CHAPTER THREE

Searching and Solving Problems

1. PROBLEM SOLVING BY SEARCHING

- Simple reflex agents are limited in what they can do
- Because, their actions are determined only by the current percept
- Furthermore, they have no knowledge of what their actions do or of what they are trying to achieve
- They can't work well in environments
  - which this mapping would be too large to store and
  - would take too long to learn
- Hence, goal-based agent is used

CONT…

- Goal-based agents can succeed by considering future actions and the desirability of their outcomes
- Goal-based agents that use more advanced factored or structured representations are usually called planning agents
- **Uninformed** and **informed** search algorithms are used to create solution for problems

2. PROBLEM-SOLVING AGENTS

- Problem-solving agent is A kind of goal-based agent, It solves problem by finding sequences of actions that lead to desirable states (goals)
- In Problem Solving Agent, there are four general steps:

❑Goal Formulation

- what are the successful world state

❑Problem Formulation

- what actions and states to consider give the goal

❑Search

- determine the possible sequence of actions that lead to the state of known values and the choosing the best sequence

❑Execute

- Give the solution preform the actions

GOAL FORMULATION

- To solve a problem, the first step is the ***goal formulation***, based on the current situation
- The goal is formulated as a set of world states, in which the goal is satisfied
- Goal-objectives that the agent is trying to achieve and, hence the actions it needs to consider.
- Reaching from initial state to à goal state
  - Actions are required
- *Actions* are the operators
  - causing transitions between world states
  - **Actions** should be abstract enough at a certain degree, instead of very detailed

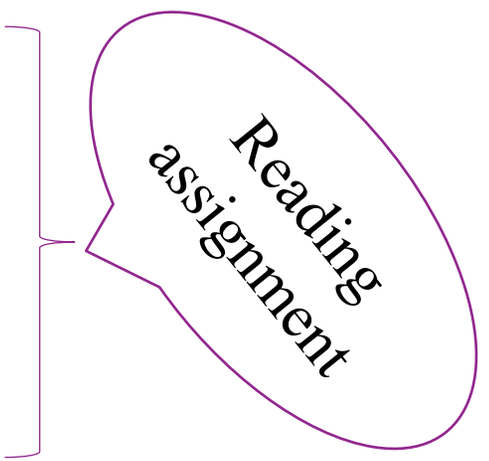
E.g., **turn left** VS **turn left 30 degree**, etc.

3. PROBLEM FORMULATION

- ***Problem formulation:*** is the process of deciding what actions and states to consider, and follows goal formulation
- An agent with several immediate options of unknown value can decide what to do by first examining ;
- different possible sequences of actions that lead to states of known value, and then choosing the best one
- This process of looking for such a sequence is called **search**
- A search algorithm takes a **problem** as input and returns a **solution** in the form of an action sequence

CONT…

- Once a solution is found, the actions it recommends can be carried out.
- This is called the **execution** phase
- Thus, we have a simple "formulate, search, execute" design for the agent
- There are four essentially different types of problems:
  - Single state problems,
  - Multiple-state problems,
  - Contingency problems, and
  - Exploration problems



CONT…

- **Well-defined problems and solutions**
  - A problem is really a collection of information that the agent will use to decide what to do
  - A problem is defined by 5 components:
    - *Initial state*
    - *Actions*
    - *Transition model or (Successor functions)*
    - *Goal Test*
    - *Path Cost*

- **State**: Description of the state of the world in which the search agent finds itself.
- **Starting / initial state**: The initial state in which the search agent is started.
- **Goal state**: If the agent reaches a goal state, then it terminates and outputs a solution (if desired).
- **Actions**: All of the agents allowed actions.
- **Solution**: The path in the search tree from the starting state to the goal state.
- **Cost function**: Assigns a cost value to every action. Necessary for finding a cost-optimal solution.
- **State space**: Set of all states.
- A ***path*** in the state space- any sequence of states connected by a sequence of actions.

CONT…

- The initial state is that the agent knows itself to be in.
- The set of possible actions available to the agent.
- The term operator is used to denote the description of an action in terms of which state will be reached by carrying out the action in a particular state
- (An alternate formulation uses a successor function S.
- Together, these define the state space of the problem: the set of all states reachable from the initial state by any sequence of actions.

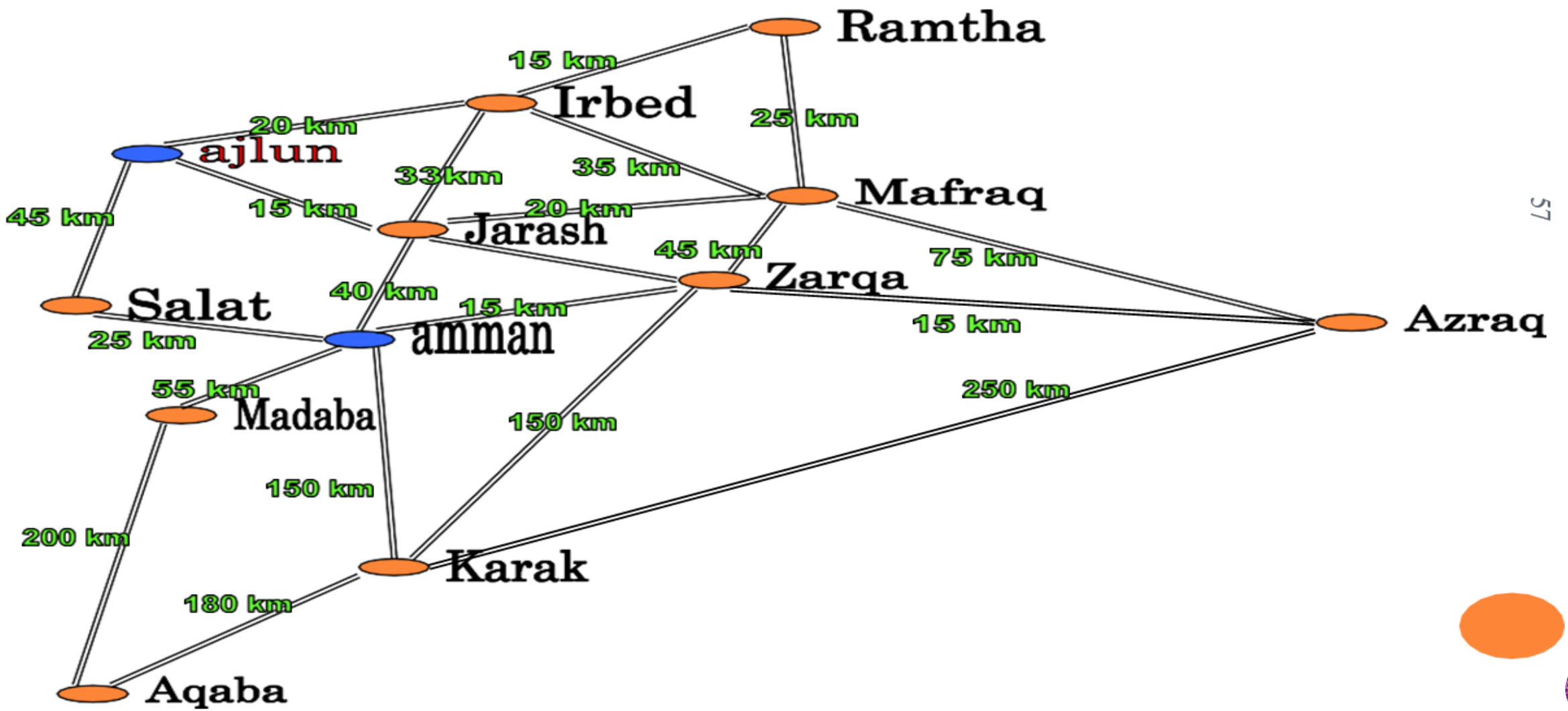
CONT…

- The ***goal test*** applied to the current state to test if the agent is in its goal
  - Sometimes there is an explicit set of possible goal states
  - Sometimes the goal is described by the properties instead of stating explicitly the set of states
    - ✓Example: Chess
      - The agent wins if it can capture the KING of the opponent on next move (checkmate).
      - No matter what the opponent does

CONT…

- A **path cost** function is a function that assigns a cost to a path.
- The cost of a path is the sum of the costs of the individual actions along the path.
- The path cost function is often denoted by g
- The *solution* of a problem is then *a path from the initial state to a state satisfying the goal test*
- to distinguish the best path from others
- ***Optimal* solution** is the solution with **lowest path cost** among all solutions

- E.g:


Ramtha
15 km
Irbed
20 km
ajlun
25 km
35 km
33km
Mafraq
45 km
15 km
20 km
Jarash
45 km
75 km
Zarqa
Salat
40 km
15 km
Azraq
25 km
amman
15 km
55 km
250 km
Madaba
150 km
150 km
200 km
Karak
180 km
Aqaba

1. Problem : To Go from Ajlun to Amman
2. Initial State : Ajlun
3. Operator / Action : Go from One City To another .
4. State Space : {Jarash , Salat , irbed, }
5. Goal Test : are the agent in Amman.
6. Path Cost Function : Get The Cost From The Map.
7. Solution :{ {Aj à Ja à Ir à Ma à Za à Am}, {ajàir àma àza àam}

……..{ajàja àam}}

▪ The 8-puzzle

❖Problem formulation

- **State**: the location of each of the eight tiles in one of the nine squares
- **Operators**: blank space moves left, right, up, or down
- **Goal test**: state matches the right figure (goal state)
- **Path cost**: each step costs 1, since we are moving one step each time.

5	4	
6	1	8
7	3	2

1	2	3
8		4
7	6	5

EXERCISE:1 RIVER CROSSING PUZZLES

1. Goat, Wolf and Cabbage problem

- A farmer returns from the market, where he bought a goata cabbage and a wolf. On the way home he must cross a river. His boat is small and unable to transport more than one of his purchases. He cannot leave the goat alone with the cabbage (because the goat would eat it), nor he can leave the goat alone with the wolf (because the goat wouldbe eaten). How can the farmer get everything safely on the other side?

  1. Identify the set of possible states and operators
  2. Construct the state space of the problem using suitable representation.

2.Towers of Hanoi puzzle

- The goal is to move all the discs from the left peg to the right one. Only one disc may be moved at a time. A disc can be placed either on an empty peg or on top of a larger disc. Try to move all the discs using the smallest number of moves possible.

