ARTIFICIAL INTELLIGENCE

SEARCH ALGORITHMS

CS-632

University Institute of Information Technology

PMAS-Arid Agriculture University Rawalpindi

TWO MAIN TYPES OF SEARCH

All search strategies are distinguished by the order in which nodes are expanded.

1. Uninformed search methods (Blind search) have access only to the problem definition.
   - Breadth-first search
   - Uniform-cost search
   - Depth-first search
   - Iterative deepening search
   - Bidirectional search
2. Informed search methods may have access to a heuristic function *h(n)* that estimate the cost of a solution from *n*.
   - Greedy best-first search
   - A* search
   - Recursive best-first search (RBFS) search

UNINFORMED SEARCH

- Sometimes we may not get much relevant information to solve a problem.
- Suppose we lost our car key, and we are not able to recall where we left. In this case, we need to search for the key with some information. Such as, in which places we used to place it.
- It may be our pant pocket or may be the table drawer.
- If it is not there, then we will search the whole house to get it.
- The best solution would be to search in the places from the table to the wardrobe. Here we need to search blindly with less clue.
- This type of search is called uninformed search or blind search.

UNINFORMED SEARCH

- Blind search – BFS, DFS, uniform cost
  - no notion concept of the “right direction”
  - can only recognize goal once it’s achieved

INFORMED SEARCH

- We can solve the problem in an efficient manner if we have relevant information, clues or hints. The clues that help solve the problem constitute heuristic information.
- Informed search is also called heuristic search.
- Instead of searching one path or many paths just like that informed search uses the given heuristic information to decide whether to explore the current state further.

INFORMED SEARCH

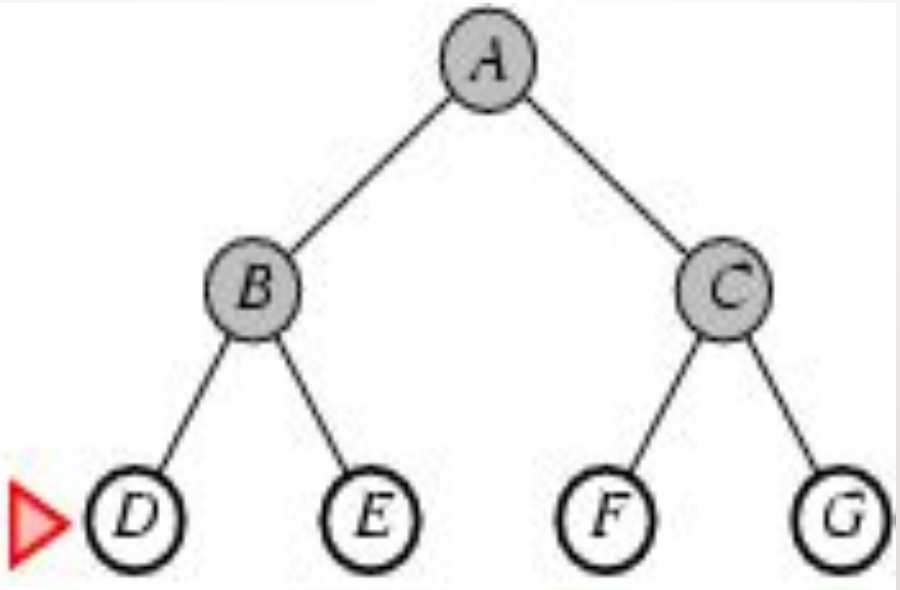
- Add domain-specific information to select the best path along which to continue searching
- Define a heuristic function, ***h(n)***, that estimates the "goodness" of a node ***n***.
- Specifically, ***h(n)*** **= estimated cost** (or distance) of minimal cost path from ***n*** to a goal state.
- The heuristic function is an estimate, based on domain-specific information that is computable from the current state description, of how close we are to a goal.

