

# Informed search

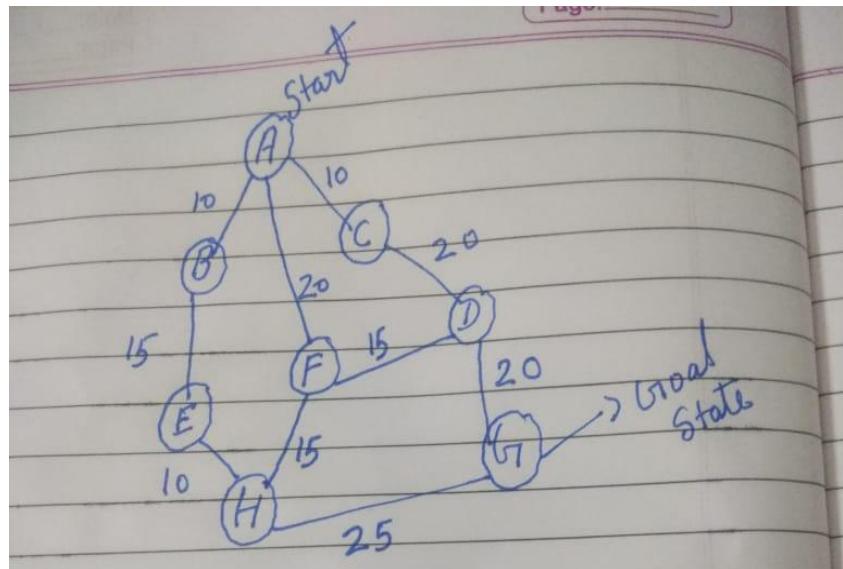
# Recap: Uninformed search algorithms

- Classic search algorithms search only based on information that has already been provided by the problem.
- These algorithms will traverse the whole search tree until they hit a solution, or else exhaust the graph. These are also called **uninformed search algorithms** or **blind search algorithms**.
- **Uninformed search algorithms** become unreliable or even intractable when the problem is more complex.

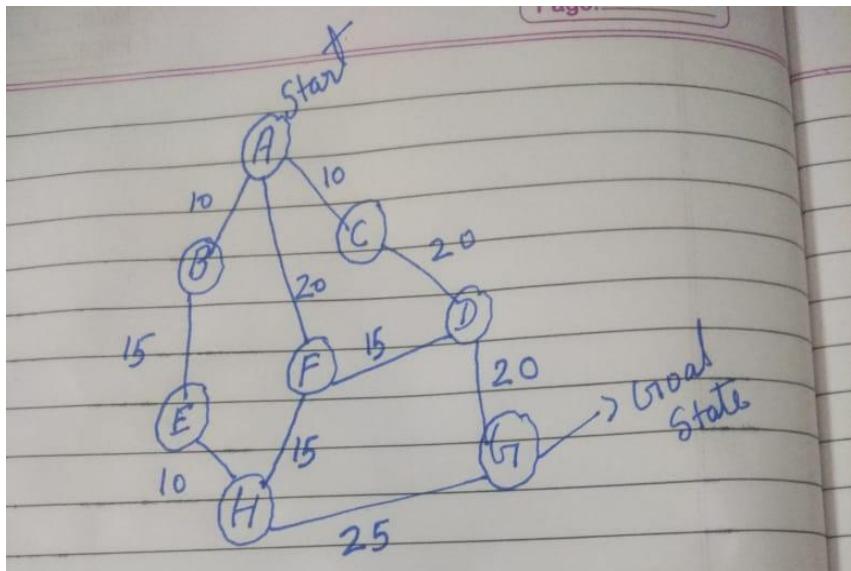
# Informed (Heuristic) search algorithms

- Heuristic search algorithms can use the knowledge beyond the problem definition itself to try paths by order of ‘promise,’ so to find solutions efficiently.
- To get this extra information, heuristic search algorithms use a heuristic function  $h(n)$ .
- Heuristic search algorithms also employ an evaluation function to help decide which nodes to explore first.
- **Heuristic search algorithms** also employ an **evaluation function** to help decide which nodes to explore first.
- Usually **heuristic search algorithms** will explore nodes in ascending order of associated value of evaluation function.
- The algorithms vary primarily across the choice of evaluation function.

# Greedy best-first search



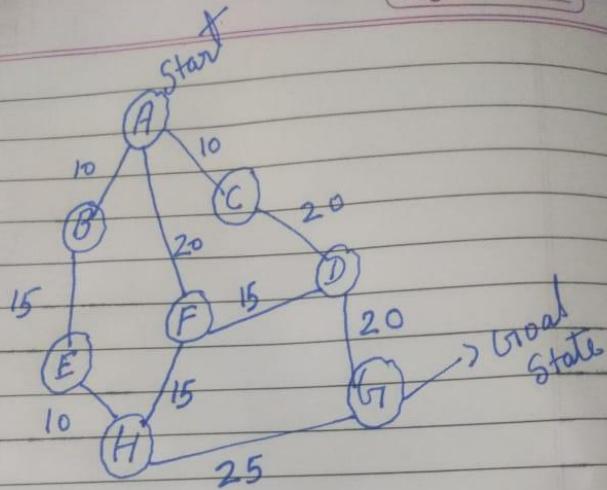
# Greedy best-first search



Heuristic  $\rightarrow$  Straight Line distance

$A \rightarrow G$	$= 3.5$	{}
$B \rightarrow G$	$= 3.0$	
$C \rightarrow G$	$= 2.5$	
$D \rightarrow G$	$= 1.8$	
$E \rightarrow G$	$= 2.2$	
$F \rightarrow G$	$= 1.7$	
$H \rightarrow G$	$= 2.3$	
$G \rightarrow G$	$= 0$	

# Greedy best-first search



Heuristic  $\rightarrow$  Straight Line distance

$$A \rightarrow G = 35$$

$$B \rightarrow G = 30$$

$$C \rightarrow G = 25$$

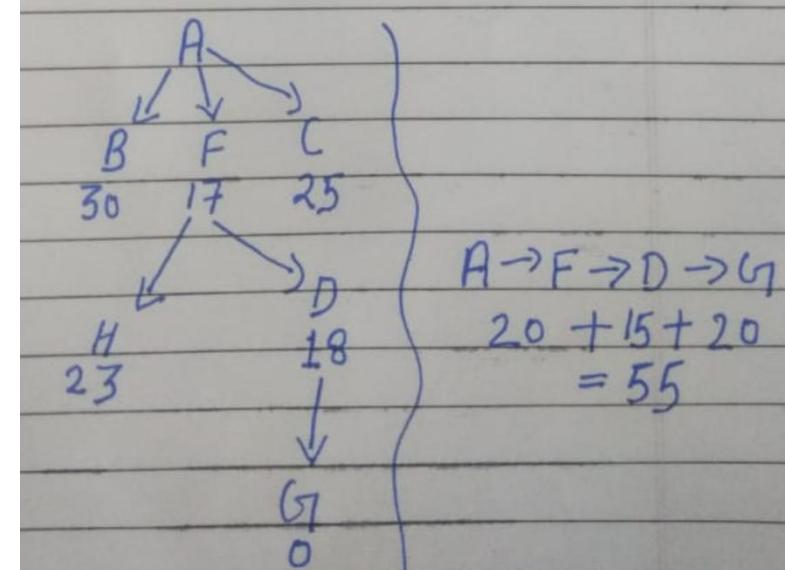
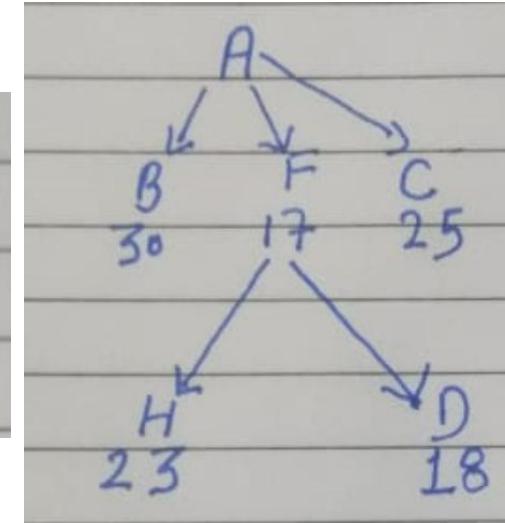
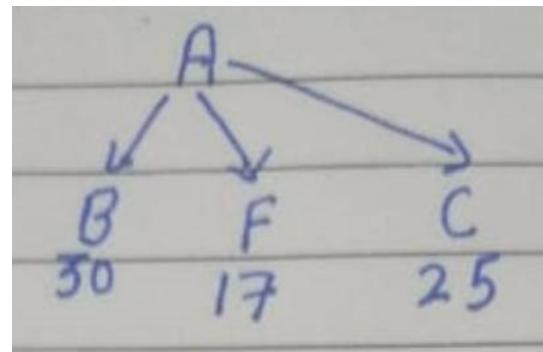
$$D \rightarrow G = 18$$

