Penerapan Algoritma Backtracking pada Permainan Greedy Knight

Ahmad Fauzul Yogiandra / 13513059
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13513059@std.stei.itb.ac.id

***Abstrak*—*Greedy Knight* merupakan permainan yang mengisahkan tentang seorang kesatria yang ingin mengumpulkan semua koin emas yang ada. Terdapat tiga belas koin emas yang terletak pada pilar yang berbeda. Kesatria akan mengunjungi setiap pilar dengan langkah khusus dan batasan yang diatur dalam permainan. Untuk meraih kemenangan, kesatria harus menemukan rute perjalanan yang tepat. Solusi persoalan tersebut dapat diselesaikan dengan mentransformasi persoalan menjadi persoalan sirkuit Hamilton dan menerapkan algoritma *backtracking*. Algoritma *backtracking* dipilih karena algoritma ini dapat mencari solusi persoalan secara mangkus.**

Kata kunci—greedy knight; backtracking; sirkut hamilton

I. PENDAHULUAN

Greedy Knight merupakan permainan yang mengisahkan tentang seorang kesatria yang ingin mengumpulkan semua koin emas yang ada. Terdapat tiga belas koin emas yang terletak pada pilar yang berbeda. Untuk dapat mengumpulkan koin-koin tersebut, kesatria akan menunggai kuda dan bergerak seperti langkah dalam permainan catur, yaitu dua blok secara vertikal lalu satu blok secara horizontal, atau dua blok secara horizontal lalu satu blok secara vertikal. Langkah dari kesatria tersebut akan terlihat seperti membentuk huruf L.

Gambar 1. Tampilan Permainan *Greedy Knight* [1]

Pada awal permainan, kesatria akan bersiap-siap di pilar di sebelah tebing. Pilar-pilar selain pilar awal memiliki kondisi yang rentan. Ketika kesatria akan berpindah ke pilar lain, pilar sebelumnya akan jatuh. Hal ini menyebabkan setiap pilar selain pilar awal hanya dapat dikunjungi sekali.

Selama permainan, kesatria akan terus melangkah dari satu pilar ke pilar lainnya. Jika kesatria berhasil mendapatkan seluruh koin yang ada dan dapat kembali ke pilar awal, ia akan memenangkan permainan.

Untuk dapat meraih kemenangan tersebut, kesatria harus memilih langkah-langkah yang tepat. Salah satu algoritma yang dapat digunakan untuk melakukan pemilihan langkah-langkah tersebut adalah algoritma *backtracking*.

II. DASAR TEORI

A. Algoritma Backtracking

Algoritma *backtracking* merupakan algoritma yang berbasis pada *Depth-First-Search (DFS)* untuk mencari solusi persoalan secara lebih mangkus. *Backtracking*, yang merupakan perbaikan dari algoritma *brute-force*, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada. Dengan metode *backtracking*, kita tidak perlu memeriksa semua kemungkinan yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan. Akibatnya, waktu pencarian dapat dihemat. *Backtracking* alami dinyatakan dalam algoritma rekursif. Kadang-kadang disebutkan pula bahwa *backtracking* merupakan bentuk tipikal dari algoritma rekursif.

Istilah *backtracking* pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Selanjutnya, R.J. Walker, Golomb, dan Baumert menyajikan uraian umum tentang *backtracking* dan penerapannya pada berbagai persoalan. Saat ini algoritma *backtracking* banyak diterapkan untuk program *games* (seperti permainan *tic-tac-toe*, menemukan jalan keluar dalam sebuah labirin, catur, dll) dan masalah-masalah pada bidang kecerdasan buatan (*artificial intelligence*).

1. Properti Umum Metode *Backtracking*

Untuk menerapkan metode *backtracking*, properti berikut didefinisikan:

a. Solusi persoalan.

Solusi dinyatakan sebagai vektor dengan *n-tuple:*

$X = (x_1, x_2, \ldots, x_n)$, $x_i \in$ himpunan berhingga S_i. Mungkin saja $S_1 = S_2 = \ldots = S_n$.

Contoh: $S_i = \{0, 1\}$, $x_i = 0$ atau 1.

b. Fungsi pembangkit nilai x_k

Dinyatakan sebagai: *T(k)*.

