Unit-2

Ch-3 Heuristic Search Techniques

HEURISTIC SEARCH TECHNIQUES:

Heuristics is a technique that improves the efficiency of a search process, possibly by sacrificing claims of completeness.

Heuristics are like tour guides. The algorithms that use heuristic functions are called heuristic algorithms.

Heuristic algorithms are more efficient because they take advantage of feedback from the data to direct the search path.

Informed search algorithms use ***heuristic functions or directed search*** 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.

Uninformed search algorithms or ***Blind Search / Brute-force*** algorithms, search through the search space all possible candidates for the solution checking whether each candidate satisfies the problem's statement.

HEURISTIC SEARCH TECHNIQUES:

In such problems, the search proceeds using current information about the problem to predict which path is closer to the goal and follow it, although it does not always guarantee to find the best possible solution.

Such techniques help in finding a solution within reasonable time and space (memory). Some prominent intelligent search algorithms are stated below:

1. Best-first Search
2. Hill Climbing
3. Greedy Search
4. A* Search
5. Problem Reduction - AO* Search

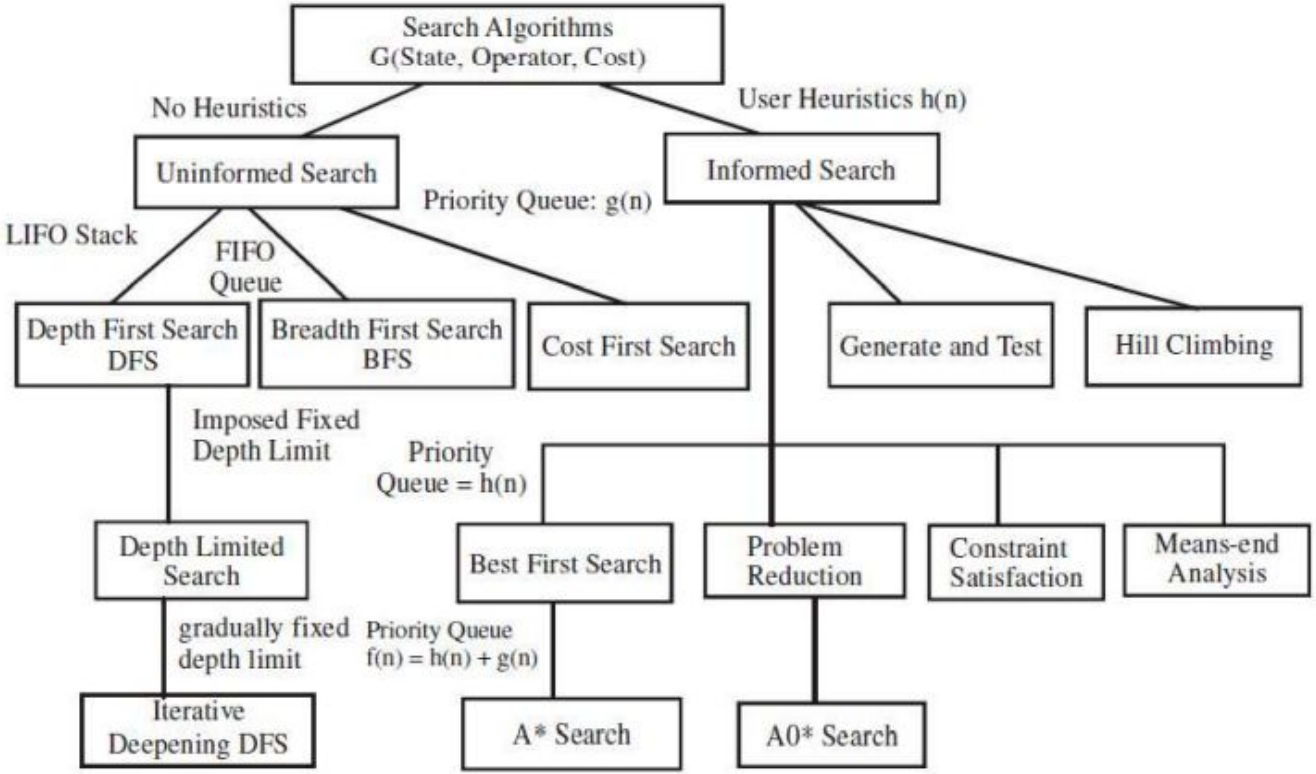


Fig. *Different Search Algorithms*

Characteristics of heuristic search

- Heuristics are knowledge about domain, which help search and reasoning in its domain.
- Heuristic search incorporates domain knowledge to improve efficiency over blind 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.

Best-First Search

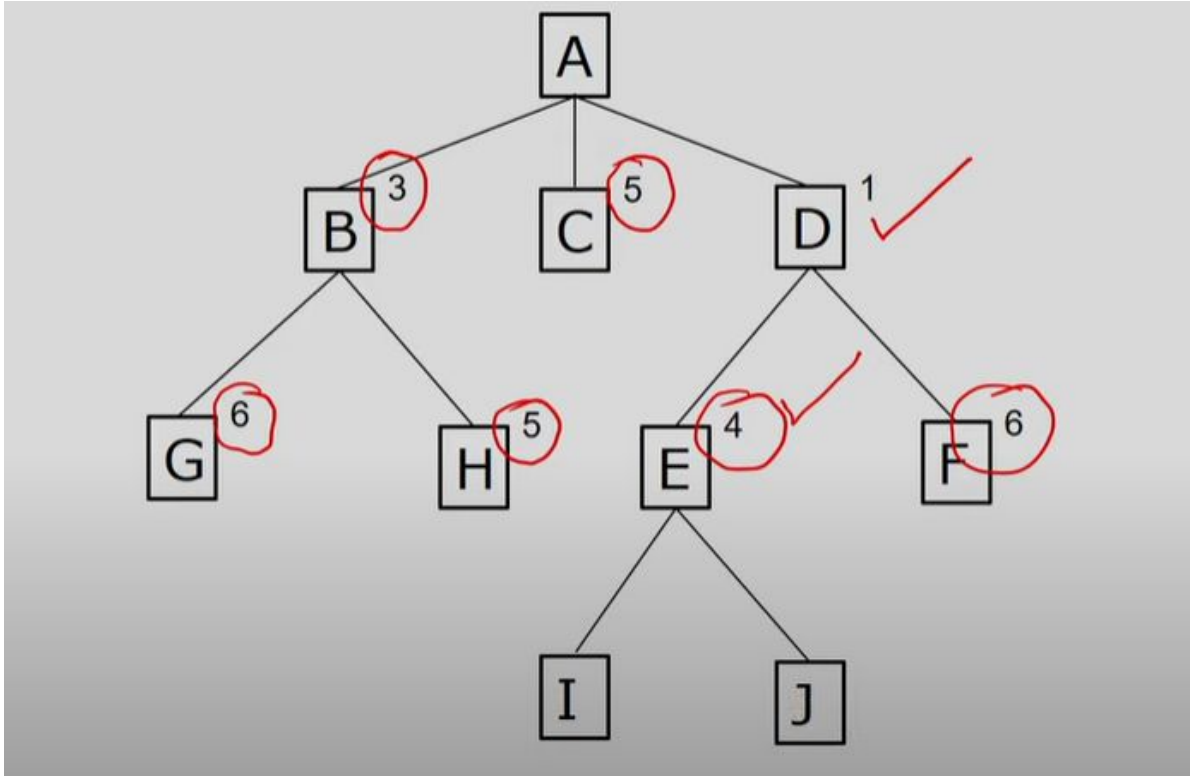
- It is a **way of combining the advantages of both depth-first and breadth-first** search into a single method.
  - Depth-First is good as it allows a solution to be found without all competing branches having to be expanded.
  - Breadth-First is good because it does not get trapped on dead-end paths.
  - Combine by following a single path at a time, but **switch path whenever some competing path looks more promising** than the current one.

Best-First Search

- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node
- Implementation:

  Order the nodes in fringe in decreasing order of desirability
- Special case:
  - A* search

Best-First Search- example



Reorder according to the heuristic merit

Algorithm A*

The most widely known form of best-first search is called A∗ Search (pronounced “A-star SEARCH search”).
It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal:

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

Since g(n) gives the path cost from the **start node to node n,** and **h(n) is the estimated cost** of the cheapest path **from n to the goal**, we have
f(n) = estimated cost of the cheapest solution through n .

A∗ search is both complete and optimal.
The algorithm is identical to UNIFORM-COST-SEARCH except that A∗ uses g + h instead of g.

