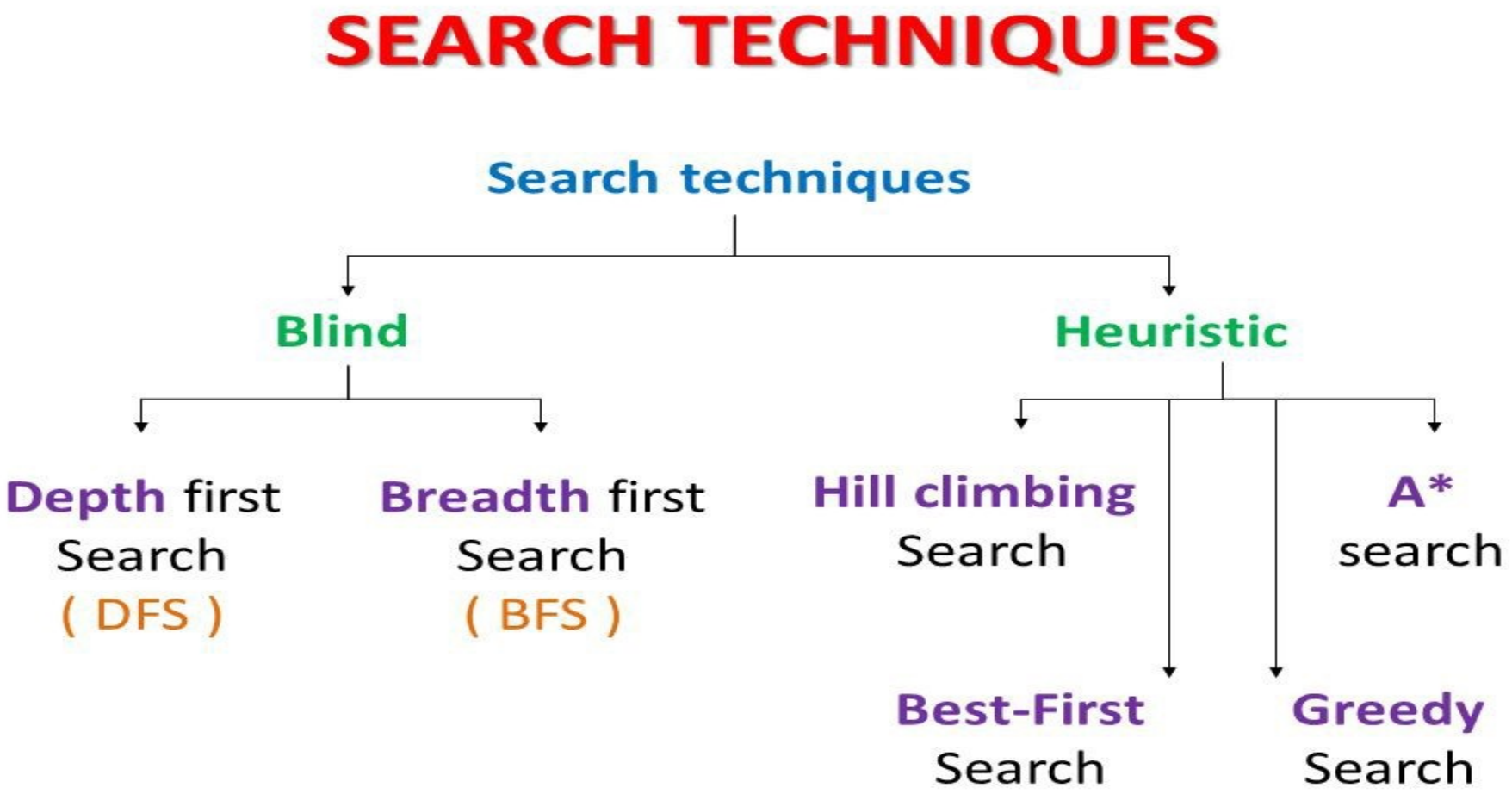
Search Techniques

- Book : Artificial Intelligence, Third Edition, Tata-Mc Graw Hill Author(s): E.Rich & K. Knight (Chapter - 3)
- https://www.javatpoint.com/search-algorithms-in-ai

Search Techniques

- **Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
  - **Search Space:** Search space represents a set of possible solutions, which a system may have.
  - **Start State:** It is a state from where agent begins **the search**.
  - **Goal state:** It is a function which observe the current state and returns whether the goal state is achieved or not.

Search Techniques



Uninformed/Blind Search

- Uninformed search does not contain any domain knowledge such as closeness, the location of the goal.
  - It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes.
  - Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search.
  - It examines each node of the tree until it achieves the goal node.

Informed/Heuristic Search

- Informed Search algorithms use domain knowledge.
- In an informed search, problem information is available which can guide the search.
- Informed search strategies can find a solution more efficiently than an uninformed search strategy.
- Informed search is also called a **Heuristic search.**
- **A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.**

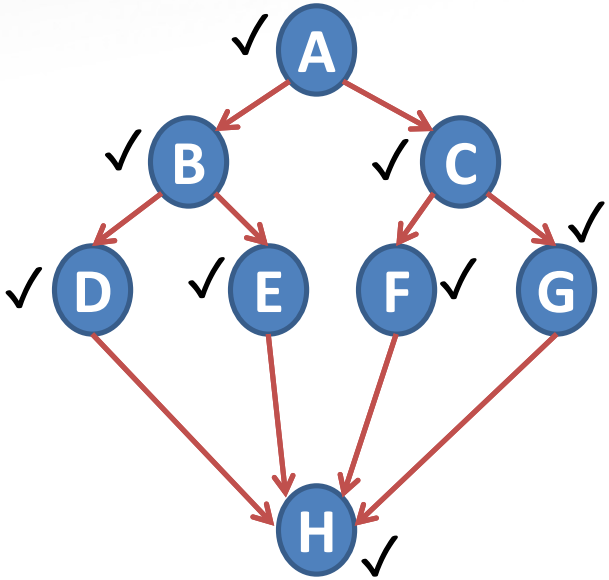
BFS - Breadth-first Search

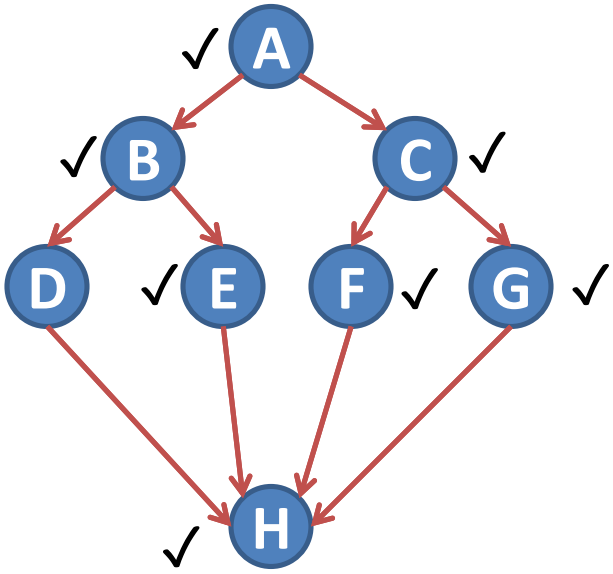
- BFS starts searching from the root node of the tree and expands all successor node **level by level.**
- BFS implemented using **FIFO queue data structure.**
- BFS will provide a solution if any solution exists.
- It **requires lots of memory** since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

DFS - Depth-first Search

- DFS is a recursive algorithm for traversing a tree or graph data structure.
- DFS starts from the root node and **follows each path to its greatest depth node before moving to the next path.**
- DFS uses a **stack data structure** for its implementation.
- DFS requires **very less memory** as it only needs to store a stack of the nodes on the path from root node to the current node.
- There is **no guarantee of finding the solution**.

BFS - DFS





Activity

- Reading – Writing
- Write Difference between DFS and BFS.

Heuristic Search

- Every search process can be viewed as a traversal of a directed graph.
- Nodes represent problem states and the arcs represent relationships between states.
- Domain-specific knowledge must be added to improve search efficiency.
- The Domain-specific knowledge about the problem includes the nature of states, cost of transforming from one state to another, and characteristics of the goals.
- This information can often be expressed in the form of Heuristic Evaluation Function.

Heuristic Search

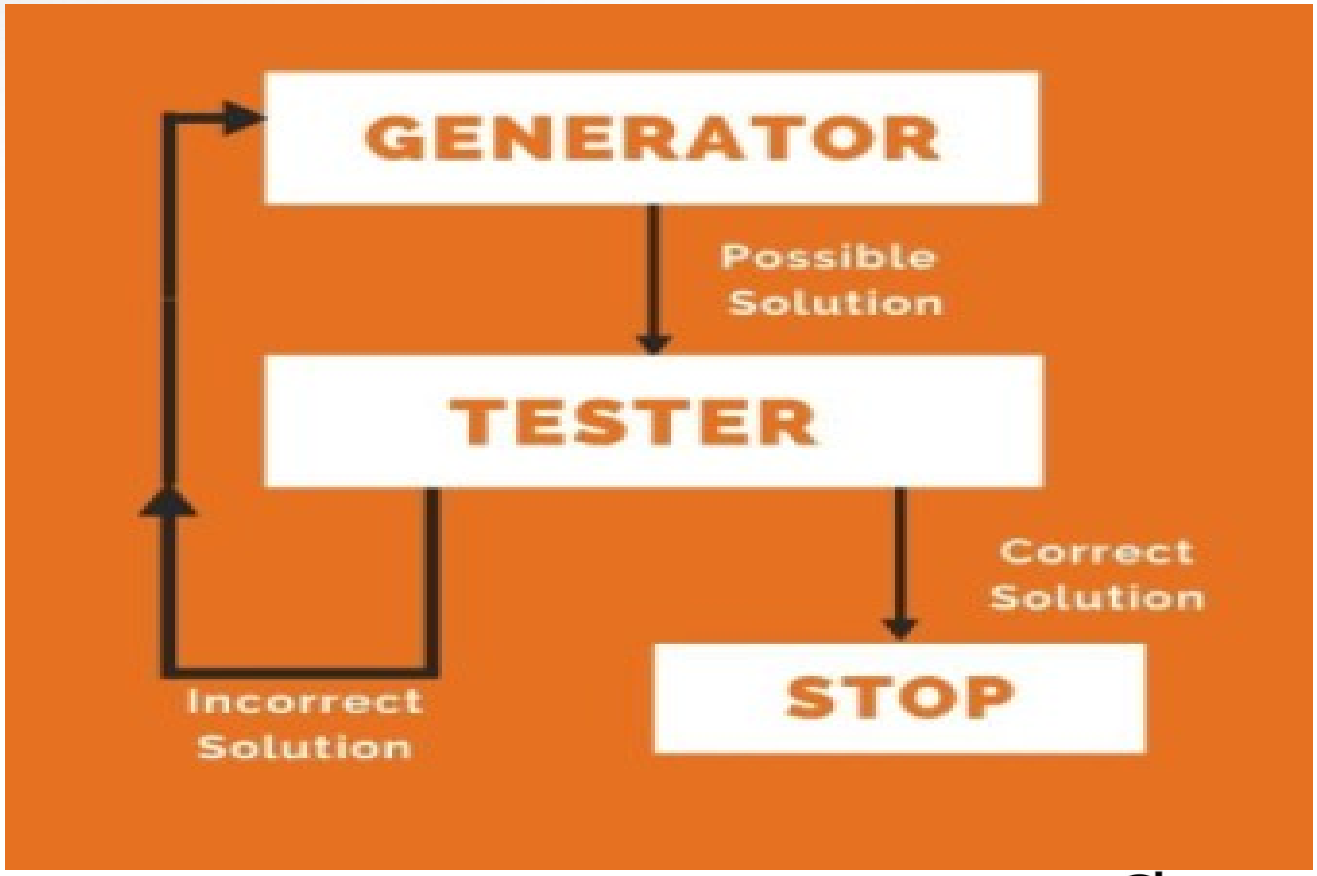
- Some prominent intelligent search algorithms are stated below:
  - Generate and Test Search
  - Best-first Search
  - A* Search
  - Constraint Search
  - Means-ends analysis

