Penerapan Algoritma Runut Balik dan Algoritma Knuth-Morris-Pratt pada Permainan Caveboy Escape

Gifari Kautsar (13512020)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13512020@std.stei.itb.ac.id

Abstrak*—Algoritma runut balik, atau backtracking, adalah algoritma yang digunakan untuk menemukan solusi dari suatu permasalahan dengan mencoba beberapa kandidat solusi secara berulang dan melakukan "backtrack" atau mundur jika kandidat tersebut tidak mungkin membentuk solusi yang benar. Algoritma ini dapat digunakan dalam berbagai persoalan seperti mencari solusi dari permainan berjenis puzzle, Caveboy Escape. Caveboy Escape adalah permainan untuk mencari jalan keluar pada sebuah bidang berukuran 5x5 sampai 6x6 dengan batasan harus memiliki jenis kotak yang sama pada setiap 3 langkah. Pada kali ini, selain dengan algoritma runut balik, akan dicoba penggabungan juga algoritma runut balik dan algoritma Knuth-Morris-Pratt (KMP) dalam mencari solusi Caveboy Escape.

***Kata kunci*—Caveman Escape, backtrack, Knuth-Morris-Pratt, Pattern Matching**

I. PENDAHULUAN

Berbagai algoritma telah ditemukan untuk mencari solusi permasalahan. Setiap algoritma ini memiliki strategina masing-masing. Algoritma *burte-force* dan algoritma *greedy* menggunakan strategi solusi langsung, algoritma runut balik dan algoritma *branch and bound* menggunakan strategi berbasis pencarian pada ruang status, algoritma *divide and conquer* dan *dynamic programming* menggunakan strategi solusi atas-bawah. Pada permainan Caveboy Escape, semua algoritma itu dapat diterapkan, namun pada kali ini akan digunakan algoritma runut balik.

Pada kali ini, selain dengan algoritma runut balik saja, akan dicoba penggunaan algoritma Knuth-Morris-Pratt (KMP) untuk mengurangi simpul-simpul pada algoritma runut balik dalam mencari solusi pada permainan Caveboy Escape.

II. ALGORITMA RUNUT BALIK[1]

Istilah *backtracking* pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950 dan terdapat juga tokoh-tokoh seperti R.J.Walker, Golomb, Baumert yang menyjikan uraian umum tentang *backtracking*.

Algoritma *backtracking* adalah algoritma pencarian solusi yang berbasis *DFS (Depth First Search)*. Algoritma *backtracking* banyak diterapkan untuk program games:

1. Permainan *tic-tac-toe*
2. Menemukan jalan keluar dari sebuah *maze*
3. Permainan *Crossword puzzle*
4. Catur

Algoritma runut balik merupakan perbaikan dari algoritma *brute force*. Pada algoritma *brute force* semua kemungkinan solusi dieksplorasi satu per satu sedangkan pada algoritma runut balik hanya pilihan yang mengarah ke solusi yang dieksplorasi, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan kembali.

2.1. Properti Umum Algoritma Runut Balik

a. Solusi persoalan
Solusi dinyatakan dalam bentuk vektor dengan tupel: $X = (x_1, x_2, x_3,\ldots,x_n)$, $x_i \in S_i$.
Mungkin terjadi $S_1 = S_2 = \ldots = S_n$
Contoh: $S_i = \{0,1\}$, $x_i = 0$ atau $x_i = 1$

b. Fungsi pembangkit
Fungsi pembangkit nilai x_k
Dinyatakan dalam predikat $T(k)$ dimana $T(k)$ membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.

