Penerapan Algoritma *Depth-first* dalam Strategi Permainan Shogi

Irgiansyah Mondo - 13521167
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : 13521167@std.stei.ac.id

Abstract*— Algoritma pencarian *depth-first* dapat diimplementasikan dalam menaksir langkah-langkah praktis yang sering digunakan dan dapat menunjukan langkah terbaik dari beberapa langkah yang sudah dipangkas menggunakan metode iterative dari pendekatan *depth-first.

***Keywords—depth-first;* shogi;DFS**

I. INTRODUCTION

Dengan kemajuan teknologi yang ada, ada begitu banyak *tools-tools* yang dapat kita gunakan untuk meningkatkan pengetahuan kita terhadap gerakan praktis dan terbaik yang kebanyakan di terapkan pada *board game* seperti catur dan shogi. Gerakan terbaik tersebut didapatkan dari aturan yang ada di game ataupun dari pengalaman pemain-pemain yang berhasil di lestarikan atau diabadikan dengan dimasukan dalam *database* sehingga dengan ini gerakan tersebut dapat diabadikan dan dipelajari oleh pemain di generasi selanjutnya. Namun dengan begitu banyaknya permainan yang berulang-ulang dan banyaknya game yang dimainkan tentunya kita tidak dapat menentukan gerakan dari pemain mana yang terbaik dan gerekan mana yang paling efisien, Maka dari itu dengan dibuatnya tools-tools yang dapat menganalisis perpindahan dan keuntungan posisi dari permainan dapat memberikan evaluasi yang sangat bermanfaat bagi pemain yang mau meningkatkan kemampuannya dengan mempelajari perpindahan dan posisi yang direkomendasikan dan disimpulkan oleh computer.

Aturan yang terdapat pada *board game* pun, pada umumnya terstruktur dan terdefinisi dengan jelas, setiap langkah diatur dengan ketat baik itu perpindahan bidak, batasan, dan kondisi menang atau kalah. Pada permainan ini juga terdapat keterbatasan aksi jumlah langkah yang dapat diambil oleh pemain yang biasanya bergantian setiap satu perpindahan. Sementara itu, dalam shogi, pemain dapat menggerakan lebih dari satu bidak dalam satu giliran, tetapi masih ada Batasan jumlah bidak yang dapat digerakan. Keterbatasan ini memungkinkan sistem computer untuk melakukan evaluasi langkah secara efisien.

Permainan shogi memiliki banyak strategi dan taktik yang rumit. Pemilihan langkah yang tepat, perencanaan jangka Panjang, dan pengambilan keputusan yang baik merupakan kunci keberhasilan dalam permainan ini. Sistem computer dapat diberikan pengetahuan strategi dan taktik ini dalam bentuk aturan dan heuristic untuk membantu dalam pengambilan keputusan.

Keputusan-keputusan yang di implementasi dengan algoritma depth-first. Algoritma ini dapat berguna pada pendekatan pencarian untuk mengeksplorasi ruang pencarian dan mencari solusi yang merujuk pada keaadaan atau simpul dalam grafik atau struktur data yang digunakan untuk merepresentasikan ruang pencarian yang direalisasikan pada keadaan atau konfigurasi papan dalam permainan shogi.

Pada pengambilan keputusan tentang langkah-langkah selanjutnya yang akan diambil pada pencarian dan pemilihan simpul berupa langkah skakmat dan solusi untuk bertahan berikutnya untuk dieksplorasi akan di implemetasikan dengan algorita depth-first menggunakan bahasa pemrograman python.

