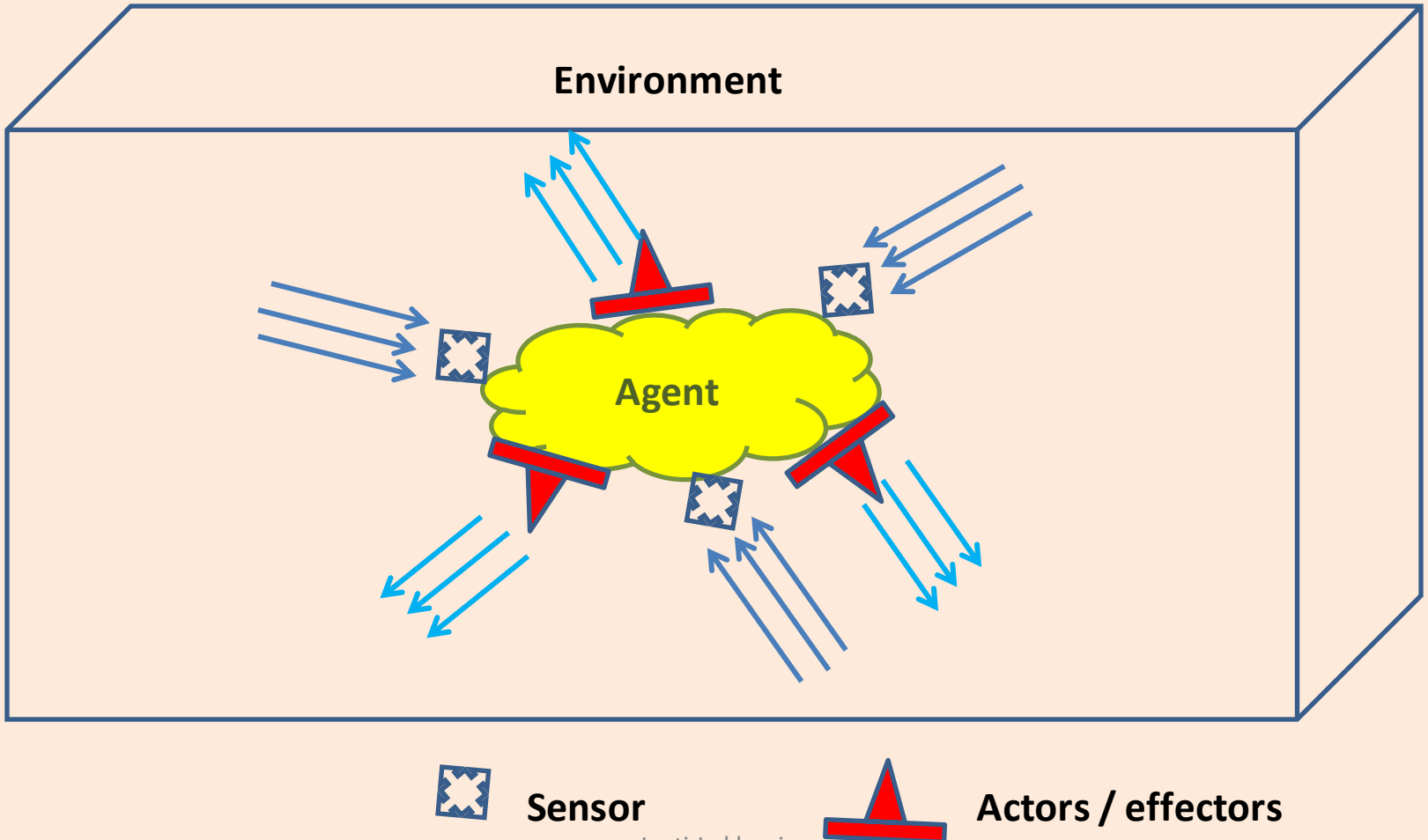
Agent

An agent is anything that can be viewed as **perceiving** its environment through **sensors and acting** upon that environment through **effectors**.



Agent

The job of AI is to design the **agent program.**

Agent program is a function that implements the **mapping** from percepts to actions.

A problem is-

a collection of information that will the agent use to decide what to do.

Together, the initial state, operator set, goal test, and path cost function define a problem.

1. **The initial state** that the agent knows itself to be in
2. **Set of possible actions** available to the agent, called **Operators**
3. **The state space of the problem**: The set of all states reachable from the initial state by any sequence of actions.
4. **A path** in the state space is simply any sequence of actions leading from one state to another.
5. **The goal test,** which the agent can apply to a single state description to determine if it is a goal state
6. **A path cost** function is a function that assigns a cost to a path.

Measuring problem-solving performance

The effectiveness of a search can be measured in at least three ways -

First, does it find a solution at all?

Second, is it a good solution (one with a low path cost)?

Third, what is the search cost associated with the time and memory required to find a solution?

The total cost of the search is-

The sum of the path cost + The search cost.

SEARCHING FOR SOLUTIONS

finding a solution—is done by a search through the state space

- applying the operators to the current state
  - thereby **generating** a new set of states.
    - The process is called **expanding the** state

This is the essence of search—choosing one option and putting the others aside for later, in case the first choice does not lead to a solution.

Search Strategy

The Search Strategy determines which state to expand first.

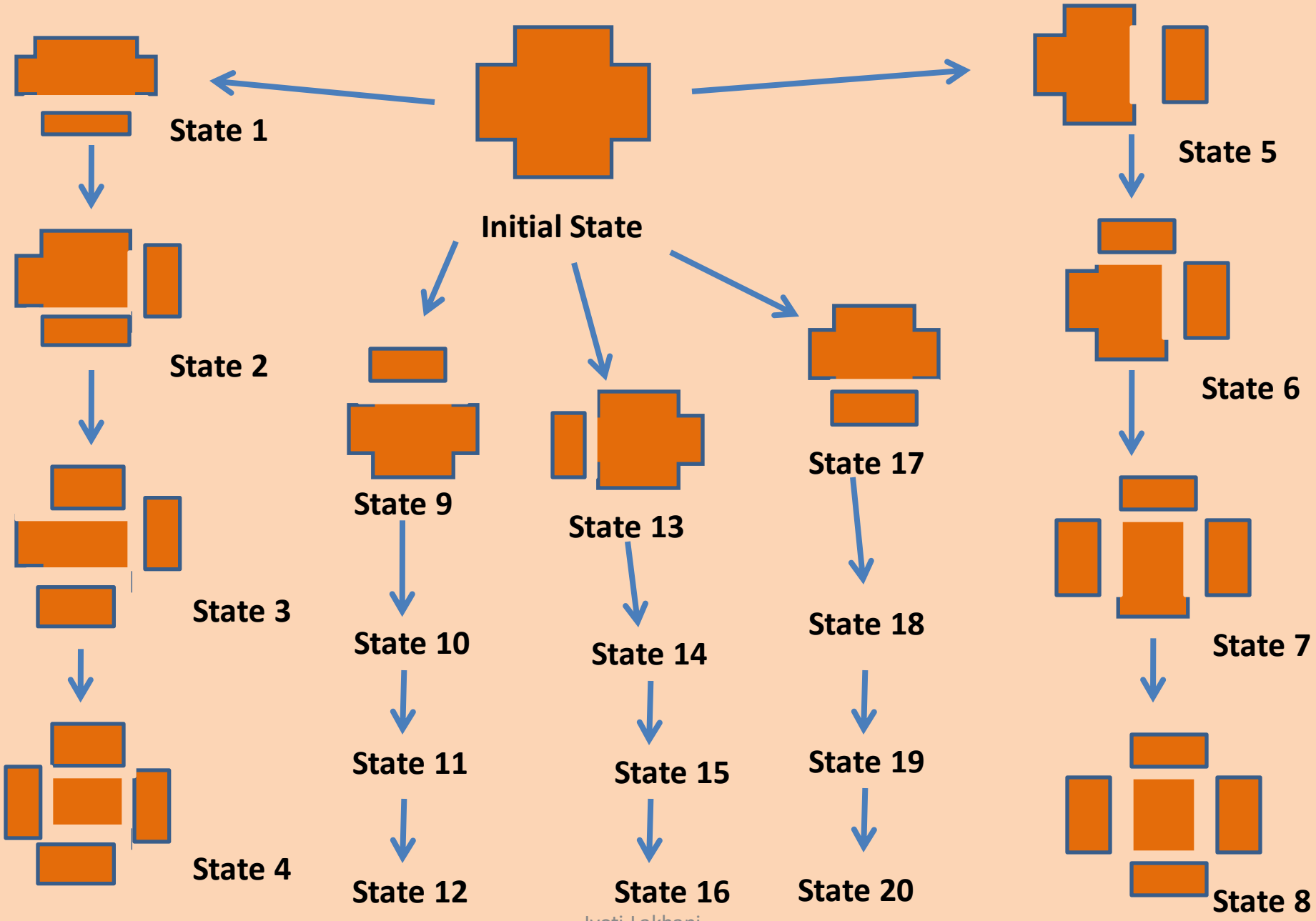
It is helpful to think of the search process as building up a **search tree** that is superimposed over the state space.

- 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, 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.