$$E \rightarrow G = 22$$

$$F \rightarrow G = 17$$

$$H \rightarrow G = 23$$

$$G \rightarrow G = 0$$



- The worst-case time and space complexity is  $O(b^m)$

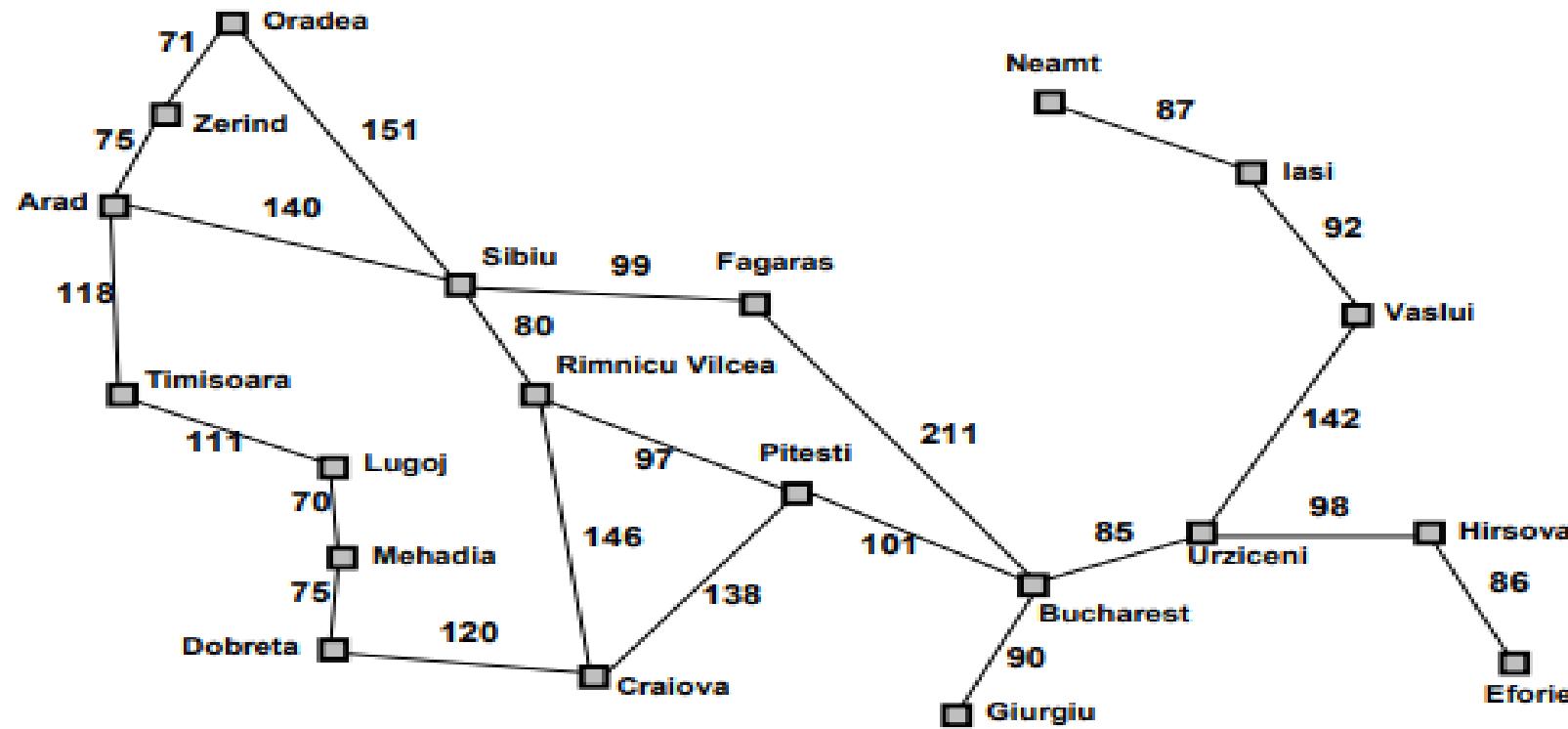
Where  $m$  is the maximum depth of the search space.

- With a good heuristic function, however, the complexity can be reduced substantially.
- The amount of the reduction depends on the particular problem and on the quality of the heuristic.

---

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

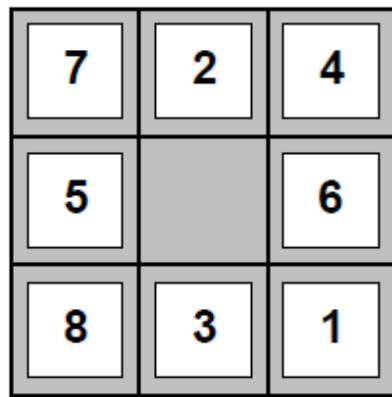
**Figure 3.16** Values of  $h_{SLD}$ —straight-line distances to Bucharest.



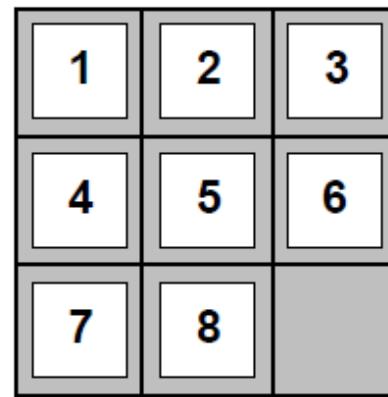
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure 3.16 Values of  $h_{SLD}$ —straight-line distances to Bucharest.

$h(n)$  = total Manhattan distance  
(i.e., no. of squares from desired location of each tile)



Start State



Goal State

# Properties of Greedy Best Search

Complete?? No—can get stuck in loops, e.g.,

lasi → Neamt → lasi → Neamt →

Complete in finite space with repeated-state checking

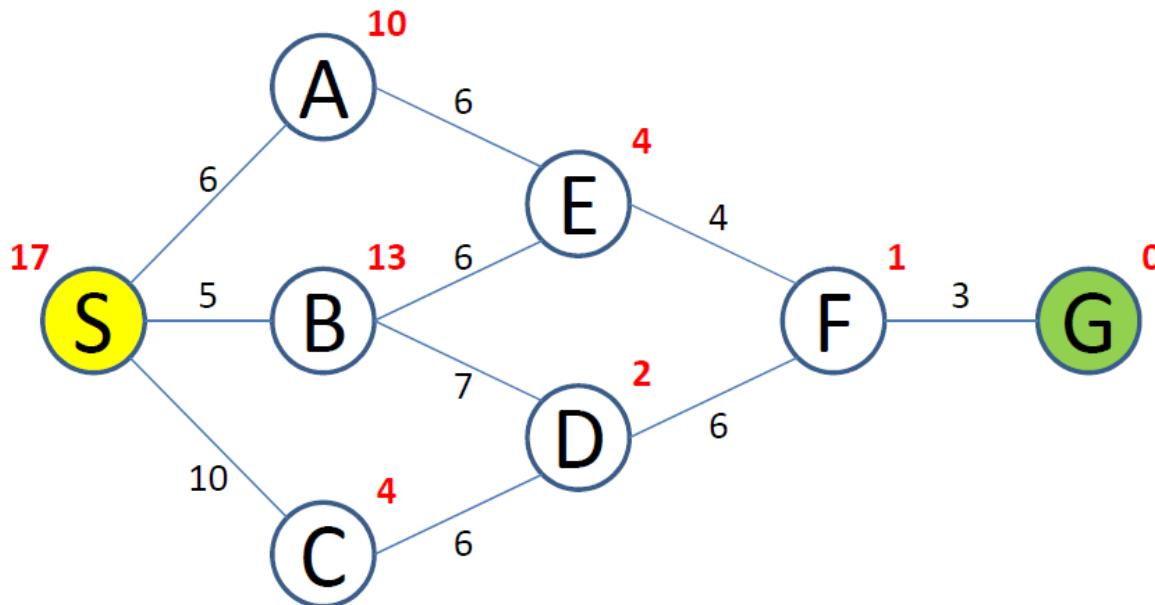
Time??  $O(b^m)$ , but a good heuristic can give dramatic improvement

Space??  $O(b^m)$ —keeps all nodes in memory

Optimal?? No

Find the optimal path from S to G using the Greedy Best First Search algorithm for the graph below. In this graph, the nodes S and G represent the source and goal states respectively.

