Aplikasi Pohon dan Bactracking pada Robot Micromouse

Dimas Faidh Muzaki - 13520156[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13520156@std.stei.itb.ac.id

***Abstract*—Micromouse merupakan sebuah kontes robot kecil cerdas yang mampu memecahkan sebuah labirin berukuran 16x16. Ada banyak macam pencapaian yang dapat digunakan oleh robot untuk menyelesaikan tugasnya. Salah satunya adalah dengan algoritma backtracking dan konsep pohon. Dengan menggunakan backtracking, robot menjelajahi labirin sambil membuat struktur data pohon. Saat selesai pohon tersebut dapat ditinjau semua litasan yang menuju ke tujuan untuk didapatkan jalan tercepat.**

***Keywords*—Robot kecil, *micromouse*, labirin, backtracking, pohon.**

I. PENDAHULUAN

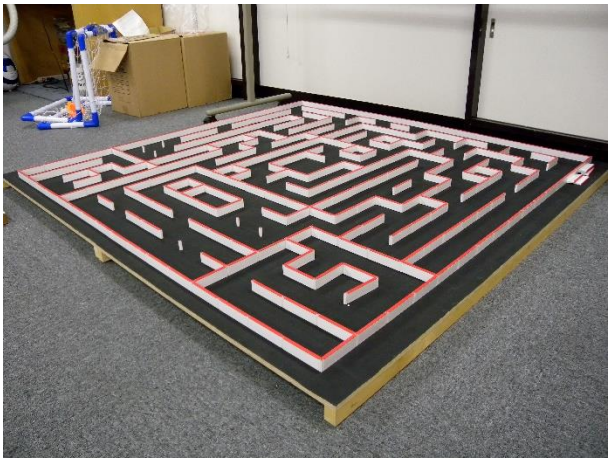


Fig. 1 Contoh labirin micromouse

Micromouse merupakan sebuah jenis perlombaan robot kecil yang populer sejak tahun 1970an. Inti dari perlombaan ini adalah mencipatakan sebah robot kecil dengan panjang dan lebar maksimum 25 x 25 cm yang dapat menelusuri labirin dan menemukan lintasan ke lokasi spesifik pada labirin. Robot tersebut harus *fully autonomous* atau bergerak dengan sendirinya saat perlombaan tanpa dikontrol oleh peserta lomba.

Arena lomba yang digunakan adalah sebuah labirin berukuran 16 x 16. Tiap petak labirin berukuran 18 x 18 cm yang dibatasi dengan tembok setinggi 5 cm. Tembok-tembok tersebut terbuat dari kayu dengan ketebalan 1.2 cm dan (pada umumnya) dicat warna putih. Lantai dari labirin berwarna pada umumnya berwarna hitam.

[1]Titik awal lintasan robot adalah pada salah satu sudut pada labirin. Titik awal tersebut harus dikelilingi oleh tiga tembok. Sementara itu, titik akhir atau *goal* yang harus dicapai adalah sebuah ruangan berbentuk persegi yang terdiri dari 4 buah petak. *Goal* hanya memiliki satu pintu masuk dan umumnya berada di tengah labirin.

Robot yang ditandingkan pada umumnya terdiri dari 3 komponen utama, yaitu sensor, motor, dan microcontroller. Sensor digunakan untuk mendeteksi keberadaan tembok disekitar robot. Hal itu dapat dicapai dengan sensor berjenis infrared, proximity, ataupun ultrasonic sensor. Motor menjadi penggerak roda untuk memungkinkan robot berpindah. Sementara microcontroller berperan sebagai otak dari robot yang mengatur pergerakan robot menggunakan algoritma yang sudah ditentukan. Dengan ketiga komponen tersebut, robot micromouse diharapkan untuk dapat melakukan beberapa pergerakan penting seperti maju, belok ke kiri dan kanan, dan berputar balik.