T(k) membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.

c. Fungsi pembatas

Dinyatakan sebagai: $B(x_1, x_2, \ldots, x_k)$.

Fungsi pembatas menentukan apakah $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi. Jika ya, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika tidak, maka $(x_1, x_2, \ldots, x_k)$ dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi.

Fungsi pembatas tidak selalu dinyatakan sebagai fungsi matematis, ia dapat dinyatakan sebagai predikat yang bernilai *true* atau *false*, atau dalam bentuk lain yang ekivalen.

2. Pengorganisasian Solusi

Semua kemungkinan solusi dari persoalan disebut ruang solusi (*solution space*). Secara formal dapat dinyatakan, bahwa jika $x_i \in S_i$, maka

$S_1 \times S_2 \times \ldots \times S_n$

disebut ruang solusi. Jumlah anggota di dalam ruang solusi adalah $|S_1| \cdot |S_2| \cdot \ldots \cdot |S_n|$.

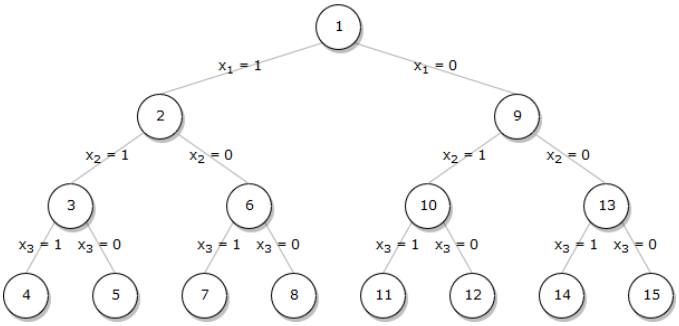
Sebagai gambaran, tinjau persoalan *Knapsack* 0/1 untuk n = 3. Solusi persoalan dinyatakan sebagai vektor (x_1, x_2, x_3) dengan $x_i \in \{0, 1\}$. Ruang solusinya adalah

{0, 1} x {0,1} x {0, 1} = {(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)}.

Dengan kata lain, pada persoalan *Knapsack* 0/1 dengan n = 3 terdapat $2^n = 2^3 = 8$ kemungkinan solusi, yaitu (0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), dan (1, 1, 1).

Penyelesaian dengan *exhaustive search* adalah dengan menguji setiap kemungkinan solusi. Setiap kemungkinan solusi diperiksa apakah ia memenuhi kendala (yakni menjadi solusi yang layak). Kalau jawabannya ya, lalu dihitung nilai fungsi objektifnya. Solusi layak yang memberikan nilai fungsi objektif yang maksimum (untuk kasus maksimasi) atau minimum (untuk kasus minimasi) merupakan solusi optimum.

Algoritma *backtracking* memperbaiki pencarian solusi secara *exhaustive search* dengan mencari solusi secara sistematis. Untuk memfasilitasi pencarian ini, maka ruang solusi diorganisasikan ke dalam struktur pohon. Tiap simpul pohon menyatakan status (*state*) persoalan, sedangkan sisi (cabang) dilabeli dengan nilai-nilai x_i. Lintasan dari akar ke daun menyatakan solusi yang mungkin. Seluruh lintasan dari akar ke daun membentuk ruang solusi. Pengorganisasian pohon ruang solusi diacu sebagai pohon ruang status (*state space tree*). Tinjau kembali persoalan *Knapsack* 0/1 untuk n = 3. Ruang solusinya diorganisasikan sebagai pohon ruang status pada Gambar 2. Lintasan dari 1 sampai 4 misalnya, menyatakan solusi $X = (1, 1, 1)$. Perhatikanlah bahwa simpul-simpul diberi urutan nomor sesuai dengan prinsip pencarian *DFS*. Metode *backtracking* menerapkan *DFS* dalam pencarian solusi.



