Assignment No: 1

Title:- Recursive and Non-recursive function

Problem statement: To calculate fibonacci numbers using recursive and non recursive function & analyse their time and space complexity

Objective:
① Create fibonacci series using recursive and non-recursive function.
② Find time and space complexity.

Theory:
The fibonacci sequence is an infinite sequence of possitive integers, starting at 0 and 1, where each succeding element is equal to the sum of its two preceeding elements
If $n^{th}$ term in fibonacci is $f_n$
then, $f_0 = 0$, $f_1 = 1$

$$F_n = F_{n-1} + F_{n-2} \quad ; \; n > 1$$

Thus, the sequence looks like
0, 1, 1, 2, 3, 5, 8, 13, .....

We can implement this using a recursive and non-recursive (iterative) approach.

Recursive Approach
Our first solution will implement recursion. This is probably the most intuitive approach since the fibonacci sequence is, by definition a recursive relation.

Let's start by defining f(n) as the function that returns the value of fn. To evaluate R(n) for n>1, we can reduce our problem into two smaller problems of the same kind R(n-1) and R(n-2). We can further reduce P(n-1) and P(n-2) to R(n-1-1) and P(n-1-2) and R(n-2-1) and R(n-2-2), respectively.

If we ~~respee~~ repeat this reductive, well eventually reach our known base cases, there by, obtain a solution a solution to R(n)
Employing this logic our algorithm for R(n) will have two steps:
1. Check if n≤1. If so, return n
2. Check if n>1. If so, call our function f with inputs n-1 and n-2 and return the sum of the two results

Pseudo code -

```
I/p → some non-negative integer n
  if n<1 then
      return n
  else
      return f(n-1)+f(n-2)
O/p → nth number in fibonacci sequence
```

Time complexity for recursive
$T(0) = T(1) = 1$
$T(n-1) \sim T(n-2)$ [approximation]

$T(n) = T(n-1) + T(n-2) + c$

$= 2T(n-1) + c$
$= 2(2T(n-2)+c) + c$
$= 4T(n-2) + 3c$
$= 8T(n-3) + 7c$
$= 2^k . T(n-k) + (2k-1)c$

for $n-k = 0 \Rightarrow n = k$
$\therefore T(n) = 2^n T(0) + (2^n - 1)c$
$= 2^n (1+c) - c$
$T(n) \sim 2^n$

Space Complexity:
Space required is proportional to maximum depth of the recursion tree which is proportional to N. Therefore space Complexity = O(N)

Iterative Approach
We will store F(0) and P(1) as 0 and 1 Next well iterate through array positions 2 to n-1. At each position i we store the sum o the two preceeding array values in f(i). Finally we return f(n-1)

Pseudocode:-
```
A[0] = 0
A[1] = 1
for i = 2, i<n do
    A[i] = A[i-1] + A[i-2]
return A[n-1]
```

Time Complexity

$T(0) = 0 + (1) = 1$

$T(n) = T(n-1) + T(0) + (T-1)$

$= O(n) + O(1) + O(1)$

Time Complexity $= O(n)$

Space Complexity

It is same for P(0), P(1) or P(100) as the values are replaced.

$\therefore O(1)$

Conclusion:-

We have successfully implemented and analysed fibonacci series using recursive and non-recursive algorithm.

Assignment No:- 2

Title:- Huffman Encoding using greedy strategy

Problem Statement:
To implement Huffman Encoding using a greedy strategy.

Objective:-
To understand and implement
① Huffman Encoding
② Greedy Strategy

Theory:-
Huffman coding is a lossless data compression algorithm the idea is to assign variable length codes to input characters. Lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallers code and the least frequent character gets the largest code.
The variable-length code assigned to input characters are "prefix-codes", means the codes (bit sequences) are assigned in such away that the code assigned to one character is not the prefix of code assigned to any other characters. This is ~~how~~ how Huffman coding makes sure that there is no ambiguity when decoding the generated bitstream.

