Artificial Intelligence and Evolutionary Computing

Lecture 2

IT/T/414A
Winter Semester 2014

Instructor: Kartick Ch. MONDAL

- State Spaces/Search Space
  - A state space is a general formulation of intelligent action.
  - A state contains all of the necessary to predict the effect of an action and to determine is if it is a goal state.
- State space search
  - It is a process used in the field of artificial intelligence (AI) in which successive configurations or states of an instance are considered, with the aim of finding a goal state with a desired property.
  - Problems are often modeled as a state space, a set of states that a problem can be in.
  - State space search often differs from traditional computer science search methods because the state space is implicit.
  - The typical state space graph is too much large to generate and store in memory.
  - Instead, nodes are generated as they are explored, and typically discarded thereafter.

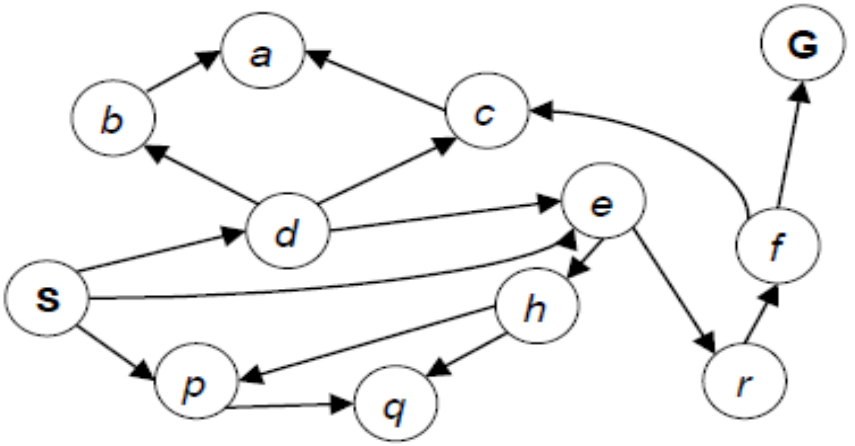
- State-space searching assumes that
  - The agent has perfect knowledge of the state space and can observe what state it is in (full observability).
  - The agent has a set of actions that have known deterministic effects.
  - Some states are goal states, the agent wants to reach one of these goal states.
  - The agent can recognize a goal state.
  - A solution is a sequence of actions that will get the agent from its current state to a goal state.

- State Space problem or Search problem consists of
  - A set of states
  - A distinguished set of states called the start state
  - A set of actions available to the agent in each state
  - An action function/successor function that, given a state and an action, returns a new state
  - A set of goal states, specified as a boolean function, goal(s), that is true when s is a goal state
  - A criterion that specifies the quality of an acceptable solution.
    - For example, any sequence of actions that gets the agent to the goal state may be acceptable,
    - Or there may be costs associated with actions and the agent may be required to find a sequence that has minimal total cost. This is called an optimal solution.

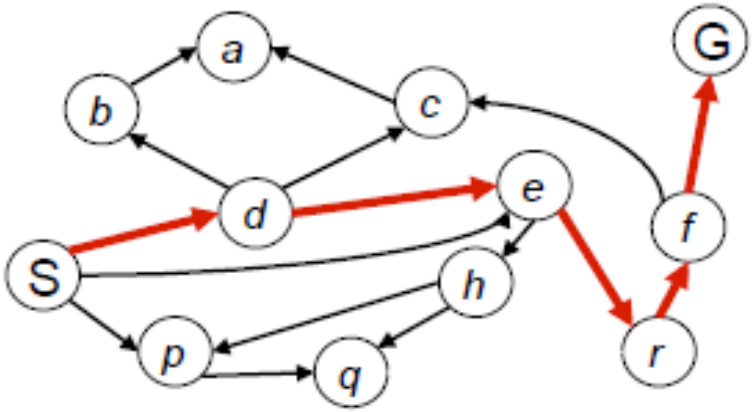
- State Space representation of problem
  - All the states the system can be in are represented as nodes of a graph.
  - An action that can change the system from one state to another (e.g. a move in a game) is represented by a link from one node to another.
  - Links may be unidirectional (e.g. Tic-tac-toe (Xs and Os), chess, can't go back) or bi-directional (e.g. geographic move).
  - Search for a solution.
  - A solution might be:
    - Any path from start state to goal state.
    - The best (e.g. lowest cost) path from start state to goal state (e.g. Travelling salesman problem).
  - It may be possible to reach the same state through many different paths (obviously true in Tic-Tac-Toe).
  - There may be loops in the graph (can go round in circle). No loops in Tic-Tac-Toe but in Travelling Salesman Problem.

State Space Graph

- The best tool to demonstrate the state-space representation of a problem is the state-space graph.
- A state space graph is a mathematical representation of a search problem
- For every search problem, there's a corresponding state space graph
- The successor function is represented by arcs
- Search space of many practical problems is so large that they cannot be represented by explicit graph and used implicit graph representation.
- An implicit graph representation is a graph whose vertices or edges are not represented as explicit objects in a computer's memory, but rather are determined algorithmically from some more concise input.

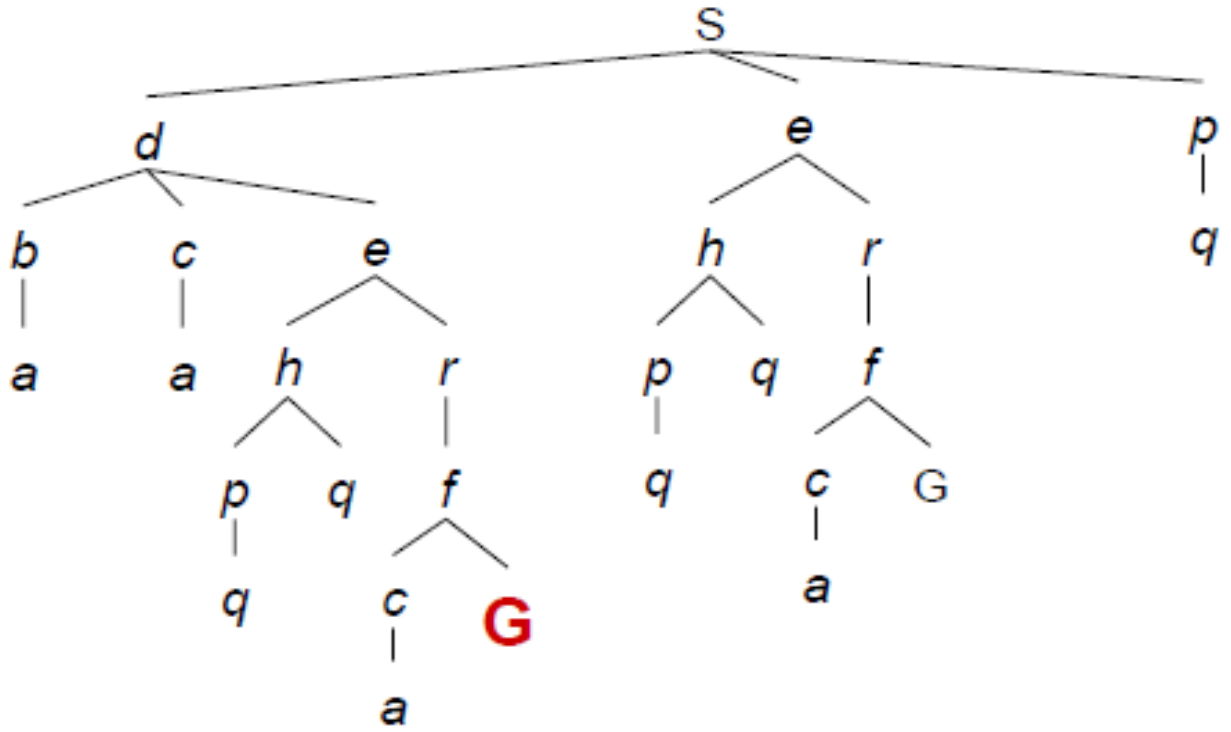


State Graphs vs. Search Trees



Each NODE in in the search tree is an entire PATH in the problem graph.

