# Pertemuan 4

# Pencarian dalam Ruang Keadaan (*State Space Search*)

## A. Konsep State, Operator, dan Goal

### 1. Pengantar

Dalam kecerdasan buatan, pencarian (*search*) adalah salah satu pendekatan utama untuk menyelesaikan masalah ketika solusi tidak diketahui secara langsung, tetapi dapat ditemukan melalui proses eksplorasi ruang kemungkinan. *State Space Search* berarti menelusuri semua keadaan (*state*) yang mungkin untuk menemukan urutan tindakan (*operator*) yang membawa sistem dari keadaan awal menuju keadaan tujuan (*goal*).

### 2. Definisi Dasar

| Istilah | Arti dalam Konteks AI |
|---|---|
| State (Keadaan) | Kondisi sistem atau lingkungan pada suatu waktu tertentu. |
| Initial State (Keadaan Awal) | Titik awal pencarian sebelum ada tindakan dilakukan. |
| Goal State (Keadaan Tujuan) | Kondisi akhir yang ingin dicapai. |
| Operator / Action | Langkah atau aturan yang dapat mengubah suatu keadaan menjadi keadaan lain. |
| Path (Jalur) | Urutan keadaan dari awal hingga mencapai tujuan. |
| Search Space (Ruang Keadaan) | Kumpulan semua kemungkinan keadaan yang dapat terbentuk dari keadaan awal melalui penerapan operator. |

### 3. Konsep "State" (Keadaan)

"State" adalah representasi dari situasi atau kondisi sistem pada satu titik waktu. Setiap *state* menggambarkan apa yang diketahui atau apa yang sedang terjadi, dan menjadi titik dasar untuk menerapkan langkah-langkah selanjutnya (operator).

Contoh 1: Puzzle 8 (8-Puzzle Problem)

- Setiap konfigurasi ubin adalah satu *state*.

- Contoh

```
[1 2 3]
[4 5 6]
[7 8 _]
```

→ Ini adalah satu state.

Jika ubin "8" digeser ke kiri, maka akan terbentuk *state baru*.

Contoh 2: Masalah Rute Kota

- State = posisi saat ini (misal di Kota A).
- Goal = mencapai Kota D.
- Operator = berpindah dari satu kota ke kota lain melalui jalan yang tersedia.

## 4. Konsep "Operator" (Operator / Action)

Operator adalah tindakan (aksi) yang menyebabkan perubahan dari satu state ke state lainnya. Operator dapat dianggap sebagai "aturan transisi" yang menentukan bagaimana sistem berpindah dari keadaan awal ke keadaan baru.

Contoh Kasus:

Contoh 1 - Puzzle 8

- Operator: Geser ubin ke kiri, kanan, atas, atau bawah.
- Aturan transisi:
- `State S1 → (Geser ubin kiri) → State S2`

Contoh 2 — Masalah Rute

- Operator: Bergerak dari satu kota ke kota lain.
- Jika kita berada di "Kota A":
  - Operator 1: Pergi ke B
  - Operator 2: Pergi ke C
  - Masing-masing menghasilkan *state baru*.

## 5. Konsep "Goal" (Tujuan)

Goal adalah kondisi akhir yang diinginkan atau solusi dari masalah. Proses pencarian akan berhenti ketika sistem mencapai *state* yang memenuhi kriteria goal.



Dalam pencarian, kita perlu fungsi khusus untuk memeriksa apakah suatu state adalah goal:

**Goal Test → True/False**

**Contoh Kasus:**

Puzzle 8

Goal:

```
[1 2 3]
[4 5 6]
[7 8 _]
```

Jika konfigurasi saat ini sama seperti di atas → solusi ditemukan.

Masalah Rute Kota

- Goal: mencapai kota "D".

  Jika posisi saat ini = D → pencarian selesai.

