Kecerdasan Buatan

Algoritma Pencarian Blind

- Breadth First Search dan Depth First Search

Oleh Politeknik Elektronika Negeri Surabaya

2017

Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer

Algoritma Pencarian Blind

- Breadth First Search
- Depth First Search

Tujuan Instruksi Umum

Mahasiswa memahami filosofi Kecerdasan Buatan dan mampu menerapkan beberapa metode Kecerdasan Komputasional dalam menyelesaikan sebuah permasalahan, baik secara individu maupun berkelompok/kerjasama tim.

Tujuan Instruksi Khusus

- Mengetahui definisi Algoritma Pencarian
- Mengetahui contoh algoritma pencarian yang hanya mencari solusi

Algoritma Pencarian

- 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* (Ruang Keadaan) adalah himpunan state yang mungkin dari suatu permasalahan
  - Himpunan *operator* digunakan untuk berpindah dari satu state ke state yang lain.
  - Ruang masalah dapat digambarkan dengan graph,
    - himpunan state dinyatakan dengan node
    - 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 → (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 → (1, 1, 1, 1)
  - Darah tujuan → (0, 0, 0, 0)

- State tujuan
  - Daerah asal → (0, 0, 0, 0)
  - Darah tujuan → (1, 1, 1, 1)

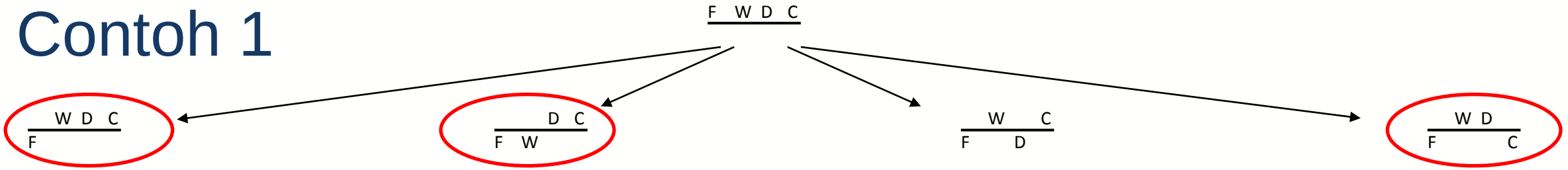
Rules

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

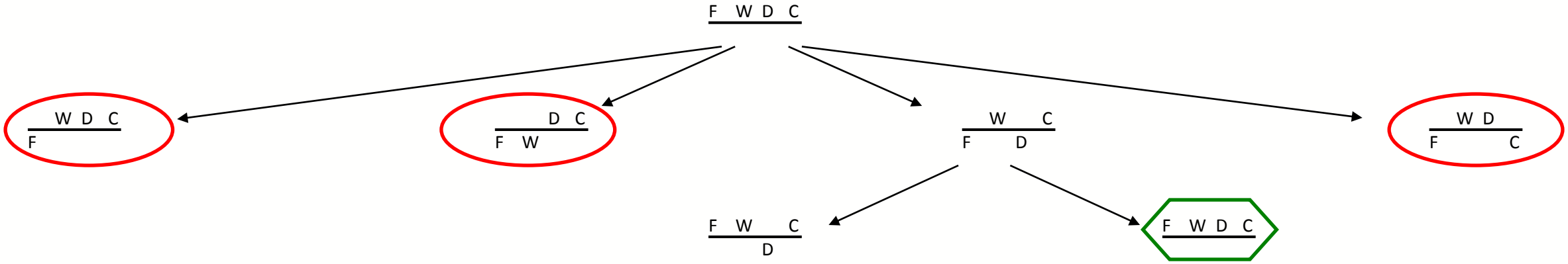
Daerah asal (S, A, Pd, Pt)	Daerah tujuan (S, A, Pd, Pt)	Rule yang dipakai
(1, 1, 1, 1)	(0, 0, 0, 0)	1
(1, 0, 1, 0)	(0, 1, 0, 1)	7
(1, 0, 1, 1)	(0, 1, 0, 0)	3
(0, 0, 1, 0)	(1, 1, 0, 1)	4
(0, 1, 1, 1)	(1, 0, 0, 0)	2
(0, 1, 0, 0)	(1, 0, 1, 1)	7
(0, 1, 0, 1)	(1, 0, 1, 0)	1
(0, 0, 0, 0)	(1, 1, 1, 1)	solusi

Contoh 1



Search Tree for “Farmer, Wolf, Duck, Corn”

