Analisis Perbandingan Algoritma *Recursive Backtracker* Pada *Maze Problem*

LAPORAN STRATEGI ALGORITMA

Ditujukan sebagai salah satu syarat untuk memperoleh nilai pada
mata kuliah Strategi Algoritma
program studi DIV Teknik Informatika

Oleh:

Shinta Raudita Octora Damayanti – 1214063

Adam Ghafara - 1214064

PROGRAM DIPLOMA IV TEKNIK INFORMATIKA

UNIVERSITAS LOGISTIK DAN BISNIS INTERNASIONAL

BANDUNG

2023

1. Pendahuluan

1.1. Pengertian *Maze Problem*

Maze problem merupakan kasus umum pada navigasi otonom atau otomatis robot. Dalam *maze problem*, menemukan jalan keluar dengan cepat adalah salah satu indikator penting untuk mengukur kualitas algoritma. Mengurangi eksplorasi yang tidak perlu dapat menyelesaikan masalah ini dengan baik

1.2. Pengertian Algoritma *Recursive Backtracker*

1.2.1. Jeff Erickson

Dosen di University Illinois Urbana-Champaign (2018)

Recursive backtracking adalah teknik algoritma untuk memecahkan masalah secara rekursif dengan mencoba membangun solusi secara bertahap, satu per satu. Algoritma ini dimulai dari titik acak dan mencoba untuk bergerak ke arah acak. Jika mencapai titik buntu, algoritma kembali ke titik terakhir yang memiliki jalur yang belum dieksplorasi dan mencoba kembali. Algoritma ini mengulangi proses ini sampai mencapai titik awal dan mengeksplorasi semua jalur yang mungkin.

1.2.2. Godfry Justo

Dosen di University Dar es Salaam (UDSM) (2021)

Pemrosesan kembali secara rekursif melihat bahwa ruang solusi masalah terdiri dari keadaan dan tindakan. Ketika kita mencoba sebuah tindakan dan gagal, kita kembali ke keadaan sebelumnya dan mencoba tindakan lain. Jika semua tindakan menghasilkan kegagalan, kita harus mencoba tindakan lain lagi.

2. Analisis dan Perancangan

2.1. Analisis

2.1.1. Analisis Berdasarkan Pendapat Shinta

Recursive backtracking adalah algoritma yang memecahkan masalah secara rekursif/berulang dengan membangun solusi satu per satu. Algoritma ini mencoba untuk bergerak ke arah acak dan jika sampai pada titik buntu, kembali ke titik terakhir yang belum dieksplorasi dan mencoba kembali.

Algoritma ini terus mengulangi proses ini sampai semua jalur yang mungkin dieksplorasi. Konsepnya adalah mencoba tindakan dan kembali ke keadaan sebelumnya

jika tindakan tersebut gagal, mencoba tindakan lainnya jika semua tindakan sebelumnya gagal.

2.1.2. Analisis Berdasarkan Pendapat Adam

Recursive Backtracking adalah salah satu metode algoritma yang digunakan untuk membuat labirin. Algoritma dimulai dengan membuat pola jaringan sel, di mana setiap sel mewakili satu titik dalam labirin. Algoritma ini kemudian memilih sel awal secara acak dan menandainya sebagai sel yang telah dikunjungi.

Algoritma *Recursive Backtracking* mengikuti berbagai tahap berikut:

1. Menentukan Ruang Pencarian, Permasalahan dibuatkan dalam ruang jaringan, dengan beberapa solusi yang tersembunyi didalamnya.
2. Memilih Jalan, Algoritma akan memilih jalan ke berbagai ruang yang dimulai dari node paling ujung.
3. Memeriksa Kelayakan, algoritma melakukan pengecekan terhadap sebagian jalur apakah layak dilalui atau tidak.
4. Jelajah, Algoritma akan menjelajah terus menerus ke berbagai jalur dalam jaringan yang dimungkinkan dapat dilalui dan meluas dari satu node dalam jaringan tersebut.
5. *Backtrack*, Jika jalur telah dilalui, algoritma akan kembali menuju node yang sebelumnya dan berjalan Kembali menuju jalur lainnya yang belum dilalui didalam jaringan.
6. *Terminate*, algoritma akan melakukan *terminate* atau menghentikan sistemnya jika sudah didapatkan solusi kelayakan ditemukan atau jika semua jalur pada ruang jaringan telah ditempuh.

2.2. Perbandingan

Secara umum, Pengertian oleh para ahli Jeff dan analisis penulis memaparkan definisi dan cara kerja algoritma *recursive backtracking* dengan lebih detail dan spesifik. Sedangkan Godfry lebih menyoroti konsep dasar dari algoritma tersebut dan memberikan gambaran umum tentang bagaimana algoritma itu bekerja.