II. TEORI DASAR

A. Shogi

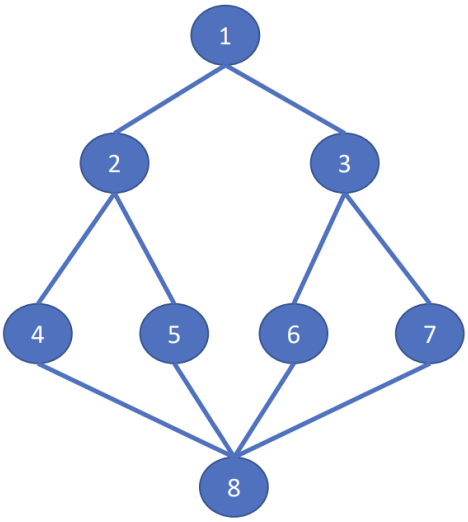
Shogi adalah permainan papan strategi yang berasal dari Jepang. Juga dikenal sebagai "catur Jepang", Shogi memiliki banyak persamaan dengan permainan catur internasional, tetapi dengan aturan dan komponen permainan yang sedikit berbeda. Shogi dimainkan oleh dua pemain yang berusaha untuk menyerang dan merebut kepingan lawan serta mencapai tujuan tertentu, yaitu memerintahkan Raja lawan. Papan Shogi memiliki ukuran 9x9 dengan kepingan yang berbeda-beda, termasuk Raja, Benteng, Meriam, dan prajurit lainnya. Dalam Permainan ini pihak yang menyerang adalah pihak hitam dan pihak yang bertahan adalah pihak putih. Ada beberapa aturan yang harus terpenuhi agar terjadinya algoritma menyerang dan bertahan, yaitu :

- Pemain hitam mengambil langkah untuk menyerang dengan inisiasi skakmat.
- Pemain hitam menyerang dengan urutan langkah seminimal mungkin.
- Pemain putih mengambil langkah semaksimal mungkin dan menghindari langkah blunder.
- Putih dapat menjatuhkan bidak hitam, kecuali raja hitam berada di tangan putih.

- Panjang urutan solusi tidak diberikan sebelumnya.
- Raja tidak boleh di ambil, harus dalam kondisi skakmat.
- Hanya satu solusi untuk menjebak raja, jika salah dalam mengambil langkah, maka dia gagal dalam menjebak raja. Jika putih salah mengambil langkah maka hitam masih memiliki kesempatan untuk menjebak raja.
- Di posisi akhir, hitam sedang tidak memiliki bidak (posisi skakmat) atau permainan telah selesai.

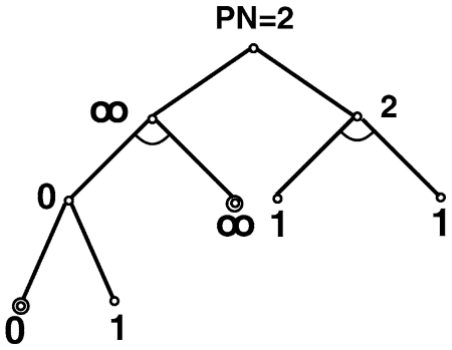
B. Depth-First Search

DFS merupakan algoritma pencarian yang digunakan dalam sistematika pohon graf. Pencarian ini diawali dengan simpul v lalu diteruskan dengan mengunjungi simpul w yang bertetangga dengan simpul v lalu diulangi DFS mulai dari simpul w. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.



Gambar 2.1 Pohon DFS (Source : Website Pak Rinaldi Munir)

Dalam kasus permainan shogi, rancangan pohon graf menggunakan simpul OR yang berarti posisi pemain hitam yang dengan aturan jika pemain hitam tidak memiliki langkah lagi untuk menyerang maka masalah tidak dapat diselesaikan dan diatur menjadi tak hingga. Simpul AND berarti posisi pemain putih yang harus menyelesaikan solusi untuk bertahan. Akar di sebut pemain pertama yaitu pemain hitam lalu seiring dengan bergantian lapis akan bergiliran setiap satu kali lapis.



