Search Algorithm

uninformed/Blind (6)

→ Breadth first search
→ uniform cost search
→ Depth first search
→ depth limited search
→ Iterative deeping depth first search
→ Bi-directional search

Informed search (5)

→ Best - first search
→ A* search
→ AO* algo
→ problem reduction
→ Hill climbing

Types of search algo
→ uninformed (Blind search) search
→ informed (Heuristic search) search

Depth first search: (↓) (DFS)

* recursive / non recursive
* tree / graph traversal

* DFS ⇒ Root → each path to its greatest depth node before moving to (↓) next node.

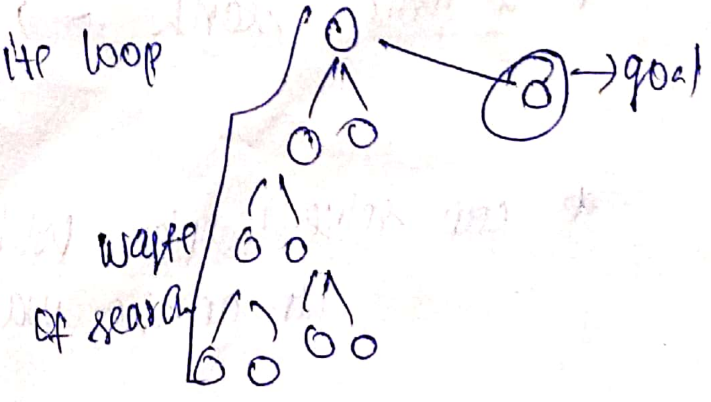
* stack DS. ~~(FIFO)~~ (LIFO)

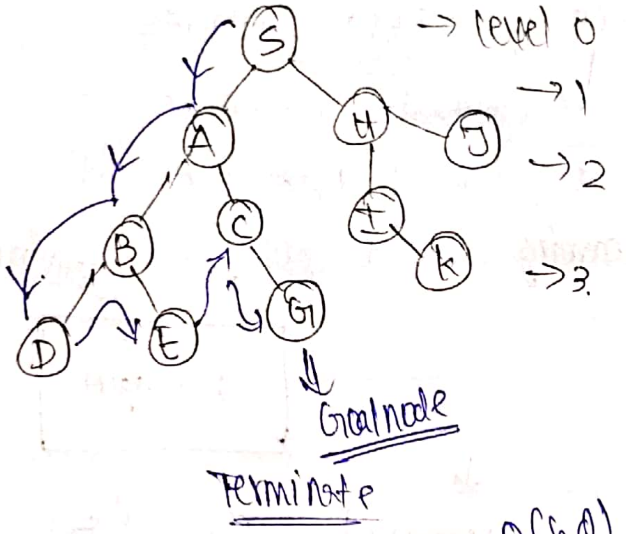
Adv:

→ less memory (root to current node)
→ less time to reach the goal node.

Dis:

→ many states keep re-occuring, no guarantee → finding solution
→ sometimes it may go to infinite loop





Back-tracking possible

completeness → It is complete within finite state space

Time complexity →

$T(n) = 1 + n^2 + n^3 + \dots + n^m$

$\boxed{O(n^m)}$ m → max depth of any node.

$O(b^d)$

$O(bd)$

Space complexity → $O(bm)$ b → depth of goal node

Optimality → $\boxed{\text{non-optimal}}$ (infinite loop)

Uninformed / Blind search.

* No domain knowledge → closness, → location of goal } not known.

* Brute force way (operation) · includes info abt how to traverse the tree & how to identify leaf & goal node.

* search without any info about search space

→ initial state operators } not known.
→ test for the goal

* Examine each node $\xrightarrow{\text{to}}$ achieve goal node

Informed search.

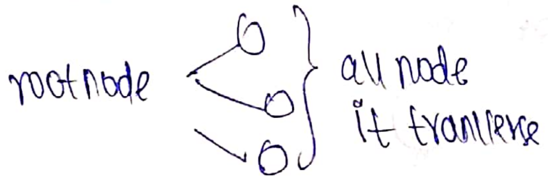
* Uses domain knowledge (prblm info)

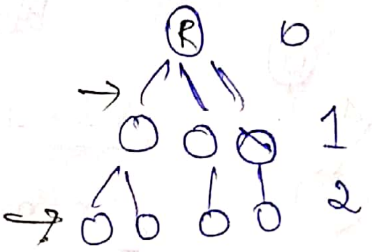
* more efficient than uninformed. (bcaues search strategies we have)

* Heuristic search → Not guarantee to find best soln/ but finds good soln/ in small time.

