

# Algoritma Pencarian Blind

Breadth First Search

Depth First Search

# Deskripsi

- Merupakan algoritma untuk mencari kemungkinan penyelesaian
- Sering dijumpai oleh peneliti di bidang AI

# Mendefinisikan permasalahan

- Mendefinisikan suatu state (ruang keadaan)
- Menerapkan satu atau lebih state awal
- Menetapkan satu atau lebih state tujuan
- Menetapkan rules (kumpulan aturan)

# A unifying view (Newell and Simon)

- Ruang masalah terdiri dari:
  - *state space* adalah himpunan state yang mungkin dari suatu permasalahan
  - Himpunan *operators* digunakan untuk berpindah dari satu state ke state yang lain.
  - Ruang masalah dapat digambarkan dengan graph, himpunan state dinyatakan dengan node dan link (arcs) menyatakan operator.

# Contoh 1

Seorang petani ingin memindah dirinya sendiri, seekor serigala, seekor angsa gemuk, dan seikat padi yang berisi menyeberangi sungai. Sayangnya, perahunya sangat terbatas; dia hanya dapat membawa satu objek dalam satu penyeberangan. Dan lagi, dia tidak bisa meninggalkan serigala dan angsa dalam satu tempat, karena serigala akan memangsa angsa. Demikian pula dia tidak bisa meninggalkan angsa dengan padi dalam satu tempat.

# State (ruang keadaan)

- State  $\rightarrow$  (Serigala, Angsa, Padi, Petani)
- Daerah asal ketika hanya ada serigala dan padi, dapat direpresentasikan dengan state  $(1, 0, 1, 0)$ , sedangkan daerah tujuan adalah  $(0, 1, 0, 1)$

# State awal dan tujuan

- State awal
  - Daerah asal  $\rightarrow (1, 1, 1, 1)$
  - Daerah tujuan  $\rightarrow (0, 0, 0, 0)$
- State tujuan
  - Daerah asal  $\rightarrow (0, 0, 0, 0)$
  - Daerah tujuan  $\rightarrow (1, 1, 1, 1)$

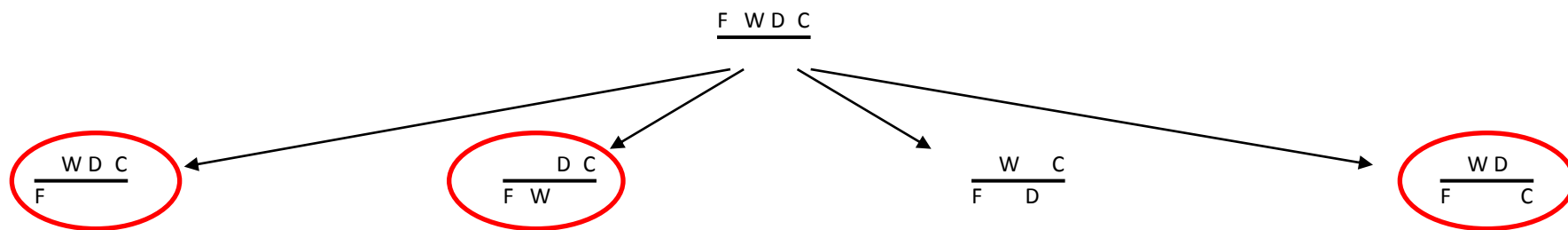
# Rules

| Aturan ke | Rule                                |
|-----------|-------------------------------------|
| 1         | Angsa menyeberang bersama petani    |
| 2         | Padi menyeberang bersama petani     |
| 3         | Serigala menyeberang bersama petani |
| 4         | Angsa kembali bersama petani        |
| 5         | Padi kembali bersama petani         |
| 6         | Serigala kembali bersama petani     |
| 7         | Petani kembali                      |



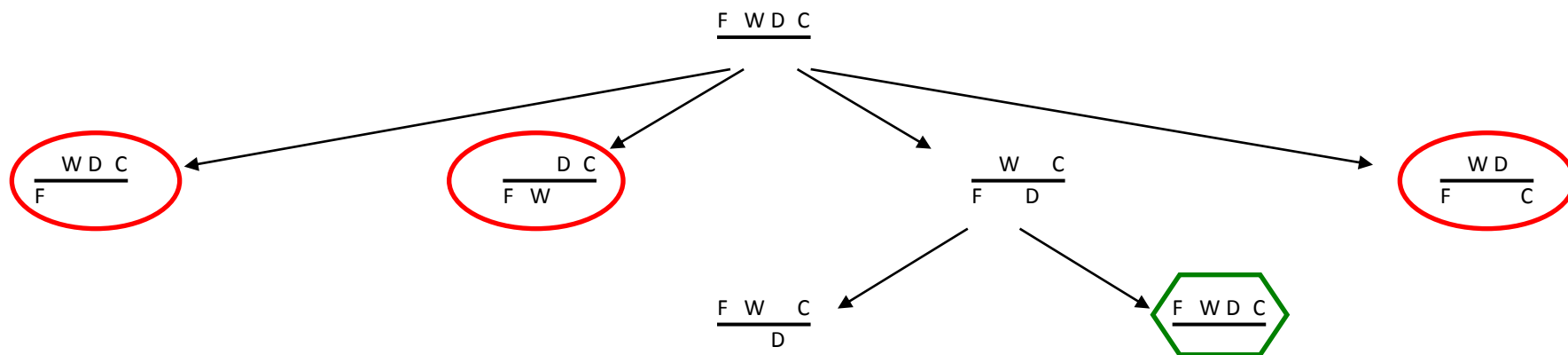
# Contoh solusi

| Daerah asal<br>(S, A, Pd, Pt) | Daerah tujuan<br>(S, A, Pd, Pt) | Rule yang<br>dipakai |
|-------------------------------|---------------------------------|----------------------|
| (1, 1, 1, 1)                  | (0, 0, 0, 0)                    | 1                    |
| (1, 0, 1, 0)                  | (0, 1, 0, 1)                    | 7                    |
| (1, 0, 1, 1)                  | (0, 1, 0, 0)                    | 3                    |
| (0, 0, 1, 0)                  | (1, 1, 0, 1)                    | 4                    |
| (0, 1, 1, 1)                  | (1, 0, 0, 0)                    | 2                    |
| (0, 1, 0, 0)                  | (1, 0, 1, 1)                    | 7                    |
| (0, 1, 0, 1)                  | (1, 0, 1, 0)                    | 1                    |
| (0, 0, 0, 0)                  | (1, 1, 1, 1)                    | solusi               |



Search Tree for "Farmer, Wolf, Duck, Corn"

Illegal State

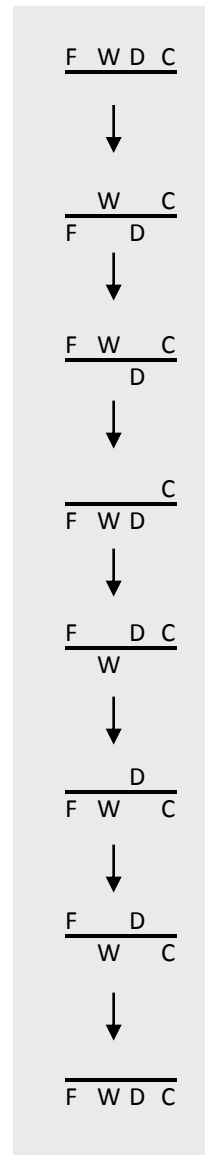
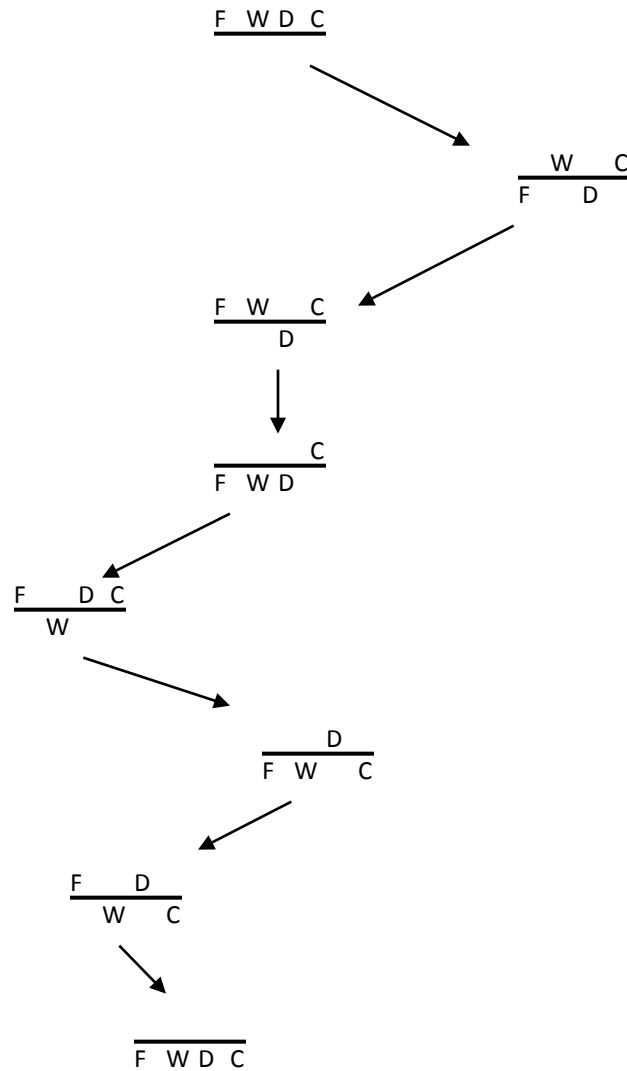


Search Tree for "Farmer, Wolf, Duck, Corn"