Search Strategies

3. Search

- Because there are many ways to achieve the same goal
  - Those ways are together expressed as a tree
  - Multiple options of unknown value at a point,
    - the agent can examine different possible sequences of actions, and choose the best
  - This process of looking for the best sequence is called ***search*.**

- The best sequence is then a list of actions, called ***solution***

Search Algorithm:

- Defined as
  - taking a *problem*
  - and returns a *solution*
- Once a solution is found
  - the agent follows the solution
  - and carries out the list of actions – execution phase
- Design of an agent

- "Formulate, search, execute"

CONT…

- SEARCHING FOR SOLUTIONS

- ***Search*** is the fundamental technique of AI.
- Possible answers, decisions or courses of action are structured into an abstract space, which is search.
- Finding out a solution is done by
  - searching through the state space
- All problems are transformed
  - as a search tree
  - generated by the initial state and successor function

SEARCH TREE

- **Initial state**
  - The root of the search tree is a **search node**
- **Expanding**
  - applying successor function to the current state
  - There by generating a new set of states
- **Leaf nodes**
  - the states having no successors

CONT…

- It is helpful to think of the search process as building up a search tree
- The root of the search tree is a search node corresponding to the initial state
- The leaf nodes of the tree correspond to states that do not have successors in the tree,
  - It is either because they have not been expanded yet, or
  - Because they were expanded, but generated the empty set
- At each step, the search algorithm chooses one leaf node to expand

CONT…

- In search tree, a node is having five components:
  - **STATE**: which state it is in the state space
  - **PARENT-NODE**: from which node it is generated
  - **ACTION**: which action applied to its parent-node to generate it
  - **PATH-COST**: the cost, g(n), from initial state to the node n itself
  - **DEPTH**: number of steps along the path from the initial state

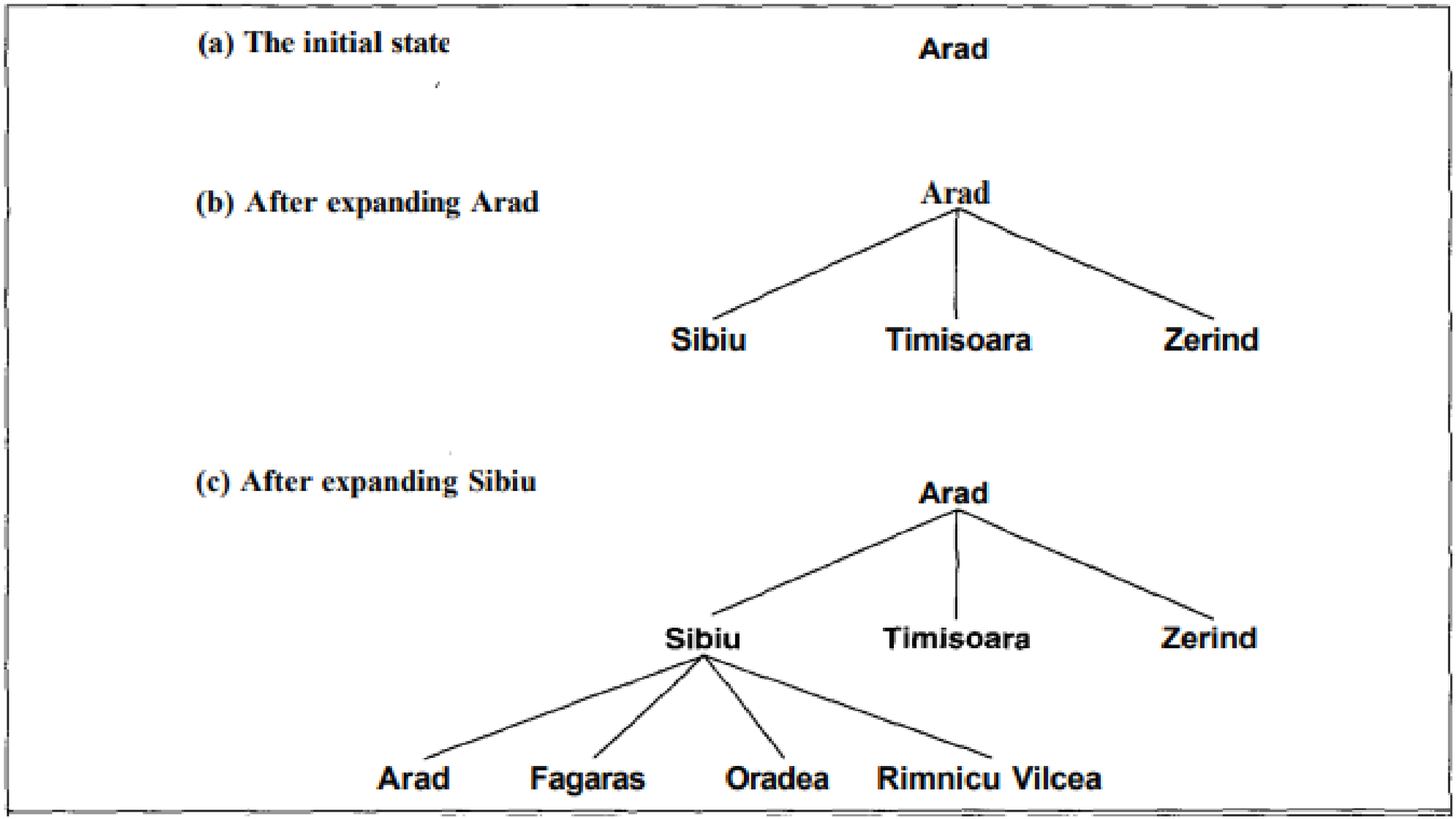
CONT…

- **Measuring problem-solving performance**
  - **Completeness:** is the strategy guaranteed to find a solution when there is one?
  - **Optimality**: does the strategy find the highest-quality solution when there are several different solutions?
  - **Time complexity**: how long does it take to find a solution?
  - **Space complexity**: how much memory is needed to perform the search?

CONT…

- In AI, complexity is expressed in
  - **b**, branching factor, maximum number of successors of any node
  - **d**, the depth of the shallowest goal node.
  - **(**depth of the least-cost solution)
  - **m**, the maximum length of any path in the state space
- Time and Space is measured in
  - number of nodes generated during the search
  - maximum number of nodes stored in memory

CONT…


(a) The initial state
Arad
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea

CONT…

SEARCH METHODS:

- **Uninformed search (blind search)**
  - ➢no information about the number of steps
  - ➢or the path cost from the current state to the goal
  - ➢search the state space **blindly**
- **Informed search, or heuristic search**
  - ➢a cleverer strategy that searches toward the goal,
  - ➢based on the information from the current state so far

CONT…

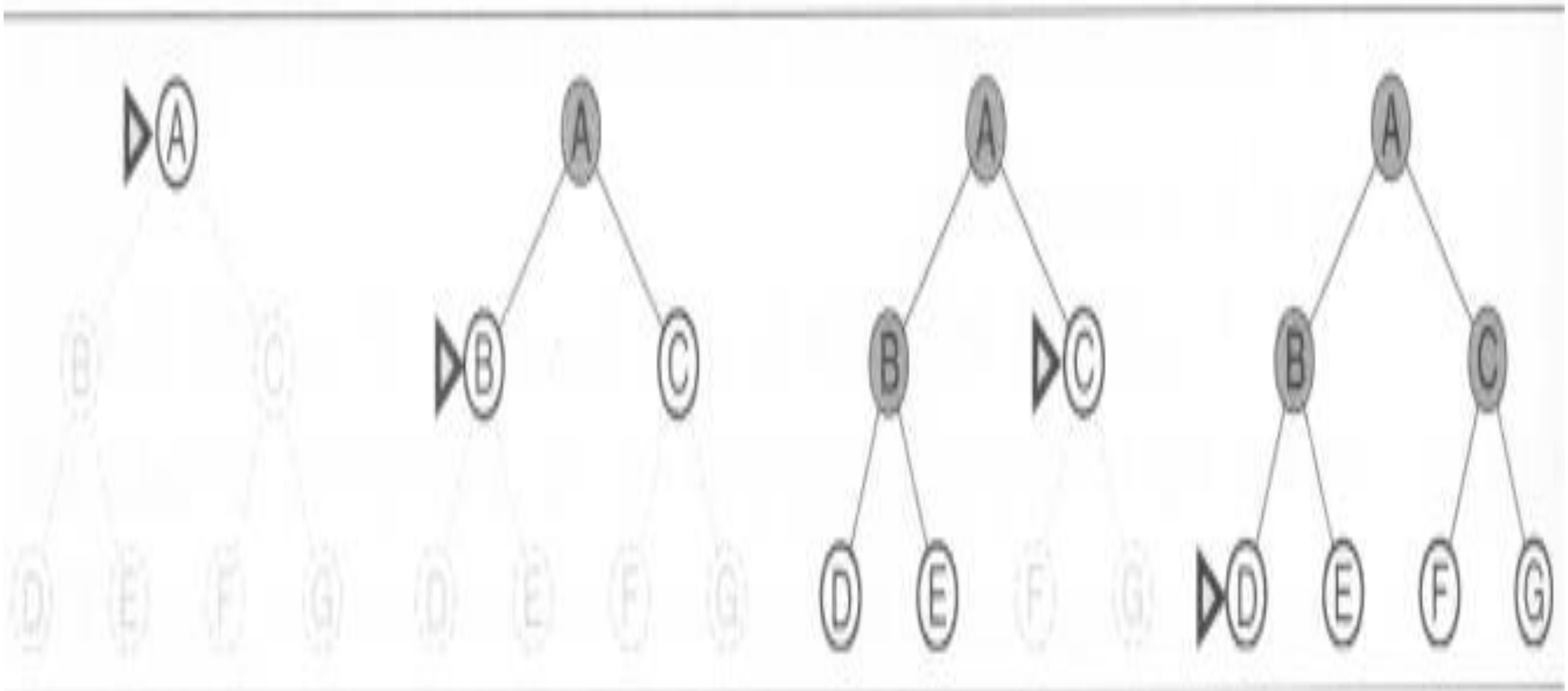
- *Uninformed search*
  - Breadth first search
  - Depth first search
  - Uniform cost search
  - Depth limited search
  - Iterative deepening search
- Informed search
  - Greedy search
  - A*-search

CONT…

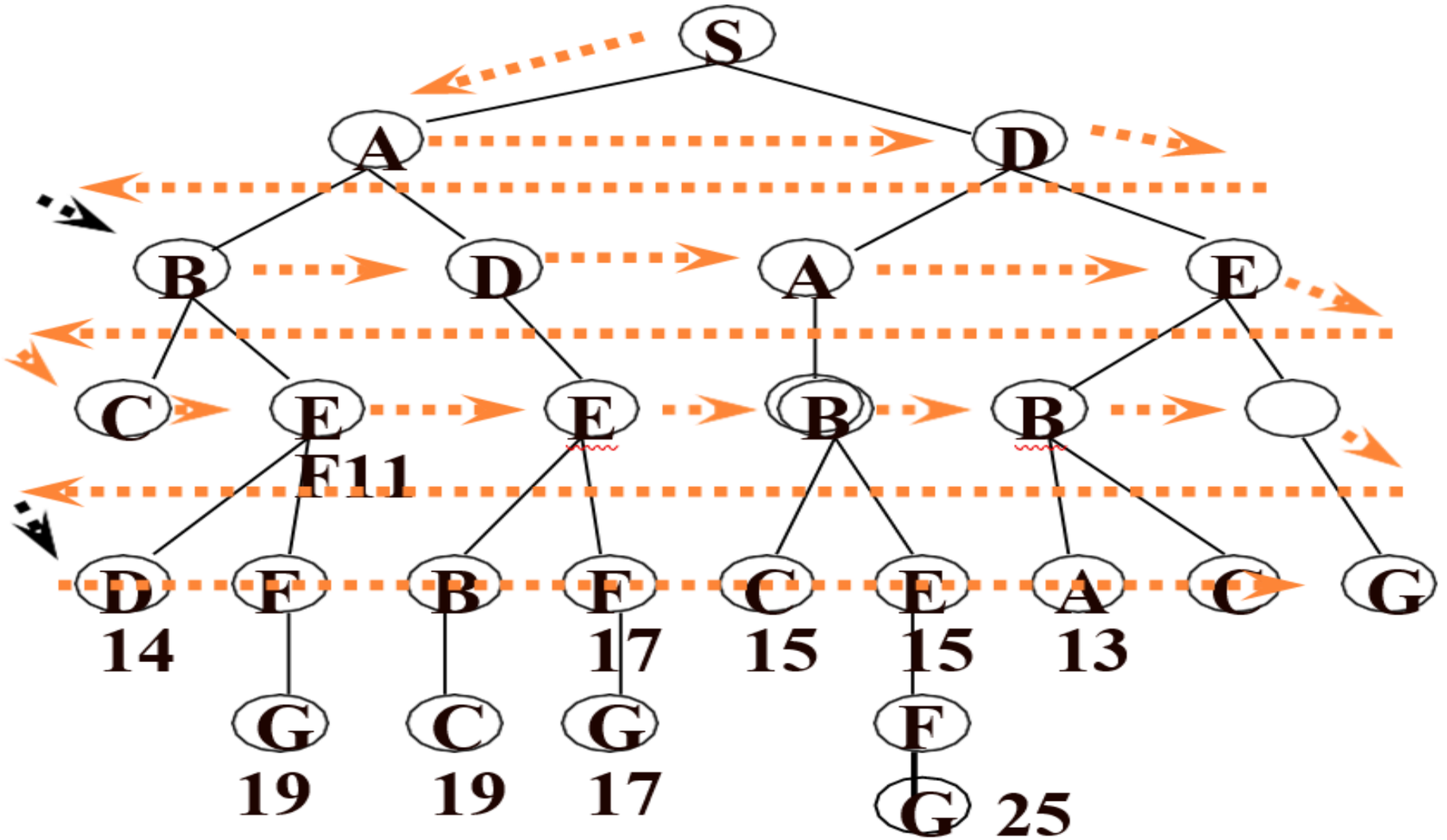
- **Breadth-first search**
  - The root node is expanded first (FIFO)
  - All the nodes generated by the root node are then expanded
  - And then their successors and so on
  - In general, all the nodes at depth **d** in the search tree are expanded before the nodes at depth **d + 1**
  - Can be implemented by GENERAL SEARCH algorithm

```
function BREADTH-FIRST-SEARCH(problem) returns a solution or failure
  return GENERAL-SEARCH(problem, ENQUEUE-AT-END)
```

