UNIT-2

Chapter-1 PROBLEM SOLVING

Problem Solving - State space Search & Control Strategies - introduction - General problem solving, - Characteristics of problems - Exhaustive searches - heuristic search techniques - iterative deepening A* - constraint satisfaction.

- **Problem Solving:** Introduction
  - It can be done by building an AI system to solve that particular problem.
  - To do this one need to define the problem statement first & generating the solution by keeping the condition in mind.
  - Some of the most popularly used problem solving with the help of AI are: chess, Travelling salesman problem., Towers of Hanoi problem, water jug problem, N-Queen problem....,

* **Problem Searching:**
  - ↳ in general searching refers to as finding information one needs.
  - ↳ searching is the most commonly used technique of problem solving in AI.
  - ↳ The process of solving a problem in general consist of
    - ↳ • Define the problem ↳ • Analyzing the problem ↳ • identification of solution
    - ↳ • choosing the solution ↳ • implementation

* Searching is most commonly used technique in problem solving using AI.

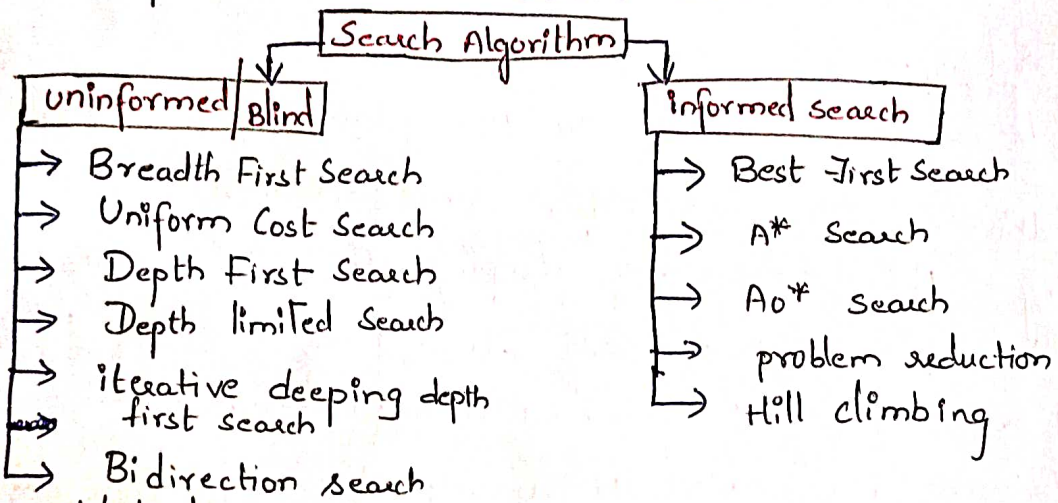
* **Properties of Search algorithms:**
  - ↳ The essential properties of search algorithm to complete the effiency of these algorithms.
  - **Completeness** - A search algorithm is said to be complete if it guarentees to return a solution if at least any solution exist for random input
  - **Optimality** - if a solution found for an algorithm is guaranteed to the best solution (lowest path cost) among all other solutions, then

such solution for is said to be optimal solution.

- Time Complexity – It is a measure of time for an algorith complete its task.
- Space Complexity – It is the maximum storage required at point during the search, as the complexity of the proble

* Types of search algorithms:

Based on the search problems we can classify search into < uninformed search (Blind search)
informed search (Heuristic search)



* Uninformed / Blind Search: Does not contain any domain kn
- it opeeates in brute force way, as it only includes information how to teaverse the tree & how to identify leaf & goal nodes.
- it applies a way in which search Tree is searched without information about the search space like initial state opeeato for the goal, so it is called blind search. It examines ea until it achieves the goal node

* Informed Search: Uses domain knowledge & problem info is availa
- informed search stategies can find a solution more efficiently th uninformed. It is also called Heuristic search.
- A heuristic is a way which might not always be guarent the best solution but guaenteed to find a good solution in time.
- It can solve much complex problem which could not be solve another way.

State Space Search:

$S: \{S, A, Action(s), Result(s,a), Cost(s,a)\}$

↓ ↓ Set of all possible actions

Total no of states → S = Start, intermediate, Goal

- Method of problem representation that facilitates easy search
- With this one can also find a path from start state to goal state while solving a problem.
- state space basically consists of four components
  - Set S starting state of problem
  - A set G containing goal states of problem
  - Set of nodes (states) in graph/tree. Each node represents the state in problem solving process.
  - Set of arcs connecting nodes. Each arc corresponds to operator that is a step in a problem solving process.

A solution path is a path through the graph from a node in S to a node in G.

The main objective of search algorithm is to determine a solution path in the graph.

There may be more than one solution paths, as there may be more than one ways of solving the problem.

Commonly used approach is to apply appropriate operator to transfer one state of problem to another.

- Example: Eight puzzle problem:

Problem statement: The eight-puzzle problem has a 3×3 grid with 8 randomly numbered (1 to 8) tiles arranged on it with one empty cell. At any point, the adjacent tile can move to the empty cell, creating a new empty cell. Solving this problem involves arranging tiles such that we get the goal state from the start state.

Start state

3	7	8
5	1	2
4	□	8

Goal state

5	3	6
7	□	2
4	1	8

Eight puzzle problem.