Gambar 2. Ruang solusi untuk persoalan *Knapsack* 0/1 dengan n = 3

3. Prinsip Pencarian Solusi dengan Metode *Backtracking*

Di sini kita hanya akan meninjau pencarian solusi pada pohon ruang status yang dibangun secara dinamis. Langkah-langkah pencarian solusi adalah sebagai berikut:

a. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti metode pencarian mendalam (*DFS*). Simpul-simpul yang sudah dilahirkan dinamakan simpul hidup (*live node*). Simpul hidup yang sedang diperluas dinamakan simpul-E (*Expand-node*). Simpul dinomori dari atas ke bawah sesuai dengan urutan kelahirannya (berdasarkan prinsip *DFS*).

b. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" sehingga menjadi simpul mati (*dead node*). Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas (*bounding function*). Simpul yang sudah mati tidak akan pernah diperluas lagi.

c. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada simpul anak yang dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan *backtrack* ke simpul hidup terdekat (simpul orangtua). Selanjutnya simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun kembali sampai lintasan tersebut membentuk solusi.

d. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk *backtracking*.

Tinjau persoalan *Knapsack* 0/1 dengan instansiasi:

$n = 3$

$(w_1, w_2, w_3) = (35, 32, 25)$

$(p_1, p_2, p_3) = (40, 25, 50)$

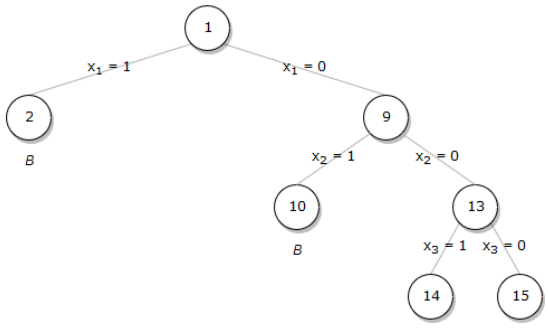
M = 30

Solusi dinyatakan sebagai $X = (x_1, x_2, x_3)$, $x_i \in \{0, 1\}$. Fungsi pembatas yang digunakan adalah kendala (*constraint*):

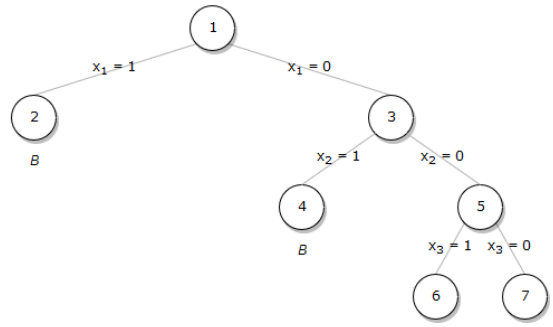
$$\sum_{i=1}^{k} w_i x_i \leq M$$

Jadi, jika vektor solusi (x_1, x_2, …, x_k) tidak memenuhi ketidaksamaan tersebut, maka lintasan di dalam pohon pencarian yang sisi-sisinya dilabeli dengan (x_1, x_2, …, x_k) tidak mengarah ke solusi sehingga lintasan tersebut tidak diperpanjang lagi. Sebaliknya, jika memenuhi, maka lintasan tersebut dapat diteruskan untuk dikembangkan.

