STATE SPACE SEARCH

Artificial Intelligence

Joaquim Filipe

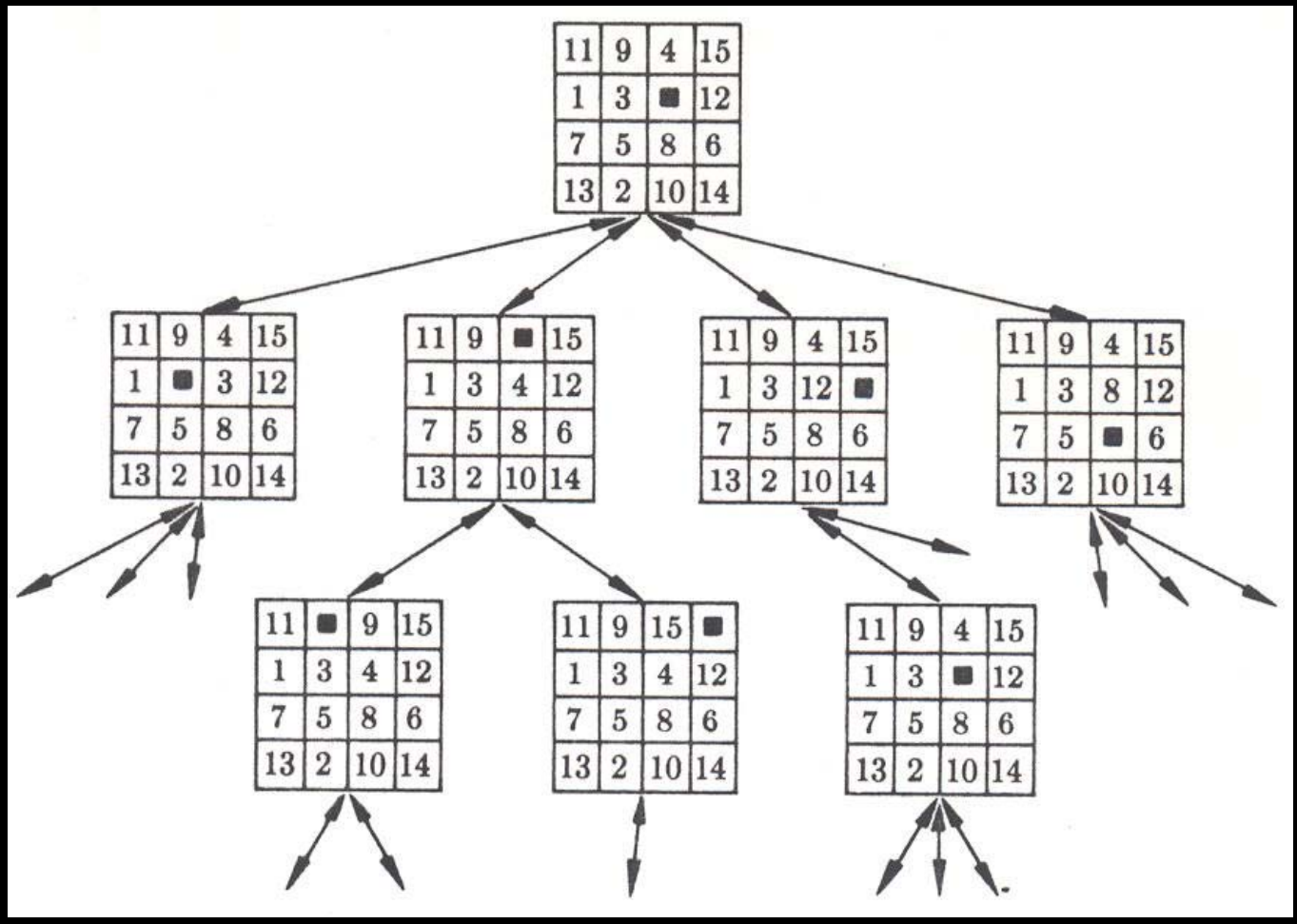
TROUBLESHOOTING

- Artificial intelligence is intended to build machines to help solve problems that people are currently better at.
- Hence, one of the prominent topics is the study and implementation of automatic problem solving methods.

STATE SPACE

- A simple way to solve problems that cannot be solved with formulas or algorithms, is to explore the space of possibilities by trying various possible paths until you find the solution.
- In this case a problem will have to be equated in terms of:
  - States
  - Operators (of state transition)
- It has yet to be defined
  - The final state (may be with a test function)

EXAMPLE (PUZZLE OF 15)



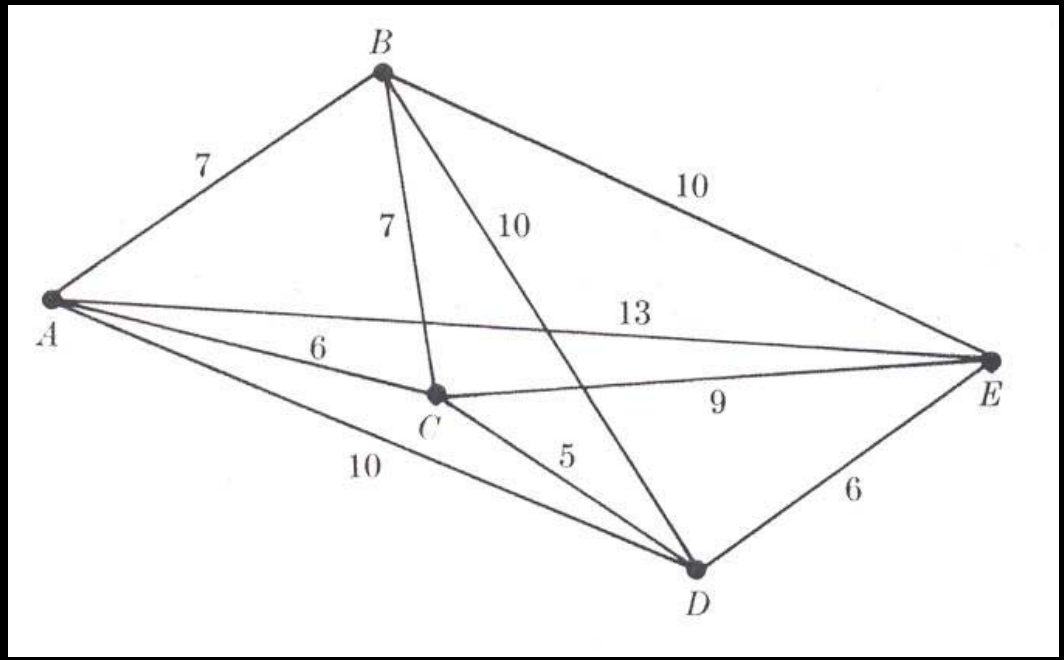
(Nils Nilsson, Problem Solving Methods in AI, p.5)

REPRESENTATION

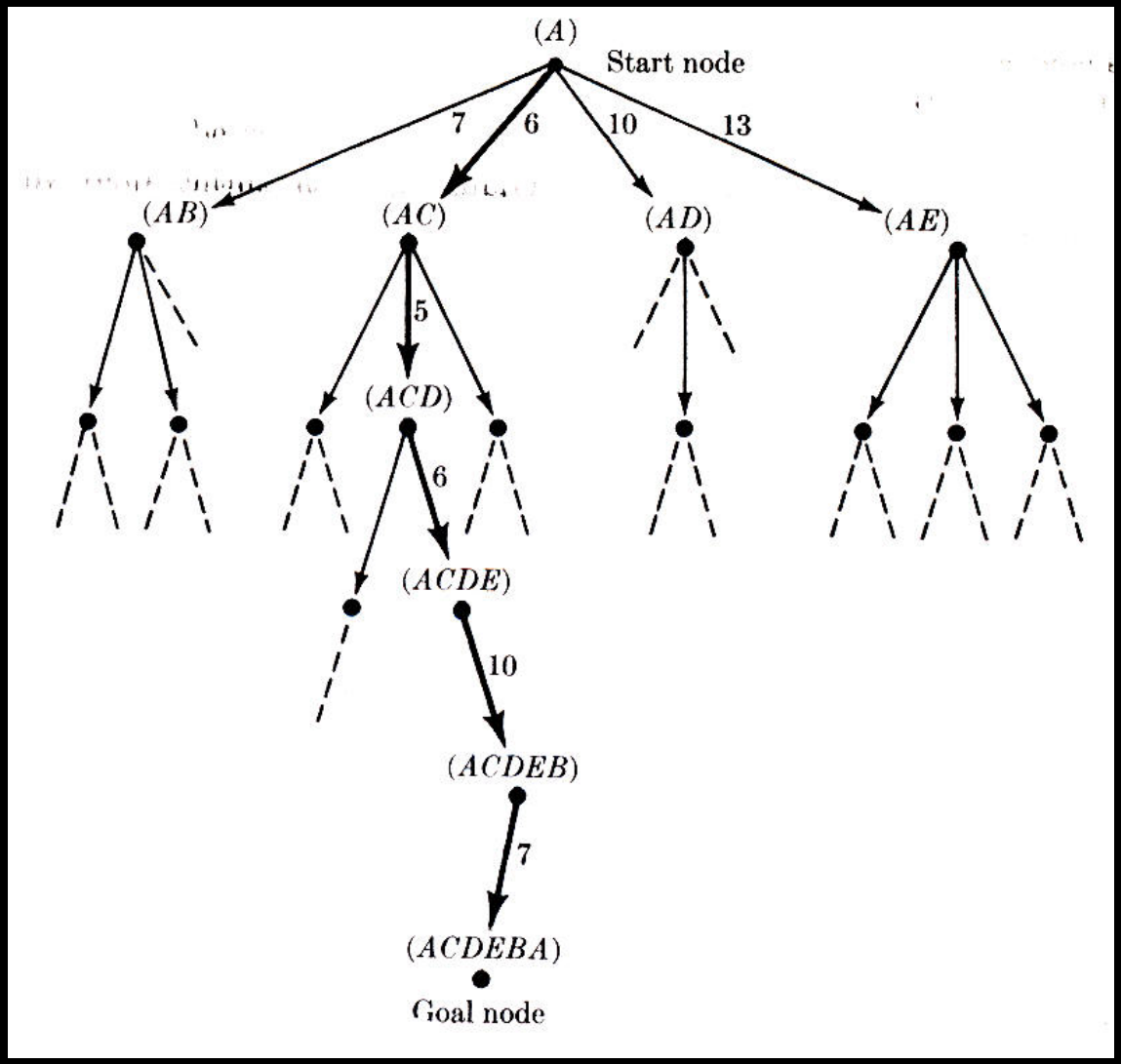
- Graph:
  - The State Space can be represented by an acyclic directed graph.
- Node:
  - each node of the graph represents a state of the problem, with some additional information (pointer to the node that generated it; etc.)
- Bow:
  - Each arc of the graph represents a state transition along the problem solving process.

EXAMPLE

- Traveling Salesman.
  - Classic problem: A traveling salesman has to plan a trip where he visits n cities only 1 time and returns to the home city, minimizing a cost (usually the distance traveled).
- Graphic representation:



PROBLEM GRAPH



(Nils Nilsson, Problem Solving Methods in AI, p.5)

REPRESENTATION OF A STATE

- A state in the traveling salesman problem can be represented as a set of 2 elements: {C, V}
  - C = Set of cities to visit.
  - V = Set of cities already visited.
- The final state is a state in which
  - C = Ø
- The initial state is the state in which C contains all cities and V = Ø.

SOLUTION

- The solution is obtained by applying operators to the state descriptions until the final state description emerges.

TREE HARVESTING STRATEGIES

- Concepts:
  - Start Node
  - Successors of a node: nodes that are generated by applying one of the legal operators.
  - Expanding a node: generating all successors
  - Pointers to the parent node: to allow you to immediately get the solution from the final state
  - List of open (nodes): list with the nodes that have not yet been expanded
  - List of closed (nodes): list with the nodes that have already been expanded

SEARCH METHODS: *BREADTH-FIRST*

1. Start node => OPEN
2. If OPEN is empty it fails.
3. Removes the first node from OPEN (n) and places it in CLOSED
4. Expands node n. Place successors at the end of OPEN, placing pointers to n.
5. If any of the successors is an objective node go out, and give the solution. Otherwise it goes to 2.

BREADTH-FIRST FLOWCHART

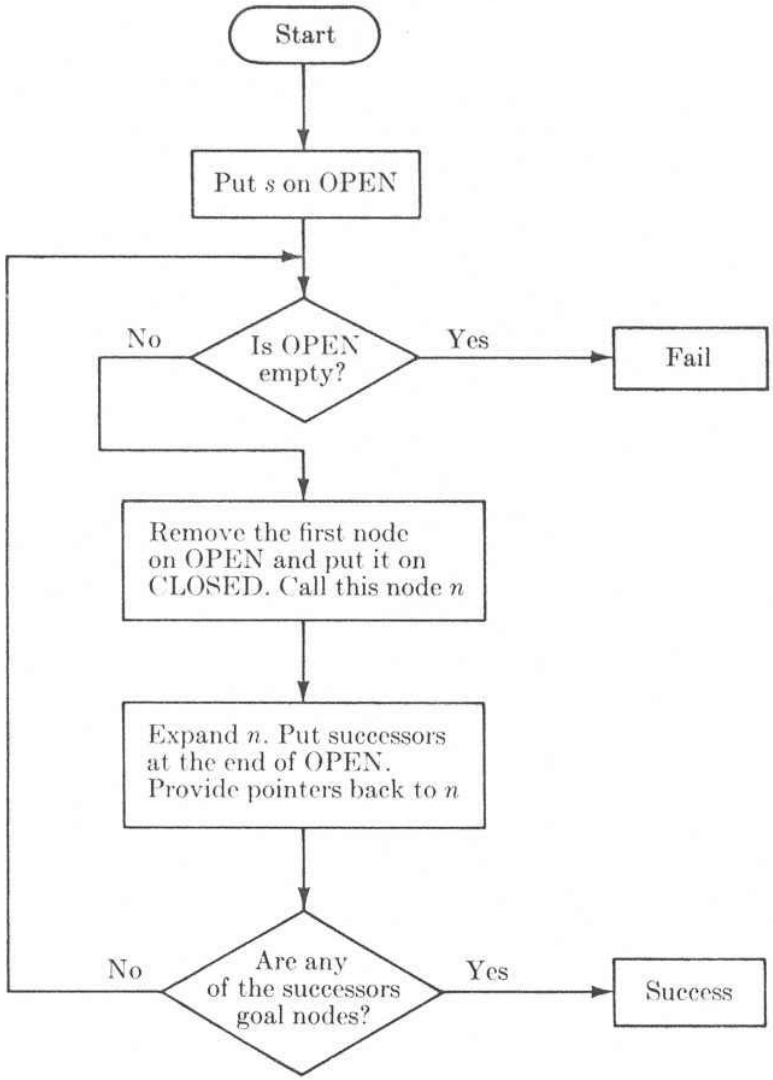


FIG. 3-1 *Flow chart of a breadth-first search algorithm for trees.*

BF FEATURES

- It is assumed that the initial node is not an objective node.
- The BF always finds the solution that corresponds to the shortest path.
- If there is no solution the method terminates with failure if the graph is finite or does not terminate if the graph is infinite.

SEARCH METHODS: UNIFORM COST

- If you are interested in minimizing cost rather than distance, and if the costs associated with the arcs are different from arc to arc, then you need to use a variant of the BF called the "uniform cost method" that guarantees cost minimization.

ALGORITHM

1. Start node(s) => OPEN. Make g(s)=0.
2. If OPEN is empty it fails.
3. Removes the node from OPEN (n) with the lowest cost (g) and puts it in CLOSED
4. If n is an objective node it terminates and gives the solution.
5. Expand the node n. Set the successors to OPEN, placing the pointers to n and calculating the g of each of the successors.
6. Go to 2.

SEARCH METHODS: DEPTH-FIRST (DEPTH-FIRST)

- The root node depth is conventionally set to zero.
- The depth of a node is 1 + the depth of its predecessor.
- A maximum depth level is set at which nodes are not expanded

ALGORITHM

1. Start node => OPEN
2. If OPEN is empty it fails.
3. Removes the first node from OPEN (n) and places it in CLOSED
4. If the depth of n is greater than d it goes to 2.
5. Expands node n. Place successors at the beginning of OPEN, placing pointers to n.
6. If any of the successors is an objective node go out, and give the solution. Otherwise it goes to 2.

FLOWCHART

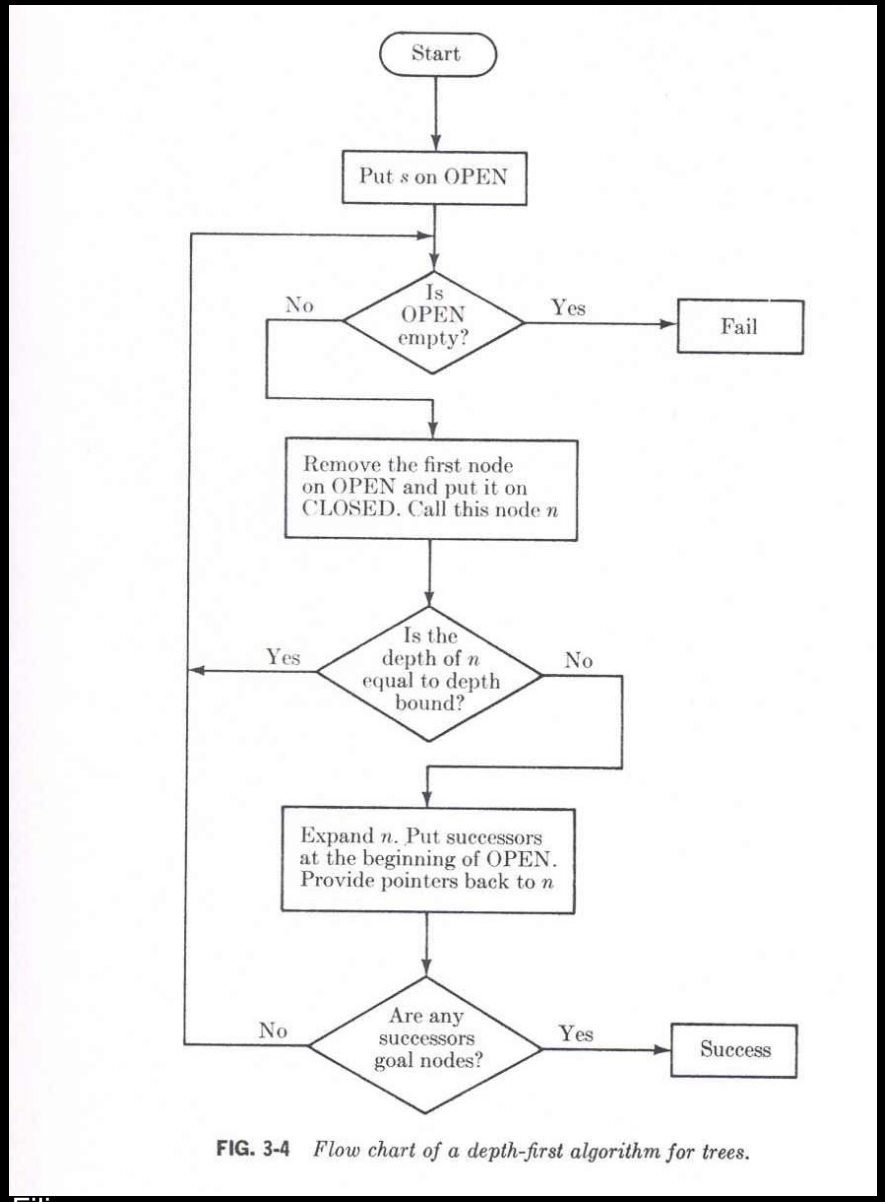


FIG. 3-4 *Flow chart of a depth-first algorithm for trees.*

CONTINUED

- DF: when successors are generated that are already OPEN or CLOSED it may be necessary to recalculate the depth of the corresponding nodes.

