Backtracking

Öğrenci Adı: Burak Atalay

Öğrenci Numarası: 22011641

Dersin Eğitmeni: M. Elif Karslıgil

Video Linki: https://youtu.be/S9GHAZWiNf4

1- Problemin Çözümü:

Brute Force: Her ihtimalin denenmesi için tahtadaki n^2 hücre herhangi bir kurala bağlı kalınmadan n adet queen ile doldurulmalı. Bu her bir hücre için queen koymak ve koymamak olarak iki ihtimale göre dallanarak yapılabilir. Fonksiyona verilen hücreye verilen queen'i yerleştirip sonraki hücre ve sonraki queen için bir çağrı yapılır. Ya da verilen hücreye verilen queen'i yerleştirmeyip sonraki hücreye şu anki queen'i yerleştirmek için bir çağrı yapılır. n. queen'in yerleştirildiği çağrıdan sonraki çağrıda doğru bir yerleştirme olup olmadığı isCorrectSolutionBruteForce fonksiyonu ile tahta bir kez gezilerek kontrol edilir. Yani aslında n^2 hücreye n adet queenin yerleştirildiği tüm durumlar deneniyor. C(n^2, n) adet durum var.

Optimized 1: Aynı satırda birden fazla queen olamayacağı bilinerek yerleşim yapabilmek için q. queen'i yerleştirmek için yapılan çağrıda queen yalnızca q. satırdaki sütunlara yerleştirilir ve sonraki queen için olan çağrı yapılır. Yine aynı şekilde n. queen yerleştirildikten sonraki çağrıda doğru yerleşim yapılıp yapılmadığı isCorrectSolutionOptimized1 fonksiyonu yardımıyla kontrol edilir. Bunda brute force'takinden farklı olarak aynı satırda birden fazla queen var mı kontrolü yok. Çünkü zaten ona göre yerleşim yapılıyor.

Optimized 2: Hem aynı satırda hem de aynı sütunda birden fazla queen olamayacağı bilinerek yerleşim yapabilmek için q. queen'i yerleştirmek için yapılan çağrıda queen yalnızca q. satırdaki başka bir queen'in yerleşmediği sütunlara yerleştirilir ve sonraki queen için olan çağrı yapılır. Sütunun kullanılıp kullanılmadığına n uzunluğunda bir diziye bakılarak O(1) karmaşıklıkta karar verilir. Dizide eğer sütun kullanıldıysa 1 kullanılmadıysa 0 değeri vardır. Yine aynı şekilde n. queen yerleştirildikten sonraki çağrıda doğru yerleşim yapılıp yapılmadığı isCorrectSolutionOptimized2 fonksiyonu yardımıyla kontrol edilir. Bunda da öncekinden farklı olarak aynı sütunda birden fazla var mı kontrolü yok.

Backtracking: Hem aynı satırda hem aynı sütunda hem aynı ana diyagonalde hem de aynı ikincil diyagonalde birden fazla queen olamayacağı bilinerek yerleşim yapabilmek için q. queen'i yerleştirmek için yapılan çağrıda queen yalnızca q. satırdaki kullanılmayan sütunlara eğer bu hücrenin ait olduğu diyagonaller de kullanılmıyorsa yerleştirilir ve sonraki queen için olan çağrı yapılır. Sütunun kullanılıp kullanılmadığına Optimized 2 yöntemindeki gibi kara verilir. Diyagonallerin kullanılıp kullanılmadığına da benzer şekilde O(1) karmaşıklıkta karar veriliyor. Ana diyagonal için "satır indisi - sütun indisi" değeri bir ana diyagonaldeki her hücre için aynıdır ve bu değer ana diyagonaller için -(n-1) ve n-1 arasında değişir. Yani toplamda 2n-1 değer, 2n-1 tane ana diyagonal var. Bu uzunlukta 0-1 değerleri alan bir dizi kullanılırsa çağrıda yerleştirilmeye çalışılan hücrenin satır sütun değerleri ile hangi diyagonale ait olduğu ve o diyagonalin kullanılıp kullanılmadığı anlaşılır. İkincil diyagonal için de aynı mantık var. "Satır + sütun" değeri bir ikincil diyagonaldeki her hücre için aynıdır ve bu değer ikincil diyagonaller için 0 ve 2n-1 aralığında değişir. Yani yine toplamda 2n-1 değer, 2n-1 ikincil diyagonal var. n. queen yerleştirildikten sonra doğru yerleşmiş mi kontrolü yapılmaz. Zaten queen'ler birbirini görmeyecek şekilde yerleşim yapılıyor.