2.2. Perancangan Algoritma

2.3. Source Code

SOURCE CODE

def carve_out_maze(x,y):
  single_cell(x, y) # starting positing of maze
  stack.append((x,y)) # place starting cell into stack
  visited.append((x,y)) # add starting cell to visited list

Penjelasan : Fungsi ini berfungsi untuk memulai pembuatan maze dengan menempatkan sel awal ke dalam maze, menambahkannya ke dalam stack, dan menambahkannya ke dalam daftar sel yang sudah dikunjungi.

SOURCE CODE

while len(stack) > 0: # loop until stack is empty
  time.sleep(.07) # slow program now a bit
  cell = [] # define cell list
  if (x + w, y) not in visited and (x + w, y) in grid: # right cell available?
  cell.append("right") # if yes add to cell list
 	
  if (x - w, y) not in visited and (x - w, y) in grid: # left cell available?
  cell.append("left")
 	
  if (x , y + w) not in visited and (x , y + w) in grid: # down cell available?
  cell.append("down")
 	
  if (x, y - w) not in visited and (x , y - w) in grid: # up cell available?
  cell.append("up")

Penjelasan : Ini adalah blok kode yang menjelaskan bagaimana mencari sel yang belum dikunjungi (yang dapat menjadi bagian dari jalur yang sedang dibangun) dalam grid maze. Baris 5: Mulai loop while hingga stack kosong. Baris 6: Untuk memperlambat program agar lebih mudah dilihat, program akan menunggu selama 0,07 detik pada setiap iterasi. Baris 7: Membuat list kosong untuk cell yang dapat dipilih selanjutnya untuk dijadikan jalur. Baris 8-11: Melakukan pemeriksaan untuk sel ke sebelah kanan, kiri, bawah, dan atas dari sel saat ini, jika sel belum dikunjungi (belum ada dalam list visited) dan ada dalam grid maze, maka sel tersebut ditambahkan ke dalam cell list. Jadi pada blok kode ini, program memeriksa sel yang belum dikunjungi di sekitar sel saat ini. Sel yang belum dikunjungi ditambahkan ke cell list.

SOURCE CODE

```
    if len(cell) > 0:                          # check to see if cell list is empty
       cell_chosen = (random.choice(cell))          # select one of the cell randomly

       if cell_chosen == "right":                    # if this cell has been chosen
          push_right(x, y)                        # call push_right function
          solution[(x + w, y)] = x, y                # solution = dictionary key = new cell,
other = current cell
          x = x + w                               # make this cell the current cell
          visited.append((x, y))                  # add to visited list
          stack.append((x, y))                    # place current cell on to stack

       elif cell_chosen == "left":
          push_left(x, y)
          solution[(x - w, y)] = x, y
          x = x - w
          visited.append((x, y))
          stack.append((x, y))

       elif cell_chosen == "down":
          push_down(x, y)
          solution[(x , y + w)] = x, y
          y = y + w
          visited.append((x, y))
          stack.append((x, y))

       elif cell_chosen == "up":
          push_up(x, y)
          solution[(x , y - w)] = x, y
          y = y - w
          visited.append((x, y))
          stack.append((x, y))
```

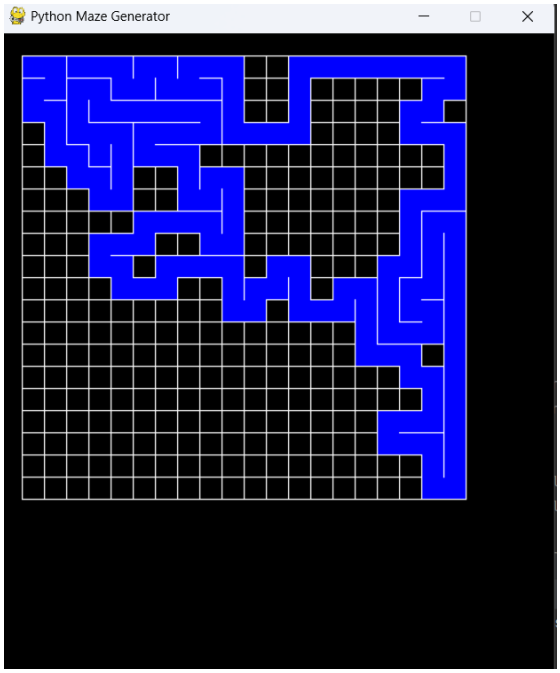
Penjelasan :	Baris 1: Memeriksa apakah ada elemen di dalam daftar stack. Baris 2: Membuat sebuah daftar kosong yang akan diisi dengan informasi sel mana saja yang tersedia untuk dipilih. Baris 3-10: Memeriksa apakah sel yang tersedia di kanan, kiri, bawah, dan atas. Jika iya, menambahkannya ke dalam daftar yang di buat pada baris 2. Baris 11: Memeriksa apakah daftar sel yang tersedia tidak kosong. Baris 12: Memilih salah satu sel secara acak dari daftar sel yang tersedia