Step involved

Huffman Encoding is an algorithm which uses frequency (or probability) feature of symbols and a binary tree structure. It consists of following 3 steps:

1. Probability (or frequency) calculation and ordering the symbols
2. Binary tree transformation
3. Assigning codes to the symbols

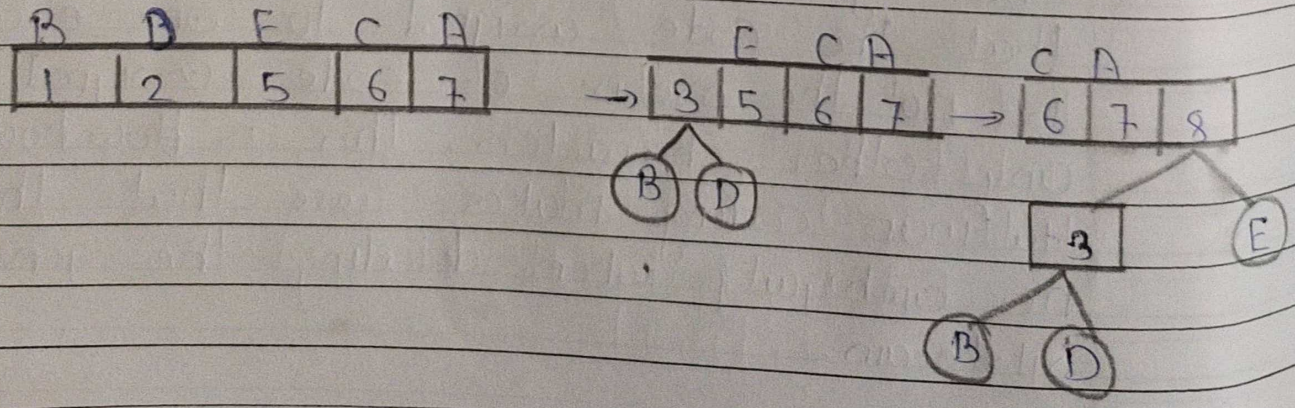
① Frequency calculation and ordering the symbols

