Algoritma Analizi

Ödev - 4

N-Queens Problem

Öğrenci Adı: Adem Alp ŞAHİN

Öğrenci Numarası: 22011090

Dersin Eğitmeni: Mehmet Amaç GÜVENSAN

Video Linki: https://youtu.be/TEOyiqr30YY

1- Problemin Çözümü:

Problemimizin çözümü 4 farklı yol ile sağlanmıştır:
1. Brute Force: C(n^2, n) olasılığın tamamı for döngüsü ve recursive yaklaşım kullanılarak çözümlere ulaşılmıştır.

2. Optimized_1: Burada en baştan her satıra sadece 1 vezir gelecek şekilde diye düşünülüp ona göre yerleştirme yapılmıştır, n^n olasılığın tamamı denenmiştir.

3. Optimized 2: Burada da satır ve sütun çakışması olmaması için çözümler ona göre manipüle edilmeye çalışarak oluşturulmuştur. n! Olasılığın tamamı denenmiştir.

4. Backtracking: Döngünün içinde Queen'in yerleştirilmeye çalışıldığı kare güvenliyse o ihtimalden recursive şekilde öteki column ve satırlar için ilerleyecek şekilde devam edilir. Güvensiz kareye gelindiyse bir önceki adıma gidilir ve bir sonraki column denenir. Çözüm bulunduğunda yani column size'a eşit olduğunda çözüm yazdırılır.

2- Karşılaşılan Sorunlar:

Brute Force Modülünü yapmaya çalışırken başta iki boyutlu dizi ile çözmeye çalıştım ancak bir süre sonra çok kompleks bir kod yazmak zorunda kaldım, sonrasında da tek boyutlu "size*size" boyutlu bir dizi ile daha rahat hallettim. Taşların Konumunu ise "col = pos % size" ve "row = pos / size" işlemleriyle hesapladım.

3- Karmaşıklık Analizi:

Brute Force:
generateCombinations(int positions[], int start, int depth, int size, int solutions[] long long int iterations[])

```
for i <- start ; i < size*size; i++
        positions[depth] = i;
        generateCombinations(positions, i + 1, depth + 1, size, solutions,
iterations);
```

O(N) = N^2*(N^2-1)*….*(N^2-N) /N! = C(N^2, N);

Optimized 1:

void generatePermutations(int *positions, int depth, int size, int *solutions, long long int *iterations)

for (col = 0; col < *size*; col++) {

positions[*depth*] = *depth* * *size* + col;

generatePermutations(*positions*, *depth* + 1, *size*, *solutions*, *iterations*, *table*);

}

O(n) = n*n**…n kere…**n = n^n

Optimized 2:

void generatePermutationsWithoutRowColConflicts(*int *positions*, *int depth*, *int size*, *int *solutions*, *long long int *iterations)*

```
int col;
  for (col = 0; col < size; col++) {
    int valid = 1;
    int j = 0;
    while (j < depth && valid) {
      if (positions[j] == col) {
        valid = 0;
      }
      j++;
    }
    if (valid)
      positions[depth] = col;
      generatePermutationsWithoutRowColConflicts(positions, depth + 1, size, solutions, iterations);
```

O(n) = N*(N-1)*(N-2)*….*1 = N!

Backtracking Mode:

```
int solveNQueensBacktrackUtil(char **table, int col, int size, long long int *iterations, int *solutions)

        for (i = 0; i < size; i++) {

        res = solveNQueensBacktrackUtil(table, col + 1, size, iterations, solutions) || res;

                table[i][col] = 0;

    }
```

Worstcase is O(N) = N!

4-Ekran Çıktıları:

Brute Force(N=8):

```
Number of valid solutions: 92
Number of iterations: 4426165368
Execution time: 79.663000 seconds
```

OPTIMIZED_1(N=8):

```
Number of valid solutions: 92
Number of iterations: 16777216
Execution time: 0.559000 seconds
```

OPTIMIZED_2(N=8):

```
Number of valid solutions: 92
Number of iterations: 40320
Execution time: 0.169000 seconds
```

BACKTRACKING(N=8):

```
Number of valid solutions: 92
Number of iterations: 2057
Execution time: 0.160000 seconds
```

runAllMods(N=8):