BREADTH-FIRST SEARCH

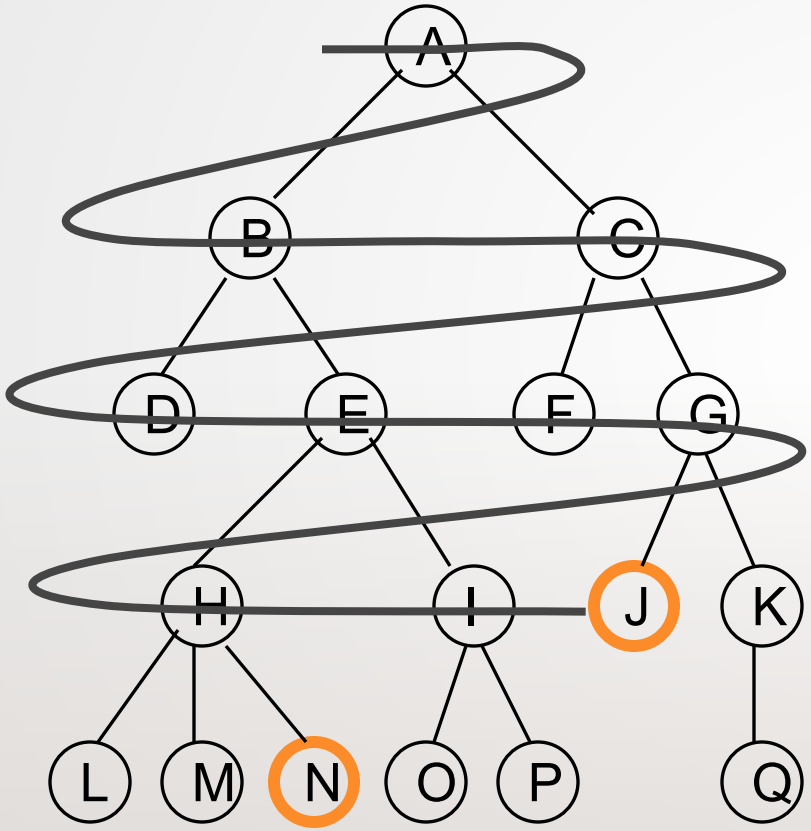
- Breadth first search (BFS), as the name implies, searches from the initial state breadth-wise.
- That is, it searches all the states in the tree level by level.
- Only after exploring all the states in one level, it will jump to the next level.
- Once the solution is found the search stops.
- BFS guarantees to find the solution if it exists.

BREADTH-FIRST SEARCH

- Expand shallowest unexpanded node
- Implementation:



BREADTH-FIRST SEARCH

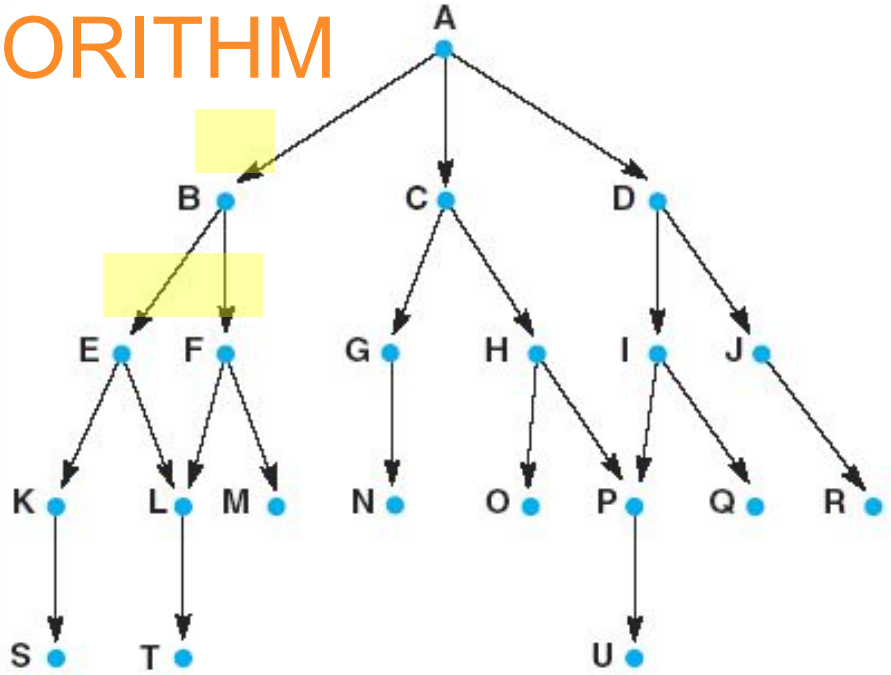


- A breadth-first search (BFS) explores nodes nearest the root before exploring nodes further away
- For example, after searching A, then B, then C, the search proceeds with D, E, F, G
- Node are explored in the order A B C D E F G H I J K L M N O P Q
- J will be found before N

