==len([1]): --> False
                for n in {1: 1, 7: 1}:
                        if ans[1]>0: --> True
                                [1].append(1) >> [1, 1]
                                ans[1]-=1 >> {1: 0, 7: 1}
#                               🟢Recursive block starts 🟢
                                dfs()
                if len([1, 1, 7])==len([1, 1]): --> False
                for n in {1: 0, 7: 1}:
                        if ans[1]>0: --> False
                        if ans[7]>0: --> True
```

```
                                [1, 1].append(7) >> [1, 1, 7]
                                ans[7]-=1 >> {1: 0, 7: 0}
#                               🟢Recursive block starts 🟢
                                dfs()
                        if len([1, 1, 7])==len([1, 1, 7]): --> True
                                res.append([1, 1, 7]) >> [[1, 1, 7]]
                                return
                                🟡 #Base Case 🟡
                                        # 🔴 Recursive block ends 🔴
                                ans[7]+=1 >> {1: 0, 7: 1}
                                [1, 1, 7].pop() >> [1, 1]
                        return [[1, 1, 7]]
                        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁
                                        # 🔴 Recursive block ends 🔴
                                ans[1]+=1 >> {1: 1, 7: 1}
                                [1, 1].pop() >> [1]
                                if ans[7]>0: --> True
                                        [1].append(7) >> [1, 7]
                                        ans[7]-=1 >> {1: 1, 7: 0}
#                                       🟢Recursive block starts 🟢
                                        dfs()
                        if len([1, 1, 7])==len([1, 7]): --> False
                        for n in {1: 1, 7: 0}:
                                if ans[1]>0: --> True
                                        [1, 7].append(1) >> [1, 7, 1]
                                        ans[1]-=1 >> {1: 0, 7: 0}
#                                       🟢Recursive block starts 🟢
                                        dfs()
                        if len([1, 1, 7])==len([1, 7, 1]): --> True
                                res.append([1, 7, 1]) >> [[1, 1, 7], [1, 7, 1]]
                                return
                                🟡 #Base Case 🟡
                                        # 🔴 Recursive block ends 🔴
                                ans[1]+=1 >> {1: 1, 7: 0}
                                [1, 7, 1].pop() >> [1, 7]
                                if ans[7]>0: --> False
                        return [[1, 1, 7], [1, 7, 1]]
                        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁
                                        # 🔴 Recursive block ends 🔴
                                ans[7]+=1 >> {1: 1, 7: 1}
                                [1, 7].pop() >> [1]
                        return [[1, 1, 7], [1, 7, 1]]
                        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁
                                        # 🔴 Recursive block ends 🔴
                                ans[1]+=1 >> {1: 2, 7: 1}
                                [1].pop() >> []
                                if ans[7]>0: --> True
                                        [].append(7) >> [7]
                                        ans[7]-=1 >> {1: 2, 7: 0}
#                                       🟢Recursive block starts 🟢
                                        dfs()
                        if len([1, 1, 7])==len([7]): --> False
                        for n in {1: 2, 7: 0}:
                                if ans[1]>0: --> True
                                        [7].append(1) >> [7, 1]
                                        ans[1]-=1 >> {1: 1, 7: 0}
#                                       🟢Recursive block starts 🟢
                                        dfs()
                        if len([1, 1, 7])==len([7, 1]): --> False
                        for n in {1: 1, 7: 0}:
                                if ans[1]>0: --> True
                                        [7, 1].append(1) >> [7, 1, 1]
                                        ans[1]-=1 >> {1: 0, 7: 0}
#                                       🟢Recursive block starts 🟢
                                        dfs()
                        if len([1, 1, 7])==len([7, 1, 1]): --> True
                                res.append([7, 1, 1]) >> [[1, 1, 7], [1, 7, 1], [7, 1, 1]]
                                return
                                🟡 #Base Case 🟡
                                        # 🔴 Recursive block ends 🔴
                                ans[1]+=1 >> {1: 1, 7: 0}
                                [7, 1, 1].pop() >> [7, 1]
                                if ans[7]>0: --> False
                        return [[1, 1, 7], [1, 7, 1], [7, 1, 1]]
                        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁
                                        # 🔴 Recursive block ends 🔴
```

```
                                ans[1]+=1 >> {1: 2, 7: 0}
                                [7, 1].pop() >> [7]
                        if ans[7]>0: --> False
                return [[1, 1, 7], [1, 7, 1], [7, 1, 1]]
                🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁
                                 # 🔴 Recursive block ends 🔴
                                ans[7]+=1 >> {1: 2, 7: 1}
                                [7].pop() >> []
                return [[1, 1, 7], [1, 7, 1], [7, 1, 1]]
                🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁
#final output
>> [[1, 1, 7], [1, 7, 1], [7, 1, 1]]
```

3. Subsets

#Medium

Given an integer array `nums` of **unique** elements, return *all possible*

subsets

(the power set).

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

Input: nums = `[1,2,3]`
Output: `[[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]`

Example 2:

Input: nums = `[0]`
Output: `[[],[0]]`

Constraints:

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`
- All the numbers of `nums` are **unique**.

Code

```
arr=[1,2,3]
def dfs(i, res, subs):
        res.append(subs[:])
        for i in range(i,len(arr)):
                subs.append(arr[i])
                dfs(i+1,res,subs)
                subs.pop()
        return res
print(dfs(0,[],[]))
```

Explanation

```
arr=[1, 2, 3]
print(dfs(0,[],[]))

dfs(i = 0,res = [], subs = []):
        [].append([]) >> res = [[]]
        for i in range(0, 3):
                [].append(1) >> subs=[1]
                dfs(0+1,[[]], [1])
                 #🟢 Recursive block starts 🟢
dfs(i = 1,res = [[]], subs = [1]):
        [[]].append([1]) >> res = [[], [1]]
```

```
        for i in range(1, 3):
                [1].append(2) >> subs=[1, 2]
                dfs(1+1,[[], [1]], [1, 2])
                 #🟢 Recursive block starts 🟢
dfs(i = 2,res = [[], [1]], subs = [1, 2]):
        [[], [1]].append([1, 2]) >> res = [[], [1], [1, 2]]
        for i in range(2, 3):
                [1, 2].append(3) >> subs=[1, 2, 3]
                dfs(2+1,[[], [1], [1, 2]], [1, 2, 3])
                 #🟢 Recursive block starts 🟢
dfs(i = 3,res = [[], [1], [1, 2]], subs = [1, 2, 3]):
        [[], [1], [1, 2]].append([1, 2, 3]) >> res = [[], [1], [1, 2], [1, 2, 3]]
        for i in range(3, 3):
        return [[], [1], [1, 2], [1, 2, 3]]
                              🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                 #🔴 Recursive block ends 🔴
                [1, 2, 3].pop() >> subs = [1, 2]
        return [[], [1], [1, 2], [1, 2, 3]]
                              🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                 #🔴 Recursive block ends 🔴
                [1, 2].pop() >> subs = [1]
                [1].append(3) >> subs=[1, 3]
                dfs(2+1,[[], [1], [1, 2], [1, 2, 3]], [1, 3])
                 #🟢 Recursive block starts 🟢
dfs(i = 3,res = [[], [1], [1, 2], [1, 2, 3]], subs = [1, 3]):
        [[], [1], [1, 2], [1, 2, 3]].append([1, 3]) >> res = [[], [1], [1, 2], [1, 2, 3], [1, 3]]
        for i in range(3, 3):
        return [[], [1], [1, 2], [1, 2, 3], [1, 3]]
                              🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                 #🔴 Recursive block ends 🔴
                [1, 3].pop() >> subs = [1]
        return [[], [1], [1, 2], [1, 2, 3], [1, 3]]
                              🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                 #🔴 Recursive block ends 🔴
                [1].pop() >> subs = []
                [].append(2) >> subs=[2]
                dfs(1+1,[[], [1], [1, 2], [1, 2, 3], [1, 3]], [2])
                 #🟢 Recursive block starts 🟢
dfs(i = 2,res = [[], [1], [1, 2], [1, 2, 3], [1, 3]], subs = [2]):
        [[], [1], [1, 2], [1, 2, 3], [1, 3]].append([2]) >> res = [[], [1], [1, 2], [1, 2, 3], [1, 3], [2]]
        for i in range(2, 3):
                [2].append(3) >> subs=[2, 3]
                dfs(2+1,[[], [1], [1, 2], [1, 2, 3], [1, 3], [2]], [2, 3])
                 #🟢 Recursive block starts 🟢
dfs(i = 3,res = [[], [1], [1, 2], [1, 2, 3], [1, 3], [2]], subs = [2, 3]):
        [[], [1], [1, 2], [1, 2, 3], [1, 3], [2]].append([2, 3]) >> res = [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3]]
        for i in range(3, 3):
        return [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3]]
                              🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                 #🔴 Recursive block ends 🔴
                [2, 3].pop() >> subs = [2]
        return [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3]]
                              🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                 #🔴 Recursive block ends 🔴
                [2].pop() >> subs = []
                [].append(3) >> subs=[3]
                dfs(2+1,[[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3]], [3])
                 #🟢 Recursive block starts 🟢
dfs(i = 3,res = [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3]], subs = [3]):
        [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3]].append([3]) >> res = [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3],
[3]]
        for i in range(3, 3):
        return [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]
                              🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                 #🔴 Recursive block ends 🔴
                [3].pop() >> subs = []
        return [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]
                              🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁
```

```
[[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]
```

4. Splitting a string into Descending Consecutive values

#splitting_string
#Medium

You are given a string `s` that consists of only digits.

Check if we can split `s` into **two or more non-empty substrings** such that the **numerical values** of the substrings are in **descending order** and the **difference** between numerical values of every two **adjacent substrings** is equal to `1`.

- For example, the string `s = "0090089"` can be split into `["0090", "089"]` with numerical values `[90,89]`. The values are in descending order and adjacent values differ by `1`, so this way is valid.
- Another example, the string `s = "001"` can be split into `["0", "01"]`, `["00", "1"]`, or `["0", "0", "1"]`. However all the ways are invalid because they have numerical values `[0,1]`, `[0,1]`, and `[0,0,1]` respectively, all of which are not in descending order.

Return `true` *if it is possible to split* `s` *as described above__, or* `false` *otherwise.*

A **substring** is a contiguous sequence of characters in a string.

Example 1:

Input: s = "1234"
Output: false
Explanation: There is no valid way to split s.

Example 2:

Input: s = "050043"
Output: true
Explanation: s can be split into `["05", "004", "3"]` with numerical values `[5,4,3].`
The values are in descending order with adjacent values differing by 1.

Example 3:

Input: s = "9080701"
Output: false
Explanation: There is no valid way to split s.

Constraints:

- `1 <= s.length <= 20`
- `s` only consists of digits.

Code

```
def splitString(s):
    def dfs(idx, prev):
        if len(s)==idx:
            return True
        for i in range(idx,len(s)):
            val = int(s[idx:i+1])
            if val==prev-1 and dfs(i+1, val):
                return True
        return False

    for i in range(len(s)-1):
        val = int(s[:i+1])
        if dfs(i+1, val):
            return True
    return False

print(splitString("0090089"))
print(splitString("10"))
print(splitString("4321"))
```

Explanation:

```
def splitString("0090089"):
        for i in range(7-1):
                #iteration 1
                i = 0
                val = int(s[:0+1]) = int("0") = 0
                if dfs(idx=1,prev=0):
                        #ignore
                        for i in range(1,7):
                                i = 1
                                val = int(s[1 : 2]) = int("0") = 0
                                if 0 == 0-1 #stop
                                i = 2
                                val = int(s[1 : 3]) = int("09") = 9
                                if 9 == 0-1 #stop
                                i = 3
                                val = int(s[1 : 4]) = int("090") = 90
                                if 90 == 0-1 #stop
                                i = 5
                                val = int(s[1 : 5]) = int("0900") = 900
                                if 900 == 0-1 #stop
                                i = 6
                                val = int(s[1 : 6]) = int("09008") = 9008
                                if 9008 == 0-1 #stop
                        return False

                # iteration 2
                i = 1
                val = int(s[:1+1]) = int("00") = 0
                if dfs(idx=2, prev = 0):
                        #ignore
                        for i in (2,7):
                                i = 2
                                val = int(s[2 : 3]) = int("9") = 9
                                if 9 == 0-1 #stop
                                i = 3
                                val = int(s[2 : 4]) = int("90") = 90
                                if 90 == 0-1 #stop
                                i = 4
                                val = int(s[2 : 5]) = int ("900") = 900
                                if 900 == 0-1 #stop
                                i = 5
                                val = int(s[2 : 6]) = int ("9008") = 9008
                                if 9008 == 0-1 #stop
                                i = 6
                                val = int(s[2 : 7]) = int ("90089") = 90089
                                if 90089 == 0-1 #stop
                        return False

                #iteration 3
                i = 2
                val = int(s[:2+1]) = int(s[:3] = int("009") = 9
                if dfs(idx =3, prev = 9):
                        #ignore
                        for i in (3,7):
                                i = 3
                                val = int(s[3 : 4]) = int("0") = 0
                                if 0 == 9-1 #stop
                                i = 4
                                val = int(s[3 : 5]) = int("00") = 0
                                if 0 == 9-1 #stop
                                i = 5
                                val = int(s[3 : 6]) = int("008") = 8
                                if 8 == 9-1 and dfs(idx = 5+1 = 6, prev = 8):
                                                                        #ignore
                                                                        for i in (6,7):
                                                                                i = 6
                                                                                val = int([6: 7]) = int("9") = 9
                                                                                if 9 == 8 - 1 # stop
                                                                        return False
                                i = 6
                                val = int(s[3 : 7]) = int("0089")= 89
                                if 89 == 9-1 #stop
                        return False
                i = 3
```