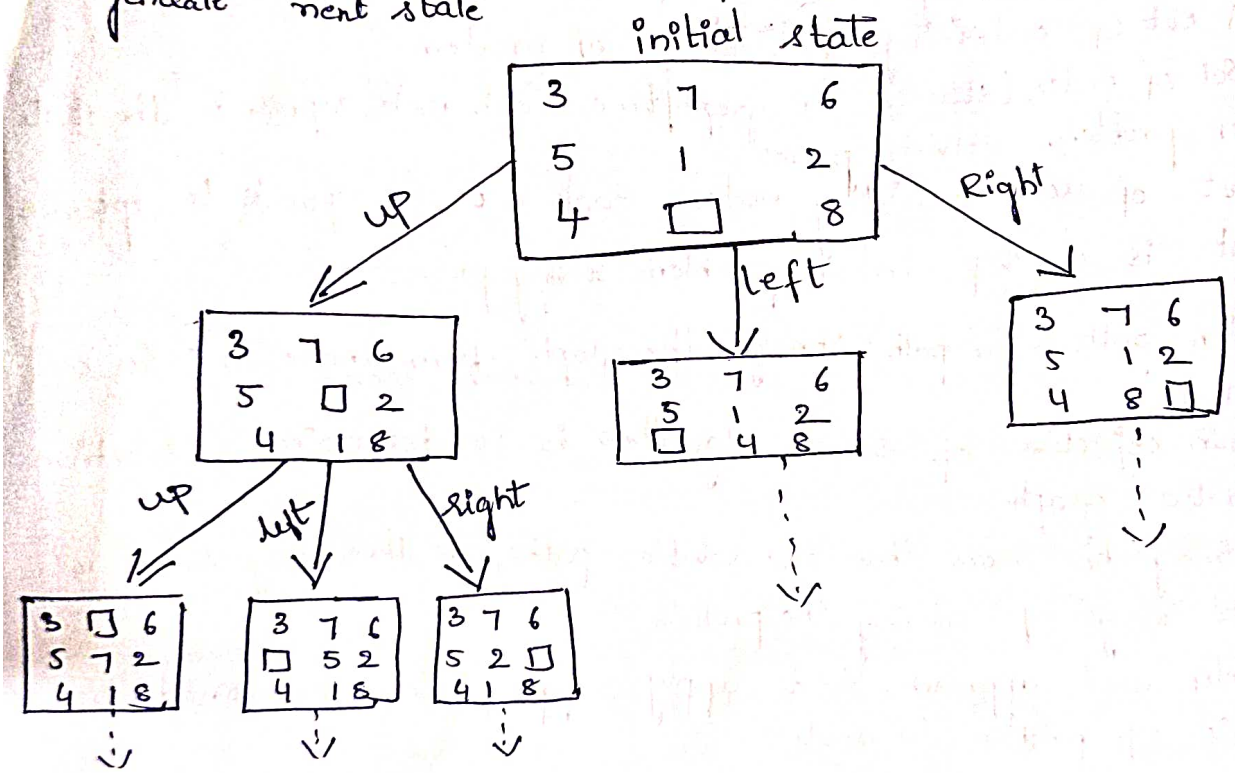
A state for this problem should keep track of the position of a tiles on the game board, with 0 representing the blank (empty position on the board. The start & goal states may be represe as follows with each list representing corresponding row:

(1). Start state : [[3,7,6]; [5,1,2]; [4,0,8]]

(2). The goal state could be represented as: [[5,3,6],[7,0,2],[4,1

(3). The operators can be thought of moving [Up, Down, Left, Right], the direction in which blank space efficiently moves.

To simplify, a search tree up to level 2, we use operators generate next state



— Continue the searching like this till we reach the goal

The exhaustive search can proceed using depth-first search breadth first search strategies.... Some intelligent searches can be made to find solution faster.

Current

Control Strategies:

- Control strategy is one of the most important components of problem solving that describes the order of application of the rules to the current state.
- These should cause motion towards solution.
- Better choose best control strategy rather than simple control strategy in the rule list.
- It should explore the solution space in systematic manner.
- To solve real world problems, effective control strategy must be used.

/* The problem can be solved by searching for a solution, the main thing is to find correct search strategy for a given problem */

There are two directions in which search should proceed.

- Data Driven Search [forward chaining] from start state
- Goal driven Search [backward chaining] from goal state

* Forward chaining:

↳ The process of forward chaining begins with known facts & works towards a conclusion

↳ In this case we begin building a tree of move sequences with the root of tree as start state

↳ The state of next level of the tree are generated by finding all rules whose left sides match with root & use their right side to create new states

Eg: Eight puzzle problem

↳ This process is continued until a configuration that matches the goal state is generated.

↳ language OPS5 uses forward reasoning rules. Rules are expressed in the form of if-then rules.

* Backward chaining:

↳ It is a goal directed strategy that begins with goal state & continues working backward, generating more subgoals that must also be satisfied to satisfy main goals until we real to start stat

↳ Prolog language uses this strategy.

- He can use both data driven & goal strategies for proble solving, depending on the nature of problem

- If there are large number of explicit goal states & on start state, then it would not be efficient to solve usin back chaining strategy because we do not know which goal s is closest to the start state.

- Therefore, the general observations are that move from ee set of states to the larger set of states & proce in the direction with lower branching factors (the ave number of nodes that can be reached directly from sing nodes).

/* So depending upon the given problem the control strat forward / backward chaining is to be considered */.

- Characteristics of Problem:

Before starting modelling the search and trying to find s for the problem one must analyze it along several key characters initially. Some of them are:

* Type of problem - There are two types of problems in real li Ignorable, Recoverable, Irrecoverable.

- Ignorable: These are problems where one can ignore the solutions These can be solved using simple control strategy.

- Recoverable: These are problems where solution steps can be undon These problems are generally puzzles played by a single player. Suc problems can be solved using back tracking using push-downs

- Irrecoverable: The problems where solution steps cannot be un Two player playing games. problems solved using planning proc The major characteristics are

→ Decomposability of a problem: Divide the problem into a set o independent smaller sub problems, solve them & ~~contribute~~ combi the solutions to get final solutions. The process of dividing sub problems continues till we get the set of smallest sub pro for which a small collection of specific rules are used.

Divide & Conquer technique is commonly used method for solving such problems. It is an important character to solve & handle different problems. These problems are generally solved using parallel processing.

→ Role of knowledge: Knowledge plays an important role in solving any problem. Knowledge could be in the form of rules & facts which help in generating search space for finding the solution.

→ Consistency of knowledge Base used in solving problem: Make sure that knowledge Base used to solve problem is consistent. Inconsistent knowledge Base will lead to wrong solution.

→ Requirement of solution: He should analyze the problem whether solution required is absolute (exact) or relative (reasonable). The path problems are best path & any-path problems.

(travelling sales man problem) (water jug)
absolute relative.

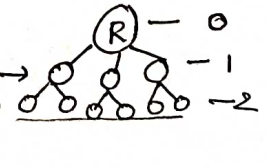
Best path problems are relatively harder than any path problems.