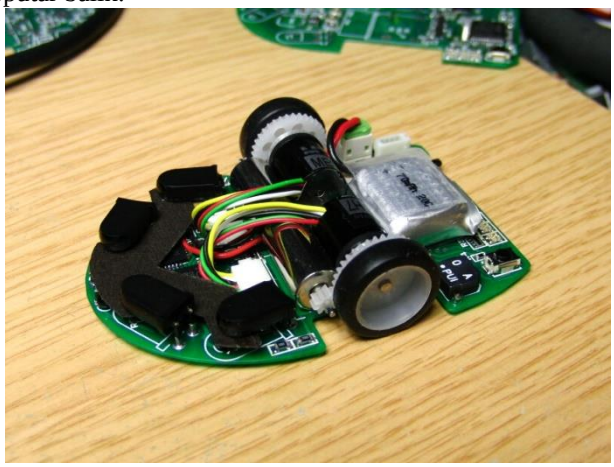


Fig. 2"Egg Torte" sebuah robot micromouse

Pemenang dari perlombaan micromouse adalah robot yang mampu mencapai *goal* dengan waktu tersingkat. Ada banyak faktor yang dapat mempengaruhi peforma robot seperti komponen dan algoritma yang digunakan. Makalah ini akan membahas contoh penggunaan konsep pohon untuk algoritma micromouse. Dalam penelusuran, robot akan bergerak dengan cara *Backtracking* dan menyimpan informasi yang dilaluinya ke dalam pohon.

II. LANDASAN TEORI

A.Graf

[2]Graf merupakan sebuah konsep yang digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara

objek-objek tersebut. Dalam hal ini, objek yang dimaksud direpresentasikan sebagai simpul *(vertex)* dan hubungan di antaranya direpresentasikan sebagai sisi *(edge).* Secara formal, graf G didefiniskan sebagai sebuah *tuple* $G = (V, E)$, yang dalam hal ini:

$V = himpunan\ tidak\ kosong\ dari\ simpul - simpul$
$= \{v1, v2, \dots, vn\}$
$E = himpunan\ sisi\ yang\ menghubungkan\ sepasang\ simpul$
$= \{e1, e2, \dots, e3\}$

Sebagai contoh, terdapat sebuah graf $G = (V, E)$ dengan:

$V = \{1, 2, 3, 4, 5, 6, 7\}$
$E = \{(1,2), (2,3), (2,4), (4,5), (4,6), (6,7)\}$

Maka, visualisasi dari graf tersebut adalah:

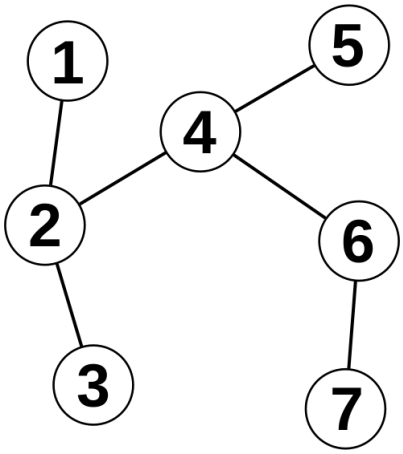


Fig. 3 Graf

B. Jenis Graf

[2]Graf dapat digolongkan menjadi beberapa jenis. Penggolongan ini dapat dilihat dari sisi yang dimiliki oleh graf. Berdasarkan keberadaan sisi gelang dan sisi ganda, graf dapat dibedakan menjadi dua jenis, yaitu

1. Graf sederhana *(simple graph)*
   Graf dapat dikategorikan sebagai graf sederhana apabila graf tersebut tidak memiliki sisi ganda ataupun sisi gelang. Sebuah graf disebut memiliki sisi ganda apabila untuk dua buah simpul pada graf terdapat lebih dari satu sisi yang menghubungkan keduanya. Sementara itu, sisi gelang adalah sebuah sisi yang memiliki simpul awal dan simpul akhir yang sama.
2. Graf tak sederhana *(unsimple graph)*
   Graf tak sederhana adalah graf yang memiliki sisi ganda, sisi gelang, ataupun keduanya.

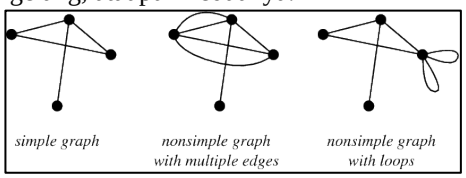


Fig. 4 Graf sederhana (kiri) dan Graf tak sederhana (tengah dan kanan)

Berdasarkan sisinya, graf dapat digolongkan menjadi dua jenis, yaitu:

1. Graf berarah *(directed graph / digraph)*
   Graf berarah memiliki arti bahwa setiap sisi yang dimiliki oleh graf mempunyai orientasi arah tertentu. Hal ini berarti jika ada sebuah jalan dari simpul satu ke simpul dua, belum tentu ada jalan dari simpul dua ke simpul satu.
2. Graf tak berarah *(undirected graph)*
   Graf tak berarah memiliki arti setiap sisi pada graf tidak memiliki arah. Sisi-sisi menghubungkan simpul bolak balik. Dalam kata lain, jika ada sebuah jalan yang menghubungkan dua simpul dapat dipakai untuk dari simpul satu ke simpul dua dan sebaliknya, dari simpul dua ke simpul satu.

