



مجموعة جامعة
الجميع الذكية
للتعليم القابضة



Master of Science in Artificial intelligent

Lecture 5

AI511

Foundations of Artificial Intelligence

Prepared by
Prof. Lamiaa Fattouh Ibrahim



www.midocean.ae

Outline

Blind Strategies

- Depth-First Strategy
 - Depth-Limited Strategy
 - Iterative – Deepening Search
- breadth first search
- Uniform-Cost Strategy

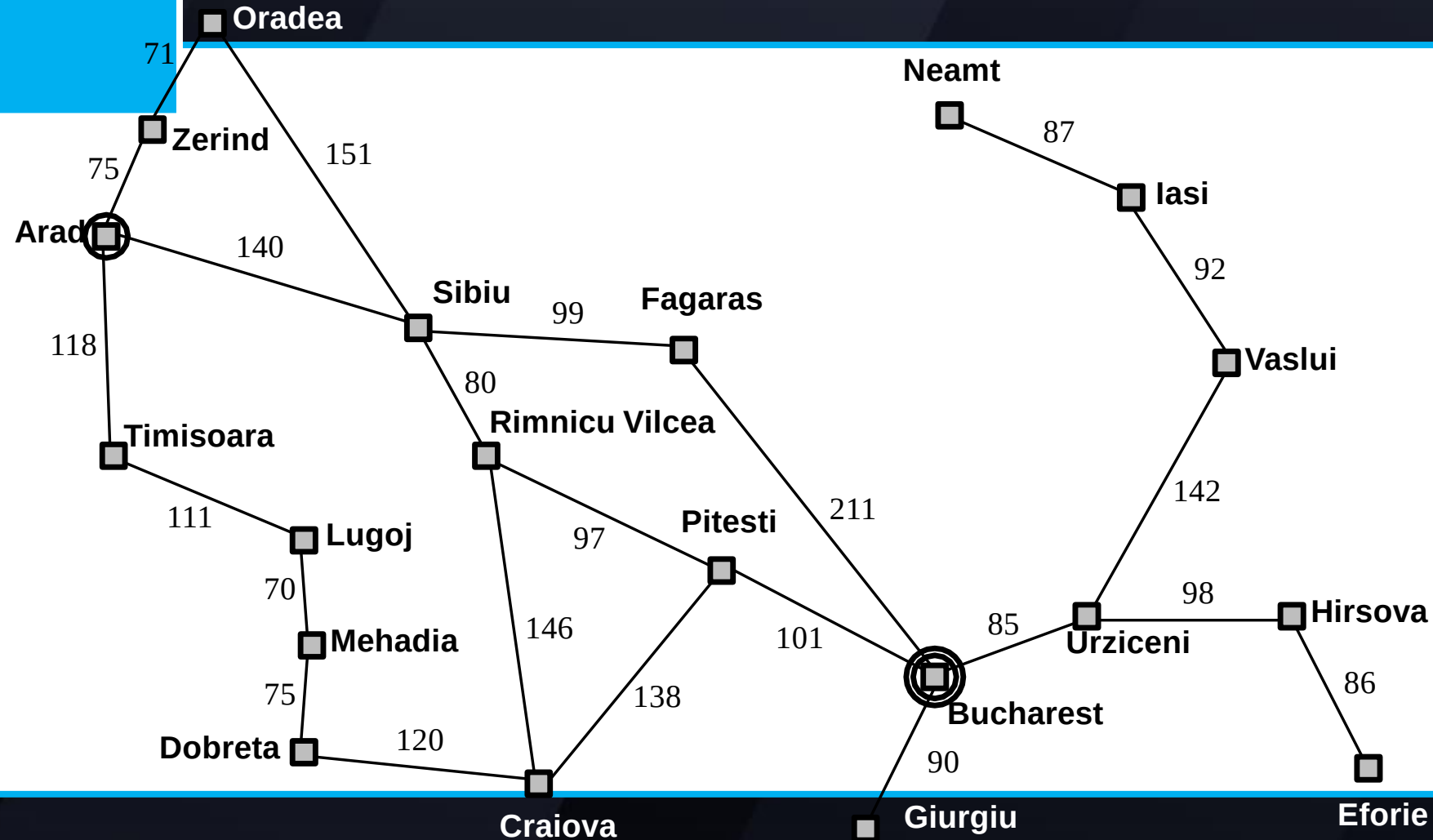


Outline

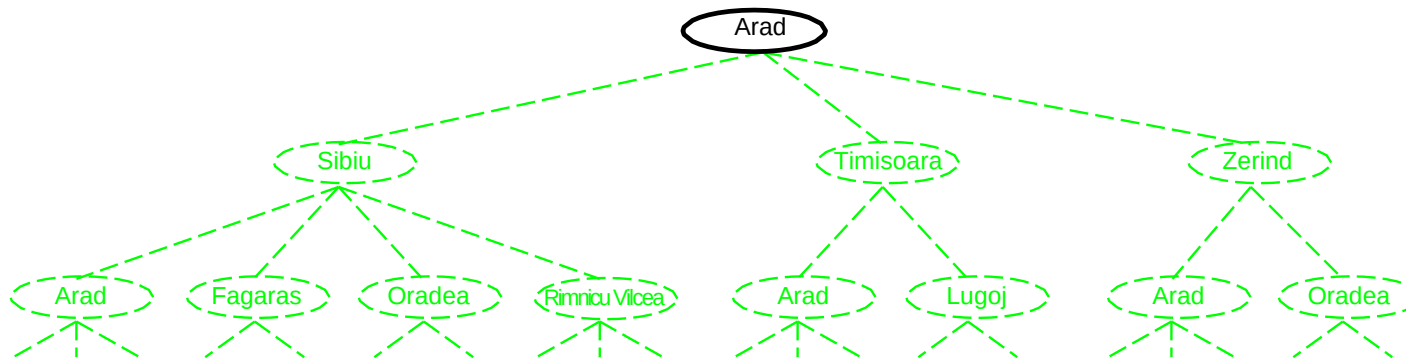
- Heuristic Function
- More on Heuristic Search & Functions
 - Best-First Search



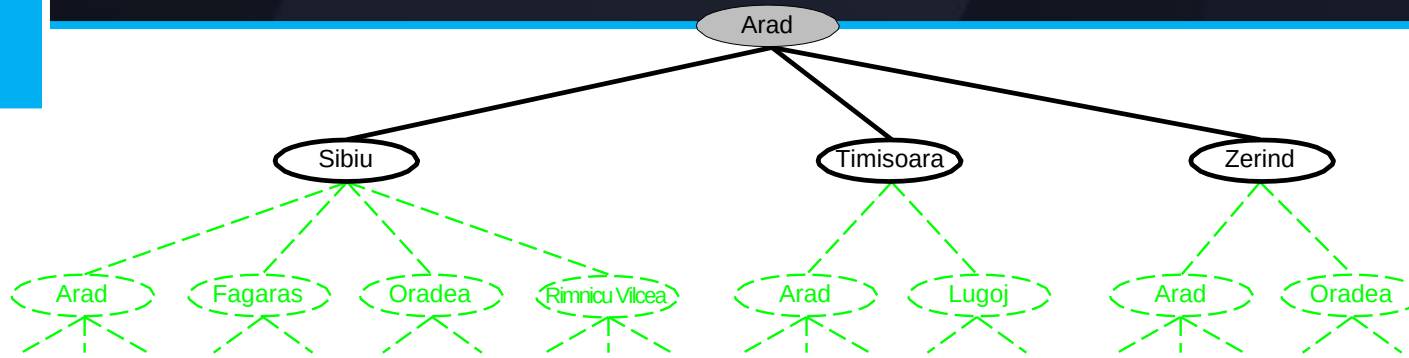
Example: Romania



Tree search example



Tree search example



Tree search example

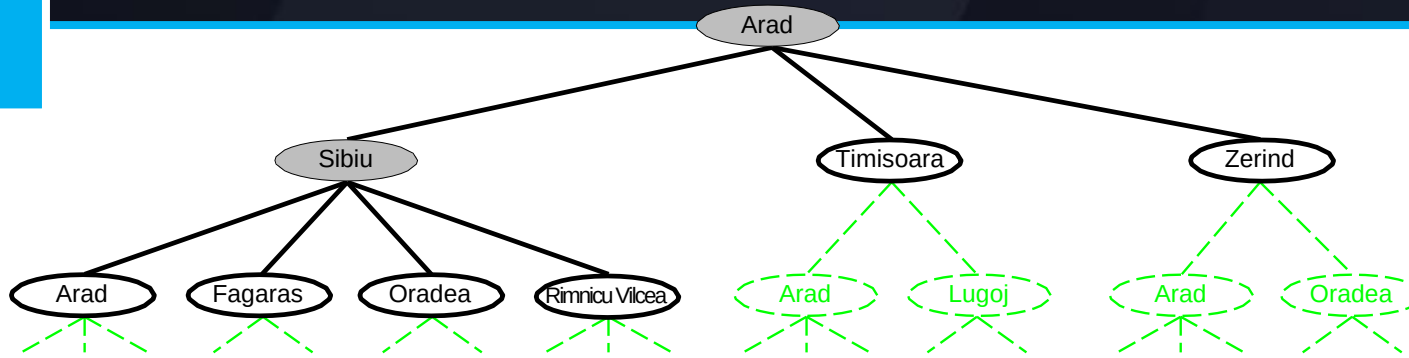
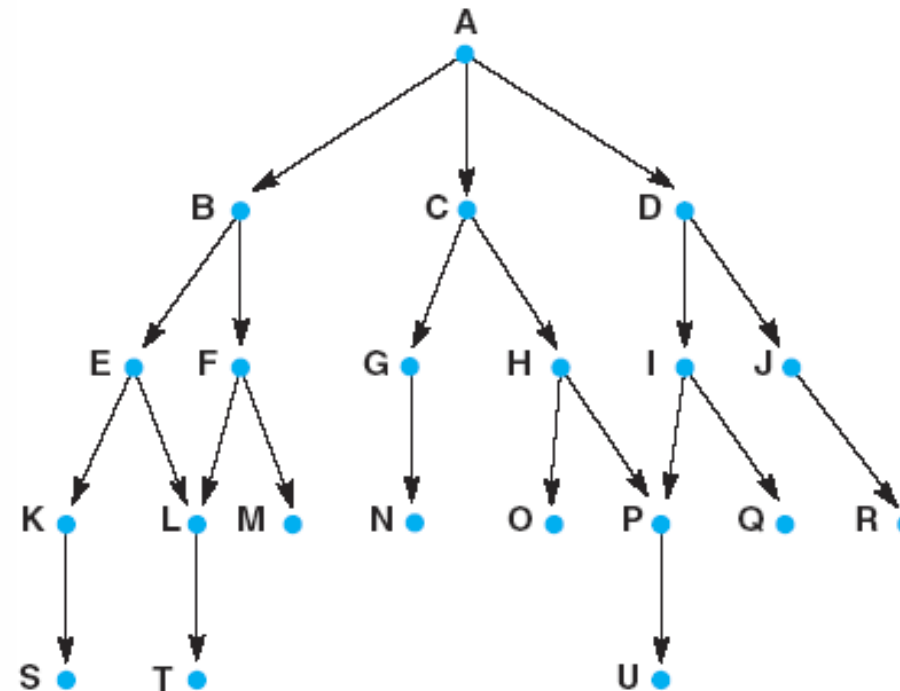


Fig 3.15 Graph for breadth - and depth - first search examples.

A **breadth first search** of the graph considers the states in the order:
A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U.



Function Breadth-First search algorithm

```
function breadth_first_search;

begin
  open := [Start];                                % initialize
  closed := [];
  while open ≠ [] do                               % states remain
    begin
      remove leftmost state from open, call it X;
      if X is a goal then return SUCCESS           % goal found
      else begin
        generate children of X;
        put X on closed;
        discard children of X if already on open or closed;
        put remaining children on right end of open % loop check
                                                    % queue
      end
    end
  end
  return FAIL                                     % no states left
end.
```



Notes:

- Child states are generated by any transition operator (such as legal moves of a game).
- Each iteration produce all children of state (X) and adds them to (open).
- States (children) are added to the right of the list and removed from the left.
- Examined states, removed from (open), are added to (closed) from the left.
- The order in which states are removed from (open) and added to (closed) determines the order of the search.
- It is necessary to detect multiple occurrences of any state so that it will not be reentered to **avoid loops**. This can be done by testing each newly generated state for membership in any of the two lists (open) and (closed). If a new state belongs to any of these lists, then it has already been visited and may be ignored
- The algorithm terminates either because the goal state is found or the list (open) becomes empty (open=[]).



- Child states are generated by inference rules, legal moves of a game, or other state transition operators.
- **Open** is maintained as a queue, first-in-first-out (FIFO) data structure.
- **Open** records the states on the “frontier” of the search at any state and **closed** records states already visited.
- **Breadth-first** search is guaranteed to find the shortest path from the start state to the goal because search considers every node at each level of the graph before going deeper into space.