Generate and Test Search

- Its Heuristic search technique.
- Its DFS with Backtracking
- It is also known as British Museum algorithm.

```
Algorithm: Generate-And-Test
1.Generate a possible solution.
2.Test to see if this is the expected solution.
3.If the solution has been found quit else go to step 1.
```

Generate and Test Search



Best First Search

- Best first search uses the concept of a priority queue and heuristic search.
- It is a search algorithm that works on a specific rule.
- The aim is to reach the goal from the initial state via the shortest path.
- Best first search combines the advantages of both DFS and BFS into a single method.
- DFS is good because it allows a solution to be found without expanding all competing branches. BFS is good because it does not get trapped on dead end paths.

Best First Search

- One way of combining BFS and DFS is to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one does.
- At each step of the Best First Search process, we select the most promising of the nodes we have generated so far.
- We then expand the chosen node by using the rules to generate its successors.
- If one of them is a solution, we can quit. If not, all those new nodes are added to the set of nodes generated so far.

Best First Search - Algorithm

- we assume two different list of nodes:
- OPEN list → is the list of nodes which have been discovered but yet not expanded(whose children/successors are yet not discovered)
-
- CLOSED list → nodes whose successors are also generated.
- f(n) = h (n) where h(n) is estimated cost from current node to the goal node

Best First Search Algorithm

Step 1: Place the starting node into the OPEN list.

Step 2: If the OPEN list is empty, Stop and return failure.

Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.

Step 4: Expand the node n, and generate the successors of node n.

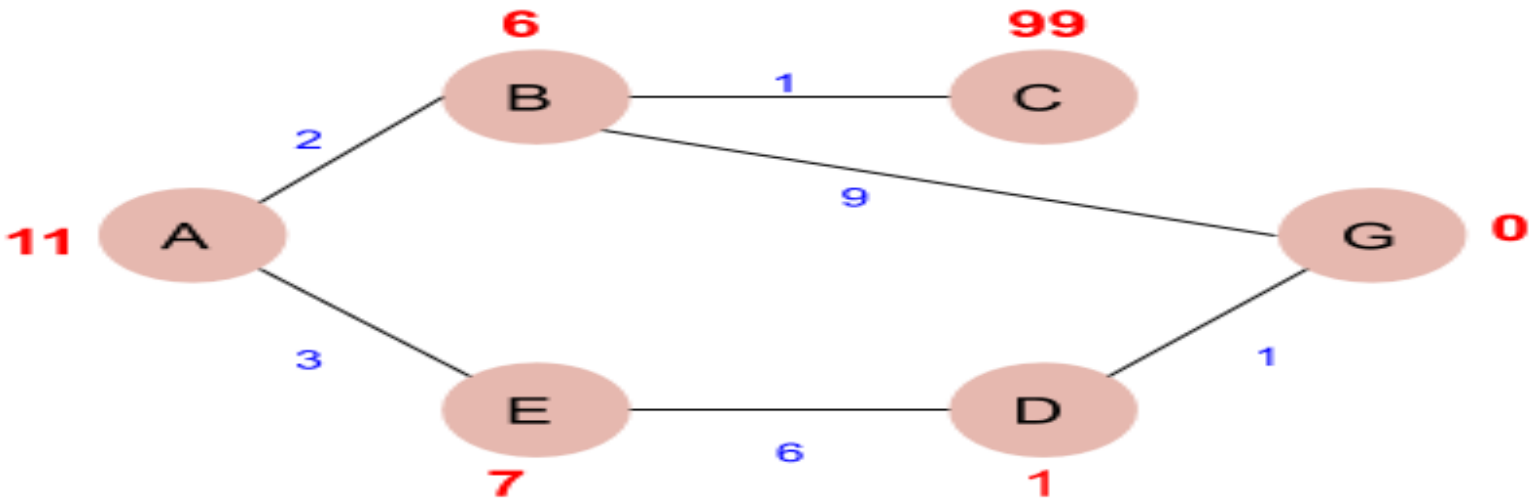
Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.

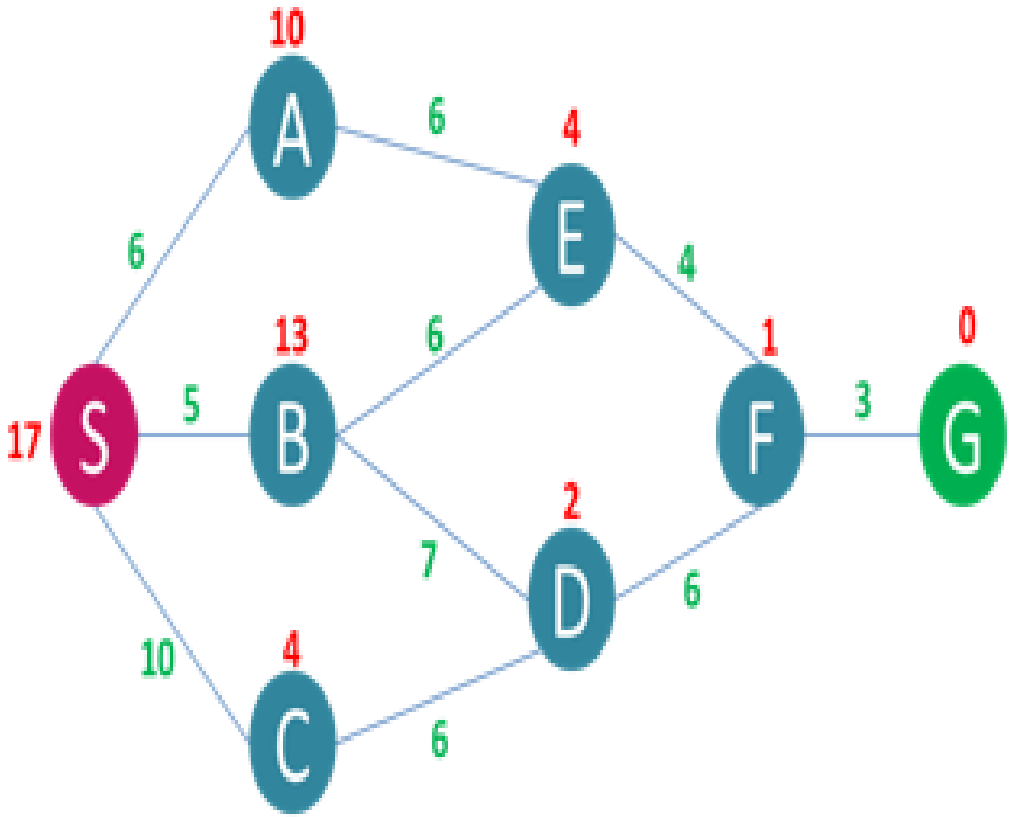
Step 7: Return to Step 2.

Best First Search - Example

- Demonstrate Best First search for following graph.
- Path = A – B – G and Total cost = 11



