Pengaplikasian Algoritma DFS dalam Pencarian Jalur Terpendek pada Permainan Ular Tangga

Arsa Daris Gintara 13519037
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519037@std.stei.itb.ac.id

***Abstraksi*—Permainan Ular Tangga adalah permainan klasik yang sering dimainkan anak-anak hingga orang dewasa dengan tujuan mencapai garis *fnisih* terlebih dahulu. Permainan ini sejatinya dilakukan oleh dua orang atau lebih sehingga mencapai blok *finish* menjadi perlombaan yang menjadi inti dari pemainan yang dalam bahasa inggris disebut *Snakes and Ladders* atau *Chutes and Ladders* ini. Permainan ini biasa dilakukan pada tabel berukuran 8x8, 10x10, atau 12x12. Algoritma DFS dapat digunakan untuk mencari jalur terpendek yang mungkin dilakukan untuk memenangkan permainan ini terlebih dahulu.**

Keywords—ular tangga; DFS; pencarian; jalur terpendek;

I. PENDAHULUAN

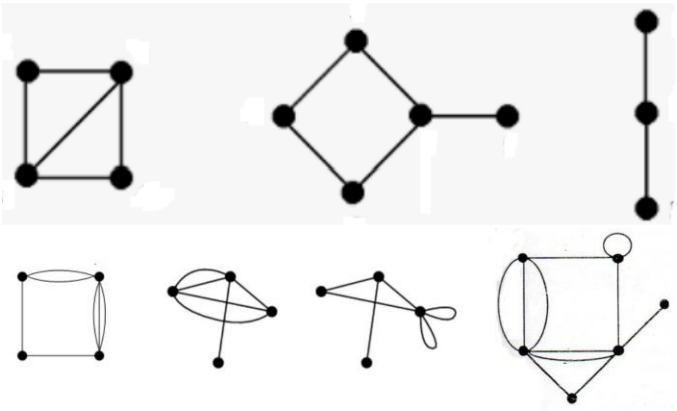
Permainan Ular Tangga adalah permainan sederhana yang sering dimainkan oleh khususnya anak-anak hingga orang dewasa. Permainan ini biasanya dilakukan oleh dua orang atau lebih dengan tujuan untuk mencapai garis *finish* terlebih dahulu. Permainan ini dimainkan pada tabel yang umumnya berukuran 8x8, 10x10, hingga 12x12 dengan tiap bloknya memiliki angka dan elemen tertentu. Pada permainan ini terdapat elemen yang menjadi "bonus" untuk pemainnya yaitu tangga yang berarti maju ke beberapa blok ke depan dan "jebakan" yaitu ular yang berarti mundur beberapa blok ke belakang. Permainan ini biasanya dimainkan dengan menggunakan bantuan dadu sebagai alat bantu giliran antar pemain.

Algoritma Depth First Search(DFS) adalah suatu algoritma pencarian yang memiliki konsep mencari secara mendalam terhadap suatu graf atau pohon yang terhubung. Algortima DFS ini dapat digunakan untuk menyelesaikan permasalahan pencarian jalur terpendek pada permainan ular tangga ini. Konsepnya algoritma ini akan mengunjungi blok *start* kemudian membangkitkan tetangga yang ada, lalu mengunjungi tetangga tersebut. Hal ini dilakukan hingga mencapai blok *finish* atau sudah tidak ada elemen yang dapat dikunjungi lagi. Jika mencapai blok yang tidak memiliki tetangga yang dapat dikunjungi maka akan dilakukan *backtrack* ke blok sebelumnya dan dilanjutkan mengunjungi tetangga lain.