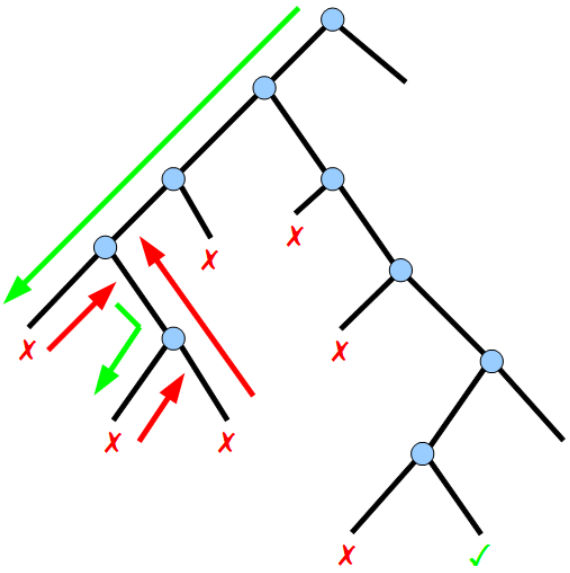
c. Fungsi pembatas
Dinyatakan dalam predikat : $B(x_1, x_2,\ldots,x_k)$
B bernilai benar jika $(x_1, x_2,\ldots,x_k)$ mengarah ke solusi. Jika benar, maka pembangkitan nilai untuk $x_k + 1$ dilanjutkan, tetapi jika false, maka $(x_1, x_2,\ldots,x_k)$ dibuang.

2.2. Prinsip Pencarian Solusi dengan Metode Runut Balik

a. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai mengikuti aturan *depth-first order* (*DFS*).
b. Simpul-simpul yang sudah dilahirkan dinamakan **simpul hidup** (*live node*).
c. Simpul-simpul yang sedang diperluas dinamakan **simpul-E** (*expand node*).
d. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang.
e. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" sehingga menjadi **simpul mati** (*dead node*)
f. Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan **fungsi pembatas**.

g. Simpul yang sudah mati tidak pernah diperluas lagi.
h. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian *backtrack* ke simpul aras diatasnya.
i. Lalu, teruskan dengan membangkitkan simpul anak lainnya.
j. Selanjutnya simpul ini menjadi simpu-E yang baru.
k. Pencarian dihentikan bila kita telah sampai pada *goal node*.

Berikut merupakan gambar ilustrasi algoritma runut balik:



Gambar 2.1 Ilustrasi Algoritma Runut Balik[2]

2.3. Kompleksitas Waktu Algoritma Runut Balik

Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemangilan rekursif. Jika jumlah simpul dalam pohon ruang status adalah 2^n atau n!, maka untuk kasus terburuk, algoritma runut balik membutuhkan waktu dalam:

a. O(p(n) 2^n) atau
b. O(q(n)n!)

Dengan p(n) dan q(n) adalah polinom derajat n yang menyatakan waktu komputasi simpul.

2.4. Skema Umum Algoritma Runut Balik menggunakan Rekursif

a. Skema rekursif

```
procedure RunutBalikR(input k:integer)
{Mencari semua solusi persoalan dengan metode
runut-balik; skema rekursif
Masukan: k, yaitu indeks komponen vektor solusi,
x[k]
Keluaran: solusi x = (x[1], x[2], …, x[n])}
Algoritma
for tiap x[k] yang belum dicoba sedemikian
sehingga ([k]←T(k)) and B(x[1], x[2], ...
,x[k])= true do
 if (x[1], x[2], ... ,x[k]) adalah lintasan dari
akar ke daun then
 CetakSolusi(x)
 endif
 RunutBalikR(k+1) {tentukan nilai untuk x[k+1]}
endfor
```

b. Skema iteratif

```
procedure RunutBalikI(input n:integer)
{Mencari semua solusi persoalan dengan metode
runut-balik; skema iteratif.
Masukan: n, yaitu panjang vektor solusi
Keluaran: solusi x = (x[1], x[2], …, x[n])}
Delarasi
 k : integer
Algoritma
 k←1
 while k > 0 do
 if (x[k] belum dicoba sedemikian sehingga
x[k]<--T(k)) and (B(x[1], x[2], ... ,x[k])=
true) then
 if (x[1],x[2],...,x[k]) adalah lintasan dari
akar ke daun then
 CetakSolusi(x)
 endif
 k←k+1 {indeks anggota tupple berikutnya}
 else {x[1], x[2], …, x[k] tidak mengarah ke
simpul solusi }
 k←k-1 {runut-balik ke anggota tupple
sebelumnya}
 endif
 endwhile
```