- The cost ( $g$ ) between two adjacent nodes is given on the edge (e.g.  $g(A, E) = g(E, A) = 6$ ). The heuristic value ( $h$ ) to reach the goal state from a node is given outside the node (e.g.,  $h(S) = 17$ ).



# A\* Search

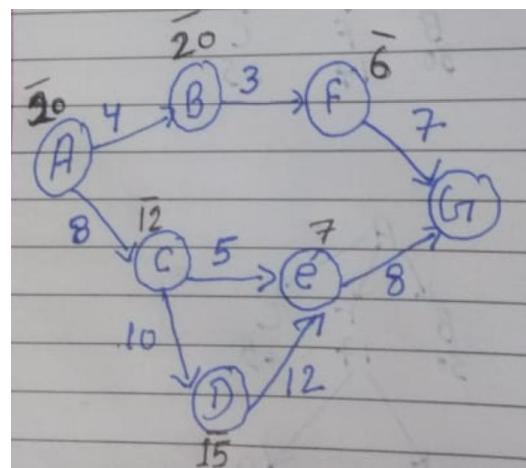
- The most common informed search algorithm is A\* search (pronounced “A-star search”), a best-first search that uses the evaluation function,

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

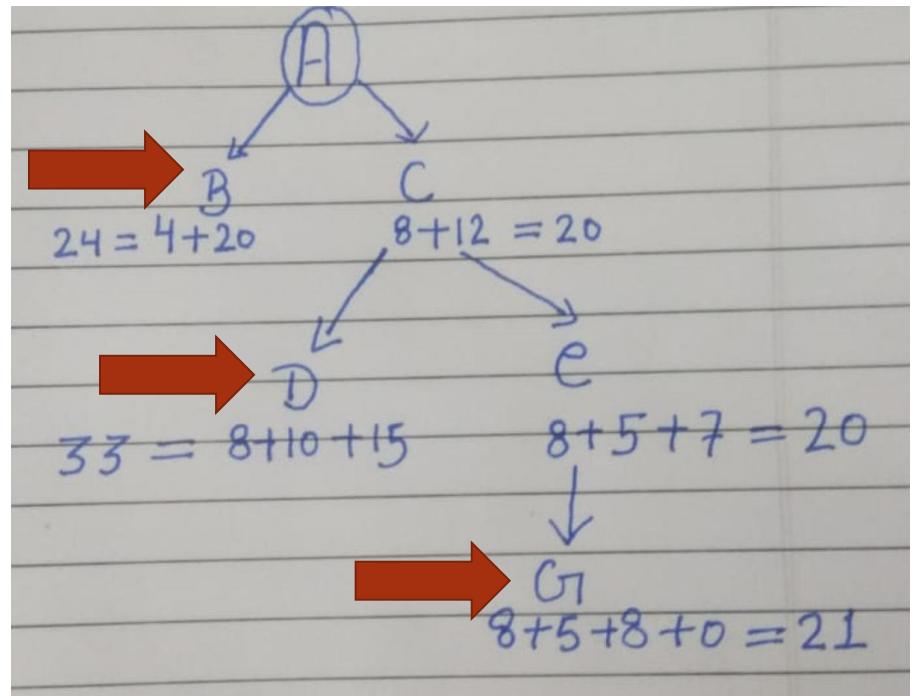
where  $g(n)$  is the path cost from the initial state to node  $n$ , and  $h(n)$  is the estimated cost of the cheapest path from  $n$  to a goal state, so we have,

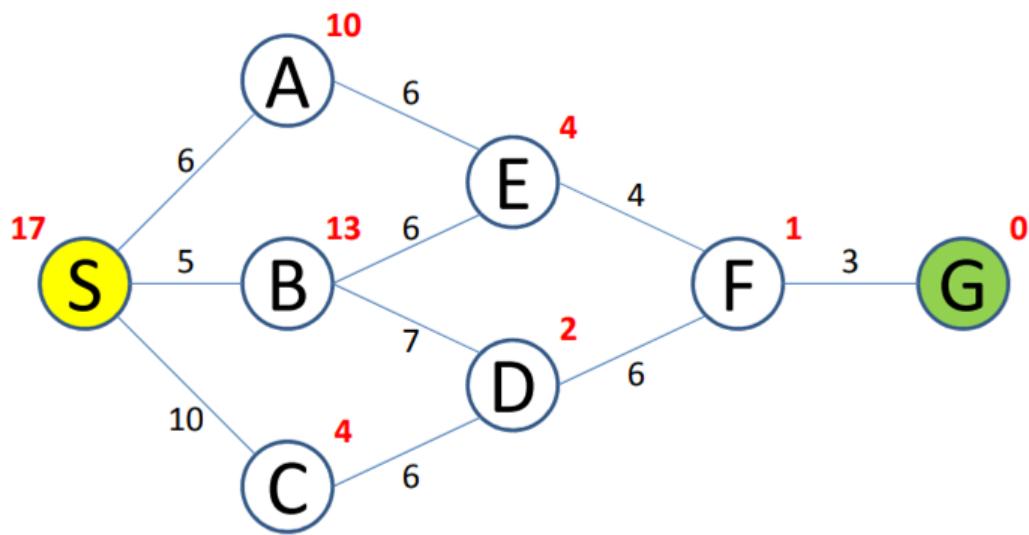
*f(n) = estimated cost of the best path that continues from n to a goal.*