Mulailah dari status awal (simpul akar) yaitu belum ada objek yang dimasukkan ke dalam *knapsack*. Beri nomor simpul akar ini dengan angka satu. Simpul 1 sekarang menjadi simpul hidup sekaligus simpul-E. Simpul 1 diperluas menjadi simpul 2 dan meng-*assign* sisi (1, 2) dengan $x_1 = 1$; sampai saat ini solusi adalah $X = (1, x_2, x_3)$. Solusi ini tidak layak karena berat objek 1 lebih besar daripada kapasitas *knapsack*. Oleh karena itu, simpul 2 dimatikan (pada Gambar 3 simpul 2 ditandai dengan *B*, yang artinya simpul itu mati karena tidak memenuhi kendala. Dengan demikian, anak-anak simpul 2 tidak pernah diperluas lagi). Lakukan *backtrack* ke simpul 1. Simpul 1 diperluas menjadi simpul 9 dan meng-*assign* sisi (1,9) dengan $x_1 = 0$; sampai saat ini solusi adalah $X = (0, x_2, x_3)$. Solusi ini layak sehingga simpul 9 dapat diperluas. Sekarang simpul 9 menjadi simpul-E. Simpul 9 diperluas menjadi simpul 10 dan meng-*assign* sisi (9, 10) dengan $x_2 = 1$; sampai saat ini solusi adalah $X = (0, 1, x_3)$. Solusi ini tidak layak karena berat objek 2 lebih besar daripada kapasitas *knapsack*. Oleh karena itu, simpul 10 dimatikan. Lakukan *backtrack* ke simpul 9. Simpul 9 diperluas menjadi simpul 13 dan meng-*assign* sisi (9, 13) dengan $x_2 = 0$; sampai saat ini solusi adalah $X = (0, 0, x_3)$. Solusi ini layak sehingga simpul 13 dapat diperluas menjadi simpul 14 dan meng-*assign* sisi (13, 14) dengan $x_3 = 1$; sampai saat ini solusi adalah $X = (0, 0, 1)$. Solusi ini layak, dan arena simpul 14 adalah simpul daun berarti kita sudah mencapai simpul solusi dengan $X = (0, 0, 1)$ dan $F = 50$. Ini adalah solusi terbaik sampai sejauh ini. Karena simpul 13 masih hidup, maka simpul 13 masih dapat diperluas menjadi simpul 15 dan men-*assign* sisi (13, 15) dengan $x_3 = 0$; sampai saat ini solusi adalah $X = (0, 0, 0)$. Solusi ini layak, dan karena simpul 15 adalah simpul daun berarti kita sudah mencapai simpul solusi dengan $X = (0, 0, 0)$ dan $F = 0$. Namun, karena total keuntungan solusi yang kedua lebih kecil dari total keuntungan solusi pertama, maka solusi optimumnya adalah $X = (0, 0, 1)$ dan $F = 50$.



Gambar 3. Pohon dinamis yang dibentuk selama pencarian untuk persoalan *Knapsack* 0/1 dengan $n = 3$, $w = (35, 32, 25)$, $p = (40, 25, 50)$



Gambar 4. Penomoran ulang simpul-simpul sesuai urutan pembangkitannya

4. Skema Umum Algoritma *Backtracking*

Di bawah ini disajikan skema umum algoritma *backtracking* dalam dua versi, versi rekursif dan versi iteratif. Skema dalam versi rekursif lebih tepat karena algoritma *backtracking* lebih alami dinyatakan dalam bentuk rekursif. Algoritma di bawah ini akan menghasilkan semua solusi.

a. Versi rekursif

```
procedure RunutBalikR(input k: integer)
{Mencari semua solusi persoalan dengan metode
backtracking; 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 (x[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
```

Pemanggilan prosedur pertama kali: RunutbalikR(1).

b. Versi iteratif

```
procedure RunutBalikI(input n: integer)
{Mencari semua solusi persoalan dengan metode
backtracking; skema iteratif
 Masukan: n, yaitu panjang vektor solusi
 Keluaran: solusi x = (x[1], x[2], …, x[n])
}
Deklarasi
  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
    endif
  endwhile      {k = 0}
```

Pemanggilan prosedur pertama kali: RunutBalikI(n).

Prosedur CetakSolusi adalah sebagai berikut:

```
procedure CetakSolusi(input x: TabelInt)
{Mencetak solusi persoalan
 Masukan: x[1], x[2], …, x[n]
 Keluaran: nilai x[1], x[2], …, x[n] tercetak ke
layar
}
Deklarasi
  k: integer
Algoritma:
  for k ← 1 to n do
    write(x[k])
  endfor
```

Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursi. Jika jumlah simpul dalam pohon ruang status adalah 2^n atau $n!$, maka untuk kasus terburuk, algoritma *backtracking* membutuhkan waktu dalam $O(p(n)2^n)$ atau $O(q(n)n!)$, dengan p(n) dan q(n) adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul.

B. Sirkuit Hamilton

Persoalan sirkuit Hamilton merupakan salah satu persoalan klasik. Persoalan ini bertujuan untuk menemukan perjalanan yang dapat mengunjungi semua simpul tepat satu kali dan kembali lagi ke simpul awal.

