Aplikasi Rekursivitas pada Metode *Recursive Backtracking* untuk menemukan solusi permasalahan sederhana labirin "Rat in a Maze"

Harry Rahmadi Munly 13517033
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517033@std.stei.itb.ac.id

***Abstract*—Rat in a Maze adalah persoalan labirin yang disederhanakan dengan aturan sebagai berikut. Terdapat matriks maze N x N yang merepresentasikan sebuah labirin. labirin ini memiliki *start* dari maze[0][0] dan memiliki tujuan akhir maze[n-1][n-1]. tikus sebagai subjek dalam masalah ini untuk bergerak menuju tujuan hanya dapat bergerak 2 arah ke kanan atau kebawah kecuali menemui jalan buntu tikus akan bergerak keatas dan kekiri. algoritma akan menghasilkan matriks solusi, dengan angka 1 yang menghubungkan start ke tujuan akhir. selain dari 1 disimbolkan dengan 0.[6]**

Labirin sendiri merupakan salah satu dari permainan teka-teki yang dapat kita temui dengan berbagai tingkat kesulitan. Labirin paling sederhana dapat kita temui di buku mewarnai anak-anak ataupun pada *mirror house* di taman hiburan. labirin juga dapat ditemukan dalam video game dan menjadi salah satu cabang lomba yaitu robotic maze. Ada banyak algoritma dan metode untuk memecahkan teka teki labirin diantaranya Trémaux's Algorithm, Depth-First Search, Wall follower, Pledge Algorithm, Random Mouse, dan banyak algoritma maupun metode lainnya. Namun dalam makalah ini akan dibahas metode recursive backtracking yang memanfatkan rekursivitas pada aplikasinya.

***Keywords*—recursive backtracking, labirin, rekurisivitas**

I. INTRODUCTION

Labirin pertama yang tercatat oleh sejarah, ditemukan dan didokumentasikan oleh Herodotus, seorang sejarawan Yunani yang mengunjungi labirin bangsa mesir (*Egyptian*) pada abad ke-5 SM. labirin ini berlokasi diatas danau Moeris dan berseberangan dengan kota crocodilopolis.[3]

Gambar 1.1 labirin pertama yang tercatat oleh sejarah

(Sumber : http://www.unmuseum.org/labegypt.jpg diakses tanggal 8 Desember 2018 pukul 3:33 PM WIB GMT+7)

Dalam perkembangannya, labirin dimanfaatkan sebagai seni seperti yaitu lukisan labirin 2 dimensi yang menjadi ciri khas dari gereja ketika penyebaran agama Kristen di eropa dan bagian dari taman yang dibangun di istana versailes oleh raja Louis XIV pada abad ke 17 M.

Di zaman modern seperti sekarang ini, selain sebagai seni ataupun hiburan labirin juga dimanfaatkan dalam bidang sains dan teknologi. labirin digunakan di laboratorium untuk meneliti kemampuan mengingat dan belajar suatu subjek penelitian. misalnya tikus yang memasuki labirin untuk mencari hadiah makanan yang diasumsikan sebagai garis finish dari labirin tersebut. Awalnya tikus akan memakan waktu yang lama karena secara acak menelurusuri labirin dan berulang kali melewati jalur yang sama. Namun, setelan berulang kali percobaaan, tikus yang sama akan memakan waktu yang semakin sedikit karena tikus tersebut mempelajari dan mengingat bentuk dari labirin tersebut secara alamiah. karena fakta ini ilmuwan menjadi yakin jika tikus merupakan subjek penelitian yang memiliki kemampuan mengingat dan belajar yang relatif tinggi dibanding subjek lainnya.

Perilaku tikus pada percobaan pertama dalam memecahkan teka-teki labirin dapat disimulasikan dalam algoritma Random Mouse[7]. Namun algoritma random mouse bukan merupakan solusi yang efisien waktu dan solusi yang naif. dan ada banyak algoritma yang lebih efisien atau mangkus salah satunya mdengan metode recursive backtreacking.

