Assignment No: 8 (B4)

Title: Constraint Satisfaction Problem

Problem statement: Implement crypt-arithmetic problem or n-queens or graph coloring problem (Branch-and-Bound and Backtracking).

Objective: To learn and implement constraint satisfaction problem.

Outcome: Students will be able to implement constraint satisfaction problem

S/W and H/W requirements:

- 64-bit open-source Linux OS
- Python

Theory:

Constraint satisfaction problem:

A constraint satisfaction problem (CSP) consists of

- a set of variables,
- a domain for each variable, and
- a set of constraints.

The aim is to choose a value for each variable so that the resulting possible world satisfies the constraints; we want a model of the constraints.

A finite CSP has a finite set of variables and a finite domain for each variable. Many of the methods considered in this chapter only work for finite CSPs, although some are designed for infinite, even continuous, domains.

The multidimensional aspect of these problems, where each variable can be seen as a separate dimension, makes them difficult to solve but also provides structure that can be exploited.

Given a CSP, there are a number of tasks that can be performed:

- Determine whether or not there is a model.
- Find a model.
- Find all of the models or enumerate the models.
- Count the number of models.
- Find the best model, given a measure of how good models are
- Determine whether some statement holds in all models.

Branch and Bound:

Used to find optimal solution to many optimization problems, especially in discrete and combinatorial optimization.
Systematic enumeration of all candidate solutions, discarding large subsets of fruitless candidates by using upper and lower estimated bounds of quantity being optimized.

Terminology

- Live node: is a node that has been generated but whose children have not yet been generated.
- E-node: is a live node whose children are currently being explored. In other words, an E-node is anode currently being expanded.
- Dead node: is a generated node that is not to be expanded or explored any further. All children of a dead node have already been expanded.

Branch-and-bound refers to all state space search methods in which all children of an E-node are generated before any other live node can become the E-node.

Used for state space search–
In BFS, exploration of a new node cannot begin until the node currently being explored is fully explored

General method
Both BFS and DFS generalize to branch-and-bound strategies
BFS is an FIFO search in terms of live nodes

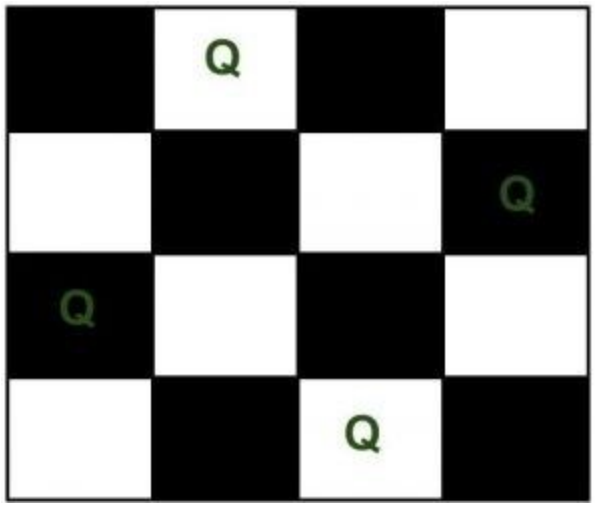
List of live nodes is a queue
DFS is LIFO search in terms of live nodes
List of live nodes is a stack

Backtracking:

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).
here are three types of problems in backtracking –
1. Decision Problem – In this, we search for a feasible solution.
2. Optimization Problem – In this, we search for the best solution.
3. Enumeration Problem – In this, we find all feasible solutions.

N-Queen Problem

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.



The expected output is a binary matrix which has 1s for the blocks where queens are placed. For example, following is the output matrix for the above 4 queen solution.

```
{
        [ 0 1 0 0 ]
        [ 0 0 0 1 ]
        [ 1 0 0 0 ]
        [ 0 0 1 0 ]
}
```

Backtracking Algorithm: The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

1) Start in the leftmost column
2) If all queens are placed return true
3) Try all rows in the current column. Do following for every tried row.
    a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.

b) If placing the queen in [row, column] leads to a solution then return True.
c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
3) If all rows have been tried and nothing worked, return false to trigger backtracking.

Testing:

Input board size	No. of solutions
1	1
2	0
8	92
10	724

Conclusion: Thus, we have successfully studied and implemented solution for n-queens problem using backtracking.