

Kocaeli Üniversitesi
Bilgisayar Mühendisliği Bölümü
Yazılım Laboratuvarı I
MULTITHREAD SAMURAI SUDOKU

Taha Uz/Mahmut Alper Yılmaz

190202013@kocaeli.edu.tr[Taha]/180202004@kocaeli.edu.tr[Alper]

ÖZET

Yazılım Laboratuvarı I Projesi olarak bizden “Multithread Kullanarak Samurai Sudoku Çözme” adlı uygulama geliştirmemiz beklenmektedir.

Proje için Python dilini kullanmayı tercih ettik. PyCharm geliştirme ortamını kullandık.

Proje dökümanında bizden, multithread yapısını kullanarak verilecek samurai sudoku üzerinden çözüm bulmamız istendi. Sudoku başlangıç değerleri .txt uzantılı dosyadan verilir ve bu dosyadan samurai sudoku tek tek gridlere yerleştirilir. Daha sonra çözüm fonksiyonuna yollanarak sudoku çözülür ve çözüldükten sonra 5 thread ve 10 thread kullanılarak ulaşılan çözümler için “matplotlib” kütüphanesinden yararlanarak grafik çizimi gerçekleştirilir ve ikisi arasında karşılaştırma yapılır. En sonunda çözülen Samurai Sudoku ‘nun çözümü kullanıcıya “pygame” ile oluşturulmuş arayüz ile gösterilir.

1.GİRİŞ

Proje için Python dilini kullanmayı tercih ettik. Pycharm geliştirme ortamını kullandık.

Python programlama dili , nesne yönelimli , yorumlamalı ve birimsel ve etkileşimli yüksek seviyeli bir programlama dilidir. Girintilere dayalı söz dizimi, dilin öğrenilmesini ve akılda kalmasını kolaylaştırır.

PyCharm, çapraz platform bir Python geliştirme ortamıdır. Kod analizleri, grafiksel hata ayıklamacısı, versiyon kontrol sistemi ile entegre ve Django ile Python web geliştirme yapılması sağlanmaktadır. Çapraz platformu Windows, OS X ve GNU/Linux işletim sistemleri üzerinde çalışır.


2.TEMEL BİLGİLER

Projemizin uygulamasını yaparken alt başlıklara ayırdık. Bu başlıklar ; Txt Samurai Sudoku Okuma, Samurai Sudoku Çözme, Threadleri Grafik ile Karşılaştırma, Samurai Sudoku

Çözümünü Arayüzde Gösterme olmak üzere sizlere 4 alt başlık altında anlatacağız.

2.1. TXT SAMURAI SUDOKU OKUMA

Bu kısımda “.txt” uzantılı dosyamızda olan aşağıdaki resimdeki format olan sudokumuzu alıyoruz.

 SamuraiSudoku.txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

```
*5*****9**9*****8*
91**7**3426**7**35
***8*3*****8*5***
**1**4*****9***1**
*7*1**6**3**5**7*
**2***7****5***2**
***9*6*****7*3***
78**3**16***35**2**41
*6*****4*6*9*8*****9*
**49*75**
*****5*****
**83*17**
*9*****8*7*3*6*****3*
25**1**37***89**7**61
***2*5*****1*5***
**4***8***2***1**
*8*6**7**1*9**7*
**9***5*****6***4**
***1*7*****2*9***
97**8**5158**3**26
*2*****9*2*****4*
```