We construct both on demand – and we construct as little as possible.



- Problem-Solving Agents / Goal-based Agents
  - Intelligent agents can solve problems by searching a state-space

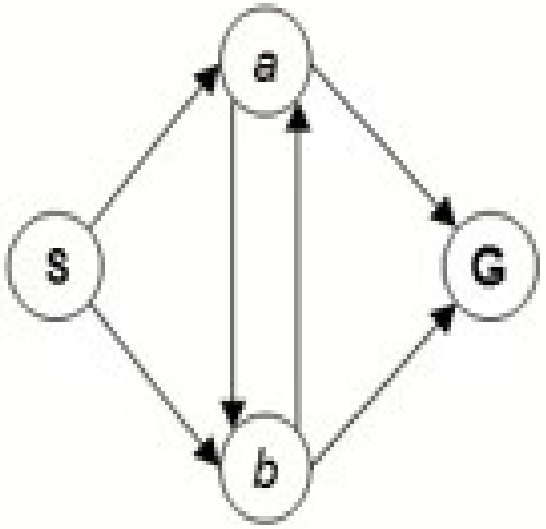
- State-space Model
  - The agent's model of the world
  - Usually a set of discrete states
  - e.g., in driving, the states in the model could be towns/cities

- Goal State(s)
  - A goal is defined as a desirable state for an agent
  - There may be many states which satisfy the goal test e.g., drive to a town with a ski-resort
  - Or just one state which satisfies the goal e.g., drive to Mammoth

- Operators (actions, successor function)
  - Operators are legal actions which the agent can take to move from one state to another

Quiz: State Graphs vs. Search Trees

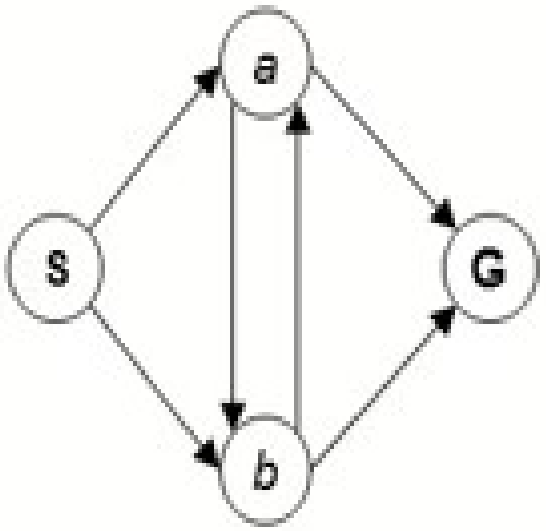
Consider this 4-state graph:



??

Quiz: State Graphs vs. Search Trees

Consider this 4-state graph:



Lots of repeated structure in the search tree!!!!!
Which one to expand first???

Example: Traveling in Romania

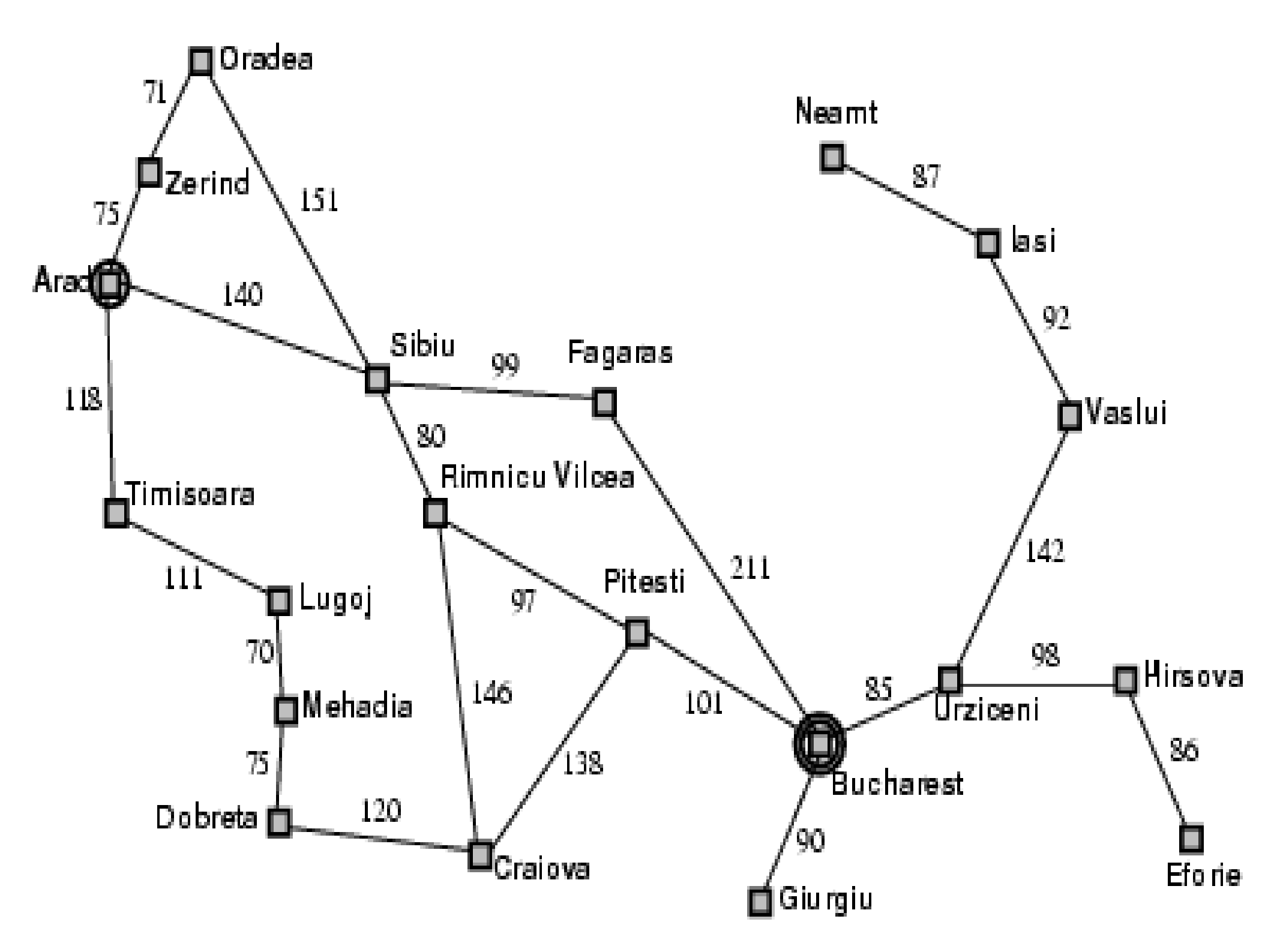
- On holiday in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

- Formulate goal:
  - Be in Bucharest

- Formulate problem:
  - States: various cities
  - Actions/operators: drive between cities

- Find solution
  - By searching through states to find a goal
  - Sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

- Execute states that lead to a solution


Oradea
71
Neamt
Zerind
87
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
211
Pitesti
Lugoj
97
98
70
Hirsova
146
85
Mehadia
101
Urziceni
75
86
138
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu

State-Space Problem Formulation
A problem is defined by four items:

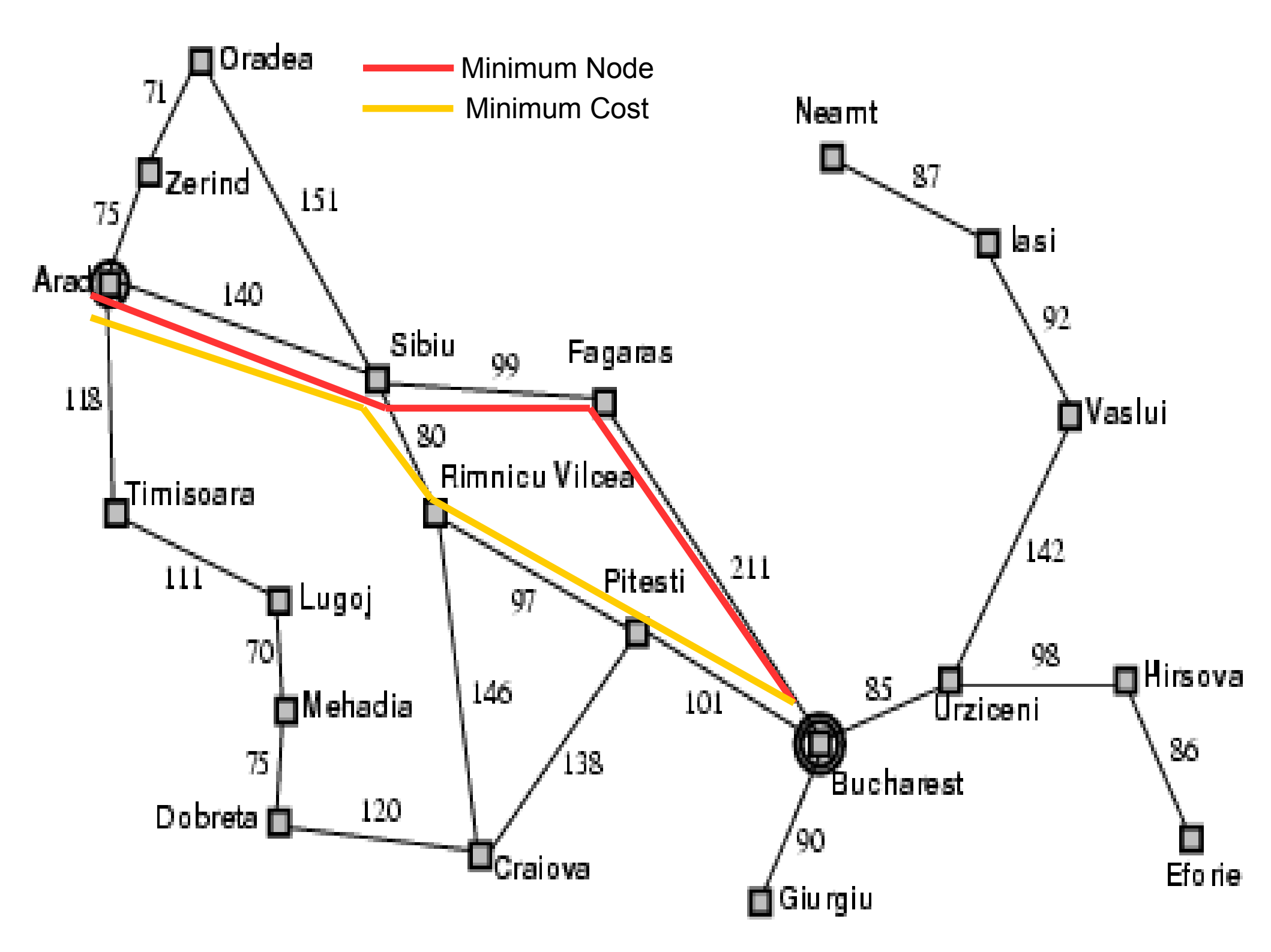
1. Initial state e.g., "at Arad"

2. Actions or successor function
   S(x) = set of action–state pairs
   e.g., *S(Arad)* = {<*Arad Zerind, Zerind*>, … }

3. Goal test (or set of goal states)
   e.g., *x* = "at Bucharest", *Checkmate(x)*

4. Path cost (additive)
   e.g., sum of distances, number of actions executed, etc.
   c(x,a,y) is the step cost, assumed to be ≥ 0

A solution is a sequence of actions leading from the initial state to a goal state

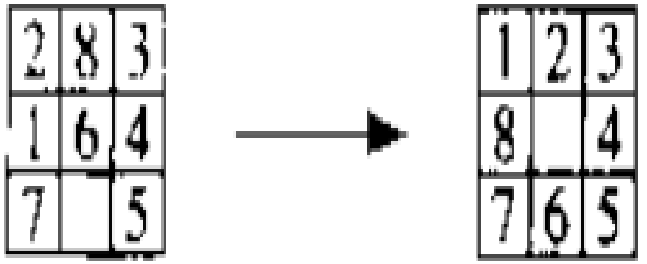

Minimum Node
Minimum Cost
Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Dobreta
120
Craiova
Bucharest
90
Giurgiu
Neamt
87
Iasi
92
Vaslui
142
98
85
Urziceni
Hirsova
86
Efo rie

Search Process

- Search Algorithm
  - Uninformed Search:
    - Where we have no problem specific reason to prefer one part of the search space to any other as far as finding a path to the goal is concerned.
      - Breadth First Search
      - Depth First Search
      - Uniformed Cost Search
  - Informed Search:
    - Heuristic Search
      - Where we have problem specific information to help focus the search
        - Best First Search
    - A* Search

Search Process Formulation

- 8 Puzzle Problem
  - Consists of 8 tiles set in a three by three array, leaving one empty or blank cell into which an adjacent tile can be slid.
  - The task is to find a sequence of tile movements that transforms an initial disposition of tiles into some particular arrangement.



8 Puzzle Problem

- We have a start state as left side of the figure
- A goal state is the array representing the configuration on the right side of the figure.
- The moves between states correspond to sliding a tile into the blank cell.
- In this problem,
  - we can imagine 8x4 different moves
    - move 1 up, move 1 down, move 1 left, move 1 right,
    - move 2 up and ... so on.
-

8 Puzzle Problem

- A compact formulation of these 8x4 moves involves move blank left, move blank right, move blank up, move blank down.
- It is not possible to apply all these moves in any given state.
- A given start state and the set of possible moves implicitly define the graph accessible from start.
- The number of nodes in the state space graph for this problem is 9! = 3,62,880

Search Process

- Search Algorithm
  - Uninformed Search:
    - Where we have no problem specific reason to prefer one part of the search space to any other as far as finding a path to the goal is concerned.
      - Breadth First Search
      - Depth First Search
      - Uniformed Cost Search
  - Informed Search:
    - Heuristic Search
      - Where we have problem specific information to help focus the search
        - Best First Search
    - A* Search

Breadth First Search

- The most simplest uninformed search procedure.
- The procedure generates an explicit state space graph by applying all possible operators to the start node, then to their successors and so on.
- Search proceeds uniformly outward from the start node.
- We apply the successor function (all possible operators) to a node which produces the entire set of nodes is called Expansion of node.
- In this search when a goal node is found, it will be the minimal path length to the goal.
- The disadvantage is that it requires the generation and storage of a tree whose size is exponential to the depth of the tree.

Breadth First Search

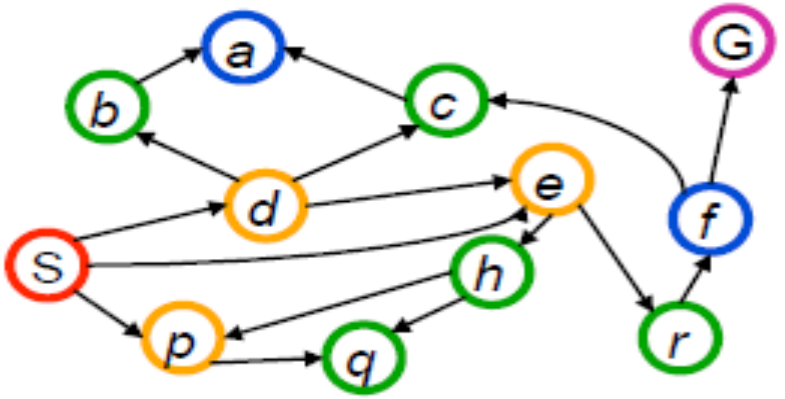
Input: A graph G and a start node s of G