### 6. Hubungan antara State, Operator, dan Goal

Proses pencarian dapat digambarkan sebagai pergerakan dalam ruang keadaan, dari *initial state* menuju *goal state* dengan menerapkan berbagai *operator.*

**Skema Hubungan:**

```
Initial State
   │
   │ apply Operator 1
   ▼
 Intermediate State
   │
   │ apply Operator 2
   ▼
Goal State (Selesai)
```

Contoh Nyata

Masalah: Menyalakan lampu di ruangan.

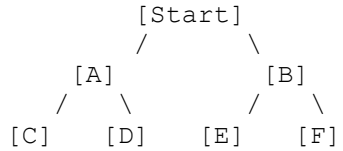
| Komponen | Contoh |
|---|---|
| State Awal | Ruangan gelap (lampu mati) |
| Operator | Menekan saklar |
| State Baru | Lampu menyala |
| Goal State | Ruangan terang (lampu hidup) |

## 7. Representasi Formal Ruang Keadaan

Ruang keadaan sering direpresentasikan sebagai graf (*graph*) atau pohon (*tree*).

- Node (simpul) → menunjukkan state.
- Edge (sisi) → menunjukkan operator atau tindakan.

Contoh Representasi Pohon




- Start = state awal
- A, B, C, D, E, F = state hasil penerapan operator
- Goal ditemukan jika salah satu node memenuhi kondisi akhir.

## 8. Contoh Kasus Lengkap — Pencarian Rute Kota

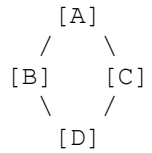
Masalah

Temukan rute dari Kota A ke Kota D.

Model Ruang Keadaan:

| Dari | Ke | Operator (Aksi) |
|---|---|---|
| A | B | Pergi ke B |
| A | C | Pergi ke C |
| B | D | Pergi ke D |
| C | D | Pergi ke D |

Representasi Graf:




Proses:

- Initial State = A
- Goal State = D
- Operator = Pindah ke kota terhubung

Langkah:

```
Mulai di A
→ Operator1: Pergi ke B
→ Operator2: Dari B ke D
→ Goal tercapai!
```

Jalur solusi: A → B → D

### 9. Elemen-Elemen Pencarian Formal dalam AI

| Elemen | Keterangan |
|---|---|
| Initial State | Titik awal pencarian |
| Action(s) | Operator yang dapat diterapkan |
| Transition Model | Fungsi yang menjelaskan hasil penerapan aksi |
| Goal Test | Uji apakah state saat ini adalah goal |
| Path Cost | Total biaya dari langkah awal ke tujuan |
| Solution | Urutan aksi dari awal hingga mencapai goal dengan biaya minimum |

### 10. Analogi Kehidupan Nyata

| Contoh Masalah | State | Operator | Goal |
|---|---|---|---|
| Menyelesaikan Sudoku | Kondisi angka pada papan | Mengisi angka | Papan terisi benar |
| Merencanakan perjalanan | Lokasi saat ini | Berpindah ke kota lain | Sampai di tujuan |
| Memasak resep | Tahapan memasak | Melakukan langkah resep | Hidangan siap disajikan |
| Menyusun jadwal kuliah | Susunan jadwal sementara | Menambah/menghapus mata kuliah | Jadwal optimal dan tidak bentrok |

## B. Strategi Pencarian: Uninformed Search (DFS, BFS)

### 1. Pengantar

Dalam proses pencarian solusi di kecerdasan buatan, kita berhadapan dengan ruang keadaan (*state space*) yang berisi banyak kemungkinan jalur menuju tujuan (*goal*). Untuk menemukan solusi, AI menggunakan strategi pencarian (search strategy). Strategi pencarian menentukan bagaimana sistem menelusuri state-space mulai dari mana, ke arah mana, dan kapan berhenti.