BREADTH-FIRST SEARCH ALGORITHM

```
function breadth_first_search;
                          Nodes are lining up to be visited in open.
begin                     closed keeps track of all the nodes visited already.
   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;   % loop check
                  put remaining children on right end of open           % queue
               end
      end
   return FAIL                                                       % no states left
end.
```

A TRACE OF BREADTH-FIRST ALGORITHM



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 goal is reached or open is empty.

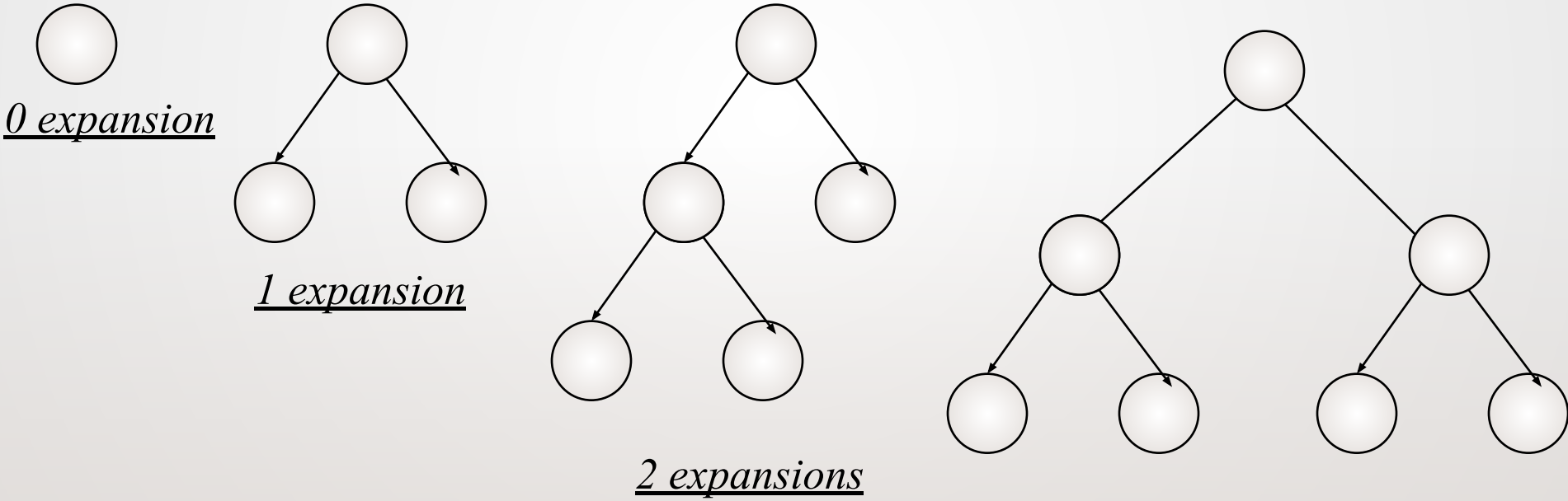
B is not the goal.
Put his children onto the queue.
Put him in closed. He is done.

Items between red bars are siblings.

