Praktikum 1 & Praktikum 2

Nama : Ammar Ghozy Tanumijaya

NRP : 5025231203

Praktikum 1

Soal

Solusi untuk Knight's Tour Problem menggunakan algoritma Warnsdorff

Solusi ini melibatkan aturan heuristik untuk menentukan langkah selanjutnya bagi sang kuda (knight) dalam catur sehingga memiliki kemungkinan terkecil untuk terjebak atau mencapai langkah buntu.

Berikut langkah-langkah implementasi algoritma Warnsdorff untuk Knight's Tour Problem:

1. Tentukan langkah awal, yaitu kotak awal kuda pada papan catur.
2. Hitung semua langkah yang mungkin dari posisi saat ini, yaitu sel-sel yang dapat dikunjungi kuda berdasarkan pola gerakannya.
3. Pilih langkah dengan jumlah kemungkinan langkah selanjutnya paling sedikit.
4. Pindahkan kuda ke posisi tersebut.
5. Perbarui daftar langkah yang mungkin, dengan menghapus sel yang telah dikunjungi.
6. Ulangi langkah 2-5 hingga seluruh papan catur (8x8 kotak) telah dikunjungi.

Jika algoritma berhasil, rute perjalanan akan ditampilkan dalam format matriks atau grafik. Jika tidak, solusi tidak ditemukan.

Metode Warnsdorff membantu mengarahkan kuda ke sekitar papan catur dengan efisien, meminimalkan peluang terjebak, dan meningkatkan kesempatan menemukan solusi untuk Knight's Tour Problem. Algoritma ini didasarkan pada ide bahwa memilih langkah yang mendekati batas catur dapat membantu mengurangi kemungkinan langkah buntu di kemudian hari.

A. Knight's Tour Closed

Program ini menyelesaikan Closed Knight's Tour, yaitu tur kuda pada papan 8×8 yang:

- Mengunjungi semua 64 kotak tepat satu kali
- Langkah terakhir harus dapat kembali ke langkah pertama (closed)

Program menggunakan *heuristik Warnsdorff* untuk memilih langkah dengan kemungkinan jalan buntu paling rendah.

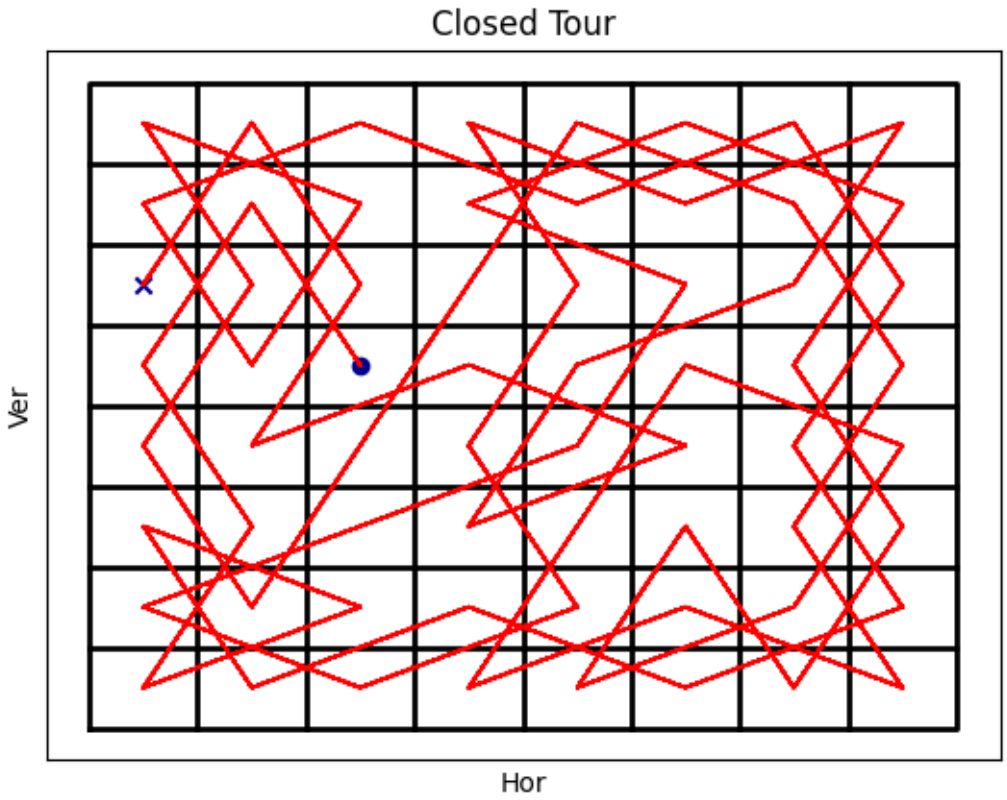
Berikut adalah implementasi Kodenya.

