

## İçindekiler

<b>BİLGİSAYAR FELSEFESİ .....</b>	<b>5</b>
<b>SORU-1: Apriori (Malum), APosteriori, Afortiori .....</b>	<b>6</b>
<b>SORU-2: Yeknesan (Invariant , Değişmez) hakkında bilgi veriniz.....</b>	<b>10</b>
<b>SORU-3: Algoritma (Algorithm) hakkında açıklama yapınız.....</b>	<b>11</b>
<b>SORU-4: Birbirini Dışlama (Mutually Exclusive) hakkında bilgi veriniz.....</b>	<b>12</b>
<b>SORU-5: WOLG (Genelliğini Kaybetmeden) hakkında bilgi veriniz. ....</b>	<b>13</b>
<b>SORU-6: Zamansal Mantıklarda Etki Alanı Yapısı (Temporal Domain Structure) hakkında bilgi veriniz.....</b>	<b>13</b>
<b>SORU-7: Cardinality (Sayısalılık) hakkında bilgi veriniz. ....</b>	<b>17</b>
<b>SORU-8: İkinci Derece Mantık (Second Order Logic) hakkında bilgi veriniz. ....</b>	<b>24</b>
<b>SORU-9: Birinci Derece Mantık (First Order Logic) hakkında bilgi veriniz.....</b>	<b>25</b>
<b>SORU-10: Zeki Vekiller (Akıllı Ajanlar, Intelligent Agents, Zeki Etmenler ) hakkında bilgi veriniz. ....</b>	<b>27</b>
<b>SORU-11: Mere Paradoksu (Mere's Paradox) hakkında bilgi veriniz. ....</b>	<b>28</b>
<b>SORU-12: Güvercin Yuvası Kaidesi (Pigeonhole Principle) hakkında bilgi veriniz. ....</b>	<b>29</b>
<b>SORU-13: Satrançta Büyük Usta Problemi hakkında bilgi veriniz. ....</b>	<b>30</b>
<b>SORU-14: Matematiksel Tümevarımın ikinci Teoremi (Second principle of mathematical induction) hakkında bilgi veriniz.....</b>	<b>31</b>
<b>SORU-15: Bağlam Yönelimli Programlama (Aspect Oriented Programming) hakkında bilgi veriniz.....</b>	<b>32</b>
<b>SORU-16: Gellish (Kontrollü Doğal Dil) hakkında bilgi veriniz. ....</b>	<b>34</b>
<b>SORU-17: Karar Problemi (Decision Problem) hakkında bilgi veriniz.....</b>	<b>35</b>
<b>SORU-18: Özyineli sayılabilir küme (Recursively Enumerable Sets) hakkında bilgi veriniz..</b>	<b>35</b>
<b>SORU-19: Hesaplanabilir Fonksiyon (Computable Function) hakkında bilgi veriniz.....</b>	<b>36</b>
<b>SORU-20: Özyineli Küme (Recursive Set) hakkında bilgi veriniz. ....</b>	<b>37</b>
<b>SORU-21: Anlamsal Bağ (Semantic Link) hakkında bilgi veriniz. ....</b>	<b>39</b>
<b>SORU-22: Küme Teorisi (Set Theory) hakkında bilgi veriniz. ....</b>	<b>40</b>
<b>SORU-23: Güç Kümesi (Kuvvet Kümesi, Power Set) hakkında bilgi veriniz. ....</b>	<b>43</b>
<b>SORU-24: Malumat Çıkarımı (Knowledge Retrieval) hakkında bilgi veriniz. ....</b>	<b>44</b>
<b>SORU-25: Malumat İfadesi (Knowledge Representation) hakkında bilgi veriniz.....</b>	<b>45</b>
<b>SORU-26: Pragma (Edimbilim, kullanımbilim, Fiili, Ameli) hakkında bilgi veriniz. ....</b>	<b>47</b>
<b>SORU-27: Haber (Predicate) hakkında bilgi veriniz. ....</b>	<b>48</b>
<b>SORU-28: Şekli Mantık (Kipler Mantığı, Modal Logic) hakkında bilgi veriniz.....</b>	<b>50</b>
<b>SORU-29: Zamansal Mantık (Temporal Logic) hakkında bilgi veriniz. ....</b>	<b>52</b>
<b>SORU-30: Minimax Ağaçları (Minimax Tree) hakkında bilgi veriniz.....</b>	<b>54</b>

<b>SORU-31: Semafor (Semaphore, Flama, İşaret) hakkında bilgi veriniz.....</b>	<b>59</b>
<b>SORU-32: Atomluluk (Atomicity) hakkında bilgi veriniz.....</b>	<b>60</b>
<b>SORU-33: Zamanın Gerçekdışılığı (Unreality of Time, Hayal-i Vakit) hakkında bilgi veriniz. .....</b>	<b>60</b>
<b>SORU-34: Zamani (Temporal, Zamansal, Zamane, Mevkuat) hakkında bilgi veriniz.....</b>	<b>61</b>
<b>SORU-35: Tehlike (Hazard) hakkında bilgi veriniz.....</b>	<b>61</b>
<b>SORU-36: İçerik Bağımsız Gramerler için Pompalama Önsavı (Pumping Lemma for Context Free Grammers) hakkında bilgi veriniz.....</b>	<b>62</b>
<b>SORU-37: Pompalama Önsavı (Pumping Lemma) hakkında bilgi veriniz.....</b>	<b>64</b>
<b>SORU-38: Uyum (Agreement, Kabul, Bağıt, Mutabakat) hakkında bilgi veriniz.....</b>	<b>64</b>
<b>SORU-39: Dolaylı sıralama (Indirect Sort, Gayrimüstakim sıralama) hakkında bilgi veriniz. .....</b>	<b>65</b>
<b>SORU-40: Soru Cevaplama (Question Answering, QA) hakkında bilgi veriniz.....</b>	<b>65</b>
<b>SORU-41: Genetik Algoritmalar (Genetic Algorithms) hakkında bilgi veriniz.....</b>	<b>66</b>
<b>SORU-42: Aday Anahtar (Candidate Key) hakkında bilgi veriniz.....</b>	<b>67</b>
<b>SORU-43: Java Bean hakkında bilgi veriniz.....</b>	<b>68</b>
<b>SORU-44: Bağlama (Coupling) hakkında bilgi veriniz.....</b>	<b>69</b>
<b>SORU-45: Principal Component Analysis hakkında bilgi veriniz.....</b>	<b>69</b>
<b>SORU-46: Eigenvalue (Özdeğer) Eigen vector (Öz yöney) Eigen Space (Öz Uzay) hakkında bilgi veriniz.....</b>	<b>69</b>
<b>SORU-47: Dik Vektörler (Orthogonal Vectors) hakkında bilgi veriniz.....</b>	<b>70</b>
<b>SORU-48: Sezgi Üstü Algoritmalar (Üstsezgisel Algoritmalar, Meta Heuristic Algorithms) hakkında bilgi veriniz.....</b>	<b>70</b>
<b>SORU-49: En kötü durum analizi (Worst Case Analysis) hakkında bilgi veriniz.....</b>	<b>72</b>
<b>SORU-50: Sezgisel Algoritmalar (Buluşsal Algoritmalar, Heuristic Algorithms) hakkında bilgi veriniz.....</b>	<b>75</b>
<b>SORU-51: Bayes Ağları (Bayesian Network) hakkında bilgi veriniz.....</b>	<b>75</b>
<b>SORU-52: Birliktelik, Münasebet ve Oluşum (Association, Aggregation and Composition) hakkında bilgi veriniz.....</b>	<b>78</b>
<b>SORU-53: Linear Programming (Doğrusal Programlama) hakkında bilgi veriniz.....</b>	<b>78</b>
<b>SORU-54: Entropi (Entropy, Dağınım, Dağıntı) hakkında bilgi veriniz.....</b>	<b>79</b>
<b>SORU-55: Doğrusal olmayan DVM ( Non-linear SVM) hakkında bilgi veriniz.....</b>	<b>81</b>
<b>SORU-56: Çok sınıflı DVM ( Multiclass SVM) hakkında bilgi veriniz.....</b>	<b>82</b>
<b>SORU-57: Kazanan Hepsini Alır (Winner Takes All) hakkında bilgi veriniz.....</b>	<b>84</b>
<b>SORU-58: Özellik Çıkarımı (Feature Extraction) hakkında bilgi veriniz.....</b>	<b>84</b>
<b>SORU-59: Şelale Modeli ( Waterfall Model ) hakkında bilgi veriniz.....</b>	<b>84</b>
<b>SORU-60: Vücubiyet (Modality) hakkında bilgi veriniz.....</b>	<b>85</b>
<b>SORU-61: Sayısalılık (Cardinality) hakkında bilgi veriniz.....</b>	<b>86</b>

<b>SORU-62: ERD ( Unsur İlişki Çizimi, Entity Relationship Diagram ) hakkında bilgi veriniz.</b>	<b>88</b>
<b>SORU-63: İlişki (Relationship) hakkında bilgi veriniz.</b>	<b>89</b>
<b>SORU-63: Unsur (Entity) hakkında bilgi veriniz.</b>	<b>91</b>
<b>SORU-64: Kıtık (Starvation) hakkında bilgi veriniz.</b>	<b>92</b>
<b>SORU-65: Durma Problemi (Halting Problem) hakkında bilgi veriniz.</b>	<b>92</b>
<b>SORU-66: Tersine Koyarak İspat (antitez, Contraposition) hakkında bilgi veriniz.</b>	<b>93</b>
<b>SORU-67: Doğrudan İspat (Direct Proofing) hakkında bilgi veriniz.</b>	<b>94</b>
<b>SORU-68: Bilgisayar Mühendisliği hakkında bilgi veriniz.</b>	<b>96</b>
<b>SORU-69: İkili Prensibi (Duality Principle, İstaniyet) hakkında bilgi veriniz.</b>	<b>100</b>
<b>SORU-70: Dış Yol Uzunluğu (External Path Length) hakkında bilgi veriniz.</b>	<b>100</b>
<b>SORU-71: Kubit (Qubit) hakkında bilgi veriniz.</b>	<b>101</b>
<b>SORU-72: Sıralama Algoritmaları (Sorting Algorithms) hakkında bilgi veriniz.</b>	<b>102</b>
<b>SORU-73: Gerekçilik (Nedensellik, Determinism) hakkında bilgi veriniz.</b>	<b>102</b>
<b>SORU-74: İstikra ile ispat (Tüme varım, Proof by Induction) hakkında bilgi veriniz.</b>	<b>103</b>
<b>SORU-75: İstikra(Tüme varım, Induction) hakkında bilgi veriniz.</b>	<b>104</b>
<b>SORU-76: burhan-ı mütenakıs (proof by contradiction, olmayana ergi) hakkında bilgi veriniz.</b>	<b>104</b>
<b>SORU-77: Binaen Burhan (İnşâa ile İspat , Proof by Construction, Binaenaleyh) hakkında bilgi veriniz.</b>	<b>105</b>
<b>SORU-78: Nazariye (Teori, Kuram, Theorem) hakkında bilgi veriniz.</b>	<b>106</b>
<b>SORU-79: Bilgi, Veri, Mâlûmat, İrfan (Knowledge, Data, Information, Wisdom) hakkında bilgi veriniz.</b>	<b>106</b>
<b>SORU-80: Normal Şekil (Canonical Form) hakkında bilgi veriniz.</b>	<b>107</b>
<b>SORU-81: Bilgi Çıkarımı (Information Extraction) hakkında bilgi veriniz.</b>	<b>107</b>
<b>SORU-82: Belirsiz Çokterimli Tam (NP-Complete, Nondeterministic Polynomial Complete) hakkında bilgi veriniz.</b>	<b>108</b>
<b>SORU-83: Açgözlü Yaklaşımı (Greedy Approach) hakkında bilgi veriniz.</b>	<b>108</b>
<b>SORU-84: Geçişli Fiiller (transitive verbs) hakkında bilgi veriniz.</b>	<b>108</b>
<b>SORU-85: İkili dil (bigram) hakkında bilgi veriniz.</b>	<b>109</b>
<b>SORU-86: Kelime Bilim (lexicology, vocabulary) hakkında bilgi veriniz.</b>	<b>109</b>
<b>SORU-87: Ekleme (apposition) hakkında bilgi veriniz.</b>	<b>110</b>
<b>SORU-88: Dönüştü (anaphora) hakkında bilgi veriniz.</b>	<b>110</b>
<b>SORU-89: İlgi Belirsizliği (reference ambiguity) hakkında bilgi veriniz.</b>	<b>110</b>
<b>SORU-90: Yapısal Belirsizlik (Structural Ambiguity) hakkında bilgi veriniz.</b>	<b>110</b>
<b>SORU-91: Kelime-İfade Belirsizliği (word-sense ambiguity) hakkında bilgi veriniz.</b>	<b>111</b>
<b>SORU-92: Belirsizlik (ambiguity, muğlaklık, ikircimlik) hakkında bilgi veriniz.</b>	<b>111</b>
<b>SORU-93: Bağlaç (conjunction) hakkında bilgi veriniz.</b>	<b>111</b>
<b>SORU-94: Edat (preposition) hakkında bilgi veriniz.</b>	<b>111</b>

<b>SORU-95: zarf (adverb) hakkında bilgi veriniz. ....</b>	<b>112</b>
<b>SORU-96: Sıfat (adjective) hakkında bilgi veriniz. ....</b>	<b>112</b>
<b>SORU-97: Fiil (verb) hakkında bilgi veriniz. ....</b>	<b>112</b>
<b>SORU-98: Özel isim (hususî isim, proper noun) hakkında bilgi veriniz. ....</b>	<b>115</b>
<b>SORU-99: Cins İsim (common noun) hakkında bilgi veriniz. ....</b>	<b>115</b>
<b>SORU-100: Soyut İsim (mücerret, abstract noun) hakkında bilgi veriniz. ....</b>	<b>115</b>
<b>SORU-101: isim (noun) hakkında bilgi veriniz. ....</b>	<b>115</b>
<b>SORU-102: Sayılabilir İsimler (count noun) hakkında bilgi veriniz. ....</b>	<b>116</b>
<b>SORU-103: kütle adı (mass noun) hakkında bilgi veriniz. ....</b>	<b>116</b>
<b>SORU-104: Sonlu Durum Makinası (Finite State Machine, Finite State Automaton) hakkında bilgi veriniz. ....</b>	<b>116</b>
<b>SORU-105: otomat yönelimli programlama (automata based programming) hakkında bilgi veriniz. ....</b>	<b>117</b>
<b>SORU-106: üst programlama yaklaşımı (metaprogramming) hakkında bilgi veriniz. ....</b>	<b>117</b>
<b>SORU-107: fonksiyonel programlama (functional programming) hakkında bilgi veriniz. ...</b>	<b>118</b>
<b>SORU-108: yapısal programlama (structured programming) hakkında bilgi veriniz. ....</b>	<b>119</b>
<b>SORU-109: çoklayıcı (multiplexer) hakkında bilgi veriniz. ....</b>	<b>120</b>
<b>SORU-110: yarım toplayıcı (half adder) hakkında bilgi veriniz. ....</b>	<b>123</b>
<b>SORU-111: doğruluk çizelgesi (truth table) hakkında bilgi veriniz. ....</b>	<b>124</b>
<b>SORU-112: En uzun Ortak Küme (longest common subsequence, Lcs) hakkında bilgi veriniz. ....</b>	<b>126</b>
<b>SORU-113: Dinamik Programlama (Dynamic programming) hakkında bilgi veriniz. ....</b>	<b>127</b>
<b>SORU-114: parçala fethet yöntemi (divide and conquer) hakkında bilgi veriniz. ....</b>	<b>128</b>
<b>SORU-115: varlık bilim (ontoloji (ontology)) hakkında bilgi veriniz. ....</b>	<b>128</b>

# BİLGİSAYAR FELSEFESİ

## **SORU-1: Apriori (Malum), APosteriori, Afortiori**

Bir bilginin malum olması, daha önceki bilgilere ihtiyacı olmadan, ispata gerek duymadan doğruluğunun kabul edilmesi.

Örneğin “bir bütünün parçalarının, bütünden küçük olması” gibi. Bu bilginin ispata ihtiyacı yoktur ve doğru olarak kabul edilebilir, bu bilgi üzerine ispat kurulabilir.

Türkçede ayrıca “önsel” kelimesi de kullanılmaktadır ve literatürde “a priori” şeklinde ayrı yazılması söz konusudur.

Ayrıca tersini ifade sadedinden “a posteriori” terimi de kullanılmaktadır. Bu terim de Türkçede “sonsal” olarak geçebilir. Buradaki ifade edilmek istenen ise bir olaydan / eylemden sonra elde edilen bilgidir. Örneğin bir deney sonucunda elde edilen ve deneysel temele dayanan bilgilerin tamamı aposteriori olarak kabul edilebilir.

A Fortiori olarak geçen terim ise, kesin sonuca varan yargı anlamındadır. Literatürde bir sonuca ulaşırken kullanılan ve sonucu etkileyen etkenlere verilen isimdir ve genelde “a fortiori argument” olarak geçer. Örneğin “bir insan ölmüşse, bu insanın nefes almadığı” kabul edilebilir ve bu kabulün ispatına ihtiyaç duyulmadan üzerine bir yargı inşa edilebilir.

Ayrıca veri madenciliğinde (datamining) kullanılan ve veriler arasındaki ilişki modelini belirleyen bir algoritmaya da apriori ismi verilmiştir.

### ***Apriori Algoritması***

Veri madenciliğinde kullanılan ve veri kümeleri veya veriler arasındaki ilişkiyi çıkarmak için geliştirilmiş algoritmanın ismidir.

Apriori algoritması, özellikle çok büyük ölçekli veri tabanları (VLDB, very large databases) üzerindeki veri madenciliği (datamining) çalışmalarında geliştirilmiştir. Genel anlamda münasebet kuralı (association rule, birliktelik kuralı) çıkarımında kullanılan bir algoritmadır. Algoritmanın amacı, veri tabanında bulunan satırlar arasındaki bağlantıyı ortaya çıkarmaktır.

Algoritmanın ismi, kendinden önceki çıkarımlara bağlı olduğu için, latince, önce anlamına gelen “prior” kelimesinden gelmektedir.

Algoritma yapı olarak, aşağıdan yukarıya (bottom-up) yaklaşımı kullanmakta olup, her seferinde tek bir elemanı incelemekte ve bu elemanla diğer adaylarla münasebetini ortaya çıkarmaya çalışmaktadır.

Ayrıca algoritmanın her eleman için çalışmasını, bir arama algoritmasına benzetmek mümkündür. Algoritma, bu anlamda sığ öncelikli arama (breadth first search) yapısında olup, sanki adayları birer ağaç (tree) gibi düşünerek bu ağaç üzerinde arıyor kabul edilebilir.

Ağaç yapısında, k elemanlı bir aday listesinden k-1 elemana baktıktan sonra, alt frekans örüntüsü yetersiz olan elemanları budamakta ve kalan elemanların üzerinden arama yapmaya devam etmektedir.

### **Örnek**

Algoritmayı daha iyi anlamak için bir örnek üzerinden inceleyelim.

Bir firmanın ürünlerink SKU numaraları ile (Stock Keeping Unit, depo numarası), veritabanında tuttuğunu düşünelim. Bu durumda firmanın satılan ürünlerden hangilerinin beraber alındığını bulma

şansı olabilir. Örneğin kasada işlem yapıldığında aynı fatura içerisinde yer alan SKU'ların beraber satıldığını kabul edelim.

Örneğin faturaların tutulduğu veri tabanı tablosundaki kayıtlar aşağıdaki şekilde olabilir:

{1,2,3,4},

{1,2},

{2,3,4},

{2,3},

{1,2,4},

{3,4},

{2,4}

Yukarıdaki her faturada, hangi ürünün diğer hangi ürünlerle birlikte satıldığı görülmektedir. Buna göre, örneğin yukarıdaki listedeki ikinci satır, firmanın sattığı ürünlerden 1 ve 2 SKU numarasına sahip ürünleri temsil etmektedir (diyelim ki 1 numaralı ürün kravat ve 2 numaralı ürün gömlek olsun, bu durumda ikinci satır, {1,2} kaydı ile bize gömlek ve kravatın beraber satıldığını anlatmaktadır)

Apriori algoritmamızın ilk adımı, her ürünün frekansını, yani kaç kere listede geçtiğini saymak olacaktır.

Ürün	Frekans
1	3
2	6
3	4
4	5

Tablo-3.1: Ürün-frekans tablosu

Yukarıdaki tabloda, her ürünün toplam satış sayısı bulunmaktadır. Bu değere frekans veya destek (support) ismi verilmektedir.

Algoritmanın ikinci adımında, asgari destek değerini belirliyoruz. Bu belirleme işlemi, üretilen tabloya göre değişebilmektedir. Yukarıdaki örnek için asgari desteğimiz 3 olsun.

Algoritmanın sıradaki adımı, ürünleri 2'li gruplara ayırmak olacak. Buradaki amaç her elemanın diğer elemanlarla olan münasebetini bulmak. Yukarıdaki tabloda, frekansı düşük olan (daha seyrek olan, sık geçmeyen) elemanları eliyoruz. Bunların sonuç listesinde de yer almayacağını kabul ediyoruz.

Bu defa tablomuzda sadece 2 elamanlı listeleri bulunduruyoruz:

Ürün	Frekans(Destek)
{1,2}	3
{2,3}	3
{2,4}	4
{3,4}	3

Tablo-3.2: Ürün-frekans tablosu

Yukarıdaki tabloda bulunan değerleri listelerdeki çiftlerden çıkmıştır. Örneğin {1,2} değeri, 3 yerde geçmektedir ve bunlar:

{1,2,3,4},

{1,2},

{1,2,4},

Listeleridir ( veya senaryomuz gereği faturalar olarak düşünülebilir). Dolayısıyla {1,2} ikilisi için 3 frekansı hesaplanmış olur.

Yukarıdaki listenin devamında 3 elemanlı listelerin tablosu da çıkarılabilir ancak ne yazık ki 3 veya 4 elemanlı liste örneğimizde çıkarılamamaktadır. Bunun sebebi asgari destek değerimizi 3 kabul etmiş olmamızdır. Örneğin {1,2,4} şeklindeki faturayı ele alalım. Bu değer sadece 2 yerde geçmektedir:

{1,2,3,4},

{1,2,4},

Ve dolayısıyla frekans değeri (destek değeri) 2 olmaktadır. Bu destek değeri, asgari destekten küçük olduğu için kabul edilmemektedir.

### Algoritma

Konu üzerindeki ilk yayını yapan Agrawal ve Srikat, tarafınan algoritmanın müsvette kodu (pseudo code) aşağıdaki şekilde verilmiştir.

$L_1 = \{\text{sık tekrarlanan öğeleri içeren küme}\}$

for( $k=2$ ;  $L_{k-1} \neq 0$  ;  $k++$ ) {

$C_k = \text{apriori-üret}(L_{k-1})$ ;

forall (işlemler  $t \in D$ ) {

$C_t \subset (C_k, t)$ ;

forall adaylar  $c \in C_t$  {



```
c.sayaç++;
```

```
}
```

```
}
```

```
 $L_k = \{ c \in C_k \mid c.sayaç = \text{enküçük} \}$ 
```

```
}
```

```
Sonuç =  $\bigcup_k L_k$ ;
```

Algoritmayı açıklayacak olursak, algoritmadaki  $L$ , veri tabanı veya veri kümemizde bulunan öğeleri temsil etmektedir. Apriori üret fonksiyonundan üretilen her bir  $C_k$  kümesi için,  $t$  bir işlem (transaction) olmak üzere, üretilen bu  $C_k$  kümesinin  $t$  işlemini tutan sayacı, her münasebette bir artırıyoruz. Ardından sıradaki  $L$  değerini alarak algoritmamız çalışmaya devam ediyor. Sonuç olarak da üretilen bu  $L$  değerlerinin birleşim kümesini döndürüyoruz.

Yukarıdaki algoritmada belirsiz olan apriori-üret fonksiyonunu ise aşağıdaki şekilde verebiliriz:

```
 $C_k$ 'ya ekle
```

```
select p.eleman1, p.eleman2, ... p.elemank-1, q.elemank-1
```

```
for( $L_{k-1}$  p'den  $L_{k-1}$  q'ya kadar)
```

```
where p.eleman1, p.eleman2, ... p.elemank-2 = q.elemank-2, p.elemank-1 < q.elemank-1
```

```
forall elemanlar  $c \in C_k$  {
```

```
forall  $c$ 'nin  $(k-1)$ -alt kümesi  $s$  için {
```

```
if(  $s \notin L_{k-1}$  ) {
```

```
 $c$ 'yi  $C_k$  'dan sil
```

```
}
```

```
}
```

```
}
```

Yukarıdaki algoritma, basitçe bütün adayların üretilerek elemanların aday kümelerine konulmasını hedeflemektedir.

## **SORU-2: Yeknesan (Invariant , Değişmez) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde, bir programın incelenmesi sırasında, herhangi bir kaziyenin (predicate, haber, önerme), çeşitli işlemler uygulanmasına karşılık yeknesan olması halidir.

Diğer bir deyişle, program çalışır ve çeşitli işlemlerden geçer, ancak bazı şeyler değişmeden kalıyor ve ne kadar işlem yapılırsa yapılsın değişmiyorsa buna yeknesan (değişmez, invariant) ismi verilir.

En basit örneği, bir değişkenin (variable) içine değer atanması ama hiç değiştirilememesidir.

```
int a = 10;
// işlemler
printf("%d",a);
```

Yukarıdaki kodda, a değerini ilgilendiren işlemler yapılmıyorsa, a için yeknesan olmuş denilebilir. Farklı bir örnek de döngülerde görülür:

```
for(int i = 0; i<10; i++){
    printf("%d", i);
}
```

Yukarıdaki döngüde,  $i < 10$  şartında bulunan 10'dan küçük olma şartı yeknesan olmuştur. Yani döngünün dönmesi sırasında değişmeden kalmaktadır. Hemen bu noktada önemli bir farkı belirtmek gerekir. Yeknesan kavramı, sabit kavramı ile (constant) karıştırılabilmektedir. Örneğin yukarıdaki döngüde bulunan 10 sayısı bir sabittir ve yeknesan değildir. 10'dan küçük olma durumu ise değişmeden kalan bir yeknesan olarak düşünülmelidir. Yukarıdaki tip yeknesan örneklerine, özel olarak daire yeknesanı (loop invariant, döngü değişmezi) ismi de verilebilir.

Bu yaklaşım, ayrıca program analizi veya problem analizi sırasında da kullanılabilir. Örneğin MU Probleminin çözümünde, yeknesan bir değer yakalamaya çalışmak gerekir. Benzer şekilde Hoare Mantığı (Hoare Logic), programların belirli bölümlerinde, ön ve son koşullar belirleyerek bunların yeknesan olmasını sağlamaktadır.

Bu yöntem, bilgisayar bilimleri dışında da kullanılır. Örneğin, kanun yapım sürecinde, bütün kanunlarda değişmeyen kötü hasletler vardır. Bu yüzden, hırsızlık veya cinayet gibi suçlar, kanunlar değişmesine karşılık, bütün kanunlarda yerini almaktadır. Bu anlamda, değişen kanun sistemleri için, hırsızlık, yeknesan bir suçtur denilebilir.

### **SORU-3: Algoritma (Algorithm) hakkında açıklama yapınız.**

Bazen biz insanlar için çok kullanılan kelimeler, tanımlanması en güç kelimeler haline dönüşebiliyor. Algoritma da sanırım bilgisayar bilimleri için benzer özellikte olan bir kelime.

Sanırım bu kelimeyi tanımlarken “bir dizi matematiksel adım” ifadesini kullanmak yerinde olur.

Bütün algoritmalar, matematiksel olarak ispatlanabilen ve dizilimi kesinlikle önem taşıyan ve bir metot anlatmasıdır.

Aslında bir kelimeyi başka bir kelime ile ifade etmek bir hatadır. Şayet iki kelime aynı şeyi anlatıyorsa, o zaman bir tanesi fazla demektir ama genelde algoritma için bu “metot” veya “sistem” kelimeleri sıkça kullanılıyor. Oysaki algoritma bunlardan farklı olarak bir, matematiksel bir dizilimdir. Evet, bir problemi çözmek için gerek metodu anlatır bu çözümde kullanılan sistemi gösterir ama atlanmaması gereken ispat edilebilirlik, karmaşıklığının ölçülebilirliği aslında bilgisayar bilimlerinin de temelini oluşturur.

Bu durumu bir örnek üzerinden açıklamaya çalışalım.

Örneğin sık kullanılan sıralama problemini (sorting problem) ele alalım. Elimizde bir dizi karıştırılmış sayı var ve biz bunları küçükten büyüğe doğru sıralamak istiyoruz. Bu durumda birisinin çıkıp ben bunları sıraladım demesi bir algoritma olmaz. Çünkü algoritmanın sistematik olarak probleme yaklaşması gerekir.

Ben bunları sıralayan bir sistem yaptım demesi de algoritma olmaz. Çünkü sistemin bütün adımlarını belirli olması ve analiz edilebilir olması gerekir.

Ben bunları sıralayan ve şu adımlardan oluşan bir sistem yaptım demesi de yeterli olmaz çünkü bu algoritmanın her durumda çalışacağının matematiksel olarak ispatlanabilmesi gerekir.

Ancak bu aşamadan sonra bir algoritmadan bahsediyoruz demektir ve tam da bu aşamadan sonra artık algoritmanın performansından bahsetmeye başlar ve algoritmamızın hafıza ve zaman ihtiyaçlarını ölçebilmeyi isteriz.

Yukarıdaki bütün bu tanımların yanında, bir algoritma elde ettikten sonra ayrıca bu algoritmanın uygulanabilir oluşu da tartışmaya açıktır. Örneğin geliştirilen algoritma, günümüz bilgisayarları için uygulanabilir olmayabilir. Bu durum algoritmamızı , algoritma olmakta çıkarmaz ancak uygulanamaz bir hale sokar (örneğin şu anda yeni yeni gelişen kuantum hesaplama algoritmaları buna birer örnek olabilir).

Ayrıca algoritmaların yaklaşımlarına göre sınıflandırılması da mümkündür. Hatta bu sınıflandırmaya uygun olarak algoritma geliştirilme metodolojileri de bulunmaktadır. Örneğin nesne yönelimli programlama (object oriented programming) , yapısal programlama (structured programming) veya fonksiyonel programlama (functional programming) , Emirli programlama (Imperative Programing), özdevinirli programlama (automat based programming) gibi.

#### **SORU-4: Birbirini Dışlama (Mutually Exclusive) hakkında bilgi veriniz.**

Birbirini dışlama özelliği, birden fazla işin birbiri ile ilişkisizliğini belirtmek için kullanılan bir terimdir. Örneğin iki işlem (process) veya iki lifin (thread) birbirinden bağımsız çalışmasını, aynı anda bir işlemi yapmamasını istediğimiz zaman birbirini dışlama özelliğini kullanabiliriz. Bazı kaynaklarda, kısaca mutex (mutually exclusive kısaltması) olarak da geçer.

İki adet birbirine paralel ilerleyen iş için (concurrent) aynı anda bir kaynağa erişme veya birbirleri için kritik olan işlemler yapma ihtimali her zaman bulunur. Örneğin bilgisayarda çalışan iki ayrı lifin (thread) JAVA dilinde ekrana aynı anda bir şeyler basmaya çalışması veya paylaşılan bir dosyaya aynı anda yazmaya çalışması, eş zamanlı programlamalarda, sıkça karşılaşılan problemlerdendir. Bu problemin çözümü için, işlemlerin senkronize edilmesi gerekir ve temel işletim sistemleri teorisinde 4 yöntem önerilir:

- Koşullu Değişkenler (Conditional Variable)
- Semaforlar (Semaphores)
- Kilitler (Locks)
- Monitörler (Monitors)

Yukarıda sayılan bu 4 yöntem, basitçe iki farklı işi senkronize hale getirmeye yarar. Şayet iki ayrı işin birbirine hiçbir şekilde karışmaması isteniyorsa (mutex) bu durumda çeşitli algoritmaların kullanılmasıyla sistemdeki işlerin ayrılması sağlanabilir. Örneğin aşağıda, iki çok kullanılan algoritmaya bağlantı verilmiştir:

- Dekker Algoritması
- Peterson Algoritması

### **SORU-5: WOLG (Genelliğini Kaybetmeden) hakkında bilgi veriniz.**

Bilgisayar bilimlerinin de temellerini oluşturan matematik'te kullanılan bir tabirdir. İngilizcede “without loss of generality” kelimelerinin baş harflerinden oluşur ve Türkçede “genelliğini kaybetmeden” şeklinde kullanılabilir.

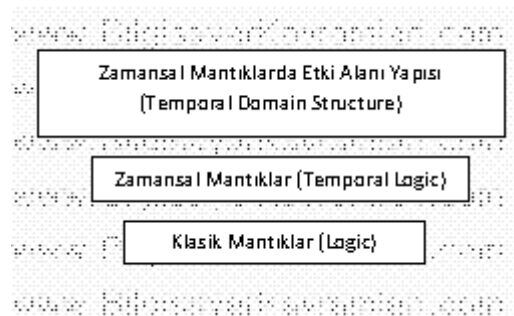
Genelde matematiksel bir ispat yapılması veya bir çıkarım sırasında kullanılır. Buradaki genellik ile kastedilen bir ispattan veya çıkarımdan önceki adımda kabul edilen genel durumun, ispatın sonuna kadar saklanmasıdır.

Örneğin bir ispat sırasında  $x$  ve  $y$  şeklinde iki değişken rastgele olarak alınıyor. (bunlar birer sayı veya sembol de olabilir)

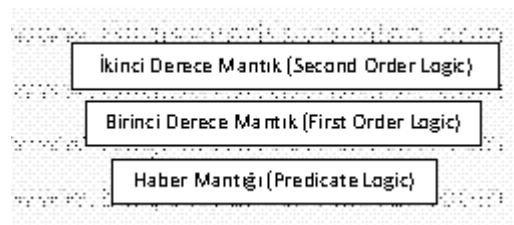
İspat sonucunda örneğin  $F(x,y)$  sonucunun doğru olduğu gösterilirse, genelliğini kaybetmeden  $F(y,x)$  sonucunun da doğru olduğu gösterilmiş olur. Burada ilk olarak seçilen  $x$  ve  $y$  sayılarının rast gele seçildiğini kabul edebilir ve bu rastgelelikten dolayı  $F(x,y)$  veya  $F(y,x)$  sonucunun elde edilmesi bu sayıların seçimindeki genelliğin kaybolmasını gerektirmez.

### **SORU-6: Zamansal Mantıklarda Etki Alanı Yapısı (Temporal Domain Structure) hakkında bilgi veriniz.**

Bilgisayar bilimlerinin özellikle yapay zeka konusu altında yer alan ve zamansal modelleme / problem çözümü konularında kullanılan bir yapıdır. Bu yapıyı, zamansal mantıklar (temporal logic) üzerinde çalışan bir etki alanı sınıflandırması olarak düşünmek mümkündür.



Yukarıdaki şekilde, bu seviyelendirme 3 katmanda gösterilmiştir. Buna göre bir zamansal mantığın (temporal logic) seviyesi, üzerine kurulu olduğu mantık seviyesine bağlıdır. Klasik mantıkları seviyelendirirken 3 seviyeden bahsetmek mümkündür:



Yukarıdaki bu seviyelendirmede, birinci derece mantık (first order logic) için düşük seviye mantık (lower order logic) veya ikinci derece mantığa (second order logic) yüksek seviye mantık (higher order logic) isimleri de verilmektedir. Sonuçta bir zamansal mantık yukarıda gösterilen 3 seviyeden herhangi birisi üzerine kurulu bir mantık olarak düşünülebilir.

Bu durumda bir zamansal mantığın seviyesinden bahsedilmesi demek, bu zamansal mantığın üzerine inşa edildiği mantığın seviyesinden bahsedilmesi demektir.

Bu yazıda incelenecek olan, zamansal mantıkların etki alanı yapısı ise, zamansal mantıklarda ortak olan ve ilişkileri analiz etmeye yarayan bir yaklaşımdır.

Bu yaklaşımda öncelikle temel işlem özelliklerini tanımlayabiliriz.

Örneğin iki zamansal belirleyici (quantifier) arasında tanımlı olan bir ilişkiye (relation) R ismini verecek olursak:

Geçişlilik özelliği (transitivity) :  $\forall xyz. x R y \wedge y R z \Rightarrow x R z$

Yansıma özelliği (nonreflexivity) :  $\forall x. \neg x R x$

Doğrusallık özelliği (linearity) :  $\forall xy. x R y \vee x = y \vee y R x$

Sol doğrusallık (left linearity)

$\forall xyz. y R x \wedge z R x \Rightarrow y R z \vee y = z \vee z R y$

Sağ doğrusallık (right linearity)

$\forall xyz. x R y \wedge x R z \Rightarrow y R z \vee y = z \vee z R y$

Başlama (begin)

$\exists x. \neg \exists y. y R x$

Bitme (end)

$\exists x. \neg \exists y. x R y$

Takip (precedence)  $\Box x.$

$\exists y. y R x$

Yoğunluk (density)  $\forall xy. x R y \Rightarrow \exists z. x R z R y$

Yukarıda sıralanan özellikler bir zamansal mantığın taşıyabileceği özelliklerdir. Genellikle bir zamansal mantıktaki bütün ilişkiler geçişli, yansımasız ve doğrusaldır.

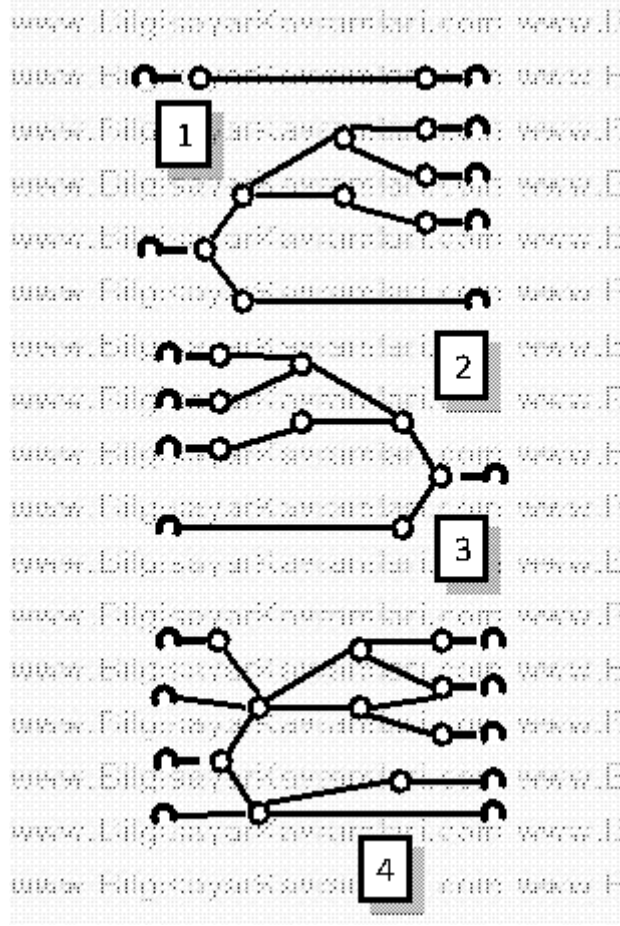
Yukarıdaki özellikleri açıklamaya çalışalım.

Bir zamansal mantığın başlama / bitme özelliğinin bulunması bu mantığın zamanda bir geçmiş ve bir de gelecek limitinin olması anlamına gelir. Tam tersi olarak takip (precedence / successor) ise zamanda sonsuz olduğu anlamına gelir. Yani bir zamansal mantığın geçmişte ve gelecekte bir sonunun olmaması demektir.

Bir zamansal mantığın yoğun olması, iki ilişki arasına üçüncü bir ilişki girebilmesi anlamına gelmektedir. Öte yandan kesikli olması bunun tam tersi şeklinde üçüncü bir ilişkinin, iki ilişki arasına girememesi anlamına gelir.

Yukarıda geçen doğrusallık ise 3 farklı şekilde anlaşılabilir. Zamanı doğrusal bir çizgi üzerine oturtursak iki yöne gidilen doğrusallık ve taraflardan birine gidilen doğrusallık şeklinde anlayabiliriz.

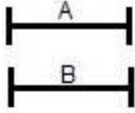
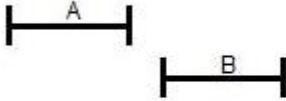
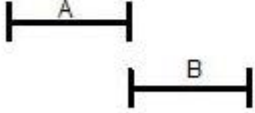
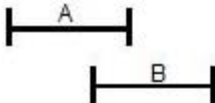
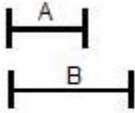
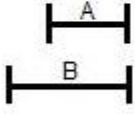
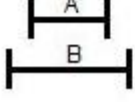
Örneğin aşağıda bu 3 farklı durum için gösterilen doğrusallık örnekleri bulunmaktadır:



Yukarıdaki şekillerden 1 numara ile gösterilen şekilde tam doğrusallıktan yani iki uca doğru giden doğrusallıktan bahsedilebilir. 2 numara ile gösterilen çizimde ise sol doğrusallık gösterilmektedir. Bu gösterimde dikkat edilirse sağ cihette bir doğrusallık bulunmamakta buna mukabil sola doğru bir doğrusal zaman çizgisi gösterilmektedir.

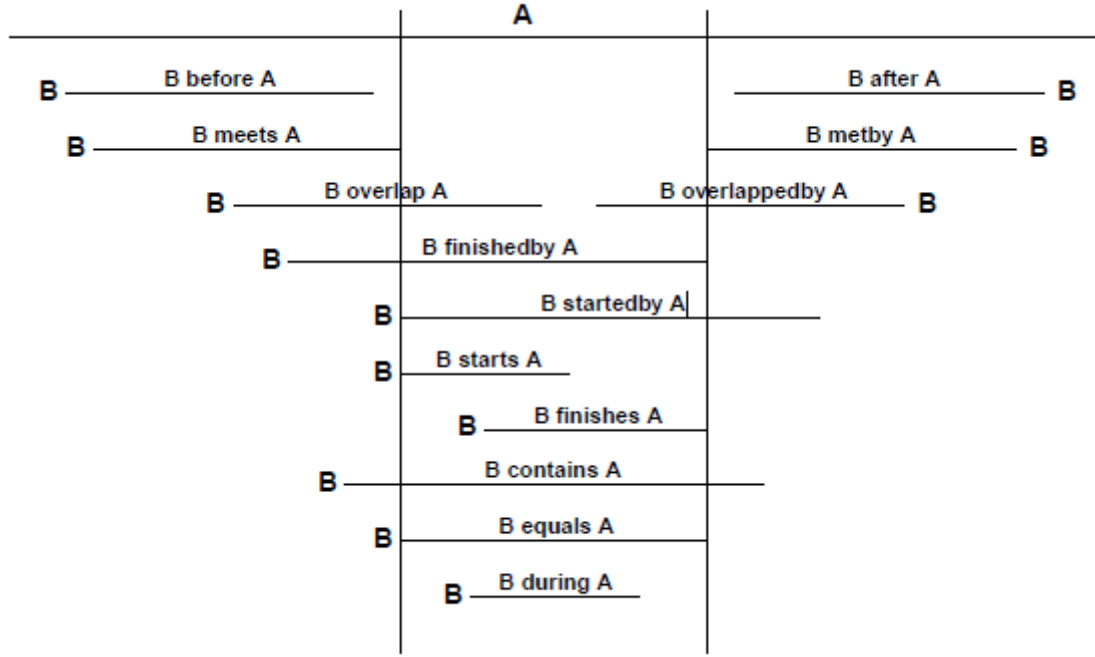
3 numaralı çizim, 2 numaralı çizimin tersine sağda doğrusallığın olduğu ve sol tarafta doğrusallığın olmadığı bir çizimdir. 4 numaralı çizim ise iki tarafta da doğrusallık bulunmayan dolayısıyla doğrusal olmayan çizimdir.

Yukarıda açıklanan zamansal mantık etki alanı yapılarına ilave olarak, zamansal mantıklarda bulunan fasıla işlemleri de ayrıca incelenmelidir. Bir zamansal mantıkta 13 temel fasıla bulunur. Bunlar aşağıdaki şekilde gösterilmiştir:

	A ile B eşit B ile A aynı zamanda
	A, B'den önce B, A'dan sonra
	A'yı B takip eder B, A tarafından izlenir
	A ile B çakışır B, A ile çakışır
	A, B'yi başlatır B, A tarafından başlatılır
	A, B'yi bitirir B, A ile biter
	A esnasında B olur B, A'yı kapsar

Yukarıdaki şekilde görülen bu temel 13 bağlantı hemen hemen bütün zamansal mantıklarda bulunan fasıla durumunu ifade eder.





Yukarıda P. Bellini tarafından yayınlanan ve “Temporal Logics for Real-Time System Specification” başlıklı çalışmadan alınan gösterimde de bir önceki şekilde bulunan çizimin biraz daha değiştirilmiş hali görülebilir.

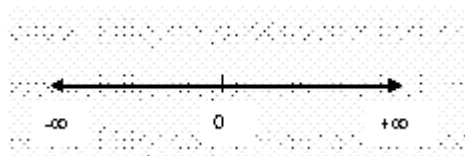
### **SORU-7: Cardinality (Sayısalılık) hakkında bilgi veriniz.**

Algoritma analizi (algorithm analysis) ve hesaplama teorisinde (theory of computation) sıkça kullanılan anlamıyla bir kümenin eleman sayısını belirtir.

Ayrıca matematik ve bilgisayar bilimleri açısından önemli bir yönü de kümelerin sonsuz olması durumunda kümeler arasındaki ilişkinin gösterilmesinde işe yaramasıdır.

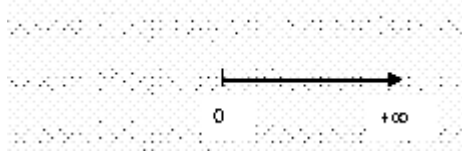
Bu durumu öncelikle sonsuz iki küme tanımlayarak anlamaya çalışalım.

Tam sayılar kümesini ele alalım. Tam sayılar kümesi sonsuz eleman içermektedir. Eksi sonsuzdan 0'a kadar ve 0'dan da artı sonsuza kadar giden sayılardan oluşur.



Dolayısıyla tam sayılar kümesinde, sonsuz eleman olduğunu söylememiz doğrudur.

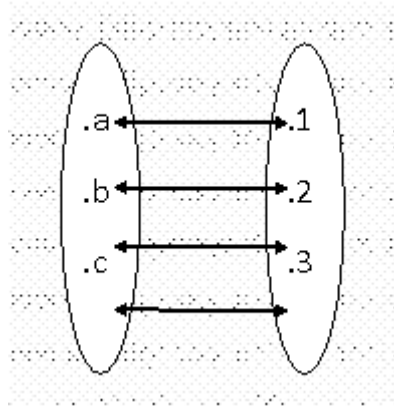
Benzer şekilde pozitif tamsayılar kümesi sıfırdan artı sonsuza kadar sonsuz sayıda eleman içerir:



Bu iki kümenin de sonsuz olduğunu söylediğimize göre gerçekten bu iki kümenin sonsuzlukları aynı mıdır? Yoksa kümelerden birisi daha büyük diğeri daha küçük sonsuz mudur?

Bu soru ile kardinallik (sayısallık, cardinality) konusuna başlayabiliriz. Bu soruyu cevaplamaadan önce iki küme arasında eşit miktarda eleman bulunmasını tanımlamamız gerekir:

Sayısallık açısından iki küme birbirine bağlanmak istenirse, bunun anlamı bu kümeler arasında birebir (one-to-one) ve üzerine (onto) bağlantı bulunması demektir.



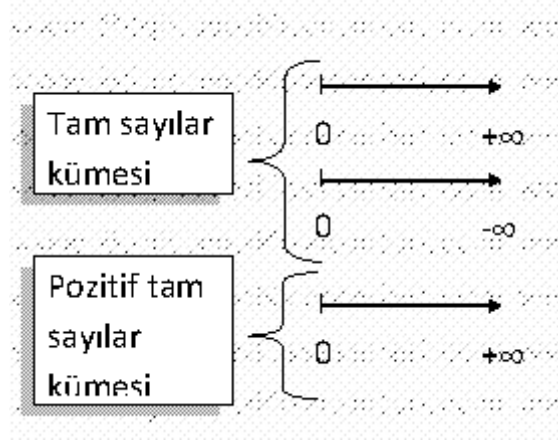
Yukarıdaki örnekte, iki küme arasında bire bir ve örten bir bağlantı gösterilmektedir. Burada birebir olması, birinci kümeden her elemanın ikinci kümeden tek bir elemanı gösteriyor olmasını ve ikinci kümedeki her elemanın da birinci kümede tek elemanı gösteriyor olması demektir. Şayet örneğin “a” elemanı hem 1 hem de 2 ile ilişkilendirilmiş olsaydı bu durumda birebir ilişkiden bahsedilemezdi. Örten olma özelliği (onto) ise boşta hiç eleman kalmamış olması yani bütün elemanların bir şekilde diğer kümede ilişkilendirilmiş olması anlamındadır.

Kümeler teorisi (set theory) ile ilgili bu kısa hatırlatmadan sonra konumuz olan sayısallık ilişkisine dönebiliriz.

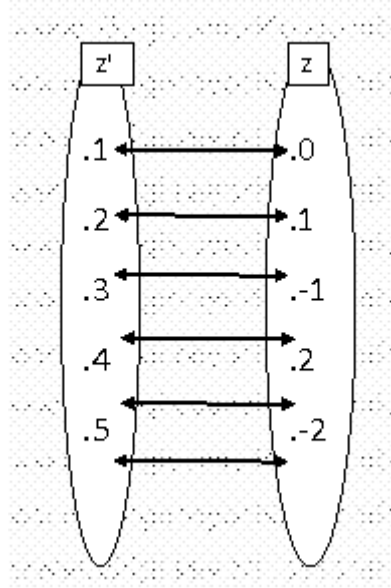
Şayet kümelerin ikisi de yukarıdaki örnekte olduğu gibi sonlu değil de yazının başında verildiği gibi sonsuz ise bu durumda sayısallık bağlantısını nasıl kurabiliriz?

Sayısallık açısından, tam sayılar kümesi ve pozitif tam sayılar kümesinin eşit olduğunu söyleyebiliriz. Bunu göstermek için yukarıda bahsettiğimiz üzere, iki küme arasındaki ilişkinin örten ve birebir olduklarını göstermemiz gerekir.