* can solve complex prblm which could not be solved in another way

1) Breadth - first search: (BFS) ⟶ (Breadth wise)





* General - graph search algo.

* FIFO queue DS

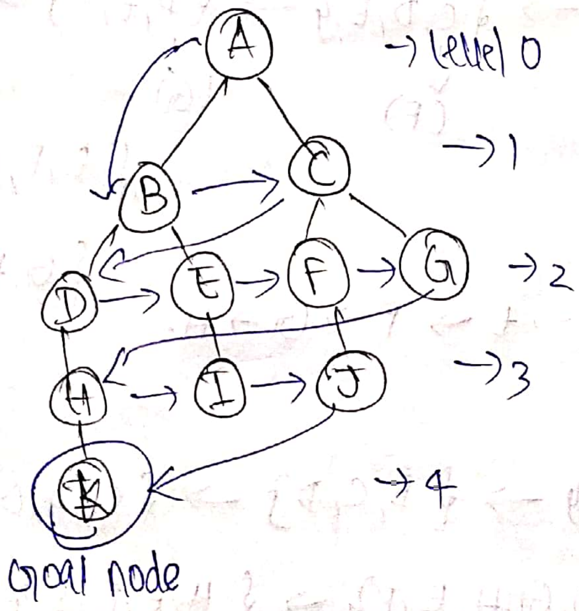
Adv:

* more than one soln. ⟶ provides minimal soln. which has least no. of steps.

Dis:

* more memory → all/each level of tree must be saved into memory to expand the next level

* more time → if soln. is far away from root node.



Time complexity →

d⇒ depth of shallow node

b⇒ is a node at every state

$T(b) = 1 + b^2 + \dots b^d$.

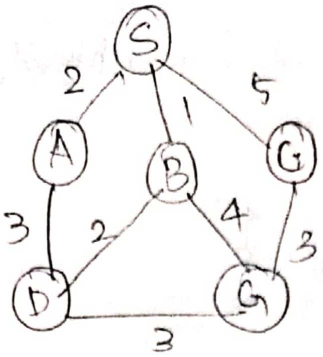
$= \boxed{O(b^d)}$

Completeness ⟶ complete.

Any how soln. reached

Space complexity ⇒ $S(b) = O(b^d)$

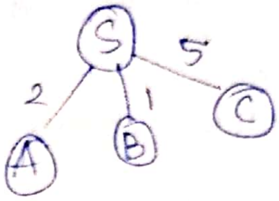
Optimality ⇒ Optimal.



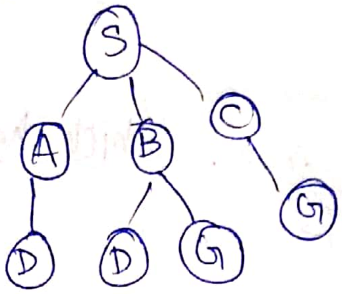
Find the root from S to G. Using BFS.?

Step - 1 → (S)

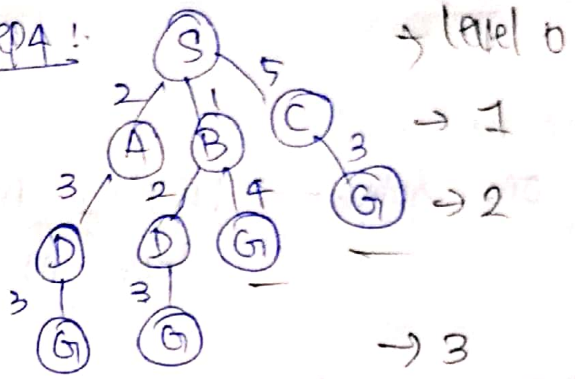
Step - 2 →



Step3:

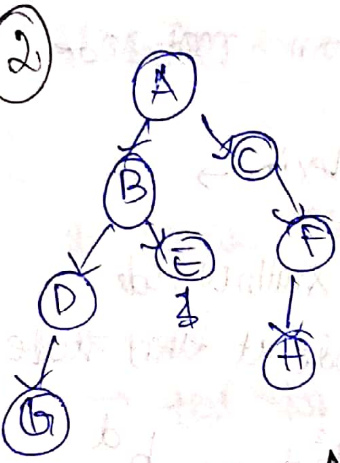


Step4 :



Ans → SBG (or) SCG.

