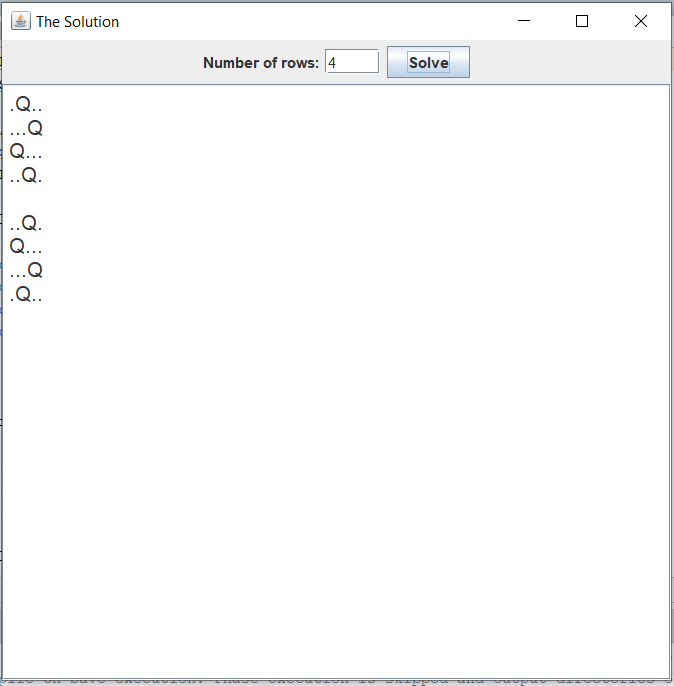
# **Project#4: N-Queens**

The definition of the problem:

The N-Queens problem is a classic problem in computer science and mathematics that involves placing N chess queens on an N×N chessboard in such a way that no two queens threaten each other. In other words, no two queens can share the same row, column, or diagonal.

The challenge is to find all possible arrangements of N queens on the chessboard that satisfy the constraint of non-attack. The N-Queens problem is a well-known example of a constraint satisfaction problem and has applications in algorithm design, artificial intelligence, and combinatorial optimization.

The problem becomes increasingly complex as the size of the chessboard (N) increases, and finding a solution for larger values of N can be computationally challenging. There are various algorithms and techniques, such as backtracking, to solve the N-Queens problem efficiently.



Design Overview:

Chessboard Representation: The chessboard is represented as a 2D array, where chessboard[row][col] is 1 if there is a queen at position (row, col) and 0 otherwise.

Threaded Exploration: Each thread is responsible for placing queens in a specific row, exploring different possibilities concurrently.

Concurrency Control: The synchronized keyword is used to control access to shared data structures, such as the chessboard

The history of the N-Queens problem:

The history of the N-Queens problem dates back to the mid-19th century. The problem is often attributed to the German chess enthusiast Max Bezzel, who posed it in 1848. However, the problem gained wider recognition when it was published by the chess player and mathematician Franz Nauck in 1850. Nauck presented the 8-Queens problem in a German chess magazine called "Schachzeitung der Berliner Schachgesellschaft."

In the original problem, the task was to place eight queens on an 8x8 chessboard in such a way that no two queens threaten each other. Over the years, the N-Queens problem has become a well-known and studied problem in the fields of mathematics and computer science, particularly in the context of combinatorial optimization and algorithm design.

The generalization of the problem to an N×N chessboard, where N is not limited to 8, allows for a broader exploration of algorithms and strategies to solve it. Various mathematical techniques and computer science algorithms, such as backtracking and constraint satisfaction methods, have been applied to find solutions efficiently and to explore the properties of the problem for different values of N.

The N-Queens problem has not only served as an interesting puzzle for mathematicians and computer scientists but has also found practical applications in testing and benchmarking algorithms, especially those related to constraint satisfaction and combinatorial problem-solving. It remains a classic problem that is often used in educational settings to teach algorithmic thinking and problem-solving skills.

