Analisis Permainan "*FLIP*" Menggunakan Algoritma Program Dinamis

Tina Yuliani Ayuningsih
Program studi Teknik Informatika
Institut Teknologi Bandung
Jl. Ganesha 10 Bandung
e-mail: if15057@students.if.itb.ac.id

ABSTRAK

Permainan / *games* merupakan hal yang sulit dipisahkan dari dunia kita sehari-hari. Dewasa ini, rata-rata permainan yang dibuat baik secara tradisional maupun modern dapat dicari solusi optimalnya menggunakan algoritma. Salah satu penerapan algoritma dalam pencarian solusi suatu permainan adalah algoritma program dinamis dalam pencarian solusi optimal di permainan *flip*. Algoritma program dinamis termasuk salah satu algoritma yang dapat menghasilkan solusi optimal. Algoritma program dinamis telah banyak diterapkan dalam kehidupan sehari-hari seperti: perkalian matriks, optimalisasi layout halaman, permainan *checkerboard, flip*, dll. Bila dibandingkan dengan algoritma *greedy*, algoritma program dinamis lebih efisien dan optimal karena algoritma program dinamis menggunakan lebih dari satu rangkaian keputusan yang memenuhi prinsip optimalitas yang akan dihasilkan.

Kata kunci: program dinamis, *flip*, *greedy*.

1. PENDAHULUAN

Program dinamis (*dynamic programming*) adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (*step*) atau tahapan (*stage*) sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Pada penyelesaian persoalan dengan metode ini:

1. Terdapat sejumlah berhingga pilihan yang mungkin
2. Solusi pada setiap tahap dibangun dari solusi tahap sebelumnya
3. Kita menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

Pada program dinamis, rangkaian keputusan yang optimal dibuat dengan menggunakan prinsip optimalitas yang berbunyi: jika solusi total optimal, maka bagian solusi sampai tahap ke k juga optimal. Prinsip optimalitas berarti bahwa jika kita bekerja dari tahap k k tahap k+1, kita dapat menggunakan hasil optimal dari tahap k tanpa harus kembali ke tahap awal.

Karakteristik program dinamis:

1. Persoalan dapat dibagi menjadi beberapa tahap (stage), yang pada setiap tahap hanya diambil satu keputusan
2. Masing-masing tahap terdiri dari sejumlah status(state) yang berhubungan dengan tahap tersebut.
3. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
4. Ongkos (cost) pada suatu tahap meningkat secara teratur (steadily) dengan bertambahnya jumlah tahapan.
5. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut
6. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.
7. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap status pada tahap k memberikan keputusan terbaik untuk setiap status pada tahap k+1
8. Prinsip optimalitas berlaku pada persoalan tersebut

Dua pendekatan yang digunakan dalam *Dynamic Progamming* adalah maju (*forward* atau *up-down*) dan mundur (*backward* atau *bottom-up*).

Misalkan x_1, x_2, …, x_n menyatakan peubah (*variable*) keputusan yang harus dibuat masing-masing untuk tahap 1, 2, …, *n*. Maka,

a. Program dinamis maju: Program dinamis bergerak mulai dari tahap 1, terus maju ke tahap 2, 3, dan seterusnya sampai tahap *n*. Runtunan peubah keputusan adalah $x_1, x_2, \ldots, x_n$.

b. Program dinamis mundur: Program dinamis bergerak mulai dari tahap *n*, terus mundur ke tahap *n* – 1, *n* – 2, dan seterusnya sampai tahap 1. Runtunan peubah keputusan adalah $x_n, x_{n-1}, \ldots, x_1$.

Empat langkah yang dilakukan dalam mengembangkan algoritma program dinamis:

1. Karakteristikkan struktur solusi optimal

2. Definisikan secara rekursif nilai solusi optimal
3. Hitung nilai solusi optimal secara maju atau mundur
4. Konstruksi solusi optimal. Solusi optimal yang dihasilkan oleh program dinamis dapat lebih dari satu buah.