Bunu daha iyi modelleyebilmek için sonsuzluk yönünü aynı tarafa çevirip ardından sayıları sıralayalım:



Tamsayılar kümesini yukarıdaki şekilde çevirecek olursak kümenin elemanlarını aşağıdaki şekilde yazabiliriz:



Yukarıdaki resimde solda gösterilen pozitif tam sayılar ( $Z^+$ ) kümesi ile sağda gösterilen tamsayılar kümesi ( $Z$ ) arasında birebir ve örten bir ilişki kurulmuş olunur. Bunun sebebi iki kümedeki her eleman diğer kümedeki bir elemene mutlaka ilişkilendirilmiş ve bu ilişki birebir olmuştur.

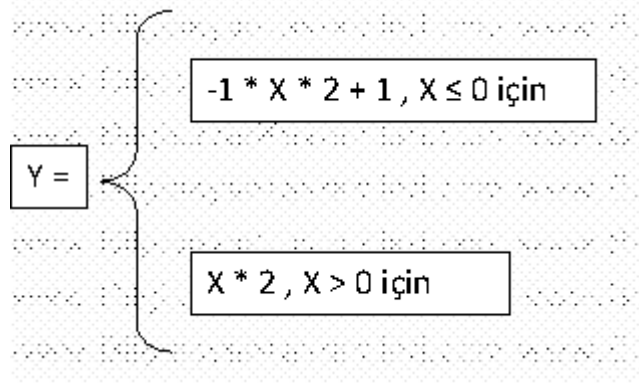
Yukarıdaki bu görsel ilişkiyi matematiksel olarak aşağıdaki şekilde yazmak da mümkündür:

$$X \in Z \text{ ve } Y \in Z^+ \text{ olmak üzere } X = Y / 2 * (-1)^{Y \bmod 2}$$

Yukarıdaki formülde tam sayı bölmesi yapıldığı kabul edilmiştir. (yani noktadan sonraki değerler göz ardı edilmiştir)

Gerçekten de  $Y = 1$  için  $X = 1 / 2 * 1 = 0$  sonucu veya  $Y = 15$  için  $X = 15 / 2 * -1 = -7$  sonucu bulunmaktadır.

Bu bağlantının tersini yazmak da mümkündür:



Örneğin  $X = 0$  için  $Y = -1 * 0 * 2 + 1 = 1$  olarak bulunur veya diğer bir örnek olarak  $X = -7$  için  $Y = -1 * -7 * 2 + 1 = 15$  olarak bulunur.

Görüldüğü üzere iki yönlüde bütün elemanları diğer kümedeki elemanlara bağlayan bir bağlantı yazılması mümkündür.

Böylelikle yukarıdaki şartımızı sağlamış ve iki küme arasında eşit miktarda eleman bulunduğunu göstermiş oluyoruz.

Bu konuyu anlatırken sorulmuş bir soruyu cevaplamak istiyorum: “Kümelerin venn diyagramlarında da, denklemlerin her ikisinde de  $Z$  kümesi,  $Z^+$  kümesine göre daha yavaş ilerliyor. Yani  $Y$  için 16 sayısına ulaşıldığında  $X$  için henüz 8 sayısına yeni ulaşılmış olunuyor. Bu durumda tam sayılar kümesi, pozitif tam sayılar kümesinden daha büyük değil midir?”

Bu soru ilk başta mantıklı gibi görülse de cevabı kesin olarak hayırdır. Bunun sebebi kümelerin sonsuz oluşudur. Yani sonsuzun sonu yoktur 😊 Bu durumu şöyle açıklayalım, yukarıdaki formülde yazılan  $X$  ve  $Y$  sayıları arasında neredeyse  $X=Y/2$  veya  $Y = 2*X$  gibi bir bağlantı vardır. Bu durumda yukarıdaki soruya temel teşkil eden düşüncüyü cevaplamak için sonsuzdaki bir sayıya  $n$  diyerek  $X=n$  için  $Y = 2n$  ‘dir sonucuna ulaşmak ve dolayısıyla  $n$  sonsuz ise  $2n$  sonsuzdan daha büyük bir sonsuzdur gibi bir hataya düşmek mümkündür. Ancak buradaki hata iki türlü cevaplanabilir,

1. Örnekteki  $n$  bir sayı olmak üzere, kümelerdeki son sayının  $n$  olduğunu söyleyemeyiz
2. Sayı kümesindeki  $n$ , sonsuzda bir sayı olmak şartıyla  $2n$ ’in sonsuzda olmadığı söyleyemeyiz.

Dolayısıyla bu iki kümede de eşit sayıda eleman bulunur denilebilir.

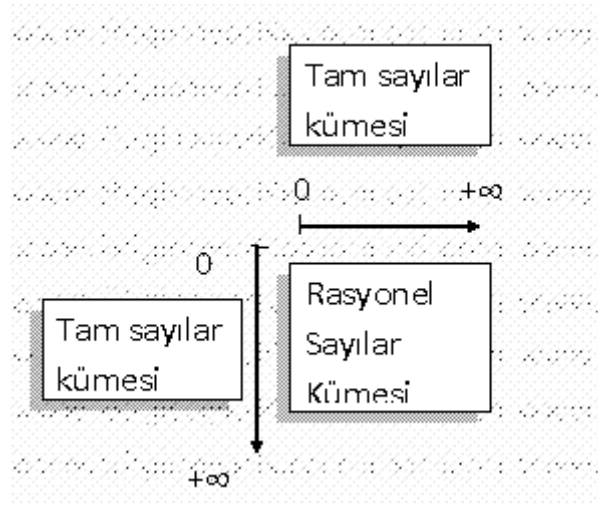
### **Rasyonel sayılar (kesirli sayılar, rational numbers) ile pozitif tam sayılar kümesi arasındaki sayısallık ilişkisi**

Yukarıdaki bu yaklaşım kullanılarak, kesirli sayılar ( rational numbers) kümesinin, tam sayılar kümesi ile aynı miktarda sonsuz eleman içerdiğini de ispatlayabiliriz. İhtiyacımız olan tek şey yukarıdaki örnekte bahsedildiği üzere pozitif tam sayılar kümesi, veya tam sayılar kümesi ile, kesirli sayılar (rasyonel sayılar) arasında bir bağlantı kurabilmek ve bu bağlantıyı modellemektir.

Öncelikle kesirli sayıları, iki boyutlu uzayda modellemekle işe başlayabiliriz. Kesirli sayılar kümesini aşağıdaki tabloda olduğu gibi yazmak mümkündür:

	1	2	3	.	n
1	1/1	1/2	1/3	.	1/n
2	2/1	2/2	2/3	.	
3	3/1	3/2	3/3	.	
.	.	.	.	.	
n	.	.	.	.	n/n

Yukarıdaki tabloda görüldüğü üzere ilk satır ve ilk sütun birer pozitif tam sayı içermek üzere, tablonun diğer elemanları bu iki sayının (üsttekinin soldakine) bölümüdür. Dolayısıyla aslında problemi aşağıdaki şekilde pozitif tamsayılar olarak modellemiş oluruz:



Diğer bir deyişle şimdilik elimizde  $(Z+ \times Z+) \leftrightarrow Q+$  şeklinde bir bağlantı bulunmaktadır. Bu bağlantıyı biraz daha ilerleterek  $Z \leftrightarrow Q$  şeklinde çevirmeye çalışalım. Çünkü bu şekilde bir bağlantı bizim bu iki kümedeki eleman sayılarının eşit olduğunu kardinallik açısından ispatlamamız olacaktır.

	1	2	3	.	n
1	1/1	1/2	1/3	.	1/n
2	2/1	2/2	2/3	.	
3	3/1	3/2	3/3	.	
.	.	.	.	.	
n	.	.	.	.	n/n

Yukarıdaki şekilde gösterilen ok yönlerinde bu sayıları, pozitif tam sayılar kümesi ile ilişkilendirirsek hem bir bağlantı yakalamış hem de birebir ve örten bir ilişki elde etmiş oluruz. Bu durumda sayılarımızı aşağıdaki şekilde ilişkilendireceğiz:

$Z+$	$Q+$
1	1/1

2	2/1
3	1/2
4	3/1
5	1/3
6	4/1
7	3/2
...	...

Yukarıda görüldüğü üzere, tam sayılar kümesindeki bütün sayılar, rasyonel sayılar kümesi ile ilişkilendirilmiştir. Ayrıca dikkat edilmesi gereken bir husus, 2/2 veya 3/3 gibi tam sayı karşılığı 1 olan ve aslında eşit olan sayıların bu ilişkide yer almamasıdır. Bu sayıların ilişkiye yerleştirilmesi durumunda birebir olma özelliği bozulacaktır.

Yukarıdaki gösterim sayesinde, bu iki kümenin, yani rasyonel sayılar kümesi ile tam sayılar kümesi arasında bir ilişki birebir ve örten olarak kurulmuş ve bu sayede bu iki kümenin eleman sayısının eşit olduğu ve dolayısıyla bu iki kümenin sayısallık (kardinallik, cardinality) açısından eşit olduğu söylenebilir.

### Sayılabilirlik (countability)

Diğer bir deyişle rasyonel sayılar kümesi sayılabilir bir kümedir (countable set) denilebilir. Aslında sayılabilirliğin (countability) tanımı, bir kümenin pozitif tam sayılar kümesi ile aynı kardinalliğe getirilebilmesidir. Yani herhangi bir kümeyi, yukarıdaki örneklere benzer şekilde pozitif tam sayılar ile birebir ve örten bir şekilde ilişkilendirebiliyorsak bu küme sayılabilir kümedir, aksi halde sayılamaz bir kümedir (uncountable set).

Bu sayılamazlık durumunu ve dolayısıyla kardinallik açısından eşit olmayan iki sonsuz kümeyi aşağıda örneklemeye çalışalım. Bu sefer kullanacağımız örnek gerçek sayılar (real numbers, reel sayılar) ile tam sayılar kümesi (integers) olsun.

### Tam sayılar ve Reel sayılar arasında sayısallık ilişkisi

Bu örnekte, tam sayılar kümesi ve reel sayılar kümesi arasında sayısallık açısından eşitlik olmadığını ispatlamaya çalışalım.

Öncelikle ispatımızı, olmayana ergi (proof by contradiction) olarak modelliyoruz. Yani öncelikle reel sayılar kümesinin sayılabilir olduğunu kabul edip, bu sayı kümesi ile pozitif tam sayılar kümesi arasında bir ilişki kuracağız. Ardından bu ilişkinin kurulamayacağını ispatlamaya çalışacağız.

Öncelikle ilişki gösterimini ve ispatı daha iyi anlayabilmek için bu ispatta ikilik tabanda sayılar kullanalım. Yani 2 sayısı ikilik tabanda 10 olarak gösterilir. Dolayısıyla 0.2 sayısı 0.10 olarak gösterilir. Benzer şekilde 0.15438 sayısı 0.11110001001110 olarak gösterilecektir. Aslında bu gösterimin herhangi özel bir önemli olmamakla birlikte ispatı daha görülür kılmak için kullanacağız.

Şimdi doğru çalışan bir ilişkimiz olduğunu kabul edelim ve aşağıdaki şekilde gösterildiği gibi, tam sayılar ile reel sayılar kümesini ilişkilendirelim:

Z+	Q+
1	0.110101010100101...
2	0.010101010110101...
3	0.100100100101110...
4	0.001010101010001...
5	0.111110101010000...
6	0.111000011111111...
...	...

Yukarıdaki şekilde görüldüğü üzere,  $Z^+ \leftrightarrow Q^+$  ilişkisi kurulmuştur. Bu ilişkide herhangi bir matematiksel bağlantı bulunmadığı gibi sonuçta bütün ilişkilerde yukarıdakine benzer bir tablo ortaya çıkacaktır. Okuyucu birazdan yapacağımız ispatı, farklı sayılar veya herhangi bir matematiksel bağlantı için de deneyebilir sonuç değişmez.

Yukarıdaki şekilde tamsayılar ve reel sayılar arasında bir ilişki kurduktan sonra, kantör (Georg Cantor) tarafından ilk kez ispatlanan sayılamazlık durumu, köşegende (diagon) bulunan sayıların ters çevrilmesi ile elde edilir.

Yukarıdaki her sayının kaçınıcı satırdaysa, o bitini alarak yeni bir sayı oluşturuyoruz.

Z+	Q+
1	0.110101010100101...
2	0.010101010110101...
3	0.100100100101110...
4	0.001010101010001...
5	0.111110101010000...
6	0.111000011111111...
...	...

Yukarıda, siyah renkle gösterilen bu sayılardan oluşan yeni sayımız: 0.110010... olarak belirlenebilir. Şimdi bu sayıdaki bütün bitlerin tersini alalım : 0.001101... olarak yeni bir sayı bulunacaktır.

Şimdi bu iki kümenin eşit olmadığını gösteren sorumuzu sorabiliriz, bulduğumuz bu yeni sayı, acaba yukarıdaki ilişkide hangi satırda yer almaktadır?

Cevap: Hiçbir satırda.

Bunun sebebi, her satırdan 1 bit alınarak oluşturulan yukarıdaki örnek sayının, tersi alındığında, her satırdan alınan bu bitin tersi alınmış ve dolayısıyla her satıra denk gelen bitin tersinden oluşan yeni bir sayı üretilmiş olmasıdır. Dolayısıyla hiçbir satırda ürettiğimiz bu yeni sayı bulunamaz.

Öyleyse,  $Z^+$  kümesi ile  $Q^+$  kümesini bağlamak için hangi bağlantı kullanılırsa kullanılsın, nasıl bir sayma yöntemi geliştirilirse geliştirilsin sonuçta yukarıdakine benzer bir durum olacak ve bu ilişkide bulunmayan bir reel sayı üretilerek iki küme arasındaki örten (onto) bağlantısı bozulacak ve reel sayılar kümesinde, tam sayılar kümesi ile ilişkilendirilmemiş bir eleman bulunacaktır.

Bu durumda reel sayılar kümesinin, tam sayılar kümesinden daha büyük olduğunu ve dolayısıyla sayılamaz olduğunu söyleyebiliriz.

### **SORU-8: İkinci Derece Mantık (Second Order Logic) hakkında bilgi veriniz.**

Bilgisayar bilimleri de dahil olmak üzere, felsefe, matematik veya dilbilim gibi pek çok alanda kullanılan bir mantık modellemesinin ismidir. Buna göre modellenecek sistemdeki varlıklar ve bu varlıklar üzerine uygulanacak olan işlemler ayrı ayrı ele alınır ve sistemde modellenen hem varlıkların hem de işlemlerin geçerli olduğu bir etki alanından söz edilebilir.

İkinci derece mantığı anlamak için basitçe birinci derece mantık (first order predicate calculus, first order logic) ile farkını anlamamız yeterlidir.

Birinci derece mantıkta, klasik mantıktan farklı olarak niceleyiciler (quantifiers) kullanılmakta ve bu niceleyicilerin geçerli olduğu bir etki alanından (domain) bahsetmek mümkündür. Ancak bu niceleyiciler üzerindeki tanımları ve işlemleri ifade eden haberler (predicates) üzerinde herhangi belirleyici bir işlem yapılamıyordu.

Örneğin aşağıdaki birinci derece mantıkta modellenmiş iki gösterimi ele alalım:

$\exists a (\text{Şadi}(a) \wedge \text{Mühendis}(a)) \rightarrow$  Öyle bazı a'lar vardır ki, bu a'nın ismi Şadi'dir ve bu a mühendistir.

$\forall a (\text{Bilgisayar Mühendisi}(a) \Rightarrow \text{Mühendis}(a)) \rightarrow$  Bütün a'lar için, a şayet bilgisayar mühendisiyse; a, aynı zamanda mühendistir.

Yukarıdaki bu gösterimlerde dikkat edileceği üzere niceleyici (quantifier) olan a üzerinde bir etki alanı tanımlanmıştır buna mukabil haberler (predicates) üzerinde bir etki alanı tanımlanmamış ve bilgisayar mühendisi tanımı olduğu gibi bırakılmıştır. İşte ikinci derece mantık tam burada devreye girer ve bu haberlerin (predicates) üzerinde de etki alanı tanımını mümkün kılar.

Bu durumu aşağıdaki örnek üzerinden anlamaya çalışalım:

$\exists \text{Şekil} (\text{Şekil}(a) \wedge \text{Şekil}(b)) \rightarrow$  Öyle bir şekilden bahsedilebilir ki, hem a hem de b aynı şekildedir.

Yukarıdaki bu durum birinci seviyede yazılamaz. Bunun sebebi şekil belirten haberin (predicate) her durumda aynı olması gerekliliğidir. Diğer bir deyişle birinci seviye mantıkta tek bir şekilden bahsedilebilir, özel bir şekil olamaz.

Bu durumu biraz daha ileri götürürsek haberler (predicates) üzerinde de haberler tanımlamak mümkündür. Yani haberin haberi şeklinde bu durumu modelleyebiliriz.

$\exists \text{Şekil} ( \text{Köşeli} ( \text{Şekil} ) \wedge \text{Şekil}(a) \wedge \text{Şekil}(b) )$

Yukarıdaki bu tanımda, bir şeklin köşeli olacağı, ve bu durumda a ve b niceleyicilerinin bu şekilden olabileceği tanımı yapılmıştır.

Bu anlamda birinci seviye mantıktan daha kapsamlı modellemeler yapılabilir.



## SORU-9: Birinci Derece Mantık (First Order Logic) hakkında bilgi veriniz.

Bilgisayar bilimlerinin de aralarında bulunduğu, başta felsefe olmak üzere, matematik ve dilbilim gibi alanlarda kullanılan bir mantık modelidir. Bu modelin özelliği kıyas ve tümden gelim yaklaşımına göre ispatlanabilir özellikte olmasıdır (deductive). Yani bir makine, veya matematiksel model tarafından bu mantık modelindeki gösterimlerin ispatlanması (verification) veya reddedilmesi (falsification) mümkündür.

Birinci derece mantık, literatürde çeşitli isimlerle adlandırılmaktadır. Örneğin, mantığın üzerine inşa edildiği yüklem ve haberlerden (predicate) ismini alan “birinci derece haber mantığı” (first order predicate logic) veya bu mantık üzerinde matematiksel işlemlere izin vermesinden dolayı birinci derece yüklem aritmetiği (first order predicate calculus) veya yine birinci derece kelimesinin sebebi olan ve mantıkta kaziyeler mantığına (önermeler, propositional logic) yakın olması dolayısıyla alçak haber mantığı (lower predicate calculus) isimleri gibi farklı isimler verilmektedir.

Birinci derece mantığın, klasik kaziyeler mantığından (önermeler mantığı, propositional logic) en önemli farkı, bu mantıkta kullanılan niceleyicilerin (quantifiers) veya bilgisayar bilimleri gözüyle değişkenlerin (variable) belirli bir tanım alanlarının olmasıdır (domain). Dolayısıyla birinci derece mantıkta yapılan modellemelerde kullanılan bütün niceleyiciler tanımlı bir alanda geçerlidir ve mantık modellerimiz bu alan için doğru veya yanlış olarak işlenir.

Bu tanımlı olma özelliğinin getirdiği bir zafiyet olarak, birinci derece mantıkta sonuz kümeler olan, doğal sayılar veya Kartezyen uzay gibi varlıkların modellenmesi ve çözümlenmesi mümkün olmamaktadır. Bu tür sonsuzluk içeren modellemeler için ikinci derece mantık (second order logic) benzeri daha kuvvetli mantık modellemelerinden yararlanmak mümkündür.

Haber mantığından birinci derece mantığa geçiş

Birinci derece mantık konusunu anlamak için öncelikle klasik, kaziyeler mantığının (propositional logic) nasıl çalıştığını hızlıca hatırlayalım.

Örneğin “Şadi mühendistir.” cümlesi, klasik mantık açısından bir kaziyedir (önermedir) ve doğru veya yanlıştır.

Benzer şekilde “Ali mühendistir.” Cümlesi de klasik mantık açısından bir kaziyedir ve tek başına ele alınarak doğruluğu veya yanlışlığı ortaya konulabilir.

Ancak klasik kaziyeler mantığında bu iki ayrı cümle (kaziyeler veya önermeler) yine bu mantıkta tanımlı olan klasik işlemler (operators) dışında birleştirme ve modelleme yapılamaz.

Örneğin bu önermelerden birisine  $p$  ve diğerine  $q$  sembolü verilip mantıkta bulunan aşağıdaki durumlar modellenebilir:

$p \Rightarrow q$  (Şadi mühendis ise Ali de mühendistir)

$p \wedge q$  (Şadi ve Ali de mühendistir)

$p \vee q$  (Şadi veya Ali mühendistir)

$\neg p$  (Şadi mühendis değildir)

Yukarıdaki bu modeller sonuçta doğru (True (T) ) veya yanlış (False (F,  $\perp$  )) sonuçlarından birisini üretir.

Birinci derece mantık, yukarıdaki bu modellemeyi öncelikle niceleyiciler (quantifier) üzerine taşır. Bu niceleyiciler, bilgisayar bilimlerindeki değişkenler (variables) olarak düşünülebilirler ve kabaca bir varlığa verilen ve mantık modelinde kullanılan isim olarak tanımlanabilir.

Yukarıdaki “Şadi mühendistir” cümlesini ele alacak olursak birinci derece mantıkta aşağıdaki şekilde göstermek mümkündür:

Şadi (a)  $\Rightarrow$  Mühendis(a)

Modeli okurken “öyle bir ‘a’ vardır ki bu ‘a’ın ismi Şadi’dir ve bu ‘a’ mühendistir” şeklinde okuyabiliriz. Burada kullanılan a bir niceleyicidir (quantifier) ve bu niceleyici üzerinde bir haber(predicate) bildirilmiştir. Bu modelde, klasik modele göre fark net bir şekilde görülmektedir. Tam bu noktada klasik mantıkta sormayacağımız ama birinci derece mantıkta sorulması gereken “bu a nasıl bir a’dır?” sorusu farkı ortaya koyar. Birinci derece mantıkta bu sorunun anlamı, bu bütün a’lar için geçerli midir, anlamındadır. Yukarıdaki tanımda da anlatıldığı üzere birinci derece mantıkta niceleyicilerin (quantifiers) bir etki alanının olması söz konusudur. Yani a ile sembolize edilen bu gösterici acaba her durum için geçerli bir sembol müdür?

Bu sorunun cevabını farkı anlayabilmek için açık bir şekilde hayır diye verebiliriz. Yukarıdaki a niceleyicisi her durumda doğru olsaydı, bütün ismi Şadi olanlar mühendistir sonucuna varılması gerekirdi ki bu doğru değildir.

İşte bu ayrım bize birinci derece mantıkta kullanılan iki önemli sembolü gösterir. Bunlardan birincisi “her” anlamına gelen ( $\forall$ ) sembol iken diğeri “öyle bazı” şeklinde okunabilecek ( $\exists$ ) semboldür.

Yukarıdaki örneğimizi düzenleyecek olursak aşağıdaki gösterim doğrudur ve sonucu da doğrudur:

$\exists a$  (Şadi (a)  $\wedge$  Mühendis(a))  $\rightarrow$  Öyle bazı a’lar vardır ki, bu a’nın ismi Şadi’dir ve bu a mühendistir.

$\forall a$  (Bilgisayar Mühendisi (a)  $\Rightarrow$  Mühendis(a))  $\rightarrow$  Bütün a’lar için, a şayet bilgisayar mühendisiyse; a, aynı zamanda mühendistir.

Yine, yukarıdaki gösterim kullanılarak söylenebilecek doğru bir model de aşağıdaki gibi olabilir:

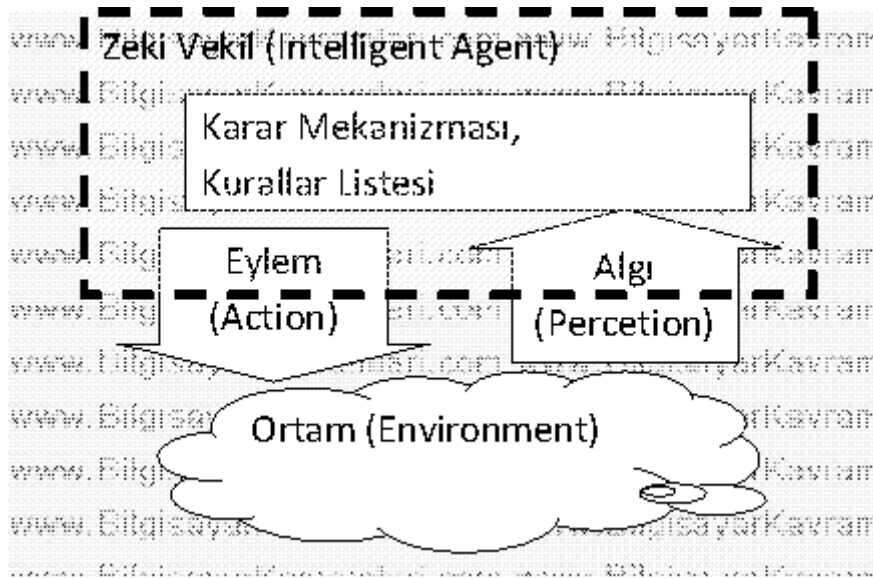
$\exists a$  (Şadi (a)  $\wedge \neg$  Mühendis(a))  $\rightarrow$  Öyle bazı a’lar vardır ki, bu a’nın ismi Şadi’dir ve bu a mühendis değildir.

**SORU-10: Zeki Vekiller (Akıllı Ajanlar, Intelligent Agents, Zeki Etmenler ) hakkında bilgi veriniz.**

Zeki vekiller (etmenler, ajanlar) kavram olarak, bilgisayar bilimlerine, felsefe, biyoloji ve ekonomi alanındaki çalışmalardan sonra girmiştir. Bu alanlardaki anlamı ve kullanımı, genellikle herhangi bir işin farklı bir vekil tarafından yürütülmesi olarak anlaşılabilir.

Bilgisayar bilimlerin açısından zeki kelimesi, bir vekilin herhangi bir işlemi belirli inisiyatifler kullanarak yerine getirmesidir. Örneğin zeki olmayan bir vekil, her adımda ve her işlemde kullanıcıya bir şeyler sorarken, zeki vekilde daha çok otonom bir yapıdan (autonomous) bahsedilebilir.

Bu anlamda her zeki vekilin (intelligent agent) , çalıştığı ortam ile iletişimini sağladığı ve bu iletişim üzerinde karar verdiği bir mekanizması vardır denilebilir.



Yukarıdaki temsili resimde, bir zeki vekil, çizgili alanda gösterilen üç ana unsurdan ibarettir. Bunlar kısaca vekilin çalıştığı ortamı gözlediği algı (perception) , vekilin bu ortamda bir işlem yapmasını sağlayan eylem (action) ve vekilin bu ortamdaki algısına göre nasıl bir eylem yapacağına karar vermesini sağlayan karar mekanizmasıdır. Bu karar mekanizması çoğu zaman bir kurallar listesi (rule base) olabileceği gibi bazı durumlarda basit bir if – else bloğu da olabilmektedir.

Russel ve Norving tarafından 2003 yılında yayınlanan yapay zeka kitabında, zeki vekiller 5 seviyede listelenmiştir. Bu seviyeleri basitten karmaşığa doğru aşağıdaki şekilde sıralayabiliriz:

1. Fiil-i Münakıs Vekiller (reflex agents)
2. Fiil-i Münakıs Kalıp Vekiller (Model-based reflex agents)
3. Hedef vekilleri ( goal-based agents)
4. Fayda vekilleri ( utility-based agents)
5. Öğrenen vekiller ( learning agents)

Reflex Agents

Basit bir koşul ve eylem sıralamasından ibaret olan vekiller. Belirlenen koşul gerçekleşince yine daha önceden belirlenen fiili yerine getirir. Kurulu bir düzenek olarak düşünülebilir. Örneğin fare kapanı, bir insan için fareyi yakalayan bir vekildir ve farenin peyniri yemesiyle birlikte fareyi yakalar. Buradaki peynir yenmesi koşul ve farenin yakalanması fiil olarak düşünülebilir. Bazı refleks ajanlarında durum takibi de yapılabilir. Örneğin fare kapanı misalinde olduğu gibi kapanın kurulu olma durumu, kapanın fareyi yakalamış olma durumu gibi durumlar ayrı ayrı tahlil edilebilir.

#### Model Bazlı Refleks etkenler

Bu tip ajanlarda (etkenlerde) ise içinde çalışılan ortam modellenir. Yani ajan kendi yapısına göre ortamı anlamaya ve bir modelini kendi hafızasında tutmaya çalışır. Bu ajanlar modeldeki durumlara göre davranış sergilerler. Yani bir önceki tipte olan refleks ajanlarının ortamdan aldıkları doğrudan koşullarından farklı olarak bu ajanların modellerinde bazı refleksler tanımlıdır.

#### Hedef Güdümlü Vekiller

Bu vekiller ise belirli bir hedefe ulaşmak için bir dizi şart-fiil gerçekleştirirler. Basit bir durum-geçiş diyagramı (state transition diagram) olarak düşünülebilecek yapılarına göre, ortamı algılayarak mevcut yapılarındaki bir duruma benzetir ve bu durumu ulaşmak istedikleri hedefe en uygun şekilde eylemlerle değiştirmeye çalışır.

#### Çıkar Amaçlı Vekiller

Hedef amaçlı vekillerden farklı olarak, durumlar arasındaki geçişin oransal olması durumudur. Yani hedef güdümlü ajanlarda bir durumdan her zaman diğer duruma geçiş hedeflenir. Çıkar amaçlı vekillerde bundan farklı olarak oransal bir fonksiyon kullanılması söz konusudur. Bu fonksiyona çıkar fonksiyonu ( fayda fonksiyonu, utility function) ismi verilir.

#### Öğrenen Etkenler

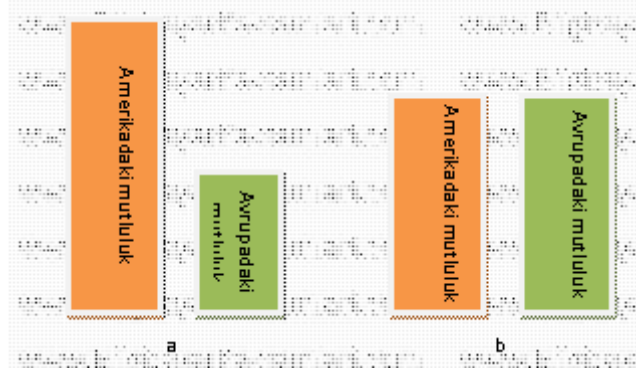
Bu etken tipinde, ortamda yapılan bazı eylemlerin beklenen sonuca nasıl hizmet ettiğine göre yeni kurallar tanımlanır. Ajanın çalıştığı ortamın bilinmemesi halinde kullanılışıdır. Kendi kurallarını ve durum makinelerini oluşturabilir veya değiştirebilirler.

#### **SORU-11: Mere Paradoksu (Mere's Paradox) hakkında bilgi veriniz.**

Olasılık teorisinde kullanılan ve bir toplumun nüfusu ve yaşam standartları arasında bir çelişki oluşturan paradokstur. Paradoksun tanımında, bir toplumun refah seviyesinin yaşamaya yetecek kadar olması (yani olabilecek en düşük seviyede olması) ve nüfusunun azami seviyede olması (olabilecek en çok sayıda olması), topluluğun az nüfusa ve yüksek refah seviyesine sahip olmasına göre tercih edilir.

Kısaca kalabalık ama kalitesiz bir nüfus, az ama müreffeh bir nüfusa yeğ tutulur.

Paradoksu görsel olarak ifade etmek gerekirse, mere aşağıdaki şekilde iki çizelgeden yararlanır:



Yukarıdaki şekilde solda gösterilen (a), Amerika kıtası keşfedilmeden önce, Amerikan yerlilerinin ve avrupadaki insanların ayrı ayrı mutluluk oranlarıdır. Yani Amerikan yerlileri, avrupadaki insanlara göre çok daha mutludurlar.

Şeklin sağ tarafında gösterilen (b) çubuklar ise, amerikanın keşfinden sonra ulaşılan ortak mutluluk seviyesidir. Buna göre amerikadaki mutluluk azalmış ve avrupadaki mutluluk artmış ve daha büyük bir topluluğa ulaşılırken Amerikan halkı daha mutsuz olmuştur.

Mere paradoksu tam bu noktada devreye girer ve sonuçta ulaşılan sağ şekildeki toplam insanların ortalama mutluluğunun sol şekildeki göre daha yüksek olduğunu gösterir.

### **SORU-12: Güvercin Yuvası Kaidesi (Pigeonhole Principle) hakkında bilgi veriniz.**

Bilgisayar bilimleri de dahil olmak üzere pek çok matematik temelli bilim ve mühendislik alanında kullanılan oldukça basit bir umdedir. İsmi güvercin yuvalarından alan bu kaideye göre yuva sayısından fazla güvercin varsa, ve bütün güvercinler bir yuvaya girecekse, en az bir yuvaya birden fazla güvercin girmek zorundadır.

Bu durumu sembollerle göstermemiz gerekirse  $n$  tane yuva ve  $m$  tane güvercin için  $m > n$  durumunda en az bir yuvada birden fazla güvercin bulunmalıdır.

Bilgisayar bilimlerinde çok sayıda sonsuz küme ile uğraşırken oldukça kullanışlı olan bu kaide, örneğin özetleme fonksiyonlarında (hashing function) bir çakışma (collision) ihtimalinin hesaplanmasında ya da kayıpsız sıkıştırma algoritmasının (lossless compression algorithm) istatistiksel analizinde kullanılabilir.

**Örnek:**  $N$  adet pozitif tam sayı arasında en az iki sayının farkı  $N-1$ 'e tam bölünebilir.

Yukarıdaki iddiayı (hipotez) güvercin yuvası prensibi ile ispatlayalım. Öncelikle  $N$  adet tam sayımız olduğunu biliyoruz. Bu sayılara  $a_1, a_2, \dots, a_N$  diyelim ve herhangi bir  $a_i$  sayısı için  $N-1$  bölümünden kalan değerini  $r_i$  diyelim. Yani  $r_i = a_i \bmod (N-1)$  olsun.

Şimdi güvercin yuvası prensibini kullanıyoruz ve diyoruz ki elimizde en fazla  $N-1$  tane farklı kalan değeri olabilir. Yani  $r_i$  değerinden en fazla  $N-1$  tane olabilir. Sebebi ise basit,  $\bmod(N-1)$  içinde en fazla  $N-1$  farklı sayı olabilir. Peki elimizde kaç sayı var?  $N$  sayı. Dolayısıyla en az iki sayının bölümünden kalan değer aynı olacaktır. Diğer bir değişle  $N$  farklı sayının  $N-1$  farklı kalana bölünmesi durumunda en az iki sayı aynı kalan sahip olacaktır. Bu sayıların farkı da aynı kalana sahip olacağından iddiamızı ispatlamış oluruz.

**Örnek:** N pozitif tam sayı için, sayıların bir kısmının yada tamamının toplamı N ile kalansız bölünebilir.

Bu iddianın ispatında da sayıları matematiksel olarak modelleyelim. Öncelikle toplamı b ile ifade edersek,  $b_1 = (a_1) \bmod(N)$ ,  $b_2 = (a_1 + a_2) \bmod(N)$ ,  $b_3 = (a_1 + a_2 + a_3) \bmod(N)$ , ...,  $b_N = (a_1 + \dots + a_N) \bmod(N)$  olarak göstermek mümkündür. Şayet sayılardan birisi 0 ise, yani b değerlerinden birisinin 0 olduğunu bulabiliyorsak, bu durumda zaten ispat yapılmış olur. Çünkü iddiamızda N ile kalansız bölünebileceğinden bahsediyorduk.

Şayet sayılardan birisi 0 değilse bu durumda toplamı elde ettiğimiz sayıların seçiminde  $a_1$  'den başlayarak  $a_i$  gibi bir sayıya kadar olanları almak yerine  $(a_{i+1} + \dots + a_j) \bmod(N) = 0$ , olduğunu gösterebiliriz.

Gerçekten de mod N işlemi sonucunda N-1 farklı sayı çıkacağını biliyoruz. Elimizde N sayı bulunduğunu düşünürsek o zaman yukarıdaki gösterime göre en az iki tane b değeri aynı olmalıdır. Örneğin bu değerler  $b_i = b_j$  olsun ve  $i < j$  olduğunu kabul edelim. Bu durumda  $(a_{i+1} + \dots + a_j) \bmod(N) = 0$  olduğu gösterilmiş olur.

### **SORU-13: Satrançta Büyük Usta Problemi hakkında bilgi veriniz.**

Bilgisayar bilimlerinde özellikle veri güvenliği konusunda kullanılan sıfır bilgi ispatı (zero knowledge proof) örneklerinden birisidir. Orijinal ismi chess grandmaster problem olarak geçer.

Problemi basitçe şu şekilde tanımlayabiliriz. Bir gün satrancı hiç bilmeyen (veya büyük usta seviyesinde bilmeyen) bir oyuncu kalkıp size bir büyük ustayı yeneceğini veya en azından beraber kalacağını söylerse bu inanması oldukça güç bir durum olur.

Bu yazı şadi evren şeker tarafından yazılmış ve bilgisayar kavramları.com sitesinde yayınlanmıştır. Bu içeriğin kopyalanması veya farklı bir sitede yayınlanması hırsızlıktır ve telif hakları yasa gereği suçtur.

Ancak bu iddia aslında mümkündür. Burada iddiada bulunan kişi bir kurnazlık yapıp kaç maç yapacağını söylememiştir. Örneğin tek maç oynayıp yeneceğini iddia etseydi bunun imkansız olduğunu söylemekte serbesttiniz ancak maç sayısı belirsiz olan bu iddiada, iddia sahibi kişi aynı anda iki büyük usta ile oynayarak en azından beraberliği garantileyebilir.

Satrancı hiç bilmeyen acemi oyuncumuz birinci ustanın yaptığı hamleleri ikinci ustayla oynadığı oyunda tekrar ederek ve ikinci ustanın hamlelerini de birinci ustanın maçında tekrar ederek aslında iki büyük ustanın karşılıklı oynamasına aracılık etmiş olur. İhtimalleri değerlendirecek olursak

- Maçı kaybedip 2. Maçı kazanabilir
- Maçı kaybedip 1.maçı kazanabilir
- Maçlar berabere bitebilir

Her üç durumda da satrancı bilmeyen acemi oyuncu iddiasını kazanmış olur.

Yukarıdaki örnek bir sıfır bilgi ispatıdır ve dolev-yao veya man in middle (ortadaki adam saldırısı) gibi saldırı modellerine örnek teşkil edebilir.

#### **SORU-14: Matematiksel Tümevarımın ikinci Teoremi (Second principle of mathematical induction) hakkında bilgi veriniz.**

Temel olarak matematiksel tümevarımın kullandığı yaklaşıma benzer. Bir farkı ispatı bir seri veya seri üreten bir fonksiyon üzerinden değil de ayrı ayrı örnekler üzerine bina etmesidir. Yani matematiksel tümevarım yönteminde aşağıdaki şekilde bir yazım mümkündür:

$$\sum_{i=1}^n f(i) = P(n)$$

Burada dikkat edilirse sayıların belirli bir üretici fonksiyondan çıkması ve sonucun bir fonksiyonda toparlanabilmesi hedeflenir. Dolayısıyla toplamın bir önceki terimi veya bir sonraki terimi bir fonksiyondan çıkar. Ancak bunun mümkün olmadığı durumlarda da tümevarım kullanılabilir.

Bu durumda ispat için aslında ispatın istendiği  $n$  sayının tamamının ispatlanması ve doğru olduğunun gösterilmesi gerekir. Diğer bir anlamda aşağıdaki şekilde  $n$ 'e kadar olan bütün terimler için ispat gerekir:

$$P(1) \wedge P(2) \wedge \dots \wedge P(n)$$

Bu terimlerin hepsinin doğru olmasının yanında şayet  $P(n+1)$ 'inde doğru olduğunu gösterebilirsek bütün sayılar için doğru olduğunu ispatlamış oluruz.

Bu ikinci teorem kapsamında kullanacağımız yöntem öncelikle  $n$  sayı için doğruluğunun ispatıdır. Ve bu  $n$  sayı için ispat sırasında  $k$  gibi  $1 < k < n$  sayısını alıp bu kısmı ispatlamaya çalışıyoruz.

#### **Örnek**

Örneğin bütün tam sayıların asal sayıların çarpımı şeklinde yazılabildiğini ispatlayalım. Yani  $n$  sayısı için  $n = a_1 a_2 \dots a_k$  şeklinde  $k$  tane asal sayının çarpımı olduğunu kabul edersek bu eşitliğin doğruluğunu ispatlamaya çalışalım.

Başlangıç adımı (basis step) olarak en küçük asal sayı olan 2'yi alırsak, 2 sayısının yine kendisine eşit tek çarpanı vardır ve kaziyemizi (önermemizi) bozmaz.

İstikra adımı (inductive step) için  $P(n+1)$  durumunun doğruluğunu ispatlamamız gerekir. Burada iki ihtimal var denilebilir:

$n+1$  asal bir sayıdır. Bu durumda asal sayıların çarpımı olarak gösterilebilir tezimizi doğrular çünkü tek çarpanı vardır o da kendisidir.

$n+1$  asal bir sayı değildir. Bu durumda  $n+1$  iki çarpan şeklinde yazılabilir (hatta bu çarpanlar en fazla eşit olabileceği (ve  $n+1$  terimi bir kare terim (bir sayının karesi) olabileceği için)  $2 \leq a \leq b < n+1$  şeklinde yazmak da doğrudur)

şayet  $n+1$  asal değilse ve



$$n+1 = ab$$

şeklinde iki sayının çarpımı olarak yazılabiliyorsa bu durumda

a veya b için de yukarıdaki şartlar doğrudur. Yani a veya b de birer asal sayıdır veya iki farklı sayının çarpımı olarak yazılabilir. Bu durum bütün sayılar çarpımlarına ayrılana kadar doğrudur.

### **SORU-15: Bağlam Yönelimli Programlama (Aspect Oriented Programming) hakkında bilgi veriniz.**

Yazılım mühendisliğinde kullanılan bir programlama yaklaşımıdır. AOP olarak kısaltılmış halde de geçer. Türkçe literatüründe bağlam / cephe / kesit / görünüm yönelimli programlama kelimelerinin hepsi farklı kaynaklarda kullanılmıştır. 1990'lı yılların ortalarında özellikle JAVA ve nesne yönelimli programlama ihe ivme kazanmış bir yaklaşımdır.

Kısaca aop'yi tanımlamak gerekirse bir program geliştirilmesi sırasında iki farklı yaklaşım görülmektedir. Birinci tip yaklaşım fonksiyonel ve programın yapacağı işlere ait yaklaşımdır. Yani programı yazdığımızda ve program çalıştığında karşılayacağı ihtiyaçlar, yapacağı işlemler şeklindeki ve iş mantığı (business logic) olarak isimlendirilebilecek bakıştır. İkinci bakış ise teknik bakıştır (technicak concerns) yani programın çalıştığı ortam, işletim sistemi, veri tabanı gibi unsurlardır.

AOP kısaca bu iki unsurlar kümesinin birbirinden ayrılmasını ve bağlam adı altında modülerleştirilmesini söyler. Yani fonksiyonel unsurlar ile teknik unsurların birbirinden ayrılmasını daha sonra modüller kapsamında bu iki unsurun birleştirilmesini söyler.

### **AOP, OOP'nin yerini alabilir mi?**

AOP yaklaşımı nesne yönelimli programlama dilleri (Object Oriented Programming, OOP) üzerinde geliştirilmektedir. Bu anlamda AOP ile OOP birbirini tam olarak ikame eden yaklaşımlar değildir. Ancak AOP yaklaşımı doğası gereği ortam olarak OOP destekleyen bir dile ihtiyaç duyar. Yani yaklaşım olarak bağlam yönelimli programlama (AOP) kullanılıyor olsa bile hâlâ sınıflar (class) yazmaya ve bu sınıfların metotlarını kodlamaya ihtiyaç vardır. AOP, bu sınıfların metotların veya özelliklerin sadece hangi bakış açısıyla modelleneceğini belirler. Yani bir proje tasarımı yapılırken sınıfların ve nesne yönelimli kodlama yaklaşımının uygulanması sırasında bakış yönelimli tercihler yapılabilir veya yapılmayabilir.

### **AOP'ye ne zaman ihtiyaç vardır?**

Bilindiği üzere normalde bir program geliştirilme sürecinde OOP kullanılması pek çok açıdan avantajlıdır ve OOP ile pek çok farklı alanda ve farklı içerikte proje geliştirilmesi mümkündür. Ancak AOP'nin farkını anlamak ve OOP'nin limitlerini görmek açısından iki noktada OOP'nin yetersiz olduğunu ve AOP'ye ihtiyaç duyduğumuzu söyleyebiliriz.

- Kod yayılması (Code scattering)
- Kestirme Fonksiyonellikler (Crosscutting Functionalities)

Yukarıdaki bu iki durumda klasik OOP yaklaşımının AOP yaklaşımına göre geri olduğu söylenebilir.



## Kod yayılımı (Code Scattering)

Bilindiği üzere nesne yönelimli programlama dillerinde nesneler arası ilişkiyi iki türlü ele alabiliriz.

- Çağırılan nesneler (invoke)
- Çağırılan nesneler (invoked)

Yani bir nesne ile diğer bir nesne arasında bir protokol bulunuyorsa, bir nesne diğerinin bir fonksiyonunu (method) çağırıyor demektir. Bu durumda nesne yönelimli programlama yaklaşımı bize bu çağırılan nesne veya methodu hakkında kaygılanmamızı ve kapalı bir kutu gibi görüp (blackbox) beklediğimiz hizmeti almamızı söyler.

Ancak herhangi bir sebeple bu aradaki çağırma işlemini yapan ve protokolümüzün üzerine kurulu olduğu fonksiyonu değiştirecek olursak. Örneğin yeni bir parametre ekleyecek olursak. Bu durumda bütün çağırılan nesnelerin de çağırma sırasındaki fonksiyonlarının değişmesi gerekir.

Yani bir metod'un parametre sayısının değişmesi gibi bir durumda bütün bu metodu çağırılan sınıflara müdahale edilmesi aslında geliştirme ve sonrada kodun yönetilmesi açısından güçlükler doğurur ve bu duruma kod yayılması (code scattering) ismi verilir

## Kestirme Fonksiyonellikler (Crosscutting Functionalities)

Kestirme fonksiyonellik kısaca nesne yönelimli yaklaşımlarda bir nesnenin modellenmesi sırasında nesneye yüklenen fonksiyonelliğin birden fazla nesne tarafından paylaşılması durumudur. Yani A nesnesinin F fonksiyonunu yüklenmesini istiyorsak ancak bu F fonksiyonunu B,C gibi diğer nesneler de yükleniyorsa veya kontrol ediyorsa bu durumda aslında A nesnesinin tam modellendiği söylenemez. Her ne kadar OOP açısından tasarlamak ve kodlamak mümkün olsa da AOP bu modele izin vermez.

Konuyu net bir örnek üzerinden anlamaya çalışalım. Klasik müşteri – sipariş örneğini düşünelim. Örneğin müşterilerimizi ve siparişleri tutan iki ayrı sınıf modellediğimizi kabul edelim. Veri bütünlüğü (data integrity) problemi olarak bilindiği üzere bir müşterinin silinmesi ancak hiç siparişi olmayınca mümkündür. Yani şayet bir müşterinin bekleyen bir siparişi varsa ve biz müşteriyi silersek, siparişi gerçekleştiremeyiz çünkü müşterinin iletişim bilgilerini kaybederiz.

Bu durumda aslında müşteri silme fonksiyonu, müşteri sınıfının bir üyesiyken sipariş sınıfının kontrol etmesi gereken bir durum oluşur. Yani müşterinin silme fonksiyonu, sipariş olmadığında çalışabilmektedir.

Bu durum üç sonuç doğurmaktadır:

1. Müşteri sınıfı siparişin olup olmadığını kontrol işlemini gerçekleştirememektedir ve tasarım itibarıyla kendi fonksiyonunu kendi başına yapamaz. Bu bir tasarım zaafıdır.
2. Müşteri sınıfı benzer şekillerde diğer veri bütünlüğü (data integrity) kurallarını da bilemek zoruna kalır. Bu da programlama açısından güçlükler doğurur.

3. Müşteri sınıfının yeniden kullanılabilirliği (code reusability) kaybedilir çünkü artık müşteri sınıfı tek başına vâir olamaz. Varlığı sipariş sınıfına bağılıdır. Yani aynı sınıf başka bir projede kullanıldığında sipariş sınıfının da kullanılması gerekir.

Yukarıdaki problemlere çözüm olarak bu veri bütünlüğü kontrolünün sipariş sınıfına konması da birşey değıştirmez çünkü bu durumda aynı problemler sipariş sınıfında tezahür eder.

AOP'nin sağladıkları

AOP yazılım geliştirme yaklaşımlarında son nesil olarak kabul edilebilir. Daha önceki fonksiyonel programlama yaklaşımlarında problem fonksiyonel elemanlara bölünerek her eleman için bir fonksiyon yazılması yoluna gidilmekteydi. Nesne yönelimli yaklaşım biraz daha ileri giderek problemi veri anlamında bütünlük arz eden nesnelerle modelledi. Son olarak bağlam yönelimli programlama (aop) ise bir adım daha ileri giderek noktakesim (pointcut) , birleşme noktası (joinpoint) ve tavsiye (advise) eklentilerinde bulunuyor. Bu terimlerin detaylarını üzerlerine tıklayarak okuyabilirsiniz.

### **SORU-16: Gellish (Kontrollü Doğal Dil) hakkında bilgi veriniz.**

Gellish dilleri sınırları ve kuralları insanlar tarafından belirlenen ve istisnası bulunmayan dillerdir. Bu anlamda programlama dilleri de dahil olmak üzere çeşitli kullanım alanları vardır. Etimolojik olarak Genel Mühendislik Dili (Generic Engineering Language) kelimelerinin baş harflerinden oluşan kelime günümüzde mühendislik uygulamalarından farklı alanlarda da kullanılmaktadır.

Genellikle karmaşıya yer verilmek istenmeyen açık ve net olunması gereken yerlerde kullanılan gellish dilini örneğin sağlık, acil durumlar, afet yönetimi gibi direktif belirtici yazılarda veya makine ve yazılımların dökümantasyonlarında görebiliriz. Buradaki amaç dil bilgisi düşük olabilecek daha az eğitimli kişilerin kolayca anlamasını sağlamak veya acil durumlarda panik halinde olabilecek kişilerin anlama güçlüklerinin ve hatalarının en aza indirgenmesidir.

Gellish dillerini oluşturan iki grup vardır:

- Kelimeler
- Kelimeler arası ilişkiler

Öncelikle kesin ve net kelimelerin, belirsizliğe yer bırakmadan anlamlarının çıkarılması gerekir. Doğal dillerde bir kelime birden fazla anlama gelebilmekte hatta bazı durumlarda mecaz veya ima ile sözlükte bile hiç yeri olmayan anlamlarda kullanılabilir. Gellish dilinin birinci adımı bu belirsilikleri ortadan kaldırmak ve her kelimeye tek bir anlam ve her anlama tek bir kelime bulmaktır.