Problem Solving As Search



There are two main approaches to searching a search tree-

Data-Driven Search

Data-driven search starts from an initial state and uses actions that are allowed to move forward until a goal is reached. This approach is also known as **forward chaining.**

Goal-Driven Search

Alternatively, search can start at the goal and work back toward a start state, by seeing what moves could have led to the goal state. This is **goal-driven search, also known as backward chaining.**

goal-driven search is preferable to data driven search

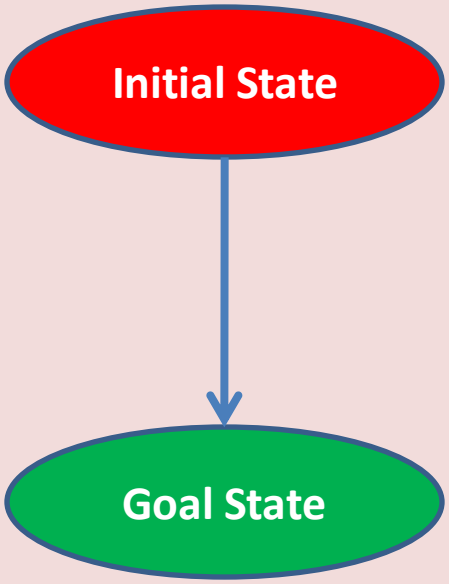
Goal-driven search and data-driven search will end up producing the same results, but depending on the nature of the problem being solved, in some cases one can run more efficiently than the other

Goal-driven search is particularly useful in situations in which the goal can be clearly specified (for example, a theorem that is to be proved or finding an exit from a maze).

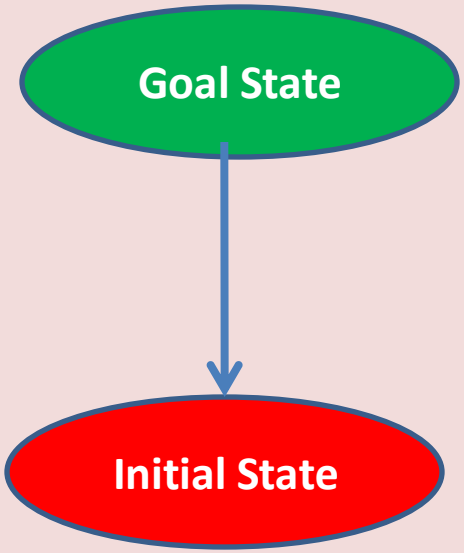
It is also clearly the best choice in situations such as medical diagnosis where the goal (the condition to be diagnosed) is known, but the rest of the data (in this case, the causes of the condition) need to be found.

Data-driven search is most useful when the initial data are provided, and it is not clear what the goal is.

It is nearly always far easier to start from the end point and work back toward the start point. This is because a number of dead end paths have been set up from the start (data) point, and only one path has been set up to the end (goal) point. As a result, working back from the goal to the start has only one possible path.

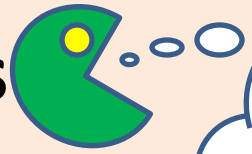


Data Driven Search
Or
Forward Chaining



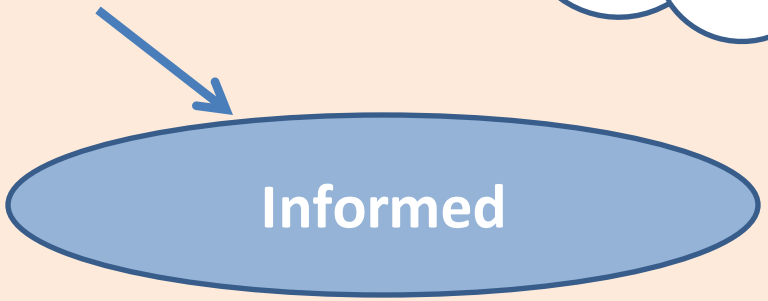
Goal Driven Search
Or
Backward Chaining

Search Methodologies
OR
Search Techniques



We are talking about Methodologies not methods

Brute Force Search



No Information Required

1. Generate and Test

SiMpLeSt

2. Depth First Search

CoMmOn

3. Breath First Search

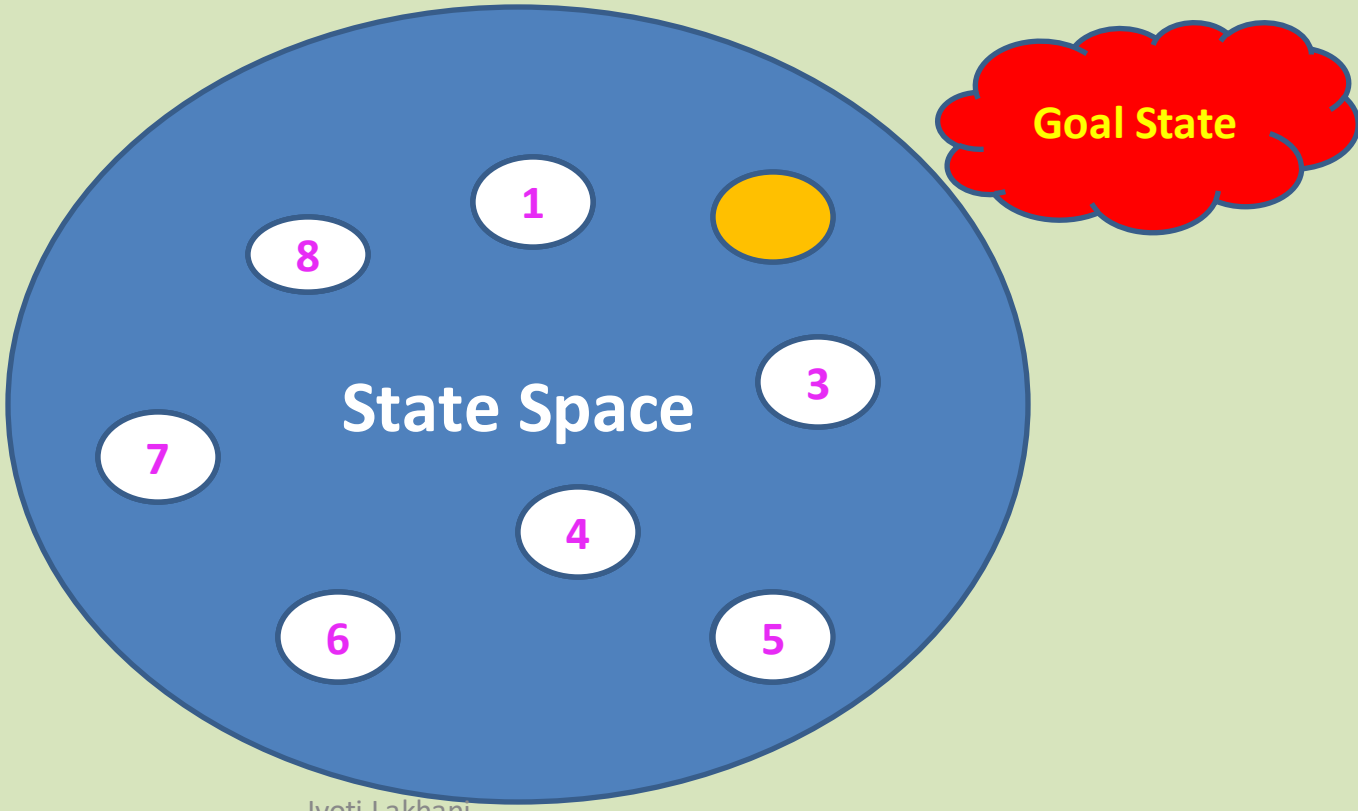
AlTeRnAtIvE

Additional Information required i.e. called Heuristics

Informed Techniques are more intelligent than Un-informed techniques

Generate and Test
Or Brute Force Search

- Simplest Approach
- Examine every node in the tree until it finds a goal
- Solve problems where there is no additional information about how to reach a solution.



To successfully operate
Generate and Test needs to have a suitable **Generator**

with three properties-

1. It must be complete

It must generate every possible solution

2. It must be non-redundant

It should not generate the same solution twice

3. It must be well informed

It should only propose suitable solutions and should not examine possible solutions that do not match the search space