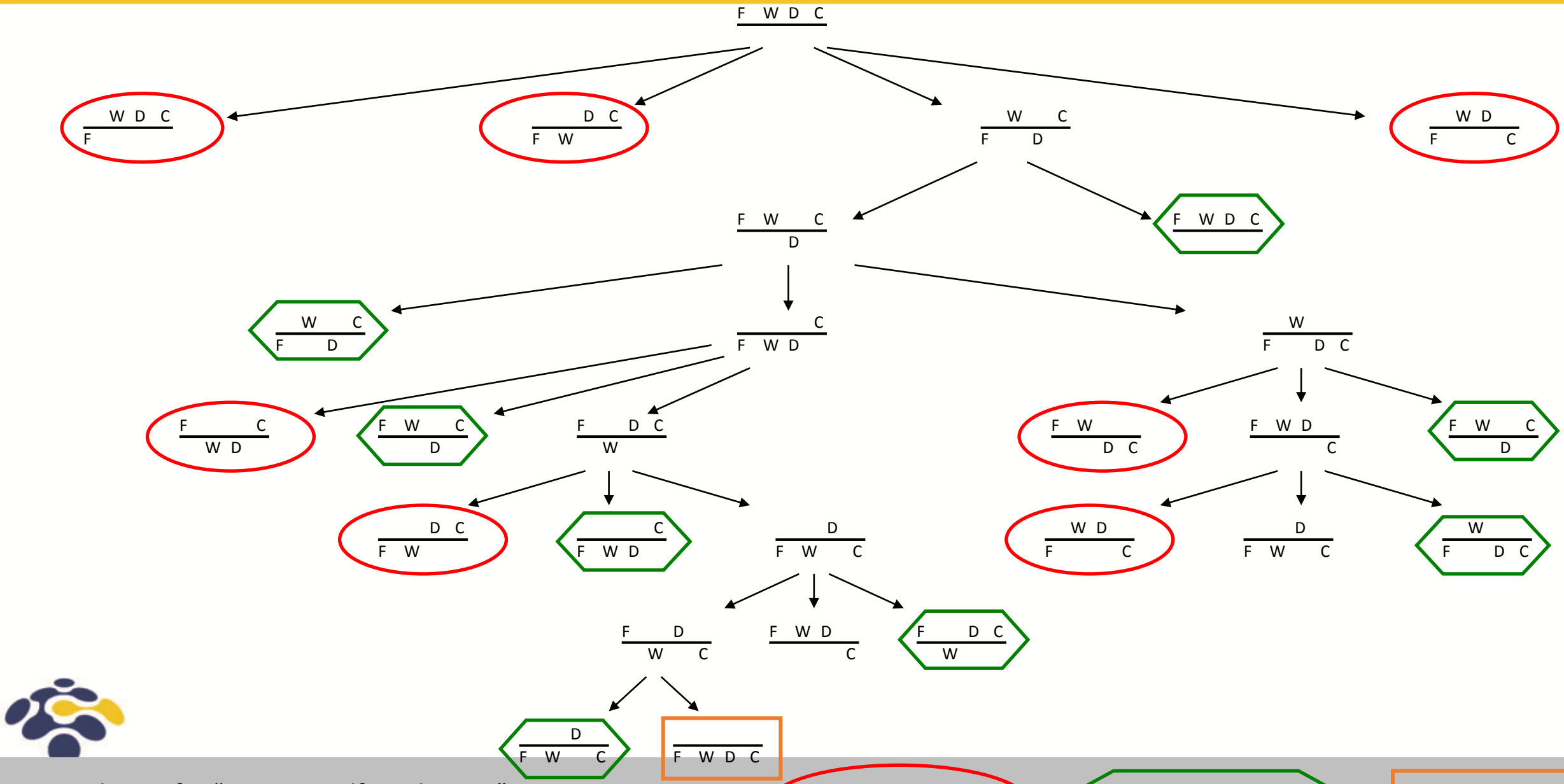
Illegal State

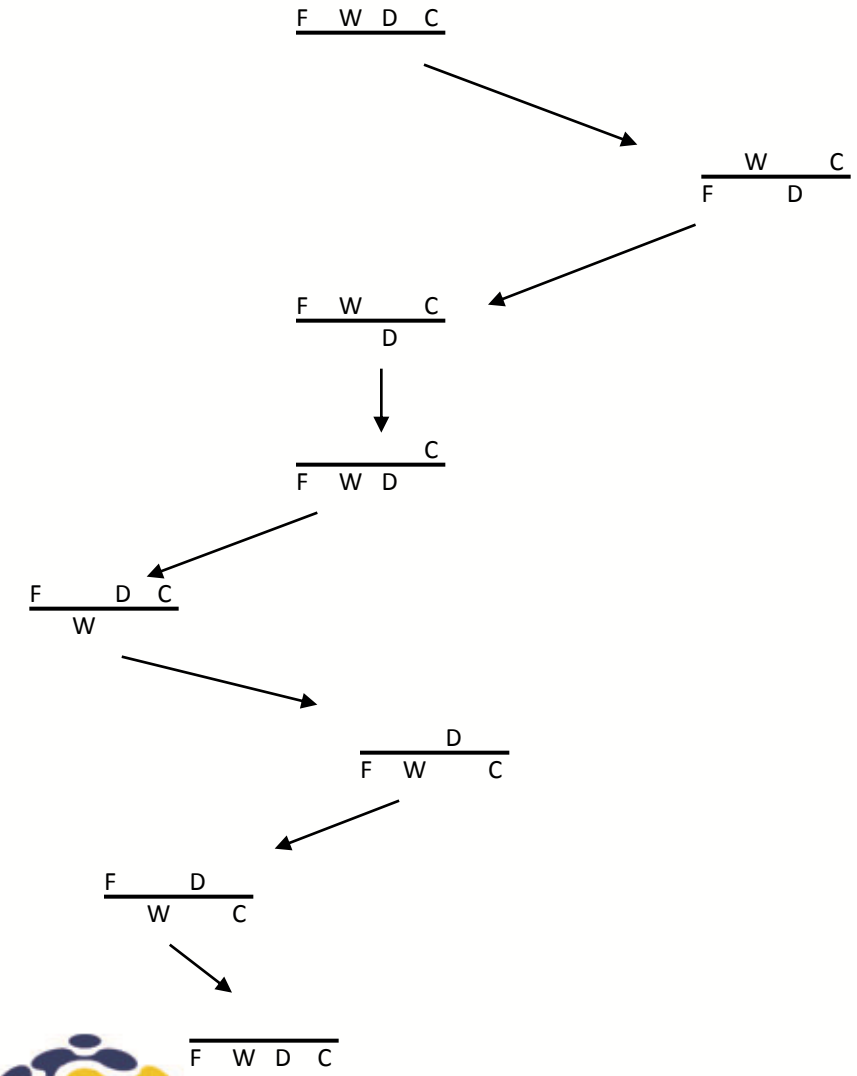


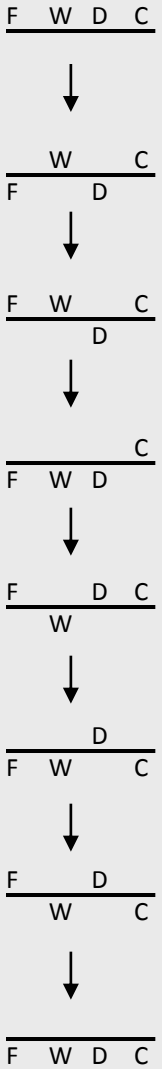
Search Tree for “Farmer, Wolf, Duck, Corn”

Illegal State

Repeated State


F W D C
W D C / F
D C / F W
W C / F D
W D / F C
F W C / D
F W D C
W C / F D
C / F W D
W / F D C
F C / W D
F W C / D
F D C / W
F W / D C
F W D / C
F W C / D
D C / F W
C / F W D
D / F W C
W D / F C
D / F W C
W / F D C
F D / W C
F W D / C
F D C / W
D / F W C
F W D C
Search Tree for “Farmer, Wolf, Duck, Corn”
Illegal State
Repeated State
Goal State





Initial State

Farmer takes duck to left bank

Farmer returns alone

Farmer takes wolf to left bank

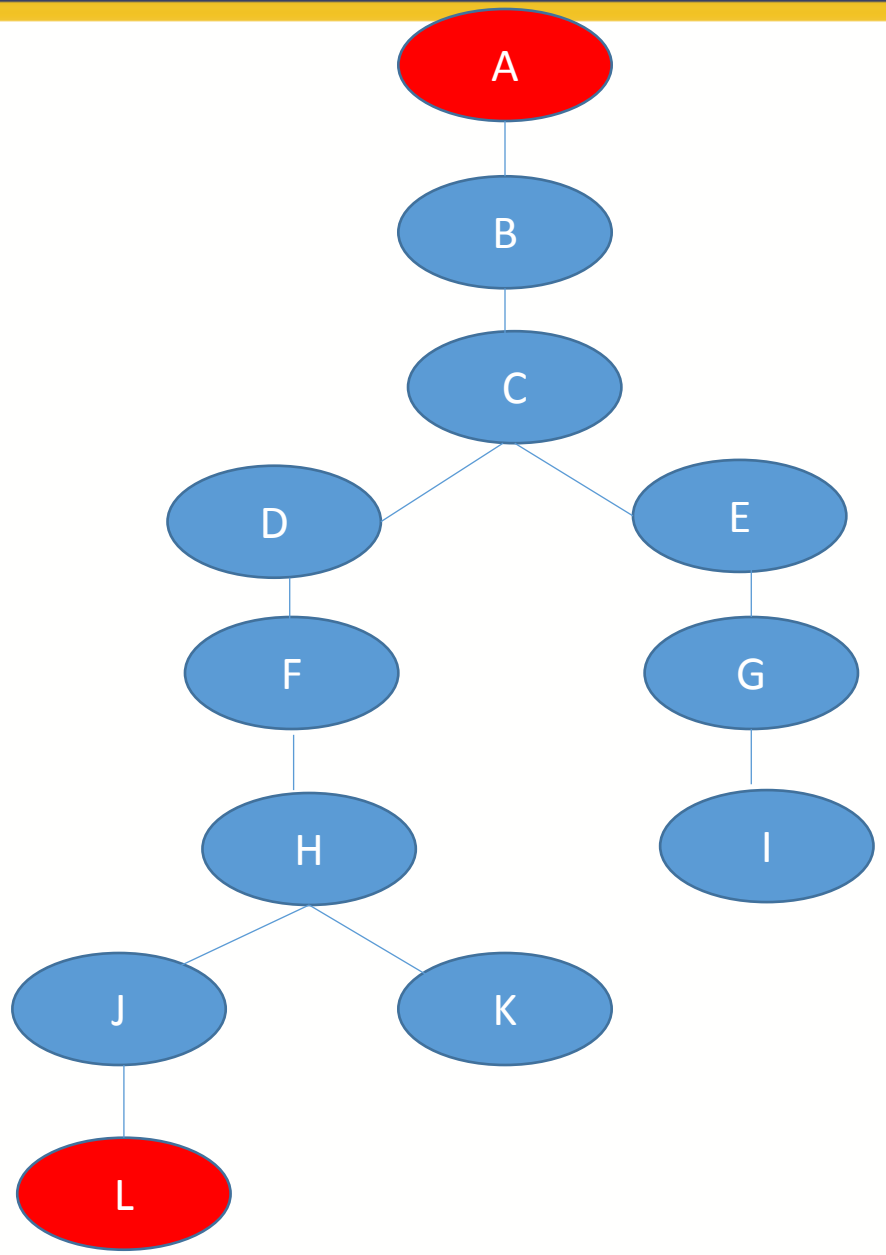
Farmer returns with duck

Farmer takes corn to left bank

Farmer returns alone

Farmer takes duck to left bank

Success!



