Sumedh

- Topics

1. Informed / Uninformed
2. DFS
3. BFS
4. Uniform Cost Search
5. Greedy search
6. Graph Search

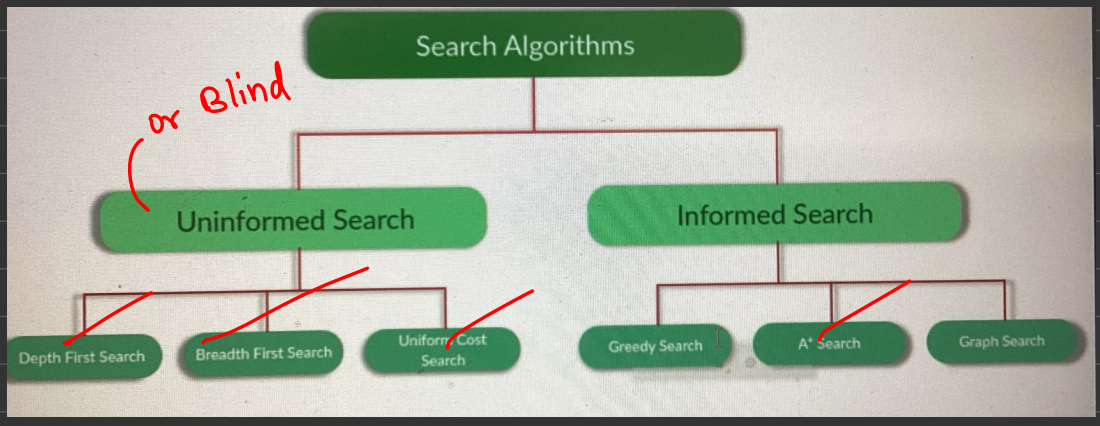
- Informed / Uninformed Search

  - Informed Search

    - No guarantee for solutⁿ but high probablity of getting solⁿ
    - Heuristic approach used to control the flow of solⁿ path
    - The approach is based on common sense, educated guesses, rule of thumb, intutive judgment.
    - Checks all possible states before finding the goal
    - Time-consuming but effective for complex problems

  - Uniformed Search

    - Generates all possible states in the state space & checks for the goal state
    - time consuming due to large state space
    - Used where error in the algorithm has severe consequences
    - May explore unnecessary paths
    - Searches blindly w/o prior knowledge

