

BİLGİSAYAR KAVRAMLARI

İçindekiler

[SORU 1: Tip 1 ve Tip 2 hatalar \(Type 1 and Type 2 error rates\)](#)

[SORU 2: MIS \(Yönetim Bilişim Sistemleri\)](#)

[SORU 3: ADSL](#)

[SORU 4: Belief Propagation \(İnanç Yayılımı\)](#)

[SORU 5: Karar Ağaçları \(Decision Tree\)](#)

[SORU 6: Security Protocol Notation \(Güvenlik Teşrifat Yazımı\)](#)

[SORU 7: Normalleştirme \(Normalisation, Normalizasyon\)](#)

[SORU 8: CDMA \(code division multiple access\)](#)

[SORU 9: Jitter \(Dalga Bozulumu\)](#)

[SORU 10: Hesaplamalı Geometri \(Computational Geometry\)](#)

[SORU 11: Apriori \(Malum\), APosteriori, Afortiori](#)

[SORU 12: Document Management Systems \(Doküman Yönetim Sistemleri\)](#)

[SORU 13: Banker Algoritması \(Banker's Algorithm\)](#)

[SORU 14: Bin Packing \(Kutulama Problemi\)](#)

[SORU 15: Etraflı Arama \(Tam Arama, Exhaustive Search\)](#)

[SORU 16: Algoritma \(Algorithm\)](#)

[SORU 17: Emil Post Makinesi](#)

[SORU 18: Monte Carlo](#)

[SORU 19: Spagetti Kod \(Spaghetti Code\)](#)

[SORU 20: Merkezi poligon sayıları \(Central Polygon Numbers\)](#)

[SORU 21: Mealy ve Moore Makineleri \(Mealy and Moore Machines\)](#)

[SORU 22: Birbirini Dışlama \(Mutually Exclusive\)](#)

[SORU 23: CFS \(Completely Fair Scheduling, Tam Adil Zamanlama\)](#)

[SORU 24: Overhead \(Ek Yük\)](#)

[SORU 25: Zaman Çizelgeleme \(TimeTabling\)](#)

[SORU 26: Dolanık Kubitler \(Entangled Qubits\)](#)

[SORU 27: Dirac Göserimi \(Dirac Notation\)](#)

[SORU 28: Amdahl Kuralı \(Amdahl's Law\)](#)

[SORU 29: Bilgi Getirimi ve Çıkarımı \(Information Retrieval and Extraction\)](#)

[SORU 30: Zeki Vekiller \(Akıllı Ajanlar, Intelligent Agents, Zeki Etmenler \)](#)

[SORU 31: Tepe Tırmanma Algoritması \(Hill Climbing Algorithm\)](#)

[SORU 32: Arama Algoritmaları \(Search Algorithms\)](#)

[SORU 33: Simulated Annealing \(Benzetilmiş Tavlama\)](#)

[SORU 34: Firmware \(Bellenim\)](#)

[SORU 35: Base64](#)

[SORU 36: Moore Yasası \(Moore's Law\)](#)

[SORU 37: Cluster Computing \(Bilgisayar Kümeleri\)](#)

[SORU 38: ASCII Tablosu \(table\)](#)

[SORU 39: Çerezler \(Cookies\)](#)

[SORU 40: Rastgele Sayılar \(Random Numbers\)](#)

[SORU 41: Turing Makinesi \(Turing Machine\)](#)

[SORU 42: Özyineli Sayılabilir Diller \(Recursively Enumerable Languages\)](#)

[SORU 43: Chomsky Hiyerarşisi \(Chomsky Hierarchy \)](#)

[SORU 44: Anlamsal Ağlar \(Semantic Network\)](#)

[SORU 45: Mana Ağları \(Semantic Webs, Anlamsal Ağ\)](#)

[SORU 46: Dizgi Eş Şekilliliği \(String Homomorphism\)](#)

[SORU 47: Augmented Transition Network \(ATN, Uzatılmış Geçiş Ağı\)](#)

[SORU 48: Aks-i Müfret \(Palindrome\)](#)

[SORU 49: Atomluluk \(Atomicity\)](#)

[SORU 50: İçerikten Bağımsız Gramer \(context free grammer, CFG\)](#)

[SORU 51: İçerikten bağımsız dil \(Context Free Language, CFL\)](#)

[SORU 52: EBNF \(Uzatılmış BNF, Extended Backus Normal Form\)](#)

[SORU 53: Dolaylı sıralama \(Indirect Sort, Gayrimüstakim sıralama\)](#)

[SORU 54: Güvenlik Saldırıları \(Security Attacks\)](#)

[SORU 55: Genetik Algoritmalar \(Genetic Algorithms\)](#)

[SORU 56: Java Bean](#)

[SORU 57: Kaba Kuvvet Algoritması \(Brute Force Attack\)](#)

[SORU 58: CASE Araçları \(Case tools\)](#)

[SORU 59: En kötü durum analizi \(Worst Case Analysis\)](#)

[SORU 60: Sezgisel Algoritmalar \(Buluşsal Algoritmalar, Heuristic Algorithms\)](#)

[SORU 61: Bayes Ağları \(Bayesian Network\)](#)

[SORU 62: Linear Programming \(Doğrusal Programlama\)](#)

[SORU 63: Entropi \(Entropy, Dağınım, Dağıntı\)](#)

[SORU 64: Self Organizing Maps \(Özdüzenleyici Haritalar\)](#)

[SORU 65: K-Ortalama Algoritması \(K-Means Algorithm\)](#)

[SORU 66: HTML \(Hyper Text Markup Language\)](#)

[SORU 67: SMTP \(Simple Mail Transport Protocol\)](#)

[SORU 68: Parçalama Ağacı \(Parse Tree\)](#)

[SORU 69: Arama Motoru \(Search Engine\)](#)

[SORU 70: Web Emeklemesi \(Web Crawling\)](#)

[SORU 71: SQL \(Structured Query Language, Yapısal Sorgulama Dili\)](#)

[SORU 72: Kabuk \(Shell\)](#)

[SORU 73: Çekirdek \(Kernel\)](#)

[SORU 74: Dahili Parçalar \(Internal Fragments\)](#)

[SORU 75: Kıtlamak \(Bölütlemek, Segmentation\)](#)

[SORU 76: Harici Parçalar \(External Fragments\)](#)

[SORU 77: Yükleyici \(Loader\)](#)

[SORU 78: Hafıza Yönetimi \(Memory Management\)](#)

[SORU 79: İşletim Sistemi \(Operating System\)](#)

[SORU 80: Özellik Çıkarımı \(Feature Extraction\)](#)

[SORU 81: 2 geçişli çeviriciler \(2 pass assemblers\)](#)

[SORU 82: Şelale Modeli \(Waterfall Model \)](#)

[SORU 83: Tasarım Kalıpları \(Tasmim Kalıpları, Design Patterns\)](#)

[SORU 84: RMI \(Remote Method Invocation, Uzaktan Metod Çağırma\)](#)

[SORU 85: Kütük \(stub, nesne vekili, object Proxy\)](#)

[SORU 86: Vücubiyet \(Modality\)](#)

[SORU 87: Sayısallık \(Cardinality\)](#)

[SORU 88: ERD \(Unsur İlişki Çizimi, Entity Relationship Diagram \)](#)

[SORU 89: Unsur \(Entity\)](#)

[SORU 90: Extranet \(Dış ağ\)](#)

[SORU 91: Intranet \(İç Ağ\)](#)

[SORU 92: Çift Yönlü İletişim \(Duplex Communication\)](#)

[SORU 93: Borulama \(Pipelining\)](#)

[SORU 94: Tip İnkılabı \(Tip Dönüştürme, Type Casting\)](#)

[SORU 95: POP3](#)

[SORU 96: Yığın İş \(Batch Job, Batch Process \)](#)

[SORU 97: Kıtlık \(Starvation\)](#)

[SORU 98: KNN \(K nearest neighborhood, en yakın k komşu\)](#)

[SORU 99: Çevirici \(Assembler\)](#)

[SORU 100: Bilgisayar Mühendisliği](#)

[SORU 101: Yerleştirme Algoritmaları \(Fitting Algorithms\)](#)

[SORU 102: Patricia ağacı \(PATRICIA Tree\)](#)

[SORU 103: Dış Yol Uzunluğu \(External Path Length\)](#)

[SORU 104: İç düğüm \(Internal Nodes\)](#)

[SORU 105: Uzaysal Çözünürlük \(Spatial Resolution\)](#)

[SORU 106: Kuantum İşleme \(Quantum Computing\)](#)

[SORU 107: Kubit \(Qubit\)](#)

[SORU 108: Doğrusal Ayrılabilirlik \(Linear Seperability\)](#)

[SORU 109: Akış Diyagramı \(Flow Chart\)](#)

[SORU 110: Yapay Sinir Ağları \(Artificial Neural Networks\)](#)

[SORU 111: Nazariye \(Teori, Kuram, Theorem\)](#)

[SORU 112: Dizgi \(String\)](#)

[SORU 113: Alfabe \(Abece, Alphabet\)](#)

[SORU 114: Sembol \(Harf, İşaret, Symbol\)](#)

[SORU 115: Kenar \(Edge\)](#)

[SORU 116: Düğüm \(Node\)](#)

[SORU 117: Nesne \(Object\)](#)

[SORU 118: Sınıf \(class\)](#)

[SORU 119: Anahtar Beyazlatma \(Key Whitening\)](#)

[SORU 120: Özetleme Fonksiyonları \(Hash Function\)](#)

[SORU 121: Trie \(Metin Ağacı\)](#)

[SORU 122: İkili Arama Ağacı \(Binary Search Tree\)](#)

[SORU 123: Ağaçlar \(tree\)](#)

[SORU 124: STTL \(A standard timetabling language \(standart bir zaman çizelgeleme dili\)\)](#)

[SORU 125: Feistel Şifreleme \(Feistel Cipher, Fesitel Ağı, Feistel Network\)](#)

[SORU 126: Sıra \(Queue\)](#)

[SORU 127: MathML \(Matematiksel İşaretleme Dili, Mathematical Markup Language\)](#)

[SORU 128: Kerckhoff Prensibi](#)

[SORU 129: Bilgi, Veri, Mâlûmat, İrfan \(Knowledge, Data, Information, Wisdom\)](#)

[SORU 130: Normal Şekil \(Canonical Form\)](#)

[SORU 131: Bilgi Çıkarımı \(Information Extraction\)](#)

[SORU 132: Belirsiz Çokterimli Tam \(NP-Complete, Nondeterministic Polynomial Complete\)](#)

[SORU 133: Merkle-Hellman Şifreleme \(Merkle-Hellman Encryption\)](#)

[SORU 134: Açgözlü Yaklaşımı \(Greedy Approach\)](#)

[SORU 135: Torba Problemi \(knapsack problem\)](#)

[SORU 136: Açık Anahtarlı Şifreleme \(Public Key Cryptography\)](#)

[SORU 137: Somut \(müşahhas\) isim \(concrete noun\)](#)

SORU 1: Tip 1 ve Tip 2 hatalar (Type 1 and Type 2 error rates)

Yazının istatistik ile ilgili kısmına geçmeden önce biraz felsefe bilginizi tazeleyerek yokluk hipotezinden bahsetmek istiyorum.

Basitçe olmayana ergi (proof by contradiction, burhan-ı mütenakis) yapısında bir ispat yöntemidir. İspatlanmak istenen istatistiksel olgu bir hipotez olarak ortaya atılır ve buna alternatif hipotez (alterantive hypothesis) ismi verilir. Ardından bu alternatif hipotezin tam tersini gösteren ve yokluğu ispat edilmesi amaçlanan yokluk hipotezi (null hypothesis) konulur.

Bu durumu bir örnek üzerinden açıklayalım. Örneğin veri tabanı teorisinde sıkça verilen ve üretilebilir veri konusunda geçen iki alanı ele alalım. Diyelim ki bir kişinin yaşı ve doğum günü arasında ilişki olduğunu, bu iki değerin birbirine bağlı olduğunu iddia ediyoruz. Bu bizim alternatif hipotezimiz olmuş oluyor.

Bu durumda ilişki olmadığını göstermek de yokluk hipotezimiz olur.

Bu iki hipotezi aşağıdaki şekilde de gösterebiliriz:

Alternatif Hipotez: ***Doğum Günü Sayısı = a Yaş + b***

Burada ifade edilen değer, doğum günleri ile yaş arasında doğrusal bir bağlantı olduğudur (linear, affine). Şimdi yokluk hipotezimizi yazalım (null hypothesis)

Doğum günü sayısı artarken yaşı değişmemesi (veya tam tersi)

Yani iki artış arasında bir bağlantı olmaması.

Ardından yapılan çok sayıdaki deneme ile görüyorsunuz ki yaş arttıkça doğum günü kutlamalarının sayısı artıyor veya doğum günü kutlamalarının sayısı arttıkça yaş ilerliyor.

Ardından diyebiliriz ki yokluk hipotezi yoktur (nullify)

Şayet yokluk hipotezinin yok olduğunu gösterebilirsek (istatistiksel olarak) o zaman alternatif hipotezimizin doğru olduğunu söyleyebiliriz. Yani doğum günü kutlaması ve yaş arasında pozitif bağlantı vardır denilebilir.

Bu konu aslında felsefede pek çok konuyu beraberinde getirmektedir. Örneğin Karl Popper'ın iddialarından birisi gerçekte alternatif ve yokluk hipotezlerinin birbirinin tersi olmasının gösterilemesi zor olduğudur.

Ancak biz konumuzla ilgili olan bu kısmı ile iktifa edeceğiz. Şimdi gelelim tip 1 ve tip 2 hata miktarlarına.

Öncelikle rastgele bir süreçten bahsediyoruz demektir (random process) ve bu süreçte beklenen ve gerçekleşen olaylar bulunuyor demektir. Aslında tam olarak yokluk hipotezimizin yok olduğunu ispatlamaya çalıştığımızı düşünelim:

	Beklenen (Expected)
--	---------------------

Gerçekleşen (Measured)		Müsbet	Menfi
	Doğru	TP (DMü)	TN (DMe)
	Yanlış	FP (YMü)	FN (YMe)

Beklentimiz müspet (pozitif) veya menfi (negatif) yönde olabilir. Beklentimize bağlı olarak gerçekleşen olay (veya ölçümlerimiz) de doğru veya yanlış olabilir.

Bu durumu bir örnek üzerinden açıklayalım. Örneğin bir e-posta koruma sistemi yazıyoruz ve sistemdeki izinsiz gönderileri (spam mail) tespit etmek istiyoruz. Yazılımımız bu gönderileri tespit edip engelleyecek. Yazılımı hazırlayıp bir süre test ettikten sonra aşağıdaki gibi bir tablo hazırlanabilir.

	İstenmeyen Tahmini		
Gerçekten İstenmeyen		İstenmeyen	İstenen
	Doğru	TP (DMü)	TN (DMe)
	Yanlış	FP (YMü)	FN (YMe)

Buna göre örneğin bizim istenmeyen posta olarak sınıflandırdığımız ve gerçekte istenen postalar TP (True Positive), bizim istenen olarak sınıflandırdığımız ancak gerçekte istenmeyen olanlar TN (True Negative), bizim istenen diye sınıflandırdığımız ancak gerçekte istenmeyen olanlar FP (False Positive) ve son olarak bizim istenen diye sınıflandırdığımız ancak gerçekte istenmeyen olanlar da FN (False Negative) olarak isimlendirilebilir.

Bunlardan FN (False Negative) Tip 2 hata oranı (Type 2 error rate) ve FP (False Positive) ise Tip 1 hata (Type 1 error rate) olarak isimlendirilir.

SORU 2: MIS (Yönetim Bilişim Sistemleri)

Basitçe veri işleme katmanı olarak görülebilecek veri tabanları, veri iletişim protokolleri, veri işleyen çeşitli yazılımların tamamını birer hareket işleme sistemi (transaction processing system) olarak görmek mümkündür. Yönetim bilişim sistemleri ise bu katmanın bir seviye üzerinde, bu sistemleri kullanarak stratejik ve taktik hedefler belirleme ve izlemeye kullanılırlar. Temel olarak bir yönetim bilişim sisteminin üç bileşeni vardır ve bunlar, teknoloji, kişiler ve veriler olarak sıralanabilir.

Akademik bir tanım kapsamında, yönetim bilişim sistemleri için hedefin karar verme ve karar destek sistemleri olarak görülmesi ve bu amaç için birey, grup ve organizasyonların bilgiyi nasıl değerlendirdiği, tasarladığı, uyarladığı, yönettiği veya uyguladığı sorgulanır. Bu anlamda yönetim bilişim sistemleri, karar destek sistemleri (decision support systems), uzman sistemler (expert systems) ve yönetsel bilgi sistemleri (executive information systems) gibi farklı isimlerle anılan sistemlerin birleşimi olarak görülebilir.

Günümüzde bir eğitim alanı olarak MIS bölümleri Türkiye'de son 10 yıl içerisinde aktif olmaya başlamış ve genelde yüksek lisans seviyesindeki programlar olarak eğitime başlamıştır. Ancak dünya çapındaki pek çok saygın üniversitede lisans, yüksek lisans ve doktora seviyelerinin tamamında eğitim verilmektedir.

Ayrıca sıkça yapılan bir hata, MIS kavramını ERP (enterprise resource packages) veya bilgi teknoloji yönetimi (information technology management) kavramları ile veya teknoloji yönetimi (technology management) kavramları ile karıştırmaktır. Genel anlamda MIS sistemlerinin amacı, bütün bu ismi geçen kavramları kullanarak stratejik seviyede bir hedefe ulaşmaktır. Yani diğer ismi geçen sistemler daha alt seviyelerde farklı amaçlar için kullanılmaktadır. Örneğin bilgi teknoloji yönetimi (information technology management) kavramı, kurumun bünyesindeki bilgi teknolojilerinin çalışmaları, yönetimi ve kaynak ayrımı gibi konularla ilgilenirken bu bilgi teknolojilerinin amacı konusu dışında kalmaktadır.

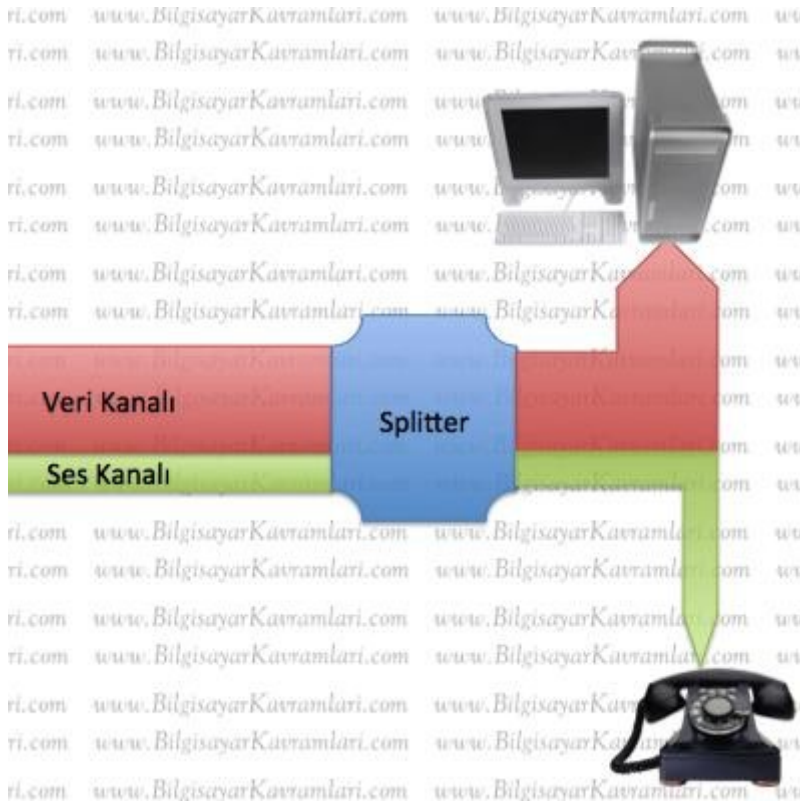
SORU 3: ADSL

ADSL kelimesi, İngilizce Asymmetric Digital Subscriber Line kelimelerinin baş harflerinden oluşmaktadır ve Türkçede asimetrik dijital üye hattı gibi bir terim ile karşılanabilir.

Teknolojinin en belirgin özelliği, giden ve gelen verilere ayrılan bant genişliklerinin simetrik olmamasıdır. Yani örneğin indirme (download) için 1Mbit hat tahsis yapılırken yükleme (upload) için 256Kbit hat tahsis yapılıyor olabilir. Bu açıdan, genelde indirme (download) ağırlıklı kullanıcılardan oluşan ev kullanıcıları için oldukça cazip bir teknolojidir.

Teknoloji basitçe bakır telefon hatları üzerinden çalışabilir. Frekans paylaşımli bir ortamda (frequency division multiplexing (FDM)) veri iletimi olduğu için, telefon hattı üzerinde, aynı anda hem telefon konuşmaları hem de veri iletimi gerçekleştirilebilmektedir. Yani veri iletişimi için ayrılan bant genişliğinin farklı frekansları veri ve ses için farklı olarak tahsis edilir. Bu durumu, aynı anda farklı frekanslardan yayın yapan radyo sinyallerine benzetmek mümkündür. Aslında herhangi bir FM alıcısı radyo, bütün frekansları o anda almaktadır ancak sadece ayarlanmış olduğu frekanstaki gelen sinyalleri sese dönüştürerek dinlememize imkan sağlar. Diğer frekansları almak istiyorsak, diğer frekanslara ayarlı farklı radyolara ihtiyaç duyarız.

Benzer durum telefon hattı üzerinden taşınan veriler için de geçerlidir. Uygun alıcı ayarı yapıldıktan sonra, hatta taşınan ses veya veri kısmı ayrıştırılabilir. Bu ayrıştırma işlemi ayırıcı (splitter) ismi verilen özel bir donanım ile yapılmaktadır.



Yukarıdaki şekilde görüldüğü üzere tek bir bakır kablo hattıyla taşınan veri ve ses hatları, splitter marifeti ile iki ayrı hatta bölünmekte ve hatlardan birisi telefona diğeri ise veri iletişimi için bilgisayara veya ADSL MODEM'e yönlendirilmektedir.

Santral tarafında, hem ses hem de veri birleştirilerek aynı hatta indirgenir. Yani evimize kadar gelen bakır kabloların aynı anda hem internet verisi hem de konuşma verisini taşıması için, ses verilerinin geldiği telefon santrali ile internet verilerinin geldiği yönlendirici (router) aynı hatta ve farklı frekanslarda birleştirilmelidir. Bu birleştirme işlemi DSLAM (Digital Subscriber Line Access Multiplexer) ismi verilen ve Türkçede “dijital abone hat erişimi çoklayıcısı” olarak çevrilebilecek bir ilave cihaz ile yapılmaktadır.



Yukarıdaki resimde, bir DSLAM sunucusu ve üzerinde bağlı olan modemler görülmektedir.

Bu işlem, genelde telefon firması tarafından telefon hattının bağlı bulunduğu santralde yapılmaktadır. Ayrıca her telefon hattı için santral kısmında ilave bir modem bulundurulmaktadır.

ADSL teknolojisi

Yukarıdaki giriş kısmından sonra teknolojinin çalışma detaylarına girebiliriz. Teknoloji basit olarak FDM veya TDM yaklaşımlarından birisini kullanır. Yani veri farklı frekans aralığından kesintisiz olarak veya farklı zamanlarda paylaşımlı olarak iletilmektedir.

Her iki teknoloji tercihi için de yükleme akışı (upstream) için ayrılan aralık, indirme akışı (downstream) için ayrılan aralıktan çok daha azdır (genelde $\frac{1}{4}$ oranında).

Örneğin frekans paylaşımı yapılan bir ortamda, annex A tipi iletişim için yükleme akışına 26,000 ile 137,825kHz arasındaki frekans bandı ayrılırken, indirme akışı için çok daha geniş bir aralık olan 138kHz ile 1104kHz arasındaki bant ayrılmaktadır.

ADSL teknolojisi, bu alanları da daha küçük parçalara bölmektedir. Yaklaşık 4.3kHz genişliğindeki bu alt kanallara terminolojide kutu anlamında “bin” ismi verilmektedir.

ADSL teknolojisi, iletişim kurma aşamasında her bini ayrı ayrı test etmekte ve SNR (signal to noise ratio, sinyal gürültü oranı) değerlerine göre bu binleri kullanıp kullanmamaya karar vermektedir. Genelde mesafeye bağlı olarak gürültünün artacağını düşünürsek, ADSL teknolojisinin neden mesafeye bağlı olarak hızının değiştiği anlaşılabilir. Yani mesafe arttıkça bazı binler daha gürültülü olduğu için kullanıma kapatılacak ve neticede de kullanılabilir bant genişliği düşecektir.

Ayrıca ADSL MODEM’ler, gönderim veya alım için bant genişliğini farklı değerlerle bu binlere dağıtabilmektedir. Örneğin bir bin’in taşıyacağı veri diğerine göre 2 veya 3 misli fazla olabilir. ADSL MODEM bu değerlere yapmış olduğu SNR testlerine göre karar vermektedir.

ADSL2+ teknolojisinde ise her binde tek bit taşıma yaklaşımı kullanılmaktadır. Bu yaklaşımda gürültülü binler hiç kullanılmaz.

Bu aşamada ADSL teknolojisi muhafazakar bir yaklaşımla bit per bin (bin başına bit) değerini düşük tutabilir. Bu yaklaşımda veri iletişiminin yavaşlaması baştan kabul edilmiş olunur ancak amaç, iletişim sırasındaki veri kaybını asgariye indirmektir. Öte yandan biraz daha cesur bir yaklaşımla daha yüksek bin başına bit değeri ile daha fazla veri transferi ve dolayısıyla daha hızlı bir iletişim hedeflenebilir. Ancak bu durumda verinin kaybolma riski daha da artacaktır. İşte bu durumda daha üstte çalışan TCP/IP gibi protokoller paket kayıpları yaşayacağından verinin tekrar ve tekrar yollanması yüzünden hızın yine yavaşlaması söz konusu olabilecektir. Burada iki uç arasında dengeli bir seçim yapılması en hızlı çözümü getirecektir.

ADSL2+ teknolojisi burada dikişsiz oran uyumu (seamless rate adaptation (SRA)) ismi verilen bir çözüm önermektedir. Bu çözüme göre iki taraf arasındaki bin başına bit haberleşmesi çok daha az iletişim ile çözülebilmektedir. Yani ADSL teknolojisi üzerinden iletişim halinde olan taraflar (ev kullanıcısı ve ADSL sunucusu) ilgili bin başına ayrılan bit değerini değiştirmeye karar verdiklerinde bu bilginin iki taraf tarafından da bilinmesi gerekmekte ve bu bilginin taraflar arasında taşınması çok daha az haberleşme aşamasında sağlanmaktadır.

ADSL teknolojisini iyileştirmek için denenen yollardan birisi de, ADSL için ayrılmış olan özel frekans aralıklarının ötesindeki frekansların kullanılmasıdır. Bu yaklaşımda birinci

problem, iki tarafta da (ADSL kullanıcısı tarafında da) özel bir cihaz bulunması ve bu yüksek frekans değerlerini algılaması gerekliliğidir. Bu ilave bir maliyet getirir.

Ayrıca ADSL verisinin telefon hatları üzerinden taşındığı unutulmamalıdır. Dolayısıyla çoğu yerde birbirine yakın geçen telefon hatları üzerinden yüksek frekans değerlerinde veri iletimi, çoğu zaman çarpaz konuşma (crosstalk) ismi verilen ve bir kablodaki veri iletişimi sırasında oluşan manyetik alanın diğer hatta etkilemesi olarak anlaşılabilecek problemin doğmasına sebep olur.

SORU 4: Belief Propagation (İnanç Yayılımı)

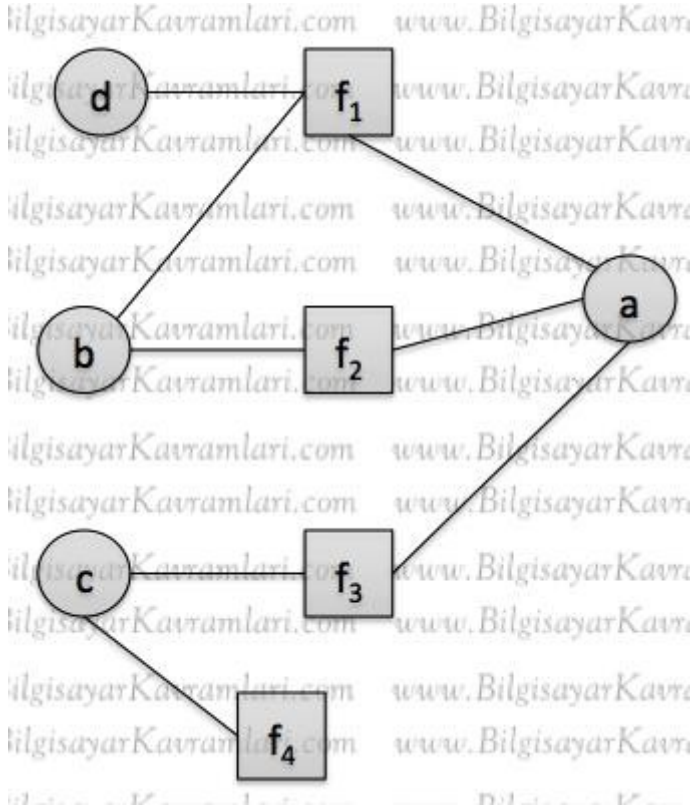
Türkçede inanç yayılması (veya iman neşri) olarak çevrilebilecek belief propagation konusu, bilgisayar bilimlerinde, makine öğrenmesi (machine learning) konusunun altında değerlendirilebilir.

Algoritma ilk olarak Judea Pearl tarafından 1982 yılında yayınlanan makalesinde duyurulmuştur. Pearl, Judea (1982). “Reverend Bayes on inference engines: A distributed hierarchical approach”. *Proceedings of the Second National Conference on Artificial Intelligence*. AAAI-82: Pittsburgh, PA. Menlo Park, California: AAAI Press. pp.133–136.)

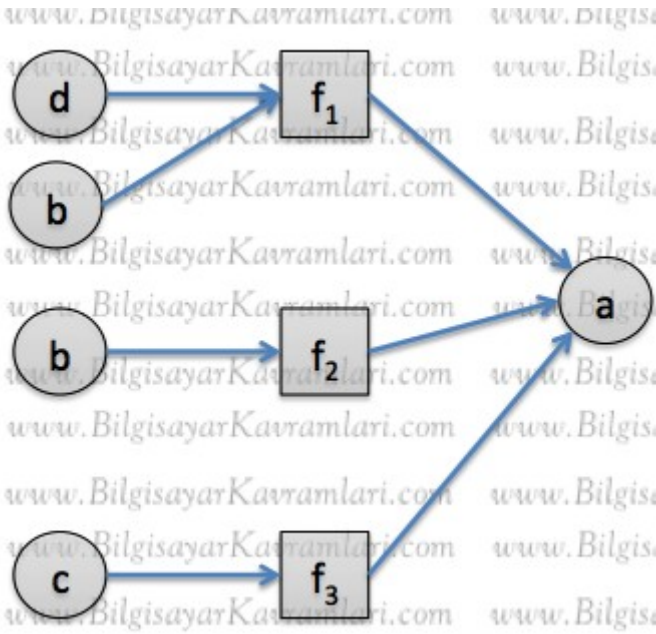
Yapı olarak mesaj geçirme (message passing) algoritmalarının bir örneği olarak görülebilir. Buradaki yayılma (neşr etmek, propagation) aslında varlıklar arasında bir mesajın geçişi anlamını taşımaktadır. Literatürde farklı kaynaklarda, toplam-çarpım mesaj geçirimi (sum-product message passing) olarak da geçmektedir. Buradaki mesaj geçişi yapılan varlıklar genelde bir şekil (Graph) üzerinde ifade edilen ve aralarında ilişkiler bulunan varlıklardır.

Esas itibarıyla Markov Rastgele Alanları (markov random fields) üzerine kurulu olan inanç yayılımı konusunu bir şekil (graph) üzerindeki varlıkların birbirlerine belirli oranlarla mesaj geçirdikleri bir alan sunmaktadır.

İnanç neşriyatı için öncelikle bir problemin bir ağaç şeklinde nasıl çözüldüğüne bakalım. Öncelikle bir çarpım şekli (factor graph) ele alıyoruz:



Bu şekilde, diyelim ki a parametresi üzerine inanç neşriyatı uygulayacağız. Bu durumda yukarıdaki şekli, aşağıdaki gibi bir ağaca (tree) dönüştürmek mümkündür:

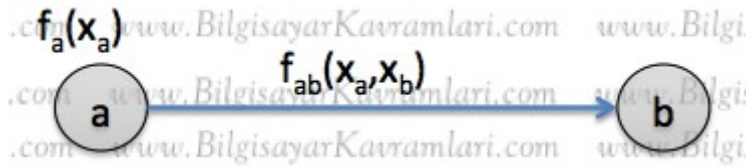


Yukarıdaki yeni şekilde, dikkat edileceği üzere, bir doğrusal ve dairesel olmayan şekil (directed acyclic graph) elde edilmiştir. Yani şekil, yönsüz şekilden (undirected graph) bir yönlü şekle (directed graph) çevrilmiştir. Bu anlamda şekli bir ağaç (tree) şeklinde ele almak mümkündür. Ayrıca şekilde b parametresi, iki ayrı fonksiyona parametre olduğu için kopyalanarak gösterilmiştir (aslında buna gerek yoktur ancak sadece konu anlaşılсын diye böyle bir yol izledim). Ayrıca sonuca etkisi olmayan f3 fonksiyonu sistemden çıkarılmıştır.

Netice olarak şekildeki 3 fonksiyonun sonuçları a parametresine birer etki olarak taşınacaktır (mesaj geçişi).

Bu anlamda, inanç neşriyatı problemini bir ağaç şeklinde düşünmek ve verilen parametreye göre problemi alt problemlere bölmek mümkündür. Hatta bu alt problemlerin birbirinden bağımsız (independent) olduğunu da söyleyebiliriz.

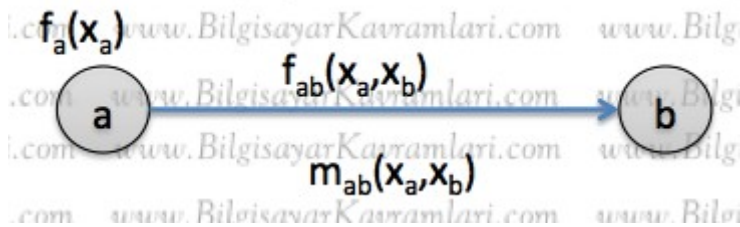
Yukarıdaki ağaç yaklaşımı üzerinden inanç neşriyatını (belief propagation) açıklamak istersek. Öncelikle iki düğüm (node) ve bir kenar (edge) üzerinde yaklaşımın nasıl çalıştığını göstermemiz gerekir:



Burada anlatılmak istenen, ağ üzerinde varlık gösteren her düğümün sonucunu döndüren bir fonksiyon ve her ilişkinin (kenar) tanımlı olduğu düğümleri aynı anda parametre olarak alan bir fonksiyonun varlığıdır. Yukarıda gösterilmemiştir ancak yine $f(b)$ şeklinde b değerini parametre alan bir fonksiyonun varlığından da bahsedilebilir.

Aslında kenarlar üzerinde ağırlık tanımlanması halinde (ki benzer durumlar yapay sinir ağı (artificial neural network) çalışmalarında veya bayez ağlarında (bayesian network) kullanılmaktadır) bu ağırlık da bir fonksiyon olarak düşünülebilir. Örneğin $f(a,b) = 0.2$ gibi.

Yukardaki gösterimi biraz daha ilerletecek ve işin içerisine bir de mesaj ekleyecek olursak:

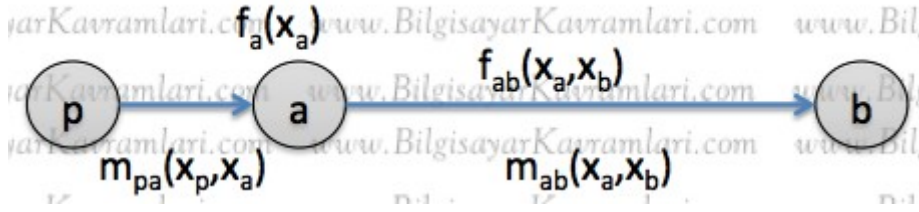


Yukarıdaki yeni şekilde eklenen m değeri, a'dan b'ye geçirilen mesajı ifade etmektedir.

Bu mesaj değeri aşağıdaki şekilde hesaplanabilir:

$$m_{ab}(x_b) = k \sum_{x_a} f_{ab}(x_a, x_b) f_a(x_a) \prod_{p \in N(a)/b} m_{pa}(x_a), \text{ olarak yazılabilir}$$

Buradaki gösterimde kabaca tanımlı olan f fonksiyonlarından yararlanılmış ve şekildeki gösterim bir toplam-çarpım algoritması (sum-product algorithm) olarak ele alınmıştır. Hesaplama kullanılan geçici p değeri, yukarıdaki a düğümüne etki eden herhangi bir p hesaplamasını ifade etmektedir. Yani a düğümüne kadar gelen etkiler çarpım sembolü ile hesaplanmıştır. Bu durum aşağıdaki şekilde düşünülebilir:



Yani denklemin çarpım sembolü kısmında bu p düğümünün etkisi ile a'ya bağlanan kenardan gelen fonksiyon değerinin a üzerindeki etkisi alınmıştır. Bu şekilde, a'ya etki eden bütün düğüm ve kenar bağlantıları (ki a'ya bağlı sadece p gösterilmiştir ancak başkaları da olabilir) a üzerinde çarpım sembolü ile ifade edildikten sonra geriye a gibi b üzerinde etkisi bulunan bütün düğümlerin toplamalarını hesaplamak kalmıştır. Denklemin toplam sembolü ile gösterilen kısmı da bu toplamayı yapmaktadır.

Yukarıdaki yaklaşımı toplamdan çıkarıp azami değeri alacak şekle getirirsek:

$$m_{ab}(x_b) = k \max_{x_{ab}} f(x_a, x_b) f_a(x_a) \prod_{p \in N(a)/b} m_{pa}(x_a), \text{ olarak yazılabilir}$$

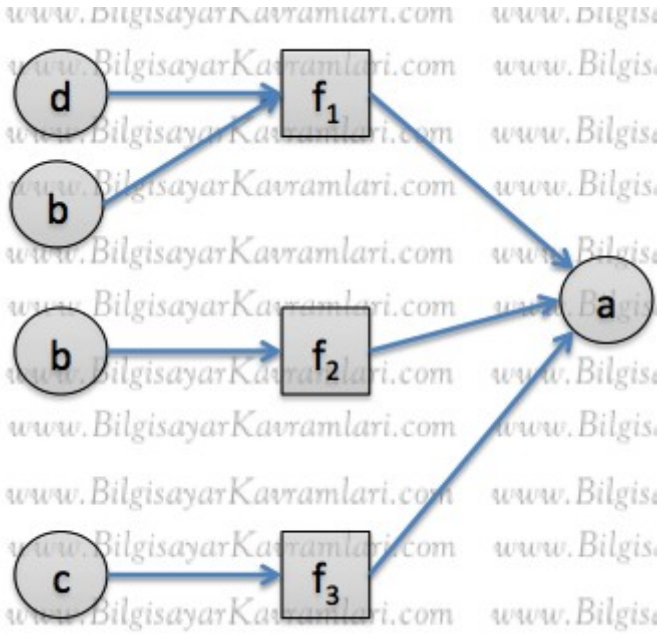
Yeni yaklaşımımızda azami değerler alınarak parametrelerin arasında en büyük değeri yani en aykırı değeri (marginal) bulmak amaçlanmıştır. Bu değerlere aykırı değer veya azami-aykırı değer (marginal veya max-marginal) ismi verilir. Kısaca aşağıdaki şekilde de gösterilebilir:

$$p_a(x_a) = k f_a(x_a) \prod_{p \in N(a)} m_{pa}(x_a), \text{ olarak yazılabilir}$$

Bu anlamda, önce çarpım sembolünü işletmek (iterate) ve bütün değişkenler için bir bir çalıştırmak mümkündür. Her çalışan değişken için aslında sistemden bir değişkenin kaldırıldığını söyleyebiliriz.

Paralel Mesaj Geçirimi

Bu parametre kaldırma işlemini paralel hale getirmek de mümkündür. Yazının başında verilen şekli hatırlayalım:



Bu şekilde bulunan f_2 ve f_3 fonksiyonları veya f_1 ve f_3 fonksiyonları, birbirinden bağımsız olarak hesaplanabilir. f_1 ve f_2 fonksiyonları ise aynı değişkene (b) bağlı oldukları için bazı durumlarda birbirini beklemek zorundadır.

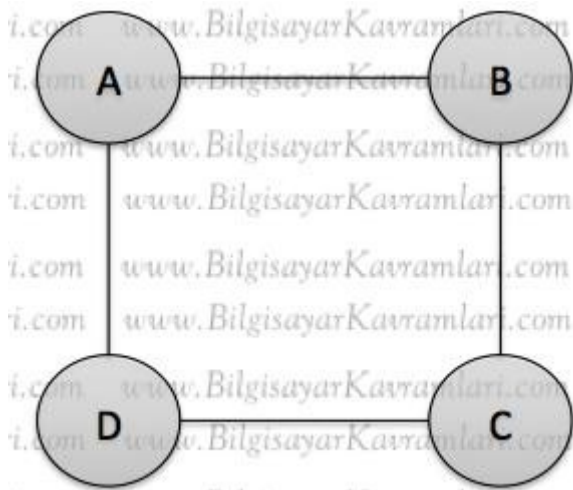
Bu paralel işleme işine de paralel mesaj geçirme (parallel message passing) ismi verilir.

Döngüsel İnanç Neşriyatı (Loopy Belief Propagation)

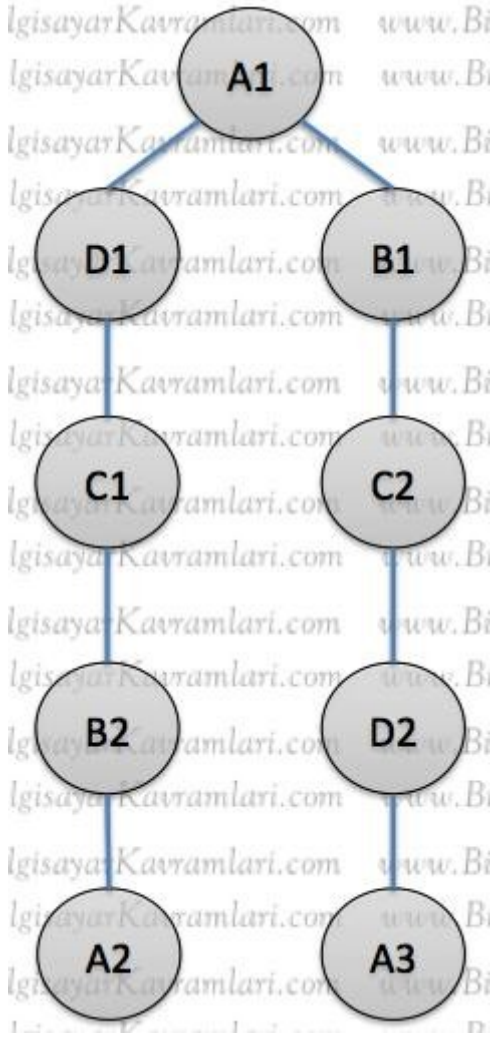
Gelelim döngüsel inanç neşriyatına (döngüsel inanç yayılımı). Bu yaklaşımda, şekilde bir dairesel (cyclic) özellik olacağı kabulü bulunur. Ancak döngüsel olarak yapılan yaklaşımın her zaman bir noktada toplanması/birleşmesi garanti edilemez.

Hatta bir noktada toplanması mümkün olsa bile, doğru marjinal değerlerde toplanması garanti edilemez.

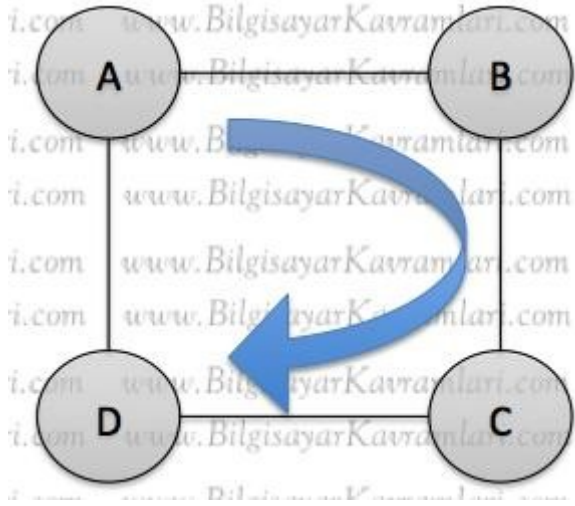
Örneğin tek döngü içeren aşağıdaki şekli ele alalım:



Bu markof rastgele alanından ařağıdaki sargısız řekli (unwrapped graph) elde etmek mmkndr.



Dolayısıyla yukarıdaki yaklaşım kullanılarak mesaj gncellemesi iki farklı ynde iřletilebilir. Buradaki ama inan ağıının, dngsel bir markof alanında alıřtırılmasıdır. alıřmanın bařarılı olması iin yukarıda gsterildiğı gibi sargısız řekil (unwrapped graph) elde edilip zerinde alıřılabileceğı gibi, orjinal markof rastgele alanında da bir yn belirlenerek alıřılabilir:



Örneğin yukarıdaki ağaç şekili çıkarılırken, A düğümünden başlanarak saat yönünde dönülen bir kol (ağacın sağ kolu) ve bu yönün tam tersi istikamette dönülen diğer bir kol (ağacın sol kolu) çizilmiştir.

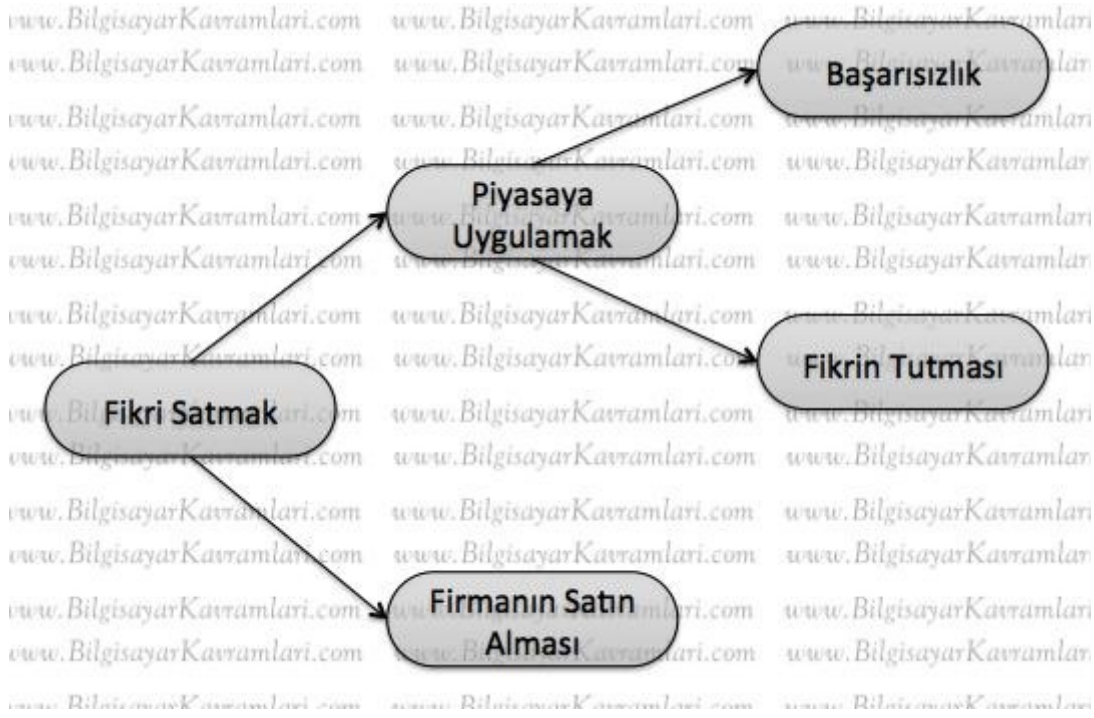
Yukarıdaki şekilde bir döngü içeren inanç ağı, kararlı hale (steady state) ulaşana kadar parametre güncellemesi ile işlenir. Yani ağda yayılım (neşriyat, propagation) sürekli devam eder. Neticede kararlı bir hal alır ve durulur.

SORU 5: Karar Ağaçları (Decision Tree)

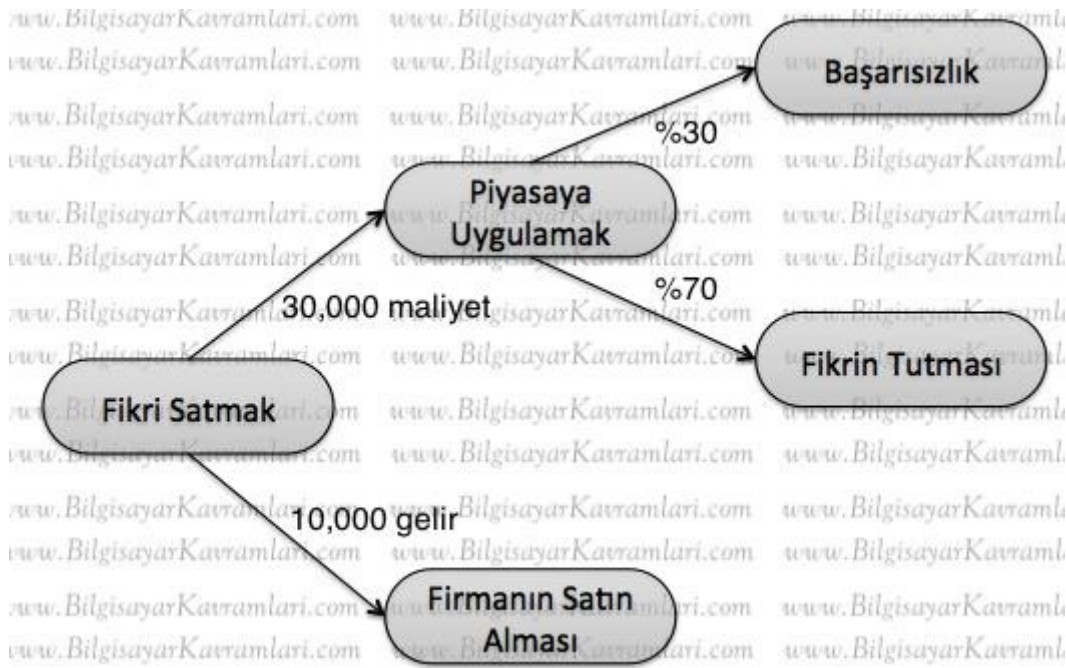
Genelde bir karar varmak için çeşitli adımlardan oluşan sistemlerde bu adımların olasılıksal değerlerini bir çizge (şekil, graph) üzerinde ağaç (tree) kullanarak göstermek için kullanılırlar.

Örneğin piyasada bize avantaj sağlayacak bir fikrimiz olduğunu düşünüyoruz ve bu fikri uygulayıp uygulamamak konusunda kararsızız. Diyelim ki fikrimizin başarı oranını %70 olarak görüyoruz. Ayrıca fikrimizi herhangi bir yatırımcıya satarak 10bin lira gelir elde edeceğimizi tahmin ediyoruz. Ayrıca fikrimizi piyasada uygulamak için 30bin lira sermayeye ihtiyacımız olsun ve başarılı olması halinde 100bin lira gelir elde etme ihtimalimiz olsun. Bu senaryoda nasıl bir karar vermemiz gerekir?

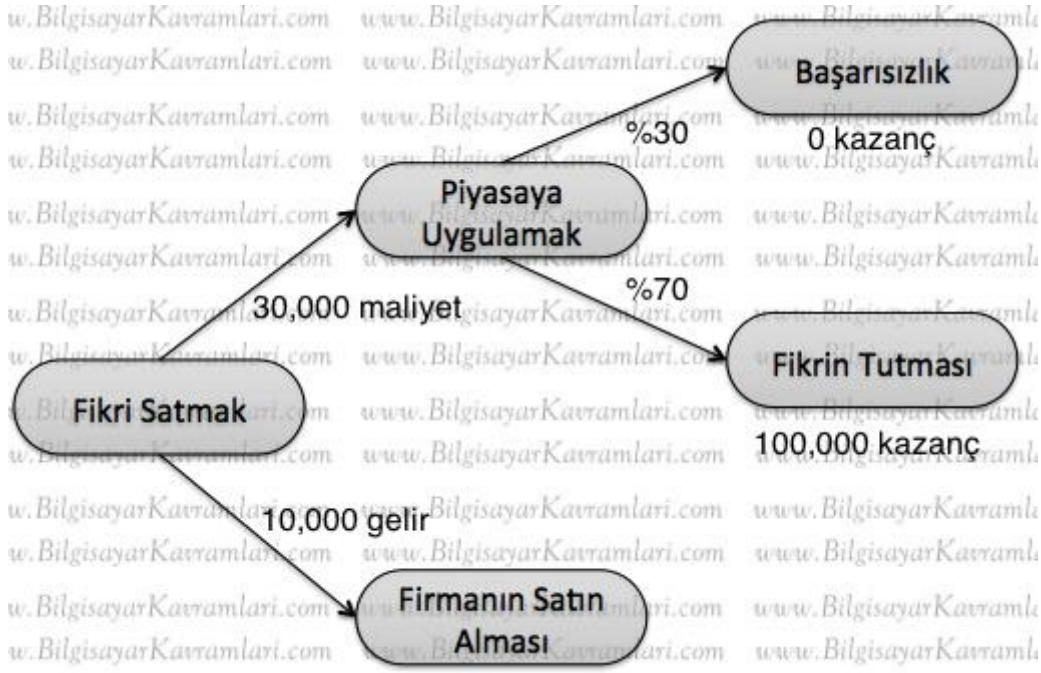
Aşağıda, bu durumu modelleyen bir karar ağacı verilmiştir.



Yukarıdaki şekilde görüldüğü üzere iki karar vermemiz söz konusu, fikrimizi ya bir firmaya satacağız ya da uygulayacağız. Ayrıca uygularsak yine iki ihtimal bulunuyor, fikrimiz ya başarılı olur ya da olmaz. Yukarıdaki bu ağaca ihtimallerin yerleştirildiği durum aşağıda gösterilmiştir:



Ayrıca fikrimizin tutması durumundaki kazancımızı ve başarısızlık durumundaki kaybımızı da ağaca ekleyebiliriz.



Son olarak ağaçta bulunan durumları ilk karar anına kadar taşımamız mümkün. Örneğin %70 ihtimalle fikrin tutması halinde 100,000 lira kazancımız olması durumu, aslında piyasaya uygulama fikrinin:

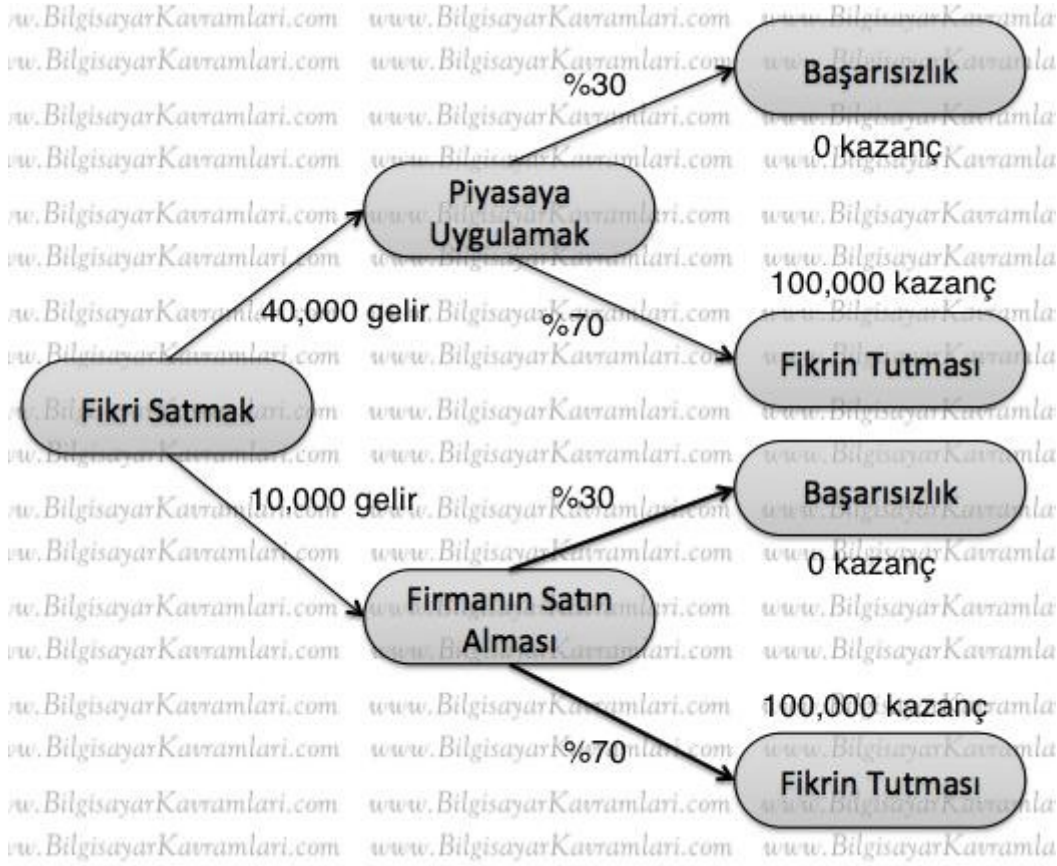
$$100,000 \times \%70 + 0 \times \%30$$

olarak formüllendirilebilecek bir gelir olduğunu gösterir. Ayrıca bu fikrin uygulanmasının 30,000 lira maliyeti olduğunu düşünürsek formülü aşağıdaki şekilde tamamlayabiliriz:

$$100,000 \times \%70 + 0 \times \%30 - 30,000$$

Sonuç olarak fikri uygulamanın bize tahmini kazancı 40,000 olarak hesaplanır. Dolayısıyla başlangıç adımındaki karar anında, fikri uygularsak riskler hesaplandıktan sonra 40,000 lira gelir elde ederken fikrimizi sattığımızda 10,000 lira gelir elde etmiş olacağız. Bu açıdan fikri uygulamaya geçirmek daha mantıklı görülmektedir.

Ayrıca firmanın fikri uygulama durumundaki gelirini de hesaplayabiliriz:

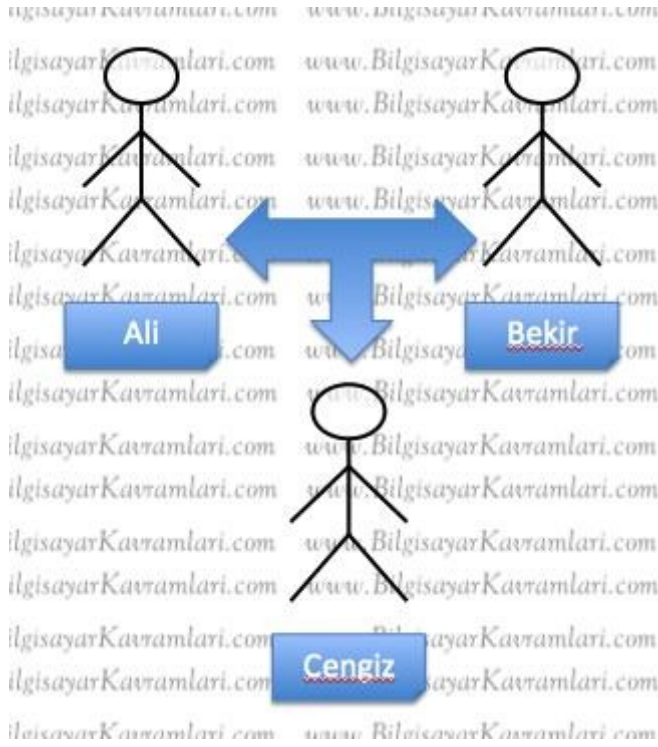


Ağacın son durumundaki ihtimaller hesaplandığında, firmanın da fikrimizi satın alması halinde 40,000 lira gelir elde edeceğini hesaplayabiliriz. Ayrıca firma fikrimizi satın aldığı için ilave olarak 10,000 lira ödeyeceğine göre firmanın bu fikri satın alıp uygulaması durumunda tahmini olarak 30,000 lira kar elde etmesi beklenir.

SORU 6: Security Protocol Notation (Güvenlik Teşrifat Yazımı)

Pek çok algoritmanın / protokolün (teşrifat) çalışmasını anlatırken standart bir gösterime gidilmiş ve bazı kurallar konulmuştur. Bu kurallara göre algoritmaları yazmak ve okumak daha kolay hale gelir.

Bu kurallara göre, öncelikle iletişim kuran iki taraf kabulü yapılır. Bu taraflar genelde alfabenin ilk iki harfi olan A ve B harfleri ile gösterilir. Ayrıca güvenlik ile ilgili kitaplarda okumayı kolaylaştırmak için Alice ve Bob isimleri kullanılmaktadır. Genelde bu iki kişinin iletişimine saldıran üçüncü kişi ise C harfi ile başlayan Charlie olarak okunmaktadır. Bu yazı kapsamında Türkçe içerik sunulduğu için bu harflere karşılık Türkçedeki Ali, Bekir ve Cengiz isimleri kullanılacaktır. Kısacası temel bir iletişim yapısı aşğıdaki şekilde resmedilebilir.



Yukarıdaki şekilde görüldüğü üzere, Ali ve Bekir arasındaki iletişime Cengiz bir şekilde dahil olabilmektedir. Bu saldırının çok sayıda çeşidi olup detayları bu yazının konusu dışındadır. Bu yazı kapsamında bir protokol anlatılırken, Ali, Bekir ve Cengiz'in yapabileceklerini göstermenin kuralları açıklanacaktır.

Kullanılan kısaltmalar:

- A : Ali (mesaj gönderen taraflardan birisi)
- B : Bekir (mesaj gönderen diğer taraf veya çoğu durumda mesajı alan taraf)
- C : Cengiz (iletişime saldıran taraf)
- K : Anahtar (ingilizcedeki Key kelimesinden gelir)
- Ka : Ali'nin anahtarı (yanındaki harf sahibini gösterir)
- PK : Umumi anahtar (ingilizcedeki Public Key kelimelerinden gelir)
- PKa : Ali'nin umumi anahtarı (bazı kaynaklarda açık anahtar olarak da geçer)
- T : Zaman bilgisi (ingilizcedeki Timestamp kelimesinden gelir ve içinde zaman bilgisi taşıyan bir mesajdır)
- Na : Nonce (yanındaki harf kimin nonce bilgisi olduğunu anlatır, örnekte Ali'nin nonce bilgisi)
- X : Açık mesaj (plain text)

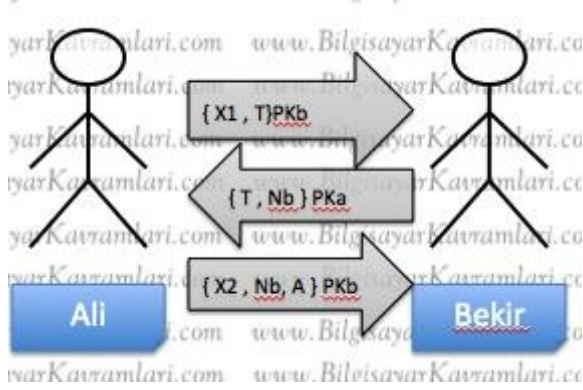
Yukarıdaki kısaltmaları kullanarak örneğin aşağıdaki teşrifatı (protocol) okuyalım:

1. $A \rightarrow B : \{ X1, T \} PKb$
2. $B \rightarrow A : \{ T, Nb \} PKa$
3. $A \rightarrow B : \{ X2, Nb, A \} PKb$

Yukarıdaki gösterimde, 1. adımda, Ali, Bekir'e 1 numaralı açık mesajı göndermiş, gönderirken bu mesajın yanına bir zaman bilgisi ekleyerek Bekir'in umumi anahtarı ile şifrelemiştir.

2. adımda, Bekir, Ali'ye cevap olarak anlık bir zaman bilgisi oluşturmuş ve yanına kendi oluşturduğu nonce bilgisini eklemiş bu bilgileri Ali'nin umumi anahtarı ile şifrelemiştir.

Son adımda ise Ali, Bekir'e mesajın ikinci kısmını ve Bekir'in nonce bilgisini, kendisini tanıtan bir A bilgisi ile birlikte yollamıştır.



Şekilde gösterilen mesajlaşma aslında teşrifat yazımında kullanılan gösterimin görsel halidir.

