EGE UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

360 Artificial Intelligence Methods (3+0)

2023–2024 SPRING SEMESTER

PROJECT-1 REPORT

DELIVERY DATE

20/04/2024

PREPARED BY

05200000114, Ahmet Alper Akın

05210000213, Mehmet Berk Bozkurt

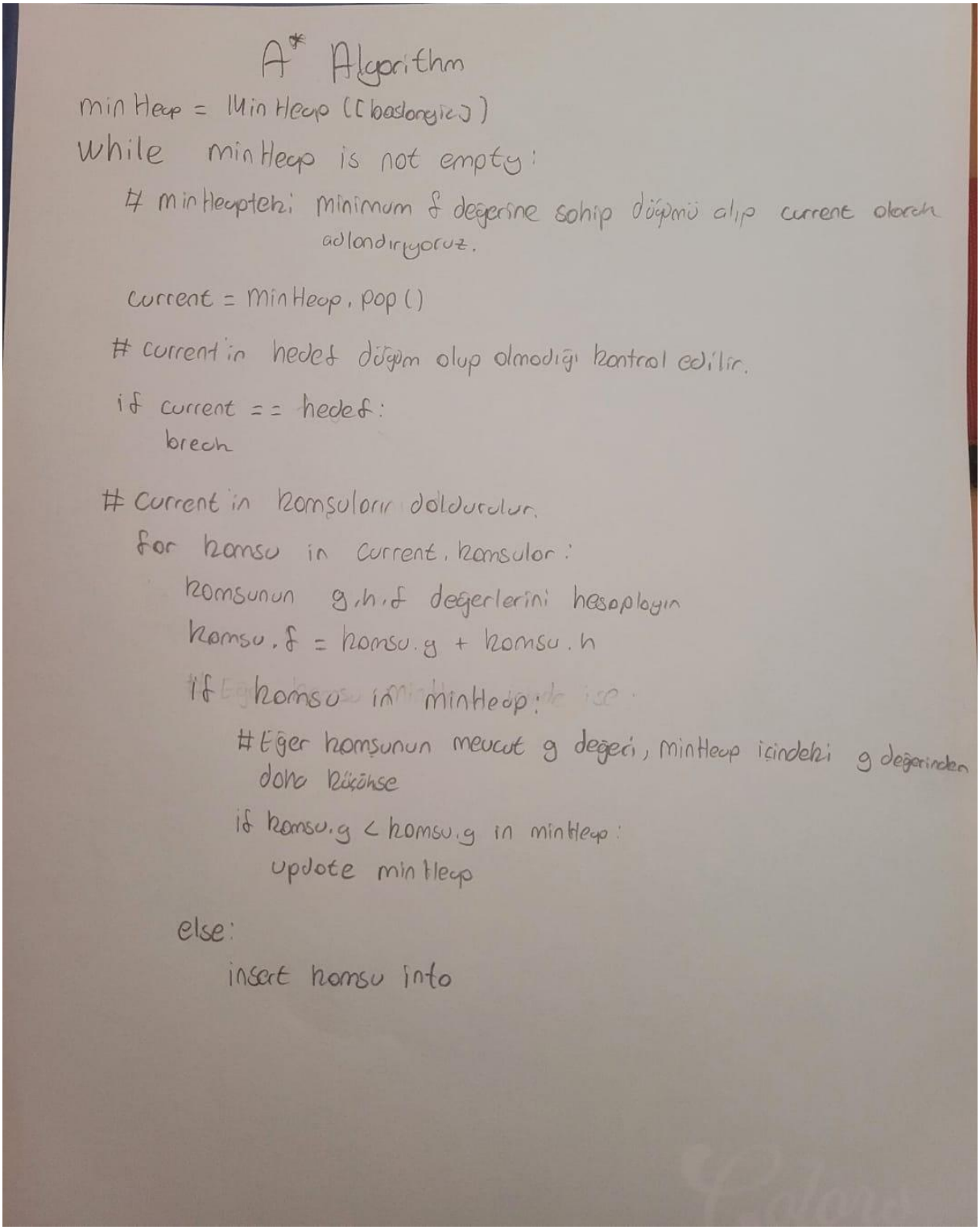
05200000057, Timurcan Yıldız

Index

1) Algorithms, Definitions, Comparison, Research and Comment

1.a Tabu Search and A* Algorithms

1.a.i Image of Tabu Search and A* Algorithms written on a paper (Scan/Photograph)



A* Algorithm

```
minHeap = MinHeap([başlangıç])
while minHeap is not empty:
    # minHeaptaki minimum f değerine sahip düğümü alıp current olarak
      adlandırıyoruz.
    current = minHeap.pop()
    # current'in hedef düğüm olup olmadığı kontrol edilir.
    if current == hedef:
        break
    # current'in komşuları doldurulur.
    for komsu in current.komsular:
        komsunun g,h,f değerlerini hesaplayın
        komsu.f = komsu.g + komsu.h
        if komsu in minHeap:
            # Eğer komşunun mevcut g değeri, minHeap içindeki g değerinden
              daha küçükse
            if komsu.g < komsu.g in minHeap:
                update minHeap
        else:
            insert komsu into
```

Tobu Search Algoritması

```
def tobu_arama (boslangic_cozumu, mohs_iterasyon , tobu_liste_boyutu):
    en_iyi_cozum = boslangic_cozumu
    mevcut_cozum = boslangic_cozumu
    tobu_listesi []
    for i in range (mohs_iterasyon):
        komsular = get_komsular (anevcut_cozum )
        en_iyi_komsu = None
        en_iyi_komsu_fitness = float ('inf')

        for komsu in komsular:
            if komsu not in tobu_listesi:
                komsu_uygunluk = uygunluk_fonksiyonu (komsu)
                if komsu_uygunluk < en_iyi_komsu_uygunluk:
                    en_iyi_komsu = komsu
                    en_iyi_komsu_uygunluk = komsu_uygunluk
        if en_iyi_komsu is None:
            break
        mevcut_cozum = en_iyi_komsu
        tobu_listesi.append (en_iyi_komsu)
        if len (tobu_listesi) > tobu_liste_boyutu:
            tobu_listesi.pop(0)
        if uygunluk_fonksiyonu (en_iyi_komsu) < uygunluk_fonksiyonu (en_iyi_cozum)

            en_iyi_cozum = en_iyi_komsu

return en_iyi_cozum
```

1.a.ii Explanation of Tabu Search algorithm

- It is a local search method.
- It improves iteratively.
- It works by improving local search and avoiding already visited points. This prevents the formation of cycles in the search space and allows for better solutions by avoiding local optima.

```python
# Amaç Fonksiyonu: Verilen bir çözümün uygunluğunu hesaplar
def amac_fonksiyonu(cozum):
    ## Burada, verilen probleme özgü olarak amaç fonksiyonumuzu tanımlıyoruz.
    ## Örneğin, toplama işlemi kullanılarak basit bir örnek sunulmuştur.
    return sum(cozum)  # Çözümün elemanlarının toplamını döndürür
```

First, we define an objective function that can vary for each problem. Finally, the tabu search algorithm will return the best value in this function.

```python
# Komşuluk Fonksiyonu: Verilen bir çözüme komşu olan çözümleri üretir
def komsu_uret(cozum):
    komsular = []
    for i in range(len(cozum)):
        for j in range(i + 1, len(cozum)):
            komsu = cozum[:]
            komsu[i], komsu[j] = komsu[j], komsu[i]  # i ve j indislerindeki elemanları yer değiştir
            komsular.append(komsu)
    return komsular
```

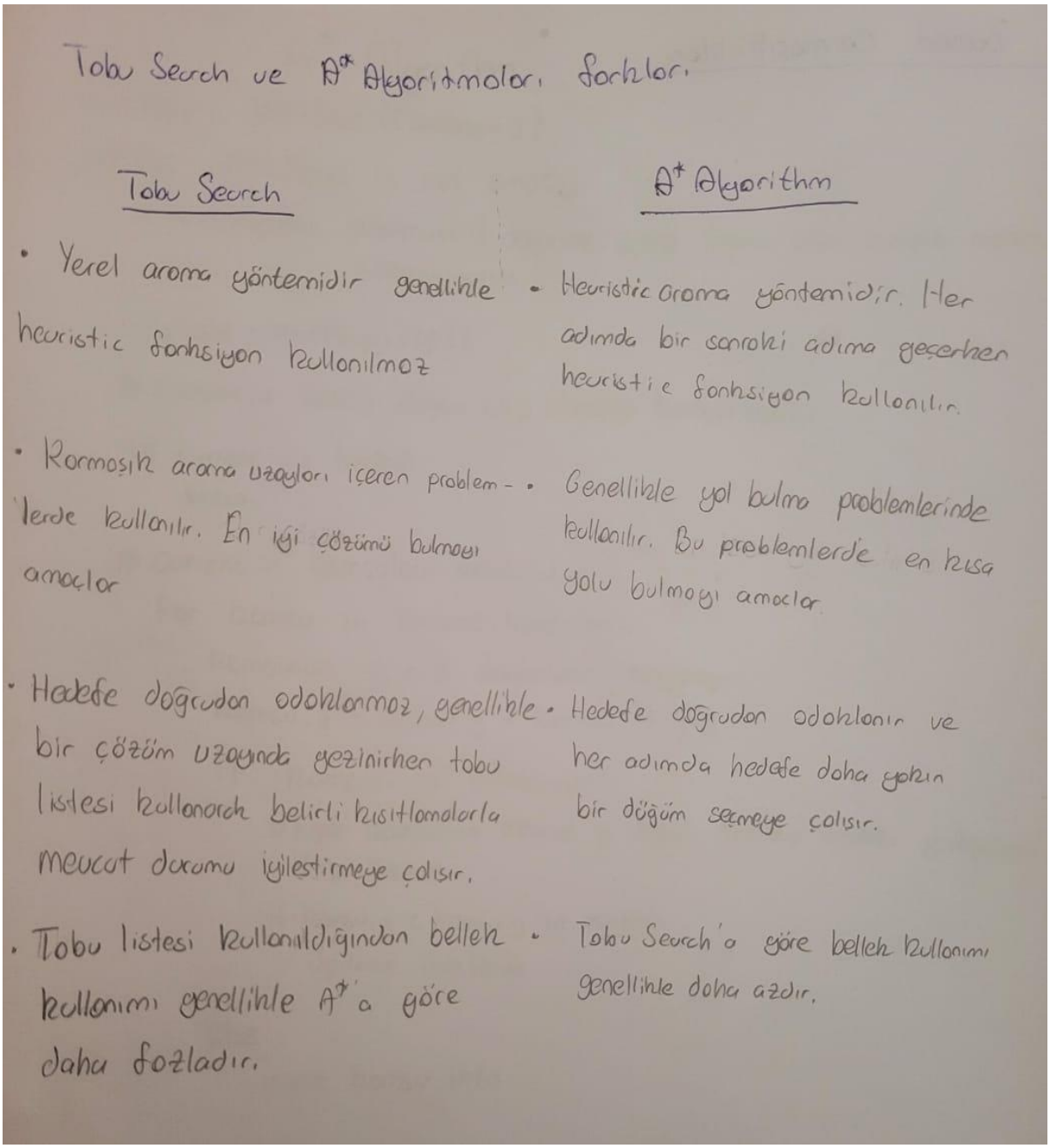
Here we have our function that generates the neighbors.

```python
# Tabu Arama Algoritması
def tabu_arama(baslangic_cozumu, maks_iterasyon, tabu_liste_boyutu):
    en_iyi_cozum = baslangic_cozumu
    mevcut_cozum = baslangic_cozumu
    tabu_listesi = []

    for _ in range(maks_iterasyon):
        komsular = komsu_uret(mevcut_cozum)
        en_iyi_komsu = None
        en_iyi_komsu_uygunluk = float('inf')

        # Komşu çözümler arasından en iyi olanı bul
        for komsu in komsular:
            if komsu not in tabu_listesi:
                komsu_uygunluk = amac_fonksiyonu(komsu)
                if komsu_uygunluk < en_iyi_komsu_uygunluk:
                    en_iyi_komsu = komsu
                    en_iyi_komsu_uygunluk = komsu_uygunluk

        if en_iyi_komsu is None:
            # Tabu listesinde yasaklanmamış komşu yoksa, aramayı sonlandır
            break

        # Yeni en iyi çözümü kabul et
        mevcut_cozum = en_iyi_komsu
        tabu_listesi.append(en_iyi_komsu)
        if len(tabu_listesi) > tabu_liste_boyutu:
            # Tabu listesi boyutunu kontrol et ve gerekiyorsa en eski girişi kaldır
            tabu_listesi.pop(0)

        # En iyi çözümü güncelle
        if amac_fonksiyonu(en_iyi_komsu) < amac_fonksiyonu(en_iyi_cozum):
            en_iyi_cozum = en_iyi_komsu


    return en_iyi_cozum
```

- Firstly, we define our initial solution as the best and current solution.
- We create a taboo list. This list keeps track of previously visited neighbors and ensures that we do not revisit them.
- The loop continues for a maximum number of iterations.
- In each iteration, the neighborhood structure of the current solution is examined.
- Then, the neighbor with the best fitness among the neighbors of the current solution, which is not in the taboo list, is selected.
- If we cannot find a neighbor not prohibited in the taboo list, we terminate the search.
- If we find the best neighbor, the current solution is updated as the best neighbor and added to the taboo list.
- If the taboo list exceeds the specified size, we remove the oldest added neighbor from the list.