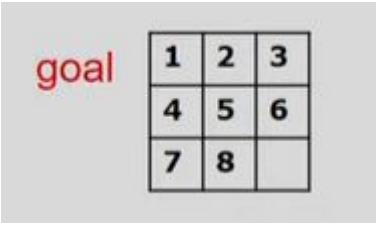
Conditions for optimality: Admissibility and consistency

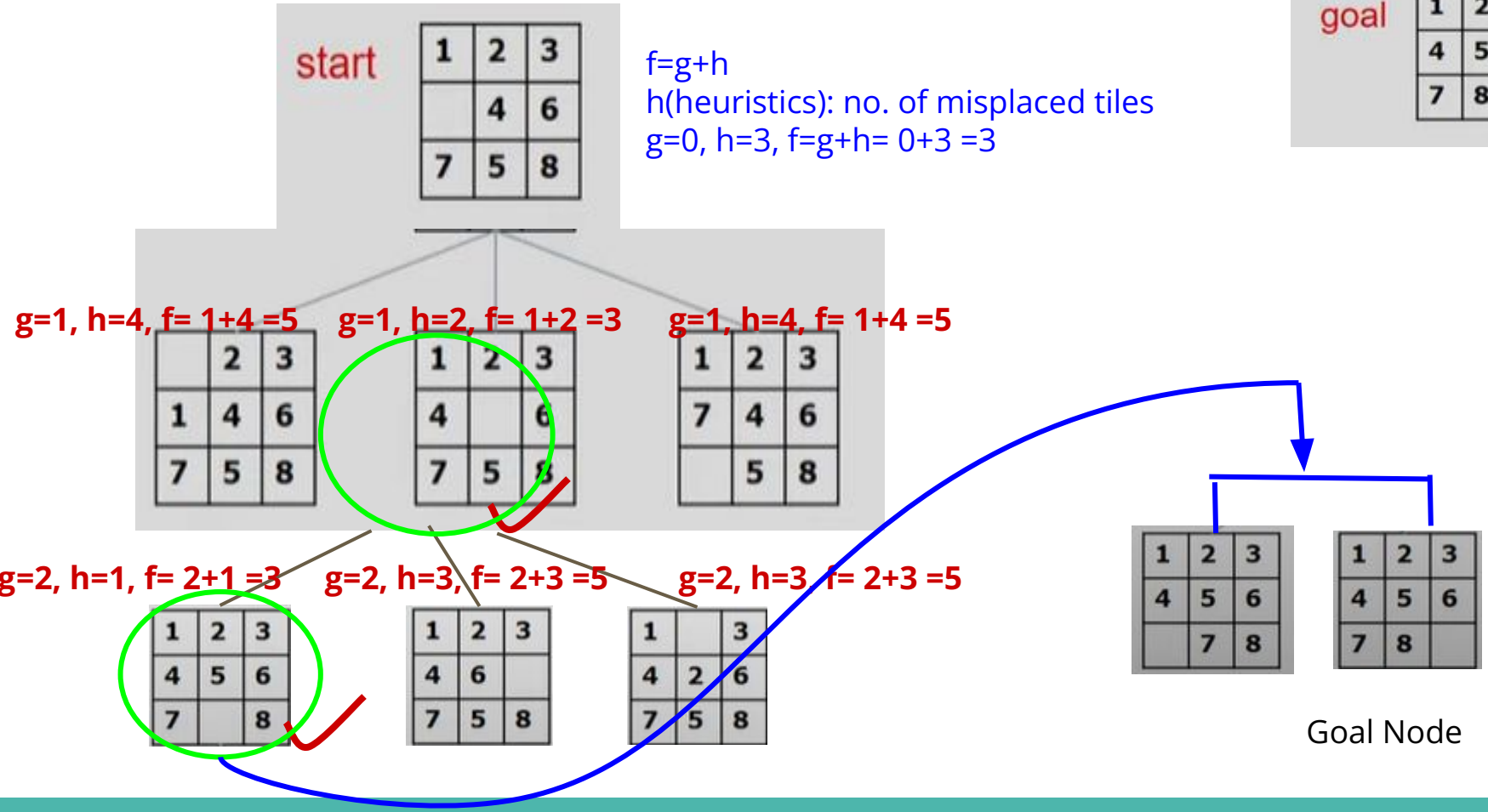
The **first condition** we require for optimality is that h(n) be an **admissible heuristic.**
An admissible heuristic is one that **never overestimates** the cost to reach the goal.
Because **g(n)** is the actual cost to reach **n** along the current path, **and f(n) = g(n) + h(n),** we have as an immediate consequence that f(n) never overestimates the true cost of a solution along the current path through **n**.
Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.

Conditions for optimality: Admissibility and consistency

A **second, slightly stronger condition called consistency (or sometimes monotonicity)** is required only for applications of A∗ to graph search.
A heuristic **h(n)** is consistent if, for every node **n** and every successor **n'** of **n** generated by any action **a**, the estimated cost of reaching the goal from **n** is no greater than the step cost of getting to **n'** plus the estimated cost of reaching the goal from **n'**:
$h(n) \leq c(n, a, n') + h(n')$.

Example Solving 8-Puzzle using A*





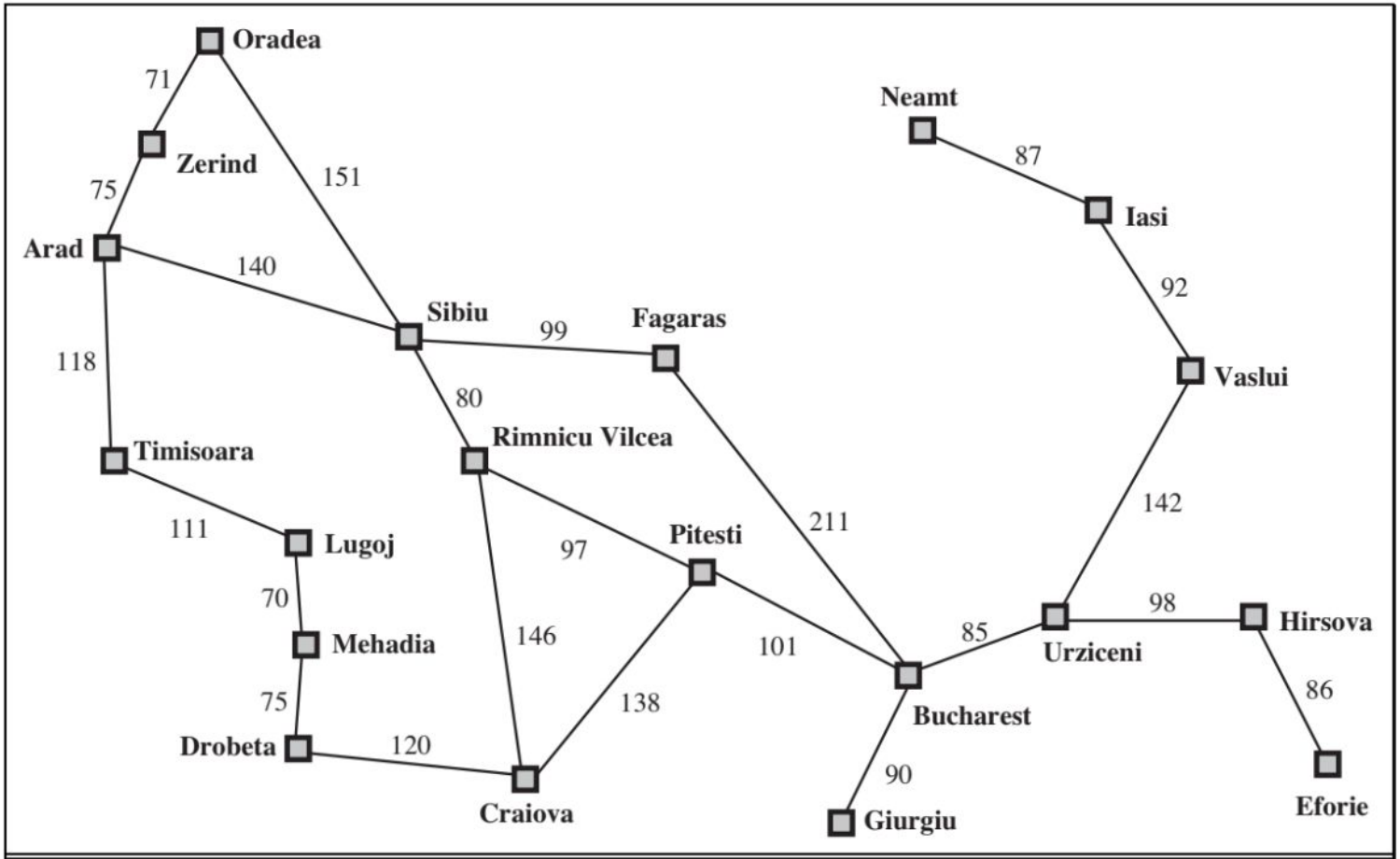
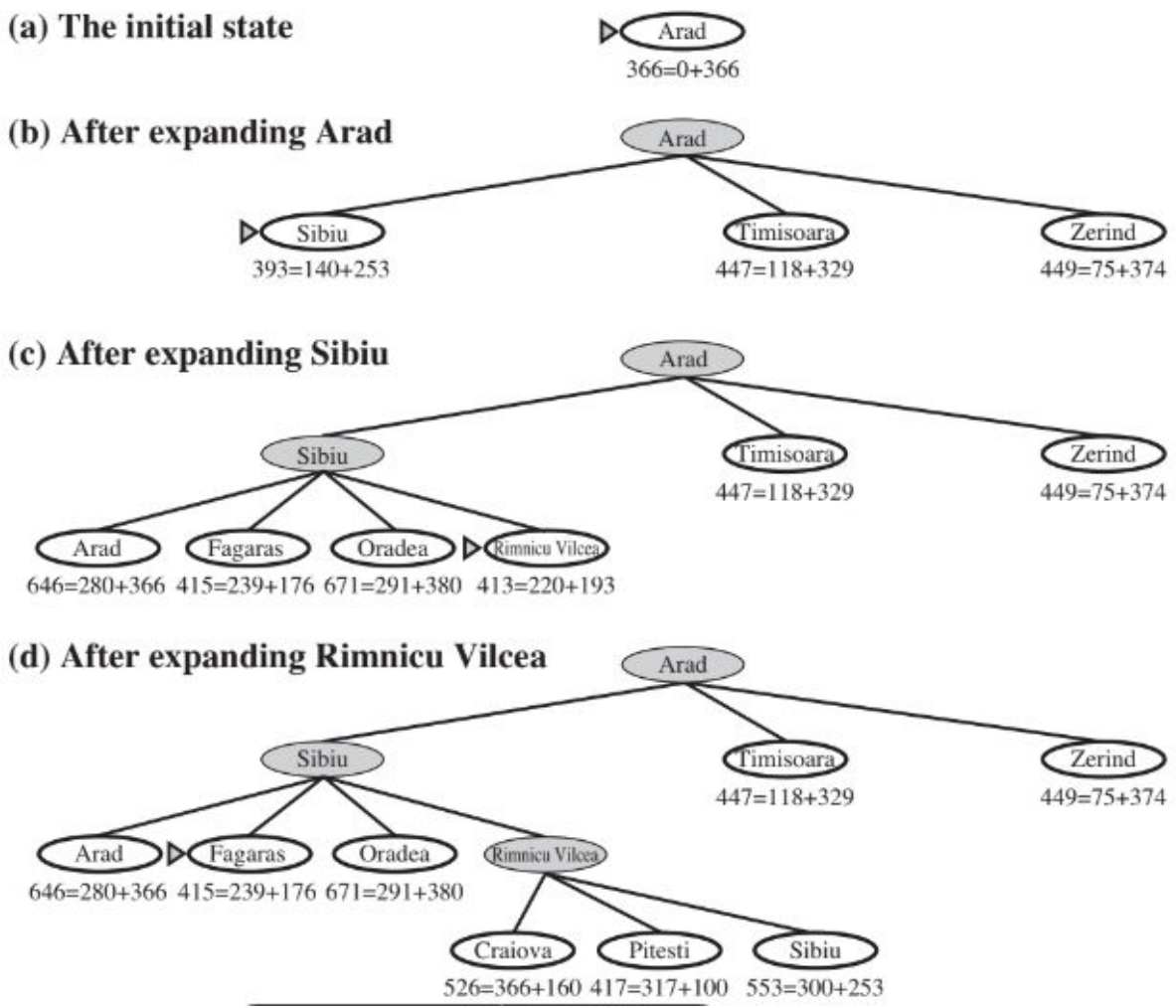