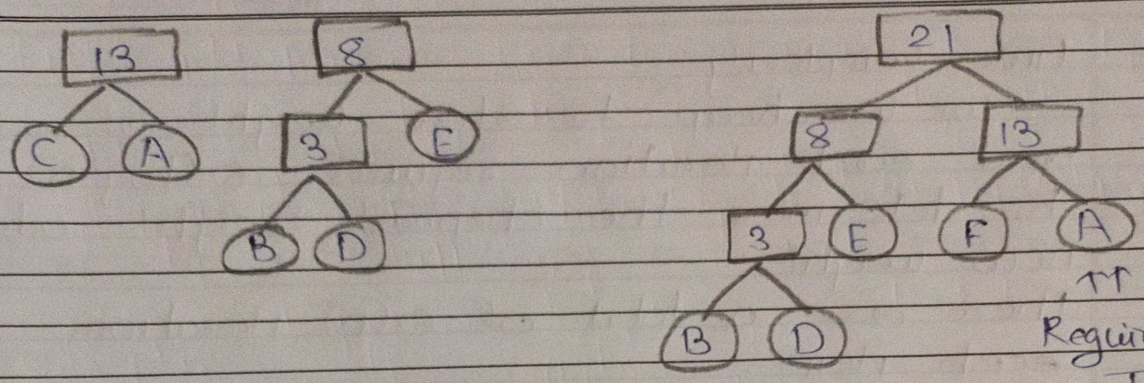
Given data → AAAAAAABCCCCCDDEEEEE

| Symbol | Freq. |
|---|---|
| A | 7 |
| B | 1 |
| C | 6 |
| D | 2 |
| E | 5 |

Sorting → B D E C A
1 2 5 6 7

② Binary Tree Transformation

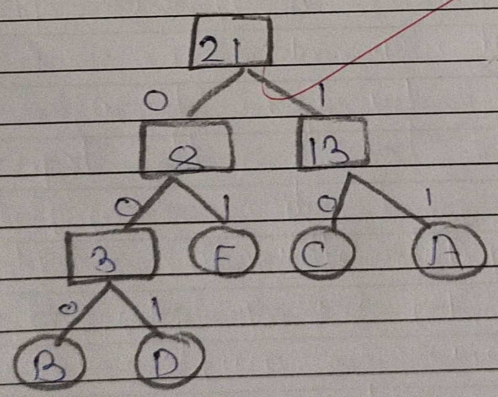





Required Huffman Tree

③ Assigning codes to Symbols

After obtaining this binary tree (Huffman Tree) the only thing left to do is assigning codes to assign code we just need to assign 1 for each time we go to a right child and a for each time we go to left child.




⟹

| Symbol | Bit Code |
|---|---|
| A | 11 |
| B | 000 |
| C | 10 |
| D | 001 |
| E | 01 |

Compression:-

We can take a quick look and see even only for 21 characters, the difference between compressed and non compressed data is non trivial

Before Compression = 21 × 8 bits = 198 bits

After compression = 7×2 bits + 1×3 bits + 6×2 bits + 2×3 bits + 5×2 bits = 45 bits

Time Complexity:

We use heap to store weight of each tree and each iteration requires O(log n) time to determine the cheapest weight add the new weight

There are a total of O(n) iterations one for each item

Thus time complexity becomes O(nlogn)

Space Complexity:-

Only one tree → O(1)

Conclusion:-

We understand the concept of Huffman Coding and Huffman tree and implemented it using python.

21/10/22

Assignment No: 3

Title: Knapsack problem using greedy Method

Problem Statement:
Write a program to solve a fractional knapsack problem using a greedy method.

Objective:
1. To understand knapsack problem
2. Implement fractional knapsack problem.

Theory:
The knapsack problem is an example of a combinational optimization problem, about finding the optimal objects among given set of objects to be placed inside a knapsack
It derives its name from: a senario where one is constrained in the number of items that can be placed inside a fixed size knapsack.
Given a set of items with specific weights and values, the aim is to get as much value into the knapsack as possible given the weight constraint of the knapsack
There are 2 methods to solve knapsack problem:
1. 0-1 knapsack.
2. fractional knapsack

Fractional knapsack
In this, we can put we can put fractions of items into the knapsack if taking the complete item is not possible.
It is solved using Greedy Method.

Steps Involved:

Lets say we have a knapsack with capacity as 100kg and we have a total of 5 items with weights $W = (20, 30, 66, 40, 60)$ and profit values $P = (10, 20, 30, 40, 50)$. Complete the maximum profit that can be achieved using fractional knapsack.

Step ①: Complete value/weight (P/W) ratio for each item

| Items | Value (P) | Weight (W) | Ratio (P/W) |
|---|---|---|---|
| $I_1$ | 10 | 20 | 0.5 |
| $I_2$ | 20 | 30 | 0.67 |
| $I_3$ | 30 | 66 | 0.45 |
| $I_4$ | 40 | 40 | 0.1 |
| $I_5$ | 50 | 60 | 0.83 |

Step 2:- Sort all items in decreasing order of their P/W ratio:-

$1 > 0.83 > 0.67 > 0.5 > 0.45$

$\therefore I_4, I_5, I_2, I_1, I_3$

Step 3:- Start puting the items into the knapsack begining from the item with the highest ratio. Put as many items as you can into the knapsack.

| Items | Values ($P_i$) | Weight ($W_i$) | Ratio | Remaining Wt | $x_i$ |
|---|---|---|---|---|---|
| $I_4$ | 10 | 40 | 1 | 100 - 40 = 60 kg | 1 |
| $I_5$ | 50 | 60 | 0.83 | 60 - 60 = 0 kg | 1 |

