SAKARYA UYGULAMALI BİLİMLER ÜNİVERSİTESİ

TEKNOLOJİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

SAKARYA
UYGULAMALI BİLİMLER
ÜNİVERSİTESİ

- YAPAY ZEKANIN İLKELERİ DERSİ PROJE RAPORU -

- SU ŞİŞESİ PROBLEMİ BFS ve DFS -

Öğretim Görevlisi : Doç. Dr. Ekin EKİNCİ

Arş. Gör. Furkan ATBAN

HAZIRLAYANLAR:

23010903055 - FATMA YAŞAR

23010903049 - SÜMEYYE GÜL

1.Proje Amacı

Bu projenin amacı, yapay zekâda temel konulardan biri olan **durum uzayında arama algoritmalarını** uygulamalı olarak öğrenmek ve analiz etmektir. Bu doğrultuda, klasik yapay zekâ problemlerinden biri olan **Su Şişesi Problemi** ele alınmıştır. Problemin çözümünde, sınırlı kapasiteye sahip iki şişe kullanılarak belirli bir hedef su miktarına ulaşılması hedeflenmiştir.

Çözüm için **Breadth-First Search (BFS)** ve **Depth-First Search (DFS)** algoritmaları uygulanarak, algoritmaların problem çözme süreçleri karşılaştırmalı olarak incelenmiştir. Ayrıca, her iki algoritmanın farklı senaryolarda nasıl çalıştığı, adım adım geçilen durumlar üzerinden analiz edilmiştir.

2.Proje İçeriği

Problemde iki farklı kapasiteye sahip su şişesi kullanılarak, belirli bir hedef miktarda su elde edilmeye çalışılmıştır.

Proje kapsamında:

- Problemin **durum uzayı** ve **geçiş kuralları** tanımlanmıştır.
- **Breadth-First Search (BFS)** ve **Depth-First Search (DFS)** algoritmaları Python dilinde uygulanmıştır.
- Farklı başlangıç durumları için algoritmalar test edilmiş, çözüm yolları adım adım gösterilmiştir.
- Her algoritmanın performansı karşılaştırılmış, çözüm bulunabilirliği değerlendirilmiştir.
- Sonuçlar tablo ve çıktı halinde görselleştirilmiştir.

3.Giriş

Bu projede, iki farklı arama algoritması olan **BFS** ve **DFS** yöntemleri kullanılarak klasik **su şişesi problemi** çözülmüştür. Amaç, verilen iki şişe kapasitesi ve hedef hacme ulaşmak için gerekli adımları belirlemektir.

4. Problem Tanımı

İki şişemiz bulunmaktadır. Her iki şişeye de su doldurma, boşaltma veya birbirlerine dökme işlemleri uygulanabilir. Verilen bir hedef miktarda su elde edilmek istenmektedir.

5. Geçiş Kuralları

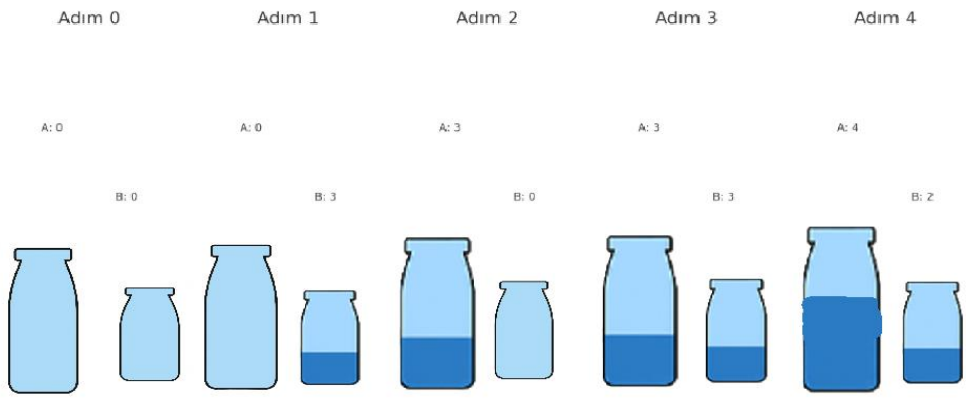
Bir durumdan diğerine geçiş yapmak için uygulanabilecek kurallar:

- **Bir şişeyi tamamen doldurmak:**
  - A şişesini doldur: `(a_kapasite, b)`
  - B şişesini doldur: `(a, b_kapasite)`

- **Bir şişeyi tamamen boşaltmak:**
  - A şişesini boşalt: `(0, b)`
  - B şişesini boşalt: `(a, 0)`

