BIL551-STS614: Yapay Zekâ

Nurikabe

Dönem Projesi

Hazırlayan : Esra ALBAYRAK ÖZDEŞ

Table of Contents

Table of Figures

Bulmacanın Tanımı, Kuralları ve Kısıtları

Nurikabe Bulmacasının Tanımı

Nurikabe, dikdörtgen şeklindeki ızgaralı bir tahtada oynanan bir mantık bulmacasıdır. Başlangıç durumunda ızgara üzerinde rakamlar bulunur, rakam içermeyen hücrelerin başlangıç rengi verilmez. Amaç, belirli kurallara uygun şekilde **adalar (beyaz hücreler)** ve **deniz (siyah hücreler)** oluşturarak verilen ipuçlarını çözmektir. Farklı boyutlarda oynamak mümkündür. Bulmaca tahtasında rakam içeren hücreler, adaların başlangıç noktalarını ve boyutlarını belirler. Rakam olmayan hücreler için iki durum vardır; ada veya deniz olabilirler. [Ref 1, Ref 2]

Oyun Tahtası Özellikleri

- **Tahta Büyüklüğü:** 4x4 (4 satır, 4 sütun) ve 5x5 (5 satır, 5 sütun) olarak kare ölçü üzerinden çözüm geliştirmeye karar verilmiştir.
- **Başlangıç Durumu:** Tahtada bazı hücrelerde rakamlar yer alır. Rakamlar bir adanın boyutunu ve başlangıç konumunu ifade eder. Diğer hücreler başlangıçta belirsizdir(null).

Kurallar ve Kısıtlar

1. **Adalar (Beyaz Hücreler):**
    - Her adada tam olarak bir numaralandırılmış hücre bulunmalıdır.
    - Numaralandırılmış her hücre bir ada hücresidir, içindeki sayı o adadaki hücre sayısıdır. Örneğin, "3" rakamı içeren bir ada toplamda 3 beyaz hücreden oluşmalıdır(Rakam içeren hücre dahil).
    - Adalar yalnızca köşe noktalarından (diyagonal) temas edebilir. Yatay veya dikey birleşime izin verilmez.
2. **Denizler (Siyah Hücreler):**
    - Siyah hücreler, ızgarada tek bir bağlantılı yapı oluşturmalıdır. Başka bir deyişle, deniz hücreleri kopuk parçalardan oluşamaz.
    - Siyah hücreler hiçbir zaman 2x2 kare veya daha büyük bir kare oluşturamaz. Örneğin, 4 siyah hücre bir kare formunda birleşemez.
3. **Tüm Hücreler Atanmalıdır:**
    - Tüm hücreler ya siyah (deniz) ya da beyaz (ada) olarak atanmalıdır.
4. **Süreklilik ve İzolasyon:**
    - Siyah hücreler tek bir bağlantılı bütün oluştururken, beyaz hücrelerin de kendi rakamlarına bağlı olarak belirli bir mantıkla büyütülmesi gerekir.
    - Rakam hücreleri çevresindeki boş hücreler adanın bir parçası olacak şekilde genişletilir. Diyagonel birleşemezler.

Kısıtların Matematiksel İfadesi

Terimler:

L : Ada hücresi(Land),

S : Deniz hücresi (Sea),

C : Hücre (Cell)

Değişkenler (Variables)

c(1,1), c(1,2), c(1,3), ….. , c(n,m)

C = {c(i,j) : 1 ≤ i ≤ n, 1 ≤ i ≤ m}

Alanlar (Domains)

Dom[c(i,j)] = {L, S}, başlangıç durumunda boş hücre ise

Dom[c(i,j)] = {L}, başlangıç durumunda numaralı hücre ise

Dom[Num(c(i,j))]=1, Ada hücresi olup numaralı bir değere sahip ise

Dom[Num(c(i,j))]=0, Ada hücresi olup numaralı bir değere sahip değil ise

Kısıtlar (Constraints)

1. Adalar

a) Her adada tam olarak bir numaralandırılmış hücre bulunmalıdır.

