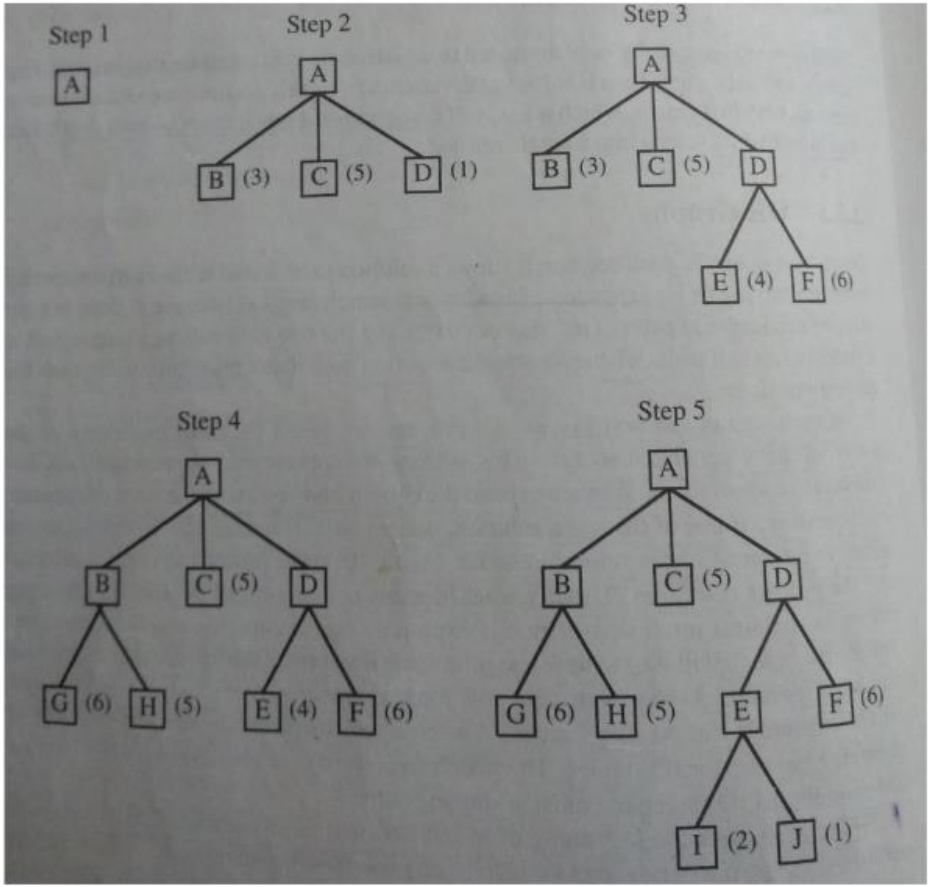
Best First Search and A*

Best First Search

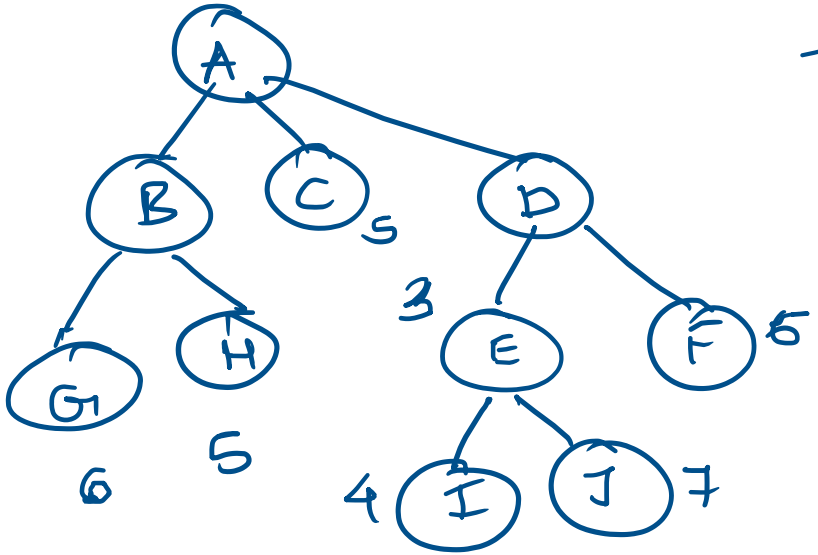
- Way of combining the advantages of both breadth first search and depth first search
- DFS: Good, as it allows a solution to be found without all competing branches having to be expanded
- BFS: Good, as it does not get trapped on dead end paths
- Best First Search: Follows a single path at a time, but switch path when some competing path looks better

Best First Search



Best-First Search V/S Steepest ascent hill climbing

1. In hill climbing, one move is selected and all the others are rejected, never to be reconsidered. In best-first search, one move is selected but the others are around so that they can be revisited later if required.
2. If successor is having lower value than current state then hill climbing halts while best-first search selects the best possible state even though it is worst than current state.



→ Even though I is having a greater heuristic value than current → Expanded.

Graph Search

- To avoid duplication of nodes
- The information stored in each node structure:
  - Description of problem state it represents
  - Desirability: how promising a state is
  - A parent link: to recover solution path
  - List of successor: to propagate the improvements
- Algorithm operate by searching a directed graph
- Such a graph is known as ‘OR’ graph as each of its branches represents an alternative problem solving path

Lists of Nodes for Graph Search

- **OPEN**
  - Nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined [generated but not explored]
  - Maintained in a priority queue
  - High priority → high promising value
- **Closed**
  - Nodes that have already been examined
  - Kept in memory to avoid duplication of nodes

- **Heuristic function for evaluation:**

$$f' = g + h'$$

g : Measure of the cost of getting from initial state to the current node [ACTUAL]

h' : Estimation of additional cost of getting from the current node to a goal state

f' : Estimation of total cost of getting from the initial state to a goal state

Best-First Search

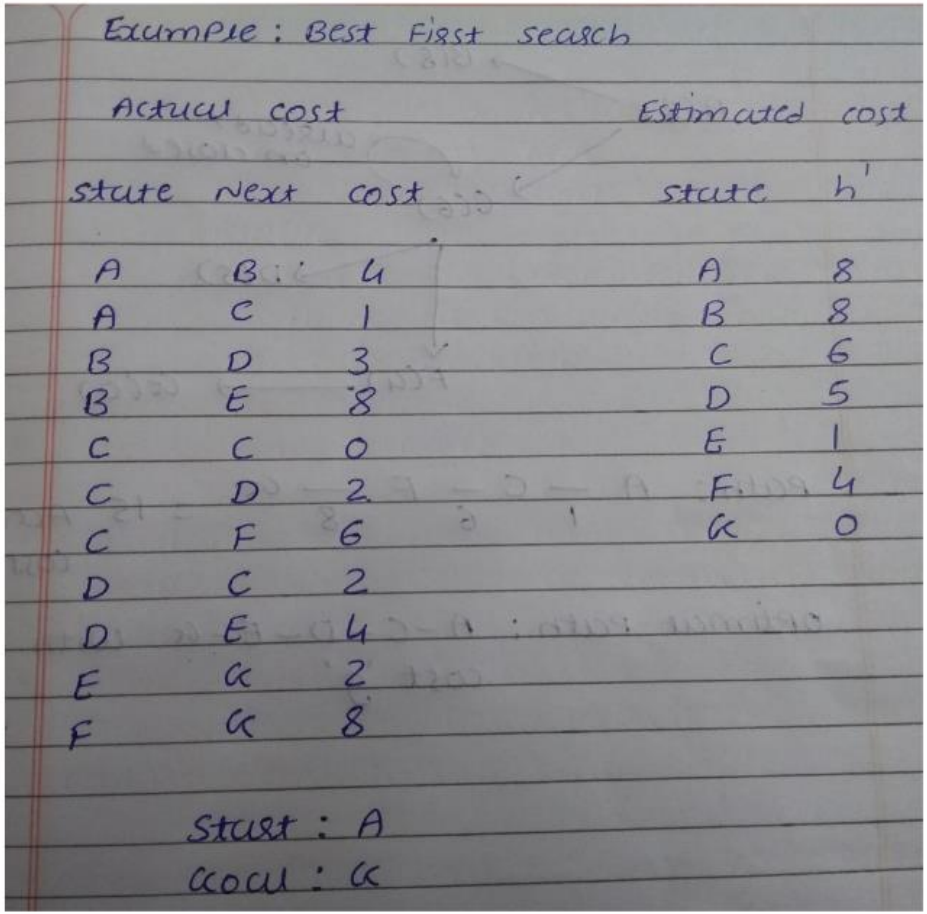
- **Algorithm:**

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 the 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 previous one. In that case, update the cost of getting to this node and to any successors that this node may already have**

Best-First Search

- **Note :**
  - **It selects vertex closest to the goal, does not consider distance from starting point**
  - **It considers only h'**
  - **Greedy in nature**