Exhaustive Searches:

Some of systematic uninformed exhaustive searches are

* BFS
* DFS
* Depth first iterative deeping/deepening
* Bidirectional Search

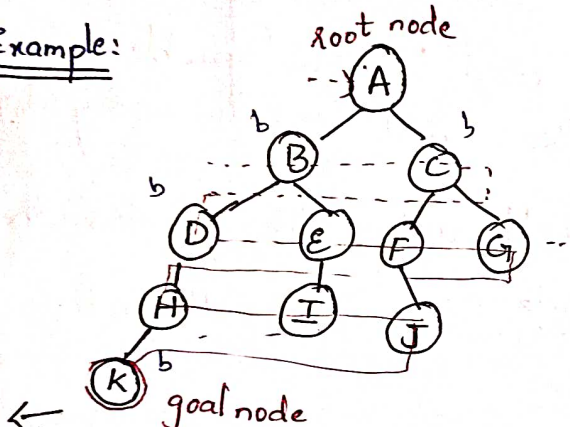
* Breadth First Search:

- It is most common search strategy for traversing a tree or graph.
- This algorithm searches breadthwise in a tree/graph, so it's called BFS
- BFS algorithm start searching from root node of the tree & expands all successor nodes at the current level before
- BFS alg is an example of general graph search alg → 

- BFS implemented using FIFO queue data structure.
queue

- **Advantages:**
  - BFS will provide a solution if any solution exists
  - If there are more than one solution for a given proble the BFS will provide minimal solution which requires numbu
- **Disadvantages:**
  - It requires lots of memory since each level of the tree saved into memory to expand the next level.
  - BFS needs lots of time if the solution is far away fro root node

- **Example:**



← Shallowest little depth.

level 0
level 1
level 2
level 3
level 4

Time complexity

d = depth of shallowes (k)

b is a node at eve

$T(b) = b^0 + b^1 + b^2 + b^3$

$= 1 + 2 + 4 + 3$
(A) (B,C) (D,E,F,G)

$T(b) = b^0 + b^1 + b^2 + \cdots + b^d$

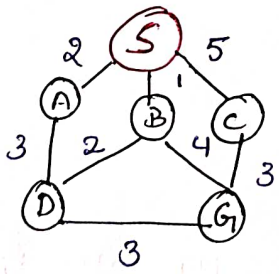
$\boxed{T(b) = O(b^d)}$

order of

Space Complexity

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

- Completeness – BFS is complete (Reached goal node)
- Optimality – optimal (if path cost is non decreasing)



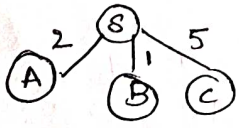
Find the route from S to G using BFS

Aim: To find G start from S

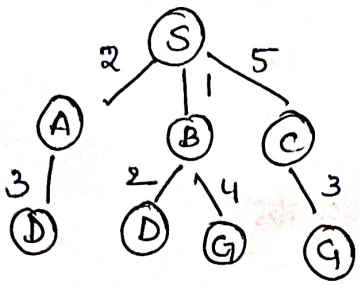
Step1: Visit root node

(S)

Step2: Visit all successor nodes of root node

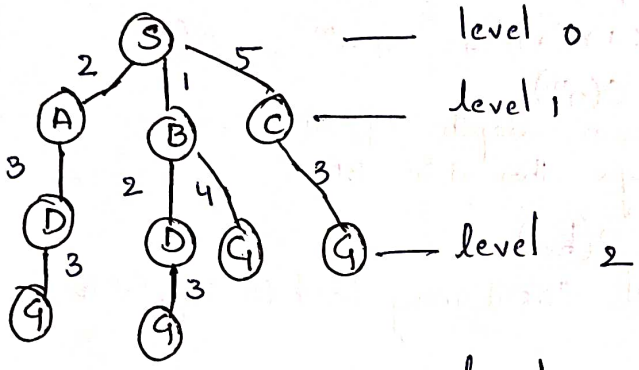


Step 3: Expand the tree with successor nodes



* It uses queue.

tep 4: Continve the expansion.



level 0
level 1
level 2
level 3

Path:

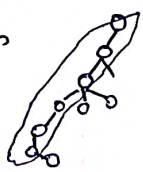
SBG (or) SGG

1 + 4 (5) 5 + 3 (8)

S to G is reached shorted path is SBG

* Depth First Search:

- It is a recursive algorithm for traversing a tree (or) graph DS.
- It is called DFS because it starts from the root & follows. Each path to its greatest depth node before moving to the next.
- DFs uses a stack DS for implementation
- The process is similar to BFS algorithm.



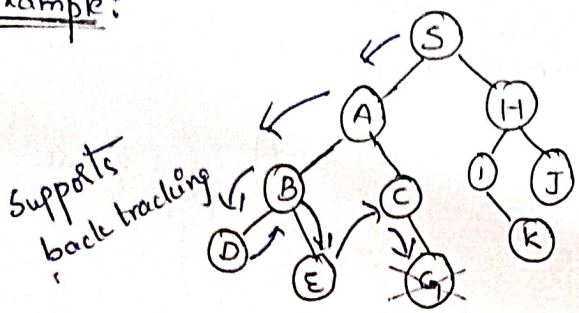
- Advantages:

- It requires very less memory as it only needs to store a stack of nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm

- Disadvantages:

- There is the possibility that many states keep re-occuring, & there is no guarantee of finding solution.
- DFS algorithm goes for deep down searching & sometimes it may go to the infinite loop.

- Example:



level 0
level 1
level 2
level 3

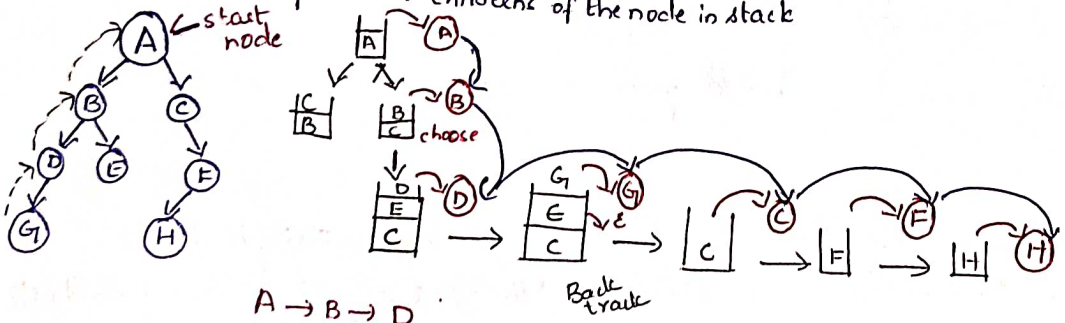
Supports back tracking

S is source node
G is goal node

— it terminates

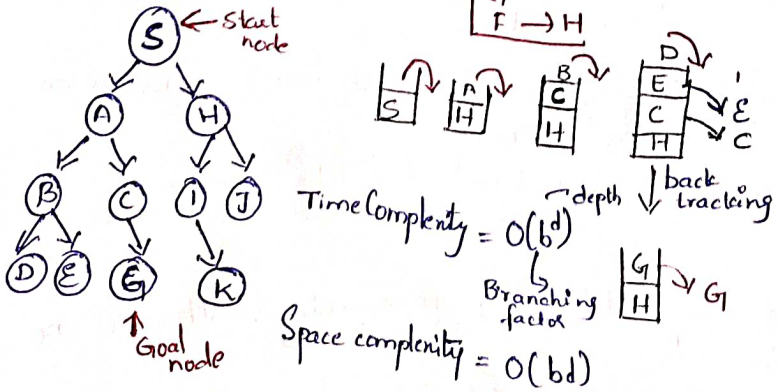
Completeness – It is complete within finite state space
Time complexity – $T(n) = 1 + n^2 + n^3 + \dots + n^m = O(n^m)$
$T(n) = O(n^m)$
m – maximum depth of any node
↘ much larger than d in BFS.
Space Complexity – $S(c) = O(bm)$
Optimal — non optimal (as it may lead to infinite loop)