$\exists!$ c(i,j) $\in$ L ve Num(c(i,j))=1

b) Numaralandırılmış her hücre bir ada hücresidir.

$\forall$ c(i,j) $\in$ Num(c(i,j))=1 ise c(i,j) $\in$ L

c) Adalar yalnızca köşe noktalarından (diyagonal) temas edebilir. Yatay veya dikey birleşime izin verilmez.

c(i,j) $\in$ L, c(k,l) $\in$ L′: L′, L≠L′

$$\forall c(i,j) \in L, \forall c(k,l) \in L', L \neq L' \Longrightarrow \begin{cases} |i-k| = 1 \wedge |j-l| = 1, (yalnızca\ diyagonal\ komsuluk) \\ |i-k| > |j-l| < 1\ yalnızca\ diyagonal\ komsuluk \end{cases}$$

2. Deniz

a) Siyah hücreler, ızgarada tek bir bağlantılı yapı oluşturmalıdır.

$\forall c(i,j) \in S, \exists c(k,l) \in S, |i-k| + |j-l| = 1$

b) Siyah hücreler hiçbir zaman 2x2 kare oluşturamazlar.

$\forall i,j (1 \leq i \leq n-1, 1 \leq j \leq m-1)$,

$\neg(c(i,j) \in S \wedge c(i+1,j) \in S \wedge c(i,j+1) \in S \wedge c(i+1,j+1) \in S)$

3. Tüm Hücreler

a) Her hücre ya ada ya da denizdir.

$$\forall c(i,j)(1\leq i\leq n, 1\leq j\leq m), c(i,j)\in\{L,S\}$$

4. Sürekli ve İzolasyon

a) Siyah hücreler tek bir bağlantılı bütün oluştururken, beyaz hücrelerin de kendi rakamlarına bağlı olarak belirli bir mantıkla büyütülmesi gerekir.

$$\forall c(i,j)\in S, \exists c(k,l)\in S, \mid i-k\mid + \mid j-l\mid =1$$

$$\forall c(k,l)\in L, \exists c(m,n)\in L, \mid k-m\mid + \mid l-n\mid =1, \text{ eğer } Num(c(i,j))>1 \text{ ise.}$$

b) Ada hücreleri kendi rakamına göre büyüme geliştirir. Numara sayısı kadar hücreden oluşur.

$$\exists! c(i,j)\in L,\ Dom[Num(c(i,j))]=1 \Longrightarrow \sum_{c(k,l)\in L(c(i,j))} 1 = Num(c(i,j))$$

Çözüm Stratejisi

1. Forward Checking (İleri Zincirleme)

- Hücreye bir değer atandıktan sonra, bu atamanın diğer hücrelerin çalışma evrenini nasıl etkilediği değerlendirilir.
- Eğer bir hücreye "Deniz" atanırsa:
  - Komşu hücrelerden bazılarının "Ada" olma ihtimali elenir.
  - Deniz sürekliliği kısıtı kontrol edilir.
- Eğer bir hücreye "Ada" atanırsa:
  - Rakamın belirttiği ada boyutuna ulaşılıp ulaşılmadığı kontrol edilir. Ada boyutuna ulaşıldıyda, diğer komşu hücreler domaininden Ada olma durumu silinir.

2. MRV (Minimum Remaining Values - Minimum Kalan Değerler)

- En az seçenekli hücrelere öncelik verilir.
- Bu strateji, çözümün dallanma faktörünü azaltarak hız kazandırır.

3. Sezgisel Yaklaşım: Minimum Rakamdan Başlayarak Genişleme

- Çözüm minimum değeri (ör. 1) olan numaralı hücreden başlamalıdır.
- Bu hücrelerin çevresinde büyütme işlemi, üstündeki hücreden başlayarak saat yönünde ilerleyerek devam etmiştir.