2- Karşılaşılan Sorunlar:

İlk zorluk, backtracking kısmında diyagonallerin kullanılıp kullanılmadığını tespit etmek için çözümde bahsettiğim O(1)'lik çözümün ilk başta aklıma gelmemesiydi. Bu yüzden queen'i yerleştirmeden önce diyagonalleri queen var mı diye dolaşıyordum. Bu çok verimsiz bir

yöntemdi. Sonrasında Optimized 2'den daha verimli bir çözüm olması gerektiğini düşündüğüm için biraz kafa yordum ve çözümde bahsettiğim eşitlikler gözüme çarptı.

İkinci zorluk, isCorrectSolution fonksiyonlarında (3 tane var) kullanılan yardımcı dizilerin fonksiyonların her çağrılmasında allocate edilmeleriydi. Bunun ilk başta bir sorun olacakğını düşünmemiştim ancak n = 8 için brute force çalıştırdığımda bitmesi yaklaşık 20 dk sürdü. Bunun malloc/calloc fonksiyonlarının heap'ten allocate etmek için sistem çağrısı yapmalarından veya programın heap segmentinde yeterli boşluk varsa bile uygun yer aramasından ve n = 8 için 4.4 milyar hamle olduğundan 4.4 milyar kere isCorrectSolutionBruteForce fonksiyonunun çağrılmasından kaynaklandığını düşündüm. Bu sebeple kullanacağım yardımcı dizileri bir struct içinde tutup bir kere allocate edip fonksiyona parametre olarak verdim. Bu sayede Brute Force n = 8 için 20 dakikadan 9 dakikaya düştü.

3- Karmaşıklık Analizi:

Karmaşıklık analizinde ara adımları saymadım. Kaç hamle yapılır durumunu hesapladım. Yani mesela n queen de yerleştirilince bir hamle olarak kabul edersek ona göre bir analiz oldu.

Brute Force:

```
void bruteForce(int** board, int n, int index, int q, int* steps, int* solutions){
    (*steps)++;
    if(q == n){ //since q is given 0 at the initial call, if it is n in this call, then we placed the nth queen in the prior call
        if(isCorrectSolution(board, n)){
            if(PRINT){ //if macro is 1 then it prints the solution
                printf("\nSolution %d...\n", (*solutions) + 1);
                drawBoard(board, n);
            }
            (*solutions)++;
        }
        return;
    }

    if(index >= n * n) return; //if index is n^2 then it is out of bounds since we have n^2 cells

    int row = index / n, col = index % n;

    board[row][col] = 1; //placing the qth queen in this cell
    bruteForce(board, n, index + 1, q + 1, steps, solutions); //making call to place (q+1)th queen in the next cell

    board[row][col] = 0; //not placing the qth queen in this cell
    bruteForce(board, n, index + 1, q, steps, solutions); //making call to place qth queen in the next cell
}
```

Her bir hücre için bir queen'in yerleştirildiği bir de yerleştirilmediği durum için birer çağrı yapılıyor. Sonucunda n. queen yerleştirildikten sonraki çağrıda doğru yerleşim olup olmadığı kontrol ediliyor. Yani aslında bu kod, n^2 hücreden n tanesini seçiyor. Yaklaşık C(n^2, n) çağrı yapar. Bu da bize O(C(n^2, n)) karmaşıklık verir. Bu çok yüksek bir karmaşıklık olduğundan benim bilgisayarım n = 8'den sonrası için bu modda çok fazla süre talep ediyor. n = 8 için 9 dakika sürdü, n = 9 için denemedim.

Optimized 1:

```
void optimized1(int** board, int n, int q, int* steps, int* solutions){
    (*steps)++;
    if(q == n){
        if(isCorrectSolution(board, n)){
            if(PRINT){
                printf("\nSolution %d...\n", (*solutions) + 1);
                drawBoard(board, n);
            }
            (*solutions)++;
        }
        return;
    }

    int i;
    for(i=0;i<n;i++){ //iterates over each column in the qth row
        board[q][i] = 1; //place qth queen in ith column of qth row
        optimized1(board, n, q + 1, steps, solutions); //making a call to place (q+1)th queen
        board[q][i] = 0; //unmarking this cell since we tried each possibility after placing qth queen in this cell
    }
}
```

