Penerapan Algoritma Backtracking untuk Menyelesaikan Permainan Hashiwokakero

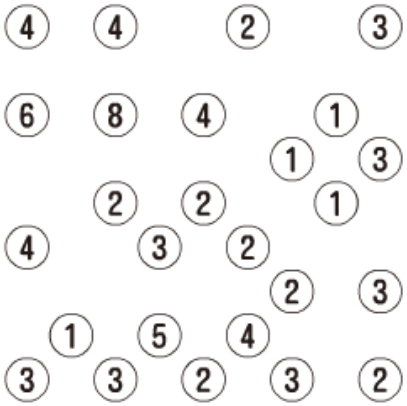
Irfan Kamil – 13510001[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13510001@std.stei.itb.ac.id

***Abstrak*—Makalah ini berisi tentang pembahasan solusi permainan hashiwokakero dengan algoritma *backtracking*. Pada makalah ini terdapat pseudocode yang dirancang dari solusi permainan hashiwokakero dan analisis pada algoritma yang dibuat apakah mangkus atau tidak.**

***Kata Kunci*—hashiwokakero solver, *backtracking*, pseudocode, kompleksitas waktu**

I. PENDAHULUAN

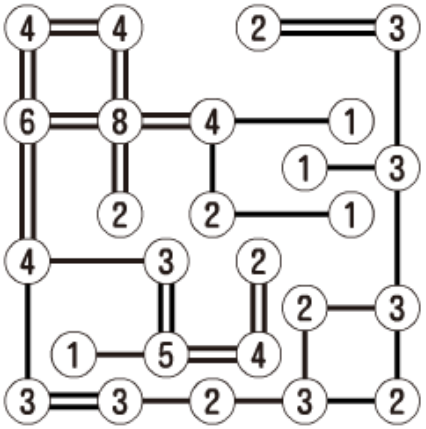
Hashiwokakero[1] adalah permainan yang dibuat oleh Nikoli. Pada permainan ini, pemain harus membuat beberapa jembatan yang menghubungkan dua buah pulau. Pulau digambarkan sebagai angka yang dilingkari dan jembatan adalah garis lurus yang menghubungkan dua buah pulau.



Gambar 1 Contoh persoalan Hashiwokakero

Angka pada lingkaran menyatakan jumlah jembatan yang menghubungakan lingkaran tersebut dengan tetangganya yang harus dipenuhi. Jumlah maksimal jembatan yang menghubungkan dua buah pulau adalah dua jembatan. Dua buah jembatan tidak boleh saling memotong.

Pada akhirnya, seluruh pulau akan saling terhubung. Tidak boleh ada himpunan pulau yang terisolasi, tidak terhubung dengan pulau lain.



Gambar 2 Contoh solusi Hashiwokakero

II. DASAR TEORI

Backtracking adalah algoritma yang berbasis pada DFS untuk mencari solusi yang lebih mangkus. Secara sistematis, algoritma *backtracking* mencari solusi persoalan diantara kemungkinan solusi yang ada. Algoritma *Backtracking* memangkas simpul yang tidak mengarah ke solusi. [4]

A. Solusi Persoalan

Solusi dinyatakan sebagai vektor dengan n-dimensi. n adalah jumlah kemungkinan ruang nilai.
X : vektor solusi
x_i : nilai pada komponen ke-i
S_i : himpunan nilai yang mungkin pada komponen ke-i

$X = (x_1, x_2, \ldots, x_n), x_i \in S_i$

B. Fungsi Pembangkit

Fungsi pembangkit membangkitkan nilai pada suatu komponen vektor solusi. Fungsi pembangkit dinyatakan sebagai T(k).

C. Fungsi Pembatas

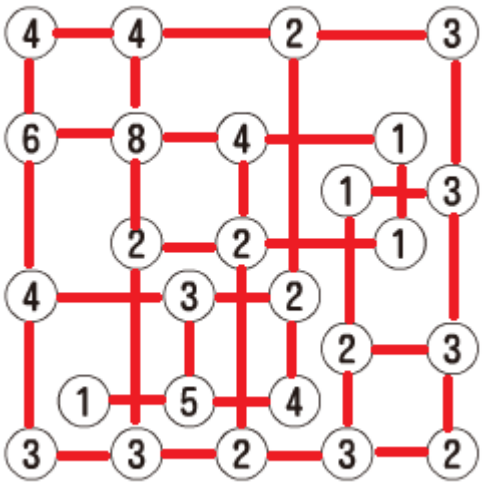
Fungsi pembatas membatasi solusi yang tidak mungkin menjadi solusi. Fungsi ini bernilai *true* apabila suatu vektor yang diperoleh mengarah ke solusi dan bernilai *false*

apabila tidak. Jika bernilai *true*, pembangkitan nilai untuk x_{k+1} dilanjutkan. Jika bernilai *false*, vektor tersebut dibuang dari kemungkinan solusi dan dilakukan *backtrack* pada x_k.

