Parul® University

ARTIFICIAL INTELLIGENCE (203105322)

Dr. Pooja Sapra, Associate Professor
Information Technology

SEARCH TECHNIQUES

__Uninformed search algorithms or Brute-force algorithms__,

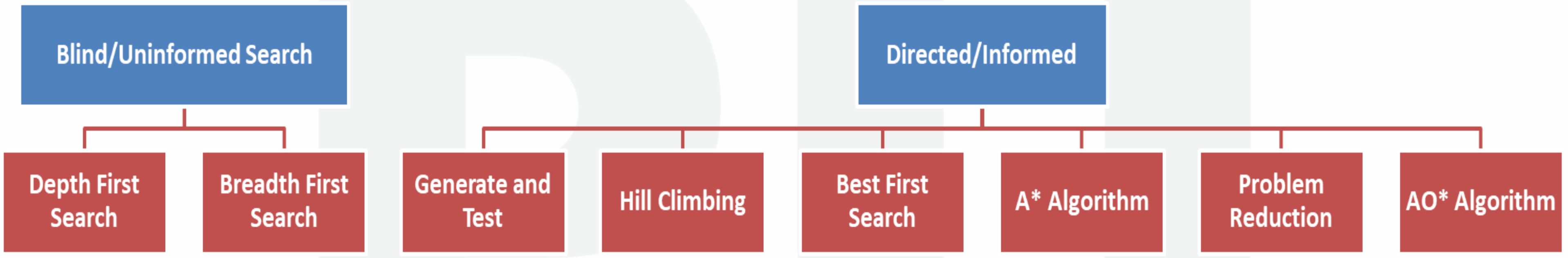
search through the search space all possible candidates for the solution checking whether each candidate satisfies the problem’s statement.

__Informed search algorithms__ use heuristic functions that are specific to the problem, apply them to guide the search through the search space to try to reduce the amount of time spent in searching

TEACHING & EXAMINATION SCHEME

Parameters	Informed Search	Uninformed Search
Known as	It is also known as Heuristic Search.	It is also known as Blind Search.
Using Knowledge	It uses knowledge for the searching process.	It doesn't use knowledge for the searching process.
Performance	It finds a solution more quickly.	It finds solution slow as compared to an informed search.
Completion	It may or may not be complete.	It is always complete.
Cost Factor	Cost is low.	Cost is high.
Time	It consumes less time because of quick searching.	It consumes moderate time because of slow searching.
Direction	There is a direction given about the solution.	No suggestion is given regarding the solution in it.
Implementation	It is less lengthy while implemented.	It is more lengthy while implemented.
Examples of Algorithms	•Greedy Search •A* Search •AO* Search •Hill Climbing Algorithm	•Depth First Search (DFS) •Breadth First Search (BFS)

SEARCH TECHNIQUES



BREADTH FIRST SEARCH

A Search strategy, in which the highest layer of a decision tree is searched completely before proceeding to the next layer is called Breadth-first search (BFS).

- In this strategy, no viable solutions are omitted and therefore it is guaranteed that an optimal solution is found.
- This strategy is often not feasible when the search space is large.

BREADTH FIRST SEARCH

```
1.  Create a variable called LIST and set it to be the starting state.
2.  Loop until a goal state is found or LIST is empty, Do
        a. Remove the first element from the LIST and call it E. If the LIST is empty, quit.
        b. For every path each rule can match the state E, Do
                (i) Apply the rule to generate a new state.
                (ii) If the new state is a goal state, quit and return this state.
                (iii) Otherwise, add the new state to the end of LIST.
```

BREADTH FIRST SEARCH

Advantages

1. Guaranteed to find an optimal solution (in terms of shortest number of steps to reach the goal).

2. Can always find a goal node if one exists (complete).

Disadvantages

1. High storage requirement: exponential with tree depth

DEPTH FIRST SEARCH

A search strategy that extends the current path as far as possible before backtracking to the last choice point and trying the next alternative path is called Depth-first search (DFS).

• This strategy does not guarantee that the optimal solution has been found.

• In this strategy, search reaches a satisfactory solution more rapidly than breadth first, an advantage when the search space is large.

DEPTH FIRST SEACRH

Depth-first search applies operators to each newly generated state, trying to drive directly toward the goal.

1. If the starting state is a goal state, quit and return success.
2. Otherwise, do the following until success or failure is signaled:

   a. Generate a successor E to the starting state. If there are no more successors, then signal failure.

   b. Call Depth-first Search with E as the starting state.

   c. If success is returned signal success; otherwise, continue in the loop.

DEPTH FIRST SEARCH

Advantages

1. Low storage requirement: linear with tree depth.

2. Easily programmed: function call stack does most of the work of maintaining state of the search.

Disadvantages

1. May find a sub-optimal solution (one that is deeper or more costly than the best solution).

2. Incomplete: without a depth bound, may not find a solution even if one exists.

HEURISTIC

- A heuristic is a method that improves the efficiency of the search process.
- These are like tour guides.
- There are good to the level that they point in general interesting directions;
- they are bad to the level that they may neglect points of interest to particular individuals.

HEURISTIC?

- Heuristics may not find the best solution every time but guarantee that they find a good solution in a reasonable time.
- These are particularly useful in solving tough and complex problems, solutions of which would require infinite time, i.e. far longer than a lifetime for the problems which are not solved in any other way.

Heuristic Function

Heuristic knowledge can be incorporated in rule based search techniques by using Heuristic Functions.

Heuristic function evaluates individual problem states and determines how desirable they are.

A heuristic function maps the problem state descriptions to measure of desirability, usually represented as numbers.

CHOOSING HEURISTIC FUNCTION?

HF must give as good estimate as possible, if a node is to be considered on a desired path to a solution.

For finding a solution, by using the heuristic technique, one should carry out the following steps:

1. Add domain—specific information to select what is the best path to continue searching along.
2. Define a heuristic function h(n) that estimates the 'goodness' of a node n. Specifically, h(n) = estimated cost(or distance) of minimal cost path from n to a goal state.

In general, there is a trade off between choosing a heuristic and savings in search time that function provides.

Image source : http://python-nitol.blogspot.com/

Characteristics of heuristic search

Heuristic is a function that, when applied to a state, returns value as estimated merit of state, with respect to goal.

Heuristics might (for reasons) underestimate or overestimate the merit of a state with respect to goal.

Heuristics that underestimate are desirable and called admissible.

Heuristic evaluation function estimates likelihood of given state leading to goal state.

Heuristic search function estimates cost from current state to goal, presuming function is efficient.

1. GENERATE & TEST

- Generate a possible solution.
- Test to see if this is actually a solution by comparing the chose point or the endpoint of the chosen path to the set of acceptable goal states.
- If a solution has been found, quit. Otherwise, return to step 1.

2. HILL CLIMBING

- It is Local Search algorithm
- It is Greedy approach of solving problem
- It examines the neighboring nodes one by one and selects the first neighboring node which optimizes the current cost as next node

HILL CLIMBING - Algorithm

Step 1 : Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.

Step 2 : Loop until the solution state is found or there are no new operators present which can be applied to current state.

a) Select a state that has not been yet applied to the current state and apply it to produce a new state.

b) Perform these to evaluate new state

i. If the current state is a goal state, then stop and return success.

ii. If it is better than the current state, then make it current state and proceed further.

iii. If it is not better than the current state, then continue in the loop until a solution is found.

Step 3 : Exit

Steepest Hill Climbing

It first examines all the neighboring nodes and then selects the node closest to the solution state as next node.

```
Step 1 : Evaluate the initial state. If it is goal state then exit else make the current state as initial state
Step 2 : Repeat these steps until a solution is found or current state does not change
        i. Let ‘target’ be a state such that any successor of the current state will be better than it;
        ii. for each operator that applies to the current state
                a. apply the new operator and create a new state
                b. evaluate the new state
                c. if this state is goal state then quit else compare with ‘target’
                d. if this state is better than ‘target’, set this state as ‘target’
                e. if target is better than current state set current state to Target
Step 3 : Exit
```

Disadvantages

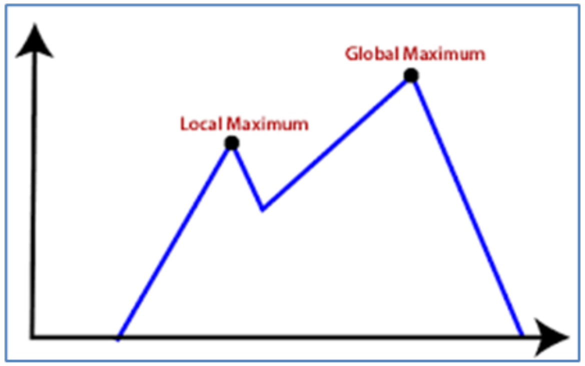
Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :

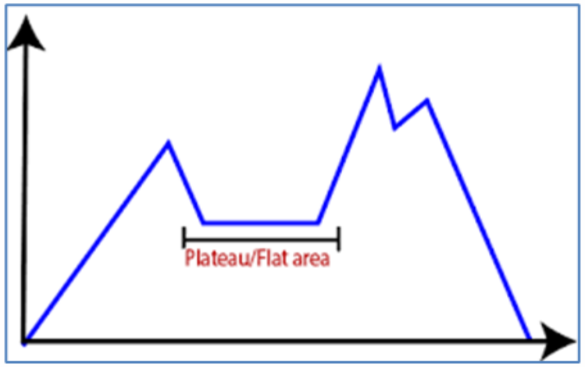
1. Local maximum : At a local maximum all neighboring states have a values which is worse than the current state. Since hill climbing uses greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.

