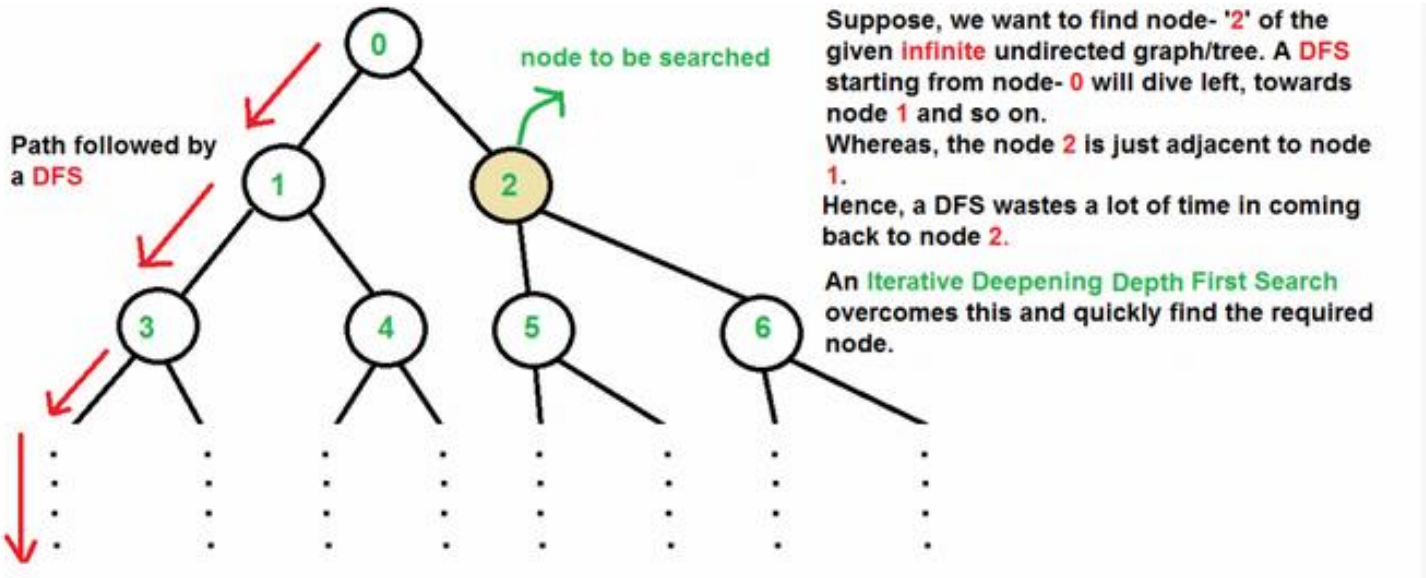
IDS AND PROBLEM CHARACTERISTICS

Dr. Krishnendu Guha
Assistant Professor (On Contract/ Tenured)
National Institute of Technology (NIT), Jamshedpur
Email: krishnendu.ca@nitjsr.ac.in

PROBLEM OF DFS

- **DFS** first traverses nodes going through one adjacent of root, then next adjacent
- The problem with this approach is, if there is a node close to root, but not in first few subtrees explored by DFS, then DFS reaches that node very late
- Also, DFS may not find shortest path to a node (in terms of number of edges)



ITERATIVE DEEPING SEARCH

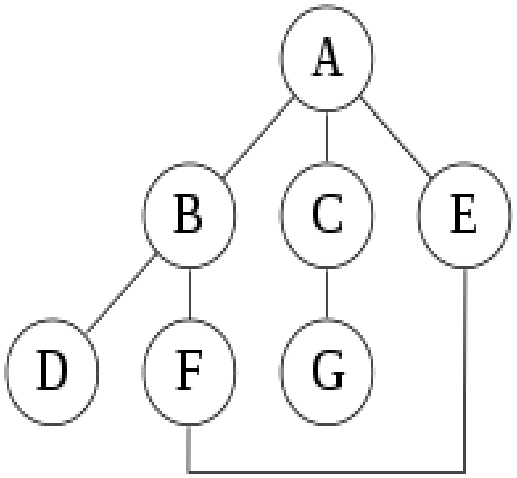
- **IDS** combines depth-first search's space-efficiency and breadth-first search's fast search (for nodes closer to root)

- Also known as IDDS (Iterative Deeping Depth First Search)