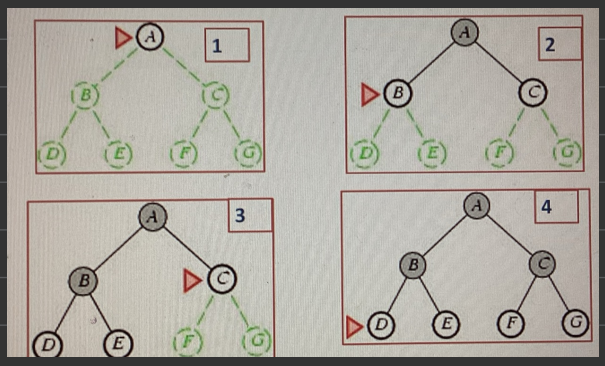


depth ltd srch Iterative DFS Bi-directⁿ srch

- Uninformed / Blind search algorithms :-

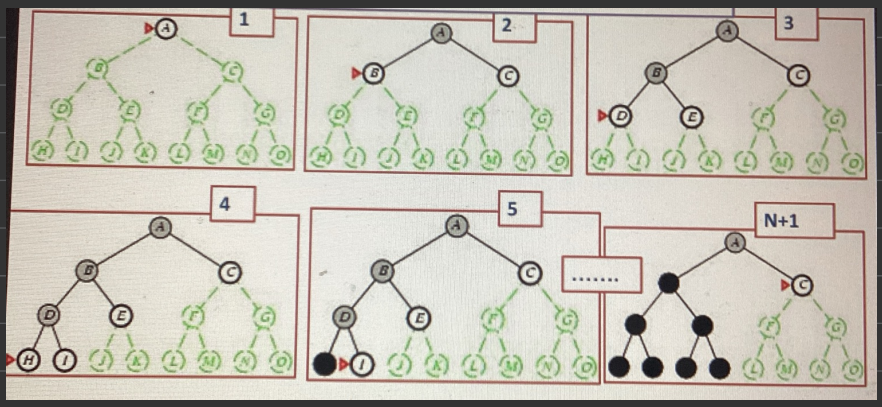
1. Breadth - First Search (BFS

- Explores all nodes at one lvl before moving to next
- Completeness : yes, always finds a solutn
- Optimality : yes, path cost is uniform
- TC : O(b^ (d+1)) [b = branching factor, d = depth of goal node]
- SC : O(b^(d+1)) [* just learn these]
- Implementatn : uses FIFO



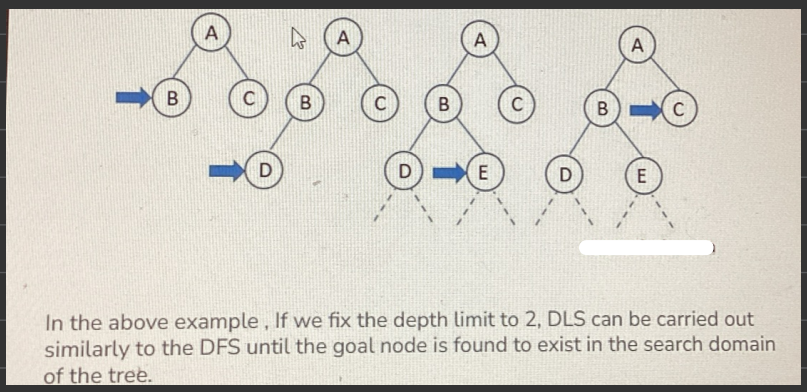
2. Depth - first Search (DFS)

- Explores one branch fully b4 backtracking
- Completeness : No, may stuck in infinite depth loop
- Optimality : No, may found a deeper solution first
- TC : O (b^ m)
- SC : O (b.m) (better than BFS)
- Implementatn : LIFO



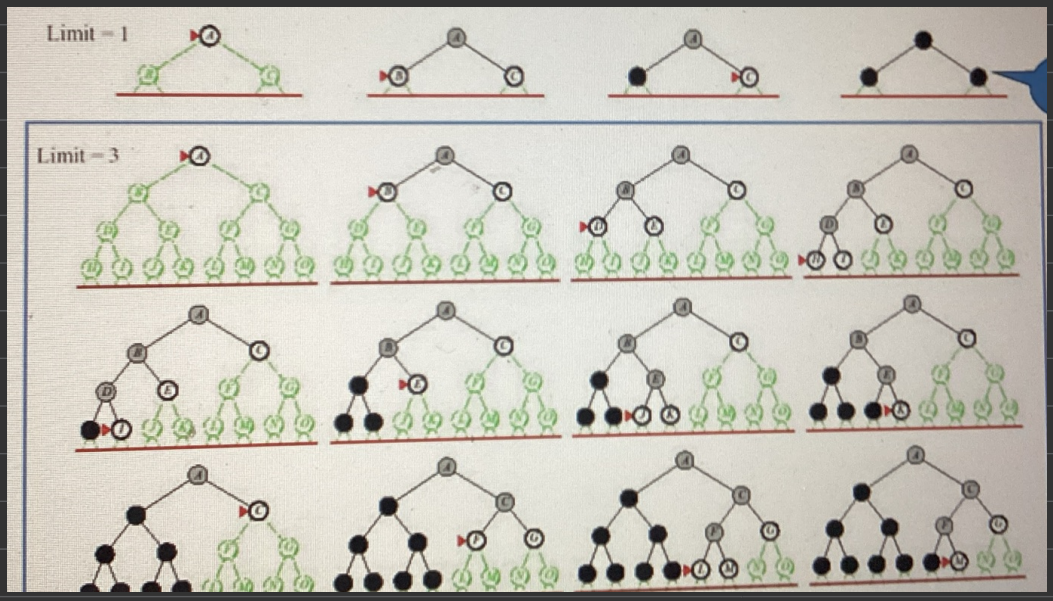
3. Depth-limited Search (DLS)