Best-First Search Example1



Example : Best First search

Actual cost

state	Next	cost
A	B	4
A	C	1
B	D	3
B	E	8
C	C	0
C	D	2
C	F	6
D	C	2
D	E	4
E	G	2
F	G	8

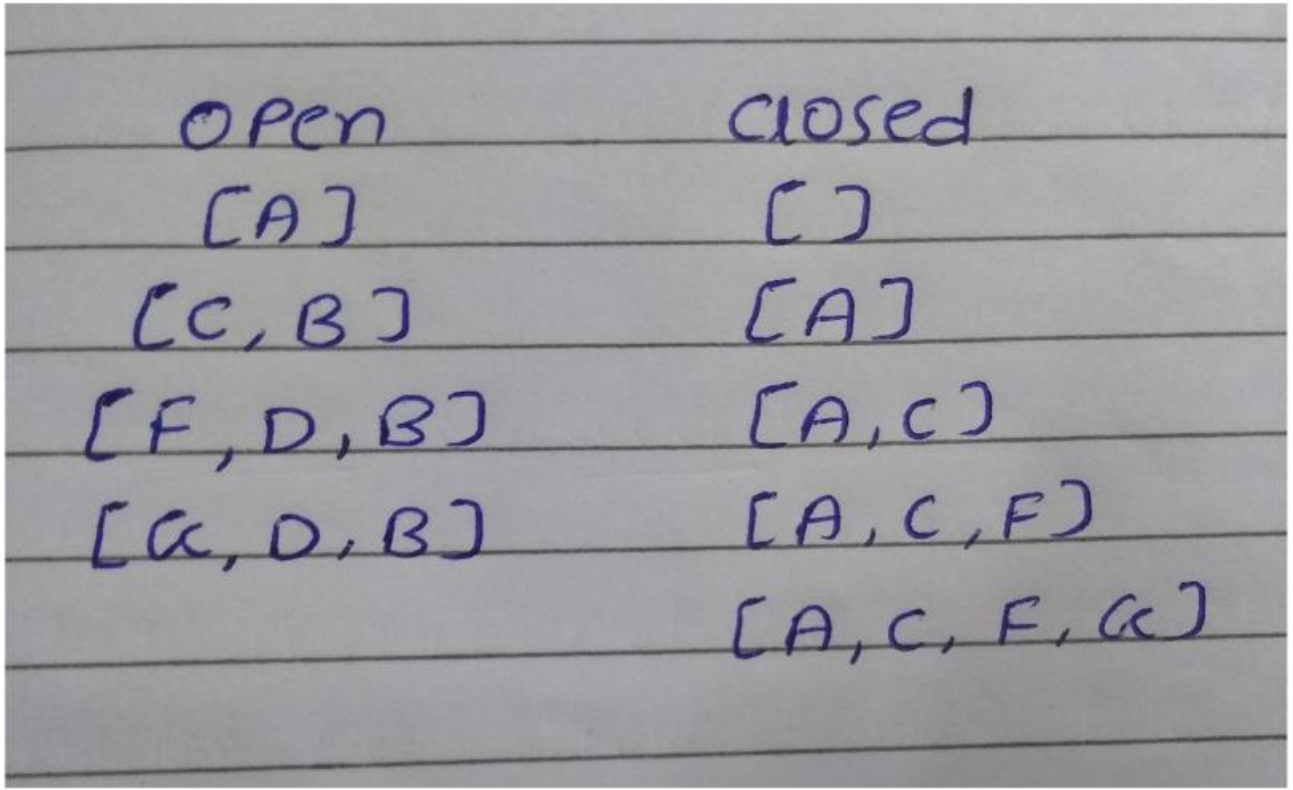
Estimated cost

state	h'
A	8
B	8
C	6
D	5
E	1
F	4
G	0

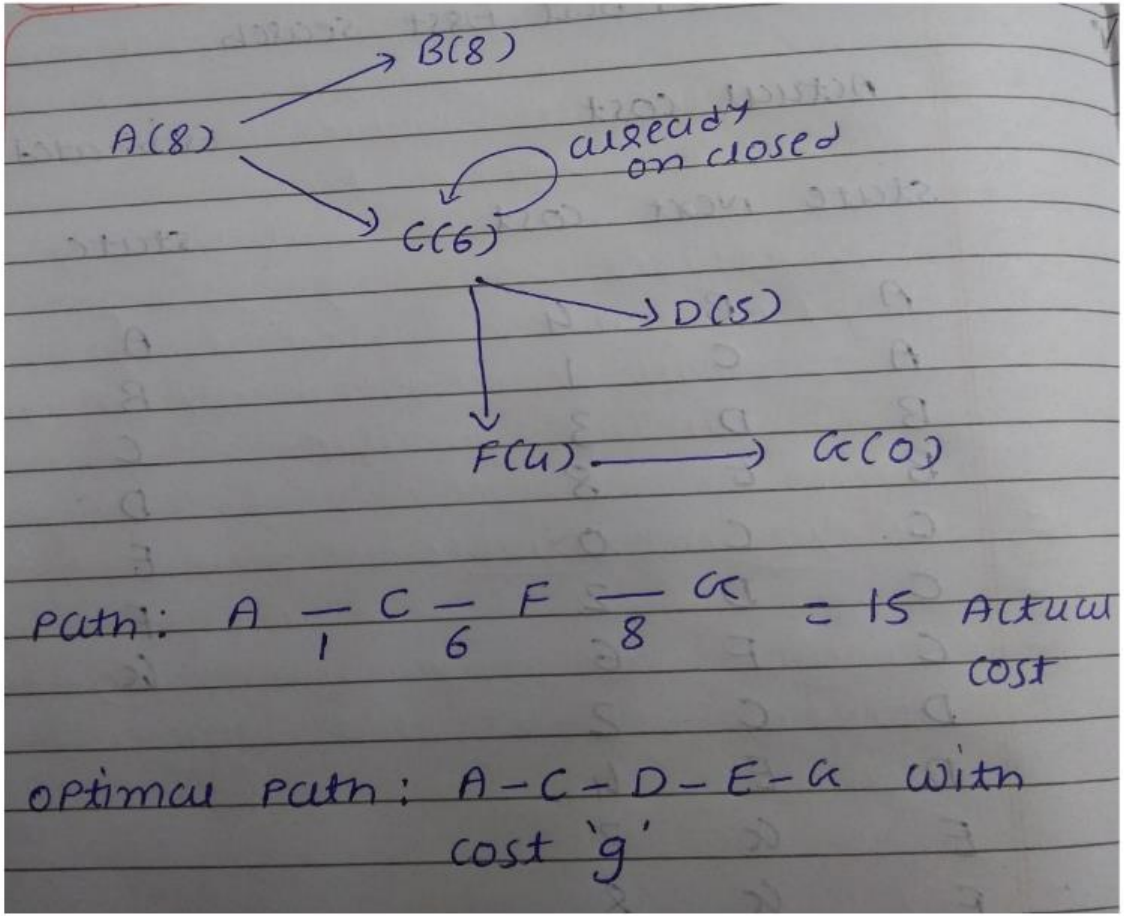
Start : A

Goal : G

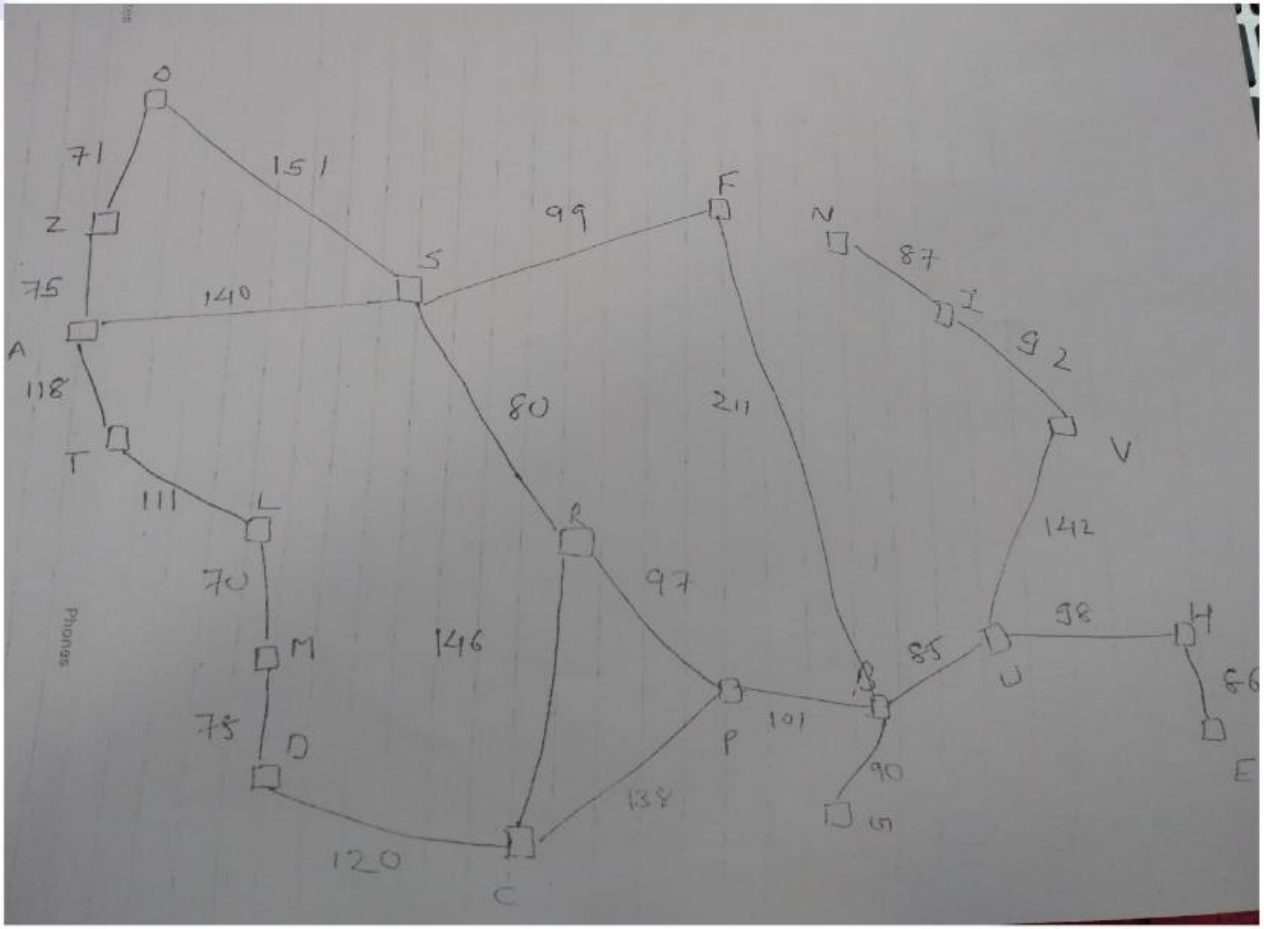
Best-First Search Example1



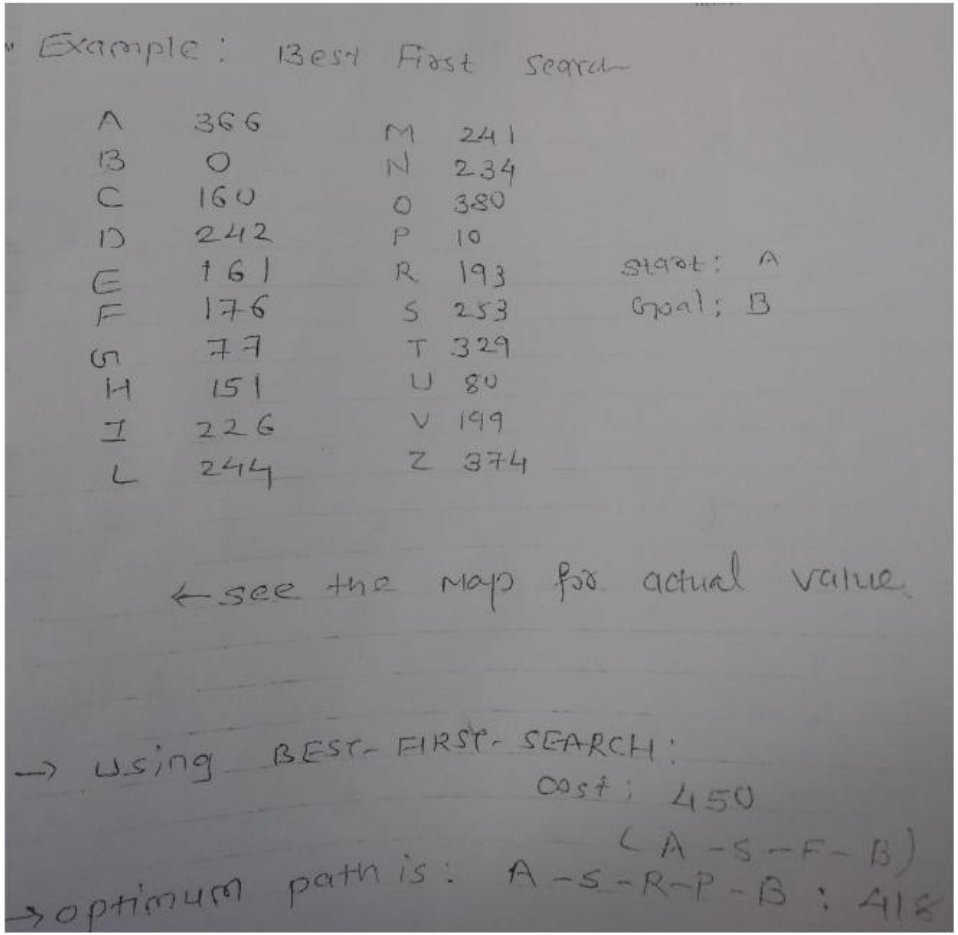
Best-First Search Example1

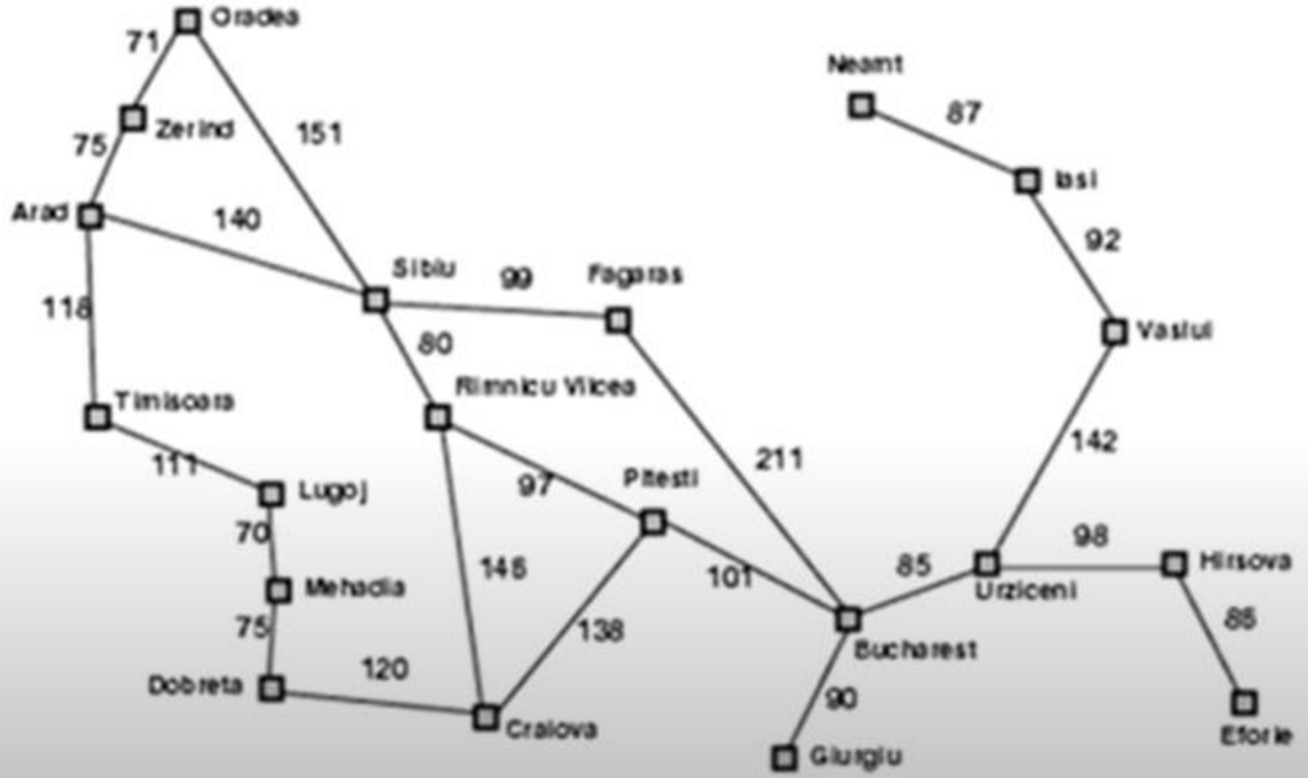


Best-First Search Example2



Best-First Search Example2





Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

II. Best First Search Algorithm

- Greedy in nature
- Does not guarantee to find a shortest path
- It selects the vertex closest to the goal
- It has some estimate (heuristic function value) of how far from the goal any vertex is
- It runs much quicker because of estimated value, which guides its way towards the goal

A*

For the purpose of path finding two algorithms we have studied

I. Dijkstra's algorithm:

- **Guarantees to find a shortest path from starting vertex using relaxation condition for positive weighted edge**