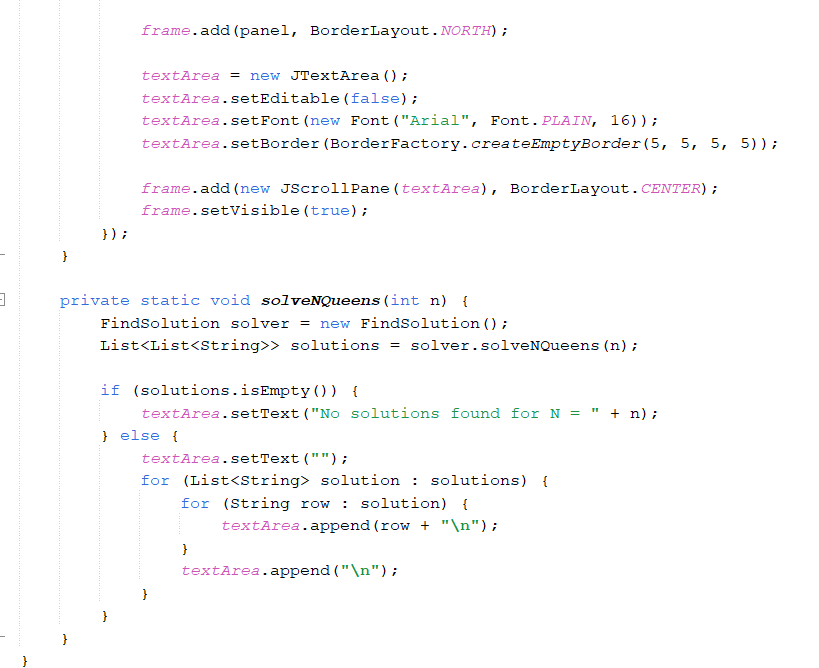
Constraints and Condition for the problem :

1. **Non-Attacking Queens:** The primary constraint is that no two queens should be able to attack each other. This means that no two queens can be in the same row, column, or diagonal.
2. **Row Constraint:** Each row must contain exactly one queen. This ensures that no two queens share the same row.
3. **Column Constraint:** Each column must contain exactly one queen. This ensures that no two queens share the same column.
4. **Diagonal Constraint:** No two queens should share the same diagonal. This includes both the main diagonals and the counter-diagonals.
5. **Unique Solutions:** In the context of finding solutions, the goal is often to find all unique arrangements of queens on the chessboard that satisfy the above constraints.
6. **Square Chessboard:** The chessboard is assumed to be square, meaning it has the same number of rows and columns. The size of the chessboard is represented by the variable N, which is the number of queens and the dimensions of the N×N board.

The challenge is to find a placement of queens on the chessboard that satisfies all these constraints. Various algorithms, such as backtracking, are commonly used to solve the N-Queens problem by exploring different arrangements and checking if they meet the specified conditions.

The algorithm to solve N Queens problem:



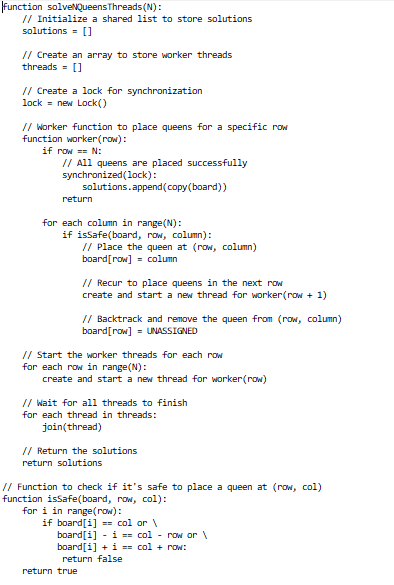


In this example:

