Penerapan Algoritma *Brute Force* dan *Backtracking* pada Permainan *Skycraper*

Zulhendra Valiant Janir (13510045)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13510045@std.stei.itb.ac.id

***Abstrak*—*Skycraper* merupakan permainan logika yang dapat mengasah keterampilan dan pola pikir pemain. *Skycraper* dikemas dalam bentuk matriks sederhana tetapi sangat menantang. Aneka tipe soal dapat dipecahkan sesuai kemampuan pemain tergantung tingkat kesulitan persoalan. Ada pun cara menyelesaikan permainan ini yaitu dengan menggunakan algoritma. Pada dasarnya soal pada permainan ini dapat dipecahkan dengan analisis heuristik, tapi pada realisasinya belum tentu dengan heuristik dapat menyelesaikan kondisi–kondisi yang tidak diketahui langkah berikutnya. Dengan menerapkan metode algoritma seperti *Brute Force* dan *Backtracking*, permainan logika seperti *Skycraper* dapat diselesaikan dengan mudah.**

***Kata Kunci*—*Skyscraper*, *Brute Force*, *Backtracking*.**

I. PENDAHULUAN

Dalam menyelesaikan suatu permasalahan, dibutuhkan strategi yang dapat menyelesaikan permasalahan secara solutif. Tentunya dalam suatu permasalahan memiliki solusi yang beraneka ragam, misalnya perbedaan dari faktor lama mendapatkan solusi dan keoptimalan solusi. Agar permasalahan yang terjadi dapat terpecahkan dengan solusi yang baik, dibutuhkanlah strategi algoritma yang cocok dan mangkus.

x		1			x
2					3
1					
4					1
x			1		x

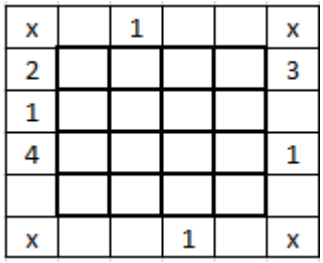
Gambar 1. Soal *Skyscraper*

x		1			x
2	3	4	2	1	3
1	4	3	1	2	
4	1	2	3	4	1
	2	1	4	3	
x			1		x

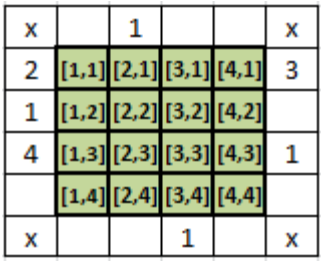
Gambar 2. Solusi *Skyscraper*

Permainan ini tentu saja memiliki aturan yang membuatnya menjadi suatu permaianan yang menarik. Matriks yang terbuat selalu berbentuk persegi dengan sisi sepanjang n. Tiap kotak berukuran 1 x 1 harus terisi dengan sebuah angka berkisar antara 1 sampai dengan n. Selain itu, pada suatu kotak berukuran 1 x 1 dengan kotak lainnya yang memiliki indeks kolom maupun indeks baris yang sama dengan kotak tersebut harus memiliki nilai angka yang berbeda. Artinya, bagi dua kotak berukuran 1 x 1 yang tidak memiliki indeks kolom yang sama maupun tidak memiliki indeks baris yang sama diperbolehkan untuk diisi dengan angka yang sama, hal ini berlaku juga untuk hubungan kotak berukuran 1 x 1 yang terletak secara diagonal. Aturan berikutnya adalah aturan di mana suatu kolom atau baris dapat dilihat dari dua prospek berbeda, yaitu prospek kanan dan kiri untuk tipe observasi baris dan prospek atas dan prospek bawah untuk tipe observasi kolom. Tiap tipe observasi di suatu kolom atau baris dapat memiliki suatu nilai maupun tidak memiliki nilai. Nilai ini berkisar antara 1 sampai dengan n. Untuk observasi yang tidak memiliki nilai, dapat direpresentasikan dengan 0. Nilai ini merepresentasikan berapa banyak kotak berukuran 1 x 1 yang dapat terlihat dari titik observasi tersebut pada kolom atau baris yang beersangkutan pada domain matriks jawaban. Angka yang berada pada domain matriks jawaban merepresentasikan tinggi kotak tersebut. Jika di belakang kotak tersebut terdapat kotak dengan nilai yang lebih kecil, maka kotak di belakangnya tidak akan terlihat dari titik observasi yang berda di depannya, dan begitu pula sebaliknya jika di belakang kotak tersebut terdapat kotak dengan nilai yang lebih besar.

