Penerapan Algoritma *Greedy* untuk Permainan *Flood It*

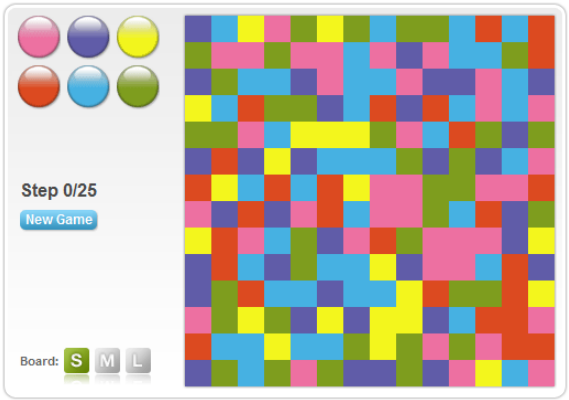
Athia Saelan / 13508029[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]if18029@students.if.itb.ac.id

***Abstrak*—*Flood it* adalah sebuah permainan komputer. Papan permainannya terdiri dari kotak-kotak kecil dengan enam jenis warna. Pemain diminta untuk terus memilih warna sedemikian sehingga seluruh kotak dalam papan tersebut pada akhirnya berwarna sama. Pada permainan ini juga dibutuhkan optimasi, yaitu jumlah langkah yang sesedikit mungkin. Algoritma *greedy* merupakan salah satu algoritma yang banyak digunakan untuk mengatasi masalah optimasi. Prinsip dari algoritma *greedy* adalah memilih langkah yang terbaik saat ini dengan harapan langkah tersebut membawa pada hasil yang optimal. Makalah ini akan memberikan beberapa solusi algoritma *greedy* untuk diterapkan pada permainan *flood it*. Selain itu, makalah ini juga menyertakan perbandingan antara beberapa solusi *greedy* yang diberikan.**

Kata Kunci*—Algoritma, *Greedy*, *Flood It

I. PENDAHULUAN

Flood it merupakan salah satu permainan komputer yang dapat dimainkan di internet, salah satunya pada [2]. Tampilan dari salah satu versi permainan ini dapat dilihat pada Gambar 1 berikut.



Gambar 1 Tampilan permainan *flood it*

Papan permainan ini berbentuk sebuah persegi besar berisi n x n buah kotak berbentuk persegi kecil. Kotak-kotak tersebut terbagi dalam 6 warna yang berbeda. Di sebelah persegi besar juga terdapat 6 buah tombol untuk masing-masing warna kotak. Tombol ini berfungsi untuk mengubah warna kotak yang paling kiri atas dan semua kotak yang telah terhubung dengannya.

Permainan dimulai dari kotak paling kiri atas. Pemain harus memperhatikan warna kotak-kotak di sekitarnya, kemudian memilih tombol yang sesuai, sedemikian sehingga warna dari kotak tersebut bisa terus menyebar dan mencapai seluruh kotak di papan.

(a) (b) (c)

Gambar 2 Ilustrasi cara permainan

Misalnya untuk Gambar 2(a), kotak paling kiri atas berwarna biru (anggap namanya k1), di sebelah kanannya berwarna kuning (k2), dan di sebelah bawahnya berwarna merah (k3). Ketika pemain menekan tombol kuning, maka k1 akan berubah warna menjadi kuning, sehingga bersatu dengan k2, seperti pada gambar 2(b). Setelah itu, jika pemain memilih tombol merah, k1 yang sudah bersatu dengan k2 akan berubah warna menjadi merah, dan bersatu dengan k3 seperti pada gambar 2(c), sehingga kini mereka telah berukuran minimal 3 kotak. Kemudian pemain harus memilih tombol selanjutnya agar semua kotak dalam papan dapat bersatu.

Objektif dari permainan ini adalah mengatur agar semua kotak pada papan permainan berwarna sama dengan jumlah langkah yang dibatasi. Namun akan lebih baik lagi jika permainan dilakukan dengan langkah sesedikit mungkin. Pembatasan jumlah langkah tergantung pada besar papannya.