II. DASAR TEORI

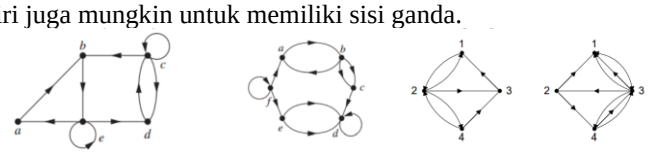
A. *Graf*

Graf adalah kumpulan objek-objek yang terdiri dari simpul dan sisi. Graf dilambangkan sebagai G = (V, E) dengan V sebagai simpul dan E sebagai sisi. Graf sendiri digunakan untuk merepresentasikan objek-objek yang diskrit dan hubungan antara objek-objek tersebut. Graf memiliki jenis-jenis tertentu. Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, maka dibagi menjadi 2 yaitu graf sederhana dan graf tak-sederhana. Sedangkan pada graf tak-sederhana sendiri dibagi lagi menjadi 2 yaitu graf ganda dan graf semu.



Gambar 1. Graf sederhana dan graf tak-sederhana (sumber [1])

Graf sederhana adalah graf yang tidak memiliki gelang maupun sisi ganda. Kemudian graf ganda adalah graf yang memiliki sisi ganda dan tidak memiliki gelang. Sedangkan graf semu adalah graf yang memiliki sisi gelang atau sisi yang menghubungkan simpul ke simpul itu sendiri, graf semu ini sendiri juga mungkin untuk memiliki sisi ganda.



Gambar 2. Graf berarah (sumber [1])

Berdasarkan orientasi arah pada sisi, graf dibagi menjadi dua yaitu graf tak-berarah dan graf berarah. Graf berarah sendiri dibagi lagi menjadi dua yaitu graf berarah dan graf-ganda berarah. Graf tak-berarah adalah graf yang sisinya tidak

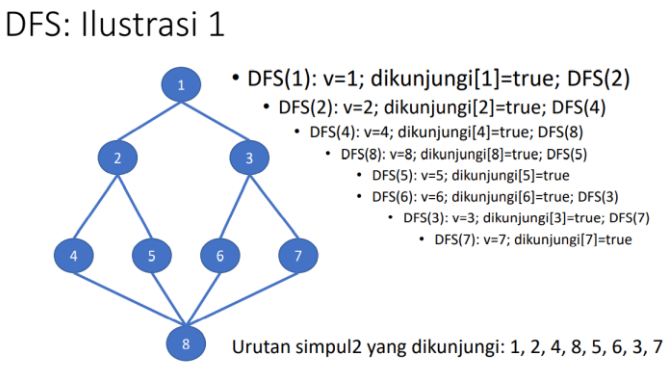
memiliki orientasi arah. Sedangkan graf berarah adalah graf yang sisinya memiliki orientasi arah, pada graf-ganda berarah terdapat sisi berarah yang ganda.

B. DFS

DFS (*Depth First Search*) adalah algoritma pencarian yang konsepnya mencari secara mendalam dari suatu graf atau pohon dengan ide *backtrack*. Algoritma ini akan menelusuri suatu graf atau pohon secara mendalam jika memungkinkan, atau akan backtrack jika tidak memungkinkan.

Secara umum langkah-langkah dari Algoritma DFS adalah sebagai berikut:

1. Traversal dimulai dari simpul v.
2. Kunjungi simpul v.
3. Kunjungi simpul w yang bertetangga dengan simpul v.
4. Ulangi DFS mulai dari simpul w.
5. 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.
6. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.



Gambar 3. Ilustrasi DFS pada graf (sumber [2])

C. Permainan Ular Tangga

Permainan Ular Tangga atau dalam bahasa inggris dikenal dengan *Snakes and Ladders* atau *Chutes and Ladders* adalah sebuah permainan tabel dengan tujuan mencapai blok *finish* dari suatu blok *start*. Permainan ini sejatinya dimainkan dua orang atau lebih dan giliran bermain dibantu menggunakan alat seperti *spinner* atau yang biasa kita ketahui dadu.

Permainan klasik ini dipercaya berasal dari India pada 2nd century B.C, dimana permainan ini merupakan alegori dari perjalanan hidup dengan naik karena takdir/karma direpresentasikan tangga dan mundur oleh nafsu/kama direpresentasikan oleh ular.