```
                val = int(s[:3+1]) = int(s[:4] = int("0090") = 90
                if dfs(idx =4, prev = 90):
                        #ignore
                        for i in range(4,7):
                                i = 4
                                val = int(s[4 : 5]) = int("0") = 0
                                if 0 == 90 -1 # stop
                                i = 5
                                val = int(s[4 : 6]) = int("08") = 8
                                if 8 == 90 -1 # stop
                                i = 6
                                val = int(s[4 : 7]) = int("089") = 89
                                if 89 == 90 -1 and dfs(idx = 6+1 = 7, prev = 89):
                                                                        if idx == len(s): # yes
                                                                                return True
                                        return True
                        return True
```

5. Word Search

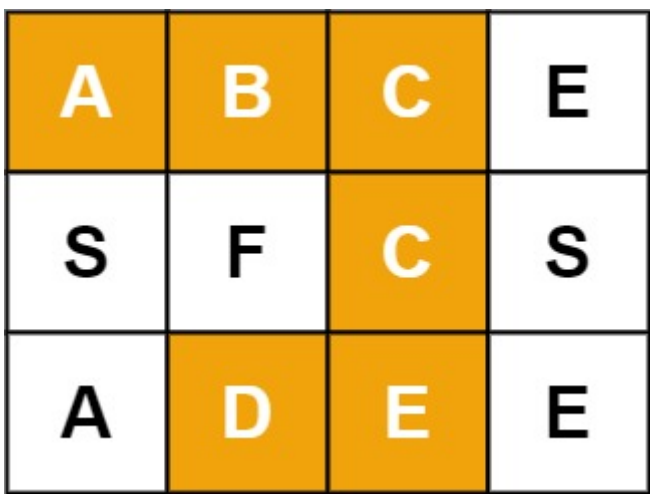
#word_search_I

#Medium

Given an `m x n` grid of characters `board` and a string `word`, return `true` *if* `word` *exists in the grid*.

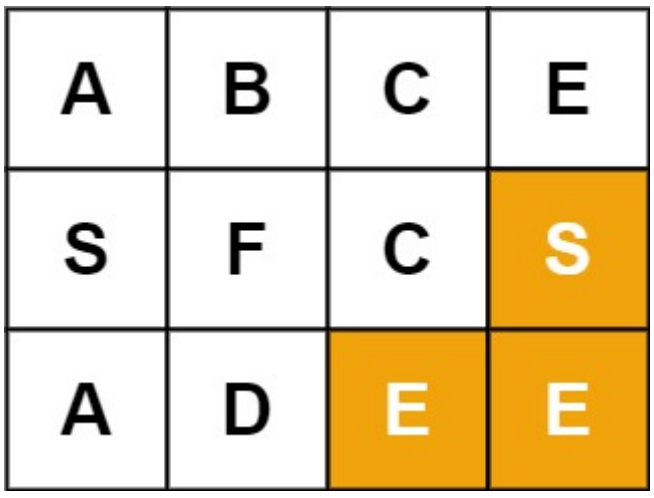
The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example 1:



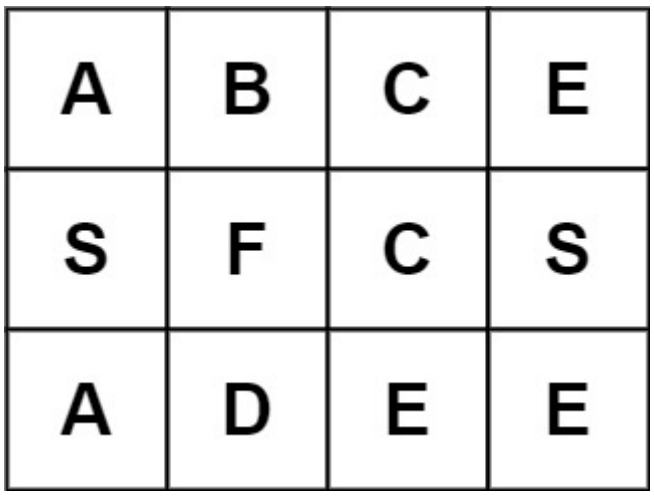
Input: `board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"`
Output: true

Example 2:



Input: `board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "SEE"`
Output: true

Example 3:



Input: `board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCB"`
Output: false

Constraints:

- `m == board.length`
- `n = board[i].length`
- `1 <= m, n <= 6`
- `1 <= word.length <= 15`
- `board` and `word` consists of only lowercase and uppercase English letters.

Follow up: Could you use search pruning to make your solution faster with a larger `board`?

Code

```python
def word_search(grid:list[list[str]],word:str)->bool:
    for r in range(len(grid)):
        for c in range(len(grid[0])):
            if explore(r, c,0):
                return True
    return False

def explore(r:int, c:int, i:int)->bool:
    if i==len(word):
        return True
    if (
        r < 0 or r >= len(grid) or
        c < 0 or c >= len(grid[0]) or
        grid[r][c] != word[i]
        ):
        return False

    temp, grid[r][c] = grid[r][c], "*"

    if(explore(r-1, c, i+1) or
        explore(r+1, c, i+1) or
        explore(r, c-1, i+1) or
        explore(r, c+1, i+1)):
        return True
    grid[r][c]=temp
    return False


if __name__ == '__main__':
    #global
    grid = [["A","B","C","E"],
         ["S","F","C","S"],
         ["A","D","E","E"]]
    word = "ABCCED"

    if not word_search(grid, word):
        print(f"Couldn't find \"{word}\"")
    else:
        print(f"\"{word}\" Found!")
```

Explanation

```
def word_search(grid = [['A', 'B', 'C', 'E'], ['S', 'F', 'C', 'S'], ['A', 'D', 'E', 'E']],word = ABCCED):
        for r in range(0, 3):
                for c in range(0, 4):
                        if explore(0, 0, 0):
                                        if 0==6: --> False
                                        if (
                                         0< 0 or 0 >= 3 or
                                         0 < 0 or c >= 4 or
                                         grid[0][0] != A
                                                ): --> False
                                        temp, grid[0][0] = grid[0][0], '*'
                                         >> grid = [['*', 'B', 'C', 'E'], ['S', 'F', 'C', 'S'], ['A', 'D', 'E', 'E']], temp = A
                                         #🟢 Recursive block starts 🟢
                                        if(explore(-1, 0, 1) or
                                                explore(1, 0, 1) or
                                                explore(0, -1, 1) or
                                                explore(0, 1, 1)):
                                elif explore(-1, 0, 1):
                                        if 1==6: --> False
                                        if (
                                         -1< 0 or -1 >= 3 or
                                         0 < 0 or c >= 4 or
                                         grid[-1][0] != B
                                                ): --> True
                                                 # 🟡 Base Case 🟡
                                                return False
                                elif explore(1, 0, 1):
                                        if 1==6: --> False
                                        if (
                                         1< 0 or 1 >= 3 or
                                         0 < 0 or c >= 4 or
                                         grid[1][0] != B
                                                ): --> True
                                                 # 🟡 Base Case 🟡
                                                return False
                                elif explore(0, -1, 1):
                                        if 1==6: --> False
                                        if (
                                         0< 0 or 0 >= 3 or
                                         -1 < 0 or c >= 4 or
                                         grid[0][-1] != B
                                                ): --> True
                                                 # 🟡 Base Case 🟡
                                                return False
                                elif explore(0, 1, 1):
                                        if 1==6: --> False
                                        if (
                                         0< 0 or 0 >= 3 or
                                         1 < 0 or c >= 4 or
                                         grid[0][1] != B
                                                ): --> False
                                        temp, grid[0][1] = grid[0][1], '*'
                                         >> grid = [['*', '*', 'C', 'E'], ['S', 'F', 'C', 'S'], ['A', 'D', 'E', 'E']], temp = B
                                         #🟢 Recursive block starts 🟢
                                        if(explore(-1, 1, 2) or
                                                explore(1, 1, 2) or
                                                explore(0, 0, 2) or
                                                explore(0, 2, 2)):
                                elif explore(-1, 1, 2):
                                        if 2==6: --> False
                                        if (
                                         -1< 0 or -1 >= 3 or
                                         1 < 0 or c >= 4 or
                                         grid[-1][1] != C
                                                ): --> True
                                                 # 🟡 Base Case 🟡
                                                return False
                                elif explore(1, 1, 2):
                                        if 2==6: --> False
                                        if (
                                         1< 0 or 1 >= 3 or
                                         1 < 0 or c >= 4 or
                                         grid[1][1] != C
                                                ): --> True
                                                 # 🟡 Base Case 🟡
                                                return False
```

```
explore(0, 0, 2):
       if 2==6: --> False
       if (
        0< 0 or 0 >= 3 or
        0 < 0 or c >= 4 or
        grid[0][0] != C
               ): --> True
                # 🟡 Base Case 🟡
               return False
explore(0, 2, 2):
       if 2==6: --> False
       if (
        0< 0 or 0 >= 3 or
        2 < 0 or c >= 4 or
        grid[0][2] != C
               ): --> False
       temp, grid[0][2] = grid[0][2], '*'
        >> grid = [['*', '*', '*', 'E'], ['S', 'F', 'C', 'S'], ['A', 'D', 'E', 'E']], temp = C
        #🟢 Recursive block starts 🟢
       if(explore(-1, 2, 3) or
               explore(1, 2, 3) or
               explore(0, 1, 3) or
               explore(0, 3, 3)):
explore(-1, 2, 3):
       if 3==6: --> False
       if (
        -1< 0 or -1 >= 3 or
        2 < 0 or c >= 4 or
        grid[-1][2] != C
               ): --> True
                # 🟡 Base Case 🟡
               return False
explore(1, 2, 3):
       if 3==6: --> False
       if (
        1< 0 or 1 >= 3 or
        2 < 0 or c >= 4 or
        grid[1][2] != C
               ): --> False
       temp, grid[1][2] = grid[1][2], '*'
        >> grid = [['*', '*', '*', 'E'], ['S', 'F', '*', 'S'], ['A', 'D', 'E', 'E']], temp = C
        #🟢 Recursive block starts 🟢
       if(explore(0, 2, 4) or
               explore(2, 2, 4) or
               explore(1, 1, 4) or
               explore(1, 3, 4)):
explore(0, 2, 4):
       if 4==6: --> False
       if (
        0< 0 or 0 >= 3 or
        2 < 0 or c >= 4 or
        grid[0][2] != E
               ): --> True
                # 🟡 Base Case 🟡
               return False
explore(2, 2, 4):
       if 4==6: --> False
       if (
        2< 0 or 2 >= 3 or
        2 < 0 or c >= 4 or
        grid[2][2] != E
               ): --> False
       temp, grid[2][2] = grid[2][2], '*'
        >> grid = [['*', '*', '*', 'E'], ['S', 'F', '*', 'S'], ['A', 'D', '*', 'E']], temp = E
        #🟢 Recursive block starts 🟢
       if(explore(1, 2, 5) or
               explore(3, 2, 5) or
               explore(2, 1, 5) or
               explore(2, 3, 5)):
explore(1, 2, 5):
       if 5==6: --> False
       if (
        1< 0 or 1 >= 3 or
        2 < 0 or c >= 4 or
        grid[1][2] != D
               ): --> True
```

```
                    # 🟡 Base Case 🟡
                return False
explore(3, 2, 5):
        if 5==6: --> False
        if (
         3< 0 or 3 >= 3 or
         2 < 0 or c >= 4 or
         grid[3][2] != D
                ): --> True
                 # 🟡 Base Case 🟡
                return False
explore(2, 1, 5):
        if 5==6: --> False
        if (
         2< 0 or 2 >= 3 or
         1 < 0 or c >= 4 or
         grid[2][1] != D
                ): --> False
        temp, grid[2][1] = grid[2][1], '*'
         >> grid = [['*', '*', '*', 'E'], ['S', 'F', '*', 'S'], ['A', '*', '*', 'E']], temp = D
         #🟢 Recursive block starts 🟢
        if(explore(1, 1, 6) or
                explore(3, 1, 6) or
                explore(2, 0, 6) or
                explore(2, 2, 6)):
              explore(1, 1, 6):
                if 6==6: --> True
                         # 🟡 Base Case 🟡
                         return True
                r=2, c=1, i=5
                        return True
                r=2, c=2, i=4
                        return True
                r=1, c=2, i=3
                        return True
                r=0, c=2, i=2
                        return True
                r=0, c=1, i=1
                        return True
                r=0, c=0, i=0
                        return True
        return True
```

6. Letter Combinations of a Phone Number

#Medium

Given a string containing digits from `2-9` inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:

Input: digits = "23"
Output: `["ad","ae","af","bd","be","bf","cd","ce","cf"]`
Example 2:

Input: digits = ""
Output: []

Example 3:

Input: digits = "2"
Output: `["a","b","c"]`

Constraints:

- `0 <= digits.length <= 4`
- `digits[i]` is a digit in the range `['2', '9']`.

Code

```
def letterCombinations(digits):
    """
    :type digits: str
    :rtype: List[str]
    """
    digitsToChar = {
        "2" : "abc",
        "3" : "def",
        "4" : "ghi",
        "5" : "jkl",
```

```
        "6" : "mno",
        "7" : "pqrs",
        "8" : "tuv",
        "9" : "wxyz"
        }
    def backtrack(i, res, currStr):
        if len(currStr)==len(digits):
            res.append(currStr)
            print(res)
            return
        for c in digitsToChar[digits[i]]:
            backtrack(i+1, res, currStr+c)
        return res

    if digits:
        return backtrack(0, [], "")
    else:
        return []
print(letterCombinations("23"))
```

Explanation

```
#inputs
digits[0]="2"
digits[1]="3"

#letter lookups
digitsToChar["2"]="abc"
digitsToChar["3"]="def"

backtrack(0,[],"")
        #doesn't hit base case
        currentString = ""
        for c in digitsToChar[digits[0]]="abc":
                c= "a"
                currentString = ""
                backtrack(0+1=1,[],""+"a")
                        # doesn't hit base case
                        currentString = "a"
                        for c in digitsToChar[digits[1]]="def":
                                c = "d"
                                backtrack(1+1=2,[],"a"+"d")
                                        #hits base case
                                        ["ad"]  #res
                                        return
                                c="e"
                                backtrack(1+1=2,["ad"],"a"+"e")
                                        #hits base case
                                        ["ad","ae"] #res
                                        return
                        c="f"
                        backtrack(1+1=2,["ad","ae"],"a"+"f")
                                #hits base case
                                ["ad","ae","af"] #res
                                return
                        #for loop ends
                return  ["ad","ae","af"] #res
#----------------------- "a" done -------------------------
c = "b"
backtrack(0+1=1,["ad","ae","af"],""+"b")
        #doesn't hit base case
        currentString = "b"
        for c in digitsToChar[digits[1]]="def"
                c="d"
                backtrack(1+1=2,["ad","ae","af"],"b"+"d")
                        #hits base case
                        ["ad","ae","af","bd"] #res
                        return
                c="e"
                backtrack(1+1=2,["ad","ae","af","bd"],"b"+"e")
                        #hits base case
                        ["ad","ae","af","bd","be"] #res
                        return
```