Figure 1 : SamuraiSudoku.txt İçeriği

Bu format Samurai Sudoku ‘ nun yukarıdan aşağıya satır satır verilmiştir. Burada ilk satırın 9 elemanı 1. Sudoku, sonraki 9 elemanı ise 2. Sudokuya aittir.

```
# Board 1
file = open("SamuraiSudoku.txt")

for j in range(9):
    SamuraiSudoku = file.readline()
    # print(SamuraiSudoku)

    temp = []
    temp2 = []
    for i in range(9):
        # print(i)
        if SamuraiSudoku[i] == "*":
            temp.append(0)
            temp2.append(0)
        elif SamuraiSudoku[i] != "\n":
            temp.append(int(SamuraiSudoku[i]))
            temp2.append(int(SamuraiSudoku[i]))
    board1.append(temp)
    board0.append(temp2)

file.close()
# print(board0)
# print(board1)

# Board 2
file = open("SamuraiSudoku.txt")

for j in range(9):
    SamuraiSudoku = file.readline()
    # print(SamuraiSudoku)
    a = 9
    b = 18
    if j > 5:
        a = 12
        b = 21
    temp = []
    temp2 = []
    for i in range(a, b, 1):
        # print(i)
        if SamuraiSudoku[i] == "*":
            temp.append(0)
            temp2.append(0)
        elif SamuraiSudoku[i] != "\n":
            temp.append(int(SamuraiSudoku[i]))
            temp2.append(int(SamuraiSudoku[i]))
    board5.append(temp)
    board7.append(temp2)
    board10.append(temp2)

file.close()
# print(board2)
```

Alttaki satırlarda bazı satırların 21 karakteri olduğunu görüyorsunuz. Onların içinde ise ortadaki sudokumuzun elemanı ile diğer 1. Ve 2. Sudokunun elemanlarının kesişimi var bunları özel bir biçimde ayırıyor ve hepsine atıyoruz. Sadece 9 karakterin olduğu satır ise ortadaki sudoku yani

```
# Board 3
file = open("SamuraiSudoku.txt")

for j in range(21):
    SamuraiSudoku = file.readline()
    # print(SamuraiSudoku)

    if j > 11:
        temp = []
        temp2 = []
        for i in range(9):
            # print(i)
            if SamuraiSudoku[i] == "*":
                temp.append(0)
                temp2.append(0)
            elif SamuraiSudoku[i] != "\n":
                temp.append(int(SamuraiSudoku[i]))
                temp2.append(int(SamuraiSudoku[i]))
        board3.append(temp)
        board8.append(temp2)

file.close()
# print(board3)

# Board 4
file = open("SamuraiSudoku.txt")

for j in range(21):
    SamuraiSudoku = file.readline()
    # print(SamuraiSudoku)

    if j > 11:
        a = 12
        b = 21
        if j > 14:
            a = 9
            b = 18
        temp = []
        temp2 = []
        for i in range(a, b, 1):
            # print(i)
            if SamuraiSudoku[i] == "*":
                temp.append(0)
                temp2.append(0)
            elif SamuraiSudoku[i] != "\n":
                temp.append(int(SamuraiSudoku[i]))
                temp2.append(int(SamuraiSudoku[i]))
        board4.append(temp)
        board9.append(temp2)

file.close()
```

5. Sudoku nun ortadaki elemanlarıdır. Aynı adımları alttaki 3. Ve 4. Sudokulara da uygulayarak bu işlemi tamamlıyoruz.

```
# Board 5
file = open("SamuraiSudoku.txt")

for j in range(21):
    SamuraiSudoku = file.readline()
    # print(SamuraiSudoku)

    if j > 5 and j < 15:
        a = 0
        b = 9
        if j < 9 or j > 11:
            a = 0
            b = 15
        temp = []
        temp2 = []
        for i in range(a, b, 1):
            # print(i)
            if SamuraiSudoku[i] == "*":
                temp.append(0)
                temp2.append(0)
            elif SamuraiSudoku[i] != "\n":
                temp.append(int(SamuraiSudoku[i]))
                temp2.append(int(SamuraiSudoku[i]))
        board5.append(temp)
        board10.append(temp2)

file.close()
```

Okuduğumuz karakterlerde ‘*’ ve ‘\n’ karakterlerini bulup özel işlem uyguluyoruz. Eğer ‘*’ ise bunu ‘0’ değerine eşitliyoruz. Diğerinde ise satırın bittiğini ve birdahaki satıra geçmeden önce o satırı bir diziye atmamız için uyarı niteliğinde kullanıyoruz. Böylelikle tüm sudokuyu 5 farklı 9 lu sudokuya bölüyoruz ve dizilere atıyoruz.