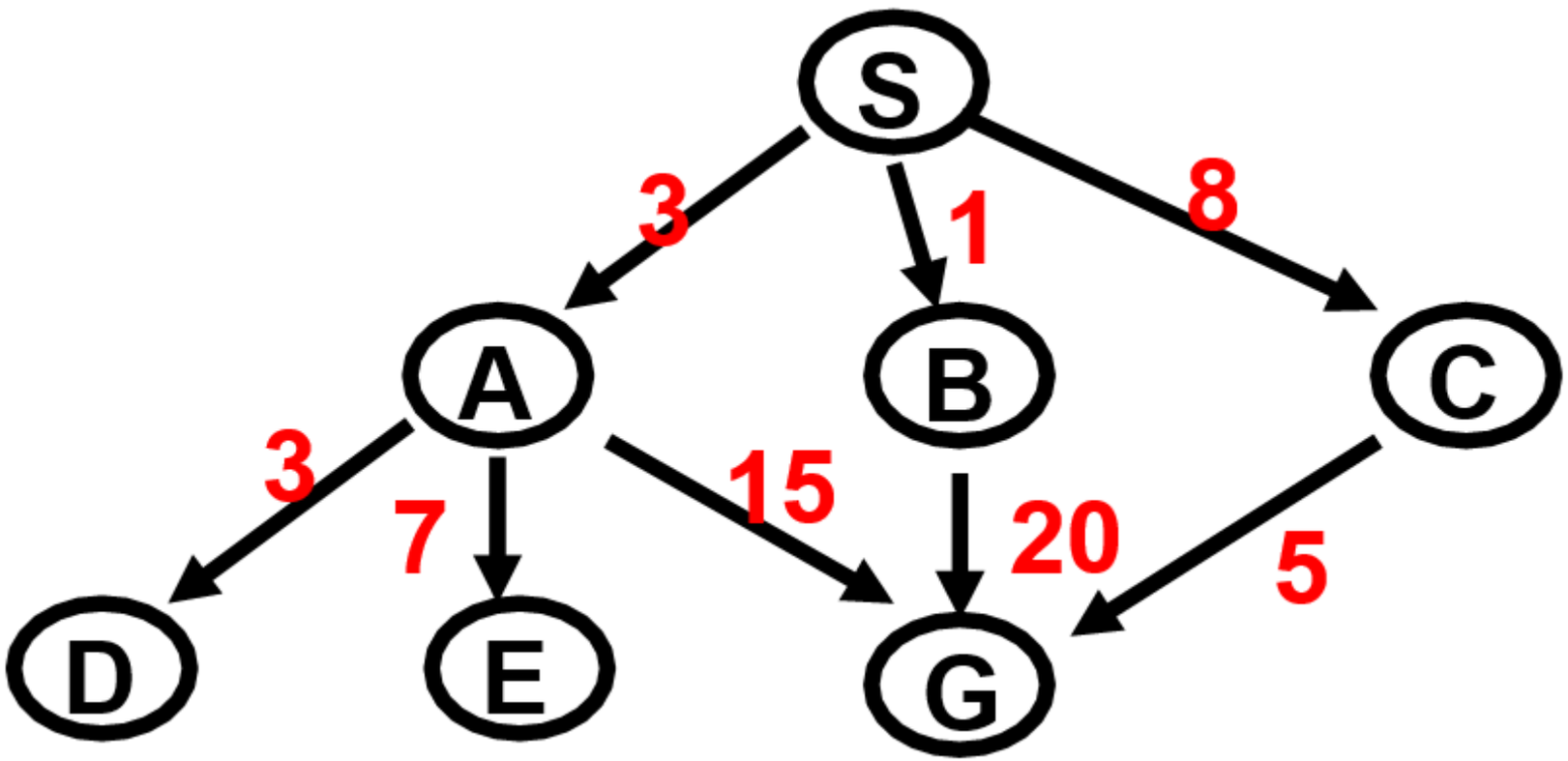
CONT…



CONT…



- **EXAMPLE FOR ILLUSTRATING UNINFORMED SEARCH STRATEGIES of Breadth first search**



CONT…

Breadth first search

Expanded node	Nodes list
	{ S0 }
S0	{ A3 B1 C8 }
A3	{ B1 C8 D6 E10 G18 }
B1	{ C8 D6 E10 G18 G21 }
C8	{ D6 E10 G18 G21 G13 }
D6	{ E10 G18 G21 G13 }
E10	{ G18 G21 G13 }
G18	{ G21 G13 }

- Solution path found is S A G , cost 18
- Number of nodes expanded (including goal node) = 7

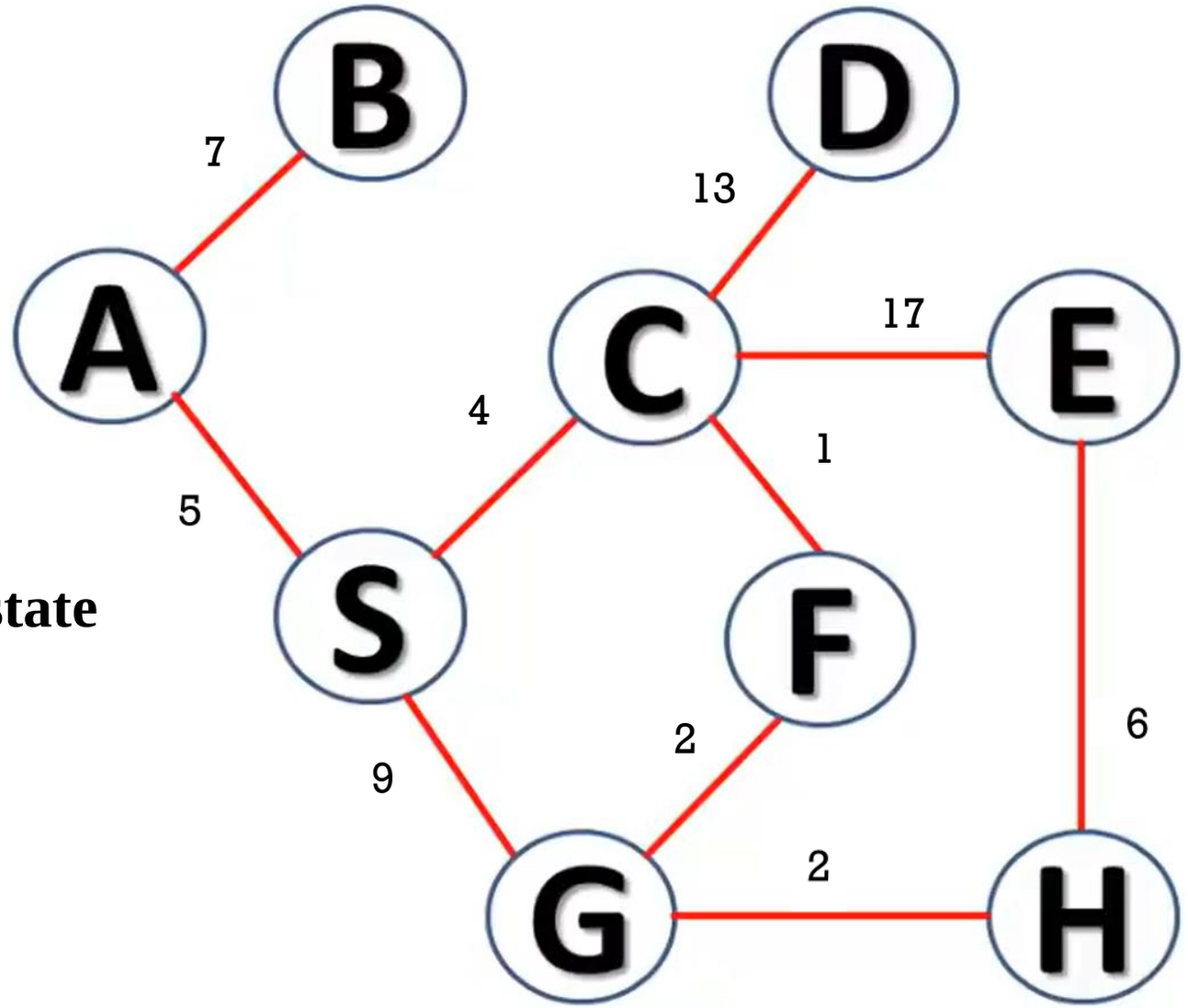
CONT…

- **Breadth first search analysis**
  - ➢Complete – find the solution eventually
  - ➢Optimal, if step cost is 1
  - ➢**The disadvantage**
    - ❖if the branching factor of a node is large, for even small instances (e.g., chess)
      - ✓the ***space complexity*** and the ***time complexity*** are enormous

CONT…

- *Properties of breadth-first search*
  - Complete? Yes (if *b* is finite)
  - Time? $1+b+b^2+b^3+\ldots+b^d = b(b^d-1) = O(b^{d+1})$
  - Space? $O(b^{d+1})$ (keeps every node in memory)
  - Optimal? Yes (if cost = 1 per step)
  - Space is the bigger problem (more than time)

Exercise 1



Initial state

Goal State

UNIFORM COST SEARCH

- **Breadth-first** finds the shallowest goal state
  - but not necessarily be the least-cost solution
  - work only if all step costs are equal
- **Uniform cost search**
  - modifies breadth-first strategy
    - by **always** **expanding the lowest-cost node**
  - The lowest-cost node is measured by the path cost $g(n)$

