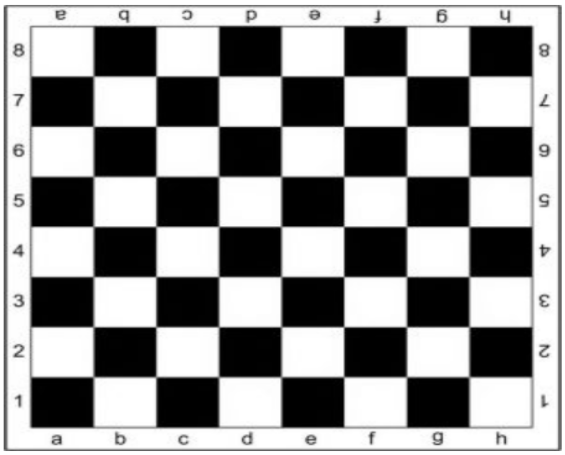
Minimisasi Langkah Bidak Kuda untuk Mencapai Suatu Petak pada Permainan Catur dengan Algoritma *Breadth First Search*

Karel Renaldi - 13519180
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519180@std.stei.itb.ac.id

***Abstract*—*Knight* / kuda merupakan salah satu bidak pada permainan catur dimana bidak kuda ini memiliki langkah yang unik yaitu bergerak secara huruf L untuk menuju suatu petak tertentu atau koordinat petak tertentu. Dari keunikan langkah bidak kuda ini pada permainan catur maka timbulah suatu *puzzle* dimana *puzzle* ini bertujuan untuk mencari langkah minimum suatu kuda untuk bergerak untuk menuju suatu titik pada petak catur. *puzzle* ini juga berguna sebagai strategi khusus pada permainan catur. Pada makalah ini akan dibahas solusi untuk *puzzle* tersebut yaitu mencari nilai minimum langkah suatu kuda untuk menuju suatu koordinat pada petak catur. Algoritma yang akan digunakan pada permasalahan ini adalah algoritma BFS(*Breadth First Search*).**

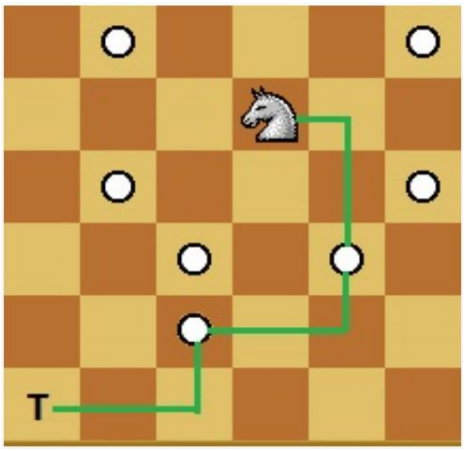
***Keywords*— *Knight*, Kuda, BFS, *Puzzle*, *Breadth First Search*.**

I. PENDAHULUAN



Gambar 1. Papan Catur
(sumber: https://gurupenjaskes.com/olahraga-catur-sejarah-aturan-dan-cara-bermainnya)

Catur merupakan sebuah permainan yang dilakukan oleh 2 orang dan juga merupakan sebuah cabang olahraga. Catur sendiri tidak memiliki sejarah pasti kapan ditemukannya namun dari beberapa sumber terpercaya yang tertulis dalam buku History of Chess yang ditulis pada tahun 1913 catur berasal dari negara india dan mulai dimainkan pada abad ke 6. Catur merupakan olahraga yang umumnya dilakukan oleh 2 orang dimana standar khusus dari papan catur yaitu berukuran 8 x 8 petak. Pada permainan catur juga terdapat 6 buah jenis bidak yaitu raja(*king)*, ratu(*queen*), benteng(*rook*), gajah(*bishop*) dan pawn(*bishop*). Selain untuk olahraga / permainan antara dua orang catur juga bisa dijadikan sebagai sebuah *puzzle*. Sampai saat ini sangat banyak *puzzle* yang dapat dibuat dari permainan catur ini salah satunya adalah *N Queen Problem*, *Knight Tour Problem*, dll. Pada makalah ini penulis akan membahas sebuah *puzzle* permainan catur yaitu *puzzle* mencari langkah minimum suatu bidak kuda untuk mencapai suatu koordinat pada petak catur. Ilustrasi dari *puzzle* tersebut ditunjukan pada gambar dibawah.