- If our best neighbor is better than the best solution, this best solution is replaced by the best neighbor.
- When the loop ends, the best solution is returned.

1.a.iii Differences of the two algorithms



Tabu Search ve A* Algoritmaları farkları.

Tabu Search	A* Algorithm
• Yerel arama yöntemidir genellikle heuristic fonksiyon kullanılmaz	• Heuristic arama yöntemidir. Her adımda bir sonraki adıma geçerken heuristic fonksiyon kullanılır.
• Karmaşık arama uzayları içeren problemlerde kullanılır. En iyi çözümü bulmayı amaçlar	• Genellikle yol bulma problemlerinde kullanılır. Bu problemlerde en kısa yolu bulmayı amaçlar.
• Hedefe doğrudan odaklanmaz, genellikle bir çözüm uzayında gezinirken tabu listesi kullanarak belirli kısıtlamalarla mevcut durumu iyileştirmeye çalışır.	• Hedefe doğrudan odaklanır ve her adımda hedefe daha yakın bir düğüm seçmeye çalışır.
• Tabu listesi kullanıldığından bellek kullanımı genellikle A*'a göre daha fazladır.	• Tabu Search'a göre bellek kullanımı genellikle daha azdır.

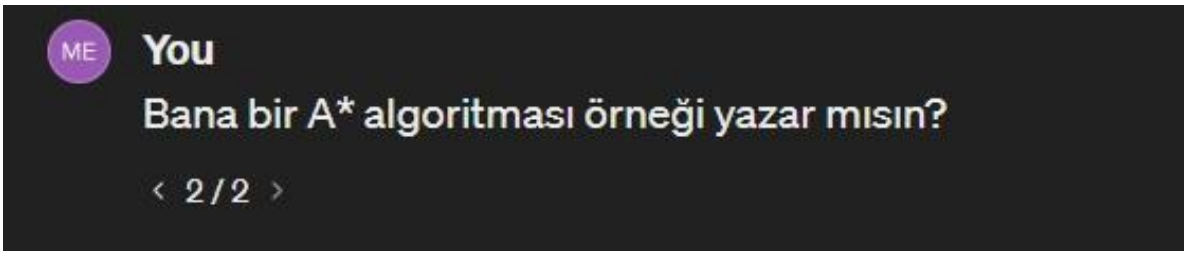
1.a.iv Time Complexities

A*: The time complexity of this algorithm varies from example to example. In our example, the worst-case time complexity is O(nlogn) and the best-case is O(1).

Tabu Search: The time complexity can vary depending on factors such as the size of the tabu list and the number of iterations. In our example, the worst-case time complexity is O(k.n^2) and the best-case is O(1), where K is the maximum number of iterations.

1.b A* and GenAI

1.b.i Making GenAI tool to write A* Code




```python
import heapq

class Node:
    def __init__(self, state, parent=None, g=0, h=0):
        self.state = state  # Durum
        self.parent = parent  # Ebeveyn düğüm
        self.g = g  # Başlangıç düğümünden bu düğüme olan toplam maliyet
        self.h = h  # Bu düğümden hedef düğüme olan tahmini maliyet

    def f(self):
        return self.g + self.h

class AStar:
    def __init__(self, start_state, goal_state, neighbors_fn, heuristic_fn):
        self.start_node = Node(start_state)
        self.goal_state = goal_state
        self.neighbors_fn = neighbors_fn
        self.heuristic_fn = heuristic_fn
```

```
    def search(self):
        open_list = []
        closed_set = set()

        heapq.heappush(open_list, (self.start_node.f(), id(self.start_node), self.sta

        while open_list:
            _, _, current_node = heapq.heappop(open_list)

            if current_node.state == self.goal_state:
                path = self._reconstruct_path(current_node)
                return path

            closed_set.add(current_node.state)

            for neighbor_state in self.neighbors_fn(current_node.state):
                if neighbor_state in closed_set:
                    continue

                g = current_node.g + 1  # Assuming uniform cost for simplicity
                h = self.heuristic_fn(neighbor_state, self.goal_state)
                neighbor_node = Node(neighbor_state, current_node, g, h)

                heapq.heappush(open_list, (neighbor_node.f(), id(neighbor_node), neig

        return None  # Hedef duruma ulaşılamadı
```

```
    def _reconstruct_path(self, node):
        path = []
        while node:
            path.append(node.state)
            node = node.parent
        return path[::-1]  # Ters çevir ve dön

# Örnek kullanım
def manhattan_distance(state, goal_state):
    x1, y1 = state
    x2, y2 = goal_state
    return abs(x1 - x2) + abs(y1 - y2)

def get_neighbors(state):
    x, y = state
    neighbors = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]  # Doğrusal hareket
    # İsteğe bağlı olarak durumun geçerli olup olmadığını kontrol edebilirsiniz
    return [(x, y) for x, y in neighbors if 0 <= x < 5 and 0 <= y < 5]

start_state = (0, 0)
goal_state = (4, 4)

astar = AStar(start_state, goal_state, get_neighbors, manhattan_distance)
path = astar.search()

if path:
    print("Yol bulundu:", path)
else:
    print("Hedefe ulaşılamadı.")
```

1.b.ii Execution of the code and test screenshots

```
ifical Intelligance/Proje 1/a_star_openai.py"
Yol bulundu: [(0, 0), (1, 0), (1, 1), (2, 1), (3, 1), (4, 1), (4, 2), (4, 3), (4, 4)]
```

1.b.iii Questions that asked to GenAI tool about details of the code and the answers

1-)

You
neden heapq kullandık?

ChatGPT
A* algoritması, genellikle bir öncelik kuyruğu (priority queue) kullanılarak uygulanır. Öncelik kuyruğu, düğümleri belirli bir önceliğe göre saklar ve en yüksek önceliğe sahip düğümü çıkarmak için hızlı erişim sağlar. Bu, A* algoritmasının her adımında en düşük maliyetli düğümü seçmek için önemlidir.

Python'da, `heapq` modülü, öncelik kuyruğu oluşturmak ve yönetmek için kullanılır. Bu modül, bir liste gibi davranan bir dizi işlev sağlar, ancak liste üzerinde çeşitli operasyonları çok daha hızlı bir şekilde gerçekleştirir. Ayrıca, bu modül, elemanlar sıralı bir şekilde eklendiğinde her zaman en küçük öğeyi hızlı bir şekilde bulmayı sağlayan bir min-heap (minimum heap) yapısını kullanır.

A* algoritması, her adımda açık listeyi güncellemek ve en düşük maliyetli düğümü seçmek için bir öncelik kuyruğu kullanır. Bu nedenle, `heapq` modülü, A* algoritmasının verimli bir şekilde uygulanmasını sağlar. Bu modül, öncelik kuyruğu işlevselliğini sağladığı için tercih edilir.

2-)

You
Yol bulundu: [(0, 0), (1, 0), (1, 1), (2, 1), (3, 1), (4, 1), (4, 2), (4, 3), (4, 4)] bu çıktıyı adım adım hedefe uzaklığı göstermemizi gösterecek şekilde düzenleyebilir misin?

< 2/2 >

```python
if path:
    print("Yol bulundu:")
    for i, state in enumerate(path):
        distance_to_goal = manhattan_distance(state, goal_state)
        print(f"Adım {i}: {state}, Hedefe Uzaklık: {distance_to_goal}")
else:
    print("Hedefe ulaşılamadı.")
```

Screenshots of new tests.

```
Yol bulundu:
Durum: (0, 0) Hedefe Mesafe: 0
Durum: (1, 0) Hedefe Mesafe: 7
Durum: (1, 1) Hedefe Mesafe: 6
Durum: (2, 1) Hedefe Mesafe: 5
Durum: (3, 1) Hedefe Mesafe: 4
Durum: (4, 1) Hedefe Mesafe: 3
Durum: (4, 2) Hedefe Mesafe: 2
Durum: (4, 3) Hedefe Mesafe: 1
Durum: (4, 4) Hedefe Mesafe: 0
```

3-)

You

Bu kodda closed_set ne işe yarıyor?

ChatGPT

`closed_set`, A* algoritması sırasında zaten kontrol edilmiş ve üzerinden geçilmiş olan düğümlerin bir kümesidir. Bu küme, algoritmanın tekrar aynı düğüme döngü içinde giderek zaman kaybetmesini önlemek için kullanılır.

A* algoritması, genellikle geniş bir arama yapar, bu da bir düğümü birden fazla kez inceleyebileceği anlamına gelir. Eğer bir düğüm zaten kapatılmışsa (yani, daha önce işlendi ve üzerinden geçildi), o zaman bu düğümü tekrar işlemeye gerek yoktur. Çünkü bu düğümün altında daha önce işlenen tüm komşuları zaten göz önünde bulundurmuşuzdur ve daha düşük maliyetli bir yol bulunamamıştır.

Bu nedenle, `closed_set` adı verilen bu küme, algoritmanın daha önce işlenmiş olan düğümleri hatırlamasını sağlar. Bir düğüm işlendikten sonra, bu düğüm `closed_set`'e eklenir ve daha sonra aynı düğüme tekrar ulaşılırsa göz ardı edilir.

Bu şekilde, algoritmanın daha hızlı çalışmasına ve daha verimli bir şekilde hedefe ulaşmasına yardımcı olur.

4-)

heapq.heappush(open_list, (self.start_node.f(), id(self.start_node), self.start_node)) burayı detaylı açıklayabilir misin=

ChatGPT

Evet, bu satır A* algoritmasında kullanılan öncelik kuyruğuna bir düğümün eklenmesini sağlar. Bir düğüm öncelik kuyruğuna eklenirken, o düğümün toplam maliyeti ve düğümün kendisi bir demet olarak eklenir.

İşte bu satırın detaylı açıklaması:

- `self.start_node.f()`: Başlangıç düğümünün toplam maliyetini hesaplayan fonksiyonu çağırır. `f()` metodu, bir düğümün başlangıç düğümünden bu düğüme olan toplam maliyetini hesaplar. Bu maliyet genellikle başlangıç düğümünden bu düğüme olan gerçek maliyet (g) ve bu düğümden hedef düğüme olan tahmini maliyet (h) toplamından oluşur. Bu toplam maliyet, A* algoritmasında düğümleri birbirinden ayırmak ve en düşük maliyetli olanı seçmek için kullanılır.
- `id(self.start_node)`: Düğümün benzersiz kimliğini alır. Bu, düğümleri öncelik kuyruğunda sıralamak için kullanılan ikinci bir kriterdir. Aynı maliyete sahip iki düğüm varsa, bunlar kimliklerine göre sıralanır. Bu, aynı maliyete sahip olan düğümleri birbirinden ayırmaya yardımcı olur ve kuyrukta tutulan düğümlerin benzersiz olmasını sağlar.
- `self.start_node`: Kendi başlangıç düğümünü ekler. Bu, gerçek düğümün kendisini eklemek anlamına gelir. Öncelik kuyruğuna eklenen demetin içinde düğümün kendisi de bulunur. Bu, sonradan düğüme erişme gereksinimi olduğunda kolaylık sağlar.

Sonuç olarak, bu satır, öncelik kuyruğuna bir düğüm eklerken, bu düğümün toplam maliyetini, benzersiz kimliğini ve kendisini içeren bir demet oluşturur. Bu, A* algoritmasının her adımında en düşük maliyetli düğümü seçmek için öncelik kuyruğunu kullanmasını sağlar.

1.b.iv Advantages, contributions of the tool and failures

Advantages:

• One of the most significant advantages, in my opinion, is that if we make a mistake in a project we're working on, it can most likely show us what the mistake is and its solution.

• If there is a term we don't know or a concept we want to learn, we can minimize the research time and get an answer.

• It is possible to develop any project from scratch with proper guidance and our own corrections.

Disadvantages:

- The number of incorrect responses can be significant. Therefore, we may need to verify the information from another source or correct the errors ourselves if we are knowledgeable about the subject.

- A reflex to constantly ask GenAI for everything can lead to laziness. Some things may be skipped without being fully understood.

1.c Definitions and Comparisons

1.c.i Ensemble Learning

Ensemble learning is a powerful technique in machine learning that combines the predictions of multiple models to enhance overall performance. By leveraging the collective knowledge of diverse models, ensemble learning can mitigate individual model biases and improve the accuracy and robustness of predictions.

Types of Ensemble Learning:

Bagging: This method trains multiple models on different subsets of the training data and averages their predictions. By reducing variance, bagging helps to prevent overfitting and improve generalization performance.

Boosting: Trains models sequentially, with each subsequent model focused on correcting the errors of its predecessors. Boosting reduces bias and enhances the ensemble's predictive power.

Stacking: Trains multiple models and combines their predictions using a meta-model. This approach allows for more complex interactions between models and can yield superior predictive accuracy.