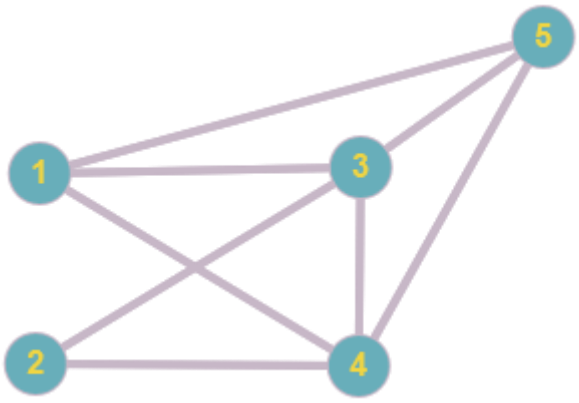
Diberikan graf terhubung $G = (V, E)$ dengan n buah simpul. Jumlah sisi yang dilalui dalam siklus Hamilton adalah n buah. Jika siklus Hamilton dimulai pada simpul $v_1 \in V$ dan simpul-simpul dalam G dikunjungi dalam urutan $v_1, v_2, \ldots, v_n, v_1$, maka sisi (v_i, v_{i+1}) dan $(v_n, 1) \in E$, $1 \le i \le n$. Jika perjalanan dimulai dari simpul 1, maka sirkuit Hamilton yang terdapat pada Gambar 5 diantaranya adalah:

1,3, 2, 4, 5, 1

1, 4, 2, 3, 5, 1

1, 5, 4, 2, 3, 1

1, 5, 3, 2, 4, 1



Gambar 5. Graf G dengan 5 buah simpul yang memiliki sirkuit Hamilton

Misalkan $X = (x_1, x_2, \ldots, x_n)$ adalah vektor solusi, yang dalam hal ini x_k menyatakan simpul i yang dikunjungi dalam sirkuit Hamilton. Misalkan sirkuit Hamilton dimulai dari simpul 1. Jadi, untuk $k = 1$, $x_1 = 1$. Untuk $1 < k < n$, maka x_k adalah sembarang simpul v yang berbeda dari $(x_1, x_2, \ldots, x_{k-1})$ dan v harus terhubung ke simpul x_{k-1}. Sedangkan x_n adalah simpul terakhir dan harus terhubung baik ke simpul x_{n-1} maupun simpul x_1.

C. Algoritma Backtracking untuk Sirkuit Hamilton

Masukan:

Matriks GRAF[1..n, 1..n] {n = jumlah simpul graf}

GRAF[i, j] = true jika ada sisi dari simpul i ke j

GRAF[i, j] = false jika tidak ada sisi dari simpul i ke j

Keluaran:

1. Tabel x[1..n], yang dalam hal ini, x[i] adalah simpul i yang dikunjungi dalam sirkuit Hamilton.

Algoritma:

1. Inisialisasi x[2..n] dengan 0, sedangkan x[1] diisi dengan 1 (karena diasumsikan siklus hamilton dimulai dari simpul 1).

   x[1] ← 1

   for i ← 2 to n do

     x[i] ← 0

   endfor

2. Panggil prosedur SirkuitHamilton (2)

```
{Kamus global}
Deklarasi
  const n = …  {jumlah simpul graf}
  type matriks = array[1..n, 1..n] of Boolean
  type tabel = array[1..n] of integer
  GRAF: matriks
  x = tabel
```

```
procedure SirkuitHamilton(input k: integer)
{Menemukan semua sirkuit Hamilton pada graf
terhubung. Sirkuit dimulai dari simpul 1
 Masukan: k adalah nomor simpul graf
 Keluaran: jika solusi ditemukan, solusi dicetak
ke piranti keluaran
}
Deklarasi
  stop: boolean
Algoritma:
  stop ← false
  while not stop do
    {tentukan semua nilai untuk x[k]}
    SimpulBerikutnya(k)       {isi x[k] dengan
simpul berikutnya}
    if x[k] = 0 then   {tidak ada simpul lagi,
habis}
      stop ← true
    else
      if k = n then    {seluruh simpul sudah
dikunjungi}
        CetakSolusi(x, n) {cetak sirkuit
Hamilton}
      else
        SirkuitHamilton(k + 1)        {cari
simpul berikutnya}
      endif
    endif
  endwhile     {stop}
```