Her bir satır bir queen'e ayrıldığı için q. queen'i q. satırdaki tüm sütunlara sırayla yerleştirir ve sonraki queen için çağrı yapar. Yani her bir queen'in n hücreye yerleşme ihtimali var. Bu, n queen için n^n hamle eder. Fonksiyon kendisini kaç kere çağırdı konusunu bu fonksiyon için recursion ağacını düşünürsek yorumlayabiliriz. İlk çağrı sonrası 0. queen için olan çağrı içinde yapılacak n çağrı var. Bu çağrıların hepsi 1. queen için yapılacak. 1. queen de n çağrı yapacak, hepsi 2. queen için. Yani 2. queen için 1 * n * n = n^2 çağrı yapılacak. Bu şekilde gidersek yerleşecek son queen olan (n-1). queen de n çağrı yapacak. Şimdi 0. queen için n^0, 1. queen için n^1, 2. queen için n^2, n-1. queen için n^(n-1) çağrı yapılmış oldu. Son queen sonrası bir çağrı daha yapılacak ve o çağrıda doğru yerleşmiş mi kontrolü olacak. Ağacın son seviyesinde n^(n-1) çağrı vardı, n^(n-1) * n = n^n çağrı yapılacak. Toplamda ise n^0 + n^1 + …. + n^n çağrı yapılacak. Bu da seri formülünden geliyor. Yerine koyduğumuzda gerçekten sonuç doğru oluyor. Ekran görüntülerindeki çağrı sayısı ile aynı çıkıyor. Ancak big O analizi yaparsak yapılacak hamle sayısı olan n^n rekörsif çağrı sayısını domine eder. Sonuç olarak karmaşıklığa theta(n^n) diyebiliriz. Theta denir çünkü en iyi durumda da en kötü durumda da bu kadar çağrı olacak, çağrı sayısını da hamle sayısı olan n^n domine ediyor.

Optimized 2:

```
void optimized2(int** board, int n, int* cols, int q, int* steps, int* solutions){
    (*steps)++;
    if(q == n){
        if(isCorrectSolution(board, n)){
            if(PRINT){
                printf("\nSolution %d...\n", (*solutions) + 1);
                drawBoard(board, n);
            }
            (*solutions)++;
        }
        return;
    }

    int i;
    for(i=0;i<n;i++){ //iterates over each column in the qth row
        if(!cols[i]){ //if ith column is not used
            board[q][i] = 1; //placing the queen in qth row and ith column
            cols[i] = 1; //marking the ith column
            optimized2(board, n, cols, q + 1, steps, solutions); //making a call to place (q+1)th queen
            board[q][i] = 0; //unmarking this cell since we tried each possibility after placing qth que
            cols[i] = 0; //unmarking the ith column
        }
    }
}
```

Queen'ler hem aynı satırda hem aynı sütunda olamayacağı için q. queen q. satırdaki sütunlara eğer o sütun kullanılmadıysa yerleştiriliyor ve her yerleştirme için çağrı yapılıyor. Bu bize 0. queen için n ihtimal, 1. queen için n-1, (n-1). queen için 1 ihtimal verir. Yani n! kadar çağrı olur. Bu da theta(n!)

karmaşıklık yapar. Optimized 1'deki gibi çağrı sayısı için uzun analiz yapmayacağım çünkü aynı şekilde hamle sayısı olan n! çağrı sayısını domine eder. İlk seviyede (0. queen) 1 çağrı, ikinci seviyede n çağrı, 3. Seviyede n.(n-1) çağrı diye gidiyor. Son seviyede n! var. Bu tüm karmaşıklığı domine eder.

Backtracking:

```
void backtracking(int** board, int n, int* cols, int* mainDiagonal, int* secondaryDiagonal, int q, int* steps, int* solutions){
    (*steps)++;
    if(q == n){
        if(isCorrectSolution(board, n)){
            if(PRINT){
                printf("\nSolution %d...\n", (*solutions) + 1);
                drawBoard(board, n);
            }
            (*solutions)++;
        }
        return;
    }

    int i;
    for(i=0;i<n;i++){ //iterating over each column
        if(!cols[i] && !mainDiagonal[q - i + n - 1] && !secondaryDiagonal[q + i]){ //if ith column and both diagonals are not used
            board[q][i] = 1; //placing the queen in qth row and ith column
            cols[i] = 1; //marking the ith column
            mainDiagonal[q - i + n - 1] = 1; //marking the main diagonal
            secondaryDiagonal[q + i] = 1; //marking the secondary diagonal
            backtracking(board, n, cols, mainDiagonal, secondaryDiagonal, q + 1, steps, solutions); //making a call to place (q+1)
            board[q][i] = 0; //unmarking this cell since we tried each possibility after placing qth queen in this cell
            cols[i] = 0; //unmarking the ith column
            mainDiagonal[q - i + n - 1] = 0; //unmarking the main diagonal
            secondaryDiagonal[q + i] = 0; //unmarking the secondary diagonal
        }
    }
}
```