(2)



Initial Queue {A} → Goal node x

{B, C} → {C, D, E} → {D, E, F} →

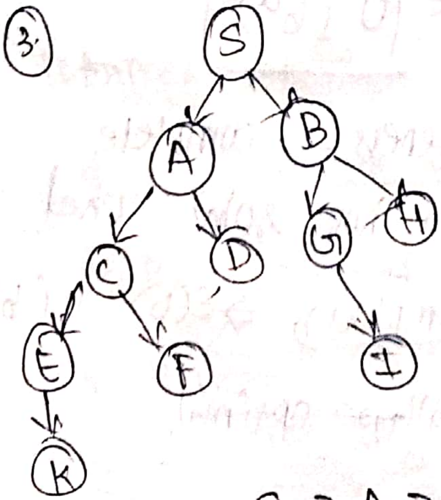
↓ {D, E} ↓ (F) ↓ (G) ↓

{E, F, G}

{G, H}

A → B → C → D → E → F → G → H.

(3)



{S}

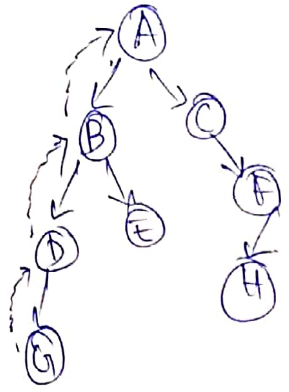
{A, B} → {B, C, D} → {C, D, G, H}

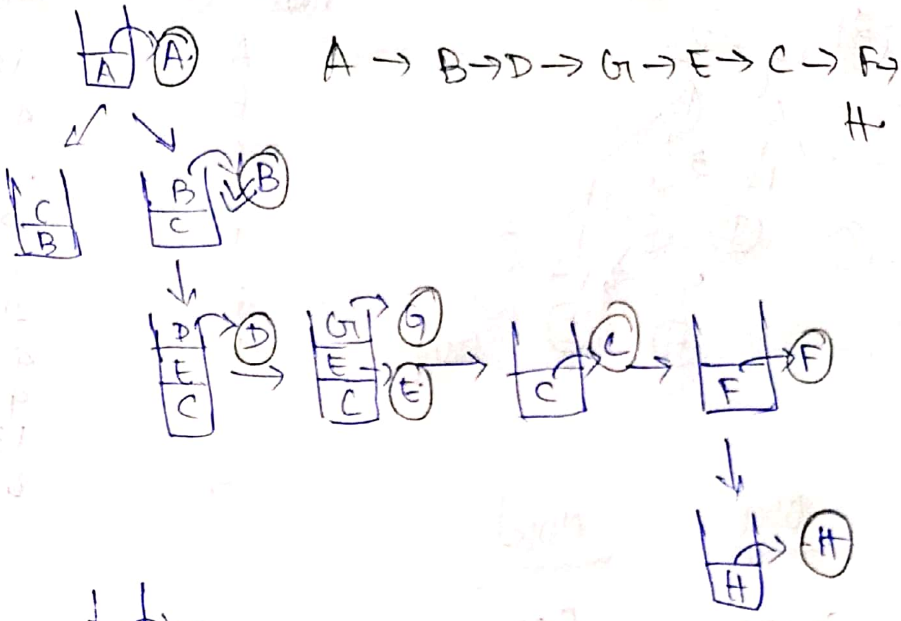
→ {D, G, H, E, F} → {H, E, F, I} →

{~~F~~, I, K} → Done

S → A → B → C → D → G → H → E → F → I → K.

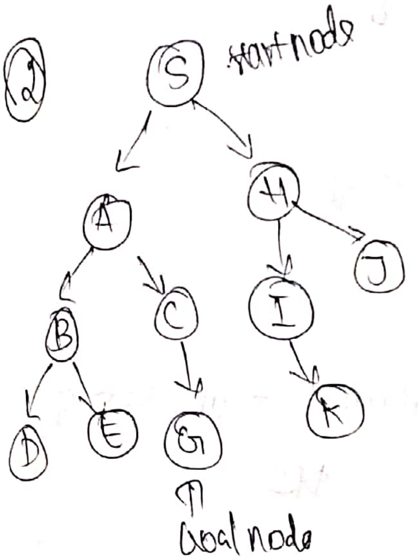
DFS :- Example :-

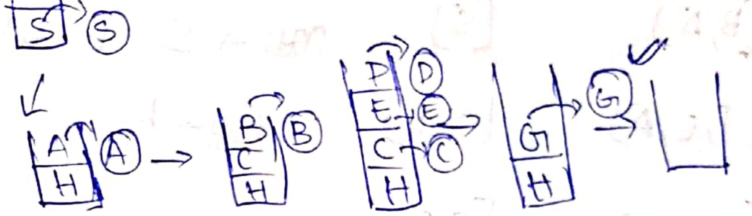




A → B→D→G→E→C→F→ H.

(2)





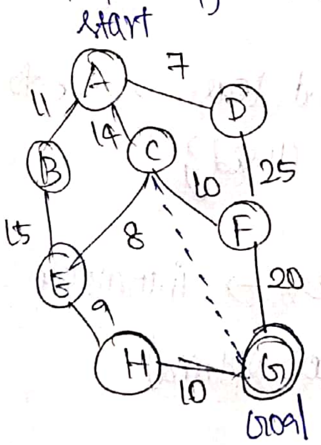
G→ final node so stop

S → A → B → D → E → C → G.

3.) Best-first search! (GREEDY SEARCH ALGO) (Heuristic (informed search) (BFS+DFS)

* Evaluation fn. to find which adjacent node is most promising.

* priority Queue → cost of nodes store.



Straight line distance:-

A→G = 40
B→G = 32
C→G = 25
D→G = 35
E→G = 19
F→G = 17
H→G = 10
G→G = 0

space complexity → $O(b^d)$

T.C → $O(b^d)$

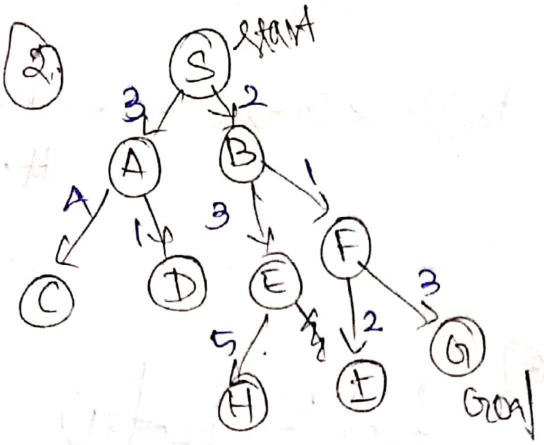
b: branching factor
d: depth

Open	Closed
[A]	[]
[C, B, D]	[A]
[B, D]	[A, C]
[F, E, B, D]	[A, C]
[E, B, D]	[A, C, F]
[G, E, B, D]	[A, C, F]
[E, B, D]	[A, C, F, G]

$\boxed{A \to C \to F \to G}$

Path (44)

2.



node (n)	H(n) → Hieuristic value
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

Open	closed
[B, A]	[S]
[F, E, A]	[S, B]
[G, I, E, A]	[S, B, F]
[I, E, A]	[S, B, F, G]

path → S → B → F → G

cost → 6.

Adv!

more efficient than BFS & DFS

Dis:

* can get stuch as DFS
* unguided depth-first seach In the worst case scenario
* Not optimal

$h(n) \leq h^*(n)$

heuristic cost   estimated cost

open [A, B] , close [S]

1. open [A] , close [S, B]
2. open [E, F, A], close [S, B, F]
3. open [E, A] , " [S, B, F]
4. open [E, A, I , G], " [S, B, F]

open [E, A, I] , " [S, B, F, G]

{write then find least → move to closed}

completeness → incomplete

⇒ not optimal