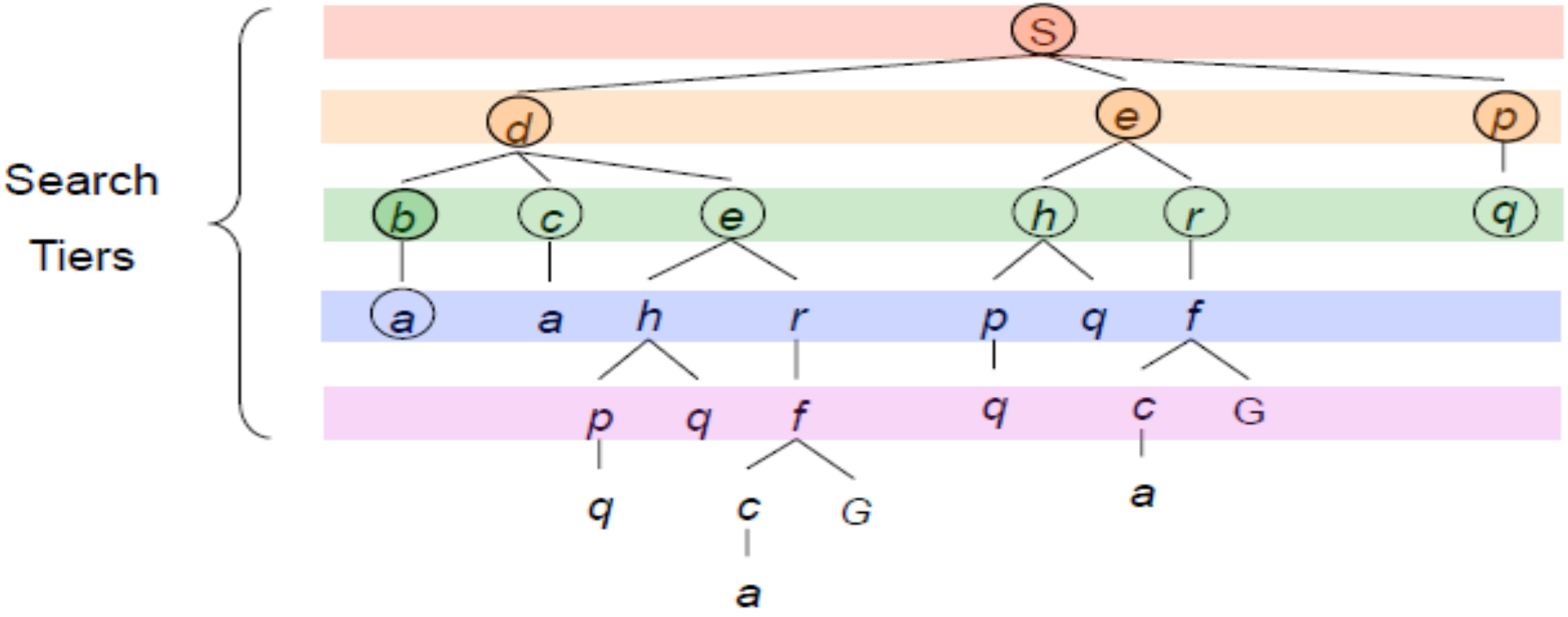
```
Procedure BFS(G,s) is
    Create a queue Q
    Create a set S
    Enqueue s onto Q
    Add s to S
    While Q is not empty loop
        t ← Q.dequeue()
        If t is what we are looking for then
            Return t
        End if
        For all edges e in G.adjacentEdges(t) loop
            u ← G.adjacentVertex(t,e)
            If u is not in S then
                Add u to S
                Enqueue u onto Q
            End if
        End loop
    End loop
    Return none
End BFS
```

Breadth First Search

Strategy: expand shallowest node first

Implementation: Fringe is a FIFO queue

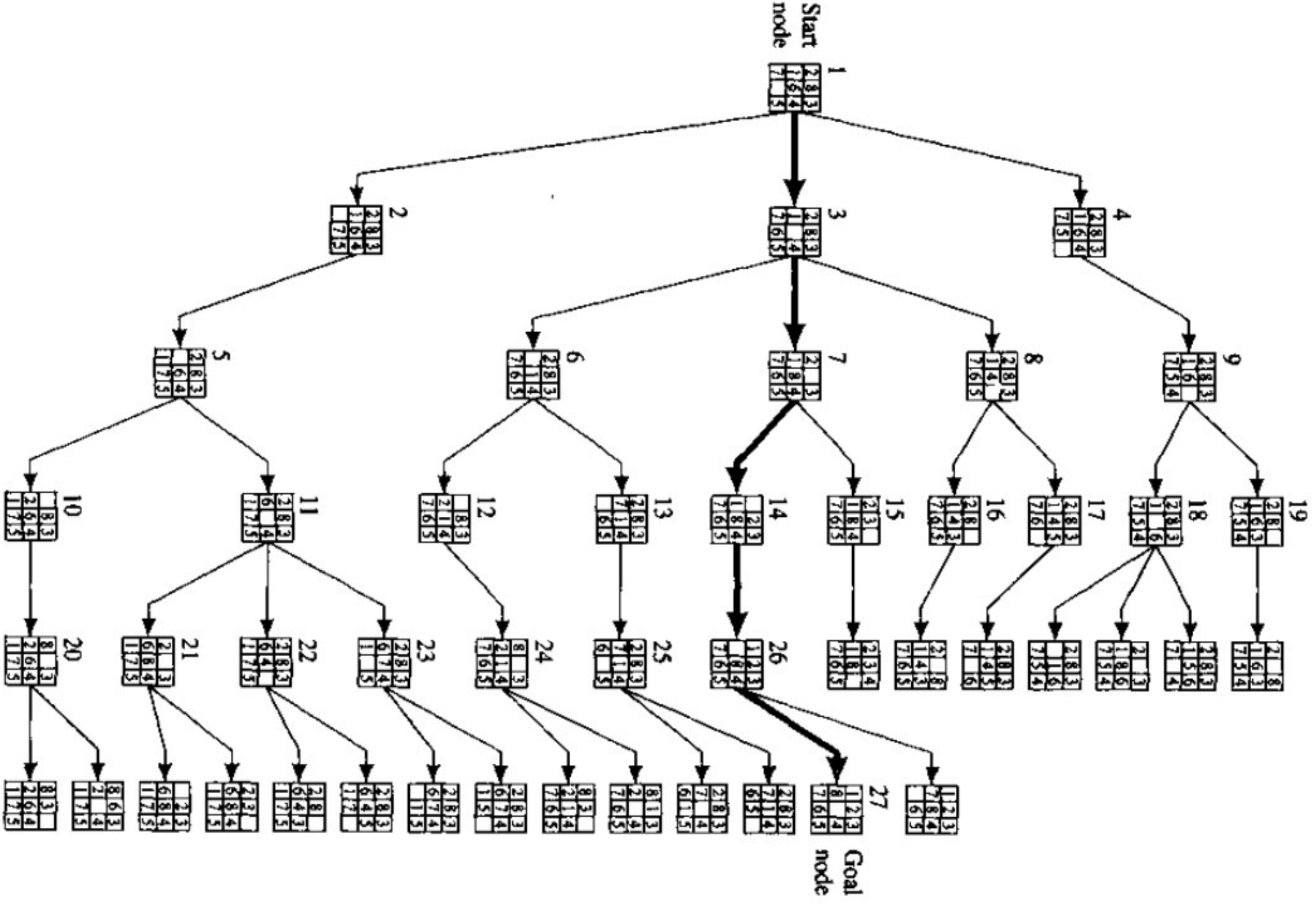




Breadth First Search

- 8 puzzle problem
  - Followed a specific order of actions to expand the node is move blank left, move blank up, move blank right, move blank down.
  - Each move is reversible, so the arc from successors back to their parents are omitted.