Gambar 1. Alat Permainan Flip

2. *Flip*

Flip adalah sebuah permainan tradisional yang dibuat dari beberapa batang kayu dan dua buah dadu. Aturan permainan *Flip* adalah sebagai berikut:

1. 12 batang kayu yang masing-masing bertuliskan angka 1-12 ditata secara berurutan.
2. Dua buah dadu dilempar sehingga didapat suatu angka.
3. Pilih batang kayu yang berjumlah sama dengan nilai yang tertera di dadu yang sudah dilempar, dan baliklah batang yang dipilih.
4. Jika ingin membatalkan pilihan, dapat dilakukan dengan membalik batang sekali lagi sehingga diperoleh posisi semula.
5. Lanjutkan permainan sehingga semua batang telah dibalik ke bawah, dan itu artinya Anda menang.
6. Namun, jika ada batang yang tidak dibalik ke bawah hingga akhir permainan, itu artinya Anda kalah.

2.1 Contoh program *Flip*

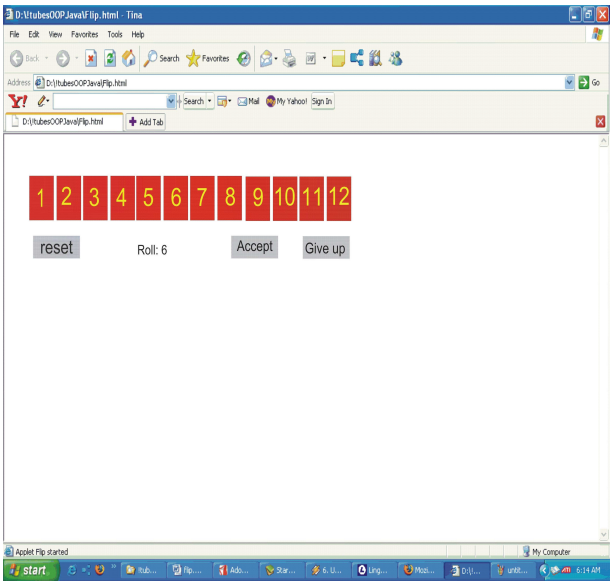
Seiring dengan berkembangnya zaman, permainan *Flip* tidak hanya berasal dari kayu, tetapi dapat dibuat dengan menggunakan bahasa pemrograman. Salah satu aplikasi *Flip* dalam bentuk perangkat lunak adalah *applet* Flip yang dibuat dengan menggunakan bahasa JAVA dan diimplementasikan di *browser*.

Contoh algoritma programnya adalah sebagai berikut:

```
class FlipTiles {

  boolean flip[] = new boolean[12];
  boolean temp[] = new boolean[12];
  int current_roll;
  int current_sum;
  Random r1 = new Random();
  int win;

  FlipTiles() {
    int i;
    for (i=0;i<12;i++) {
      this.flip[i] = false;
      this.temp[i] = false;
    }
    current_roll=
(int)(r1.nextDouble()*6)+(int)(r1.nextDoubl
e()*6)+2;
    current_sum = 0;
    win = 0;
  }
  void Reset(int i) {
    this.flip[i-1] = false;
  }

  void Reset() {
    int i;
    for (i=0;i<12;i++) {
      this.flip[i] = false;
      this.temp[i] = false;
    }
    current_roll                          =
(int)(r1.nextDouble()*6)+(int)(r1.nextDoubl
e()*6)+2;
    current_sum = 0;
    win = 0;
  }

  void Accept () {
    if (current_sum!=current_roll) return;
    for (int i=0;i<12;i++) {
      if (temp[i])
        flip[i] = true;
      temp[i] = false;
    }
    current_roll=
(int)(r1.nextDouble()*6)+(int)(r1.nextDoubl
e()*6)+2;
    current_sum = 0;
    boolean done=true;
    for (int i=0;i<12;i++) {
      if (!flip[i]) done=false;
    }
    if (done) this.win = 1;
  }

  void Flip(int i) {
    if (!this.flip[i-1]) {
```

```
      if (!this.temp[i-1]) {
        this.temp[i-1] = true;
        current_sum+=i;
      }
      else {
        this.temp[i-1] = false;
        current_sum-=i;
      }
    }
  }
}
```