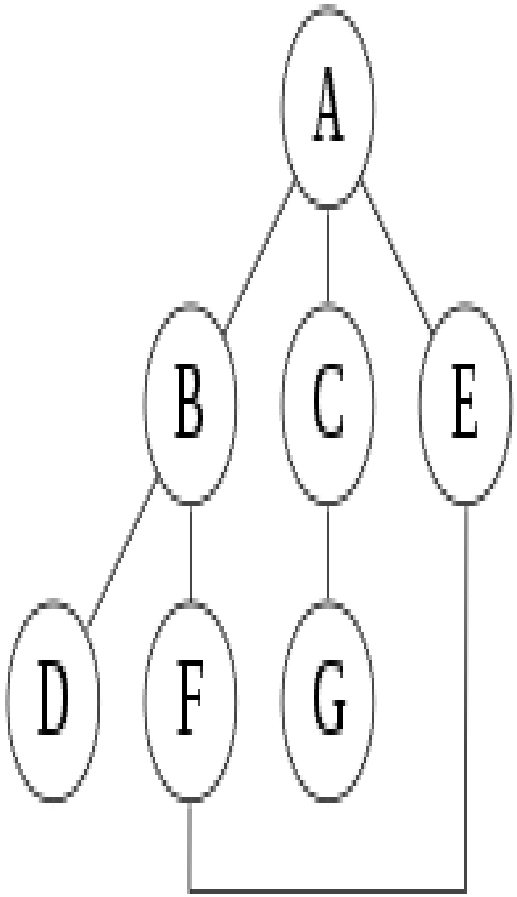
- **How does IDDFS work?**
- IDDFS calls DFS for different depths starting from an initial value
- In every call, DFS is restricted from going beyond given depth. So basically we do DFS in a BFS fashion.

- **PROBLEM OF DFS**
- DFS results in visiting nodes in the order A, B, D, F, E, A, B, D, F, E, etc. forever

- Gets caught in the A, B, D, F, E cycle and never reaching C or G



- IDS prevents this loop and will reach the following nodes on the following depths, assuming it proceeds left-to-right as above:
- **Depth 0: A**
- **Depth 1: A, B, C, E**
- (Iterative deepening has now seen C, when a conventional depth-first search did not.)
- **Depth 2: A, B, D, F, C, G, E, F**
- (It still sees C, but that it came later. Also it sees E via a different path, and loops back to F twice.)
- **Depth 3: A, B, D, F, E, C, G, E, F, B**
- For this graph, as more depth is added, the two cycles "ABFE" and "AEFB" will simply get longer before the algorithm gives up and tries another branch



- An important thing to note is, we visit top level nodes multiple times
- The last (or max depth) level is visited once, second last level is visited twice, and so on
- It may seem expensive, but it turns out to be not so costly, since in a tree most of the nodes are in the bottom level
- So it does not matter much if the upper levels are visited multiple times.

TIME COMPLEXITY

- **Time Complexity: $O(b^d)$**
  - Suppose we have a tree having branching factor 'b' (number of children of each node), and its depth 'd', i.e., there are **b^d** nodes.
- In an iterative deepening search, the nodes on the bottom level are expanded once, those on the next to bottom level are expanded twice, and so on, up to the root of the search tree, which is expanded d+1 times. So the total number of expansions in an iterative deepening search is-
  - $(d)b + (d-1)b^2 + ... + 3b^{d-2} + 2b^{d-1} + b^d$
  - That is, Summation$[(d + 1 - i)\ b^i]$, from i = 0 to i = d
  - Which is same as $O(b^d)$
  - After evaluating the above expression, we find that asymptotically IDDFS takes the same time as that of DFS and BFS, but it is indeed slower than both of them as it has a higher constant factor in its time complexity expression
- IDDFS is best suited for a complete infinite tree