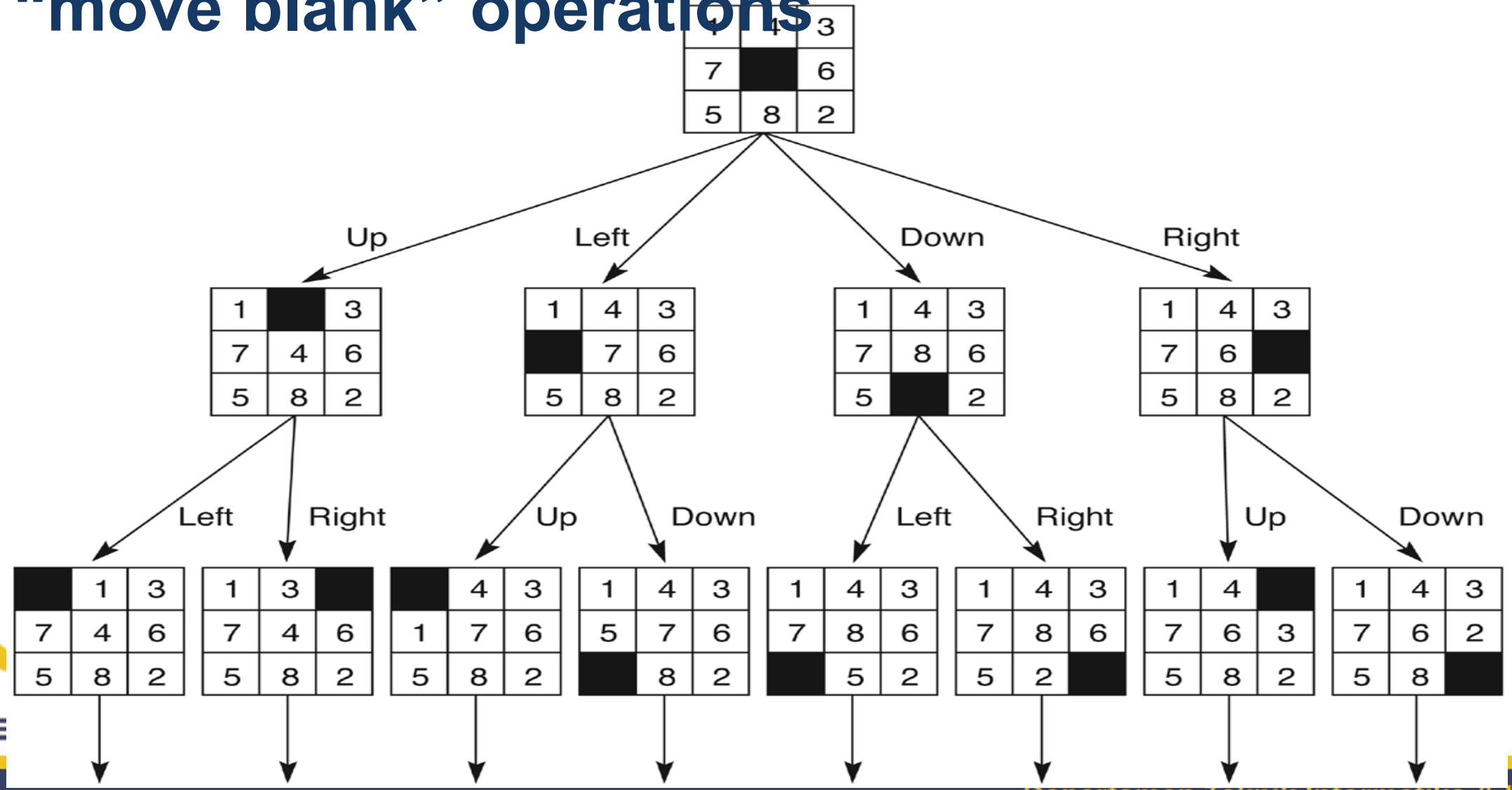
Node A = Node awal
Node L = Node Akhir

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



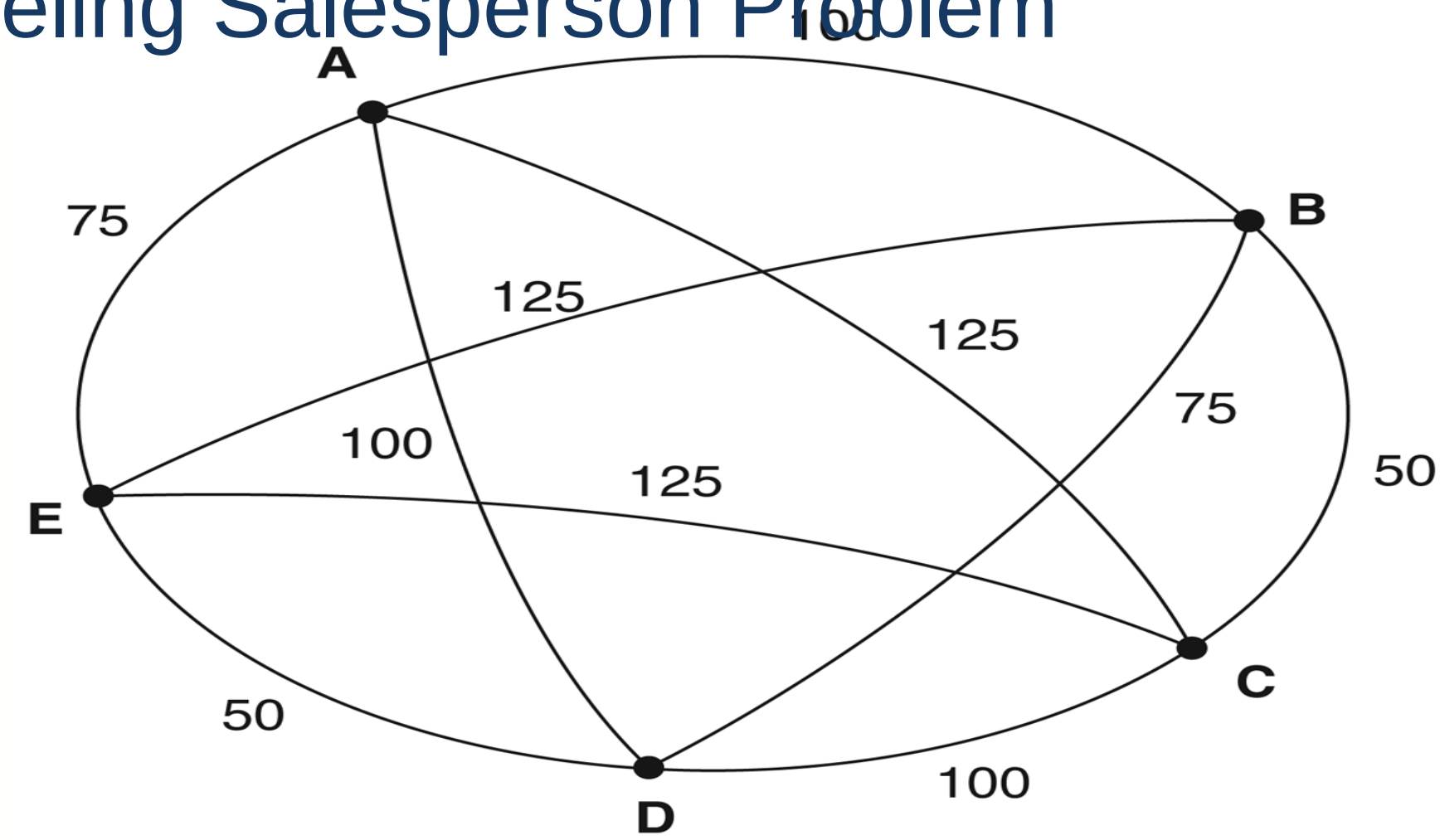
The 8-puzzle problem

- 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.

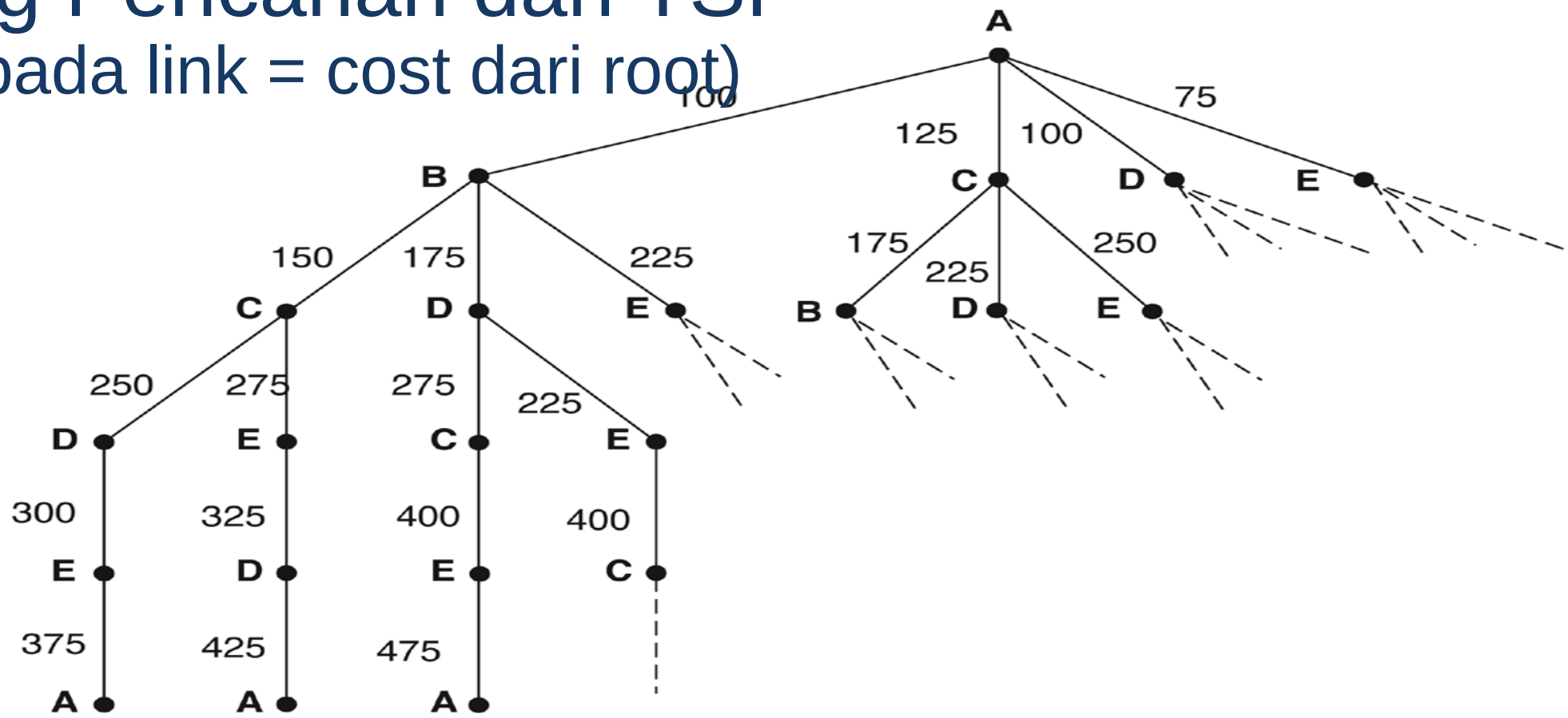
Contoh 3
Traveling Salesperson Problem



Traveling salesperson problem

- Seorang sales mengunjungi dari satu kota ke kota lain sebanyak n kota dan kembali ke kota asal. Carilah jarak terpendek untuk mengunjungi n kota tersebut.
- state space:
- operators:
- initial state:
- goal state:

Ruang Pencarian dari TSP
(label pada link = cost dari root)