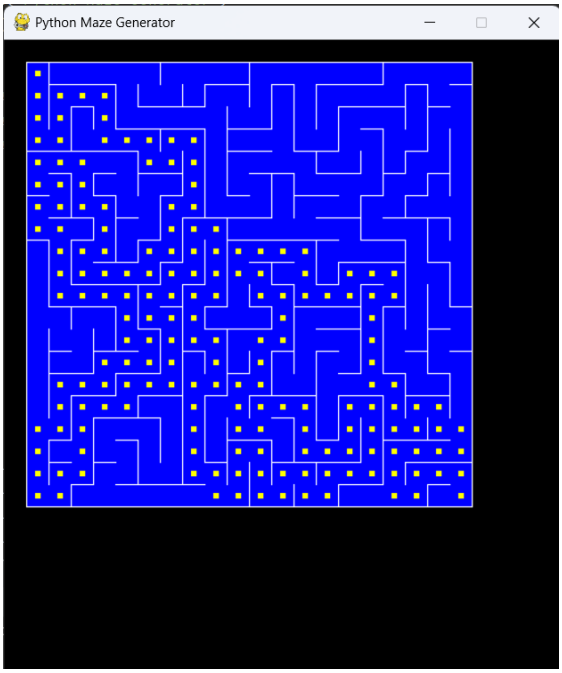
	pada baris 2. Baris 14-21: Mengecek sel mana yang telah dipilih, kemudian memanggil fungsi yang sesuai untuk memahat dinding untuk menghubungkan sel saat ini dan sel yang dipilih, menambahkan solusi pada kamus yang menunjukkan di mana dinding telah dipahat, memperbarui koordinat x dan y ke posisi sel yang dipilih, menandai sel yang dipilih sebagai sudah dikunjungi dan menambahkannya ke dalam stack.

SOURCE CODE else: x, y = stack.pop() # if no cells are available pop one from the stack single_cell(x, y) # use single_cell function to show backtracking image time.sleep(.05) # slow program down a bit backtracking_cell(x, y) # change colour to green to identify backtracking path	
Penjelasan :	Baris 1: Jika tidak ada sel yang tersedia untuk dijelajahi, maka pop satu sel dari tumpukan dan simpan koordinatnya ke dalam variabel x dan y. Baris 2: Memanggil fungsi single_cell untuk menunjukkan gambar backtracking. Baris 3: Menggunakan fungsi time.sleep() untuk memperlambat program sedikit agar mudah dipahami. Baris 4: Memanggil fungsi backtracking_cell untuk menandai jalur backtracking dengan warna hijau.

3. Hasil Output Algoritma

Setelah dilakukan perancangan pada algoritma, didapatkan hasil *output* algoritma berikut.





Pada hasil *output* berikut, algoritma melakukan pencarian jalur terlebih dahulu ke berbagai jalur yang dapat dilalui, algoritma ditandai dengan kotak warna biru. Sistem Algoritma akan memilih jalur node yang dapat dilalui. Jika node menuju jalan buntu, sistem akan melakukan *Backtracking* dimana sistem akan mundur dari node yang telah dilalui menuju node yang sebelumnya.

Algoritma akan terus menerus mencari jalur hingga semua kotak menjadi biru, sehingga membentuk jalur labirin yang sebenarnya. Selanjutnya, sistem akan Kembali menuju titik awal labirin dan membentuk petunjuk jalan yang sebenarnya yang memberikan jalan keluar ke titik akhir dari labirin.

4. Kesimpulan

Setelah dilakukan analisis terhadap pendapat ahli dan penulis serta perancangan pada Algoritma *Backtracking* dalam *Maze Problem* serta hasil, didapatkan kesimpulan berikut:

1. *Maze Problem* dikenalkan sebagai kasus yang umum terjadi pada AI, dimana sistem mencari jalan keluar dengan cepat adalah indikator penting yang dapat mengukur kualitas algortima AI tersebut.

2. Algoritma *Recursive Backtracker* didefinisikan sebagai teknik algoritma yang dapat memecahkan masalah secara rekursif dengan mencoba membangun solusi secara bertahap, satu per satu, dengan memulai dari titik acak dan mencoba bergerak ke arah yang acak.
3. Berdasarkan pengertian ahli, analisis Jeff dan analisis penulis menjelaskan definisi dan cara kerja algoritme secara lebih rinci dan spesifik, sementara Godfry menyoroti konsep dasar dan memberikan gambaran umum tentang cara kerjanya.