Perbandingan Algoritma *Backtracking* dan Algoritma *Hybrid Genetic* untuk Menyelesaikan Permainan Calcudoku

Who? Michael Adrian
2013730039
michaeladrian39@gmail.com

From? Jurusan Teknik Informatika
Fakultas Teknologi Informasi dan Sains
Universitas Katolik Parahyangan

When? 6 Desember 2016

Perbandingan Algoritma *Backtracking* dan Algoritma *Hybrid Genetic* untuk Menyelesaikan Permainan Calcudoku

Who? Michael Adrian
2013730039
michaeladrian39@gmail.com

From? Jurusan Teknik Informatika
Fakultas Teknologi Informasi dan Sains
Universitas Katolik Parahyangan

When? 6 Desember 2016

Calcudoku

- Salah satu jenis permainan teka-teki aritmatika dan logika
- Dikenal juga sebagai KenKen, KenDoku, atau Mathdoku
- Diciptakan pada tahun 2004 oleh Tetsuya Miyamoto, seorang guru matematika dari Jepang
- Diciptakan untuk melatih kemampuan matematika dan logika dengan cara yang menyenangkan

Sebagai salah satu jenis permainan teka-teki aritmatika dan *grid*, Calcudoku, atau dikenal juga sebagai KenKen, KenDoku, atau Mathdoku, diciptakan pada tahun 2004 oleh seorang guru matematika dari Jepang yang bernama Tetsuya Miyamoto untuk memenuhi tujuannya untuk melatih kemampuan matematika dan logika siswa-siswinya dengan cara yang menyenangkan. Nama KenKen diambil dari kata bahasa Jepang yang berarti kepandaian. Permainan yang mengasah otak ini dengan cepat menyebar ke seluruh Jepang dan Amerika Serikat, menggantikan permainan teka-teki silang di banyak koran. Permainan ini kemudian menjadi sensasi di seluruh dunia setelah munculnya versi *online* dan *mobile* dari permainan teka-teki ini, khususnya menarik untuk pecinta permainan teka-teki angka seperti Sudoku.

Aturan Permainan

- Pemain diberikan sebuah *grid* dengan ukuran $n \times n$
- n biasanya antara 3 sampai dengan 9
- *Grid* ini harus diisi dengan angka 1 sampai dengan n
- Dalam setiap baris setiap angka hanya muncul sekali
- Dalam setiap kolom setiap angka hanya muncul sekali
- *Grid* dibagi ke dalam *cage*
- *Cage* adalah sekelompok sel yang dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel dengan angka tujuan dan operator yang telah ditentukan
- Angka-angka dalam setiap *cage* harus mencapai angka tujuan jika dihitung menggunakan operator yang telah ditentukan
- Angka tujuan dan operasi yang telah ditentukan ditulis di sudut kiri atas *cage*

Aturan Permainan
- Pemain diberikan sebuah *grid* dengan ukuran $n \times n$
- n biasanya antara 3 sampai dengan 9
- *Grid* ini harus diisi dengan angka 1 sampai dengan n
- Dalam setiap baris setiap angka hanya muncul sekali
- Dalam setiap kolom setiap angka hanya muncul sekali
- *Grid* dibagi ke dalam *cage*
- *Cage* adalah sekelompok sel yang dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel dengan angka tujuan dan operator yang telah ditentukan
- Angka-angka dalam setiap *cage* harus mencapai angka tujuan jika dihitung menggunakan operator yang telah ditentukan
- Angka tujuan dan operasi yang telah ditentukan ditulis di sudut kiri atas *cage*

Seperti dalam Sudoku, dalam teka-teki ini, pemain diberikan sebuah *grid* dengan ukuran $n \times n$, dengan n biasanya antara 3 sampai dengan 9. *Grid* ini harus diisi dengan angka 1 sampai dengan n sehingga dalam setiap baris setiap angka hanya muncul sekali, dalam setiap kolom setiap angka hanya muncul sekali. Perbedaannya dengan Sudoku adalah, Calcudoku dibagi ke dalam *cage* (sekelompok sel yang dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel dengan angka tujuan dan operator yang telah ditentukan), dan angka-angka dalam setiap *cage* harus mencapai angka tujuan jika dihitung menggunakan operator yang telah ditentukan. Angka tujuan dan operasi yang telah ditentukan ditulis di sudut kiri atas *cage*.

Operator-Operator Matematika

- Ada 5 kemungkinan operator:
  - $+$ (penjumlahan)
  - - (pengurangan)
  - $\times$ (perkalian)
  - $\div$ (pembagian)
  - $=$ (sama dengan)
- Jika operasi matematika yang ditentukan adalah pengurangan atau pembagian, maka ukuruan *cage* harus berukuran dua sel

Operator-Operator Matematika
- Ada 5 kemungkinan operator:
  - + (penjumlahan)
  - - (pengurangan)
  - × (perkalian)
  - ÷ (pembagian)
  - = (sama dengan)
- Jika operasi matematika yang ditentukan adalah pengurangan atau pembagian, maka ukuruan cage harus berukuran dua sel

Ada lima kemungkinan operator:

1. $+$, sebuah operator *n*-ary yang menandakan penjumlahan.
2. -, sebuah operator biner yang menandakan pengurangan.
3. $\times$, sebuah operator *n*-ary yang menandakan perkalian.
4. $\div$ sebuah operator biner yang menandakan pembagian.
5. $=$, (simbol ini biasanya dihilangkan), sebuah operator uner yang menandakan persamaan.

Jika operasi matematika yang ditentukan adalah pengurangan atau pembagian, maka ukuruan *cage* harus berukuran dua sel. Pada beberapa versi dari teka-teki ini, hanya angka tujuan yang diberikan, dan pemain harus menebak operator dari setiap *cage* untuk menyelesaikan teka-tekinya.

Contoh Permainan

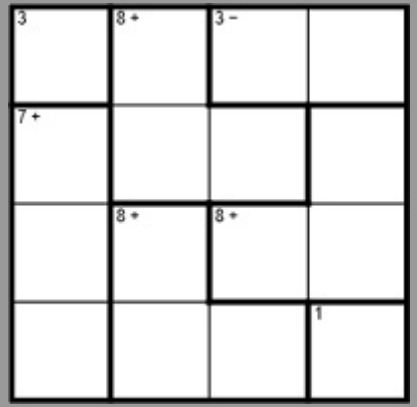

Figure 1: Contoh permainan teka-teki dengan ukuran *grid* 4 x 4 yang belum diselesaikan.

