Penerapan Algoritma Runut-Balik (*Backtracking*) pada Permainan Nurikabe

Putri Amanda Bahraini

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
e-mail: if14041@students.if.itb.ac.id

ABSTRAK

Algoritma Runut-Balik (*Backtracking*) merupakan algoritma yang banyak digunakan pada pencarian solusi dari permainan-permainan yang mengandalkan logika. Algoritma ini merupakan pengembangan lebih lanjut dari algoritma DFS (*Depth First Search*) dan sudah terbukti dapat menghasilkan solusi dalam waktu yang lebih cepat dari algoritma pendahulunya. Salah satu permainan yang dapat dipecahkan dengan algoritma Runut-Balik (*Backtracking*) adalah Nurikabe, sebuah permainan logika yang dapat digolongkan sebagai permainan NP-Problem. Inti dari permainan ini adalah mengidentifikasi penentuan kotak berwarna hitam dan kotak berwarna putih dengan menggunakan petunjuk yang diberikan pada awal permainan.

Kata kunci: *Backtracking*, *Depth First Search*, *Nurikabe, NP-Problem*

1. PENDAHULUAN

Nurikabe adalah salah satu jenis permainan teka-teki pengasah logika yang akhir-akhir ini populer dimainkan oleh masyarakat dunia. Permainan ini dikeluarkan oleh **Nikoli**, perusahaan **Jepang** yang juga berjasa menemukan permainan **Sudoku**.

Permainan Nurikabe ini dapat digolongkan sebagai masalah **NP-Complete**, yaitu masalah yang jumlah waktu komputasi pencarian solusinya tidak lebih besar daripada fungsi polinomial n dari masalah itu sendiri. Bila sebuah algoritma dapat diimplementasikan untuk pencarian solusi sebuah masalah NP-Complete, berarti algoritma ini juga dapat digunakan untuk pencarian solusi masalah-masalah NP-Complete lainnya.

Pada makalah ini, penulis akan mencoba untuk mengimplementasikan **algoritma Runut-Balik** (*Backtracking*) pada pencarian solusi permainan Nurikabe. Algoritma Runut-Balik (*Backtracking*) ini dipilih oleh penulis karena dinilai sebagai algoritma paling efektif pada pencarian solusi masalah permainan logika pada umumnya.

2. NURIKABE

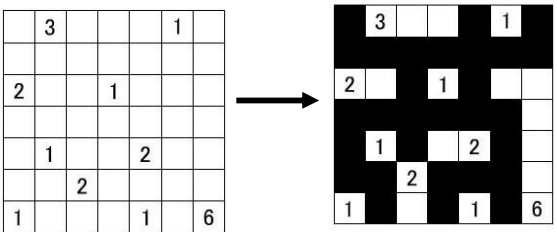
2.1 Pengenalan Singkat Nurikabe

Nurikabe, atau yang dikenal juga dengan nama *Islands in the Stream* dan *Cell Structure*, adalah sebuah permainan logika yang diciptakan oleh perusahaan **Jepang** bernama **Nikoli** pada tahun 1991.

Nama 'Nurikabe' itu sendiri diambil dari sebuah cerita rakyat Jepang, yaitu makhluk halus yang menjelma menjadi sebuah tembok tak terlihat. Nurikabe si makhluk halus ini, sering memblokir jalan dengan tujuan menyesatkan pejalan kaki di malam hari.

Permainan Nurikabe dimainkan pada sekumpulan kotak berbentuk matriks **N x N**. Pada awal permainan, beberapa dari kotak –kotak tersebut sudah terlebih dahulu diisi oleh bilangan bulat positif (lebih besar dari 0).

Inti dari permainannya adalah, pemain ditugaskan untuk mewarnai beberapa kotak yang kosong (tidak diisi oleh angka) dengan **warna hitam**, seperti pada gambar di bawah ini :



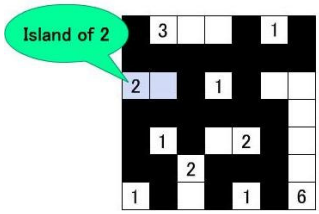
Gambar 1. Matriks kondisi awal dan akhir dari permainan Nurikabe

Umumnya, kotak hitam diumpamakan sebagai 'aliran sungai' (*stream*) dan kotak putih diumpamakan sebagai 'pulau' (*island)*. Penentuan 'pulau' dan 'aliran sungai' pada permainan ini harus dilakukan dengan mematuhi beberapa *constraint* yang akan dijelaskan pada bagian selanjutnya.