Rat in a maze mensimulasikan perilaku tikus laboratorium ideal yang lebih efektif dan efisien dalam memecahkan teka-teki labirin ini. tikus laboratorium pada Rat in maze dikatakan ideal karena memilki Batasan. hanya dapat bergerak ke kanan dan kebawah labirin relative dilihat dari atas dan percobaan tikus menyelesaikan labirin selalu dimulai dari pojok kiri atas dan selalu memiliki tujuan akhir pojok kanan bawah relatif labirin ketika dilihat dari atas dengan menggunakan metode recursive backtracking.

II. DASAR TEORI

A. Rekursif

Suatu fungsi diakatakan rekursif ketika fungsi tersebut memanggil dirinya sendiri. Proses rekursif disebut sebagai rekursi (*recursion).* Rekursif tidak hanya ditemukan pada fungsi ataupun algritma saja. tetapi rekursif juga dapat kita temui dalam kehidupan sehari-hari.contohnya seperti ketika kita berada didalam ruangan dengan 2 kaca yang saling berhadapan maka akan terlihat foto yang diambil seperti berikut

Gambar 2.1 contoh rekursivitas dalam kehidupan sehari-hari (Sumber : http://steve-patterson.com/wp-content/uploads/2015/09/infinitemirror.jpg diakses tanggal 10 Desember 2018 pukul 08:12 PM WIB GMT+7)

selain itu membuat pohon juga bisa dilakukan secara rekursif seperti gambar berikut

Gambar 2.2. proses pembuatan pohon secara rekursi (Sumber : https://introcs.cs.princeton.edu/java/23recursion/images/recursive-trees.png diakses tanggal 10 Desember 2018 pukul 08:17 PM WIB GMT+7)

Setiap fungsi rekursif memiliki dua bagian yaitu :

1) Basis
Basis adalah bagian dari fungsi rekursif yang didefinisikan secara eksplisit (tidak memanggil dirinya sendiri). Bagian ini berfungsi untuk fungsi rekursif tidak menjadi *infinite loop* dan memberikan nilai yang terdefinisi.
2) Rekurens
Rekurens adalah bagian yang memanggil fungsi rekursif kembali. Setiap rekurens berjalan menuju basis yang dituju.

Salah satu kasus sederhana yang dapat diselesaikan dengan rekursif adalah menghitung jumlah jabat tangan yang terjadi. misalkan ada n orang di suatu acara setiap dari orang tersebut harus berjabat tangan berapa jabat tangan yang terjadi?

Gambar 2.3 jabat tangan (Sumber http://www.nu.or.id/o-client/nu_or_id/pictures/post/big/15298544215b2fb9d5842f6.jpg diakses tanggal 10 Desember 2018 pukul 08:20 PM WIB GMT+7)

Untuk menyelesaikan dengan rekursif tentukan basis dan rekursi untuk kasus ini. berikut proses rekonstruksi kasus jabat tangan dari kasus jabat tangan yang paling sederhana.

1) mulai dari definisi jabat tangan, jabat tangan akan terjadi ketika ada 2 orang (2 orang 1 jabat tangan).
2) selanjutnya, ketika orang ketiga datang dia akan bersalaman dengan 2 orang tadi sehingga (3 orang 1+2 jabat tangan).
3) selanjutnya ketika orang keempat datang datang maka dia akan menjabat tangan 3 orang diawal (4 orang 1+2+3 jabat tangan).
4) begitu selanjutnya untuk orang ke-5 dan seterusnya.

jika n menyatakan jumlah orang dan f(n) menyatakan jumlah jabat tangan maka,

Basis : jika n=2 maka f(n) = 1
Rekursi : jika n>2 maka f(n) = n-1 + f(n-1)

dengan fungsi rekursif diatas dapat kita uji dengan n=3 maka didapat f(n)= 6 =1+2+3. sama seperti yang telah direkonstruksi di awal.