```
                        c="f"
                        backtrack(1+1=2,["ad","ae","af","bd","be"], "b"+"f")
                                #hits base case
                                ["ad","ae","af","bd","be","bf"] #res
                                return
                        #for loop ends
                return  ["ad","ae","af","bd","be","bf"] #res
        #------------------------ "b" done -------------------------
        c = "c"
        backtrack(0+1=1,["ad","ae","af","bd","be","bf"],""+"c")
                #doesn't hit base case
                currentString = "c"
                for c in digitsToChar[digits[1]]="def"
                        c="d"
                        backtrack(1+1=2,["ad","ae","af","bd","be","bf"], "c"+"d")
                                #hits base case
                                ["ad","ae","af","bd","be","bf","cd"] #res
                                return
                        c="e"
                        backtrack(1+1=2,["ad","ae","af","bd","be","bf","cd"],"c"+"e")
                                #hits base case
                                ["ad","ae","af","bd","be","bf","cd","ce"] #res
                                return
                        c="f"
                        backtrack(1+1=2,["ad","ae","af","bd","be","bf","cd","ce"],"c"+"f")
                                        #hits base case
                                        ["ad","ae","af","bd","be","bf","cd","ce","cf"]  #res
                                        return
                                #for loop ends
                        return ["ad","ae","af","bd","be","bf","cd","ce","cf"]  #res
                #------------------------ "c" done -------------------------

        return ["ad","ae","af","bd","be","bf","cd","ce","cf"] #res
```

7. Combination Sum

#Medium

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all **unique combinations** of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the

frequency

of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

Input: `candidates = [2,3,6,7], target = 7`
Output: `[[2,2,3],[7]]`
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.

Example 2:

Input: `candidates = [2,3,5], target = 8`
Output: `[[2,2,2,2],[2,3,3],[3,5]]`

Example 3:

Input: `candidates = [2], target = 1`
Output: []

Constraints:

- `1 <= candidates.length <= 30`
- `2 <= candidates[i] <= 40`
- All elements of `candidates` are **distinct**.

- `1 <= target <= 40`

Code

```
def dfs(i,combi,total,res):
    if total==target:
        res.append(combi[:])
        return
    if i>=len(candidates) or total>target:
        return
    combi.append(candidates[i])
    dfs(i,combi,total+candidates[i],res)
    combi.pop()
    dfs(i+1,combi,total,res)

    return res

if __name__ == '__main__':
  target=7
  candidates=[2,3,6,7]
  print(dfs(0,[],0,[]))
```

Explanation

```
def dfs(0, [],0, []):
        0 == 7? >> No #ignore
        (0>=4? >> No) or (0>7? >> No)  #ignore
        [].append([2,3,6,7][0]>> 2) >> combi = [2]

        dfs(i=0, combi=[2], total=0+2=2,res=[])
                #recursion block
                2 == 7? >> No #ignore
                (0>=4? >> No) or (2>7? >> No) #ignore
                [2].append([2,3,6,7][0]>> 2) >> combi = [2,2]

                dfs(i=0, combi=[2,2], total=2+2=4,res=[])
                        #recursion block
                        (4 == 7)? >> No #ignore
                        (0>=4? >> No) or (4>7? >> No) #ignore
                        [2,2].append([2,3,6,7][0]>> 2) >> combi = [2,2,2]


                        dfs(i=0, combi=[2,2,2], total=4+2=6,res=[])
                                #recursion block
                                (6 == 7)? >> No #ignore
                                ( 0>=4? > No ) or (6>7 ? >> No) # ignore
                                [2,2,2].append([2,3,6,7][0]>> 2) >> combi = [2,2,2,2]

                                dfs(i=0, combi=[2,2,2,2], total=6+2=8,res=[])
                                        # recursion block
                                        (8==7)? >> No # ignore
                                        (0>=4? >> No) or (8 > 7? >> Yes): # can't ignore
                                                return

                                [2,2,2,2].pop() >> [2,2,2]

                                dfs(i+1=0+1=1, combi=[2,2,2], total = 6, res = [])
                                        # recursion block
                                        (6 == 7)? >> No #ignore
                                        ( 1>=4? > No ) or (6>7 ? >> No) # ignore
                                        [2,2,2].append([2,3,6,7][1]>> 3) >> combi = [2,2,2,3]
                                        dfs(i=1, combi = [2,2,2,3] total = 6+3 = 9, res = [])
                                                # recursion block
                                                (9==7)? >> No
                                                (1>=4 ? >> No) or (9>7? >> Yes): # can't ignore
                                                        return
                                                #recursion block ends
                                        [2,2,2,3].pop() >> combi = [2,2,2]
```

```
                                dfs(i+1=1+1=2, combi = [2,2,2], total = 6, res = [])
                                        #recursion block
                                        (6==7)? >> No
                                        (2 >=4 >> No ) or (6>7 ? >> No) # ignore
                                        [2,2,2].append([2,3,6,7][2]>> 6) >> combi = [2,2,2,6]

                                        dfs(i=2, combi = [2,2,2,6], total = 6+6 = 12, res=[])
                                                # recursion block
                                                (12==7)?>> No #ignore
                                                (2>=4)? >> No or (12>7 >> Yes): # can't ignore
                                                        return
                                                #recursion block ends
                                [2,2,2,6].pop() >> combi = [2,2,2]
                                dfs(i+1=2+1=3, combi = [2,2,2], total = 6, res = [])
                                        #recursion block
                                        (6 == 7) ? >> No #ignore
                                        (3 >= 4? >> No) or (6 > 7 ? >> No) #ignore
                                        [2,2,2].append([2,3,6,7][3]>>[7]) >> combi = [2,2,2,7]
                                        dfs(i=3, combi = [2,2,2,7], total = 6+7 = 13, res = [])
                                                (13 == 7)? >> No #ignore
                                                (3>=4? >> No) or (13>7? >> Yes): # can't ignore
                                                        return
                                                        # recursion block ends
                                        [2,2,2,7].pop() >> combi = [2,2,2]
                                        dfs(i+1=3+1=4, combi = [2,2,2], total = 6, res = [])
                                                (6==7)? >> No #ignore
                                                (4>=4? >> Yes) or (6>7 ? >> No) # can't ignore
                                                        return
                                                # recursion block ends
                                        return [] # res=[]

                                return [] # res = []

                        [2,2,2].pop() >> combi = [2,2]
                        dfs(i+1=1,combi=[2,2],total=4, res = [])
                                #recursion block
                                (4==7)? >> No #ignore
                                (1>=4? >> No) or (4>7> No) #ignore
                                [2,2].append([2,3,6,7][1]>>3) >> combi = [2,2,3]
                                dfs(i = 1, combi = [2,2,3], total=4+3 = 7, res = [])
                                        #recursion block
                                        (7==7) >> Yes #can't ignore
                                                [].append([2,2,3]) >> res = [[2,2,3]]
                                                return
                        return [[2,2,3]]
                [2,2].pop() >> combi = [2]
                dfs(i+1=1, combi=[2], total=2,res=[[2,2,3]])
                        #ignore
                        #ignore
                        [2].append([2,3,6,7][1] >> 3) >> combi = [2,3]
                        dfs(i=1, combi = [2,3], total = 2+3 = 5, res = [[2,2,3]])
                                #ignore
                                #igore
                                [2,3].append(3) >> combi = [2,3,3]
                                dfs(i = 1, combi = [2,3,3], total = 5+3 = 8, res = [[2,2,3]])
                                        total>target:
                                                return
                                        [2,3,3].pop() >> combi= [2,3]
                                        dfs(i+1 = 1+1 = 2,combi=[2,3], total = 5, res=[[2,2,3]])
                                                #ignore
                                                #ignore
                                                [2,3].append(6) >> combi = [2,3,6]
                                                dfs(i = 1, combi = [2,3,6], total = 5+6 = 11, res = [[2,2,3]])
                                                        total>target:
                                                                return
                                        return [[2,2,3]]
                        [2,3].pop() >> combi = [2]
                        dfs(i+1 = 2, combi = [2],total = 2, res = [[2,2,3]])
                                #ignore
                                #ignore
                                [2].append(6) >> combi = [2,6]
                                dfs(i = 2, combi = [2,6], total = 2+6 = 8, res = [[2,2,3]])
                                        total>target:
                                                return
                                        [2,6].pop() >> combi = [2]
                                        dfs(i+1 = 3, combi = [2], total = 2, res= [[2,2,3]])
```

```
                                #ignore
                                #ignore
                                [2].append(7) >> [2,7]
                                dfs(i = 3, )


                    return [[2,2,3]]
```

8. Combination Sum II

#Medium

Given a collection of candidate numbers (`candidates`) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sum to `target` .

Each number in `candidates` may only be used **once** in the combination.

Note: The solution set must not contain duplicate combinations.

Example 1:

Input: `candidates = [10,1,2,7,6,1,5], target = 8`
Output:
`[ [1,1,6], [1,2,5], [1,7], [2,6] ]`

Example 2:

Input: `candidates = [2,5,2,1,2], target = 5`
Output:
`[`
`[1,2,2],`
`[5]`
`]`

Constraints:

- `1 <= candidates.length <= 100`
- `1 <= candidates[i] <= 50`
- `1 <= target <= 30`

Code

```
def backtrack(pos, res, combi, target):
    if target == 0:
        res.append(combi[:])
        return

    for i in range(pos, len(candidates)):
        if candidates[i] > target:
            break  # Skip candidates that are too large, as the list is sorted.

        if i > pos and candidates[i] == candidates[i - 1]:
            continue  # Skip duplicates to avoid duplicate combinations.

        combi.append(candidates[i])
        backtrack(i + 1, res, combi, target - candidates[i])
        combi.pop()
    return res
if __name__ == '__main__':
  candidates = [10, 1, 2, 7, 6, 1, 5]
  target = 8
  candidates.sort()
  print(backtrack(0, [], [], target))
```

Process

```
[]
[1]
[1,1]
[1,1,2] <-- append(5)? ❌ 5>4
[1,1]
[1,1,5] <-- append(6)? ❌ 6>1
[1,1]
[1,1,6] ✅ --> [[1,1,6]]
[1,1] <-- append(7)? ❌ 7>6
[1]
[1,2]
[1,2,5] ✅ --> [[1,1,6],[1,2,5]]
[1,2] <-- append(6)? ❌ 6>5
[1]
[1,5] <-- append(6) ? ❌ 6>2
[1]
[1,6] <-- append(7)? ❌ 7>1
[1]
[1,7] ✅ --> [[1,1,6],[1,2,5],[1,7]]
[1]
[ ]
[2]
[2,5] <-- append(6)? ❌ 6>1
[2,6] ✅ --> [[1,1,6],[1,2,5],[1,7],[2,6]]
[2] <-- append(7)? ❌ 7>6
[ ]
[5] <-- append(6)? ❌ 6>3
[ ]
[6] <-- append(7)? ❌ 7>2
[ ]
[7] <-- append(10)? ❌ 10>1
[ ]
>> [[1,1,6],[1,2,6],[1,7],[2,6]]
```

Explanation

```
if __name__ == '__main__':
        candidates = [10, 1, 2, 7, 6, 1, 5]
        target = 8
        candidates.sort() >> [1, 1, 2, 5, 6, 7, 10]
        print(backtrack(0, [], [], 8))
```

```
def backtrack(pos = 0, res = [], combi = [], target = 8):
        if target == 0: --> False
        for i in range((0, 7)):
                if 1 > 8: --> False
                if 0 > 0 and 1 == 10 : --> False
                [].append(1)            >> combi = [1]
        #Recursive Block Starts (1)

        backtrack(pos = 1, res = [], combi = [1], target = 7):
                if target == 0: --> False
                for i in range((1, 7)):
                        if 1 > 7: --> False
                        if 1 > 1 and 1 == 1 : --> False
                        [1].append(1)           >> combi = [1, 1]
        #Recursive Block Starts(1,1)

        backtrack(pos = 2, res = [], combi = [1, 1], target = 6):
                if target == 0: --> False
                for i in range((2, 7)):
                        if 2 > 6: --> False
                        if 2 > 2 and 2 == 1 : --> False
                        [1, 1].append(2)                >> combi = [1, 1, 2]
        #Recursive Block Starts(1,1,1)
```

```
backtrack(pos = 3, res = [], combi = [1, 1, 2], target = 4):
        if target == 0: --> False
        for i in range((3, 7)):
                if 5 > 4: --> True
                        break
                #Ending for loop
        return []

#Recurive Block Ends(1,1,1)
                [1, 1, 2].pop() >> combi = [1, 1]
                if 5 > 6: --> False
                if 3 > 2 and 5 == 2 : --> False
                [1, 1].append(5)                >> combi = [1, 1, 5]
#Recursive Block Starts(1,1,2)

backtrack(pos = 4, res = [], combi = [1, 1, 5], target = 1):
        if target == 0: --> False
        for i in range((4, 7)):
                if 6 > 1: --> True
                        break
                #Ending for loop
        return []

#Recurive Block Ends(1,1,2)
                [1, 1, 5].pop() >> combi = [1, 1]
                if 6 > 6: --> False
                if 4 > 2 and 6 == 5 : --> False
                [1, 1].append(6)                >> combi = [1, 1, 6]
#Recursive Block Starts(1,1,3)

backtrack(pos = 5, res = [], combi = [1, 1, 6], target = 0):
        if target == 0: --> True
                [].append([1, 1, 6])    >> res = [[1, 1, 6]]
                return
        #Reached Base Case
#Recurive Block Ends(1,1,3)
                [1, 1, 6].pop() >> combi = [1, 1]
                if 7 > 6: --> True
                        break
                #Ending for loop
        return [[1, 1, 6]]

#Recurive Block Ends(1,1)
                [1, 1].pop()    >> combi = [1]
                if 2 > 7: --> False
                if 2 > 1 and 2 == 1 : --> False
                [1].append(2)           >> combi = [1, 2]
#Recursive Block Starts(1,2)

backtrack(pos = 3, res = [[1, 1, 6]], combi = [1, 2], target = 5):
        if target == 0: --> False
        for i in range((3, 7)):
                if 5 > 5: --> False
                if 3 > 3 and 5 == 2 : --> False
                [1, 2].append(5)                >> combi = [1, 2, 5]
#Recursive Block Starts(1,2,1)

backtrack(pos = 4, res = [[1, 1, 6]], combi = [1, 2, 5], target = 0):
        if target == 0: --> True
                [[1, 1, 6]].append([1, 2, 5])   >> res = [[1, 1, 6], [1, 2, 5]]
                return
        #Reached Base Case
#Recurive Block Ends(1,2,1)
                [1, 2, 5].pop() >> combi = [1, 2]
                if 6 > 5: --> True
                        break
                #Ending for loop
        return [[1, 1, 6], [1, 2, 5]]