Cara Kerja

1. **Inisialisasi Papan Catur:**
    - Papan catur diwakili sebagai matriks 8x8.
    - Nilai-1 menandakan kotak belum dikunjungi.
2. **Validasi Langkah:**
    - Fungsi *is_valid_move* mengecek apakah langkah kuda berada dalam batas papan dan kotak belum dikunjungi.
3. **Pemilihan Langkah Berdasarkan Aksesibilitas:**
    - Fungsi *get_valid_moves* menghitung jumlah langkah valid dari setiap posisi calon langkah dan memilih langkah dengan aksesibilitas minimum (metode Warnsdorff).
4. **Algoritma Utama:**
    - Fungsi *knight_tour* menggunakan rekursi untuk mencari jalur kuda, dimulai dari posisi awal pengguna.
    - Jika tur selesai (64 langkah), algoritma berhenti.
    - Jika tur bersifat "open" (tidak ada jalur balik ke posisi awal), papan hasil dicetak.
5. **Visualisasi:**
    - Langkah-langkah kuda disusun menjadi daftar koordinat untuk divisualisasikan menggunakan *matplotlib*.
    - Papan catur digambar dengan batas, grid, dan jalur perjalanan kuda.

Hasil

Program menghasilkan rute perjalanan kuda yang memenuhi syarat Closed Tour di papan catur dan memvisualisasikan rute tersebut dengan diagram. Berikut adalah hasil ketika program dijalankan:



B. Knight's Tour Open

Open Knight's Tour adalah tur kuda pada papan 8×8 yang memenuhi syarat:

- Setiap kotak dikunjungi tepat satu kali
- Tidak wajib kembali ke posisi awal (beda dengan Closed Tour)

Jadi selama 64 kotak terisi dan langkah terakhir tidak dapat kembali ke langkah pertama menggunakan langkah kuda → itu disebut open tour.

Berikut adalah implementasi Kodenya.