Yukarıdaki örnekte, örneğin 3. adımda bulunan Nb bilgisinin, aslında 2. adımda alınan ve Bekir'in Ali'ye yolladığı nonce bilgisi olduğunu biliyoruz, çünkü nonce bilgisi ilgili kişi tarafından üretilebilir. Ali, Nb üretemeyeceğine göre, 3. adımda yollanan Nb'nin alınmasının tek yolu 2. adımda ulaşan Nb'nin bu adımda geri yollanmasıdır. Bu tip yollamaların sebebi genelde mesajın yolda bir saldırı sonucu bozulup bozulmadığını anlamaktır.

SORU 7: Normalleştirme (Normalisation, Normalizasyon)

Genel olarak çok sayıda sınav sorusunda normalleştirme için hayali bazı tablolar sunulur ve bu tabloların normalleştirilmesi (normalizasyonu) istenir.

Örnek olarak aşağıdaki tabloyu ele alalım.

A	B	C	D
1	1	1	2
1	2	1	3
2	2	1	4
2	1	1	5

Yukarıdaki tabloda görüldüğü üzere 4 adet kolon ve her kolonda çeşitli sayılar ile gösterilmiş değerler bulunmaktadır. Bu tablonun üzerinde normalleştirme yapmaya başlamadan önce bazı konuları açıklayalım.

Örneğin yukarıdaki tabloda bir aday anahtar (candidate key) bulmamız istense ne yaparız?

Bir aday anahtar bulma işlemi için en kolay yol, tablonun birincil anahtarını (primary key) bulmaktan geçer. O halde sorumuzu öncelikle tablonun birincil anahtarını (primary key) bulmak üzere değiştirelim:

Yukarıdaki tabloda, satır bazlı olarak tekrar etmeyen bir kolon var mıdır?

Cevap : D kolonudur. Dikkat edilirse D kolonu, her satırda farklı bir değer almıştır. Demek ki tek bir kolonun birincil anahtar (primary key) olmasını istersek, D kolonunu seçmemiz yerinde olur.

Peki birden fazla kolon alınması durumunda hangi kolonları seçebiliriz?

Cevap: D kolonunun zaten her satırda tekrarsız olduğunu (unique) biliyoruz. Dolayısıyla D kolonu ile birlikte hangi kolon alınır alınsın bu kolonlar da tekrarsız (unique) olacaktır.

Ancak acaba D kolonunu almadan bir asil anahtar (primary key) bulunabilir mi?

Cevap: Evet vardır. Örneğin (A,B) ikilisi de tablodaki her satırda tek başına tekrarsız (unique) olma özelliğindedir. Bir önceki cevabımızdan çıkardığımız üzere, (A,B) ikilisi ile birlikte alınacak diğer bir kolon da (örneğin C) bu durumda birincil anahtar (primary key) olma özelliğini taşıyacaktır.

Yukarıdaki cevapların doğru olmasına karşılık, birincil anahtar (primary key) seçimi yapılırken en az sayıda kolonu içeren alternatifin seçilmesi yerinde olur. Bu anlamda, yukarıdaki anahtar ihtimallerinin hepsi birer aday anahtar (candidate key) olarak değerlendirilebilir ancak birincil anahtar olarak D kolonun tek başına seçilmesi yerinde olur.

Gelelim bir diğer soruya:

Yukarıdaki tabloya, aşağıdaki şekilde ilave bir kolon eklediğimizi düşünelim:

A	B	C	D	E
1	1	1	2	X
1	2	1	3	Y
2	2	1	4	Y
2	1	1	5	X

Yeni tablomuzda bulunan fonksiyonel bağılıkları (functional dependency) çıkarmaya çalışalım:

Öncelikle, biliyoruz ki D kolonu birincil anahtardır (primary key) dolayısıyla zaten D kolonu bütün kolonların fonksiyonel olarak bağımlı olduğu bir kolondur ve aşağıdaki satırların tamamı doğrudur:

$D \rightarrow A$, $D \rightarrow B$, $D \rightarrow C$, $D \rightarrow E$

Ayrıca bir dipnot olarak, her kolonun kendisine fonksiyonel bağılı olduğunu söyleyebiliriz (self functional dependency), dolayısıyla $D \twoheadrightarrow D$ ifadesi de doğrudur ancak bu ifade, genelde veritabanı normalleştirilmesinde bir anlam ifade etmediğinden göz ardı edilir.

Gelelim diğer fonksiyonel bağımlılıklara. Tabloda ikili üçlü ve dördü fonksiyonel bağımlılıklar bulunabilir. Örneğin yukarıdaki sorularda (A,B) çiftinin de bir aday anahtar olduğundan bahsetmiştik. Bu durumda $(A,B) \twoheadrightarrow C$, $(A,B) \twoheadrightarrow D$, $(A,B) \twoheadrightarrow E$ ifadeleri de doğru olacaktır. Hatta D ile birlikte herhangi bir kolonun alınması da doğrudur. Örneğin $(C,D) \twoheadrightarrow A$, $(C,D) \twoheadrightarrow B$, $(C,D) \twoheadrightarrow E$ ifadeleri de doğrudur. Benzer şekilde D bir aday anahtar, birden fazla kolon birleşimini de fonksiyonel olarak ifade eder. Örneğin $D \twoheadrightarrow (A,B,C)$ veya $D \twoheadrightarrow (B,C,E)$ veya $(A,B) \twoheadrightarrow (C,D,E)$ ifadelerinin tamamı da doğru kabul edilebilir.

Ancak acaba tek kolon seviyesinde başka fonksiyonel bağımlılık bulunabilir mi? Bu sorunun cevabı aslında bir kolon değişirken diğer bir kolonun değerlerinin bu değişimi yansıtmayı yansıtmadığıdır.

Örneğin yukarıdaki tabloda, B ve E kolonları birlikte değişmektedir. Diğer bir deyişle $B(1) \twoheadrightarrow E(X)$ ve $B(2) \twoheadrightarrow E(Y)$ bağlantısı bulunmaktadır. Yani B tablosundaki 1 ile E tablosundaki X ve B tablosundaki 2 ile E tablosundaki Y arasında bir birliktelik söz konusudur.

Bu anlamda, $B \twoheadrightarrow E$ ve $E \twoheadrightarrow B$ ifadeleri doğrudur. Ancak konunun daha iyi anlaşılması için tabloda ufak bir oynama yapalım:

A	B	C	D	E
1	1	1	2	X
1	2	1	3	Y
2	2	1	4	Y
2	3	1	5	X

Yukarıdaki yeni tabloda, B kolonunun son satırı 3 olarak değiştirilmiştir. Bu durumda acaba $B \twoheadrightarrow E$ ve $E \twoheadrightarrow B$ ifadelerinden bahsedilebilir mi?

Tablonun yeni halinde E'deki her değişim, B'de bir değişimle karşılandığı için $B \twoheadrightarrow E$ ibaresi doğrudur ancak ne yazık ki $E \twoheadrightarrow B$ ibaresi kullanılamaz çünkü E, B'deki değişimleri göstermek için yetersizdir.

Ayrıca yukarıdaki tabloda, C kolonu, bütün diğer kolonlara fonksiyonel olarak bağımlıdır. Bunun sebebi C kolonunun sabit olması ve dolayısıyla bütün kolonlar tarafından doğası gereği fonksiyonel bağımlılığının ifade edilmesinin mümkün olmasıdır.

Gelelim tablomuzu normalleştirmeye. Tablomuzda bulunan fonksiyonel bağımlılıkları normalleştirme aşamasında kullanacağız. Normalleştirme işleminin amacını kısaca tabloda tekrar eden veri bırakmamak (veya en aza indirmek) olduğunu söyleyebiliriz.

Yukarıdaki son halini almış tablomuzda, $B \twoheadrightarrow E$ bağımlılığının bulunması bize aşağıdaki durumu oluşturma imkanı sağlar:

A	B	C	D
1	1	1	2
1	2	1	3
2	2	1	4
2	3	1	5

Tablo K

B	E
1	X
2	Y
3	X

Tablo M

Görüldüğü üzere, B kolonu ve E kolonu ayrı bir tabloda tutularak, E kolonunun B kolonuna fonksiyonel bağımlılığından yararlanılmıştır. Bu durumda E kolonunun orjinal tabloda yer almasına gerek yoktur. A,B,C veya D kolonları ile birlikte karşılığı olan E kolonunu bir kişinin sorgulamak istemesi halinde, iki tablo arasında bir birleştirme (join) işlemi uygulanacak ve istenen veriye kolaylıkla ulaşılabilecektir. Bu durumda K tablosundaki B kolonu bir yabancı anahtar (foreign key) olmuş ve M tablosundaki B kolonu ise birincil anahtar (primary key) olmuştur denilebilir.

Gelelim C kolonuna. Bu kolon da herhangi bir kolona fonksiyonel bağımlı kabul edilebilir demiştik. O halde aşağıdaki şekilde bölünebilir:

A	B	D
1	1	2
1	2	3
2	2	4
2	3	5

Tablo K

B	C	E
1	1	X
2	1	Y
3	1	X

Tablo M

Yukarıdaki gösterim doğru olmasına karşılık, aşağıdaki gösterim de doğrudur:

A	B	D
1	1	2
1	2	3
2	2	4
2	3	5

Tablo K

B	E
1	X
2	Y
3	X

Tablo M

A	C
1	1
2	1

Tablo N

Veya diğerk bir çözüm olarak aşağıdaki çözüm de doğrudur:

A	B	D
1	1	2
1	2	3
2	2	4
2	3	5

B	E
1	X
2	Y
3	X

E	C
X	1
Y	1

Tablo K

Tablo M

Tablo P

Yukarıdaki bütün çözümler doğrudur. Görüldüğü üzere, normalleştirme işleminde, tekrar eden satırlar elenmiş ve mümkün olduğunca tekrarsız satırların bırakılması amaçlanmıştır.

Burada bir soru, $D \rightarrow A$ özelliğini neden kullanmadık şeklinde sorulabilir. Yani D kolonu zaten A kolonunu veya B kolonunu belirlemektedir, o halde bu özelliği de kullanarak bir bölme işlemine daha gidilebilir mi?

Cevabı hem evet hem de hayırdır. Teorik olarak bahsedildiği gibi bir bölme olabilir. Ancak bu bölmenin hiçbir faydası olmaz. Aşağıda göstermeye çalışalım:

A	D
1	2
1	3
2	4
2	5

B	D
1	2
2	3
2	4
3	5

Tablo K Tablo Q

Bir önceki şekilde bulunan tablo K'yı tablo K ve tablo Q olarak bölmeye çalıştım. Görüldüğü üzere herhangi bir satır sayısında azalma olmamıştır. Bu tip birincil anahtar kullanarak bölme işlemleri mümkün olmakla birlikte genelde normalleştirme anlamında bir fayda sağlamaz. Bu şekilde böldükten sonra, iki tablo arasında (örneğimizdeki tablo K ve Q) sayısalılık açısından (cardinality) birebir (one-to-one) ilişki kurulmaktadır ki teorik olarak iki tablo arasında birebir ilişki varsa aslında bu iki tablo, bir tablonun ikiye bölünmüş halidir denilebilir. Genelde birebir ilişki, veritabanında hız amacıyla kullanılan (bazı kolonlara istatistiksel olarak çok nadir erişim yapıldığı biliniyorsa) veya veritabanı kısıtlarından dolayı kullanılan (örneğin veri

tabanımızın tablo başına sadece 10 kolon tutmaya izin verdiği durumda 15 kolonlu bir tablo oluşturmak için) bir özelliktir.

SORU 8: CDMA (code division multiple access)

Bilgisayar bilimlerinde, özellikle ağ (network) konusunda geçen ve bir ortamı, birden fazla veri kanalının iletişimi için kullanılan yöntemlerden birisidir. Literatürde sıkça geçen diğer çok kanallı veri iletişim yöntemleri, TDMA (time division multiple access , zaman paylaşımli çoklu erişim) ve FDMA (frequency division multiple access, frekans paylaşımli çoklu erişim) yöntemleridir.

CDMA yöntemini bu diğer meşhur iki yöntem ile karşılaştırmak için genelde şu şekilde bir örnek verilir. Örneğin bir odada birden çok kişinin konuşarak haberleştiğini düşünelim. TDM yaklaşımında, kişiler sırayla ve teker teker konuşmakta, ilgili alıcı konuşan kişinin mesajını almaktadır. FDM yaklaşımında, kişiler farklı ses tonları ile konuşmakta ve dolayısıyla alıcı olan kişi, ilgili ses tonuna dikkatini vererek iletilen mesajı almaktadır. CDMA yaklaşımında ise, kişiler farklı lisanlarda konuşmakta, dolayısıyla o lisanı bilen kişiler tarafından algılanmakta, diğer kişiler tarafından iletilen veri gürültü olarak algılanıp dikkate alınmamaktadır.

Örnek

Konuyu bir örnek üzerinden açıklamaya çalışalım. Örneğin 4 farklı veri kanalı üzerinden veri akmakta olsun ve bunları CDMA yöntemi ile tek bir kanaldan taşımak isteyelim.

- V1: 1101
- V2: 0010
- V3: 1010
- V4: 0011

Yukarıdaki şekilde verilen 4 farklı verinin CDMA ile nasıl taşındığını anlatalım. Verileri ilk adımda farklı frekans değerine sahip işaretler ile kodluyoruz (code). Örneğimizde kullanacağımız 4 farklı kodumuz aşağıdaki şekilde olsun:

- K1: 1111
- K2: 1010
- K3: 1100
- K4: 1001

Verilerin, kodlar tarafından işlenebilmesi için ve 4 farklı verimiz olduğu için, verilerin genliğini 4 misli şeklinde düünebiliriz. Buna göre örnek olarak son veri için kodlamayı anlatalım:

V4 : 0000 0000 1111 1111 (Gösterim için 0011 verisinin her elemanını 4 kere tekrarladım.)

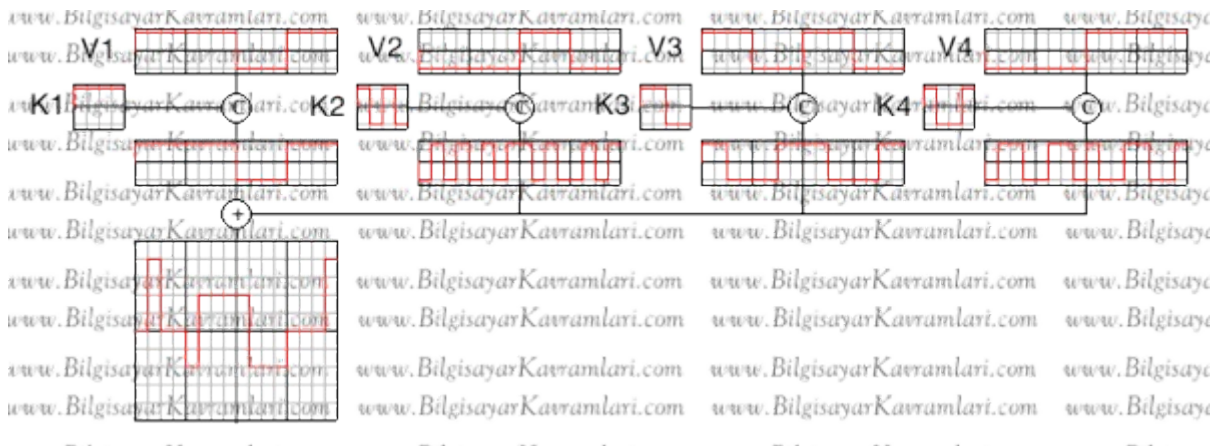
K(V4,K4) : 0110 0110 1001 1001 (V4'ün, K4 ile kodlanması sonucunda, V4 üzerindeki 1 değerleri için K4'ün kendisi, V4 üzerindeki 0 değerleri için ise K4'ün tersi gelmektedir. Daha basit anlamda her V4 dörtlüsü (uzun şekilde yazılmış halini düşünün) ile K4 değerlerinin özel

veyasının (XOR) tersi alınır !((0000 0000 1111 1111) XOR (1001 1001 1001 1001)) şeklinde)

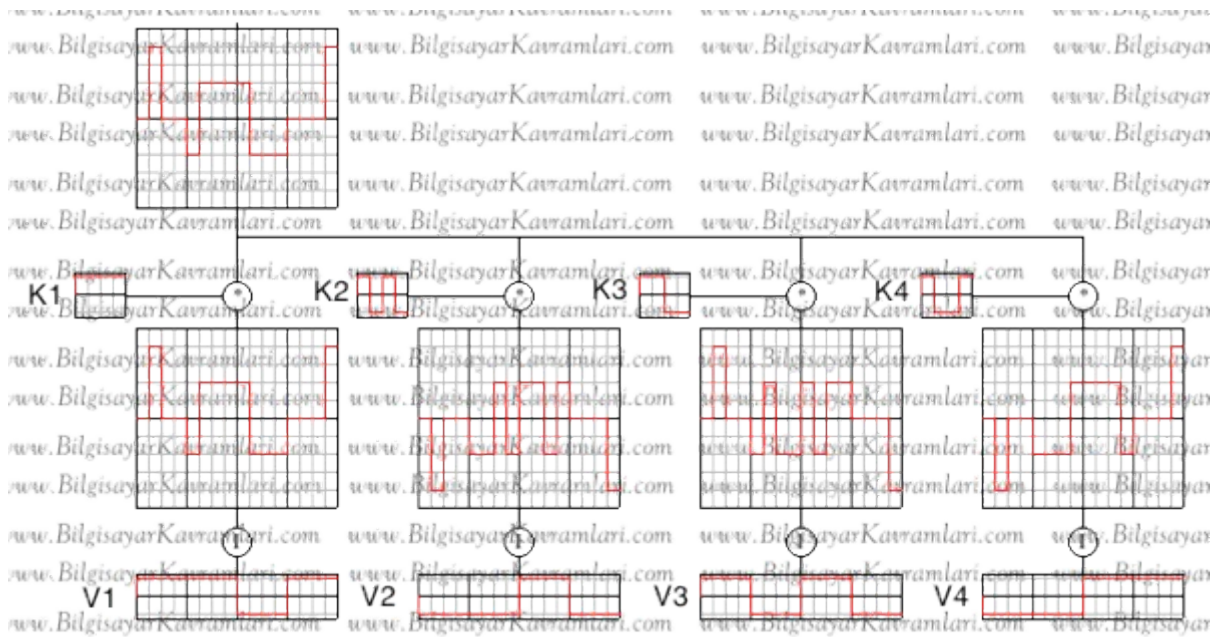
Sonuçta 4 farklı veri ve 4 farklı kodlama için aşağıdaki sonuçlara ulaşılır:

- K(V1,K1) : 1111 1111 0000 1111
- K(V2,K2) : 0101 0101 1010 0101
- K(V3,K3) : 1100 0011 1100 0011
- K(V4,K4) : 0110 0110 1001 1001

CDMA algoritmamızda, son adım olarak yukarıdaki değerleri topluyoruz. Toplamın ve yukarıdaki işlemlerin görsel olarak ifadesi aşağıdaki şekildedir:



Yukarıdaki toplama işlemi sonucunda elde edilen verilerin, her birisinin farklı alıcılar tarafından alınmak istediğini düşünelim. Bu durumda her alıcı, almak istediği göndericinin kodlama değerini kendisinde ayarlayacak ve yukarıda elde edilen sonuç verisini kendisinde işleyecektir. Bu durum aşağıdaki şekilde gösterilmektedir:



Yukarıda görüldüğü üzere her kodlama değeri sonucunda açılan veri, orjinal olarak kodlanan ilgili veridir. Örneğin K1 kodlamasından açılan veri V1 olarak bulunmuştur. Bu işlem diğer kodlamaları engellememektedir.

CDMA yöntemi, günümüzde de kullanılan UMTS teknolojisinin temelini oluşturur. UMTS (universal mobile telecommunication system, evrensel hareketli telekomunikasyon sistemi) teknolojisi, CDMA2000 teknolojisinden sonra (IMT Multi Carrier, inter mobile telecommunications, hareketli telekomunikasyonlar arası çoklu taşıyıcı olarak da bilinir) geliştirilen ve CDMA 2000 teknolojisi ile rekabeti amaçlayan bir teknolojidir. CDMA2000 de, UMTS'in temeli olan W-CDMA de birer 3G teknolojisidir ve cep telefonlarının aynı anda iletişimi için kullanılmaktadır.

SORU 9: Jitter (Dalga Bozulumu)

Genelde sinyal işleme konularında geçen bir terim olan jitter (dalga bozulumu), bilgisayar bilimlerinde, ağ (networking), çoklu ortam uygulamaları (multi media) veya resim işleme (image processing) gibi konularda geçmektedir. Jitter kavramı, kısaca bir sinyalin olması gereken değere göre hatalı dalga değeri vermesidir.



Örneğin yukarıdaki şekilde bir dijital sinyal görülmektedir (resmin üstünde). Bu sinyalin bozulmuş hali resmin ortasında ve bozulmadan kaynaklanan jitter değeri resmin ortasında gösterilmiştir.

Dalgada yaşanan sürekli bir bozulma olmasından dolayı, jitter terimi, faz bozulması veya faz gürültüsü (phase noise) olarak da tanımlanabilir. Dalganın anlık bir noktasında yaşanan gürültüden, bu anlamda farklıdır.

Sinyalde yaşanan ve sürekli olan bu bozulmanın da bir periyodundan (veya frekansından) bahsedilebilir. Dalga bozulumu frekansı (jitter frequency), bu tanıma göre, dalgada yaşanan bozulmaların en büyük değerleri arasındaki mesafedir. Diğer bir deyişle, yukarıdaki şekilde görülen ve bozulum yaşanan dalgaların frekansıdır. Dalga frekansı hesaplanırken, en büyük değerler arasındaki fark alınabileceği gibi en küçük değerler arasındaki fark da alınabilir.

Dalga bozulumunun yaşandığı yere göre farklı isimlendirmelerin kullanılması mümkündür.

Sarnıçlama Dalga Bozulumu (Sampling Jitter): Bu kavram, genelde işaret (sinyal) üzerinde uygulanan çevirimler sırasında ortaya çıkar. DAC (digital to analog converter, dijital verinin analog veriye çevirimi) veya tersi olan ADC (analog to digital converter, analog verinin dijital veriye çevirimi) işlemleri belirli bir zaman almaktadır. O halde sinyal işlenirken, beklenen zamana göre gecikmeli olarak sonuç elde edilecek ve nihayetinde bir dalga bozulumu yaşanacaktır.

Örneğin sarnıçlama yapılan (belirli aralıklarla örnekler alınan, sampling) bir sistemin, ses, ışık veya hız gibi sürekli (continuous) bir işaret (signal) olduğunu kabul edelim. Bu işaretin belirli zamanlarda değerinin okunarak dijital ortama çevirimi, burada bahsedilen gecikmeler ve kaymalara neticede de sarnıçlama dalga bozulumuna sebep olacaktır.

Paket Dalga Bozulumu: Bilgisayar ağlarında, bazı durumlarda, paketlerin belirli sıklıkta (frequency) iletilmesi beklenir. Bu sıklığın bozulması da bir dalga bozulumu (jitter) olarak kabul edilebilir. Bilgisayar ağlarındaki dalga bozulumu (jitter) aslında başlı başına bir hizmet kalitesi (quality of service) konusudur ve daha çok kabul gören PDV (packet delay variation) terimi altında kullanılmaktadır.

Yukarıda verilen örnekler daha da arttırılabilir. Örneğin bir CD-ROM'dan okuma sırasında, CD üzerindeki verinin aranması sırasında geçen süre, herhangi bir veri transfer yazılımı veya devresinin, veriyi göndermeye başlamasında geçen süre, kuantum kapılarının (quantum gates), elektron dönüşünden kaynaklanan (spin based) çalışma gecikmesi veya aktarılacak istenen verinin kanal kapasitesinin çok üzerinde olmasından dolayı, verinin bir kısmının tıkanıklık (congestion) ile karşılaşması ve bu yüzden beklenen zamandan daha geç transfer edilmesi gibi durumlar birer dalga bozulumu (jitter) örneğidir.

SORU 10: Hesaplamalı Geometri (Computational Geometry)

Esas itibarıyla mühendis kelimesinin kökü olan, hendese kelimesini büyük ölçüde karşılayan geometri kelimesinin başına hesaplamalı gibi bir kelime getirmek anlamlı değildir. Çünkü zaten geometri hesaplamalı bir çalışma alanıdır. Buradaki hesaplamalı kelimesi daha çok bilgisayar ile işlenen anlamını taşımaktadır. Yani hesaplamalı geometri ile kasıt (computational geometry) bilgisayar ile işlenen geometri çalışmalarıdır. İngilizcede kullanılan computational kelimesi bu anlamda, bilgisayarı (computer) çağırması açısından daha doğru bir tercihtir.

Geometri ve bilgisayar çalışmalarının kesişim noktası olan hesaplamalı geometri konusuna bilgisayar bilimleri açısından bakıldığında, geometrik problemleri çözen algoritmaların geliştirilmesi ve iyileştirilmesi (optimisation) olarak düşünülebilir.

Hesaplamalı geometri çalışmalarının ilk çıkışı, bilgisayar grafikleri konusunda yapılan çalışmalardır. Örneğin bir bilgisayar animasyonu veya bilgisayar destekli çizim tasarım ve üretim yazılımları (CAD / CAM) hesaplamalı geometrinin doğuşuna ve gelişmesine öncülük etmiştir.

Günümüzde, ayrıca robot çalışmaları, coğrafi bilgi sistemleri (geographic information systems, GIS) veya entegre devre tasarımı (integrated circuit , IC) gibi konularda da hesaplamalı geometriden yoğun olarak istifade edilmektedir.

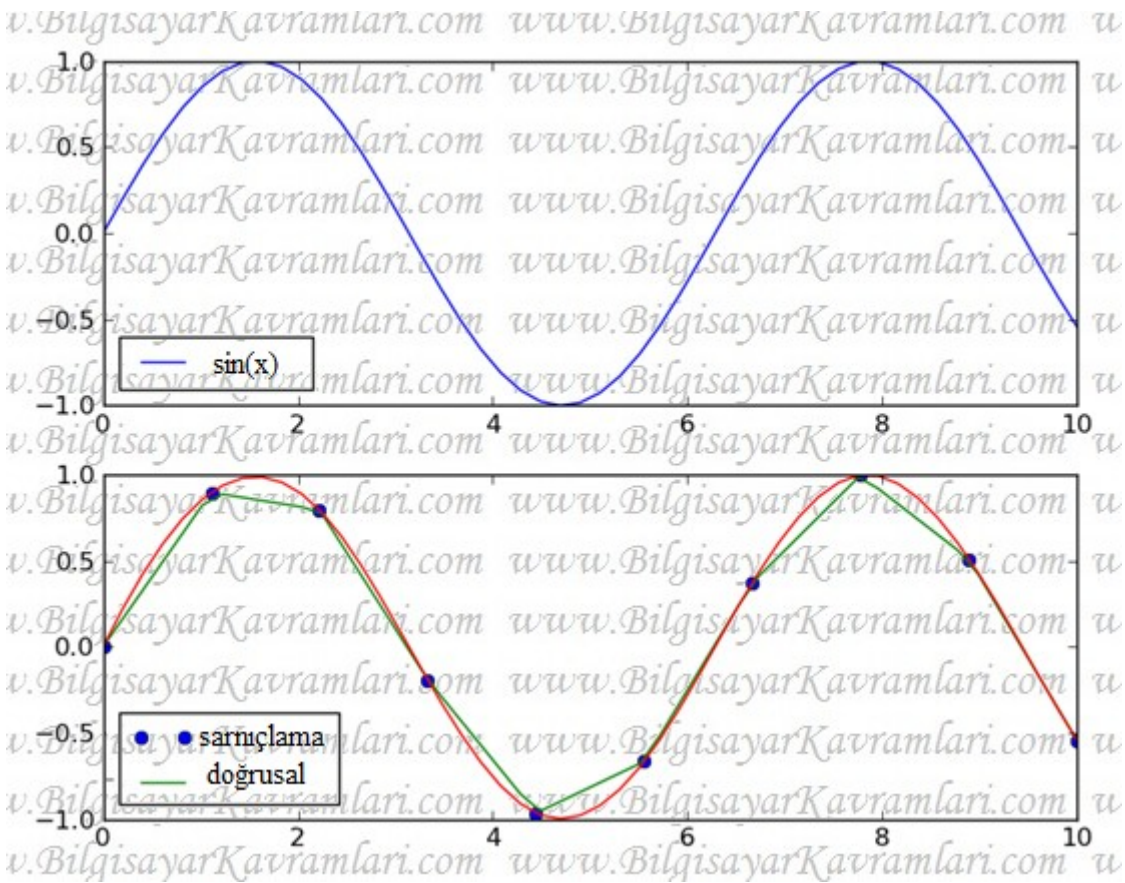
Yapı olarak çoğu bilgisayar bilimleri çalışmasında görüldüğü üzere iki farklı usül görülebilir.

Terkîbî hesaplamalı hendese (combinatorial computational geometry): Bilgisayar bilimlerinde bulunan terkihi çalışmaların (combinatorial) bir alt grubudur. Genel olarak birden fazla ihtimali içersen sonlu sayıdaki elemalı kümelerin (finite element sets) üzerinde çalışan ve bu kümeler üzerinde ihtimal veya ittihat (birleştirme) (combination) işlemleri yapan çalışmalara, birleştirme anlamında terkip (combinatorial) ismi verilmektedir. Bu alan, bilgisayar bilimleri açısından da önemli bir yere sahiptir.

Sayısal hesaplamalı hendese (numerical computational geometry): Bilgisayar bilimlerinde, karşılaşılan problemlerin çözülmesi için geliştirilen diğer bir yöntem de sayısal analiz yöntemlerini kullanmaktır. Buna göre sistemden sonlu sayıda noktanın analizi yapılmaktadır. Örneğin sarnıçlama yöntemi ile seçilen noktalar üzerinde hesaplama yapıp sisteme genellemek gibi yöntemler kullanılır.

Yukarıdaki iki farklı hesaplamalı geometri usulünün farkını anlatmak için bir misal verelim.

Örneğin aşağıda (üstteki mavi çizimle gösterilen) fonksiyon, basit bir $\sin(x)$ fonksiyonudur.



Bu fonksiyonun analitik değerinin $\sin(x)$ olduğunu bilmeden, fonksiyona çeşitli değerler vererek şekli çizilirse, alttaki resimde görüldüğü üzere doğrusal bir fonksiyon bulunur. Bu fonksiyon, orjinal fonksiyona yakın ancak tam değildir. Örneğin integral almak gibi basit geometrik işlemler, yukarıdaki ilk fonksiyon üzerinden analitik olarak yapılabileceği gibi, ikinci fonksiyondaki örnek noktalar üzerinden de yapılabilir (bkz. Aradeğer ile integral hesabı (integral by interpolation))

SORU 11: 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 madeciliğinde (datamining) kullanılan ve veriler arasındaki ilişki modelini belirleyen bir algoritmaya da apriori ismi verilmiştir.

SORU 12: Document Management Systems (Doküman Yönetim Sistemleri)

Genel olarak bir doküman yönetim sistemi (document management system), herhangi bir organizasyondaki dokümanların (yazışmalar, evraklar, formlar veya resimler gibi) takibini yapmayı, tasnif ederek erişimini kolay hale getirmeyi, sorgulamalara ve aramalara cevap vermeyi hedefler.

Daha büyük ölçekte, içerik yönetim sisteminin bir parçası olarak görülebilir.

Bir doküman yönetim sisteminde, şart olmamakla birlikte genelde aşağıdaki öğeler bulunur:

Üst bilgi (Metadata): Bir dokümanın, doğrudan içeriğine ait olmayan ancak dokümanın algılanması ve kullanılması sırasında işe yarayan bilgilerdir. Örneğin bir resmin çekildiği tarih, veya sisteme kaydedildiği tarih, dokümanın üst bilgisidir ancak bu bilgi resmin içinde yer almak zorunda değildir.

Havuz (repository): Doküman yönetim sistemlerinde (document management systems) kullanılan ve dokümanların saklandığı alandır. Basitçe havuza nasıl erişileceği, kimlerin ne kadarına erişeceği veya dokümanların havuzda nasıl tutulacağı, doküman yönetim sistemi tarafından belirlenir. Örneğin dokümanların doğru bir veri tabanı yönetim sistemi tarafından (database management system , DBMS) nasıl kullanılacağı bir havuz problemidir.

Bütünleşme (Integration): Dokümanlar ile uygulamalar arasındaki ilişkidir. Örneğin bir kullanıcı bilgisayarında tuttuğu hesaplama tablolarını bir uygulama üzerinden kullanmaktayken (örneğin MS Excel veya Open Office Calc gibi) bu veriler doküman yönetim sisteminin havuzunda durmaktadır. Kullanıcı şeffaf bir doküman yönetim sisteminde

(transparent) sadece uygulama ile muhatap olurken, doküman yönetim sistemi, arka planda kullanıcının verileri ile uygulamayı bütünleştirmektedir.

Fihristleyici (Indexer): Dokümanların kolay erişilmesi için belirli bir fihrist yapısına oturtulmasını sağlayan programdır. Örneğin dokümanların ağaç yapısında (tree) veya özetleme tablolarında (hash table) tutulması, arama sırasında, daha hızlı erişilmesini sağlar. İşte fihristleyici bu görevi üstlenerek uygun veri yapısı üzerinde (data structure) dokümanlara erişim sağlamaktadır.

Getirme (Retrieval): Dokümanın havuzdan getirilmesi işlemidir. Çeşitli sistemlerde oldukça karmaşık yapılara ulaşabilir. Örneğin internet üzerinde arama yapan bir motorun (örneğin google) desteklediği arama cümlelerini kabul eden bir doküman yönetim sisteminin, fihristleyici üzerinde doğru erişimleri yapması ve getirmesi, normalden daha karmaşık algoritmalar gerektirmektedir. Hatta günümüzde doğal dil işleme (natural language processing) kullanılarak, kullanıcıların konuşur gibi, sistemden doküman aramaları ve doküman yönetim sisteminin (DMS) bu dokümanı bulup getirmesi mümkün hale gelmiştir. Ayrıca getirilen sonuçların kullanıcıya gösterilmesi sırasındaki sıralama da ayrıca önem taşımaktadır. Örneğin 100 adet doküman döndüren bir aramanın sonucunda hangi dokümanın ilk sırada gösterileceği, skorlama algoritmaları ile çözülmektedir ve bütün bu işlemler “getirme” (retrieval) olarak geçmektedir.

Dağıtım (Distribution): Bu terim aslında literatürde bilgisayar bilimi temelli kişiler için bir karmaşıklık oluşturmaktadır. Literatürde iki farklı anlamda kullanıldığı görülebilir. Birincisi çok yoğun ve yüksek miktarda verilerin saklandığı ortamlarda, kaynakların verimli kullanılması için dokümanların dağıtılmasıdır. Bu klasik bilgisayar bilimleri tanımı olup tamamen yük kaygısı ile performans artırma amaçlı yapılan bir işlemdir. Örneğin bir bankanın her şubesinin kendi dokümanını, kendi şubesinde tutması ve verilere erişimde hız kazanması gibi. Öte yandan doküman yönetim sistemlerinde bir dokümanın, taraflar arasında dağıtılması da bir problemdir. Örneğin hukuki bir metni, bütün muhataplarının okumuş olduğunun garanti edilmesi, hatta dijital olarak imzalaması (tebliğ / tebellüğ edilmesi) bir sözleşmenin bütün ilgili ve sadece ilgili kişiler tarafından okunması gibi problemler de dağıtım başlığı altında geçmekte olup bazı doküman yönetim sistemlerinde desteklenen özelliklerdendir.

Güvenlik (Security): Doküman yönetim sistemindeki kullanıcıların kendi yetkileri (authentication) çerçevesinde erişime sahip olması ve dokümanların tam olarak korunuyor olmasıdır.

İş akışı (workflow): Bir dokümanın izlemesi gereken yolları belirtir. Örneğin bir satış işlemi sırasında, önce satış sözleşmesinin taraflarca onaylanması, satıcı tarafın deposuna sevk emrinin geçirilmesi, depodan çıkan ürünlerin taşıyıcı firma tarafından onayı, aradaki sigorta evraklarının tanzimi, alıcı firma tarafından malların alındığını onaylayan dokümanlar ve nihayetinde para transferini onaylayan banka dokümanlarının tamamı doküman yönetim sistemi tarafından takip edilip doğru sıra ile, doğru taraflar tarafından, evrak atlanmadan yapılmış mı diye sorgulanabilir. Elektronik doküman yönetim sistemleri (electronic document management system), bu kontrol işlemlerinin tamamının otomatik olarak yapılmasını ve ilgili kullanıcılara raporlanmasını hedefler.

Sürümleme (Versioning) : Bir doküman üzerinde değişik zamanlarda yapılan çalışmaların takip edilmesini hedefler. Örneğin dokümanda yapılan her işlem yeni bir sürüm numarası ile saklanır ve kullanıcılar gerekli gördükleri durumlarda, eski sürümlere dönerek işlem yapabilir.

Yayınlama (Publishing): Dokümanın tam olarak doğru algılanmasını sağlamak amacıyla, doküman üzerinde, yanlış anlamalara mahal verebilecek bütün eksiklik ve hataların düzeltilmesi, tahsis edilmesi ve redaksiyonu işlemidir. Yayına hazırlanan bir dokümanın standartlarının belirlenmesinde ayrıca dokümanlar kullanılmaktadır.

SORU 13: Banker Algoritması (Banker's Algorithm)

Bilgisayar bilimlerinde işletim sistemi tasarımı konusunda geçen ve kaynaklar üzerindeki kilitlenmeyi (deadlock)engelleme amaçlı algoritmadır. Algoritma Dijkstra tarafından geliştirilmiştir.

Algoritmanın temel 3 durumu ve 2 şartı bulunur:

Bilmesi gerekenler:

1. Her işlem (process) ne kadar kaynağa ihtiyaç duyar?
2. Her işlem (process) şu anda ne kadar kaynağı elinde tutmaktadır?
3. Şu anda ne kadar kaynak ulaşılabilir durumdadır?

Yukarıdaki bu bilgileri bildikten sonra kaynak ayrılması sırasında aşağıdaki şartları uygular:

1. Şayet talep edilen kaynak, azami kaynaktan (maximum) fazla ise izin verme
2. Şayet talep edilen kaynak eldeki kaynaktan fazla ise, kaynak boşalana kadar işlemi (process) beklet.

Yukarıdaki algoritma detayında, kaynak olarak geçen değer, işletim sistemi için herhangi bir şey olabilir (hafıza (RAM), giriş çıkış işlemleri (I/O), gerçek sistemler için çalışma zamanı gibi)

Banker algoritması ayrıca çalışması sırasında yukarıdaki takipleri yapabilmek için bazı veri yapılarına ihtiyaç duyar. Aşağıdaki veri yapıları için n, sistemdeki işlem (process) sayısı ve m birbirinden farklı kaynak sayısı (resource) olmak üzere:

Müsait : m adet elemanı olan bir dizidir. Dizin her elemanı o kaynak tipinden ne kadar müsait olduğunu tutar. Örneğin $M[i] = 5$ değerinin anlamı, i kaynak tipinden beşinin müsait olduğudur.

Azami : iki boyutlu dizi ile tutulur ve $n \times m$ boyutlarındadır. Her işlem için ilgili kaynaktan ne kadar ayırım yapıldığı dizide işaretlenir. Örneğin $Az[2][3]=5$ gösteriminin anlamı, 2. işlemin 3. kaynak üzerinde 5 birimlik ayırım hakkı olduğudur. Diğer bir deyişle 2. işlem, 3. kaynaktan 5 birimden fazla kullanmaz, kullanmak istemez, kullanamaz.

Ayırım : yine iki boyutlu bir dizidir ve yine boyutu $n \times m$ olarak tutulur. Her işlemin her kaynağın ne kadarını kullandığını gösterir. Örneğin $Ay[2][3] = 4$ gösteriminin anlamı, o anda, 2. işlemin 3. kaynaktan 4 birim kullanıyor olduğudur (ayırması olduğudur).

İhtiyaç: Benzer şekilde $n \times m$ boyutlarında bir dizi olarak tutulur ve her işlemin her kaynaktan ne kadar ihtiyaç duyduğunu tutar. Örneğin $I[2][3] = 1$ gösteriminin anlamı, 2. işlemin, 3. kaynaktan 1 birim ihtiyacı olduğudur.

Algoritmanın çıktısı güvenli veya değildir (safe state, unsafe state). Buna göre algoritma, işlemlerin çalışıp bitme ihtimali varsa güvenli sonucunu döndürürken, işlemler, birbirini kilitliyorsa bu durumda güvensiz sonucunu döndürür.

Hesaplama:

Algoritma, güveni veya güvensiz sonucuna ulaşmak için adıma bağlı olarak, aşağıdaki hesaplamaları yapar:

$$\text{Müsait} = \text{Müsait} - \text{İhtiyaç}$$

$$\text{Ayrım} = \text{Ayrım} + \text{İhtiyaç}$$

$$\text{İhtiyaç} = \text{Azami} - \text{Ayrım}$$

Bu durumu bir örnek üzerinden açıklayalım. Örneğin A,B,C isimli üç kaynağımız olsun ve bu kaynakların müsaitlik durumları aşağıdaki şekilde olsun:

Kaynaklar

	A	B	C
10	5	7	

Ayrıca 5 adet işlemimiz olsun ve bu işlemlerin anlık olarak azami ihtiyaçları, ayrılmış olan kaynaklar ve ihtiyaçları aşağıda verildiği gibi olsun:

	Azami			Ayrım		
	A	B	C	A	B	C
P0	7	5	3	0	1	0
P1	3	2	2	2	0	0
P2	9	0	2	3	0	2
P3	2	2	2	2	1	1
P4	4	3	3	0	0	2

Soru bu durumun güvenli bir durum olup olmadığıdır. Diğer bir deyişle acaba bir kilitlenme riski bulunur mu? Öncelikle yukarıdaki ayrım tablosunda bulunan kaynakları toplayalım:

$$\text{A kaynağı için : } 0 + 2 + 3 + 2 + 0 = 7$$

$$\text{B kaynağı için : } 1 + 0 + 0 + 1 + 0 = 2$$

$$\text{C kaynağı için : } 0 + 0 + 2 + 1 + 2 = 5$$

yukarıdaki toplama işlemleri, ilgili kaynaktan, başlangıçta ayrılmış miktarların toplamıdır ve kaynağın bulunduğu kolonun toplanması ile bulunabilir.

Şimdi başlangıçtaki kaynak miktarından bu toplamı çıkararak başlangıçta bir işlemin çalışması için müsait kaynakları bulalım:

$$\text{A : } 10 - 7 = 3$$

$$B : 5 - 2 = 3$$

$$C: 7 - 5 = 2$$

Bu kaynaklarla başlayarak acaba sistem kilitlenme olmadan çalışabilir mi?

Yukarıda verilen değerlerin üzerinden ihtiyaç tablomuzu hesaplayalım:

	Azami	-	Ayrım	=	İhtiyaç
	A B C		A B C		A B C
P0	7 5 3		0 1 0		7 4 3
P1	3 2 2		2 0 0		1 2 2
P2	9 0 2		3 0 2		6 0 0
P3	2 2 2		2 1 1		0 1 1
P4	4 3 3		0 0 2		4 3 1

Yukarıdaki işlemten anlaşılacağı üzere Azami tablosundan, Ayrım tablosu çıkarılmış ve ihtiyaçlar bulunmuştur.

Bu durumda, sistem güvenli denilebilir çünkü ihtiyaç tablosundaki hiçbir değer, müsait dizimizdeki değeri geçmemektedir. Demek ki sistem bütün ihtiyaçlara cevap verebilecek kapasitededir.

Bu durumun daha iyi anlaşılması için P1, P2, P3, P4, P0 sırası ile çalışmayı görelim:

	Azami	-	Ayrım	=	İhtiyaç	Müsait	
	A B C		A B C		A B C	A B C	
P1	3 2 2		2 0 0		1 2 2	3 3 2	3 3 2
P3	2 2 2		2 1 1		0 1 1	5 3 2	
P4	4 3 3		0 0 2		4 3 1	7 4 3	
P2	9 0 2		3 0 2		6 0 0	7 4 5	
P0	7 5 3		0 1 0		7 4 3	10 4 7	
						10 5 7	<<< Kaynaklar

Görüldüğü üzere işlemlerin sonucunda ilk kaynak değerlerine geri dönmüştür (aslında bütün işlemler bitince müsait olan kaynak değeri, başlangıçtaki değerdir). Yukarıdaki çalışma sırası (yani P1, P2, P3, P4, P0 sırası) güvenli çalışma olarak kabul edilir. Örneğimizi biraz daha ilerletelim : Acaba P1 işlemi (1,0,2) ihtiyacında olsaydı yine de güvenli durumda olur muyduk? İhtiyaç kontrolümüz (1,0,2)<= (1,2,2) olacaktı ve müsait durumumuz : (1,0,2)<= (3 3 2) olacaktı Bu durumda tablomuz aşağıdaki şekilde olabilirdi:

	Azami	Ayrım	İhtiyaç	Müsait
	A B C	A B C	A B C	A B C
P0	7 5 3	0 1 0	7 4 3	2 3 0<<<
P1	3 2 2	3 0 2<<<	0 2 0<<<	
P2	9 0 2	3 0 2	6 0 0	
P3	2 2 2	2 1 1	0 1 1	
P4	4 3 3	0 0 2	4 3 1	

Bu durum güvenli midir? Evet !

Çalışma sırası olarak P1, P3, P4, P0 ve P2 sırasında olması durumunda işlemler kilitlenme olmaksızın başarı ile çalışacaktır. Örneğin P4 için ihtiyacı (3,3,0) olarak yeniden ayarlasaydık

güvenli bir durum elde edemeyecektik çünkü bu adımdaki (3,3,0) değeri o anda müsait olan (2,3,0) değerinden büyük olacaktı ve işlem (process) çalışamayacaktı.

SORU 14: Bin Packing (Kutulama Problemi)

İyileştirme problemleri açısından klasik bir örnektir (optimisation problems). Problem basitçe bir kutunun içerisine en az boş alan bırakarak, eşyaların en iyi şekilde nasıl yerleştireceği olarak düşünülebilir.

Aslında problemi boyutlara göre incelersek aşağıdaki şekilde bir liste yapılabilir:

Tek boyutlu kutulama (1D bin packing) :Bu problemde amaç bir çizgi veya hat gibi görülebilecek yapının içerisine farklı boyutlardaki çizgileri yerleştirmek olarak düşünülebilir.

Örneğin zaman çizelgelemesinde, bir kişinin yapacağı işleri, zaman çizgisinin üzerine yerleştirmesi (ve her işin farklı miktarda zaman gerektirdiğini ve kişinin çalışma saatlerinin sınırlı olduğunu düşünürsek en fazla işi en az zamanda (örneğin 8 saatlik mesailer içinde) yapması) bir problemidir. Buradaki hem kişinin yapacağı işler hem de bu işlerin yerleştirileceği zaman çizgisi tek boyutludur.

Daha basit olması açısından örneğin 100m uzunluğundaki bir ipi 7 ve 9m uzunluğundaki parçalara en az fire ile bölmek istiyoruz, en verimli bölme işleminde kaç adet 7 ve kaç adet 9 uzunluğunda ipimiz olur gibi soruları düşünebiliriz. Bu tip sorular tek boyutlu kutulama problemleridir.

İki boyutlu kutulama problemleri (2D bin packing optimization): Bu grupta bir tabloda ve iki boyuttan bahsedilebilir. Örneğin kot pantolon üreten bir tekstil firmasında farklı boyutlardaki Pantolon kalıplarının en az fire ile 5x5m büyüklüğündeki bir kare kumaştan kesilmesi isteniyor olsun. Bu problem iki boyutlu (x ve y boyutları) bir kutulama problemidir. Benzer bir problem, bir gazetedeki seri ilanların, en az fire ile sayfaya yerleştirilmesi olarak da düşünülebilir.

3 boyutlu kutulama (3D bin packing) problemin en zor şekli olarak tanımlanır ve 3 boyutlu bir kutunun içerisine konulan her şeklin farklı x,y ve z boyutlarında şekiller olması olarak düşünülebilir. Problemin genel tanımını yaptığımız için belirteyim, örneğin ev taşıma sırasında çıkan eşyaların kutulanması olarak düşünebilirsiniz. Bu durumu daha da karmaşık yapmaktadır çünkü kutular farklı boyut ve şekillerdedir (örneğin silindir bir varil veya küp veya dikdörtgenler prizması gibi kutuların içerisine yerleştirme yapılmakta) ve kutulanan şekillerde farklıdır ve hatta girintilidir (concave , non-convex) (örneğin avize, koltuk, sandalye gibi birbirinin içine girebilen nesneleri düşününüz). 2 ve 3 boyutlu paketleme, paketlenen nesnelerin girintili olup olmamasına göre ikiye ayrılmaktadır. Dış büyük nesnelerin paketlenmesi nispeten daha basit bir problemidir. Ancak nesnelerin iç büyük olması halinde problem biraz daha karmaşıklaşır.

Hatta literatürdeki kısıtlı aramalarım sonucunda ulaşabildiğim kadarıyla tam olarak iç büyük nesnelerin paketlenemediği bir sonuç ne yazık ki bulamadım. Örneğin iki vidanın en verimli paketlenmesi sırasında vidaların girinti çıkıntılarının üstüste gelmesi gerektiğini tecrübi olarak biliyoruz. N adet vida için bu durum birbirini tekrar eden bir hal alır. Gerçekten farklı boylarda ve adım sıklığında ve çaplarda vidalar verilse bu durumda en verimli paketlemeyi yapabilen bir algoritma henüz görmedim.

Paketlenen nesnelere göre problemin sınıflandırılması:

Paketlenen nesne çeşitlerinin sabit olması ve ön tanımlı olması halinde problem homojen olarak tanımlanır. Örneğin tek boyutlu kutulama probleminin tanımı sırasında verilen ve “100m uzunluğundaki bir ipi 7 ve 9m uzunluğundaki parçalara en az fire ile bölmek” şeklinde geçen örnek bu tip homojen (homogenous) bir yapıdadır. Buna karşılık heterojen bir problemde, paketlenen nesnelerin tipleri ya tamamen birbirlerinden farklıdır ya da aynı tipte çok az tekrar vardır. Yine tek boyutlu problem örneğinde verilen zaman çizgisi üzerinde farklı uzunluklardaki randevuların yerleştirilmesi bu tiptendir.

Bu anlamda aşağıdaki problemler, kutulama probleminin birer özel hali olarak düşünülebilir:

- kamyon yükleme problemi (truck loading),
- konteyner yükleme problemi (Container loading problem, CLP)
- şerit paketleme problemi (Strip Packing problem, SPP)

Yukarıda, problemin tanımını yaptıktan sonra çözümlere bir göz atalım:

Homojen tek boyutlu problem çözümü

Şayet problem tek boyutlu ise ve homojen nesnelerin paketlenmesi olarak problemin çözülmesi isteniyorsa problem oldukça basit demektir ve basit matematiksel hesaplamalar ile problemi çözebiliriz.

Örneğin tek nesne ve tek paket varsa işlem basitçe paketin nesneye bölümü olarak bulunur (zaten burada zor Bir şey de yok):

Örneğin 100m uzunluğundaki bir ipten kaç tane 5m uzunluğunda ip kesilebilir:

$$100 / 5 = 20$$

biraz daha zorlaştırıp ip sayısını 2 çeşide veya 3 çeşide çıkarırsak problem np-tam (np-complete) bir hal alır. Örneğin aşağıdaki kodu inceleyelim:

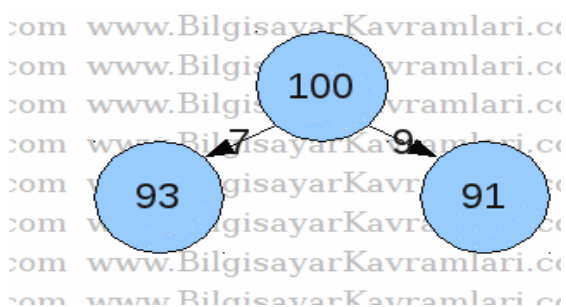
```

1  #include <stdio.h>
2
3  int bul(int a,int b,int n,int m,int k){
4      if(k==0){
5          printf("\ncozum : %d*%d + %d*%d = %d",a,n,b,m,k);
6          return 1;
7      }
8      if(k<0)
9          return -1;
10     int p = bul(a,b,n+1,m,k-a);
11     int q = bul(a,b,n,m+1,k-b);
12     if(p>q)
13         return p;
14     return q;
15 }
16 //www.bilgisayarkavramlari.com
17 int main(){
18     int k = 100;
19     int a = 7;
20     int b = 9;
21     while(bul(a,b,0,0,k)<0){
22         printf("optimum cozum yok %d icin cozum deneniyor",--k);
23     }
24     return 0;
25 }

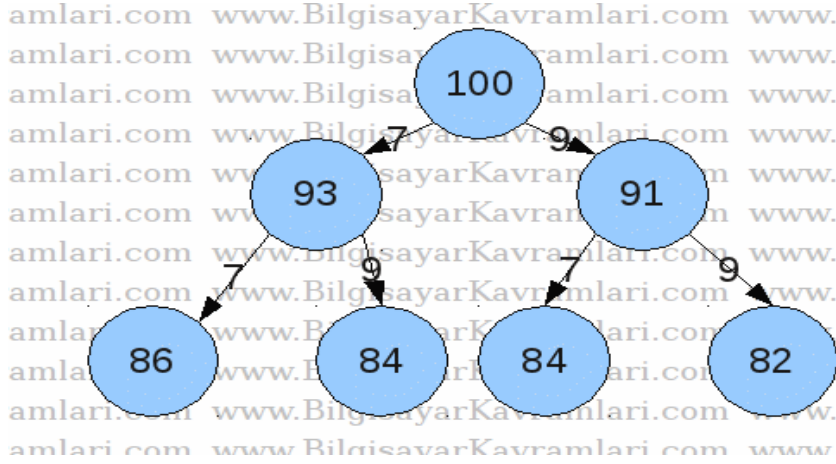
```

Kodda görüldüğü üzere bütün ihtimaller denenmektedir. Basitçe herhangi bir k değeri için, k-a ve k-b değerlerini denemekte ve denenen duruma göre a veya b değerini bir arttırmaktadır. Aslında kodumuz basit bir ikili ağaç (binary tree) oluşturmaktadır:

Önce 9 veya 7 ile başlanması ihtimalleri:



Sonra bu ihtimallerin de 7 veya 9 azalma ihtimalleri:



Yukarıda gösterildiği gibi her düğümden yine ikiye ihtimal indirerek bir alt seviyeye geçilebilir. Neticede 0 olana kadar yapılan bir aramadır ve 0 sonucuna birden farklı yoldan ulaşılabilir.

Kodumuz çalıştırıldığında çözüm olarak aşağıdaki sonuçları üretmektedir:

cozum : $7*4 + 9*8$

cozum : $7*13 + 9*1$

Gerçekten de problemin iki farklı çözümü bulunmaktadır.

Yukarıdaki kod basit bir hesaplama ile, 2'nin üstlerinin toplamı kadar adım hesaplamaktadır.

Bu değer yukarıdaki ağaçtan çıkarılabilir:

ilk düğüm için tek ihtimal 2^0

ikinci seviye için iki ihtimal: 2^1

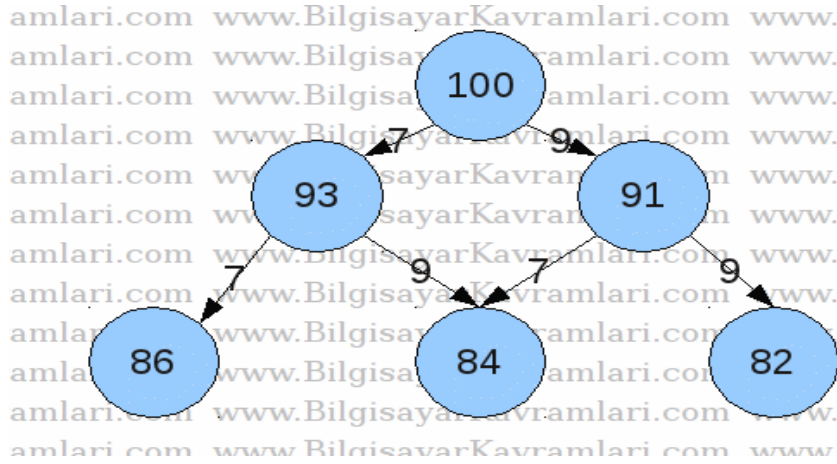
üçüncü seviye için dört ihtimal 2^2

şeklinde gitmektedir ve örneğin problemimiz üçüncü seviyede çözülsüydi (sonuç 0 olsaydı) o zaman karmaşıklığımız bu değerlerin toplamı olacaktı ve $2^0 + 2^1 + 2^2$ şeklinde hesaplanacaktı.

Bu değer, ikili ağaçlardan bilindiği üzere $2^n - 1$ şeklinde hesaplanabilir.

Görüldüğü üzere yukarıdaki algoritma $O(2^n)$ değerinde bir karmaşıklığa sahiptir ve bu değer bir çok terimli (polynom) değildir yani algoritmanın karmaşıklık sınıfı np-tam (NP-Complete) olarak belirtilebilir. Ayrıca yukarıdaki k değeri için bir çözüm bulunmuştur ancak çözüm bulunamasa bu değer k terim için denenecekti. Yani 100 için çözüm yoksa bir yaklaşığı olan 99 için ardından iki yaklaşığı 98 için ... Bu işlem hiç çözüm bulunamaması halinde k terim için denenecekti.

Yukarıdaki problem, dinamik programlama (dynamic programming) kullanılarak iyileştirilebilir. Bunun sebebi arama işlemi sırasında bazı sonuçların tekrar etmesidir. Örneğin yukarıdaki ikili ağacı aşağıdaki şekilde çizebiliriz:



Farka dikkat ederseniz, $93 - 9 = 84$ ve aynı zamanda $91 - 7 = 84$ olduğu görülür. Bu durumda aslında 84 değeri bir önceki kodda iki farklı durum için aranmaktayken şimdi tek bir durum için aransın istiyoruz. Elbette 84 sadece bir örnektir ve buna bağlı olarak çok sayıda tekrar eden değer bulunmaktadır.

Hesaplanan bu değerleri bir dizi (array) içerisinde tutup tekrar hesaplanmasını engellemek istiyoruz:

```

1  #include <stdio.h>
2  //www.bilgisayarkavramlari.com
3  int main(){
4      int k = 100;
5      int a = 7;
6      int b = 9;
7      int cozum[101][2]; // k = 100 icin 101
8      for(int i = 0;i<=k;i++){
9          cozum[i][0] = -1;
10         cozum[i][1] = -1;
11     }
12     cozum[0][0] = 0; // 0 icin 0 tane 7
13     cozum[0][1] = 0; // 0 icin 0 tane 9 kullanilir
14     for(int i = a;i<=k;i++){
15         if(cozum[i-a][0]>=0){
16             cozum[i][0] = cozum[i-a][0]+1;
17             cozum[i][1] = cozum[i-a][1];
18         }
19         else if(cozum[i-b][0]>=0){
20             cozum[i][0] = cozum[i-b][0];
21             cozum[i][1] = cozum[i-b][1]+1;
22         }
23     }
24     for(int i = 0;i<=k;i++){
25         printf("\n%d*%d + %d*%d = %d",a,cozum[i][0],b,cozum[i][1],i);
26     }
27     return 0;
28 }

```

Yukarıdaki yeni kodda, bir dizi içerisinde tek döngü ile sonucu hesaplattık. Buna göre algoritmamız iki elemanlı bir diziyi kullanmakta, dizinin 0. elemanları 7lerin sayısını ve 1. elemanları ile 9ların sayısını saymaktadır.

Kodumuz ilk başta 0 için 0 tane 7 ve 0 tane 9 gerektiği gerçeği ile çalışmaya başlıyor. 14-23. satırlar arasındaki döngü basitçe i. terim için i-a ve i-b değerlerine bakıyor. Şayet i. terim için i-a veya i-b değerinde bir çözüm varsa (nereden geldiğini önemsemeksizin) bu çözüme bulduğu koşulu ilave ederek mevcut i değeri için çözümü kaydediyor. Şayet bu iki terim de bulunmuyorsa o zaman bir sonraki i değerine geçiyor.

Ekran çıktısı aşağıdaki şekildedir:

```
7*9 + 9*8 = 80
7*9 + 9*2 = 81
7*8 + 9*9 = 82
7*8 + 9*3 = 83
7*12 + 9*0 = 84
7*11 + 9*7 = 85
7*11 + 9*1 = 86
7*10 + 9*8 = 87
7*10 + 9*2 = 88
7*9 + 9*9 = 89
7*9 + 9*3 = 90
7*13 + 9*0 = 91
7*12 + 9*7 = 92
7*12 + 9*1 = 93
7*11 + 9*8 = 94
7*11 + 9*2 = 95
7*10 + 9*9 = 96
7*10 + 9*3 = 97
7*14 + 9*0 = 98
7*13 + 9*7 = 99
7*13 + 9*1 = 100shedai@pardus2011 ~ $
```

Son 20 satır görülmekle birlikte daha önceki satılar alıntılanmamıştır. Ayrıca son 20 satırda görüldüğü üzere, tamamına ait bir çözüm bulunmaktadır. Örneğin 88 sayısı için $7*10 + 9*2 = 88$ sonucuna ulaşılmıştır. En son satırda ise 100 için $7*13 + 9*1$ sonucu görülmektedir.

Görüldüğü üzere birden fazla sonuç olsa bile tek bir sonucu görmekteyiz bunun sebebi veri yapısının bir sonuç üretmek üzere tasarlanmış olmasıdır. Elbette daha farklı veri yapıları kullanılarak diğer çözümleri de gösteren sonuçlar elde edilebilir.

Yukarıdaki algoritmanın karmaşıklığı ise bir öncekine göre oldukça iyi sayılabilecek $O(n)$ olarak bulunur. Bunun sebebi dizideki her elemanın üzerinden tek bir kere geçiyor olması ve dolayısıyla tek bir döngünün (koddaki 14-21 satırlar arası) çalışıyor olmasıdır.

SORU 15: Etraflı Arama (Tam Arama, Exhaustive Search)

Literatürde tam arama veya etraflı arama olarak geçmektedir. İngilizcede “exhaustive search” terimi kullanılır. Genel olarak, arama algoritmalarının performansını arttırmak için kullanılan bir yöntemdir.

Bir arama algoritmasının tam arama (exhaustive search) olabilmesi için aşağıdaki şartları sağlaması gerekir:

- Bir değer bulunmadığını söylemeden önce bütün değerlere bakmış veya bütün ihtimalleri değerlendirmiş olmalıdır.
- Arama uzayında hareket edilirken (Arama işlemi sırasında) sistematik bir yöntem kullanılmalıdır. Diğer bir deyişle arana değerlere tekrar tekrar bakılmamalı, ve her adımdan sonra hangi değere bakılacağı belirlenmelidir.

Yukarıdaki şartları sağlayan arama algoritmalarına etraflı arama veya tam arama ismi verilir. Ancak yukarıdaki ikinci maddenin bir istisnası bulunur. Bazı durumlarda arama algoritmasını hızlandırmak için rast gele bir değer ataması (randomness) kullanılabilir. Bu durum bir çelişki olarak görülmemelidir. Algoritmanın içerisinde bir rastgelelik bulunsa bile, aranan değerleri

tekrar aramaması ve üretilecek olan rastgeleliğin tam kontrol altında tutulması halinde, arama algoritması etraflı arama olarak sınıflandırılabilir.

Etraflı arama algoritmalarını aynı zamanda birer geri izleme algoritması (back tracking algorithm) olarak isimlendirmek mümkündür. Aslında bütün etraflı arama algoritmaları (Exhaustive search algorithms) birer geri izleme algoritmasıdır (back tracking algorithm). Ancak bu cümlemin tersi doğru değildir. Yani bütün geri izleme algoritmalarının etraflı arama algoritması olduğunu söyleyemeyiz. Buradaki fark, etraflı arama algoritmalarının, arama işlemi sırasında aranacak olan değerlerin bir kısmını budamasından kaynaklanır (prunning). Yani bazı değerlerin bakılmasına gerek kalmayacak şekilde arama işlemini hızlandırır.

Bu anlamda doğrusal arama (linear search), kaba kuvvet araması (brute force algorithm) gibi temel arama algoritmaları birer etraflı arama kabul edilebilir.

SORU 16: Algoritma (Algorithm)

Bazan 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 17: Emil Post Makinesi

Bilgisayar bilimlerinde özellikle özdevinirli kuramında (automata theory) geçen bir makine modelidir. Yapı olarak Turing makinesine (Turing machine) çok benzer hatta ufak farklılıklar dışında neredeyse aynı olduğu söylenebilir. Turing makinesini geliştiren Alan Turing ile bağımsız olarak geliştirilmiştir. Bu anlamda Post makinesi bazı kaynaklarda Post-Turing makinesi (Post Turing Machine) olarak da geçmektedir.