Prosedur SimpulBerikutnya menentukan simpul yang berikutnya untuk membentuk sirkuit Hamilton.

```
procedure SirkuitBerikutnya(input k: integer)
{Menentukan simpul berikutnya untuk membentuk
simpul Hamilton
 Masukan: k
 Keluaran: nilai untuk x[k]
 K. Awal: x[1], x[2], …, x[k-1] adalah lintasan
yang terdiri atas k-1 simpul berbeda.
 K. Akhir: x[k] berisi simpul berikutnya dengan
nomor yang lebih tinggi yang:
     (i) belum muncul dalam x[1], x[2], …, x[k-1]
     (ii) terhubung oleh sebuah sisi ke x[k-1]
   Jika tidak memenuhi kedua kondisi itu, maka
x[k] = 0. Jika k = n, maka harus diperiksa apakah
x[k] terhubung ke x[1]
}
Deklarasi
  stop: boolean
  j : integer
Algoritma:
  stop ← false
  while not stop do
    x[k] ← (x[k] + 1) mod (n + 1)     {simpul
berikutnya}
    if x[k] = 0 then
      stop ← true
    else
      if GRAF[x[k-1], x[k]]   {ada sisi dari x[k]
ke x[k-1]} then
        {periksa apakah x[k] berbeda dengan
simpul-simpul x[1], x[2], …, x[k-1]}
        sama ← false
        j ← 1
        while (j <= k - 1) and (not sama) do
          if x[j] = x[k] then
            sama = true
          else
            j ← j + 1
          endif
        endwhile       {j > k - 1 or sama}
        if not sama    {berarti simpul x[k]
berbeda} then
          if (k < n)   {belum semua simpul
dikunjungi} or {atau}
             ((k = n) and (GRAF[x[n], 1])) {ada
sisi dari x[n] ke x[1]}
          then
            stop ← true
          endif
        endif
    endif
  endwhile     {stop}
```

III. RUMUSAN MASALAH

Permasalahan yang harus dipecahkan adalah persoalan menemukan urutan langkah yang tepat sehingga kesatria pada permainan *Greedy Knight* dapat memenangkan permainan. Untuk memenangkan permainan, kesatria harus mengumpulkan seluruh koin emas yang ada dan dapat kembali ke pilar awal.

Jumlah dari koin emas yaitu tiga belas keping. Masing-masing koin emas terletak pada pilar-pilar yang berbeda. Posisi dari setiap pilar ditunjukkan oleh Gambar 6. Penomoran diberikan untuk memudahkan dalam menyelesaikan persoalan.

14	11	7	3
13	10	6	2
12	9	5	1
	8	4	

Gambar 6. Posisi pilar pada permainan *Greedy Knight*

Pada permainan ini, kesatria akan menunggai kuda dan bergerak seperti langkah dalam permainan catur, yaitu dua blok secara vertikal lalu satu blok secara horizontal, atau dua blok secara horizontal lalu satu blok secara vertikal. Langkah dari kesatria tersebut akan terlihat seperti membentuk huruf L.

Dengan menggunakan langkah tersebut, kesatria dapat berpindah dari satu pilar ke pilar lainnya. Tabel 1 menunjukkan keterhubungan antar pilar jika menggunakan setiap kemungkinan langkah tersebut.

TABEL I. KETERHUBUNGAN ANTAR PILAR

No. Pilar	Pilar yang Terhubung
1	7, 8, 10
2	4, 9, 11
3	5, 10
4	2, 10, 12
5	3, 11, 13
6	8, 12, 14
7	1, 9, 13
8	1, 6, 13
9	2, 7, 14
10	1, 3, 4
11	2, 5, 12
12	4, 6, 11
13	5, 7, 8
14	6, 9

