Algoritma Analizi 4. Ödev

N-Queen Problem

Öğrenci Adı: Mehmet Ali Duran

Öğrenci Numarası: 21011090

Dersin Eğitmeni: Mehmet Amaç Güvensan

Video Linki: https://youtu.be/IQuzn9PSem4

1- Problemin Çözümü

Verilen problemde bizden bir N-vezir problemi çözümü gerçekleştirmemiz istenmiştir. Bu problem, satranç tahtasında N adet vezirin birbirini tehdit etmeyecek şekilde yerleştirilmesini amaçlar. Çözümümüz, bu probleme yönelik çeşitli algoritmaların bir simülasyonunu içerir ve her bir yöntemin performans analizi yapılır.

Çözümde **brute force**, **optimized-1**, **optimized-2** ve **backtracking** yöntemleri kullanılmıştır. Bununla birlikte, her bir yöntemin performansını ölçmek için zaman hesaplaması yapılmış ve çözümlerin toplam sayısı kullanıcıya raporlanmıştır. Problemin çözümüne yönelik genel adımlar şu şekilde sıralanmıştır:

1. Kullanıcıdan satranç tahtasının boyutu olan NNN değeri alınır. NNN değeri en az 4 olmak zorundadır.
2. Her çözüm yöntemi birer fonksiyon olarak ayrı ayrı tanımlanmıştır:
    - **Brute Force**: Tüm olası kombinasyonları dener ve geçerli tahtaları kontrol eder.
    - **Optimized-1**: Sadece sütun çakışmalarını kontrol ederek çözümleri bulur.
    - **Optimized-2**: Sütun ve çapraz çakışmaları kontrol ederek çözümleri optimize eder.
    - **Backtracking**: Rekürsif bir şekilde tüm çözümleri bulur ve geçersiz durumlarda geriye döner.
3. Kullanıcıya, seçilen yöntem için çözüm sayısı ve geçen süre rapor edilir.
4. Tüm yöntemlerin sıralı bir şekilde çalıştırıldığı bir allMethods fonksiyonu da eklenmiştir.
5. Performans ve çözüm doğruluğunu ölçmek için zaman ölçümü ve çözüm doğrulama işlemleri yapılmıştır.

Simülasyonda, satranç tahtası bir matris olarak temsil edilmiştir ve algoritmalar, matris üzerinde işlem yapmıştır. Tüm çözümler, hataların minimuma indirilmesi ve performansın artırılması amacıyla yapılandırılmıştır.

2- Karşılaşılan Sorunlar

Problemin çözümünde karşılaşılan temel sorunlardan biri, süre ölçüm işlemleri için aynı kodun tekrar tekrar yazılmasıydı. Her algoritmanın süre ölçüm işlemini gerçekleştirmek için aynı kodun birden fazla kez yazılması, kodun okunabilirliğini ve sürdürülebilirliğini olumsuz etkiliyordu. Bu sorunu çözmek amacıyla, `executeAndMeasureTime` adında bir fonksiyon yazılmıştır. Bu fonksiyon, bir referans olarak verilen fonksiyonu çalıştırmakta ve çalışma süresini ölçerek kullanıcıya rapor etmektedir. Bu sayede süre ölçüm işlemi modüler hale getirilmiş ve kod tekrarından kaçınılmıştır.

Bunun yanı sıra, kullanıcıların farklı algoritmaları tekrar tekrar çalıştırabilmesi gerekliliği dikkate alınmıştır. Kullanıcıların bu işlemleri birden fazla kez deneyebileceği göz önüne alınarak, menü yapısı bir döngü içerisine alınmıştır. Böylelikle kullanıcılar, algoritmaları istedikleri kadar çalıştırabilir, satranç tahtasının boyutunu değiştirebilir ve işlemlerini sonlandırabilir. Bu yaklaşım, programın esnekliğini ve kullanıcı etkileşimini artırmıştır

3- Karmaşıklık Analizi

1. Brute Force Yöntemi

Brute force yöntemi, tüm olası pozisyonları denetler ve bu pozisyonlardan hangilerinin geçerli bir çözüm olduğunu kontrol eder. Satranç tahtasında NNN adet vezir yerleştirileceğinden toplam N×NN \times NN×N hücre vardır ve bu hücrelerin her biri için vezir yerleştirme veya yerleştirmeme durumu vardır.

- **Toplam Kombinasyon Sayısı:** 2^(N*N)
- **Geçerli Çözümleri Kontrol Etme:** Her kombinasyonun geçerli olup olmadığını kontrol etmek O(N^2)'dir, çünkü tahtadaki tüm hücrelere bakmak gerekir.

Zaman Karmaşıklığı:

O(2^(N*N)*N^2)

Bu, çok yüksek bir karmaşıklık derecesine sahiptir ve sadece küçük N değerleri için pratik olarak çalışabilir.