A trace of breadth_first_search on the graph of Figure 3.13

1. **open** = [A]; **closed** = []
2. **open** = [B,C,D]; **closed** = [A]
3. **open** = [C,D,E,F]; **closed** = [B,A]
4. **open** = [D,E,F,G,H]; **closed** = [C,B,A]
5. **open** = [E,F,G,H,I,J]; **closed** = [D,C,B,A]
6. **open** = [F,G,H,I,J,K,L]; **closed** = [E,D,C,B,A]
7. **open** = [G,H,I,J,K,L,M] (as L is already on open); **closed** = [F,E,D,C,B,A]
8. **open** = [H,I,J,K,L,M,N]; **closed** = [G,F,E,D,C,B,A]
9. and so on until either U is found or **open** = []

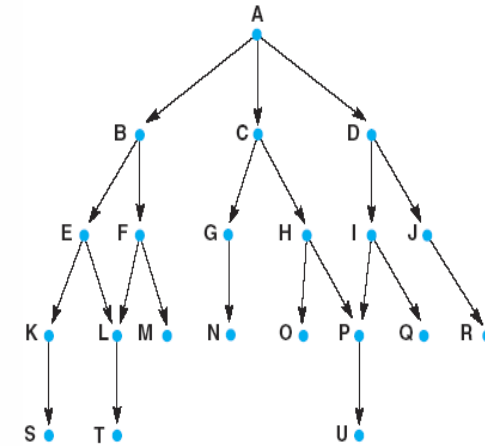
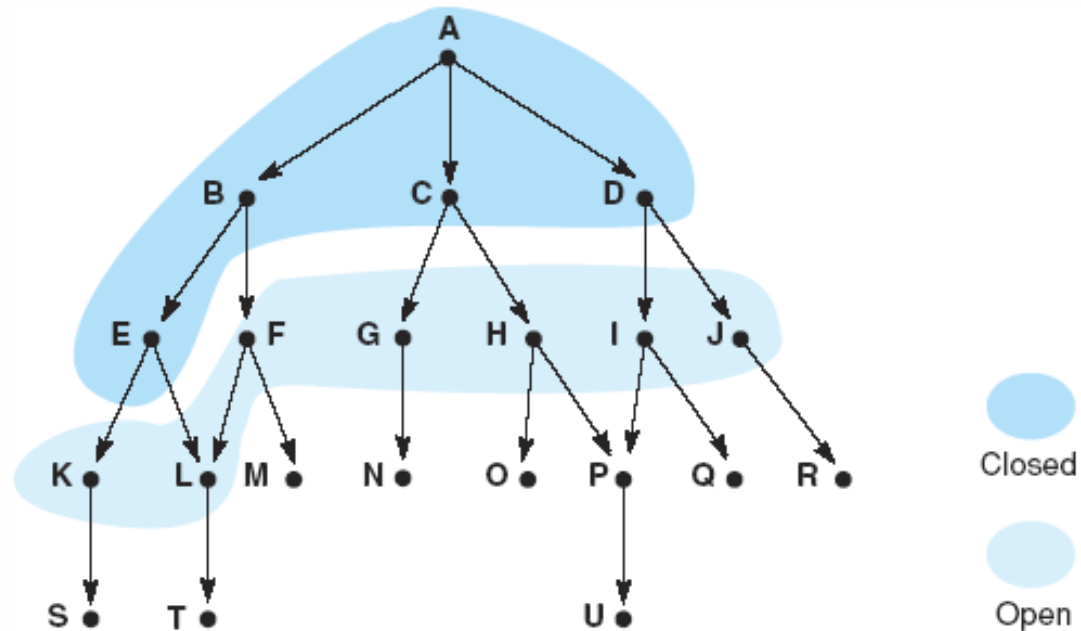


Fig 3.16 Graph of Fig 3.15 at iteration 6 of breadth-first search. States on open and closed are highlighted.

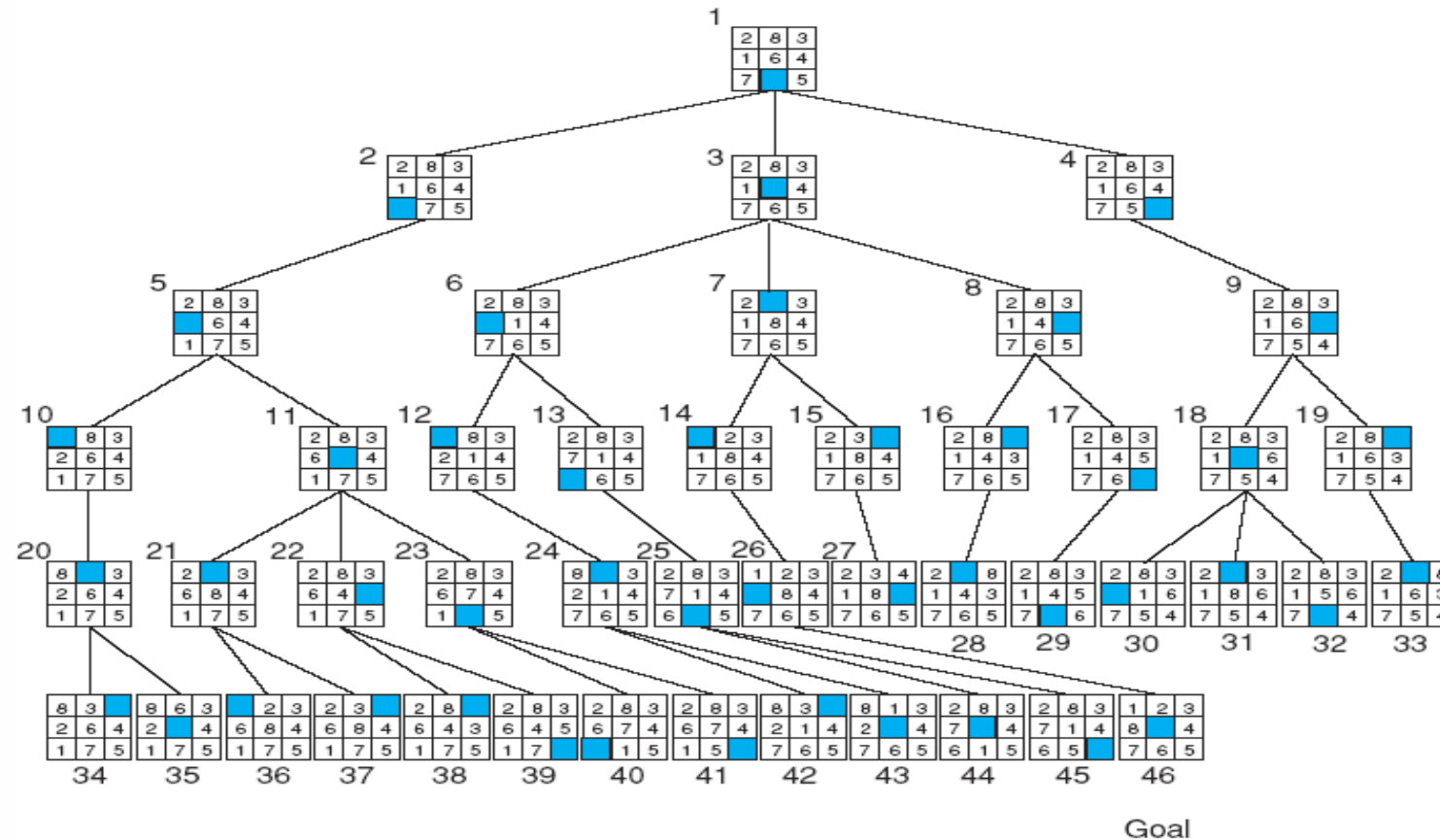
open = [(D,A), (E,B), (F,B), (G,C), (H,C)]; closed = [(C,A), (B,A), (A,nil)]



6. **open = [F,G,H,I,J,K,L]; closed = [E,D,C,B,A]**

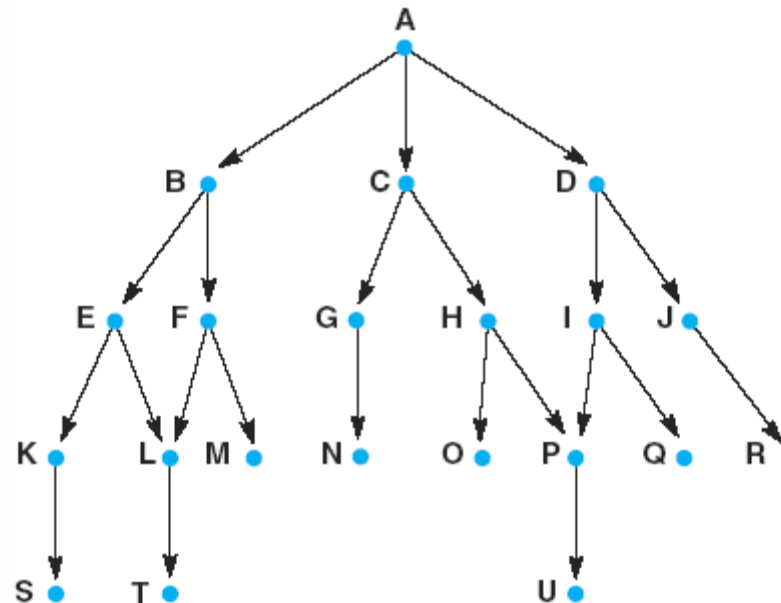


Fig 3.17 Breadth-first search of the 8-puzzle, showing order in which states were removed from open.



Graph for depth-first search examples

A Depth first search of the graph considers the states in the order:
A,B,E,K,S,L,T,F,M,C,G,N,H,O,P,U,D,I,Q,J,R



Function Depth-First-Search algorithm

```

begin
    open := [Start];                                % initialize
    closed := [ ];
    while open ≠ [ ] do                             % states remain
        begin
            remove leftmost state from open, call it X;
            if X is a goal then return SUCCESS        % goal found
            else begin
                generate children of X;
                put X on closed;
                discard children of X if already on open or closed;
                put remaining children on left end of open
            end
        end
    end;
    return FAIL
end.

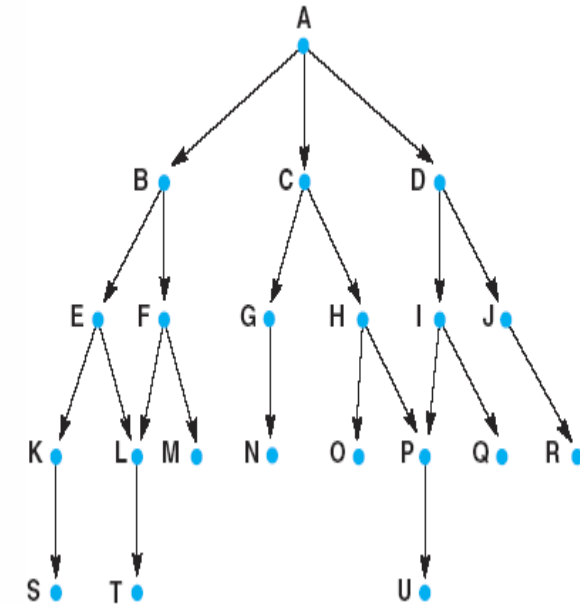
```

% loop check
% stack
% no states left



A trace of Depth-First-Search on the graph of Figure 3.13

1. **open = [A]; closed = []**
2. **open = [B,C,D]; closed = [A]**
3. **open = [E,F,C,D]; closed = [B,A]**
4. **open = [K,L,F,C,D]; closed = [E,B,A]**
5. **open = [S,L,F,C,D]; closed = [K,E,B,A]**
6. **open = [L,F,C,D]; closed = [S,K,E,B,A]**
7. **open = [T,F,C,D]; closed = [L,S,K,E,B,A]**
8. **open = [F,C,D]; closed = [T,L,S,K,E,B,A]**
9. **open = [M,C,D]**, as L is already on **closed**; **closed = [F,T,L,S,K,E,B,A]**
10. **open = [C,D]; closed = [M,F,T,L,S,K,E,B,A]**
11. **open = [G,H,D]; closed = [C,M,F,T,L,S,K,E,B,A]**



In depth-first search algorithm, the descendant states are added and removed from the left end of open.

Open is maintained as a **stack**, last-in-first-out LIFO structure.

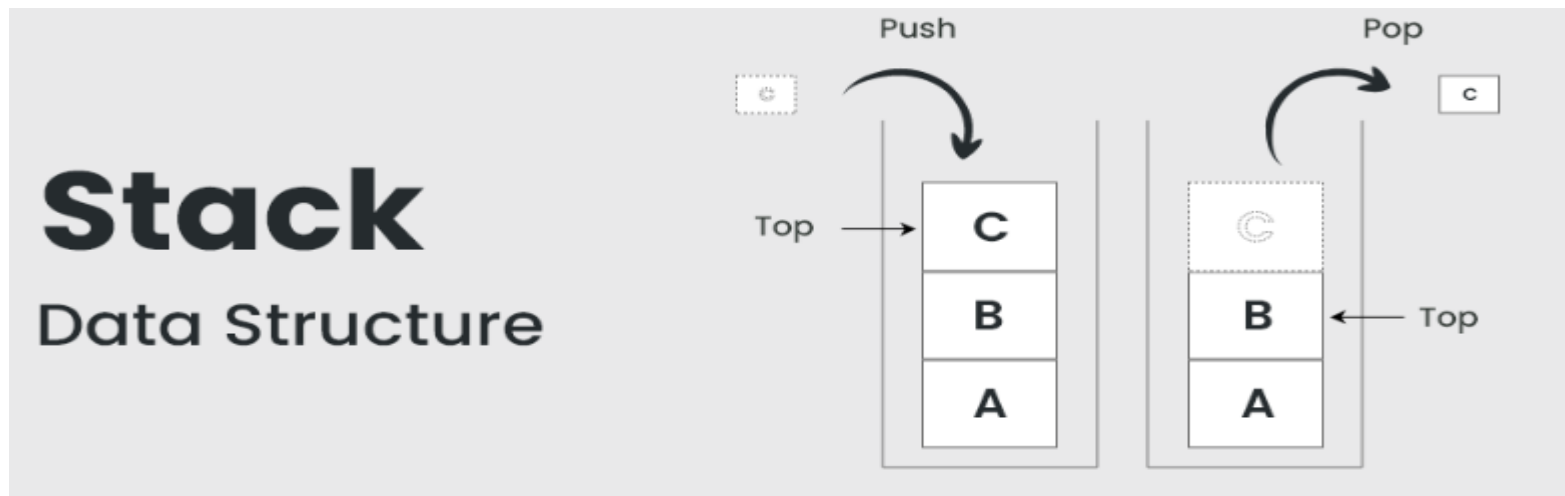


Fig 3.18 Graph of fig 3.15 at iteration 6 of depth-first search. States on open and closed are highlighted.

