# Penerapan Algoritma Runut-Balik untuk Menyelesaikan Teka-Teki Magnet

Isa Mujahid Darussalam 13517002
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*13517002@std.stei.itb.ac.id*

***Abstrak*—Teka-teki magnet adalah suatu teka-teki di mana disediakan sebuah papan yang dapat diisi dengan suatu magnet dengan kutub (+,-) atau magnet netral. Tujuan dari teka-teki ini adalah mampu menyusun magnet pada papan namun tidak melanggar batasan-batasan yang ada dalam permainan ini. Dalam makalah ini akan dijelaskan bagaimana langkah-langkah menyelesaikan permainan teka-teki magnet ini menggunakan algoritma *backtracking*.**

***Kata kunci—Magnet puzzle, Backtracking, Teka-teki magnet, runut-balik.***

## I. PENDAHULUAN

Permainan *puzzle* atau teka-teki merupakan jenis permainan yang sering ditemui, bahkan eksistensinya sudah banyak sejak zaman dahulu. Permainan berjenis ini lumayan digemari karena dapat mengasah kemampuan berpikir dari pemainnya. Semakin tinggi tingkat kerumitan suatu teka-teki, maka semakin terlatih kemampuan berpikir terutama perihal pemecahan masalah (*problem solving*).

Secara istilah, permainan teka-teki dapat diartikan sebagai sebuah permainan yang dibuat untuk mengasah kemampuan berpikir dan logika dari pemainnya. Pada zaman sekarang, permainan teka-teki dapat ditemukan dalam berbagai macam bentuk dan media, seperti permainan tradisional, hingga permainan pada komputer atau telepon pintar.

Contoh-contoh dari permainan teka-teki antara lain permainan tradisional seperti congklak, *jigsaw puzzle*, hingga permainan sudoku adalah beberapa contoh dari permainan teka-teki. Pada makalah kali ini permainan teka-teki yang akan dibahas adalah teka-teki magnet.

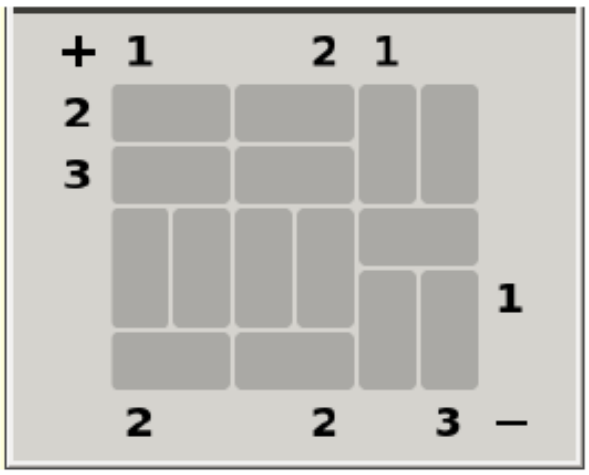
Ada berbagai macam langkah untuk menyelesaikan teka-teki magnet. Kita dapat menggunakan algoritma *brute-force*, namun tentunya algoritma ini kurang baik dari segi efisiensinya. Oleh sebab itu, saya mengimplementasikan algoritma runut-balik untuk menyelesaikan teka-teki magnet ini.

## II. LANDASAN TEORI

### A. *Permainan teka-teki magnet*

Permainan teka-teki magnet adalah suatu jenis permainan teka-teki dengan tujuan mengisi papan permainan dengan dua jenis magnet, yaitu magnet yang memiliki kutub positif dan negatif serta magnet yang netral atau tidrak bermuatan.