Algoritmanın Temel Adımları

A) Forward Checking (İleri Zincirleme)

Forward Checking algoritması, bir hücreye değer atanmasının diğer hücrelerin geçerli domainlerini nasıl etkilediğini kontrol eder. Örneğin:

- Bir hücre "deniz" olarak atanırsa, komşu hücrelerden bazılarının "ada" olma ihtimali elenir.
- Ada boyutuna ulaşılırsa, o adaya komşu diğer hücreler "deniz" olarak atanır.

B) MRV (Minimum Remaining Values)

MRV stratejisi, en az domain seçeneği kalan hücreleri öncelikli olarak ele alır. Bu, çözüm sürecinde dallanma faktörünü azaltarak çözümü hızlandırır.

C) Bağlılık ve Kısıtların Kontrolü

Süreklilik kuralına göre:

- Siyah hücreler tek bir bağlantılı yapı oluşturmalıdır.
- Siyah hücreler 2x2 kare oluşturmamalıdır.
- Adalar yalnızca köşe teması yapabilir.

D) Geriye Dönük İzleme (Backtracking)

Geriye dönük izleme algoritması, çözümün doğruluğu kısıtlarla doğrulanana kadar sistematik bir arama yöntemi kullanır. Kısıtları ihlal eden adımlarda geriye dönülür.

Kaynak Kodun İncelenmesi

Fonksiyon 1: initialize_domains

Bu fonksiyon, oyun tahtasındaki hücrelerin domainlerini tanımlar. Şu adımlar uygulanır:

- Boş hücreler hem "L" (ada) hem de "S" (deniz) olabilecek şekilde domain atanır.
- Sayısal hücreler yalnızca belirlenen ada boyutuna uygun domainle sınırlandırılır.

```python
def initialize_domains(game_board, file=None):
    """
    Hücreler için domainleri başlatır ve numaralandırılmış hücreleri tanımlar.
    - Boş hücreler hem ada hem deniz olabilir.
    - Numaralı hücreler yalnızca belirli ada boyutlarını temsil eder.
    """
    domains = {}
    numerical_cells = []
    for x in range(game_board.shape[0]):
        for y in range(game_board.shape[1]):
            if game_board[x, y] is None:
                # Boş hücre için domain
                domains[(x, y)] = ['L', 'S']
            elif isinstance(game_board[x, y], int):
                # Numaralı hücre için domain ve ada tanımı
                domains[(x, y)] = [f'L{game_board[x, y]}']
                numerical_cells.append({'konum': (x, y), 'deger': game_board[x, y]})

    # Domain bilgilerini dosyaya yazdır
    if file:
        file.write("Başlangıç domainleri:\n")
        for cell, domain in domains.items():
            file.write(f"{cell}: {domain}\n")
        file.write("\n")

    # Sayısal hücreleri dosyaya logla
    if file:
        file.write("Numerical hücreler:\n")
        for cell in numerical_cells:
            file.write(f"{cell}\n")
        file.write("\n")

    if file:
        file.write(f"Toplam ada sayısı: {len(numerical_cells)}\n\n")

    return domains, numerical_cells
```

Figure I: Fonksiyon 1

Bu fonksiyon, oyun tahtasındaki hücrelerin domainlerini tanımlar. Şu adımlar uygulanır:

- Boş hücreler hem "L" (ada) hem de "S" (deniz) olabilecek şekilde domain atanır.

- Sayısal hücreler yalnızca belirlenen ada boyutuna uygun domainle sınırlandırılır.

Fonksiyon 2: forward_checking

Bir hücreye değer atandıktan sonra komşu hücrelerin domainlerini günceller. Bu, geçerli olmayan durumların önceden elenmesini sağlar.

