PENERAPAN ALGORITMA BFS DALAM PEMECAHAN PERMAINAN MINESWEEPER

Sesdika Sansani
NIM 13507047

Program studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
e-mail: sesdika@students.itb.ac.id

ABSTRAK

Minesweeper **merupakan salah satu permainan standar yang ada pada sistem operasi Windows atau sistem operasi UNIX. Masalah yang diangkat dalam permainan ini adalah menemukan seluruh ranjau pada petak yang telah disediakan tanpa meledakkannya. Cara menemukan ranjau adalah dengan menggunakan petunjuk yang tertera sebagai nilai pada petak non-ranjau yang telah terbuka.**
Salah satu cara yang digunakan untuk memecahkan masalah ini adalah dengan menggunakan algoritma BFS (*Breadth First Search*) atau Pencarian Melebar. Algoritma ini memiliki karakteristik pencarian melebar, yaitu mengunjungi seluruh anak-anak dari suatu simpul atau keadaan status dahulu baru setelah itu memperluasnya kembali.
Algoritma ini digunakan sebagai bagian dari pemecahan masalah, yaitu untuk memfasilitasi proses penelusuran petak-petak yang ada pada papan *minesweeper*. Jika saat bertemu dengan petak angka, akan dilakukan algoritma lain untuk menentukan posisi ranjau relatif terhadap petak tersebut.
Menurut penelitian, masalah *minesweeper* tergolong ke dalam masalah *NP-complete*. Hal ini dikarenakan masalah *minesweeper* tidak dapat diselesaikan dalam orde waktu polinomial.

Kata kunci: *minesweeper*, *BFS*, *NP-complete*.

1. PENDAHULUAN

Pemecahan permainan *minesweeper* merupakan salah satu masalah yang termasuk ke dalam golongan masalah *NP-complete*. Bahkan *Clay Mathematics Institute of Chambridge*, Massachusetts (CMI) menggolongkannya ke dalam salah satu dari tujuh masalah millenium yang dikenal dengan nama *Prize Problems*. CMI akan menganugrahkan $1 juta bagi yang dapat memecahkan salah satu masalah tersebut terkait hal masalah P = NP.

Karena *minesweeper* termasuk ke dalam *NP-complete*, tidak ada algoritma yang dapat digunakan untuk memecahkan masalah tersebut dalam waktu polinomial. Walaupun begitu, tetap ada langkah-langkah yang dapat digunakan untuk mencoba memecahkan masalah tersebut.

2. MINESWEEPER

Ketika permainan ini dimulai, pemain dihadapkan dengan petak-petak kotak abu-abu. Jumlah petak bergantung pada tingkat *skill* yang dipilih pemain, tingkat *skill* yang lebih tinggi memiliki jumlah petak lebih banyak. Jika pemain mengklik salah satu petak tanpa ranjau, akan keluar angka dari petak. Angka tersebut mengindikasikan jumlah dari petak tetangga (tidak lebih dari 8) yang terdapat ranjau. Dengan menggunakan lojik, dengan jumlah petak yang banyak terbuka, pemain dapat menggunakan informasi tersebut untuk menarik kesiimpulan apakah petak lain bebas dari ranjau atau terisi oleh ranjau, dan meneruskan mengklik petak lain untuk membersihkannya atau menandai dengan gambar bendera yang mengindikasikan adanya ranjau di petak tersebut.

Pemain dapat menempatkan gambar bendera di petak manapun yang terindikasi terdapat ranjau dengan mengklik kanan petak. Mengklik kanan petak yang telah ditandai bendera kadang-kadang (bergantung konfigurasi) akan mengubah gambar bendera menjadi tanda tanya (?) yang mengindikasikan petak tersebut mungkin terdapat atau tidak terdapat ranjau.