Berikut adalah ilustrasi papan permainan magnet yang masih kosong:



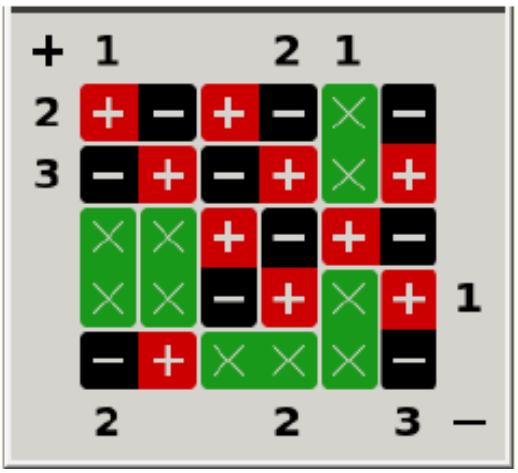

**Gambar 1.1: Papan permainan teka-teki magnet yang masih kosong.**
(Sumber : https://www.techiedelight.com/magnet-puzzle/, diakses 25 April 2019 pukul 20:05 GMT+7).

Permainan ini memiliki beberapa batasan (constraints) antara lain:

1. Pada satu petak magnet, apabila mengisi dengan magnet yang memiliki kutub positif dan negatif, maka kutub positif hanya boleh berpasangan dengan kutub negatif. Tidak boleh ada magnet yang bermuatan positif-positif atau negative-negatif.

2. Magnet satu dengan yang lain yang berimpitan tidak boleh memiliki pasangan kutub yang sama. Dengan kata lain, kutub positif hanya boleh berimpitan dengan kutub negatif dan sebaliknya.
3. Pada bagian atas, kiri, bawah, dan kanan terdapat angka-angka. Angka ini melambangkan jumlah kutub yang harus dipatuhi. Bagian atas melambangkan jumlah kutub positif pada suatu kolom, bagian kiri melambangkan jumlah kutub positif pada suatu baris, bagian bawah melambangkan jumlah kutub negatif pada suatu kolom, dan bagian kanan melambangkan jumlah kutub negative pada suatu baris.

Dari batasan-batasan di atas, maka solusi dari papan magnet pada gambar 1.1 adalah sebagai berikut:




**Gambar 1.2: Solusi papan permainan gambar 1.1**
(Sumber : https://www.techiedelight.com/magnet-puzzle/, diakses 25 April 2019 pukul 20:06 GMT+7).

Perhatikan gambar 1.2 di atas. Semua batasan pada permainan telah terpenuhi, yaitu tidak ada magnet dengan kutub yang sama saling berimpit serta jumlah kutub sesuai dengan klu yang terdapat pada atas, kiri, bawah, dan kanan dari papan permainan.

### *B. Algoritma runut-balik*

Algoritma runut-balik pertama kali diperkenalkan oleh D. H. Lehmer pada tahun 1950. R.J Walker, Golomb, dan Baumert kemudian menjelaskan secara umum tentang algoritma runut-balik.

Algoritma runut-balik adalah perbaikan atau versi lebih baik daripada algoritma DFS. Algoritma ini digunakan untuk menemukan solusi dari suatu permasalahan dengan mengeksplorasi tiap-tiap kandidat yang mengarah ke solusi dan mengabaikan kandidat simpul-simpul yang ternyata tidak mungkin memenuhi kondisi dari solusi.[2][3]

Algoritma runut-balik dapat disebut sebagai suatu metode yang efektif, terstruktur, dan sistematis dibandingkan dengan algoritma *exhaustive search/brute force*. Pada *exhaustive search*, semua kemungkinan solusi dieksplorasi satu per satu. Namun, pada algoritma runut-balik, hanya pilihan-pilihan yang mengarah kepada solusi yang akan dipertimbangkan, sedangkan pilihan-pilihan yang tidak mengarah ke solusi tidak akan dipertimbangkan lagi. Simpul-simpul yang tidak mengarah kepada solusi tersebut dipangkas atau sebutan lainnya yaitu *pruning*.