Optimized 2'deki kontrole ek olarak burada diyagonal kontrolü de var. Yine q. queen q. satırdaki sütunlara eğer ki sütun ve diyagonaller başka bir queen tarafından kullanılmıyorsa yerleştirilir. Burada Optimized 1 ve Optimized 2'deki gibi net bir karmaşıklık tight bound veremeyiz çünkü kullanılan sütunlar ve diyagonaller kesişebilir. Mesela 0. queen için yine n ihtimal var ve 1. queen için n-3 ihtimal var çünkü kullanılan sütuna ek olarak ana ve ikinci diyagonallere de yerleşemez. Ancak 2. Queen n-6 ihtimal var diyemeyiz. Mesela 2. queen için 0. queen'in ana diyagonali ile 1. queen'in sütunu kesişebilir. Bu sebeple net bir karmaşıklık verilemez. Bu 0. queen'in yerleştirildiği her hücre için değişkenlik gösterir ancak performansın Optimized 2'den daha iyi olduğu açık. Yani O(n!) gibi bir üst sınır verilebilir. Ama tabii ki her halükarda n!'den daha az çağrı olacak.

4-Ekran Çıktıları:

NOT: Ekran görüntülerinde dikkat edersek the number of recursion calls bir de the number of moves yazdırılıyor. N queen de yerleştirilince bir hamle (move) saydım.

Input 1:

n = 5 - mod: brute force

Output 1:

```
----------------------------------------
Brute Force...
----------------------------------------

Solution 1...

    00   01   02   03   04
  -----------------------------
00 | Q |    |    |    |    |
  -----------------------------
01 |    |    | Q |    |    |
  -----------------------------
02 |    |    |    |    | Q |
  -----------------------------
03 |    | Q |    |    |    |
  -----------------------------
04 |    |    |    | Q |    |
  -----------------------------


Solution 2...

    00   01   02   03   04
  -----------------------------
00 | Q |    |    |    |    |
  -----------------------------
01 |    |    |    | Q |    |
  -----------------------------
02 |    | Q |    |    |    |
  -----------------------------
03 |    |    |    |    | Q |
  -----------------------------
04 |    |    | Q |    |    |
  -----------------------------
```

```
Solution 3...

    00   01   02   03   04
  -----------------------------
00 |    | Q |    |    |    |
  -----------------------------
01 |    |    |    | Q |    |
  -----------------------------
02 | Q |    |    |    |    |
  -----------------------------
03 |    |    | Q |    |    |
  -----------------------------
04 |    |    |    |    | Q |
  -----------------------------


Solution 4...

    00   01   02   03   04
  -----------------------------
00 |    | Q |    |    |    |
  -----------------------------
01 |    |    |    |    | Q |
  -----------------------------
02 |    |    | Q |    |    |
  -----------------------------
03 | Q |    |    |    |    |
  -----------------------------
04 |    |    |    | Q |    |
  -----------------------------
```

```
Solution 5...

    00   01   02   03   04
  -----------------------------
00 |    |    | Q |    |    |
  -----------------------------
01 | Q |    |    |    |    |
  -----------------------------
02 |    |    |    | Q |    |
  -----------------------------
03 |    | Q |    |    |    |
  -----------------------------
04 |    |    |    |    | Q |
  -----------------------------


Solution 6...

    00   01   02   03   04
  -----------------------------
00 |    |    | Q |    |    |
  -----------------------------
01 |    |    |    |    | Q |
  -----------------------------
02 |    | Q |    |    |    |
  -----------------------------
03 |    |    |    | Q |    |
  -----------------------------
04 | Q |    |    |    |    |
  -----------------------------
```

```
Solution 7...

    00   01   02   03   04
  -----------------------------
00 |    |    |    | Q |    |
  -----------------------------
01 | Q |    |    |    |    |
  -----------------------------
02 |    |    | Q |    |    |
  -----------------------------
03 |    |    |    |    | Q |
  -----------------------------
04 |    | Q |    |    |    |
  -----------------------------


Solution 8...

    00   01   02   03   04
  -----------------------------
00 |    |    |    | Q |    |
  -----------------------------
01 |    | Q |    |    |    |
  -----------------------------
02 |    |    |    |    | Q |
  -----------------------------
03 |    |    | Q |    |    |
  -----------------------------
04 | Q |    |    |    |    |
  -----------------------------
```

```
Solution 9...

    00    01    02    03    04
  -----------------------------
00 |     |     |     |     |  Q  |
  -----------------------------
01 |     |  Q  |     |     |     |
  -----------------------------
02 |     |     |     |  Q  |     |
  -----------------------------
03 |  Q  |     |     |     |     |
  -----------------------------
04 |     |     |  Q  |     |     |
  -----------------------------


Solution 10...

    00    01    02    03    04
  -----------------------------
00 |     |     |     |     |  Q  |
  -----------------------------
01 |     |     |  Q  |     |     |
  -----------------------------
02 |  Q  |     |     |     |     |
  -----------------------------
03 |     |     |     |  Q  |     |
  -----------------------------
04 |     |  Q  |     |     |     |
  -----------------------------
```

```
-------------------------------------------------
Brute Force...
the number of recursion calls: 136811
the number of moves: 53130
the number of solutions: 10
Time spent: 0.005960 seconds
-------------------------------------------------
```

