VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

Mini Project Report
on

Backtracking – Nqueens problem, Sum of subsets

Submitted in partial fulfillment of the requirements for the Analysis and Design of Algorithms Laboratory (BCSL404) for the IV Semester of the Bachelor of Engineering in Information Science and Engineering, Visvesvaraya Technological University, Belagavi.

Submitted by

Suhas Kumar - 1RN22IS155

Suhas Varna - 1RN22IS158

Tanmay T M - 1RN22IS162

Under the Guidance of

Ms. Varshitha M K
Assistant Professor
Department of ISE

ESTD:2001
An Institute with a Difference

Department of Information Science and Engineering

RNS Institute of Technology
Dr. Vishnuvardhan Road, Rajarajeshwari Nagar post,
Channa Sandra, Bengaluru-560098

2023-24

RNS INSTITUTE OF TECHNOLOGY

Dr. Vishnuvardhan Road, Rajarajeshwari Nagar post,

Channa Sandra, Bengaluru - 560098

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work titled **"Backtracking – Nqueens problem, Sum of subsets" has** been successfully completed by **Suhas Kumar (1RN22IS155), Suhas Varna (1RN22IS158), Tanmay T M(1RN22IS162),** bonafide students of **RNS Institute of Technology, Bengaluru** in partial fulfillment of the requirements as part of **Analysis and Design of Algorithms Laboratory** [BCSL404] for the award of degree in **Bachelor of Engineering in Information Science and Engineering** of **Visvesvaraya Technological University, Belagavi** during the academic year **2023-2024**.

_____

Ms. Varshitha M K
Assistant Professor,
Department of ISE

ACKNOWLEDGMENT

The fulfillment and joy associated with the successful completion of any assignment would be incomplete without acknowledging the individuals who made it possible. Their consistent guidance and support have crowned our efforts with success.

We would like to express my profound gratitude to the **Management of RNS Institute of Technology** for providing a conducive environment to conduct the mini project work.

We would like to express our gratitude to our Director **Dr. M K Venkatesha**, and our Principal **Dr. Ramesh Babu H S** for their support and inspiration towards the pursuit of knowledge.

We wish to express our heartfelt gratitude to **Dr. Suresh L**, Professor and Head of the Department of Information Science and Engineering, for being the driving force and mastermind behind our project.

We extend our heartfelt thanks to **Ms. Varshitha M K**, Assistant Professor, Department of Information Science and Engineering, for guiding and evaluating the project.

We would like to thank all teaching and non-teaching staff of Information Science & Engineering department, who have directly or indirectly helped us to carry out the project.

Place: Bengaluru

Date:

Suhas Kumar [1RN22IS155]

Suhas Varna [1RN22IS158]

Tanmay T M [1RN22IS162]

Table of Contents

ABSTRACT

This project explores the use of backtracking algorithms through the N-Queens and Sum of Subsets problems. Backtracking is a methodical way of trying out various possibilities to solve problems that involve searching through a large set of potential solutions. We provide detailed explanations, step-by-step procedures, and code examples for each problem, demonstrating how backtracking is applied. Specifically, for the N-Queens problem, we accept 'n' queens, represent them on an NxN chess board, and print the board. For the Sum of Subsets problem, we accept a set S and a subset sum D, printing all possible subset sum solutions. We analyse the performance and complexity of these algorithms to understand their efficiency. By comparing the two problems, we highlight the strengths and limitations of backtracking. The goal is to gain a deep understanding of backtracking's effectiveness in solving complex problems by the above two methods.

Chapter 1

INTRODUCTION

1.1 Problem statement

The problem involves solving two distinct tasks related to classic algorithmic challenges. First, you need to develop an efficient algorithm to solve the N-Queens problem, where you are given an integer 'n' representing the number of queens and an NxN chessboard. Your task is to place 'n' queens on the chessboard such that no two queens threaten each other, and print the board configuration for each valid solution. Second, you need to implement a solution for the subset sum problem. Given a set of integers 'S' and a target sum 'D', your goal is to find and print all possible subsets of 'S' that sum up to 'D'. Both tasks require careful consideration of computational efficiency and correctness to handle potentially large inputs and constraints effectively.

