Strategi Memenangkan Permainan Domino dengan Mengunakan Algoritma Brute Force dan Greedy

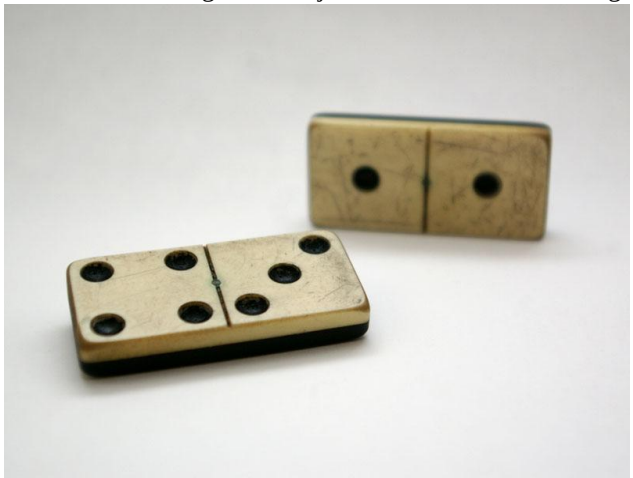
Edwin Romelta - 13508052
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
If18052@students.if.itb.ac.id

***Abstract*—Makalah ini membahas tentang memengankan permainan domino dengan algoritma brute force dan algoritma greedy. Meskipun banyak algoritma yang mulai muncul dalam penyelesaian berbagai macam masalah. Algoritma brute force dan greedy masih cukup mangkus dalam penyelesaiannya yang dapat menyelesaikan permasalahan yang sangat banyak. Meskipun hasilnya yang mungkin tidak optimal, algoritma ini masih cukup digemari oleh banyak orang karena sifatnya yang tidak sulit dimengerti dan banyak permasalahan yang dapat diselesaikannya. Sudah sangat lama domino dimainkan olah masyarakat. Banyak cara strategi permainan domino yang dimainkan oleh orang –orang. Meskipun begitu tidak ada dari seluruh strategi itu yang dipastikan dapat untuk memenangkan permainan domino.**

***Index Terms*—domino,algoritma,brute force,greedy.**

I. INTRODUCTION

Domino adalah permain yang mengunakan balok atau kartu. Tiap kartu atau balok domino meminiki 2 buah nilai. Mulai dari 0|0 sampai dengan 6|6. Jika pada kartu atau balok tersebut memiliki nilai yang sama maka balok atau kartu tersebut disebut juga dengan balak.Dengan total balok atau kartu sebanyak 28 buah. Jumlah pemain permainan ini mulai dari 2 orang sampai dengan 4 orang. Pada makalah ini yang akan dibahas adalah permainan domino dengan jumlah 4 orang.

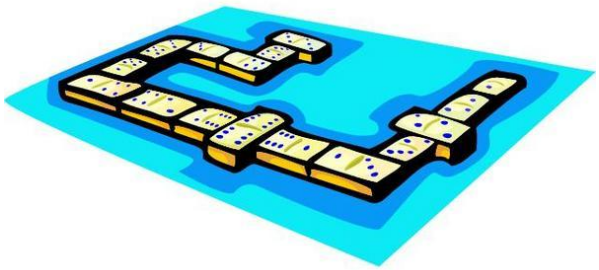


Cara bermain domino sangat mudah. Setiap pemain akan diberikan 7 buah balok yang akan ia gunakan. Pemain yang pertama kali bermain harus manaruhkan salah satu balok balak yang dimilikinya, jika pemain pertama tidak memiliki balak maka pemain pertama tidak dapat berjalan dan giliran permain berikutnya yang jalan. Jika sudah ada 1 balok yang keluarkan, maka pemain berkutnya hanya dapat menurunkan balok yang memiliki nilai sama dengan salah satu nilai pada balok yang ada. Lalu pemain selanjutnya dan seterusnya hanya dapat mengisi pada ujung-ujung balok yang belum disii. Sehingga hanya ada 2 buah tempat yang dapat disi oleh pemain selanjutnya. Pemenang dari permainan domino adalah pemain yang pertama kali menghabiskan balok yang dimilikinya. Jika pada permainan tersebut tidak ada pemain yang dapat menghabiskan baloknya terlebih dahulu dan tidak ada lagi pemain yang dapat menaruhkan baloknya maka permainan diangap selesai, dengan pemenang yang memiliki nilai balok yang paling sedikit.

