TSSM
Page No. :

Assignment no-1

Title: Write a Program non recursive and recursive Program to calculate fabonacci number.

objective :- Should to able to Perform non recursive Program to calculate fabonacci number and analyse their time and Space complexity.

Prerequisite: 1) Basic of Python or Java Programming.
2) concept of recursive and non recursive form
3) Basics of time and Space Complexity.

introduction :-
introductive to fabonacci series named after italian mathamatical Pisana bogolla later known as fabonacci search.

what is fibonacci series: The fabonacci series is the sequence of number is called fabonacci search.

Be given as 0, 1, 1, 2, 3, 5, 8, 13 21, 31. it can be observed that every term can be calculate.

given the terms f_0 and second term f_1 as 0 and 1 the third term.

$$f_2 = 0 + 1 = 1$$
$$f_3 = 1 + 1 = 2$$
$$f_4 = 2 + 1 = 3$$

Fabonacci sequence formula:

$$f_0 = 0 \text{ and } f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

example:

input $n = 2$
output $= 1$
input $n = 9$
output $= 34$

The list of fabonacci number are calculated

Fn	Fabonacci number
0	0
1	1
2	1
3	2
4	3

Algorithm:

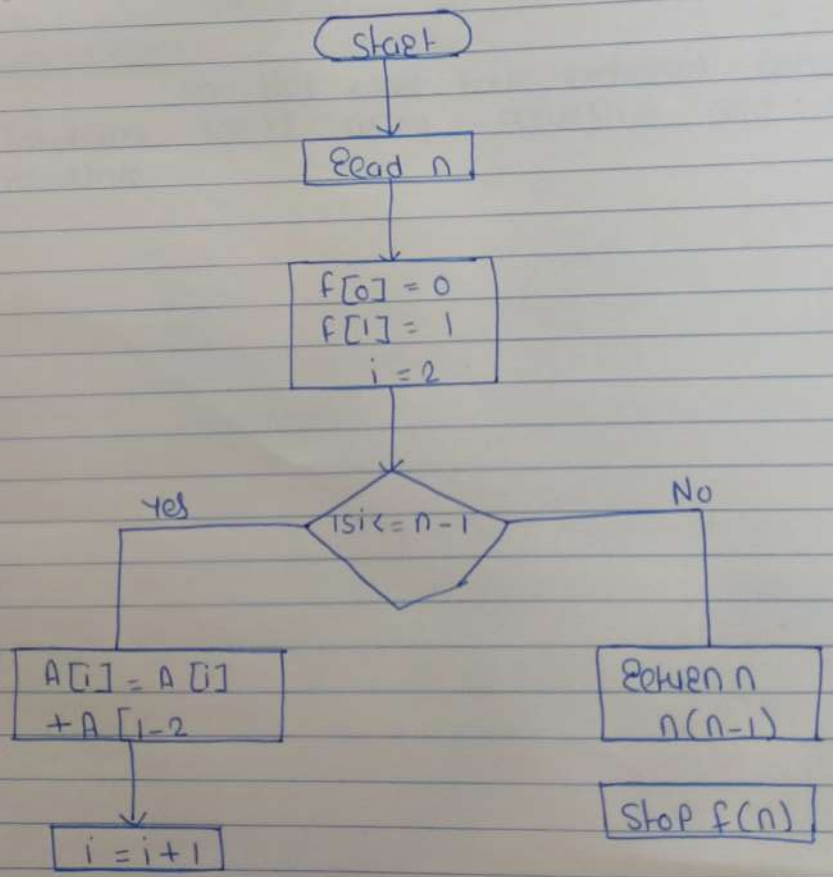
Step1: Find the smallest fabonacci number greater than or equal to n. let this number be fibm. two fabonacci no preceding it be fibmm1 and fibmm2.

Step 2: while the array has elements to be inspected.
a) compare x with the last element of the range covered by fibmm2.
b) if x maches, return index.
c) else if x matches is less than the element move the three fabonacci variables two fabonacci down indicating elimination of approximately rear two-third of the removing array.
d) else x is greater than the element move the three fabonacci variables two fabonacci down.

Step3: Since there might be a single element remaining for comparison check if fibmm2 is 1- if yes compare x with thats remaining element if match return index.

Fn	Fibonacci Numbee
5	5
6	8
7	13
8	21
9	34

Flowchart :



Application :

1) it is used to grouping of numbers and used to study differents other special mathematics sequence

2) it is used in cryptography.

Conclusion :

in this way have explored concept of fabonacci series using recursive and non recursive.

