07/12/2021

MODULE : 3

BACKTRACKING

Backtracking alg. is applicable to the wide range of algorithm. The key point of it is a binary choice that means 'Yes' or 'No'. Whenever the backtracking has choice 'No' that means the alg. has encountered a deadend, and it backtracks one step and tries a diff path for choice YES. The Backtracking resembles a DFS tree in a di-graph, where graph is either a tree or atleast doesnot have any cycle.

The soln for the pblm according to the backtracking can be represented as implicit graph on which backtracking performs an intelligent DFS, so as to provide one or all possible soln to the given pblm. The whole task is accomplished by maintaining partial solns, as the search proceeds. It can be seen that such partial soln bind the fn where a complete soln to the pblm can be obtained. Initially no soln to the pblm is known when search proceeds a new element is added to the partial soln which in turns the remaining possibilities for a complete soln.

The search is successful, if a complete soln for the pblm is defined, in this case the search either terminate or continue to search for all other possible soln. If at any stage the search is unsuccessful that means the partial solns, constructed so far are unable to define the complete soln, then the search backtracks one step. It should be noted that, the element from the partial soln is also removed from backtracking

In many applications of the backtrack method, the desired soln is expressible as an n-tuple $(x_1, x_2, x_3, \ldots x_n)$ where the x_i are chosen from finite set S_i. Often the pblm to be solved calls for finding 1 vector that maximizes a criterion fn $P(x_1, x_2 \ldots x_n)$

Suppose m_i is the size of the set S_i, then there are $m = m_1, m_2, \ldots m_n$ that are possible candidates for satisfying the fn P. Its basic idea is to build up the soln vector one component at a time and to use modifying criterion fn $P(x_1, x_2 \ldots x_i)$, whether the vector being formed has any chance of success

Constrains can be divided into two categories:

i) Explicit Constraints
Are rules that restric each x_i to take on values only from a given set.

ii) Implicit Constraints
Are rules that determine which of the tuples in the soln phase satisfy by criterion function

9/12/2021 Thursday. N- Queens Problem.

Q_1			
x	x	Q_2	
x	x	x	x

Q_1			
			Q_2
x	Q_3	x	x
x	x	x	x

	Q_1		
x	x	x	Q_2
Q_3	x	x	x
x	x	Q_4	x

The famous companitorial N Queen's Pblm is to place N-Queens on an NxN chess board that no two queens attack each other by being in the same row, column, or diagonal.

It can be seen that, for N = 1, the pblm has a trivial soln and no solution exists for N = 2 and N = 3.

4- Queens Problem
Given a 4x4 chess board, let us no. the rows & columns of chess board from 1-4, ∴ we have to place 4 queens on a chess board s.t. no two queens attack each other. We number this as Q_1, Q_2, Q_3 & Q_4.
Each queen must be placed on a diff row, so we place queen Q_i on row i.
First we place queen Q_1 on the very first acceptable position, i.e (1,1). The first acceptable position for queen Q_2 is (2,3). But later this position proves to be dead end as no position is left for placing Q_3 safely. So we back-track one step & place queen Q_2 in (2,4), the next possible location. Each node describes its partial solution, one possible soln is shown above. For other soln, the whole method is repeated for the whole partial soln.

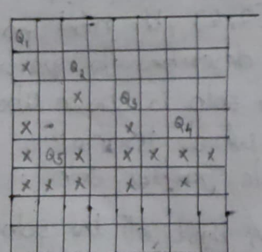
It can be seen that all the soln to the 4 queens pblm can be represented as 4-tuples (t_1, t_2, t_3, t_4) where t_i represents the column in which Queen Q_i is placed.

The explicit constrains for this are :

$S_i = \{1, 2, 3, 4\}$ where $1 \le i \le 4$

The implicit constraint is that no queen can be placed in the same row, same column or same diagonal.

8-Queens Problem



We can formulate soln to 8-Queens problem which need 8 tuples for the representation t_1 to t_8, where t_i represents the column on which queen Q_i is placed in. The soln phase consists of all 8! permutations.