GRAPHS INSTEAD OF TREES

- The previous methods assume that the state space has a tree-like structure.
- If the state space is a graph you need to modify the algorithms:
  - Breadth-first: recognize whether a successor state is already in OPEN or CLOSED and in that case do not put the corresponding node in OPEN.
  - Uniform cost:
    - If the successor (n_{suc}) is in OPEN n_{suc} is not added if $g(n_{suc}) > g(n_a)$ otherwise n_{suc} replaces n_a.
    - If the successor is in CLOSES ignore n_{suc}.

CONT.

Depth-first

- If a node with the same state is open, that means that this node has a cost less than or equal to that of the node generated now. Abandon the generated node.
- If it is in closed and has a higher or equal cost:
  - It deletes the old node and places the generated node in open and places pointers on the successors from the old node to the generated node.
- Otherwise:
  - leaves the generated node.

COMBINATORIAL EXPLOSION

- The BF, Uniform Cost and DF methods do an exhaustive search, so they are called blind or uninformed methods.
- For many problems this exhaustive search becomes impractical, not solving the combinatorial explosion problem.
- A smarter alternative is needed.

HEURISTIC OR "INFORMED" METHODS

- Sometimes it is possible to use empirical rules to speed up the search.
  - The central idea is to avoid considering all alternatives, focusing attention only on those that are of most interest.
  - Need to evaluate the "interest" of the nodes: *evaluation functions*.
  - These rules are specific to the problem at hand and do not always work.
- Methods that use this kind of knowledge: heuristic search methods. Also called: informed search methods.

USING EVALUATION FUNCTIONS

- It is considered that an evaluation function of node interest f(n) can be defined.
- By convention the list of OPEN nodes is sorted in ascending order of f(n), where f(n) is the value of the evaluation function applied to node n.
- An algorithm that uses the previous convention to search a state space whose structure is of the acyclic graph type consists of the sequence of steps shown on the next slide.

SORTED SEARCH ALGORITHM

1. Start node(s) => OPEN. Make f(s)=0.
2. If OPEN is empty it fails.
3. Removes the node from OPEN (n) with the lowest cost (f) and places it in CLOSED
4. Expand the node n. Calculate the f of each of the successors.
5. Put the successors that do not already exist in OPEN or CLOSED in the OPEN list, in order of f by placing the pointers to n.
6. If any successor is an objective node terminates and gives the solution.
7. Associates the successors already in OPEN or CLOSED with the smaller of the values of f (existing or now calculated). Puts in OPEN the successors that were in CLOSED whose f values have fallen. Redirects to n the pointers to all nodes whose f-values have fallen.
8. Go to 2.

ALGORITHM A*

- The previous algorithm does not specify the type of evaluation function. If this consists of **f(n)=g(n)+h(n)** where g(n) is the cost of node n and h(n) is its heuristic value, this family of algorithms is called A.
- If you modify the ordered search algorithm so that the objective state test is performed on the node n that is selected after setting all successors to OPEN, you have the algorithm A*.

OPTIMAL SEARCH ALGORITHM

- The evaluation function f'(n)=g(n)+h'(n) gives an estimate of the total cost of the minimum cost path that passes through n.
- Note: The uniform cost algorithm finds the optimal (lowest cost) solution.
- h'(n)≡0 => the A* algorithm coincides with that of uniform cost, and finds the optimal solution.
- It can be shown that **if h' is a lower bound of h**, the A* algorithm continues to find the optimal solution. See pp. 60 ff. Nils Nilsson.

ADMISSIBILITY

- An algorithm is said to be admissible if, for any graph, it always finds the optimal path to the goal, as long as this path exists.
- If h' is a lower bound of h then the A* algorithm is admissible.
- Admissibility implies that:
  - When A* expands a node n it has already found an optimal path for n.
  - When A* expands a node n the evaluation function f' is not greater than the actual cost f.

HEURISTIC INFORMATION

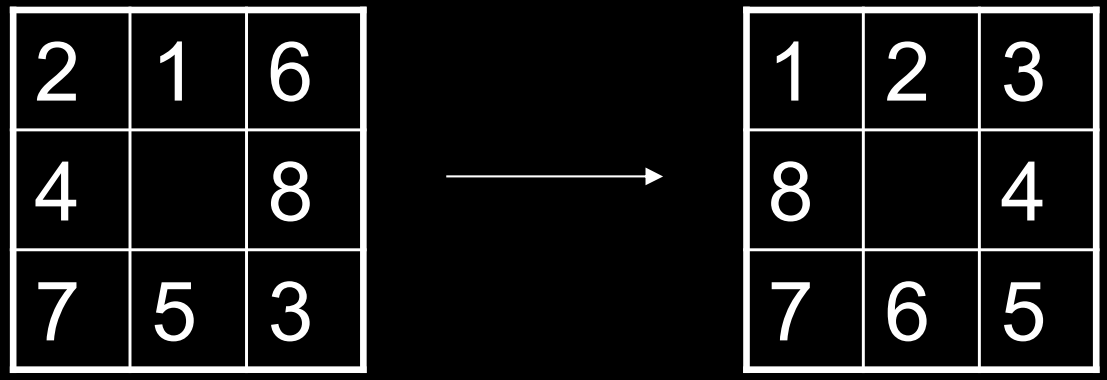
- Using h'(n)≡0 reflects a total absence of knowledge about the application domain, so while the algorithm is admissible, it is impractical.
- An algorithm A is more informed than an algorithm B if $h_A > h_B$ for all states except the objective ones.
- Example: 8-puzzle with h=W where W is the number of wrong pieces. This algorithm is more informed than the uniform cost one (h'=0).

CONSISTENCY

- It can be shown that if the heuristics are consistent A* never expands more nodes than an A algorithm with less or equal heuristic information.
  - Consistent $h'(m) - h'(n) \leq h(m) - h(n)$, i.e.: the estimated path cost between two nodes is a lower (or equal) threshold of the actual cost.
    - This characteristic is usually checked as long as the heuristic does not change throughout the search process. Example: h=W is consistent.

APPLICATION EXAMPLE

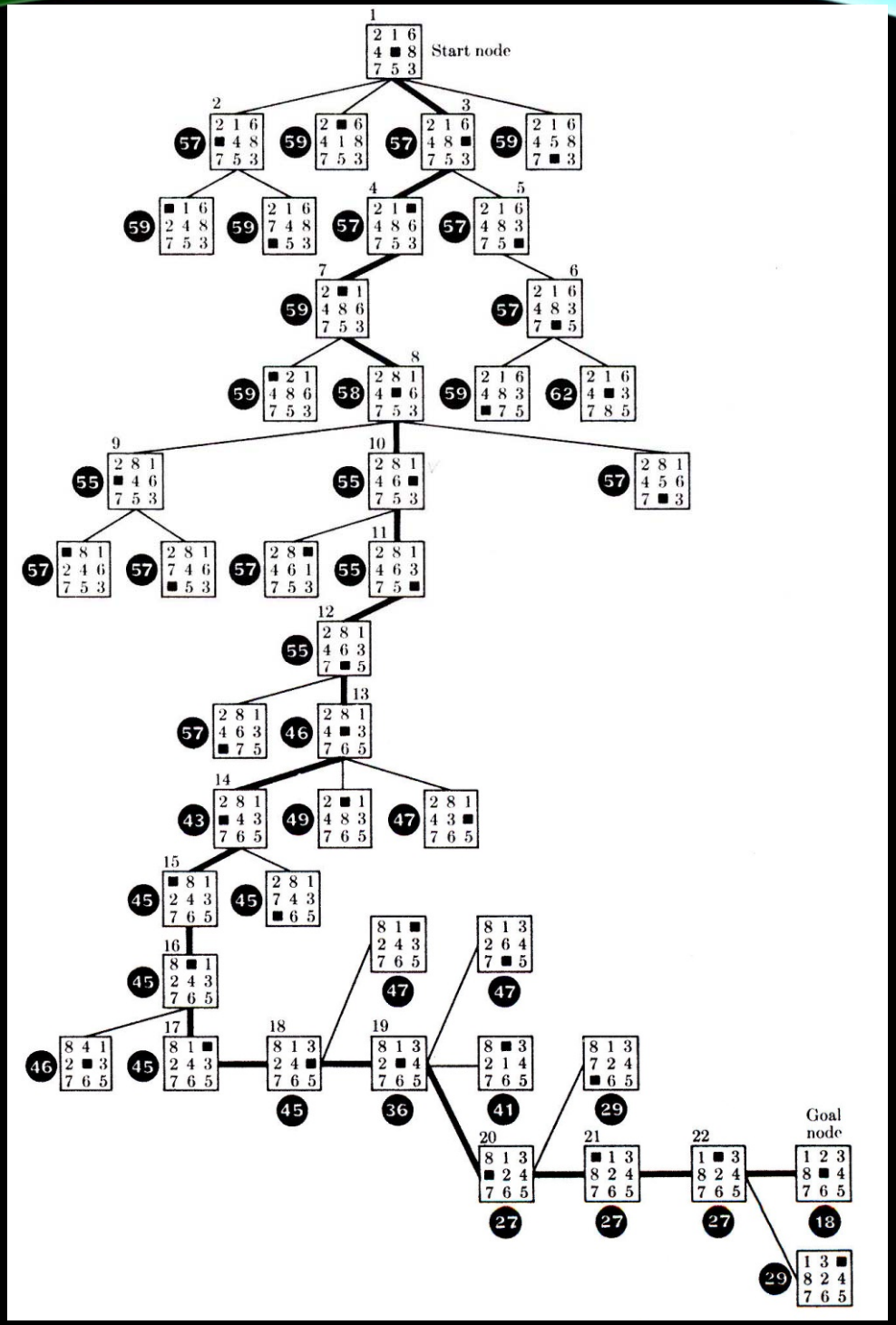
- Consider the following 8-puzzle problem:



- Consider the heuristic function **h'=P+3S** where
  - P is the sum of the distances from each piece to its right home assuming no obstacles,
  - S is the sum of a score for each piece in, when circling around the central square, 2 is added if the piece is not followed by its correct successor and 0 otherwise; 1 is added for a wrong piece on the central square.

CHECK THE FOLLOWING GRAPH

- The values in black balls represent the evaluation function f' = g + h'
- The numbers outside the balls represent the order of expansion of the nodes in the graph
- Are the heuristics used admissible?
- Is it guaranteed to find the least cost(optimal) solution?



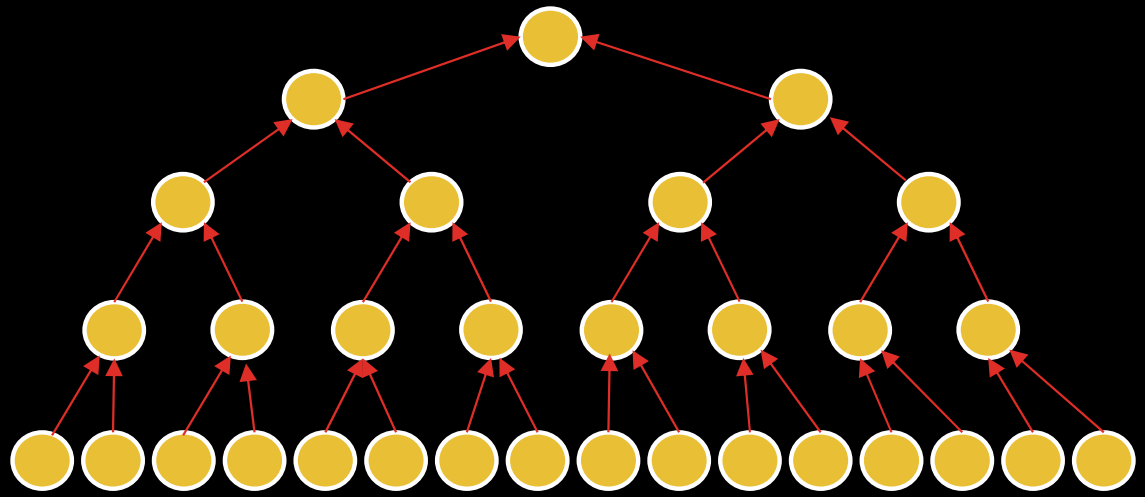
(Nils Nilsson, Problem Solving Methods in AI, p.67)

COMPARISON WITH BF

- In the previous graph, A* generated a total of 43 nodes and the solution has 18 levels.
- How many nodes would the breadth-first algorithm generate to find the same solution?
  - An average branching factor B=2 can be assumed.
  - With this value of B the number of nodes on each level is B^n .
  - The expected value (mean) for the total number of nodes generated by the BF is given by T_m =(MIN+MAX)/2 where MIN is the minimum number of nodes generated and MAX is the maximum number of nodes generated.
  - MIN = $2^1 + 2^2 +...+ 2^{17} +2$
  - MAX = $2^1 + 2^2 +...+ 2^{18}$
  - $T_m = [2*(2^1 + 2^2 +...+ 2^{17})+(2+2^{18})]/2$
- The BF would present a graph of over 150,000 nodes, instead of the 43 presented in the previous graph by A* with the P+3S heuristic.

COMPARISON OF BF AND DF

- It may also be of interest to compare the two previously studied uninformed algorithms (breadth-first and depth-first). To facilitate the calculations it is considered that the solution exists at level L=4 and that you have a branching factor B=2.



1 ... 2^1 = 2

2 ... 2^2 = 4

3 ... 2^3 = 8

4 ... 2^4 = 16

BF VS. DF (WITH D=L)

- Applying, in the case of BF, the method described in the previous slide, the following values are obtained:
  - MIN = 2 + 4 + 8 + 2 = 16
  - MAX= 2 + 4 + 8 + 16 = 30
  - T_m = (16 + 30)/2 = 23
- Similarly, for DF, if d=4, we have:
  - MIN = 2 + 2 + 2 + 2 + 2 = 8
  - MAX= 2 + 4 + 8 + 16 = 30
  - T_m = (8 + 30)/2 = 19
- Apparently DF is better than BF.

DF (WITH D>L)

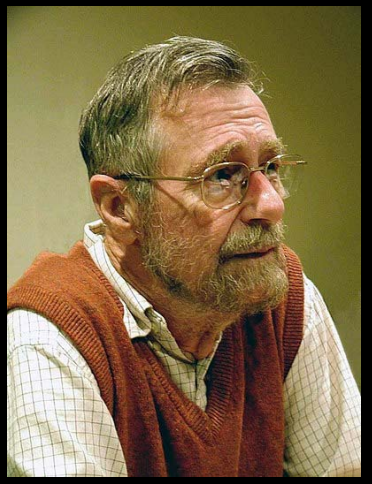
- If now the maximum depth level, d, is greater than L, the BF-DF relationship changes.
- Consider d=5:
  - MIN = 2 + 2 + 2 + 2 + 2 = 8
  - MAX= 2 + 4 + 8 + 16 + 32 - 2 = 60
  - T_m = (8 + 60)/2 = 34
- Consider d=6:
  - MIN = 2 + 2 + 2 + 2 + 2 = 8
  - MAX= 2 + 4 + 8 + 16 + 32 + 64 - 2 - 4 = 120
  - T_m = (8 + 120)/2 = 64
- In both situations DF is worse than BF.

DIJKSTRA'S ALGORITHM

This algorithm solves the shortest path problem.

PSEUDOCHOOD:

1. Assign a value of zero to the minimum cost estimate for node **s** (the root of the tree) and infinity to the estimates for all other nodes in the graph;
2. At each step, find the node **u, which** has not yet been processed, that has the smallest distance to **s**.
3. See for each node **v**, neighbor of **u**, whether it is better to keep the current distance from **v** or to update by making the path S→u and then u→v. Note that the path S→u has already been fixed and possibly has connections in between.



Edsger Dijkstra
1956

COMPARISON WITH OTHER STATE SPACE SEARCH ALGORITHMS

- If the arcs have unit value, Dijkstra's algorithm is equal to BF.
- If the arcs of the graph have arbitrary positive weights this algorithm determines the minimum cost path. In this case, Dijkstra's algorithm is equal to the uniform cost algorithm.
- Dijkstra's algorithm is a special case of A* where the heuristic is zero.

PERFORMANCE MEASURES

- There are certain measures that while not completely determining heuristic power can be useful for comparing various search techniques.
  - Penetrance: $P = \frac{L}{T}$

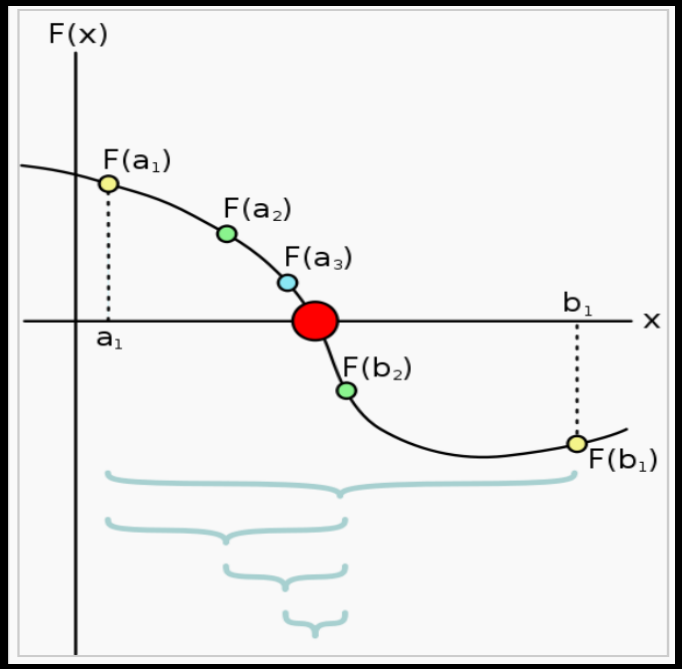
where L is the length of the path to the goal and T is the total number of nodes generated

  - Average branching factor

$$B + B^2 + \ldots + B^L = T \quad \text{or} \quad \frac{B}{B-1}(B^L - 1) = T$$

SOLVING HIGHER ORDER EQUATIONS

- Bisection method



- Newton-Raphson method

http://faculty.washington.edu/dbp/SAPACLISP-1.x/basic-math.lisp

STATE SPACE SEARCH ALGORITHMS

Limited Memory Search

IDA*
ITERATIVE DEEPENING A*

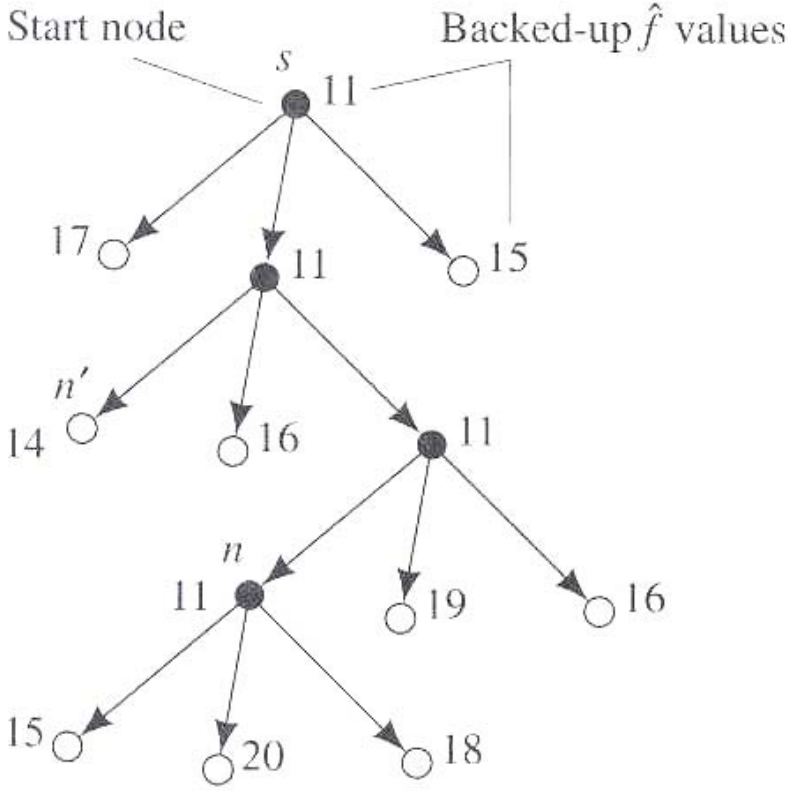
- The memory requirements of the uninformed methods increase exponentially with the depth of the objective state in the search space.
- Using heuristics does not avoid this problem, although it does reduce the branching factor.
- IDA* appears in 1985 (Korf)
- Can be implemented in parallel (Powley, Ferguson and Korf 1993)
- IDA* guarantees the discovery of the optimal solution, provided an admissible heuristic is used.