Algoritma runut-balik merupakan salah satu algoritma yang popular dan telah banyak diterapkan untuk menyelesaikan berbagai permasalahan di bidang ilmu computer. Beberapa contoh penerapan dari algoritma runut-balik antara lain untuk menyelesaikan permasalahan seperti *N-Queens*, *knight tour*, sudoku, *scrabble*, pewarnaan graf, parsing, dan banyak lainnya.

Algoritma backtracking mempunyai beberapa properti umum seperti:

1. Solusi permasalahan.
   Solusi dinyatakan sebagai vector dengan *n-tuple:* X = ($x_1$, $x_2$, …,$x_n$ ), dengan $x_i \in S_i$, di mana $x_i$ adalah langkah yang diambil ketika membuat solusi permasalahan. Nilai dari setiap S dimungkinkan $S_1 = S_2 = \ldots = S_n$.
2. Fungsi pembangkit nilai $x_k$.
   Fungsi pembangkit dinyatakan sebagai T(k). T(k) menyatakan sebuah nilai atau langkah yang dapat dilakukan. Hasil dari T(k) dapat disebut sebagai $x_k$, yaitu komponen dari vektor solusi.
3. Fungsi pembatas
   Fungsi pembatas dinyatakansebagai predikat B($x_1$, $x_2$, …, $x_k$). B akan bernilai *true* apabila ($x_1$, $x_2$, …, $x_k$) mengarah ke solusi. Jika B bernilai *true*, maka pembangkitan nilai untuk $x_{k+1}$ dilanjutkan, tetapi apabila B bernilai *false*, maka ($x_1$, $x_2$, …, $x_k$) dibuang.

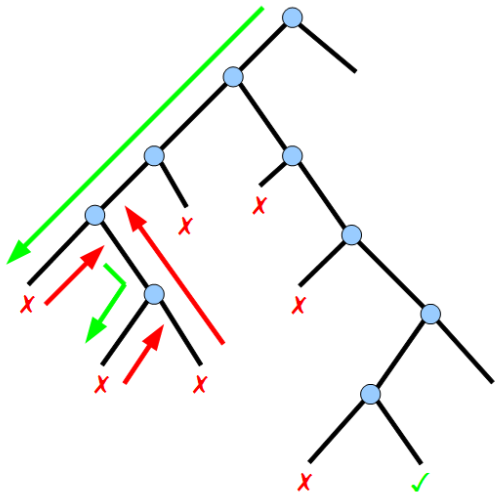
Semua kemungkinan solusi pada dari persoalan pada algoritma runut-balik disebut sebagai **ruang solusi (***solution space*). Sebagai contoh, akan ditinjau ruang solusi untuk permasalahan *Knapsack* 0/1 untuk nilai n = 3. Solusi persoalan untuk permasalahan ini dapat dinyatakan sebagai ($x_1$, $x_2$, $x_3$) dengan $x_i \in \{0,1\}$ dengan 0 menyatakan barang tersebut tidak diambil, dan 1 menyatakan barang tersebut diambil. Maka, didapatkan ruang solusinya adalah: {(0,0,0), (0,1,0), (0,0,1), (1, 0, 0), (1, 1, 0) , (1, 0, 1), (0, 1, 1), (1, 1, 1)}.

Ruang solusi direpresentasikan dalam bentuk pohon. Tiap simpul pada pohon menyatakan status (*state*) dari persoalan, sedangkan sisi (cabang) merupakan nilai-nilai dari $x_i$. Solusi

yang mungkin dinyatakan sebagai lintasan dari akar ke daun. Seluruh lintasan dari akar ke daun membentuk ruang solusi, namun solusi yang digunakan hanyalah solusi yang memenuhi kondisi dari permasalahan dan tidak melanggar batasan-nbatasn yang telah ditetapkan. Pengorganisasian ruang solusi dalam bentuk pohon diacu sebagai **pohon ruang status** (*state space tree*).