To overcome local maximum problem : Utilize backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.

2. Plateau : On plateau all neighbors have same value . Hence, it is not possible to select the best direction.

To overcome plateaus : Make a big jump. Randomly select a state far away from current state. Chances are that we will land at a non-plateau region





3. BEST FIRST SEARCH

ADVANTAGES OF DFS

DFS is good because it allows a solution to be found without all competing branches having to be expanded.

ADVANTAGES OF BFS

BFS is good because it does not get branches on dead-end paths.

BEST FIRST SEARCH

To combine the two 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 BFS search process, we select the most promising of the nodes we have generated so far.
This is done by applying an appropriate heuristic function to each of them.

BEST FIRST SEARCH

Best First Search generates the successors of the chosen node, applies the heuristic function to them, and adds them to the list of open nodes, after checking to see if any of them have been generated before.

By doing this check, we can guarantee that each node only appears once in the graph, although many nodes may point to it as a successor.

Data Structures Required:
OPEN — nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined (i.e., had their successors generated).
CLOSED — nodes that have already been examined. We need to keep these nodes in memory if we want to search a graph rather than a tree since whenever a new node is generated, we need to check whether it has been generated before.
HEURISTIC FUNCTION - $f' = g + h'$

BEST FIRST SEARCH

1. Start with OPEN containing just the initial state.

2. Until a goal is found or there are no nodes left on OPEN do:

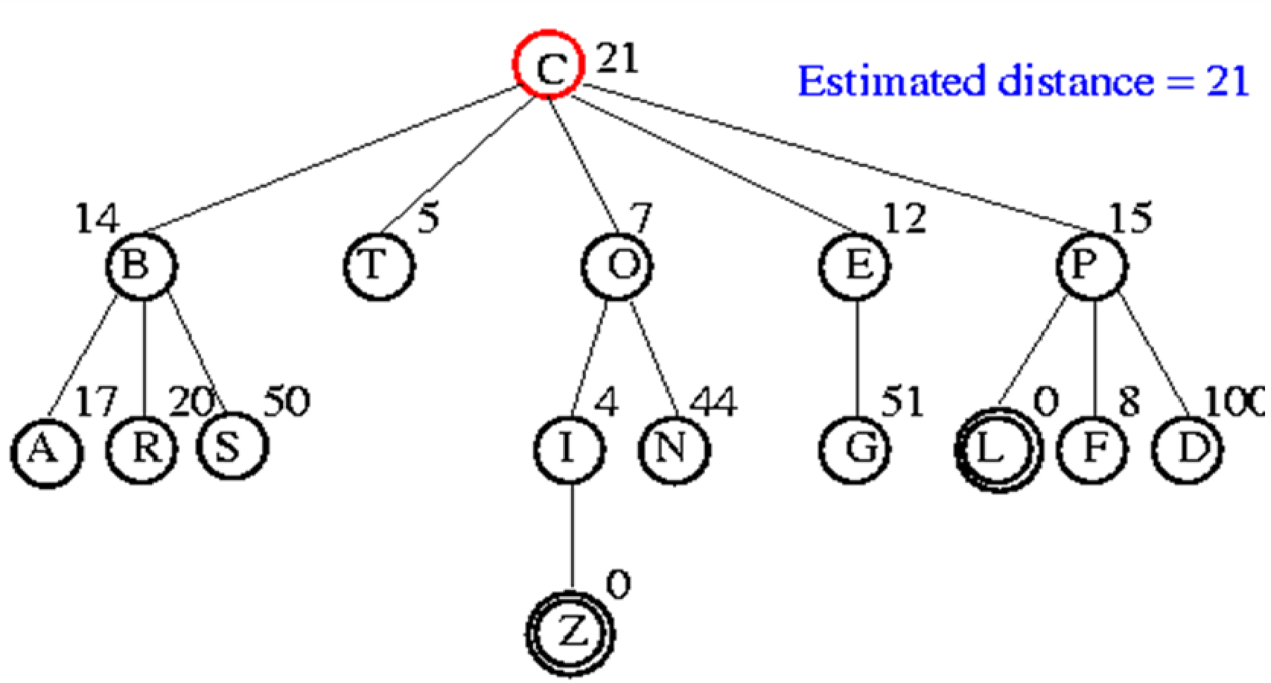
   a) Pick them best node on OPEN.
   b) Generate its successors.
   c) For each successor do:
      i) if it has not been generated before, evaluate it, add it to OPEN, and record its parent.
      ii) If it has been generated before, change the parent if this new path is better than the previous one.
         In that case, update the cost of getting to this node and to any successors that this node may already have.

DIGITAL LEARNING CONTENT

BEST FIRST SEARCH

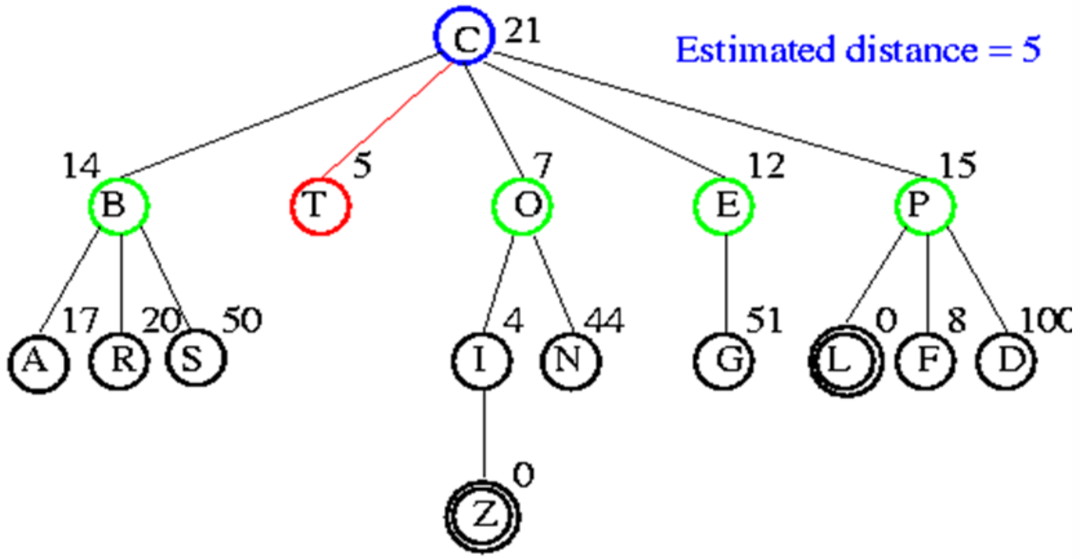
Open: C

Closed: —



BEST FIRST SEARCH

Now, C is added to Closed, and B, T, O, E and P are added to Open.
Open: T, O, E, B, P
Closed: C
Now, T has the least distance hence, T is added to Closed.
Open: O, E, B, P
Closed: C, T

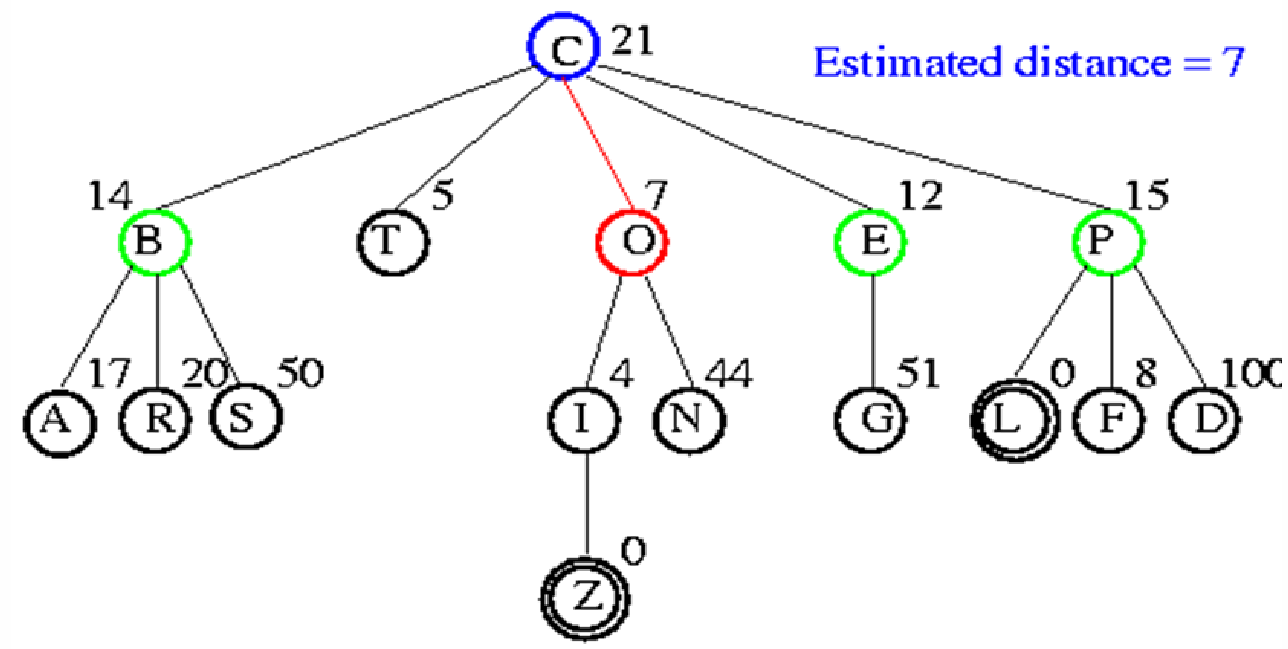


BEST FIRST SEARCH

As T does not have any successors, the next node from open that is O is removed from Open and added to closed.

