MARUDHAR KESARI JAIN COLLEGE FOR WOMEN
DEPARTMENT OF COMPUTER APPLICATIONS

SUBJECT NAME: DESIGN AND ANALYSIS OF ALGORITHMS
SUBJECT CODE: CCA53

UNIT-5 : PROBLEM SOLVING METHODS

The General Method – The 8– Queens Problem – Sum of Subsets– Graph Coloring – Hamiltonian Cycles –Branch and Bound: General Method – LC Branch and Bound – FIFO Branch and Bound.

GENERAL METHOD OF BACKTRACKING

Introduction

Backtracking is a type of technique that is based on a particular algorithm to solve a basic problem. It basically uses the recursive call function to get a particular solution by creating or building a solution stepwise with increasing values and time.

This algorithm is applied to the particular specific types of problems :

1 . Problems that require decision-making and are used to find a good solution for the problem.

2. The problems that can be optimized and are used to find the better solution that can be applied.

3. In the case of enumeration kind of problems, the algorithm is used to find the set of all easy and workable solutions for the given problem

For Backtracking problems, the algorithm finds a path sequence for the solution of the problem that keeps some checkpoints,i.e, a point from where the given problem can take a backtrack if no workable solution is found out for the problem.

Backtracking – a schematic process of trying out different types of sequences of the various decision until it reaches the conclusion.
Some terms related to backtracking are :

- Live Node: Nodes that can further generate are known live nodes.
- E Node: the nodes whose children are generated and become a successor node.
- Success Node: if the node provides a solution that is feasible.
- Dead Node: A node that cannot be further generated and does not provide a particular solution.

There are many problems that can be solved by the use of a backtracking algorithm, and it can be used over a complex set of variables or constraints, they are basically of two types :

1. Implicit constraint: a particular rule used to check how many each element is in a proper sequence is related to each other
2. Explicit constraint: the rule that restricts every element to get chosen from the particular set.
3. Some applications of backtracking include the N-Queen Problem, Sum of subset problem, Graph coloring, Hamilton cycle.
4. When to use a Backtracking algorithm?
5. When we have multiple choices, then we make the decisions from the available choices. In the following cases, we need to use the backtracking algorithm:
6. A piece of sufficient information is not available to make the best choice, so we use the backtracking strategy to try out all the possible solutions.
7. Each decision leads to a new set of choices. Then again, we backtrack to make new decisions. In this case, we need to use the backtracking strategy.
8. How does Backtracking work?
9. Backtracking is a systematic method of trying out various sequences of decisions until you find out that works. Let's understand through an example.
10. We start with a start node. First, we move to node A. Since it is not a feasible solution so we move to the next node, i.e., B. B is also not a feasible solution, and it is a dead-end so we backtrack from node B to node A.

The algorithm is as follows :

```
Backtrak(n)
        if n is not the solution
                return false
        if n is new solution
                add the list of soliution to the list

backtrack(expdn n)
```

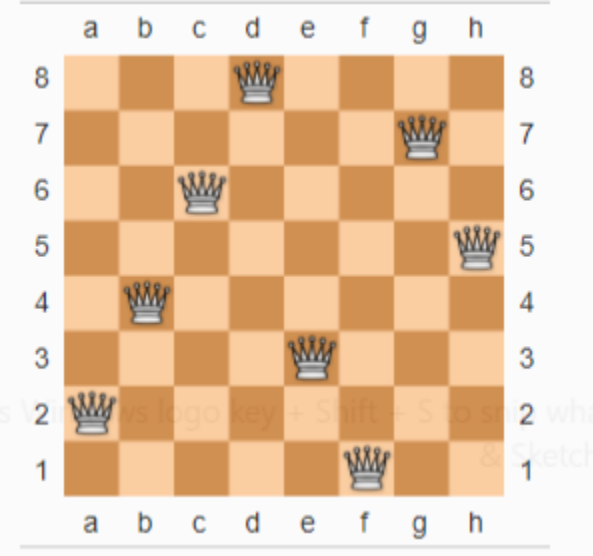
Backtracking is a vital tool for solving different problems. Though the time complexity of the algorithm is comparatively higher, it is still needed to explore different kinds of algorithms.

The backtracking algorithm is required to solve various problems such as the:

- Hamilton problem
- The knight's tour problem
- Maze solving problem
- N queen problem

N-queens problem

The n- queen problem is yet another example of problems that can be solved using backtracking. In the n-queens problem, a number of n queens are placed in a chessboard of n x n dimensions, in such a way that no queens attack each other by being in the same diagonal, row, or column. It is generally seen in an 8 x 8 matrix. The generalized version, that is the 8- queens problem was proposed by Max Bezel.