IDA* ALGORITHM

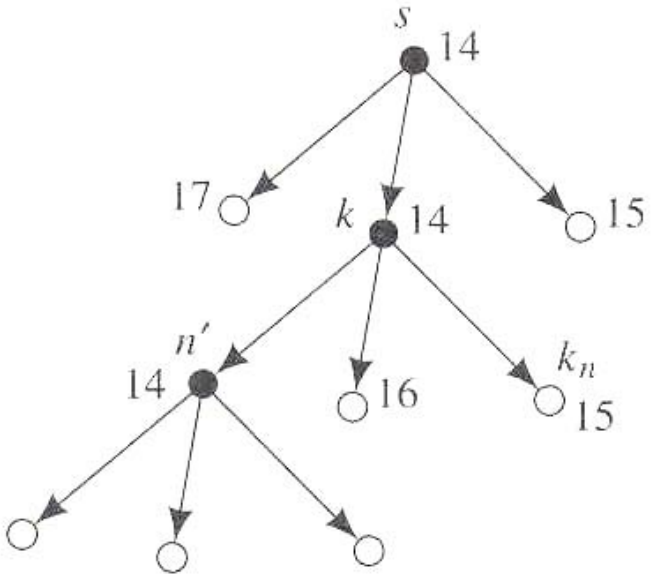
- The depth search method is applied a number of times, with varying depth thresholds, but where the limit ("cost cut-off") is given in terms of f.
- In the first search the threshold L is given by f'(n0) = g(n0) + h'(n0) = h'(n0), where n0 is the initial node.
- The cost of the optimal path can be equal to the threshold but not higher since the heuristic has to be admissible, i.e. h(n0) >= h'(n0)
- Only nodes with f'(n)<=L are expanded
- If the solution is not found, a new threshold L1 is used such that L1=min(F(n)) where F(n) is the set of nodes that have been visited but not yet expanded.

PERFORMANCE

- Since IDA* is a search-in-depth type it only needs to store in memory a number of nodes equal to the largest branch explored.
- In a problem where the values of f' are different for all nodes in a state space the number of iterations can be equal to the number of nodes with f' less than the cost of the optimal path.
- In this case, the solution being given by a sequence of N nodes the A* exploits O(N) while the IDA* requires 1+2+...+N i.e. $O(N^2)$.



(a) RBFS has just expanded node n but has not yet backed up the $\hat{f}$ values of its successors



(b) $\hat{f}$ values have been backed up, the subtree below k_n has been discarded, and search continues below n'

Figure 9.9

Recursive Best-First Search

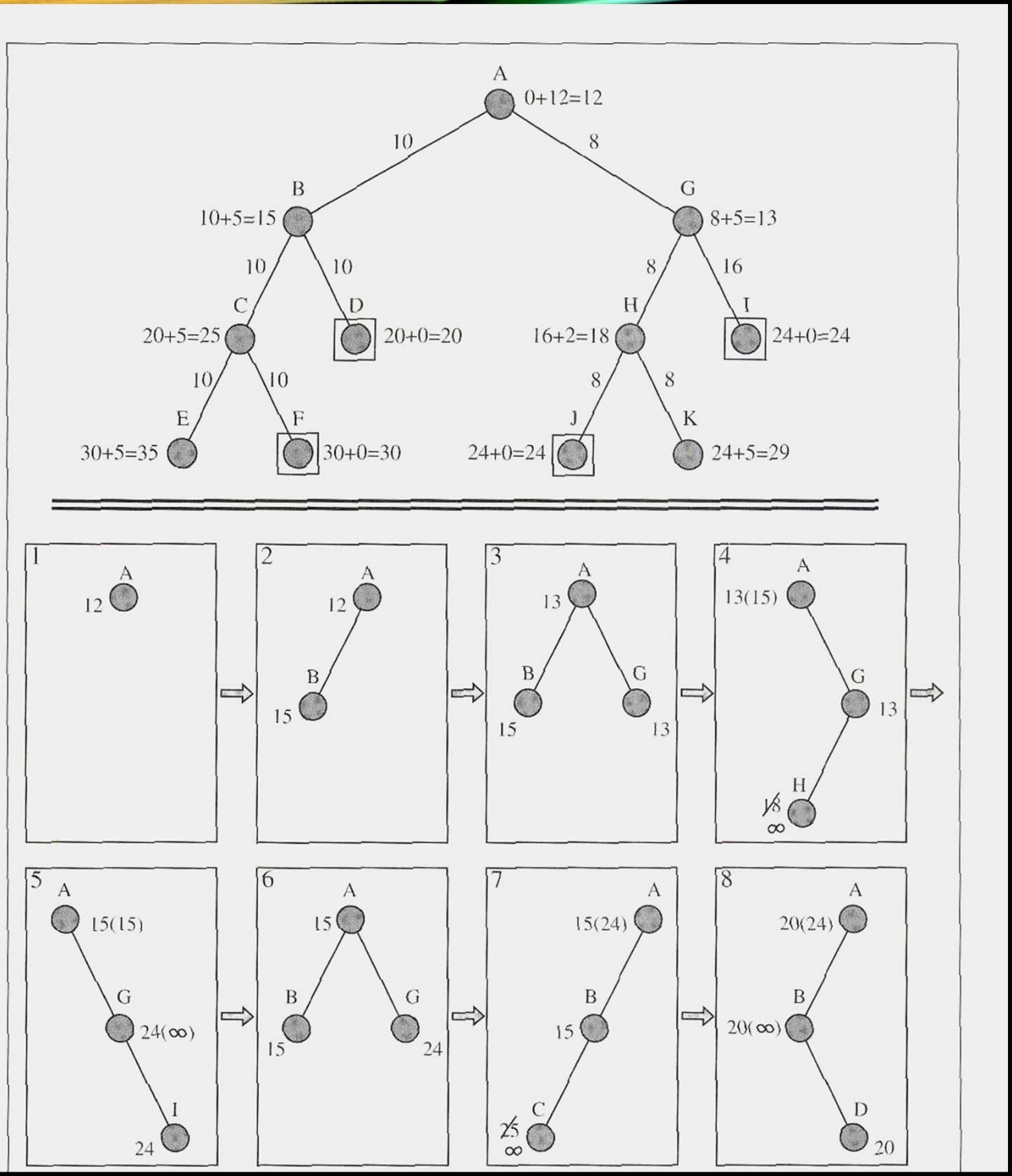
RBFS - RECURSIVE BEST-FIRST SEARCH

- It is a variant of IDA* where
  - the f' values of the successors of a node n are calculated and
  - accordingly, the f' values of node n and all its predecessors are recalculated (backup of the optimal f' value)
- The recalculated value of node n with successors n_i is

  $f'(n) = \min[n_i]\ f'(n_i)$
- If one of the successors of node n, n_i , has the lowest f' value of OPEN then that is the next expanded node.
- If another OPEN node, m, has the lowest f' value, then the algorithm eliminates all nodes up to the common predecessor except their direct successors, continuing to search from node m.

SMA* SIMPLIFIED MEMORY-BOUNDED A*

- The SMA* can use all available memory to perform the search, which increases efficiency.
  - Avoids repeating states, as far as memory allows
  - It is complete if the memory is sufficient to store the shortest path to the solution.
  - It is optimal if memory is sufficient to store the least cost (optimal) path to the solution.
  - If you have enough memory to store the entire state tree the search is equivalent to A*.


A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29
1
2
3
4
5
6
7
8
12
15
13
13(15)
15(15)
24(∞)
24
15(24)
20(24)
20(∞)
20
25
∞
18
∞

ALGORITHM

- The basic idea of SMA* is that when it is necessary to generate a successor but there is no memory available, it is necessary to make room by "forgetting" one of the previously generated nodes.
- Rules:
  - The SMA* prefers to forget the node with higher f'. In case of a tie it removes the one with the lowest level.
  - Before removing a subtree, it stores in the predecessor node of that subtree information about the quality of the best path in the forgotten subtree.
  - When a successor is generated, its value is propagated upwards (back-up) by means of the minimum rule, similarly to RBFS.
  - A node whose depth N equals the available memory limit (in terms of number of nodes) is immediately valued with f=∞

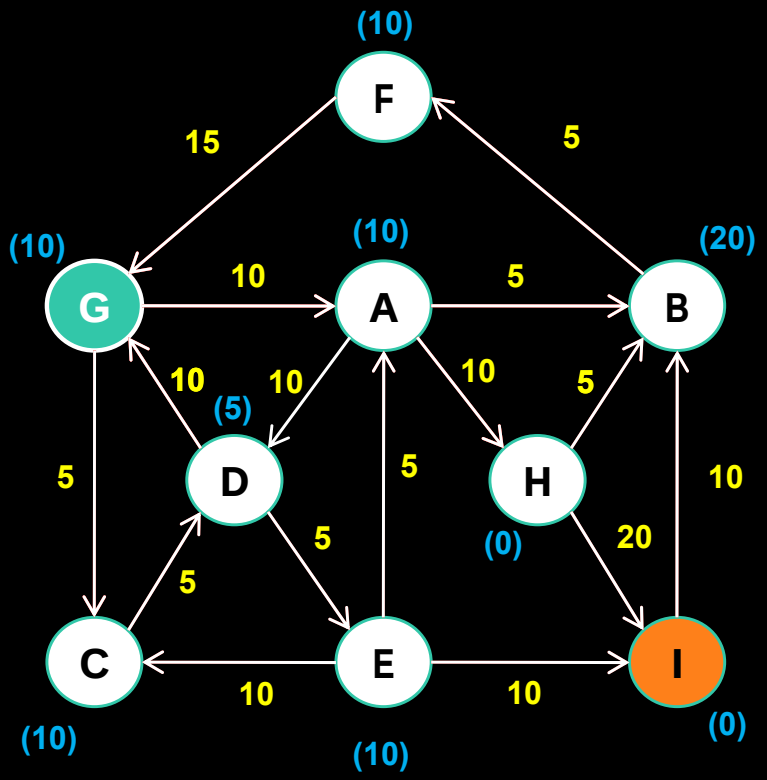
FINAL SMA ASSESSMENT*

- SMA* may be better than A* on problems with strongly connected state spaces.
- SMA* may be unable to solve problems that A* can solve if it is necessary to repeatedly generate the same subtrees when oscillating between candidate paths.

ALGORITHM COMPARISON

- Effectiveness:
  - Coming up with a solution
    - Optimal solution.
- Efficiency:
  - Use resources more economically
    - Minimum space
    - Minimum time

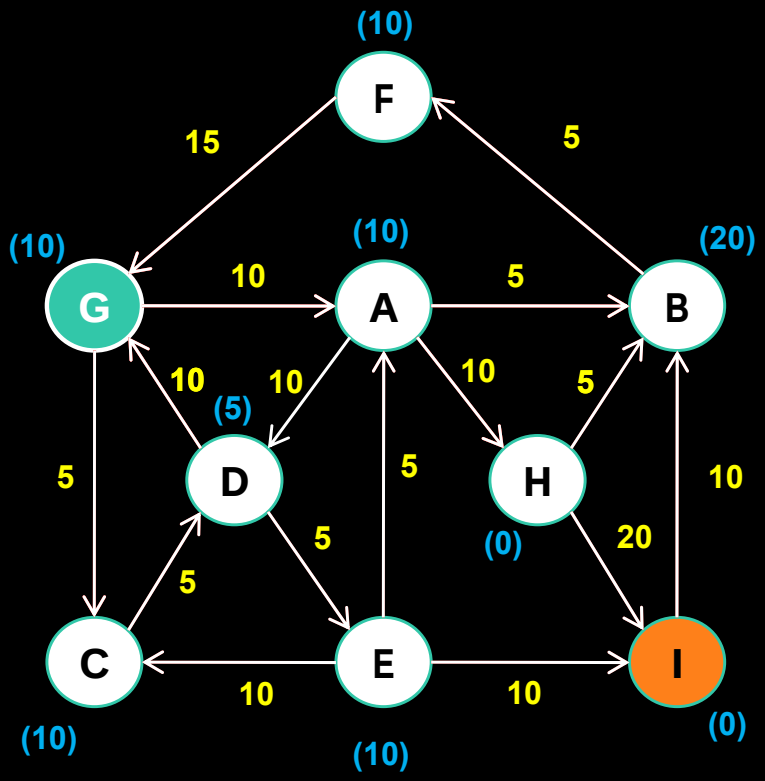
COMPARATIVE ANALYSIS SIMULATION



- A*
- IDA* - Iterative Deepening A*
- RBFS - Recursive Best First Search
- SMA* - Simplified Memory-Bound A*

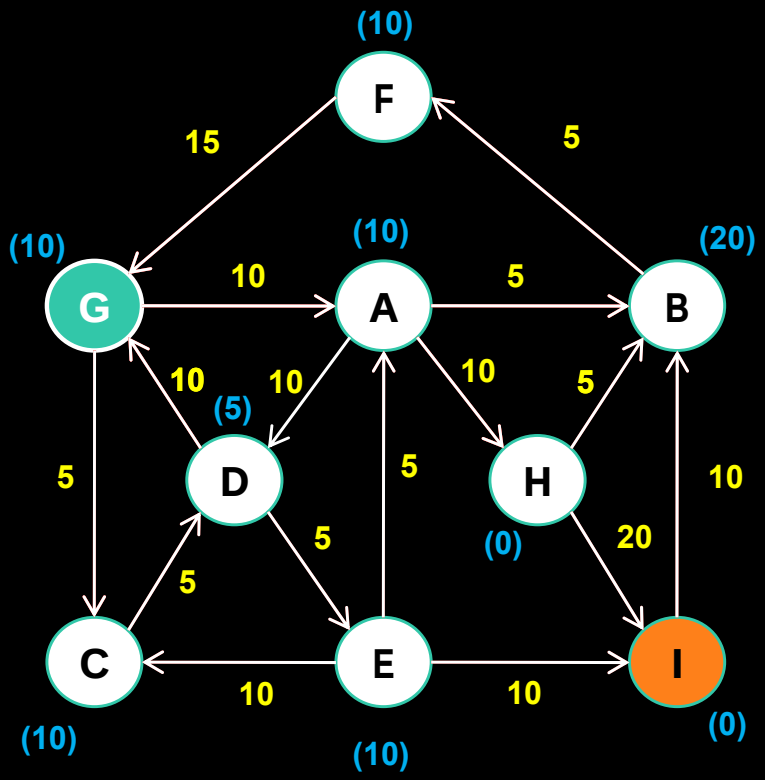
This and following slides regarding simulation are based on the example provided by Prof. Cédric Grueau