Suppose two queens are placed at positions (i,j) & (k,l), then these are on the same diagonal if $i-j = k-l$ or $i+j = k+l$, ie, two queens lie on the same diagonal iff absolute value of $|j-l| = |i-k|$

A simple algorithm yielding a soln to the N-queen puzzle for n=1 or any $N \ge 4$:

Step 1:
Divide ~~N/12~~ N by 12, remember the remainder

Step 2:
Write a list of even no. from 2 to N in order

Step 3:
If the remainder is 3 or 9 move 2 to the end of the list

Step 4:
Write odd no.s 1-N in inorder, ~~work~~ if the remainder is 8 switch pairs

Step 5:
If remainder is 2, switch the place of 1 and 3 then move 5 to the end of the list.

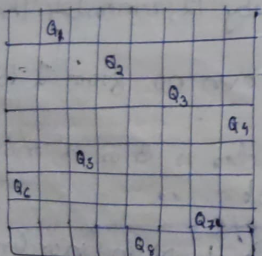
Step 6:
If remainder is 3 or 9, move 1 and 3 to the end of the list.

Step 7:
Place the first column ~~8~~ queen in the row, with the first row in the list and place the 2nd column queen with the 2nd row in the list

$12\overline{)8}$

Example
N = 8

Step 1: Remainder = 8
Step 2: 2,4,6,8,
Step 4: 1,3,5,7
(1,3) (5,7) $\xrightarrow[\text{pairs}]{\text{switch}}$ (3,1) (7,5)
(2,4,6,8,3,1,7,5)



Algorithm NQueens (k,n)

```
// Using backtracking, this procedure
   prints all
// possible placements of n queens on an
// nxn chess board so that they
// are non-attacking
{
   for i = 1 to n do
   {
       if Place (k,i), then
       {
           x(k) = i;
           if (k=n), then write (x[1:n]);
           else NQueen(k+1, n);
       }
   }
}
```