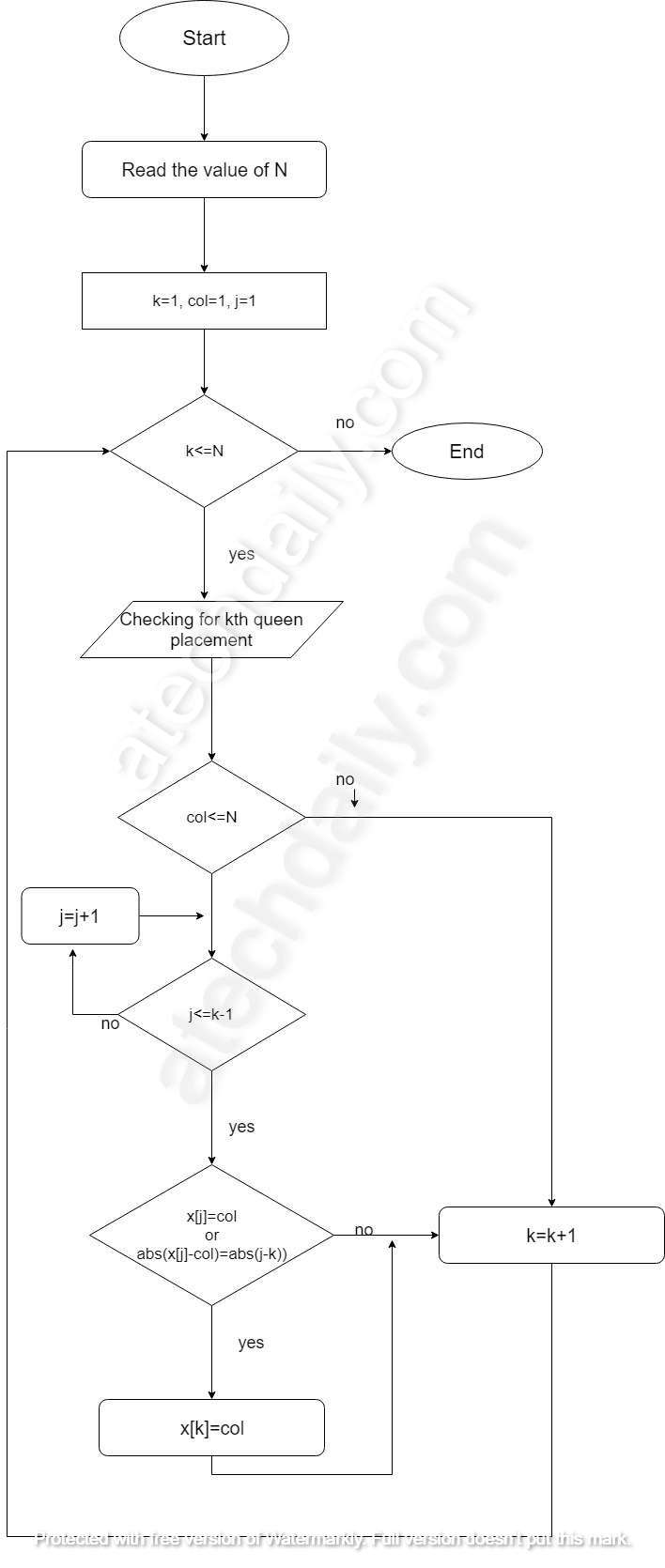
* The solveNQueens initializes a list of solutions and creates worker threads for each row.
* Each worker thread represents a row and explores possible placements for the queen.
* The **Findsolution** class contains the logic for solving the N-Queens problem for a specific row.
* The **synchronized** block is used to safely add solutions to the shared list.

Note that this example might not be the most efficient way to parallelize the N-Queens problem due to the overhead of thread creation and synchronization. Depending on the specific requirements and constraints, more sophisticated parallelization techniques may be considered

Pseudocode:



Note: This pseudocode is a high-level representation of the algorithm. The actual implementation in a specific programming language would require additional details, such as thread creation and synchronization mechanisms, which may vary based on the programming language and libraries used.

Flowchart:

Deadlock:

A deadlock is a situation in computing where two or more processes are unable to proceed because each is waiting for the other to release a resource. In other words, a deadlock is a state in a system where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Deadlocks can occur in various concurrent systems, including multi-threaded applications and distributed systems. They can result from a combination of resource contention and the absence of proper synchronization mechanisms. The four necessary conditions for a deadlock to occur are:

1. **Mutual Exclusion:** At least one resource must be held in a non-sharable mode, meaning only one process can use the resource at a time.
2. **Hold and Wait:** A process must be holding at least one resource and waiting to acquire additional resources held by other processes.
3. **No Preemption:** Resources cannot be forcibly taken away from a process. They must be released voluntarily by the process holding them.
4. **Circular Wait:** A set of processes must exist such that each process is waiting for a resource held by the next process in the set.

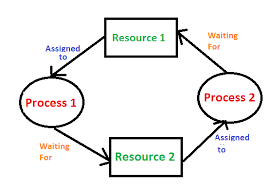
When all four conditions are satisfied, a deadlock can occur, and the system may reach a state where no progress is possible.

Example (using two processes, P1 and P2, and two resources, R1 and R2):

1. P1 acquires R1.
2. P2 acquires R2.
3. P1 requests R2 but is blocked because P2 holds it.
4. P2 requests R1 but is blocked because P1 holds it.

At this point, both processes are blocked, and a deadlock occurs.

Preventing deadlocks involves addressing one or more of the necessary conditions. Techniques such as resource allocation graphs, deadlock detection, and deadlock recovery can be employed to manage and mitigate the impact of deadlocks in a system.



In N-Queens problem:

In the context of the N-Queens problem, it's not common to encounter deadlock issues because the typical algorithms used to solve the N-Queens problem, such as backtracking, do not involve the kind of resource allocation and synchronization that can lead to deadlocks.

The N-Queens problem is a combinatorial problem where the challenge is to find a safe placement of queens on a chessboard. The algorithms for solving the N-Queens problem usually rely on recursive exploration of the solution space and do not involve the kind of resource contention that leads to deadlocks.