- **Bir şişedeki suyu diğer şişeye dökmek:**
  - A'dan B'ye dök:
    - Miktar: `min(a, b_kapasite - b)`
    - Yeni durum: `(a - miktar, b + miktar)`
  - B'den A'ya dök:
    - Miktar: `min(b, a_kapasite - a)`
    - Yeni durum: `(a + miktar, b – miktar)`

Geçiş Kuralları Adımları :



6.Kod Yapısı ve Açıklamaları

1.EBOB Fonksiyonu : En Büyük Ortak Bölenini hesaplar. EBOB, hedef miktarın şişelerle elde edilip edilemeyeceğini kontrol etmek için kullanılır. Hedef, EBOB'a bölünebiliyorsa çözüm mümkündür.

Çalışma Prensibi:
EBOB hesaplanır. Eğer b sıfırsa, a değeri EBOB olarak döndürülür. Aksi takdirde, a ve b'nin modülünü alarak işlem devam eder. Bu, b sıfır olana kadar tekrarlanır.

```
# EBOB (GCD) hesaplama fonksiyonu
def ebob(a, b):
    if b == 0:
        return a
    return ebob(b, a % b)
```

2. BFS Fonksiyonu (`bfs_agaci(a_kapasite, b_kapasite, hedef)`): Bu fonksiyon, su şişesi problemini BFS yöntemiyle çözerek en kısa çözüm yolunu bulur.

- **Başlangıç**: (0, 0) durumuyla başlar.
- **Yapı**: Kuyruk kullanılarak her durum sırayla işlenir.
- **Ziyaret Takibi**: Aynı durum tekrar işlenmez.
- **Geçişler**: Şişeleri doldurma, boşaltma ve birbirine dökme işlemleriyle yeni durumlar üretilir.
- **Ebeveyn Haritası**: Hedefe ulaşıldığında çözüm yolu geriye dönük izlenerek oluşturulur.

- **Çıktı**: Hedefe ulaşılırsa çözüm yolu, ulaşılmazsa None döner.

```
# BFS (Genislik oncelikli Arama) ile cozum bulma
def bfs_agaci(a_kapasite, b_kapasite, hedef):
    ziyaret_edilenler = set()
    kuyruk = deque()
    ebeveyn_haritasi = {}

    baslangic = (0, 0)
    kuyruk.append(baslangic)
    ziyaret_edilenler.add(baslangic)
    ebeveyn_haritasi[baslangic] = None
```

3. **DFS Fonksiyonu (`dfs_agaci(a_kapasite, b_kapasite, hedef)`):** Bu fonksiyon, problemi DFS yöntemiyle çözmeye çalışarak çözüm yolunu derinlemesine arar.

- **Başlangıç**: (0, 0) durumuyla başlar.
- **Yapı**: Yığın kullanılır, derinlik öncelikli ilerleme yapılır.
- **Ziyaret Takibi**: Her durum yalnızca bir kez ziyaret edilir.
- **Geçişler**: BFS ile aynı geçiş kuralları uygulanır.
- **Ebeveyn Haritası**: Çözüm bulunduğunda yol geriye izlenerek oluşturulur.
- **Çıktı**: Hedefe ulaşılıyorsa çözüm yolu, değilse None döner.

```
# DFS (Derinlik Oncelikli Arama) ile cozum bulma
def dfs_agaci(a_kapasite, b_kapasite, hedef):
    ziyaret_edilenler = set()
    yigin = []
    ebeveyn_haritasi = {}

    baslangic = (0, 0)
    yigin.append(baslangic)
    ziyaret_edilenler.add(baslangic)
    ebeveyn_haritasi[baslangic] = None
```

7.Test Senaryoları

Senaryo 1: *A Kapasite: 4, B Kapasite: 3, Hedef: 2*

BFS Test Senaryoları:

- **Başlangıç Durumu Doğrulama:**
  - Başlangıç durumu `(0, 0)` olmalı.
- **Geçişlerin Doğruluğu:**
  - `(0, 0)` → `(4, 0)` geçişi A şişesinin doldurulmasıyla yapılmalı.
  - `(4, 0)` → `(0, 3)` geçişi B şişesinin doldurulmasıyla yapılmalı.
  - `(0, 3)` → `(4, 3)` geçişi A şişesinin doldurulmasıyla yapılmalı.
  - `(4, 3)` → `(1, 3)` geçişi A'dan B'ye dökme işlemiyle yapılmalı.