Prinsip pencarian solusi dengan algoritma runut-balik adalah mengikuti aturan *depth-first order* (DFS), yaitu solusi dicari dengan membentuk lintasan dari akar menuju ke daun. Simpul-simpul yang telah dilewati/dimunculkan dinamakan sebagai **simpul hidup** (*live node*), sedangkan simpul-simpul yang sedang diperluas dinamakan sebagai **simpul ekspan** (*expand node*).

Setiap kali simpul ekspan diperluas, lintasan yang dibangun oleh pohon ruang solusi akan bertambah panjang. Jika saat melakukan perluasan ternyata lintasan yang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut dibunuh sehingga simpul tersebut menjadi **simpul mati** (*dead node*) yang tidak akan diperluas lagi. Fungsi yang digunakan untuk membunuh simpul ekspan disebut sebagai fungsi pembatas (*bounding function*). Jika pembentukan lintasan berakhir pada simpul mati, maka dilakukan runut-balik ke simpul dengan level di atasnya. Lalu, proses dilanjutkan kembali dengan melakukan perluasan pada simpul anaknya yang lain sehingga simpul tersebut menjadi simpul ekspan yang baru. Pencarian akan dihentikan apabila kita telah sampai pada **simpul tujuan** (*goal node*).



**Gambar 2.1: Pohon ruang status algoritma runut-balik.**
(Sumber : https://www.w3.org/2011/Talks/01-14-steven-phenotype/, diakses 26 April 2019 pukul 07:21 GMT+7).

Secara umum, algorima *runut balik* diterapkan dengan fungsi rekursif dengan *pseudocode* sebagai berikut:

```
boolean solve(Node n) {
  if n is a leaf node {
    if the leaf is a goal node
      return true
    else
      return false
    } else {
        for each child c of n {
            if solve(c)
              return true
        }
        return false
    }
}
```

Algoritma runut-balik juga dapat diterapkan dalam fungsi iteratif atau non-rekursif dengan meamnfaatkan tipe data *stack*.

## III. Penyelesaian Teka-Teki Magnet dengan Algoritma Runut-Balik

Untuk menyelesaikan teka-teki magnet, sebenernya bisa saja dengan mencoba-coba seluruh kemungkinan yang ada (*brute force*). Namun tentunya cara ini tidak mangkus dan akan memerlukan waktu yang relatif lebih lama dibandingkan dengan algoritma runut-balik.

Apabila digunakan algoritma *brute-force*, misalkan kita memiliki papan permainan magnet berukuran 5 × 5. Satu petak dapat diisi dengan tiga kemungkinan yaitu dua kemungkinan magnet berkutub positif, kemungkinan magnet berkutub negatif, dan kemungkinan diisi oleh magnet bermuatan netral. Maka semua kemungkinan kombinasi yang dapat dihasilkan adalah sebanyak $3^{5 \times 5} = 847.288.609.443$ kemungkinan kombinasi. Jumlah tersebut sangat besar dan tentunya tidak efisien.

Sebelum memulai membuat fungsi untuk menyelesaikan teka-teki magnet dengan algoritma runut-balik, kita perlu menentukan properti-properti pada permasalahan ini yaitu solusi permasalahan, fungsi pembangkit, dan fungsi pembatas.

Kita dapat memandang papan permainan dengan sebagai sebuah matriks berukuran M × N. Kemudian sebagai kesepakatan, digunakan tanda '+' yang menandakan bagian kutub positif pada magnet, tanda '-' yang menandakan bagian kutub negatif dari magnet, dan tanda X yang melambangkan magnet tersebut netral. Maka solusi permasalahan berupa vektor *n-tuple* dengan X = ($x_{11}$, $x_{12}$, ..., $x_N$). Dengan batasan yaitu $x_i \in \{ +, -, X\}$, dengan $1 \leq i \leq M$, dan $1 \leq j \leq N$. Dari solusi permasalahan tersebut, solusi dari permasalahan teka-teki magnet adalah tiap petak pada vektor telah terisi dengan magnet yang sesuai.