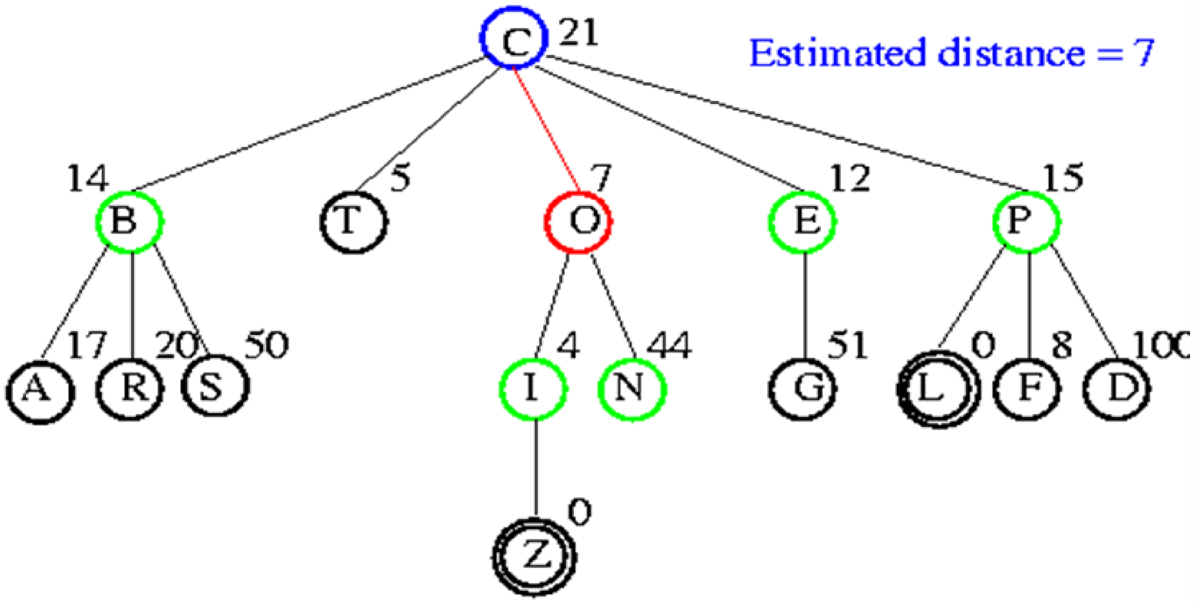
Open: E, B, P

Closed: C, T, O



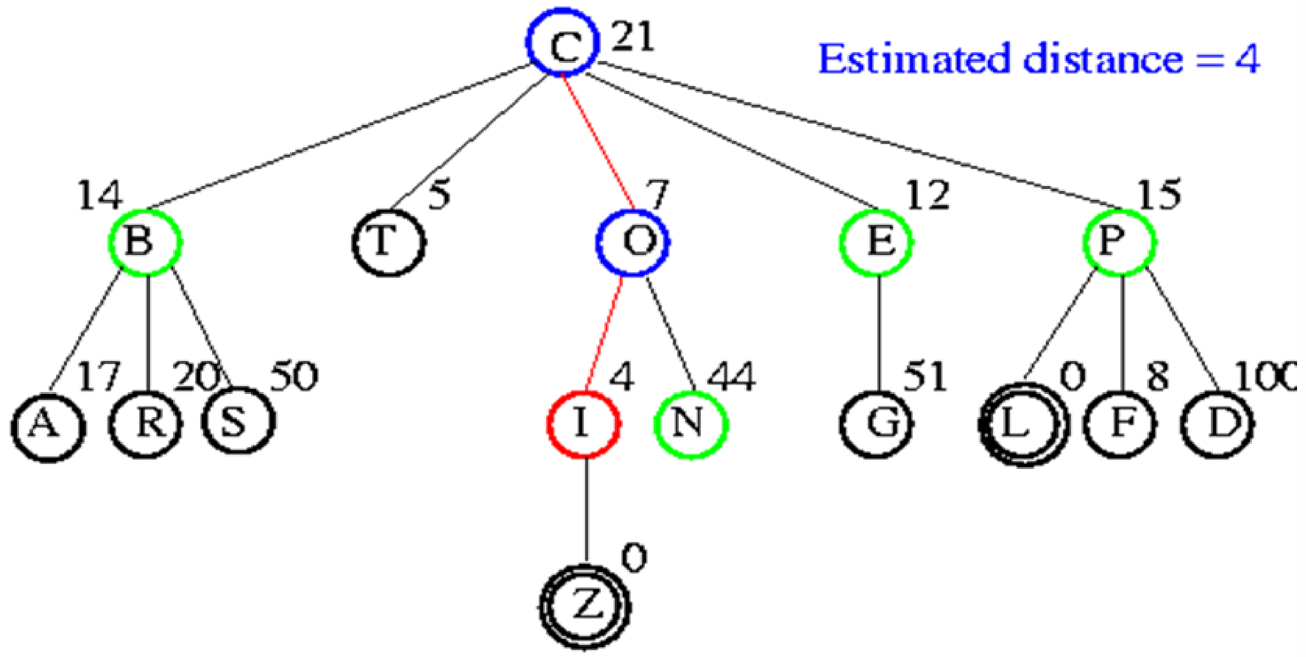
BEST FIRST SEARCH

The successors of node O that is node I and N are added to Open.
Open: I, E, B, P, N
Closed: C, T, O



BEST FIRST SEARCH

Now, node I is removed from Open and added to closed.
Open: E, B, P, N
Closed: C, T, O, I

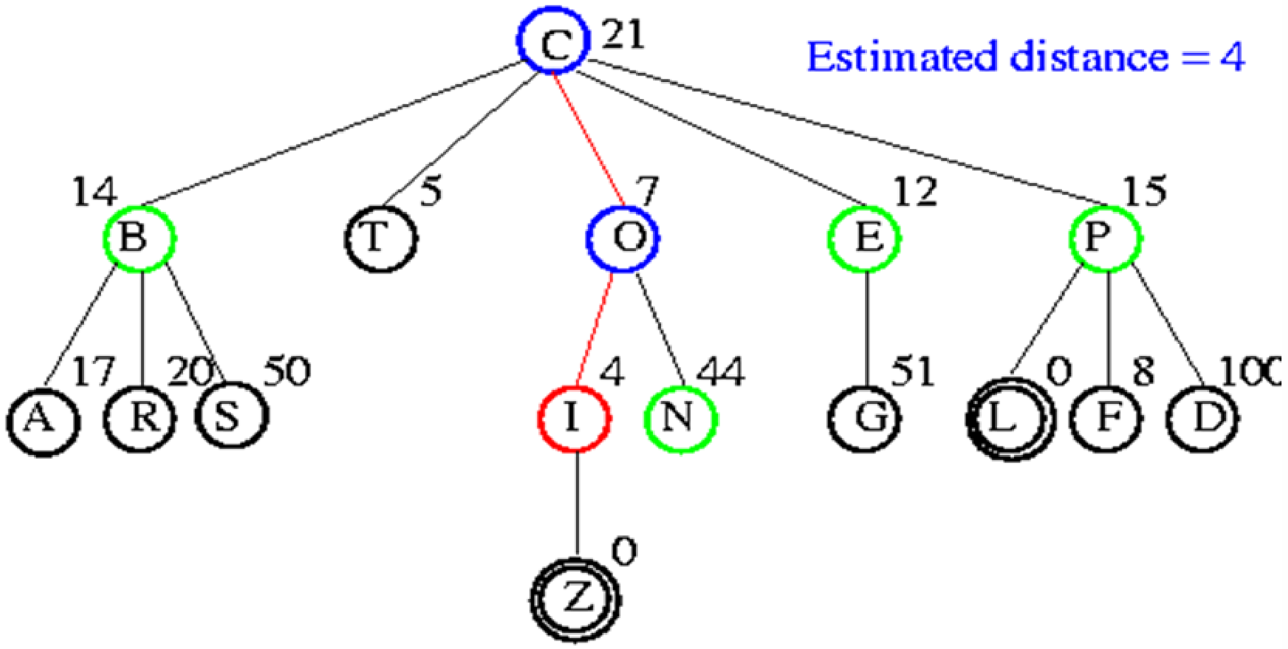


Parul® University

DIGITAL LEARNING CONTENT

BEST FIRST SEARCH

The successor of I that is Z is added to Open.
Open: Z, E, B, P, N
Closed: C, T, O, I