#Recurive Block Ends(1,2)
                [1, 2].pop()    >> combi = [1]
                if 5 > 7: --> False
                if 3 > 1 and 5 == 2 : --> False
                [1].append(5)           >> combi = [1, 5]
#Recursive Block Starts(1,3)
```

```
backtrack(pos = 4, res = [[1, 1, 6], [1, 2, 5]], combi = [1, 5], target = 2):
        if target == 0: --> False
        for i in range((4, 7)):
                if 6 > 2: --> True
                        break
                #Ending for loop
        return [[1, 1, 6], [1, 2, 5]]

#Recurive Block Ends(1,3)
                [1, 5].pop()    >> combi = [1]
                if 6 > 7: --> False
                if 4 > 1 and 6 == 5 : --> False
                [1].append(6)           >> combi = [1, 6]
#Recursive Block Starts(1,4)

backtrack(pos = 5, res = [[1, 1, 6], [1, 2, 5]], combi = [1, 6], target = 1):
        if target == 0: --> False
        for i in range((5, 7)):
                if 7 > 1: --> True
                        break
                #Ending for loop
        return [[1, 1, 6], [1, 2, 5]]

#Recurive Block Ends(1,4)
                [1, 6].pop()    >> combi = [1]
                if 7 > 7: --> False
                if 5 > 1 and 7 == 6 : --> False
                [1].append(7)           >> combi = [1, 7]
#Recursive Block Starts(1,5)

backtrack(pos = 6, res = [[1, 1, 6], [1, 2, 5]], combi = [1, 7], target = 0):
        if target == 0: --> True
                [[1, 1, 6], [1, 2, 5]].append([1, 7])   >> res = [[1, 1, 6], [1, 2, 5], [1, 7]]
                return
        #Reached Base Case
#Recurive Block Ends(1,5)
                [1, 7].pop()    >> combi = [1]
                if 10 > 7: --> True
                        break
                #Ending for loop
        return [[1, 1, 6], [1, 2, 5], [1, 7]]

#Recurive Block Ends(1)
                [1].pop()       >> combi = []
                if 1 > 8: --> False
                if 1 > 0 and 1 == 1 : --> True
                        continue
                if 2 > 8: --> False
                if 2 > 0 and 2 == 1 : --> False
                [].append(2)            >> combi = [2]
#Recursive Block Starts(2)

backtrack(pos = 3, res = [[1, 1, 6], [1, 2, 5], [1, 7]], combi = [2], target = 6):
        if target == 0: --> False
        for i in range((3, 7)):
                if 5 > 6: --> False
                if 3 > 3 and 5 == 2 : --> False
                [2].append(5)           >> combi = [2, 5]
#Recursive Block Starts(2,1)

backtrack(pos = 4, res = [[1, 1, 6], [1, 2, 5], [1, 7]], combi = [2, 5], target = 1):
        if target == 0: --> False
        for i in range((4, 7)):
                if 6 > 1: --> True
                        break
                #Ending for loop
        return [[1, 1, 6], [1, 2, 5], [1, 7]]

#Recurive Block Ends(2,1)
                [2, 5].pop()    >> combi = [2]
                if 6 > 6: --> False
                if 4 > 3 and 6 == 5 : --> False
                [2].append(6)           >> combi = [2, 6]
#Recursive Block Starts(2,2)

backtrack(pos = 5, res = [[1, 1, 6], [1, 2, 5], [1, 7]], combi = [2, 6], target = 0):
```

```
        if target == 0: --> True
                [[1, 1, 6], [1, 2, 5], [1, 7]].append([2, 6])   >> res = [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]
                return
        #Reached Base Case
#Recurive Block Ends(2,2)
                [2, 6].pop()    >> combi = [2]
                if 7 > 6: --> True
                        break
                #Ending for loop
        return [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]

#Recurive Block Ends(2)
                [2].pop()       >> combi = []
                if 5 > 8: --> False
                if 3 > 0 and 5 == 2 : --> False
                [].append(5)            >> combi = [5]
#Recursive Block Starts(3)

backtrack(pos = 4, res = [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]], combi = [5], target = 3):
        if target == 0: --> False
        for i in range((4, 7)):
                if 6 > 3: --> True
                        break
                #Ending for loop
        return [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]

#Recurive Block Ends(3)
                [5].pop()       >> combi = []
                if 6 > 8: --> False
                if 4 > 0 and 6 == 5 : --> False
                [].append(6)            >> combi = [6]
#Recursive Block Starts(4)

backtrack(pos = 5, res = [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]], combi = [6], target = 2):
        if target == 0: --> False
        for i in range((5, 7)):
                if 7 > 2: --> True
                        break
                #Ending for loop
        return [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]

#Recurive Block Ends(4)
                [6].pop()       >> combi = []
                if 7 > 8: --> False
                if 5 > 0 and 7 == 6 : --> False
                [].append(7)            >> combi = [7]
#Recursive Block Starts(5)

backtrack(pos = 6, res = [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]], combi = [7], target = 1):
        if target == 0: --> False
        for i in range((6, 7)):
                if 10 > 1: --> True
                        break
                #Ending for loop
        return [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]

#Recurive Block Ends(5)
                [7].pop()       >> combi = []
                if 10 > 8: --> True
                        break
                #Ending for loop
return [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]
```

9. N-Queens Problem

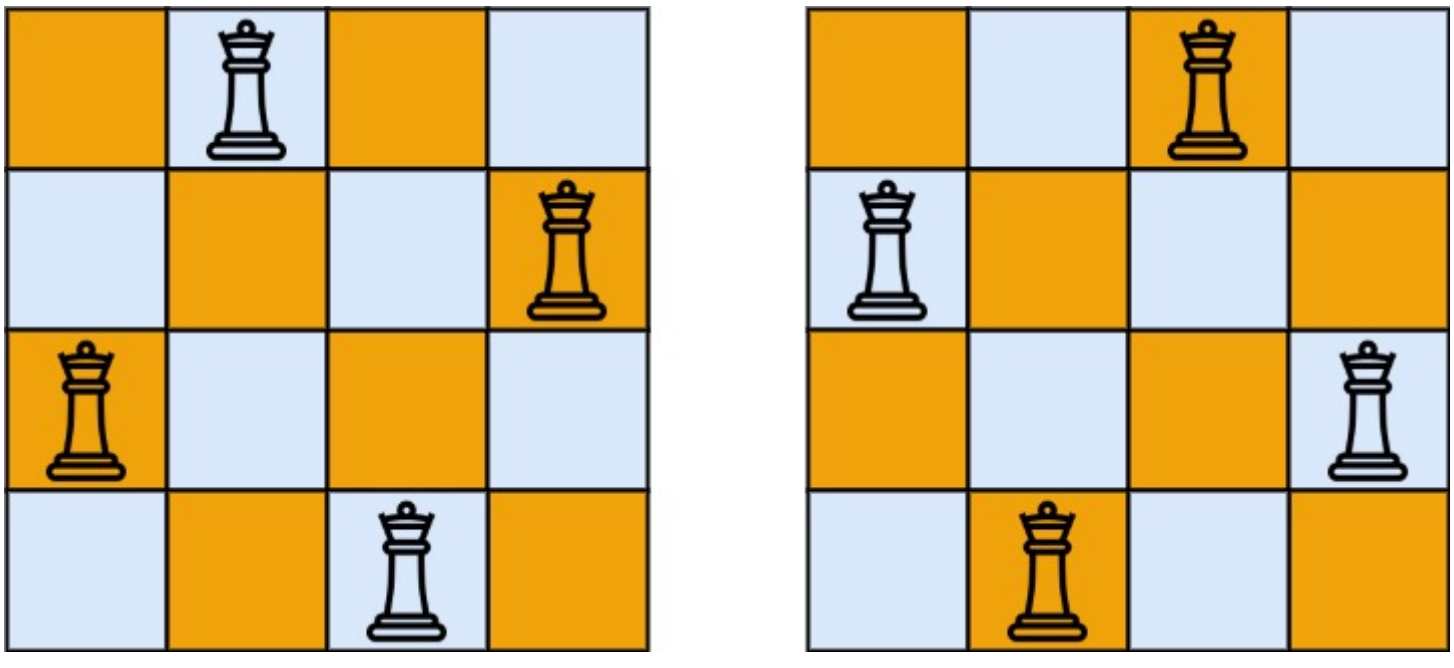
#Hard

The **n-queens** puzzle is the problem of placing `n` queens on an `n x n` chessboard such that no two queens attack each other.

Given an integer `n`, return *all distinct solutions to the **n-queens puzzle***. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space, respectively.

Example 1:



Input: n = 4
Output: `[[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]`
Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above

Example 2:

Input: n = 1
Output: `[["Q"]]`

Code

```
import pprint
col = set()
posDiag = set()
negDiag = set()
n = int(input("Give me n: "))
res = []
board = [["."] * n for i in range(n)]
def backtrack(r):
        if  r==n:
                res.append(["".join(row) for row in board])
                return
        for c in range(n):
                if c in col or (r+c) in posDiag or (r-c) in negDiag:
                        continue
                col.add(c)
                posDiag.add(r + c)
                negDiag.add(r - c)
                board[r][c] = "Q"
                backtrack(r+1)

                col.remove(c)
                posDiag.remove(r + c)
                negDiag.remove(r - c)
                board[r][c] = "."
        return res
pprint.pprint(backtrack(0))
```

Explanation

```
backtrack(0):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in set() or 0 in set() or 0 in set(): --> False
                set().add(0) >> {0}
                posDiag.add(0 + 0) >> {0}
                negDiag.add(0 - 0) >> {0}
                board[0][0] = 'Q'
>>
```

```
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(0+1) >> backtrack(1):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0} or 1 in {0} or 1 in {0}: --> True
                        continue
                #iteration 2
                if 1 in {0} or 2 in {0} or 0 in {0}: --> True
                        continue
                #iteration 3
                if 2 in {0} or 3 in {0} or -1 in {0}: --> False
                {0}.add(2) >> {0, 2}
                posDiag.add(1 + 2) >> {0, 3}
                negDiag.add(1 - 2) >> {0, -1}
                board[1][2] = 'Q'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 2} or 2 in {0, 3} or 2 in {0, -1}: --> True
                        continue
                #iteration 2
                if 1 in {0, 2} or 3 in {0, 3} or 1 in {0, -1}: --> True
                        continue
                #iteration 3
                if 2 in {0, 2} or 4 in {0, 3} or 0 in {0, -1}: --> True
                        continue
                #iteration 4
                if 3 in {0, 2} or 5 in {0, 3} or -1 in {0, -1}: --> True
                        continue
                #iteration 5
                if 4 in {0, 2} or 6 in {0, 3} or -2 in {0, -1}: --> False
                {0, 2}.add(4) >> {0, 2, 4}
                posDiag.add(2 + 4) >> {0, 3, 6}
                negDiag.add(2 - 4) >> {0, -1, -2}
                board[2][4] = 'Q'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 2, 4} or 3 in {0, 3, 6} or 3 in {0, -1, -2}: --> True
                        continue
                #iteration 2
                if 1 in {0, 2, 4} or 4 in {0, 3, 6} or 2 in {0, -1, -2}: --> False
                {0, 2, 4}.add(1) >> {0, 1, 2, 4}
                posDiag.add(3 + 1) >> {0, 3, 4, 6}
                negDiag.add(3 - 1) >> {0, 2, -1, -2}
                board[3][1] = 'Q'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
```

```
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 1, 2, 4} or 4 in {0, 3, 4, 6} or 4 in {0, 2, -1, -2}: --> True
                        continue
                #iteration 2
                if 1 in {0, 1, 2, 4} or 5 in {0, 3, 4, 6} or 3 in {0, 2, -1, -2}: --> True
                        continue
                #iteration 3
                if 2 in {0, 1, 2, 4} or 6 in {0, 3, 4, 6} or 2 in {0, 2, -1, -2}: --> True
                        continue
                #iteration 4
                if 3 in {0, 1, 2, 4} or 7 in {0, 3, 4, 6} or 1 in {0, 2, -1, -2}: --> False
                {0, 1, 2, 4}.add(3) >> {0, 1, 2, 3, 4}
                posDiag.add(4 + 3) >> {0, 3, 4, 6, 7}
                negDiag.add(4 - 3) >> {0, 1, 2, -2, -1}
                board[4][3] = 'Q'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(4+1) >> backtrack(5):
        if  r==n: --> True
                🟡 #Base Case 🟡
                res.append([''.join(row) for row in board])
>>
[['Q....', '..Q..', '....Q', '.Q...', '...Q.']]
                return
                #Maximum recursive depth reached
                🔴 #End of recursive block🔴

                {0, 1, 2, 3, 4}.remove(3) >> {0, 1, 2, 4}
                posDiag.remove(4 + 3) >> {0, 3, 4, 6}
                negDiag.remove(4 - 3) >> {0, 2, -2, -1}
                board[4][3] = '.'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 5
                if 4 in {0, 1, 2, 4} or 8 in {0, 3, 4, 6} or 0 in {0, 2, -2, -1}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {0, 1, 2, 4}.remove(1) >> {0, 2, 4}
                posDiag.remove(3 + 1) >> {0, 3, 6}
                negDiag.remove(3 - 1) >> {0, -2, -1}
                board[3][1] = '.'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 3
                if 2 in {0, 2, 4} or 5 in {0, 3, 6} or 1 in {0, -2, -1}: --> True
                        continue
                #iteration 4
                if 3 in {0, 2, 4} or 6 in {0, 3, 6} or 0 in {0, -2, -1}: --> True
                        continue
                #iteration 5
                if 4 in {0, 2, 4} or 7 in {0, 3, 6} or -1 in {0, -2, -1}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁
```