II. ALGORITMA PENYELESAIAN

Banyak sekali algoritma yang dapat menyelesaikan permainan domino ini. Dari banyak algoritma tersebut tidak ada yang dapat memastikan pemain domino untuk selalu menang dalam permainan. Karena bagaimana pun juga faktor keberuntungan juga masuk dalah permainan ini.

Dari banyak algoritma penyelesaian yang dapat menyelesaikan permainan ini. Hanya akan dibahas 2 algoritma yaitu algoritma brute force dan greedy.



Algortima Brute Force

Pada algortima ini pemain akan langsung mengeluarkan balok yang pertama kali ia temukan tanpa dipilih besar nilai balok yang akan dikeluarkannya. Yang penting balok yang dikeluarkan itu valid.

Pseudecode Brute Force

```
  Function isValid(Balok b) -> integer
{mengecek apakah balok b dapat
ditempatkan disalah satu ujung
Mengembalikan 0 jika tidak dapat
dikeluarkan mengembalikan 1 untuk
ujung atas mengembalikan 2 untuk ujung
bawah}
DEKLARASI

ALGORITMA
  If(TOP == b.up)
      ->1
  If(BOT = b.down)
      ->2
  ->0

  Procedure SolveWithBrute()-> Balok
  {Mengeluarkan balok dari pemain}
  DEKLARASI

  ALGORITMA
  for i=1 to CurrentBalok
      if(isValid(Balok[i]) != 0)
            --CurrentBalok
            -> Balok[i]
  -> NULL
```

Dari kode diatas dapat dilihat bahwa algorima brute force tidak mempedulikan nilai yang keluar. Algoritma ini hanya mementingkan pemain dapat mengeluarkan kartu secepat mungkin.

Algoritma Greedy

Berbeda dengan algoritma brute force. Algoritma greedy memilih balok mana yang paling pantas untuk dikeluarkan. Karena algoritma ini memilih sebelum dikeluarkan tentu saja ada kriteria balok yang pantas untuk dikeluar.

Beberapa contoh kriteria yang pantas untuk dikeluarkan adalah :

- Greedy by Value
- Greedy by Balak
- Greedy by Count

Greedy by Value

Greedy by Value akan mengeluarkan balok yang mempunyai nilai yang paling terbesar terlebih dahulu. Sehingga jumlah nilai balok yang masih ada di pemain berkurang secara maksimal.

0	0						
1	1	2					
2	2	3	4				
3	3	4	5	6			
4	4	5	6	7	8		
5	5	6	7	8	9	10	
6	6	7	8	9	10	11	12
Up/Lef	0	1	2	3	4	5	6

Menurut gambar diatas semakin ke kanan bawah berarti balok tersebut memiliki nilai yang paling besar.

Dengan mengunakan greedy ini jika permainan selesai tanpa ada pemain yang mengeluarkan semua baloknya , algoritma ini menambahkan kemungkinan pemain untuk menang, karena balok – balok yang nilainya besar telah dikeluarkan pada saat balok tersebut mungkin untuk dikeluarkan. Meskipun begitu, tidak ada yang menjamin bahwa pemain pasti menang jika tidak ada balok lagi yang dapat dikeluarkan oleh pemain lainnya. Dan algoritma ini tidak membantu meningkatkan kemungkinan tidak berakhir dengan salah satu pemain berhasil mengeluarkan seluruh baloknya.