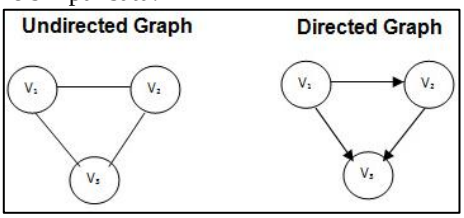


Fig. 5 Graf tak berarah (kiri) dan graf tak berarah (kanan)

C. Terminologi Graf

[2]Misal dengan menggunakan graf kita sebelumnya yaitu G dengan:

$V = \{1,2,3,4,5,6,7\}$
$E = \{(1,2), (2,3), (2,4), (4,5), (4,6), (6,7)\}.$

Kita dapat mendefinisikan beberapa terminologi pada graf, sebagai berikut:

1. Ketetanggaan *(Adjacent)*
   Dua buah simpul dikatakan *bertetangga* bila keduanya terhubung langsung. Dalam hal ini terdapat sisi yang menghubungkan keduanya. Pada graf kita, simpul 1 *bertetangga* dengan simpul 2 karena memiliki sisi (1,2)
2. Bersisian *(Incidency)*
   Untuk sembarang sisi $e = (u, v)$ dapat dikatakan bahwa *e* bersisian dengan simpul u atau *e* bersisian dengan simpul *v*. Pada graf G dapat dikatakan bahwa sisi (1,2) bersisian dengan simpul 1 dan 2.
3. Deajat *(Degree)*
   Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut. Derajat suatu simpul *v* dinotasikan dengan *d(v).* Pada graf G, derajat dari simpul 2 adalah $d(2) = 3$ karena ada sisi (1,2), (2,3), dan (2,4) yang bersisian dengan simpul 2.
4. Lintasan *(Path)*
   Lintasan yang menghubungkan simpul awal v_o ke simpul tujuan v_n merupakan sebuah barisan berselang-seling simpul-simpul dan sisi-sisi yang yang berbentuk v0 , e1 , v1 , e2 , v2 ,... , vn –1 , en , vn sedemikian sehingga e1 = (v0 , v1), e2 = (v1 , v2), ... , en = (vn-1 , vn) adalah sisi-sisi dari graf G. Panjang dari sebuah lintasan adalah n dengan n adalah jumlah sisi dalam lintasan tersebut. Pada graf G, lintasan dari simpul 1 ke simpul 5 adalah lintasan dengan panjang 3 dan sisi-sisi (1,2),(2,4), dan (4,5)
5. Sirkuit / Siklus *(Circuit / Cycle)*
   Sebuah lintasan dengan simpul awal dan simpul akhir yang sama dapat disebut sebagai siklus atau sirkuit. Pada sirkuit, tidak boleh sebuah sisi dilintasi dua kali. Graf G tidak memiliki siklus untuk setiap sisinya
6. Keterhubungan *(connected)*

Dua buah simpul *u* dan *v* dikatakan terhubung apabila terdapat lintasan dari u ke v. Suatu graf tak berarah dikatakan graf terhubung jika untuk setiap pasang simpul v_i dan v_j pada himpunan *V* terdapat lintasan dari v_i dan v_j.

D. Pohon

[2]Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit

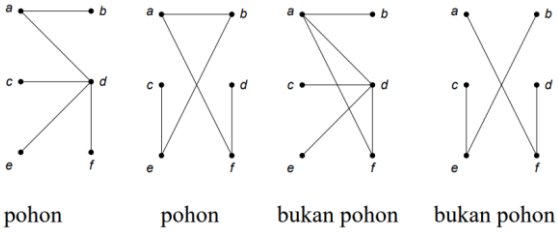


Fig. 6 Contoh pohon dan bukan pohon

Graf ketiga bukan pohon karena terdapat sirkuit a,d,f,a. Graf keempat bukan pohon karena simpul a dan simpul b tidak terhubung.

E. Pohon berakar (rooted tree)

[2]Pohon berakar adalah pohon dengan salah satu simpulnya dianggap sebagai akar dan sisi-sisinya diberi arah menjauhi akar sehingga menjadi graf berarah.

