Proje Raporu

Proje Adı: Akıllı Arama Simülatörü

Grup Üyeleri:

- [Hatice Sena BAYAR] ([22060348])
- [İlayda ÇETİN] ([22060392])

1. Giriş

Bu proje, uninformed ve bilgili informed arama stratejilerinin performansını bir labirent ortamı üzerinde simüle etmek amacıyla gerçekleştirilmiştir. Proje kapsamında **Genişlik Öncelikli Arama (BFS)**, **Derinlik Öncelikli Arama (DFS)** ve **A* Arama Algoritması** labirent çözümü için uygulanmıştır.

Temel amaç, farklı labirent yapıları ve başlangıç/hedef koşulları altında algoritmaların etkinliklerini; **genişletilen düğüm sayısı**, **bulunan yolun maliyeti (uzunluğu)** ve **algoritmanın çalışma süresi** gibi metrikler üzerinden karşılaştırmaktır.

2. Yöntem (Kullanılan Algoritmalar ve Uygulama)

2.1 Labirent Yapısı (Maze Sınıfı)

Simülatör, **maze.py** dosyasında tanımlanan Maze sınıfı üzerinden çalışır.

- Labirentler, .txt dosyalarından okunur ve S (Başlangıç), G (Hedef), # (Duvar) ve . (Yol) karakterlerinden oluşur.
- Hareket, labirent üzerinde yukarı, aşağı, sola ve sağa olmak üzere dört yönde (komşu düğümler) gerçekleştirilir.

2.2 Arama Algoritmaları (algorithms.py)

Üç temel arama algoritması **algorithms.py** dosyasında uygulanmıştır:

1. **Genişlik Öncelikli Arama (BFS):** Bir kuyruk (queue) kullanarak tüm komşuları katman katman ziyaret eder ve bu sayede **en kısa (optimal) yolu** bulmayı garanti eder.
2. **Derinlik Öncelikli Arama (DFS):** Bir yığın (stack) kullanarak, olabildiğince derinlere inmeye çalışır. Çözümü hızlı bulabilir ancak **optimal (en kısa) yolu bulmayı garanti etmez**.

3. **A* Arama Algoritması (A* Search):** Maliyeti (cost_so_far, g(n)) ve sezgiseli (heuristic, h(n)) birleştirerek öncelikli kuyruk (heapq) kullanan bilgili bir arama yöntemidir.

   Sezgisel Fonksiyon: Bu projede **Manhattan Mesafesi** kullanılmıştır: h(n) = |x_{hedef} - x_n| + |y_{hedef} - y_n|. Bu sezgisel, labirent ortamında kabul edilebilir (admissible) bir sezgiseldir ve A*'ın optimal yolu bulmasını sağlar.

2.3 Görselleştirme

Tüm arama süreçleri, **visualizer.py** modülü kullanılarak matplotlib kütüphanesi aracılığıyla **adım adım** görselleştirilmiştir. Bu sayede, her bir algoritmanın labirentte izlediği arama yolu ve genişlettiği düğümler anlık olarak takip edilmiştir.

3. Test Senaryoları

Algoritmaları farklı koşullar altında karşılaştırmak amacıyla iki farklı labirent ve başlangıç/hedef çifti belirlenmiştir:

Senaryo	Labirent Dosyası	Başlangıç Koordinatı	Hedef Koordinatı	Senaryo Amacı
Senaryo 1	maze1.txt	(0, 0)	(6, 4)	Algoritmaların standart bir labirent üzerindeki genel performansını karşılaştırma.
Senaryo 2	maze2.txt	(1, 1)	(9, 9)	Daha karmaşık bir labirent yapısında (uzun dar koridorlar) algoritmaların verimlilik ve keşfetme stratejisi farklılıklarını test etme.

4. Sonuçlar, Karşılaştırmalar ve Değerlendirme

Tüm algoritmaların iki farklı senaryoda çalıştırılması sonucunda elde edilen metrikler aşağıdaki tabloda özetlenmiştir.