Bu kelimeleri içeren sözlüğe akıllı sözlük (Smart dictionary) ismi verilmektedir. Buradaki kelimeler anlamsal olarak ilişkilendirilmiştir. Hatta kelimelerin anlam ilişkileri sonucunda bir ontoloji (ontology) çıkmaktadır.

Gellish dilleri doğal dillerin bir alt kümesi olarak düşünülebilir. Örneğin Gellish Türkçe (Gellish Turkish) denilince Türkçede bulunan kelimeler ve dilbilgisi kuralları kapsamında kavramları ve bu kavramlar arası ilişkileri belirleyen bir dil kast edilmektedir. Bu dil zaten

Türkçe için olan şeylerden oluşacaktır. Bu anlamda Esperanto gibi yeniden türetilmiş diller ile karıştırılmamalıdır. Ancak owl (web ontology language) gibi anlamsal ağ (Semantic web) gösterim dilleri ile karşılaştırılabilir.

### **SORU-17: Karar Problemi (Decision Problem) hakkında bilgi veriniz.**

Bilgisayar bilimlerinin de içinde bulunduğu pek çok bilim ve mühendislik dalını yakından ilgilendiren hesaplanabilirlik teorisi (computability theory) konusundaki problemlerden birisidir.

Problemi basitçe tanımlama gerekirse bir koşulun (ki biz buna karar ismini vereceğiz) sağlanıp sağlanmadığını evet-hayır şeklinde ikili olarak (duality) sorgulamaktır.

Örneğin  $x$  gibi bir sayının ikiye tam bölünüp bölünememesi bir karar problemidir.

Bir karar probleminin sonucunun bulunup bulunamayacağı belirliyse bu tip problemlere kararı mümkün problemler (kararlaştırılabilir problemler, decidable problems) denilmektedir. Örneğin bir sayının ikiye bölümünden kalanın 0 olup olmaması bu tipten bir problemidir. Ancak bazı problemler bu gruba girmez. Örneğin  $\pi$  sayısının noktadan sonraki bütün hanelerinin bulunması gibi. Bu problemin çözülüp çözilemeyeceği belirsizdir. Dolayısıyla problem kararı mümkün problem sınıfından değildir.

Bilgisayar bilimleri mantığı ile konuya yaklaşacak olursak bir problemin algoritmik olarak gösterilmesi veya gösterilememesi karar probleminin mümkün olup olmasını belirlemektedir.

Örneğin durma problemi (halting problem) bu anlamda önemli kararı mümkün olmayan problemlerdendir (undecidable problems).

Bu anlamda kararı mümkün problemleri üç grupta incelemek mümkündür:

kararı mümkün problemler (decidable problems): Şayet algoritma bir özyineli küme (recursive set) ise çözümü mümkün problemidir

Kararı kısmen mümkün problemler (semidecidable, partially decidable, provable, solvable) : Şayet algoritma özyineli sayılabilir küme ise (recursively enumerable set) bu durumda algoritmaya çözümü kısmen mümkün problem ismi verilir

Kararı mümkün olmayan problemler (undecidable problems): kararı kısmen mümkün problemler de dahil olmak üzere şayet bir algoritma özyineli bir küme (recursive set) üretiliyorsa bu gruba dahil edilir.

### **SORU-18: Özyineli sayılabilir küme (Recursively Enumerable Sets) hakkında bilgi veriniz.**

Hesaplanabilirlik teorisine (Computability Theory) bir sayı kümesi elemanlarının tamamının bir algoritma için çalışıp son bulma şartını sağlıyorsa özyineli sayılabilir küme olarak sınıflandırılır.

Daha basit bir anlatımla kümede bulunan bütün elemanlar bir algoritma için, o algoritmanın bitmesini sağlayacak elemanlar olmalıdır.

Daha akademik bir tanımla bir özyineli hesaplanabilir fonksiyon (Recursively Computable Function) için etki alanı olarak  $S: s_1, s_2, s_3, \dots$  kümesi tanımlanıyorsa bu kümedeki bütün elemanlar  $s_1, s_2, s_3, \dots$  için fonksiyona girdi (argument, parameter) olma imkanı bulunmalıdır.

Tanım itibariyle özyineli kümenin (recursive set) alt kümesi kabul edilen bu kümenin farkı kümenin elemanı olmayan durumlarda sonucun kestirilememesidir.

Yani özyineli kümenin tanımının hatırlayacak olursak:

$$f(s_i) = 1 \quad s_i \in S$$

$$f(s_i) = 0 \quad s_i \notin S$$

şeklinde gösterilen ve  $S$  kümesinin her elemanı olan  $s_i$  için 1 veya 0 şeklinde bir sonuç çıkaran kümeydi. Özyineli sayılabilir küme ise

$$f(s_i) = 0 \quad s_i \in S$$

$$f(s_i) = \text{belirsiz} / \text{hesaplanamaz} \quad s_i \notin S$$

şeklinde gösterilen bir fonksiyon olmalıdır.

### **SORU-19: Hesaplanabilir Fonksiyon (Computable Function) hakkında bilgi veriniz.**

Hesaplanabilirlik teorisinin (Computability Theory) temel taşlarından birisi olan özel bir fonksiyon (function) tipidir. Bu fonksiyonların özelliği herhangi bir formal dilbilgisi (formal grammar) yardımıyla açıklanmayan fonksiyonlar olmalarıdır.

Genellikle karıştırıldıkları için karmaşıklık teorisi (complexity theory) ile hesaplanabilirlik teorisi (computability theory) arasındaki farkı bu fonksiyonlar üzerinde de vurgulamakta yarar vardır. Basitçe bir fonksiyonun hesaplanabilir olması o fonksiyonun bir sonucunun çıkması anlamına gelir.

Örneğin Church-Turing testine göre bir fonksiyonun hesaplanabilir olması bu fonksiyonun bir makine ile (veya elektronik devre ile veya bilgisayar programı ile) modellenilebiliyor olmasını ve sınırsız zaman ve depolama imkanı (storage) verildiğinde bir şekilde biteceği anlamına gelir.

Bu bağlamda bir fonksiyonun hesaplanabilir olması en verimli şekilde hesaplanması (efficiency) veya en kısa zamanda veya en kısa depolama ihtiyacıyla çalışıp bitmesi anlamından farklıdır. Bu tip kaygılar daha çok karmaşıklık teorisi (complexity theory) konusunun çalışma alanına girmektedir.

Hesaplanabilir fonksiyonları basitçe bir parametre havuzundaki elemanlar için sonuç döndüren fonksiyonlar olarak düşünebiliriz. Örneğin  $f: A_1 \times A_2 \times \dots \times A_n \rightarrow B$  şeklinde tanımlı bir fonksiyon alalım. Bu durumda  $f(a_1, a_2, \dots, a_n) = b$  olarak gösterilebilir ve  $a_i \in A_i, i = 1, \dots, n$  ve  $b \in B$ 'dir denilebilir. Klasik bir fonksiyonu bu şekilde tanımladıktan sonra hesaplanabilir fonksiyonu bu fonksiyonda sonuç bulan anlamında  $\downarrow$  sembolü ile göstererek

$f(a_1, a_2, \dots, a_n)\downarrow = b$  ile gösterilen fonksiyonun  $a_1, a_2, \dots, a_n$  argümanları için  $b$  sonucunu veren fonksiyon olduğunu söyleyebiliriz.

Örneğin bir dilin hesaplanabilir olması bu dildeki bütün kelimeler için doğru ve bu dilden olmayan bütün kelimeler için de yanlış sonucunu veren bir fonksiyon bulunabilmesine bağlıdır. Bu fonksiyona da hesaplanabilir fonksiyon ismi verilir.

Örneğin bir düzenli ifade (regular expression) ile gösterilen dildeki bütün üretilebilen kelimeler için doğru sonucu veren fonksiyon ve üretilemeyen bütün kelimeler için yanlış sonucu veren fonksiyon bu tip bir fonksiyondur.

Bu tanımı biraz daha genişleterek bir A kümesi ve bir bu küme üzerindeki bir f fonksiyonu için Turing makinesi (turing machine) üretilebiliyorsa (ya da bir kahin makinesi (oracle) ) hesaplanabilir bir kümemiz ve hesaplanabilir bir fonksiyonumuz bulunuyor demektir. Bu duruma A-hesaplanabilir (A-computable) veya f-hesaplanabilir (f-computable) isimleri de verilir.

### **SORU-20: Özyineli Küme (Recursive Set) hakkında bilgi veriniz.**

Hesaplanabilirlik teorisine (Computability Theory) göre bir doğal sayılardan oluşan bir kümedeki bütün elemanlar bir algoritmanın belirli bir zaman sonra sona ermesini sağlıyorsa bu kümeye özyineli küme ismi verilir. Şayet kümenin elemanlarından bir veya daha fazlası algoritmanın belirli bir zamanda bitmesini sağlamıyorsa bu kümeye hesaplanamaz (noncomputable) veya karar verilemez (undecidable) ismi verilir.

Özyineli küme terimi yerine hesaplanabilir küme (computable set) veya karar verilebilir küme (decidable set) terimleri de kullanılır.

Daha akademik bir tanımla şayet hesaplanabilir bir fonksiyonun aldığı parameterleri tutan bir S kümesi varsa ve bu fonksiyon kümedeki elemanlar için doğru ve kümede olmayan elemanlar için yanlış sonuç üretiyorsa (C dilinde 0 ve 1 olarak kabul edilebilir) bu kümeye özyineli küme (recursive set) ismi verilir.

$$f(s_i) = 1 \text{ } s_i \in S$$

$$f(s_i) = 0 \text{ } s_i \notin S$$

şartlarını sağlıyorsa özyineli küme ismi verilir.

Yukarıdaki tanıma göre boş küme, bütün doğal sayılar kümesi veya asal sayılar kümesi gibi kümeler özyineli küme olarak kabul edilebilirler.

### **Örnek Soru:**

S isminde bir küme aşağıdaki şekilde tanımlanıyor olsun:

$$3 \in S$$

$$x \in S \text{ ve } y \in S \text{ ise } x + y \in S$$

Yukarıdaki S kümesinin 3'e tam bölünebilen pozitif tam sayılar kümesi olduğunu gösteriniz.

### **Çözüm:**

Yukarıda tanımı yapılan kümenin 3'e tam bölünebilen pozitif tam sayı kümesi olduğunu göstermek için bu tanıma A kümesi diyelim. Yani S yukarıda tanımlanan küme ve A da olduğunu göstermemiz istenen küme olsun.

Bu durumda  $A = S$  olduğunu göstermemiz gerekiyor. Bunu göstermenin bir yolu da aynı anda

hem  $A \subseteq S$  hem de  $S \subseteq A$  olduğunu göstermemizdir. Yani şayet A, S'in alt kümesi ve aynı anda S de A'nın alt kümesiye bu iki küme eşittir denilebilir.

Önce A'nın S'in alt kümesi olduğunu ispatlayalım. Bunun için matematiksel tümevarım teoreminden (mathematical induction principle) istifade edeceğiz.

başlangıç adımımız (basis step) : 3 olarak kabul edersek

istikra adımı olarak  $P(n)$  gibi bir fonksiyon tanımlayalım.  $P(n)$  sayının 3'e bölünebilme özelliğini kullanan fonksiyon olsun dolayısıyla

$P(n) = 3n$  olsun. Görüldüğü üzere  $P(n)$  fonksiyonu her zaman 3'e tam bölünebilen sayılar üretir.

S kümesindeki herhangi bir k sayısını alalım. Bilindiği üzere k sayısı 3'e bölünebilen sayıdır (iddia gereği)

Dolayısıyla  $k = 3n$  şeklinde yazılabilir.

İstikra gereği (induction) bu serideki bir sonraki sayıyı bulmak için S kümesindeki en küçük eleman olan 3 ile sayımızı topluyoruz.

Öyleyse  $k+3$  yada  $3n+3$  sayısı serideki k'dan sonra gelen sayı olur.

Bu gösterimi A kümesi için yazacak olsaydık

$P(n) = 3n$  olacaktı ve bir sonraki sayı  $P(n+1) = 3n + 3$  olacaktı.

Dolayısıyla görülebileceği üzere

$$P(n) = k$$

$$P(n+1) = k+3$$

eşitliklerinin ikisi de , yani istikra (tümevarım, induction) adımlarımızın ikisi de sağlanmış oluyor.

Şimdi ispatın ikinci kısmına geçelim ve S'in A'nın alt kümesi olduğunu gösterelim. Bunun için S'in özyineli (recursive) tanımından faydalanabiliriz.

Öncelikle başlangıç adımı olan 3, S kümesinin elemanıdır. Ayrıca  $3 = 3 \times 1$  şeklinde yazılabilir. Buradan S kümesindeki bütün elemanların  $3x$  şeklinde yazılabileceğini söyleyebiliriz. Bunu göstermek için S kümesinin tanımındaki ikinci parçayı yani  $x+y$ 'nin A'nın bir üyesi olduğunu göstermeliyiz.

Şayet  $3|x$  ve  $3|y$  ise (yani  $x$  ve  $y$  ayrı ayrı 3'e tam olarak bölünebiliyorsa) o zaman  $3|x+y$ 'dir denilebilir ( $x+y$ , 3'e tam bölünebilir)

Yukarıdaki bu bölünebilirlik iddiasını ispatlamak için

$3|x$  ve  $3|y$  durumunda

$x = 3s$  ve  $y = 3t$  olarak yazılabilir. Buradaki  $s$  ve  $t$  değerleri 3 sayısının herhangi bir çarpanıdır.

Dolayısıyla  $x + y = 3s + 3t$  olarak yazılabilir ve buradan  $3(s+t)$  sonucuna ulaşılır ki  $s+t$  değeri her ne olursa olsun  $3(s+t)$  değeri 3'e tam bölünebilir.

Yukarıdaki iki ispat sonucunda  $A=S$  olduğu söylenebilir ve özyineli kümenin bu özelliğinden faydalanarak ispat yapılmıştır.

### **SORU-21: Anlamsal Bağ (Semantic Link) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde yapay zeka konusunda özellikle de doğal dil işleme ile ilgili yapılan çalışmaların önemli bir kısmını anlambilim (sematic) kaplar. Kısaca bir metin veya ortamdan elde edilen bilginin anlamını çıkarmak ve bu anlamı kullanışlı hale getirmek anlambilimin (semantics) çalışma alanına girmektedir.

Anlambilimsel bağlar ise bu çıkarımı ve gösterimi yapılan bilgilerin birbirine nasıl bağlandığını tutmaktadır. Bir anlamda iki etki alanı (domain) arasındaki geçişi gösteren fonksiyonlar (functions) olarak kabul edilebilirler.

Bu bağların kendileri de ayrıca anlambilimsel çalışmanın bir parçası olmaktadır. Örneğin aşağıdaki cümleyi ele alalım:

“Ali, Ayşe ile Ahmetin oğludur”

Yukarıdaki bu cümlede üç kişiden bahsedilmektedir: Ali, Ahmet, Ayşe

Şayet Ahmet'in erkek ismi ve Ayşe'nin de kadın ismi olduğu biliniyorsa yukarıdaki cümleden çıkarılabilecek anlamlardan bazıları aşağıdadır:

- Ali'nin annesi Ayşedir
- Ali'nin babası Ahmettir
- Alinin yaşı Ahmetin yaşından küçüktür.
- Ayşenin yaşı Alinin yaşından büyüktür.

Ayrıca yukarıdaki cümlelere ilave olarak istatistiksel dilbilim (probabilistic linguistic) ile olaya yaklaşırsak:

- Ahmetin karısının ismi Ayşedir.
- Ayşenin kocasının ismi Ahmettir.

Sonuçlarına varılabilir. Buradaki gizli kabul (hidden premises) Ahmet ile Ayşenin evli olduklarıdır.

Yukarıdaki bu çıkarımların tamamı anlambilimsel bağ (semantic link) örnekleridir. Yani anlamını çıkardığımız üç varlık (Ali,Ahmet ve Ayşe) arasındaki bağları göstermektedirler.

Anlambilimsel bağlar, Anlambilim ağlarının (Semantic web ve Semantic Network olarak geçmektedir) temelini oluşturmaktadırlar.

Benzer çıkarımlar daha makine diline yakın olan formal diller için (formal languages) yapılabilir.

Örneğin XML dökümanlarında benzer çıkarımlar yapılabilir. Bir kütüphanenin XML dosyasını düşünelim. Muhtemelen kitap bilgisi tutulurken yazar, yayın evi, yayın yılı gibi ilave bilgilerle tutulacaktır.

```
<kitap yazar="Şadi Evren ŞEKER" isim="Programlama ve Veri Yapılarına Giriş" />
```

şeklinde bir satırdan oluşan XML girdisini yorumlarsak.

- “Programlama ve Veri Yapılarına Giriş” isimli bir kitap vardır
- “Programlama ve Veri Yapılarına Giriş” isimli kitabı “Şadi Evren ŞEKER” yazmıştır.
- “Şadi Evren ŞEKER” bir kitap yazmıştır.

benzeri sonuçlarına varılabilir.

Benzer durumlar farklı ortamlarda yapılan analiz çalışmaları ile çıkarılabilir. Buradaki amaç ulaşılan anlambilimsel neticedir.

## **SORU-22: Küme Teorisi (Set Theory) hakkında bilgi veriniz.**

Bilgisayar bilimleri de dahil olmak üzere pekçok bilim ve mühendisliğin kullandığı kümeler teorisinde göre bir küme basitçe boş veya belirli sayıda elemanı bulunan grubun ismidir. Buna göre bir kümenin elemanları bulunabilir ve ayrıca kümelere kolaylık olması için isimler verilebilir.

Küme teorisine göre bir eleman bir kümede bir kere bulunabilir yani aynı elemandan iki tane bulunan küme olamaz. Ayrıca küme teorisinde elemanların sırasının önemi yoktur.

Boş küme hiç elemanı olmayan küme demektir ve  $\emptyset$  sembolü ile gösterilir.

Evrensel küme bütün kümeleri kapsayan ve bir kümenin sahip olabileceği en büyük sınırdır (buradan anlaşılacağı üzere sınırsız küme yoktur) ve U harfi ile gösterilir (ingilizcedeki Universal kelimesinin baş harfi ile)

### **Kümeler üzerindeki işlemler**

Kümeler üzerinde küme teorisi kapsamında çeşitli işlemler yapılabilir. Bu işlemler aşağıda sıralanmıştır ve her durum için birer örnek verilmiştir. Örnekler için aşağıdaki kümeleri esas alınız:

$$A = \{ 1,2,a,b \}$$



$$B = \{ 2, a, c, 4 \}$$

Kesişme işlemi (intersection) iki kümenin ortak elemanlarını kapsayan kümesini elde edilmesidir ve  $\cap$  sembolü ile gösterilir. Örneğin  $A \cap B$  işlemi sonucunda A ve B’de bulunan ortak elemanlar alınır.

$$A \cap B = \{ 2, a \}$$

Birleşme işlemi (Union) iki kümenin bütün elemanlarını içeren bir kümenin elde edilmesidir ve  $\cup$  sembolü ile gösterilir. Örneğin  $A \cup B$  işlemi sonucunda A ve B’de bulunan bütün elemanlar alınır. Küme teorisine göre bir kümede bir elemandan bir tane bulunabileceği için ortak elemanlar birer kere tekrarlanır.

$$A \cup B = \{ 1, 2, 4, a, b, c \}$$

Tümleyen işlemi (Complement) bir kümenin kendisi dışında kalan kısmıdır. Yani evrensel kümeden bir küme çıkarıldığında elde edilen kümeye o kümenin tümleyeni denir. Kümenin üzerine çizgi konularak gösterilir. A) gibi.

### **Küme teorisindeki kurallar**

Vahdet kuralı (Hüveiyet Kaidesi, Birlik kuralı, Identity law): Basitçe bir kümenin varlığını ve birliğini belirleyen kuraldır. Buna göre bir kümenin var olması yok olmaması ve kainatta bir yer kaplaması ile mümkündür. Dolayısıyla boş küme ile birleştiğinde değişmeden kendisi geri çıkıyorsa (yok değilse) ve evrensel küme ile de kesiştiğinde değişmeden kendisi geri çıkıyorsa (kainatta yer kaplıyorsa) bu kümenin vahdetinden bahsedilebilir.

$$A \cap U = A$$

$$A \cup \emptyset = A$$

şartlarını sağlamalıdır.

Kibriyet kuralı (Tahakküm Kaidesi, Domination law): Bu kurala göre evrensel küme bütün kümeler üzerine birleşme yönünden (union) mütebairdir. Yani bir küme evrensel küme ile birleşirse varlığını yitirir ve birleşme işlemi sonucunda evrensel küme çıkar. Benzer şekilde boş küme de kesişim yönünden mütebairdir ve bir kümenin boş küme ile kesişimi yine boş küme verir.

$$A \cup U = U$$

$$A \cap \emptyset = \emptyset \quad (\text{Mahrumiyet Kaidesi, Exclusion law})$$

Tekrar kuralı (Denk Güçlülük Kaidesi, Idempotent law): Bu kural bilgisayar bilimlerinde ve matematikte kullanılan bir işlemin (operator) tekrarlanması durumunda sonucun değişmediğini anlatan kuraldır. Basitçe kesişim ve birleşim işlemlerinin bir küme üzerinde tekrarlanması durumunda kümenin kendisi geri elde edilir.

$A \cup A = A$  olur ayrıca  $A \cap A \cup A \cap A \cup A \cap A \cup A \cap A = A$  şeklinde istenildiği kadar işlem tekrarlanabilir sonuç değişmez.

$A \cap A = A$  için de sonuç aynıdır.  $A \cap A \cap A \cap A \cap A \cap A \cap A \cap A = A$  şeklinde işlem istenildiği kadar tekrar edilebilir sonuç değişmez.

Tamamlama kuralı (Mütemmim Kaidesi, Complement Law, Tümleme kuralı): Bir kümenin kendisi dışında geri kalanlardan geri kalan yine kendisidir. Daha basitçe bir kümenin tümleyeni alındığında evrensel kümeden geri kalanlar alınmış olur. Bu geri kalanların tümleyeni alındığında da orjinal küme geri elde edilir.

$$\overline{\overline{A}} = A$$

Yer değiştirme özelliği (Commutative Law): Kesişme ve birleşme işlemlerinin yer değiştirme özelliği bulunur.

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

Birleşme Özelliği (İttihâd Özelliği, Associative Law): Aynı nitelikteki işlemlerin önceliklerinin değişmesi durumudur. İşlem önceliği (operator precedence) aşağıdaki şekilde parantezlerle belirlenmiştir. Buna göre parantez yapısı değişince sonuç değişmez. Benzer durum işlemin soldan sağa veya sağdan sola çalışması durumunda da değişmez.

$$A \cup (B \cup C) = (A \cup B) \cup C$$

$$A \cap (B \cap C) = (A \cap B) \cap C$$

Dağılma özelliği (Distributive law): Bir işlemin diğer bir işlem üzerine dağılması özelliğidir. Örneğin kesişimin birleşim veya birleşimin kesişim üzerine dağılma özelliği bulunur.

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

### **Alt küme ve üst küme kavramları**

Bir kümenin bütün elemanlarının diğer bir kümenin elemanlarına birebir eşit olması durumunda bu iki küme için eşit yorumu yapılabilir.

Eşitlik durumunda şayet kümelerden birisi örneğin A kümesi, diğer kümeden örneğin B kümesi ilave olarak ve eşitliği bozarak fazla elemana sahipse bu durumda A kümesi B kümesinin üst kümesidir (Super set) ve B kümesi de A kümesinin alt kümesidir (subset).

Diğer bir ifadeyle A kümesi B kümesinin bütün elemanlarını içeriyor ve ilave olarak elemanlar da içeriyorsa bu durumda A kümesi B kümesinin üst kümesi olur ve A kümesi B kümesini kapsar denilebilir.  $B \subset A$  veya  $A \supset B$  sembolleri ile gösterilebilir.

Şayet  $A \supset B$  durumu söz konusuysa A'nın eleman sayısı, B'nin eleman sayısından fazladır yorumu yapılabilir.

Ayrıca öz alt küme (proper sub set) olarak isimlendirebileceğimiz bir alt küme tanımı, kümenin kendisine eşit olmayan alt kümelerdir. Yani normalde bir kümenin alt kümeleri arasında kendisi de sayılabilir, şayet kümenin kendisi dışındaki alt kümeleri kast ediliyorsa bu kümelere öz alt küme ismi verilir.

### **SORU-23: Güç Kümesi (Kuvvet Kümesi, Power Set) hakkında bilgi veriniz.**

Ayrık matematikte (Discrete Math) kullanılan bir terimdir. Basitçe bir kümenin boş küme de dahil olma üzere bütün altkümelerini içere kümedir.

$A^2$  olarak da gösterilir , A kümesinin kuvvet kümesi olarak okunur (bazı kaynaklarda güç kümesi olarak da geçer)

$A = \{a, b\}$  ise  $2^{\{a, b\}} = \{\{a, b\}, \{a\}, \{b\}, \emptyset\}$  olarak gösterilir.

Yukarıdaki tanımdan da çıkarılabileceği üzere n elemanlı bir kümenin güç kümesinde  $2^n$  eleman bulunur.

### **Örnek**

Boş kümenin, güç kümesi nedir. Bu soruyu cevaplamak için boş kümeyi nasıl kabul edeceğimize bakmamız gerekir. Şayet boş küme bir eleman olarak kabul ediliyorsa (boşlukta bir varlıktır diye başlayan varlık felsefesine atfen) bu durumda

$A = \{\emptyset\}$  için güç kümesi  $= \{\emptyset, \{\emptyset\}\}$  olarak bulunacaktır. (1 elemanlı kümenin 2 elemanlı güç kümesi inşa edilmiştir  $2^1 = 2$  )

Şayet boş küme boşluk olarak kabul edilirse bu durumda  $A = \{\emptyset\} = \{\emptyset\}$  olarak tek elemanlı sonuç bulunur. (0 elemanlı kümenin 1 elemanlı güç kümesi inşa edilmiştir  $2^0 = 1$  )

Ayrık matematik açısından ilk tanım doğru kabul edilmektedir ancak literatürde ikinci kabule de rastlanmaktadır. Dolayısıyla bu yazıda taraf tutulmaksızın iki durum da verilmiştir.

### **Burali-Forti Paradox ( Burali-Forti Açmazı)**

Tüm kümelerin kümesine **T** diyelim. **T**'nin tüm alt kümelerinden oluşan kümeye (*power set*) de **A** diyelim.  $|A|$  ile A kümesinin eleman sayısını ve  $|T|$  ile de T kümesinin eleman sayısını ifade edecek olursak.

Biliyoruz ki:  $|A| > |T|$  olmalıdır, çünkü  $2^m > m$

Fakat **T** tanım gereği tüm kümeleri kapsıyor, dolayısıyla  $|A| > |T|$  koşulu nedeniyle, **A** kümesi tüm kümelerin kümesinden de büyük bir küme olmalıdır.

Yukarıdaki paradoksun çözümü modern küme teorisinde bütün kümeleri ifade eden bir küme tanımlanmasına izin verilmeyerek çözülmüştür.

## **SORU-24: Malumat Çıkarımı (Knowledge Retrieval) hakkında bilgi veriniz.**

Malumat çıkarımı aslında bilişsel bilimin (cognitive science) çalışma alanlarından olmakla beraber insanlığın çok eskiden beri kendi adına yaptığı bir eylemin ismidir. Basitçe insanoğlunun gözlemleyerek, okuyarak, dinleyerek, düşünerek veya benzeri eylemlerle dış veya iç dünyasından bir malumat elde etme işlemidir.

Bilgisayar bilimleri açısından bu eylemin önemi, bu eylemin bilgisayarlar tarafından yapılmasının hedeflenmesidir. Örneğin makine öğrenmesi (machine learning) veya soru cevaplama (question answering) gibi yapay zeka çalışmalarının temelinde malumat çıkarımı (knowledge retrieval) bulunmaktadır.

Mesela bir metinden sorulan sorulara cevap veren bir bilgisayar programı olması tasarlanıyor. Programın bu durumda metni işleyerek bir malumat çıkarması ve bu malumata ilişkin soruları doğru cevaplaması beklenir.

Malumat çıkarımının diğer konularla karışmaması açısından daha alt seviyedeki çıkarımlara (retrieval) örnekler verelim:

Örneğin veri çıkarımı (data retrieval) işlemi için veri tabanları (database management systems) birer örnek olabilir.

Benzer şekilde bilgi çıkarımı (information retrieval) işlemi için de internet arama motorları (web search engines) örnek olabilir.

Ancak her iki seviyeli çıkarım işleminde de kaynakların anlatmak istedikleri anlama erişmek için çok sayıda işlem yapmak gerekir. Malumat çıkarımı işlemi bu adımı atlayarak istenen öze daha kolay ulaşmayı ve doğru şekilde gösterilmesini ( malumat gösterimi (knowledge representaion) ) hedeflemektedir.

Aşağıda “Yiyu Yao, Yi Zeng, Ning Zhong, Xiangji Huang” tarafından yazılan Knowledge Retrieval (KR) başlıklı yazıdan alınma karşılaştırmalı bir tablo görülmektedir.

	<b>Data Retrieval</b> <b>(Veri Çıkarımı)</b>	<b>Information Retrieval</b> <b>(Bilgi Çıkarımı)</b>	<b>Knowledge Retrieval</b> <b>(Malumat Çıkarımı)</b>
Match (eşleme)	Boolean match (Boole Eşlemesi)	partial match, best match (kısmi eşleme, tam eşleme)	partial match, best match (kısmi eşleme, tam eşleme)
Inference (İstidlal)	deductive inference (Tümdengelim)	inductive inference (Tümevarım)	deductive inference, inductive inference, associative reasoning, analogical reasoning
Model	deterministic model	statistical and probabilistic model	semantic model, inference model (Anlambilimsel model, istidlalen model)

	(belirli model)	(istatistiksel model)	
Query (Sorgulama)	artificial language (Yapay zeka)	natural language (Doğal dil)	knowledge structure, natural language (Malumat yapısı, doğal dil)
Organization	table, index (Tablo, indis)	table, index (Tablo, indis)	knowledge unit, knowledge structure (malumat ünitesi, malumat yapısı)
Representation (Gösterim)	number, rule (Sayılar, kurallar)	natural language, markup language (Doğal dil, işaretleme dili)	concept graph, predicate logic (Haber Mantiğı), production rule, frame, semantic network (Çerçeve anlambilimsel ağ), ontology (Varlıkbilim)
Storage	Database (veritabanı)	document collections	knowledge base (Malumat Tabanı)
Retrieved Results (Çıkarılmış Sonuçlar)	data set (veri kümesi)	sections or documents	a set of knowledge unit

Yukarıdaki tabloda farklı problem konuları için iç başlık yani Veri, Bilgi ve Malumat çıkarımları gösterilmiştir.

### **SORU-25: Malumat İfadesi (Knowledge Representation) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde oldukça önemli konulardan biriside işlenen veri, bilgi, malumat veya ifranın (data,information,knowledge, wisdom) gösterilmesi ve işlenebilmesidir.

Temel olarak bilgisayar bilimlerinin yapay zeka konusunun araştırma alanına giren malumat ifadesi konusu “nasıl düşünüyoruz?” sorusuna da cevap aramaktadır. Aslında yapay zeka konusunda yapılan çalışmaların neredeyse tamamı insanı model alan ve insanın düşünce tarzını bilgisayarlar üzerinde uygulamaya çalışan bir yol tutmaktadır. Malumat ifadesi konusu da benzer şekilde insanın elde ettiği bir malumatı nasıl ifade ettiğini veya nasıl işlediğini incelemekte ve bu inceleme sonucunu bilgisayarlar ile ifade etmeye çalışmaktadır.

Örneğin bir külliyyatın (söylev, discourse) içersinde geçen onlarca cümleden tek bir anlam çıkarmak ve çıkan bu anlamı bir model üzerinden bilgisayarda gösterebilmek ancak bu cümlelerin ve cümleler arası ilişkilerin ve cümlelerin üzerine kurulduğu geçmiş bilgilerin doğru bir şekilde modellenmesi ile olabilir.

Örneğin insanın düşünürken zihninde oluşan semboller veya belleğinde oluşsan geçmiş tecrübeleri veya kurduğu mantık önermeleri birer model oluşturmaktadır.

### **Malumat İfadesinin temel sorunları**

- Temel olarak malumat ifadesi konusunda geçen soruları ve ilgi alanlarını aşağıdaki şekilde sıralamak mümkündür:
- Bir malumatın hal-i hazırda insanlar tarafından nasıl gösterildiği
- Temel olarak malumatın ne olduğu ve nasıl gösterilebileceği
- Gösterim yönteminin belirli bir alanla sınırlı olup olmayacağı (bütün malumatları gösteren tek bir gösterim bulunabilir mi?)
- Bir gösterim yönteminin formal diller (formal languages) tarafından ne kadar işlenebilir olduğu (makine diline ne kadar yakın olacağı)

Yukarıdaki bu soruların dışında temel olarak bilinmelidir ki yapay zekanın herhangi bir alanında bir problemin çözümünde malumatın ne kadar iyi modellendiği önemli bir role sahiptir. İyi modellenmiş ve kullanım olanakları (üzerinde tanımlı işlemler (operators) ) genişletilmiş bir malumat gösterimi problem çözümünde oldukça işlevsel olabilmektedir.

Örneğin sayılar birer ve sayılar üzerinde yapılan işlemleri bir malumat olarak kabul edelim. Bu durumda onluk sistemi içeren hint-arap sayı sistemi (günümüzde kullanılan sayı sistemi) ile roman sayı sistemi (roma rakamlarının kullanıldığı sayı sistemi) karşılaştırıldığında aslında iki sayı sistemi de aynı malumatı ifade etmektedirler. Fakat bilindiği üzere roma rakamları ile yapılan işlemlerin kullanılışlığı ve işlevselliği oldukça düşüktür.

### **Bazı temel malumat ifade alanları**

Bu bölümde malumat ifadesinin kullanıldığı temel yapay zeka problem gruplarından bahsedilecektir. Aslında bu bölümde bulunan konular kabaca yapay zekanın çalışma alanlarının bir listesi niteliğindedir çünkü malumat ifadesi bütün yapay zeka çalışmalarında bir şekilde gerekmektedir.

### **Dilde malumat gösterimi**

Yapay zekanın bir alt bölümü olan doğal dil işleme ve formal diller olarak ikiye ayrılabilir bu başlıktaki amaç bir doğal dili (insanların konuştuğu ingilizce veya türkçe gibi diller) ile bir bilgisayar dili (C veya JAVA gibi) bir malumatı gösterecek şekilde tasarlamak ve geliştirmektir. Amaç bilgisayar bilimleri açısından bu dillerin bilgisayardaki ifadeleridir. Elbette doğal dildeki malumatın gösterimi ile dilbilim (linguistic) veya dil felsefesi alanları çalışmaktadır ancak yapay zekanın bakış açısında göre amaç doğal dillerin de bilgisayar dünyasındaki modellenmesi ve bu dillerdeki malumatların çözümlenerek işlenmesidir.

Kabaca bir dilin modellenmesi ve işlenmesi insan beyninin düşünme şeklinin ve ulaşabileceği sınırlarının tamamen modellenmesi ile mümkündür şeklinde bir görüş ileri sürülebilir.

### **Varlıkbilim gösterimleri**

Malumat ifadesinin kullanıldığı önemli alanlardan birisi de varlıkbilimdir. (onthology) Bu alanda elde edilen malumatlar belirli bir model üzerinde ifade edilerek çıkışma tespit edilmeye çalışılır.

Örneğin “Napolyonun motorsikletinin seri numarası nedir?” sorusuna cevap olarak “Napolyon zamanında motorsiklet yoktu” cevabının verilebilmesi için soruda modellenen ontolojinin çelişkiyi algılaması gerekir. Bu konuda örneğin owl gibi modelde daha çok kesin ve belirli (definite and deterministic) diller kullanılır.

## Yapılar ve bağlantılar

İnsanlar tarafından malumatı göstermenin diğer bir yoluda oluşturulan tabloların ve yapıların kullanılmasıdır. Örneğin çarpım tablosu oldukça eski zamandan beri insanlar tarafından oluşturulan ve bir malumatı tutmaya yarayan tablodur.

Örneğin bilgisayar bilimlerinde kullanılan karnaugh haritaları (Karnaugh map) veya doğruluk çizelgeleri (truth tables) benzer amaçlara hizmet eden ve örneğe ait malumat içeren tablolardır.

Benzer şekilde üretilen karar ağaçları gibi yapılar da bu grupta sayılabilir. İşin özünde insalık tarihi boyunca oluşturulan bütün malumat modelleme enstrümanları (çizimler, tablolar veya yapılar) bu grubun üyesi sayılabilir.

Bu grubun en belirgin özelliği belirli bir alana (domain) bağımlı üretilen modeller olmalarıdır ve bu modeller genelde bir işin parçası olarak kullanılırlar. Üzerlerinde tuttukları malumatı göstermenin muhtemelen farklı birkaç yolu daha bulunmaktadır.

### **SORU-26: Pragma (Edimbilim, kullanımbilim, Fiili, Ameli) hakkında bilgi veriniz.**

Genel olarak dildeki cümlelerin ve kelimelerin anlattıklarından daha ötede bulunan anlamı ifade eder. Örneğin bir kişiye “saatin var mı?” diye bir soru sorulursa buradaki anlam aslında saatin kaç olduğunun sorulmasıdır. Yani bu cümledeki pragmatik ifade saatin sorgulanmasıdır. Yukarıdaki bu soruya kişi “Evet var” şeklinde bir cevap verirse sorudaki pragmayı kaçırmış olur.

Doğal dil işleme çalışmalarında pragma önemli bir rol oynar. Bir bilgisayarın pragmatik yapıyı anlayamaması durumunda doğru cevap vermesi beklenemez. SPEECH ACTS

Pragma konusundaki önemli kavramlardan birisi de (îmâ, IMPLICATURE)’dur. Buna göre bir kişinin kurduğu cümleden yapılabilecek çıkarımlar yine pragma çalışmasının parçasıdır. Örneğin

“Benim iki çocuğum var”

Cümlesinden benim çocuğum olmadığı veya tek çocuğum olduğu çıkarımları hatalı olur. Ancak bu cümleyi duyan birisi benim 3 çocuğum olduğunu söylerse bu iddia yukarıdaki cümle ile çelişmez. Yani 3 çocuğu olan birisi için 2 çocuğu olduğu da doğrudur.

“Benim sadece iki çocuğum var”

Cümlesi bu anlamda daha doğrudur. Ancak pragma çalışmasına göre yukarıdaki ilk cümleden de benim 3 çocuğum olduğu sonucunu çıkarmamak gerekir.

Diğer önemli bir konu da önkabuldür (presupposition). Örneğin

“kızım çok zekidir”

Cümlesine göre bir kızım olduğu kabul edilmiştir.

Önkabul ile ilgili bir durum da bu kabulün iptalidir (cancel). Örneğin

“Bu sene bütün notlarım sınıf ortalamasına eşit”

“Çünkü sınıfta tek öğrenci var”

Yukarıdaki ilk cümlede sınıf ortalaması ile kişinin sınıfı grup ismi (mass noun) kabul edeceği ve sonucunda kalabalık (birden fazla) olacağını düşünmesi beklenir. İkinci cümle ile bu ön kabul iptal edilerek sınıfın tek kişilik olduğu söylenmiştir.

Pragma çalışmaları ayrıca Söylev ( Discourse) çalışmalarını da kapsar. Bu çalışmaların amacı birden fazla cümlenin birbirine bağlanma şeklini incelemektir.

### **SORU-27: Haber (Predicate) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde önemli bir rol oynayan dilbilimi ve dil felsefesinin önemli unsurlarından birisidir. Bir cümlenin iki önemli unsurundan birisi olarak kabul edebiliriz. Haber-müpteda ilişkisi (Subject-Predicate) veya Özne-yüklem ilişkisi de denilebilir.

İçerik

1. Haberin dilbilimsel incelemesi
2. Dilbiliminde haber sınıfları
  - a. Hal haberleri
  - b. kişisel haberler
  - c. Nevi Haberler
3. Dağıtıcı ve birleştirici tip haberler

Yani örneğin “Benim kalemim kırmızı” cümlesindeki “Benim kalemim” müptedayı (özne,subject) ifade ederken “kırmızı(dır)” kelimesi ise haberi ifade etmektedir.

Burada üretilen haber-müpteda birleşimlerinden kaziyeler mantığı inşa etmek (önergeler, prepositional logic) mümkündür. Dolayısıyla aslında yapılan ifadenin üzerine bir matematik inşası da mümkündür. Zaten böylede olmuştur ve Haber Cebiri (Predicate Calculus) ismi verilen ve özellikle yapay zeka programlamasında oldukça önemli bir yer tutan bir matematik dalı mevcuttur.

Bu yazıda öncelikle dilbilim ve doğal dil işleme açısından haber (predicate) konusu incelenecek ardından bu konu üzerine inşa edilmiş matematik (predicate calculus) incelenecektir.

### **1 Haberin Dilbilimsel İncelemesi**

Güncel anlambilimsel (semantic) çalışmalara göre bir haber (predicate) doğruluğu ifade etmek için kurulan bir cümle parçasıdır. Yani kalemim kırmızıdır cümlesindeki ifade gibi. Ayrıca bu ifadeyi güçlendirmek veya genişletmek için çeşitli sıfat ve isimlerden yararlanılabilir. Müpteda-Haber şeklinde olan cümlelere isim cümlesi ismi de verilir.

Örneklerle durumu açıklamak gerekirse:

Ali uykudadır. (isim cümlesi, haberi hal durum zarfı)

Ali kitap okuyordur. (isim cümlesi doğrudan nesneyi işaret ediyor, Şimdiki zaman )



Ali parkta. (isim cümlesi, yer yön zarfı)

## 2 Dilbiliminde haber sınıfları

Dilbiliminde haberler anlamları bakımından çeşitli sınıflarda incelenebilir. Greg N. Carison tarafından yapılan bu sınıflama aşağıda verilmiştir.

### a Hal haberleri (State level predicates, s-l predicates , h-s haber)

Bir hali, durumu bildiren haberlerdir. Ali uykulu. Ali aç. Ali kokmuş. cümlelerinde olduğu gibi bir durum belirtirler. Buradaki durum belirtisi bir zamanla kısıtlıdır. Yani Alinin uykulu olması aç olması veya korkmuş olması geçici bir haldir.

### b Kişisel haberler (Individual level predicates, I-l predicates, k-s haber)

Bir kişilik durumunu belirtirler. Örneğin Ali hızlıdır. Ali akıllıdır. Cümlelerinde olduğu gibi. Bu tip haberler hal haberlerinin genelleştirilmesi ile de elde edilebilir. Örneğin Ali korkaktır cümlesi bir önceki örnekte bulunan Ali korkmuş haberinin genelleştirilmesidir.

Bu grupta bulunan haberler kişi ile özdeşleşmiş haberlerdir ve ilgili kişiyi betimlerler.

Dilbilgisi (grammatic) olarak bu kelimeleri sınıflandırmak ve s-l veya i-l şeklinde yorumlamak ne yazık ki mümkün değildir. Kelimelerin taşıdıkları anlamlar bu grupta önemli rol oynar.

Örneğin geçmiş zamanda ifade edilen

- Ali uykuluydu
- Ali cesurdu

Cümlelerinden ilki h-s haber iken ikincisi k-s haber olmuştur.

### c Nevi haberler (Kind level predicates, k-l predicates, n-s haber)

Bu haber grubunda belirtilen haberin bir varlığın çeşidini veya şeklini betimlemesi beklenir. Bu grubun kişisel haberlerden ayrılması için bir çeşit bir cins hakkında haber bildirmesi gerekir.

Kediler uysaldır

cümlesinde olduğu şekliyle bir cinsin bir nevîni belirtmektedir.

Örneğin insanlar hırslıdır cümlesini kişisel seviye haberden ayıran özellik insanın genellenmiş bir nevî oluşundandır. Benzer cümle Ali hırslıdır şeklinde olsaydı k-s haber olacaktı.

Yukarıdaki örneklerden de anlaşılacağı üzere k-s haber ile n-s haber ifade ettikleri faile göre sınıflanmaktadır.

## 3 Birleştirici ve dağıtıcı haberler

Haberler ifade ettikleri varlığa göre birleştirici veya dağıtıcı olabilirler. Örneğin bir haberde varlıkların birleşmesi bir başkasında ise varlıkların dağılması ifade edilebilir.

Örneğin Gazete dağıtmak, Dünyaya yayılmak, Haber yaymak, Oyuncaklarını dağıtmak gibi dağıtıcı tip haberlerin yanında

Mektupları toplamak, Oyuncaklarını topladı, Bulmacayı birleştirmek gibi birleştirici tip haberler de mevcuttur.

Haberlerin dağıtıcı veya birleştirici olması haberin ifade ettiği anlama göre sınıflandırılabilir. Ayrıca birleştirici tip habelerin ifade ettikleri varlıklar çoğul iken dağıtıcı tip haberlerin ifade ettiği varlıklar tekil veya çoğul olabilir.

### **SORU-28: Şekli Mantık (Kipler Mantığı, Modal Logic) hakkında bilgi veriniz.**

Mantığın bir türü olan şekli mantığında şekiller (modal) bir kaziyenin (önerme) doğruluğunu göstermek için kullanılır. Genel olarak şekil mantığında gösterilen 3 tip şekil bulunur:

- olabilirlik (possibility)
- ihtimal (probability)
- gereklilik (necessity)

Doğal dil açısından ve dilbilim gözüyle şekli mantığa bakarsak, aslında yukarıdaki bu liste ve şekil mantığının dayanağı İngilizcede açıkça kullanılan ve kelime zamana ve bakışlarını etkileyen modallara bağlıdır. Yani İngilizcede “*eventually, formerly, can, could, might, may, must, ought, need*” gibi modalların her birisi bu yukarıda sayılan üç tipten birisine girmektedir.

Türkçe için tam bir karşılığı olmayan bu şekillerin karşılıkları kipler ve fiil zamanları ile karşılanmaktadır.

Mantık bilimi açısından konu incelendiğinde aslında dört tip işlemin (operator) üzerinde tanımlı olduğu bir mantıktan bahsedilmektedir. Bu işlem tipleri aşağıdaki şekilde sıralanabilir:

- Alethic (gerekirlik)
- Deontic (Ahlaksal)
- Axiological (belistsel)
- Epistemic (Bilgisel)

Yukarıda sıralanan bu şekli mantık tipleri aşağıda açıklanmıştır.

#### **Alethic (Gerekirlik) şekli**

Bu şekli mantık türünde bir kaziyenin (önerme) yargılanması gerekirlik açısından yapılır.

Örneğin “Çember kare olamaz” cümlesindeki olamamazlık ile “Ali mühendis olamaz” cümlesindeki olamamazlık arasında fark vardır. Birincisi doğası itibarıyla imkansız iken (olmaması gerekir iken) ikincisinin olmamasının gerekliliğinden sadece bir yorum veya kanaat olarak bahsedilebilir. Dolayısıyla ilk kaziye için alethic (gerekirlik şeklinden doğru) denilirken ikinci kaziye için non-alethic (gerekirlik şeklinden yanlış) denilebilir.

Klasik olarak gerekirlik şeklini üç başlıkta toplayabiliriz:

- olabilirlik (possible) : Kaziyenin yanlış olaması gerekmiyorsa

- gerekirlik (necessary) : Kaziyenin yanlış olma ihtimali yoksa
- şartlı (contingent) : Kaziyenin yanlış olması gerekmiyorsa ve aynı anda kaziyenin doğru olması da gerekmiyorsa

Yukarıdaki bu üç grup için kaziyelerde gerekirlik aranabilir. Buradaki üçüncü madde biraz karışık gelebilir. Bu maddeye göre bir kaziyenin (önerme) aynı anda hem doğru hem de yanlış olmaması durumu basitçe “doğruluğu mümkün ama gerekli değil” şeklinde açıklanabilir.

Gerekirlik şeklini ayrıca gerekirlik dünyası açısından da incelemek mümkündür. Yani bir kaziye şayet fiziksel kurallar dahilinde gerekirse bu gerekliğe fiziksel gerekirlik (Physical alethic) ismi verilebilir. Benzer şekilde, fiziküstü (metafiziksel, tinsel) gerekirlikten de söz edilebilir. Örneğin “Aya yürüyerek gidip geldim” cümlesindeki gerekirlik fiziksel bir gerekirlik şeklindedir. Benzer şekilde “Allah ikidir” cümlesi İslami açıdan non-alethic (gerekirlik şeklinden yanlış) bir önermedir.

### Epistemic (Bilgisel) şekli

Bu şekil bilgiye dayalı olarak kaziyelerin (önerme) doğruluğunu sınar. Daha basit bir ifadeyle kişinin konu hakkında emin olması veya önermesini kesin olarak bildiği temellere dayandırması sorgulanır.

Örneğin “Ali borsanın yükseleceğine inanıyor” cümlesi ile ” Alinin bütün bilgisine dayanarak borsa yükselebilir” cümlesi arasında fark vardır. İlk cümlede bir kanaat bir düşünce dillendirilirken ikinci cümlede kişinin bilgisine dayanarak kesin bir yargı ifade edilmektedir. Yani ilk cümle örneğinde kişinin tam anlamıyla inanması beklenmez veya kişinin elinde aksini de gösteren durumlar bulunabilir. Ancak ikincisi kişinin bilgisine dayanarak kesinlik arz eder.

Örneğin “borsanın yükseleceğine inanıyorum” cümlesindeki bilgi şekli ile “Davud’un peygamber olduğuna inanıyorum” cümlesindeki bilgi şekli arasında fark vardır. Birinci bilgi tipinde fiziksel bilgilere dayanarak ulaşılmış bir önerme bulunurken ikinci tip cümlelerde tinsel (metafiziksel) bir inanış (kanaat, yargı, kaziye) bulunmaktadır.

Her iki cümle yapısında da ortaya konan yargı geçmiş bilgi ve tecrübeler üzerine inşa edilen bilgisel bir yargıdır. Dolayısıyla yargı kişiden kişiye değişmekle birlikte bütün insanlar aynı yargıda bulunsalar bile doğru olduğu sonucuna varılmaz.

Bu ifadeleri şekli mantıkta (modal logic) göstermek için iki önemli işlem (operator) kullanılır.

L harfi ile gösterilen işlemde ” X ...’e inanıyor” şeklinde önermeler ifade edilir.

M harfi ile gösterilen işlemlerde ise “X’in bütün bilgisine dayanarak Y doğru olabilir” şeklinde önermeler ifade edilir.