Breadth First Search



Search Process

- Search Algorithm
  - Uninformed Search:
    - Where we have no problem specific reason to prefer one part of the search space to any other as far as finding a path to the goal is concerned.
      - Breadth First Search
      - Depth First Search
      - Uniformed Cost Search
  - Informed Search:
    - Heuristic Search
      - Where we have problem specific information to help focus the search
        - Best First Search
    - A* Search

Depth First Search

- It generates the successors of a node just one at a time by applying individual operator instead of a successor function.
- A trace is left at each node to indicate additional operators can eventually be applied if needed.
- A decision is taken about which node should apply first and which next.
- As soon as a successors is generated, one of its successor is generated and so on.
- A depth bound is use to prevent the search process from running away.
-

Depth First Search

- No successor is generated after the specified depth bound
- This kind of search obliges us to save that part of the search tree consisting of the path currently being explored and traces at the not yet fully expanded node along that path
- The memory requirement of the search process is linear in the depth bound.
- The disadvantage is the goal reaching path is not guaranteed to have minimal length or optimal path.
- It finds the LEFTMOST solution regardless of depth and cost.
- Another disadvantages is that we may have to explore a large part of the search space to find a shallow goal if it is the only goal and expanded late in the process.

Depth First Search

- **Input:** A graph G and a vertex v of G
- **Output:** All vertices reachable from v labeled as discovered
- A recursive implementation of DFS:
- **Procedure DFS(G,v):**

```
Label v as discovered
For all edges from v to w in G.adjacentEdges(v) do
    If vertex w is not labeled as discovered then
        Recursively call DFS(G,w)
```

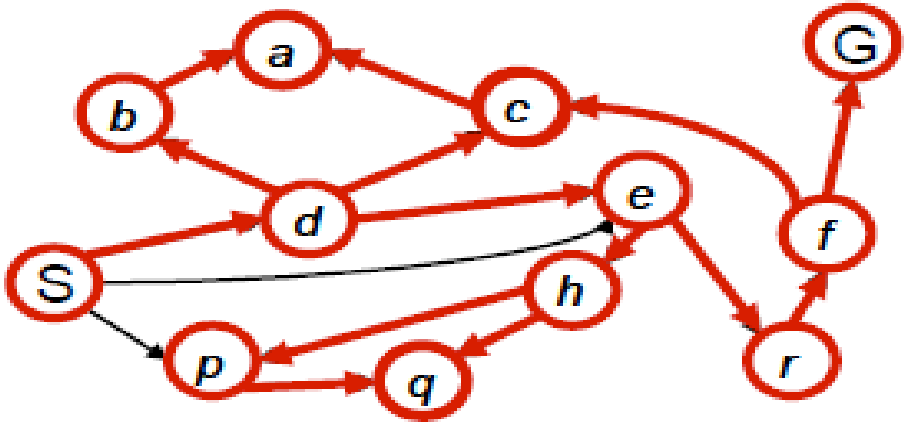
- A non-recursive implementation of DFS:
- **Procedure DFS-iterative(G,v):**

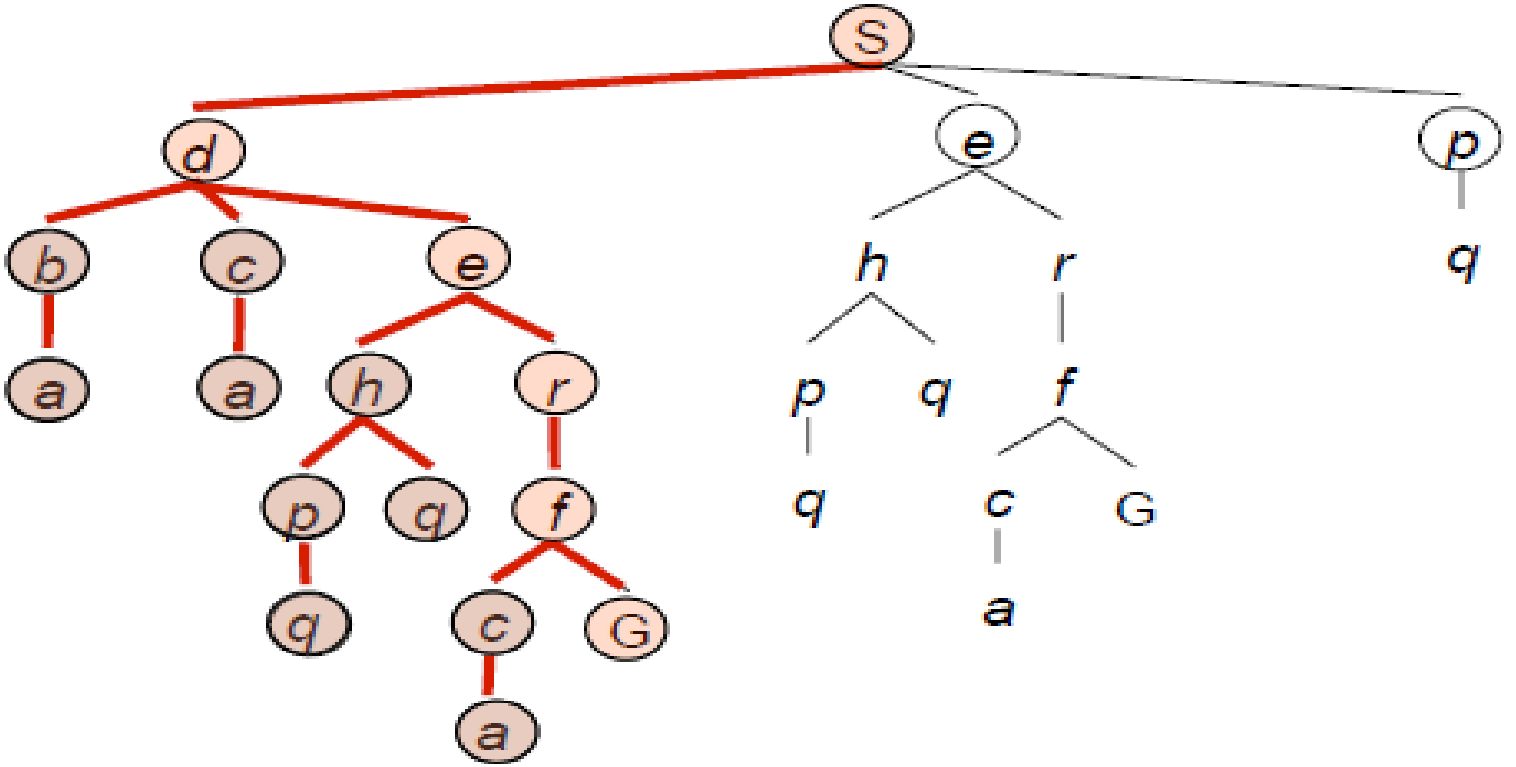
```
Let S be a stack
S.push(v)
While S is not empty
    v ← S.pop()
    If v is not labeled as discovered:
        Label v as discovered
        For all edges from v to w in G.adjacentEdges(v) do
            S.push(w)
```