Advantages of Ensemble Learning:

Improved accuracy: **By combining diverse models, ensemble learning can achieve better predictive accuracy than any individual model.**

Reduced variance: **By averaging the predictions of multiple models, ensemble learning can reduce the variance of the overall prediction, making it more reliable.**

Reduced bias: **By training models on different subsets of the data, ensemble learning can reduce the bias of the overall prediction.**

Robustness: **Ensemble learning is more robust to noise and outliers in the data than individual models.**

Ensemble learning applications:

Image classification: **Ensemble learning has been successfully applied to image classification tasks, such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).**

Natural language processing: **Ensemble learning is also used in natural language processing tasks, such as sentiment analysis and text classification.**

Fraud detection: **Ensemble learning can be used to detect fraudulent activities, such as credit card fraud and insurance** fraud.

1.c.ii LLM (Large Language Model)

LLMs are basically supercharged language models in the world of AI. They're trained on massive amounts of text data, which makes them really good at understanding and generating human language.

Here's the cool part: LLMs can do a bunch of things, like:

Generating text: They can write different kinds of creative content, translate languages, and even write different kinds of code.

Understanding text: They can analyze your writing and tell you things like sentiment (is it positive or negative?), or answer your questions in a comprehensive way.

Completing tasks: Imagine giving an LLM instructions and it finishing them for you, like summarizing a long document or writing different parts of an email.

How do LLMs work?

LLMs use a kind of neural network called a transformer. This thing is like a super-complex math machine that analyzes the relationships between words and sentences. The more data the LLM is trained on, the better it gets at understanding these connections.

Some things to keep in mind about LLMs:

They're still under development: Even though they're impressive, LLMs can still make mistakes. They might generate text that sounds reasonable but isn't quite right, or they might misunderstand your instructions.

Data bias: Since LLMs are trained on massive amounts of data, they can reflect the biases that exist in that data. It's important to be aware of this limitation.

1.c.iii Devin

Devin represents a cutting-edge advancement in the realm of large language models (LLMs), specifically tailored for natural language processing (NLP) tasks. Leveraging its substantial size and sophisticated transformer architecture, Devin exhibits exceptional proficiency in comprehending and producing human-like text across a wide spectrum of linguistic complexities.

With its extensive training on vast corpora of text data, Devin has mastered the nuances of language semantics and syntax, enabling it to excel in tasks such as text generation, sentiment analysis, language translation, and question answering. Its ability to capture intricate contextual dependencies and generate coherent responses has established Devin as a formidable contender in the field of NLP.

Researchers and developers rely on Devin's capabilities to tackle diverse challenges in NLP, from summarizing lengthy documents to facilitating multilingual communication. Its adaptability and performance make it a valuable asset for applications spanning content generation, information

retrieval, and beyond, paving the way for innovative advancements in natural language understanding and generation.

1.c.iv CycleGAN

CycleGAN stands as a pioneering deep learning architecture renowned for its proficiency in unpaired image-to-image translation tasks. Its distinctive feature lies in its capacity to learn mappings between two domains without relying on explicit pairings in the training data. This is made possible through the innovative use of cycle-consistency loss, which ensures that translated images maintain fidelity to their original counterparts. By enforcing this constraint, CycleGAN can seamlessly transform images from one domain to another while preserving crucial visual details, enabling applications ranging from artistic style transfer to domain adaptation in computer vision.

The versatility of CycleGAN extends across a spectrum of applications in image processing and computer vision. Whether it's transferring artistic styles between images, performing image colorization, or adapting images between different domains, such as day to night or photo to painting, CycleGAN offers a robust solution. Researchers and practitioners harness the capabilities of CycleGAN to explore novel avenues in image manipulation and domain adaptation, leveraging its ability to learn meaningful mappings between diverse image domains without the need for paired data. This opens up exciting opportunities for creative expression and practical image processing tasks, propelling the field of computer vision forward.

1.c.v GPT-4 vs Gemini vs Claude 3

Overview:

GPT-4 is a language model developed by OpenAI.

Gemini is a language model developed by Google AI.

Claude 3 is a language model developed by Anthropic.

Technical Specifications:

Feature	GPT-4	Gemini	Claude 3
Number of Parameters	175 billion	1.5 trillion	137 billion
Model Type	Transformer	Meena-T5	Transformer
Training Data	The Pile	The Pile, Google AI datasets	The Pile, web text

Performance:

Claude 3 outperforms GPT-4 and Gemini in most benchmark tests.

Claude 3 is better at understanding and responding to complex queries.

Gemini is faster and uses fewer resources than GPT-4.

Use Cases:

GPT-4 can be used for various tasks such as text generation, translation, and code writing.

Gemini is used in Google AI's internal tools and products.

Claude 3 can be used for research and development purposes.

Strengths:

GPT-4: Versatile, wide range of capabilities.

Gemini: Fast, resource-efficient.

Claude 3: Powerful, good at understanding and responding to complex queries.

Weaknesses:

GPT-4: Can sometimes produce inconsistent and incorrect responses.

Gemini: Limited use outside of Google AI's internal tools and products.

Claude 3: Still under development, not fully tested.

Which One is Better?

It depends on your needs and priorities.

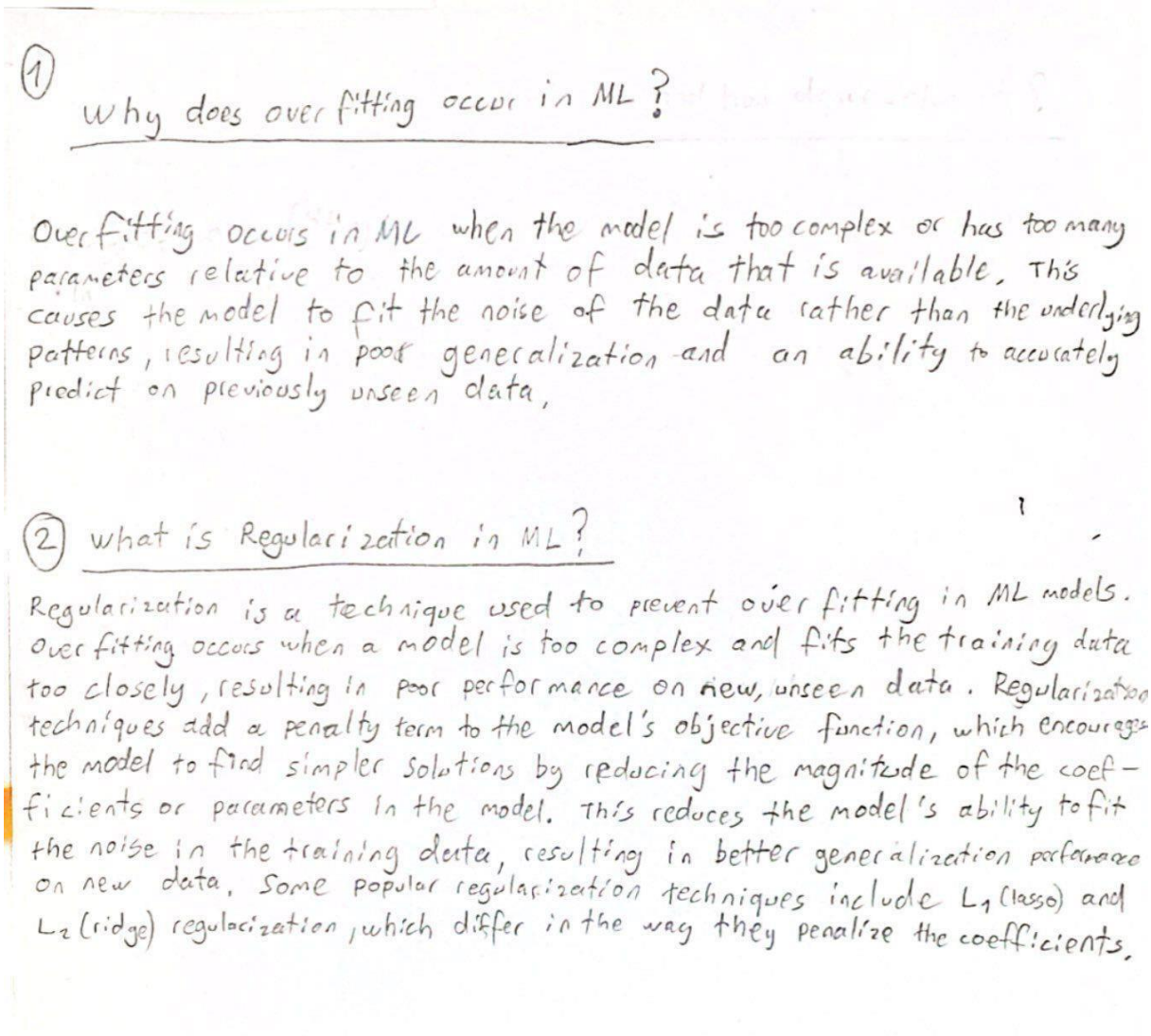
If you are looking for a versatile language model, GPT-4 could be a good choice.

If you are looking for a fast and resource-efficient language model, Gemini could be a good choice.

If you are looking for a powerful language model that is good at understanding and responding to complex queries, Claude 3 could be a good choice.

1.d Machine Learning Interview Questions

1.d.i Interview questions and answers that may be asked when entering a company working in the field of Machine Learning (Scan/Photo)



① Why does overfitting occur in ML?

Overfitting occurs in ML when the model is too complex or has too many parameters relative to the amount of data that is available. This causes the model to fit the noise of the data rather than the underlying patterns, resulting in poor generalization and an ability to accurately predict on previously unseen data.

② What is Regularization in ML?

Regularization is a technique used to prevent over fitting in ML models. Overfitting occurs when a model is too complex and fits the training data too closely, resulting in poor performance on new, unseen data. Regularization techniques add a penalty term to the model's objective function, which encourages the model to find simpler solutions by reducing the magnitude of the coefficients or parameters in the model. This reduces the model's ability to fit the noise in the training data, resulting in better generalization performance on new data. Some popular regularization techniques include L_1 (lasso) and L_2 (ridge) regularization, which differ in the way they penalize the coefficients.

1.d.ii Asking the same questions to ChatGPT and Gemini tools and grading the answers

Question 1: Why does overfitting occur in ML

Gemini >> Overfitting in Machine Learning happens when a model becomes too focused on the specific details and noise present in the training data, hindering its ability to perform well on new, unseen data. It's like memorizing every detail on a practice test instead of understanding the underlying concepts. There are a few key reasons why overfitting occurs:

Insufficient Training Data: If a model doesn't have enough data to learn from, it might latch onto random fluctuations or quirks in the training data instead of capturing the general patterns. Imagine trying to learn a language from just a few sentences - you might pick up some specific phrases but wouldn't understand the broader grammar and vocabulary.

Model Complexity: A model with too many parameters or features is like having a very elaborate recipe with dozens of ingredients. While it might perfectly recreate the training dish (data), it becomes overly sensitive to minor variations and struggles to adapt to new ingredients (unseen data).

Imbalanced Training Data: If your training data is skewed heavily towards one class or category, the model can become overly specialized in predicting that dominant class. It's like training for a race with only left turns - you might become very good at those but struggle with anything else.

These factors essentially make the model too good at fitting the training data, losing sight of the bigger picture and generalizability required for real-world performance.

chatGPT >> Overfitting occurs in machine learning when a model learns to perform well on the training data but fails to generalize to unseen data or new examples. It happens due to the model learning from noise or irrelevant patterns in the training data rather than capturing the underlying relationships that generalize well to new data. Several reasons contribute to overfitting:

1.Complexity of the Model: If the model is too complex relative to the amount of training data available, it can learn to memorize the training examples rather than learning the underlying patterns. This leads to poor generalization to new data.

2.Insufficient Training Data: When the amount of training data is limited, there's a higher risk of overfitting. With fewer examples, the model might capture noise as if it were signal, leading to poor generalization.

3.Noise in the Data: If the training data contains noise or irrelevant features, the model might learn to fit this noise rather than the underlying patterns, resulting in overfitting.

4.Inadequate Regularization: Regularization techniques such as L1 or L2 regularization are used to prevent overfitting by adding penalties to the model's loss function. If regularization is too weak or not applied effectively, the model may still overfit.

5.Data Leakage: Data leakage occurs when information from the test or validation set inadvertently influences the training process. This can lead to overly optimistic performance estimates during training and ultimately result in overfitting to the validation set.

6.Biased Sampling: If the training data is not representative of the population the model will be applied to, it might learn patterns that are specific to the training set but do not generalize well.

To mitigate overfitting, techniques such as cross-validation, regularization, feature selection, early stopping, and using more data can be employed. These techniques help in building models that generalize well to unseen data, thus reducing the risk of overfitting.

Grading :

Gemini's Answer: Gemini provides a comprehensive explanation of overfitting in machine learning, covering key reasons why it occurs along with apt analogies to illustrate each point. The response is clear, well-structured, and covers various aspects of overfitting.

I would rate Gemini's answer as a solid 9 out of 10.

GPT's Answer: GPT provide a detailed explanation of overfitting, covering reasons such as model complexity, insufficient training data, noise in the data, inadequate regularization, data leakage, and biased sampling. Each reason is explained concisely, and GPT suggest techniques to mitigate overfitting effectively.

I would rate GPT's answer as an 8.5 out of 10.

Question 2: What is Regularization in ML ?