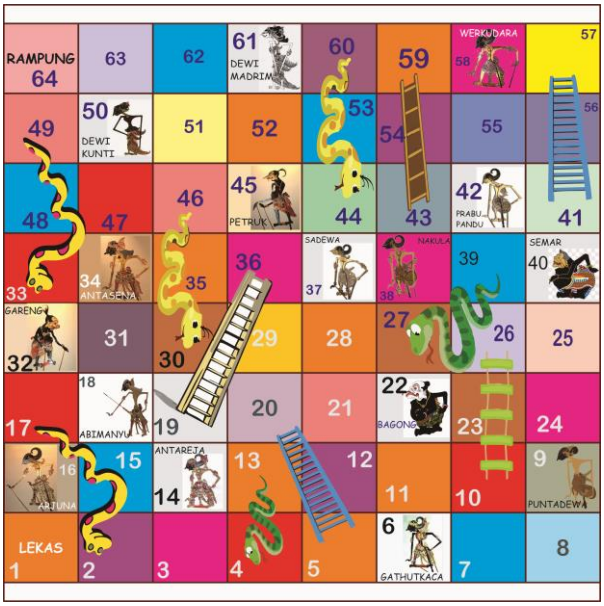
Gambar 4. Ilustrasi Permainan Ular Tangga (sumber [4])

Pada permainan ular tangga, seperti namanya terdapat elemen ular. Ular yang dimaksud disini adalah "jebakan" ketika menginjak suatu blok yang memiliki elemen ini yang akan berakibat pemain turun ke blok tertentu dimana ekor dari ular berada yang tentunya merugikan pemain yang menginjaknya. Elemen ular ini tentunya dihindari oleh pemain ular tangga ini.

Pada permainan ular tangga ini pula, dikenal elemen tangga yang merupakan "bonus" ketika menginjak suatu blok yang memiliki elemen ini. Pemain yang menginjak blok dengan elemen ini akan mendapatkan bonus maju beberapa blok ke depan sesuai dengan ujung dari tangga yang diinjaknya.

III. Pengaplikasian Algoritma DFS dalam Pencarian Jalur Terpendek pada Permainan Ular Tangga

Untuk dapat menggambarkan pengaplikasian DFS pada permainan ini, digunakan contoh ular tangga berukuran 8x8 sederhana seperti pada gambar 3 dibawah. Pengaplikasiannya sendiri digunakan bahasa Python.



Gambar 5. Contoh Ular Tangga Berukuran 8x8 (sumber [5])

Ular tangga tersebut direpresentasikan dengan *list* dengan tiap indeks nya merepresentasikan nomor bloknya. Nomor blok ini sendiri memiliki pengecualian pada blok yang merupakan tangga atau ular, dengan nomor blok pada list ini berupa blok tujuan. Terlihat pada gambar indeks 0 diisi dengan blok 1, indeks 1 diisi dengan blok 2, indeks 4 yang merupakan blok 5 dengan tangga yang menuju blok 20, indeks 59 yang merupakan blok 60 dengan ular yang menuju blok 44, dan seterusnya.

```
board = [1,2,3,4,20,6,7,8,
         9,26,11,12,4,14,15,16,
         2,18,36,20,21,22,23,24,
         25,26,27,28,29,30,31,32,
         33,34,35,36,37,38,27,40,
         57,42,59,44,45,30,47,48,
         33,50,51,52,53,54,55,56,
         57,58,59,44,61,62,63,64]
```

Gambar 6. Representasi Ular Tangga dalam Bentuk List

A. Penjelasan Algortima

Secara garis besar algoritma DFS yang digunakan pada pengaplikasian ini sebagai berikut:

while belumMencapaiTujuan() and stillQueue() do

kunjungiPopQueue()

foreach tetanggaPopQueue()

if tetanggaNotVisited then

pushQueue()

endif

endforeach

endwhile

Lebih lanjut, pada awal program queue akan diinisialisasi dengan blok awal yaitu blok 1 dengan indeks 0 pada representasi list. Queue (simpul hidup) juga diinisialisasi berupa tetangga dari blok 1 tersebut yang merupakan hasil lemparan dadu dengan kemungkinan dadu yang mungkin muncul adalah 1, 2, 3, 4, 5, dan 6. Sehingga tetangga tersebut merupakan penjumlahan dari indeks 0 dengan kemungkinan dadu yang muncul.