II. DASAR TEORI

Algoritma *greedy* adalah salah satu algoritma yang sering digunakan untuk memecahkan masalah optimasi. Secara harfiah, *greedy* berarti tamak. Algoritma *greedy* adalah algoritma yang tamak, maksudnya algoritma ini akan mencari solusi optimum pada setiap langkahnya (optimum lokal), dengan prinsip "*take waht you can get now!*". Meskipun begitu, solusi ini belum tentu menjadi solusi yang optimum global.

Walaupun solusi yang dihasilkan *greedy* belum tentu

optimal, algoritma ini seringkali dapat menghasilkan solusi yang hampir optimal. Selain itu, algoritma ini juga memiliki kompleksitas yang relatif rendah sehingga menjadi pilihan yang cukup baik untuk masalah yang butuh diselesaikan dengan cepat walaupun hasilnya tidak terlalu optimal.

Untuk masalah optimasi, elemen-elemen algoritma *greedy* secara umum adalah sebagai berikut.

a. Himpunan kandidat (C)
b. Himpunan solusi (S)
c. Fungsi seleksi
d. Fungsi kelayakan
e. Fungsi objektif

Skema umum algoritma *greedy* dapat dirumuskan sebagai berikut.

1. Inisialisasi S dengan kosong
2. Pilih sebuah kandidat dari C dengan fungsi seleksi
3. Kurangi C dengan kandidat yang sudah dipilih
4. Periksa apakah kandidat yang dipilih bersama-sama dengan himpunan solusi membentuk solusi yang layak. Jika iya, masukkan kandidat tersebut ke dalam himpunan solusi.
5. Periksa apakah himpunan solusi sudah memberikan solusi yang lengkap. Jika iya, selesai. Jika tidak, ulangi lagi dari langkah 2 [1].

III. APLIKASI

Dari penjelasan di atas, dapat dilihat bahwa permainan *flood it* sangat cocok dengan algoritma *greedy*. Secara alami, pemain akan cenderung memilih warna yang tampaknya akan paling banyak bermanfaat pada saat itu, meskipun belum tentu warna itu yang terbaik dipilih.

Untuk menggunakan algoritma *greedy* tersebut, pertama-tama, elemen-elemen algoritma *greedy* untuk kasus ini harus ditentukan. Ada lima elemen yang telah dibahas pada bab II.

1. Himpunan kandidat, dalam kasus ini adalah keenam warna yang dapat dipilih.
2. Himpunan solusi, yaitu warna yang dipilih. Namun, warna yang dipilih akan langsung dimasukkan ke dalam papan permainan sehingga tidak disimpan di dalam himpunan.
3. Fungsi seleksi, tergantung strategi yang dipilih.
4. Fungsi kelayakan, dalam hal ini, jumlah kotak yang tergabung bertambah. Fungsi ini tidak selalu digunakan, karena pada beberapa strategi *greedy*, kandidat yang tidak layak tidak akan terpilih oleh fungsi seleksi.
5. Fungsi objektif, merupakan tujuan dari permainan, yaitu semua kotak berwarna sama dengan jumlah langkah minimum.

Setelah itu, harus dipilih strategi *greedy* seperti apa yang akan dilakukan. Ada beberapa pendekatan yang akan dibahas pada makalah ini, yaitu *greedy* berdasarkan jumlah, luas permukaan, dan panjang garis batas.

A. Jumlah

Cara paling mudah dalam menentukan warna yang akan dipilih adalah dengan melihat jumlah kotak berbatasan yang berwarna sama. Pada setiap langkah, pilihlah warna yang paling banyak berbatasan dengan daerah yang sudah terbentuk. Untuk lebih jelasnya dapat dilihat pada gambar berikut.