Gemini >> Regularization in machine learning is a set of techniques used to prevent overfitting. Overfitting, as you know, occurs when a model memorizes the training data too well, losing its ability to generalize to unseen data.

Regularization works by adding a penalty term to the model's loss function. This penalty term discourages the model from learning overly complex patterns in the training data, forcing it to focus on capturing the underlying generalizable trends.

Here are some common types of regularization techniques:

L1 regularization (Lasso regularization): This technique adds the absolute value of the model's coefficients to the loss function. This tends to shrink some coefficients towards zero, effectively removing them from the model and reducing its complexity.

L2 regularization (Ridge regression): This technique adds the square of the model's coefficients to the loss function. This shrinks all the coefficients towards zero, but to a lesser extent than L1 regularization. It helps prevent large coefficient values that could contribute to overfitting.

Dropout: This technique is commonly used in neural networks. During training, a random subset of neurons is dropped from the network. This prevents the network from becoming too reliant on any specific features or neurons, promoting better generalization.

Regularization introduces a trade-off between the model's fit to the training data and its ability to generalize to unseen data. By adjusting the strength of the regularization penalty, you can find a balance that optimizes the model's performance.

Here are some benefits of using regularization:

Reduces overfitting: This is the primary benefit, leading to better performance on unseen data.

Improves model interpretability: By shrinking coefficients towards zero, L1 regularization can help identify features that are less important for the model's predictions.

However, there are also some potential drawbacks:

Increased bias: Regularization can introduce some bias into the model, meaning it might underfit the training data slightly.

Tuning hyperparameter: The strength of the regularization penalty is a hyperparameter that needs to be tuned to find the optimal value.

Overall, regularization is a powerful tool for improving the generalization performance of machine learning models. By understanding its principles and different techniques, you can effectively combat overfitting and build robust models.

chatGPT >> Regularization in machine learning is a technique used to prevent overfitting by adding a penalty term to the model's loss function. The objective of regularization is to encourage the model to learn simpler patterns that generalize well to unseen data rather than fitting the training data too closely.

There are primarily two types of regularization commonly used in machine learning:

1.L1 Regularization (Lasso Regression): In L1 regularization, also known as Lasso Regression, the penalty term added to the loss function is the absolute value of the coefficients of the model multiplied by a regularization parameter (λ). Mathematically, the L1 penalty term can be represented as $\lambda * \sum|\beta i|$, where βi represents the coefficients of the model. L1 regularization encourages sparsity in the model by driving some coefficients to zero, effectively performing feature selection.

2.L2 Regularization (Ridge Regression): In L2 regularization, also known as Ridge Regression, the penalty term added to the loss function is the squared magnitude of the coefficients of the model multiplied by a regularization parameter (λ). Mathematically, the L2 penalty term can be represented as $\lambda * \sum(\beta i^2)$, where βi represents the coefficients of the model. L2 regularization penalizes large coefficients, leading to smoother model weights and reducing the impact of outliers in the training data.

Both L1 and L2 regularization techniques help in reducing overfitting by penalizing overly complex models. The choice between L1 and L2 regularization depends on the specific characteristics of the dataset and the desired properties of the model. Additionally, the regularization parameter (λ) controls the strength of regularization, with larger values of λ resulting in stronger regularization and potentially simpler models.

Regularization is an essential tool in the machine learning toolbox, particularly when dealing with high-dimensional data or limited training samples, as it helps in building models that generalize well to unseen data.

Grading :

Gemini's Answer: Gemini provides a thorough explanation of regularization in machine learning, covering its purpose, techniques (L1 and L2 regularization, as well as dropout), benefits, and potential drawbacks. The explanation is clear, well-structured, and covers various aspects of regularization comprehensively.

I would rate Gemini's answer as a solid 9.5 out of 10.

GPT Answer: GPT provide a detailed explanation of regularization, covering its purpose, types (L1 and L2 regularization), mathematical representations, and how each technique helps in preventing overfitting. The explanation is clear and concise, focusing on the fundamental principles of regularization.

I would rate GPT's answer as a 9 out of 10.

2) Problem Solving and Coding

2.a Solving the Eight Queen Problem with Hill Climbing / Solving the Bridge and Torch Problem with A*

2.a.i Codes and Outputs

```
from random import randint
import time

N = 8
randomRestartCounter = 0;
changeCounter = 0;

#2 boyutlu bir satranç tahtası üzerinde rastgele bir state oluşturulur
def configureRandomly(board, state):

    # Iterating through the
    # column indices
    for i in range(N):

        #Vezirin kaçıncı satıra yerleştirileceğini bulmak için tam sayı bir index
rastgele alınır
        state[i] = randint(0, 100000) % N;
        #Rastgele alınan satırın i. elemanına vezir yerleştirilir
        board[state[i]][i] = 1;

#2 boyutlu bir array olan satranç tahtasını yazdırmak için kullanılan fonksiyon
def printBoard(board):

    for i in range(N):
        print(*board[i])

# Satranç tahtasındaki vezirlerin kaçıncı satırın (elemanın değeri) hangi sütununa
(index) konulduğunu gösteren durumu yazdırır
def printState( state):
    print(*state)

#İki state'in birbirinin aynısı olup olmadığını anlamamıza yarayan fonksiyon
def compareStates(state1, state2):

    for i in range(N):
        if (state1[i] != state2[i]):
            return False;

    return True;

# Oluşturulan tahtanın tüm kutularına aynı değeri verir
```

```
def fill(board, value):

    for i in range(N):
        for j in range(N):
            board[i][j] = value;

# Vezirlerin yerleştirilmiş olduğu bir satranç tahtasında kaç çift taşın birbirini yediği hesaplanır
def calculateObjective( board, state):

    #Bir satranç tahtasındaki vezirlerin saldırı patikasının üzerinde kaç tane taş olduğunu sayan sayaç
    attacking = 0;

    # For döngüsü ile satranç tahtasının üzerinde taş olan kareleri dolaşılır
    for i in range(N):

        # Her state[i]'nci satırda vezir, i'nci sütuna yerleştirilir.

        # Taşın solunda herhangi bir taş olup olmadığını kontrol etmek için satır aynı tutulur, sütun indeksi 0 oluncaya yada bir taşla karşılaşılıncaya kadar bir azaltılır
        row = state[i]
        col = i - 1;
        while (col >= 0 and board[row][col] != 1) :
            col -= 1

        #Bir taşla karşılaşıldığında sayaç bir arttırılır
        if (col >= 0 and board[row][col] == 1) :
            attacking += 1;

        # Taşın sağında herhangi bir taş olup olmadığını kontrol etmek için satır aynı tutulur, sütun indeksi "eleman sayısı - 1" oluncaya yada bir taşla karşılaşılıncaya kadar bir arttırılır
        row = state[i]
        col = i + 1;
        while (col < N and board[row][col] != 1):
            col += 1;
        #Bir taşla karşılaşıldığında sayaç bir arttırılır
        if (col < N and board[row][col] == 1) :
            attacking += 1;

        # Taşın sol üst çaprazında herhangi bir taş olup olmadığını kontrol etmek için satır da sütun da 0 oluncaya yada bir taşla karşılaşılıncaya kadar bir azaltılır
        row = state[i] - 1
        col = i - 1;
        while (col >= 0 and row >= 0 and board[row][col] != 1) :
            col-= 1;
            row-= 1;
```

```
        #Bir taşla karşılaşıldığında sayaç bir arttırılır
        if (col >= 0 and row >= 0  and board[row][col] == 1) :
            attacking+= 1;

        # Taşın sağ alt çaprazında herhangi bir taş olup olmadığını kontrol etmek
için satır da sütun da "eleman sayısı - 1" oluncaya yada bir taşla karşılaşılıncaya
kadar bir arttırılır
        row = state[i] + 1
        col = i + 1;
        while (col < N and row < N  and board[row][col] != 1) :
            col+= 1;
            row+= 1;
        #Bir taşla karşılaşıldığında sayaç bir arttırılır
        if (col < N and row < N and board[row][col] == 1) :
            attacking += 1;

        # Taşın sol alt çaprazında herhangi bir taş olup olmadığını kontrol etmek
için satır bir arttırılır, sütun bir eksiltilir
        row = state[i] + 1
        col = i - 1;
        while (col >= 0 and row < N  and board[row][col] != 1) :
            col -= 1;
            row += 1;
        #Bir taşla karşılaşıldığında sayaç bir arttırılır
        if (col >= 0 and row < N and board[row][col] == 1) :
            attacking += 1;

        # Taşın sağ üst çaprazında herhangi bir taş olup olmadığını kontrol etmek
için satır bir azaltılır, sütun bir arttırılır
        row = state[i] - 1
        col = i + 1;
        while (col < N and row >= 0  and board[row][col] != 1) :
            col += 1;
            row -= 1;
        #Bir taşla karşılaşıldığında sayaç bir arttırılır
        if (col < N and row >= 0 and board[row][col] == 1) :
            attacking += 1;

    # Bir taş başka bir taş ile kesiştiğinde diğer taş da mevcut taşla kesiceği için
aynı kesişimleri iki kez almamak için sonuç ikiye bölünür
    return int(attacking / 2);

# Durumu verilmiş bir satranç tahtasını oluşturulmak için kullanılan fonksiyon
def generateBoard( board, state):
    fill(board, 0);
    for i in range(N):
        board[state[i]][i] = 1;

# Bir durumu başka bir duruma kopyalayan fonksiyon
def copyState( state1, state2):
```

```
    for i in range(N):
        state1[i] = state2[i];

# Bir durumdan sonra gelen komşu durumu hesaplayan fonksiyon
def getNeighbour(board, state):

    # Elde edebileceğimiz en iyi komşuu elde etmek istediğimiz için optimal state ve
optimal board oluşturulur
    opBoard = [[0 for _ in range(N)] for _ in range(N)]
    opState = [0 for _ in range(N)]

    #Optimal state'e ve optimal board'a mevcut durum ve mevcut board kopyalanır
    copyState(opState, state);
    generateBoard(opBoard, opState);

    #Mevcut boardın değeri hesaplanır
    opObjective  = calculateObjective(opBoard, opState);

    # Komşu durum ve komşu board oluşturulur
    NeighbourBoard = [[0 for _ in range(N)] for _ in range(N)]
    NeighbourState = [0 for _ in range(N)]

    #Bir soraki gelecek olan durum ve board, mevcut durum ve board üzerinden
hesaplanılacağı için mevcut durum ve board komşu durum ve board'a kopyalanır.
    copyState(NeighbourState, state);
    generateBoard(NeighbourBoard, NeighbourState);

    # Mevcut board'ın tüm kareleri üzerinde deneme yanılma yoluyla testler yaparak
daha optimal bir durum/board bulunması hedeflenir
    for i in range(N):
        for j in range(N):

            # Eğer mevcut board'da taşın konulduğu yer ile döngüde üzerinde
dolaştığımız kare aynı değil ise
            if (j != state[i]) :

                #Taşın mevcut konumu yeni konum ile değiştirilir
                NeighbourState[i] = j;
                NeighbourBoard[NeighbourState[i]][i] = 1;
                NeighbourBoard[state[i]][i] = 0;

                # Yeni oluşan board'un (komşunun) değeri hesaplanır
                temp = calculateObjective( NeighbourBoard, NeighbourState);


                #Eğer komşu durumda birbirini yiyen taş sayısı daha az ise yeni
optimal durum komşu durum olur
                if (temp < opObjective) :
                    opObjective = temp;
```

```
                    copyState(opState, NeighbourState);
                    generateBoard(opBoard, opState);

                # Yapılan değişiklik geri alınarak komşu durum mevcut duruma geri
getirilir.Amaç bir adım uzakliktaki komşular içerisinden en iyisini bulmaktır.
                NeighbourBoard[NeighbourState[i]][i] = 0;
                NeighbourState[i] = state[i];
                NeighbourBoard[state[i]][i] = 1;

    # Optimal komşu durum,mevcut durum olarak ayarlanır
    copyState(state, opState);
    fill(board, 0);
    generateBoard(board, state);

def hillClimbing(board, state):

    global randomRestartCounter
    global changeCounter

    # Komşu board ve state oluşturulur

    neighbourBoard = [[0 for _ in range(N)] for _ in range(N)]
    neighbourState = [0 for _ in range(N)]

    copyState(neighbourState, state);
    generateBoard(neighbourBoard, neighbourState);

    print("-----------------------------------------------")
    printBoard(board);

    while True:

        #Optimal komşu, mevcut state üzerinden elde edileceği için komşu board ve
state'e mevcut state ve board kopyalanır
        copyState(state, neighbourState);
        generateBoard(board, state);

        # Optimal komşu elde edilir

        getNeighbour(neighbourBoard, neighbourState);

        #Eğer komşu durum ile mevcut durum birbirinin aynısı ise bu mevcut durumdan
daha optimal bir durum olmadığı yani bir çukurda(değeri en düşük state'i aradığımız
için) olduğumuz anlamına gelir.
        if (compareStates(state, neighbourState)) :

            if(calculateObjective(board,state) == 0) :#Eğer state'in değeri 0 ise bu
optimal değeri bulduğumuz anlamına gelir
                printBoard(board);
                break;
```