Gambar 2. Ilustrasi minimisasi langkah bidak kuda
(sumber: https://www.geeksforgeeks.org/minimum-steps-reach-target-knight-set-1/)

Puzzle yang akan dibahas pada makalah ini tidak terbatas pada papan catur 8 x 8 namun, permainan puzzle ini digeneralisasi sehinnga puzzle ini bisa dicari solusinya sampai N x N. Secara garis besar permasalahan dari puzzle ini bisa dicari solusinya dengan membentuk sebuah pohon pencarian / pohon ruang status menggunakan algoritma pencarian secara BFS (*Breadth First Search*).

II. DASAR TEORI

A. *Graf*

Secara definisi Graf G merupakan pasangan himpunan (V, E) dan di notasikan sebagai berikut : G = (V, E). V dalam himpunan G merupakan himpunan tidak kosong dari simpul – simpul (vertices) dinotasikan sebagai berikut : V = {v1, v2, ..., vn}. E dalam himpunan G merupakan himpunan sisi (edges) yang menghubungkan sepasang atau 2 buah simpul (vertices) dinotasikan sebagai berikut : E = {e1, e2, ..., en}.

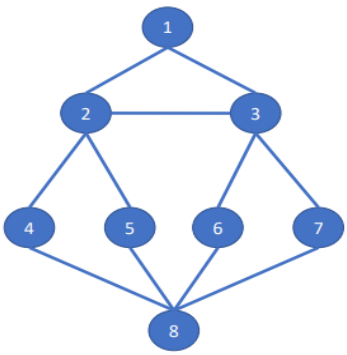
B. *Breadth First Search dan Depth First Search*

Pada teori graf terdapat banyak metode traversal graf yaitu metode traversal secara *breadth first search* (BFS) atau secara singkatnya bisa dibilang pencarian secara melebar, *depth first search* (DFS) atau secara singkatnya bisa dibilang pencarian secara mendalam.

1. BFS

   BFS / *Breadth First Search* merupakan sebuah algoritma pencarian secara melebar yang diterapkan pada sebuah graf baik graf berarah maupun graf tidak berarah. Secara umum algoritma BFS ini melakukan pencarian yang mengunjungi setiap simpul / *vertices* pada sebuah graf secara preorder yaitu mengunjungi suatu simpul kemudian mengunjungi semua simpul yang bertetangga dengan simpul awal yang telah dikunjungi dilakukan secara terus menerus sampai ditemukan sebuah simpul / *vertices* solusi atau semua simpul sudah dikunjungi. Secara matematis algoritma dari pencarian BFS ini dapat ditulis sebagai berikut:

   1) Memilih simpul awal / simpul *source* lalu masukkan kedalam sebuah *queue*
   2) Lakukan *dequeue* pada sebuah *queue* lalu bangkitkan semua tetangga dari hasil *dequeue*
   3) Investigasi hasil yang dibangkitkan pada langkah 2 apakah simpul / *vertices* merupakan simpul tujuan atau bukan lalu investigasi juga apakah simpul tersebut sudah dikunjungi atau belum
   4) Jika simpul tersebut bukan merupakan sebuah solusi maka cek apakah simpul tersebut sudah dikunjungi atau belum jika belum maka masukkan simpul tersebut kedalam queue, jika sudah maka lakukan pengecekan ke simpul yang lain. Jika pengecekan semua *vertices* sudah dilakukan dan belum ditemukan simpul solusi maka ulangi langkah 2 jika sudah maka pencarian berhenti.



Gambar 3. Ilustrasi BFS
(sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik)

Untuk penjelasan lebih lengkapnya kita bisa lihat pada gambar 3. Misalkan kita ingin melakukan traversal secara BFS pada gambar 3 maka pada langkah yang dituliskan diatas kita bisa membuat solusinya seperti dibawah ini :

1) Inisiasi dengan *vertices* = 1 dan *queue* = {1}
2) Iterasi 1 dengan *vertices* = 1 dan *queue* = {2, 3}
3) Iterasi 2 dengan *vertices* = 2 dan *queue* = {3, 4, 5}
4) Iterasi 3 dengan *vertices* = 3 dan *queue* = {4, 5, 6, 7}
5) Iterasi 4 dengan *vertices* = 4 dan *queue* = {5, 6, 7, 8}
6) Iterasi 5 dengan *vertices* = 5 dan *queue* = {6, 7, 8}
7) Iterasi 6 dengan *vertices* = 6 dan *queue* = {7, 8}
8) Iterasi 7 dengan *vertices* = 7 dan *queue* = {8}
9) Iterasi 8 dengan *vertices* = 8 dan *queue* = {}

Jadi solusi pencarian secara BFS adalah 1, 2, 3, 4, 5, 6, 7, 8.