Figure 3.2 A simplified road map of part of Romania.

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

A* Search


(a) The initial state
Arad
366=0+366
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
393=140+253
447=118+329
449=75+374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
415=239+176
671=291+380
413=220+193
(d) After expanding Rimnicu Vilcea
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
415=239+176
671=291+380
Craiova
Pitesti
Sibiu
526=366+160
417=317+100
553=300+253

A* Search

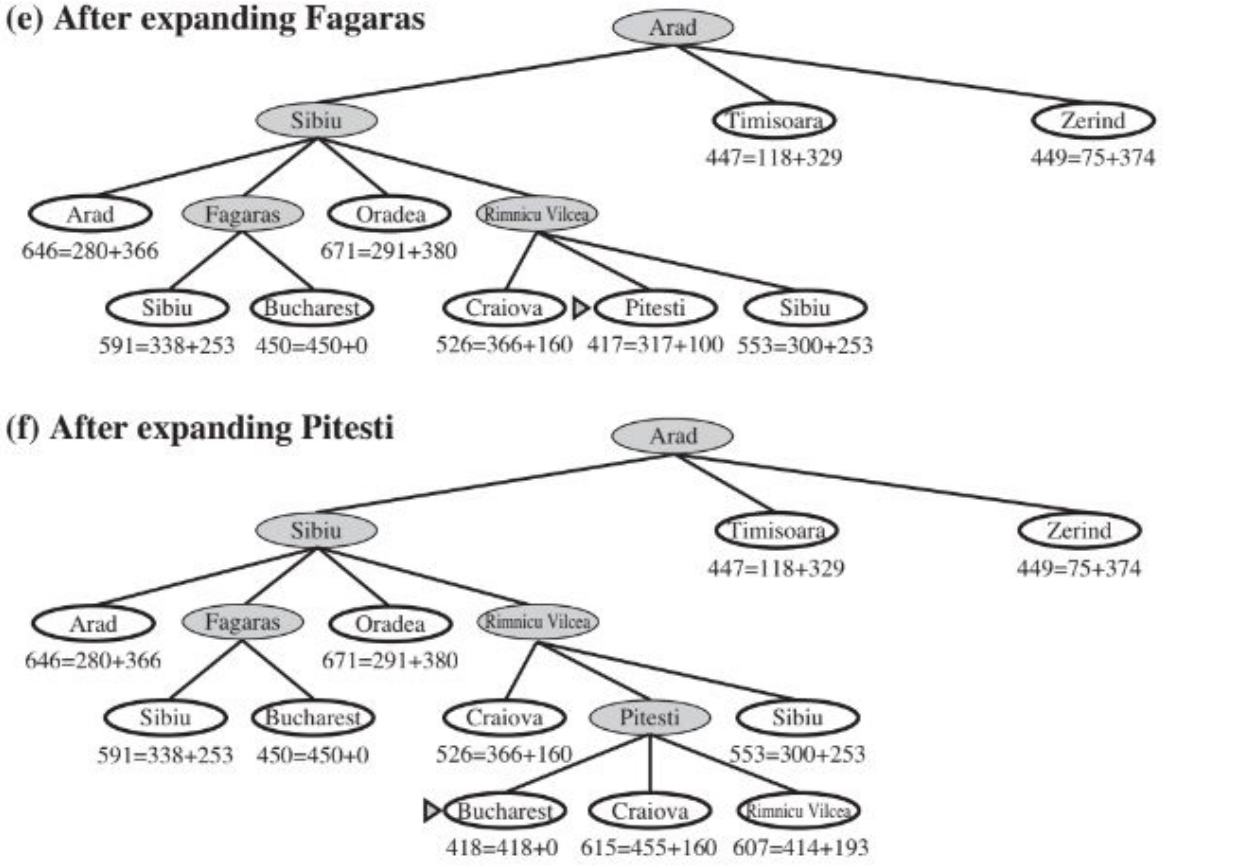
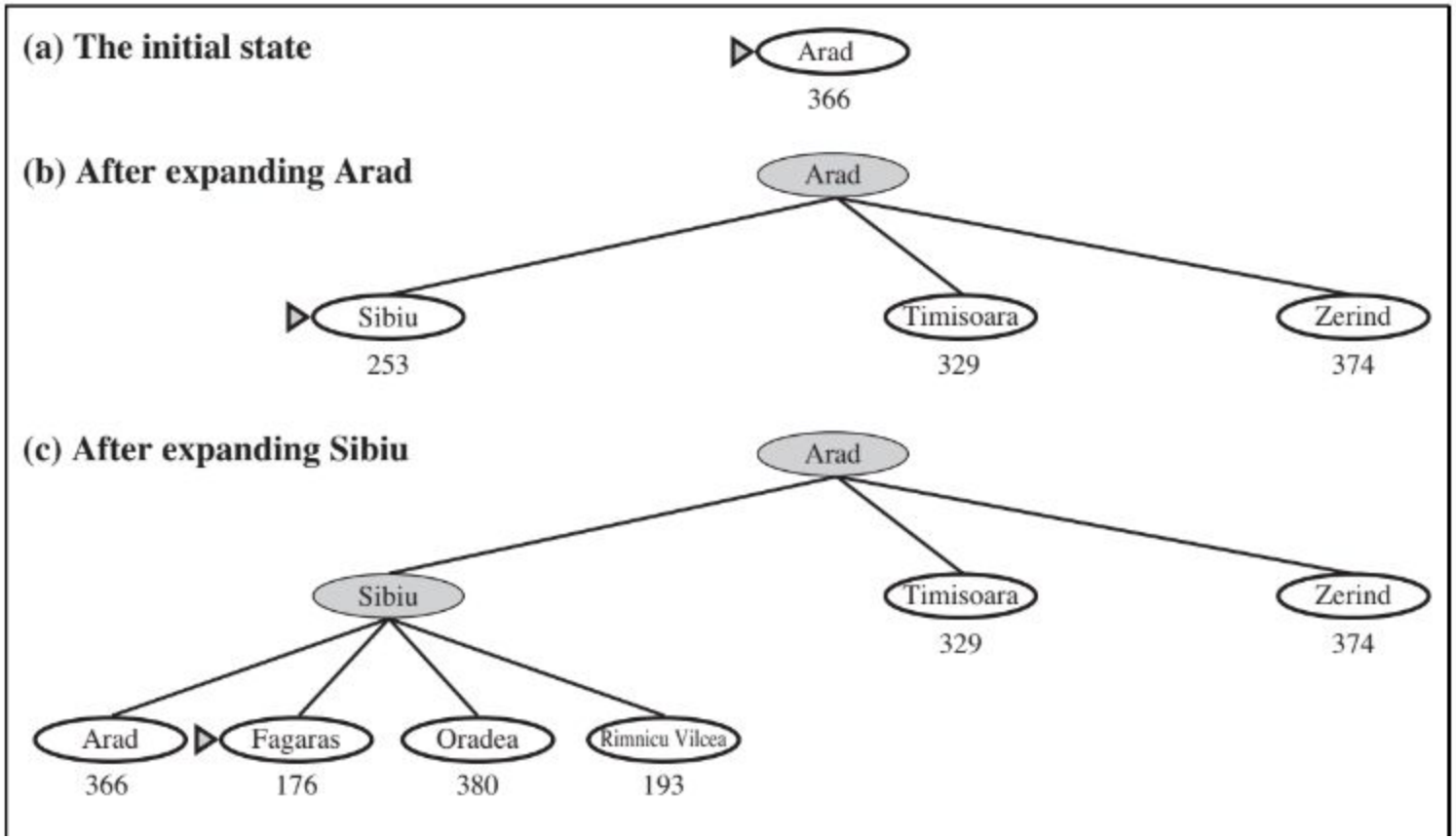
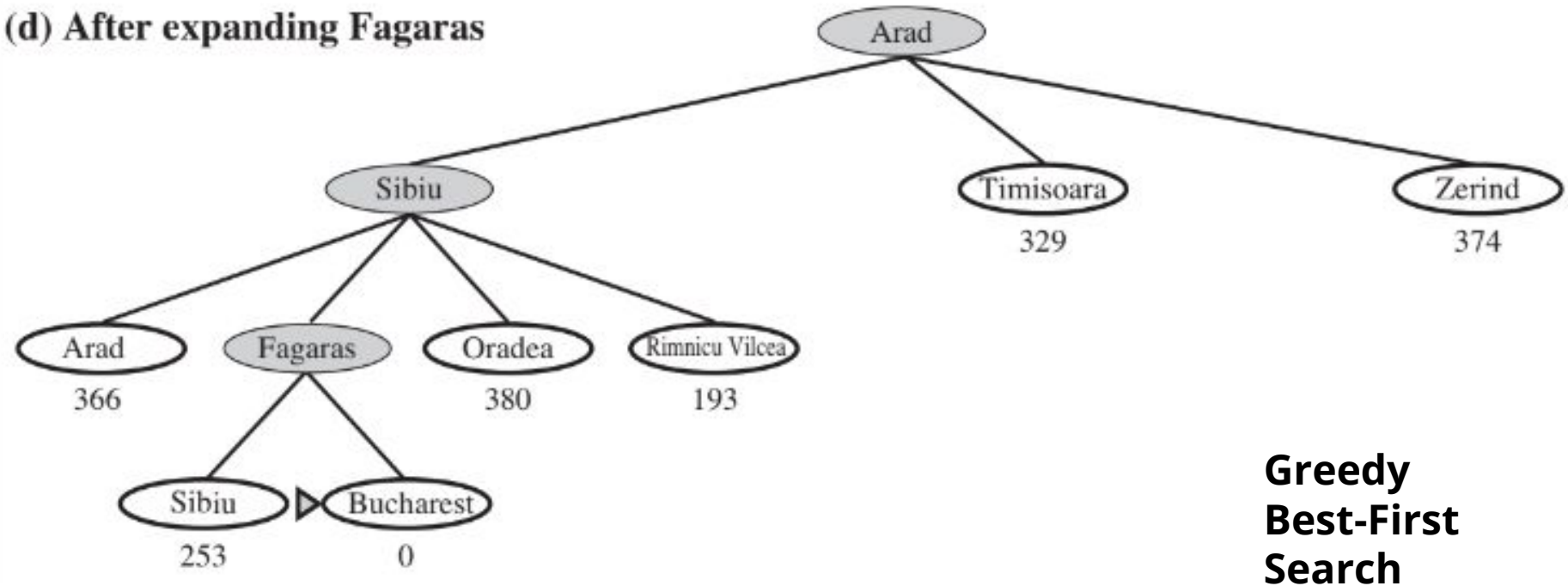