Turing makinesinde kullanılan bant mantığına benzer bir şekilde, Post makinesinde de kutular bulunmaktadır. İki yöne doğru sonsuz sayıda kutunun bulunduğu bir üretim bandı gibi düşünülebilir. Yani kutulara erişim ardışıktır (sequential) ve kutular ileri veya geri olmak üzere iki yönde hareket edebilir. Post makinesi burada sanki bir çalışanın üretim bandındaki kutular üzerinden çalıştığı kabulü ile yola çıkar.

Post makinesinde, başlangıç durumunda sonsuz kutu bulunmakta ancak bunların sonlu bir kısmı işaretli, geri kalan sonsuz kısmı ise işaretli kabul edilmektedir. Yani başlangıç durumunda makinemizin hafızasında bir bilgi olmasını istiyorsak, bu bilgiyi kutulara işaretliyoruz. Geri kalan kutular ise işaretli oluyorlar.

Post makinesi, aynı anda tek bir kutuya erişim sağlayabilir. Buna göre kutulara aşağıdaki işlemler uygulanabilir:

1. Boş olan bir kutu işaretlenebilir
2. İşaretli olan bir kutu silinerek işaretli hale getirilebilir.
3. Sağdaki kutuya geçilebilir
4. Soldaki kutuya geçilebilir
5. Şu anda bakılan kutunun işareti okunabilir.

Post makinesinin kendisine özel yazım kuralları bulunmaktadır (notation). Herhangi bir adımda (bu adıma örnek için i diyelim), herhangi bir işlem (operation) yukarıdaki adımlardan birini parametre almaktadır. Örneğin O_i [3] şeklindeki yazım, i . adımda makinemizin sağdaki kutuya geçeceği şeklinde okunabilir. Ayrıca makinemizin yukarıdaki işlemlerden 5ncisini işletmesi durumunda sonraki adımlarda çatallanma olur. Yani okunan değer işaretli veya işaretli oluşuna göre makinemiz farklı davranabilir. Bunun dışındaki işlemler bir akış şeklinde sonraki adıma geçer.

Yukarıdaki ilk modele ilave olarak zaman içinde yapılan ilaveler ile birlikte bugün Post-Turing makinesi olarak anılan makine aşağıdaki 7 işlemi yapabilmektedir.

PRINT 1 | 1 yaz

PRINT 0 | 0 yaz

GO RIGHT | sağa git

GO LEFT | sola git

GO TO STEP i IF 1 IS SCANNED | şayet 1 okursan i. adıma git

GO TO STEP i IF 0 IS SCANNED | şayet 2 okursan i. adıma git

STOP | bitir

Yukarıdaki son haliyle, post makinesi, Turing makinesinin yapabildiği işlemleri yapabilecek seviyeye ulaşmış olur. Yukarıdaki örnek ikilik tabanda (binary) bir makine olup, dilenirse sembol tablosu genişletilerek daha farklı işaretleri kutulara yerleştirmesi de sağlanabilir.

SORU 18: Monte Carlo

Bilgisayar bilimleri de dahil olmak üzere pek çok istatistiksel hesabın gerektiği alanda kullanılan bir yaklaşımın ismidir. İsmi bir kumarhane olan monte carlo'dan gelmektedir ve kumarhanede oynanan oyunlardan çıkmış bir yöntemdir.

Yöntemin genel yapısı aşağıdaki şekilde özetlenebilir:

1. Sisteme girenler için bir alan tanımı (domain) yap
2. Bu tanımlı alandan bir dağılıma uygun olarak rast gele girdiler üret
3. Girdiler üzerinde belirli bir hesaplama yap (bu hesaplamayı bir fonksiyon olarak düşünebiliriz ve sisteme göre değişebilir)
4. Sonuçları topla

Yukarıdaki adımları bir örnek üzerinden anlatmaya çalışalım. Örneğin bir karenin içerisine kenarları teğet olacak şekilde yerleştirilen bir dairenin alanı ile karenin alanı oranının $\pi/4$ olduğu biliniyor olsun. Yukarıda anlatılan adımlarla π sayısını bulmaya çalışalım:

1. Yere bir kare bir de daire, anlatıldığı gibi çiz.
2. Çizilen bu şeklin üzerine tekdüze dağılımda (uniform distribution) noktalar serp (örneğin pirinç veya boncuk gibi nesneleri rast gele ser)
3. Şeklin üzerinde noktaların düştüğü alanlara göre noktaları ayırarak say.
4. Bu saydığın değerleri birbirine böl ve beklenen değere yakın bir değer çıktığını ölç

Yukarıdaki sistemin başarısı iki önemli hususa dayanmaktadır. Öncelikle dağılımımız gerçekten tam bir tekdüze (uniform) özelliğinde olmalıdır. Şayet dağılımımızda hata varsa sistem başarısız olur. Ayrıca örnekte verilen noktaların sayısı yeterince fazla olmalıdır. Örneğin tek bir nokta ile yukarıdaki deneyin yapılması durumunda başarısız olunur. Sayı arttıkça beklenen sonuca ulaşmamız daha da kesinleşir.

Yukarıdaki sistemde görüldüğü üzere sistemin girdilerinin doğru tespit edilmesi, dağılımın doğru bir şekilde seçilmesi ve ardından yeterli miktarda deney yapılarak sonuçların doğru şekilde toplanması halinde sistem her zaman başarılı çalışacaktır.

SORU 19: Spagetti Kod (Spaghetti Code)

Bilgisayar kodlamasında, bir kodun okunabilirliğinin düşük olması, yani kod takibinin zor olması durumunda, koda verilen isimdir.

Genellikle yapısal programlama dillerinde (structured programming languages) fonksiyonların bulunması ile birlikte GOTO veya JMP gibi, kodun içerisinde bir yerden başka bir yere atlayan komutların kaldırılması mümkündür.

Bu tip komutların kullanılması durumunda, kodun hem diğer programcılar tarafından okunabilirliği düşer, hem de kodun karmaşıklığı kestirilemez bir hâle gelebilir.

Spagetti kodu engellemek için 3 temel adım kullanılır:

1. Kodda bulunan ve koşula bağlanması istenen satırlar için if bloğu kullanılır
2. Kodda bulunan ve tekrarlanması istenen satırlar için döngü kullanılır
3. Kodda bulunan ve parametrize edilmesi istenen satırlar için (aynı satırların farklı değerlerle çalışması isteniyorsa) fonksiyon kullanılır.

Yukarıdaki bu üç adım tamamlandıktan sonra kodda herhangi bir GOTO satırı kalmamalıdır.

Ayrıca spagetti kodundan esinlenerek katmanlı mimarinin kullanıldığı ve projelerin birden fazla seviyeye bölünerek ele alındığı yaklaşımlara lazanya kod (lasagna code) ; veya özellikle nesne yönelimli programlama dillerinde (object oriented programming languages), program parçalarının birbiri ile olan ilişkisinin asgariye indirildiği ve neredeyse her parçacığın tek başına varlığını sürdürebildiği kodlama yaklaşımlarına da ravioli kod (ravioli code) isimleri verilmektedir.

SORU 20: Merkezi poligon sayıları (Central Polygon Numbers)

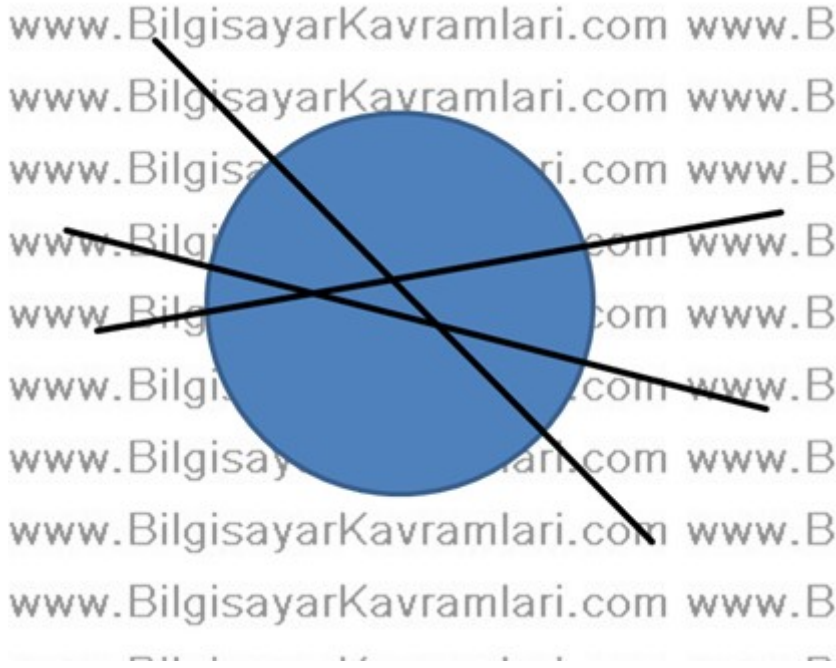
Bir dairenin verilen doğru sayısı ile kaç farklı parçaya bölünebileceğini veren sayı serisidir.

1, 2, 4, 7, 11 şeklindeki sayılara, merkezi poligon sayıları ismi verilir. Bu sayılar, verilen doğru sayısına göre, bir daireyi kaç farklı şekle böldüğünü gösterir.

Örneğin yukarıdaki sayı serisini eksi olmayan tam sayılar ile birebir eşleştirirsek

0	1	2	3	4	5
1	2	4	7	11	16

Yukarıdaki gibi bir tablo elde ederiz. Bu tablodaki ilk satır kaç doğru kullandığımız, ikinci satır ise kaç parça elde ettiğimizdir. Örneğin ilk sütunda 0 doğru kullanılmış ve daire en fazla tek parçaya ayrılabilmiştir. İkinci sütunda tek bir doğru ile daire iki parçaya ayrılmıştır. 3 doğru kullanarak daire en fazla 7 parçaya ayrılabilir. Bu durumu aşağıdaki şekilde görebilirsiniz:



Yukarıdaki şekilde görüldüğü üzere daire 7 farklı parçaya bölünmüştür.

Bu sayılara aynı zamanda, tembel pizzacı serileri (lazy caterer's sequence) isimleri verilir. Bu ikinci isimlendirme, bir pizzacının, pizzasını en az bıçak darbesi ile en çok parçaya ayırma hevesinden gelmektedir ◀◀

Floyd üçgeninin (Floyd's Triangle) ilk sütununu oluşturan bu seri, aşağıdaki formül ile hesaplanabilir:

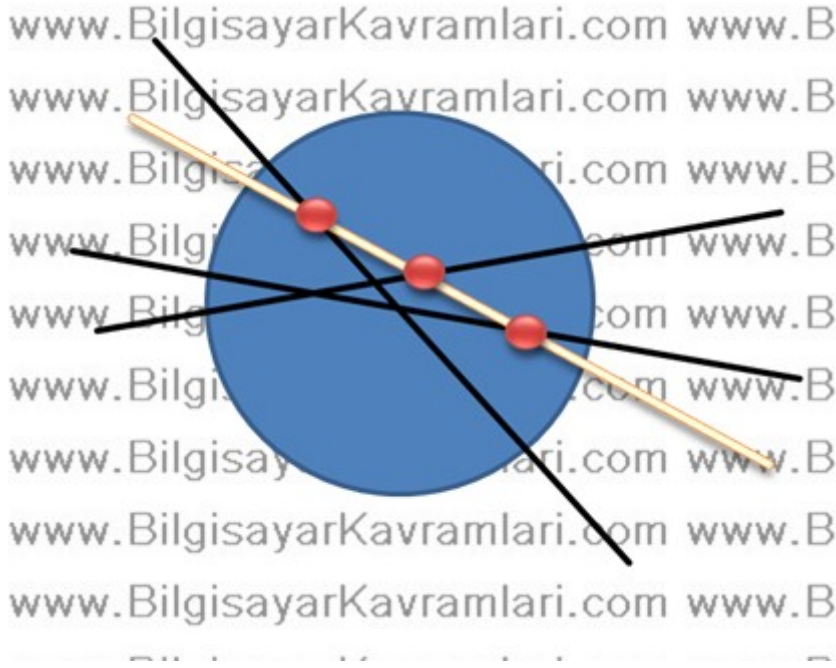
$$k = (n^2 + n + 2) / 2$$

Bu formülün çıkarılışı ise özyineli bir denklemin (recursive equation) çözümünden gelmektedir.

Dairenin her yeni çizgi ile bölünmesini aşağıdaki şekilde formülleyebiliriz:

$$T(n) = T(n-1) + n$$

Bu denkleme göre her yeni çizgi, daha önceki çizgileri keserek o ana kadar olan çizgi sayısının bir fazlası yeni parça oluşturacaktır. Örneğin, yukarıdaki şekle yeni bir çizgi ekleyerek bu durumu görelim:



Yukarıdaki şekilde görüldüğü üzere eklenen yeni doğru, daha önceden bulunan bütün doğrular ile kesişmelidir. Şekilde bu kesişim noktaları işaretlenmiştir. Yeni çizgi, şekle eklendikten sonra şekildeki alan sayısı 11'e yükselir. Daha önceden vâ r olan 3 doğru ile kesiştikten sonra, yeni 4 alan çıkmaktadır. Dolayısıyla denklemimiz:

$T(4) = T(3) + 4$, yani ilk üç doğru için oluşan alan + 4 yeni alan şeklinde özetlenebilir.

Bu denklemin çözümüne geçecek olursak:

$T(n) = T(n-1) + n$, denklemi yukarıdan aşağıda doğru açıyoruz (top-down approach)

$$T(n) = T(n-2) + n-1 + n$$

$T(n) = T(0) + 1 + 2 + \dots + n$, Denklemdaki $T(0)$ değeri 1'dir çünkü daireyi kesmek için hiçbir çizgi kullanılmazsa (çizgi sayısı 0 ise) daire tek parçadır.

$T(n) = 1 + 1 + 2 + \dots + n$, Gauss'un 1'den n'e kadar olan sayıların toplamı formülünü yazarsak

$T(n) = 1 + n(n+1) / 2$, toplama işlemini yaparsak

$$T(n) = (n^2 + n + 2) / 2$$

Şeklinde denklemi çözmüş oluruz. Bu denklemin ve bu denklemden üretilen sayı serisinin bir diğer özelliği ise, üçgen sayılarının bir fazlası olmasıdır (triangle numbers).

SORU 21: Mealy ve Moore Makineleri (Mealy and Moore Machines)


Bilgisayar bilimlerinde sıkça kullanılan sonlu durum makinelerinin (finite state machine, FSM veya Finite State Automaton , FSA) gösteriminde kullanılan iki farklı yöntemdir. Genelde literatürde bir FSM'in gösteriminde en çok moore makinesi kullanılır. Bu iki yöntem (mealy

ve moore makinaları) sonuçta bir gösterim farkı olduğu için bütün mealy gösterimlerinin moore ve bütün moore gösterimlerinin mealy gösterimine çevrilmesi mümkündür.

Klasik bir FSM'de bir giriş bir de çıkış bulunur (input / output). Bu değerlerin nereye yazılacağı aslında iki makine arasındaki farkı belirler.

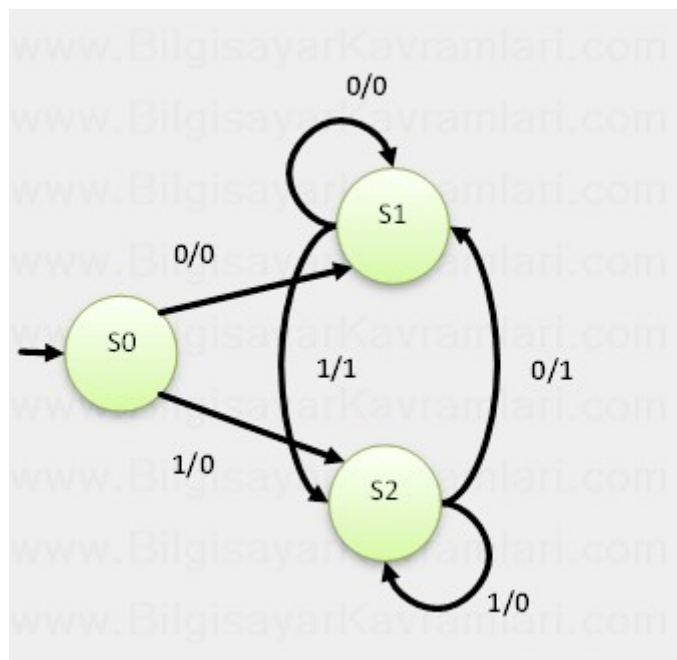
Moore makinelerinde çıkış değerleri düğümlere (node) yazılırken, giriş değerleri kenarlar (edges) üzerinde gösterilir.

Mealy makinelerinde ise giriş ve çıkış değerleri kenarlar (edges) üzerinde aralarına bir taksim işareti (slash) konularak gösterilir. Örneğin 1/0 gösterimi, girişin 1 ve çıktının 0 olduğunu ifade eder.

Basit bir örnek olarak özel veya (exclusive or, ) işlemini ele alalım ve her iki makine gösterimi ile de çizmeye çalışalım.

Girdi 1	Girdi 2	Çıktı
0	0	0
0	1	1
1	0	1
1	1	0

Klasik bir XOR kapısı, yukarıdaki doğruluk çizelgesinde (truth table) gösterildiği üzere iki giriş ve bir çıkıştan oluşur. Bizim makinemiz de ilk girdiden sonra ikinci girdiyi aldığı anda beklenen çıktıyı vermeli. Ayrıca makine sürekli olarak çalışmaya devam edecek. Örneğin 101011 şeklinde bir veri akışı sağlanması durumunda, sonuç olarak ~~1~~0~~0~~1~~0~~0~~0~~1~~0~~1 değeri hesaplamasını isteriz.



Yukarıdaki gösterim bir mealy makinesidir. Makinede görüldüğü üzere 3 farklı durum arasındaki geçişler üzerine iki adet değer yazılmıştır. Bu değerlerden ilki giriş ikincisi ise çıkış değeridir.

Makineyi beraber okumaya çalışalım.

Makinenin boşluktan bir ok ile başlayan durumu, yani örneğimizdeki S0 durumu, başlangıç durumudur. Bu durumdan başlanarak gelen değerlere göre ilgili duruma geçilir.

Örneğimizi hatırlayalım. Giriş olarak 101011 dizgisini (string) almayı planlamıştık. Bu durumda ilk bitimiz 1 olarak geliyor ve S0 durumunda 1 girişi ile S2 durumuna geçiyoruz. Burada geçiş sırasında kullanılan kenarın üzerindeki değeri okuyalım: 1/0 bunun anlamı 1 geldiğinde geçilecek kenar olması ve çıktının 0 olmasıdır. Yani şu anda çıktımız 0

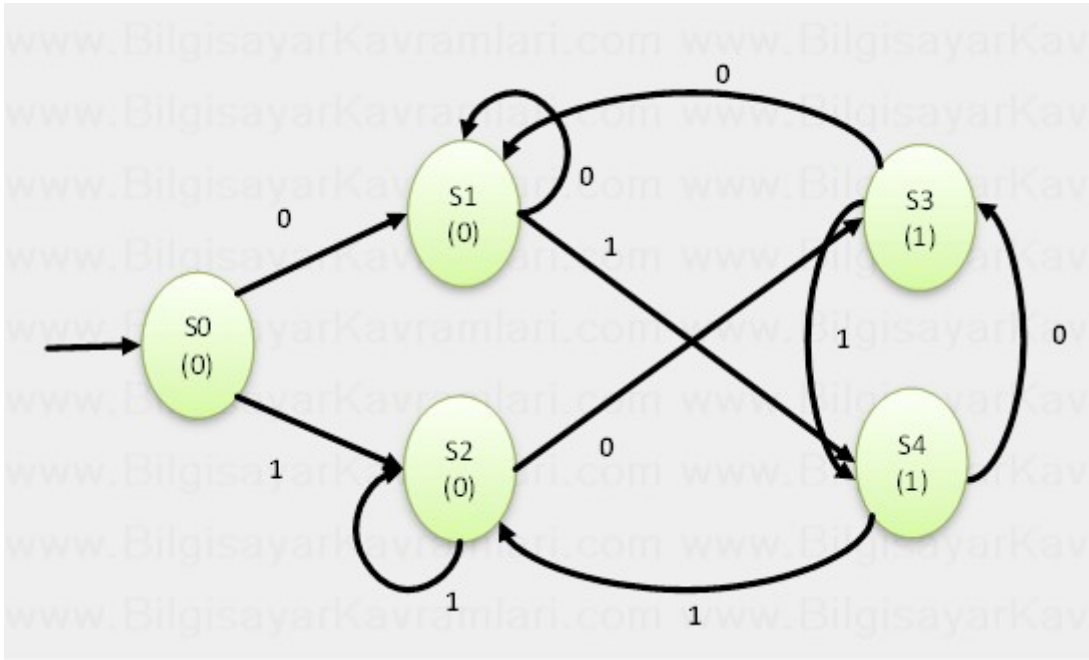
Ardından gelen değer 0 (yani şimdiye kadar 01 değerleri geldi). En son makinemizdeki durum, S2 durumuydu, şimdi yeni gelen değeri bu durumdaki kollardan takip ediyor ve S1'e giden 0/1 kenarını izliyoruz. Bu kolu izleme sebebimiz, S2 durumundan gidilen tek 0 girdisi kolu olmasıdır. Bu kol üzerindeki ikinci değer olan 1 ise, çıktının 1 olduğudur. Yani buraya kadar olan girdiyi alacak olsaydık 01 için 1 çıktısı alacaktık.

İşlemlere devam edelim ve durumları yazmaya çalışalım:

Gelen Değer	Durum	Çıkış	Yeni Durum
1	S0	0	S2
0	S2	1	S1
1	S1	1	S2
0	S2	1	S1
1	S1	1	S2
1	S2	0	S2

Yukarıdaki tablomuzun son halinde, çıkış değeri olarak 0 okunmuştur. Yani örneğimizin neticesi 0 olacaktır.

Aynı örneği moore makinesi olarak tasarlayacak olsaydık:



Moore makinesinde, mealy makinesine benzer şekilde boşluktan gelen bir ok, başlangıç durumunu belirtir.

Makinenin, durumlarında, mealy makinesinde olmayan değerler eklenmiştir. Bu değerler, ilgili durumdaki çıktıyı gösterir. Örneğin makinemiz S1 durumundayken çıktı 0 olarak okunabilir.

Şimdi moore makinesinde, aynı örneği çalıştırıp sonucu karşılaştıralım.

Giriş olarak 101011 dizgisini (string) almayı planlamıştık. Bu durumda ilk bitimiz 1 olarak geliyor ve S0 durumunda 1 girişi ile S2 durumuna geçiyoruz. Bu geçiş sonucunda geldiğimiz S2 durumunda okunan çıktı değeri 0 yani sonuç şimdilik 0.

Ardından gelen 0 değeri ile S3 durumuna geçiyoruz ve çıktımız 1 oluyor. Çünkü S3 durumu 1 çıktısı veren durumdur. Bu şekilde durumları ve durumlar arasındaki geçişleri izlersek, aşağıdaki tabloyu çıkarabiliriz:

Gelen Değer	Durum	Çıkış	Yeni Durum
1	S0	0	S2
0	S2	1	S3
1	S3	1	S4
0	S4	1	S3
1	S3	1	S4
1	S4	0	S2

Görüldüğü üzere, makinemiz, örnek girdi için S2 durumunda sonlanıyor ve bu durumda çıktımız 0 olarak okunuyor.

SORU 22: Birbirini Dışlama (Mutually Exclusive)

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.

SORU 23: CFS (Completely Fair Scheduling, Tam Adil Zamanlama)

Linux 2.6.23 sürümünden sonra çekirdekte (kenel) kullanılan zamanlama algoritmasıdır (CPU Scheduling). Algoritmanın özelliği, CPU meşguliyetini (CPU Utilisation) azami seviyeye getirmek ve işlemciden azami derecede istifade etmektir.

2.6.23 sürümünden önce Linux çekirdeğinde kullanılan O(1) zamanlama (O(1) scheduling) algoritması, performans kriterini sistemdeki bekleme sırası (ready queue) üzerine kurmaktaydı. CFS algoritması ise işlemlerin bekletildiği veri yapısını, kırmızı-siyah ağacına (red black tree) çevirmiştir.

Ayrıca CFS algoritmasında, zaman aralıkları nano saniyeler cinsinden hesaplanmakta ve dolayısıyla çok küçük zaman aralıklarında yapılan işlemler, sezgisel algoritmaların (heuristic algorithm) kullanılması gereğini ortadan kaldırmaktadır.

Zamanlama yaklaşımında, ayrıca işlem grupları oluşturulmakta ve gruplar arasında adalet sağlanması hedeflenmektedir. Bu sayede kullanıcı ile iletişimde olan masa üstü uygulamaları gibi uygulamalara daha fazla öncelik verilerek, arka planda çalışan daha düşük öncelikli sunucu görevlerinin, işlemci üzerindeki yükü azaltılmaktadır.

SORU 24: Overhead (Ek Yük)

Genel olarak bir işin yapılması için, gereken ek maliyetlere verilen isimdir. Örneğin bir kamyonun, bir yükü taşıması için, kendisini de taşıması gerekir. Kendisini taşımasının maliyeti, bu işlemdeki ek yüküdür (overhead).

Bilgisayar bilimlerinde, çeşitli alanlarda farklı anlamlarla kullanılmaktadır.

Örneğin veri iletişimi (network) konusunda ek yük (overhead) denildiğinde genelde bir veriyi iletmek için kullanılan teşrifatin (protokol) kendi içindeki ilave haberleşmeleri kast edilir. Örneğin TCP/IP protokolünü ele alalım. Veriyi doğrudan iletmek yerine öncelikle 3 yönlü el sıkışma (three way hand shaking) işlemi ile taraflar arasında iletişim kurulur. Ardından her taşınan veri için TCP/IP paketinin başlık ve sonluk bilgileri de taşınır (ki bu bilgilerin içerisinde örneğin paketin nereden gelip nereye gittiği bilgisi bulunur). İşte veri iletmek için ayrılan kanal veya kaynakların bir kısmının, veriyi taşımak yerine protokole özgü ilave bilgileri taşıması genelde veri iletişimi (network) açısından ek yük (overhead) olarak isimlendirilir.

Programlama dilleri açısından çağırma ek yükü (call overhead) ismi verilen kavram, bir fonksiyonun çağırılması sırasında yaşanan ek yüküdür. Bir programlama dilinde, herhangi bir fonksiyon çağırıldığında, bu fonksiyonun çalışması sonucunda döneceği yer bir yığında (stack) tutulur. Fonksiyon çalışıp işi bittikten sonra program akışına bu noktadan devam edilir. Bu şekilde fonksiyonlar birbirini çağırarak devam edebilir. Örneğin A fonksiyonu B'yi, B fonksiyonu C'yi ... şeklinde birbirilerini çağırdıklarında fonksiyonun çalışması için gereken yere ilave olarak fonksiyon bilgisi için ilave bir yer tutulması gerekir. Bu yere ve bu yer üzerine yapılan işlemlere çağırma ek yükü (call overhead) ismi verilir.

İşletim sistemleri açısından zamanlama ek yükü (scheduling overhead) ismi verilen kavram, bir zamanlama algoritmasının, işlemler arasında geçiş yaparken kaybettiği kaynaklardır. Örneğin kesintili zamanlama (preemptive scheduling) kullanılan algoritmalarda bu ek yük miktarı (overhead) artacaktır.

SORU 25: Zaman Çizelgeleme (TimeTabling)

Klasik bir optimizasyon problemidir. Basitçe belirli bir süreye, en verimli şekilde belirli kurallara uyarak olayları yerleştirmeyi hedefler. Örneğin öğrencilerin haftalık programının yapılması, doktor / hemşirelerin nöbet çizelgeleri, televizyon kanallarının yayın akışları gibi.

Daha basit anlaşılacağı için öğrenci programlarını örnek vererek konuyu açıklamaya çalışayım. Bir öğrenci probleminde, sistemin uyması istenen kısıtlar(constraints) şunlar olabilir: Örneğin öğrencilerin haftada 7 adet ders almaları gerekiyor. Bu dersler 3'er saat olsun ve her ders 2+1 şeklinde bölünecek olsun. Yani 3 saat üst üste olmayacak 2 saat farklı bir günde 1 saat farklı bir günde olacak. Dersi veren hocalardan 1. Dersi veren hoca, pazartesiye gelemiyor olsun. 3. Dersi veren hoca aynı zamanda farklı bir sınıfın farklı bir dersini veriyor olsun. ...

Yukarıda bu saydıklarımız klasik bir öğrenci programı hazırlanması sırasında yaşanan problemlerden sadece bazılarıdır. Bilgisayar mühendisliği, bu problem için çözüm geliştiren programların yazılması ile ilgilenir ve bu problem, tanımı itibariyle NP-Complete sınıfında kabul edilebilir. Yani problemin çözülmesi polinom olmayan bir zamanda üstsel olarak artan hesaplama gerektirir. Örneğin n adet kistasın olduğu (constraint) bir sisteme yeni bir kistas girildiğinde, sistemin daha önceki n kistas ile bu yeni girilen kistasın uyumunu kontrol etmesin ve çakıştırmamaya çalışması gerekir. Görüldüğü üzere her yeni kistas kendinden öncekilerin çarpımı kadar, yani basit bir hesaplama ile $n!$ kontrol gerektirir.

Zaman çizelgeleme problemlerinin çözümü için çeşitli seviyelerde yöntemler geliştirilmiştir. Bunlardan ilki, problemin modellenmesidir. Problem modellemesi için kullanılan yöntemlerden birisi de TTML (time tabling markup language) kullanımıdır.

Problem çözümü için kistas programlama (constraint programming) yaklaşımları kullanılabilir.

Ayrıca problemin verimli bir çözümü olup olmadığı veya hangi yöntemin kullanılması gerektiği bilinmediği durumlarda, genetik programlama yaklaşımları olumlu sonuçlar verebilmektedir. Bu yüzden literatürde, zaman çizelgeleme problemlerinin genetik algoritmalar ile çözümüne de rastlanmaktadır.

SORU 26: Dolanık Kubitler (Entangled Qubits)

Kuantum mekaniği üzerinde yapılan çalışmalar göstermiştir ki, dolanık kubitler (entangled qubits) birbiri ile özel bir bağa sahiptir ve bazı kurallara uyarlar. Buradaki bağlantıya çevirim ilişkisi (spin correlation) ismi de verilir.

Basitçe iki kuantum parçacığının aynı anda üretildiğini düşünelim, birisinin yukarı çevirim değerinin ise aşağı çevirim olduğunu düşünelim. Bu durumda iki kuantum parçacığının dolanık olması söz konusudur ve bu iki parçacığın sürekli olarak ters çevirimde olduğu kabul edilir ve bu duruma çevirim ters ilişkisi (spin anti-correlated) ismi verilir.

Dolanık kubitlerin bir özelliği de bu ilişkilerini her yerde göstermeleridir. Literatürde bu durum için yerel olmayan (non-local) terimi kullanılır. Yani ilişkileri, herhangi bir yerle sınırlı değildir ve kubitlerin ayrı yerlere yollanması durumunda da ilişkileri bozulmaz.

Kuantum haberleşmesi ve güvenli veri iletimi için oldukça önemli olan bu özellik, üretilen iki kubitin farklı yerlerde aynı özelliği göstermesinin yanında, kubitlerden birisinin açılması ve okunması halinde, bu kubitin bozulmasına da dayanmaktadır.

Kubit ölçümleri, kubitin bilgisini bozmaktadır. Bunun anlamı, bir kubit sadece bir kere okunabilir ve bu okumadan sonra kararsız hale gelir.

Şayet elimizde dolanık iki kubit bulunuyorsa, bu kubitlerden birisi okunduğunda artık dolanıklık özelliğini yitirir. Dolayısıyla iki kubit üretilip, bu kubitlerden birisinin karşı tarafa yollanması durumunda, yolda bir saldırgan tarafından okunursa artık orijinal kubit ile olan ilişkisi bozulmuş olur.

SORU 27: Dirac Göserimi (Dirac Notation)

Kuantum hesaplamasının gelişmesi ile birlikte, kubit (qubit) kavramını göstermek için bir notasyona ihtiyaç duyulmuştur. Bu ihtiyaç Dirac tarafından geliştirilen bir gösterimle karşılanabilmektedir. Bazı kaynaklarda bra-ket olarak da geçer.

Bra-ket gösterimi $\langle | \rangle$ şeklinde sembolize edilebilir. Buradaki bra kısmı $\langle |$ olurken ket kısmı $| \rangle$ olmuş olur. Yani İngilizcedeki parantez anlamına yakın bir kelimeyi parçalara bölerek (aslında barcket kelimesi, İngilizcede parantez anlamına gelir), parantezi iki alt parçada gösterebiliriz.

Bu gösterim basitçe elimizdekiler ve istediklerimizi ayırarak göstermeye yarar.

Elimizde olanları ket kısmına koyuyoruz. Örneğin $|p\rangle$ gösterimi, parçacığın p momentumunda olduğunu ifade etmektedir. Daha farklı belirgin olarak $|p=2.1\rangle$ gösterimi,

parçacığın 2.1 momentumuna sahip olduğunu veya $|x=2\rangle$ gösterimi, parçacığın 2 konumunda bulunduğunu ifade eder. Bu anlamda, elimizdeki bilgileri gösteren ket kısmı, aslında başlangıç vektörü veya başlangıç durumu şeklinde de adlandırılabilir.

Öte yandan $\langle|$ bra gösterimi ise ulaşmak istediğimiz hali, veya beklediğimiz durumu göstermeye yarar. Örneğin $\langle x=1.25|$ gösterimi bize, parçacığın, 1.25 konumunda bitmesini istediğimizi veya böyle bir beklentimiz olduğunu gösterir. Bu durumda, örneğin $\langle x=1.25 | x=2 \rangle$ gösterimi, parçacığın 2 konumunda başlayarak 1.25 konumunda bitmesi anlamına gelir.

Genelde mevcut durumu ifade etmek için ket kısmında ψ sembolü kullanılır. Örneğin $|\psi\rangle$ gösterimi, mevcut durumun ψ vektörü olduğunu ifade eder.

Kubitler için olası durumlardan iki tanesi 1 ve 0 olma durumudur ki bu durumda kubitler bizim bildiğimiz klasik bitler gibi davranır. Bu durumları göstermek için $|0\rangle$ veya $|1\rangle$ gösterimi kullanılabilir. Elbette unutulmaması gereken bir durum, kubitlerin, klasik bitlerden farklı değerler alabileceğidir. Örneğin kubitler, 0 ve 1 arasındaki herhangi bir doğrusal değeri alabilir.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

Şeklindeki gösterimde, ψ değeri, yukarıda verilen α değeri kadar 0 ve β değeri kadar 1'dir. Yani bu iki değer arasında bir yerde kabul edilen bir vektördür. Bu vektörün uzunluğunu 1 olarak kabul edersek, Pisagor bağlantısından $|\alpha|^2 + |\beta|^2 = 1$ olmalıdır.

Ket gösterimi, vektörel bir gösterimdir. Diğer bir deyişle, $|\psi\rangle$ gösterimi aslında $[\psi]$ şeklinde gösterilebilen bir kolon vektördür.

Bra gösterimi ise satır vektörüdür.

Örneğin ket gösterimi için aşağıdaki şekilde bir vektörden bahsedilebilir:

$$|\psi\rangle = [a_1 \ a_2 \ a_3 \ \dots \ a_n]$$

Benzer şekilde bra gösterimi için yukarıdaki bu matrisin tersyüzü (transpoze) alınmıştır denilebilir:

$$\langle \varphi| = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

SORU 28: Amdahl Kuralı (Amdahl's Law)

Çok işlemcili ortamlarda, paralel çalışma sonucunda elde edilebilecek azami kazancı tahmin etmek için kullanılır. Gene Amdahl tarafından geliştirilen bu kurala göre paralel çalışma sonucunda zaman kazanımı formüllemiştir.

Basit bir örnekle, 100 saatlik çalışmanın %20'lik kısmı paralel hale getirilebiliyorsa, %80'lik kısım normal çalışacak bu durumda, algoritma en iyi paraleleştirmeye bile tabi olsa en fazla 20 saatlik zaman kazanılabilecektir.

Buradan anlaşılacağı üzere, çalışma süresi hesaplanırken, çalışmanın paralel kısmına harcanan zaman ve tekil kısmına harcanan zaman toplanmalıdır:

$$T = T_s + T_p \text{ (} T_s : \text{tekil zaman (single) , } T_p : \text{paralel zaman (parallel))}$$

$$\frac{1}{(1 - P) + \frac{P}{T}} \text{ (} P: \text{Paralel, } T: \text{Tekil süre olmak üzere)}$$

Yukarıdaki formülde, paralelleştirme sonucunda kazanılan süre hesaplanmıştır.

Örnek

Yukarıda verilen formülün nasıl kullanıldığını bir örnek üzerinden anlamaya çalışalım.

Örneğin programımızın %70'lik kısmını, normalden 20 misli hızlı çalıştırabiliyor olalım.

$$\frac{1}{(1 - 0.70) + \frac{0.70}{20}} = 2.985$$

Olarak hızlanmayı bulabiliriz. Diğer bir değişle, programın %70'lik kısmını 20 misli hızlı çalıştırabilirsek, toplamdaki hız artışıımız yaklaşık 3 misline çıkmaktadır.

SORU 29: Bilgi Getirimi ve Çıkarımı (Information Retrieval and Extraction)

Bilgisayar bilimlerinin yaygınlaşması ve kullanım alanlarının artması ile birlikte gündeme gelen bir konudur. Bilgi getirimi konuları üzerine yapılan çalışmaların gelişmesi sonucunda, artık lisans seviyesinde, üniversitelerde okutulan bir ders halini almıştır.

Bilgi getirim çalışmalarının amacı, genellikle düzenli bir yapıya sahip olmayan bir kaynaktan, istenilen yapıda bilginin elde edilmesidir. Örneğin bir metin içerisinde, metnin hangi konu ile ilgili olduğunun çıkarılması, bir bilgi getirimidir.

Bilgi getirimi, günümüzde özellikle internet üzerinde çalışan arama motorları sayesinde oldukça popüler olmuştur. Aslında bilgi üzerinde yapılan her arama bir bilgi getirimi kabul edilebilir. Örneğin elektronik postalarınız arasında yaptığınız bir arama da bilgi getirimidir.

Getirim yapılacak ortamın hazırlanması da bilgi getiriminin bir parçasıdır. Bu hazırlama işlemine özel olarak bilgi çıkarımı (information extraction) ismi de verilir. Örneğin getirim yapılacak dokümanların sınıflandırılması, bu dokümanlarda bulunan bilgilerin indekslenmesi gibi işlemler, ileride yapılacak getirim işlemlerine zemin hazırlar. Bilgi getirimi, üzerinde çalıştığı veri ortamına göre, yapılandırılmamış, yarı yapılandırılmış veya tam yapılandırılmış gibi sınıflara ayrılabilir. Bilgi getiriminin yapıldığı verinin büyüklüğüne göre, farklı ölçeklerde de incelenebilir.

Örnek

Konunun daha iyi anlaşılması açısından, bir gerçek hayatta belki de pek çok kişinin farkında olmadan yaptığı bir bilgi getirimini ele alalım. Herkesin tanıdıklarını ve tanıdıklarının

tanıdıklarını bir matriste tuttuğunu düşünelim. Yani siz, arkadaşlarınızı bir matriste tutuyorsunuz ve arkadaşlarınızın birbirini tanıyıp tanımamasını da bu matriste işaretliyorsunuz.

	Ali	Ahmet	Ayşe	Mehmet	Veli
Ali	1	0	1	1	0
Ahmet	0	1	1	0	1
Ayşe	1	1	1	1	0
Mehmet	1	0	1	1	0
Veli	0	1	0	0	1

Yukarıdaki tabloda, birbiri ile tanışmış olanları 1, birbirini tanımayanları ise 0 ile işaretledik. Unutulmaması gereken bir nokta, bu bilginin, bilgisayar tarafından işlenebilir olmasıdır. Yukarıdaki tabloda ikilik tabanda sayılar kullanılmış ve veri getirmenin yapılması kolaylaştırılmıştır. Bu tip bilgi getirimlerine ayrıca ikilik bilgi getirimi (binary information retrieval) ismi de verilir.

Yukarıdaki tabloya bakarak, örneğin bir organizasyona arkadaşlarınızı çağırarak istiyorsunuz ve işlerin hızlı yürümesi için, herkesi siz çağırarak yerine en az kişiye ulaşıp herkesi haberdar etmek istiyorsunuz. Bu durumda en verimli çözüm, tek kişiyi aradığınız ve bu tek kişiden oluşan bir zincirin dolaşılabilmesidir.

Örnek zincirimize göre, örneğin siz Ali'yi arayarak başlarsanız, Ali → Ayşe → Ahmet → Veli şeklinde bir zincirden sonra Mehmet'e ulaşamamaktadır. Bir çözüm olarak Ayşe hem Ahmet'e hem de Mehmet'e haber verebilir veya alternatif bir zincir şu şekilde olabilir: Mehmet → Ali → Ayşe → Ahmet → Veli, dolayısıyla herkes tek bir kişiyi arayarak herkese ulaşılmış olur. Elbette bu zincirin oluşturulmasının, bilgisayarlar tarafından yapılması mümkündür ve bunun için bilgisayarın işleyebildiği bir veri modeline ihtiyaç vardır. Yukarıdaki bu ikilik tabandaki tablo da, bize bu veri modelini sunar.

Örneğin bir romandaki kişiler arasında yukarıdakine benzer bir tablo hazırlanmak isteseydi. Bu tabloyu romandan otomatik olarak çıkaran bilgisayar yazılımı, bilgi çıkarımı (information retrieval) yapmış olacaktı.

Bilgi Getirimi Formülleri

Bilgi getirimi çalışmalarında ayrıca iki önemli ölçüm kriteri bulunur.

- Kesinlik (Precision) : Getirilen bilginin ne kadarı, istenilen bilgiyle ilgilidir?
- Hassasiyet (Recall): Getirilmesi gereken bilginin ne kadarı getirilmiştir?

Buradaki kesinlik kavramı genelde p harfi ile gösterilir ve getirilen bilgidaki doğru sonuçların, getirilen bilginin tamamına oranı olarak hesaplanır.

$$Kesinlik(Precision) = \frac{\{ilgili\ getirim\} \cap \{bütün\ veri\ çıkarımı\}}{\{bütün\ veri\ çıkarımı\}}$$

Hassasiyet kavramı da genelde r harfi ile gösterilir ve getirilen doğru sonuçların, getirilmesi gereken doğru sonuçlara oranı ile hesaplanır.

$$Hassasiyet(Recall) = \frac{\{ilgili\ getirim\} \cap \{bütün\ veri\ çıkarımı\}}{\{ilgili\ veri\ çıkarımı\}}$$

Yukarıdaki bu tanımlar ışığında, F_1 skoru, bu değerlerin harmonik ortalamasıdır (harmonic mean):

$$F_1 Skoru = 2 \frac{Kesinlik \cdot Hassasiyet}{Kesinlik + Hassasiyet} = 2 \frac{pr}{p + r}$$

F skorunun bir β değerine bağlanması da mümkündür. Bu durumda , F_1 skoru yerine F_β skoru terimi kullanılır

$$F_\beta Skoru = (1 + \beta^2) \frac{Kesinlik \cdot Hassasiyet}{\beta^2 \cdot Kesinlik + Hassasiyet} = (1 + \beta^2) \frac{pr}{\beta^2 \cdot p + r}$$

Ters İndeks (Inverted Index)

Bilgi getirimi konusundaki önemli kavramlardan birisi de indekslemedir. Normalde bir metin içerisindeki bütün bilgiler indekslidir. Yukarıda verdiğimiz ve bir romanda geçen kahramanları hatırlayalım. Bu kahraman bilgileri roman içerisinde geçtikleri sayfa itibarıyla indekslidir, yani romanı okuyan birisi kolaylıkla aşağıdakine benzer bir tablo hazırlayabilir:

1. Sayfada Ali ve Ahmet'ten bahsediliyor.
2. Sayfada Ayşe ve Ali'den bahsediliyor.

...

Bu indeksleme anlamlı olduğu gibi, bazı durumlarda kullanışsız olabilir. Örneğin Ali, romanın kaç sayfasında zikredilmektedir veya Ali'nin geçip Ahmet'in geçmediği bir yer var mıdır? Veya Ayşenin, romanda ilk kez geçtiği sayfa hangisidir gibi sorular sorulursa, yukarıdaki indeks üzerinde ilave işlemler yapılması ve arama algoritmalarının kullanılması gerekir.

Bunun yerine, kişi bazlı aramaları kolaylaştırmak için, yukarıdaki indeksi ters çevirmek mümkündür.

Ali → 1 , 2

Ahmet → 1

Ayşe → 2

Şeklinde herkesin geçtiği sayfa numaralarını indekslersek, yukarıda sıraladığımız soruların hepsine çok daha hızlı cevaplar verebiliriz.

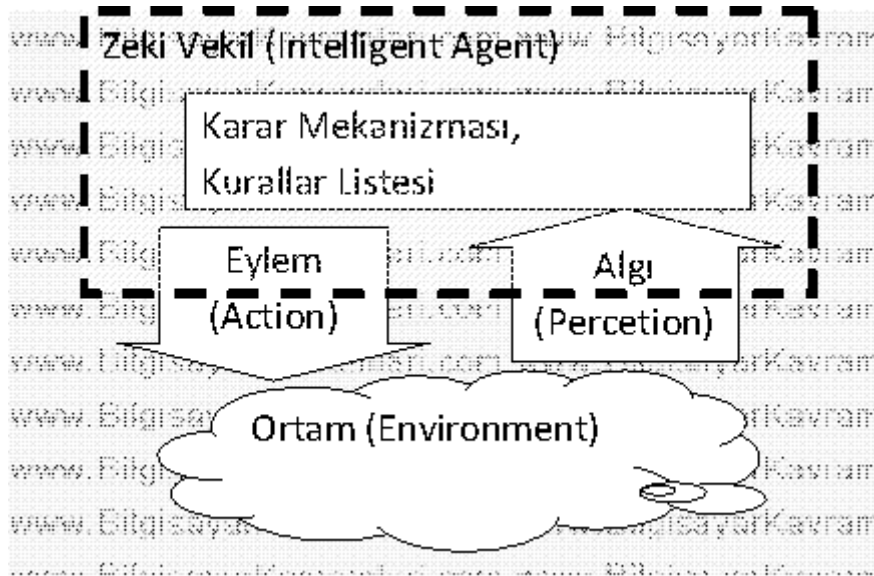
Buradan anlaşılacağı üzere, sorgulanan verinin tipine göre, indeksin dönüştürülmesi gerekebilir ve şayet normalde verinin üzerinde çalıştığı indeks, hızlı erişim gibi kaygılardan dolayı ters çevrilirse bu yeni indekse ters indeks ismi verilir.

SORU 30: Zeki Vekiller (Akıllı Ajanlar, Intelligent Agents, Zeki Etmenler)

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

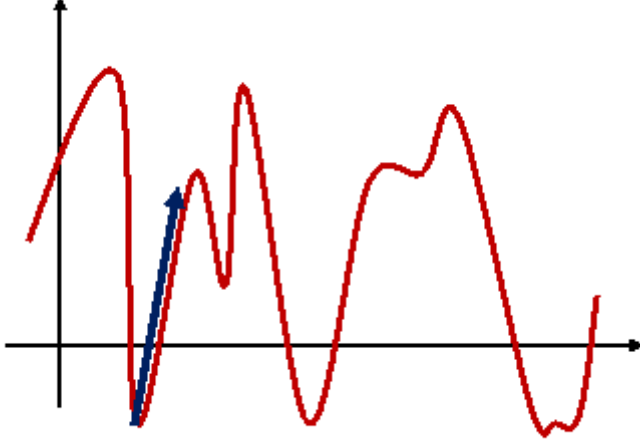
Hedef amaçlı vekillerden farklı olarak, durumlar arasındaki geçişin oransal olması durumudur. Yani hedef güdümlü ajanlarda bir durumdan her zaman diğer duruma geçiş hedeflenir. Çıkar amaçlı vekillerde bundan farklı olarak oransal bir fonksiyon kullanılması söz konusudur. Bu fonksiyona çıkar fonksiyonu (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 kullanılışlıdır. Kendi kurallarını ve durum makinelerini oluşturabilir veya değiştirebilirler.

SORU 31: Tepe Tırmanma Algoritması (Hill Climbing Algorithm)

Bilgisayar bilimlerinde kullanılan [arama algoritmalarından](#) birisidir. Arama işleminin yapıldığı grafikteki tepelerden ismini alır. Basitçe bir grafikte bulunan en düşük noktanın aranması sırasında grafikte yapılan hareketin aslında tepe tırmanmaya benzemesinden ismini almaktadır.



Örneğin yukarıdaki şekilde gösterilen ok temsili bir tepe tırmanma işlemidir. Burada arama yapan algoritma aslında bir çukur bulmuş ancak daha iyisi için tepe tırmanmaktadır denilebilir.

Elbette olayı iki boyutlu bir grafik üzerinde yapılan bir arama veya tırmanılan bir tepe olarak görmek, algoritmanın önemini kavramayı engelleyebilir. Aslında burada anlatılmak istenen bilgisayar bilimleri de dahil olmak üzere, çeşitli bilim ve mühendislik alanlarında, birden fazla çözümü olan problemler için en iyi çözümün arandığı iyileştirme (optimization) problemleridir.

Yani şayet bir sistemin ya da bir programın daha iyi hale getirilmesi isteniyorsa, sistemin verdiği sonuçlara göre arama işlemi yapılarak iyileştirme amaçlanabilir. İşte bu noktada tepe tırmanma algoritması (hill climbing algorithm) de dahil olmak üzere çeşitli arama algoritmaları devreye girer. Örneğin literatürde sıkça kullanılan [seyyar tüccar problemi \(travelling salesman problem\)](#) bu problemlerden birisidir. Tepe tırmanma algoritması verilen bir harita için verilen bir sonucu iyileştirmeye çalışır. Elbette amaç bütün ihtimalleri denemeden iyi bir sonuç bulabilmektir.

Tepe tırmanma algoritması, arama algoritmaları arasındaki en iyi sonucu verene algoritma değildir. Ancak kodlanması ve tasarımının basit oluşundan dolayı sıklıkla kullanılır.

Tepe tırmanma algoritması çeşitleri

Algoritmanın üzerinde çeşitli iyileştirmeler yapılarak daha iyi sonuçlar elde edilmeye çalışılmıştır. Literatürde sıkça kullanılan bir iki tepe tırmanma algoritmasını farkları ile birlikte açıklamaya çalışalım.

Klasik tepe tırmanma algoritmasında amaç arama için merkez kabul edilen (veya başlangıç noktamız diyebileceğimiz) noktadan komşu olan noktaları gezerek daha iyi sonuçlar elde etmektir. Temel olarak bir grafikte rastgele seçilen bir nokta için 3 farklı ihtimal bulunmaktadır:

1. Noktanın bir tarafında problem iyileşirken diğer tarafında kötüleşmektedir. Bu durumda iyi tarafa doğru tırmanma algoritmamız devam eder.

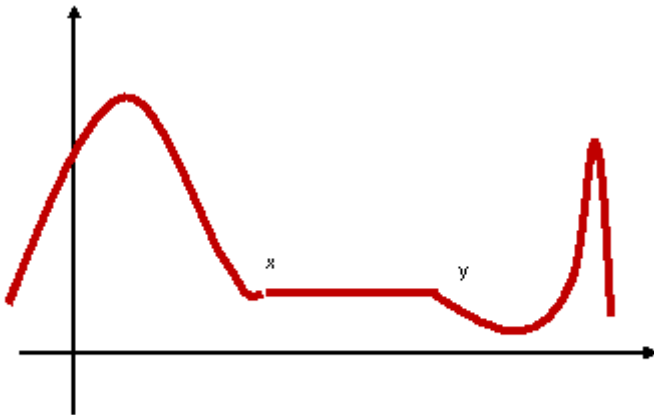
2. Noktanın iki tarafında da problem sonucu kötüleşmektedir. Bu durumda bulunduğumuz nokta problem için en iyi noktalardan (optimum points) birisidir. Elbette bu en iyi sonuç olmayabilir yani bu sonuçtan daha iyi sonuçlar olabilir ancak klasik tepe tırmanma algoritması bu diğer sonuçları bulamaz ve bu noktada kalır.
3. Noktanın iki tarafında da problem iyileşiyordur. Yani tesadüfen bulunduğumuz nokta aslında problem için ulaşılabilir en kötü noktalardan birisidir. Bu durumda tepe tırmanma algoritması yönlerden birisini seçerek tırmanmaya devam eder. Farklı bir çeşit olarak iki yöne de tırmanan algoritma bulunmaktadır.

Klasik tepe tırmanma algoritmasından farklı olarak steepest ascent hill climbing algoritmasında, bulunabilen bütün sonuçlar arasından bir seçim yapılır. Bu algorithmada da klasik tepe tırmanma algoritmasında da sorun aynıdır. Şayet arama işlemi sırasında bir yerel çukura (local minimum) rastlanılırsa bu durumdan algoritma kendisini kurtarmayarak en doğru sonucu bulamayabilir.

Buna karşılık olasılıksal tepe tırmanma algoritmasında (stochastic hill climbing algorithm) bütün komşuların aranması ve komşuların verdiği sonuca göre hareket etmek yerine, rast gele olarak bir komşunun seçilmesi söz konusudur. Şayet gidilen bu komşu beklenen yönde bir iyileştirme sağlıyorsa, bu yönde aramaya (tırmanmaya) devam edilir, şayet beklenen iyileştirme sağlanamıyorsa, bu durumda daha farklı bir komşu denenir.

Yukarıdaki tepe tırmanma algoritmalarının yanında rastgele başlangıç tepe tırmanma algoritması (random restart hill climbing algorithm) şaşırtıcı derecede iyi sonuç veren bir algoritmadır. Bu algoritma basitçe bir x durumunu başlangıç kabul eder ve daha iyi bir durum bulunca başlangıç durumunu bu daha iyi duruma kaydırır. Algoritma iyi durum buldukça başlangıç durumunu kaydıran ancak bulamadığı durumlarda da aramaya devam eden bir yapıya sahiptir. Rastgele başlangıç tepe tırmanma algoritmasına bazı kaynaklarda pompalı tüfek tepe tırmanma algoritması (shotgun hill climbing algorithm) ismi de verilmektedir.

Tepe tırmanma algoritmaları genel olarak yerel bir başarı noktasında (local optimum point) takılmak gibi bir zafiyete sahiptirler. Daha iyi sonuçlar için [simulated annealing](#) gibi arama algoritmaları kullanılabilir.



Örneğin yukarıdaki şekilde x ve y noktaları arasında bir düzlük bulunmaktadır. Başlangıç noktası olarak bu aralıktaki herhangi bir noktadan başlanırsa algoritma komşuları aradığında daha iyi veya daha kötü bir sonuç bulamayacağı için hatalı karar verebilir.

Ayrıca tepe tırmanma algoritmaları problem sonuçlarında nadiren de olsa aynı sonucun elde edilmesi durumunda belirsizce hatalı sonuçlar verebilir. Bu şekildeki durumlara algoritmadaki düzlükler ismi verilir ve algoritma hangi yöne gidilirse gidilsin daha iyi bir sonuç çıkaramaz (hep aynı sonucu çıkarır). Elbette doğası gereği tepe tırmanmaya hazırlanmış algoritmamız, tepe bulamayınca hata yapmaktadır.

Algoritmanın kodlanması

Algoritma iki boyutlu bir grafik için rastgele yön seçimi yapılması durumunda aşağıdaki şekilde kodlanabilir:

```
1 // www.bilgisayarkavramlari.com
2 i = başlangıçNoktası
3 j = başlangıçNoktası
4 while( i<= j){
5     i= KomşuNokta (i)
6 }
```

Yukarıdaki kodda, basitçe başlangıç noktasından başlanarak, daha iyi sonuçlar bulunduğu sürece bir sonraki komşu noktaya hareket öngörülmüştür.

Ancak problemin çok boyutlu olması yani bir noktanın birden fazla komşusu bulunması durumunda bütün komşuların kontrol edilmesi ve en iyisinin bulunması daha sonra bu komşuya doğru hareket edilmesi gerekeceği için algoritmayı aşağıdaki şekilde yazmak mümkündür:

```
1 // www.bilgisayarkavramlari.com
2 i = başlangıçNoktası
3 j = başlangıçNoktası
4 while( i<= j){
5     int bayrak=1;
6     for(int k = 0;k<komşuSayısı;k++){
7         if( i >= komşuNokta(i,k) ){
8             i = komşuNokta(i,k)
9             bayrak = 0;
10        }
11    }
12    if(bayrak)
13        return i;
14 }
```

Yukarıdaki yeni kodda, bir noktanın birden çok komşusu olduğu kabul edilmiş ve bütün komşuları dolaşarak en iyi komşu alınmış, ardından daha iyi komşu bulunduğunda bu işlem devam ettirilmiştir. Nihayetinde hiçbir komşu, bulunan son komşudan daha iyi olmayınca program sonlandırılmıştır.

SORU 32: Arama Algoritmaları (Search Algorithms)

Bilgisayar bilimlerinde, çeşitli veri yapılarının (data structures) üzerinde bir bilginin aranması sırasına kullanılan algoritmaların genel ismidir. Örneğin bir dosyada bir kelimenin aranması, [bir ağaç yapısında \(tree\)](#) bir düğümün (node) aranması veya bir [dizi \(array\)](#) üzerinde bir verinin aranması gibi durumlar bu algoritmaların çalışma alanlarına girer.

Yapısal olarak arama algoritmalarını iki grupta toplamak mümkündür.

- Uninformed Search (Bilmeden arama)
- Informed Search (Bilerek arama)

Arama işleminin bilmeyerek yapılması demek, arama algoritmasının, probleme özgü kolaylıkları barındırmaması demektir. Yani her durumda aynı şekilde çalışan algoritmalara uninformed search (bilmeden arama) ismi verilir. Bu aramaların bazıları şunlardır:

- Listeler (diziler (array)) üzerinde çalışan arama algoritmaları:
 - [Doğrusal Arama \(Linear Search\)](#)
 - [İkili arama \(binary search\)](#)
 - [İnterpolasyon Araması \(Ara değer araması, Interpolation Search\)](#)
- [Şekiller \(graflar \(Graphs\)\)](#) üzerinde çalışan algoritmalar
 - [Sabit Maliyetli Arama \(Uniform Cost Search\)](#)
 - [Floyd Warshall algoritması](#)
 - [Prim's Algoritması](#)
 - [Kruskal Algoritması](#)
 - [Dijkstra Algoritması](#)
 - [Bellman Ford Algoritması](#)
 - [İkili arama ağacı \(Binary Search Tree\)](#)
 - [Prüfer dizilimi](#)
 - [Ağaçlarda Sığ öncelikli arama \(breadth first search\)](#)
 - [Şekillerde \(Graph\) sığ öncelikli arama \(Breadth First Search, BFS\)](#)
 - [Derin öncelikli arama \(depth first search\)](#)
 - [Derin Limitli Arama \(Depth Limited Search\) Algoritması](#)
 - [Yinelemeli Derinleşen Derin Öncelikli Arama Algoritması \(Iterative Deepening Depth First Search, IDDFS\)](#)
 - [Patricia Ağaçları](#)
 - [Trie Ağaçları \(metin ağaçları, trie trees\)](#)
 - [B-ağaçları \(B-Tree\)](#)
- Metin arama algoritmaları (bir yazı içerisinde belirli bir [dizgiyi \(string\)](#) arayan algoritmalar)
 - [Horspool Arama Algoritması](#)
 - [Knuth-Morris Prat arama algoritması](#)
 - [Boyer-Moore Arama algoritması](#)
 - [Kaba Kuvvet Metin Arama Algoritması \(Brute Force Text Search, Linear Text Search\)](#)
 - [DFA Metin Arama Algoritması](#)

Arama işleminin bilerek yapılması ise, algoritmanın probleme ait bazı özellikleri bünyesinde barındırması ve dolayısıyla arama algoritmasının problem bazlı değişiklik göstermesi demektir. Bu algoritmaların bazıları aşağıda listelenmiştir:

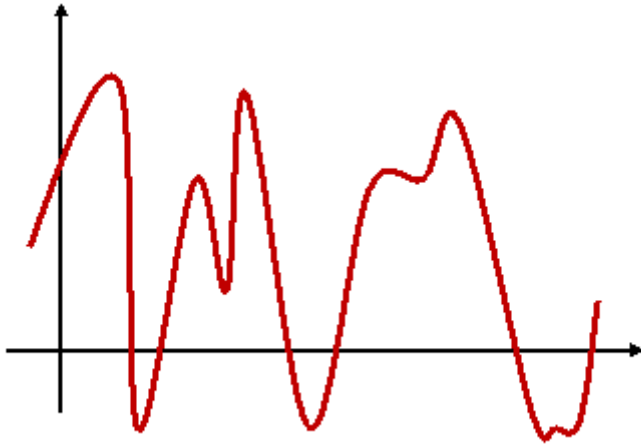
- [Minimax Ağaçları](#)
- [Simulated Annealing \(Benzetimli Tavlama\) algoritması](#)
- [Tepe Tırmanma Algoritması \(Hill Climbing Algorithm\)](#)
- [Arı sürüsü arama algoritması \(bees search algorithm\)](#)
- [A* Araması \(astar search\)](#)
- [Geri izleme \(backtracking\)](#)
- [Işın arama \(beam search\)](#)

SORU 33: Simulated Annealing (Benzetilmiş Tavlama)

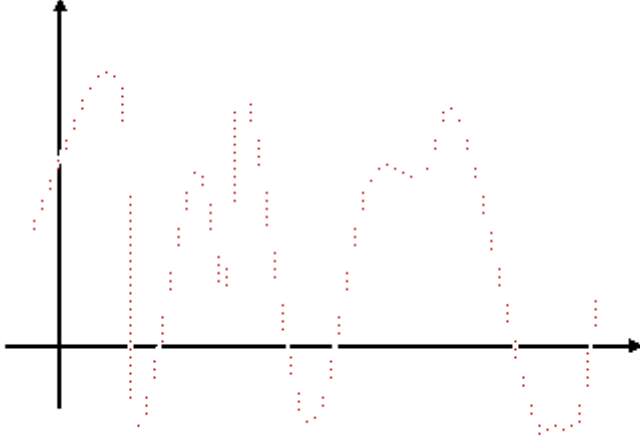
Bilgisayar bilimlerinde, özellikle hesaplama alanında kullanılan algoritmalarından birisidir. İsmi demir tavlama veya demiri ısıtmak anlamına gelen annealing (tavlama) kelimesinden almıştır. Algoritmanın amacı, herhangi bir problem için genel iyileştirme (global optimization) elde etmektir. Diğer bir deyişle, herhangi bir fonksiyonun ya da ölçümün genel minimum veya maksimum (global minimum) elde etmek olarak düşünülebilir.

1. Simulated annealing (benzetilmiş tavlama) nerede işe yarar?

Genellikle hesaplamalı bilimlerde bir deneyin yada nümerik sonuçların anlık değerleri elde edilir. Örneğin aşağıdaki grafiği ele alalım:



Yukarıdaki şekil herhangi bir indis değerlerine sahip fonksiyon olabilir. Yukarıdaki 2 boyutlu grafiğin indisleri yada formüllendirilmesi ile ilgilenmeden grafiğin en küçük olduğu değeri bulmaya konsantre olalım. Şayet yukarıdaki grafiği veren bir formül elimizde varsa, bu formülün üzerinde çeşitli matematiksel yöntemler uygulayarak (örneğin türev alarak) bir sonuç elde edebiliriz. Ancak yukarıdaki grafiği veren bir formülümüzün olmadığını ve yukarıdaki değerleri ancak belirli aralıklarla yapılan ölçümler sonucu elde ettiğimizi düşünelim.



Örneğin yukarıdaki noktalar ile gösterilen değerlerin ölçüldüğü ayrık bir (discrete) sistemimiz olduğunu düşünelim. Bu duruma yapay zeka problemlerinde sezgisel sonuçların (heuristic) elde edildiği problemlerde veya gerçek hayattaki farklı zamanlarda yapılan ölçümlerde sıkça rastlanır. Benzer bir durum olarak ölçme maliyetinin yüksek olduğunu da kabul edilebilir. Örneğin her ölçümün yüksek maliyet getirdiği ve dolayısıyla kısıtlı sayıda ölçüm yaparak en düşük değeri bulacağımız x eksenini değerini aradığımızı düşünelim.

İşet yukarıdaki 2 boyutlu grafikte kullanılabilecek benzetimli tavlama (simulated annealing) algoritması, matematiksel olarak bir fonksiyonla modellenemeyen, nümerik ve ayrık uygulamalarda kullanılabilir. Ayrıca problemin kaç boyutlu olduğunun da bir önemi yoktur. Örneğin 3, 4 veya daha fazla boyutlu problemlerde de kolaylıkla uygulanabilir.

2. simulated annealing (benzetilmiş tavlama) nasıl çalışır?