Open=[L, F, C, D];
Closed = [S, K, E, B, A]

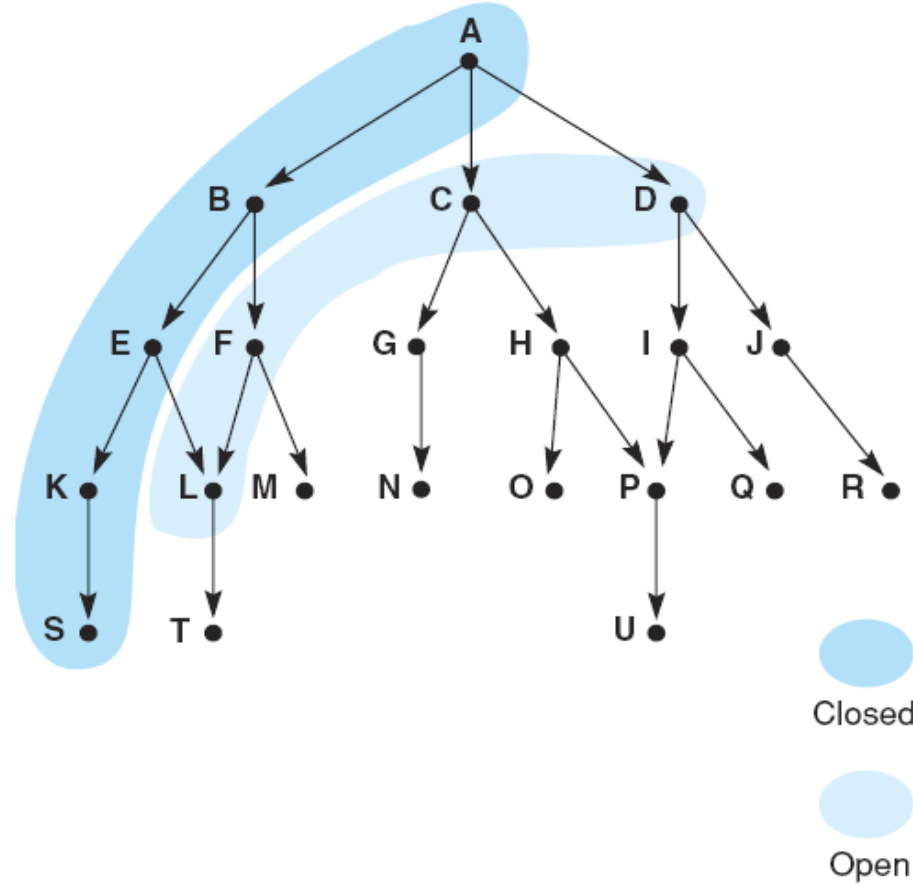
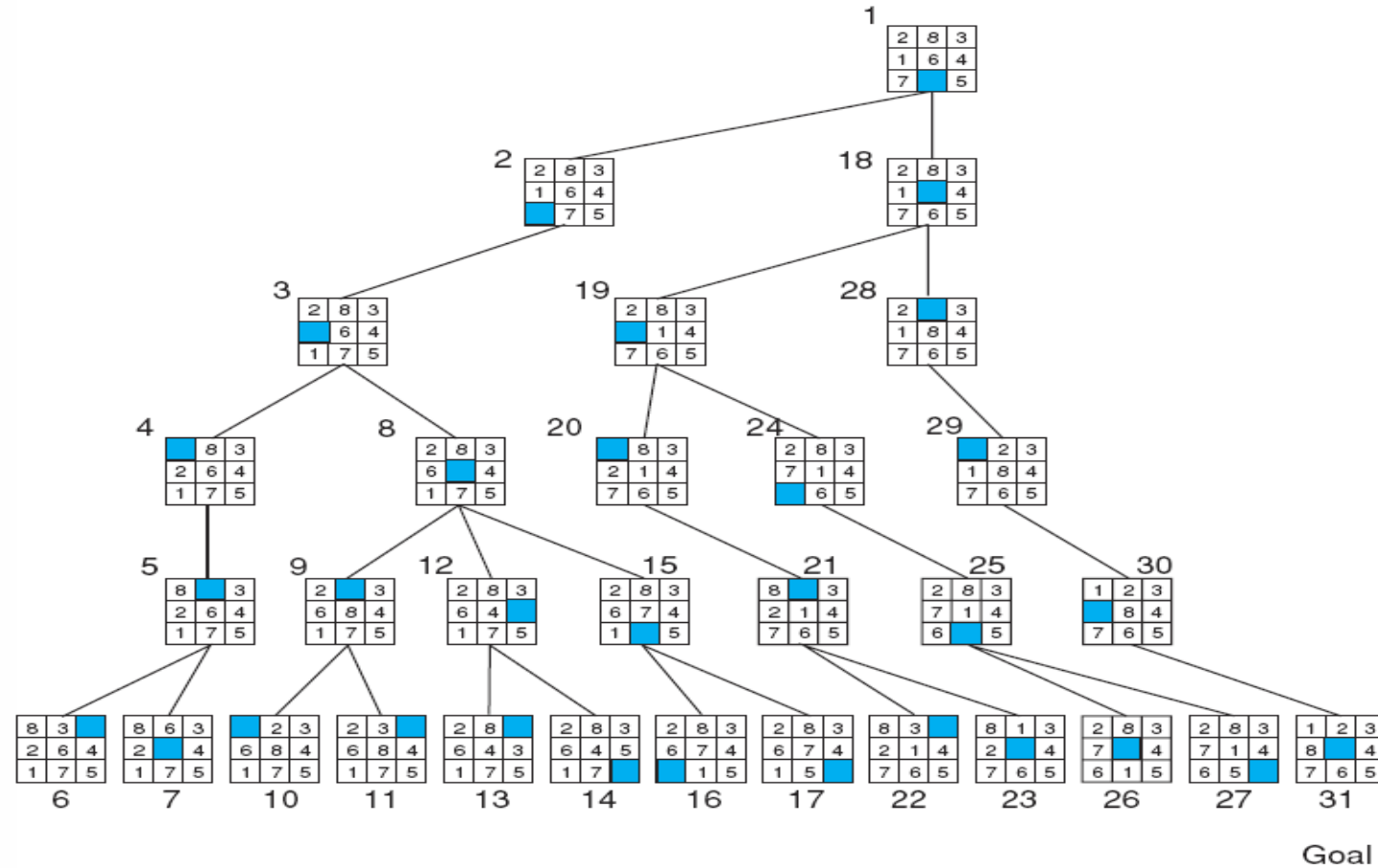
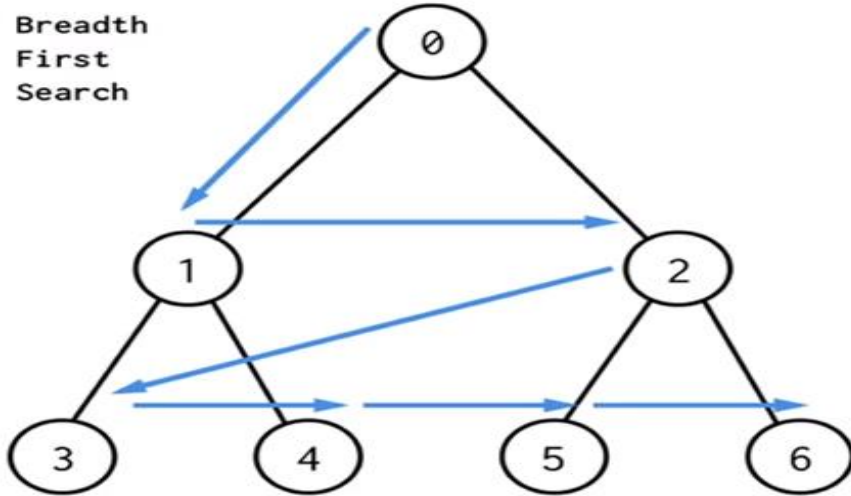


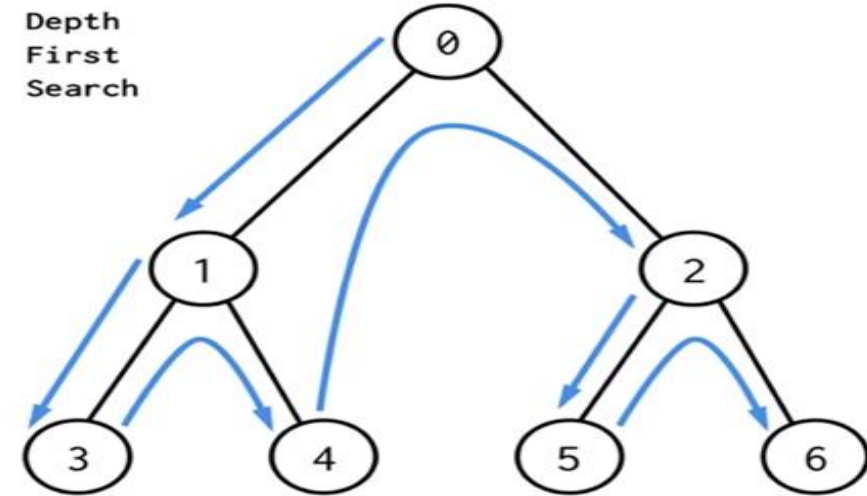
Fig 3.19 Depth-first search of the 8-puzzle with a depth bound of 5.



Breadth
First
Search



Depth
First
Search



Depth-First Strategy *versus* Breadth-First Strategy

In Depth-First Search, the FRINGE (*frontier*) is implemented using a ____ (*data-structure*) ?

While in Breadth-First Search, the FRINGE (*frontier*) is implemented using a ____ (*data-structure*) ?



Depth- Limited Search

- The embarrassing failure of depth-first search in infinite state spaces can be alleviated by supplying depth-first search with a predetermined depth limit ℓ .
- That is, nodes at depth ℓ are treated as if they have no successors.
- The depth limit solves the infinite-path problem.
- Depth-first search can be viewed as a special case of depth-limited search with $\ell = \infty$.



Depth- Limited Search

Depth-first with depth cutoff k (maximal depth below which nodes are not expanded)

Notice that depth-limited search can terminate with two kinds of failure:

- the standard failure value indicates no solution.
- the cutoff value indicates no solution within the depth limit.



Iterative – Deepening Search

Repeat for $k = 0, 1, 2, \dots$:

Perform depth-first with depth cutoff k



Iterative – Deepening Search

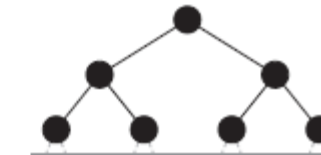
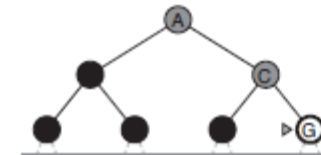
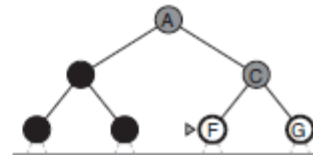
Limit = 0