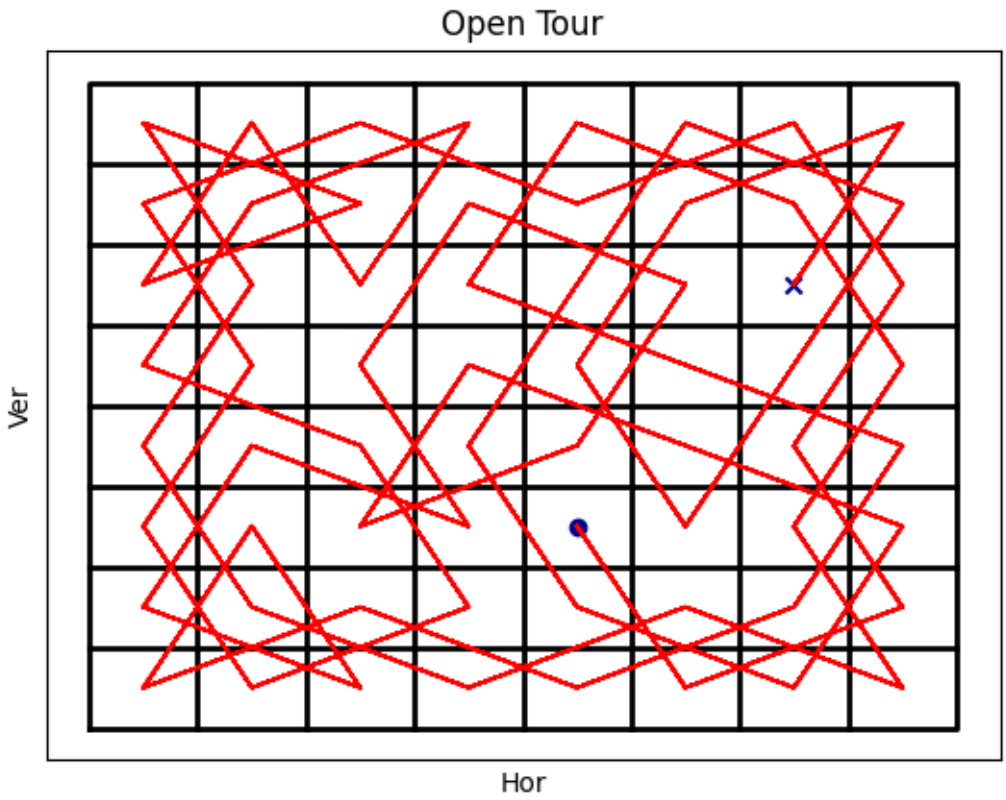
Cara Kerja

1. **Inisialisasi Papan Catur:**
    - Sama seperti Program 1, menggunakan array linear ukuran 64 untuk merepresentasikan papan.
    - Nilai -1 menandakan kotak belum dikunjungi.

2. **Validasi Langkah:**
    - Fungsi *limits* mengecek batas papan.
    - Fungsi *isempty* mengecek apakah kotak belum dikunjungi.
3. **Pemilihan Langkah Berdasarkan Warnsdorff:**
    - Fungsi *nextMove* menentukan langkah berikutnya berdasarkan aksesibilitas minimum dari posisi calon langkah.
4. **Cek Kondisi Closed Tour:**
    - Setelah semua langkah selesai, fungsi *neighbour* mengecek apakah langkah terakhir dapat kembali ke posisi awal.
    - Jika iya, tur selesai dan matriks langkah dicetak.
5. **Visualisasi:**
    - Sama seperti Program 1, langkah-langkah kuda divisualisasikan menggunakan *matplotlib*.

Hasil

Program menghasilkan rute perjalanan kuda yang memenuhi syarat Open Tour di papan catur dan memvisualisasikan rute tersebut dengan diagram. Berikut adalah hasil ketika program dijalankan:



C. Perbedaan Kedua Algoritma

Berikut adalah perbedaan utama antara kedua algoritma yang digunakan tersebut:

1. **Jenis Tur**
    - Program 1: Mencari Open Knight's Tour (tur kuda terbuka), di mana kuda tidak perlu kembali ke posisi awal setelah langkah terakhir.
    - Program 2: Mencari Closed Knight's Tour (tur kuda tertutup), di mana langkah terakhir kuda harus dapat kembali ke posisi awal.
2. **Representasi Papan**
    - Program 1:
        - Menggunakan matriks 2D ukuran 8x8 *(board[x][y])* untuk mewakili papan catur.
        - Nilai -1 menunjukkan kotak belum dikunjungi.
    - Program 2:
        - Menggunakan array 1D ukuran 64 *(a[y * N + x])* untuk merepresentasikan papan catur.
        - Nilai -1 juga digunakan untuk kotak yang belum dikunjungi.
3. **Algoritma Pemilihan Langkah**
    - Program 1:
        - Menggunakan metode berbasis aksesibilitas minimum.
        - Fungsi *get_valid_moves* menghitung jumlah langkah valid dari setiap calon langkah dan memilih langkah dengan jumlah langkah valid paling sedikit (*Warnsdorff Heuristic*).
        - Ada elemen *randomization* untuk mengacak pemilihan langkah dengan aksesibilitas sama.
    - Program 2:
        - Langsung mengimplementasikan algoritma Warnsdorff dengan memulai dari langkah acak.
        - Tidak menghitung ulang langkah valid setelah langkah dipilih.
4. **Pengecekan Keberhasilan**
    - Program 1:
        - Menggunakan fungsi *is_closed_tour* untuk memastikan tur bersifat open.
        - Memeriksa apakah langkah terakhir tidak memiliki jalur balik ke posisi awal.
    - Program 2:
        - Menggunakan fungsi *neighbour* untuk memastikan tur bersifat closed.
        - Memeriksa apakah langkah terakhir dapat langsung kembali ke posisi awal.
5. **Pencarian Solusi**
    - Program 1:
        - Mencari solusi satu kali untuk menghasilkan tur open.
        - Jika tur yang ditemukan bersifat tertutup, program akan mencoba lagi.
    - Program 2:

        - Terus mencari solusi hingga menemukan closed tour.
        - Akan mengulang algoritma dari awal jika tidak ada solusi yang ditemukan.
6. **Visualisasi Hasil**
    - Program 1 dan Program 2:
        - Sama-sama menggunakan *matplotlib* untuk menggambar papan catur dan jalur perjalanan kuda.
        - Bedanya, Program 1 menandai jalur untuk open tour, sedangkan Program 2 menandai jalur untuk closed tour.
7. **Fokus pada Kondisi Akhir**
    - Program 1: Tidak memerlukan syarat tambahan di langkah terakhir selain menyelesaikan 64 langkah.
    - Program 2: Langkah terakhir harus memastikan jalur balik ke posisi awal, sehingga memenuhi syarat closed tour.

Ringkasan Perbedaan Utama

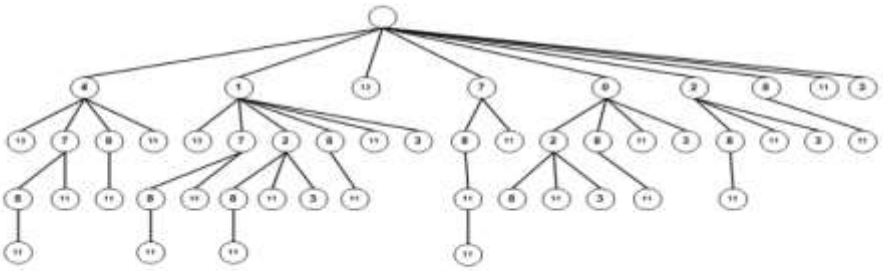
- **Tujuan Akhir**: *Open Tour* vs *Closed Tour*.
- **Representasi Data:** *Matriks 2D* vs *Array 1D*.
- **Pemilihan Langkah:** Aksesibilitas minimum dengan acak vs Algoritma Warnsdorff langsung.
- **Pengecekan Akhir**: Tidak ada jalur balik vs Harus ada jalur balik ke awal.

Praktikum 2

Soal

TUGAS PRAKTIKUM - 2

Implementasikan sebuah program untuk menyelesaikan permasalahan "Largest Monotonically Increasing Subsequence"



Aplikasi tree untuk mencari largest monotonically increasing subsequence dari urutan bilangan : 4, 1, 13, 7, 0, 2, 8, 11, 3

Penyelesaian Soal Largest Monotonically Increasing Subsequence (LMIS)

Longest Monotonically Increasing Subsequence (LMIS) adalah subsequence dari sebuah array yang menunjukkan elemen-elemen berurutan secara monoton meningkat.