```
                🔴 #End of recursive block🔴

                {0, 2, 4}.remove(4) >> {0, 2}
                posDiag.remove(2 + 4) >> {0, 3}
                negDiag.remove(2 - 4) >> {0, -1}
                board[2][4] = '.'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {0, 2}.remove(2) >> {0}
                posDiag.remove(1 + 2) >> {0}
                negDiag.remove(1 - 2) >> {0}
                board[1][2] = '.'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 4
                if 3 in {0} or 4 in {0} or -2 in {0}: --> False
                {0}.add(3) >> {0, 3}
                posDiag.add(1 + 3) >> {0, 4}
                negDiag.add(1 - 3) >> {0, -2}
                board[1][3] = 'Q'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 3} or 2 in {0, 4} or 2 in {0, -2}: --> True
                        continue
                #iteration 2
                if 1 in {0, 3} or 3 in {0, 4} or 1 in {0, -2}: --> False
                {0, 3}.add(1) >> {0, 1, 3}
                posDiag.add(2 + 1) >> {0, 3, 4}
                negDiag.add(2 - 1) >> {0, 1, -2}
                board[2][1] = 'Q'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 1, 3} or 3 in {0, 3, 4} or 3 in {0, 1, -2}: --> True
                        continue
                #iteration 2
                if 1 in {0, 1, 3} or 4 in {0, 3, 4} or 2 in {0, 1, -2}: --> True
                        continue
                #iteration 3
                if 2 in {0, 1, 3} or 5 in {0, 3, 4} or 1 in {0, 1, -2}: --> True
                        continue
                #iteration 4
                if 3 in {0, 1, 3} or 6 in {0, 3, 4} or 0 in {0, 1, -2}: --> True
                        continue
                #iteration 5
```

```
                if 4 in {0, 1, 3} or 7 in {0, 3, 4} or -1 in {0, 1, -2}: --> False
                {0, 1, 3}.add(4) >> {0, 1, 3, 4}
                posDiag.add(3 + 4) >> {0, 3, 4, 7}
                negDiag.add(3 - 4) >> {0, 1, -2, -1}
                board[3][4] = 'Q'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 1, 3, 4} or 4 in {0, 3, 4, 7} or 4 in {0, 1, -2, -1}: --> True
                        continue
                #iteration 2
                if 1 in {0, 1, 3, 4} or 5 in {0, 3, 4, 7} or 3 in {0, 1, -2, -1}: --> True
                        continue
                #iteration 3
                if 2 in {0, 1, 3, 4} or 6 in {0, 3, 4, 7} or 2 in {0, 1, -2, -1}: --> False
                {0, 1, 3, 4}.add(2) >> {0, 2, 1, 3, 4}
                posDiag.add(4 + 2) >> {0, 3, 4, 6, 7}
                negDiag.add(4 - 2) >> {0, 1, 2, -2, -1}
                board[4][2] = 'Q'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(4+1) >> backtrack(5):
        if  r==n: --> True
                🟡 #Base Case 🟡
                res.append([''.join(row) for row in board])
>>
[['Q....', '..Q..', '....Q', '.Q...', '...Q.'],
 ['Q....', '...Q.', '.Q...', '....Q', '..Q..']]
                return
                #Maximum recursive depth reached
                🔴 #End of recursive block🔴

                {0, 2, 1, 3, 4}.remove(2) >> {0, 1, 3, 4}
                posDiag.remove(4 + 2) >> {0, 3, 4, 7}
                negDiag.remove(4 - 2) >> {0, 1, -2, -1}
                board[4][2] = '.'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.']]
                #iteration 4
                if 3 in {0, 1, 3, 4} or 7 in {0, 3, 4, 7} or 1 in {0, 1, -2, -1}: --> True
                        continue
                #iteration 5
                if 4 in {0, 1, 3, 4} or 8 in {0, 3, 4, 7} or 0 in {0, 1, -2, -1}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {0, 1, 3, 4}.remove(4) >> {0, 1, 3}
                posDiag.remove(3 + 4) >> {0, 3, 4}
                negDiag.remove(3 - 4) >> {0, 1, -2}
                board[3][4] = '.'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
```

```
 ['.', '.', '.', '.', '.']]
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..']]

        ▩▩▩▩ #End of a level ▩▩▩▩

                🔴 #End of recursive block🔴

                {0, 1, 3}.remove(1) >> {0, 3}
                posDiag.remove(2 + 1) >> {0, 4}
                negDiag.remove(2 - 1) >> {0, -2}
                board[2][1] = '.'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 3
                if 2 in {0, 3} or 4 in {0, 4} or 0 in {0, -2}: --> True
                        continue
                #iteration 4
                if 3 in {0, 3} or 5 in {0, 4} or -1 in {0, -2}: --> True
                        continue
                #iteration 5
                if 4 in {0, 3} or 6 in {0, 4} or -2 in {0, -2}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..']]

        ▩▩▩▩ #End of a level ▩▩▩▩

                🔴 #End of recursive block🔴

                {0, 3}.remove(3) >> {0}
                posDiag.remove(1 + 3) >> {0}
                negDiag.remove(1 - 3) >> {0}
                board[1][3] = '.'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 5
                if 4 in {0} or 5 in {0} or -3 in {0}: --> False
                {0}.add(4) >> {0, 4}
                posDiag.add(1 + 4) >> {0, 5}
                negDiag.add(1 - 4) >> {0, -3}
                board[1][4] = 'Q'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block ▩▩
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 4} or 2 in {0, 5} or 2 in {0, -3}: --> True
                        continue
                #iteration 2
                if 1 in {0, 4} or 3 in {0, 5} or 1 in {0, -3}: --> False
                {0, 4}.add(1) >> {0, 1, 4}
                posDiag.add(2 + 1) >> {0, 5, 3}
                negDiag.add(2 - 1) >> {0, 1, -3}
                board[2][1] = 'Q'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
```

```
        for c in range(5):
                #iteration 1
                if 0 in {0, 1, 4} or 3 in {0, 5, 3} or 3 in {0, 1, -3}: --> True
                        continue
                #iteration 2
                if 1 in {0, 1, 4} or 4 in {0, 5, 3} or 2 in {0, 1, -3}: --> True
                        continue
                #iteration 3
                if 2 in {0, 1, 4} or 5 in {0, 5, 3} or 1 in {0, 1, -3}: --> True
                        continue
                #iteration 4
                if 3 in {0, 1, 4} or 6 in {0, 5, 3} or 0 in {0, 1, -3}: --> True
                        continue
                #iteration 5
                if 4 in {0, 1, 4} or 7 in {0, 5, 3} or -1 in {0, 1, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {0, 1, 4}.remove(1) >> {0, 4}
                posDiag.remove(2 + 1) >> {0, 5}
                negDiag.remove(2 - 1) >> {0, -3}
                board[2][1] = '.'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 3
                if 2 in {0, 4} or 4 in {0, 5} or 0 in {0, -3}: --> True
                        continue
                #iteration 4
                if 3 in {0, 4} or 5 in {0, 5} or -1 in {0, -3}: --> True
                        continue
                #iteration 5
                if 4 in {0, 4} or 6 in {0, 5} or -2 in {0, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {0, 4}.remove(4) >> {0}
                posDiag.remove(1 + 4) >> {0}
                negDiag.remove(1 - 4) >> {0}
                board[1][4] = '.'
>>
[['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {0}.remove(0) >> set()
                posDiag.remove(0 + 0) >> set()
                negDiag.remove(0 - 0) >> set()
                board[0][0] = '.'
>>
[['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 2
                if 1 in set() or 1 in set() or -1 in set(): --> False
                set().add(1) >> {1}
```

```
                posDiag.add(0 + 1) >> {1}
                negDiag.add(0 - 1) >> {-1}
                board[0][1] = 'Q'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(0+1) >> backtrack(1):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {1} or 1 in {1} or 1 in {-1}: --> True
                        continue
                #iteration 2
                if 1 in {1} or 2 in {1} or 0 in {-1}: --> True
                        continue
                #iteration 3
                if 2 in {1} or 3 in {1} or -1 in {-1}: --> True
                        continue
                #iteration 4
                if 3 in {1} or 4 in {1} or -2 in {-1}: --> False
                {1}.add(3) >> {3, 1}
                posDiag.add(1 + 3) >> {1, 4}
                negDiag.add(1 - 3) >> {-1, -2}
                board[1][3] = 'Q'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {3, 1} or 2 in {1, 4} or 2 in {-1, -2}: --> False
                {3, 1}.add(0) >> {0, 3, 1}
                posDiag.add(2 + 0) >> {1, 2, 4}
                negDiag.add(2 - 0) >> {2, -1, -2}
                board[2][0] = 'Q'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 3, 1} or 3 in {1, 2, 4} or 3 in {2, -1, -2}: --> True
                        continue
                #iteration 2
                if 1 in {0, 3, 1} or 4 in {1, 2, 4} or 2 in {2, -1, -2}: --> True
                        continue
                #iteration 3
                if 2 in {0, 3, 1} or 5 in {1, 2, 4} or 1 in {2, -1, -2}: --> False
                {0, 3, 1}.add(2) >> {2, 0, 3, 1}
                posDiag.add(3 + 2) >> {1, 2, 5, 4}
                negDiag.add(3 - 2) >> {1, 2, -1, -2}
                board[3][2] = 'Q'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
        if  r==n: --> False
        for c in range(5):
```

```
                #iteration 1
                if 0 in {2, 0, 3, 1} or 4 in {1, 2, 5, 4} or 4 in {1, 2, -1, -2}: --> True
                        continue
                #iteration 2
                if 1 in {2, 0, 3, 1} or 5 in {1, 2, 5, 4} or 3 in {1, 2, -1, -2}: --> True
                        continue
                #iteration 3
                if 2 in {2, 0, 3, 1} or 6 in {1, 2, 5, 4} or 2 in {1, 2, -1, -2}: --> True
                        continue
                #iteration 4
                if 3 in {2, 0, 3, 1} or 7 in {1, 2, 5, 4} or 1 in {1, 2, -1, -2}: --> True
                        continue
                #iteration 5
                if 4 in {2, 0, 3, 1} or 8 in {1, 2, 5, 4} or 0 in {1, 2, -1, -2}: --> False
                {2, 0, 3, 1}.add(4) >> {2, 0, 3, 1, 4}
                posDiag.add(4 + 4) >> {1, 2, 5, 4, 8}
                negDiag.add(4 - 4) >> {0, 1, 2, -1, -2}
                board[4][4] = 'Q'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q']]
                🟢 #Starting Recursive block 🟢
backtrack(4+1) >> backtrack(5):
        if  r==n: --> True
                🟡 #Base Case 🟡
                res.append([''.join(row) for row in board])
>>
[['Q....', '..Q..', '....Q', '.Q...', '...Q.'],
 ['Q....', '...Q.', '.Q...', '....Q', '..Q..'],
 ['.Q...', '...Q.', 'Q....', '..Q..', '....Q']]
                return
                #Maximum recursive depth reached
                🔴 #End of recursive block🔴

                {2, 0, 3, 1, 4}.remove(4) >> {2, 0, 3, 1}
                posDiag.remove(4 + 4) >> {1, 2, 5, 4}
                negDiag.remove(4 - 4) >> {1, 2, -1, -2}
                board[4][4] = '.'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.']]
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {2, 0, 3, 1}.remove(2) >> {0, 3, 1}
                posDiag.remove(3 + 2) >> {1, 2, 4}
                negDiag.remove(3 - 2) >> {2, -1, -2}
                board[3][2] = '.'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 4
                if 3 in {0, 3, 1} or 6 in {1, 2, 4} or 0 in {2, -1, -2}: --> True
                        continue
                #iteration 5
                if 4 in {0, 3, 1} or 7 in {1, 2, 4} or -1 in {2, -1, -2}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴
```

```
                {0, 3, 1}.remove(0) >> {3, 1}
                posDiag.remove(2 + 0) >> {1, 4}
                negDiag.remove(2 - 0) >> {-1, -2}
                board[2][0] = '.'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 2
                if 1 in {3, 1} or 3 in {1, 4} or 1 in {-1, -2}: --> True
                        continue
                #iteration 3
                if 2 in {3, 1} or 4 in {1, 4} or 0 in {-1, -2}: --> True
                        continue
                #iteration 4
                if 3 in {3, 1} or 5 in {1, 4} or -1 in {-1, -2}: --> True
                        continue
                #iteration 5
                if 4 in {3, 1} or 6 in {1, 4} or -2 in {-1, -2}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {3, 1}.remove(3) >> {1}
                posDiag.remove(1 + 3) >> {1}
                negDiag.remove(1 - 3) >> {-1}
                board[1][3] = '.'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 5
                if 4 in {1} or 5 in {1} or -3 in {-1}: --> False
                {1}.add(4) >> {1, 4}
                posDiag.add(1 + 4) >> {1, 5}
                negDiag.add(1 - 4) >> {-1, -3}
                board[1][4] = 'Q'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {1, 4} or 2 in {1, 5} or 2 in {-1, -3}: --> False
                {1, 4}.add(0) >> {0, 1, 4}
                posDiag.add(2 + 0) >> {1, 2, 5}
                negDiag.add(2 - 0) >> {2, -1, -3}
                board[2][0] = 'Q'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 1, 4} or 3 in {1, 2, 5} or 3 in {2, -1, -3}: --> True
                        continue
                #iteration 2
```

```
                if 1 in {0, 1, 4} or 4 in {1, 2, 5} or 2 in {2, -1, -3}: --> True
                        continue
                #iteration 3
                if 2 in {0, 1, 4} or 5 in {1, 2, 5} or 1 in {2, -1, -3}: --> True
                        continue
                #iteration 4
                if 3 in {0, 1, 4} or 6 in {1, 2, 5} or 0 in {2, -1, -3}: --> False
                {0, 1, 4}.add(3) >> {3, 0, 1, 4}
                posDiag.add(3 + 3) >> {1, 6, 2, 5}
                negDiag.add(3 - 3) >> {0, 2, -1, -3}
                board[3][3] = 'Q'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {3, 0, 1, 4} or 4 in {1, 6, 2, 5} or 4 in {0, 2, -1, -3}: --> True
                        continue
                #iteration 2
                if 1 in {3, 0, 1, 4} or 5 in {1, 6, 2, 5} or 3 in {0, 2, -1, -3}: --> True
                        continue
                #iteration 3
                if 2 in {3, 0, 1, 4} or 6 in {1, 6, 2, 5} or 2 in {0, 2, -1, -3}: --> True
                        continue
                #iteration 4
                if 3 in {3, 0, 1, 4} or 7 in {1, 6, 2, 5} or 1 in {0, 2, -1, -3}: --> True
                        continue
                #iteration 5
                if 4 in {3, 0, 1, 4} or 8 in {1, 6, 2, 5} or 0 in {0, 2, -1, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {3, 0, 1, 4}.remove(3) >> {0, 1, 4}
                posDiag.remove(3 + 3) >> {1, 2, 5}
                negDiag.remove(3 - 3) >> {2, -1, -3}
                board[3][3] = '.'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 5
                if 4 in {0, 1, 4} or 7 in {1, 2, 5} or -1 in {2, -1, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {0, 1, 4}.remove(0) >> {1, 4}
                posDiag.remove(2 + 0) >> {1, 5}
                negDiag.remove(2 - 0) >> {-1, -3}
                board[2][0] = '.'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 2
                if 1 in {1, 4} or 3 in {1, 5} or 1 in {-1, -3}: --> True
                        continue
```