Algoritma (Labirent)	Başlangıç	Hedef	Yol Uzunluğu	Genişletilen Düğüm	Süre (ms)
BFS (maze1.txt)	(0, 0)	(6, 4)	35	57	8369.35
DFS (maze1.txt)	(0, 0)	(6, 4)	43	53	7667.84
A* (maze1.txt)	(0, 0)	(6, 4)	35	51	7263.68
BFS (maze2.txt)	(1, 1)	(9, 9)	17	43	6816.86
DFS (maze2.txt)	(1, 1)	(9, 9)	31	41	7492.49
A* (maze2.txt)	(1, 1)	(9, 9)	17	21	3887.27

4.1 Çözüm Maliyeti (Yol Uzunluğu) Karşılaştırması

- **Optimal Çözüm:** BFS ve A* algoritmaları, her iki senaryoda da en kısa yolu bulmuşlardır (S1: 35 adım; S2: 17 adım). Bu, her iki algoritmanın da **optimal (en iyi) çözümü bulma** özelliğini doğrular.
- **DFS Dezavantajı:** DFS algoritması her iki labirentte de daha uzun yollar bulmuştur (S1: 43 adım; S2: 31 adım). Bu durum, DFS'in kör bir şekilde derinlere inme eğilimi nedeniyle suboptimal çözümler üretebileceğini göstermektedir.

4.2 Verimlilik (Genişletilen Düğüm Sayısı ve Süre) Karşılaştırması

Kriter	En Verimli Algoritma	Açıklama
Genişletilen Düğüm (Verimlilik)	**A***	A*, her iki labirentte de en az düğümü genişleten algoritmadır (S1: 51, S2: 21). Bu, sezgisel (Manhattan Mesafesi) fonksiyonunun arama alanını etkin bir şekilde daralttığını kanıtlar.
Süre (ms)	**A***	A*, en az düğüm genişletme sayesinde, her iki senaryoda da en hızlı sonuca ulaşan algoritmadır (S2'de 3887.27 ms ile açık ara farkla öndedir).
DFS Yanıltması	DFS	DFS, BFS'ten daha az düğüm genişletebilir (S2'de 41 vs 43) ve daha kısa sürede çözümü bulabilir (S1'de 7667 ms vs 8369 ms), ancak bu avantajı daha uzun bir yol maliyeti pahasına elde etmiştir.

4.3 Değerlendirme

Algoritma	Avantaj	Dezavantaj
BFS	Optimal (en kısa) yolu bulmayı garantiler.	Arama alanını genişlettiği için A*'a göre daha fazla düğüm genişletir ve daha yavaştır.
DFS	Hafıza kullanımı genellikle düşüktür. Sığ bir çözüm varsa hızlı bulabilir.	Optimal yolu bulmayı garanti etmez ve gereksiz yere labirentin derinliklerine inerek zaman kaybedebilir.
A*	Hem optimal çözümü garantiler hem de sezgisel sayesinde en az düğümü genişleterek en verimli çalışır.	Sezgisel fonksiyonun kalitesine bağlıdır; kötü bir sezgisel performansı düşürebilir (Bu projede iyi çalışmıştır).

Sonuç: Simülatör sonuçlarına göre, verilen labirent ortamı için **A* Arama Algoritması**, hem çözüm kalitesi (optimal yol) hem de verimlilik (en az genişletilen düğüm ve en kısa süre) açısından en üstün performansı sergilemiştir. BFS, optimal yolu bulsa da, A* kadar verimli değildir. DFS ise hızlı olabilir ancak bulunan yolun kalitesi düşüktür.

5. Kaynakça

- [Proje Kodu] Labirent yapısı, algoritmalar ve görselleştirme implementasyonları.
- [Kütüphane] Python 3.x, Matplotlib (Görselleştirme), NumPy (Matris işlemleri), Tabulate (Tablo oluşturma).
- [Yararlanılan Dış Kaynaklar] [Varsa, kodun veya metodun alındığı makale, GitHub deposu veya web sitesi buraya yazılmalıdır.]