Maximum weight that can be placed inside knapsack using items $= \sum x_i \cdot w_i = (1 \times 40) + (1 \times 60) =$ 100 kg

Maximum profit achieved $= \sum x_i p_i = (1 \times 40) + (1 \times 50)$
$= 90$

Time Complexity
Time complexity of knapsack will be equal to addition of time required for sorting and for looping to find maximum profit and also time required to calculate P/w ratio

So it becomes $O(n \log n) + O(n) + O(n)$
Time complexity $= \max = O(n \log n)$

Space Complexity
It is equal to $O(1)$ is no more resources are used

Application of fractional knapsack
1. Portfolio Optimization
2. Home energy management
3. Power allocation management
4. Cutting stock problem
5. Resource management in software industry
6. Cutting least waste from raw materials

Conclusion:-
We understood the concept of knapsack problem and solving it using fractional knapsack

Assignment No:- 4

Title :-

0/1 knapsack problem using Dynamic programming

Problem Statement:

Write a problem to solve a 0-1 knapsack problem using dynamic programming or branch and bound strategy.

Objective:-

1. learn about 0-1 knapsack problems
2. Implement 0-1 knapsack problem using dynamic programming

Theory:-

As we know that in knapsack problem we have to find optimal number of items that can be placed inside a knapsack to provide maximum profit under the constraint of weight capacity of knapsack.

We can solve this problem using two methods

1. Fractional knapsack (Greedy Method)
2. 0/1 knapsack (Dynamic programing or Branch n bound)

We already discussed fractional knapsack problem in previous assignments using Greedy method.

0/1 knapsack

In 0/1 knapsack Problem

1. Items are indivisible

② We have to take the item either completely or leave it completely we can not take the fraction of any item.

③ It can be solved using Dynamic Programming or Branch n bound

0/1 Knapsack using Dynamic programming

For the given set of items and knapsack capacity = 5 kg, find the optimal solution for the 0/1 knapsack

$w_i = (2, 3, 4, 5)$ ; $n = 4$

$P_i = (3, 4, 5, 6)$ ; $M = 5$ kg

Step ①: Draw a table (say) 'T' with $(n+1) = 4+1 = 5$ rows and $(M+1) = 5+1 = 6$ columns. fill as the values in $0^{th}$ columns and $0^{th}$ row with 0

Step ②: For all the other values fill the values using the formula

$T(i, j) = \max[T(i-1, j), A + T(i-1, j - w_i)]$

= maximum value of the selected items if we can take items 1 to i and have weight restrictions of j.

We get the following table:

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 3 | 3 | 3 | 3 |
| 2 | 0 | 0 | 3 | 4 | 4 | 7 |
| 3 | 0 | 0 | 3 | 4 | 5 | 7 |
| 4 | 0 | 0 | 3 | 4 | 5 | 7 |

T - Table

The last entry represents the maximum possible value that can be put into the knapsack

∴ Maximum profit value = $\boxed{7}$

Step (3) To identify items that must be put into the knapsack to obtain the maximum profit

(1) Consider the last column of the table
(2) Start scanning the entries from bottom to top.
(3) On encountering an entry whose value is not
(4) Same as the value stored in the entry immediately above it, mark the row label of that entry
(4) After all the entries are scanned the marked labels present the items that must be put into the knapsack.

In our formed table.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 3 | 3 | 3 | 3 |
| 2 | 0 | 0 | 3 | 4 | 4 | 7 |
| 3 | 0 | 0 | 3 | 4 | 5 | 7 |
| 4 | 0 | 0 | 3 | 4 | 5 | 7 |

We marked row labelled as 1 and 2 following the rule. Thus items that must be put into the knapsack to obtain maximum profit 7 are: $I_1$ & $I_2$

Conclusion:-

Thus, we understood working of 0/1 knapsack problem using Dynamic programming and implement the same.