## A\* search



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

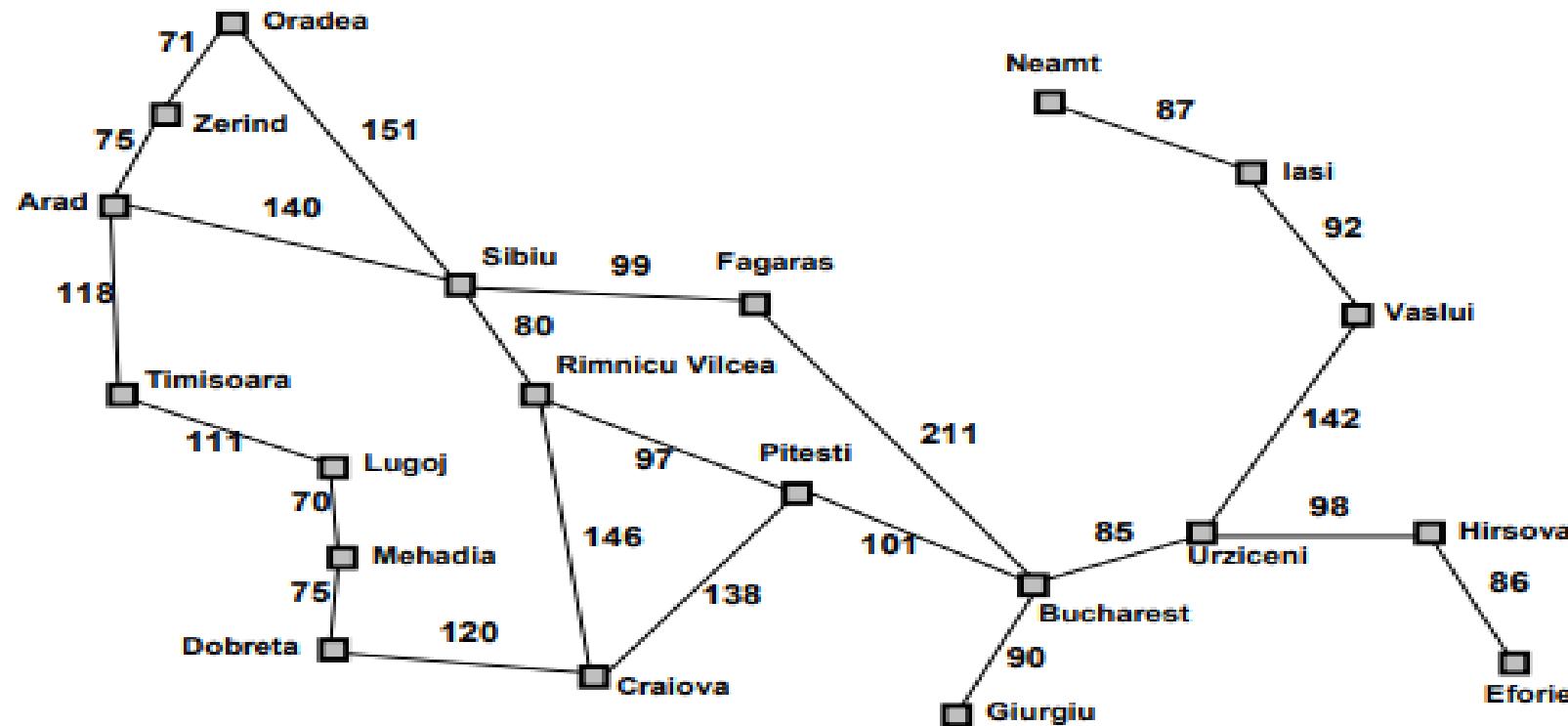




---

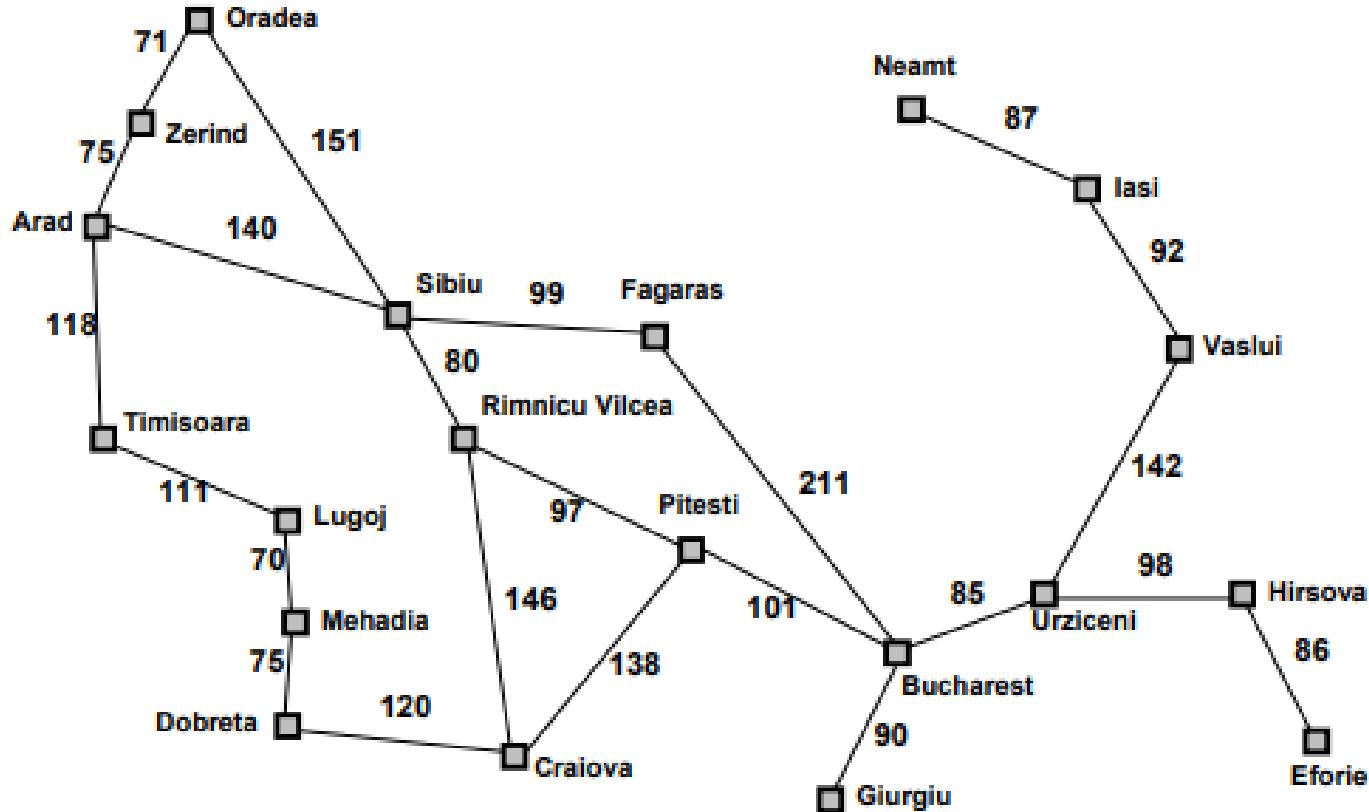
<b>Arad</b>	366	<b>Mehadia</b>	241
<b>Bucharest</b>	0	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	253
<b>Giurgiu</b>	77	<b>Timisoara</b>	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80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

**Figure 3.16** Values of  $h_{SLD}$ —straight-line distances to Bucharest.



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 3.16 Values of  $h_{SLD}$ —straight-line distances to Bucharest.



Straight-line distance  
to Bucharest

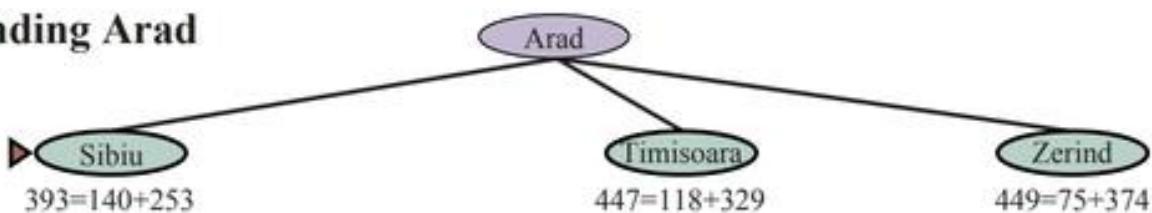
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Straight-line distance to Bucharest	
<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	178
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	98
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