BREADTH-FIRST SEARCH TREE

Branching factor: *number of nodes generated by a node parent (we called here "b")*

- *Here after b=2*

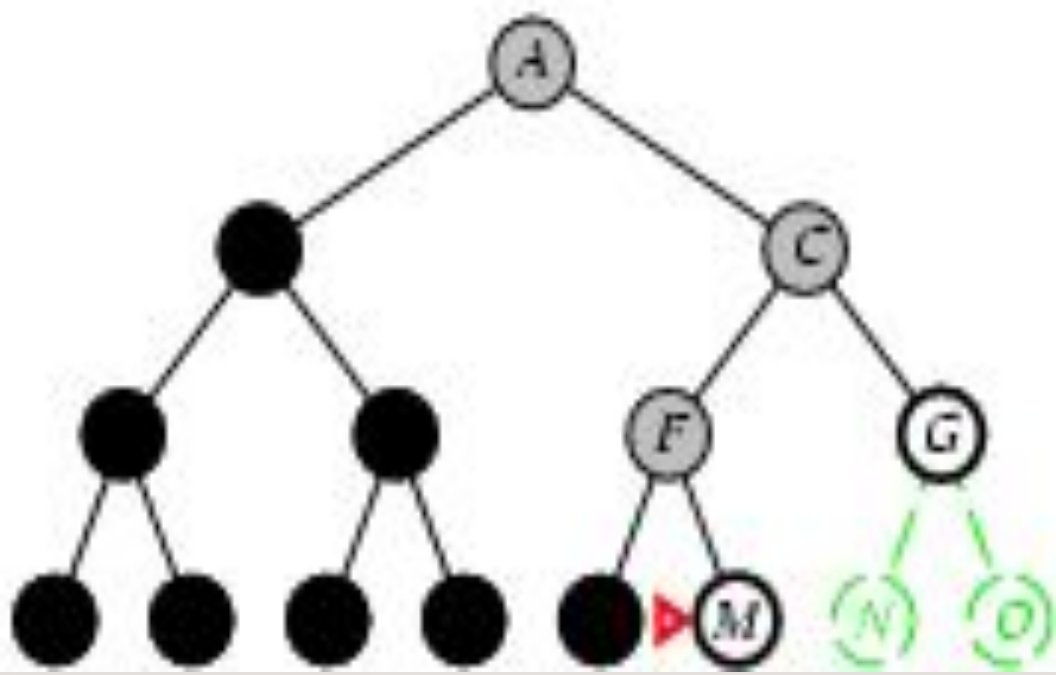


DEPTH-FIRST SEARCH

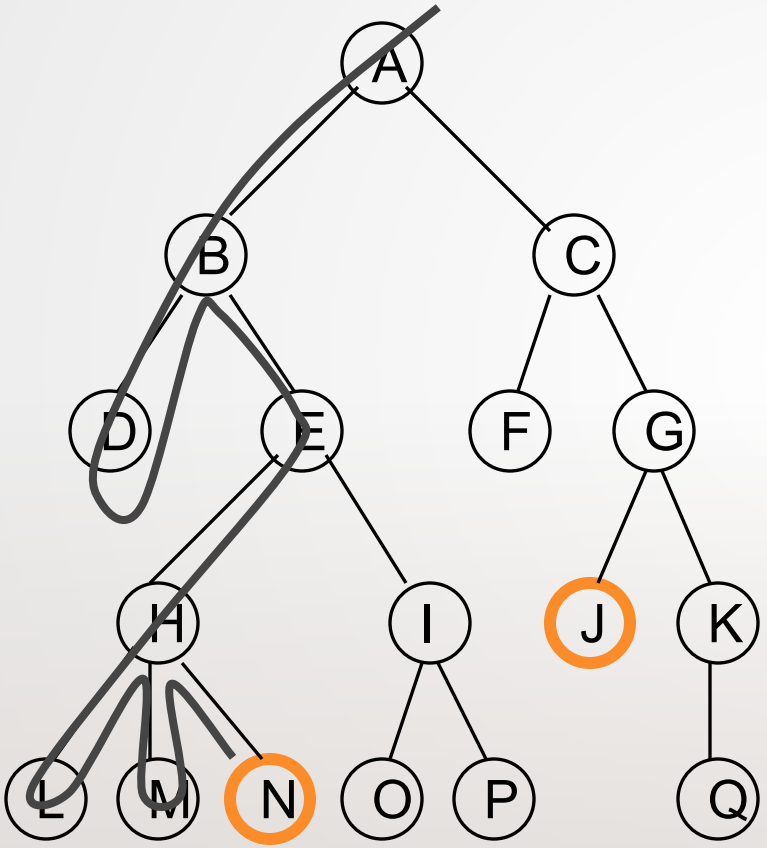
- Explore only one branch deeper till the solution is found or there is no new state to explore and then start searching the adjacent level.
- If the solution exists in the first branch, then depth first search can find the solution quickly.
- If the solution exists in some other branches farther away, there won't be much difference between depth first search and breadth first search.

