Penyelesaian Persoalan *N-Queens* Secara *Depth First Search* Memanfaatan Pola Bit dan Kesimetrian

Stevens Jethefer[1], Chandra Gondowasito[2], Anthony Hartanto[3]

Sekolah Teknik Elektro dan Informatika
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if14080@students.if.itb.ac.id[1], if14100@students.if.itb.ac.id[2],
if14102@students.if.itb.ac.id[3]

Abstrak

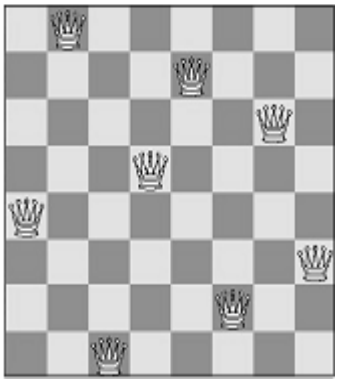
Persoalan *N-Queens* merupakan salah satu persoalan pencarian klasik dalam bidang Intelegensia Buatan. Persoalan ini memiliki banyak penerapannya dalam kehidupan, misalnya dalam masalah perutean dan pengetesan VLSI 2 dimensi dan *parallel optical computing*. Persoalan *N-Queens* adalah menempatkan N buah bidak ratu pada NxN papan catur sehingga tidak ada bidak ratu yang saling menyerang.

Dalam makalah ini kami memaparkan penyelesaian persoalan *N-Queens* yaitu mencari semua solusi yang ada dengan memanfaatkan stuktur data bit dan mengeksploitasi kesimetrian dari solusi. Metode penyelesaian yang kami gunakan adalah secara *Depth First Search*. Pengeksploitasian simetri dan penggunakan struktur bit mampu menyelesaikan persoalan *N-Queens* dengan lebih sangkil dan mangkus daripada menggunakan metode *Depth First Search* yang biasa.

***Kata kunci**: N-Queens, bit, simetri.*

1. Pendahuluan

Persoalan *N-Queens* adalah persoalan menempatkan N buah bidak ratu dalam suatu papan catur berukuran N x N sehingga tidak ada ratu yang saling menyerang, yaitu tidak ada dua bidak ratu yang terletak pada satu baris, satu kolom, dan satu diagonal yang sama.



Gambar 1 Salah satu solusi persoalan N-Queens untuk N=8.

Dalam makalah ini, kami menjelaskan cara penyelesaian persoalan *N-Queens* secara *Depth First Srearch* dengan memanfaatkan struktur data pola bit dan konsep simetri untuk mempercepat pencarian solusi.

Semua kode program yang dijelaskan dalam makalah ini adalah dalam bahasa pemrograman C dan bisa di-*download* di situs:
http://students.if.itb.ac.id/~if14100/queens/

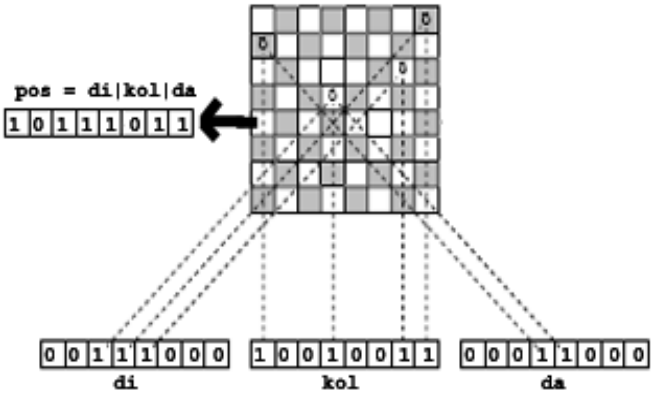
2. Solusi Dasar

Dalam menyelesaikan persolan ini, tiap baris dari papan catur direpresentasikan sebagai pola bit. Bit terkanan dari pola bit merepresentasikan kolom ke-1. Bit berisi 0 bila kolom yang direpresentasikannya belum ditempati oleh bidak ratu dan berisi 1 bila telah ditempati.