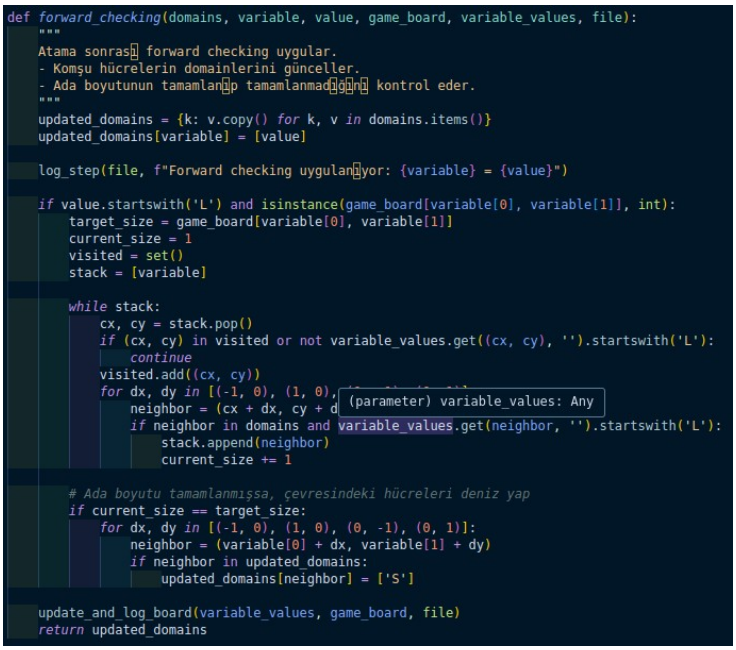


Figure II: Fonksiyon 2

Bir hücreye değer atandıktan sonra komşu hücrelerin domainlerini günceller. Bu, geçerli olmayan durumların önceden elenmesini sağlar.

Fonksiyon 3: select_next_variable

MRV stratejisini uygular ve en az domain seçeneği olan hücreyi belirler. Sayısal hücreler önceliklidir.

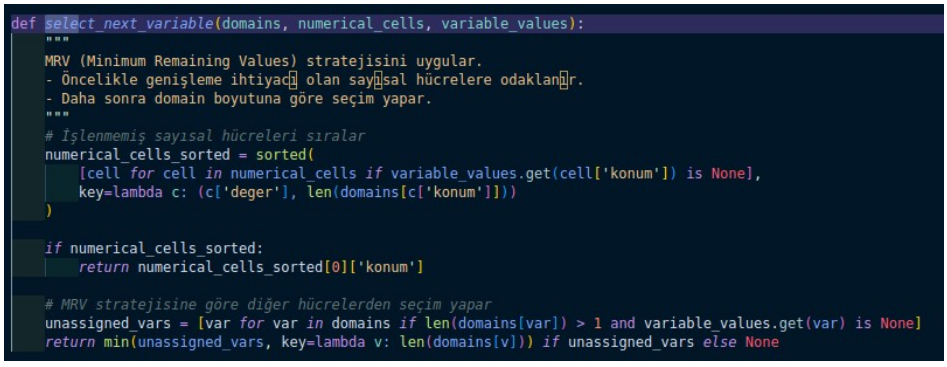
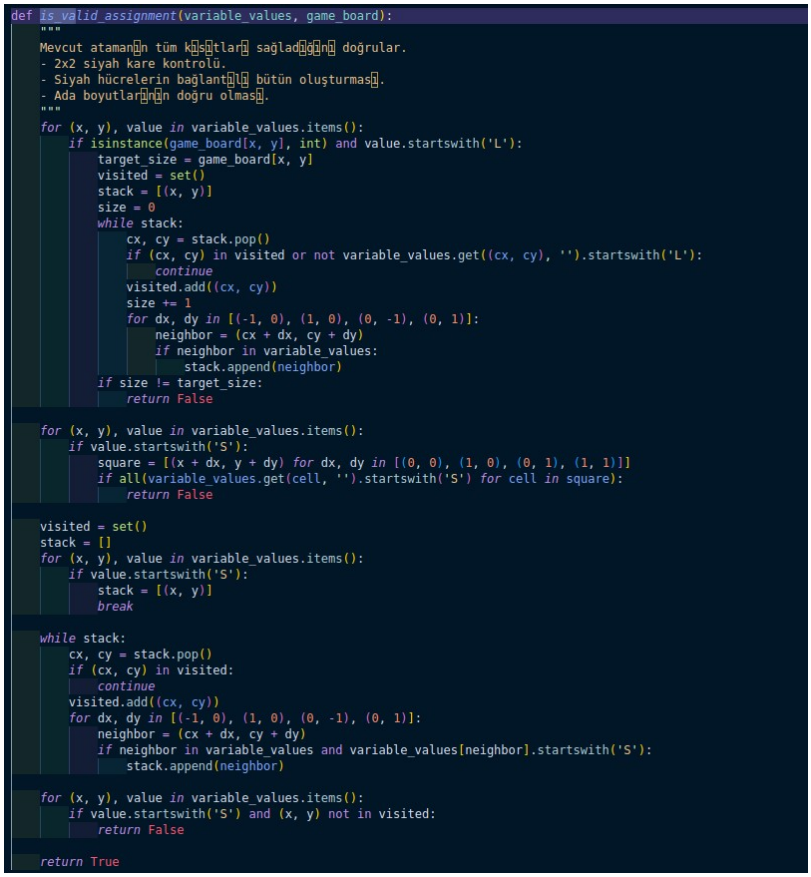