Example : Find Shortest path using best first search

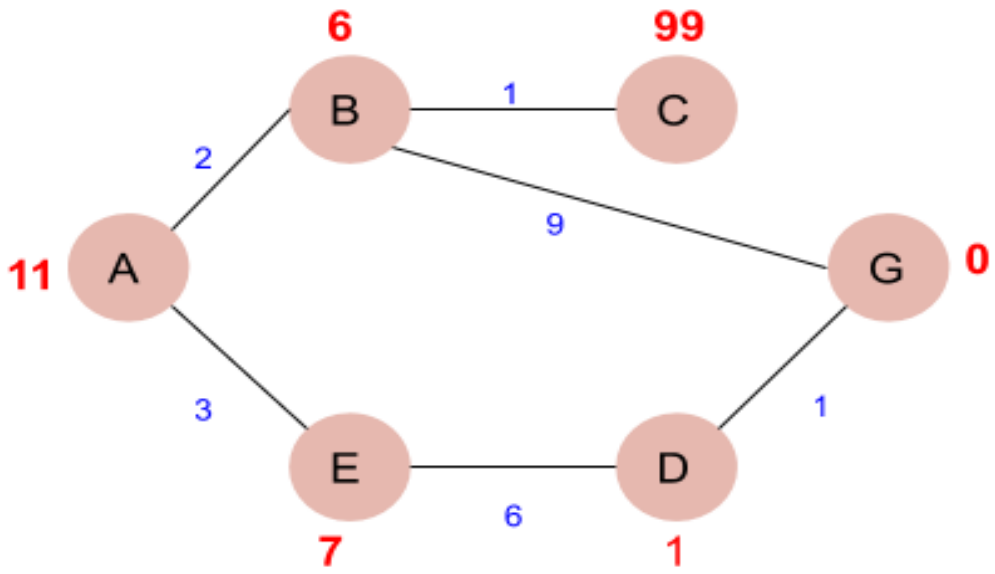


Path = S→C→D→F→G

Total Cost = 25

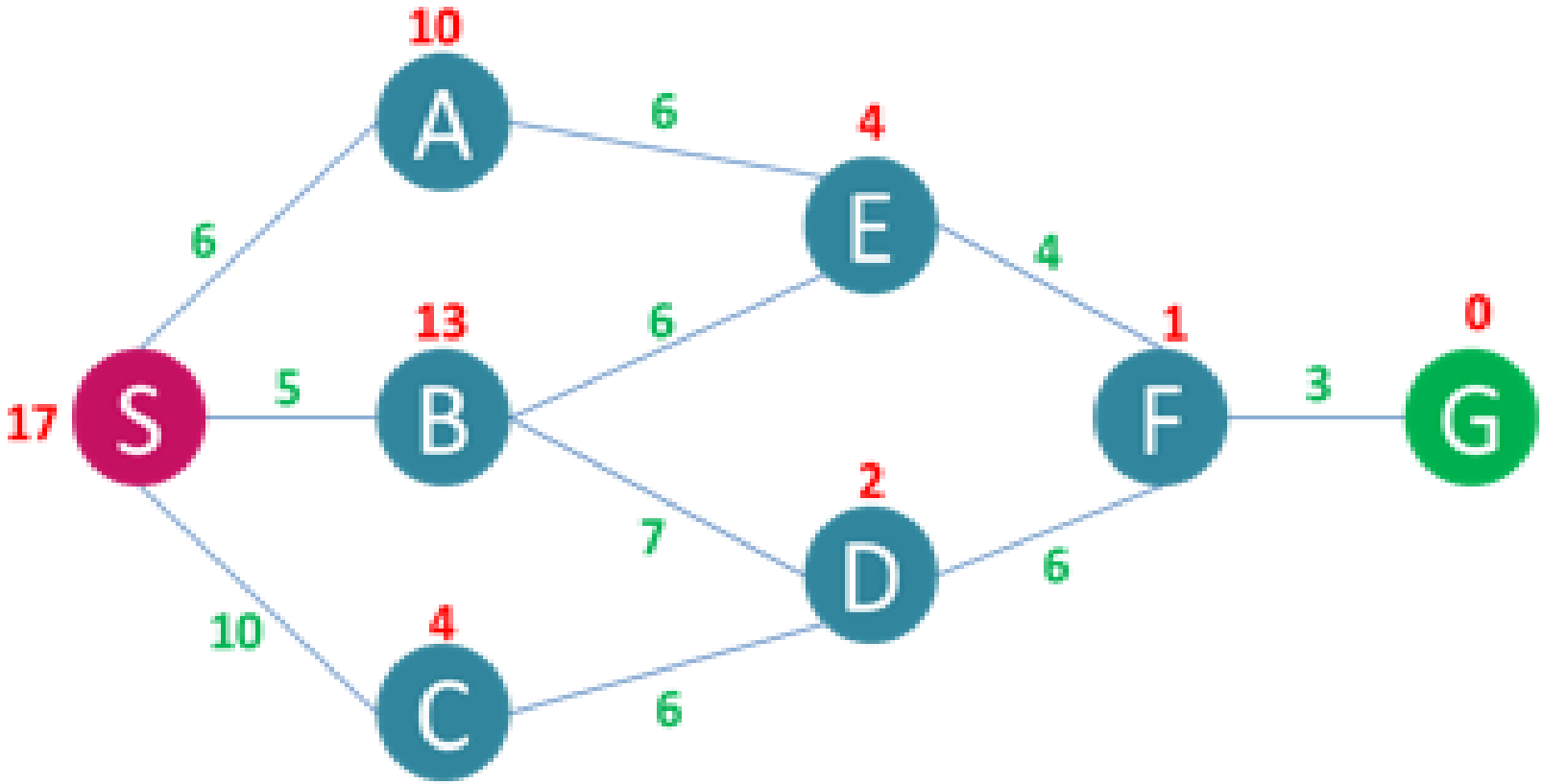
A* Best First Search

- A* algorithm works based on heuristic methods and this helps achieve optimality.
- f(n) = g(n) + h(n)
- Path = A-E-D-G

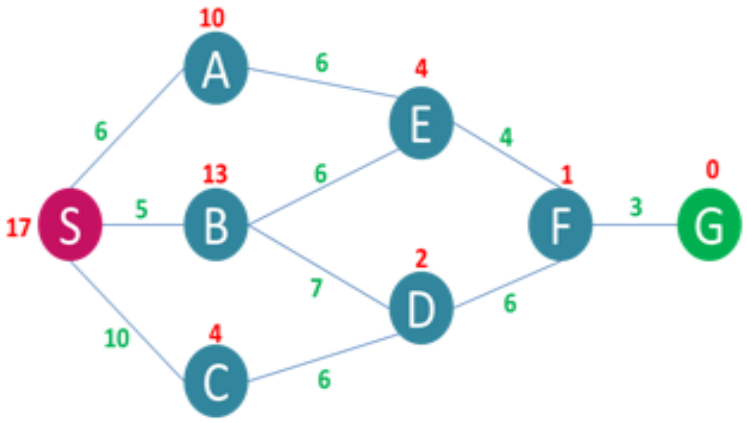


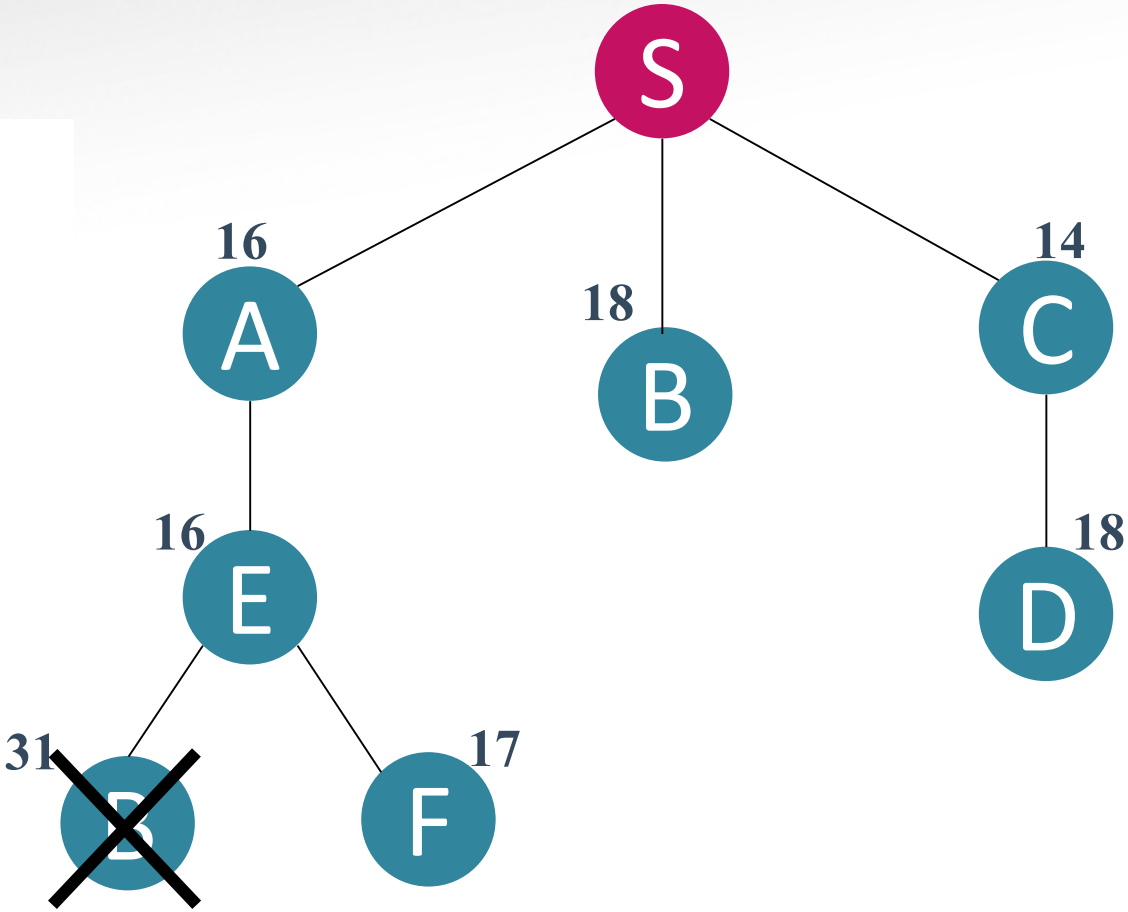
Change at RKU

Example : Find Shortest path using A* Best First Search

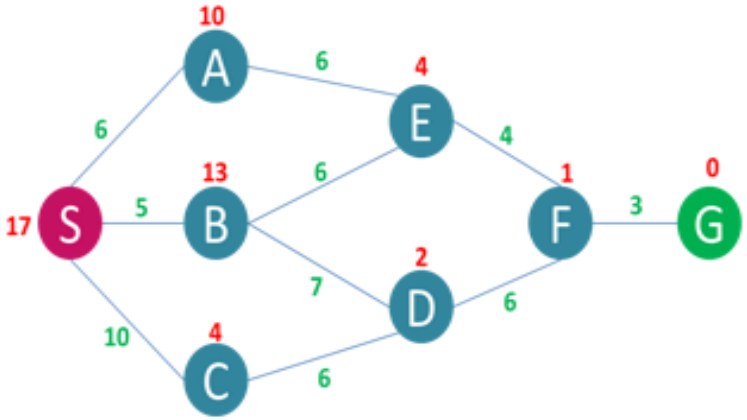


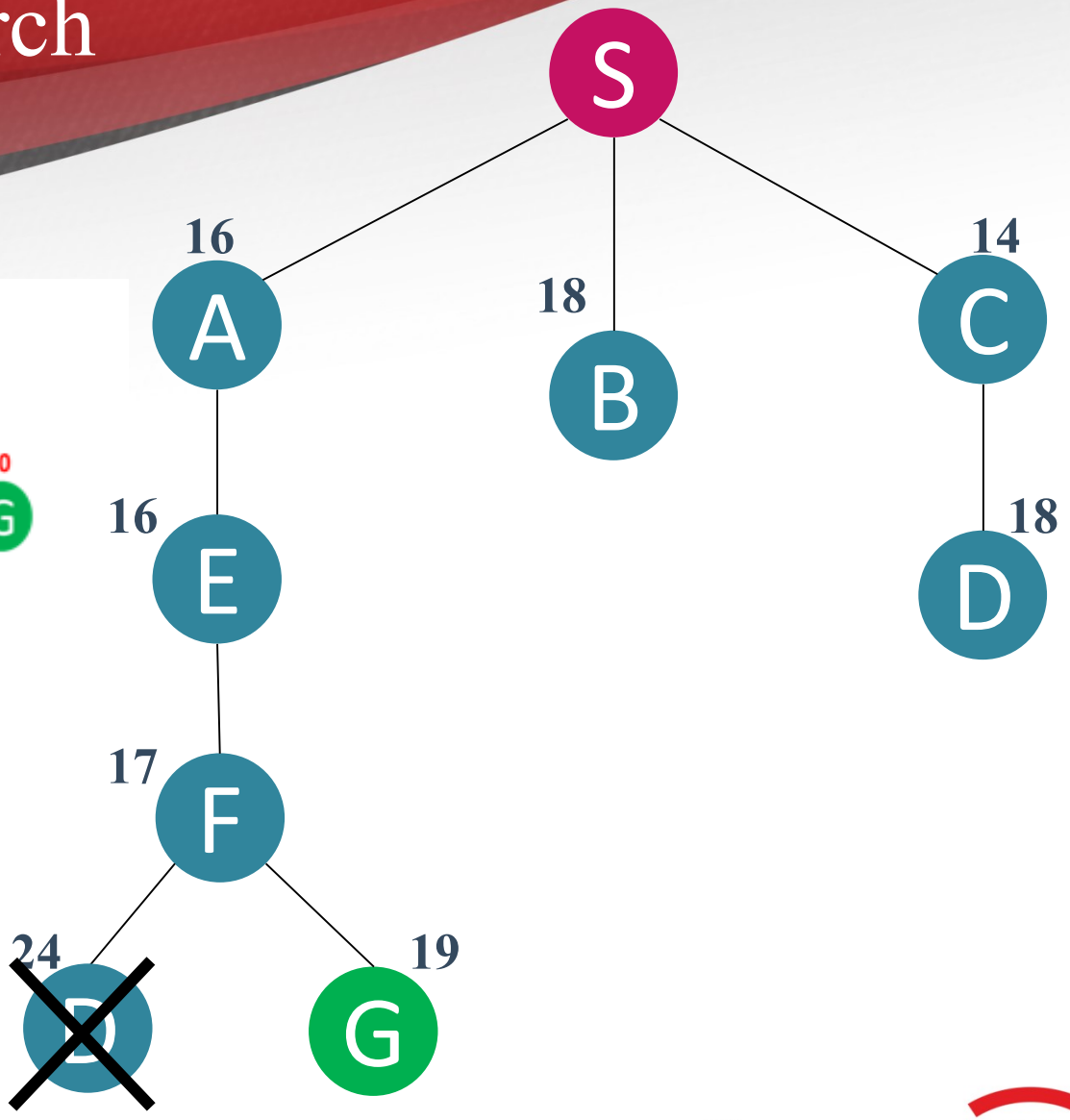
Example : Find Shortest path using A* Best First Search