Figure 3.24 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 3.22.



Greedy Best-First Search



Greedy Best-First Search

Local Search-
Hill Climbing

https://docs.google.com/presentation/d/1H4719x9pPx2hvE1KkNSC01cu7Jom4pXYEPJtcglaTXY/edit?usp=sharing

Problem Reduction

AO* Algo

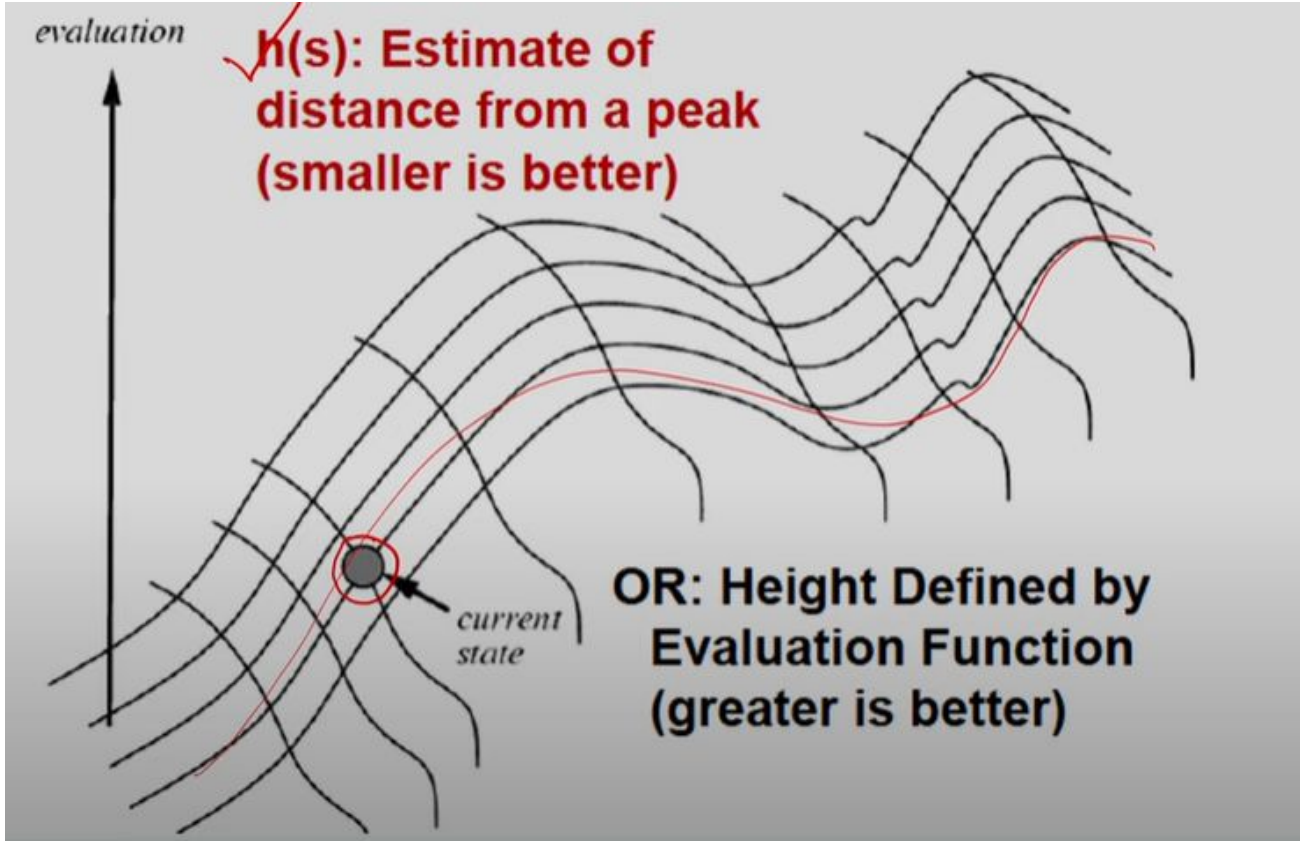
https://www.geeksforgeeks.org/ao-algorithm-artificial-intelligence/

Local Search Algorithms

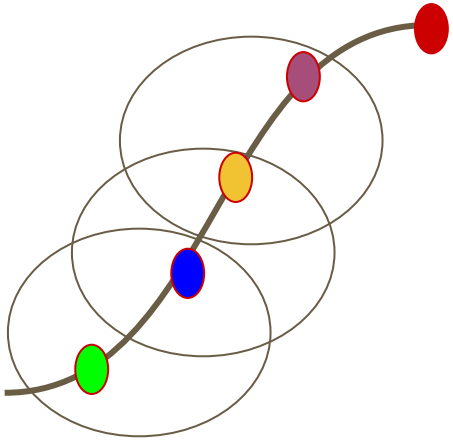
- Local search:
  - Use single current state and move to neighboring states.
- Idea: start with an initial guess at a solution and incrementally improve it until it is one
- Advantages:
  - Use very little memory
  - Find often *reasonable* solutions in large or infinite state spaces.
- Useful for pure optimization problems.
  - Find or approximate best state according to some objective function

Hill Climbing- A Local Search Algorithm

Hill climbing is a variant of Generate - and-Test in which feedback from the test procedure is used to help the generator decide which direction to move in the search space.