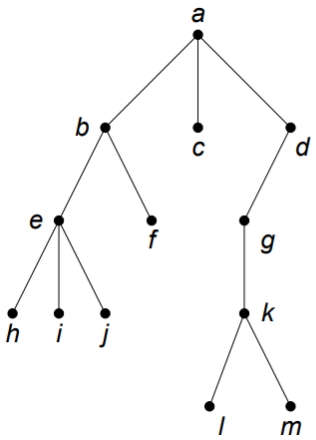


Fig. 7 Pohon Berakar

F. Terminologi Pohon Berakar

Dengan melihat pohon berakar pada *Fig. 7*, kita dapat mendefinisikan beberapa terminologi untuk pohon berakar sebagai berikut:

1. Anak dan orangtua (*child and parent)*
   b, *c*, dan *d* adalah anak-anak dari simpul a. Sementara itu, a adalah orangtua dari *b*, *c*, dan *d*.
2. Lintasan *(path)*
   Lintasan dari *a* ke *j* adalah a, b, e, j. Panjang lintasannya adalah 3.
3. Saudara kandung *(sibling)*
   f adalah saudara kandung e, tetapi g bukan saudara kandung *e*, karena orangtua mereka berbeda.
4. Upapohan *(subtree)*
   Pohon berakar yang memiliki akar yang merupakan bagian dari pohon berakar lainnya merupakan upapohon dari pohon lain tersebut. Sebagai contoh, pohon berakar b dengan simpul *V = (b, e, f, h, i, j)* adalah upapohon dari pohon berakar *Fig. 7.*
5. Derajat *(derajat)*
   Derajat sebuah simpul adalah jumlah anak yang dimiliki simpul tersebut.
6. Daun *(leaf)*
   Daun adalah simpul yang tidak memiliki anak. Simpul *h, i, j, f, c, l,* dan *m* adalah daun.
7. Simpul dalam *(internal node)*
   Simpul dalam adalah simpul yang memiliki anak. Namun, akar sebuah pohon berakar tidak termasuk simpul dalam.
8. Tingkat *(level)*
   Tingkat sebuah simpul pada pohon berakar adalah jarak dari akar ke simpul tersebut. Tingkat dimulai dari nol. Tingkat simpul g adalah 2.
9. Kedalaman *(depth)*
   Kedalaman adalah tingkat maksimum dari suatu pohon.

G. Rekursi

[3]Rekursi merupakan proses mendefinisikan sebuah masalah ataupun solusi menjadi sebuah bentuk yang mengandung dirinya sendiri. Rekursi dapat menjadi alat yang sangat berguna dalam menulis algoritma. Sebuah algoritma rekursi harus mempunyai beberapa hal:

1. Kasus Basis
2. Proses yang menuju ke basis
3. Pemanggilan rekursi

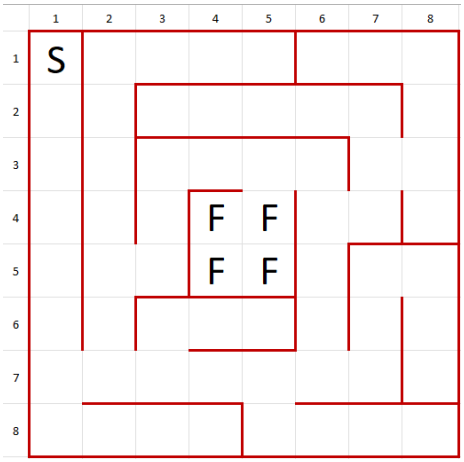
H. Backtracking

Backtracking atau telusur balik adalah salah satu teknik algoritma untuk memecahkan masalah secara recursive. Algoritma *backtracking* bersifat *brute force*, artinya *backtracking* mencari semua kombinasi solusi yang ada dan mengeleminasi solusi yang tidak mencapai tujuan sehingga hanya ada solusi yang benar.

III. PEMBAHASAN

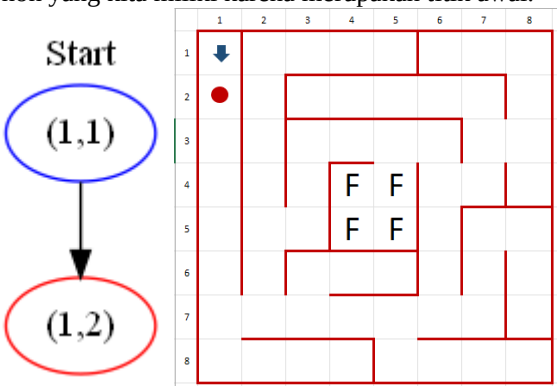
A. Ide

Algoritma backtracking dan konsep pohon dapat digunakan sama-sama oleh robot micromouse untuk memecahkan labirin. Sebelum itu, kita bahas terlebih dahulu *approach* yang kita lakukan. Perhatikan labirin berukuran 8 x 8 di bawah:

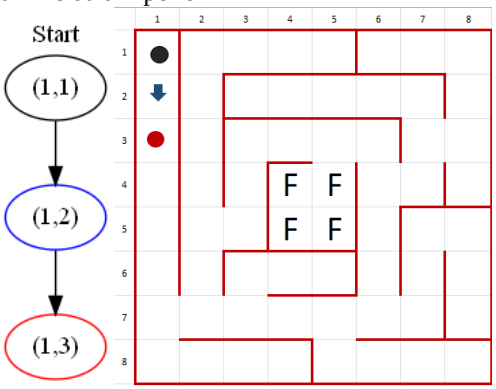


S merupakan titik awal, F merupakan 4x4 di tengah labirin yang merupakan titik-titik tujuan kita. Oleh karena itu, kita dapat menyebut bahwa titik awal kita adalah (1,1) dan titik akhir

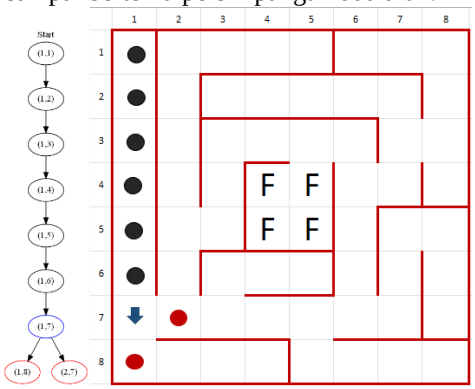
terdiri atas (4,4), (4,5), (5,4), dan (5,5). Dengan konsep pohon kita dapat menggambarkan tiap petak sebagai simpul pohon dengan koordinat sebagai id simpul. (1,1) adalah simpul akar dari pohon yang kita miliki karena merupakan titik awal.



Lingkaran biru merupakan titik letak kita berada. Dari posisi kita berada, kita tahu bahwa kita dapat bergerak ke petak (1,2), maka tambahkan petak yang bisa kita tempati ke dalam pohon yang kita miliki dan tandai dengan warna merah. Kita akan berjalan menelusuri labirin sambil meng-*update* pohon yang kita miliki. Selain itu, kita juga akan menandai petak yang sudah kita lewati dengan lingkaran hitam. Dan tidak lupa untuk setiap petak yang tempati, kita tambahkan petak selanjutnya yang bisa kita singgahi ke dalam pohon



Kita terus membuat simpul yang kita lewati sebagai anak dari pohon yang kita punya dan mengecek semua petak yang dapat disinggahi sampai bertemu persimpangan dua arah.



Dari posisi kita saat ini, kita dapat bergerak ke dua petak berbeda yaitu (1,8) dan (2,7). Dalam menangani kasus seperti ini, kita perlu menentukan prioritas saat bertemu simpangan. Kasus terburuk dari sebuah simpangan adalah pada saat ada 3 arah yang bisa kita lalui.

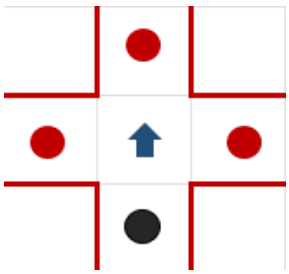
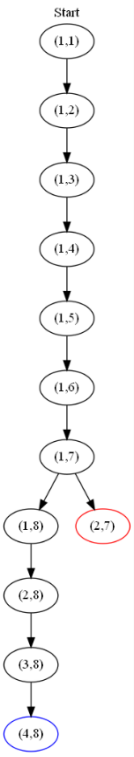


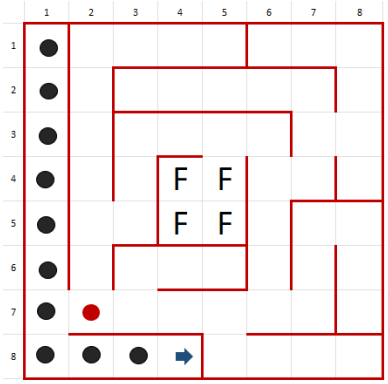
Fig. 8 Persimpangan 3 arah

Oleh karena itu, mari kita tentukan prioritas sebagai berikut:

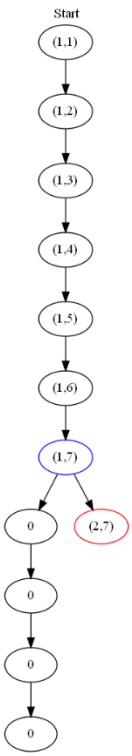
1. Jika bertemu persimpangan (satu, dua, atau tiga arah) prioritaskan untuk bergerak lurus.
2. Jika tidak bisa lurus, bergerak ke kiri.
3. Jika tidak bisa bergerak ke kiri, bergerak ke kanan

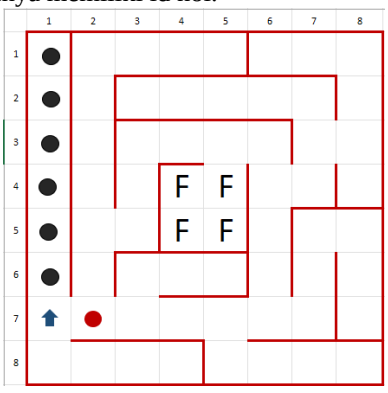
Kembali ke labirin, kita sudah memiliki prioritas untuk simpangan. Oleh karena itu mari kita lanjutkan penelusuran.





Sekarang yang terjadi adalah tidak ada sama sekali petak yang dapat disinggahi dan petak yang kita singgahi sekarang bukan titik tujuan. Ini berarti jalan yang kita lalui adalah jalan buntu dan bukan merupakan solusi. Berdasarkan konsep *backtracking*, kita akan bergerak kembali menelusuri pohon yang telah kita buat sampai menemukan simpul yang memiliki anak yang belum disinggahi (dalam ilustrasi kita adalah anak merah) dan memilih titik merah tersebut. Saat menelusur balik, kita akan menandakan petak/simpul pada pohon dengan angka nol untuk mengisyaratkan bahwa jalan tersebut bukan solusi sehingga nantinya upapohon tersebut akan seluruhnya memiliki id nol.

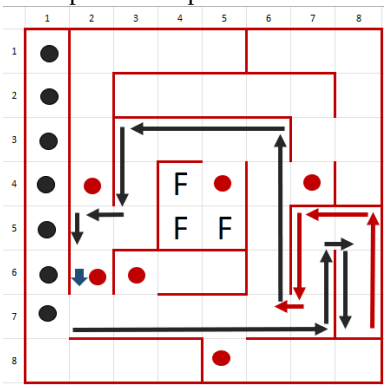




Sejauh ini kita memiliki aturan pengerjaan:

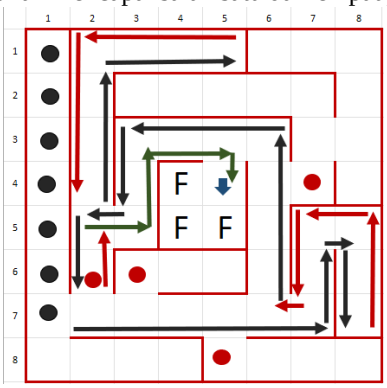
1. Mulai pohon dengan (1,1) sebagai akar
2. Bergerak ke simpul selanjutnya sesuai prioritas.
3. Tambahkan petak yang dapat disinggahi ke dalam pohon sebagai anak dari simpul saat ini.
4. Jika tidak ada petak yang dapat disinggahi, telusur balik pohon yang sudah dibuat sampai bertemu simpul yang memiliki anak merah dan matikan upapohon buntu.
5. Jika petak sekarang bukan tujuan akhir, kembali ke aturan nomor 2.

Dengan menggunakan aturan di atas, kita akan menelusuri labirin sampai mencapai *state* seperti berikut:



Dapat kita amati bahwa jika melanjutkan penelusuran berdasarkan aturan sebelumnya, kita akan kembali ke petak yang sudah pernah dilalui. Setelah itu pergerakan kita hanya akan berputar putar di lintasan yang sama. Untuk menghindari hal tersebut kita perlu menambahkan aturan lain yaitu tidak menambahkan simpul baru ke pohon jika petak sudah dilalui sebelumnya. Dengan struktur data pohon yang kita miliki, hal ini dapat kita capai dengan melihat apakah simpul yang ingin kita tambahkan merupakan *ancestor* dari simpul yang kita singgahi sekarang.

Tidak ada petak yang kita singgahi. Maka dari itu kita akan kembali seperti saat menemukan jalan buntu. Penelusuran kita lanjutkan kita akan mencapai salah satu dari empat petak tujuan:



Author names and affiliations are to be centered beneath the title and printed in Times 11-point, non-boldface type. Multiple authors may be shown in a two- or three-column format, with their affiliations below their respective names. Affiliations are centered below each author name, italicized, not bold. Include e-mail addresses if possible. Follow the author information by three blank lines before main text.

