Algoritma Analizi-5

Oyun Tahtası

Versiyon 1.0

Prepared by Onur Demir

Grup 2-Proje

onur.demir2@std.yildiz.edu.tr

Video Linki Anlatım: https://youtu.be/AlTsRvPQYb0

Video Linki Gösterim: https://youtu.be/kDBMhy2rjzU

İçerik Tablosu

Revizyon Tarihi

İsim	Tarih	Değişim Sebebi	Versiyon
Onur Demir	31.12.2022	Rapor bitirildi ve Video çekildi	1.0
Onur Demir	03.01.2022	Video tekrar çekildi girdiler dışarıdan renkler ile alındı	1.1

Video Linki :
Bu video linkinde algorima anlatılmıştır fakat detay mod ve girdi alınması diğer videoda gösterilmiştir.(normal mod ile çalıştıma bulunmaktadır.)
https://youtu.be/AlTsRvPQYb0

Bu video linkin detay mod ve renk girdisi verilerek yazılması yapılmıştır.
https://youtu.be/kDBMhy2rjzU

1. Giriş

1.1 Problem

NxN'lik bir oyun tahtasında N adet renk her satırda farklı sıra ile yer almaktadır. Bir satırdaki renklerin sıralanışı, renkler sağa doğru dairesel kaydırılarak değiştirilebilmektedir. Örneğin satırdaki renkler sırası ile kırmızı, mavi, yeşil, mor ise satır 1 defa sağa kaydırıldığında yeni sıralama mor, kırmızı, mavi, yeşil olur. Bir defa daha sağa kaydırılırsa yeşil, mor, kırmızı, mavi elde edilir. Son durumda her sütunda her renkten sadece 1 tane olacak şekilde oyun tahtasını geri-izleme(backtracking) yöntemi ile rekürsif olarak düzenleyen algoritmayı tasarlayınız.

1.2 Problem Tanımı

NxN lik bir oyun tahtası için en son oluşacak kazanma durumu bir "state-space" ağacı oluşturularak bulunabilir. Bunun için bir "state-space" ağacı üzerinden geri izleme uygulanabilmektedir. (Örnek oyun tablosu girişi aşağıdaki gibidir ve her bir sütunda renkler farklı olmalıdır.Bunun için sütünlarda sağa doğru kaydırmalar uygulanacaktır.)

Kırmızı	Mavi	Yeşil
Yeşil	Kırmızı	Mavi
Mavi	Kırmızı	Yeşil

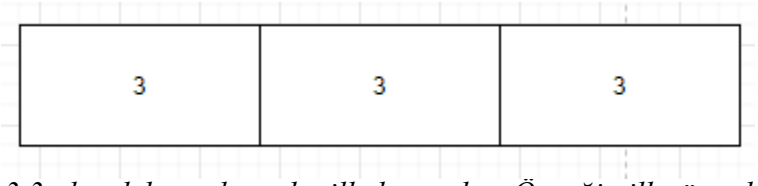
2. Yöntem

2.1 Yana kaydırma için state-space matrisi oluşturulması

Oluşturulan rastgele matris için her bir sütun için toplam sağa kayma renk sayısı kadar olursa tekrar başlangıç durumuna geleceği için oyun haritasındaki her bir sütun için renk sayısı kadar sağa kayma olacaktır bu durumların her biri "state" dir.

Kırmızı	Mavi	Yeşil

Örneğin yukarıdaki üç renk için KMY , YKM, MYK şeklinde 3 durum vardır. Her bir sütun için bir "state matrisi" oluşturulur. Toplam durum sayısı sütun sayısı kullanlarak oluşturulur.



Başlangıç durumu 3 3 3 olarak hesaplanır bu ilk durumdur. Örneğin ilk sütunda kayma olursa 2 3 3 matrisi elde edilir. Böylece N ^N lik durumların hepsi bu matriste tutulacaktır.
Doğru çıktı üretilmesi için tüm sütunların aynı sıra ile yazılmış olması gerekmektedir. Bu şart arama uzayını azaltacaktır ve daha hızlı 3 sayısının 1 e erişmesini sağlaaycaktır. Örneğin 2 3 3 durumunda tekrar ilk sütüna backtracking ile dönüyorsa sonunda 1 kere dönebileceği değer kalır ve bu değer üzerinden hepsi sıralı ise hızlı bir şekilde sonuca yakınsayacaktır. Eğer sütunlar sırasız ise 1 3 3 olacak ve başka farklı döngü oluşmayacaktır.

Çözüm için üretilen yöntem her seferinde bir üstündeki sütuna geri dönecek şekilde son satıra hiçbir sütun kesişimi olmadan gitmeye çalışmaktır.