```
                #iteration 3
                if 2 in {1, 4} or 4 in {1, 5} or 0 in {-1, -3}: --> False
                {1, 4}.add(2) >> {2, 1, 4}
                posDiag.add(2 + 2) >> {1, 4, 5}
                negDiag.add(2 - 2) >> {0, -1, -3}
                board[2][2] = 'Q'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {2, 1, 4} or 3 in {1, 4, 5} or 3 in {0, -1, -3}: --> False
                {2, 1, 4}.add(0) >> {0, 2, 1, 4}
                posDiag.add(3 + 0) >> {1, 3, 4, 5}
                negDiag.add(3 - 0) >> {0, 3, -1, -3}
                board[3][0] = 'Q'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 2, 1, 4} or 4 in {1, 3, 4, 5} or 4 in {0, 3, -1, -3}: --> True
                        continue
                #iteration 2
                if 1 in {0, 2, 1, 4} or 5 in {1, 3, 4, 5} or 3 in {0, 3, -1, -3}: --> True
                        continue
                #iteration 3
                if 2 in {0, 2, 1, 4} or 6 in {1, 3, 4, 5} or 2 in {0, 3, -1, -3}: --> True
                        continue
                #iteration 4
                if 3 in {0, 2, 1, 4} or 7 in {1, 3, 4, 5} or 1 in {0, 3, -1, -3}: --> False
                {0, 2, 1, 4}.add(3) >> {0, 2, 1, 4, 3}
                posDiag.add(4 + 3) >> {1, 3, 4, 5, 7}
                negDiag.add(4 - 3) >> {1, 0, 3, -1, -3}
                board[4][3] = 'Q'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(4+1) >> backtrack(5):
        if  r==n: --> True
                🟡 #Base Case 🟡
                res.append([''.join(row) for row in board])
>>
[['Q....', '..Q..', '....Q', '.Q...', '...Q.'],
 ['Q....', '...Q.', '.Q...', '....Q', '..Q..'],
 ['.Q...', '...Q.', 'Q....', '..Q..', '....Q'],
 ['.Q...', '....Q', '..Q..', 'Q....', '...Q.']]
                return
                #Maximum recursive depth reached
                🔴 #End of recursive block🔴

                {0, 2, 1, 4, 3}.remove(3) >> {0, 2, 1, 4}
                posDiag.remove(4 + 3) >> {1, 3, 4, 5}
                negDiag.remove(4 - 3) >> {0, 3, -1, -3}
                board[4][3] = '.'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
```

```
 ['.', '.', '.', '.', '.']]
                #iteration 5
                if 4 in {0, 2, 1, 4} or 8 in {1, 3, 4, 5} or 0 in {0, 3, -1, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {0, 2, 1, 4}.remove(0) >> {2, 1, 4}
                posDiag.remove(3 + 0) >> {1, 4, 5}
                negDiag.remove(3 - 0) >> {0, -1, -3}
                board[3][0] = '.'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 2
                if 1 in {2, 1, 4} or 4 in {1, 4, 5} or 2 in {0, -1, -3}: --> True
                        continue
                #iteration 3
                if 2 in {2, 1, 4} or 5 in {1, 4, 5} or 1 in {0, -1, -3}: --> True
                        continue
                #iteration 4
                if 3 in {2, 1, 4} or 6 in {1, 4, 5} or 0 in {0, -1, -3}: --> True
                        continue
                #iteration 5
                if 4 in {2, 1, 4} or 7 in {1, 4, 5} or -1 in {0, -1, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {2, 1, 4}.remove(2) >> {1, 4}
                posDiag.remove(2 + 2) >> {1, 5}
                negDiag.remove(2 - 2) >> {-1, -3}
                board[2][2] = '.'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 4
                if 3 in {1, 4} or 5 in {1, 5} or -1 in {-1, -3}: --> True
                        continue
                #iteration 5
                if 4 in {1, 4} or 6 in {1, 5} or -2 in {-1, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {1, 4}.remove(4) >> {1}
                posDiag.remove(1 + 4) >> {1}
                negDiag.remove(1 - 4) >> {-1}
                board[1][4] = '.'
>>
[['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.']]
```

```
        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {1}.remove(1) >> set()
                posDiag.remove(0 + 1) >> set()
                negDiag.remove(0 - 1) >> set()
                board[0][1] = '.'
>>
[['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 3
                if 2 in set() or 2 in set() or -2 in set(): --> False
                set().add(2) >> {2}
                posDiag.add(0 + 2) >> {2}
                negDiag.add(0 - 2) >> {-2}
                board[0][2] = 'Q'
>>
[['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(0+1) >> backtrack(1):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {2} or 1 in {2} or 1 in {-2}: --> False
                {2}.add(0) >> {0, 2}
                posDiag.add(1 + 0) >> {1, 2}
                negDiag.add(1 - 0) >> {1, -2}
                board[1][0] = 'Q'
>>
[['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 2} or 2 in {1, 2} or 2 in {1, -2}: --> True
                        continue
                #iteration 2
                if 1 in {0, 2} or 3 in {1, 2} or 1 in {1, -2}: --> True
                        continue
                #iteration 3
                if 2 in {0, 2} or 4 in {1, 2} or 0 in {1, -2}: --> True
                        continue
                #iteration 4
                if 3 in {0, 2} or 5 in {1, 2} or -1 in {1, -2}: --> False
                {0, 2}.add(3) >> {3, 0, 2}
                posDiag.add(2 + 3) >> {5, 1, 2}
                negDiag.add(2 - 3) >> {1, -2, -1}
                board[2][3] = 'Q'
>>
[['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {3, 0, 2} or 3 in {5, 1, 2} or 3 in {1, -2, -1}: --> True
                        continue
                #iteration 2
                if 1 in {3, 0, 2} or 4 in {5, 1, 2} or 2 in {1, -2, -1}: --> False
```

```
                {3, 0, 2}.add(1) >> {1, 3, 0, 2}
                posDiag.add(3 + 1) >> {4, 5, 1, 2}
                negDiag.add(3 - 1) >> {2, 1, -2, -1}
                board[3][1] = 'Q'
>>
[['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {1, 3, 0, 2} or 4 in {4, 5, 1, 2} or 4 in {2, 1, -2, -1}: --> True
                        continue
                #iteration 2
                if 1 in {1, 3, 0, 2} or 5 in {4, 5, 1, 2} or 3 in {2, 1, -2, -1}: --> True
                        continue
                #iteration 3
                if 2 in {1, 3, 0, 2} or 6 in {4, 5, 1, 2} or 2 in {2, 1, -2, -1}: --> True
                        continue
                #iteration 4
                if 3 in {1, 3, 0, 2} or 7 in {4, 5, 1, 2} or 1 in {2, 1, -2, -1}: --> True
                        continue
                #iteration 5
                if 4 in {1, 3, 0, 2} or 8 in {4, 5, 1, 2} or 0 in {2, 1, -2, -1}: --> False
                {1, 3, 0, 2}.add(4) >> {1, 3, 0, 2, 4}
                posDiag.add(4 + 4) >> {4, 5, 1, 2, 8}
                negDiag.add(4 - 4) >> {0, 2, 1, -2, -1}
                board[4][4] = 'Q'
>>
[['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q']]
                🟢 #Starting Recursive block 🟢
backtrack(4+1) >> backtrack(5):
        if  r==n: --> True
                🟡 #Base Case 🟡
                res.append([''.join(row) for row in board])
>>
[['Q....', '..Q..', '....Q', '.Q...', '...Q.'],
 ['Q....', '...Q.', '.Q...', '....Q', '..Q..'],
 ['.Q...', '...Q.', 'Q....', '..Q..', '....Q'],
 ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'],
 ['..Q..', 'Q....', '...Q.', '.Q...', '....Q']]
                return
                #Maximum recursive depth reached
                🔴 #End of recursive block🔴
                {1, 3, 0, 2, 4}.remove(4) >> {1, 3, 0, 2}
                posDiag.remove(4 + 4) >> {4, 5, 1, 2}
                negDiag.remove(4 - 4) >> {2, 1, -2, -1}
                board[4][4] = '.'
>>
[['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {1, 3, 0, 2}.remove(1) >> {3, 0, 2}
                posDiag.remove(3 + 1) >> {5, 1, 2}
                negDiag.remove(3 - 1) >> {1, -2, -1}
                board[3][1] = '.'
>>
[['.', '.', 'Q', '.', '.'],
```

```
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 3
                if 2 in {3, 0, 2} or 5 in {5, 1, 2} or 1 in {1, -2, -1}: --> True
                        continue
                #iteration 4
                if 3 in {3, 0, 2} or 6 in {5, 1, 2} or 0 in {1, -2, -1}: --> True
                        continue
                #iteration 5
                if 4 in {3, 0, 2} or 7 in {5, 1, 2} or -1 in {1, -2, -1}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q']]

        🏁🏁🤦🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {3, 0, 2}.remove(3) >> {0, 2}
                posDiag.remove(2 + 3) >> {1, 2}
                negDiag.remove(2 - 3) >> {1, -2}
                board[2][3] = '.'
>>
[['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 5
                if 4 in {0, 2} or 6 in {1, 2} or -2 in {1, -2}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {0, 2}.remove(0) >> {2}
                posDiag.remove(1 + 0) >> {2}
                negDiag.remove(1 - 0) >> {-2}
                board[1][0] = '.'
>>
[['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 2
                if 1 in {2} or 2 in {2} or 0 in {-2}: --> True
                        continue
                #iteration 3
                if 2 in {2} or 3 in {2} or -1 in {-2}: --> True
                        continue
                #iteration 4
                if 3 in {2} or 4 in {2} or -2 in {-2}: --> True
                        continue
                #iteration 5
                if 4 in {2} or 5 in {2} or -3 in {-2}: --> False
                {2}.add(4) >> {2, 4}
                posDiag.add(1 + 4) >> {2, 5}
                negDiag.add(1 - 4) >> {-2, -3}
                board[1][4] = 'Q'
>>
[['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
```

```
                if 0 in {2, 4} or 2 in {2, 5} or 2 in {-2, -3}: --> True
                        continue
                #iteration 2
                if 1 in {2, 4} or 3 in {2, 5} or 1 in {-2, -3}: --> False
                {2, 4}.add(1) >> {1, 2, 4}
                posDiag.add(2 + 1) >> {3, 2, 5}
                negDiag.add(2 - 1) >> {1, -2, -3}
                board[2][1] = 'Q'
>>
[['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {1, 2, 4} or 3 in {3, 2, 5} or 3 in {1, -2, -3}: --> True
                        continue
                #iteration 2
                if 1 in {1, 2, 4} or 4 in {3, 2, 5} or 2 in {1, -2, -3}: --> True
                        continue
                #iteration 3
                if 2 in {1, 2, 4} or 5 in {3, 2, 5} or 1 in {1, -2, -3}: --> True
                        continue
                #iteration 4
                if 3 in {1, 2, 4} or 6 in {3, 2, 5} or 0 in {1, -2, -3}: --> False
                {1, 2, 4}.add(3) >> {3, 1, 2, 4}
                posDiag.add(3 + 3) >> {6, 3, 2, 5}
                negDiag.add(3 - 3) >> {0, 1, -2, -3}
                board[3][3] = 'Q'
>>
[['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {3, 1, 2, 4} or 4 in {6, 3, 2, 5} or 4 in {0, 1, -2, -3}: --> False
                {3, 1, 2, 4}.add(0) >> {0, 3, 1, 2, 4}
                posDiag.add(4 + 0) >> {4, 6, 3, 2, 5}
                negDiag.add(4 - 0) >> {0, 1, 4, -2, -3}
                board[4][0] = 'Q'
>>
[['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(4+1) >> backtrack(5):
        if  r==n: --> True
                ??? #Base Case 🟡
                res.append(['.join(row) for row in board])
>>
[['Q....', '..Q..', '....Q', '.Q...', '...Q.'],
 ['Q....', '...Q.', '.Q...', '....Q', '..Q..'],
 ['.Q...', '...Q.', 'Q....', '..Q..', '....Q'],
 ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'],
 ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'],
 ['..Q..', '....Q', '.Q...', '...Q.', 'Q....']]
                return
                #Maximum recursive depth reached
                🔴 #End of recursive block🔴

                {0, 3, 1, 2, 4}.remove(0) >> {3, 1, 2, 4}
                posDiag.remove(4 + 0) >> {6, 3, 2, 5}
                negDiag.remove(4 - 0) >> {0, 1, -2, -3}
                board[4][0] = '.'
>>
```

```
[['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 2
                if 1 in {3, 1, 2, 4} or 5 in {6, 3, 2, 5} or 3 in {0, 1, -2, -3}: --> True
                        continue
                #iteration 3
                if 2 in {3, 1, 2, 4} or 6 in {6, 3, 2, 5} or 2 in {0, 1, -2, -3}: --> True
                        continue
                #iteration 4
                if 3 in {3, 1, 2, 4} or 7 in {6, 3, 2, 5} or 1 in {0, 1, -2, -3}: --> True
                        continue
                #iteration 5
                if 4 in {3, 1, 2, 4} or 8 in {6, 3, 2, 5} or 0 in {0, 1, -2, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {3, 1, 2, 4}.remove(3) >> {1, 2, 4}
                posDiag.remove(3 + 3) >> {3, 2, 5}
                negDiag.remove(3 - 3) >> {1, -2, -3}
                board[3][3] = '.'
>>
[['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 5
                if 4 in {1, 2, 4} or 7 in {3, 2, 5} or -1 in {1, -2, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {1, 2, 4}.remove(1) >> {2, 4}
                posDiag.remove(2 + 1) >> {2, 5}
                negDiag.remove(2 - 1) >> {-2, -3}
                board[2][1] = '.'
>>
[['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 3
                if 2 in {2, 4} or 4 in {2, 5} or 0 in {-2, -3}: --> True
                        continue
                #iteration 4
                if 3 in {2, 4} or 5 in {2, 5} or -1 in {-2, -3}: --> True
                        continue
                #iteration 5
                if 4 in {2, 4} or 6 in {2, 5} or -2 in {-2, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {2, 4}.remove(4) >> {2}
                posDiag.remove(1 + 4) >> {2}
                negDiag.remove(1 - 4) >> {-2}
```

```
                board[1][4] = '.'
>>
[['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....']]

