Kocaeli Üniversitesi

Bilgisayar Mühendisliği Bölümü

Yazılım Laboratuvarı I

Multithread Kullanarak Samurai Sudoku Çözme

Oğuz Narlı
180201074@kocaeli.edu.tr

Alperen Tan
190201054@kocaeli.edu.tr

Giriş

Yazılım Laboratuvarı dersinin 2.Projesi kapsamından bizden verilen Samurai Sudokunun Multithreading yapısı kullanılarak gerçekleştirilmesi istenmektedir. Bununla beraber sudokunun çözümü esnasında threadlerin aynı hücreyi çözdürme ve hesaplama gerçekleştirirken olabilecek olası çakışmaları önlenmesi için senkronizasyon yapılması da gerekmektedir. Proje isterlerine baktığımızda 5 eş zamanlı thread ve 10 eş zamanlı çalışan threadin uygulanması ve bunların çözüm sürelerinin grafiğe dökülmesi istenmektedir. Bu testin amacı eş zamanlı çözüm sayısını arttırdığımızda çözüm süresindeki değişimin incelenmesidir. Projemiz C# programla dili .NET 5.0 sürümü ile gerçekleştirilmiştir. Arayüz için WindowsForm yapısı kullanılmıştır.

Özet:

Uygulamayı kullanan kullanıcı txt belgesine girdiği sudoku değerleriyle multithreading yöntemini kullanarak 5 thread ya da 10 thread ile sudokunun çözümüne ulaşabilir, her iki yöntem arasındaki farkı grafiksel olarak inceleyebilir, bilgisayarın gerçekleştirdiği işlem adımlarını görüntüleyebilir.

Veri Tabanı:

Projenin işlem adımlarını localhost'ta depoladığımız MYSQL veri tabanına connection stringleri ile bağlantısını sağlayarak kaydettik.

Sudokunun Çözdürülmesi:

Samuray Sudokuyu çözdürmek için literatür taraması gerçekleştirdiğimizde Backtracking[1] ve Crook algoritması[2] ön plana çıkmaktadır. Biz projemizde klasik sudoku çözüm tekniklerine ek olarak Backtracking kullanmayı tercih ettik. Bunun nedeni Crook algoritmasını Backtracking' e daha karmaşık bulmamızdır. Ancak bu sefer Backtracking algoritması özyinelemeli yapıya sahip olduğundan senkronizasyon sorunu ve sudokuların kendi içerisinde birden fazla çözüm tekniğine sahip olması sorunu ile karşılaşacaktık. Halbuki Samuray Sudoku tek çözüme sahiptir. Bu yüzden ilk olarak ortalama sudoku oynayan kişilerin çözüm tekniklerini algoritmaya döküp gerçekleştirdikten sonra çözüm tıkandığında deneme ve sınamanın gerçekleşmesi için Backtracking kullandık.

Backtracking Algoritması peki nedir? Bilgisayar bilimlerinde bir değerin aranması veya bir hedefe ulaşmak için kullanılan özyinelemeli bir algoritmadır. Burada bir amaç bulunmalı ve amaca ulaşan çeşitli yollar arasından bir doğru seçim aranıyor olmalıdır. Tanıma tekrar bakmak gerekirse istediğimizi gerçekleştirmek için uygun bir algoritma gibi duruyor ancak sanıldığı gibi değil. Nedeni ise Samuray Sudoku'nun alt sudokularını kendi içinde çözdürdüğümüzde birden fazla sonuç verme ihtimali olmasıdır ayrıca thread olarak çözdürmek istediğimizde ortak alan bulunamazsa çözüm işe yaramamaktadır. Peki biz bu algoritmayı nasıl kullanılabilir kıldık. Bunun için önce klasik sudoku teknikleri ile sudokunun belli değerlerini bulduktan sonra sudokunun bulunan değerlerinin yol göstericiliği ile çözüme gitmektedir. Bu gidiş yolunu nasıl kullanmayı karar verdik diye sorulacak olursa ilk olarak sadece backtracking ile çözüme gidilmeye çalışılmış ancak çözülememiştir. Ardından kendi yöntemimizi kullandığımızda Samuray Sudokunun büyük çoğunluğunun

çözüldüğü fakat hala eksikler olduğunu fark ettik. Biz de bu eksikleri Backtracking ile kapanabileceğini düşündük.