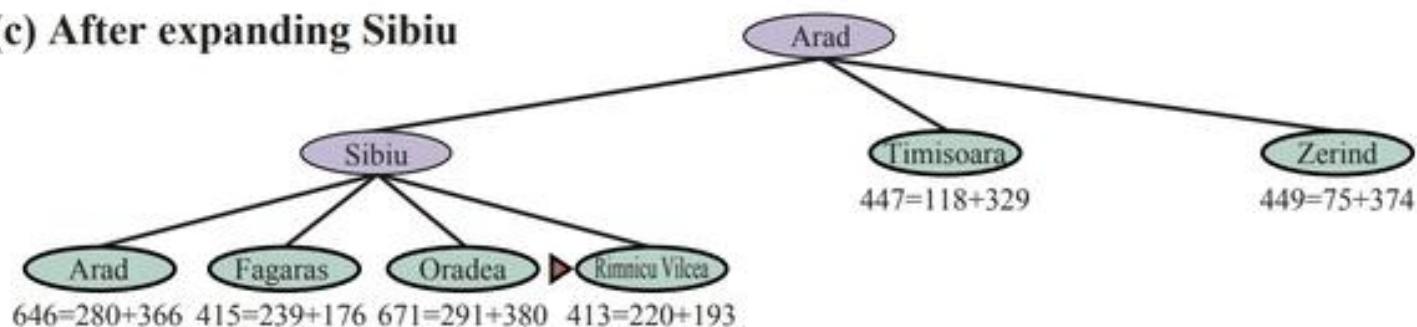
**(a) The initial state**



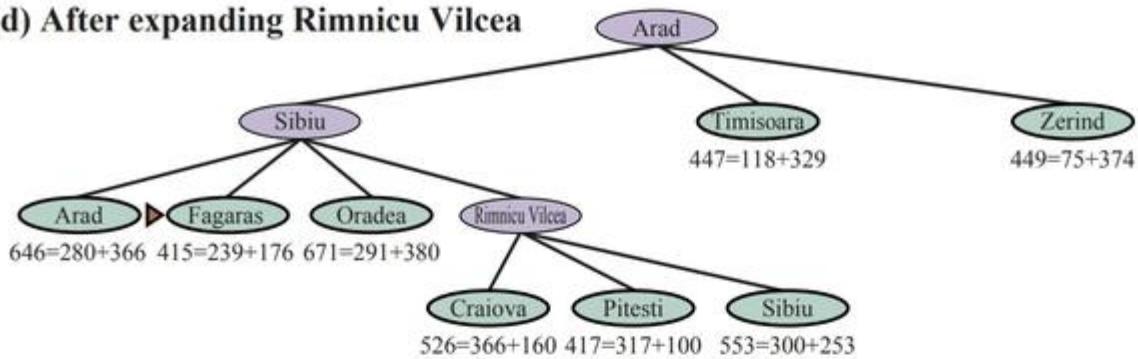
**(b) After expanding Arad**



**(c) After expanding Sibiu**



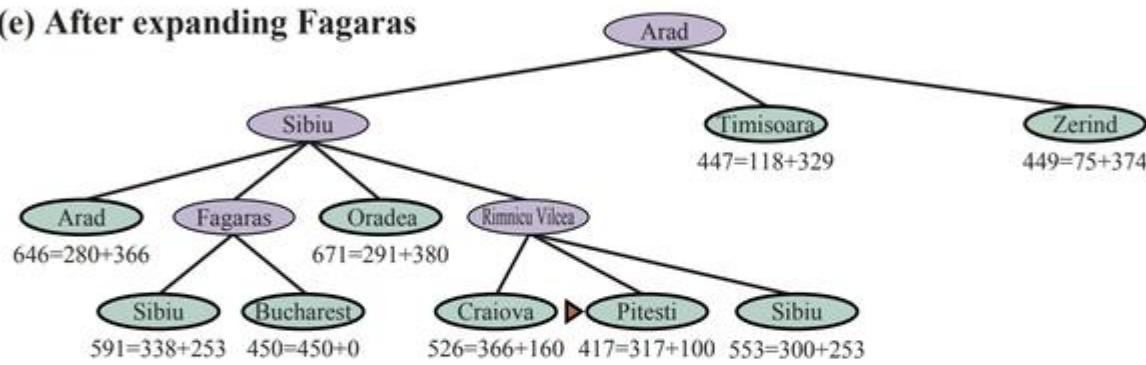
(d) After expanding Rimnicu Vilcea



Straight-line distance  
to Bucharest

<b>Arad</b>	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

(e) After expanding Fagaras



(f) After expanding Pitesti

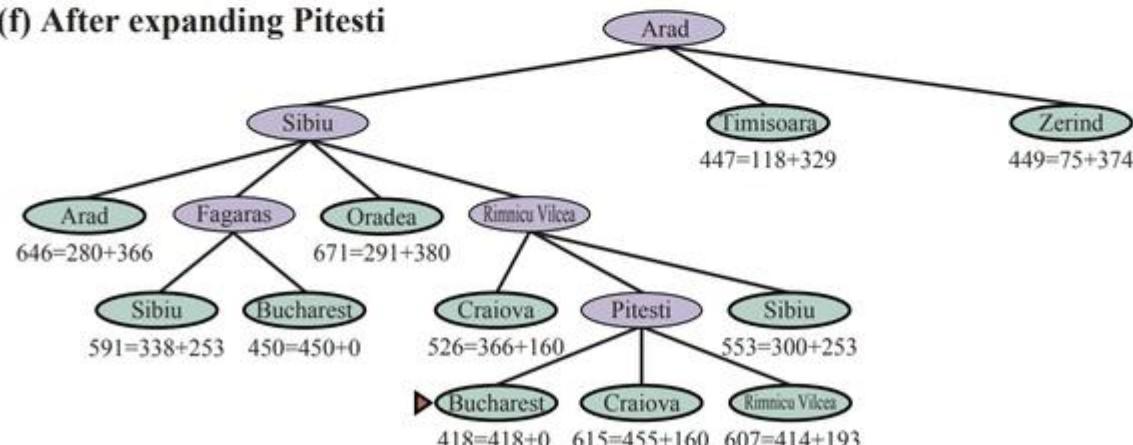
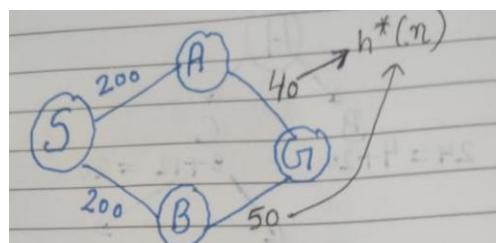


Figure 3.18 Stages in an A\* search for Bucharest. Nodes are labeled with  $f = g + h$ . The  $h$  values are the straight-line distances to Bucharest taken from Figure 3.16.

# A\* search : Admissibility

- The 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.
- Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.
- Straight-line distance is admissible because the shortest path between any two points is a straight line, so the straight line cannot be an overestimate.
- $h(n) \leq h^*(n)$  -----Underestimation
- $h(n) > h^*(n)$  -----Overestimation

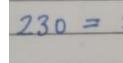
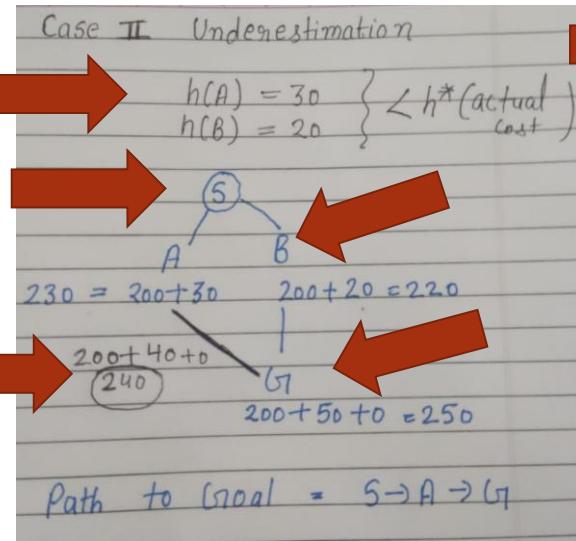
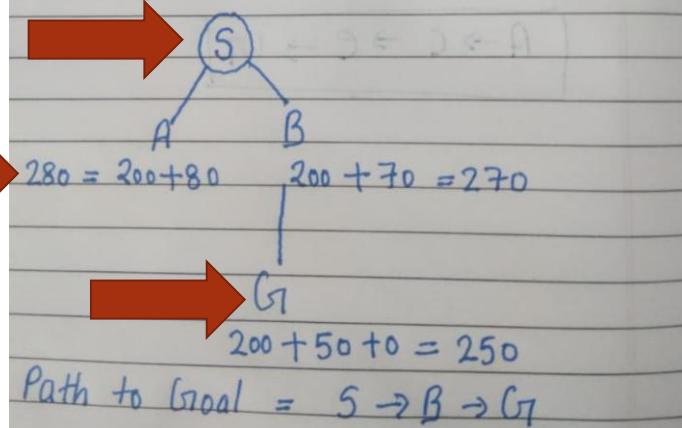
# A\* search : Admissibility