Illegal State

Repeated  
State





Initial State

Farmer takes duck to left bank

Farmer returns alone

Farmer takes wolf to left bank

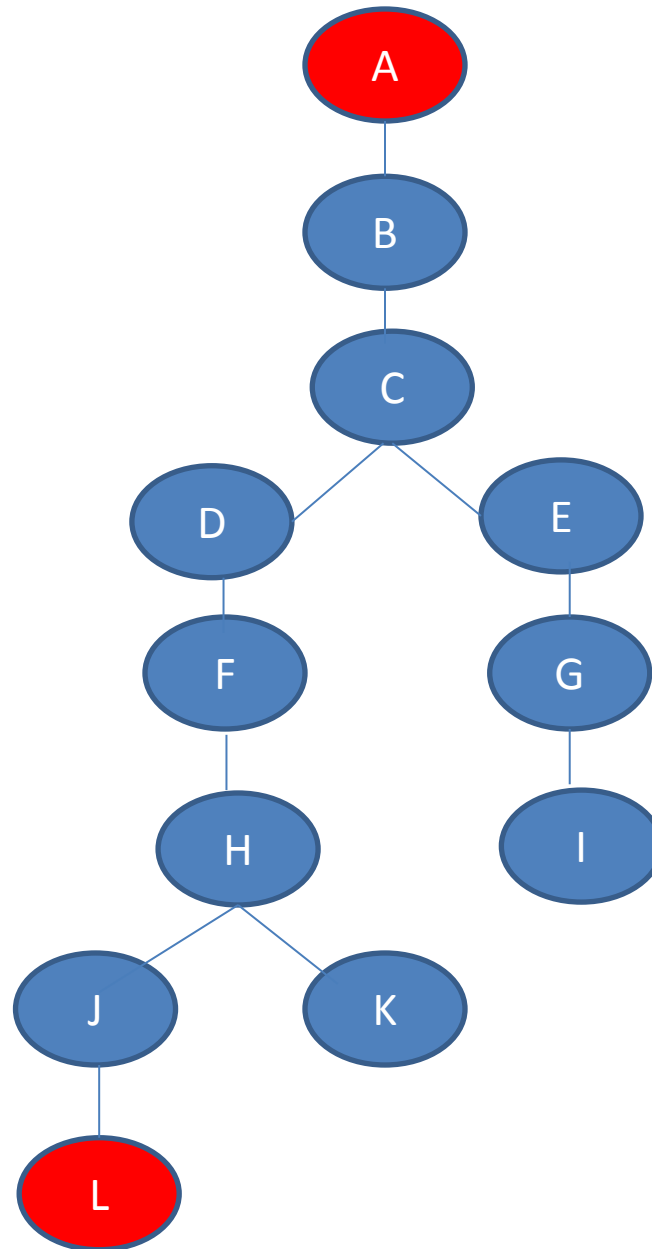
Farmer returns with duck

Farmer takes corn to left bank

Farmer returns alone

Farmer takes duck to left bank

Success!



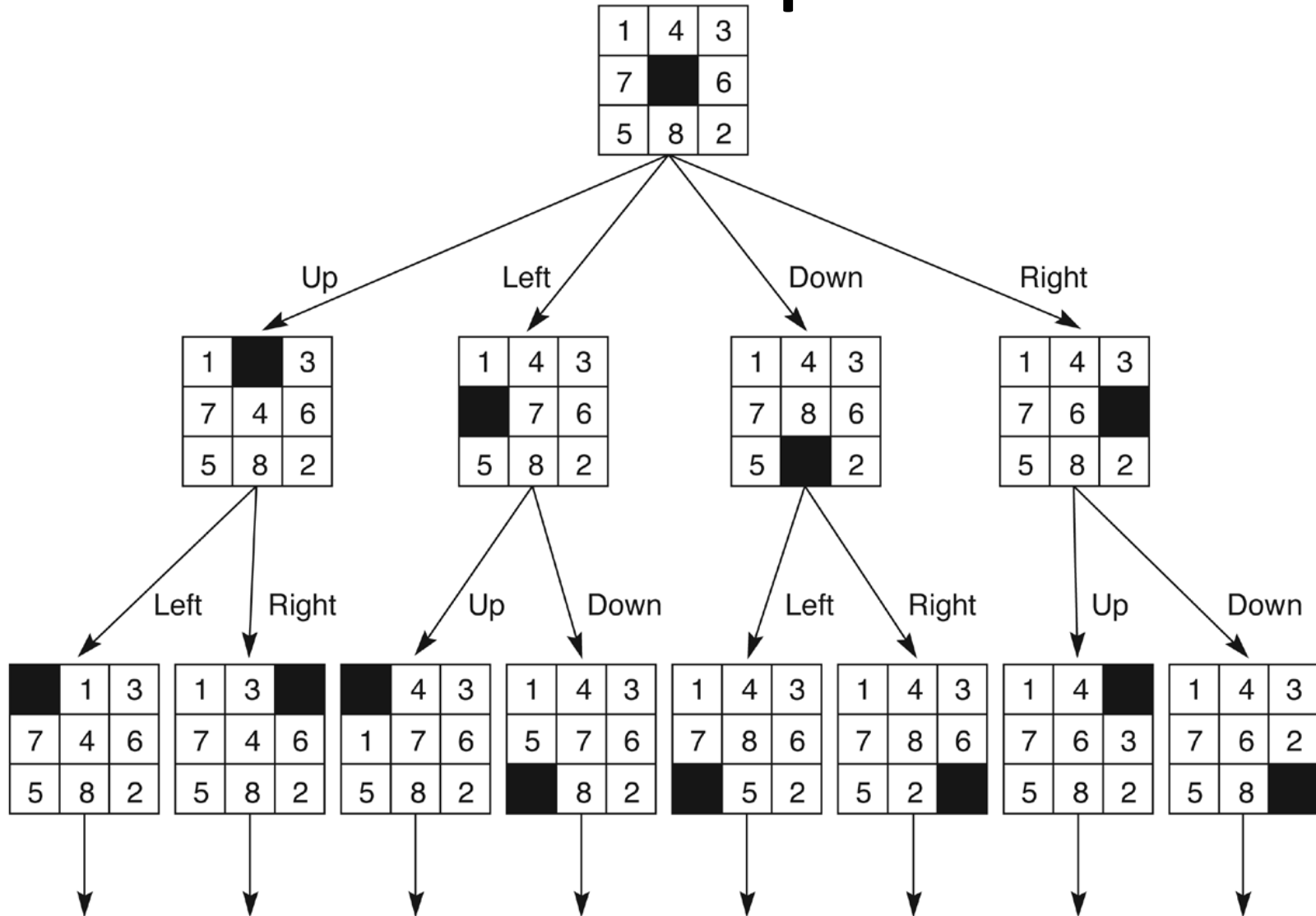
Node A = Node awal  
Node L = Node Akhir

## Contoh 2 – Eight Puzzle

|   |   |   |
|---|---|---|
| 1 | 4 | 3 |
| 7 |   | 6 |
| 5 | 8 | 2 |

|   |   |   |
|---|---|---|
| 1 | 4 | 3 |
| 7 | 6 | 2 |
| 5 | 8 |   |

# State space of the 8-puzzle generated by “move blank” operations





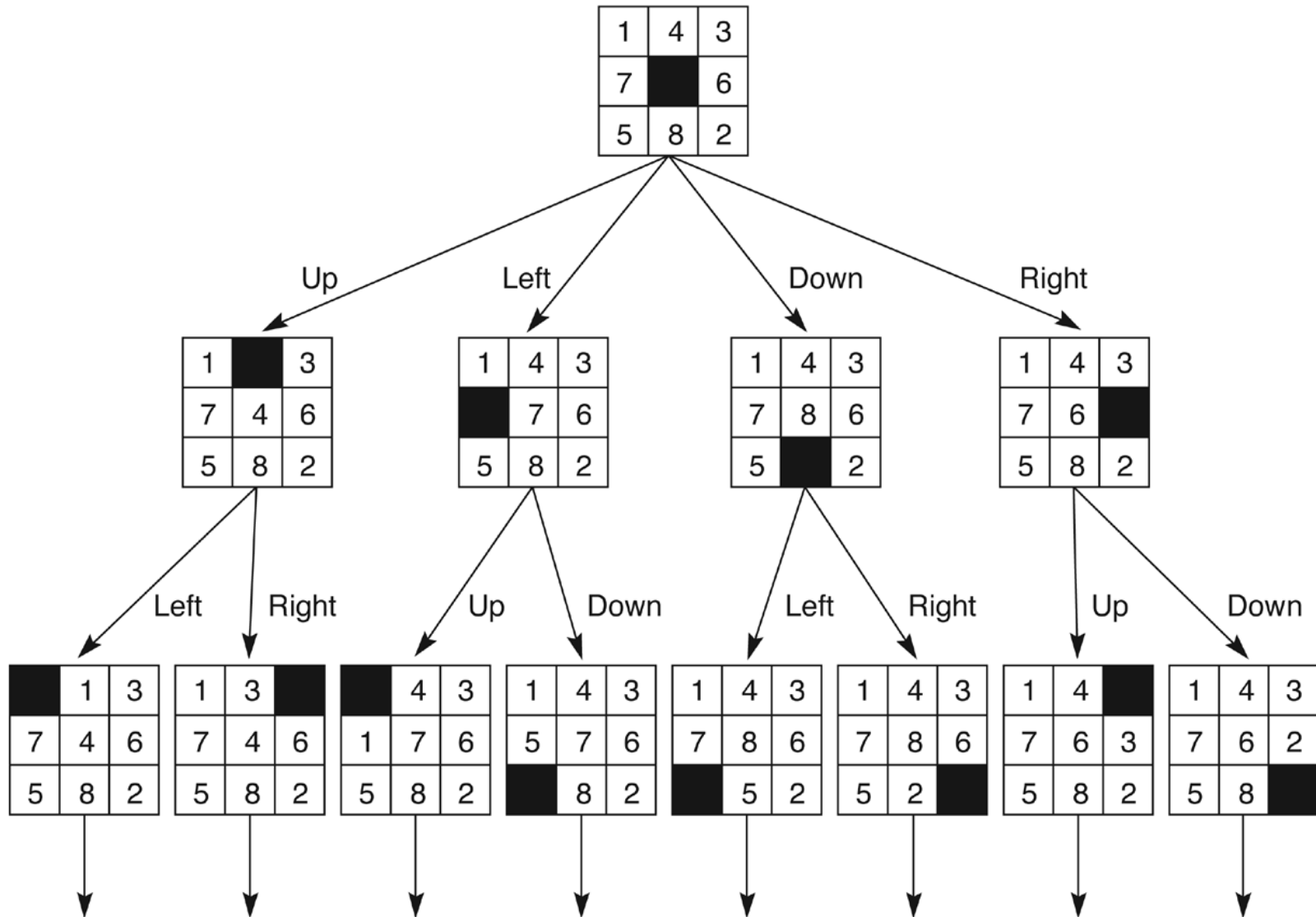
# The 8-puzzle problem as state space search

- State : posisi board yang legal
- Operator : up, left, down, right
- Initial state (state awal) : posisi board yang telah ditentukan
- Goal state (state akhir) : posisi board yang telah ditentukan

Catatan :