Bazı kaynaklarda L işlemi için  $\Box$  sembolü ve M işlemi için de  $\Diamond$  sembolü kullanılmaktadır. Bu işlemler kullanılarak kaziye denklemleri çözülebilir. Yani örneğimize geri dönecek olursak Ali borsanın yükseleceğine inanıyorsa (  $Lp$  ) şeklinde gösterilebilir. Benzer şekilde “Alinin bütün bilgisine dayanarak borsa yükselecektir” cümlesini de (  $Mp$  ) olarak gösterebiliriz.

Bu iki cümle arasında kişini bilgisine dayanarak ortaya konan şekil farkı bulunmaktadır.

## **SORU-29: Zamansal Mantık (Temporal Logic) hakkında bilgi veriniz.**

Bilgisayar mühendisliğinin önemli parçalarından birisi de modellemedir. Çeşitli alanlarda veri modellemesi yapılan bilgisayar bilimlerinin, modellemeye ihtiyaç duyduğu bir konu da zaman modellemesidir. Yani kazyelerin (önerme, predicate) ifade ettikleri zamanı modellemek için bir sistem geliştirilmesi gerekmektedir.

Aslında antik yunan ve Aristo zamanından beri üzerinde çalışılan zaman modellemesi konusunda yapılan ilk çalışmalar nispeten ilkel sayılabilecek birinci dereceden zamansal mantık modelleri üzerine kuruludur.

Bu mantığın durumsal veya evrensel değişkenler üzerine kurulu olanına kazyeler mantığı (önergeler mantığı , propositiona logic) ismi verilir.

Yine aynı zamansal mantığın ikili ihtimaller ve mantık denklemleri üzerine kurulu olması durumunda da bool cebiri, ikili cebir (boolean algebra, binary logic) isimleri verilir.

Yukarıdaki bu üç farklı mantığı anlatmak için bir örnek üzerinde duralım. Örneğin “İnsanlar ölümlüdür” kazyesinin doğruluğunu sorgularken zamansal bir bakış izlenmez ve bütün zamanlardaki doğruluğu sınanır.

Buna karşılık “Karnım aç” kazyesinin doğruluğu zaman göre değişmektedir. Yani bazan aç ve bazan tok olabilirim. Bu durumda ise bütün zamanlar değil sadece anlık bir tahlil yapılır. Yani aynı anda hem aç hem de tok olamam dolayısıyla tahlil eden kişi belki bu çelişkiyi sınavabilir.

Yukarıdaki bu iki örnekte kullanılan mantığın farkını görmeliyiz. Birincisi kazyeler mantığı (predicate calculus) olarak incelenebilirken ikincisi zamansal mantık (temporal logic) için bir örnek olabilir.

Ancak unutulmaması gerken nokta her mantık için doğruluğun sorgulanabilir olduğudur.

### **Mantıksal İşlemler (Logical Operators)**

Kısacası Zamansal mantık için geçerli olan ik durum söz konsudur. Bunlardan birincisi mantıksa önermenin doğruluğunun sınanmasıdır. Bu işlem mantıksal operatörler (logical operators) ile yapılır ve bu operatörler

- ve (And)  $\wedge$
- veya (or)  $\vee$
- ise (implication)  $\rightarrow$
- değil (not)  $\neg$

şeklinde sıralanabilir.

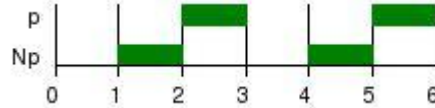
### **Zamansal İşlemler (Temporal Operators)**

Ayrıca zamana ait işlemler (operators) ise aşağıdaki tabloda verildikleri gibi sıralanabilir:

- Tekli işlemler (unary operators):
  - Sonra (Next)

- Gelecek (Future)
- Genel (Globally)
- Hep (All)
- Mevcut (Exists)
- İkili işlemler (Binary Operators):
  - -e Kadar (Until)
  - -den sonra (Release)

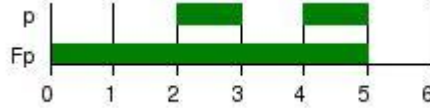
Yukarıda listelenen bu işlemleri aşağıdaki şekillerle göstermek ve örneklerle açıklamak mümkündür.



Yukarıdaki şekilden sonra operatörü anlaşılabilir. Buradaki 1-2 ve 4-5 aralığındaki eylemden sonra 2-3 ve 5-6 aralığındaki eylem gelmiştir.

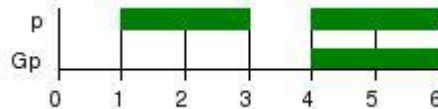
İşlem (operatör) N harfi ile gösterilir (bazı kaynaklarda neXt kelimesindeki X harfi ile de gösterilir). Kazıye (önerme, haber, predicate) p harfi ile gösterilmiştir. Dolayısıyla ilk eylem (1-2 aralığındaki eylem) bittikten sonra diğer eylem başlamıştır (2-3 aralığındaki eylem) yani p kaziyesi, Np'ye göre sonradır.

Örneğin “Şöyle bir olaydan sonra p gibi bir olay olacaktır” tipindeki cümleler bu gruba örnektir.



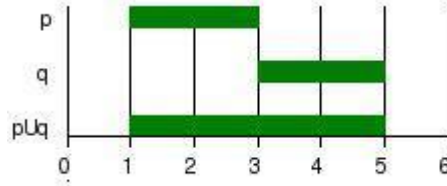
Yukarıdaki şekilde gelecek işlemi (future operator) görülebilir. İşlem F harfi ile gösterilir. Burada p önermesi Fp'ye göre ilerdedir. Bazı kaynaklarda nihayet (finally) olarak da geçmektedir.

Örneğin “gelecekte bir zamanda p gibi birşey olacaktır” şeklindeki cümleler bu gruba örnektirler

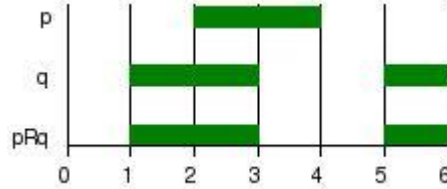


Yukarıdaki işlem genel zamanı ifade etmektedir. (Global operator) Önerme p sonradan gelen bir yolu ifade etmektedir.

Örneğin “p durumunda hep böyle olacaktır” şeklindeki cümleler bu mantık işlemine örnektirler.



Yukarıdaki örnek ikili işlemlerden (binary operators) -e kadar (until) ifadesi için çizilmiştir. Yani p ve q önermelerinden p'nin bittiği anda başlayan q ifadesi  $pUq$  ile gösterilebilir.



Yukarıdaki şekilde ise -den sonra (release) işlemi gösterilmiştir. Burada p ve q eylemleri için p'den sonra q anlamında işleme tabi tutulması söz konusudur.

### Birleşik işlemler

Yukarıdaki mantık işlemlerinin bir kısmının aynı örnek üzerinde kullanılması da mümkündür. Aşağıda bir veya iki önerme için birden fazla işlemin aynı anda kullanıldığı örnekler verilmiştir.

### **SORU-30: Minimax Ağaçları (Minimax Tree) hakkında bilgi veriniz.**

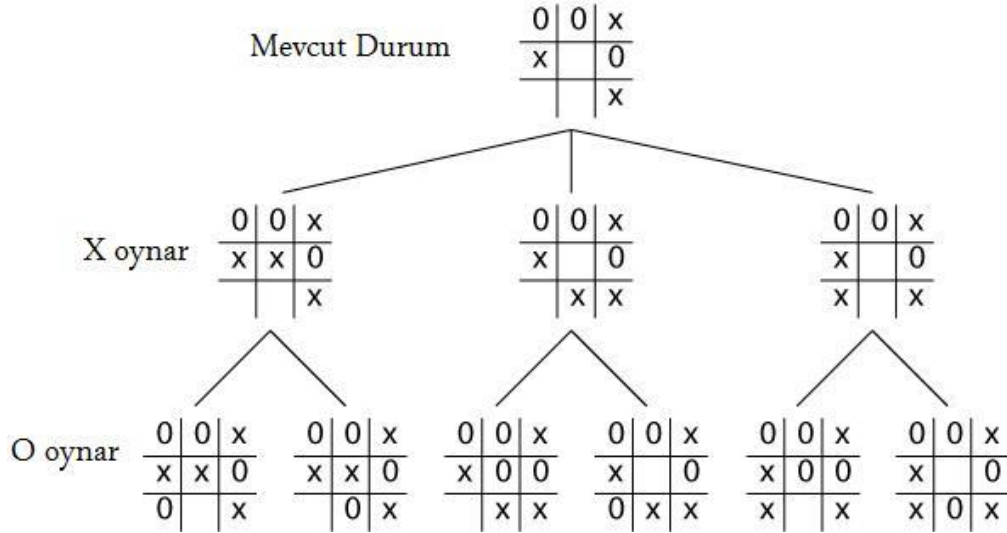
Bilgisayar mühendisliğinde, yapay zeka konusunda kullanılan bir karar ağacı türüdür. Aslında minimax ağaçları bilgisayar bilimlerine işletme bilimindeki oyun teorisinden (game theory) girmiştir.

Temel olarak sıfır toplamlı bir oyunda (zero sum game), yani birisinin kaybının başka birisinin kazancı olduğu (veya tam tersi) oyunlarda karar vermek için kullanılışıdır. Örneğin çoğu masa oyunu (satranç, othello, tictactoe gibi) veya çoğu finansal oyunlar (borsa gibi) veya çoğu kumar oyunları sıfır toplamlı oyunlar arasında sayılabilir (yani birisinin kaybı başka birisinin kazancıdır ve sonuçta toplam sıfır olur).

Yukarıda bahsedilen bu oyunlarda doğru karar verilmesini sağlayan minimax ağacı basitçe kaybı asgariye indirmeye (minimize etmeye) ve dolayısıyla kazancı azamiye çıkarmaya (maximize etmeye) çalışır.

Ağaç basitçe her düğümde (node) farklı alternatiflerin değerlerini hesaplar. Son düğümden (yapraklardan ,leaf) yukarıya doğru değerleri seçerek gelir ve en sonunda bütün ağaçtaki en doğru seçenek seçilmiş olur.

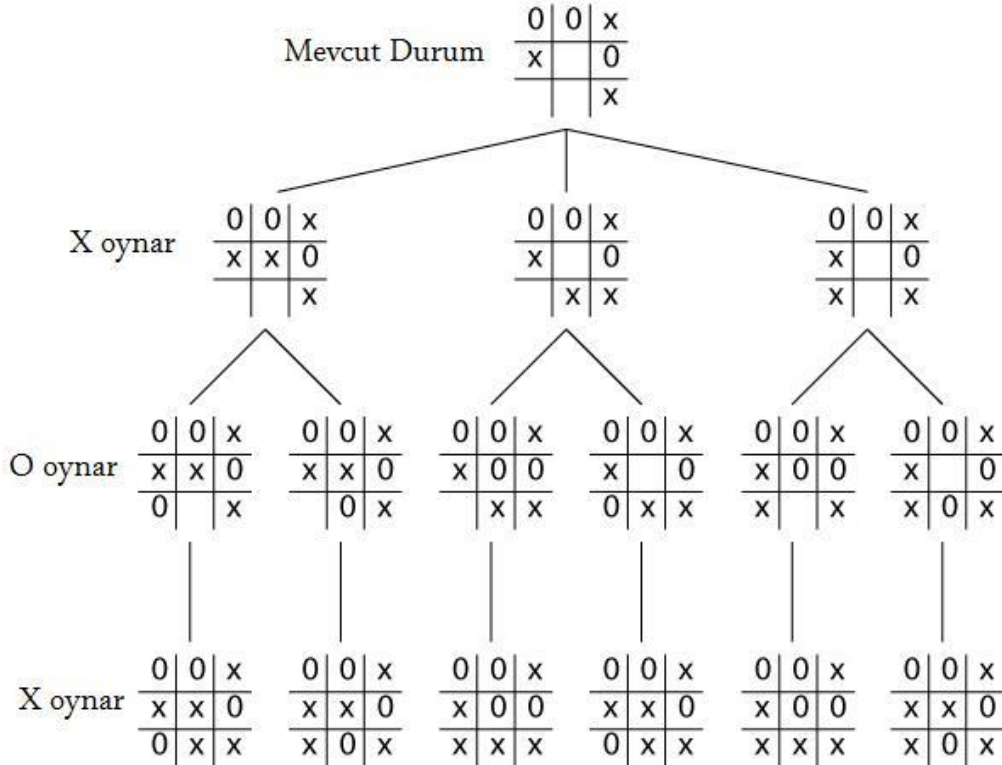
Örneğin Tic Tac Toe oyununu ele alalım.  $3 \times 3$  boyutundaki kare bir tabloda sırasıyla X ve O harflerinin taraflarca yazıldığı bu oyunda, oyunun durumuna göre aşağıdakine benzer bir karar ağacı çıkması mümkündür:



Yukarıda ağacın kökünü (root) oluşturan ilk durumda oyunun mevcut hali görülmektedir. Ardından karar verecek taraf olan ve tahtaya X sembolü koyan oyuncu kendi oyununu analiz eder. X 'in oynanabileceği 3 ayrı ihtimal bulunmaktadır ve bütün bu ihtimaller ağaçta farklı birer alt düğüm (node) olarak gösterilmiştir.

Ağacın daha alt seviyesinde ise X'in oynanabileceği her ihtimalde, O'nun oynanabileceği ihtimaller gösterilmiştir.

Bu durumda X oynayacak olan oyuncu bütün ihtimalleri görmüş olur ve ilave olarak kendisinin yapacağı son hamleleri aşağıdaki şekilde hesaplar:



Yukarıdaki şekile X'in yapacağı son hamleninde hesaplanışını görüyoruz. Bir minimax ağacının hesaplanması sırasında kaç seviye gidileceğine seviye (ply) ismi verilir. Örneğin

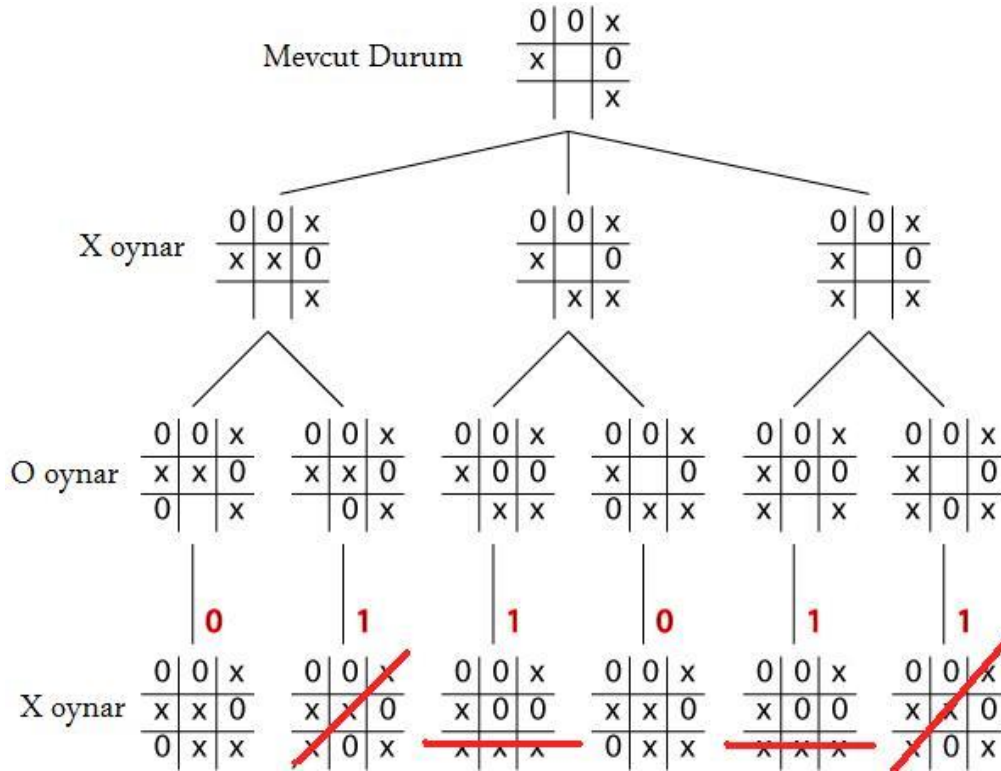
yukarıdaki ilk şekildeki minimax ağacımı 2 seviye (ply) bir ağaçken, yukarıdaki son ağacımız 3 seviyeli bir ağaçtır.

Minimax ağacının seviyesinin artması sonucun daha kesin bulunması ve yapay zekamızın başarısını arttırır. Ancak bunun karşılığında bilgisayarın hafızasında (RAM, memory) daha fazla bilginin tutulması gibi bir bedel ödenir. Örneğin satranç veya GO gibi ihtimallerin çok yüksek olduğu oyunlarda seviyenin belirli bir limitin altında tutulması gerekir.

Bilgisayarın yukarıdaki ağaca bakarak bir karar vermesi çok güçtür. Kararın sayısal bir değere oturması için yukarıdaki her hamle durumunu puanlamamız ve bundan sonra bilgisayarın minimax yaklaşımı ile seçim yapmasını istememiz gerekir.

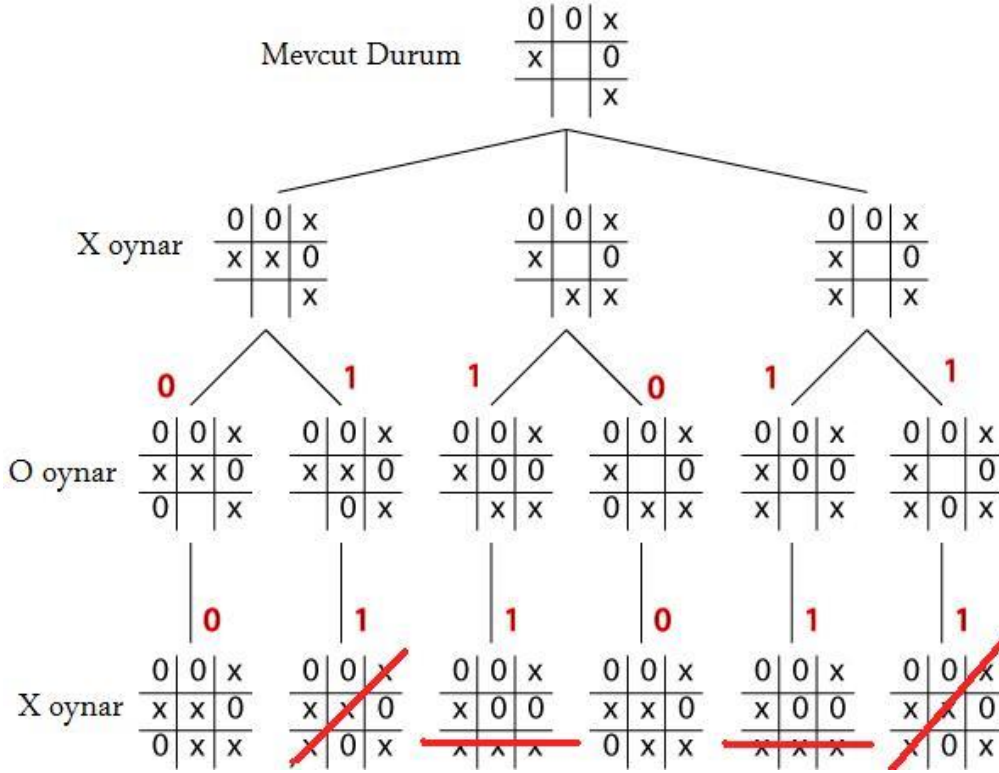
Örneğin bilgisayarın (Burada X olarak oynadığını düşünelim) kazanma durumlarına 1 ve kaybetme durumlarına 0 diyelim.

Bu durumda ihtimaller ve puanlar aşağıdaki şekilde olacaktır:



Yukarıdaki şekilden de anlaşılacağı üzere bilgisayar kazancının en fazla olduğu ihtimali seçmek ister. Basit bir hesapla bilgisayarın kazanç durumlarının toplamalarını ağacın bir üst seviyesine taşıyalım:



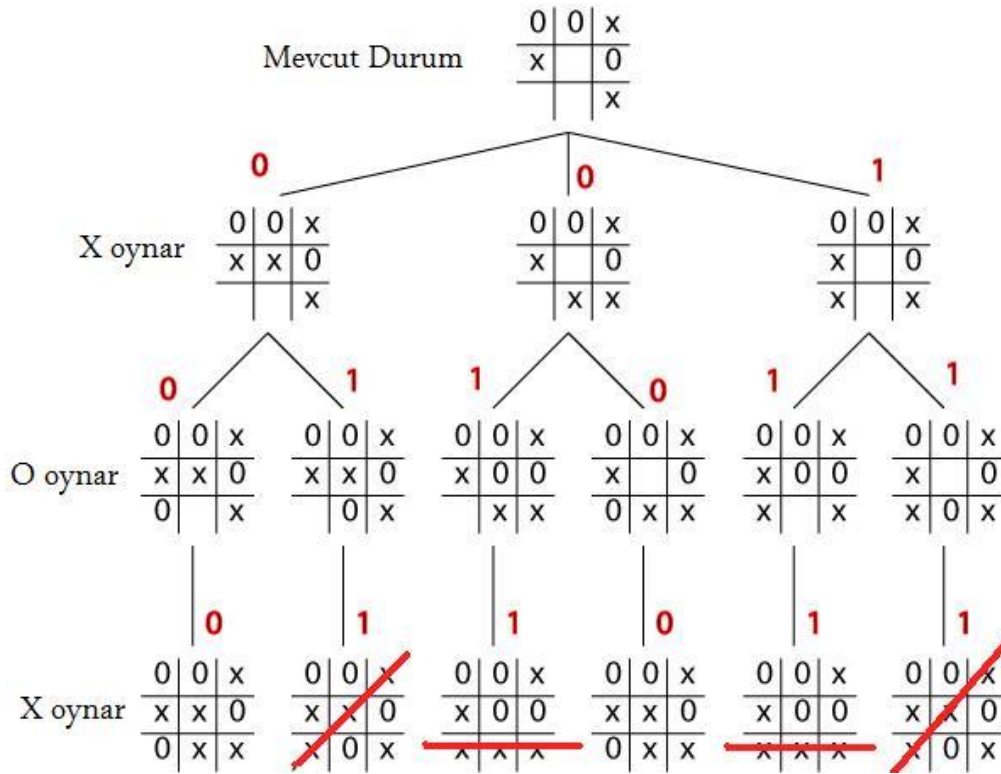


Yukarıda ağacın son seviyesinden karar vermemiz gereken pozisyona doğru bir seviye puanlar taşınmıştır. Yani kararın verilmesi gereken pozisyon ağacın kökündeki (en üst seviyesindeki) pozisyonudur ve oyun sonu en altta gösterilmektedir. İşte bizde puanlarımızı karara bir seviye yaklaşıyoruz.

Bu yaklaştırma işlemi sırasında dikkat edilmesi gereken X'in hamle sırasındaki azami (maximum) değerlerin taşınmasıdır çünkü X kendi hamlesinin en çok puan getirdiği durumu almak ister. Ancak yukarıdaki grafikteki son seviyede X'in tek hamlesi olduğu için mecburen (bir seçim yapılmaksızın) puanlar bir seviye yukarı taşınmıştır.

O'nun hamlesinin bulunduğu bir üst seviyede ise bir seçim yapılabilir. Çünkü O hamlesini iki farklı yere yapabilir. Bu durumda O'nun kendisi için en avantajlı yere oynayacağını düşünürsek bu seviyede asgari (minimum) değerlerin alınması gerekir.

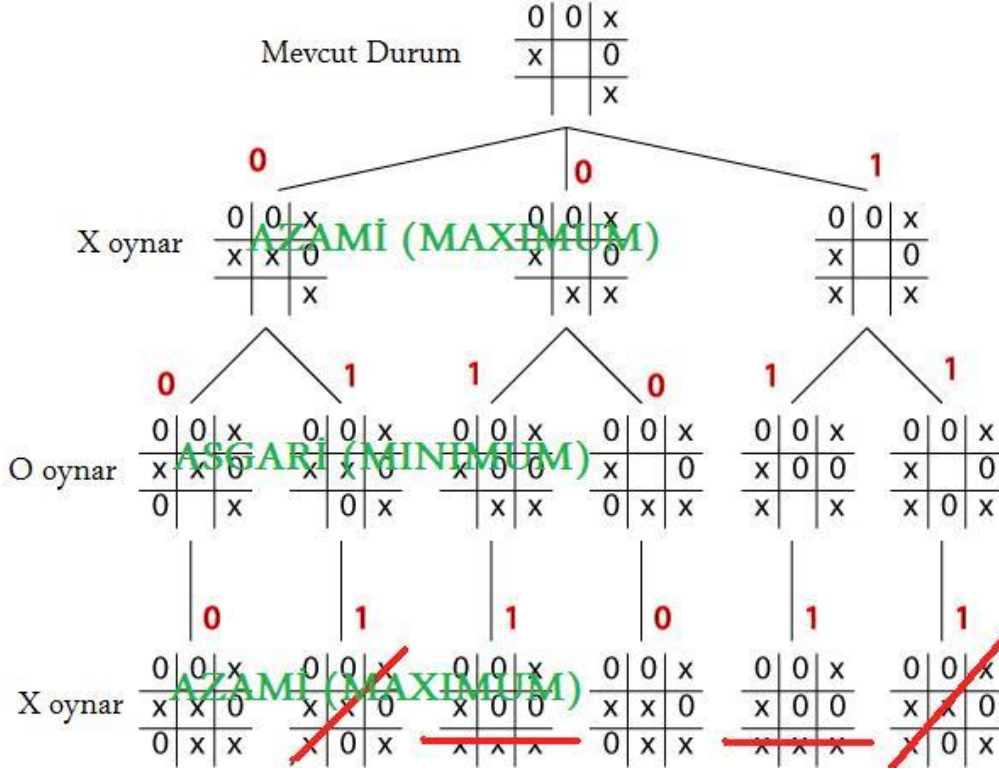
Aşağıda O'nun hamlesi için alınmış asgari değerler bulunuyor:



Görüldüğü gibi ihtimallerin en düşük değerleri alınmıştır.

Taşınan bu son seviyeyle artık bilgisayar hamlesine karar verebilir çünkü mevcut durum için hangi hamlenin en kârlı olduğu (max) görülmektedir. Yukarıdaki ağaçta en sağdaki değer olan 1 puanındaki hamleyi yapması durumunda bilgisayar her ihtimalde oyunu kazanmaktadır.

Sonuç olarak alınan değerler aşağıdaki şekilde gösterilebilir:



Yani bir min bir max alınmaktadır ve bu işlem bu şekilde devam etmektedir. Ağacımızın ismi de zaten buradan türetilmiş minimax ağacı olarak geçmektedir. Aynı anlamda farklı başlangıç değerinde maximin ağacı da literatürde geçmektedir. Bu ağaç da tamamen aynıdır ancak kaygı kendi hamleleri üzerine değil rakibin hamleleri üzerine kuruludur.

### SORU-31: Semafor (Semaphore, Flama, İşaret) hakkında bilgi veriniz.

Bilgisayar bilimlerinde özellikle de işletim sistemi ve müşterek programlamada (concurrent programming, eş zamanlı programlamada) sıkça kullanılan bir eşleme (synchronization) yöntemidir. Yani birden fazla işin (process) aynı anda çalışması halinde birbirleri için risk arzettikleri kritik zamanlarda (critical sections) birbirlerini beklemesini sağlayan bir mekanizmadır.

Basitçe bir değişken veya bir mücerret veri yapısı (abstract data type, soyut veri tipi, adt) üzerine kurulmuş olup, bu veri yapısı içerisindeki bilgiye ve fonksiyonlara göre çalışmaktadır.

Semaforların çalışması sırasında bölünmezlik (atomicity) ön plandadır. Yani bir semafor'un içerisinde yapılan birden fazla iş, program tarafından sanki tek bir iş gibi algılanmalı ve araya başka işin girmesine izin verilmemelidir.

Semaforlar kullanım alanları ve tasarımları itibariyle ikiye ayrılır:

- ikili semaforlar (binary semaphores)
- tam sayı semaforları (integer semaphores)

ilk semafor tipi olan ikili semaforlar sadece iki işlem (process) arasında eşleme (Synchronization) sağlar ve üçüncü bir iş için tasarlanmamıştır. Tam sayı semaforları ise istenilen miktarda işlemi kontrol edebilir.

### **SORU-32: Atomluluk (Atomicity) hakkında bilgi veriniz.**

Latince bölünemez anlamına gelen atom kökünden üretilen bu kelime, bilgisayar bilimlerinde çeşitli alanlarda bir bilginin veya bir varlığın bölünemediğini ifade eder.

Örneğin programlama dillerinde bir dilin atomic (bölünemez) en küçük üyesi bu anlama gelmektedir. Mesela C dilinde her satır (statement) atomic (bölünemez) bir varlıktır.

Benzer şekilde bir verinin bölünemezliğini ifade etmek için de veri tabanı, veri güvenliği veya veri iletimi konularında kullanılabilir.

Örneğin veri tabanında bir işlemin (transaction) tamamlanmasının bölünemez olması gerekir. Yani basit bir örnekle bir para transferi bir hesabın değerinin artması ve diğer hesabın değerinin azalmasıdır (havale yapılan kaynak hesaptan havale yapılan hedef hesaba doğru paranın yer değiştirmesi) bu sıradaki işlemlerin bölünmeden tamamlanması (atomic olması) gerekir ve bir hesaptan para eksildikten sonra, diğer hesaba para eklenmeden araya başka işlem giremez.

Benzer şekilde işletim sistemi tasarımı, paralel programlama gibi konularda da bir işlemin atomic olması araya başka işlemlerin girmemesi anlamına gelir.

Örneğin sistem tasarımında kullanılan check and set fonksiyonu önce bir değişkeni kontrol edip sonra değerini değiştirmektedir. Bir değişkenin değeri kontrol edildikten sonra içerisine değer atanmadan farklı işlemler araya girerse bu sırada problem yaşanması mümkündür. Pek çok işlemci tasarımında buna benzer fonksiyonlar sunulmaktadır.

Genel olarak bölünmezlik (atomicity) geliştirilen ortamda daha düşük seviyeli kontroller ile sağlanır. Örneğin işletim sistemlerinde kullanılan semafor'lar (semaphores), kilitler (locks), koşullu değişkenler (conditional variables) ve monitörler (monitors) bunlar örnekler ve işletim sisteminde bir işlemin yapılması öncesinde bölünmezlik sağlayabilirler.

Kullanılan ortama göre farklı yöntemlerle benzer bölünmezlikler geliştirilebilir. Örneğin veritabanı programlama sırasında koşul (condition) veya kilit (lock) kullanımı bölünmezliği sağlayabilir.

### **SORU-33: Zamanın Gerçekdışılığı (Unreality of Time, Hayal-i Vakit) hakkında bilgi veriniz.**

Literatüre McTaggart tarafından kazandırılmış olan bu çalışmaya göre zaman gerçek bir varlık değildir. Zamanın gerçek olmadığını ispatlamak için iki önerme ortaya atılır:

1. Olayların zamanlarını açıklamak için başka olaylara ihtiyaç duyması (yani aslında zaman kavramı olaylar sıralaması olarak ele alınırsa bir olayın zamanı ancak başka olayın zamanına izafe edilerek anlaşılabilir)
2. Olayların zamanlamalarının sabit oluşu. Yani bir olay başka bir olaydan sonraysa hep sonradır, bu dizilimin değişmesi söz konusu olamaz. Yani zamandaki olayların yer değiştirmesi mümkün değildir.

Yukarıdaki bu iki kaziyeden (önerme, predicate) yola çıkarak zamanın hayalî olduğu sonucuna varılabilir. Burada ilk algılanması gereken yukarıdaki ilk kaziyenin zamandaki dizilimi ifade etmede yetersiz olduğu ve hemen ardından yukarıdaki iki kaziyenin birbiri ile çeliştiğidir.

### **SORU-34: Zamani (Temporal, Zamansal, Zamane, Mevku) hakkında bilgi veriniz.**

Zamani kavramları açıklamak için kullanılan terimdir. Basitçe insanın zaman algısı ve bu algı üzerine kurulu olan felsefi ve yaşamsal düşünceleri geçmiş, şimdi ve gelecek üzerine kuruludur. Bu durumda zamani kavramlarda bu değerlerin etkisidine şekillendirilmektedir.

Aslında din ve felsefede derin tartışmalar açmış bu konuya insanlığın var olduğu tarih boyunca hemen her çağda rastlanmaktadır ve neredeyse bütün filozofların çalışmalarında yer bulmuştur. Örneğin din açısından zamanın yaratılmış olması (tanrıdan başka yaratılmayanın olmayışı), insan olarak bizlerin kısıtlı bir zaman kavramı içerisinde yaşıyor olduğumuz sonucunu doğurmaktadır.

Gerek ontolojik gerek epistemolojik tartışmalara da sebep olan bu konuda örneğin Heidegger'in "Sein und Zeit" (zaman ve varlık) veya Nietzsche'nin Sonsuz Dönüş (Eternal Recurrence) çalışmaları örnek olarak gösterilebilir.

Bilgisayar bilimleri açısından konunun önemi özellikle yapay zeka çalışmalarında ön plana çıkmaktadır. Çalışma alanı olarak insan zekasını modellemek ve daha çok taklit etmek amacını belirlemiş olan yapay zeka, konu ile ilgili değişik alt alanlarda çalışmaktadır. Örneğin soru cevaplama (question answering) , doğal dil işleme (natural language processing) bu konulardan bazılarıdır.

Bu çalışmalardaki amaç, insanın zaman algısını modelleyerek bu algı üzerinden suni bir zeka üretmek veya taklit etmektir.

Zaman kavramı ayrıca mühendisliğin pekçok alanında bir parametre veya boyut olarak da kullanılır ancak bu kullanımlarda zamanın anlamının önemi yoktur. Örneğin gerçek zamanlı sistemlerde (real time systems) zamana bağlı olarak işlerin takip edilmesi gerekir ancak burada zaman sadece takip edilen ve anlamıyla ilgilenilmeyen bir bilgidir.

### **SORU-35: Tehlike (Hazard) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde özellikle de mantıksal devre tasarımı sırasında karşılaşılan bir durumdur. Basitçe sistemde oluşan veya oluşabilecek tehlikeleri ifade eder. Yani örneğin sistemdeki kapıların (ve, veya, yahut kapıları) yanlış çalışması sonucunda oluşan tehlikelerdir. Temel olarak 3 ayrı grupta toplamak mümkündür:

- Sabit Tehlikeler (Static Hazards)
- Mühharrik Tehlikeler (Dinamik Tehlikeler, Dynamic Hazards)
- Fonksiyonel Tehlikeler (Functional Hazards)

**Sabit tehlikeler** basitçe, girdinin (input) değişmesi halinde sonucun (output) değişmemesi gerekirken değişmesi durumudur.

Bu tehlike durumu için iki ayrı çözüm olabilir. Birincisinde devreye geciktirmek (Delay) için ilave devrelerin eklenmesi, ikincisinde ise devrenin hatasının düzeltilmesi için ilave devrelerin eklenmesi söz konusu olabilir. Sabit tehlikeler, müdahale edilmesi nispeten basit tehlikelerdir.

**Dinamik tehlikelerde** ise sorun genelde bir girdi için farklı zamanlarda farklı hatalı sonuçların alınması şeklinde tanımlanabilir. Yani sabit tehlikede olduğu gibi sürekli aynı hata değil, ya farklı hatalar ya da bazan doğru sonuçların alınması durumudur.

Basitçe bir devreden dinamik tehlikelerin kaldırılması için bütün sabit tehlikelerin kaldırılması yeterlidir. Çünkü genelde dinamik hatalar büyük ve karmaşık devrelerde, alt parçaların sabit tehlikelerinden ortaya çıkmaktadır.

**Fonksiyonel Tehlikeler** çözülmesi imkansız olan tehlikelerdir. Tanım olarak birden fazla girdinin (input) aynı anda değişmesi sırasında oluşan tehlikelerdir. Bu tehlikelerin ortadan kaldırılmasının tek yolu, tek girdi (input) ile tehlikenin oluşturulması veya tespit edilmesidir. Bu sayede tehlike dinamik veya sabit bir tehlike haline dönüşerek çözülebilir.

### **SORU-36: İçerik Bağımsız Gramerler için Pompalama Önsavı (Pumping Lemma for Context Free Grammers) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde bir dilin, içerik bağımsız gramer (context free grammar, CFG) ile gösterilemeyeceğini ispatlamaya yarar. Yani pompalama ön savı sayesinde bir dilin CFG olmadığı ispatlanabilir ancak olduğu ispatlanamaz. Şayet pompalama önsavını geçemiyorsa CFG değildir denilebilir ancak geçmesi olmasını gerektirmez.

Pomplama önsavı (pumping lemma) kısaca bir dili aşağıdaki gramere uydurmaya çalışır:

$$s = uvxyz$$

Elimizde ispatı ile uğraştığımız L dili olsun ve yukarıdaki s kelimesini bu dilden bir kelime olarak üretelim. Ve bu kelime  $|vxy| \leq p$ ,  $|vy| \geq 1$  (p burada pompalama boyutudur) şartlarını sağlasın. Şimdi bu kelimeyi aşağıdaki şekilde pompayalım:

$$s = uv^i xy^i z$$

Şayet bu pompalama sırasında üretilen  $i \geq 0$  için kelimelerde L dilindense o halde bu dil içerik bağımsız gramer CFG ile ifade edilebilir, şayet oluşan kelimeler L dili tarafından kapsanmıyorsa bu durumda da L dili, CFG olarak ifade edilemez sonucuna varılabilir.

### **Örnek:**

Pompalama önsavını (pumping lemma) kullanarak aşağıdaki dilin CFG olmadığını ispatlayalım:

$$L = \{a^i b^i c^i \mid i > 0\}$$

Yukarıdaki dili, tanımımızdaki

$$s = uvxyz$$

şekline getirmeye çalışalım ve bu sırada  $|vxy| \leq p$ ,  $|vy| \geq 1$  şartlarını sağlamaya çalışalım. Bu durumda ilk kelimemiz:

$$s = a^l b^l c^l$$

olarak yazılacaktır. Bu yazımdan u ve z değerlerinin boş olabileceğini düşünürsek

$$v = a$$

$$x = b$$

$$y = c$$

olarak yazılabilir. Bunun dışındaki şartların hiç birisi  $s = uvxyz$  şartını ve  $s = a^i b^i c^i \mid i > 0$  şartını aynı anda sağlamaz.

Şimdi ilk kelimemizi yazdığımıza göre, kelimemizi pompalayarak çıkan sonuçların yine bu dilde olup olmadığını kontrol edelim:

Yukarıdaki denemede  $i=1$  için yazmıştık, şimdi  $i=2$  için  $uvxyz$  değerlerini bulmaya çalışalım.

Böyle bir değer bulunamaz çünkü

$$s = a^2 b^2 c^2$$

değerini ancak

$$s = uv^i xy^i z$$

pompalamasında  $i=2$  yazarak sağlamaya çalışabiliriz ve bu durumda da x değeri olan b değeri 1 tane kalacağı için problem olacaktır. Daha basit bir ifadeyle CFG gösteriminde 3 değer birden aynı sayıda olması garanti edilemez, ancak iki değer aynı sayıda olabilir. Çünkü yukarıdaki v ve y değerleri pompalandığında artarken x değeri tek değer olarak kalacaktır.

Benzer şekilde

$$u = a$$

$$v = b$$

$$y = c$$

olması durumunda u,

$$v = a$$

$$y = b$$

$$z = c$$

oması durumunda da  $z$  değerleri tek kalacak ve diğer harfler pompalanırken dengeyi bozacaktır.

Yukarıdaki durumda  $L = \{a^i b^j c^i \mid i > 0\}$  dilinin bir CFG gösterimi ile gösterilemeyeceği ispatlanmış olur.

### **SORU-37: Pompalama Önsavı (Pumping Lemma) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde dil tasarımı (language design, compiler design) konusunda önemli araçlardan birisidir. Bu önsava (lemma) göre şayet bir dil, bir herhangi bir gruba ( içerik bağımsız dil (context free language) veya düzenli ifadeler (Regular expression) yada farklı bir dil grubu ) dahil olarak kabul ediliyorsa, bu dil ne kadar pompalanırsa pompalansın yine aynı dil grubuna dahil olmalıdır.

Daha açık bir ifadeyle, bir dil sonsuz kümeyi kapsıyorsa (o dil kurallarıyla üretilebilecek sonuçların bir sınırı yoksa) bu dili ne kadar pompalarsak pompalayalım sonuçta yine aynı dil grubundan olmalıdır. Ve ancak bu sayede bu dilden üretilebilen bütün sonuçların aynı dil grubundan olduğu iddia edilebilir.

Yukarıdaki tanımlarda geçen pompalamak işlemi bir dil gramerinin bir kısmının şişirilmesi, arttırılması veya çoğaltılması olarak algılanabilir.

Pompalama önsavı genelde bir dilin iddia edildiği dil grubunda olmadığını ispatlamak için kullanılan bir çelişki ile ispat (proof by contradiction) yöntemidir.

### **SORU-38: Uyum (Agreement, Kabul, Bağıt, Mutabakat) hakkında bilgi veriniz.**

Bilgisayar bilimlerinin bir çalışma alanı olan doğal dil işleme (Natural language processing) konusunda ve dolayısıyla dilbilim (linguistic) konusunda kullanılan bir kavramdır. Bu kavrama göre bir dilde aynı anlama gelen kelimelerin sayısı, cinsiyet, kişi veya duruma göre farklı kelimelerle ifade edilmesidir.

Örneğin Türkçede çoğul kelimelerin kullanımı bir uyum örneğidir. Kelimebilim (lexicology) açısından aynı ifade olan “el” ve “eller” kelimelerini ele alalım. Bu iki kelimedede aynı anlamdaki varlığı işaret etmekte ancak ikinci kelime çoğul anlamda kullanılmaktadır. Bu durumda, Türkçede kelimenin sayısal olarak uyumundan (agreement) bahsedebiliriz.

Benzer şekilde “evim” ve “evimiz” kelimeleri arasında da kişi anlamında bir uyum vardır. Yani ilk kelime 1. tekil şahsın varlığını işaret ederken ikinci kelime 1. çoğul şahsın varlığını işaret etmektedir.

Cinsiyete göre de ayırım yapılan Türkçedeki bu kelimeler genelde birleşik kelime veya yabancı dillerden giren kelimelerdir. Örneğin bilimadamı , bilimkadını veya biliminsanı ayırımı birleşik kelimeye örnektir. Müdür veya müdüre, memur veya memure ayırımı Türkçeye giren arap-fars akımına örnek olabilir. Benzer şekilde kuzen veya kuzin ayırımı Türkçeye giren latin (fransız) akımına örnek olabilir. Bu örneklerin hepsinde kastedilen aynı varlık iken, kelimeler arasında sadece kastedilen varlığın cinsiyet farkı söz konusudur.

Aslında Türkçede de rastlanan ve genelde Arapça, Farsça gibi dillerden giren kelimelerde yaşanan bu durum, Latin, Sami, Germen veya Slav dil ailelerinde sıkça görülmektedir. Örneğin



Japoncada da neredeyse az görülen kabullerin en sık görüldüğü dillerden birisi de Swahili dilidir.

### **SORU-39: Dolaylı sıralama (Indirect Sort, Gayrimüstakim sıralama) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde sıralama işleminin çok büyük veriler üzerinde yapılması durumunda tercih edilen bir sıralama yöntemidir. Basitçe sıralama işleminin doğrudan verilerin yerinin değiştirilemsi ile değil de daha çok bu verileri gösteren gösterici (pointer) veya nesne atıfları (object referrer) ile yapılmasıdır.

Örneğin sistemimizdeki öğrencileri sıralamak isteyelim. Her öğrenci bilgisi de sistemde 1MB kadar yer kaplıyor olsun (kişisel bilgileri ve bağlı olduğu bilgilerin tutulduğunu ve çok yer kapladığını düşünelim). Basit bir sıralama işleminde  $O(n \log n)$  vakit harcanağını düşünürsek, ortalama 100 elemanlı bir listenin sıralanması için 700 civarı işlem gerekecektir. Bunun anlamı bilgisayarın hafızasında (RAM) 700 MB civarı bir bilginin hareket ettirilmesi demektir.

Bunun yerine her veriyi gösteren birer atıf (pointer, gösterici) bulunsa ve bu atıflar hareket ettirilse (tahminen her birisi öğrencilerin numarasını ve öğrencinin hafızadaki (RAM) yerini tutan yapılardan oluşacağı için birkaç byte'ı geçmeyecektir, kabaca 10byte kapladığını düşünürsek) yapılan işlem, hafızada 7KB civarında verinin hareket ettirilmesi demektir.

İlkine göre oldukça avantajlı olan ikinci durumda, hafızada ekstradan bir bilgi tutulmuştur. Bu ilk başta hafızanın verimsizleşmesi (fazladan veri tutulması) anlamına gelse de aslında yapılan işlemlerin süresini kısaltması ve kopyalama işlemi sırasında açılan ilave hafıza alanlarını azaltması açısından bir kazanımdır.

Sonuçta elde edilen ve gerçek nesneleri gösteren bu ilave listenin kullanılması ile gerçek verilerin hafızada sıralanması da mümkündür. Ancak buna gerek duyulmaz çünkü veriye erişmek için bu ilave liste kullanılabilir.

### **SORU-40: Soru Cevaplama (Question Answering, QA) hakkında bilgi veriniz.**

Doğal dil işleme (natural language processing) çalışmalarının bir parçası olan soru cevaplama çalışmalarında amaç, doğal dildeki bir soruya doğru cevap verebilmektir. Soru cevaplama çalışmalarını bir kaç farklı şekilde gruplamak mümkündür.

Unutulmaması gereken, soru cevaplama çalışmalarının hedefinde insan gibi davranabilen bilgisayar programları bulunmaktadır. Yani ulaşılmak istenen noktada aynı soruyu bir insana sorduğunuzda alabileceğiniz olsaydı bir cevabı bilgisayardan almak istersiniz. Daha fazla bilgi için Turing Testine bakabilirsiniz.

#### **Soru alanına göre QA ( Question domain)**

Çoğu zaman çalışmaların kolaylaştırılması için soruların belirli bir metin içerisinde cevaplanması istenir. Yani bir insanın hayatı boyunca öğrendiği şeylerin bilgisayar tarafından bir insan gibi cevaplanması her ne kadar bu çalışmayı yapan kişilerin hayali olsa da henüz bu seviyeye ulaşmak için çok erkendir. Bunun yerine, günümüz çalışmalarında, bilgisayara yine doğal dilde yazılmış bir metin verilerek bu metin içerisinde sorulan soruların cevaplanması beklenmektedir. Bu anlamda soru cevaplama iki farklı soru alanı belirlenebilir:

- açık alanlı sorular (open domain)
- kapalı alanlı sorular (close domain)

ilk gruba örnek olarak belirli bir konudaki soruları düşünebiliriz. Örneğin sadece ilaç sektörüne yönelik sorular veya spor sorularını düşünebiliriz.

Örneğin “Türkiyenin en çok kazanan oyuncusu kimdir?” sorusuna sinema konusunda farklı bir cevap veya spor konusunda farklı bir cevap verilebilir. Şayet soruların alanları tanımlıysa bu problem ortadan kalkar, aksi halde soru cevaplayan program bu ayrımı yapabilmelidir.

### **Cevap tipine göre QA**

Bazı soruların cevapları kısa bir kelime ile verilebilen ve genelde belirli cevaplardır. Örneğin “Türkiye Cumhuriyetinin ilk Cumhurbaşkanı kimdir?” sorusunun tek bir cevabı vardır ve herkes bu soruya aynı cevabı verir.

Ancak “Türkiye Cumhuriyetinin ilk Cumhurbaşkanı nasıl birisidir?” gibi bir sorunun cevabı belirli bir cevap değildir. Yani sorunun her muhatabı farklı cevaplar verebilir. Bu durumda cevabın belirliliğine(deterministic) göre soru cevaplama çalışmalarını ayırmak mümkündür. Dolayısıyla cevap tiplerine göre sorular ikiye ayrılabilir:

- Belirli Cevaplar (Deterministic Answers)
- Belirsiz Cevaplar (Nondeterministic Answers)

### **Soru Cevaplama Yöntemine Göre:**

Soru cevaplama sırasında kullanılan yönteme göre yapılan çalışmaları iki ayrı grupta toplamak mümkündür.

- Sığ Yöntemler (shallow methods)
- Derin Yöntemler (deep methods)

Şayet kullanılan yöntem cümle içerisindeki basit yapıların yakalanması, cümle içinde geçen bazı kelimelerden sorunun anlamının çıkarılması gibi tekniklere dayanıyorsa bu cevaplama yöntemine sığ yöntem ismi verilmektedir.

Şayet kullanılan yöntem, kelimelerde başlayarak kelimebilim (morphology), cümle dizilimi (syntax) ve anlambilim (semantics) gibi aşamalara ayrılıyorsa hatta, hazf (ellipsis), dönüştü (anaphora), eş atıf çözümü (coreference resolution), ilgi belirsizliği (reference ambiguity), ontoloji ve benzeri problemleri de çözüyorsa sığ yöntem ismi verilebilir.

### **SORU-41: Genetik Algoritmalar (Genetic Algorithms) hakkında bilgi veriniz.**

Bilgisayar bilimlerinin doğa bilimlerinden (biyoloji) öğrendiği ve kendi problemlerini çözmek için kullandığı bir yöntemdir. Bu algoritmada genetikte kullanılan temel 3 işlem kullanılır. Bu üç işlemin alt tipleri ayrıca açıklanacaktır ancak bu üç temel işlem:

- Çaprazlama (Crossover)
- Mutasyon (Mutation)
- Başarılı gen seçimi (Selection)

Yukarıdaki ilk iki işlem aslında bir genin değişmesinde rol oynayan iki temel işlemdir. Bu iki temel işlemle (çarpazlama ve mutasyon) değişen genler arasından seçim yapılması (selection) ise genetik algoritmalarda kullanılan ve başarı elde etmeyi sağlayan yöntemdir.

Seçme işlemi için turnuva seçimi (tournament selection) veya tesadüf değeri içeren rulet seçimi (roulette wheel selection ) yöntemleri kullanılabilir.

#### **SORU-42: Aday Anahtar (Candidate Key) hakkında bilgi veriniz.**

Veritabanı teorisinde bir tabloyu tek başına tanımlayamay yeterli olan kolona verilen isimdir. Daha resmi bir tanımlamayla:

1. Bir ilişkide bir aday anahtarın tekrar edildiği iki satır bulunamaz
2. 1. adımda bahsedilen anahtarın bir alt kümesi bulunmaması.

İlişkisel veri tabanında bir tablo veya tablolar arası ilişki tanımlandıktan sonra elde edilen sonuç kümesindeki kolonlardan herhangi birisi yukarıdaki şartları sağlamak şartıyla aday anahtar olabilir.