Figure III: Fonksiyon 3

MRV stratejisini uygular ve en az domain seçeneği olan hücreyi belirler. Sayısal hücreler önceliklidir.

Fonksiyon 4: is_valid_assignment

Atamışların tüm kısıtları karşılayıp karşılamadığını kontrol eder. Siyah hücrelerin bağlılığı, 2x2 kare oluşumu ve ada boyutları doğrulanır.



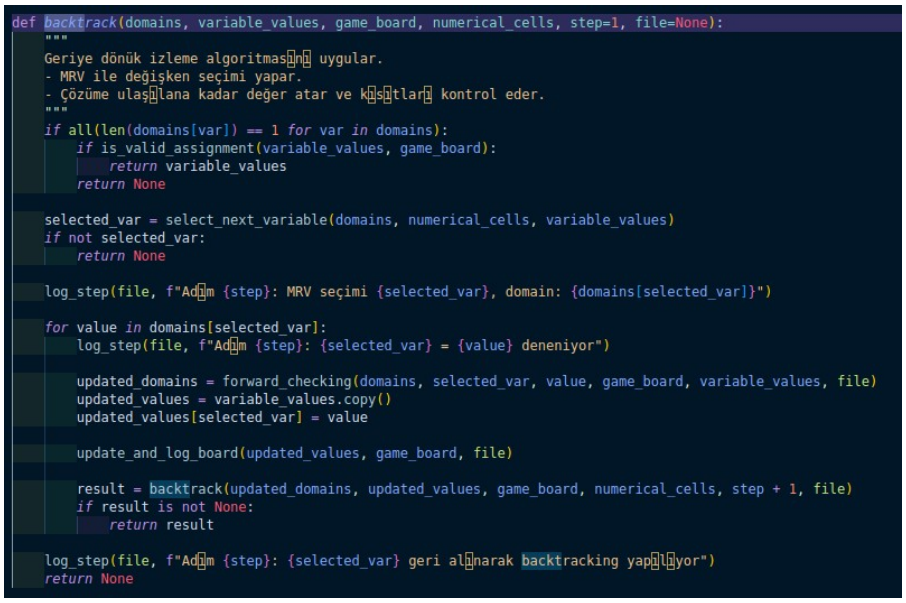
```
def is_valid_assignment(variable_values, game_board):
    """
    Mevcut atamanın tüm kısıtları sağladığını doğrular.
    - 2x2 siyah kare kontrolü.
    - Siyah hücrelerin bağlantılı bütün oluşturması.
    - Ada boyutlarının doğru olması.
    """
    for (x, y), value in variable_values.items():
        if isinstance(game_board[x, y], int) and value.startswith('L'):
            target_size = game_board[x, y]
            visited = set()
            stack = [(x, y)]
            size = 0
            while stack:
                cx, cy = stack.pop()
                if (cx, cy) in visited or not variable_values.get((cx, cy), '').startswith('L'):
                    continue
                visited.add((cx, cy))
                size += 1
                for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
                    neighbor = (cx + dx, cy + dy)
                    if neighbor in variable_values:
                        stack.append(neighbor)
            if size != target_size:
                return False

    for (x, y), value in variable_values.items():
        if value.startswith('S'):
            square = [(x + dx, y + dy) for dx, dy in [(0, 0), (1, 0), (0, 1), (1, 1)]]
            if all(variable_values.get(cell, '').startswith('S') for cell in square):
                return False

    visited = set()
    stack = []
    for (x, y), value in variable_values.items():
        if value.startswith('S'):
            stack = [(x, y)]
            break

    while stack:
        cx, cy = stack.pop()
        if (cx, cy) in visited:
            continue
        visited.add((cx, cy))
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            neighbor = (cx + dx, cy + dy)
            if neighbor in variable_values and variable_values[neighbor].startswith('S'):
                stack.append(neighbor)

    for (x, y), value in variable_values.items():
        if value.startswith('S') and (x, y) not in visited:
            return False

    return True
```