Pada saat awal yaitu dimulai dari baris pertama, pola bit terisi dengan 0 sebanyak N. Bit sisa yang tidak dipakai karena berlebih diisi dengan 1 agar tidak bisa ditempati bidak ratu nantinya. Diperlukan 3 pola bit (3 `unsigned`) untuk merepresentasikan papan catur berukuran NxN. Gambar 2 mengilustrasikan pemakaian 3 pola bit tersebut.

Variabel `kol` adalah pola bit yang berisi 1 bila pada kolom bit yang bersesuaian telah ditempati bidak ratu, variabel `di` berisi bit 1 untuk bit-bit yang terserang oleh bidak-bidak ratu dari arah diagonal kirinya, sedangkan variabel `da` berisi bit 1 bila kolom yang direpresentasikan bit tersebut terserang bidak ratu dari arah diagonal kanannya. Variabel `pos` adalah pola bit yang berisi bit 1 di kolom bit yang telah ditempati bidak ratu atau yang terserang oleh bidak ratu lsebelumnya dari arah diagonal. Jadi

bidak ratu hanya bisa ditempatkan pada kolom bit yang berisi bit 0. Pola bit `pos` ini menunjukkan kondisi awal baris yang akan diproses.



Gambar 2. Pola bit `pos` yang merepesen-tasikan kondisi baris 5 untuk 8-Queens.

Perulangan kemudian digunakan untuk memperoleh semua posisi yang masih bisa ditempati bidak ratu pada baris yang direpresentasikan oleh `pos`. Tiap posisi tersebut dapat diperoleh dengan cara berikut:

```
current=~poss&(poss+1);
```

Di bawah ini adalah salah satu posisi yang bisa ditempati oleh bidak ratu yaitu kolom 3 berdasarkan kondisi pada gambar 2:

```
~pos     01000100
 pos+1   10111100
 ---------------- &
 current 00000100
```

Selanjutnya `pos` diperbarui agar bisa dipakai untuk memperoleh posisi bidak ratu yang lainnya:

```
poss|=current;
```

Fungsi rekursif dipanggil untuk memproses baris selanjutnya dengan mem-*passing*-kan 3 pola bit yang akan merepresentasikan kondisi pos untuk baris berikutnya:

`(di|current)<<1, kol|current,` dan `(da|current)>>1.`

Bila telah mencapai baris ke-N dan `pos` masih memiliki bit 0 berarti jumlah solusi yang diperoleh bertambah. Berikut adalah potongan dari program queens0.c yaitu fungsi rekursif untuk mencari jumlah semua solusi dengan `N` adalah panjang papan catur dan `SOLUSI` adalah jumlah solusi persoalan yang akan dicari:

```
void Queens(unsigned ld,
            unsigned cols,
            unsigned rd,
            int baris)
{
  unsigned current,poss=ld|cols|rd;
  if(poss<UINT_MAX)
    if(baris<N)
      do{
        poss|=current=~poss&(poss+1);
        Queens( (ld|current)<<1,
                 cols|current,
                (rd|current)>>1,
                 baris+1);
      }while(poss<UINT_MAX);
  else ++SOLUSI;
}
```

Pemanggilan fungsi tersebut pertama kali adalah dengan:

```
Queens(all,0,0,1);
```

dengan `all` adalah:

```
all=UINT_MAX^((1<<N)-1);
```

3. Mempercepat Eksekusi Program dengan Mengurangi Pemanggilan Fungsi Rekursif

Fungsi rekursif diatas bisa lebih dioptimalkan melihat kenyataan bahwa memanggil fungsi di dalam fungsi memerlukan "biaya" yang cukup berarti dalam permasalahan ini. Untuk mengurangi pemanggilan fungsi rekursif maka sebelum memutuskan memanggil `Queens()` di dalam `Queens()`, `lebih dahulu` kita cek apakah `pos<UINT_MAX`. Berikut versi `Quens()` yang memakai cara ini dalam program queens1.c:

```
void Queens(unsigned ld,
            unsigned cols,
            unsigned rd,
            int baris)
{
   if(baris<N){
     unsigned current,poss=ld|cols|rd;
     do{
        poss|=current=~poss&(poss+1);
        if( (  ((ld|current)<<1) |
               (cols|current)    |
               ((rd|current)>>1)
             )< UINT_MAX)
        {
           Queens( (ld|current)<<1,
                    cols|current,
                    (rd|current)>>1,
                    baris+1);
        }
     }while(poss<UINT_MAX);
   }else ++SOLUSI;
}
```

4. Perbaikan Cara Pemecahan dengan Memanfaatkan Sifat Pencerminan dari Solusi

Meskipun program di atas sudah cukup mangkus akibat pengunaan struktur data bit, cara

penyelesaiannya masih belum memanfaatkan sifat pencerminan dari solusi.