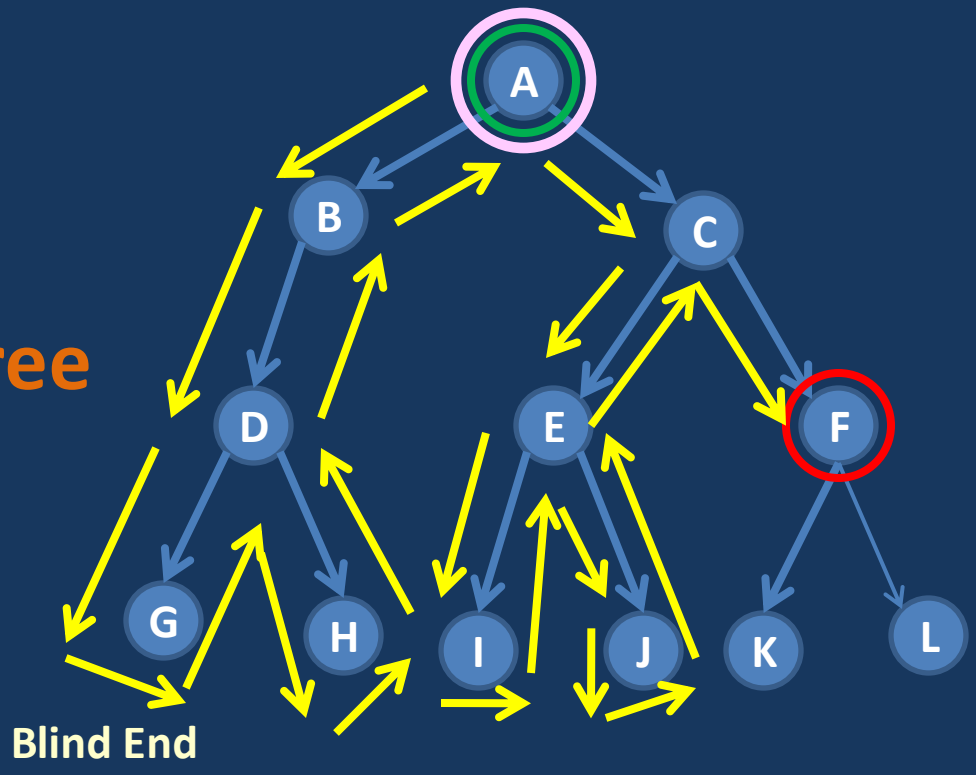
Depth-First Search

it follows each path to its

greatest depth

before moving on to the next path

Search Tree



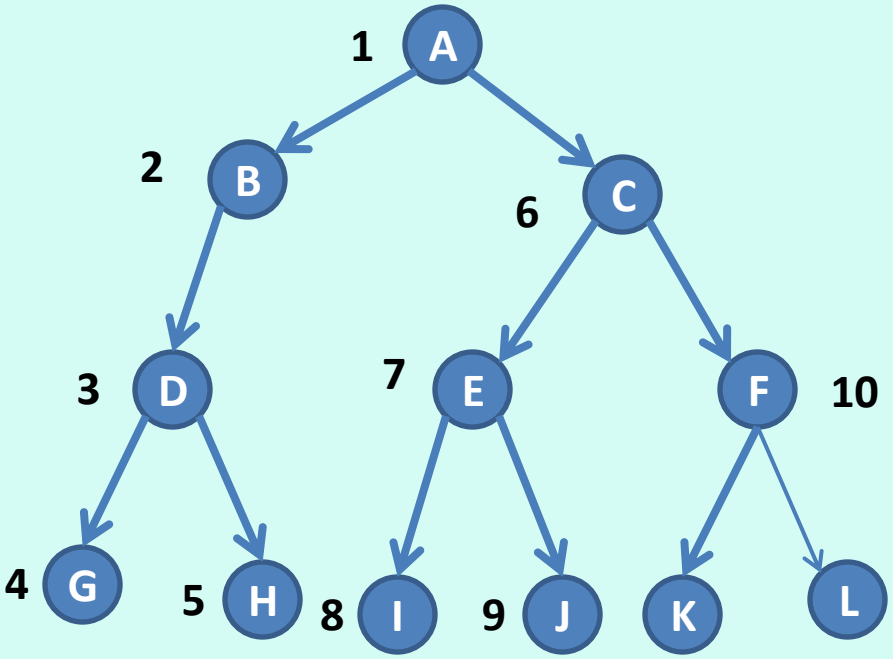
Initial State

Current State

Goal State

Depth-first search is often used by computers for search problems such as locating files on a disk, or by search engines for spidering the Internet.

Search Sequence A B D G H C E I J F



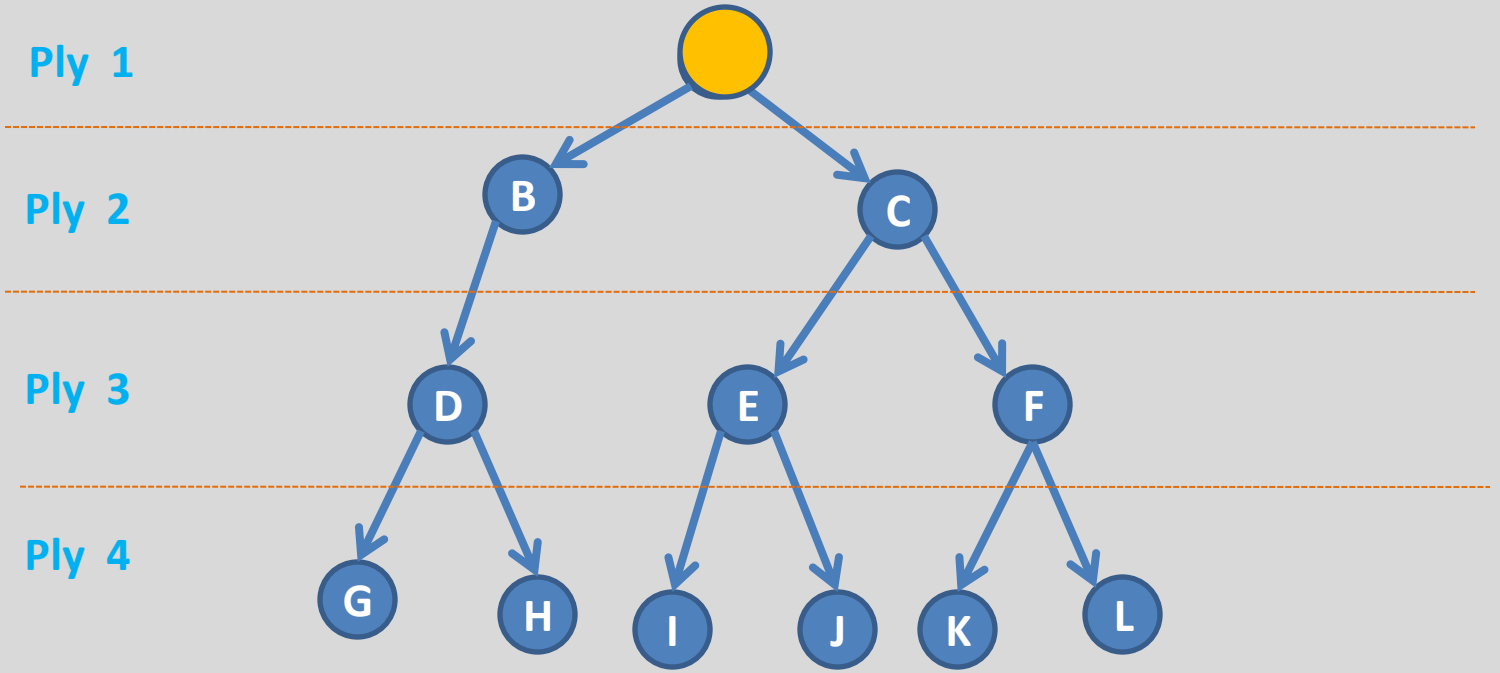
Depth First Algorithm

```
Function depth ()
{
        queue = []; // initialize an empty queue
        state = root_node; // initialize the start state
        while (true)
        {
                if is_goal (state)
                        then return SUCCESS
                else add_to_front_of_queue
                        (successors (state));
                if queue == []
                        then report FAILURE;
                state = queue [0]; // state = first item in
                                        queue
                remove_first_item_from (queue);
        }
}
```

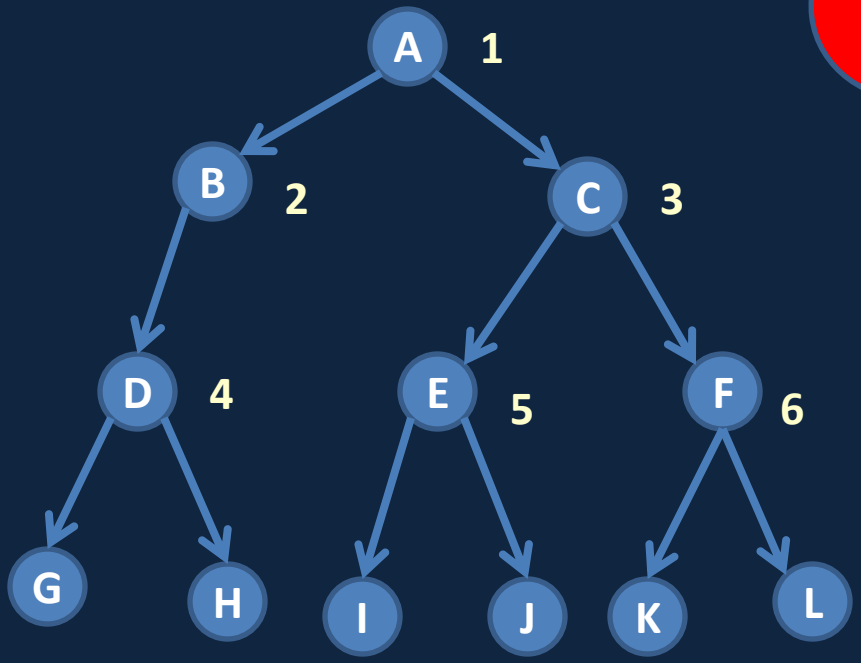
Breadth-First Search

This approach involves traversing a tree by breadth rather than by depth.

The breadth-first algorithm starts by examining all nodes one level (sometimes called one **ply) down from the** root node.