        ▓▓ ▓▓ ▓▓ #End of a level ▓▓ ▓▓ ▓▓ ▓▓

                🔴 #End of recursive block🔴

                {2}.remove(2) >> set()
                posDiag.remove(0 + 2) >> set()
                negDiag.remove(0 - 2) >> set()
                board[0][2] = '.'
>>
[['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 4
                if 3 in set() or 3 in set() or -3 in set(): --> False
                set().add(3) >> {3}
                posDiag.add(0 + 3) >> {3}
                negDiag.add(0 - 3) >> {-3}
                board[0][3] = 'Q'
>>
[['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block
backtrack(0+1) >> backtrack(1):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {3} or 1 in {3} or 1 in {-3}: --> False
                {3}.add(0) >> {0, 3}
                posDiag.add(1 + 0) >> {1, 3}
                negDiag.add(1 - 0) >> {1, -3}
                board[1][0] = 'Q'
>>
[['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 3} or 2 in {1, 3} or 2 in {1, -3}: --> True
                        continue
                #iteration 2
                if 1 in {0, 3} or 3 in {1, 3} or 1 in {1, -3}: --> True
                        continue
                #iteration 3
                if 2 in {0, 3} or 4 in {1, 3} or 0 in {1, -3}: --> False
                {0, 3}.add(2) >> {2, 0, 3}
                posDiag.add(2 + 2) >> {4, 1, 3}
                negDiag.add(2 - 2) >> {0, 1, -3}
                board[2][2] = 'Q'
>>
[['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
```

```
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {2, 0, 3} or 3 in {4, 1, 3} or 3 in {0, 1, -3}: --> True
                        continue
                #iteration 2
                if 1 in {2, 0, 3} or 4 in {4, 1, 3} or 2 in {0, 1, -3}: --> True
                        continue
                #iteration 3
                if 2 in {2, 0, 3} or 5 in {4, 1, 3} or 1 in {0, 1, -3}: --> True
                        continue
                #iteration 4
                if 3 in {2, 0, 3} or 6 in {4, 1, 3} or 0 in {0, 1, -3}: --> True
                        continue
                #iteration 5
                if 4 in {2, 0, 3} or 7 in {4, 1, 3} or -1 in {0, 1, -3}: --> False
                {2, 0, 3}.add(4) >> {4, 2, 0, 3}
                posDiag.add(3 + 4) >> {7, 4, 1, 3}
                negDiag.add(3 - 4) >> {0, 1, -1, -3}
                board[3][4] = 'Q'
>>
[['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {4, 2, 0, 3} or 4 in {7, 4, 1, 3} or 4 in {0, 1, -1, -3}: --> True
                        continue
                #iteration 2
                if 1 in {4, 2, 0, 3} or 5 in {7, 4, 1, 3} or 3 in {0, 1, -1, -3}: --> False
                {4, 2, 0, 3}.add(1) >> {1, 4, 2, 0, 3}
                posDiag.add(4 + 1) >> {5, 7, 4, 1, 3}
                negDiag.add(4 - 1) >> {3, 0, 1, -1, -3}
                board[4][1] = 'Q'
>>
[['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(4+1) >> backtrack(5):
        if  r==n: --> True
                🟡 #Base Case 🟡
                res.append([''.join(row) for row in board])
>>
[['Q....', '..Q..', '....Q', '.Q...', '...Q.'],
 ['Q....', '...Q.', '.Q...', '....Q', '..Q..'],
 ['.Q...', '...Q.', 'Q....', '..Q..', '....Q'],
 ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'],
 ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'],
 ['..Q..', '....Q', '.Q...', '...Q.', 'Q....'],
 ['...Q.', 'Q....', '..Q..', '....Q', '.Q...']]
                return
                #Maximum recursive depth reached
                🔴 #End of recursive block🔴

                {1, 4, 2, 0, 3}.remove(1) >> {4, 2, 0, 3}
                posDiag.remove(4 + 1) >> {7, 4, 1, 3}
                negDiag.remove(4 - 1) >> {0, 1, -1, -3}
                board[4][1] = '.'
>>
[['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.']]
                #iteration 3
                if 2 in {4, 2, 0, 3} or 6 in {7, 4, 1, 3} or 2 in {0, 1, -1, -3}: --> True
                        continue
```

```
                #iteration 4
                if 3 in {4, 2, 0, 3} or 7 in {7, 4, 1, 3} or 1 in {0, 1, -1, -3}: --> True
                        continue
                #iteration 5
                if 4 in {4, 2, 0, 3} or 8 in {7, 4, 1, 3} or 0 in {0, 1, -1, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {4, 2, 0, 3}.remove(4) >> {2, 0, 3}
                posDiag.remove(3 + 4) >> {4, 1, 3}
                negDiag.remove(3 - 4) >> {0, 1, -3}
                board[3][4] = '.'
>>
[['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {2, 0, 3}.remove(2) >> {0, 3}
                posDiag.remove(2 + 2) >> {1, 3}
                negDiag.remove(2 - 2) >> {1, -3}
                board[2][2] = '.'
>>
[['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 4
                if 3 in {0, 3} or 5 in {1, 3} or -1 in {1, -3}: --> True
                        continue
                #iteration 5
                if 4 in {0, 3} or 6 in {1, 3} or -2 in {1, -3}: --> False
                {0, 3}.add(4) >> {4, 0, 3}
                posDiag.add(2 + 4) >> {6, 1, 3}
                negDiag.add(2 - 4) >> {1, -2, -3}
                board[2][4] = 'Q'
>>
[['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {4, 0, 3} or 3 in {6, 1, 3} or 3 in {1, -2, -3}: --> True
                        continue
                #iteration 2
                if 1 in {4, 0, 3} or 4 in {6, 1, 3} or 2 in {1, -2, -3}: --> False
                {4, 0, 3}.add(1) >> {1, 4, 0, 3}
                posDiag.add(3 + 1) >> {4, 6, 1, 3}
                negDiag.add(3 - 1) >> {2, 1, -2, -3}
                board[3][1] = 'Q'
>>
[['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
```

```
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {1, 4, 0, 3} or 4 in {4, 6, 1, 3} or 4 in {2, 1, -2, -3}: --> True
                        continue
                #iteration 2
                if 1 in {1, 4, 0, 3} or 5 in {4, 6, 1, 3} or 3 in {2, 1, -2, -3}: --> True
                        continue
                #iteration 3
                if 2 in {1, 4, 0, 3} or 6 in {4, 6, 1, 3} or 2 in {2, 1, -2, -3}: --> True
                        continue
                #iteration 4
                if 3 in {1, 4, 0, 3} or 7 in {4, 6, 1, 3} or 1 in {2, 1, -2, -3}: --> True
                        continue
                #iteration 5
                if 4 in {1, 4, 0, 3} or 8 in {4, 6, 1, 3} or 0 in {2, 1, -2, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {1, 4, 0, 3}.remove(1) >> {4, 0, 3}
                posDiag.remove(3 + 1) >> {6, 1, 3}
                negDiag.remove(3 - 1) >> {1, -2, -3}
                board[3][1] = '.'
>>
[['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 3
                if 2 in {4, 0, 3} or 5 in {6, 1, 3} or 1 in {1, -2, -3}: --> True
                        continue
                #iteration 4
                if 3 in {4, 0, 3} or 6 in {6, 1, 3} or 0 in {1, -2, -3}: --> True
                        continue
                #iteration 5
                if 4 in {4, 0, 3} or 7 in {6, 1, 3} or -1 in {1, -2, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {4, 0, 3}.remove(4) >> {0, 3}
                posDiag.remove(2 + 4) >> {1, 3}
                negDiag.remove(2 - 4) >> {1, -3}
                board[2][4] = '.'
>>
[['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {0, 3}.remove(0) >> {3}
                posDiag.remove(1 + 0) >> {3}
                negDiag.remove(1 - 0) >> {-3}
                board[1][0] = '.'
```

```
>>
[['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 2
                if 1 in {3} or 2 in {3} or 0 in {-3}: --> False
                {3}.add(1) >> {1, 3}
                posDiag.add(1 + 1) >> {2, 3}
                negDiag.add(1 - 1) >> {0, -3}
                board[1][1] = 'Q'
>>
[['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {1, 3} or 2 in {2, 3} or 2 in {0, -3}: --> True
                        continue
                #iteration 2
                if 1 in {1, 3} or 3 in {2, 3} or 1 in {0, -3}: --> True
                        continue
                #iteration 3
                if 2 in {1, 3} or 4 in {2, 3} or 0 in {0, -3}: --> True
                        continue
                #iteration 4
                if 3 in {1, 3} or 5 in {2, 3} or -1 in {0, -3}: --> True
                        continue
                #iteration 5
                if 4 in {1, 3} or 6 in {2, 3} or -2 in {0, -3}: --> False
                {1, 3}.add(4) >> {4, 1, 3}
                posDiag.add(2 + 4) >> {6, 2, 3}
                negDiag.add(2 - 4) >> {0, -2, -3}
                board[2][4] = 'Q'
>>
[['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {4, 1, 3} or 3 in {6, 2, 3} or 3 in {0, -2, -3}: --> True
                        continue
                #iteration 2
                if 1 in {4, 1, 3} or 4 in {6, 2, 3} or 2 in {0, -2, -3}: --> True
                        continue
                #iteration 3
                if 2 in {4, 1, 3} or 5 in {6, 2, 3} or 1 in {0, -2, -3}: --> False
                {4, 1, 3}.add(2) >> {2, 4, 1, 3}
                posDiag.add(3 + 2) >> {5, 6, 2, 3}
                negDiag.add(3 - 2) >> {1, 0, -2, -3}
                board[3][2] = 'Q'
>>
[['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {2, 4, 1, 3} or 4 in {5, 6, 2, 3} or 4 in {1, 0, -2, -3}: --> False
                {2, 4, 1, 3}.add(0) >> {0, 2, 4, 1, 3}
                posDiag.add(4 + 0) >> {4, 5, 6, 2, 3}
```

```
                negDiag.add(4 - 0) >> {1, 0, 4, -2, -3}
                board[4][0] = 'Q'
>>
[['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(4+1) >> backtrack(5):
        if  r==n: --> True
                🟡 #Base Case 🟡
                res.append([''.join(row) for row in board])
>>
[['Q....', '..Q..', '....Q', '.Q...', '...Q.'],
 ['Q....', '...Q.', '.Q...', '....Q', '..Q..'],
 ['.Q...', '...Q.', 'Q....', '..Q..', '....Q'],
 ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'],
 ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'],
 ['..Q..', '....Q', '.Q...', '...Q.', 'Q....'],
 ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'],
 ['...Q.', '.Q...', '....Q', '..Q..', 'Q....']]
                return
                #Maximum recursive depth reached
                🔴 #End of recursive block🔴
                {0, 2, 4, 1, 3}.remove(0) >> {2, 4, 1, 3}
                posDiag.remove(4 + 0) >> {5, 6, 2, 3}
                negDiag.remove(4 - 0) >> {1, 0, -2, -3}
                board[4][0] = '.'
>>
[['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 2
                if 1 in {2, 4, 1, 3} or 5 in {5, 6, 2, 3} or 3 in {1, 0, -2, -3}: --> True
                        continue
                #iteration 3
                if 2 in {2, 4, 1, 3} or 6 in {5, 6, 2, 3} or 2 in {1, 0, -2, -3}: --> True
                        continue
                #iteration 4
                if 3 in {2, 4, 1, 3} or 7 in {5, 6, 2, 3} or 1 in {1, 0, -2, -3}: --> True
                        continue
                #iteration 5
                if 4 in {2, 4, 1, 3} or 8 in {5, 6, 2, 3} or 0 in {1, 0, -2, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....']]
        🏁🏁🏁🏁 #End of a level 🏁🏁🏁

                🔴 #End of recursive block🔴

                {2, 4, 1, 3}.remove(2) >> {4, 1, 3}
                posDiag.remove(3 + 2) >> {6, 2, 3}
                negDiag.remove(3 - 2) >> {0, -2, -3}
                board[3][2] = '.'
>>
[['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 4
                if 3 in {4, 1, 3} or 6 in {6, 2, 3} or 0 in {0, -2, -3}: --> True
                        continue
                #iteration 5
                if 4 in {4, 1, 3} or 7 in {6, 2, 3} or -1 in {0, -2, -3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....']]
```

```
        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {4, 1, 3}.remove(4) >> {1, 3}
                posDiag.remove(2 + 4) >> {2, 3}
                negDiag.remove(2 - 4) >> {0, -3}
                board[2][4] = '.'
>>
[['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {1, 3}.remove(1) >> {3}
                posDiag.remove(1 + 1) >> {3}
                negDiag.remove(1 - 1) >> {-3}
                board[1][1] = '.'
>>
[['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 3
                if 2 in {3} or 3 in {3} or -1 in {-3}: --> True
                        continue
                #iteration 4
                if 3 in {3} or 4 in {3} or -2 in {-3}: --> True
                        continue
                #iteration 5
                if 4 in {3} or 5 in {3} or -3 in {-3}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {3}.remove(3) >> set()
                posDiag.remove(0 + 3) >> set()
                negDiag.remove(0 - 3) >> set()
                board[0][3] = '.'
>>
[['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 5
                if 4 in set() or 4 in set() or -4 in set(): --> False
                set().add(4) >> {4}
                posDiag.add(0 + 4) >> {4}
                negDiag.add(0 - 4) >> {-4}
                board[0][4] = 'Q'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(0+1) >> backtrack(1):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
```