Algoritmanın çalışması aslında isminin de geldiği demir tavlama işlemine benzer. Yani nasıl demir tavlama işlemi sırasında bir demir parçayı ısıtıp sonra soğumaya bırakıyorsak, herhangi bir sayısal ölçüme de benzeri yaklaşım uygulanabilir.

Şayet demir tavlama problemini, demiri oluşturan ufak hücrelerin ısınması ve soğuması olarak düşünecek olursak, ki bu hücreler daha sonra problemimizin örneklendiği her bir deney veya nümerik sonuç olarak modellenecektir, hücreler arası geçiş ve hücrelerin zamana bağlı değerlerini elde etmek mümkündür.

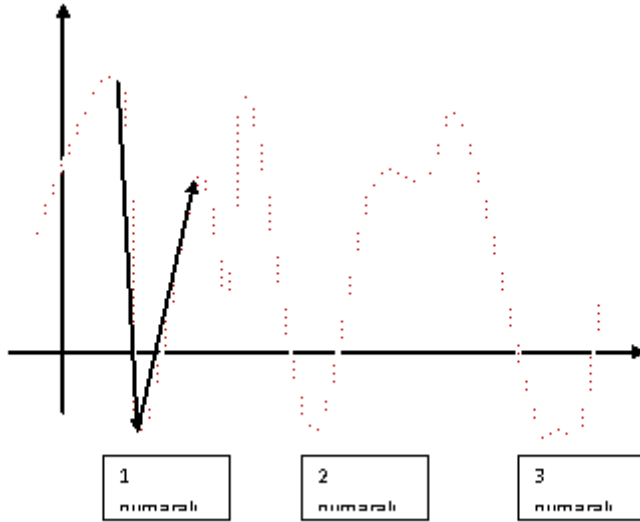
Amacımız genel bir minimum noktası bulmak olduğuna göre, problemin farklı zamanlardaki sonuçlarını ele alıp bu sonuçlardan iyi olanına doğru hareket etmemiz gerekir. Örneğin s durumu ve s' durumları arasından, $P(e, e', T)$ olasılık değerine göre seçim yapılır. Buradaki $e = E(s)$ ve $e' = E(s')$ olarak hesaplanan s ve s' için enerji değerleridir. Yani grafikteki karşılık değerleri veya deneyimizin sonuç değerleri olarak düşünülebilir. T değeri ise ısı olarak geçer ki bunu da deneydeki ölçüm zamanları olarak kabul etmek mümkündür.

Kısacası $P(e, e', T)$ değeri bir olasılık sonucu çıkarır. Bu sonuç bize s durumundan s' durumuna geçişin ne kadar kabul edilebilir olduğunu olasılıksal olarak verir.

Burada daha basit bir açıklama olarak şöyle düşünebiliriz. Şayet e' değeri, e değerinden yüksekse, yani yeni durumumuzdaki enerji daha yüksek bir enerjiyse, sistem soğumuyor

ısınıyor demektir. Bu ise sistemin daha kötüye gittiğinin bir işaretidir ve doğal olarak kabul edilir bir durum değildir.

Ancak sistemin hep iyiye gitmesi hiç kötüye gitmemesini istememiz durumunda yerel minimumla (local minimum) karşılaşırız. Örneğin yukarıda temsili resmi verilen şekli hatırlayalım:



Yukarıdaki şekilde düşey eksenini enerji olarak kabul edersek, soldaki okun gösterdiği değer soğuma olarak düşünülebilir. Yani farklı durumlar için enerji değerleri azalmıştır. İkinci okta ise enerji değerleri artmış yani elde edilen durumlarda ısınma olmuştur. Şimdi yukarıdaki örnek için, şayet ısınan durumları, yani okun yukarı yönlü olduğu durumları kabul etmez ve grafikte sağa doğru hareket etmezsek 1 numaralı çukurda takılıp kalma ihtimali olur. Bu durumda sistemimiz hiçbir zaman, daha iyi sonuçlar olan 2 ve 3 numaralı çukurlara ulaşamayabilir.

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

Bu şekilde elde edilen sonuçlara yerel minimum (local minimum) ismi verilir ve algoritmamızın yukarıda açıklanan şekilde davranması durumuna açgözlü yaklaşımı (greedy approach) ismi verilir. Kısaca ilk bulduğu minimum değerine atlayan ve dolayısıyla sistemdeki daha düşük minimum değerleri bulamayan algoritma haline gelir. (açgözlülük insanı kör ettiği gibi algoritmaları da kör eder)

Bu duruma çözüm olarak e'-e değerindeki değişimler gözlenir. Bu değerler arasındaki değişimlerin artması, bu değerleri üreten s ve s' durumlarının kabulünü zorlaştırır. Ancak iki durumda ölçülen enerji değerlerinin birbirine çok yaklaşması durumunda sistem kabul edilir. Bu sayede sistemin yükseldiği durumlarda da sistem dolaşmaya devam eder ancak enerji değerlerinin birbirine iyice yaklaşması sonucunda minimum değerler bulunmuş olunur.

3. Simulated annealing'in kodlanması

Yukarıdaki tanımı müsvedde kod olarak yazalım:

```
1 durum SimulatedAnnealing(){
2     //www.bilgisayarkavramlari.com
3     s = f(0)          //fonksiyondan ilk değeri al
4     e = E(s)          //enerjisini hesapla
5     Siyi = s          //şimdilik en iyi durum ilk durumdur
6     Eiyi = e          // en iyi enerji de ilk durumun enerjisidir
7     for(i=1;i<=MAX;i++){    //i, ilk durumdan son duruma kadar dolaşsın
8         s' = komsu (s) // s durumunun komşusu s' durumudur
9         e' = E(s') // komşunun enerjisini hesapla
10        if( Eiyi < e'){ //şayet en iyi enerjiden daha iyi bir enerji bulduysak
11            Siyi = s' // en iyi durum s' olsun
12            Eiyi = e' // en iyi enerji de e' olsun
13        }
14        if( P( e, e', T( i / MAX)) > rand () ){
15            // şayet enerjiler ve ısı ile üretilen olasılık değeri
16            // devam edilecek kadar büyükse devam edelim ve yeni durumumuz s' olsun
17            s = s'
18            e = e' // yeni duruma gitmiş olduk
19        }
20    } // maksimum denemeye ulaşıldıysa çıkabiliriz
21    return Siyi; //şimdiye kadar bulduğumuz en iyi sonuç durumunu döndür
22 }
```

Yukarıdaki kod, görüldüğü üzere komşu durumlar üzerinde hareket ederek (yada olasılık durumuna göre etmeyerek) en iyi sonucu bulmaya çalışır.

Yukarıdaki kodun diğer bir özelliği ise sezgi üstü yaklaşımlar için kullanışlı olmasıdır. Dikkat edilirse, algoritmanın taradığı bütün durumlar s ve s' durumları olarak geçmekte ve dilenirse kaydedilebilmektedir. Bu durum normal bir soğuma sürecinden farklı olarak (gerçek hayatta herhangi bir t anında sadece bir durum vardır) çok durumu tutabilme imkanı sunar.

Ayrıca yukarıdaki algorithmada yapılacak deneme sayısı kodun 7. Satırındaki döngü ile limitlenebilmektedir. Dolayısıyla bir kişi en iyi ama en maliyetli çözüm için bu MAX değerini arttırabilir. Elbette en iyi çözüm demek daha çok deneme demek ve bu da yüksek maliyet demektir. Maliyetin düşürülmesi için deneme sayısının kısılması ise, en iyi genel sonucun bulunamaması ihtimalini yükseltir. Elbette şanslıysak ve genel minimum f(0) değeriye hiç deneme yapmamıza da gerek yoktur ◀◀

Benzetilmiş tavlama yönteminin (simulated annealing), bir özelliği de, aynı problem için birden fazla çözümü olmasıdır. Bunun sebebi yukarıdaki kodda da belirsiz olarak bırakılan, komsu(), P(), T() fonksiyonlarının (functions) problemin çözümündeki seçilme şeklidir. Yani aynı problemi iki farklı bilgisayar bilimcisi iki farklı komşuluk fonksiyonu, ısı değeri ve olasılık değerine göre çözebilir. Burada probleme özgü olarak modelleme yapmak ve problemin karakteristiğinin doğru analiz edilmesi önem taşır.

SORU 34: Firmware (Bellenim)

Firmware kelimesi, İngilizcede iki kelimenin birleşmesinden oluşur. Firm anlam olarak şirket ware ise mal anlamına gelir. Örneğin yazılım kelimesinin karşılığı olan software Türkçede

tam çevirim ile yumuşak mal anlamına gelebilir. Burada ware kelimesinin zaman içerisinde (software kelimesinin yaygınlaşması ile) giderek yazılım anlamına kayması söz konusudur. Yani yazılım dünyasında kullanılan çoğu terimde yazılı bir kod veya program için sadece ware eklentisi yeterli görülmektedir. (middleware, software, firmware, shareware, spyware veya warez gibi kelimlerdeki ware kısmı genelde yazılı kod anlamındadır)

Kelimenin etimolojisine hızlıca baktıktan sonra Türkçe karşılığına geçebiliriz. Genelde bilgisayar dünyasında kavramlar ve dolayısıyla kelimeler İngilizceden geldiği için yoğun bir Türkçede İngilizce terimlere karşılık bulma sıkıntısı yaşıyoruz. Genelde de tam olarak karşılığı olan anlamı veremiyoruz. Türkçede firmware kelimesini direk İngilizcedeki gibi kullananların yanında belenim veya gömülü kod terimlerini kullananlarda bulunmaktadır. Ayrıca tam anlamında şirket kodu veya üretici kodu gibi terimlerde kullanılabilir.

Peki nedir firmware?

Firmware en kaba tabiriyle bir cihazın üzerindeki yazılımdır. Üretici firma tarafından üretim aşamasında cihaza yazılır ve genelde de hiç değiştirilmeden cihazın ömrüne paralel bir ömür sürer.

Örneğin günlük hayatımızda arabaların, hesap makinelerinin, uydu alıcılarının, kahve makinelerinin veya dijital kameraların çoğunda böyle bir yazılım bulunur. Nadiren de olsa bu yazılımın güncellenmesi ve yenilenmesi gerekir. Güncelleme işlemi yine genelde firmalar tarafından sağlanan kodların cihaza yüklenmesi şeklinde olur. Yüklemek için ya cihazın üzerinde bulunan özel bir iletişim ara yüzünden faydalanılır veya cihaza donanımsal olarak müdahale gerekir. Örneğin bir arabanın üzerindeki bir çipin kodlanması için bu çipe özel olarak bağlanması gerekir ve bunun için ilave donanıma ihtiyaç duyulur.

Öte yandan bir adsl modem'in üzerindeki yazılım için FTP veya http portlarından erişim sağlanarak kolayca güncelleme yapılabilir.

SORU 35: Base64

Veri güvenliği konusunda kullanılan kodlama (encoding) algoritmalarından birisidir. Basitçe bir bilginin farklı semboller ile gösterilmesi işlemidir. Bu semboller alfabe'deki harflerin büyük/küçük sıralanması ve sayılardan oluşur. Bir base64 sisteminin kullandığı semboller aşağıda verilmiştir:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/-

Yukarıda toplam 64 sembol bulunmaktadır. Dolayısıyla her sayıya bir karşılık gelir. Örneğin 0 sayısal değeri için A , 63 sayısal değeri için / sembolü gibi.

Bir metnin base64'e çevrilmesi işlemi ise ikilik tabanda ascii kodları ile metni kodlayıp ardından metni base64 için tamamlamak ve yukarıdaki sembollerle ifade etmektir. Bu işlemin nasıl yapıldığını adım adım inceleyelim.

Base64 ile kodlamak istediğimiz mesajımız "www.bilgisayarkavramlari.com" olsun. Bu mesajdaki her harfin karşılığı olan ASCII kodunu bulalım:

119 119 119 46 98 105 108 103 105 115 97 121 97 114 107 97 118 114 97 109 108 97 114
105 46 99 111 109

Yukarıdaki dizilimde her sembol için karşılığı olan ASCII kodu yazılmıştır. Buna göre örneğin mesajımızın ilk harfi olan w için 119 veya mesajın son harfi olan m için 109 sayısı tablodan bulunmuştur.

Yukarıdaki çevrilmiş ve onluk tabandaki sayıları ikilik tabana çevirirsek:

0111 0111 0111 0111 0111 0111 0010 1110 0110 0010 0110 1001 0110 1100 0110 0111
0110 1001 0111 0011 0110 0001 0111 1001 0110 0001 0111 0010 0110 1011 0110 0001
0111 0110 0111 0010 0110 0001 0110 1101 0110 1100 0110 0001 0111 0010 0110 1001
0010 1110 0110 0011 0110 1111 0110 1101

Mesajını buluruz. Yukarıdaki mesajda her 8 bit (ikil) ASCII tablosunda bir sayıya karşılık gelir. Örneğin mesajın ilk 8 biti (ikili) 0111 0111, onluk tabana çevrilirse 119 yapar ve bu değer tablodaki w sembolüne karşılık gelir. Yukarıda toplam 224 bit (ikil) bulunmaktadır. Çünkü mesaj uzunluğu 28 karakterdir ve her karakter için 8 bit kullanılmıştır.

Yukarıdaki mesajın base64 ile kodlanması sırasında mesajın eksik bulunan bitlerinin tamamlanması gerekir. Buradaki hesap basitçe mesajın boyutunun 24'ün katı olana kadar 8'er bitlik 0 (sıfırların) eklenmesidir.

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 yasası gereği suçtur.

Mesajda toplam 224 bit bulunuyor ve 224, 24'e tam olarak bölünemiyor. Bu durumda sayımız 24'e tam bölünebilmesi için 240'a tamamlanması gerekir. Sonuçta mesajımız aşağıdaki şekilde olacaktır:

0111 0111 0111 0111 0111 0111 0010 1110 0110 0010 0110 1001 0110 1100 0110 0111
0110 1001 0111 0011 0110 0001 0111 1001 0110 0001 0111 0010 0110 1011 0110 0001
0111 0110 0111 0010 0110 0001 0110 1101 0110 1100 0110 0001 0111 0010 0110 1001
0010 1110 0110 0011 0110 1111 0110 1101 0000 0000 0000 0000

Görüldüğü üzere sonuna 16 adet 0 eklenmiştir. Şimdi artık mesajımız base64 kodlamasında işlenmeye hazırdır. Mesajı bu sefer 6 bitlik parçalara bölüp onluk sisteme çevirebiliriz. Buradaki amaç bizim kodlamamızdaki 64 sembolden hangisine karşılık geldiğini bulmaktır.

Örneğin mesajın ilk 6 biti 011101 'dir ve onluk sistemde 29 yapar. Bu bizim kodlamamızda yukarıda verilen sembollerden 29. Sembol ile gösterileceğini ifade eder. Bu sembol A harfi 0 olarak kabul edilip sayılırsa d harfi olarak bulunur. Benzer şekilde bir sonraki 6lık grup alınırsa 110111 sayısını onluk tabanda 55 sayısı bulunur ve bizim kodlamamızda 55 sayısı ile 3 sembolü gösterilmektedir.

Yukarıdaki bu işlemi bütün mesaj için uygularsak sonuçta www.bilgisayarkavramlari.com mesajı için aşağıdaki sonuç bulunur:

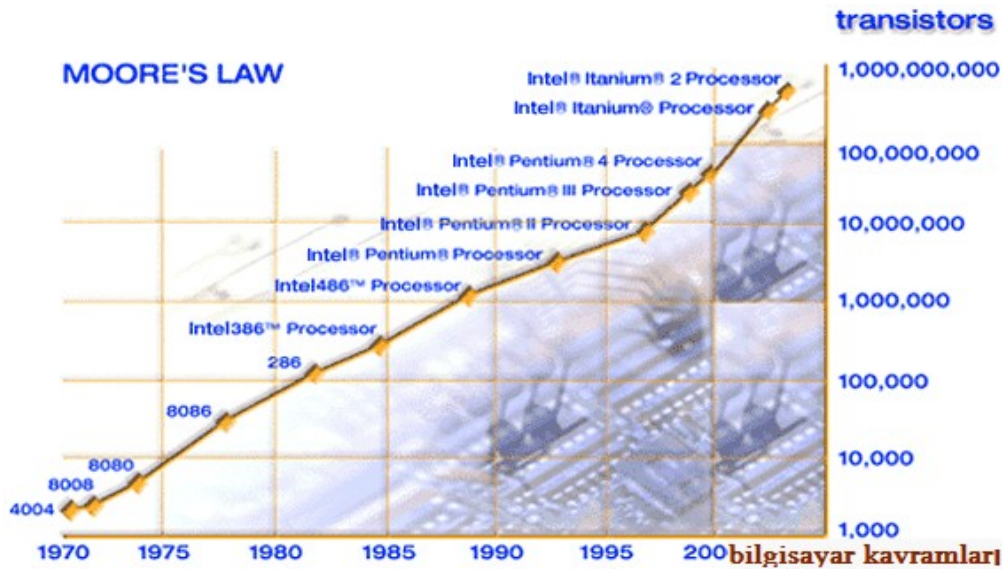
d3d3LmJpbGdpc2F5YXJrYXZyYW1sYXJpLmNvbQ==

Yukarıdaki mesaj tam bir base64 çevrimidir. Burada mesajın sonunda bulunan == sembollerinin bizim kodlamamızda yer almadığına dikkat ediniz. Bu semboller mesajın ayrılmasını sağlayan son eklerdir.

Base64 örneğin MIME protokolünün içerisinde gönderilen mesajlara uygulanan bir kodlamadır.

SORU 36: Moore Yasası (Moore's Law)

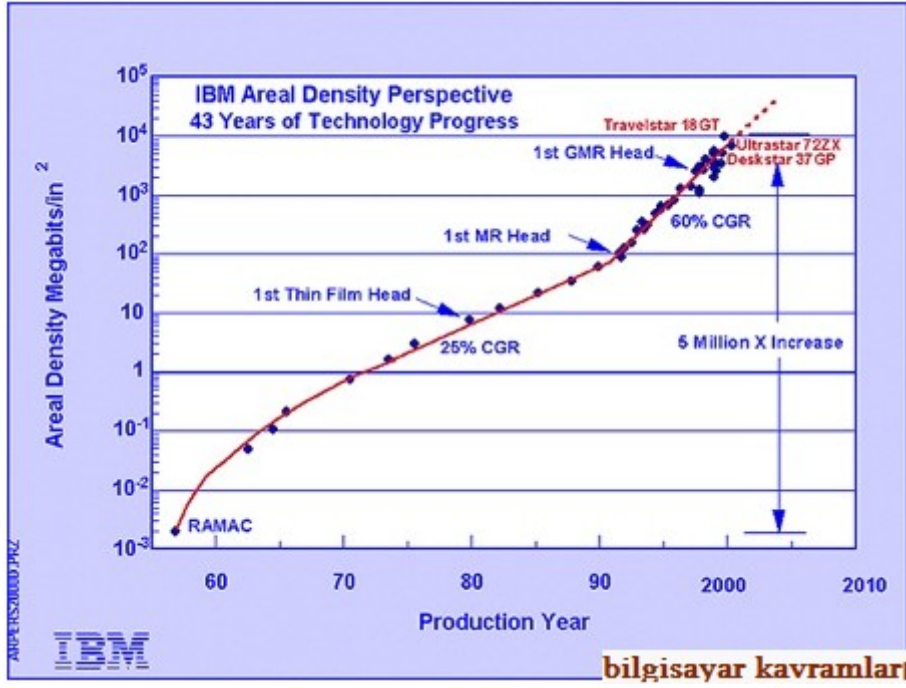
Moore yasası, bir yasa olmaktan çok, bilgisayar donanımında kullanılan transistör sayısı ile ilgili bir istatistiki gözlemdir. Bu yasaya göre her iki yılda aynı hacim içerisine sığan transistör sayısı ikiye katlanmaktadır. Aşağıda emsnow sitesinden alınmış ve yıllarla göre işlemci teknolojisindeki atlamaları ve transistör sayılarını gösteren bir şekil bulunmaktadır:



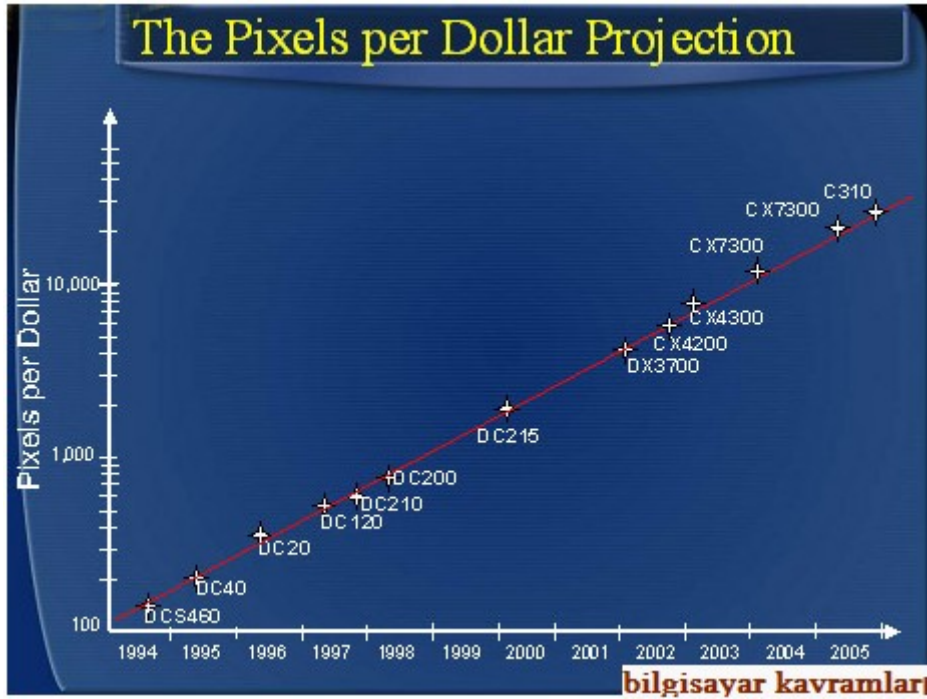
Yukarıdaki bu şekilde takip edilebileceği üzere gelişim doğrusal bir yapıya sahiptir. Yasayı ortaya atan Gordon E. Moore 1970 yılında henüz 2200 adet transistör içeren işlemciler için bu yasayı söylediğinden beri gelişme doğrusunda büyük bir sapma olmamıştır. İstatistiksel olarak bu doğrunun devam etmesi beklenmektedir.

Moore yasasının farklı uygulamaları bilgisayar hafızaları (RAM) veya bir dijital kameranın kaç piksellik resim çekebildiği gibi farklı alanlara da uygulanmıştır.

Örneğin internet bağlantı hızı yoğunluğu için IBM tarafından hazırlanan aşağıdaki grafikte benzer bir doğrusallık göze çarpar:



Benzer bir grafik ise dijital fotoğraf makinelerindeki imgecik (pixel) yoğunluğunun fiyata oranı için aşağıdaki şekilde hazırlanmıştır:

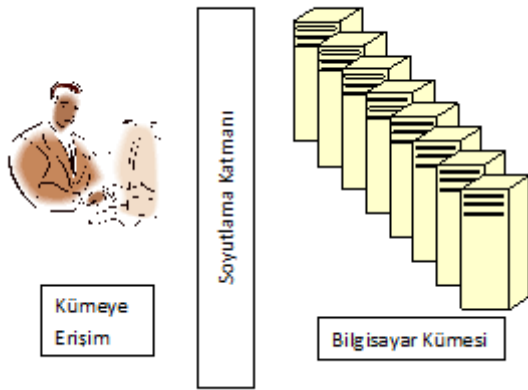


Yukarıdaki bu grafiği hazırlayan kişinin adına bu artışa da hendy's law (Hendy yasası) ismi verilmektedir. Bu grafikte, bilgisayar fiyatları hakkında da bilgi edinilebilir. Örneğin piksel başına fiyat değeri yıllara göre azalmakta, diğer bir deyişle bir dolar başına piksel sayısı giderek artmaktadır. Bu durum bilgisayar donanım fiyatlarındaki, ücret / performans dengesini göstermektedir.

SORU 37: Cluster Computing (Bilgisayar Kümeleri)

Bilgisayar bilimlerinde, daha fazla işlem gücü elde etmek amacıyla birden fazla bilgisayarın tek bir bilgisayar gibi çalışmasına verilen isimdir. Genelde birden fazla bilgisayar birbirine oldukça hızlı bir ağ bağlantısı ile bağlanır ve bilgisayarların üzerinde çalıştırılan özel yazılımlar ile istenen işin paylaşılması hedeflenir. Literatürde kümeleme veya İngilizce olarak clustering terimleri de kullanılmaktadır.

Bilgisayarların üzerinde çalışan işletim sisteminin tek bir işletim sistemi gibi davranması durumuna dağıtık işletim sistemi (distributed operating system) ismi verilir. Burada kullanıcı veya çalıştırılması istenen işler kümeye (cluster) tek bir noktadan erişir ve işin arka planda nasıl yapıldığı ile arasında bir soyutlama katmanı (abstraction layer) bulunur.



Diğer bir deyişle bilgisayar kümelerinde hedeflenen amaç, kullanıcıların yada çalışan işlemlerin (process) kümenin iç yapısından bağımsız olması ve kümenin her durumda istenen işi en verimli şekilde çalıştırmasıdır.

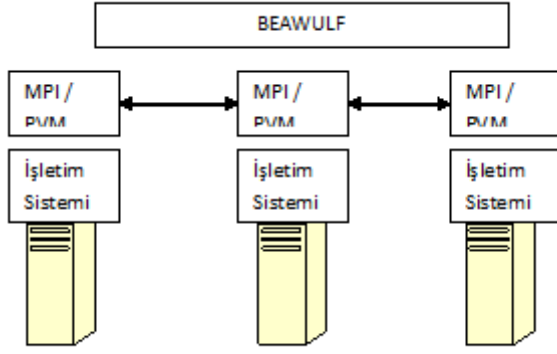
Ancak bu durum ne yazık ki günümüzde tam olarak gerçekleştirilememiş bir hayaldir. Bunun en önemli sebebi yük dağılımının (load balancing) her durum için en verimli şekilde yapılmasını sağlayacak bir otomat bulunamamasıdır.

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 yasası gereği suçtur.

Örneğin bir matris çarpımı işlemi yaparken kullanılacak dağıtım ile bir fraktal üretimi yada resim işleme sırasında yapılacak dağıtım farklı olabilir.

Tam bu noktada işlem kümeleri (Computing clusters) terimi devreye girer. Bu tip özel kümelerde amaç sadece özel bir iş yada iş tipi için verimli hale getirilmiş kümeler ve yazılımlar üretmektir.

Örneğin sadece hava durumu tahmini veya sadece sonlu eleman analizi (finite element analysis) için geliştirilmiş bir kümede özel bir müdahale gerekmeden işlem yapılabilir. Genellikle PVM (parallel virtual machine) veya MPI (message passing interface) benzeri alt yapılar kullanılarak sağlanan bu paralelleştirme işlemlerine işletim sistemlerinin üzerinde çalışan beowulf gibi katmanlar örnek gösterilebilir.



Yukarıdaki şekilde üç bilgisayar üzerinde çalışan bu yapı temsil edilmiştir. Bilgisayarların üzerinde çalışan işletim sistemleri. Bu işletim sistemleri üzerinde çalışan paralel işlem katmanı ve en üstte çalışan işlem kümesi yazılımı.

Küme bilgisayarların farklı bir uygulama şeklide ızgara hesaplamalarıdır (grid computing). Bu tip dağıtımli işlemelerde (distributed computing) amaç yukarıda anlatılan işlem kümelerinden (computing clusters) farklı olarak daha genel amaçlara yönelik işlem gücü elde etmektir. Ayrıca burada tam bir bölünmüşlükten bahsedebiliriz. Bazı durumlarda ağ üzerinde iletişimin oldukça az olması ve işlem yapan bilgisayarların birbiri ile neredeyse hiç konuşmaması bile söz konusudur. Örneğin kullanan SETI@Home projesi buna örnek gösterilebilir. Bu projede insanlar bilgisayarlarını kullanmadıklarında devreye giren basit bir program bütün dünyadan insanların bilgisayarında işlem yapıp sonuçları merkezi sunucuya bildiriyor. Bu sayede uzun süre boş duran bilgisayarların işlem gücü zayi olmamış oluyor ve astronomi konusunda çeşitli hesaplamalarda kullanılıyor.

SORU 38: ASCII Tablosu (table)

Bilgisayar bilimlerinde kullanılan ve her sembolü sayısal olarak ifade etmeye yarayan tablolardan birisidir. Aslında günümüzde en çok kullanılanıdır. ASCII harfleri American Standard Code for Information Interchange kelimelerinin baş harflerinden oluşmaktadır. Kelime olarak 1973 yılında, bu alandaki ihtiyacı doldurmak amacıyla ANSI tarafından (American National Standards Institute , Amerikan ulusal standartlar enstitüsü) tarafından ilk kez kullanılmıştır.

Basitçe bilgisayarın işlediği sinyalleri (ki bu sinyalleri 1 ve 0 olarak göstermek mümkündür), insanların anlayabileceği sembollere çevirmek için kullanılır. Bu tablonun ilk 7 [bit \(ikil\)](#) oluşan kısmı aşağıda verilmiştir.

0	NUL	32	espace	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(72	H	104	h
9	HT	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	UT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	SLE	48	0	80	P	112	p
17	CS1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SIB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~

Tablo aslında 8 [bit \(ikil\)](#) için kullanılabilir ve bu durumda 256 karakter içerir ancak son bitin (ikil) kullanımı sonucunda çıkan tablo dillere göre değişmektedir. Bu son bitin eklenmiş haline uzatılmış ASCII (extended ascii) ismi de verilmektedir.

Dec Hex Char Dec Hex Char Dec Hex Char Dec Hex Char

128	80	160	A0	192	C0	224	E0
129	81	161	A1	193	C1	225	E1
130	82	162	A2	194	C2	226	E2
131	83	163	A3	195	C3	227	E3
132	84	164	A4	196	C4	228	E4
133	85	165	A5	197	C5	229	E5
134	86	166	A6	198	C6	230	E6
135	87	167	A7	199	C7	231	E7
136	88	168	A8	200	C8	232	E8
137	89	169	A9	201	C9	233	E9

138 8A	170 AA	202 CA	234 EA
139 8B	171 AB	203 CB	235 EB
140 8C	172 AC	204 CC	236 EC
141 8D	173 AD	205 CD	237 ED
142 8E	174 AE	206 CE	238 EE
143 8F	175 AF	207 CF	239 EF
144 90	176 B0	208 D0	240 F0
145 91	177 B1	209 D1	241 F1
146 92	178 B2	210 D2	242 F2
147 93	179 B3	211 D3	243 F3
148 94	180 B4	212 D4	244 F4
149 95	181 B5	213 D5	245 F5
150 96	182 B6	214 D6	246 F6
151 97	183 B7	215 D7	247 F7
152 98	184 B8	216 D8	248 F8
153 99	185 B9	217 D9	249 F9
154 9A	186 BA	218 DA	250 FA
155 9B	187 BB	219 DB	251 FB
156 9C	188 BC	220 DC	252 FC
157 9D	189 BD	221 DD	253 FD
158 9E	190 BE	222 DE	254 FE
159 9F	191 BF	223 DF	255 FF

Yukarıdaki taloda 128 ile 255 arasındaki uzatılmış ascii tablosunun karakter karşılıkları görülmektedir. Dikkat edileceği üzere Türkçe karakterler de bu tabloda bulunmaktadır.

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

Yukarıdaki tabloda bulunan sembollerin ikilik tabana çevrilmesi mümkündür. Yukarıdaki sayılar 10'luk tabanda olduğu için, örneğin “ş” harfini iklik tabana çevirmek istediğimizde 159 sayısını iklik tabana çevirmemiz yeterlidir.

$$(159)_{10} = (100011111)_2$$

şeklinde yazılabilir.

Örneğin aşağıdaki C kodunu ele alalım:

```
#include <stdio.h>
int main() {
    printf("%c", 159);
}
```

Yukarıdaki kod çalıştırıldığında ascii tablosunda 159 değerine sahip olan ş harfi ekrana basılacaktır.

Benzer şekilde aşağıdaki C kodu işlemi tersinden yapacaktır:

```
#include <stdio.h>
int main() {
    printf("%d", 'a');
}
```

Yukarıdaki kodda a karakterinin sayısal değeri (%d ile basılan değeri) kullanılmıştır. Bu durumda ekranda sayısal olarak 97 görülecektir.

SORU 39: Çerezler (Cookies)

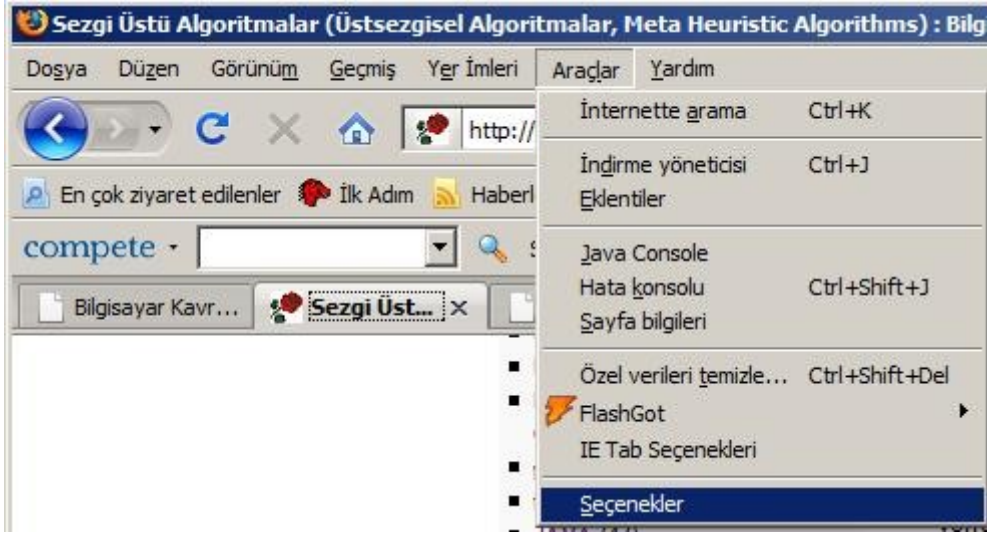
İnternet üzerinde, özellikle de web sayfaları üzerinde gezinirken kullanılan ufak kayıt dosyalarına verilen isimdir. Basitçe bir web sitesi internet üzerinden yayın yaparken bazan bağlanan kullanıcılar hakkında bilgi tutma ihtiyacı duyar. Genelde bu bilgiyi tutmanın iki yolu vardır. Birincisi sunucu üzerindeki bir veri tabanı veya farklı bir veri saklama yapısı içinde tutulması. Diğeri ise istemci (client) üzerinde saklamak.

Bu yazının konusu istemci (client) üzerinde veri saklama teknolojilerinden çerez ismi verilen (cookie) dosyaları anlatmaktır.

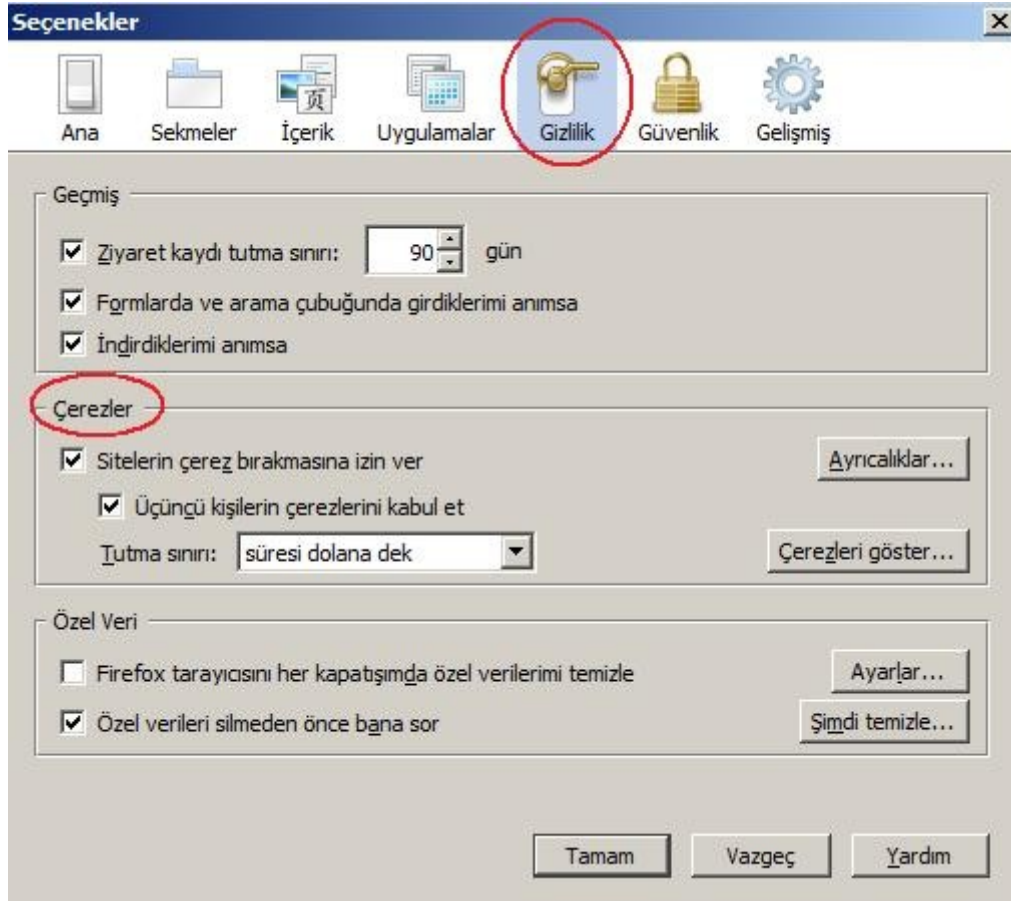
Literatürde, çerezler HTTP protokolü üzerinden taşındıkları için HTTP çerezleri (http cookie) olarak da geçmektedirler.

Çerezlerin internet gezgininde ayarlanması

Çerezlerin detayına ve programlamasına geçmeden önce çerezleri kullanıcıların kendi bilgisayarlarında nasıl ayarlayabileceklerini açıklayalım (ekran görüntüleri ve menü yerleri Firefox 3 sürümünden alınmıştır) . Örneğin firefox internet gezginindeki çerez ayarları aşağıdaki şekilde yapılabilir :



Öncelikle ayarların yapılacağı ekrana Araçlar > Seçenekler tıklanarak girilebilir:

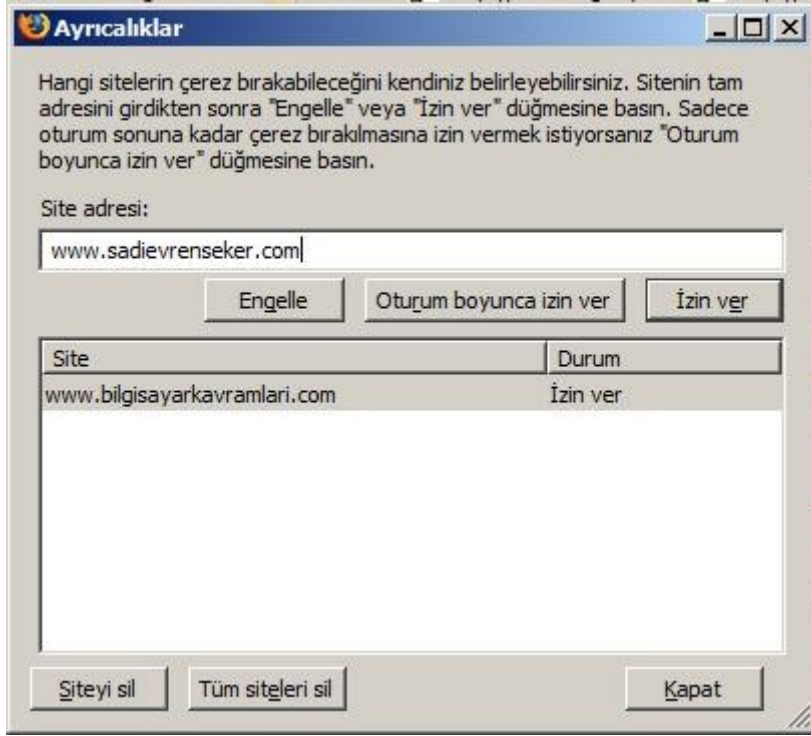


Yukarıda görüldüğü şekilde açılan seçenekler diyaloğunda Gizlilik sekmesi altında Çerezler bölümü bulunmaktadır. Bu bölümde istenirse sitelerin çerez bırakmasına izin verilebilir veya bu izin kaldırılabilir.

Temel olarak kullanıcıların böyle bir izni verme veya kaldırma hakkı bulunmaktadır çünkü sonuçta site tarafından kullanıcının bilgisayarına bir dosya kaydedilecektir. Yazının ilerleyen

kısımlarında da anlatılacağı üzere bu izin kötü amaçla kullanılabilmekte ve kullanıcılar için tehdit oluşturabilmektedir. Dolayısıyla kullanıcı dilerse bu seçeneği kapatabilir.

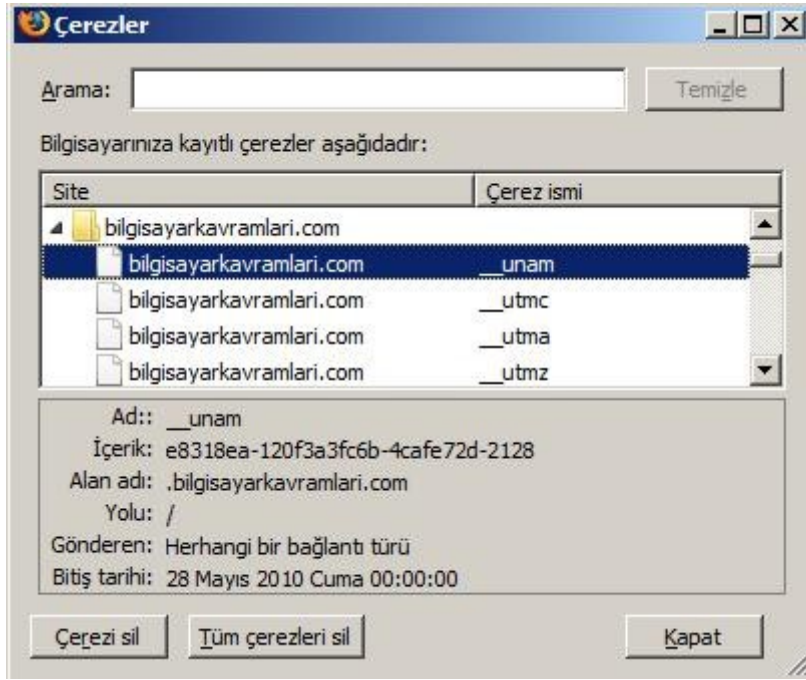
Bu temel özellik bütün internet gezginlerinde ortak olmakla beraber yukarıdaki resimde de görüldüğü üzere Firefox internet gezgininde ilave bazı özellikler bulunmaktadır. Bunlardan birincisi ayrıcalık tanıma özelliğidir. Kullanıcılar isterlerse site bazlı olarak özel ayar yapabilirler ve kuralı sadece belirli sitelere uygulayabilirler.



Örneğin yukarıdaki resimde ayrıcalık diyalogu açıldığında çıkan ekran bulunmaktadır. Burada www.bilgisayarkavramlari.com sitesine izin verilmiştir. Benzer şekilde www.sadievrenseker.com sitesi eklenmek üzere yazılmıştır. Dilenirse Engelle düğmesi ile engellenebilir veya izin ver düğmesi ile izin verilebilir. Ortada bulunan Oturum boyunca izin ver düğmesi ise siteye bir girişlik izin vermek ve siteden çıkıldıktan sonra çerezin temizlenmesi anlamına gelir.

Eklenen siteler istenirse alt tarafta bulunan siteyi sil veya tüm siteleri sil düğmeleri ile silinebilir.

Ayarlar ekranından çerezleri göster düğmesi ile de aşağıdaki ekrana geçilebilir:



Bu ekranda site bazlı olarak çerezlere ve bilgilerine erişmek mümkündür. Bilgisayarımıza bir sitenin bıraktığı çerezi ve detaylarını buradan görebiliriz. Elbette çoğu site çerez içeriği olarak şifreli bilgi tutmaktadır. Bunun sebebini güvenlik kısmında anlayacağız.

Çerezlerin yukarıdaki ekranda da görüldüğü üzere bitiş tarihleri bulunmaktadır. Yani bir çerez istenirse belirli bir süreliğine yollanabilir. Örneğin sitemize giren kişinin alışveriş sırasında sepetine eklediği eşyaların sadece 1 saat boyunca geçerli olmasını bundan sonra tekrar sitemize girerse sepetinin boşalmasını istiyor olalım. Bu durumda çerezi oluştururken bitiş tarihi olarak 1 saat ileri tarihi eklememiz yeterli olacaktır.

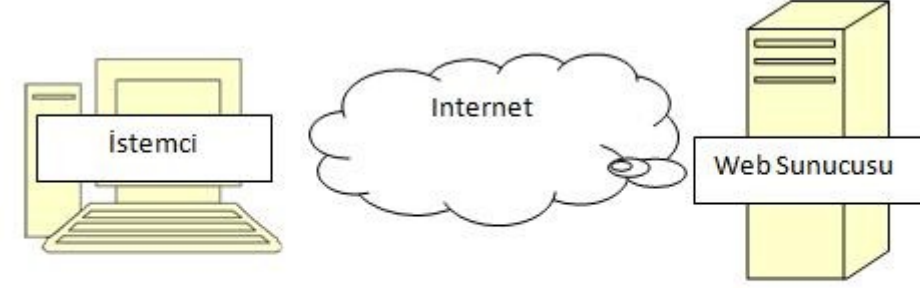
Internet gezginleri süresi dolan çerezleri saklamaz (veya saklamayabilir) yani bu tip süresi dolmuş çerezlerin silinip silinmemesi internet gezgininin insiyatifindedir ancak yine de süres dolan çerezlerin kullanılamayacağını bilmemiz yeterlidir.

Çerezlerin çalışması

Bir çerez, normal bir HTTP paketi ile kullanıcı tarafından talep edilir ve sunucu tarafından oluşturularak kullanıcıya yollanır.

HTTP protokolü üzerinden yapılan veri transferi request / response (talep / arz (istek cevap)) şeklinde olmaktadır. Yani istemci (müşteri , client) bilgisayar sunucu (server) bilgisayarından bir bilgiyi talep eder (request) ve sunucu bilgisayar bu bilgiye cevap olarak bir sonuç arz eder.

Bu iletişim şeklinde arz edilen (sunucudan dönen) bilgi içerisinde bir çerez bilgisi bulunabilir. Bu bilgiyi HTTP protokolü desteklemektedir:



```
GET /index.html HTTP/1.1
Host: www.bilgisayarkavramlari.com
```

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: isim=deger
(Sayfanın istenen içeriği)
```

```
GET /spec.html HTTP/1.1
Host: www.bilgisayarkavramlari.com
Cookie: isim=deger
Accept: */*
```

Örneğin yukarıdaki şekilde adım adım bir HTTP protokolü üzerinden iletişim temsil edilmiştir. Önce istemci tarafı internet üzerinden bir HTTP paketi ile www.bilgisayarkavramlari.com sitesinde bulunan index.html dosyasını talep etmektedir.

Ardından sunucu bu talebe yine bir HTTP paketi ile cevap vermektedir. Bu paketin içeriğine dikkat edilecek olursa cevap HTTP 1.1 sürümü ile yapıldığını göstermektedir. Ayrıca 200 OK mesajı dönmüştür. Yani istenen sayfanın bulunduğu ve başarılı bir talep olduğu anlamında bir HTTP kodu ile talep cevaplanmıştır.

Burada HTTP paketinde bizim için önemli olan Set-Cookie bökümüdür. Bu bölümde bir çerez'in istemciye yollandığı ifade edilmektedir ve isim=değer ibaresiyle herhangi bir bilgi istemciye çerez olarak yüklenmi olur.

Buradaki isim=değer bizim belirlediğimiz bir ismin değeridir. Örneğin : “Giriştarihi = 28102009 ” şeklinde bir bilgi olabilir.

Bu şekilde istemci tarafı bilgiyi talep ettikten ve içeriğinde çerez bulunan bir bilgi geldikten sonra bir önceki bölümde açıkladığımız üzere bu bilgi bilgisayarımızda bir dosya olarak saklanır.

Bundan sonraki talepler (farklı sayfa veya aynı sayfanın tekrar talep edilmesi gibi), bu çerez bilgisi de HTTP paketine ilave edilir.

PHP dilinde çerez programlama

Sunucu tarafı betik dili (server side scripting language) php ile çerez programlamak mümkündür. Bir php sayfasında çerez belirlemek için aşağıdaki satırlar kullanılır:

```
<?php
setcookie("kullanici", "Şadi Evren ŞEKER", time()+3600);
?>
```

Yukarıdaki örnek kodda php sayfasında setcookie fonksiyonu marifetiyle “kullanici” adında bir değişken tanımlanmış ve içeriğine Şadi Evren ŞEKER bilgisi eklenmiştir. Fonksiyonun son parametresi ise çerezin yaşam süresidir. Burada yine php içerisinde tanımlı olan time() fonksiyonu kullanılarak mevcut zaman bilgisi sistemden (sunucunun saatinden) okunmuş ve bu süreye 3600 saniye (tam olarak 60 dakika veya 1 saat) ilave edilerek çerezin geçerli olacağı son an belirlenmiştir.

Yukarıdaki kodda belki dikkat çekmeyebilir ancak setcookie fonksiyonu her zaman için sayfanın en başında bulunmalıdır. Bunun sebebi daha önce de açıkladığımız HTTP paketinin başlık kısmında yer alan bir bilgi olmasıdır. Şayet sayfanın içeriğinde bir yerlerde bu bilgi gönderilirse HTTP protokolünün bu bilgiyi ayırma şansı kalmaz.

Sitemizde yukarıdaki sayfa ile bir çerez üretilerek bu çerezin içerisinde kullanıcı=Şadi Evren ŞEKER bilgisi konulmuştur. Bu bilgiye erişilmek istendiğinde (yine bu sayfadan veya aynı sitedeki herhangi başka bir sayfadan) aşağıdaki şekilde ulaşılabilir:

```
<?php
echo $_COOKIE["kullanici"];
print_r($_COOKIE);
?>
```

Yukarıdaki kodun ilk satırı ile \$_COOKIE sistem değişkeni (ki bu değişken bir dizidir (array)) içerisinde bulunan “kullanici” bilgisine erişilmiştir. Bu dizinin tamamının içeriğini görmek için ikinci satırda bulunan ve dizi içeriğini basmaya yarayan print_r fonksiyonundan yararlanılabilir.

JSP dilinde çerez programlama

JAVA’nın web tabanlı arayüzü kabul edebileceğimiz java server pages (jsp) ile de yukarıdaki php koduna benzer çerez tanımları yapmak mümkündür.

Örneğin aşağıdaki sayfa kodunu inceleyerek JSP üzerinden nasıl çerez kullanıldığını anlamaya çalışalım :

```
<%@ page language="java" import="java.util.*"%>
<%
User username = new User();
Date now = new Date();
String timestamp = now.toString();
Cookie cookie = new Cookie ("username",username);
cookie.setMaxAge(365 * 24 * 60 * 60);
response.addCookie(cookie);
%>
<html>
<body>
<p><a href="iki.jsp">ikinci sayfa icin buraya basiniz</a><p>
</body>
```

```
</html>
```

Yukarıdaki kodda basit bir web sayfası html dilinde kodlanmıştır. JSP dilinden hatırlanacağı üzere `<% %>` blokları arasındaki kod JSP'ye aittir. Bu alan kolay okunsun diye yukarıda kırmızı renkle belirlenmiştir.

Sayfamızda JSP alanı içerisinde amacımız Cookie sınıfından (class) bir nesne (object) üretmektir. Bu nesnenin özelliği bir dizgi (string) ve bir nesne (object) parametre almasıdır. Yukarıdaki kodda bulunan :

```
Cookie cookie = new Cookie ("username",username);
```

Satırı aslında JSP için çerez kodlamanın yapıldığı satırdır. Bu satıra dikkat ederseniz Cookie yapıcısının (constructor) içerisine birinci parametre olarak bir dizgi (string) verilmiştir. Bu yazı yani "username" ilerde çerezimize erişmek için kullanacağımız bir etiket olarak düşünülebilir. Bu etiketle erişilecek olan bilgi ise Cookie yapıcısının (constructor) ikinci parametresi olan ve daha önceden bir nesne olarak tanımlanmış olan username değişkenidir.

Yukarıdaki örnekte username nesnesi, User ismindeki bir sınıftan türetilmiştir. Siz uygulamanızda cookie olarak saklamak istediğiniz bir nesneyi buraya yerleştirebilirsiniz.

Yukarıdaki şekilde HTTP paketine yerleştirilen bir çereze yine JSP kodunu kullanarak erişmek için aşağıdaki kodlama işinize yarayabilir:

```
<%@ page language="java" %>
<%
    String cookieName = "username";
    Cookie cookies [] = request.getCookies ();
    Cookie myCookie = null;
    if (cookies != null)
    {
        for (int i = 0; i < cookies.length; i++)
        {
            if (cookies [i].getName().equals
(cookieName))
            {
                myCookie = cookies[i];
                break;
            }
        }
    }
%>

<html>
<body>
<%
    if (myCookie == null) {
%>
        <%=cookieName%> isminde bir çerez bulunamadı.

    } else {
%>
        <p>Merhaba: <%=myCookie.getValue() %>.

    }
%>
</body>
```

</html>

Yukarıdaki kodda request.getCookies() fonksiyonu ile, sitemizden erişilebilen bütün çerezler alınmıştır. Ardından bir döngü ile bu çerezler arasında ismi “username” olan çerez aranmıştır. Bulunan bu çerez myCookie isimindeki çerezin içerisine konularak HTML sayfasının içine myCookie.getValue() ile ekrana yazılmıştır.

ASP dilinde çerezlerin kullanımı

ASP Microsoft tarafından geliştirilen bir sunucu tarafı betik dilidir (server side scripting language). Bu anlamda JSP ve PHP’ye benzemektedir. Aşağıdaki örnek kod ile bir ASP sayfası üzerinden nasıl çerez üretildiğini anlayabiliriz:

```
<%  
Response.Cookies("kullanici")="Şadi Evren ŞEKER"  
Response.Cookies("kullanici").Expires=#May 10,2010#  
%>
```

Yukarıdaki kodda daha önceki dillerde de gördüğümüz üzere çerezin ismi ve değeri atanmıştır. Çerezimize isim olarak kullanıcı ismi verilmiş ve değer olarak ilk satırda içeriğine “Şadi Evren ŞEKER” değeri konulmuştur. Çerezin yaşam süresi ise 10 Mayıs 2010 olarak ikinci satırda atanmıştır.

Aşağıdaki kod ile, yukarıda atanan içeriğe farklı bir sayfadan erişebiliriz:

```
<%  
abc=Request.Cookies("kullanici")  
response.write("Çerez bilgisi=" & abc)  
%>
```

Yukarıdaki kodda, abc isimindeki değişkene öncelikle HTTP paketinden kullanıcı isimli değişken içeriği okunmuştur. Kodun ikinci satırında bu bilgi response.write ile istemciye geri yollanmış ve ekranda görüntülenmiştir.

Çerezler ve güvenlik

Yukarıda da açıklandığı üzere, çerezlerin siteler tarafından serbestçe erişilebilir olması bazı güvenlik sorunlarını da beraberinde getirmektedir. Aşağıda bu sorunlardan bazıları açıklanmıştır:

İz takibi : Bu güvenlik zaafiyeti birden fazla siteye tek bir elde yerleştirilen çerezlerde olur. Örneğin reklam yayını yapan bir şirket, reklamını yayınladığı yerlerde aynı zamanda çerezini de yayınlatabilir. Bu tip çerezlere, üçüncü parti çerezler (third party cookies) ismi verilir. Bunun sebebi sitenin esas yayıncısı ve siteyi o anda ziyaret etmekte olan istemci (client) dışında üçüncü bir kişinin çerezi olmasıdır.

İşte bu üçüncü parti çerezler siteyi ziyaret eden kişinin bilgisayarına kaydedilir. Şayet reklam veren şirket isterse ziyaretçinin girdiği bütün sitelerin izini sürebilir. Yani reklamının yayımlandığı hangi sitelerin kullanıcı tarafından ziyaret edildiğini takip etmesi mümkündür.

Çerez kaçırlması (cookie hijacking): Bilindiği üzere çerezlerin içerisinde site ve kullanıcı ile ilgili çeşitli bilgiler tutulmaktadır. Saldırgan bir taraf sunucu ve istemci arasında gidip gelen

bu çerezleri takip ederek veya istemcinin bilgisayarında saklanan çerez dosyalarına erişerek çeşitli bilgileri ele geçirebilir. Bu noktada sunucu üzerinden çerez programlayan tarafın oldukça hassas davranması ve kişisel bilgileri, şifre kullanıcı detayı gibi bilgileri çerez üzerinde tutmaması gerekir. Elbette internet gibi bir ortamda bu hassasiyet herkesten beklenemez bu da çerezlerin güvenlik açısından oluşturduğu bir problemdir. Diğer bir çözüm ise çerezlerin içindeki bilgilerin şifreli (encrypted) tutulmasıdır. Elbette bütün şifreler bir gün kırılabilir ancak bu vakit alacaktır ve daha organize bir saldırı gerektirecektir.

Çerez zehirlenmesi (cookie poisoning): Çerez zehirlenmesindeki amaç, istemci taraftan sunucuya giden bilgilerin amaçlı olarak değiştirilmesidir. Örneğin bir alışveriş sitesinde, müşterilerin sepet bilgileri çerezde tutuluyor olsun. Ve büyük bir hata olarak müşterilerin sepetlerindeki eşyaların toplam fiyatı ve dolayısıyla müşterinin ödeyeceği fiyatın da çerezde tutulduğunu düşünelim. Kötü niyetli birisi bu değeri çerez üzerine elle müdahale ederek değiştirebilir ve aslında ödemesi gereken değerden çok daha ucuza sepetindeki eşyaları satın alabilir. Bu tip saldırılara çerez zehirlenmesi ismi verilir.

Oturum saldırısı (session hijacking): Çerezlerin sıkça kullanıldığı yerlerden birisi de oturum bilgilerinin saklanmasıdır. Örneğin kullanıcının en son eriştiği sayfa, kullanıcının site üzerindeki ayarları veya kullanıcıya ait sitede tutulan bilgiler çerezlerde tutularak kullanıcının sonraki bağlantılarında sitede kaldığı yerden devam etmesi amaçlanır. Bu durum bir güvenlik zaaflığı doğurur. Örneğin saldırgan bir kişi bu bilgileri kullanarak aslında hiç olmayan bir kullanıcı ile sitede erişme izni olmayan sayfalara sanki en son bu sayfada kalmış gibi erişebilir. Veya sahip olmadığı yetkiye çerez üzerinde değişiklikler yaparak erişmeye çalışabilir. Bu saldırının tek mantıklı çözümü ise daha dikkatli web siteleri programlamak ve programcıların bu tip saldırılara karşı dikkatli olmasıdır.

SORU 40: Rastgele Sayılar (Random Numbers)

Bilgisayar bilimlerinde çeşitli amaçlarla rastgele sayılara ihtiyaç duyulur. Örneğin şifreleme algoritmalarında önemli bir role sahip olan rastgele sayılar şifreleme işleminin gizliliği ve güvenilirlik açısından önemlidir. Benzer şekilde bir oyun programlanırken veya bir simülasyon sırasında rastgele cereyan eden olaylar modellenirken rastgele sayılara (random numbers) ihtiyaç duyulur ve bu sayıların gerçekten rastgele olması beklenir.

Örneğin kaydırma şifrelemesi (shift cipher) kullanımı sırasında rast gele bir anahtar üretilmesi istendiğini düşünelim. Bilgisayarın her zaman için aynı rast gele sayıyı üretmesi istenmeyen bir durumdur çünkü bu durumda sisteme bir kere saldırarak anahtarı ele geçiren saldırganın bundan sonraki saldırılarda anahtarı bulmak için vakit kaybetmesine gerek kalmaz. Benzer şekilde farklı sayılar üreten ama sayıların tahmin edilmesi mümkün olduğu durumlarda da sistemin karmaşıklığı azaltılarak sistemin getirdiği karmaşıklık sonucu beklenenden oldukça kısa sürelerde sisteme saldırı gerçekleştirilebilir.

Bu anlamda ne yazık ki bilgisayarların rastgele sayı üretiminde oldukça başarısız olduklarını söylemek gerekir. Bilgisayarlar tasarımları ve yapıları itibarıyla rastgeleliğe yer bırakmayan belirli ve her adımında olacak şeylerin önceden tasarlandığı makinelerdir. Bu anlamda bilgisayarlarda rastgele bir bilgi üretmek oldukça zordur.

Bilgisayarlar kullanılarak rast gele sayı üretimi için en kolay yollardan birisi müsvette rastgele sayı üretimi yöntemleridir (pseudo random number generator). Bu sayı üretim yöntemlerindeki amaç aslında belirli bir matematiksel fonksiyona dayanarak bir dizi sayı

retmektir. Ancak bu matematiksel fonksiyon gizlenerek retilen sayının tahmin edilmesi engellenmeye alışılır.

rneęin en basit msvette rastgele sayı retelerinden birisi olan aŗaęıdaki rneęi inceleyelim:

$$F(i) = 3^i \bmod 7$$

Yukarıdaki fonksiyonu $i = \{1,2,3,4 \dots n\}$ gibi sayılar iin hesaplırsak her seferinde aynı sayı serisinin tekrarladığını grrz. Bu anlamda bu sayı serisine dairesel grup (cyclic group) ismi verilebilir. Ancak burada enteresan olan bir zellik aslında mod 7 olması itibariyle 7 sayıdan oluŗacak olan sayıların hangi sırayla geleceęinin tahminin g olmasıdır. rneęin bu sayı serimizi hesaplırsak :

$\{1,3,2,6,4,5,1\}$ sayılarını buluruz. ($3^0 \bmod 7 = 1$, $3^1 \bmod 7 = 3$, $3^2 \bmod 7 = 2$ řeklinde devam eden seri)

ŗimdi bu noktada durup dŗnelim. rneęin yukarıdaki F fonksiyonu ile son retilen sayı 6 olsun. Bir sonraki sayının ka olacaęını řayet F fonksiyonunu bilmiyorsak tahmin etmemiz gtr. Elbette buradaki rnekte mod 7 alındığı iin retilebilecek sayılar 7 tanedir ve bu ihtimal sayısı olduka azdır. Ancak daha yksek mod tabanlarında rneęin 20 haneli bir modler fonksiyonda bu deęerin tahmin edilmesi ok daha g olur.

Ancak yine de sayılarımız arasında bir baęlantı bulunmaktadır ve kt niyetli birisi (rneęin oyunda hile yapmak isteyen birisi veya řifreleme sistemine saldırmak isteyen birisi) bu baęlantıyı zerek (veya casusluk marifetiyle ęrenerek) kt niyetli olarak istifade edebilir.

Bu anlamda Gvenli rastgele'nin tanımını yapmamız gerekir.

Gvenli Rastgele (Secure Random)

Gvenli rastgele sayılar her defasında bir besleme (seed) deęeri ile retilen sayılardır. Buna gre sayının retilmesi belirli bir fonksiyona baęlı deęildir. Fonksiyonun en az bir yerinde dıŗarıdan gelen bir besleme deęeri bulunur ve bu deęere gre fonksiyonun rettięi sonu deęişir.

Genelde bilgisayarlar aısından dıŗarıdan besleme almak zordur. Ya ilave bir donanıma ya da ne yaptığının farkında olan eęitimli bir kiŗiye ihtiya duyulur. Bunların ikisi de maliyetli olduęu iin daha az maliyetli ve dolayısıyla daha az gvenli olan zaman faktr kullanılır.

Bu yazı řadi evren řeker tarafından yazılmış ve bilgisayararkavramlari.com sitesinde yayınlanmaktadır.

Yani bilgisayarın saati kullanılarak rastgele sayı reteci beslenir. Umulur ki saldırgan taraf bu besleme zamanını bilmedięi iin rastgele sayıyı tahmin etmesi de zor olur. Ancak bir nceki msvette rastgele sayı retimine gre daha gvenli olan bu yntem de ne yazık ki tam olarak gvenli deęildir nk saldırgan taraf her zaman iin besleme zamanını tahmin edebilir.

İlave donanım kullanılması durumunda rneęin kararsız bir elektronik devreden veya atom altı paracıklardan veya doęal bazı olaylardan (rneęin gneŗteki patlamalardan)

faaydalanılabılır. Elbette faydalanılan besleyici kaynağın (seeder) karmaşıklığına göre sistemin maliyeti artacaktır.

Diğér bir sık kullanılan besleme (seeder) yöntemi ise insan faktörüdür. Sonuçta bilgisayarların çoğunun etrafında insanlar bulunmaktadır ve insanların yaptığı eylemler tam olarak düzenli sayılmaz. Hatta bazı durumlarda tamamen düzensizdir ◀◀.

Örneğın bir insanın klavyede yazı yazması sırasında tuşlara basış hızı rastgele kabul edilebilir. En tecrübeli klavye kullanan insanın bile her harfe basış hızı ve her defasında basış hızı mili saniyeler cinsinden mutlaka farklılık gösterir. Bu özellik kullanılarak örneğın bir kullanıcının bir metni yazması sırasında tuşlara basış zamanlarından rastgele sayılar üretilebilir.

Diğér bir yöntem ise bilgisayarın kararsızlığıdır. Evet yapı olarak bilgisayarlar kararlídır ancak günümüz bilgisayarlarında çok işlemlı (multi processing) işletim sistemleri çalışmaktadır ve bir işlemin yapılma süresi o andaki bilgisayarın durumuna göre değişmektedir. Örneğın bilgisayar saatinin mili saniye hanesindeki değeri okumak istersek ve bu işlemi belirli bir sürede bir tekrar edersek bilgisayar bize hep aynı sonucu verecektir. Ancak bu tekrarları belirli bir kompleksliğın üzerindeki işten sonra yaparsak bu durumda bilgisayar elindeki kaynağa göre farklı zamanlarda işi bitireceğı için bize verdiği saat bilgisi değişebilecektir. Tabi saldırgan taraf bu durumda bilgisayarımızdaki o anda çalışan işlem miktarını tahmin etmek zorunda kalacaktır.

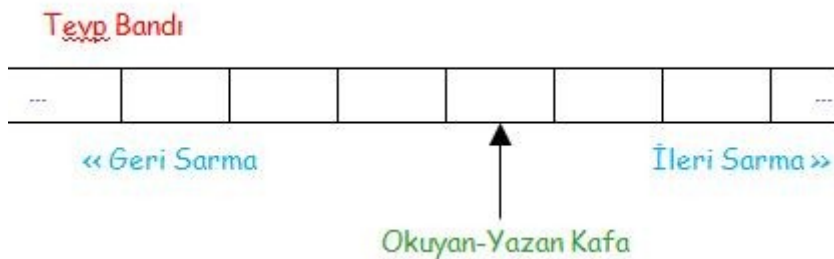
Ancak bu son örneklerde gösterilen ve literatürde kendinden beslemeli (self-seeding) olarak geçen yöntemler sonuçta bilgisayarda bulunan bazı özellikleri kullandığı için tahmin edilmesi ve saldırılması nispeten kolay yöntemlerdir.

SORU 41: Turing Makinesi (Turing Machine)

Bilgisayar bilimlerinin önemli bir kısmını oluşturan [otomatlar \(Automata\)](#) ve [Algoritma Analizi \(Algorithm analysis\)](#) çalıştırmalarının altındaki dil bilimin en temel taşlarından birisidir. 1936 yılında Alan Turing tarafından ortaya atılan makine tasarımı günümüzde pekçok teori ve standardın belirlenmesinde önemli rol oynar.

Turing Makinesinin Tanımı

Basitçe bir kafadan (head) ve bir de teyp bandından (tape) oluşan bir makinedir.



Makinede yapılabilecek işlemler

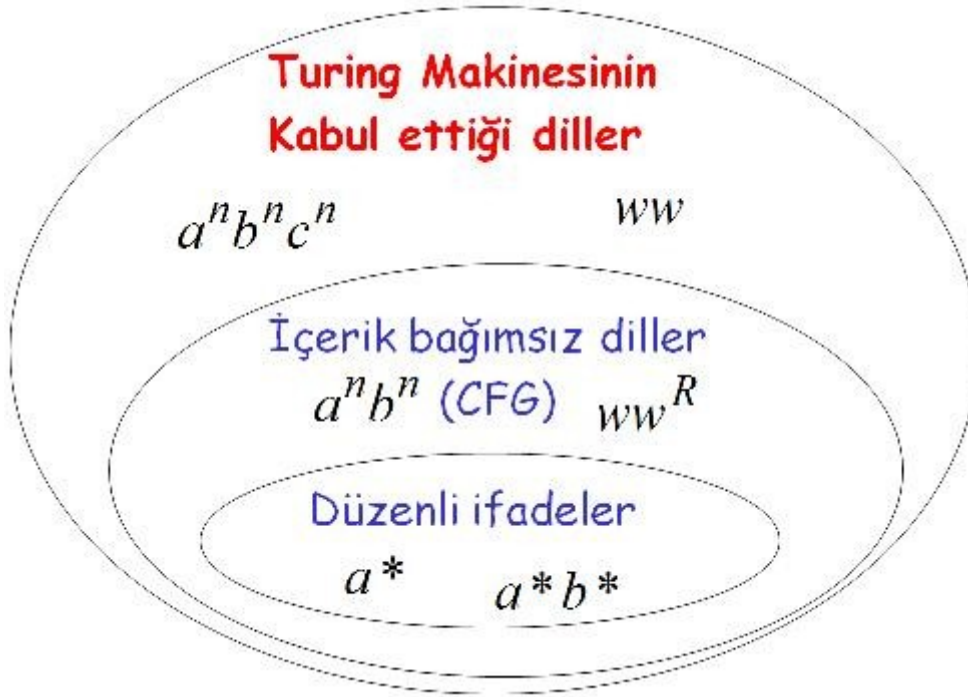
- Yazmak
- Okumak

- Bandı ileri sarmak
- Bandı geri sarmak

şeklinde sıralanabilir.

Chomsky hiyerarşisi ve Turing Makinesi

Bütün teori bu basit dört işlem üzerine kurulmuştur ve sadece yukarıdaki bu işlemleri kullanarak bir işin yapılıp yapılamayacağı veya bir dilin bu basit 4 işleme indirgenip indirgenemeyeceğine göre diller ve işlemler tasnif edilmiştir.



Bu sınıflandırma yukarıdaki venn şeması ile gösterilmiştir. Aynı zamanda [chomsky hiyerarşisi \(chomsky hierarchy\)](#) için 1. seviye (type-1) olan ve Turing makinesi ile kabul edilebilen diller bütün tip-2 ve tip-3 dilleri yani içerik bağımsız dilleri ve düzenli dilleri kapsamaktadır. Ayrıca ilave olarak içerik bağımsız dillerin işleyemediği (üretemediği veya parçalayamadığı (parse)) $a^n b^n c^n$ şeklindeki kelimeleri de işleyebilmektedir. Düzenli ifadelerin işleyememesi konusunda bilgi için [düzenli ifadelerde pompalama savı \(pumping lemma in regular expressions\)](#) ve [içerik bağımsız dillerin işlemeyemesi için de içerik bağımsız dillerde pompalama savı \(pumping lemma for CFG\)](#) başlıklı yazıları okuyabilirsiniz.

Turing Makinesinin Akademik Tanımı

Turing makineleri literatürde akademik olarak aşağıdaki şekilde tanımlanır:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \diamond, F)$$

Burada M ile gösterilen makinenin parçaları aşağıda listelenmiştir:

Q sembolü sonlu sayıdaki durumların kümesidir. Yani makinenin işleme sırasında aldığı durumardır.

Γ sembolü dilde bulunan bütün harfleri içeren alfabeyi gösterir. Örneğin ikilik tabandaki sayılar ile işlem yapılyorsa $\{0,1\}$ şeklinde kabul edilir.

Σ sembolü ile makineye verilecek girdiler (input) kümesi gösterilir. Girdi kümesi dildeki harfler dışında bir sembol taşıyamayacağı için $\Sigma \subseteq \Gamma$ demek doğru olur.

δ sembolü dilde bulunan ve makinenin çalışması sırasında kullanacağı geçişleri (transitions) tutmaktadır.

\diamond sembolü teyp bandı üzerindeki boşlukları ifade etmektedir. Yani teyp üzerinde hiçbir bilgi yokken bu sembol okunur.

q_0 sembolü makinenin başlangıç durumunu (state) tutmaktadır ve dolayısıyla $q_0 \subseteq Q$ olmak zorundadır.

F sembolü makinenin bitiş durumunu (state) tutmaktadır ve yine $F \subseteq Q$ olmak zorundadır.

Örnek Turing Makinesi

Yukarıdaki sembolleri kullanarak örnek bir Turing makinesini aşağıdaki şekilde inşa edebiliriz.

Örneğin basit bir kelime olan a^* düzenli ifadesini (regular expression) Turing makinesi ile gösterelim ve bize verilen aaa şeklindeki 3 a yı makinemizin kabul edip etmediğine bakalım.

Tanım itibariyle makinemizi aşağıdaki şekilde tanımlayalım:

$$M = \{ \{q_0, q_1\}, \{a\}, \{a, x\}, \{q_0 a \rightarrow a R q_0, q_0 x \rightarrow x L q_1\}, q_0, x, q_1 \}$$

Yukarıdaki bu makineyi yorumlayacak olursak:

Q değeri olarak $\{q_0, q_1\}$ verilmiştir. Yani makinemizin ik idurumu olacaktır.

Γ değeri olarak $\{a, x\}$ verilmiştir. Yani makinemizdeki kullanılan semboller a ve x'ten ibarettir.

Σ değeri olara $\{a\}$ verilmiştir. Yani makinemize sadece a girdisi kabul edilmektedir.

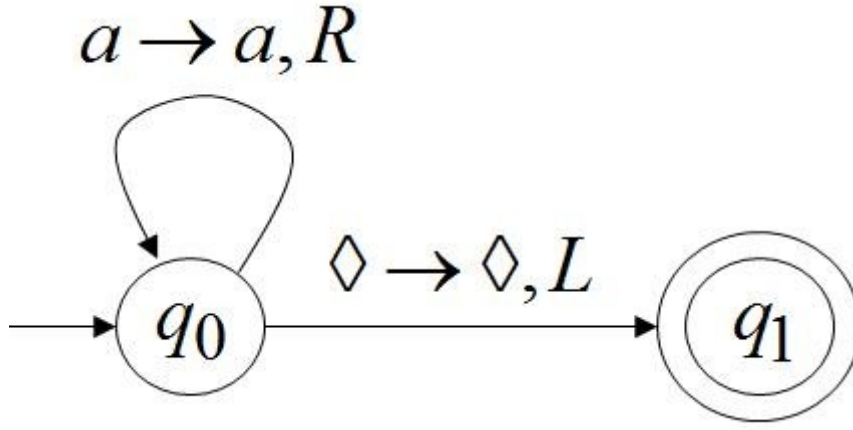
δ değeri olarak iki geçiş verilmiştir $\{q_0 a \rightarrow a R q_0, q_0 x \rightarrow x L q_1\}$ buraadki R sağa sarma L ise sola sarmadır ve görüleceği üzere Q değerindeki durumlar arasındaki geçişleri tutmaktadır.

\diamond değeri olarak x sembolü verilmiştir. Buradan x sembolünün aslında boş sembolü olduğu ve bantta hiçbir değer yokken okunan değer olduğu anlaşılmaktadır.

q_0 ile makinenin başlangıç durumundaki hali belirtilmiştir.

F değeri olarak q_1 değeri verilmiştir. Demek ki makinemiz q_1 durumuna geldiğinde bitmektedir (halt) ve bu duruma gelmesi halinde bu duruma kadar olan girdileri kabul etmiş olur.

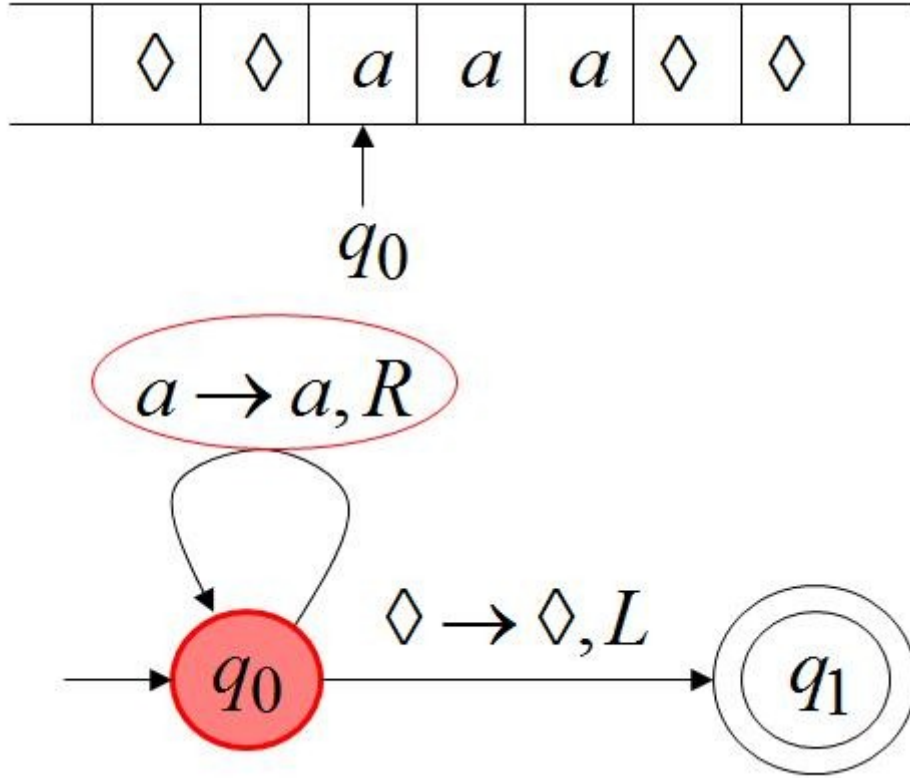
Yukarıdaki bu tanımlı görsel olarak göstermek de mümkündür:



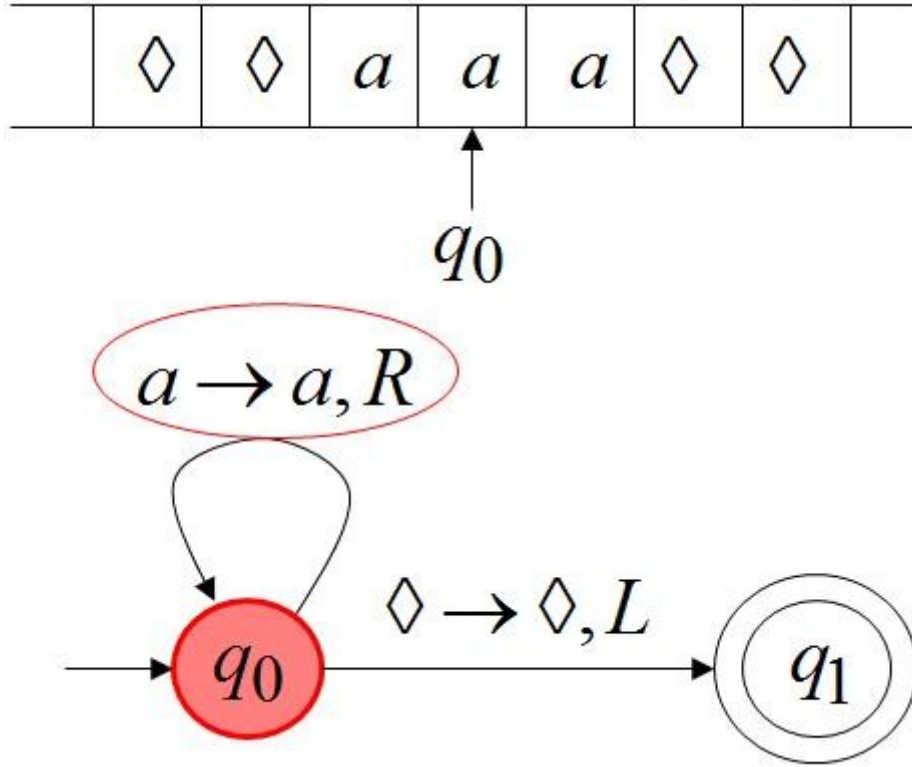
Yukarıdaki bu temsili resimde verilen turing makinesi çizilmiştir.

Makinemizin örnek çalışmasını ve bant durumunu adım adım inceleyelim.

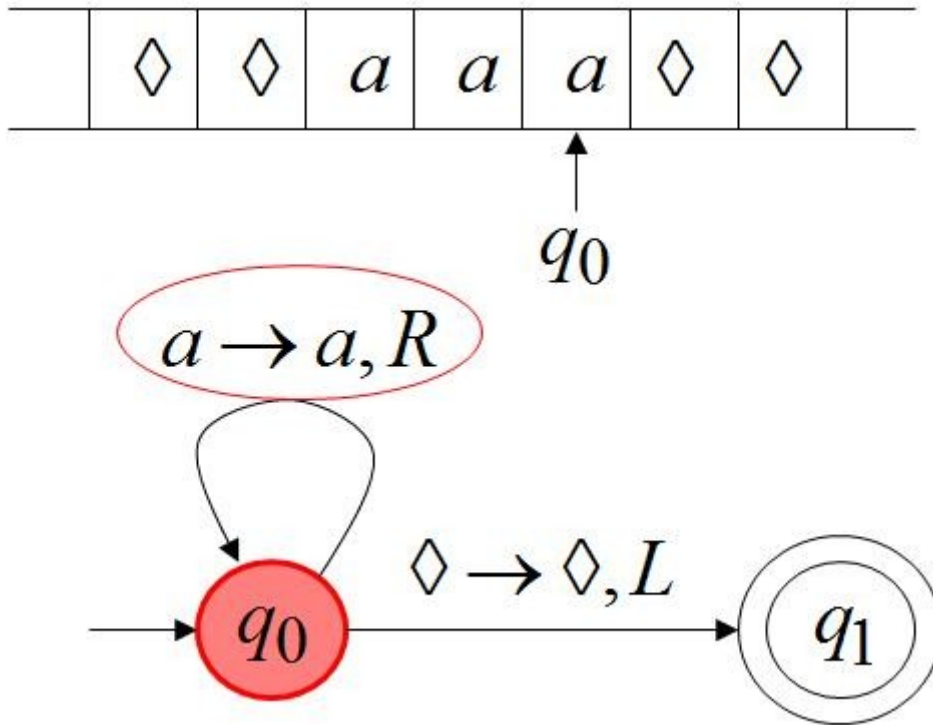
Birinci adımda bandımızda aaa (3 adet a) yazılı olduğunu kabul edelim ve makinemizin bu aaa değerini kabul edip etmeyeceğini adım adım görelim. Zaten istediğimiz de aaa değerini kabul eden bir makine yapabilmektir.



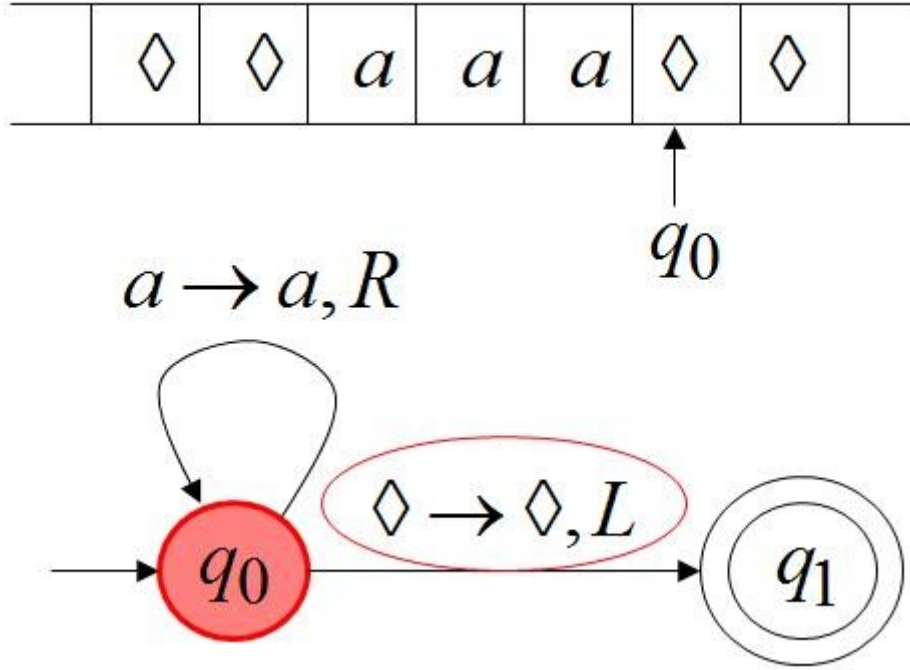
Yukarıdaki ilk durumda bant üzerinde beklenen ve kabul edilip edilmeyeceği merak edilen değerimiz bulunuyor. Makinemizin kafasının okuduğu değer a sembolü. Makinemizin geçiş tasarımına göre q_0 halinde başlıyoruz ve a geldiğinde teybi sağa sarıp yine q_0 durumunda kalmamız gerekiyor.



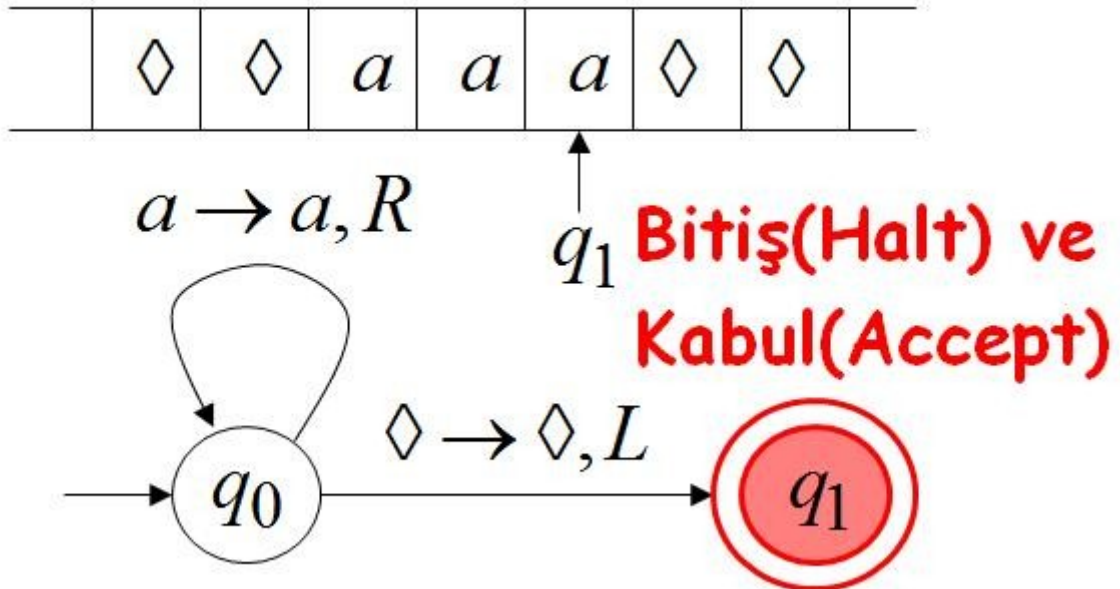
Yeni durumda kafamızın okuduğu değer banttaki 2. a harfi ve bu durumda yine q_0 durumundayken teybi sağa sarıp yine q_0 durumunda kalmamız tasarlanmıştır



3. durumda kafamızın okuduğu değer yine a sembolü olmakta ve daha önceki 2 duruma benzer şekilde q_0 durumundayken a sembolü okumanın sonucu olarak teybi sağa sarıp q_0 durumunda sabit kalıyoruz.



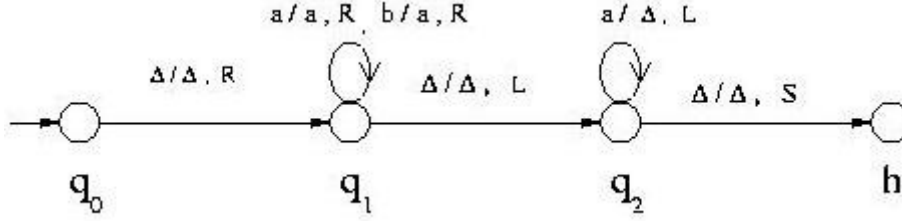
4. adımda teypten okuduğumuz değer boşluk sembolü \square oluyor. Bu değer makinemizin tasarımında q_1 durumuna gitmemiz olarak tasarlanmış ve teybe sola sarma emri veriyoruz.



Makinenin son durumunda q_1 durumu makinenin kabul ve bitiş durumu olarak tasarlanmıştı (makinenin tasarımındaki F kümesi) dolayısıyla çalışmamız burada sonlanmış ve giriş olarak aaa girdisini kabul etmiş oluyoruz.

2. Örnek

Hasan Bey'in sorusu üzerine bir örnek makine daha ekleme ihtiyacı zuhur etti. Makinemiz $\{a,b\}$ sembolleri için çalışsın ve ilk durum olarak bandın en solunda başlayarak bandta bulunan sembolleri silmek için tasarlansın. Bu tasarımı aşağıdaki temsili resimde görülen otomat ile yapabiliriz:



Görüldüğü üzere makinemizde 4 durum bulunuyor, bunlardan en sağda olan h durumu bitişi (halt) temsil ediyor. Şimdi bu makinenin bir misal olarak “aabb” yazılı bir bandta silme işlemini nasıl yaptığını adım adım izah etmeye çalışalım.

Aşağıda, makinenin her adımda nasıl davranacağı bant üzerinde gösterilmiş ve altında açıklanmıştır. Sarı renge boyalı olan kutular, kafanın o anda üzerinde durduğu bant konumunu temsil etmektedir.

◇	◇	a	a	b	b	◇
---	---	---	---	---	---	---

q0 durumunda başlıyoruz. Ve boşluk ile bandı sağa sarıyoruz:

◇	◇	a	a	b	b	◇
---	---	---	---	---	---	---

a veya b değeri okundukça bant sağa sarılmaya devam ediyor ve q1 durumunda kalıyoruz.

◇	◇	a	a	b	b	◇
---	---	---	---	---	---	---

◇	◇	a	a	b	b	◇
---	---	---	---	---	---	---

Okunan değer b ise banda geri a değeri yazılıyor

◇	◇	a	a	a	b	◇
---	---	---	---	---	---	---

www.bilgisayarkavramlari.com

◇	◇	a	a	a	a	◇
---	---	---	---	---	---	---

En sağda boşluk değerini okuyunca (◇) Sağa sarma işlemi bitiyor ve geri dönüyoruz

◇	◇	a	a	a	a	◇
---	---	---	---	---	---	---

◇	◇	a	a	a	◇	◇
---	---	---	---	---	---	---

◇	◇	a	a	◇	◇	◇
---	---	---	---	---	---	---

◇	◇	a	◇	◇	◇	◇
---	---	---	---	---	---	---

◇	◇	◇	◇	◇	◇	◇
---	---	---	---	---	---	---

Tekrar boşluk (◇) görülünce makine bitiyor.

Geri sarma işlemi sırasında a değerleri silinmiş oluyor

Netice olarak Hasan Bey'in sorusuna temel teşkil eden ve örneğin q1 üzerindeki döngülerden birisi olan b/a,R geçişi, banttan b okunduğunda banta a değerini yaz manasındadır.

SORU 42: Özyineli Sayılabilir Diller (Recursively Enumerable Languages)

Muntazam dillerden (formal languages) birisi olan ve bu özelliği ile Mantık, Matematik ve Bilgisayar bilimlerinin çalışma alanına giren bir dil çeşididir. Sınıflandırma olarak Chomsky Hiyerarşisinde (Chomsky Hierarchy) 0. seviye olan (Type 0) bu dile uygun bütün diller birer düzenli ifade (regular expression) ile gösterilebilir.

Muntazam dil (formal language) olması dolayısıyla dilde üretilen her özyineli sayılabilir kelime kümesi (recursively enumerable set) bütün kelimelerden çıkarılabilecek güç kümesinin (power set) bir alt kümesi olmak zorundadır. Bu anlamda bir özyineli sayılabilir dili aşağıdaki üç farklı tanımla tanımlamak mümkündür:

Bir dilde bulunan alfabeden üretilebilen bütün kelimeleri kabul eden dil yani Σ sembolü ile dilimizdeki alfabe yi yani harfler kümesini gösterecek olursak L ile gösterilen dilimiz Σ^* ile üretilebilen kelimeler kümesidir.

[Turing makinesi \(Turing machine\)](#) ile işlenebilen veya [hesaplanabilir bir fonksiyon \(computable function\)](#) bulunabilen dildir. Burada dikkat edilecek nokta dilin sonsuz olabileceğidir. Yani dilimizde sonsuz sayıda tekrar bulunabilir. [Turing makinesi](#) ve [hesaplanabilirlik teorisi \(computability theory\)](#) dikkatle okunacak olursa dilin sonsuz olmasının bir sakıncası yoktur. Teorik olarak sonlu zamanda bitiyor olması yeterlidir ve dil sonsuz bile olsa sonlu zamanda işleyen bir makine veya fonksiyon bulunabilir.

Şayet bir dilde üretilen bir [dizgi \(string\)](#) için bir [Turing makinesi \(turing machine\)](#) veya [hesaplanabilir bir fonksiyon \(computable function\)](#) bulunuyorsa, diğer bir ifadeyle ürettiğimiz [turing makinesi](#) şu üç durumdan birisini yapıyorsa:

- [bitmek \(halt\)](#)
- [döngü \(loop\)](#)
- red etmek (reject)

bu durumda bu dil özyineli sayılabilir dildir denilebilir.

Özyineli sayılabilir diller temel olarak [chomsky hiyerarşisinde \(chomsky hierarchy\)](#) bulunan bütün diğer dilleri kapsar. Bu anlamda hiyerarşinin en alt seviye dilidir ve diğer dillere göre çok daha esnekler.

SORU 43: Chomsky Hiyerarşisi (Chomsky Hierarchy)

Bilgisayar bilimlerinin özellikle dil alanında yapılan çalışmalarında [muntazam dilleri \(formal languages\)](#) tasnif etmek için kullanılan bir yapıdır. Literatürde Chomsky–Schützenberger hiyerarşisi olarak da geçmektedir.

Bilindiği üzere ([muntazam diller \(formal languages\)](#) veya [CFG](#) yazısından da okunabileceği üzere) [muntazam dillerin](#) dört özelliği bulunur. Bunlar özellikle [içerikten bağımsız dillerin \(context free languages\)](#) da temelini oluşturmuştur.

- sonlular (terminals)
- devamlılar (nonterminals)
- üretim kuralları (devamlıların değerlerini belirleyen geçiş kuralları)
- Başlangıç devamlısı

Örneğin aşağıdaki [içerikten bağımsız dilbilgisini \(context free grammar\)](#) ele alalım

$$S \rightarrow AB$$

$$A \rightarrow Aa \mid \varepsilon$$

$$B \rightarrow b$$

Yukarıdaki bu [CFG](#) tanımındaki sonlular (terminals) $\{a,b,\varepsilon\}$, devamlılar (nonterminals) $\{S,A,B\}$ olarak tanımlanır. üretim kuralı olarak (production rules) S,A,B 'nin açılımları

gösteren ve \rightarrow sembolü ile belirtilen satırlar bulunmaktadır. Son olarak başlangıç devamlısı (nonterminal) değeri olarak S kabul edilmiştir. Başlangıç devamlısı böyle bir kural bulunmamasına karşılık genelde S harfi (start kelimesinden gelmektedir) ile gösterilmekte ve ilk satırda bulunmaktadır.

Yukarıdaki bu CFG şayet [düzenli ifadeye \(regular expression\)](#) çevrilirse a^*b şeklinde yazılabilir. Aslında burada anlatılan değer $a^n b$ şeklinde de gösterilebilen istenildiği kadar a değeri alan (hiç almaya da bilir) ve sonra mutlaka b ile biten dildir.

Chomsky yukarıdaki şekilde tanımlanan [muntazam diller \(formal languages\)](#) için bir sınıflandırmaya gitmiş ve 4 seviye belirlemiştir.

- Type-0 (tip 0) Sınırlandırılmamış diller (unrestricted grammars)
- Type-1 (tip 1) İçerik duyarlı diller (context-sensitive grammars)
- Type-2 (tip 2) [İçerikten bağımsız diller \(context free languages\)](#)
- Type-3 (tip 3) [Düzenli diller \(regular grammars\)](#)

şeklinde 4 seviye isimlendirilmiştir. Bu seviyelerde iler gidildikçe (seviye arttıkça) dili bağlayan kurallarda sıkılaşmakta ve dil daha kolay işlenebilir ve daha belirgin (deterministic) bir hal almaktadır.

Tip-0 dilleri basitçe [turing makinesi \(turing machine\)](#) tarafından çalıştırılabilen ve bir şekilde bitecek olan dillerdir ([hesaplanabilir diller \(computable languages\)](#)). Örneğin [özyineli sayılabilir diller \(recursive enumerable languages\)](#) bu seviyeden kabul edilir.

Tip-1 dilleri için içerik duyarlı diller örnek gösterilebilir. Basitçe $\alpha A \beta \rightarrow \alpha \gamma \beta$ şeklindeki gösterime sahip bir dildir. Buradaki A devamlı (nonterminal) ve α, γ, β birer sonlu (terminal) terimdir. Bu sonlulardan α ve β boş harf (yani ϵ veya bazı kaynaklarda λ) olabilir ancak γ mutlak bir değere sahip olmalıdır (yani boş olamaz) Ayrıca bu tipte

$$S \rightarrow \epsilon$$

şeklinde bir kurala da izin verilmektedir. Burada S devamlısının sağ tarafta olmaması gerekmektedir. Bu diller doğrusal bağlı otomatlar (linear bounded automaton) ile gösterilebilir. Doğrusal bağlı otomatlar, [turing makinelerinin](#) özel bir halidir ve [Turing makinesinde](#) bulunan bantın sabit uzunlukta olduğu (çalışmanın sabit zaman sonra sona ereceği) kabul edilir.

Tip-2 diller ise [CFG](#) ile ifade edilebilen [içerikten bağımsız dillerdir \(context free languages\)](#). Bu dillerin özelliği $A \rightarrow \gamma$ şeklinde gösterilmeleridir. Buradaki γ değeri sonlular (terminals) ve devamlılar (nonterminals) olabilmektedir. Bu diller [aşağı sürüklemeli otomatlar \(push down automata PDA\)](#) tarafından kabul edilen dillerdir ve hemen hemen bütün programlama dillerinin temelini oluşturmaktadırlar. (programlama dillerinin neredeyse tamamı bu seviye kurallarına uymaktadırlar)

Tip-3 diller ise [düzeltilmiş ifadeler \(regular expressions\)](#) ile ifade edilebilen (veya üretilebilen veya parçalanabilen (parse)) dillerdir. Bu dillerin farkı

$$A \rightarrow \alpha \text{ ve}$$

$$A \rightarrow \alpha B$$

şeklindeki kurallar ile gösterilebilmeleridir.

Yukarıdaki bu tanımları bir tabloda toplamak gerekirse:

Seviye	Dil Örneği	Otomat Uygulaması	Kuralları
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite state automaton	$A \rightarrow \alpha$ ve $A \rightarrow \alpha B$

Yukarıdaki seviyeler bütün dilleri kapsamak için yeterli değildir ayrıca yukarıda gösterilen seviyelere giren yukarıdaki tablo dışında diller de bulunmaktadır.

SORU 44: Anlamsal Ağlar (Semantic Network)

Bilgisayar bilimlerinde özellikle yapay zeka çalışmalarında bilgisayarların öğrenme sürecini belirlerken, bilgisayar tarafından yapılan çıkarımların gösterilmesi ve ilişkilendirilmesi ciddi bir problemdir.

Bir bilgisayar yazılımının içinde bulunduğu durum veya karşılaştığı vaka hakkında yaptığı çıkarımları göstermesine semantics (anlambilim) ismi verilmektedir. Bilgisayarların makine öğrenmesi (machine learning) sürecinde elde ettikleri malumatların gösterilmesi (knowledge representation) ve bu malumatın çıkarımı (knowledge retrieval) işlemlerinin üzerine kurulu olan anlamsal ağlar (semantic network) basitçe bilgisayarların gösterdiği çıkarımlar arasındaki bağlardır.

Doğal dil işleme (natural language processing) konusunda önemli bir yere sahip olan ve pekçok ülkede üzerinde çalışılan wordnet projesi anlamsal ağlara bir örnek olabilir. Yani bir dilde bulunan kelimeler aslında bir dilin ifade ettiği kavramları göstermektedir. Yani peynir kelimesini basit bir kelime olarak görmek de mümkündür bir varlık olarak görmek de. WordNet'in amacı kelimeler arasındaki bu anlamsal bağları çıkararak işaretlemektir.

Anlamsal ağ kelimesi ingilizcedeki iki farklı terim için kullanılmaktadır. Semantic web ile anlatılmak istenen internet üzerinde oluşan anlamsal ağdır. Semantic network ise bir etki alanı için oluşturulmuş amaca yönelik ağdır.

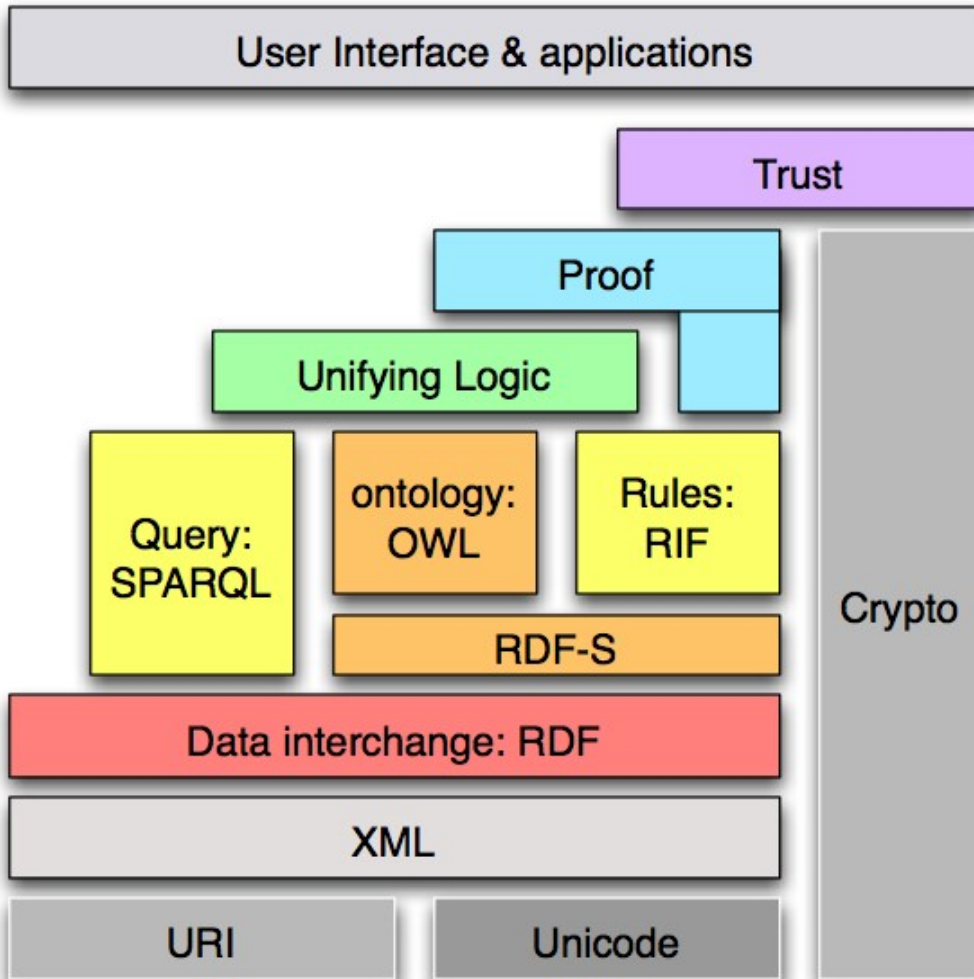
SORU 45: Mana Ağları (Sematic Webs, Anlamsal Ağ)

İnternetin (world wide web) bir alt uzayı olarak düşünülebilecek anlambilimsel ağlar, internet üzerinde bulunan ve doğal dilde yapılan yapılara bir alternatiftir.

Anlambilimsel ağlar, bir bilgi kaynağının makinelere (bilgisayarlar) tarafından işlenebilecek ve bu işleme sonucunda anlamı tam olarak anlaşılabilir ağlardır.

Mânâ ağları (anlambilimsel ağlar) üzerinde yapılan çalışmalarda henüz tam bir kesinlik yoktur. Farklı çalışma grupları, farklı standart ve alanlarda çalışmaktadırlar. Ancak Kaynak Tanım Çerçevesi (resource description framework) ve XML kullanımları neredeyse standartlaşmıştır. Örneğin varlıkbilim (onthology) seviyesinde yapılan web ontology language (OWL) çalışması RDF kullanımının bir örneği olarak görülebilir. Ayrıca malumat seviyesi gösterimler (knowledge representation) için çeşitli alternatifler de bulunmaktadır.

Mânâ ağlarının çıkış sebebi bilgisayarların malumat seviyesinde (knowldege) işlem yapma yetersizliğidir. Bir bilgisayarın işleyebileceği ve işlemi sonucunda çıkarabileceği anlam ile insanların anladığı anlam çok farklıdır. Örneğin “maymun” kelimesinin hangi dilden Türkçeye girdiğini araştırmak ve cevap olarak Asurcadan girdiğini bulmak insan için ulaşılabilir bir araştırma süreci iken, bilgisayarların bu konuda işlem yapmaları çelişik açıklamaları temizlemeleri ve bir malumat sahibi olmaları günümüz teknolojisinde insana göre oldukça ilkeldir. İşte bu sebeplerden oldayı İnternet üzerinde HTML ile yapılan ve insanların okuyup anladığı sayfaların yanında bilgisayarların anlayabildiği mana ağları türemiştir.



Yukarıda W3C'nin sitesinden alınmış mana ağı yığını (semantic web stack) resmi görülmektedir. Yukarıdaki değişik katmanlarda bulunan teknolojiler w3c tarafından standartlaştırılmış teknolojilerdir.

Yukarıdaki yığına (stack) bakılınca en alt seviyede URI (Unique resource identifier) seviyesi görülmektedir. Bu seviye mana ağlarının bulunduğu servis sağlayıcılarının internet üzerindeki adresleridir. Yine alt seviyelerden Unicode ile anlatılmak istenen ise verinin işlenmesi ve iletimi sırasında kullanılacak olan kodlama standardıdır.

Örneğin XML dosyalarının saklanması ve iletimi sırasında XML dosyalarının UTF (unicode transformation table) tablolarına uygun olması istenmektedir.

Yukarıdaki şekilde dikkat edilebilecek bir nokta ise Crypto katmanının en üst seviyeye kadar çıkıyor olmasıdır. Bunun sebebi kullanılacak olan dijital imza (digital signature) ve benzeri güvenlik uygulamalarının her katmandan kontrol edilebiliyor olması ve katmanların (örneğin XML veya RDF) güvenlik parçaları barındırmamasıdır. Yani örneğin XML teknolojisinin içerisinde doğrudan bir güvenlik çözümü söz konusu değildir. Yine kullanıcı katmanı (user interface & application) ile geri kalan mana ağı yığını (semantic web stack) arasında bulunan güven (Trust) katmanı da bu amaca hizmet etmektedir ve kullanıcının şifre katmanı (crypto) ile erişimini ayarlamaktadır.

SPARQL yine w3c tarafından standartlaştırılmış RDF sorgulama dilidir. Yani bir RDF dökümanını işleyerek sorgulamaya yarayan dillerden birisidir.

SORU 46: Dizgi Eş Şekilliliği (String Homomorphism)

Dizgiler (strings) üzerinde tanımlı bir işlemdir. Basitçe bir alfabe üzerinden çıkarılan bir dizgi (string) ile diğer bir dizgi arasındaki harf atamalarının aynı olması durumudur.

Sayısal bir örnek olması açısından üç harfli bir kelimenin ilk harfi ile ikinci harfi arasında 3 harf, ikinci ile üçüncü harfi arasında 7 harf olduğunu düşünelim. Bu tanıma uyan alfabedeki harf sayısı kadar farklı kelime bulunabilir. Örneğin adi kelimesi veya bej kelimesi bu anlamda birbirinin eş şekillisidir (homomoprhic).

Bu tanımlı daha da genişletmek mümkündür. Şayet bir kelimedeki harflerin yerine yeni harfler konulurken orjinal kelimedeki aynı harfler tek bir harf ile değiştiriliyorsa bu orjinal kelime ve yeni kelim birbirinin eş şekillisi olur.

Yani yukarıdaki örneğe dönecek olursak adi ile örneğin kat kelimesi de eş şekillidir a->k , d->a , i -> t olarak değiştirilmiştir. Ancak adi kelimesi ile mum kelimesi eş şekilli olamaz.

Bu anlamda ikame şifrelemeleri (yerine koyma şifrelemeleri, substitution cipher) birer eş şekilli örnekleridir.

SORU 47: Augmented Transition Network (ATN, Uzatılmış Geçiş Ağı)

Bilgisayar bilimlerinde özellikle de yapay zeka konusunda ve buna bağlı diğer alt dallarda (örneğin doğal dil işleme) kullanılan bir graf teori (graph theory) gösterimidir. Kelime anlamı olarak uzatılmış geçiş ağı (tehir-i intikal şebekesi) denilen ağların amacı toplanan bilgilere göre bir karar vermek ve karar verme işlemi sırasında da bir belirsizlik (ambiguity) bulunuyorsa, karar verme işlemini yeterli bilgi toplanana kadar tehir etmektir (geciktirmektir, augment). Ağın ismi de buradan gelmektedir.

Temel olarak sonlu otomatları (Finite state automats) baz alan ağlarda her düğüm (node) bir dilbilimsel (linguistic) bilgiyi içermektedir. Kirişler (edges, arcs) ise bu dilbilimsel üniteler arasındaki geçiş imkanlarını gösterir.

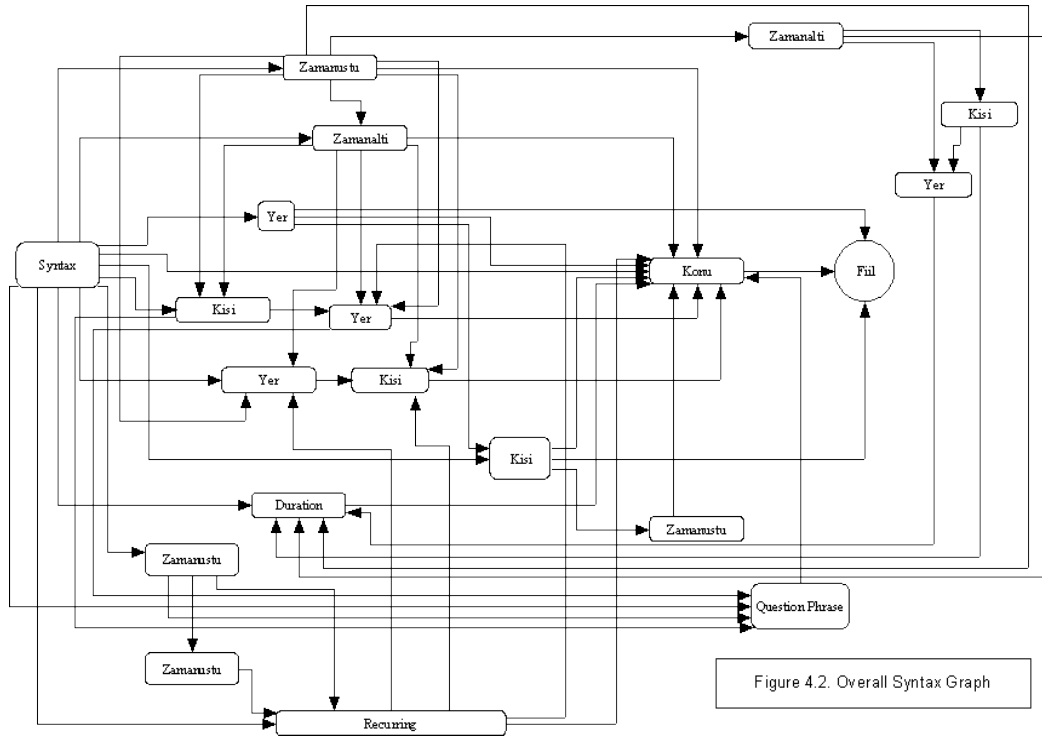


Figure 4.2. Overall Syntax Graph

Örneğin yukarıdaki şekilde Türkçe cümleler ile randevuları tutan bir takvim programı uygulaması olan yüksek lisans tezinden alınma doğal dildeki kelime grupları arasındaki geçişleri gösteren bir ATN görülmektedir.(tezin detaylarına www.shedai.net/tusa adresinden erişilebilir)

Input: "on ocak ikibin ile on mart ikibiniki arasında haftada bir ali bey ile bahçeli okulda saat onda ikişer saatlik toplantılar var"
 (There are meetings with mister ali at the school with garden between tenth of January two thousand and tenth of march two thousand and two with the period of two weeks and each one is two hours long at ten o'clock)

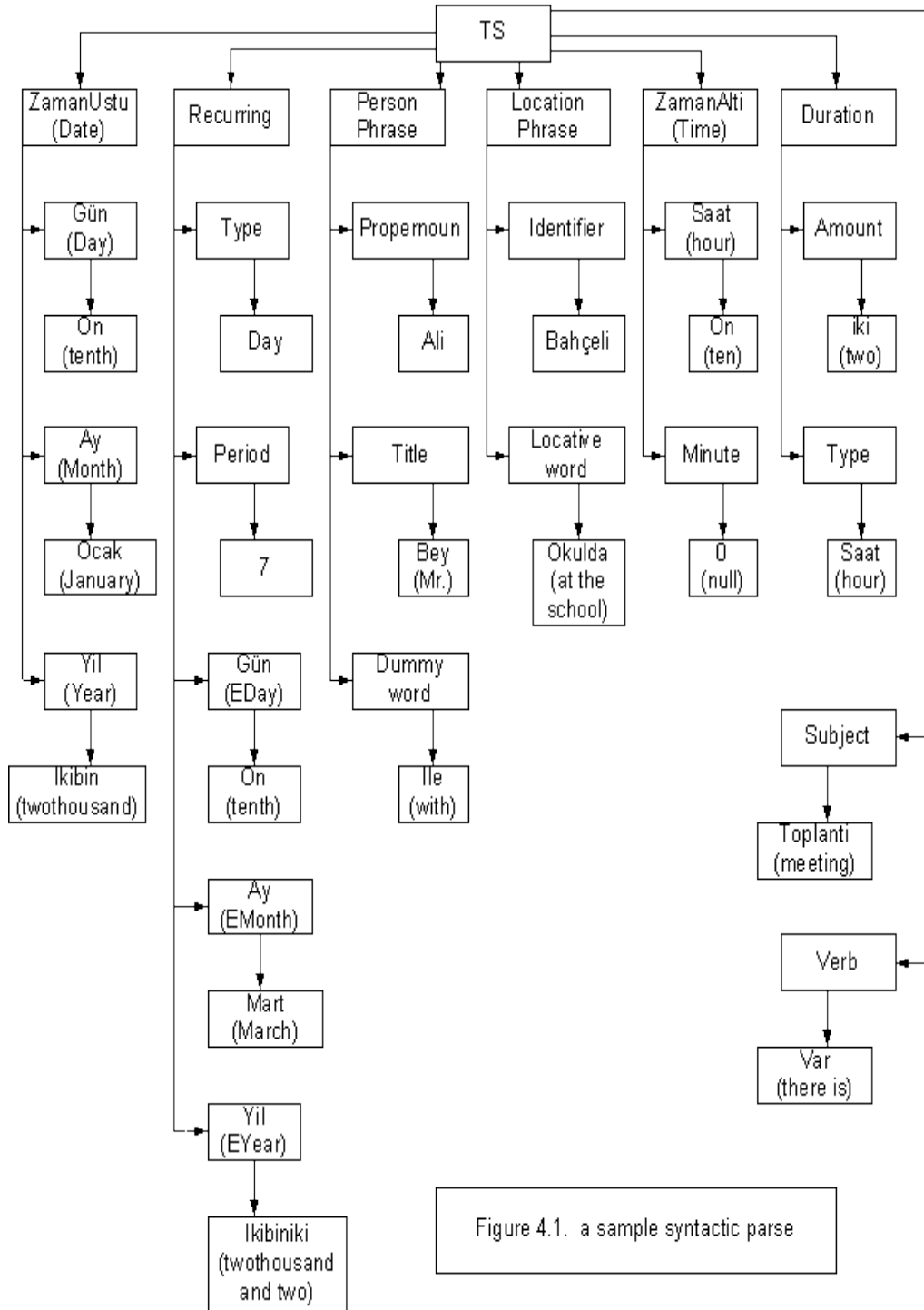


Figure 4.1. a sample syntactic parse

Yukarıda ise bir önceki şekilde görülen ATN kullanılarak “on ocak ikibin ile on mart ikibiniki arasında haftada bir ali bey ile bahçeli okulda saat onda ikişer saatlik toplantılar var” cümlesinin parçalama ağacı (parse tree) gösterilmiştir.

SORU 48: Aks-i Müfret (Palindrome)

Edebiyatta bir sanat olan aksi müfret, bir kelimenin ya da bir cümlenin baştan ve sondan okunuşunun aynı olması durumudur. Bazı kaynaklarda bedii sanatı olarak da zikredilmektedir.

Örneğin Yasin sûresi 40. ayette (Küllün fi felek) “Aya erişmek güneşe düşmez. Gece de gündüzü geçemez. Her biri bir yörüngede yüzerler.” (Yasin: 40) Ayetteki harfler şöyledir: (K L F Y F L K) ve tersten okunuşu kendisi ile aynıdır.

Genellikle 3 harfli kısa kelimelerde kolaylıkla bulunabilecek olan duruma örnekler: mum, yay, asa, ata ... şeklindedir ve baştan ve sondan okunduğunda aynı anlamı verir.

Bir yazının aksi müfret olduğunun anlaşılması için yığın (stack) yapısından faydalanılabilir. Buna göre yazının yarısı yığına konulduktan (push) sonra yazının geri kalanı ile yığından alınan (pop) değerler karşılaştırılır ve aynıysa bu yazı bir palindrome olarak değerlendirilir.

Algoritması:

*String oku,
Stringin yarısına kadar karakter karakter push et,
Stringin yarısından sonraki her karakter için bir karakter pop edip karşılaştır,
karşılaştırma sonuna kadar tutuyorsa palindromdur
değilse palindrome değildir.*

Palindrome Kontrolü Yapan Kod (Gizem Hanımına yazdığı bu kod için teşekkür ederim)

```
#include <stdio.h>
#include <stdlib.h>

void ispalindrome(char* s){
    int i=0;int son;
    for (i=0;i<strlen(s);i++){
        if (s[i]==s[strlen(s)-i-1]){
            son=1;}
        else {
            son=0;}
    }
    if (son==1){
        printf("ntrue");}
    else {
        printf("nfalse");}
}