CONT…

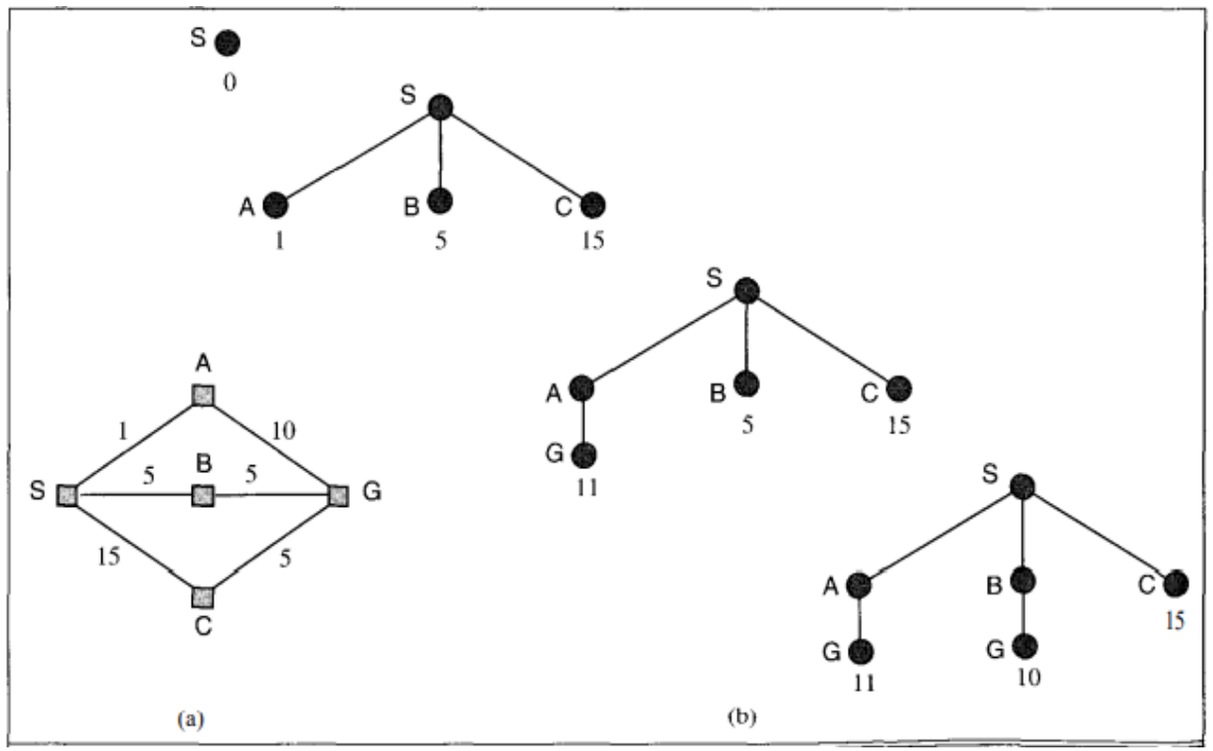
- The first found solution is guaranteed to be the cheapest
  - least in depth
  - But restrict to ***non-decreasing*** path cost

- It is easy to see that breadth-first search is just uniform cost search with g(n) = DEPTH(n)

- When certain conditions are met:
  - the first solution that is found is guaranteed to be the cheapest solution,
  - because if there were a cheaper path that was a solution,
  - it would have been expanded earlier, and thus would have been found first

CONT…

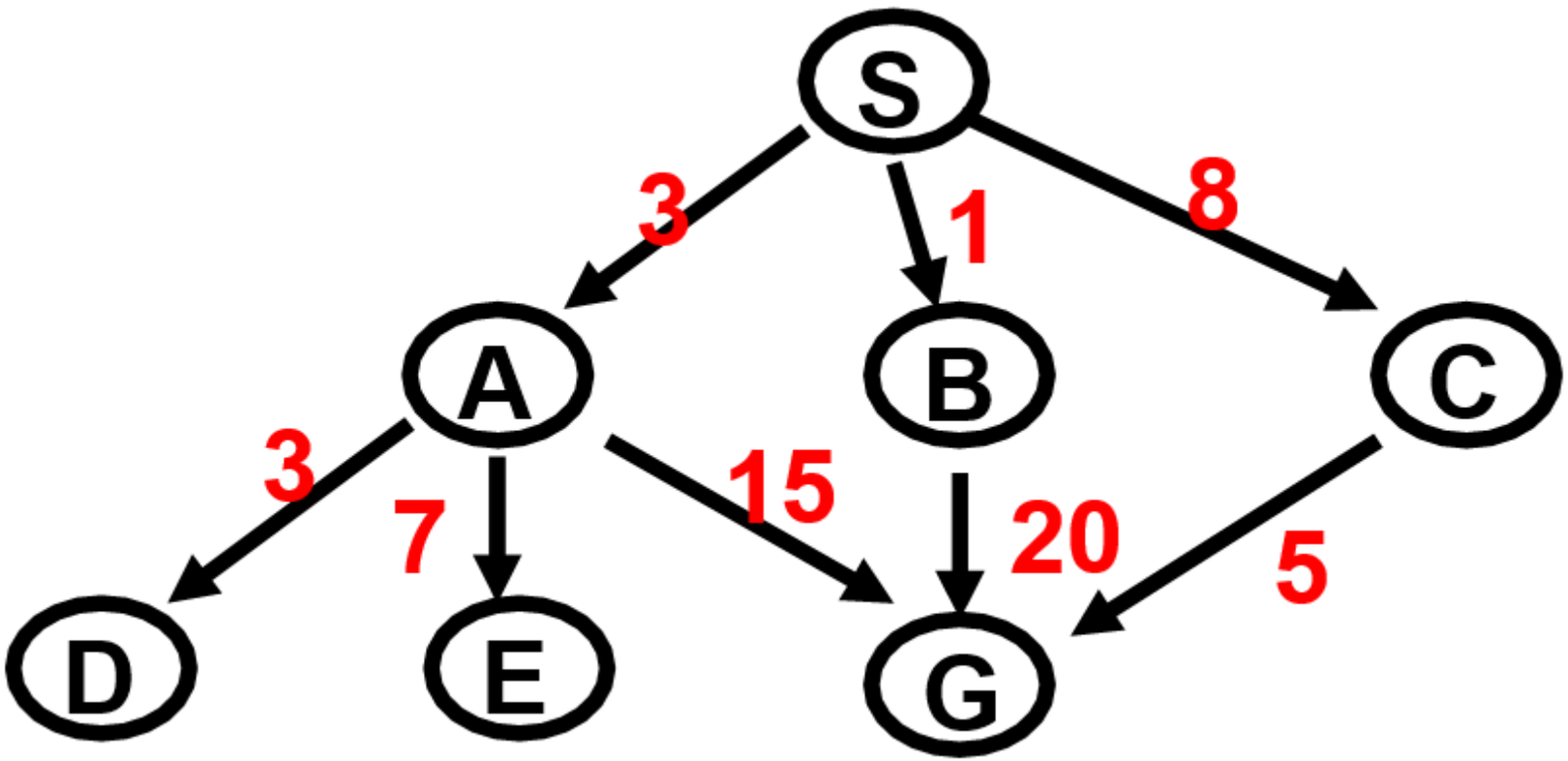
- Consider the following route finding problem



A route-finding problem:
(a) The state space, showing the cost for each operator,

(b) Progression of the search.
Each node is labelled with g(n).
At the next step, the goal node with g = 10 will be selected

- **EXAMPLE FOR ILLUSTRATING UNINFORMED SEARCH STRATEGIES of uniform cost search**



CONT…

- Consider the above example in BFS

UNIFORM-COST SEARCH

Expanded node	Nodes list
	{ S0 }
S0	{ B1 A3 C8 }
B1	{ A3 C8 G21 }
A3	{ D6 C8 E10 G18 G21 }
D6	{ C8 E10 G18 G1 }
C8	{ E10 G13 G18 G21 }
E10	{ G13 G18 G21 }
G13	{ G18 G21 }

- Solution path found is S C G, cost =13
- Number of nodes expanded (including goal node) = 7

- Uniform Cost search (ANALYSIS)

- optimal
- complete
- Time and space complexities
  - Reasonable, It takes O(b^L) time and O(b^L) space, where L is the depth limit
- suitable for the problem
  - having a large search space
  - and the depth of the solution is not known

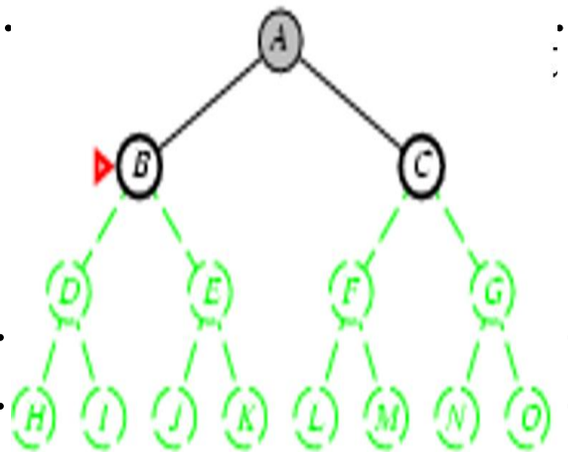
- PROPERTIES OF Uniform depth SEARCH
  - ¢ Complete? Yes
  - ¢ Time? O(b^L)
  - ¢ Space? O(b^L)
  - ¢ Optimal? Yes, if step cost = 1

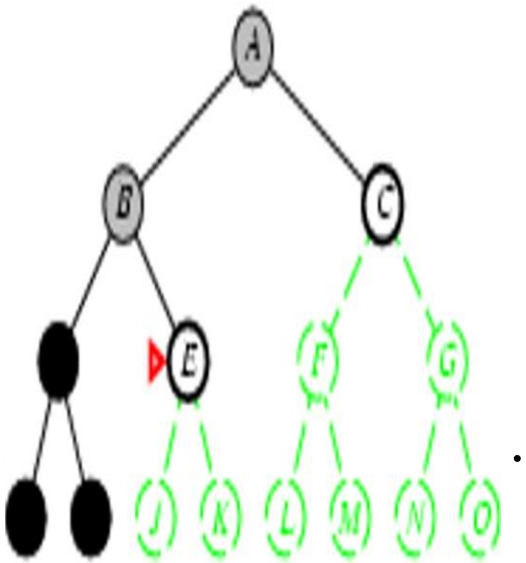
DEPTH FIRST SEARCH

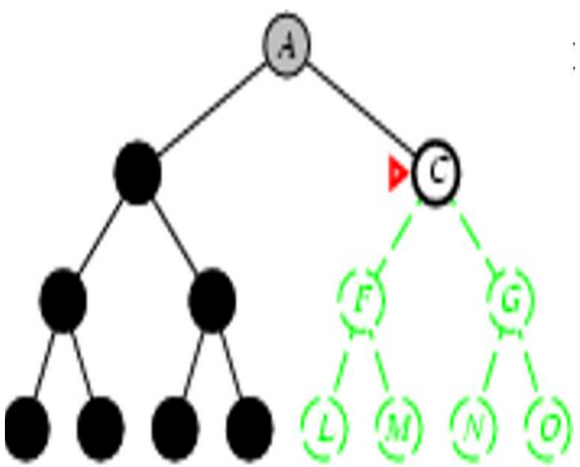
- Always expands one of the nodes at the
  - ***deepest*** level of the tree
- Only when the search hits the end / Dead end
  - goes back and expands nodes at shallower levels
  - Dead end à leaf nodes but not the goal
- Backtracking search
  - only one successor is generated on expansion
  - rather than all successors
  - fewer memory