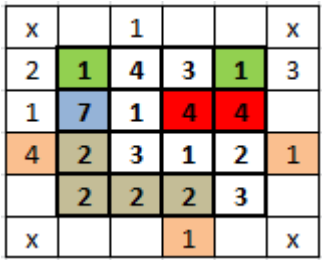
Persoalan *Skycraper* telah disusun sedemikian baiknya sehingga dalam pengerjaannya hanya akan ditemukan tepat satu buah solusi.



Gambar 3. Contoh ukuran matriks n=4

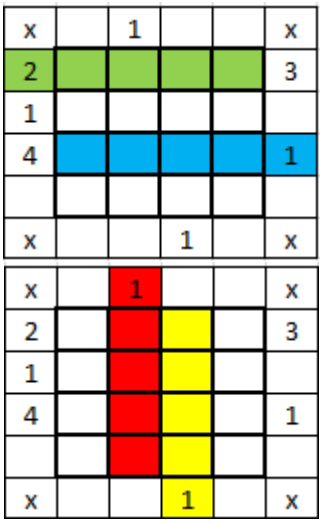


Gambar 4. Penomoran indeks

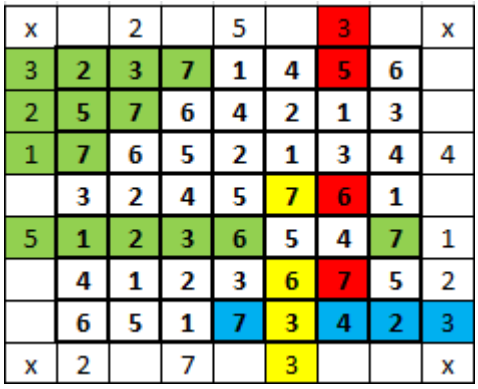


Gambar 5. Kesalahan pengisian

Gambar 6. Empat tipe observasi



Gambar 7. Pembagian domain pada titik observasi

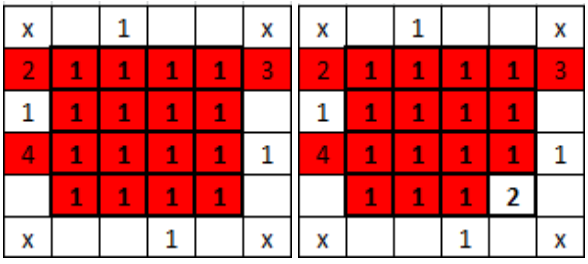


Gambar 8. Kotak berukuran 1 x 1 yang terlihat dari suatu titik observasi

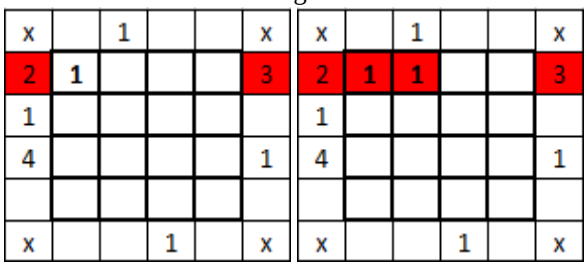
Permainan ini bersifat polinomial di mana domain pada langkah selanjutnya dari pencarian solusi masih dapat diperkirakan karena tidak ada perubahan atau kemunculan kemungkinan baru pada tiap langkah. Maka dari itu pada permainan *Skycraper* kali ini, dibutuhkan strategi terbaik agar metode yang digunakan dapat menghasilkan solusi yang memuaskan, yaitu dengan memecahkan persoalan serta membandingkan algoritma antara algoritma *Brute Force* dengan algoritma *Backtracking*.

II. Dasar Teori

Terdapat dua algoritma yang akan dibandingkan untuk memecahkan masalah pada permaianan *Skyscraper* kali ini. Keduanya merupakan algoritma sekuensial di mana tahap pencarian solusi merupakan tahap pencarian solusi sebelumnya dengan suatu komponen yang diubah menjadi tahap pencarian yang lebih baik.