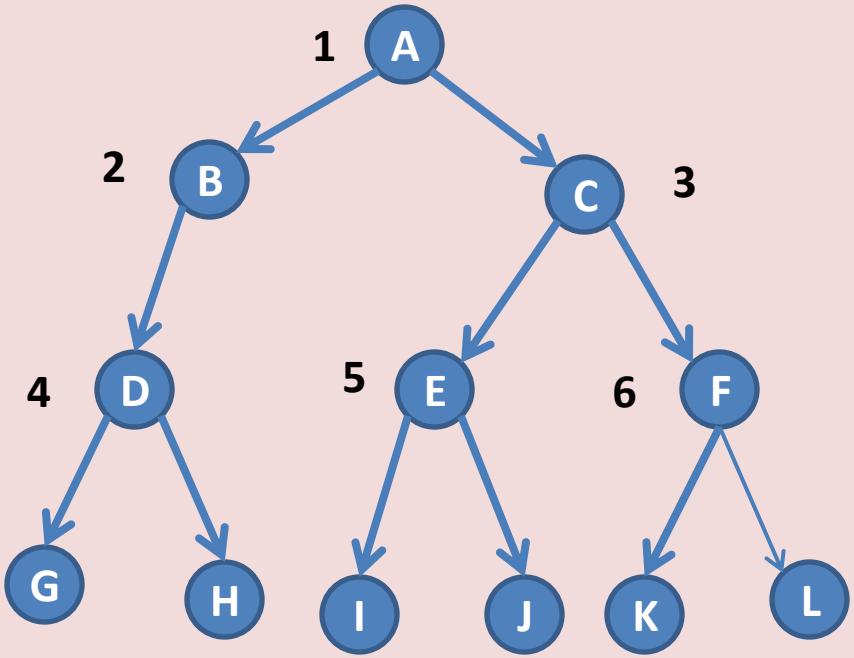
IF goal Then
success
Else
search continues to the next ply
Else
Fail



Breadth-first search is a far better method to use in situations where the tree may have very deep paths

Breadth-first search is a poor idea in trees where all paths lead to a goal node with similar length paths

Search Sequence A B C D E F G H I J



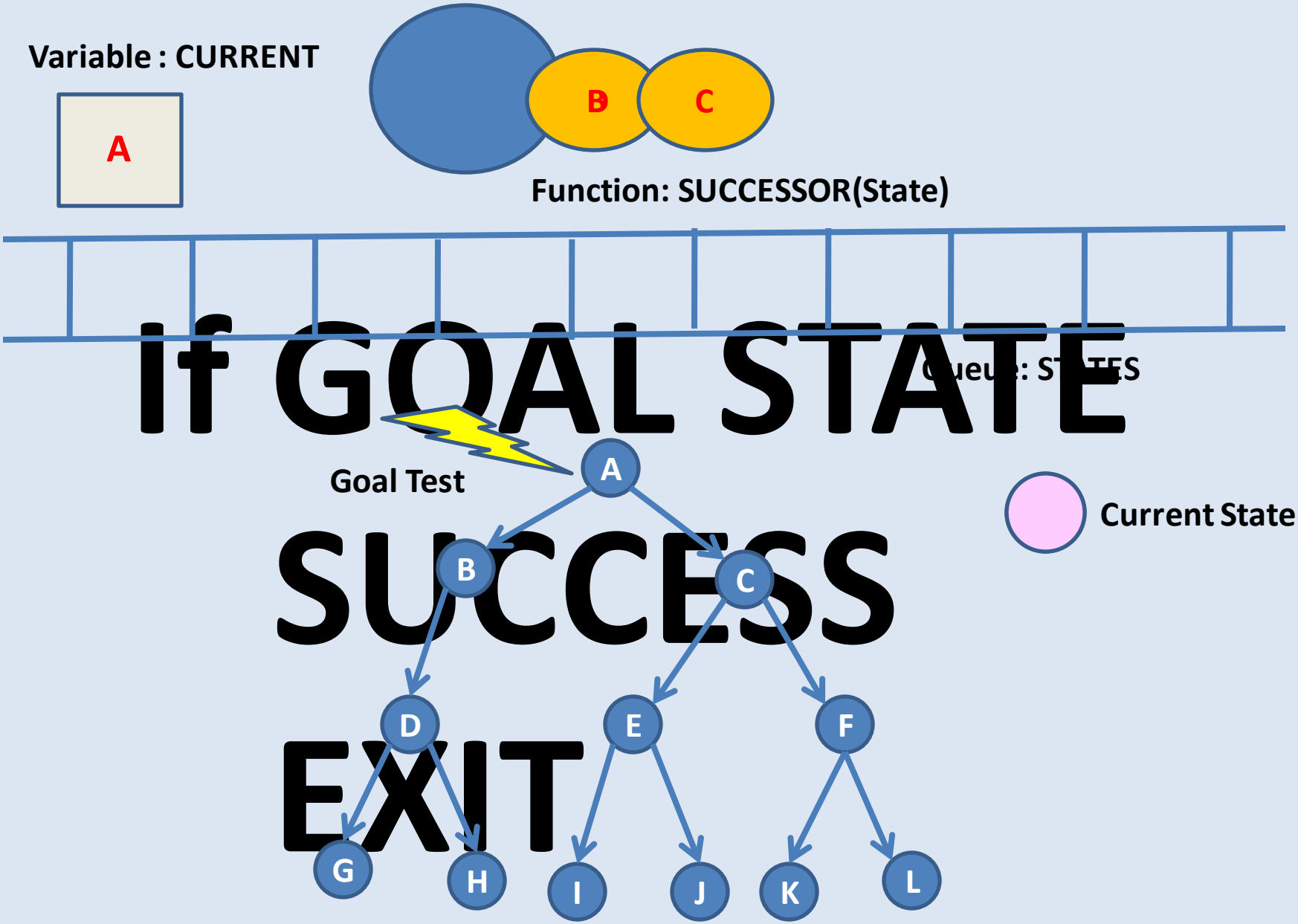
Breadth First Algorithm

```
Function breadth ()
{
queue = []; // initialize an empty queue
state = root_node; // initialize the start state
        while (true)
        {
         (state if is_goal)
                        then return SUCCESS
                else add_to_back_of_queue
                        (successors (state));
                if queue == []
                        then report FAILURE;
                        state = queue [0];
                remove_first_item_from (queue);
        }
}
```

Difference Between Depth and Breadth First Search

Scenario	Depth First Search	Breath First Search
Some paths are extremely long, or even infinite	Performs badly	Performs well
All paths are of similar length	Performs well	Performs well
All paths are of similar length, and all paths lead to a goal state	Performs well	Wasteful of time and memory
High branching factor	Performance depends on other factors	Performs poorly

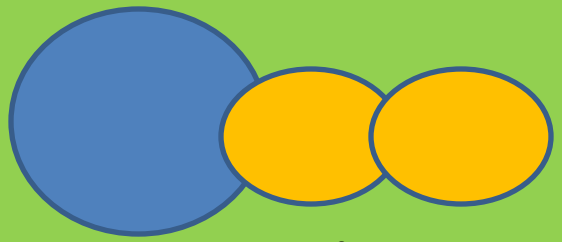
Implementing Breadth-First Search



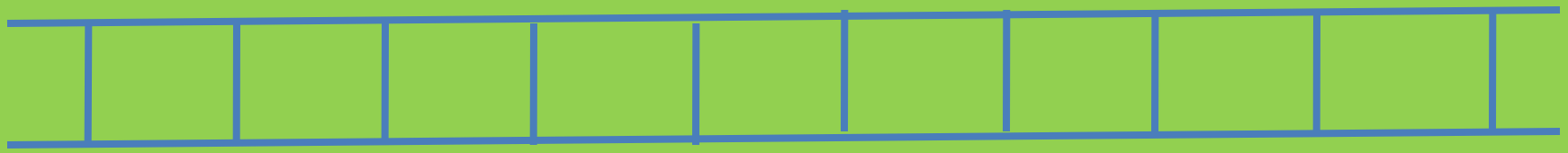
Step	Current State	Queue	Comment
1	A	[]	Queue Empty
2	A	[BC]	A is not a Goal State, Expend A
3	B	[C]	B is Current State
4	B	[CD]	B is not a Goal State, Expend B
5	C	[D]	
6		. . .	
7			

Implementing Depth-First Search

Variable : CURRENT

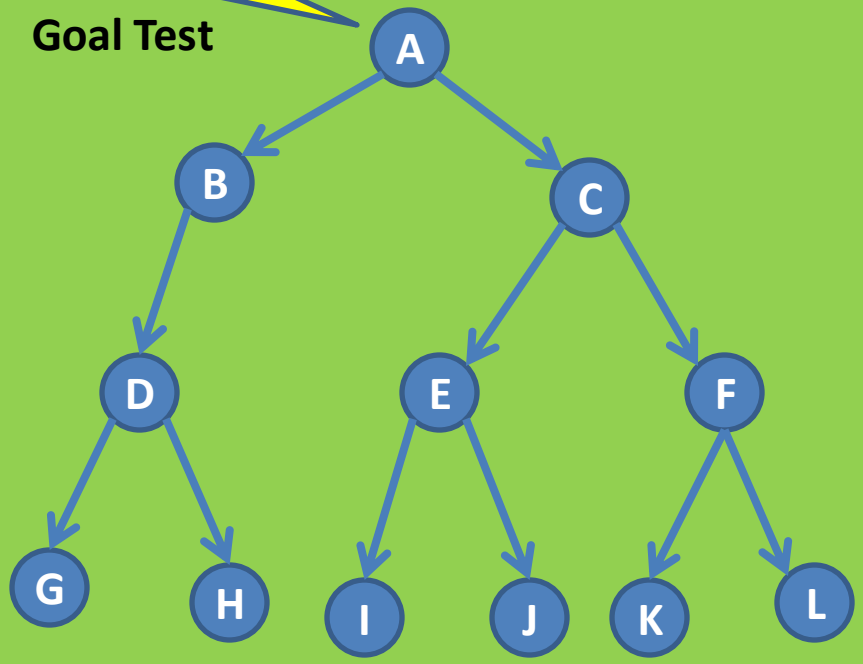


Function: SUCCESSOR(State)



Stack: STATES

Goal Test



Current State

Properties of Search Methods

Complexity

- Space Complexity
- Time Complexity

Depth-first search is very efficient in space because it only needs to store information about the path it is currently examining

But it is not efficient in time because it can end up examining very deep branches of the tree.