III. ALGORITMA KNUTH-MORRIS-PRATT[1]

Algoritma Knuth-Morris-Pratt dikembangkan oleh D. E. Knuth, bersama-sama dengan J. H. Morris dan V. R. Pratt. Dalam algoritma pencarian *string* dengan menggunakan *brute force*, ketika ditemukan ketidakcocokan *pattern* dengan teks, maka *pattern* tersebut digeser satu karakter ke kanan. Sedangkan jika menggunakan algoritma KMP (Knuth-Morris-Pratt), informasi yang digunakan untuk melakukan jumlah pergeseran setiap terjadi ketidakcocokan akan dikelola dalam algoritma ini. Kemudian algoritma ini menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak seperti algoritma *brute force* yang hanya digeser satu karakter. Dengan algoritma KMP, waktu pencarian dapat dikurangi secara signifikan. Dalam algoritma KMP, terdapat beberapa definisi yang digunakan :

Misalkan A adalah alfabet dan $x = x_1, x_2, x_3,..,x_k$, $k \in$ **N**, adalah *string* yang panjangnya *k* yang dibentuk dari karakter-karakter di dalam alfabet *A*.

a. Awalan (*prefix*) dari *x* adalah upa-*string* (*substring*) *u* dengan $u = x_1x_2x_3,..,x_{k-1}$, $k \in \{1,2, \ldots, k-1\}$ dengan kata lain, *x* diawali dengan *u*.
b. Akhiran (*suffix*) dari *x* adalah upa-*string* (*substring*) *u* dengan $u = x_k$–b x_k–b+1 … x_k , $k \in \{1, 2, \ldots, k-1\}$ dengan kata lain, *x* diakhiri dengan *v*.
c. Pinggiran (*border*) dari *x* adalah upa-*string* (*substring*) *r* sedemikian sehingga $r = x_1x_2x_3,..,x_{k-1}$ dan $u = x_k$–b x_k–b+1 … x_k , $k \in \{1, 2, \ldots, k-1\}$, dengan kata lain, pinggiran dari *x* adalah upa-*string* yang keduanaya awalan dan juga akhiran sebenarnya dari *x*.

Algoritma KMP melakukan proses awal terhadap *pattern* P dengan menghitung fungsi pinggiran yang

mengindikasikan pergeseran *s* terbesar yang mungkin dengan menggunakan perbandingan yang dibentuk sebelum pencarian string. Dengan menggunakan fungsi pinggiran ini, dapat dicegah pergeseran yang tidak berguna seperti yang terjadi pada algoritma *brute force*. Fungsi pinggiran tersebut hanya bergantung pada karakter-karakter di dalam *pattern*, bukan karakterkarakter yang ada di dalam teks. Fungsi pinggiran *b(j)* didefinisikan sebagai ukuran awalan terpanjang dari P yang merupakan akhiran dari P[1..*j*]. Sebagai contoh, misalnya *pattern* P = abcabd. Nilai fungsi pinggiran untuk setiap karakter dalam *pattern* P sebagai berikut:

j	1	2	3	4	5	6
P[j]	a	b	c	a	b	d
b(j)	0	0	0	1	2	0

Algoritma untuk menghitung pinggiran sebagai berikut:

```
procedure HitungPinggiran (input m : integer, P
: array[1..m] of char, output b : array[1..m] of
integer)
{ Menghitung nilai b[1..m] untuk pattern P[1..m]
}
Deklarasi
k,q : integer

Algoritma:
b[1]<-0
q<-2
k<-0
for q<-2 to m do
    while ((k > 0) and (P[q] != P[k+1])) do
        k<-b[k]
    endwhile
    if P[q]=P[k+1] then
        k<-k+1
    endif
    b[q]=k
endfor
```

Selanjutnya dilakukan percobaan pencocokan *pattern* P dengan sebuah teks T = abcabcabd. Awal pencocokan digambarkan sebagai berikut:

Teks	a	b	c	a	b	c	a	b	d
Pattern	a	b	c	a	b	d			

Kemudian ditemukan ketidakcocokan pada posisi ke 6. Jumlah pergeseran setelah itu ditentukan oleh pinggiran dari awalan P yang bersesuain. Awalan yang diperoleh yaitu abcab, dengan panjang l = 5 dan pinggiran terpanjang untuk string P[1..5] adalah *ab* dengan nilai fungsi pinggirannya 2 sehingga jarak pergeseran selanjutnya adalah $l - b = 5 - 2 = 3$. Setelah digeser sebanyak 3 karakter akhirnya *pattern* P ditemukan dalam teks T.