Pseudecode Greedy by Value

```
  Function isValid(Balok b) -> integer
{mengecek apakah balok b dapat
ditempatkan disalah satu ujung
Mengembalikan 0 jika tidak dapat
dikeluarkan mengembalikan 1 untuk
ujung atas mengembalikan 2 untuk ujung
bawah}
DEKLARASI

ALGORITMA
  If(TOP == b.up)
      ->1
  If(BOT = b.down)
      ->2
  ->0

  Function Value(Balok b) ->integer
  {mengembalikan jumlah nilai pada
suatu balok}
  DEKLARASI

  ALGORITMA
  -> b.up+b.down

  Procedure SolveWithGreedy()-> Balok
  {Mengeluarkan balok dari pemain}
  DEKLARASI
  Balok temp
```

```
  ALGORITMA
  temp <- NULL
  for i=1 to CurrentBalok
      if(isValid(Balok[i]) != 0)
            if(Value(Balok[i])        >
Value(temp))
                  temp<- Balok[i]
  if(temp != NULL)
      --CurrentBalok
  ->temp
```

Greedy by Value juga bisa dibuat dengan menyusun terlebih dahulu balok – balok yang diterima sesuai dengan jumlah nilai balok yang terbesar sampai yang terkeci. Lalu sisanya sama dengan algoritma Brute Force. Dengan begitu ketika bermain pemain dapat berjalan lebih cepat dibandingkan dengan algoritma greedy. Tetapi membutuhkan waktu untuk menyusun terlebih dahulu.

Pseudecode Sorted by value Brute Force

```
  Procedure SortbyMax(Balok[] balok)
  {mengesort balok dari terbesar ke
terkecil}
  DEKLARASI
  Balok temp
  Balok swap
  Int it
  ALGORITMA
  for i=1 to 7
      temp <- NULL
      it<-NULL
      for j=i to 7
            if(balok[j] > temp)
                  temp<-balok[j]
                  it<-j
      swap<-balok[i]
      balok[i]<-temp
      balok[it]<-swap

  Function isValid(Balok b) -> integer
{mengecek apakah balok b dapat
ditempatkan disalah satu ujung
Mengembalikan 0 jika tidak dapat
dikeluarkan mengembalikan 1 untuk
ujung atas mengembalikan 2 untuk ujung
bawah}
DEKLARASI

ALGORITMA
  If(TOP == b.up)
      ->1
  If(BOT = b.down)
      ->2
  ->0

  Procedure SolveWithBrute()-> Balok
  {Mengeluarkan balok dari pemain}
  DEKLARASI

  ALGORITMA
  for i=1 to CurrentBalok
      if(isValid(Balok[i]) != 0)
            --CurrentBalok
                  -> Balok[i]
  -> NULL
```

Greedy by Balak

Karena balak sangat mungkin untuk tidak dapat keluar maka kita perlu mengeluarkan balak terlebih dahulu. Sehingga ketika kita punya kesemaptan untuk mengeluarkan balak maka kita keluarkan balak tersebut telebih dahulu. Balak perlu dikeluarkan jika bisa karena kemungkinan keluarnya sangat kecil dibandingkan balok yang lain yang memiliki 2 buah nilai. Belum lagi karena jumlah balok pada satu nilai itu adalah 7 atau ganjil. Sehingga jika semua berpasangan balaklah yang paling mungkin tidak punya pasangan. Kecuali berpasangan diantara 2 balok yang memiliki nilai yang sama dengan balok balak.

Jika seluruh balak sudah dikeluarkan baru kita menyelesaikan peermainan selanjutnya seusai dengan algoritma yang lainnya. Seperti dengan greedy by value, atau greedy by count.

Pseudecode Greedy by balak

```
  Function isValid(Balok b) -> integer
{mengecek apakah balok b dapat
ditempatkan disalah satu ujung
Mengembalikan 0 jika tidak dapat
dikeluarkan mengembalikan 1 untuk
ujung atas mengembalikan 2 untuk ujung
bawah}
DEKLARASI

ALGORITMA
  If(TOP == b.up)
      ->1
  If(BOT = b.down)
      ->2
  ->0

  Fuction IsBalak(balok b)-> boolean
  {mengembalikan true jika balok yang
dimasukan adalah balak dan
mengembalian false jika balok yang
dimasukan bukan balak}
  DEKLARASI

  ALGORITMA
  If(b.up == b.down)
      ->true
  ->false

  Function Value(Balok b) ->integer
  {mengembalikan jumlah nilai pada
suatu balik}
  DEKLARASI

  ALGORITMA
  -> b.up+b.down

  Procedure SolveWithGreedy()-> Balok
  {Mengeluarkan balok dari pemain}
  DEKLARASI
```

```
  Balok temp

  ALGORITMA
  For i=1 to CurrentBalok
      If(isValid(balok[i])          AND
isbalak(balok[i]))
            ->balok[i]

  temp <- NULL
  for i=1 to CurrentBalok
      if(isValid(Balok[i]) != 0)
            if(Value(Balok[i])        >
Value(temp))
                  temp<- Balok[i]
  if(temp != NULL)
      --CurrentBalok
  ->temp
```