III. Penerapan Algoritma Backtracking

Setiap pulau dianggap sebagai *node* dan setiap hubungan *node* yang mungkin dianggap sebagai *edge*. *Edge* ini dapat berari tidak ada jembatan, atau ada jembatan yang menghubungkan dua buah *node*. *Edge* yang bukan merupakan jembatan diberi nilai 0, *edge* yang mengandung jembatan diberi nilai 1 atau 2, sesuai jumlah jembatan. Setiap *edge* memiliki status apakah sudah diproses atau belum.

X : nilai seluruh *edge* yang merupakan solusi
x_i : nilai pada *edge* ke-i
S_i : {0, 1, 2}
n : jumlah *edge* pada permainan
k : jumlah *edge* yang sudah diproses



Gambar 3 Edge yang mungkin pada solusi

A. Solusi Persoalan

Solusi didapatkan apabila k sama dengan n dan apabila vektor yang didapat (x_1, x_2, …, x_k) dikomposisikan pada fungsi pebatas bernilai *true*.

B. Fungsi Pembangkit

Fungsi pembangkit akan membangkitkan nilai 0 pada *edge* ke-k dan membuat *edge* ke-k telah diproses. Fungsi pembangkit dipanggil ketika vektor yang didapat (x_1, x_2, …, x_{k-1}) mengarah pada solusi. Maka dari itu dibangkitkan x_k dan dimasukkan ke dalam vektor.

C. Fungsi Pembatas

Fungsi pembatas didefinisikan sebagai aturan yang harus dipenuhi pada permainan hashiwokakero. Berikut ini aturan yang harus dipenuhi.

1. Setiap *node* memiliki jumlah jembatan yang sama dengan nilai *node*

Cara melakukan pengecekan ini adalah dengan meninjau dua buah *node* bertetangga yang dihubungkan oleh *edge* ke-k.

Untuk kedua *node*:

- Apabila semua *edge* pada *node* tersebut sudah diproses dan total nilai *edge* tersebut sama dengan nilai *node* tersebut, kriteria ini terpenuhi.
- Apabila semua *edge* pada *node* tersebut sudah diproses dan total nilai *edge* tersebut tidak sama dengan nilai *node* tersebut, kriteria ini tidak terpenuhi.
- Apabila ada *edge* yang belum diproses, kalikan dengan 2 jumlah *node* yang belum diproses lalu tambahkan dengan nilai *node* yang sudah diproses. Apabila nilai ini dibawah nilai *node*, kriteria ini tidak terpenuhi. Apabila nilai ini diatas sama dengan nilai *node*, kriteria ini terpenuhi.
- Selain syarat di atas, kriteria tidak terpenuhi

2. Tidak ada suatu jembatan yang memotong jembatan lain

Cara melakukan pengecekan ini adalah dengan membandingkan *edge* ke-k dengan semua *edge* sebelumnya

Untuk setiap *edge* ke-1 hingga *edge* ke-k-1:

- Apabila *edge* berpotongan dengan *edge* ke-k, kriteria tidak terpenuhi

3. Setiap *node* harus tercipta hubungan dengan seluruh *node*

Cara melakukan pengecekan ini adalah dengan mengambil salah satu *node* dari *edge* ke-k dan menelusuri apakah dari *node* tersebut dapat mengunjungi seluruh *node*. Apabila seluruh *node* dapat dikunjungi, kriteria ini terpenuhi. Ketika *node* yang telah dikunjungi masih memiliki *edge* yang belum diproses, asumsikan kriteria ini masih terpenuhi karena bisa saja *edge* yang belum diproses menghubungkan *node* yang telah dikunjungi dengan seluruh *node*. Apabila *node* tersebut tidak dapat mengunjungi seluruh *node* dan semua node yang telah dikunjungi, *edge*-nya sudah diproses, kriteria ini tidak terpenuhi.

D. Backtrack

Aksi backtrack dilakukan apabila vektor yang sudah didapatkan menyalahi fungsi pembatas. Hal yang dilakukan pada backtrack ini adalah:

Misal didapatkan vektor (x_1, x_2, …, x_k)

- Nilai pada x_k ditambahkan dengan 1
- Apabila $x_k \notin S_k$, *edge* ke-k diubah menjadi belum diproses lalu kurangi nilai k dengan 1. Jika $k > 0$, lakukan backtrack pada vektor (x_1, x_2, …, x_k). Jika $k = 0$ maka solusi tidak ditemukan.