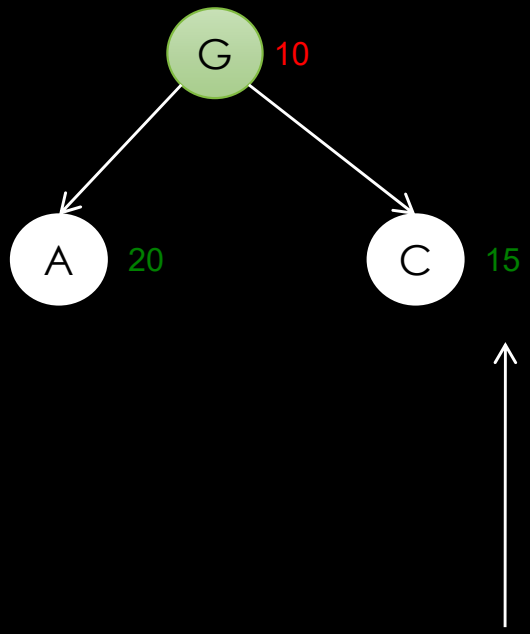
ALGORITHM A*



Note: $h(n)$ values are identified in blue in parentheses

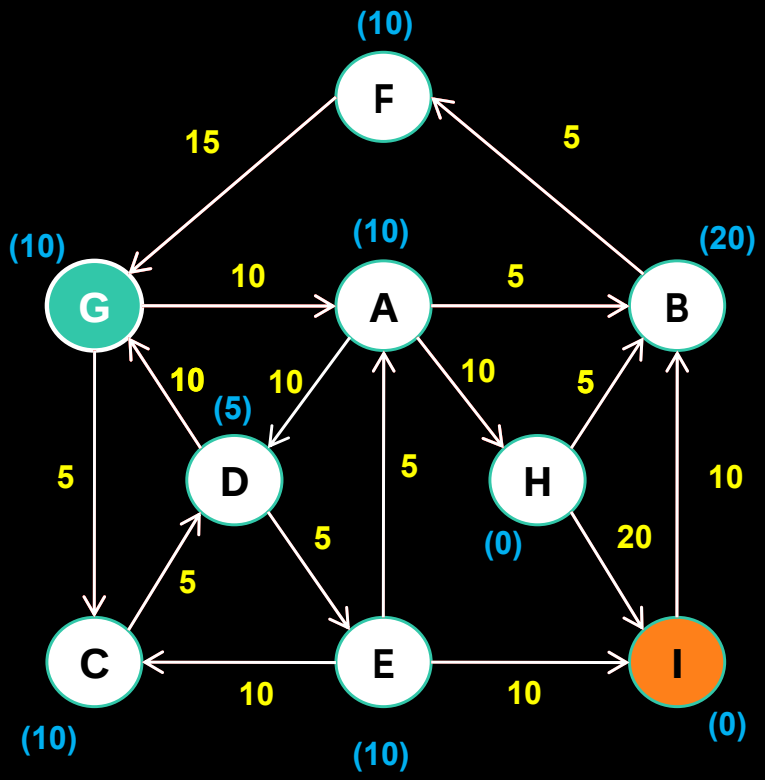
ALGORITHM A*

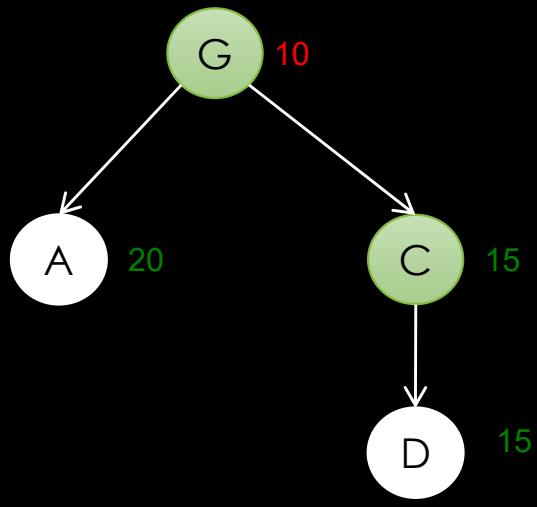




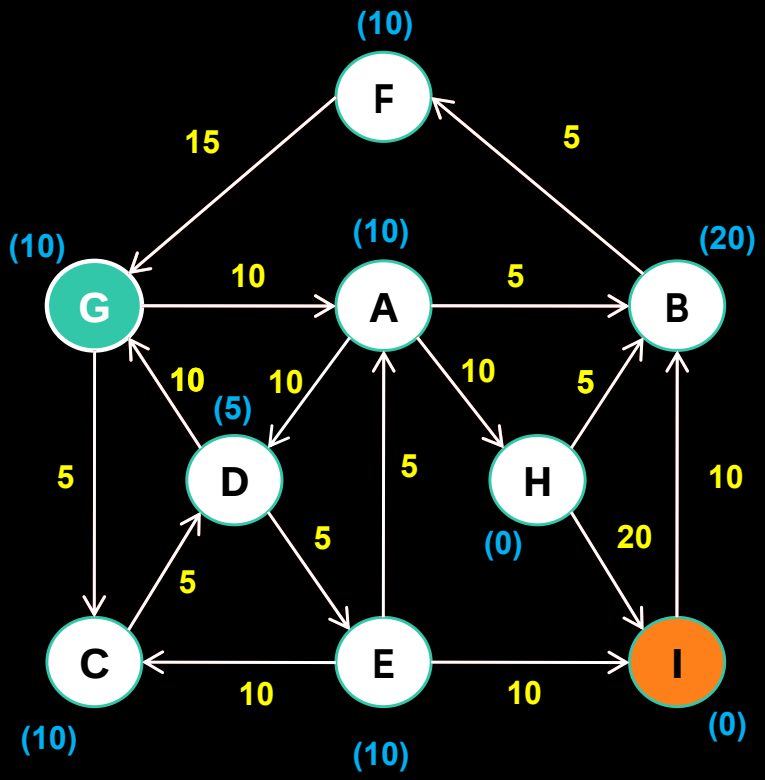
Note: *f(n)* values are identified in green

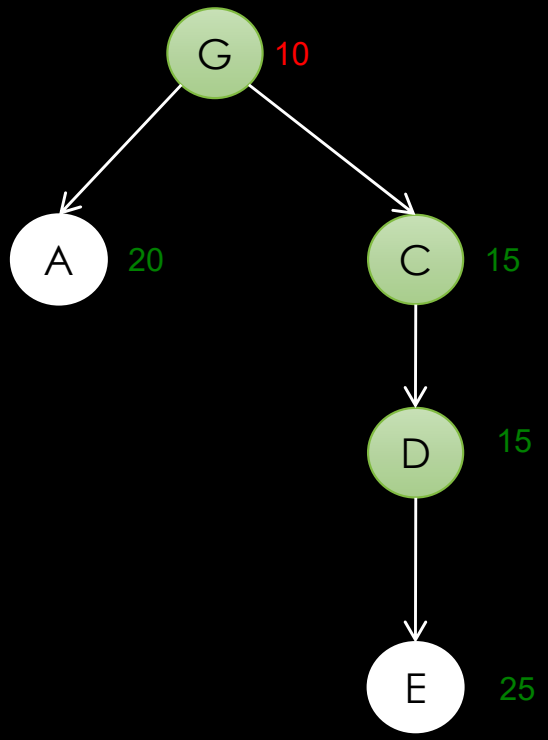
ALGORITHM A*


(10)
F
15
5
(10)
(10)
(20)
G
10
A
5
B
10
10
10
5
(5)
5
D
5
H
10
5
(0)
20
5
C
E
I
10
10
(10)
(10)
(0)

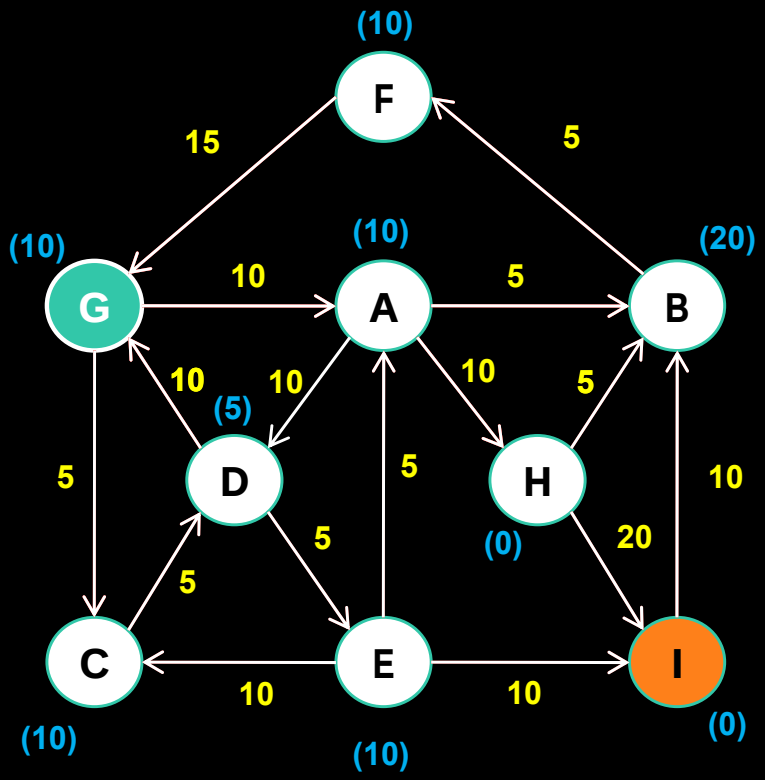

G
10
A
20
C
15
D
15

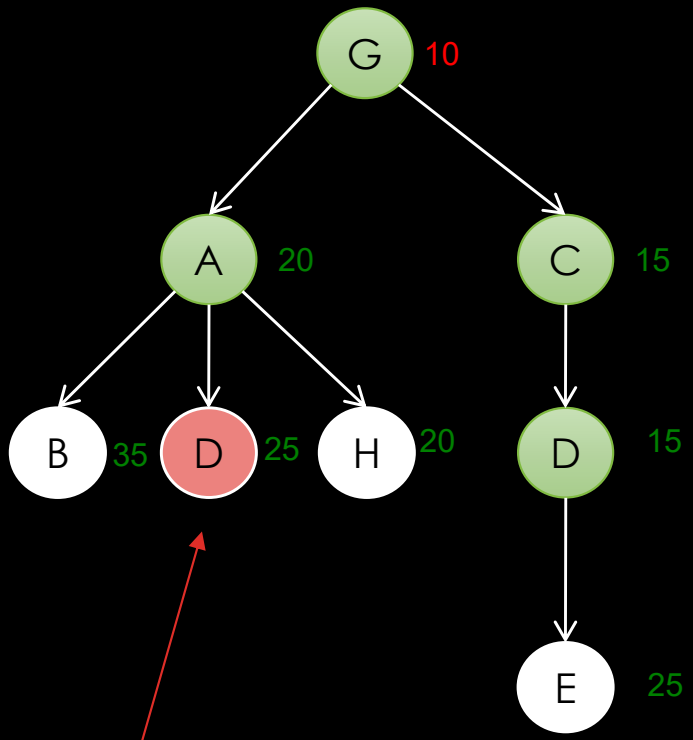
ALGORITHM A*


(10)
F
15
5
(10)
(10)
G
10
A
5
B
(20)
10
10
10
5
(5)
5
D
5
H
10
(0)
20
5
C
E
I
10
10
(10)
(10)
(0)


G
10
A
20
C
15
D
15
E
25

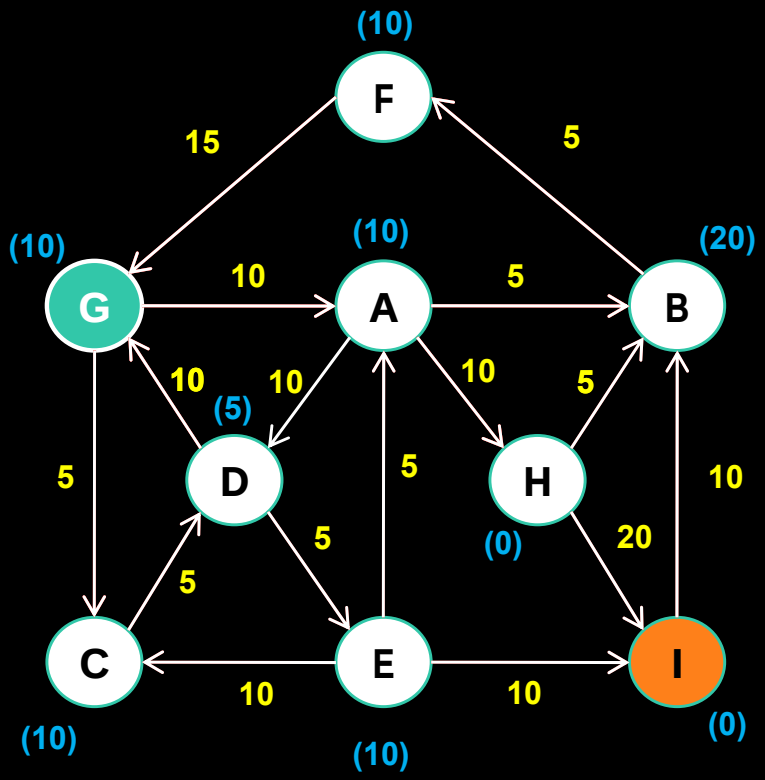
ALGORITHM A*

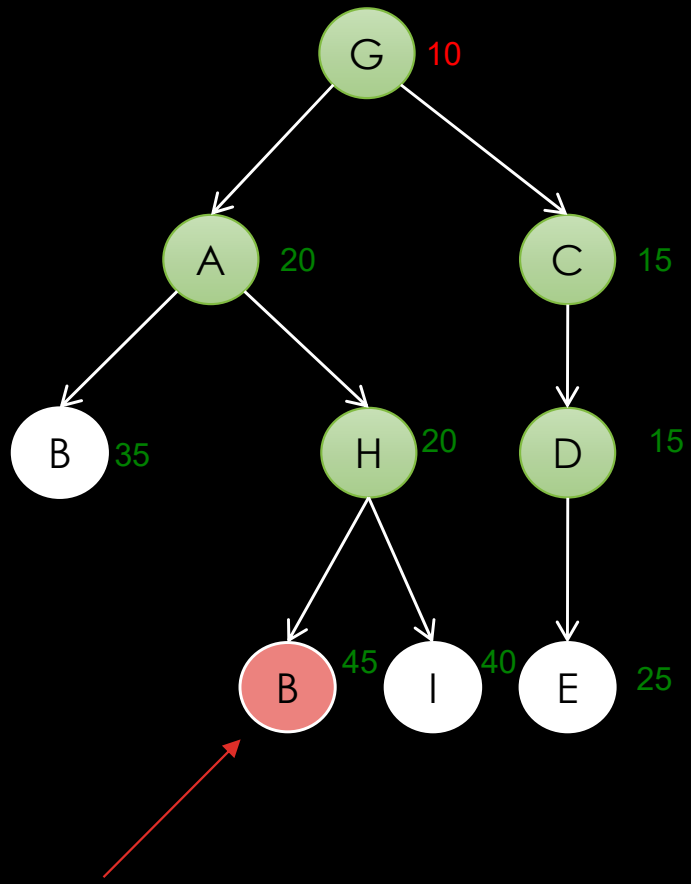




Note: You don't put node D in open because it is already in closed and is less costly

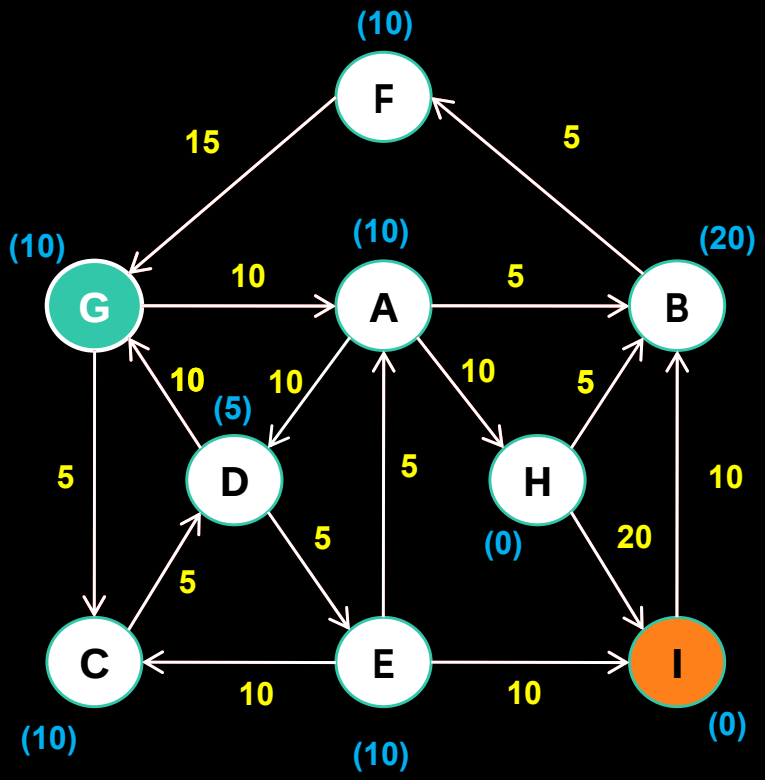
ALGORITHM A*

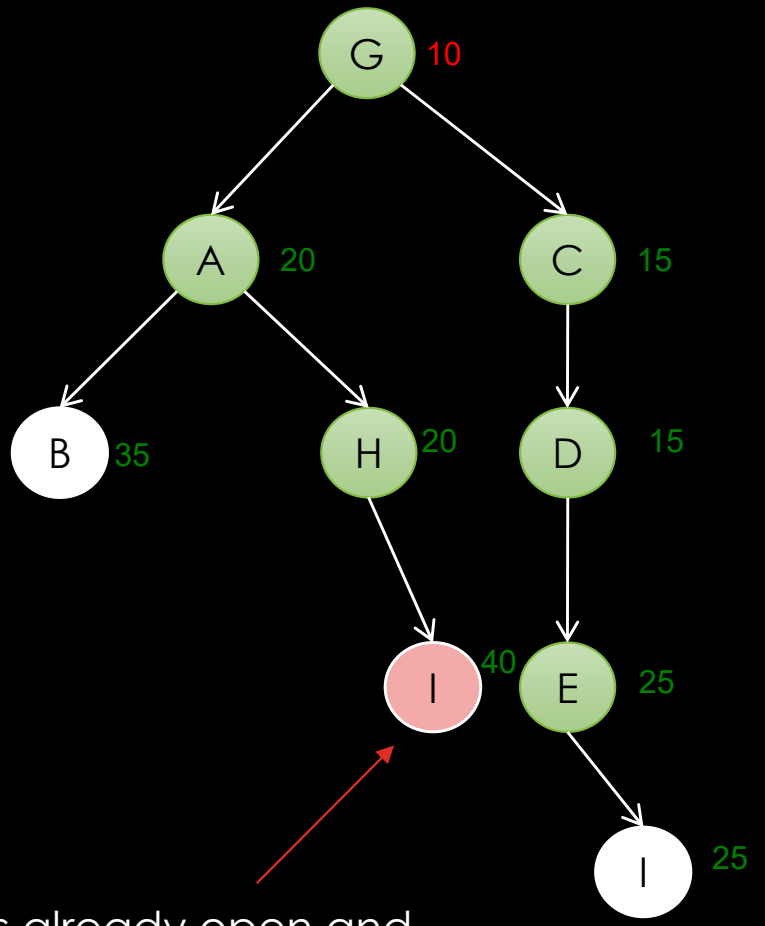




Note: You don't put node B in open because it is already in open and is less costly