Teks	a	b	c	a	b	c	a	b	d
Pattern				a	b	c	a	b	d

IV. CAVEBOY ESCAPE

Caveboy Escape merupakan permainan yang tersedia untuk iPad, iPhone, Android, dan Blackberry. Caveboy Escape adalah permainan berjenis *puzzle* yang menggabungkan jenis permainan *match-3* dengan jenis permainan *connecting tile* dan menantang pemain untuk membantu Caveboy untuk keluar dari labirin. Labirin merupakan bidang berukuran 5x5 sampai 6x6 yang berisi kotak-kotak yang memiliki jenis tertentu.

Gambar 4.1 Tampilan awal permainan Caveboy Escape[3]

Pada permainan ini, Caveboy bisa melangkah ke kiri, kanan, atas, dan bawah. Ada batasan pada setiap langkahnya, yaitu harus melangkah pada jenis kotak yang sama pada setiap 3 langkah.

Gambar 4.2 Ilustrasi permainan Caveboy Escape[3]

Pada setiap level pada Caveboy Escape, labirin memiliki kombinasi kotak pada labirin yang berbeda-beda. Setiap labirin ini pasti memiliki solusi untuk mencapai titik akhir dari titik awal. Selain mencari solusi, pemain juga ditantang dengan waktu. Pemain dapat

mengumpulkan bintang pada setiap level yang jumlahnya tergantung dengan seberapa cepat pemain menemukan solusi.

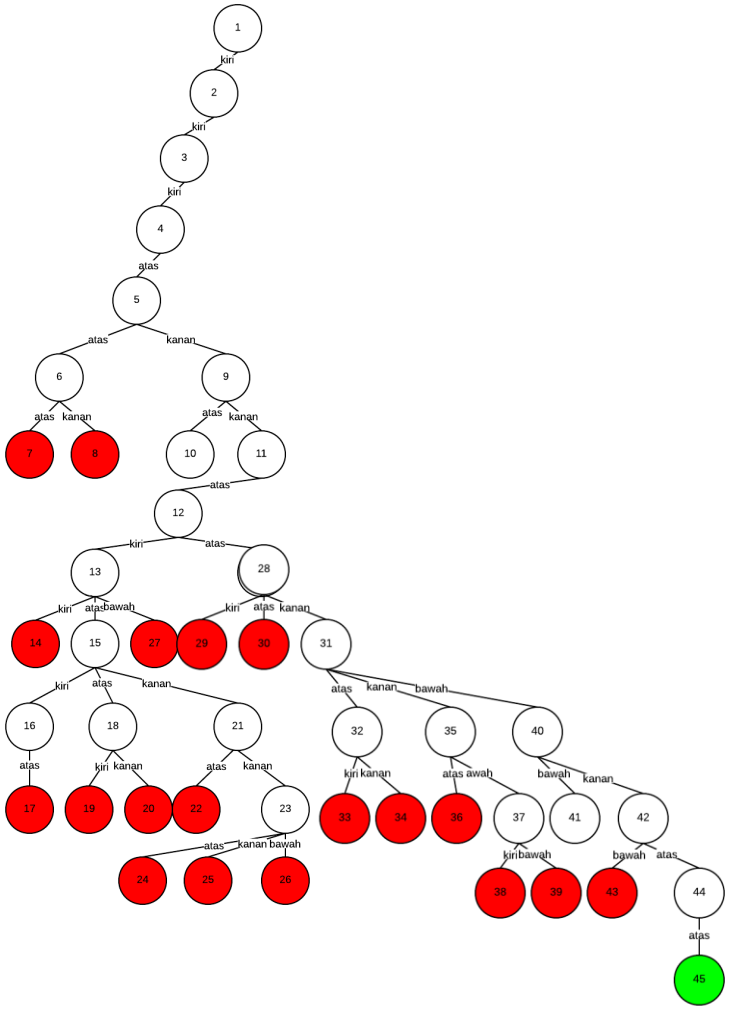
V. Pencarian Solusi Caveboy Escape dengan Algoritma Runut Balik

Dalam permainan Caveboy Escape, Caveboy dapat melangkah ke kiri, kanan, atas, dan bawah. Ada batasan dalam langkahnya yaitu harus melangkah pada jenis kotak yang sama pada setiap 3 langkah. Proses pencarian solusi dengan menggunakan algoritma runut balik adalah sebagai berikut.