1.2 Motivation

The motivation behind solving the N-Queens problem and the subset sum problem efficiently is grounded in their broad applicability across various fields within computer science and optimization. The N-Queens problem is significant in areas such as artificial intelligence and game theory, where solving complex combinatorial problems quickly can lead to better algorithm design and strategic decision-making. Efficient algorithms for the N-Queens problem can enhance solutions in fields like robotics and puzzle-solving by reducing computation time and enabling the handling of larger problem instances.

Similarly, solving the subset sum problem efficiently is crucial for applications in cryptography, decision-making, and resource allocation. In cryptography, subset sum problems are related to security challenges and key management, where finding optimal solutions can impact the robustness of encryption methods. In resource allocation and financial modeling, efficiently solving subset sum problems can lead to better decision-making and optimization of resources. Overall, advancements in algorithms for these problems contribute to improved performance and practicality in scenarios requiring precise and timely solutions.

1.3 Objective

The objective of solving the N-Queens problem and the subset sum problem efficiently is to enhance the performance and feasibility of complex computational tasks across various domains. For the N-Queens problem, the goal is to develop algorithms that can quickly and accurately place 'n' queens on an NxN chessboard such that no two queens threaten each other, enabling efficient solutions for applications in AI, game theory, and combinatorial optimization. In the case of the subset sum problem, the objective is to identify all subsets of a given set that sum up to a specified target, which is crucial for optimizing resource allocation, cryptographic security, and decision-making processes. Efficient solutions to these problems are essential for reducing computation time, improving system performance, and handling larger datasets, thereby advancing the practical applications of algorithms in real-world scenarios.

Chapter 2

ALGORITHM TECHNIQUE USED

2.1 Introduction

The algorithms presented here are fundamental techniques in combinatorial optimization and computational problem-solving: the Sum of Subsets and the N-Queens Problem. Both algorithms are used to solve specific classes of problems involving subsets and arrangements, and they utilize different approaches to achieve efficient solutions.

Sum of Subsets is a classic problem in combinatorial optimization, where the goal is to find all subsets of a given set of integers that sum up to a specific target value. The algorithm leverages the divide-and-conquer strategy to explore potential subsets, ensuring that only those which meet the criteria are considered. This approach is essential for solving various practical problems related to resource allocation and scheduling.

N-Queens Problem is a classic problem in constraint satisfaction and combinatorial optimization. It involves placing N queens on an N×N chessboard such that no two queens threaten each other. This problem is a well-known example of backtracking algorithms and constraint satisfaction problems. The solution involves systematically exploring potential placements and backtracking when constraints are violated.

2.2 Operations

Sum of Subsets:

1. Subset Generation:

- The algorithm recursively explores the inclusion and exclusion of elements to find subsets that sum up to the desired target. It evaluates both inclusion and exclusion of each element (left and right tree evaluations) to explore all possible subsets.

2. Backtracking and Branch and Bound:

- It employs a backtracking and branch and bound technique where each branch represents including or excluding an element, and the bounds are used to cut off branches that cannot lead to a solution

(i.e., when the sum exceeds the target or when remaining elements cannot reach the target).

3.Recursive Approach:

- The recursive function explores all potential subsets and prints those that meet the sum criteria. The approach ensures that all subsets are considered while optimizing by cutting off unnecessary computations.

N-Queens Problem:

1. Backtracking:

- The algorithm uses backtracking to place queens on the chessboard. It systematically places queens in different columns and rows while ensuring that no two queens attack each other. If a conflict arises, the algorithm backtracks and tries different placements.

2. Feasibility Check:

- It uses a feasibility check function to ensure that each queen placement does not conflict with previously placed queens. This involves checking both row and diagonal constraints to ensure a valid configuration.

3. Recursive Exploration:

- The algorithm recursively places queens on the board, moving through columns and rows, and uses backtracking to correct placements when conflicts are detected.

2.3 Advantages and Limitations

Sum of Subsets:

Advantages:

☐ **Efficiency in Subset Search:**

- By using recursive and branch-and-bound techniques, the algorithm efficiently explores possible subsets and prunes unnecessary computations.