  D[V] = min (D[V] , D[W] + C [W,V])

- **It selects vertex closest to the starting points**
- **It is a slow algorithm**

- **A* combines heuristic approaches like greedy Best First Search and formal approach like Dijkstra's algorithm**

$$f' = g + h'$$

Algorithm

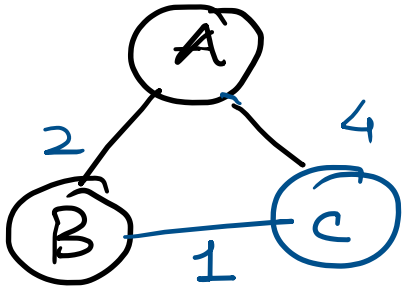
1. Start with OPEN containing only the initial node. Let us call it S. Set its g value to 0 and h’ value to whatever it is. Set its f’ value to F’=g+h’ = 0+h’ = h’.
2. Closed list = Empty.
3. Until a goal is found repeat the following:
    a. No node in OPEN -> Failure.
    b. Otherwise Pick a node in OPEN with lowest f’ value. Call it **bestnode**. Remove it from OPEN. Place it in Closed. Check it is the goal node. If it is , then exit. Otherwise generate successors of bestnode. Do not point to them yet. First we need to check if they have already been generated.
        i. For each successors do the following:

a. Compute the value of g(successor). g(successor) = g(bestnode) + cost of getting from bestnode to successor.
b. See if successor is already in OPEN. If so, then check if the successor parent link should be reset to bestnode. It should be if the path we have found to successor is cheaper than the old path. If old path cheaper then do nothing. Otherwise reset the successor parent link to point to best node and record the updated g value and f value of node.
c. If successor is not in OPEN, see if it is in CLOSED.

Cont.

If it is in CLOSED, see if the new path is better than the old path. If we have found a new path that is better than the old path, then we must propagate the improvement to the node's successors. This is a little bit tricky task. The successor points to its successors and so on. To propagate the new cost downward we may do depth-first search traversal of the tree starting from the successor.

D. If the successor is neither on OPEN or CLOSED then add the successor to OPEN and add it to bestnode's successor. Compute its f' value.



→ Now,

$h(A) = 2$

$h(B) = 4$

$h(C) = 3$

→ OPEN → A

→ Now, calculate $f(A) = g(A) + h(A)$

$g(A) = 0 \quad h(A) = 2$

So, $f(A) = 2$.

→ Bestnode = A.

→ Successors of A → B and C.

⇒ So, B → Not in OPEN, nor in CLOSED.

C → Not in OPEN, nor in CLOSED.

⇒ Check. f(B) →

f(C).

$g(B) = g(A) +$ cost of getting from A to B

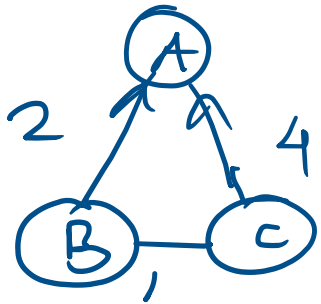
$= 0 + 2 = 2$

$f(B) = 2 + 4 = 6$

$g(C) = g(A) + \text{cost of getting from A to C}$

$= 0 + 4 = 4$

$f(C) = 3 + 4 = 7$ (Parent link A)


A
2
4
B
C
1

Now, Choosing B. Bestpath = B

→ So, successor of B

→ So, C is successor of B.

So, now. $f(c) = g(c) + h(c)$

$g(c) = g(\text{betnode}) + \text{cost of getting from betnode to C}$

$= 2 + 1 = 3$

$<$ initial $g(c)$

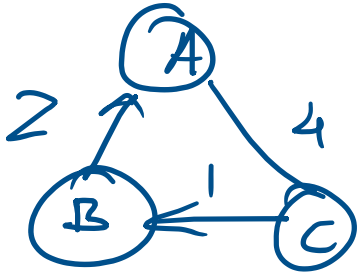
So, update $g(c) = 3$

and the cost of

$f(c) = 3 + 3 = 6$

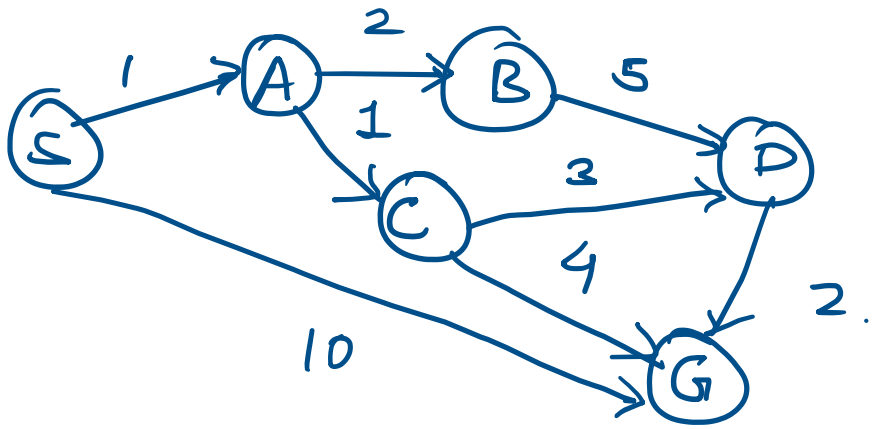
Now, parent of c → updated.

→ which is



→ And the goal = cost = 3

Instead of Best first search cost = 4



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Actual cast g(n)

Start	Next	Cast
S	A	1
S	G1	10
A	B	2
A	C	1
C	D	3
B	D	5
C	G1	4
D	G1	2

Estimated cast

State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution:-

→ start with Initial Node → S →

OPEN = S = Bestnode. , CLOSED = Empty.

→ Now, Successors of S,

A and G.

→ $g(A) = g(\text{Bestnode}) + \text{cost of getting from Bestnode to A.}$

$\therefore \quad g(A) = 0 + 1 = 1$

$f(A) = g(A) + h(A) = 1 + 3 = 4.$

$$g(G) = 0 + 10 = 10$$

$$f(G) = 10 + 0 = 10$$

→ So, choosing with minimum f value.

OPEN = [A, G] CLOSED = S

→ Choose node 'A'. Bestnode = A.

→ successors of A are B, C

so, g(B) = g(Bestnode) + cost of getting from BestNode to Node B.

$$g(B) = 1 + 2 = 3$$

$$g(C) = 1 + 1 = 2$$

$$f(B) = 3 + 4 = 7$$

$$f(C) = 2 + 2 = 4.$$

$f(c)$ is less. So, select Node C. Bestnode = C.

OPEN = [G, C, B] CLOSED = [S, A]

Now, here C is selected

successors of C.

→ G, D. Now,

G already in OPEN. with parent

pointing to S.

Check new g(G1) value for node G1.

g(G1) = g(Bestnode) + cost of getting from Bestnode to G1.

= 6 which is less than previous cost.

From all the nodes in the OPEN, the one with lowest f value is selected next to explore. So, here Node B has f value 7, node D has f value 11, node G has f value 6. So, Node G is selected and is checked whether it is the goal node. Since it is the goal node the search ends there.

So, update the parent link for G.