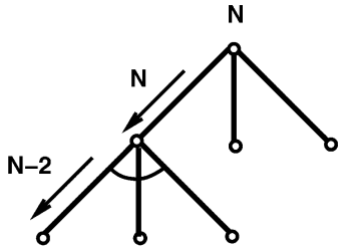
Gambar 2.2 Ilustrasi Graf

Langkah Penjabarang algoritma adalah sebagai berikut :

- n adalah simpul daun maka (1). Jika n adalah simpul AND terminal, N diselesaikan dan PN(n) = 0. (2) Jika n adalah simpul OR terminal makan n tidak dapat diselesaikan dan n = tak hingga.
- PN(n) = 1 jika simpul daun belum terselesaikan.
- Jika simpul OR memiliki anak di bawahnya maka PN(n) = min(PN(ni)) untuk $1 \leq i \leq k$.
- jika simpul AND memiliki anak dibawahnya maka PN(n) = $\sum$ PN(ni) untuk $1 \leq i \leq k$.

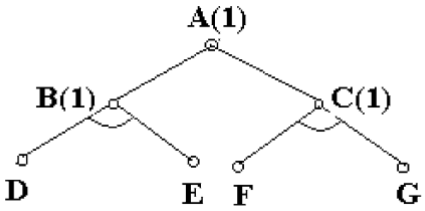
C. Depth-first Iterative Depeening

FS Algoritma ini akan memakan waktu yang sangat banyak kecuali jika anak dari pohon graf yang memiliki lapisan yang sedikit, sehingga algoritma ini masi memiliki banyak kekurangan dalam hal kepastian waktu yang dipakai untuk mencari solusi yang optimal. Iterative-deepening search (IDS) adalah metode pencarian yang tidak menderita kelemahan pencarian berdasarkan kedalaman (depth-first search) . IDS melakukan pencarian berdasarkan kedalaman hingga kedalaman 1 pada iterasi pertama, dan jika solusi tidak ditemukan pada iterasi pertama, maka pencarian dilanjutkan hingga kedalaman 2 pada iterasi berikutnya, dan seterusnya. Semua node yang dihasilkan pada iterasi sebelumnya dibuang. Pada gambar 2.3, N1 = 3. Kemudian, dari terkiri menjadi N1. Jika N1 lebih besar dari N, maka ekspansi dikiri dibatalkan, dan proses pencarian kembali ke node OR induk, dan melakukan ekspansi berikutnya yang berakar pada node OR tersebut. Namun, jika N1 tidak melebihi ambang N, maka terkiri di-expand dengan memberikanN - N1 + 1 kepada node OR anak terkiri. Sehingga untuk menyelesaikan node AND dalam paling banyak N ekspansi, sebuah node anak harus diselesaikan dalam paling banyak N dikurangi jumlah saudara kandung, karena setiap saudara kandung akan membutuhkan setidaknya 1 ekspansi.



Gambar 2.3 Nth iterasi

Dalam Gambar 2.4 Angka yang terletak di dalam tanda kurung pada setiap node menunjukkan batasan yang diberikan kepada node tersebut. Pada iterasi pertama , akar A menerima Batasan 1. A di-expand, dan node B dan C dihasilkan. Pencarian dilanjutkan pada node B (dari kiri) dengan ambang 1, dan dengan meng-expand B, node D dan E dihasilkan. Hal ini melebihi batasan yang diberikan sebesar 1. batasan dari node B menjadi 2. Selanjutnya, pencarian dilanjutkan pada node C dengan batasan 1, dan menghasilkan node F dan G dengan meng-expand C, yang juga melebihi batasan 1. node C dan A menjadi 2.



Gambar 2.4 Skenario Batasan dalam kurung

III. IMPLEMENTASI ALGORITMA

Berikut adalah Implementasi dari algoritma depth-first :

```python
def ITERATE(root):
    PN = {root: 1}

    while True:
        result = EXPAND(root, PN[root])

        if PN[root] == 0:
            return "proven"
        if PN[root] == float('inf'):
            return "disproven"
```