B. Metode *Backtracking*

Backtracking adalah metode dalam algoritma yang menyoba berbagai kemungkinan jawaban sampai jawaban tersebut memenuhi semua syarat yang dibutuhkan. namun dengan mengeliminasi semua kemungkinan jawaban yang tidak memenuhi syarat untuk mencapai solusi akhir. Setiap kemungkinan jawaban hanya diperiksa satu kali[5] berdasarkan pengertian tersebut metode backtracking memiliki kemiripan dengan metode yang paling naif yaitu *brute force* namun *backtracking* adalah versi perbaikan dari metode *brute force*.

C. Kompleksitas Algoritma

Algoritma merupakan sekumpulan urutan intruksi untuk melakukan suatu perhitungan atau menyelesaikan persoalan.[2] Algoritma yang efektif/mangkus adalah algoritma yang meminimumkan kebutuhan waktu dan ruang.[1]

Kebutuhan waktu dan ruang suatu algoritma bergantung pada ukuran masukan, atau jumlah data yang diproses. Ukuran masukan disimbolkan n. Misalnya, untuk mengurutkan 1000 buah elemen larik, maka n adalah 1000; menghitung 6! maka n = 6; dan lain-lain.

Terdapat dua macam kompleksitas Algoritma,

1) Kompleksitas waktu T(n), adalah jumlah operasi yang dilakukan suatu algoritma.
2) Kompleksitas ruang, S(n), adalah ruang memori yang dibutuhkan algoritma ketika mengeksekusi algoritma tersebut.

Kompleksitas waktu sendiri dibedakan 3 macam, yaitu :
- Tmax(n) : kompleksitas waktu untuk kasus terburuk.
- Tmin(n) : kompleksitas waktu untuk kasus terbaik.
- Tavg(n) : kompleksitas waktui untuk kasus rata-rata.

D. Kompleksitas Waktu Asimtotik

Dalam penerapannya kita tidak perlu mengetahui kompleksitas waktu secara presisi, tapi kita cukup tahu gambaran kasar kebutuhan waktu dari suatu algoritma, dan seberapa cepat fungsi tersebut tumbuh yang dapat dilihat dari grafik kompleksitas waktu (T(n)) yang dihasilkan .

Perbedaan kinerja suatu algoritma baru terlihat saat masukan dari program atau nilai n sangat besar. Dikarenakan pada zaman modern sekarang ini kecepatan komputer dalam mengeksekusi instruksi sudah mencapai kecepatan yang sangat cepat.

Untuk komputer yang sudah sangat jarang diapakai saja yaitu komputer dengan prosesor generasi Pentium memiliki kecepatan eksekusi instruksi mencapai 100 MIPS[3] atau 10^8 IPS(Instruction Per Second). walaupun nilai ini tidak eksak dan berbeda-beda untuk setiap asitektur komputer. jika diasumsikan suatu komputer memiliki kecepatan 10^8 IPS maka perbedaan lama eksekusi program dengan kompleksitas algoritma T(n) = n baru akan terasa ketika n bernilai 10^{10} yaitu sekitar 100 detik atau 1,67 menit. nilai 10^{10} merupakan nilai yang sangat besar. Karena itu kinerja suatu algoritma baru akan terlihat ketika nilai n sangat besar yang dalam contoh diatas ($n>10^{10}$).

kasus berikutnya tanpa memperdulikan kecepatan eksekusi komputer dan dengan hanya memperhatikan nilai dari kompleksitas waktu akan dibandingkan 2 buah kompleksitas waktu dalam table berikut.

$T1(n) = 2n^2 + 3n + 2$

$T2(n) = n^2$

n	T1(n)	T2(n)
1	7	1
10	232	100
100	20.032	10.000
1000	2.003.000	1.000.000

Tabel 2.1 Perbandingan Kompleksitas Waktu

Dalam contoh kasus diatas dapat dilihat, nilai fungsi $T1(n)=2n^2+3n+2$ tidak terlalu jauh hasilnya dengan $T2(n)=n^2$. karena itu dapat ditarik kesimpulan Fungsi-fungsi dengan kompleksitas waktu yang relatif mirip, memiliki orde yang sama. Pada contoh ini, kedua fungsi tersebut berorder O(n2). Kita menggunakan notasi *big-O* untuk menghitung secara asimptotik laju pertumbuhan waktu eksekusi algoritma dengan batas atas (*upper-bound*).