The n- queen problem must follow the following rules:

- There must be at least one queen in each column.
- There must be at least one queen in each row.
- There must be at least one queen in each diagonal.

For example, let us take a chessboard of 4 x 4 dimensions, For the 4 x 4 chessboard, we will have 4 queens, q1, q2, q3, and q4. They are to be placed in such a way that they do not each other. So we need to put each queen in a different row, let us say row "i". So, now we put the q1 in (1,1) and accordingly put q2, so they don't attack each other. The first position to successfully achieve it is (2,3). But by doing so there will be no acceptable positions left for q3, so we backtrack q2, and put it in (2,4) which

is the next best possible place on the chessboard. Then we can put q3 on (3,2), but if we do so we will not be able to find a position for q4. This time we have to backtrack from q1. We put q1 in the second-best position, which is (1,2), similarly, q2 goes to (2,4), q3 is placed in (1,2), and q4 is finally allotted to (4,3). Therefore, we get the solution : (2, 4, 1, 3). It is one of the possible solutions for the 4 – queens problem, another solution can be (3, 1, 4, 2), which is achieved by repeating the entire process for partial solutions.

Algorithm for the n- queens problem:

```
Begin
 if there is a queen at the left of the current col, then
    return false
 if there is a queen at the left upper diagonal, then
    return false
 if there is a queen at the left lower diagonal, then
    return false;
 return true //otherwise it is a valid place
End
```

Applications of the n- queen problem:

The n- queen problem also has many practical applications, such as it is used in VLSI testing, traffic control, parallel memory storage schemes, and deadlock prevention.

It can also be used for the traveling salesperson problem or TSP.

For the n- queen problems we must keep the following terminologies in mind

Distinct solution

It is found by performing rotation on the chessboard.

Moves in chess

We must be aware of the movements of the queen.

Subset problem

The subset problem is one of the problems solved using backtracking. In this problem, we need to find the subset of elements that are selected from a given set whose sum adds up to a given number K, it is assumed that the set consists of non-negative values, and there are no duplicates present.

One way of solving it is using the backtracking algorithm. We use exhaustive search to

consider all subsets irrespective of whether they satisfy the given constraints or not. A systematic consideration is made using backtracking to select the elements.

Let us understand it better by using an example.

Let us take a set of 5 integers, {4, -2, 2, 1}.

So, we need to find a subset whose sum is equal to 5.

There can be many approaches to doing so, one method is the naive approach. Here we use a search that generates all the subsets of the original array, this approach will have 2n possibilities. The time complexity will also be exponentially higher. So, we consider all these subsets in O(N) linear time and check if the sum of the elements matches or not.

Solving it using dynamic programming

Let us consider an array A, which consists of n number of non-negative elements. Find a subset X, so that the sum of all the elements equals the given number K.

A = {2, 3, 5, 7, 10}

Sum = 14.

We will start by creating a table with rows and columns i and j.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0
5	1	0	1	1	0	1	0	1	1	0	0	0	0	0	0
7	1	0	1	1	0	1	0	1	0	1	1	0	1	0	1
10	1	0	1	1	0	1	0	1	0	1	1	0	1	1	1

Graph coloring

Definition :

It is a process or procedure of assigning colors to each corner or vertex in a particular graph in such a way that no particular adjacent vertices or corners get the same color.

It's main objective is to reduce the amount of colors or number of colors while coloring in a given graph.

Vertex coloring is used efficiently and therefore is used more often. There are other graph coloring methods like Edge Coloring and Face Coloring which can be transformed into a vertex coloring method.

Edge Coloring :

Where none of the vertex is adjacent to two edge of the same color.

Face Coloring :

Or the other name is Geographical Map Coloring, used for geographical purposes.

Chromatic number of the graph :

It is the smallest number of colors that is required for coloring a graph.

How to Color a Graph :

We should follow the steps given below to color a given graph :

Firstly, arrange the given vertices of the given graph in a particular order.

Then, select the first corner and color it with the first color.

Similarly, select the next vertex and color it with the color that is lowest numbered which has not been used as a color on any of the corners and if all the vertices are colored with similar color then introduce a new color to it. Keep repeating these steps unless all the vertices are colored properly.