Hill Climbing Algo



Procedure: Hill Climbing

Initialize **current** to starting state

Loop:

Let **next** = highest valued successor of **current**

If value(**next**) < value(**current**) return **current**

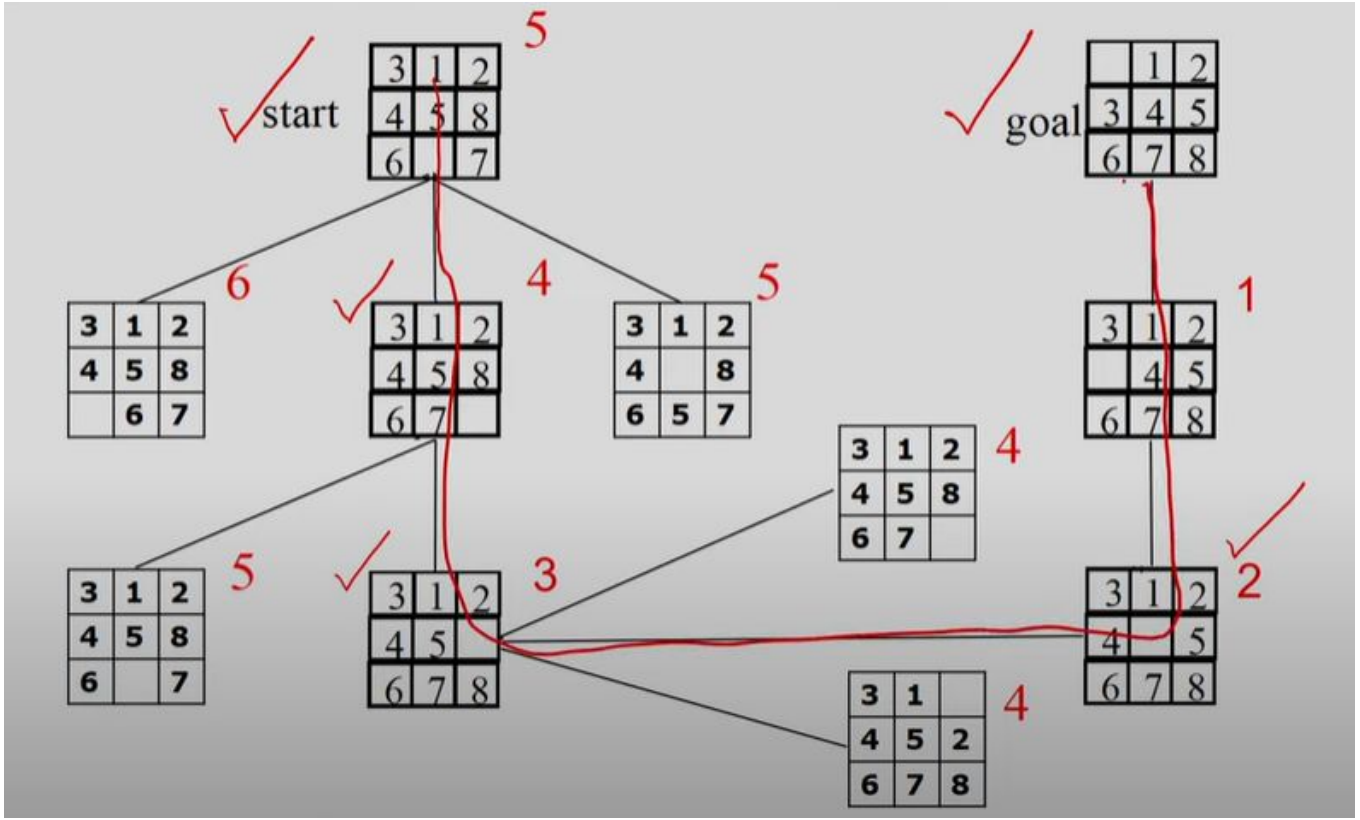
Else let **current=next**

Hill Climbing Algo

Variants-

- **Simple Hill Climbing -**
  - Basic method in which the first state that is better than the current state is selected.
- **Steepest - Ascent Hill Climbing -**
  - Consider ALL moves from the **current** state and select the best one as the **next** state.

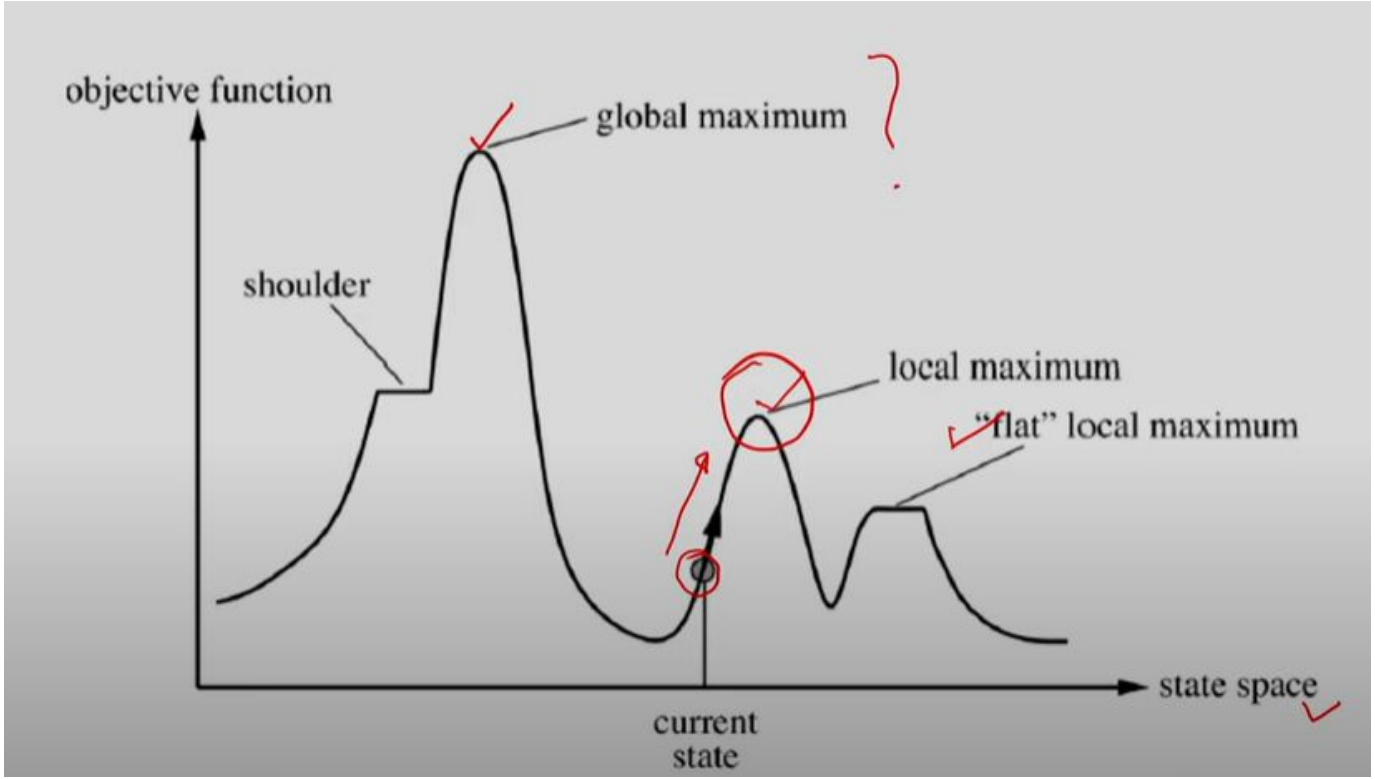
Hill Climbing Search



Hill Climbing Search

- Properties:
  - Terminates when a peak is reached
  - Does not look ahead of the immediate neighbours of the current state.
  - Chooses randomly among the set of best successors, if there is more than one.
  - Does not backtrack, since it does not remember where its been.