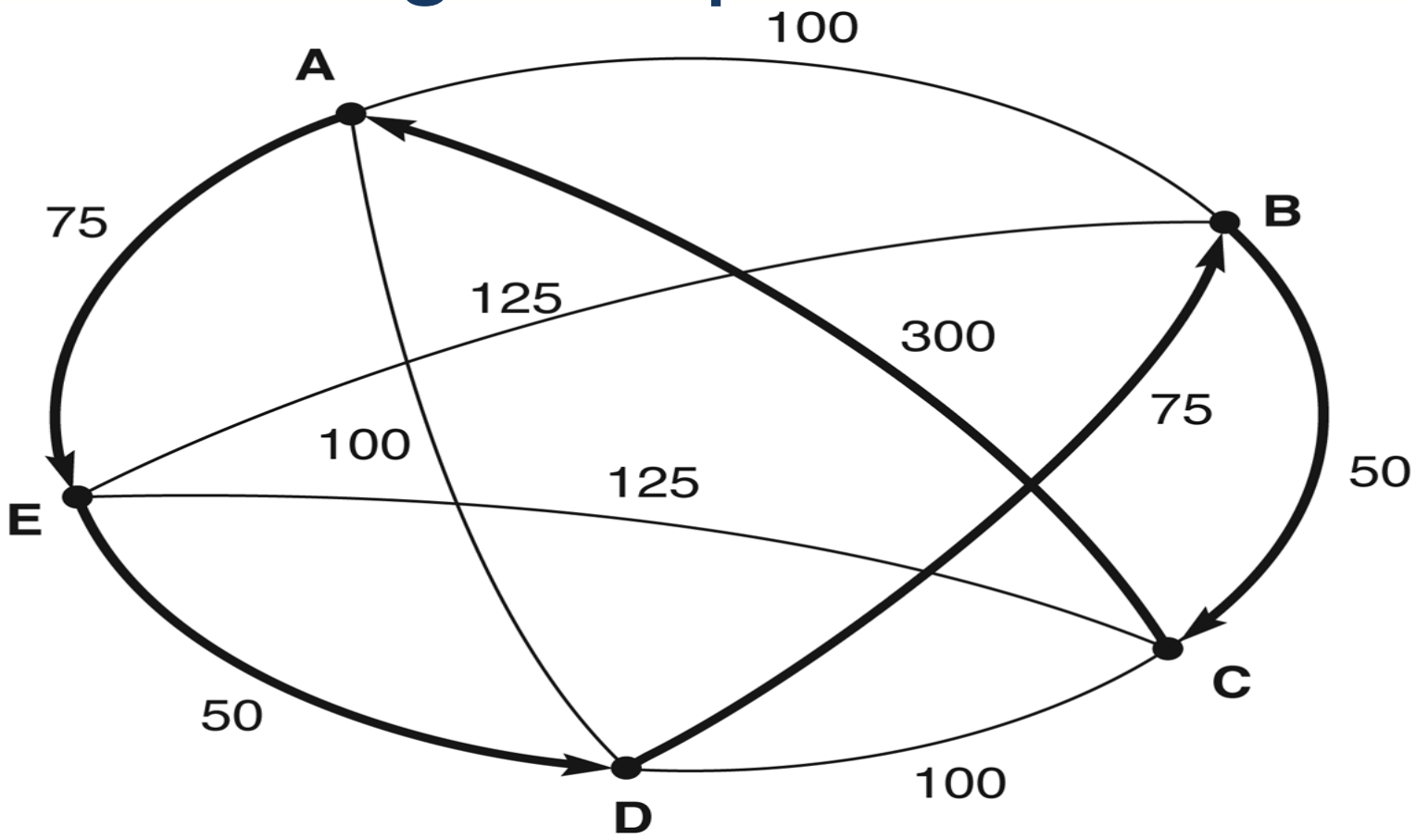
Path: ABCDEA
Cost: 375

Path: ABCEDA
Cost: 425

Path: ABDCEA
Cost: 475

. . .

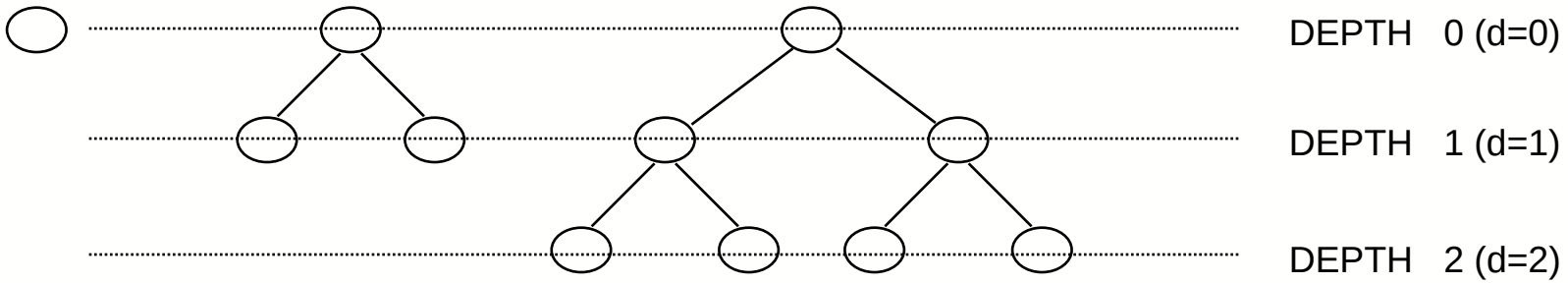
Nearest neighbor path



Nearest neighbor path = AEDBCA (550)

Minimal cost path = ABCDEA (375)

Breadth-First Search (BFS)



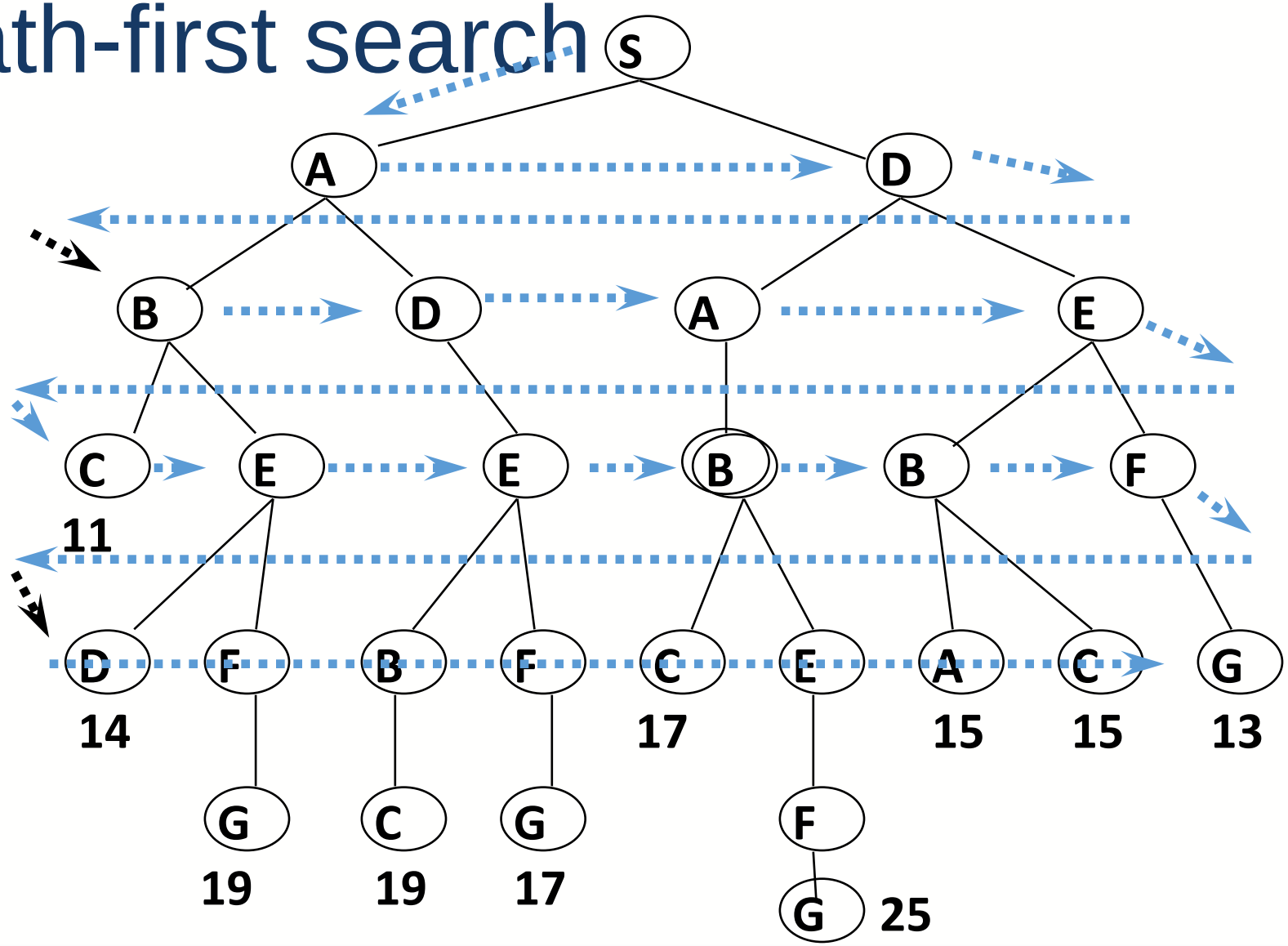
Algoritma BFS

BreadthFirstSearch(**state space** Σ=<S,P,I,G,W>)