Limit = 1

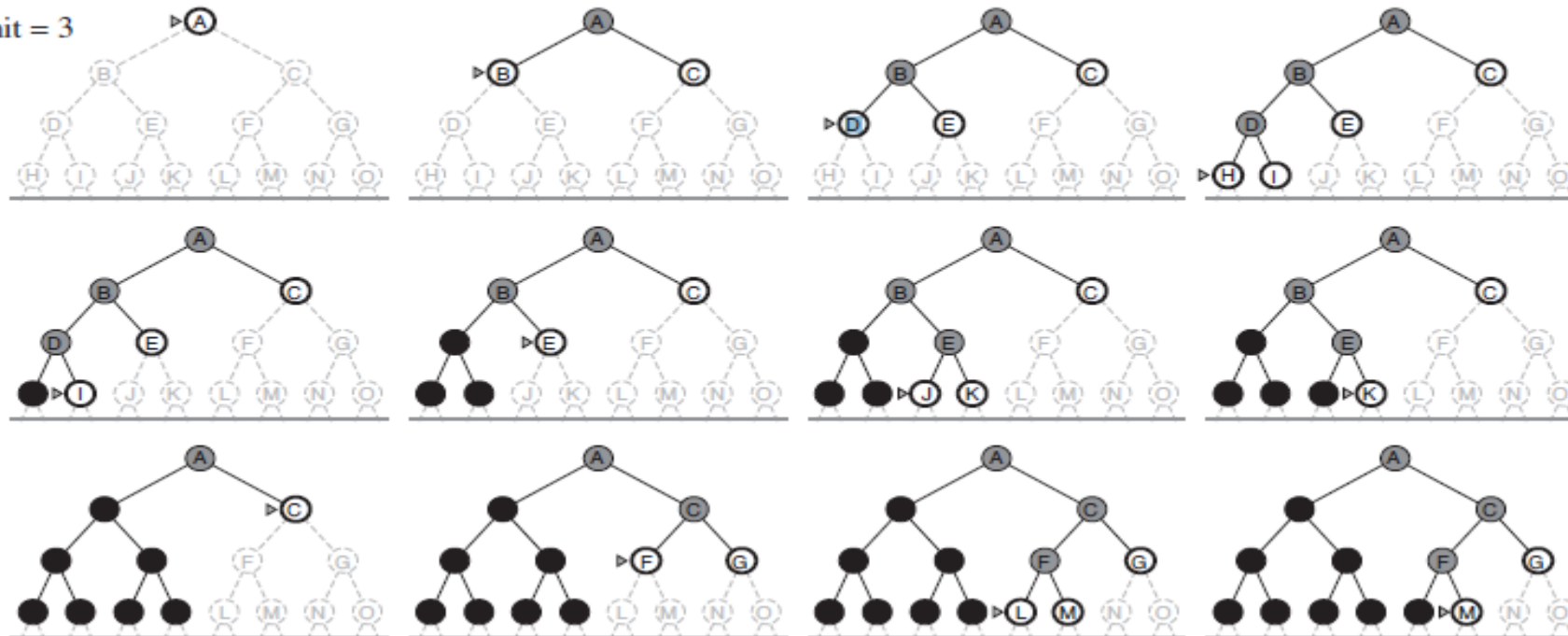


Limit = 2



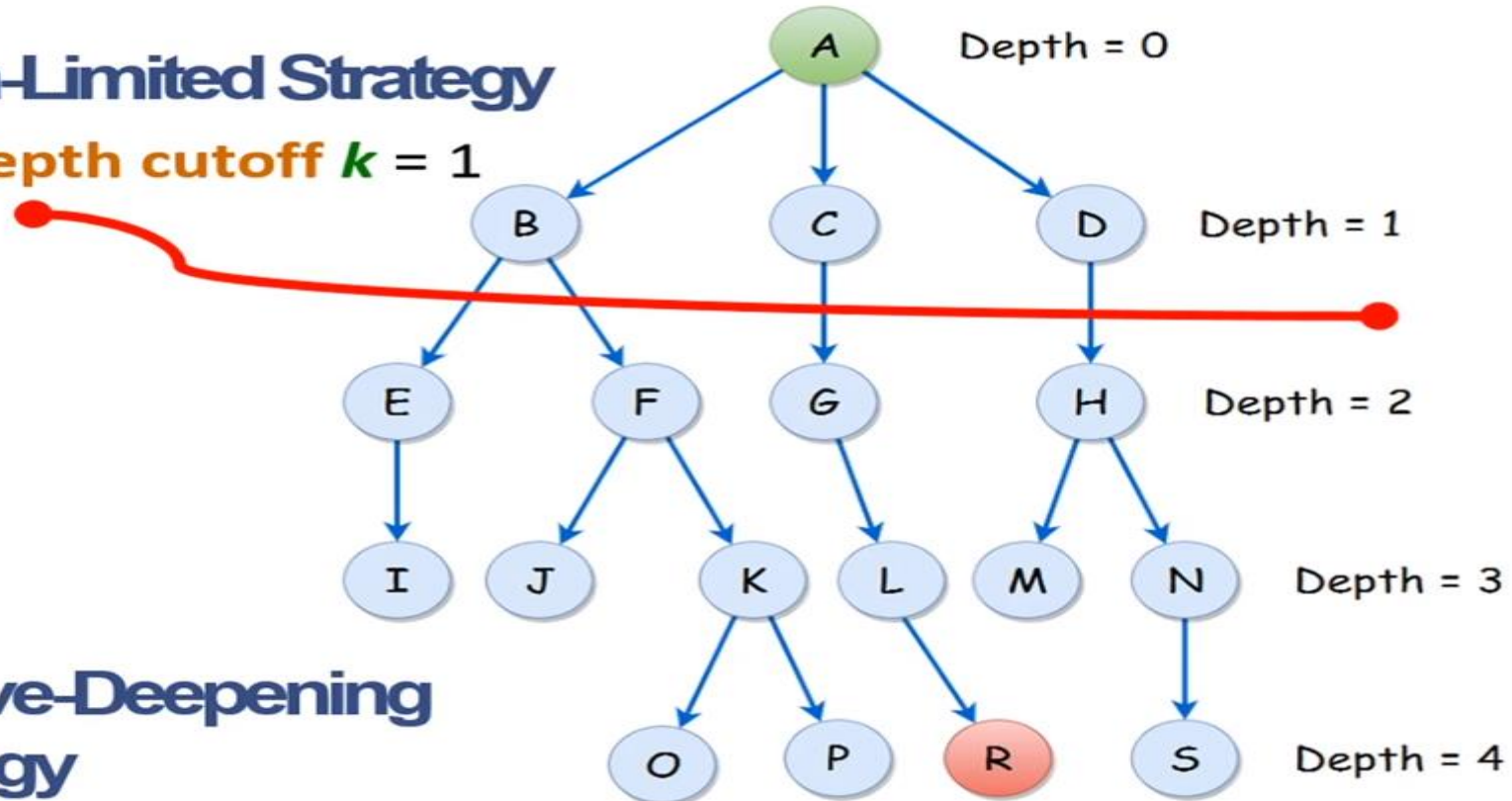
Iterative – Deepening Search

Limit = 3



Depth-Limited Strategy

E.g., **depth cutoff** $k = 1$



Iterative-Deepening Strategy

E.g., Repeat for $k = 0$ to 4

- Perform depth-first with depth cutoff k

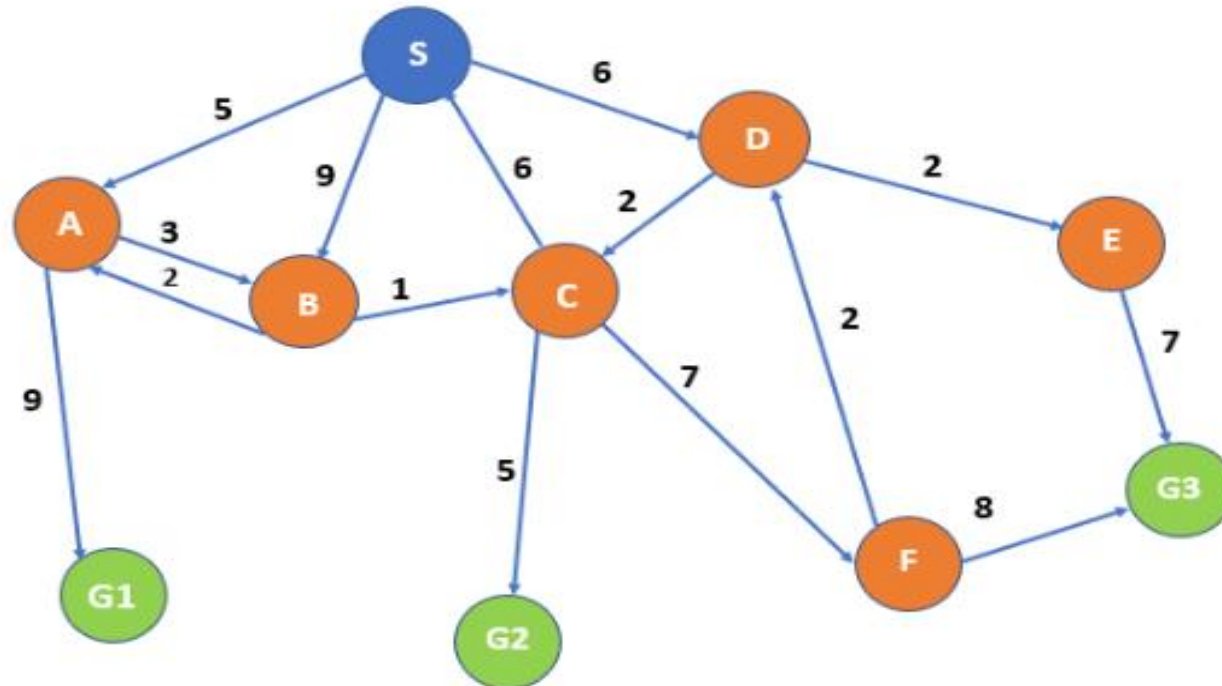


Uniform-Cost Search (UCS)

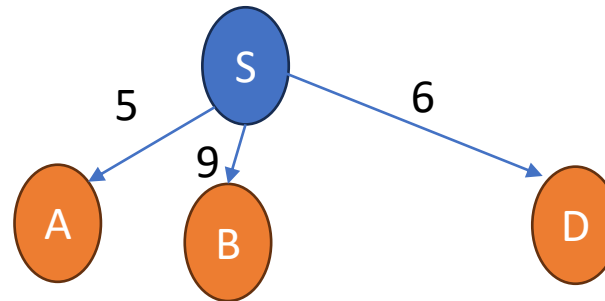
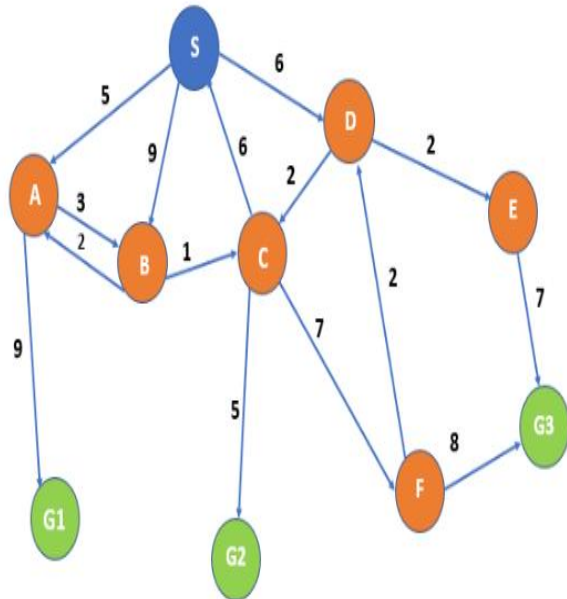
Uniform -Cost Search (UCS) can be applied to find the least-cost path through a graph by maintaining an ordered list of nodes in order of descending cost. This allows us to evaluate the least cost path first.



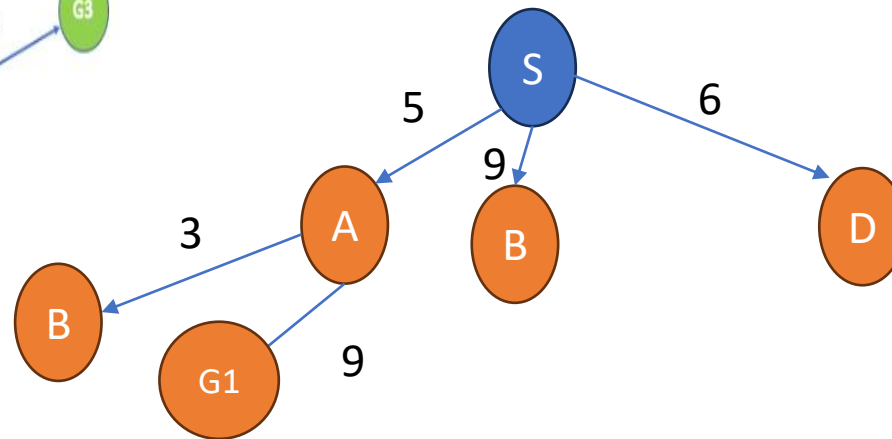
EXAMPLE 1



EXAMPLE 1



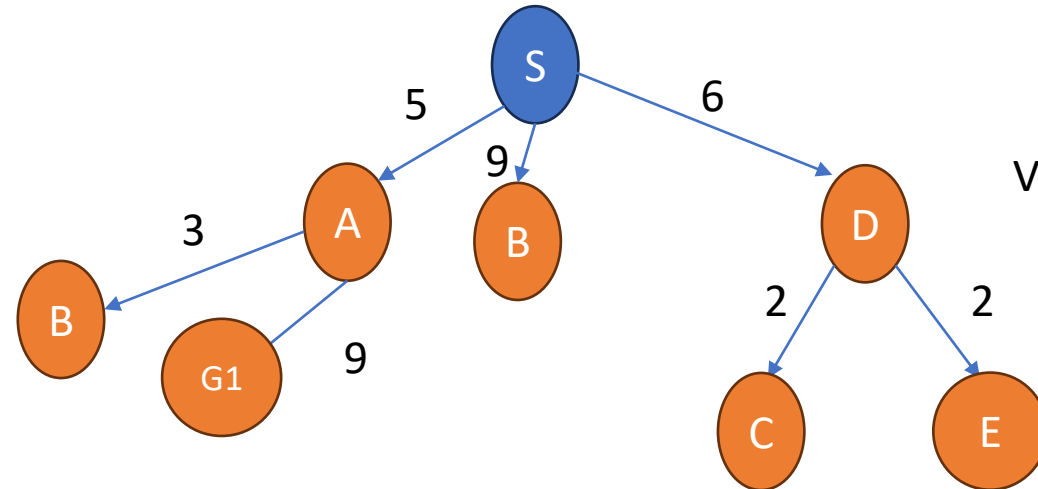
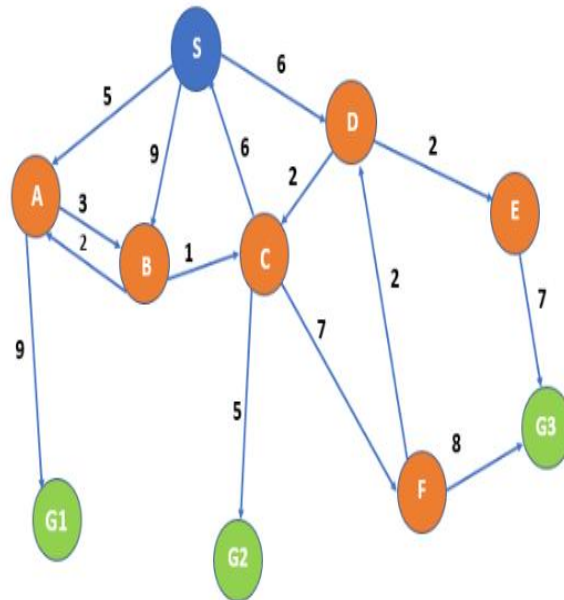
Visit node: S