Tampilan dari aplikasi *Flip* berupa *applet* adalah sebagai berikut:



Gambar 2. *Applet Flip*

2.2. Skenario Permainan *Flip*

```
Skenario kalah:
Posisi: 1 2 3 4 5 6 7 8 9 10 11 12
Dadu: 9
Keputusan: Balik 1 dan 8
Posisi: _ 2 3 4 5 6 7 _ 9 10 11 12
Dadu: 8
Keputusan: Balik 2 dan 6
Posisi: _ _ 3 4 5 _ 7 _ 9 10 11 12
Dadu: 7
Keputusan: Balik 7
Posisi: _ _ 3 4 5 _ _ _ 9 10 11 12
Dadu: 6
Tidak bisa melakukan flip. Pemain kalah

Skenario menang:
Posisi: 1 2 3 4 5 6 7 8 9 10 11 12
Dadu: 9
Keputusan: Balik 9
Posisi: 1 2 3 4 5 6 7 8 _ 10 11 12
Dadu: 8
Keputusan: Balik 8
Posisi: 1 2 3 4 5 6 7 _ _ 10 11 12
Dadu: 7
Keputusan: Balik 2 dan 5
Posisi: 1 _ 3 4 _ 6 7 _ _  10 11 12
Dadu: 10
Keputusan: Balik 4 dan 6
Posisi: 1 _ 3 _ _ _ 7 _ _  10 11 12
Dadu: 12
Keputusan: Balik 12
Posisi: 1 _ 3 _ _ _ 7 _ _  10 11 _
Dadu: 12
Keputusan: Balik 1 dan 11
Posisi: _ _ 3 _ _ _ 7 _ _  10 _ _
Dadu: 10
Keputusan: Balik 10
Posisi: _ _ 3 _ _ _ 7 _ _  _ _ _
Dadu: 12

Keputusan: Balik 3 dan 7
Posisi: _ _ _ _ _ _ _ _ _  _ _ _
Semua angka sudah dibalik. Pemain menang
```

2.3.Analisis Permainan *Flip*

Flip merupakan permainan yang menggunakan algoritma program dinamis yang memiliki 2^{12}=4096 status. Dari analisis yang dilakukan, diperoleh fakta menarik bahwa *Flip* merupakan permainan yang tidak rasional karena kesempatan menang adalah 1 kali setiap 276 kali permainan.

Untuk menentukan ketanggapan(responsibilitas) permainan *Flip* ini, diperlukan petunjuk untuk membandingkan peluang kemenangan. Dua buah petunjuk yang dapat digunakan adalah *Lex-Min (Lexically Minimum)*dan *Lex-Max*(*Lexically Maximum*). Sebagai contoh, di posisi awal, untuk nilai dadu 8, dengan menggunakan *Lex Max*, maka angka yang dipilih adalah 4-3-1 sedangkan *Lex Max*, angka yang dipilih adalah 8. Kedua jawaban ini sama-sama memiliki alasan yang kuat. Berdasarkan hasil percobaan, dengan menggunakan *Lex-Min*, peluang untuk menang adalah 1 setiap 7671 permainan. Sedangkan dengan menggunakan *Lex-Max*, peluang untuk menang adalah 1 setiap 290 permainan.

Dengan menggunakan *Lex-Max*, jika nomor batang yang dihasilkan dari pemutaran dadu belum dibalik, maka, solusi optimal adalah membalik 1 batang yang bertuliskan sama dengan jumlah angka dadu. Namun, jika angka yang dihasilkan dadu tidak ada di batang , maka dapat digantikan dengan pasangan angka yang terdiri 2 atau 3 buah angka yang berbeda.

Angka dadu: 8

Angka Flip yang dipilih: 8 atau 7-1 atau 2-6, atau 3-5.