Gambar 9. Langkah *Brute Force*

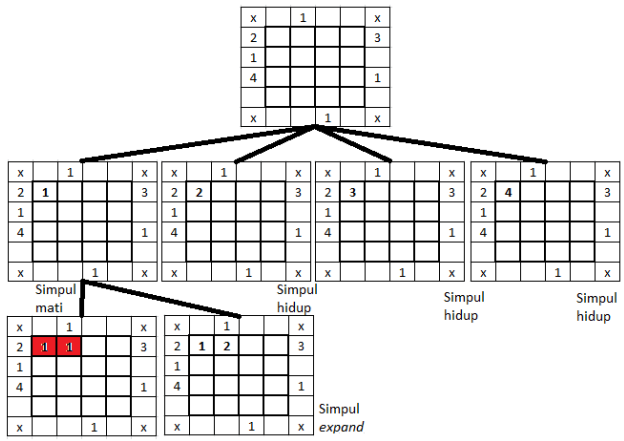


Gambar 10. Langkah *Backtracking*

Algoritma *Brute Force* adalah algoritma lempang yang bertujuan untuk mendapatkan solusi dari sebuah masalah secara sederhana, langsung, dan jelas. Semua kemungkinan solusi akan dicoba dan divalidasi hingga mendapatkan solusi yang benar-benar sesuai dengan aturan permasalahan. Pada umumnya, algoritma ini

tidak cerdas dan tidak mangkus, karena kebutuhannya dalam jumlah besar dalam melakukan pencarian solusi, terutama pada masalah dengan kebutuhan yang besar. Algoritma ini juga dijuluki sebagai algoritma naif. Walaupun begitu, algoritma ini selalu menjadi dasar pencarian algoritma atau pengembangan algoritma yang lebih mangkus. Selain itu, pada algoritma yang lebih mangkus lainnya, algoritma ini dapat menjadi basis dari sebuah permasalahan pada algoritma lainnya. Dari segi implementasi, algoritma ini lebih mudah untuk diimplementasikan karena kesederhanaannya pada aspek pola pikir dan metode pemecahan masalah.

Sedangkan untuk algoritma *Backtracking* merupakan algoritma berbasis *Deepening First Search*. Algoritma ini merupakan algoritma perbaikan dari algoritma *Brute Force* yang lebih mangkus. Algoritma ini tidak mengecek seluruh kemungkinan yang ada, tetapi lebih mengeksplorasi kemungkinan solusi yang lebih memungkinkan untuk menjadikannya menjadi sebuah solusi. Oleh karena itu, algoritma ini dapat mengurangi waktu pencarian solusi daripada penggunaan algoritma *Brute Force*. Algoritma ini juga merupakan bentuk tipikal dari algoritma rekursif. Algoritma ini memiliki atribut yang harus didefinisikan dalam permasalahan, yaitu solusi persoalan, fungsi pembangkit, dan fungsi pembatas. Pada pohon *Backtracking* terdapat tiga macam simpul, yaitu simpul hidup berupa simpul yang berkemungkinan menuju pada sebuah solusi, simpul *expand* atau simpul yang sedang dijelajahi serta merupakan realisasi dari fungsi pembangkit dan simpul mati yang merupakan simpul yang tidak akan mengarah ke solusi serta telah dibatasi oleh fungsi pembatas. Jika pencarian barakhir pada simpul mati, maka pencarian dilanjutkan dengan simpul hidup