2.2 Peraturan Dalam Nurikabe

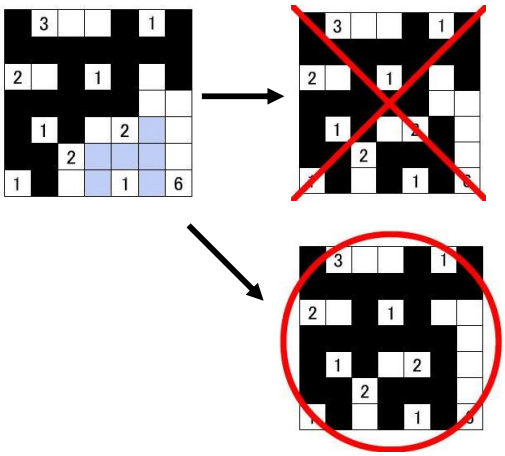
Berikut adalah peraturan-peraturan yang harus dipatuhi pada permainan Nurikabe :

1. Tidak diperbolehkan untuk mewarnai kotak yang sudah terisi oleh angka.
2. Angka-angka yang tertera pada matriks menandakan berapa jumlah kotak putih yang ada di sekitarnya ditambah kotak angka itu sendiri, contohnya dapat dilihat pada gambar di bawah ini :



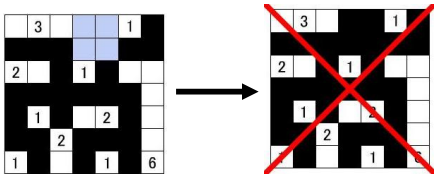
Gambar 2. Angka '2' berarti 2 kotak pulau

3. Kotak-kotak hitam harus saling terhubung satu sama lain sehingga membentuk sebuah 'aliran sungai' yang tak terputus.



Gambar 3. Tidak diperbolehkan adanya lebih dari satu aliran sungai

4. Kotak-kotak hitam tidak membentuk persegi berukuran 2x2 atau lebih besar dari itu.



Gambar 4. Tidak diperbolehkan adanya kumpulan kotak-kotak hitam yang membentuk persegi 2x2 atau lebih

2.3 Langkah Penyelesaian Nurikabe

Berikut adalah standar langkah-langkah pencarian solusi untuk permainan Nurikabe :

1. Isi semua kotak kosong yang bersisian (atas, bawah, kiri, kanan) dengan kotak berisi angka '1'.
2. Isi semua kotak kosong yang terjepit di antara kotak hitam dan tidak terhubung dengan angka manapun dalam matriks tersebut.
3. Tandai dengan tanda dot ● semua kotak-kotak yang bila dijadikan kotak hitam, maka akan membentuk suatu persegi.
4. Dengan bantuan tanda dot ● tersebut dan angka-angka pada matriks, cobalah tentukan kotak mana yang merupakan 'pulau' dan mana yang merupakan 'aliran sungai'.
5. Periksalah apakah kotak-kotak hitam sudah terhubung satu sama lain hingga membentuk satu aliran sungai.

3. ALGORITMA RUNUT-BALIK (*BACKTRACKING*)

3.1 Pengenalan Singkat Algoritma Runut Balik (*Backtracking*)

Runut-Balik (*Backtracking*) adalah algoritma komputasi yang merupakan hasil pengembangan dari algoritma DFS (*Depth First Search*) dengan tujuan untuk lebih memangkuskan proses pencarian solusi. Dengan menggunakan algoritma Runut-Balik (*Backtracking*), kita dapat menghemat waktu pencarian solusi, karena algoritma ini memungkinkan program untuk tidak memeriksa seluruh alternatif langkah-langkah pencarian solusi, hanya langkah-langkah yang memang dinilai mengarah pada solusi saja yang memang akan dipertimbangkan.

Saat ini, algoritma Runut-Balik (*Backtracking*) banyak digunakan pada pencarian solusi *game*s dan masalah-masalah pada bidang kecerdasan buatan (*artificial intelligence*)

3.2 Properti Umum Algoritma Runut-Balik (*Backtracking*)

Berikut adalah properti umum pada algoritma Runut-Balik (*Backtracking*) :

1. **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

2. **Fungsi pembangkit nilai x_k (*T(k)*)**
   $T(k)$ berfungsi untuk membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.
3. **Fungsi pembatas** (pada beberapa persoalan, fungsi ini disebut fungsi kriteria)
   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 benar, 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.