However, it's important to note that if you're implementing a parallel or concurrent version of an algorithm to solve the N-Queens problem, you need to be cautious about synchronization issues. Deadlocks could potentially occur if you are not careful about how threads acquire and release resources or if you use inappropriate synchronization mechanisms.

For example, if you use locks or other synchronization primitives to manage access to shared data structures, you need to ensure that the locking strategy doesn't create a circular wait condition, which is one of the conditions necessary for a deadlock.

In summary, while the N-Queens problem itself doesn't inherently lead to deadlocks, the specific implementation details, especially in a concurrent or parallel setting, need to be carefully managed to avoid potential synchronization issues.

Examples of Deadlock:

*In A real-world example:*

would be traffic, which is going only in one direction. Here, a bridge is considered a resource.

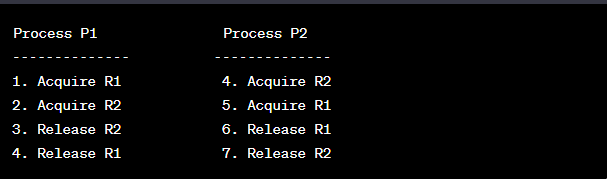
*In operating system:*

Deadlock happens when two or more process need some resource to complete their execution that is held by the other process.

*N-Queens:*

Certainly, let's go through a simple example of a deadlock scenario using a basic resource allocation scenario. Although this example doesn't directly relate to the N-Queens problem, it illustrates the conditions that can lead to a deadlock.

Consider two processes, P1 and P2, and two resources, R1 and R2. The processes need to acquire both resources to complete their tasks.

**

Now, let's analyze the steps:

1. P1 acquires R1.
2. P2 acquires R2.
3. P1 is waiting for R2, held by P2.
4. P2 is waiting for R1, held by P1.

At this point, both processes are waiting for a resource held by the other, forming a circular wait. This situation satisfies the necessary conditions for a deadlock: mutual exclusion (resources can only be held by one process at a time), hold and wait (processes hold resources and wait for more), no preemption (resources cannot be forcibly taken away), and circular wait.

In a real-world scenario with proper resource management, processes would release resources after completing their tasks, preventing deadlocks. However, if resource allocation and release are not managed carefully, deadlocks can occur. Techniques such as deadlock detection and prevention mechanisms are used in operating systems to address these issues.

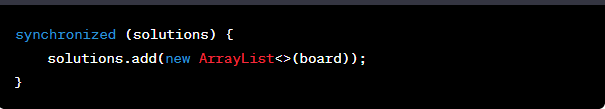
Explanation of solution for deadlock problem in our code:

In the context of the N-Queens problem solution using threads, the potential for deadlocks is relatively low, as the algorithms typically involve backtracking and recursion, without complex resource allocation and locking mechanisms. However, when working with threads, it's always a good practice to be aware of synchronization issues.

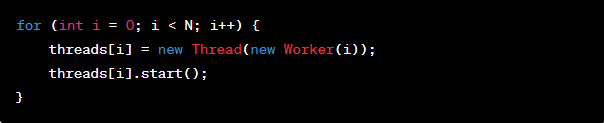
In the provided Java code example for the N-Queens problem with threads, I didn't explicitly use locks or other synchronization primitives that could lead to deadlocks. Each worker thread independently explores a portion of the solution space and updates its local data structures without interfering with other threads.

Here are some aspects of the code that help mitigate deadlock concerns:

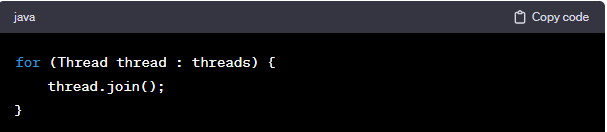
1. **Synchronization on Shared Data:**
   * The **synchronized** block is used when adding solutions to the shared **solutions** list. This ensures that multiple threads do not interfere with each other when updating the shared data structure.

**

1. **Independent Worker Threads:**
   * Each worker thread is responsible for a specific row and explores possibilities independently. This reduces the likelihood of contention for shared resources



1. **Joining Threads:**
   * After creating the threads, the program waits for all threads to finish using the **join** method. This ensures that the main program doesn't proceed until all worker threads have completed their tasks



While this example is designed to minimize the chances of deadlocks, it's important to note that in more complex multithreaded applications, careful consideration of shared resources, proper synchronization, and avoidance of circular dependencies is crucial to prevent deadlocks. In scenarios where locks are used, it's important to follow best practices to minimize the risk of deadlocks, such as acquiring locks in a consistent order and releasing them in a timely manner.