2. DFS

   DFS / *Depth First Search* merupakan sebuah algoritma pencarian secara mendalam yang diterapkan pada sebuah graf, baik graf berarah maupun graf yang tidak berarah. Secara umum algoritma BFS ini melakukan pencarian yang mengunjungi setiap simpul / *vertices* pada sebuah graf secara mendalam, sederhananya adalah setiap simpul dikunjungi dulu sampai ke kedalaman paling akhir lalu melakukan *backtrack* lalu melanjutkan pencariannya kembali secara mendalam. Secara matematis algoritma pencarian DFS ini dapat ditulis sebagai berikut :

   1) Memilih simpul awal / simpul *source*
   2) Kunjungi simpul yang bertetangga dengan simpul nomor 1
   3) Ulangi pemanggilan DFS dari simpul yang didapat pada nomor 2
   4) Ketika mencapai sebuah simpul / *vertices* sedemikian rupa sehingga semua simpul yang bertetangga dengan simpul tersebut sudah dikunjungi semua maka dilakukan runut-balik atau *backtrack* ke simpul terakhir yang dikunjungi sebelumnya dan memiliki tetangga yang dikunjungi jika tidak lakukan terus proses runut-balik ini
   5) Pencarian menggunakan algoritma DFS ini berakhir apabila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi

   Untuk penjelasan lebih lengkap mengenai DFS ini kita bisa lihat kembali gambar 3. Misalkan kita ingin

melakukan traversal secara DFS pada gambar 3 maka pada langkah yang dituliskan diatas kita bisa membuat solusinya seperti dibawah ini :

1) DFS(1) : v = 1 dan *vertices* 1 dikunjungi => DFS(2)
2) DFS(2) : v = 2 dan *vertices* 2 dikunjungi => DFS(4)
3) DFS(4) : v = 4 dan *vertices* 4 dikunjungi => DFS(8)
4) DFS(8) : v = 8 dan *vertices* 8 dikunjungi => DFS(5)
5) DFS(5) : v = 5 dan *vertices* 5 dikunjungi => Backtrack
6) DFS(6) : v = 6 dan *vertices* 6 dikunjungi => DFS(3)
7) DFS(3) : v = 3 dan *vertices* 3 dikunjungi => DFS(7)
8) DFS(7) : v = 7 dan *vertices* 7 dikunjungi => selesai

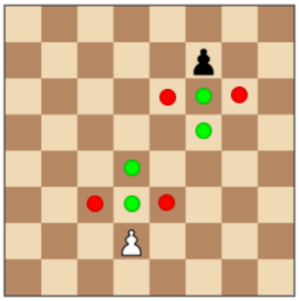
Jadi solusi pencarian dengan menggunakan algoritma DFS adalah 1, 2, 4, 8, 5, 6, 3, 7.

Jadi secara garis besar dapat kita lihat bahwa sebenarnya pencarian menggunakan algoritma BFS dan DFS hampir memiliki sifat yang sama namun algoritma DFS memiliki algoritma yang bersifat rekursif.

C. Catur

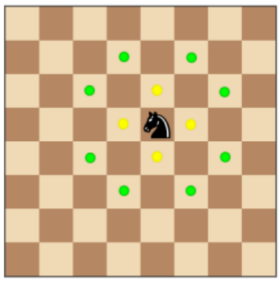
Catur merupakan sebuah permainan sekaligus sebuah cabang olahraga yang dimainkan umumnya oleh 2 orang. Catur sendiri ditemukan pada abad ke-6 di india dan sampai saat ini catur sangat menjadi mendunia. Dalam permainan catur terdapat beberapa jenis bidak catur tersebut dan terdapat beberapa peraturan permainan dari permainan catur ini. Jenis – jenis bidak pada permainan catur ini diantaranya adalah :

- Pion / *pawn* dimana pion ini memiliki aturan gerak yaitu maju sebanyak 1 / 2 kali pada pergerakan pertama dan hanya bisa maju 1 kali pada pergerakan selanjutnya.



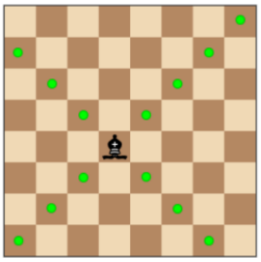
Gambar 4. Langkah Pion (Sumber : https://gurupenjaskes.com/olahraga-catur-sejarah-aturan-dan-cara-bermainnya)

- Kuda / *knight* dimana *knight* ini memiliki aturan gerak yaitu gerak membentuk huruf “L” atau bisa dibilang bergerak dengan kombinasi langkah 2 dan 1 pada sumbu x (kanan/kiri) dan sumbu y (maju / mundur).