Now, $f(G) = 6 + 0 = 6$

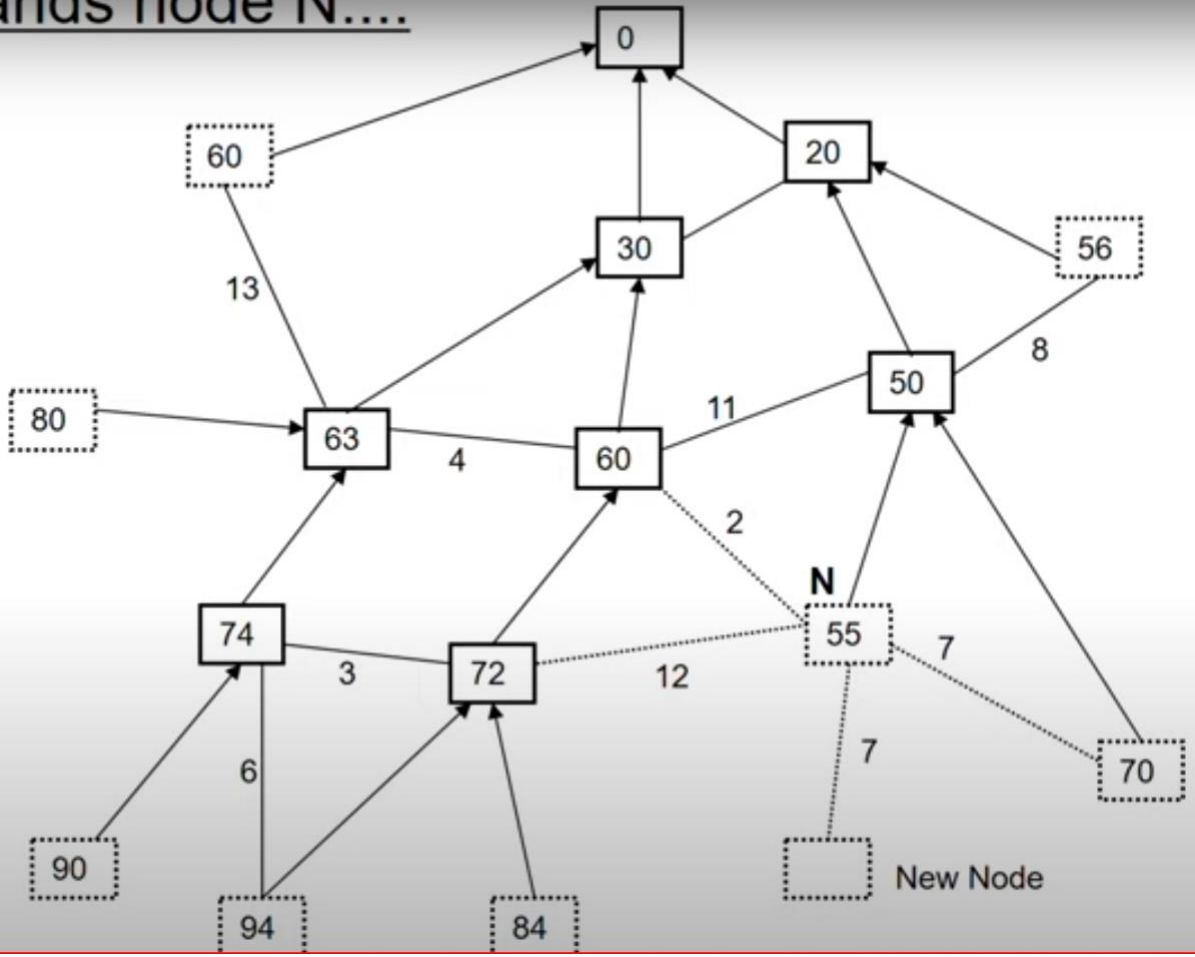
$f(D) = g(D) + h(D) = 2 + 3 + 6 = 11$

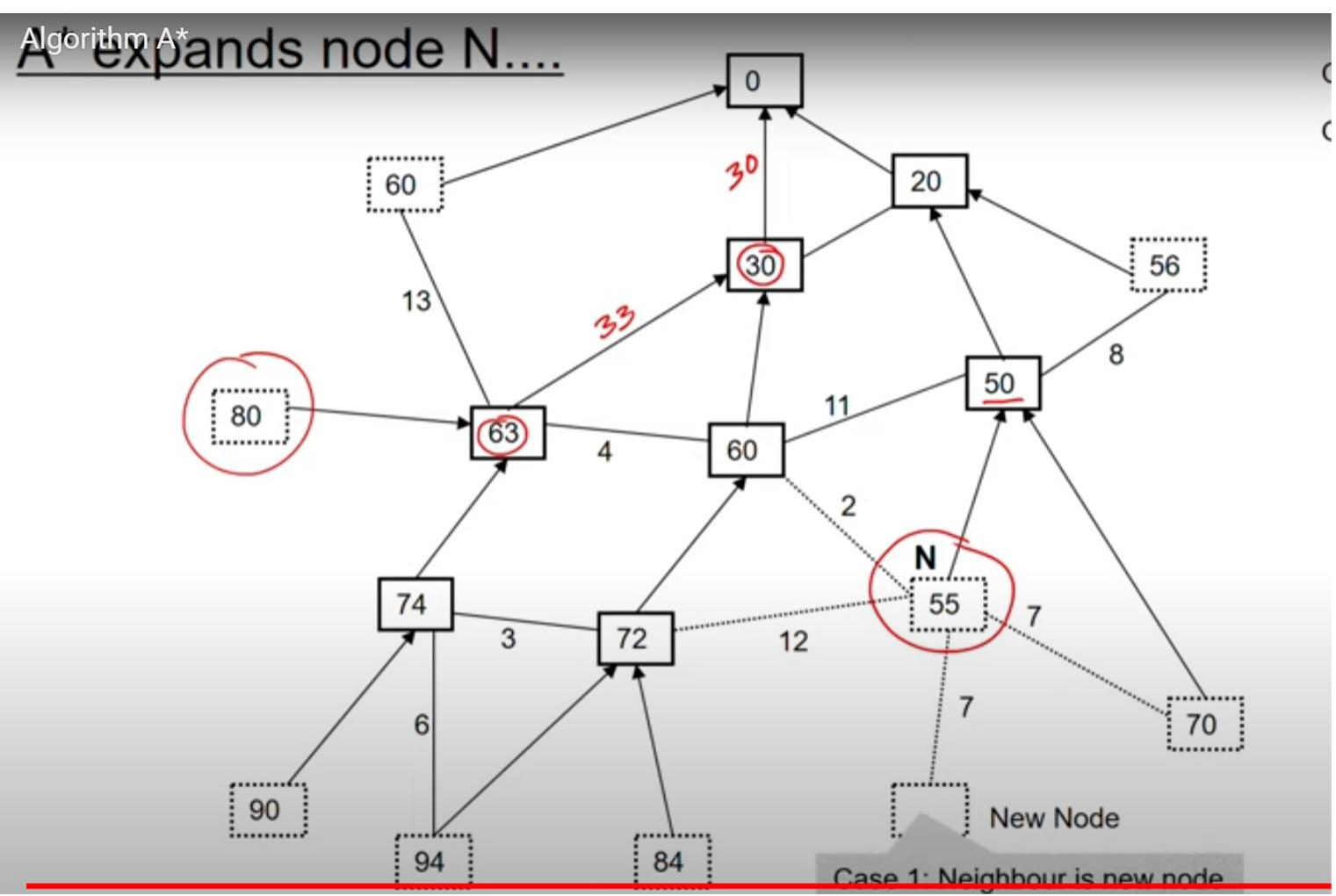
So, select → G. → goal node.

So, total cost = 6 (optimal path)

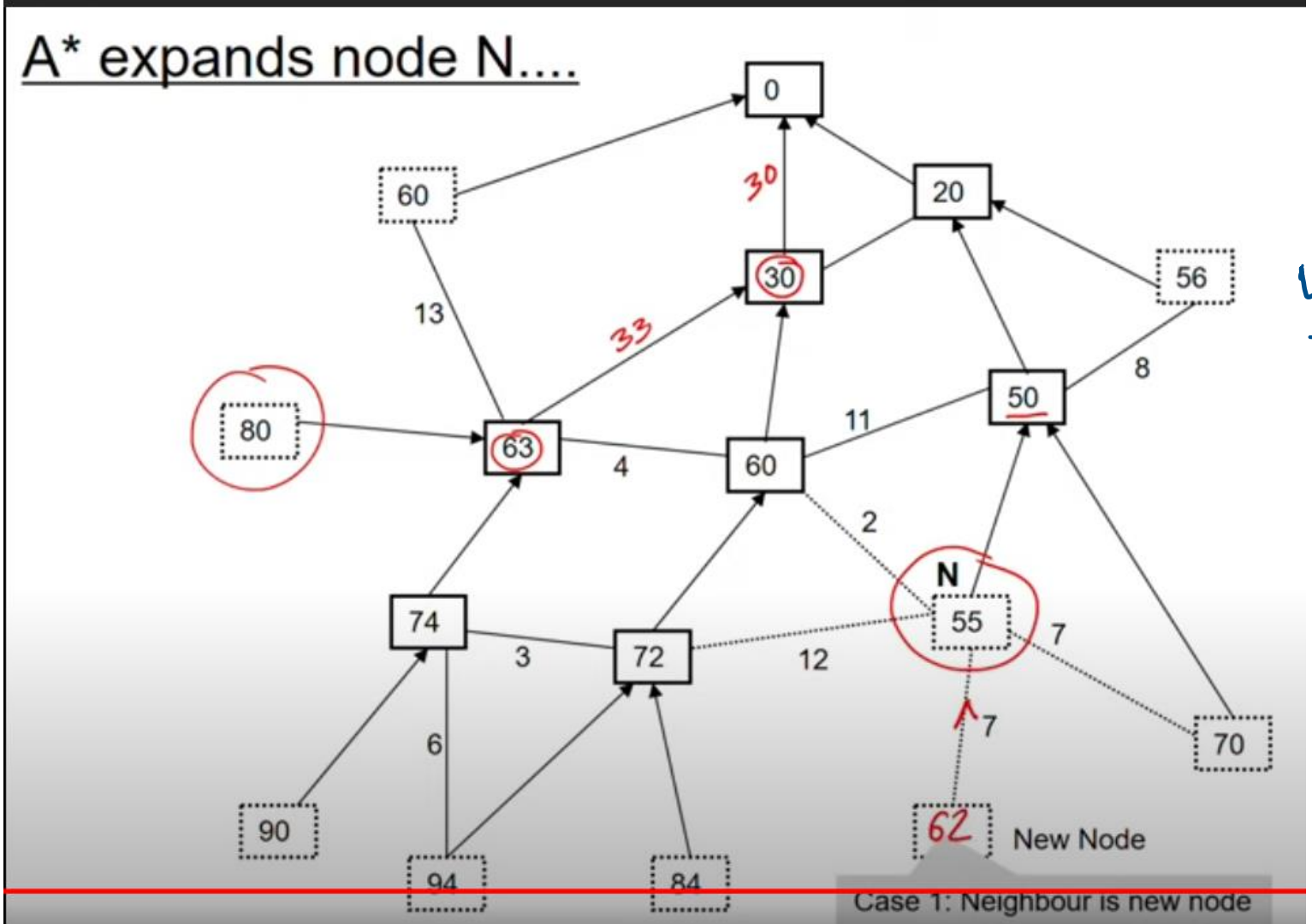
Algorithm A*

A* expands node N....