Dari satu solusi yang telah diperoleh kita bisa mendapatkan maksimal 8 buah solusi yang unik:

a. Empat solusi pertama:
   i. Solusi itu sendiri,
   ii. Solusi dirotasi 90 derajat,
   iii. Solusi dirotasi 180 derajat,
   iv. Solusi dirotasi 270 derajat.
b. Empat solusi lainnya bisa diperoleh dengan mencerminkan tiap solusi diatas secara horisontal terhadap sumbu papan catur.

Pencerminan terhadap sumbu papan catur caranya adalah dengan hanya menggunakan (N+1)/2 bit terkanan dari pola bit saat berada pada baris pertama. Bila N genap maka baris-baris selanjutnya tetap memanfaatkan N bit.

Gambar 3. Lingkaran hitam adalah posisi yang mungkin untuk baris 1 untuk N genap

Untuk N ganjil aturannya adalah : bila posisi bidak ratu pada baris pertama berada sebelum kolom (N+1)/2 maka baris-baris selanjutnya tetap memakai N bit. Bila posisi bidak ratu pada baris pertama berada di kolom (N+1)/2 maka baris kedua hanya menggunakan ((N+1)/2)-2 bit terkanan saja karena bila bit lainnya diproses akan menghasilkan solusi hasil pencerminan dari pemrosesan ((N+1)/2)-2 bit terkanan sebelumnya, yang berarti akan terjadi penghitungan ganda solusi. Gambar 4 memperlihatkan tempat yang boleh diisi oleh bidak ratu pada baris dua. Baris-baris selanjutnya tetap menggunakan N bit.

Gambar 4. Lingkaran hitam adalah posisi yang mungkin untuk baris 2 untuk N ganjl bila bidak pada baris pertama berada di tengah-tengahl.

Jumlah solusi yang sebenarnya bila menggunakan cara pencerminan ini adalah dua kali jumlah solusi yang diperoleh. Berikut potongan program dari queens2.c yang menggunakan pendekatan pencermianan saja:

```
void Queens1(unsigned ld,
             unsigned cols,
             unsigned rd,
             int baris)
{
  if(baris<N){
    unsigned current, poss=ld|cols|rd;
    if((br==1)||(br==2)&&(d==batas)&&(N&1))
       poss|=((1<<N)-1)^((batas<<1)-1);
    do{
       poss|=current=~poss&(poss+1);
       if( ( ((ld|current)<<1) |
             (cols|current)    |
             ((rd|current)>>1)
           )< UINT_MAX)
       {
          Queens1( (ld|current)<<1,
                   cols|current,
                   (rd|current)>>1,
                   baris+1);
       }
    }while(poss<UINT_MAX);
  }else ++SOLUSI;
}
```

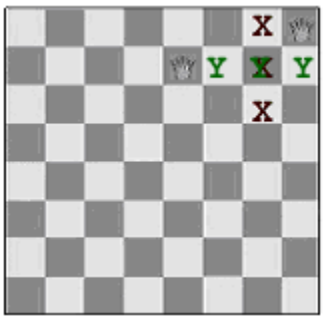
dengan

```
batas=1<<((N-1)>>1);
```

5. Perbaikan Program dengan Memanfaatkan Semua Sifat Kesimetrian dari Solusi

Meskipun perbaikan dengan mengeksploitasi pencerminan telah lebih mempercepat eksekusi program, sifat simetri lainnya dari solusi masih belum diterapkan. Pengoptimalan masih bisa dilakukan dengan memanfaatkan prinsip rotasi dan simetri berbagai arah yang bisa diterapkan terhadap solusi yang telah diperoleh. Metode ini terbagi 2 pola:

i. Bila pada baris pertama bidak ratu ditempatkan pada kolom pertama (bit terkanan).