Hill Climbing - Search space features



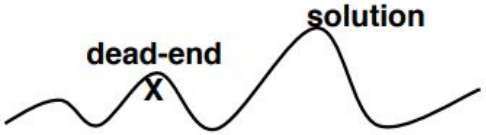
Problems with Hill Climbing

- **Local Maxima:** peaks that are not the highest points in the space
  - State that is better than all its neighbours but is not better than some other states farther away.
- **Plateaus:** the has a broad flat region that gives the search algorithm no direction (random walk)
  - Whole set of neighbouring states have same values! Not possible to determine the best direction.
- **Ridges:** dropoffs to the sides;

Problems with Hill Climbing

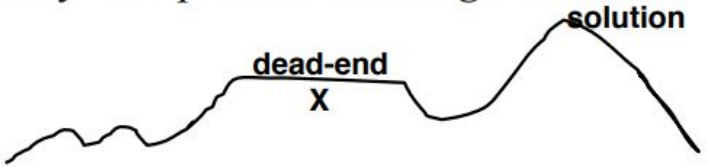
- Can get stuck at a *local maximum*.

Local maximum



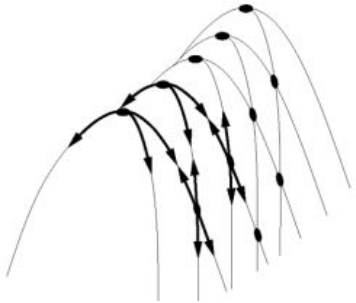
- Unable to find its way off a *plateau with single moves*.

Plateau



- Cannot climb along a narrow *ridge* when almost all steps go down (continuous space).

Ridge/Knife edges



Especially serious when you can not evaluate all potential moves

Some ways to overcome the drawbacks of Hill Climbing-

- **Backtrack** to some earlier node and try going to some different direction-good way of dealing with the **Local maxima.**
- Make **big jump** in some direction to try to get to a new section of the search space- good way to dealing with the **Plateaus.**
- Apply two or more rule before doing the test. This corresponds to **moving in several directions at once-** this is a particularly good strategy for dealing with **Ridges.**

Simulated Annealing- a variation of Hill Climbing

In **Simulated Annealing,** at the beginning of the process, some **downhill** moves may be made.

The idea is to do enough exploration of the whole space early on so that the final solution is relatively insensitive to the starting state.

This should **lower the chances of getting caught at a local maximum, a plateau, or a ridge."**

Sometimes take downhill steps **You may have to get worse to get better!!**

Simulated Annealing-

The three differences are: -

- The annealing schedule must be maintained
- Moves to worse states may be accepted
- It is good idea to maintain, in addition to the current state, the best state found so far.

Simulated Annealing-

In this we attempt to **minimize** rather than maximize the value of the **objective function (heuristic function).**

Thus this process is one of **valley descending** in which the object function is the **energy level.**

Simulated annealing as a computational process is patterned after the physical process of **annealing**, in which physical substances such as metals are melted and then gradually cooled until some solid state is reached.

Physical substances usually move from higher energy configurations to lower ones, so the valley descending occurs naturally.

Simulated Annealing-

But there is some possibility that a transition to a higher energy state will occur.

Probability of transition to higher energy state is given by function:

$$P = e^{-\Delta E/kt}$$

Where **ΔE** is the **positive change in the energy level**, **T is the temperature and K is Boltzmann constant.** The algorithm for simulated annealing is slightly different from the simplehill climbing

Algorithm: Simulate Annealing

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. Initialize BEST-SO-FAR to the current state.

3. Initialize T according to the annealing schedule

4. Loop until a solution is found or until there are no new operators left to be applied in the current state.

Continued on next slide.....

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. Compute:

- ΔE = (value of current) – (value of new state)
- If the new state is a goal state, then return it and quit.
- If it is a goal state but is better than the current state, then make it the current state. Also set BEST-SO-FAR to this new state.
- If it is not better than the current state, then make it the current state with probability p' as defined above. This step is usually implemented by invoking a random number generator to produce a number in the range [0, 1]. If the number is less than p', then the move is accepted. Otherwise, do nothing.

c. Revise T as necessary according to the annealing schedule

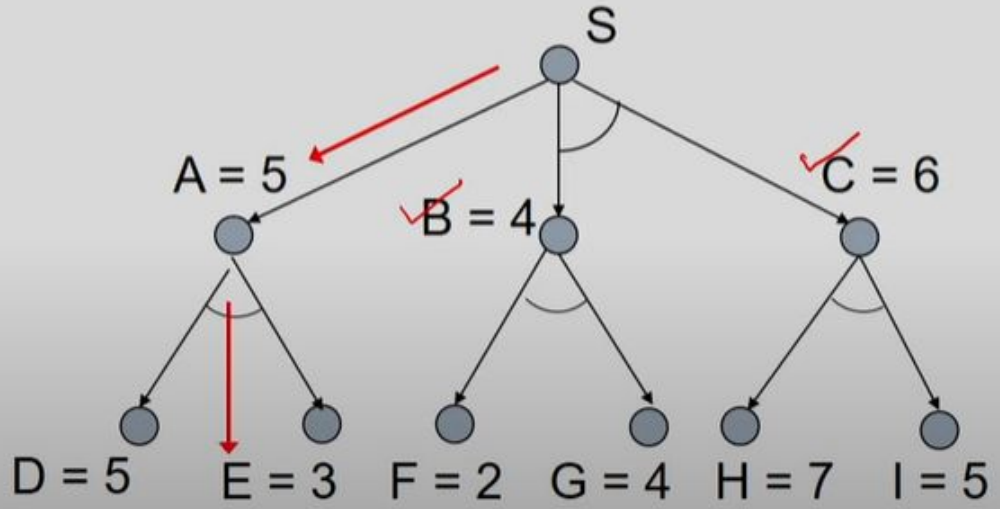
5. Return BEST-SO-FAR as the answer

Best-First Algorithm ?

- To find solution in an AND-OR graph we need an algorithm similar to best-first search; but with the ability to handle the AND arc appropriately.

Edge cost = 1 unit.
Best-first Search ? Edge B-FG

Involving B-FG; Also C-HI
Path S-A; A-DE Better



AO* : A Heuristic Search Procedure

Use a heuristic function h(n); estimate of the cost of an optimal solution from node n to set of terminal nodes.

Each node will have an associated **h value**; serves as measure of goodness of the node.

AO* doesn't explore all the solution paths once it got a solution.

Need an algorithm **similar to best first search** but with the ability to handle the AND arcs appropriately.

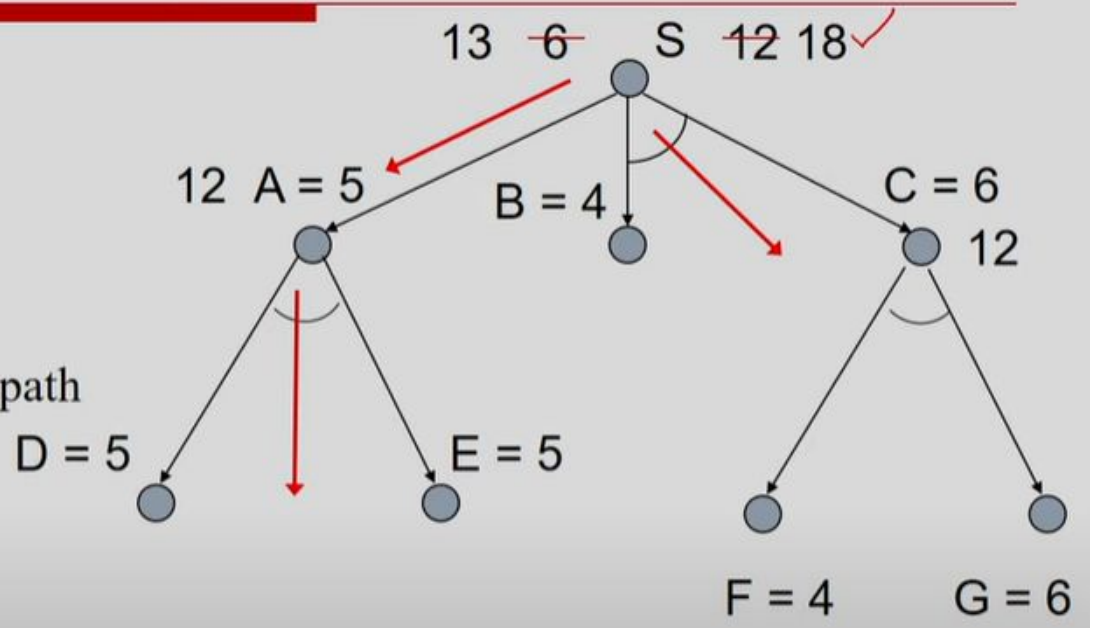
AO*: A Heuristic Search Procedure

More promising?

After node C is expanded;
Nodes – F and G

How promising?

Evaluate h values – revise on path



Nodes – D and E

How promising?

Evaluate h values – back-up

Edge cost = 1 unit.