Pengelompokkan algoritma berdasarkan notasi-O besar (diurutkan dari yang tercepat sampai yang terlambat):

Notasi-O	**Nama**
O(1)	*Constant*
O(log n)	*Logarithmic*
O(n)	*Linear*
O(n log n)	*Log Linear*
$O(n^2)$	*Quadratic*
$O(n^3)$	*Cubic*
$O(2^n)$	*Exponential*

Tabel 2.2. Pengelompokan dan pengurutan Notasi-O berdasarkan kecepatan eksekusinya

E. Kombinatorika

1) Kaidah dasar menghitung

Di dalam kombinatorial, kita barus mengbitung (counting) semua kemungkinan pengaturan objek. Dua kaidah dasar yang digunakan sebagai teknik menghitung. Dalam kombinatrorial adalah kaidah perkalian (rnle of product) dan kaidah penjumlahan (rule of sum). Kedua kaidah ini dapat digunakan untuk memecahkan banyak masalah persoalan menghitung.[1]

1) Kaidah perkalian(rule of product)
   Percobaan1:p hasil
   Percobaan2:q hasil
   Percobaan1 dan percobaan2:p x q hasil

2) Kaidah penjumlahan(rule of sum)
   Percobaan1:p hasil
   Percobaan2:q hasil
   Percobaan 1 atau percobaan 2 :p + q hasil

2) Permutasi

Sebuah permutasi adalah penyusunan objek-objek yang ada pada suatu himpunan. Urutuan penyusunan r buah elemen dari sebuah himpunan dengan n buah elemen dinamakan permutasi-r, dilambangkan dengan P(n,r).[2]
Contoh :
Misalkan S = {1,2,3}. Susunan 3,1,2 adalah permutasi dari S. Sedangkan susunan 3,2 adalah permutasi-2 dari S.

Jika n dan r adalah bilangan bulat, dan 0 <= r <= n. Maka :

$$P(n, r) = n! / (n-r)!$$

3) Kombinasi

Kombinasi adalah bentuk khusus dari permutasi. Pada kombinasi, urutan kemunculan dari elemen diabaikan. Maka, sederhananya kombinasi-r dari elemen-elemen sebuah himpunan adalah *subset* dari himpunan dengan elemen sebanyak r.[2]

Banyaknya kombinasi-r sebuah himpunan dengan n buah elemen, dengan n adalah sebuah bilangan bulat tidak negatif dan r adalah bilangan bulat, dan 0 <= r <= n, adalah :

$$C(n,r) = n!r! (n-r)!$$

III. PEMBAHASAN

A. Algoritma Penyelesaian

Rat in a Maze adalah persoalan labirin yang disederhanakan memiliki aturan sebagai berikut :

1) Terdapat matriks maze N x N yang merepresentasikan sebuah labirin. Elemen matriks maze berupa angka 1 dan 0. angka 1 menandakan jalan yang bisa dilewati. sedangkan angka 0 merupakan dinding.
2) Labirin ini memiliki *start* dari maze[0][0] dan memiliki tujuan akhir maze[n-1][n-1].
3) tikus sebagai subjek dalam masalah ini hanya dapat bergerak 2 arah ke kanan atau kebawah.
4) output dari algoritma adalah matriks sol yang memiliki elemen angka 1 dan 0. angka 1 jalan yang dapat dilewati dari titik awal ke tujuan akhir, elemen matriks sol selain 1 disimbolkan dengan angka 0.