## 2. Jenis Strategi Pencarian

Secara umum strategi pencarian dibagi menjadi dua kategori besar:

| Jenis Pencarian | Ciri Utama |
|---|---|
| Uninformed Search (Blind Search) | Tidak memiliki pengetahuan tambahan tentang lokasi goal; hanya menggunakan struktur ruang keadaan. |
| Informed Search (Heuristic Search) | Menggunakan informasi tambahan (heuristik) untuk memperkirakan jarak ke goal. |

Pada pertemuan ini kita fokus pada Uninformed Search: BFS (Breadth-First Search) dan DFS (Depth-First Search).

## 3. Konsep Dasar Uninformed Search

Uninformed Search disebut juga *blind search*, karena algoritma tidak tahu posisi goal dan harus menelusuri semua kemungkinan secara sistematis.

Sistem hanya mengetahui:

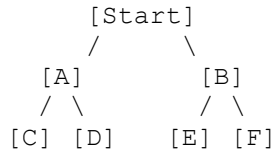
- State awal
- Operator yang dapat diterapkan
- Cara memeriksa apakah state adalah goal (*goal test*)

## 4. Representasi Pencarian: Pohon Keadaan (*State Space Tree*)

Pencarian biasanya digambarkan sebagai **pohon (*tree*)**, di mana:

- Node (simpul) mewakili *state*
- Edge (garis) mewakili *operator* (aksi atau transisi antar state)

Contoh sederhana:




Goal dicapai ketika salah satu node = *goal state*.

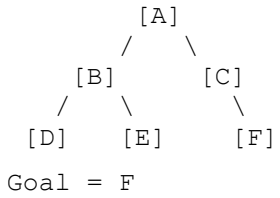
## 5. Breadth-First Search (BFS)

BFS (Pencarian Lebar-Dahulu) adalah metode pencarian yang menelusuri semua node pada level tertentu terlebih dahulu sebelum melanjutkan ke level berikutnya. BFS menggunakan struktur data antrian (*queue*) dengan prinsip FIFO (*First In, First Out*).

Langkah-langkah BFS:

1. Mulai dari node awal (*root*).
2. Tambahkan node awal ke dalam antrian.
3. Selama antrian tidak kosong:
   - Keluarkan node dari antrian (*visit*).
   - Jika node = goal → selesai.
   - Jika belum → masukkan semua anak (*successor*) ke dalam antrian.

Contoh




Goal = F

Langkah BFS:

1. Antrian awal: [A]
2. Kunjungi A → tambah B, C → [B, C]
3. Kunjungi B → tambah D, E → [C, D, E]
4. Kunjungi C → tambah F → [D, E, F]
5. Kunjungi D (bukan goal)
6. Kunjungi E (bukan goal)
7. Kunjungi F (goal ditemukan)

Urutan Kunjungan:

A → B → C → D → E → F

Kelebihan BFS

- Menemukan solusi terpendek (minimum depth).

- Lengkap: akan selalu menemukan solusi jika ada.

Kelemahan BFS

- Menggunakan banyak memori, karena semua node disimpan dalam antrian.
- Tidak efisien untuk ruang keadaan besar (eksplorasi sangat luas).

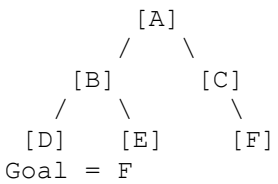
## 6. Depth-First Search (DFS)

DFS (Pencarian Dalam-Dahulu) adalah metode pencarian yang menelusuri node sedalam mungkin terlebih dahulu, baru kemudian menelusuri cabang lainnya. DFS menggunakan struktur data tumpukan (*stack*) dengan prinsip LIFO (*Last In, First Out*).

Langkah-langkah DFS:

1. Mulai dari node awal (root).
2. Tambahkan node ke dalam tumpukan.
3. Selama tumpukan tidak kosong:
   - Ambil node paling atas (pop).
   - Jika node = goal → selesai.
   - Jika belum → masukkan semua anak node ke tumpukan (urutan penting).