Figure IV: Fonksiyon 4

Atamışların tüm kısıtları karşılayıp karşılamadığını kontrol eder. Siyah hücrelerin bağlılığı, 2x2 kare oluşumu ve ada boyutları doğrulanır.

Fonksiyon 5: backtrack

Geriye dönük izleme algoritması, tüm domainler sabitlenene kadar çözüm arar. Yanlış atamalarda geriye dönülerek alternatif yollar denenir.



```
def backtrack(domains, variable_values, game_board, numerical_cells, step=1, file=None):
    """
    Geriye dönük izleme algoritmasını uygular.
    - MRV ile değişken seçimi yapar.
    - Çözüme ulaşılana kadar değer atar ve kısıtları kontrol eder.
    """
    if all(len(domains[var]) == 1 for var in domains):
        if is_valid_assignment(variable_values, game_board):
            return variable_values
        return None

    selected_var = select_next_variable(domains, numerical_cells, variable_values)
    if not selected_var:
        return None

    log_step(file, f"Adım {step}: MRV seçimi {selected_var}, domain: {domains[selected_var]}")

    for value in domains[selected_var]:
        log_step(file, f"Adım {step}: {selected_var} = {value} deneniyor")

        updated_domains = forward_checking(domains, selected_var, value, game_board, variable_values, file)
        updated_values = variable_values.copy()
        updated_values[selected_var] = value

        update_and_log_board(updated_values, game_board, file)

        result = backtrack(updated_domains, updated_values, game_board, numerical_cells, step + 1, file)
        if result is not None:
            return result

    log_step(file, f"Adım {step}: {selected_var} geri alınarak backtracking yapılıyor")
    return None
```

Figure V: Fonksiyon 5

Geriye dönük izleme algoritması, tüm domainler sabitlenene kadar çözüm arar. Yanlış atamalarda geriye dönülerek alternatif yollar denenir.