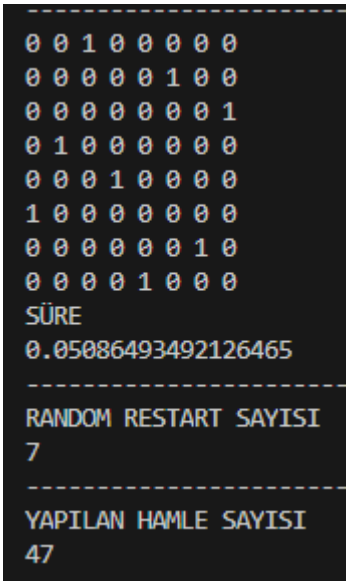
```
            else:#Eğer state'in değeri 0 değil ise local minimumda sıkışıldığı
anlamına gelir ve random restart yapılır
                configureRandomly(neighbourBoard,neighbourState);
                changeCounter += calculateDifference(board,neighbourBoard);
                randomRestartCounter +=1
                print("RANDOM RESTART YAPILDI");
         #Eğer komşu durum ile mevcut durumdan birbirinden farklı ama değerleri aynı
ise bu bir plato/shoulder üzerindedir.Random restart yapılır
        elif (calculateObjective(board, state) == calculateObjective(
neighbourBoard,neighbourState)):

            configureRandomly(neighbourBoard,neighbourState);
            changeCounter += calculateDifference(board,neighbourBoard);
            randomRestartCounter += 1;
            print("RANDOM RESTART YAPILDI");

        else:#Mevcut durum ile komşu durum birbirinden farklıdır ve komşu durum daha
optimaldir.Bir sonraki döngünün başında mevcut durum şuanki komşu durum olarak
ayarlanır
            changeCounter += 1;

        print("-----------------------------------------")
        printBoard(neighbourBoard);
        print("-----------------------------------------")

#İki board'ın birbiriyle aynı olup olmadığı anlaşılır
def calculateDifference(board,newBoard):
    difference = 0;
    for i in range(N):

        for j in range(N):

            if(board[i][j]!=newBoard[i][j]):

                difference += 1;

    return int(difference/2);

#8 Queens probleminin ne kadar sürede çözüleceğini bulmak için başlatılma zamanı
alınır.
startTime = time.time()

#Bir board ve state oluşturulur
state = [0] * N
board = [[0 for _ in range(N)] for _ in range(N)]

# Oluşturulan state ve board'dan rastgele bir başlangıç durumu ve board'ı elde
edilir
configureRandomly(board, state);
```

```
# Hill climbing ile problem çözdürülür
hillClimbing(board, state);
#Executionun sonlanma zamanı alınır
endTime = time.time()
#Sonlanma zamanından başlama zamanı çıkartılarak execution süresi bulunur
duration = endTime - startTime;

print("SÜRE")
print(duration)
print("-----------------------------------------------------------")
print("RANDOM RESTART SAYISI")
print(randomRestartCounter)
print("-----------------------------------------------------------")
print("YAPILAN HAMLE SAYISI")
print(changeCounter)
print("-----------------------------------------------------------")
```

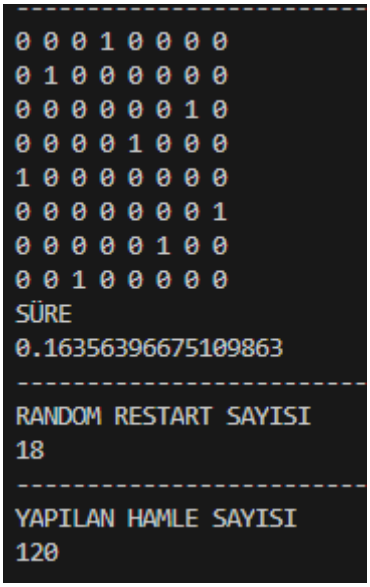
Run No	Number of relocations	Number of Random restart	Execution Time
1	47	7	0.05086493492126465
2	120	18	0.16356396675109863
3	4	0	0.008975982666015625
4	51	7	0.052858591079711914
5	23	3	0.027924299240112305
6	26	4	0.02892327308654785
7	12	1	0.015958070755004883
8	46	7	0.04787158966064453
9	69	10	0.0747995376586914
10	46	7	0.04584550857543945
11	60	8	0.06682038307189941
12	72	10	0.07878994941711426
13	52	7	0.05241250991821289
14	37	5	0.046874046325683594
15	25	3	0.02593207359313965

Table with number of relocations, number of random restarts and execution times

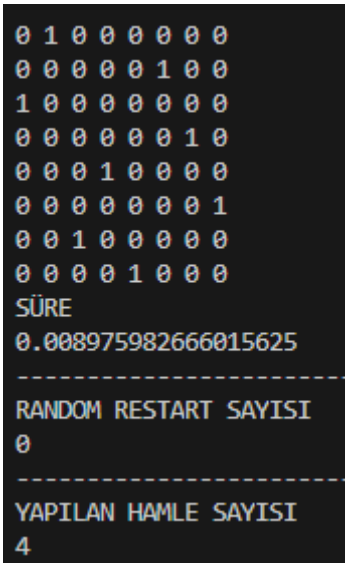
Run No:1



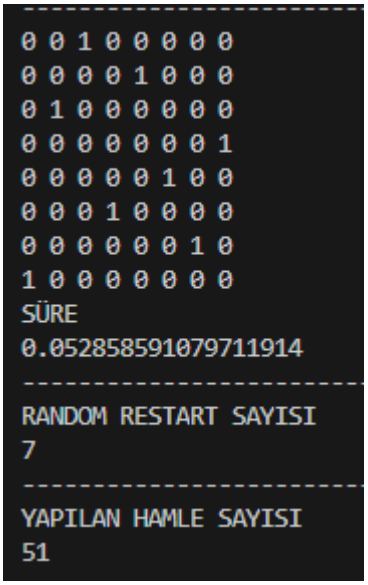
Run No:2



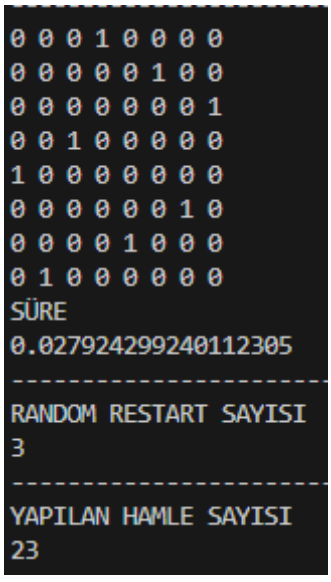
Run No:3



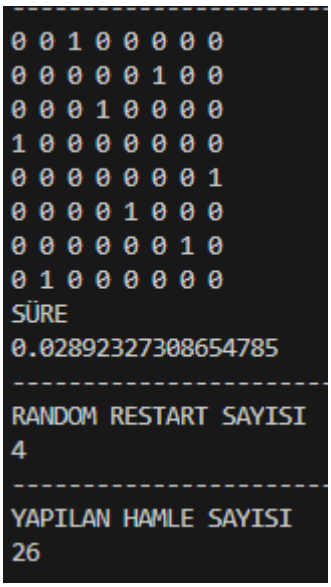
Run No:4



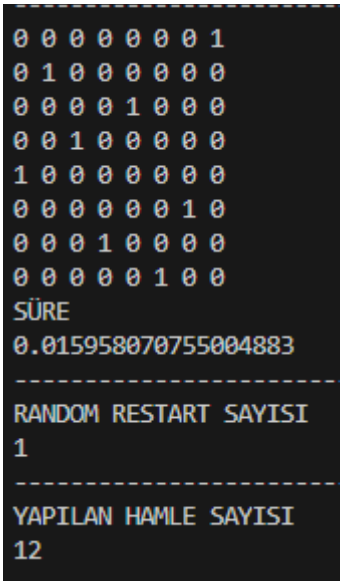
Run No:5



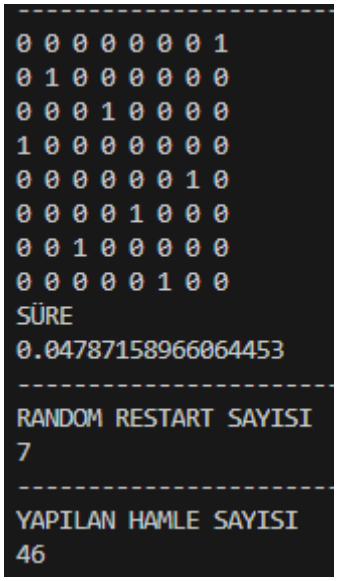
Run No:6



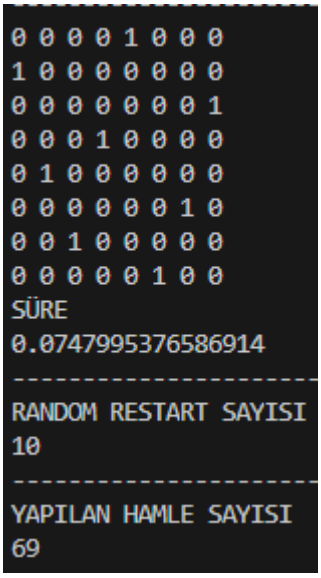
Run No:7



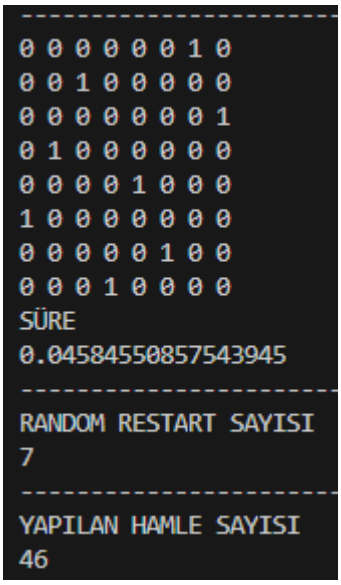
Run No:8



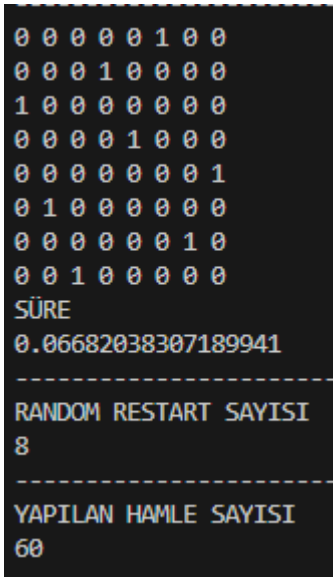
Run No:9



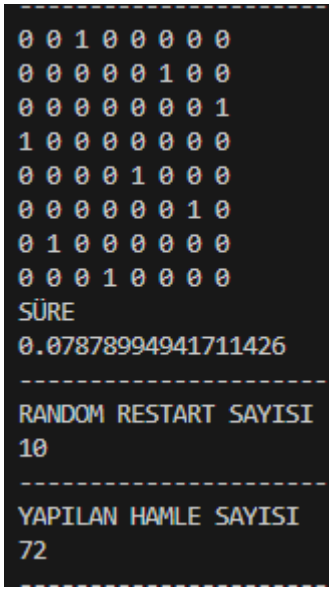
Run No:10



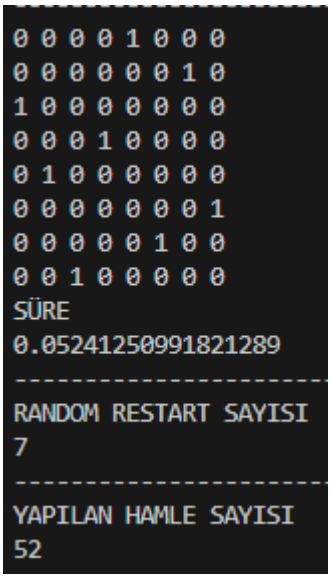
Run No:11



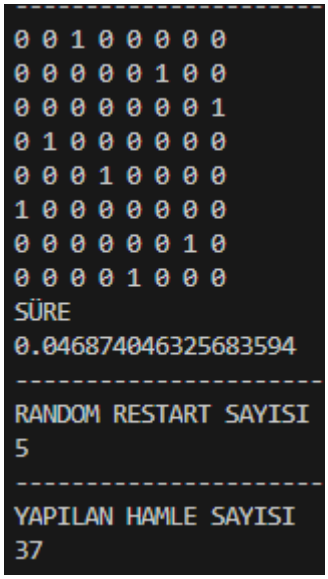
Run No:12



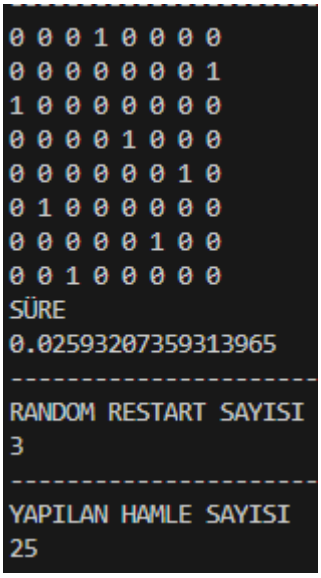
Run No:13



Run No:14



Run No:15



2.a.ii What is Stochastic Hill Climbing? Explain which part of the Hill Climbing source code needs to be changed.