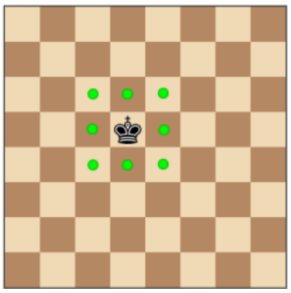
Gambar 5. Langkah Kuda (Sumber : https://gurupenjaskes.com/olahraga-catur-sejarah-aturan-dan-cara-bermainnya)

- Gajah / *bishop* dimana *bishop* ini memiliki aturan bergerak secara diagonal.



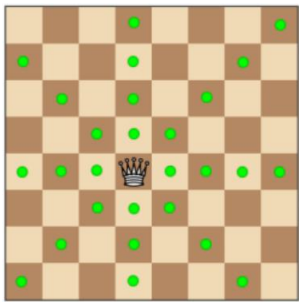
Gambar 6. Langkah Kuda (Sumber : https://gurupenjaskes.com/olahraga-catur-sejarah-aturan-dan-cara-bermainnya)

- Raja / *king* dimana *king* ini memiliki aturan bergerak ke 8 arah mata angin dengan jumlah langkah gerak sebanyak 1.



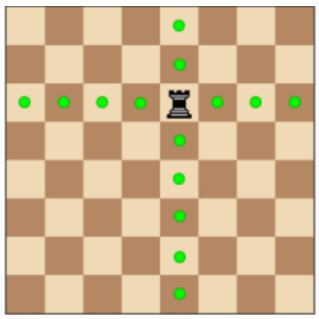
Gambar 7. Langkah Raja (Sumber : https://gurupenjaskes.com/olahraga-catur-sejarah-aturan-dan-cara-bermainnya)

- Ratu / *queen* dimana ratu ini memiliki aturan bergerak ke 8 arah mata angin dimana jumlah langkang gerak dari ratu ini bebas tergantung petak / space kosong yang ada.



Gambar 8. Langkah Mentri (Sumber : https://gurupenjaskes.com/olahraga-catur-sejarah-aturan-dan-cara-bermainnya)

- Benteng / *rook* dimana benteng memiliki aturan bergerak secara vertikal maupun horizontal dengan jumlah langkah tergantung dari jumlah space kosong yang ada.

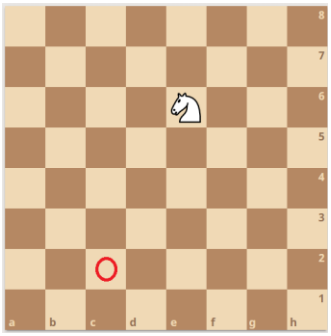


Gambar 9. Langkah Benteng (Sumber : https://gurupenjaskes.com/olahraga-catur-sejarah-aturan-dan-cara-bermainnya)

III. PEMBAHASAN

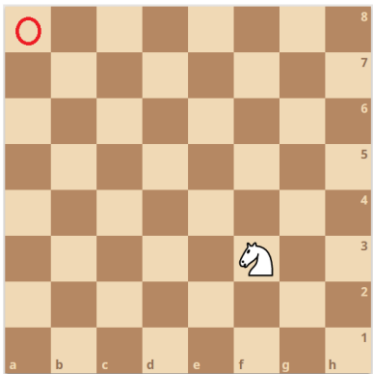
Dalam bagian ini penulis akan melakukan studi kasus dan pembahasan dari topik minimisasi langkah bidak kuda untuk mencapai suatu petak pada permainan catur dengan algoritma BFS (*Breadth First Search*). Terlebih dahulu pada makalah ini penulis akan mencoba 2 kasus uji untuk persoalan topik yang akan dibahas. Berikut kasus 2 kasus uji tersebut yang akan digunakan :

1) Kuda pada posisi e6 dan titik yang akan dituju pada posisi c2.