Örneğin aşağıdaki ilişkiyi ele alalım:

A	B	C	D
1	a1	a1	a1
2	a1	b1	b1
3	c1	b1	b1
4	b1	c1	c1

Yukarıdaki ilişkide bariz bir şekilde A kolonu tekrarlanmayan ve sonuçta bütün tabloyu tek başına tanımlayabilen bir değişkendir. Dolayısıyla A kolonunun aday anahtar olduğu söylenebilir. Şayet A değişkeni ilişkide bulunmasaydı:

B	C	D
a1	a1	a1
a1	b1	b1
c1	b1	b1
b1	c1	c1

Yukarıdaki ilişkide bütün tabloyu tek başına tanımlayan bir anahtar bulunmamaktadır. Çünkü her değişkenin farklı değeri için diğer diğer değişkenler de değişmektedir.

Yukarıdaki ilişkide aday anahtar olarak {B,C} değişkenlerinin birleşimi alınacak olursa bu durumda bir aday anahtar elde edilmiş olur, çünkü bu iki tabloyu tanımlayabilmektedir. Yani ilişkideki her farklı satırı gösteren farklı bir B,C ikilisi bulunabilir.

Yukarıdaki örneğe özel olarak {B,C}, {~~C,D~~} [!!düzeltildi, buradaki C ve D kolon ikilisi aday anahtar olmak için yetersizdir, Sayın İlksen Cosar'a teşekkür ederim!!] ve {B,D} ikililerinin hepsi birer aday anahtar olarak kabul edilebilir.

Ancak {B,C,D} üçlüsünün bir aday anahtar olarak kabul edilmesi mümkün değildir. Çünkü bu durum aday anahtarın tanımında listelediğimiz 2. madde ile çelişir. Yani B,C,D üçlüsünün herhangi bir alt kümesinin (örneğin B,C) aday anahtar olamaması gerekmektedir.

Bu gibi daha alt aday anahtarlara bölünebilen üst anahtarlara (B,C,D gibi) super anahtar (Super key) ismi verilir.

Bu tanımdan yola çıkarak aday anahtar için en küçük süper anahtar tanımı da yapılabilir.

### **SORU-43: Java Bean hakkında bilgi veriniz.**

İsmi bir kahve makinesinden alan JAVA'nın ilk başlardan beri sembolü olan kahveden türemiş bir kavram olan java bean'in sembolü de kahve çekirdekleridir (bean kelimesini çekirdek olarak çevirmek mümkündür)

Basitçe java bean, tekrar kullanılabilir bir yazılım bileşenidir (reusable software component). Daha detaylı bakıldığında aslında her java bean bir yada birden çok sınıftan (class) oluşmuş ve tek başına çalışma yeteneği olan bileşenlerdir.

Tek başına çalışabilen bu bileşenler, daha gelişmiş programlar oluşturmakta kullanılırlar. Düşük bağıllık (coupling) ve yüksek uyumdaki (cohesion) program parçalarının bir araya getirilmesi ile daha modüler bir yaklaşım elde etmek ve büyük bir projeyi parçalara bölmek mümkündür.

JAVA Bean'lerin klasik nesne yönelimli modellemedeki (object oriented modelling) sınıf (class) bölmesinden farkı daha üst seviyeli olmaları ve nesnel bölmelerden daha çok kavramsal bölmelere gidilebilmesidir.

Örneğin bir yazılımda veritabanı kullanıyor olalım. Kullanıcıların şifreleri ile giriş yaptıkları bu veritabanı modülünü ele alalım. sınıf (class) yaklaşımında bir veritabanı sistemindeki kullanıcılar (kullanıcı bilgileri, isim, şifre gibi) , veritabanı nesneleri (tablolar) veritabanından geçici alınan bilgiler (veri tabloları, data table) veya kullanıcıların sistemde açtıkları oturumlar (session) ayrı ayrı birer sınıfta tanımlanır.

Oysaki bütün bu sınıfları birleştirerek tek bir java bean yapmak mümkündür. Tek başına çalışan bu java bean projenin bir modülü olup bu modül kullanılarak daha büyük sistemlerin inşası mümkündür.

JAVA Bean'lerin bir diğer özelliği ise geliştirme sürecinde çalıştırılabilir olmalarıdır. Sonuçta tek başına çalışan bu bileşenler, kod geliştirme (Development) zamanında da çalışabilir ve yazılımı geliştiren kişilere anlık olarak kullanma ve yaptığı her işlemi test etme imkanı sağlar.

JAVA Beanler ayrıca şu 3 özelliği barındırmalıdır:

- Public Constructor
- Serializable olmalıdırlar yani Serializable arayüzünü (interface) uygulamalıdırlar (implements)
- Erişim metodları bulunmalıdır (getter /setter methods)

Yani bir java bean'in hiç constructor fonksiyonuna ihtiyacı olmasa bile bir adet boş yapıcı (empty constructor) bulunmalıdır. Ayrıca hiçbir değişkenin erişimi public olmamalıdır. Bunun yerine bu değişkene erişen getter ve setter fonksiyonları bulunmalıdır.

#### **SORU-44: Bağlama (Coupling) hakkında bilgi veriniz.**

Yazılım mühendisliğinde modelleme sırasında sistemde bulunan varlıkların ilişkilerini belirlemeye yarayan bir terimdir. Genellikle yapışma (cohesion) teriminin tersi anlamda kullanılır. Yani yüksek bağlama (high coupling) düşük yapışma (low cohesion) anlamında gelmektedir.

Nesnelerin birbirine bağlanması (coupling) ve yapışması (cohesion) arasındaki en önemli fark bağlanmanın düşük olmasının yani yapışmanın yüksekte olmasının iki nesnenin birbirinden mümkün olduğunca izole olmasıdır. Yani bir nesnede yapılan bir değişim diğer nesnede güncelleme gerektirmez. Ayrıca iki nesne iyi tanımlanmış ve kontrolü kolay bir arayüz üzerinden iletişim içerisindedirler.

Sistem modellemeye ve nesneler arası ilişki tanımlamada düşük bağlama ve yüksek yapışma (low coupling high cohesion) sloganı kullanılır. Bu sayede nesnelerin birbirinden mümkün olduğunca bağımsız olması ve sistemin modüler bir tasarıma sahip olması hedeflenir. Ancak bazı durumlarda performans ve hız endişesinden dolayı bu kural çiğnenebilir.

#### **SORU-45: Principal Component Analysis hakkında bilgi veriniz.**

Bilgisayar bilimlerinde boyut indirmeye yarayan bir yöntemdir. Kısaca iki bilgi arasında bir bağlantı varsa bu bağlantı sayesinde iki veriden birisini tutmak ve bağlantıyı tutmak iki bilginin de geri bulunabilmesini sağlar. Kısaca PCA olarak da ifade edilen bu terime göre bir veri kümesinin (veri matrisinin , data matrix) kovaryans matrisinin (covariance matrix) veya tekil değer çıkarımının (singular value decomposition) yöntemi ile elde edilen basitleştirilmiş halidir.

#### **SORU-46: Eigenvalue (Özdeğer) Eigen vector (Öz yöney) Eigen Space (Öz Uzak) hakkında bilgi veriniz.**

Bir yöneyin (vector) bir dönüşüme (transformation) uğramasından sonra boyutunun değişmesinden bağımsız olarak hâlâ yönü aynı kalıyorsa bu dönüşüm yöneyine (vector) öz yöney (eigen vector) ismi verilir.

Bu yön değiştirmeyen ancak uzunluk (büyüklük) değiştiren öz yöneyin yapmış olduğu değişim aslında sayısal bir uzunluk olarak hesaplanabilir (örneğin yöneyin iki misline çıkması veya yarısına inmesi gibi) işte bu hesaplanan sayısal değere (sabite, scalar) öz değer (eigen value) ismi verilir.

Yukarıdaki bu öz değerlerin (eigen value) oluşturduğu dönüşümlerin 0 olduğu matristir.

Bilgisayar bilimlerinde bir varlığın yönünü kaybetmeden değer değiştirmesi durumunda eigen öneki sıkça kullanılır. Örneğin eigenfunction (öz fonksiyon) yönü aynı olan ancak fonksiyon uygulandıktan sonra sadece miktarın değiştiği fonksiyon anlamındadır. Benzer şekilde eigenstate (öz durum), eigenmode (öz hal), eigenfrequency (öz frekans, öz sıklık) aynı anlamları katan ön eklerdir.

#### **SORU-47: Dik Vektörler (Orthogonal Vectors) hakkında bilgi veriniz.**

Matematikte iki vektörün birbirine dik olması durumuna ortogonal (dik) yöney (vector) ismi verilir.

Bilgisayar bilimlerinde ve felsefede iki kavramın birbirine etkisinin 0(sıfır) olması anlamında kullanılır. Yani kabaca bu iki vektörün ortak bir izdüşümü söz konusu değildir. Şayet vektörler dik olmasaydı bir vektörü diğerinin trigonometrik fonksiyonu (örneğin sinüs veya kosinüs) katsayısı olarak yazmak mümkün olurdu.

Örneğin bir insanların boyları ve kazandıkları maaş arasında hiçbir bağlantı yoktur. Şayet bu iki veriyi bir düzlem üzerinde göstermek isteseydik maaş ve boy iki farklı boyut (eksen) olacaktı.

Buna karşılık insanların boyları ve kiloları arasında tam olmasa da bir bağlantı vardır. İnsanların boyu uzadıkça kilosu artar (Arımayada bilir ama genelde artar) dolayısıyla bu iki vektör arasında bir bağlantı olması söz konusudur.

Yukarıdaki örneği modellemek için iki vektörün kullanılmasının yanında olayları ve modellemeleri açıklamak için de kullanılabilir. Örneğin a fonksiyonu b fonksiyonu ile ortogonaldır demek bu iki fonksiyon birbirini hiç etkilemiyor demektir.

Ayrıca bilgisayar grafiklerinde kamera açısına bir şeklin izdüşümünü belirlerken tam dik olarak izdüşümünün alınması anlamına gelen orthogonal projection (dik izdüşüm) kavramında bu anlamdadır.

#### **SORU-48: Sezgi Üstü Algoritmalar (Üstsezgisel Algoritmalar, Meta Heuristic Algorithms) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde kullanılan algoritma tiplerinden birisi de sezgisel algoritmalar. Temel olarak çalışmalarında kesinlik bulunmayan bu algoritmalar ya her zaman aynı performans ile çalışmaz ya da her zaman sonuç vermeyi garanti etmez ancak yine de problemi iyileştirme (optimisation) için kullanışlı algoritmalar.

Üstsezgisel algoritmalar ise bu sezgisel algoritmalar üzerinde çalışan bir karar mekanizmasıdır. Yani bir problem için 3 farklı yöntem kullanabileceğimizi ve bu yöntemlerin hepsinin farklı açılardan avantajlı olan sezgisel algoritmalar olduğunu düşünelim. Bu sezgisel yöntemlerden hangilerinin seçileceğinin seçilmesine metaheuristic (sezgi üstü) algoritma ismi verilir.

Günümüzde pek çok alanda çözüm için birden fazla yöntem geliştirilmiş ve bu yöntemler probleme göre iyileştirilmiştir. Ancak bir problem için birden fazla olan çözüm yöntemleri arasında seçim yapma ihtiyacı bu yolla ortadan kaldırılmış olur. Basitçe bir sezgi üstü algoritma bu algoritmalar arasında seçim yapmakta ve en başarılı olanı çalıştırmaktadır.

Algoritma seçimine karar veren bu mekanizma ise çoğu zaman istatistiksel verilere dayanarak çalışmaktadır. Ancak bu algoritmanın da sezgisel olarak yazıldığı durumlar mevcuttur.

#### **Meta Heuristic ve Hyper Heuristic Ayrımı**

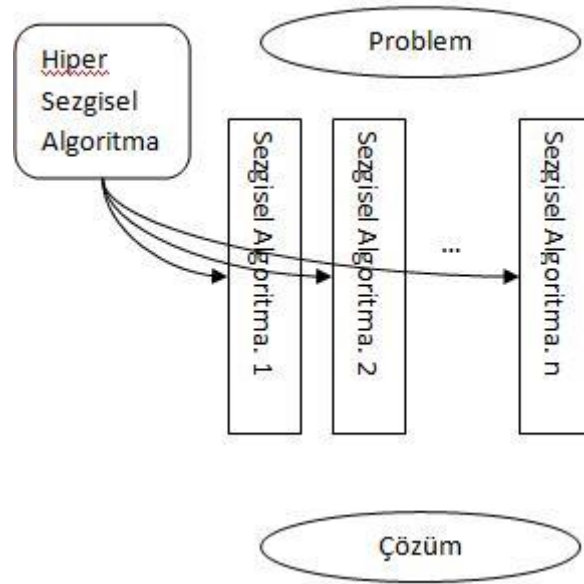
Mustafa Özgür Bey'in yorumu üzerine bu başlığı ekleme ihtiyacı duydum. Terim ve anlam olarak birbirine çok yakın olan meta heuristic (üst sezgisel algoritmalar) ve hyper heuristic (hiper sezgisel algoritmalar) arasındaki ayrım aslında bir probleme çözüm aranılan uzaya göre farklılaşmaktadır. Yani iki yaklaşımda da bir probleme sezgisel olarak (heuristic) çözüm aranmakta fakat iki yaklaşımdaki çözüm aranan uzaylar farklı olmaktadır.

Basitçe sezgi üstü algoritmalar (meta heuristic) arama işlemi problemin çözüm kümesinde (search space) yapılırken hiper sezgisel (hyper heuristic) algoritmalar arama işlemi sezgisel uzayda (heuristic search space) yapılır.

Bu anlamda hiper sezgisel bir yaklaşımda problemin çözümünden daha çok hangi sezgisel algoritma ile çözüme daha verimli ulaşılacağına karar verilmeye çalışılır. Yani bir problemin birden fazla sezgisel çözümü varsa bunlardan hangisinin daha başarılı olacağına karar verilmesine hiper sezgisel (hyper heuristic) ismi verilir.

Buna karşılık sezgi üstü algoritmalar (meta heuristic algorithms) probleme getirilen sezgisel yaklaşımı bir kara kutu (black box) olarak kabul eder ve çözüm için geliştirilen bu algoritmanın detayıyla ilgilenmez. Tek yaptığı çözümde kullanılan bir yada daha fazla fonksiyonu en verimli şekilde getirmeye çalışmaktır. Bu fonksiyonlara hedef fonksiyon (goal function , objective function) ismi verilir.

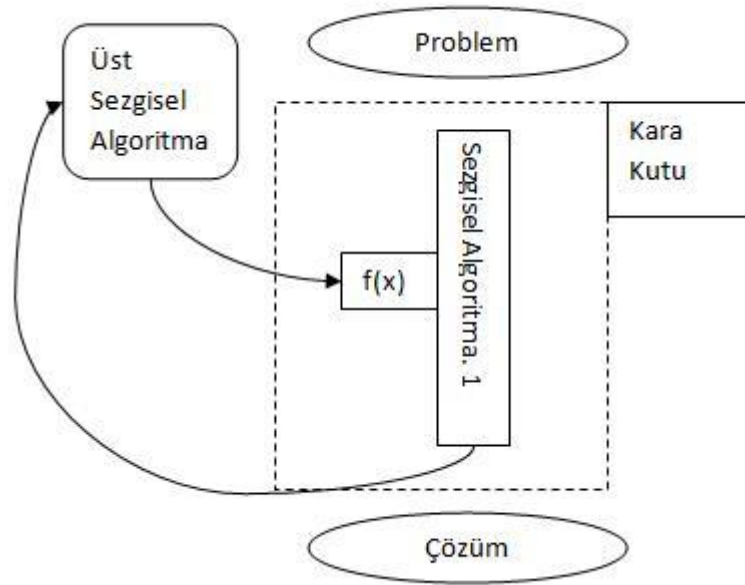
Bu ayrımı bir şekil ile ifade edecek olursak hiper sezgisel yaklaşım için:



Yukarıdaki şekilde görüldüğü üzere aynı problemde çözüme ulaşmak için kullanılan çok sayıdaki sezgisel algoritmadan hangisinin seçileceğine hiper sezgisel algoritma karar vermektedir. Benzer şekilde birden fazla sezgisel algoritmanın arka arkaya uygulanması gibi durumlarda da karar veren algoritmaya hiper sezgisel algoritma ismi verilir.

Üst sezgisel algoritma ise:





Yukarıdaki şekil çizilebilir. Şekilde görüldüğü üzere üst sezgisel algoritmaya konu olan sezgisel algoritma bir tanedir. Üst sezgisel algoritma bu sezgisel algoritmanın üzerinde çalışarak bu algoritmayı verimli hale getirmeye (optimisation) çalışır. Bunun için de genellikle algoritmanın bir fonksiyonunu (ki bu fonksiyona hedef fonksiyon (goal function, objective function) ismi verilir) iyileştirmeye çalışır.

Bu anlamda üst sezgisel algoritmanın konusu sanki giriş ve çıkış değerlerine göre bir kara kutuya (black box) benzetilebilir.

### Meta-Hyper Heuristic ( Üst Hiper Sezgisel) algoritmalar

Yukarıdaki farkı araştırırken karşılaştığım bir yaklaşımda hem üst sezgisel hem de hiper sezgisel yaklaşımların birleştirilmiş olduğu çözüm modelleri oldu. Bu yaklaşımda hem kullanılan sezgisel algoritmaların verimliliştirilmesi için hedef fonksiyonlara ( goal function , objective function) müdahale edilmekte hem de bu algoritmalar arasından seçim yapılmaktadır.

### **SORU-49: En kötü durum analizi (Worst Case Analysis) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde bir algoritmanın incelenmesi sırasında sıkça kullanılan bu terim çalışmakta olan algoritmanın en kötü ihtimalle ne kadar başarılı olacağını incelemeye yarar.

Bilindiği üzere bilgisayar bilimlerinde yargılamalar kesin ve net olmak zorundadır. Tahmini ve belirsiz karar verilmesi istenmeyen bir durumdur. Bir algoritmanın ne kadar başarılı olacağını belirlenmesi de bu kararların daha kesin olmasını sağlar. Algoritmanın başarısını ise çalıştığı en iyi duruma göre ölçmek yanıltıcı olabilir çünkü her zaman en iyi durumla karşılaşılmaz.

Algoritma analizinde kullanılan en önemli iki ölçü hafıza ve zaman kavramlarıdır. Yani bir algoritmanın ne kadar hızlı çalıştığı ve çalışırken ne kadar hafıza ihtiyacı olduğu, bu algoritmanın performansını belirleyen iki farklı boyuttur.



En iyi algoritma en hızlı ve en az hafıza ihtiyacı ile çalışan algoritmadır. İşte en kötü durum analizi olayın bu iki boyutu için de kullanılabilir. Yani en kötü durumdaki hafıza ihtiyacı ve en kötü durumdaki hızı şeklinde algoritma analiz edilebilir.

Limit teorisindeki master teoremden büyük O ile gösterilen (big-oh) değer de bu en kötü durumu analiz etmektedir. Bu yüzden en kötü durum analizine, büyük O gösterimi (Big-O notation) veya algoritmanın sonsuza giderken nasıl değiştiğini anlatmak amacıyla büyüme oranı (growth rate) isimleri verilmektedir.

## Örnek

Bir çok terimli fonksiyonun (polynomial function) big-o değerini hesaplamaya çalışalım. Örnek olarak fonksiyonumuz aşağıdaki şekilde olsun:

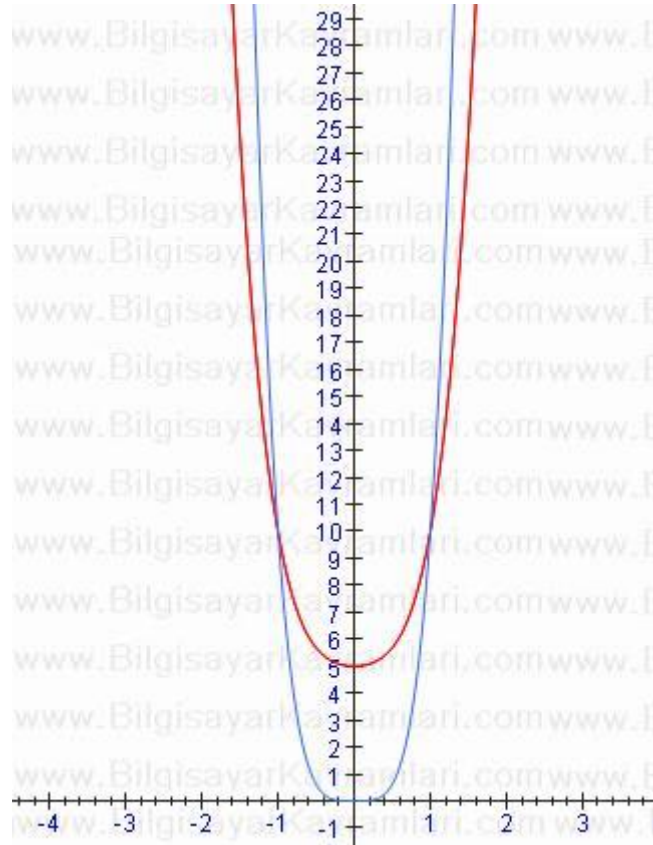
$$f(x) = 3x^4 + 2x^2 + 5$$

Fonksiyonun üst asimtotik sınırı (asymptotic upper bound), her zaman için fonksiyona eşit veya daha yüksek değer veren ikinci bir fonksiyondur.

Bu durumda, yukarıdaki  $f(x)$  fonksiyonu için  $O(x^4)$  denilmesinin anlamlı, herhangi bir sayı ile  $x^4$  değerinin çarpımının,  $f(x)$  fonksiyonuna eşit veya daha yüksek üreteceğidir. Bu durum aşağıdaki şekilde gösterilebilir:

$$f(x) \leq cx^4$$

Gerçekten de bu değer denenirse,  $c=4$  için aşağıdaki çizim elde edilebilir:



Yukarıdaki mavi renkte görülen fonksiyon  $4x^4$  ve kırmızı renkte görülen fonksiyon da  $f(x)$  fonksiyonudur. Görüldüğü üzere  $x>1$  için  $4x^4$  fonksiyonu,  $f(x)$  fonksiyonundan büyüktür. Dolayısıyla tanımımıza  $x>1$  koşulu eklenebilir.

Kısaca  $f(x) = O(g(x))$

tanımı,  $f(x) \leq c.g(x)$

şeklinde yorumlanabilir. Buradaki  $c$  değeri herhangi sabit bir sayıdır ve sonuçta elde edilen değer eşit veya daha büyüktür.

Büyük-O (Big-O) gösterimi ile kardeş olan bir gösterim de küçük-o (small-o) gösterimidir. Bu iki gösterim arasındaki temel fark küçük-o gösteriminde asimptotik üst sınır (asymptotic upper bound) fonksiyonunun tamamıncan büyük olmasıdır. Yani büyük-O gösterimindeki eşitlik durumu yoktur.

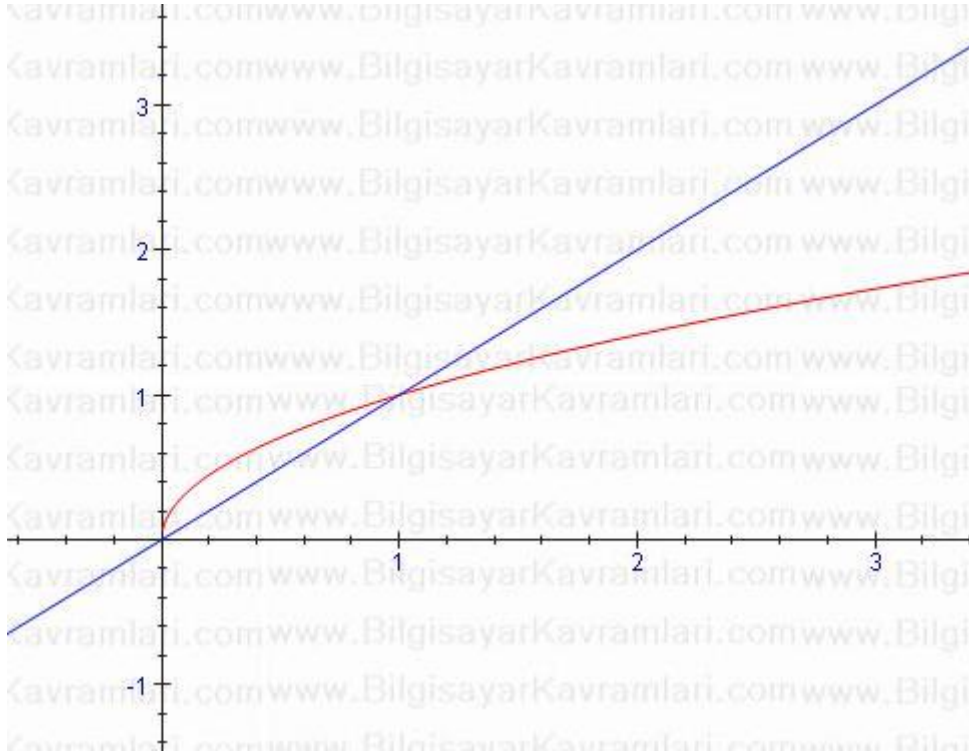
$f(x) = o(g(x))$  için

$f(x) < c g(x)$  şartı sağlanmalıdır.

Dolayısıyla  $f(x) = O(f(x))$  tanımı doğru olurken  $f(x)=o(f(x))$  tanımı hatalıdır.

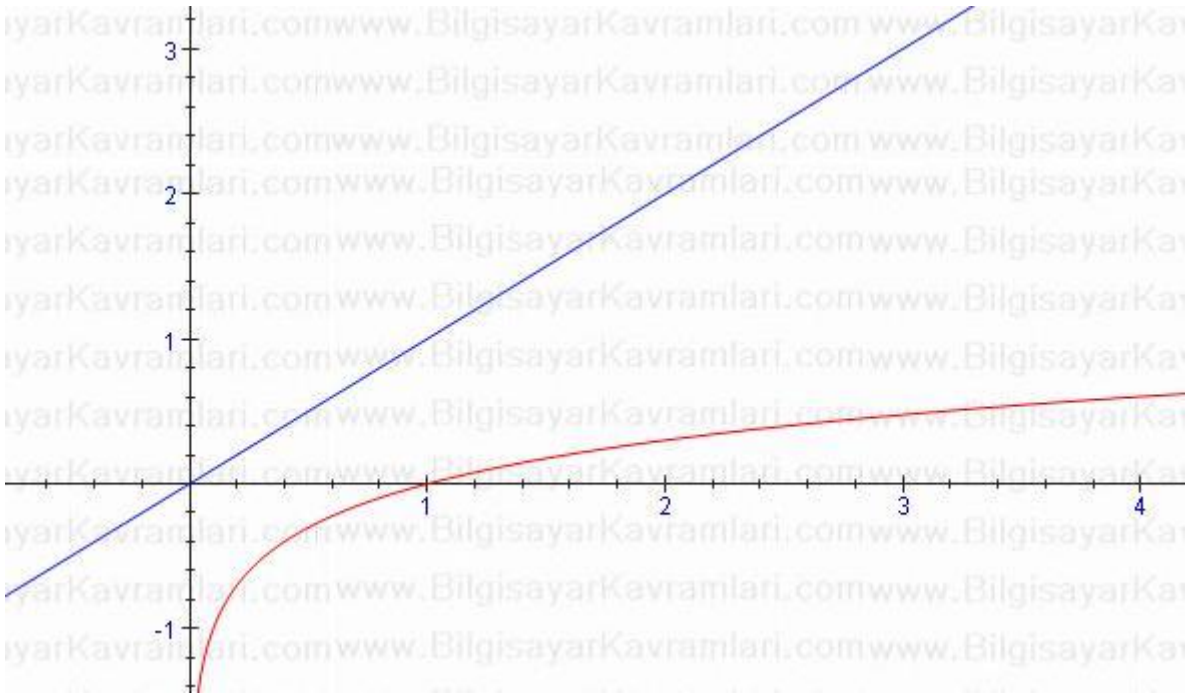
Bazı örnekler aşağıda verilmiştir:

$\sqrt{x}=o(x)$



Yukarıda görüldüğü üzere  $x>1$  için  $\sqrt{x}=o(x)$  doğrudur. Ancak  $x\geq 1$  durumunda  $\sqrt{x}=O(x)$  yazılmalıdır.

$$\log(x) = o(x)$$



Yukarıdaki şekillerde kırmızı ile gösterilen  $f(x)$  fonksiyonlarının small-o fonksiyonları mavi renk ile çizilmiştir.

#### **SORU-50: Sezgisel Algoritmalar (Buluşsal Algoritmalar, Heuristic Algorithms) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde sezgisel (heuristics) bir yaklaşımın problem çözümüne uygulandığı algoritmalar. Uygulanan yöntemin doğruluğunun ispat edilmesi gerekmez, tek istenen karmaşık bir problemi daha basit hale getirmesi veya algoritmanın tatmin edici bir sonuç bulabilmesidir.

Genel olarak bir problemin çözümü sırasında bilgisayar bilimlerinde iki amaçtan birisi güdülür. Ya problemin çözümü hızlı olmalı ve her zaman için bu çözüm elde edilebilmelidir. Bu sebepten dolayı en kötü durum (worst case) analizi sıkça yapılmaktadır.

Sezgisel algoritmalarda bu güdülen iki ihtimalden birisi göz ardı edilir. Yani ya probleme hızlı bir çözüm üretilir ama problemi her zaman çözeceği garanti edilemez ya da problemi makul bir zamanda çözer ama her zaman aynı hızda çözüleceği garanti edilmez.

Sezgisel algoritmalar gerçek hayatta hergün kullandığımız yaklaşımlardır. Örneğin bir yerden başka bir yere giderken yön duygumuza dayanarak ve yolun bizi nereye çıkaracağını hiç bilmeden hareket etmek ve yol ayrımlarında sezgisel olarak seçim yapmak böyle bir yaklaşımdır.

#### **SORU-51: Bayes Ağları (Bayesian Network) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde veri modelleme ve durum geçişi ifade etmek için kullanılan yöntemlerden birisidir. Literatürde bayes network veya belief network (inanç ağı) olarak da

geçen ağların özelliğ istatistiksel ağlar olmaları ve düğümler (nodes) arası geçiş yapan kolların (edges) istatistiksel kararlara göre seçilmesidir.

Bayes ağları yönlü dönüşsüz ağlardır (directed acyclic network) ve her düğüm ayrı bir değişkeni ifade eder. Ayrıca bu değişkenler (rastgele değişkenler, random variables) arasındaki sıralama da bayes ağları ile gösterilebilir (basitçe bir düğümden diğer düğüme geçiş sırası).

Bayes ağlarının daha geniş hali de belirsiz karar ağaçlarıdır (uncertain decision trees).

Bayes ağlarını anlayabilmek için Bayes teoremini hatırlamakta yarar vardır.  
**Bayes Teoremi :**

$$P(A_i|B) = \frac{P(BA_i)}{P(B)} = \frac{\text{ilk denklemdaki değer yerine yazılırsa}}{\text{ikinci denklemdaki değer yerine yazılırsa}} \\ = \frac{P(B|A_i) \cdot P(A_i)}{P(B|A_1) + \dots + P(B|A_n)}$$

bu  
eşitlik bayes theorem'idir.

**Örnek** Yay üreten 3 Makine için aşağıdaki hata ve üretim oranları veriliyor:

	Bozuk Yay oranı	İmalattaki oranı
I. Makine	%2	%35
II. Makine	%1	%25
III. Makine	%3	%40

Üretilen yaylardan rasgele birisi  
seçildiğinde bozuk olma olasılığı:  
 $P(B) = P(I)P(B|I) + P(II)P(B|II) + P(III)P(B|III),$

$P(B) = \frac{35}{100} \cdot \frac{2}{100} + \frac{25}{100} \cdot \frac{1}{100} + \frac{40}{100} \cdot \frac{3}{100} = \frac{215}{10000}$  olarak  
hesaplanır.

Eğer yay bozuk çıktıysa bunun 3.  
makineden çıkma olasılığı aşağıdaki şekilde hesaplanır:

$$P(III|B) = \frac{P(III)P(B|III)}{P(B)} = \frac{\frac{40}{100} \cdot \frac{3}{100}}{\frac{215}{10000}} = \frac{120}{215}$$

**Örnek:**

Bütün nüfusta yapılan bir  
hastalık testinde (+) pozitif sonucun doğru bilinme olasılığı 0.99 ve

(-) negatif sonucun doğru bilinme olasılığı 0.95 olarak veriliyor.  
Hastalığın bütün nüfusta görülme olasılığı ise  
0.0001 ise test sonucu pozitif çıkan bir kişinin hasta olma  
olasılığı nedir?

D olayı kişinin hasta olma olasılığı  
olsun

D' olayı kişinin hasta olmama  
olasılığını göstermiş olur.

Buna göre  $P(D) = 0.0001$  olur

S olayı ise testin (+) çıkma  
olasılığı olsun.

S' olayı testin (-) çıkma  
olasılığını göstermiş olur.

Buna göre  $P(S'|D') = 0.95$  olduğuna  
göre  $P(S|D') = 0.05$  olur.

Soruda istenen durum  $P(D|S)$  ile  
gösterilebilir ve şartlı ihtimal hesabından

$$P(D|S) = \frac{P(D \cap S)}{P(S)} \text{ olduğunu biliyoruz}$$

ancak

burada

$P(D \cap S)$  bilgisi bilinmemektedir Dolayısıyla  
bayes teoremini uygulamayı deneyebiliriz. Buna göre

$$P(D|S) = \frac{P(S|D) \cdot P(D)}{P(S|D_1)P(D_1) + \dots + P(S|D_n) \cdot P(D_n)}$$

olarak

bulunur. S olayı için iki ihtimal bulunduğundan ( test sonucu  
ya (+) ya da (-) çıkabilir) olasılık modeli aşağıdaki şekilde  
olur:

$$P(D|S) = \frac{P(S|D) \cdot P(D)}{P(S|D) \cdot P(D) + P(S|D') \cdot P(D')} \text{ olur.}$$

Sonuç ise

$$\frac{0.99 \times 0.0001}{0.99 \times 0.0001 + 0.05 \times (1 - 0.0001)} = \frac{1}{506}$$

olarak bulunur.

## **SORU-52: Birliktelik, Münasebet ve Oluşum (Association, Aggregation and Composition) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde sistem modelleme ve mimari tasarım sırasında sıkça kullanılan ve genelde birbirine karıştırılması kolay olan konulardır. Özellikle nesne yönelimli programlama (object oriented programming) konusundaki gelişmelerle birlikte kullanılan UML modellerinde nesneler ve sınıflar arası ilişkilerde sıkça rastlanmaktadır.

Temel olarak birliktelik ve münasebet ilişki türleri birer oluşum çeşidi olarak düşünülebilir. Yani oluşum (composition) elde etmek için verilerimiz veya nesnelerimiz arasında kurabileceğimiz ilişki türleri birliktelik (association) veya münasebet (aggregation) olabilir.

Bu iki ilişki türünün arasındaki en temel fark ise birisinin verileri kendi içinde barındırması diğerinin ise atıfta bulunmasıdır.

Örneğin münasebet (aggregation) ilişkisinde bir nesneden diğer nesneye atıfta bulunulur. Bu durum bir ders ile öğrenci arasındaki ilişkiye benzer. Dersin bitmesi ve ders nesnesinin (object) yok olması durumunda bu dersi alan öğrencilerin varlığı devam eder.

Tersi bir örnek bir ev ile odaları arasında kurulabilir. Yani nasıl bir önceki örnekte dersi alan öğrenciler varsa (dolayısıyla dersin öğrencileri varsa) ev'in de odaları vardır. Ev nesnesinden odalara atıfta bulunan bir değişken benzer şekilde tanımlanır. Ancak bu sefer farklı olan ev nesnesinde bu objelerin yaşıyor olmasıdır ve ev yok olursa oda objelerinin de yok olacağıdır.

Bu durum C++ gibi dillerde gösterici (pointer) kullanılarak çözülebilir. Yani ilk örnekteki münasebet ilişkisi için ders nesnesinden öğrenci nesnelere birer gösterici tanımlanabilir.

Birliktelik ilişkisi içinse ev nesnesinin içerisinde odalar tanımlanmalıdır.

## **SORU-53: Linear Programming (Doğrusal Programlama) hakkında bilgi veriniz.**

Problem çözümünde ve iyileştirmelerde (optimization) kullanılan yaklaşımlardan birisidir. Buradaki amaç bir problemi teşkil eden parametrelerin doğrusal bir formda olması ve problem uzayını doğrusal olarak alanlara bölmesidir.

Doğrusal bir fonksiyon aşağıdaki şekilde yazılabilir:

$$f(x_1, x_2, x_3, \dots, x_n) = c_1x_1 + c_2x_2 + c_3x_3 + \dots + c_nx_n$$

Yukarıdaki fonksiyonun doğrusal olmasının sebebi birinci dereceden olmasıdır. Aynı zamanda bu fonksiyon çok değişkenli (multi variable) bir fonksiyondur. (n adet farklı değişkeni bulunmaktadır ve fonksiyonda x değişkeni ile gösterilmiştir)

Yukarıdaki gösterimi matris haline çevirerek aşağıdaki şekilde de göstermek mümkündür:

$$c^T x \text{ (matris çarpımı olduğu için çarpanları veren c matrisinin transpoz'u alınmıştır)}$$

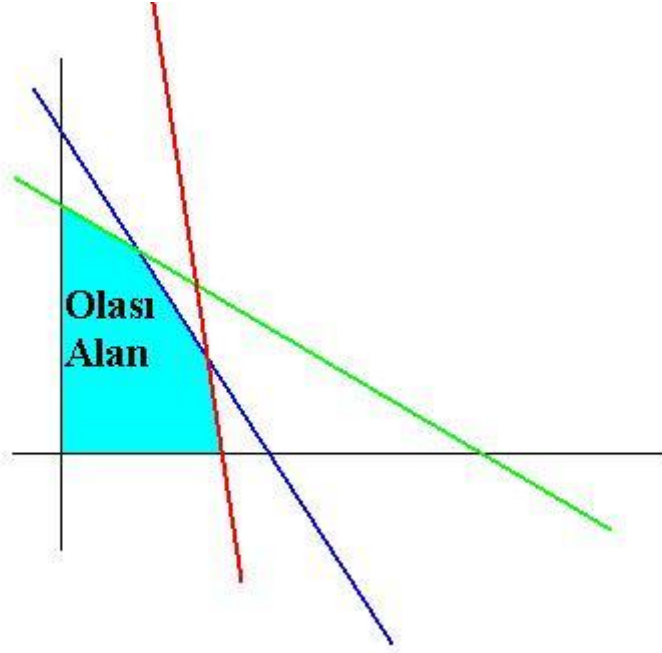
Amacımız bu ifadeyi azami hale getirmektir (maximisation)

Ayrıca bu iyileştirmeyi (optimization) engelleyen durumları (koşulları) da aşağıdaki şekilde ifade etmek mümkündür:

$$Ax \leq b$$

Bu ifadedeki A matris çarpanları olurken b ise bir vektörel değerdir.

Sonuçta bu koşulların her birisi uzayda bir doğru ifade edecek ve bu doğrular ulaşılabilir sınırları belirleyerek bu sınırlar içerisindeki alanda en iyi noktayı (optimum point) almak isteyeceğiz.



yukarıdaki şekilde 3 ayrı doğrusal denklem ve koordinat eksenleri arasında kalan ve problemimizin çözümü için olası alanı gösteren grafik verilmiştir.

Buna göre denklemimizdeki x değişkenleri 0'dan büyük olmak ve verilen doğrusal denklemlerden küçük olmak zorundadır.

Çözümü bulmak için geliştirilen algoritmalarından en tanınanı simplex algoritması ismi verilen algoritmadır.

#### **SORU-54: Entropi (Entropy, Dağılım, Dağıntı) hakkında bilgi veriniz.**

Bir sistemin düzensizliğini ifade eden terimdir. Örneğin entropi terimini bir yazı tura atma işleminde 1 bitlik (ikil) ve %50 ihtimallik bir değer olarak görebiliriz. Burada paranın adil olduğunu ve yazı tura işleminin dengeli bir şekilde gerçekleştiğini düşünüyoruz. Şayet para hileli ise o zaman sistemin entropisi (üretilen sayıların entropisi) %50'den daha düşüktür. Çünkü daha az düzensizdir. Yani hileli olan tarafa doğru daha düzenli sonuç üretir. Örnek olarak sürekli tura gelen bir paranın ürettiği sayıların entropisi 0'dır (sıfırdır).

Entropi terimi ilk kez shannon tarafından bilgisayar bilimlerinde veri iletişimde kullanılmıştır. Dolayısıyla literatürde Shannon Entropisi (Shannon's Entropy) olarak da geçen kavrama göre bir mesajı kodlamak için gereken en kısa ihtimallerin ortalama değeri alfabede bulunan sembollerin logaritmasının entropiye bölümüdür. Yani kabaca alfabemizde 256 karakter varsa bu sayının logaritmasını (  $\log 256 = 8$ 'dir) mesajın entropisine böleriz. Yani mesajdaki değişim ne kadar fazla ise o kadar fazla kodlamaya ihtiyacımız vardır. Diğer bir deyişle alfabemiz 256 karakterse ama biz sadece tek karakter yolluyorsak o zaman entropi 0 olduğundan  $0/256 = 0$  farklı kodlamaya (0 bite) ihtiyacımız vardır. Veya benzer olarak her harften aynı sıklıkta yolluyorsak bu durumda  $256/8 = 8$  bitlik kodlamaya ihtiyaç duyulur.

Bilgisayar bilimleri açısından daha kesin bir tanım yapmak gerekirse elimizdeki veriyi kaç bit ile (ikil) kodlayabileceğimize entropi ismi verilir. Örneğin bir yılda bulunan ayları kodlamak için kaç ikile ihtiyacımız olduğu ayların dağılımıdır.

Toplam 12 ay vardır ve bu ayları 4 ikil ile kodlayabiliriz:

0000 Ocak

0001 Şubat

0010 Mart

0011 Nisan

0100 Mayıs

0101 Haziran

0110 Temmuz

0111 Ağustos

1000 Eylül

1001 Ekim

1010 Kasım

1011 Aralık

Görüldüğü üzere her ay için farklı bir bilgi girilmiş ve girilen 12 ay için 4 bit yeterli olmuştur. Dolayısıyla yılın aylarının entropisi 4'tür.

Genellikle bir bilginin entropisi hesaplanırken  $\log_2 n$  formülü kullanılır. Burada n birbirinden farklı ihtimal sayısını belirler. Örneğin yılın aylarında bu sayı 12'dir ve  $\log_2 12 = 3.58$  olmaktadır. 0.58 gibi bir bit olamayacağı için yani bilgisayar kesikli matematik (discrete math) kullandığı için 4 bit gerektiğini söyleyebiliriz.

Farklı bir örnek olarak veri tabanında bulunan kişilerin cinsiyetinin tutulacağı alan 1 bitlik olacaktır. Çünkü kadın/erkek alternatifleri tek bit ile tutulabilir:



0 Kadın

1 Erkek

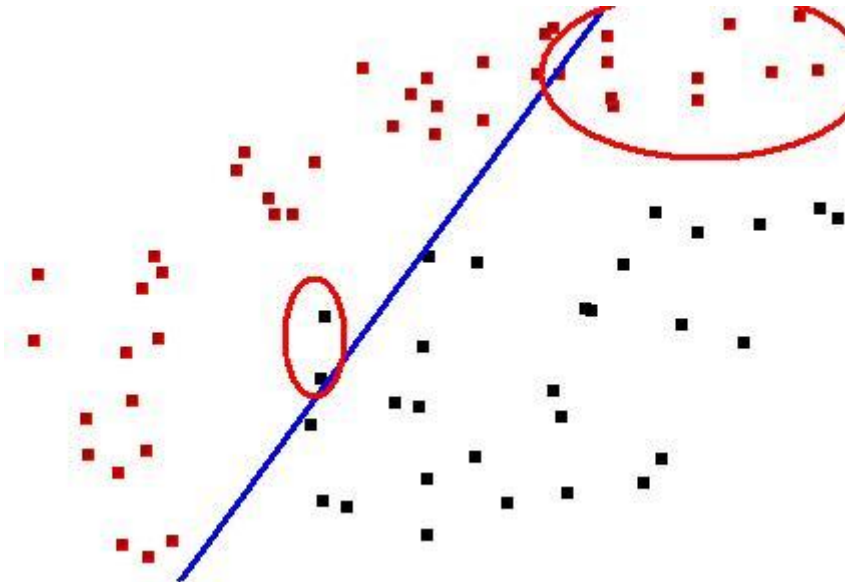
şeklinde. dolayısıyla cinsiyet alanının entropisi 1'dir.

Yukarıdaki örnekte veritabanında 5 karakterlik bir dizgi (string) alanı tutmak gereksizdir. Çünkü entropi bilgisi bize 1 bitin yeterli olduğunu söyler. 5 karakterlik bilgi (ascii tablosunun kullanıldığı düşünülürse)  $5 \times 8 = 40$  bitlik alan demektir ve 1 bite göre 40 misli fazla gereksiz demektir.

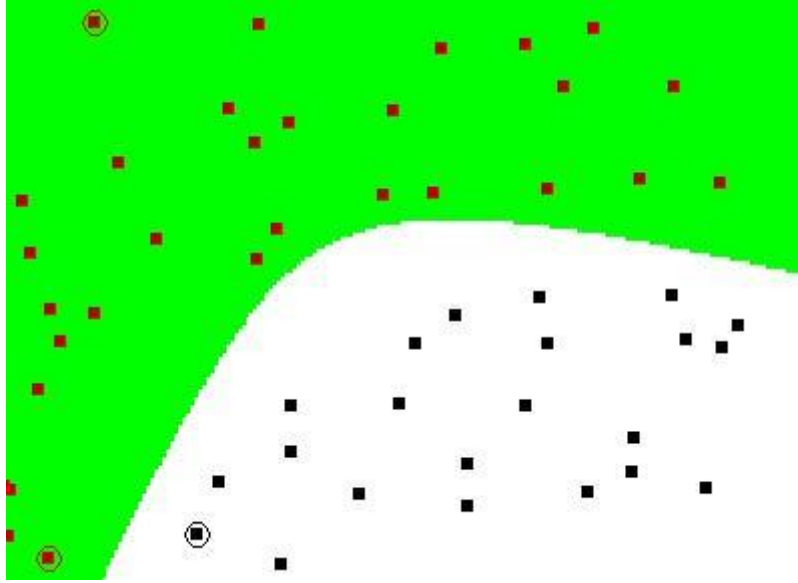
Entropi terimi veri güvenliğinde genelde belirsizlik (uncertainty) terimi ile birlikte kullanılır. Belirsizlik bir mesajda farklılığı oluşturan ve saldırgan kişi açısından belirsiz olan durumdur. Örneğin bir önceki örnekteki gibi veri tabanında Kadın ve Erkek bilgilerini yazı olarak tuttuğumuzu düşünelim. Şifreli mesajımız da “fjass” olsun. Saldırgan kişi bu mesajdan tek bir biti bulursa tutulan bilgiye ulaşabilir. Örneğin 3. bitin karşılığının k olduğunu bulursa verinin erkek olduğunu anlayabilir. Dolayısıyla bu örnekte belirsizliğimiz 1 bittir.

#### **SORU-55: Doğrusal olmayan DVM ( Non-linear SVM) hakkında bilgi veriniz.**

Destekçi vektör makinelerinde (Support vector machines) ayrım bir aşırıdüzlem (hyperplane) üzerinde iki grubu ayırdan bir doğru ve bu doğruya bağlı bir tolerans (offset) ile yapılmaktadır. Ancak bu durum her zaman beklendiği kadar iyi sonuç vermeyebilir. Örneğin aşağıdaki şekilde doğrusal bir ayrım yapılması durumunda yanlış sınıflandırma yapılan örnekler (misclassification) biraz daha dikkatli ve doğrusal olmayan ikinci şekildeki gibi ayrılırlarsa tam olarak sınıflandırılabilirlerdir.



Yukarıdaki şekilde verilen örnekler aşağıdaki şekilde doğrusal olmayan bir yolla sınıflandırılabilirler:



Bu sınıflandırmanın yapılabilmesi için çekirdek hilesi (kernel trick) adı verilen bir yöntem kullanılabilir.

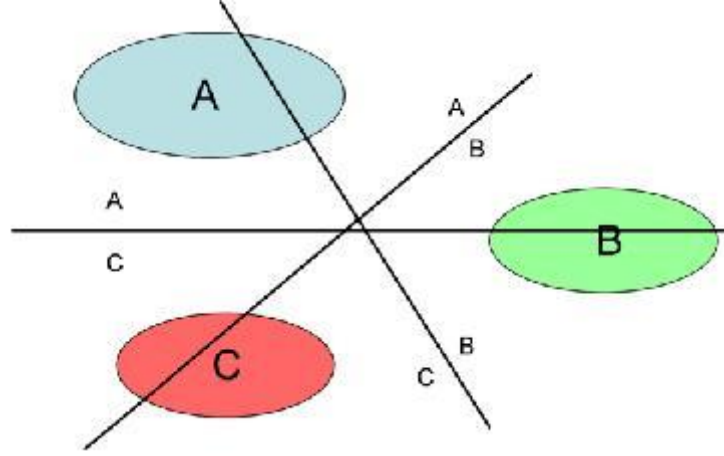
#### **SORU-56: Çok sınıflı DVM ( Multiclass SVM) hakkında bilgi veriniz.**

Bilindiği üzere destekçi vektör makinaları ( Support Vector Machine ) iki sınıfın ayrılmasında kullanılmaktadır. Yani bir DVM marifetiyle iki sınıf belirli bir tolerans (offset) değerinde birbirinden ayrılmakta ve bu ayırım sonunda çıkarılmış olan öznelik vektörlerine (feature vectors) göre aşırıdüzlem (hyperplane) üzerinde iki düzlem elde edilmektedir.