Basic Idea of AO*

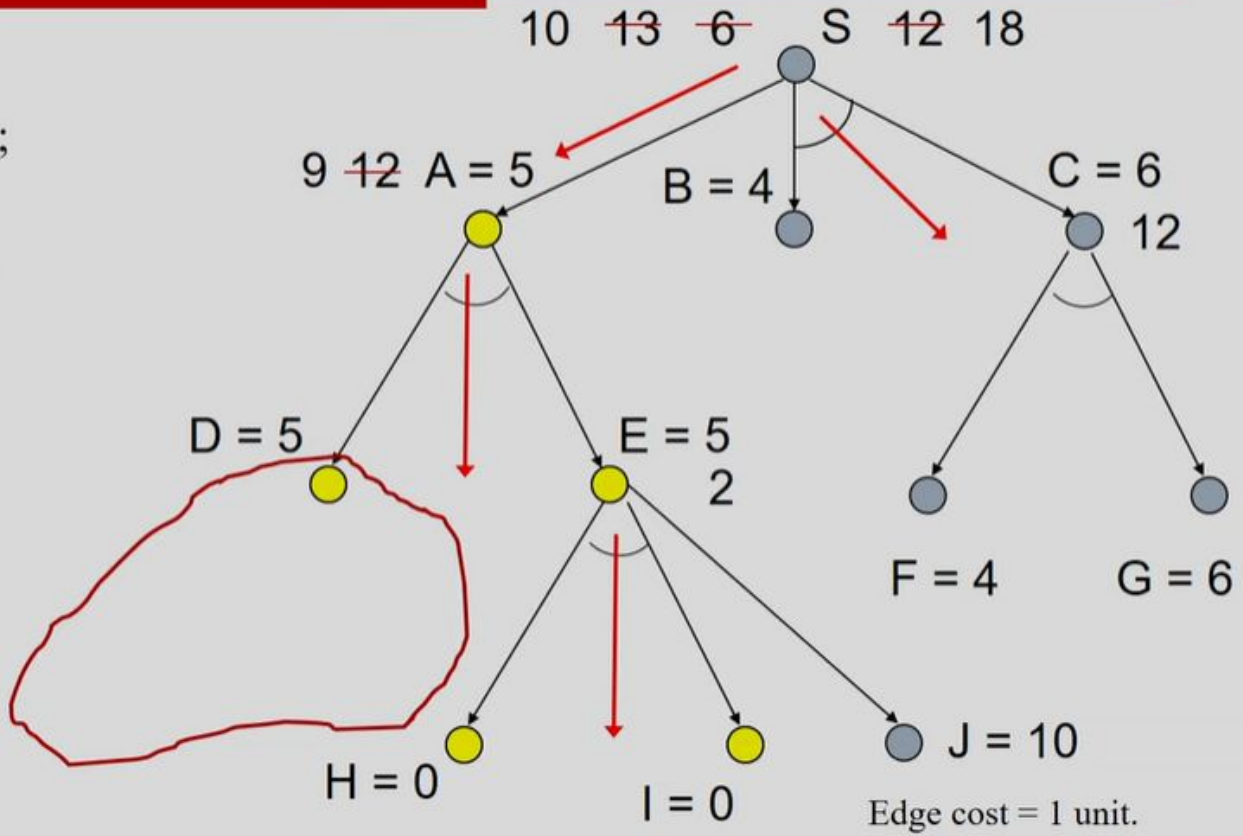
- Top-down graph growing
  - Picks out best available partial solution sub-graph from explicit graph by tracing down *marked* connectors.
    - Here *marked* – indicates the current best partial solution graph from each node in the search graph.
  - One of the nonterminal leaf nodes of this best partial solution graph is expanded.

AO*: A Heuristic Search Procedure

Node E is expanded;
Nodes – H-I; J

SOLVE - Labelling

Sub-tree at Node D
SOLVED!



Basic Idea of AO*

- Bottom-up cost-revising, connector-marking
  - SOLVE-labeling.
  - If a direction has all successors SOLVED then n is marked SOLVED.
  - Starting with the node just expanded, the procedure revises its cost (using the newly computed cost of its successors).
    - Marks the outgoing connector on the estimated best path to terminal nodes.
    - Propagated upward in the graph.

AO* Algorithm

Create a search graph G = <s> ; q(s) = h (s) ; If s is a terminal node, mark s SOLVED

Until s labeled SOLVED do:

Top-down graph growing - picks out best available partial solution sub-graph

begin

a. **Compute G/ partial solution graph in G** by tracing down marked connectors in G from s.

b. **Select any nonterminal leaf node n of G/.**

c. **Expand n**, place successors in G, For each successor not already in G let q(successor) = h(successor). Label SOLVED all successors that are terminal nodes.

d. Let S := {n}.

Until S is empty do :

begin

a. Remove a node, m, from S which has no descendent in G also in S (minimal node).

b. Revise cost for m

q(m)=min [c +q(n1)+…+q(nk)].

Mark chosen connector.

If **all successors through the connectors are SOLVED** then mark m SOLVED.

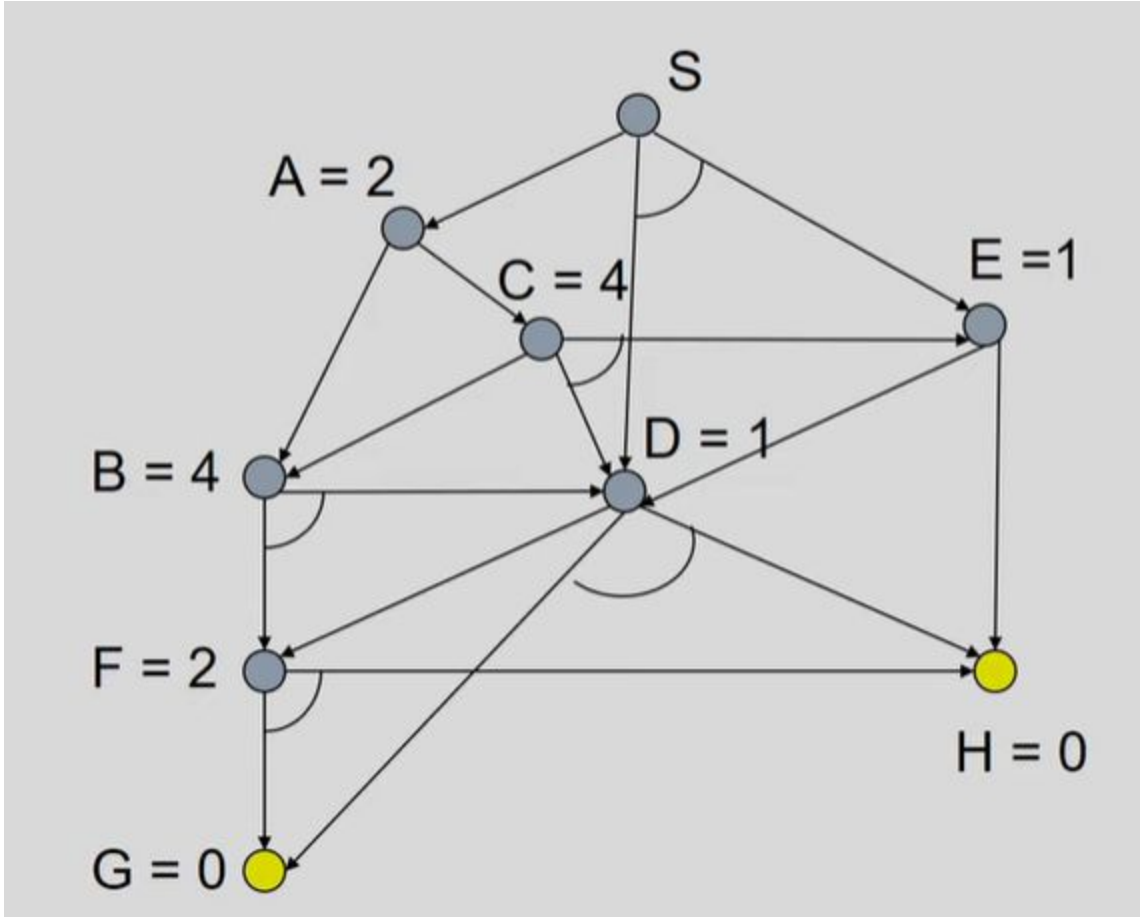
c. **If m SOLVED or changed q(m) then add to S all "preferred" parents of m.**

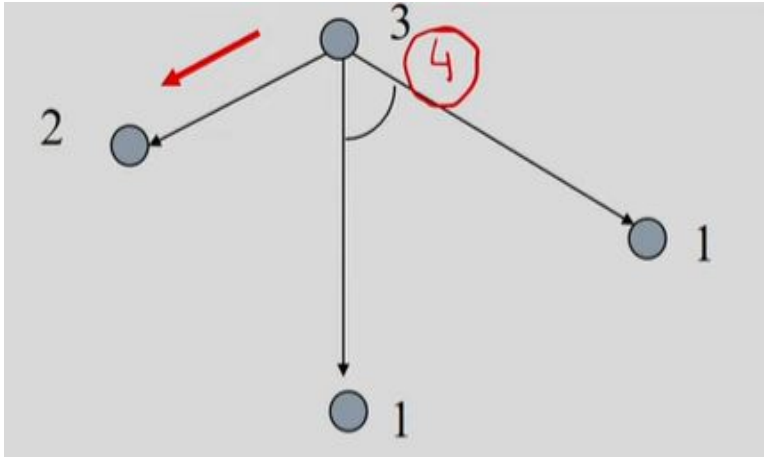
End.