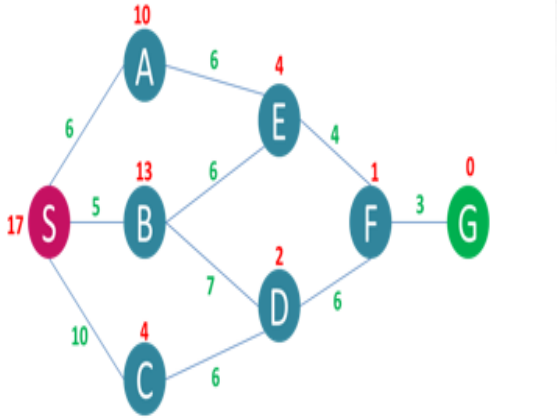


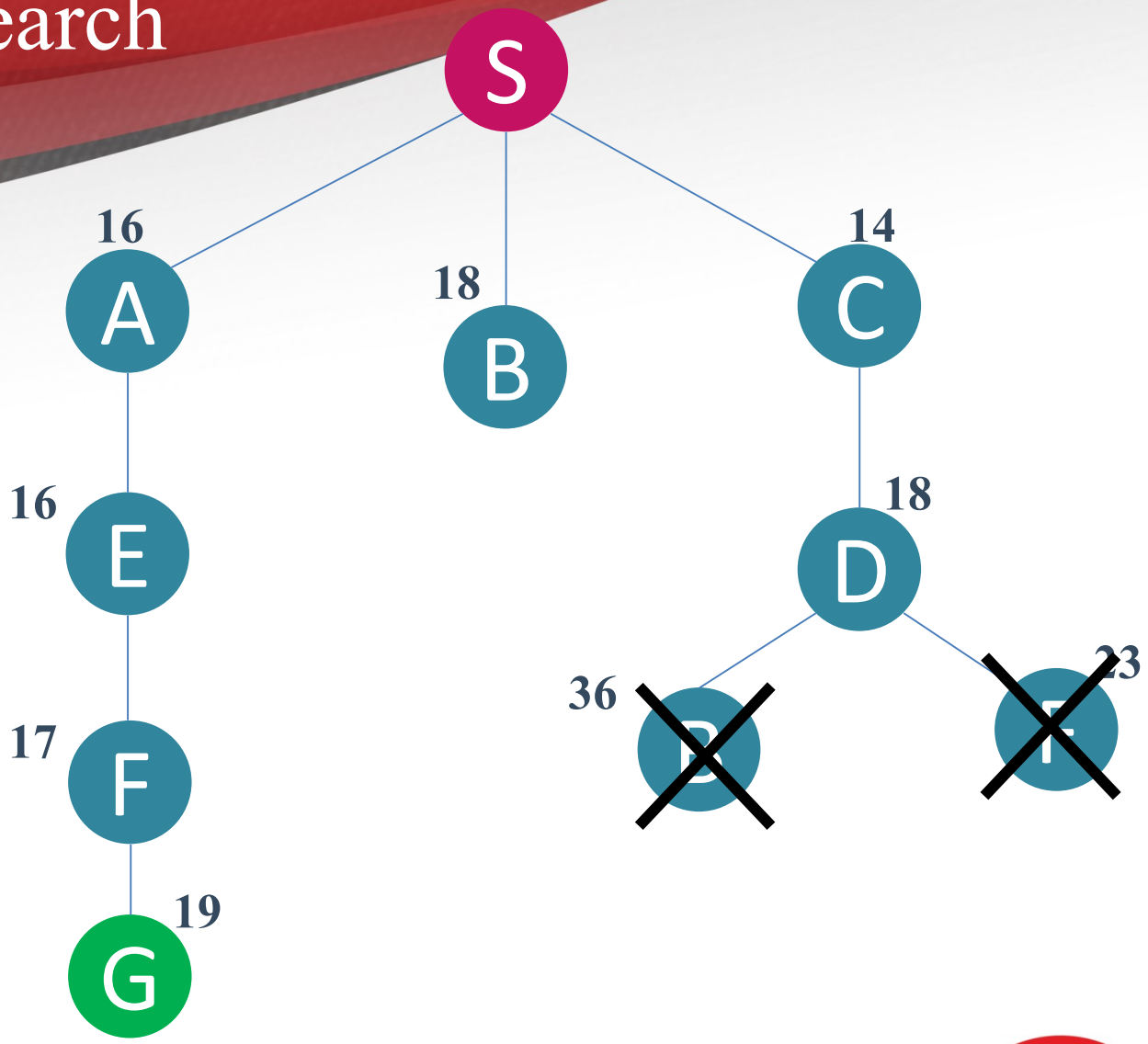
Example : Find Shortest path using A* Best First Search



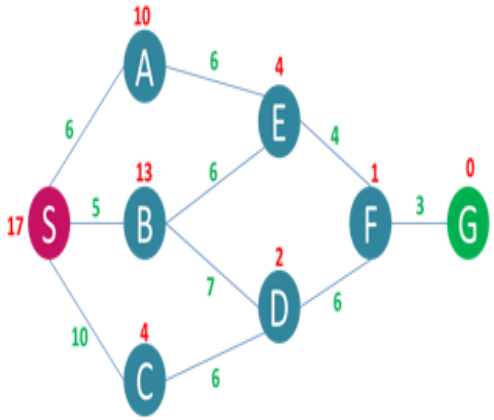


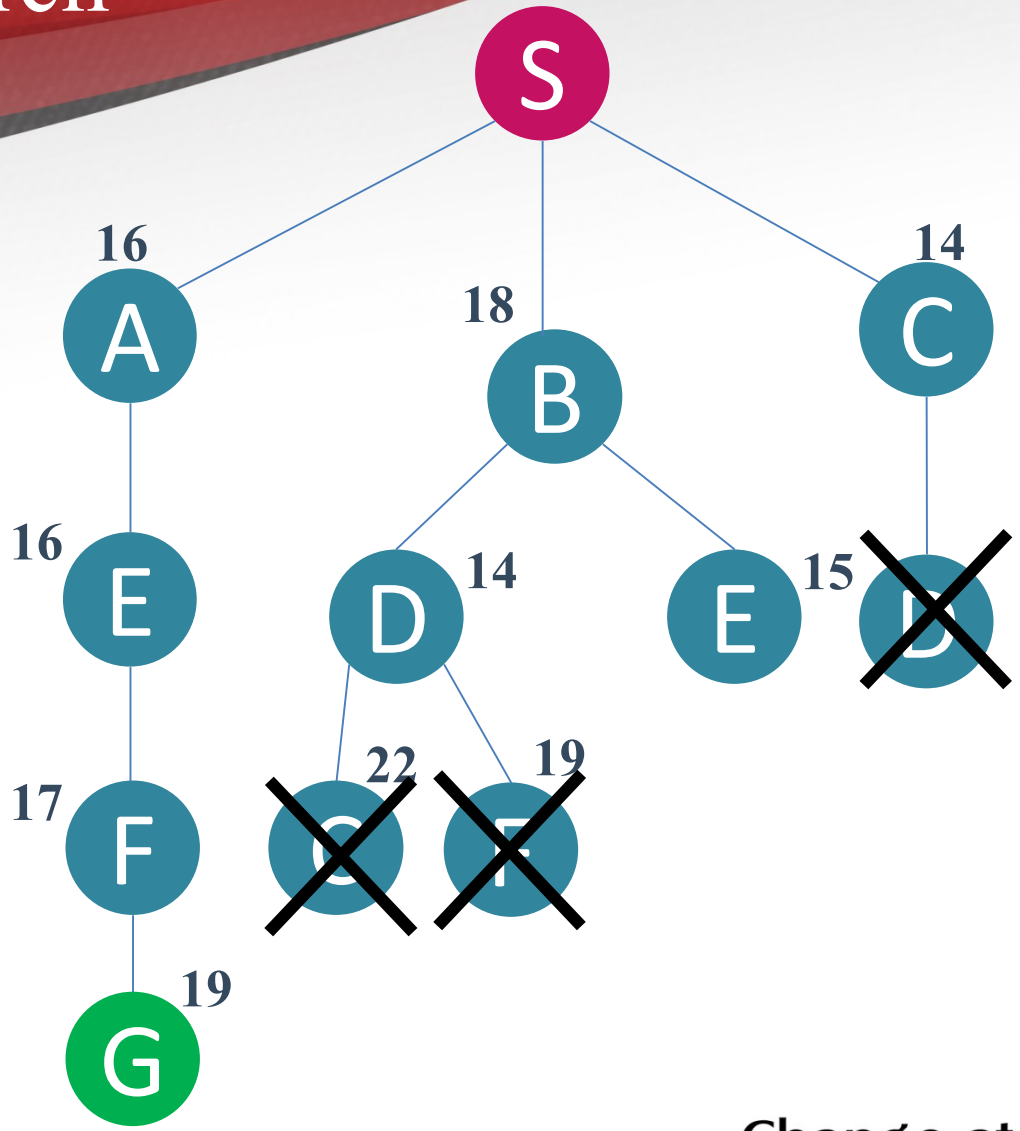
Example : Find Shortest path using A* Best First Search