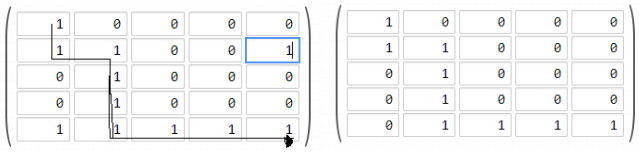
Algoritma *backtracking* secara umum sebagai fungsi rekursif memiliki komponen sebagai berikut :

- Basis : jika koordinat tikus berada adalah tujuan akhir, cetak matriks solusi
- Rekurens : 1) tandai kordinat tikus dengan 1 pada matriks solusi.
  2) jika dikanan koordinat tikus valid untuk dilewati (bukan dinding dan luar maze), bergerak ke kanan dan cek basis
  3) jika dibawah koordinat tikus valid untuk dilewati, bergerak ke bawah dan cek basis
  4) jika dikanan dan dibawah koordinat adalah dinding atau jalan buntu maka lakukan backtrack

```
  #include<stdio.h>

  /* ukuran kolom atau baris dari matriks maze asumsi saat
ini N = 4 */
  #define N 4

  bool solveMazeRek(int maze[N][N], int x, int y, int
solusi[N][N]);

  /* fungsi untuk mencetak matriks solusi */
  void printSolusi(int solusi[N][N])
  {
      for (int i = 0; i < N; i++)
      {
          for (int j = 0; j < N; j++)
              printf(" %d ", solusi[i][j])
          printf("¥n");
      }
  }

  /* fungsi untuk mengecek apakah x,y masih berada di dalam
labirin */
  bool isSafe(int maze[N][N], int x, int y)
  {
      /* jika (x,y) diluar labirin mengembalikan false*/
      if(x >= 0 && x < N && y >= 0 && y < N && maze[x][y] ==
1)
          return true;
      else
      return false;
  }

  /*untuk mencetak matriks solusi ini cukup memanggil fungsi
ini (fungsi main). ada beberapa kemungkinan jalan untuk sebuah
permasalahan tetapi fungsi ini hanya mencetak 1
  .*/
  bool solveMaze(int maze[N][N])
  {
      int solusi[N][N] = { {0, 0, 0, 0},
          {0, 0, 0, 0},
          {0, 0, 0, 0},
          {0, 0, 0, 0}
      };

      if(solveMazeUtil(maze, 0, 0, solusi) == false)
      {
          printf("solusi tidak ada");
          return false;
      }

      printSolusi(solusi);
      return true;
  }

  /* fungsi rekursif yang menggunakan metode recursive
backtracking */
  bool solveMazeRek(int maze[N][N], int x, int y, int
solusi[N][N])
  {
      /* jika (x,y adalah tujuan akhir) return true */
      if(x == N-1 && y == N-1)
      {
          solusi[x][y] = 1;
          return true;
      }

      // mengecek apakah maze[x][y] valid
      if(isSafe(maze, x, y) == true)
      {
          // tandai x,y sebagai bagian dari solusi
          solusi[x][y] = 1;

          /* bergerak maju arah x */
          if (solveMazeUtil(maze, x+1, y, solusi) == true)
              return true;

          /* apabila bergerak maju arah x tidak menghasilkan
          solusi
                bergerak kebawah di arah y */
          if (solveMazeUtil(maze, x, y+1, solusi) == true)
              return true;

          /* apabila dikanan dan dibawah tikus adalah dinding
         atau jalan buntu lakukan BACKTRACKING dan
              coret x,y sebagai solusi */
          solusi[x][y] = 0;
          return false;
      }

      return false;
  }

  // program driver untuk menguji algoritma
  int main()
  {
      int maze[N][N]  =  { {1, 0, 0, 0},
```

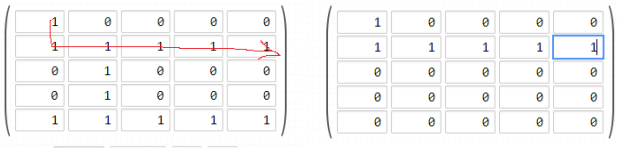
```
        {1, 1, 0, 1},
        {0, 1, 0, 0},
        {1, 1, 1, 1}
    };

    solveMaze(maze);
    return 0;
}
```

berikut merupakan ilustrasi dari matriks maze masukan di bagian driver pada algoritma.