Arama Ağacı

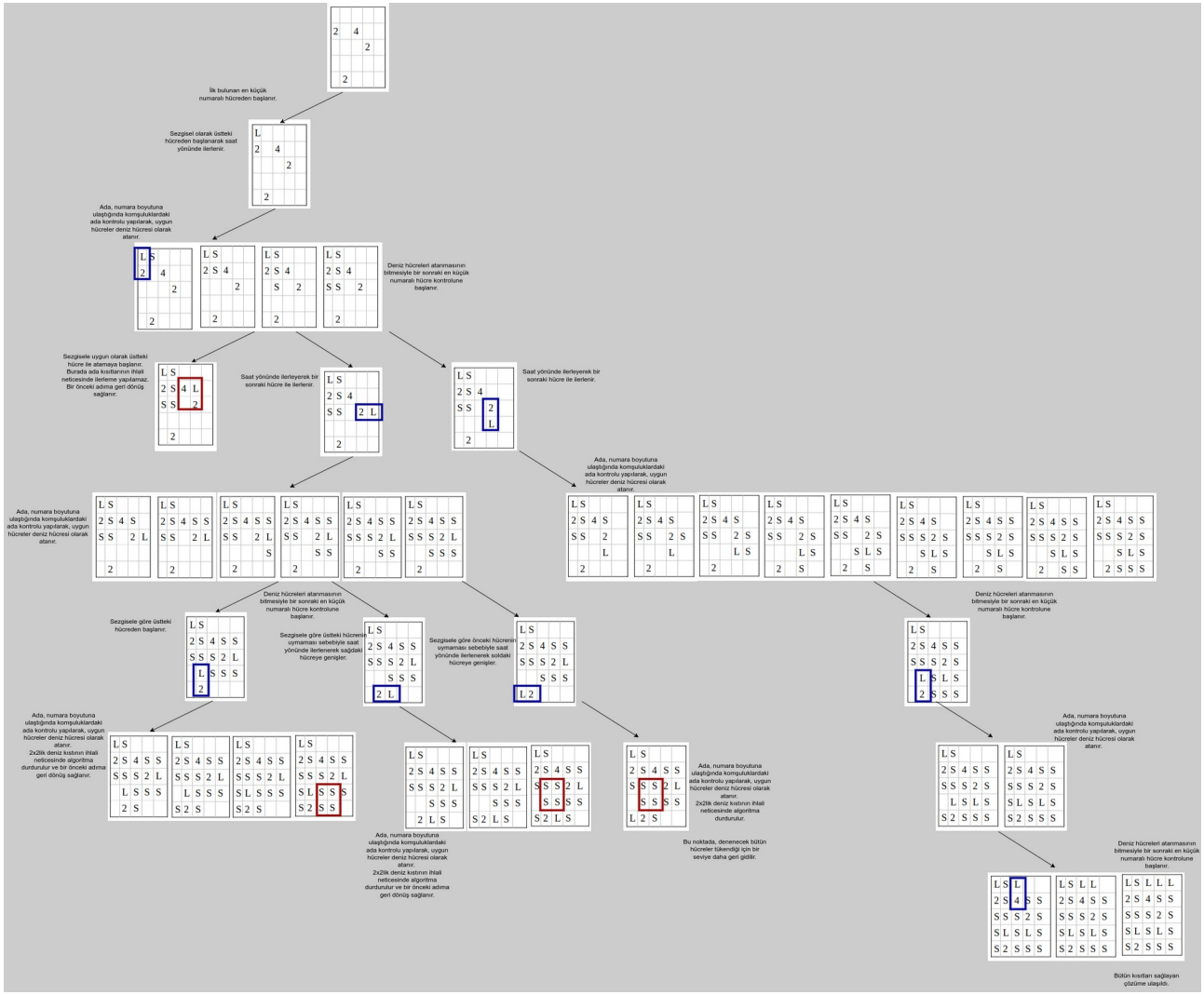


Figure VI: Arama Ağacı

Sonuçlar

Yazılan Python kodu Ubuntu 22.04.5 LTS işletim sistemi üzerinde geliştirilmiştir. Bilgisayarın teknik özellikleri aşağıda sıralandığı gibidir;

- 16 GB RAM,
- Intel Corei7-6700HQ @ 2.60GHz × 8 işlemci,
- NVIDIA Corporation GM107M [GeForce GTX 950M] / Mesa Intel® HD ekran kartı.

1. **Durum**

   Gerçeklenen senaryo :

   2 NaN NaN 3, NaN NaN NaN NaN, NaN NaN NaN NaN, 4 NaN NaN NaN

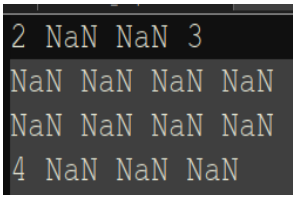


Figure VII: Girdi 1

Son ekran çıktısı :

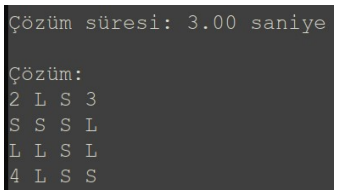


Figure VIII: Çıktı 1

2. **Durum**

   Gerçeklenen senaryo :