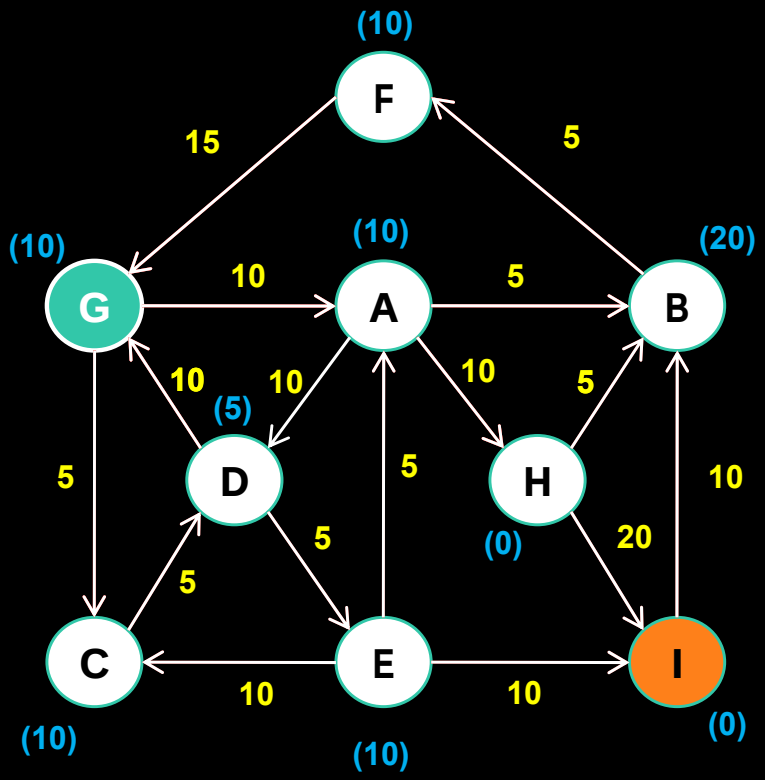
ALGORITHM A*

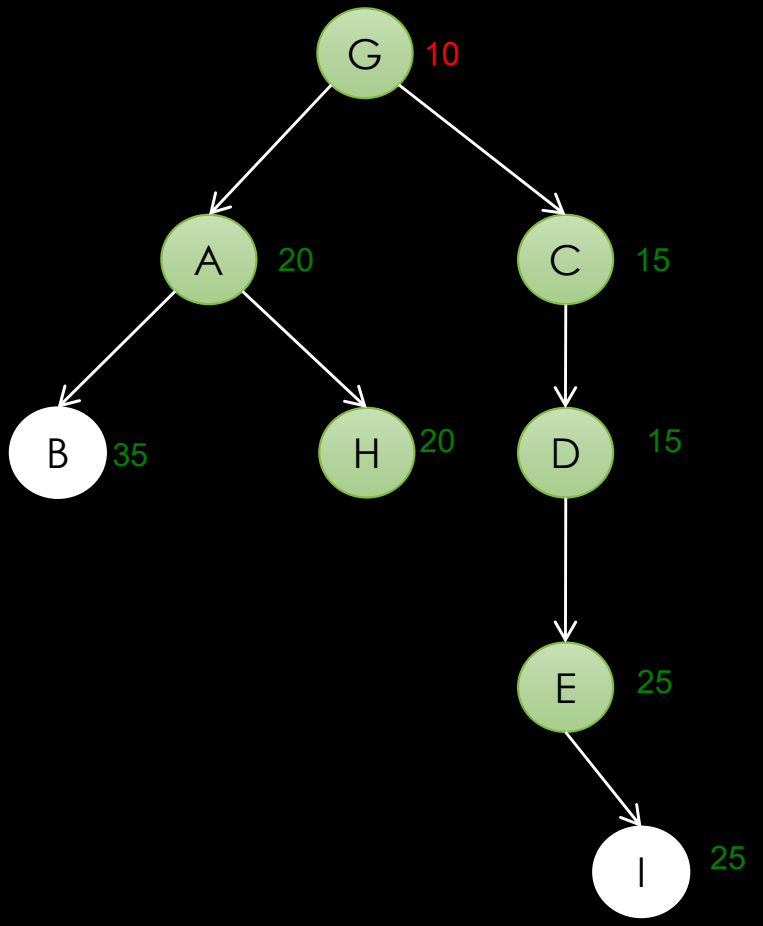




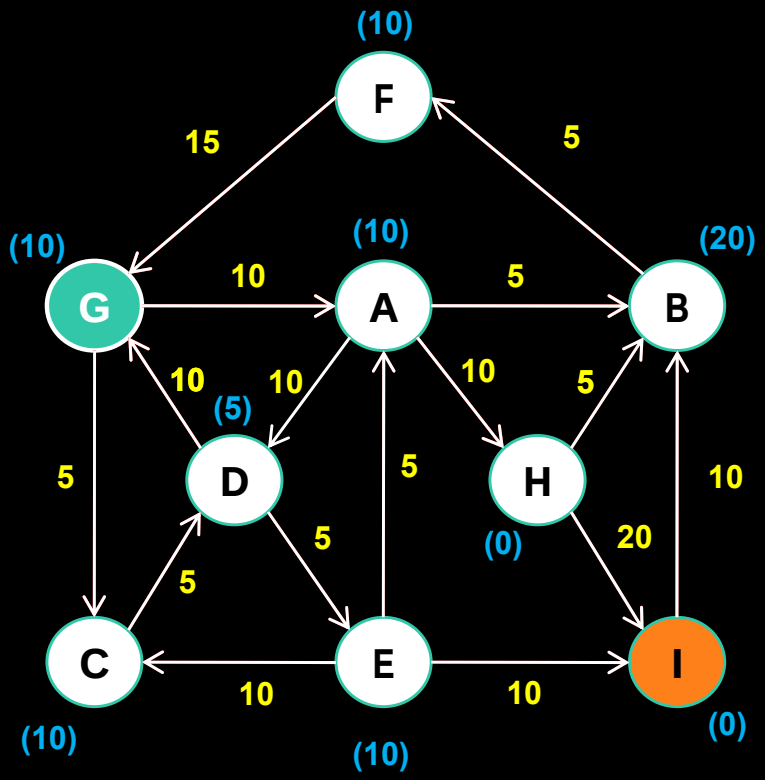
Note: Node I is already open and has higher cost. In this case the old node is removed and the one with lower cost is kept

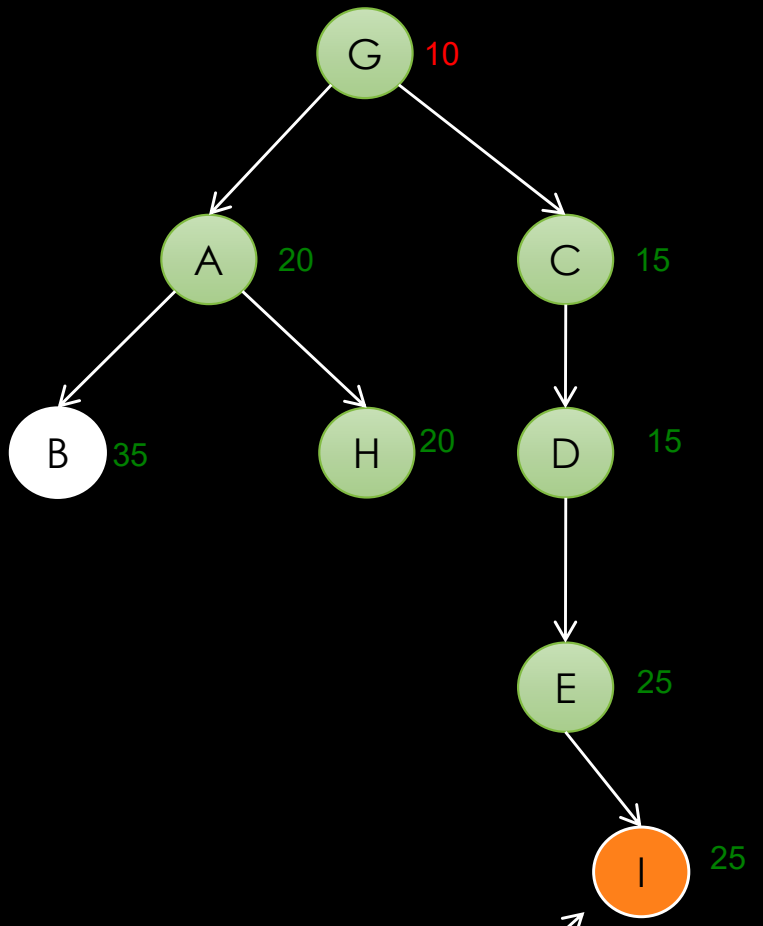
ALGORITHM A*


(10)
F
15
5
(10)
(10)
G
10
A
5
B
(20)
10
10
10
5
(5)
D
5
H
10
5
5
(0)
20
5
C
E
I
10
10
(10)
(10)
(0)


G
10
A
20
C
15
B
35
H
20
D
15
E
25
I
25

ALGORITHM A*



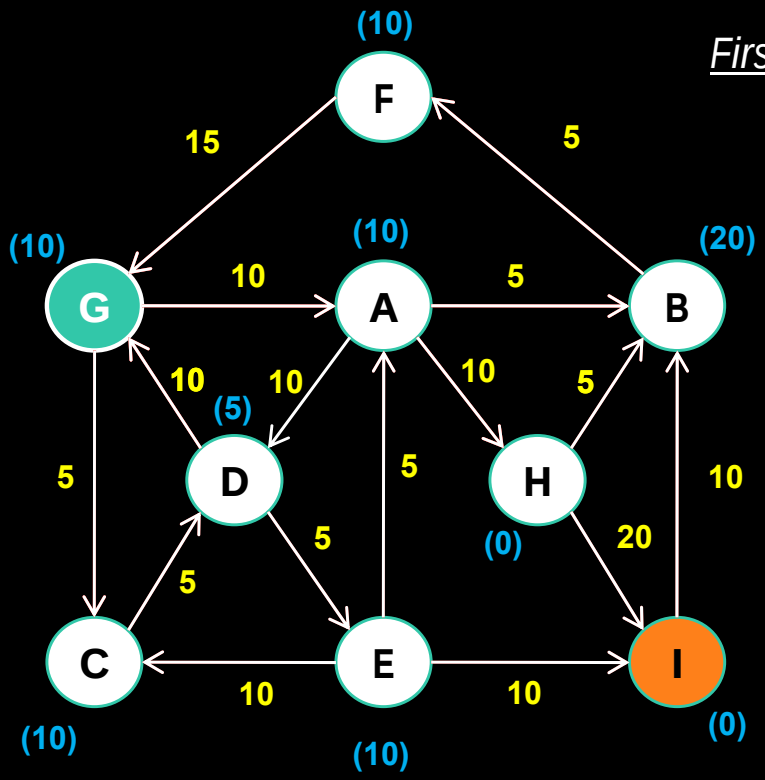


Stop and give the solution: G, C, D, E, I

COMPARATIVE ANALYSIS SIMULATION

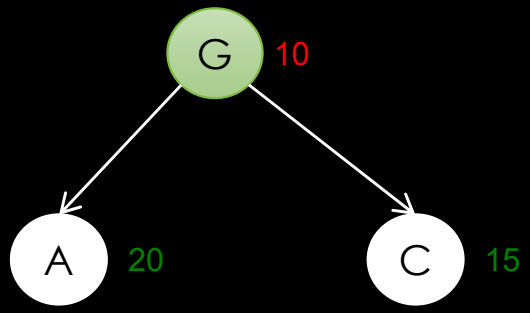
- A*
- IDA* - Iterative Deepening A*
- RBFS - Recursive Best First Search
- SMA* - Simplified Memory-Bound A*

IDA* ALGORITHM

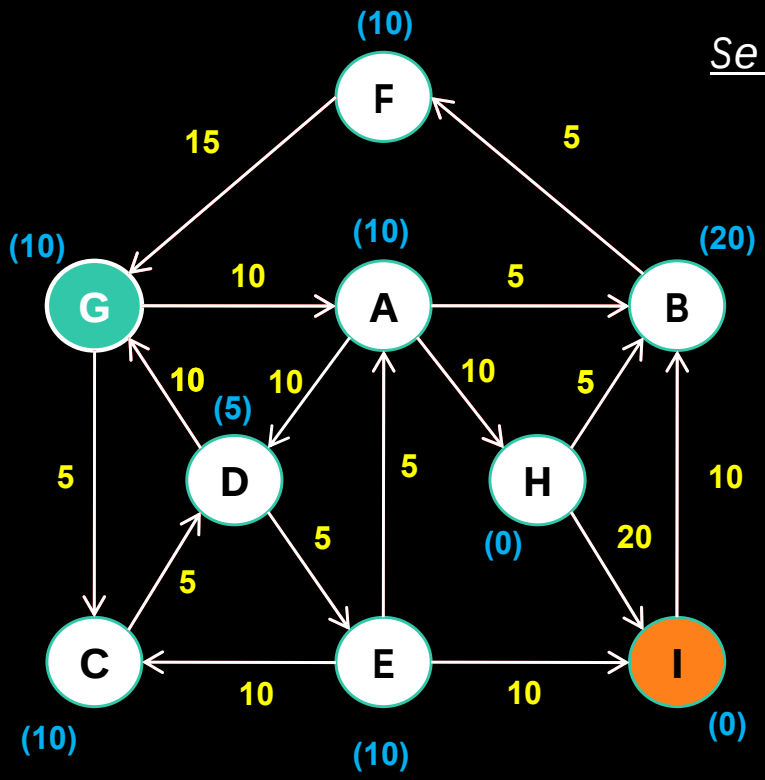


First iteration:

limit = 10

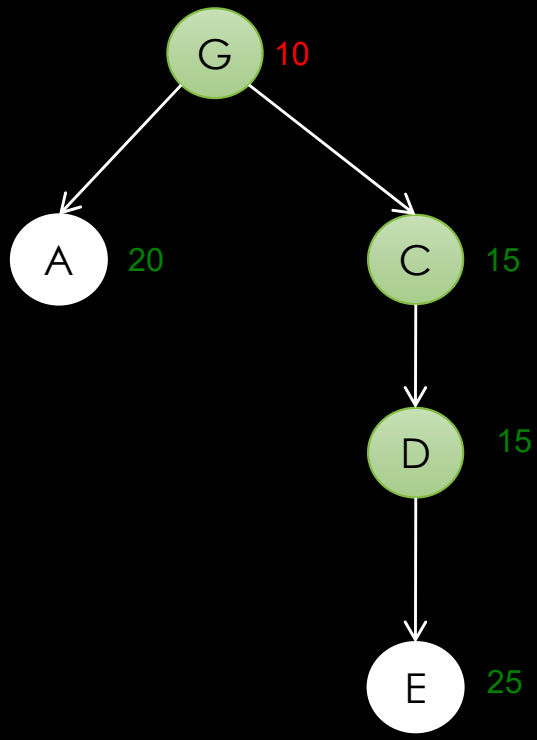


IDA* ALGORITHM

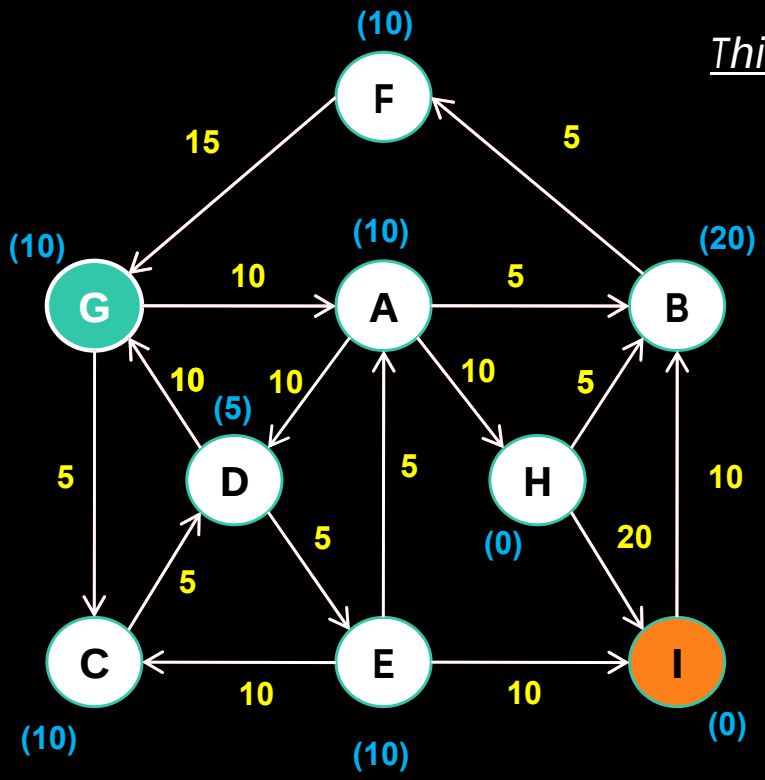


Second iteration:

limit = 15

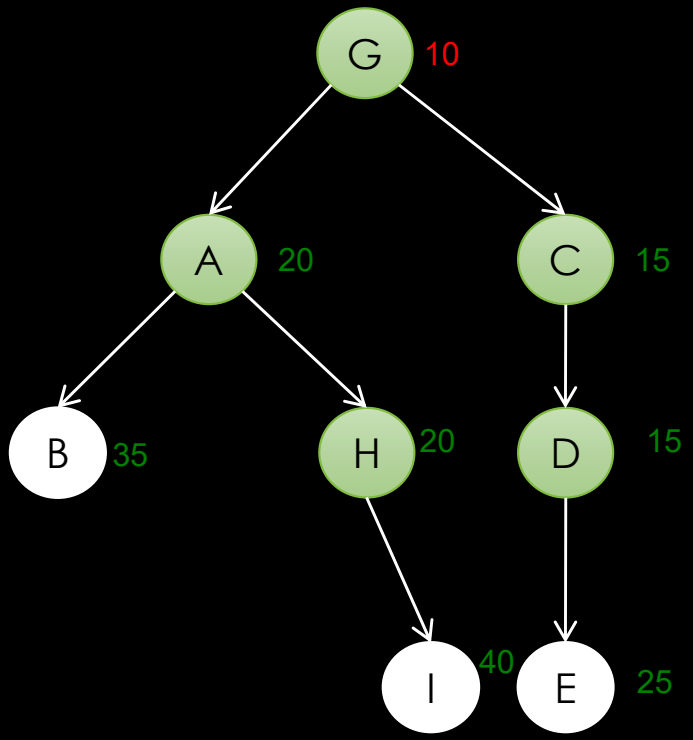


IDA* ALGORITHM

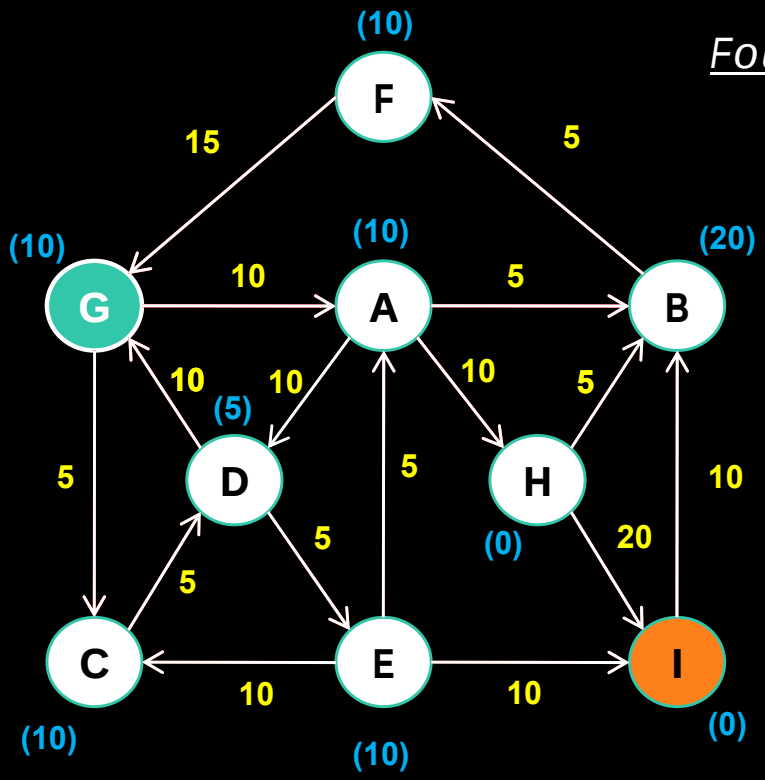


Third iteration:

limit = 20

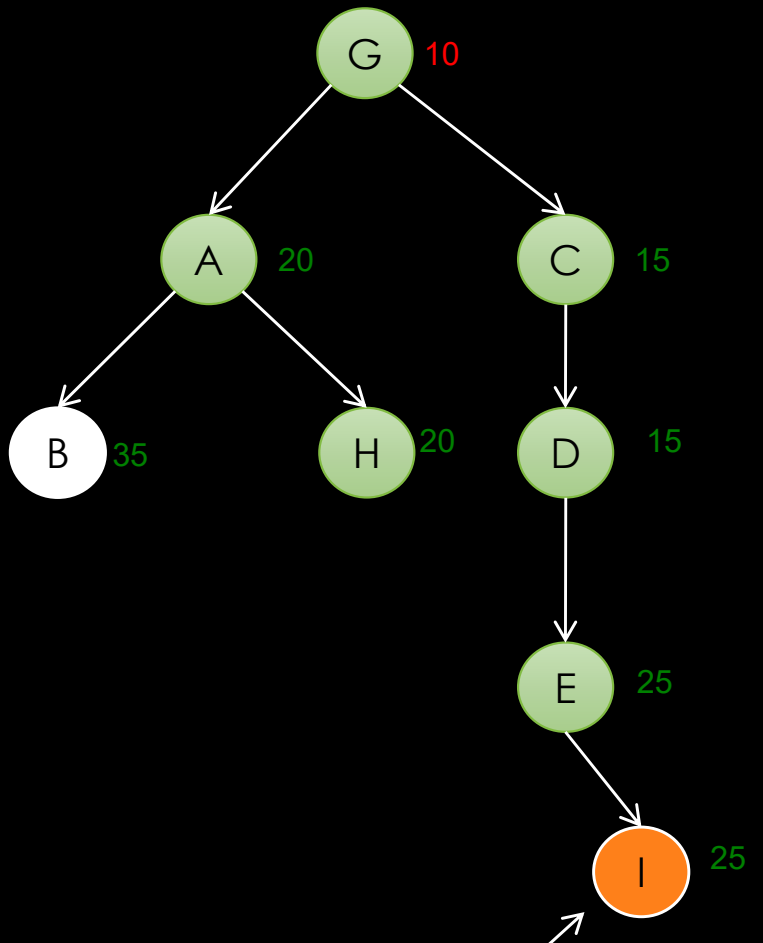


IDA* ALGORITHM



Fourth iteration:

limit = 25

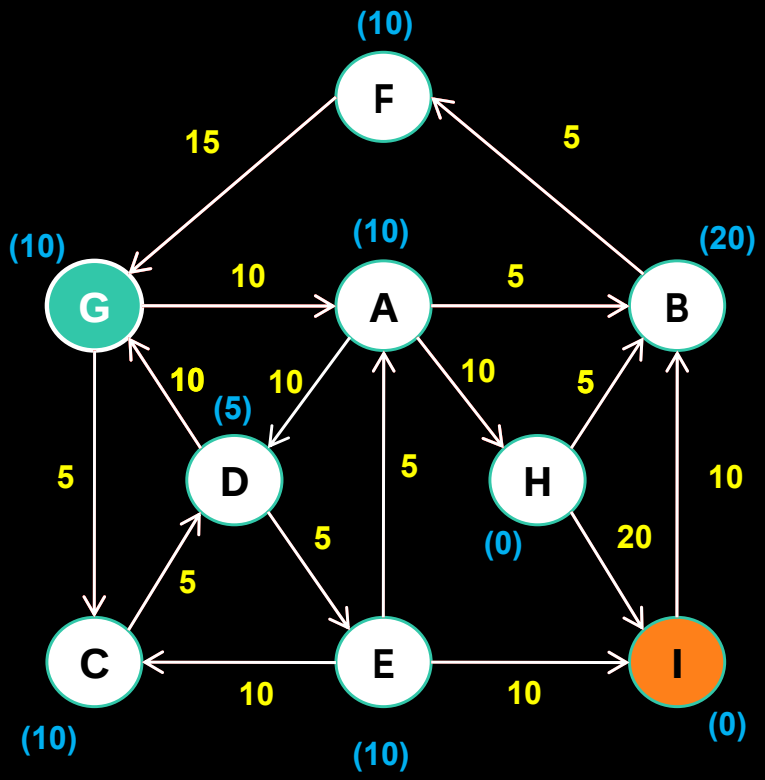


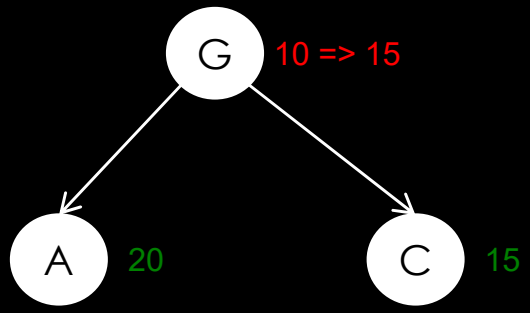
Stop and give the solution: G, C, D, E, I

COMPARATIVE ANALYSIS SIMULATION

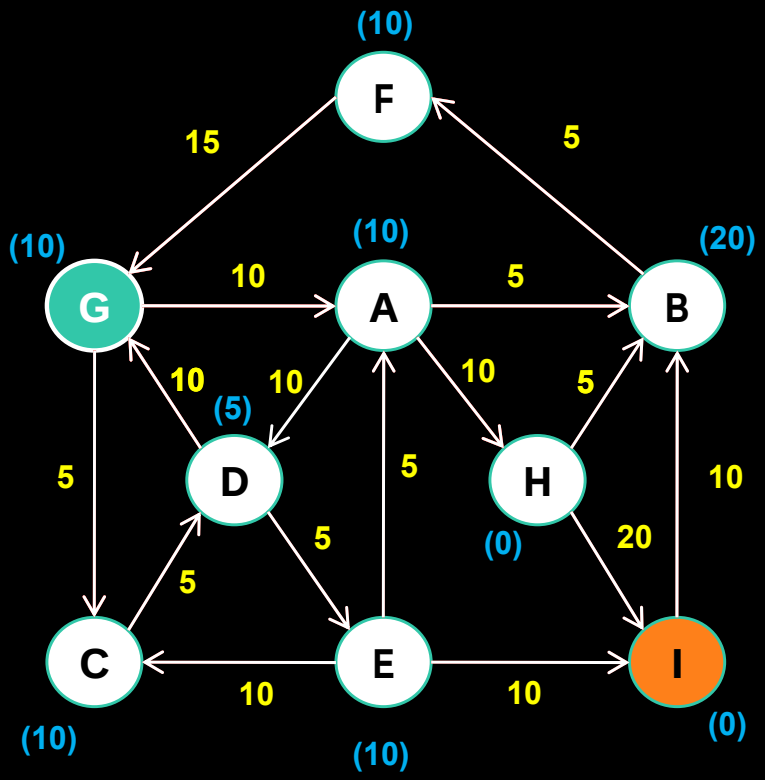
- A*
- IDA* - Iterative Deepening A*
- RBFS - Recursive Best First Search
- SMA* - Simplified Memory-Bound A*