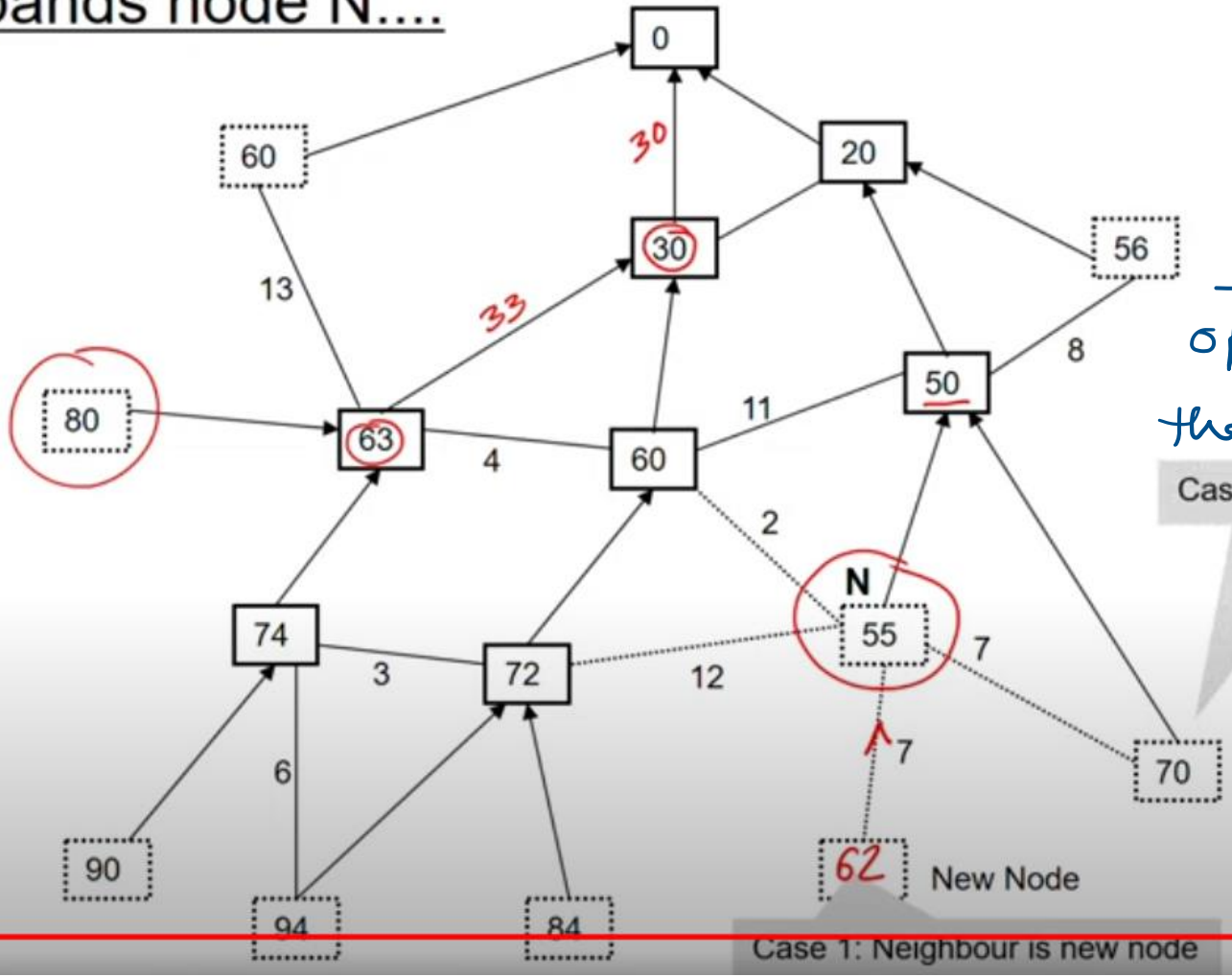


Neighbours of Node N are 60, 50, 72, 70 and newly unexplored node.



Update the g(curr) of new node to 62.

A* expands node N....



CLOSED node g-value

OPEN node g-value

→ Neighbour is open. so compare the newly computed

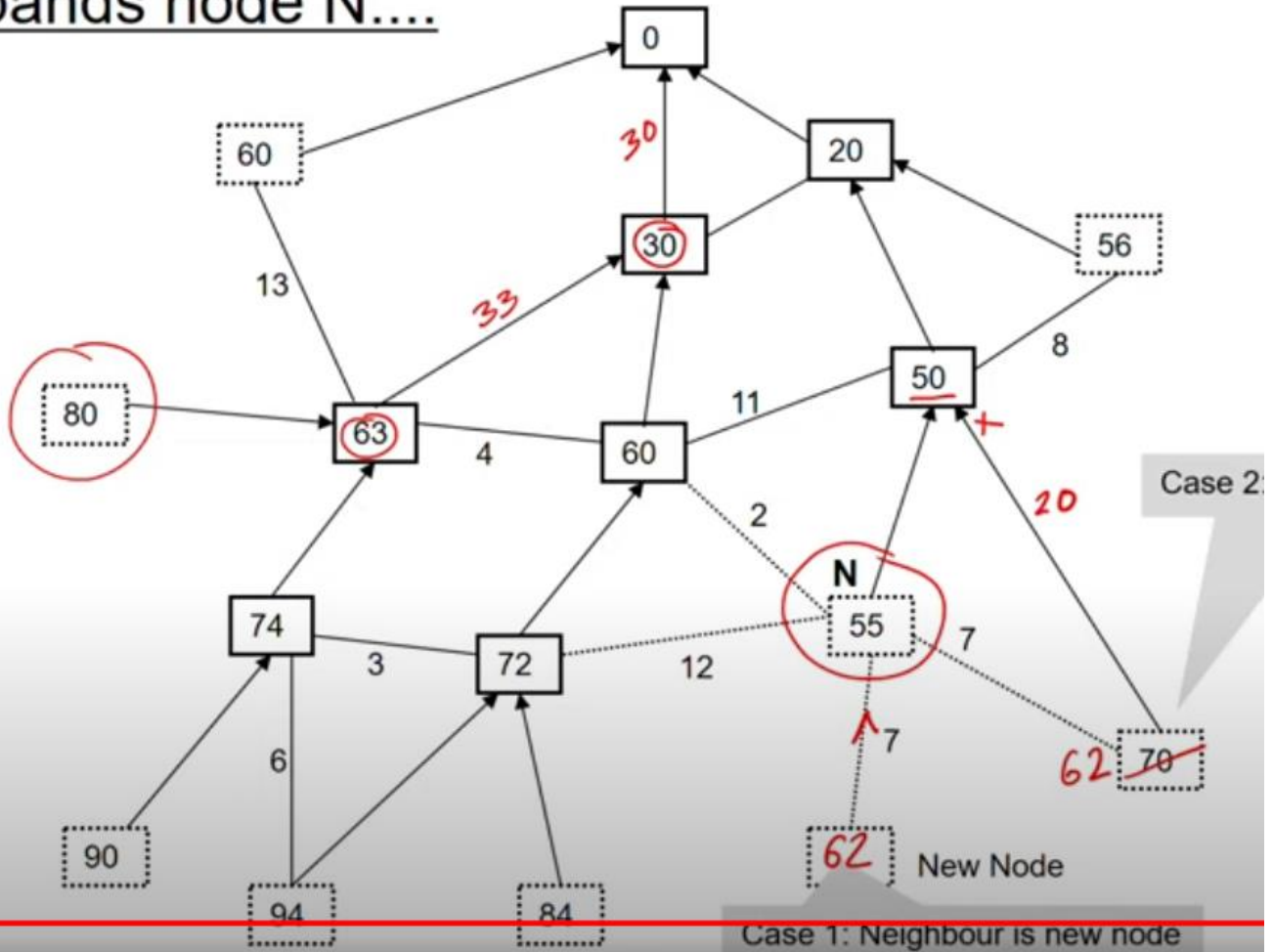
Case 2: Neighbour in OPEN

cost & old cost.