Algorithm Place (k,i)

```
/* Returns true if a queen can be placed
in kth row & 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;

Assignment:
Write algorithm & explain to find the max & min. element from a list of elements using D and C technique.

15/12/2021
Wednesday

Sum of Subsets

Suppose we are given n distinct +ve nos and we have to find all combinations of these numbers whose sums are m are called sum of subset problem

In this problem, we have to find a subset S of given set S = {$S_1, S_2, \dots S_n$} where the element of set S are n positive integers in such a manner that s' ∈ S and sum of the ~~subset~~ element of subset is equal to a +ve integer m.

If a given set n = {1, 2, 3, 4} and m = 5, then s' = {1, 4} or s' = {2, 3}

The sum of subset pblm can be solved using the backtracking approach. In this implicit trees is created which is a binary tree, the root of the tree is selected in such a way that no decision is yet taken on any input. We assume that the elements of a given set are arranged in an ↑ order.

The left child of the root node indicates that we have to include S_1 from set S and the right child of the root node indicates that, we have to exclude S_1 proceeding to next level.

Starting from the root left child indicates inclusion of S_2 and right child indicates exclusion of S_2. Each node stores the sum of the partial solution elements. If at any stage, the number = m then the search is successful and terminates

The dead end in the tree occurs only when either of the following two conditions:
i) Sum of S is too large
ii) Sum of S is too small.

We consider a backtracking using fixed tuple sized strategy. In this case the element x_i, of soln vector is either '1' or '0' depending on whether the weight w_i is included or not.

A bounding function is:

$B_k(x_1, x_2, \ldots x_k) = \text{true}$ iff

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \leq m$$

Clearly $x_1, x_2, \ldots x_k$ cannot lead to an answer node if this condition is not satisfied.

Example

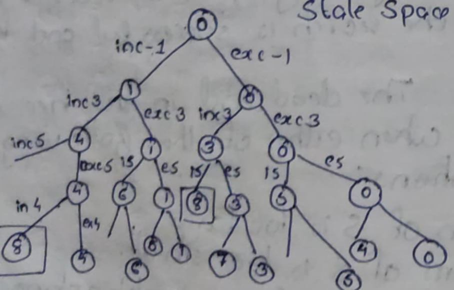
$S = \{1, 3, 5, 4\}$

$m = 8$

$S_1 = \{1, 3, 4\}$

$S_2 = \{3, 5\}$

State Space Tree



$\{1, 3, 4\}$ $(1, 2, 4)$

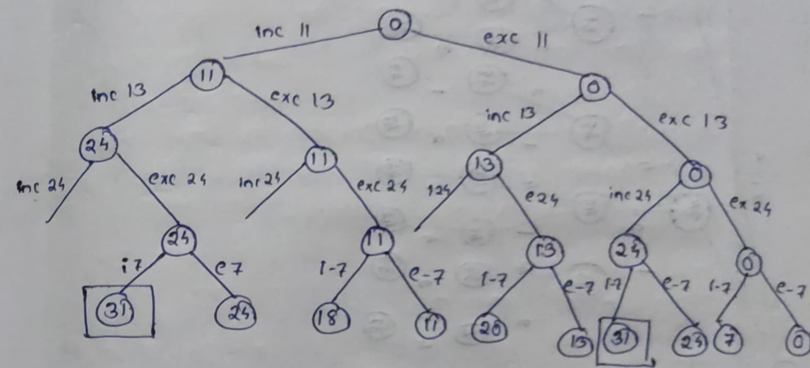
$(1, 1, 0, 1)$

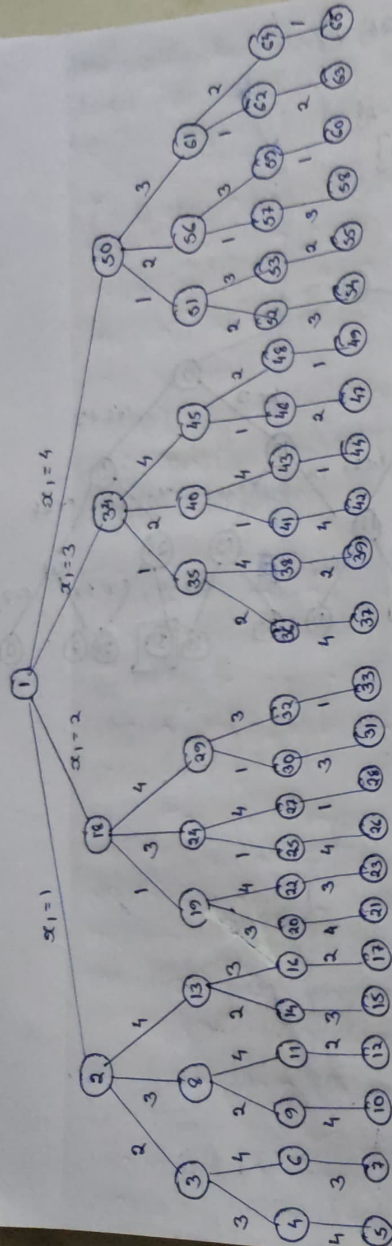
$S = \{11, 13, 24, 7\}$

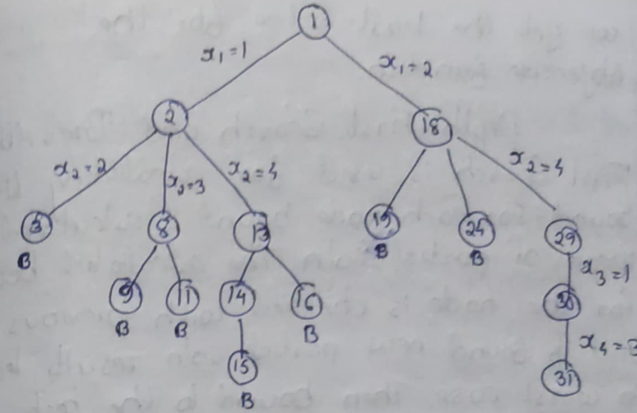
$m = 31$

$S_1 = \{11, 13, 7\}$

$S_2 = \{24, 7\}$







17/12/21
Friday

Branch and Bound.

The Branch and Bound technique like backtracking explores the implicit graph and deals with the optimal soln to a given problem. In this technique at each stage we calculate the bound for a particular node and check whether this bound will be able to give the solution or not

That means we calculate how far we are from the solution in a graph. If we find that at any node the solution so obtained is appropriate but the remaining solution is leading to a worst case, then we leave this part of the graph without exploring. It can be seen that optimal soln in a implicit graph where we get the least value for the objective function

Depth-First Search and Breadth First Search is used for calculating the bound. For each node, bound is calculated by means of partial soln. The calculated bound for the node is checked with previous node and if found new partial soln result lead to worst case. Then bound to the best soln so far selected and we leave this part without exploring further

Branch and Bound is a general algorithmic method for finding optimal soln of various optimization problem. It is basically an enumeration approach in a fashion that proves the non-promising space tree

The first one is a smart way of covering possible region by several smaller feasible subregion since the procedure may be repeated recursively to each of the sub-regions and all produced sub regions naturally form a tree structure

The term Branch and Bound refers to all State Space Search methods in which all children of the E-node are generated before any other live node can become the E-node

In Branch and Bound terminology a BFS like State Space Search will be called FIFO search as the list of live nodes is a FIFO list. A D Search like State Space Search will be called LIFO search as a list of live node is a LIFO list.

As in the case of backtracking boundy function are used to help avoiding the generation of subtree that donot contain an answer node.

Algorithm LCSearch (t)