DEPTH FIRST SEARCH

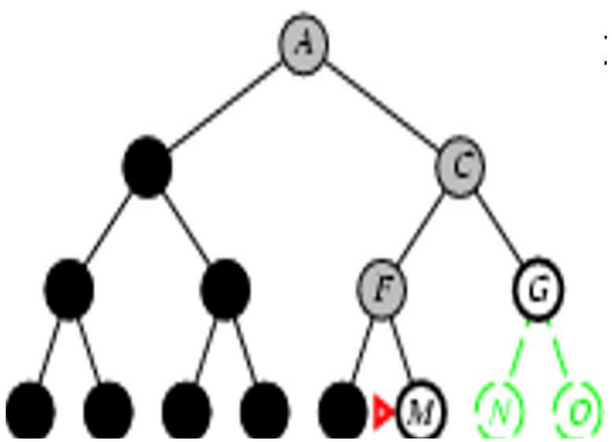
- Expand deepest unexpanded node Look **A-N steps**
- Implementations: A







N step



CONT…

- Consider the above example in BFS
- **DEPTH-FIRST SEARCH**

Expanded node	**Nodes list**
	{ S0 }
S0	{ A3 B1 C8 }
A3	{ D6 E10 G18 B1 C8 }
D6	{ E10 G18 B1 C8 }
E10	{ G18 B1 C8 }
G18	{ B1 C8 }

- Solution path found is S A G, cost 18
- Number of nodes expanded (including goal node) = 5

CONT…

- Not complete
  - because a path may be infinite or looping
  - then the path will never fail and go back try another option
- Not optimal
  - it doesn't guarantee to the best solution
- It overcomes
  - the time and space complexities

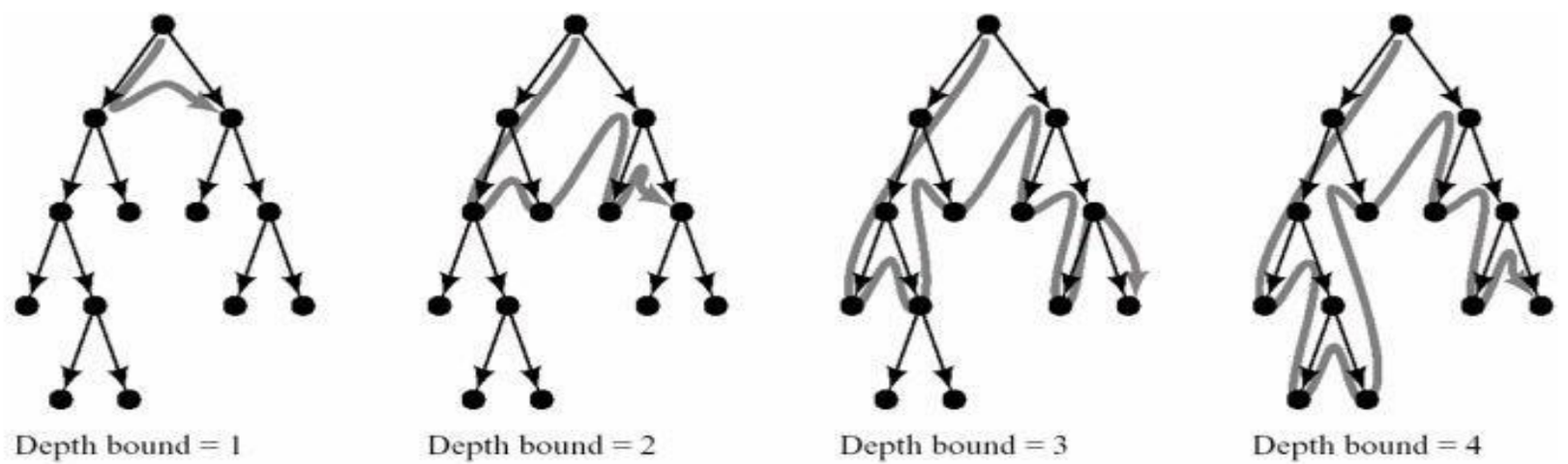
CONT…

PROPERTIES OF DEPTH-FIRST SEARCH

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path à complete in finite spaces
- Time? $O(b^m)$ terrible if *m* is much larger than *d*
  - but if solutions are dense, may be much faster than breadth-first
- Space? *O(bm),* i.e., linear space!
- Optimal? No

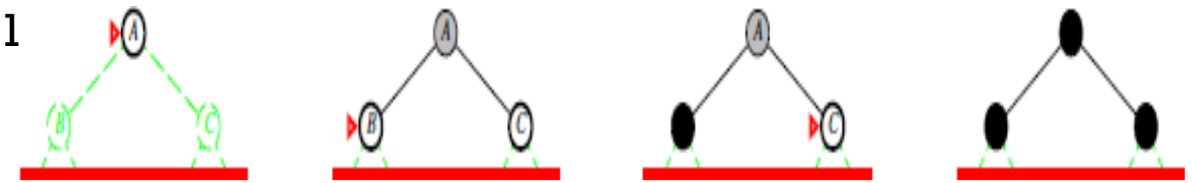
ITERATIVE DEEPENING SEARCH

- Is a modified versions of DFS
- No choosing of the best depth limit
- It tries all possible depth limits:
  - first 0, then 1, 2, and so on
  - combines the benefits of depth-first and breadth-first
  - It will terminate when the depth limits reaches d, depth of shallowest goal node, with success message.

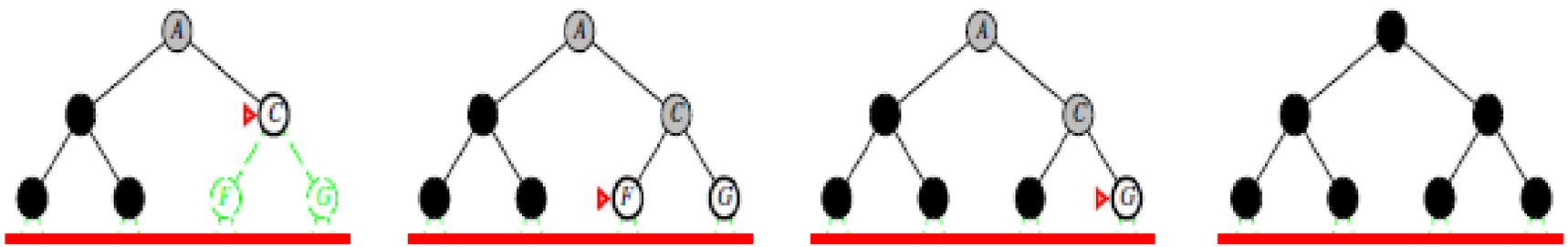


Limit = 0

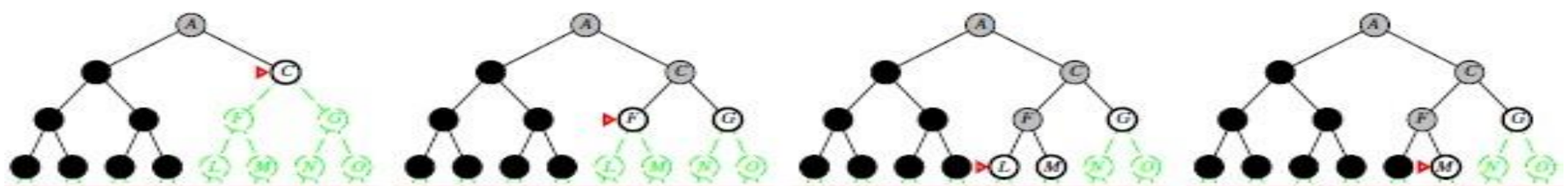
Limit 1



Limit = 2



Limit = 3



- ITERATIVE DEEPENING SEARCH (ANALYSIS)

- Optimal: IDS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.
- Complete: this algorithm is complete is if the branching factor is finite.
- Time and space complexities
  - reasonable
- suitable for the problem
  - having a large search space
  - and the depth of the solution is not known

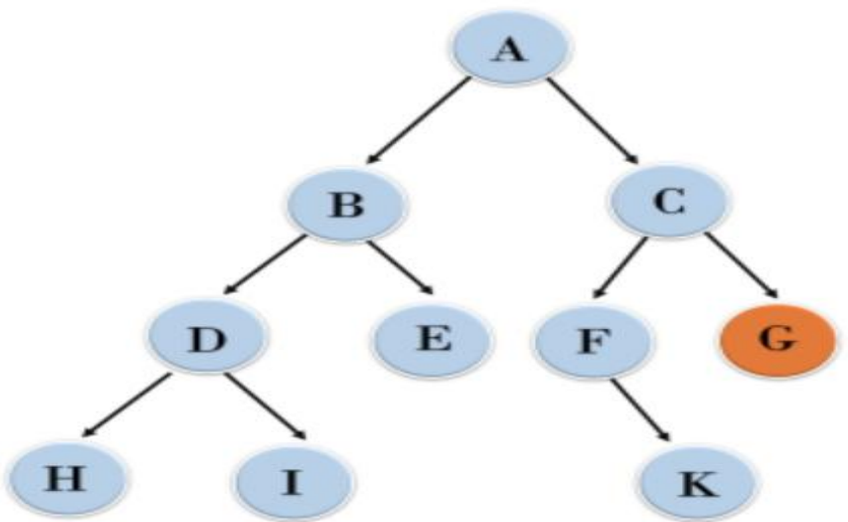
- PROPERTIES OF ITERATIVE DEEPENING SEARCH
  - ¢ Complete? Yes
  - ¢ Time? $(d+1)b0 + d\ b1 + (d-1)b2 + \ldots + bd = O(bd)$
  - ¢ Space? $O(bd)$
  - ¢ Optimal? Yes, if step cost = 1

- **Advantages:**

➢It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

- **Disadvantages:**

➢The main drawback of IDS is that it repeats all the work of the previous phase. Example:



1'st Iteration-----> A
2'nd Iteration----> A, B, C
3'rd Iteration------>A, B, D, E, C, F, G
4'th Iteration------>A, B, D, H, I, E, C, F, K, G