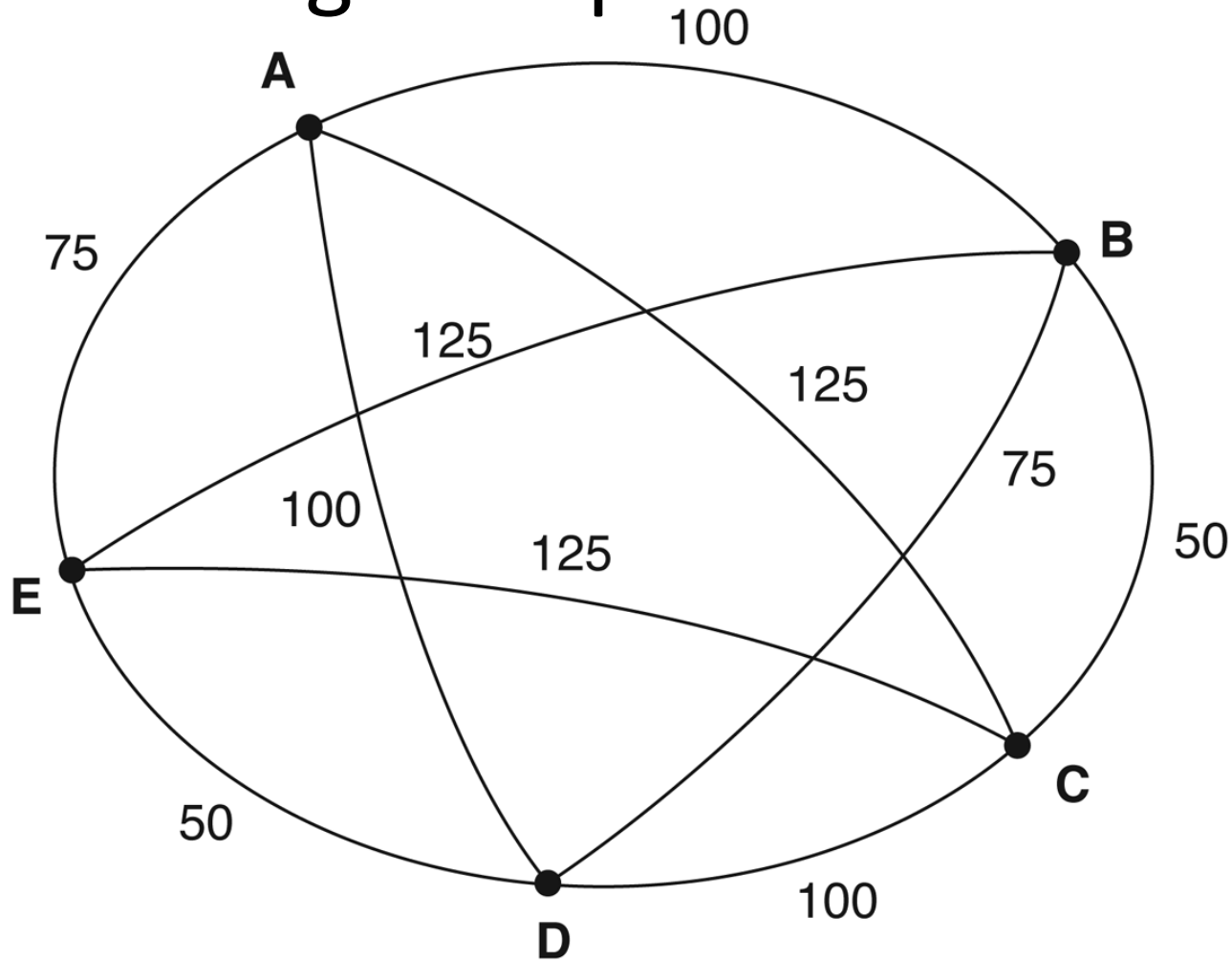
Yang ditekankan disini bukan solusi dari 8-puzzle, tapi lintasan/path dari state awal ke state tujuan.

# State space of the 8-puzzle (repeated)



# Contoh 3

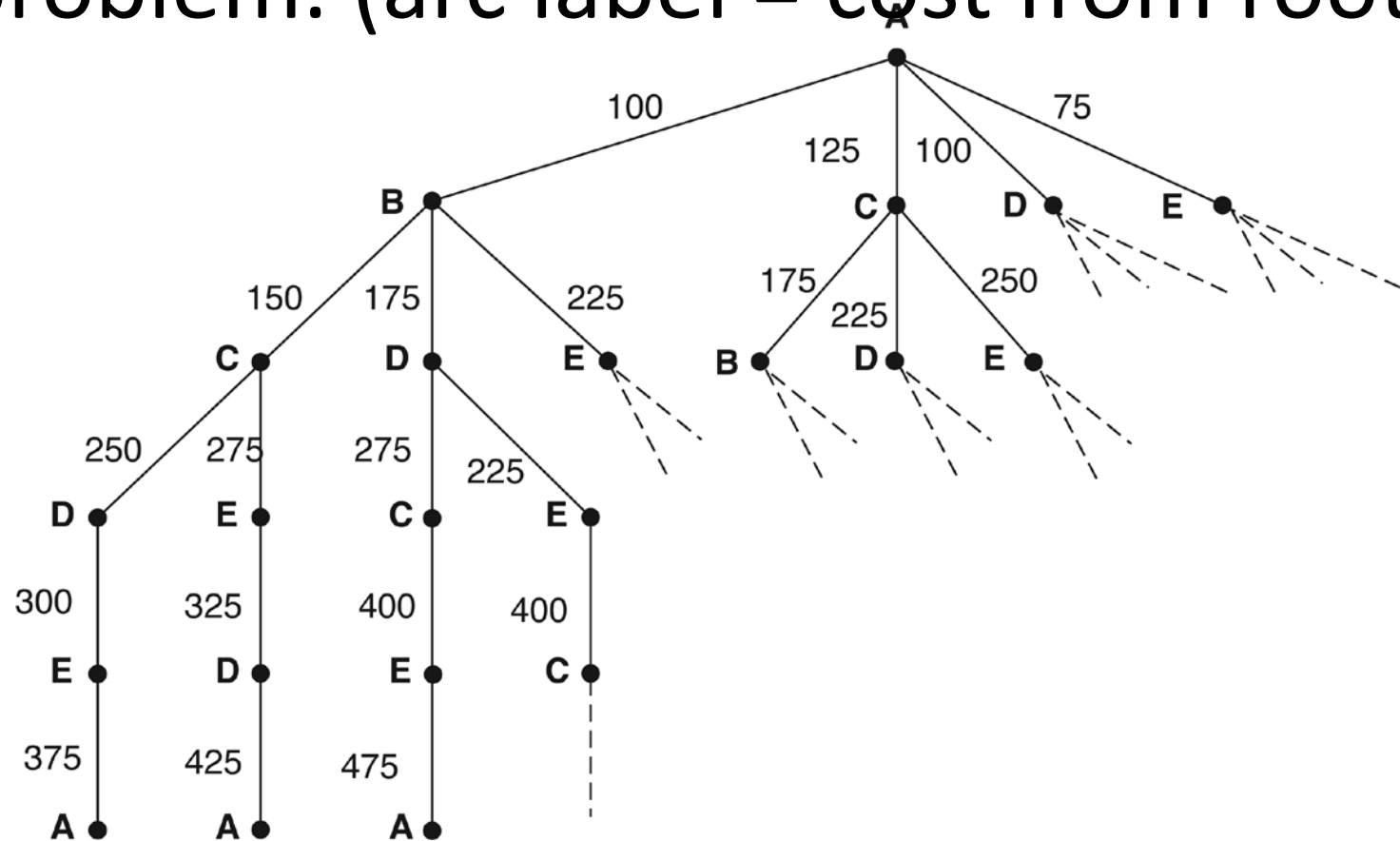
## Traveling Salesperson Problem



## Traveling salesperson problem as state space search

- The salesperson has  $n$  cities to visit and must then return home. Find the shortest path to travel.
- state space:
- operators:
- initial state:
- goal state:

# Search of the traveling salesperson problem. (arc label = cost from root)



Path:  
ABCDEA

Cost:  
375

Path:  
ABCEDA

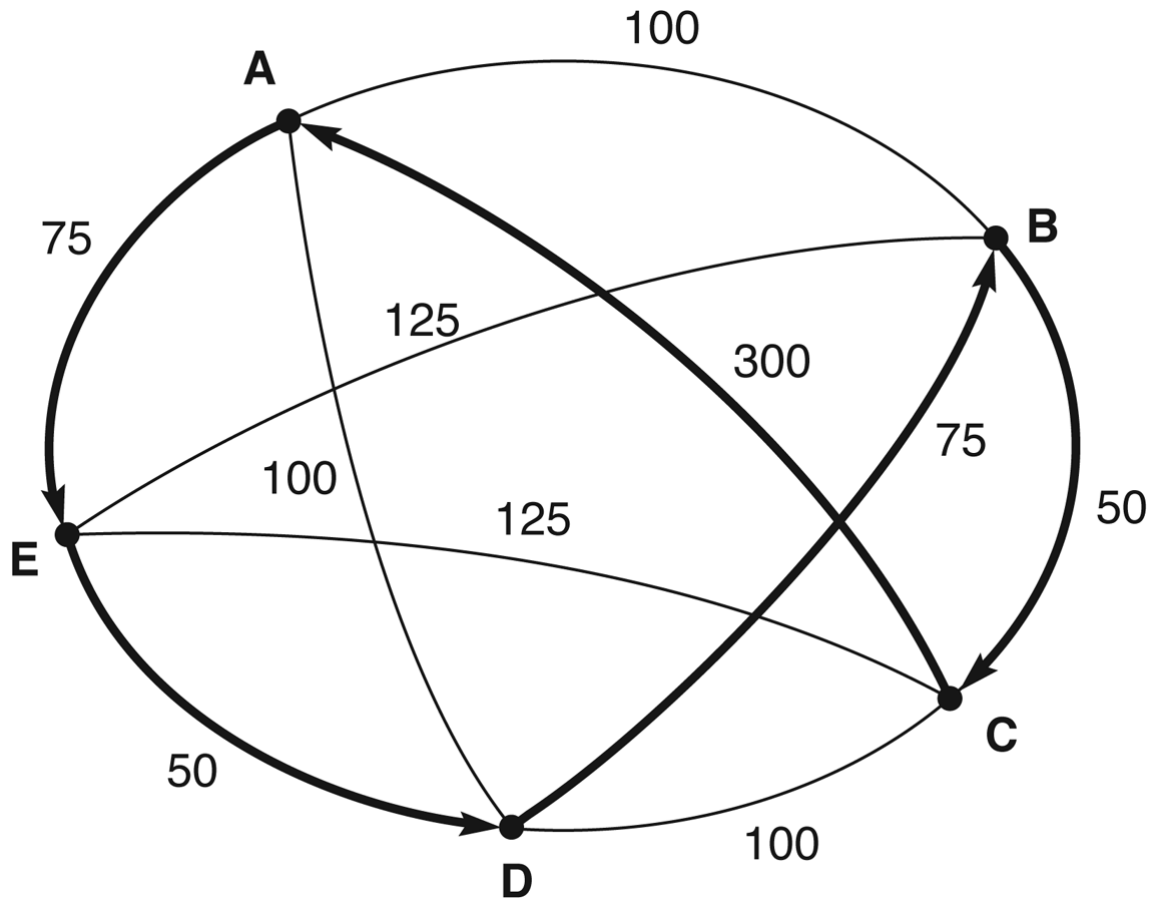
Cost:  
425

Path:  
ABDCEA

Cost:  
475

...