55 + 7 = 62 < 70.

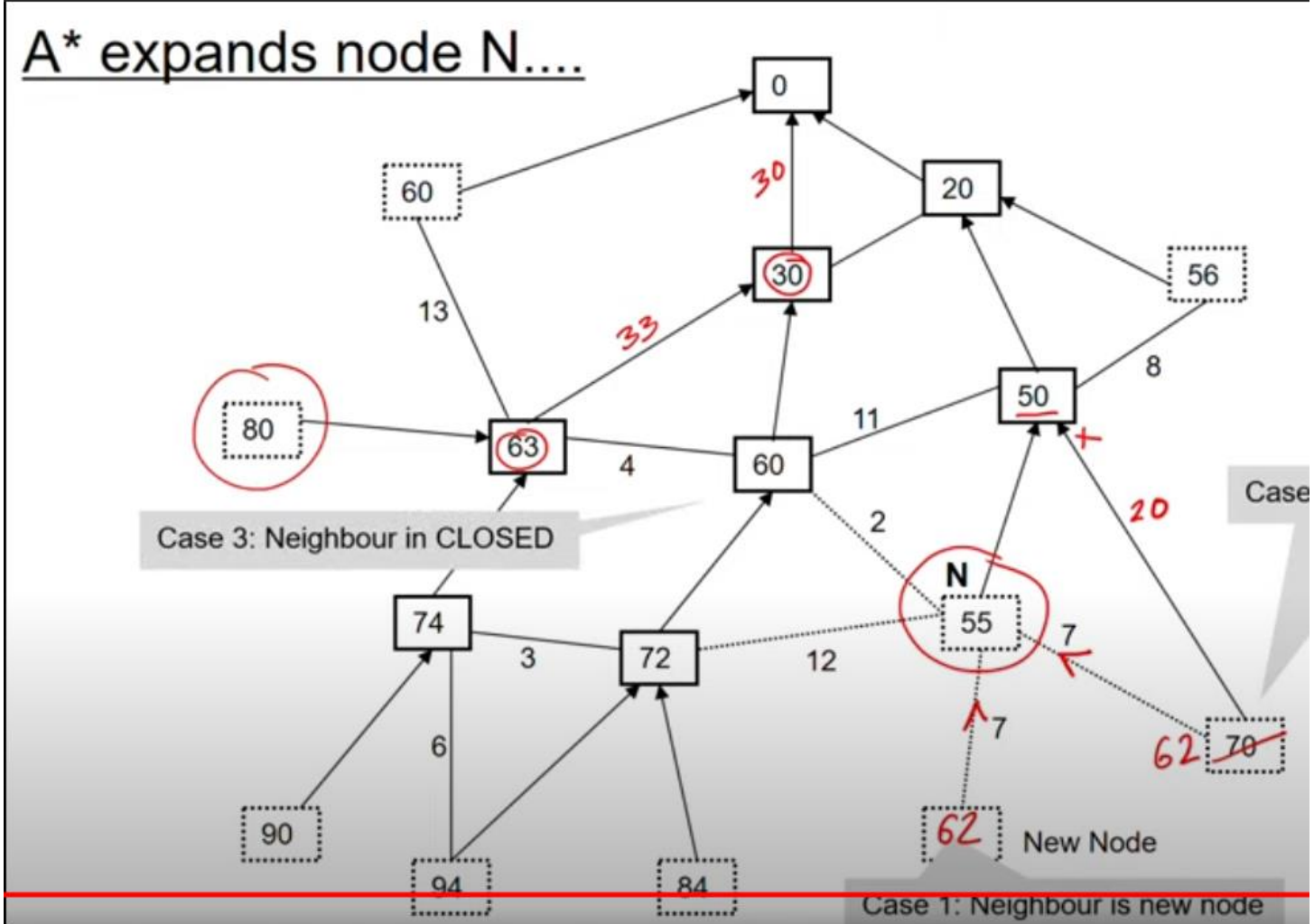
So, update parent link.

A* expands node N....