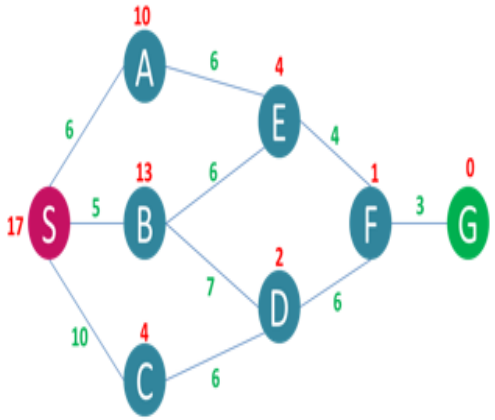


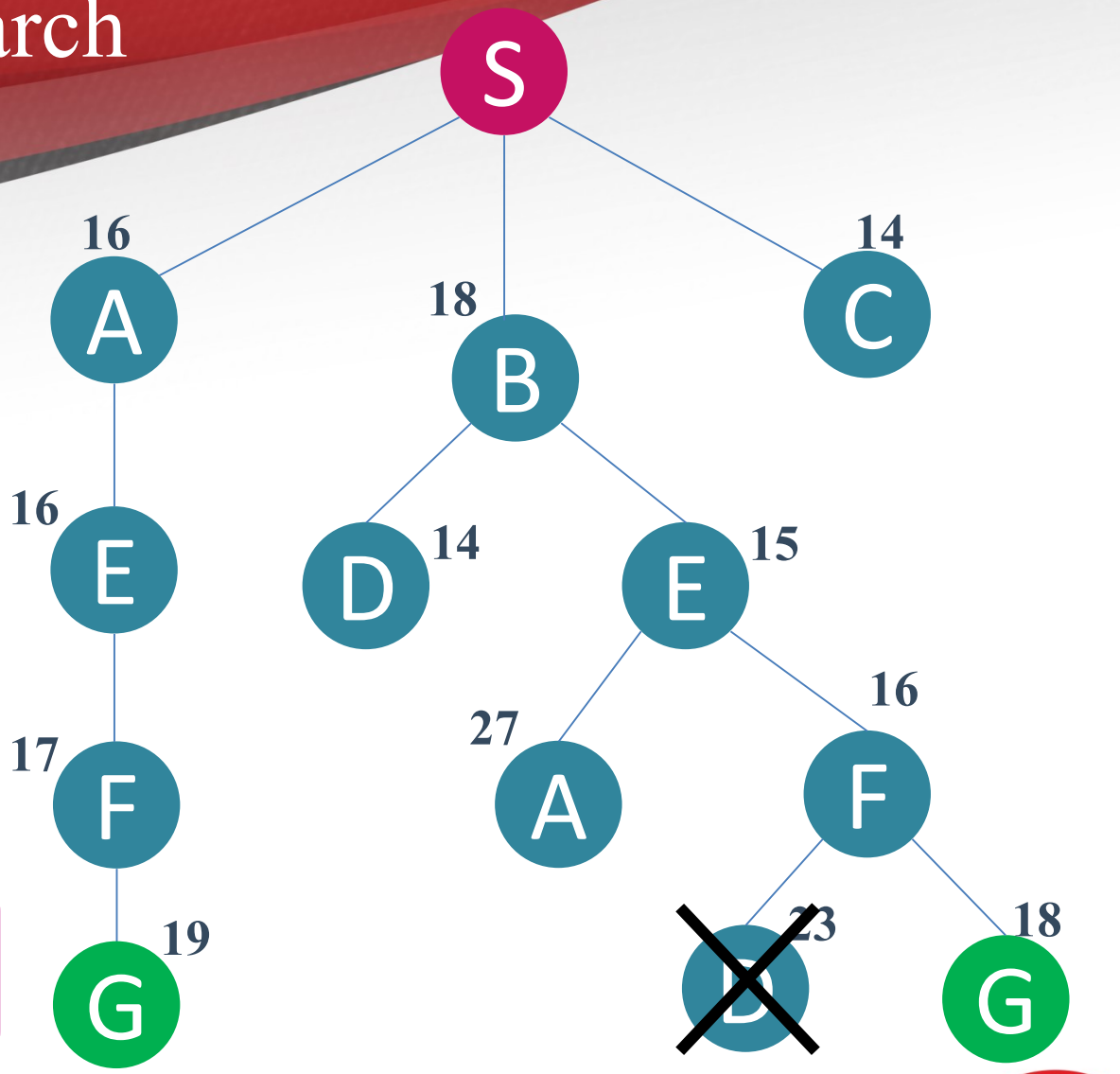
Example : Find Shortest path using A* Best First Search





Example : Find Shortest path using A* Best First Search





$S \rightarrow B \rightarrow E \rightarrow F \rightarrow G$

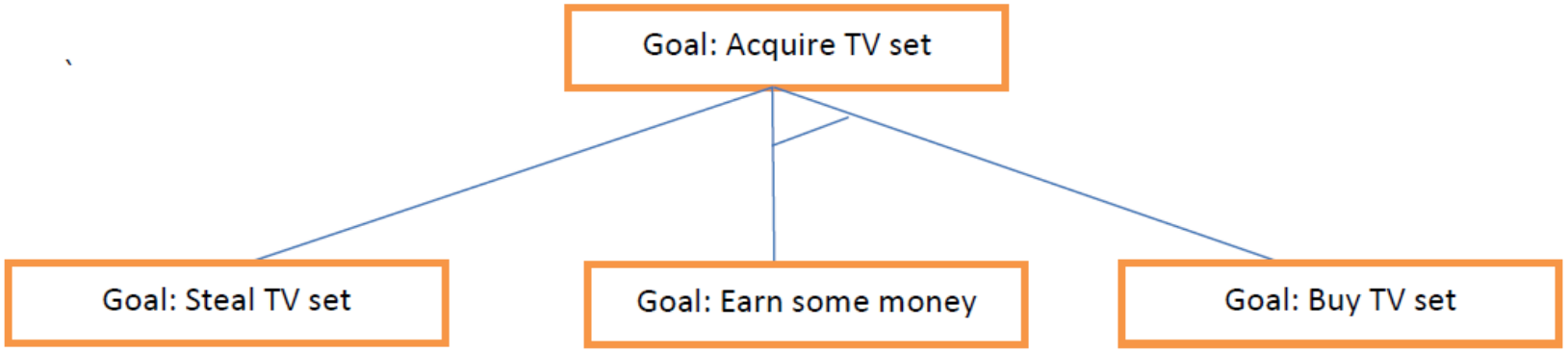
Total Cost = 18

A* Algorithm

- Step1: Place the starting node in the OPEN list.
- Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
- Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
- Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.
- Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
- Step 6: Return to Step 2.

Problem Decomposition

- When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution.
- It use AND-OR graph to find more than one solution.



A SIMPLE AND-OR GRAPH

AO* Algorithm

- AO* is informed search based on BFS.
- It is based on problem decomposition.
- It use AND-OR graph to find more than one solutions.
- AND-OR graph is useful for representing the solution of problems that can be solved by decomposing them into set of smaller problems.
- All of which must be solved.

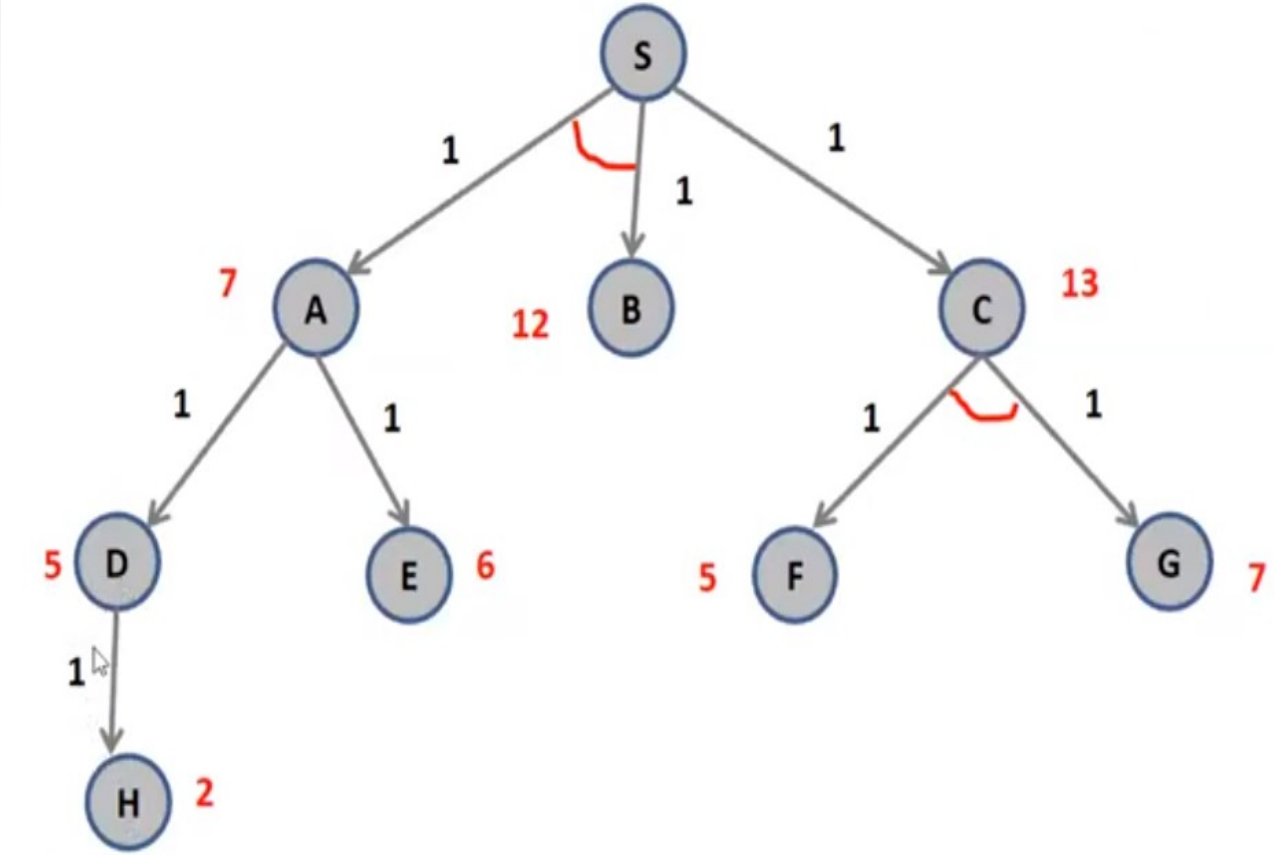
AO* Algorithm

- Node in AO* algorithm will point both down to its successors and up to its parent nodes. (backtracking).
- Each node in graph will have heuristics value associated with it.
- Cost function $f(n) = g(n) + h(n)$
- $g(n)$ = actual cost / edge cost
- $h(n)$ = heuristics cost
- Here assume $g(n) = 1$

AO* Algorithm – Example-1

Cost = Min (18,15)