The second and following pages should begin 2 cm from the top edge. On all pages, the bottom margin should be 2.7 cm from the bottom edge of the page for A4 paper.

Kita telah menempati salah satu dari empat petak tujuan. Jika dilihat dari struktur data pohon yang kita miliki, maka solusi dari labirin adalah lintasan dari akar pohon ke daun yang merupakan petak tujuan. Namun bila dilihat, masih banyak titik titik merah yang menandakan masih ada lintasan lintasan lain yang belum pernah dicoba. Maka dari itu, Langkah yang dapat kita lakukan selanjutnya adalah menandai daun yang merupakan petak tujuan. Bentuk tanda bisa berbagai macam, salah satunya adalah membuat simpul anak bernama “goal”. Setelah itu kita kembali melakukan telusur balik untuk mencoba semua kemungkinan lintasan. Tanda dari telah dicobanya semua lintasan yang mungkin adalah kita menemukan simpul akar pada saat telusur balik. Berikut adalah pohon yang kita miliki pada saat selesai melakukan *backtracking.*

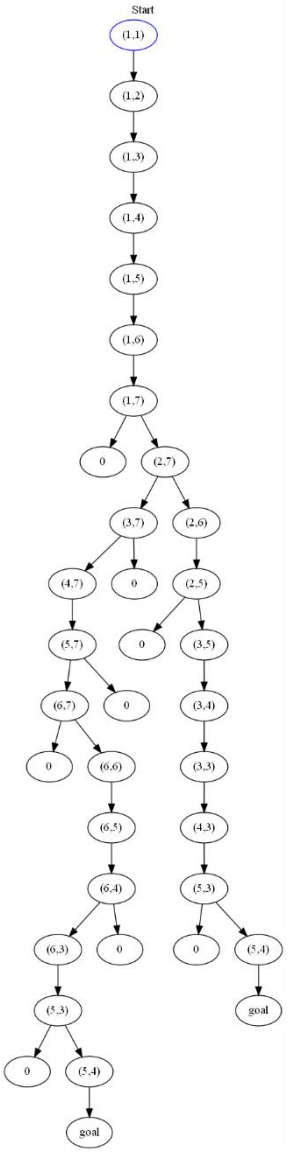


Fig. 9 Pohon Final

Seperti pada *Fig. 9,* terdapat dua buah simpul goal pada pohon. Hal tersebut memiliki makna bahwa terdapat dua solusi lintasan untuk labirin tersebut. Kita dapat mencari tahu panjang tiap lintasan untuk mendapatkan lintasan dengan jarak terpendek, untuk kemudian kita pakai.

B. Implementasi pada Robot

Kita dapat menerapkan ide yang sebelumnya kita bahas ke dalam algoritma yang digunakan oleh robot *micromouse.* Namun, ada beberapa hal yang perlu diperhatikan.

Robot tidak mengenal koordinat kartesius. Yang diketahui robot hanya keberadaan tembok di sisi kiri, kanan. Selain itu, robot hanya bisa bergerak maju, berputar ke kiri, berputar ke kanan, dan berputar balik. Oleh sebab itu, kita perlu menambahkan beberapa informasi dan aturan untuk dipakai oleh robot.

Salah satu cara mengatasinya adalah dengan mencatat informasi orientasi robot. Dengan mengetahui orientasi robot, maka robot bisa tahu posisi setelah robot bergerak maju dengan benar berdasarkan posisi tertentu. Kita dapat mendefinisikan orientasi utara, selatan, barat, timur. Namun, orientasi real yang ada, melainkan merupakan orientasi semu. Orientasi utara mengarah ke arah yang dihadap oleh robot saat memulai. Jika robot berputar ke kiri maka orientasi diubah menjadi timur. Begitu juga untuk barat dan selatan. Dengan begitu, kita dapat menetapkan bahwa jika orientasi robot adalah utara, maka saat robot bergerak maju posisi yang tadinya (x,y) menjadi (x+1,y), jika orientasi robot adalah selatan dan robot bergerak maju maka posisi yang

tadinya (x,y) menjadi (x-1,y). Untuk timur dan barat, kita harus memberikan perhatian lebih karena dipengaruhi oleh posisi mulai robot. Kita tidak nilai x dan y kita bernilai negatif. Untuk itu maka perlu kita kondisikan berdasarkan simpangan pertama yang ditemukan oleh robot. Jika simpangan pertama terletak di kiri, maka untuk orientasi timur menjadi (x,y+1) dan barat menjadi (x,y-1). Sedangkan jika simpangan pertama terletak di kanan, maka untuk orientasi timur menjadi (x,y-1) dan barat menjadi (x,y+1). Dengan begitu robot akan dapat menentukan pergerakan yang harus dilakukannya berdasarkan orientasi, letak petak sekarang, dan petak yang akan dituju.