1. Proses dimulai dengan menginisiasi langkah pertama dari Caveboy. Titik awal dianggap sebagai langkah pertama. Langkah tersebut dijadikan sebagai *node* pertama atau akar dalam sebuah pohon, lalu tandai node pertama sebagai node yang valid. Pada kali ini, urutan pemilihan langkah adalah kiri, atas, kanan, bawah.
2. Lalu lakukan pengecekan pada anak pertama dari *node* pertama, lalu telusuri anak dari *node* pertama tersebut.
3. Karena jenis kotak harus sama pada setiap tiga langkah, maka *node* pada kedalaman 2+3*k*, dengan *k* bulat, tidak perlu dilakukan pengecekan terhadap parrent-nyadan selalu dapat ditelusuri karena node tersebut merupakan langkah awal dari 2 langkah lainnya.
4. Lalu, ambil langkah berikutnya dan lakukan pengecekan terhadap langkah sebelumnya. Jika langkah tersebut memiliki jenis kotak yang sama, maka lanjutkan ke langkah sebelumnya. Jika langkah tersebut tidak valid, maka ambil langkah lain dan lakukan pengecekan.
5. Kotak yang telah diambil tidak dapa diambil lagi
6. Jika semua langkah pada sebuah *node* tidak valid, maka dilakukan *backtrack*.

Berikut pembentukan pohon algoritma runut balik pada pencarian solusi Caveman Escape level 39.

Gambar 5.1. Caveboy Escape level 39



Gambar 5.2. Pohon solusi Caveboy Escape level 39

Pada percobaan pencarian solusi Caveman Escape level 39, terbentuk pohon dengan jumlah node 45, dengan node berwarna merah berarti tidak menuju ke solusi (backtrack) dan node berwarna hijau adalah solusi. Dengan mengikuti langkah-langkah yang dibentuk melalui algoritma runut balik, maka diperoleh solusi sebagai berikut.

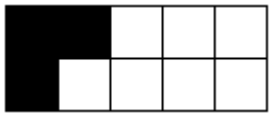
Gambar 5.3. Solusi Caveboy Escape level 39

VI. Pencarian Solusi Caveboy Escape dengan Knutt-Morris-Pratt dan Algoritma Runut Balik

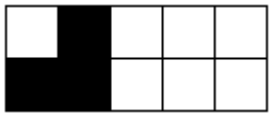
Penggunaan algoritma KMP pada pencarian solusi Caveboy Escape dilakukan untuk mencari susunan tiga kotak berurutan dapat dilalui oleh Caveboy. Dengan diketahuinya susunan apa saja yang terdapat pada labirin, percobaan pengambilan langkah dapat direduksi. Pada kali ini, labirin yang akan diamati adalah labirin dengan ukuran 5x5. Kemungkinan usunan tiga kotak tersebut adalah:

1. Pada susunan pertama, pattern yang terbentuk adalah: x0000xx, dengan x adalah jenis kotak dan 0 adalah jenis kotak selain x. Fungsi pinggiran dari *pattern* ini adalah 0000011.

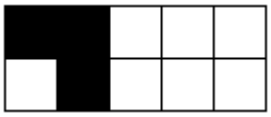
2. Pada susunan pertama, pattern yang terbentuk adalah: xx000x, dengan x adalah jenis kotak dan 0 adalah jenis kotak selain x. Fungsi pinggiran dari *pattern* ini adalah 010001



3. Pada susunan pertama, pattern yang terbentuk adalah: x000xx, dengan x adalah jenis kotak dan 0 adalah jenis kotak selain x. Fungsi pinggiran dari *pattern* ini adalah 000011.