Fungsi pembangkit pada permasalahan ini menerima masukan berupa kondisi papan permainan magnet saat ini. Kemudian pada petak yang masih kosong diisi dengan tanda +, -, ataupun X sehingga terbentuk kondisi papan yang baru.

Properti yang terakhir yaitu fungsi pembatas yang akan melakukan validasi apakah status pada simpul saat ini melanggar batasan atau tidak. Terdapat tiga batasan dalam permasalahan ini. Pertama, dalam satu petak magnet, pasangan kutub pada satu petak harus berlainan, yaitu positif-negatif. Tidak boleh ada petak dengan pasangan positif-positif atau negatif-negatif. Kedua, magnet yang saling berimpit harus memiliki muatan yang berlainan. Magnet dengan kutub positif hanya boleh berimpit dengan magnet berkutub negatif dan magnet berkutub negatif hanya boleh berimpit dengan magnet berkutub positif. Batasan yang terakhir, setiap baris dan kolom harus sesuai dengan ketentuan jumlah kutub yang telah ditetapkan. Aturannya, angka pada bagian atas adalah jumlah kutub positif yang harus dipenuhi pada kolom tertentu, pada bagian kiri adalah jumlah kutub positif yang harus dipenuhi pada baris tertentu, pada bagian bawah adalah jumlah kutub negatif yang harus dipenuhi pada kolom tertentu, dan pada bagian kanan adalah jumlah kutub negatif yang harus dipenuhi pada baris tertentu.

Secara umum, berikut adalah langkah-langkah penyelesaian teka-teki magnet:

1. Representasikan papan permainan awal dalam bentuk matriks yang berisi salah satu dari karakter T, B, L, atau R. Untuk petak vertikal pada papan, T menandakan bagian atas magnet dan B menandakan bagian bawah magnet. Untuk petak horizontal, L menandakan bagian kiri magnet dan R menandakan bagian kanan magnet. Berikut adalah contoh representasi papan permainan dalam bentuk matriks:

   ```
   [L R L R T T]
   [L R L R B B]
   [T T T T L R]
   [B B B B T T]
   [L R L R B B]
   ```

   Kemudian, representasikan pula batasan angka pada bagian atas, kiri, bawah, dan kanan dalam senarai. Tandai bagian yang memperbolehkan jumlah kutub positif dan negatif berapapun dengan tanda -.
2. Buat papan permainan baru yang memiliki ukuran sama dengan papan permainan awal. Papan oermainan baru ini yang digunakan untuk menyimpan solusi. Inisialisasi setiap petak pada papan permainan baru dengan tanda X, yaitu tanda bahwa petak tersebut diisi oleh magnet bermuatan netral.
3. Panggil fungsi solve, yaitu fungsi yang akan menyelesaikan teka-teki magnet dengan metode *backtracking*. Fungsi solve menerima parameter papan permainan awal, baris dan kolom yang saat ini sedang diproses, batasan angka bagian atas, bawah, kiri, dan kanan dari permainan, dan papan permainan baru untuk menyimpan solusi. Fungsi solve akan mengembalikan *true* apabila papan permainan bisa diselesaikan, dan mengembalikan *false* apabila tidak bisa diselesaikan. Fungsi solve ini memiliki langkah-langkah sebagai berikut:

   a. Basis dari kasus ini adalah apabila setiap petak pada matriks telah diperiksa sampai ke baris dan kolom terakhir. Lakukan validasi papan, apakah sudah sesuai dengan batasan angka yang terdapat pada bagian atas, kiri, bawah, dan kanan. Apabila sama, maka kembalikan *true*, apabila berbeda kembalikan *false*.
   b. Apabila belum mencapai basis, maka pertama lakukan pengecekan apakah kolom yang saat ini sedang diproses melebihi jumlah kolom pada papan. Apabila melebihi, maka harus pindah baris. Pindah baris dilakukan dengan mengatur kolom yang diproses menjadi bernilai 1, dan baris yang diproses menjadi baris+1.
   c. Lakukan pengecekan pada baris dan kolom yang saat ini diperiksa apakah pada papan permainan awal, baris dan kolom yang diperiksa berisi karakter ‘R’ atau ‘B’. Apabila betul, maka panggil fungsi rekurens dengan memanggil solve, namun dengan melakukan *passing* parameter kolom yang diproses dijumlahkan 1 (Ekspan baris dan kolom selanjutnya).
   d. Lakukan pengecekan pada baris dan kolom yang saat ini diperiksa apakah pada papan permainan awal, baris dan kolom yang diperiksa berisi karakter ‘L’ dan pada kolom berikutnya berisi karakter ‘R’. Apabila betul, maka lakukan pengecekan berikut:
      - Validasi jika bagian L diisi dengan karakter ‘+’ dan bagian R diisi karakter ‘-’ itu tidak melebihi jumlah kutub positif dan negatif yang diperbolehkan pada bagian atas dan bawah. Apabila tidak melebihi, maka isi papan permainan baru dengan pasangan (+, -). Kemudian, panggil fungsi rekurens solve, dengan *passing* parameter kolom yang diproses dijumlahkan 2. Apabila fungsi rekurens solve mengembalikan hasil *true*, maka akhiri fungsi solve dengan mengembalikan *true* juga. Sebaliknya, lakukan pruning pada simpul tersebut dan runut-balik ke simpul sebelumnya dengan mengatur baris dan kolom yang telah diisi dengan pasangan (+, -) menjadi tanda X (magnet netral).
      - Validasi jika bagian L diisi dengan karakter ‘-’ dan bagian R diisi karakter ‘+’ itu tidak melebihi

jumlah kutub positif dan negatif yang diperbolehkan pada bagian atas dan bawah. Apabila tidak melebihi, maka isi papan permainan baru dengan pasangan (-, +). Kemudian, panggil fungsi rekurens solve, dengan *passing* parameter kolom yang diproses dijumlahkan 2. Apabila fungsi rekurens solve mengembalikan hasil *true*, maka akhiri fungsi solve dengan mengembalikan *true* juga. Sebaliknya, lakukan pruning pada simpul tersebut dan runut-balik ke simpul sebelumnya dengan mengatur baris dan kolom yang telah diisi dengan pasangan (-, +) menjadi tanda X (magnet netral).

e. Lakukan pengecekan pada baris dan kolom yang saat ini diperiksa apakah pada papan permainan awal, baris dan kolom yang diperiksa berisi karakter 'T' dan pada baris berikutnya berisi karakter 'B'. Apabila betul, maka lakukan pengecekan berikut:
   - Validasi jika bagian T diisi dengan karakter '+' dan bagian B diisi karakter '-' itu tidak melebihi jumlah kutub positif dan negatif yang diperbolehkan pada bagian kiri dan kanan. Apabila tidak melebihi, maka isi papan permainan baru dengan pasangan (+, -). Kemudian, panggil fungsi rekurens solve, dengan *passing* parameter kolom yang diproses dijumlahkan satu. Apabila fungsi rekurens solve mengembalikan hasil *true*, maka akhiri fungsi solve dengan mengembalikan *true* juga. Sebaliknya, lakukan pruning pada simpul tersebut dan runut-balik ke simpul sebelumnya dengan mengatur baris dan kolom yang telah diisi dengan pasangan (+, -) menjadi tanda X (magnet netral).
   - Validasi jika bagian T diisi dengan karakter '-' dan bagian B diisi karakter '+' itu tidak melebihi jumlah kutub positif dan negatif yang diperbolehkan pada bagian kiri dan kanan. Apabila tidak melebihi, maka isi papan permainan baru dengan pasangan (-, +). Kemudian, panggil fungsi rekurens solve, dengan *passing* parameter kolom yang diproses dijumlahkan satu. Apabila fungsi rekurens solve mengembalikan hasil *true*, maka akhiri fungsi solve dengan mengembalikan *true* juga. Sebaliknya, lakukan pruning pada simpul tersebut dan runut-balik ke simpul sebelumnya dengan mengatur baris dan kolom yang telah diisi dengan pasangan (-, +) menjadi tanda X (magnet netral).