Çok sınıflı DVM ile kast edilen ise ayrımı yapılacak (sınıflandırılacak) grupların ikiden fazla olması durumudur. Bu durumda aşağıdaki üç yaklaşımdan birisi tercih edilebilir:

- Problemin ikili gruplara indirgenmesi (bire bir yaklaşımı, one-to-one)
- Problemin tek gruptan bütün gruplara modellenmesi ( bire çok yaklaşım , one-to-many)
- çok sınıf sıralama DVM'leri (multiclass ranking SVM)

Bu yaklaşımlardan en çok kullanılan ilk yaklaşıma (bire bir yaklaşıma) göre örneğin bir girdinin 4 sınıftan hangisine gireceğini bulmak isteyelim. Sınıflarımız A, B, ve C olsun. Bu sınıflar öncelikle ikili gruplar halinde SVM yöntemi ile eğitilerek sınıfların birbirine göre SVM çarpanları çıkarılır. Bu ikilileri aşağıdaki şekilde gösterilmiştir:



yukarıdaki şekilde her iki sınıf için ayrı bir SVM eğitilmiştir. Yeni bir örneğin sınıflandırılması sırasında ikili olarak önce sorgulama yapılır yani AB, BC ve AC ikilileri sorgulanır ve bu sorgulardan hangisine daha çok cevap alınırsa bu sınıfa konulur. Örneğin aşağıdaki sorgular için çıkan sonuçları ele alalım:

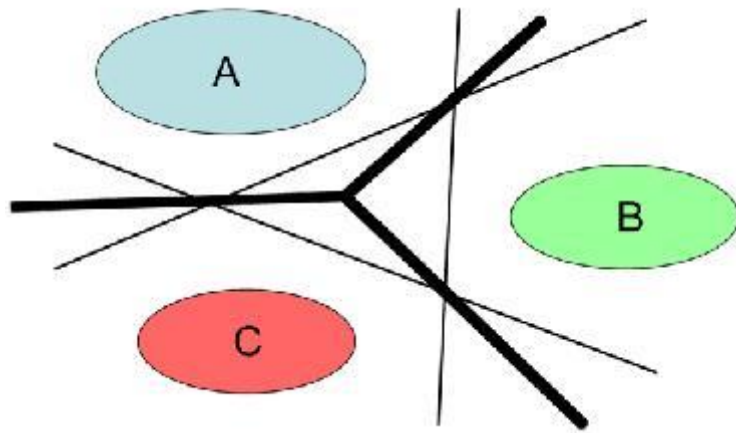
AB -> A

BC -> B

AC -> A

Yukarıdaki 3 sorgudan ikisinde A sonucuna ulaşılmıştır dolayısıyla yeni gelen örnek, A sınıfına daha yakın kabul edilmiştir.

Çoklu sınıflandırmada kullanılan bire çok yaklaşımına (one to many) göre de problem bir “kazanan hepsini alır” (winner takes all) haline dönüşmektedir. Bu yaklaşımda yeni gelen bir girdinin sınıflandırılacağı alternatiflerden birisi diğerlerine göre baskın olmakta ve sonuçta bu sınıf seçilmektedir. Ancak bu yaklaşım bir önceki yaklaşıma göre daha az tercih edilmektedir. Aşağıda bu yöntem kullanılarak üç sınıf arasında çıkarılan DVM gösterilmiştir:



Yukarıdaki şekilde de görüldüğü üzere önce 3 adet doğru denklemi bulunmuş (şekildeki ince çizgiler) sonra bu doğru denklemlerinin birleştirilmesi ile 3 sınıfı birbirinden ayırmaya yarayan genel bir DVM elde edilmiştir (şekilde kalın çizgiler)

Son olarak çoklu sınıf sıralama (multi class ranking) yaklaşımına göre sınıflar üzerinde bir DVM çalışarak bu sınıfların bütün üyelerini içeren bir sınıflandırma yapmaya çalışır. Ancak bu yaklaşım diğer iki yaklaşımdan çok daha az tercih edilir olmasında bu yaklaşımda eğitim süresinin diğerlerine göre inanılmaz oranda yüksek olması ve sonucun bulunamama ihtimali önemli rol oynar.

#### **SORU-57: Kazanan Hepsini Alır (Winner Takes All) hakkında bilgi veriniz.**

Güncel hayattan bir deyim olan kazanan hepsini alır yaklaşımına göre bir yarış içindeki iki veya daha fazla şeyden birisi diğerlerine üstünlük sağlar ve diğerlerini yok eder. Bilgisayar bilimlerinde yapay sinir ağlarında kullanılan bir terimdir.

Yapay sinir ağlarında ( Artificial neural networks) kullanılan bu terime göre, eğitim sırasında bir nöronun diğer nöronları engellemesi durumudur. Buna göre belirli bir zamandan sonra tek bir nöron aktif olacak ve diğer nöronlar bu baskın nöronun dolaylı görevini yitirecektir.

#### **SORU-58: Özellik Çıkarımı (Feature Extraction) hakkında bilgi veriniz.**

Bir sisteme giren girişlerin bütün bir bilgi olarak değil de bu bilgiyi oluşturan vasıflardan bazılarının çıkarılması ve sistemin bu vasıflar üzerine kurulması durumudur. Örneğin bir miktar resimden içinde çimen bulunanların tespit edilmesi isteniyor olsun. Bilindiği üzere çimenler yeşildir ve resimlerden yeşil tonun ağırlıkta olanlarının çimen içermesi ihtimali yüksektir. Öyleyse sisteme giren bir resmin tamamının işlenmesi yerine resmin histogramının (renk kodlarının dağılımının ) işlenmesi buna bir örnek olabilir. Resim, basitçe histogramından çok daha karmaşık ve büyük bir veridir. Bu veri histogramı (tekrar dağılımı) çıkarılmak suretiyle küçültülmüş ve istenen amaca yönelik olarak işlenmiştir.

Özellik çıkarımı (vasıflandırma, feature extraction) işlemi bir boyut azaltma (dimension reduction, dimensionality reduction) işlemidir. Buna göre karmaşık olan bir verinin boyutları azaltılarak daha basit bir problem haline indirgenir.

Doğru yapılmış bir özellik çıkarımı ve bu özelliklere uygun bir sistem tasarımı sonucun başarılı olması ve performansını etkileyen unsurlardır.

Ayrıca özellik çıkarımı sonucunda elde edilen birden fazla özelliğin karşılığını tutan veri yapısına özellikli vektörü (feature vector) adı da verilmektedir.

Resim işleme (image processing) ve yapay sinir ağlarında (artificial neural networks) sıkça kullanılan özellik çıkarımı üzerinde de çalışmalar sürmektedir. Örneğin borsa verilerinin ses sinyallerinin veya doğaya yönelik ölçümlerin her geçen gün arttığını görmekteyiz. Bu gelişmeler gelecekte daha başarılı özellik çıkarımı ve dolayısıyla daha başarılı sistemlerin kurulmasını sağlayacaktır.

#### **SORU-59: Şelale Modeli ( Waterfall Model ) hakkında bilgi veriniz.**

Yazılım mühendisliğinde kullanılan bir yazılım projesi yönetim modelidir. Bu model aşağıdaki 4 temel merhaleden oluşmaktadır:

- tahlil (analiz, analysis)
- tasım (tasarım, design)

- tatbik (uygulama, implementation)
- tecrübe (test, test)

Yazılım mühendisliğindeki diğer modellere temel teşkil eden bu modelde yukarıdaki aşamalar sırasıyla izlenir. Aşamalar arası geçişleri oldukça sıkı olan bu modelde geri dönüşler oldukça maliyetli olmaktadır.

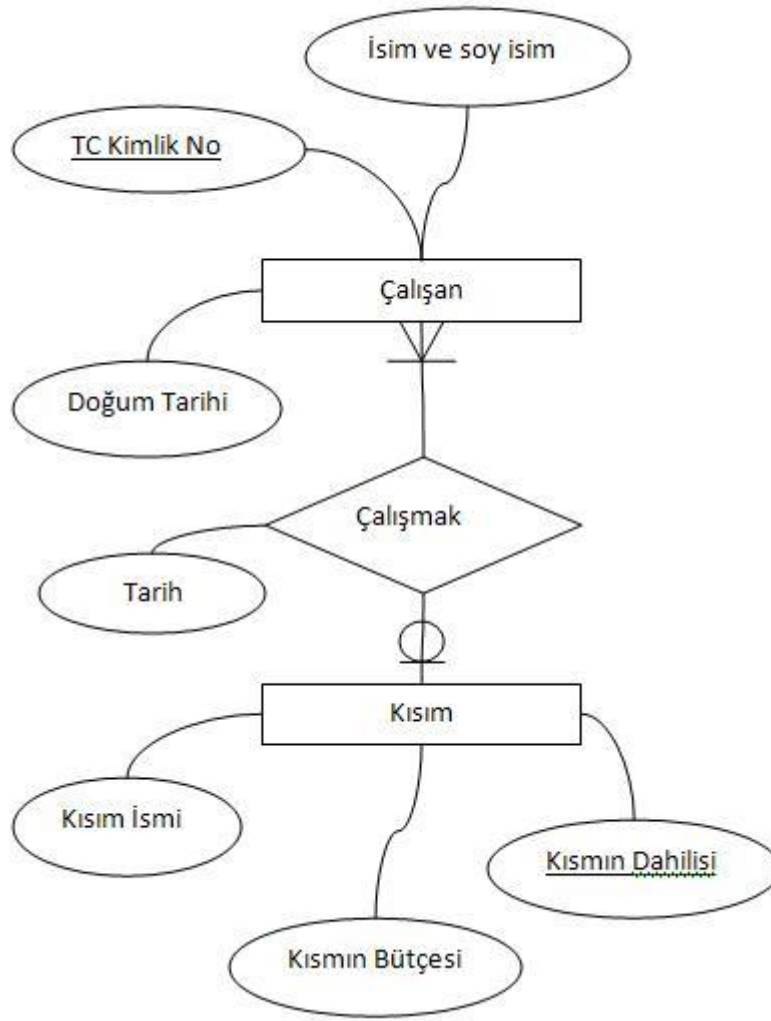
Buna göre ilk aşamada sistemin tahlili yapılır ve uygulanmak istenen yapı tam olarak ortaya konulur. Bu tahlil aşamasından sonra SRS (Software requirements specifications , yazılım ihtiyaç özellikleri) ismi verilen bir döküman ve bu dökümanla birlikte bir tahlil raporu (analiz raporu) çıkarılır.

Sonra bu tahlil ışığında bir tasımim (tasarım) yapılır ve çözüm yolları ortaya konulur. Tasımim aşaması sonunda da SDD (Software design document, tasımimname, yazılım tasarım dökümanı) adı verilen bir döküman hazırlanır.

İsteklerin ve çözümlerinin dökümantasyonunun ardından uygulamaya geçilir ve bu çözümler tatbik edilir. Bu aşamadan sonra sistem somut (müşahhas) bir hâl almış olur ve yapılan hataları daha net görme imkanı doğar. Bu hataların tam olarak ortaya konulduğu aşama ise tecrübe (test) aşamasıdır. Bu son aşamadan sonra bir test raporu çıkarılarak gerekli adımlara geri dönölür ve hatalar telafi edilerek sistem istenen hale getirilir.

#### **SORU-60: Vücubiyet (Modality) hakkında bilgi veriniz.**

Yazılım mühendisliği (software engineering) ve veritabanı tasarımı (database design) konularında sistem modellenmesi aşamasında sıkça karşılaşılan bir problem de sistemde modellenen unsurlar (entity) arasındaki ilişkinin (relationship) vücubiyetidir ( modality ) . Bu terim bir unsurun diğerini gerektirmesi anlamında kullanılmaktadır. Mesela sistemimizde bir çalışan bir de kısım (department) unsuru bulunsun. Her çalışanın bir kısmı bulunur ve kısım unsurunun vâir olması çalışana bağlıdır. Çalışanı olmazsa kısmın bir anlamı kalmaz. Buna karşılık bir kısım bulunmasa da çalışan vâir olabilir. O halde çalışan için vâicib unsur ( gerekir unsur, mandatory) , kısım için ise ihtiyarî unsur (seçimli, optional) denilebilir. Bu durum ERD çiziminde aşağıdaki şekilde gösterilir:



Yukarıda da tasvir edildiği üzere kısım unsuru ile çalışan unsuru arasında vücutiyet açısından vacib-ihiyari (mandatory-optional) ilişkisi bulunmaktadır. Yani ihtiyari olan (seçimlik olan, optional) unsur kısımdır ve bu bir O harfi ile ilişkiyi gösteren dal üzerine yerleştirilmiştir. Buna mukabil çalışanın mevcudiyeti vacib olup (gerekli olup, mandatory) bir çizgi ile bu durum gösterilmiştir.

Şekilde bulunan diğer kaz ayağı ve çizgi ise ilişkinin sayısallığını (cardinality) göstermektedir. Şekilde de görüldüğü gibi bir ERD çiziminde bu iki özellik aynı anda gösterilebilmektedir.

#### **SORU-61: Sayısallık (Cardinality) hakkında bilgi veriniz.**

Unsurlar (Entities) arasındaki sayısal bağlantıyı ifade etmek için kullanılan bir terimdir. Literatürde bazı kaynaklarda sayılabilirlik olarak da geçmektedir. Buna göre bir unsur ile diğer unsur arasında aşağıdaki üç ilişki şekline birisi olmalıdır:

- Birebir one-to-one
- Bire çok one-to-many
- Çok çok many-to-many

Bu durumlara birer misal verecek olursak:

- Bir alıřanın cep telefonu numarası ile TC kimlik no arasında birebir iliřki vardır. Yani her farklı TC kimlik no için farklı bir cep telefonu numarası söz konusudur. Bu iliřki tipi bire bir iliřkidir.
- Bir alıřan ile, alıřtığı kısım arasında bire ok iliřki vardır. Yani bir kısımda (department) birden ok alıřan alıřırken, her alıřan sadece bir kısma baėlıdır. Bu iliřki tipi de bire ok iliřkiye rnektir.
- Bir ėrenci unsuru (entity) ile hoca unsuru (entity) arasında ise oka ok iliřki tipi vardır. nk bir ėrencinin birden fazla hocası olabilirken bir hocanın da birden fazla ėrencisi bulunur.

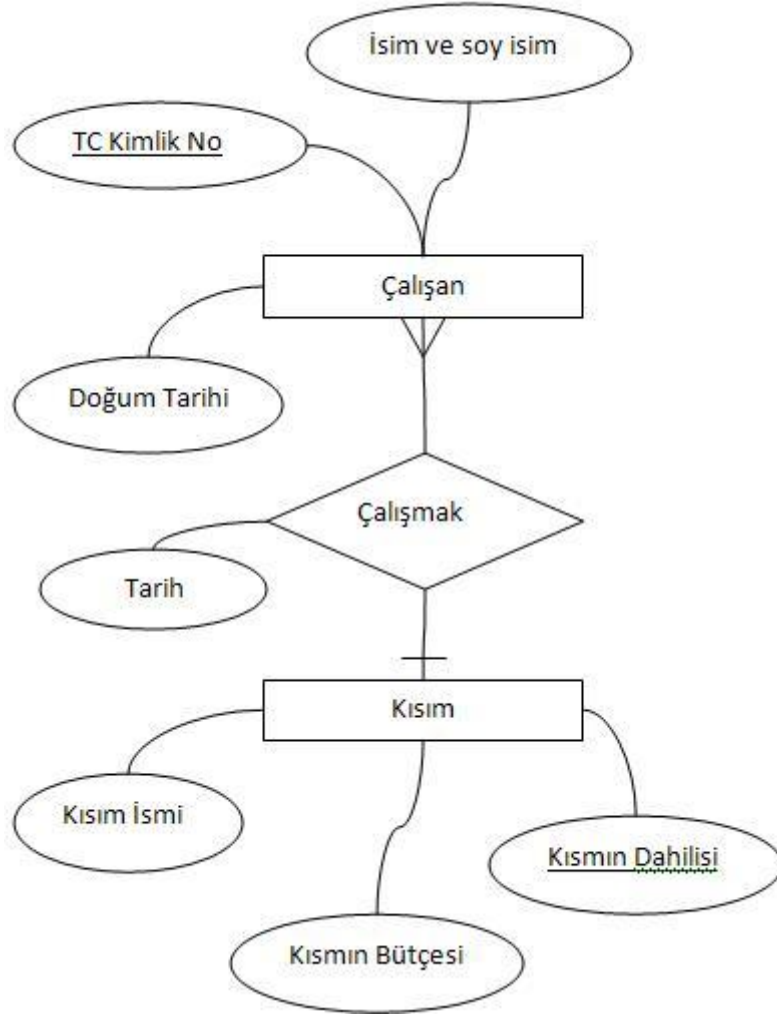
Yukarıdaki bu ikiřki tiplerinin hepsinin veritabanı teorisi için anlamları ok byktr. İliřki trlerine gre kabaca ařaėıdaki yorulmar yapılabilir. (Bunlar genel yorumlar olup istisnaları bulunmaktadır, buradaki ama okuyucuya fikir vermektir)

řayet iki unsur (entity) arasında birebir iliřki varsa bu iki unsurun aslında ayrılmalarna gerek yoktur. ok byk ihtimalle bir unsurun iki farklı paralarıdır ve tek bir atı altında birleřtirilmesi veritabanı teorisi aısından daha doėrudur.

řayet iki unsur arasında oka ok iliřki tipi varsa o halde bu iliřki tipi bire ok tipinden iki iliřkiye indirgenmelidir.

Sonuç olarak veritabanında sadece teke ok iliřki tipi elde etmek isteriz bunun sebebi yukarıda da anlatıldığı zere birebir iliřki tipinin gereksiz oluřu ve oka ok iliřki tipinin hem performans hemde hafıza olarak sistemde sorun ıkartmasıdır. Bu indirgeme konularını Normal Forms ve Composition konuları altında okuyabilirsiniz.

ERD izimleri aısından olaya bakıldığında sayısalılık ( cardinality ) kaz ayaėı veya birim olarak ifade edilir. Ařaėıda bir rnek zerinde bu durum gsterilmiřtir:



Yukarıda iki unsur arasındaki ilişkinin (relation) sayısalılığı gösterilmiştir. Buna göre bir çalışanın bir tane kısmı olabilirken bir kısımda birden çok çalışan bulunabilmektedir. Dolayısıyla ilişkinin çalışana bakan tarafı çok, kısma bakan tarafı ise tek olarak gösterilmiştir. Bu gösterimde çok olan taraf kaz ayağı, tek olan taraf ise bir çizgi ile ifade edilmektedir.

Yukarıdaki bu ilişki tipi bire çok ilişki tipine bir örnektir. Çokta çok olması durumunda iki tarafta da kaz ayağı olurken, teke tek olması durumunda iki tarafta da çizgi ile gösterilmektedir.

### **SORU-62: ERD ( Unsur İlişki Çizimi, Entity Relationship Diagram ) hakkında bilgi veriniz.**

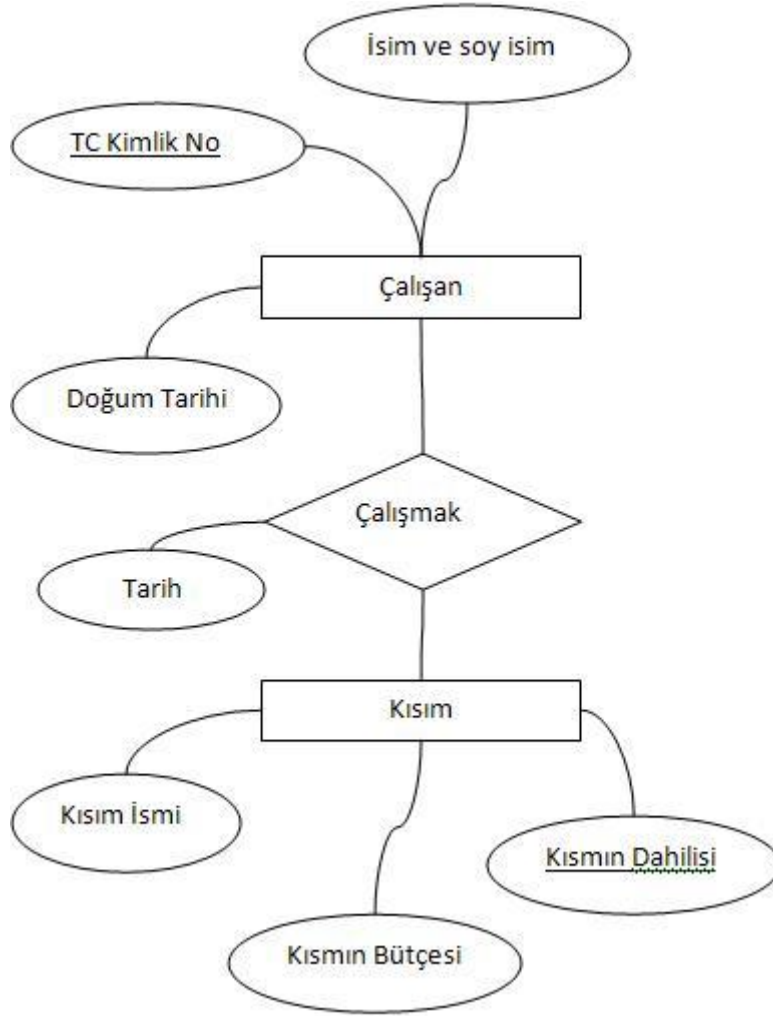
Yazılım mühendisliği (Software engineering) ve veritabanı tasarımında (database design) sıkça kullanılan bu çizim yöntemine göre, modellenmek istenen sistemdeki unsurlar (Entities) çıkarılarak bu unsurlar arasındaki ilişkiler (relationships) tanımlanır.

Unsurların özellikleri (attributes), anahtarları (keys) belirlenerek sistemin tamamını kapsayan bir model çizilir ve bu model üzerinde tasarım yapılır.

Tasarımın bitmesinin ardından, tasarımdaki bütün unsurlar ve ilişkiler birer tabloya dönüştürülerek veritabanı uygulaması tamamlanmış olur.



Aşağıda basit bir ERD çizimi bulunmaktadır:



Yukarıdaki bu ERD çizimine göre iki unsur olan kısım ve çalışan arasında bir çalışmak ilişkisi bulunmaktadır. Yukarıda her unsurun ve unsurlar arasındaki ilişkinin özellikleri de verilmiştir.

Bu modelleme yönteminde dikkat edilecek önemli bir husus ise sistemimizde bulunması gerçekten gerekli olan özelliklerin ve unsurların modellenmesidir.

ERD çiziminin üzerinde ayrıca ilişki tiplerini tutmak da mümkündür. Bu ilişki tiplerini iki çatıda incelemek mümkündür. İlişkilerin sayısallığını tutan cardinality ilişki şekli ve ilişkilerin gerekliliğini tutan modality (tazannum eden, vacib-i vucud, birşeyin varlığının diğer şeyin varlığını gerektirip gerektirmemesi) ilişki tipi bu iki çatıdır.

### **SORU-63: İlişki (Relationship) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde ilişki kavramı çok çeşitli anlamlarda kullanılabilir. Örneğin nesnelerin birbiri ile olan ilişkisi veya veritabanındaki tabloların ilişkisi gibi.

### **Veritabanı teorisindeki tablo ilişkisi**

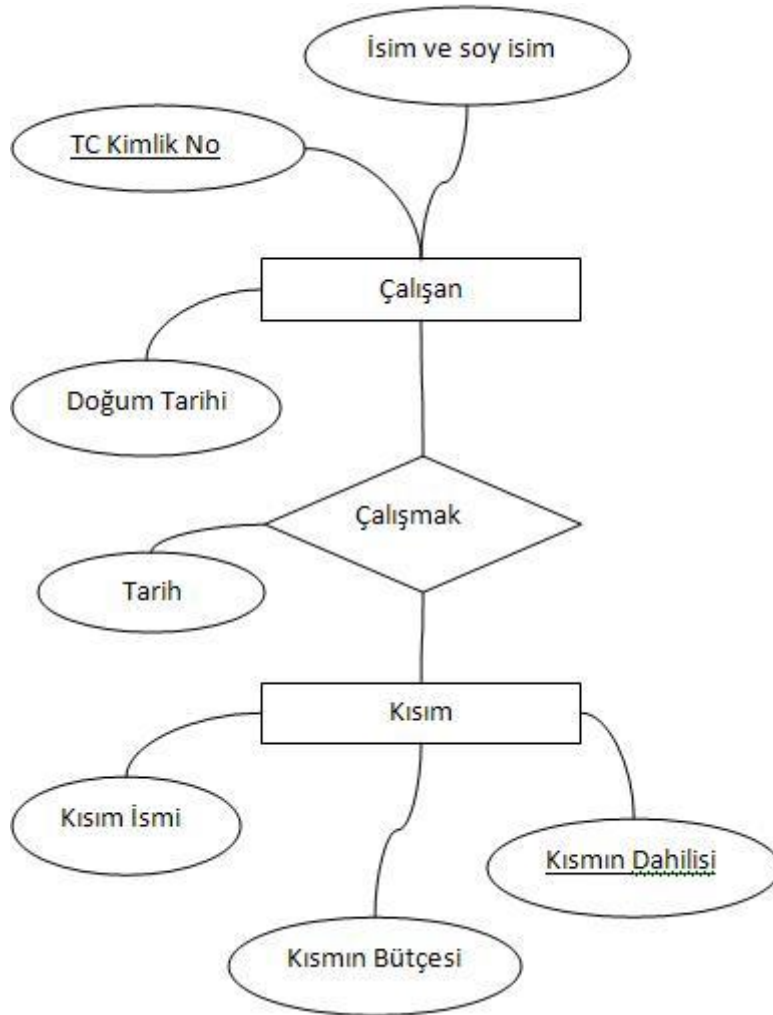
Temel olarak bir ilişkiisel veritabanını modellemekte kullanılan ERD (Unsur ilişki çizimi, entity relationship diagram) üzerinden bir ilişkiyi anlamak mümkündür. Bilindiği üzere bir ERD, unsurlardan (entities) oluşmaktadır ve ERD üzerinde bu unsurların ilişkisi modellenmektedir. Aşağıdaki cümleyi ele alalım:

“Ali muhasebe kısmında çalışıyor”

Bu cümlede “Ali” isminde bir çalışandan ve muhasebe departmanında bahsediliyor. Dolayısıyla bu cümleden çıkan anlama göre bir “Çalışan” unsuru bir de “Kısım” unsuru bulunmakta ve Ali, çalışan unsurunun, muhasebe ise departman unsurunun birer örneği olmaktadır.

Dolayısıyla bu iki unsur ( çalışan ve kısım) arasında çalışmak ilişkisi olduğunu yukarıdaki cümleden anlıyoruz.

Yukarıdaki bu cümleyi aşağıdaki model ile tasvir etmek mümkündür:



Yukarıda iki unsur arasındaki ilişki görüntülenmiştir. Buna göre ilişkinin de özellikleri olabilir (descriptive attributes). Örneğin yukarıdaki tarih özelliği çalışma ilişkisinin başladığı tarihi tutmak için tasarlanmıştır. Yani bir çalışanın bir kısımda ne kadar zamandır çalıştığı bilgisine ulaşılabilir.

### SORU-63: Unsur (Entity) hakkında bilgi veriniz.

Veritabanı tasarımı (database design) ve yazılım mühendisliğinde (software engineering) sıkça kullanılan bir tasarım yöntemi, modellenmek istenen sistemdeki unsurları çıkararak bu unsurların özelliklerini ve bu unsurlar arasındaki ilişkileri tutmaktır.

Temel olarak bir unsur nesne yönelimli programlama mantığında olan her nesneye benzetilebilir. Ancak bir unsurun bir nesneden temel farkı, ihtiyaç duyulduktan sonra hemen her şeyin birer unsur olabileceğidir. Örneğin insan sınıfının bir özelliği olarak isimi ele alalım. Bu özellik yeterli olgunlukta ise (ünvan, ilk, orta, soy isim gibi karmaşık bir yapıya sahipse) bir unsur olarak değerlendirilmek zorundadır. Ancak nesne yönelimli programlamada bir sınıf olması mantıksızdır.

- Bir unsur'un tanımını aşağıdaki şekilde yapmak mümkündür:
- Unsur (Entity) gerçek hayattandır.
- Her unsurun özellikleri (attributes) vardır
- Her özelliğin alabileceği değerler (domain) tanımlıdır
- Bir unsuru oluşturan özellikler içerisinde bu unsuru tek başına tanımlayan bir anahtar (key) seçilebilir
- Unsurlar kümelenebilir. (Çalışanlar kümesi hem sekreterleri hem de müdürleri kapsar)

Yukarıdaki şekilde tanımlanan bir unsurun ERD (Entity relationship diagram) üzerindeki gösterimi aşağıdaki şekildedir. :



yukarıda bir çalışan unsuru ve bu unsurun özellikleri gösterilmiştir. Burada bulunan özellikler bir çalışanın sahip olduğu bütün özellikler değildir. Bir unsuru sistem modellemesinde kullanırken dikkat edilecek husus bu unsurun sistemin modellenmesi için gerekli olan özelliklerinin (attributes) bulundurulmasıdır. Bunun dışındaki özellikler sistemin tasarımını etkilemeyeceği için modellemeye de alınmaz.

Yukarıdaki tasvirde dikkat edilecek bir husus da “TC Kimlik No” özelliğinin altının çizili olmasıdır. Bunun sebebi bu özelliğin tek başına bir çalışanı diğer çalışanlardan ayırt etmeye yarayan bir özellikli oluşudur. Bu tip özelliklere anahtar (key) ismi verilmektedir.

#### **SORU-64: Kıtık (Starvation) hakkında bilgi veriniz.**

Bir algoritmada sıra bekleyen işlere bir türlü sıra gelmemesi durumudur. Teorik olarak sıradaki her işe birgün sıra gelecektir ancak fiiliyatta bu bir türlü gerçekleşmeyebilir.

Bu tip problemler genelde öncelik tanımlanmış olan algoritmalarda çıkar.

Şöyle bir örnek düşünelim, elimizde uzunlukları 4,5,6 olan işler olsun ve en kısa işi tercih eden bir algoritmamız olsun (Shortest Job First, SJF) . Algoritmamız en kısa olan 4 uzunluğundaki işten başlayacaktır. Ve diyelim ki 4 bitir bitmez veya henüz bitmeden uzunluğu 3 olan başka bir iş gelsin. Algoritmamız 5 uzunluğundaki iş yerine daha kısa olan 3 uzunluğundaki işe öncelik verecektir.

Bu süreç böylece sonsuza kadar gidebilir. Yani henüz 5 ve 6 uzunluğundaki işlere sıra gelmeden hep daha kısa olan işlerin gelerek önceliği alması ve 5 ve 6 uzunluğundaki işlere hiçbir zaman sıra gelmemesi söz konusudur.

#### **SORU-65: Durma Problemi (Halting Problem) hakkında bilgi veriniz.**

Problem kısaca bir programın bir zaman sonra durup durmayacağını belirsizliği üzerine tartışmadır.

Yani basitçe elimizde bir program ve bu programın parametresi olsun (programa verilebilen bir girdi). Programın bitip bitmeyeceğini bilemeyiz.

Peki bunu nasıl ispatlarız? Burada ispat için olmayana ergi (proof by contradiction) kullanılabilir. Öncelikle problemimizi modelleyelim:

$D(P,G)$  : P, programının verilen G girdisi ile durup durmayacağı olsun.

Buna göre  $D(P,G)$  sonucunda ya durur ya da durmaz sonuçlarından birisi çıkacak.

Biliyoruz ki bir programın kendisi de bir girdidir. En azından programın kendisi bir bilgidir ve bu bilginin yazıldığı dilde bir metin olması ve bir veri şeklinde gösterilmesi mümkündür. Öyleyse  $D(P,P)$  gösterimi bir programa kendisinin girdi olarak verilmesi durumudur. Basitçe bir programı girdi alan bir program var ve bu program girdi olarak aldığı programın bitip bitmeyeceğine karar verecek şekilde düşünülebilir. Çünkü yapmak istediğimiz tam da budur. Bir programın bitip bitmediğine karar veren bir program bulmak istiyoruz, öylesyse acaba bu bulmaya çalıştığımız ve bir programın bitip bitmediğine karar veren programın kendisinin bitip bitmeyeceğine aynı program karar verebilir mi?

$D(P,P)$ : kendisinin bitip bitmeyeceğine karar vermeye çalışan programımız olsun.

Şimdi olmayana ergi (proof by contradiction) kullanılacağı için  $D(P,P)$ 'nin terisini almalıyız. Öyle bir  $T(P)$  tanımlayalım ki üzerinde çalıştığımız P programı için  $D$ 'nin tersi olsun ve  $D(P,P)$  durur dediğinde  $T(P)$  durmaz,  $D(P,P)$  durmaz dediğinde ise  $T(P)$  durur desin.

Şimdi artık çelişkimizi ortaya koyabiliriz.  $D(T,T)$  şayet sonuç olarak durmaz çıkarırsa tanımı itibarıyla biliyoruz ki  $T(T)$  durur çıkarmalıdır. Tersisi durumda  $D(T,T)$  şayet sonuç olarak durur çıkarırsa,  $T(T)$  durmaz çıkarmalıdır. Buradaki iki durumda da çelişki vardır çünkü T girdisi

$D(T,T)$  işleminde şayet durur çıkarıyorsa  $T(T)$  işleminde de durur çıkarmalıdır. (Tanımı itibarıyla  $T$  ya durur ya durmaz, birisinde durur diğerinde durmaz çıkarması bir çelişkidir). Tersini durum olan  $D(T,T)$  için durmaz çıkarıyorsa  $T(T)$  için de durmaz çıkarmalıdır. Ancak görüldüğü üzere bu sonuçların tam tersi çıkarılmıştır. Bu durumda bir çelişkidir.

Sonuç olarak  $D$  her zaman doğru sonuçları veremez. Bunun anlamı durup durmayacağına karar verilebilen bir  $D$  yazılamaz demektir.

### **Basit Anlatımı**

Çok saygı duyduğum bir hocam tarafından, burada yazılanların arasındaki bağlantıda problem olduğu söylendi . Bunun üzerine olayı çok daha basit bir şekilde anlatmaya çalışıyorum.

Durma problemi, bir programın durup durmayacağının asla bilinmemesidir.

Yani hiçbir zaman bir program yazıp, bütün programların bitip bitmeyeceğine karar veremeyiz, böyle bir program yazılamaz.

Bunu anlamamanın en kolay yolu, basit bir öz çelişkiden (self contradiction) geçer.

Örneğin bir program yazıp, programların bitip bitmeyeceğini (durup durmayacağını, halt) bulmayı hedeflediğimizi düşünelim. Ve bu programa  $T$  ismini verelim.

Dolayısıyla  $T(P)$  gibi bir yapı olacak ve  $T$  programımız,  $P$  programını parametre olarak alıp, bu programın bitip bitmeyeceğini söyleyecektir.

Şimdi öz çelişki (self contradiction) oluşturan duruma bakalım ve soralım acaba  $T(T)$  ne döndürür?

Yani programların bitip bitmeyeceğini söyleyen  $T$  programını kendisine parametre olarak verirsek ne olur?

Diğer bir deyişle, bir program yazıp, programların bitip bitmeyeceğini bulmak istiyoruz, peki bu program kendisinin bitip bitmeyeceğini söyleyebilir mi?

Elbetteki hayır.

Dolayısıyla bir programın bitip bitmeyeceğini bulan başka bir program yazılırsa, bu program en az kendisinin bitip bitmeyeceğini söyleyemeyecek ve dolayısıyla bütün programların bitip bitmeyeceğini söyleyebilecek bir program yazılamayacaktır.

### **SORU-66: Tersine Koyarak İspat (antitez, Contraposition) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde de kullanılan ispat yöntemlerinden birisi bir önermenin tersini ispatlayarak önermenin doğruluğunu göstermektir.

Doğrudan ispat yöntemlerinde  $p \rightarrow q$  zinciri kullanılmaktadır. Bu yaklaşımda ise  $\neg p \rightarrow \neg q$  yaklaşımı ile iki önermenin de tersi alınır ve birbirini izlemesi gereken önermeler oldukları gösterilir.

Basit bir örnek ile durumu inceleyelim. Mesela  $n$  gibi bir tam sayı için şayet  $3n + 2$  işleminin sonucu tek sayı ise  $n$  tek sayıdır denilebilir. Bu iddiayı tersine koyarak ispatlayalım.

Öncelikle doğrudan ispat yöntemi ile ispat yapmaya çalışalım. Biliyoruz ki  $2k+1$  gösterimi bir tek sayı gösterimidir. Şayet  $3n+2$  tek sayı ise  $3n+2 = 2k + 1$  olduğu bir  $n$  ve  $k$  ikilisi bulunabilmelidir. Bu eşitlik basitleştirilince :

$3n + 1 = 2k$  gibi bir sonuca varılır ki bu eşitlikten bir yorum ve çıkarım yapmak mümkün değildir.

Doğrudan ispat yönteminin başarısızlığı üzerine dolaylı ispat yöntemlerinden tersine koyarak ispat yöntemini deneyelim:

Öncelikle önermenin tersini alalım. Buna göre  $3n+2$  tek sayı ise  $n$  de tek sayıdır ifadesinin tersini alıp  $n$  çift sayıdır önermesi ile çıkalım. Çift sayı olmak demek  $n = 2k$  gibi bir ifadenin bulunması demektir.

$3n + 2$  denkleminde  $n$  yerine  $2k$  koyulursa  $3(2k) + 2 = 6k + 2 = 2(3k + 1)$  bu sonuca göre  $3n+2$  değeri çift olmalıdır (çünkü 2 ile çarpılan bir değer çifttir). Oysaki bu durum olamaz çünkü başlangıçta bu değer tek olduğunu kabul ederek başladık. Dolayısıyla ters durum ispatı göstermiş oldu ki  $3n+2$  tekse ve  $n$  çiftse bir çelişki olmaktadır, o halde  $3n+2$  tekse  $n$  de tektir.

### **SORU-67: Doğrudan İspat (Direct Proofing) hakkında bilgi veriniz.**

Bilgisayar bilimlerinin pekçok alanında da kullanılan ispat yöntemlerinden en basitidir. Bu yöntemde göre ispatlanmak istenen durum genelde  $p \rightarrow q$  şeklinde bir önermenin (kaziye) ispatının diğer bir önermeyi (kaziye) gerektirdiği bir dizilimdir ve birisinin ispatı diğerini gerektirir.

Örneğin:  $n$  sayısı tek sayı ise  $n^2$ 'nin de tek sayı olduğunu ispatlayalım.

Öncelikle biliyoruz ki bir sayının tek sayı olması  $2k+1$  şeklinde yazılabilmektedir. Yani  $2k+1$  formülündeki  $k$  yerine hangi tam sayı konulursa konulsun sonuçta elde edilen değer tek sayıdır.

Şimdi  $n$  tek sayı ise  $n$  için  $2k+1$  yazılabilir o halde  $n = 2k+1 \rightarrow n^2 = (2k+1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$

biliyoruz ki  $2k^2 + 2k$  bir tam sayıdır ve diyelim ki bu tam sayı  $\phi$  olsun,  $\phi = 2k^2 + 2k$  ise

$n^2 = 2(k^2 + 2k) + 1$  değeri yerine  $2(\phi) + 1$  yazabiliriz. Buradaki  $\phi$  bir tam sayı olduğuna göre elde edilen  $n^2$  değerinin tam sayı olduğu gösterilmiş olur çünkü  $2\phi + 1$  her zaman tam sayı verir.

### **Örnekler (İbrahim beyin talebi üzerine ekliyorum)**

Yorum kısmında İbrahim bey örnek istemişler bunun üzerine bir iki klasik ve literatürde sık rastlanan örneği burada açıklıyorum. Öncelikle biraz daha kolay anlaşılabilir günlük örneklerle başlayalım. Aristo mantığının da temeli olan basit önermeleri ele alalım.

Örneğin “her sabah güneş doğar” gibi bir kaziye (önermeyi) ele alarak başlayalım. Bir kişi saatine bakıp 6.00 olduğunu görüyorsa. Saat 6.00 şayet sabah olarak kabul ediliyorsa, öyleyse güneş doğacaktır.

Şayet yukarıdaki zincirde ilk önerme yani “her sabah güneş doğar” önermesi doğruysa ve diğer zincirdeki bağlantılarda doğruysa gerçekten de güneş doğacaktır ve böylelikle güneşin doğacağı ispatlanmış olur.

Yukarıdaki örnekte de görüldüğü üzere doğrudan ispatlarda amaç  $p \rightarrow q$ , yani  $p$  doğruysa  $q$  da doğrudur şeklinde bir bağlantı bulabilmektir. Bu bağlantının ardından  $p$  ispatlanır yada doğru kabul edilir ve  $q$  bu vesileyle ispatlanmış olunur.

Farklı bir örneği ele alalım. Birinci önermemizde (kaziye) iki açının toplamının 90 derece olduğunu söyleyelim (buna  $p$  diyelim). İkinci kaziyede ise bu iki açının müttemmim (bütünleyen açılar, tümleyen açılar, complementary) olduğunu söyleyelim. Bu ikinci kaziyeye de  $q$  diyelim.

Yukarıdaki sistemde şayet ilk kaziye doğruysa ikincisinin de doğru olması gerekir. Yani doğrudan ispatın gereği olarak  $p \rightarrow q$  bağlantısı bulunur. Burada genelde ilk kaziyeye şart (condition) ikinci kaziyeye ise faraziye (hipotez, varsayım, hypothesis) ismi verilir. Yani tam anlamıyla ikinci kaziyenin doğruluğunu farz etmek için birinci kaziye şarttır.

Yine ibrahim beyin talebine dönecek ve farklı bir örnek çözecek olursak, bu sefer örnek olarak aşağıdaki ispatı ele alalım:

$r$  gibi bir sayının, bir çok terimlinin (polinom) kökü olarak kabul edilebilmesi için  $p(r) = 0$  olması gerekir.

Şimdi ispatını yapmak istediğimiz faraziyeyi ortaya atalım:

Şayet  $r_1$  ve  $r_2$ ,  $p(x) = x^2 + bx + c$  çok terimlisinin (polynom) ayrı iki köküyse,  $r_1 + r_2 = -b$  ve  $r_1 r_2 = c$  olduğu söylenebilir.

İspatı:

Öncelikle  $p(x)$  çok terimlisini çarpanlarına ayıralım ( $r_1$  ve  $r_2$  birer kök olduğuna göre)

$$p(x) = (x - r_1)(x - r_2)$$

sağ tarafta işlem yaparsak

$$p(x) = x^2 - (r_1 + r_2)x + r_1 r_2$$

olarak bulunur. Bu çok terimliyi ilk yazdığımız haldeki çok terimli ile taraf tarafa eşitlersek sonuçta ispatlamak istediğimiz durumun doğruluğu ortaya çıkar

$$x^2 - (r_1 + r_2)x + r_1 r_2 = x^2 + bx + c$$

dolayısıyla  $r_1 + r_2 = -b$  ve  $r_1 r_2 = c$  ‘dir denilebilir.

Yuakrıda görüldüğü üzere ispatın yapılabilemesi için  $r_1$  ve  $r_2$ , kök olmalıdır. Yani ikinci kaziyenin ispatı için ilki şarttır.

### **SORU-68: Bilgisayar Mühendisliği hakkında bilgi veriniz.**

Bu yazının amacı genel olarak bilgisayar mühendisliğini ve terimsel bazı problemleri açıklamaktır. Bu yazı, içinde bulunulan bilgisayararkavramlari.com sitesinin amacı olan akademik ve bilimsel açıklamaların dışına çıkmaktadır. Ancak bu yazı ile tartışılmakta olan bazı konulara açıklık getirilmek amaçlanmış ve sitenin genel okuyucu kitlesinin bilgilendirilmesi amaçlanmıştır. Her zaman olduğu gibi buradaki konularda yanlış veya eksik olduğunu düşündüğünüz konular için iletişime geçebilirsiniz.

### **Bilgisayar Mühendisliği mi / Bilimleri mi?**

Bilgisayar mühendisliği yabancı bir dilden giren hemen her kelime (ya da kelime grubu) gibi sancılı bir terimdir. Kavram olarak gerek bilgisayarları gerekse bilgisayar mühendisliğini ithal etmiş ülkemizde, ingilizcedeki “computer science” ve “computer engineering” kavramlarını karşılayan ne yazık ki tek terim bulunmaktadır. Her ne kadar Bilgisayar Bilimleri gibi sonradan bazı terimler denenmiş olsa da içerik karmaşası sayesinde bu yazının yazıldığı tarih itibarıyla Türkiye’deki “bilgisayar mühendislikleri” hala iki terimi de karşılayan eğitimler vermeye çalışmaktadır.

Bu iki bölüm (computer science / Engineering) arasındaki temel fark mühendisliğin özellikle Amerika’daki üniversitelerde Elektronik fakültesine bağlı olması ve daha çok donanım, işlemci programlama, elektronik ağırlıklı çalışmasıdır. Bilgisayar bilimleri ise daha çok algoritma analizi, dil tasarımı, yapay zeka gibi nispeten soyut (mücerret) konular üzerinde çalışmaktadır. Her ne kadar iki disiplinin de ortak noktaları olarak kabul edilebilecek bilgisayar ağları, işletim sistemleri gibi konular olsa da aslında iki disiplin birbirinden ayrılmaktadır.

Türkiyede ise bu durum tek bir isimle (Bilgisayar Mühendisliği) karşılanmakta olup bu bölümlerin içeriği ağırlıklı olarak bilgisayar bilimleri olarak şekillenmektedir. Her ne kadar çeşitli üniversitelerde Bilgisayar Bilimleri, Yazılım Mühendisliği, Bilişim Teknolojileri gibi bilgisayar mühendisliğine yakın bölümler açılmış olsa da genelde aynı üniversitelerin bilgisayar mühendisliği bölümü de bulunmakta ve genelde içeriği yukarıda da anlatıldığı üzere şekillenmektedir.

### **Bir disiplin olarak bilgisayar mühendisliği**

Her disiplinin var olabilmesi için bilimsel bazı dayanaklarının olması gerekir. Özellikle de sayısal çalışmalarda bir disiplinin varlığı, matematiksel bir dayanağa bağlıdır.

Aslında bilimlerin sınıflandırılması uzun ve tartışmalı bir konudur ancak bilgisayar mühendisliğinin bir uygulamalı bilim (applied science) olduğu konusunda uzlaşma vardır. Yani daha çok matematiksel kavram ve kuramların uygulandığı bir alan olarak görülebilir.

Bilimleri şayet doğal bilimleri (fennî ilimler) ve beşerî ilimler (insana dayalı bilimler) olarka sınıflandırırsanız, bilgisayar mühendisliğinin insana bağlı biri bilim olduğu kesindir. Bu sınıflandırmada temel rol, doğadan öğrenilen veya insan tarafından üretilen bilim olmasına göre yapıldığına göre, bilgisayar bilimlerinin büyük çoğunluğu fizik, biyoloji, kimya gibi doğadan



öğrenilen bilimlerden oluşmamakta bunun yerine felsefe, matematik gibi insan tarafından koyulan kurallar ve uygulamalardan oluşmaktadır.

Bilgisayar mühendisliğinin(bilimlerinin) en çekirdeğinde algoritma kavramının ve algoritma analizinin olduğunu söylemek doğru olur. Her ne kadar bu disiplinin temeline ayrı matematik (Discrete mathematics) oturtan yorumlar olsa da günün birinde sürekli çalışan (continuous) bir bilgisayar hakim olabilir ve bu olduğunda bu bilgisayarlar ve üzerlerindeki çalışmalar da bilgisayar mühendisliğinin konusu olacağı ve bu durumda sürekli matematikten bahsedileceği için bu ayrım çok doğru değildir.

Ancak algoritma analizi, yani bir işin nasıl yapılacağını adım adım akışının çıkarılması ve bu adımların ve akışın iyileştirilmesi (optimize) her zaman ve her problem için geçerli olacak bir durumdur.

Aslında bu açıdan bakıldığında bilgisayarların olmadığı zamanlarda da bilgisayar mühendisliğinin var olması gerektiği ortaya çıkabilir. Bunun sebebi biraz daha İngilizce terimin içerisindeki anlamdan kaynaklanmaktadır. Computer terimini Türkçeye çevirirsek yönetici, idare edici, hükmedici anlamlarının yanında hesaplayıcı, işleyici, işlem yapıcı gibi terimlerle karşılayabiliriz. Buradaki anlam, örneğin bir robotun bir üretim bandını kontrol etmesindeki anlamı da taşımaktadır. Elbette bilgisayarlardan önce de üretim veya yönetim yapılmaktaydı, bu yönetimin düşük veya üst seviye olmasına bakılmadan insanlar tarafından yapıyordu. Ancak günümüzde bu bilgisayarlar ile yer değiştirmiş ve hızla da değiştirmeye devam etmektedir. Yine bir örnek üzerinden bakılacak olursa, trafiği günümüzde bilgisayarlar kontrol ediyor ve teknolojimiz tamamen trafiği izleyip öğrenerek en verimli şekilde örneğin bir kavşağı yönetecek bilgisayar yapmaya yeterli (ve örnekleri var) ancak bilgisayarlardan önceki dönemlerde bu iş kavşağa yerleştirilen bir polis memuru ile yapılıyor veya kavşaktakiler kaderlerine bırakılarak sürücülerin problemi çözmesi bekleniyordu.

Bu durumda bilgisayarlardan önceki zamanlarda da aslında insanlar tarafından yapılan işin bir algoritma üretilmesi ve bu algoritmanın daha iyi hale getirilmesi olarak yorumlanması yanlış olmaz.

## **Bilgisayarlar ve İnsanlar**

Bilgisayarlar insanların yerine geçebilir mi veya bilgisayarlar insanların işlerinin sonu olur mu gibi tartışmalara bir iki farklı açıdan bakmam mümkün.

Birincisi bilgisayarlar daha çok anamalcı (kapitalist, capitaist) bir yaklaşımın ürünüdür. Burada amaç insandan bağımsız olarak para ile kontrol edilmesi, üretilmesi ve yönetilmesi kolay bir sistem kurmaktır. Burada teknolojiyi anamal yoğun (capital intensive) ve emek yoğun (labour intensive) olarak ikiye ayırmak mümkün. Seçilen teknoloji insan gücüne daha çok ihtiyaç duyuyorsa emek yoğun, para ve sermaye birikimine daha çok ihtiyaç duyuyorsa anamal yoğun olarak isimlendirilebilir.

Bilgisayarların ağırlığının artması ise hiç şüphesiz emek yoğun sistemlerden anamal yoğun sistemlere geçişin bir göstergesidir. Örneğin bin kişilik bir fabrikayı artık on kişi ile işletebiliyorsanız daha çok paraya daha az insana ihtiyacınız vardır.

Bu durum hemen bilgisayarların, insanların işlerine ve gelirlerine son verdiği olarak yorumlanmamalıdır. Bu görüşü savunan pek çok kişiye karşılık bu görüşün tersini savunan

kişiler yeni açılan iş sahalarını ve gelişen teknoloji ile insanların daha az çalışmalarına (ve dolayısıyla geçinmek için daha az gelire ) artık ihtiyaç duymalarını göstermektedir.

### **İnsan ve makine çizgisi**

Yukarıda bilgisayarların giderek daha hakim olduğu bir dünya çizilirken buradaki insan ve makineler arasındaki sınırdan bahsetmek gerekir.