- DFS with a depth limit to avoid ∞ loop
- Completeness : No, if the goal is beyond limit
- Optimality : No, deeper optimal soln may be missed



4. Iterative deepening DFS (ID-DFS)

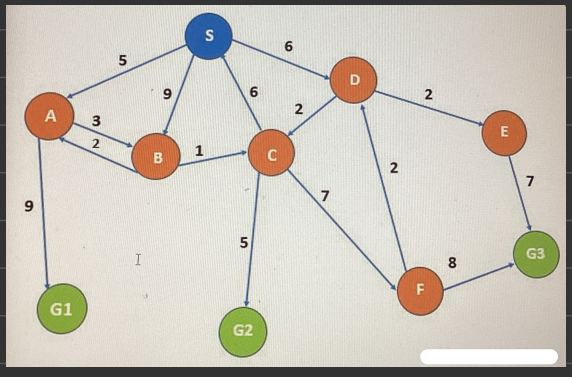
- Repeated DFS with inc. depth limits
- Completeness : Yes
- Optimality : Yes, if step cost = 1
- TC : O(b^d)
- SC : O(bd)
- better than BFS as doesn't store all nodes



5. Uniform Cost Search (UCS)

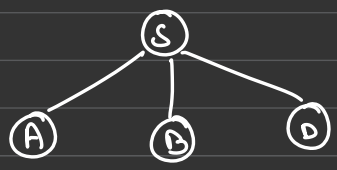
- This algo comes in play when a diff cost is there at each node
- Primary goal is to find cheapest cost path
- It expands the lowest-cost node first (priority queue)
- Completeness : Yes
- Optimality : Yes, always find least-cost path
- TC / SC : $O(b^{(1+C^*/s)})$ (C^* is optimal cost)
- Equivalnt to BFS if all step costs same

eg :

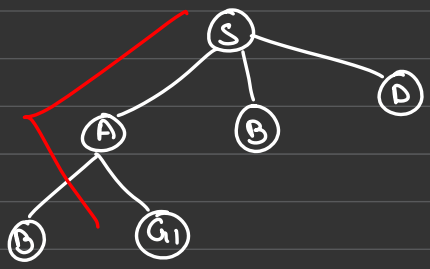


S-1 :

S-2 :

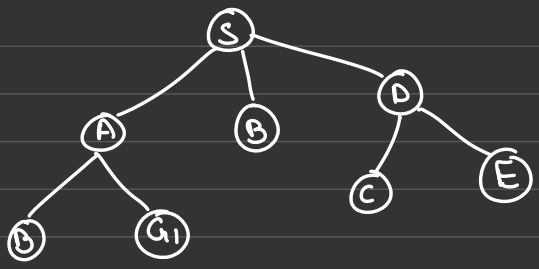


S-3 :



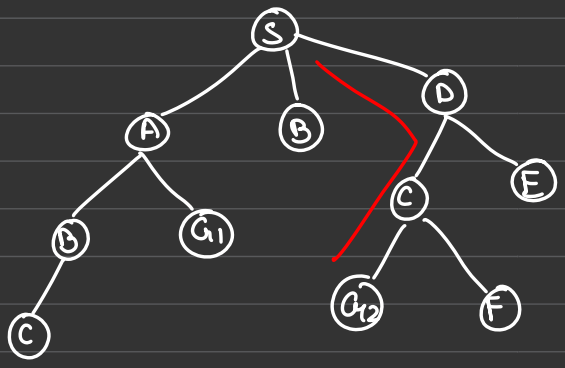
G1 check

S-4 :