Gambar 11. Pohon keputusan *Backtracking*

III. PENERAPAN ALGORITMA

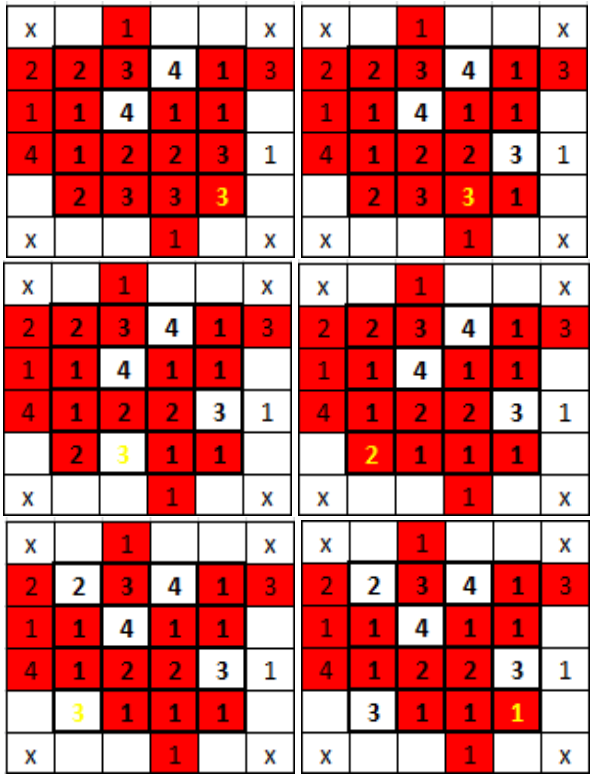
A. Algoritma *Brute Force*

Seperti yang telah dijelaskan sebelumnya, algoritma *Brute Force* akan mencari solusi dengan mencoba semua kemungkinan yang ada. Dalam penerapannya di Skyscraper, algoritma ini memulai dengan mengisi semua kotak jawaban dengan angka 1. Kemudian dengan penomoran kotak adalah 1 sampai dengan n^2, nilai kotak dengan indeks tertinggi akan ditambah dengan bilangan 1. Jika kotak tersebut bernilai n, maka nilai kotak itu menjadi berniali 1 dan kotak dengan indeks = indeks_kotak_tersebut-1 akan ditambah dengan bilangan 1 jika pada kotak yang ditambahkan dengan 1 tidak memiliki nilai sebesar n. Jika memiliki nilai sebesar n, maka nilai kotak tersebut diganti dengan bilangan 1 dan algoritma mengecek kotak dengan indeks = indeks_kotak_tersebut – 1, dan seterusnya. Jika nilai pada kotak dengan indeks=1 yang bernilai n mendapat giliran untuk ditambah dengan 1, maka persoalan dinyatakan tidak memiliki solusi sama sekali.

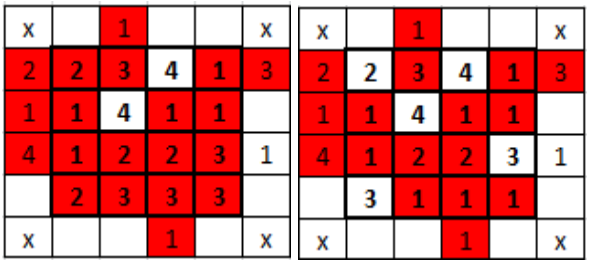
x		1			x
2	1	2	3	4	3
1	5	6	7	8	
4	9	10	11	12	1
	13	14	15	16	
x			1		x

Gambar 12. Penomoran indeks

Dalam setiap satu proses penambahan selalu diiringi dengan proses pengecekan apakah jawaban telah mencapai solusi yang diinginkan.



Gambar 13. Enam langkah dalam satu proses penambahan



Gambar 14. Satu proses penambahan

Solusi ditemukan jika jawaban sesuai dengan aturan *Skycraper*. Dalam ilustrasi berupa gambar di atas, jika tidak ada kotak observasi maupun jawaban yang berwarna merah, maka solusi telah ditemukan.

```
Procedure NextBruteForce(){
{Prosedur untuk mencapai kondisi
domain jawaban berikutnya pada Brute Force}
Kamus
  i,j : integer
  flag : boolean
Algoritma
  i=GetSize()-1
  j=GetSize()-1
  if(GetKotak(i,j)>0 and GetKotak(i,j)<GetSize())
     then SetKotak(i,j,Getn(i,j)+1)
  else
     SetKotakn(i,j,1)
     flag=true
     while(flag and Solusi)do
       if(i==0 and j>0)then
         i=GetSize()-1
         j--
       else if(i>0)then
         i--
       else if(i==0 and j==0)
         Solusi=false
       if(GetKotak(i,j)>0 and GetKotak(i,j)<GetSize()
          and Solusi)then
         SetKotak(i,j,GetKotak(i,j)+1)
         flag=false
       else if(GetKotak(i,j)==GetSize() and Solusi)
         SetKotak(i,j,1);
```

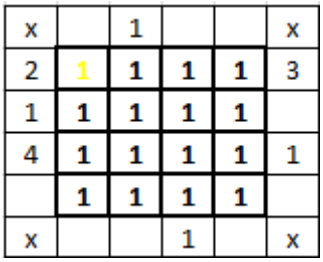
Gambar 15. *Pseudo code* penambahan

```
Function ValidBruteForce() → boolean
{Fungsi yang menyatakan terpenuhinya
syarat solusi pada jawaban}
Kamus
  i,j : integer
  flag : boolean
Algoritma
  i=0
  flag=true;
  while(i<GetSize() and flag)
    if(i<GetSize())then
      flag=CekXUp(i)
      if(not flag)then
         CekXDown(i)
    else
      CekXLeft(i-GetSize())
      if(not flag)then
        CekXRight(i-GetSize())
   i++
  →flag
```