Properties of Search Methods

Completeness

A Search Method should guaranteed to find a goal state if one exists

Breadth-first search is complete, but depth-first search is not because it may explore a path of infinite length and never find a goal node that exists on another path.

Properties of Search Methods

Optimality

A search method is **optimal if it is guaranteed to find the best solution that** exists.

Breadth-first search is an optimal search method, but depth-first search is not.

Depth-first search returns the first solution it happens to find, which may be the worst solution that exists.

Because breadth-first search examines all nodes at a given depth before moving on to the next depth, if it finds a solution, there cannot be another solution before it in the search tree.

Properties of Search Methods

Irrevocability

Methods that use backtracking are described as tentative

Methods that do not use backtracking, and which therefore examine just one path, are described as irrevocable.

Depth-first search is an example of tentative search.

Using Heuristics for Search: Informed Search

Depth-first and breadth-first search were described as brute-force search methods.

This is because they do not employ any special knowledge of the search trees they are examining

but simply examine every node in order until the goal.

This kind of information is called a **heuristic**

Heuristics can reduce an impossible problem to a relatively simple one.

Heuristic Search Uses a Heuristic Evaluation Function or Heuristic Function

Heuristic Function

Heuristic Function apply on a Node

It will Give the Estimate of Distance of goal from Node

H(Node) = Distance of Node from Goal

Suppose there are two nodes m and n and

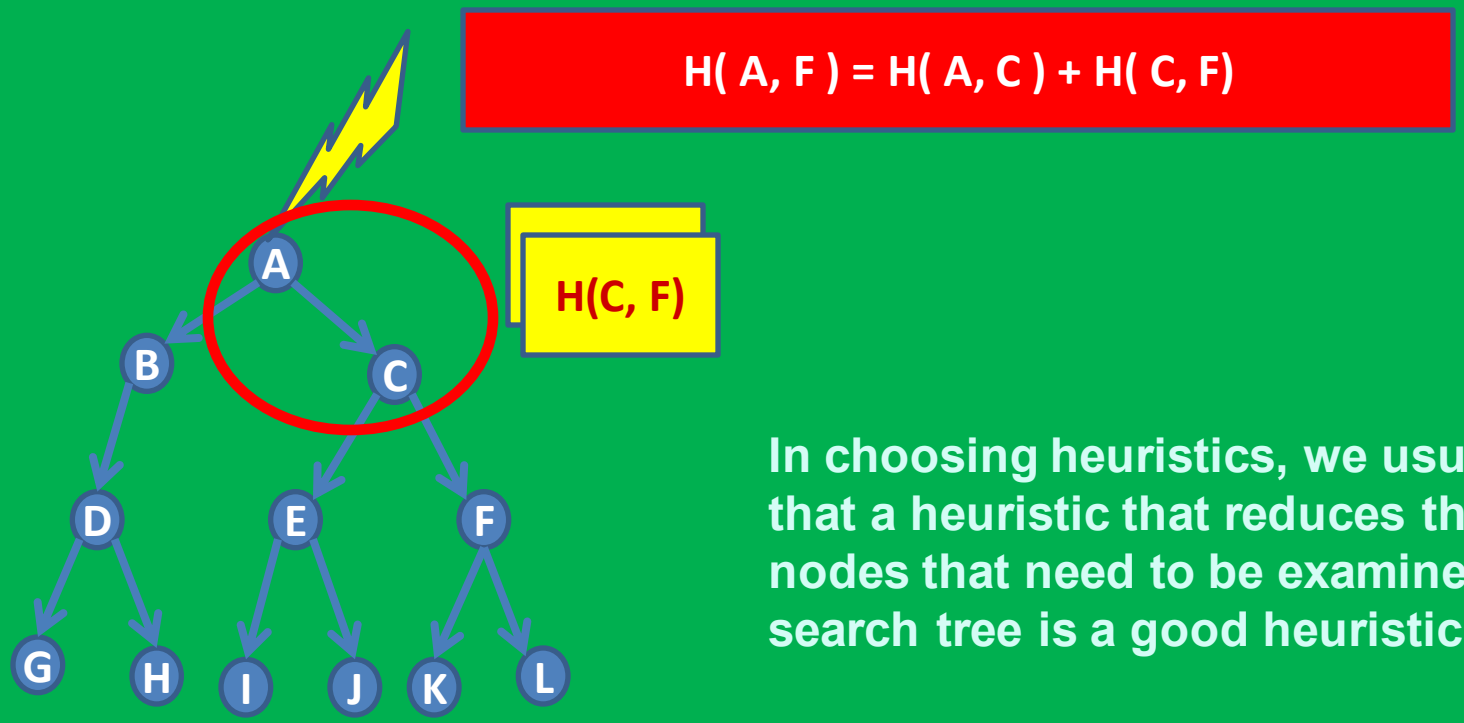
H(m,g) < H(n,g)

m is more likely to be on an ***optimal*** **path to the goal node than** ***n.***

Suppose " F " is the Goal node

Heuristic Estimate from A to Goal node F is the Summation of –

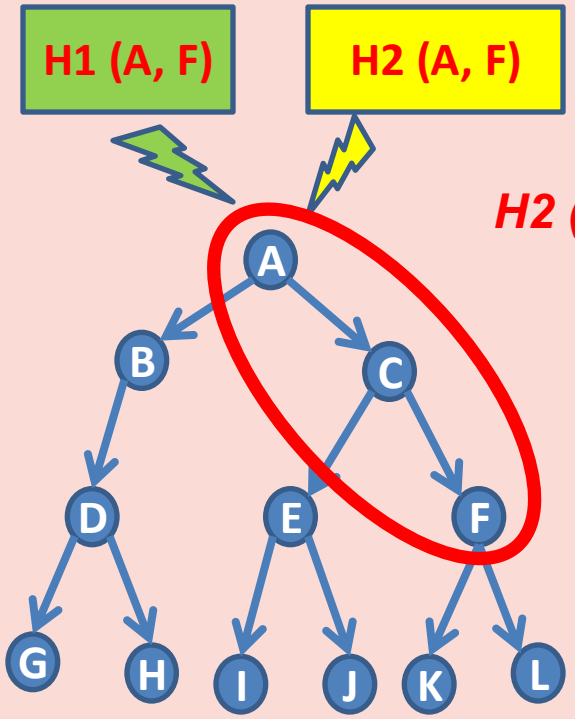
H(A,C) and H(C,F)



In choosing heuristics, we usually consider that a heuristic that reduces the number of nodes that need to be examined in the search tree is a good heuristic.

More than One Heuristics can be applied to the same search

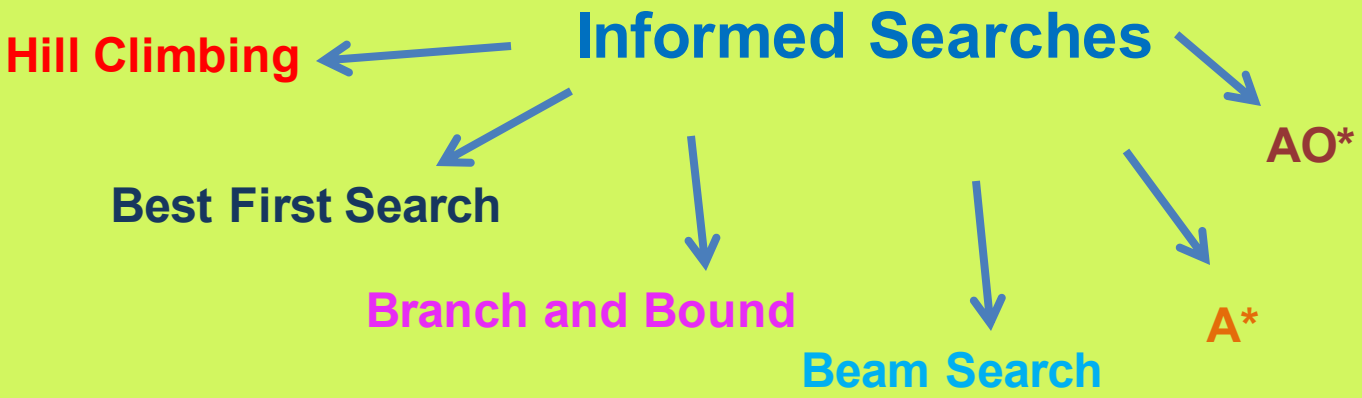
Suppose we have to Reach the Goal Node , which is F



H2 (node, Goal) ≥ H1 (node, Goal)

H2 dominates H1

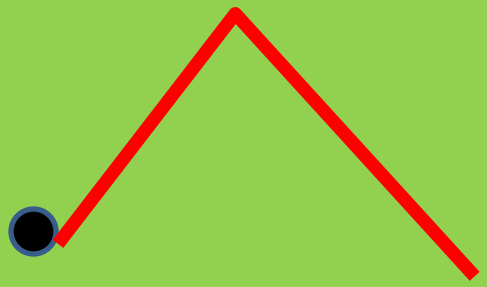
Which means that a search method using heuristic h2 will always perform more efficiently than the same search method using *h1.*



Hill Climbing

In examining a search tree,
hill climbing will move to the first successor node that is "better" than the current node

—in other words, the first node that it comes across with a heuristic value lower than that of the current node.