2.2. SAMURAI SUDOKU ÇÖZME

Sudokuyu çözerken örneğin ; 9'lu sudokunun 0'a 0 noktasını ele alalım. Bu noktanın x ve y eksenlerindeki sayılara bakıyoruz. Eğer o noktaya atayacağımız sayı ile eşit bir sayı yok ise ikinci kontrol evresine geçiş yapıyoruz.

```
for x in range(9):
    if grid[row][x] == num:
        return False

for x in range(9):
    if grid[x][col] == num:
        return False
```

İkinci evrede bulunduğumuz 0' a 0 noktamızın bulunduğu '3x3' lük kare matrisinin içinde bu noktaya koyacağımız sayı var mı yok mu onu kontrolüne bakıyoruz.

```
for i in range(3):
    for j in range(3):
        if grid[i + startRow][j + startCol] == num:
            return False
```

Eğer bu kontrolden de geçiyor ise o noktaya sayımızı koyuyoruz. Tabi diğer sudokular ile kesişen noktalara ayrı kural uyguluyoruz.

Bu kurallar şu şekilde ; örneğin 1. Sudokuyu ele alalım. Buraya koyacağımız sayı ortadaki yani 5. Sudokuyu ve sağ üstteki yani 2. Sudokuyu da etkilemekte. Bunun kontrolü şu şekilde sağlıyoruz. 6'ya 6 noktasını ele alacak olalım. Bu noktaya değer atarken yukarıdaki adımları tamamladıktan sonra bir de 5. Sudokunun x ve y eksenini kontrol ediyor.

```
if row > 5 and col > 5:
    for x in range(9):
        if grid5[row - 6][x] == num:
            return False

    for x in range(9):
        if grid5[x][col - 6] == num:
            return False
```

Bunun yapmasının sebebi diğer sudokuların ortadaki sudoku ile kesişim noktalarına geldiklerinde çözdükleri her kareyi ortadaki sudokuya da otomatik olarak kayıt etmesinden dolayıdır. Bu kontrol sağlandıktan sonra 5. Sudokunun bulunduğu '3x3' lük matrisini tekrardan kontrol ediyor.

```
for i in range(0, 3, 1):
    for j in range(0, 3, 1):
        if grid5[i][j] == num:
            return False
```

Tüm bu işlemlerden geçtikten sonra sayı hepsinden doğru şekilde geçtiyse bu sayı o noktaya atanıyor ve bir sonraki noktaya geçiş yapılıyor.

Bu işlemler ile 5 sudokuyu da birbiri ile eş zamanlı olarak haberleştirip çözerek bu Samurai Sudokunun çözümünü elde etmiş oluyoruz.

2.3. THREADLERİ GRAFİK İLE KARŞILAŞTIRMA

Samurai Sudoku'yu bizden 5 thread ve 10 thread kullanarak çözmemiz istenmekteydi. Bu kullandığımız 2 çeşit yöntemi birbiri ile karşılaştırarak performanslarını grafik üzerinden kıyaslamamız istenmekteydi. Bunun için bizde iki yöntem kendini çözerken "time" kütüphanesinin yardımıyla bir kutu içerisindeki sayıyı bulmasını zamana karşı olan değerini bulduk.

```
if threadKind == 5:
    end_5 = time.time()
    if end_5 - start_5 != 0.0:
        foundTime_5_thread.append((end_5 - start_5))

if threadKind == 10:
    end_10 = time.time()
    if end_10 - start_10 != 0.0:
        foundTime_10_thread.append((end_10 - start_10))
```

Yukarıdaki kod parçaçığında da anlaşılacağı gibi burada geçen süreyi bir dizi parçasına atamasını yapıyoruz.

Daha sonra ise bu dizi içerisindeki aynı süreler bize o zamanda kaç tane kare bulunduğunun bilgisini veriyor. Bu veriyi çıkarmak için diziyi döngü içerisine tutuyoruz ve içerisindeki tekrarlanan elemanların sayılarını elde ederek o süre içerisinde kaç tane kare bulunduğunu öğreniyoruz.