Assignment No:-5

Title:- N-queen using Backtracking

Problem Statement:-

Design N-queen matrix having first queen placed use Backtracking to place remaining queen to generate the final N-Queen matrix.

Objective:-

1. Understand backtracking
2. Implement 4-Queens problem using backtracking

Theory:-

Backtracking:- Backtracking can be defined as a general algorithm technique that considers searching every possible combination in order to solve a computational problem.

Backtracking Algorithm:-

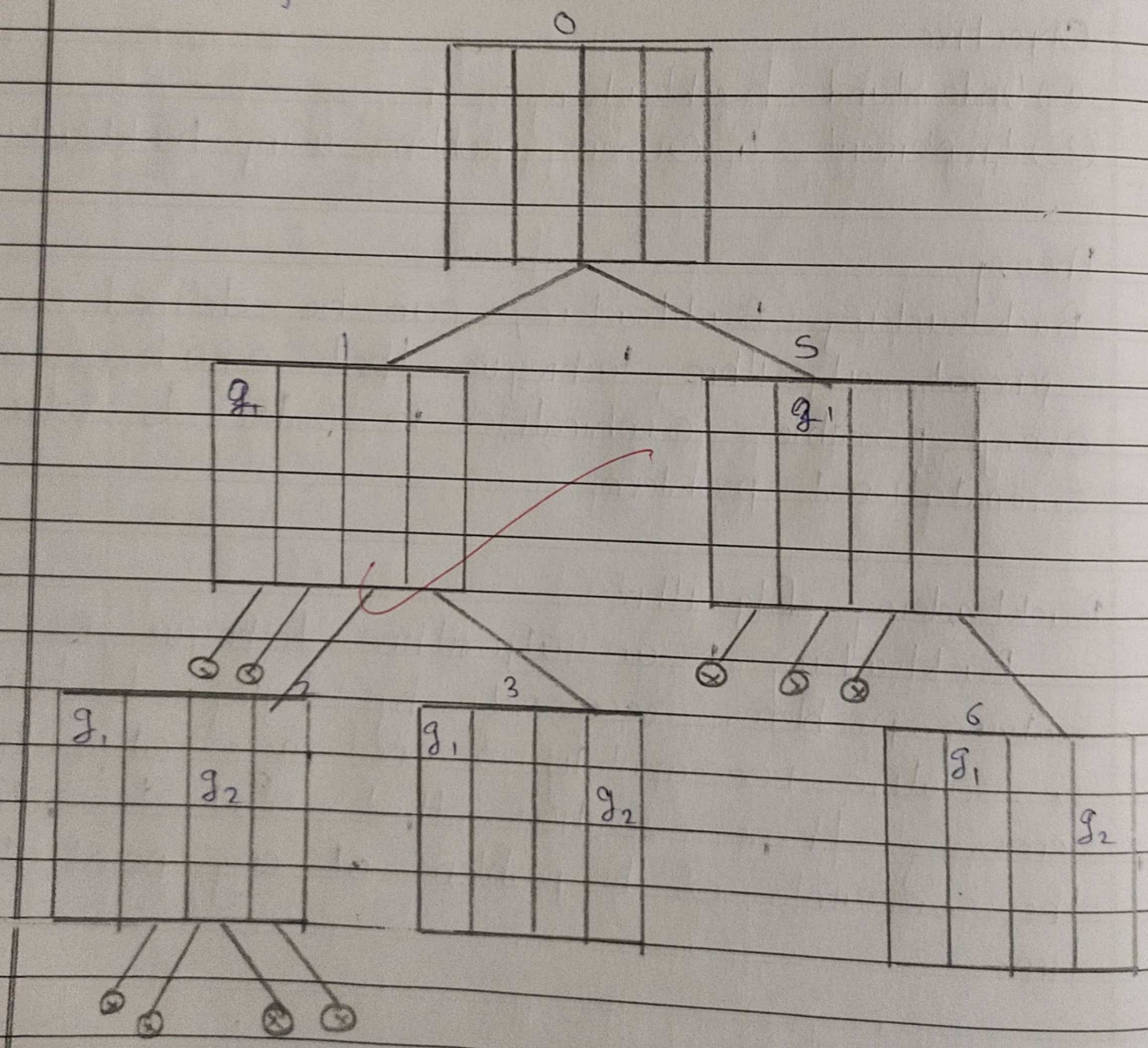
Backtracking is an algorithm technique for solving problems recursively by trying to build a solution incrementally 1 one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point at time