Page No. :

Assignment No-2

Title: Write a Program to implement huffman encoding using Strategy

Objective: Student should be able to Solve Huffman encoding and greedy method.

Prerequisite:
1) Basic of Python or Java
2) Concept of greedy method.

Introduction:
What is greedy method? A greedy method is an approach for solving a Program by selecting the best option available at the movement it doesn't worry whether current best result.

advantages of greedy approached:
1) this algorithm easier describe

2) This algorithm can Perform better than other algorithm.

Page No.:

Disadvantages:

1) It's doesn't produce the optimal solution.

2) for example, suppose we want to find longest path in the graph below root to leaf.

Huffman encoding :-

1) Huffman coding is a tecnic of compressing data to reduce it's size without losing of details it's was first developed devid Huffman
2) Huffman coding is famous greedy algorithm
3) it is need to length encoding
4) it assign need to assign variable length code all character.

example:

A file contains the following characters with the frequencies as shown in Huffman coding is used for data compression.

Algorithm:

Step-1:

Create a leaf node for each unique character and build min heap all leaf nodes.

Page No.:

Step 2:- Extract two nodes with the minimum frequency from the min heap.

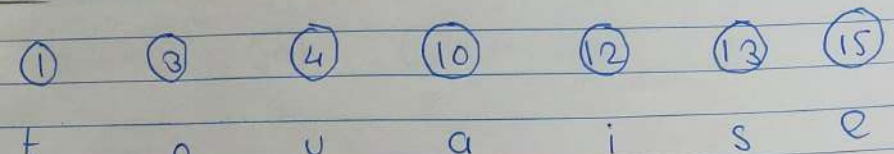
Step 3:-
Create a new internal node with a frequency equal to the sum of the two nodes frequencies make the first extracted node as it's leaf child and the other extracted node as it's right child. add this node to the min heap.

Step 4:
Repeat Step 2 and Step 3 until the heap contain any one node The remaining node is the root node and the tree is complete.

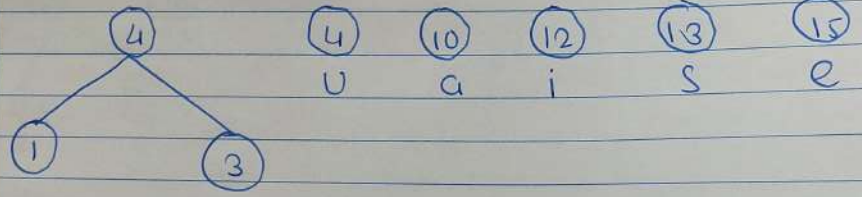
Character	Frequencies
a	10
e	15
i	12
o	3
u	4
s	13
t	1

Page No. :

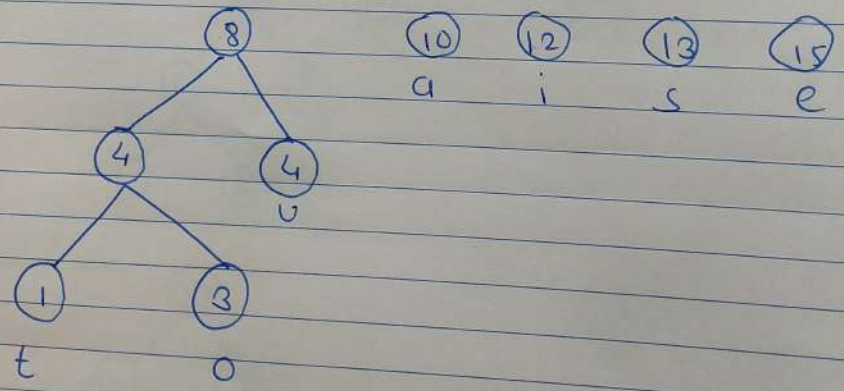
Step 1:



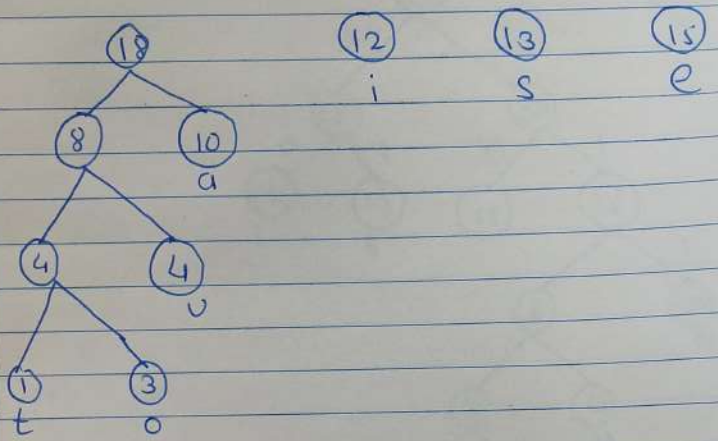
Step 2:



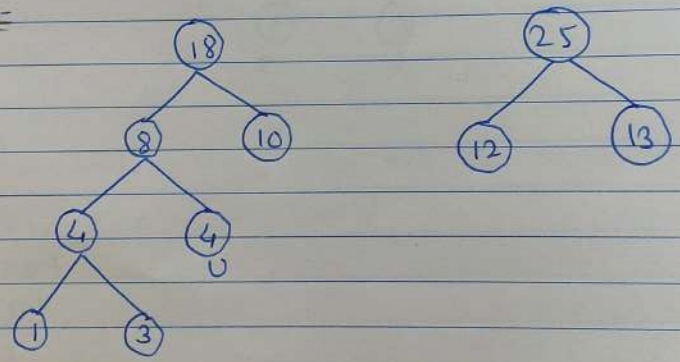
Step 3:-



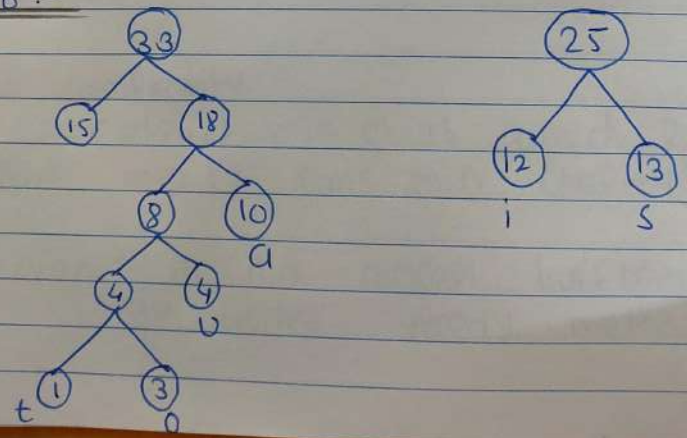
Step 4:



Step 5:-

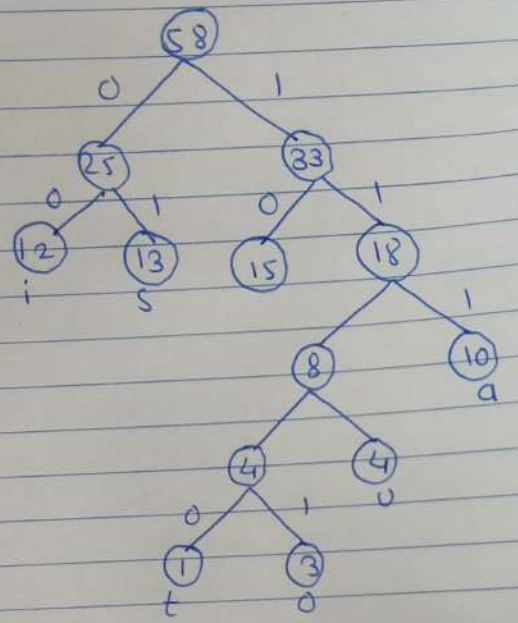


Step 6:-



Page No. :

Step 7:



a = 111
e = 10
i = 00
o = 11001
u = 1101
s = 01
t = 11000

Time complexity:

extract min () is called 2x(n-1) as extract min () calls min (heapify)

Conclusion: in this concept huffman encoding ~~is~~ using greedy method.

TSSM
Page No. :

Assignment No-3

Title: Write a Program to solve a fractional knapsack Problem using greedy method.

Objective:
Student should be able to understand and solve factoinal knapsack using greedy method.

Prerequisite :-

1) Basic of Java Programming.
2) Concept of greedy method
3) Factional knapsk Problem

introduction :
knapsack Problem

which items should be placed into knapsack such thats.
The value of Profit obtained by Putting The item in knapsack is maximum and the weight limit of the knapsack does not exceed.

knapsack Problem vaients :
1) Functional Problem
2) 0/1 knapack Problem

fractional knapsack problem:

1) as the name suggest items are divisible here.

2) we can even put the fraction of any item into the knapack if taking the complete item is not.

3) it solving using greedy method.

example:

item	weight	value
1	5	30
2	10	40
3	15	45
4	22	77
5	25	90

$n = 5$

$\omega = 60$ kg

$(\omega_1, \omega_2, \omega_3, \omega_4, \omega_5) = (5, 10, 15, 22, 25)$

$(b_1, b_2, b_3, b_4, b_5) = (30, 40, 45, 77, 90)$

Solution:

compaie the value weight ratio of each item.