3.3 Prinsip Algoritma Runut-Balik (*Backtracking*)

Berikut adalah Prinsip algoritma Runut-Balik (*Backtracking*) :

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan pencarian DFS (*Depth First Search*). Simpul-simpul yang sudah dilahirkan dinamakan **simpul hidup** (*live node*). Simpul hidup yang *sedang* diperluas dinamakan **simpul-E** (*Expand-node*).
2. 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.
3. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan runut-balik ke simpul hidup terdekat (simpul orangtua). Selanjutnya simpul ini menjadi simpul-E yang baru.
4. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

3.4 Skema Umum Algoritma Runut-Balik (*Backtracking*)

Ada 2 skema yang dapat digunakan dalam implementasi algoritma Runut-Balik (*Backtracking*), yaitu :

1. Skema rekursif, dan
2. Skema iteratif

Pada pencarian solusi permainan Nurikabe ini, penulis hanya akan membuat algoritma Runut-Balik (*Backtracking*) dengan skema iterative saja.

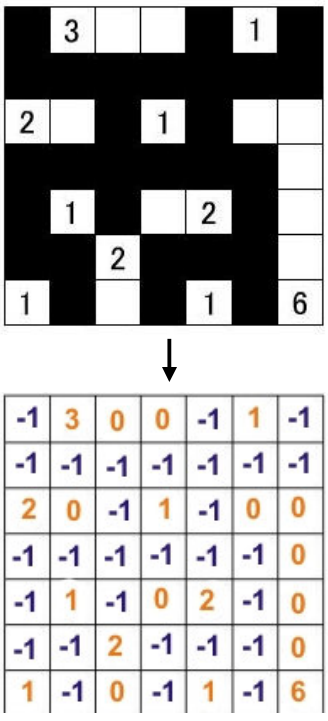
4. PENCARIAN SOLUSI NURIKABE DENGAN IMPLEMENTASI ALGORITMA RUNUT BALI K (*BACKTRACKING*)

4.1 Struktur Data

Matriks pada permainan Nurikabe direpresentasikan dalam bentuk *rectangular array* (array 2 dimensi). Dimana elemen dari *rectangular array* tersebut adalah sebuah integer yang berisi 3 pengelompokan angka, yaitu :

1. **Bilangan positif [1..N]** untuk mewakili angka yang memprediksikan jumlah pulau.
2. ***Integer* '0'** digunakan untuk merepresentasikan area kotak putih / island
3. ***Integer* '-1'** untuk area kotak hitam/*stream*.

Atau untuk lebih jelasnya, dapat dilihat pada gambar berikut ini :



Gambar 5. Representasi matriks Nurikabe pada program

Sedangkan untuk tabel solusi, akan dimasukkan dalam sebuah *table of point*, yang berisi *tuple* {baris,kolom} yang akan diisi dengan kotak hitam, sesuai dengan urutannya.