Stochastic Hill Climbing is a Hill Climbing method unlike normal Hill Climbing, which doesn't directly move to the most optimal neighbor, but randomly moves to states that are better than the current one. For this Hill Climbing code to transform into a Stochastic Hill Climbing algorithm, the optimal neighbors found in the "getNeighbour" function should be stored in an array, and then a random neighbor should be selected from this array

3) Password Breaking with Genetic Algorithms

3.a Codes

```
public class GeneticAlgorithm
{
    // Bir populasyonda  bulunan kromozom sayısı
    private const int POPULATION_SIZE = 100;

    // Kullanılabilir genler
    private const string GENES = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP" +
                                 "QRSTUVWXYZ 1234567890, .-;:_!\"#%&/()=?@${[]}";

    // Bulunup kırılmak için şifre belirtilir
    private const string KEY = "(Generative AI) ";

    // Mutasyon, Çaprazlama vb. fonksiyonlarda kullanmak üzere Random bileşeni eklenilir
    private static readonly Random random = new Random();

    // Başlangıç ve bitiş sayıları verilmiş olan ardışık sayıların içerisinden rastgele bir
    // integer döndürecek fonksiyon. Rastgele index elde etmek için kullanılır.
    1 usage
    private static int RandomNum(int start, int end)
    {
        return random.Next(start, end + 1);
    }
```

```
// Gen dizisi; aranan şifreye daha yakın olan, yani daha çok umut vaad eden kromozomlara
// çaprazlama işleminde daha çok rastlanabilmesi için kullanılan bir fonksiyondur.
// Amaç daha iyi olan kromozomlar üzerinden ilerleyerek sonuca yaklaşmaktır
2 usages
private static int genomeRoulette(int start, int end)
{
    double p = random.NextDouble();

    //Jenerasyon Fitness değerlerine göre sıralandığı için istenen sonuca
    //daha yakın olan kromozomlar ilk kısımlarda yer alacaktır

    if (p < 0.5)
    {
        return random.Next(start, (start + end) / 2);

    }
    else if (p < 0.8)
    {
        return random.Next(start, end + 1);
    }
    else
    {
        return random.Next((start + end) / 2, end + 1);
    }
}
```

```
// Kromozomları fitness değerlerine göre sıralayan fonksiyon
1 usage
private class FitnessComparer : IComparer<Cromosome>
{
    public int Compare(Cromosome ind1, Cromosome ind2)
    {
        return ind1.FitnessValue.CompareTo(ind2.FitnessValue);
    }
}

// Mutation fonksiyonu ile rastgele bir şekilde genom oluşturan fonksiyon
1 usage
private static string CreateCromosome()
{
    int len = KEY.Length;
    char[] genome = new char[len];
    for (int i = 0; i < len; i++)
    {
        genome[i] = Mutation();
    }
    return new string(genome);
}
```

```
// Rastgele bir şekilde gen değiştirmemizi yarayan fonksiyon
2 usages
private static char Mutation()
{
    int len = GENES.Length;
    int r = RandomNum(0, len - 1);
    return GENES[r];
}
```

```
//Genetik Algoritmanın temel yapı taşı olan kromozom sınıfı
14 usages  1 exposing API
private class Cromosome
{
    // Her kromozomun bir içeriği yani genomu vardır
    8 usages
    public string dna { get; }
    // Her kromozomun istenen sonuca ne kadar yakın olduğunu gösteren bir uygunluk değeri (Fitness Value) vardır
    6 usages
    public int FitnessValue { get; }

    //Generator fonksiyon
    2 usages
    public Cromosome(string chromosome)
    {
        dna = chromosome;
        FitnessValue = CalculateFitness();
    }

    // Fitness değerini hesaplayan fonksiyon
    1 usage
    private int CalculateFitness()
    {
        // Kod parçası, dna ve KEY koleksiyonlarını karşılaştırarak eşleşmeyen her çift için 1,
        // eşleşen çiftler için ise 0 döndürerek kromozomun fitness değerini hesaplar.
        return dna.Zip(KEY, (a:char, b:char) => a == b ? 0 : 1).Sum();
    }
```

```
    // Çaprazlama ile yeni jenerasyona ait bir kromozom oluşturma işlemi gerçekleştirilir
    1 usage
    public Cromosome Cross(Cromosome parent2)
    {
        // Yeni jenerasyona ait yeni bir kromozom oluşturulur
        char[] childChromosome = new char[dna.Length];

        //Çaprazlama işlemi gerçekleştirilir
        for (int i = 0; i < dna.Length; i++)
        {
            double p = random.NextDouble();
            if (p < 0.45)
                childChromosome[i] = dna[i];
            else if (p < 0.90)
                childChromosome[i] = parent2.dna[i];
            else
                childChromosome[i] = Mutation();//Düşük bir olasılıkla da olsa childChromosome
                                //geni parentlarından değil, mutasyon ile rastgele bir şekilde alır
        }
        return new Cromosome( chromosome: new string(childChromosome));
    }
}
```

```
public static void Main()
{
    //Algoritmanın ne kadar sürede gerçekleşeceiğini belirlemek için kronometre çalıştırılır.
    var watch = System.Diagnostics.Stopwatch.StartNew();

    // Şifrenin kaçıncı jenerasyonda kırıldığını takip etmek için kullanılan bir sayaç
    int generation = 0;

    List<Cromosome> population = new List<Cromosome>();
    //Genetik algoritmanın hangi aşamada sonlanacağını belirlemek için kullanılan bool değer
    bool didKeyBroke = false;

    //ilk populasyon oluşturulur
    for (int i = 0; i < POPULATION_SIZE; i++)
    {
        string genome = CreateCromosome();
        population.Add( item: new Cromosome(genome));
    }
```

```
while (!didKeyBroke)
{
    // populasyon fitnes değerlerine göre artan sırada sıralanır
    population.Sort(new FitnessComparer());

    //Fitness değeri ne kadar küçükse şifreye o kadar yaklaşılmış demektir
    //Eğerki populasyonun ilk kromozomu (yani populasyonun sonuca en yakın kromozomu)
    //fitness değeri 0 ise şifre kırılmış demektir.Algoritma sonlanır
    if (population[0].FitnessValue == 0)
    {
        didKeyBroke = true;
        break;
    }

// Eğer algoritma sonlanmadıysa bir sonraki jenerasyon oluşturulmaya başlanır
List<Cromosome> newGeneration = new List<Cromosome>();

// Bir önceki jenerasyonun ilk %10'luk kısmı yani jenerasyonun en iyileri bir sonraki jenerasyona aktarılır. Buna Elitizm denir
// Amaç umut verici olan kromozomlar üzerinden ilerlemeye devam edilerek sonuca ulaşmaktır
int s = (10 * POPULATION_SIZE) / 100;
for (int i = 0; i < s; i++)
    newGeneration.Add(population[i]);

// Populasyonun %10'u Elitizm ile belirlendiği için geri kalan %90'lık kısım çaprazlama ile belirlenir
s = (90 * POPULATION_SIZE) / 100;
for (int i = 0; i < s; i++)
{
    int len = population.Count;
    int r = genomeRoulette(0, 50);//Genom Çarkı ile çaprazlamda kullanılacak olan parentlar belirlenir.
                    //Sonuca daha yakın olan kromozomların çaprazlanmada kullanılma olasılığı daha yüksektir
    Cromosome parent1 = population[r];
    r = genomeRoulette(0, 50);
    Cromosome parent2 = population[r];
    Cromosome newCromosome = parent1.Cross(parent2);
    newGeneration.Add(newCromosome);//Yeni kromozom oluşturulur
}
//Yeni jenerasyon, asıl jenerasyonun yerini alarak sonraki işlemlerin kendisi üzerinden devam edilmesini sağlar
population = newGeneration;
//Kaçıncı jenerasyonda DNA'nın hangi durumda olduğu ve sonuca ne kadar yakın olduğu yazdırılarak algoritmanın takibi sağlanır
Console.WriteLine("Generation: " + generation + "\t" +
                "String: " + population[0].dna + "\t" +
                "Fitness: " + population[0].FitnessValue);

generation++;
}

    //Kronometre sonlandırılarak işlemin gerçekleşme süresi hesaplanır
    watch.Stop();

    Console.WriteLine("Generation: " + generation + "\t" +
                    "String: " + population[0].dna + "\t" +
                    "Fitness: " + population[0].FitnessValue + "\t" +
                    "Execution Time: " + watch.ElapsedMilliseconds + " Milisecond");
}
```

Run no 1 with 100 Chromosome

```
Generation: 0   String: R3[]4raI=Je/#{ #        Fitness: 13
Generation: 1   String: R3[]4raI=Je/#{ #        Fitness: 13
Generation: 2   String: RGen_ywn}Ue_AT%y        Fitness: 11
Generation: 3   String: RGen_ywn}Ue_AT%y        Fitness: 11
Generation: 4   String: &3dEerwIDde eI)F        Fitness: 10
Generation: 5   String: RGenerwnDUe eI%F        Fitness: 8
Generation: 6   String: RGenerwnDUe eI%F        Fitness: 8
Generation: 7   String: (Gdn_rLnMQe AI)         Fitness: 6
Generation: 8   String: (Gdn_rLnMQe AI)         Fitness: 6
Generation: 9   String: (Gener4I08e AI),        Fitness: 5
Generation: 10  String: (Gener4I08e AI),        Fitness: 5
Generation: 11  String: (Gener4I08e AI),        Fitness: 5
Generation: 12  String: (Gener4I08e AI),        Fitness: 5
Generation: 13  String: (GenerLtvQe AL)         Fitness: 4
Generation: 14  String: (GenerLtvQe AL)         Fitness: 4
Generation: 15  String: (GenerLtvQe AL)         Fitness: 4
Generation: 16  String: (Gener}tv:e AI)         Fitness: 3
Generation: 17  String: (Gener}tv:e AI)         Fitness: 3
Generation: 18  String: (Gener}tv:e AI)         Fitness: 3
Generation: 19  String: (Gener0thve AI)         Fitness: 2
Generation: 20  String: (Gener0thve AI)         Fitness: 2
Generation: 21  String: (Gener0thve AI)         Fitness: 2
Generation: 22  String: (Gener0thve AI)         Fitness: 2
Generation: 23  String: (Gener0thve AI)         Fitness: 2
Generation: 24  String: (Gener0thve AI)         Fitness: 2
Generation: 24  String: (Gener0thve AI)         Fitness: 2
Generation: 25  String: (Gener0thve AI)         Fitness: 2
Generation: 26  String: (Gener0thve AI)         Fitness: 2
Generation: 27  String: (Gener0thve AI)         Fitness: 2
Generation: 28  String: (Gener0thve AI)         Fitness: 2
Generation: 29  String: (Gener0thve AI)         Fitness: 2
Generation: 30  String: (Gener0thve AI)         Fitness: 2
Generation: 31  String: (Gener0thve AI)         Fitness: 2
Generation: 32  String: (Gener0thve AI)         Fitness: 2
Generation: 33  String: (Gener,tive AI)         Fitness: 1
Generation: 34  String: (Gener,tive AI)         Fitness: 1
Generation: 35  String: (Gener,tive AI)         Fitness: 1
Generation: 36  String: (Gener,tive AI)         Fitness: 1
Generation: 37  String: (Gener,tive AI)         Fitness: 1
Generation: 38  String: (Gener,tive AI)         Fitness: 1
Generation: 39  String: (Generative AI)         Fitness: 0
```

```
Execution Time: 24 Milisecond
```

Run no 2 with 100 Chromosome

```
Generation: 0   String: )1HESuFz6WX$]I="        Fitness: 15
Generation: 1   String: )BpSSIFx6WX$AI=r        Fitness: 14
Generation: 2   String: :BpSeKFx6=X$AIKr        Fitness: 13
Generation: 3   String: GIIC43[%60e ]I=         Fitness: 12
Generation: 4   String: GIIC43[%60e ]I=         Fitness: 12
Generation: 5   String: P8eKeLo5Cva =IX         Fitness: 10
Generation: 6   String: P8eKeLo5Cva =IX         Fitness: 10
Generation: 7   String: WFeneLf1ine}JIV         Fitness: 9
Generation: 8   String: WFeneLf1ine}JIV         Fitness: 9
Generation: 9   String: JW)n.$ "ive AIX         Fitness: 8
Generation: 10  String: )B8ne0p7ive AIK         Fitness: 7
Generation: 11  String: )B8ne0p7ive AIK         Fitness: 7
Generation: 12  String: $Dene0o5ive AIT         Fitness: 6
Generation: 13  String: $Dene0o5ive AIT         Fitness: 6
Generation: 14  String: $Dene0o5ive AIT         Fitness: 6
Generation: 15  String: VDener01ive AI#         Fitness: 5
Generation: 16  String: VDener01ive AI#         Fitness: 5
Generation: 17  String: VDener01ive AI#         Fitness: 5
Generation: 18  String: VDener01ive AI#         Fitness: 5
Generation: 19  String: VDener01ive AI#         Fitness: 5
Generation: 20  String: )Generj1ive AIT         Fitness: 4
Generation: 21  String: )Generj1ive AIT         Fitness: 4
Generation: 22  String: (Fenerajive AIA         Fitness: 3
Generation: 23  String: (Fenerajive AIA         Fitness: 3
Generation: 24  String: (Generalive AIq         Fitness: 2
Generation: 25  String: (Generalive AIq         Fitness: 2
Generation: 26  String: (Generalive AIq         Fitness: 2
Generation: 27  String: (Generalive AIq         Fitness: 2
Generation: 28  String: (Generalive AIq         Fitness: 2
Generation: 29  String: (Generalive AIq         Fitness: 2
Generation: 30  String: (Generalive AIq         Fitness: 2
Generation: 31  String: (Generalive AIq         Fitness: 2
Generation: 32  String: (Generalive AIq         Fitness: 2
Generation: 33  String: (Generalive AIq         Fitness: 2
Generation: 34  String: (Generalive AIq         Fitness: 2
Generation: 35  String: (Generalive AIq         Fitness: 2
Generation: 36  String: (Generalive AIq         Fitness: 2
Generation: 37  String: (Generative AIR         Fitness: 1
Generation: 38  String: (Generative AIR         Fitness: 1
Generation: 39  String: (Generative AIR         Fitness: 1
Generation: 40  String: (Generative AIR         Fitness: 1
Generation: 41  String: (Generative AIR         Fitness: 1
Generation: 42  String: (Generative AI)         Fitness: 0
```

```
Execution Time: 21 Milisecond
```