Klasik sudoku çözüm yöntemleri nasıl projeye entegre edildi? Bunu projemize uygulamak istediğimizde ilk olarak sudoku çözümüyle ilgili teknikleri inceledik. Bunun için gözümüze en çok çarpan yöntem sıkıştırma yöntemidir[4]. Sıkıştırma yöntemi bir hücrenin alabileceği değerleri 3x3'lük hücre içerisinde diğer hücrelerde alıp alamayacağını kontrol etmektedir. Eğer alabileceği tek yer orasıysa hücre içerisine olabilecek o değer yerleştirilir. Daha sonrasında 3x3'lük hücre dışında satır ve sütun boyunca da kontrol edip bu yöntemin kullanılabileceğini keşfettik. Bunu nasıl keşfettik diyecek olursak [5] numaralı bağlantıda verilen web sitesinde örnek sudokuyu adım adım çözdürdüğümüzde sıkıştırma yöntemini satır ve sütun içerisinde de yaptığını fark ettik. Bu yöntemi kullandıkça belli süre sonra olası değerlerinde tek sayıya düştüğü fark ettik. Ancak bunu algoritmaya döktüğümde büyük çoğunluğunu çözdüğümüzü ancak eksiklikler olduğunu fark ettik. Bunu muhtemelen mevcut sitede verilen kuralları uygularken gözden kaçırdığımız bir yöntem olduğunu düşünmekteyiz. Bunu da kapamak için Backtracking algoritmasını kullanmayı uygun gördük.

Algoritmamız inşa etme aşamasına geldiğimizde ise bulduğumuz yöntemi bulmak için önce tüm hücrelerin alabileceği olası sayıları bulunur. Ardından boş olan her hücre teker teker dolaşarak olası sonuçların diğer hücrelerin olası sonuçlarında içerip içermediği incelenir. Bu işlem çözüm tekrar aynı sonucu verene kadar devam eder. Eğer çözüm tekrara girip boş alanlar mevcutsa Backtrackinge sokularak çözüm tamamlanır.

Algoritma inşa edildikten sonra yöntem alt sudokulara bölünerek algoritma her thread içerisinde bütün alt sudokular içerisinde gerçekleştirilmektedir. Ortak alanlarının çözümü 5 li thread için orta alanın çözümünü yapan threade ,10'lu için ise orta alanın sağ tarafını ve sol tarafını gerçekleştiren 2 threade verilmiştir. Threadler 5. ve 9. arasında satır veya sütunda ve 11.-15. satır veya sütundayken lock işlemi ile senkronize edilmektedir. Bu alanlardayken senkronizasyon yapılmasının sebebi ise her boş hücre için olası sonuçlar her çözümde tekrarlandığından çözümün devamında olası değerlerin önceki değerler üzerinden yapılması ve çözümün çökmesinin önlenmesidir.