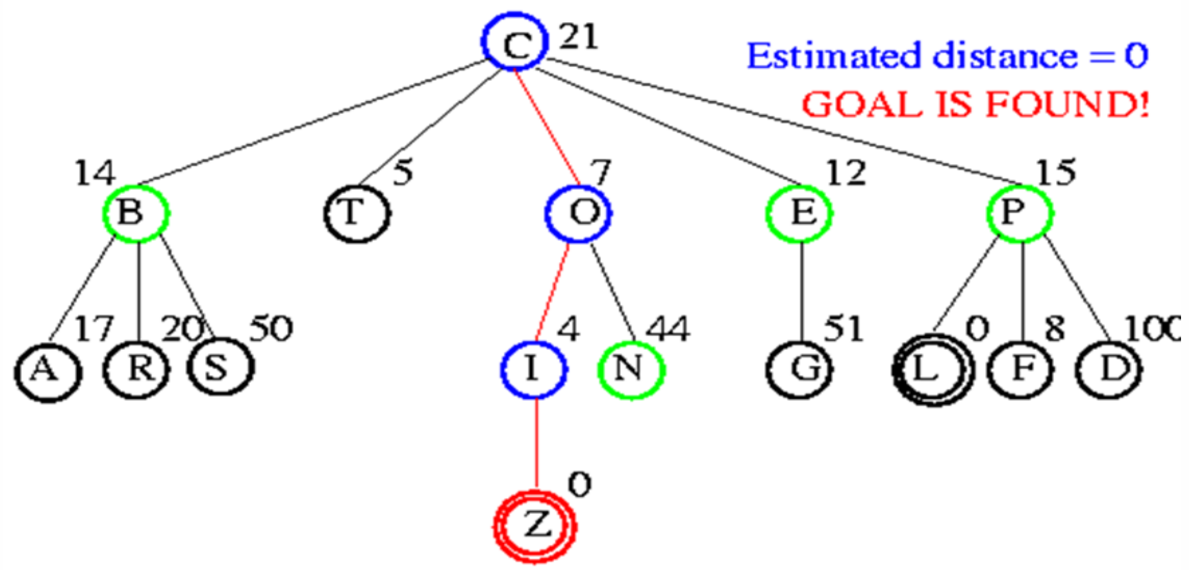


Parul® University

DIGITAL LEARNING CONTENT

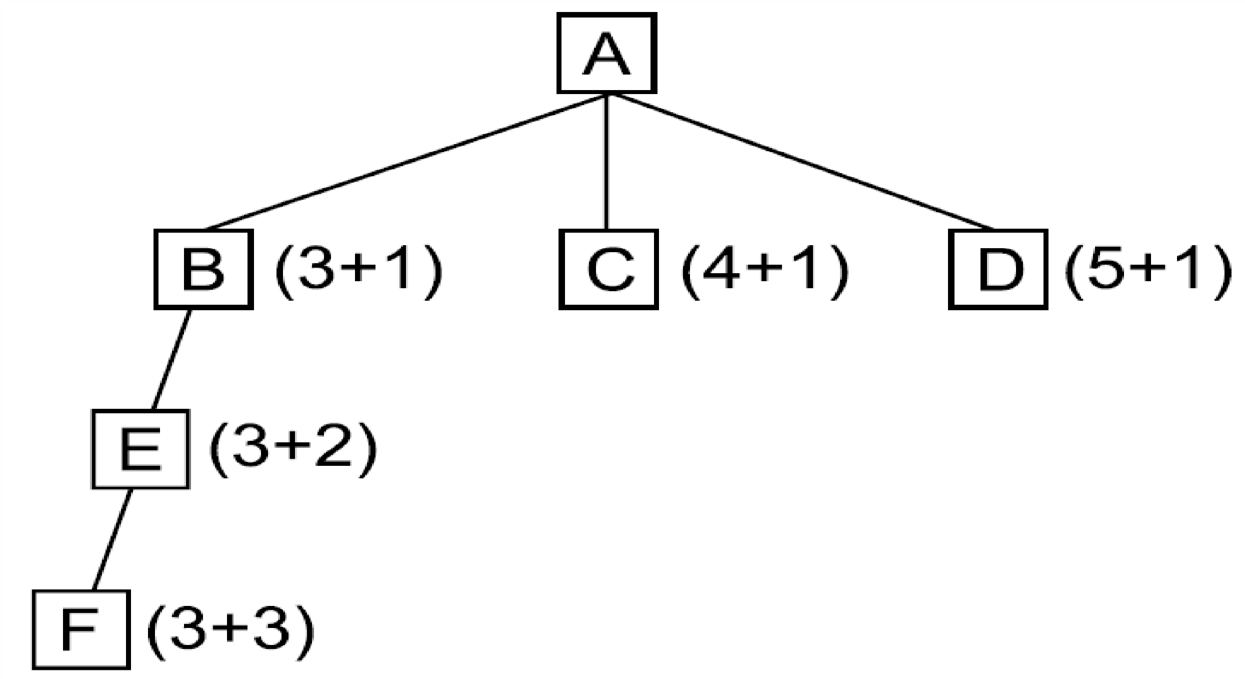
BEST FIRST SEARCH

Now, node Z is removed from Open and added to closed.
Open: E, B, P, N
Closed: C, T, O, I, Z

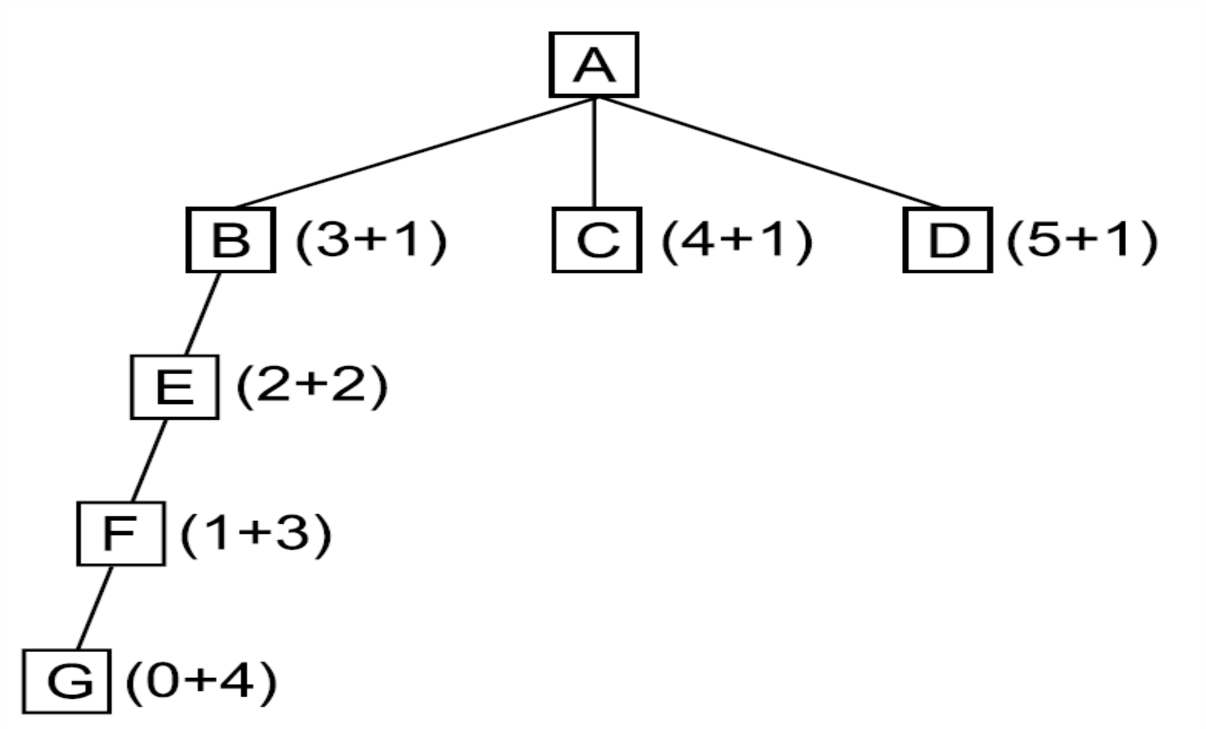


The Goal is found. The final path is C – O – I – Z.

h' Underestimates h



h′ Overestimates h



A* ALGORITHM

It is the combination of Dijkstra’s algorithm and the Best first search.
It can be used to solve many kinds of problems, like finding the shortest path through a search space to the goal state using the heuristic function.
This technique finds minimal cost solutions and is directed to a goal state called A* search.
In A*, the * is written for optimality purposes.

A* ALGORITHM

A* requires the heuristic function to evaluate the cost of the path that passes through the particular state.

•It can be defined by the following formula.

•f(n) = g(n) + h(n)

Where

•*g(n): The actual cost path from the start state to the current state.*

•*h(n): The actual cost path from the current state to the goal state.*

•*f(n): The actual cost path from the start state to the goal state.*

For the implementation of the A algorithm, we will use two arrays namely OPEN and CLOSE.*

•**OPEN:** An array that contains the nodes that have been generated but have not been yet examined.

•**CLOSE:** An array that contains the nodes that have been examined.

A* ALGORITHM

- Step 1: Place the starting node into OPEN and find its f (n) value.
- Step 2: Remove the node from OPEN, having the smallest f (n) value. If it is a goal node then stop and return success.
- Step 3: Else remove the node from OPEN, find all its successors.
- Step 4: Find the f (n) value of all successors; place them into OPEN and place the removed node into CLOSE.
- Step 5: Go to Step-2.
- Step 6: Exit.

A* ALGORITHM

Advantages of A* Star

•It is complete and optimal.

•It is the best one from other techniques.