Contoh Permainan

Figure 1: Contoh permainan teka-teki dengan ukuran *grid* 4 x 4 yang belum diselesaikan.

Contoh Solusi

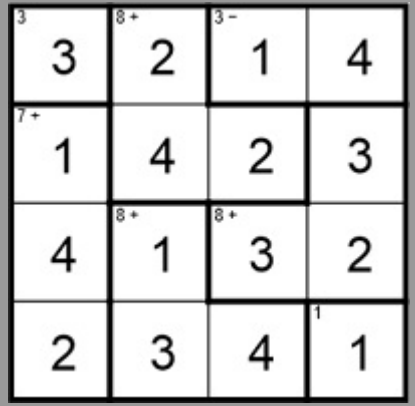

Figure 2: Solusi untuk permainan teka-teki Calcudoku yang diberikan pada Gambar 1.

Contoh Solusi
3 2 1 4
1 4 2 3
4 1 3 2
2 3 4 1
Figure 2: Solusi untuk permainan teka-teki Calcudoku yang diberikan pada Gambar 1.

Permasalahan Utama dalam Menyelesaikan Calcudoku

- Untuk menyelesaikan sebuah teka-teki Calcudoku, pemain harus pertama-tama memahami dua permasalahan utama dari teka-teki ini, yaitu:
  - Angka-angka mana yang harus dimasukkan ke dalam sebuah *cage*
  - Dalam urutan apa angka-angka tersebut harus dimasukkan ke dalam sebuah *cage*
- Cara yang paling mudah untuk menyelesaikan teka-teki ini adalah dengan mengeliminasi angka-angka yang sudah digunakan dan mencoba satu per satu angka yang mungkin (*trial and error*).

Permasalahan Utama dalam Menyelesaikan Calcudoku

- Untuk menyelesaikan sebuah teka-teki Calcudoku, pemain harus pertama-tama memahami dua permasalahan utama dari teka-teki ini, yaitu:
  - Angka-angka mana yang harus dimasukkan ke dalam sebuah cage
  - Dalam urutan apa angka-angka tersebut harus dimasukkan ke dalam sebuah cage
- Cara yang paling mudah untuk menyelesaikan teka-teki ini adalah dengan mengeliminasi angka-angka yang sudah digunakan dan mencoba satu per satu angka yang mungkin (*trial and error*).

Untuk menyelesaikan sebuah teka-teki Calcudoku, pemain harus pertama-tama memahami dua permasalahan utama dari teka-teki ini, yaitu:

1. Angka-angka mana yang harus dimasukkan ke dalam sebuah *cage*
2. Dalam urutan apa angka-angka tersebut harus dimasukkan ke dalam sebuah *cage*

Seperti kebanyakan permainan teka-teki angka, cara yang paling mudah untuk menyelesaikan teka-teki ini adalah dengan mengeliminasi angka-angka yang sudah digunakan dan mencoba satu per satu angka yang mungkin (*trial and error*).

Tahapan Pengisian Calcudoku

- Dalam pengisian teka-teki ini ada dua tahapan, yaitu:
  - Mencari *cage* yang hanya berukuran 1 sel
  - Mencari mencari *cage* yang hanya mempunyai satu kemungkinan kombinasi angka

Tahapan Pengisian Calcudoku
Dalam pengisian teka-teki ini ada dua tahapan, yaitu:
Mencari cage yang hanya berukuran 1 sel
Mencari mencari cage yang hanya mempunyai satu kemungkinan kombinasi angka

Dalam pengisian teka-teki ini ada dua tahapan, yaitu:

1. Mencari *cage* yang hanya berukuran 1 sel, karena *cage* ini tidak menghasilkan pertanyaan angka apa dan urutan apa. Tahap ini adalah tahap yang paling jelas. Contoh, pada Gambar 1, *cage* pada sudut kiri atas dan *cage* pada sudut kanan bawah hanya berukuran 1 sel, dan dapat langsung diisi dengan angka tujuannya.
2. Mencari mencari *cage* yang hanya mempunyai satu kemungkinan kombinasi angka, sehingga masalah angka-angka apa yang harus diisi dalam *cage* tersebut terjawab. Contoh, *cage* pada sudut kanan atas mempunyai aturan "3-", artinya angka tujuannya adalah 3 dengan menggunakan operasi pengurangan. Satu-satunya pasangan angka dari himpunan {1,2,3,4} yang akan menghasilkan angka 3 saat satu angka dikurangkan dari angka yang lainnya adalah {1,4}. Namun masalahnya adalah urutan angka-angka yang harus dimasukkan. Dalam kasus ini, untungnya, sel pada sudut kanan bawah sudah diisi dengan angka 1, maka angka 1 tidak bisa digunakan lagi pada kolom yang paling kanan. Jadi, dengan menggunakan cara eliminasi, sel pada sudut kanan atas harus diisi dengan angka 4 dan sel di sebelah kirinya, yaitu sel pada baris yang paling atas dan kolom ketiga dari kiri, harus diisi dengan angka 1. Hal ini memberikan solusi untuk sel pada baris yang paling atas dan kolom kedua dari kiri, yaitu angka 2, karena angka 2 adalah angka yang

Tahapan Pengisian Calcudoku

- Seiring dengan meningkatnya tingkat kesulitan, langkah berikutnya tidak akan langsung muncul dengan jelas
- Kadang-kadang, pemain mencapai titik dimana langkah berikutnya tidak pasti
- Pemain harus menebak langkah-langkah berikutnya dan melihat apakah langkah ini akan menghasilkan solusinya. Jika tidak, pemain harus mundur kembali ke titik ketidakpastian tersebut.

Tahapan Pengisian Calcudoku

- Seiring dengan meningkatnya tingkat kesulitan, langkah berikutnya tidak akan langsung muncul dengan jelas
- Kadang-kadang, pemain mencapai titik dimana langkah berikutnya tidak pasti
- Pemain harus menebak langkah-langkah berikutnya dan melihat apakah langkah ini akan menghasilkan solusinya. Jika tidak, pemain harus mundur kembali ke titik ketidakpastian tersebut.

Seiring dengan meningkatnya tingkat kesulitan, langkah berikutnya tidak akan langsung muncul dengan jelas. Kadang-kadang, pemain mencapai titik dimana langkah berikutnya tidak pasti. Pemain harus menebak langkah-langkah berikutnya dan melihat apakah langkah ini akan menghasilkan solusinya. Jika tidak, pemain harus mundur kembali ke titik ketidakpastian tersebut.

Mendefinisikan Permasalahan Calcudoku

- Sebuah teka-teki Calcudoku dengan ukuran $n \times n$, dengan n melambangkan jumlah sel dalam satu baris atau kolom, mempunyai n^2 sel
- Sel yang terletak dalam baris b dan kolom k diberi label $C_{b,k} = bn + k$
- Nilai dari sel tersebut adalah $V(C_{b,k}) \in \{1, 2, ..., n\}$.
- Sebuah *cage*, yang diberi label A_i adalah sebuah himpunan dari sel, yaitu $A_i = \{C_{b,k}\}$
- Setiap *cage* terhubung dengan satu operator aritmatika $O_i \in \{+, -, \times, \div\}, =$ dan satu angka tujuan $H_i \in N$

Mendefinisikan Permasalahan Calcudoku

- Sebuah teka-teki Calcudoku dengan ukuran $n \times n$, dengan n melambangkan jumlah sel dalam satu baris atau kolom, mempunyai n^2 sel
- Sel yang terletak dalam baris b dan kolom k diberi label $C_{b,k} = bn + k$
- Nilai dari sel tersebut adalah $V(C_{b,k}) \in \{1, 2, ..., n\}$.
- Sebuah cage, yang diberi label A_i adalah sebuah himpunan dari sel, yaitu $A_i = \{C_{b,k}\}$
- Setiap cage terhubung dengan satu operator aritmatika $O_i \in \{+, -, \times, \div\}, =$ dan satu angka tujuan $H_i \in N$

Sebuah teka-teki Calcudoku dengan ukuran $n \times n$, dengan n melambangkan jumlah sel dalam satu baris atau kolom, mempunyai n^2 sel. Sel yang terletak dalam baris b dan kolom k diberi label $C_{b,k} = bn + k$ dan nilai dari sel tersebut adalah $V(C_{b,k}) \in \{1, 2, ..., n\}$. Sebuah *cage*, yang diberi label A_i adalah sebuah himpunan dari sel, yaitu $A_i = \{C_{b,k}\}$. Setiap *cage* terhubung dengan satu operator aritmatika $O_i \in \{+, -, \times, \div\}$ dan satu angka tujuan $H_i \in N$.

Mendefinisikan Permasalahan Calcudoku

- 3 aturan dalam mendefinisikan masalah dalam Calcudoku adalah sebagai berikut
- $|A_i| = 1 \rightarrow O_i = \phi$, artinya setiap *cage* yang jumlah selnya 1 dengan operasi matematika yang terkait dengan *cage* tersebut bersifat homeomorfik (setara).
- $O_i \in -, \div \rightarrow |A_i| = 2$, artinya jika operasi yang digunakan dalam sebuah *cage* adalah pengurangan atau pembagian, maka jumlah sel dalam *cage* tersebut harus 2.
- $\forall C_{b,k} \rightarrow C_{b,k} \in \exists! A_i$, artinya setiap sel hanya boleh menjadi anggota dari satu dan hanya satu *cage*.

Mendefinisikan Permasalahan Calcudoku
- 3 aturan dalam mendefinisikan masalah dalam Calcudoku adalah sebagai berikut
- $|A_i| = 1 \rightarrow O_i = \phi$, artinya setiap *cage* yang jumlah selnya 1 dengan operasi matematika yang terkait dengan *cage* tersebut bersifat homeomorfik (setara).
- $O_i \in -, \div \rightarrow |A_i| = 2$, artinya jika operasi yang digunakan dalam sebuah *cage* adalah pengurangan atau pembagian, maka jumlah sel dalam *cage* tersebut harus 2.
- $\forall C_{b,k} \rightarrow C_{b,k} \in \exists! A_i$, artinya setiap sel hanya boleh menjadi anggota dari satu dan hanya satu *cage*.

Menurut Johanna, Lukas, dan Saputra, tiga aturan dalam mendefinisikan masalah dalam Calcudoku adalah sebagai berikut:

1. $|A_i| = 1 \rightarrow O_i = \phi$, artinya setiap *cage* yang jumlah selnya 1 dengan operasi matematika yang terkait dengan *cage* tersebut bersifat homeomorfik (setara)
2. $O_i \in -, \div \rightarrow |A_i| = 2$, artinya jika operasi yang digunakan dalam sebuah *cage* adalah pengurangan atau pembagian, maka jumlah sel dalam *cage* tersebut harus 2
3. $\forall C_{b,k} \rightarrow C_{b,k} \in \exists! A_i$, artinya setiap sel hanya boleh menjadi anggota dari satu dan hanya satu *cage*

Mendefinisikan Permasalahan Calcudoku

- Tujuan dari teka-teki ini adalah untuk mencari nilai $V(C_{b,k})$ dan memenuhi persyaratan berikut
- $|A_i| = 1 \wedge C_{b,k} \in A_i \rightarrow V(C_{b,k}) = H_i$, artinya jika sel adalah bagian dari sebuah *cage* yang jumlah selnya 1, maka nilai dari sel tersebut adalah angka tujuan dari *cage* tersebut
- $O_i \in \{-\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow |V(C_{a,b}) - V(C_{p,q})| = H_i$, artinya nilai absolut dari hasil pengurangan nilai kedua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut
- $O_i \in \{\div\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow V(C_{a,b})/V(C_{p,q}) = H_i$, artinya nilai dari hasil pembagian nilai kedua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut
- $O_i \in \{+\} \rightarrow \sum_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya nilai dari hasil penjumlahan dari nilai semua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut
- $O_i \in \{\times\} \rightarrow \prod_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya nilai dari hasil perkalian dari nilai semua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut

Mendefinisikan Permasalahan Calcudoku
- Tujuan dari teka-teki ini adalah untuk mencari nilai $V(C_{b,k})$ dan memenuhi persyaratan berikut
- $|A_i| = 1 \wedge C_{b,k} \in A_i \rightarrow V(C_{b,k}) = H_i$, artinya jika sel adalah bagian dari sebuah cage yang jumlah selnya 1, maka nilai dari sel tersebut adalah angka tujuan dari cage tersebut
- $O_i \in \{-\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow |V(C_{a,b}) - V(C_{p,q})| = H_i$, artinya nilai absolut dari hasil pengurangan nilai kedua sel di dalam cage tersebut adalah angka tujuan dari cage tersebut
- $O_i \in \{\div\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow V(C_{a,b})/V(C_{p,q}) = H_i$, artinya nilai dari hasil pembagian nilai kedua sel di dalam cage tersebut adalah angka tujuan dari cage tersebut
- $O_i \in \{+\} \rightarrow \sum_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya nilai dari hasil penjumlahan dari nilai semua sel di dalam cage tersebut adalah angka tujuan dari cage tersebut
- $O_i \in \{\times\} \rightarrow \prod_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya nilai dari hasil perkalian dari nilai semua sel di dalam cage tersebut adalah angka tujuan dari cage tersebut

Menurut Johanna, Lukas, dan Saputra, tujuan dari teka-teki ini adalah untuk mencari nilai $V(C_{b,k})$ dan memenuhi persyaratan berikut:

1. $|A_i| = 1 \wedge C_{b,k} \in A_i \rightarrow V(C_{b,k}) = H_i$, artinya jika sel adalah bagian dari sebuah *cage* yang jumlah selnya 1, maka nilai dari sel tersebut adalah angka tujuan dari *cage* tersebut.
2. $O_i \in \{-\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow |V(C_{a,b}) - V(C_{p,q})| = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah pengurangan, maka nilai absolut dari hasil pengurangan nilai kedua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut.
3. $O_i \in \{\div\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow V(C_{a,b})/V(C_{p,q}) = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah pembagian, maka nilai dari hasil pembagian nilai kedua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut.
4. $O_i \in \{+\} \rightarrow \sum_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah penjumlahan, maka nilai dari hasil penjumlahan dari nilai semua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut.
5. $O_i \in \{\times\} \rightarrow \prod_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah perkalian, maka nilai dari hasil perkalian dari nilai semua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut

Algoritma *Backtracking*

- Sebuah algoritma umum yang mencari solusi dengan mencoba salah satu dari beberapa pilihan, jika pilihan yang dipilih ternyata salah, komputasi dimulai lagi pada titik pilihan dan mencoba pilihan lainnya
- Untuk bisa melacak kembali langkah-langkah yang telah dipilih, maka algoritma harus secara eksplisit menyimpan jejak dari setiap langkah yang sudah pernah dipilih, atau menggunakan rekursi (*recursion*)
- Rekursi dipilih karena jauh lebih mudah daripada harus menyimpan jejak setiap langkah yang pernah dipilih
- Hal ini menyebabkan algoritma ini biasanya berbasis DFS (*Depth First Search*)

Algoritma *Backtracking*
- Sebuah algoritma umum yang mencari solusi dengan mencoba salah satu dari beberapa pilihan, jika pilihan yang dipilih ternyata salah, komputasi dimulai lagi pada titik pilihan dan mencoba pilihan lainnya
- Untuk bisa melacak kembali langkah-langkah yang telah dipilih, maka algoritma harus secara eksplisit menyimpan jejak dari setiap langkah yang sudah pernah dipilih, atau menggunakan rekursi (*recursion*)
- Rekursi dipilih karena jauh lebih mudah daripada harus menyimpan jejak setiap langkah yang pernah dipilih
- Hal ini menyebabkan algoritma ini biasanya berbasis DFS (*Depth First Search*)

Algoritma *backtracking* adalah sebuah algoritma umum yang mencari solusi dengan mencoba salah satu dari beberapa pilihan, jika pilihan yang dipilih ternyata salah, komputasi dimulai lagi pada titik pilihan dan mencoba pilihan lainnya. Untuk bisa melacak kembali langkah-langkah yang telah dipilih, maka algoritma harus secara eksplisit menyimpan jejak dari setiap langkah yang sudah pernah dipilih, atau menggunakan rekursi (*recursion*). Rekursi dipilih karena jauh lebih mudah daripada harus menyimpan jejak setiap langkah yang pernah dipilih. Hal ini menyebabkan algoritma ini biasanya berbasis DFS (*Depth First Search*).

Algoritma *Backtracking*

- Pertama kali diperkenalkan pada tahun 1950 oleh D.H. Lehmer sebagai perbaikan algoritma *brute force*
- Algoritma ini terbukti efektif untuk menyelesaikan banyak permainan logika karena algoritma itu terutama berguna untuk menyelesaikan masalah-masalah *constraint satisfaction*, di mana sekumpulan objek harus memenuhi sejumlah batasan

Algoritma *Backtracking*

- Pertama kali diperkenalkan pada tahun 1950 oleh D.H. Lehmer sebagai perbaikan algoritma *brute force*
- Algoritma ini terbukti efektif untuk menyelesaikan banyak permainan logika karena algoritma itu terutama berguna untuk menyelesaikan masalah-masalah *constraint satisfaction*, di mana sekumpulan objek harus memenuhi sejumlah batasan

Algoritma *backtracking* pertama kali diperkenalkan pada tahun 1950 oleh D.H. Lehmer sebagai perbaikan algoritma *brute force*. Algoritma ini lalu dikembangkan lebih lanjut oleh R.J. Walker, S.W. Golomb, dan L.D. Baumert. Algoritma ini terbukti efektif untuk menyelesaikan banyak permainan logika (misalnya *tic tac toe*, *maze*, catur, dan lain-lain) karena algoritma itu terutama berguna untuk menyelesaikan masalah-masalah *constraint satisfaction*, di mana sekumpulan objek harus memenuhi sejumlah batasan.

Sifat-Sifat Umum Algoritma *Backtracking*

- Implementasi algoritma *backtracking* memiliki beberapa sifat umum, yaitu:
  - Ruang solusi (*solution space*)
  - Fungsi pembangkit (*generating function*)
  - Fungsi pembatas (*generating function*)

Sifat-Sifat Umum Algoritma *Backtracking*
Implementasi algoritma backtracking memiliki beberapa sifat umum, yaitu:
Ruang solusi (solution space)
Fungsi pembangkit (generating function)
Fungsi pembatas (generating function)

Ruang Solusi

- Solusi untuk masalah ini dinyatakan sebagai sebuah vektor X dengan *n-tuple*:

$$X = (x_1, x_2, ..., x_n), x_i \in S_i$$

di mana adalah mungkin bahwa:

$$S_1 = S_2 = ... = S_n$$

- n adalah jumlah sel dalam satu baris atau kolom
- X adalah sebuah *tuple* yang berukuran n^2, yang mereprentasikan isi dari setiap sel dalam *grid*, dimulai pada sel pada sudut kiri atas, lalu bergerak ke sel di sebelah kanannya dalam baris yang sama, jika sudah mencapai sel yang paling kanan maka bergerak ke sel yang paling kiri pada baris dibawahnya, hingga berakhir di sel pada sudut kanan bawah
- S_i adalah sebuah himpunan yang berisi angka-angka dari 1 sampai n

Ruang Solusi
- Solusi untuk masalah ini dinyatakan sebagai sebuah vektor X dengan *n-tuple*:
$X = (x_1, x_2, ..., x_n), x_i \in S_i$
di mana adalah mungkin bahwa:
$S_1 = S_2 = ... = S_n$
- n adalah jumlah sel dalam satu baris atau kolom
- X adalah sebuah tuple yang berukuran n^2, yang mereprentasikan isi dari setiap sel dalam grid, dimulai pada sel pada sudut kiri atas, lalu bergerak ke sel di sebelah kanannya dalam baris yang sama, jika sudah mencapai sel yang paling kanan maka bergerak ke sel yang paling kiri pada baris dibawahnya, hingga berakhir di sel pada sudut kanan bawah
- S_i adalah sebuah himpunan yang berisi angka-angka dari 1 sampai n

Solution space
Solusi untuk masalah ini dinyatakan sebagai sebuah vektor X dengan *n-tuple*:

$$X = (x_1, x_2, ..., x_n), x_i \in S_i$$

di mana adalah mungkin bahwa:

$$S_1 = S_2 = ... = S_n$$

n adalah jumlah sel dalam satu baris atau kolom. X adalah sebuah *tuple* yang berukuran n^2, yang mereprentasikan isi dari setiap sel dalam *grid*, dimulai pada sel pada sudut kiri atas, lalu bergerak ke sel di sebelah kanannya dalam baris yang sama, jika sudah mencapai sel yang paling kanan maka bergerak ke sel yang paling kiri pada baris dibawahnya, hingga berakhir di sel pada sudut kanan bawah. S_i adalah sebuah himpunan yang berisi angka-angka dari 1 sampai n.

Fungsi Pembangkit

- Fungsi pembangkit X_k dinyatakan sebagai:

$$T(k)$$

di mana $T(k)$ membangkitkan nilai X_k, dari 1 sampai n, yang merupakan komponen dari vektor solusi

Fungsi Pembangkit
- Fungsi pembangkit X_k dinyatakan sebagai:
$T(k)$
di mana $T(k)$ membangkitkan nilai X_k, dari 1 sampai n, yang merupakan komponen dari vektor solusi

Fungsi pembangkit X_k
Fungsi pembangkit X_k dinyatakan sebagai:
$T(k)$
di mana $T(k)$ membangkitkan nilai X_k, dari 1 sampai n, yang merupakan komponen dari vektor solusi.

Fungsi Pembatas

- Fungsi pembatas dinyatakan sebagai:

$$B(x_1, x_2, ..., x_k)$$

di mana B bernilai *true* jika $(x_1, x_2, ..., x_k)$ mengarah ke solusi. Jika B bernilai *true*, maka nilai $x_k + 1$ akan terus dibangkitkan, dan jika B bernilai *false*, maka $(x_1, x_2, ..., x_k)$ akan dibuang

Fungsi Pembatas

- Fungsi pembatas dinyatakan sebagai:

$B(x_1, x_2, ..., x_k)$

di mana B bernilai *true* jika $(x_1, x_2, ..., x_k)$ mengarah ke solusi. Jika B bernilai *true*, maka nilai $x_k + 1$ akan terus dibangkitkan, dan jika B bernilai *false*, maka $(x_1, x_2, ..., x_k)$ akan dibuang

Fungsi pembangkit X_k Fungsi pembatas
Fungsi pembatas dinyatakan sebagai:

$$B(x_1, x_2, ..., x_k)$$

di mana B bernilai *true* jika $(x_1, x_2, ..., x_k)$ mengarah ke solusi. Jika B bernilai *true*, maka nilai $x_k + 1$ akan terus dibangkitkan, dan jika B bernilai *false*, maka $(x_1, x_2, ..., x_k)$ akan dibuang.

Ruang Solusi

- Disusun dalam sebuah struktur berbentuk pohon (*tree*)
- Setiap simpul (*node*) merepresentasikan keadaan masalah
- Setiap sisi (*edge*) diberi label x_i
- Jalur dari akar (*root*) ke daun (*leaf*) merepresentasikan sebuah jawaban yang mungkin
- Semua jalur yang dikumpulkan bersama-sama membentuk ruang solusi
- Struktur pohon ini disebut sebagai *state space tree*

Ruang Solusi
- Disusun dalam sebuah struktur berbentuk pohon (*tree*)
- Setiap simpul (*node*) merepresentasikan keadaan masalah
- Setiap sisi (*edge*) diberi label x_i
- Jalur dari akar (*root*) ke daun (*leaf*) merepresentasikan sebuah jawaban yang mungkin
- Semua jalur yang dikumpulkan bersama-sama membentuk ruang solusi
- Struktur pohon ini disebut sebagai *state space tree*

Ruang solusi untuk algoritma *backtracking* disusun dalam sebuah struktur berbentuk pohon (*tree*), di mana setiap simpul (*node*) merepresentasikan keadaan masalah dan sisi (*edge*) diberi label x_i. Jalur dari akar (*root*) ke daun (*leaf*) merepresentasikan sebuah jawaban yang mungkin, dan semua jalur yang dikumpulkan bersama-sama membentuk ruang solusi. Struktur pohon ini disebut sebagai *state space tree*. Gambar 3 menggambarkan contoh sebuah *state space tree*.

Ruang Solusi

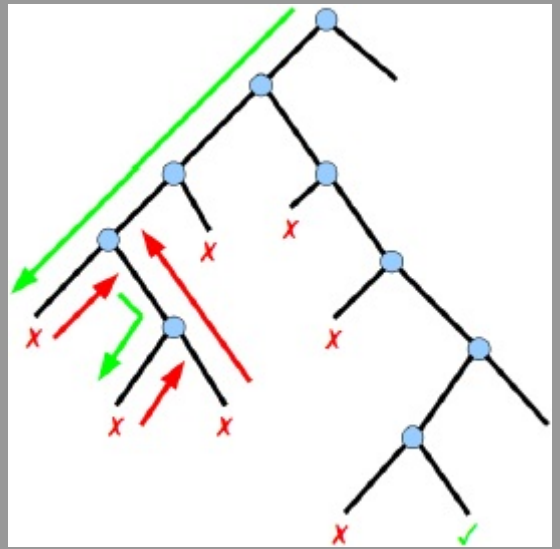

Figure 3: Ilustrasi *State space tree* yang digunakan dalam algoritma *backtracking*

Ruang Solusi

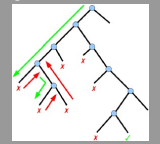

Figure 3: Ilustrasi State space tree yang digunakan dalam algoritma backtracking

Langkah-Langkah Penggunaan *State Space Tree*

- Langkah-langkah dalam menggunakan *state space tree* untuk mencari solusi adalah:
  - Solusi dicari dengan membangun jalur dari akar ke daun menggunakan algoritma DFS
  - Simpul yang terbentuk disebut sebagai simpul hidup (*live nodes*)
  - Simpul yang sedang diperluas disebut sebagai *expand nodes* atau *E-nodes*
  - Setiap kali sebuah *E-node* sedang diperluas, jalur yang dikembangkannya menjadi lebih panjang
  - Jika jalur yang sedang dikembangkan tidak mengarah ke solusi, maka *E-node* dimatikan dan menjadi simpul mati (*dead node*)

Langkah-Langkah Penggunaan *State Space Tree*
- Langkah-langkah dalam menggunakan *state space tree* untuk mencari solusi adalah:
  - Solusi dicari dengan membangun jalur dari akar ke daun menggunakan algoritma DFS
  - Simpul yang terbentuk disebut sebagai simpul hidup (*live nodes*)
  - Simpul yang sedang diperluas disebut sebagai *expand nodes* atau *E-nodes*
  - Setiap kali sebuah *E-node* sedang diperluas, jalur yang dikembangkannya menjadi lebih panjang
  - Jika jalur yang sedang dikembangkan tidak mengarah ke solusi, maka *E-node* dimatikan dan menjadi simpul mati (*dead node*)

Langkah-langkah dalam menggunakan *state space tree* untuk mencari solusi adalah [**?**]:

- Solusi dicari dengan membangun jalur dari akar ke daun menggunakan algoritma DFS.
- Simpul yang terbentuk disebut sebagai simpul hidup (*live nodes*).
- Simpul yang sedang diperluas disebut sebagai *expand nodes* atau *E-nodes*.
- Setiap kali sebuah *E-node* sedang diperluas, jalur yang dikembangkannya menjadi lebih panjang.
- Jika jalur yang sedang dikembangkan tidak mengarah ke solusi, maka *E-node* dimatikan dan menjadi simpul mati (*dead node*).

Langkah-Langkah Penggunaan *State Space Tree*

- Lanjutan dari slide sebelumnya:
  - Fungsi yang digunakan untuk mematikan *E-node* adalah implementasi dari fungsi pembatas
  - Simpul mati tidak akan diperluas
  - Jika jalur yang sedang dibangun berakhir dengan simpul mati, proses akan mundur ke simpul sebelumnya
  - Simpul sebelumnya terus membangkitkan simpul anak (*child node*) lainnya, yang kemudian menjadi *E-node* baru
  - Pencarian selesai jika simpul tujuan tercapai

Langkah-Langkah Penggunaan *State Space Tree*
- Lanjutan dari slide sebelumnya:
- Fungsi yang digunakan untuk mematikan *E-node* adalah implementasi dari fungsi pembatas
- Simpul mati tidak akan diperluas
- Jika jalur yang sedang dibangun berakhir dengan simpul mati, proses akan mundur ke simpul sebelumnya
- Simpul sebelumnya terus membangkitkan simpul anak (*child node*) lainnya, yang kemudian menjadi *E-node* baru
- Pencarian selesai jika simpul tujuan tercapai

Langkah-langkah dalam menggunakan *state space tree* untuk mencari solusi adalah [**?**]:

- Fungsi yang digunakan untuk mematikan *E-node* adalah implementasi dari fungsi pembatas.
- Simpul mati tidak akan diperluas.
- Jika jalur yang sedang dibangun berakhir dengan simpul mati, proses akan mundur ke simpul sebelumnya.
- Simpul sebelumnya terus membangkitkan simpul anak (*child node*) lainnya, yang kemudian menjadi *E-node* baru.
- Pencarian selesai jika simpul tujuan tercapai.

Kompleksitas Waktu

- Setiap simpul di dalam *state space tree* terkait dengan panggilan rekursif
- Jika jumlah simpul di dalam pohon $2n$ atau $n!$, maka pada kasus terburuk untuk algoritma *backtracking* ini memiliki kompleksitas waktu $O(p(n)2n)$ atau $O(q(n)n!)$, dengan $p(n)$ dan $q(n)$ sebagai polinomial dengan n-derajat menyatakan waktu komputasi untuk setiap simpul

Kompleksitas Waktu

- Setiap simpul di dalam *state space tree* terkait dengan panggilan rekursif
- Jika jumlah simpul di dalam pohon $2n$ atau $n!$, maka pada kasus terburuk untuk algoritma *backtracking* ini memiliki kompleksitas waktu $O(p(n)2n)$ atau $O(q(n)n!)$, dengan $p(n)$ dan $q(n)$ sebagai polinomial dengan n-derajat menyatakan waktu komputasi untuk setiap simpul

Setiap simpul di dalam *state space tree* terkait dengan panggilan rekursif. Jika jumlah simpul di dalam pohon $2n$ atau $n!$, maka pada kasus terburuk untuk algoritma *backtracking* ini memiliki kompleksitas waktu $O(p(n)2n)$ atau $O(q(n)n!)$, dengan $p(n)$ dan $q(n)$ sebagai polinomial dengan n-derajat menyatakan waktu komputasi untuk setiap simpul.

Ruang Solusi

- Ruang solusi untuk sebuah permainan teka-teki Calcudoku dengan *grid* yang berukuran $n \times n$ adalah $X = (x_1, x_2, ..., x_m), x_i \in \{1, 2, ..., n\}$, dengan $m = n^2$
- Fungsi pembangkit membangkitkan sebuah integer secara berurutan dari 1 sampai n sebagai x_k
- Fungsi pembatas menggabungkan tiga fungsi pemeriksa pembatas (*constraint checking*), yaitu:
  - Fungsi pemeriksa kolom (*column checking*)
  - Fungsi pemeriksa baris (*row checking*)
  - Fungsi pemeriksa *grid* (*grid checking*)

Ruang Solusi

- Ruang solusi untuk sebuah permainan teka-teki Calcudoku dengan *grid* yang berukuran $n \times n$ adalah $X = (x_1, x_2, ..., x_m), x_i \in \{1, 2, ..., n\}$, dengan $m = n^2$
- Fungsi pembangkit membangkitkan sebuah integer secara berurutan dari 1 sampai n sebagai x_k
- Fungsi pembatas menggabungkan tiga fungsi pemeriksa pembatas (*constraint checking*), yaitu:
  - Fungsi pemeriksa kolom (*column checking*)
  - Fungsi pemeriksa baris (*row checking*)
  - Fungsi pemeriksa *grid* (*grid checking*)

Ruang solusi untuk sebuah permainan teka-teki Calcudoku dengan *grid* yang berukuran $n \times n$ adalah $X = (x_1, x_2, ..., x_m), x_i \in \{1, 2, ..., n\}$, dengan $m = n^2$. Fungsi pembangkit membangkitkan sebuah integer secara berurutan dari 1 sampai n sebagai x_k. Fungsi pembatas menggabungkan tiga fungsi pemeriksa pembatas (*constraint checking*), yaitu fungsi pemeriksa kolom (*column checking*), fungsi pemeriksa baris (*row checking*), dan fungsi pemeriksa *grid* (*grid checking*).

Fungsi Pembatas

- Fungsi pemeriksa kolom menghasilkan nilai *true* jika x_k belum ada di dalam kolom dan menghasilkan nilai *false* jika sebaliknya
- Fungsi pemeriksa baris menghasilkan nilai *true* jika x_k belum ada di dalam baris dan menghasilkan nilai *false* jika sebaliknya
- Fungsi pemeriksa *grid* memeriksa operator pada *grid* dan memeriksa berdasarkan operator yang telah ditentukan

Fungsi Pembatas
- Fungsi pemeriksa kolom menghasilkan nilai *true* jika x_k belum ada di dalam kolom dan menghasilkan nilai *false* jika sebaliknya
- Fungsi pemeriksa baris menghasilkan nilai *true* jika x_k belum ada di dalam baris dan menghasilkan nilai *false* jika sebaliknya
- Fungsi pemeriksa *grid* memeriksa operator pada *grid* dan memeriksa berdasarkan operator yang telah ditentukan

Fungsi pemeriksa kolom menghasilkan nilai *true* jika x_k belum ada di dalam kolom dan menghasilkan nilai *false* jika x_k sudah ada di dalam kolom.
Fungsi pemeriksa baris menghasilkan nilai *true* jika x_k belum ada di dalam baris dan menghasilkan nilai *false* jika x_k sudah ada di dalam baris.

Fungsi pemeriksa *grid* memeriksa operator pada *grid* dan memeriksa berdasarkan operator yang telah ditentukan.

Operator-Operator untuk Fungsi Pemeriksa *Grid*

- Ada 5 operator yang digunakan dalam fungsi ini, yaitu:
  - Operator penjumlahan (+), fungsi menghasilkan nilai *true* jika hasil penjumlahan semua nilai yang ada pada *grid* ditambah dengan x_k kurang dari atau sama dengan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya
  - Operator pengurangan (-), fungsi menghasilkan nilai *true* jika kedua sel dalam *grid* kosong, atau jika ada satu sel yang kosong dan hasil dari x_k dikurangi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dikurangi dengan x_k menghasilkan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya

Operator-Operator untuk Fungsi Pemeriksa *Grid*
Ada 5 operator yang digunakan dalam fungsi ini, yaitu:
Operator penjumlahan (+), fungsi menghasilkan nilai *true* jika hasil penjumlahan semua nilai yang ada pada *grid* ditambah dengan x_k kurang dari atau sama dengan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya
Operator pengurangan (-), fungsi menghasilkan nilai *true* jika kedua sel dalam *grid* kosong, atau jika ada satu sel yang kosong dan hasil dari x_k dikurangi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dikurangi dengan x_k menghasilkan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya

Ada 5 operator yang digunakan dalam fungsi ini, yaitu:

- Operator penjumlahan (+), fungsi menghasilkan nilai *true* jika hasil penjumlahan semua nilai yang ada pada *grid* ditambah dengan x_k kurang dari atau sama dengan nilai tujuan, dan menghasilkan nilai *false* jika jumlah semua nilai yang ada pada *grid* ditambah x_k lebih dari nilai tujuan.
- Operator pengurangan (-), fungsi menghasilkan nilai *true* jika kedua sel dalam *grid* kosong, atau jika ada satu sel yang kosong dan hasil dari x_k dikurangi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dikurangi dengan x_k menghasilkan nilai tujuan, dan menghasilkan nilai *false* jika ada satu sel kosong dan hasil dari x_k dikurangi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dikurangi dengan x_k tidak menghasilkan nilai tujuan.

Operator-Operator untuk Fungsi Pemeriksa *Grid*

- Lanjutan dari slide sebelumnya:
  - Operator perkalian ($\times$), fungsi menghasilkan nilai *true* jika hasil perkalian dari semua nilai yang ada pada *grid* dikali dengan x_k kurang dari atau sama dengan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya
  - Operator pembagian ($\div$), fungsi menghasilkan nilai *true* jika kedua sel dalam *grid* kosong, atau jika ada satu sel yang kosong dan hasil dari x_k dibagi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dibagi dengan x_k menghasilkan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya
  - Operator $=$, fungsi akan menghasilkan nilai *true* jika x_k sama dengan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya

Operator-Operator untuk Fungsi Pemeriksa *Grid*
- Lanjutan dari slide sebelumnya:
  - Operator perkalian ($\times$), fungsi menghasilkan nilai *true* jika hasil perkalian dari semua nilai yang ada pada *grid* dikali dengan x_k kurang dari atau sama dengan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya
  - Operator pembagian ($\div$), fungsi menghasilkan nilai *true* jika kedua sel dalam *grid* kosong, atau jika ada satu sel yang kosong dan hasil dari x_k dibagi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dibagi dengan x_k menghasilkan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya
  - Operator $=$, fungsi akan menghasilkan nilai *true* jika x_k sama dengan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya

Ada 5 operator yang digunakan dalam fungsi ini, yaitu:

- Operator perkalian ($\times$), fungsi menghasilkan nilai *true* jika hasil perkalian dari semua nilai yang ada pada *grid* dikali dengan x_k kurang dari atau sama dengan nilai tujuan, dan menghasilkan nilai *false* jika hasil perkalian dari semua nilai yang ada pada *grid* dikali dengan x_k lebih dari nilai tujuan.
- Operator pembagian ($\div$), fungsi menghasilkan nilai *true* jika kedua sel dalam *grid* kosong, atau jika ada satu sel yang kosong dan hasil dari x_k dibagi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dibagi dengan x_k menghasilkan nilai tujuan, dan menghasilkan nilai *false* jika ada satu sel yang kosong dan hasil dari x_k dibagi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dibagi dengan x_k tidak menghasilkan nilai tujuan.
- Operator $=$, fungsi akan menghasilkan nilai *true* jika x_k sama dengan nilai tujuan, dan menghasilkan nilai *false* jika x_k tidak sama dengan nilai tujuan.

State Space Tree

- *State space tree* bersifat dinamis, berkembang secara terus-menerus sampai solusi ditemukan
- Tinggi pohon yang dikembangkan untuk menyelesaikan sebuah teka-teki dengan ukuran $n \times n$ seharusnya memiliki tinggi $n^2 + 1$ saat mencapai simpul tujuannya, dengan jalur dari simpul akar ke simpul tujuan merepresentasikan semua angka yang digunakan untuk mengisi *grid* dari sel pada sudut kiri atas ke sel pada sudut kanan bawah

State Space Tree

- *State space tree* bersifat dinamis, berkembang secara terus-menerus sampai solusi ditemukan
- Tinggi pohon yang dikembangkan untuk menyelesaikan sebuah teka-teki dengan ukuran $n \times n$ seharusnya memiliki tinggi $n^2 + 1$ saat mencapai simpul tujuannya, dengan jalur dari simpul akar ke simpul tujuan merepresentasikan semua angka yang digunakan untuk mengisi *grid* dari sel pada sudut kiri atas ke sel pada sudut kanan bawah

State space tree bersifat dinamis, berkembang secara terus-menerus sampai solusi ditemukan. Tinggi pohon yang dikembangkan untuk menyelesaikan sebuah teka-teki dengan ukuran $n \times n$ seharusnya memiliki tinggi $n^2 + 1$ saat mencapai simpul tujuannya, dengan jalur dari simpul akar ke simpul tujuan merepresentasikan semua angka yang digunakan untuk mengisi *grid* dari sel pada sudut kiri atas ke sel pada sudut kanan bawah.

Cara Kerja Algoritma *Backtracking* Secara Singkat

- Singkatnya, langkah-langkah dasar dari implementasi algoritma *backtracking* dapat dijelaskan sebagai berikut:

1. Carilah sel pertama atau sel yang kosong di dalam *grid*
2. Isilah sel dengan sebuah angka dimulai dari 1 sampai *n* sampai sebuah angka yang berlaku (*valid*) ditemukan atau sampai angka sudah melebihi *n*
3. Jika angka untuk sel berlaku, ulangi langkah 1 dan 2
4. Jika angka untuk sel sudah melebihi *n* dan tidak ada angka dari 1 sampai *n* yang berlaku untuk sel tersebut, mundur ke sel sebelumnya dan cobalah kemungkinan angka berikutnya yang berlaku untul sel tersebut
5. Jika tidak ada lagi sel yang kosong, solusi sudah ditemukan

Cara Kerja Algoritma *Backtracking* Secara Singkat
Singkatnya, langkah-langkah dasar dari implementasi algoritma *backtracking* dapat dijelaskan sebagai berikut:
1. Carilah sel pertama atau sel yang kosong di dalam *grid*
2. Isilah sel dengan sebuah angka dimulai dari 1 sampai *n* sampai sebuah angka yang berlaku (*valid*) ditemukan atau sampai angka sudah melebihi *n*
3. Jika angka untuk sel berlaku, ulangi langkah 1 dan 2
4. Jika angka untuk sel sudah melebihi *n* dan tidak ada angka dari 1 sampai *n* yang berlaku untuk sel tersebut, mundur ke sel sebelumnya dan cobalah kemungkinan angka berikutnya yang berlaku untul sel tersebut
5. Jika tidak ada lagi sel yang kosong, solusi sudah ditemukan

Singkatnya, langkah-langkah dasar dari implementasi algoritma *backtracking* dapat dijelaskan sebagai berikut [**?**]:

1. Carilah sel pertama atau sel yang kosong di dalam *grid*.
2. Isilah sel dengan sebuah angka dimulai dari 1 sampai *n* sampai sebuah angka yang berlaku (*valid*) ditemukan atau sampai angka sudah melebihi *n*.
3. Jika angka untuk sel berlaku, ulangi langkah 1 dan 2.
4. Jika angka untuk sel sudah melebihi *n* dan tidak ada angka dari 1 sampai *n* yang berlaku untuk sel tersebut, mundur ke sel sebelumnya dan cobalah kemungkinan angka berikutnya yang berlaku untul sel tersebut.
5. Jika tidak ada lagi sel yang kosong, solusi sudah ditemukan.