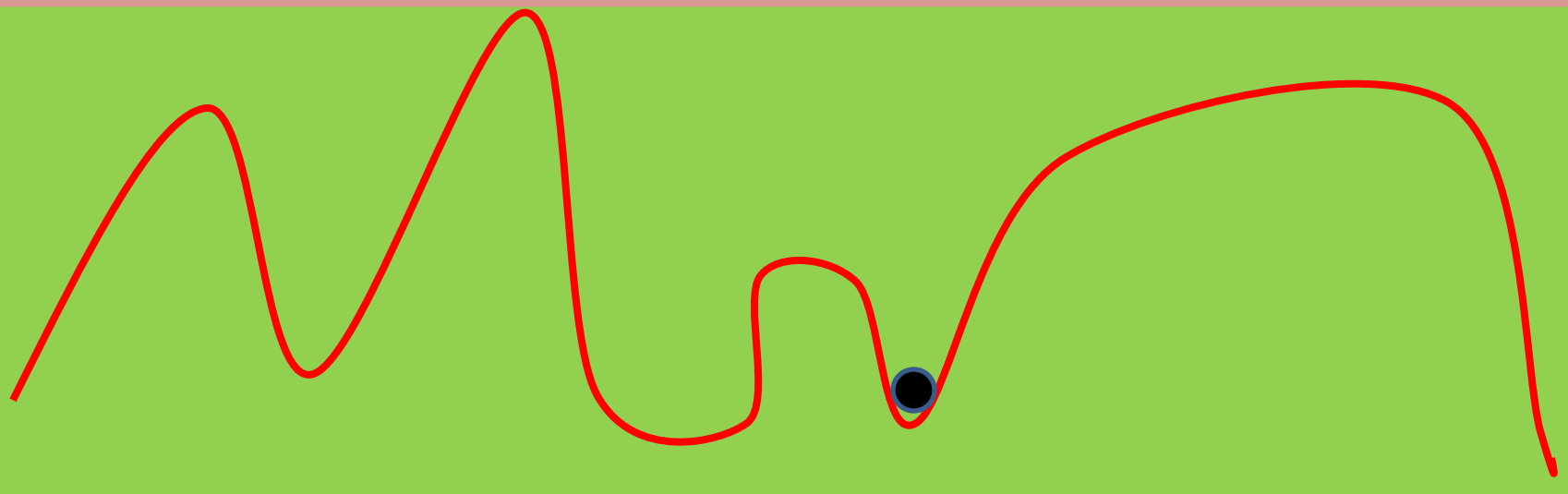
If all directions lead lower than your current position, then you stop and assume you have reached the summit. As we see later, this might not necessarily always be true.

Steepest Ascent Hill Climbing

Similar to Hill Climbing

Except that rather than moving to the first position you find that is higher than the current position

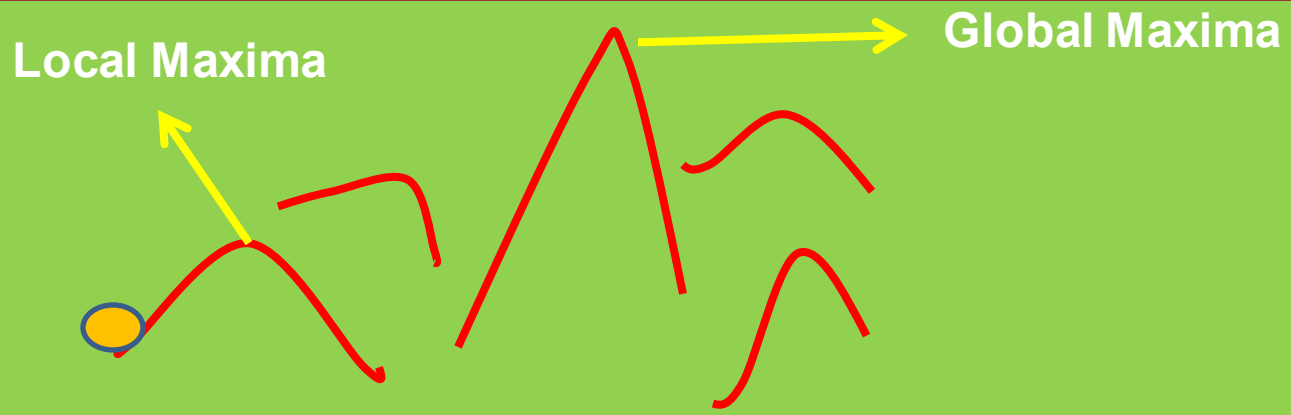
you always check around you in all four directions and choose the position that is highest.



Three Problems with Hill Climbing

1. Local Maxima or Foot Hill

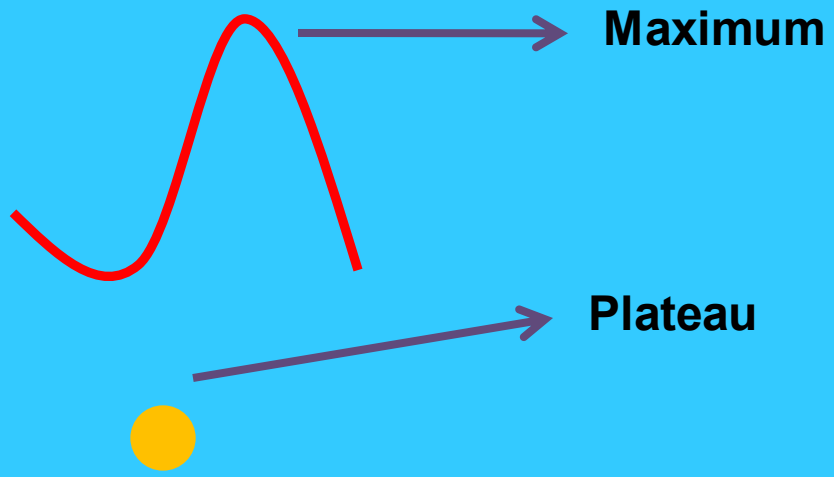
A local maximum is a part of the search space
that appears to be preferable to the parts around it,
but which is in fact just a foothill of a larger hill



Three Problems with Hill Climbing

2. Plateau

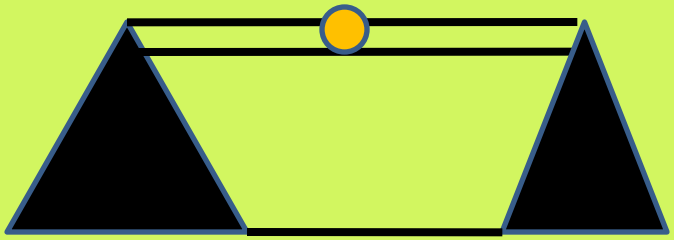
A plateau is a region in a search space where all the values are the same



Three Problems with Hill Climbing

3. Ridge

A ridge is a long, thin region of high land with low land on either side.