•It is used to solve very complex problems.

•It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A*.

Disadvantages of A* Star

•This algorithm is complete if the branching factor is finite and every action has a fixed cost.

•The speed execution of the A* search is highly dependent on the accuracy of the heuristic algorithm that is used to compute h (n).

•It has complexity problems.

A* ALGORITHM-EXAMPLE

2	8	3
1	6	4
7		5

Initial State

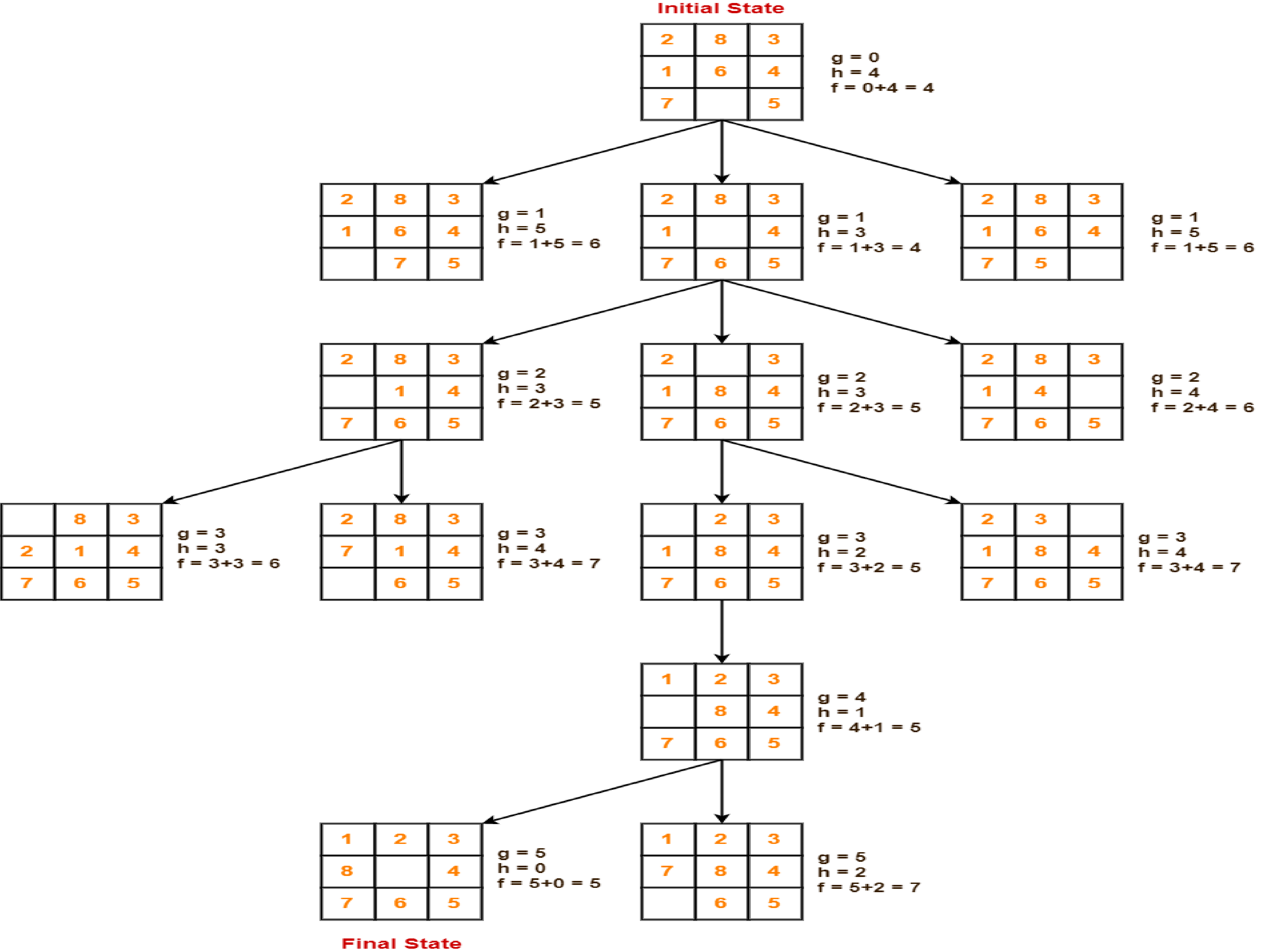
1	2	3
8		4
7	6	5

Final State

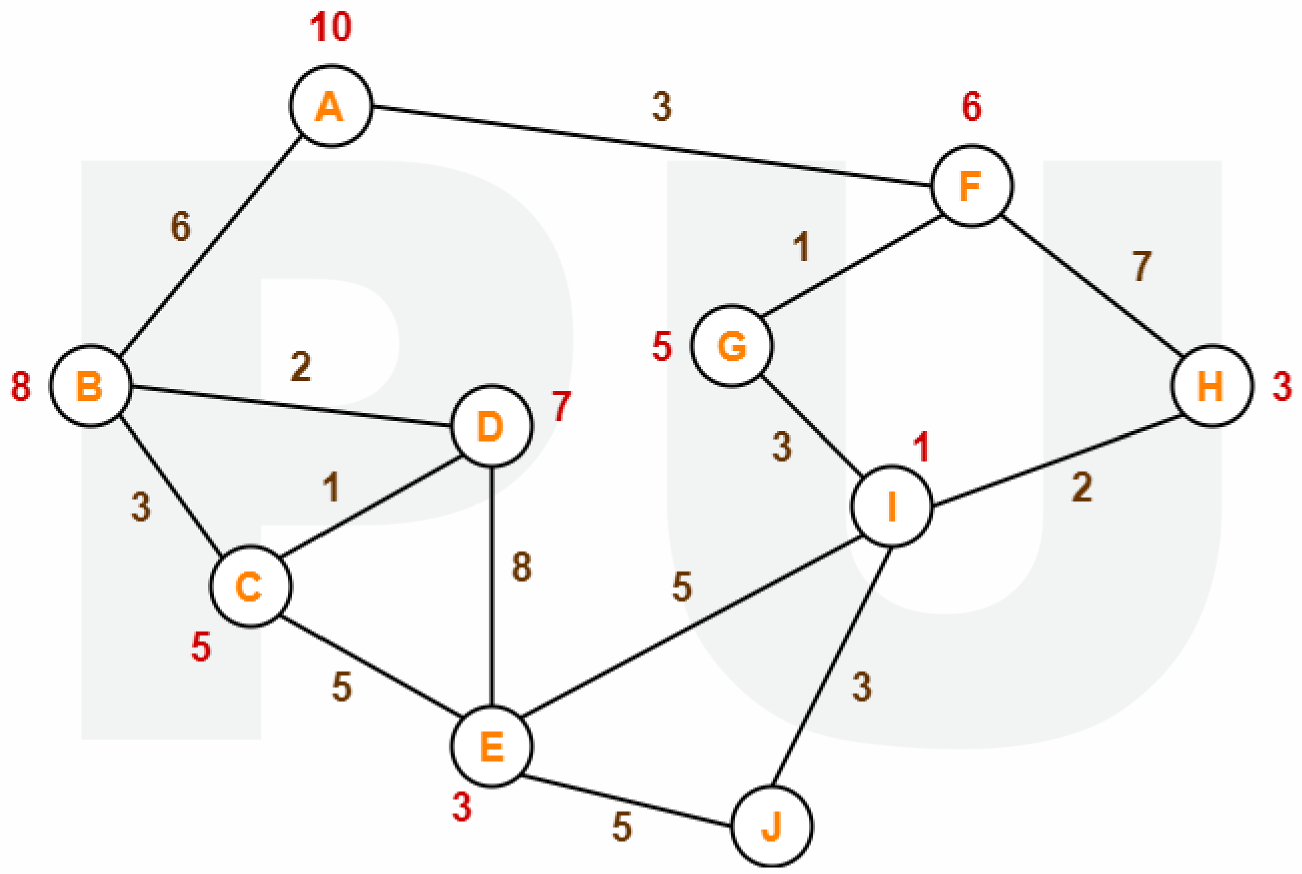
Find the most cost-effective path to reach the final state from initial state using A* Algorithm.
Consider g(n) = Depth of node and h(n) = Number of misplaced tiles.

A* ALGORITHM-EXAMPLE

- A* Algorithm maintains a tree of paths originating at the initial state.
- It extends those paths one edge at a time.
- It continues until final state is reached.


Initial State
2 8 3 / 1 6 4 / 7 _ 5
g = 0
h = 4
f = 0+4 = 4
2 8 3 / 1 6 4 / _ 7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3 / 1 _ 4 / 7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3 / 1 6 4 / 7 5 _
g = 1
h = 5
f = 1+5 = 6
2 8 3 / _ 1 4 / 7 6 5
g = 2
h = 3
f = 2+3 = 5
2 _ 3 / 1 8 4 / 7 6 5
g = 2
h = 3
f = 2+3 = 5
2 8 3 / 1 4 _ / 7 6 5
g = 2
h = 4
f = 2+4 = 6
_ 8 3 / 2 1 4 / 7 6 5
g = 3
h = 3
f = 3+3 = 6
2 8 3 / 7 1 4 / _ 6 5
g = 3
h = 4
f = 3+4 = 7
_ 2 3 / 1 8 4 / 7 6 5
g = 3
h = 2
f = 3+2 = 5
2 3 _ / 1 8 4 / 7 6 5
g = 3
h = 4
f = 3+4 = 7
1 2 3 / _ 8 4 / 7 6 5
g = 4
h = 1
f = 4+1 = 5
1 2 3 / 8 _ 4 / 7 6 5
g = 5
h = 0
f = 5+0 = 5
1 2 3 / 7 8 4 / _ 6 5
g = 5
h = 2
f = 5+2 = 7
Final State

A* ALGORITHM-EXAMPLE-2



A* ALGORITHM-EXAMPLE-2

Step-01:

- We start with node A.
- Node B and Node F can be reached from node A.