---

2. Optimized-1 Yöntemi

Optimized-1 yöntemi, brute force yönteminden daha verimli bir şekilde çalışır çünkü sütun çakışmalarını kontrol eder. Sütun kontrolü, tüm sütunlara ayrı ayrı vezir yerleştirmeyi garanti eder. Ancak çapraz kontroller yapılmadığı için doğruluk sınırlıdır.

- **Her Satırda Sütunları Kontrol Etme:** Bir satırda vezir yerleştirmek O(N)'dir.
- **Toplam Satır Sayısı:** N
- **Geçerli Çözümleri Kontrol Etme:** Sütun çakışmalarını kontrol etmek O(1)'dir.

Zaman Karmaşıklığı:

O(N!)

Bu yöntem, sütun çakışmalarını kontrol ettiği için brute force yöntemine göre daha hızlıdır, ancak çapraz kontroller yapılmadığından hatalara açıktır.

---

3. Optimized-2 Yöntemi

Optimized-2 yöntemi, hem sütun hem de çapraz çakışmaları kontrol eder. Bu, vezirlerin doğru bir şekilde yerleştirilmesini garanti eder. Her adımda sütun ve çapraz çakışmaları kontrol etmek, performansı optimize eder.

- **Sütun Kontrolü:** O(N)
- **Çapraz Kontrol:** Her bir vezir için en fazla O(N) kontrol gerekir.

- **Her Satır İçin Kontrol:** O(N)

Toplam Zaman Karmaşıklığı:

O(N×N!)=O(N^2*(N-1)!)

Bu yöntem, brute force'a göre daha hızlıdır ve çapraz çakışmaları kontrol ederek doğruluğu artırır.

---

4. Backtracking Yöntemi

Backtracking yöntemi, rekürsif olarak her satır için vezir yerleştirmeye çalışır. Eğer bir çözüm geçerli değilse, önceki adıma geri dönerek başka bir çözüm arar. Bu yöntem, yalnızca geçerli çözümleri kontrol ettiği için diğer yöntemlere göre daha verimlidir.

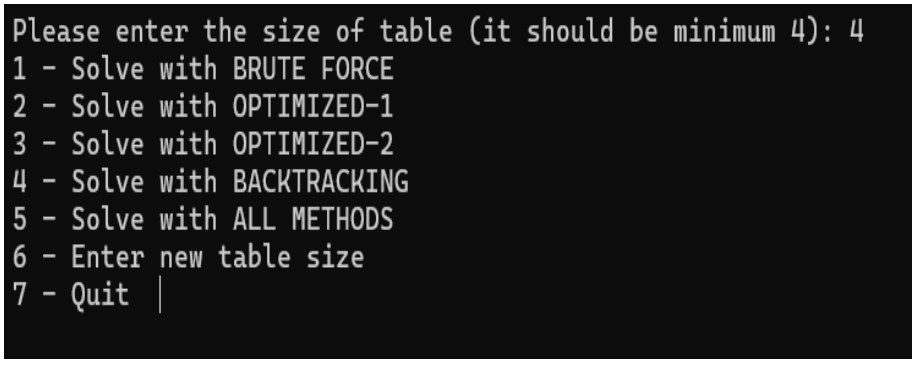
- **Her Satır İçin Sütun Kontrolü:** O(N)
- **Her Adımda Çapraz ve Sütun Kontrolleri:** O(1)
- **Tüm Kombinasyonları Gözden Geçirme:** O(N!)

Zaman Karmaşıklığı:

O(N!)

Backtracking yöntemi, sadece geçerli çözümleri kontrol ettiği için hem doğru hem de performans açısından en iyi yöntemdir. Özellikle büyük N değerleri için diğer yöntemlerden daha pratiktir.

4- Ekran Çıktıları



N=4 için Brute Force

```
Solution 1:
 0 0 1 0
 1 0 0 0
 0 0 0 1
 0 1 0 0

Solution 2:
 0 1 0 0
 0 0 0 1
 1 0 0 0
 0 0 1 0

Time elapsed 0.01 sn
solutionCount for bruteForce = 2
1 - Solve with BRUTE FORCE
2 - Solve with OPTIMIZED-1
3 - Solve with OPTIMIZED-2
4 - Solve with BACKTRACKING
5 - Solve with ALL METHODS
6 - Enter new table size
7 - Quit
```

N=10 için Back Tracking

```
 0 0 0 0 1 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 0 0 0

Solution 724:
 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 1 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 1 0 0 0 0
 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 0 0 0
 0 0 0 1 0 0 0 0 0 0

Time elapsed 7.83 sn
solutionCount for backtracking = 724
1 - Solve with BRUTE FORCE
2 - Solve with OPTIMIZED-1
3 - Solve with OPTIMIZED-2
4 - Solve with BACKTRACKING
5 - Solve with ALL METHODS
6 - Enter new table size
7 - Quit
```