```
// Search t for an answer node.
{
    if *t is an answer node then output
    *t and return;
    E = t; // E-node
    Initialize the list of live nodes to be empty;
    repeat
    {
        for each child x of E do
        {
            if x is an answer node then output
            the path from x to t and return;
            Add(x); // x is a new live node.
            (x → parent) := E; // Pointer for path
                                   to root.
        }
```

if there are no more live nodes then
{
write ("No answer node"); return;
}
E = Least();
}
until(false);
}

The search for an answer node can often be speeded by using an intelligent ranking ~~branching~~ function $\hat{c}(\cdot)$ for live nodes. The next E node is selected on the basis of this branching function. The ideal way is to assign branchs would be on the basis of the additional computational effort, cost needed to reach an answer node from the live node.

For any node n, the cost would be:
i) The no. of nodes in the subtree n that need to be generated before an answer node is generated
ii) The no. of levels the nearest answer node is from x.

Let $\hat{g}(x)$ be an estimate of the additional effort to needed to reach an answer node from x. The node x is assigned a rank using a fn $\hat{c}(\cdot)$ s:

$$\hat{c}(x) = f(h(x)) + \hat{g}(x)$$

where $h(x)$: Cost of reaching x from the root.

function f: Any non-decreasing fn

A search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ to select the next ~~key~~ E-node would always choose for its next E-node, a live node with least $\hat{c}$. Hence such a strategy is called an LC search

N^2-1 Puzzle

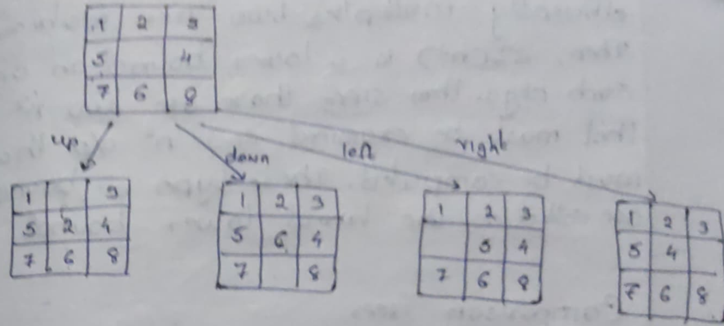
15-Puzzle consist of 15 number tiles on a square frame with a capacity of 16 tiles. Our objective is to transform the initial arrangement into the goal arrangement through a series of legal moves. The 15-puzzle has diff. sized variables. The smallest size involved a board 2x3 and is called the 5-puzzle.

The 8-puzzle involves a board 3x3

The 35 puzzle involves a board 6x6.

The family of these puzzle is called as the N-puzzle, where N stands for no of tiles. In all of the n-puzzle we use the tiles in the goal state where ordered from left-right and top-bottom with an empty space located in the bottom right corner. It is known as the family of n-puzzle belongs to the class of NP complete problems. which means that the no of paths grows exponentially with no. of tiles and finding the shortest path from the start to the goal where required performing an exaustive search