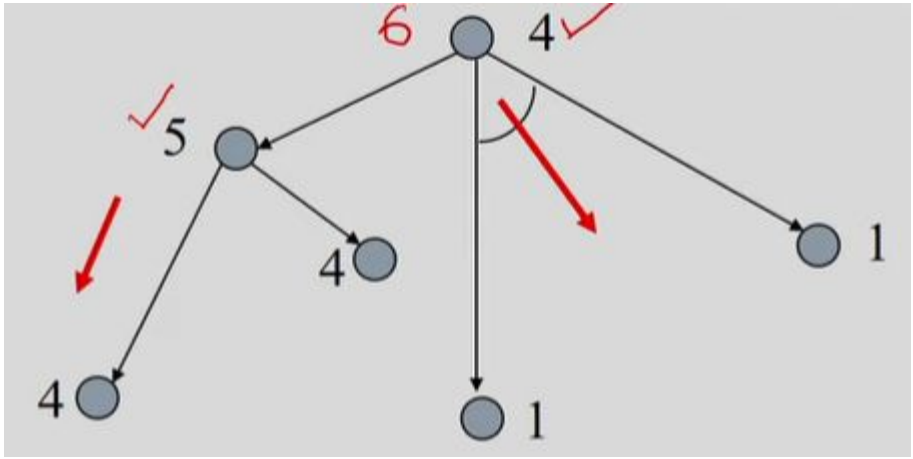
End.

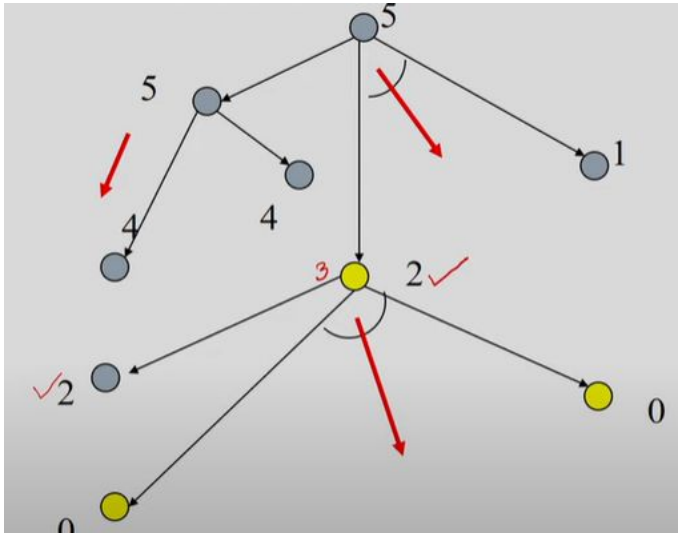
Bottom-up, cost-revising, connector marking; SOLVED labelling procedure

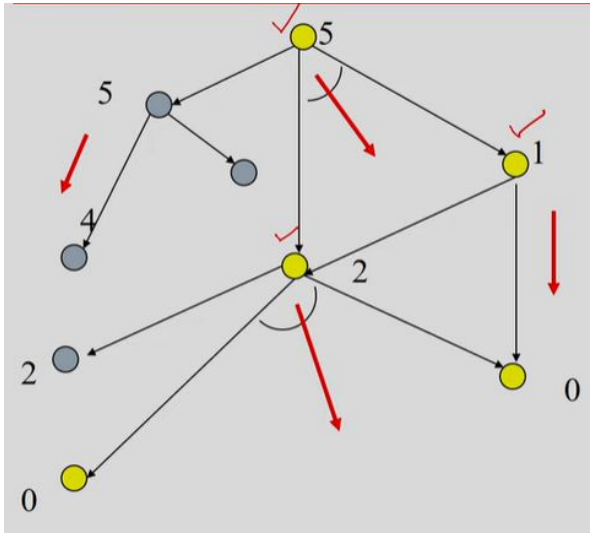
Example- AO*



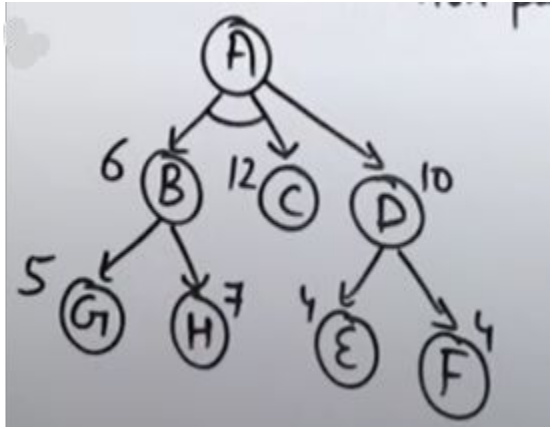

3
4
2
1
1

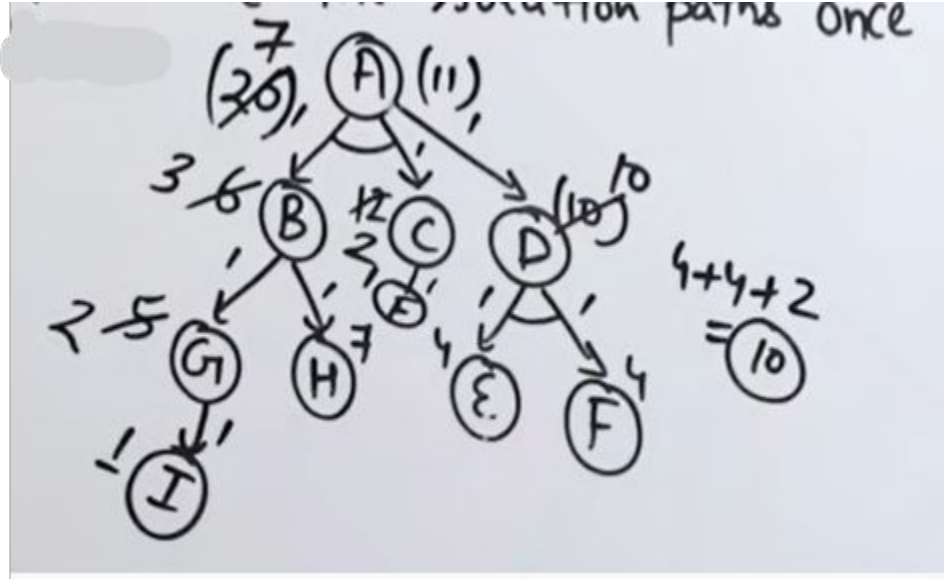

6
4
5
4
1
4
1


5
5
1
4
4
3
2
2
0
0


5
5
1
4
2
2
0
0

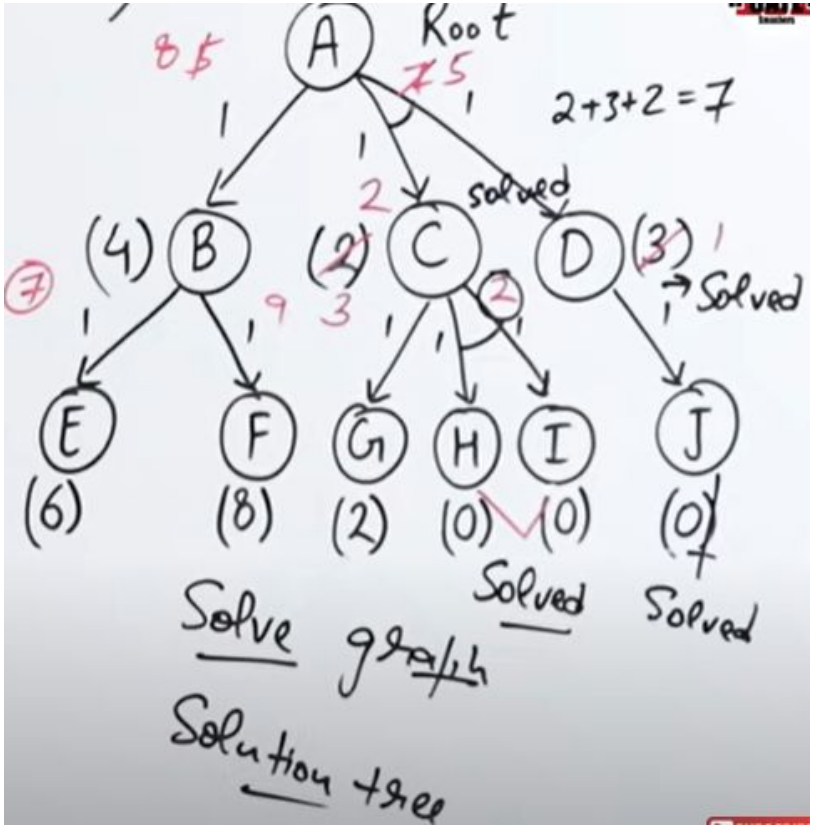
Example- AO*





Cost of each edge is 1.

AO*


A
Root
2+3+2=7
solved
B
C
D
(4)
(2)
(3)
Solved
E
F
G
H
I
J
(6)
(8)
(2)
(0)
(0)
(0)
Solved
Solved
Solve graph
Solution tree