Visit node: S, A



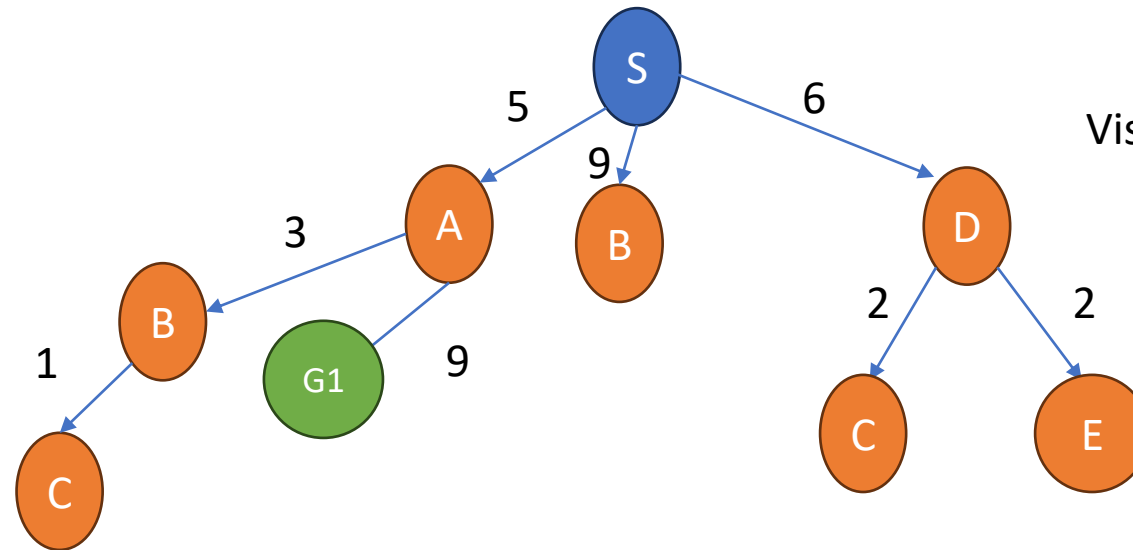
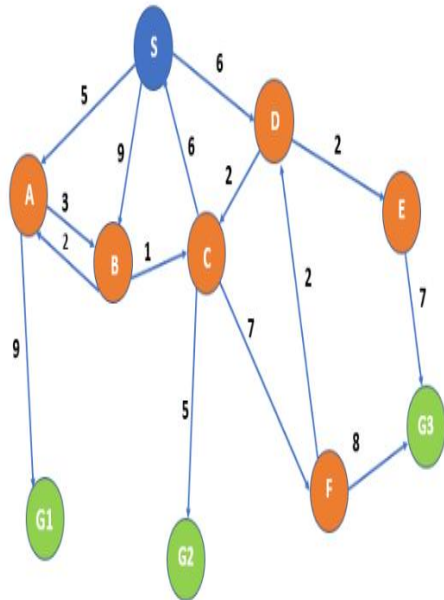
EXAMPLE 1



Visit node: S, A, D



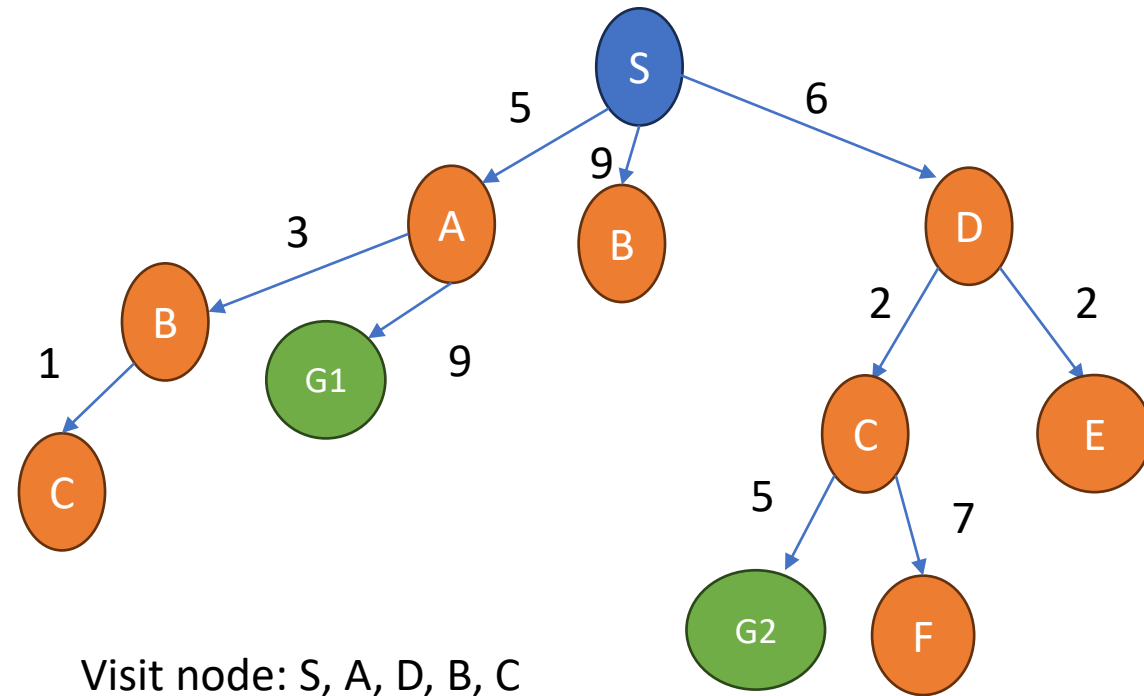
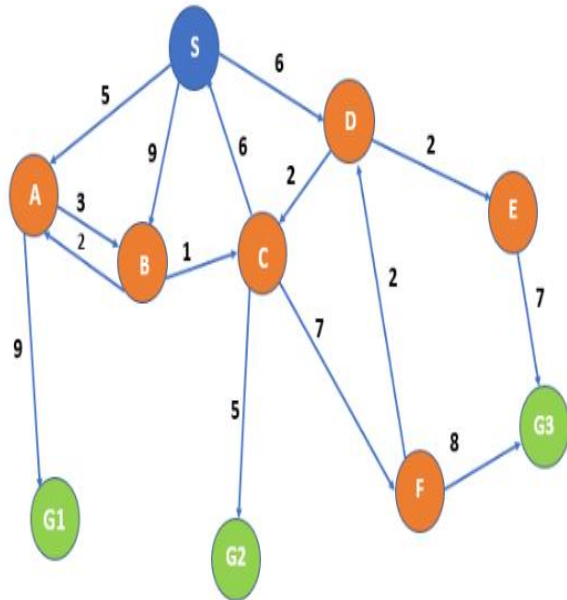
EXAMPLE 1

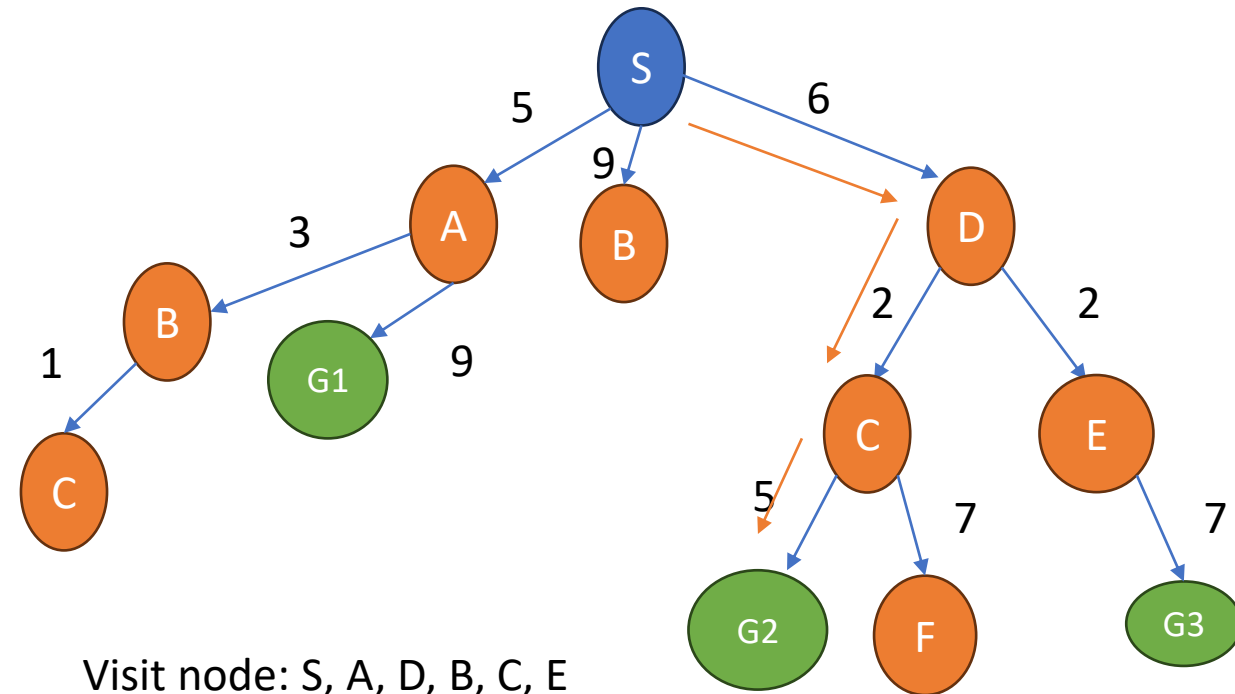


Visit node: S, A, D, B



EXAMPLE 1

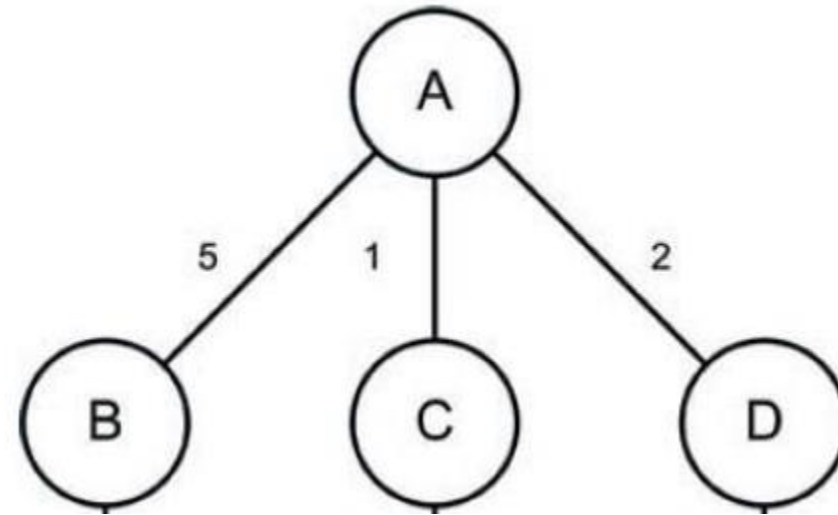
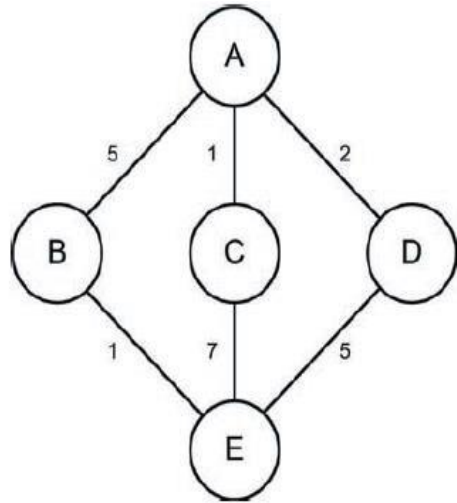


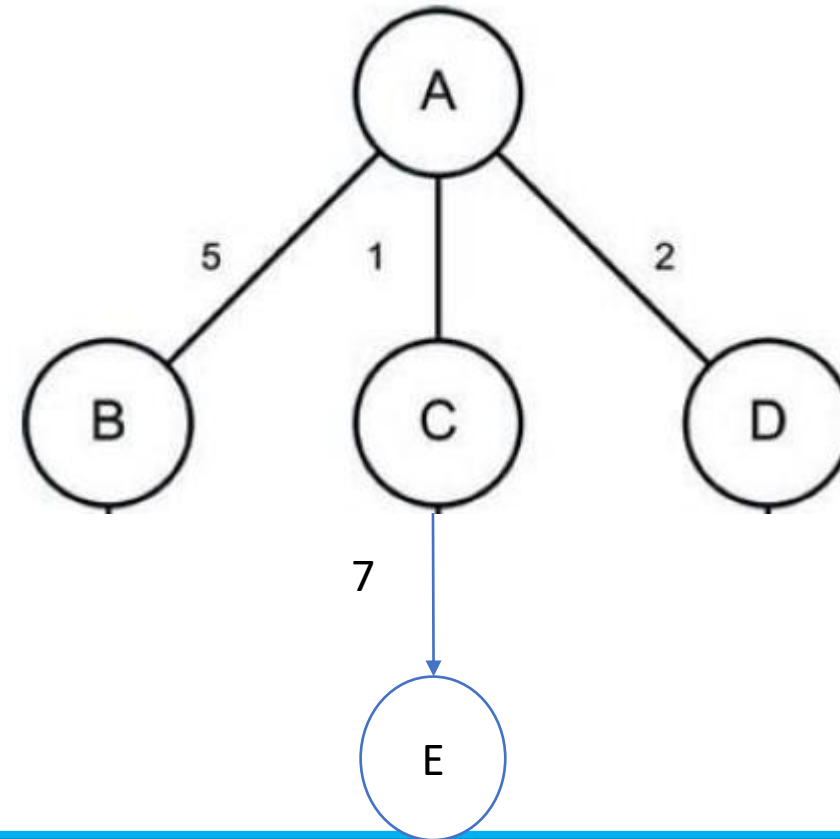
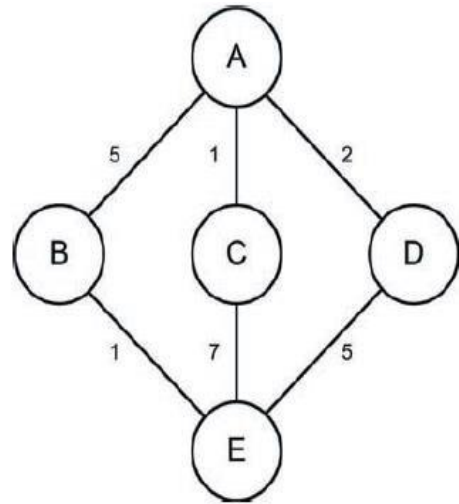