Gambar 16. *Pseudo code* pengecekan *Brute Force*

B. Algoritma *Backtracking*

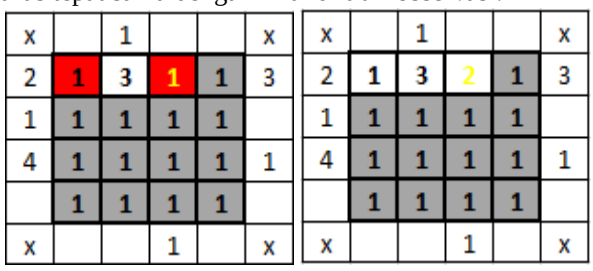
Pada algoritma *Backtracking*, domain jawaban juga memiliki indeks penomoran yang serupa. Perbedaannya yaitu pada letak memulai algoritmanya, yaitu pada domain jawaban dengan kotak berindeks=1. Namun pada awal keberjalanan algoritma ini, seluruh domain jawaban dikosongkan atau diacuhkan. Setelah itu algoritma akan meletakkan diri pada kotak berindeks=1 dan memberi nilai pada kotak tersebut sebesar 1.



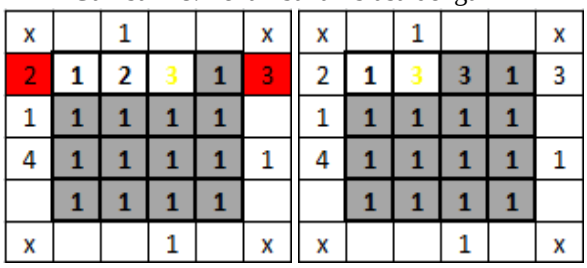
Gambar 17. Inisialisasi algoritma *Backtracking*

Terdapat dua macam proses untuk melanjutkan algoritma Backtracking, yaitu pengubahan kotak dengan indeks<indeks_kotak_saat_ini atau proses perpindahan maju terhadap indeks kotak saat ini.

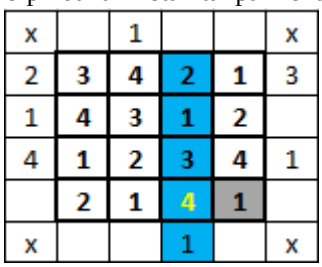
Untuk proses pengubahan kotak sebelum indeks kotak saat ini terjadi jika validasi tidak berhasil yang dikarenakan kotak-kotak dengan indeks sebelumnya masih belum benar. Pemrosesan ini dimulai dengan menambahkan nilai kotak saat ini dengan bilangan 1. Jika suatu saat pada kotak saat ini bernilai>n, maka kotak saat ini diacuhkan dan berpindah ke kotak dengan indeks=indeks_kotak_saat_ini-1, kemudian kotak yang baru diubah nilainya menjadi nilai_sebelumnya+1. Jika masih ada kejadian nilai>n, maka kotak tersebut akan diacuhkan juga, dan seterusnya. Jika indeks_kotak_diacuhkan=1, maka dinyatakan tidak ada solusi pada persoalan *Skycraper* tersebut. Jika pada suatu nilai masih bersifat kurang dari n dan pada proses validasi mendapatkan perlakuan khusus berupa 'acuhkan', maka algoritma tidak akan menambah nilai tersebut, tapi langsung mengacuhkannya. Ini terjadi karena pada suatu titik observasi menyatakan bahwa jumlah kotak yang terlihat telah melebihi yang seharusnya. Jika kotak yang terlihat masih belum seharusnya, maka titik observasi akan menyatakan baik-baik saja, kecuali jika posisi kotak sekarang berada pada tepi kanan atau tepi bawah, jumlah kotak yang terlihat harus tepat sama dengan nilai di titik observasi.