Diinisialisasikan juga sebuah list yang merepresentasikan suatu indeks/blok sudah dikunjungi atau belum dengan inisialisasi awal bernilai False.

```
def DFSSnL(theBoard):
  visitedBlock = [False for i in range(len(theBoard))]
  currBlockIndex = 0
  currBlock = theBoard[currBlockIndex]
  theGoal = theBoard[-1]
  toReturn = []
  theQueue = [] # in index

  # initialize first block visited
  theQueue.append(currBlockIndex)

  # tetangga currBlock awal
  for i in range(1,7):
    if (not(visitedBlock[currBlockIndex+i])):
      theQueue.append(currBlockIndex+i)
```

Gambar 7. Inisialisasi

Kemudian diiterasi selama belum mencapai blok tujuan yaitu blok 64 atau indeks 63 dan queue (simpul hidup) masih ada. Iterasi diawali dengan pengambilan elemen pertama pada queue atau bisa disebut pop. Elemen yang diambil ini ditandai telah dikunjungi sehingga nantinya tidak dikunjungi lagi. Elemen ini selanjutnya ditambahkan pada jalur pencarian hasil.

Selanjutnya pada elemen yang diambil, dicari tetangga-tetangga yang belum dikunjungi yang merupakan hasil penjumlah indeks saat ini dengan lemparan dadu dengan kemungkinan dadu yang mungkin muncul adalah 1, 2, 3, 4, 5, 6.

Jika pada elemen yang dipilih tidak memiliki tetangga, maka akan dilakukan *backtrack*, dengan elemen terakhir dari jalur dihapus dari jalur pencarian. Kemudian dilanjutkan iterasinya.

Jika pada elemen pertama pada queue ini memiliki tetangga maka, tetangga tersebut akan dimasukkan menjadi elemen-elemen pertama menggeser elemen-elemen yang sudah ada sebelumnya. Kemudian dilanjutkan iterasi DFSnya.

```
iterasi = 1
  while (len(theQueue) != 0 and currBlock != theGoal):
    currBlockIndex = theQueue[0]
    currBlock = theBoard[currBlockIndex]
    visitedBlock[currBlockIndex] = True

    # print
    print("Iterasi", iterasi)
    print("Current Block =", currBlock)

    # pengaman untuk tangga dan ular
    currBlockIndex = currBlock-1
    visitedBlock[currBlockIndex] = True

    # tambah turn dan hapus indeks 0 queue
    toReturn.append(currBlock)
    theQueue.pop(0)
    print("Simpul hidup = ", end='')
    print("[", end='')
    for sh in theQueue:
      print(str(sh+1) + ",", end='')
    print("]", end='')
    print("\n")

    if (currBlock == theGoal):
      break
    else:
      # find tetangga
      toAddtheQueue = []
      j = 1
      while (j <= 7 and len(theBoard) > currBlockIndex+j):
        if (not(visitedBlock[currBlockIndex+j])):
          toAddtheQueue.append(currBlockIndex+j)
        j = j+1

      # has no tetangga
      if (len(toAddtheQueue) == 0):
        toReturn.pop(-1)
      else:
        tmpQueue = theQueue
        theQueue = toAddtheQueue
        for k in tmpQueue:
          theQueue.append(k)

    iterasi = iterasi+1
```

Gambar 8. Iterasi

B. Ilustrasi DFS

Ilustrasi DFS pada contoh ular tangga diatas adalah sebagai berikut dalam representasi tabel:

Tabel 1. Ilustrasi DFS

Iterasi ke-	**Blok**	**Queue (Simpul Hidup)**
1	1	2,3,4,5,6,7
2	2	3,4,5,6,7,8,2,3,4,5,6,7
3	3	4,5,6,7,8,9,3,4,5,6,7,8,2,3,4,5,6,7
4	4	5,6,7,8,9,10,4,5,6,7,8,9,3,4,5,6,7,8,2,3, 4,5,6,7
5	20	6,7,8,9,10,11,5,6,7,8,9,10,4,5,6,7,8,9,3, 4,5,6,7,8,2,3,4,5,6,7
6	21	22,23,24,25,26,27,6,7,8,9,10,11,5,6,7,8, 9,10,4,5,6,7,8,9,3,4,5,6,7,8,2,3,4,5,6,7
7	22	23,24,25,26,27,28,22,23,24,25,26,27,6, 7,8,9,10,11,5,6,7,8,9,10,4,5,6,7,8,9,3,4, 5,6,7,8,2,3,4,5,6,7
8	23	24,25,26,27,28,29,23,24,25,26,27,28,22, 23,24,25,26,27,6,7,8,9,10,11,5,6,7,8,9, 10,4,5,6,7,8,9,3,4,5,6,7,8,2,3,4,5,6,7
9	24	25,26,27,28,29,30,24,25,26,27,28,29,23, 24,25,26,27,28,22,23,24,25,26,27,6,7,8, 9,10,11,5,6,7,8,9,10,4,5,6,7,8,9,3,4,5,6, 7,8,2,3,4,5,6,7
10	25	26,27,28,29,30,31,25,26,27,28,29,30,24, 25,26,27,28,29,23,24,25,26,27,28,22,23, 24,25,26,27,6,7,8,9,10,11,5,6,7,8,9,10,4, 5,6,7,8,9,3,4,5,6,7,8,2,3,4,5,6,7
11	26	27,28,29,30,31,32,26,27,28,29,30,31,25, 26,27,28,29,30,24,25,26,27,28,29,23,24, 25,26,27,28,22,23,24,25,26,27,6,7,8,9, 10,11,5,6,7,8,9,10,4,5,6,7,8,9,3,4,5,6,7, 8,2,3,4,5,6,7
12	27	28,29,30,31,32,33,27,28,29,30,31,32,26, 27,28,29,30,31,25,26,27,28,29,30,24,25, 26,27,28,29,23,24,25,26,27,28,22,23,24, 25,26,27,6,7,8,9,10,11,5,6,7,8,9,10,4,5, 6,7,8,9,3,4,5,6,7,8,2,3,4,5,6,7
13	28	29,30,31,32,33,34,28,29,30,31,32,33,27, 28,29,30,31,32,26,27,28,29,30,31,25,26, 27,28,29,30,24,25,26,27,28,29,23,24,25, 26,27,28,22,23,24,25,26,27,6,7,8,9,10, 11,5,6,7,8,9,10,4,5,6,7,8,9,3,4,5,6,7,8, 2,3,4,5,6,7
14	29	30,31,32,33,34,35,29,30,31,32,33,34,28, 29,30,31,32,33,27,28,29,30,31,32,26,27, 28,29,30,31,25,26,27,28,29,30,24,25,26, 27,28,29,23,24,25,26,27,28,22,23,24,25, 26,27,6,7,8,9,10,11,5,6,7,8,9,10,4,5,6,7, 8,9,3,4,5,6,7,8,2,3,4,5,6,7
Seterusnya		