f. Sampai pada tahap ini, maka cukup abaikan petak saat ini dan panggil fungsi rekurens solve dengan *passing* parameter nilai komol yang diproses sekarang ditambah 1.

g. Apabila fungsi solve sampai pada tahap ini, maka kembalikan *false*, karena semua pengecekan pada bagian atas tidak ada yang berhasil mengembalikan *true*. Itu artinya, papan permainan tidak memiliki solusi.

4. Cetak papan permainan baru pada layar apabila fungsi solve mengembalikan nilai *true.*

Berikut *pseudocode* supaya lebih tergambarkan alur dari algoritma runut-baliknya:

```
bool solve(char board[M][N], int row, int col,
           int top[], int left[], int bottom[],
           int right[], char str[M][N])
{board adalah papan permainan baru. row dan col
masing masing adalah baris dan kolom yang sedang
diproses. top, left, bottom, dan right masing-
masing adalah batasan jumlah kutub pada atas(+),
bawah(-), kiri(+), dan kanan (-). str adalah papan
permainan awal}
ALGORITMA
    {Sudah mencapai baris dan kolom terakhir}
    if (row >= M and col >= N)
        if (papan tidak melanggar batasan)
            return true
        return false
    {jika col melebihi jumlah kolom pada papan}
    if (col > N)
        col = 1
        row = row + 1
    {jika papan yg diproses berisi R atau B}
    if (str[row][col] == 'R'
        or str[row][col] == 'B')
        if (solve(board, row, col + 1, top,
                  left, bottom, right, str))
            return true
    {jika papan yg diproses berisi L lalu R}
    if (str[row][col] == 'L'
        and str[row][col+1]  == 'R')
        if (aman meletakkan +)
            board[row][col] = '+'
            board[row][col + 1] = '-'
            if (solve(board, row, col+2,
                      top,left,bottom,right,str))
                return true
            {tidak menghasilkan solusi,backtrack}
            board[row][col] = 'X'
            board[row][col + 1] = 'X'
        if (aman meletakkan -)
            board[row][col] = '-'
            board[row][col + 1] = '+'
            if (solve(board, row, col+2,
                      top,left,bottom,right,str))
                return true
            {tidak menghasilkan solusi,backtrack}
            board[row][col] = 'X'
            board[row][col + 1] = 'X'
```

```
    {jika papan yg diproses berisi T lalu B}
    if (str[row][col] == 'T'
        and str[row+1][col]  == 'B')
        if (aman meletakkan +)
            board[row][col] = '+'
            board[row+1][col] = '-'
            if (solve(board, row, col+1,
                      top,left,bottom,right,str))
                return true
            {tidak menghasilkan solusi,backtrack}
            board[row][col] = 'X'
            board[row+1][col] = 'X'
        if (aman meletakkan -)
            board[row][col] = '-'
            board[row+1][col] = '+'
            if (solve(board, row, col+1,
                      top,left,bottom,right,str))
                return true
            {tidak menghasilkan solusi,backtrack}
            board[row][col] = 'X'
            board[row+1][col] = 'X'

    {abaikan row & col saat ini, panggil rekurens}
    if (solve(board, row, col + 1,
              top, left, bottom, right, str))
        return true
    {tidak ada solusi ditemukan, return false}
    return false
```

## IV. EKSPERIMEN

Pada bagian ini, akan dilakukan pengujian untuk tiga variasi papan permainan teak-teki magnet.