Gambar 5. X adalah tempat yang tidak boleh ditempati oleh bidak ratu selanjutnya

Bila pada baris pertama bidak ratu ditempatkan di kolom 1 (bit terkanan) maka bidak ratu selanjutnya lainnya tidak boleh ditempatkan pada posisi X. Hal ini untuk menghindari penghitungan ganda dari solusi yang telah diperoleh dari proses sebelumnya.

Penjelasannya adalah karena kita selalu mencari posisi untuk bidak ratu mulai dari bit 0 terkanan maka tempat bertanda Y sudah pernah diproses, berarti pula tempat bertanda X sudah pernah diproses. Hal ini mudah dilihat bila kita merotasi papan 90 derajat searah jarum jam dan mencerminkannya secara horisontal.

Untuk memproses hal ini kita buat fungsi `QueensPojok()` yang terdapat dalam program queens3.c yang memodifikasi fungsi `Queens()` sebelumnya.

```
int batas;
unsigned SOLUSI8;

void QueensPojok(unsigned ld,
                 unsigned cols,
                 unsigned rd,
                 int baris)
{
   if(baris<N){
     unsigned current,poss=ld|cols|rd;
     if(baris<batas) poss|=2;
        while(poss<INT_MAX){
           poss|=current=~poss&(poss+1);
           if( (  ((ld|current)<<1) |
                  (cols|current)    |
                  ((rd|current)>>1)
                )<UINT_MAX)
          {
            QueensPojok( (ld|current)<<1,
                         cols|current,
                         (rd|current)>>1,
                          baris+1);
          }
     }
   }else ++SOLUSI8;
}
```

Dengan pemanggilannya adalah:

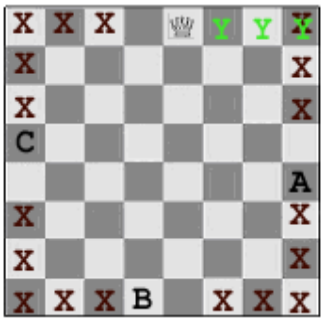
```
int temp;

for(batas=2;batas<(N-1);++batas){
    temp=1<<batas;
    QueensPojok( (2|temp)<<1,
                 all | 1| temp,
                 temp>>1,
                 3);
}
```

Jumlah solusi sebenarnya untuk pola ini adalah `SOLUSI8*8`.

ii. Bila pada baris pertama bidak ratu ditempatkan bukan pada kolom pertama.

Bila pada baris pertama bidak ratu ditempatkan bukan pada kolom pertama yaitu mulai kolom kedua sampai kolom N/2 maka bidak-bidak ratu selanjutnya tidak boleh ditempatkan pada X agar tidak terjadi penghitungan ganda solusi yang telah didapat sebelumnya. Batas penggunaan bit pada baris pertama adalah hanya sampai N/2 bukan (N+1)/2 seperti cara pencerminan sebelumnya . Untuk N ganjil, dengan cara ini maka pencarian solusi yang dimulai pada kolom (N+1)/2 baris pertama dilakukan dengan menganggap baris ke-1 adalah kolom ke-1 dan selanjutnya, yang berarti secara tidak sengaja telah dilakukan oleh proses sebelumnya.



Gambar 6. X adalah tempat yang tidak boleh ditempati bidak ratu

Penjelasannya gambar 6 adalah: karena kita selalu mencari posisi untuk ditempati bidak ratu mulai dari tempat kosong terkanan dahulu maka tempat bertanda Y sudah pernah diproses, berarti pula tempat bertanda X sudah pernah diproses. Hal ini bisa dilihat dengan merotasikan 90, 180, atau 270 derajat dan mencerminkan tiap hasil rotasi secara horisontal sehingga bisa terlihat menghasilkan kondisi papan yang sama dengan gambar 6.

Bila dalam suatu solusi terdapat bidak ratu pada posisi A atau B atau C maka perlu dicek apakah perotasian solusi sebesar 90 derajat(kasus A) atau 180 derajat(kasus B) atau 270 derajat(kasus C) menghasilkan posisi yang sama dengan solusi yang kita peroleh ini.