kode tersebut sebagai inisialisasi awal root sebagai node akar. Berikut dilengkapi dengan inisialisasi fungsi expand dan generate untuk memberikan data suksesor lalu disimpan di fungsi puintable.

```python
def EXPAND(n, PN):
    c = FINDTABLE(n)

    if c == 0 or c > PN[n]:
        PN[n] = c
        return

    if GENERATE(n) == 0:
        if n is an AND node:
            PN[n] = 0
        if n is an OR node:
            PN[n] = float('inf')

        PUTINTABLE(n)
        return

    PUTINTABLE(n)

    if n is an OR node:
        for ni in successors(n):
            EXPAND(ni, PN[n])
            if PN[ni] == 0:
                break

        PN[n] = min(PN[ni] for ni in successors(n))
        PUTINTABLE(n)
        return

    if n is an AND node:
        for ni in successors(n):
            PN[ni] = FINDTABLE(ni)
```

Dengan adanya findtable kita dapat memeriksa node n sudah terpecahkan atau proof number-nya melebihi threshold yang diberikan. Jika ya, maka nilai proof number dari node n diperbarui dan fungsi dihentikan. Dan juga emeriksa apakah node n memiliki successor nodes. Jika tidak, maka node tersebut dianggap sebagai leaf node. Jika node n adalah AND node, maka proof number-nya diatur menjadi 0. Jika node n adalah OR node, maka proof number-nya diatur menjadi tak terhingga (float('inf')). Selanjutnya, informasi node n disimpan dalam transposition table menggunakan fungsi PUTINTABLE(n). Juga kelanjutan dari kode tersebut adalah sebagai berikut :

```
if n is an AND node:
    for ni in successors(n):
        PN[ni] = FINDTABLE(ni)

    if sum(PN[ni] for ni in successors(n)) > PN[n]:
        PN[n] = max(PN[ni] for ni in successors(n))
        PUTINTABLE(n)
        return

    for ni in successors(n):
        PN[ni] = 1

        while True:
            EXPAND(ni, PN[ni])

            if PN[ni] == 0:
                break

            if PN[ni] > PN[n]:
                PN[n] = PN[ni]
                PUTINTABLE(n)
                return

    PN[n] = 0
    PUTINTABLE(n)
    return
```

IV. KESIMPULAN DAN SARAN

Algoritma depth-first yang di implemetasikan cenderung memakan banyak waktu dan tidak optimal dalam kehidupan nyata yang membutuhkan anilisis lebih dari puluhan kemungkinan bahkan ratusan kemungkinan yang terjadi. kemungkin yang dilakukan pada awal game yang tentunya memiliki berbagai ribuan pola dikarenakan masih banyaknya petak dan bidak yang belum memasukan pertengahan permainan yang harusnya dapat dianalisis seluruh perilaku bidak pada awal permainan tersebut. Masih terdapat kekurangan dan *bug* pada kode tersebut sehingga belum bisa melakukan tes kasus dan makalah ini hanya berisi metode algoritma sebagai bahan referensi di masa depan dan akan dikembangkan lebih lanjut untuk dapat menjadikan langkah awal untuk menemukan algortima yang lebih optimal dan efisien.

UCAPAN TERIMA KASIH

Penulis Mengucapkan puji dan syukur atas berkahnya bisa menyelasaikan makalah ini dan penulis juga mengucapkan kepada para dosen pengampu mata kuliah Strategi Algoritma yang telah membersamai dalam menuntut ilmu.

REFERENCES

[1] Munir, Rinaldi. 2021. Depth First Search https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf . Diakses pada 22 Mei 2023

[2] Ilda, Hiroyuki 2001. Computer Shogi https://www.sciencedirect.com/science/article/pii/S0004370201001576?via%3Dihub. Diakses pada 22 Mei 2023

[3] L.V. Allis, Searching for solutions in games and artificial intelligence, Ph.D. Thesis, University of Limburg, Maastricht, The Netherlands, 1994.

[4] L.V. Allis, M. van der Meulen, H.J. van den Herik, Proof-number search, Artificial Intelligence 66 (1) (1994) 91–124.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

Irgiansyah Mondo 13521167