Midterm Project: N-queens on RaspberryPiZero

The N-Queens problem is a classic challenge in artificial intelligence and computational theory, requiring a solution to the task of placing N queens on an N x N chessboard in such a way that no two queens threaten each other. This problem has factorial complexity O(n!), meaning that the number of possible arrangements grows exponentially with the size of the board. In the early days of AI research, solving the N-Queens problem for even medium values of N posed a considerable challenge due to the limitations of brute-force computation. Even today, solving it with large board sizes requires intelligent algorithmic design, especially when working with resource-constrained hardware.

This project aims to address the N-Queens problem efficiently on a Raspberry Pi Zero, a single-board computer with minimal computational resources: 512MB of RAM and a single-core processor clocked at under 1GHz. The Pi Zero's limited power requires careful selection and optimization of algorithms, as well as mindful management of both processing load and memory. Through this work, I intend to explore the trade-offs between algorithmic efficiency and hardware constraints, taking a streamlined approach that maximizes performance within the Pi Zero's restricted capabilities.

This choice of hardware and approach also aligns with my longstanding interest in retro computing. Working with older or simpler devices, like the Pi Zero, provides unique insights into algorithm efficiency and code optimization, as every byte and cycle becomes meaningful. This project is a step towards understanding how to design algorithms that perform well under strict limitations, offering insights not only into solving specific problems like N-Queens but also into broader principles of efficiency and resourcefulness in computing.

Methodology To solve the N-Queens problem on a constrained device like the Raspberry Pi Zero, I tested and evaluated multiple algorithms, ultimately implementing backtracking for its efficiency. Initially, I attempted to solve the problem using standard depth-first search (DFS), a systematic approach that explores every potential arrangement of queens before backtracking. However, DFS proved highly inefficient for larger board sizes, taking an impractical amount of time to complete solutions for boards larger than 10 x 10. DFS lacked the capacity to eliminate unpromising solution paths early, causing it to waste computational cycles on infeasible configurations.

Recognizing this limitation, I shifted to a backtracking approach. Backtracking is an algorithmic method that allows the system to identify and discard nonviable paths as soon as they are identified, preventing further exploration down dead-end branches. In the context of the N-Queens problem, backtracking quickly eliminates arrangements where queens are in attacking positions, avoiding unnecessary recursive calls and saving both memory and processing time. This early pruning of the search tree enabled the algorithm to handle significantly larger board sizes within a reasonable time frame on the Pi Zero.

Within the backtracking framework, I applied optimizations to reduce the already low resource usage further. For example, the algorithm was modified to efficiently manage memory by tracking queen placements in a minimalistic way, avoiding unnecessary allocations. These optimizations helped achieve a solution for board sizes up to 22 x 22 in approximately 30 seconds—a substantial improvement over the initial DFS implementation.

Backtracking alone provided a sufficient solution on this hardware, and extensive additional optimizations were unnecessary. The methodology for this project demonstrates how a shift to a more efficient algorithm can make a computationally challenging problem feasible on a limited device, reaffirming the effectiveness of backtracking for problems requiring both depth and pruning in the search space.

The transition from depth-first search (DFS) to backtracking yielded substantial improvements in performance, enabling the Raspberry Pi Zero to solve the N-Queens problem efficiently up to a board size of 22. With backtracking, the algorithm achieved a runtime of only 5 seconds for a 20 x 20 board, demonstrating the effectiveness of early pruning to reduce the search space. For a 22 x 22 board, the runtime increased to around 30 seconds, a manageable timeframe given the constraints of the hardware.

For board sizes larger than 22 x 22, however, the Pi Zero faced significant limitations. Boards larger than 22 often required several minutes to complete, and some configurations did not finish at all, reflecting the limits of the Pi Zero's single-core processor and 512MB of RAM. At these sizes, the exponential increase in potential configurations began to outpace the Pi Zero's capacity, illustrating the practical boundary of what this device can achieve with backtracking alone.

Comparing these results to the earlier DFS implementation, the difference was striking. While DFS could solve smaller boards, it struggled to handle any size above 10, where runtimes became unacceptably long due to the lack of path pruning. The transition to backtracking not only reduced the runtime for board sizes over 10, but it also allowed the Pi Zero to reach the upper threshold of 22 x 22 within reasonable timeframes.

These results highlight the performance gains achievable through algorithmic optimization, especially on constrained hardware. By utilizing backtracking, I was able to push the computational limits of the Pi Zero and solve board sizes well beyond the initial performance limitations encountered with DFS.

This project demonstrates how careful algorithm selection and optimization can enable a constrained device like the Raspberry Pi Zero to tackle computationally intensive problems such as the N-Queens challenge. Starting with an attempt using depth-first search (DFS), I found that the limitations of this approach—particularly its inability to eliminate infeasible paths—made it impractical for larger board sizes. Transitioning to a backtracking algorithm proved highly effective, allowing for significant improvements in performance by pruning dead-end paths early. This shift enabled the Pi Zero to handle board sizes up to 22 x 22, with runtimes of around 5 seconds for a 20 x 20 board and 30 seconds for a 22 x 22 board. For board sizes above 22, however, the device's hardware constraints limited further scalability, as solution times increased to several minutes or failed to complete.

In future work, there is potential to explore multi-device solutions with clustered Pi Zeros or experiment with legacy systems like the Sun SparcStation. This project has provided valuable insights into the relationship between algorithm design and hardware limitations, emphasizing how thoughtful problem-solving techniques can unlock unexpected capabilities even on minimalistic platforms.