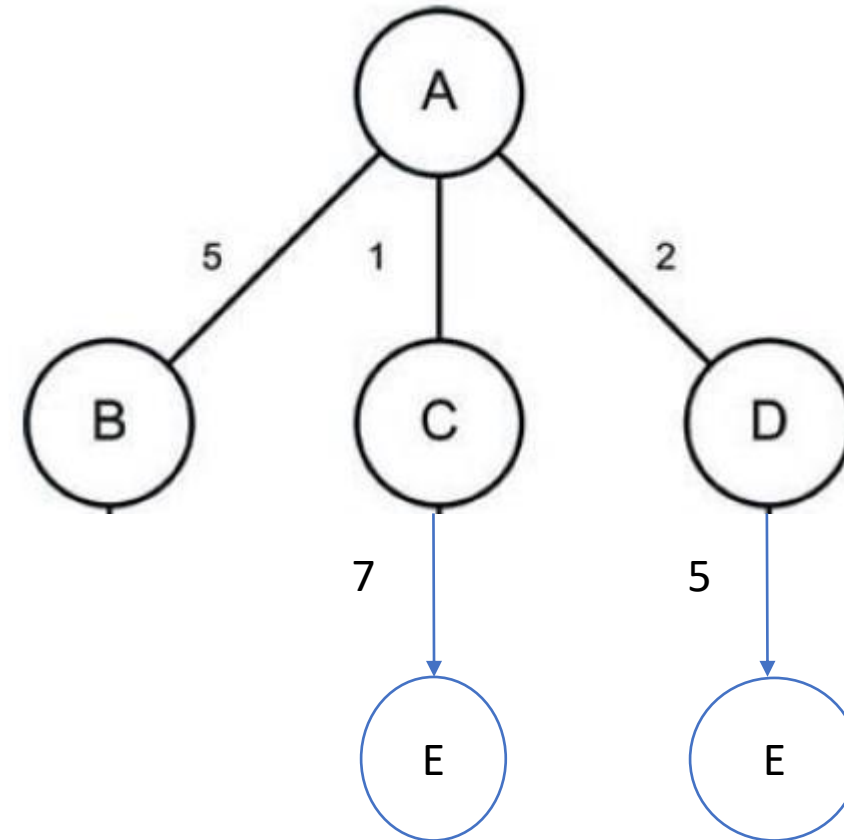
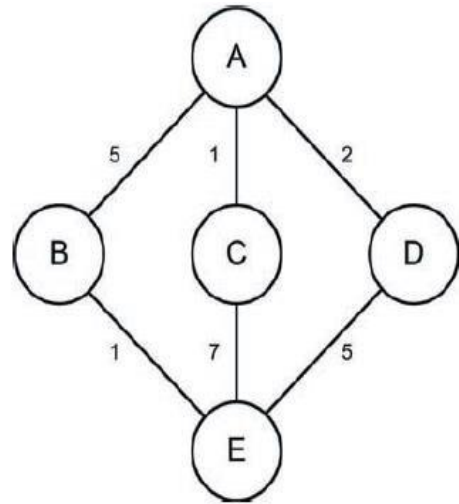
Visit node: S, A, D, B, C, E

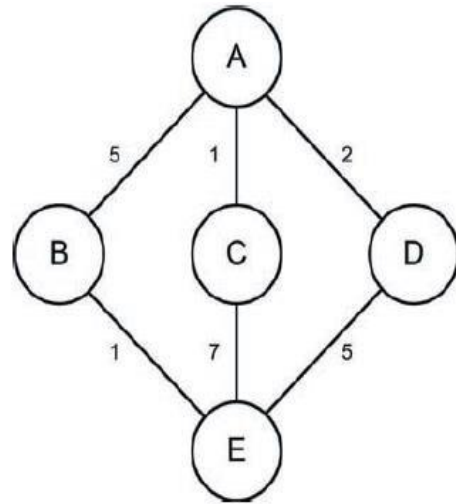
PATH= S D C G2

EXAMPLE 2

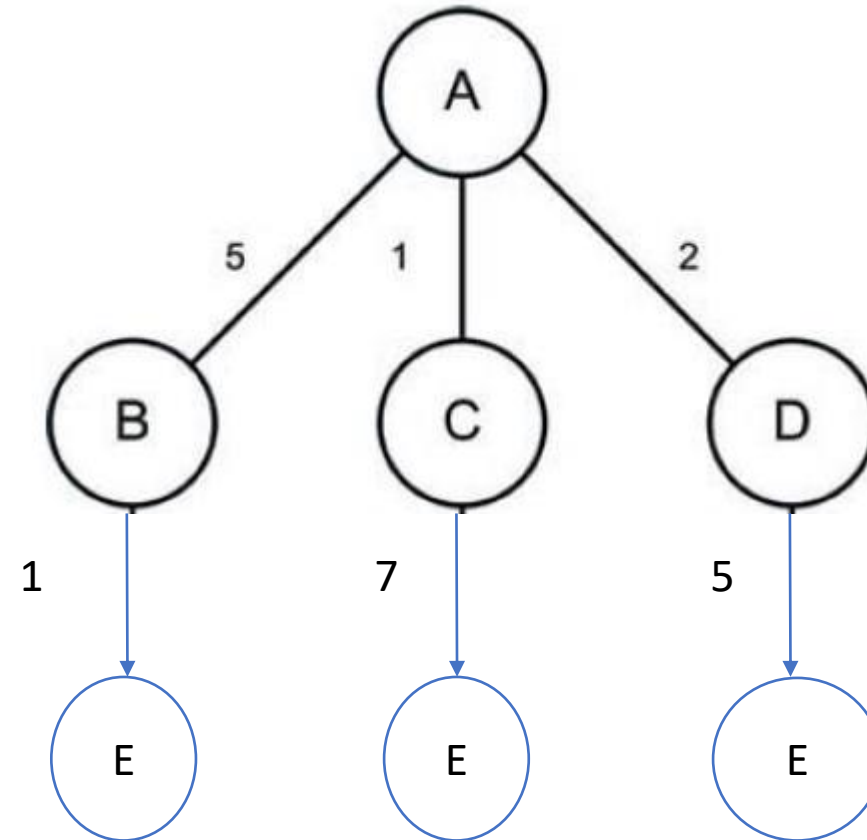








MIN COST = 6
PATH= A B E



ALGORITHM ADVANTAGES

- Each of the algorithms has advantages and disadvantages based on the graph to be searched.
- If the tree is shallow, then BFS is the best choice.
- If the tree is deep, but a solution is known to be shallow in the graph, then IDS is a good choice.
- If the graph is weighted, then UCS should be used as it will always find the best solution where DFS and BFS will not.



Informed Search



Informed Search methods

- Informed search methods incorporate **heuristics**, which determine the quality of any state in the search space.
- In a graph search, this results in a **strategy** for node expansion (which node should be evaluated next).



Informed Search methods

What is a Heuristic?

“A heuristic is a simple or efficient rule for solving a given problem or making a decision.”

- **Heuristics** can be mental shortcuts that ease the cognitive load of making a decision.
- An **approach** to problem solving, learning, or discovery that employs a practical **method** that is not guaranteed to be optimal or perfect.



A* Algorithm

- A* search algorithm works by expanding the path that has a minimum value of the function:

$$f(\text{path}) = g(s) + h(s)$$

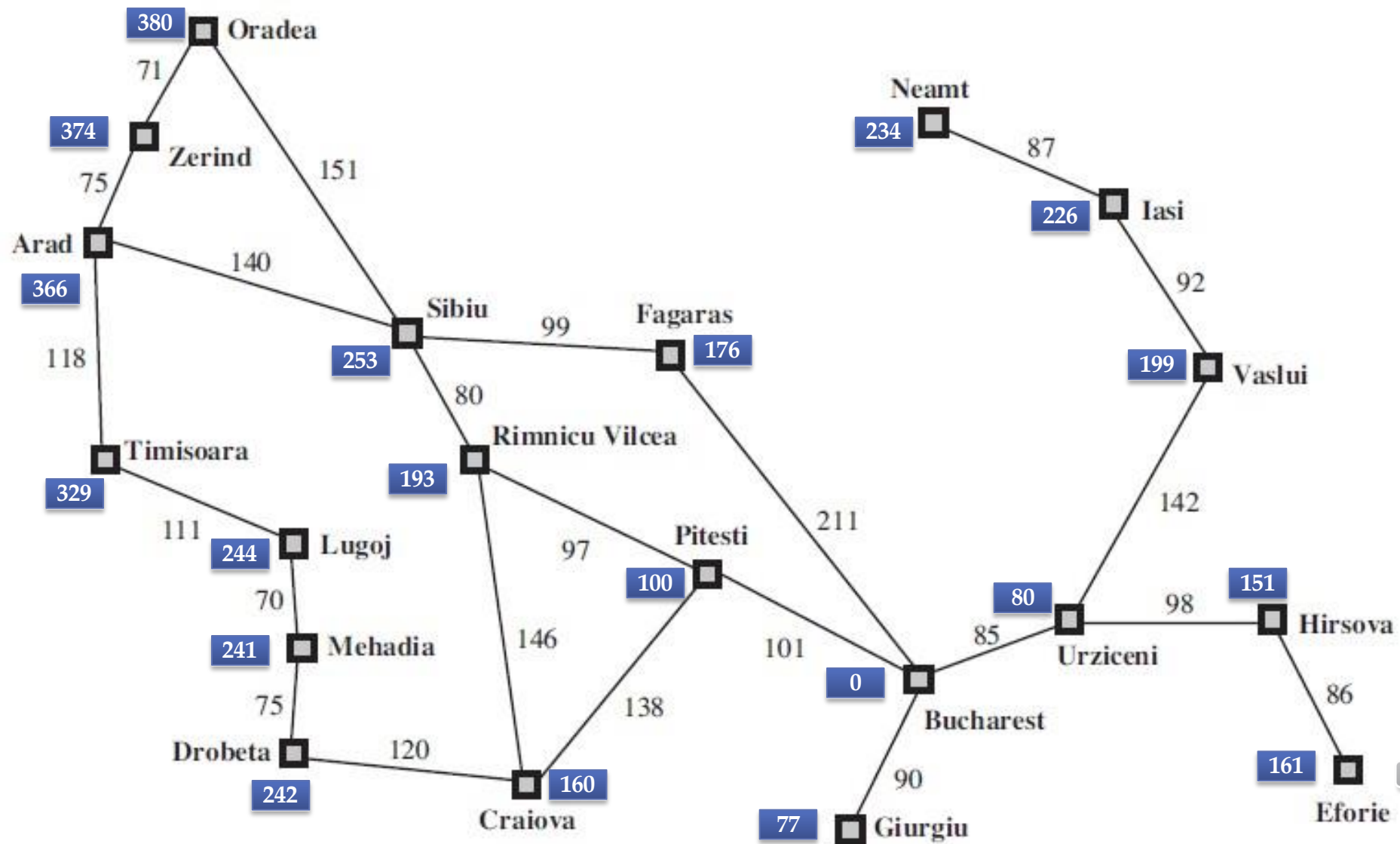
- Where $g(s)$ is the actual path cost to s .
- $h(s)$ is the estimated distance from s to the goal.
- $h(s)$ is a heuristic function that we can use to estimate the distance (cost) from current state to the goal state.
- When A* ends, it gives the actual path cost because the h component at the goal state is 0.



A* Algorithm

- This algorithm always gets the least cost path if $h(s)$ is optimistic / less than the actual cost.





