

Bermanfaat
dan Mendunia



Universitas Padjadjaran

Searching: BFS dan DFS

Dr. Intan Nurma Yulita, M.T

unpad.ac.id



Tentang Saya



Dr. Intan Nurma Yulita, M.T

Dosen Departemen Ilmu Komputer Universitas Padjadjaran

Ketua Pusat Riset Kecerdasan Artifisial dan Big Data Universitas Padjadjaran (2021-2023)

Presiden Indonesian Association for Pattern Recognition (INAPR) (2023-sekarang)

Kepala Pusat Inovasi Pengajaran dan Pembelajaran Universitas Padjadjaran (2024-sekarang)

intan.nurma@unpad.ac.id



Teknik Dasar AI

- **Searching (Pencarian)**
AI mencari solusi terbaik dari berbagai kemungkinan.
- **Reasoning (Penalaran)**
AI menganalisis informasi untuk mengambil keputusan.
- **Planning (Perencanaan)**
AI menyusun langkah-langkah untuk mencapai tujuan.
- **Learning (Pembelajaran)**
AI belajar dari data untuk meningkatkan kinerjanya.



Apa itu Algoritma Pencarian?

Algoritma pencarian digunakan dalam kecerdasan buatan (AI) untuk menemukan solusi dari suatu permasalahan dengan menjelajahi ruang keadaan (*state space*).

Secara umum, terdapat dua kategori pencarian:

1. **Pencarian Buta (Blind Search)**

Tidak memiliki informasi tambahan tentang jalur terbaik menuju solusi selain ruang keadaan yang didefinisikan.

2. **Pencarian Heuristik (Informed Search)**

Menggunakan fungsi heuristik untuk mempercepat pencarian solusi.



Konsep Dasar Pencarian Buta

Pencarian buta tidak memiliki pengetahuan tentang jalur mana yang lebih baik, sehingga eksplorasi dilakukan secara sistematis.

Algoritma	Strategi	Struktur Data	Keunggulan	Kelemahan
BFS	Menelusuri semua node pada level yang sama sebelum turun ke level berikutnya	Queue (FIFO)	Selalu menemukan solusi optimal jika bobot langkah seragam	Konsumsi memori besar karena menyimpan semua node di antrian
DFS	Menelusuri satu cabang pohon pencarian hingga dalam, lalu kembali ke atas jika buntu	Stack (LIFO)	Konsumsi memori lebih kecil dibanding BFS	Bisa masuk ke pencarian tak terbatas (infinite loop) jika tidak dibatasi



Representasi Permasalahan dalam AI

Permasalahan dapat direpresentasikan dalam bentuk **graf** atau **pohon keputusan**, yang terdiri dari:

- **State (keadaan):** Kondisi sistem pada suatu waktu tertentu.
- **Operator:** Langkah yang memungkinkan perpindahan dari satu state ke state lainnya.
- **State awal:** Kondisi awal permasalahan.
- **State tujuan:** Kondisi akhir yang ingin dicapai.
- **Ruang keadaan:** Kumpulan semua kemungkinan state yang dapat dicapai dari state awal menggunakan operator.



Studi Kasus: Petani, Serigala, Domba, dan Kubis









<https://plastelina.net/wolf-sheep-cabbage-fullscreen/>



Studi Kasus: Petani, Serigala, Domba, dan Kubis

Seorang **petani(farmer)** harus menyeberangkan **serigala (wolf)**, **domba (sheep)**, dan **kubis (cabbage)** ke seberang sungai menggunakan perahu kecil. Namun, ada beberapa **kendala**:

- Jika **serigala dan domba** ditinggalkan bersama, **serigala akan memakan domba**.
  
- Jika **domba dan kubis** ditinggalkan bersama, **domba akan memakan kubis**.   
- Perahu hanya bisa membawa **satu objek** selain petani dalam sekali perjalanan.

Pertanyaannya: Bagaimana cara menyeberangkan semua dengan aman ke seberang sungai?



Representasi Masalah dalam AI

State (Keadaan)

Setiap **state** direpresentasikan sebagai (F, W, S, C) , di mana:

- F = posisi **Petani** (0 = kiri, 1 = kanan)
- W = posisi **Serigala** (0 = kiri, 1 = kanan)
- S = posisi **Domba** (0 = kiri, 1 = kanan)
- C = posisi **Kubis** (0 = kiri, 1 = kanan)

State Awal: $(0, 0, 0, 0)$ → Semua di sisi kiri

State Tujuan: $(1, 1, 1, 1)$ → Semua di sisi kanan



Aturan Perpindahan (Operators)

Petani memiliki beberapa pilihan dalam setiap langkah:

1. Menyeberang sendiri
2. Menyeberang dengan serigala (🐺)
3. Menyeberang dengan domba (🐑)
4. Menyeberang dengan kubis (🥬)

Namun, perpindahan harus menghindari kondisi berbahaya seperti:

- $(0, 0, 1, 1) \rightarrow$ Domba dan kubis bersama tanpa petani ❌
- $(0, 1, 1, 0) \rightarrow$ Serigala dan domba bersama tanpa petani ❌



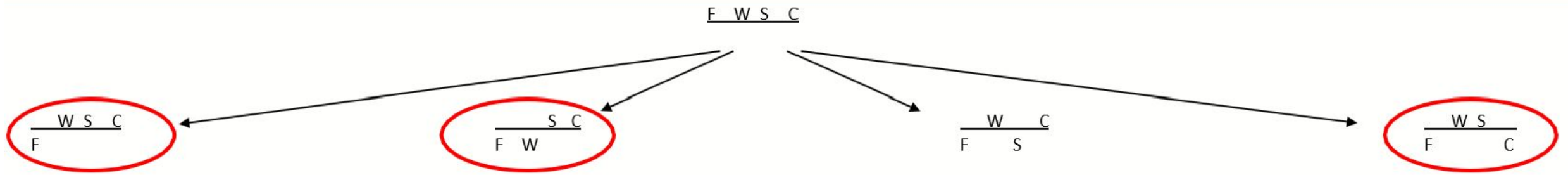
Graf Pencarian

Graf dapat digambarkan sebagai **pohon pencarian**, di mana setiap node adalah state dan setiap edge adalah aksi perpindahan.

State (F, W, S, C)	Aksi yang Dilakukan
(0,0,0,0)	Petani membawa domba ke kanan
(1,0,1,0)	Petani kembali sendiri ke kiri
(0,0,1,0)	Petani membawa kubis ke kanan
(1,0,1,1)	Petani membawa domba kembali ke kiri
(0,0,0,1)	Petani membawa serigala ke kanan
(1,1,0,1)	Petani kembali sendiri ke kiri
(0,1,0,1)	Petani membawa domba ke kanan ✓ (SOLUSI) (1,1,1,1)

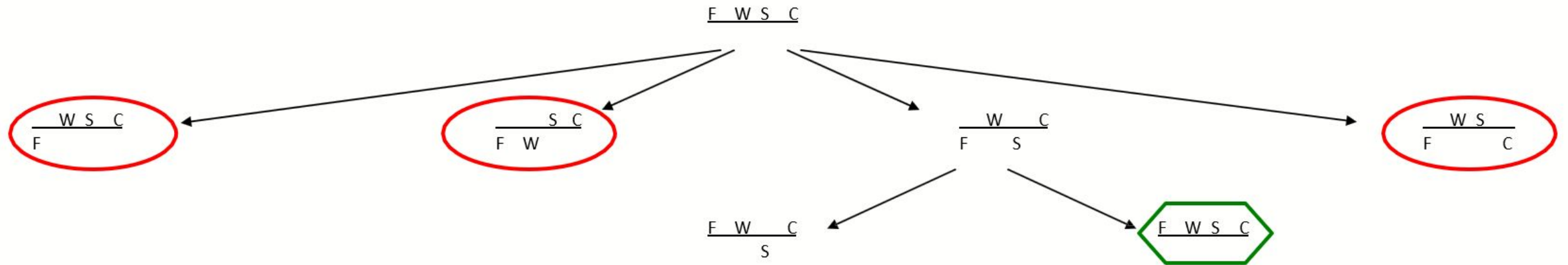


Graf Pencarian



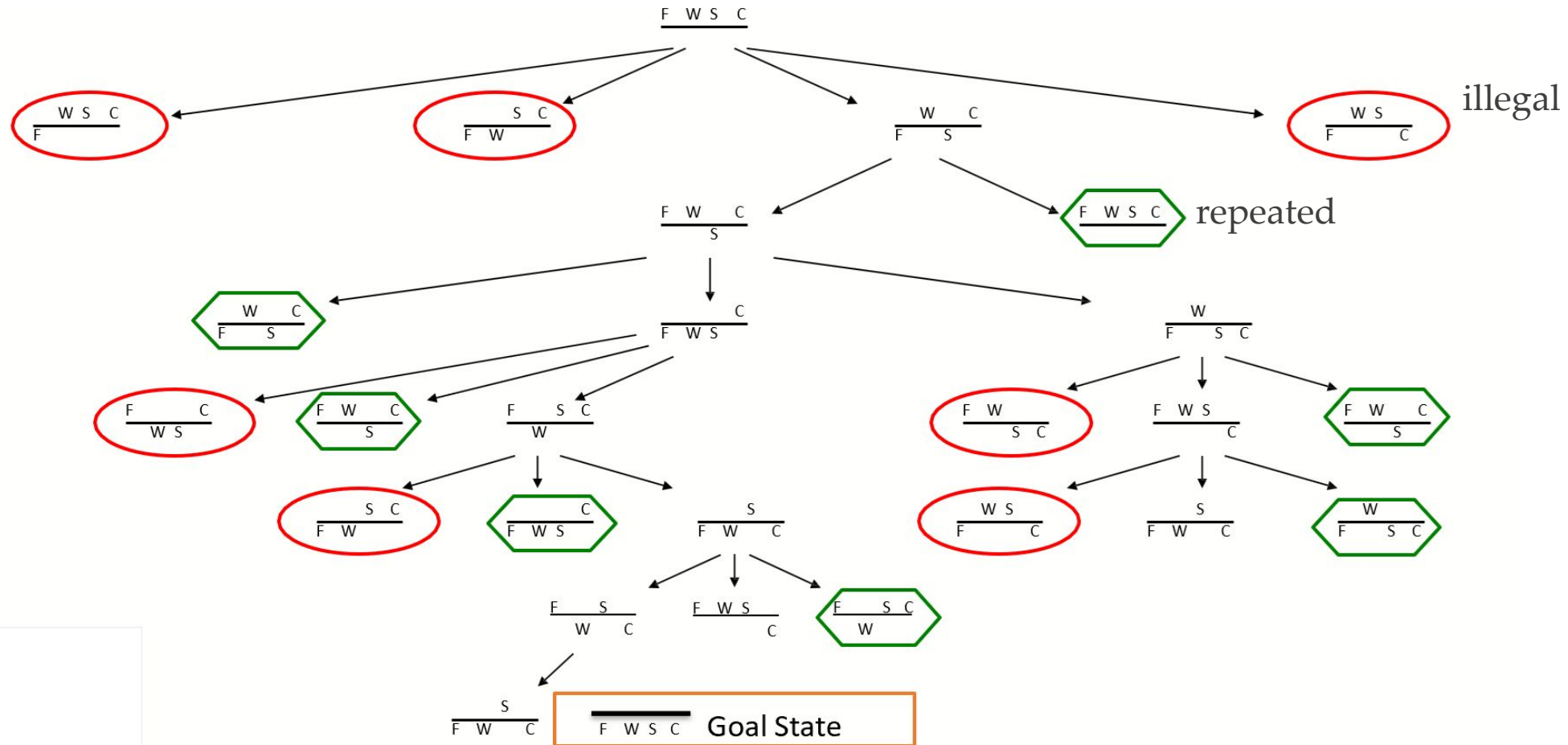


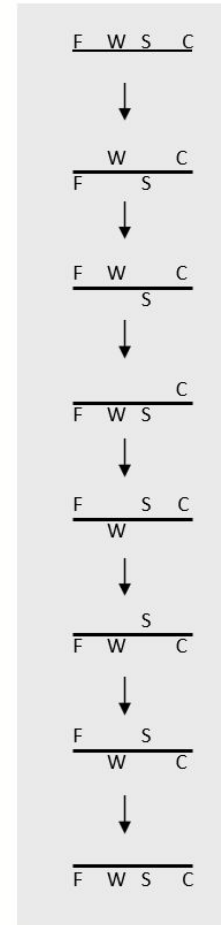
Graf Pencarian





Graf Pencarian







Kasus 2 : Eight Puzzle

1	4	3
7		6
5	8	2

1	4	3
7	6	2
5	8	



Deskripsi Permasalahan

Eight Puzzle adalah sebuah permainan teka-teki berbasis grid 3×3 , di mana terdapat **8 ubin bernomor (1-8)** dan **1 ruang kosong** (biasanya dilambangkan dengan $*$). Tujuan dari permainan ini adalah untuk mengatur ubin sehingga membentuk pola yang diinginkan (*goal state*) dengan cara **menggeser ubin ke ruang kosong**.



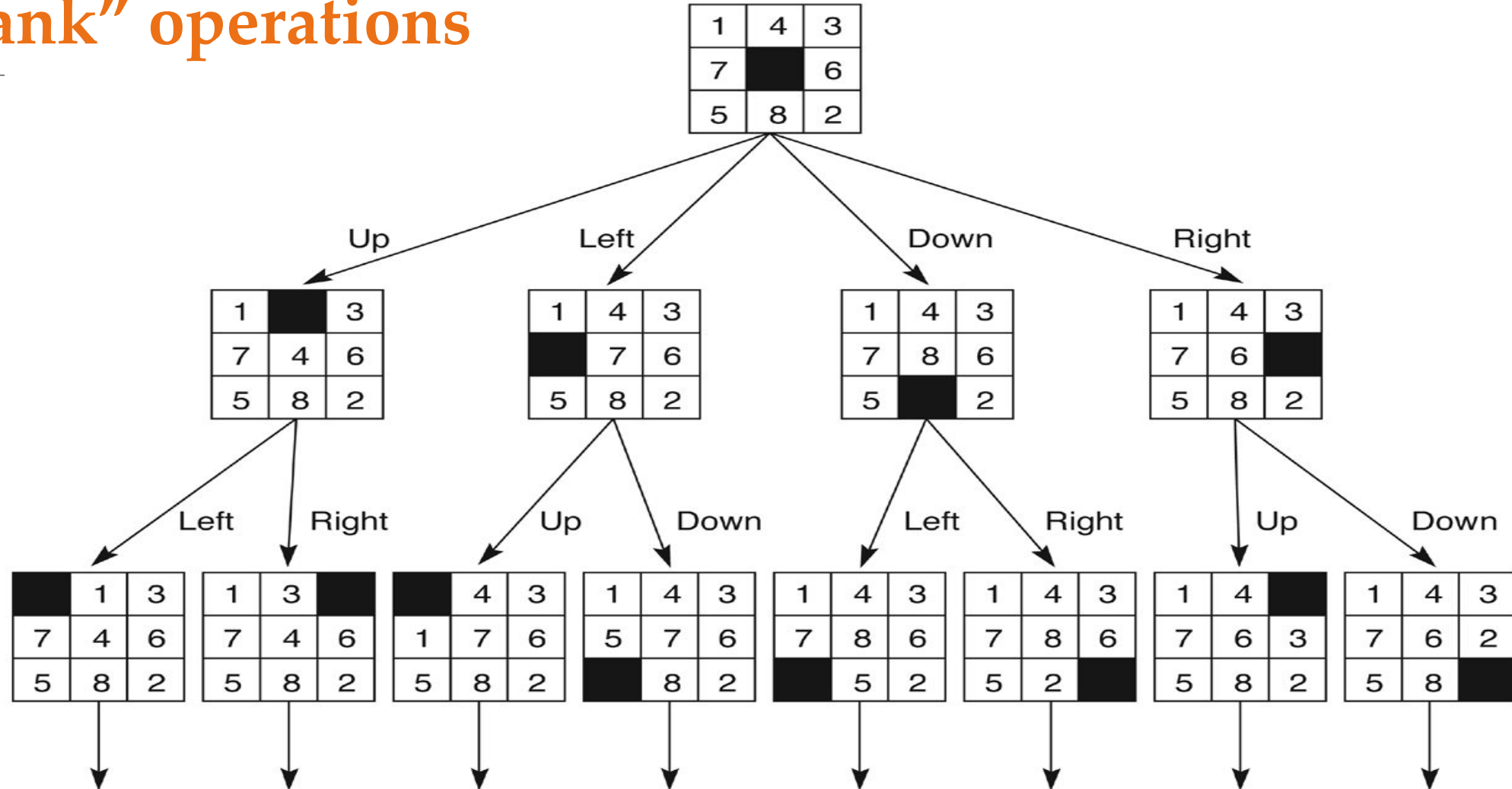
Representasi Masalah dalam AI

Eight Puzzle dapat direpresentasikan dalam bentuk **state space**, di mana:

- **State (Keadaan):** Posisi ubin dalam grid.
- **Operator (Aksi):** Pergerakan ubin ke ruang kosong (atas, bawah, kiri, kanan).
- **State Awal:** Susunan ubin yang diberikan pada awal permainan.
- **State Tujuan:** Susunan ubin yang telah ditentukan sebagai penyelesaian.

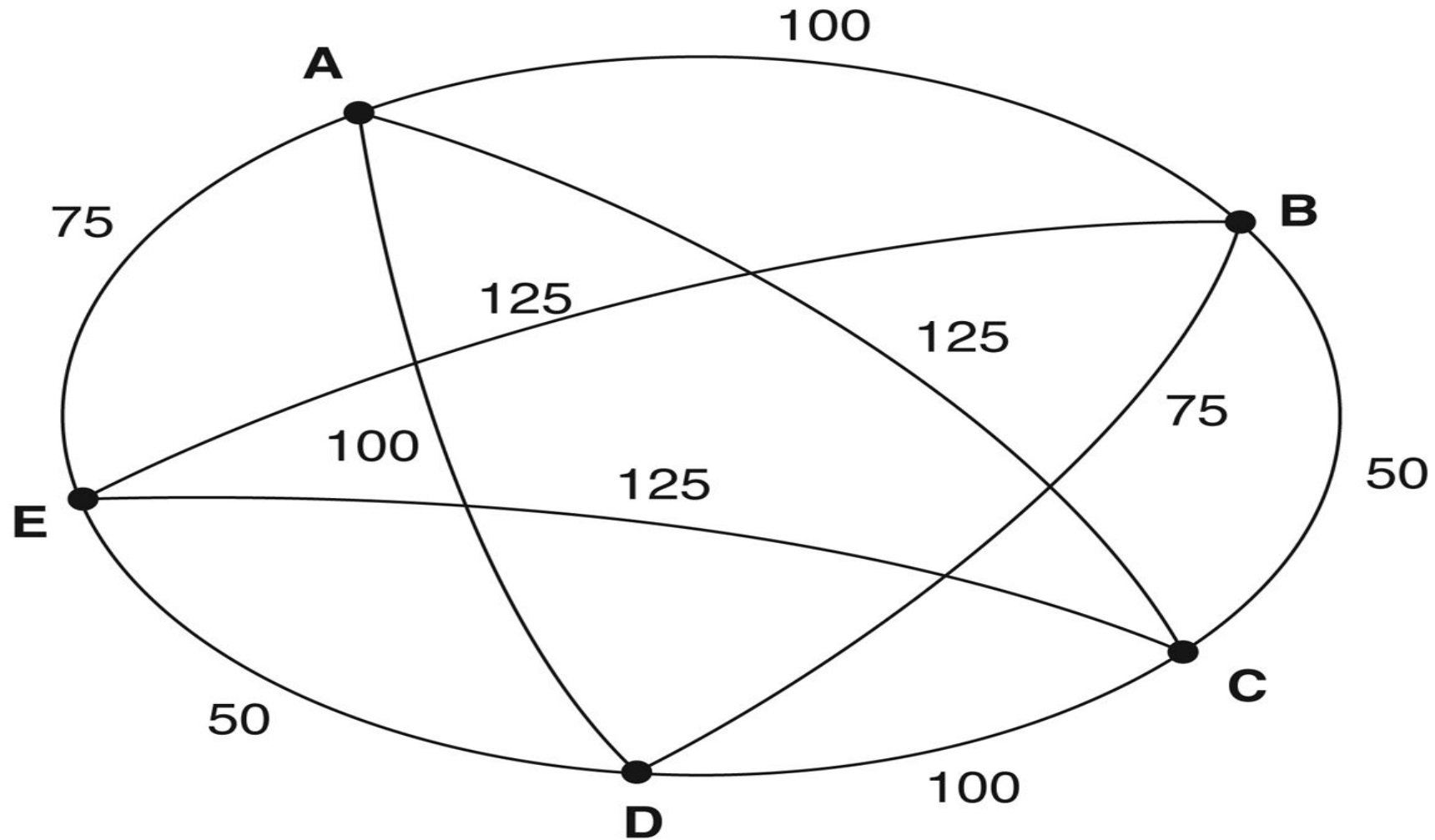


State space of the 8-puzzle generated by “move blank” operations





Kasus 3: Traveling Salesperson Problem





Apa itu Traveling Salesperson Problem (TSP)?

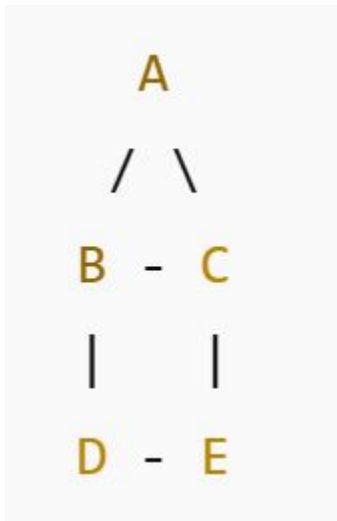
Traveling Salesperson Problem (TSP) adalah permasalahan optimasi klasik dalam ilmu komputer dan matematika kombinatorial. **Seorang salesperson (penjual)** harus mengunjungi **n kota**, mengunjungi setiap kota tepat satu kali, dan kembali ke kota awal dengan jarak atau biaya perjalanan sekecil mungkin.



Representasi dalam AI dan Graf

TSP dapat direpresentasikan sebagai **graf berbobot**:

- **Node (Vertex)** = Kota yang harus dikunjungi.
- **Edge (Sisi)** = Rute antara dua kota dengan bobot (jarak atau biaya).
- **Bobot** = Jarak antara dua kota (biasanya diwakili oleh matriks jarak).

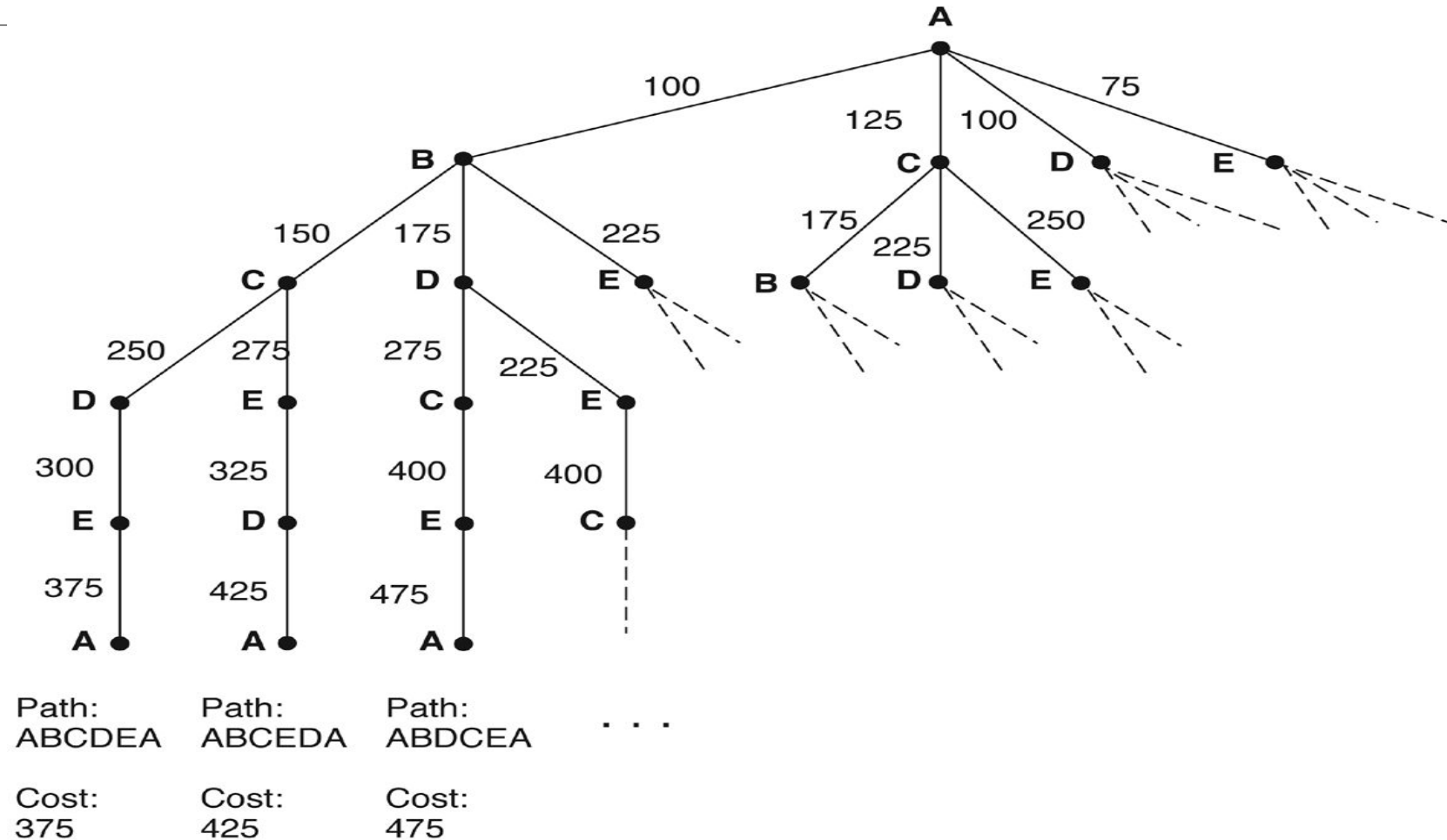


	A	B	C	D	E
A	0	10	15	20	25
B	10	0	35	25	30
C	15	35	0	30	20
D	20	25	30	0	15
E	25	30	20	15	0



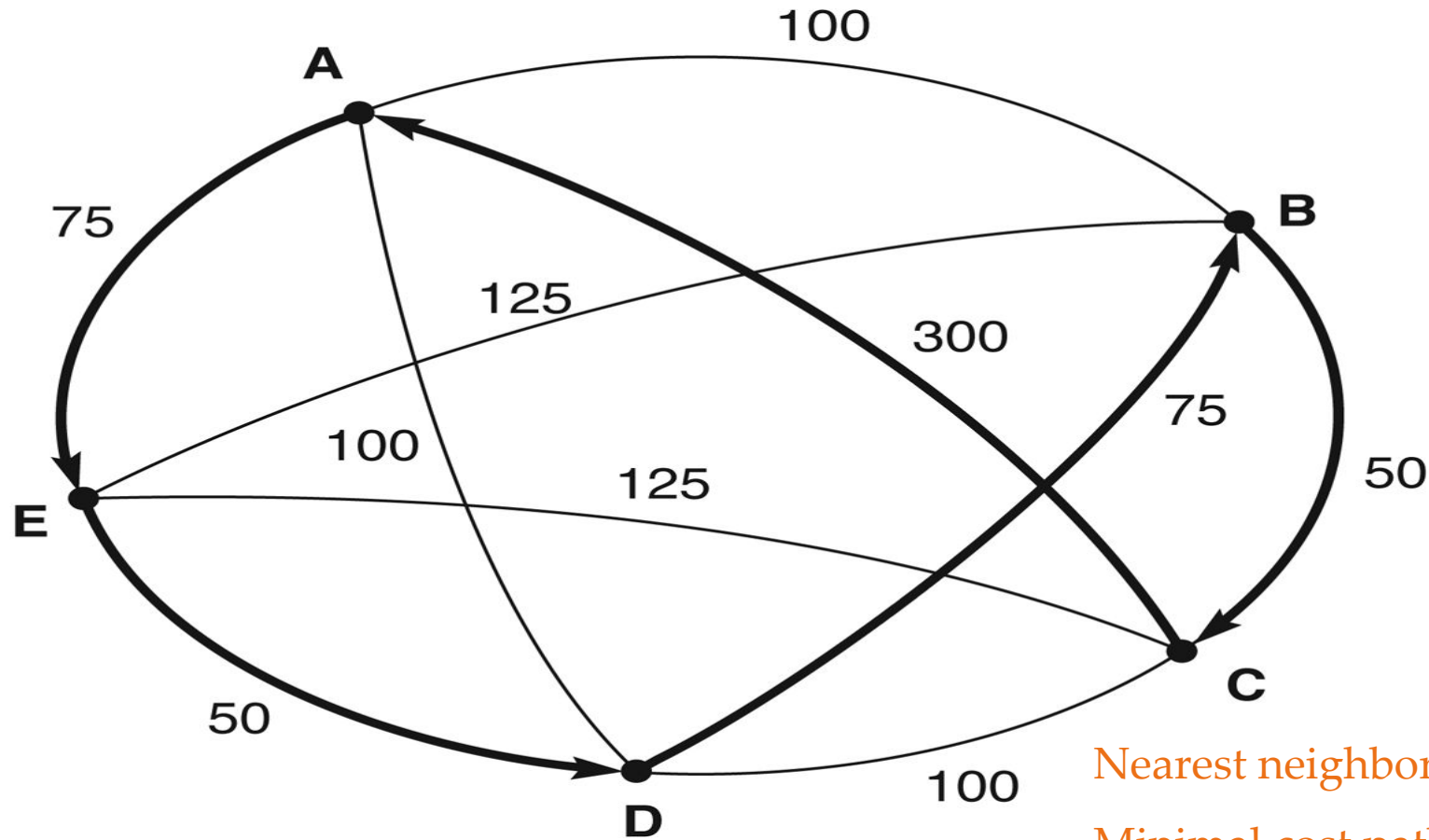
Ruang Pencarian dari TSP

(label pada link = cost dari root)





Nearest neighbor path

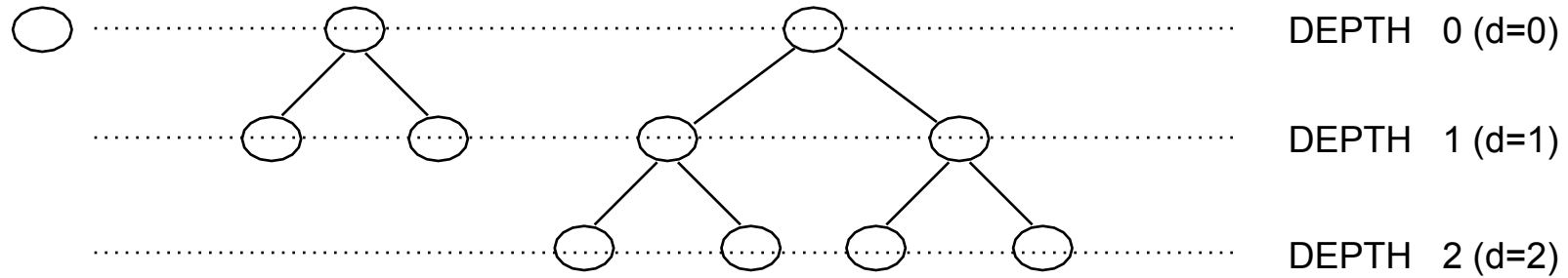


Nearest neighbor path = AEDBCA (550)

Minimal cost path = ABCDEA (375)



Breadth-First Search (BFS)





Algoritma 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, Σ) **then return** x

Insert($x, Closed$)

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

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

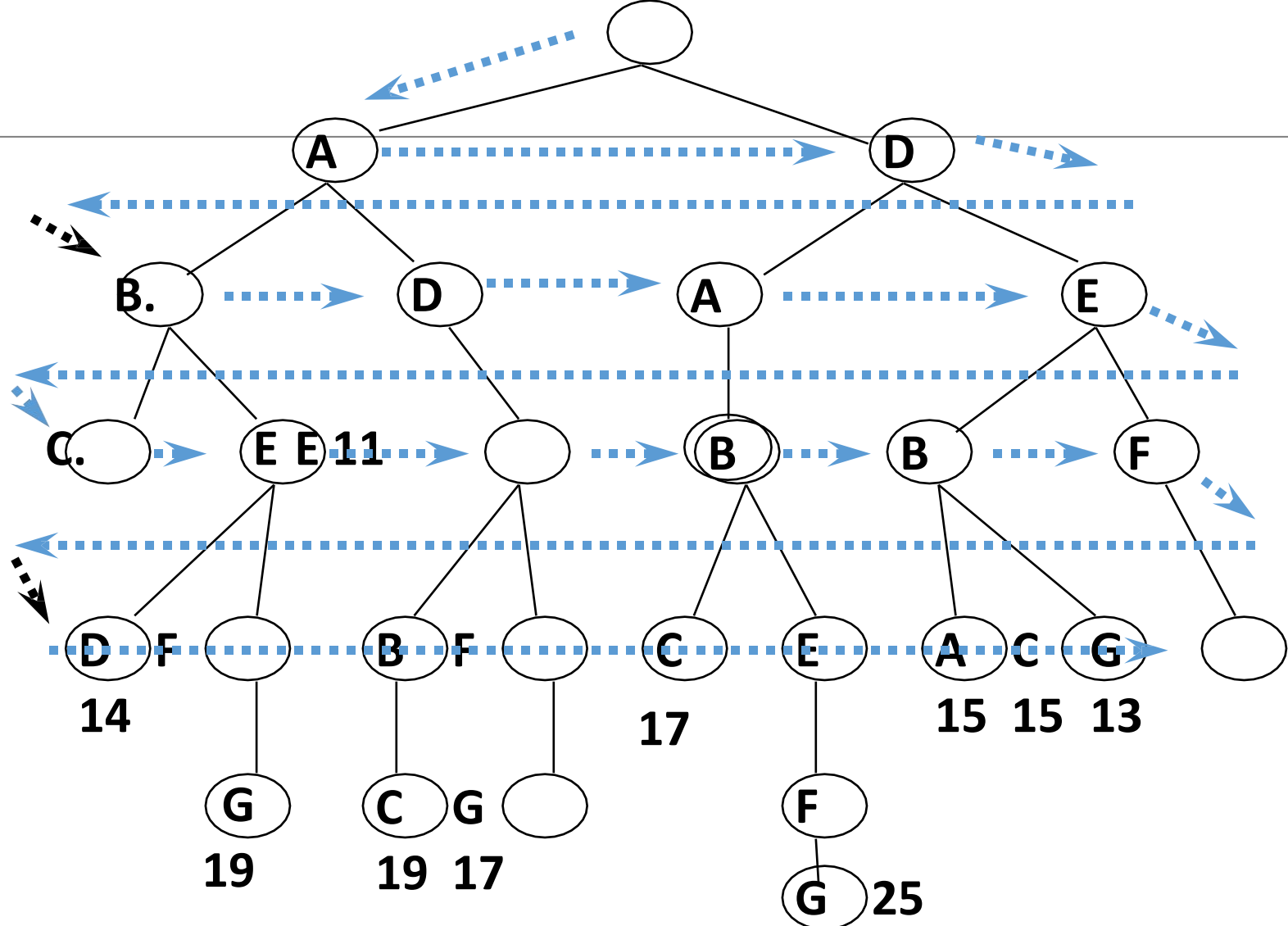
EnQueue($y, Open$)

return *fail*

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

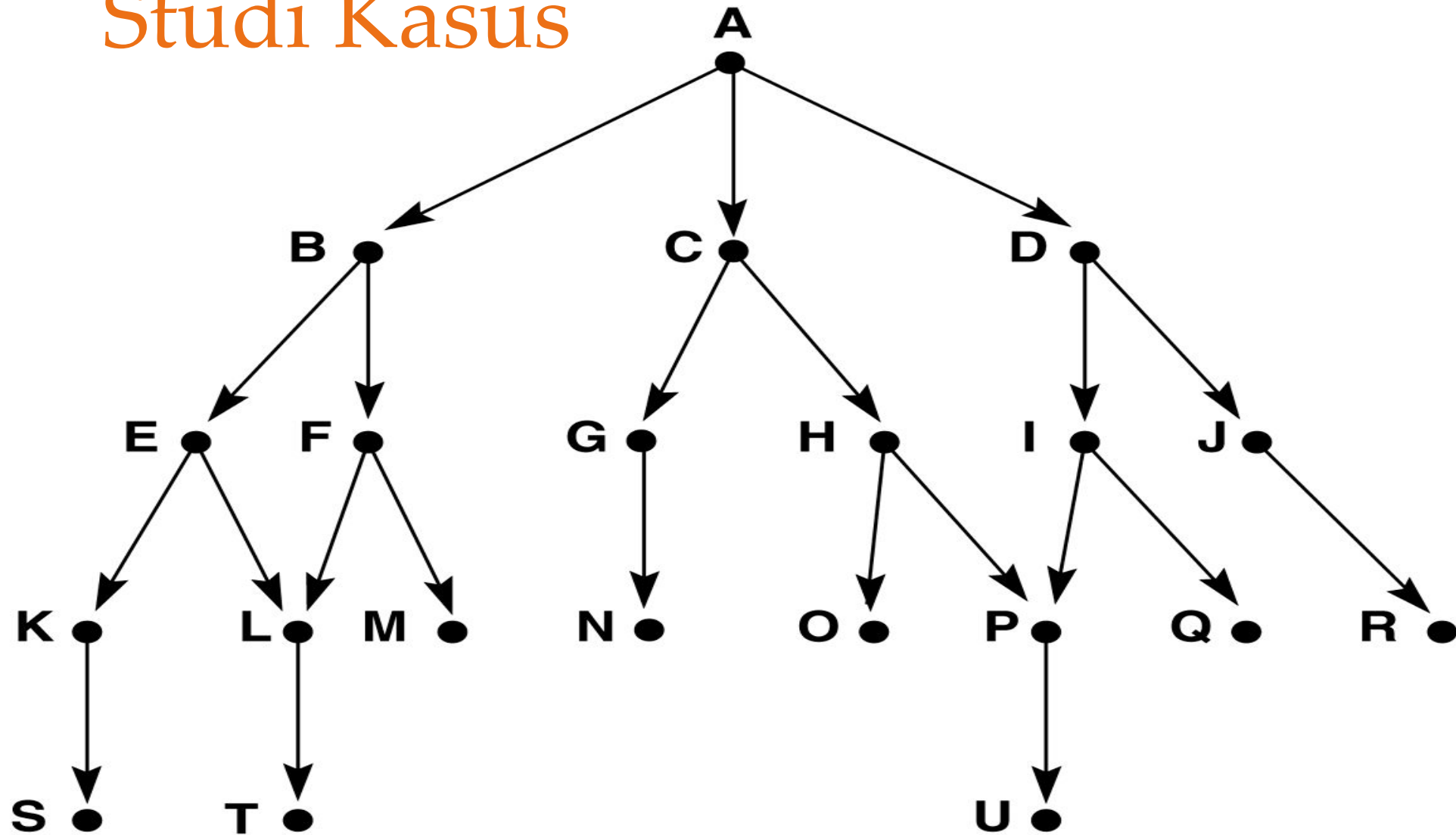


Breath-first search





Studi Kasus

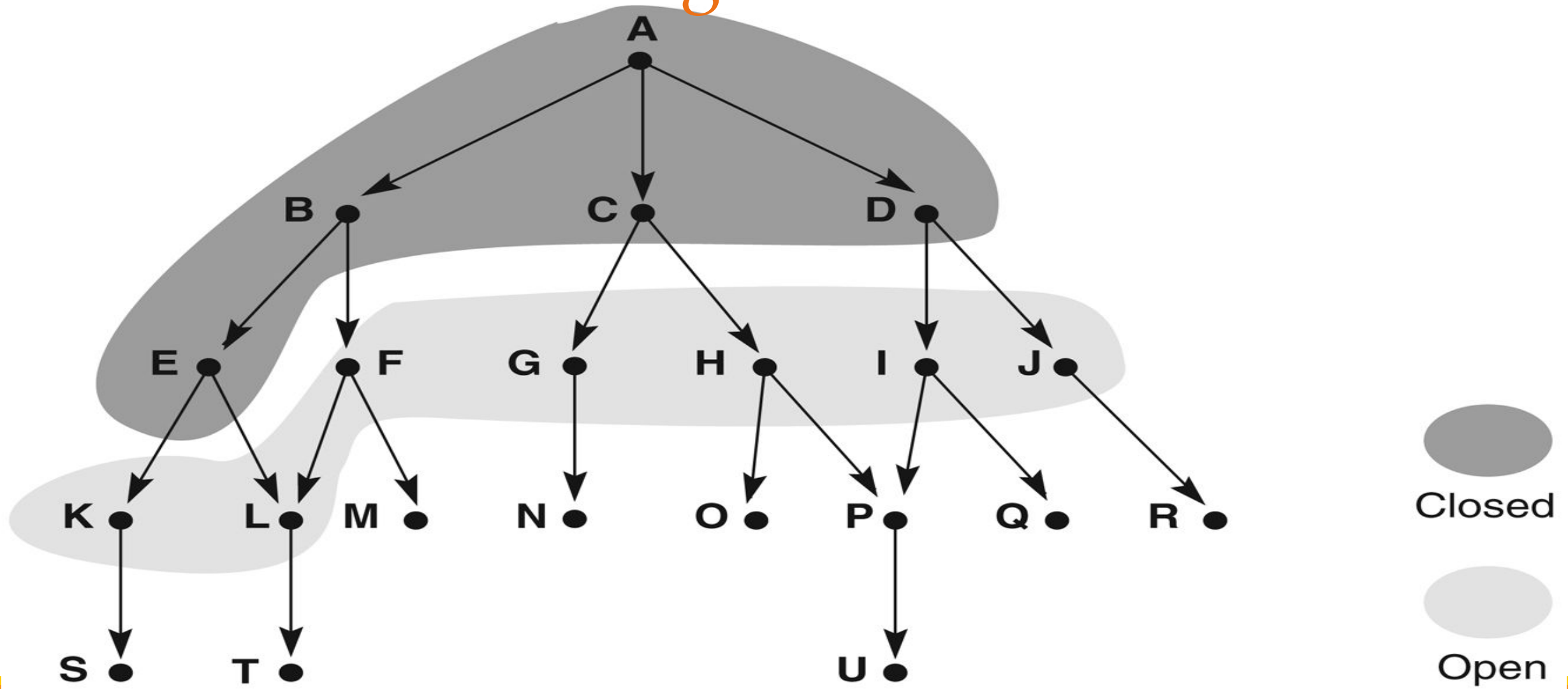


Trace dengan BFS

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** = []

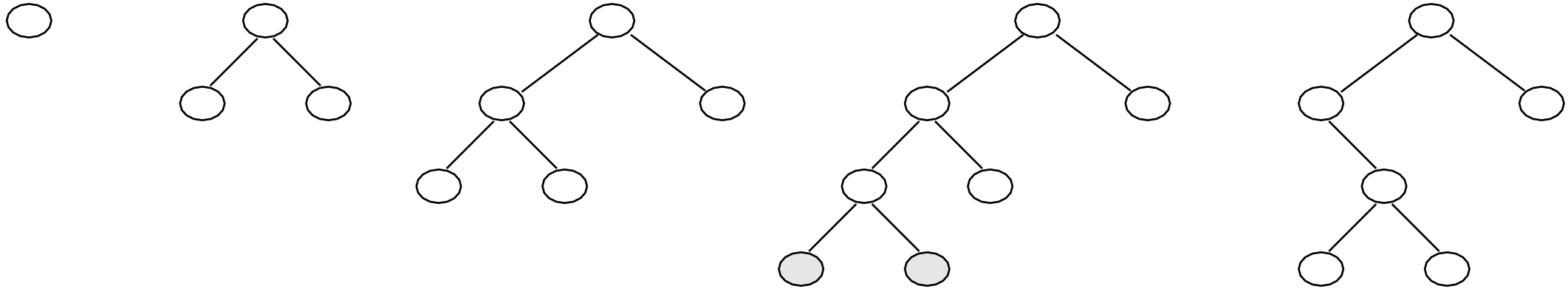


Pada Iterasi 6 dengan BFS





Depth-First Search





Algoritma DFS

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

$\{I\}$

Closed $\leftarrow \emptyset$

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

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

if *Goal*(x, Σ) **then return** x

Insert(x, Closed)

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

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

Push(y, Open)

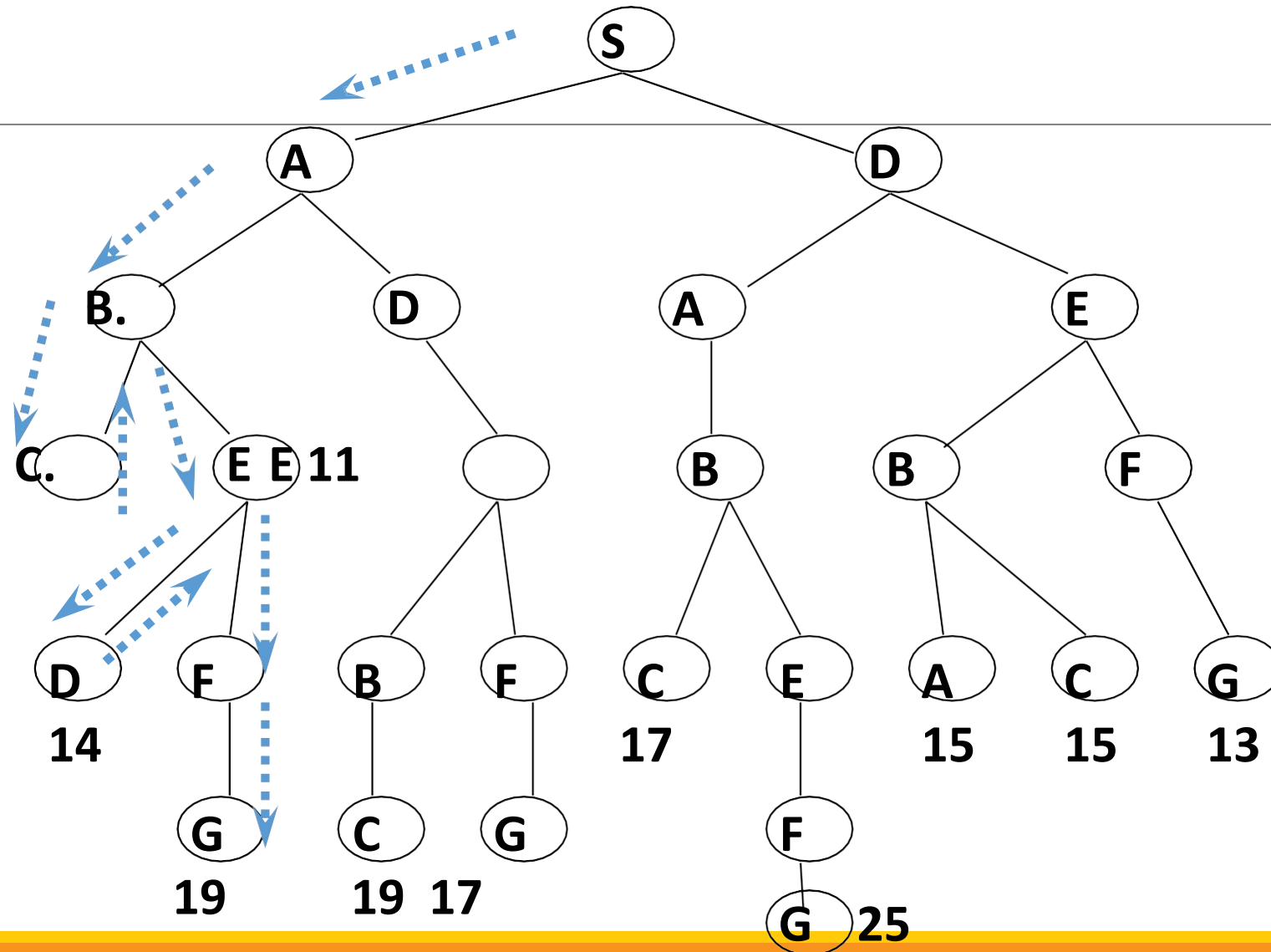
return *fail*

Open diimplementasikan dengan Stack, menerapkan konsep LIFO.

Closed □ langsung ditampilkan ke layar.

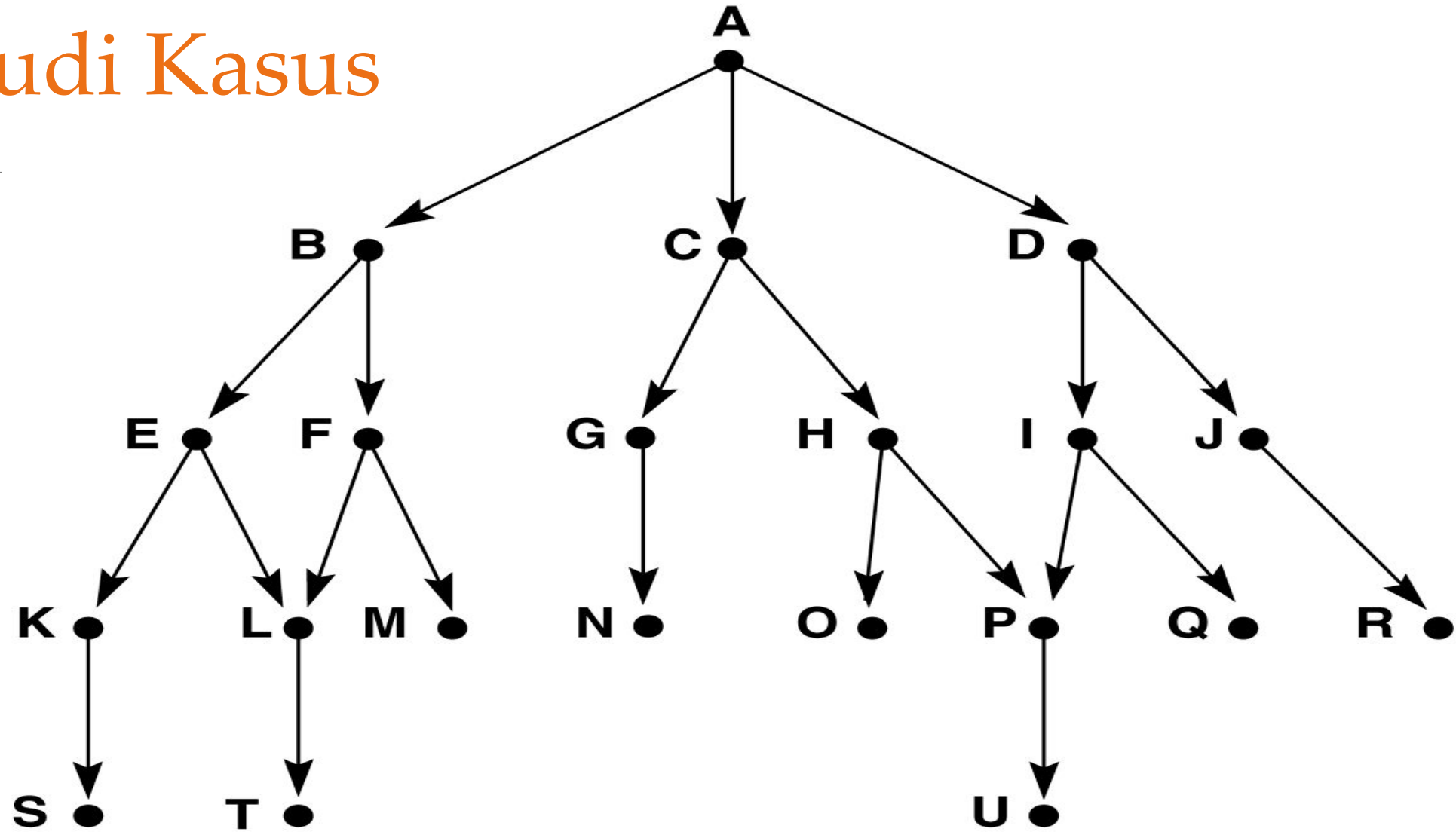


Depth-first search





Studi Kasus



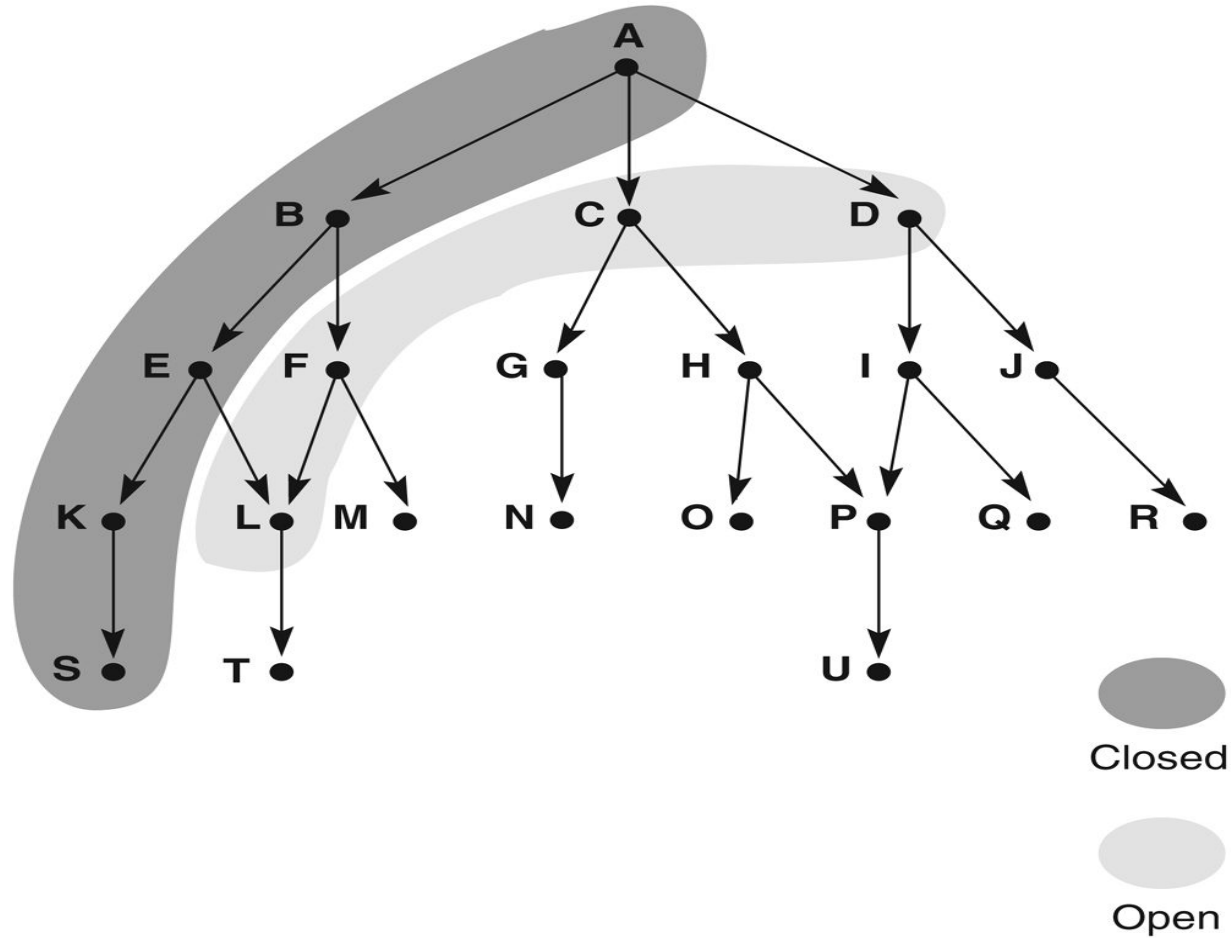


Trace of DFS

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]**

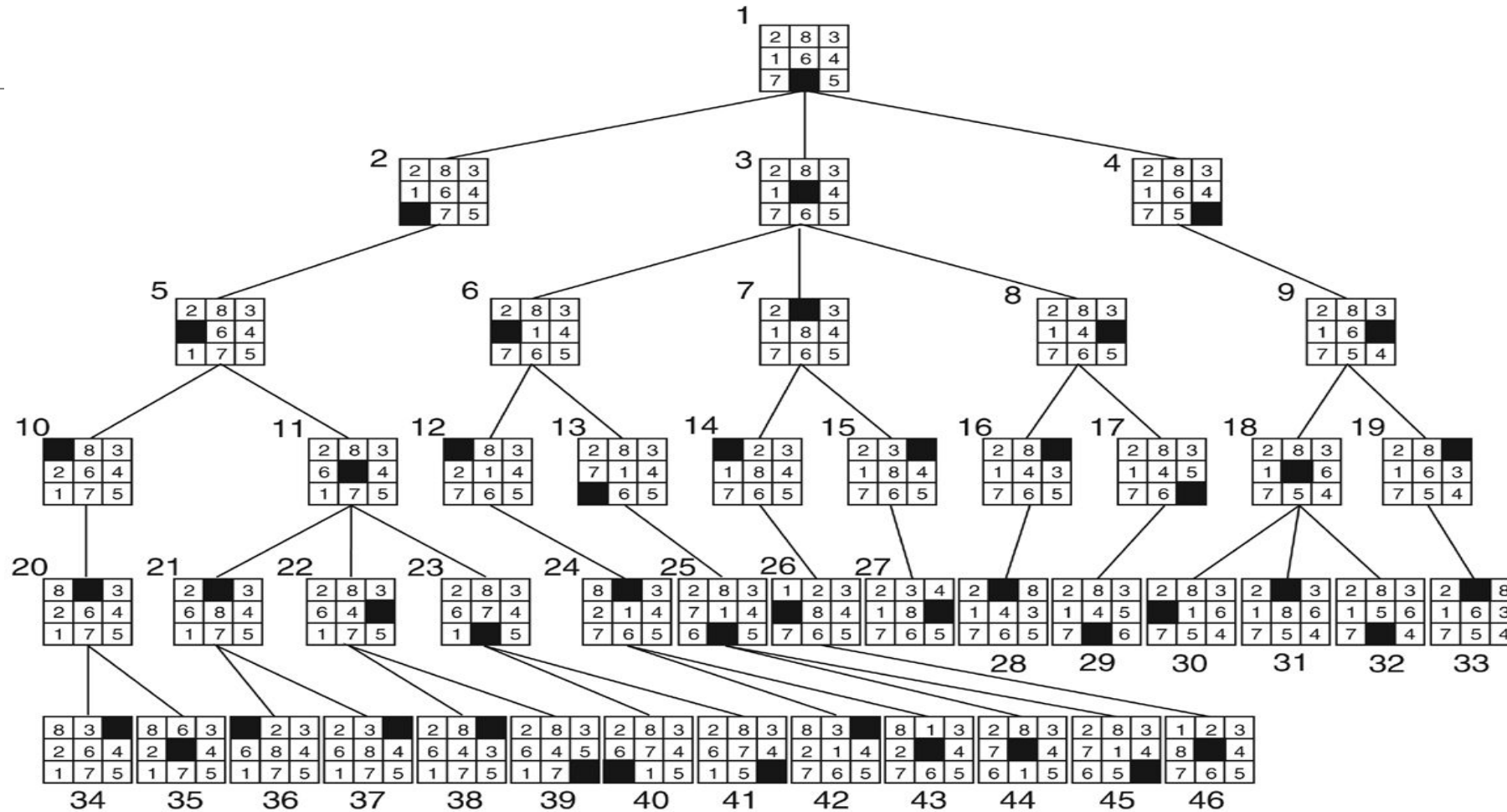


Iterasi 6 pada DFS





BFS, label = urutan state yang dihapus dari OPEN

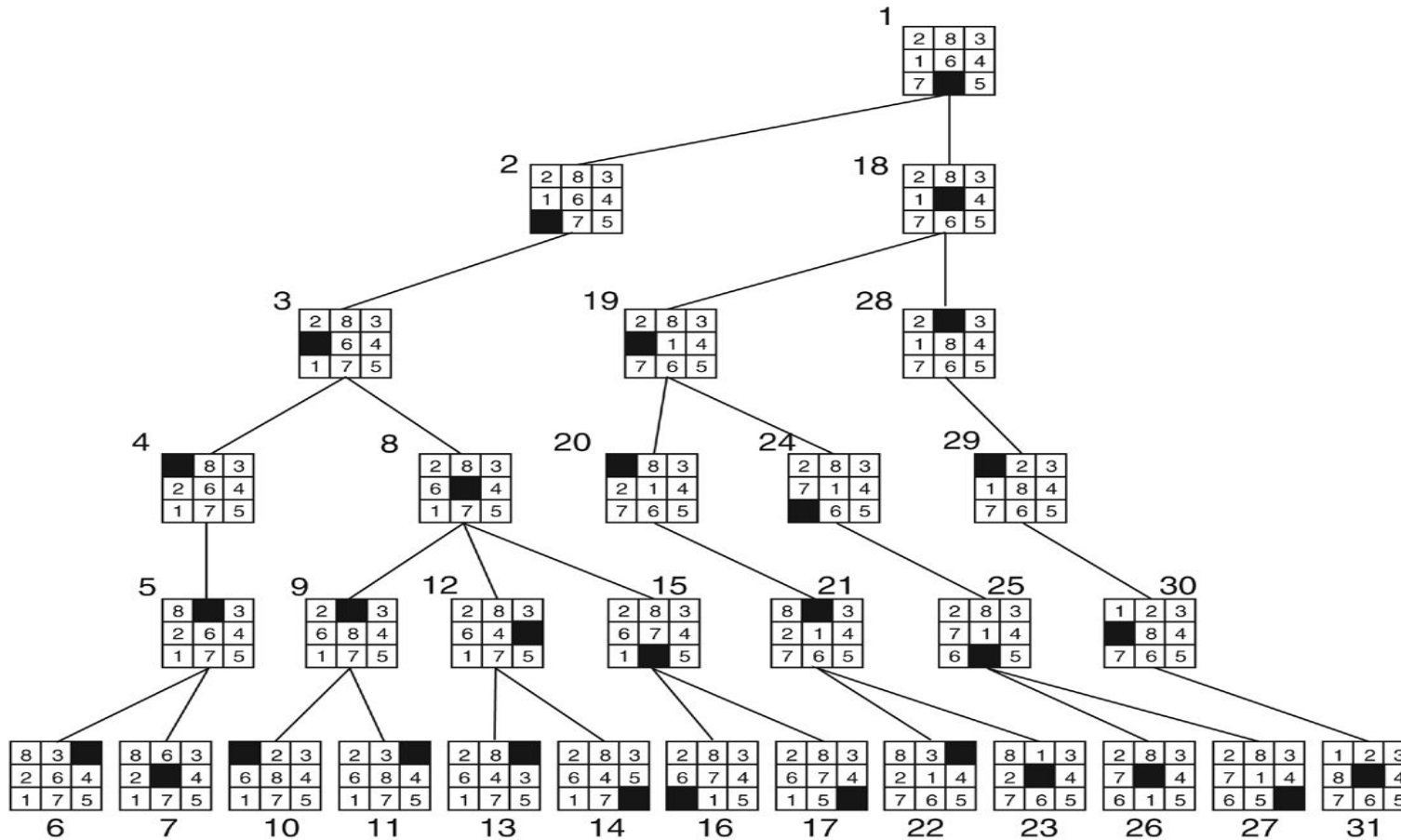


Goal



DFS dengan batas kedalaman 5,

label = urutan
state dihapus dari
OPEN





BFS VS DFS

Kriteria	Breadth-First Search (BFS)	Depth-First Search (DFS)
Strategi Pencarian	Menelusuri semua node pada satu level sebelum turun ke level berikutnya.	Menelusuri satu cabang pohon pencarian hingga dalam sebelum kembali ke node sebelumnya.
Struktur Data	Menggunakan Queue (FIFO).	Menggunakan Stack (LIFO).
Completeness (Kelengkapan)	✓ Selalu menemukan solusi jika ada.	✗ Tidak selalu menemukan solusi, bisa masuk dalam loop tak terbatas jika tidak dikontrol.
Optimality (Keoptimalan)	✓ Optimal jika semua langkah memiliki bobot yang sama.	✗ Tidak selalu optimal karena bisa menemukan solusi di jalur yang lebih panjang.
Time Complexity (Kompleksitas Waktu)	$O(b^d)$, di mana b adalah faktor percabangan dan d adalah kedalaman solusi.	$O(b^m)$, di mana m adalah kedalaman maksimum pencarian.
Space Complexity (Kompleksitas Memori)	$O(b^d)$, membutuhkan memori besar karena menyimpan semua node di setiap level.	$O(bm)$, lebih hemat memori karena hanya menyimpan node dalam jalur aktif.
Keunggulan	<ul style="list-style-type: none">- Selalu menemukan solusi jika ada.- Cocok untuk pencarian solusi terpendek.	<ul style="list-style-type: none">- Lebih efisien dalam memori.- Cocok untuk masalah yang memiliki banyak solusi dan solusinya dalam.
Kelemahan	<ul style="list-style-type: none">- Membutuhkan banyak memori.- Tidak efisien jika ruang pencarian sangat besar.	<ul style="list-style-type: none">- Bisa masuk dalam loop tak terbatas jika tidak dikontrol.- Tidak menjamin solusi optimal.



Latihan: Kasus Cannibals dan Missionaries



<https://plastelina.net/cannibals-missionaries-fullscreen/>



Latihan: Kasus Cannibals dan Missionaries

Deskripsi Permasalahan

Tiga misionaris dan tiga kanibal berada di satu sisi sungai, dan mereka harus menyeberang ke sisi lain menggunakan perahu yang hanya bisa membawa **maksimal dua orang dalam sekali perjalanan**.

Aturan:

- Perahu harus selalu berisi minimal **satu orang saat berlayar**.
- **Jumlah kanibal tidak boleh lebih banyak daripada jumlah misionaris** di sisi mana pun, jika tidak, kanibal akan memakan misionaris.

Implementasikan BFS dan DFS dalam Python untuk menemukan solusi dengan jumlah langkah minimum.



TERIMA KASIH