☐ **Applicability:**

- Useful in various applications such as resource allocation, scheduling, and combinatorial optimization problems.

Limitations:

☐ **Computational Complexity:**

- The recursive approach can lead to exponential time complexity in the worst case, especially for large sets of numbers.

☐ Memory Usage:

- Storing subsets and managing recursion can lead to high memory usage for large inputs.

N-Queens Problem:

Advantages:

☐ **Systematic Exploration:**

- The backtracking approach systematically explores all possible placements of queens and efficiently finds solutions.

☐ **Visual Representation:**

- Provides a clear and visual representation of the solution, useful for educational purposes.

Limitations:

☐ **Scalability Issues:**

- The algorithm can become computationally expensive for larger values of N due to the exponential growth of possible placements.

☐ **Complexity in Implementation:**

- Implementing the backtracking approach requires careful handling of constraints and recursive calls, which can be complex for larger problems.

2.4 Applications

Sum of Subsets:

- ☐ **Resource Allocation:**

- Useful in problems involving the allocation of limited resources where subsets need to meet specific criteria.

- ☐ **Financial Planning:**

- Applied in financial contexts for problems involving investment portfolios and budget management.

- ☐ **Scheduling Problems:**

- Helps in scheduling tasks or events where certain constraints and targets need to be met.

N-Queens Problem:

- ☐ **Constraint Satisfaction:**

- Serves as a fundamental problem in constraint satisfaction and is used in various optimization and scheduling tasks.

- ☐ **Game Theory and AI:**

- Useful in game theory and artificial intelligence for designing algorithms that involve arrangement and placement problems.

- ☐ **Educational Tool:**

- Commonly used as an educational tool to teach concepts of backtracking and constraint satisfaction in algorithms.

Chapter 3

SYSTEM REQUIREMENTS SPECIFICATION

1. **Operating System:**

- Windows, Linux, macOS (as long as GMP library is available and compatible)

2. **Programming Language:**

- C

3. **Front end:**

- Command Line Interface (CLI) (The code operates in a terminal or command prompt)

4. **Memory Requirements:**

- RAM: Minimum 1 MB (recommended to have at least 1 GB for practical use with large numbers)
- Storage: Minimal storage required for source code and GMP library, typically less than 10 MB

5. **Dependencies:**

- GMP Library: Ensure GMP (GNU Multiple Precision Arithmetic Library) is installed and linked correctly. This library is used for arbitrary-precision arithmetic operations.

6. **Libraries:**

- Standard C Library: Includes stdio.h for input/output operations and math.h for absolute(abs) method implementation and <time.h> for computing execution time.
- GMP Library: Includes gmp.h for multi-precision arithmetic

7. Compilation:

- Compile with a C compiler like GCC with GMP library link

8. Error Handling:

- Input Validation: Implement checks for invalid or malformed input, ensuring the program handles errors gracefully.
- Memory Management: Ensure proper allocation and deallocation of memory to prevent leaks and ensure stability.

9. Testing:

- Test Cases: Prepare a set of test cases with varying sizes of inputs to validate correctness and performance.
- Performance Benchmarks: Measure execution time for different input sizes to assess performance and efficiency.

10. Documentation:

- User Guide: Provide instructions for compiling, running, and using the program.
- Code Documentation: Include comments and documentation in the code to explain the implementation details and algorithm steps.

Chapter 4

ALGORITHM

N-queens prblem

```
Algorithm NQueens(k, n)
// Using backtracking, this procedure prints all
// possible placements of n queens on an n × n
// chessboard so that they are nonattacking.
{
    for i := 1 to n do
    {
        if Place(k, i) then
        {
            x[k] := i;
            if (k = n) then write (x[1 : n]);
            else NQueens(k + 1, n);
        }
    }
}
```

```
Algorithm Place(k, i)
// Returns true if a queen can be placed in kth row and
// ith column. Otherwise it returns false. x[ ] is a
// global array whose first (k − 1) values have been set.
// Abs(r) returns the absolute value of r.
{
    for j := 1 to k − 1 do
        if ((x[j] = i) // Two in the same column
            or (Abs(x[j] − i) = Abs(j − k)))
                // or in the same diagonal
            then return false;
    return true;
}
```