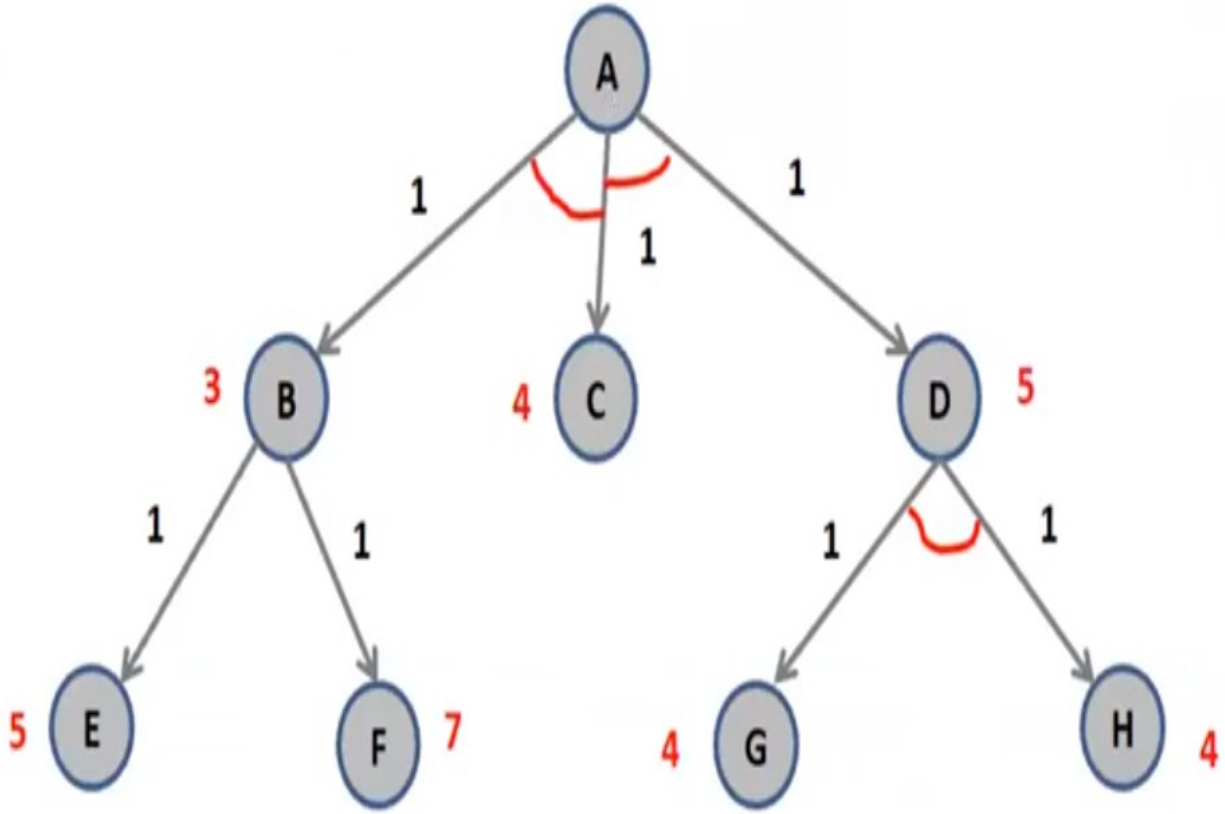
= 15



AO* Algorithm – Example-2

Cost = Min (12,16)

= 12



AO* Algorithm

1. Initialise the graph to start node
2. Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved
3. Pick any of these nodes and expand it and if it has no successors call this value *FUTILITY* otherwise calculate only *f'* for each of the successors.
4. If *f'* is 0 then mark the node as *SOLVED*
5. Change the value of *f'* for the newly created node to reflect its successors by back propagation.
6. Wherever possible use the most promising routes and if a node is marked as *SOLVED* then mark the parent node as *SOLVED*.
7. If starting node is *SOLVED* or value greater than *FUTILITY*, stop, else repeat from 2.

Constraint Satisfaction Problem

CSP - is a problem that requires its solution within some limitations or conditions also known as constraints.

It consists of the following:

1. Variables: which stores the solution (finite set)

   (V = {V1, V2, V3,....., Vn})

2. Domain: set of discrete values known as domain from which the solution is picked (D = {D1, D2, D3,.....,Dn})

3. Constraints: (C = {C1, C2, C3,....., Cn})

Constraint Satisfaction Problem

Example:

- Cryptarithmetic problem
- Crossword puzzle
- Sudoku game
  - fill the empty squares with numbers ranging from 1 to 9 in such a way that no row, column or a block has a number repeating itself.
  - consider the Sudoku problem again. Suppose that a row, column and block already have 3, 5 and 7 filled in. Then the domain for all the variables in that row, column and block will be {1, 2, 4, 6, 8, 9}.

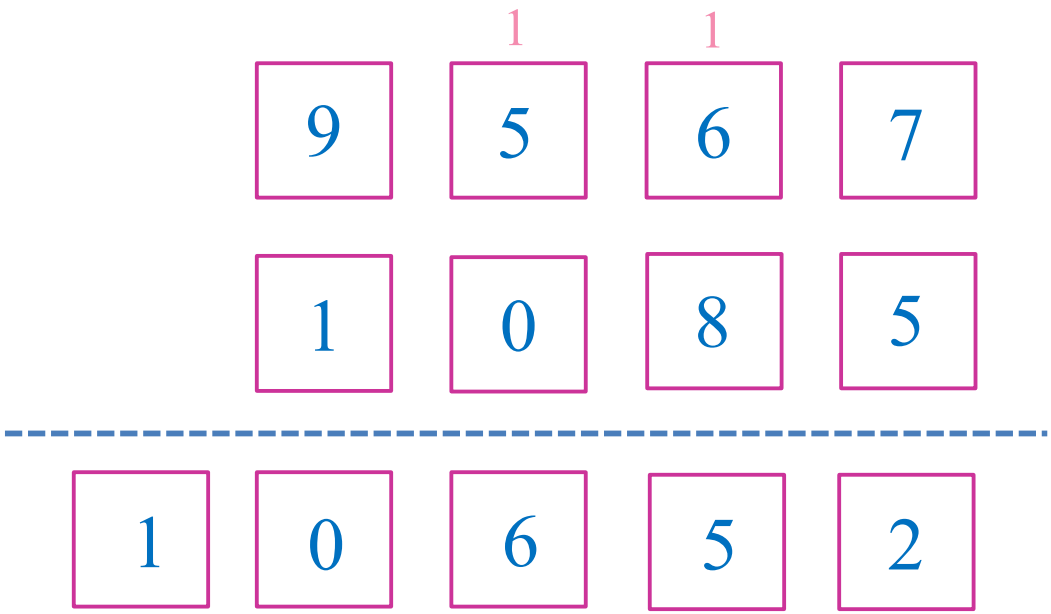
Crypt-arithmetic Puzzle

Initial state:

- Assign values between 0 to 9.
- No two letters have the same value.
- The sum of the digits must be as shown.

```
  S E N D
+ M O R E
---------
M O N E Y
```

S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2



More Examples of Crypt-arithmetic Puzzle

```
  BASE
+BALL
------
 GAMES
------
```

B	7
A	4
S	8
E	3
L	5
G	1
M	9

More Examples of Crypt-arithmetic Puzzle

```
  YOUR
 +YOU
------
 HEART
```

Y	9
O	4
U	2
R	6
H	1
E	0
A	3
T	8

Means-Ends Analysis

- MEA is mixture of the two directions forward / Backward is appropriate for solving a complex and large problem.
- It first solve the major part of a problem and then go back and solve the small problems arise during combining the big parts of the problem. Such a technique is called Means-Ends Analysis.
- The MEA analysis process centered on the evaluation of the difference between the current state and goal state.

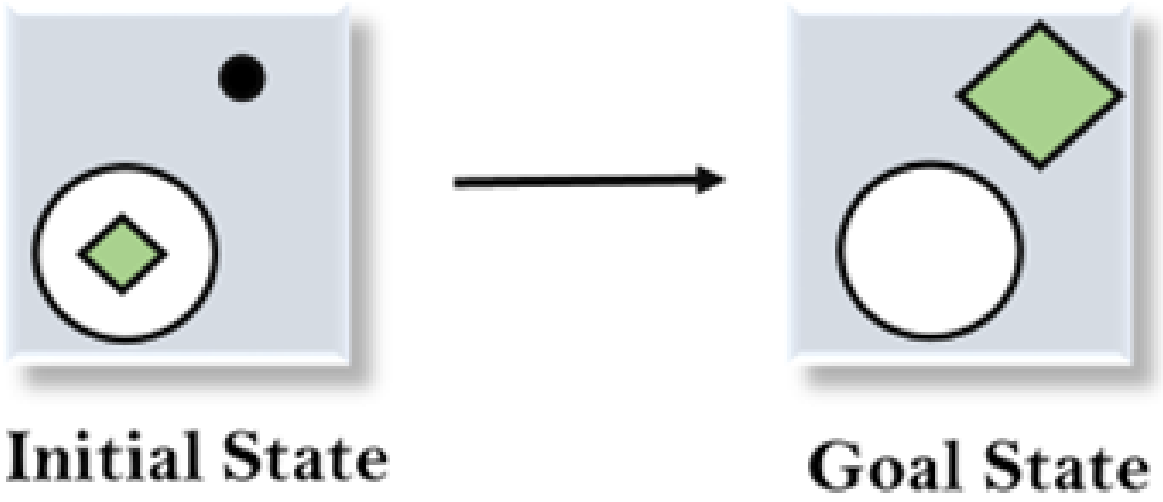
Means-Ends Analysis

- Steps of MEA

1. First, evaluate the difference between Initial State and final State.
2. Select the various operators which can be applied for each difference.
3. Apply the operator at each difference, which reduces the difference between the current state and goal state.

Means-Ends Analysis

- Example

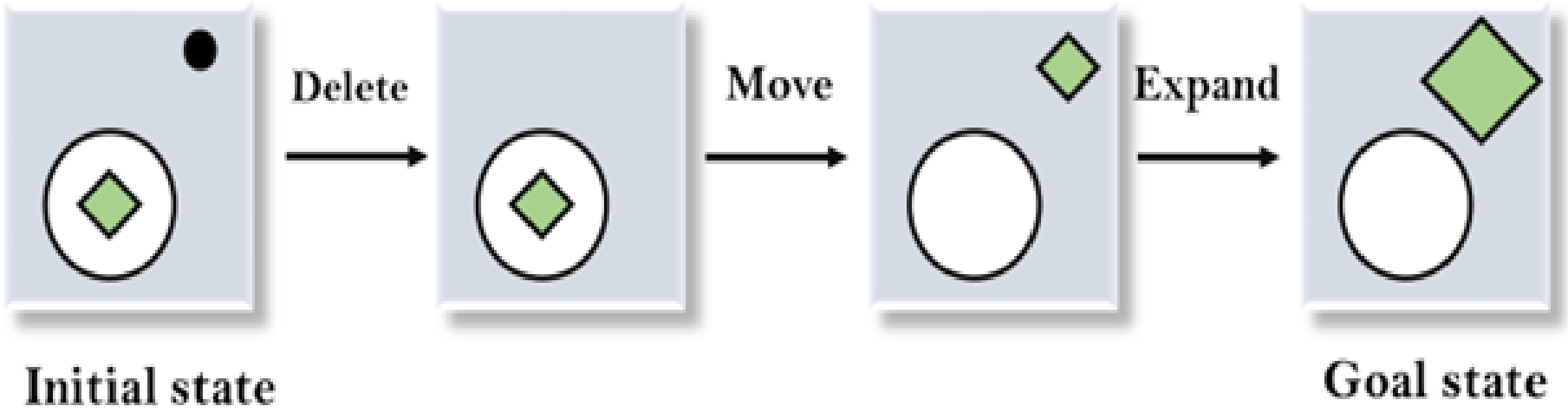


Means-Ends Analysis

- **Solution**
- we will first find the differences between initial states and goal states, and for each difference, we will generate a new state and will apply the operators. The operators we have for this problem are:
- Move
- Delete
- Expand