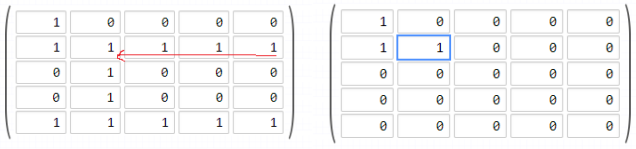
Gambar 3.1 : (kiri) jalan eksekusi program (kanan) matriks solusi akhir yang dihasilkan

Permasalahan dengan matriks diatas masih sederhana, berikut uji kasus yang sedikit lebih rumit.



Gambar 3.2 : (kiri) jalan eksekusi program, (kanan) matriks solusi sementara yang dihasilkan

Ketika program memasuki tahap seperti gambar 3.2 proses backtracking dilakukan.



Gambar 3.3 : proses backtracking

Proses backtracking dilakukan, dan berhenti ketika tikus dapat bergerak kebawah karena bawah dari koordinat valid untuk dilewati, program memasuki rekurens (3) “jika dibawah koordinat tikus valid untuk dilewati, bergerak ke bawah dan cek basis”.

langkah selanjutnya diilustrasikan pada gambar 3.1

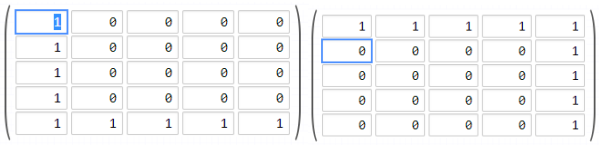
B. Kompleksitas Waktu antara *Brute Force* dan *Recursive Bactracking*

Seperti yang dibahas di bagian pendahuluan permasalah *Rat in a Maze* dapat di selesaikan dengan algoritma random mouse[7]. menyempurnakan random mouse yang bisa melakukan perulangan pada jalan yag telah dilewati dan diragukan bisa menyelesaikan labirin, diperkenalkan Algoritma *brute force*.

Algoritma brute force singkatnya menggenerate semua kandidat matriks dengan elemen 1 dan 0 yang merupakan matriks solusi dari permasalahan rat in a maze ini.

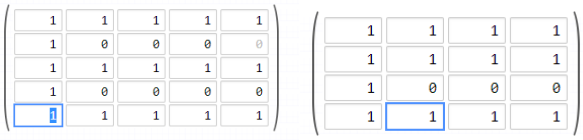
Dengan mengunakan kaidah perkalian pada kombinatorika diperoleh kompleksitas waktu *brute force* Tmax(N) = 2^{N2}! dan Tmin(N) =2^{N2}. dengan N adalah ukuran kolom matriks maze yang merupakan matriks persegi N x N. sehingga *brute force* dapat dinotasikan dalam Big-O O(N) = 2^{N2}!. fungsi eksponesial faktorial. fungsi ini sangatlah tidak mangkus karena memakan waktu yang sangat lama.

Sedangkan algoritma recursive backtracking, kompleksitas waktu Tmin(N) = 2N – 1, dengan syarat start selalu di matriks ujung kiri atas dan tujuan akhir selalu berada di pojok kanan bawah matriks. berikut adalah contoh matriks yang dapat diselesaikan algoritma recursive backtracking dengan kompleksitas waktu Tmin



gambar 3.2 contoh matriks maze yang menghasilkan kompleksitas waktu Tmin (N = 5)

kompleksitas waktu terburuk algoritma recursive backtracking dapat diilustrasikan dengan gambar berikut,



gambar 3.3 contoh matriks maze yang menghasilkan kompleksitas waktu terburuk (N ganjil di kiri dan N genap di kanan)

Matriks maze diatas menghasilkan kompleksitas waktu terburuk dikarenakan proses backtracking. Nilai kompleksitas waktu terburuknya adalah, Tmax(N) = N*(N/2+1)+N atau jika dinyatakan dengan notasi Big O, O(N) = N^2.