Algorithm: Sum of Subsets

```
Algorithm sumofsubset (s,k,r)
//s  is weight considered so far, k is stage, r is total remaining weight
{    x[k]=1; // Selecting the first weight
     if (s+w[k] = m) then   // terminal condition and print one soluion and continue
         write(x[1:k]);        // for remaining solution
    // select weight considered i.e Generate  left subtree of state space tree
     if (s+w[k]+w[k+1] <= m) then
           sum of sub (s+w[k], k+1, r- w[k]);

//Dead end reached, do not consider w_k , select w_k+1 i.e generate
right subtree of state space tree.
     if ((s+ r - w[k] >= m) and (s+ w[k+1] <= m)) then
     {  x{k]=0;      //Backtrack and consider k+1 weight
        sum of sub (s, k+1, r- w[k]);
     }
}
```

Chapter 5

IMPLEMENTATION

```c
#include <stdio.h>
#include <time.h>
#include <math.h> // for abs() function

// Global variables for sum of subsets problem
int i,j;
int x[10], w[10], count, d;

void sum_of_subsets(int s, int k, int rem) {
    x[k] = 1;
    if (s + w[k] == d) {
        printf("Subset %d: ", ++count);
        for (i = 0; i <= k; i++)
            if (x[i] == 1)
                printf("%d ", w[i]);
        printf("\n");
    } else if (s + w[k] + w[k + 1] <= d)
        sum_of_subsets(s + w[k], k + 1, rem - w[k]);

    if ((s + rem - w[k] >= d) && (s + w[k + 1] <= d)) {
        x[k] = 0;
        sum_of_subsets(s, k + 1, rem - w[k]);
    }
}

int place(int x[], int k) {
    for (i = 1; i < k; i++) {
        if ((x[i] == x[k]) || (abs(x[i] - x[k]) == abs(i - k)))
            return 0;
    }
    return 1; // feasible
}
```

```c
int nqueens(int n) {
    int x[10], k, count = 0;

    k = 1; // select the first queen
    x[k] = 0; // no positions allocated
    while (k != 0) { // until all queens are present
        x[k]++; // place the kth queen in next column
        while ((x[k] <= n) && (!place(x, k)))
            x[k]++; // check for the next column to place queen

        if (x[k] <= n) {
            if (k == n) { // all queens are placed
                printf("\nSolution %d\n", ++count);
                // Print the top border of the chessboard
                for (i = 0; i < n; i++) printf(" ---");
                printf("\n");

                for (i = 1; i <= n; i++) {
                    // Print the row content
                    for (j = 1; j <= n; j++) {
                        printf("| %c ", j == x[i] ? 'Q' : 'X');
                    }
                    printf("|\n");
                    // Print the bottom border of the current row
                    for (j = 0; j < n; j++) printf(" ---");
                    printf("\n");
                }
            } else {
                ++k; // select the next queen
                x[k] = 0; // start from the next column
            }
        } else {
            k--; // backtrack
        }
    }
    return count;
}
```

```c
void solve_sum_of_subsets() {
    int n, sum = 0;
    clock_t start, end;
    double cpu_time_used;

    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter the elements in increasing order: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &w[i]);
        sum += w[i];
    }
    printf("Enter the sum: ");
    scanf("%d", &d);

    start = clock();

    if ((sum < d) || (w[0] > d))
        printf("No subset possible\n");
    else
        sum_of_subsets(0, 0, sum);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken to execute: %f seconds\n", cpu_time_used);
}

void solve_nqueens() {
    int n;
    clock_t start, end;
    double cpu_time_used;

    printf("Enter the size of chessboard: ");
    scanf("%d", &n);

    start = clock();
    int solutions = nqueens(n);
    end = clock();
```

```
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

    printf("\nThe number of possibilities are %d\n", solutions);
    printf("Time taken to execute: %f seconds\n", cpu_time_used);
}

int main() {
    int choice;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Sum of Subsets Problem\n");
        printf("2. N-Queens Problem\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                solve_sum_of_subsets();
                break;
            case 2:
                solve_nqueens();
                break;
            case 3:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice, please try again.\n");
        }
    }
}
```