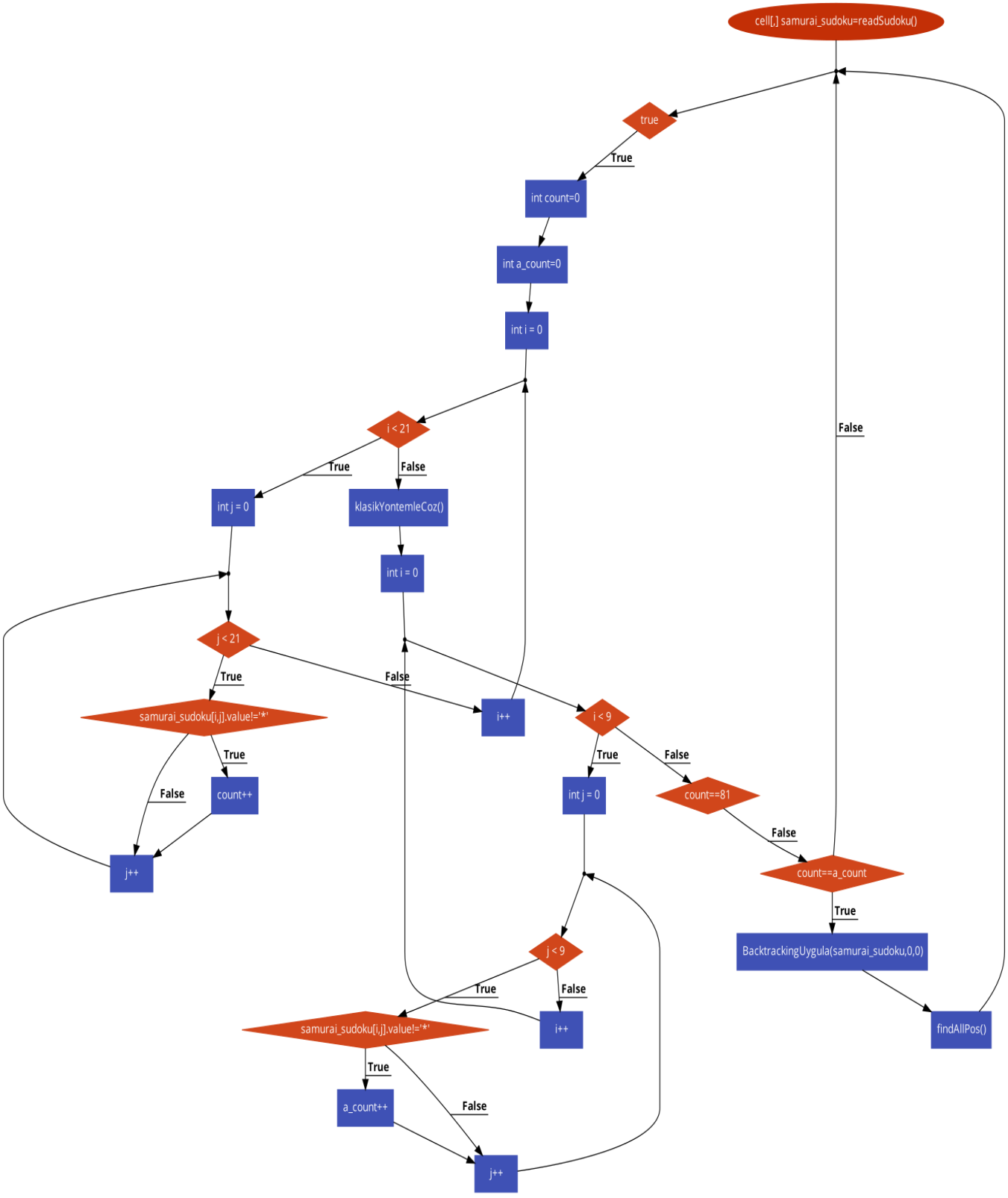
Proje Yazılım Mimarisi:

Projemizi inşa ederken ilk olarak tüm hücrelerin değerleri ve alabileceği değerleri içerdiği sınıf oluşturulmuş ve bu sınıfın türünden 21 e 21'lik matris tanımlanmıştır. Ardından tüm sudoku txt dosyasından okunduktan sonra okunan değerler matrise atanmıştır. Ardından her bir hücrenin alabileceği değerleri bulan fonksiyon oluşturulmuş olabilecek her değer ise listede tutulmuştur. Sonra hücrelerin çözümünü sağlayan fonksiyonlar oluşturulmuştur. Bu fonksiyonlarda tüm hücrelerde dolaşıp ilgili hücrenin bulunduğu 3*3'lük hücre, satır ve sütun içerisinde alabileceği değerlerin sadece kendisinde yerleşip yerleşemediğini kontrol etmektedir. Ardından backtracking uygulayarak sonucu vermektedir.

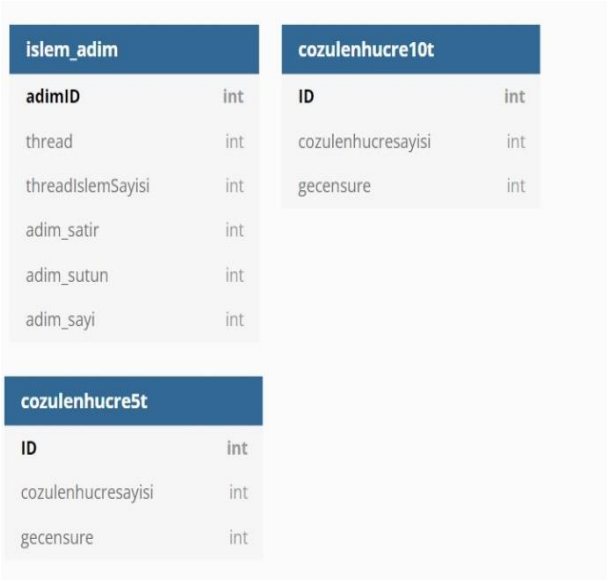
Sonuç:

Bu projede C# programlama diliyle sudoku çözüm algoritmasını ve multithreading yapısını, problemlerini ve senkronizasyonunu kodlayarak öğrendik.