Applications of the Graph Coloring :

- Used in creating schedule or timetable: Every problem or important thing can be highlighted or represented using a graph where each vertex is a particular subject.

Therefore, it will be a graph coloring method where the least number of slots of time is used and is equal to a chromatic number of the given graph.

- Used in digital equipment such as Mobile Radio Frequency Assessment: When a particular type of frequency is assigned to the tower, the other tower towers must have different

frequencies in that particular location. So, in order to assign frequencies and a minimum number of frequencies can be represented using graph coloring, where each tower will represent a corner or vertex and the edge between the two towers can be represented as the range for each other.

- Most widely used in Register Allocation: While optimization process in a compiler, the register allocation is a type of process for assigning a huge number of program target variables upon the least number of CPU registers. It is also a method of graph coloring.
- Used in Bipartite Graphs: It can be used to check if a given graph is a Bipartite graph or not by using two colors to color the graph. If that given graph is two colorable then that graph is Bipartite else it's not.
- Used in Map coloring: widely used for geographical purposes for geographical maps of different countries where no same state or country can have a similar color.

Hamiltonian cycle

The Hamiltonian path is the path that visits every vertex exactly once in an undirected graph. And the Hamiltonian cycle is a Hamiltonian path that has an edge from the last vertex to the first vertex. The Hamiltonian cycle is also known as the Hamiltonian circuit. It is named after Sir William Rowan Hamilton. He devised a puzzle, known as the icosian game, in this puzzle the path along the polyhedron edges of a dodecahedron was sought.
Input for the Hamiltonian graph can either be directed or undirected graph.

In the Hamiltonian problem we check if the Hamiltonian cycle is present in the graph or not.

To generate a space tree we keep in mind the following restrictions:

- The ith vertex in the path must be next to the (i-1)th vertex in any path.
- The first vertex in the graph and the (n-1)th vertex must be adjacent to each other.
- The ith vertex cannot be the first in the (i-1)th vertex in the path.

The Hamiltonian cycle problem has many applications. It helps in time scheduling, and the choice of travel routes and network topology. It also plays an important role in other areas such as graph theory, algorithm design, and computational complexity.

The algorithm is as follows:

Input: a graph G(V, E)
Output: a graph G' without any surplus edges.
Deletion (G)
{ for each vertex v in V
For each edge e incident to v
If e is a surplus edge
Remove "e".
}

There can be two approaches to solving the Hamiltonian cycle problem:

1. The naive approach

In this we generate all the possible configurations of the vertices and print a configuration that will satisfy the requirements. So there will be n! possible configurations.

2. The backtracking algorithm

In this we create an empty path array and add a vertex to it. Then we keep adding more vertices after checking if its adjacent to the previous added vertices. We also check if its already added or not, if it is already added we add it as a part of the solution.

GENERAL METHOD OF BRANCH AND BOUND

Introduction :

Branch and bounds are basically an algorithm that is used to get the correct method of the solution

for the use of the discrete, combination, and general mathematical problems that need to be optimized. The branch and bound algorithm basically explores the amount of entire space for the search of the possible solutions and it then provides an optimal solution.

This algorithm consists of a step-wise enumerate approximate possible solutions by exploring the search space entirely. So basically all the possible solutions that build at first get a rooted decision tree.

If the problem that is given is discrete in a manner for the optimization problem then the branch and bound is a better choice. The discrete optimization is the subpart of the optimization where the variables in that particular problem should belong to that discrete set.

The algorithm branch and bound work efficiently on the combination optimization problems. It looks actually for the best solution to be provided for a particular problem in that entire space of the solution.

The bounds that are in the function that needs to be optimized are actually merged with that value of the latest best solution. It basically allows that algorithm to find the parts of the solution space totally.

Purpose :

To provide and maintain the lowest cost paths to the concerned target. Once the solution is found it just keeps improving that solution. Branch and bound algorithms are implemented in deep bounded search and depth-first search.

We can control the quality of the solution that is expected even if the solution is not yet found or introduced into that algorithm. This algorithm basically applies to those complex problems that have a finite number of solutions, in which those solutions can get represented as a particular sequence of options. The first portion of the branch and bound requires many choices that will branch out into the solution space in order to get the finite solutions.

Advantages :

In the algorithm of branch and bound it doesn’t explore the nodes in that tree. Therefore the complexity of time of the algorithm of branch and bound is less as time-consuming compared to

other algorithms.

If in such a case the given problem is not very large and if it is possible to do the branching of the nodes in a reasonable amount of time, then it finds an approximate optimal solution for that particular problem.