Best First Search = Features of Depth First Search

+

Features of Breadth First Search

Depth First Search is good becoz it –

Found solution without expanding all competing branches

Breadth First is good becoz it –

Does not get trapped on dead end paths

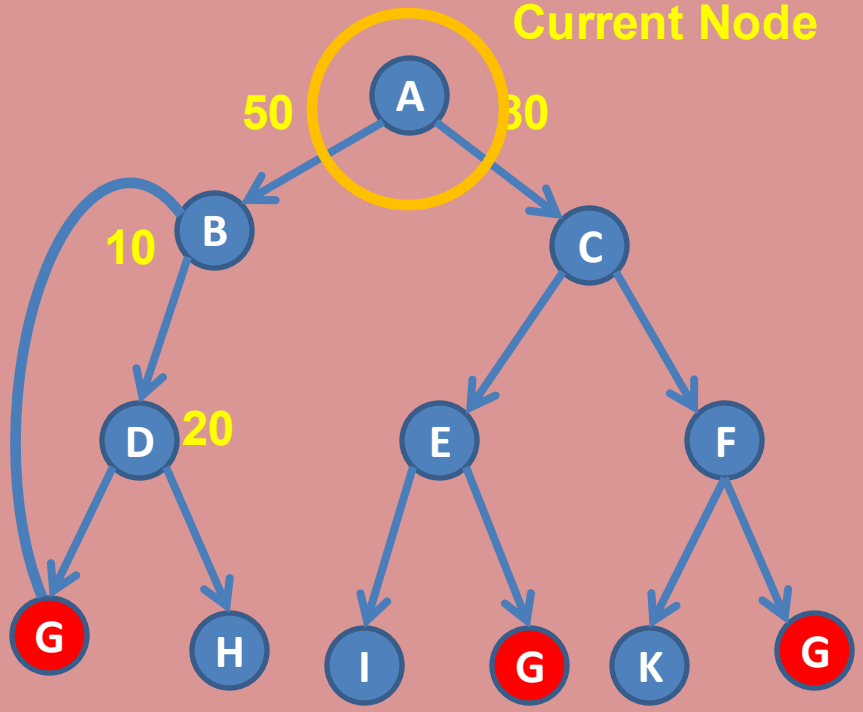
Best First search combine both these characters

At each step of Best First Search process

select the most promising nodes,

we have generated so far

Best First Search



$$f(n) = h(n)$$

Beam Search

Same like Best First Search,
but n promising states are kept for future considerations

A* Algorithm

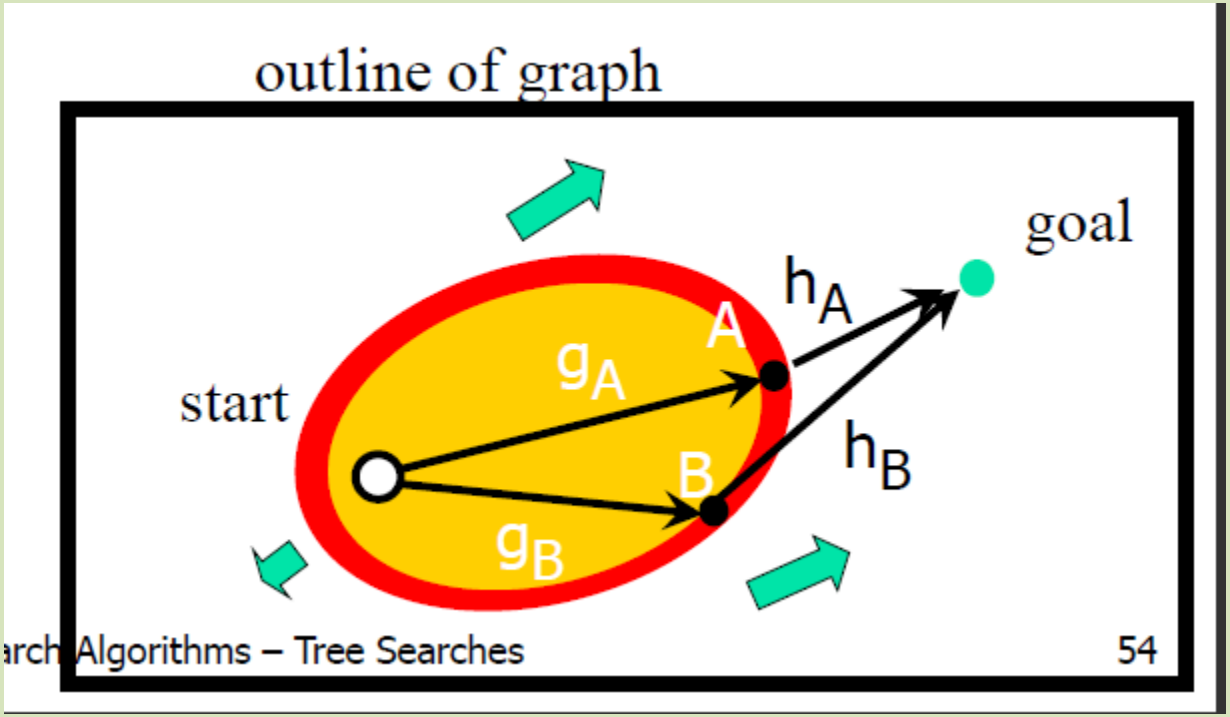
Variation of Best First Search

Along each node
on a path to the goal,
A* generates all successor nodes and
estimates the distance (cost)
from the start node to the goal node

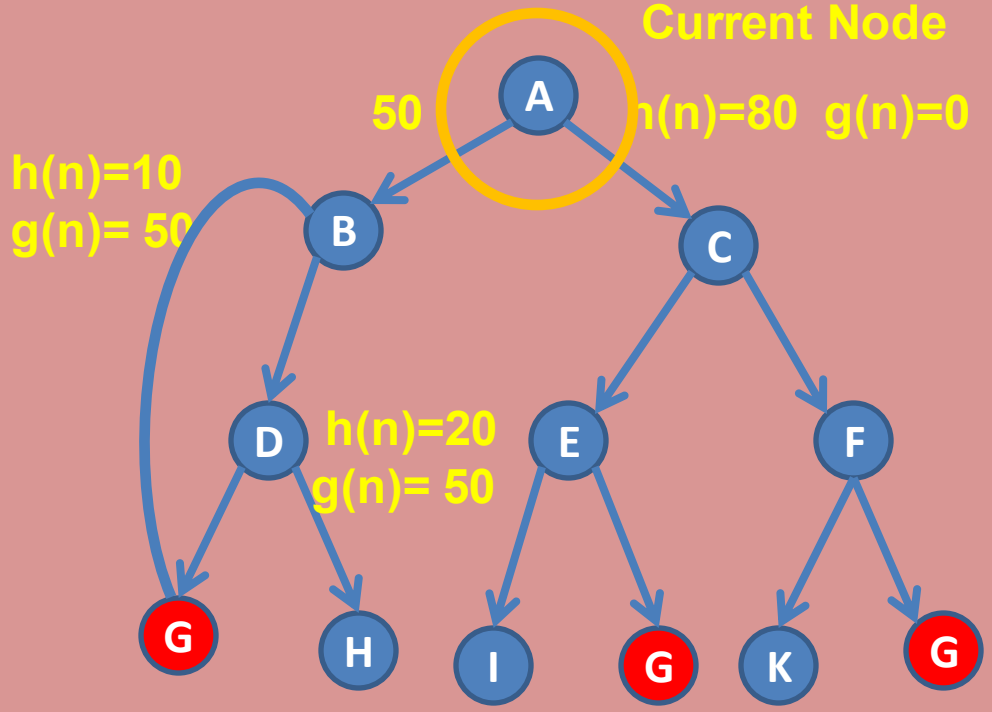
A * Combines the cost so far and the estimated cost to the goal

That is evaluation function f(n) = g(n) + h(n)