Average Generation Count for 100 Chromosome: 40.5

Run no 1 with 10000 Chromosome

```
Generation: 0   String: g75@Wr$Riv4d,eB/      Fitness: 13
Generation: 1   String: WGPneDP3/v;SyI%       Fitness: 10
Generation: 2   String: WGeneMatimE A?d@      Fitness: 7
Generation: 3   String: (OeneraQiv5 AI)       Fitness: 3
Generation: 4   String: (Generative AI)       Fitness: 0
```

Execution Time: 132 Milisecond

Run no 2 with 10000 Chromosome

```
Generation: 0   String: #O]v.r?{VUe32lv       Fitness: 13
Generation: 1   String: JGenyrK.iY"D,I)H      Fitness: 9
Generation: 2   String: (G!n;rathY" ,I)7      Fitness: 7
Generation: 3   String: (Genera.iPe AI)       Fitness: 2
Generation: 4   String: (Generative AI)       Fitness: 0
```

Execution Time: 117 Milisecond

Average Generation Count for 10000 Chromosome: 4

As the number of chromosomes increased, the execution time increased because of the number of mutations and crossover operations increased.

3.b Explain crossover, mutation and selection functions in the code

3.b.i Mutation

```
// Rastgele bir şekilde gen değiştirmemizi yarayan fonksiyon
2 usages
private static char Mutation()
{
    int len = GENES.Length;
    int r = RandomNum(0, len - 1);
    return GENES[r];
}
```

This function provides for random replacement of genes. Using this function we can randomly generate a genome.

```
// Mutation fonksiyonu ile rastgele bir şekilde genom oluşturan fonksiyon
1 usage
private static string CreateCromosome()
{
    int len = KEY.Length;
    char[] genome = new char[len];
    for (int i = 0; i < len; i++)
    {
        genome[i] = Mutation();
    }
    return new string(genome);
}
```

Also, during crossover, there is a low probability that the gene will be taken from the mutation and not from the parent. This is important to increase the diversity of the genetic algorithm and to allow new solutions to be discovered.

```
    else
        childChromosome[i] = Mutation();//Düşük bir olasılıkla da olsa childChromosome
                        //geni parentlarından değil, mutasyon ile rastgele bir şekilde alır
}
```

3.b.j Cross-over

Cross-over creates a new child from two parental chromosomes, increasing the genetic diversity of the population and allowing new solutions to be discovered.

```
    // Çaprazlama ile yeni jenerasyona ait bir kromozom oluşturma işlemi gerçekleştirilir
    1 usage
    public Cromosome Cross(Cromosome parent2)
    {
        // Yeni jenerasyona ait yeni bir kromozom oluşturulur
        char[] childChromosome = new char[dna.Length];

        //Çaprazlama işlemi gerçekleştirilir
        for (int i = 0; i < dna.Length; i++)
        {
            double p = random.NextDouble();
            if (p < 0.45)
                childChromosome[i] = dna[i];
            else if (p < 0.90)
                childChromosome[i] = parent2.dna[i];
            else
                childChromosome[i] = Mutation();//Düşük bir olasılıkla da olsa childChromosome
                                //geni parentlarından değil, mutasyon ile rastgele bir şekilde alır
        }
        return new Cromosome( chromosome: new string(childChromosome));
    }
}
```

First, a new chromosome is created. A random number is generated for each gene in the chromosome. If the number is less than 0.45, the gene is taken from the existing chromosome and added to the child chromosome. If it is between 0.45 and 0.90, the gene is taken from the other parent chromosome and added to the child chromosome. If the random number is greater than 0.90, a new mutation is created and added to the child chromosome.

3.b.k Selection

The genomeRoulette function allows good chromosomes to be used with a higher probability. In a population list sorted by fitness values, chromosomes closer to the desired result will be found at the top of the list.

```
// Gen dizisi; aranan şifreye daha yakın olan, yani daha çok umut vaad eden kromozomlara
// çaprazlama işleminde daha çok rastlanabilmesi için kullanılan bir fonksiyondur.
// Amaç daha iyi olan kromozomlar üzerinden ilerleyerek sonuca yaklaşmaktır
2 usages
private static int genomeRoulette(int start, int end)
{
    double p = random.NextDouble();

    //Jenerasyon Fitness değerlerine göre sıralandığı için istenen sonuca
    //daha yakın olan kromozomlar ilk kısımlarda yer alacaktır

    if (p < 0.5)
    {
        return random.Next(start, (start + end) / 2);

    }
    else if (p < 0.8)
    {
        return random.Next(start, end + 1);
    }
    else
    {
        return random.Next((start + end) / 2, end + 1);
    }
}
```

The function randomly selects a number from start to middle with a probability of 50%, completely random with a probability of 30%, and from middle to end with a probability of 20%.

```
// Bir önceki jenerasyonun ilk %10'luk kısmı yani jenerasyonun en iyileri bir sonraki jenerasyona aktarılır. Buna Elitizm denir
// Amaç umut verici olan kromozomlar üzerinden ilerlemeye devam edilerek sonuca ulaşmaktır
int s = (10 * POPULATION_SIZE) / 100;
for (int i = 0; i < s; i++)
    newGeneration.Add(population[i]);

// Populasyonun %10'u Elitizm ile belirlendiği için geri kalan %90'lık kısım çaprazlama ile belirlenir
s = (90 * POPULATION_SIZE) / 100;
for (int i = 0; i < s; i++)
{
    int len = population.Count;
    int r = genomeRoulette(0, 50);//Genom Çarkı ile çaprazlamda kullanılacak olan parentlar belirlenir.
                    //Sonuca daha yakın olan kromozomların çaprazlanmada kullanılma olasılığı daha yüksektir
    Cromosome parent1 = population[r];
    r = genomeRoulette(0, 50);
    Cromosome parent2 = population[r];
    Cromosome newCromosome = parent1.Cross(parent2);
    newGeneration.Add(newCromosome);//Yeni kromozom oluşturulur
}
```

The best 10% of the previous generation is passed on to the next generation. The remaining 90% are determined by crossing. The genomeRoulette function is used to determine the parent chromosomes in the crossover.

3.c Comparison of solution times for different words (“GenAI”)

Run 1 With 100 Chromosome

```
Generation: 0   String: G}:_O   Fitness: 4
Generation: 1   String: G}:_O   Fitness: 4
Generation: 2   String: RmkAI   Fitness: 3
Generation: 3   String: GehAO   Fitness: 2
Generation: 4   String: GehAO   Fitness: 2
Generation: 5   String: GegAI   Fitness: 1
Generation: 6   String: GegAI   Fitness: 1
Generation: 7   String: GenAI   Fitness: 0
```

```
Execution Time: 12 Milisecond
```

Run 2 With 100 Chromosome

```
Generation: 0   String: tgn$V   Fitness: 4
Generation: 1   String: G,{!I   Fitness: 3
Generation: 2   String: G,{!I   Fitness: 3
Generation: 3   String: G6FAI   Fitness: 2
Generation: 4   String: G6FAI   Fitness: 2
Generation: 5   String: G6FAI   Fitness: 2
Generation: 6   String: G/nAI   Fitness: 1
Generation: 7   String: G/nAI   Fitness: 1
Generation: 8   String: G/nAI   Fitness: 1
Generation: 9   String: G/nAI   Fitness: 1
Generation: 10  String: G/nAI   Fitness: 1
Generation: 11  String: G/nAI   Fitness: 1
Generation: 12  String: G/nAI   Fitness: 1
Generation: 13  String: GenAI   Fitness: 0
```

```
Execution Time: 25 Milisecond
```

Average Generation Count for 100 Chromosome: 10

Run 1 With 10000 Chromosome

```
Generation: 0   String: Ee0Ag   Fitness: 3
Generation: 1   String: GenA3   Fitness: 1
Generation: 2   String: GenAI   Fitness: 0
```

```
Execution Time: 57 Milisecond
```

Run 2 With 10000 Chromosome

```
Generation: 0   String: Gf:A2   Fitness: 3
Generation: 1   String: GenA@   Fitness: 1
Generation: 2   String: GenAI   Fitness: 0
```

```
Execution Time: 61 Milisecond
```

Average Generation Count for 10000 Chromosome: 59

Because the target word is shorter, fewer mutations and cross-overs are performed to reach the target. Hence the execution time is lower.

4) Machine Learning with Python [Classifier]

4.a Python Standardization Example

4.a.i Codes

```
from sklearn.preprocessing import StandardScaler import
numpy as np

#Örnek proje kağıdında örnek olarak verilen
https://www.askpython.com/python/examples/standardize-data-inpython
ile kendi örneğimiz olmak üzere iki farklı yolla yapılmıştır

#Örnek1
#1.Adım: Örnek bir veri oluşturalım

veri = ([2,9,3],[6,7,1],[5,4,7])

#Her bir sütunun yani aynı tipteki verinin ortalaması alınır
ortalama = np.mean(veri, axis = 0)
#for sütun in range(len(data[0])):
#   toplam = 0
#
#    for satır in range(len(data)):
#        toplam += data[satır][sütun]
#    ortalama[sütun] = toplam / len(data)
#ortalama yukarıdaki gibi de alınabilir ancak numpy kütüphanesi
yardımıyla daha kolay almayı tercih ettik

#2.Adım: Veri setindeki aynı tipteki verilerin standart sapması
hesaplanır
standart_sapma = np.std(veri,axis = 0)

#3.Adım: Standartization formulü olan "(Veri - Ortalama)/Standart
Sapma" formulünden yola çıkılarak Standartization hesaplanır
standardize_veri = (veri - ortalama) / standart_sapma

#Aynı işlem for döngüsü ile aşağıdaki gibi de gerçekleştirilebilir
```

```
#satır = len(veri)
#column = len(veri[0])
#standardized_veri = []

#for i in range(satır):
#    row = []
#    for j in range(column):
#        row.append((veri[i][j] - ortalama[j]) / standart_sapma[j])
#    standardized_veri.append(row)

#4.Adım Sonuçların Yazdırılması

print("Veri:") print(veri)

print("\nOrtalama:") print(ortalama)

print("\nStandart Sapma:") print(standart_sapma)

print("\nStandardize Edilmis Veri:") print(standardize_veri)

#Örnek 2
#1.Adım StandardScaler bir değişkene atanır
scaler = StandardScaler()

#2.Adım 1.Örnekte oluşturulan data fit_transform metodu ile
Standartize edilir
scaled_standardize_veri = scaler.fit_transform(veri)

#3.Adım Sonuç yazdırılır
print("\nStandardScaler Kullanılarak Standardize Edilmis Veri:")
print(scaled_standardize_veri)
```

4.a.ii Difference from Normalization

Standardization involves rescaling the data to have a mean of 0 and a standard deviation of 1, ensuring that all variables are on the same scale for accurate analysis, particularly in multivariate scenarios. This process is crucial in mitigating the influence of variables measured on different scales, as failing to standardize can result in disproportionate contributions to the analysis. On the other hand, normalization, such as Feature Scaling or Min-Max Normalization, scales the data between 0 and 1, maintaining the distribution of the data. While normalization is beneficial for ensuring consistent positive scales, it can lead to the loss of outliers. In contrast, standardization, exemplified by methods like Z Score Scaling, ensures that all variables have a mean of zero and a standard deviation of one, facilitating comparisons across variables and minimizing the disproportionate impact of differently scaled features in analyses.

4.b Finding or Creating Dataset

4.b.i Information about the dataset

The Breast Cancer dataset from the UCI ML Repository Data Folder was used fort he classification application of the breast cancer. In this dataset:

- 569 sample data
- 30 attributes
- 2 classes

The attributes used in this dataset are as follows:

- Radius: Distances from center to points on the perimeter.
- Texture: Standard deviation of gray-scale values.
- Perimeter: Perimeter of cyst.
- Area: Area of cyst.
- Smoothness: Local variation in radius lengths.
- Compactness: Compactnes of a cyst is a feature that determines how regular or compressed its shape is relative to a measure f its volüme or area.
- Concavity: Severity of concave portions of the contour.
- Concave Points: The number of concave portions of the contour.
- Symmetry
- Fractal Dimension: Fractal dimension of a cyst is obtained by measuring the fractal dimension of each cyst in a series and then taking the average of these measurements, representing the overall complexity or irregularity level of the cysts.