44	63	64,63,64,62,63,61,62,56,61,55,56,54,55, 56,53,54,55,56,52,53,54,55,56,51,52,53, 54,55,49,50,51,52,53,54,48,49,50,51,52, 46,47,48,49,50,51,61,62,63,64,60,61,62, 63,64,59,60,61,62,63,64,42,43,44,45,46, 47,41,42,43,44,45,39,40,41,42,43,44,38, 39,40,41,42,43,37,38,39,40,41,42,36,37, 38,39,40,41,35,36,37,38,39,40,34,35,36, 37,38,39,33,34,35,36,37,38,32,33,34,35, 36,37,31,32,33,34,35,36,30,31,32,33,34, 35,29,30,31,32,33,34,28,29,30,31,32,33, 27,28,29,30,31,32,26,27,28,29,30,31,25, 26,27,28,29,30,24,25,26,27,28,29,23,24, 25,26,27,28,22,23,24,25,26,27,6,7,8,9, 10,11,5,6,7,8,9,10,4,5,6,7,8,9,3,4,5,6,7, 8,2,3,4,5,6,7
45	64	64,63,64,62,63,61,62,56,61,55,56,54,55, 56,53,54,55,56,52,53,54,55,56,51,52,53, 54,55,49,50,51,52,53,54,48,49,50,51,52, 46,47,48,49,50,51,61,62,63,64,60,61,62, 63,64,59,60,61,62,63,64,42,43,44,45,46, 47,41,42,43,44,45,39,40,41,42,43,44,38, 39,40,41,42,43,37,38,39,40,41,42,36,37, 38,39,40,41,35,36,37,38,39,40,34,35,36, 37,38,39,33,34,35,36,37,38,32,33,34,35, 36,37,31,32,33,34,35,36,30,31,32,33,34, 35,29,30,31,32,33,34,28,29,30,31,32,33, 27,28,29,30,31,32,26,27,28,29,30,31,25, 26,27,28,29,30,24,25,26,27,28,29,23,24, 25,26,27,28,22,23,24,25,26,27,6,7,8,9, 10,11,5,6,7,8,9,10,4,5,6,7,8,9,3,4,5,6,7, 8,2,3,4,5,6,7

C. Hasil Pencarian

Jalur terpendek yang didapatkan dengan menggunakan algoritma DFS adalah 1-2-3-4-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-40-57-58-59-44-45-47-48-50-51-52-53-54-55-56-61-62-63-64.
Dengan total giliran 42 kali.

D. Penjelasan Jalur

Pada iterasi ke 1 dikunjungi blok awal yaitu blok 1 (indeks 0 dalam representasi list) dan dibangkitkan tetangga yang belum dikunjungi yaitu blok 2,3,4,5,6, dan 7. Masuk ke hasil jalur pencarian.

Pada iterasi ke 2 – 4 dikunjungi blok biasa tak berelemen dan dibangkitkan tetangga yang belum dikunjungi. Ditambahkan ke hasil jalur pencarian.

Pada iterasi ke 5 dikunjungki blok 5 yang berelemen tangga sehingga pemain langsung maju ke blok nomer 20 yang merupakan ujung dari tangga. Dibangkitkan juga tetangga yang belum dikunjungi dari blok 20. Masuk ke hasil jalur pencarian.

Pada iterasi ke 6 – 23 dikunjungi blok biasa tak berelemen dan dibangkitkan tetangan yang belum dikunjungi pada masing masing blok yang dikunjungi pada iterasi tersebut. Ditambahkan ke hasil jalur pencarian.

Pada iterasi ke 24 dikunjungi blok 39 berelemen ular yang menuju ke blok 27 (sudah dikunjungi), maka dilakukan *backtrack.*

Pada iterasi ke 25 dikunjungi blok tetangga berikutnya dari elemen terakhir dikunjungi yaitu menuju blok 40. Dibangkitkan tetangga dari blok ini yang belum bertetangga dan blok 40 dimasukkan ke jalur pencarian.

Pada iterasi ke 26 dikunjungi blok 41 yang berelemen tangga sehingga maju ke blok 57 dan dibangkitkan tetangga dari blok 57. Blok 57 masuk ke jalur pencarian.

Pada iterasi ke 27-28 dikunjungi blok biasa tak berelemen dan dibangkitkan tetangga yang belum dikunjungi. Setiap blok masuk ke hasil jalur pencarian.

Pada iterasi ke 29 dikunjungi blok 60 yang berelemen ular sehingga mundu ke blok 44 yang merupakan ekor dari ular. Blok 44 masuk ke hasil jalur pencarian.

Pada iterasi ke 30 dikunjungi blok 45 yang merupakan blok biasa tak berelemen. Masuk ke hasil jalur pencarian.

Pada iterasi ke 31 dikunjungi blok 46 yang merupakan ular menuju blok 30 (sudah dikunjungi) sehingga dilakukan *backtrack*.