Means-Ends Analysis

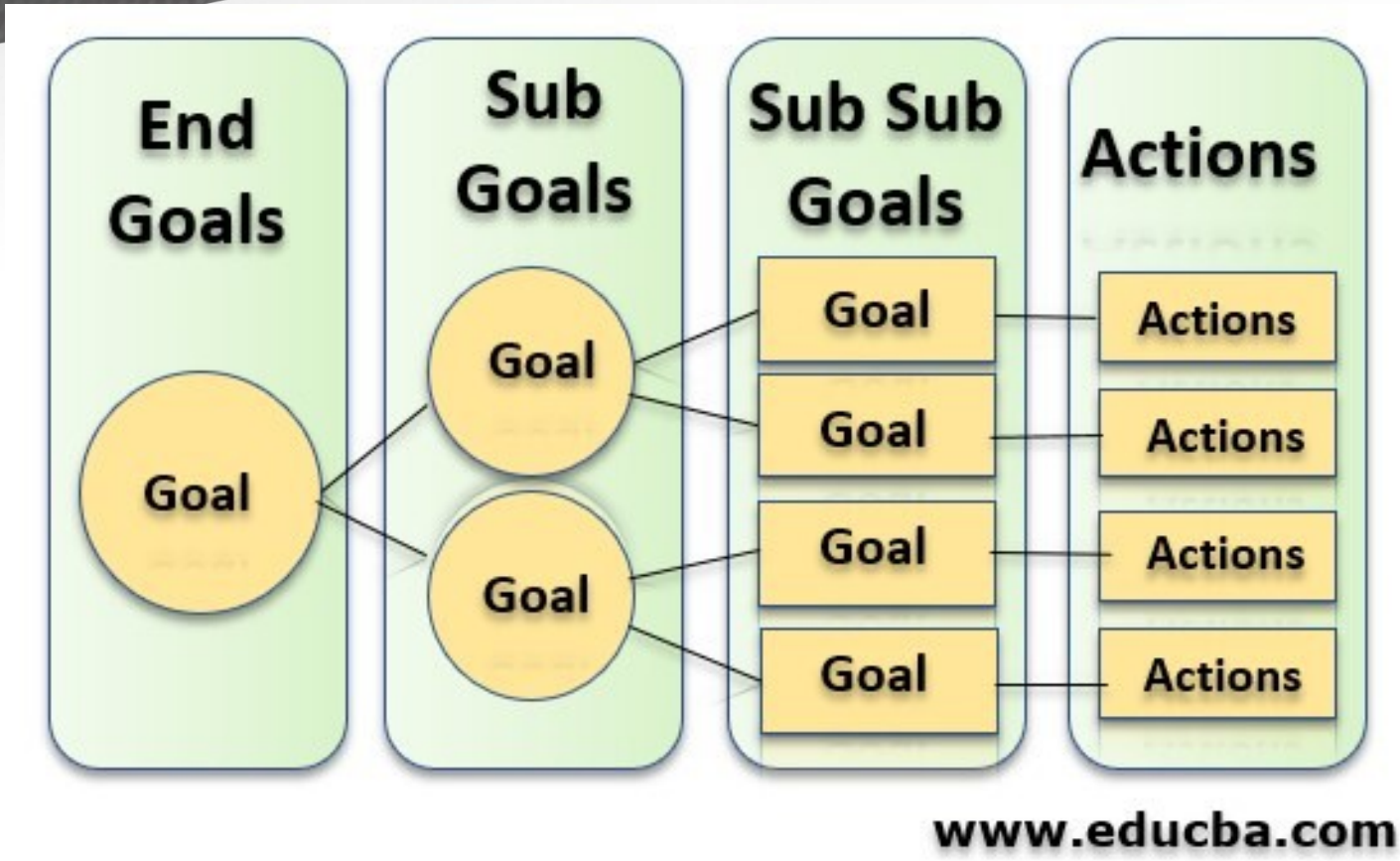
- **Solution**



Means-Ends Analysis

- But sometimes it is possible that an operator cannot be applied to the current state.
- So, we create the sub-problem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called Operator **Subgoaling.**

Means-Ends Analysis



Hill Climbing Algorithm

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing value to find the best solution to the problem.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.

Features of Hill Climbing Algorithm

- **Generate and Test variant: I**t takes the feedback from the test procedure. Then this feedback is utilized by the generator in deciding the next move in search space.

- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.

- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

Types of Hill Climbing

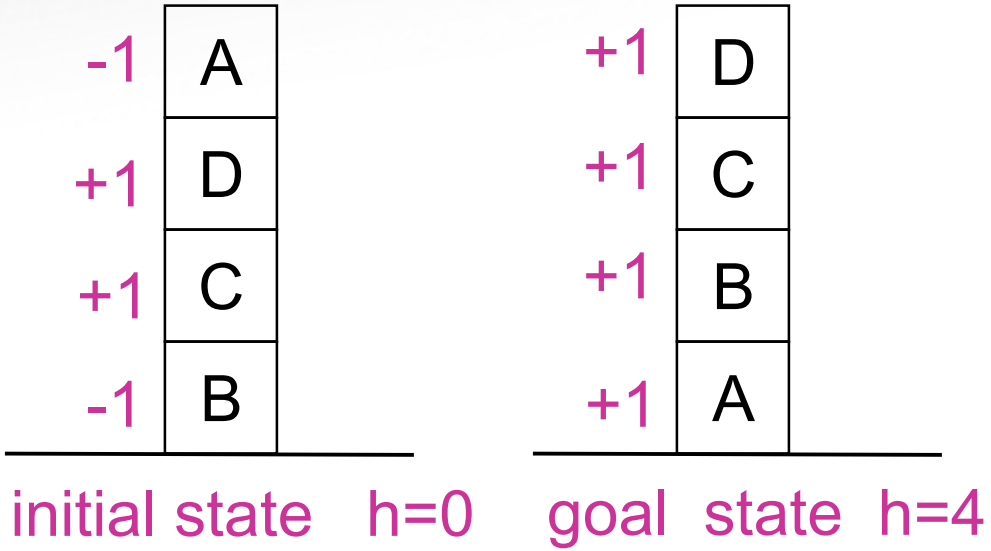
1. **Simple Hill climbing:** It examines the neighboring nodes one by one and selects the first neighboring node which optimizes the current cost as the next node.
2. **Steepest-Ascent Hill climbing:** It first examines all the neighboring nodes and then selects the node closest to the solution state as of the next node.

In simple hill climbing, the **first closer node** is chosen, whereas in steepest ascent hill climbing all successors are compared and the **closest to the solution** is chosen.

Block World problem using Simple Hill Climbing

- **World consists of**
- A flat surface such as a table.
- A set of identical blocks which are identified by letters.
- The blocks can be stacked one on one to form towers.
- Stacking is achieved using a robot arm which has fundamental operations and states.
- The robot can hold one block at a time and only one block can be moved at a time.

Block World problem using Simple Hill Climbing

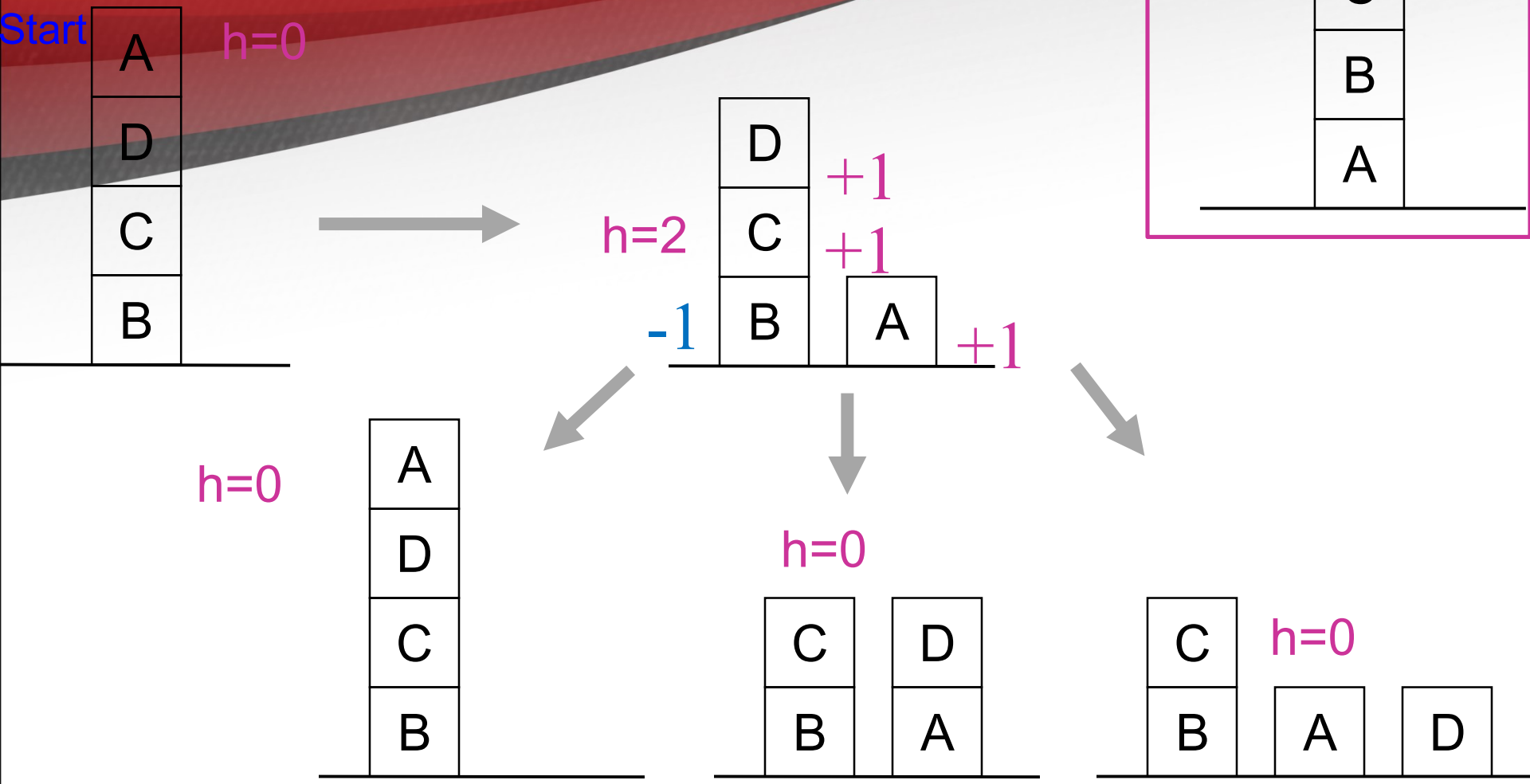


Local heuristic:

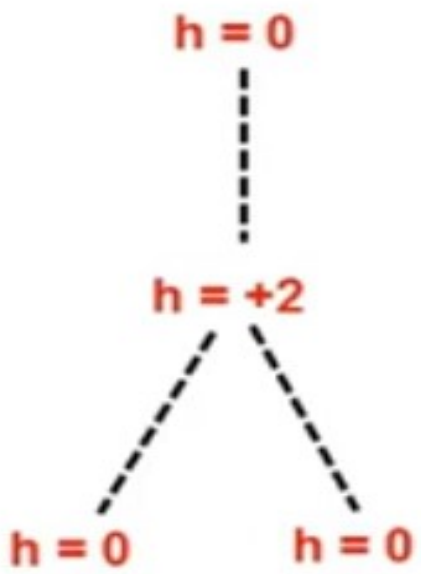
+1 for each block that is resting on the thing it is supposed to be resting on

-1 for each block that is resting on a wrong thing.