I the third iteration, the algorithm will find the goal

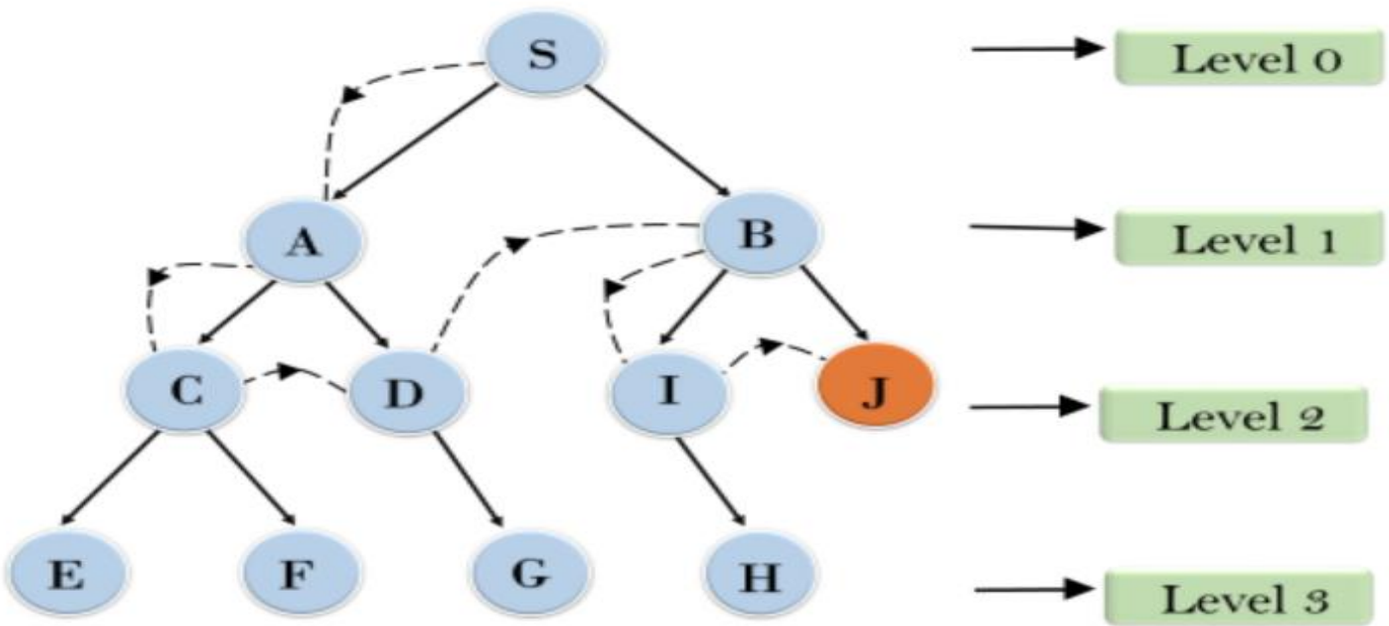
DEPTH LIMITED SEARCH

- Depth-limited search avoids the pitfalls of depth-first search by imposing a cutoff on the maximum depth of a path.
- This cutoff can be implemented with a special depth-limited search algorithm
- We are guaranteed to find the solution if it exists, but we are still not guaranteed to find the shortest solution first.
- **depth-limited** search is **Not complete** and **not optimal.**

CONT…

- If we choose a depth limit that is too small, then depth-limited search is not even complete.
- The time and space complexity of depth-limited search is similar to depth-first search
- It takes O(b^l) time and O(b^l) space, where l is the depth limit.

Example:



- Depth-limited search can be terminated with two Conditions of failure:

➢Standard failure value: It indicates that problem does not have any solution.

➢Cutoff failure value: It defines no solution for the problem within a given depth limit.

- **Advantages:**

➢Depth-limited search is Memory efficient.

- **Disadvantages:**

➢Depth-limited search also has a disadvantage of incompleteness.

➢It may not be optimal if the problem has more than one solution.

- **Completeness:** DLS search algorithm is complete if the solution is above the depth-limit.
- **Time Complexity:** Time complexity of DLS algorithm is **$O(b^{\ell})$**.
- **Space Complexity:** Space complexity of DLS algorithm is **$O(b \times \ell)$**.
- **Optimal:** Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $\ell > d$.

BIDIRECTIONAL SEARCH

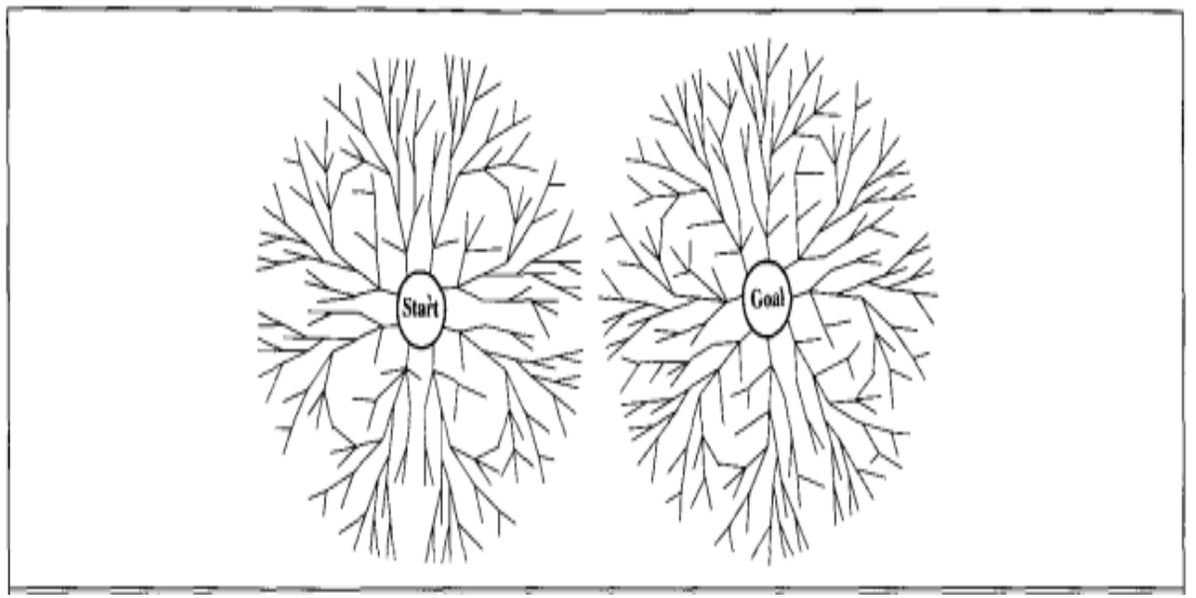
- Run two simultaneous searches:
  - ✓one forward from the initial state another backward from the goal
  - ✓stop when the two searches meet
- For problems where the branching factor is b in both directions:
  - ✓ bidirectional search can make a big difference

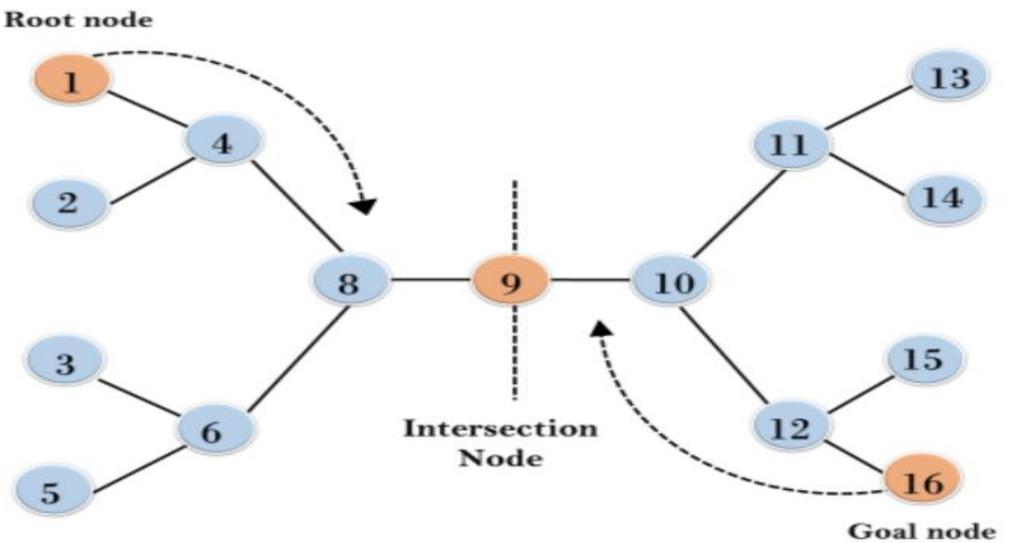
CONT…

- If we assume as usual that there is a solution of depth d:
  - ✓then the solution will be found in $O(2b^{d/2}) = O(b^{d/2})$ steps,
  - ✓because the forward and backward searches each have to go only half way.
- However, computing backward is difficult
- The main question is, what does it mean to search backwards from the goal?
- Searching backward means generating predecessors successively starting from the goal node.
- can the actions be reversible to compute its predecessors?

CONT…

- What can be done if there are many possible goal states?
- We need to decide what kind of search is going to take place in each half
- For example the following figure shows two breadth-first searches





- **Advantages:**

➢Bidirectional search is fast.

➢Bidirectional search requires less memory

- **Disadvantages:**

➢Implementation of the bidirectional search tree is difficult.

➢**In bidirectional search, one should know the goal state in advance.**

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

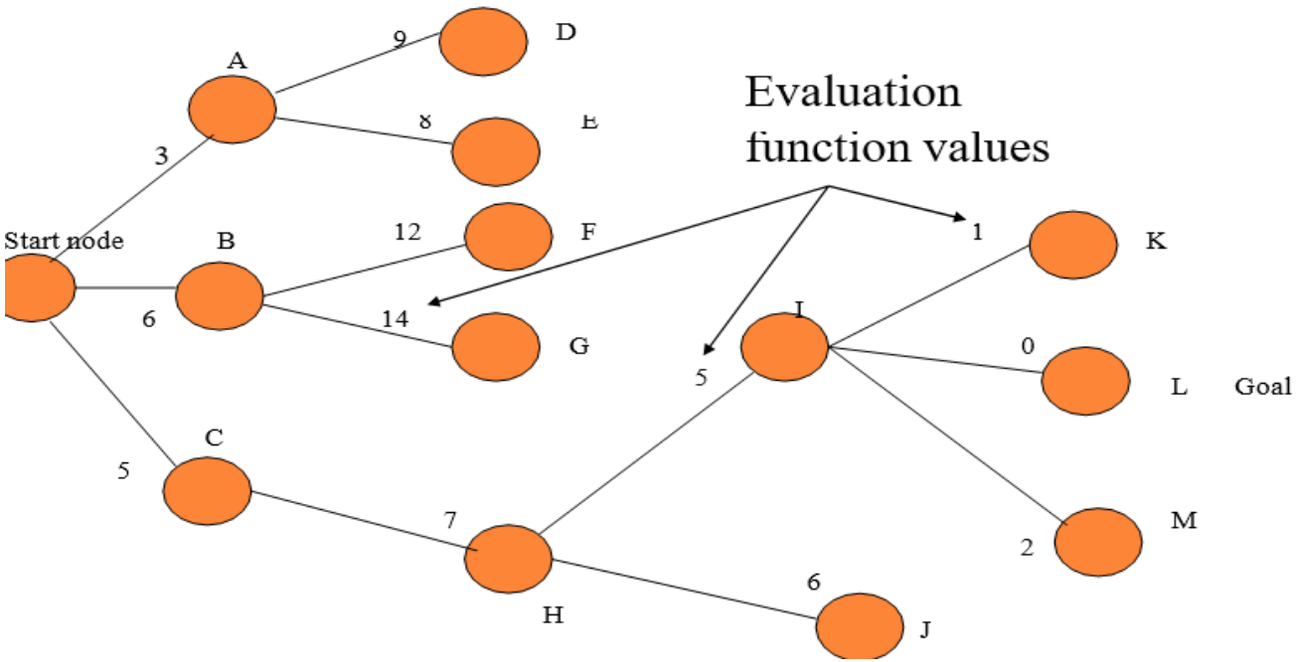
Figure 3.17 Evaluation of search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

INFORMED SEARCH/HEURISTIC SEARCH

- An informed search is more efficient than an uninformed search because in informed search, along with the current state information, some additional information is also present, which make it easy to reach the goal state.
- Blind search – BFS, DFS, uniform cost
  - no notion concept of the "right direction"
  - can only recognize goal once it's achieved
- Heuristic search
- Estimates the cost of the minimal cost path from node ***n*** to a **goal** state
- *h(n)* >= 0 for all nodes *n*
- *h(n)* = 0 Implies that *n* is a goal node
- *h(n)* = ¥ Implies that *n* is a dead end from which a goal cannot be reached.
- All domain knowledge used in the search is *encoded* in the heuristic function *h*.
- *Weak method* :because of the limited way that domain-specific information is used to solve a problem
- Sort nodes in the node list by increasing values of an evaluation function *f(n)* that incorporates domain-specific information