3. Uygulama

3.1 Program Çıktısı

Rastgele verilen matris girdisi için doğru haritaların girildiğini görmek için 100 kere main fonksiyonu çağrılmıştır. Bu çağrıların uç noktalarına breakpointler konularak hangi tabloda doğru çıktı olduğu gözlenlenmiştir.

Test case çıktı format detay için aşağıda belirtildiği gibidir.
***1**.Önce tablo yazdırılır üzerinde hangi tabloda olduğu belirtilir.*
***2**.Aranacak keşişim intersection first ile belirtilit bu satırın üstündeki satırlar ile keşişim var mı diye taranır.*
***3.**Bir sütunda karşılaştırılan değerler belirtilir. Bunlar aynıysa keşişim var demektir ve çıkılmaktadır.*
***4. '**rotate_right' satırın döndürüldüğünü ifade etmektedir ve döndürülmüş satır bunun peşinden yazdırılmaktadır. (Bu satırla karşılaştırma yapılacaktır.)*
***5.**Eğer kesişim varsa backtrackin fazına geçilir ve state değerleri yazılır.*
***6**.Bu değerlerden ilki 1 ise veya tüm satırlar gezildi ise tablo keşfetme algoritmasından çıkılır ve sırasıyla yanlış ve doğru harita yazdırır.*

TEST CASE 1:

3 1 2 | 3 1 2 | 3 1 2 tablosu sıralı olarak yazdırılabilecek bir tablodur. Üçüde ardışıl sıralı. Bu örneğe normal uygulama çıktısı aşağıdadır.

```
What is the mode you using?(normal/n -- detail/d)n
What is size of game table which you want create?(1<N<10)3
                normal mode printing
        first state map

-->1 ,3 ,2 ,
1 ,2 ,3 ,
2 ,1 ,3 ,


Wrong Game Map Created.
                normal mode printing
        first state map

-->2 ,3 ,1 ,
1 ,2 ,3 ,
1 ,3 ,2 ,


Wrong Game Map Created.
                normal mode printing
        first state map

-->3 ,1 ,2 ,
3 ,1 ,2 ,
3 ,1 ,2 ,


         result state map

2 ,3 ,1 ,
3 ,1 ,2 ,
1 ,2 ,3 ,
Right Game Map Created.
```

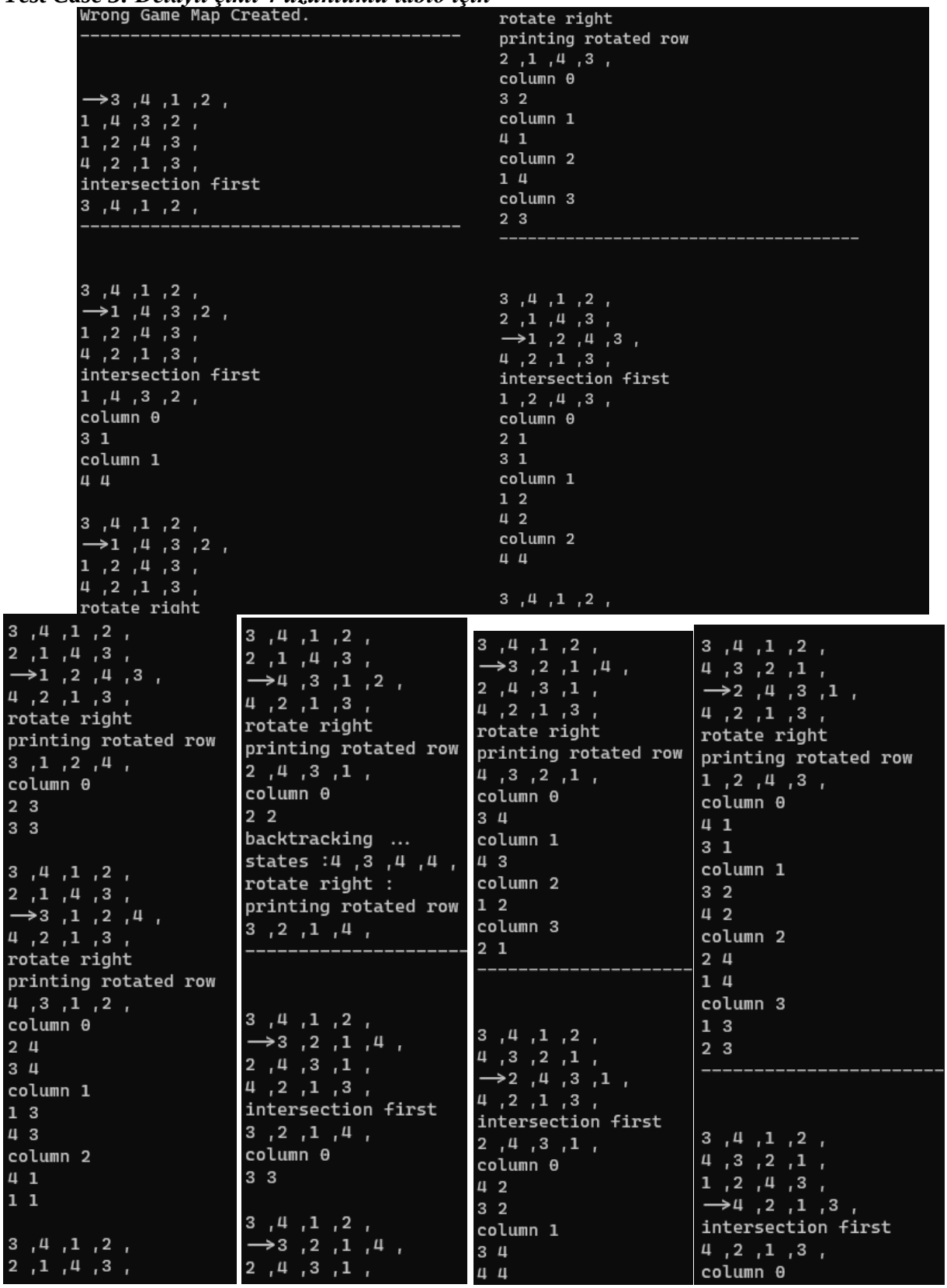
Test Case 2 : Detaylı çıktı 3 uzunluklu tablo için