Input 2:

n = 5 – mod : all modes

(çözümler bundan sonraki örnekler için yazdırılmadı çünkü çok fazla görüntü gerekiyor.)

Output 2:

```
-------------------------------------------------
Brute Force...
the number of recursion calls: 136811
the number of moves: 53130
the number of solutions: 10
Time spent: 0.005960 seconds
-------------------------------------------------
```

```
-------------------------------------------------
Optimized 1...
the number of recursion calls: 3906
the number of moves: 3125
the number of solutions: 10
Time spent: 0.000626 seconds
-------------------------------------------------
```

```
--------------------------------------------------
Optimized 2...
the number of recursion calls: 326
the number of moves: 120
the number of solutions: 10
Time spent: 0.000377 seconds
--------------------------------------------------
```

```
--------------------------------------------------
Backtracking...
the number of recursion calls: 54
the number of moves: 10
the number of solutions: 10
Time spent: 0.000374 seconds
--------------------------------------------------
```

Input 3:

n = 6 – mod : all modes

Output 3:

```
--------------------------------------------------
Brute Force...
the number of recursion calls: 4782991
the number of moves: 1947792
the number of solutions: 4
Time spent: 0.119366 seconds
--------------------------------------------------
```

```
--------------------------------------------------
Optimized 1...
the number of recursion calls: 55987
the number of moves: 46656
the number of solutions: 4
Time spent: 0.002369 seconds
--------------------------------------------------
```

```
--------------------------------------------------
Optimized 2...
the number of recursion calls: 1957
the number of moves: 720
the number of solutions: 4
Time spent: 0.000147 seconds
--------------------------------------------------
```

```
------------------------------------------------
Backtracking...
the number of recursion calls: 153
the number of moves: 4
the number of solutions: 4
Time spent: 0.000095 seconds
------------------------------------------------
```

Input 4:

n = 7 – mod : all modes

Output 4:

```
------------------------------------------------
Brute Force...
the number of recursion calls: 204045619
the number of moves: 85900584
the number of solutions: 40
Time spent: 5.563460 seconds
------------------------------------------------
```

```
------------------------------------------------
Optimized 1...
the number of recursion calls: 960800
the number of moves: 823543
the number of solutions: 40
Time spent: 0.044903 seconds
------------------------------------------------
```

```
------------------------------------------------
Optimized 2...
the number of recursion calls: 13700
the number of moves: 5040
the number of solutions: 40
Time spent: 0.001784 seconds
------------------------------------------------
```

```
------------------------------------------------
Backtracking...
the number of recursion calls: 552
the number of moves: 40
the number of solutions: 40
Time spent: 0.001398 seconds
------------------------------------------------
```

Input 5:

n = 8 – mod : all modes

Output 5:

```
--------------------------------------------------
Brute Force...
the number of recursion calls: 10261319121
the number of moves: 4426165368
the number of solutions: 92
Time spent: 560.880000 seconds
--------------------------------------------------
```

```
--------------------------------------------------
Optimized 1...
the number of recursion calls: 19173961
the number of moves: 16777216
the number of solutions: 92
Time spent: 3.644000 seconds
--------------------------------------------------
```

```
--------------------------------------------------
Optimized 2...
the number of recursion calls: 109601
the number of moves: 40320
the number of solutions: 92
Time spent: 1.181000 seconds
--------------------------------------------------
```

```
--------------------------------------------------
Backtracking...
the number of recursion calls: 2057
the number of moves: 92
the number of solutions: 92
Time spent: 1.148000 seconds
--------------------------------------------------
```