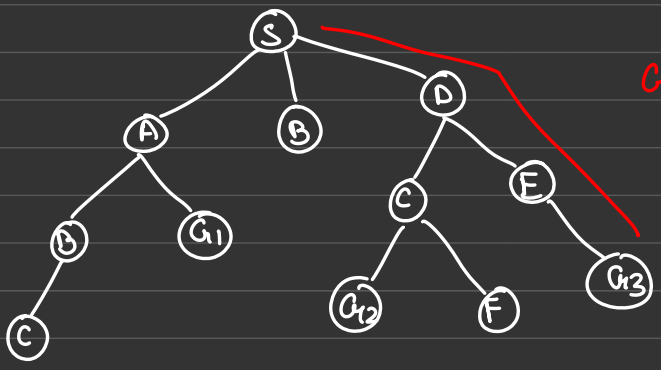
S-5 :

S-6 :



G2 check

S-1 :



G3 check

- Summary

Algorithm	Space	Time	Complete?	Optimal?
BFS	O(b^d)	O(b^d)	Yes	Yes (if uniform cost)
DFS	O(bm)	O(b^m)	No	No
UCS	O(b^(1+C*/s))	O(b^(1+C*/s))	Yes	Yes
DLS	O(l)	O(b^l)	No	No
IDS	O(d)	O(b^d)	Yes	Yes

- Maze Game

- BFS SOLUTION?
  - S-1-2-3-5-8-10-12-14-16-19-G
  - SEARCH SOLUTION FOUND IN **12 STEPS**

S	12	14	16	X
1	10	G	19	20
2	8	X	21	17
3	5	7	18	15
4	6	9	11	13

- DFS SOLUTION?
  - S-1-2-3-6-5-8-9-10-11-13-16-18-G
  - SEARCH SOLUTION FOUND IN **14 STEPS**

S	21	20	19	15
1	22	G	18	14
2	23	17	16	13
3	6	7	12	11
4	5	8	9	10

- Informed Search Algorithms

1. Greedy Best first Search ("Best, not Breadth")

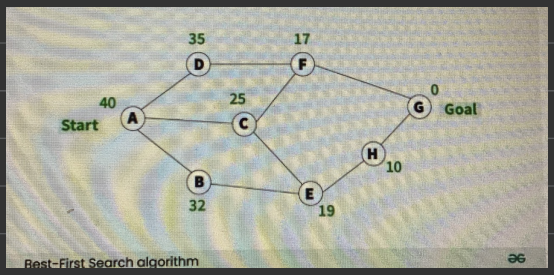
Its a heuristic based algo that aims to find most promising path based only on heuristic func (hn)

Working :-

- It evaluates the cost of each possible path using hn
- It always expands node with the lowest hn

- does not consider already incurred cost, only the estimated future cost

eg :



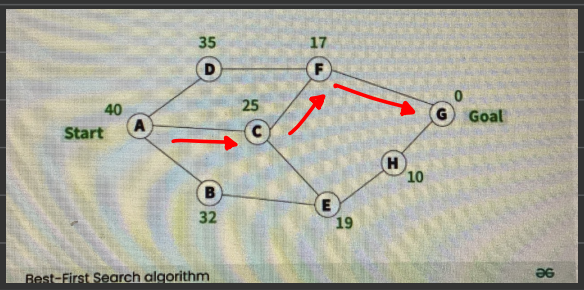
Node	Heuristic Value $h(n)$
A	-
B	32
C	25
D	35
E	19
F	17
G	0

here C has least h_n

from C → F OR E

C → F least h_n

∴ A → C → F → G



Advantages of Greedy Best-First Search

- ✔ Simple and easy to implement
- ✔ Fast execution in many cases
- ✔ Uses less memory compared to other algorithms
- ✔ Good for problems where a heuristic can provide an accurate estimate

Disadvantages

- ✘ Not optimal – Can get stuck in local optima.
- ✘ Incomplete – Might miss the shortest path if the heuristic is not accurate.
- ✘ Might explore unnecessary paths if the heuristic misguides the search.