C. Kelebihan dan kekurangan Algoritma recursive Backtracking

Algoritma recursive backtracking pada makalah ini dapat dikembangkan untuk persoalan labirin yang lebih umum. Namun tentu saja dibutuhkan konsep atau metode baru yaitu stack.

Selain recursive backtracking masih banyak lagi algoritma lain yang dapat dipakai untuk menyelesaikan permasalahan labirin secara umum seperti algoritma Tremaux, algoritma pledge, wall follower, dan algoritma chain. diantara algoritma tersebut algoritma recursive backtracking dapat diterapkan tanpa harus tahu lintasan labirin dari atas dan dapat digunakan oleh manusia.

Kekurangan recursive backtracking adalah hanya bisa menghasilkan satu solusi lintasan. kekurangan recursive backtracking pada makalah ini merupakan solusi untuk

permasalahan yang masih terbatas yaitu “Rat in a Maze”.

V. KESIMPULAN

Dengan mengaplikasikan rekursivitas pada algoritma yang naif, *brute force*, dan menggabungkan dengan metode *backtracking* kita dapat menghasilkan algoritma yang memiliki yang jauh lebih efektif dari pada *brute force*. Dibandingkan dengan algoritma brute force yang memiliki kompleksitas $O(N) = 2^{N2}!$ yang memakan waktu eksekusi relative sangat lama dan bukan algoritma yang mangkus. algoritma backtracking rekursif memilki kompleksitas waktu $O(N) = N^2$ dan termasuk dalam fungsi kuadrat. sehingga dapat disimpulkan algoritma recursive backtracking merupakan algoritma yang mangkus untuk permasalahan “Rat in a Maze”.

Persoalan pada labirin tidak hanya terbatas pada “Rat in a Maze”. Rat in a maze adalah aturan-aturan tertentu yang membatasi persoalan labirin menjadi relative lebih sederhana. Algoritma recursive backtracking pada makalah ini dirancang untuk menyelesaikan masalah “Rat in a Maze”. sehingga algoritma ini masih bisa dikembangkan untuk bisa menyelesaikan masalah umum labirin dengan menambahkan konsep stack pada algoritmanya.

VII. UCAPAN TERIMA KASIH

Pertama-tama, penulis mengucapkan syukur kepada Tuhan Yang Maha Esa atas segala nikmat yang telah diberikan sehingga penulis dapat menyelesaikan makalah ini. Penulis juga mengucapkan terima kasih kepada dosen mata kuliah Matematika Diskrit, bapak Rinaldi Munir, ibu Harlili, dan khususnya kepada bapak Judhi Santoso yang telah mengajar kelas K03 selama satu semester ini. Penulis juga ingin mengucapkan terima kasih kepada keluarga penulis yang membuat motivasi penulis selalu terjaga. Terakhir, penulis ingin mengucapkan terima kasih kepada seluruh pihak yang telah berkontribusi dalam pengerjaan makalah ini, baik secara langsung maupun tidak langsung.

REFERENCES

[1] Munir, Rinaldi. *Matematika Diskrit*, Bandung: Informatika, 2012, edisi kelima

[2] Rosen, K.H., *Discrete Mathematics and Its Application*, New York: McGraw-Hill, 2012, edisi ketujuh.

[3] https://www.webopedia.com/TERM/M/MIPS.html diakses tanggal 9 Desember 2018 pukul 4:49 PM WIB GMT+7

[4] http://www.unmuseum.org/maze.htm diakses tanggal 8 Desember 2018 pukul 1:53 PM WIB GMT+7

[5] https://www.geeksforgeeks.org/the-knights-tour-problem-backtracking-1/ diakses tanggal 8 Desember 2018 pukul 10:20 AM WIB GMT+7

[6] https://www.geeksforgeeks.org/rat-in-a-maze-backtracking-2/ diakses tanggal 8 Desember 2018 pukul 08:11 AM WIB GMT+7

[7] http://www.astrolog.org/labyrnth/algrithm.htm tanggal 8 Desember 2018 pukul 2:23 PM WIB GMT+7

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, 3 Desember 2017

Harry Rahmadi Munly
13517033