IV. Pseudocode Penyelesaian Permainan Hashiwokakero

```
function BATAS1(k:integer)->boolean
{true jika e[k] memenuhi kriteria
pertama, false jika tidak}

DEKLARASI
  valid : boolean
  nProcessed, nUnprocessed : integer
  tProcessed : integer

ALGORITMA
  valid <- true
  for each node on e[k] as n
  nProcessed <- 0
  nUnProcessed <- 0
  tProcessed <- 0
  tUnProcessed <- 0
  for each edge on n as ez
    if IsProcessed(ez)
      nProcessed <- nProcessed + 1
      tProcessed <- tProcessed +
Value(ez)
    else
      nUnprocessed <- nUnprocessed + 1
    end if
  end for each
  if (nUnprocessed = 0 and tProcessed
= Value(n))
    valid = true
  else
  if (nUnprocessed = 0 and tProcessed
<> Value(n))
    -> false
  else
  if (nUnprocessed*2 + tProcessed >=
Value(n))
    valid = true
  else
    -> false
  end if
  end for each
  -> valid
```

```
function BATAS2(k:integer)->boolean
{true jika e[k] memenuhi kriteria
kedua, false jika tidak}

DEKLARASI
  valid : boolean
  nProcessed, nUnprocessed : integer
  tProcessed : integer

ALGORITMA
  valid <- true
  for each edge as ez
    if (ez <> e[k] and
IsProcessed(ez))
      if CROSSING(ez,e[k])
        -> false
      end if
    end if
  end for each
  -> true

function CROSSING(e1 : edge, e2 :
edge) -> boolean
{true jika e1 berpotongan dengan e2,
false jika tidak}

DEKLARASI
  px1, py1, px2, py2, px3, py3, px4,
py4 : integer
  d, pxi, pyi : integer

ALGORITMA
  px1 <- e1.vertices.first.x
  py1 <- e1.vertices.first.y
  px2 <- e1.vertices.second.x
  py2 <- e1.vertices.second.y
  px3 <- e2.vertices.first.x
  py3 <- e2.vertices.first.y
  px4 <- e2.vertices.second.x
  py4 <- e2.vertices.second.y
  d <- (px1-px2)*(py3-py4) - (py1-
py2)*(px3-px4)
  if (d = 0)
    -> false
  end if
  pxi <- ((px3-px4)*(px1*py2-py1*px2)-
(px1-px2)*(px3*py4-py3*px4))/d;
  pyi <- ((py3-py4)*(px1*py2-py1*px2)-
(py1-py2)*(px3*py4-py3*px4))/d;
  -> (px1 < pxi and pxi < px2 or py1 <
pyi and pyi < py2 or
    px1 > pxi and pxi > px2 or py1 >
pyi and pyi > py2) and
    (px3 < pxi and pxi < px4 or py3 <
pyi and pyi < py4 or
      px3 > pxi and pxi > px4 or py3 >
pyi and pyi > py4)
```

```
function BATAS3(k:integer)->boolean
{true jika e[k] memenuhi kriteria
pertama, false jika tidak}

DEKLARASI
  n : Node
  visited : array of boolean
  s : Stack of Node

ALGORITMA
  for each visited as vi
    vi <- false
  end for each
  n <- e[k].verticies.first
  s.push(n)
  while not IsEmpty(s)
    n <- s.pop()
    visited[n.index] <- true
    for each edge on n as ez
      if (IsProcessed(ez))
        if (not
visited[Neighbor(n,ez)])
          s.push(Neighbor(n,ez))
        end if
      else
```

```
        -> true
      end if
    end for each
  end while
    -> IsAllNodeVisited(visited)
```

```
procedure Hashi_Solve(input n : array
of Node, input e : array of Edge)
{ Menyelesaikan puzzle hashiwokakero
yang diberikan }
DEKLARASI
  k, N : integer

ALGORITMA
  k <- 1
  N <- 0
  for each e as ez
  {hitung jumlah edge yang mungkin}
    N <- N + 1
  end for each
  e[1].value <- 0
  {inisialisasi angka dengan 0}
  e[1].processed <- true
  while k > 0 do
    while (e[k].value <= 2) and (not
BATAS1(k) or not BATAS2(k) or not
BATAS3(k)) do
      {periksa apakah edge menuju
solusi}
      e[k].value <- e[k].value + 1
    end while
    {e[k].value > 2 or BATAS1(k) and
BATAS2(k) and BATAS3(k)}

    if e[k].value <= N then
    {mengarah ke solusi}
      if k = N then
      {apakah solusi sudah lengkap?}
        CetakSolusi(n,e)
      else
        k <- k + 1
        {pergi ke baris berikutnya}
        e[k].value <- 0
        {inisialisasi angka dengan 0}
        e[k].processed <- true
      end if
    else
      e[k].processed <- false
      k <- k - 1
  { runut-balik ke baris sebelumnya }
    end if
  end while
  { k = 0 }
```