2. A* Search Algorithm

A* (A-star) is a improved search algo that balances efficiency & accuracy by combining the features of

1. Dijkstra's algo (consider cost from start to curr node)
2. Greedy Best first srch (uses heuristic for estimated remaining cost)

Key Idea :-

A* evaluates based on total est. cost

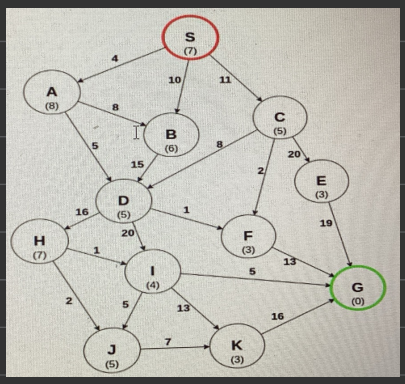
(evaluatⁿ func) $f(n) = g(n) + h(n)$

- $g(n)$: actual cost from start to node n
- $h(n)$: est. cost from n to goal node

Working :-

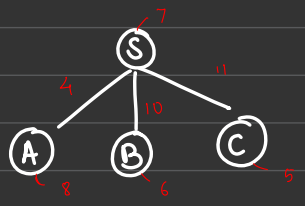
1. Maitain 2 lists
   - Open (n to be explored)
   - Closed (n already ")
2. Start with ini n, add it to open list
3. Pick n with lowest f(n)
4. Expand the node by adding neighbour to open list
5. Repeat Until goal achieved

eg :



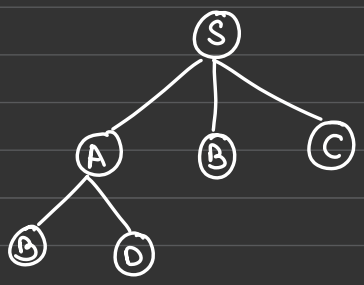
S-1 : Start at (S)

A , B , C neighbour to S

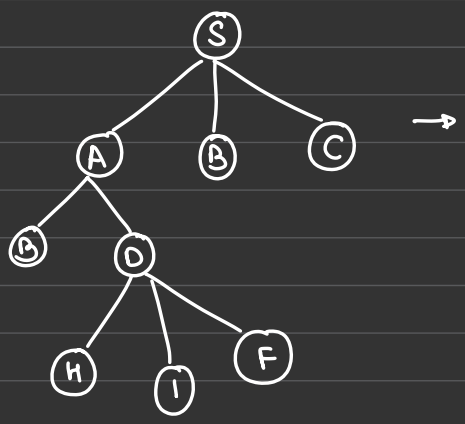


A → 8 + 4 = 12
B → = 16
C → = 16

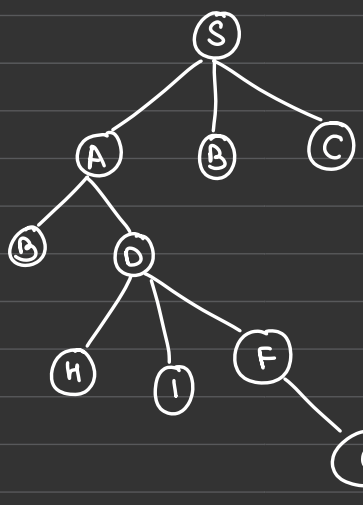
S-2 lowest is (A)



Similarly



→



(Can explore B & C to cross-check)

(Goal

Advantages of A*

- ✔ **Optimal** – Always finds the shortest path.
- ✔ **Complete** – Will find a solution if one exists.
- ✔ **Efficient** – Avoids unnecessary explorations.

Disadvantages

- ✘ **Memory-intensive** – Stores multiple paths, can be slow for very large graphs.
- ✘ **Performance depends on heuristic** – A bad heuristic can slow down the search.

- Comparing both

Feature	Greedy Best-First Search	A* Search
Uses heuristic	✅ Yes	✅ Yes
Uses path cost $g(n)$	❌ No	✅ Yes
Finds optimal solution	❌ No	✅ Yes
Memory usage	🔹 Low	🔹 High
Speed	⚡ Fast	🏆 Efficient

3. Generate & test Algo

A very simple search strategy that works by generating possible soln n testing them to see if they satify goal condn

Algo steps :-

1. Generate a possible soln
2. Test if soln is goal state
3. If not, generate another
4. Repeat until goal achieved

eg : travelling salesman problem (TSP)