Depth First Search

Strategy: expand deepest node first

Implementation: Fringe is a LIFO stack

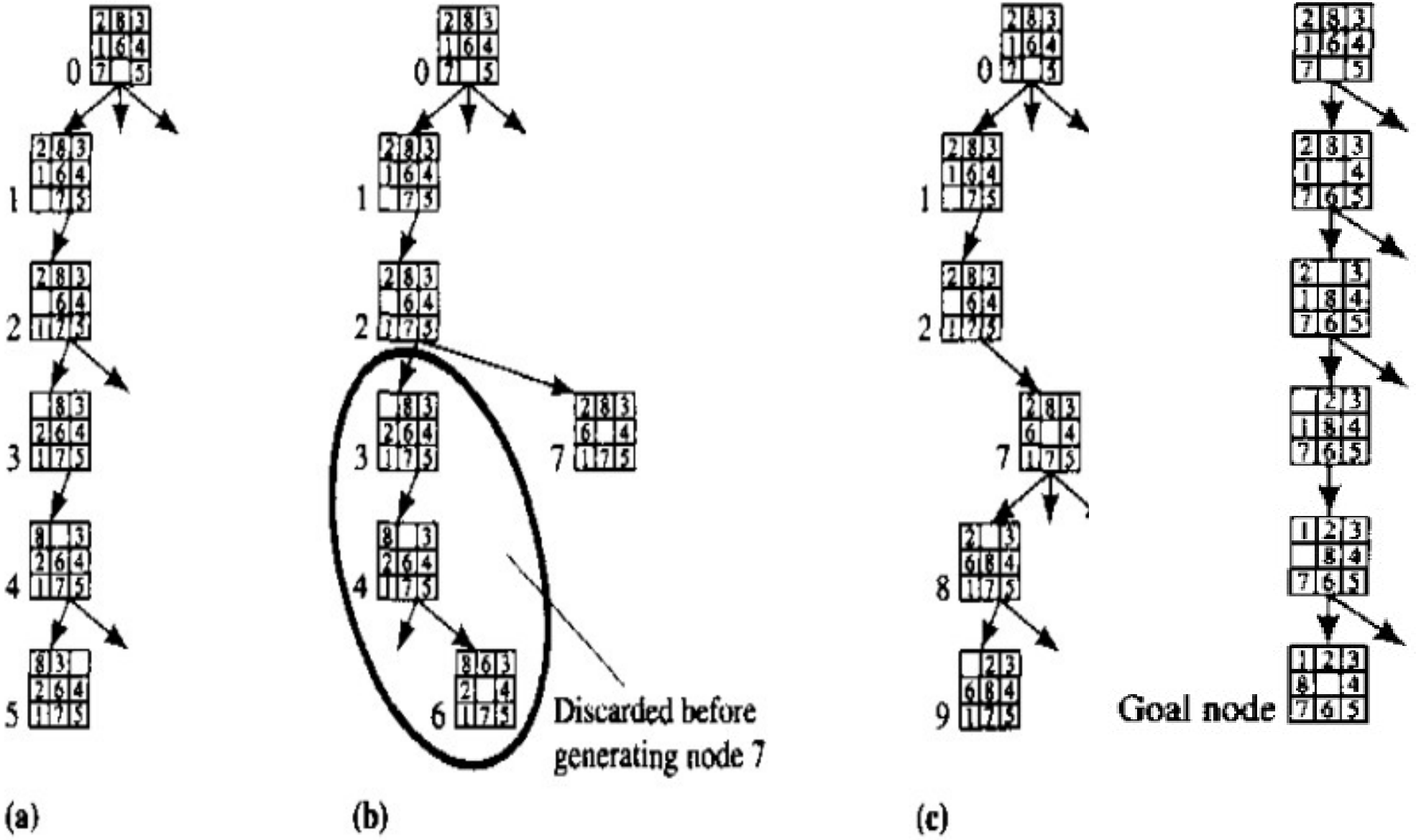




Depth First Search

- 8 Puzzle problem has been considered with depth bound 5
- Followed a specific order of actions to expand the node is move blank left, move blank up, move blank right, move blank down.
- Each move is reversible, so the arc from successors back to their parents are omitted
- The number at the left side of each node shows the order in which the nodes are generated.

Depth First Search



Quiz: Depth First Search

- When will BFS outperform DFS??

- When will DFS outperform BFS??

Search Process

- Search Algorithm
  - Uninformed Search:
    - Where we have no problem specific reason to prefer one part of the search space to any other as far as finding a path to the goal is concerned.
      - Breadth First Search
      - Depth First Search
      - Uniformed Cost Search
  - Informed Search:
    - Heuristic Search
      - Where we have problem specific information to help focus the search
        - Best First Search
    - A* Search

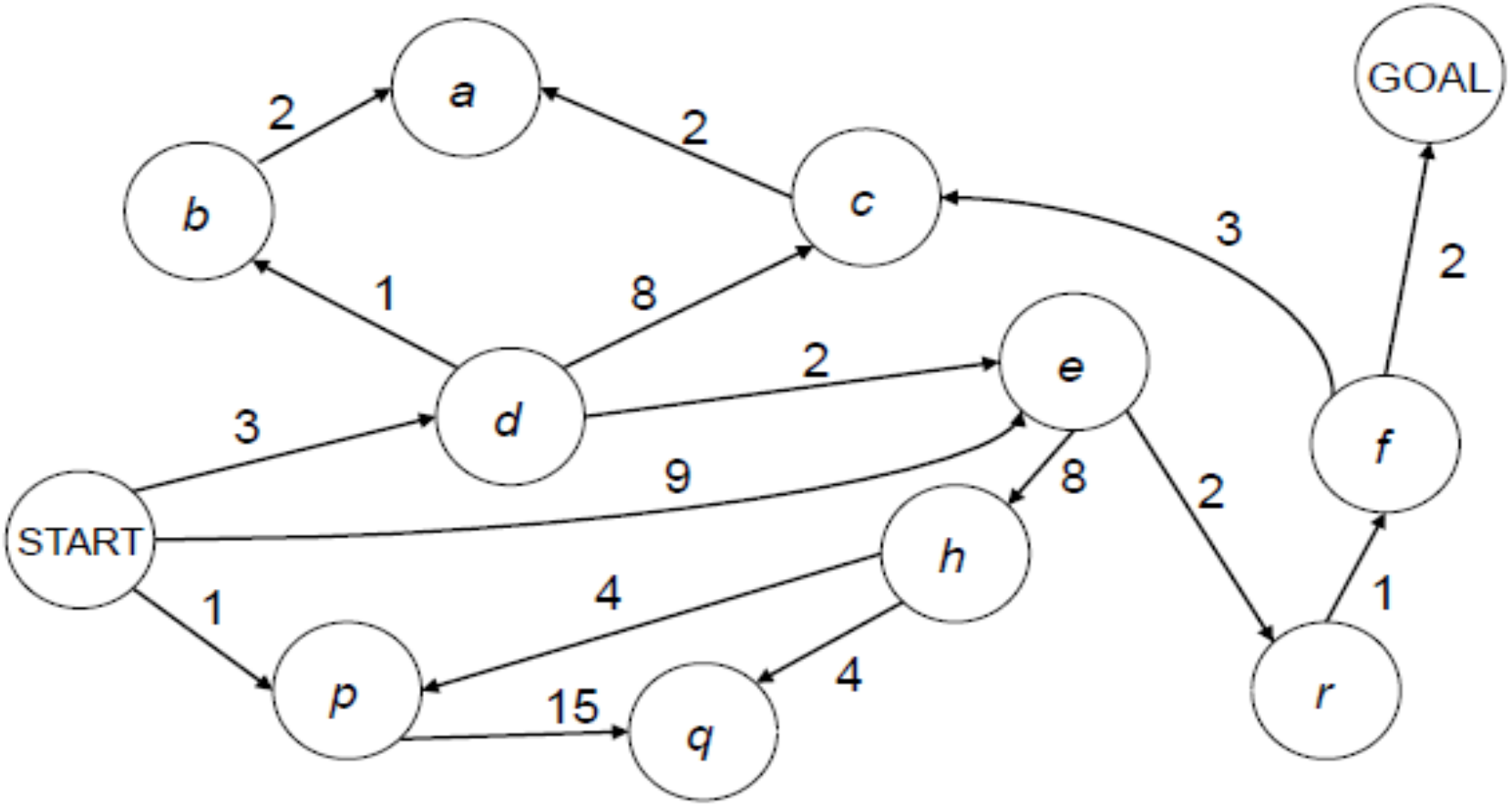
Uniform Cost Search (UCS)

- It is a variant of breadth first search.
- Also called Cheapest First Search/Cost Sensitive Search
- Here the node expansion done outward from the start node along contours of equal cost rather than among contours of equal depth
- If the cost of all arcs in the graph is identical then uniform cost search is same as breadth first search.
- It can be considered to be a special case of heuristic search procedure

Uniform Cost Search (UCS)

- The creation of the tree is not a part of the algorithm. It is just for visualization.
- The algorithm returns a path which is optimal in terms of cost.
- Unlike DFS where the maximum depth had the maximum priority, UCS gives the minimum cumulative cost the maximum priority.

Costs on Actions



Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path.

We will quickly cover an algorithm which does find the least-cost path.

