CS300 - Fall 2024-2025 Sabancı University Homework #1
N-Queen Problem

Due date: 27 October 2024 23:59

1 Introduction

The N-Queens problem is a generalized version of the 8-Queens problem, a mathematical puzzle first published in 1848 by the German chess composer Max Bezzel. The objective is to place N queens on an NxN chessboard such that no two queens threaten each other. In chess, a queen can move horizontally, vertically, and diagonally. For this homework, you will implement a computer program that utilizes **Stacks** to determine and display all possible solutions to the N-Queens problem. Your stack implementation **MUST** be template-based, allowing you to store either basic data types or custom structs to facilitate solving the problem. The exact format for displaying the solutions will be specified later in this document.

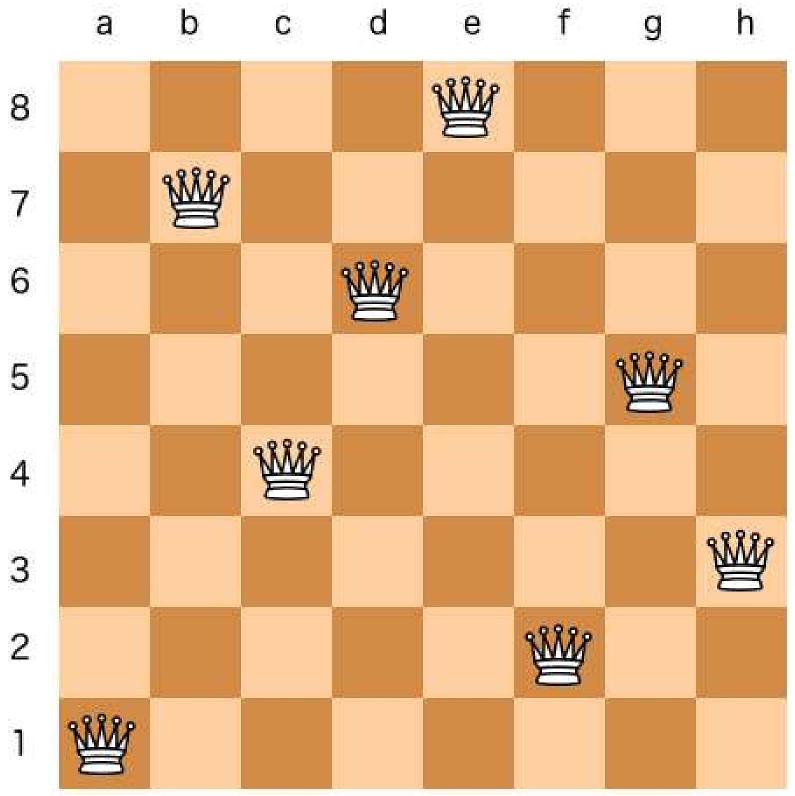

Figure 1: An example solution for the 8 Queen problem

2 Program Flow

In this problem, we need to find all possible arrangements of N queens on an NxN chessboard such that no two queens threaten each other. We'll use a stack-based approach to implement the backtracking algorithm.

2.1 Algorithm Overview

1. Start with an empty chess board.
2. Begin placing Queens row by row starting from index 0.
3. For each row, do the following:
   (a) Try placing a queen in each column.
   (b) If a safe position is found, move to the next row.
   (c) If no safe positions are found, **Backtrack** to the previous row.
4. Repeat until all N queens are placed and all solutions are found.

2.2 Backtracking with a Stack

We'll use a stack to manage the backtracking process. This approach allows us to efficiently explore all possibilities without using recursion. Here's how it works:

1. Initialize an empty stack.
2. Push the initial state (first row, no queens placed) onto the stack.
3. While the stack is not empty:
   (a) Examine the top state on the stack.
   (b) If a solution is found (N queens placed), record it.
   (c) If a safe position is found in the current row:
      - Update the board state.
      - Push a new state for the next row onto the stack.
   (d) If no safe position is found or we've tried all columns:
      - Pop the current state from the stack (backtrack).
      - If the stack is not empty, try the next column in the previous row.
4. Continue until the stack is empty, which means all possibilities have been explored.

By using a stack, we can easily backtrack when we reach a dead end. Popping a state from the stack effectively undoes the last queen placement, allowing us to try a different position.

2.3 Optimizing with a Safe Matrix

We strongly suggest you to use a safe matrix to track available positions across the chess board. To use safe matrix, let's define what corresponds to a safe placement in the N-Queens problem:

Safe Placement Definition

A position (row, column) on the chessboard is considered **safe** for queen placement if and only if:

- No other queen is present in the same row
- No other queen is present in the same column
- No other queen is present on the same diagonal

To efficiently determine whether a position is safe for queen placement, we recommend using a "safe matrix." This is a 2D matrix that keeps track of how many queens are attacking each position on the board.

1. Initialize the safe matrix with all zeros, indicating all positions are initially safe.
2. When placing a queen:
   - Increment the count in the safe matrix for all positions the queen attacks (same row, column, and diagonals).
3. When removing a queen (backtracking):
   - Decrement the count in the safe matrix for all positions the queen was attacking.
4. To check if a position is safe:
   - Simply check if the corresponding value in the safe matrix is zero.

You may think safe matrix as **chess board** directly. Using a safe matrix allows for constant-time checking of whether a position is safe, instead of having to check all previously placed queens each time. This optimization significantly improves the algorithm's efficiency, especially for larger board sizes.