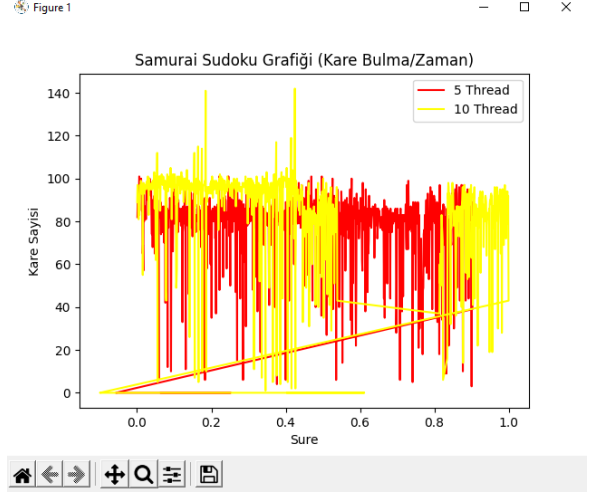
```
bul_kare_sayi = []
bul_sure = []
temp_i = 0.0
for i in foundTime_5_thread:
    if i != temp_i:
        if 3 > i > 2:
            bul_kare_sayi.append(foundTime_5_thread.count(i - 2))
            bul_sure.append(i - 2.1)
        elif 2 > i > 1:
            bul_kare_sayi.append(foundTime_5_thread.count(i - 1))
            bul_sure.append(i - 1.1)
        else:
            bul_kare_sayi.append(foundTime_5_thread.count(i))
            bul_sure.append(i)
        temp_i = i
```

Bunun aynı işlemini 10 Thread olana da uyguluyoruz.

```
bul_kare_sayi2 = []
bul_sure2 = []
temp_i2 = 0.0
for i in foundTime_10_thread:
    if i != temp_i2:
        if 4 > i > 3:
            bul_kare_sayi2.append(foundTime_10_thread.count(i - 3.1))
            bul_sure2.append(i - 3.1)
        elif 3 > i > 2:
            bul_kare_sayi2.append(foundTime_10_thread.count(i - 2.1))
            bul_sure2.append(i - 2.1)
        elif 2 > i > 1:
            bul_kare_sayi2.append(foundTime_10_thread.count(i - 1.1))
            bul_sure2.append(i - 1.1)
        else:
            bul_kare_sayi2.append(foundTime_10_thread.count(i))
            bul_sure2.append(i)
        temp_i2 = i
```

Bu işlemlerden sonra “matplotlib” kütüphanesini çağırarak “plot()” fonksiyonunun içerisine x(süre) ve y(bulunan kare sayısı) noktalarını yollayarak grafiğimizi çizdiriyoruz.

```
# Grafik
plt.plot(bul_sure, bul_kare_sayi, color="red", label="5 Thread")
plt.plot(bul_sure2, bul_kare_sayi2, color="yellow", label="10 Thread")
plt.title("Samurai Sudoku Grafiği (Kare Bulma/Zaman)")
plt.xlabel("Sure")
plt.ylabel("Kare Sayisi")
plt.legend()
plt.show()
```



2.4. SAMURAI SUDOKU ÇÖZÜMÜNÜ ARAYÜZDE GÖSTERME

Samurai Sudoku’nun çözümünü kullanıcıya göstermek için “pygame” kütüphanesinin içerisindeki arayüz sistemini kullandık. İlk olarak arayüz kısmının gridlerini Samurai Sudoku’ ya uygul olacak biçimde çizdirdik. Daha sonra bu kutuların içerisine çözümümüzü yerleştirdik.

```
for x in range(0, WINDOW_WIDTH, blockSize):
    k_1 = 3
    for y in range(0, WINDOW_HEIGHT, blockSize):
        rect = pygame.Rect(x, y, blockSize, blockSize)

        if 232 > x > 188 and 108 > y > -22:
            pygame.draw.rect(SCREEN, WHITE, rect, 1)
        elif 441 - 189 > x > 441 - 232 - 21 and 441 + 22 > y > 441 - 108 - 21:
            pygame.draw.rect(SCREEN, WHITE, rect, 1)
        elif 441 + 22 > x > 441 - 108 - 21 and 441 - 189 > y > 441 - 232 - 21:
            pygame.draw.rect(SCREEN, WHITE, rect, 1)
        elif 108 > x > -22 and 232 > y > 188:
            pygame.draw.rect(SCREEN, WHITE, rect, 1)
        else:
```