A* Algorithm calculates f(B) and f(F).
- f(B) = 6 + 8 = 14
- f(F) = 3 + 6 = 9

Since f(F) < f(B), so it decides to go to node F.

Path- A → F

A* ALGORITHM-EXAMPLE-2

Step-02:

Node G and Node H can be reached from node F.

A* Algorithm calculates f(G) and f(H).
- $f(G) = (3+1) + 5 = 9$
- $f(H) = (3+7) + 3 = 13$

Since f(G) < f(H), so it decides to go to node G.

Path- A → F → G

A* ALGORITHM-EXAMPLE-2

Step-03:

Node I can be reached from node G.

A* Algorithm calculates f(I).
f(I) = (3+1+3) + 1 = 8
It decides to go to node I.

Path- A → F → G → I

A* ALGORITHM-EXAMPLE-2

Step-04:

Node E, Node H and Node J can be reached from node I.

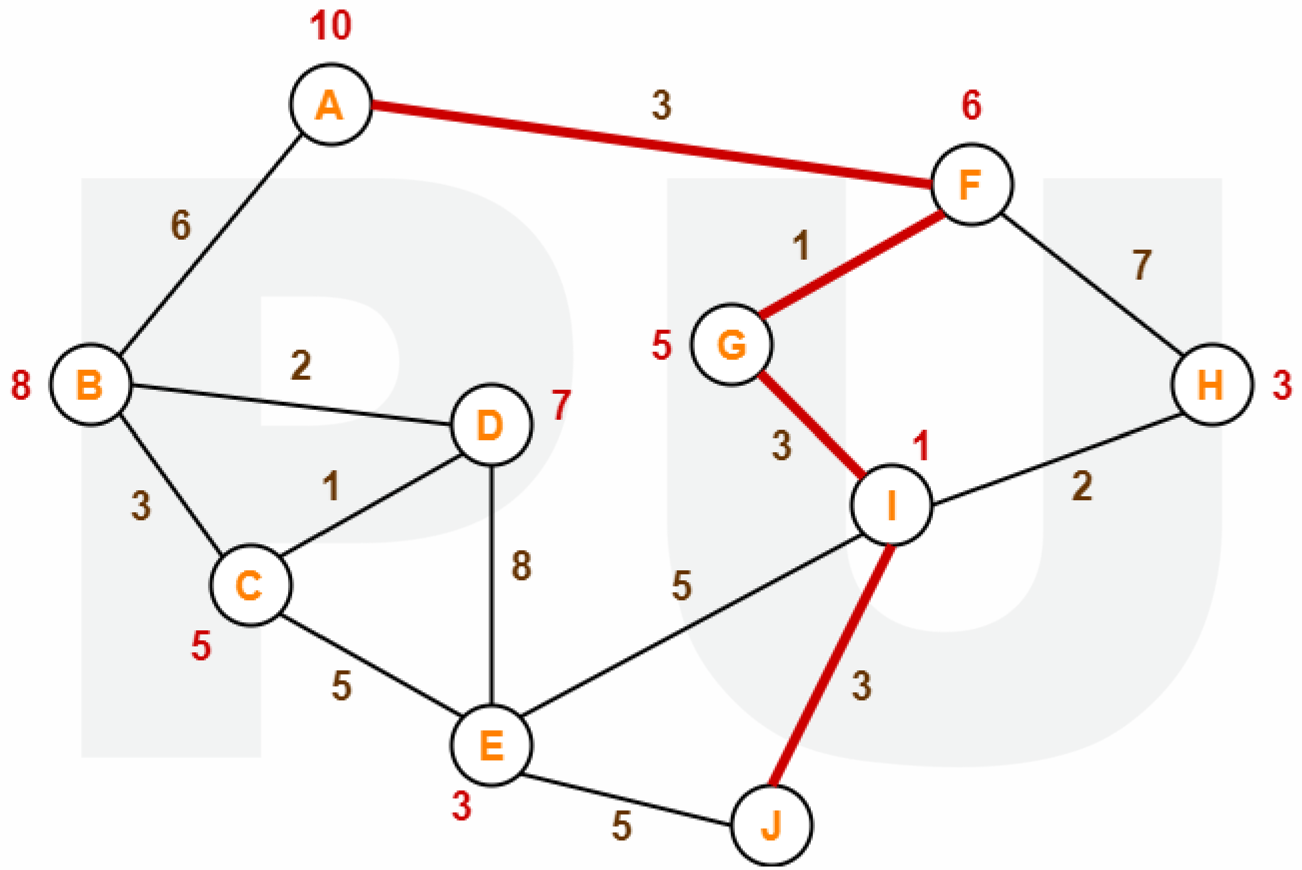
A* Algorithm calculates f(E), f(H) and f(J).

- f(E) = (3+1+3+5) + 3 = 15
- f(H) = (3+1+3+2) + 3 = 12
- f(J) = (3+1+3+3) + 0 = 10

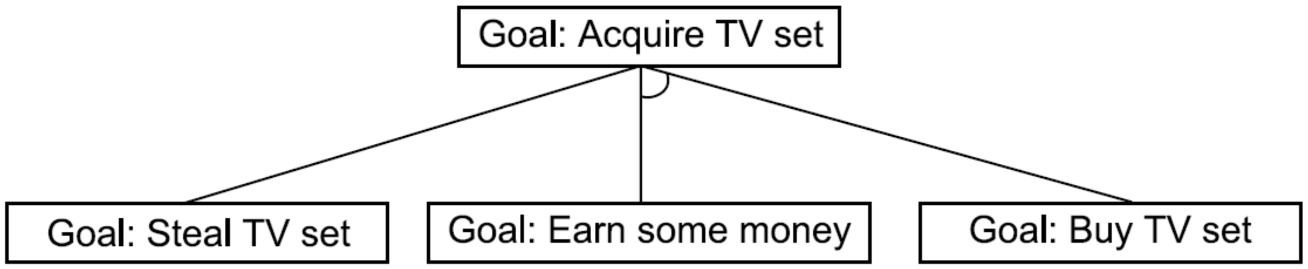
Since f(J) is least, so it decides to go to node J.

Path- A → F → G → I → J

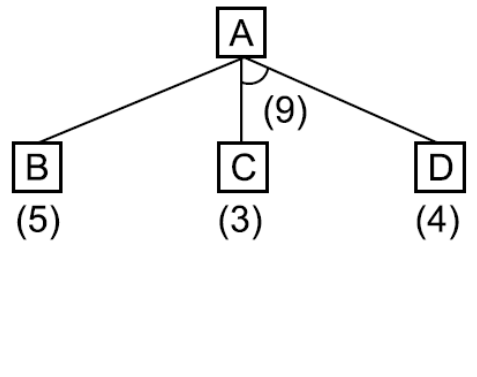
A* ALGORITHM-EXAMPLE-2



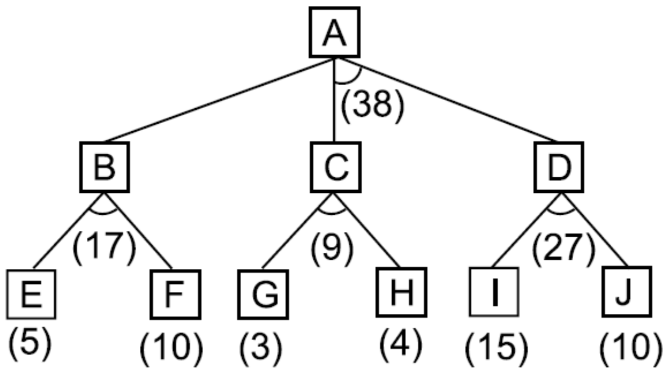
A Simple AND-OR Graph



AND-OR Graphs



(a)



(b)

Algorithm : Problem Reduction

1. **Initialize the graph to the starting node.**

2. **2. Loop until the starting node is labeled *SOLVED* or until its cost goes above *FUTILITY*:**

(a) Traverse the graph, starting at the initial node and following the current best path, and accumulate the set of nodes that are on that path and have not yet been expanded or labeled as solved.