Contoh:




Langkah DFS:

1. Tumpukan awal: [A]
2. Kunjungi A → tambah C, B → [C, B]
3. Kunjungi B → tambah E, D → [C, E, D]
4. Kunjungi D → (tidak ada anak) → kembali
5. Kunjungi E → (tidak ada anak) → kembali

6. Kunjungi C → tambah F → [F]
7. Kunjungi F (goal ditemukan)

Urutan Kunjungan:

A → B → D → E → C → F

Kelebihan DFS

- Memori efisien (hanya menyimpan jalur saat ini).
- Cocok untuk ruang keadaan besar.
- Cepat jika solusi berada di cabang terdalam sebelah kiri.

Kelemahan DFS

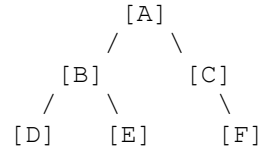
- Tidak menjamin solusi terpendek.
- Dapat terjebak dalam loop (jika graf memiliki siklus).
- Tidak lengkap jika ruang pencarian tak terbatas.

## 7. Perbandingan BFS vs DFS

| Aspek | BFS (Breadth-First Search) | DFS (Depth-First Search) |
|---|---|---|
| Arah Penelusuran | Level demi level (melebar) | Cabang demi cabang (mendalam) |
| Struktur Data | Queue (Antrian) | Stack (Tumpukan) |
| Solusi Terpendek? | Ya | Tidak selalu |
| Kelengkapan | Lengkap | Tidak lengkap |
| Kebutuhan Memori | Besar | Kecil |
| Waktu Eksekusi | Lambat pada pohon besar | Cepat pada jalur dalam |
| Risiko Loop | Kecil | Besar (jika tidak dikontrol) |

## 8. Visualisasi Proses BFS dan DFS

Struktur Pohon Pencarian



BFS Traversal

```
Level 0: A
Level 1: B, C
Level 2: D, E, F
→ Urutan: A, B, C, D, E, F
```

DFS Traversal

```
A → B → D → E → C → F
```

### 9. Analogi Kehidupan Nyata

| Situasi | BFS | DFS |
|---|---|---|
| Mencari teman di kampus | Menemui semua orang di lantai 1 dulu sebelum naik ke lantai 2 | Menelusuri satu koridor sampai ujung sebelum pindah koridor lain |
| Menyusun tugas kuliah | Mengerjakan semua tugas satu per satu | Menyelesaikan satu tugas secara mendalam sampai selesai baru pindah ke lainnya |
| Mencari file di folder | Membuka semua folder satu level dulu | Menelusuri satu folder sampai paling dalam |

## C. Evaluasi Efisiensi Algoritma (Kompleksitas Waktu dan Ruang)

### 1. Pengantar Evaluasi Efisiensi

Dalam kecerdasan buatan, algoritma pencarian sering berhadapan dengan ruang keadaan (*state space*) yang sangat besar. Untuk menilai seberapa baik sebuah algoritma bekerja, diperlukan evaluasi efisiensi, yang diukur melalui:

| Jenis Kompleksitas | Keterangan |
|---|---|
| Kompleksitas Waktu (Time Complexity) | Seberapa cepat algoritma menemukan solusi (jumlah langkah atau simpul yang dievaluasi). |
| Kompleksitas Ruang (Space Complexity) | Seberapa besar memori yang dibutuhkan untuk menyimpan simpul selama proses pencarian. |

Tujuan utama analisis efisiensi adalah mencari algoritma yang cepat, hemat memori, dan tetap menemukan solusi yang benar.

## 2. Konsep Kompleksitas dalam Pencarian AI

Kompleksitas Waktu (*Time Complexity*)

- Mengukur berapa banyak simpul (node) yang di-*generate* atau dievaluasi selama pencarian.
- Biasanya dinyatakan dalam notasi Big-O (O-notation) terhadap faktor b dan d, di mana:
  - b = *branching factor* = rata-rata jumlah cabang dari setiap node.
  - d = *depth* = kedalaman solusi (level di mana goal ditemukan).