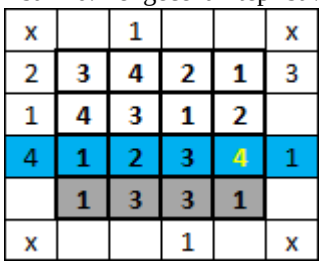
Gambar 18. Penambahan biasa dengan 1



Gambar 19. Perpindahan kotak tanpa menunggu nilai=4

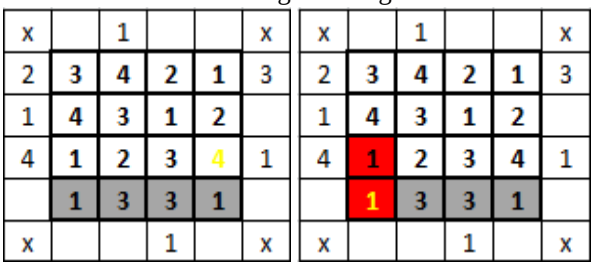


Gambar 20. Pengecekan tepi bawah



Gambar 21. Pengecekan tepi kanan

Untuk proses pemindahan maju terjadi jika domain pada kondisi itu dapat tervalidasi. Maka indeks kotak saat ini akan ditambah dengan bilangan 1.



Gambar 22. Perpindahan maju

Untuk setiap proses pemindahan status kotak sekarang, selalu ada proses validasi. Proses validasi dibagi menjadi dua, yaitu proses validasi bahwa tidak ada kesamaan bilangan pada kolom dan baris yang sama dengan kotak saat ini serta bahwa nilai di empat titik observasi yang melibatkan indeks kotak sekarang masih dapat terpenuhi.

```
Procedure NexBacktracking(i,j:integer[0..GetSize()-1])→boolean
{Prosedur untuk mencapai kondisi domain jawaban berikutnya pada
Backtracking. i dan j adalah posisi baris dan kolom sekarang}
Kamus
  flag : boolean
Algoritma
  if(ValidBacktracking())then
    if(i==GetSize)then
      Returni(0)
      Returnj(j++)
    else if(i==j and i==GetSize())then
      ReturnSolusi()
    else
      Returni(i++)
  else
    if(GetKotak(i,j)>0 and GetKotak(i,j)<GetSize())then
      SetKotak(i,j,GetKotak(i,j)+1)
    else
      flag=true
      while(flag and Solusi)do
        if(i==0 and j>0)then
          i=GetSize()-1
          j--
        else if(i>0)then
          i--
        else if(i==0 and j==0)
          Solusi=false
        if(GetKotak(i,j)>0 and GetKotak(i,j)<GetSize() and Solusi)
           then
          SetKotak(i,j,GetKotak(i,j)+1)
          flag=false
        else if(GetKotak(i,j)==GetSize() and Solusi)
          SetKotak(i,j,1)
```

Gambar 23. *Pseudo code* melanjutkan *Backtracking*

```
Function ValidBacktracking(i,j:integer[0..GetSize()-1]) → boolean
{Fungsi yang menyatakan terpenuhinya
syarat solusi pada jawaban, i dan j}
Kamus
  flag : boolean
Algoritma
  flag=true;
  flag=CekDifferentLeft(j)
  if(not flag)then
    flag=CekDifferentUp(i)
  if(not flag)then
    flag=CekXStillPossibleRight(i)
  if(not flag)then
    flag=CekXStillPossibleLeft(i)
  if(not flag)then
    flag=CekXStillPossibleUp(i)
  if(not flag)then
    flag=CekXStillPossibleDown(i)
  →flag
```

Gambar 24. *Pseudo code* validasi algoritma *Backtracking*

V. KESIMPULAN DAN SARAN

Dengan menggunakan dua buah algoritma yang berbeda, persoalan *Skycraper* sesulit apa pun masih dapat ditemukan solusinya. Namun secara kemangkusan, algoritma *Brute Force* terlalu memakan waktu lama dikarenakan pengecekan yang tidak perlu berupa kondisi pengecekan turunan dari kondisi yang salah. Sedangkan untuk *Backtracking* dapat memangkas kondisi yang salah sehingga tidak ada pengecekan turunan dari kondisi yang salah. Kemangkusan ini menyebabkan total waktu pengecekan oleh algoritma *Backtracking* lebih cepat daripada pengecekan oleh algoritma *Brute Force*.

Dengan kemangkusan algoritma *Backtracking*, untuk pencarian solusi dari permasalahan permaianan logika seperti permainan *Skycraper*, lebih baik menggunakan algoritma *Backtracking* daripada algoritma *Brute Force*.

REFERENCES

[1] Munir, Rinaldi. 2006. *Diktat Kuliah IF2251 Strategi Algoritmik.* Program Studi Teknik Informatika ITB, hal 8-11,125-127.
[2] www.conceptispuzzles.com (21 Desember 2012)

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, 21 Desember 2012

Zulhendra Valiant Janir (13510045)