Algoritma ini juga bisa di percepat dengan mengsort balak terlebih dahulu baru di sort dengan algoritma greedy yang dilakukan jika semua balak yang dimiliki pemain telah keluar.

Psoudecode Sorted by balak Brute Force

```
  Procedure BalakFirst(Balok[] b)
  {mengurutkan balak pada awal – awal
urutan}
  DEKLARASI
  Balok swap
  Int sum

  ALGORITMA
  sum <- 0
  for i=1 to 7
      if(isBalak(b[i]))
            ++sum
            Swap<-b[sum]
            b[sum]<-b[i]
            b[i]<-swap


  Function isValid(Balok b) -> integer
{mengecek   apakah   balok   b   dapat
ditempatkan disalah satu ujung
Mengembalikan   0   jika   tidak   dapat
dikeluarkan   mengembalikan   1   untuk
ujung atas mengembalikan 2 untuk ujung
bawah}
DEKLARASI

ALGORITMA
  If(TOP == b.up)
      ->1
  If(BOT = b.down)
      ->2
  ->0

  Fuction IsBalak(balok b)-> boolean
  {mengembalikan true jika balok yang
dimasukan     adalah     balak     dan
mengembalian false jika balok yang
dimasukan bukan balak}
  DEKLARASI
```

```
  ALGORITMA
  If(b.up == b.down)
      ->true
  ->false

  Procedure SolveWithBrute()-> Balok
  {Mengeluarkan balok dari pemain}
  DEKLARASI

  ALGORITMA
  for i=1 to CurrentBalok
      if(isValid(Balok[i]) != 0)
            --CurrentBalok
            -> Balok[i]
  -> NULL
```

Greedy by count

Greedy ini menerapkan safe play. Yaitu kita mengeluarkan balok yang nilai salah satu bagiannya paling terbanyak dari pada opsi yang bisa dikeluarkan yang ada. Dengan begitu sedikit kemungkinan kita terkena pass. Sehingga kita dapat mengeluarkan semua balok yang ada di tangan kita. Namun karena kita bermain safe sangan memungkinkan untuk balok yang bernilai besar sangat lama keluar. Sehingga jika seluruh pemain tidak ada lagi yang dapat berjalan, pemain mungkin masih memiliki balok yang nilainya besar.