Pada iterasi ke 32 dikunjungi blok tetangga yang lain dari blok sebelumnya yaitu blok 47 yang merupakan blok biasa dan dimasukkan ke dalam hasil jalur pencarian.

Pada iterasi ke 33 dikunjungi blok 48 yang merupakan blok biasa dan dimasukkan ke dalam hasil jalur pencarian.

Pada iterasi ke 34 dikunjungi blok 49 yang berelemen ular menuju blok 33 yang sudah dikunjungi sehingga dilakukan *backtrack*.

Pada iterasi ke 35 dikunjungi blok tetangga yang lain dari blok sebelumnya yaitu blok 50 yang merupakan blok biasa dan masuk ke hasil jalur pencarian.

Pada iterasi ke 36 – 41 dikunjungi blok biasa tak berelemen yang merupakan hasil pembangkitan dari blok sebelumnya. Dimasukkan ke dalam rute hasil pencarian.

Pada iterasi ke 42 dikunjungi blok yang melompati blok-blok yang sudah dikunjungi sebelumnya sehingga masuk ke blok 61 yang merupakan blok biasa tak berelemen dan masuk ke rute hasil pencarian.

Pada iterasi ke 43 – 44 dikunjungi blok biasa tak berelemen dan masuk ke rute hasil pencarian.

Pada iterasi ke 45 dikunjungi blok 64 yang merupakan blok *finish* sehingga keluar dari iterasi dan didapatkan rute hasil pencarian dengan banyak jalur sebesar 42, *backtrack* sebanyak 3 kali, dan memiliki jalur 1-2-3-4-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-40-57-58-59-44-45-47-48-50-51-52-53-54-55-56-61-62-63-64.

IV. KESIMPULAN

Pada bab sebelumnya didapatkan hasil iterasi dan hasil jalur pencarian suatu contoh ular tangga berukuran 8x8 dengan didapat jalur sejumlah 42 jalur. Hal ini menandakan bahwa Algoritma DFS pada pengaplikasian pencarian jalur terpendek pada permainan ular tangga tidak selamanya optimum (minimum), sebab masih dapat ditemukan jalur lain yang lebih minimum.

VIDEO LINK AT YOUTUBE *(Heading 5)*

https://youtu.be/7u2Be8EF0tg

UCAPAN TERIMAKASIH

Atas berkat rahmat Allah SWT Tuhan Yang Maha Esa penulis dapat menyelesaikan karyanya yang berjudul "Pengaplikasian Algoritma DFS dalam Pencarian Jalur Terpendek pada Permainan Ular Tangga". Penulis juga mengucapkan terima kasih terhadap kedua orang tua yang selalu mendukung dan menyanyangi penulis, teman-teman penulis yang juga mendukung penulis, dan para Bapak dan Ibu Dosen ITB yang telah membimbing mahasiswanya.

REFERENCES

[1] Munir, Rinaldi. "Graf Bagian 1". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf (Diakses 10 Mei 2021)

[2] Munir, Rinaldi. "Breadth/Depth First Search (BFS/DFS) Bagian 1". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf (Diakses 10 Mei 2021)

[3] Munir, Rinaldi. "Breadth/Depth First Search (BFS/DFS) Bagian 2". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf (Diakses 10 Mei 2021)

[4] Arneson, Erik. "Chutes and Ladders – Snakes and Ladders". https://www.thesprucecrafts.com/chutes-and-ladders-snakes-and-ladders-411609 (Diakses 10 Mei 2021)

[5] Hassa, Alikta. "Ular Tangga Wayang". https://aliktahassa.wordpress.com/2019/08/22/set-permainan-ular-tangga-wayanh/ular-tangga-wayang/ (Diakses 10 Mei 2021)

[6] Educative. "What is Depth First Search?". https://www.educative.io/edpresso/what-is-depth-first-search (Dieakses 10 Mei 2021)

[7] Programiz. "Depth First Search". https://www.programiz.com/dsa/graph-dfs (Diakses 10 Mei 2021)

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.

Surabaya, 11 Mei 2021

Arsa Daris Gintara 13519037