The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, there are 3 radius feature that include Mean Radius, Radius Standard Error and Worst Radius.

4.c Two different classifier and results

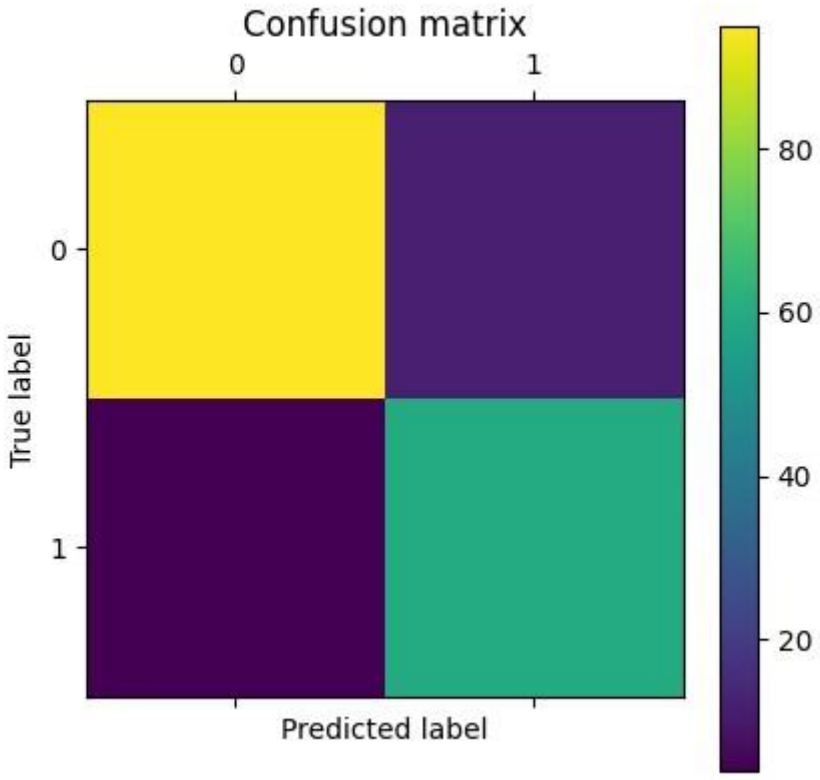
4.c.i Source Code

4.c.i.i Decision Tree

```
#DECISION TREE from sklearn.tree import
DecisionTreeClassifier from sklearn.model_selection
import train_test_split from sklearn.metrics import
confusion_matrix from ucimlrepo import
fetch_ucirepo import matplotlib.pyplot as plt from
sklearn.model_selection import cross_validate

#dataset sisteme yüklenir
breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)
# data (as pandas dataframes)
X = breast_cancer_wisconsin_diagnostic.data.features
y = breast_cancer_wisconsin_diagnostic.data.targets
 #Decision Tree ile Sınıflandırma yapacağımız fonksiyon bir değişkene atanır
clf = DecisionTreeClassifier(random_state=0)

#Veri seti, test seti ve eğitim seti olarak parçalara ayrılır
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size = 0.7,
test_size = 0.3, random_state = 0, stratify = y)

# Eğitim Verisi ile eğitimi gerçekleştiriyoruz
clf.fit(X_train,y_train)

#Sonucu alıp yazdıralım
test_sonuc = clf.predict(X_test)
print(test_sonuc)

#Bir confusion matrixi oluşturalım cm =
confusion_matrix(y_test, test_sonuc)
print(cm)
#Oluşturulan confusion matrixi ayrı bir pencerede
görselleştirilir plt.matshow(cm) plt.title('Confusion matrix')
plt.colorbar() plt.ylabel('True label') plt.xlabel('Predicted
label') plt.show()
```

The confusion matrix of the code is: [[95 12] [4 60]]

Image of the Confusion Matrix of Decision Tree Code



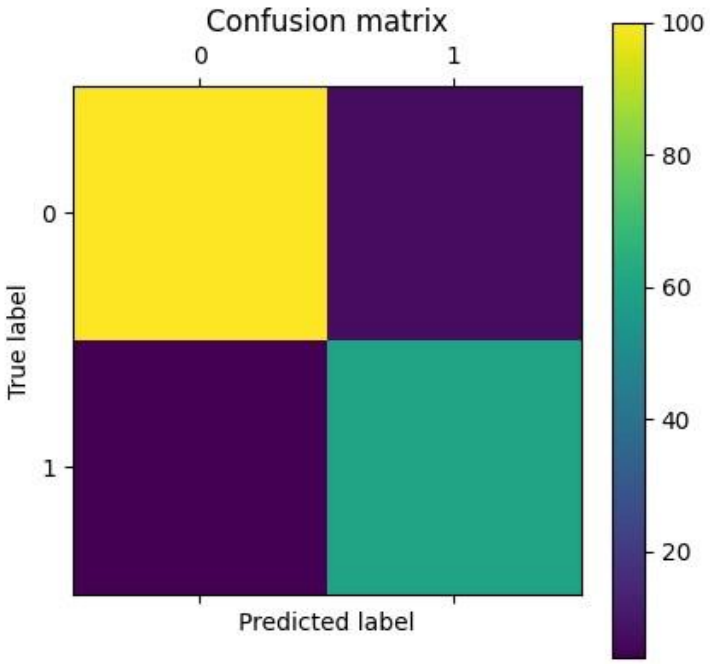
4.c.i.ii Random Forest

```
#RANDOM FOREST from sklearn.ensemble import
RandomForestClassifier from sklearn.model_selection
import train_test_split from sklearn.metrics import
confusion_matrix from ucimlrepo import
fetch_ucirepo import matplotlib.pyplot as plt from
sklearn.model_selection import cross_validate


# dataset sisteme yüklenir
breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)
# data (as pandas dataframes)
X = breast_cancer_wisconsin_diagnostic.data.features
y = breast_cancer_wisconsin_diagnostic.data.targets.values.ravel()
#Random Forest ile Sınıflandırma yapacağımız fonksiyon bir değişkene atanır
rfc = RandomForestClassifier(random_state=0)
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size = 0.7,
test_size = 0.3, random_state = 0, stratify = y)

# Eğitim Verisi ile eğitimi gerçekleştiriyoruz rfc.fit(X_train,y_train)

#Sonucu alıp yazdıralım test_sonuc
= rfc.predict(X_test)
print(test_sonuc)

#Bir confusion matrixi oluşturalım cm =
confusion_matrix(y_test, test_sonuc)
print(cm)
#Oluşturulan confusion matrixi ayrı bir pencerede görselleştirilir
plt.matshow(cm) plt.title('Confusion matrix') plt.colorbar()
plt.ylabel('True label') plt.xlabel('Predicted label') plt.show()
```

The confusion matrix of the code is: [[100 7][4 60]]

Image of the Confusion Matrix of Random Forest Code



4.c.ii Confusion Matrix

The Confusion Matrix is a tool used to evaluate the performance of classification algorithms. It compares the actual and predicted classes and summarizes the classification results. A classification model assigns data points to specific classes. The Confusion Matrix compares these predictions with the actual class labels to assess the accuracy of the predistions.

When comparing the given confusion matrices, we can say that Random Forest outperforms Decision Tree in terms of classification.

4.d Predicting the class of an instance that its class is unknown

4.d.i Source Code

4.c.i.i Decision Tree

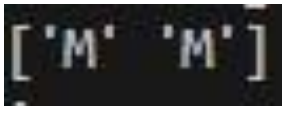
```
#İki adet örnek veri oluşturulup bu veri setilerin hangi sınıfa ait
olduğu bulunur test_ornek =
([17.52,19.51,99.84,564.7,0.1132,0.2338,0.1325,0.08356,0.1499,0.08892,0.7891
,1.968,4.373,50.78,0.007907,0.05756,0.06443,0.01623,0.02947,0.006643,15.43,2
4.71,103.1,856.5,0.1146,0.4762,0.4922,0.1685,0.2334,0.09032],
              [16.44,18.24,90.77,516.9,0.1002,0.1339,0.29966,0.09064,0.1216,
0.04396,0.4175,1.3772,4.184,62.76,0.006808,0.06612,0.04646,0.01848,0.02121,0
.003321,18.34,22.22,119.4,873.6,0.1607,0.6830,0.3104,0.1697,0.2851,0.03473])
ornek_sonucu = clf.predict(test_ornek) print(ornek_sonucu)
```

4.c.i.ii Random Forest

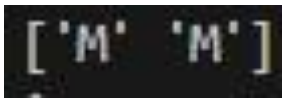
```
#İki adet örnek veri oluşturulup bu veri setilerin hangi sınıfa ait
olduğu bulunur test_ornek =
([17.52,19.51,99.84,564.7,0.1132,0.2338,0.1325,0.08356,0.1499,0.08892,0.7891,1
.968,4.373,50.78,0.007907,0.05756,0.06443,0.01623,0.02947,0.006643,15.43,24.71
,103.1,856.5,0.1146,0.4762,0.4922,0.1685,0.2334,0.09032],
              [16.44,18.24,90.77,516.9,0.1002,0.1339,0.29966,0.09064,0.1216,0.
04396,0.4175,1.3772,4.184,62.76,0.006808,0.06612,0.04646,0.01848,0.02121,0.003
321,18.34,22.22,119.4,873.6,0.1607,0.6830,0.3104,0.1697,0.2851,0.03473])
ornek_sonucu = rfc.predict(test_ornek) print(ornek_sonucu)
```

4.d.ii Screenshots

4.c.i.i Decision Tree



4.c.i.ii Random Forest



4.f 10-Fold Cross Validation

4.f.i Source Code

```
#Accuracy,Precision ve Recall değerlerini kıyaslayalım
scoring = ["accuracy",'precision_macro', 'recall_macro']
#10 katmanlı cross validation gerçekleştirilir scores =
cross_validate(clf, X, y, scoring=scoring, cv = 10)
sorted(scores.keys())
["accuracy", 'test_precision_macro',
'test_recall_macro']
#Sonuçların ortalama değeri alınır accuracy_mean =
scores['test_accuracy'].mean() precision_mean =
scores['test_precision_macro'].mean() recall_mean =
scores['test_recall_macro'].mean()

#Sonuçlar formatlanarak yazdırılır
print("Accuracy: {:.2f}".format(accuracy_mean))
print("Precision: {:.2f}".format(precision_mean))
print("Recall: {:.2f}".format(recall_mean))
```

4.f.ii Classification Performance with 10-Fold Cross Validation

Model	Accuracy	Precision	Recall
Decision Tree	0.92	0.91	0.91
Random Forest	0.96	0.96	0.96

5-) Time Spent and division of Labor Details

Ahmet Alper Akın: He did item 4 and item 2, he did item 3 together with Berk.
(25 Hours)
Mehmet Berk Bozkurt: He did item 1.a, 1.b, helped Alper in item 4.a, helped Alper in item 3.
(17 Hours)
Timurcan Yıldız: He did item 1.c, 1.d. In other items, especially in item 2, he helped, checked and prepared the report. (15 Hours)

6) Self-assessment Table

	İstenen Özellik	**Var**	**Açıklama**	**Tahmini Not**
1a	Algoritmalar + Karmaşıklıklar (10)	☒	There were some disagreements about the time complexity, but everything that was requested in the article was done.	9
1b	Gen AI ile A* Öğrenme ve Kodlama (10)	☒	GenAI was used effectively and the desired things were done.	10
1c	Tanım ve Karşılaştırma (10)	☒	Concepts such as Ensemble Learning, LLM, Devin, and CycleGAN were explained. GPT-4, Gemini, and Claude 3 were compared to each other.	10
1d	Araştırma ve Yorum (10)	☒	We've analized Machine Learning based interview questions and graded them to show how related and effective they are	10
2	Problem Çözme ve Kodlama (10)	☒	The 8 Queens problem has been solved using the	10

			Hill Climbing algorithm. The basic principles of the Hill Climbing algorithm have been successfully implemented and explained. Additionally, what Stochastic Hill Climbing algorithm is and how it can be implemented have also been specified.	
3	Genetik Algoritmalar ile Şifre Kırma (15)	☒	Attention was paid to the important details of the genetic algorithm. By using features such as mutaston, cross-over, selection effectively, a genetic algorithm that can reach the password in the desired way was obtained.	15
4	Makine Öğrenmesi (15)	☒	An example of standardization is provided, and the difference with "Normalization" is explained. Subsequently, two different classification algorithms, Decision Tree and Random Forest, are experimented with. The confusion matrices of these algorithms are provided. Finally, these algorithms are compared	15

			across various domains using "10-Fold Cross Validation".	
5	İşbölümü Detayları (5)	☒	Throughout the project, it has been clearly indicated which areas the team members worked on.	5
6	Özd. Tablosu + Rapor (15)	☒	The tasks performed throughout the project and the expected points to be gained from these tasks have been specified.	15
			100 üzerinden Toplam Not:	99