Contoh: Jika b = 3 dan d = 4 → jumlah simpul maksimum = b^(d+1) − 1 = $3^5$ − 1 = 242 simpul.

Kompleksitas Ruang (*Space Complexity*)

- Mengukur jumlah memori yang dibutuhkan untuk menyimpan node (baik yang sedang dievaluasi maupun yang menunggu diproses).
- Umumnya proporsional dengan jumlah node aktif dalam struktur data (stack atau queue).

## 3. Analisis Kompleksitas Breadth-First Search (BFS)

Prinsip BFS

- Mengeksplorasi semua node pada level saat ini sebelum naik ke level berikutnya.
- Menggunakan struktur data queue (antrian) → semua node pada level disimpan di memori.

Kompleksitas Waktu BFS

- Dalam kasus terburuk, BFS perlu mengunjungi semua node sampai mencapai goal di kedalaman d.
  → Waktu ≈ O(b^(d+1))

Contoh:

Jika setiap node memiliki 3 anak (b=3) dan solusi ada di level ke-4 (d=4), jumlah node maksimum yang dievaluasi:

$1 + 3 + 3^2 + 3^3 + 3^4 = 121$ node

Kompleksitas Ruang BFS

- BFS harus menyimpan semua node di antrian sebelum naik ke level berikutnya
  → sangat boros memori.
  → Ruang ≈ O(b ^ (d+1))

Artinya: BFS cepat menemukan solusi terpendek, tetapi boros ruang.

Contoh Visual:

```
Level 0: 1 node
Level 1: b = 3
Level 2: b² = 9
Level 3: b³ = 27
Total node yang disimpan ≈ 1 + 3 + 9 + 27 = 40 node
```

Ringkasan BFS

| Aspek | Nilai |
|---|---|
| Waktu (Time) | O(b ^ (d+1)) |
| Ruang (Space) | O(b ^ (d+1)) |
| Lengkap? | Ya (selalu menemukan solusi) |
| Optimal? | Ya (menemukan solusi terpendek) |
| Efisiensi | Cepat tapi boros memori |

## 4. Analisis Kompleksitas Depth-First Search (DFS)

Prinsip DFS:

- Menelusuri jalur sedalam mungkin sebelum kembali ke atas.
- Menggunakan stack (tumpukan) → hanya menyimpan satu jalur aktif dan sebagian kecil node lainnya.



Kompleksitas Waktu DFS

- DFS dapat menjelajahi seluruh ruang hingga kedalaman maksimum m, walaupun goal ada di level d.

  → Waktu ≈ O(b ^ m)

m = kedalaman maksimum ruang pencarian (bisa jauh lebih besar dari d). Jadi DFS bisa sangat lambat jika solusi jauh di bawah atau tidak ada sama sekali.

Kompleksitas Ruang DFS

- DFS hanya menyimpan jalur aktif (dalam tumpukan) dan node turunan yang belum selesai.

  → Ruang ≈ O(b × m)

Artinya: DFS hemat memori tetapi berisiko menjelajah jalur yang tidak efisien.

Ringkasan DFS

| Aspek | Nilai |
|---|---|
| Waktu (Time) | O(b ^ m) |
| Ruang (Space) | O(b × m) |
| Lengkap? | Tidak (bisa terjebak loop) |
| Optimal? | Tidak selalu |
| Efisiensi | Hemat memori tapi bisa lama |

## 5. Perbandingan Kompleksitas BFS vs DFS

| Aspek | Breadth-First Search (BFS) | Depth-First Search (DFS) |
|---|---|---|
| Strategi Penelusuran | Level demi level | Jalur mendalam dahulu |
| Struktur Data | Queue (antrian) | Stack (tumpukan) |
| Waktu (Time Complexity) | O(b ^ (d+1)) | O(b ^ m) |
| Ruang (Space Complexity) | O(b ^ (d+1)) | O(b × m) |
| Lengkap (Complete) | Ya | Tidak selalu |
| Optimal (Shortest Path) | Ya (jika semua biaya sama) | Tidak |
| Kelebihan | Menemukan solusi terpendek | Hemat memori |
| Kelemahan | Sangat boros memori | Dapat menelusuri terlalu dalam |