Perbedaan antara *Lex-Max* dan solusi optimal sangat signifikan. Hal itu dapat dibuktikan dengan beberapa contoh sbb:

```
 1 2 _ 4 5 6 _ 8 _ __ __ __
Roll 11
Optimal (6,5): 1 in 8.6;
Lex-Max (8,2,1): 1 in 15.1
1 2 _ 4 5 6 _ _ _ 10 11 __
Roll 9
Optimal (5,4): 1 in 119.3;
Lex-Max (6,2,1): 1 in 243
```

```
1 _ 3 _ _ 6 _ 8 _ 10 __ __
Roll 9
Optimal (6,3): 1 in 29.4;
Lex-Max (8,1): 1 in 44.7
```

Program dinamis dapat dimodifikasi dengan mudah untuk melacak jumlah kesalahan putaran (bagian pembahasan program dinamis). Tabel berikut ini menunjukkan bagaimana jumlah kesalahan yang diperbolehkan mempengaruhi kemenangan.

Meskipun memperbolehkan 5 kesalahan membuat permainan ini realistis dalam hal kesempatan menang dan memperbolehkan 10 kesalahan masih tidak dapat membuat permainan ini terlalu mudah untuk dimainkan. Kelemahan dari permainan ini adalah permainan ini kurang responsif terhadap aksi pemain.

Tabel1: Peluang Memperbolehkan Kesalahan(kesempatan menang 1 dalam…)

Kesalahan	Optimal	*Lex-Min*	*Lex-Max*
0	276	7671	290
1	103	1434	109
2	55	470	58
3	35	206	37
4	24	107	26
5	18	64	19
6	14	41	15
7	11	29	12
8	9.3	21	9.9
9	7.9	16	8.4
10	6.8	13	7.2

Analisis di atas mengasumsikan bahwa pemain dipaksa untuk membuat gerakan ketika gerakan yang legal masih ada. Kita dapat membuat permainan ini lebih responsif pada aksi pemain dengan mmperbolehkan pemain untuk mengklaim sebuah kegagalan secara strategi meskipun gerakan legal masih memungkinkan. Sebagai contoh jika hanya ada 2 dan 5 yang belum dibalik, jika dadu yang dilempar pemain menghasilkan angka 5, ini lebih baik untuk menganggap sebagai kegagalan dan mengharapkan angka 7 pada dadu yang diputar selanjutnya daripada membalik 5 kemudian mengharapkan angka dadu 2 di putaran selanjutnya.

Dalam kasus ini, pengambilan keputusan untuk *Lex Min* dan *Lex Max* tidak berubah, tapi keputusan optimal melakukan beberapa hal secara strategi untuk memperbolehkan kesalahan.

Tabel2: Peluang Memperbolehkan Kesalahan (kesempatan menang 1 dalam…)

Failures	Optimal	*Lex-Min*	*Lex-Max*
0	276	7671	290
1	100	1434	109
2	52	470	58
3	32	206	37
4	22	107	26
5	16	64	19
6	12	41	15
7	9.7	29	12
8	7.9	21	9.9
9	6.7	16	8.4
10	5.8	13	7.2

Perbedaan keseluruhan dengan *Lex-Max* masih tidak begitu ideal, sehingga *Lex-Max* menyisakan awal petunjuk yang baik. Seperti permainan dasar, ada beberapa status di mana perbedaan antara *Lex-Max* dan optimal masih ekstrem. Sebagai contoh, jika masih ada kesalahan yang diperbolehkan, pemain seharusnya tidak pernah meninggalkan 1 sebagai satu-satunya batang yang belum dibalik karena tidak ada putaran dadu yang dapat membuat batang ini dibalik.

Memperbolehkan kesalahan terlihat meningkatkan kepelikan permainan. Tanpa adanya kesalahan, ada 1297 status/putaran dadu (dari 4096*12=49152) di mana keputusan optimal bukanlah *Lex-Max*. Dengan 10 kesalahan yang diperbolehkan, ini berarti meningkatkan 14,975 dari 491,520.

2.4.Pemodelan dan Implementasi Permainan *Flip*

Formulasi *Flip*

Pertama, kita mendefinisikan status dan keputusan. Sebuah status akan berhubungan dengan pengaturan batang dan diberikan berupa 12 string 0-1, biasanya dinyatakan sebagai s. s[i]=1 jika batang ke i belum dibalik, sedangkan s[i]=0 jika batang ke i sudah dibalik. Sehingga ada 2^12=4096 status. Kami akan menyebut semua sudah dibalik sebagai status 0.

