INSTITUT TEKNOLOGI DEL

MATERI PRAKTIKUM

11S3242 - Machine Learning (+P)

Mata Kuliah	:	Kecerdasan Buatan (P)
Tanggal	:	11 September 2025
Semester	:	5
Topik	:	*Uninformed Search*
Minggu ke/Session ke	:	3/03
Aktivitas	:	Praktikum
Waktu pengerjaan	:	110 Menit
Setoran	:	Softcopy Laporan, Code, Video Presentasi (Upload YT)
Batas Penyerahan	:	Sabtu, 13 September 2025, Pukul 21.00 WIB
Tempat Penyerahan	:	Ecourse (Laporan dikumpulkan)
Tujuan Praktikum	:	1. Mahasiswa mampu menjelaskan konsep pemecahan masalah dengan pencarian. 2. Mahasiswa mampu menjelaskan contoh pemecahan masalah dengan pencarian di dunia nyata. 3. Mahasiswa mampu menerapkan metode Tree dan Graph untuk pemecahan masalah dengan pencarian.

Petunjuk Praktikum:

1. Mahasiswa sudah mempelajari materi *Dasar Uninformed Search*
2. Mahasiswa belajar secara mandiri dengan mencari sumber-sumber lain baik cetak,
   offline maupun *online*.
3. Mahasiswa harus mengikuti semua prosedur yang diberikan pada praktikum.
4. Apabila ada pertanyaan pada setiap langkah pengerjaan praktikum, silahkan berkonsultasi pada TA.

Praktikum 2: Pencarian Tanpa Informasi (Uninformed Search)

Selamat datang di praktikum Artificial Intelligence!

Kita akan mengimplementasikan dua algoritma paling fundamental dari kategori ini:

1. **Breadth-First Search (BFS)**: Mengekspansi node yang paling dangkal terlebih dahulu.
2. **Depth-First Search (DFS)**: Mengekspansi node yang paling dalam terlebih dahulu.

Tujuan Pembelajaran:

- Mampu merepresentasikan masalah graf dalam struktur data Python.
- Mampu mengimplementasikan algoritma BFS dari awal.
- Mampu mengimplementasikan algoritma DFS dari awal.
- Menganalisis dan membandingkan hasil dari kedua algoritma.

Bagian 1: Representasi Graf

Cara yang paling umum dan mudah dalam Python adalah menggunakan *dictionary* sebagai *adjacency list*.

- **Keys**: Setiap *key* dalam *dictionary* akan menjadi sebuah node (misal: 'S', 'A', 'B').
- **Values**: *Value* dari setiap *key* adalah sebuah *list* yang berisi tetangga-tetangganya.

Untuk saat ini, kita akan mengabaikan *cost* atau bobot dari setiap edge karena BFS dan DFS standar tidak menggunakannya.

Tugas 1: Buat Representasi Graf

Lengkapi kode di bawah ini untuk merepresentasikan graf yang ditunjukkan dalam slide.

Bagian 2: Implementasi Breadth-First Search (BFS)

BFS menjelajahi graf level demi level. Algoritma ini menggunakan struktur data **Queue** (Antrian) yang bersifat *First-In, First-Out* (FIFO) untuk mengelola node mana yang akan dikunjungi selanjutnya.

Sifat BFS:

- **Complete**: Ya, pasti menemukan solusi jika ada.
- **Optimal**: Ya, jika semua *cost* antar node sama (misal, bernilai 1). BFS akan menemukan jalur terpendek dalam hal jumlah langkah.
- **Kompleksitas Waktu & Ruang**: O(bd), di mana b adalah *branching factor* dan d adalah kedalaman solusi.

Tugas 2: Implementasi Fungsi BFS

Lengkapi fungsi `bfs` di bawah ini. Fungsi ini harus mengembalikan dua hal:

1. Urutan node yang dikunjungi (`visited_order`).
2. Jalur dari `start_node` ke `goal_node` (`path`).

Gunakan `collections.deque` untuk implementasi *queue* yang efisien.

Menjalankan dan Menganalisis BFS

Sekarang, mari kita jalankan fungsi BFS pada `toy_graph` dengan titik awal 'S' dan tujuan 'G'.

Analisis Hasil BFS:

Bandingkan output di atas dengan hasil yang ditunjukkan dalam slide `AI_W03S02-Uninformed_Search_Examples.pdf` (halaman 4-12).

- **Order of Visit (Slide):** S, A, B, C, D, E, G
- **Urutan Kunjungan (Kode Anda):** Seharusnya sangat mirip. Mungkin ada sedikit perbedaan dalam urutan A, B, C (menjadi B, A, C misalnya) tergantung pada bagaimana Anda mendefinisikan *list* tetangga di graf. Namun, properti *level-by-level* tetap terjaga: 'S' (level 0) dikunjungi, lalu semua tetangganya (level 1), dan seterusnya.
- **Path (Jalur):** Jalur yang ditemukan oleh kode kita adalah `S -> B -> G`. Ini adalah salah satu jalur terpendek dalam hal jumlah *edge* (2 langkah).