DEPTH-FIRST SEARCH

- Expand deepest unexpanded node
- Implementation:



DEPTH-FIRST SEARCH



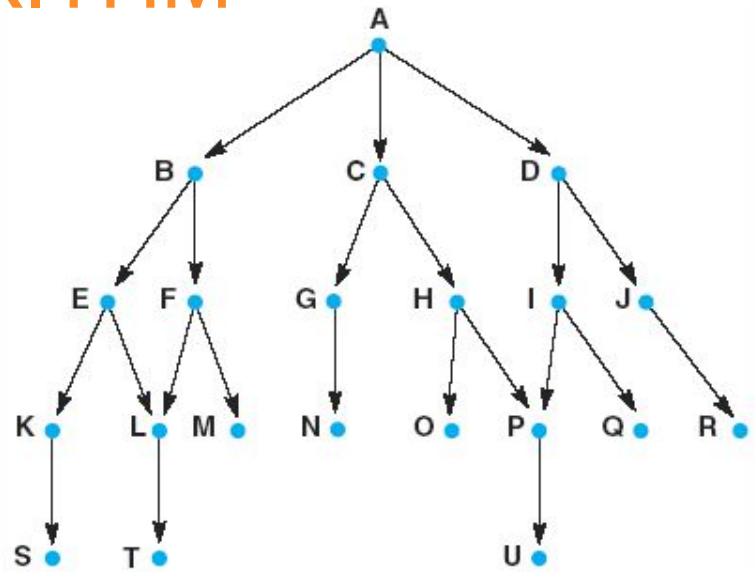
- A depth-first search (DFS) explores a path all the way to a leaf before backtracking and exploring another path
- For example, after searching A, then B, then D, the search backtracks and tries another path from B
- Node are explored in the order A B D E H L M N I O P C F G J K Q
- N will be found before J