SPACE COMPLEXITY

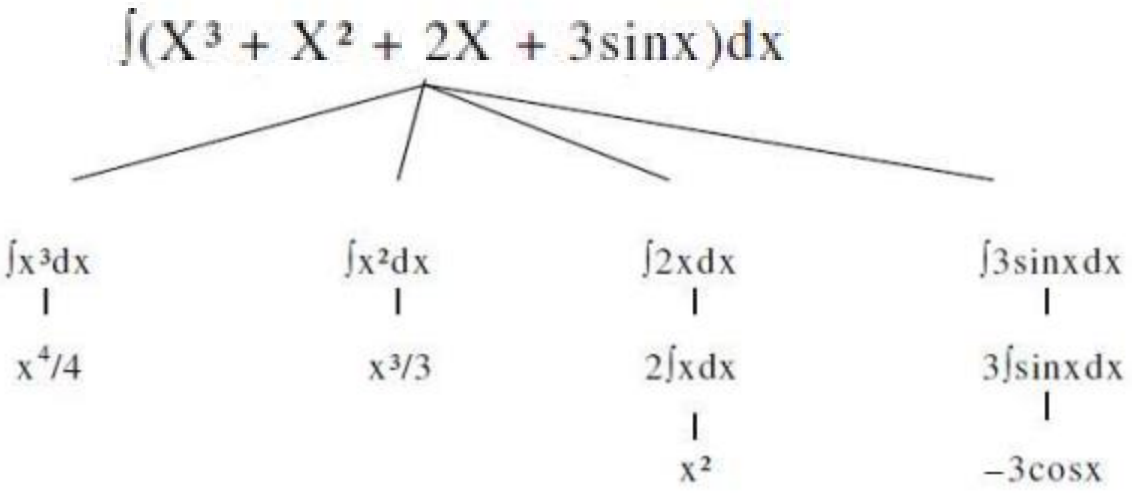
- **Space Complexity: O(d)**
- Where d is the depth of the goal
- **Proof**
- Since IDDFS, at any point, is engaged in a depth-first search, it need only store a stack of nodes which represents the branch of the tree it is expanding
- Since it finds a solution of optimal length, the maximum depth of this stack is d , and hence the maximum amount of space is O (d)
- In general, iterative deepening is the preferred search method when there is a large search space and the depth of the solution is not known

PROBLEM CHARACTERISTICS

For problem solving, analysis of the problem is important based on the following considerations:

- Decomposability of the problem into a set of independent smaller subproblems
- Possibility of undoing solution steps, if they are found to be unwise
- Predictability of the problem universe
- Possibility of obtaining an obvious solution to a problem without comparison of all other possible solutions
- Type of the solution: whether it is a state or a path to the goal state
- Role of knowledge in problem solving
- Nature of solution process: with or without interacting with the user

PROBLEM DECOMPOSITION



- This problem can be solved by breaking it into smaller problems, each of which we can solve by using a small collection of specific rules
- Using this technique of problem decomposition, we can solve very large problems very easily
- This can be considered as an intelligent behavior

CAN SOLUTION STEPS BE IGNORED?

Solving mathematical theorem:

- first we proceed considering that proving a lemma will be useful.
- Later we realize that it is not at all useful.
- We start with another one to prove the theorem.
- **Here we simply ignore the first method**

8-puzzle problem to solve:

- we make a wrong move and realize that mistake.
- But here, the control strategy must keep track of all the moves, so that we can backtrack to the initial state and start with some new move.

playing chess

- Here, once we make a move we never recover from that step

- **Thus, the three important classes are:**
- **Ignorable**, in which solution steps can be ignored
  - Eg: Theorem Proving
- **Recoverable**, in which solution steps can be undone
  - Eg: 8-Puzzle
- **Irrecoverable**, in which solution steps cannot be undone
  - Eg: Chess

IS THE PROBLEM UNIVERSE PREDICTABLE?

- **Consider the 8-Puzzle problem**
- Every time we make a move, we know exactly what will happen
- This means that it is possible to plan an entire sequence of moves and be confident what the resulting state will be
- We can backtrack to earlier moves if they prove unwise

- **Suppose we want to play Bridge**
- We need to plan before the first play, but we cannot play with certainty
- So, the outcome of this game is very uncertain
- To solve uncertain outcome problems, we follow the process of plan revision as the plan is carried out and the necessary feedback is provided
- The disadvantage is that the planning in this case is often very expensive.

IS GOOD SOLUTION ABSOLUTE OR RELATIVE?

- Consider the problem of answering questions based on a database of simple facts such as the following
- 1. Siva was a man.
- 2. Siva was a worker in a company.
- 3. Siva was born in 1905.
- 4. All men are mortal.
- 5. All workers in a factory died when there was an accident in 1952.
- 6. No mortal lives longer than 100 years.

- Suppose we ask a question: 'Is Siva alive?'
- By representing these facts in a formal language, such as predicate logic, and then using formal inference methods we can derive an answer to this question easily.
- There are two ways to answer the question:

Method I:

- 1. Siva was a man.
- 2. Siva was born in 1905.
- 3. All men are mortal.
- 4. Now it is 2008, so Siva's age is 103 years.
- 5. No mortal lives longer than 100 years.

Method II:

- 1. Siva is a worker in the company.
- 2. All workers in the company died in 1952.

Answer: So Siva is not alive

We are interested to answer the question;

it does not matter which path we follow

If we follow one path successfully to the correct answer, then there is no reason to go back and check another path to lead the solution


THANK
YOU
GRACIAS
BOLZÏN
MERCI
BÏYAN
SHUKRIA
DANKSCHEEN
TASHAKKUR ATU
SUKSAMA
ARIGATO
SHUKURIA
GRAZIE
MEHRBANI
JUSPAXAR