- `(1, 3)` → `(3, 0)` geçişi B şişesinin boşaltılmasıyla yapılmalı.
- `(3, 0)` → `(1, 0)` geçişi A'dan B'ye dökme işlemiyle yapılmalı.
- `(1, 0)` → `(3, 3)` geçişi B şişesinin doldurulmasıyla yapılmalı.
- `(3, 3)` → `(0, 1)` geçişi A'dan B'ye dökme işlemiyle yapılmalı.
- `(0, 1)` → `(4, 2)` geçişi A şişesinin doldurulmasıyla yapılmalı.

❖ **Hedef Durum Testi:**
- Son durumda, `(4, 2)` olmalı ve hedef doğru bir şekilde ulaşılmalı.

DFS Test Senaryoları:

❖ **Başlangıç Durumu Doğrulama:**
- Başlangıç durumu `(0, 0)` olmalı.

❖ **Geçişlerin Doğruluğu:**
- `(0, 0)` → `(0, 3)` geçişi B şişesinin doldurulmasıyla yapılmalı.
- `(0, 3)` → `(3, 0)` geçişi B'den A'ya dökme işlemiyle yapılmalı.
- `(3, 0)` → `(3, 3)` geçişi B şişesinin doldurulmasıyla yapılmalı.
- `(3, 3)` → `(4, 2)` geçişi A şişesinin doldurulup A'dan B'ye dökme işlemiyle yapılmalı.

❖ **Hedef Durum Testi:**
- A kabında 2 litre su bulunduğu `(4, 2)` hedef durumu doğrulanmalı.

Senaryo 2: *A Kapasite: 5, B Kapasite: 2, Hedef: 1*

BFS Test Senaryoları:

❖ **Başlangıç Durumu Doğrulama:**
- Başlangıç durumu `(0, 0)` olmalı.

❖ **Geçişlerin Doğruluğu:**
- `(0, 0)` → `(5, 0)` geçişi A şişesinin doldurulmasıyla yapılmalı.
- `(5, 0)` → `(0, 2)` geçişi B şişesinin doldurulmasıyla yapılmalı.
- `(0, 2)` → `(5, 2)` geçişi A şişesinin doldurulmasıyla yapılmalı.
- `(5, 2)` → `(3, 2)` geçişi A'dan B'ye dökme işlemiyle yapılmalı.
- `(3, 2)` → `(2, 0)` geçişi B şişesinin boşaltılmasıyla yapılmalı.
- `(2, 0)` → `(3, 0)` geçişi A şişesinin doldurulmasıyla yapılmalı.
- `(3, 0)` → `(2, 2)` geçişi B şişesinin doldurulmasıyla yapılmalı.
- `(2, 2)` → `(1, 2)` geçişi A'dan B'ye dökme işlemiyle yapılmalı.

❖ **Hedef Durum Testi:**
- Son durumda, `(1, 2)` olmalı ve hedef doğru bir şekilde ulaşılmalı.

DFS Test Senaryoları:

- **Başlangıç Durumu Doğrulama:**
  - Başlangıç durumu `(0, 0)` olmalı.
- **Geçişlerin Doğruluğu:**
  - `(0, 0)` → `(0, 2)` geçişi B şişesinin doldurulmasıyla yapılmalı.
  - `(0, 2)` → `(2, 0)` geçişi B'den A'ya dökme işlemiyle yapılmalı.
  - `(2, 0)` → `(2, 2)` geçişi B şişesinin tekrar doldurulmasıyla yapılmalı.
  - `(2, 2)` → `(4, 0)` geçişi B'den A'ya dökme işlemiyle yapılmalı.
  - `(4, 0)` → `(4, 2)` geçişi B şişesinin doldurulmasıyla yapılmalı.
  - `(4, 2)` → `(5, 1)` geçişi A şişesinin doldurulup A'dan B'ye dökme işlemiyle yapılmalı.
- **Hedef Durum Testi:**
  - B kabında 1 litre su bulunduğu `(5, 1)` hedef durumu doğrulanmalı.

Senaryo 3: *A Kapasite: 6, B Kapasite: 4, Hedef: 2*

BFS Test Senaryoları:

- **Başlangıç Durumu Doğrulama:**
  - Başlangıç durumu `(0, 0)` olmalı.
- **Geçişlerin Doğruluğu:**
  - `(0, 0)` → `(6, 0)` geçişi A şişesinin doldurulmasıyla yapılmalı.
  - `(6, 0)` → `(0, 4)` geçişi B şişesinin doldurulmasıyla yapılmalı.
  - `(0, 4)` → `(6, 4)` geçişi A şişesinin doldurulmasıyla yapılmalı.
  - `(6, 4)` → `(2, 4)` geçişi A'dan B'ye dökme işlemiyle yapılmalı.
- **Hedef Durum Testi:**
  - Son durumda, `(2, 4)` olmalı ve hedef doğru bir şekilde ulaşılmalı.