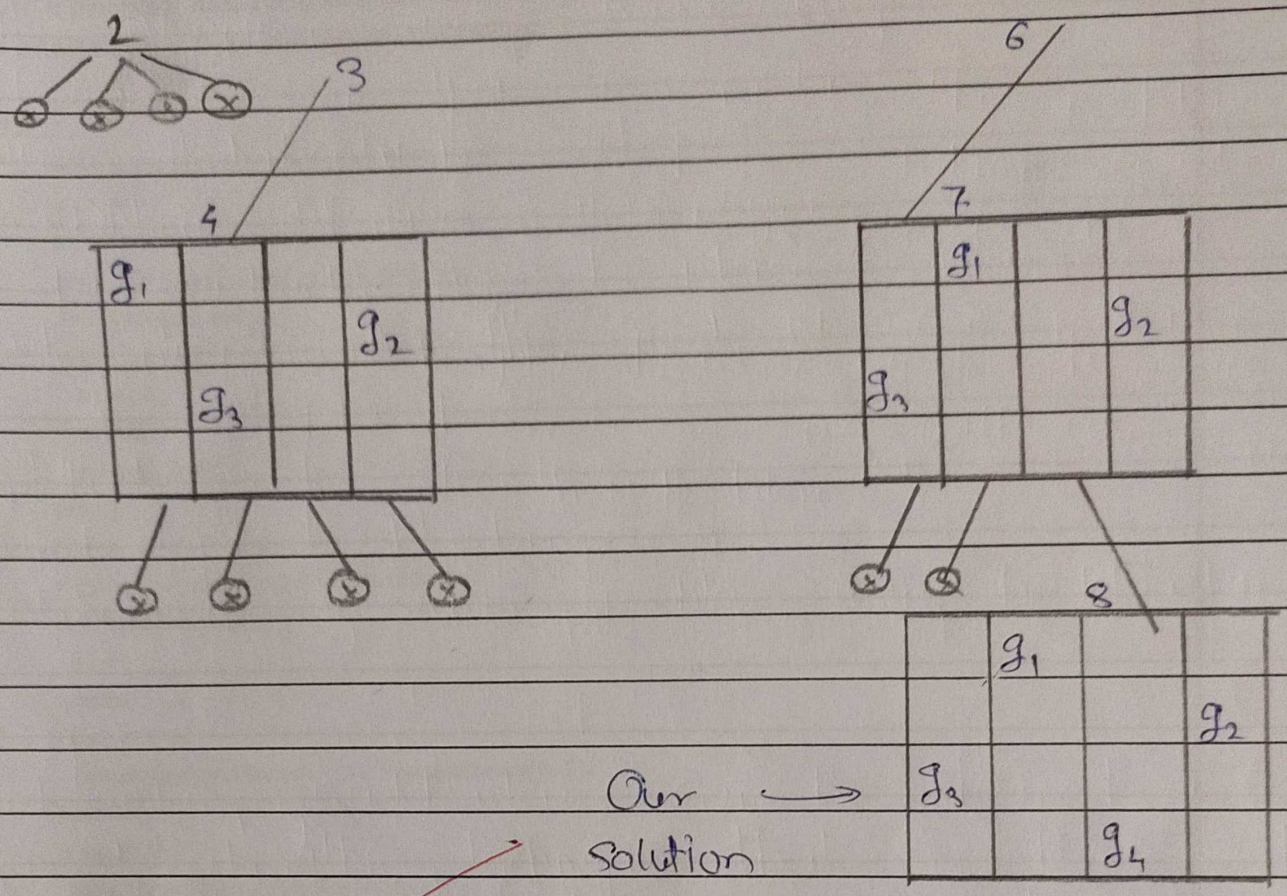
N-Queens Problem:-

N-Queen problem aims to find an arrangement of N-Queen on a N×N chessboard such that no queen attack each other by being in the same row, column or diagonal.

It can be seen that for N = 1, the problem has a trivial solution and no solution exists for N=2 and N=2 So let us consider N = 4 ie 4-Queens problem

Each queen will be placed in a single different row ie $q_i$ placed on row we will use backtracking to solve this problem.






Time Complexity:-

Time Complexity of N-Queens problem as $O(n!)$

Auxilary space:- $O(N^2)$

Conclusion:-

We understood the concept of backtracking and used it to solve N-Queens problem.

Assignment No:-1

* Title:-

→ Predict the price of the uber ride from a given pickup point to the agreed drop off location.

* Problem Statement:-

Predict the price of uber ride from a given pickup point to the agreed drop-off location perform the following tasks.

1] Pre-process the dataset
2] Identify outliers
3] Check the correlation
4] Implement linear regression & random forest regression models.
5] Evaluate the models and compare their respective scores like R2, RMSE, etc.

* Objective

1] Pre-process the dataset
2] Identify outliers
3] Check the correlation
4] Implement linear regression & random forest regression models.
5] Predict the price of uber ride from a given pickup point to the agreed drop-off location