1. Pengujian pertama

Pada pengujian pertama ini, papan permainan yang digunakan adalah sebagai berikut:

```
[L R L R T T]
[L R L R B B]
[T T T T L R]
[B B B B T T]
[L R L R B B]
```

Kemudian, batasan-batasan yang digunakan adalah:

```
top[] = {1, -1, -1, 2, 1, -1}
bottom[] = {2, -1, -1, 2, -1, 3}
left[] = {2, 3, -1, -1, -1}
right[] = {-1, -1, -1, 1, -1}
```

Keluaran program adalah sebagai berikut:

```
+ - + - X -
- + - + X +
X X + - + -
X X - + X +
- + X X X -
Durasi: 0.000173 sec.
```

**Gambar 4.1: Hasil pengujian pertama.**

2. Pengujian kedua

Pada pengujian kedua, papan permainan yang digunakan adalah:

```
[T T T]
[B B B]
[T L R]
[B L R]
```

Kemudian, batasan-batasan yang digunakan adalah:

```
top[] = [2, -1, -1]
bottom[] = [-1, -1, 2]
left[] = [-1, -1, 2, -1]
right[] = [0, -1, -1, -1]
```

Keluaran program adalah sebagai berikut:

```
+ X +
- X -
+ - +
- + -
Durasi: 2.1e-05 sec.
```

**Gambar 4.2: Hasil pengujian kedua.**

3. Pengujian ketiga

Pada pengujian ketiga, papan permainan yang digunakan adalah:

```
[T T L R T T]
[B B T T B B]
[L R B B T T]
[T T L R B B]
[B B L R L R]
```

Kemudian, batasan-batasan yang digunakan adalah:

```
top[] = [-1, -1, -1, -1, 2, -1]
bottom[] = [2, 1, -1, -1, -1]
left[] = [3, 1, -1, -1, 2, -1]
right[] = [-1, 1, 2, -1, 2]
```

Keluaran program adalah sebagai berikut:

```
- X + - + X
+ X X X - X
- + X X + -
+ - X X - +
- + - + X X
Durasi: 0.892384 sec.
```

**Gambar 4.3: Hasil pengujian ketiga.**

## V. KESIMPULAN

Ada banyak algoritma untuk menyelesaikan teka-teki magnet. Salah satu algoritma yang efektif adalah algoritma runut-balik. Pada makalah ini dijelaskan langkah-langkah penyelesaian teka-teki magnet dengan algoritma runut-balik. Selain dari cara yang digunakan pada makalah ini, masih banyak cara lain untuk menyelesaikan teka-teki magnet dengan algoritma runut-balik, misalnya menganggap papan permainan dalam bentuk senarai satu dimensi.

## VI. UCAPAN TERIMA KASIH

Terima kasih saya ucapkan kepada sdr. Ahmad Mutawalli, yang telah memberikan bantuan dalam bentuk papan untuk mengerjakan makalah ini.

## REFERENCES

[1] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Runut-balik-(2018).pdf, diakses 26 April 2019 pukul 07:20 GMT+7.

[2] Gurari, Eitan. *CIS 680: DATA STRUCTURES: Chapter 19: Backtracking Algorithms.* 1999. Diakses dari : https://web.archive.org/web/20070317015632/http:/www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch19.html, diakses 26 April 2019 pukul 08.05 GMT+7.

[3] https://www.cis.upenn.edu/~matuszek/cit594-2012/Pages/backtracking.html, diakses 26 April 2019 pukul 08.05 GMT+7.

[4] https://www.chiark.greenend.org.uk/~sgtatham/puzzles/js/magnets.html, diakses 24 April 2019 pukul 21.05 GMT+7.

[5] https://www.techiedelight.com/magnet-puzzle/, diakses 24 April 2019 pukul 20.37 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, 10 Desember 2018

Isa Mujahid Darussalam