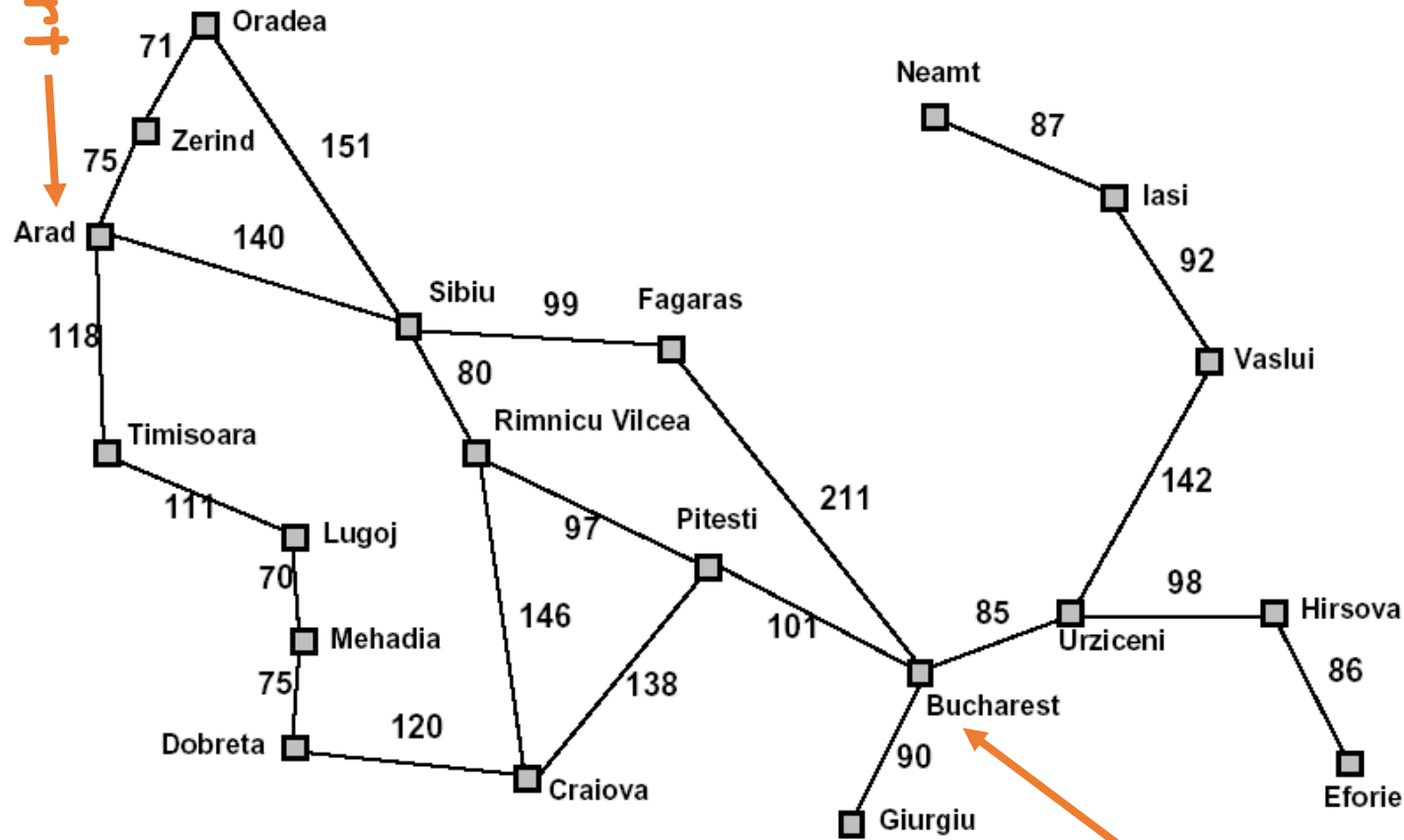
A* Search

- Hart, Nilsson & Rafael 1968
 - Best-first search with $f(n) = g(n) + h(n)$
where $g(n)$ = sum of edge costs from start to n
and $h(n)$ = estimate of lowest cost path n-->goal
 - If $h(n)$ is **admissible** then search will find optimal solution.



Back to Romania

start



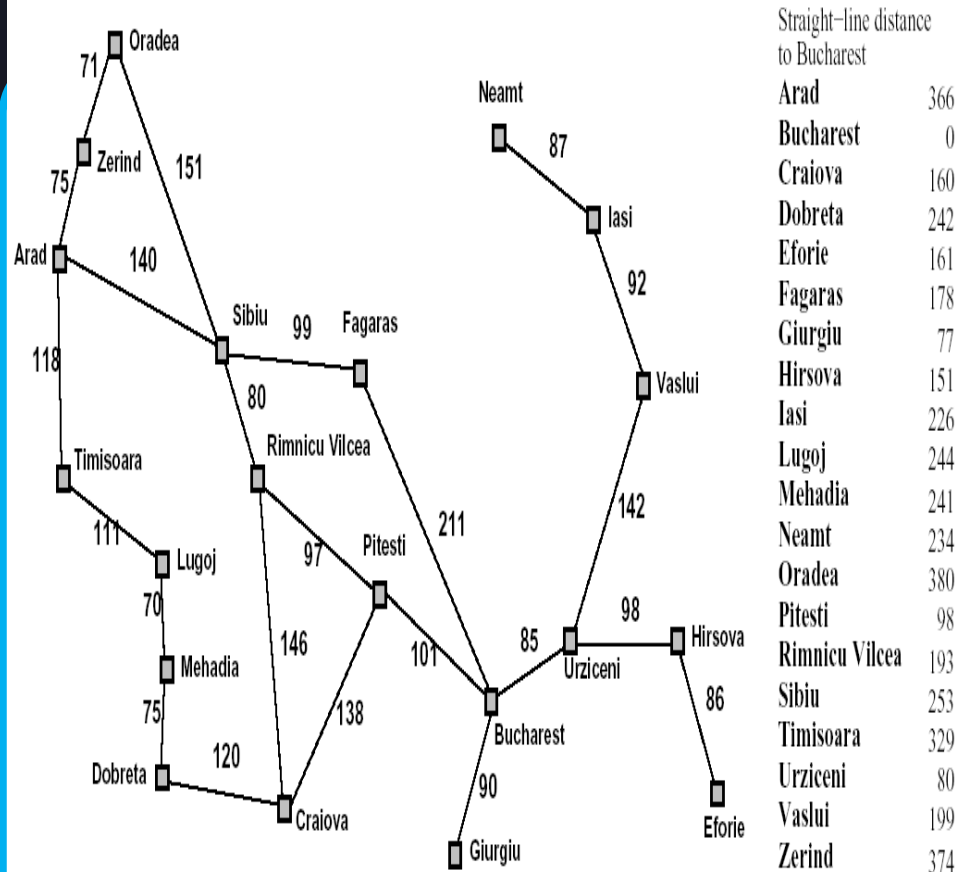
Straight-line distance
to Bucharest

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

end

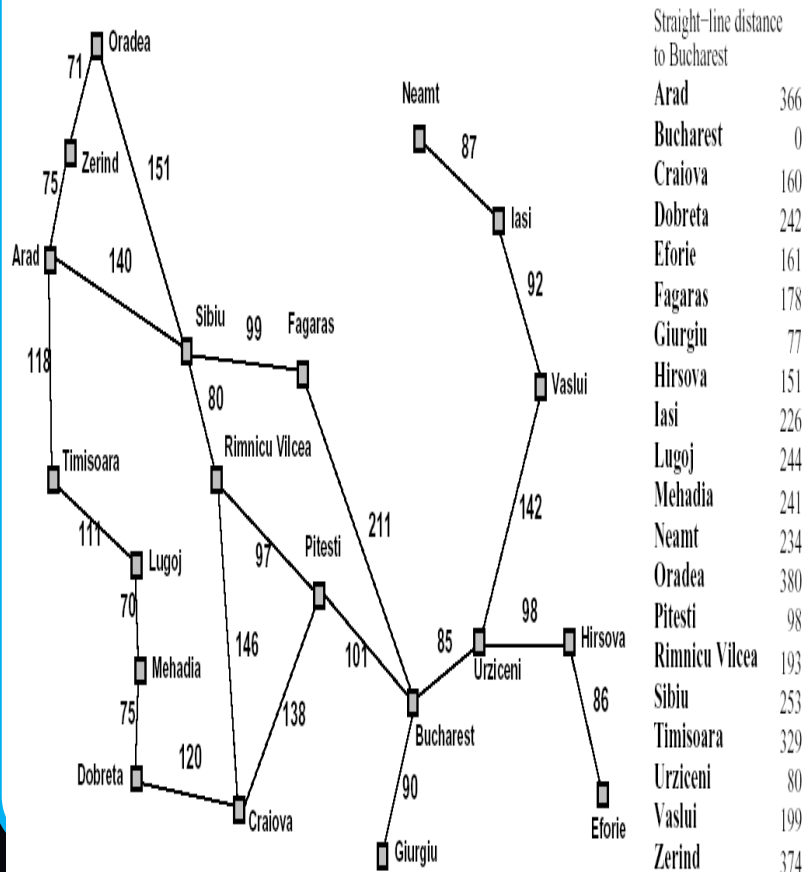


A* for Romanian Shortest Path

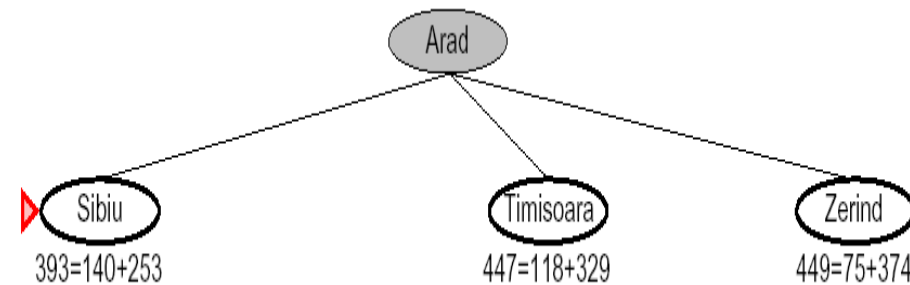


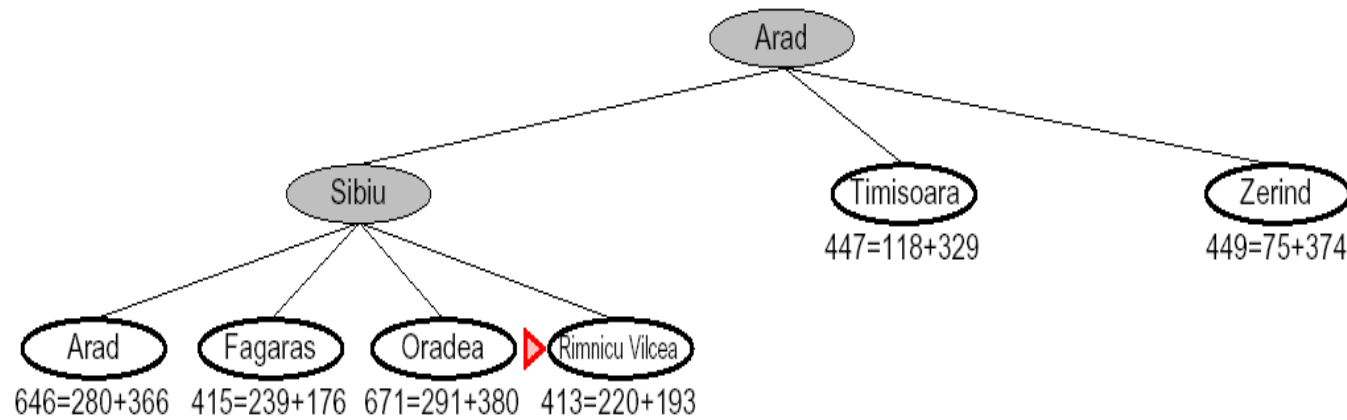
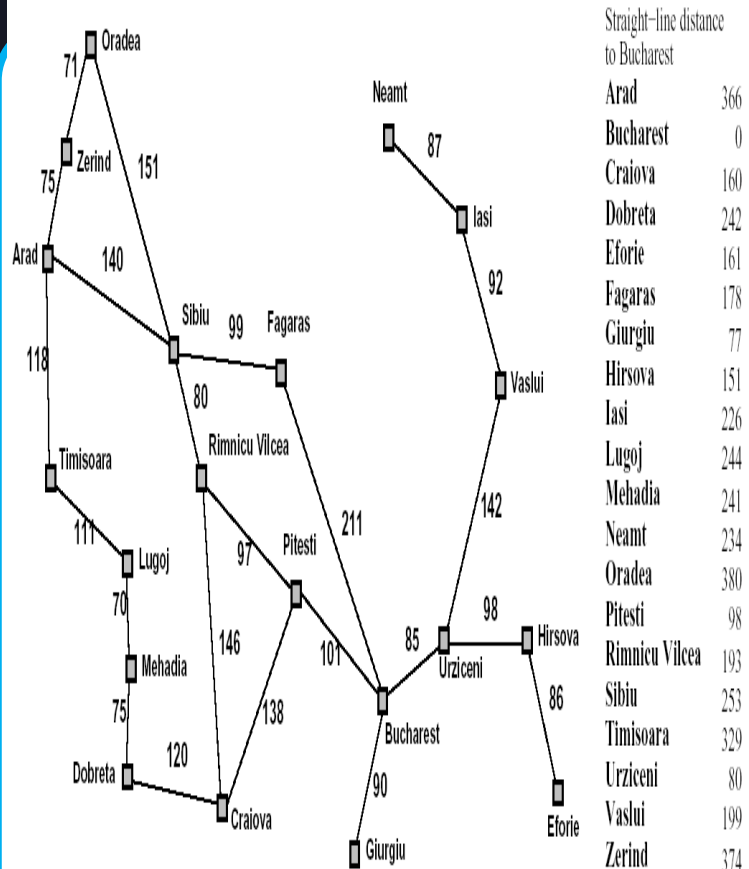
Arad
 $366 = 0 + 366$