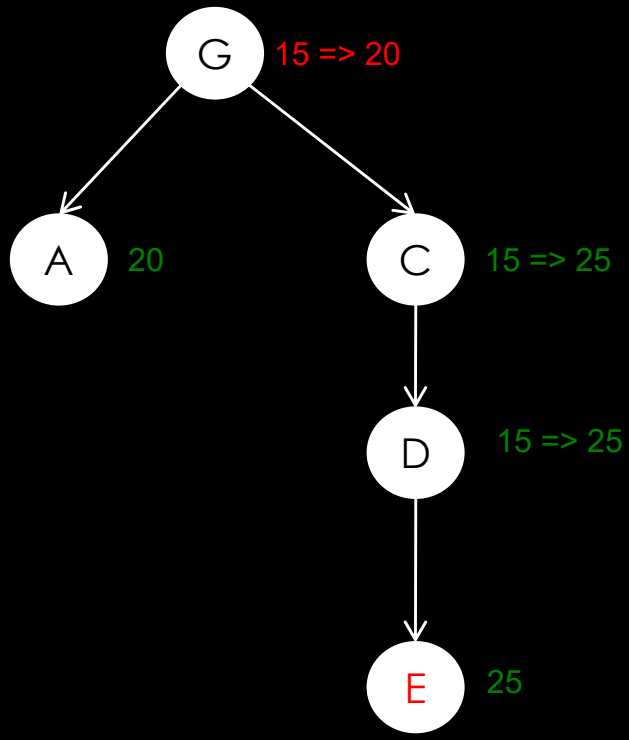
RBFS ALGORITHM



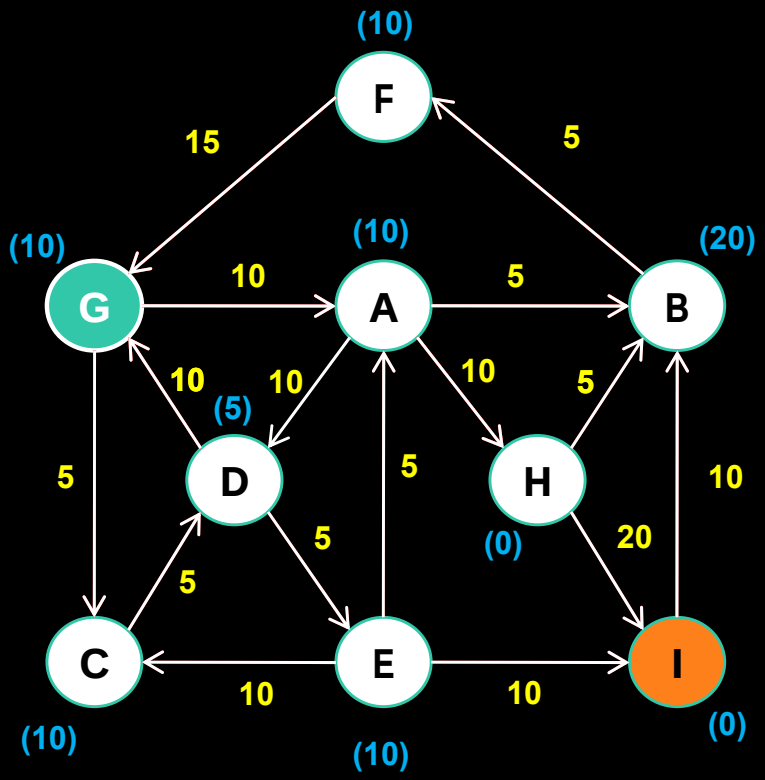


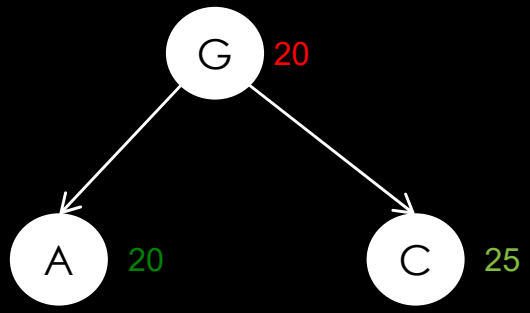
RBFS ALGORITHM



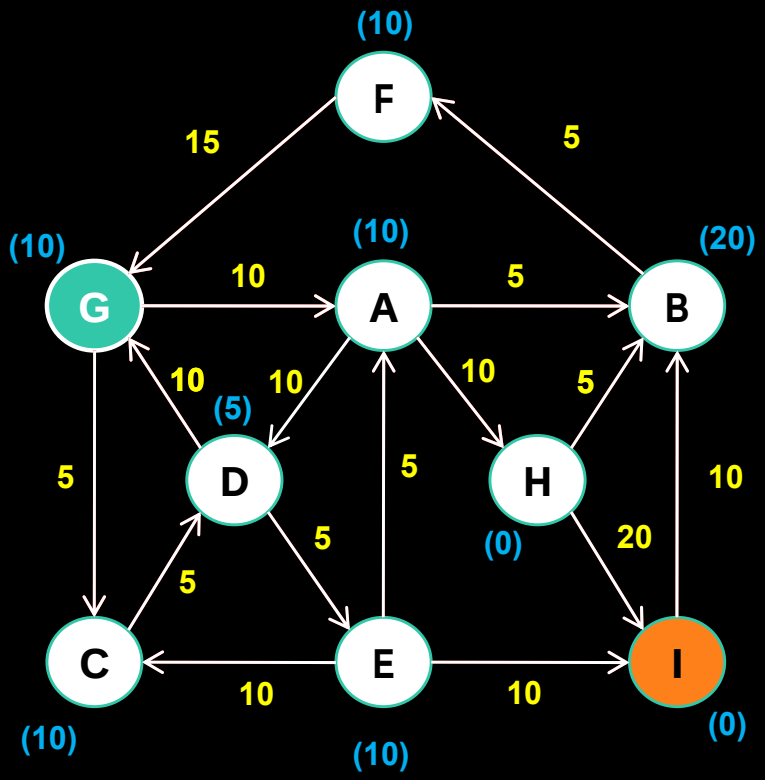


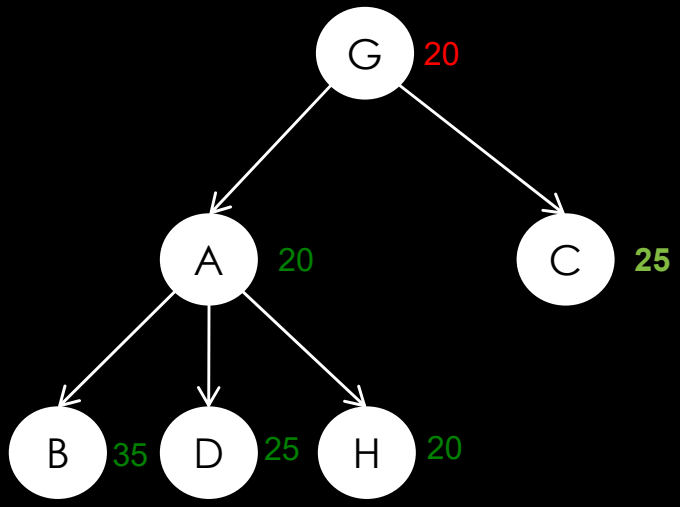
RBFS ALGORITHM



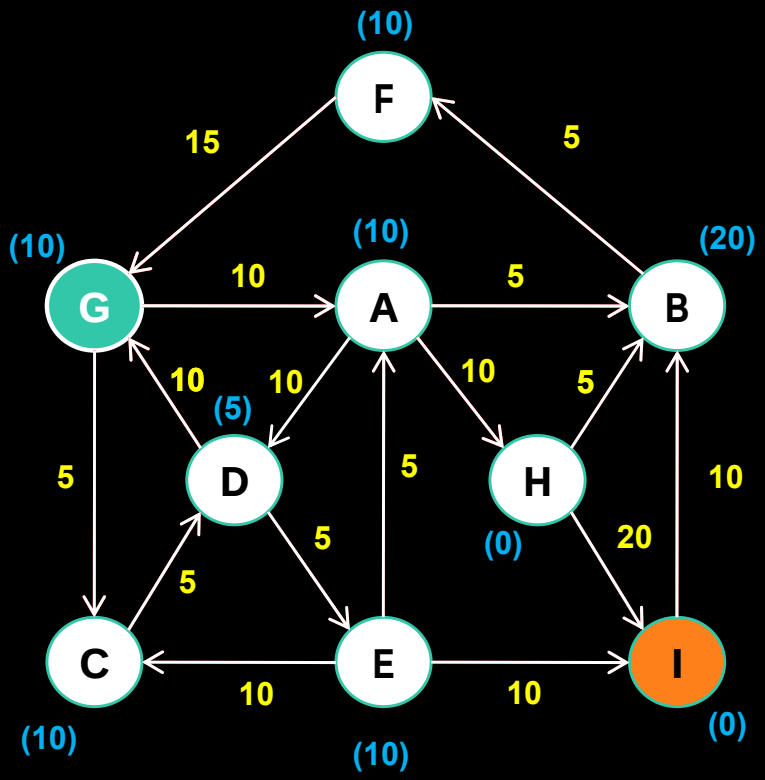


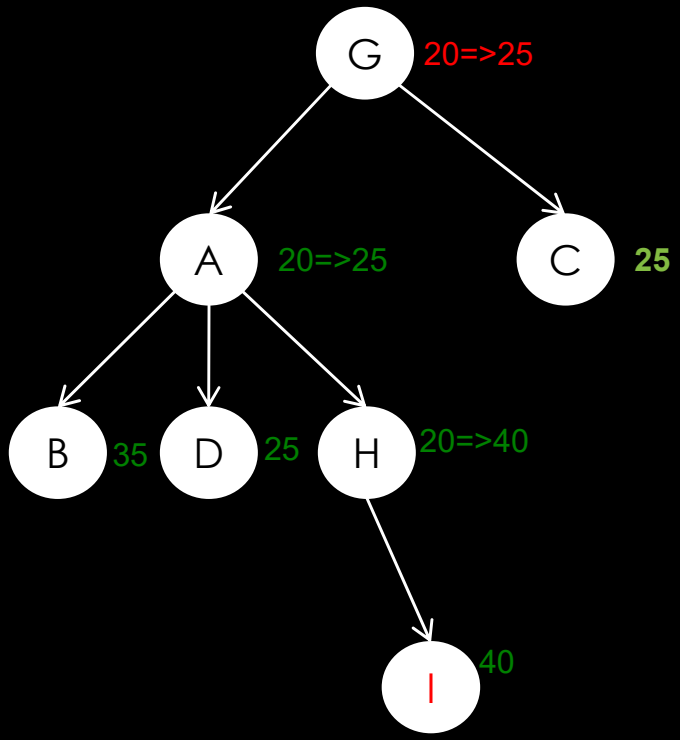
RBFS ALGORITHM



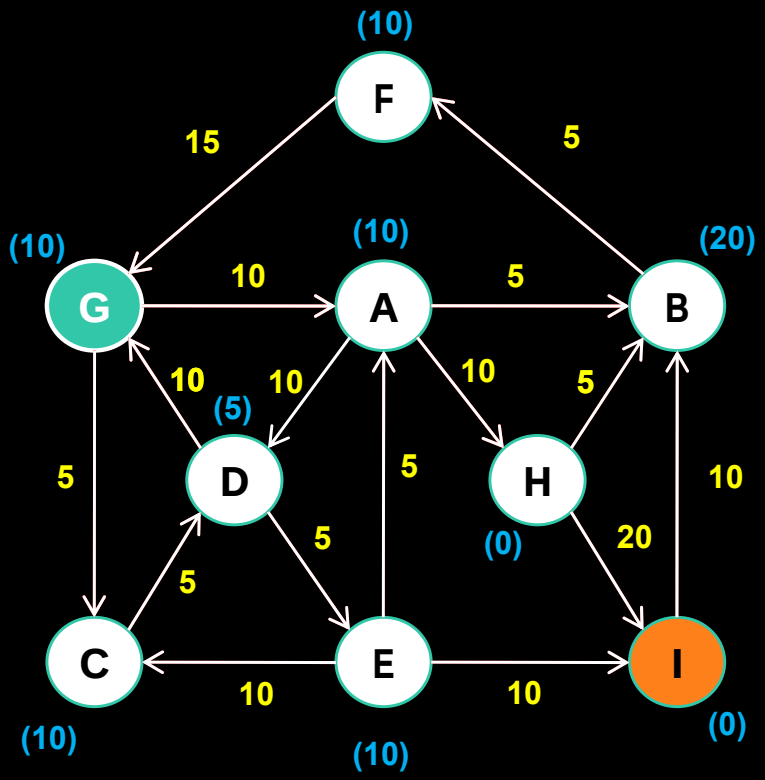


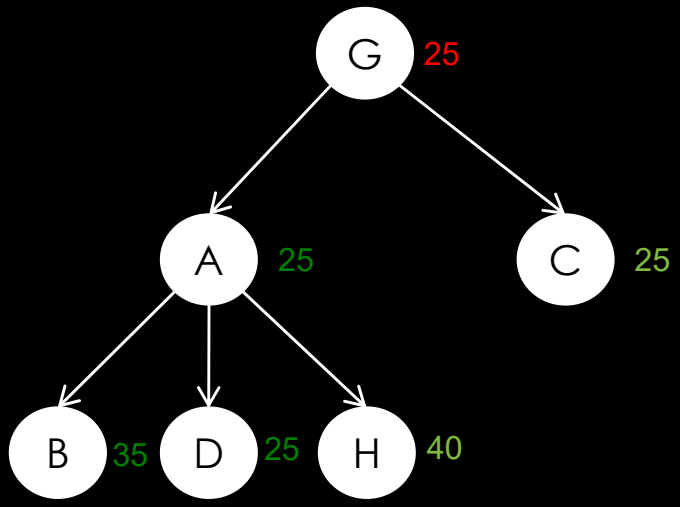
RBFS ALGORITHM



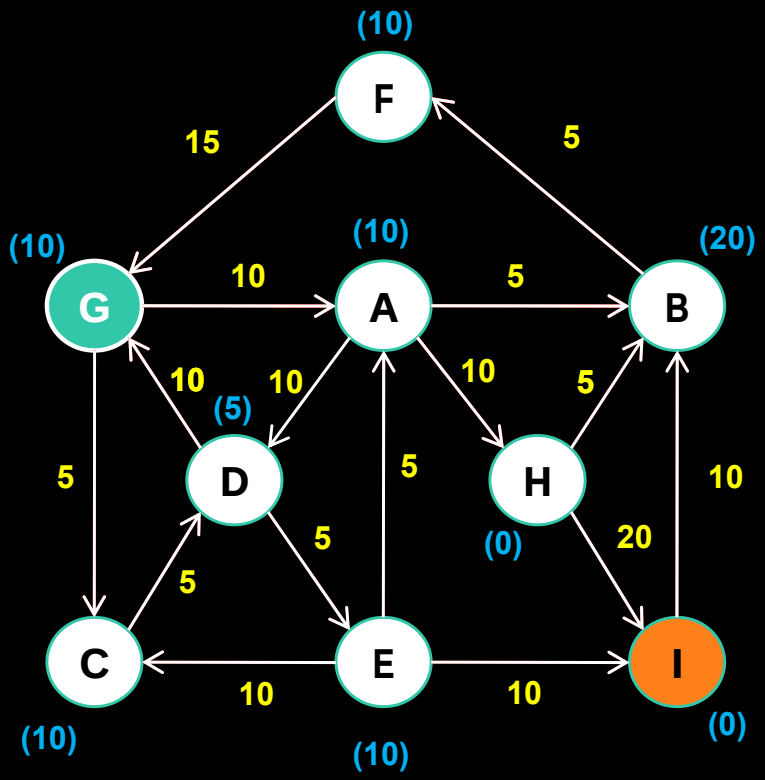


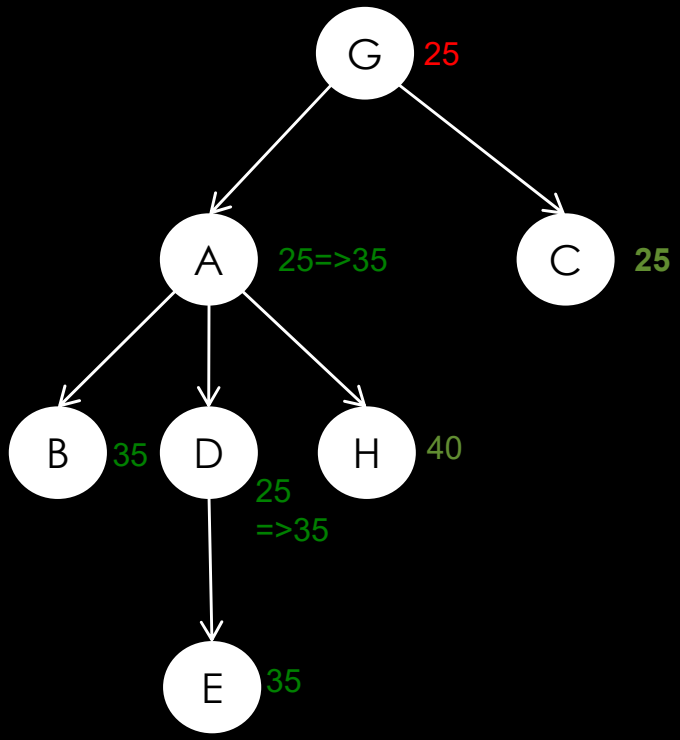
RBFS ALGORITHM



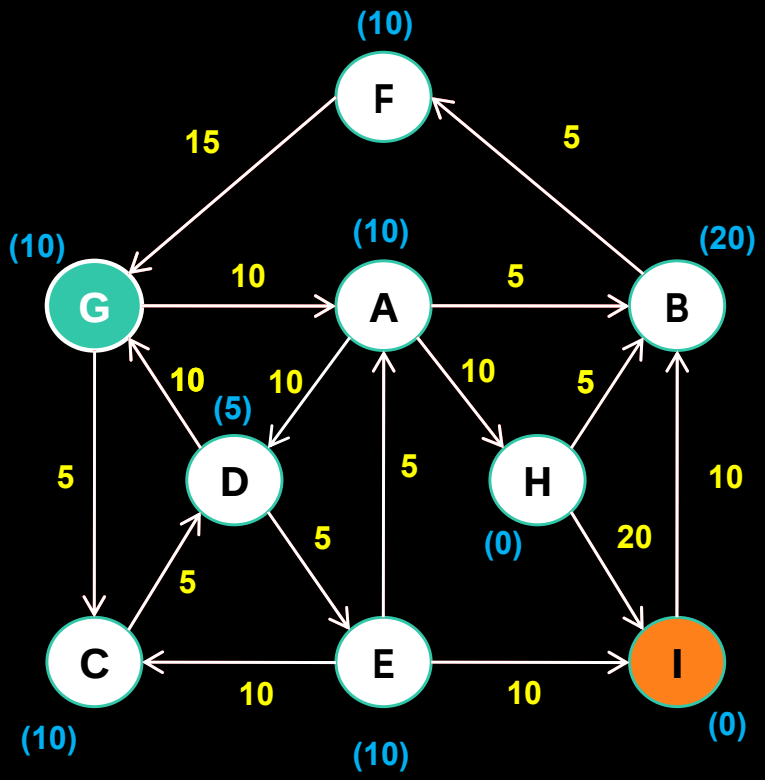


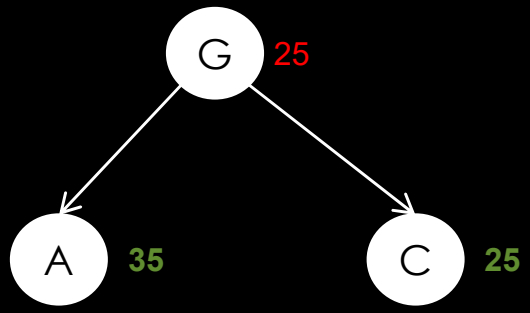
RBFS ALGORITHM



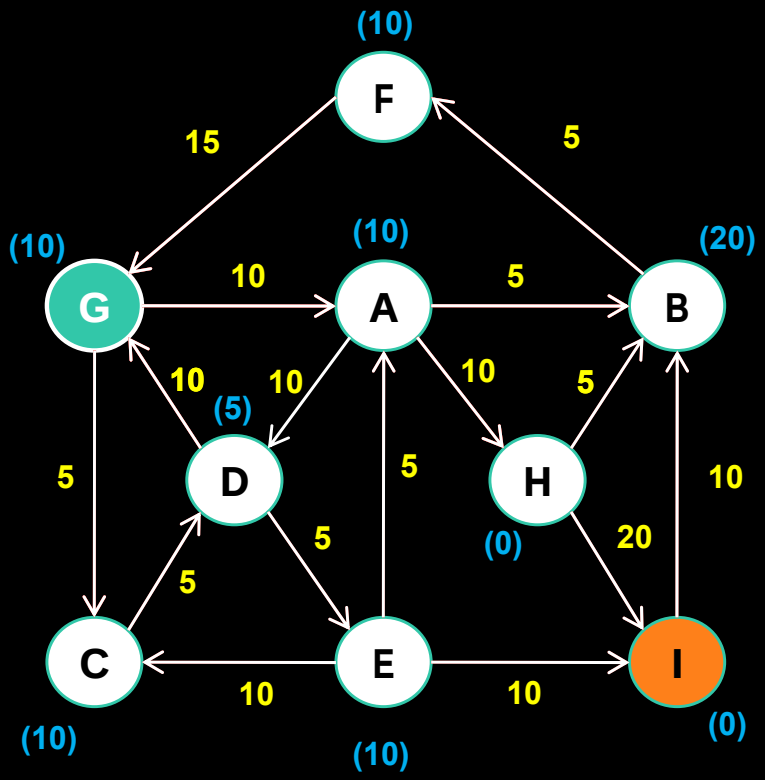


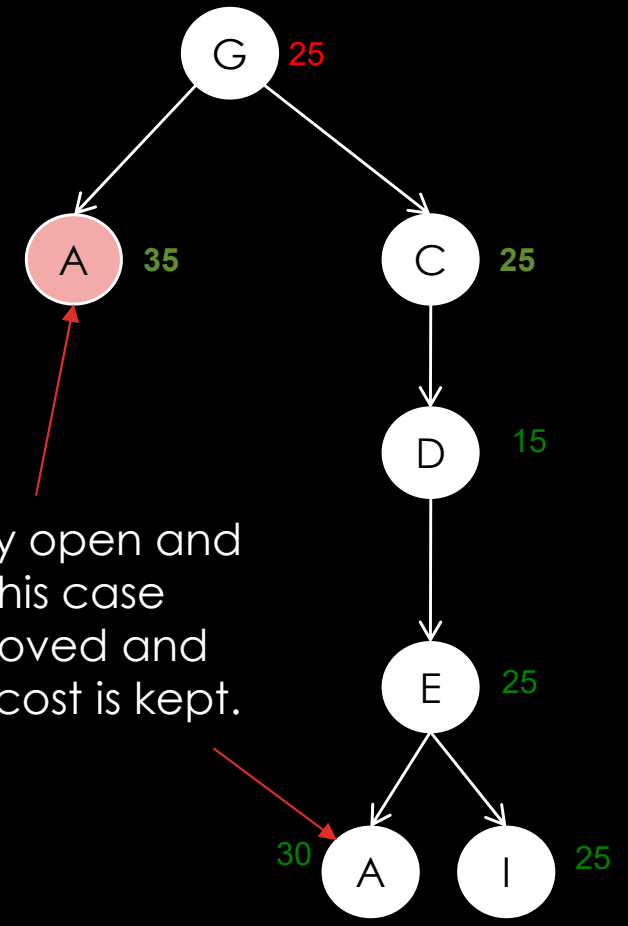
RBFS ALGORITHM





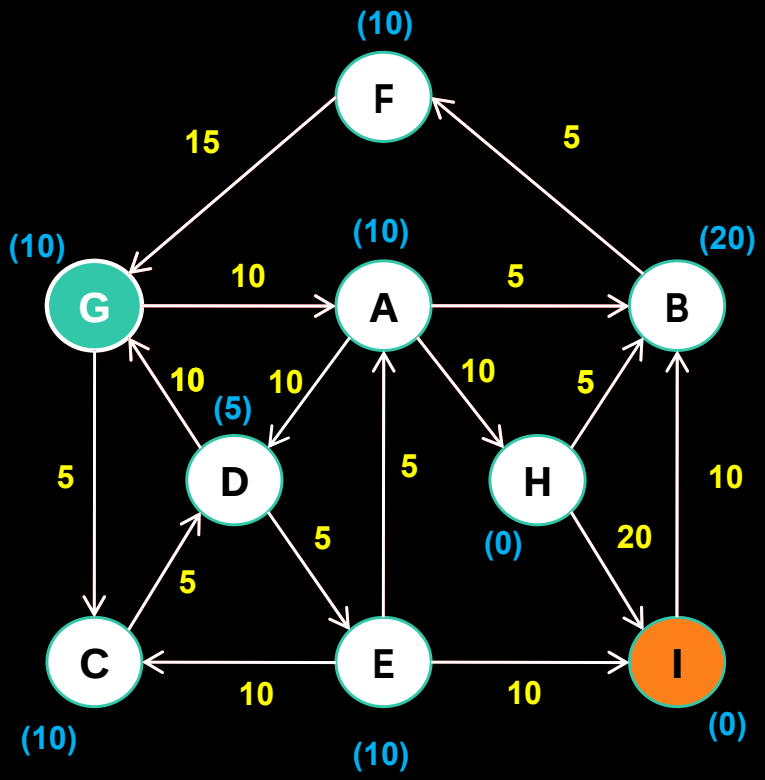
RBFS ALGORITHM

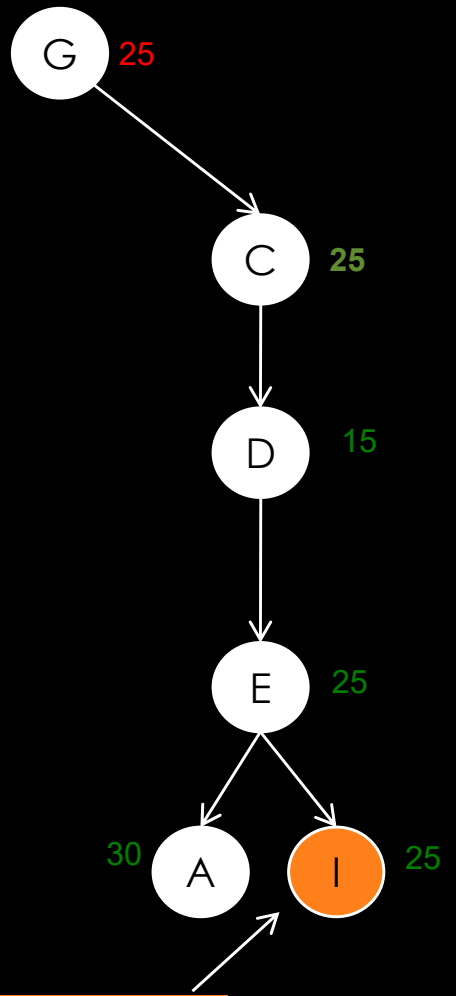




Note: Node A is already open and has higher cost. In this case the old node is removed and the one with lower cost is kept.

RBFS ALGORITHM



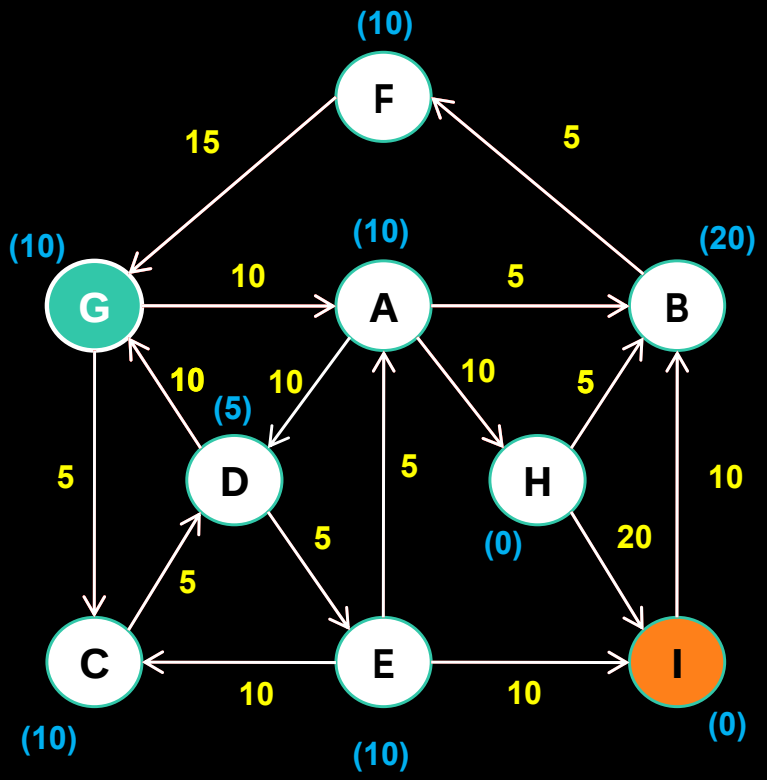


Stop and give the solution: G, C, D, E, I

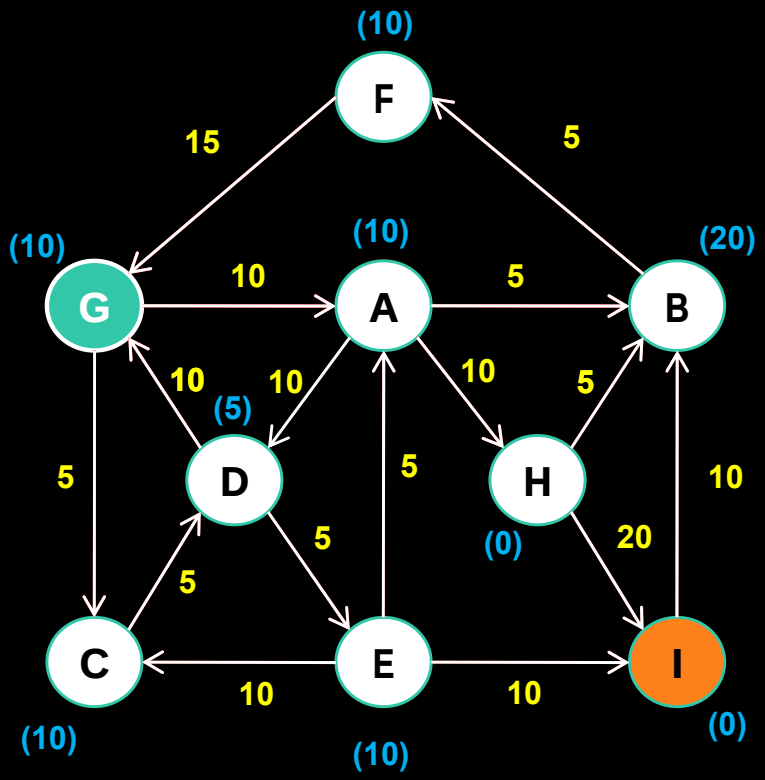
COMPARATIVE ANALYSIS SIMULATION

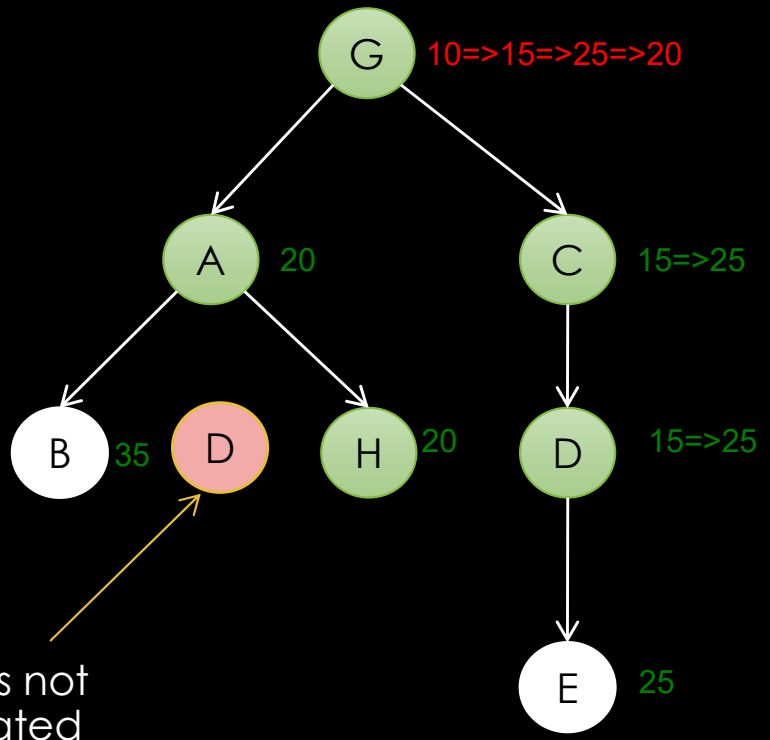
- A*
- IDA* - Iterative Deepening A*
- RBFS - Recursive Best First Search
- SMA* - Simplified Memory-Bound A*