V. Analisis dan Pembahasan

A. Kompleksitas Waktu Algoritma Penyelesaian Permainan Hashiwokakero

Tinjau nilai pada *edge* sebagai ruang yang bisa dicoba nilainya. Setiap *edge* dapat bernilai 0, 1, atau 2. Jadi, jumlah kemungkinan pada suatu *edge* adalah 3.

Kemungkinan ini dicoba untuk setiap *edge* yang mungkin. Jadi, jumlah percobaan adalah:

3^n (1)

dengan n adalah jumlah *edge*.

Pada masing-masing percobaan, diterapkan fungsi pembatas.

- Fungsi pembatas pertama mencoba maksimal 7 *edge* di sekitar 2 buah *node*
  O(n) = c (2)
- Fungsi pembatas kedua mengecek *edge* ke-k dengan *edge* 1 sampai k-1
  O(n) = n (3)
- Fungsi pembatas ketiga menelusuri seluruh *node* dengan melalui *edge* yang sudah diproses
  O(n) = n (4)

Jadi, dengan mengalikan jumlah percobaan dengan fungsi pembatas

$O(n) = (c + n + n) \cdot 3^n = n \cdot 3^n$ (5)

B. Tipe Persoalan Algoritma Penyelesaian Permainan Hashiwokakero

Tipe persoalan algoritma penyelesaian permainan hashiwokakero adalah NP-Complete[2]. Konsekuensi dari tipe NP-Complete adalah tidak terdapat solusi dengan kompleksitas waktu polinomial. Jadi, algoritma *backtracking* adalah algoritma yang mangkus untuk persoalan ini.

C. Validitas Solusi yang Dihasilkan

Selama percobaan, dari 5 contoh kasus yang diselesaikan, solusi yang ditampilkan algoritma ini valid. Algoritma ini seolah-olah mencoba semua kemungkinan seperti *brute force* dengan tidak mencoba yang sudah pasti gagal[4]. Jadi, untuk setiap puzzle yang memiliki solusi, algoritma ini dapat mencari solusi dari puzzle tersebut.

D. Perbaikan pada Algoritma

Algoritma *backtracking* dapat menyelesaikan persoalan hashiwokakero. Untuk mempercepat pencarian solusi, dapat diterapkan aturan heuristik untuk memproses suatu edge yang nilainya sudah dapat dipastikan dari *puzzle* yang tersedia.

Beberapa teknik heuristik yang dapat dilakukan adalah[3]:

1. Jumlah maksimal nilai semua edge pada suatu *node* sama dengan nilai *node*
2. *Node* dengan nilai 1 memiliki satu buah edge yang mungkin
3. Teknik jumlah tetangga yang mungkin
4. Teknik *node* sisa
5. Teknik isolasi

Setelah aturan heuristik permainan diterapkan, bisa jadi sudah tercipta solusi atau belum. Misalkan didapatkan vektor $(x_1, x_2, \ldots, x_k)$ dari hasil heuristik. Vektor ini dipastikan sudah mendekati solusi karena memenuhi aturan permainan. Pada vektor ini tidak diperlukan proses *backtracking*. Kemudian, lakukan proses *backtracking*

pada *edge* yang belum diproses.

V. KESIMPULAN

Algoritma *backtracking* dapat diterapkan untuk menyelesaikan permainan hashiwokakero. Algoritma ini mangkus untuk penerapan masalah ini.

REFERENSI

[1] http://www.nikoli.com/en/puzzles/hashiwokakero/ akses pada 21 Desember 2012

[2] D. Anderson, "HASHIWOKAKERO Is NP-Complete". Aarhus: Department of Computer Science Aarhus University.

[3] R. F. Malik, "Solving Hashiwokakero Puzzle Game with Hashi Solving Techniques and Depth First Search", *Buletin Teknik Elektro dan Informatika Unsri*, 2012, pp. 61-68

[4] R. Munir, "Diktat Kuliah IF3051 Strategi Algoritma", Bandung : STEI ITB, 2009.

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, 21 Desember 2012

Irfan Kamil
(13510001)