Bila rotasi 90 derajat menghasilkan solusi yang sama maka dari solusi ini hanya menghasilkan 2 solusi unik (termasuk dirinya sendiri). Hal ini terjadi karena bila rotasi 90 derajat sama dengan asalnya maka begitu rotasi 180 derajat dan 270 derajat akan menghasilkan sama dengan posisi asal. Lebih jauh lagi, pencerminan terhadap tiap hasil rotasi ini terhadap sumbu horisontal tentu akan sama dengan pencerminan dari solusi awal. Jadi cuma ada 2 solusi unik yaitu solusi itu sendiri dan pencerminannya secara horisontal.

Bila rotasi 180 derajat menghasilkan solusi yang sama dengan asalnya maka dari solusi ini hanya menghasilkan 4 solusi unik (termasuk dirinya sendiri). Hal ini terjadi karena bila rotasi 180 derajat sama dengan posisi asal maka begitu pula pencerminan dari hasil rotasi ini terhadap sumbu horisontal akan sama dengan pencerminan dari solusi asalnya. Hal ini berarti pula rotasi solusi sebesar 90 derajat akan sama dengan hasil rotasi 270 derajat yang juga berdampak bahwa pencerminannya masing-masing hanya menghasilkan 2 solusi baru yang sama. Jadi hanya

ada 4 solusi yang bisa dibentuk bila perotasian solusi sebesar 180 derajat hasilnya sama dengan dirinya..

Untuk menangani proses ini dibuatlah fungsi rekursif `QueensDalam()` dan fungsi `Check()` untuk mengecek rotasi. Kode program tidak kami tampilkan di sini karena memerlukan banyak ruang. Program yang mengimplementasikan hal ini adalah queens3.c yang bisa di-*download* di situs yang telah disebutkan sebelumnya.

6. Pengoptimalan Program dengan Peng-*unroll*-an Perulangan

Usaha pengoptimalan terakhir dilakukan dengan meng-*unroll* setiap perulangan yang ada. Hal ini dilakukan dengan mengubah perulangan menjadi bentuk `for` tanpa kondisi berhenti di dalam parameter `for`, kondisi berhenti ditempatkan dalam isi perulangan dalam bentuk `break`.

Pada mesin modern, meng-*unroll* perulangan dapat menghindari *pipeline stalls* sehingga akan mengurangi percabangan dan meningkatkan *instrucion-level parallelism*[1] yang bisa mempercepat waktu eksekusi program. Dalam program terakhir (queens4.c), peng-*unroll*-an diatas delapan kali tidak lagi mempercepat waktu eksekusi program.

7. Analisi Hasil Eksekusi Program

Tabel 1memperlihatkan waktu eksekusi program dari queens4.c. Waktu eksekusi program ini lebih cepat dibandingkan dengan waktu eksekusi dari queens0.c, queens1.c, queens2.c,dan queens3.c. Waktu eksekusi ketiga program lainnya tidak kami cantumkan karena keterbatasan tempat.

Penggunaan simetri sangat berdampak terhadap eksekusi program, waktu eksekusi queens1.c adalah dua kali lebih lama daripada queens4.c. Peng-*unroll*-an perulangan mampu mempercepat waktu eksekusi program sekitar 15%. Jumlah solusi yang dihasilkan untuk keempat program adalah sama dan telah sesuai dengan data yang kami miliki.

Tabel 1. Waktu Eksekusi Rata-Rata queens4.c pada Intel Celeron 2.4GHz(RAM 256 MB) dengan kompilator gcc-2.95.2

N	Jumlah Solusi	Waktu Eksekusi
10	724	0,000 detik
11	2680	0,000 detik
12	14200	0,016 detik
13	73712	0,039 detik
14	365596	0,2 detik
15	2279184	1,234 detik
16	14772512	7,85 detik
17	95815104	1 menit 50 detik

7. Kesimpulan

Pemecahan persoalan *N-Queens* dengan mengeksploitasi simetri dan menggunakan bit sebagai struktur data mampu mempercepat eksekusi program. Metode pengeksploitasian simetri ini mampu mempercepat eksekusi program hingga hampir dua kali lipat dibandingkan dengan metode tanpa pengeksploitasian simetri.

Daftar Pustaka

[1] B. Jon, *Programming Pearls*, Second Edition, Addison Wesley, NewYork, 2000.