- Algorithm:
  - Enter ROOT node on stack (push)
  - Do until stack is not empty
    - (a) Remove node (POP)
      - (i) if node is Goal node stop
      - (ii) push all childrens of the node in stack



A → B → D → G → E → C → F → H

DFS



S → A → B → D → E → C → G

Time Complexity = $O(b^d)$ (d: depth)
Branching factor

Space complexity = $O(bd)$

Depth limited Search: (DLS)

It is similar to DFS with a predetermined limit. It can solve the drawback of infinite path in DFS. In this algorithm, the node at the depth limit will treat as it has no successor nodes together.

Depth limited search can be terminated with two conditions of failure:

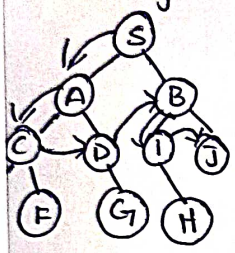
- standard failure value (indicates that problem does not have any solution)
- cutoff failure value (defines no soln for the problem within a given depth limit)

Advantages:

- memory efficient

Disadvantages:

- incompleteness
- it may not be optimal if the problem has more than one solution.



level 0
level 1 — limit = level 2
level 2
goal node
level 3

$S \rightarrow A \rightarrow C$
$D \rightarrow B$
$I \rightarrow J$

$T(C) = O(b^l)$
$S(C) = O(b \wedge l)$

Depth First Iterative Deepening: [DFID]

- Combination of both DFS & BFS
- Best depth limit is found out by gradually increasing limit
  initially $d = 0$
  for every iteration increase by 1

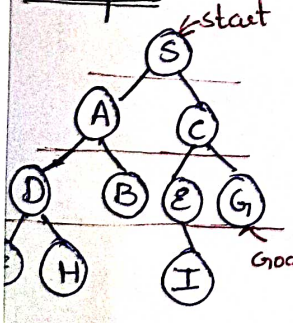
Advantages:

incorporates benefits of both BFS + DFS
(fast search, less memory)

Disadvantages:

Repeat the work process.

Example:



0
1
2
3

1st iteration $d = 0$ [S]

2nd iteration $d = 0+1 = 1$ [S → A → C]

3rd iteration $d = 1+1 = 2$ [S → A → D → B → C → E → G]

G — Goal node

if not 4th iteration.

Space complexity — $O(d)$

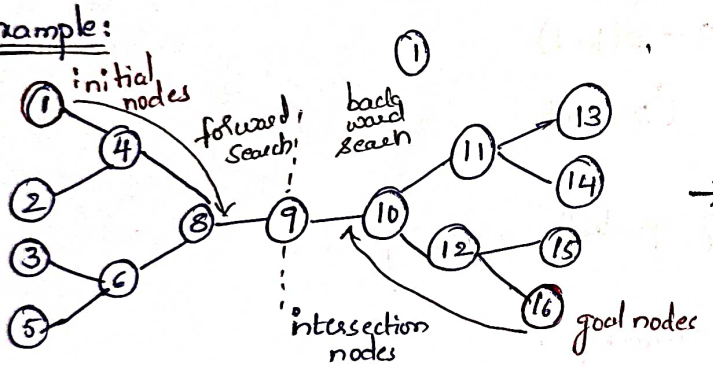
Time complexity — $O(b^d)$ Always less deeper than DFS

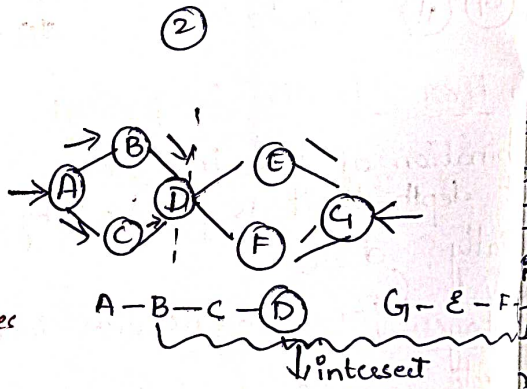
* Bidirectional Search:

- It runs two simultaneous searches, one from initial state called as forward search & other from goal node called backward search to find the goal.
- It replaces one single search graph with two small subgraphs in which one starts the search from initial vertex & other starts from goal vertex.
- The search stops when these two graphs intersect each other.
- Bidirectional search can use search techniques such as BFS, DFS, etc.

Searches < fbw (start)
bfw (goal)

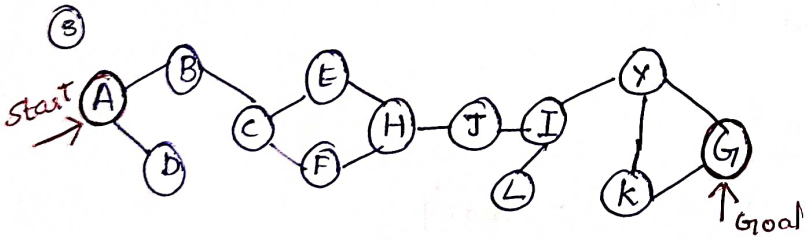
- **Advantages:**
  - Bidirectional search is fast
  - Requires less memory.
- **Disadvantages:**
  - implementation of the bidirectional search tree is difficult.
  - in bidirectional search, one should know the goal state in advance.
- **Example:**





Search [Stop] condition when [Graph] intersects

A → B → C → [D] → F → E →



forward	Backward
A	G
ABD	GKY
ABDC	GKYI
ABDCEF	GKYILJ
ABDCEF(H)	GKYILJ(H)

A → B → D → C → E → F → [H] → J → L → I → Y → K → G

forward | backward

Heuristic Search Techniques:

Heuristic technique is a criterion for determining which among several alternatives will be most effective to achieve some goal.

This technique improves the efficiency of a search process possibly by sacrificing claims of systematic & completeness.

It no longer guarantees to find the best solution but almost always finds a very good solution.

Using good heuristics, we can hope to get good solution to hard problems in less than exponential time.

Two types of heuristics < General purpose / Special purpose

General purpose heuristics:

A general purpose heuristics for combinatorial problem is nearest neighbor algorithm that works by selecting the locally superior alternative.