```
                if 0 in {4} or 1 in {4} or 1 in {-4}: --> False
                {4}.add(0) >> {0, 4}
                posDiag.add(1 + 0) >> {1, 4}
                negDiag.add(1 - 0) >> {1, -4}
                board[1][0] = 'Q'
>>
[['.', '.', '.', '.', 'Q'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 4} or 2 in {1, 4} or 2 in {1, -4}: --> True
                        continue
                #iteration 2
                if 1 in {0, 4} or 3 in {1, 4} or 1 in {1, -4}: --> True
                        continue
                #iteration 3
                if 2 in {0, 4} or 4 in {1, 4} or 0 in {1, -4}: --> True
                        continue
                #iteration 4
                if 3 in {0, 4} or 5 in {1, 4} or -1 in {1, -4}: --> False
                {0, 4}.add(3) >> {3, 0, 4}
                posDiag.add(2 + 3) >> {5, 1, 4}
                negDiag.add(2 - 3) >> {1, -1, -4}
                board[2][3] = 'Q'
>>
[['.', '.', '.', '.', 'Q'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {3, 0, 4} or 3 in {5, 1, 4} or 3 in {1, -1, -4}: --> True
                        continue
                #iteration 2
                if 1 in {3, 0, 4} or 4 in {5, 1, 4} or 2 in {1, -1, -4}: --> True
                        continue
                #iteration 3
                if 2 in {3, 0, 4} or 5 in {5, 1, 4} or 1 in {1, -1, -4}: --> True
                        continue
                #iteration 4
                if 3 in {3, 0, 4} or 6 in {5, 1, 4} or 0 in {1, -1, -4}: --> True
                        continue
                #iteration 5
                if 4 in {3, 0, 4} or 7 in {5, 1, 4} or -1 in {1, -1, -4}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {3, 0, 4}.remove(3) >> {0, 4}
                posDiag.remove(2 + 3) >> {1, 4}
                negDiag.remove(2 - 3) >> {1, -4}
                board[2][3] = '.'
>>
[['.', '.', '.', '.', 'Q'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 5
                if 4 in {0, 4} or 6 in {1, 4} or -2 in {1, -4}: --> True
                        continue
```

```
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {0, 4}.remove(0) >> {4}
                posDiag.remove(1 + 0) >> {4}
                negDiag.remove(1 - 0) >> {-4}
                board[1][0] = '.'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 2
                if 1 in {4} or 2 in {4} or 0 in {-4}: --> False
                {4}.add(1) >> {1, 4}
                posDiag.add(1 + 1) >> {2, 4}
                negDiag.add(1 - 1) >> {0, -4}
                board[1][1] = 'Q'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {1, 4} or 2 in {2, 4} or 2 in {0, -4}: --> True
                        continue
                #iteration 2
                if 1 in {1, 4} or 3 in {2, 4} or 1 in {0, -4}: --> True
                        continue
                #iteration 3
                if 2 in {1, 4} or 4 in {2, 4} or 0 in {0, -4}: --> True
                        continue
                #iteration 4
                if 3 in {1, 4} or 5 in {2, 4} or -1 in {0, -4}: --> False
                {1, 4}.add(3) >> {3, 1, 4}
                posDiag.add(2 + 3) >> {5, 2, 4}
                negDiag.add(2 - 3) >> {0, -1, -4}
                board[2][3] = 'Q'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {3, 1, 4} or 3 in {5, 2, 4} or 3 in {0, -1, -4}: --> False
                {3, 1, 4}.add(0) >> {0, 3, 1, 4}
                posDiag.add(3 + 0) >> {3, 5, 2, 4}
                negDiag.add(3 - 0) >> {3, 0, -1, -4}
                board[3][0] = 'Q'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
```

```
                if 0 in {0, 3, 1, 4} or 4 in {3, 5, 2, 4} or 4 in {3, 0, -1, -4}: --> True
                        continue
                #iteration 2
                if 1 in {0, 3, 1, 4} or 5 in {3, 5, 2, 4} or 3 in {3, 0, -1, -4}: --> True
                        continue
                #iteration 3
                if 2 in {0, 3, 1, 4} or 6 in {3, 5, 2, 4} or 2 in {3, 0, -1, -4}: --> False
                {0, 3, 1, 4}.add(2) >> {2, 0, 3, 1, 4}
                posDiag.add(4 + 2) >> {6, 3, 5, 2, 4}
                negDiag.add(4 - 2) >> {2, 3, 0, -1, -4}
                board[4][2] = 'Q'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', 'Q', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(4+1) >> backtrack(5):
        if  r==n: --> True
                🟡 #Base Case 🟡
                res.append([''.join(row) for row in board])
>>
[['Q....', '..Q..', '....Q', '.Q...', '...Q.'],
 ['Q....', '...Q.', '.Q...', '....Q', '..Q..'],
 ['.Q...', '...Q.', 'Q....', '..Q..', '....Q'],
 ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'],
 ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'],
 ['..Q..', '....Q', '.Q...', '...Q.', 'Q....'],
 ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'],
 ['...Q.', '.Q...', '....Q', '..Q..', 'Q....'],
 ['....Q', '.Q...', '...Q.', 'Q....', '..Q..']]
                return
                #Maximum recursive depth reached
                🔴 #End of recursive block🔴

                {2, 0, 3, 1, 4}.remove(2) >> {0, 3, 1, 4}
                posDiag.remove(4 + 2) >> {3, 5, 2, 4}
                negDiag.remove(4 - 2) >> {3, 0, -1, -4}
                board[4][2] = '.'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 4
                if 3 in {0, 3, 1, 4} or 7 in {3, 5, 2, 4} or 1 in {3, 0, -1, -4}: --> True
                        continue
                #iteration 5
                if 4 in {0, 3, 1, 4} or 8 in {3, 5, 2, 4} or 0 in {3, 0, -1, -4}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....'],
['....Q', '.Q...', '...Q.', 'Q....', '..Q..']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁

                🔴 #End of recursive block🔴

                {0, 3, 1, 4}.remove(0) >> {3, 1, 4}
                posDiag.remove(3 + 0) >> {5, 2, 4}
                negDiag.remove(3 - 0) >> {0, -1, -4}
                board[3][0] = '.'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 2
                if 1 in {3, 1, 4} or 4 in {5, 2, 4} or 2 in {0, -1, -4}: --> True
                        continue
                #iteration 3
                if 2 in {3, 1, 4} or 5 in {5, 2, 4} or 1 in {0, -1, -4}: --> True
```

```
                        continue
                #iteration 4
                if 3 in {3, 1, 4} or 6 in {5, 2, 4} or 0 in {0, -1, -4}: --> True
                        continue
                #iteration 5
                if 4 in {3, 1, 4} or 7 in {5, 2, 4} or -1 in {0, -1, -4}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....'],
['....Q', '.Q...', '...Q.', 'Q....', '..Q..']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴🔴

                {3, 1, 4}.remove(3) >> {1, 4}
                posDiag.remove(2 + 3) >> {2, 4}
                negDiag.remove(2 - 3) >> {0, -4}
                board[2][3] = '.'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', 'Q', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 5
                if 4 in {1, 4} or 6 in {2, 4} or -2 in {0, -4}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....'],
['....Q', '.Q...', '...Q.', 'Q....', '..Q..']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴🔴 #End of recursive block🔴

                {1, 4}.remove(1) >> {4}
                posDiag.remove(1 + 1) >> {4}
                negDiag.remove(1 - 1) >> {-4}
                board[1][1] = '.'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 3
                if 2 in {4} or 3 in {4} or -1 in {-4}: --> False
                {4}.add(2) >> {2, 4}
                posDiag.add(1 + 2) >> {3, 4}
                negDiag.add(1 - 2) >> {-1, -4}
                board[1][2] = 'Q'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(1+1) >> backtrack(2):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {2, 4} or 2 in {3, 4} or 2 in {-1, -4}: --> False
                {2, 4}.add(0) >> {0, 2, 4}
                posDiag.add(2 + 0) >> {2, 3, 4}
                negDiag.add(2 - 0) >> {2, -1, -4}
                board[2][0] = 'Q'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
```

```
                🟢 #Starting Recursive block 🟢
backtrack(2+1) >> backtrack(3):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {0, 2, 4} or 3 in {2, 3, 4} or 3 in {2, -1, -4}: --> True
                        continue
                #iteration 2
                if 1 in {0, 2, 4} or 4 in {2, 3, 4} or 2 in {2, -1, -4}: --> True
                        continue
                #iteration 3
                if 2 in {0, 2, 4} or 5 in {2, 3, 4} or 1 in {2, -1, -4}: --> True
                        continue
                #iteration 4
                if 3 in {0, 2, 4} or 6 in {2, 3, 4} or 0 in {2, -1, -4}: --> False
                {0, 2, 4}.add(3) >> {3, 0, 2, 4}
                posDiag.add(3 + 3) >> {6, 2, 3, 4}
                negDiag.add(3 - 3) >> {0, 2, -1, -4}
                board[3][3] = 'Q'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(3+1) >> backtrack(4):
        if  r==n: --> False
        for c in range(5):
                #iteration 1
                if 0 in {3, 0, 2, 4} or 4 in {6, 2, 3, 4} or 4 in {0, 2, -1, -4}: --> True
                        continue
                #iteration 2
                if 1 in {3, 0, 2, 4} or 5 in {6, 2, 3, 4} or 3 in {0, 2, -1, -4}: --> False
                {3, 0, 2, 4}.add(1) >> {1, 3, 0, 2, 4}
                posDiag.add(4 + 1) >> {5, 6, 2, 3, 4}
                negDiag.add(4 - 1) >> {3, 0, 2, -1, -4}
                board[4][1] = 'Q'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', 'Q', '.', '.', '.']]
                🟢 #Starting Recursive block 🟢
backtrack(4+1) >> backtrack(5):
        if  r==n: --> True
                🟡 #Base Case 🟡
                res.append([''.join(row) for row in board])
>>
[['Q....', '..Q..', '....Q', '.Q...', '...Q.'],
 ['Q....', '...Q.', '.Q...', '....Q', '..Q..'],
 ['.Q...', '...Q.', 'Q....', '..Q..', '....Q'],
 ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'],
 ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'],
 ['..Q..', '....Q', '.Q...', '...Q.', 'Q....'],
 ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'],
 ['...Q.', '.Q...', '....Q', '..Q..', 'Q....'],
 ['....Q', '.Q...', '...Q.', 'Q....', '..Q..'],
 ['....Q', '..Q..', 'Q....', '...Q.', '.Q...']]
                return
                #Maximum recursive depth reached
                🔴 #End of recursive block🔴

                {1, 3, 0, 2, 4}.remove(1) >> {3, 0, 2, 4}
                posDiag.remove(4 + 1) >> {6, 2, 3, 4}
                negDiag.remove(4 - 1) >> {0, 2, -1, -4}
                board[4][1] = '.'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', 'Q', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 3
                if 2 in {3, 0, 2, 4} or 6 in {6, 2, 3, 4} or 2 in {0, 2, -1, -4}: --> True
```

```
                        continue
                #iteration 4
                if 3 in {3, 0, 2, 4} or 7 in {6, 2, 3, 4} or 1 in {0, 2, -1, -4}: --> True
                        continue
                #iteration 5
                if 4 in {3, 0, 2, 4} or 8 in {6, 2, 3, 4} or 0 in {0, 2, -1, -4}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....'],
['....Q', '.Q...', '...Q.', 'Q....', '..Q..'], ['....Q', '..Q..', 'Q....', '...Q.', '.Q...']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {3, 0, 2, 4}.remove(3) >> {0, 2, 4}
                posDiag.remove(3 + 3) >> {2, 3, 4}
                negDiag.remove(3 - 3) >> {2, -1, -4}
                board[3][3] = '.'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['Q', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 5
                if 4 in {0, 2, 4} or 7 in {2, 3, 4} or -1 in {2, -1, -4}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....'],
['....Q', '.Q...', '...Q.', 'Q....', '..Q..'], ['....Q', '..Q..', 'Q....', '...Q.', '.Q...']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴🔴

                {0, 2, 4}.remove(0) >> {2, 4}
                posDiag.remove(2 + 0) >> {3, 4}
                negDiag.remove(2 - 0) >> {-1, -4}
                board[2][0] = '.'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', '.', 'Q', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 2
                if 1 in {2, 4} or 3 in {3, 4} or 1 in {-1, -4}: --> True
                        continue
                #iteration 3
                if 2 in {2, 4} or 4 in {3, 4} or 0 in {-1, -4}: --> True
                        continue
                #iteration 4
                if 3 in {2, 4} or 5 in {3, 4} or -1 in {-1, -4}: --> True
                        continue
                #iteration 5
                if 4 in {2, 4} or 6 in {3, 4} or -2 in {-1, -4}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....'],
['....Q', '.Q...', '...Q.', 'Q....', '..Q..'], ['....Q', '..Q..', 'Q....', '...Q.', '.Q...']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {2, 4}.remove(2) >> {4}
                posDiag.remove(1 + 2) >> {4}
                negDiag.remove(1 - 2) >> {-4}
                board[1][2] = '.'
>>
[['.', '.', '.', '.', 'Q'],
 ['.', '.', '.', '.', '.'],
```

```
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
                #iteration 4
                if 3 in {4} or 4 in {4} or -2 in {-4}: --> True
                        continue
                #iteration 5
                if 4 in {4} or 5 in {4} or -3 in {-4}: --> True
                        continue
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....'],
['....Q', '.Q...', '...Q.', 'Q....', '..Q..'], ['....Q', '..Q..', 'Q....', '...Q.', '.Q...']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

                🔴 #End of recursive block🔴

                {4}.remove(4) >> set()
                posDiag.remove(0 + 4) >> set()
                negDiag.remove(0 - 4) >> set()
                board[0][4] = '.'
>>
[['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.'],
 ['.', '.', '.', '.', '.']]
        return [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.',
'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..',
'....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q', '..Q..', 'Q....'],
['....Q', '.Q...', '...Q.', 'Q....', '..Q..'], ['....Q', '..Q..', 'Q....', '...Q.', '.Q...']]

        🏁🏁🏁🏁 #End of a level 🏁🏁🏁🏁

# final output
[['Q....', '..Q..', '....Q', '.Q...', '...Q.'],
 ['Q....', '...Q.', '.Q...', '....Q', '..Q..'],
 ['.Q...', '...Q.', 'Q....', '..Q..', '....Q'],
 ['.Q...', '....Q', '..Q..', 'Q....', '...Q.'],
 ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'],
 ['..Q..', '....Q', '.Q...', '...Q.', 'Q....'],
 ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'],
 ['...Q.', '.Q...', '....Q', '..Q..', 'Q....'],
 ['....Q', '.Q...', '...Q.', 'Q....', '..Q..'],
 ['....Q', '..Q..', 'Q....', '...Q.', '.Q...']]
```