The 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;   % loop check
               put remaining children on left end of open              % stack
            end
      end;
   return FAIL                                                   % no states left
end.
```

This is the only difference between depth-first and breadth-first.

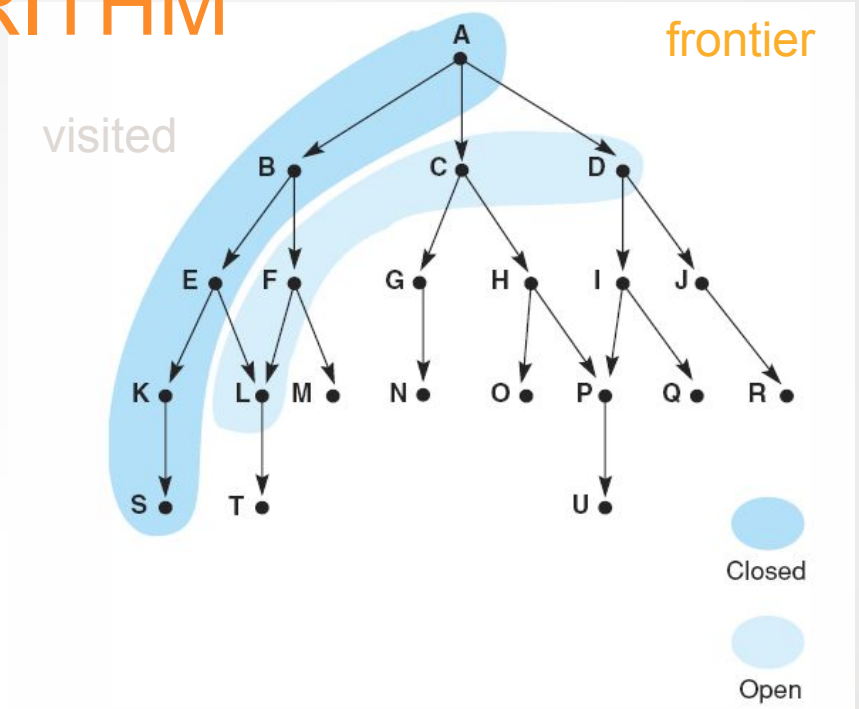
A TRACE OF DEPTH-FIRST ALGORITHM



top of stack

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]