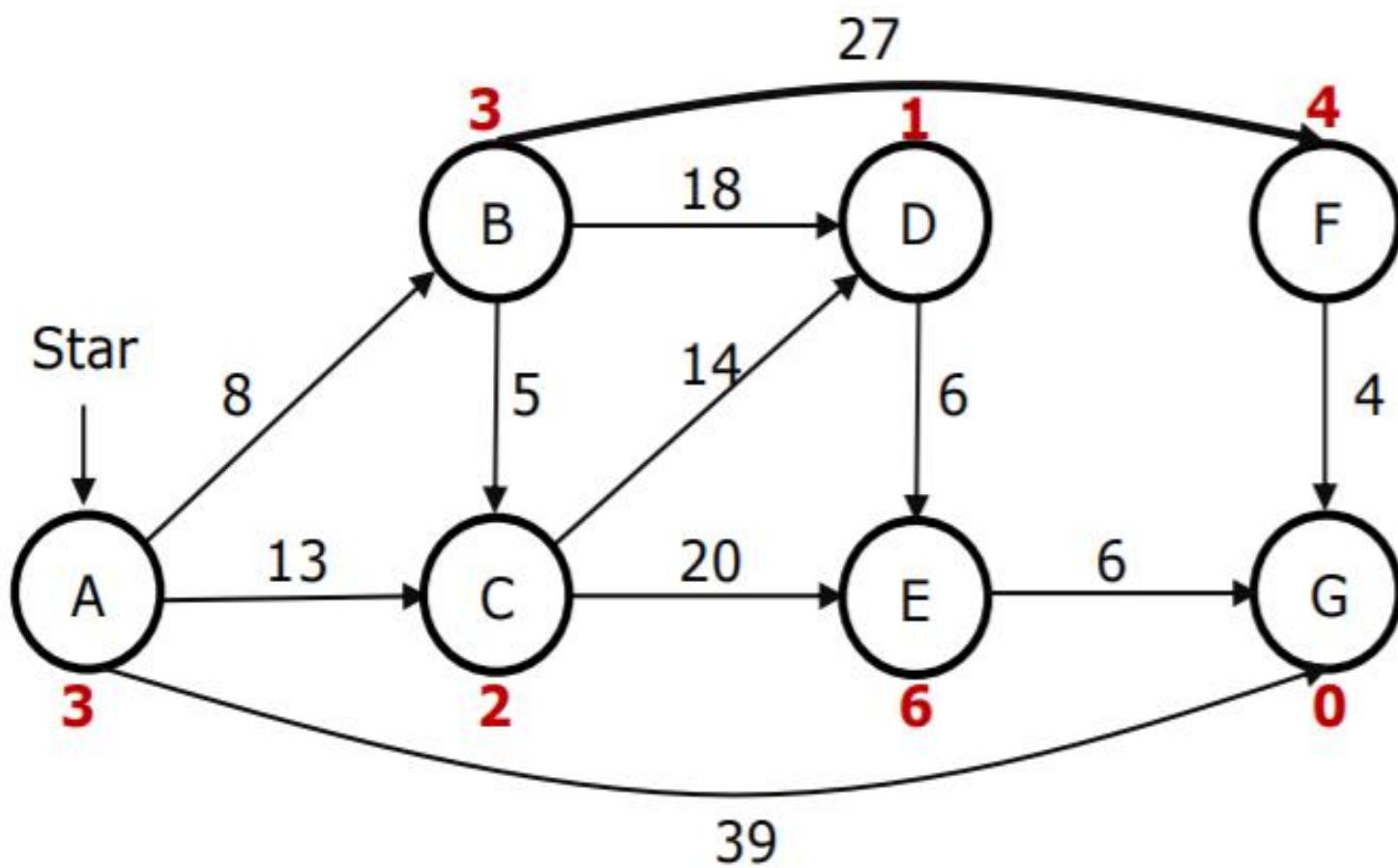


Case I Overestimation

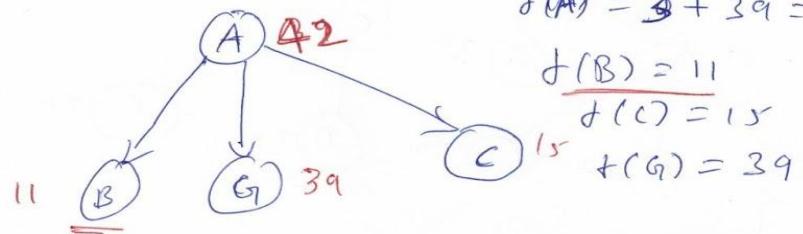
$$\left. \begin{array}{l} h(A) = 80 \\ h(B) = 70 \end{array} \right\} > h^*(actual)$$

cost

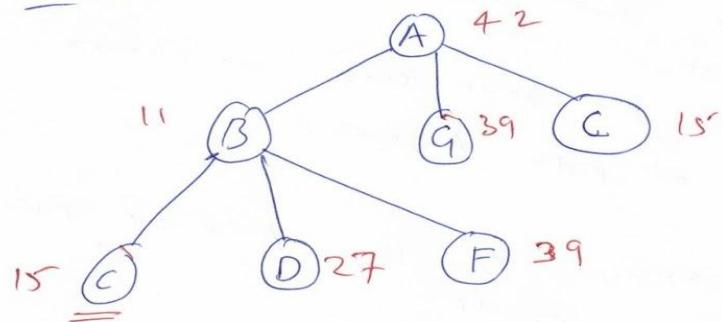




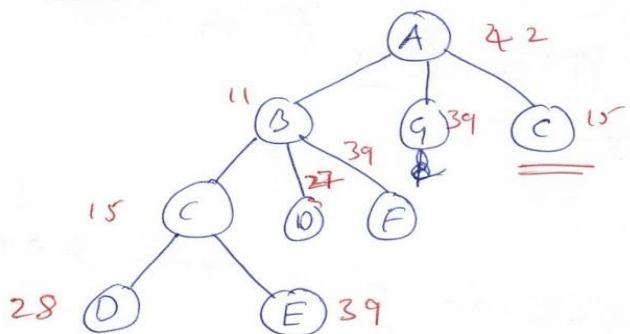
Start Step 1:- Expand 'A'  $f(n) = g(n)$



Step 2:- expand 'B'

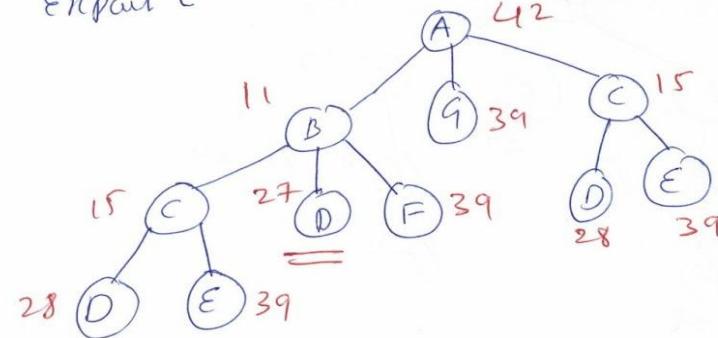


Step 3:- expand 'C'

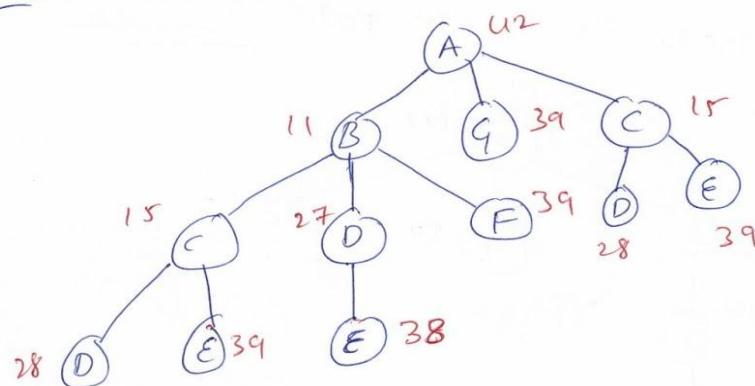


Step 4:- expand 'C' which is child of 'A'

Expect C



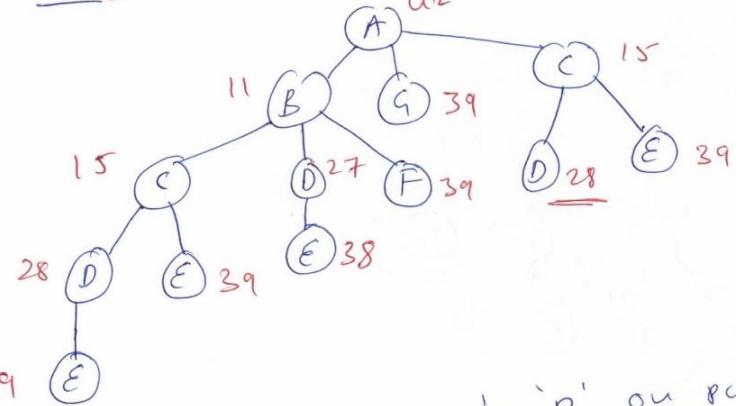
Step 5: Expand 'D' which is minimum & child of



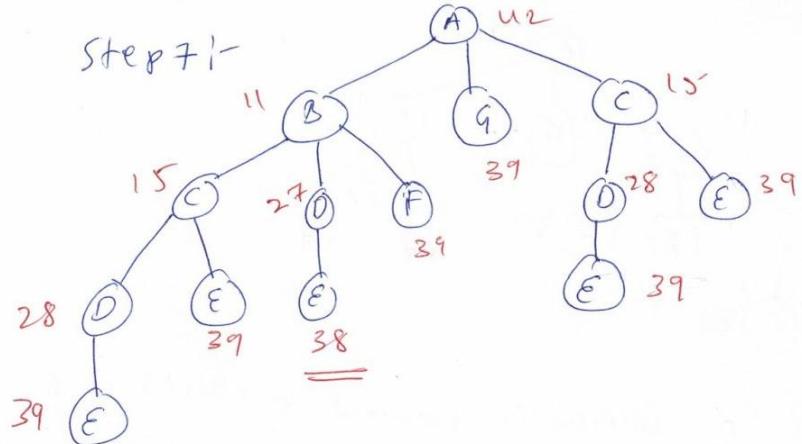
we have two same heuristic nodes will choose either of these. In our case we choose node 'D' on path  $A \rightarrow B \rightarrow C$