Gambar 3 Ilustrasi cara perhitungan jumlah

Pada gambar tersebut, dapat dilihat bahwa daerah yang sudah terbentuk di kiri atas berbatasan dengan dua buah kotak merah dan satu buah kotak kuning. Jika seperti itu, pemain akan lebih baik apabila memilih warna merah untuk langkah selanjutnya.

Untuk strategi ini, *pseudocode*-nya sebagai berikut.

```
procedure greedy1

kamus
   c : kandidat;
   warnapilihan : warna;
   w : warna {warna di papan[1][1]}
   i : integer;
   max : integer;
   tmp: integer;

algoritma
   max := 0;
   w := papan[1][1];
   while not win do
      for i:=1 to 6
         if c[i] ≠ w then
            papantemp := papan;
            tmp := jml(w,c[i],1,1);
            if tmp > max then
               max := temp;
               warnapilihan := c[i];
      pilihwarna(warnapilihan);
```

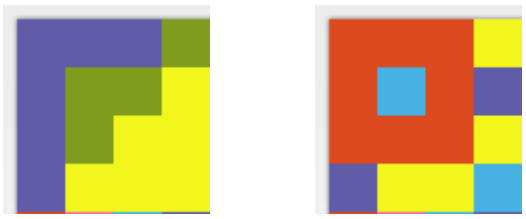
Fungsi seleksi pada kode tersebut adalah fungsi jml. Fungsi ini berfungsi untuk menghitung jumlah kotak yang berbatasan yang berwarna sesuai dengan warna yang akan dipilih. Fungsi ini diimplementasikan secara rekursif dengan skema sebagai berikut.

```
function jml(
   wlama,wcoba: warna;
   i,j: integer)→integer

kamus
   hasil : integer

algoritma
   hasil := 0;
```

```
   if (dalam batas papan) then
      if papan[i][j] = wcoba then
         hasil := 1;
      else if papan[i][j] = wlama then
         papan[i][j] := sudah_dikunjungi;
         hasil :=
            jml(wlama,wcoba,i+1,j) +
            jml(wlama,wcoba,i,j+1) +
            jml(wlama,wcoba,i-1,j) +
            jml(wlama,wcoba,i,j-1);
   }
}
return hasil;
```

Pada strategi ini, fungsi kelayakan tidak dibutuhkan. Secara logika, jika jumlah kotak yang berbatasan banyak, jumlah kotak yang tergabung pasti akan bertambah.

Namun cara ini merupakan cara yang sangat naif, karena bisa saja jumlah yang berbatasannya banyak, namun tidak mempengaruhi banyak kotak yang lain. Sebagai contohnya, tinjau kondisi pada Gambar 4 di bawah ini.



(a) (b)

Gambar 4 Kelemahan algoritma berdasarkan jumlah

Pada Gambar 4(a), jumlah kotak hijau yang berbatasan ada 4 kotak, sedangkan kotak kuning hanya 1 kotak. Namun cukup jelas terlihat bahwa pemilihan warna kuning akan memberikan hasil yang kemungkinan besar lebih baik daripada pemilihan warna hijau.

Selain itu, ada juga masalah yang timbul seperti pada Gambar 4(b). Pada gambar tersebut, memang hanya ada sebuah kotak biru yang berbatasan. Namun, pada kenyataannya, kotak tersebut akan dianggap empat karena jumlah sisi yang berbatasannya ada empat buah.

B. Luas Permukaan

Karena kekurangan algoritma berdasarkan jumlah, dibuatlah alternatif baru, yaitu algoritma *greedy* berdasarkan luas permukaan. Luas permukaan di sini maksudnya adalah luas permukaan yang akan bertambah ketika suatu warna dipilih. Jadi, pada setiap langkah, warna akan dipilih adalah warna yang jika dipilih akan membentuk daerah yang paling luas.

Pseudocode untuk strategi ini tidak jauh berbeda dengan strategi yang sebelumnya.