## 6. Contoh Numerik

Misalkan:

- Branching factor (b) = 2
- Kedalaman solusi (d) = 5
- Kedalaman maksimum (m) = 10

| Algoritma | Kompleksitas Waktu | Kompleksitas Ruang |
|---|---|---|
| BFS | $O(2^6)$ = 64 node | $O(2^6)$ = 64 node |
| DFS | $O(2^{10})$ = 1024 node | $O(2 \times 10)$ = 20 node |

Kesimpulan:

- BFS lebih cepat menemukan solusi, tetapi membutuhkan 3× lebih banyak memori.
- DFS lebih ringan di memori, tapi berpotensi menjelajah node jauh lebih banyak.

## 7. Trade-Off antara Waktu dan Ruang

| Kondisi | Algoritma yang Lebih Cocok |
|---|---|
| Ruang keadaan kecil dan solusi dangkal | BFS |
| Ruang keadaan besar dan memori terbatas | DFS |
| Solusi tidak diketahui kedalamannya | DFS dengan batas (*Depth-Limited Search*) |
| Perlu solusi optimal | BFS |

AI modern sering menggabungkan keduanya → Iterative Deepening Search (IDS): kombinasi BFS (optimal) dan DFS (hemat ruang).

## 8. Faktor yang Mempengaruhi Efisiensi

1. Branching Factor (b) — Semakin besar nilai b, semakin eksplosif jumlah node.
2. Kedalaman Solusi (d) — Menentukan berapa banyak langkah yang diperlukan untuk mencapai goal.

3. Struktur Ruang Keadaan — Jika graf memiliki banyak jalur redundan, kompleksitas meningkat.
4. Strategi Pemilihan Node — Urutan eksplorasi mempengaruhi waktu dan ruang pencarian.

## Soal

### Pencarian dalam Ruang Keadaan (State Space Search)

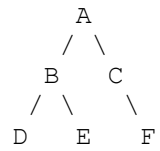
### A. Pemahaman Konsep Dasar

1. Jelaskan dengan kata-kata Anda sendiri apa yang dimaksud dengan ruang keadaan (*state space*) dalam kecerdasan buatan dan bagaimana konsep ini digunakan untuk memecahkan masalah.
2. Sebutkan dan jelaskan tiga elemen utama dalam model pencarian AI (state, operator, dan goal).
   Berikan satu contoh nyata untuk masing-masing elemen tersebut.
3. Dalam konteks permainan puzzle 8, bagaimana Anda merepresentasikan:
   - (a) state awal,
   - (b) operator, dan
   - (c) state tujuan (goal)?
4. Mengapa representasi masalah dalam bentuk pohon atau graf sangat penting dalam proses pencarian?
5. Apa perbedaan mendasar antara operator dan path (jalur) dalam ruang keadaan?

### B. Analisis Strategi Pencarian

6. Jelaskan langkah-langkah umum algoritma Breadth-First Search (BFS) dan sebutkan struktur data yang digunakan.
7. Jelaskan langkah-langkah umum algoritma Depth-First Search (DFS) dan perbedaan prinsip kerjanya dengan BFS.

8. Perhatikan struktur graf berikut:




Jika tujuan (goal) = F,

jelaskan urutan node yang dikunjungi menggunakan:

- a. BFS
- b. DFS

9. Dalam konteks ruang keadaan yang sangat besar, mengapa DFS lebih hemat memori dibandingkan BFS?

   Jelaskan dari sisi struktur data dan penyimpanan node.

10. Berikan contoh situasi nyata di mana BFS lebih tepat digunakan dibanding DFS, serta alasan pemilihannya.