Day 2
20/02/2024

Recursion and Backtracking

Recursion:

* Recursion is a Programming technique where a function calls itself in order to solve a problem.

* Think of it as a way to breakdown a big Problem into Smaller, more manageble pieces, where each piece is Solved in the Same way as the big problem.

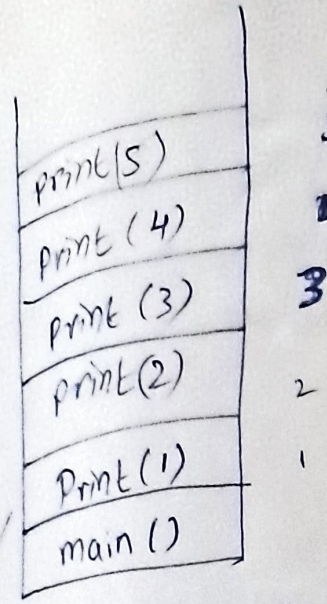
~~Why recursion~~:

```
Public static void print (int n) {

    if (n==5) {
    System.out.println(5);
    return;
    }
  System.out.println (n);
  // recursive call
  // if you are calling a function againand again, you can
     Treat it as a seperate call in the stack
  print (n+1);

Public Static void main (String [] args) {
    // print first 5 numbers : 12345
  print (1);
}
```

it will continuosly prints if we dont give base condition

** Base condition:

in recursion it is a condition where our recursion will stop making new calls



every call of function will take some memory

* No base condition: if there is no base condition function calls will keep happening, stack will be filled again and again

* memory of computer will exceed the limit which will cause stack overflow error.

Why we need Recursion:

* it helps us in solving bigger / complex problems in a simpler way.

* you can convert the recursion solutions into iteration and vice versa

* Space complexity is not constant because of recursive calls.

* it helps us in breaking down bigger problems into smaller problems.

Visualizing Recursion: ***

program start

main ()

↓

Print (1)

↓

Print (2)

↓

Print (3)

↓

Print (4)

↓

Print (5)

program over

Recursion Tree

Question: find n^{th} fibonacci number

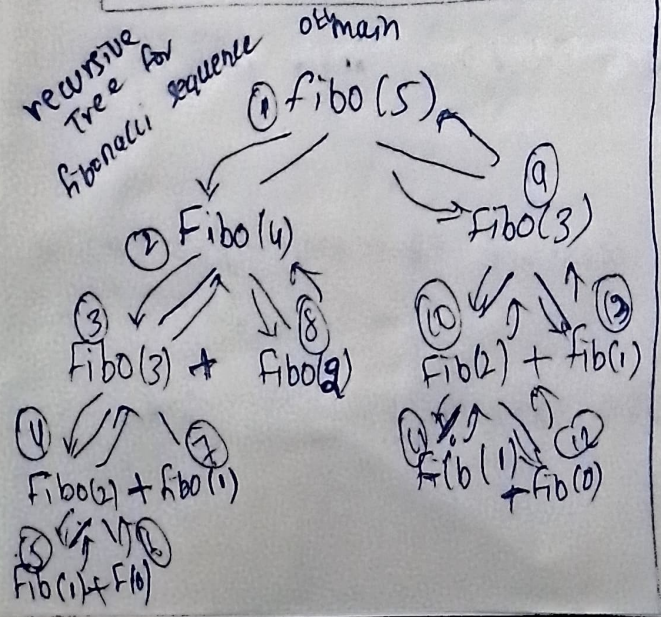
0th 1st 2nd 3rd 4th 5th 6th 7th

0, 1, 1, 2, 3, 5, 8, 13 .....

This is known as recurrence relation

$$\boxed{fibo(N) = fibo(N-1) + fibo(N-2)}$$

recursive tree for fibonacci sequence



* Break it down into smaller problems
* The base condition is represented by answers we already have

in this case, we know that $f(0)=0$
$f(1)=1$

This is base condition

Note: How can you ~~identify~~ understand and approach a Problem.

* identify if you can break down your problem into smaller Problems
* write the recurrence Relation if needed
* draw the recursive tree
* about the tree
  - → see the flow of functions, how they are getting in Stack
  - → identify the left tree calls and Right tree calls
  - → Draw the tree and pointers again and again using pen and paper
  - → use a debugger to see the flow
* See How the values are returned at each step and
* See where the function call will come out of, in the end you will come out of the main function

Variables : Arguments
Return type
Body of function

Day 3
21-02-2024

Back Tracking:

The Back Tracking is a systematic method for solving problems by incrementally building candidates to the solutions and abandoning a candidate ("backtracking") as soon as it is determined that this candidate cannot possibly lead to a final solution.

How Back Tracking works!

* Back tracking is often applied to solve constraint satisfaction problems such as puzzeles, combinatorial problems and parsing.
* The process involves exploring the possible steps in a space of configurations and recursively proceeding forward to build a solution.
* if current path does not lead to a solution, the algorithm backtracks to explore the next possible path.

Key Concepts!

1. Decision Tree: Backtracking algorithms visulize the problem as a tree of choices, where each node represents a partial solution

2. States: configurations of the problem at any point can be partial or complete, invalid states don't satisfy constraints or lead to solutions

3. Base case: condition indicating a solution is found, ending recursion. valid final states are recorded or returned

4. Choices: Algorithm explores all choices from current state, extending solutions and diving deeper into the tree

5. Constraints: conditions solutions must meet, used to prune the decision tree and eliminate non-viable paths.

6. Goal: the problem's objective, often to find one or all valid solutions.

Process:

The General Process of Backtracking involves.

1. Choose: Select a path ~~notes~~ or option to explore.
2. Constraint: check if the current path violates any constraints. if it does, backtrack.
3. Goal check: determine if the current path satisfies the goal conditions. if so, record or return the solution
4. Recursion: Recursively apply these steps to extend the path and explore further options.
5. Backtrack: if a path is fully explored or found to be invalid backtrack to the previous decision point and try different path.

* Backtracking is effective for complex problem-solving but computationally intensive, exploring many or All paths.

Optimizations like pruning are essential for efficiency it explores solution spaces, tests and disregards unfit partial solutions until finding a satisfactory solution or exhausting options.