Thus from the initial arrangement 4 moves are possible. The only legal move are the one in which a tile adjacent to the empty spot is moved to ES. Each move creates a new arrangement & is called the state of the puzzle. The initial and goal arrangement are called the initial and goal states. The state space of an initial state consist of all states that can be reached from the initial state. The most straight forward way to solve the puzzle would be to search the state space for the goal space & use the path from the initial stage to the goal state as the answer



Lower Bound Theory

The concept of lower bound theory establish that the given algorithm is the most efficient possible. The way this is done is by discovering a function g(n) that is a lower bound on the time that any algorithm must take to solve the given problem. If we have an algorithm whose computing time is the same order as g(n) then we know that asymptotically we can do more better

Deriving good lower bounds is more difficult than devising efficient algorithm. However, for many problems, it is possible to easily observe that a lower bound identical to n exists, where n is the no of i/p. Suppose, we wish to find an algorithm that efficiently multiplies two $n \times n$ matrices. Then, $\Omega(n^2)$ is a lower bound on any such algorithm since there are two n^2 i/p that must be examined and n^2 o/p that must be computed. These types of bounds are called as the trivial lower bounds.

Comparison Trees

Comparison trees are useful for modeling or deriving lower bounds on problems such sorting and searching.

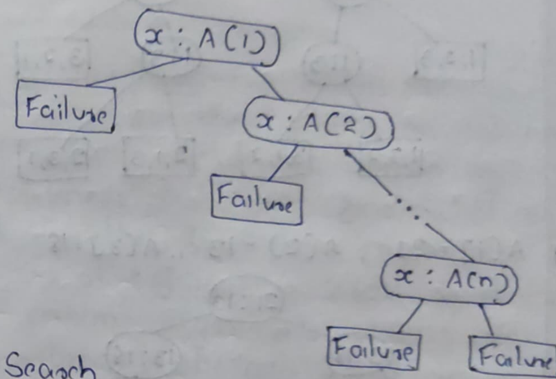
Suppose that we are given a set S of distinct values on which an ordering relation < holds. The ordered searching pblm ask whether a given element, $x \in S$ occurs within the elements $A[1:n]$. that are ordered so that $A[1] < A[2] < \ldots < A[n]$

If x is in $A[1:n]$ then we have to determine an i b/w 1 and n.

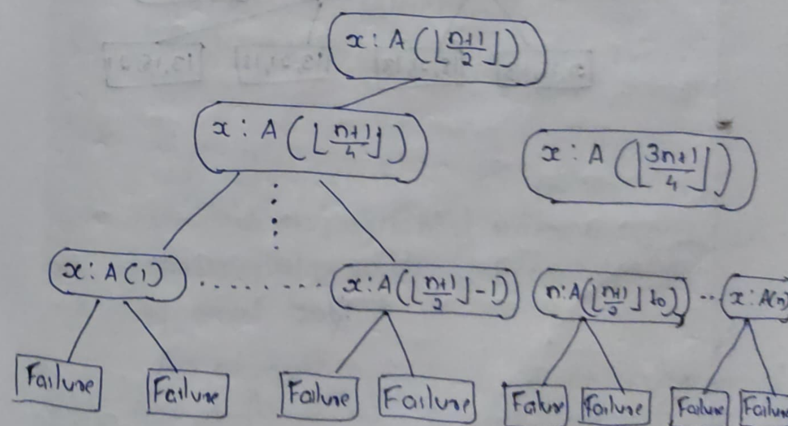
Following tree shows a comparison tree for searching: One using linear search and one using binary search.

The comparison tree for any search algorithm must contain atleast n internal nodes corresponding to n different values of i for which $x = A[i]$

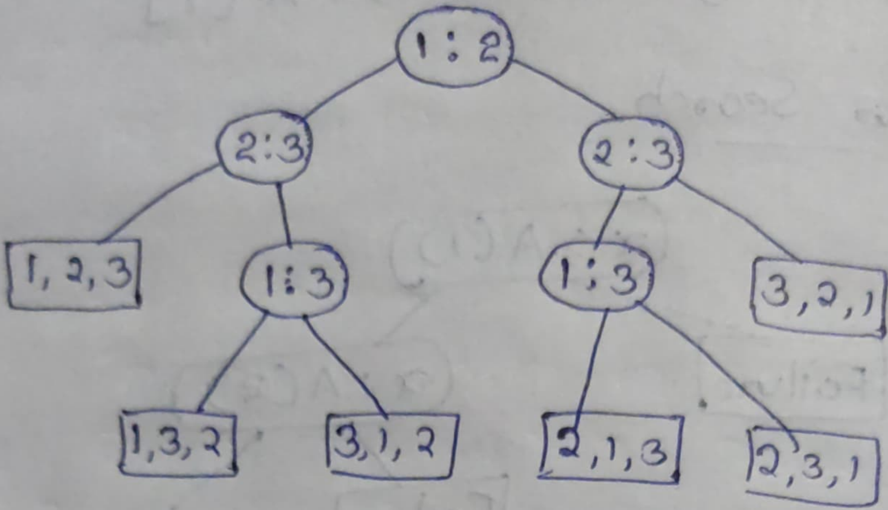
Linear Search



Binary Search



Comparison Tree for Sorting



Q: A[1] = 21 , A[2] = 13 , A[3] = 18