# Nearest neighbor path

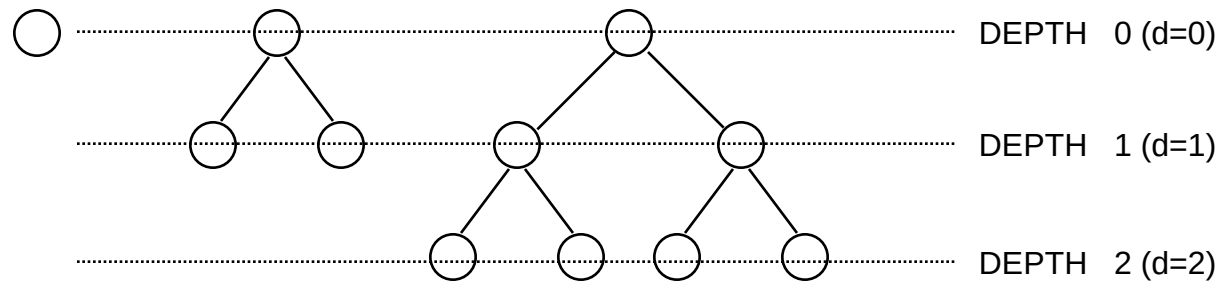


Nearest neighbor path = AEDBCA (550)

Minimal cost path = ABCDEA (375)

# Search

## ◆ Breadth-First Search



# 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;                      % loop check

| put remaining children on right end of open | % queue |
|---------------------------------------------|---------|
| 1                                           | 100     |
| 2                                           | 100     |
| 3                                           | 100     |
| 4                                           | 100     |
| 5                                           | 100     |
| 6                                           | 100     |
| 7                                           | 100     |
| 8                                           | 100     |
| 9                                           | 100     |
| 10                                          | 100     |
| 11                                          | 100     |
| 12                                          | 100     |
| 13                                          | 100     |
| 14                                          | 100     |
| 15                                          | 100     |
| 16                                          | 100     |
| 17                                          | 100     |
| 18                                          | 100     |
| 19                                          | 100     |
| 20                                          | 100     |
| 21                                          | 100     |
| 22                                          | 100     |
| 23                                          | 100     |
| 24                                          | 100     |
| 25                                          | 100     |
| 26                                          | 100     |
| 27                                          | 100     |
| 28                                          | 100     |
| 29                                          | 100     |
| 30                                          | 100     |
| 31                                          | 100     |
| 32                                          | 100     |
| 33                                          | 100     |
| 34                                          | 100     |
| 35                                          | 100     |
| 36                                          | 100     |
| 37                                          | 100     |
| 38                                          | 100     |
| 39                                          | 100     |
| 40                                          | 100     |
| 41                                          | 100     |
| 42                                          | 100     |
| 43                                          | 100     |
| 44                                          | 100     |
| 45                                          | 100     |
| 46                                          | 100     |
| 47                                          | 100     |
| 48                                          | 100     |
| 49                                          | 100     |
| 50                                          | 100     |
| 51                                          | 100     |
| 52                                          | 100     |
| 53                                          | 100     |
| 54                                          | 100     |
| 55                                          | 100     |
| 56                                          | 100     |
| 57                                          | 100     |
| 58                                          | 100     |
| 59                                          | 100     |
| 60                                          | 100     |
| 61                                          | 100     |
| 62                                          | 100     |
| 63                                          | 100     |
| 64                                          | 100     |
| 65                                          | 100     |
| 66                                          | 100     |
| 67                                          | 100     |
| 68                                          | 100     |
| 69                                          | 100     |
| 70                                          | 100     |
| 71                                          | 100     |
| 72                                          | 100     |
| 73                                          | 100     |
| 74                                          | 100     |
| 75                                          | 100     |
| 76                                          | 100     |
| 77                                          | 100     |
| 78                                          | 100     |
| 79                                          | 100     |
| 80                                          | 100     |
| 81                                          | 100     |
| 82                                          | 100     |
| 83                                          | 100     |
| 84                                          | 100     |
| 85                                          | 100     |
| 86                                          | 100     |
| 87                                          | 100     |
| 88                                          | 100     |
| 89                                          | 100     |
| 90                                          | 100     |
| 91                                          | 100     |
| 92                                          | 100     |
| 93                                          | 100     |
| 94                                          | 100     |
| 95                                          | 100     |
| 96                                          | 100     |
| 97                                          | 100     |
| 98                                          | 100     |
| 99                                          | 100     |
| 100                                         | 100     |

end

end

```
return FAIL % no states left
```

end.



# Search strategies - BFS

*BreadthFirstSearch*(**state space**  $\Sigma = \langle S, P, I, G, W \rangle$ )

Open  $\leftarrow \{I\}$

Closed  $\leftarrow \emptyset$

**while** Open  $\neq \emptyset$  **do**

$x \leftarrow DeQueue(Open)$

**if** *Goal*( $x, \Sigma$ ) **then return**  $x$

*Insert*( $x, Closed$ )

**for**  $y \in Child(x, \Sigma)$  **do**

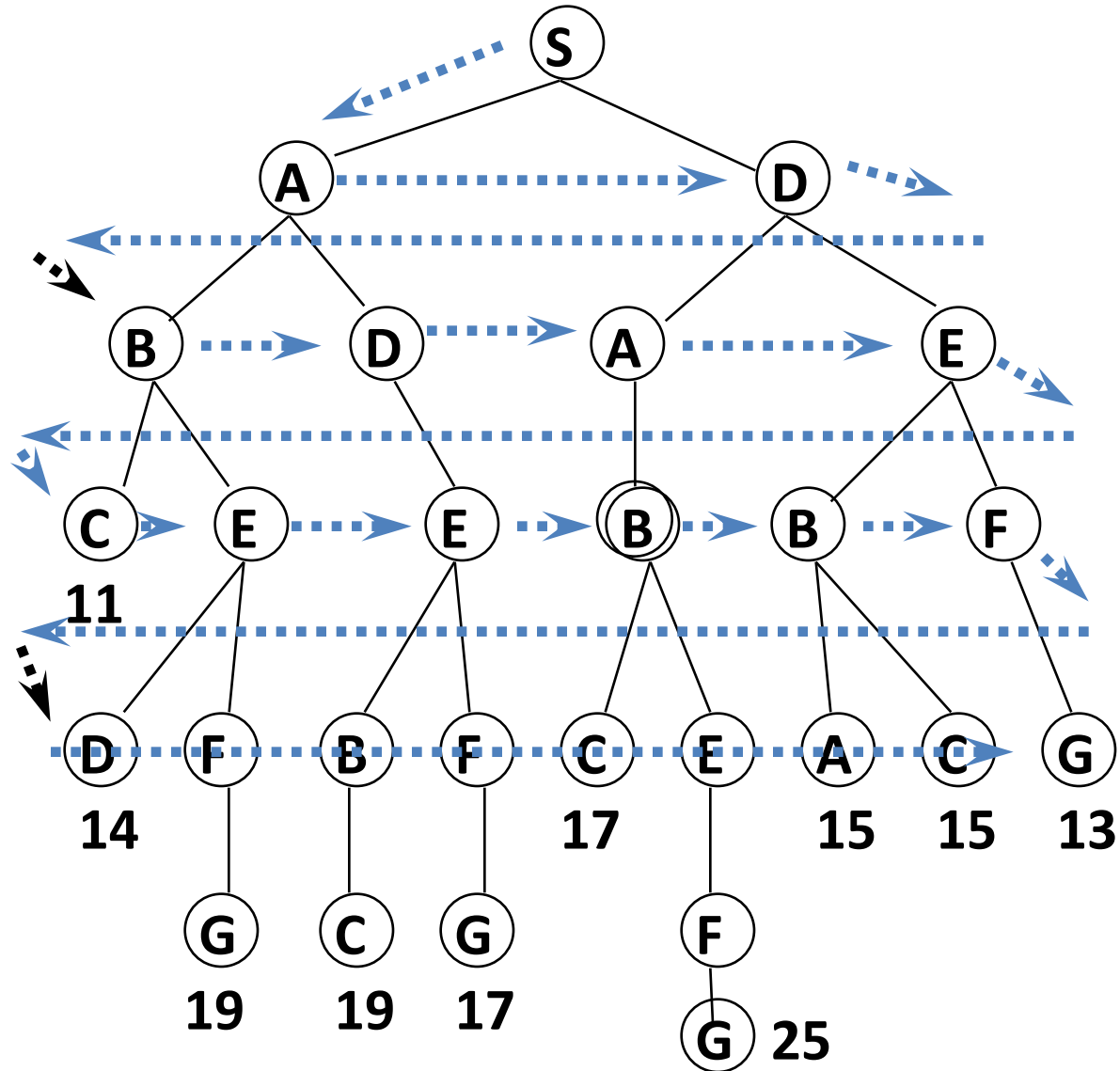
**if**  $y \notin Closed$  and  $y \notin Open$  **then**

*EnQueue*( $y, Open$ )

**return** *fail*

Open is implemented as *queue* (FIFO); Closed can be an arbitrary data structure for sets

# Breath-first search



# Breadth-first search

Expand shallowest unexpanded node

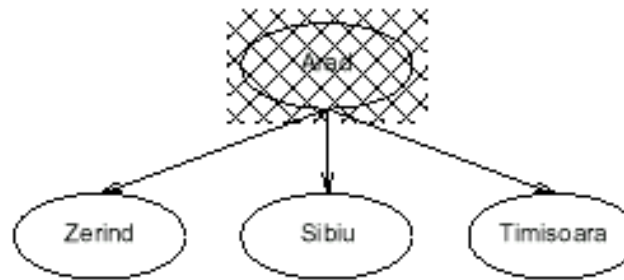
Implementation:

QUEUEINGFN = put successors at end of queue

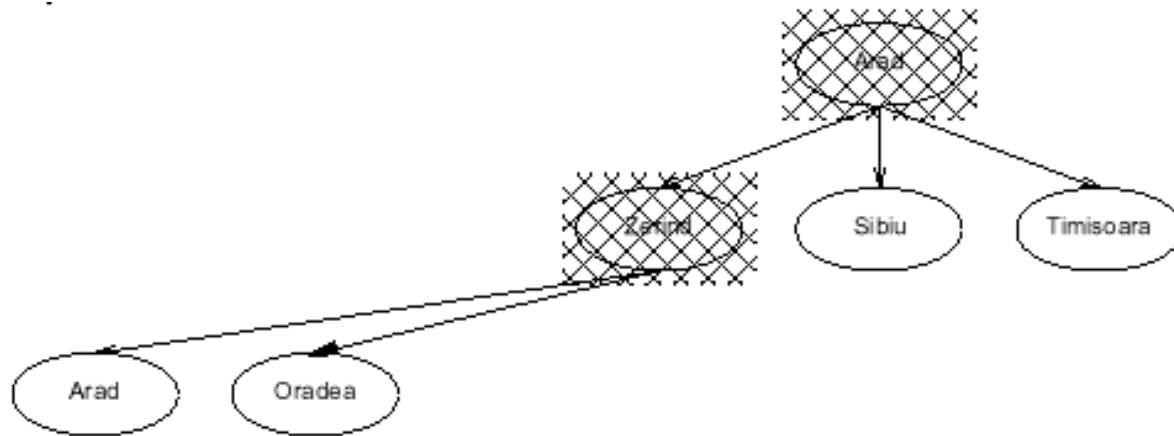


Arad

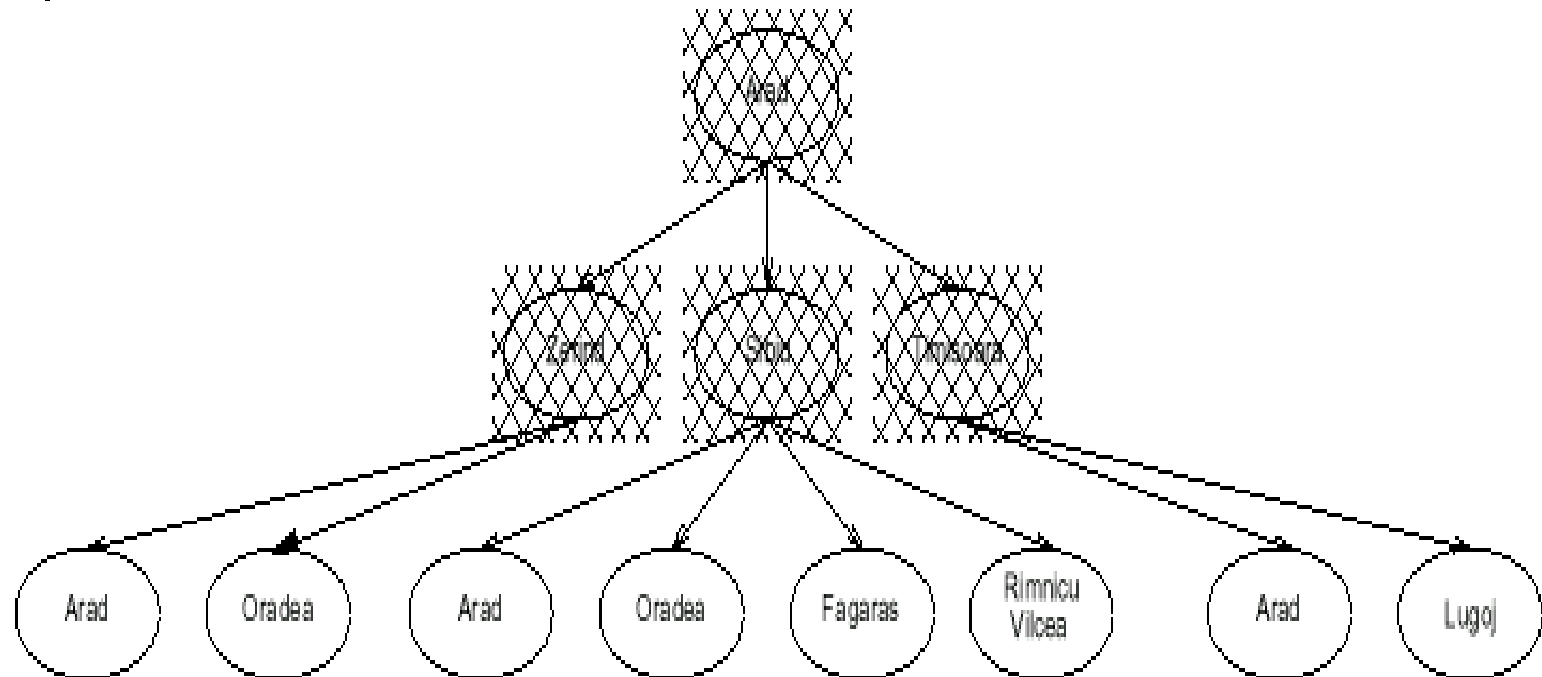
# Breadth-first search



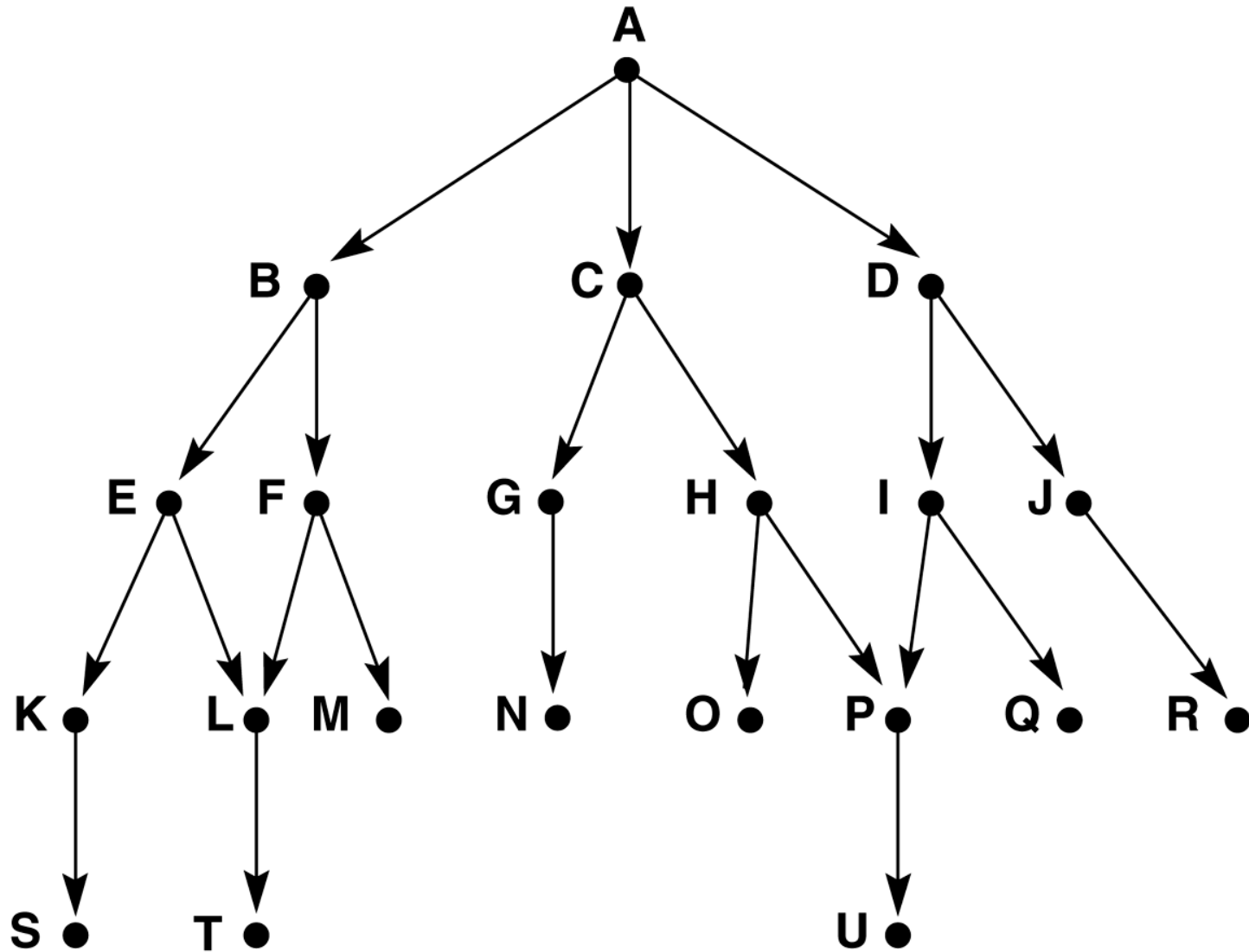
# Breadth-first search



# Breadth-first search



# Graph for BFS and DFS (Fig. 3.13)

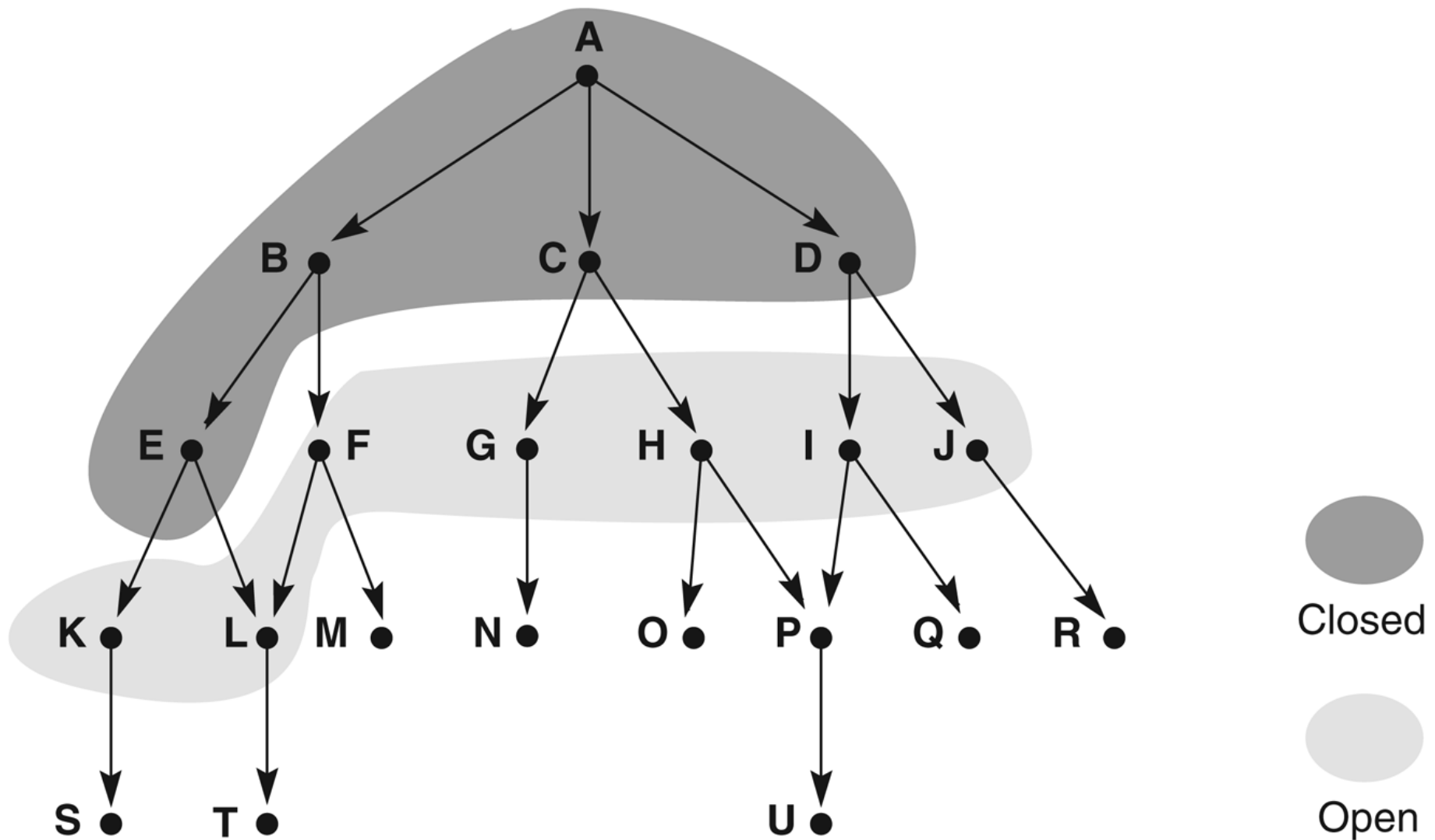


# Trace of BFS on the graph of Fig. 3.13

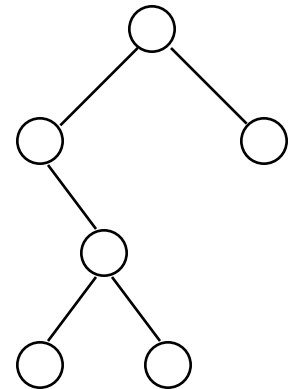
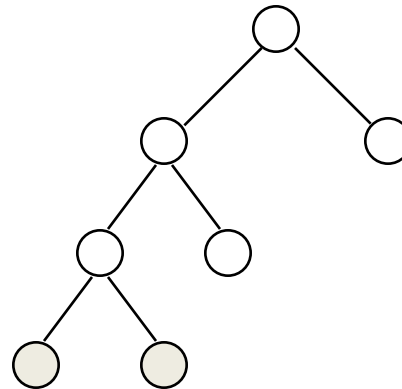
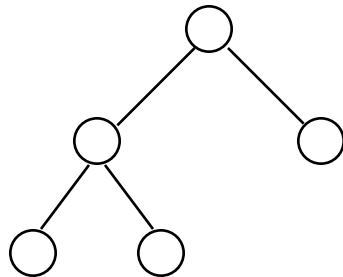
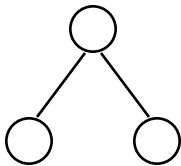
1. **open** = [A]; **closed** = [ ]
2. **open** = [B,C,D]; **closed** = [A]
3. **open** = [C,D,E,F]; **closed** = [B,A]
4. **open** = [D,E,F,G,H]; **closed** = [C,B,A]
5. **open** = [E,F,G,H,I,J]; **closed** = [D,C,B,A]
6. **open** = [F,G,H,I,J,K,L]; **closed** = [E,D,C,B,A]
7. **open** = [G,H,I,J,K,L,M] (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** = [ ]