~~Step 6~~

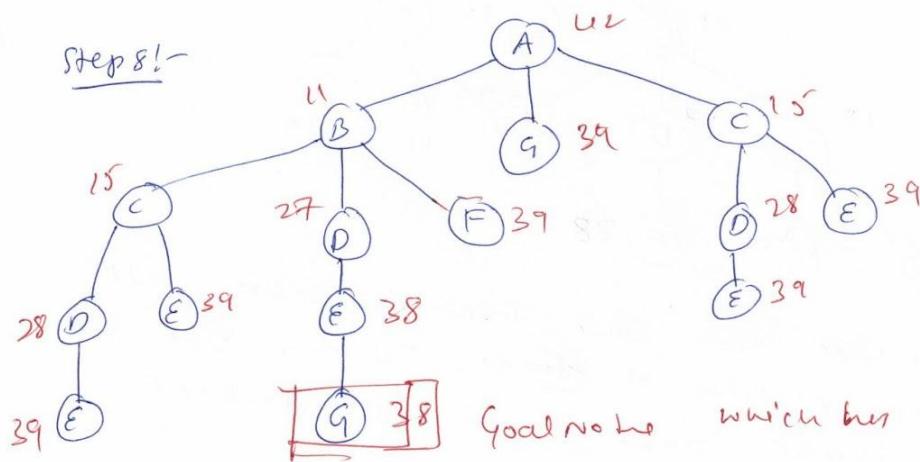
Step 6  $\rightarrow$  expand 'o' on  $A \rightarrow B \rightarrow C$  ~~each~~



Step 6: minimum node is 'o' on path A → C



minimum wage is ' $\epsilon$ ' on path  $A \xrightarrow{\underline{B}} D \xrightarrow{\epsilon} E$



optimal

Path :-

$$A \rightarrow B \rightarrow D \rightarrow E \rightarrow G$$

$$\boxed{\text{Cost} = 38}$$

# A\* search is complete.!!

- *Whether A\* is cost-optimal depends on certain properties of the heuristic.*
- A key property is admissibility an admissible heuristic is one that never overestimates the cost to reach a goal. (An admissible heuristic is therefore optimistic.)
- With an admissible heuristic, A\* is cost-optimal.
- Suppose the optimal path has cost  $C^*$ , but the algorithm returns a path with cost  $C > C^*$ . **Then there must be some node n which is on the optimal path and is unexpanded.**

- The notation  $g^*(n)$  denote the cost of the optimal path from the start to  $n$ , and  $h^*(n)$  denote the cost of the optimal path from  $n$  to the nearest goal, we have:

$$f(n) > C^* \text{ (otherwise } n \text{ would have been expanded)}$$

$$f(n) = g(n) + h(n) \text{ (by definition)}$$

$$f(n) = g^*(n) + h(n) \text{ (because } n \text{ is on an optimal path)}$$

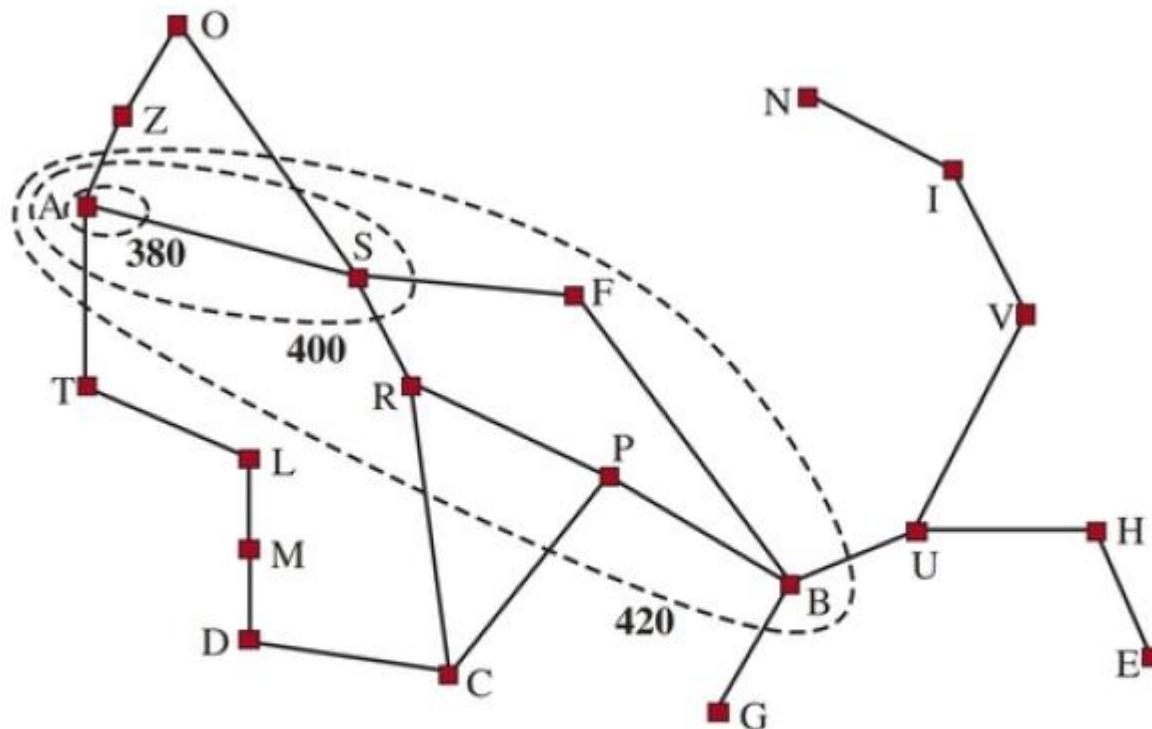
$$f(n) \leq g^*(n) + h^*(n) \text{ (because of admissibility, } h(n) \leq h^*(n))$$

$$f(n) \leq C^* \text{ (by definition, } C^* = g^*(n) + h^*(n))$$

- a suboptimal path must be wrong—it must be that A\* returns only cost-optimal paths.

# Search contours

- A useful way to visualize a search is to draw contours in the state space, just like the contours in a topographic map.



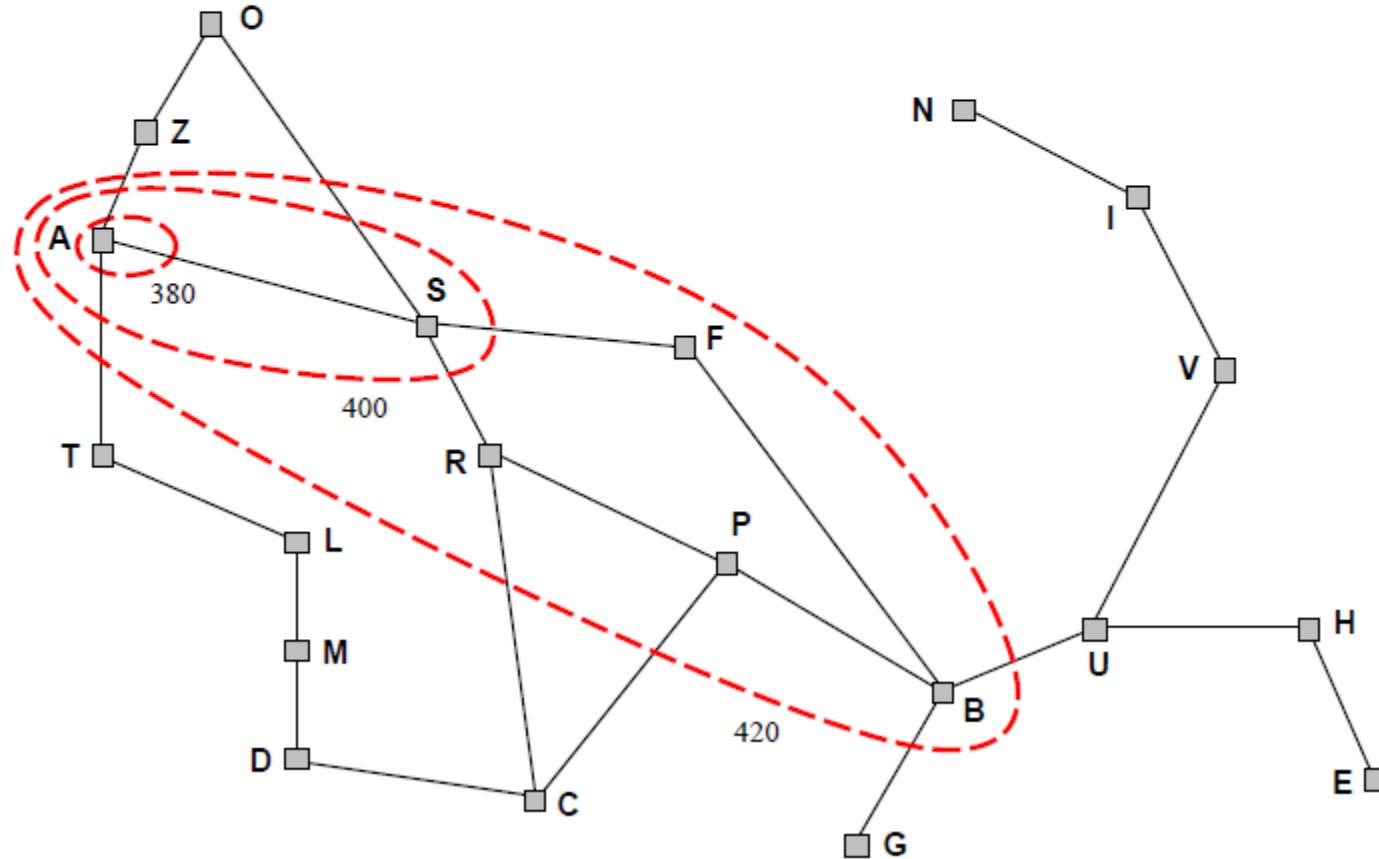
**Figure 3.20** Map of Romania showing contours at  $f = 380$ ,  $f = 400$ , and  $f = 420$ , with Arad as the start state. Nodes inside a given contour have  $f = g + h$  costs less than or equal to the contour value.