BASET-FIRST-SEARCH

- Idea: use an evaluation function *f(n)* for each node
  - estimate "desirability“(lack of it)
  - Expand most desirable unexpanded node
- *When the nodes are ordered so that the one with the best evaluation expanded first, the result is called **best-first** strategy*



- A sample tree for best-first- search

Table: Search process for Best-First Search

Step	# Node being expanded	Children	Available nodes that are not yet expanded	Node chosen
1	S	(A:3), (B:6), (C:5)	(A:3), (C:5) (B:6)	(A : 3)
2	A	(D:9), (E:8)	(B:6), (C:5), (D:9),(E : 8)	(C:5)
3	C	(H:7)	(B:6), (D:9),(E : 8), H(:7)	(B:6)
4	B	(F:12), (G:14)	(D:9), (E:8), (H:7), (F:12), (G:14)	(H:7)
5	H	(I:5). (J:6)	(D:9), (E:8), (F:12), (G:14), (I:5), (J:6)	(I:5)
6	I	(K:1), (L:0), (M:2)	(D:9), (E:8), (F:12), (G:14), (J:6), (K:1), (L:0), (M:2)	(L,0)
7	Search stops			

GREEDY SEARCH

- A Best-first search that uses *h* to select the *next node* to expand is called *Greedy Search*
- Goal: Minimizes the estimated cost to reach a goal
  - expands the node that seems to be *closest* to a goal
  - utilizes a heuristic function
    - *h(n)* = estimated cost from the *current node* to a *goal*
    - *heuristic functions* are problem-specific
    - often straight-line distance(*hSLD*) for route-finding and similar problems
  - often better than depth-first, although worst-time complexities are equal or worse (space)
- Greedy search is neither *optimal* nor *complete*.

CONT…

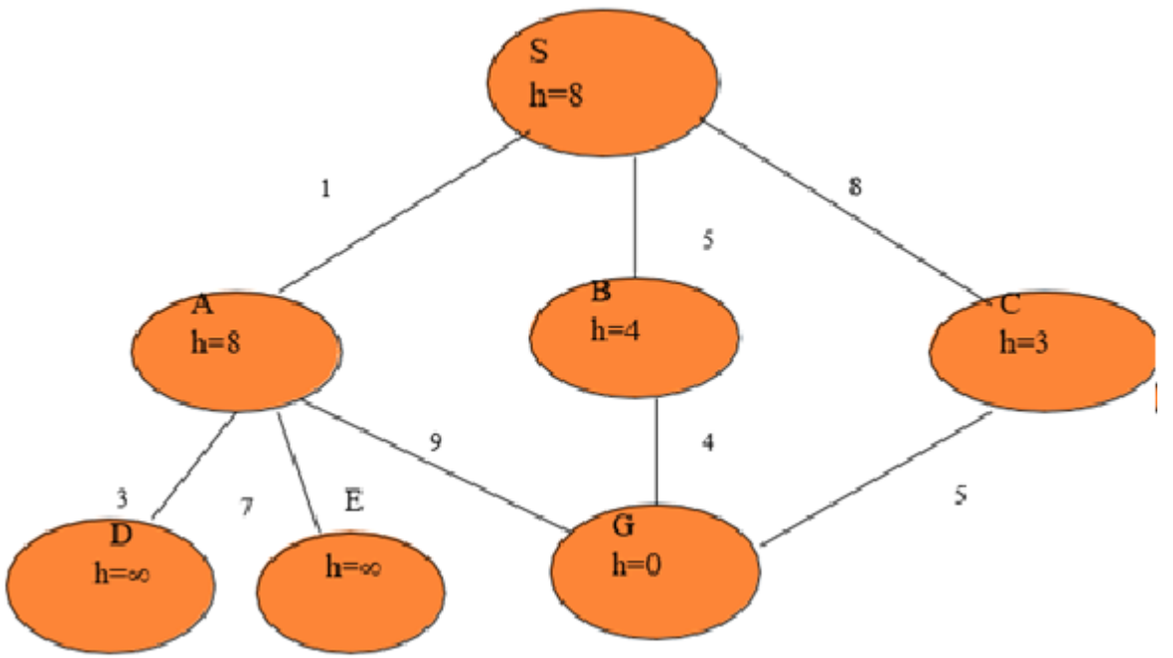
GREEDY SEARCH –EXAMPLE

- Use as an evaluation function *f(n)=h(n)*, sorting nodes in the nodes list by increasing values of *f*.
- Selects the node to expand that is believed to be **closest**

(i.e., **smallest *f* values**) to a goal node.

# of nodes tested 0, expanded 0

Expanded node	**Open list**
	(S:8)

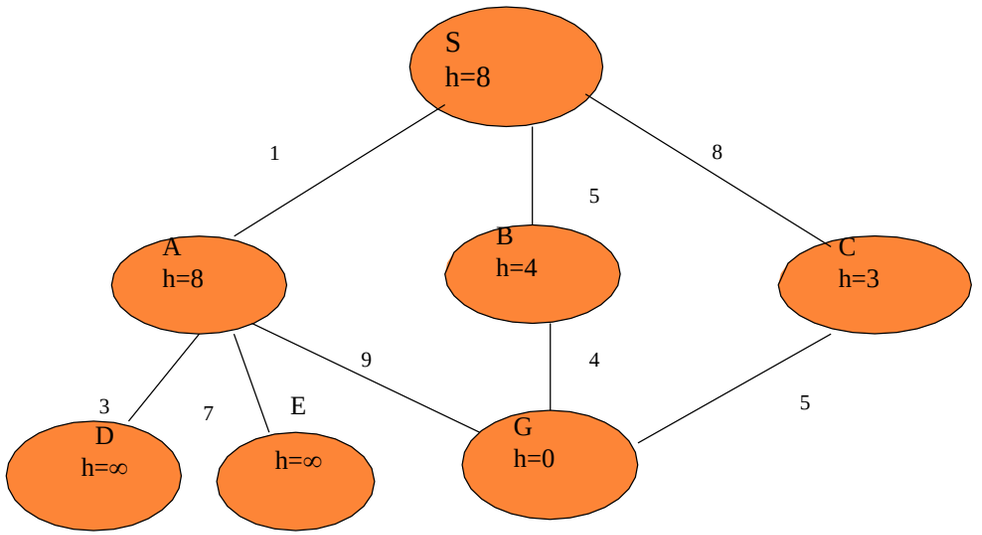


GREEDY SEARCH –EXAMPLE

f(n)= h(n)

# of nodes tested 1, expanded 1

Expanded node	**OPEN list**
	(S:8)
S not goal	(C:3,B:4,A:8)

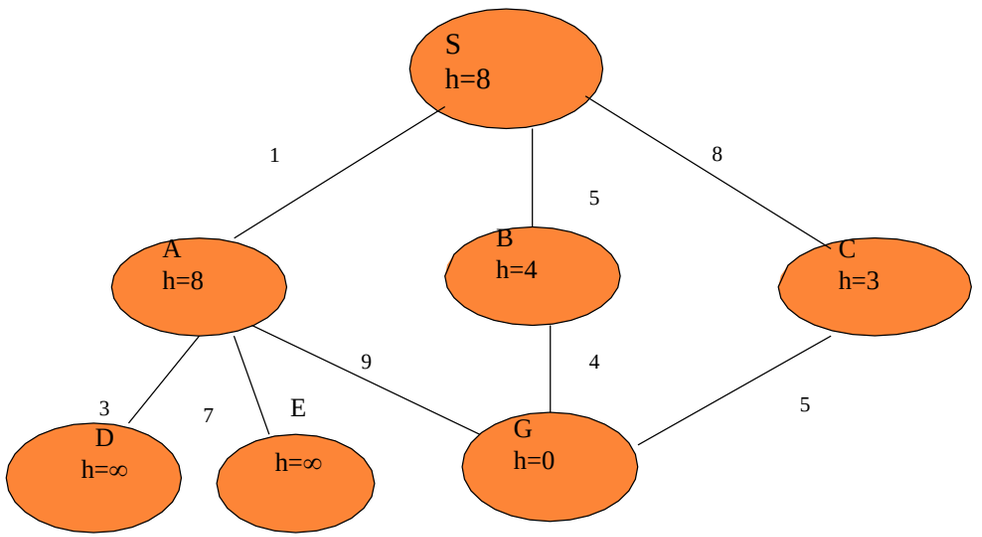


GREEDY SEARCH –EXAMPLE

f(n)= h(n)

# of nodes tested 2, expanded 2

Expanded node	**OPEN list**
	(S:8)
S	(C:3,B:4,A:8)
C is not goal	(G:0,B:4,A:8)

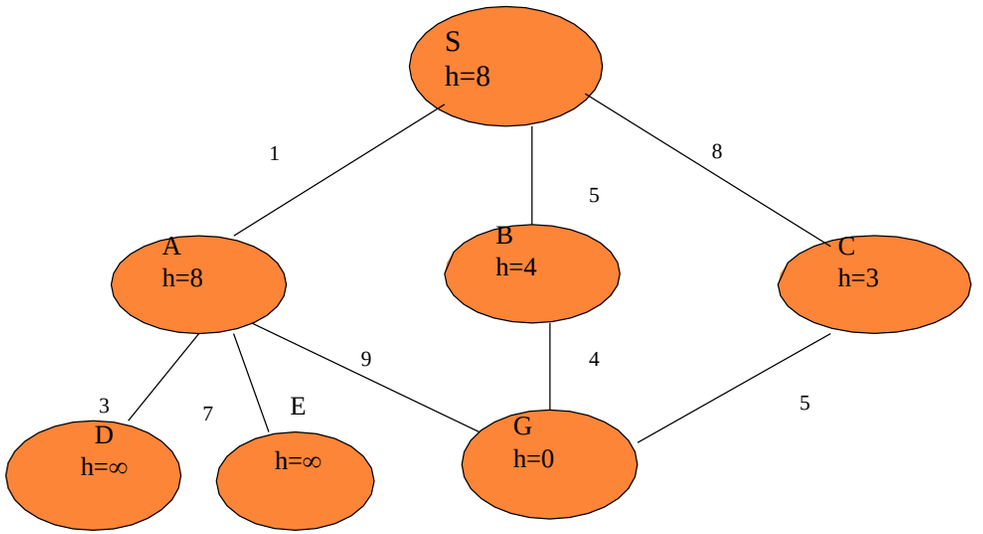


GREEDY SEARCH –EXAMPLE

f(n)= h(n)

# of nodes tested 3, expanded 2

Expanded node	**OPEN list**
	(S:8)
S	(C:3,B:4,A:8)
C	(G:0,B:4,A:8)
G goal	(B:4.A:8)



Path: S,C,G Cost: 13

CONT…

PROPERTIES OF GREEDY BEST-FIRSTSEARCH

- *Complete?*
  - No – can get stuck in loops,
- *Time?*
  - *O(bm) – worst case* (like Depth First Search)
  - But a good heuristic can give dramatic improvement
- *Space?**O(bm)* – keeps all nodes in memory

- *Optimal?* No

- Advantages:

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

➢This algorithm is more efficient than BFS and DFS algorithms.