# Graph of Fig. 3.13 at iteration 6 of BFS



# Depth-First Search



# Depth\_first\_search algorithm

[illegible]

# Search strategies - DFS

*DepthFirstSearch*(**state space**  $\Sigma = \langle S, P, I, G, W \rangle$ )

Open  $\leftarrow \{I\}$

Closed  $\leftarrow \emptyset$

**while** Open  $\neq \emptyset$  **do**

$x \leftarrow \text{Pop}(\text{Open})$

**if** *Goal*( $x, \Sigma$ ) **then return**  $x$

*Insert*( $x, \text{Closed}$ )

**for**  $y \in \text{Child}(x, \Sigma)$  **do**

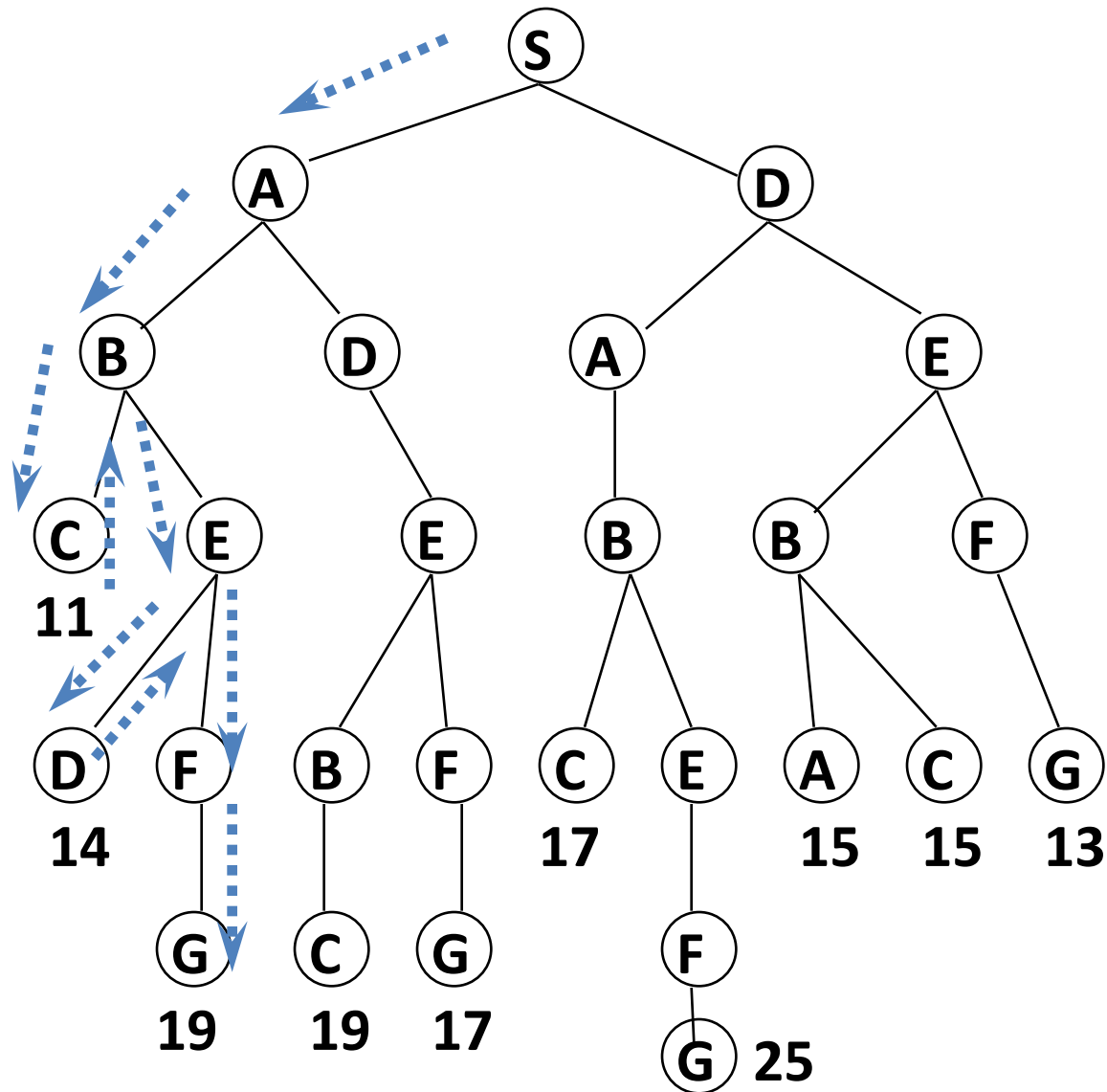
**if**  $y \notin \text{Closed}$  and  $y \notin \text{Open}$  **then**

*Push*( $y, \text{Open}$ )

**return fail**

Open is implemented as *stack* (LIFO); Closed can be an arbitrary data structure for sets

# Depth-first search



# Depth-first search

Expand deepest unexpanded node

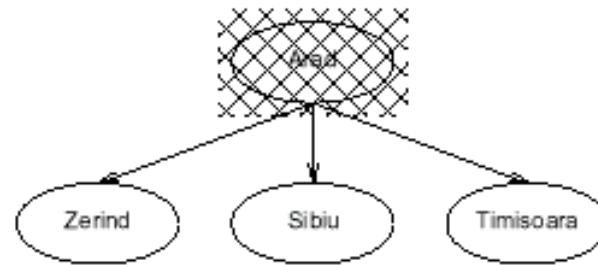
Implementation:

QUEUEINGFN = insert successors at front of queue

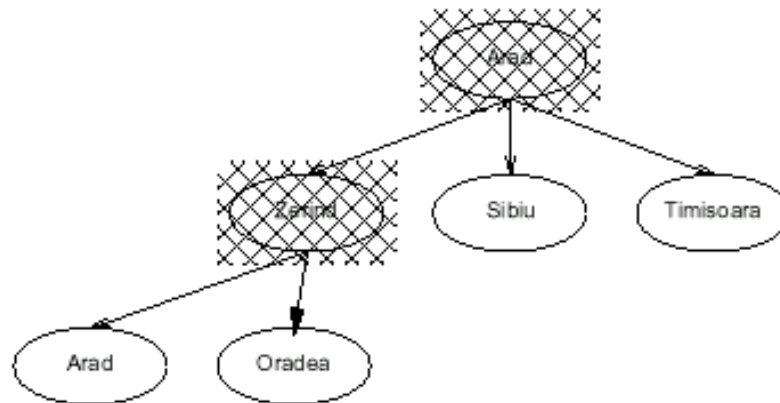


Arad

# Depth-first search

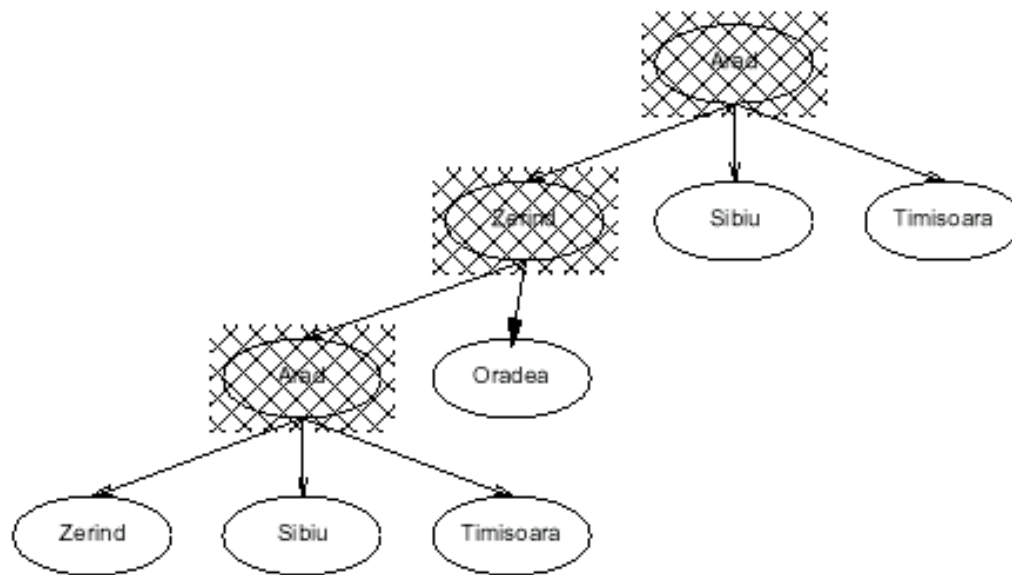


# Depth-first search



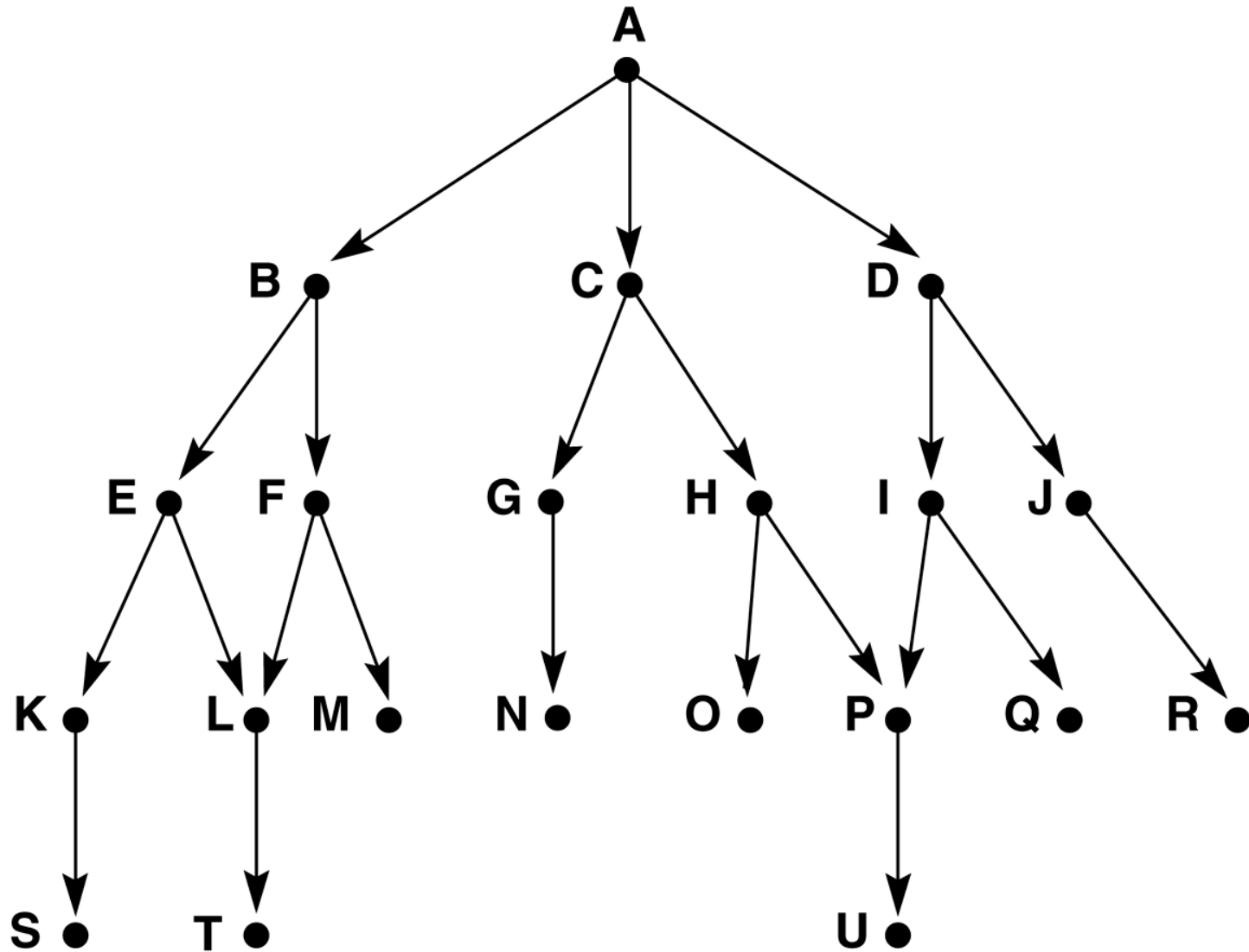


# Depth-first search



I.e., depth-first search can perform infinite cyclic excursions  
Need a finite, non-cyclic search space (or repeated-state checking)

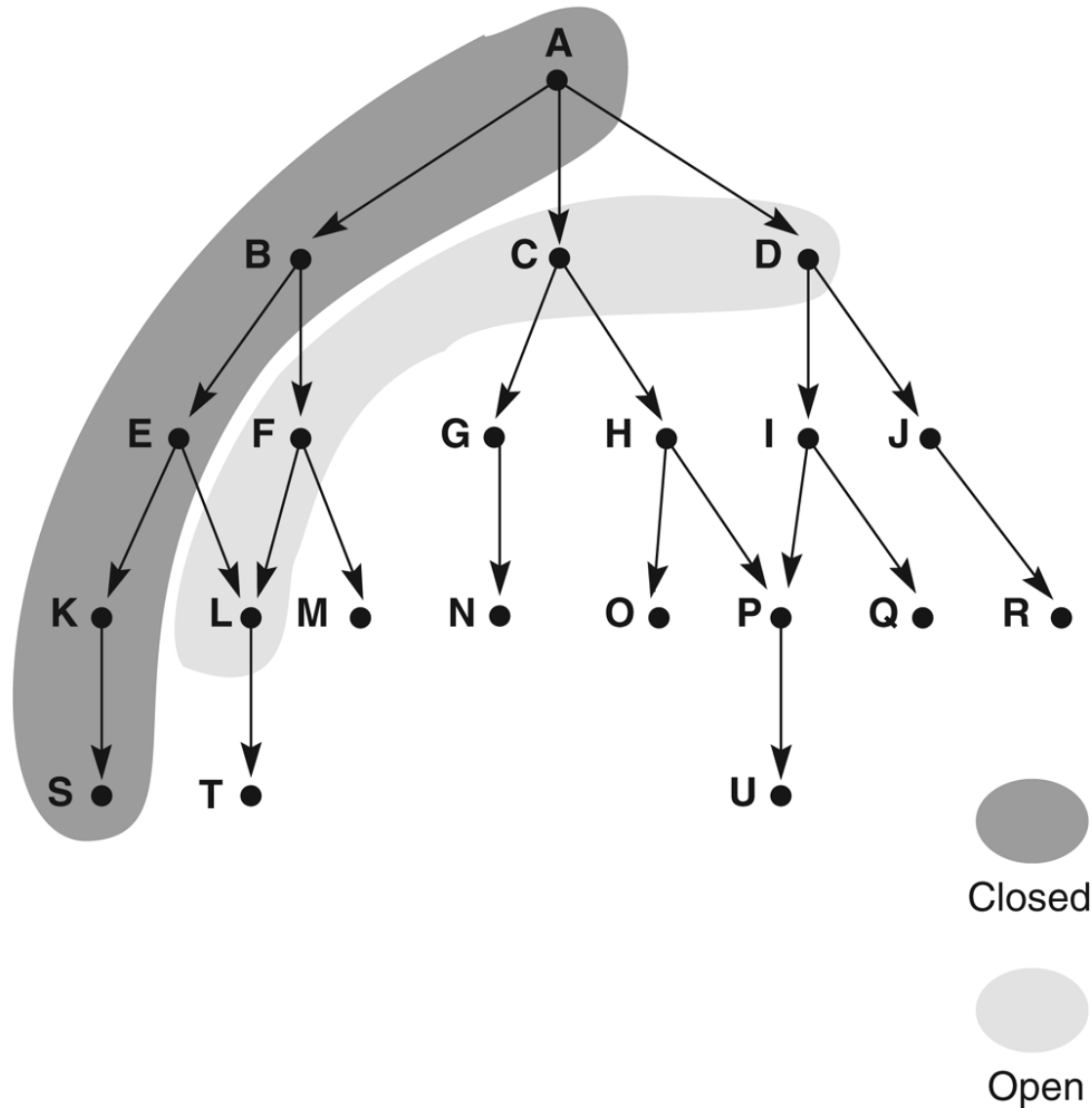
# Graph for BFS and DFS (Fig. 3.13)