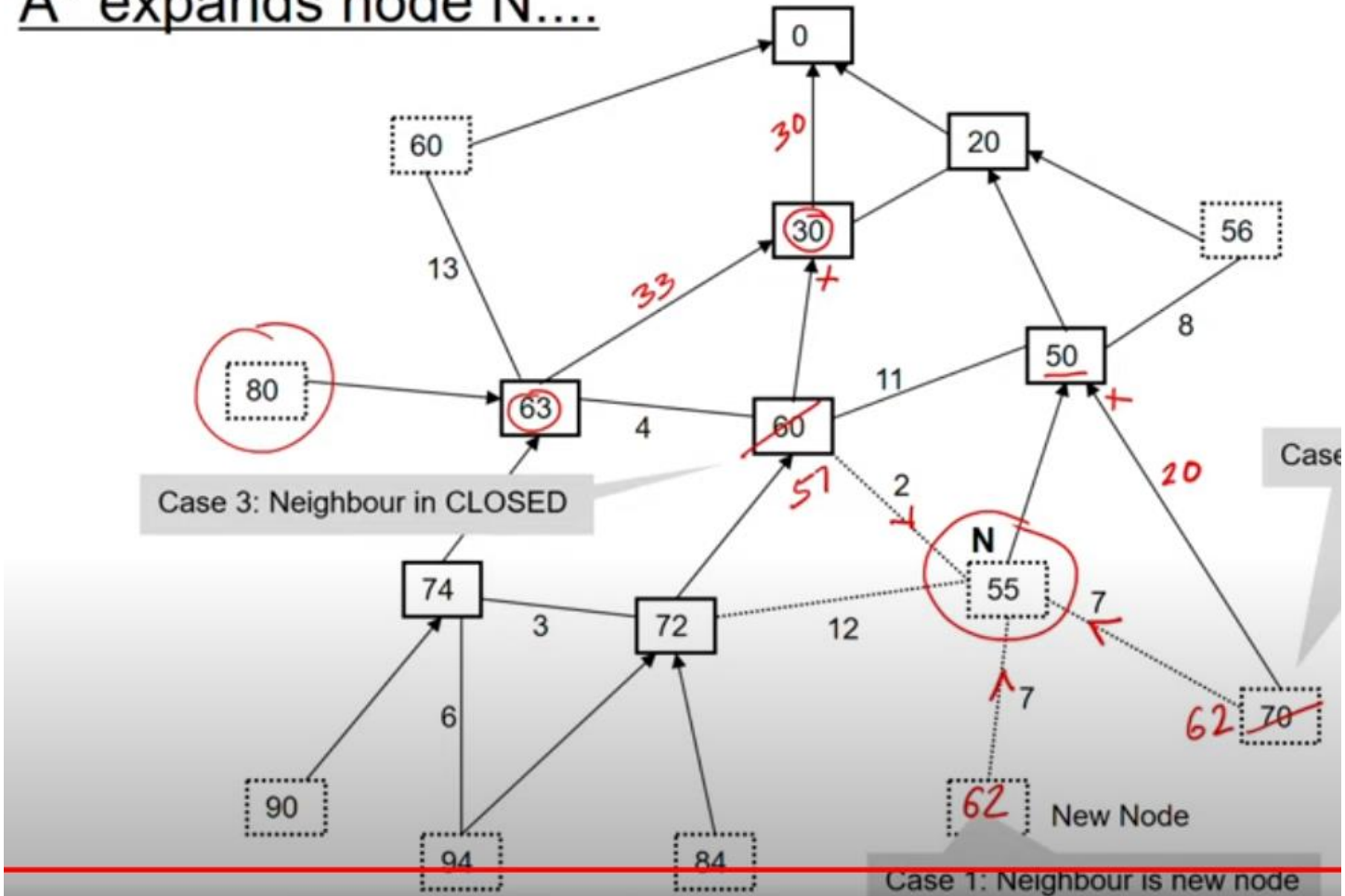


Case 1: Neighbour is new node

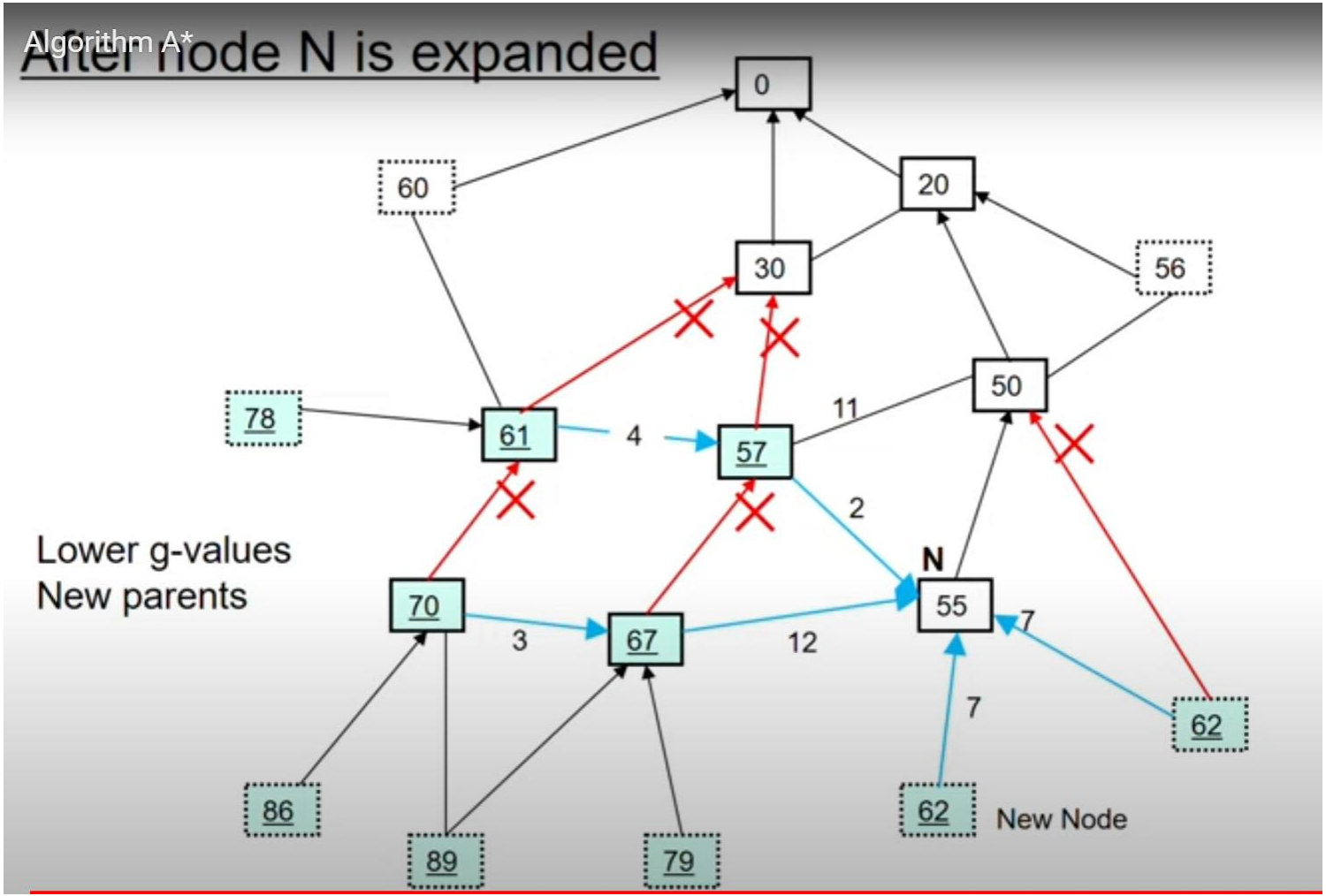
A* expands node N....



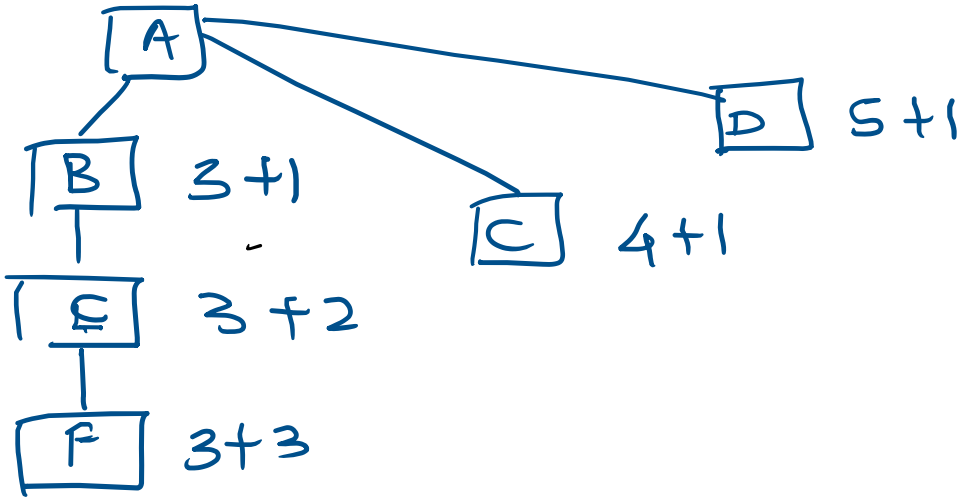
A* expands node N....



Algorithm A*


After node N is expanded
0
60
20
30
56
78
61
4
57
11
50
2
N
Lower g-values
New parents
70
3
67
12
55
7
7
62
86
89
79
62
New Node

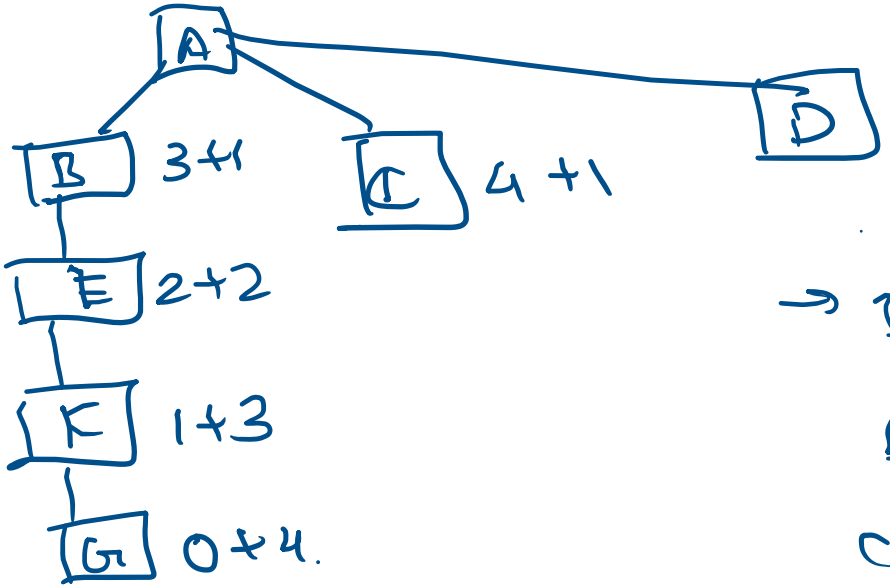
Admissibility of A*



Underestimate

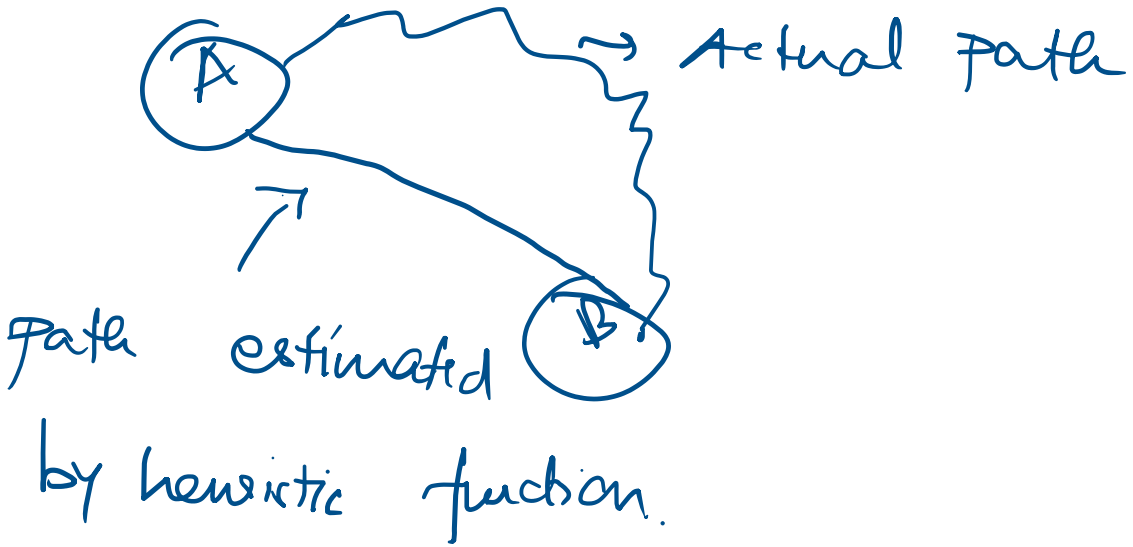
↓

Now using the underestimate (F) path not correct → so move to next



→ D could have been a less costly path.

But overestimating (causes us to not explore the path).


A
B
Actual path
Path estimated
by heuristic function.

References

https://www.gatevidyalay.com/tag/a-algorithm-example-ppt/

https://www.youtube.com/watch?v=qYP6fR8BoxY

https://www.youtube.com/watch?v=JY1IHmFJyfI