   NaN NaN NaN 1, 1 NaN NaN NaN, NaN NaN 2 NaN, 1 NaN NaN NaN

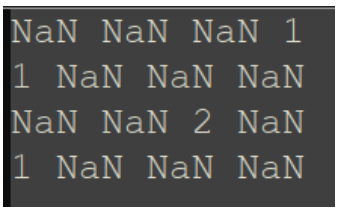


Figure IX: Girdi 2

Son ekran çıktısı :

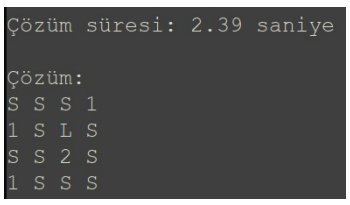


Figure X: Çıktı 2

3. Gerçeklenen senaryo :

   2 NaN NaN NaN, NaN NaN NaN NaN, NaN NaN NaN NaN , NaN 1 NaN 4

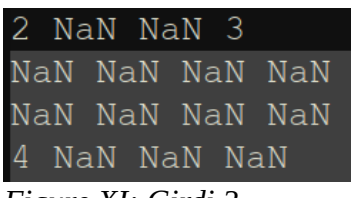


Figure XI: Girdi 3

Son ekran çıktısı :

```
Çözüm süresi: 3.59 saniye

Çözüm:
2 L S 3
S S S L
L L S L
4 L S S
```

Figure XII: Çıktı 3

4. Durum

Gerçeklenen senaryo :

2 NaN NaN 2, NaN NaN NaN NaN, NaN NaN NaN NaN, NaN 2 NaN 1

```
2 NaN NaN 2
NaN NaN NaN NaN
NaN NaN NaN NaN
NaN 2 NaN 1
```

Figure XIII: Girdi 4

Son ekran çıktısı :

```
Çözüm süresi: 1.15 saniye

Çözüm:
2 L S 2
S S S L
S L S S
S 2 S 1
```

Figure XIV: Çıktı 4

5. Durum

Gerçeklenen senaryo :

NaN NaN NaN NaN NaN , 2 NaN 4 NaN NaN, NaN NaN NaN 2 NaN, NaN NaN NaN NaN NaN, NaN 2 NaN NaN NaN

```
NaN NaN NaN NaN NaN
2 NaN 4 NaN NaN
NaN NaN NaN 2 NaN
NaN NaN NaN NaN NaN
NaN 2 NaN NaN NaN
```

Figure XV: Girdi 5

Son ekran çıktısı :

```
Çözüm süresi: 1882.41 saniye

Çözüm:
L S L L L
2 S 4 S S
S S S 2 S
S L S L S
S 2 S S S
```

Figure XVI: Çıktı 5

6. **Durum**

   Gerçeklenen senaryo :

   NaN NaN NaN NaN NaN , NaN 1 NaN 1 NaN, NaN NaN 2 NaN 3, NaN NaN NaN NaN NaN, 2 NaN NaN NaN NaN

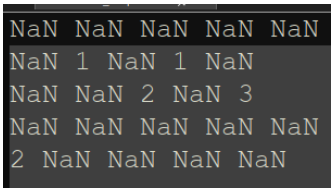


Figure XVII: Girdi 6

Son ekran çıktısı :

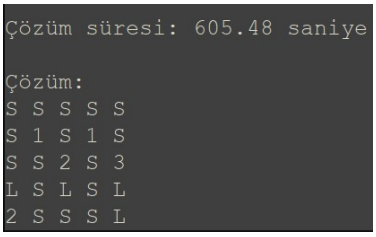


Figure XVIII: Çıktı 6

Referans Listesi

1. https://en.wikipedia.org/wiki/Nurikabe_(puzzle)
2. https://www.logicgamesonline.com/nurikabe/
3. Russell, S., & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach*. Pearson.
4. https://www.logicgamesonline.com/nurikabe/
5. Constraint Satisfaction Problem Theory: Van Roy, P., & Haridi, S. (2004). *Concepts, Techniques, and Models of Computer Programming*. MIT Press.