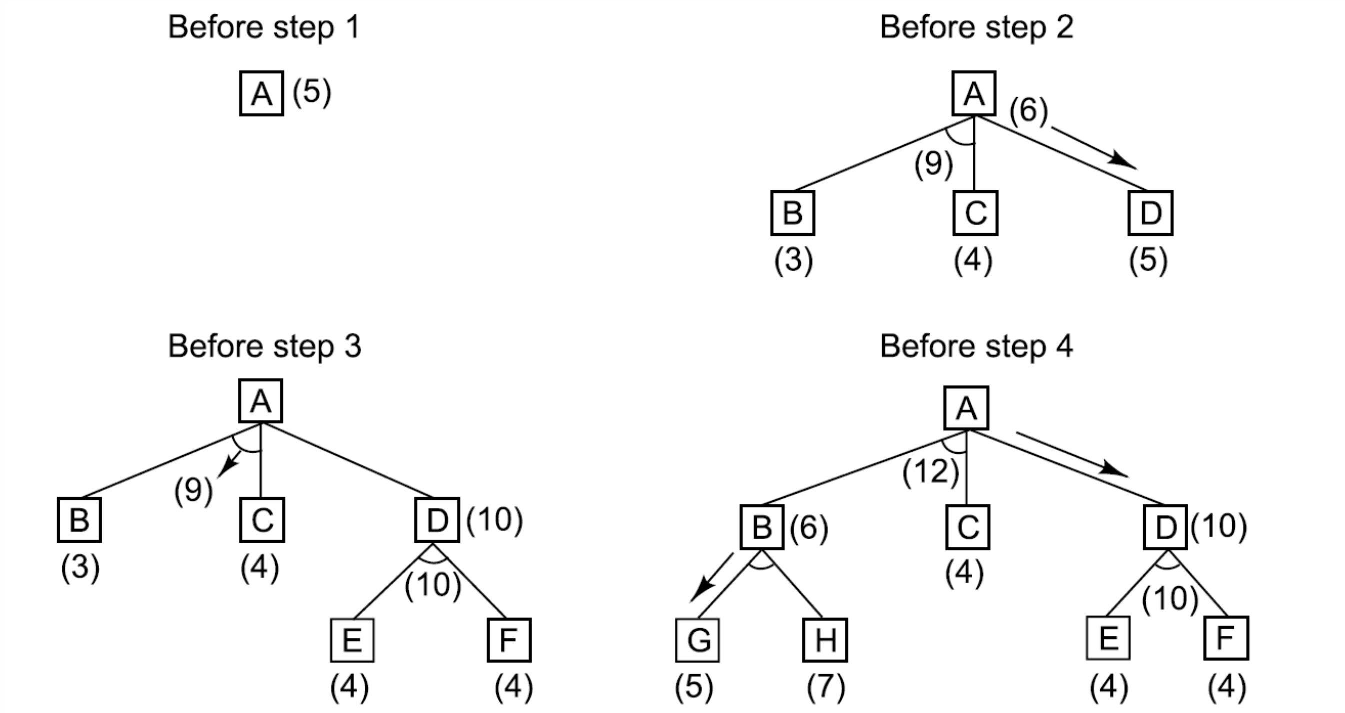
(b) Pick one of these unexpanded nodes and expand it. If there are no successors, assign *FUTILITY* as the value of this node. Otherwise, add its successors to the graph and for each of them compute *f '* (use only *h'* and ignore *g,* for reasons we discuss below). If of any node is 0, mark that node as *SOLVED.*

(c) Change the *f '* estimate of the newly expanded node to reflect the new information provided by its successors. Propagate this change backward through the graph. If any node contains a successor arc whose descendants are all solved, label the node itself as *SOLVED.* At each node that is visited while going up the graph, decide which of its successor arcs is the most promising and mark it as part of the current best path.

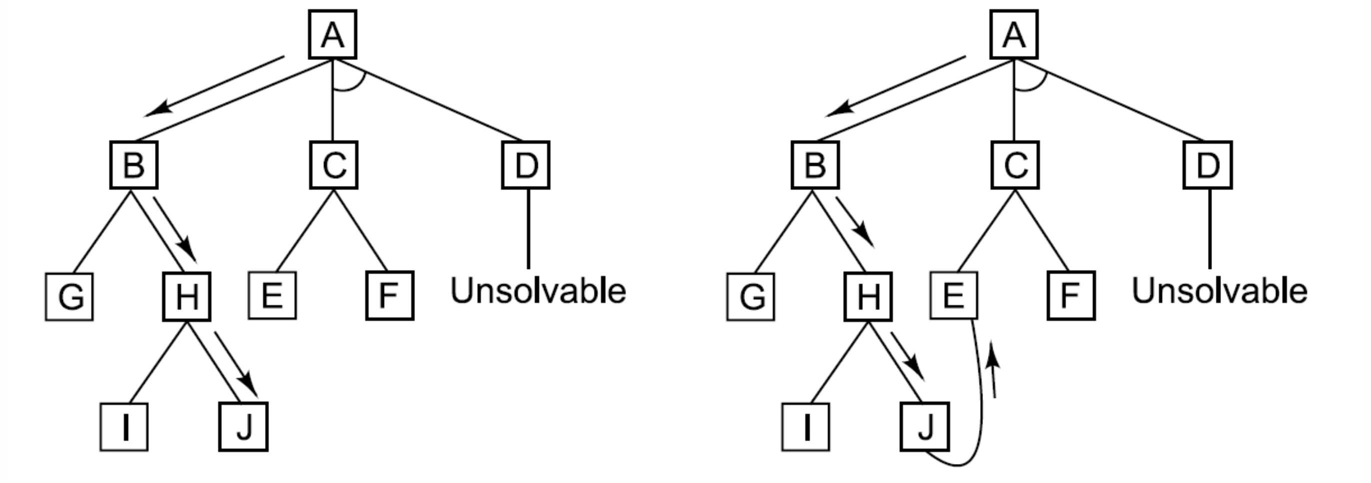
Parul® University

DIGITAL LEARNING CONTENT

The Operation of Problem Reduction



A Longer Path May Be Better



Algorithm : CONSTRAINT PROPAGATION

1. **Initialize the graph to the starting node.**

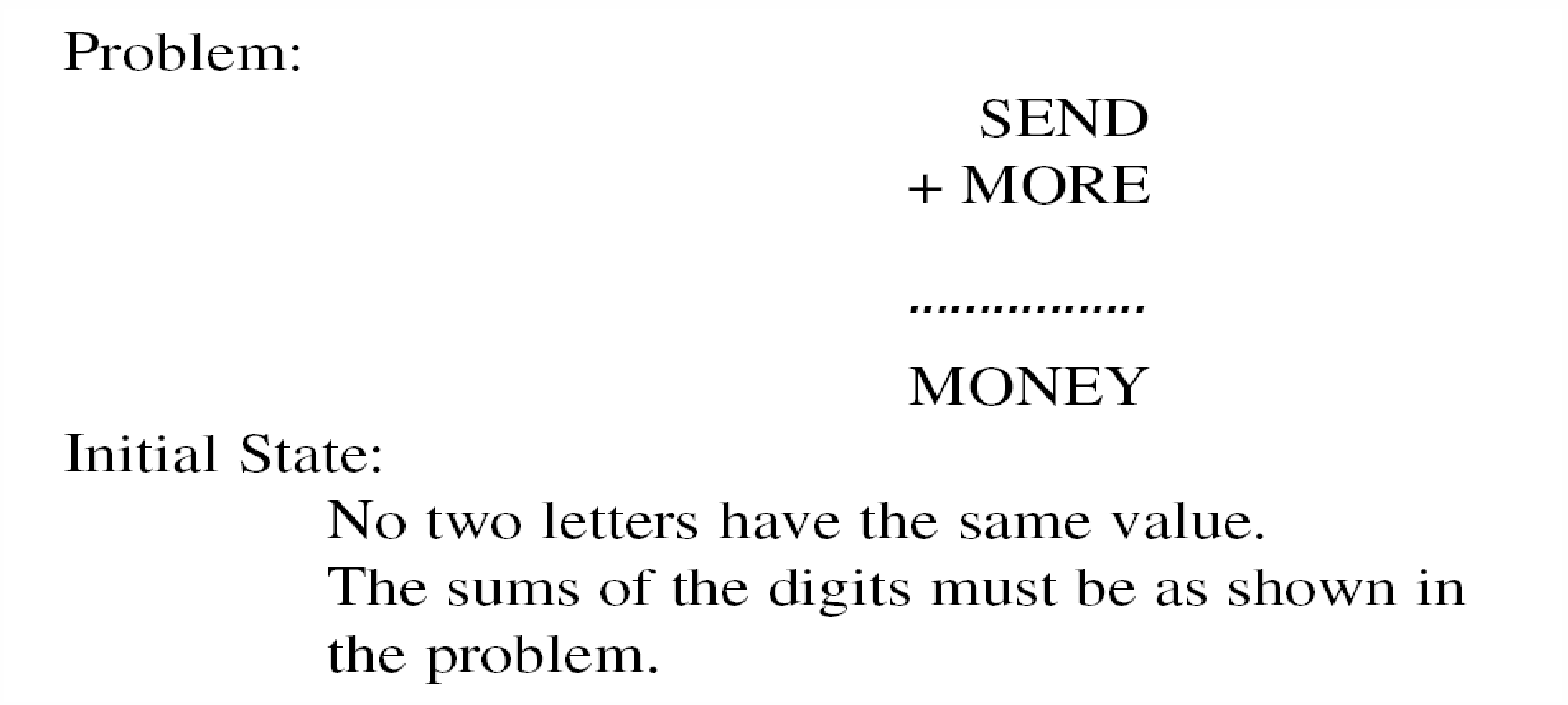
2. **2. Loop until the starting node is labeled *SOLVED* or until its cost goes above *FUTILITY*:**

(a) Traverse the graph, starting at the initial node and following the current best path, and accumulate the set of nodes that are on that path and have not yet been expanded or labeled as solved.