You should implement and maintain this safe matrix throughout the solution to achieve optimal performance. As for some N

2.4 Input validation

Your program should prompt the user to input a value for N, where N must be greater than 0. If the user inputs a value of N that is less than or equal to 0, it should be considered invalid, as a chessboard cannot have zero or negative tiles. In such cases, the program should display an appropriate message to inform the user that the input is invalid, and it should not proceed with generating solutions and terminate immediately. You can check Sample Runs for the format of this message. Additionally, the input will always be an integer, so there is no need to check whether the user inputs a double, character, or string. You may also assume that $N < 14$.

2.5 Output Format

Your program should find all the solutions and output them to a text file named `X_solutions.txt`, where `X` is the value inputted for `N`. The text file should first print the total number of solutions found, followed by each solution on its own line. For example, if the program finds 4 solutions, the file should contain 6 lines: one for the number of solutions one empty line and four for the solutions themselves.

Each solution line should begin with the phrase `"Solution {i}:"`, where `i` represents the solution number (from 1 to n). After this, an array should represent the snapshot of the board's state. This array should be one-dimensional, where the indices represent the rows, and each number at a given index corresponds to the column where the queen is placed in that row. In this array, the first column is represented by 0 and the last column by `N-1`. An example output case is as follows:

```
Total solutions for N=4: 2

Solution 1: [1, 3, 0, 2]
Solution 2: [2, 0, 3, 1]
```

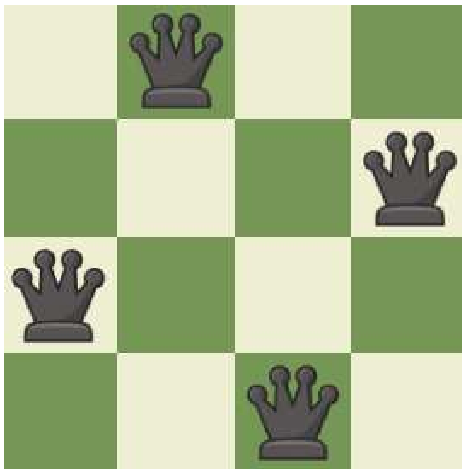
Figure 2: Solution 1

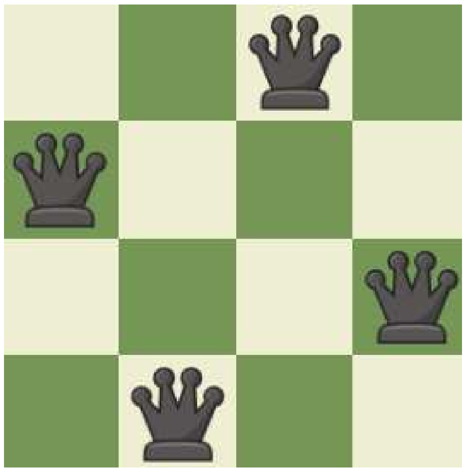
Figure 3: Solution 2

3 Sample Runs

Sample Run 1

```
Enter the value of N for the N-Queens problem: 6
Solutions have been saved to '6queens_solutions.txt'
```

Content of 6queens_solutions.txt:

```
Total solutions for N=6: 4

Solution 1: [1, 3, 5, 0, 2, 4]
Solution 2: [2, 5, 1, 4, 0, 3]
Solution 3: [3, 0, 4, 1, 5, 2]
Solution 4: [4, 2, 0, 5, 3, 1]
```

Sample Run 2

```
Enter the value of N for the N-Queens problem: 0
Invalid input. Please enter a positive integer
```

4 General Rules and Guidelines about Homeworks

The following rules and guidelines will be applicable to all homeworks, unless otherwise noted.

4.1 How to get help?

You may ask questions to TAs (Teaching Assistants) of CS300. Office hours of TAs can be found at SUCourse.

4.2 What and Where to Submit

Students are expected to strictly follow the submission guidelines to ensure a smooth grading process. Failure to comply with these rules may result in a deduction of 5 or more penalty points depending on the severity of the issue.

4.2.1 Submission Guidelines

- You should submit **exactly 3 files**: `stack.h`, `stack.cpp`, which are the stack implementations, and `main.cpp`.
- The files must be named `stack.h`, `stack.cpp`, and `main.cpp` precisely as specified. Any deviation from these names will result in a **grade of 0**.
- Do **not** compress or zip the files. Submit the 3 files directly.
- Do not add any additional characters or phrases to the file name.
- Ensure that this file is the latest version of your homework program.

4.2.2 Submission Method

- Submit via `SUCourse` ONLY.
- Submissions through e-mail or other means (e.g., paper) will receive no credit.
- Successful submission is a requirement of the homework. If the submission cannot be successfully graded, the grade will be **0**.

4.3 Grading and Objections

Your programs should strictly follow the guidelines regarding input and output order. Additionally, you must use the exact same prompts as provided in the Sample Runs. Failure to do so may result in a failed grading process, which could lead to receiving a grade of zero or losing points.

4.3.1 Grading

- Your code should compile in order to be assessed. Any code that fails to compile will automatically receive a grade of **0**.
- You must submit exactly 3 files: **stack.h**, **stack.cpp** and **main.cpp**. Submitting files with different names will not be considered and may result in a grade of **0**.
- You must have a correct `Stack` implementation in your code. Even if your program produces the correct output, we will check the functionality of your `Stack` implementation. If your `Stack` does not work correctly, points will be deducted.
- Having a correct program is necessary, but not sufficient to earn the full grade. Factors that will affect your grade include:

  - Proper comments and indentation.
  - Meaningful and understandable identifier names.
  - Informative introduction and prompts.
  - Efficient use of required functions.
  - Avoiding unnecessarily long programs and code duplications.

- Submit your own work, even if it is not fully functional. Submitting similar programs is easily detectable.

4.3.2 Grade Announcements

- Grades will be posted on SUCourse, and you will receive an announcement simultaneously.
- The grading policy and test cases will be included in the announcement.

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
CS300 Team