Chapter 6

RESULT

```
Menu:
1. Sum of Subsets Problem
2. N-Queens Problem
3. Exit
Enter your choice: 1
Enter number of elements: 6
Enter the elements in increasing order: 35 45 55 65 75 85
Enter the sum: 120
Subset 1: 35 85
Subset 2: 45 75
Subset 3: 55 65
Time taken to execute: 0.010000 seconds
```

```
Menu:
1. Sum of Subsets Problem
2. N-Queens Problem
3. Exit
Enter your choice: 2
Enter the size of chessboard: 4

Solution 1
 --- --- --- ---
| X | Q | X | X |
 --- --- --- ---
| X | X | X | Q |
 --- --- --- ---
| Q | X | X | X |
 --- --- --- ---
| X | X | Q | X |
 --- --- --- ---

Solution 2
 --- --- --- ---
| X | X | Q | X |
 --- --- --- ---
| Q | X | X | X |
 --- --- --- ---
| X | X | X | Q |
 --- --- --- ---
| X | Q | X | X |
 --- --- --- ---

The number of possibilities are 2
Time taken to execute: 0.010000 seconds
```

Chapter 7

CONCLUSION

In conclusion, efficient solutions to the N-Queens problem and the subset sum problem are crucial for advancing various fields such as artificial intelligence, cryptography, and resource optimization. The N-Queens problem benefits from efficient algorithms that reduce computational complexity, making it feasible to solve larger instances and apply these solutions to complex combinatorial and strategic scenarios. Similarly, solving the subset sum problem efficiently is vital for optimizing resource allocation and improving decision-making processes in areas like financial modeling and cryptography. Future enhancements for these problems include:

Future enhancements

- **Hybrid Approaches:**
  - **Combining Methods:** Develop hybrid algorithms that merge different techniques, such as combining backtracking with dynamic programming for the N-Queens problem or integrating exact algorithms with heuristic methods for subset sum problems to optimize performance based on problem size and constraints.

- **Optimized Recursive Implementations:**
  - **Reducing Overhead:** Improve recursive approaches by minimizing recursion depth and managing memory more efficiently, possibly through iterative implementations or advanced memory management strategies.

- **Memory Optimization:**
  - **Efficient Use:** Enhance memory management to handle larger problem sizes more effectively, potentially using memory-efficient data structures or optimized allocation strategies.

- **Parallel Processing:**
  - **Multithreading:** Implement parallel algorithms to leverage multi-core processors for faster solution times, particularly for solving large instances of the N-Queens problem or handling extensive subset sum calculations.

- **Adaptive Algorithms:**
  - **Dynamic Adjustment:** Create adaptive algorithms that adjust their strategies based on problem size and available resources, optimizing performance dynamically.

- **Integration with Modern Libraries:**
  - **Library Support:** Ensure compatibility with high-performance libraries and tools that offer optimized algorithms or hardware acceleration, enhancing overall efficiency.

- **Extended Precision:**
  - **Higher Precision Arithmetic:** Adapt algorithms to support higher precision where necessary, broadening their applicability in scientific and cryptographic contexts.

- **Improved Error Handling:**
  - **Robust Detection:** Enhance mechanisms for detecting and managing errors to ensure reliable and accurate solutions.

- **User Interface Enhancements:**
  - **Graphical Interface:** Develop intuitive graphical user interfaces (GUIs) to make these algorithms more accessible, providing interactive input, result visualization, and configuration ease.

- **Educational Tools:**
  - **Learning Resources:** Create educational tools and resources to help users understand the complexities and applications of these algorithms, including tutorials, visualizations, and interactive simulations.

REFERENCES

• **Knuth, D. E. (1998).** *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley.

• **Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009).** *Introduction to Algorithms (3rd ed.)*. MIT Press.

• **Knuth, D. E. (1973).** *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley.

• **Wikipedia:** N-Queens problem and Subset sum problem.

• **GeeksforGeeks:** N-Queens problem and Subset sum problem.

• **LaTeX Wikibook:** Algorithms for Combinatorial Problems.