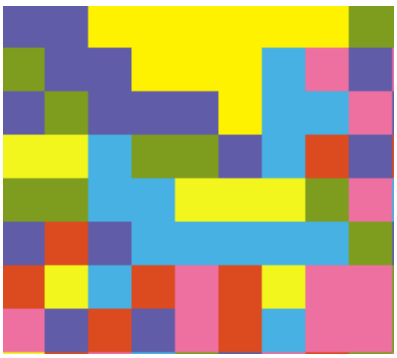
```
procedure greedy2

kamus
   c : kandidat
   warnapilihan : warna;
   w : warna {warna di papan[1][1]}
   i : integer;
   max : integer;
   tmp: integer;

algoritma
   max := 0;
   w := papan[1][1];
   while not win do
      for i:=1 to 6
         if c[i] ≠ w then
            papantemp := papan;
            tmp := luas(w,c[i],1,1,0);
            if tmp > max then
               max := temp;
               warnapilihan := c[i];
      pilihwarna(warnapilihan);
```

Yang berbeda pada kode ini hanyalah fungsi seleksinya, yaitu luas. Fungsi ini berfungsi untuk menghitung luas yang akan terjadi jika suatu warna dipilih. Fungsi ini juga diimplementasikan secara rekursif dengan skema sebagai berikut.

```
function luas(
   wlama,wcoba: warna;
   i,j,status: integer)→integer

kamus
   hasil : integer

algoritma
   hasil := 0;
   if (dalam batas papan) then
      if papan[i][j] = wcoba then
         papan[i][j] := sudah_dikunjungi;
      hasil := 1 +
            luas(wlama,wcoba,i+1,j) +
            luas(wlama,wcoba,i,j+1) +
            luas(wlama,wcoba,i-1,j) +
            luas(wlama,wcoba,i,j-1);
      else if papan[i][j] = wlama
      and status = 0 then
         papan[i][j] := sudah_dikunjungi;
         hasil := 1 +
            luas(wlama,wcoba,i+1,j) +
            luas(wlama,wcoba,i,j+1) +
            luas(wlama,wcoba,i-1,j) +
            luas(wlama,wcoba,i,j-1);
   return hasil;
```

Sama seperti cara sebelumnya, strategi ini pun tidak membutuhkan fungsi kelayakan. Pada strategi ini, seleksi dilakukan berdasarkan jumlah yang bertambah, sehingga sudah pasti yang terpilih adalah yang luasnya bertambah.

Meskipun begitu, algoritma ini pun ternyata belum tentu optimal, misalnya untuk kondisi pada Gambar 5 di bawah ini.



Gambar 5 Kelemahan algoritma berdasarkan luas

Pada awalnya, jumlah kotak yang telah tergabung ada tujuh kotak. Dapat dilihat bahwa pemilihan warna kuning tampak akan memberikan keuntungan yang besar. Jika pemain memilih warna kuning, kotak yang telah tergabung bertambah menjadi 17 kotak.

Meskipun begitu, pemain juga dapat memilih warna biru muda untuk langkah selanjutnya, dengan hasil yang sedikit lebih rendah, yaitu 15 kotak. Namun, jika pemain memilih warna biru muda, kotak yang dapat terpengaruhi selanjutnya akan jauh lebih banyak. Ketika pemain memilih warna kuning, maka jumlah kotak yang terpengaruhi selanjutnya hanya bertambah lima kotak, sedangkan jika memilih biru muda dapat bertambah belasan kotak.

Untuk perbandingan yang lebih teliti, dapat dibuat dua kemungkinan. Pertama, pemain memilih warna kuning kemudian biru. Kedua, pemain memilih warna biru terlebih dahulu, baru kemudian kuning. Jumlah kotak yang dihasilkan untuk setiap langkahnya dituliskan pada Tabel 1 berikut.

Tabel 1 Perbandingan hasil dari kedua kemungkinan pengambilan langkah

Kemungkinan	Langkah 1	Langkah 2
1 (kuning, biru)	17	29
2 (biru, kuning)	15	31

Setelah dibandingkan, walaupun awalnya lebih sedikit, pemilihan warna biru muda terlebih dahulu ternyata memberikan keuntungan yang lebih besar. Meskipun hanya sedikit, apabila hal ini terus dilakukan, ada kemungkinan hasilnya akan semakin optimal.