Algorithm:

Step 1:
For each item, compute its value weight ratio. Sort items by the ratio in descending order.

Step 2:
Arrange all the items in decreasing order of their value / weight ratio Start with the highest ratio item Put items into the bag until The next item on the list cannot fit

Step 3:
Try to fill any remaining capacity with The next item on the list that can fit.

Put as many item as you can into the knapsack.

Example:

items	weight	value	ratio
1	5	30	6
2	10	40	4
3	15	45	3
4	22	77	3.5
5	25	90	3.6

Page No. :

Step-2 :
Sort all the item decreasing order of value.

1	2	5	4	3
(6)	(4)	(3.6)	(3.5)	(3)

Step 3:
Start the Putting item one by one

knapack weight	items	cost
60	0	0
55	11	30
45	11, 12	70
20	11, 1215	160

Total cost of knapsack

$= 160 + (20/22) \times 77$

$= 160 + 70$

$= 230$ units.

Time complexity :
1) The main time taking steps is the sorting at all items in decreasing order of the value ratio.
2) The whole loop take O(n) time
3) The average time complexity of quick sort O(n logn)

4) Therefore total time taking including is the sort $O(n \log n)$

Conclusion :

in This way we have explored concept of fractional knapsack using greedy method.

TSSM
Page No.

Assignment No-4

Title: Write a program to solve 0-1 knapack problem using dynamic programming or bound strategy

Objective :-

Students should be able to understand and solve 0-1 knapack problem using dynamic programming.

Prerequisite :-

1) Basic Python Programming
2) Concept of dynamic Programming
3) 0/1 knapsack problem

Introduction :

What is dynamic Programming? dynamic Programming is also used in optimization problems like divide and conquer method.

Dynamic Programming solves problem by combining the solution of subproblems.

Dynamic Programming algorithm solves each sub-problem just once and then save its answer in table therefore avoiding the work of re-computing the answer every time.

Two main properties of problem suggest the given problem solved using dynamic programming.

The properties are overlapping sub-problem and optional substructure.
Dynamic programming also combines solution in sub problem.
The computed solution are stored in a table so the don't have to be recompound.

Application:
1) matrix chain multiplication
2) longest common subsequence
3) travelling salsman problem

Knapsack Problem:

1) A knapsack with limited weight capacity
2) Fell item each having some weight having and value.

Example:
For the given set of item and knapack capacity = 5 kg find the optimal solution for the 0/1 knapsack problem making dynamic approach.

TSSM
Page No. :

Algorithm:

Step-1:
Draw a table say 'T' with (n+1) number of rows and (w+1) number of columns
Fill all the boxes of 0^{th} row and 0^{th} column with zeroes as shown.

Step 2:
Start filling the table row wise top to bottom from left to right.
Use the following formula.
T(i,j) = max {T(i-1,j), value i + T(i-1, j - weight i)}

Step 3:-
To identify the items that must be Put into the knapsack to obtain that maximum Profit.
- Consider the last column of The Table
- Start Scanning the entries from bottom to Top.
- on encountering an entry whose value is not same as value stored in entry immediately
- after all entries are Scanned the marked labels represent The items that must be Put into The knapsack.

TSSM
Page No. :

item	weight	value
1	2	3
2	3	4
3	4	5
4	5	6

$n = 4$
$w = 5$ kg
$(w1, w2, w3, w4) = (2, 3, 4, 5)$
$(b1, b2, b3, b4) = (3, 4, 5, 6)$

Answer :

given data

knapsack capacity $(w) = 5$ kg
number of item $(n) = 4$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					
5	0					

T - table

step 2.

Start filling to table row wise top to bottom using following formula.

$T(i,j) = \max\{T(i-1,j), value + T(i-1, j-weight)\}$

Finding $T(1-1)$
we have
$i = 1$
$j = 1$
$(value) = (value) = 3$
$(weight)_1 = weight = 2$

Substracting the value we get

$T(1,1) = \max\{T(1-1,1), 3 + T(1-1), 1-2)\}$
$T(1,1) = \max\{T(0,1), 3 + T(0,-1)\}$
$T(1,1) = T(0,1)$ {ignore $T(0,-1)$}
$T(1,1) = 0$

Finding $T(1,2)$
we have
$i = 1$
$j = 2$
$(value) = (value) = 3$
$(weight) = (weight) = 2$