Pseudecode Greedy by count

```
  Function max(int x, int y)->int
  {mengembalikan   masukan   mana   yang
tebesar diantara x dan y jika besar
inputan x megnembalikan 1, dan jika
lebih    besar    inputan    y    maka
mengembalikan 2}
  DEKLARASI

  ALGORITMA
  If(x>=y)
      ->1
  ->2

  Function      findbalok(int      x,int
y,balok[] b)->balok
  {mencari balok dar b yang memiliki
nilai x|y atau y|x}
  DEKLARASI

  ALGORITMA
  For i=1 to currentbalok
      If(b[i].up == x AND b[i].down ==
y OR b[i].up == y AND b[i].down == x)
      -> balok[i]
  ->NULL

  Function listmax(int[7] l)->int[7]
  {mengembalikan list yang mengurutkan
index l bedasarkan terbesar ke yang
terkecil}
  DEKLARASI
  Int[7] L
  Int max
```

```
  ALGORITMA
  For i=1 to 7
      Max <- -1
      For j=i to 7
            If(l[j]>max)
                  Max<-j
      L[i]<-j
      L[i]<- -1
  ->L

  Function findelse(balok[] b,int x)->
balok
  {mengembalikan nilai yang salah
satunya memiliki nilai yang masih
banyak serupa pada tangan}
  DEKLARASI
  Int[7] list
  Int[7] idx
  ALGORITMA
  For i = 1 to 7
      Int[i]<-0
  For i = i to 7
      ++list[b[i].up]
      ++list[b[i].down]
  Idx <- Listmax(list)
  For i=1 to 7
      If(findbalok(x,idx[i])  !=  NULL)
            -> findbalok(x,idx[i])

  Function count(balok[] b,int value)
->integer
  {menghitung jumlah balok yang
nilainya sama dengan nilai yang
dicari}
  DEKLARASI
  int sum

  ALGORITMA
  For i=1 to currentbalok
      If(b[i].up == value OR b[i].down
== value)
            ++sum
  -> sum

  Function isValid(Balok b) -> integer
{mengecek apakah balok b dapat
ditempatkan disalah satu ujung
Mengembalikan 0 jika tidak dapat
dikeluarkan mengembalikan 1 untuk
ujung atas mengembalikan 2 untuk ujung
bawah}
DEKLARASI

ALGORITMA
  If(TOP == b.up)
      ->1
  If(BOT = b.down)
      ->2
  ->0

  Function Value(Balok b) ->integer
  {mengembalikan jumlah nilai pada
suatu balok}
  DEKLARASI
```

```
  ALGORITMA
  -> b.up+b.down

  Procedure SolveWithGreedy()-> Balok
  {Mengeluarkan balok dari pemain}
  DEKLARASI

  ALGORITMA
  If(count(Balok[],TOP),count(Balok[],B
OT)) == 1
       ->Findelse(Balok[],TOP)
  ->Findelse(Baloc[],BOT)
```

Seluruh strategi algoritma permain domino diatas adalah strategi bermain secara defensive atau bertahan. Tentu saja ada startegi permain dengan offensive.

Kita dapat bermain secara offensive seperti salah satu startegi berikut:

- Menghitung balok – balok yang belum keluar
  Kita dapat melawan musuh dengan menghitung balok – balok yang belum keluar. Misalnya, kita memiliki balok bernilai 4 sebanyak 4 buah. Kita menahan balok yang banyak tersebut dan menunggu sampai 2 buah balok bernilai 4 keluar. Ketika balok – balok tersebut keluar berarti tinggal 1 lagi balok bernilai 4 yang belum keluar. Kita tinggal menunggu salah satu unjung di isi dengan balok yang bernilai 4 tersebut. Saat salah satu ujung bernilai 4, kita tidak perlu mengisinya terlebih dahulu, melainkan kita membuat kedua ujung pada papan tersebut menjadi 4, dengan demikian seluruh pemain selain kita pas, sehingga kita dapat dengan mudah memenangkan pemain, karena seluruh pemain selain kita lebih banyak jumlah baloknya dari pada kita.
  Namun cara ini cukup riskan, karena jika balok – balok tersebut tidak keluar dan misalnya balok yang jumlahnya banyak di kita tersebut adalah balok bernilai 6 maka, cukup riskan jika kita terlalu lama menahan balok tersebut untuk keluar.
- Mengingat balok yang tidak dimiliki lawan
  Dengan memingat balok yang tidak dimilik salah satu pemain kita dapat membuat pemain tersebut kalah. Tetapi untuk memperkirakan balok yang tidak dimiliki lawan tersebut tidaklah gampang. Kita harus menghitung balok – balok yang sudah dikeluarkan. Biasanya tidak disengaja lawan tersebut pass karena tidak memiliki balok yang harus dikeluarkan. Barulah kita mengingat balok yang tidak dimiliki lawan tersebut. Tetapi cara ini cukup buruk karena cukup lama untuk mengungu pemain tersebut tidak memiliki salah satu nilai balok. Karena untuk menciptakan kejadian tersebut tidaklah mudah karena permbagian balok yang tidak merata yang disebabkan oleh balok yang diacak terlebih dahulu sebelum di bagikan. Startegi ini juga bisa digunakan juka

kita sudah mengetahui balok apa yang tidak dimiliki lawan. Sebelum itu kita tidak dapat berbuat apa-apa.