4.2 Pseudo-Code Solusi

```
function isSquareStream (input : m : matriks of integer, n : integer) -> boolean
{mengembalikan true jika ada daerah kotak hitam berbentuk persegi 2x2 atau lebih pada matriks n x n. Mengembalikan false jika tidak ditemukan}
Deklarasi
   i,j : integer
   found : boolean
Algoritma
   i <- 0 {untuk indeks baris}
   j <- 0 {untuk indeks kolom}
   found <- false

   while not found and (i < n) do
      while not found and (j < n) do
        if (m[i,j] adalah kotak hitam) then
           {cek apakah kotak di sekelilingnya
            membentuk persegi 2x2 atau lebih
            dengan kotak tersebut}
            if (ada kotak hitam yang membentuk
           persegi) then
               found <- true
           endif
        endif
        j <- j + 1
      endwhile
      i <- i + 1
   endwhile

  ->found
```

```
function isMoreThanOneStream (input : m : matriks of integer, n : integer) -> boolean
{mengembalikan true jika ada daerah kotak-kotak hitam yang ada membentuk lebih dari satu stream}
Deklarasi
   i,j, streamcount : integer
Algoritma
   i <- 0 {untuk indeks baris}
   j <- 0 {untuk indeks kolom}
   streamcount <- 0 {menghitung jumlah stream}

   while (streamcount <= 1) and (i < n) do
      while (streamcount <= 1) and (j < n) do
       if (m[i,j] adalah kotak hitam) then
          {mengecek apakah kotak tersebut
           membentuk rangkaian stream, kalau
          benar, maka streamcount bertambah 1}
       endif
       j <- j + 1
      endwhile
      i <- i + 1
   endwhile

  ->(streamcount <= 1)
```

```
function isUnconnectedIsland(input : m : matriks of integer, n : integer) -> boolean
{mengembalikan true jika ada daerah kotak putih yang dikelilingi stream dan tidak terhubung dengan kotak angka}
Deklarasi
   i,j : integer
   found : boolean
Algoritma
   i <- 0 {untuk indeks baris}
   j <- 0 {untuk indeks kolom}
   found <- false

   while not found and (i < n) do
      while not found and (j < n) do
        if (m[i,j] adalah kotak hitam) then
           {cek apakah kotak di sekelilingnya
            Terhubung dengan angka}
            if (tidak terhubung dengan angka)
            then
               found <- true
           endif
        endif
        j <- j + 1
      endwhile
      i <- i + 1
   endwhile

  ->found
```

```
function isNotAppropriateIslands (input : m : matriks of integer, n : integer) -> boolean
{mengembalikan true jika daerah kotak-kotak putih tidak melebihi jumlah pulau yang dispesifikasi. Mengembalikan false jika sebaliknya}
Deklarasi
   i,j,countisland, angka, a, b : integer
   found : boolean
Algoritma
   i <- 0 {untuk indeks baris}
   j <- 0 {untuk indeks kolom}
   a <- 0 {untuk indeks baris kotak putih}
   b <- 0 {untuk indeks kolom kotak putih }
   countisland <- 0 {menghitung jumlah pulau}
   angka <- 9999

   found <- false

   while not found and (i < n) do
      while not found and (j < n) do
        if (m[i,j] adalah kotak putih) then
           {cek apakah kotak di sekelilingnya
           terhubung dengan angka putih dan
           angka}
           a <- i
           b <- j
           while (countisland <= angka) do
             countisland = countisland + 1
             if (m[a,b] menemukan bilangan
             angka) then
                  angka <- m[a,b]
             else if (menemukan space kotak
              putih lainnya) then
               a = baris kotak putih selanjutnya
               b = kolom kotak putih selanjutnya
             endif
           endwhile

           if (countisland >= angka) or (angka =
          9999) then
               found <- true
           endif
        endif
        countisland = 0
        j <- j + 1
```

```
      endwhile
      i <- i + 1
   endwhile

  ->found
```

```
function isValid(input : m : matriks, n : integer, i : integer, j : integer) -> boolean
{mengembalikan true jika ada pengisian matriks pada baris i dan kolom j tidak menyalahi constraint permainan. Mengembalikan false jika sebaliknya}
Deklarasi
   temp : matriks of integer {berukuran n}
Algoritma
   copy isi matriks m ke dalam matriks temp
   isi temp[i,j]dengan kotak hitam

  ->(not isUnconnectedIsland(temp,n)) and
    (not  isMoreThanOneStream(temp,n)) and
    (not isSquareStream(temp,n))and
    (isNotApropriateIsland(temp, n)
```

Seperti yang dapat kita lihat, fungsi pembatas pada pencarian solusi ini, diwakilkan oleh fungsi isValid( ) di atas.