Substracting the value we get
$T(1,2) = \max\{T(1-1), 3 + T(1,1) 2-2)$
$T(1,2) = \max\{T0,2) 3 + T(0,0)$
$T(1,2) = \max\{0, 3+0\}$
$T(1,2) = 3$

Finding T (1,3)
we have
i = 1
j = 4
(value)$_i$ = (value)$_i$ = 3
(weight)$_i$ = (weight)$_i$ = 2

Substituting The values we get

T(1,4) = max {T(1-1)4), 3+T (1-1), 4-2)}
T(1,4) = max {T (0,4), 3+ T (0,2)}
T (1,4) = max {0, 3+0}
T (1,4) = 3

Time complexity :

i) it takes $\phi(n)$ time for tracing the solution process traces n rows

2) it takes $\phi(nw)$ time to fill $(n+1)(w+1)$ table entries.

3) Thus overall $\phi(nw)$ time taken 0/1 using dynamic Programming

Conclusion :

in This way we explored concept of 0/1 knapsack using dynamic approach.

TSSM
Page No. :

Assignment no-5

Tittle:

design n-queens matrix having first queen placed use backtracking to place remaining queen to generate the final n-queens matrix.

objective:

Student should be able to understand and solve n-queen problem and understand basics of backtracking.

introduction / Theory:

introduction to backtracking:

many problems are difficult to solve algorithmically backtracking makes it possible to solve at least some large instance of difficult problem. suppose we have to make a series of decion among variables choice

Each decision leds to a new set of choices.

Some sequence of choice is more than one choices may be sdution of problem.

what is backtracking:

backtracking is the finding the solution of a problem where by the solution depends on previous step taken.

For example maze problem The solution depend on all the steps you take one by one if any of Those step is wrong Then it will be abled lead the solution. in the maze problem we first choose path and continue moving along it. But once we understand the particular Path is incorrect Then we just come back and change it This is backtracking basically its in the backtracking we first take and the we see taken is correct or not i.e whether it will give a correct answer or not.

* Thus The general steps:

1) Start with The Sub-solution
2) if not Then come back & change The solution and continue again.

application of backtracking:
1) N-queens problem
2) Sum of subset problem
3) graph colouring
4) Hamilton cycles

what is N-queens problem: N-queen is the classical example of back tracking

Algorithm:

Step-1:

Start in the left most column

Step 2:

if all queens are placed return true.

Step-3:

Try all rows in the current column no The following for every tried row.
- 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
- if Placing queen in leads to a solution Then return true.
- if Placing queen doesn't lead to a solution unmark this [row, column] (backtrack) & go to step (a) to try other rows.

Step 4: if all rows have been tried and nothing worked, return false to trigger backtracking.

N queen Problem is defined as given N x N chess board.

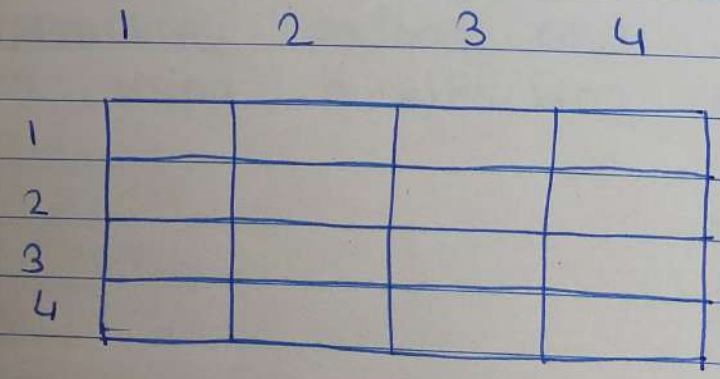
for N=1 this is trival cas for N=2 and N=3 a Solution is not Possible so start with N=4 and will generalize N queen.

* Algorithm :-

1) Start the leftmost column
2) if all queen are placed return true
3) Try all rows in current column
4) if all rows have been tried and nothing worked, return false to trigger backtracking.

Example :

given 4x4 chessboard arrange four queen in a way such that no two queen attack each other.



we have arrage the queen q1, q2, q3 and q4 in chess board we will put with queens in row 1st The start with position (1,1) q1

only queen so there no issue.

All Possible solution for four queen are show in diagram 1 and 2

	1	2	3	4
1		Q1		
2				Q2
3	Q3			
4			Q4	

	1	2	3	4
1			Q1	
2	Q2			
3				Q3
4		Q4		

Dig [1]

one function will take row number of queen size the board and board itself N-queen (row, n, N)

Conclusion :-

in this we have explored concept of backtracking method and solve N-queen problem using backtracking method.