C. Panjang Garis Batas

Hal tersebut membawa pada algoritma yang ketiga, yaitu *greedy* berdasarkan panjang garis batas. Pada algoritma ini, warna yang akan dipilih adalah warna yang akan membentuk hasil yang memiliki garis batas paling panjang pada setiap langkah. Garis batas yang panjang akan berpengaruh pada jumlah kotak yang bisa dipengaruhi pada langkah selanjutnya.

Pseudocode untuk strategi ini pun tidak jauh berbeda. Namun, pada strategi yang ini, akan digunakan fungsi kelayakan, karena pada saat kotak yang belum terhubung tinggal sedikit, panjang garis batas tidak akan bertambah, bahkan akan berkurang. Misalnya ketika hanya tinggal sebuah kotak yang belum terhubung, garis batasnya adalah 4, sedangkan jika dihubungkan, garis batasnya akan habis menjadi 0.

```
procedure greedy3
kamus
   c : kandidat
   warnapilihan : warna;
   w : warna {warna di papan[1][1]}
   i : integer;
   max : integer;
   tmp: integer;

algoritma
   max := 0;
   w := papan[1][1];
   while not win do
      for i:=1 to 6
         if c[i] ≠ w then
            papantemp := papan;
            tmp := pjg(w,c[i],1,1,0);
            if tmp > max
            and luas > luasawal then
               max := temp;
               warnapilihan := c[i];
      pilihwarna(warnapilihan);
```

Fungsi kelayakan yang dilakukan adalah pada bagian `luas > luasawal`. Fungsi seleksinya adalah fungsi pjg, yang berfungsi untuk menghitung panjang garis batas yang dihasilkan jika suatu warna dipilih. Skema fungsi panjang adalah sebagai berikut.

```
function pjg(
   wlama,wcoba: warna;
   i,j,status: integer)→integer

kamus
   hasil : integer

algoritma
   hasil := 0;
   if (dalam batas papan) then
      if papan[i][j] = wcoba then
         papan[i][j] := sudah_dikunjungi;
      hasil :=
            pjg(wlama,wcoba,i+1,j) +
            pjg(wlama,wcoba,i,j+1) +
            pjg(wlama,wcoba,i-1,j) +
            pjg(wlama,wcoba,i,j-1);
      else if papan[i][j] = wlama
      and status = 0 then
         papan[i][j] := sudah_dikunjungi;
         hasil :=
            pjg(wlama,wcoba,i+1,j) +
            pjg(wlama,wcoba,i,j+1) +
            pjg(wlama,wcoba,i-1,j) +
            pjg(wlama,wcoba,i,j-1);
      else if papan[i][j] ≠ 'x' then
         hasil := 1;
   return hasil;
```

Namun, bagaimanapun, algoritma ini pun belum tentu akan menghasilkan jumlah langkah minimal karena belum ada rumus matematika yang membuktikannya.