Bu noktada sıkça sorulan ve kurgu bilim filimlerine de sıkça konu olan “birgün insan gibi düşünen bilgisayarlar olabilecek mi?” ve tabi bunun bir adım sonrasındaki soru “bilgisayarlar birgün dünyayı ele geçirecek mi” gibi sorular bilgisayarların güncel hayata girmesinden çok önce çeşitli tartışmalara konu olmuş ve değişik kriterler ortaya konulmuştur.

bu kriterlerin en meşhurlarından birisi de Turing Testi olarak geçen testtir. Basitçe bir odadaki iki monitörün birisinin arkasında bir bilgisayar, diğerinin arkasında da bir insan oturmaktadır. Sorulan sorular ile hangi monitörün arkasında insan oturduğu anlaşılamıyorsa, söz konusu bilgisayar Turing Testini geçmiş sayılır. Şayet soruların hepsine insan gibi cevap verebilen bir bilgisayar yapılabilirse (ki bu soruların büyük kısmı insan gibi düşünmeyi gerektirir) bu durumda evet bir gün insanları tamamen ikame edebilecek (insanların yerine geçebilecek) bilgisayarlar yapmak mümkündür.

Ancak her konuda olduğu gibi bu konuda da Turing testine inananlar ve inanmayanlar olarak bilim dünyası ikiye ayrılmıştır. Kişisel olarak bu testin geçilebileceğine inanmamamla birlikte yapılan çalışmaların, bu testteki sorulara her geçen gün daha başarılı yanıtlar vermek yolunda ilerlediğini kabul etmek gerekir.

İşte bu noktada insa ve makine çizgisi devreye girmekte ve bir sistemde makinelerin yapacağı işler ile insanların yapacağı işler arasında bir sınır çizmektedir. En azından şimdilik bilgisayarlar herşeyi yapamadığına göre her sistemde insana ihtiyaç vardır. İnsanın devreye girdiği noktada ise arada bir sınır oluşmaktadır.

Bu sınırdan ise bilgisayarlar ile iletişim kurulan bir arayüze ihtiyaç duyulur. Pekçok kere görsel sanatlarında konusu olan bu sınırdan çeşitli ekranlar veya donanımlar vasıtasıyla insanın bilgisayarlar ile iletişim kurması üzerinde çalışılmıştır.

Bilgisayar mühendisliğinin bir kısım çalışmaları da bu noktada yoğunlaşmaktadır. Örneğin tasarım ve görsel öğelerin üretilmesi konuları, veya bilgisayarlar tarafından üretilen verilerin görüntülenmesi gibi konularda çalışma alanları hızla artmaktadır.

### **Bilgisayar Mühendisliği Eğitim İçeriği**

Üniversitedeki bir bilgisayar mühendisliği bölümünde olabilecek anabilim dallarını sıralamak istersek aşağıda yapılan listeye benzer bir liste elde edilebilir:

- Algoritma analizi (Algoritmalar teorileri olarak da isimlendiriliyor (Algorithm Analysis, Theory of Algorithms))
- Yazılım Mühendisliği (Software Engineering)
- Dosya yönetimi ve Veri tabanları (File organisation and Database management systems)
- İşletim sistemleri (Operating Systems)

- Otomata ve Programlama dilleri (Derleyici (Compiler) tasarımı olarak da geçiyor ve Automata Theory)
- Veri iletişimi ve ağ yönetimi (networking, data communication and networks)
- Yapay zeka (Artificial Intelligence)
- Veri Güvenliği ve Şifreleme (Cryptology)
- Bilgisayar Grafikleri ve Resim işleme (Computer Graphics and image processing)
- Mikro işlemciler (Micro processors veya Micro controllers)
- Bilgisayar Mimarisi (Computer Organisation)

şekliden saymak mümkündür. Bu listeye ekleme ve çıkarmalar yapılabilir. Ayrıca bu listede bulunmayan ancak bu listedeki gruplardan birkaçına birden giren pek çok çalışma alanı vardır. Örneğin Internet programlama konusunu hem veri iletişimine hemde algoritmalara koymak hatta bu konuyu bir proje yönetimi olarak ele alırsak yazılım mühendisliğine de dahil etmek mümkündür. Veya uzman sistemler, yapay sinir ağları veya doğal dil işleme gibi konuları yapay zekanın bir alt dalı olarak incelemek veya tamamen ayırmak da mümkündür.

Aslında bilgisayar mühendisliği eğitimi de bu anabilim dallarının hepsinden en az birer giriş ders olarak hepsi hakkında bir fikir sahibi olmaktır. Bu konuların tam olarak ne oldukları ve işe yarar seviyede bilgi alınması ancak yüksek lisans ve sonrasında mümkün olmaktadır.

Ne yazık ki bilgisayar mühendisliği bölümleri, piyasadaki eleman ihtiyacı ve buna bağlı istihdam olanaklarından dolayı akademide yeterli öğretim görevlisinin bulunmadığı ve pek çok üniversitede yukarıdaki konuların hepsinde çalışan hocaların bulunmasının istisna olduğu bölümlerdir. Dolayısıyla pek çok bilgisayar mühendisi yukarıdaki bu ana konuların çoğunu görme şansı olmadan mezun olmaktadır.

### **Türkiyede Bilgisayar Mühendislerinin Çalışma Alanları**

Bir ülkenin teknoloji üretim ve kullanım seviyesi olmak üzere iki farklı seviyesinden bahsetmek mümkündür. Türkiye'deki teknoloji üretim seviyesi ne yazık ki kullanılan seviyeye oranla düşüktür. Yani kullandığımız teknolojik gelişmelerin bir kısmını üretmek yeyrine hazır almaktayız. Bu durumda teknolojik üretimin gerektirdiği ve bir önceki başlıkta sayılan bazı anabilim dallarında insan istihdam edilememesi anlamına gelmektedir.

Genel bir bakışla Türkiye'deki bilgisayar mühendisleri şu alanlarda çalışmaktadır:

- Akademik çalışmalar (üniversite ve şirketlerin ar-ge bölümleri)
- Yazılım geliştirme
- Veritabanı yönetimi
- Sistem yönetimi (ağ yönetimi ve sunucu yönetimini kapsar şekilde)
- pazarlama ve satış sonrası veya öncesi teknik destek

Yukarıdaki liste ağırlıklı olarak bilgisayar mühendisliği veya benzeri bölümlerden mezun olan kişilerin Türkiye'de çalıştığı alanlardır. Ağırlıklı olarak pazarlama ve yazılım geliştirme konularında istihdam edilmelerine karşılık göreceli olarak çok az da olsa yukarıdaki listeye girmeyen bilgisayar mühendisliği alanlarında da çalışan kişiler vardır.

### SORU-69: İkillik Prensibi (Duality Principle, İstaniyet) hakkında bilgi veriniz.

Din ve felsefede benzer anlamlara gelmesine karşılık bu yazının amacı bilgisayar bilimleri için önemli olan matematikteki ikilik prensibini açıklamaktır.

Bir matematikçi bu kavramı basitçe şöyle açıklayabilir “boyalı elimle bir cama ellesem ve elimin izi camda çıksa, camın her iki yönünden gördüğüm görüntü birbirinin ikilidir (dual)”.

Bu açıklama aslında kavramın ne olduğunu göstermektedir. Örneğin iktisatta kârlılık arttırmak için yapılan bir çalışmanın maliyet azaltmak olması ve arttırmak (maximization) ve azaltmak (minimization) kavramlarının birbirinin ikili (dual) olması gibi.

Örneğin bilgisayar bilimlerinde mantık işlemlerinde (Boole algebra) kullanılan aşağıdaki eşitlikleri inceleyelim:

#### Yer Değiştirme (Commutative Law)

$$(a) \quad A \quad V \quad B \quad = \quad B \quad V \quad A$$
$$(b) \quad A \wedge B = B \wedge A$$

#### Birleştirme (Associate Law)

$$(a) \quad (A \quad V \quad B) \quad V \quad C \quad = \quad A \quad V \quad (B \quad V \quad C)$$
$$(b) \quad (A \wedge B) \wedge C = A \wedge (B \wedge C)$$

#### Dağılma (Distributive Law)

$$(a) \quad A \quad \wedge \quad (B \quad V \quad C) \quad = \quad (A \quad \wedge \quad B) \quad V \quad (A \quad \wedge \quad C)$$
$$(b) \quad A \quad V \quad (B \quad \wedge \quad C) \quad = \quad (A \quad V \quad B) \quad \wedge \quad (A \quad V \quad C)$$

#### Kendisi Kuralı (Identity Law)

$$(a) \quad A \quad V \quad 0 \quad = \quad A$$
$$(b) \quad A \wedge 1 = A$$

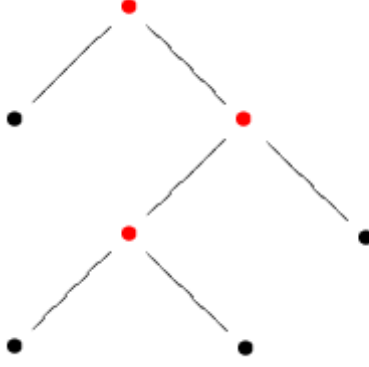
Yukarıdaki örneklerde görüldüğü üzere her eşitlikte ikil bir karşılık gösterilmiştir (a eşitliğinin ikili b, b eşitliğinin ikili de a eşitliğidir).

Örneğin mantık işlemlerinde yukarıdaki kurallar çerçevesinde DeMorgan kuralı geliştirilmiş ve bu kurala göre bütün V( veya, or) işlemleri  $\Lambda$  (ve, and) işlemi olarak yazılabilir. Tabi buradaki kuralımız her kaziyenin (önerme) tersinin alınmasıdır.

Not : Bu yazıya gelen sorulara cevap olarak. Dinde ikillik için örneğin yahudilikteki yetzer ha-ra (kötü olanı yapmak) ve yetzer ha-tov (iyi olanı yapmak) veya kurandaki zariyat 41/49 “herşeyi çift yarattık ki düşünüp ders alasınız”, Rahman 55/17 “iki doğunun ve iki batının rabbi” ayetlerine dayanan tefsirlerde ve şeytan ve melek ayırımında (ki bu ikillik hemen bütün semavi dinlerde vardır), veya Manihaizm ve Mecusilikteki “Nur” ve “Karanlık” kavramlarındaki (Yani iyilik tanrısı olarak aydınlık (ateşe) tapınmak ve kötülük tanrısı olarak karanlık) kavramlarında veya Monistik (tekçi) yaklaşımı benimsemiş özellikle katolik hristiyanlıktaki yine özellikle descartes’tan sonra açığa çıkan ruh ve beden ayırımında ikillik kavramları görülür. Elbette bu yazdıkların sadece basit birer örnek olarak alınmıştır ve heps ile ilgili çok geniş ve detaylı kaynaklar mevcuttur.

### SORU-70: Dış Yol Uzunluğu (External Path Length) hakkında bilgi veriniz.

Bir ağacın dış düğümlerine ayrı ayrı ulaşılması için geçilmesi gereken yol miktarıdır. Örneğin aşağıdaki ağaç için bu değeri hesaplayalım:



yukarıdaki ağaçta kırmızı renkli düğümler iç düğümdür. Siyah renk ile gösterilen düğümleri ise dış düğümlerdir.

Buna göre kökten başlandığında ağacın sol tarafında 1 adet dış düğüm vardır ve erişim 1 yolla yapılır. Ağacın sağında 3 adet dış düğüm vardır. Bunlardan en soldakine 3 sağındakine 3 en sağdakine ise 2 yol ile ulaşılır. Dolayısıyla bu ağacın dış düğüm sayısı  $1+3+3+2 = 9$  olarak bulunur.

Bir ağacın iç yol uzunluğu (internal path length) biliniyorsa dış yol uzunluğu aşağıdaki şekilde hesaplanabilir:

$$E = I + 2n$$

I: iç yol uzunluğu

E: dış yol uzunluğu

n: [iç düğüm sayısıdır.](#)

Yukarıdaki örnekte iç yol uzunluğu 3'tür. İç düğüm sayısı da 3 tür. Dolayısıyla  $E = 3 + 2 \times 3 = 9$  bulunabilir.

### **SORU-71: Kubit (Qubit) hakkında bilgi veriniz.**

Günümüz bilgisayar teknolojilerinin üzerine inşa edilmiş olan Von Neumann bilgisayarlarında en düşük veri ünitesi ikildir (bit). Benzer şekilde kuantum bilgisayarları içinde kubit (qubit = quantum bit) kullanılmaktadır. Normal ikilde (bit) sadece 1 ve 0 değerleri depolanabilirken bir kubit içinde 0, 1 veya her ikisi birden bulunabilmektedir. Bu konuyu daha iyi anlayabilmek için kubit kavramını daha detaylı okuyabilirsiniz.

Bir kubit, bazı elementlerin atomları üzerin inşa edilmiştir. Bu konuda hidrojen güzel bir örnek olabilir. Bilindiği üzere hidrojen atomu basit bir elementtir ve bir elektron ve bir çekirdekte oluşmaktadır. Elektronlar ise çeşitli enerji seviyesinde bulunabilirler. Bu enerji seviyeleri 0 veya 1 değerlerini göstermek için kullanılacak olsun. Basitçe elektronun en düşük yörüngede bulunması 0 en yüksek yörüngede bulunması ise 1 olarak ifade edilecek olsun.

Bu konuda ilave bir bilgi de elektronların LASER marifetiyle yörüngelerinin değiştirilebildiği ve yüksek ve düşük yörüngeler arasında hareket ettirilebildiğidir. Burada LASER sisteme foton

katmakta ve kısaca değeri 0 ile 1 arasında değiştirmektedir. Bu işlemi basit bir olumsuz (not) operatörüne benzetebiliriz. Yani mantıksal olarak giriş değerinin olumsuzunu almaktadır.

Kubitlerin normal ikilerden (bit) ayrıldığı nokta ise sisteme yörünge değiştirmek için gereken LASER etkisinin yarısının yapılması durumudur. İşte normal bir bitten kubitin ayrıldığı noktada burasıdır. Bu durumda elektron her iki yörüngede de bulunmaktadır. Bu duruma süper konum (Super position) adı verilmektedir.

Super konumun hesaplamalarda da sağladığı müspet etkiler bulunmaktadır. Kuantum paralelliği (Quantum parallelism) adı da verilen bu işlemde kubitlerin heri ki durumu da göz önünde bulundurulmaktadır. Yani kubit 0 veya 1 durumunda olduğunda sonucun alacağı iki farklı değer ayrı ayrı hesaplanmış gibi tek bir işlemde hesaplanmaktadır.

## **SORU-72: Sıralama Algoritmaları (Sorting Algorithms) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde verilmiş olan bir grup sayının küçükten büyüğe (veya tersi) sıralanması işlemi yapan algoritmalara verilen isimdir. Örneğin aşağıdaki düzensiz sayıları ele alalım:

5 9 2 3 7 11 -4 6

Bu sayıların sıralanmış hali

-4 2 3 5 6 7 11

olacaktır. Bu sıralama işlemi yapmanın çok farklı yolları vardır ancak bilgisayar mühendisliğinin temel olarak üzerine oturduğu iki performans kriteri buradaki sonuçları değerlendirmede önemli rol oynar.

- Hafıza verimliliği (memory efficiency)
- Zaman verimliliği (Time efficiency)

Temel olarak algoritma analizindeki iki önemli kriter bunlardır. Bir algoritmanın hızlı çalışması demek daha çok hafızaya ihtiyaç duyması demektir. Ters durumda da bir algoritmanın daha az yere ihtiyaç duyması daha yavaş çalışması demektir. Ancak bir algoritma hem zaman hem de hafıza olarak verimliyse bu durumda diğer algoritmalarından başarılı sayılabilir.

Genellikle verinin hafızada saklanması sırasında veriyi tutan bir belirleyici özelliğinin olması istenir. Veritabanı teorisinde birincil anahtar (primary key) ismi de verilen bu özellik kullanılarak hafızada bulunan veriye erişilebilir. Bu erişim sırasında şayet belirleyici alan sıralı ise erişimin logaritmik zamanda olması mümkündür. Şayet veri sıralı değilse erişim süresi doğrusal (linear) zamanda olmaktadır.

## **SORU-73: Gerekircilik (Nedensellik, Determinism) hakkında bilgi veriniz.**

Bir olayın başka bir olayı gerektirmesi durumudur. Basitçe neden-sonuç ilişkisine dayanlı felsefi yaklaşımdır. Buna göre bütün olaylara nesnel bir yaklaşımda bulunulur ve bu yaklaşım her zaman aynı sonucu verir.

gerekircilik yaklaşımı aynı zamanda bir düşünce yapısının incelenmesinde de kullanılabilir. Örneğin ekonomik nedesellik (economic determinism) altında bir tarihi olayı incelemek bu tarihi olaydaki her sebep-sonuç ilişkisinin bir ekonomik açıklamayla yapılabileceğini iddia etmek demektir.

#### **SORU-74: İstikra ile ispat (Tüme varım, Proof by Induction) hakkında bilgi veriniz.**

Bir kaziyeyi (önerme) ispat ederek nazariye (teorem) elde etme yöntemidir. İstikra cüz'îler (tikeller) den küllî (tümel) ye gitme yöntemidir dolayısıyla örneklerden yola çıkarak her zaman için geçerli bir sonuç elde ederek ispat yapılır.

Her istikra için bir esas(basis) bir de istikra(induction) safhası(step) bulunur. Bu iki safhanın ispatı bütün durumların ispatı demektir. Çünkü isikra edilen herşeyin dayanacağı bir esas bulunmalıdır.

Bu durum şöyle bir örnek ile de ifade edilebilir örneğin iki ayaklı bir sandalyenin ayakta durabilmesi için başka bir iki ayaklı sandalyeye dayanması gerekir, onunda ayakta durması için bir diğerine ve böylece bütün sandalyelerin bir diğerine dayanması gerekir. En başta ise bir adet 4 ayaklı sandalyeye ihtiyaç vardır ki bütün birbirine dayanan sandalyeler gelip bu sandalyeye dayansın. İşte şayet en başta bulunan bu 4 ayaklı sandalye yanı esas safhası (basis step) ispat edilirse ve her sandalyeninde birbirine dayanarak ayakta duracağı yani istikra safhası (induction step) ispat edilirse istenilen sayıdaki sandalyenin ayakta durabileceği ispatlanmış olur.

Daha sayısal bir örnek için 0'dan n'e kadar olan sayıların toplamının  $(n * (n + 1)) / 2$  olduğunu ispatlayalım.

bu ispattaki esas safhamız (basis step)  $n = 1$  durumu olsun ve bu durumda  $(1 * (1 + 1)) / 2 = 2 / 2 = 1$  olarak denklemimizin doğru çalıştığını gösterebiliriz.

istikra safhamız (induction step) ise  $n + 1$  için olan durum olmalıdır. Bu durum da  $((n + 1) * ((n + 1) + 1)) / 2$  olur. daha sade bir şekilde  $((n + 1) * (n + 2)) / 2$  yazılabilir.

şayet istikra safhamızı ispatlayabilirsek esas safhamızdaki ispata dayandırarak istikra edebiliriz ( bütün durumlar için genelleme yapabiliriz).

şayet n sayının toplamı  $(n * (n + 1)) / 2$  ise  $n+1$  sayının toplamı da  $(n * (n + 1)) / 2 + (n + 1)$  olmalıdır.

isikra safhasında elde ettiğimiz sonuç da  $n+1$  sayının toplamını vermektedir o halde :

$$((n + 1) * (n + 2)) / 2 = (n * (n + 1)) / 2 + (n + 1)$$

eşitliği doğru olmalıdır denilebilir. Bu eşitlik çözülecek olursa sonucun  $1=1$  doğruluğu ile ispat edildiği görülür.

Dolayısıyla bir esas alınmış ve ispat edilmiştir. Ayrıca bu esasa bina edilen istikra adımları da ispat edilmiştir. Dolayısıyla bütün pozitif sayılar için doğru sonuç veren bir istikra yapılmıştır. Bu duruma tam istikra da denilebilir.

### **SORU-75: İstikra(Tüme varım, Induction) hakkında bilgi veriniz.**

İstikra lügatte, etraflıca düşünüp araştırma (tetebbu') anlamına gelir. Bir yöntem olarak ise cüz'iler (tikeller) den küllî (tümel) ye gitme yöntemidir. Mantıkçılara göre istikrâ, küllînin hükmünü vermek için cüz'iler hakkında verilen hükümleri kapsayan önermelerden oluşan bir sözdür. Bir başka deyişle, o, cüzlerinin genelinde bulunması nedeniyle küllî hakkında verilen hükümdür.

İstikrâ iki kısma ayrılır: a) Tam İstikra; Kıyasun Mukassim. O, bütün cüzleriyle küllîye delâlet eden ve onun hakkında hüküm veren kıyastır. Bu, çok az kullanılmakla birlikte, kesinlik ifade eden istikradır. Mesela, her cisim ya hayvan ya bitki ya da cansız madde (cemâd) dir. Bunlardan her biri yer kaplar (metehayyiz), o halde her cisim yer kaplar, şeklindeki bir kıyas bu türdendir. b) Nâkıs İstikra. Bu, küllîye, sadece cüz'ilerinin çoğu ile delâlet eden ve onun hakkında hüküm veren istikradır. Bu yüzden o, kıyasın türlerinden biri olarak kabul edilir ve zan ifade eder. Mesela, insan, at, eşek, öküz vb. araştırdığımız hayvanlardan gözlemlediğimize göre, her hayvanın çiğneme esnasında alt çenesi hareket eder, şeklindeki istikra bu türdendir ve timsah gibi hayvanlar buna muhalefet eder.

### **SORU-76: burhan-ı mütenakıs (proof by contradiction, olmayana ergi) hakkında bilgi veriniz.**

Çok kullanılan ispat yöntemlerinden birisidir. Buna göre ispatlanmak istenen kaziyenin (önermenin) tersinin yanlışlığı ispat edilirse sonuca ulaşılmış ve bir nazariye (teorem) elde edilmiş demektir.

---

Basit bir günlük örnek şu şekilde verilebilir. Örneğin Ali, Ahmetin kapıdan girdiğini gördü ve Ahmet'in elbiselerinin kuru olduğunu gördü. Ali yağmur yağmadığını ispatlamak için yağmurun yağdığını farz edebilir. Bu durumda "Şayet yağmur yağsaydı Ahmet'in üzeri ıslak olurdu" çıkarımını yapar. Sonuç olarak Ahmet'in üzeri kuru o halde yağmur yağmıyor çıkarımı ile yağmurun yağmadığını ispatlamış olur.

---

Farklı bir örnek ise  $\sqrt{2}$  sayısının kesirli ifadesinin imkansız olduğunu (irrasyonel sayı olduğunu) göstermek olabilir.

Bu durumda öncelikle kaziyenin (önermenin) tersini doğru kabul etmek gerekir ve farz edelim ki  $\sqrt{2}$  sayısı kesirli bir sayı olsaydı o halde

$\sqrt{2} = n/m$  şeklinde bir kesir ile ifade edilebilirdi.

Kesirli sayılarda bilindiği üzere pay ve paydadaki sayılar tam sayı olmak zorundadır. Ayrıca yine hatırlanırsa kesirli sayılarda hem pay hem de paydadaki sayı aynı tam sayıya bölünürse değer bozulmazdı. Bu durumda m ve n sayılarının aynı anda çift sayı olamayacağını kabul etmek gerekir (velev ki çift olsalardı 2 ile sadeleştirilirler ve yine çift olmayan bir sonuca ulaşıldı)

Eşitliğin her iki tarafını n sayısı ile çarparsak eşitlik bozulmaz:

$$n \sqrt{2} = m$$



Her iki tarafın karesi alınırsa yine eşitlik bozulmaz

$$2n^2 = m^2$$

Yukarıdaki eşitlik dikkatlice incelendiğinde  $m^2$  ifadesi başka bir ifade olan  $n^2$  ifadesinin 2 mislidir. Bilindiği üzere bir ifade başka bir ifadenin 2 misli ise bu ifade çift sayıdır. Yani yukarıdaki örnekte bulunan  $m^2$  sayısı çift sayı olmak zorundadır.

Ancak ne yazık ki başta kabul ettiğimiz ve  $m$  sayısının tek sayı olduğu gerçeği ile bu durum tenakuza düşmektedir (çelişmektedir) çünkü hiçbir tek sayının karesi çift olamaz. Bu durumun daha doğru gösterimi aşağıdaki şekilde yapılabilir:

$m$  sayısı çift sayı ise  $m=2k$  şeklinde yazılabilir.

o halde  $2n^2 = (2k)^2$  şeklinde önceki eşitlikte yerine konulması da doğru olmalıdır.

$2n^2 = 4k^2$  eşitliğinde her iki tarafa da 2'ye bölünürse  $n^2 = 2k^2$  eşitliği elde edilir.

Bu durum ise daha önce de ifade edildiği üzere  $n^2$  sayısının çift olması demektir. Yine hatırlanırsa hiçbir tek sayının karesi çift olamaz. Ve yine hatırlanırsa aynı anda hem  $n$  hem de  $m$  çift olamaz.

---

Görüldüğü üzere yukarıdaki iki örnekte de öncelikle ıspatlanmak istenen kaziyenin (önermenin) tersinin doğru olduğu farz edilmiş ve bu faraziyenin yanlışlığı ıspatlanmıştır. Bu yanlışlık gösterilirken de iki farklı yargı elde edilip bu yargıların tenakuza (çelişkiye) düşmelerinden faydalanılmıştır.

Unutulmamalıdır ki bu yöntemin geçerli olabilmesi için ikili mantık içinde çalışmak gerekir (yani doğru ile yanlışın birbirinin tersi olduğu mantık, 3. bir ihtimalin olmadığı mantık).

### **SORU-77: Binaen Burhan (İnşâa ile İspat , Proof by Construction, Binaenaleyh) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde kullanılan ıspat yöntemlerinden birisidir. Bu yöntemde bir varlığın oluşmasının gösterilmesi hedeflenir. Örneğin aşağıdaki teoriyi inşaa yöntemi ile ıspat edelim:

“2’den büyük her çift  $n$  sayısı için  $n$  düğüm içeren 3-düzenli graf bulunur”

Öncelikle  $k$ -düzenli graf tanımını hatırlayalım:

Bir graf üzerindeki her düğümün “ $k$ ” kadar komşusu bulunması durumuna  $k$ -düzenli graf denilir. Örneğin aşağıdaki graf 2-düzenli bir graftır çünkü her düğümün derecesi 2’dir.

Dolayısıyla ıspatlanmak istenen nazariyede bize düğüm derecelerinin en az 3 ve daha fazla olabildiği bildirilmiş bir  $k$ -düzenli graf isteniyor.

Bir graftaki kenarları aşağıdaki küme ile ifade etmek mümkündür:

$$K =$$

$$\{ \{i, i+1\} \mid 0 \leq i \leq n-2 \} \cup \{ n-1, 0 \} \}$$

$$\cup \{ \{i, i+n/2\} \mid 0 \leq i \leq n/2-1 \}$$

Yukarıda iki satır olarak tanımlanan kenarlar kümesindeki ilk satırı komşu düğümleri gösteren satır ve ikinci satırı karşı düğümleri gösteren satır olarak düşünebiliriz. Buna göre  $n$  adet düğüm içeren bir grafta her düğüm içi kendisinden önce ve kendisinden sonraki düğümlere birer kenar olduğu kabul edilirse (ilk satır) ayrıca karşısındaki düğüme de bir kenar olduğu kabul edilirse (alttaki satır) bu durumda her düğümün 3. derece olması mümkündür denilebilir.

Grafın bir çember olmasını sonsuz sayıda düğüm içermesi olara düşünebiliriz. Bu çember üzerindeki bir noktayı komşusu olan 2 noktaya bağlayan birer kenar (mavi ile gösterilmiştir) ve tam karşısında buluna noktaya bağlayan (kırmızı ile gösterilmiştir) birer kenar çizilmesi durumunda  $n > 2$  düğümü bulunan herhangi bir graf için 3-düzenli graf bulunabileceğini tasvir etmiş oluruz.

Yukarıdaki örnekte kullanılan ispat yöntemine bakıldığında bir varlığı oluşturan diğer varlıkların ispatta kullanıldığını görürüz. Örneğin 3-düzenli graf kavramını oluşturan bir düğüm ve 3 kenar varlıklarının ayrı ayrı gösterilmesi ve bu varlıkların üzerine ispatın inşası mümkün olmuştur.

### **SORU-78: Nazariye (Teori, Kuram, Theorem) hakkında bilgi veriniz.**

Bilgisayar bilimleri açısından matematiksel olarak ispat edilmiş kazyeler (önergeler, statements) birer nazariyedir. Bazı kazyelerin (önergelerin) doğruluğu ise sırf farklı teorilerin ispatına yardımcı oluyor diye ispatlanır. Bu tip kazyelere (önergelere) ise önkuram (önsav, lemma) adı verilmektedir. Bazı durumlarda ise bir nazariyenin ispatı bize bazı başka neticelerin doğru olduğunu gösterir. Bu tarz kendiliğinden doğruluğu anlaşılan kazyelere (önergelere) ise tabîî sonuç anlamına gelen sonurgu (corollary) denilmektedir.

### **SORU-79: Bilgi, Veri, Mâlûmat, İrfan (Knowledge, Data, Information, Wisdom) hakkında bilgi veriniz.**

İngilizce'de terim olarak yerleşmiş ve bilgisayar bilimleri için hayatî öneme sahip kelimeleri Türkçede çoğu zaman sadece bilgi ile karşılayan tercümeler bulunuyr. Ancak bu 4 kelime de farklı anlamlara sahip ve aralarındaki farklar burada anlatılacaktır. Öncelikle İngilizcede bulunan ve Türkçede aynı kelimeyle karşılanan 4 kelime için 4 farklı karşılık bulalım. Bunlar aşağıda verilmiştir:

Data : Veri

Information : Mâlûmat

Knowledge : Bilgi

Wisdom : İrfan, Bilgelik

Yukarıdaki kelimeler sırayla verilmiştir ve bu sırada verinin işlenmesi ve işlendikçe daha değerli sonuçlara erişilmesi mümkündür. Yani basitçe İrfan, bilgiden; bilgi, mâlûmattan; mâlûmat ise veriden işlenerek çıkarılır ve en ham hal olan veri en değersiz ve çoğu zaman en az işe yaryan şeyken irfan en değerli ven çok işe yarayan sonuç hâline gelmektedir.

Örneğin bir arabanın hızını veri olarak kabul edebiliriz.

Bu hızın ne anlama geldiği, örneğin tehlikeli bir hız veya varılacak yere geç kalmaya sebep olacak bir hız oluşu mâlûmat olarak kabul edilebilir.

Bu hızn arttırılması azaltılması diğer geçmiş tecrübeler ile kıyaslanması gibi eylemlerin sonuç ve yolu ise bilgi olarak kabul edilebilir.

Şöförlük bilgisi, tecrübe ve melekesi ise irfan olarak kabul edilebilir. Yani örneğin Şöför Ali denildiğinde, Ali'nin şöförlük bilgeliğine sahip olduğu kabul edilir.

Yukarıdaki örnekte bütün kelimeler tek bir örnek üzerinde gösterilmiştir. Elbette şöförlük bilgeliğinin altında hız dışında sayısız farklı veri, ve bu verilerin sunduğu mâlûmatlar ve bu mâlûmatlardan elde edilen bilgiler bulunmaktadır. Dolayısıyla bilgelikten veriye kadar giden yol belirsiz(nondeterministic) bir yoldur. Bu yolun tersi de ne yazık ki belirsizdir çünkü hız verisi şöförlük için kullanılabileceği gibi örneğin muavinlik, yolculuk, işletmecilik (otobüs işletmesi) gibi farklı irfanlar için de anlamlı olabilir. Dolayısıyla bu kavramları doğrusal bir yapıya oturtmak yanlış olacaktır.

### **SORU-80: Normal Şekil (Canonical Form) hakkında bilgi veriniz.**

Bir bilginin normal gösterimidir. Örneğin bir çok terimlinin (polynom) normal gösterimi üssel olarak büyükten küçüğe doğrudur.

Değer olarak  $x^2+x+20$  ile  $x + 20 + x^2$  aynı olmasına karşılık bu çok terimlinin (polynom) normal şekli (canonical form)  $x^2+x+20$  'dir.

### **SORU-81: Bilgi Çıkarımı (Information Extraction) hakkında bilgi veriniz.**

Bilgi çıkarımı konusu, genellikle bir metin üzerinde doğal dil işleme kullanılarak belirli kriterdeki bilgileri elde etmeyi hedefler. Bu işlem sırasında örneğin bir kalıba uygun olan verilerin çıkarılması istenebilir. Amaç çok miktardaki veriyi otomatik olarak işleyen bir yazılım üreterek insan müdahalesini asgarî seviyeye indirmektir. Bilginin çıkarılacağı ortam genellikle yazılı metinlerdir ancak bu metinlerin bulunacağı ortamlar değişebilir örneğin veri tabanları, internet üzerindeki dökümanlar veya taranmış metinler bu verinin kaynağını oluşturabilir.

Bilgi kelime anlamı olarak verinin işlenmiş ve anlaşılabilir veriyi ifâde etmektedir. Dolayısıyla veri kaynağı olan metinlerden işlenmiş bilginin çıkarılması işlemine bilgi çıkarımı denilmektedir. Örneğin gazete haberleri veri kaynağı olarak kabul edilsin. Bu veri kaynağından şirket birleşmeleri ile ilgili bir bilginin çıkarılması işlemine bilgi çıkarımı denilebilir. (a şirketi ile b şirketinin birleştiğinin gazete haberlerinden anlaşılması gibi)

Bilgi çıkarım işleminin en zor adımlarından birisi de veriyi işlerken belirli bir yapıya oturtmaktır. Örneğin internet üzerinde yayınlanan verilerin herhangi bir standart yapısı bulunmamakta, veriler dağınık halde istenildiği gibi yayınlanmaktadır. Bu verilerin düzenli bir hale getirilmesi için XML ve benzeri teknolojilerden faydalanarak bilgi çıkarımı işleminin basitleştirilmesi hedeflenmektedir. Bu konudaki güncel uygulamalardan birisi de internet üzerinde yayın yapan kurumların birer ağ hizmeti (web service) kurarak uygulamaların karşılıklı iletişimine izin vermeleridir.

Ayrıca çok sayıda yazılım bilgi çıkarımı aşamasına alt yapı hazırlamak amacıyla çeşitli ortamlardan (örneğin internet) veri toplayarak bunları düzenli bir halde yapılandırır.

Güncel uygulamalarda bilgi çıkarımı sırasında sık rastlana problemler şunlardır:

- Varlık isimlerinin algılanması (Named Entity Recognition): Bu konu, bir veri kaynağında geçen varlıkların ve isimlerinin algılanmasını hedefler. Bu varlıklar kişi, bina, yer, soyut varlıklar veya sayısal ifadeler olabilir.

- Eş Atıf (Coreference) : Aynı varlığa işaret eden birden fazla kelimenin tespiti. Örneğin zamirlerin isimlere atıfta bulunması gib. Bu bağlantılar sonucunda bir isim zinciri elde edilebilir.
- Terim Çıkarımı (Terminology Extraction): Verilen metne ait terimlerin çıkarılması

**SORU-82: Belirsiz Çokterimli Tam (NP-Complete, Nondeterministic Polynomial Complete) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde problem sınıflamada kullanılan sınıflardan birisidir. Bu sınıfa giren problemler için çözümleme zamanı arttıkça artan (super increasing) yapıya sahip olmaktadır. Buna göre her adımdaki çözümleme zamanı kendinden çözümleme zamanlarından daha fazladır.

Problem yapı olarak artan zamanda çözüldüğü için de bu problem tiplerinin çokterimli zamanda (polynomial time) çözülmesi mümkün değildir. Bu problemin tanımında ayrıca Turing Makinesinden de yararlanılır.

Turing makineleri beliri (deterministic) ve belirsiz (nondeterministic) olarak ikiye ayrılır. Buna göre NP-Complete bir problem Belirsiz Turing Makinesi tarafından belirli zamanda çözülebilmektedir.

Ayrıca problemleri kümelerken diğer bir küme olan P kümesi yani çokterimli (polynomial) kümesi, NP kümesinin bir alt kümesi olarak kabul edilir. Bir problemin NP kümesinde olduğunu ispatlamanın yollarından birisi de bu problemi NP kümesinde bulunan başka bir probleme dönüştürmektir.

**SORU-83: Açgözlü Yaklaşımı (Greedy Approach) hakkında bilgi veriniz.**

Algoritma üretme yöntemlerinden birisi olan açgözlü yaklaşımına göre mümkün olan ve sonuca en yakın olan seçim yapılır. Yani basitçe bir seçim yapılması gerektiğinde sonuca en çok yaklaştıracak olan seçimin yapılmasını önerir. Ancak mâlum olduğu üzere bu seçim her zaman için en iyi seçim değildir.

Örneğin para üzeri verilmesi (coin exchange problem) için açgözlü yaklaşımının kullanılmasını düşünelim. Bu problemde bir satıcı kendisinden alışveriş yapan kişiye para üzeri vermektedir. Ödenmesi gereken miktar bu örnekte 24 olsun ve para birimlerimiz 20, 19, 5, 1 olsun. (yani para birimi olarak bu para birimleri bulunuyor)

açgözlü yaklaşımına göre satıcı 24'ü tamamlamak için elindeki para birimlerinden sonuca en çok yaklaştıran 20lik birimi seçecektir. Daha sonra geriye kalan boşluğu ( $24-20=4$ ) doldurmak için elindeki tek imkan olan 4 tane 1lik birimle dolduracaktır ve toplamda 5 adet bozuk parayı müşteriye geri verecektir. Oysaki aynı problem bir 19luk bir de 5lik bozuk paralar ile çözülerek 2 bozuk para vermek mümkün olabilirdi.

**SORU-84: Geçişli Fiiller (transitive verbs) hakkında bilgi veriniz.**

Geçişli fiiller, fiillerin nesne alabilenlerini içeren bir alt kategorisidir. Örneğin yemek fiili geçişli bir fiildir çünkü “ekmeği yedi” şeklinde bir nesneye atıfta bulunabilir ancak örneğin güldü kelimesi geçişsiz bir fiildir çünkü “birşeyi gülemeyiz”. Fiilin geçişli olup olmadığının anlaşılması için fiile “neyi,kimi,ne ” sorusu sorulabilir ve fiil buna cevap verebiliyorsa geçişli kabul edilebilir.

Geçişsiz fiillere “-(i)t, (i)r, tir” ekleri getirilerek geçişli fiil haline getirilebilir.

Örneğin yukarıdaki gülmek fiili geçişsiz bir fiildir. Güldürmek şeklinde -ür eki eklenerek geçişli olabilir ve kimi sorusuna cevap verebilir.

“Aliyi güldürdük”

cümlesinde olduğu gibi.

aynı zamanda zaten geçişli olan bir fiile “-(i)t, (i)r, tir” ekleri getirilerek geçişlilik derecesi de arttırılabilir.

Örneğin yukarıdaki örnekte yemek fiili zaten geçişlidir. Yedirmek fiili (ir) eki aldığında ettirgen özelliğe sahip olur.

“Ekmeği yedirdi”

cümlesinde olduğu gibi eylemi başka bir faile yaptırma fiili haline dönüşür.

Daha fazla bilgi için geçişsiz fiiller (intransitive verbs) ve çift geçişli fiiller (bitransitive verbs) bakabilirsiniz.

### **SORU-85: İkili dil (bigram) hakkında bilgi veriniz.**

Bir dilde kelimebilimsel olarak iki ihtimalin bulunması durumudur. Örneğin L1 ve L2 olarak iki farklı sözlüksel (lexical) grupumuzun olduğunu düşünelim. Ve yine örneğin bu iki grup N ve V (isim ve fiil) grupları olsun. Bir kelimenin dahil edileceği grubun belirsizliği (veya istatistiksel olarak belirliliği) bigram olarak ifade edilir. Örneğin “uç” kelimesi Fiil veya İsim grubuna dahil edilebilir. Doğal dil işleme dünyasında bigram diller istatistiksel diller olarak kabul edilirler ve bir olasılık değerlendirmesine tâbî tutulurlar. Örneğimiz için  $P(N|uç)$  veya  $P(V|uç)$  şekline iki adet farklı şartlı olasılık çıkarılabilir.

### **SORU-86: Kelime Bilim (lexicology, vocabulary) hakkında bilgi veriniz.**

Kelime bilimi, doğal dil için tanımlanmış ve kelime seviyesindeki çalışmaların bir araya geldiği bilimin ismidir. Basitçe bir dili meydana getiren en küçük anlamlı birim kelimedir. Her kelimenin ifade ettiği bir anlam bulunur.

Dil felsefesindeki zayıf bir akıma göre “ismi olan herşey vardır” denilebilir. Yani bir şeyin varlığını ispat için isminin olması yeterlidir. Bu yaklaşımın tersi ele alınırsa olmayan birşeyin de ismi olamaz.

Basit anlamda varlıklara(yaratıklara) isim verilmesinin sonucunda kelimeler ortaya çıkar. Burada belki, soyut varlıkların (mücerret isimlerin, (abstract noun)) anlaşılması, somut varlıkların (müşahhas isimlerin (concrete names)) varlığını anlamak kadar kolay olmayabilir.

Aslında her kelime varlıkları ifade eden birer atıftır. Yani kalem kelimesi bu kelimeyi okuyan herkes için geçmişindeki farklı bir bilgiye bir atıftır. Herkesin kalem kelimesi ile aklına gelen varlık farklı olabilir. Yine kelime olarak “kalemler, kalemde, kalemin” gibi ek almış halleri de

aynı varlığa atıftır. Hepsine ifade edilen varlık aynıdır. İşte bu ifade edilen varlığa dil biliminde lexeme (zâtı) ismi verilir. Yani varlığın bizzatihi kendisi anlamındadır.

Yukarıdaki üç kelime (kalemler, kalemde, kalemin) anlatılan anlam aynıken şekilleri farklıdır. Bu şekillerin incelendiği bilime şekil bilim anlamında morphology ismi verilir. Dolayısıyla farklı şekillerde olan kelimeler (morpheme) aynı varlığı (lexeme) ifade için kullanılabilir.

#### **SORU-87: Ekleme (apposition) hakkında bilgi veriniz.**

Cümle içerisinde bir kelimenin bir veya daha fazla kelime grubu ile ifade edilmesidir. İngilizce için genelde bir isim ile isim kelime grubu arasında kurulan ilişki türüdür. Örneğin “Alice, long haired girl, has followed the white rabbit” cümlesindeki “long haired girl” alice’i anlatan isim kelime grubudur. Benzeri durum Türkçe için isim ve alt cümle arasında kurulabilir. Örneğin “O yıllarda düşmanları tarafından da “muhteşem” olarak anılan Kânunî osmanlı devletinin padişahıydı” cümlesindeki “o sıralar düşmanları tarafından da “muhteşem” olarak anılan” cümlecisi Kânunî’nin belirleyicisi olarak kullanılır.

Ekleme olarak anılmasının sebebi bir kelime hakkında ilave bilgiler sunması ve cümleye eklenerek tek bir kelimeye atıfta bulunmasıdır.

#### **SORU-88: Dönüştü (anaphora) hakkında bilgi veriniz.**

Doğal dil işleme açısından dönüştüler, bir kelimenin daha ileriki cümlelerde işaret edilmesi durumunda ortaya çıkar. Örneğin “Ali çantasını evde unuttu. O okulda gitti” cümlelerinde O kelimesi aliye, çantaya veya eve işaret edebilir. Bu kelimelerin hepsi isimdir ve bir zamir ile işaret edilebilecek kelimelerdir. Dönüştü durumu bu ve benzeri cümlelerde ortaya çıkar. Her zaman bir zamir kullanılması gerekmez, benzer durumlar örneğin gizli özneyle de ortaya çıkabilir. Örneğin “O gün, Ali’nin çiftlikteki ilk günüydü. Bütün gün ata bindi.” cümlelerinden ikinci cümlede gizli özne Ali’dir. Benzer bir durum da fiil kelime gruplarını ifade etmek için kullanılabilir. Örneğin “Bugün Ali’yi başkanlıktan azlettim. Beklenmedik durumlarda bunu yapma yetkim bulunuyor” cümlelerindeki “bunu” kelimesi bir önceki cümledeki “başkanlıkta azletme” eylemine işaret etmektedir.

Bir kelimeye ileriki cümlelerde işaret edilmesinin belirlenmesi ve bu işaretin tam olarak hangi kelimeye yapıldığının ortaya konması dönüştü çözümleme (anaphora resolution) olarak geçer. Örneğin yukarıdaki örneklerde zamir veya gizli öznenin taşıdığı anlam bir önceki cümledeki Ali’dir yargısı bu çözümlemenin sonucudur.

#### **SORU-89: İlgı Belirsizliği (reference ambiguity) hakkında bilgi veriniz.**

Doğal dil işleme açısından bir kelimenin hangi kelime ile ilişkili olduğunun muğlak olduğu durumlardır. Genelde zamirler için bu problem yaşanır. Örneğin “Ali ile Veli kavga ettiler. O çok sinirliydi.” cümlelerindeki O zamirinin tam olarak Ali’ye mi Veli’ye mi işaret ettiği belirsizdir.

#### **SORU-90: Yapısal Belirsizlik (Structural Ambiguity) hakkında bilgi veriniz.**

Bir kelimenin yapısal olarak ifade ettiği bilginin belirsiz olmasıdır. Tam bir yapısal belirsizliğe örnek “yaşlı kadın ve erkekler” örneğinde görülebilir. Burada yaşlı olan sadece kadın mıdır?

Yoksa kadın ve erkekler midir? Elbette bu soruya bağı olarak çoğul olan kimdir sorusu da sorulabilir (kadınlar mı kadın mı ?)

Yapısal belirsizlikler genelde bir kelime-ifade belirsizliği durumunun bağlaç ile birlikte kullanılmasında oluşurlar. Örneğin “bakan gözler” kelimeleri için, Aliye bakan gözler, Devletin menfaatlerini bir millet vekillerinden seçilen bir bakan gözler şeklinde üç farklı anlam kazandırılabilir. Burada bağlanmış kelimeler fiil-fiil, isim-fiil olmasına göre ayrılırlar.

#### **SORU-91: Kelime-İfade Belirsizliği (word-sense ambiguity) hakkında bilgi veriniz.**

Bir kelimenin ifade ettiği anlamdaki belirsizliktir. Anlam belirsizliği de denilebilir. Genellikle bir dilde bulunan eşsesli kelimelerden kaynaklanan belirsizliklerdir. Örneğin Türkçede yüz kelimesi türkçede 4 farklı anlama gelmektedir ( sayı, denizde yüzmek, deri yüzmek ve çehre anlamlarında). Bu kelimenin cümle içerisinde kullanımı da problem oluşturmakta ve cümlede tam olarak hangi anlama geldiğini belirlemek gerekmektedir.

#### **SORU-92: Belirsizlik (ambiguity, muğlaklık, ikircimlik) hakkında bilgi veriniz.**

Doğal dil işleme açısından belirsizlik bir cümlede birden fazla anlamı bulunduran kelime, kelime grubu yada cümle sonucunun olmasıdır.

#### **SORU-93: Bağlaç (conjunction) hakkında bilgi veriniz.**

İki farklı anlamsal kavramı birbirine bağlamak için kullanılan kelime tipleridir. Uyum bağlaçları (coordinate conjunction) ve Şart bağlaçları (Subordinate conjunction) (veya zarf bağlaçları veya yan cümlecik bağlaçları) olarak iki grupta incelenebilir.

Uyum bağlaçları anlamca birbirine yakın iki grubu bağlamak için kullanılır. Örneğin “ali ve veli okula gitti” cümlesindeki ve bağlacı anlamca uyumlu iki kelime olan ali ve veli fâillerini birbirine bağlamıştır.

Şart bağlaçlarında ise genelde iki cümle bir şart üzerinden birbirine bağlanır. Örneğin “yağmur yağar ise şemsiyemi açarım” cümlesindeki ise bağlacı iki cümleyi (yağmur yağmak ve şemsiyeyi açmak) birbirine bir şartla bağlamıştır. Başka bir örnekte “güneş doğduğu ile saldırı başlayacak” cümlesinde ile bağlacı yine iki cümlecği birbirine bağlamaktadır.

Bağlaçlar temel olarak kelime gruplarını birbirine bağlamakta kullanılırlar.

isimler: kediler ve köpekler [dolaşıyordu]

sıfatlar: siyah ve kırmızı [renkli elbise]

zarflar: hızlı ve akıcı [konuşuyordu]

fiiller: oynadı ve kazandı

#### **SORU-94: Edat (preposition) hakkında bilgi veriniz.**

Bir isim kelime grubunun cümle içerisindeki rolünü belirten kelime çeşididir. Örneğin “ile” edadı genelde bir NP (noun phrase, isim kelime grubunun) kullandığı aracı ifade eder. Örneğin



“ankaraya uçak ile gitti” cümlesindeki gitmek fiilinin uçak aracı ile olması gibi veya “tahtaya kalem ile resim çizdi” cümlesindeki çizmek fiilinin kalem aracı ile olması gibi. Bazı durumlarda da bir eylemin ikinci etmenini (co-agent) ifâde için kullanılır. Örneğin “ali ile tahtayra resim çizdi” cümlesindeki “ile” edadı ikinci bir etmeni ifade etmek için kullanılmıştır.

Edatlar, bazı durumlarda isim kelime gruplarını bağlamak için de kullanılabilirler. Ayrıca bir isim kelime grubunun yapabileceği eylemi belirtmek için de kullanılırlar (animate). Yani her eylem her isim kelime grubu tarafından yapılamaz. Örneğin “balta camı kırdı” ve “çocuk camı kırdı” cümlelerinden ilkinde balta kelimesi bir canlandıran (etken, fail, animate) değildir çünkü . Buna mukâbil ikinci cümlede çocuk kelimesi etken olarak kabul edilebilir.

#### **SORU-95: zarf (adverb) hakkında bilgi veriniz.**

Zarflar bir fiili (verb) bir sıfatı (adjective) veya başka bir zarfı tanımlamak için kullanılan kelimelerdir. Kelimeler kök olarak isim veya sıfattan türetilebilir. Bu türetme işlemi sırasında genelde sıfat olarak kullanılan kelimeler ek almazken isimden gelen kelimelerin isimden sıfat yapım eki kullandıkları görülür. Örneğin “hızlı sürmek” kelime grubunda hızlı kelimesi sürmek fiilini , “sarı benekli kumaş” kelime grubunda ise sarı kelimesi benekli kelimesini tanımlamaktadır.

Zarf kelime grupları tanım itibariyle ya fiil, yada sıfat ile bitmelidir. Zarf kelime grupları genelde tek başlarına bir anlam ifade etmezler ancak betimledikleri sıfat veya fiil ile anlamlı hale gelirler ve genelde bu fiil veya sıfatın özelliği olarak algılanır ve sınıflandırılırlar.