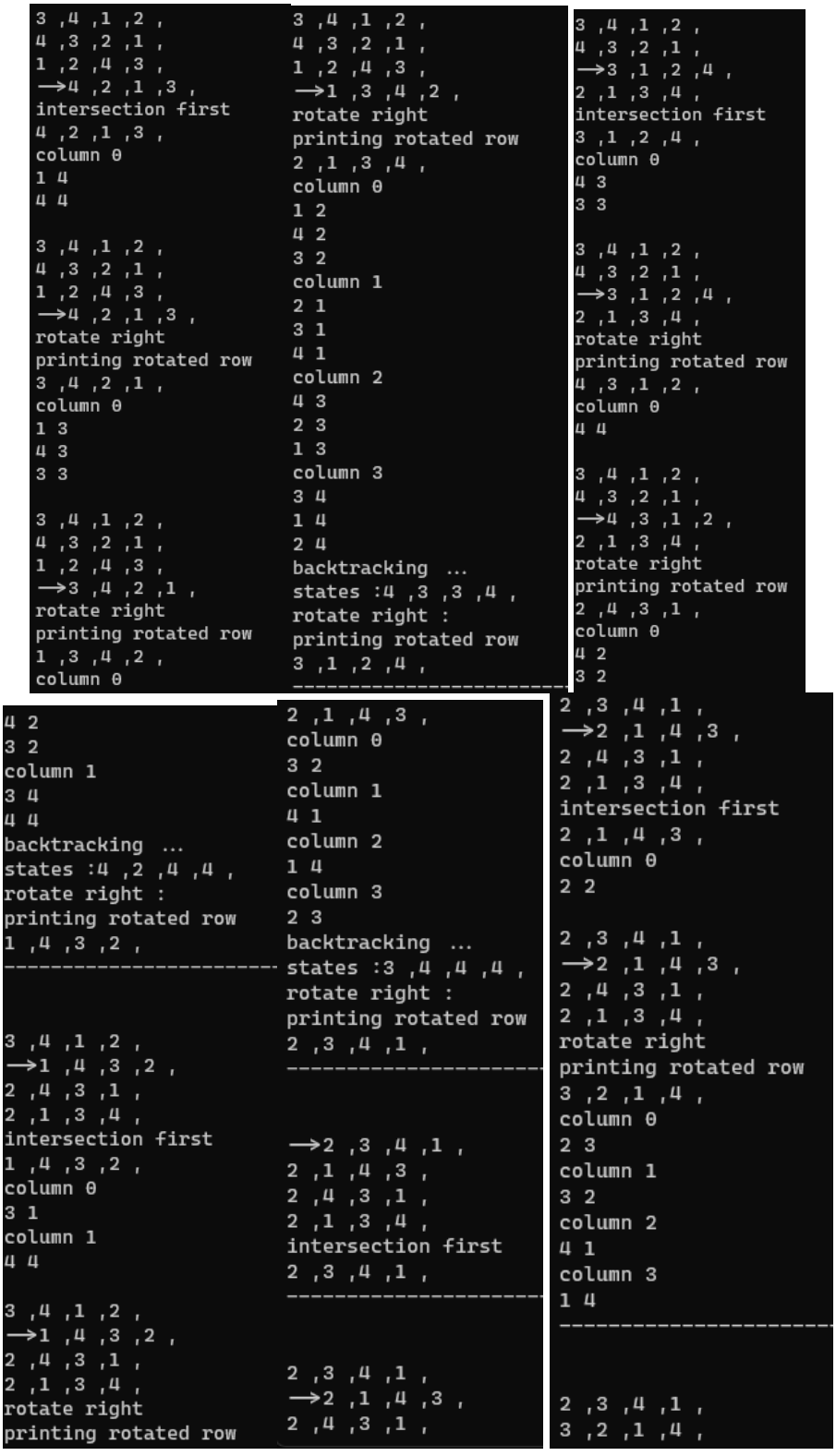
```
What is the mode you using?(normal/n -- detail/d)d
What is size of game table which you want create?(1<N<10)3
-----------------------------------


-->1 ,3 ,2 ,
1 ,2 ,3 ,
2 ,1 ,3 ,
intersection first
1 ,3 ,2 ,
-----------------------------------


1 ,3 ,2 ,
-->1 ,2 ,3 ,
2 ,1 ,3 ,
intersection first
1 ,2 ,3 ,
column 0
1 1

1 ,3 ,2 ,
-->1 ,2 ,3 ,
2 ,1 ,3 ,
rotate right
printing rotated row
3 ,1 ,2 ,
column 0
1 3
column 1
```

```
column 1
3 1
column 2
2 2

1 ,3 ,2 ,
-->3 ,1 ,2 ,
2 ,1 ,3 ,
rotate right
printing rotated row
2 ,3 ,1 ,
column 0
1 2
column 1
3 3
backtracking ...
states :2 ,3 ,3 ,
rotate right :
printing rotated row
2 ,1 ,3 ,
--------------------------------------


-->2 ,1 ,3 ,
2 ,3 ,1 ,
2 ,1 ,3 ,
intersection first
2 ,1 ,3 ,
--------------------------------------
```