# Optimality of A\* (more useful)

Lemma: A\* expands nodes in order of increasing  $f$  value\*

Gradually adds “ $f$ -contours” of nodes (cf. breadth-first adds layers)

Contour  $i$  has all nodes with  $f = f_i$ , where  $f_i < f_{i+1}$



## Properties of A\*

Complete?? Yes, unless there are infinitely many nodes with  $f \leq f(G)$

Time?? Exponential in [relative error in  $h \times$  length of soln.]

Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand  $f_{i+1}$  until  $f_i$  is finished

A\* expands all nodes with  $f(n) < C^*$

A\* expands some nodes with  $f(n) = C^*$

A\* expands no nodes with  $f(n) > C^*$

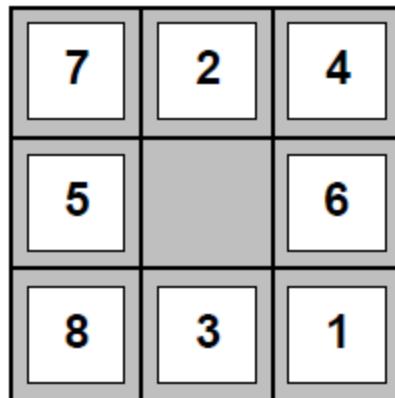
# Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$  = number of misplaced tiles

$h_2(n)$  = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State



Goal State

$$h_1(S) = ??$$

$$h_2(S) = ??$$

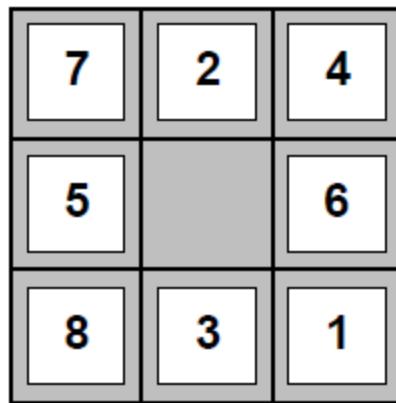
## Admissible heuristics

E.g., for the 8-puzzle:

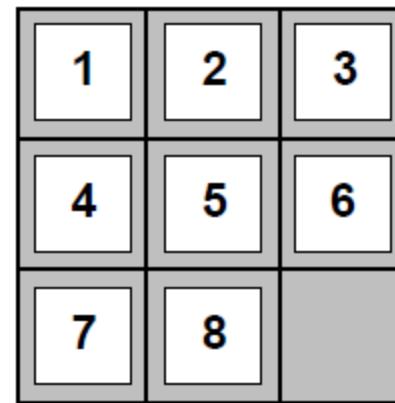
$h_1(n)$  = number of misplaced tiles

$h_2(n)$  = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State



Goal State

$$h_1(S) = ?? \quad 6$$

$$h_2(S) = ?? \quad 4+0+3+3+1+0+2+1 = 14$$

## Relaxed problems

Admissible heuristics can be derived from the **exact** solution cost of a **relaxed** version of the problem

If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then  $h_1(n)$  gives the shortest solution

If the rules are relaxed so that a tile can move to **any adjacent square**, then  $h_2(n)$  gives the shortest solution

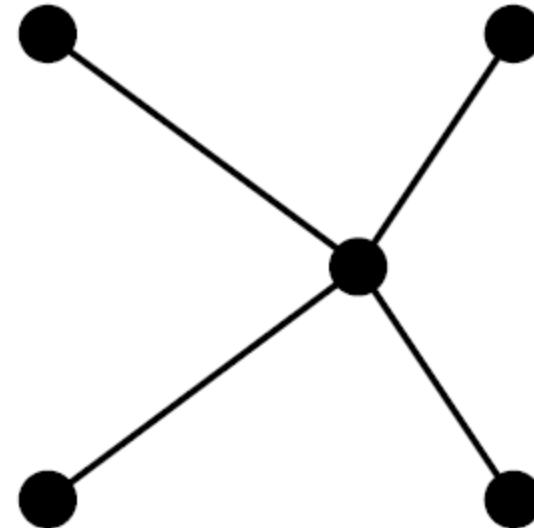
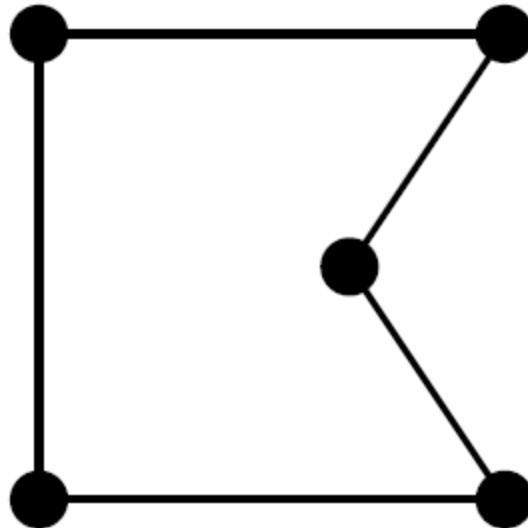
Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem

- A problem with fewer restrictions on the actions is called a relaxed problem.
- The state-space graph of the relaxed problem is a supergraph of the original state space because the removal of restrictions creates added edges in the graph.
- *Hence, the cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.*

## Relaxed problems contd.

Well-known example: travelling salesperson problem (TSP)

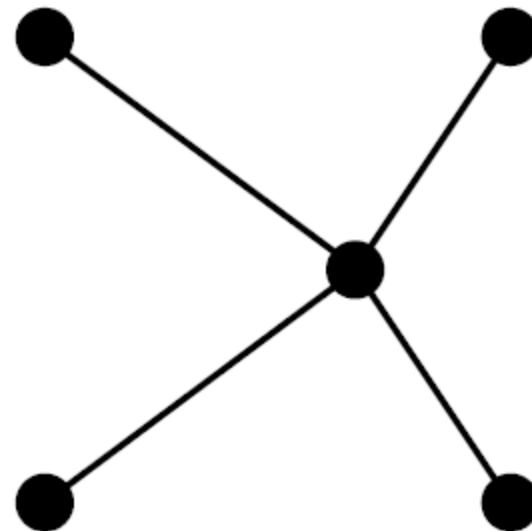
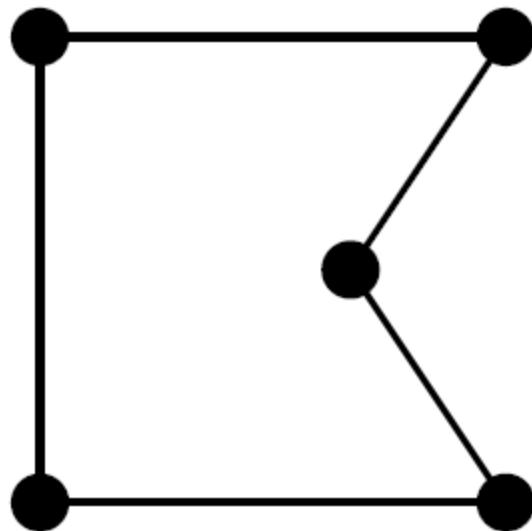
Find the shortest tour visiting all cities exactly once



## Relaxed problems contd.

Well-known example: travelling salesperson problem (TSP)

Find the shortest tour visiting all cities exactly once



Minimum spanning tree can be computed in  $O(n^2)$   
and is a lower bound on the shortest (open) tour

## Summary

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

Greedy best-first search expands lowest  $h$

- incomplete and not always optimal

$A^*$  search expands lowest  $g + h$

- complete and optimal
- also optimally efficient (up to tie-breaks, for forward search)

Admissible heuristics can be derived from exact solution of relaxed problems