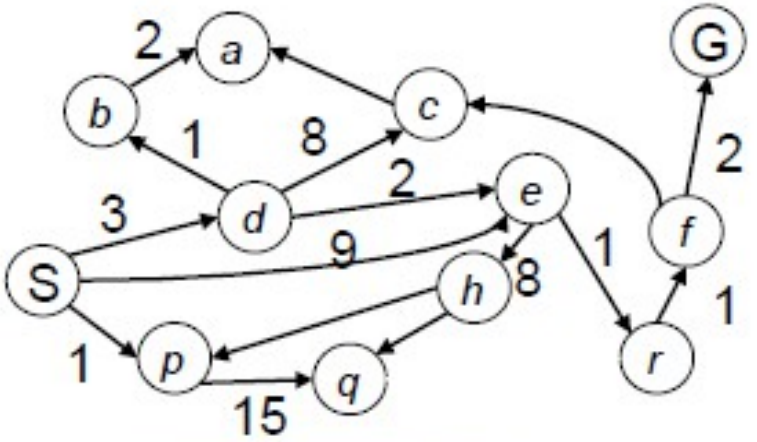
Uniform Cost Search

```
Procedure UniformCostSearch(Graph, root, goal)
    Node := root, cost = 0
    Frontier := priority queue containing node only
    Explored := empty set
    Do
        If frontier is empty
            Return failure
        Node := frontier.pop()
        If node is goal
            Return solution
        Explored.add(node)
        For each of node's neighbors n
            If n is not in explored
                If n is not in frontier
                    Frontier.add(n)
                Else if n is in frontier with higher cost
                    Replace existing node with n
```

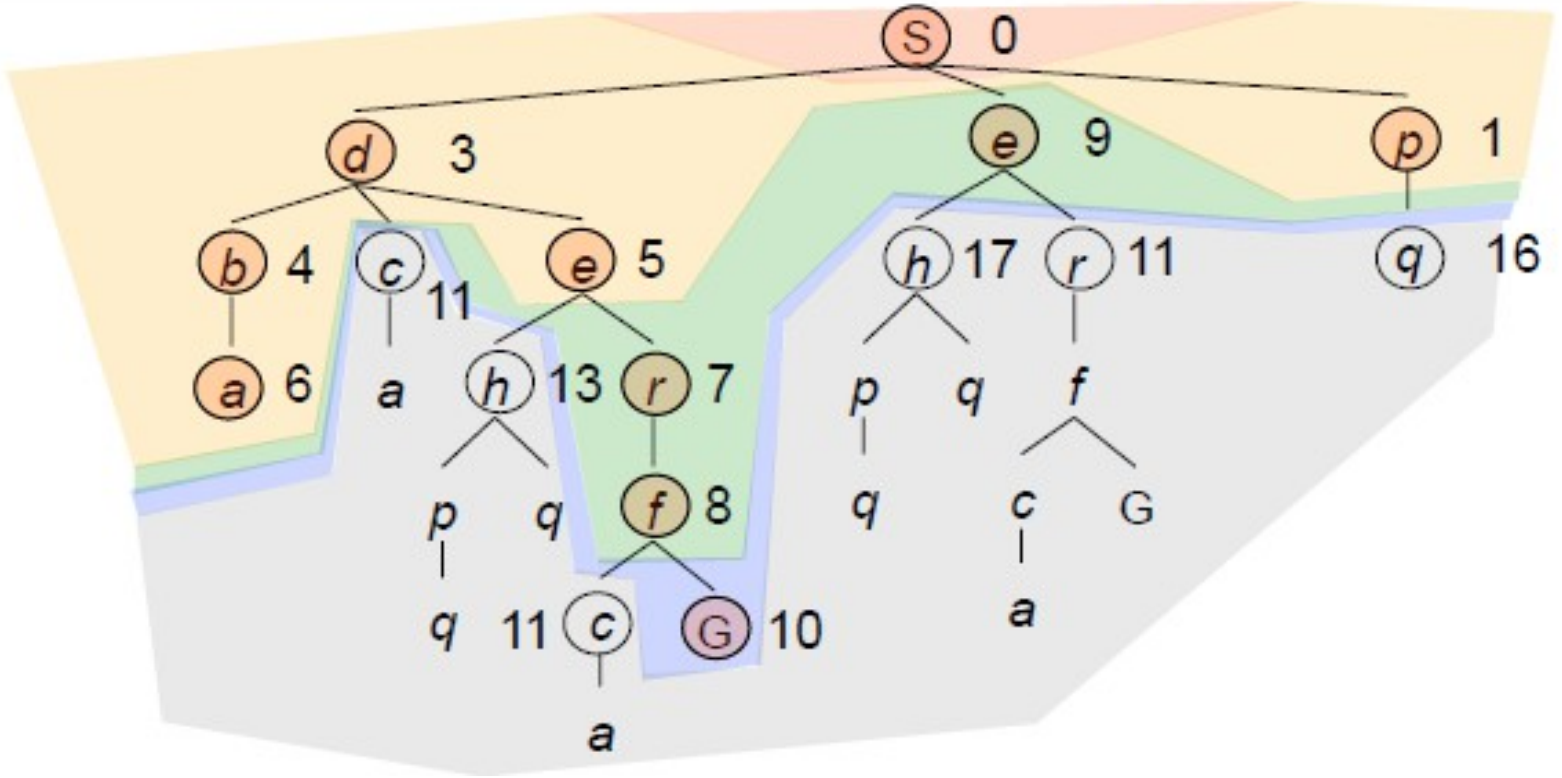
Uniform Cost Search

Expand cheapest node first:

Fringe is a priority queue

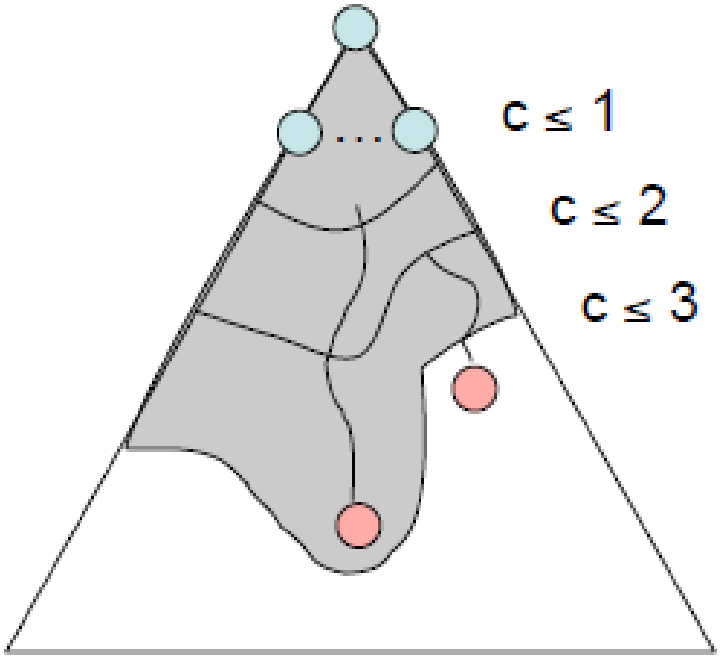


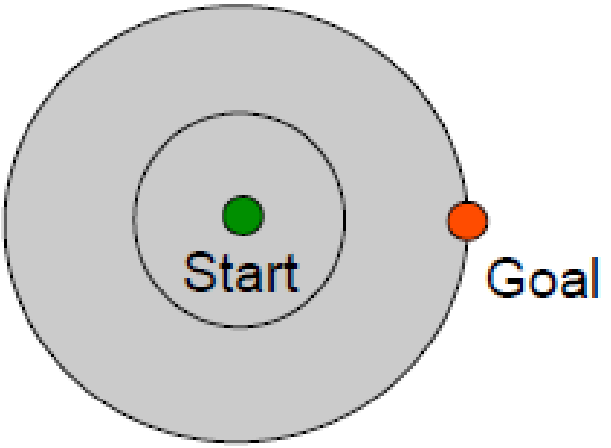
Cost contours



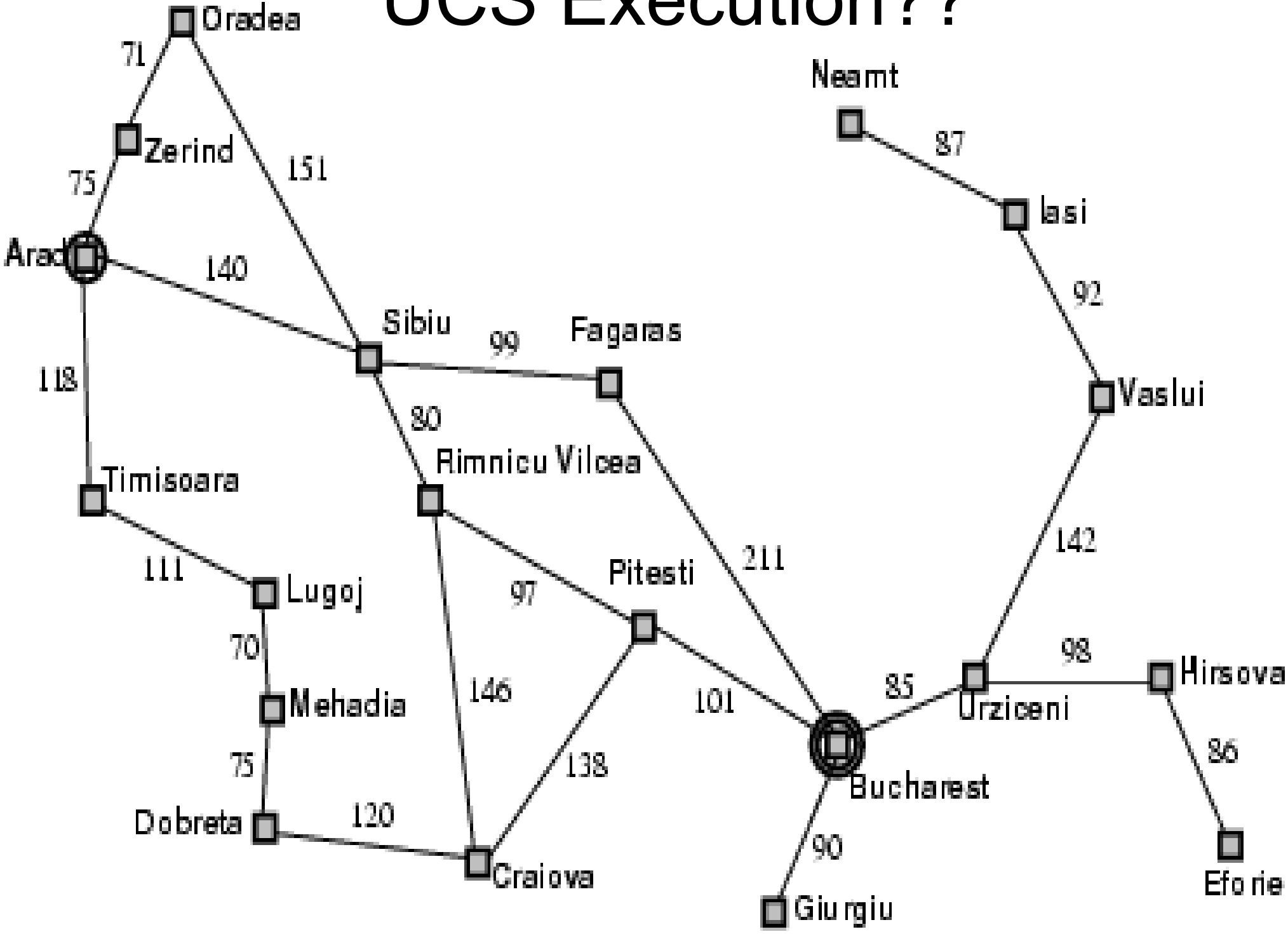
Uniform Cost Issues

- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location



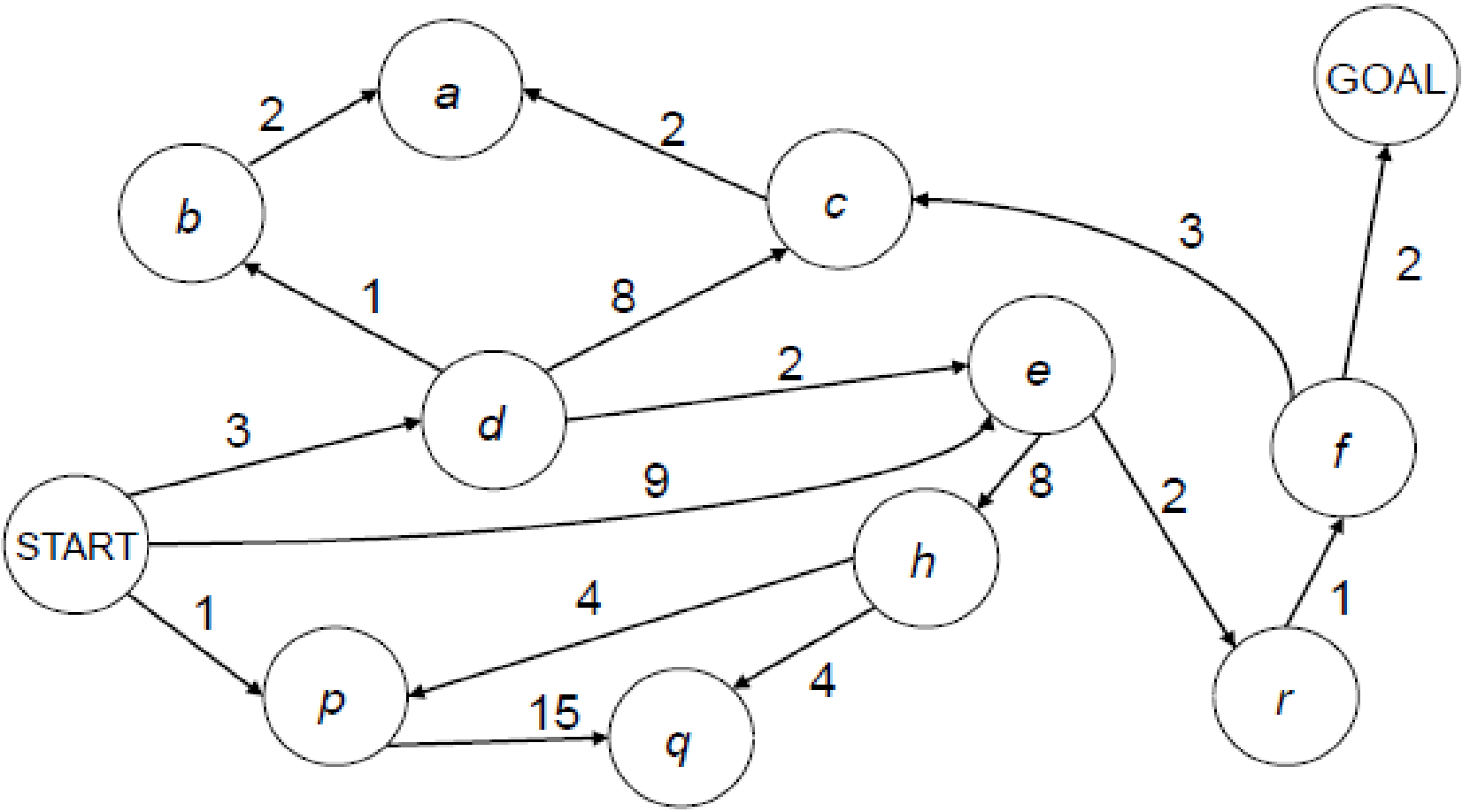


UCS Execution??



Execution

- Iteration in terms of [Path, Cumulative Cost]??



Search Process

- Search Algorithm
  - Uninformed Search:
    - Where we have no problem specific reason to prefer one part of the search space to any other as far as finding a path to the goal is concerned.
      - Breadth First Search
      - Depth First Search
      - Uniformed Cost Search
  - **Informed Search:**
    - **Heuristic Search**
      - **Where we have problem specific information to help focus the search**
        - **Best First Search**
    - **A* Search**