- Disadvantages:

➢It can behave as an unguided depth-first search in the worst case scenario.

➢It can get stuck in a loop as DFS.

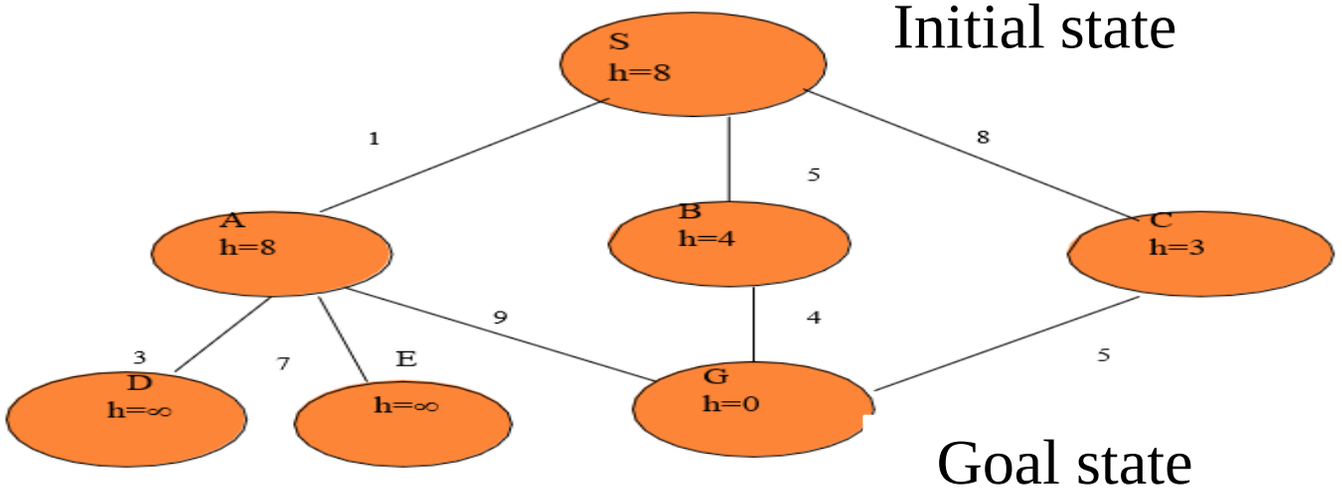
➢This algorithm is not optimal.

A* SEARCH

- **Best-known** form of best-first search.
- Goal: minimizes the total path cost
- Idea: avoid expanding paths that are already expensive.
- A∗ search uses an *admissible heuristic*

  i.e., h(n) <= h∗(n) where h∗(n) is the *true cost from n to a* goal.
- (Also require h(n) = 0, so h(G) = 0 for any goal G.)
- E.g., h(n) never over estimates the actual road distance
- Theorem: A∗ search is optimal if h is admissible.
- Evaluation function f(n)=g(n) + h(n) in A*
  - g(n) the cost (so far) to reach the node.
  - h(n) estimated cost to get from the node to the goal.
  - f(n) estimated total cost of path through *n* to goal.

- Example for A* search



Summary of **g(n)**, **h(n)**, **f(n) = g(n) + h(n)**, as well as **h*(n)**, the hypothetical perfect heuristic:

n	g(n)	h(n)	f(n)	h*(n)
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	inf	inf	inf
E	8	inf	inf	inf
G	10/9/13	0	10/9/13	0

Notice that since h(n) <= h*(n) for all n, h is admissible
Optimal path = S B G
Cost of the optimal path = 9

Summary of g(n), h(n), f(n) = g(n) + h(n), as well as h*(n), the hypothetical perfect heuristic:

n	g(n)	h(n)	f(n)	h*(n)
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	inf	inf	inf
E	8	inf	inf	inf
G	10/9/13	0	10/9/13	0

Notice that since h(n) <= h*(n) for all n, h is admissible
Optimal path = S B G
Cost of the optimal path = 9

A* Search: $f(n) = g(n) + h(n)$

node expanded	nodes list
	{ S(8) }
S	{ A(9) B(9) C(11) }
A	{ B(9) G(10) C(11) D(inf) E(inf) }
B	{ G(9) G(10) C(11) D(inf) E(inf) }
G	{ C(11) D(inf) E(inf) }

Solution path found is S B G. #nodes expanded = 4.

- Advantages:

➢A* search algorithm is the best algorithm than other search algorithms.

➢A* search algorithm is optimal and complete.

➢This algorithm can solve very complex problems.

- Disadvantages:

➢It does not always produce the shortest path as it mostly based on heuristics and approximation.

➢A* search algorithm has some complexity issues.

➢The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Avoiding Repeated States

- Avoiding repeated states in search algorithms is critical to improving efficiency and preventing unnecessary computations.

1. Use a Visited/Closed Set

- Maintain a set (or similar data structure) to track visited states. Before expanding a state:
- Check if the state is already in the visited set.
- If it is, skip processing that state. Otherwise, add the state to the visited set and proceed.

2. Use Recursive Backtracking with Marking

- In backtracking approaches, mark a state as visited before recursion and unmark it after returning.

3. Iterative Deepening (for Depth-First Search)

- Limit the depth of exploration and incrementally increase it.

This can implicitly reduce the chance of revisiting irrelevant states in some scenarios.

Constraint Satisfaction problem

- A **Constraint Satisfaction Problem (CSP)** is a type of problem where the goal is to find a solution that satisfies a set of constraints.
- CSPs are widely used in artificial intelligence (AI) for solving problems such as scheduling, assignment, and resource allocation.
- We use a **factored representation** for each state: a set of variables, each of which has a value.
- A problem is solved when each variable has a value that satisfies all the constraints on the variable.
- A problem described this way is called a **constraint satisfaction problem**, or CSP.

Con't…

- A constraint satisfaction problem consists of three components, Variable, Domain, and Constraints.
- **Variables**:
  The entities to be assigned values.
  Example: X,Y,Z
- **Domains**:
  The possible values that each variable can take.
  Example: $X \in \{1,2,3\}$, $Y \in \{1,2\}$, $Z \in \{a,b,c\}$
- **Constraints**:
  Rules that restrict the values the variables can simultaneously take.
  Example: $X \neq Y$, $X+Z \leq 5$

Con't…

- To solve a CSP, we need to define a state space and the notion of a solution.
- Each state in a CSP is defined by an **assignment** of values to some or all of the variables, {Xi = vi, Xj = vj, . . .}.
- An assignment that does not violate any constraints is called a **consistent** or legal assignment.
- A **complete assignment** is one in which every variable is assigned, and a **solution** to a CSP is a consistent, complete assignment.
- A **partial assignment** is one that assigns values to only some of the variables.

Con’t…

Types of CSPs

- **Finite CSP**:
  Variables have finite domains (e.g., scheduling, Sudoku).
- **Infinite CSP**:
  Variables can take values from infinite domains (e.g., real numbers in equations).
- **Binary CSP**:
  Constraints involve pairs of variables (e.g., X<Y).
- **Higher-order CSP**:
  Constraints involve three or more variables (e.g., X+Y+Z=10, A<B<C)

Example of a CSP

- **Problem:**
- Imagine we have a map with four regions: A, B, C, and D. We want to color each region such that no two adjacent regions have the same color. We have three colors available: Red, Green, and Blue..
- **Variables**: Regions (e.g., A,B,C,D).
- **Domains**: Colors (e.g., {Red, Green, Blue}).
- **Constraints**: Adjacent regions cannot have the same color (e.g., A≠B, A≠C, B≠C, B≠D,C≠D).

Solving CSPs

1. Backtracking Search:

- A systematic search algorithm that tries assigning values to variables and backtracks when constraints are violated.
- **Improvement Techniques**:
  - **Forward Checking**: Eliminate inconsistent values from domains of unassigned variables.
  - **Constraint Propagation**: Enforce constraints (e.g., Arc Consistency using AC-3 algorithm).

2. Local Search:

- Start with a random solution and iteratively improve by making small changes.
  Example: Min-Conflicts heuristic for resolving violated constraints.

Con't…

3. Heuristics:

- **Most Constrained Variable (MRV)**: Assign a value to the variable with the fewest legal values.
- **Least Constraining Value (LCV)**: Choose the value that leaves the most options for other variables.

4. Consistency Algorithms:

- **Node Consistency**: Ensure individual variables satisfy unary constraints.
- **Arc Consistency (AC-3)**: Ensure every value of one variable has a consistent value in another.

Applications of CSP

- **Scheduling**: Allocating resources to tasks subject to constraints.
- **Timetabling**: Assigning classes to timeslots without conflicts.
- **Puzzle Solving**: Sudoku, N-Queens problem.
- **Configuration Problems**: Designing systems subject to compatibility constraints.
- **Logical Reasoning**: Modeling and solving logical inference problems.

Games as Search Problems

- **Game theory** views any multi-agent environment as a game, provided that the impact of each agent on the others is “significant,” regardless of whether the agents are cooperative or competitive.
- **Games as Search Problems** is a fascinating concept in artificial intelligence, where solving a game is modeled as a search through a space of possible moves, leading to a winning, losing, or drawn outcome.

Components of Games as Search Problems

- A game can be formally defined as a kind of search problem with the following elements:

1. **STATE SPACE**: Represents all possible configurations of the game. For example, in chess, each possible arrangement of pieces on the board is a state.
2. **S0**: The initial state, which specifies how the game is set up at the start
3. **PLAYER(p):** Defines which player has the move in a state.
4. **ACTIONS(a):** Returns the set of legal moves in a state.
5. **RESULT(p, a):** The transition model, which defines the result of a move
6. **TERMINAL-TEST(s):** A terminal test, which is true when the game is over and false otherwise. States where the game has ended are called terminal states.
7. **UTILITY(s, p):** A utility function, defines the final numeric value for a game that ends in terminal state s for a player p.

Example: Tic-Tac-Toe

- **State Space**: All possible board configurations (up to 3^9=19,683 states, considering the 3x3 grid).
- **Initial State**: Empty board.
- **Players**: Player 1 (X) and Player 2 (O).
- **Actions**: Placing X or O in any empty cell.
- **Transition Model**: Updates the board after a move.
- **Terminal States**: A row, column, or diagonal is filled with the same symbol (win), or no empty cells remain (draw).
- **Utility Function**:
  - +1 for X wins, -1 for O wins, 0 for a draw.

Search Algorithms for Games

- **Minimax Algorithm**:
  - A recursive strategy to minimize the possible loss in a worst-case scenario.
  - Alternates between maximizing and minimizing players' moves.
- **Alpha-Beta Pruning**:
  - Optimized version of Minimax.
  - Prunes branches of the search tree that cannot affect the final decision.
- **Monte Carlo Tree Search (MCTS)**:
  - Uses random sampling to evaluate states, common in games with large state spaces like Go.

Single-Agent vs. Multi-Agent Search

- **Single-Agent Games**: The search focuses on solving puzzles or finding optimal paths (e.g., Sudoku, Maze).
- **Multi-Agent Games**: Include adversarial dynamics, requiring strategies to handle opponents (e.g., Chess, Checkers).

•Real-World Applications

- **AI in Board Games**: Chess engines like Stockfish, Go AI like AlphaGo.
- **Video Games**: AI controlling enemies or NPCs in real-time strategy or role-playing games.
- **Decision-Making**: Autonomous agents navigating adversarial environments.

THE ENDS OF CHAPTER 3