#### **SORU-96: Sıfat (adjective) hakkında bilgi veriniz.**

Genel anlamda isim olarak kabul edilebilen ancak başka bir ismin bir özelliğini belirtmeye yarayan kelimelerdir. Kısaca ADJ olarak kullanılırlar. Örneğin, kırmızı, güzel veya yavaş birer sıfattır. Bu örneklerin ortak yanı kelimelerin birer özellik ifade etmeleridir. Ayrıca anlam itibariyle birer özellik belirtmeyen isimlerde ek alarak bu gruba dahil olabilir. Örneğin cam kelimesi isimdir ancak “camlı ev” tamlamasındaki cam kelimesi -lı eki alarak sıfat haline gelmiştir.

Sıfatların diğer bir kullanımı ise isimlere benzer şekilde cümlede bir fiil ifade etmesidir. Bu işlem için örneğin olmak veya -dır eki kullanılabilir. “Araba yeşildir” cümlesinde olduğu gibi. Bu durum ingilizcede bulunan yardımcı fiillerin de fiil kabul edilmesinden kaynaklanmaktadır.

#### **SORU-97: Fiil (verb) hakkında bilgi veriniz.**

Bir eylemi belirten kelimelerdir. Doğal dil işleme dünyasında daha çok V harfi olarak kısaltılırlar. Genelde bir fiilin ektisi altında kalan bir obje (mefûl) ve bu fiili yapan kişi (subje, Fâil) bulunur. Ayrıca fiilin yapıldığı zaman ve fiilin yapılış şeklini belirten belirleyicilerde bulunabilir.

Örneğin “Ali Baba kapıyı açtı.” cümlesindeki Ali Baba fâil, kapı mefûl ve açmak fiidir. Açmak fiiline eklenen -tı eki ise eylemin geçmiş bir zamanda olduğunu belirtmektedir.

#### **Köklerine göre fiiller**



Türkçede, isim köklü kelimeler de fiil olarak kullanılabilir. Bunun için kelimelerin sonuna isimden fiil yapma eki adı verilen yapım eklerinden en az birisinin gelmesi gerekir. Örneğin “Ali bahçeyi suladı” cümlesinde sulamak fiilinin kökü su kelimesidir ve su kelimesi bir isimdir.

Türkçede ayrıca isimlerinde fiil gibi kullanılması mümkündür. Örneğin “Her öğrenci bir insandır” cümlesindeki insan bir isimdir ancak -dır ekini alarak fiil gibi kullanılmıştır. Bu cümlelere isim cümleleri denilmez ancak İngilizce’de bu cümleler de birer fiil cümlesi olarak kabul edilmektedir çünkü İngilizcede bulunan “am is are” yardımcı fiilleri birer fiil olarak kabul edilmiştir. Doğal dil işleme sırasında bu farkın iki alternatifi de kullanılabilir.

### **Zamanlarına göre fiiller**

Dilbilimde zaman olarak fiiller 3 zamanda incelenir:

- Geçmiş zaman
- Şimdiki zaman
- Gelecek zaman

Ancak Türkçe zaman bakımından biraz daha farklı zamanlar sunar. Buna göre Türkçede fiil zamanları basit ve bileşik zamanlı olarak iki grupta incelenebilir. Basit fiil zamanları:

- Rivayet Geçmiş Zaman (Gelmiş)
- Hikaye Geçmiş Zaman (Geldi)
- Geniş Zaman (Gelir)
- Şimdiki Zaman (Geliyor)
- Gelecek Zaman (Gelecek)

Yukarıda da görüldüğü üzere geçmiş zaman Türkçede iki farklı grupta bulunurken bir de geniş zaman eklenmiştir.

Ayrıca Türkçede birleşik fiil zamanlarının kullanılması da mümkündür. Bu zamanlar:

- Hikaye Bileşik Zaman (Geliyor-dum, Gelecek-dim, Gelmiş-dim, Gelir-dim, Geldiy-dim)
- Rivayet Bileşik Zaman (Geliyor-muşum, Gelecek-mişim, Gelmiş-mişim, Gelir-mişim, Geldiy-mişim)
- Şart Bileşik Zaman (Geliyor-sam, Gelecek-sem, Gelmiş-sem, Gelir-sem, Geldiy-sem)
- Katmerli Bileşik Zaman (Birden fazla birleşik zaman bulunduran fiiller, Gelecek idiysem gibi)

olarak sıralanabilir. Yukarıda göstermek için mekanik olarak üretilen bu fiillerde, Hikaye geçmiş zamanın rivayeti Türkçede bulunmaz. Yani geldiymiş fiili hatalıdır.

Ayrıca Türkçede:

- Emir kipinin hikayesi
- Emir kipinin rivayeti
- Dilek-şart kipinin şartı
- İstek kipinin şartı

- Emir kipinin şartı
- Hikaye geçmiş zamanın rivayeti

şeklinde sıralayabileceğimiz birleşik zamanlı fiiller kullanılmaz ve bunların üretilmesi bir dilbilgisi hatası olarak kabul edilebilir.

HİKÂYE KİPİ	
Hikâye geçmiş zamanın hikâyesi	Geldiydim
Rivayet geçmiş zamanın hikâyesi	Gelmiştim
Şimdiki zamanın hikâyesi	Geliyordum
Gelecek zamanın hikâyesi	Gelecektim
Geniş zamanın hikâyesi	Gelirdim
Dilek-şart kipinin hikâyesi	Gelseydim
Gereklilik kipinin hikâyesi	Gelmeliydim
İstek kipinin hikâyesi	Geleydim
Emir kipinin hikâyesi	YOK
RİVAYET KİPİ	
Hikâye geçmiş zamanın hikâyesi	YOK
Rivayet geçmiş zamanın hikâyesi	Gelmişmişim
Şimdiki zamanın rivayeti	Geliyormuşum
Gelecek zamanın rivayeti	Gelecekmişim
Geniş zamanın rivayeti	Gelirmişim
Dilek-şart kipinin rivayeti	Gelseymişim
Gereklilik kipinin rivayeti	Gelmeliymişim
İstek kipinin rivayeti	Geleymişim
Emir kipinin rivayeti	YOK
ŞART KİPİ	
Hikâye geçmiş zamanın hikâyesi	Geldiysem
Rivayet geçmiş zamanın hikâyesi	Gelmişsem
Şimdiki zamanın şartı	Geliyorsam
Gelecek zamanın şartı	Geleceksem
Geniş zamanın şartı	Gelirsem
Dilek-şart kipinin şartı	YOK
Gereklilik kipinin şartı	Gelmeliysem
İstek kipinin şartı	YOK
Emir kipinin şartı	YOK

Yukarıdaki tabloda bütün zamanların ve kiplerin gelmek fiilini birinci tekil şahıs için çekimini görmek mümkündür.

## Fiil Kiplerinde Anlam Kayması

Fiil kipleri eklendikleri fiillerin anlamlarını değiştirmektedirler ve kipler sınıflandırıldıkları grup içerisindeki anlamı getirmekle mes'uldür. Ancak bazı durumlarda bir fiil kipi sorumlu olduğu anlam dışında kullanılarak farklı bir anlama gelebilir. Bu şekilde fiillerin kiplerinin anlamlarının değişmesine fiil kipinde anlam kayması ismi verilir.

Aşağıda fiil kiplerindeki kayma ve kaydıktan sonraki anlamları listelenmiştir.

- Şimdiki zaman kipinde kayma
  - Gelecek zamana kayma (Ali yarın okula gidiyor (gidecek))
  - Rivayet geçmiş zamana kayma (Ali on yıl önce Ahmete ziyarete gidiyor (gitmiş))
  - Geniş zamana kayma (Ali her gün okula gidiyor (gider))
- Geniş zaman kipinde kayma
  - Rivayet geçmiş zamana kayma (Ertesi günü Ali okula gider (gitmiş))
  - Gelecek zamana kayma (Yarın haber gelir (gelecek))
  - Şimdiki zamana kayma (Ali peşimden gelir (geliyor))
  - Emir kipine kayma (Ali çantamı getirirsin (getir))
- Gelecek zaman kipinde kayma
  - Emir kipine kayma (Yarın kitabım gelecek (getirilsin))
  - Gereklilik kipine kayma (İnsanda biraz utanma olacak (olmalı))

Yukarıdaki bu listede kiplerin anlam kaymaları gösterilmiştir.

### **SORU-98: Özel isim (hususî isim, proper noun) hakkında bilgi veriniz.**

Bir varlığa has, sadece o varlığı işaret eden isimlere verilen isimdir. Örneğin Şadi Evren ŞEKER bir insan isimdir ve sadece bir insanı ifade eder. Bir anlamda cins ismin tersi niteliğinde olarak bir çeşit veya tür değil de sadece bir bireyi temsil eden isimlerdir. Örneğin kedi cins isim olurken tekir hususi bir isimdir ve kedi cinsinden bir bireye işaret eder.

### **SORU-99: Cins İsim (common noun) hakkında bilgi veriniz.**

Bir tür, çeşit yada cinsi belirtmek için kullanılan isimlerdir. Cins isimler, isimler alanında tanımlı küme isimleri gibi düşünülebilir ve birer sınıf belirtirler. Örneğin insan, kedi, ders veya üniversite birer cins isim örnekleridir. Cins ismin özel haline özel isim (proper noun) denilir.

### **SORU-100: Soyut İsim (mücerret, abstract noun) hakkında bilgi veriniz.**

Maddî bir varlığa işaret etmeye isimlerdir. Mücerret isim olarak da nitelenen bu isimler somut isimlerin (müşahhas isimlerin) tersi olarak kabul edilir. Fiziksel olarak vâir olmayan nesneleri ifade etmek için kullanılır. Örneğin rüya, felsefe veya oyun gibi kelimeler birer soyut isimdirler.

### **SORU-101: isim (noun) hakkında bilgi veriniz.**

Bir nesneyi veya soyut bir kavramı niteleyen kelimedir. Doğal dil işleme dünyasında kısaca N harfi ile gösterilir.

### **SORU-102: Sayılabilir İsimler (count noun) hakkında bilgi veriniz.**

Sayılması mümkün isimlerdir örneğin kalem, bardak, at veya ağaç gibi isimler sayılabilir.

### **SORU-103: kütle adı (mass noun) hakkında bilgi veriniz.**

Grup ismi yada sürü ismi adı da verilmiştir. Sayılamaz varlıklara verilen isimlerdir. Örneğin kum, su, orman veya sürü gibi. İsimlerin sayılamaz veya sayı belirtmez olmaları itibarıyla bu gruba dahil olmaları gerekir. İsimlerin nitelediği varlıkların sayılamaz olması gerekmez. Örneğin ağaçlar sayılabilir ve dolayısıyla bir ormandaki ağaç sayısı bilinebilir ancak orman yine de grup ismidir çünkü sayılan orman değil ağaçlardır.

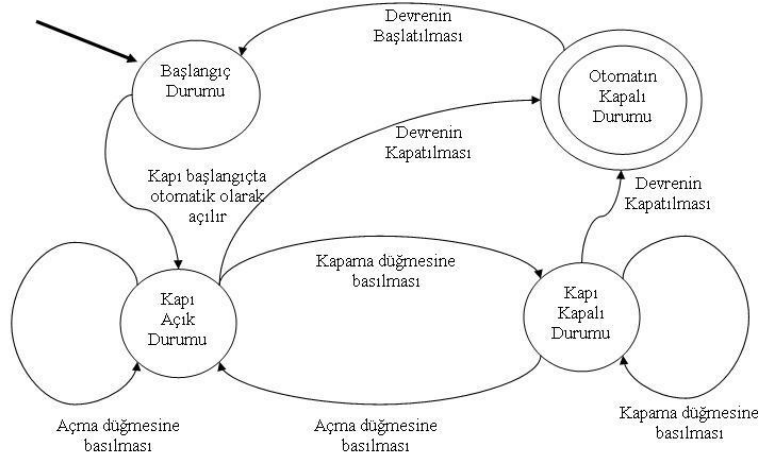
### **SORU-104: Sonlu Durum Makinası (Finite State Machine, Finite State Automaton) hakkında bilgi veriniz.**

Sonlu durum makinaları bir çizim şeklidir. Bu çizim şeklinde çeşitli durumlar ve bu durumlar arası geçiş şekilleri gösterilir. Örneğin aşağıda basit bir kapı açma ve kapama makinesi verilmiştir:



Yukarıdaki şekilde makine, açık durumdan kapalıya geçmek için kapama düğmesine basılmasını bekler. Ters durumda da kapalıdan açığa geçmek için kapama düğmesinin basılmasını bekler. Durum makinemizde kapalı durumdayken kapama düğmesine ve açık durumdayken açma düğmesine basılması bir durum değişikliği doğurmaz ve istenildiği kadar basılabilir.

Yukarıdaki şekilde bir başlangıç veya bitiş durumu belirtilmemiştir. Yani makine sonsuza kadar çalışmaktadır ve herhangi bir başlangıç koşulu yoktur. Yukarıdaki bu makinenin bir de açma ve kapama tuşları olduğunu düşünelim. Bu durumda makinenin başlangıcı açma düğmesi ve bitiş de kapama düğmesi ile olacaktır. Aşağıdaki şekilde yeni makinemizin çizimi gösterilmiştir:



Yukarıdaki şekilde sonlu durum makinemize ilave olarak başlangıç ve bitiş durumları da eklenmiştir. Buna göre kalın gidi oku başlangıç durumunu gösterir ve makinemiz buradan başlar. Çift çember içindeki durum ise bitiş durumudur ve istenirse makine burada sona erdirilir, veya çalışmaya devam edebilir. Görüldüğü üzere yukarıdaki şekilde kapı otomatı başlatıldıktan sonra istenildiği kadar açma kapama işlemi yapılmakta, gelen bu komutlara göre kapının durumu değişmektedir. Kapı otomatı kapatıldıktan sonra gelen açma ve kapama emirleri doğal olarak icra edilemez.

### **SORU-105: otomat yönelimli programlama (automata based programming) hakkında bilgi veriniz.**

otomat yönelimli programlama yaklaşımı, kaynağını otomatlar (automata)’dan alır ve sonlu durum makinaları (finite state machine, FSM) ile tasarlanan bir makinenin kodlanmasını hedefler.

Basitçe C dilindeki switch komutlarının dallanmasına benzer bir şekilde her durumdan bir sonraki duruma geçiş yapan bu programlama yaklaşımında amaç durumlar arası geçişin tasarıma uygun olarak kolay bir şekilde gerçekleşmesidir. Bunun için çeşitli dil çevirici araçlar olduğu gibi günümüz dillerinin pek çoğunda kullanılan eylem bazlı programlama (event based programming) aslında bir otomat yönelimli programlama tipidir.

Bu eylem bazlı programlama yaklaşımında yapılan, her eylem için bir alt program tanımlayarak, gerçekleşen olaylar sonucunda bu alt programlara yönlendirme yapılmasıdır. Örneğin visual basic, C++ veya JAVA gibi dillerde ekrandaki bir düğmeye (button) tıklanması durumunda bir fonksiyonun çağırılması veya javascript için onclick event, (tıklama eylemi) bu yaklaşıma birer örnektir.

### **SORU-106: üst programlama yaklaşımı (metaprogramming) hakkında bilgi veriniz.**

Üst programlama, mevcut programlama yaklaşımlarının üzerinde yeni bir yaklaşım geliştirerek programlama yapan programlama yapma anlamına gelir. Yani üst programlama ile bir program geliştirilirken, alt programlama yöntemleri harmanlanır ve kod üretilir.

Bu yaklaşımın ilginç kullanımlarından birisi de kendi kendini programlayan programların üretilmesidir. Yani üretilen kod, başka bir programa ait olmayıp bizzat üreticinin kendi parçası

olmaktadır. Her iki ihtimalde de, üst programlama ile kast edilen, programın bir program üretmesidir.

Üst programın yazıldığı dile üst dil, alt programların her birisine de nesne dil veya nesne program adı verilmektedir. üst dil ile nesne dilin aynı olması durumunda, yani kendi kendini programlama durumunda buna da yansıma (relection) adı verilir.

Günümüzde en çok kullanılan örneklerine sunucu tarafı betiklerde (server side scripting) rastlanmaktadır. Örneğin PHP ile yazılan aşağıdaki örnek kodda 10 kere ekrana “bilgisayar.kavramlari.com” basan kodu üreten örnek kod verilmiştir.

```
echo "";  
for ($i=0; $i<10; $i++) {  
echo "
```

bilgisayarkavramlari.com

```
“;  
}  
echo “”;  
?>
```

yukarıdaki kodda HTML kodunu da kapsayan satırlar üretilmiş ve PHP dili ile HTML dilinde kod üretilmiş olmuştur.

Diğer sık kullanım alanlarından birisi ise derleyici teorisinde (compiler theory) sık kullanılan araçlar olan lex ve yacc programlarıdır. Bu programlar aracılığı ile bir programlama dili üretmek oldukça basittir ve bu programları kullanacak olan kişi örneğin yacc için basitçe tanımlanmış bir parçalama ağacını (parse tree) girdi olarak vermekte bu ağaçtan ise bir C kodu üretilmektedir.

### **SORU-107: fonksiyonel programlama (functional programming) hakkında bilgi veriniz.**

Programlama yaklaşımlarından birisi olan fonksiyonel programlama günümüz dillerinin neredeyse tamamında kullanılmaktadır. Bu yaklaşımda matematik fonksiyonlarında olduğuna benzer bir şekilde alt programlar tanımlanmakta ve bu alt programların değişik argümanlar ile çalışması sağlanmaktadır. Bu yaklaşım basitçe:

- Kod tekrarını engellemekte ve aynı kodun farklı şartlar için tekrar tekrar çalışmasını sağlamaktadır
- Kodun okunabilirliğini arttırmakta ve kod analizini daha kolay hale getirmektedir.
- Programın tasarlanması aşamasında tasarımcıya modüler yaklaşım yapmasını sağlamaktadır.

Bir dilin fonksiyonel olması dilde fonksiyon veya prosedüre benzeri özellikler bulundurması ile sağlanır. Bu özellikleri tanımlamak gerekirse:

- Sıfır veya daha fazla argüman ile giriş yapılabilen
- Sıfır veya daha fazla argüman ile çıkış yapabilen
- İç yapısında dilin izin verdiği alt programları barındıran yapıdır.

Yukarıdaki bu maddeleri bir kara kutu (blackbox) yaklaşımı olarak da düşünebiliriz. Yani alt programların, dış dünya ile (programın geri kalanı ile) olan tek bağlantıları almış oldukları ve geri döndürmüş oldukları argümanlardır (parametrelerdir).

Aşağıda bir fonksiyon örneği verilmiştir: (JAVA, C , C++, C# dillerinde kabul edilir koddur):

```
int topla ( int a, int b){  
return a + b;  
}
```

Yukarıdaki kod incelendiğinde, a ve b, fonksiyona verilen argümanlardır (parametrelerdir). Dönüş değeri olarak tam sayı (integer) tipi kullanılmış ve bu durum return komudu ile belirtilmiştir. a+b eylemi ise bir alt programdır. Yani bu fonksiyon çağrıldığında icra edilen program parçasıdır.

Fonksiyonel programlama, yapısal programlamanın gerektirdiği bir yaklaşımdır. Buna göre fonksiyonel programlama kullanılan bütün diller yapısal programlama yaklaşımına uygundur denilebilir. Ancak tersi doğru değildir. Her ne kadar yapısal programlamanın tanımında bir alt programın varlığı zarurî olsa da bu alt program basit bir if bloğu olarak da düşünülebilir. Yani okuyucu kod blokları ile fonksiyonları karıştırmamalıdır.

Nesne yönelimli programlama yaklaşımınlarında fonksiyonel programlama kullanılmaktadır. Dolayısıyla her nesne yönelimli programlama yaklaşımı, fonksiyonel programlamayı barındırmaktadır. Nesne yönelimli programlama terminolojisinde, fonksiyonlara metod ismi verilmektedir.

### **SORU-108: yapısal programlama (structured programming) hakkında bilgi veriniz.**

yapısal programlama 1900lü yılların ortalarında programlama taleplerinin artması ile gelişen bir programlama felsefesidir. Buna göre programların analizi, tasarımları, kodlaması ve testleri arasındaki mantık uyumunu sağlamak amacıyla bir standarda gidilmiş ve aşağıdaki yapı çıkmıştır. Yapısal programlama amaç problemi alt parçalara bölerek bu parçaların çözümlerinin birleştirilmesidir. Bu yönüyle parçala fethet (Divide and conquer) yaklaşımı olarak kabul edilebilir.

yapısal programlamanın ortaya atılmasındaki sebepler:

1. goto komudunun karmaşıklığı: goto (atla, git veya jump) satırı bir kodun analizi, okunabilirliği ve testlerini neredeyse imkansız hale sokabilecek kadar karışmasını sağlayabilir. Bunun en basit sebebi akışların kontrol edilemez halde olmasıdır. Her ne kadar tersi iddialar da bulunsu güncel dillerin pek çoğu goto komutlarını sakıncalı bulmuş olsalar gerek bu komdu dilin doğal bir özelliği değil ama ek bir özelliği gibi barındırmaktadırlar.
2. tasarımda kullanılan yöntemlerin uyarlanma zorluğu: yapısal programlama öncesinde tam olarak ortaya atılmış ve genel kullanıma sahip, formal bir analiz ve tasarım sistemi bulunmuyordu. Yapısal programlama düşünme mantığında bir yenilik getirmesi hasebiyle tasarım ve analiz aşamalarında da güncel akış çizelgeleri (flow chart) öncülük etmiştir.

Yapısal programlama sahip bir dilde kontrol işlemleri (şartlar) aşağıdaki şekilde üçe ayrılırlar:

1. Akış (sequence) bir alt programdan diğerine geçiş işlemi. (fonksiyon veya prosedür, bkz. prosedürel programlama )
2. iki alt programdan birisini bir bool mantık işlemine göre çalıştırmak. ( if , eğer )

3. bir şart sağlanana kadar bir alt programın çalıştırılması (döngüler, loop , iteration, for, while)

yukarıdaki şartları destekleyen bir dil yapısal programlama mantığına sahiptir denilebilir. Yukarıdakilerden birisinin eksik olması yapısal programlamaya sahip olmaması için yeterlidir. Günümüzde gelişen ihtiyaçlar ile artık tek bir yaklaşıma sahip diller yoktur. Bunun yerine pek çok farklı yaklaşımlara sahip diller bulunmaktadır. Örneğin C, C++, JAVA gibi diller yapısal programlayı yukarıdaki şartları destekledikleri için barındırmaktadırlar. Ancak bu diller farklı yaklaşımları da bünyelerinde barındırmaktadır.

Aşağıda yapısal programlama yaklaşımına göre tasarım yapmayı ve bir akış çizelgesi (flow chart) çizmeyi adımlara bölmüş yaklaşım verilmiştir:

1. Yapılması istenen adımları basit bloklara böl
2. Her bloğu tek bir çıkışı olacak şekilde yeniden tasarla veya böl
3. Bu çıkış noktalarını kullanarak blokları birbirine bağla
4. Tekrarlı bloklar için döngüleri ve döngülerin koşullarını tanımla
5. Çatallanma (dallanma, fork) için şartları tanımla (if)
6. Bağlantı eksikleri bulunan blokların son bağlantılarını tamamla

Yukarıdaki adımlar tanımlandıktan sonra basit yorumlar yapılabilir. Örneğin bir değişken tanımlayarak ilk değer olarak 0 atarsak, her bloğa girişinde bir arttırsak ve her bloktan çıkışında bir azaltsak program sonunda değişkenin değerinin 0 olması beklenir.

Yapısal programlama, bir programlama yaklaşımı olup, güncel gelişmelerle birlikte kullanılmaya devam etmektedir. Örneğin nesne yönelimli programlama yaklaşımlarını kullanan dillerin neredeyse tamamı yapısal programlamayı da bünyelerinde barındırmaktadır.

### **SORU-109: çoklayıcı (multiplexer) hakkında bilgi veriniz.**

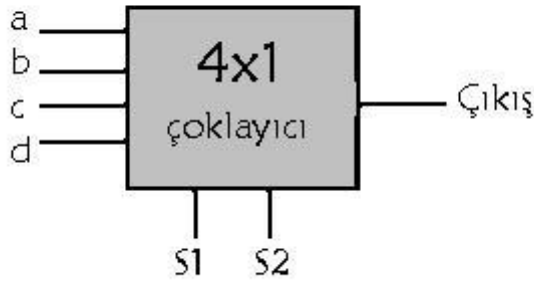
Çok sayıdaki girişin tek bir giriş üzerinden taşınmasıdır. Amaç çok sayıdaki girişin (örneğin 4 giriş) tek bir çıkışa düşürülmesidir. Çalışma mantığı, anlık olarak tek bir girişi çıkışa vermek şeklindedir. Yani 4 girişten sadece bir tanesi çıkış ile kısa devre halindedir, diğer girişler ise ihmal edilir. Hangi girişin çıkışa verileceğini belirlemek için bir seçme işlemi yapılması gerekmektedir. Bu seçme işlemini yapan bitlere seçici bit(select bit) adı verilmektedir.

Örneğin 4 girişi olan bir devre için (Girişler a,b,c ve d isminde olsun) 2 adet seçici bit gerekmektedir çünkü 4 bit ancak 2 bit ile adreslenebilir (2 üzeri 2) bu durumu gösteren temsili tablo aşağıda verilmiştir:

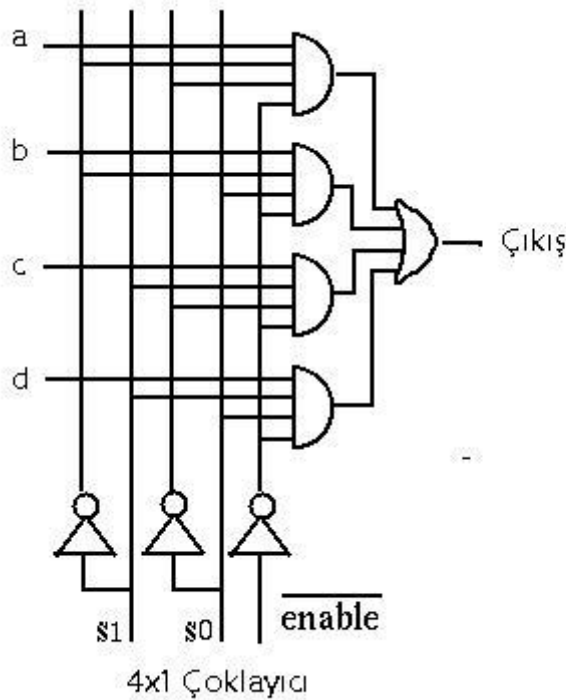
S1	S2	Ç
--	--	-
0	0	a
0	1	b
1	0	c
1	1	d

yukarıdaki tablonun çalıştığı devrenin görüntüsü aşağıda verilmiştir.





Yukarıdaki şeması verilen devrenin girişlerinden hangisini çıkış ile bağlanacağına seçici bitler karar verirler. Bu devrenin tasarımı aşağıda verilmiştir:



Yukarıda tanımı verilen özellik kullanılarak devre tasarımında kısaltmaya gidilebilir. Buna göre karnaugh haritasında verilen her hücre tasarımın sonucunda bir girişin bağlanması ile yapılmaktadır. Bu özellik kullanılarak bir tam toplayıcıyı, kod çözücü devre yardımı ile tasarlayalım:

Öncelikle tam toplayıcı devrenin çalışmasını hatırlayalım:

tam toplayıcı devrenin doğruluk çizelgesini hatırlayalım (3 bitlik giriş için 2 bit çıkışı olan ve girişlerin sayısal değerlerinin toplandığı devre idi):

A	B	C	E	T
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0

1 1 0 1 0  
1 1 1 1 1

dolayısıyla yukarıda A B ve C girişlerinin toplam değerleri T ve E bitlerinde verilmiştir.

bu doğruluk çizelgesinin karnaugh haritası aşağıda verilmiştir:

T	BC				
A		00	01	11	10
0			1		1
1	1	1		1	

E	BC				
A		00	01	11	10
0				1	
1			1	1	1

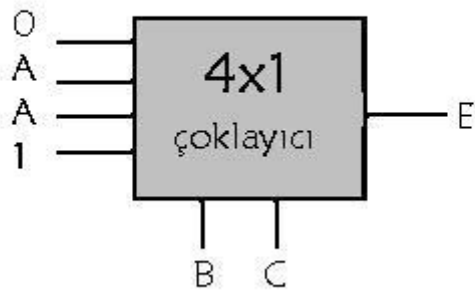
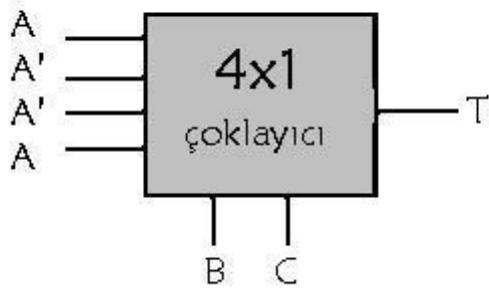
Şimdi bu karnaugh haritasının her sütunu için geçerli olan terimleri bulalım:

T	BC				
A		00	01	11	10
0			1		1
1	1	1		1	
		A	A'	A	A'

E	BC				
A		00	01	11	10
0				1	
1			1	1	1
		0	A	1	A

yukarıdaki tabloda, bir tam toplayicinin karnaugh haritasının üzerinde sütun bazlı olarak işlem yapılmıştır. Buna göre her sütunda (yain BC ikili ihtimali için) çıkan sonuç en alt satırda gösterilmiştir. Örneğin E biti için, BC ikilisi 11 olduğunda sonuç her zaman 1 çıkmaktadır (A bitinin sonuca bir etkisi yoktur) benzer şekilde yine E biti için BC ikilisi 0 olduğunda sonuç her zaman 0 olmaktadır (yine A bitinin sonuca bir etkisi yoktur) örneğin E biti için BC ikilis 01 olduğunda sonuç A bitine bağlıdır. Şayet A biti 1 ise sonuç 1 , A biti 0 ise sonuç 0 çıkmaktadır. Bu durumda da BC ikilisinin 01 olduğu durum için sonuç A'dır denilebilir. O halde yukarıda verilen bu özelliklerden faydalanarak bir çoklayıcı (multiplexer) devresi ile tam toplayıcı tasarlayalım.

Bu işlemden önce dikkat edilmesi gereken bir husu, karnaugh haritalarında sütun numaraları yazılırken 00, 01, 11, 10 sıralaması ile gitmesidir. Oysaki çoklayıcı devrenin doğruluk çizelgesine dikkat edilirse sıralama 00, 01, 10, 11 şeklinde gitmektedir. Bu yüzden karnough haritasındaki son iki sütun çoklayıcı devrede yer değiştirmektedir. Okuyucu buna dikkat etmelidir.



Yukarıdaki iki ayrı devrede iki ayrı çıkış değeri için çözüm yapılmıştır. Buna göre kod çözücü devre veya “ve” ve “veya” kapıları ile tasarlanan bir tam toplayıcı ile aynı işi yapan yukarıdaki devrede çoklayıcının bize sağlamış olduğu avantaj kullanılarak daha az elemanla tasarım yapılmıştır.

Çoklayıcı devreler günümüzde kullanılan pekçok devrenin temelinde bulunmaktadır. Örneğin ağ iletişiminde kullanılan HUB cihazının tasarımını basit bir multiplexerdir.

### **SORU-110: yarım toplayıcı (half adder) hakkında bilgi veriniz.**

ikilik tabanda verilen iki giriş değerini toplayan devredir. Buna göre A ve B girişleri için aşağıdaki tablo elde edilir. (aşağıdaki tablodaki + işareti önermeler arası veya ile karıştırılmamalıdır. + işareti toplamayı ifade eder)

A	B	E	T
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

yukarıda verilen çizelgede A ve B sayılarının toplamı E ve T değerlerinde gösterilmiştir. Örneğin A 1 ve B 1 değerleri için toplam 2 olmaktadır ( $1+1=2$ ) bu değer için ikilik tabandaki karşılığı 10'dır. 2 çıkış olmasının sebebi toplanan sayıların tek bit ile ifade edilememesindendir. Yukarıda verilen toplam değerlerini veren devreyi tasarlarlarken E ve T ikillerini (Bit) ayrı ayrı düşünmek gerekir. Bu toplama işlemini yapan devrenin tasarımında karnaugh haritalarından faydalanılırsa T ve E ikilleri (bit) için aşağıdaki haritalar çizilebilir:

T	A	
B		
	0	1
0		1
1	1	

E	A	
B		
	0	1
0		
1		1

yukarıda verilen şekilde T ve E bitlerinin çıkış değerleri karnaugh haritası üzerinde işaretlenmiştir. Komşuluk durumu olan 1 değeri olmadığı için iki bit değeri de sadeleştirilemeden aşağıdaki önermeler halinde yazılmak durumundadır:

$$T = AB' + A'B$$

$$E = AB$$

Bu devrenin tasarımı aşağıdaki şekilde verilmiştir:

yukarıdaki şekilde, A ve B girişleri için T ve E bitlerini veren örnek yarı toplayıcı şekli verilmiştir.

Şayet dikkat edilirse E kapsının sonuç değerleri yahut işlemi (“özel veya” (xor)) sonuçları ile aynıdır. Buna göre E devresinin “ve” kapısı , T devresinin ise “özel veya” (xor) kapısı olduğu görülür.

### **SORU-111: doğruluk çizelgesi (truth table) hakkında bilgi veriniz.**

Mantıksal işlemlerin tahlil edilmesinde kullanılan önemli âletlerden birisidir. Buna göre herhangi bir mantıksal önermenin (kaziye) muhtemel sonuçları bu tablo vasıtasıyla gösterilebilir. Çalışma şekli önermede (kaziye) bulunan giriş değerlerinin bütün muhtemel girişleri için bir satır oluşturmak ve sonucunu ayrı ayrı hesaplamak şeklinde yapılır. Misal olarak çok kullanılan “ve” işlemini mütâlaa edelim. Bu işlemin iki önerme için bir bağlayıcılık özelliği bulunmaktadır ve bu işlem 2 farklı önermenin (kaziye) aynı anda gerçekleşmesi durumunu doğru, diğer durumları hatalı kabul eder. Aşağıda iki farklı önerme verilmiştir:

1. üniversitede öğrenci olmak
2. devre analizi yapabilmek

bu durum aşağıdaki tabloda ve bağlacı ile ifade edilmiştir:

üöo	day	VE (And)
0	0	0
0	1	0
1	0	0
1	1	1

yukarıdaki tabloda, 1. önerme üöo (üniversitede öğrenci olmak) ve 2. önerme day (devre analizi yapabilmek) şeklinde ifade edilmiştir. Buna göre yukarıdaki tablonun ilk satırının anlamı:

üniversitede öğrenci olmak ve devre analizi yapabilmek (ikisi de 0 olduğu için), olarak yorumlanabilir. Sonuç ise 0'dır yani üniversitede öğrenci olmak ve devre analizi yapabilmek bu örnek için olumsuzdur.

Yani yukarıdaki tabloda her satırda bir ihtimal incelenmiş, neticede ise bütün ihtimaller tek bir tabloda gösterilmiştir. İşte bu tabloya doğruluk çizelgesi (truth table) denilmektedir.

Benzer şekilde aynı önermeler için “veya” işlemi incelenirse:

1. üniversitede öğrenci olmak
2. devre analizi yapabilmek

bu durum aşağıdaki tabloda veya bağlacı ile ifade edilmiştir:

üöö	day	VEYA (Or)
0	0	0
0	1	1
1	0	1
1	1	1

Yukarıda anlatılanlara göre herhangi bir önermenin doğruluk çizelgesi çıkarılabilir. Örneğin yukarıda anlatılmış olan “ve” ve “veya” bağlaçları (âtıfları) kullanılarak aşağıdaki F değerinin doğruluk çizelgesi inşa edilebilir:

$F = A'B + AB$  eşitliği için:

A	B	F
0	0	0
0	1	1
1	0	0
1	1	1

Yukarıdaki 2 girişli denklemler için oluşturulan doğruluk çizelgesi nasıl bütün olası sonuçları gösteriyorsa, aynı durum daha fazla girişi olan örnekler için de kullanılabilir. Örneğin aşağıda 3 farklı giriş için (p q r) doğruluk çizelgesi verilmiştir:

p	q	r	qr	$p+(qr)$
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	0	1
0	1	1	1	1
0	1	0	0	0
0	0	1	0	0
0	0	0	0	0

Yukarıdaki çizelgede dikkat edilecek olan husus 3 farklı giriş için toplam 8 muhtemel (2 üzeri 3) hal olmasıdır ve her hal için bir satır yazılmasıdır. Dikkat edilirse hiç bir satır diğerinin tekrarı değildir. Yukarıdaki tablo aşağıdaki şekilde de yazılabilir:

p	q	r	qr	$p+(qr)$
D	D	D	D	D
D	D	Y	Y	D
D	Y	D	Y	D
D	Y	Y	Y	D
Y	D	D	D	D
Y	D	Y	Y	Y
Y	Y	D	Y	Y

Y Y Y Y Y

Yukarıdaki tabloda D harfi doğru, Y harfi ise Yanlış sonuçları ifade etmektedir. Yani önermelerin doğru ve yanlışlığına göre sonucun nasıl olduğu bu tablodan görülebilir.

### **SORU-112: En uzun Ortak Küme (longest common subsequence, Lcs) hakkında bilgi veriniz.**

İki küme arasındaki ortak elemanların (sıralı olmak şartıyla) en uzun ortaklığını arar. Örnek:

A-> {X,M,J,Y,A,U}

B-> {M,Z,J,A,W,X,U}

olarak verilmiş olsun. Bu iki kümenin, sırası bozulmadan ortak olan en uzun alt kümesi:

LCS -> {M,J,A,U} olarak bulunur.

Bu problem karmaşıklık açısından NP-hard problemlere bir örnektir. Aynı zamanda çözüm için önerilen yöntemler incelendiğinde dinamik programlamanın anlaşılması için ideal bir örnektir.

Örnek bir çözüm yöntemini inceleyelim:

Problemin ilk akla gelen ve en basit çözümü kümelerden birisini ana küme seçerek, bu kümedeki elemanları teker teker diğer küme ile sınamak olabilir.

Örneğin A kümesi seçilmiş olsun. A kümesinin ilk elemanından son elemanına kadar ortak elemanlara bakılacaktır. Bu durumda ilk eleman X, B kümesinde aranacaktır, daha sonra X ile birlikte U aranacaktır. ihtimallerin sonuna gelindiğinde bu ikili bırakılıp yeni bir ihtimal denenecektir:

Örnek çalışma:

A kümesinden X seçilir. B kümesinde sınanır. Karşılığı bulununca LCS1 olarak kaydedilir. devamı olan harflerden uygun olan seçilir (bu örnekte U) LCS1'e eklenir.

LCS1->{X,U}

LCS2->{M,J,A,U}

LCS3->{J,A,U}

LCS4->Y'nin eşi diğer kümede yok

LCS5->{A,U}

LCS6->{U}

yukarıdaki yöntemle göre A, kümesinin bütün elemanları başlangıç elemanı olarak denenmiştir. Fakat dinamik programlama yaklaşımı incelenecek olursa bu işlemin çok daha kısa zamanda yapılabileceği görülür:

Aşağıda bir çözüm yöntemi olarak tablo kullanılması önerilmiştir. Buna göre tablodaki her hücre, ilgili satır ve sütun başlığına kadar olan en uzun ortak kümeyi tutmaktadır:

yukarıdaki şekilde örnek bir tablo verilmiştir. Buna göre verilen problemdeki en uzun ortak kümeyi bulmak hedeflenmektedir. Tablonun sağ alt köşesinde bulunan bu küme sorunun cevabıdır. Sorunun çözümünde tablo oluşturulurken her hücre kendisinin bir üstündeki veya bir solundaki hücreyi aynen kopyalamakta, ve üzerine satır ve sütun ismini şayet ortaksa yazmaktadır. Örneğin son oluşturulan sağ alt köşedeki küme, solundaki ve tepesindeki kümelerin kopyalanması ile oluşur. Bu kopyalama işlemine U elemanı ilave edilir çünkü bu hücrenin bulunduğu sütun ile satır ismi aynıdır.

Bu işlem sonunda aşağıdaki ortak küme uzunluklarını gösteren tablo elde edilir:

		0	1	2	3	4	5	6	7
			M	Z	J	A	W	X	U
-----		-----							
0			0	0	0	0	0	0	0
1	X		0	0	0	0	0	0	1
2	M		0	1	1	1	1	1	1
3	J		0	1	1	2	2	2	2
4	Y		0	1	1	2	2	2	2
5	A		0	1	1	2	3	3	3
6	U		0	1	1	2	3	3	3

Buna göre her seferinde kendinden önceki elemanlara bakılması önlenmiş olur. Yani örneğin tablonun 4,5 hüccesine bakıldığında A harfleri kontrol edilmektedir. Bu kontrol sırasında kendisinden önceki harflerin kontrolüne gerek kalmaz. (tekrar B kümesinin MZJ harflerine bakılmaz), bu sayede ortak yapılan işlemler, dinamik programlama ilkesine uygun olarak elenmiş olur.

**SORU-113: Dinamik Programlama (Dynamic programming) hakkında bilgi veriniz.**

Bir problem tahlil ve çözüm yöntemi olan dinamik programlama yapı olarak parçala fethet yöntemine benzer. Tek farkı problemi parçalara böldükten sonra aynı problemin tekrarı olan parçaları bir kerede çözüp her tekrar için ayrı bir çözüm yapmamasıdır.

Örneğin fibonacci serilerini ele alalım, Bu seriyi üreten örnek kod aşağıda verilmiştir:

```
int fibonacci(int n)
{
    if (0 == n) {
        return 0;
    } else if (1 == n) {
        return 1;
    } else {
        return fibonacci(n - 2) + fibonacci(n - 1);
    }
}
```

Yukarıda verilen bu recursive (kendi kendini çağırın) koda bakıldığında ve kodun tahlili yapıldığında aşağıdaki fonksiyon iç içe çağırma ağacı (recursion tree ) fark edilir:

```

fibonacci(4)
+-----+
|               |
| fibonacci(2)   | fibonacci(3)
| +-----+     | +-----+
| |             | |             |
| | fibonacci(0)| | fibonacci(1)| fibonacci(2)
| |           | | |             |
| |         | | | +-----+
| |       | | | |             |
| |     | | | | fibonacci(0) | fibonacci(1)

```

yani yukarıdaki örnekte, fibonacci(4) fonksiyonu için çağırma işlemleri sırasıyla gösterilmiştir. Dikkat edilirse fonksiyonlar açıldığında kendisinden önceki iki sayının toplamını bulmakta, bu işlemi yaparken de ortak elemanlar kullanmaktadır. Örneğin fibonacci(2) fonksiyonu ağacın iki farklı yerinde bulunmaktadır ve iki farklı kere içi hesaplanmıştır. İşte dinamik programlamada amaç bunu kaldırarak bir kerede çözüme ulaşmaktır.

Dinamik programlamada aşağıdaki adımlar takip edilebilir:  
Verimli bir çözüm için problemin yapsınının çıkarılması  
Kendini çağıran bir şekilde (Recursive) verimli çözüme değer atanması  
Verimli çözümün değerini aşağıdan yukarı (bottom-up) olarak hesaplanması  
Hesaplanan bu çözümle daha verimli bir çözüm varsa aranıp üretilmesi

(4. adım şayet tek çözüm varsa kullanılmaz)

#### **SORU-114: parçala fethet yöntemi (divide and conquer) hakkında bilgi veriniz.**

Bu yöntem algoritma analizinde çok kullanılan, bir algoritmayı tahlil etmek veya yeni bir algoritma oluşturmak için kullanılan yaklaşımlardan birisidir.

Bu yaklaşıma göre problem ufak ve çözülmesi nispeten daha kolay olan parçalara bölünür. Her parça ayrı ayrı çözüldükten sonra sonuçlar birleştirilerek genel problemin çözümü elde edilir.

#### **SORU-115: varlık bilim (ontoloji (ontology)) hakkında bilgi veriniz.**

ontoloji (varlıkbilim) terim olarak olmak ya da olmamak kavramı üzerinde durur. Yani birşeyin var olup olmamasından, nasıl olduğuna kadar uzanan süreç ontolojidir.

Bu anlamda, birşeyin var olup olmaması ile ilgilenen epistemolojinin üzerine, varlıkları gruplayan ve bu gruplar arası ilişkileri belirleyen bir katman olarak düşünülebilir.

Ontolojinin kökleri felsefenin bir alt konusu olan dil bilimine (yada dil felsefesine) dayanmaktadır. Buna göre kelimelerin anlamlarından yola çıkılarak, cümle ve hatta paradigmalara anlaşılmaları ve tam olarak olup olmadıkları, varsa nasıl oldukları ve hangi gruba ait oldukları ve hatta bu gruplar arası ilişkiler incelenmektedir.

Buna göre örneğin, araba, bisiklet, okul, öğrenci gibi kelimeler birer anlam ifade etmekte ve her kelimenin ait olduğu bir grup bulunmaktadır. Bu noktada iki türlü kabul vardır. Çoğunluğunu platon okulunun üyelerinin oluşturduğu grup, her kelimenin bir varlık ifade ettiğini söylerler (basit anlamda kalem, kitap gibi kelimelerin birer varlığa denk düştüğünü düşünebiliriz, daha ileri anlamda ise insanın olmayan şeylere isim koyamayacağı iddia edilebilir). diğer grup ise her kelimenin bir varlık karşılığı olamayacağını iddia etmektedirler.

İddia olunur ki, ontoloji de diğer felsefe konuları gibi insan kaynaklıdır, yani insanın düşünce yapısının bir tezahürüdür. Bu tersini ispatlamak çok zor olan ama genelde doğruluğu kabul edilebilecek bir önermedir.

Ontoloji, felsefenin pek çok alanı gibi, diğer pek çok bilime kaynak olmuş önemli konulardan birisidir. Örneğin bilgisayar bilimlerindeki ontolojik arama motorları, felsefedeki bu konudan faydalanmaktadır.



Ontolojinin dayandığı temel soru “Var olan nedir?” şeklinde özetlenebilir. Bu soru bir seviye daha ileriye götürülerek aşağıdaki sorular sorulabilir: “varlık nedir?”, “var olmak bir özellik midir?”, “birşeyin var olmadığını söylemek tam olarak ne demektir?”, “birşeyin var olduğunu veya var olmadığını iddia eden cümleler birer sav mıdır, daha fazlası mıdır?” “Nesne nedir?”, “madde nedir?”, “fiziksel olmayan kelimelerin var olduğunu söylemek ne demektir? (zaman, sayılar, ruh, Allah gibi)” “bir varlığın özelliği ne demektir? Özellikleri ile madde arasında nasıl bir ilişki vardır? Varlığın özelliğinin değişmesi ne demektir? (örneğin farabi’nin felsefesinde “şekil cevhere bulaşır” denilmesi gibi)”

Ontoloji iki kavramdan oluşmaktadır. Bu iki kavrama Çevreseller (külli, tümel, universal) ve Öz (maddenin cevheri, töz, substance) denilebilir. Örneğin “insan” bir varlıktır, bir cevherdir, “insanlık” bu varlığın külli halidir. yani insan varlığının etrafıca kapsanmış halidir yani çevresel faktörleri içerir. (benzer durum, çay ile çaydanlık arasında bulunabilir, çay varlığın özü, çaydanlık ise çevresel bir varlıktır) Cevher-külli (veya öz-tümel) ilişkisi benzer şekilde insan ve çevresi arasında da yapılmaktadır. İnsan kendi varlığını nasıl kanıtlayabilir, ve bu varlığın dışında varlıkların olduğunu nasıl ispatlayabilir? Bu durum Descartes’ın “cogito ergo sum” (düşünüyorum öyleyse varım) yorumu ile veya Fraud’un superego yorumu ile örtüşmektedir.

İşte tam bu noktada felsefenin bir konusu olan ve uzun süre felsefe insanları çevresinde tartışılmış olan ontoloji konusunun bilgisayar bilimleri ile ilişkisine bakabiliriz. Buna göre, bilgisayar bilimleri insanın öz varlığının sorgulanmasının ötesinde çevresindeki varlıkların sınıflandırılması alanında ontolojiden faydalanmaktadır. Bu sayede bir veri modeli ortaya konulmakta ve bilgisayar bilimlerinin çeşitli alanlarında (yapay zeka (artificial intelligence), anlambilimsel ağ ( the Semantic Web) , yazılım mühendisliği (software engineering), biyometikal bilim (biomedical informatics) ve bilgi mimarisi ( bilgi gösterimi (information architecture , knowledge representation) ) bu modellerden faydalanılmaktadır.

Buna göre bilgisayar bilimlerinde kullanılan temel ontolojik terimler aşağıda verilmiştir:

- \* Bireyler (Individuals) : basit en alt seviye varlıklar
- \* Sınıflar (Classes) : Kümeler, gruplar, nesne tipleri gibi
- \* Özellikler (Attributes): yapısal özellikleri, yapabildikleri, paramterleri gibi
- \* Bağlantılar (ilişkiler, Relations) : nesnelerin birbiri ile iletişim kurma şekilleri
- \* Olaylar (Events) : özellik veya bağlantılardaki değişimler

yukarıdaki bu kavramlar çerçevesinde hemen herşey modellenilebilir. Bazı örnek modellemeler aşağıda verilmiştir:

Yukarıdaki şekilde, varlıklar arası ilişkiler ontolojik (varlık bilimsel) olarak ifade edilmiştir. Örneğin bir mühendis, bir öğrenci ve bir hoca, hepsi birer insandırlar. Dolayısıyla insan olmanın gerektirdiği (boy kilo yaş gibi) özelliklere sahiptirler. Benzer şekilde, mühendislik öğrencisi de felsefe öğrencisi de birer öğrencidirler ve her ikisi de öğrenci olmanın gerektirdiği, bir hocadan ders alma ilişkisini bulundurmaktadır.

Bilgisayar Bilimlerinde kullanılan bazı ontoloji dilleri:

Geleneksel Diller (kendi modelleme sistemleri ve arayüzleri bulunmaktadır)

- \* CycL
- \* DOGMA (Developing Ontology-Grounded Methods and Applications)
- \* F-Logic (Frame Logic)
- \* KIF (Knowledge Interchange Format)
- o Ontolingua based on KIF
- \* KL-ONE
- \* KM programming language
- \* LOOM (ontology)
- \* OCML (Operational Conceptual Modelling Language)
- \* OKBC (Open Knowledge Base Connectivity)
- \* PLIB (Parts LIBrary)
- \* RACER

Berliteçli (markup) diller

Genellikle XML yapısı üzerine oturtulmuş standart bir yapısı olan dillerdir.

- \* DAML+OIL
- \* Ontology Inference Layer (OIL)
- \* Web Ontology Language (OWL)
- \* Resource Description Framework (RDF)
- \* RDF Schema
- \* SHOE