Gambar 10. Kasus Uji 1

2) Kuda pada posisi f3 dan titik yang akan dituju pada posisi a8



Gambar 11. Kasus Uji 2

Sebelum melakukan pembahasan pada 2 kasus uji diatas penulis akan menjelaskan terlebih dahulu algoritma penyelesaian kasus diatas. Berikut langkah-langkah penyelesaian algoritma diatas :

1. Buat sebuah larik 2 dimensi berukuran NxN dimana N merupakan ukuran papan catur (untuk papan catur yang umum nilai N adalah 8). larik ini digunakan untuk menyimpan nilai apakah suatu petak sudah ditempati atau belum.
2. Tandai posisi awal kuda sebagai *visited*.
3. Buatlah sebuah *queue* untuk menyimpan posisi-posisi selanjutnya yang akan dikunjungi
4. Lakukan *dequeue* pada *queue* lalu bangkitkan semua tetangga *vertices* yang berasal dari *vertices* tersebut. *vertices* yang dibangkitkan berasal dari langkah" kuda yang mungkin dilakukan yaitu langkah kuda yang merupakan kombinasi 1 langkah dan 2 langkah membentuk sebuah huruf "L".
5. Cek apakah *vertices* tersebut merupakan solusi atau bukan jika iya maka pencarian ditemukan jika tidak lanjut ke langkah berikutnya
6. Cek setiap *vertices* yang dibangkitkan lalu cek apakah *vertices* tersebut memenuhi *constraint* yang berlaku jika iya maka, masukkan *vertices* tersebut kedalam sebuah *queue* yang awal lalu total stepnya ditambah 1. Setelah selesai lakukan pengulangan pada nomor 4 sampai ditemukan solusi. Jika semua petak sudah dikunjungi dan tidak ditemukan solusi maka pencarian melakukan *return* tidak berhasil.

Dari analisis diatas kita dapat menemukan solusi dari *problem* diatas dengan kompleksitas big-O sebesar $O(N^2)$. Kompleksitas $O(N^2)$ merupakan kompleksitas pada kasus terburuk karena setiap kotak yang berukuran NxN dikunjungi semuanya satu-satu. Untuk lebih jelasnya penulis akan mencantumkan implementasi kode pada kasus diatas dengan bahasa pemrograman *python*. Berikut implementasi pada kasus minimisasi langkah bidak kuda :

```python
class Chess_cell:
    def __init__(self, x, y, total_steps):
        self.x = x
        self.y = y
        self.total_steps = total_steps


def safe_position(x: int, y: int, n: int):
    return x >= 1 and x <= n and y >= 1 and y <= n
```

Gambar 12. Potongan kode *constraint* dan petak catur

Implementasi dari gambar 12 merupakan implementasi kode untuk *constraint* setiap kali kuda berpindah posisi dan potongan kode kelas *Chess_cell* merupakan kelas untuk posisi catur dan menyimpan total step dari catur yang nantinya merupakan sebuah solusi dari permasalahan diatas.

```python
def bfs_minimum_steps(knight_pos: list, target_pos: list,  n: int):
    # Knight steps
    dx = [1, -1, -2, 2, 1, -1, 2, -2]
    dy = [-2, -2, -1, -1, 2, 2, 1, 1]

    # Queue
    queue: Chess_cell = []
    queue.append(Chess_cell(knight_pos[0], knight_pos[1], 0))

    # Cell
    visited = [[False for j in range(n + 1)] for i in range(n + 1)]
    visited[knight_pos[0]][knight_pos[1]] = True

    while(len(queue) != 0):
        curr: Chess_cell = queue.pop(0)

        if(curr.x == target_pos[0] and curr.y == target_pos[1]):
            return curr.total_steps

        for i in range(8):
            new_x = curr.x + dx[i]
            new_y = curr.y + dy[i]

            if(safe_position(new_x, new_y, n) == True and not(visited[new_x][new_y])):
                queue.append(Chess_cell(new_x, new_y, curr.total_steps + 1))
                visited[new_x][new_y] = True

    return -1
```

Gambar 13. Potongan kode fungsi untuk mencari solusi permasalahan diatas

Implementasi dari gambar 13 merupakan implementasi kode untuk mencari solusi permasalahan diatas. Implementasi kode diatas berdasarkan langkah-langkah yang sudah dijabarkan pada halaman – halaman sebelumnya.