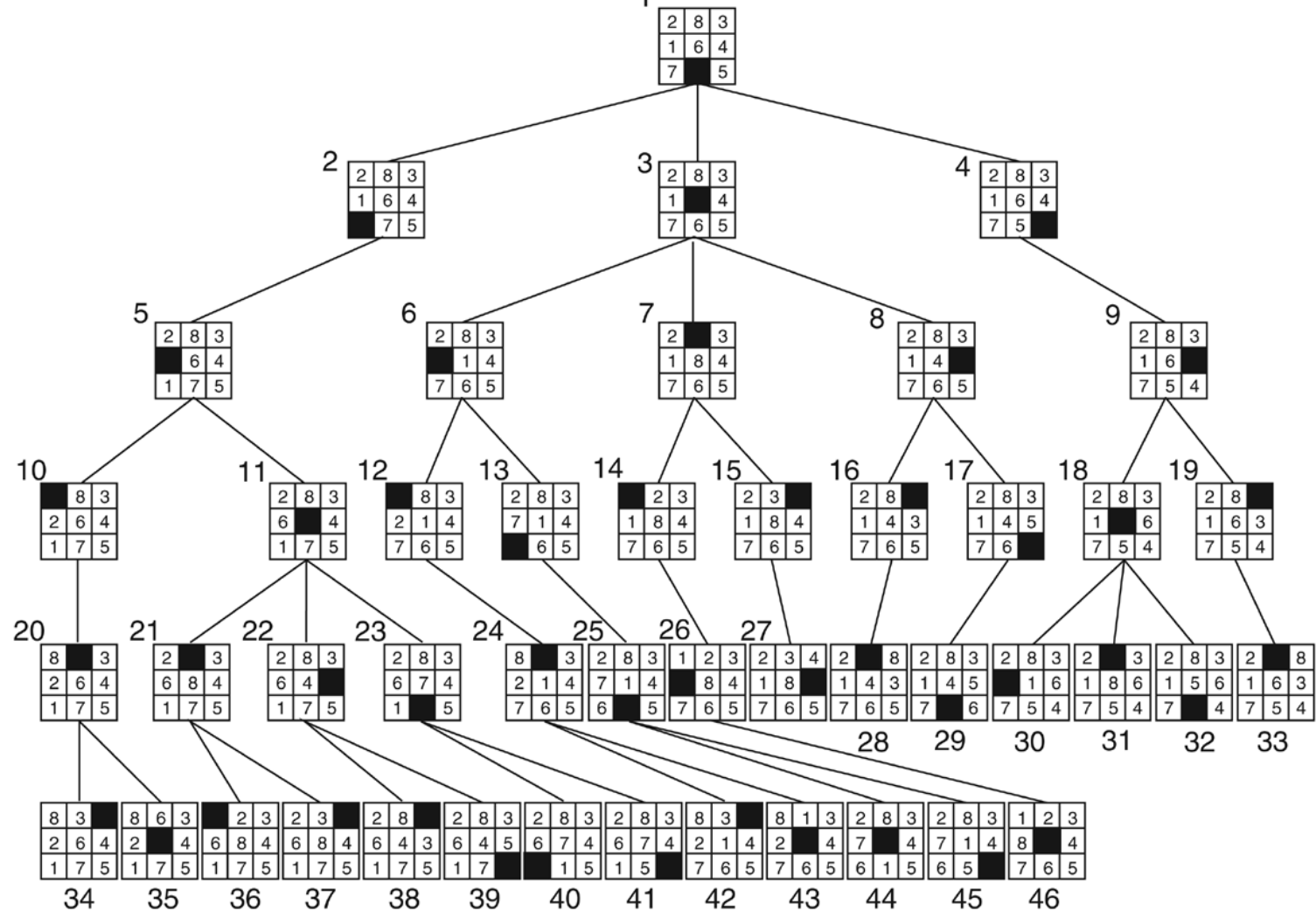
# Trace of DFS on the graph of Fig. 3.13

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]**

# Graph of Fig. 3.13 at iteration 6 of DFS

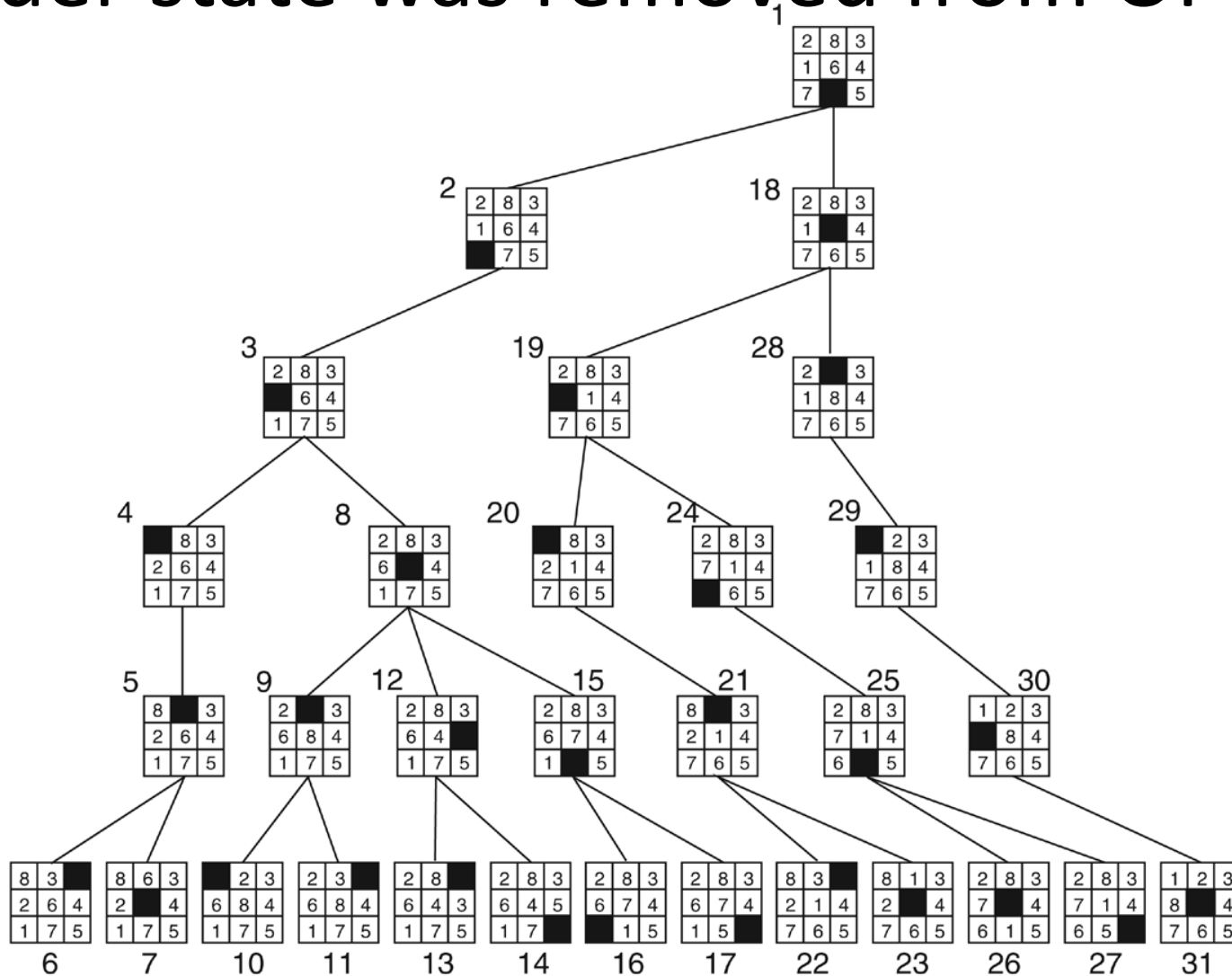


# BFS, label = order state was removed from OPEN



Goal

DFS with a depth bound of 5, label =  
order state was removed from OPEN



Goal

# “Blind search”

- BFS dan DFS disebut “Blind search” dalam arti bahwa metode ini tidak memiliki pengetahuan tentang masalah sama sekali selain ruang permasalahan
- Metode BFS dan DFS disebut juga brute-force search, uninformed search, atau weak method
- Dengan metode ini tidak dapat berharap terlalu banyak, tapi metode ini memberikan:
  - Worst-case scenarios
  - dasar dari algoritma berikutnya

# Search strategies

- A search strategy is defined by picking the **order of node expansion**
- Strategies are evaluated along the following dimensions:
  - **completeness**: does it always find a solution if one exists?
  - **time complexity**: number of nodes generated
  - **space complexity**: maximum number of nodes in memory
  - **optimality**: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - $b$ : maximum branching factor of the search tree
  - $d$  is the *depth of the solution*
  - $m$  is the *maximum depth of the tree*



# Properties of BFS dan DFS

- Completeness: (Does it always find a solution?)
- Time complexity: (How long does it take?)
- Space complexity: (How much memory does it take?)
- Optimality: (Does it always insure the “best”)

# Properties of breadth-first search

- Completeness: Yes, if  $b$  is finite
- Time complexity:  $O(b^d)$ , i.e., exponential in  $d$  (*Rem:  $b$  is no. of branches*)
- Space complexity:  $O(b^d)$ , keeps every node in memory
- Optimality: Yes, if cost = 1 per step; not optimal in general
- where  $b$  is the *branching factor*,  $d$  is the *depth of the solution*

# Kelebihan BFS

- Tidak akan menemui jalan buntu
- Jika ada satu solusi, maka BFS akan menemukannya. Dan jika ada lebih dari satu solusi, maka solusi minimum akan ditemukan.

# Kelemahan BFS

- Membutuhkan memori yang cukup banyak, karena menyimpan semua node dalam satu pohon.
- Membutuhkan waktu yang cukup lama, karena akan menguji  $n$  level untuk mendapatkan solusi pada level ke- $(n+1)$

# Properties of depth-first search

- Completeness: No, fails in infinite state-space
- Time complexity:  $O(b^m)$   
terrible if  $m$  is much larger than  $d$   
but if solutions are dense, may be much faster than breadth-first
- Space complexity:  $O(bm)$
- Optimality: No

where  $b$  is the *branching factor*,  $m$  is the *maximum depth of the tree*

# Kelebihan DFS

- Membutuhkan memori yang relatif kecil, karena hanya node-node pada lintasan yang aktif saja yang disimpan.
- Secara kebetulan, metode DFS akan menemukan solusi tanpa harus menguji lebih banyak lagi dalam ruang keadaan.

# Kelemahan DFS

- Memungkinkan tidak ditemukan tujuan yang diharapkan
- Hanya akan mendapatkan 1 solusi pada setiap pencarian