An estimated cost of the cheapest solution via n



A* Search

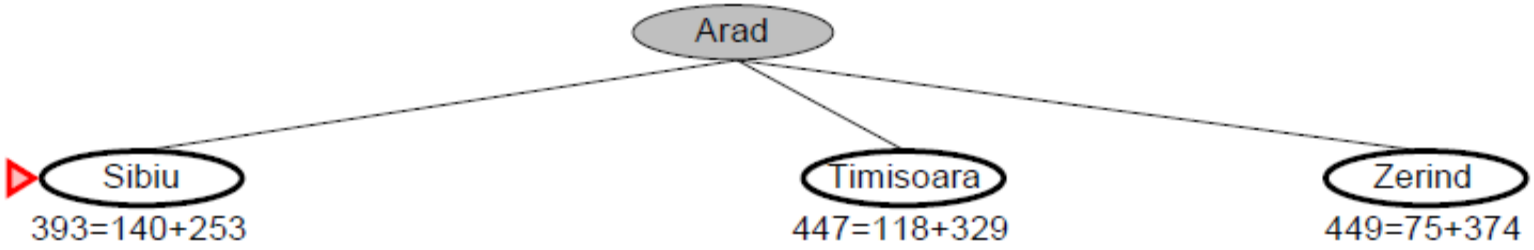


$$f(n)= g(n) + h(n)$$

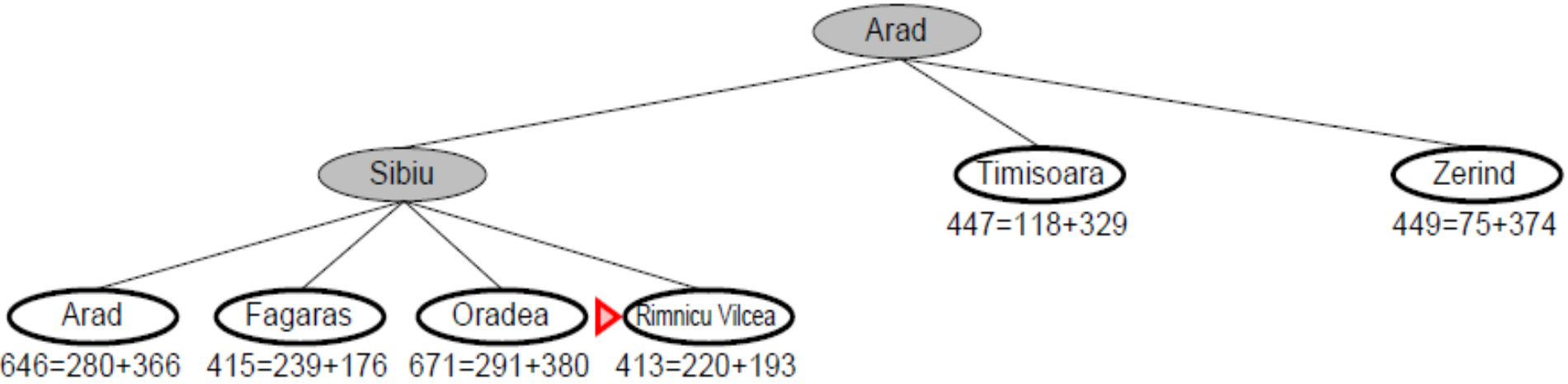
A* search example

366=0+366

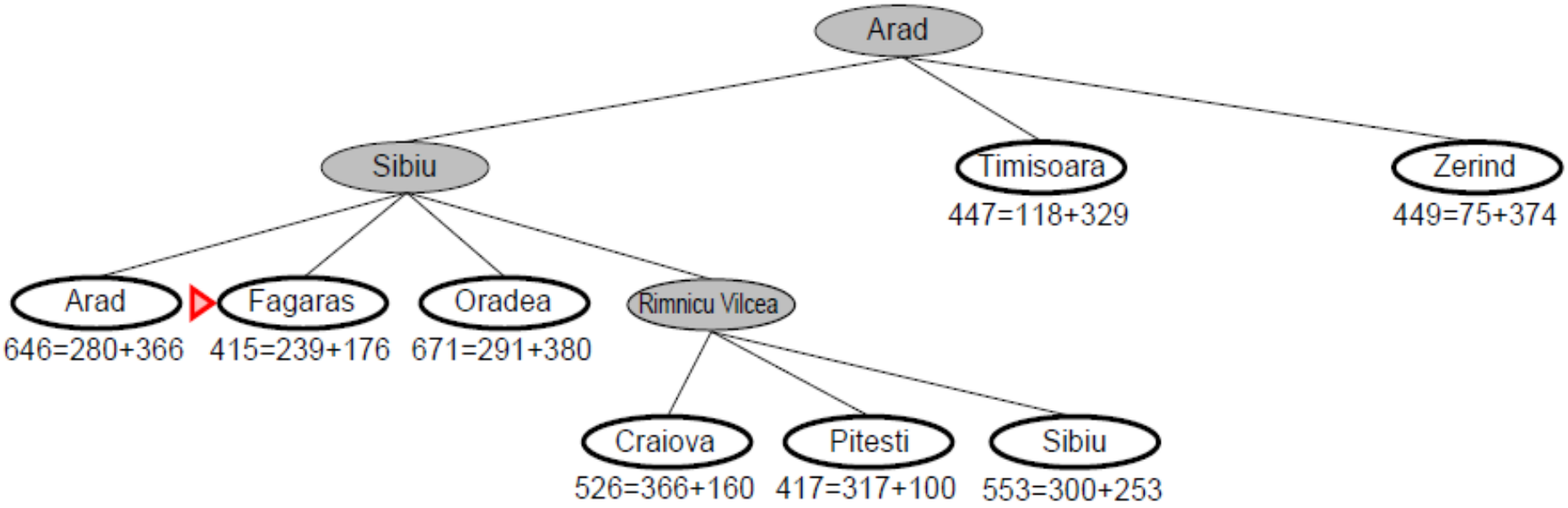
A* search example



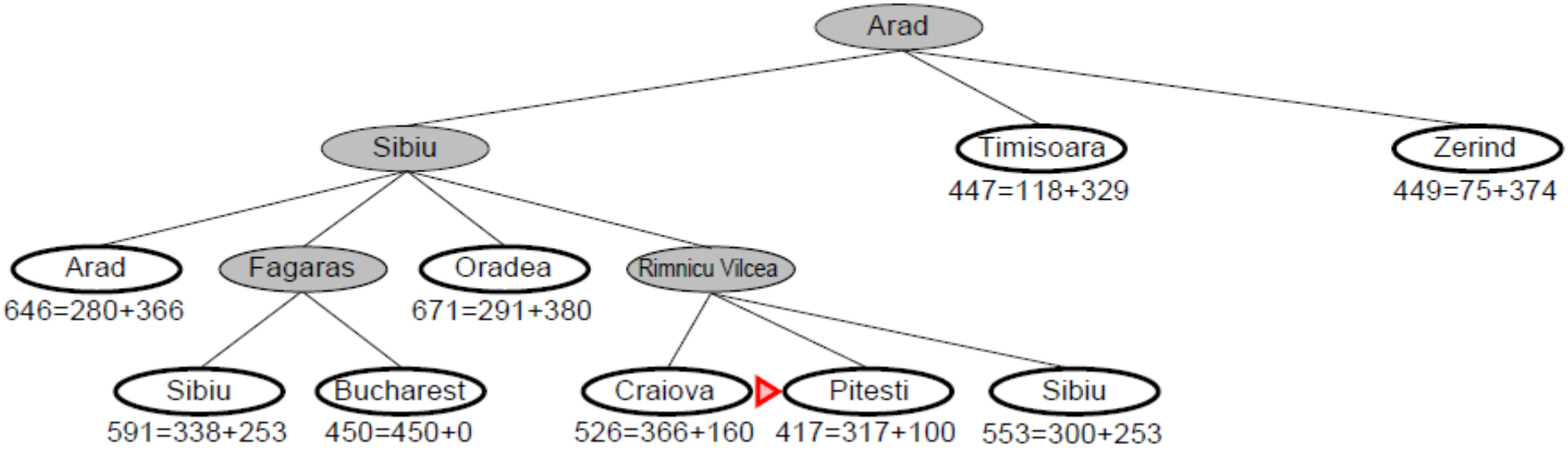
A* search example



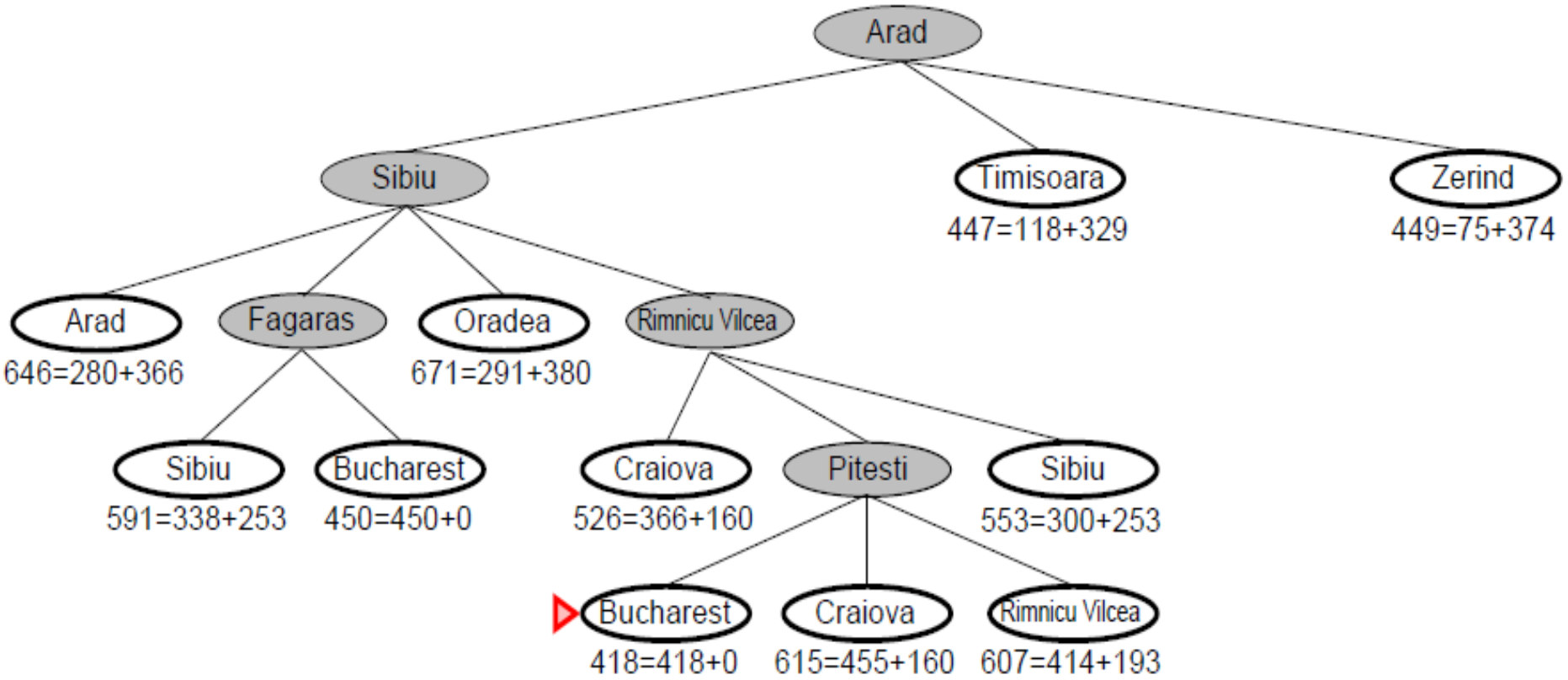
A* search example



A* search example



A* search example



```
A* (Start, Goal)
{
        CLOSE={ }
        OPEN= { root }
        g[ root ] = 0
        f [ root ]
}
```