6] Evaluate models & compare their respective scores like R2, RMSE, etc.

* Theory

→ ① Pre processing steps are:

Load data in Pandas
Drop unseful columns

Drop ~~unsuseful~~ rows with missing values
Create dummy variables
Handle missing data
Convert the data frame to Numpy
Divide dataset into training & testing data

2] Identify outliers
Visualization : a] Using Box plot
b] Using Scatter Plot

Z - Score :-

```
from scipy import stats
import numpy as np
z=np.abs(stats.zscore(df['DIS']))
print(z)
```

IQR -

Inter Quartile Range
= Quartile 3 - Quartile 1

3] Check correlation
Positive correlation
Both variables change in the same direction
Neutral Correlation
No relationship in the change of variables
Negative correlation
variables change in oppositive directions

Covariance
Cov (x,y) = (Sum (x-mean(x)) * (y-mean(y)) * 1/(n-1)
* Person's correlation
Person's correlation coefficient
= Covariance (x,y) / (std (x) * std (y))

4) Implementation

Regression helps to understand relationship betn dependent & independent variables.

Linear regression means finding the best fitting linear that explains the variability betn dependent & independent features very well.

Simple Linear Regression uses slope intercept form.

$y = b_0 + b_1 x$

To find best line model uses cost function

Mostly MSE (Mean Squared Error) or least squares method is used as cost function.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - Y^i)^2$$