DFS Test Senaryoları:

- **Başlangıç Durumu Doğrulama:**
  - Başlangıç durumu `(0, 0)` olmalı.
- **Geçişlerin Doğruluğu:**
  - `(0, 0)` → `(0, 4)` geçişi B şişesinin doldurulmasıyla yapılmalı.
  - `(0, 4)` → `(4, 0)` geçişi B'den A'ya dökme işlemiyle yapılmalı.
  - `(4, 0)` → `(4, 4)` geçişi B şişesinin doldurulmasıyla yapılmalı.
  - `(4, 4)` → `(6, 2)` geçişi A şişesinin doldurulup A'dan B'ye dökme işlemiyle yapılmalı.
- **Hedef Durum Testi:**
  - B kabında 2 litre su bulunduğu `(6, 2)` hedef durumu doğrulanmalı.

8.Durum Uzayları

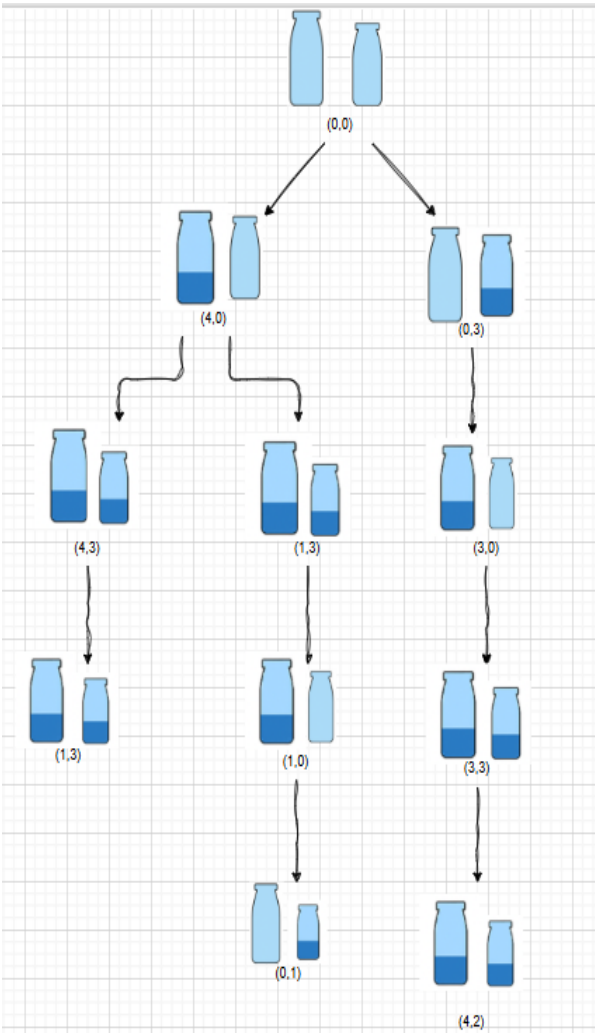
Senaryo 1 (A = 4, B = 3, Hedef = 2)

BFS Durum Uzayı:

```
(0, 0),(4, 0),(0, 3),(4, 3),(1, 3),(3, 0),(1, 0),(3, 3),(0, 1),(4,2)
```

BFS ile Bulunan Çözüm Yolu:

```
(0, 0) → (0, 3) → (3, 0) → (3, 3) → (4, 2)
```



(0,0)
- ⟶ (4,0) // 4'lük kabı doldur
- ⟶ (0,3) // 3'lük kabı doldur

(4,0)
- ⟶ (4,3) // 4'lükten 3'lüğe dök → 3'lük doldu
- ⟶ (1,3) // 4'lükten 3'lüğe dök → 4'lükte 1 kaldı, 3'lük doldu

(0,3)
- ⟶ (3,0) // 3'lükten 4'lüğe dök → 4'lükte 3 olur, 3'lük boşalır

(4,3)
- ⟶ (1,3) // 3'lük boşaltılır, sonra tekrar dökülür (aynı durum gelebilir)