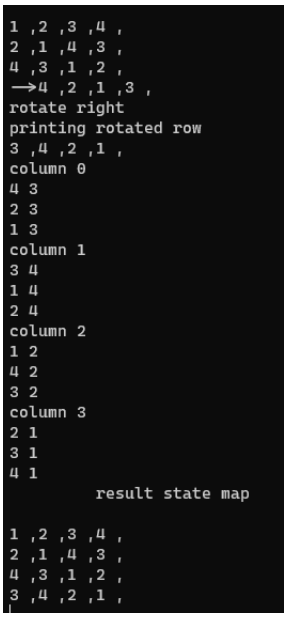
```
2 ,1 ,3 ,
-->2 ,3 ,1 ,
2 ,1 ,3 ,
intersection first
2 ,3 ,1 ,
column 0
2 2

2 ,1 ,3 ,
-->2 ,3 ,1 ,
2 ,1 ,3 ,
rotate right
printing rotated row
1 ,2 ,3 ,
column 0
2 1
column 1
1 2
column 2
3 3

2 ,1 ,3 ,
-->1 ,2 ,3 ,
2 ,1 ,3 ,
rotate right
printing rotated row
3 ,1 ,2 ,
column 0
2 3
column 1
1 1
backtracking ...
Wrong Game Map Created.
--------------------------------------
```

Test Case 3: Detaylı çıktı 4 uzunluklu tablo için





Şeklinde ilerlemektedir sonunda aşağıdaki çıktı oluşmaktadır.



burada son durumda tüm çıktılar için keşişim bulunmadığı görülmektedir.

4. Sonuç

4.1 Yer ve Zaman Karmaşıklığı

Problemin yer karmaşıklığı tablo üzerinde işlemler yapıldığı için O(N^2) olacaktır. Eğer bu matrisi girdi olarak saymazsak durumları tutan tablonun boyutuna bağlı oalcaktır bu da satır sayısıdır bu durumda O(N) denilebilir.

Problemin zaman karmaşıklığı tüm durumlar denenirse O(N^n) olacaktır fakat tablo üzerinde geri dönüşlerle yapılan elemeler sayesinde tüm uzay taranmaz ve budamalar yapılmış olur. Bu sayede en iyi durumda O(N) ile keşif yapılabilir bu durumda durumları tutan matriste geri dönme olmadan direk keşişimsiz olması durumudur. **Eğer kesişimler varsa kendinden sonraki sütundaki değerler taranır ve keşişmeyen bulunursa üste geçilir ve üstekinde çevirme işlemi yapılarak önceki durumdaki iki aşağı satırdaki oluşan tüm durumlar elenmiş olmaktadır**. *Bu yüzden duruma O(N^2 * C) diyebiliriz. N^2 değeri sütun tarama ve satırda ilerlemeden gelmektedir. C katsayısı ise her bir durumda ne kadar sütun ilerleyeceği ile ilgili bir katsayıdır Bu sayı N den N^n e kadar tüm değerler olabilir. Bu durumda O(N^3) ve O(N^n) arasında denilebilir.*