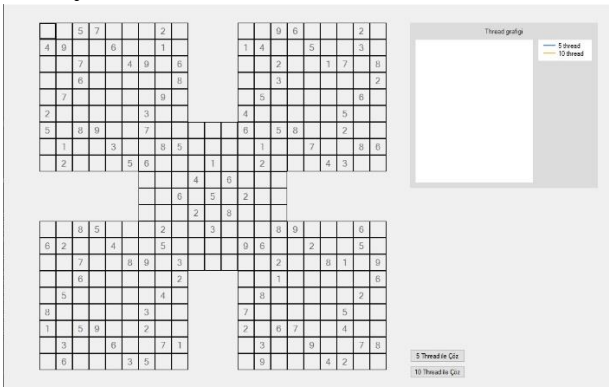
Proje Akış Şeması:



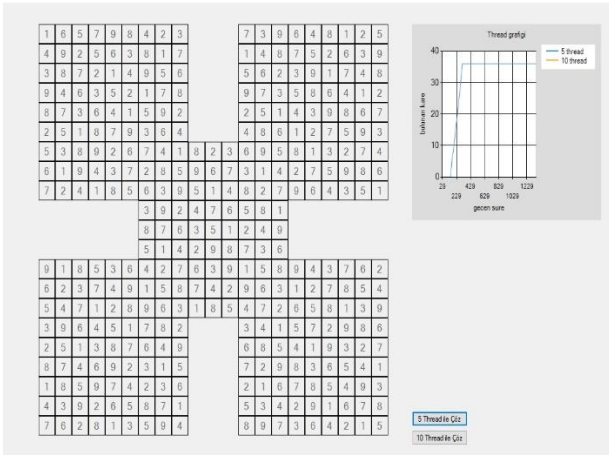
Veri Tabanı Diyagramı:



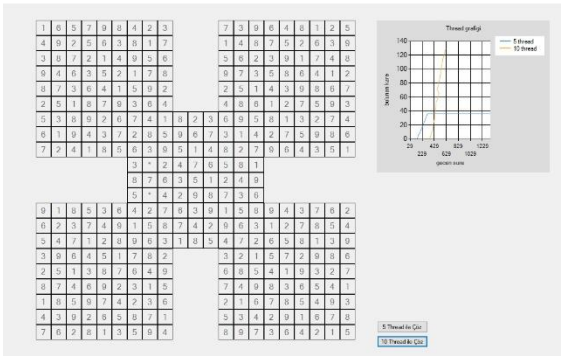
Deneysel Sonuçlar:



Açılış ekranı (txt'den veri çekildi)



5 thread çözümü



10 thread çözümü

adimID	thread	threadIslemSayisi	adim_satir	adim_sutun	adim_sayi
1	1	1	6	8	1
2	1	2	6	13	9
3	1	3	7	14	4
4	1	4	8	8	9
5	1	5	8	12	8
6	1	6	11	14	6
7	1	7	12	12	1
8	1	8	13	14	3
9	1	9	14	7	6
10	1	10	6	13	9
11	2	1	1	2	2
12	1	11	11	14	6
13	1	12	12	9	6
14	3	1	1	18	6
15	1	13	6	8	1
16	3	2	2	13	6
17	1	14	6	13	9
18	2	2	1	8	7
19	3	3	4	12	2
20	1	15	7	10	6
21	2	3	5	7	6
22	3	4	5	14	6
23	1	16	11	14	6
24	3	5	8	16	6
25	1	17	12	9	6

İşlem adımlarının veri tabanında gösterimi

ID	cozulenhucresayisi	gecensure
1	0	84
2	0	160
3	0	179
4	0	190
5	0	195
6	0	219
7	0	234
8	0	267
9	0	301
10	0	293
11	0	296
12	0	311
13	0	314
14	0	314
15	3	336
16	5	343
17	10	357
18	13	373
19	23	389
20	25	419
21	34	447
22	36	440
23	40	449
24	44	466
25	62	479

Çözülen kare-geçen süre verileri

Kaynakça:

[1]https://www.geeksforgeeks.org/sudoku-backtracking-7/

[2]https://towardsdatascience.com/solve-sudoku-more-elegantly-with-crooks-algorithm-in-python-5f819d371813

[3]https://bilgisayarkavramlari.com/2009/11/01/geri-izleme-algoritmasi-backtracking-algorithm/

[4] https://playwithcsharpdotnet.blogspot.com/2020/07/develop-sudoku-game-using-basic-csharp-codes.html