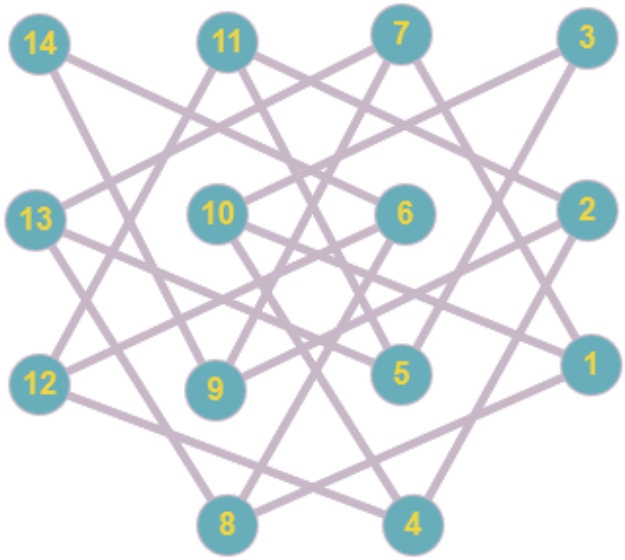
Selain menggunakan langkah khusus tersebut, permainan ini juga memiliki tantangan lain, yaitu kesatria hanya dapat mengunjungi pilar-pilar selain pilar awal tepat satu kali. Hal ini disebabkan karena pilar-pilar selain pilar awal rentan runtuh.

Tantangan untuk menemukan perjalanan sehingga dapat mengunjungi semua pilar tepat satu kali tersebut sama halnya dengan tantangan pada persoalan sirkuit Hamilton. Oleh karena itu, dengan adanya kesamaan tersebut, persoalan pada permainan ini dapat ditransformasi menjadi persoalan sirkuit Hamilton.

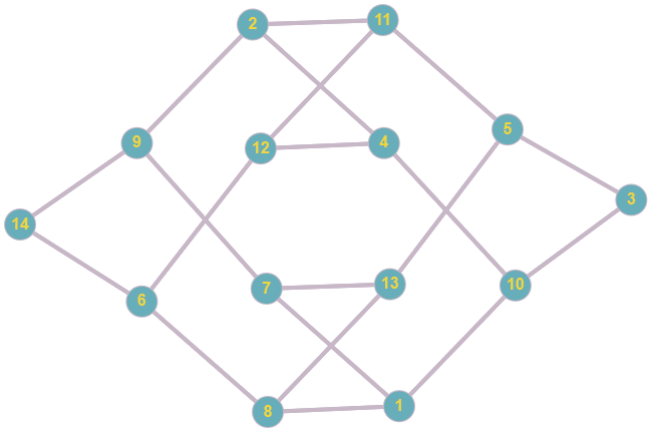
IV. PEMECAHAN MASALAH

A. *Pembentukan Sirkuit Hamilton*

Berdasarkan Tabel 1, keterhubungan antar pilar, sirkuit Hamilton dapat dibentuk. Gambar 7 menunjukkan sirkuit Hamilton tersebut. Dan untuk mempermudah memperhatikan sirkuit, dilakukan pengaturan posisi simpul. Hal ini ditunjukkan pada Gambar 8.



Gambar 7. Sirkuit Hamilton pada permainan *Greedy Knight*

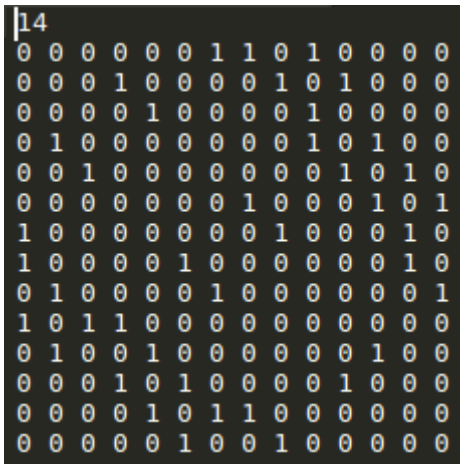


Gambar 8. Sirkuit Hamilton pada *Greedy Knight* dengan pengaturan posisi simpul

B. *Penerapan Algoritma Backtracking*

Dengan menggunakan algoritma *backtracking* untuk sirkuit Hamilton yang telah dipaparkan sebelumnya, solusi dari permainan *Greedy Knight* dapat diperoleh. Masukan yang diberikan berupa graf terhubung G dengan 14 buah simpul. Masukan untuk program yang dibuat dengan menerapakan algoritma tersebut ditunjukkan pada gambar 9. Baris pertama menyatakan jumlah simpul. Baris-baris berikutnya menyatakan

matriks keterhubungan antar simpul. Matriks keterhubungan antar simpul juga dapat dilihat pada Gambar 10. Baris dari matriks menyatakan nilai i dan kolom dari matriks menyatakan nilai j.