(1,3)
- ⟶ (1,0) // 3'lük kabı boşalt

(3,0)
- ⟶ (3,3) // 3'lük kabı doldur

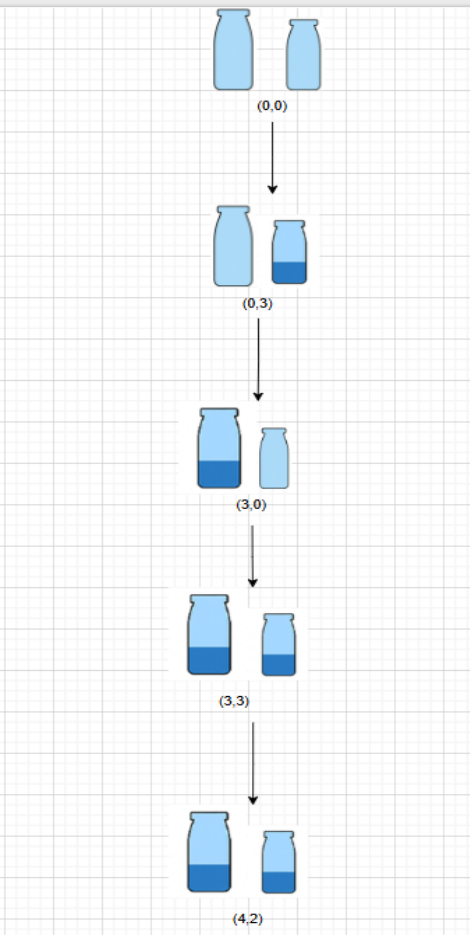
(1,0)
- ⟶ (0,1) // 4'lükten 3'lüge boşalt 4'lük boş, 3'lükte 1 kalır

(3,3)
- ⟶ (4,2) // 3'lükten 4'lüge doldur tamamen 3'lükte 2 kalır

DFS Durum Uzayı:

(0, 0), (0, 3), (3, 0), (3, 3), (4, 2)

DFS ile Bulunan Çözüm Yolu:
(0, 0) → (0, 3) → (3, 0) → (3, 3) → (4, 2)



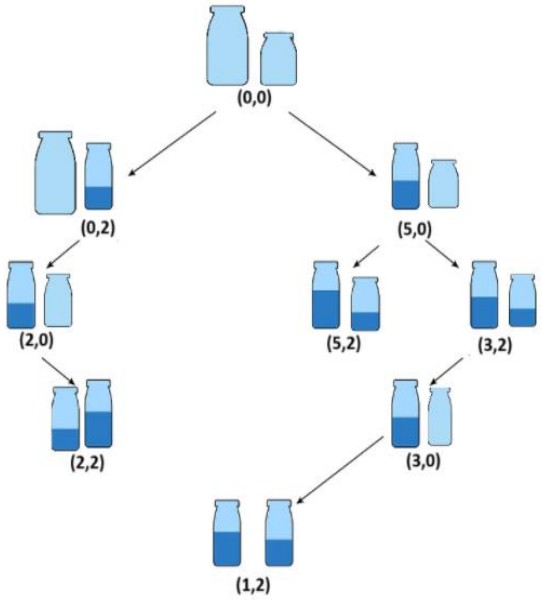

```
0, 0)  // Başlangıç, her iki kap da boş
  └──> (0, 3)  // 3 litrelik kabı doldur
       └──> (3, 0)  // 3 litrelik sudan 4 litrelik kaba dök
            └──> (3, 3)  // 3 litrelik kabı tekrar doldur
                 └──> (4, 2)  // 3 litrelik sudan 4 litrelik kaba 2 litre dök → Hedef: 2 litre
```

Senaryo 2 (A = 5, B = 2, Hedef = 1)

BFS Durum Uzayı:

(0, 0), (5, 0), (0, 2), (5, 2), (3, 2), (2, 0), (3, 0), (2, 2), (1, 2)

BFS ile Bulunan Çözüm Yolu:
(0, 0) → (5, 0) → (3, 2) → (3, 0) → (1, 2)