For such algorithm, it is often possible to prove an upper bound, on the error.

It provides reassurance that we are not paying too high a price in accuracy for speed.

In many AI problems, it is often difficult to measure precisely the goodness of a particular solution.

For real world problem, it is often useful to introduce heuristics on the basis of relatively unstructured knowledge.

- In AI approaches, behavior of algorithms is analyzed by running them on computer as contrast to analyzing algorithm mathematically.
- There are at least many reasons for such adhoc approaches in AI

---

* It is as fun to see a program do something intelligent than to prove it.
* Since AI problem domains are usually complex. It is generally not possible to produce analytical proof that a procedure will work.
* It is not even possible to describe the range of problems well enough to make statistical analysis of program behaviour meaningful.

- It is important to keep performance in mind while designing algorithm.
- One of the most important analysis of the search process is to

find no. of nodes in a complete search tree of depth 'd' ching factor 'f'. that is $f * d$.

- This simple analysis motivates to look for improveme the exhaustive search procedures & to find an upper bound search time which can be compared with exhaustive searc
- The searches will use some domain knowledge are called informed Search Strategies.

* Branch & Bound Search (Uniform Cost Search):

[It is a way to combine space saving of depth first search wi -tic information.

↳ Optimal solution is choosen among many solution paths.

↳ Idea in Branch & Bound search is to maintain lowest path goal found so far, and its cost.

↳ Typically used with Depth first Search

↳ Two parts < Branch : several choices are found
Bound : setting bound on solution quality
↳ Pruning: Trimming off branches where soluti is poor.]

- Used for Weighted tree/graph Traversal
- Goal is to path finding to goal node with lowest cumulative
- Node expansion is based on path costs.
- Priority queue is used for implementation.
- Used to find optimal path
- Priority queue — high priority to minimum cost
- Supports back tracking.

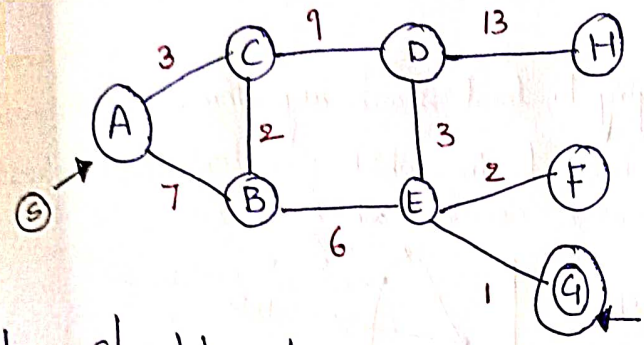
o Advantages:

Optimal solution

o Disadvantages:

↳ Stuck in infinite loop

Example:



— Cost function denoted by $g(x)$

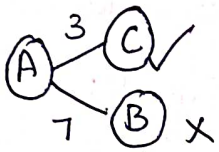
Start
- Generate a possible solution
- Test if it is a goal
- If not go to start else quit

end

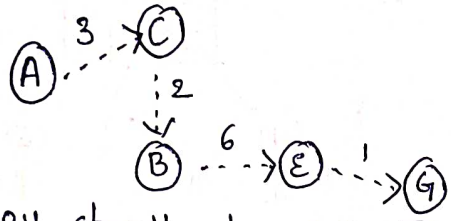
We should choose path with lowest cumulative cost ie path optimal path

& Based on priority queue, high prigity should be given to path with minimum cost

if



Given G is goal node ; the path is



path to goal node

$A \xrightarrow{3} C \xrightarrow{2} B \xrightarrow{6} E \xrightarrow{1} G$

(12)

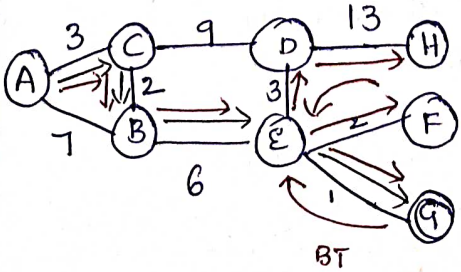
But you should also visit (H) & (F)

Goal node (H)

You should not go to first as we already started & the traversal is at node (G) you should move from G to H by covering F in between as the seach is uniform cost search. We also use back tracking. So path is

$A \xrightarrow{3} C \xrightarrow{2} B \xrightarrow{6} E \xrightarrow{1} G \xrightarrow{1} E \xrightarrow{2} F \xrightarrow{2} E \xrightarrow{3} D \xrightarrow{13} H$

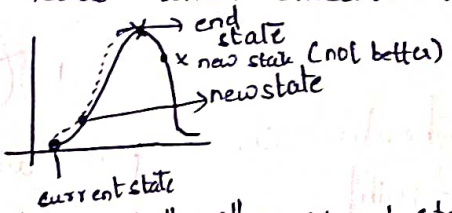
(33)



BT

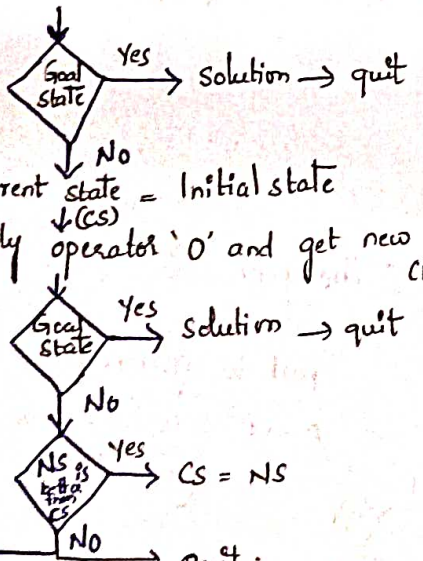
* Hill climbing:

- Quality measurement turns DFS into Hill climbing (variant of generate strategy).
- Optimization technique belonging to local search algorithm.
- Variant of generate & test method in which feedback from test procedure is used to help generator decide which direction to move in search space.
- Follows local search algorithm
- Greedy approach
- No back tracking



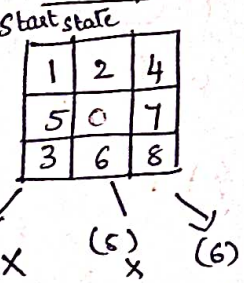
if newstate is better than current state then
newstate = current state

∘ Flow chart:



NS = Newstate ; CS = current

∘ Example:



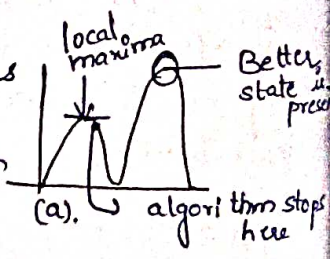
NS better than
else quit
Don't ba

↳ He will choose this.