Gambar 9. Masukan untuk program

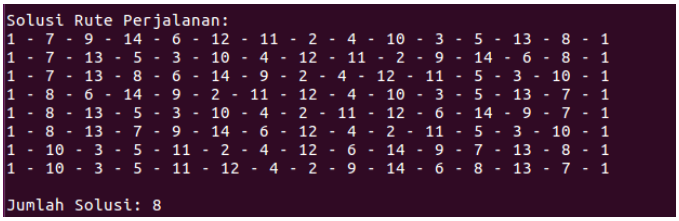
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	0	0	0	0	1	1	0	1	0	0	0	0
2	0	0	0	1	0	0	0	0	1	0	1	0	0	0
3	0	0	0	0	1	0	0	0	0	1	0	0	0	0
4	0	1	0	0	0	0	0	0	0	1	0	1	0	0
5	0	0	1	0	0	0	0	0	0	0	1	0	1	0
6	0	0	0	0	0	0	0	1	0	0	0	1	0	1
7	1	0	0	0	0	0	0	0	1	0	0	0	1	0
8	1	0	0	0	0	1	0	0	0	0	0	0	1	0
9	0	1	0	0	0	0	1	0	0	0	0	0	0	1
10	1	0	1	1	0	0	0	0	0	0	0	0	0	0
11	0	1	0	0	1	0	0	0	0	0	0	1	0	0
12	0	0	0	1	0	1	0	0	0	0	1	0	0	0
13	0	0	0	0	1	0	1	1	0	0	0	0	0	0
14	0	0	0	0	1	1	0	0	1	0	0	0	0	0

Gambar 10. Matriks masukan keterhubungan antar simpul

G[i, j] = 1 = true jika ada sisi simpul i ke j

G[i, j] = 0 = false jika tidak ada sisi simpul i ke j

Berdasarkan masukan tersebut, program akan menghasilkan keluaran berupa sejumlah jalur perjalanan untuk meraih kemenangan pada permainan *Greedy Knight*. Gambar 11 menunjukkan keluaran dari program. Dan gambar 12 menunjukkan tampilan saat memenangkan permainan *Greedy Knight*.



Gambar 11. Keluaran dari program

Gambar 12. Tampilan saat memenangkan permainan *Greedy Knight* [1]

V. KESIMPULAN

Persoalan pada permainan *Greedy Knight* dapat ditransformasi menjadi persoalan klasik, yaitu persoalan sirkuit Hamilton. Dengan melakukan transformasi, persoalan pada permainan dapat lebih mudah dipahami dan diselesaikan.

Penggunaan algoritma *backtracking* dalam penyelesaian persoalan memberikan solusi yang mangkus. Dari implementasi yang telah dilakukan, terdapat delapan solusi rute perjalanan yang dapat dipilih untuk meraih kemenangan pada permainan *Greedy Knight*.

UCAPAN TERIMA KASIH

Pertama, penulis ingin mengucapkan puji dan syukur kepada Tuhan Yang Maha Esa karena dengan rahmat dan karunia-Nya penulis dapat menyelesaikan makalah dengan judul "Penerapan Algoritma *Backtracking* pada Permainan *Greedy Knight*" dengan baik. Penulis juga berterima kasih kepada para dosen pengajar mata kuliah IF2211 Strategi Algoritma, Dr. Masayu Leylia Khodra, Dr. Nur Ulfa Maulidevi, S.T, M.Sc., dan Dr. Ir. Rinaldi Munir, M.T., atas bimbingannya selama ini dalam mengajar dan memberikan ilmu sehingga penulis mampu untuk membuat makalah ini. Penulis juga berterima kasih kepada rekan-rekan yang telah memberikan semangat dan dorongan kepada penulis.

REFERENSI

[1] http://www.novelgames.com/en/greedy/. Diakses pada 12 Mei 2018.

[2] Munir, Rinaldi. 2009. Diktat Kuliah IF2211 Strategi Algoritma. Bandung.

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, 13 Mei 2018

Ahmad Fauzul Yogiandra – 13513059