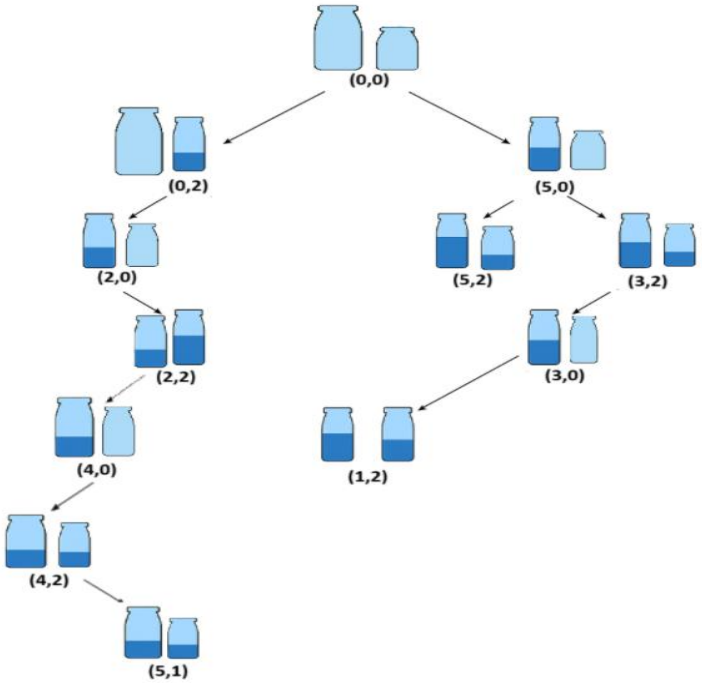

```
(0,0)
 ├──> (5,0)      // 5'lik kabı doldur
 └──> (0,2)      // 2'lik kabı doldur

(5,0)
 ├──> (5,2)      // 5'lik kaptan 2'liğe dök (2'lik doldu)
 └──> (3,2)      // 5'lik kaptan 2'liğe dök → 2'lik doldu, 5'likte 3 kaldı

(0,2)
 └──> (2,0)      // 2'lik kaptan 5'liğe dök (sonuç: 2'lik boş, 5'likte 2 oldu)

(3,2)
 └──> (3,0)      // 2'lik kabı boşalt

(2,0)
 └──> (2,2)      // 2'lik kabı doldur

(3,0)
 └──> (1,2)      // 3'lükten 2'liğe dök → 5'likte 1 kaldı, 2'lik doldu
```

DFS Durum Uzayı:

(0, 0), (0, 2), (2, 0), (2, 2), (4, 0), (4, 2), (5, 1)

DFS ile Bulunan Çözüm Yolu:

(0, 0) → (0, 2) → (2, 0) → (2, 2) → (4, 0) → (4, 2) → (5, 1)



(0,0) // Her iki kap da boş
└──> (0,2) // 2'lik kabı doldur

(0,2)
└──> (2,0) // 2'lik kaptaki suyu 5'lik kaba dök → 5'likte 2, 2'lik boş

(2,0)
└──> (2,2) // 2'liği tekrar doldur

(2,2)
└──> (4,0) // 2'liği 5'liğe dök → 5'likte toplam 4 olur

(4,0)
└──> (4,2) // 2'lik kabı tekrar doldur

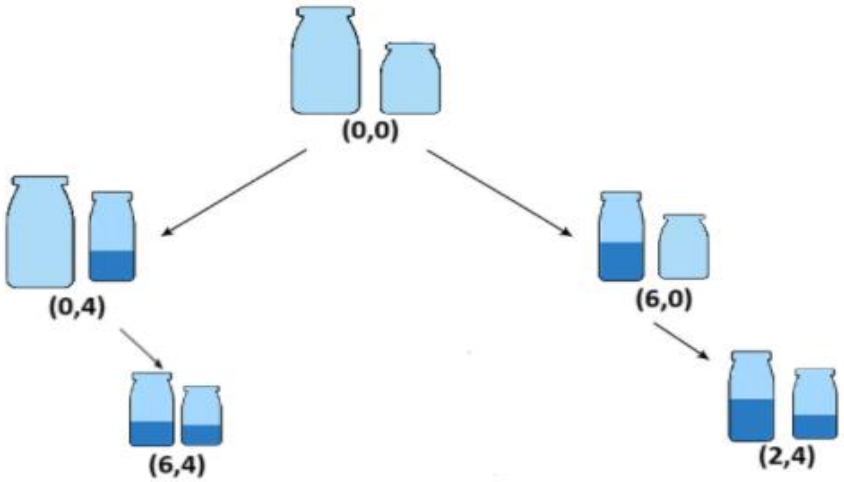
(4,2)
└──> (5,1) // 2'lik kaptan 5'liğe dök → 5'lik doldu (5), 2'likte 1 kaldı

Senaryo 3 (A = 6, B = 4, Hedef = 2)

BFS Durum Uzayı:

`(0, 0), (6, 0), (0, 4), (6, 4), (2, 4)`

BFS ile Bulunan Çözüm Yolu:
`(0, 0) → (6, 0) → (2, 4)`

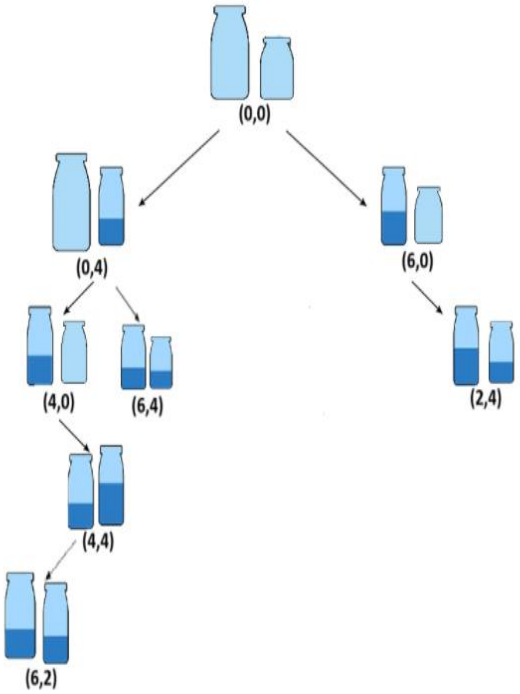



```
(0,0)                 // Başlangıç, her iki kap da boş
 ├──> (0,4)           // 4 litrelik kabı doldur
 │     ├──> (4,0)     // 4 litrelik sudan 6 litrelik kaba dök
 │     │
 │     └──> (6,4)     // 0 litrelik sudan 6 litrelik kaba doldur
 └──> (6,0)           // 6 litrelik kabı doldur
       └──> (2,4)     // 6 litrelikten 4 litrelik kaba 2 litre dök → Hedef: 2 litre
```

DFS Durum Uzayı:

(0, 0), (0, 4), (4, 0), (4, 4), (6, 2)

DFS ile Bulunan Çözüm Yolu:
(0, 0) → (0, 4) → (4, 0) → (4, 4) → (6, 2)

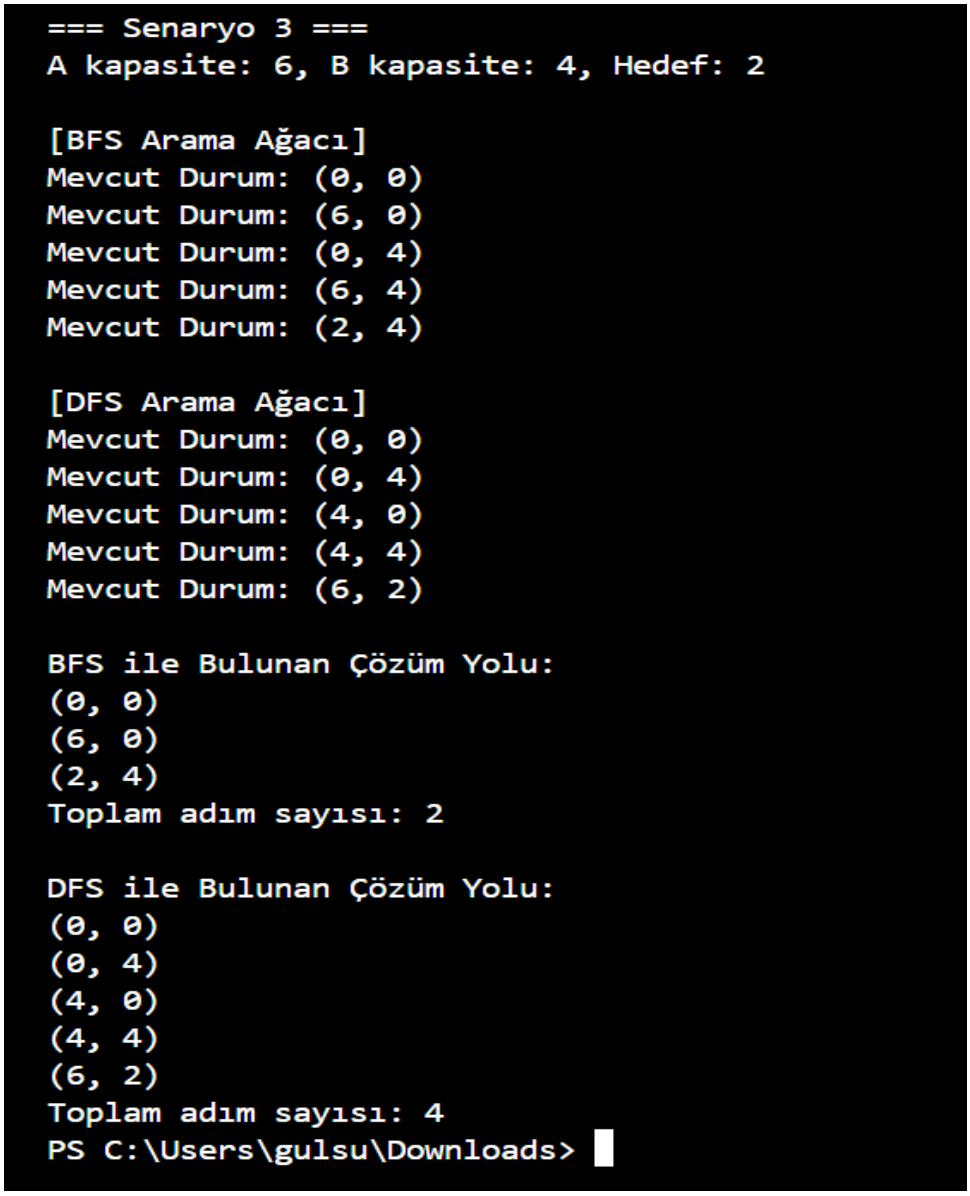



```
(0,0)                  // Başlangıç, her iki kap da boş
 └──> (0,4)             // 4 litrelik kabı doldur
       └──> (4,0)       // 4 litrelik sudan 6 litrelik kaba dök
             └──> (4,4) // 4 litrelik kabı tekrar doldur
                   └──> (6,2) // 4 litrelik sudan 6 litrelik kaba 2 litre dök → Hedef: 2 litre
```