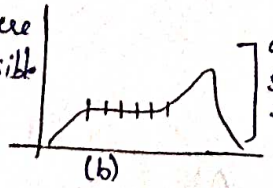
∘ Limitations with hill climbing:

The search process may reach to a position that a solution but from there no move improves the situation. This happen if we have reach local maximum, a plateau, or a ridge.

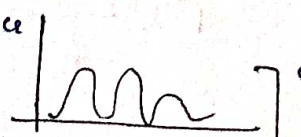
(a) Local maxima: It is a state better than all its neighbours but not better than some other which are far away. From this state all moves looks to be worse. In such situation, backtrack to some earlier state & try going in different direction to find a solution.



(b). Plateau: It is a flat area of search space where all neighbours states has the same value. It is not possible to determine the best direction. In such situation make a big jump to some directions & try to get to new section of search space.



(c). Ridge: It is an area of search space that is higher than surrounding areas but that cannot be traversed by single moves in any one direction. It is a special kind of local maxima. Here apply two more rules before doing the test ie moving in several direction at once.

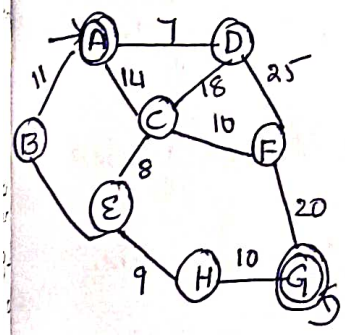


Beam Search:

- Optimized Version of **Best First Search**
- Heuristic search algorithm
- Explores a graph by expanding the most promising node in a limited set.
- Reduces memory requirement
- Greedy algorithm.
- Only predetermined no of best partial solutions are kept as candidates */

Given Beam Value (B) = Beam width (β)

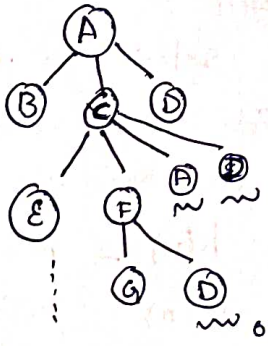
predetermined no of best partial soln are kept as candidates



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

Best first Search — uses sorting priority ~~A~~ C ~~B~~ D



C to G = 25
Choose C

C children E, F, A, E

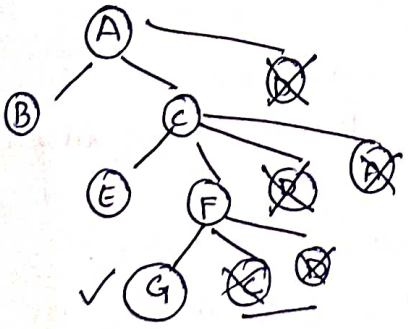
~~A~~ already visited

E → G = 17
choose E

Beam Search

predetermined value of candidates are considered.

Consider (B = 2)



B − G = 32 ; D → G = 35, C → G = 25

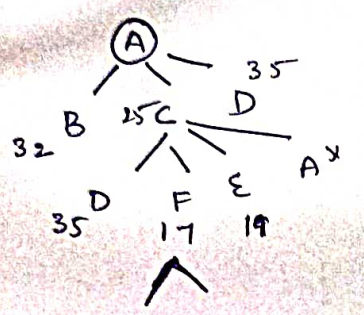
in B & C, E is better.

D, A already visited ∴ optimal

E → G − 19
F → G − 17 ✓

C, D already visited

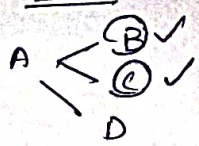
It is similar to Best first search but we search only using beam value



~~X X~~ B D F E

but in beam search

$\beta = 2$



as $\beta = 2$

You can consider best 2 states.

With this you can minimize

if

Beam search $O(b^d)$

Branching factor.

Only concentrates on space complexity.

* Best First Search:

- It is expanding the best partial path from current node to goal node.
- Greedy Search
- It always selects the path which appears best at that moment.
- Combination of both BFS & DFS.
- It uses the heuristic function $h(n) <= h^*(n)$ & searches.

  $h(n)$ = heuristic cost

  $h^*(n)$ = estimated cost
- The greedy best first algorithm is implemented by priority queue.

° Algorithm steps: (in general)

1. Place starting node into open list.
2. if the open list is empty, stop & return failure.
3. Remove the node n, from open list which has lowest value of h(n), & p it in the closed list.
4. Expand node n, & generate the successors of node n.
5. Check each successor of node n & find whether the node is a node | not. If any successor node is goal node, then return su & terminate the search, else return to further step.
6. for each successor node, algorithm checks for evaluation function f & then check if the node has not been in both list, then add it to list
7. Return to step 2.

Advantages:

Best first search can switch between BFS & DFS by gaining the advantages of both the algorithms.

More efficient

Disadvantages:

It can behave as an unguided depth first search In the worst case scenario

It can get stuck in a loop as DFS

It is not optimal.

Uses evaluation algorithm (function) to decide whether which adjacent node is most promising & then explore.

priority queue is used to store cost of nodes.

Algorithm

priority Queues 'PQ' (sorted order) containing initial states

Loop

if PQ = empty Return FAIL

Else

NODE ← Remove-first(PQ)

if NODE = GOAL

Return Path from initial to NODE

Else

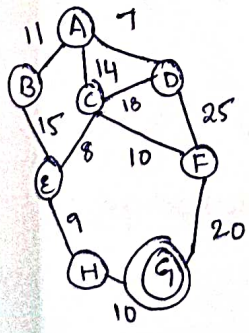
Generate all successors of NODE

& insert newly generated NODE

into PQ according to cost value.

END Loop.

Example:



Given Straight line distance

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

Best-first search

open	closed
[A]	[ ]
[C, B, D] (Successor nodes)	[A]
[B, D]	[A, C]
[F, E, B, D] (Child nodes)	[A, C]
[E, B, D]	[A, C, F]
[G, E, B, D] (Goal node)	[A, C, F]
[E, B, D]	[A, C, F, G]

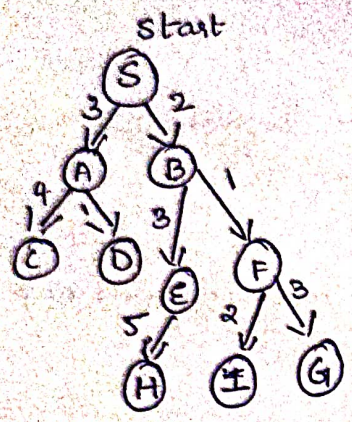
∴ Best path from A to G is 14+10+20 = 44

[A, C, F, G]

Space Complexity $O(b^d)$

b is branching factor
d is depth.