Bagian 3: Implementasi Depth-First Search (DFS)

DFS menjelajahi sejauh mungkin ke dalam satu cabang sebelum melakukan *backtracking*. Algoritma ini menggunakan struktur data **Stack** (Tumpukan) yang bersifat *Last-In, First-Out* (LIFO).

Sifat DFS:

- **Complete**: Tidak, bisa terjebak dalam *loop* atau cabang tak terbatas. Namun, menjadi *complete* jika grafnya terbatas dan kita melacak node yang sudah dikunjungi.
- **Optimal**: Tidak, seringkali tidak menemukan jalur terpendek.
- **Kompleksitas Waktu**: O(bm), di mana m adalah kedalaman maksimum graf.
- **Kompleksitas Ruang**: O(bcdotm), jauh lebih efisien daripada BFS dalam hal memori.

Tugas 3: Implementasi Fungsi DFS

Lengkapi fungsi `dfs` di bawah ini. Mirip dengan BFS, fungsi ini harus mengembalikan urutan kunjungan dan jalur yang ditemukan. Anda bisa menggunakan *list* Python sebagai *stack* (`append()` untuk *push* dan `pop()` untuk *pop*).

Menjalankan dan Menganalisis DFS

Mari kita jalankan fungsi DFS pada graf yang sama.

Analisis Hasil DFS:

Bandingkan output di atas dengan hasil yang ditunjukkan dalam slide `AI_W03S02-Uninformed_Search_Examples.pdf` (halaman 13-21).

- **Order of Visit (Slide):** S, A, D, F, G
- **Urutan Kunjungan (Kode Anda):** `S, A, D, F, G`. Urutan ini menunjukkan sifat "mendalam" dari DFS. Dari 'S', ia langsung menuju 'A', lalu ke 'D', lalu 'F', dan akhirnya menemukan 'G'. Ia tidak mengunjungi 'B' atau 'C' terlebih dahulu.
- **Path (Jalur):** Jalur yang ditemukan adalah `S -> A -> D -> F -> G`. Perhatikan bahwa jalur ini **bukan** yang terpendek dalam hal jumlah langkah (4 langkah vs 2 langkah pada BFS). Ini membuktikan bahwa DFS tidak optimal.

Bagian 4: Latihan - Studi Kasus Peta

Sekarang, gunakan fungsi BFS dan DFS yang telah Anda buat untuk menyelesaikan masalah pada "Map Example" (halaman 22, `AI_W03S02-Uninformed_Search_Examples.pdf`).

Tujuan: Temukan jalur dari **Las Vegas** ke **Calgary**.

1. Representasi Graf Peta

Graf untuk peta sudah disediakan di bawah ini.

2. Jalankan Pencarian

Gunakan sel di bawah ini untuk memanggil fungsi `bfs` dan `dfs` Anda pada `map_graph`.

3. Analisis dan Kesimpulan Latihan

Tuliskan pengamatan Anda di sini:

- **Jalur BFS**: Jalur yang ditemukan BFS adalah `Las Vegas -> Salt Lake City -> Helena -> Calgary`. Ini adalah jalur terpendek dalam hal jumlah kota yang dilewati (3 langkah).
- **Jalur DFS**: Jalur yang ditemukan DFS kemungkinan besar akan jauh lebih panjang, misalnya `Las Vegas -> Los Angeles -> San Francisco -> Portland -> Seattle -> Calgary` (5 langkah) atau rute lain yang lebih "dalam". Hasil persisnya bisa berbeda tergantung implementasi, tetapi hampir pasti bukan jalur terpendek.
- **Perbandingan**: Latihan ini secara nyata menunjukkan perbedaan utama antara BFS dan DFS. BFS sangat baik untuk menemukan jalur terpendek jika biaya setiap langkah sama.

Sementara itu, DFS, meskipun lebih hemat memori, tidak menjamin solusi optimal dan bisa mengambil rute yang sangat berbelit-belit.

Kesimpulan Praktikum

Dalam praktikum ini, kita telah berhasil:

1. Merepresentasikan sebuah graf menggunakan *dictionary* di Python.
2. Mengimplementasikan algoritma **Breadth-First Search (BFS)** yang menggunakan *queue* untuk menemukan jalur terpendek berdasarkan jumlah langkah.
3. Mengimplementasikan algoritma **Depth-First Search (DFS)** yang menggunakan *stack* untuk menjelajahi graf secara mendalam.
4. Mengamati secara langsung bahwa **BFS bersifat optimal** untuk masalah dengan biaya langkah seragam, sedangkan **DFS tidak**.

Tugas Praktikum

1. Silahkan buatkan contoh lain dari metode BFS dan DFS yang berbeda dari contoh praktikum. Untuk implementasi, terapkan metode BFS dan DFS pada tugas Graph map yang sudah dikumpulkan.
2. Buatkan video untuk keseluruhan video.