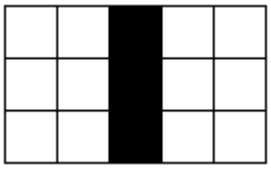


4. Pada susunan pertama, pattern yang terbentuk adalah: xx0000x, dengan x adalah jenis kotak dan 0 adalah jenis kotak selain x. Fungsi pinggiran dari *pattern* ini adalah 0100001.

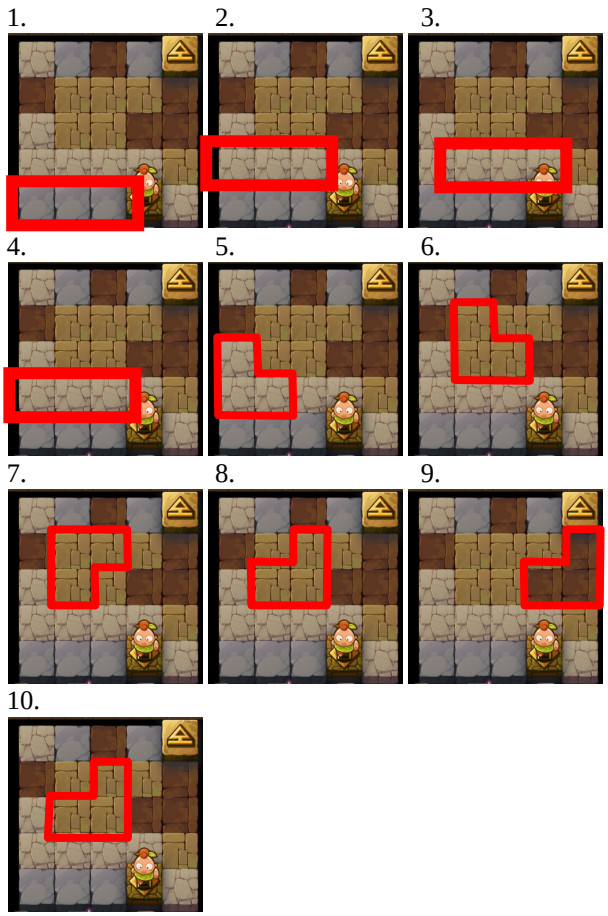


5. Pada susunan pertama, pattern yang terbentuk adalah: xxx, dengan x adalah jenis kotak. Fungsi pinggiran dari *pattern* ini adalah 012.

6. Pada susunan pertama, pattern yang terbentuk adalah: x0000x0000x, dengan x adalah jenis kotak dan 0 adalah jenis kotak selain x. Fungsi pinggiran dari *pattern* ini adalah 00000123456



Dengan penggunaan algoritma KMP bisa ditemukan pattern apa saja yang terdapat pada labirin dan pattern tersebut merupakan langkah yang dapat diambil. Dari algoritma KMP tersebut, diketahui indeks awal dan akhir dari sub-labirin yang memenuhi pattern. Misal pada Caveboy Escape level 39, terdapat 10 pattern yang dapat terpenuhi. Berikut pattern yang terpenuhi:



Gambar 6.1. Pattern pada Caveboy Escape level 39

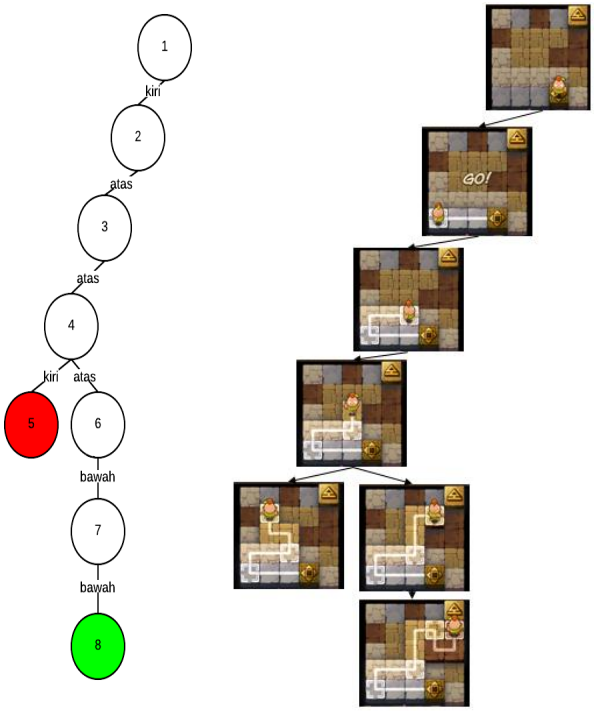
Setelah didapatkan indeks awal dan akhir setiap pattern pada labirin, maka dilakukan algoritma runut balik sebagai berikut.