- Menahan ujung yang hanya dapat diisi dengan balok milik kita
  Cara ini cukup sulit dilakukan juga. Kita harus menghitung balok – balok yang sudah keluar. Jika sudah 6 buah balok yang keluar dan balok untuk mengisi ujung tersebut adalah balok kita barulah kita bisa mengunakan strategi ini. Sebisa mungkin kita mengisi ujung lain karena pemain lain hanya bisa mengisi ujung lain. Dengan begitu kemungkinan pemain lain pass sangat meningkat, karena mereka hanya mingisi satu ujung saja. Sedangkan kita masih bisa mengisi ujung lain jika kita sudah tidak dapat mengisi ujung yang lain lagi. Namun, jika kita sudah mengisi ujung tersebut startegi ini sudah tidak dapat digunakan lagi.

Strategi algoritma offensive ini hanya bisa digunakan jika kondisinya dipenuhi, untuk menciptakan kondisi tersebut sangatlah sulit, kondisi tersebut juga tidak pasti keluar di tiap permainan. Sehingga cukup riskan untuk digunakan.
Maka sebelum kita dapat mengunakan startegi offensive kita tidak bisa apa – apa selain mengunakan strategi defensive. Setelah condisi dipenuhi barulah kita menganti ke strategi offensive. Dan jika strategi offensive tidak dapat digunakan lagi, maka kita terpaksa kembali ke strategi defensive.

III. Analisis Algoritma

Dari beberapa algoritma yang dibahas dibagian kedua ,yaitu :

- Brute Force
- Greedy by Value
- Greedy by Balak
- Greedy by Count

Tidak ada yang dapat memastikan untuk selalu menang.

Brute Force merupakan algoritma yang terlemah karena tidak berfikir untuk mengeluarkan balok yang mana, hanya mengutamakan cepatnya mengeluarkan balok.

Greedy by Value, Balak, count hanya memikirkan diri sendiri, tidak mempertimbangkan kemungkinan yang terjadi dari lawan. Padahal kemungkinan tersebut dapat dihitung.

Dari seluruh algoritma yang ada di atas, seluruh algoritma tersebut memliki sifat defensive pada permainan. Karena tidak ada dari algritma diatas yang membuan lawan kalah. Melainkan menahan diri untuk menang.

Untuk membuat algoritma offensive atau memlawan pemain lain. Diperlukan banyak perhitungan, seperti menghitung balok yang sudah keluar dan memperkirakan kemungkinan siapa yang memiliki balok yang belum keluar tersebut. Bisa juga mengingat pemain yang pass dan mengingat balok yang tidak dia miliki dan membuat kedua ujung pada papan menjadi kedua nilai tersebut.

Namun hal itu cukup sulit untuk diimplementasikan. Karena untuk membuat algoritma offensive, pemain harus menentukan pilihannya untuk menyerang pemain lain atau bertahan untuk menang. Untuk membuat strategi algoritma seperti itu tentu saja sangat mudah karena kita harus memperhitungkan, berapa balok yang keluar, berapa banyak balok yang keluar, balok yang tidak dimiliki pemain lain, balok – balok yang dimiliki kita, permainan sudah pada turn keberapa, dan lain lain.

Untuk itu kita perlu menangani pergantian strategi di tengah permainan. Dan hal ini cukup sulit. Meskipun begitu, masih mungkin untuk pergantian strategi ini untuk diimplementasika caranya.

VI. Kesimpulan

Untuk memengakan permain domino kita tidak bisa hanya bertahan tetapi kita perlu menyerang. Dan untuk menang juga tidak bisa menyerang saja, tatapi perlu juga bertahan. Sehingga kita harus dapat memilih kapan kita harus ganti strategi. Tidak menutup kemungkinan untuk ganti startegi dari salah satu bertahan ke strategi bertahan lainnya, juga sebaliknya untuk memenangkan permainan. Sehingga dengan kata lain :

- Diperlukan keseimbangan atara offensive-defensive bermain.
- Tidak ada satupun strategi yang memastikan untuk selalu menang
- Faktor keberuntungan sangant berpengaruh pada permainan ini dalam balok yang didapatkan pada awal – awal dan permerataan balok – balok.

References

[1] Ir. Rinaldi M.T,Diktat Kuliah Strategi Algoritma,pp 22 - 84

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

ttd

Edwin Romelta - 13508052