Kasus uji 1 akan dicoba dicari solusinya menggunakan potongan kode diatas. Berikut percobaan untuk kasus uji 1 diatas :

```python
if __name__ == '__main__':
    knight_pos = [5, 6]
    target_pos = [3, 2]

    res = bfs_minimum_steps(knight_pos, target_pos, 8)
    if(res != -1):
        print("total steps {}".format(res))
    else:
        print("no steps")
```

```
(base) λ python minimum-steps-knight.py
total steps 2
```

Gambar 14, 15. Potongan kode main untuk kasus uji 1, Solusi kasus uji 1

Untuk kasus uji 1 kita dapat melihat bahwa langkah minimum dari koordinat <5, 6> (5 merupakan sumbu-x dimana rentang nilai x berada pada $1 \leq x \leq N$ dan 6 merupakan sumbu-y dimana rentang nilai y berada pada $1 \leq y \leq N$) menuju koordinat <3, 2> adalah 2 langkah. Banyak kombinasi untuk mencapai titik tersebut dengan 2 langkah salah satu kemungkinan yang mungkin secara intuitif dapat terlihat dari posisi <5, 6> ke posisi <4, 4> lalu ke posisi <3, 2>. Selanjutnya penulis akan mencari solusi untuk kasus uji 2 menggunakan potongan kode yang sama seperti diatas. Berikut percobaan untuk kasus uji ke-2 :

```python
if __name__ == '__main__':
    knight_pos = [6, 3]
    target_pos = [1, 8]

    res = bfs_minimum_steps(knight_pos, target_pos, 8)
    if(res != -1):
        print("total steps {}".format(res))
    else:
        print("no steps")
```

```
(base) λ python minimum-steps-knight.py
total steps 4
```

Gambar 16, 17. Potongan kode main untuk kasus uji 2, Solusi kasus uji 2

Untuk kasus uji 2 kita dapat melihat bahwa langkah minimum dari koordinat <6, 3> (6 merupakan sumbu-x dimana rentang nilai x berada pada $1 \leq x \leq N$ dan 3 merupakan sumbu-y dimana rentang nilai y berada pada $1 \leq y \leq N$) menuju koordinat <1, 8> adalah 4 langkah. Banyak kombinasi untuk mencapai titik tersebut dengan 4 langkah salah satu kemungkinan yang mungkin secara intuitif dapat terlihat dari posisi <6, 3> ke posisi <5, 5> lalu ke posisi <3, 4> lalu ke posisi <2, 6> lalu yang terakhir ke posisi tujuan yaitu posisi <1, 8>.

Dari kedua kasus uji diatas dapat disimpulkan bahwa untuk persoalan pada papan catur berukuran 8x8 bisa dibilang hampir ditemukan untuk segala kemungkinan *test case*. Selain itu kode diatas juga tidak hanya berlaku untuk papan catur berukuran 8 x 8 namun kode diatas bisa untuk menghitung *problem* dengan nilai N yang besar (secara umum bisa dikatakan bisa ditemukan solusi sampai ukuran N x N).

Pada papan catur berukuran 8 x 8 masih mudah untuk menentukan solusi untuk permasalahan diatas namun untuk N yang semakin membesar makin sulit untuk ditemukan solusinya sehingga disinilah algoritma *Breadth First Search* dibutuhkan.

IV. KESIMPULAN

Algoritma BFS (Breadth First Search) ini bisa dikatakan selalu berhasil untuk mencari solusi permasalahan minimisasi langkah bidak catur untuk mencapai suatu koordinat petak tertentu. Algoritma ini juga cukup baik jika N dari papan catur tidak terlalu besar. Hasil dari percobaan kasus uji diatas juga selalu memberikan nilai yang valid sehingga bisa dikatakan percobaan diatas berhasil. Adapun solusi alternatif diatas yaitu menggunakan pendekatan dynamic programming, algoritma A*, dll.

V. UCAPAN TERIMAKASIH

Puji syukur pertama-tama saya panjatkan kepada Tuhan Yang Maha Esa karena berkat dan rahmat-Nya saya dapat menyelesaikan makalah ini dengan baik dan benar. Saya juga mengucapkan terima kasih kepada Bapak Rinaldi Munir selaku pengajar mata kuliah Strategi Algoritma dan pembimbing dalam pembuatan makalah ini. Tidak lupa juga saya ucapkan terimakasih kepada keluarga dan teman-teman tercinta yang tidak henti-hentinya memberikan dukungan moral serta bantuan yang tidak bisa disebutkan satu persatu.

VI. LINK YOUTUBE

https://www.youtube.com/watch?v=WOt3JRK4h2M

VII. REFERENSI

[1]https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf
[2]https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf
[3] https://gurupenjaskes.com/olahraga-catur-sejarah-aturan-dan-cara-bermainnya
[4]https://www.geeksforgeeks.org/minimum-steps-reach-target-knight/

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, 11 Mei 2021

Karel Renaldi 13519180