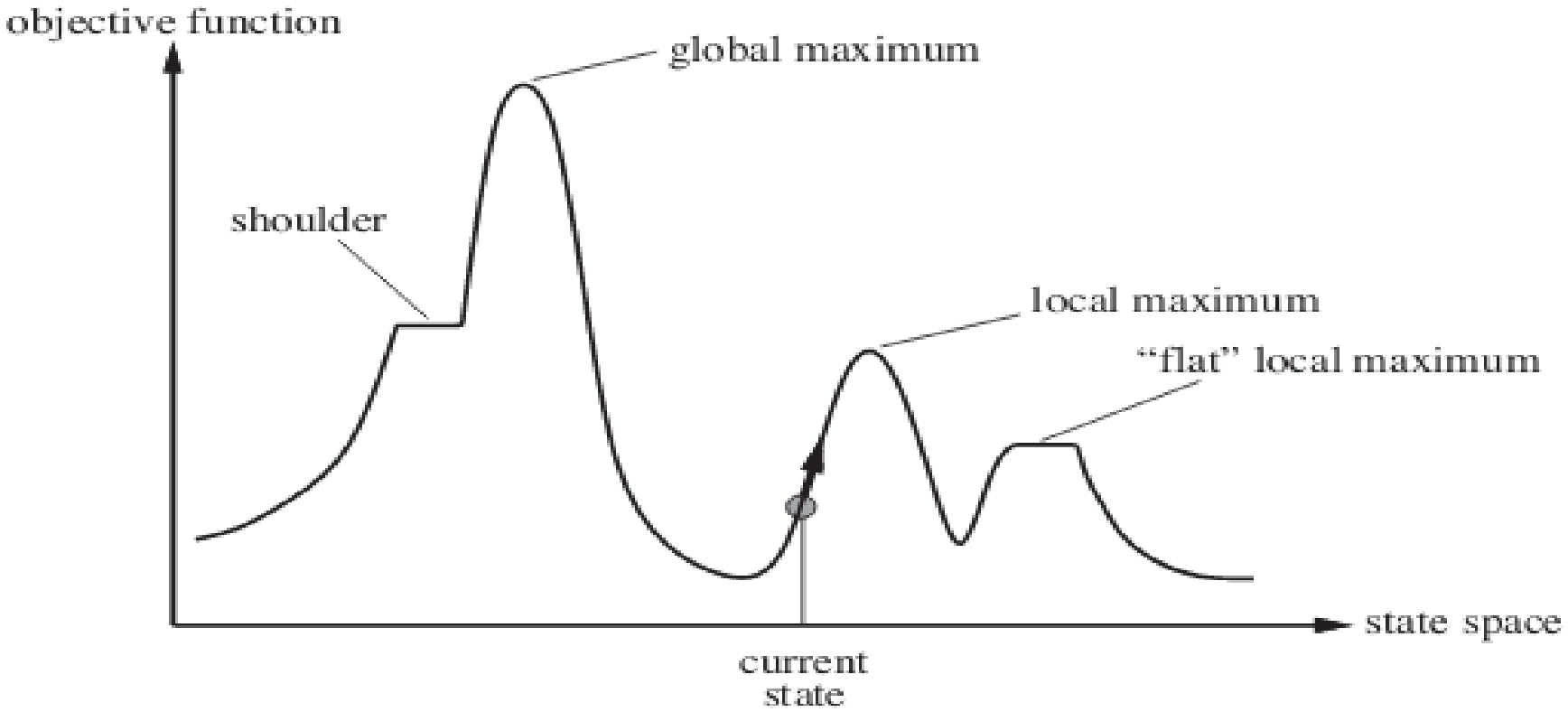
Block World problem using Simple Hill Climbing



Block World problem using Hill Climbing – Local Maxima



Drawbacks of Hill Climbing



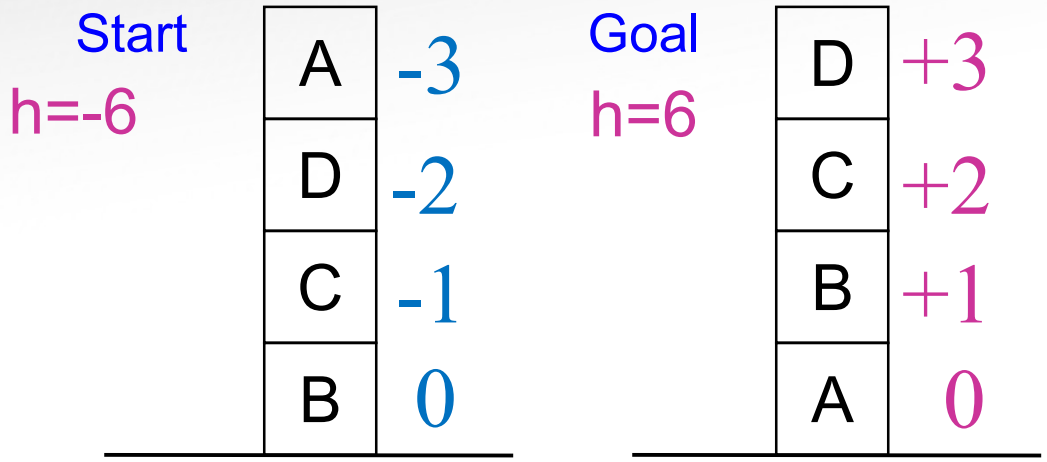
Drawbacks of Hill Climbing

- Local Maxima: a local maximum is a state that is better than all its neighbors but is not better than some other states further away.
- To overcome local maximum problem: Utilize backtracking technique. Maintain a list of visited states and explore a new path.

- Plateau (Flat): a plateau is a flat area of the search space in which, a whole set of neighboring states have the same values.
- To overcome plateaus: Make a big jump. Randomly select a state far away from current state.

- Ridge: is a special kind of local maximum. It is an area of the search space that is higher than surrounding areas and that itself has slop.
- To overcome Ridge: In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

Block World problem using Steepest-Ascent Hill climbing



Global heuristic:

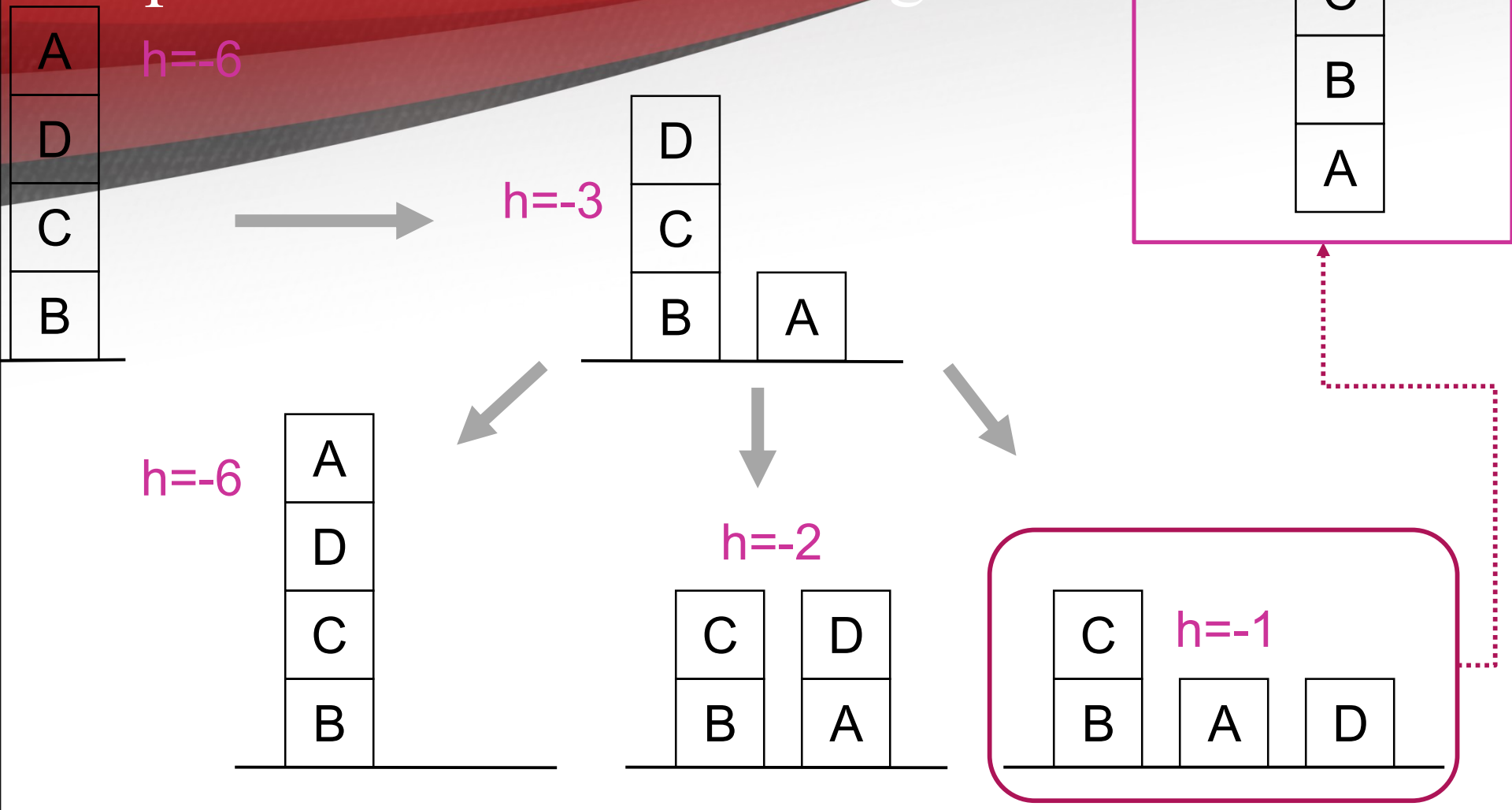
For each block that has the correct support structure:

+1 to every block in the support structure.

For each block that has a wrong support structure:

-1 to every block in the support structure.

Block World problem using Steepest-Ascent Hill climbing



Simple Hill Climbing - Algorithm

1. Evaluate the initial state. If it is also goal state, then return it and quit. Otherwise continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
   a. Select an operator that has not yet been applied to the current state and apply it to produce a new state.
   b. Evaluate the new state
      i. If it is the goal state, then return it and quit.
      ii. If it is not a goal state but it is better than the current state, then make it the current state.
      iii. If it is not better than the current state, then continue in the loop.

Steepest-Ascent Hill Climbing Algorithm

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state:
   a. Let S be a state such that any possible successor of the current state will be better than S.
   b. For each operator that applies to the current state do:
      Apply the operator and generate a new state
      Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to S. If it is better, then set S to this state. If it is not better, leave S alone.
   c. If the S is better than the current state, then set current state to S.