Elemen-elemen pada subsequence ini tidak harus bersebelahan dalam array asal, namun tetap harus mengikuti urutan seperti pada array asal. Tujuan dari masalah ini adalah menemukan subsequence terpanjang yang memenuhi kriteria tersebut.

Salah satu cara untuk menyelesaikan masalah LMIS adalah dengan menggunakan pendekatan tree. Ide dasar dari metode ini adalah:

- Menjadikan elemen-elemen array sebagai node dalam sebuah pohon.
- Elemen yang berada di indeks kanan dan memiliki nilai lebih besar dibanding elemen saat ini akan menjadi anak dari elemen tersebut.
- Penyusunan tree dilakukan secara rekursif, sehingga terbentuk leaf di ujung pohon ketika tidak ada elemen di indeks kanan yang lebih besar dari elemen tersebut.
- Jalur terpanjang dari root hingga leaf mewakili subsequence terpanjang yang monoton meningkat.

Langkah Penyelesaian

1. **Pembuatan Tree:**

   Pohon dibuat dengan root bernilai -∞ (atau None dalam implementasi program) untuk memastikan semua elemen pada array bisa menjadi anaknya. Setiap elemen hanya akan memiliki anak berupa elemen-elemen yang lebih besar dan berada di indeks kanan.

2. **Evaluasi Path Terpanjang:**

   Jalur dari root ke leaf dihitung. Jalur dengan panjang maksimum dipilih sebagai LMIS. Ini dilakukan dengan traversal pohon untuk mengevaluasi semua jalur yang mungkin.

3. **Output LMIS:**

   Dari pohon yang dibuat, semua subsequence terpanjang yang mungkin akan dicetak. Jika terdapat beberapa subsequence dengan panjang sama, semuanya akan ditampilkan.

Contoh Kasus

Misalkan array original adalah [4, 1, 13, 7, 0, 2, 8, 11, 3]. Dengan metode ini, kita dapat menemukan semua subsequence terpanjang seperti berikut:

- [4, 7, 8, 11]
- [1, 7, 8, 11]
- [1, 2, 8, 11]
- [0, 2, 8, 11]

Semua subsequence di atas memiliki panjang 4, sehingga panjang LMIS dari array tersebut adalah 4.

Penjelasan Implementasi Code

Program berikut menyelesaikan masalah ini menggunakan Python:

1. **TreeNode Class**

   Merepresentasikan setiap node dalam tree. Setiap node memiliki nilai dan daftar anak (children).

2. **Fungsi untuk Membangun Tree**

   Fungsi *construct_tree* dan *construct_tree_helper* digunakan untuk membangun tree. Tree dibuat secara rekursif dengan menambahkan elemen yang lebih besar sebagai anak dari node saat ini.

3. **Fungsi untuk Mencari Jalur Terpanjang**

   Fungsi *find_all_longest_paths* mencari semua jalur terpanjang dari root ke leaf dalam tree. Jalur dengan panjang maksimum dipilih sebagai LMIS.

4. **Visualisasi Tree**

   Fungsi *plot_tree* memvisualisasikan tree menggunakan pustaka matplotlib untuk membantu memahami struktur pohon yang dibuat.

5. **Fungsi Utama**

   Fungsi *main* menginisialisasi array input, membangun tree, mencari subsequence terpanjang, dan mencetak hasilnya.

Kesimpulan

Pendekatan ini memungkinkan visualisasi lengkap dari semua subsequence yang mungkin. Dengan input array **[4, 1, 13, 7, 0, 2, 8, 11, 3]**, program menemukan LMIS dengan panjang 4, yaitu [**4, 7, 8, 11**], **[1, 7, 8, 11**], **[1, 2, 8, 11**], dan **[0, 2, 8, 11].**

Output

```
Input Array: [4, 1, 13, 7, 0, 2, 8, 11, 3]
All Longest Increasing Subsequences:
[4, 7, 8, 11]
[1, 7, 8, 11]
[1, 2, 8, 11]
[0, 2, 8, 11]
```