Çizimi gerçekleştirirken aynı zamanda kutuların içerisine sayıları “txt” içerisinden çekerken uyguladığımız matematiksel işlemlerin bir benzerini arayüzdeki kutuların içerisine yerleştirirken de uyguladık.

```
else:
    pygame.draw.rect(SCREEN, BLACK, rect, 1)
    myfont = pygame.font.SysFont('Arial', 15)
    if y < 21 * 9 and x < 21 * 9:
        textsurface = myfont.render(str(board[1][j]), False, (0, 0, 0))
        j += 1
        j %= 9
        SCREEN.blit(textsurface, (y + 5, x))
    elif y > 21 * 11 and x < 21 * 9:
        textsurface = myfont.render(str(board[2][j]), False, (0, 0, 0))
        j += 1
        j %= 9
        SCREEN.blit(textsurface, (y + 5, x))
    elif y > 21 * 11 and x > 21 * 11:
        textsurface = myfont.render(str(board[4][1 - 3][j]), False, (0, 0, 0))
        j += 1
        j %= 9
        SCREEN.blit(textsurface, (y + 5, x))
    elif x < 21 * 9 and x > 21 * 11:
        textsurface = myfont.render(str(board[3][1 - 3][j]), False, (0, 0, 0))
        j += 1
        j %= 9
        SCREEN.blit(textsurface, (y + 5, x))
    elif 21 * 15 > y > 21 * 5 and 21 * 15 > x > 21 * 5:
```

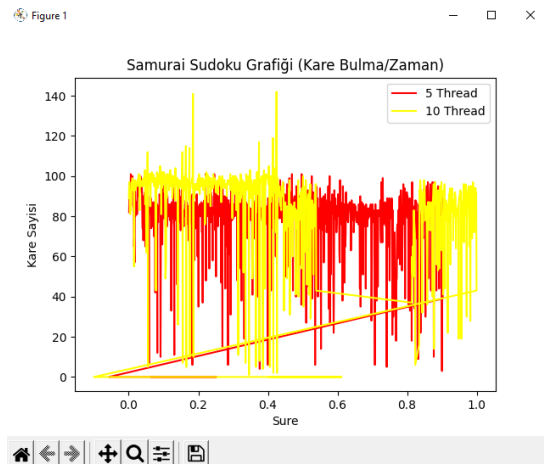
Bunları çizdirirken ortadaki sudokuya ise diğer sudokuların kesişim noktaları dışındaki noktaları çizdirmemiz gerekmekte. Bunun için ise sadece kesişmeyen noktaları koşul içerisinde kontrolünü sağlayarak arayüzde çizimini sağladık.

```
elif 21 * 15 > y > 11 * 5 and 21 * 15 > x > 21 * 5:

    if 21 * 12 > y > 21 * 8:
        textsurface = myfont.render(str(board5[k][k_1]), False, (0, 0, 0))
        SCREEN.blit(textsurface, (y + 5, x))
        k_1 += 1
    if (k_1 == 6):
        k += 1
    if 21 * 12 > x > 21 * 8:
        textsurface = myfont.render(str(board5[t][t_1]), False, (0, 0, 0))
        SCREEN.blit(textsurface, (y + 5, x))
        t_1 += 1
    if (t_1 == 9):
        t += 1
        t_1 = 0
```

Bu işlemlerle birlikte kullanıcının çözümünü istediği Samurai Sudoku 'nun çözümünü kullanıcıya göstermiş olduk.

3.DENEYSEL SONUÇLAR

[illegible]

4.SONUC

Bu proje multi thread sisteminin nasıl çalıştığını ve Samurai Sudoku çözüm algoritmasını yani bir algoritmayı nasıl multi thread yöntemine uyarlayıp çözüm elde edileceğini öğretti.

5.REFERANSLAR

<https://stackoverflow.com/questions/33963361/how-to-make-a-grid-in-pygame>

<https://www.geeksforgeeks.org/sudoku-backtracking-7/>

[Samurai Sudoku \(samurai-sudoku.com\)](http://samurai-sudoku.com)

<https://stackoverflow.com/questions/16992038/inline-labels-in-matplotlib>

6.AKIŞ DİAGRAMI