A TRACE OF DEPTH-FIRST ALGORITHM



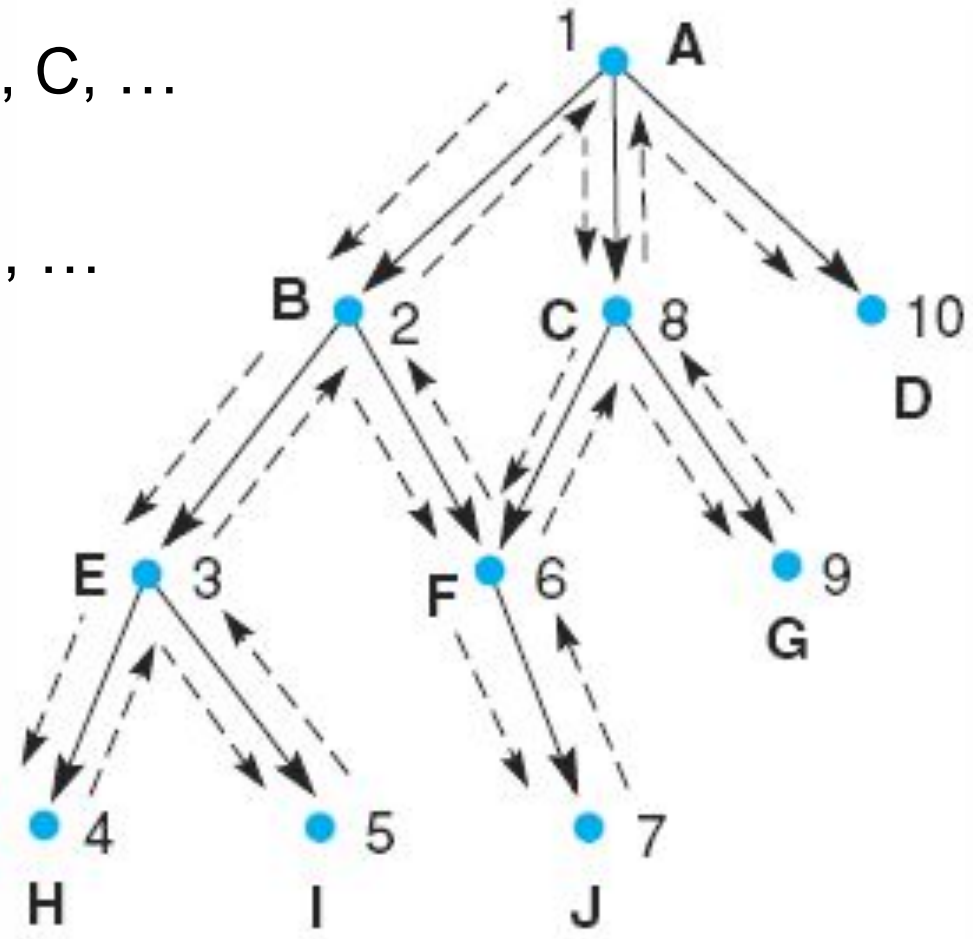
Snapshot at iteration 6

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]**

SEARCH SEQUENCE OF DEPTH-FIRST AND BREADTH-FIRST

Breadth first: A, B, C, …

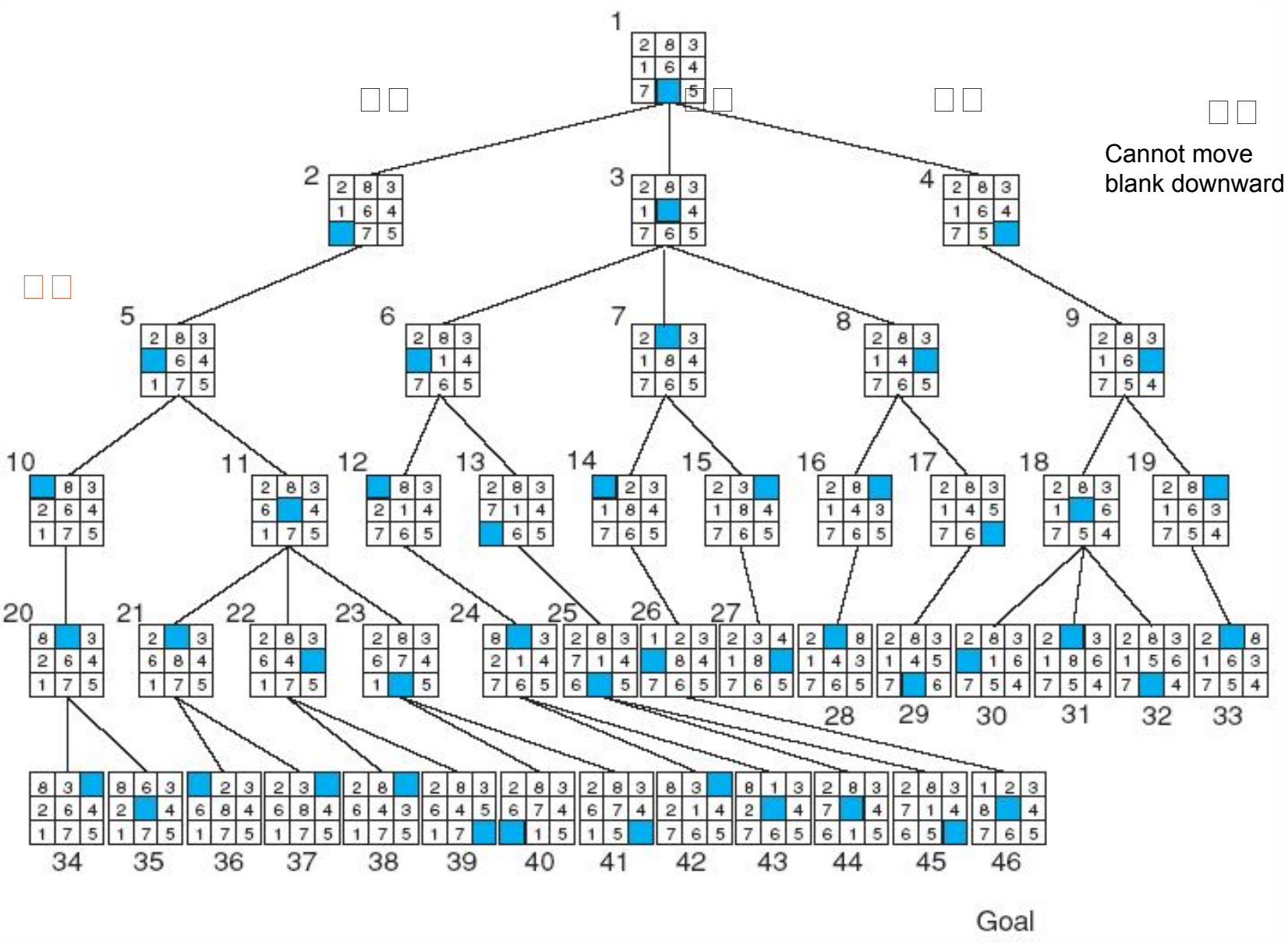
Depth first: 1, 2, 3, …



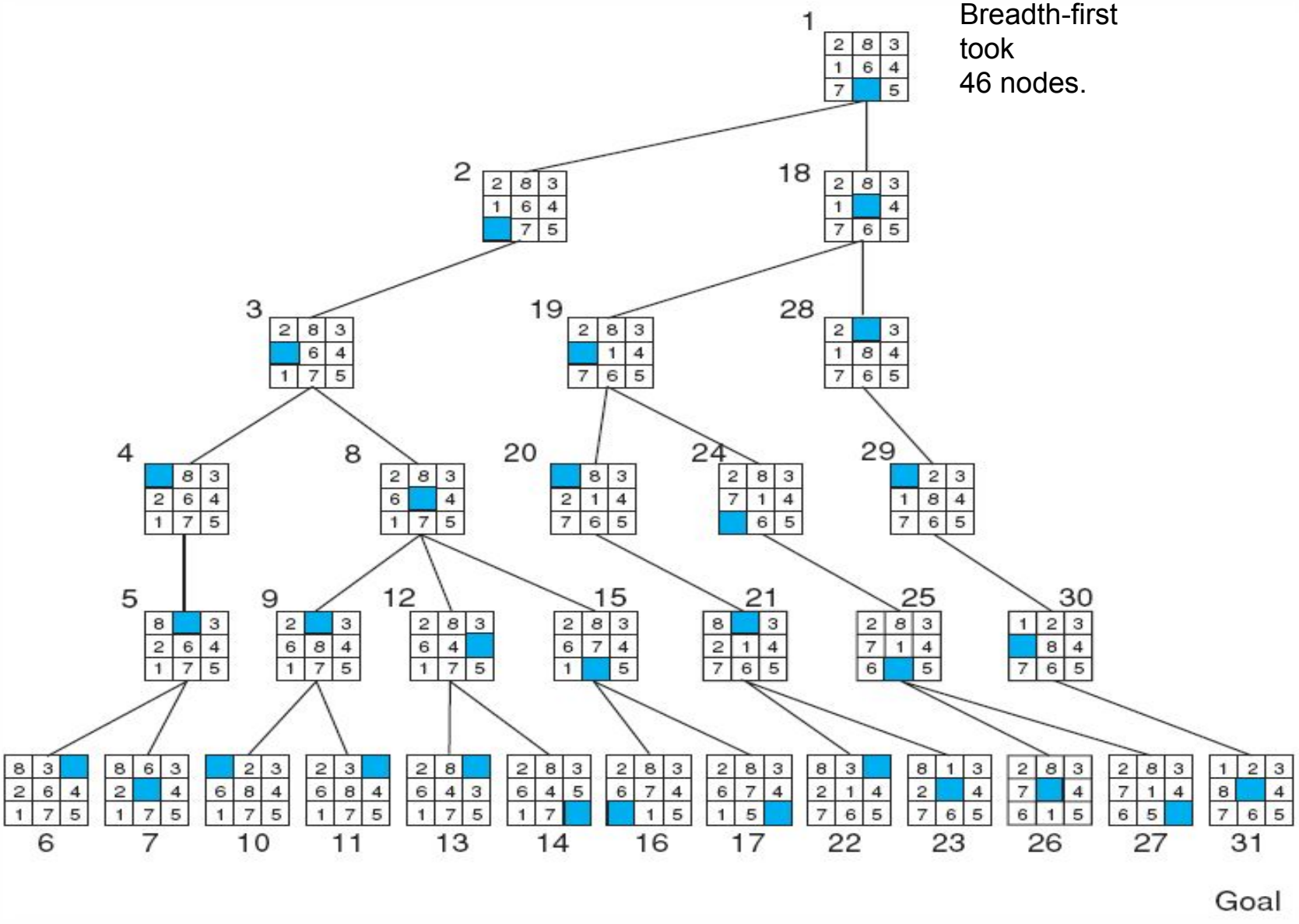
COMPARING THE ORDERING OF SEARCH SEQUENCES

- Determine the order of nodes (states) to be examined
- Breadth-first Search
  - When a state is examined, all its children are examined, one after another.
  - Explore the search space in a level-by-level fashion.
- Depth-first search
  - When a state is examined, all its children and their descendants are examined before any of its siblings.
  - Go deeper into the search space where possible.

APPLICATION OF BREADTH-FIRST ON 8-PUZZLE GAME



APPLICATION OF DEPTH-FIRST ON 8-PUZZLE (DEPTH BOUND 5)



Thank You