Definisikan f(s) sebagai kemungkinan menang di bawah permainan optimal pada status s. Sebuah keputusan d(s,r) (di mana d(s,r) adalah status yang dipilih setelah permulaan di s dan pemutaran r) harus dibuat untuk setiap status s dan pemutaran dadu r. Nyatakan peluang pemutaran r sebagai p(r).

$$F(s)=S_r p(r)f(d(s,r)) \quad (1)$$

Untuk menentukan d(s,r), kita perlu menentukan status mana yang mungkin dimulai dari s dan putaran r. Anggap R(r) sebagai set dengan panjang 12 string 0-1 di mana masukan ”1” bertambah sampai r. Sebagai contoh, untuk r=5, R(r) terdiri dari susunan string:

```
000010000000
100100000000
011000000000
```

Nyatakan $s \pounds t$ untuk dua vektor s dan t dengan cara normal, dengan setiap kmponen s kurang dari atau sama dengan komponen t.

$$d(s,r) = \text{argmin } u \in R(r), u \leq s \{f(s-u)\} \quad (2)$$

Dalam rumus (2), u merepresentasikan keputusan yang layak untuk status s dan putaran r. Jika set untuk argmin

kosong, maka peluang yang bersangkutan untuk menang adalah 0.

Akhirnya, ketika semua batang sudah dibalik, kita sukses, jadi kita dapat mendasarkan pogram dinamis kita di:

$$F(0)=1 \quad (3)$$

Dari poin (1),(2),(3) kita memiliki formulasi program dinamis kita. Untuk menyelesaikan program dinamis kita, kita dapat menghitung f(s) secara rekursif. Kita perlu mengatur perhitungan kita supaya hanya menggunakan nilai kalkulasi sebelumya. Ini dilakukan dengan mengatur status dalam susunan yang meningkat jumlah ”1” dalam representasi mereka karena begitu sebuah batang sudah dibalik, tidak dapat dibalik kembali.

Kode yang digunakan untuk menyelesaikan rekursif ini menggunakan struktur data yang efisien untuk memodelkan permasalahan ini. Nilai-nilai R(r) disimpan dalam list berkait (*linked list*), dan semua nilai status dan putaran dikemas dalam nilai integer, sebagai contoh nilai 9 digunakan untuk merepresentasikan string 110001000000.

Hanya diperlukan sedikit perubahan untuk memformulasikan versi flip di mana kesalahan putaran diperbolehkan. Ruang status harus meliputi jumlah kesalahan (dalam bentuk (s,b) di mana b adalah jumlah kesalahan). Nilai kesalahan meningkat ketika tidak ada keputusan yang layak untuk sebuah status dan putaran atau ketika mendeklarasikan kesalahan sebagai keputusan optimal.

.

IV. KESIMPULAN

Program dinamis (*dynamic programming*) adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (*step*) atau tahapan (*stage*) sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan.

Optimalisasi kemenangan dalam permainan *Flip* dapat dianalisis menggunakan program dinamis, Yang dirumuskan sbb:

$$F(s)=S_r p(r)f(d(s,r)) \quad (1)$$

$$d(s,r) = \text{argmin } u \in R(r), u \leq s \{f(s-u)\} \quad (2)$$

$$F(0)=1 \quad (3)$$

Pendekatan pencarian solusi dari permainan *Flip* dapat dilakukan dengan pendekatan *Lex Max*, *Lex Min*, dan pendekatan optimal.

REFERENSI

[1]Munir, Rinaldi, *Diktat kuliah IF2251 Strategi Algoritmik*, Bandung, 2005

[2]Michael,”*Building a Better Game through Dynamic Programming*”, Graduate School of Industrial Administration Carnegie Mellon University,2005

[3]http://en.wikipedia.org/wiki/Dynamic_programming .Tanggal akses: 22 Mei 2007 pukul 12.57.