Gambar 1 Gambar permainan *Minesweeper* tingkat pemula

3. NP-COMPLETE

Kebutuhan waktu algoritma yang mangkus bervariasi, mulai dari $O(1)$, $O(\log \log n)$, $O(\log n)$, $O(n)$, $O(n^2)$, dan $O(n^3)$. Semua algoritma tersebut dikenal sebagai solusi polinomial dikarenakan kebutuhan waktunya secara asimptotik dibatasi oleh fungsi polinomial. Misalnya $\log(n) < n$ untuk semua $n \geq 1$. Sebaliknya, ada persoalan yang tidak terdapat solusi waktu polinomial untuk menyelesaikannya, misalnya TSP yang memiliki kompleksitas $O(n!)$.

Polynomial-time algorithm adalah algoritma yang kempleksitas waktu kasus-terburuknya dibatasi oleh fungsi polinom dari ukuran masukannya. Di luar algoritma tersebut, algoritmanya dikenal dengan *nonpolynomial-time algorithm*.

P Problems adalah himpunan semua persoalan keputusan yang dapat dipecahkan oleh algoritma dengan kebutuhan waktu polinom. *NP Problems* adalah himpunan persoalan keputusan yang dapat diselesaikan oleh algoritma non-deterministik dalam waktu polinom. Algoritma non-deterministik adalah algoritma yang berhadapan dengan beberapa opsi pilihan, dan algoritma memiliki kemampuan untuk menerka opsi pilihan yang tepat.

Semua persoalan P juga adalah NP, sebab tahap menerka tidak terdapat di dalam persoalan P. Karena itu, P adalah himpunan bagian dari NP. Namun, belum ada yang bisa membuktikan apakah masalah P = NP atau P tidaksamadengan NP. Pertanyaan ini penting sebab kebanyakan persoalan keputusan adalah NP. Karena itu, jika P = NP, maka akan ada banyak persoalan keputusan yang dapat dipecahkan secara mangkus dengan algoritma yang kebutuhan waktunya polinom. Namun kenyataannya, banyak ahli yang telah gagal menemukan algoritma waktu-polinom untuk persoalan NP. Karena itu, cukup aman kalau kita menduga-duga bahwa P tidaksamadengan NP.

NP-Complete (NPC) adalah persoalan NP yang paling sulit. Sebuah persoalan *X* dikatakan NPC jika:

1. *X* termasuk ke dalam kelas NP
2. Setiap persoalan di dalam NP dapat direduksi dalam waktu polinom menjadi *X*

Penjelasannya adalah sebagai berikut: untuk persoalan *X* di dalam NPC, pertama harus dipahami bahwa *X* adalah NP. Kemudian, kita seharusnya dapat mereduksi sembarang persoalan lain di dalam NP dengan transformasi sederhana menjadi instance persoalan *X*. Efeknya, jika transformasi ini dapat dilakukan, maka jika algoritma dalam waktu polinom ditemukan untuk *X*, maka semua persoalan di dalam NP dapat diselesaikan dengan mangkus. Dengan kata lain, jika *X* adalah NPC dan termasuk ke dalam P – yaitu algoritma yang mangkus (polinom, deterministik) untuk *X* ditemukan – maka P = NP. Hal ini karena transformasi tersebut sederhana (membutuhkan waktu polinom).

4. ALGORITMA BFS

Pada teori graf, BFS adalah algoritma pencarian pada graf yang dimulai dari simpul akar dan menelusuri seluruh simpul tetangganya. Kemudian untuk setiap simpul terdekat, algoritma ini menelusuri simpul tetangganya yang belum ditelusuri, dan seterusnya hingga menemukan solusi.

BFS adalah metode pencarian yang bertujuan untuk memperluas dan memeriksa semua simpul pada graf atau urutan kombinasi dengan pencarian secara sistematis melalui setiap solusi. Dengan kata lain, ia akan melakukan pencarian secara mendalam pada keseluruhan graf atau urutan tanpa memperhatikan tujuan hingga menemukan tujuan tersebut. Algoritma ini tidak menggunakan algoritma heuristik.

Dari sudut pandang algoritma, semua simpul anak didapatkan dengan memperluas simpul yang ditambahkan pada *queue FIFO*.

Langkah-langkah pada algoritma ini adalah sebagai berikut:

- Traversal dimulai dari simpul *v*
- Algoritma:
  1. Kunjungi simpul *v*
  2. Kunjungi semua simpul yang bertetangga dengan simpul *v* terlebih dahulu.
  3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.
- Jika graf berbentuk pohon berakar, maka semua simpul pada aras *d* dikunjungi lebih dahulu sebelum mengunjungi simpul-simpul pada aras *d* + 1.

5. ANALISA PERMAINAN

5.1 Pola dan Pemecahan

Terdapat banyak pola dan petak bernomor yang mungkin timbul selama permainan yang dapat dikenali hanya memiliki satu kemungkinan konfigurasi ranjau yang ada di sekitarnya. Dalam ketertarikan untuk menyelesaikan dengan cepat, seringnya lebih mudah untuk memproses pola yang telah diketahui terlebih dahulu, dan melanjutkan bagian yang tidak pasti setelahnya. Ada beberapa metode kasar untuk memecahkan masalah pada permainan *minesweeper* tanpa harus menebak.

5.1.1 Analisa Single-Square

Ada dua kasus khusus yang memiliki perhatian ekstra pada analisa ini?

- Jika jumlah petak *unclicked* (*blank* atau berbendera) yang bertetangga sama dengan angka pada petak bernomor, semua petak *unclicked* pasti adalah ranjau.
- Untuk petak bernomor apa pun, jika jumlah ranjau yang telah ditemukan yang bertetangga dengan petak tersebut sama dengan angka pada petak, semua petak lain yang bertetangga dengan petak bernomor tersebut pasti aman.

Gambar 2 *a* dan *b* aman untuk dibuka, karena 3 telah dipenuhi oleh ranjau tetangga

Gambar 3 *a* dan *b* pasti ranjau; petak yang dapat memenuhi 3 adalah *a* dan *b*

5.1.2 Analisa Multiple Square

Untuk memecahkan *puzzle* yang lebih kompleks, seseorang perlu mempertimbangkan lebih dari satu petak sekali waktu. Beberapa strategi yang dapat digunakan adalah:

- Jika terdapat dua angka bertetanggaan, selisih antara kedua angka tersebut sama dengan selisih jumlah ranjau untuk tiga petak yang bertetangga dengan masing-masing petak yang tidak bertetangga dengan angka lain. Contohnya: jika angka-angka ini digantikan dengan 3, seluruh petak yang bertetangga ke angka lebih tinggi yang tidak dipakai bersama dengan yang lain adalah ranjau, dan semua yang berhadapan adalah aman.
- Dengan metode yang sama, kadang-kadang tidak dapat diketahui bahwa terdapat sejumlah ranjau pada sejumlah petak (tanpat perlu mengetahui petak mana yang aman dan petak yang yang didiami ranjau) dan informasi tersebut dapat digunakan untuk menemukan informasi mengenai petak lain.

Variabel/yang tak dikenal adalah petak yang belum-terbuka, dan pembatasnya adalah petak tetangga yang telah terbuka. Algoritma pada analisa ini terdiri dari mencoba setiap kombinasi ranjau yang memenuhi setiap angka pada petak tetangga, dan dari situ membuat kesimpulan. Untuk *puzzle* yang lebih besar, ini adalah proses yang memakan-waktu untuk komputer, tetapi ahli *minesweeper* mungkin mampu melihat dengan cepat petak mana yang memerlukan prosedur ini dan petak mana yang dapat memberikan keerhasilan. Dua aturan di atas benar-benar kasus khusus.

Contoh: Petak pojok dan 3 petak tetangga telah terbuka. Huruf di sini adalah petak belum-terbuka dan merupakan variabel.

Gambar 4 Gambar untuk contoh pada analisa *Multi-Square*

Mencoba setiap kombinasi dengan membabi buta memberikan 4 konfigurasi valid (di luar 2^5), diberi nama {*a*,*b*,*c*,*d*,*e*} = {1,0,1,0,0}, {0,1,1,0,0}, {1,0,0,1,0} dan {0,1,0,1,0}, 1 merepresentasikan ranjau.

Hal yang ditemukan pada semua konfigurasi adalah variabel 3 tidak pernah sebagai ranjau. Kesimpulannya adalah pada semua kemungkinan konfigurasi yang valid, petak *e* aman, dan dapat dengan aman dibuka. Sejalan dengan itu, jika petak ditandai sebagai ranjau pada setiap konfigurasi yang valid, petak tersebut pastilah ranjau.

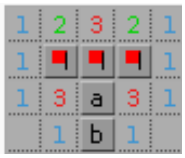
5.1.3 Analisa Final

Digunakan di akhir permainan, analisa ini dapat digunakan untuk membuka seluruh petak ketika seluruh petak pada papan adalah aman atau dapat diperlihatkan terdapat ranjau.

5.2 Elemen Tebakan

Pada seluruh implementasi *minesweeper*, terdapat kemungkinan petak-petak yang dibangkitkan tidak dapat

dipecahkan kecuali dengan menebak. Contohnya adalah kondisi berikut:



Gambar 5 Gambar minesweeper yang memerlukan penebakan elemen

Pemain harus menebak apakah a atau b adalah ranjau.

Constraint satisfaction problem dapat membantu sedikit dalam mengestimasi kemungkinan bahwa petak terdapat ranjau; daftar seluruh kombinasi yang valid dan hitung berapa kali setiap petak didiami ranjau. Jika kepadatan dari ranjau diketahui (atau diestimasi selama permainan), pemain dapat mengambil petak yang kemungkinan kecil mengandung ranjau.

Contoh nyata lain yang dibutuhkan untuk menebak adalah ketika petak *unclicked* dengan sepenuhnya dikelilingi oleh ranjau, atau kombinasi dari ranjau dan batas pinggir dari jendela permainan. Pada kasus ini, karena tidak ada angka menyentuh petak *unclicked*, pemain tidak memiliki informasi mengenai kemungkinan petak *unclicked* adalah ranjau. Akan tetapi, tetap ada strategi yang bagus ketika menghadapi situasi yang memungkinkan pemain untuk menghindar dari tebakan sederhana: mainkan sisa permainan dan abaikan petak tersebut. Ketika angka yang belum dibuka, ruangan tak berbendera yang tertinggal sama dengan jumlah sisa ranjau yang belum ditandai, maka sisa seluruh ruangan adalah ranjau.

6. PENERAPAN ALGORITMA BFS SEBAGAI BAGIAN PEMECAHAN MASALAH MINESWEEPER

Solusi yang digunakan untuk memecahkan masalah minesweeper dilakukan dengan cara menelusuri petak-petak yang memliki tetangga ranjau yang dapat ditentukan dengan pasti. Untuk melakukan penelusuran tersebut, digunakanlah algoritma BFS.

Seperti yang telah dijelaskan sebelumnya, algoritma BFS akan mengunjungi setiap tetangga dan akan memperluasnya. Definisi tetangga pada pemecahan di sini adalah petak tetangga yang memiliki keberadaan ranjau tetangganya secara pasti sehingga petak tetangga tersebut aman untuk dibuka.

Untuk memecahkan persoalan *minesweeper* ini, struktur data yang digunakan adalah sebagai berikut:

- Queue → untuk menyimpan petak bernilai 1 s.d. 8 yang tidak dapat diperluas lagi.
- Petak → struktur data yang menyimpan informasi petak mengenai posisi dan status petak tersebut. Status-status yang mungkin ada antara lain: 0 = petak kosong; 1..8 = nilai petak yang mungkin; 9 = bertanda bendera; 10 = ditandai tanda tanya; 999 = belum dibuka.
- Papan → terdiri dari petak-petak pada permainan *minesweeper*

Bagian utama dari program berisi kalang yang akan melakukan penelusuran ke petak-petak yang dapat dibuka berdasarkan informasi yang ada. Jika penelusuran terhenti, petak-petak dalam `Queue` akan ditelusuri untuk menemukan ranjau atau petak bernomor lain dengan menggunakan prinsip analisa permainan yang telah diuraikan sebelumnya.

Proses penelurusan BFS yang digunakan adalah dengan menggunakan metode rekursif dengan prosedur yang diberi nama `CekPetak`. Basis dari prosedur ini adalah jika menemui kondisi petak yang sedang dikunjungi belum memiliki tetangga yang pasti aman. Jika pasti aman, maka prosedur `CekPetak` akan dipanggil kembali untuk tetangga-tetangganya. Algoritma prosedur ini yaitu sebagai berikut:

```
procedure CekPetak (Input pt : Petak)
{ I.S. Petak yang belum terbuka
tidaksamadengan 0
  F.S. Penelusuran petak-petak pada Papan
tidak dapat dilanjutkan
  Proses : Melakukan penelusuran petak-
petak pada Papan secara rekursif hingga
petak-petak tidak dapat ditelusuri lagi.
Proses ini akan membuat variabel global
Papan membuka salah satu petaknya jika
petak tersebut aman untuk dibuka atau
menandai dengan bendera jika petak
tersebut sudah pasti adalah ranjau. }

Deklarasi
  ranjauTerbuka : boolean
  i : integer
  petakTemp : Petak

Algoritma
  belumPasti ← false
  if (pt.flag <> 0) then     { basis }
    ranjauTerbuka ← cekSingleSquare(pt)
    if (not ranjauTerbuka) then
      insertQueue(pt)
    endif
  else                          { rekurens }
    i traversal [1..8]
      petakTemp ← petakArah(1)
      if (petakTemp.flag = 999 or
(petakTemp.flag ≥ 1 and petakTemp.flag ≤
8)) then
        CekPetak(petakTemp)
      endif
  endif
```

Fungsi `cekSingleSquare` adalah fungsi yang mengecek kepastian petak-petak tetangga dari petak yang sedang dikunjungi. Fungsi ini menggunakan analisa *Single-Square*. Jika dalam proses pengecekan tetangga ranjau ada ranjau yang dapat dibuka, maka *flag* petak tetangga tersebut diubah menjadi *flag* bendera dan mengembalikan nilai *true*. Sedangkan prosedur `insertQueue` adalah prosedur yang memasukkan petak ke dalam `Queue`.

Prosedur lain yang terlibat untuk memecahkan minesweeper dengan menggunakan analisa permainan yang telah diuraikan adalah sebagai berikut:

```
procedure TelusurPetakInQueue ()
{
  I.S. Sembarang
  F.S. Petak-petak yang memiliki petak
tetangga yang dapat dibuka telah dihapus
dan tetangga tersebut telah dibuka.
  Proses : Menelusuri petak-petak dalam
Queue dimulai dari head. Jika tetangga
dari petak pada head dapat dibuka, seluruh
tetangga akan dibuka dan head dihapus.
Pengecekan kepastian tetangga dilakukan
dengan analisa Single-Square dan Multi-
Square. Jika head memiliki tetangga yang
belum bisa dibuka, head tersebut diambil
dan diinsert kembali ke dalam Queue. Jika
selama 1 siklus Queue tidak mengalami
perubahan, proses penelusuran dihentikan.
}
Deklarasi
  headQ : Petak
  patokan : Petak
  n : integer
  ranjauTerbuka : boolean
  selesai : boolean
  pertama : boolean

Algoritma
  n ← panjangQueue()
  selesai ← false
  pertama ← true
  while (not selesai) do
    headQ ← headQueue()
    ranjauTerbuka  ← cekMultiSquare(headQ)
    if (ranjauTerbuka) then
      { nothing to do }
    else
      headQ ← deleteQueue()
      addQueue(headQ)
      if (pertama) then
        patokan ← headQ
        pertama ← false
      endif
    endif
    if (isQueueEmpty() or (headQ = patokan
and panjangQueue() = n)) then
    { queue tidak mengalami perubahan atau
queue telah kosong, hentikan penelusuran }
      selesai ← true
    else if (headQ = patokan) then
    { penelusuran memasuki siklus baru }
      n ← panjangQueue()
      pertama ← true
    endif
  endwhile
```

Fungsi `cekMultiSquare` adalah fungsi yang mengecek kepastian petak-petak tetangga dari petak yang sedang dikunjungi dengan menggunakan analisa *Multi-Square*. Jika dalam proses pengecekan tetangga ranjau ada ranjau yang dapat dibuka, maka *flag* petak tetangga tersebut diubah menjadi *flag* bendera dan mengembalikan nilai *true*.

Algoritma pemecahan minesweeper selengkapnya adalah:

Prosedur-prosedur di atas digunakan dalam program utama, yaitu sebagai berikut:

```
{ Kamus Global }
Deklarasi
  type Petak <
    i: integer
  { koordinat vertikal relatif dari pojok
kiri atas }
    j: integer
  { koordinat horizontal relatif dari
pojok kiri atas }
    nilai: integer
  { nilai dari petak →
    0: petak kosong;
    1..8: nilai petak yang mungkin;
    9: ditandai bendera;
    10: ditandai tanda tanya;
    999: belum dibuka
   }
  >
  Papan : array [1..n] [1..n] of Petak
  petak : Petak

  function terkenaRanjau(Input ptk :
Petak)→ boolean
  { Mengembalikan true jika petak yang
sedang dikunjungi merupakan petak, false
jika sebaliknya }

  function allRanjauTerbuka()→ boolean
  { true jika semua ranjau telah
tertebak, false jika sebaliknya }

  function belumTelusur(Input ptk :
Petak) → boolean
  { true jika Petak ptk belum pernah
dikunjungi, false jika sebaliknya }

  procedure CekPetak (Input pt : Petak)

  procedure TelusurPetakInQueue ()

Algoritma
  while (not terkenaRanjau(petak) or
   allRanjauTerbuka()) do
    petak ← randomPetak()
```

```
    if (belumTelusur(petak)) then
      CekPetak(petak)
      TelusurPetakInQueue()
    endif
  endwhile
```

7. ANALISA SOLUSI

Algoritma BFS pada persoalan kali ini digunakan sebgai metode penelusuran petak-petak. Algoritma ini dipilih karena sifat pencariannya yang menyebar sehingga proses penelusuran seluruh petak dapat lebih cepat dilakukan dibandingkan algoritma pencarian lain seperti DFS.

Secara keseluruhan, solusi ini belum dapat menghasilkan hasil yang akurat. Ada kalanya program tidak dapat memperluas petak yang sedang ditelusuri lagi sehingga program harus mengambil petak selanjutnya secara random. Akibatnya, terdapat kemungkinan program salah mengambil petak dan petak tersebut adalah ranjau sehingga permainan menjadi berakhir.

Seperti yang sudah dipaparkan sebelumnya, masalah ini temrasuk ke dalam masalah *NP-complete* dan pemecahan yang telah dipaparkan belum dapat memberikan solusi polinomial. Hal ini secara kasar dapat dilihat dari kompleksitas algoritma BFS sendiri, yaitu $O(b^d)$ (b : jumlah simpul yang dibangkitkan, d : ketinggian saat solusi ditemukan).

8. KESIMPULAN

Algoritma BFS dapat digunakan sebagai bagian dari pemecahan masalah *minesweeper*, yaitu untuk menelusuri petak-petak yang ada di papan *minesweeper*. Secara keseluruhan, pemecahan yang dipaparkan belum dapat memberikan solusi yang akurat dan belum dapat memberikan waktu pemecahan polinomial.

REFERENSI

[1] http://en.wikipedia.org/wiki/Breadth-first_search
Waktu akses: 5 Januari 2010, pukul 09.53 WIB

[2] http://en.wikipedia.org/wiki/*Minesweeper*_%28computer_game%29
Waktu akses: 22 Desember 2009. pukul 11.36 WIB

[3] http://webmail.informatika.org/~rinaldi/Stmik/2009-2010/Teori%20P,%20NP,%20dan%20NP-Completeness.ppt
Waktu akses: 22 Desember 2009, pukul 10.47 WIB

[4] http://www.claymath.org/millennium/
Waktu akses: 30 Desember 2009, pukul 16.03 WIB

[5] Munir, Rinaldi. *Diktat Kuliah IF3051 Strategi Algoritma*. Program Studi Teknik Informatika ITB. 2009.