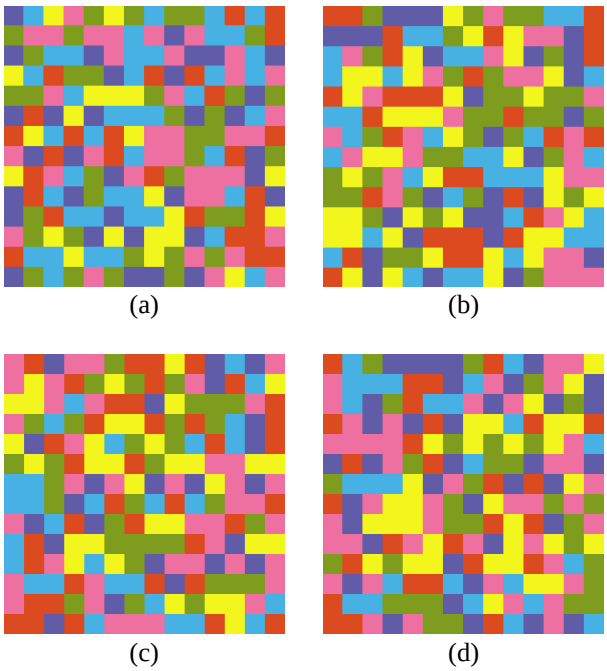
IV. Hasil dan Pembahasan

Setelah meninjau ketiga jenis algoritma *greedy* tersebut, akan lebih baik jika dilakukan uji coba untuk mengetahui algoritma mana yang lebih baik dan efisien. Untuk itu, ketiga algoritma tersebut telah diujicobakan untuk menyelesaikan 4 permainan. Hasil dari uji coba tersebut ditampilkan dalam tabel berikut.

Tabel 2 Hasil pengujian algoritma *greedy*

Permainan	**Strategi *greedy***		
	Jumlah	Luas	Panjang batas
1	27	26	24
2	30	21	23
3	28	28	24
4	29	28	25

Tabel tersebut menyatakan jumlah langkah yang diperlukan untuk setiap algoritma *greedy* dalam menyelesaikan masing-masing permainan. Papan permainan yang diujicobakan dapat dilihat pada Gambar 6 berikut.



(a) (b)

(c) (d)

Gambar 6 Keempat papan yang digunakan untuk uji coba, (a) permainan 1, (b) permainan 2, (c) permainan 3, dan (d) permainan 4

Dari hasil tersebut, dapat dilihat bahwa secara umum *greedy* berdasarkan panjang batas menghasilkan jumlah langkah yang lebih efisien. Meskipun begitu, algoritma tersebut tidak selalu menghasilkan nilai yang optimal. Contohnya pada permainan ke-2, dapat dilihat bahwa *greedy* berdasarkan luas memberikan hasil yang lebih optimum daripada *greedy* berdasarkan panjang batas.

Selain itu, *greedy* berdasarkan jumlah cenderung memberikan hasil yang paling tidak optimum, bahkan tidak dapat menyelesaikan masalah, karena pada permainan sebenarnya jumlah langkah dibatasi sebanyak 25 langkah. Namun, misalnya pada permainan 3 *greedy* berdasarkan jumlah berhasil menyaingi *greedy* berdasarkan luas, dan tidak menutup kemungkinan pada suatu kasus strategi yang pertama ini akan menghasilkan jumlah langkah yang optimal.

Meskipun algoritma *greedy* belum dapat memberikan hasil yang optimal, algoritma ini memberikan solusi yang hampir optimal dengan cara yang cukup sederhana. Cara ini mungkin lebih baik daripada cara lain yang pasti akan memberikan hasil optimal dengan waktu pencarian yang lama, misalnya BFS.

V. Simpulan dan Saran

Dari pembahasan di atas, dapat disimpulkan:

1. Algoritma *greedy* yang diberikan belum bisa memberikan solusi yang optimal untuk permainan *flood it*, tapi dapat memberikan solusi yang hampir optimal.
2. Algoritma *greedy* berdasarkan panjang batas memberikan hasil yang cenderung paling baik.

Saran untuk pengembangan selanjutnya :

1. Dengan kelebihan dan kekurangan masing-masing, ketiga strategi *greedy* dapat dikombinasikan dengan komposisi yang tepat sehingga keuntungannya maksimal.
2. Algoritma ini dapat digunakan sebagai pendekatan heuristik dalam menggunakan algoritma yang lebih kompleks.

Daftar Pustaka

[1] Rinaldi Munir, "Diktat Kuliah IF3051 Strategi Algoritma", Institut Teknologi Bandung, 2009, hal 26–31.

[2] http://www.labpixies.com/gadget_page.php?id=10, 5 Desember 2010

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, 8 Desember 2010

Athia Saelan
13508029