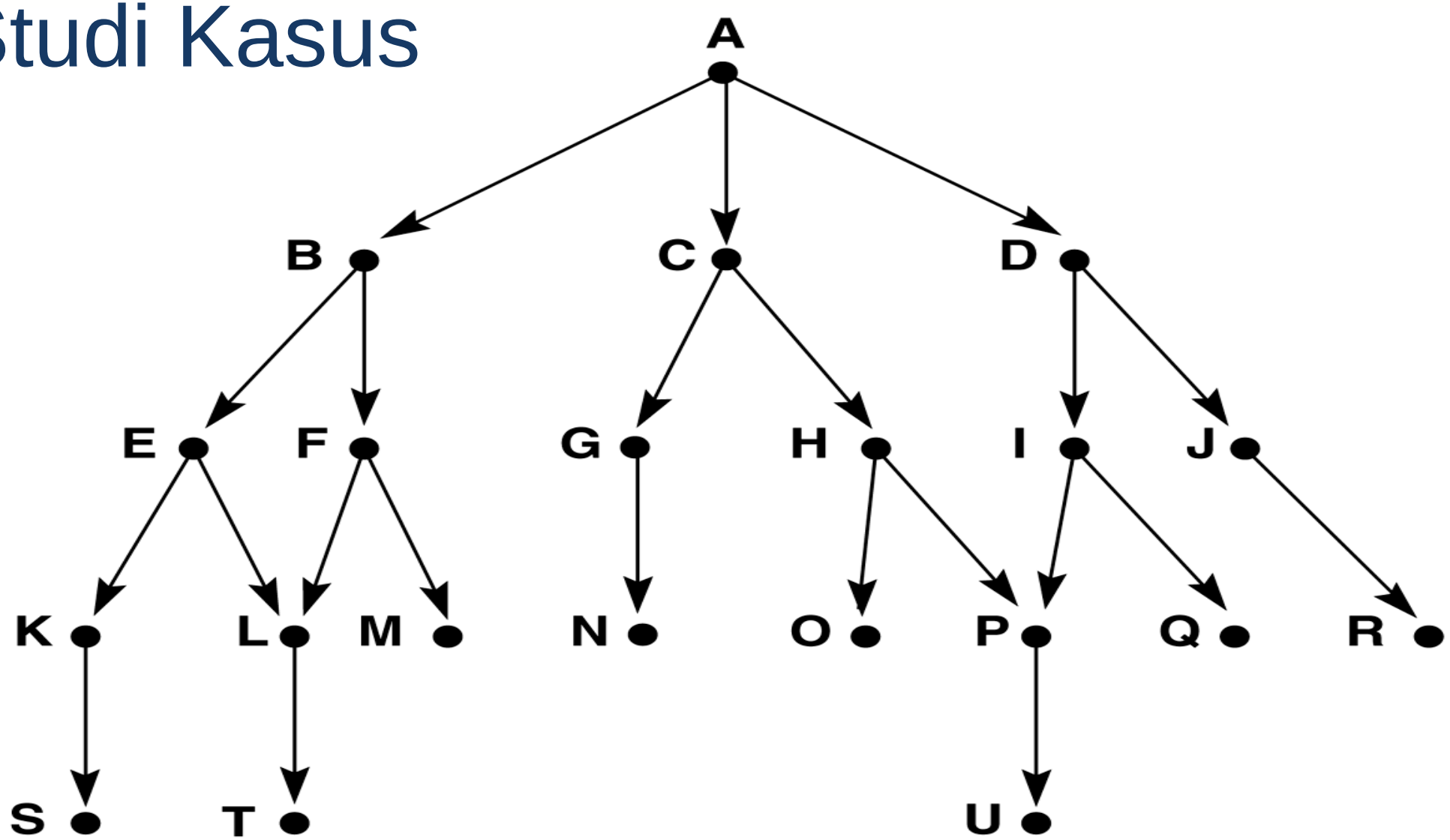
Open $\leftarrow$ {I}
Closed $\leftarrow \varnothing$
while Open $\neq \varnothing$ **do**
x $\leftarrow$ *DeQueue*(Open)
if *Goal*(x, Σ) **then return** x
Insert(x,Closed)
for y $\in$ *Child*(x ,Σ) **do**
if y$\notin$Closed and y$\notin$Open **then**
EnQueue(y,Open)
return *fail*

Open diimplementasikan dengan queue, menerapkan konsep FIFO. Closed → langsung ditampilkan ke layar.

Breath-first search



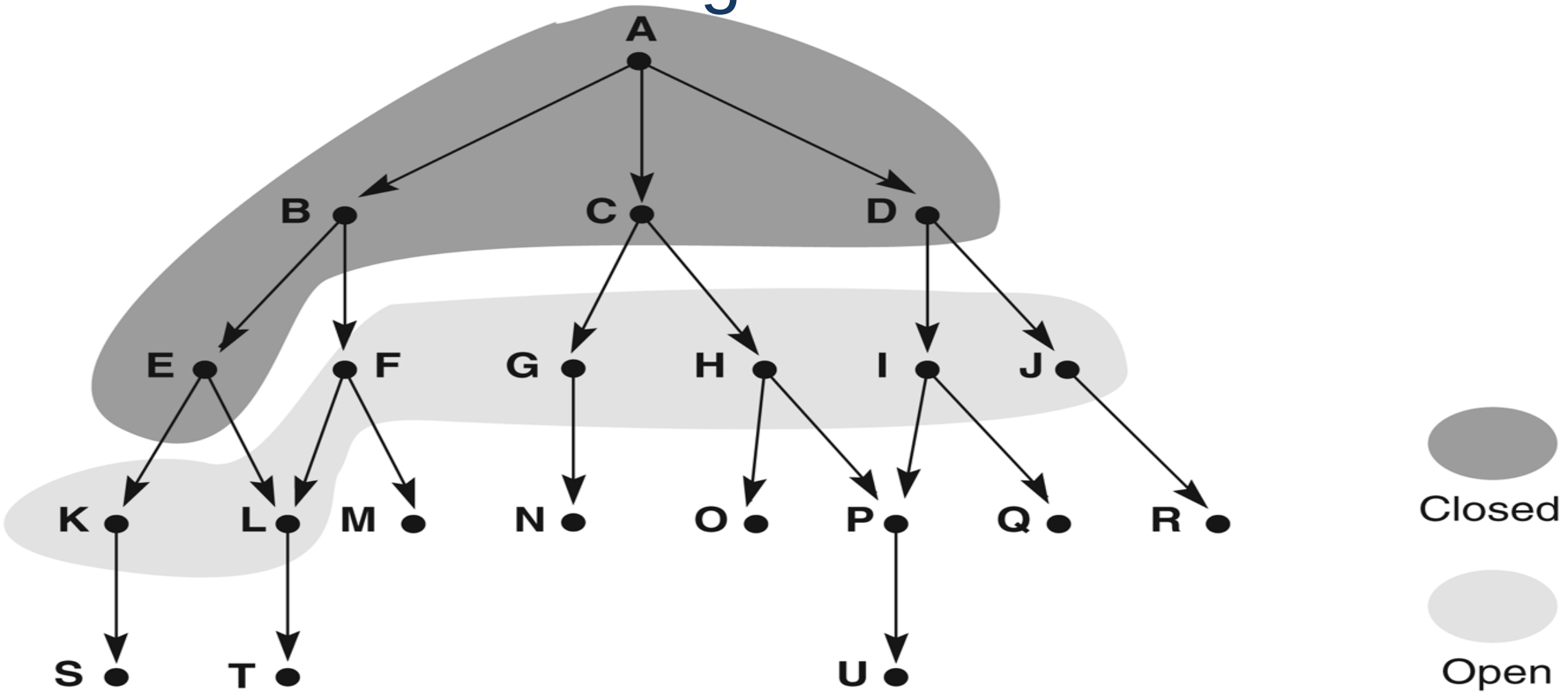
Studi Kasus



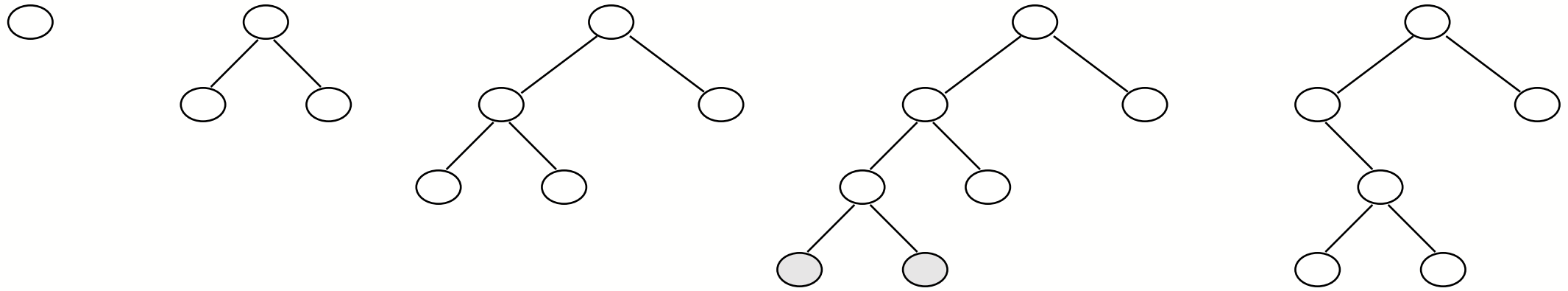
Trace dengan BFS

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** = [ ]

Pada Iterasi 6 dengan BFS



Depth-First Search



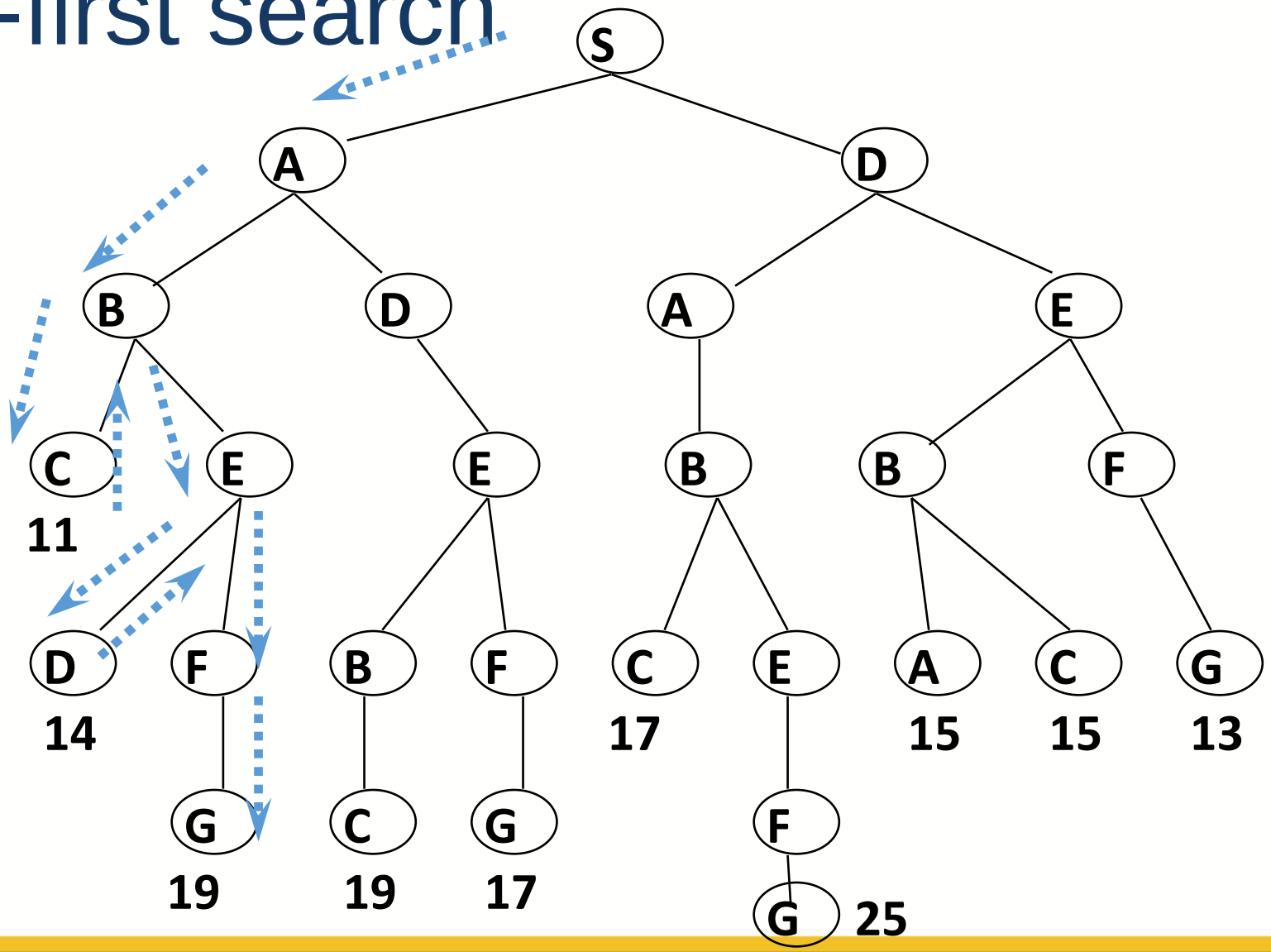
Algoritma DFS

DepthFirstSearch(**state space** Σ=<S,P,I,G,W>)

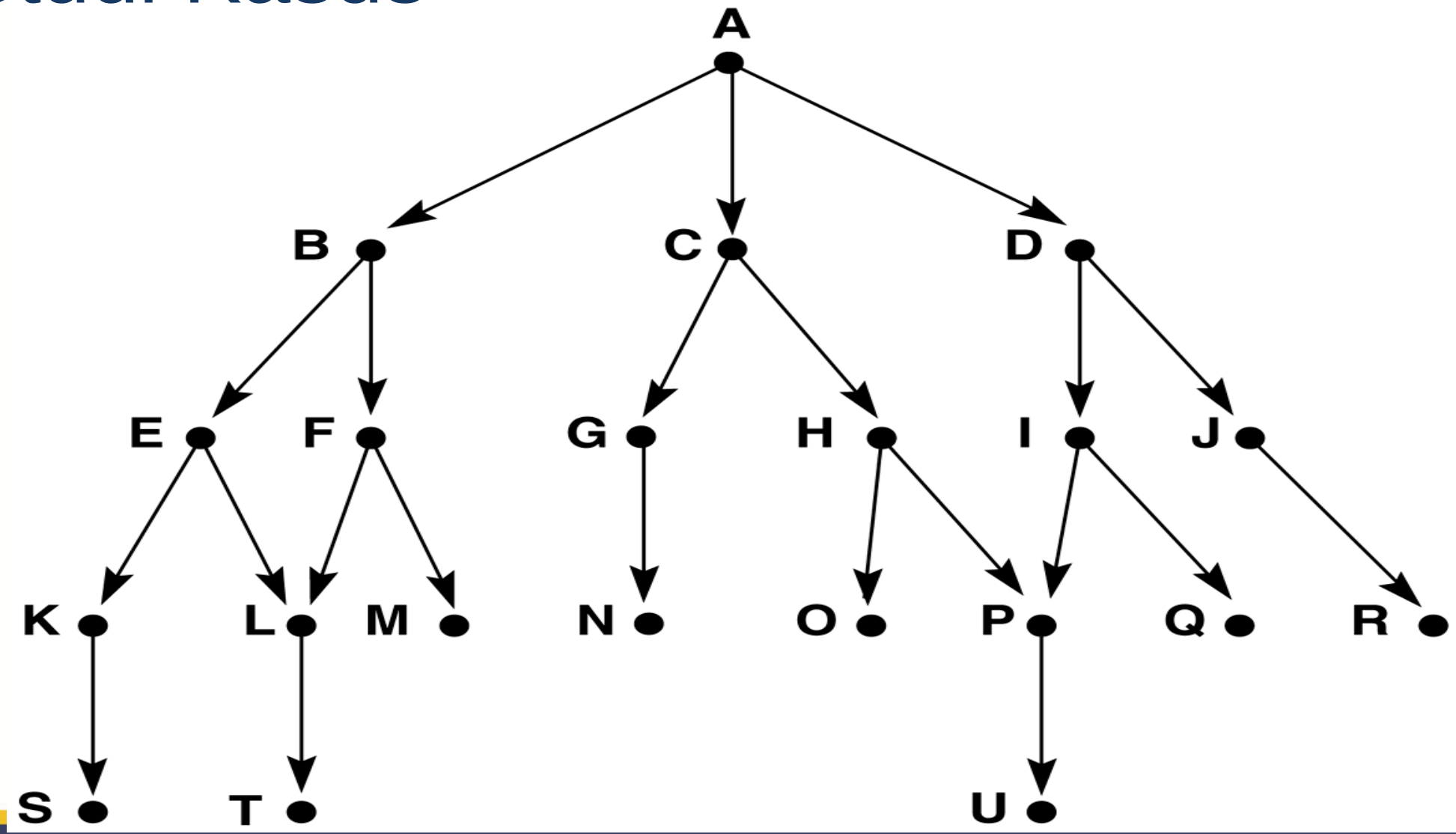
Open $\leftarrow$ {I}
Closed $\leftarrow \varnothing$
while Open $\neq \varnothing$ **do**
x $\leftarrow$ *Pop*(Open)
if *Goal*(x, Σ) **then return** x
Insert(x,Closed)
for y $\in$ *Child*(x ,Σ) **do**
if y$\notin$Closed and y$\notin$Open **then**
Push(y,Open)
return *fail*

Open diimplementasikan dengan Stack, menerapkan konsep LIFO. Closed → langsung ditampilkan ke layar.

Depth-first search



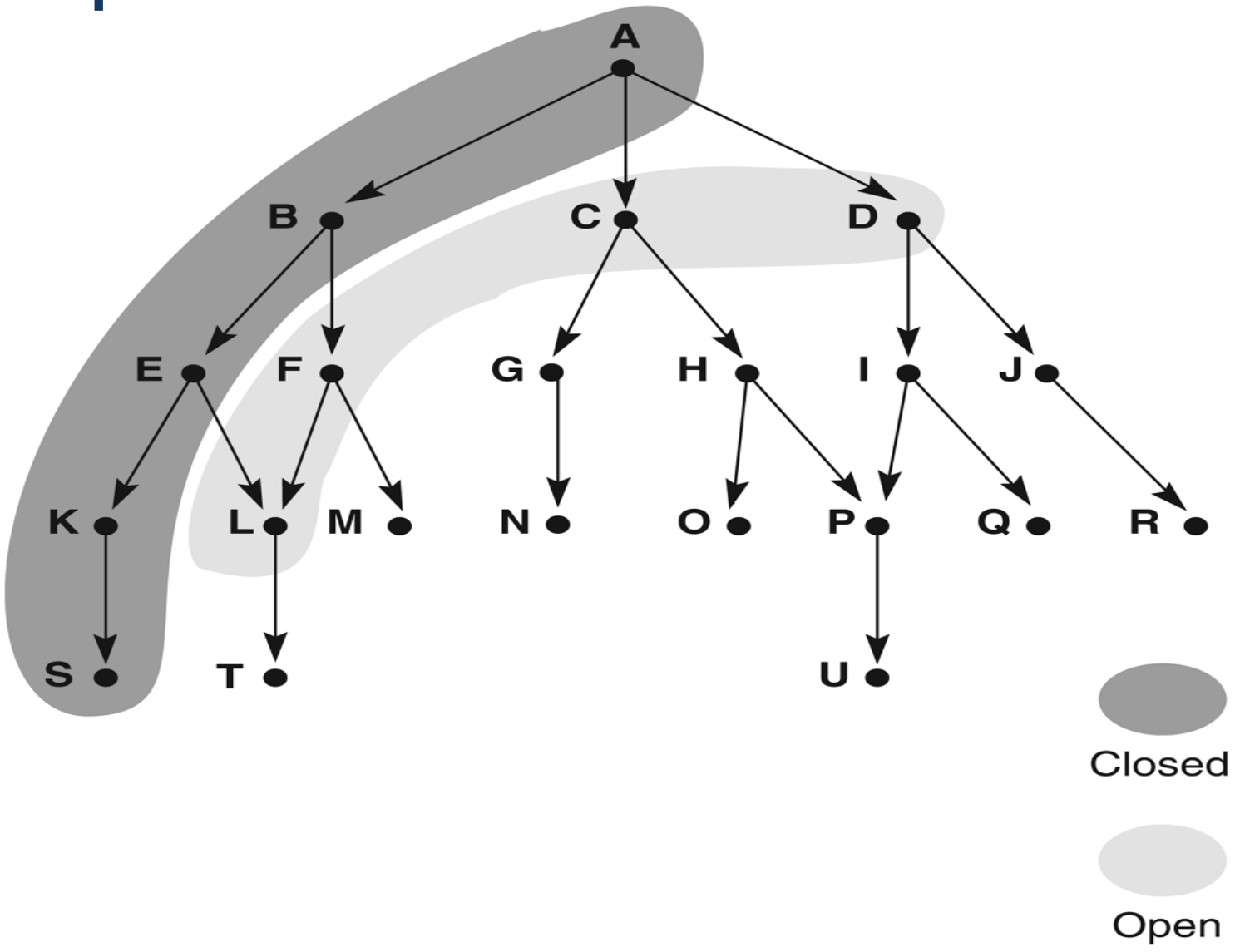
Studi Kasus



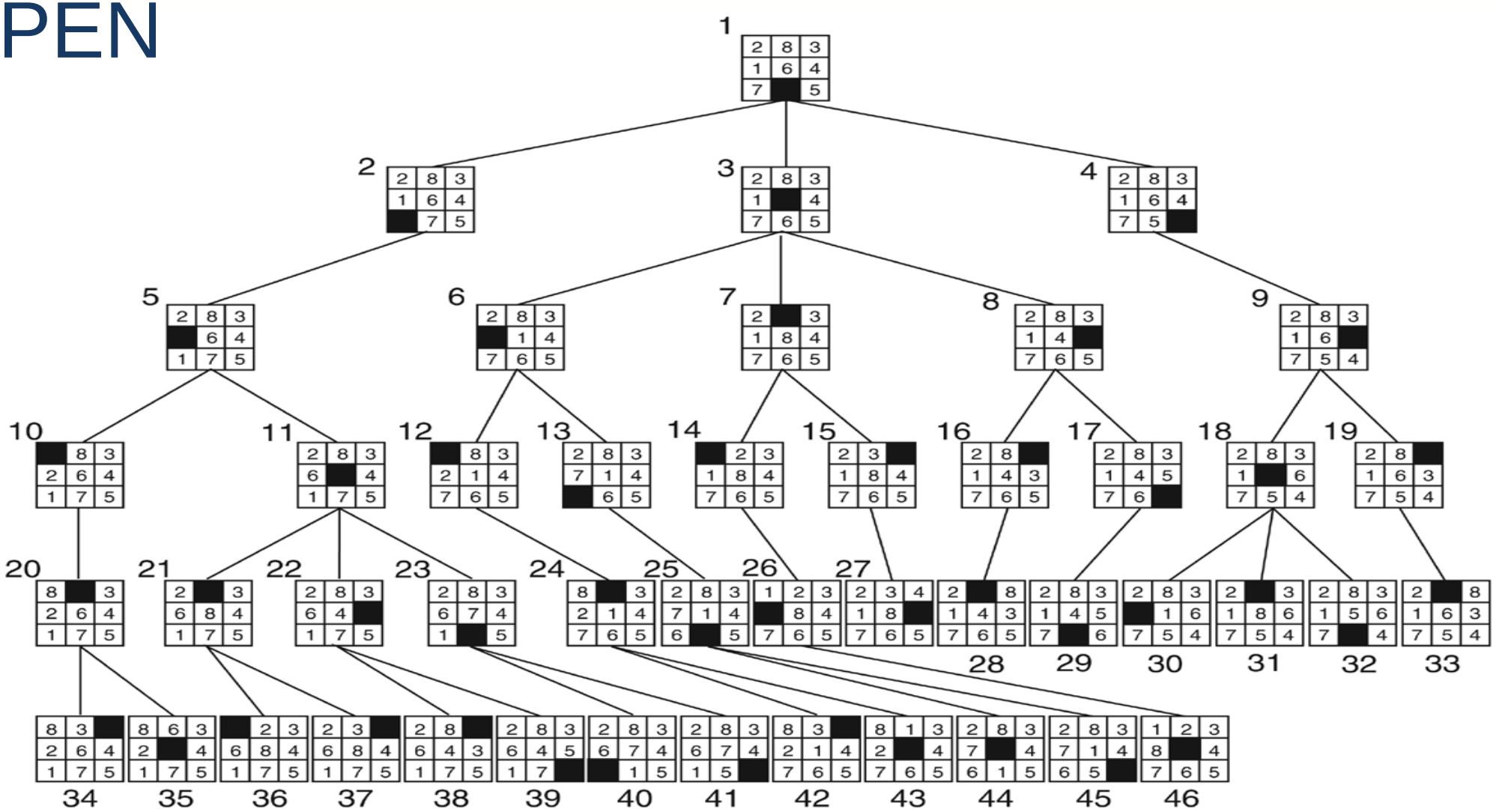
Trace of DFS on the graph of Fig. 3.13

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]**

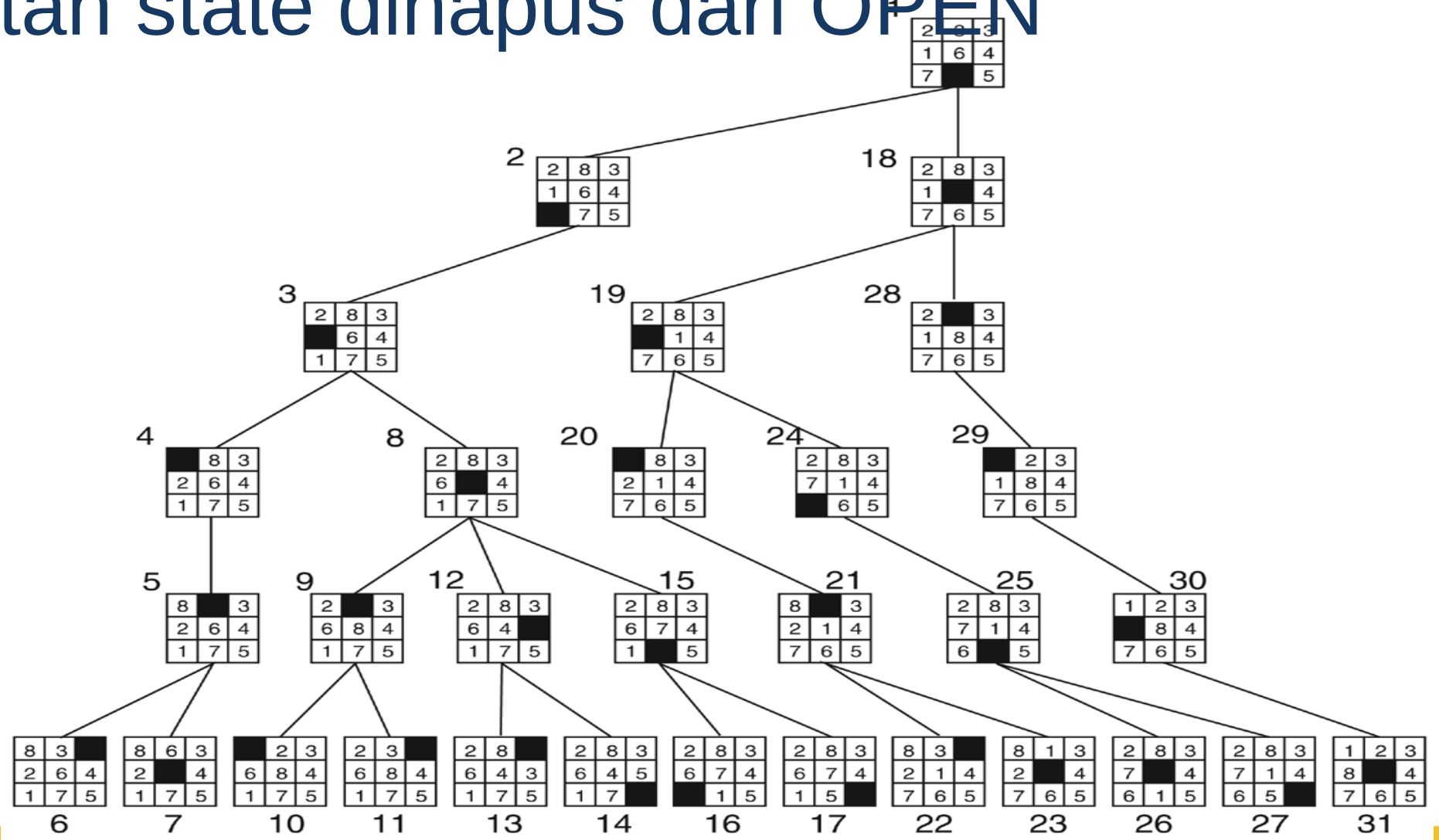
Iterasi 6 pada DFS



BFS, label = urutan state yang dihapus dari OPEN



DFS dengan batas kedalaman 5, label = urutan state dihapus dari OPEN



Pencarian Blind/Sederhana

- 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

Strategi Pencarian (*Search Strategy*)

- Strategi pencarian didefinisikan dengan memilih urutan perluasan node.
- Strategi dievaluasi sebagai berikut:
  - completeness: apakah selalu menemukan solusi jika solusi itu ada?
  - time complexity: jumlah node yang dihasilkan
  - space complexity: jumlah maksimum node di memori
  - optimality: apakah selalu menemukan solusi dengan biaya paling rendah?
- Kompleksitas waktu dan ruang diukur dari segi
  - b: faktor percabangan maksimum dari search tree
  - d adalah kedalaman solusinya
  - m adalah kedalaman maksimum dari tree

Evaluasi untuk Algoritma BFS

- 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)

Evaluasi untuk Algoritma DFS

- 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*

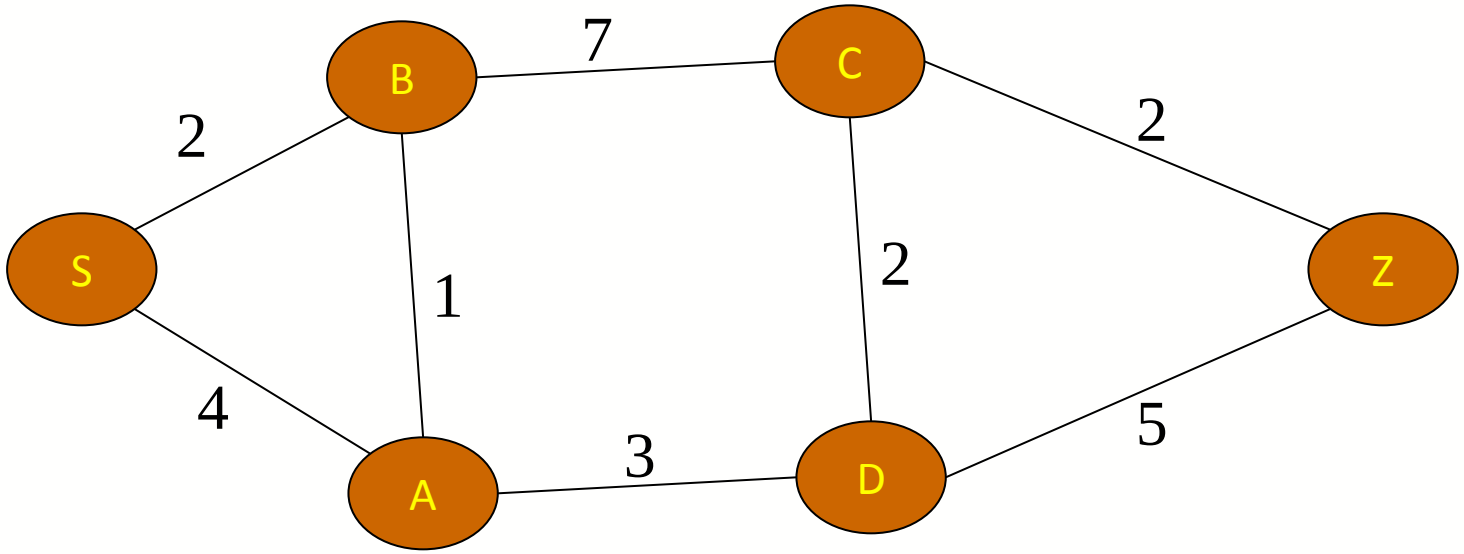
Kelebihan DFS

- 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

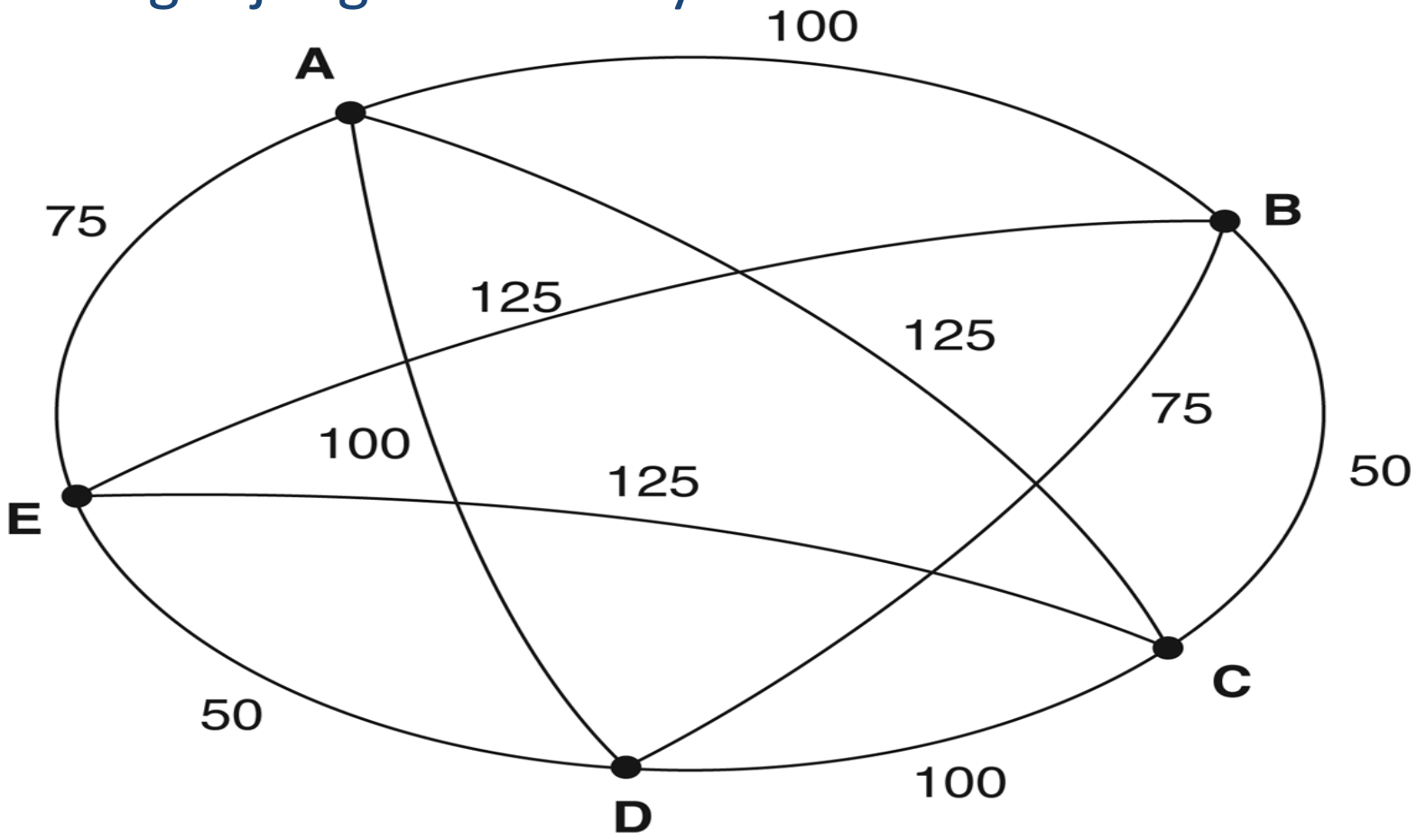
- Memungkinkan tidak ditemukan tujuan yang diharapkan
- Hanya akan mendapatkan 1 solusi pada setiap pencarian

Tugas 1



Tugas 2
Traveling Salesperson Problem

- Dari kota A mengunjungi kota lainnya dan kembali ke kota A



Tugas

Selesaikan Tugas 1 dan Tugas 2:

- Representasikan kasus diatas dengan tree
- Selesaikan kasus diatas dengan metode:
  - Breadth First Search
  - Depth First Search

bridge to the future

http://www.eepis-its.edu