- Example 2



node(n)	H(n)
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

2 + 1 + 3

6

* A* Algorithm: (A score algorithm)

- Uses heuristic function h(n) & cost to reach the node 'n' from st state g(n)
- Finds shortest path through search space. 'Ay star' by Hart in 197
- Fast & optimal result.

$$f(n) = g(n) + h(n)$$

f(n): total estimated cost; g(n): cost to reach the node; h(n): heuristic value

f scores are generated from graph

- Brand & Bound + Best search
- has dynamic programming principles
- uses queue

- Algorithm:

(i) Enter starting node in OPEN list.
(ii) If OPEN list is empty return FAIL
(iii) Select node from OPEN list which has smallest value of (g+h).
↳ if node = Goal, return success
(iv) Expand node 'n' & generate all successors.
↳ compute gen g+h for each successor node
(v) if node 'n' is already OPEN/CLOSED, attach to back pointer
(vi) Goto (ii).

- Advantages:

- Best searching algorithm
- Optimal & complete
- Solves complex problems

- Disadvantages:

- Doesn't always produces shortest path
- Some complexity issue
- Requires more memory.

Optimal Solution by A* algorithm:

A* algorithm finds optimal solution if heuristic function is carefully designed & is underestimated.

Underestimation:

if you can guarantee that h never overestimates actual value from current to goal, then A* algorithm ensures to find an optimal path to a goal if one exists.

By underestimating heuristic value, we have to wasted some effort but eventually we can discover goal node

Overestimation:

if we are overestimating heuristic value of each node in a graph/tree, then it is not guaranteed to find the shortest path.

Admissibility of A*:

if start state itself is goal state

A search algorithm is admissible, if for any graph, it always terminates in an optimal path from start state to goal state if path exists. A* algorithm always terminates with the optimal path in case h is an admissible heuristic function.

Monotonic Function:

A heuristic function h is monotonic if (neither increasing nor decreasing)

i) $\forall$ states X_i & X_j such that X_j is successor of X_i.

ii) $h(X_i) - h(X_j) \leq Cost(X_i, X_j)$ ie actual cost is going from X_i to X_j

iii) $h(Goal) = 0$

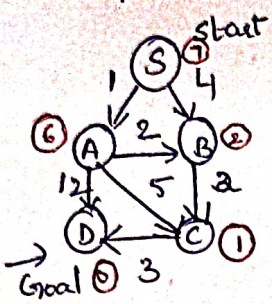
Here heuristic is locally admissible, ie consistently finds the minimal path to each state they encounter in search

Monotonic property is search space which is everywhere locally consistent with heuristic function employed ie reaching each state with shortest path from its ancestors.

A cost function f is monotone if $f(N) \leq f(succ(N))$

for any admissible cost function f, we can construct a monotonic admissible function

• Example :



$f(n) = g(n) + h(n)$ (child)

$S \to A = 1 + 6 = 7$

$S \to B = 4 + 2 = 6$ (child (n))

$S \to B \to C = \underset{(g)}{4+2} + 1 = 7$

$S \to B = 4$, $B \to C = 2$ | c is child, $h = 1$

$S \to B \to C \to D = \underset{(S \to B)\ (B \to C)\ (C \to D)}{4 + 2 + 3} + \underset{(D\ child)\ h=0}{0} = 9$

(g)

$\boxed{S \to B \to C \to D = 9}$ Route 1

To check optimality let us check another path ie 'A'

$S \to A \to B = \frac{1+2}{g} + \frac{2}{h} = 5$

$S \to A \to C = \frac{1+5}{g} + \frac{1}{h} = 7$

$S \to A \to D = \frac{1+12}{g+h} = 13$ $\therefore \boxed{S \to A \to D = 13}$ Route 2

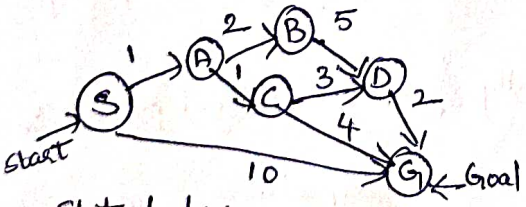
$S \to A \to B \to C = \frac{1+2+2}{g} + \frac{1}{h} = 6$

$S \to A \to B \to C \to D = \frac{1+2+2+3}{g} + \frac{0}{h} = 8$

$\therefore \boxed{S \to A \to B \to C \to D = 8}$ Route 3

$S \to A \to C \to D = \frac{1+5+3}{g} + \frac{0}{h} = 9$ $\therefore \boxed{S \to A \to C \to D = 9}$ Route 4

So the shortest route is $\boxed{R_3 = S \to A \to B \to C \to D = 8}$



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

$\therefore \boxed{R_2 = S \to A \to C \to G = 6}$ is optimal cost.

$S \to A = 1 + 3 = 4$

$\boxed{S \to G = 10 + 0 = 10}$ $\boxed{S \to G = 10}$ R_1

$S \to A \to B = 1 + 2 + 4 = 7$

$S \to A \to C = 1 + 1 + 2 = 4$ ✓

$S \to A \to C \to D = 1 + 1 + 3 + 6 = 11$

$S \to A \to C \to G = 1 + 1 + 4 + 0 = 6$

$\boxed{S \to A \to C \to G = 6}$ R_2

$S \to A \to B \to D = 1 + 2 + 5 + 6 = 14$

$S \to A \to B \to D \to G = 1 + 1 + 3 + 2 + 0 = 7$

$\boxed{S \to A \to B \to D \to G = 7}$ R_3

$S \to A \to B \to D \to G = 1 + 2 + 5 + 2 + 0 = 10$

$\boxed{S \to A \to B \to D \to G = 10}$ R_4

Iterative Deepening A*:

It is a combination of depth first iterative deepening & A* algorithm (IDA*).

Successive iterations are corresponding to increasing values of the total cost of a path rather than increasing depth of a search.