9.Test Çıktıları

```
=== Senaryo 1 ===
A kapasite: 4, B kapasite: 3, Hedef: 2

[BFS Arama Ağacı]
Mevcut Durum: (0, 0)
Mevcut Durum: (4, 0)
Mevcut Durum: (0, 3)
Mevcut Durum: (4, 3)
Mevcut Durum: (1, 3)
Mevcut Durum: (3, 0)
Mevcut Durum: (1, 0)
Mevcut Durum: (3, 3)
Mevcut Durum: (0, 1)
Mevcut Durum: (4, 2)

[DFS Arama Ağacı]
Mevcut Durum: (0, 0)
Mevcut Durum: (0, 3)
Mevcut Durum: (3, 0)
Mevcut Durum: (3, 3)
Mevcut Durum: (4, 2)

BFS ile Bulunan Çözüm Yolu:
(0, 0)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
Toplam adım sayısı: 4

DFS ile Bulunan Çözüm Yolu:
(0, 0)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
Toplam adım sayısı: 4
```

```
=== Senaryo 2 ===
A kapasite: 5, B kapasite: 2, Hedef: 1

[BFS Arama Ağacı]
Mevcut Durum: (0, 0)
Mevcut Durum: (5, 0)
Mevcut Durum: (0, 2)
Mevcut Durum: (5, 2)
Mevcut Durum: (3, 2)
Mevcut Durum: (2, 0)
Mevcut Durum: (3, 0)
Mevcut Durum: (2, 2)
Mevcut Durum: (1, 2)

[DFS Arama Ağacı]
Mevcut Durum: (0, 0)
Mevcut Durum: (0, 2)
Mevcut Durum: (2, 0)
Mevcut Durum: (2, 2)
Mevcut Durum: (4, 0)
Mevcut Durum: (4, 2)
Mevcut Durum: (5, 1)

BFS ile Bulunan Çözüm Yolu:
(0, 0)
(5, 0)
(3, 2)
(3, 0)
(1, 2)
Toplam adım sayısı: 4

DFS ile Bulunan Çözüm Yolu:
(0, 0)
(0, 2)
(2, 0)
(2, 2)
(4, 0)
(4, 2)
(5, 1)
Toplam adım sayısı: 6
```



Senaryoların Karşılaştırmalı Analizi:

Senaryo 1 – A:4, B:3, Hedef:2

- **BFS ve DFS aynı çözüm yolunu** bulmuş (aynı düğümler, aynı adım sayısı).
- Adım sayısı: 4

- Bu, hedefe ulaşmak için arama uzayının simetrik veya basit olabileceğini gösteriyor. DFS'in dezavantajları burada görünmüyor.

Senaryo 2 – A:5, B:2, Hedef:1

- **BFS daha kısa çözüm bulmuş (4 adım)**.
- DFS ise daha uzun bir yol izleyip **6 adımda hedefe ulaşmış**.
- Bu durum, DFS'in derinlere dalıp hedefi geçici olarak gözden kaçırabileceğini ve optimal olmayan yollar bulabileceğini gösteriyor.
- BFS, hedefe en kısa yolu bulduğu için daha verimli.

Senaryo 3 – A:6, B:4, Hedef:2

- BFS sadece **2 adımda çözüme ulaşmış**, çok verimli.
- DFS ise **4 adımda** ulaşmış, yine daha uzun bir yol seçmiş.
- Burada da BFS'in optimal çözüm üretme avantajı açıkça görülüyor.