(b) Pick one of these unexpanded nodes and expand it. If there are no successors, assign *FUTILITY* as the value of this node. Otherwise, add its successors to the graph and for each of them compute *f '* (use only *h'* and ignore *g,* for reasons we discuss below). If of any node is 0, mark that node as *SOLVED.*

(c) Change the *f '* estimate of the newly expanded node to reflect the new information provided by its successors. Propagate this change backward through the graph. If any node contains a successor arc whose descendants are all solved, label the node itself as *SOLVED.* At each node that is visited while going up the graph, decide which of its successor arcs is the most promising and mark it as part of the current best path.

A Cryptarithmetic Problem



Problem:

```
   SEND
 + MORE
 ..................
  MONEY
```

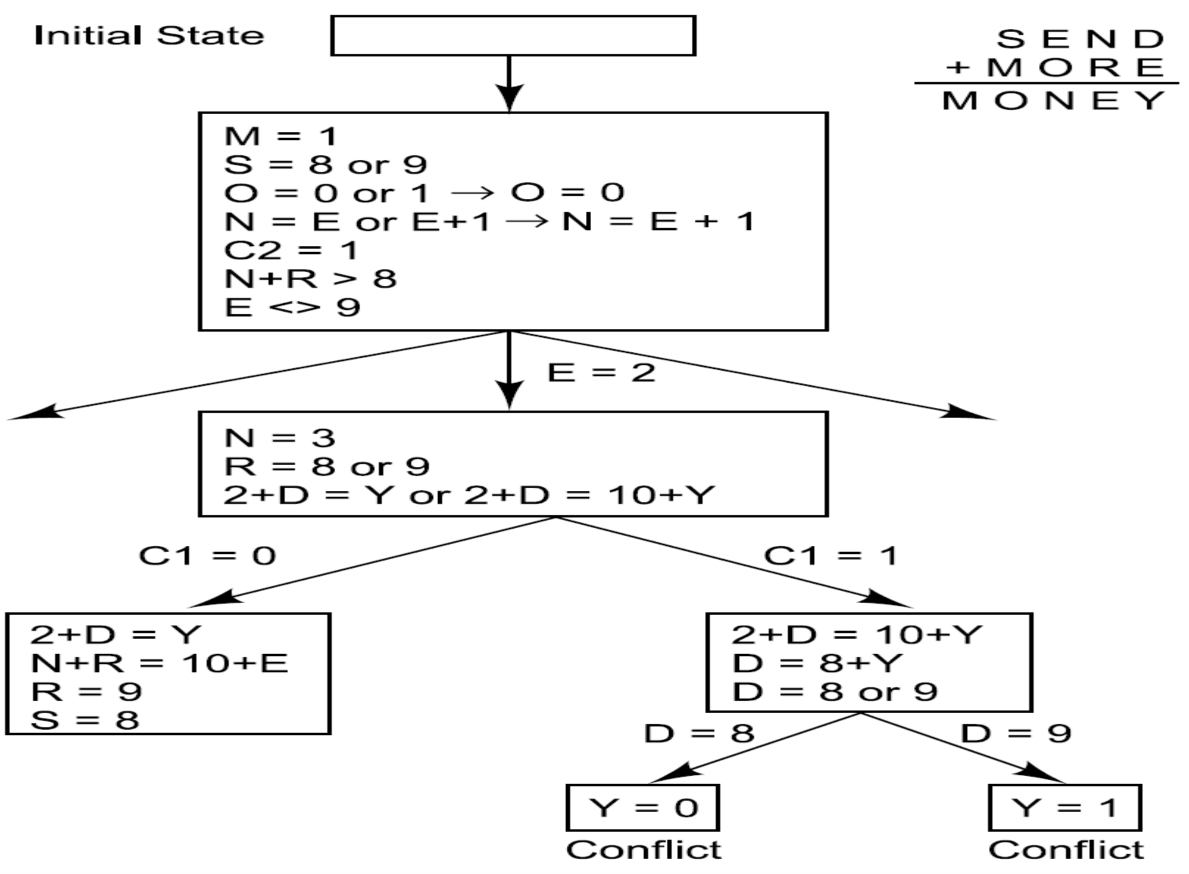
Initial State:

No two letters have the same value.
The sums of the digits must be as shown in the problem.

Parul® University

DIGITAL LEARNING CONTENT

A Cryptarithmetic Problem



Means End Analysis (MEA)

- Means-Ends Analysis is problem-solving techniques used in Artificial intelligence for limiting search in AI programs.
- It is a mixture of Backward and forward search technique.
- The MEA technique was first introduced in 1961 by Allen Newell, and Herbert A. Simon in their problem-solving computer program, which was named as General Problem Solver (GPS).
- The MEA analysis process centered on the evaluation of the difference between the current state and goal state.

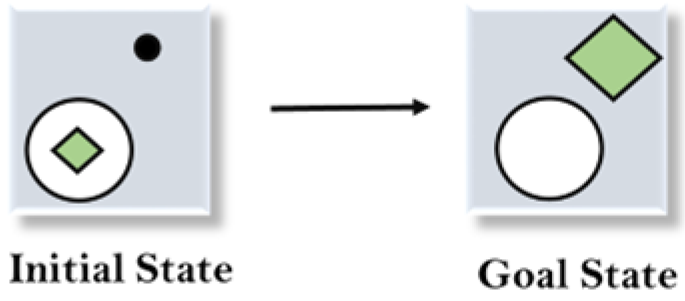
HOW MEA WORKS?

The means-ends analysis process can be applied recursively for a problem. It is a strategy to control search in problem-solving. Following are the main Steps which describes the working of MEA technique for solving a problem.

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.

EXAMPLE

Let's take an example where we know the initial state and goal state as given below. In this problem, we need to get the goal state by finding differences between the initial state and goal state and applying operators.



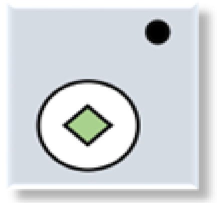
EXAMPLE

To solve the above problem, 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**

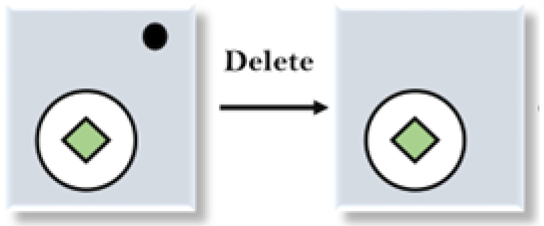
EXAMPLE

1. **Evaluating the initial state:** In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.



Initial state

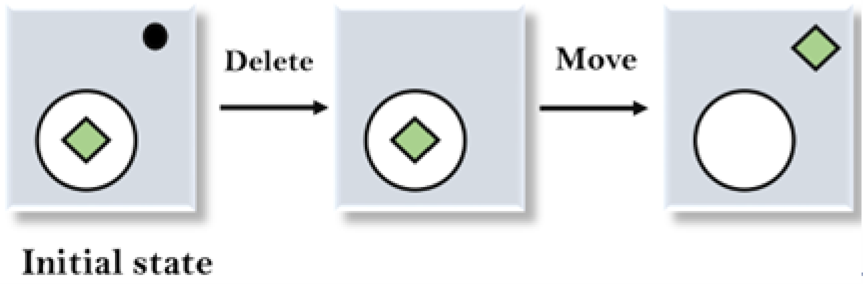
2. **Applying Delete operator:** As we can check the first difference is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the **Delete operator** to remove this dot.



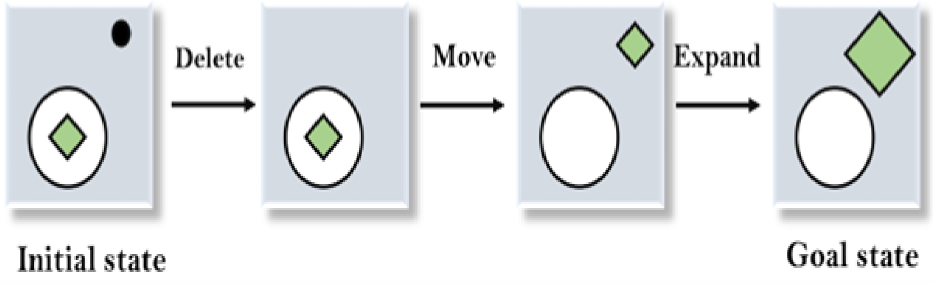
Initial state

EXAMPLE

3. **Applying Move Operator:** After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the square is outside the circle, so, we will apply the **Move Operator**.



4. **Applying Expand Operator:** Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the square, so, we will apply **Expand operator**, and finally, it will generate the goal state.



Algorithm : Means End Analysis (MEA)

1. Compare *CURRENT* to *GOAL.* If there are no differences between them then return.

2. Otherwise, select the most important difference and reduce it by doing the following until success or failure is signaled:

(a) Select an as yet untried operator *O* that is applicable to the current difference. If there are no such operators, then signal failure.

(b) Attempt to apply *O* to *CURRENT.* Generate descriptions of two states: *O-START,* a state in which *O*'s preconditions are satisfied and *O-RESULT,* the state that would result if *O* were applied in *OSTART.*

(c) If
(FIRST-PART ← MEA(CURRENT, O-START))
and
(LAST-PART ← MEMO-RESULT, GOAL))
are successful, *then* signal success and return the result of concatenating
FIRST-PART, O,* and *LAST-PART.

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in