Algorithm:

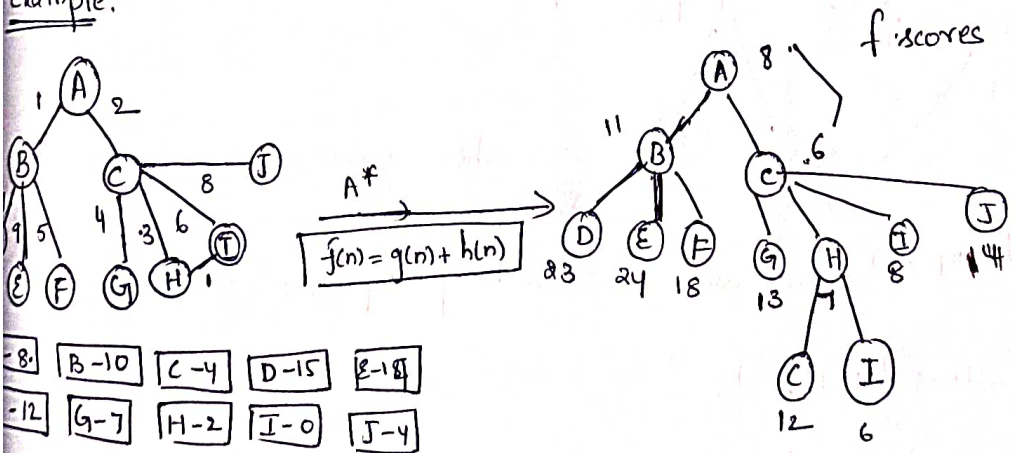
For each iteration, perform a DFS pruning off a branch when it its total cost (g+h) exceeds a given threshold.

The initial threshold starts at the estimate cost of start state & increases for each iteration of algorithm

The threshold used for the next iteration is the minimum cost of all values exceeded the current threshold.

These steps are repeated until we find a goal node.

Example:



Step-1: Consider least value a threshold

Threshold = 6

→ (A) 8 (8 > 6)

f-value (8)

Step-2:

Threshold = 7

→ (A) 8 (8 > 7

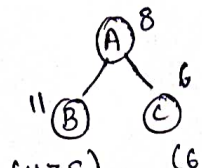
f-value (8)

Step3:

Threshold = 8

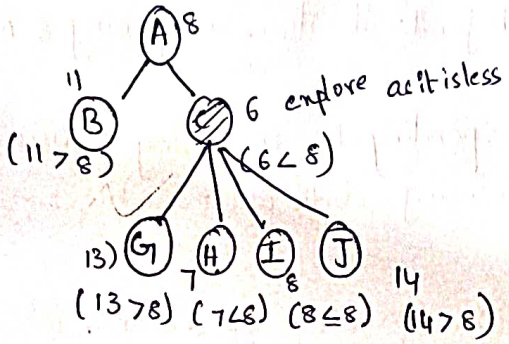
(A) 8 (8 = 8)

then expand

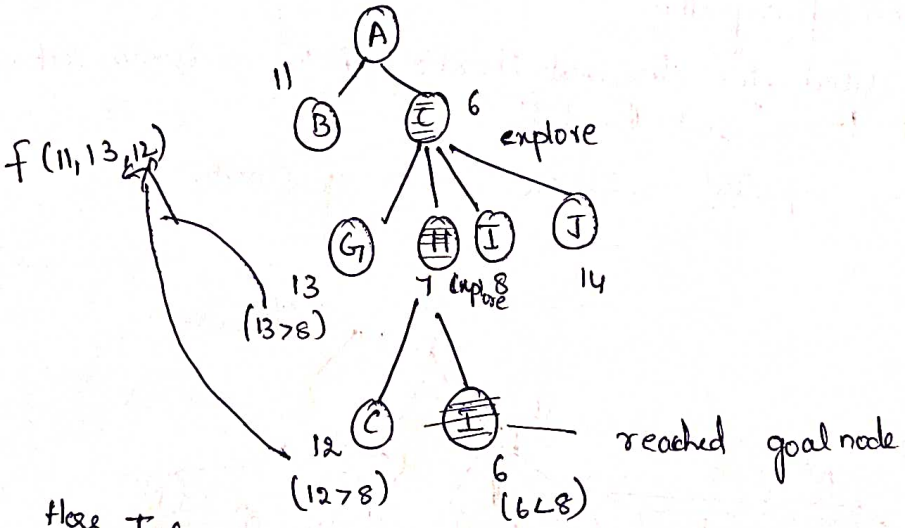


f values (11)

Values of B / nodes of B



F values (11, 13, Now explore depth wise.



Here F (14 > 8) but we are considering depth first

So f values (11, 13, 12)

Here we need not to store all values as in A* to reach goal node but values of threshold is calculated Here (11, 13, 12) is required to reach path A → C → H → I

- Advantages:
  - optimal path.
  - Both advantages of DFID & A*
- Disadvantages:
  - suffers from too little memory.

Constraint Satisfaction:

Many AI problems can be viewed as problems of constraint satisfaction in which the goal is to solve some problem state that satisfies a given set of constraints instead of finding optimal path to solution.
Such problems are called Constraint Satisfaction (CS) problems.

Search can be made easier in those cases in which the solution is required to satisfy local consistency conditions. Some examples are

A cryptography problem: A number puzzle problem in which a group of arithmetical operations has some/all of its digits replaced by letters & the original digits must be found. In such a puzzle each letter represents a unique digit.

N-Queen problem: The condition is that on the same row/column/diagonal, no two of the queens attack each other.

A map colouring problem: Given a map, color regions of map using three colours, blue, red, & black such that no two neighbouring countries have the same colour.

In general we can define constraint satisfaction problem as follows:

- set of variables $\{x_1, x_2, x_3 \cdots, x_n\}$ with each $x_i \in D_i$ with possible values and
- set of constraints, ie relations, that are assumed to hold between the values of variables.

The problem is to find, for each $i, 1 \leq i \leq n$, a value of $x_i \in D_i$, so that all constraints are satisfied. A CS problem is usually represented as an undirected graph, called Constraint Graph in which the nodes are the variables & the edges are the binary constraints. We can easily see that a CSP can be given an incremental formulation as standard search problem.

Start state: the empty assignment, ie all variables are unassigned.
Goal state: all the variables are assigned values while satisfy constraints.
Operator: assigns values to any unassigned variable, provided that it does not conflict with previously assigned variable.

Every solution must be a complete assignment & therefore appears at

depth n if there are variables. Further search tree extends only to depth n & hence depth-first search algorithms are most popular for CSPs.

- **Algorithm:**
- Start
- until a complete solution is found (or) all paths have lead to dead ends
{
- select an unexpanded node of the search graph:
- apply the constraint inteface rules to the selected node to generate all possible new constraint.
- if the set of constraint contain a contradiction, then report that this path is dead end.
- if the set of constraint describes a complete solution, then report success.
- if neither a contradiction nor a complete solution has been found, then apply the problem space rules to generate new partial solutions that are consistent with the current set of constraint. Insert these partial solutions into the search graphs.

}
- stop

Example:

Crypt Arithmetic puzzle: