Pemanfaatan Pohon dalam Realisasi Algoritma *Backtracking* untuk Memecahkan *N-Queens Problem*

Halida Astatin (13507049)
Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika,
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung, email: if17049@students.if.itb.ac.id

***Abstract* –** Backtracking *adalah salah satu strategi yang dapat digunakan untuk memecahkan suatu masalah yang memiliki banyak kemungkinan solusi yang perlu diuji secara bertahap. Tahap-tahap pencarian solusi yang ditelusuri oleh algoritma ini kemudian dapat dimisalkan sebagai suatu pohon solusi. Saat ini* backtracking *banyak digunakan dalam program-program komputer yang mengaplikasikan kecerdasan buatan. Salah satu contoh permasalahan yang dapat diselesaikan dengan memanfaatkan algoritma* backtracking *adalah* N-Queens Problem, *yaitu masalah penempatan N buah bidak Ratu dalam suatu papan catur berukuran N x N sedemikian rupa agar bidak-bidak ratu tersebut tidak dapat saling memakan.*

***Kata Kunci**:* backtracking, *pohon solusi*, N-Queens Problem, *catur*

1. PENDAHULUAN

Teknologi Informasi (*Information Technology—IT*) adalah salah satu cabang ilmu yang ruang pengembangannya masih sangat luas. Pemrograman, sebagai salah satu bagian dari teknologi informasi, tentu terkena imbas perkembangan ilmu ini. Pemrograman saat ini telah berkembang pesat. Semakin banyak fasilitas yang ditawarkan oleh aplikasi-aplikasi komputer yang ada saat ini—semakin banyak pekerjaan manusia yang dipermudah oleh adanya aplikasi komputer yang tersedia.

Salah satu hasil kemajuan teknologi informasi ini adalah lahirnya *Artificial Intelligence (A.I.*) atau kecerdasan buatan. Kecerdasan buatan ini dapat membantu suatu program untuk bekerja dengan 'pemikiran sendiri' sehingga meminimisasi campur tangan pengguna. Kecerdasan buatan saat ini umum digunakan dalam permainan-permainan komputer yang memungkinkan mode *player-versus-computer*, seperti catur atau *scrabble*. Dalam mewujudkan kecerdasan buatan ini, salah satu algoritma yang umum digunakan adalah algoritma *backtracking*.

Algoritma *backtracking* bekerja sedemikian rupa sehingga tahap-tahap pencarian solusi yang dilaluinya dapat dilambangkan sebagai suatu pohon solusi. Karenanya, dalam pembuatan program dengan algoritma *backtracking*, proses-prosesnya bersifat rekursif seperti halnya pemrosesan pohon. Dalam makalah ini akan dibahas studi kasus penyelesaian masalah *N-Queens* dengan algoritma *backtracking* dan representasi pohonnya.

2. DEFINISI

Sebelum membahas lebih jauh mengenai aplikasi pohon dalam penanganan *N-Queens Problem*, perlu dibahas lebih lanjut definisi dari pohon itu sendiri dan algoritma *backtracking* yang akan digunakan untuk menyelesaikan permasalahan.

2.1. Pohon

Pohon adalah salah satu representasi struktur data yang telah lama digunakan. Konsep pohon merupakan pengembangan dari konsep graf dan telah digunakan sejak tahun 1857 oleh matematikawan Arthur Cayley untuk menghitung jumlah senyawa kimia. Definisi pohon adalah graf khusus yang tak berarah dan tidak mengandung sirkuit.

Graf sendiri dapat didefinisikan sebagai himpunan (*V*, *E*) dimana *V* adalah himpunan tidak kosong dari simpul-simpul (*vertices* atau *nodes*) $\{v_1, v_2, \ldots, v_n\}$ dan *E* adalah himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul $\{e_1, e_2, \ldots, e_n\}$. Notasi graf disingkat $G = (V, E)$. Dalam konsep pohon, suatu pohon dapat mempunyai hanya sebuah simpul tanpa sisi. Artinya, *V* tidak boleh berupa himpunan kosong, namun *E* boleh kosong.

Definisi pohon yang dijelaskan sebelumnya seringkali disebut pula **pohon bebas**. Pada kebanyakan aplikasi pohon, salah satu simpul diperlakukan sebagai akar (*root*). Pohon yang sebuah simpulnya diperlakukan sebagai akar dinamakan **pohon berakar** (*rooted tree*). Pada pohon berakar, simpul-simpul lain akan dapat dicapai dari akar dengan suatu lintasan. Sembarang pohon tak berakar dapat diubah menjadi pohon

berakar dengan cara memilih sebuah simpul untuk dijadikan akar. Pemilihan simpul ini akan berpengaruh ke bentuk pohon berakar yang terbentuk.

Berikut ini beberapa istilah yang berhubungan dengan pohon:

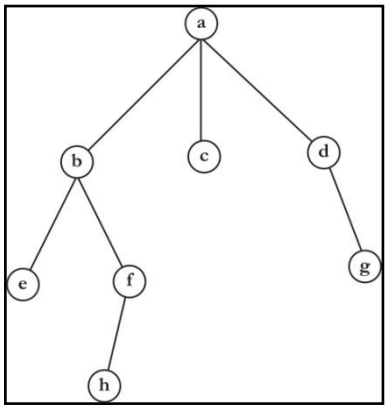
a) Anak (*child*/*children*) dan Orangtua (*parent*)
Misalkan terdapat suatu simpul x dan y. Simpul y dikatakan sebagai anak simpul x jika ada sisi dari simpul x ke y. Sebaliknya, x kemudian disebut orangtua (*parent*) y.

b) Lintasan (*path*)
Lintasan dari simpul v_1 ke simpul v_2 adalah runtutan simpul-simpul v_1, v_2, …, v_k sedemikian sehingga v_i adalah orangtua dari v_{i+1} untuk $1 \leq i < k$.

c) Saudara kandung (*sibling*)
Simpul yang berorangtua sama adalah saudara kandung satu sama lain.

d) Derajat (*degree*)
Derajat sebuah simpul pada pohon berakar adalah jumlah upapohon (atau jumlah anak) pada simpul tersebut. Derajat suatu pohon adalah derajat maksimum dari semua simpul.

e) Simpul-simpul
Simpul yang berderajat nol atau tidak mempunyai anak disebut daun (*leaf*). Simpul yang mempunyai anak disebut simpul dalam (*internal nodes*).



Gambar 1: Contoh pohon untuk penjelasan istilah

Contoh pohon tersebut memiliki 8 simpul, berderajat 3, dan berkedalaman 3. Simpul a adalah akar pohon tersebut, simpul e, g, dan h adalah daun. Simpul b, c, dan d adalah saudara kandung. Simpul b adalah orangtua simpul e dan f, sebaliknya simpul e dan f adalah anak simpul b.

Pohon berakar yang memiliki derajat *n* disebut **pohon *n-ary***. Pohon *n-ary* banyak digunakan dalam berbagai bidang ilmu dan dalam kehidupan sehari-hari. Pohon contoh yang telah diberikan adalah suatu pohon 3-*ary*. Pohon *n-ary* disebut teratur atau penuh jika seluruh cabangnya mempunyai tepat *n* buah anak. Penggunaan pohon *n-ary* dalam informatika adalah sebagai suatu model yang merepresentasikan suatu struktur. Contoh aplikasi lainnya adalah untuk pohon keluarga atau suatu susunan organisasi. Pohon *n-ary* yang umum digunakan adalah pohon biner, dimana n = 2. Dalam pemrograman, struktur pohon *n-ary* direpresentasikan dengan multilist, kecuali untuk pohon biner yang memiliki struktur khusus.

2.2. Backtracking

Saat ini, telah beredar banyak program komputer yang memanfaatkan teknologi AI atau kecerdasan buatan. Umumnya, kecerdasan buatan diprogram untuk dapat menyelesaikan suatu permasalahan yang memiliki serangkaian upamasalah lain yang masing-masing memiliki banyak kemungkinan solusi. Dalam hal ini, solusi yang dipilih untuk satu upamasalah akan berpengaruh pada kemungkinan solusi upamasalah berikutnya.

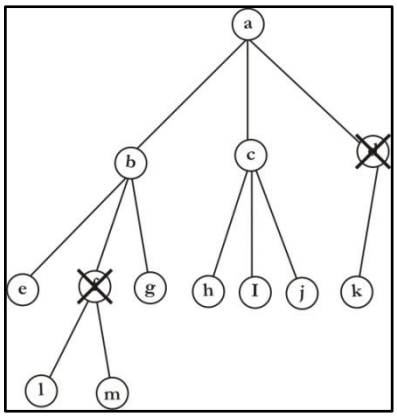
Algoritma yang dapat digunakan untuk menemukan seluruh atau sebagian solusi untuk permasalahan seperti ini diantaranya *brute force* dan *backtracking*. Kedua algoritma tersebut mencari solusi dari suatu masalah dengan cara membuat kandidat-kandidat solusi secara bertahap. Namun demikian, *backtracking* dapat dikatakan lebih sebagai penyempurnaan dari algoritma *brute force*. Pada pencarian solusi dengan *brute force*, program akan membuat seluruh kemungkinan solusi, baru kemudian menguji satu persatu apakah solusi yang telah dibuat dapat memenuhi spesifikasi solusi akhir yang dicari. Berbeda dengan *brute force*, algoritma *backtracking* akan berhenti memproses suatu kandidat solusi segera setelah sampai pada suatu tahap dimana kandidat solusi tersebut terbukti tidak mengarah pada solusi akhir. Contohnya jika terdapat sebuah kata yang terdiri atas 8 huruf kemudian program diminta untuk membentuk suatu kata baru yang terdiri dari 2 huruf vokal dan 2 konsonan. Ketika pencarian sampai pada huruf ke-tiga dan ketiga huruf tersebut merupakan konsonan, maka kandidat solusi ini akan langsung gugur. Program tidak akan melanjutkan pencarian huruf ke-empat.

Kelemahannya, algoritma *backtracking* ini hanya bisa diaplikasikan terbatas pada tipe permasalahan yang memiliki solusi yang dapat dicari secara sistematis dan bertahap seperti yang telah dijelaskan sebelumnya. Terdapat masalah-masalah

yang tidak bisa diselesaikan dengan menggunakan *backtracking*, misalnya menemukan suatu nilai yang diminta pada tabel yang tidak terurut. Namun ketika algoritma ini dapat diaplikasikan, *backtracking* dapat bekerja jauh lebih cepat dari *brute force* karena jumlah kandidat solusi yang dapat 'dibuang' dengan *backtracking* cukup besar.

Algoritma *backtracking* berbasis pada DFS (*depth-first search*) atau pencarian mendalam dengan tujuan mencari solusi permasalahan secara lebih mangkus. Mekanisme penyelesaian dengan menggunakan *backtracking* berprinsip pada metode rekursif. Untuk menyelesaikan keseluruhan masalah, dibutuhkan sebuah solusi untuk upamasalah pertama, kemudian upamasalah-upamasalah lainnya akan dicoba untuk diselesaikan secara rekursif berdasarkan solusi pertama tersebut. Jika kemungkinan solusi yang sedang dicoba gagal, atau jika tujuan program adalah untuk menemukan seluruh solusi yang mungkin, maka dilakukan *backtrack* untuk menguji kemungkinan solusi selanjutnya. Proses *backtrack* akan selesai ketika tidak ada lagi solusi yang mungkin untuk menyelesaikan upamasalah paling awal.

Salah satu fungsi yang dimiliki oleh algoritma *backtracking* dan menjadi ciri khasnya adalah fungsi pemangkasan (*pruning*). Andaikan tahap-tahap pencarian solusi suatu masalah direpresentasikan dalam bentuk pohon solusi, proses pemangkasan akan dilakukan terhadap simpul-simpul yang tidak mengarah kepada solusi. Jika suatu simpul telah dipangkas, simpul-simpul yang menjadi anak dan turunan dari simpul tersebut otomatis tidak akan diproses, karena memangkas sebuah simpul sama halnya membuang seluruh lintasan yang berada di bawah simpul tersebut.



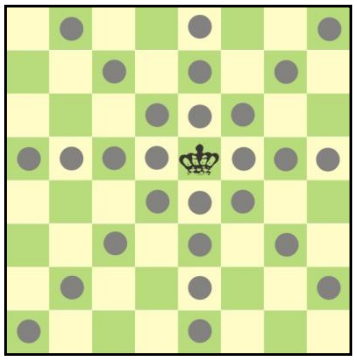
Gambar 2: Ilustrasi proses pemangkasan (*pruning*)

Berdasarkan gambar tersebut, simpul yang dipangkas adalah simpul d dan f. Akibat pemangkasan tersebut, simpul k yang merupakan anak simpul d, serta simpul l dan m yang merupakan anak simpul f tidak akan diproses.

Algoritma *backtracking* banyak digunakan dalam pembuatan permainan komputer, misalnya *tic-tac-toe*, labirin, dan catur. Selain itu algoritma ini merupakan metode paling efisien untuk *parsing* dan banyak masalah optimasi kombinatorial lainnya. *Backtracking* juga digunakan oleh bahasa pemrograman logika seperti Prolog, Icon, dan Planner.

3. N-QUEENS PROBLEM

N-Queens Problem adalah salah satu permasalahan yang paling umum digunakan dalam berbagai buku pegangan untuk menjelaskan algoritma *backtracking*. Inti permasalahan yang diajukan adalah bagaimana menempatkan *N* buah bidak ratu dalam suatu papan catur berukuran *N* x *N* sedemikian rupa sehingga tidak satupun dari bidak ratu tersebut dapat memakan bidak ratu yang lain dalam satu gerakan. Sesuai dengan gerakan bidak ratu standar, suatu bidak ratu hanya dapat bergerak lurus dalam satu kolom, baris, atau diagonal. Untuk itu sebuah solusi harus dapat mengatur penempatan bidak ratu sehingga tidak ada dua bidak yang terletak dalam suatu kolom, baris, atau diagonal yang sama.



Gambar 3: Ruang gerak bidak ratu dalam permainan catur

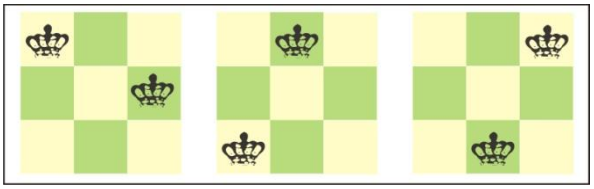
N-Queens problem pertama kali diusulkan sebagai *8 Queens Problem*, sesuai dengan ukuran papan catur sebenarnya yaitu 8 x 8, pada tahun 1848 oleh seorang pecatur Max Bezzel. Banyak matematikawan, termasuk Gauss dan Georg Cantor, telah berusaha mencari solusi permasalahan ini. Solusi pertama dipublikasikan oleh Franz Nauck pada tahun 1850—termasuk solusi permasalahan umumnya, yaitu ketika jumlah bidak ratu dimisalkan sebagai *N*. Adalah Edsger Djikstra yang menggunakan masalah ini pada tahun 1972 untuk memperkenalkan pemrograman terstruktur, dengan menyelesaikan masalah tersebut menggunakan algoritma DFS *backtracking*.

3.1. Pencarian solusi untuk kasus $N < 4$

Menurut riset Neil Sloane nomor A002562, dalam *N-Queens Problem*, untuk N = 1, 2, dan 3, jumlah solusi yang valid adalah 1, 0, 0. Artinya, untuk N < 4, solusi hanya tersedia untuk N = 1.

Untuk N = 1, cukup jelas bahwa solusi dapat ditemukan, karena kemungkinan cara penempatan bidak ratu hanya 1 dan tidak mungkin ada bidak yang saling memakan karena hanya ada 1 buah bidak ratu. Sebaliknya, untuk N = 2, cukup jelas bahwa tidak ada solusi yang mungkin. Pada suatu papan catur 2 x 2, dimanapun bidak ratu pertama diletakkan, bidak tersebut akan dapat mencapai petak lain dalam papan dalam satu gerakan, sehingga tidak ada tempat yang memungkinkan untuk meletakkan bidak ratu yang kedua.

Berdasarkan konsep kombinatorial, untuk penempatan 3 bidak ratu dalam suatu papan catur berukuran 3 x 3 terdapat 504 $\left(\frac{(3*3)!}{((3*3)-3)!}\right)$ kemungkinan solusi. Jumlah kemungkinan ini dapat dikurangi apabila kita tambahkan syarat bahwa ketiga bidak tersebut tidak dapat berada dalam suatu kolom yang sama. Dengan mengaplikasikan syarat tersebut, maka jumlah kemungkinan solusi menjadi 3^3 = 27 kemungkinan. Namun demikian, dari 27 kemungkinan yang ada, tidak satupun yang mengarah pada solusi akhir. Buktinya adalah sebagai berikut:



Gambar 4: Penempatan dua bidak pertama untuk N = 3

Dari gambar tersebut dapat dilihat bahwa setelah penempatan dua buah bidak ratu, tidak ada petak dalam papan catur yang dapat ditempati oleh bidak ratu ke-tiga.

3.2. Pencarian solusi dengan Backtracking

Masih mengacu pada Sloane's A002562, untuk $N \geq$ 4, selalu terdapat solusi untuk *N-Queens Problem*. Berdasarkan konsep kombinatorial, untuk sembarang nilai *N*, *N-Queens Problem* memiliki kemungkinan penyelesaian sebanyak $\left(\frac{N!}{N*(N-1)!}\right)$ kemungkinan. Apabila kemudian diterapkan syarat bahwa dua buah bidak ratu tidak dapat terletak dalam satu kolom yang sama, maka nilai tersebut dapat dikurangi menjadi N^N kemungkinan solusi.

Apabila permasalahan ini dicoba untuk diselesaikan menggunakan algoritma *brute force*, maka program akan membentuk seluruh kemungkinan solusi, baru kemudian menguji apakah solusi tersebut memenuhi spesifikasi yang diinginkan. Untuk nilai N relatif kecil, contohnya N = 8, pencarian solusi dengan cara ini masih dapat diaplikasikan, namun untuk nilai N ekstrem besar, contohnya N = 1.000.000, pencarian dengan teknik *brute force* sangat boros memori dan tidak mangkus. Untuk itu, penyelesaian *N-Queens Problem* lebih baik dicari dengan menggunakan algoritma *backtracking*. Pencarian solusi dengan teknik ini akan mengurangi penggunaan memori, karena teknik *backtracking* akan melakukan pemangkasan terhadap sebagian besar kemungkinan solusi yang ada.

Proses pencarian solusi secara bertahap dapat dimisalkan sebagai suatu pohon solusi. Tiap-tiap langkah akan menjadi suatu simpul dalam pohon solusi tersebut. Untuk tiap *N-Queens Problem*, dengan menerapkan syarat bahwa dua buah bidak ratu tidak dapat terletak dalam suatu kolom yang sama, pohon solusi yang terbentuk adalah pohon *N-ary* teratur. Artinya, tiap-tiap simpul yang terbentuk akan memiliki N buah anak. Akar pohon solusi tersebut adalah papan kosong atau belum ada bidak catur yang diletakkan. Pohon inilah yang kemudian akan digunakan untuk proses *backtracking*.

Algoritma pencarian solusi tersebut diproses secara rekursif seperti pemrosesan pohon pada umumnya. Mekanisme algoritma ini adalah dengan mengunjungi satu persatu simpul yang ada dalam pohon solusi dan menguji apakah simpul tersebut memenuhi spesifikasi simpul yang dapat ditempati oleh sebuah bidak ratu. Seandainya simpul tersebut bukan merupakan simpul yang valid, maka simpul tersebut akan dipangkas, dan proses pencarian berlanjut ke salah satu saudara kandung simpul tersebut. Apabila tidak ada lagi saudara kandung simpul yang dapat diuji, maka pencarian akan bergerak mundur (*backtrack*) ke orangtua simpul, lalu dilanjutkan ke saudara kandung orangtua simpul. Proses akan berakhir ketika proses *backtrack* telah kembali ke akar pohon *N-ary*.

```
procedure CariSolusi (input Pohon P1)

Deklarasi:
procedure Pangkas ( )
{Membuang simpul yang sedang
dikunjungi karena tidak mengarah ke
solusi valid, dan anak dari simpul
tersebut tidak akan diproses lagi}

procedure Maju ( )
{Selama simpul yang sedang dikunjungi
bukan daun, melanjutkan pencarian ke
```

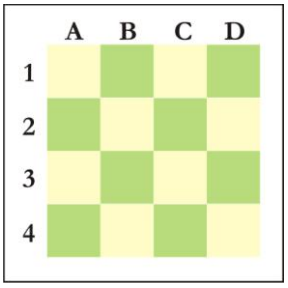
```
anak paling kiri simpul yang sedang
dikunjungi}

procedure Backtrack ( )
{Mundur sampai simpul yang masih
mempunyai anak yang belum dikunjungi,
berhenti jika mencapai akar}

Algoritma:

while ((simpul bukan daun) and
      (simpul mengarah ke solusi)) do
      Maju ( )
      CariSolusi ( )
{kondisi berhenti: simpul tidak
mengarah ke solusi atau simpul adalah
daun}

if (simpul tidak mengarah ke solusi)
      Pangkas ( )
      Backtrack ( )
      CariSolusi ( )
else {simpul adalah daun}
       if (solusi valid) then
             output (solusi)
       Backtrack ( )
       CariSolusi ( )
```

Untuk melihat lebih jauh aplikasi *backtracking* pada permasalahan ini, mari kita tinjau proses pencarian solusi untuk $N = 4$. Kemungkinan penempatan 4 buah bidak ratu pada suatu papan catur 4 x 4 secara sembarang tanpa ada syarat khusus adalah $\left(\frac{4*4!}{4*3!}\right)$ = 43.680 kemungkinan. Jika kemudian diaplikasikan syarat bahwa tidak boleh ada dua dari empat bidak tersebut yang terletak dalam kolom yang sama, maka total solusi yang mungkin adalah 4^4 = 256 kemungkinan. Pada kenyataannya, dari riset Neil Sloane yang telah diacu sebelumnya, jumlah solusi *N-Queens Problem* untuk nilai $N = 4$ hanya 2 solusi. Kedua solusi yang ditawarkan itupun sebenarnya merupakan satu susunan yang sama, hanya dicerminkan terhadap sumbu vertikal (hasil yang sama juga diperoleh dengan mencerminkan terhadap sumbu horizontal).

Dengan menggunakan algoritma tersebut, urutan pencarian solusi untuk $N = 4$ adalah sebagai berikut:

1) Akar pohon adalah suatu papan catur 4 x 4 yang masih kosong.
2) Dibentuk suatu pohon solusi, dengan simpul-simpul aras ke-1 adalah petak-petak pada kolom ke-1, simpul-simpul aras ke-2 adalah petak-petak pada kolom ke-2, dan seterusnya. Simpul disusun dengan cara petak yang paling atas dalam papan catur diletakkan di simpul paling kiri (diproses pertama).
3) Dengan menggunakan algoritma yang telah dibuat sebelumnya, bidak-bidak ratu akan dicoba untuk diletakkan pada papan tersebut.
4) Proses akan berakhir jika seluruh solusi telah ditemukan dan tidak ada lagi solusi yang mungkin.

Untuk menggambarkan lebih jelas proses kerja algoritma *backtracking*, akan dimisalkan penempatan empat bidak ratu pada papan catur berikut ini:

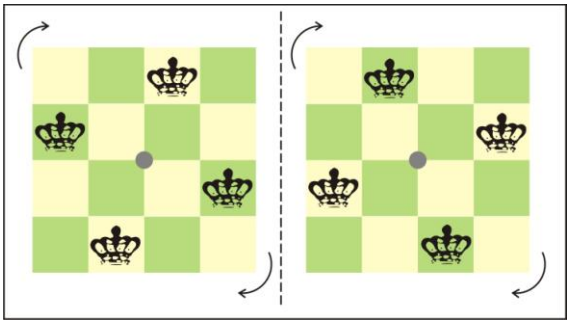


Gambar 5: Papan catur 4 x 4 untuk penjelasan langkah penyelesaian

Berdasarkan proses penyusunan pohon solusi yang telah disebutkan, maka dalam pencarian solusi, bidak pertama akan diletakkan pertama kali pada kolom A1. Pencarian kemudian akan dilanjutkan ke kolom B, dan simpul B1 dan B2 akan otomatis dipangkas karena sudah tidak mungkin menghasilkan solusi (B1 terletak pada baris yang sama dengan bidak pertama, B2 terletak pada diagonal yang sama). Pada kolom B, bidak akan diletakkan pada B3, kemudian pencarian dilanjutkan ke kolom C. Dengan posisi bidak saat ini, tidak ada petak dalam kolom C yang bisa ditempati oleh bidak ke-tiga, sehingga program akan melakukan *backtrack* kembali ke kolom B. Simpul B3 akan dipangkas dan bidak pada kolom B akan dipindahkan ke petak B4. Dengan melanjutkan pencarian solusi menggunakan teknik serupa di atas, bidak pada kolom C akan diletakkan pada C2.

Masalah akan kembali muncul ketika bidak akan ditempatkan di kolom D, karena ternyata tidak ada petak yang dapat ditempati bidak ratu ke-empat. Akibatnya, program akan melakukan *backtrack*, memangkas simpul C2, C3, dan C4 (karena dua yang terakhir juga tidak mengarah pada solusi), kemudian melakukan *backtrack* sampai kolom A (karena pada kolom B seluruh simpul telah dipangkas). Simpul A1 kemudian dipangkas dan pencarian berlanjut dengan penempatan bidak ratu

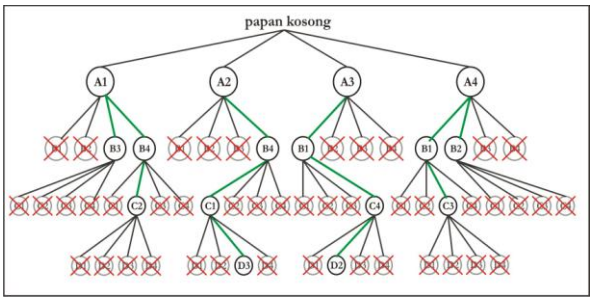
pertama di petak A2.



Gambar 6: Dua solusi 4-*Queens Problem*, keduanya memiliki simetri putar 90º dan merupakan pencerminan satu sama lain

Menggunakan cara yang serupa dengan sebelumnya, solusi akhir yang didapatkan adalah A2-B4-C1-D3 dan A3-B1-C4-D2. Kedua solusi ini sama-sama memiliki simetri putar 90º. Keduanya sebenarnya adalah satu solusi yang sama, dimana solusi kedua dapat diperoleh dengan mencerminkan solusi pertama terhadap sumbu vertikal ataupun horizontal.

Pohon solusi yang diperlukan untuk mencari solusi akhir dari permasalahan ini adalah suatu pohon 4-*ary*. Artinya, dalam pohon tersebut tiap-tiap simpulnya memiliki 4 simpul anak, sehingga pada akhirnya terdapat $4^4 = 256$ simpul ditambah satu buah simpul akar. Dengan proses backtracking, jumlah simpul yang tersisa akan berkurang akibat proses pemangkasan.



Gambar 7: Pohon solusi 4-Queens Problem setelah melalui proses backtracking

Dari pohon hasil *backtracking* tersebut, dapat dilihat bahwa jumlah kemungkinan solusi yang dipangkas oleh algoritma ini cukup banyak. Dari 257 simpul yang seharusnya terbentuk, akhirnya hanya tersisa 61 simpul, dan hanya 17 diantaranya yang merupakan simpul hidup. Dengan demikian dapat dilihat bahwa persoalan dapat diselesaikan dengan lebih mangkus dengan menggunakan aplikasi pohon pada algoritma *backtracking*. Algoritma ini dapat diterapkan pula untuk nilai *N* lainnya.

4. KESIMPULAN

Pohon adalah suatu pengembangan konsep graf yang memiliki banyak aplikasi dalam kehidupan sehari-hari. Contoh penggunaan pohon misalnya dalam pohon keluarga atau susunan organisasi. Dalam pemrograman, yang umum digunakan adalah pohon *n-ary* teratur, yaitu pohon yang setiap simpulnya memiliki tepat *n* buah simpul anak.

Konsep pohon dapat diaplikasikan pada pemrograman dengan teknik *backtracking*. Menggunakan teknik ini, tahap-tahap pencarian solusi akan dibentuk menjadi suatu pohon solusi, dan dicari solusi akhirnya dengan cara mengunjungi simpul satu persatu. Salah satu permasalahan yang dapat diselesaikan dengan teknik *backtracking* adalah *N-Queens Problem*.

DAFTAR REFERENSI

[1] Munir, Rinaldi, *Diktat Kuliah IF2091Struktur Diskrit*. Prodi Teknik Informatika Institut Teknologi Bandung, 2003.

[2] Liem, Inggriani, *Diktat Struktur Data*. Prodi Teknik Informatika Institut Teknologi Bandung, 2008.

[3] Backtracking <http://www.cs.rpi.edu/~hollingd/psics/notes/backtracking.pdf> diakses 26 Desember 2008 pukul 17:08.

[4] Chapter 19: Backtracking Algorithm < http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch19.html#QQ1-51-128> diakses 2 Januari 2009 pukul 23:22.

[5] Free On-Line Dictionary of Computing, "Backtracking", "DFS" <http://www.foldoc.org> diakses 2 Januari 2009 pukul 23:02.

[6] Munir, Rinaldi, Slide Kuliah Strategi Algoritmik, "Algoritma Runut-balik (*backtracking*) bagian 1" <http://informatika.org/~rinaldi/Stmik/2006-2007> diakses 1 Januari 2009 pukul 18:36.

[7] Recursive Backtracking < www.stanford.edu/class/cs106x/handouts/HO19.pdf> diakses 26 Desember 2008 pukul 17:08.

[8] Wikipedia, "Backtracking" <http://en.wikipedia.org/wiki/backtracking> diakses 25 Desember 2008 pukul 23:35.

[9] Wikipedia, "Eight Queens Puzzle" <http://en.wikipedia.org/wiki/eight_queens_problem> diakses 25 Desember 2008 pukul 23:46.