1. Proses dimulai dengan menginisiasi langkah pertama dari Caveboy. Titik awal dianggap sebagai langkah pertama. Langkah tersebut dijadikan sebagai *node* pertama atau akar dalam sebuah pohon, lalu tandai node pertama sebagai node yang valid. Pada kali ini, urutan pemilihan langkah adalah kiri, atas, kanan, bawah.
2. Lalu lakukan pengecekan pada anak pertama dari *node* pertama. Pengecekan dilakukan dengan cara apakah kotak pada langkah tersebut merupakan indeks awal atau indeks akhir pada pattern yang terdapat pada labirin atau bukan.
3. Bila pada sebuah langkah terdapat lebih dari satu indeks sebuah pattern, maka langkah selanjutnya harus ditentukan.
4. Langkah berikutnya adalah pengambilan langkah dari indeks awal (jika pada node parrent ditemukan indeks akhir) atau indeks akhir (jika pada node parrent ditemukan indeks awal) dan lakukan

pengecekan seperti pada tahap nomor 2. Jika langkah tersebut tidak valid, maka ambil langkah lain dan lakukan pengecekan.

5. Kotak yang telah diambil tidak dapat diambil lagi
6. Jika semua langkah pada sebuah *node* tidak valid, maka dilakukan *backtrack*.

Berikut pembentukan pohon algoritma runut balik pada pencarian solusi Caveman Escape level 39.



Gambar 6.2. Pohon solusi Caveboy Escape level 39

Dengan menggunakan algoritma KMP dan algoritma runut balik, terbentuk pohon dengan jumlah node 8 untuk mencapai solusi. Solusi yang diperoleh sama dengan solusi yang dihasilkan hanya dengan algoritma runut balik

VII. KESIMPULAN

Dalam mencari solusi dari permainan Caveboy Escape, dapat digunakan berbagai jenis algoritma. Salah satu algoritma tersebut adalah algoritma runut balik. Yang menjadi batasan untuk mengetahui suatu langkah menuju solusi atau bukan dilihat dari aturan Caveboy Escape yang mengharuskan setiap tiga langkah dari Caveboy harus sama. Jika aturan tersebut tidak terpenuhi maka akan dilakukan *backtrack*.

Adanya aturan setiap tiga langkah harus sama, memungkinkan penggunaan algoritma KMP untuk menentukan kotak mana saja pada labirin yang dapat diambil sehingga dapat membentuk tiga kotak yang sama. Dengan mengetahui kotak yang memenuhi pattern tersebut dapat diketahui langkah mana saja yang dapat diambil sehingga Caveboy bisa lebih menuju ke solusi. Sehingga dapat dilihat pada pembahasan sebelumnya bahwa dengan penggabungan algoritma KMP dengan algoritma runut balik dapat menghasilkan hasil yang lebih baik daripada penggunaan algoritma runut balik saja.

VIII. ACKNOWLEDGMENT

Ucapan terima kasih saya ucapkan kepada Dr. Rinaldi Munir dan Dr. Masayu Leyla Khodra atas segala bimbingannya dalam slide presentasi dan diktat untuk mengerjakan makalah ini. Terima kasih turut diberikan kepada teman-teman yang telah membantu memberikan ide, dan ikut turut dalam memberikan referensi-referensi yang saya dapat sehingga saya dapat menyelesaikan makalah ini.

REFERENCES

[1] Munir, Rinaldi, Diktat Kuliah IF2211 Strategi Algoritma. Program Studi Teknik Informatika ITB, 2013.
[2] http://www.w3.org/2011/Talks/01-14-steven-phenotype/, diakses 19 Mei 2014 pukul 1:34
[3] http://www.appxplore.com/index.php/games/caveboy-escape, diakses 19 Mei 2014 pukul 2:12

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, 19 Mei 2014

Gifari Kautsar (13512020)