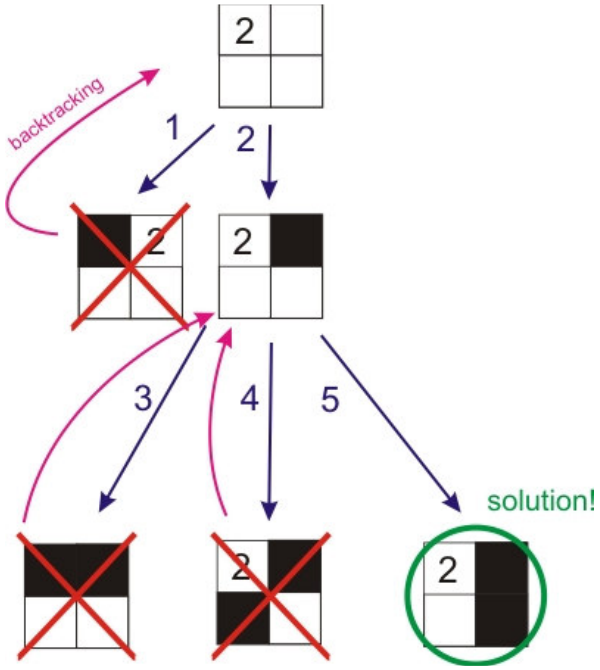
```
procedure CetakSolusi (input : i, j, n : integer, input/output : t : table of point)
{merupakan tabel yang berisi koordinat-koordinat dalam matriks yang merupakan solusi permainan nurikabe
masukan : table of point kosong, integer baris i dan kolom j yang merupakan solusi
keluaran : table of point yang berisi solusi final}
Deklarasi
Algoritma
   t[n] <- (i,j)
```

```
procedure Solving(input n : integer, input/output : m : matriks of integer)
{prosedur untuk menemukan solusi}
{masukan : matriks inisial dengan ukuran nxn}
{keluaran : matriks yang sudah diisi dengan solusi atau tidak ada solusi}
Deklarasi
   isFinis : boolean
   temp : matriks of integer
   t : table of solution {untuk solusi}
   i, j, indexsolusi : integer
Algoritma
   isFinish <- false
   i <- 0
   j <- 0
   indexsolusi <- 0

   (kopi isi matriks inisiasi  m ke dalam temp)

   while not isFinish do
        {periksa apakah kotak tersebut dapat
        diisi dengan kotak hitam}
        if (isValid(temp,n,i,j)) and temp[i,j]
       bukan kotak hitam atau angka) then
            temp[i,j] <- -1 {isi dengan kotak
            hitam}
            CetakSolusi(i,j,indexsolusi,t)
            Indexsolusi <- indexsolusi + 1
            j <- j + 1
         else if (not isValid(temp,n,i,j)) or
         temp[i,j] adalah kotak hitam atau angka)
         (backtracking)
          i <- diisi dengan baris setelah
          (BACKTRACK)
          j <- diisi dengan kolom backtrack
        else if (Menemukan solusi) then
          isFinish <- true
      i <- i + 1
   endwhile

if (isFinish) then
   output (“menemukan solusi yaitu :”  t)
else
   output (“Tidak menemukan solusi”)
endif
```

Pada pencarian solusi ini, program akan mencoba mengisi kotak satu persatu sampai menemukan solusi final. Alternatif langkah yang dinilai tidak membawa pada solusi tidak diteruskan.

4.3 Hasil Penerapan Pseudo-Code pada Nurikabe

Hasil penerapan algoritm backtracking pada nurikabe dengan ukuran 2x2 adalah sebagai berikut :



Gambar 6. Pohon pencarian solusi matriks nurikabe dengan ukuran 2x2

Maka solusi persoalan Nurikabe di atas adalah t : {(1,2), (2,2)}

Dengan pengambilan langkah sebanyak 5 kali dan backtracking sebanyak 3 kali.

5. KESIMPULAN

Permainan Nurikabe merupakan permainan yang sangat bagus untuk mengasah logika. Tingkat kerumitannya merupakan daya tarik tersendiri bagi kita untuk menemukan solusinya. Dengan adanya program komputer dengan implementasi algoritma komputasi tertentu, hal ini dapat memudahkan kita untuk menemukan solusi permainan Nurikabe baik dalam skala kecil maupun besar. Hal ini dapat berguna bagi kita, salah satunya adalah untuk menyerahkan pencarian solusi pada komputer, bila kita sudah tidak sanggup menemukan solusinya, sehingga kita dapat mengembangkan kemampuan logika kita dengan lebih baik lagi.

Penggunaan algoritma Runut-Balik (*Backtracking*) pada implementasi program pencarian solusi Nurikabe ini didasarkan pada fakta bahwa algoritma ini merupakan algoritma yang sudah banyak digunakan untuk pemecahan solusi permainan logika dan juga dinilai paling mangkus pada prakteknya.

REFERENSI

[1] Wikipedia. *Nurikabe* http://en.wikipedia.org/wiki/Nurikabe diakses pada tanggal 16 Mei pukul 20.00 WIB.

[2] Nurikabe Interactive Tutorials http://www.nikoli.co.jp/en/puzzles/nurikabe/ diakses pada tanggal 16 Mei 2008 pukul 20.00 WIB.

[3] Nurikabe – Online Puzzle Game http://www.puzzle-nurikabe.com/ diakses pada tanggal 16 Mei pukul 20.00 WIB.

[4] Wikipedia. *NP-Complete* http://en.wikipedia.org/wiki/NP-complete_problems#NP-complete_problems diakses pada tanggal 16 Mei pukul 20.00 WIB.

[5] Munir, Rinaldi. *Diktat Kuliah Strategi Algoritmik IF2251 Strategi Algoritmik*. Penerbit ITB. Bandung. 2006.