SMA* ALGORITHM (MAX = 8 NODES)



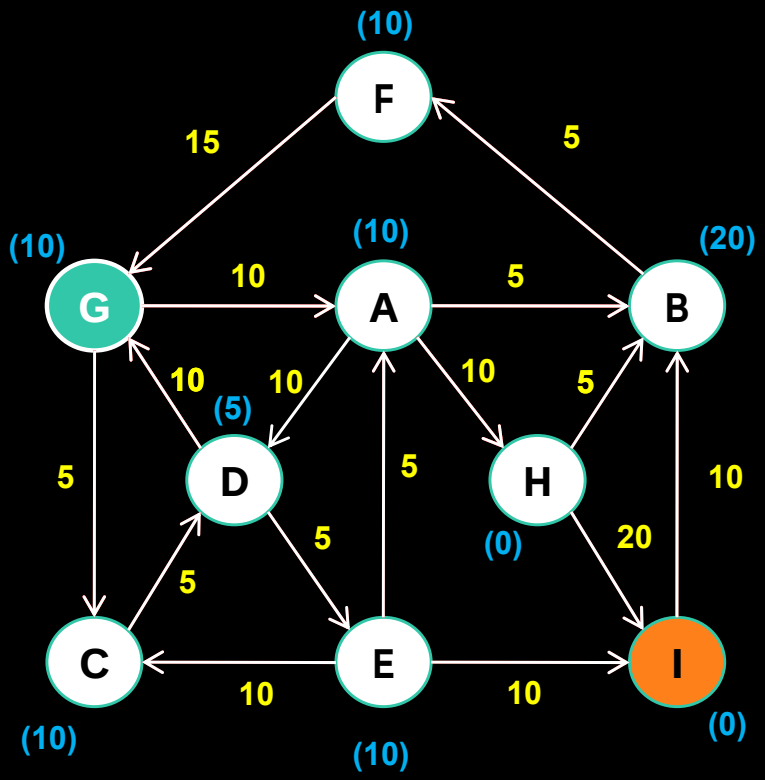
SMA* ALGORITHM (MAX = 8 NODES)

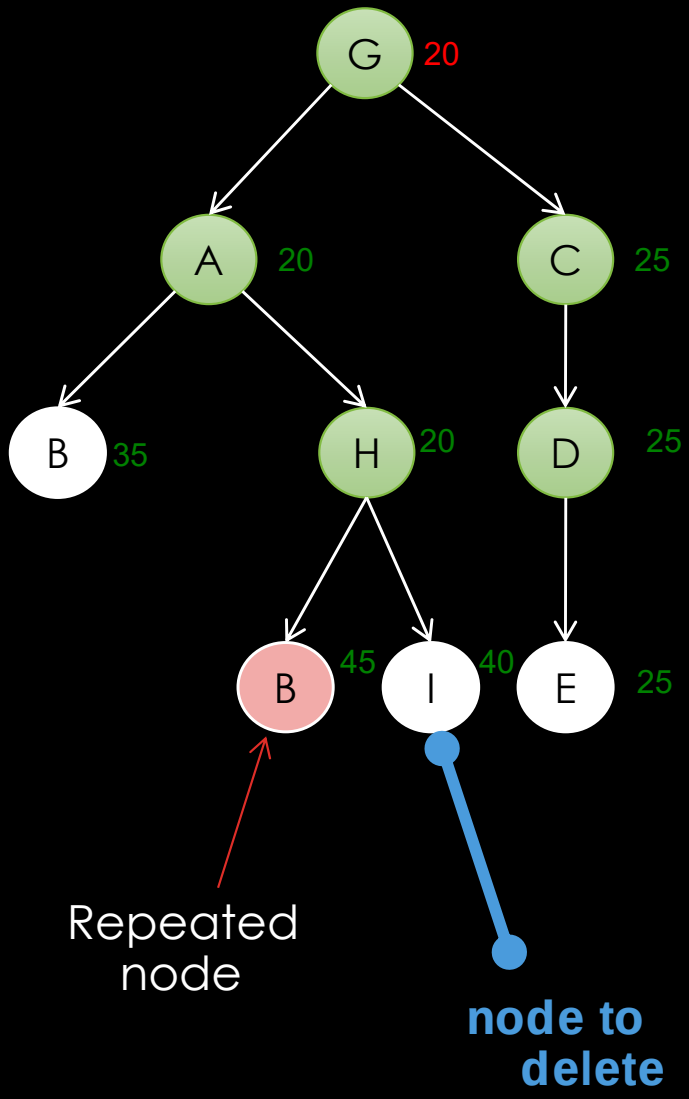




node D is not regenerated because the state would not have a lower cost than the closed node (25)

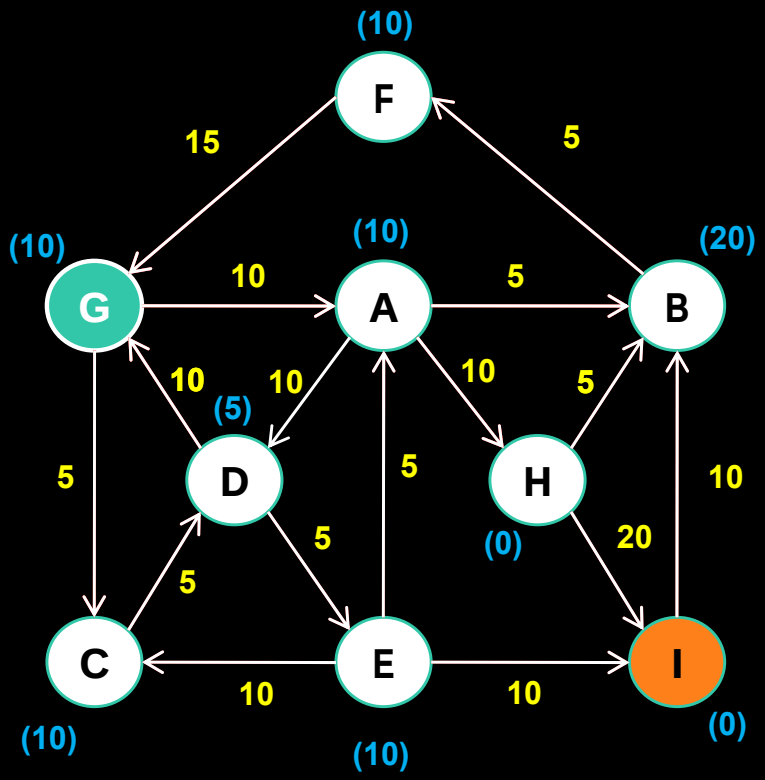
SMA* ALGORITHM (MAX = 8 NODES)

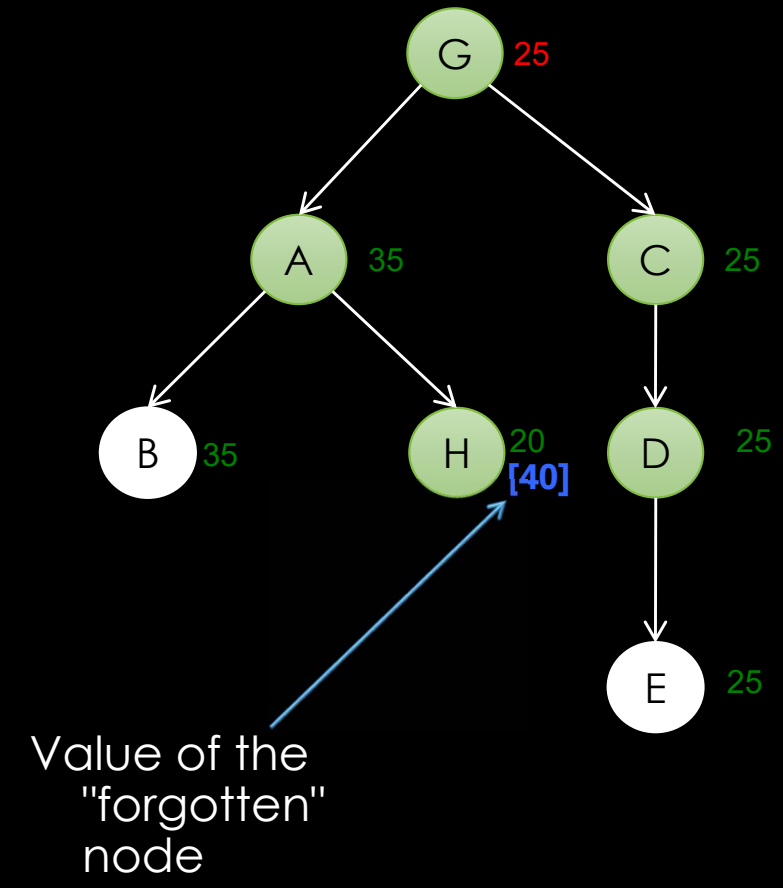




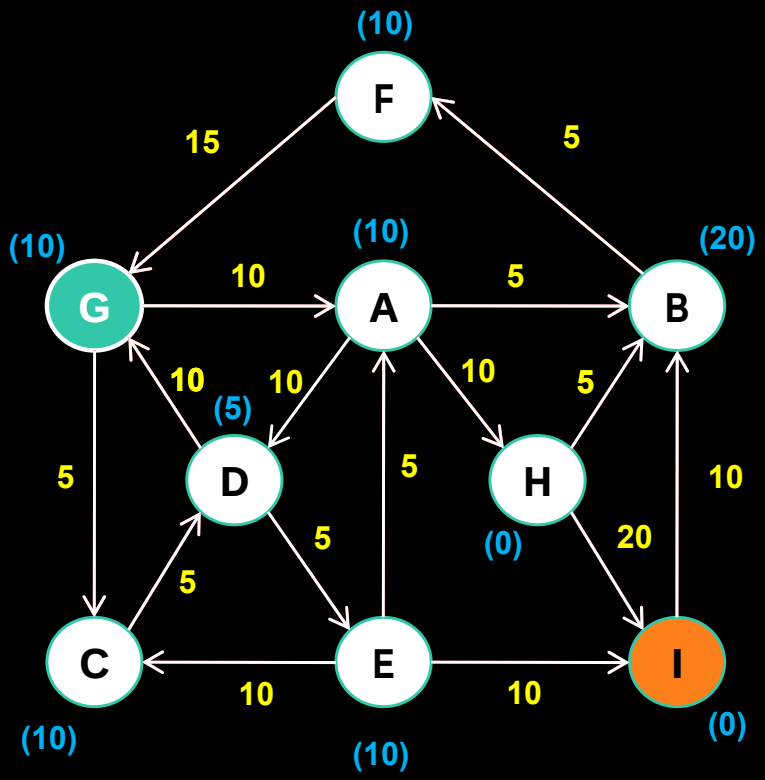
So far the normal behavior of A* has been observed, but the maximum memory capacity has been reached.

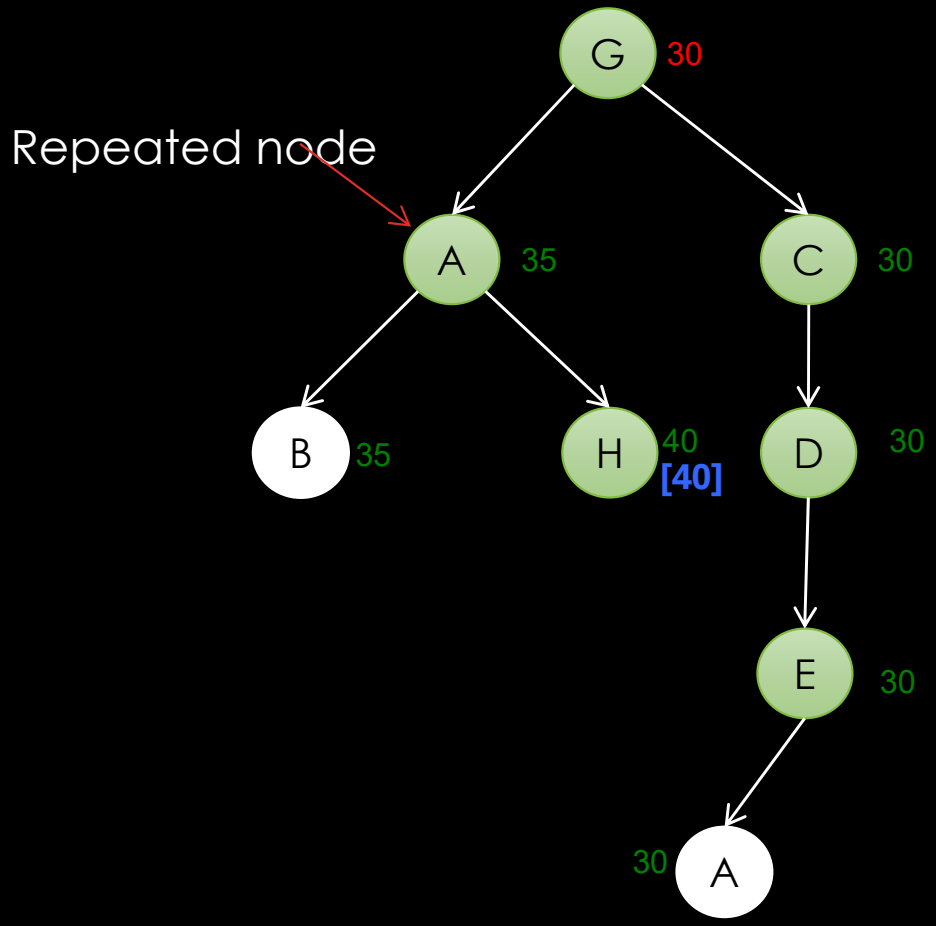
SMA* ALGORITHM (MAX = 8 NODES)



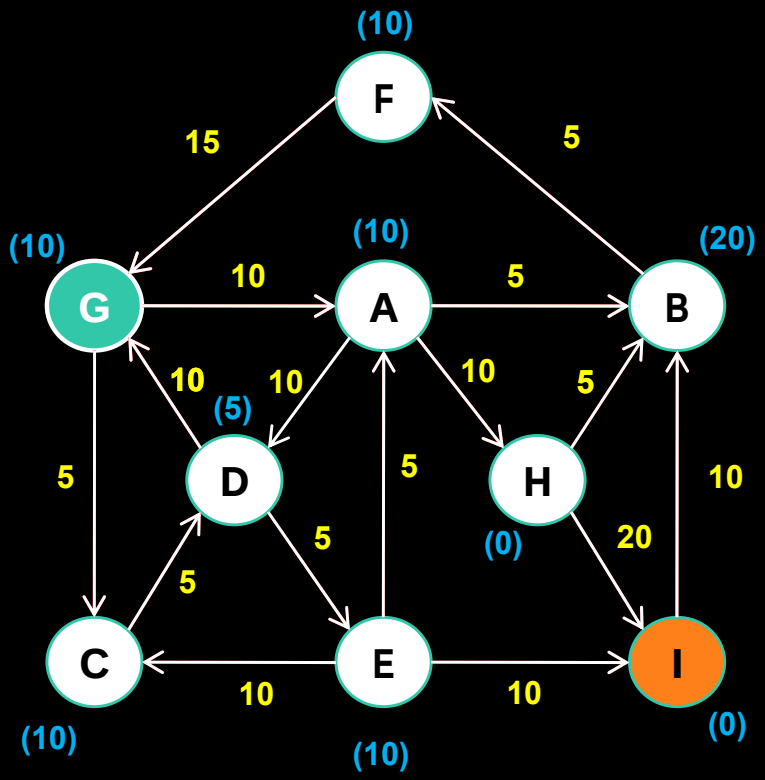


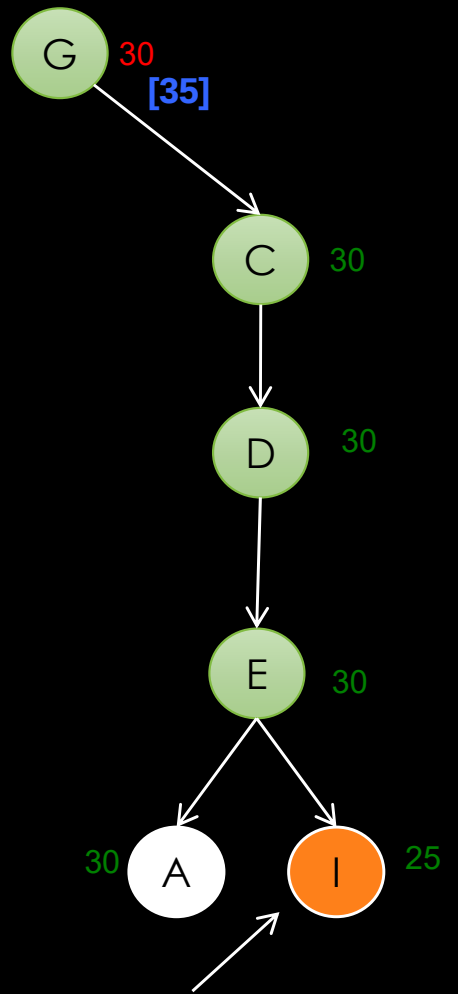
SMA* ALGORITHM (MAX = 8 NODES)





SMA* ALGORITHM (MAX = 8 NODES)





Stop and give the solution: G, C, D, E, I

EXERCISE 1: THE BRIDGE

- Four people want to cross from one side of a bridge to the other. On the bridge they can only pass two people at a time. It is night time and to cross the bridge you need a torch. There is only one torch. Each of the people takes a different amount of time to cross the bridge: one takes 1 minute, one takes 2 minutes, one takes 5 minutes, and the last one takes 10 minutes.
- Present
  - a description for the **state of** the problem.
  - Display the list of **operators**
  - a rule for final state evaluation (consider that the minimum time is 19 minutes).

CONT.

- Solve the problem on paper indicating the **initial state**, the **objective state**, the **operators** and **states** from the initial state to the objective state.
- State the first 4 steps of the algorithm A* using the following function f'(n):
- f'(n) = g(n) + h'(n), where:

  g(n) = Time already spent

  h'(n) = Estimate of the time that will be spent equal to the sum of the times of the people at the initial margin.

  Is this heuristic admissible? Justify.

EXERCISE 2: TOWERS OF HANOI

- Consider the Tower of Hanoi problem where you want to move n disks of different sizes stacked from one position to another, and there is an intermediate position. You can only move one disk at a time, and you cannot place a larger disk on top of a smaller one.
- Define the problem's state space, operators, and a solution evaluation function.
- Write the search tree using the BF algorithm with n=4 from the following starting position: ((1 3) (2 4) ())