C. Improvisasi

Dengan menggunakan algoritma *backtracking*, kita mencoba semua kemungkinan lintasan yang ada, bahkan lintasan buntu berkali kali. Untuk menghemat waktu, kita bisa menyimpan informasi petak yang menuju ke ujung buntu agar hanya cukup dilewati sekali.

Gerakan yang harus dilakukan oleh robot dapat disimpan ke dalam sebuah stack. Dengan begitu, robot tidak perlu melakukan komputasi pada saat melakukan telusur balik dan dalam mengeksekusi lintasan paling pendek. Dengan stack, robot dapat mengimplementasikan beberapa trik untuk mempercepat pergerakan. Sebagai contoh jika pada stack terdapat beberapa jalan lurus berurutan, maka robot dapat meningkatkan kecepetan untuk melewatinya. Contoh lain, apabila robot menemukan lintasan zigzag dan ukuran robot cukup kecil, robot dapat melaluinya hanya dengan berputar ke derajat yang benar dan bergerak lurus dengan cepat.

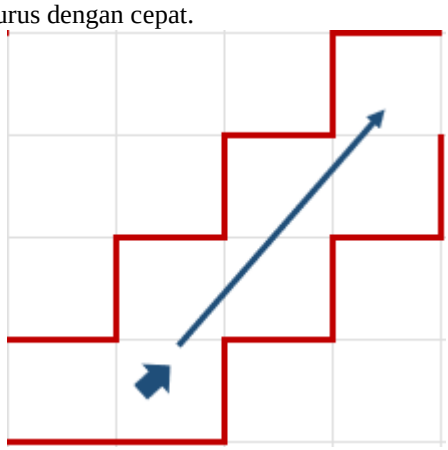


Fig. 10 ZigZag

IV. Simpulan dan Saran

Robot *micromouse* dapat menggunakan algoritma backtracking dalam menemukan solusi labirin dengan cara mencoba semua kemungkinan lintasan yang ada. Dengan cara ini, kita mengetahui semua solusi lintasan untuk selanjutnya dapat kita lakukan pencarian lintasan terpendek. Namun, perlu ada beberapa kalibrasi pada robot. Hal itu disebabkan oleh keterbatasan pengetahuan robot akan sekitarnya.

Algoritma *backtracking* memberikan kita semua kemungkinan lintasan. Namun proses mendapatkannya bisa lama dan perlu mengembangkan pohon menjadi sangat besar. Hal tersebut bisa menjadi ancaman mengingat adanya peraturan batasan waktu pencarian sebesar 10 menit. Selain itu, memori microcontroller yang digunakan robot juga terbatas. Oleh sebab itu, perlu dicari cara yang lebih efektif dan efisien dalam menumakan solusi lintasan labirin.

Berdasarkan apa yang ditulis seharusnya dicoba untuk dibuatkan robot micromouse sungguhan. Namun, keterbatasan waktu membuat hal itu tidak dapat terwujud.

V. Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas berkah dan rahmatnya, penulis dapat menulis makalah ini dapat diselesaikan sebenar-benarnya. Terima kasih kepada seluruh keluarga, teman, kerabat, dan pasangan penulis atas support yang diberikan selama menjalani perkuliahan. Terima kasih kepada jajaran dosen mata kuliah IF2120, terutama Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc. atas ilmu yang diberikan. Tak lupa asisten-asisten mata kuliah IF2120 yang telah membantu keberjalanan kuliah selama semester ini.

Referensi

[1] Misra, R., & Adler, R. 2018. Micromouse Competition Rules. 1, 2–4.
[2] Munir, Rinaldi. 2006. Diktat Kuliah Matematika Diskrit. Institut Teknologi Bandung: Bandung.
[3] H. James. Recursion. University of Utah: Utah https://www.cs.utah.edu/~germain/PPS/Topics/recursion.html
[4] Nugraha, M. I ,Dkk. 2011. Pencarian Jalur Terpendek untuk Robot Micromouse dengan Menggunakan Algoritma Backtracking. Politeknik Elektronika Negeri Surabaya: Surabaya.

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, 14 Desember 2021

Dimas Faidh Muzaki/13520156