The algorithm of branch and bound get a minimal path in order to reach the optimal solution for that particular problem. Therefore it does not keep on repeating nodes while it explores the tree.

Disadvantages :

This branch and bound algorithm are mostly time consuming. It depends on the size of that particular problem and the number of particular nodes in that given tree can be too large in its worst case.

Secondly, the process of parallelization is also too difficult in that bound and branch algorithm.

Least cost branch and bound

Branch and bound is an algorithm to find the optimal solution to various optimization problems. It is very similar to the backtracking strategy, only just in the backtracking method state-space tree is used. The benefit of the branch and bound method is that it does not limit us to only a certain way of traversing the tree. Though it also has certain drawbacks, one being it is much slower and gets an exponential time complexity in the worst cases. The branch and bound method are applicable to many discrete combinational problems.

The branch and bound can be solved using FIFO, LIFO, and least count strategies. The selection rule used in the FIFO or LIFO linked list can sometimes be erroneous, as it does not give preference to certain nodes, and hence may not find the best response as quickly. To overcome this drawback we use the least cost method, which is also considered to be an intelligent method as it uses an intelligent ranking method. It is also referred to as the approximate cost function “C”.

The least-cost method of branch and bound selects the next node based on the Heuristic Cost Function, and it picks the one with the least count, therefore it is one of the best methods. In the 0/1 knapsack problem, we need to maximize the total value, but we cannot directly use the least count branch and bound method to solve this.

Therefore, we convert this into a minimization problem by taking the negative of the given values.

The steps to solve the 0/1 Knapsack problem is as follows:

We need to sort the items based on the value/ weight ratio.

A dummy node is inserted in the priority queue.

Then we follow the following steps until the priority queue is empty:

The peak element from the priority queue is extracted and inserted into the current node.

When the upper bound of the current node is less than the minimum lower bound then we proceed further with the next element.

We do not consider the elements with upper bounds more than the minimum lower bound as the upper bound stores the best value, and if the best value is itself not optimal than the minimum lower bound then exploring further is of no use.

- Then update the path array.
- We check the node level of the current node, if the lower node level of the current node is less than the final lower bound then we update the final path and final lower bound. If it is not less than the final lower bound then we continue with the next element.
- The lower and upper bounds of the right child of the current node are calculated.
- Then we calculate the lower and upper bounds of the left child of the current node if the current item can be inserted in a knapsack.
- We finally update the minimum lower bound and insert the children, in case the upper bound is less than the minimum lower bound.

FIFO Branch and Bound solution

FIFO Branch and Bound solution is one of the methods of branch and bound.

Branch and Bound is the state space search method where all the children E-node that is generated before the live node, becomes an E- node.
FIFO branch and bound search is the one that follows the BFS like method. It does so as the list follows first in and first out.

Some key terms to keep in mind while proceeding further:

What is a live node?

A live node is the one that has been generated but its children are not yet generated.

What is an E node?

An E node is the one that is being explored at the moment.

What is a dead node?

A dead node is one that is not being generated or explored any further. The children of the dead node have already been expanded to their full capacity.

Branch and Bound solution have three types of strategies:

- FIFO branch and bound.
- LIFO branch and bound.
- Least Cost branch and bound.

To begin with, we keep the queue empty. Then we assume a node 1.

In FIFO search the E node is assumed as node 1. After that, we need to generate children of Node 1. The children are the live nodes, and we place them in the queue accordingly. Then we delete node 2 from the queue and generate children of node 2.

Next, we delete another element from the queue and assume that as the E node. We then generate all its children.

In the same process, we delete the next element and generate their children. We keep repeating the process till the queue is covered and we find a node that satisfies the conditions of the problem. When we have found the answer node, the process terminates.

Let us understand it through an example:
We start by taking an empty queue:

In the given diagram we have assumed that node 12 is the answer node. As mentioned we start by assuming node 1 as the E node and generate all its children.

2 3 4

Then we delete node 1 from the queue, take node 2 as the E node, and generate all its children:

3 4 5 6

We delete the next element and generate children for the next node, assuming it to be the E node.

4 5 6

We repeat the same process. The generated children of 4, that is node 9 are killed by the boundary function.

5 6

Similarly, we proceed further, but the children of nodes 5, meaning the nodes 10, and 11 are also generated and killed by boundary function. The last standing node in the queue is 6, the children of node 6 are 12, and it satisfies all the conditions for the problem, therefore it is the answer node.