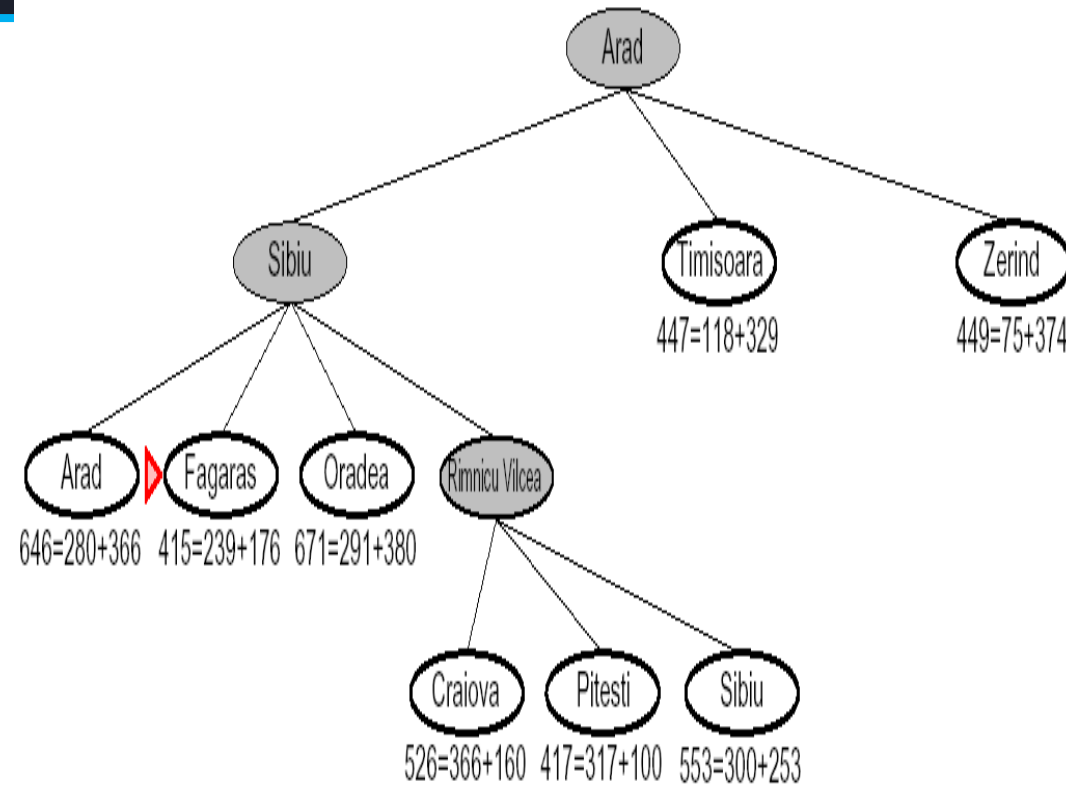
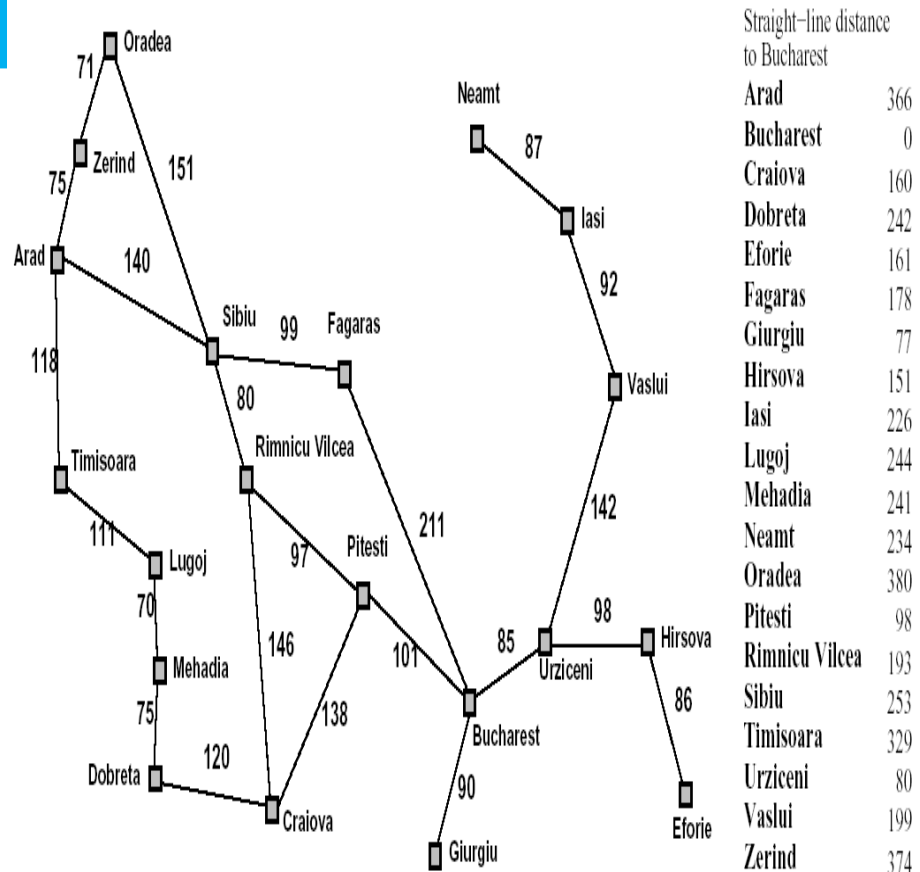


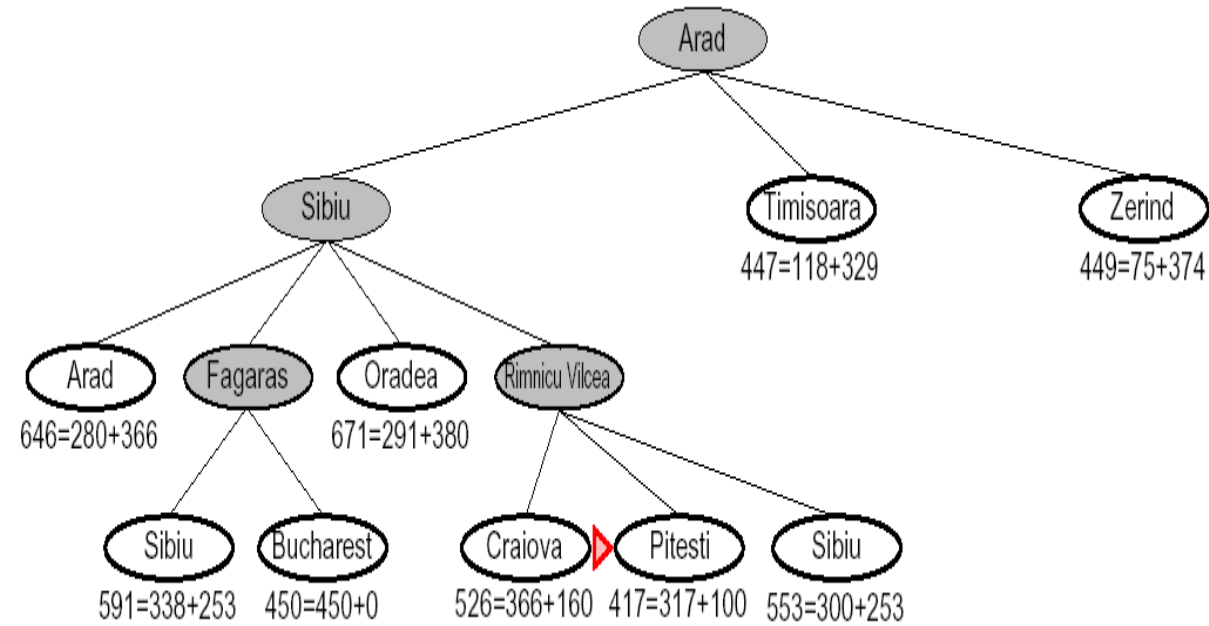
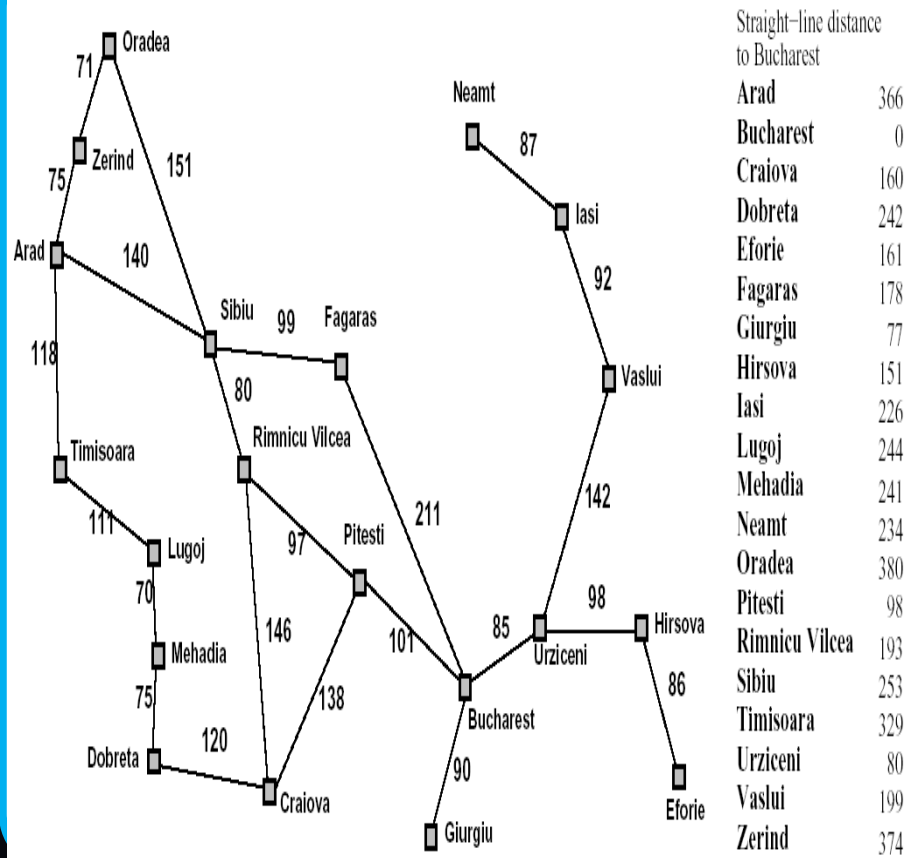


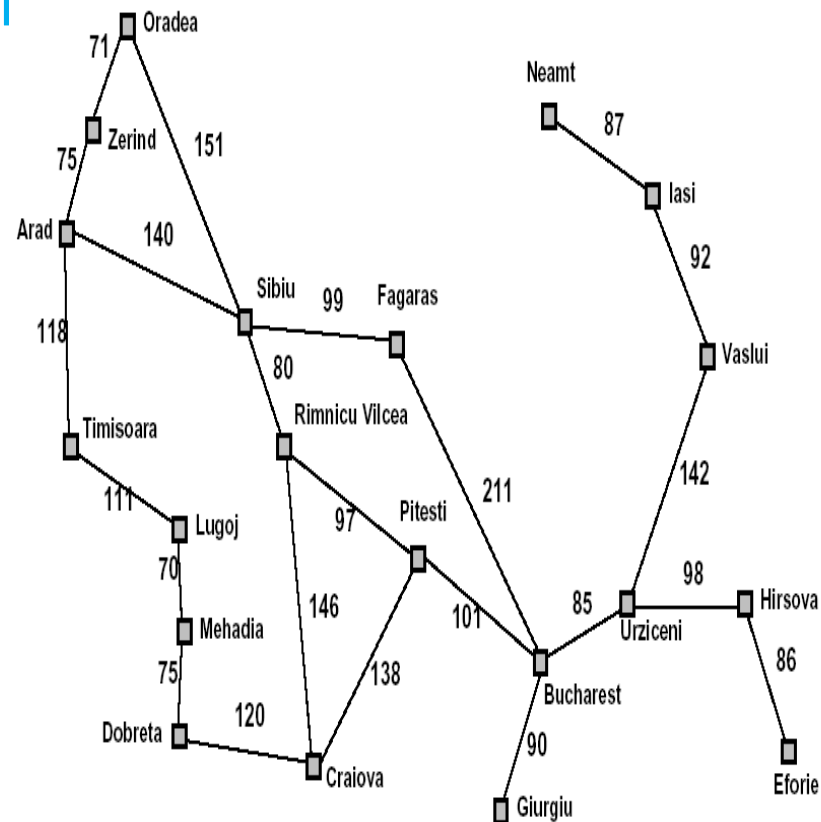
$$f(n) = g(n) + h(n)$$





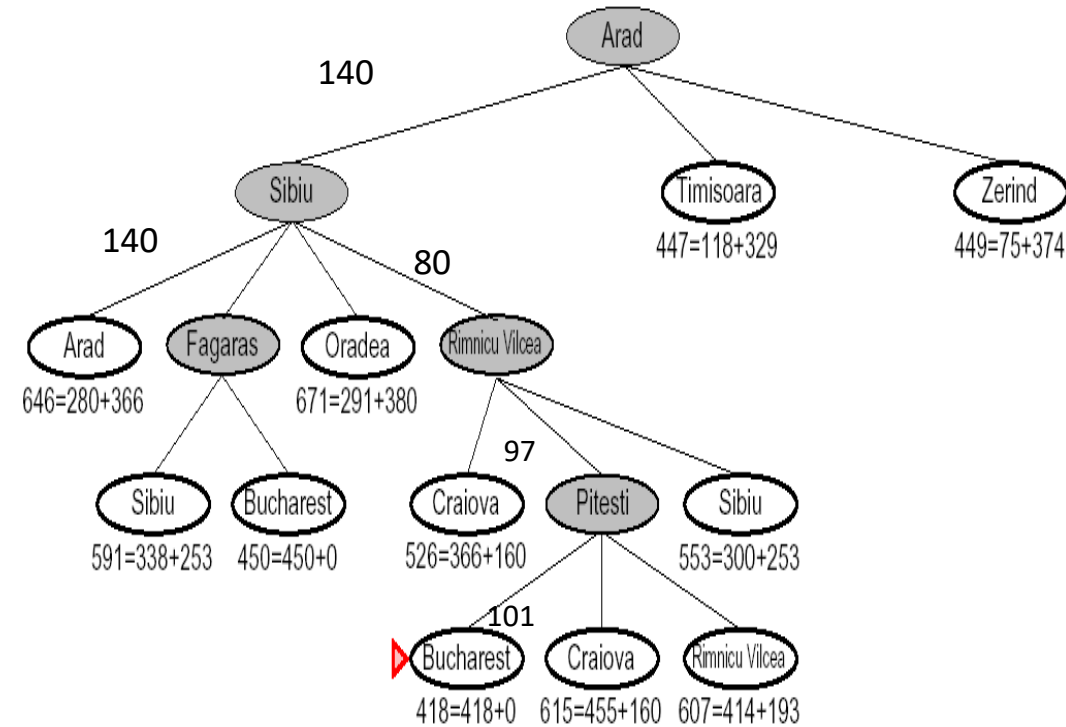






Straight-line distance
to Bucharest

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



Summary

Blind Strategies

- Depth-First Strategy
 - Depth-Limited Strategy
 - Iterative – Deepening Search
- breadth first search
- Uniform-Cost Strategy
- Heuristic Strategies





مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY



Thank You