$\longrightarrow$ Loss function

where $Y_i$ = Actual value

$Y^i$ = Predicted value

$n$ = number of records

$Y_i - Y^i$ = Loss function

Multiple Linear Regression

$y = b_0 + b_1 x_1 + b_2 x_2 + \dots + b_n x_n$

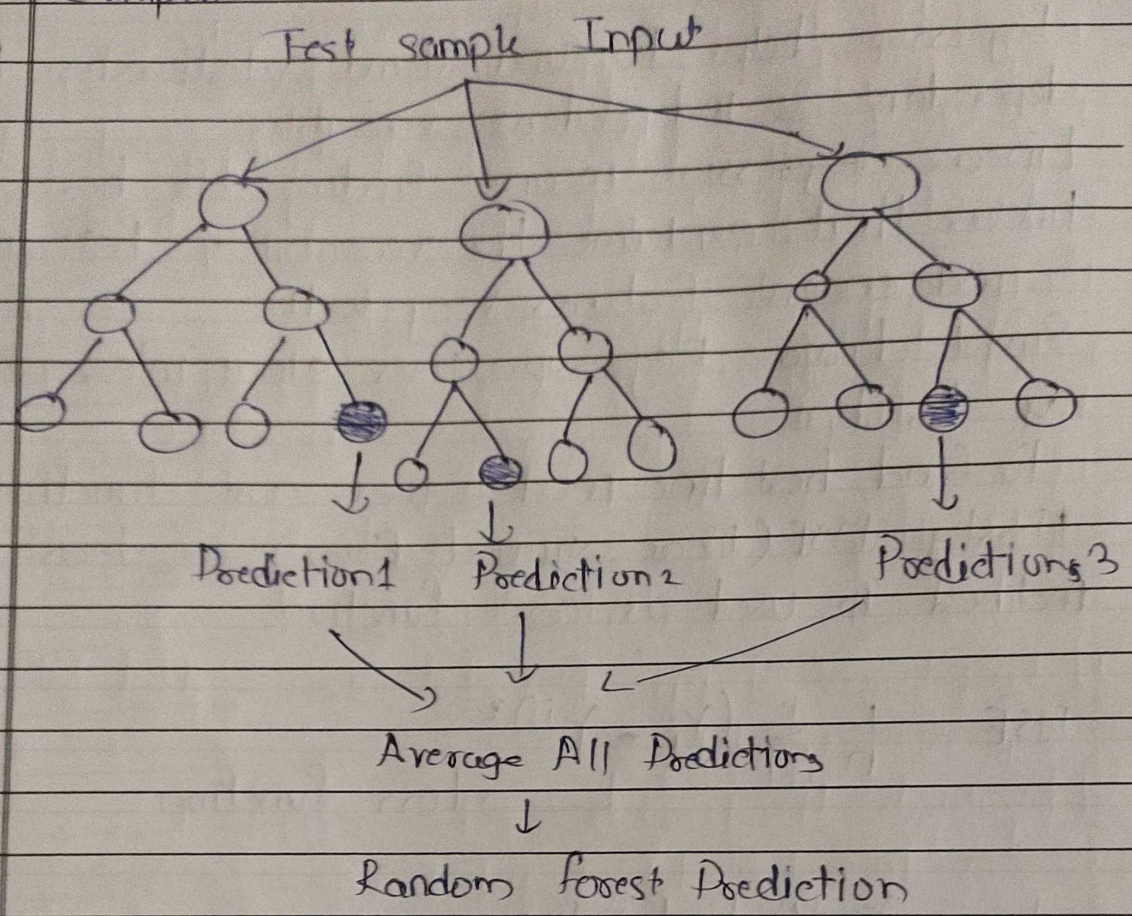
Random Forest Regression

It is supervised learning algorithm that uses ensemble learning for regression.

Ensemble learning combines predictions from multiple machine learning algorithms to make a more accurate predictions than a single model.

example.

example.




Trees trees run in parrel with no interaction amongst them. Random forest Regression documentation shows some parameter like.

1. n-estimators
2. Criterion
3. max-depth
4. max-features
5. bootstrap
6. max-samples.

Predicting result:- y-pred = regression.predict (X-test)
In regression, factors such as mean squared error, root mean squared error, $R^2$, adjusted $R^2$, etc are used to find performance

from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
$R^2$ score tells us how well our model is fitted to the data by comparing it to average line of dependent variable.

Conclusion
We studied how to predict the price of Uber ride from a given pickup point to the agreed drop-off location using linear regression & random forest regression models. Also evaluated the models and compare their respective score like R2, RMSE