int main(void){
    ispalindrome("radar");
    ispalindrome("kabak");
    ispalindrome("yimirta");
    getch();
}
```

SORU 49: Atomluluk (Atomicity)

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. Pekçok 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 örnektir 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 50: İçerikten Bağımsız Gramer (context free grammer, CFG)

Bilgisayar bilimlerinde, dil tasarımı sırasında kullanılan bir gramer tipidir. Basitçe bir dilin kurallarını (dilbilgisini, grammer) tanımlamak için kullanılır.

Örneğin:

$S \rightarrow a$

Yukarıdaki dil tanımında bir büyük harfle gösterilen (S) bir de küçük harfle gösterilen (a) sembolleri bulunmaktadır. Bu satır, S devamlısının(nonterminal) a sonuncusuna(terminal) dönüştüğünü göstermektedir. Kısaca dildeki kuralları ifade etmek için büyük harfli semboller devamlıları (nonterminal) ve küçük harfli semboller sonucuları (terminal) ifade etmekte, \rightarrow ok işareti ise, işaretin solundaki devamlının (nonterminal), sağındaki sembolle gösterilebileceğini ifade etmektedir.

Bir dili içerikten bağımsız (context-free) yapan, o dilin bir belirsiz aşağı sürüklememli otomat (non deterministic pushdown automata) tarafından üretilebilir olmasıdır.

İçerikten bağımsız diller, programlama dilleri olarak sıklıkla kullanılmaktadır, bilinen çoğu programlama dili aslında birer içerikten bağımsız dil özelliğindedir ve bu dillerin kurallarının tanımlandığı gramerlerde içerikten bağımsız gramerlerdir (context free grammar).

Bu anlamda, YACC gibi programlama ortamlarında, bir dil tasarlamak ve içerikten bağımsız kurallar yazarak dili tanımlamak mümkündür.

CFG gösterimi ne yazık ki doğal diller (natural languages) için kullanılamaz. Ya da kullanılsa bile bir doğal dilin tamamını kapsayacak bir CFG gösterimi çıkarılamaz. Örneğin doğal dillerde kelimebiliminin (lexicology) bir parçası olarak sıkça kullanılan uyum (agreement) veya atıf (reference) kullanımları CFG ile gösterilemeyen özelliklerdir. Yani daha net bir ifadeyle CFG gösterimi için dildeki anlamların belirli olması gerekir. Çeşitli durumlarda belirsizlik içeren doğal diller için ise bu durum imkansızdır.

CFG tanımı

Temel olarak bir içerik bağımsız gramer dört özellik içermelidir. Bunlar sonlular (terminals), devamlılar (nonterminals), bağlantılar (relation) ve başlangıç sembolü (starting symbol) olarak sıralanabilir. Bir gramerin tanımı sırasında kullanılan bu kümeler aşağıdaki şekilde yazılabilir:

$$G = (V , \Sigma , R , S)$$

Bu gösterimdeki gramer (G) , V devamlıları, Σ sonluları, R bağlantıları, S ise başlangıç sembolünü göstermektedir.

Örneğin

$$S \rightarrow aSb \mid ab$$

şeklinde tanımlanan bir dilde:

$$G = (\{S\} , \{a,b\} , \{S \rightarrow aSb \mid ab\} , S)$$

gösterimi kullanılabilir.

SORU 51: İçerikten bağımsız dil (Context Free Language, CFL)

Bilgisayar bilimlerinde bir dilin tasarımı sırasında, içerik bağımsız bir gramer ile oluşturulması durumudur. Basitçe bir aşağı sürüklemeli otomat (push down automata) tarafından kabul edilen dil çeşididir. Bazı kaynaklarda bağlamdan bağımsız dil olarak da geçmektedir.

Örneğin çok meşhur $L = \{a^n b^n , n > 0\}$ dilini ele alalım. Bu dil örneğinin bu kadar meşhur olmasının ve önemli olmasının sebebi bir düzenli ifade (regular expression) ile yazılmasının imkansız oluşu ancak içerikten bağımsız dil ile yazılmasının mümkün olmasındandır.

Şimdi bu dilin aşağı sürüklemeli otomatını aşağıdaki şekilde çıkarabiliyoruz:

$$\begin{aligned}\delta(q_0, a, z) &= (q_0, a) \\ \delta(q_0, a, a) &= (q_0, a) \\ \delta(q_0, b, a) &= (q_1, x) \\ \delta(q_1, b, a) &= (q_1, x) \\ \delta(q_1, b, z) &= (q_f, z)\end{aligned}$$

$$\delta(state_1, read, pop) = (state_2, push)$$

Yukarıdaki PDA (push down automaton) tasarımında dikkat edilirse iki durum (state) arasındaki geçişler ile yukarıdaki dili tasarlamak mümkündür. Bu sayede bu dilin bir içerik bağımsız dil olduğu söylenebilir.

Ayrıca yukarıdaki tanımda kullandığımız ” içerik bağımsız bir grammer ile oluşturulması durumu” ifadesini de açıklayarak buna da bir örnek verelim ve dilimizin ($L = \{a^n b^n, n > 0\}$) CFG (context free grammer, içerik bağımsız gramer) karşılığını aşağıda yazalım:

$$S \rightarrow aSb \mid ab$$

Yukarıdaki yazılışta, dilin sonucu ab veya aSb olarak çıkacaktır ancak S devamlısı (nonterminal) bitmek için bir sonuncuya (terminal) ihtiyaç duyacaktır bu değer de yine ab olacaktır.

Sonuçta yukarıdaki gramer ile istenilen uzunlukta sırasıyla a ve b lerden oluşan dil tasarlanabilir ve üretilen bütün dillerde a’nın sayısı ile b’nin sayısı eşittir.

SORU 52: EBNF (Uzatılmış BNF, Extended Backus Normal Form)

Bilgisayar bilimlerinde dil tasarımı konusunda kullanılan [backus normal şeklinin \(backus normal form\)](#) özel bir halidir. Basitçe standart BNF’te yazılan kuralların birleştirilerek daha sade yazılmasını hedefler.

Bu durumu aşağıdaki örnek üzerinden görebiliriz:

Örneğin BNF olarak yazılan dilimize göre:

$$\langle \text{EGER} \rangle ::= \text{if}(\langle \text{KOSUL} \rangle) \mid \text{if}(\langle \text{KOSUL} \rangle) \text{ else}$$

şeklinde bir satırı bulunsun. Bu satırın anlamı dilimizde bir EGER döz dizilimi (syntax), if komutu ve parantez içinde bir koşuldan oluşabilir veya bu if ve parantez içerisindeki koşulu bir else komutu izleyebilir.

Yukarıdaki bu BNF yazılımını EBNF olarak aşağıdaki şekilde yazabiliriz:

$$\langle \text{EGER} \rangle ::= \text{if}(\langle \text{KOSUL} \rangle) [\text{else}]$$

Yukarıdaki bu yeni satırda dikkat edileceği üzere köşeli parantezler arasında bir else komutu bulunmaktadır. Bunun anlamı, EGER komutu “if(KOSUL)” olarak tanımlanır ve şayet

istenirse bu komuta ilave olarak else komutu eklenebilir. Yani köşeli parantez içerisindeki komut isteğe bağlıdır.

Yukarıdaki bu yeni yazılım aslında sadece gösterimde bir farklılık oluşturmaktadır. Bunun dışında, EBNF'in kullanım alanı ve işlevi BNF ile aynıdır.

EBNF'in BNF'ten farklı olarak getirdiği ifade şekilleri aşağıda listelenmiştir:

İfade		Kullanımı
Tanımlama	definition	=
Üleştirme	concatenation	,
Bitirme	termination	;
Seçim (Veya)	separation	
Çift Tırnak	double quotation marks	" ... "
Tek Tırnak	single quotation marks	' ... '
İsteğe bağlı	option	[...]
Tekrarlı	repetition	{ ... }
Gruplama	grouping	(...)
Yorum	comment	(* ... *)
Özel dizilim	special sequence	? ... ?
Hariç	exception	-

Yukarıdaki tabloda ilk 6 ifade standart BNF gösteriminde de kullanılan ifadelerdir. Ancak son 6, koyu renkle yazılmış ifade EBNF için gelen yeni eklentilerdir.

Bu kullanımlardan isteğe bağlı (option) olma durumunu gördük. Şimdi diğer durumları inceleyelim:

Tekrarlı ifade ({ } işaretleri arasındaki ifadeler), [düzenli ifadelerde \(regular expression\)](#) kullanılan * işlemine benzetilebilir. Bu işlem basitçe bir bilginin istenildiği kadar tekrar edilmesi anlamına gelir.

Örneğin programlama dillerinin çoğunda kullanılan C tipi yorum'u düşünelim (comment). Bu yorumlarda istenilen kelimeler yazılabilir. Bu durumda yorum satırının tanımı aşağıdaki şekilde olabilir

<YORUM> ::= "/*", { <harf> }, "*/"

<harf> ::= a | b | ... | z

Yukarıdaki EBNF tanımında a'dan z'ye kadar olan harfler, <harf> olarak tanımlanmış, ardından bu tanım <YORUM> içerisinde istenildiği kadar tekrarlanabilir anlamında { } işaretleri arasına yerleştirilmiştir.

EBNF'de ilave olarak dil tasarımcısının istediği yere kendi yorumlarını eklemesi de mümkündür. Buna göre tasarımcı (* *) işaretleri arasına istediği bilgiyi yazabilmektedir. Bu bilgi BNF işlemine tabi tutulmamaktadır.

Pasif saldırılar (Passive Attacks)

olarak sıralanabilir. İlk saldırı yönteminde saldırgan aktif olarak rol oynar ve sistemin içerisine dahil olur. Sistemi savunan tarafın, saldırganı yakalama veya tepit etme ihtimali yüksektir.

İkinci saldırı şekli olan pasif saldırılarda ise, saldırgan taraf pasif davranmakta ve çoğu zaman sadece sistemi gözetlemekle yetinmektedir. Bu saldırı şeklindeki saldırganın yakalanması çoğu zaman daha güç olmaktadır.

Pasif saldırı yöntemlerine örnek olarak Mesaj içeriklerinin açılması (release of message content) veya Trafik analizi (traffic analysis) gibi yöntemler sayılabilir.

Aktif saldırı yöntemleri ise Taklit (masquerade), Hizmet Reddi (Denial of Service) , Tekrarlama (Replay), Mesaj Değiştirme (Modification of Message) ve benzeri yöntemler olara sıralanabilir.

SORU 55: Genetik Algoritmalar (Genetic Algorithms)

Bilgisayar bilimlerinin doğa bilimlerinden (biyoloji) öğrendiği ve kendi problemlerini çözmek için kullandığı bir yöntemdir. Bu algorithmada 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 (çaprazlama 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.

Ayrıca çaprazlama yöntemleri için aşağıda sıralanan çeşitli çaprazlama tipleri kullanılabilir:

- [Parçalı Eşleme Çaprazlaması \(Partially Match Crossover\)](#)
- [Sıralı Çaprazlama \(Order Crossover\)](#)
- [Döngü Çaprazlaması \(Cycle crossover\)](#)
- [Kenar Sıralama Çaprazlaması \(Edge Recombination Crossover\)](#)

Yukarıda sayılan çaprazlama yöntemlerinin dışında mutasyon için de çeşitli alt seçenekler bulunmaktadır:

- [Tersleme \(Inversion\)](#)
- [Ekleme \(Insertion\)](#)
- [Çıkarma \(Displacement\)](#)
- [Yer Değiştirme \(Reciprocal Exchange, Swap\)](#)

SORU 56: Java Bean

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 57: Kaba Kuvvet Algoritması (Brute Force Attack)

Veri güvenliği konusundaki en basit saldırı yöntemidir. Bir işin çok zeki olmayan ama güce dayalı çözümü misali her zaman en uzun çözüm yoludur ve her zaman bir çözüme ümidi vardır 😊

Basitçe bir şifreli metnin alabileceği bütün anahtar ihtimalleri veya bütün açık metin ihtimallerinin denenmesine dayanır.

Örneğin en basit şifreleme yöntemlerinden Kaydırma Şifrelemesini (Shift Cipher) ele alalım. Anahtar uzunluğu en fazla 26 olabilir (türkçe için 29). Kaba kuvvet saldırısında bütün bu ihtimaller denenir. Yani şifreli metin 1 kaydırılır ve okunmaya çalışılır, 2 kaydırılır ve okunmaya çalışılır, ve böylece anlamlı bir metin çıkana kadar bütün ihtimaller denenir.

Kaydırma şifrelemesi gibi ilkel bir şifrelemede hemen sonuç veren oldukça etkili bu yöntem gelişmiş ve karmaşık saldırılarda giderek başarısız hale gelmektedir.

Ayrıca şifre çözme işlemleri sırasında sözlük saldırısı (Dictionary attack) kullanılarak hızlandırmak mümkündür. Bu yöntemle anahtarın alabileceği ihtimallerin azaltılması ve mümkün olan bir küçük kümenin denemesi mümkündür.

SORU 58: CASE Araçları (Case tools)

Bilgisayar bilimlerini yazılım mühendisliği alanında kullanılan araçların genel ismidir. Computer Aided Software Engineering kelimelerinin baş harflerinden oluşan bu araçların amacı yazılım geliştirme süreçlerinin kontrol edilebilmesi , ölçeklenebilmesi ve kolay yönetilebilmesidir.

CASE araçlarını aşağıdaki kriterlere göre birkaç bölümde incelemek mümkündür:

Kod üreten yazılımlar (Source code generation tools)

UML (Unified modelling language) tasarım araçları

Veri Modelleme araçları (Data Modelling tools)

Versiyon kontrol ve Yazılım ayarlama araçları (Configuration tools)

Model Dönüşüm yazılımları (Model Transformation)

Kod Yeniden üretim yazılımları (Code refactoring)

SORU 59: En kötü durum analizi (Worst Case Analysis)

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 teoremda 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:

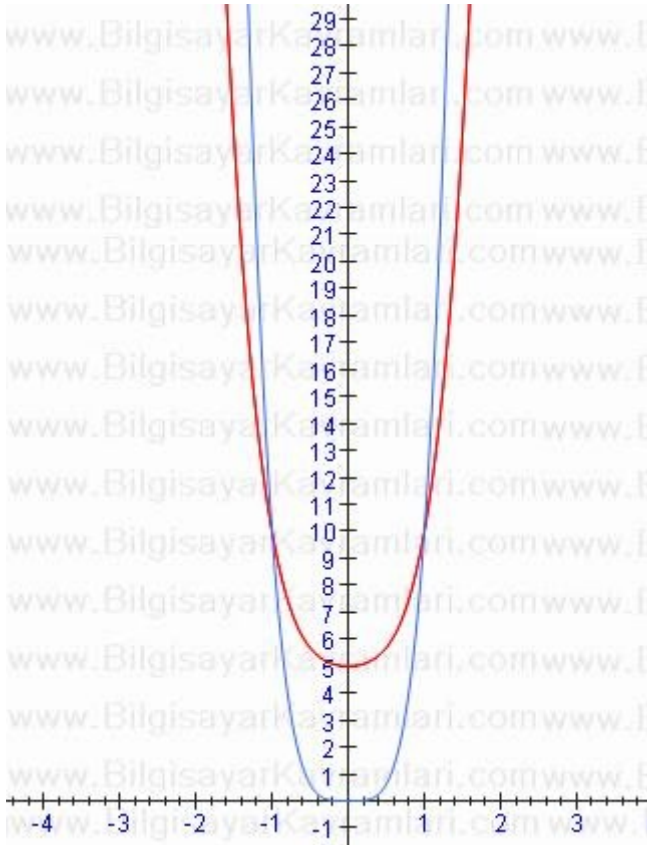
$$f(x) = 3x^4 + 2x^2 + 5$$

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, $f(x) \leq c \cdot g(x)$

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.

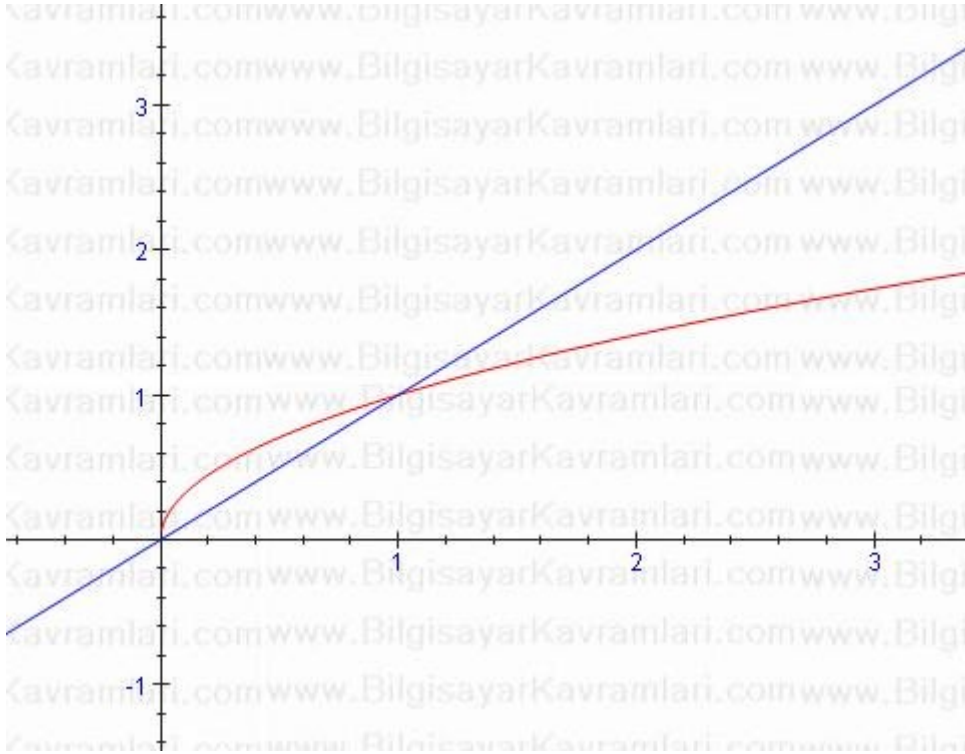
$f(x) = o(g(x))$ için

$f(x) < c \cdot g(x)$ şartı sağlanmalıdır.

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

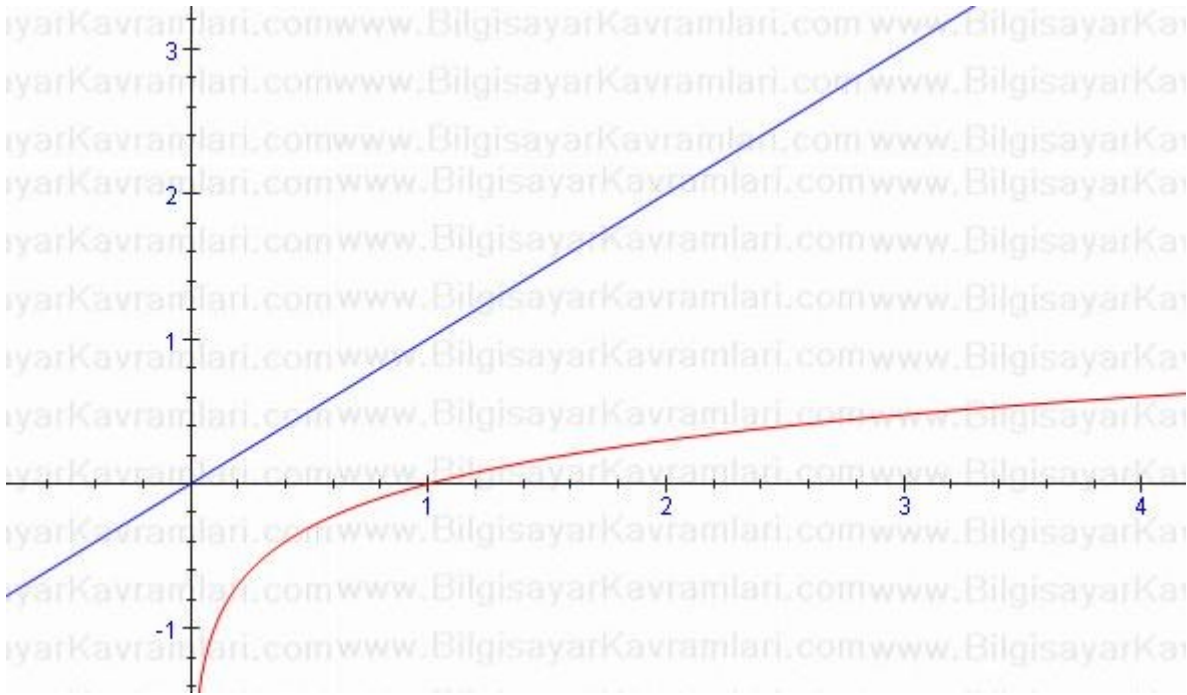
Bazı örnekler aşağıda verilmiştir:

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



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 60: Sezgisel Algoritmalar (Buluşsal Algoritmalar, Heuristic Algorithms)

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.

Sezgisel olarak kullanılabilecek bazı algoritmalar aşağıdadır:

- [Minimax Ağaçları](#)
- [Simulated Annealing \(Benzetimli Tavlama\) algoritması](#)
- [Tepe Tırmanma Algoritması \(Hill Climbing Algorithm\)](#)

- [Arı sürüşü arama algoritması \(bees search algorithm\)](#)
- [A* Araması \(astar search\)](#)
- [Geri izleme \(backtracking\)](#)
- [Işın arama \(beam search\)](#)

SORU 61: Bayes Ağları (Bayesian Network)

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 özelliği istatistiksel ağlar olmaları ve [düğümler \(nodes\)](#) arası geçiş yapan [kolların \(edges\)](#) istatistiksel kararlara göre seçilmesidir.

Bayes ağları yönlü döngüsüz 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:
 $P(B) = P(I)P(B|I) + P(II)P(B|II) + P(III)P(B|III),$

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

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 hastalık testinde (+) nufusta pozitif sonucun doğru bilinme olasılığı 0.99 ve (-) negatif sonucun doğru bilinme olasılığı 0.95 olarak veriliyor. Hastalığın ise bütün nufusta 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

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

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 62: Linear Programming (Doğrusal Programlama)

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 bölmeleridir.

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:

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

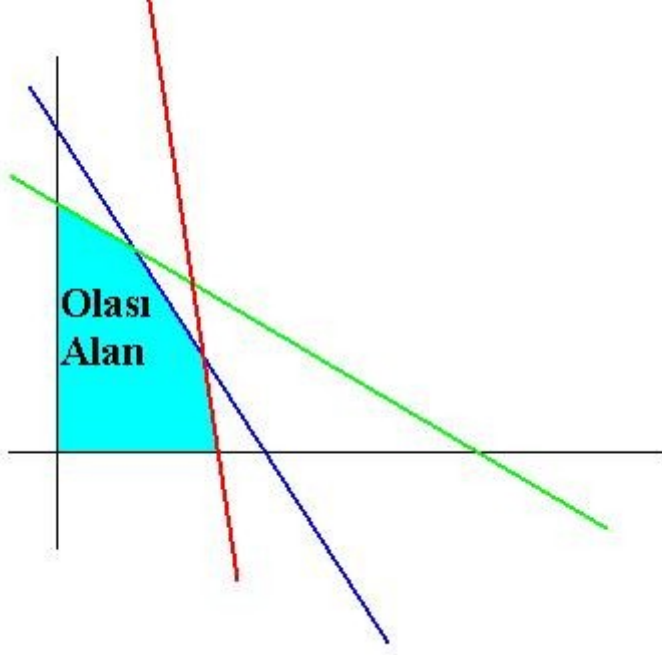
Amacımız bu ifadeyi azami hale getirmektir (maximisation)

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 ulaşılacak 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 63: Entropi (Entropy, Dağınım, Dağıntı)

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 64: Self Organizing Maps (Özdüzenleyici Haritalar)

Özdüzenleyici haritalar, yapay sinir ağlarının özel bir biçimidir ve eğitimleri sırasında [gözetimsiz eğitim](#) kullanılmaktadır.

İlk kez Kohonen ismindeki finlandiyalı bilim adamı tarafından geliştirildiği için kohonen haritası (kohonen map) ismi de verilen bu ağlar diğer bütün yapay sinir ağları gibi iki farklı şekilde çalışmaktadır.

İlk çalışma şeklinde sistem kendini eğitmektedir. Bu çalışma şeklindeyken rekabetçi öğrenme (competitive learning) kullanılır.

İkinci çalışma şekli olan haritalama merhalesinde ise ağ, gelen yeni girdiyi doğru haritalamak için çalışır.

Temel olarak çok boyutlu girdilerin (multi dimensional inputs) daha az boyuttaki çıktılara indirgenmesine dayanan çalışma mantığı problemin basitleştirilmesini amaçlayan bir boyut azaltma (dimension reduction) işlemidir.

Yapısal olarak [ileri beslemeli ağlara \(feed forward networks\)](#) örnek olabilecek olan SOM, çok küçük miktardaki nöronlar için [k-ortalama \(k-means\)](#) algoritmasına benzer davranmaktadır. Sayının artması ile SOM'un da farkı ortaya çıkmaktadır.

Algoritmanın çalışması aşağıda anlatılmıştır:

1. Ağımızdaki nöronların ağırlık değerlerini rasgele olarak başlatıyoruz
2. Giriş vektörlerini alıyoruz. (Sistemdeki hedef vektörlerimiz)
3. Haritadaki bütün değerler dolaşılıyor ve :
 1. Giriş vektörü ile dolaşılmakta olan harita değeri arasındaki mesafe öklit mesafesi (euclid distance) olarak hesaplanır.
 2. En kısa mesafeye sahip olan düğüm alınır (bu yönteme en uygun (best matching unit , BMU) ismi verilir)
4. Bu seçtiğimiz en uygun düğüme komşu olan bütün düğümler güncellenerek giriş vektörüne yaklaştırılır. (Aşağıdaki formül kullanılır):
 1. $\mathbf{Wv}(t+1) = \mathbf{Wv}(t) + \Theta(t)\alpha(t)(\mathbf{D}(t) - \mathbf{Wv}(t))$
5. $t < \lambda$ olduğu sürece 2. adıma dönülerek işlemler tekrar edilir.

Yukarıdaki formüldeki değerler kısaca aşağıda açıklanmıştır:

t = şimdiki adım

λ = adım üzerindeki zaman limiti
 \mathbf{w} = şimdiki ağırlık vektörü (weight vector)
 \mathbf{D} = hedeflenen giriş değeri
 $\Theta(t)$ = komşuluk fonksiyonu
(en uygun komşudan ne kadar uzağa gidileceği)
 $\alpha(t)$ = zamana bağlı öğrenme limiti

Yukarıda verilen algoritmanın pekçok dilde uygulanması mümkündür. Algoritmadan da anlaşılacağı üzere yaşanan sorunlardan birisi zamandır. Algoritma n girdili bir sistem için vektörlerin oluşturulması ve bütün elemanların üzerinden her döngüde geçilmesi zaman problemi çıkarmaktadır. Ancak SOM'un boyut küçültücü özelliği göz ardı edilmemelidir. Bu özelliğin doğru kullanılması ile çok uzun süreler alacak işlemler kısa zamanda çözülebilir.

Ayrıca bu algoritmanın, müteharrik özdüzenleyici haritalar (dinamik özdüzenleyici haritalar, dynamic SOM) ismi verilen bir yaklaşımla giriş değerlerinin sabit olmayıp sürekli değişim içinde olduğu örnekler üzerinde çalışan bir uygulaması da vardır.

SORU 65: K-Ortalama Algoritması (K-Means Algorithm)

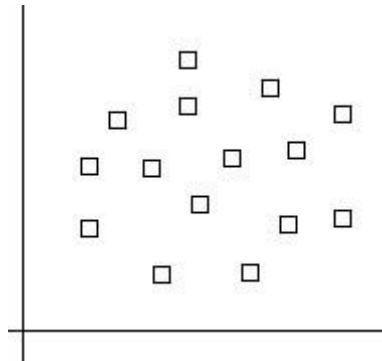
Sınıflandırmada (clustering) kullanılan algoritmalarından birisidir. Amaç [özellik çıkarımı \(Feature extraction\)](#) yapılmış bir grup verinin birden fazla sınıfa göre doğru sınıflandırılmasıdır.

Kullanılan matematiksel yöntem her sınıf için merkez belirlenen noktaya uzaklığa (aynı zamanda bu hata miktarıdır) göre yeni sınıfların yerleştirilmesidir.

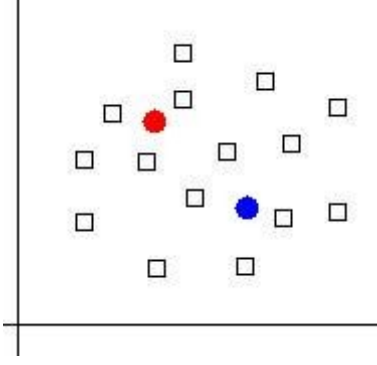
Algoritma temel olarak 4 aşamadan oluşur:

1. Sınıf merkezlerinin belirlenmesi
2. Merkez dışındaki örneklerin mesafelerine göre sınıflandırılması
3. Yapılan sınıflandırmaya göre yeni merkezlerin belirlenmesi (veya eski merkezlerin yeni merkeze kaydırılması)
4. Kararlı hale (stable state) gelinene kadar 2. ve 3. adımların tekrarlanması

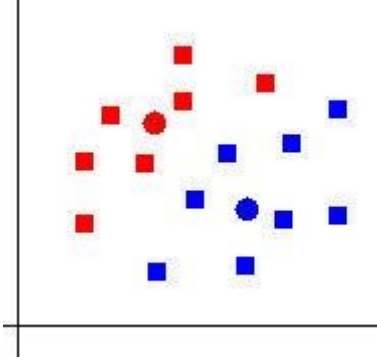
Çalışmayı daha net anlamak için aşağıdaki örnek uzaya dağılmış olan örnekleri inceleyelim:



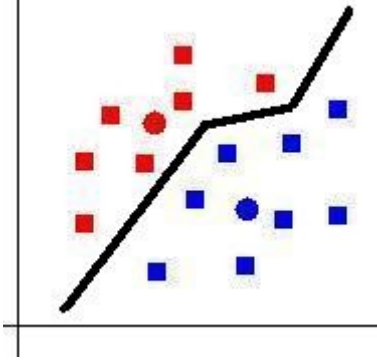
Yukarıda verilen ve uzayda koordinatları kodlanmış olan örnekler için iki adet hedef küme tanımlıyoruz. (iki sınıf ve bu sınıfların karakterlerini tanımlıyoruz)



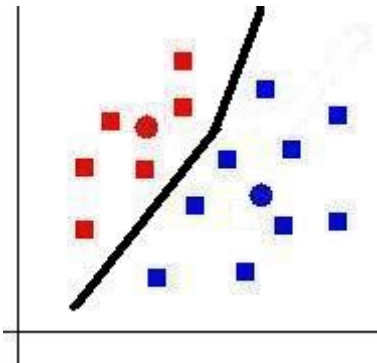
Bu sınıf tanımlarına uzaklıklarına göre (örneğin [öklit mesafesi \(euclid distance\)](#)) bütün örneklerimizi sınıflandırıyoruz. (hangi renge daha yakınsa)



Oluşan sınıfları ayıran bir hat aşağıdaki şekilde çizilebilir:



Daha önceden sınıflandırdığımız örneklerin merkezlerini buluyoruz. (yuvarlak ile gösterilen ve sınıf karakteristiğini temsil eden ilk örneklerin yerini değiştirmek olarak da düşünülebilir)



Merkezleri hareket ettirdikten sonra örneklerden bazıları yeni merkezlere daha yakın olabilir. Buna göre örnek kümelerimizin sınıflandırılmasını güncelliyoruz.

Yukarıda son hali gösterilen k-means algoritmasında yeni merkezler ve her örneğin hangi sınıfa girdiği bulunmuştur.

SORU 66: HTML (Hyper Text Markup Language)

HTML, hipermetin işaretleme dili.

Kısaca İnternet sayfalarının kodlanması için kullanılan dildir. Temel olarak bütün internet tarayıcılarının desteklediği dildir. Teknoloji basitçe şu şekilde çalışır:

- HTML dilinde hazırlanmış bir dosya sunucuda bulunmaktadır
- İnternette gezen birisi bu dosyayı talep eder
- Dosya isteyen kişiye yollanır
- Dosyayı indiren kişi internet tarayıcısında bu dosyayı gösterir

Yukarıdan da anlaşılacağı üzere HTML dilini çalıştıran ve sonuçları gösteren taraf istemci (client) tarafıdır. İstemci tarafında internet tarayıcısı olarak güncel uygulamalardan herhangi birisinin (örneğin Mozilla Firefox, Internet Explorer, Opera gibi) yüklü olması yukarıdaki işlemleri yapmak için yeterlidir.

HTML dilinin standartları W3C (W 3 consortium, w3 konsorsiyumu, w3birliği) tarafından belirlenmektedir. Kısaca her web sitesinin başında olan 3 tane w yani WWW (world wide web, dünya çapında ağ) harfleri w3 olarak kısaltılmıştır. Bu birlik internet üzerinde güncel olan pekçok teknolojinin standartlarının belirlendiği ve üzerinde araştırmaların yapıldığı bir birlikteliktir ve bütün dünyadan çok sayıda şirket ve organizasyon tarafından desteklenmektedir. Birliğe www.w3c.org web adresinden erişilebilir.

Bu sitede HTML nedir sorusuna: “HTML web üzerinde yayın dilidir” şeklinde cevap verilmektedir.

Bu yazı yazılırken HTML 5.0 standartlarının yayınlandığı w3c çeşitli aralıklarla yeni eklentilerle birlikte HTML üzerinde değişiklikler yapmaktadır.

HTML dilinin gelişimine bakılacak olursa aşağıdaki adımları görebiliriz:

1991 yılında CERN’de HTML’in ilk etiketlerini (tags) içeren liste yayınlandı. 12 komuttan oluşan bu liste 1992 yılında güncellendi

1993 yılında ilk defa HTML ismi IETF (internet engineering task force) tarafından yayınlandı ve standartları belirlendi

1993 yılında yaşanan sıkıntılar sonucunda HTML+ isimli bir standart ortaya atıldıysa da 1995 yılında gelen HTML 2.0 ile bu dilde eklenti olarak gelen tablolar, metin hizalama ve figür gibi eklentiler için çözümler sunuldu

1995 yılında HTML 2.0 IETF tarafından standartlaştırıldı. Ayrıca 1997 yılına kadar, form üzerinden dosya yüklemek, tablolar, istemci tarafı resim haritaları (isamap) ve dünya çapındaki dil desteğinin artırılması gibi eklentiler yapıldı.

1997 yılında ilk defa W3C tarafından bir standart olarak HTML 3.2 duyuruldu ve IETF , HTML ile ilgili bölümünü lağvederek bu bölümü tamamen W3C'e devretti. HTML 3.2 ile gelen yenilikler formüllerin gösterimindeki kolaylıklardı. İlk defa MathML isimli formül gösterim dili standartlaştı ayrıca yazıların yanıp sönmesi (blink) veya kayan yazı (marquee) gibi görüntüsel eklentiler yapıldı.

1997 yılının sonunda HTML 4.0 üç alternatif ile ortaya çıktı. En büyük özelliği biçim sayfalarını (style sheet) desteklemesi olan HTML 4.0 Strict ile geçmişten gelen ve Netscape ağırlıklı etiketleri tamamen yasaklıyor, Transitional ile bu etiketlere izin veriyor ve Frameset ile de sadece frame etiketlerini destekliyordu.

2000 yılında XML desteği ile birlikte XHTML dili ortaya çıktı

2008 yılının başında ise HTML 5.0 bir çalışma projesi olarak gündemde ve resmi olarak kullanılmayı bekliyor

SORU 67: SMTP (Simple Mail Transport Protocol)

SMTP, Simple Mail Transport Protocol , Basit Mektup İletim Merasimi kelimelerinin baş harflerinden oluşan ve isminden de anlaşılacağı üzere internet üzerinde mektuplaşmaya (mailing) yarayan bir protokoldür.

İnternet üzerinde mektup okuyan kişiler bilindiği üzere anlık olarak internette bulunmayabilirler. Bu tip mesajlaşmalar anlık ileti (instant mesagging) amacına yönelik özel yazılımlarla yapılmaktadır. Bunun yerine her kullanıcının mektup sunucusu (mail server) üzerinde bir posta kutusu bulunmakta ve bu kutular üzerinde mektupları birikmektedir. Ardından kullanıcı bağlanarak kutusundaki mektupları okumaktadır. Bu işin yapıldığı ve kullanıcıların mektuplarını okuduğu protokol ise POP3 (post office protocol) olarak bilinmektedir.

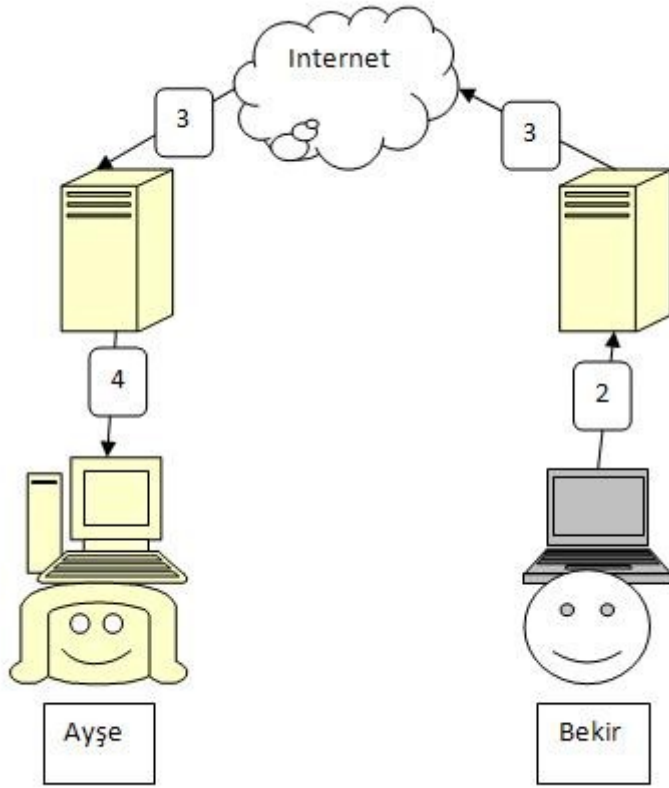
SMTP protokolü mektupları okurken değil; mektupları gönderirken kullanılan protokoldür. Bu protokolde amaç gönderilecek olan mektubun hedefteki alıcının sunucusuna iletilmesidir. Bu iletim yapıldıktan sonra zaten kullanıcının posta kutusunda mektup bekleyecektir. Dolayısıyla SMTP'nin birinci görevi hedef sunucuyu bulmak ve alıcının kutusuna mektubu bırakmaktır.

SMTP protokolü 25. port üzerinden çalışır. 3 temel aşamadan oluşur:

- Selamlaşma (handshaking)
- İletim (message transfer)
- Kapatma (closure)

İlk adımda sunucular aralarında iletişimi başlatmak için gerekli olan bilgileri geçirirler. Ardından mesaj transferi başlar ve mesaj (ya da mesajlar) yollanır. Son olarak aradaki bağlantı kapatılır.

SMTP protokolünü kullanan sunucular arasında mesajlaşma yapılırken bir sıra (queue) kullanılır. Yani bir sunucunun ileteceği birden çok mesaj varsa bu mesajlar sırasıyla iletilirler ve beklemekte olan mesajlar için bir sıra (queue) kullanılır.



Yukarıdaki şekilde bulunan ve numaralandırılmış olan 6 adımı aşağıda açıklamaya çalışalım:

1. adımda Bekir, Ayşe'ye bilgisayarında bulunna program üzerinden (user agent) mektup yazmaktadır.
2. adımda Bekir mektubunu yollar ve Bekir'in bilgisayarındaki program, Bekir'in SMTP sunucusuna (smtp server) bağlanarak mektubu sunucuya yükler
3. adımda Bekir'in SMTP sunucusu internet üzerinden Ayşe'nin sunucusunu bulur ve bu sunucuya mektubu teslim eder.
4. adımda Ayşe mektubu kendi bilgisayarındaki program marifetiyle (user agent) görüntüler.

Bu adımlar sırasında olan mesajlaşma SMTP protokolünde aşağıdakine benzer bir haldedir:

```
S: 220 bilgisayar kavramlari.com
İ: HELO sadievrenseker.com
S: 250 Hello sadievrenseker.com, pleased to meet you
İ: MAIL FROM: <bekir@sadievrenseker.com>
S: 250 bekir@sadievrenseker.com... Sender ok
İ: RCPT TO: <ayse@bilgisayarkavramlari.com>
S: 250 ayse@bilgisayarkavramlari.com ... Recipient ok
İ: DATA
S: 354 Enter mail, end with "." on a line by itself
```

```
İ: Ben Bekir;  
İ: Bilgisayar kavramlarında yeni neler var?  
İ: .  
S: 250 Message accepted for delivery  
İ: QUIT  
S: 221 sadievrenseker.com closing connection
```

Yukarıdaki mesajlaşmada, bilgisayararkavramlari.com sunucusuna bağlanarak bekir isimli kullanıcının mesajı bekir@sadievrenseker.com adresinden ayse@bilgisayararkavramlari.com adresine gönderilmiştir (İ: istemci (sadievrenseker.com), S: sunucu tarafıdır (bilgisayararkavramlari.com)). mesaj:

“Ben Bekir;

Bilgisayar kavramlarında yeni neler var?”

şeklinde. Bu mesajlaşmanın ilk 3 satırı sunucular arası selamlaşma (handshaking), son 2 satırı da kapatma (closure) işlemidir. Bu satırlar dışındaki satırlar mesajın gönderilmesi işlemidir. Şayet birden fazla mesaj gönderilecek olsaydı istemci yeniden MAIL FROM komutuyla yeni bir mektup transferi başlatacaktı.

SORU 68: Parçalama Ağacı (Parse Tree)

Parçalam işlemi (parsing) bilgisayar bilimlerinde çeşitli amaçlar için kullanılmaktadır. Özellikle de dil ile ilgili işlemlerin hemen hepsinde ihtiyaç duyulan bir işlem. Örneğin bir programlama dilinde yazılan komutların algılanması için öncelikle kelimelerin parçalanması (parse) gerekir. Benzer şekilde doğal dil işleme (natural language processing) işlemlerinde de doğal dilde bulunan kelimelerin algılanması bir parçalamadan (ek ve köklerin ayrılmasından) sonra gerçekleşmektedir.

Çeşitli sebeplerle kullanılan parçalama ağaçları basitçe verilen bir dilbilgisine (grammar) göre verilen cümlelerin (veya kelimelerin) nasıl parçalandığını şekilsel olarak gösteren ağaçlardır:

Örneğin aşağıda [BNF yapısında](#) verilmiş dili ele alalım:

$\langle \text{dil} \rangle ::= \langle \text{işlem} \rangle$

$\langle \text{işlem} \rangle ::= \langle \text{işlem} \rangle + \langle \text{terim} \rangle \mid \langle \text{işlem} \rangle - \langle \text{terim} \rangle \mid \langle \text{terim} \rangle$

$\langle \text{terim} \rangle ::= \langle \text{terim} \rangle * \langle \text{unsur} \rangle \mid \langle \text{terim} \rangle / \langle \text{unsur} \rangle \mid \langle \text{unsur} \rangle$

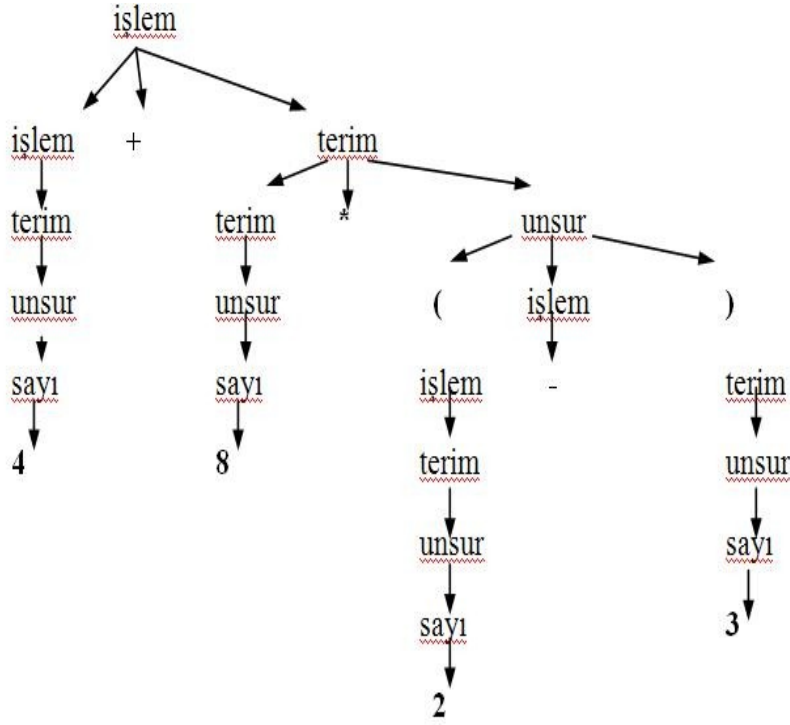
$\langle \text{unsur} \rangle ::= \text{sayı} \mid (\langle \text{işlem} \rangle)$

$\langle \text{sayı} \rangle ::= 1|2|3\dots|9|0$

Bu dilde aşağıdaki örneğin nasıl parçalandığını inceleyelim:

$4+8*(2-3)$

Bu dilde tanımlı olan yukarıdaki işlemin parçalama ağacı aşağıdaki şekildedir:

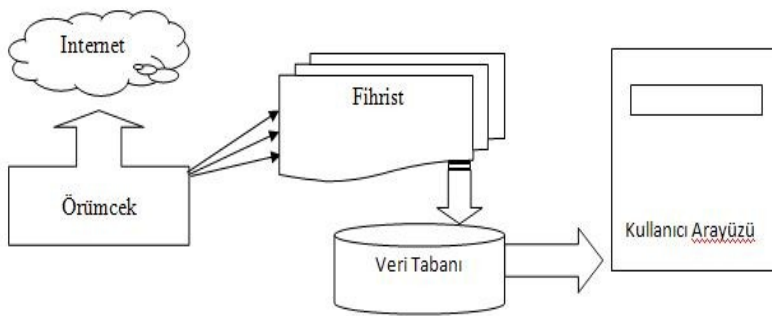


yukarıdaki şekilde devamlı (nonterminal) terimlerden sonuncu (terminal) terimlere kadar bir BNF dilinin nasıl açıldığı gösterilmiştir.

SORU 69: Arama Motoru (Search Engine)

Basit anlamada, Internet üzerinde dolaşarak bulduğu kelimeleri fihristleyen ve daha sonra yapılan aramalarda sorgulanan kelimelere cevap olarak site adreslerini döndüren yazılımlara verilen isimdir.

Bir arama motorunun yapısı aşağıdaki şekilde çizilebilir:



Yukarıdaki şekilde de görüldüğü üzere bir arama motorunda bulunan örümceğin öncelikli görevi internet üzerinde dolaşmak ve bilgi toplamaktır.

Fihrist bu toplanan verileri saklamakta ve ileride yapılacak olan sorgular için en verimli şekilde hazır tutmaktadır.

Mimariye bağlı olarak bir veritabanı sistemde yer alabilir. Şayet veritabanı bulunursa fihristin tuttuğu veriler bu veritabanında saklanabilir.

Kullanıcı arayüzü ise kullanıcıların sorgu yapmaları ve bu sorguları göstermek için kullanılır.

SORU 70: Web Emeklemesi (Web Crawling)

[Arama motorlarının](#) tasarımı sırasında kullanılan önemli unsurlardan birisi de internet üzerindeki bağlantıları (link) izleyerek bilgi toplayan ve bütün Internet'i gezip sayfa ve sayfalardaki kelimeleri çıkaran internet örümcekleridir.

Bir örümceğin bünyesinde web sayfalarını indirip bu sayfadaki bağlantıları çıkaran ve daha sonra bu bağlantılara devam eden yapıya web emekleyicisi adı verilir. Bu isim bebeklerin veya sürüngenlerin kullandıkları ve hedefe ulaşmak için yaptıkları bir takım metodolojik işlem den gelmektedir. Benzer şekilde internet üzerinde emekleyen bir örümcek de metodolojik olarak aşağıdaki işlemleri yapmaktadır:

- Bir internet sayfasını indirmek
- Sayfadaki bağlantıları çıkarmak
- Sayfadaki anahtar kelimeleri çıkarmak
- Kelime ve sayfa bilgisini fihristleyiciye (indexer) geçirmek
- Bulduğu bağlantılara devam ederek yukarıdaki adımları bu yeni bağlantılar için de tekrar etmek

Bu işlemler yapılırken bir web emekleyicisi (web crawler) aşağıdaki hususlara göre sınıflandırılabilir:

Derin emekleme (deep crawling): Kabaca sayfanın ne kadar detayını dolaşip fihristleyeceğimiz. Yani bir web sitesindeki bağlantıları takip edip diğer web sitelerine geçilip geçilmeyeceği.

Çerçeve desteği (Frame Support): sayfalarda bulunan çerçeveleri destekleyip desteklemediğimiz

robots.txt dosyası desteği: web sitelerinin girişinde yer alan bu dosyanın desteklenip desteklenmediği

üst robot etiketi (meta robot tag): web sayfalarının üst bölgesinde (meta tags) yer alan robot etiketinin emekleyici (crawler) tarafından desteklenip desteklenmediği

Tam döküman (full body text): gezilen web sayfasındaki bütün dökümanın mı yoksa bir kısmının mı çıkarılacağı

engellenen kelimeler (stop words): gezilen sayfada bulunan kelimelerden arama için anlamsız olan ve çok yer kaplayan kelimelerin (ve, veya, ile, ise gibi) fihristlenmemesi

Üst tanım (meta description): [HTML sayfalarının](#) üst kısmında (meta tags) bulunan tanım bilgilerinin fihristlenip fihristlenmeyeceği

Üst anahtar kelimeler (meta keywords): HTML sayfalarının üst kısmında (meta tags) bulunan anahtar kelimelerin fihristlenip fihristlenmeyeceği

Bazı Emekleyici Kavramları aşağıda açıklanmıştır:

Odaklanmış Emekleyici (Focused Crawling): Belirli bir hedefin aranmasına yönelik olarak geliştirilen emekleme programlarıdır. Örneğin bir masa üstü arama (desktop search) sırasında, belirli bir kelimenin arandığını düşünelim. Emekleyici yazılımımız, aranan kelimeye yönelik olarak dolaşma yapıyorsa ve örneğin belirli dosya türlerini dolaşıyor, dolaşılan verileri fihristlemek (indexing) yerine sadece kelime arama için çalışıyorsa, odaklanmış emekleme (focused crawling) yapıyor denilebilir.

Dağıtık Emekleme (Distributed Crawling): Bir arama işleminin, birden fazla bilgisayara bölünmesi ve dağıtılmış olarak webte dolaşma işleminin gerçekleşmesi durumudur. Genelde aşağıdaki sebeplerle dağıtık emekleme yapılabilir:

- Veri kaynağına olan yakınlık: Örneğin dünya çapındaki internet emeklemesi için coğrafi sebepler ve her veriye her yerden aynı hızda erişilememesi bir sebeptir. Benzer şekilde bir şirkette veri kaynaklarının emeklenmesi (dolaşılması, crawling) sırasında güvenlik sebepleriyle her bilgiye her yerden erişilememesi bir sebep olabilir.
- Depolama ihtiyaçlarının azaltılması: Birden fazla emekleyici bulunması durumunda, her emekleyicinin tutması gereken site ve sitelere bağlı kaynak miktarının azalması söz konusudur. Verimli bir dağıtık emekleme durumunda, emekleyicilerin birbiri ile senkronize olması ve aynı kaynağa birden fazla emekleyicinin erişmemesi önemlidir.
- İşlem yükünün azaltılması: Birden fazla emekleyicinin bulunduğu ortamda, toplam işlem yükünün, birden fazla bilgisayar bölünmesi mümkündür.

Dağıtık emekleme işlemleri sırasında emekleyicilerin birbiri ile uyumlu çalışması için (senkronize) [özetleme fonksiyonları \(hashing functions\)](#) kullanılabilir. Emekleyiciler aralarında bu özet bilgiyi değiştirerek uyumlu çalışabilirler.

Masa Üstü Emekleyicileri (Desktop Crawlers): Genelde kişisel bilgisayarlarda bulunan dosyaların veya veri kaynaklarını dolaşan emekleyicilerdir. Bazı durumlarda şirket içi veya kurum içi (üniversiteler, hastaneler gibi) dolaşma da planlanabilir. Genel olarak web emekleyicileri ile farkları aşağıda listelenmiştir:

- Verinin bulunması çok daha kolaydır
- Verinin değişme tarihini öğrenmek kısmen mümkündür (dosyaların değişme tarihi)
- Disk ve işlemci kullanımı konusunda daha kısıtlı imkanlar vardır.
- Çok çeşitli dosya tiplerinin tanınması gerekir.
- Verinin kişiselliği önemlidir ve kişisel verilere erişimin kontrol altında tutulması gerekir.

SORU 71: SQL (Structured Query Language, Yapısal Sorgulama Dili)

Yapısal sorgulama dili (SQL) ilişkisel veritabanı yönetim sistemlerinde (relational database management system) kullanılan ve sistem üzerinde programcının istediği veriyi organize etmesi, sorgulaması, ekleme veya değiştirme yapması kısacası veriyi yönetmesi amacına yöneliktir.

SQL kullanımı ve yapısı itibari ile bir komut satır dili olarak da düşünülebilir. Yani kullanıcılar bir komut satırında komutları çalıştırır gibi SQL cümlelerini (statements) girmekte ve sonuçları satır satır görmektedirler.

Günümüzde pekçok sistemin kullandığı SQL standartlarını ISO (International Standards organisation, uluslar arası standartlar enstitüsü) belirlemektedir. Güncel olarak bulunabilecek standartlardan en eskisi 1999 yılında yayınlanmıştır. Bu yıldan önceki standartları destekleyen ortamlar artık yok denecek kadar azdır.

1999 yılından sonra sırasıyla 2003, 2005 ve son olarak 2008 yıllarında standartlarda değişiklikler olmuştur kısaca bu yıllardaki değişikliklerin getirdikleri aşağıdaki şekilde anlatılabilir:

1999 yılında: Regular expression, özyineli sorgular (recursive queries), tetikler (triggers), fonksiyonel programlama (procedural programming) ve çok ilkel anlamda nesne yönelimli programlama özellikleri kazandırılmıştır

2003 yılındaki en önemli katkı XML desteğidir. özellikle nesne yönelimli programlama yapılan ortamlar için büyük bir avantaj sağlamıştır.

2005 yılında XML'in kullanımının artmasıyla birlikte XQuery ismi de verilen ve tamamen veritabanını bir XML kaynağı (örneğin bir dosya gibi) görmeye yarayan sorgulama dili standartlaşmıştır. Veriler arasında klasik veritabanı bağı kurulmasının yanında XML bağlarının kurulması da mümkün hale gelmiştir.

2008 yılında ise ilk defa 2003 yılında kazandırılan pencere fonksiyonları (window functions) netleştirilmiş ve olukça esnek bir yapıya kavuşturulmuştur. (pencere fonksiyonlarını bilmeyenler için kısaca veri öbekleri üzerinde çalışan fonksiyonlar olduğunu söyleyebiliriz. Örneğin hareketli ortalama alma gibi)

SQL dili yapısal olarak 3 ana grupta incelenebilir:

- DML: Data Manipulation Language , Veriyi işlemeye yarayan komutlar topluluğudur (örneğin select komutu)
- DCL: Data Control Language, Veri üzerinde kontrol yapmaya yarayan komutlar topluluğudur. (örneğin grant revoke)
- DDL: Data Definition Language, veriyi tanımlayama yarayan komutlar topluluğudur. (örneğin create, drop, truncate gibi)

Örnek DML (Zeynep Hn. Talebi üzerine)

MS SQL için belirli tarih aralığında bir tabloyu filitrelemek isteyelim. Yani tablodaki sadece belirli tarihe uygun olanlar gelecek. Örnek olarak aşağıdaki tabloyu ele alalım:

KullanıcıID	KonusmaTarihi	Süresi
1	2008-11-08 00:00:00.000	5
2	2009-07-01 11:22:00.000	3
3	2009-07-02 13:00:00.000	11
4	2001-11-08 00:00:00.000	2

Yukarıdaki tablo temsili olarak verilmiştir. Konuşmaya başlangıç tarih / saat ve sürelerinin tutulduğunu düşünelim. Buradaki konuşmaTarihi kolonunun DATETIME tipinde olduğunu kabul ediyoruz. (farklı tiplerde sorun yaşayabilirsiniz bu durumda çevirim yapmanız gerekebilir).

Örnek sorgumuz (query) aşağıdaki şekilde olsun:

```
SELECT * FROM yukaridakiTablo WHERE
KonusmaTarihi >= '2009-07-01'AND
KonusmaTarihi <'2009-07-21'
```

Yukarıdaki sorguda tarih aralığı sorgulanmıştır. Buna göre Temmuz ayının 1 ile 21 (21 dahil değil 1 dahil) günleri arasındaki değerler istenir ve sonuç aşağıdaki şekilde olur:

KullanıcıID	KonusmaTarihi	Süresi
2	2009-07-01 11:22:00.000	3
3	2009-07-02 13:00:00.000	11

Yukarıdaki tabloda kriteri sağlamayan satırlar elenmiş ve istediğimiz sonuç görüntülenmiştir. Ayrıca çok sık yapıldığı için aşağıdaki **HATALI** sql satırlarını da anlatmak istiyorum:

```
SELECT * FROM yukaridakiTablo WHERE
KonusmaTarihi = '2009-07-01'
```

Yukarıdaki bu sorguda hiç bir sonuç dönemz. Bunun sebebi saat kısmı verilmeden sadece tarih ile sorgulanmasıdır. Yani tam olarak tarihin eşit olması için saat kısmının 00:00:00.000 olması gerekir ve bu durumu sağlayan kayıt yoktur.

Benzer şekilde:

```
SELECT * FROM yukaridakiTablo WHERE
KonusmaTarihi = '2008-11-08'
```

Sorgusu verilseydi sadece 1. satır sonuç olarak dönecektir.

Bu tip tarih eşitliği kontrollerinde ogünden büyük eşit ve ertesi günden küçük tarihleri kontrol etmeniz daha doğru olur.

```
SELECT * FROM yukaridakiTablo WHERE
KonusmaTarihi >= '2009-07-01'AND
KonusmaTarihi <'2009-07-02'
```

Şeklinde kontrol edilebilir.

Diğer bir bilinçsiz kullanım da BETWEEN kelimesidir. BETWEEN kelimesi her iki tarihi de dahil eder:

```
SELECT * FROM yukaridakiTablo WHERE
KonusmaTarihi BETWEEN '2009-07-01'AND '2009-07-02'
```

şeklinde girilen bir sorgu için hem 01 temmuz hem de 02 temmuzdaki bütün konuşmalar görüntülenecektir.

SORU 72: Kabuk (Shell)

Bilgisayar bilimlerinde kabuk kelimesi daha çok çevreleyici, kaplayıcı anlamında kullanılmaktadır. Örneğin işletim sistemlerinde çekirdeğin (kernel) dış dünya ve kullanıcılar ile iletişim kurmasını sağlayan işletim sisteminin parçasına kabuk (shell) adı verilmektedir.

İşletim Sistemlerinde Kabuk

Kabuğun en temel görevini bir komut satırı (command line interface) olarak tanımlayabiliriz. Örneğin DOS, LINUX veya UNIX işletim sistemlerinde komutları alan ve bu komutlar dahilinde çekirdeğe işlemleri geçiren modül olarak düşünülebilir.

Örneğin basit bir taşıma işlemi:

```
mv a.txt b.txt
```

komuduyla yapılabilir. Bu dosya taşıma işlemi sırasında dosya sisteminde bazı bilgilerin değiştirilmesi söz konusudur (örneğin FAT Tablosundaki kayıt ya da inode değerleri gibi) bu değişiklikler çekirdek tarafından yapılır.

komut satırının daha gelişmiş olarak kabul edilebilen Grafik Arayüzü (Graphical User Interface, GUI) eklentileri ile kabuğun yaptığı işlerde pek farklılık olmasa da kullanıcıya sunulan işlemlerin şekli ve kullanıcının işlem yapabilme kabiliyeti arttırılmıştır. Ancak temelde bir işletim sisteminin çekirdeği ile kabuğun ilişkisi aynıdır.

Kabuk programlama (shell programming) ismi verilen bir programla tipi de kullanıcı işlemleri yapılan bu kabuktaki işlemleri bir program dahilinde kullanıcı iletişimi olmaksızın çalıştırmayı amaçlar.

DOS üzerindeki bat dosyaları (Batch files) ve linux ve unix üzerindeki shell programming ve windows üzerindeki vbscript ve javascript (csh ve jsh) bu tip kabuk programlamaya örneklerdir. Burada kabuğun yaptığı işlemler kullanılarak yine kabuğun yaptığı birleştirilmiş işlemler elde edilir. Örneğin sistemdeki her kullanıcının dizinlerinin içerisinde bir adet yardım dosyası koymak isteyelim. bunu yapan hazır bir komut yoktur. Ancak bir dosyayı kopyalayan komudumuz 'cp' her kullanıcı için tekrar tekrar çalıştırılarak bu işlem yapılabilir. İşte bir sistem yöneticisi örneğin 100 kullanıcı için teker teker bu kopyalama işlemini yapmak yerine bir kabuk programı yazarak bu işlemi yaptırmaktadır.

SORU 73: Çekirdek (Kernel)

Bilgisayar bilimlerinin farklı alanlarında kullanılmasına karşılık, çekirdek kavramı genelde birşeyin merkezi veya kalbi şeklinde tabir edilebilecek anlamlara gelmektedir.

İşletim sistemlerinde çekirdek:

İşletim sisteminin temel fonksiyonlarının icra edildiği kısımdır. Kullanıcılar ile iletişim kuran kabuk (shell) sadece dış işleri yapmaktan sorumlu olup, işletim sisteminin bütün temel fonksiyonları çekirden üzerinde çözülür.

Bir çekirdeğin temel görevleri aşağıdaki şekilde sıralanabilir

- Giriş çıkış işlemlerinin yönetilmesi (I/O management): Örneğin klavye, fare veya ekran gibi dış donanımların yönetilmesi bu donanımların hafıza ve işlem ihtiyaçlarının sistem kaynakları içerisinde çözülerek tasarlanan zaman ve tasarlanan başarıyla çalışmalarını sağlamaktır.
- İşlem yönetimi (process management): Bir işletim sisteminde çalışan programların ve programların ürettiği işlemlerin (process) yönetilmesi işidir. Bilindiği üzere her işlemin sistemden sürekli olarak talepleri olmaktadır. Bu taleplerin karşılanması ve işlemlerin belirli bir ahenk ve adil bir sıra ile çalışmasını sağlamak gibi görevler işletim sisteminin çekirdeği tarafından yürütülür.
- Hafıza yönetimi (memory management): İşletim sisteminin çekirdeği, kendisi de dahil olmak üzere, o anda çalışan bütün işlemlerin hafıza gereksinimini, en verimli şekilde karşılamak zorundadır. Bunun için sayfalama (paging) ve kıtalama (Segmentation) işlemlerinin yapılması.
- Aygıt yönetimi: Sisteme bağlı çalışan aygıtların kontrolü, bu aygıtların işlemci ve hafıza ihtiyaçlarının karşılanması ve işletim sisteminin diğer parçalarının bu aygıtlara erişimi.
- Dosya yönetimi: Disk üzerinde tutulan dosyaların takibi, bu dosyaların disk üzerinde verimli bir şekilde tutulması, hızlı erişilmesi, güvenliğinin sağlanması ve dosyalama ile ilgili kopyalama, taşıma, okuma, yazma gibi işlemlerin icrası.

Yukarıdaki temel işlemler bir işletim sisteminin çekirdeğini oluştururken her zaman bulunması gereken durumlar değildir. Örneğin bazı işletim sistemleri disk bile olmayan ortamda çalışmaktadır (günümüzdeki cep telefonu ve kişisel asistanlar (PDA, personal digital assistant) bunlara birer örnek olabilir) bu durumda doğal olarak bir disk kontrolünden bahsetmek mümkün değildir.

SORU 74: Dahili Parçalar (Internal Fragments)

Birden fazla işlemin bir işletim sistemi üzerinde çalıştırılması sırasında hafızadaki işlemlerini belirli bir düzene göre yerleştirilmesi gerekir.

Bu yerleştirme sırasında çıkan problemlerden birisi de parçalar (fragments) 'dir. Buna göre işletim sisteminin önünde iki ihtimal bulunmaktadır. Ya hafızayı sabit slotlara bölüp işlemlere bu slotlardan ihtiyaç duyduğu kadar verecektir (örneğin slotların boyu 200birimlik olsun, işlemlerden birisi 700 birim isterse 4 slot bu işleme verilecek, sonuç olarak $4 \times 200 = 800$ birim ayrılmış olacak ve 100 birim hafıza israf olacaktır , bu yaklaşıma sayfalama (paging) ismi verilir) ya da her işleme ihtiyaç duyduğu kadar hafıza alanı ayrılacaktır bu durumda da işlemler arasında boşluklar oluşacaktır. Bu boşluklara da harici parça (external fragment) ismi verilir.

Aşağıda sayfalama (paging) sırasında oluşan bir dahili boşluk (iç boşluk, internal fragment) tasvir edilmiştir:

Sayfa 1 (Page 1)	İşlem 1 (Process 1)
Sayfa 2 (Page 2)	İşlem 1 (Process 1)
Sayfa 3 (Page 3)	İşlem 1 (Process 1)
Sayfa 4 (Page 4)	İşlem 1 (Process 1)
Sayfa 5 (Page 5)	İşlem 2 (Process 2)

Yukarıdaki örnekte de görüldüğü üzere 4. sayfada işlem 1'in tam kullanmaması dolayısıyla bir boşluk oluşmuştur. Bu boşluk başka işlemler tarafından asla kullanılamaz çünkü bu sayfa (page) artık işlem 1 için ayrılmıştır.

Örneğin 200 birimlik sayfa boyutuna sahip bir sayfalama da 700 birim kullanan bir işlem yukarıdaki şekilde görüldüğü gibi 3 tam sayfa bir de yarım sayfa kullanacaktır. İşte bu yarım sayfada oluşan boşluğa dahili boşluk (internal fragment) ismi verilir.

SORU 75: Kıtalamak (Bölütlemek, Segmentation)

İşletim sistemlerinin temel görevlerinden birisi olan hafıza yönetimi (memory management) için kullandıkları çözüm yöntemlerinden birisidir. Bilindiği üzere bir işletim sistemi birden fazla işlem (multi process) çalıştırıyorsa bu durumda işletim sisteminin hafızayı bu işlemler arasında ihtiyaçlarına uygun bir şekilde dağıtması gerekir.

Bu dağıtımda bir işleme tam olarak ihtiyacı olduğu kadar yer ayırma yaklaşımına kıtalama (bölütlemek , segmantasyon , segmentation) ismi verilir.

yaklaşım basitçe aşağıdaki şekilde gösterildiği gibidir:

100	1. İşlem (Process 1)
400	2. İşlem (Process 2)
500	3. İşlem (Process 2)
800	

Yukarıda, hafızada (RAM) çalışmakta olan 3 ayrı işlem ve bu işlemlerin hafıza adresleri görülmektedir. Dolayısıyla bir işlemin fiziksel adresi kendi içerisindeki mantıksal adresten farklı olmaktadır.

Yani her işlem kendi adresleme sistemini kullanır ve hafızada nerede çalıştığından habersizdir. 1. İşlem için 50 numaralı adres, 2. işlem için 50 numaralı adres ve 3. işlem için 50 numaralı adres hep kendi 50 numaralı adresleridir. Basit bir hesaplama ile bu adresler 1. işlem için 150, 2. işlem için 450 ve 3. işlem için 550 numaralı gerçek adreslerken, işlemler bundan habersiz olarak kendi 50 numaralı adreslerine erişmeye çalışırlar.

İşlemlerin talep ettiği adresler (demanding addresses) kendi adresleridir ve bu adreslere mantıksal adres (logical addresses) ismi verilir. Gerçekte bilginin bulunduğu adresler ise fiziksel adreslerdir (physical addresses). Bu iki bilgi arasında dönüşüm aşağıdaki formülle yapılabilir:

fiziksel adres = kitanın başlangıcı + mantıksal adres

mantıksal adres = fiziksel adres – kitanın başlangıcı

Yukarıdaki bu formüllere göre iki adres arasındaki dönüşüm kitanın (segment) hafızadaki adresi kadardır. Bu adresler kıta tablosu (segment table) ismi verilen bir tabloda aşağıdaki şekilde tutulur:

Limit	Başlangıç (Base)
300	100
100	400
300	500

Yukarıdaki sistemde 3 kıta (segment) olan bir sistem söz konusudur. Burada kıta numaraları verilmemiştir bunun sebebi yukarıdaki sıra ile birer kıta numarası olmasıdır. Örneğin 2 numaralı segment 300 limit ve 500 başlangıç değerine sahiptir.

Bu tabloda yer alan başlangıç değeri kitanızın hafızaya yüklendiği adres olurken limit değeri de hafızada ne kadar yer kapladığını göstermektedir.

Limit değerinin tutulmasının sebebi bir işlemin başka işlemlerin adreslerine erişmesini engellemektir. Dolayısıyla her erişim aşağıdaki kontrolden geçer:

başlangıç + mantıksal adres < başlangıç + limit

şayet bu sorunun cevabı hayır ise bu durumda ilgili kitanın (segment) dışında bir alana erişilmeye çalışılıyor demektir ve hata verilir.

kıtalama işlemi sırasında adresleme kıta numarası ve mantıksal adres ikilisinden oluşmaktadır.

(s,d) ikilisi (segment, offset) (kıta, mantıksal adres) ikilisi anlamındadır.

Örneğin yukarıdaki kıta tablosunda (1,40) gerçek adres olarak (Fiziksel adres, physical address) 440 olmaktadır (1,120) bir hatadır.

SORU 76: Harici Parçalar (External Fragments)

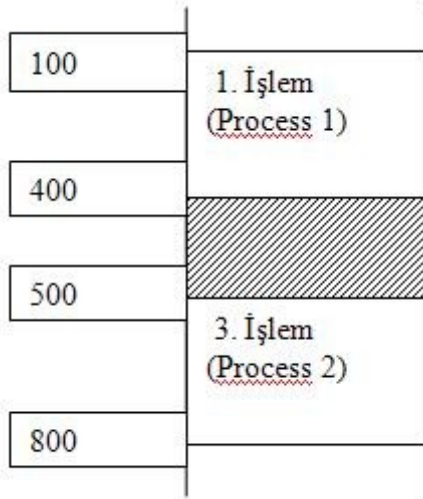
Hafıza yönetimi sırasında kullanılan kitalama (bölütleme, segmentation) hafızadaki her işleme tam olarak istediği kadar yer ayırmaya çalışır. Bu yaklaşımda, işlemler arasında oluşabilecek boşluklara verilen isim harici parçalar (dış parçalar, external fragments)'dir.

Her işleme ihtiyaç duyduğu kadar yer ayırmak ilk başta daha verimli gibi görölse de bu çözümde de boşluklar ve hafıza israfı söz konusudur. Sayflama (Paging) yaklaşımında slotların içine olan boşluklara iç parça (internal fragment) ismi verilirken kitalama (segmentation) sırasında oluşan boşluklara dış parça (external fragment) ismi verilir. Bu boşluklar aşağıdaki şekilde oluşur.

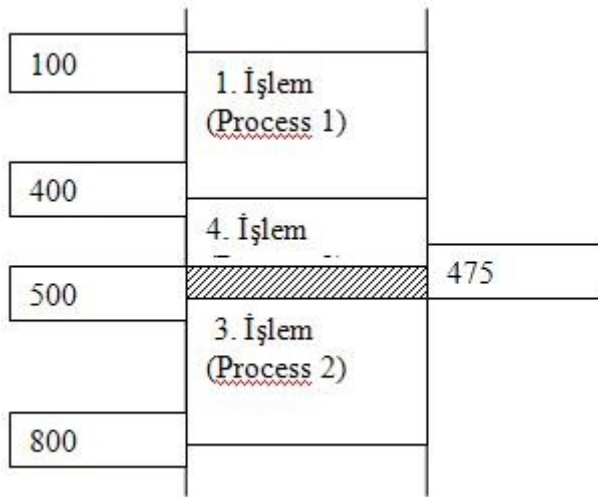
100	1. İşlem (Process 1)
400	2. İşlem (Process 2)
500	3. İşlem (Process 2)
800	

Yukarıdaki şekilde görüldüğü üzere sırasıyla 1, 2 ve 3. işlemler gelmiş hafızada gösterilen alanlar bu işlemlere ayrılmıştır. Şimdi bu işlemlerden örneğin 2. işlemin çalışıp bittiğini düşünelim. Bunun anlamı 2. işlem hafızadan kaldırılacak demektir.

2. işlem hafızadan kaldırıldıktan sonra oluşan tablo aşağıdaki şekildedir:



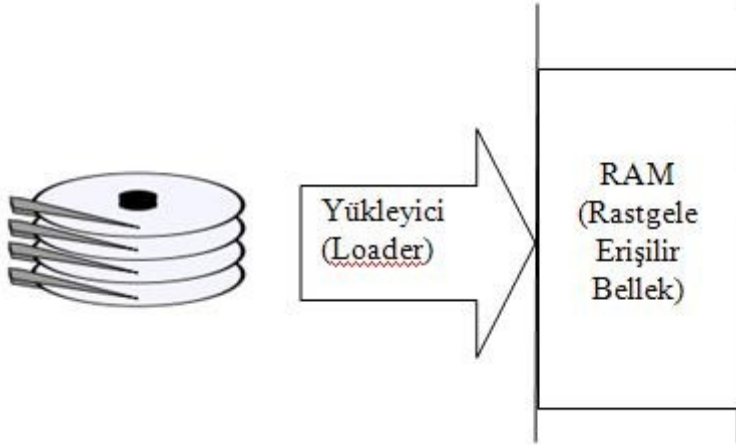
Yukarıda da görüldüğü üzere 400 ile 500 arasındaki hafıza alanı boşalmıştır. Buradaki boşluk verimsiz bir boşluktur bunun sebebi 100 boyutunda başka bir işlem gelmediği sürece bu alan kullanılamaz. Hele bir de aşağıdaki gibi örneğin 75 boyutunda bir işlem gelirse arada kalan 25 birimlik boşluğun kullanılabilirliği oldukça azalır.



işte bu alanlara bilgisayar bilimlerinde verilen isim harici parça (external fragment)'dir.

SORU 77: Yükleyici (Loader)

Yükleyiciler basitçe bir programı diskten alıp hafızaya yüklemekle sorumlu programlardır.



Bir program yazıldıktan ve derlendikten (compile) sonra programın makine dilindeki karşılığı elde edilir. Bu karşılık tam bir kod olmayıp harici kütüphanelerden faydalanyor olabilir. Bu kütüphaneler de programa dahil edilip tam bir program elde edildikten sonra (yani bağlandıktan sonra (linker)) program artık çalıştırılmaya hazırdır.

Programın çalışması ise programın CPU üzerinde yürütülmesi ile olabilir ve bunun için programın öncelikle hafızaya (RAM) yüklenmesi gerekir. Burada yükleyici (loader) devreye girer. Yükleyici makine dilindeki bu kodu alarak işletim sisteminin işaret ettiği adrese programı yükler. Buradan sonrası işletim sistemi tarafından yürütülür.

SORU 78: Hafıza Yönetimi (Memory Management)

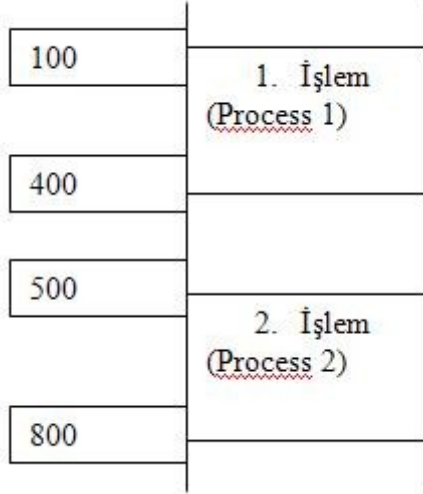
Bir işletim sisteminin (operating system) birden fazla işlem çalıştırması durumunda (multi process) bu işlemlerin hafızayı nasıl paylaşacakları ve hafızanın nasıl daha verimli kullanacağı hafıza yönetiminin konusudur.

Şayet işletim sisteminde tek işlem (process) çalışıyorsa bu çözülmesi çok daha kolay bir durumdur. İşlemler derlendikten (compile) sonra hafızaya yükleyici (loader) ismi verilen bir programla yüklenir ve işlemci (CPU) bu işlemi çalıştırmaya başlar. İşlem bitene kadar bilgisayar bu işlemin kontrolünde olur ve ancak işlem tamamen bittikten sonra hafızadan kaldırılarak yerine yenisi konulabilir ve farklı bir işlem çalıştırılabilir.

Ancak bir işletim sistemi birden fazla işlemi çalıştırmak gerekiyorsa burada bir iki problem ortaya çıkar.

İşlemlerin hafıza adresleri: Çalışan işlemler programlanır veya derlenirken hafızayla ilgili işlemleri gereği adres bilgileri içerirler. Örneğin bir program hafızadaki sabit bir adrese (sözgelimi 150 numaralı adres olsun) erişmek isteyebilir.

Bilindiği üzere hafızada aynı adrese sahip tek yer bulunmaktadır. Dolayısıyla örneğin 150 numaralı adrese erişme talebi tek bir yerdir oysaki programların adres erişme talepleri genelde yine programın farklı bir alanına erişmek içindir. Buradaki problem hafızada birden fazla işlemin yüklenmesi durumunda aynı adrese sadece bir işlemin yüklenecek olması ve her işlemin farklı adreslerde çalışacak olmasıdır. Dolayısıyla hafızadaki adresler hareketli olacak (dynamic, dynamic addressing) ve her işlem artık farklı bir adres aralığına yüklenecektir.



Örneğin yukarıdaki şeklin hafızayı (RAM) temsil ettiğini farz edelim. Bu resimde de görüldüğü üzere iki işlem aynı anda hafızaya yüklenmiş bu işlemlerden ilki 100-400 aralığında yüklü ikincisi ise 500-800 aralığında yüklüdür.

Örneğimize geri dönecek olursak 150 numaralı hafıza adresi 1. işlemin yüklü olduğu adrestir. 2. işlemin buraya erişmesi ise risklidir. Bunun çok basit iki temel sebebi vardır.

1. Güvenlik ihlali olabilir. Yani örneğin bu adreste bir kullanıcının şifresi yazılı olabilir ve başka bir işlem buraya erişirse (ki bu başka bir işlem başka birisinin işlemi de olabilir) bu şifreyi ele geçirebilir
2. çalışan işlemlerin birbirine müdahalesinin sonucu kestirilemez. Yani siz işletim sisteminde bir kelime işlem programı çalıştırdığınızı düşünün (örneğin word) herhangi birisi girip sizin wordde yazıklarınızı daha kaydetmeden ekranda yazılıyken değiştirebilir, sonuçlar her zaman bu örnek kadar masum da olmak zorunda değildir. Pek çok ihtimalde sistemin çökmesi veya kalıcı olarak zarar görmesi mümkündür.

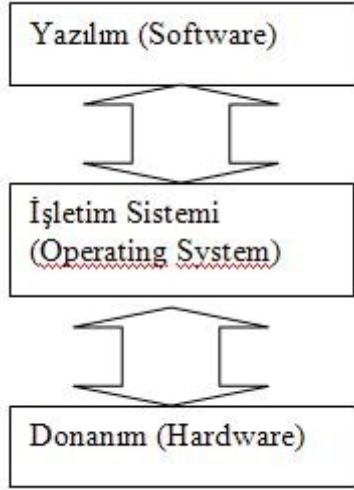
İşte hafıza yönetimi kapsamında

- bir işlemin talep ettiği (demanding address) ile RAM'in gerçek adresi (fiziksel adres, physical address) arasındaki bağı kurulması (ki bu işlem için mantıksal adres (Logical address) ismi verilen bir yöntem kullanılmaktadır.)
- Aynı anda çalışan birden fazla programın hafızayı daha verimli kullanması için hafızada oluşan boşlukların (fragmentation) azaltılması.
- [İşlemler arası haberleşme ve senkronizasyon](#) ihtiyaçları giderilerek her işlemin kendi ardes alanında (own address space) çalışmasının garanti edilmesi

gibi problemler çözülmektedir. Bu problemlerin çözülmesi için [sayfalama \(Paging\)](#) ve [kıtalama \(bölütleme, segmentation\)](#) ismi verilen yöntemler kullanılmaktadır.

SORU 79: İşletim Sistemi (Operating System)

İşletim sisteminin görevi temel olarak donanım (ve diğer sistem kaynakları) ile bilgisayarda çalışan ve bu kaynakları talep eden program (veya processler) arasında ilişki kurmak ve kaynak yönetimini kontrol etmektir. Aşağıdakine benzer bir katmanlı yaklaşım bu anlamda doğru kabul edilebilir:



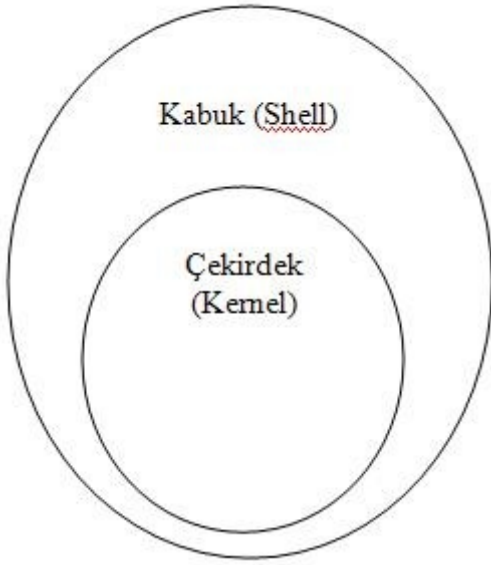
İşletim sisteminin günümüzdeki anlamını anlamak için belkide kelimenin gelişimini ve kısa bir tarihini bilmekte yarar olabilir. İşletim sistemleri olmadığı zamanlarda bilgisayarların yönetilmesinden sorumlu olan işletmenler (operators) bugün işletim sistemlerinin yaptığı işlemleri temel olarak elle yapıyorlardı.

Örneğin kullanıcıların çalıştırmak istedikleri programları varsa, bu işletmene gidip programlarını teslim ediyor sonra işletmenin verdiği tarih ve saatte tekrar giderek programlarının sonuçlarını alıyorlardır. Bu sırada işletmen bilgisayar üzerinde gerekli kaynak ayarlamaları ve sistem ayarlarını (configuration) yaparak programları belirli bir sırada çalıştırıyordu.

Gelişen yazılımlarla işletmenlerin bu yaptıklarını artık programlar yapabiliyor ve gelen program talebini yukarıdakine benzer şekilde çözüyorlar.

İşletim sistemlerinin bugün kü halini almasındaki en önemli tarihi kırılmalardan birisi de bir işletim sisteminde aynı anda birden fazla işlemin çalışabilmesidir (multi processing). Bu gelişme ile birlikte hafızanın ve işlemcinin paylaşılması problemi doğmuştur.

Bir işletim sistemini iki farklı parça olan çekirdek (kernel) ve kabuk (shell) 'den müteşekkil olarak görmek mümkündür. Buna göre çekirdek kısmında işletim sisteminin temel fonksiyonları icra edilirken, kabuk kısmı işletim sisteminin kullanıcı iletişiminden, kullanıcılardan gelen taleplerin çekirdeğe taşınması ve sonuçların kullanıcının talebi doğrultusunda işlenmesi gibi işlemlerden mesuldür.



İşletim sistemleri ile ilgili bilgisayar kavramları.com sitesinde yer alan pek çok yazıyı yine aynı isimli kategorinin altından okuyabilirsiniz.

SORU 80: Özellik Çıkarımı (Feature Extraction)

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 81: 2 geçişli çeviriciler (2 pass assemblers)

Bir çeviricinin (assembler). Assembly dilinde yazılmış kaynak kod üzerinden iki kere geçen halidir. Buna göre tek geçişli olan çeviricilerden farklı olarak dilde etiket (label) tanımları yapılabilmekte ve bu etiketlerin anlamları bir tabloda tutularak daha sonra kullanılabilir.

2 geçişli bir çevirici sırasıyla şu işlemleri yapar:

1. Geçişte koddaki semboller (symbols) ve ifadeler (literals) tanımlanarak hafızada saklanır
2. Geçişte hedef kod (makine dilinde) üretilir

Bu sırada kullanılan veri yapıları (data structures) aşağıda açıklanmıştır:

Konum Sayacı (Location Counter (LC)): Programın mevcut durumunu ve bundan sonra çalıştıracağı kodu tutar.

İşlem Kodu Tercüme Tablosu (Op-code translation table): Kaynak kodda bulunan komutların ve dile özgü işlemlerin çevirimlerini tutan tablodur.

Dizgi saklama belleği (String Storage Buffer (SSB)): Kaynak kodda geçen dizgilerin (kelime ve yazıların (strings)) ASCII kod karşılıklarının durduğu tablodur.

İşaret tablosu (Configuration Table): Dizgi saklama belleğinde saklanan dizgilerin üretilecek olan kodda tam olarak nereye tekabül ettiğini tutan tablodur. (bir dizginin (string) anlamı belirlenince bu tablo marifetiyle üretilecek koddaki yeri bulunarak buraya yazılacaktır)

Yukarıdaki şekilde iki geçişli bir çeviricinin (two pass assembler) her iki aşaması tasvir edilmiştir. Buna göre assembly dilindeki kaynak kodun üzerinden bir kere geçen çevirici (Assembler) dildeki sembolleri ve dizgileri (Strings) algılayarak geçici bir veri yapısında tutar. İkinci geçişte ise bu veri yapılarını da kullanarak makine dilini üretir.

Aşağıda örnek bir kod ve bu kodun çalışması anlatılmıştır:

assembly dilinde kaynak kod	hafızadaki adresi	Hafızadaki makine kodu
START 0100H		
LDA #0	0100	01
	0101	00
	0102	00
LDX #0	0103	05
	0104	00
	0105	00
LOOP: ADD LIST, X	0106	18
	0107	<u>01</u>
	0108	<u>12</u>
	0109	2C
TIX COUNT	010A	<u>01</u>
	010B	<u>15</u>
	010C	38
JLT LOOP	010D	<u>01</u>
	010E	<u>06</u>
	010F	4C
RSUB	0110	00
	0111	00
LIST: WORD 200	0112	00
	0113	02
	0114	00
COUNT: WORD 6	0115	00
	0116	00
	0117	06
END		

2. geçişte
yerleştirilmiştir

2. geçişte
yerleştirilmiştir

2. geçişte
yerleştirilmiştir

Yukarıdaki temsili kod'un çalıştırılması sırasında oluşturulan tablolar aşağıda verilmiştir:

Sembol Tablosu

<i>Symbol</i>	<i>Address</i>
LOOP	0106
LIST	0112
COUNT	0115

Yukarıdaki tabloda, kodda bulunan her sembol için (dilde tanımsız kelimeler için) birer kayıt tutulması ve bu kelimelerin geçtiği adreslerin tutulmasıdır.

İşaret Tablosu:

<i>Adresi (offset)</i>	<i>Sembolun SSB işaretçisi</i>
0007	DC00
000A	DC05

İşaret tablosunda (configuration table) da kodumuzda karşılığı belirsiz olan sembollerin adresleri bulunur. Örneğin LIST kelimesi SSB içerisinde DC00 adresindedir. LIST kelimesinin işaret tablosuna yerleştirilme sebebi ise henüz tanımlanmadan kodun içerisinde kullanılmış olmasıdır.

SSB:

DC00	4CH	L
DC01	49H	I
DC02	53H	S
DC03	54H	T
DC04	5EH	Ayırma sembolü ASCII olarak
DC05		C

SORU 82: Şelale Modeli (Waterfall Model)

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

- tahlil (analiz, analysis)
- tasmim (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 tasmim (tasarım) yapılır ve çözüm yolları ortaya konulur. Tasmim aşaması sonunda da SDD (Software design document, tasmimname, 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 dönülür ve hatalar telafi edilerek sistem istenen hale getirilir.

SORU 83: Tasarım Kalıpları (Tasmim Kalıpları, Design Patterns)

[Yazılım mühendisliğinde](#) sıkça kullanılan tasarım kalıplarının bir kütüphane haline getirilmesi ve bu kütüphanenin ileriki projelerde kullanılmasıyla proje geliştirme sürelerinin kısaltılması hedeflenmektedir. Bu kütüphane (design patterns library) genelde farklı yapılar içerebilmesine karşılık aşağıdaki gruplarda toparlanabilir:

- Web tasarım kalıpları (web design patterns)
- UML tasarım kalıpları (UML Design patterns)
- Grafik arayüzü tasarım kalıpları (GUI design patterns)

Web tasarımı ile ilgili pek çok kalıp kütüphanesi internet üzerinden ulaşılabilir durumdadır. Özellikle AJAX teknolojisinin hızla gelişmesi sonrasında internet üzerinde yayın yapan pek çok gelişmiş kurum (yahoo, google gibi) kendi kütüphanelerini geliştirerek internet kullanıcılarına açmışlardır.

UML tasarım kalıplarının amacı ise özellikle nesne yönelimli programlama yapılan ortamlarda belirli şablonların saklanarak ileriki projelerin içerisinde kullanılmasıdır. Bu amaçla çeşitli sınıflandırmalara gidilmiştir. Örneğin yapısal tasarımların yer aldığı ve daha çok bileşenlerin konumlandırılmalarını belirleyen mimari tasarım kütüphanesi (architectural design library) veya yazılım tasarımında kullanılan sınıf diyagramlarının (class diagrams) içerildiği GOF bunlardandır.

Burada GOF'tan bahsetmekte yarar var. GOF, gango of four (dört gangster diye çevrilebilir)'un kısaltmasıdır. Kısaca yazılım mühendisliği konusunda meşhur kitaplardan "Design Patterns: Elements of Reusable Object-Oriented Software (ISBN 0-201-63361-2)" kitabının yazarı olan 4 kişiye (Erich Gamma, Richard Helm, Ralph Johnson ve John Vlissides) ithafen bu isim verilmiştir. Bu kitapta da anlatılan ve bu kişilerin geliştirdikleri tasarım yöntemine göre 3 farklı grup tasarım sınıfından bahsedilebilir, bu sınıfları ve geliştirme aşamalarını içeren tablo aşağıda verilmiştir:

	Creational (Oluşturucu)	Structural (Yapısal)	Behavioral (Davranışsal)
Interface (arayüz)		AdapterBridge Composite Facade	
Responsibility (Sorumluluk)	Singleton	FlyweightProxy	Chain of ResponsibilityMediator Observer
Construction (oluşum)	Abstract FactoryBuilder Factory Model Prototype		Memento
Operation (İşlem)			CommandInterpreter State Strategy Template Method
Extension (Uzatma)		Decorator	IteratorVisitor

Yukarıdaki tabloda GOF içerisinde yer alan 3 önemli geliştirme tasarımına göre sınıflandırma yapılmıştır.

Ayrıca yukarıda bulunmayan ve muvazi fiiller (eşzamanlı işler, concurrent processes) konusunu içeren tasarım kalıpları da bulunmaktadır.

SORU 84: RMI (Remote Method Invocation, Uzaktan Metod Çağırma)

Dağıtık programlamanın bir parçası olan RPC(Remote Procedure Call) üzerine inşa edilmiş olan ve nesne yönelimli programlama akımıyla birlikte gelişmiş olan RMI yaklaşımı basitçe bir kodun bir parçası olan methodun başka bir kod tarafından çağrılmasını sağlar.

Temel olarak sistemde rmiregistry ismi verilen bir kayıt üzerinde RMI için çağrılması uygun methodların bir kaydı tutulur. Bu methodlardan birisini çağırarak isteyen sınıf (class) ise kayıt üzerinden ilgili methodu çağırıp sonucu bir nesne olarak kendisine alır.

Dağıtık Nesne Mimarisi adı da verilen bu mimaride iki taraf bulunur istemci (client) ve sunucu (server). Klasik bir RMI modelinde sunucu kendi üzerinde objeler oluşturur ve bu objelerin üzerindeki metodların çağrılması (invoke) için bekler. İstemci ise uzaktaki bu nesnelere birer referans oluşturur ve bu referans marifetiyle nesneler üzerindeki methodları çağırır. Bu işlemi sırasıyla 3 aşamada ele almak mümkündür:

- Uzaktaki nesneye erişilmesi: Bu aşamada uzaktaki nesnenin konumunun belirlenmesi ve erişmek için gerekli adımların tamamlanması yapılır. Bu işi yapmanın pek çok farklı yolu olmasına karşılık en temel ve hızlı yollarından birisi RMI isim hizmeti olan RMIREgistry üzerinde bir kayıt marifetiyle ulaşmaktır.
- Uzaktaki nesne ile iletişim: Bağlantı kurulduktan sonra bu [nesne](#) üzerinde bir fonksiyonun çağrılması, bu fonksiyona parametre geçilmesi veya sonucunun alınması gibi iletişim işlemleri yapılır. Ancak programcı bu işlemlerin hiçbirinden haberdar olmaz. Sanki yerel bir nesneyi çağırıyor gibi kodunu yazar ve çalıştırır.
- Uzaktaki nesnenin, yerel bir sınıf tanımının oluşturulması. Bu aşama ise yerel nesnenin uzaktaki nesneyi tanıması için gerekli olan aşamadır. Yani yereldeki nesneler uzakta bulunan nesnenin hangi fonksiyonlarını çağırabileceklerini bilmelidirler. Bu yüzden yerelde bir kopya sınıf bilgisi tutulur.

SORU 85: Kütük (stub, nesne vekili, object Proxy)

Nesne yönelimli programlamanın gelişmesiyle birlikte dağıtık nesne mimarisi denilen bir kavram gün yüzüne gelmiştir. Bu kavrama göre bir nesne yönelimli projeyi farklı bilgisayarlar üzerinde dağıtmak ve projenin bu farklı bilgisayarlar üzerindeki farklı nesneler ile iletişim kurmasını sağlamak mümkündür.

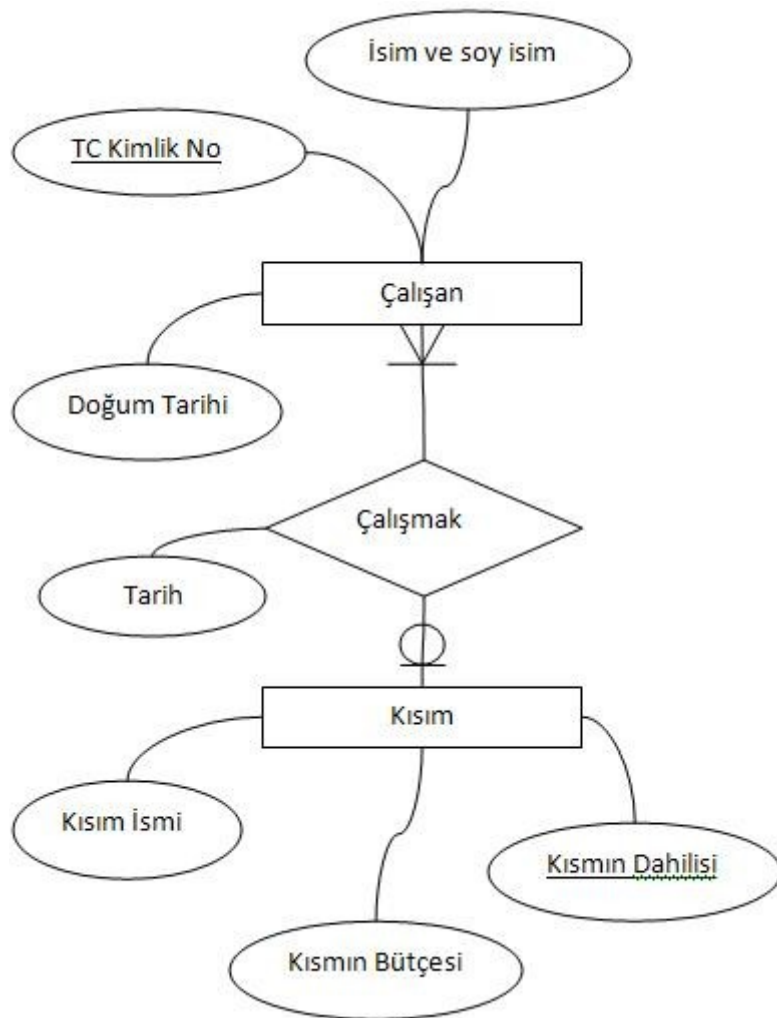
Bu işlem için örneğin JAVA dilindeki RMI (uzaktan metod çağırma , remote method invocation) özelliğini kullanmak istediğimizi kabul edelim. RMI yapısında yerel nesnelerin (istemciye nesnelerin (client objects)) uzaktaki nesneleri (remote objects, sunucudaki nesneler) tanıyıp çağırabilmeleri için uzaktaki nesnelerin de yerelde birer temsili olması gerekir. Bu temsil işlemini yerine getiren ve JVM'ler (Java Virtual Machine) arası nesne transferini yerine getiren yereldeki gölge nesnelere kütük (Stub) veya nesne vekili (object Proxy) isimleri verilir.

Buradaki amaç programcının çağırdığı nesneleri yereldeymiş gibi çalıştırıyor olması ve sonuçlarını yerel bir nesneymiş gibi alıyor olmasıdır. Ancak işin aslında nesne sadece bir

görüntüden ibaret olup yaptığı iş gelen çağırma taleplerini (invokation) uzaktaki nesneye iletmek ve uzaktaki nesnenin sonucunu yereldeki çağıran nesneye geri iletmeektir.

SORU 86: Vücubiyet (Modality)

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 vâic 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 vücubiyet açısından vacib-ihtiyari (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 87: Sayısallık (Cardinality)

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 çalışanın cep telefonu numarası ile TC kimlik no arasında birebir ilişki vardır. Yani her farklı TC kimlik no için farklı bir cep telefonu numarası söz konusudur. Bu ilişki tipi bire bir ilişkidir.
- Bir çalışan ile, çalıştığı kısım arasında bire çok ilişki vardır. Yani bir kısımda (department) birden çok çalışan çalışırken, her çalışan sadece bir kısma bağlıdır. Bu ilişki tipi de bire çok ilişkiye örnektir.
- Bir öğrenci unsuru (entity) ile hoca unsuru (entity) arasında ise çok çok ilişki tipi vardır. Çünkü bir öğrencinin birden fazla hocası olabilirken bir hocanın da birden fazla öğrencisi bulunur.

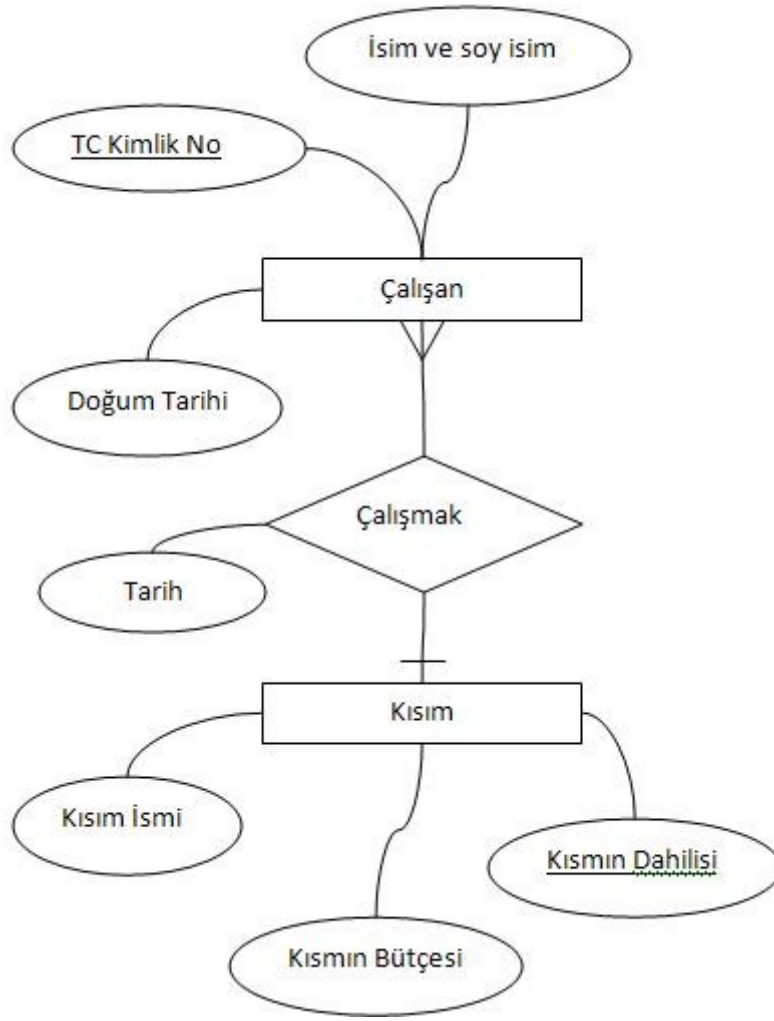
Yukarıdaki bu ilişki tiplerinin hepsinin veritabanı teorisi için anlamları çok büyüktür. İlişki türlerine göre kabaca aşağıdaki yorulmar yapılabilir. (Bunlar genel yorumlar olup istisnaları bulunmaktadır, buradaki amaç okuyucuya fikir vermektir)

Şayet iki unsur (entity) arasında birebir ilişki varsa bu iki unsurun aslında ayrılmasına gerek yoktur. Çok büyük ihtimalle bir unsurun iki farklı parçalarıdır ve tek bir çatı altında birleştirilmesi veritabanı teorisi açısından daha doğrudur.

Şayet iki unsur arasında çok çok ilişki tipi varsa o halde bu ilişki tipi bire çok tipinden iki ilişkiye indirgenmelidir.

Sonuç olarak veritabanında sadece teke çok ilişki tipi elde etmek isteriz bunun sebebi yukarıda da anlatıldığı üzere birebir ilişki tipinin gereksiz oluşu ve çok çok ilişki tipinin hem performans hemde hafıza olarak sistemde sorun çıkartmasıdır. Bu indirgeme konularını Normal Forms ve Composition konuları altında okuyabilirsiniz.

ERD çizimleri açısından olaya bakıldığında sayısallık (cardinality) kaz ayağı veya birim olarak ifade edilir. Aşağıda bir örnek üzerinde bu durum gösterilmiştir:



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. Çokla çok olması durumunda iki tarafta da kaz ayağı olurken, teke tek olması durumunda iki tarafta da çizgi ile gösterilmektedir.

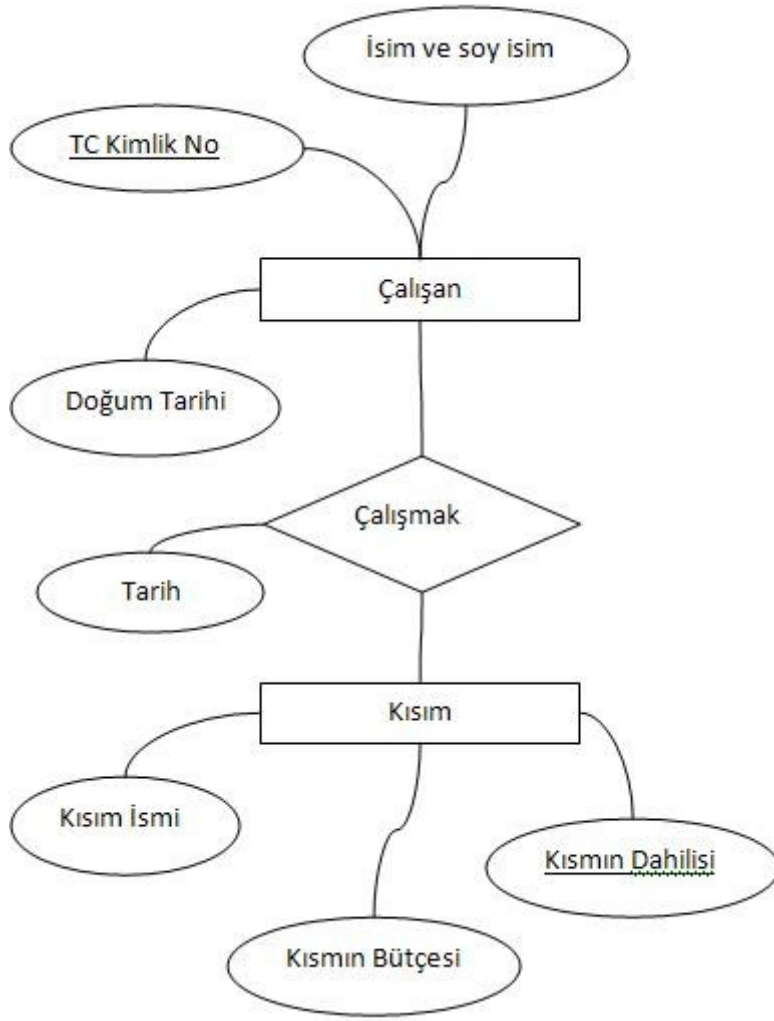
SORU 88: ERD (Unsur İlişki Çizimi, Entity Relationship Diagram)

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 89: Unsur (Entity)

[Veritabanı tasarımında \(database design\)](#) 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 herşeyin birer unsur olabileceğidir. Örneğin insan sınıfının bir özelliği olarak isimi ele alalım.

Bu özelli 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 90: Extranet (Dış ağ)

Kabaca bir kurumun dışarıya açık ağı anlamına gelmektedir. Örneğin bir firmanın kendi sunucularına erişim için kurmuş olduğu kurumsal ağı (Intranet) dışarıya açılması ve iş yaptığı çeşitli firmaların erişimi için dışarıdan erişilebilir bir ağ sağlaması durumudur.

Farklı bir bakışla Intranet'in (iç ağ) bir parçası olarak görülüp, dışarıdan bağlanacaklara açılmış bir parça olarak yorumlamak mümkündür.

Genellikle kurumsal yapıları bağlayan işten işe (Business to Business (B2B)) ağlar ve son kullanıcıları kurumlara bağlayan işletim müşteriye (Business to consumer (B2C)) kuruluşlar bu yapının altında düşünülebilir.

Ağ yapılandırması açısından Intranet'i tek bir VPN (Virtual Private Network, Sanal Kişisel Ağ) olarak düşünecek olursak, dış ağ (extranet) kavramını bu VPN'e ilave dış VPN'ler olarak düşünmek mümkündür.

SORU 91: Intranet (İç Ağ)

Gelişen ağ teknolojileri ile birlikte İnternet'in özelleştirilmesi de mümkün olmuştur. Örneğin bir şirketin posta sunucuları (mail servers), web sunucuları (web servers) DNS'i, FTP sunucuları ve benzeri pekçok sunucusu bulunmaktadır. Şayet şirket bu sunucuları kendisine özgü olarak sadece kendi çalışanlarının erişebileceği şekilde ayarlarsa bu ağ tipine Intranet (iç ağ) ismi verilmektedir.

Kısaca bir kurumun (şirket, üniversite, dernek gibi) kendisine özgü bir internet kurması durumudur.

Tek merkezli olup bütün sunucuların bir merkezden yönetildiği ve kullanıcıların çeşitli yerlerden bağlanabildiği sistemler olduğu gibi, çok merkezli olup her merkezde farklı yönetimlerin olduğu uygulamalar da bulunmaktadır. Örneğin çok uluslu ve çok şubeli bir yapıda her şubenin kendi sunucularını barındırması mümkündür.

Intranet uygulamalarının erişimi için tamamen özel hatların kullanılması mümkün olduğu gibi İnternet üzerinden de erişmek mümkün olabilmektedir.

Örneğin tamamen özel bir Intranet uygulamasında kullanıcılar ya şirketlerinden ağı özel kablolu ile bağlanmakta ya da şirketin özel telefon hatlarına bağlanarak Intranete ulaşmaktadır.

İnternet üzerinden erişilen uygulamalarda ise kullanıcılar çeşitli güvenlik aşamalarını geçerek ağı ulaşmaktadır. Bu tip erişimde iki farklı ağda paketler gittiği için tünelleme (tunnelling) kullanılabilir. Yani Intranette gidecek olan paketler İnternet paketlerinin içerisine konularak iletilebilir.

SORU 92: Çift Yönlü İletişim (Duplex Communication)

Bir iletişimin tipini belirlemek için kullanılan terimdir. Kabaca bir otayolun tek yönlü veya çift yönlü olması mümkündür. Çift yönlü otayolun ise tek şerit veya çift şerit olması mümkündür.

Duplex terimi aynı anda iki tarafında iletişim kurduğu sistemler için kullanılmıştır. İkiye ayırmak mümkündür:

- Full Duplexing (aynı anda çift yönlü iletişim)
- Half Duplexing (paylaşımlı çift yönlü iletişim)

Full duplex iletişimde aynı anda iki taraf da konuşabilmektedir. Örneğin telefon konuşmalarında her iki tarafta aynı anda konuşabilmekte ve iki tarafta birbirini

duyabilmektedir. Genellikle tek hat üzerinde birden fazla tarafın iletişim kurması [frekans paylaşımı ile iletişime mümkün olmaktadır \(Frequency division multiplexing\)](#)

Half duplex iletişimde ise bir taraf konuşurken diğer taraf beklemek zorundadır. Örneğin telsiz konuşmalarında bir taraf konuşurken diğer taraf konuşamaz. Konuşmaya çalışırsa iki taraf da birbirini duyamaz. Bu durumda hattın doğru zamanlamasının yapılması gerekir ([Zaman paylaşımı ile iletişim, Time division multiplexing](#))

SORU 93: Borulama (Pipelining)

ismini boru hatlarının işleyişinden alan yaklaşım, kısaca bir işlem borunun sonundayken, borunun başından yeni bir işin konulabileceğini anlatmaktadır.

Buna göre örneğin bir işin (process) çalışması için 4 farklı safhadan geçmesi gerekiyor olsun:

Fetch (almak)

Decode (algılamak)

Execute (çalıştırmak)

Store (saklamak)

Bu dört safhanın her birisi her işlem için tekrarlanacaktır. Yani örneğin P1 isimli işlem sırasıyla Fetch, Decode, Execute ve store aşamalarından geçecektir.

Bu durum bize her aşamada ayrı bir işlemi çalıştırmak gibi bir avantaj doğurur. Aşağıdaki temsili resimde bu durum açıklanmıştır:

Zamanlama	Fetch (almak)	Decode (algılamak)	Execute (çalıştırmak)	Store (saklamak)
T1	P1			
T2	P2	P1		
T3	P3	P2	P1	
T4	P4	P3	P2	P1
T5	P5	P4	P3	P2

Yukarıdaki tabloda, zaman en solda gösterilmiştir. Yani anlık olarak T1 zamanından sonra T2 zamanı gelmektedir. Bu akışta göre en üstte olan ilk satırda sisteme P1 işlemi alınmaktadır. İkinci zamanda (T2) ise P1 işlemi decode aşamasına geçtiğinde P2 işlemi sisteme alınarak Fetch edilmektedir.

Bu işlem böylece devam etmekte en sonunda T5 zamanında P1 işi bitirilmiş, P2 işi Store edilmekte iken P5 işi işlenmeye başlanarak fetch edilmektedir.

Görüldüğü üzere yapılacak iş 4 aşamaya bölündüğünde aynı anda 4 farklı iş işlenmektedir. Bu işleme yöntemi sanki bir borudan akar gibi (örneğin T5 anında, P1 borudan (pipeline) çıkmış, P2 borunun bir ucunda P5 ise diğer ucunda gibi düşünülebilir) olduğu için bu sisteme borulama (pipelining) ismi verilmiştir.

Günümüz işlemci mimarilerinde de oldukça yaygın olarak kullanılan bu yaklaşım sayesinde işlem gücünün artırılması mümkündür.

SORU 94: Tip İnkılabı (Tip Dönüştürme, Type Casting)

Programlama dilinde bir değişkenin (variable) tipinin değiştirilmesi anlamındadır. Temel olarak değişkenlerin tanımlandığı andan itibaren bir tipi bulunur. Bunun her ne kadar bazı dillerde istisnası olsa da (php, visual basic, perl gibi) bu dillerde de değişkenin tipi yine de tipsiz olarak belirtilmektedir.

İşte tanımlama sırasında sahip olunan tipin daha sonradan değiştirilmesi tip inkılabı (typecasting) olarak adlandırılır. Örneğin aşağıdaki C dilindeki kodu ele alalım:

```
int a;  
a=10;  
float f;  
f = a;
```

Yukarıdaki örnekte bir int tipinde değişken olan a'nın değeri float tipindeki f değişkeninin içerisine atanmıştır (Assignment). Bu satırdan sonra f değişkeninin içinde 10.0 değeri float olarak bulunacaktır.

Tersi olarak aşağıdaki kodu inceleyelim:

```
float f;  
f=3.14;  
int a;  
a = f;
```

Yukarıdaki kodda ise float olan değişkenin değeri int olan a değişkeninin içerisine atanmak istenmiştir. Çoğu derleyici (compiler) tarafından hatalı bir satır olarak görülen ve derleme sırasında hata veren yukarıdaki kodun son satırını aşağıdaki şekilde düzeltmeniz gerekebilir:

```
a = (int) f;
```

Bu satırın anlamı f değişkeninin içerisinde değeri al tipini int yap ve a değişkeninin içerisine koy demektir. İşte burada yapılan işlem tip değiştirmektir ve (int) olarka geçen komut bir tip inkılabı operatörüdür (type casting operator). Bu işlemden sonra dikkat edilecek bir husus da a değişkeninin içerisinde artık 3 sayısal değeri olduğu ve .14 küsuratının artık kaybedildiğidir. (örneğin 3.90 değeri konulsaydı yine 3 değeri alınacaktı, burada bir yuvarlama yapılmaz sığıldığı kadarı alınır, yani float değişkendeki ondalıklı kısım atılarak tam sayı kısmı int değişkenin içerisine konulur)

Nesne yönelimli programlama dillerinde tip dönüştürmek için özel fonksiyonlar da yazılabilir. Örneğin JAVA'da kullanılna toString() (String (Dizgi, metin) dönüştür) fonksiyonu buna güzel bir örnektir:

```
class insan{  
    String isim;  
    int boy;  
    public String toString(){  
        return " isim : " + isim + " boy : " + boy;  
    }  
}
```

```

}
class deneme{
    public static void main(String args[]){
        insan ali = new insan();
        ali.isim = "Ali Baba";
        ali.boy = 180;
        System.out.println(ali);
    }
}

```

Yukarıdaki kod parçasında iki farklı sınıf (class) bulunmaktadır. İlk sınıfta insan tipi tanımlanmış ve bu sınıfın içerisine bir toString() fonksiyonu yerleştirilmiştir. İkinci sınıf'ın içerisinde ise main fonksiyonu bulunmakta ve çalıştığında ali isminde bir insan nesnesi tanımlamakta ardından bu nesnenin ismine "Ali Baba" boyuna ise 180 değerlerini koymaktadır.

Son satırda olan işlem ise oldukça ilginçtir. Bu satırda System.out.println fonksiyonu ile ali nesnesi ekrana basılmıştır. Ancak ali nesnesi bilindiği üzere insan sınıfındandır ve bilindiği üzere println fonksiyonu ekrana dizgi (string) basmaktadır ve println'in insan basan bir üzerine yüklemesi (overload) bulunmamaktadır.

Bu durumda JAVA otomatik olarak insan sınıfından türetilmiş olan ali nesnesini dizgiye (string) çevirmeye çalışır. Bunun ise tek yolu bu sınıftaki toString() fonksiyonunu çağırmasıdır. Ve sonuç olarak ekrana aşağıdaki satır basılır:

isim : Ali Baba boy : 180

SORU 95: POP3

POP3 protokolü, post office protocol (postahane, postane protokolünün) kısaltılmışıdır. Bu protokolün çalışma mantığı sürekli bağlı kalmayan kullanıcıların gelen iletilerinin sunucuda saklanmasına dayanır.

Buna göre her kullanıcının, sunucu üzerinde bir posta kutusu bulunur ve gelen iletiler (e-postalar, e-mails) bu kutuda biriktirilir. Kullanıcı sunucuya bağlandığı zaman kutusunda bulunan mektupları kendi bilgisayarına çekerek bağlı olmadığı süre içerisinde de okuyabilir. Sunucuda bulunan mektupları ise ister siler ister olduğu gibi bırakır.

POP1 ve POP2 protokollerinin gelişmiş hali olan POP3'ten sonra günümüzde henüz yaygın olarak kullanılmayan ve klasör özelliği, mektubu parçalara ayırmak gibi özellikleri içeren POP4 protokolü de bulunmaktadır.

POP3 protokolü çok özel bir ayar yapılmadıktan sonra 110 numaralı port üzerinden hizmet vermektedir. Aşağıda örnek bir POP3 protokolü iletişimi gösterilmiştir:

```

S: <wait for connection on TCP port 110>
C: <open connection>
S:  +OK POP3 server ready <sadi@bilgisayarkavramlari.com>
C:  APOP sadi c4c9334bac560ecc979e58001b3e22fb
S:  +OK sadi's maildrop has 2 messages (320 octets)
C:  STAT
S:  +OK 2 320
C:  LIST

```



```
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
C: RETR 1
S: +OK 120 octets
S: <the POP3 server sends message 1>
S: .
C: DELE 1
S: +OK message 1 deleted
C: RETR 2
S: +OK 200 octets
S: <the POP3 server sends message 2>
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
C: <close connection>
S: <wait for next connection>
```

Yukarıdaki örnekte S harfi ile başlayan satırlar Sunucu (server) mesajları ve C harfiyle başlayanlar İstemci (Client) mesajları olmaktadır. Yukarıdaki iletişimde öncelikle sadi@bilgisayarkavramlari.com adresinin sunucu üzerinde sisteme girişi yapılmakta ardından sunucuda bulunan 2 mesajdan ilkinin okuyup silmekte ve son olarak 2. mesajı okuyarak sistemden çıkmaktadır.

SORU 96: Yığın İş (Batch Job, Batch Process)

Bilgisayar bilimlerinde, belirli bir zamanda yapılması planlanan yoğunlukla kullanıcı etkileşimi gerektirmeyen işlerin biriktirilmesi.

Örneğin sistemin yedeğinin alınması için 10 ayrı bilgisayara bağlanılarak her bilgisayardan dosyalar alınıp sunucuya kaydedilecek olsun. Bu işlemi şirketin kapalı olduğu gece 3.00'da yapmak istiyoruz. Bunun için bir yığın iş (batch process) hazırlayarak sistemde saklanır. Beklenen zaman geldiğinde bu iş otomatik olarak çalışır ve sistemin yedeğini alır.

Çeşitli işletim sistemlerinde farklı kullanımları vardır.

Örneğin windows işletim sisteminde bulunan zamanlanmış görevler (scheduled tasks) vasıtasıyla istenilen zamanda bir işin çalışması mümkündür. Daha eskiden DOS işletim sisteminde .bat uzantılı dosyalar da batch file (yığın dosya) olarak literatürde geçerler. Bu dosyalarda çalışacak olan komutlar arka arkaya sıralanmakta ve sırayla çalıştırılmaktaydı.

Bu dosyalardan en meşhuru da Autoexec.bat dosyasıdır. İşletim sisteminin özel dosyalarından olan bu dosya ilk açılışta okunan ve sistem açılınca yapılması istenen özel işlerin sıralandığı dosyaydı. Windows işletim sistemiyle birlikte başlangıç (startup) grubunda bulunan işler buna benzetilebilir.

Unix (veya linux) işletim sisteminde ise örneğin “at” komutu ile istenilen zamanda çalışmak üzere bir iş tanımlanabilir.

Örneğin “cp /etc/passwd /yedekek/passwd” komutu etc altındaki passwd dosyasını /yedekek dizinine kopyalar. Bu işlemi otomatik olarak saat 15’te çalıştırmak istersek

```
cp /etc/passwd /yedekek/passwd | at 1500
```

veya benzer şekilde

at 1500

```
> cp /etc/passwd /yedeck/passwd
```

```
> ^D (burada çıkmak için shift+D tuşlarına basıyoruz)
```

şeklinde sisteme 15.00'da çalışacak otomatik bir iş tanımlamış oluruz.

Bir işin belirli periyotlarla sürekli tekrarlanması isteniyorsa cron ismi verilen daemon (sistem arkaplan işi) üzerinde tanımlama yapılabilmektedir.

SORU 97: Kıtlık (Starvation)

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.

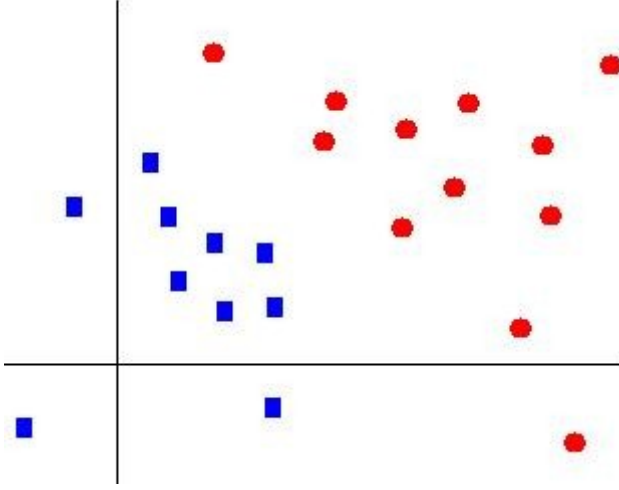
Şöyle bir örnek düşünelim, elimizde uzunlukları 4,5,6 olan işler olsun ve en kısa işi tercih eden bir algoritmamız olsun (Shortest Job First, SJF) . Algoritmamız en kısa olan 4 uzunluğundaki işten başlayacaktır. Ve diyelim ki 4 biter bitmez veya henüz bitmeden uzunluğu 3 olan başka bir iş gelsin. Algoritmamız 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 98: KNN (K nearest neighborhood, en yakın k komşu)

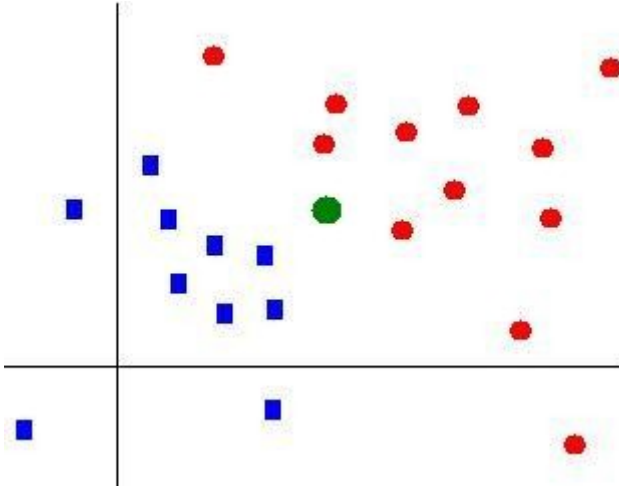
Sınıflandırmada (classification) kullanılan bu algoritmaya göre sınıflandırma sırasında çıkarılan özelliklerden (feature extraction), sınıflandırılmak istenen yeni bireyin daha önceki bireylerden k tanesine yakınlığına bakılmasıdır.

Örneğin $k = 3$ için yeni bir eleman sınıflandırılmak istensin. bu durumda eski sınıflandırılmış elemanlardan en yakın 3 tanesi alınır. Bu elemanlar hangi sınıfa dahilse, yeni eleman da o sınıfa dahil edilir. Mesafe hesabından genelde öklit mesafesi (euclid distance) kullanılabilir.

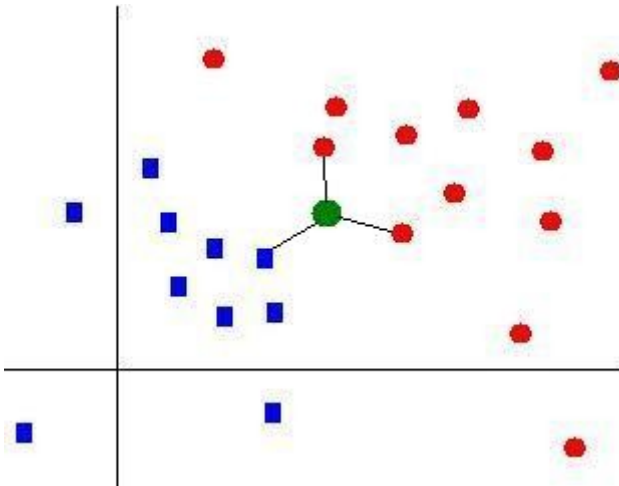


örneğin yukarıda verilen ve özelliklerine göre 2 boyutlu koordinat sistemine yerleştirilmiş olan örnekleri ele alalım. Bu örneklerin birbirinden ayrılması doğrusal ayrıştırma (linear discrimination) problemidir ve buradaki yöntemlerle çözülür.

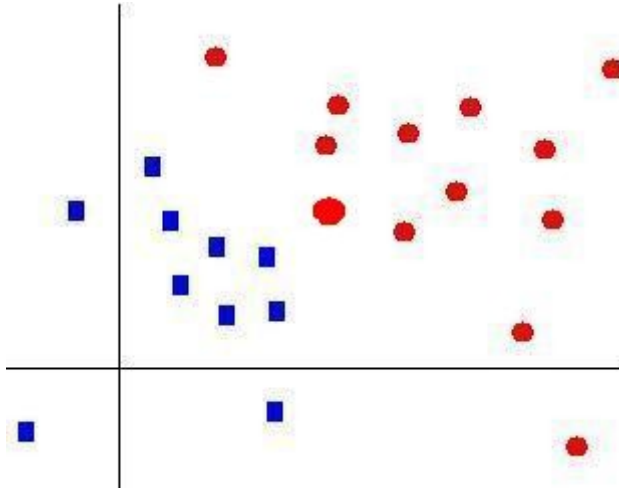
KNN yöntemine göre aşağıdaki şekilde yeni bir üyenin geldiğini düşünelim:



Yukarıdaki bu yeni gelen üyenin en yakın olduğu 3 üyeyi (3 nearest neighbors) tespit edelim.



En yakın 3 üyenin iki tanesi kırmızı yuvarlak üyeler olduğuna göre yeni üyemizi bu şekilde sınıflandırabiliriz:



SORU 99: Çevirici (Assembler)

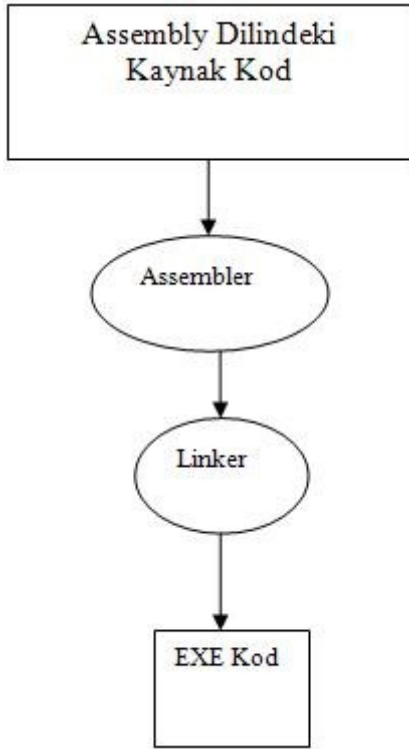
Bilgisayar bilimlerinde iki farklı kavram için assembler kelimesi kullanılmaktadır. Birincisi Assembly dili adı verilen ve makine diline (machine language) çok yakın düşük seviyeli (low level language) için kullanılan ve nesne kodunu (object code) makine koduna (machine code) çeviren dildir. İkincisi ise birleştirmek, monte etmek anlamında örneğin nesne yönelimli dillerde nesnelerin birleştirilmesi monte edilip büyük parçaların çıkarılması anlamında kullanılmaktadır.

Assembly dili, CPU üzerinde programcıya tanınmış olan yönergeleri (instructions) alarak bunları işlemcinin doğrudan çalıştırdığı makine kodları haline getirir. CPU üzerindeki bu yönergeler genelde üretici tarafından belirlenir ve tamamen teknoloji bağımlıdır. Diğer bir deyişle aynı kod farklı yapıdaki işlemciler üzerinde çalışmaz. Hatta çoğu zaman aynı mimarinin farklı nesilleri arasında bile sorun yaşanır.

Assembly dili macro destekleyen ve desteklemeyen olarak ikiye ayrılır. Tarihi gelişim sürecinde ihtiyaç üzerine dilde macro yazmak ve yazılan bu macroları tekrar tekrar kullanmak imkanı doğmuştur. Bu tip assembly dilini makine diline çeviren çeviricilere de macroassembler adı verilir.

Assembly dilleri temel olarak fonksiyon(function) veya prosedür (procedure) desteği barındırmazlar. Hatta döngü ve koşul operatörleri de yok denilebilir. Bunun yerine programın içerisinde istenen bir satıra (adrese) gidilmesini sağlayan GOTO komudu kullanırlar.

Yapısal programlama (structured programming) dokusuna ters olan bu yaklaşım dilin performans kaybı ve düşük seviye olmasından kaynaklanmaktadır.



Yukarıdaki şekilde assembly kaynak kodunda verilen bir girişin assembler'dan geçerek bağlayıcı (linker) marifetiyle çalışabilir (Exe code) haline gelişini tasvir edilmiştir.

SORU 100: Bilgisayar Mühendisliği

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, İngilizce'deki "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.

Türkiye'de 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 mühendisliđi

Her disiplinin var olabilmesi için bilimsel bazı dayanaklarının olması gerekir. Özellikle de sayısal çalışmalarda bir disiplinin varlığı, matematiksel bir dayanađa bađlıdır.

Aslında bilimlerin sınıflandırılması uzun ve tartışmalı bir konudur ancak bilgisayar mühendisliğinin bir uygulamalı bilim (applied science) olduđu konusunda uzlaşma vardır. Yani daha çok matematiksel kavram ve kuramların uygulandığı bir alan olarak görülebilir.

Bilimleri şayet dođal bilimleri (fennî ilimler) ve beşerî ilimler (insana dayalı bilimler) olarka sınıflandırırsanız, bilgisayar mühendisliğinin insana bađlı biri bilim olduđu kesindir. Bu sınıflandırmada temel rol, dođadan öğrenilen veya insan tarafından üretilen bilim olmasına göre yapıldığına göre, bilgisayar bilimlerinin büyük çoğunluğu fizik, biyoloji, kimya gibi dođadan 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 dođru olur. Her ne kadar bu disiplinin temeline ayrık matematiđi (Discrete mathmetics) oturtan yorumlar olsa da günün birinde sürekli çalışan (continous) bir bilgisayar hakim olabilir ve bu olduğunda bu bilgisayarlar ve üzerlerindeki çalışmalar da bilgisayar mühendisliğinin konusu olacađı ve bu durumda sürekli matematikten bahsedileceđi için bu ayrım çok dođru deđildir.

Ancak algoritma analizi, yani bir işin nasıl yapılacađının 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 gerektiđi ortaya çıkabilir. Bunun sebebi biraz daha ingilizce 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 tarafında yapılıyordu. Ancak günümüzde bu bilgisayarlar ile yer deđiştirmiş ve hızla da deđiştirmeye devam etmektedir. Yine bir örnek üzerinden bakılacak olursa, trafiđi günümüzde bilgisayarlar kontrol ediyor ve teknoloğimiz tamamen trafiđi izleyip öğrenerek en verimli şekilde örneğin bir kavşağı yönetecek bilgisayar yapmaya yeterli (ve örnekleri var) ancak bilgisayarlardan önceki dönemlerde bu iş kavşađa yerleştirilen bir polis memuru ile yapıyor 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 pekçokkişiye 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 akademiye 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 yeyne 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 101: Yerleştirme Algoritmaları (Fitting Algorithms)

Bilgisayar bilimlerinde kısıtlı bir alanın verimli kullanılması için geliştirilmiş algoritmalar. Örneğin sınırlı bir [hafıza \(RAM \)](#) içerisine en verimli şekilde programları yerleştirmek, işletim sistemleri için bir problemdir. Benzer problemlerle gerçek hayatta da sıkça karşılaşmaktadır. Örneğin bir deponun verimli kullanılması veya bir kamyonun verimli yüklenmesi veya haftalık bir ders programına derslerin verimli yerleştirilmesi gibi.

Bu problemlerin çözümü için genelde bin packing ismi verilen bir algoritma kullanılır. Bu yazının amacı bin packing algoritması da dahil olmak üzere genel olarak izlenen yerleştirme stratejilerini açıklamaktır.

Temel olarak bir yerleştirme işleminde şu 3 yöntemden birisi izlenebilir

- İlk bulunan yere yerleştirme (First fitting)
- En iyi yerleştirme (best fitting)
- En kötü yerleştirme (worst fitting)

Bu yöntemler isimlerinden de anlaşılacağı üzere kısaca:

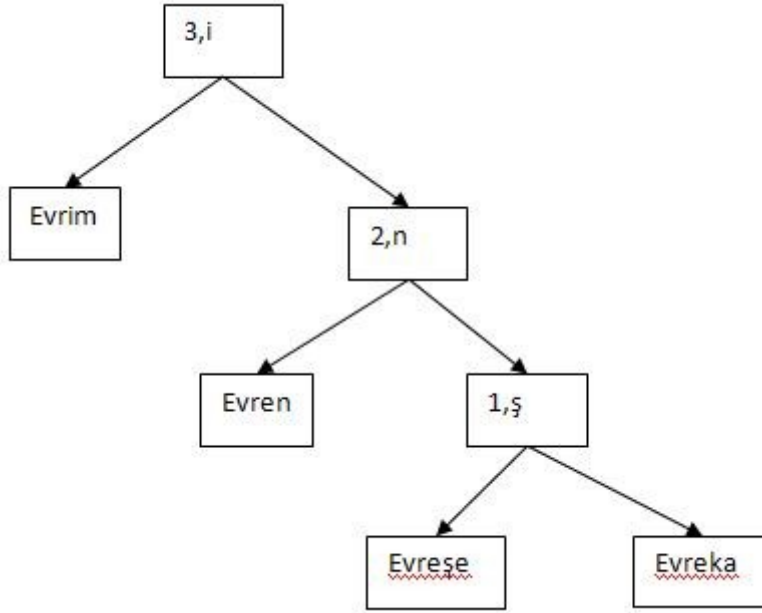
Yerleştirmek istediğimiz şeyi (hafızaya yerleştirilecek bir program, depodaki bir kutu gibi) ilk bulduğumuz boş yere yerleştiriyorsak buna first fitting, en az boş yer kalacak şekilde bir yer arıyor ve yerleştiriyorsak buna best fitting şayet yerleştirdikten sonra en fazla boş yer kalacak yere yerleştiriyorsak buna da worst fitting ismi verilir.

SORU 102: Patricia ağacı (PATRICIA Tree)

Bilgisayar bilimlerinde sıkça kullanılan [TRIE ağacının](#) özel bir hali olan patricia ağacında genellikle sözlüksel olarak (lexiconically) veriler tutulur. Radix ağacı (radix tree) ve farklı ikil ağacı (crit bit tree) ile oldukça benzer olan patricia ağacının, TRIE ağacından en büyük farkı tutulan verilerin ortak olan noktalarından sonra farklılaşan yönlerine göre dallanma olmasıdır.

Aşağıda verilen kelimelerin ağaçta tutulmaları gösterilmiştir:

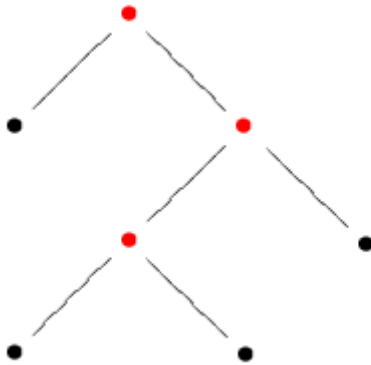
- Evren
- Evreşe
- Evreka
- Evrensel



Yukarıdaki şekilde de gösterildiği üzere ağacın dallanmaları verilen kelimelerin birbiri ile farklılaştıkları noktalarda olmaktadır.

SORU 103: Dış Yol Uzunluğu (External Path Length)

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

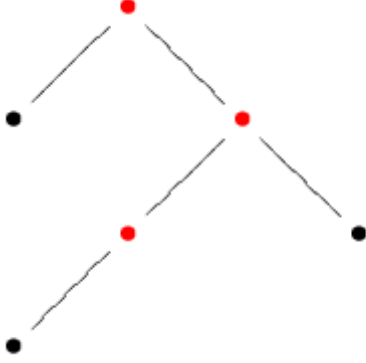
n: iç düğüm sayısıdır.

Yukarıdaki örnekte iç yol uzunluğu 3'tür. İç düğüm sayısı da 3 tür. Dolayısıyla $E = 3 + 2 \times 3 = 9$ bulunabilir.

Yukarıdaki grafiğin iç yol uzunluğu için ilgili yazıyı okuyabilirsiniz.

SORU 104: İç düğüm (Internal Nodes)

Bir ağacın, uçlarında olmayan düğümleridir. Örneğin aşağıdaki grafikte iç düğümler kırmızı, dış düğümler ise siyah renk ile gösterilmiştir:



SORU 105: Uzaysal Çözünürlük (Spatial Resolution)

En basit anlamda bir uzayda bulunan örneklerin birbirine olan uzaklığını belirtir. Yani bir uzaydan (örneğin 3 boyutlu bir ortamdan) bir örnek alındığında (örneğin bir kalemin bilgisi alındığında) bu bilgiler arası mesafe, örneğin çözünürlüğünü belirler. Buna göre mesafe kısaldıkça alınan örnek sayısı artar ve dolayısıyla kalemin daha çok detayı hakkında bilgi sahibi olunur ve çözünürlük artmış olur. Mesafeler uzadıkça ise örnek sayısı azalırken kalem hakkında daha az bilgi sahibi olunur.

Örneğin aşağıda orijinali verilen resmi ele alalım:



Sapancada ay ierken

Bu resimin uzaysal özünürlüğü 400×300 pikseldir ve azaltılarak aynı boyutlarda bir resim 40×30 [piksele](#) indirgenirse (yani 10 [imgecięe \(pixel\)](#) 1 imgecik denk düşecek şekilde indirgenirse) ařağıdaki halde görülür:



Sapancada ay ierken resmi 10'a 1 oranında imgecik (pixel) azaltmasına gidilmiř

Görüldüğü üzere resmin ana hatları aynı kalmakta ancak resmi oluřturan imgecikler (pixel) birer kare olarak düşünölürse bu karelerin boyutu artmaktadır.

Bu sayede ikinci resim 1. resimden ok daha az yer kaplamakta ancak daha az veri iermektedir.

SORU 106: Kuantum İřleme (Quantum Computing)

Kuantum bilgisayarları basitçe veriyi işlemek için çok küçük parçacıklar kullanır. Örneğin her gün yolda görebileceğimiz basit bir çakıl taşı aslında bir kuantun işlemi olarak kabul edilebilir. Temelde çakıl taşının yaptığı iş uzayda (kainatta) çok küçük parçacıkların bir arada durmasını sağlaması ve neticede bir konumlandırma işlemi yapmasıdır.

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.

Kuantum hesaplamalarının en büyük farklılığı Kuantum paralelligidir. Kuantum paralelligi (Quantum parallelism) adı da verilen bir 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.

Kuantum işleme sırasında performans avantajı sağlandığı bir gerçektir. Bu avantajın nasıl sağlandığını anlamak için 200 haneli bir sayıyı çarpanlarına ayıracağımızı düşünelim. Bu işlem günümüz teknolojisindeki 1500 kadar bilgisayarın paralel çalışması ile yaklaşık 700bin yıl sürmektedir. Kuantum bilgisayarları kullanılarak bu işlem ise yaklaşık bir kaç milyon işlem ile sonuca ulaşmaktadır. Buradaki temel fark aynı anda birden fazla durumun kuantum bilgisayarları ile paralel olarak işlenebilmesidir (Kuantum paralelligi).

SORU 107: Kubit (Qubit)

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 üzerine inşa edilmiştir. Bu konuda hidrojen güzel bir örnek olabilir. Bilindiği üzere hidrojen atomu basit bir elementtir ve bir elektron ve bir çekirdekten 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 ikillerden (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 paralelligi (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 108: Doğrusal Ayrılabilirlik (Linear Seperability)

Yapay sinir ağlarının en basit anlamda incelenebilmesi için problemi iki adet ikil haneleri olan (binary digists) bir girdiye bir de tek ikil (binary) çıktıya sahip bir örnek üzerinden inceleyelim.

Aşağıda iki farklı fonksiyonun gerçeklik çizelgesi (doğruluk tablosu, truth table) verilmiştir. A ve B değerleri girişi C ise çıkışı ifade etsin:

F fonksiyonu için :

A B C

0 0 1

0 1 0

1 0 1

1 1 1

G fonksiyonu için

A B C

0 0 0

0 1 1

1 0 1

1 1 0

Olarak tanımlanmış olan bu fonksiyonların ilki (F fonksiyonu) doğrusal olarak ayrılabilirken ikincisi (G fonksiyonu) doğrusal olarak ayrılabilir değildir. Bu durumu aşağıdaki karnaugh haritaları (karnaugh map) üzerinde göstermeye çalışalım:

	0	1		0	1
1	1	1		0	1
0	1	0		1	0

Görüldüğü üzere ilk fonksiyonun (F) karnaugh haritası üzerinde bir doğru ile sonuçları iki gruba bölmek doğrusal ayırım yapmak mümkün iken (linearly seperable) ikinci fonksiyon (G) için aynı doğrusal ayırım işlemi yapılamaz. Aynı zamanda ikinci fonksiyon sonuçları için karnaugh haritası dışında farklı bir harita çizmek ve yine doğrusal olarak ayıran bir çizgi elde etmek de imkansızdır. Yani kolon veya satırları yer değiştirdiğinizde bütün alternatiflerde doğrusal olarak ayrılamayan bir sonuç elde edersiniz.

Bir problemin yada fonksiyonun doğrusal olarak ayrılabilir olup olmaması problemin yapay sinir ağı tarafından çözülebilirliğinin kolaylığını da belirlemektedir. Ne yazık ki doğrusal olarak ayrılamayan problemleri yapay sinir ağı ile çözmek çok daha zordur.

Dikkat edilirse yukarıdaki doğrusal olarak ayrılamayan fonksiyon bir yahut (özel veya exclusive or)) fonksiyonudur. Literatüre de bu şekilde girmiş olan doğrusal olarak ayrılamayan fonksiyonlar (veya kısaca “XOR problem (yahut problemi)”) programlama dünyasındaki “Hello World” yazdırmak kadar meşhurdur.

SORU 109: Akış Diyagramı (Flow Chart)

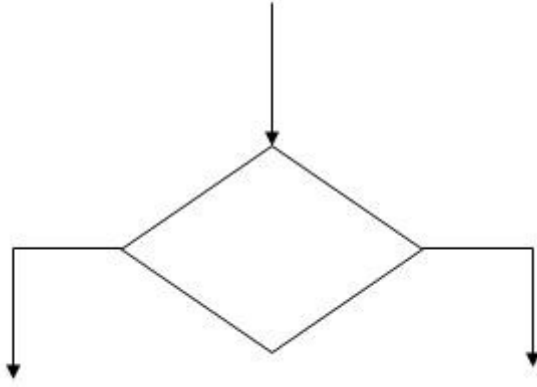
Bilgisayar algoritmalarında, algoritmanın görsel olarak tasvir edilebilmesi için geliştirilmiş bir çizim yöntemidir. Buna göre [yapısal programlamaya](#) uygun olarak geliştirilmiş bir programlama dili aşağıdaki üç temel özelliği içerir:

- Bir alt programa dallanmak (fork)
- [Bir alt programı tekrarlamak \(loop\)](#)
- [Bir alt programı icra etmek \(execute\)](#)

İşte temel olarak bu 3 işlemi yapabilen her programlama dili [yapısal programlamaya \(structured programming\)](#) uygun olarak kabul edilir. Bu işlemleri tanımlayabilen bir akış diyagramı ise basitçe yapısal programlama uygun dilleri modellemek için kullanılabilir.

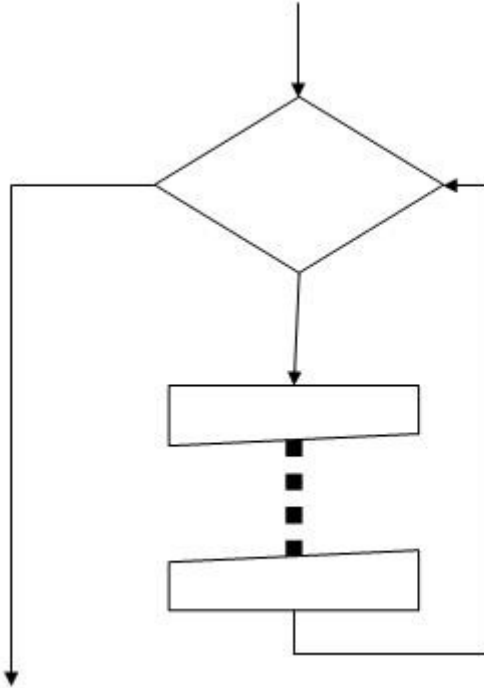
Yukarıdaki her işlemin çizim karşılığı aşağıda verilmiştir:

Dallanma (fork):



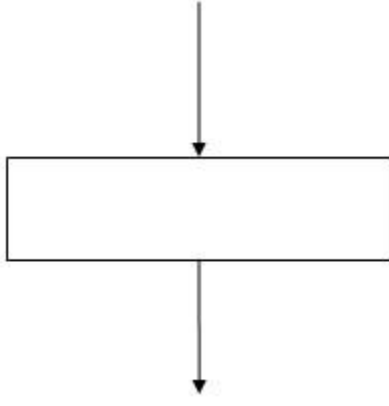
Yukarıdaki şekilde de gösterildiği üzere akış diyagramında (flow chart) gelmekte olan akış bir koşula bağlı olarak sola veya sağa doğru dallanmaktadır.

Döngü (loop):



Yukarıda görüntülenen tasvirde gelen akış bir seçime göre sola veya aşağı dallanmaktadır. Bu dallanma işlemi bittikten sonra koşula geri dönerek sorgu tekrarlanmakta ve şart sağlandığı sürece aynı [alt grup](#) (burada siyah kareler ile gösterilmiştir) tekrarlanmaktadır. Bu şart sağlanmadığı takdirde sola dallanarak döngüden çıkılır.

Çalıştırma (Execute):



Çalıştırma işlemi bir programın en basit anlamda yaptığı her satırdır (statement). Basitçe bir [alt program \(su bprogram, subroutine\)](#) olarak kabul de edilebilir.

SORU 110: Yapay Sinir Ağları (Artificial Neural Networks)

Bilindiği üzere bilgisayarlar insanlara göre çok daha hızlı işlemler gerçekleştirebilmektedir örneğin toplama çarpma gibi matematiksel işlemler insanlara göre çok daha hızlı yapılabilmektedir. Ancak bazı işlemlerin gerçekleştirilmesi sırasında bilgisayarlarda modelleme sorunu ortaya çıkmaktadır. Yani her problem matematiksel yöntemlerle modellenememekte bunun sonucu olarak da bilgisayarların işlem gücü insanlara göre verimsiz kullanılmaktadır. Örneğin ilk okul çağındaki bir insan basit bir kedi ve köpek resmini birbirinden çok kısa sürede ayırabilmekte ve kendisine verilen resimleri doğru olarak sınıflandırmaktadır. Benzeri bir işlem için bilgisayarlarda çok uzun işlem süreleri gerekmektedir.

Yapay sinir ağlarının çıkışı tam da bu noktaya denk gelmektedir. Yani bu çalışma konusunun amacı çeşitli sebeplerle insanların bilgisayarlardan daha hızlı yapabildiği işlemleri, bilgisayarların da benzer hızlarda yapılması için yeni modeller geliştirmek ve bilgisayarların işlem gücünü verimli olarak kullanmaktır.

Yapay sinir ağları çalışmaları, bilgisayar mühendisliğinin bir alt disiplini olan yapay zekanın çalışma alanına girmektedir. Bu disiplinde insan beyni ve beyinde bulunan nöronların çalışmalarını taklit etmek hedeflenmektedir.

İnsan beynine benzer bir çalışma şekli bulunduğu ve bunu taklit eden bir disiplin olduğu için “Yapay” sinir ağları (Artificial Neural Network) olarak da adlandırılmakta olan çalışma bazı kaynaklarda sadece sinir ağları (neural network) olarak isimlendirilmektedir. Bu isim karışıklığının sebebi yapılan çalışmanın yapay olup olmamasıyla ilgili felsefi bir tartışmadan öte olmayıp bu site ve bu sitedeki konularla ilgili değildir. (Daha çok bilim tarihi veya bilim felsefesi konularında benzeri tartışmaya cevaplar aranabilir (örneğin “What is an artifact” sorusunun cevabının aranması gibi)). Bu sitede iki isim de yeri geldiğinde özgürce kullanılmaktadır.

SORU 111: Nazariye (Teori, Kuram, Theorem)

Bilgisayar bilimleri açısından matematiksel olarak ispat edilmiş kazyeler (önermeler, statements) birer nazariyedir. Bazı kazyelerin(önermelerin) doğruluğu ise sırf farklı teorilerin ıspatına yardımcı oluyor diye ıspatlanır. Bu tip kazyelere (önermelere) ise önkuram (önsav, lemma) adı verilmektedir. Bazı durumlarda ise bir nazariyenin ıspatı bize bazı başka neticelerin doğru olduğunu gösterir. Bu tarz kendiliğinden doğruluğu anlaşılan kazyelere(önermelere) ise tabiî sonuç anlamına gelen sonurgu (corollary) denilmektedir.

SORU 112: Dizgi (String)

Bir dilde bulunan ve o dilin tanımlı olan alfabesi içerisindeki sembollerin çeşitli sayılarda ve çeşitli sırada dizilmesi ile elde edilen yazılardır.

Örneğin bir dildeki alfabe aşağıdaki şekilde tanımlı olsun:

$$\Sigma_1 = \{0,1\}$$

Buna göre dilimizde sadece “0” ve “1” sembolleri tanımlı demektir. Bu dilde örneğin $w_1=0$ veya $w_2=10101011010$ gibi bir dizgi elde etmek mümkündür.

Bir dizginin belirli bir kısmını içeren dizgiye ise alt dizgi adı verilir. Örneğin $w_3=1011$ dizgisi w_2 dizgisinin bir altdizgisidir.

Ayrıca iki dizginin arka arkaya eklenmesine de üleştirme(concatenation) denilir.Örneğin w_1 ile w_3 dizgilerinin üleştirilmiş hali $w_4=01011$ olur.

Dizgiler ile ilgili diğer yazılar:

- [Dizgi Parçalayıcısı \(String Tokenizer\)](#)
- [Dizgi Karşılaştırma \(String Comparison\)](#)
- [Veri Tabanı Dizgi işlemleri \(Database String manipulations\)](#)
- [En uzun ortak küme \(longest common subsequence\) problemi](#)
- [Dizgi Eş şekilliliği \(String Isomorphism\)](#)
- [C dilinde dizgi kopyalama \(strcpy\)](#)
- [Alt Dizgi \(Substring\)](#)
- [Dizgi Hizalama \(String Alignment\) Problemi](#)

Dizgi Karşılaştırma/Arama/Mesafe algoritmaları:

- [Boyer Moore Dizgi Karşılaştırması \(Boyer Moore String Comparison Algorithm\)](#)
- [Knuth Moris Prat Dizgi Arama Algoritması \(Knuth Morris Prat Algorithm\)](#)
- [Dizgi Eş şekilliliği \(String Isomorphism\)](#)
- [Needleman Wunsch Yaslama Algoritması \(String Alignment\)](#)
- [Smith Waterman Yaslama Algoritması \(String Alignment\)](#)
- [Hunt MacLlor Yaslama Algoritması \(String Alignment\)](#)
- [Horspool Algoritması \(Dizgi arama algoritması\)](#)
- [Levenstein Mesafesi \(Levenshtein Distance\)](#)
- [Hamming Mesafesi \(Hamming Distance\)](#)
- [Diff komutu](#)
- [Jaccard İndeksi, mesafesi ve katsayısı \(Jaccard Index\)](#)
- [Dice Sorensen Benzerliği \(Dice Sorensen Similarity\)](#)

Dizgi Parçalama (Parsing) algoritmaları:

- [LL\(1\) parçalama algoritması](#)
- [SLR\(1\) parçalama algoritması](#)
- [Earley Parçalama Algoritması](#)

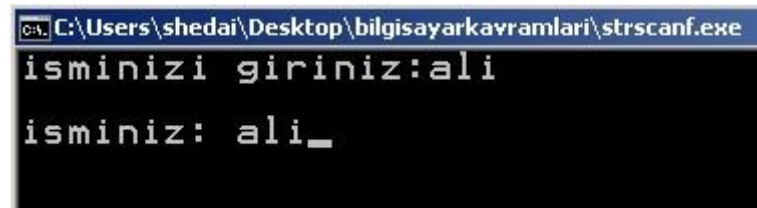
C ile dizgi okuma

C dilinde klavyeden dizgi (String) okumak için kullanılan en basit fonksiyon scanf fonksiyonudur. Bu fonksiyonu basit bir uygulamada aşağıdaki şekilde kullanabiliriz:

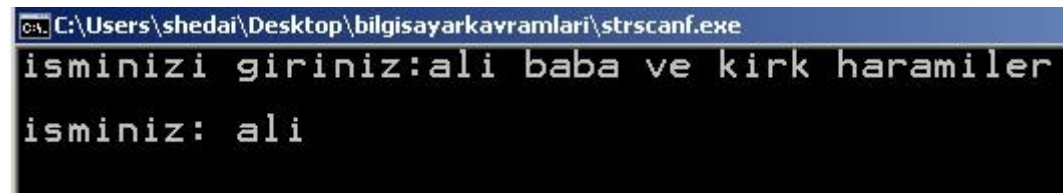
```
1  #include <stdio.h>
2  #include <conio.h>
3  //www.bilgisayarkavramlari.com
4  int main(){
5      char isim[100];
6      printf("isminizi giriniz:");
7      scanf("%s",isim);
8      printf("\nisminiz: %s",isim);
9      getch();
10 }
```

Yukarıdaki kodda, 5. satırda tanımlanan karakter dizisi (string) içerisine 7. satırda %s parametresi ile scanf fonksiyonu kullanılarak bir dizgi okunmuştur. Bu dizginin içeriği kodun 8. satırında ekrana basılmıştır.

Örneğin yukarıdaki kod aşağıdaki şekilde çalıştırılabilir:



Görüldüğü üzere, kullanıcı isim olarak “ali” girmiş ve ekranda, girdiği bu dizgiyi görmüştür. Ancak aynı kodu çalıştırarak aşağıdaki şekilde bir dizgi girilirse problem yaşanır:



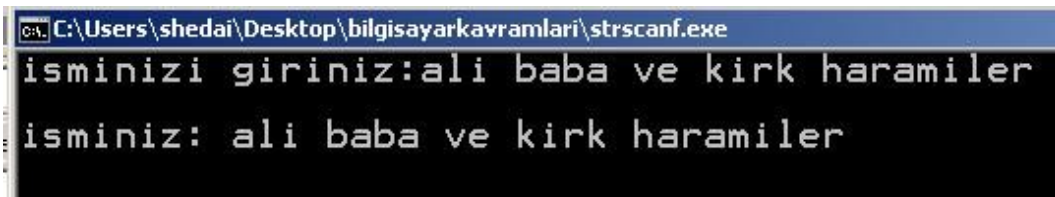
Yukarıdaki girdide “ali baba ve kirk haramiler” şeklinde 5 kelimeden oluşan bir dizgi girilirken, bu dizginin sadece ilk kelimesi scanf tarafından okunmuştur. Aslında burada bir hata yoktur çünkü scanf fonksiyonu, boşluk karakteri veya satır sonu gibi karakterlere kadar olan dizgileri okur. Yukarıdaki gibi birden fazla kelimeden oluşan dizgiler okunmak istendiğinde, aşağıdaki kodda da gösterilen gets fonksiyonu kullanılabilir:

```

1  #include <stdio.h>
2  #include <conio.h>
3  //www.bilgisayarkavramlari.com
4  int main(){
5      char isim[100];
6      printf("isminizi giriniz:");
7      gets(isim);
8      printf("\nisminiz: %s",isim);
9      getch();
10 }

```

Yukarıdaki kodda bir önceki koda göre sadece 7. satırda bulunan scanf fonksiyonu, gets fonksiyonu ile değiştirilmiştir. Kodumuzun yeni halini çalıştırdığımızda, aşağıdaki şekilde birden fazla kelime okuyabildiğimizi görürüz:



Dizgilerin Eşitliği

İki farklı değişkende bulunan dizginin eşit olup olmadığı, programlama dillerinde bulunan klasik operatörler ile yapılamaz.

Örneğin aşağıdaki kodlama C dili açısından doğru olsa bile mantıksal olarak hatalıdır (logic error):

```
char a[100] = "bilgisayarkavramlari.com";
```

```
char b[100] = "bilgisayarkavramlari.com";
```

```
if(a==b)
```

Yukarıdaki kodun, son satırında bulunan eşitlik kontrolü, C dili açısından ne derleme (compile) ne de çalışma (run-time) hatası döndürmez. Ancak buradaki karşılaştırma aslında iki dizinin (array) hafızada (RAM) aynı yeri gösterip göstermediğini sorgulamaktadır.

Yukarıdaki kod, her zaman için yanlış (false) döner ve if kontrolüne hiçbir zaman girilemez. Bunun yerine dizgilerin içeriklerinin karakter karakter kontrol edilmesi ve dizgilerin boyutlarının eşit olup olmadığının sorgulanması gerekir.. Bizim iki dizginin eşitliğinden anladığımız genelde budur. Bu kontrole literatürde derin karşılaştırma (deep compare) ismi verilmektedir.. Klasik olarak yapılan a==b kontrolü ise sığ karşılaştırma (shallow compare) olarak geçmektedir.

Derin karşılaştırma için kendimiz bir fonksiyon yazabileceğimiz gibi, C dilinde var olan string.h kütüphanesindeki strcmp fonksiyonunu kullanabiliriz. Bu fonksiyon iki dizgiyi (string) sözlük sıralamasına göre karşılaştırır (lexiconically) ve şayet eşitse 0 değerini döndürür.

C dilinde 0 değeri mantıksal olarak yanlış (false) olduğu için eşit olmaları durumunda bir if bloğunun çalışmasını istiyorsak, aşağıdaki şekilde yazabiliriz:

```
if(!strcmp(a,b))
```

yukarıdaki kontrol, a ve b dizgilerinin eşitliği durumunda geçer.

Aynı kontrol JAVA veya C# gibi dillerde, String sınıfının bulunması sayesinde, ilave bir kütüphane ve fonksiyona gerek kalmadan çözülebilir.

Örneğin JAVA dili için:

```
String a= "www.bilgisayarkavramlari.com";
```

```
String b= "www.bilgisayarkavramlari.com";
```

```
if(a.equals(b))
```

kontrolü yapılması yeterlidir. Java dilinde, bulunan ve dizgileri karşılaştırmak için kullanılan equals fonksiyonu C# dilinde ilk harfi büyük olarak Equals şeklinde yazılarak çalıştırılabilir (burada bir kere daha JAVA'yı taklit ederken, java kodlarının çalışmaması için özel gayret sarf eden microsoft geliştiricilerini selamlıyoruz.)

Dizgilerin birbirine eklenmesi (concatenate, üleştirme) için C ve C++ gibi dillerde strcat fonksiyonu kullanılabilir.

Örneğin:

```
char a[100]= "www.";
```

```
char b[100]= "bilgisayarkavramlari.com";
```

```
strcat(a,b);
```

```
printf("%s",a);
```

şeklindeki bir kod, ekrana "www.bilgisayarkavramlari.com" sonucunu basacaktır.

Aynı üleştirme işlemi, JAVA veya C# için basit bir toplama (+) işlemi ile yapılabilir.

```
String a="www.";
```

```
String b="bilgisayarkavramlari.com";
```

```
System.out.println(a+b);
```

yukarıdaki kod, ekrana "www.bilgisayarkavramlari.com" sonucunu basarken aynı kodu C# dilinde sadece son satırını Console.write(a+b) olarak değiştirerek deneyebilirsiniz.

SORU 113: Alfabe (Abece, Alphabet)

Bilgisayar bilimlerinde kullanılan ve yazıları ifade etmeye yarayan sembollerden oluşmuş kümelere verilen isimdir. Buna göre bir dildeki olası bütün semboller kullanılarak oluşturulan alfabeler kullanılarak metinlerin elde edilmesi mümkündür.

Bilgisayar bilimlerindeki alfabelerde bulunan semboller sınırlı sayıda kabul edilmiştir. Örneğin aşağıda çeşitli semboller içeren alfabe örnekleri verilmiştir:

$$\Sigma_1 = \{0,1\}$$

$$\Sigma_2 = \{a,b,c,d,e,f\}$$

Örneğin yukarıdaki Σ_1 alfabeti “0” ve “1” sayılarını birer sembol olarak kabul etmiştir ve bu semboller dışındaki semboller bu alfabede tanımlı değildir.

Alfabeler kullanılarak elde edilen dizgiler (string) w harfi ile ifade edilir. Buna göre örneğin $w = \text{debdebe}$ dizgisi Σ_2 üzerinde tanımlı bir dizgidir.

Bir dizginin boyutu $|w|$ işareti ile gösterilir ve o dizgideki harf sayısına eşittir. Örneğin yukarıda verilen $w = \text{debdebe}$ dizgisi için $|w| = 7$ olarak kabul edilir. Bir dizginin içeriği boş olması durumunda ise $|w| = 0$ olarak kabul edilir.

SORU 114: Sembol (Harf, İşaret, Symbol)

Bilgisayar bilimlerinde kullanılan ve yazıları ifade etmeye yarayan en küçük ifade birimine verilen isimdir. Buna göre bir dildeki olası bütün semboller kullanılarak oluşturulan alfabeler kullanılarak metinlerin elde edilmesi mümkündür.

Bilgisayar bilimlerindeki alfabelerde bulunan semboller sınırlı sayıda kabul edilmiştir. Örneğin aşağıda çeşitli semboller içeren alfabe örnekleri verilmiştir:

$$\Sigma_1 = \{0,1\}$$

$$\Sigma_2 = \{a,b,c,d,e,f\}$$

Örneğin yukarıdaki Σ_1 alfabeti “0” ve “1” sayılarını birer sembol olarak kabul etmiştir ve bu semboller dışındaki semboller bu alfabede tanımlı değildir.

Alfabeler kullanılarak elde edilen dizgiler (string) w harfi ile ifade edilir. Buna göre örneğin $w = \text{debdebe}$ dizgisi Σ_2 üzerinde tanımlı bir dizgidir.

Bir dizginin boyutu $|w|$ işareti ile gösterilir ve o dizgideki harf sayısına eşittir. Örneğin yukarıda verilen $w = \text{debdebe}$ dizgisi için $|w| = 7$ olarak kabul edilir. Bir dizginin içeriği boş olması durumunda ise $|w| = 0$ olarak kabul edilir.

SORU 115: Kenar (Edge)

Bir graf üzerindeki her çizgiye kenar adı verilir. kenarlar düğümleri birleştirdikleri için bu ismi almışlardır. Graf teorisinde bir kenarı ifade etmek için birleştirdiği düğümlerin isimleri yazılır. Örneğin aşağıdaki şekildeki “a” kenarını ifade etmek için (A,B) gösterimi kullanılır.

SORU 116: D ğ m (Node)

Bir graf  zerindeki her noktaya d ğ m adı verilir. D ğ mler, kenarlar kendi  zerlerinde birleřtiėi i in bu ismi almıřlardır.

Graf teorisine g re bir d ğ m n derecesi o d ğ mde bulunan kenar sayısıdır.  rnepin ařaėıdaki grafta A d ğ m n n derecesi 3't r.

SORU 117: Nesne (Object)

Nesne y nelimli programlama dillerinde bir varlıėın bizatihi kendisidir. Daha basit bir řekilde yařayan bir varlık  nce tanımlanmalıdır, bu tanımın yapılmasına sınıf adı verilir daha sonra bu sınıftan yařayan bir ya da daha  ok varlıklar oluřturulur bu her varlıėa da nesne adı verilir.

Bu durum doėal dillerdeki  zel isimlere benzetilebilir. Sınıflar ise daha  ok cins isimler gibi d ř n lebilir.  rneėin kedi kelimesi bir cins isimdir ve daha  ok bir sınıfı belirtir. Mesela kedilerin d rt ayaėı vardır demek ya da kedilerin g zleri gece az ıřıkta parlar demek doėrudur. Ancak kedilerin k rk  sarıdır demek yanlıřtır.   nk  kedilerin k rk  farklı renklerde olabilir ama  zel bir kedinin k rk  sarı olabilir.  rneėin tekir isimli kedinin k rk  sarıdır c mlesi doėrudur.

Daha basit bir ifadeyle sınıf tanımlarının ger ek hayatta  rnekleri yoktur, ger ek hayatta var olanlar nesnelerdir. Sınıflar bu nesnelerin tanımlandıėı yerlerdir.

SORU 118: Sınıf (class)

Nesne y nelimli programlama dillerinde bir varlıėın (nesne, object) tanımıdır. Daha basit bir řekilde yařayan bir varlık  nce tanımlanmalıdır, bu tanımın yapılmasına sınıf adı verilir.

Bu durum doėal dillerdeki cins isimlere benzetilebilir. Varlıklar ise daha  ok  zel isimler gibi d ř n lebilir.  rneėin kedi kelimesi bir cins isimdir ve daha  ok bir sınıfı belirtir. Mesela kedilerin d rt ayaėı vardır demek ya da kedilerin g zleri gece az ıřıkta parlar demek doėrudur. Ancak kedilerin k rk  sarıdır demek yanlıřtır.   nk  kedilerin k rk  farklı renklerde olabilir ama  zel bir kedinin k rk  sarı olabilir.  rneėin tekir isimli kedinin k rk  sarıdır c mlesi doėrudur.

Daha basit bir ifadeyle sınıf tanımlarının ger ek hayatta  rnekleri yoktur, ger ek hayatta var olanlar nesnelerdir. Sınıflar bu nesnelerin tanımlandıėı yerlerdir.

SORU 119: Anahtar Beyazlatma (Key Whitening)

řifreleme y ntemlerinde kaba kuvvet (brute force) saldırısının g  leřtirilmesi i in uygulanan yaklařımlardan birisidir. Buna g re blok řifreleme y nteminde ilk bloktan  nce ve son bloktan sonra mesaj ile anahtarın  zel veyası (exclusive or) alınarak sistem karmařıklıėı arttırılır.

İlk kullanıldığı algoritmalarından birisi olan DES-X algoritmasında klasik DES algoritmasında kullanılan 56 bit'lik anahtardan farklı olarak iki farklı 64 bit uzunluğunda anahtar kullanılmaktadır.

Anahtar beyazlatmasını DES-X üzerindeki kullanımı yukarıdaki şekilde tasvir edilmiştir.

SORU 120: Özetleme Fonksiyonları (Hash Function)

Özetleme fonksiyonlarının çalışma şekli, uzun bir girdiyi alarak daha kısa bir alanda göstermektir. Amaç girende bir değişiklik olduğunda bunun çıkışa da yansmasıdır.

Buna göre özetleme fonksiyonları ya veri güvenliğinde, verinin farklı olup olmadığını kontrol etmeye yarar ya da verileri sınıflandırmak için kullanılır.

Anlaşılması en basit özetleme fonksiyonu modülo işlemidir. Buna göre örneğin mod 10 işlemini ele alalım, aşağıdaki sayıların mod 10 sonuçları listelenmiş ve gruplanmıştır:

Sayılar: 8,3 ,4,12,432,34,95,344,549,389,2339,349,54,81,17,62,94,67,44,9

Demet (Buket, Bucket)	Sayılar					
0						
1	81					
2	12	432	62			
3	3					
4	4	34	344	54	94	44
5	95					
6						
7	17	67				
8	8					
9	389	2339	349	9		

Kısaca yukarıdaki sayıların hepsi 1 haneli bir sayıya özetlenmiştir. Örneğin 81 -> 1, 344 -> 4 gibi. Elbette aynı sayıya özetlenen birden fazla sayı bulunmaktadır. Bu duruma çakışma (collusion) adı verilmektedir.

Özeteleme fonksiyonlarının ingilizcesi olan Hash kelimesinin kökü arapçadan girmiş olan haşhaş kelimesi ile aynıdır. Ve insan üzerinde yapmış olduğu deformasyondan esinlenerek hash function'a giren bilgilere yapmış olduğu deformasyondan dolayı bu ismi almıştır.

Bazı özetleme fonksiyonları aşağıda listelenmiştir:

- [MD5 \(Message Digest 5\)](#)
- [SHA-1 \(Secure Hashing Algorithm\)](#)
- [HAVAL](#)

Özetleme fonksiyonlarının (hashing functions) kullanıldığı bazı güvenlik teşrifatları (protocols)

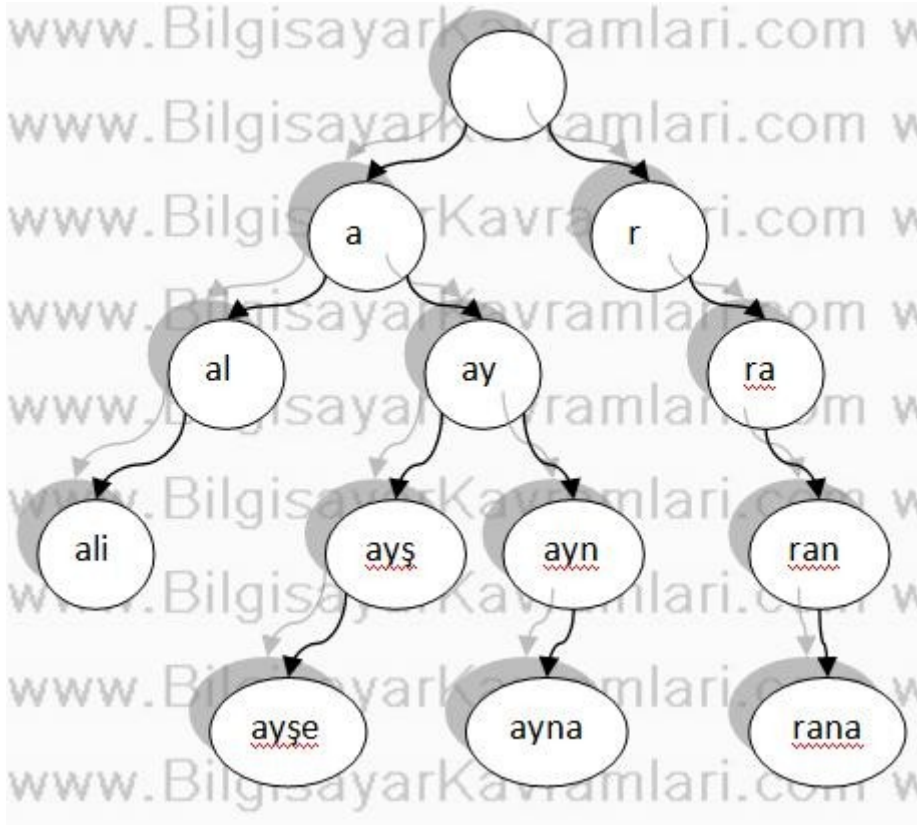
- [KERBEROS](#)
- [PY Şifrelemesi](#)
- [TEA \(Tiny Encryption Algorithm\)](#)
- [Zaman Etiketlemeli Yetkilendirme](#)
- [Document Management Systems](#)

Özetleme fonksiyonlarının (hashing functions) kullanıldığı bazı veri iletişim şekilleri

- [Doğrusal Sondalama \(Linear Probbing\)](#)
- [İkinci Dereceden Sondalama \(Quadratic Probbing\)](#)
- [Çift Özetleme \(Double Hashing\)](#)
- [Doğrusal Bölüm \(Linear Quotient\)](#)
- [LICH BLICH RLICH](#)
- [EISCH REISCH BEISCH](#)
- [CCI \(Computed Chaining Insertion\)](#)
- [SimHash \(Benzerlik Özetlemesi\)](#)

SORU 121: Trie (Metin Ağacı)

Metin ağaçları, her düğümün kendisinden sonra gelen harfi işaret ettiği ağaçlardır. Basitçe ağacın üzerine bir metin kodlanabilir ve bu metni veren ağacın üzerinde tek bir yol izlenebilir (deterministic). Durum aşağıdaki örnek üzerinde daha rahat anlaşılabilir:



Yukarıdaki ağaçta dikkat edilirse kök düğüm her zaman boş metni (string) ifade etmektedir. Bu boş metin hangi harf ile devam edilirse ilgili kolu takip eder ve gitmiş olduğu düğüm o ana kadar geçmiş olduğu kollarındaki harflerin birleştirilmiş halidir. Bir düğümünden bir harf taşıyan sadece bir kol çıkabilir.

Trie ağacının ismi **retrieval** kelimesinin ortasındaki 4 harften gelmektedir.

Metin ağaçlarının (trie), ikili arama ağaçlarına göre en önemli avantajları bir metni aramanın, metin boyutu kadar işlem gerektirmesidir. İkili arama ağaçlarında ise bu süre $\log n$ kadar varkit almaktadır. Buradaki n , ağaçtaki düğüm sayısıdır dolayısıyla ikili arama ağaçları, ağaçtaki bilgiye göre hızlı veya yavaş çalışırken, metin ağaçları, ağaçta ne kadar bilgi bulunduğundan bağımsız olarak çalışırlar.

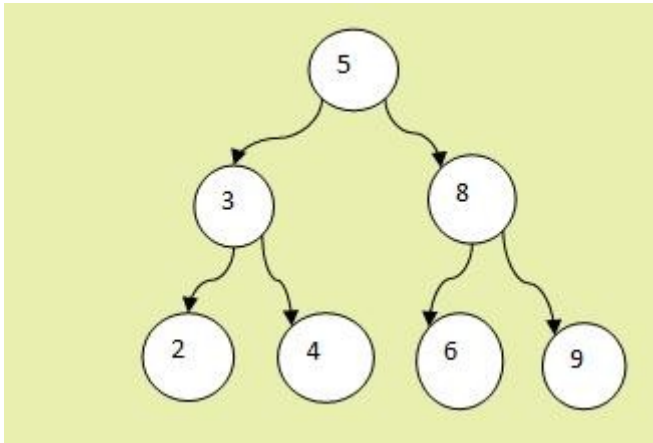
Metin ağaçları hafızayı da verimli kullanırlar çünkü bir metin ağacının en derin noktası, ağaç üzerindeki en uzun metin kadardır. İkili ağaçlarda ise bu derinlik eklenen düğüm sayısına göre en kötü ihtimalle düğüm sayısı kadar olabilmektedir.

Ayrıca metin ağaçları en uzun önek eşlemesi (longest prefix matching) gibi problemlerin çözümünde de avantaj sağlar.

SORU 122: İkili Arama Ağacı (Binary Search Tree)

İkili ağaçların (Binary Tree) özel bir hali olan ikili arama ağaçlarında, düğümlerde duran bilgilerin birbirine göre küçüklük büyüklük ilişkisi bulunmalıdır. Örneğin tam sayılardan(integer) oluşan veriler tutulacaksa bu verilerin aralarında küçük-büyük ilişkisi bulunmaktadır.

İkili arama ağacı, her düğümün solundaki koldan ulaşılacak bütün verilerin düğümün değerinden küçük, sağ kolundan ulaşılacak verilerin değerinin o düğümün değerinden büyük olmasını şart koşar.



Örneğin yukarıda bir ikili arama ağacı tasvir edilmiştir. Bu ağaçta dikkat edilecek olursa kökün solunda bulunan bütün sayılar kökten küçük ve sağında bulunan bütün sayılar kökten büyüktür.

İkili arama ağacına bu kurala uygun olarak ekleme çıkarma veya silme işlemleri yapılabilir. Ancak yapılan her işlemten sonra ikili arama ağacının yapısı bozulmamış olmalıdır.

İkili arama ağacında arama işlemi:

Yukarıda da anlatıldığı üzere ağacın üzerinde duran verilerin özel bir şekilde sıralı bulunması gerekir. Buna göre herhangi bir değeri arayan kişi ağacın kök değerine bakarak aşağıdaki 3 eylemden birisini yapar:

1. Aranan sayı kökteki değere eşittir -> Aranan değer bulunmuştur
2. Aranan sayı kökteki değerden küçüktür -> Kökün sol kolu takip edilir
3. Aranan sayı kökteki değerden büyüktür -> Kökün sağ kolu takip edilir

İkili arama ağaçlarının önemli bir özelliği ise öz çağırıcı(recursive) oluşudur. Buna göre bir ağacın sol kolu veya sağ kolu da birer ağaçtır. Bu özellikten faydalanarak örneğin C dilinde aşağıdaki şekilde bir arama fonksiyonu yazılabilir:

```
dugum * ara(dugum *kok, int deger){
    if(deger == kok->veri){
        return kok;
    }
    else if(deger > kok->veri ){
        return ara(kok->sag, deger);
    }
    else{
        return ara(kok->sol,deger);
    }
}
```

Yukarıdaki bu örnekte daha önce ikili ağaçlar konusunda tanımlanmış bir düğüm struct'ı kullanılmıştır. Koda dikkat edilecek olursa özçağırıcı bir yapı görülebilir. Yani fonksiyonun içerisinde kendisi çağırılmış ve yeni kök değeri olarak mevcut kökün sol veya sağ değeri verilmiştir. Arana değer her durumda değişmediği için alt düğümlere kadar aynı olarak indirilmiştir.

Yukarıdaki koddaki önemli bir eksiklik ise kodun ağaç üzerinde bir sayıyı bulamaması durumunun tasarlanmamış olmasıdır. Bu durumda basitçe NULL değer kontrolü ile aşılabilir. Buna göre sol veya sağa gitmeden önce sol veya sağda bir değer olduğuna bakılmalı şayet değer varsa gidilmelidir. Şayet değer yoksa ağaçta bu sayının bulunma ihtimali yoktur ve bu sayının olmadığını belirten bir ifade (örneğin NULL) döndürülmelidir. Kodun bu şekilde tashih edilmiş hali aşağıda verilmiştir:

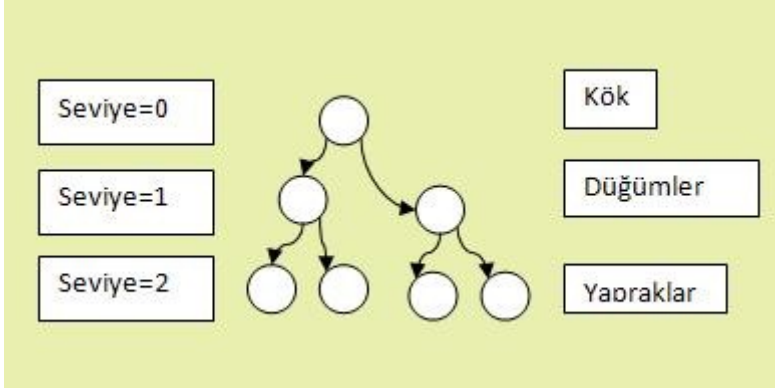
```
dugum * ara(dugum *kok, int deger){
    if(deger == kok->veri){
        return kok;
    }
    else if(deger > kok->veri ){
        if(kok->sag != NULL)
            return ara(kok->sag, deger);
        else
            return NULL;
    }
    else{
        if(kok->sol != NULL)
            return ara(kok->sol,deger);
        else
            return NULL;
    }
}
```

}

SORU 123: Ağaçlar (tree)

Bir [graf](#) şayet [bağlı grafsa](#) ve hiç [döngü](#) içermiyorsa bu grafa ağaç adı verilir.

Bilgisayar bilimlerinin önemli veri tutma yöntemlerinden birisi de ağaçlardır. Buna göre veriler bir ağaç yapısına benzer şekilde (kök gövde yapraklar) tutulur.



Örneğin yukarıdaki ağaç tasvirinde 7 düğümden (node) oluşan ve yapraklarında (leaf) 4 düğüm bulunan bir ağaç gösterilmiştir. Bu ağacın derinliği (depth) 2 dir ve her seviyenin(level) değeri yanında verilmiştir. Ağaçların 1 tane başlangıç düğümü bulunur ve bu başlangıç düğümüne kök(root) denilir.

Özel olarak yukarıdaki ağacın her düğümünden sadece ikişer alt düğüme bağlantı bulunduğu için bu ağaca ikili ağaç (binary tree) adı da verilebilir.

Bilgisayar mühendisliğinde sıkça kullanılan ağaçardan bazıları hakkında daha detaylı bilgi almak için üzerine tıklayabilirsiniz:

- [İkili ağaçlar \(Binary Tree\)](#)
- [İkili arama ağaçları \(Binary Search Tree\)](#)
- [Trie \(Metin Ağacı\)](#)
- [AVL Ağacı \(AVL Tree\)](#)
- [Yığıt Ağacı \(Heap\)](#)
- [Dikişli Ağaçlar \(Threaded Trees\)](#)
- [B Ağacı \(B Tree\)](#)
- [Patricia Ağacı \(Patricia Tree\)](#)
- [Parçalam Ağacı \(Parse Tree\)](#)
- [Mini Max Ağaçları \(Minimax Trees\)](#)
- [Petri Ağları \(Petri Networks\)](#)
- [Asgari Tarama Ağacı \(minimum spanning tree\)](#)

SORU 124: STTL (A standard timetabling language (standart bir zaman çizelgeleme dili))

STTL -> A standard timetabling language (standart bir zaman çizelgeleme dili)

Bu dil ilk olarak Jeff Kingston tarafından zaman çizelgeleme problemlerine bir girdi standardı elde etmek için önerilmiştir. Dilin çıkışında kullanılan problem bir lisede haftalık ders programlarının düzgün yerleştirilmesini hedefliyordu. Nesne yönelimli (Object oriented) bir dil olarak önerilen STTL dilinde varlıklar arası miras ilişkisi bulunmaktadır örneğin:

```
class TEACHER inherit RESOURCE
max_class_load:INTEGER
unavailable_times:SET[TIME]
cats:SET[TEACHER_CATEGORY]
functions
defects:SEQ[DEFECT]=
(
defect_check(TeacherClash, clashes) +
defect_check(TeacherOverload, overload) +
defect_check(TeacherDayOverload, day_overload)
)
end
```

Yukarıdaki STTL alıntısında Öğretmen (TEACHER) ile kaynak (RESOURCE) arasında miras ilişkisi kurulmuştur. Yukarıdaki tanımlamadan sonra aşağıdaki varlığın dilde tanımlanması mümkündür:

```
TEACHER("Knott", 22, {}, {English})
```

Buna göre Knott isminde bir öğretmen haftalık azami 22 saatlik ders yükü ve herhangi bir ders engeli bulunmadan English kategorisinin altında tanımlanmıştır.

STTL dilinin diğer özelliklerinden birisi de kaynak tanımlamasıdır. Örneğin dersin yapılacağı sınıfın, ders için kullanılacak malzemelerin birer kaynak olarak tanımlanması ve dolayısıyla zamanlamaya dahil edilmesi mümkündür.

```
r01= ROOM("r01", {DramaRoom})
r02 = ROOM("r02", {ScienceLab})
```

```
Yr7C = STUDENT_GROUP("Yr7C")
Yr7K = STUDENT_GROUP("Yr7K")
```

Örneğin yukarıda sınıf ve öğrenci grupları birer kaynak olarak tanımlanmıştır. Zaman modellemesi içinde alternatifler sunan STTL dilinde örneğin aşağıdaki şekilde bir zaman tanımlaması yapmak mümkündür:

```
TIME({2}, INTERVAL(1, 14), {Mon}, {INTERVAL(t(9, 30), t(12, 30))})
```

Yukarıdaki tanımlamada, 2. sömestr için 1den 14. haftalara kadar Pazartesi günleri saat 9.30 ile 12.30 arası kastedilmiştir. Dolayısıyla tekrarlı bir olayı ifade etmektedir.

Sonuç olarak zaman çizelgelemesine göre hazırlanmış olan bu dilde kesin tarihlerin belirlenmesi zaman aralıklarının ifade edilmesi mümkün olmakla beraber bağıl zaman ifadeleri ve zamanın belirsiz olması durumları düşünülmemiştir. Örneğin A ile B olayları olsun ve tek bilinenin A olayının B olayından sonra olması durumu olduğunda bunu göstermek bu dil ile mümkün değildir.

SORU 125: Feistel Şifreleme (Feistel Cipher, Fesitel Ağı, Feistel Network)

Adını, alman şifreleme uzmanı Horst Feistel'den alan şifreleme methodu, [blok şifreleme](#) kullanılmaktadır ve güncel pek çok blok şifreleme yöntemine temel teşkil etmektedir.

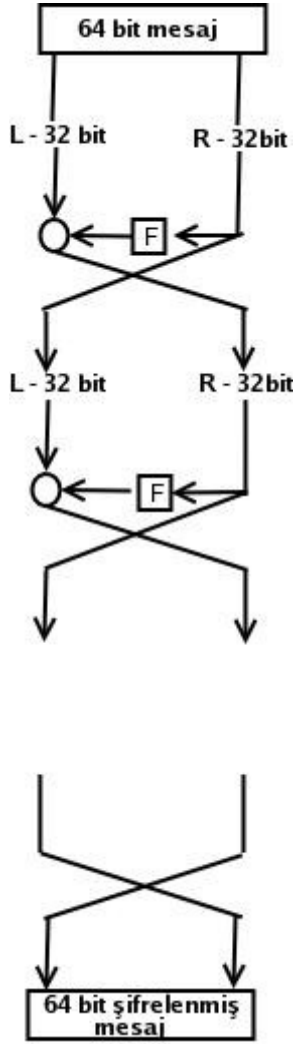
Feistel şifrelemesinin en büyük avantajı şifreleme (encryption) ve açma (decryption) işlemlerinin çok benzer olması ve hatta bazı durumlarda aynı olmasıdır. Bu yapıda şifreleme ve açma arasındaki tek fark, anahtar sırasıdır ve bu durum kodlama çalışmasını ve yapılacak işi neredeyse yarıya indirmektedir.

Bir feistel ağını temel olarak aşağıdaki 3 işlemi çeşitli sıralar ile çeşitli miktarlarda tekrar etmektedir:

1. Verinin bitlerinin yerinin değiştirilmesi (genelde [permütasyon şifrelemesi](#) olarak da bilinir)
2. Basit doğrusal olmayan fonksiyon icrası ([Yerine koyma şifrelemesi](#) olarak da bilinir)
3. Doğrusal karıştırma (yahut işlemi ([özel veya, XOR](#)) olarak da bilinir)

Yukarıdaki bu yapıya göre feistel şifrelemelerinin temel dayanak noktası verinin bitleri arasında permütasyon alınması ve yerine koyma yapılmasıdır. Bu durum Shannon Teorisindeki karıştırma ve dağıtma (confusion and diffusion) ilkesi olarak binir ve permütasyon işlemi dağıtma (diffusion), yerine koyma işlemi ise karıştırma (confusion) olarak kabul edilir.

Aşağıda temel bir fiestal ağının çizimi verilmiştir:



Yukarıdaki şekilde de görüldüğü üzere giriş metni öncelikle iki parçaya ayrılır ve bu parçalar farklı sıralarda F fonksiyonuna konularak özel veya XOR ile birleştirilir. Fiestel ağını kullanan sistemlerin temel farkı kullandıkları F fonksiyonlarıdır. Bu sistemlerden bazıları:

- [DES \(Digital Encryption Standard\)](#)
- [Blowfish](#)

SORU 126: Sıra (Queue)

Sıra, basit veri yapılarından birisidir. Buna göre bir sıraya ilk giren ilk çıkar (FIFO , first in first out fifo). Bazı kaynaklarda kuyruk kelimesi de kullanılır. Basitçe bir gişe önündeki bilet kuyruğu veya bilet sırası olarak düşünülebilir. Bu yapının olmazsa olmaz iki adet fonksiyonu bulunur:

enqueue(veri) veya push(veri) -> verilen veriyi sıraya koyar

deque() veya pop() -> sıradan bir veri çıkararak sıranın boyutunu azaltır ve çıkarılan veriyi çağrıldığı yere döndürür.

Komutlar	Sıranın durumu
----------	----------------

enqueue(10)	10
enqueue(20)	10 -> 20
enqueue(30)	10 -> 20 -> 30
dequeue()	20 -> 30
dequeue()	30
dequeue()	Boş

Yukarıda bir sıra üzerinde yapılan ekleme ve çıkarma işlemleri gösterilmiştir. Yukarıda sıranın durumu olarak gösterilen yapıda bir bağlı liste üzerinden verilerin nasıl eklenip çıkarıldığı tasvir edilmiştir. Bir sıra için [bağlı liste](#) kullanılabileceği gibi [dizi \(Array\)](#) de kullanılabilir. Ayrıca yukarıdaki [bağlı listede](#), listenin sağına eklenerek solundan çıkarma işlemi yapılmıştır. Bu işlemin tam tersi olan soluna yeni verileri ekleyerek sağından çıkarma işlemi de yapılabilir.

Dizi kullanıldığında karşılaşılan problem dizinin boyutunun sınırlı olmasıdır. Bu durumda dizideki verilerin dizi boyutunu aşınca daha büyük başka bir diziye taşınması gerekir. Ayrıca dizinin belirli bir elemanından sayılar dequeue yapılıyorsa dizideki elemanların dequeue yapıldıkça kaydırılması da gerekir.

Aşağıda [bağlı liste \(linked list\)](#) ve [dizi \(Array\)](#) kullanılarak yazılmış iki adet kod örneği verilmiştir:

- [Bağlı liste \(linked list\) kullanarak sıra \(queue\) örnek kodu \(C dilinde\)](#)
- [Dizi \(Array\) kullanarak sıra \(array-based queue\) örnek kodu \(C dilinde\)](#)
- [Bağlı liste \(linked list\) kullanarak sıra \(queue\) örnek kodu \(C++ Dilinde\)](#)

Sıra (Queue) içerisine veri ekleme fonksiyonu genel olarak enqueue(sıraya koymak, sıralamak) ve sıradaki bir elemanın sıradan alınmasına ise dequeue (sıradan almak, çıkarmak) ismi verilmektedir.

SORU 127: MathML (Matematiksel İşaretleme Dili, Mathematical Markup Language)

MathML dili, genellikle internet üzerinde matematiksel formül ve sembollerin ifâde edilmesi için geliştirilmiş dildir. Dil W3C tarafından geliştirilmiş ve halen gelişmelere göre eklentiler yapılmaktadır.

Örneğin ikinci dereceden çok terimlilerin (polynom) köklerini bulmak için kullanılan delta değerini ele alalım:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Bu denklemin MathML ile gösterimi aşağıdaki şekildedir:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
```

```
<mi>x</mi>
```

```
<mo>=</mo>
```



```

<mfrac>
  <mrow>
    <mrow>
      <mo>-</mo>
      <mi>b</mi>
    </mrow>
    <mo>&PlusMinus;</mo>
    <msqrt>
      <msup>
        <mi>b</mi>
        <mn>2</mn>
      </msup>
      <mo>-</mo>
      <mrow>
        <mn>4</mn>
        <mo>&InvisibleTimes;</mo>
        <mi>a</mi>
        <mo>&InvisibleTimes;</mo>
        <mi>c</mi>
      </mrow>
    </msqrt>
  </mrow>
  <mrow>
    <mn>2</mn>
    <mo>&InvisibleTimes;</mo>
    <mi>a</mi>
  </mrow>
</mfrac>
</math>

```

Yukarıda da görüldüğü üzere MathML dili XML'in özel bir halidir. Bu dilde bulunan her elemanın anlamı ve kullanım detayları için W3C tarafından hazırlanan siteye başvurulabilir <http://www.w3.org/Math/testsuite/>

SORU 128: Kerckhoff Prensibi

Veri güvenliğinde şifreleme yöntemleri için ortaya atılmış bir prensiptir. Bu prensibe göre bir şifreleme sisteminin aşağıdaki özellikleri bulunmalıdır.

1. Sistemin tamamı veya bir parçası matematiksel olarak geri açılmaz olmalıdır.
2. Sistemin gizli tutulması gerekmemelidir. Sistemin çalışma mantığı düşmanın eline geçebilir.
3. Yazılı notlar kullanılmadan sistemde kullanılan anahtarın değiştirilmesi mümkün olmalı ve tarafların talebine göre değiştirilebilir olmalıdır.
4. Telgraf iletişimine uygun olmalıdır.
5. Sistem taşınabilir olmalı ve sistemin çalışması için birkaç kişinin bir araya gelmesi gerekmemelidir.
6. Sistemin kullanımı basit olmalı uzun eğitimler ve kurallar içermemelidir.

SORU 129: Bilgi, Veri, Mâlûmat, İrfan (Knowledge, Data, Information, Wisdom)

İngilizce'de terim olarak yerleşmiş ve bilgisayar bilimleri için hayati öneme sahip kelimeleri Türkçede çoğu zaman sadece bilgi ile karşılayan tercümeler bulunur. 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 yaran şeyken irfan en değerli ve 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 hızın artırılması 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 bilgeliğin veriye kadar giden yol belirsiz(nondeterministic) bir yoldur. Bu yolun tersi de ne yazık ki belirsizdir çünkü hız

verisi şöförlik 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 130: Normal Şekil (Canonical Form)

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 131: Bilgi Çıkarımı (Information Extraction)

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 132: Belirsiz Çokterimli Tam (NP-Complete, Nondeterministic Polynomial Complete)

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 yararlanır.

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. En meşhur problemlerden bazıları: Altküme toplam problemi RSA Merkle-Hellman

SORU 133: Merkle-Hellman Şifreleme (Merkle-Hellman Encryption)

Merkle-Hellman bir açık anahtar şifreleme yöntemidir. Bu yöntemde göre bir umumî anahtar bir de hususî anahtar bulunması gerekir. RSA'den farklı olarak şifreleme işlemi tek yönlü çalışır. Yani umumî anahtar sadece şifreleme (encryption) hususî anahtar ise sadece açma (decryption) işlemleri için kullanılabilir.

Anahtar **üretme** **safhâsı:**
Pozitif doğal sayılardan oluşan hızla aratan (superincreasing) bir seri (sequence) alınır. Bu seriye $S = (s_1, s_2, \dots, s_n)$ diyelim.

Bu serideki sayıların toplamından daha büyük bir p sayısı alalım. Rasgele bir r sayısı alalım ancak bu r sayısı p sayısı ile aralarında asal olsun. Yukarıda bulunan bu sayıların tamamı (p ve r sayıları) ile S serisi hususî şifre olarak saklanacaktır.

Şimdi umumî şifre hesabı için:

$B = (B_1, B_2, \dots, B_n)$ sayılarını hesaplayabiliriz. Bu serinin hesabında: $B_i = r \cdot S_i \mod p$ formülü kullanılır ve bütün seri S serisine bağlı olarak hesaplanır. Sonuçta elde edilen B serisi umumî anahtar olarak kullanılır.

Şifreleme **safhâsı:**
n-bitlik bir mesajı şifrelerken (bu mesaj a olsun) her bit ayrı ayrı ele alınır. Buna göre: $a = (a_1, a_2, \dots, a_n)$ olarak düşünölsün. Buradaki her a terimi mesajın ilgili bitidir (örneğin a_{13} ,

mesajın 13. biti olacaktır) ve her bit(ikil) 1 veya 0 olabilmektedir. Yani a serisinin ilgili elemanı 0 veya 1 olabilir.

$$c = \sum_{i=1}^n a_i b_i$$

olarak serinin toplamını veren c değeri hesaplanır ve şifreli metin olarak karşı tarafa yollanır.

Açma

safhâsı:

Daha önceden toplam değeri içeren c değeri aşağıdaki şekilde hesaplanmıştır:

$$c = \sum_{i=1}^n a_i b_i$$

Bu mesaja saldırmak isteyen kişinin uğraşması gereken zorluk alt küme toplam problemi zorluğudur çünkü saldıran kişi B değerlerini bilmemektedir ve tahmin etmesinin tek yolu NP-Complete olarak saldırmaktır.

Ancak hususî anahtarlar olan B r ve p değerleri biliniyorsa bu şifrenin açılması oldukça basittir. Açmak için r ve p değerleri kullanılarak modüler tersi olan f değeri bulunur. (Bu işlem örneğin uzatılmış öklit algoritması (extended euclid algorithm) ile kolayca yapılabilir)

Sonuç olarak her a değeri (c / Si) x f mod p formülü ile hesaplanabilir.

Yani problemde her sayı grubu farklı bir çarpanla çarpılmakta ve daha önceki çarpanlarla karıştırılmaması için bu çarpanlar serisi hızlı artan bir seri seçilmektedir.

Merkle-Hellman yönteminin dayandığı zorluk alt küme toplam problemi (subset sum problem)'dir (knapsack(torba) algoritmasının özel bir halidir). Problem genel olarak bir Belirsiz Çokterimli Tam Problemi (NP-Complete) yapısındadır. Bu problem tipi tanımı gereği çözüm zamanı önceden kestirilemezdir. Ancak Merkle-Hellman şifrelemesinde şayet her üretilen sayı kümesi, kendinden önceki kümelerden daha büyükse (superincreasing, hızlı artan) bu durumda problemin çözümü çokterimli zamanda (polynomial time) kestirilebilir ve bu kestirme işlemi için açgözlü yaklaşımı (greedy approach) kullanılabilir.

SORU 134: Açgözlü Yaklaşımı (Greedy Approach)

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 135: Torba Problemi (knapsack problem)

Torba problemi basitçe bir torbanın içerisine en fazla eşyanın yerleştirilmesini hedefler. Problem hırsız örneğinden daha iyi anlaşılabilir. Buna göre bir hırsız çantasına ağırlıkça en az, pâyâca en çok eşyayı doldurmak ister. Bu durumda her eşyanın

Aşağıdaki her özel problem durumu için eşyaların pahası p_i olsun ve her eşyanın ağırlığı a_i olsun. Çantanın taşıyabileceği azamî kapasite de c_i olarak tanımlansın.

0-1 Torbla problemi: Bu problem tipinde eşyalar ya tamamen alınır ya da tamamen bırakılır. Alınacak olan eşyanın bir kısmını almak mümkün değildir. Dolayısıyla bir eşyanın alınıp alınmamasını x_i ile gösterecek olursak problem aşağıdaki şekilde modellenebilir:

$$\sum_{i=0}^n p_i x_i, \text{ değeri mümkün olduğunca büyük olmalıdır.}$$

$$\sum_{i=0}^n a_i x_i \leq c_i, \text{ olmalı ve burada } x_i \in \{0,1\} \text{ olarak alınmalıdır.}$$

Bu modelde görüldüğü üzere, x_i değeri, 0 veya 1 olabilmektedir. 0 olması durumunda i elemanından alınmıyor 1 olduğunda ise i elemanının tamamı alınıyor demektir.

Örnek

Elimizde aşağıdaki eşyalar bulunsun:

1. saat, 0.2 kg 100TL
2. fotoğraf makinesi 0.5kg 300TL
3. kamera 0.7kg 700TL
4. cep telefonu 0.3kg 500TL
5. anahtar 0.1kg 10TL

Şimdi problemimiz, elimizdeki bir torbaya (örneğin 1kg kapasiteli) yukarıdaki eşyaları en fazla para edicek şekilde yerleştirmek istememiz.

bu durumda $c_i=1$ kg olarak tanımlamış oluyoruz.

Çözümde ise $i = \{ 3,4 \}$ kümesi alınıyor. Yani kamera ve cep telefonunu aldığımızda 1200 TL ile en pahalı ve torbamıza sığan kümeyi almış oluyoruz.

Sınırlı Torba Problemi (Bounded Knapsack Problem): Bu problem tipinde ise her eşyadan alınabilecek miktarda sınır vardır. Bu problemin modeli aşağıda verilmiştir:

$$\sum_{i=0}^n p_i x_i, \text{ değeri mümkün olduğunca büyük olmalıdır.}$$

$$\sum_{i=0}^n a_i x_i \leq c_i, \text{ olmalı ve burada } 0 \leq x_i \leq b_i \text{ olarak alınmalıdır.}$$

Sınırsız Torba Problemi (Unbounded Knapsack Problem): Bir önceki sınırsız torba probleminden farklı olarak bu problemde alınabilecek malzemelerin herhangi bir sınırı bulunmamaktadır.

Örnek

Bu problem tipinde ise örneğimizi değiştirmemiz gerekiyor.

1. elma 2TL , 4kg
2. armut 5TL, 3kg
3. kiraz 4TL, 6kg
4. portakal 3.5TL, 5kg

şeklinde verilen ürünlerden örneğin 10kg'lık bir fileye en pahalı nasıl bir seçim yapılmalıdır sorusu sorulabilir.

Bu durumda her üründen değişik miktarlarda var. Örneğin en fazla 4kg elma alabiliyoruz bu değer modelimizdeki b_i ile gösterilen ve bir üründen en fazla alınabilen değerdir. Toplamda en fazla alabileceğimiz değer ise hala c_i 'dir. Bu problem tipinde her üründen alınan miktar ise değişmektedir ve bu miktar x_i ile gösterilir.

Örneğin yukarıdaki problemin çözümünde 1kg portakal fileye konulacaksa, bu durumda $x_4 = 1$, $b_4 = 5$, $p_4 = 3.5$ olarak alınmalıdır.

Torba problemi, veri güvenliği (cryptography) konusunda genellikle alt küme toplam problemi olarak kullanılır ve örneğin merkle-hellman yönteminde bu özelliğinden faydalanılır.

Torba problemi aynı zamanda dinamik programlama (dynamic programming) için de oldukça elverişli bir problemdir ve aşağıdaki yaklaşımla çözülebilir:

Torba probleminin bir diğer çözüm önerisi açgözlü yaklaşımı (greedy approach) iledir. Bu çözüme göre alınabilecek en yüksek meblağdaki eşya torbaya doldurularak her seferinde

kalan yer için mümkün olan en değerli eşya aranır. En sonunda yer kalmayana veya kalan yere bir eşya konulamayana kadar bu şekilde gidilir.

Açgözlü yaklaşımının (greedy approach) bir dez avantajı her zaman en verimli sonucu üretememesidir.

SORU 136: Açık Anahtarlı Şifreleme (Public Key Cryptography)

Asimetrik şifreleme yöntemi olarak da bilinen bu yöntemde kullanıcıların 2 adet şifresi bulunur. Bu şifrelerden birisi herkese açık (umûmî, public key) diğeri ise gizli (private, husûsî) şifredir. Çalışma mantığına göre umumî olan şifre herkese rahatça dağıtılabilir ve bu şifreden hususî olan şifreye ulaşmanın matematiksel bir yolu bulunmamalıdır. Ayrıca umumî şifre ile şifrelenmiş mesajın hususî şifre ile açılmasının bir yolunun bulunması gerekir.

Açık anahtar şifrelemesinin en önemli iki kullanım alanı şunlardır:

Açık anahtar şifrelemesi (public key encryption): Buradaki amaç bir kişiye yollanacak olan mesajın umumî şifre ile şifrelenerek yollanması ve ancak alıcı tarafın hususî şifresi ile açılabilmesidir. Bu sayede örneğin A kişisine mesaj yollayacak olan herkes, A kişinin umumî şifresini alarak bu şifreye göre şifreleme yapar. Sonuçta mesaj A kişisine yollanır ve A kişisi bu mesajı sadece kendisinin bildiği hususî şifresi ile açar.

Sayısal imsa (Digital Signature): Buradaki amaç bir mesajın yanlış kişilere ulaşmasını engellemek ve tam olarak kimden geldiğinden emin olmaktır. Gönderen tarafın kendi hususî şifresi ile şifrelediği mesajı herkes yine gönderen kişinin umumî şifresi ile açarak bu kişiden geldiğinden emin olabilir.

Açık anahtar şifrelemesi gerçek hayattaki bir posta kutusu örneğine benzetilebilir. Buna göre ilgili kişinin adresini bilen herkes bu kişinin posta kutusuna bir mektup bırakabilir. Ancak bu posta kutusunu sadece kutunun sahibi ve dolayısıyla kutuyu açacak anahtarı olan kişi açarak mektupları alabilir.

Bazı uygulamaları:

- [Diffie-Hellman](#)
- [Merkle-Hellman](#)
- [RSA](#)
- [El Gamal](#)

SORU 137: Somut (müşahhas) isim (concrete noun)

Fiziksel olarak var olan nesnelere verilen isimdir. Örneğin elma, ağaç, araba veya insan gibi. Soyut (mücerret) isimlerin tersi niteliğindedir. Daha fazla bilgi için isim tanımına bakabilirsiniz.