İçindekiler

[**AUTOMATA (OTOMATLAR, ÖZDEVİNİRLER)** 4](#_Toc377922307)

[**SORU-1: Emil Post Makinesi hakkında bilgi veriniz.** 4](#_Toc377922308)

[**SORU-2: Post Karşılık Problemi (Post Correspondence Problem) hakkında bilgi veriniz.** 5](#_Toc377922309)

[**SORU-3: Mealy ve Moore Makineleri (Mealy and Moore Machines) hakkında bilgi veriniz.** 7](#_Toc377922310)

[**SORU-4: Eşit a ve b bulunduran Turing makinesi örneği veriniz.** 10](#_Toc377922311)

[**SORU-5: Von Neumann Makinesi hakkında bilgi veriniz.** 14](#_Toc377922312)

[**SORU-6: Ve/Veya bağlacı normal şeklleri (Conjunction / Disjunction Normal Form)** 15](#_Toc377922313)

[**SORU-7: CYK Parçalama** 16](#_Toc377922314)

[**SORU-8: DFA Metin Arama Algoritması (DFA Text Search)** 22](#_Toc377922315)

[**SORU-9: Push Down Automata** 26](#_Toc377922316)

[**SORU-10: Özyineli Geçiş Ağları (Reursive Transition Networks)** 33](#_Toc377922317)

[**SORU-11: Turing Makinesi (Turing Machine)** 35](#_Toc377922318)

[**SORU-12: Özyineli Sayılabilir Diller (Recursively Enumerable Languages)** 42](#_Toc377922319)

[**SORU-13: Chomsky Hiyerarşisi ( Chomsky Hierarchy )** 43](#_Toc377922320)

[**SORU-14: Muntazam Diller (Formal Languages)** 45](#_Toc377922321)

[**SORU-15: Chomsky Normal Şekili (Chomsky Normal Form)** 48](#_Toc377922322)

[**SORU-16: Sözdizim (Syntax)** 51](#_Toc377922323)

[**SORU-17: Markof Modeli (Markov Model)** 54](#_Toc377922324)

[**SORU-18: Hamilton Yolu (Hamiltonian Path,hamiltonian circuit)** 57](#_Toc377922325)

[**SORU-19: Augmented Transition Network (ATN, Uzatılmış Geçiş Ağı)** 59](#_Toc377922326)

[**SORU-20: Knuth Morris Prat Algoritması (KMP Algorithm)** 61](#_Toc377922327)

[**SORU-21: Atomluluk (Atomicity)** 62](#_Toc377922328)

[**SORU-22: Tehlike (Hazard)** 63](#_Toc377922329)

[**SORU-23: İçerik Bağımsız Gramerler için Pompalama Önsavı (Pumping Lemma for Context Free Grammers)** 63](#_Toc377922330)

[**SORU-24: Düzenli İfadelerde Pompalama Önsavı (Pumping Lemma for Regular Expressions)** 65](#_Toc377922331)

[**SORU-25: Pompalama Önsavı (Pumping Lemma)** 68](#_Toc377922332)

[**SORU-26: İçerikten Bağımsız Gramer (context free grammer, CFG)** 68](#_Toc377922333)

[**SORU-27: İçerikten bağımsız dil (Context Free Language, CFL)** 69](#_Toc377922334)

[**SORU-28: EBNF (Uzatılmış BNF, Extended Backus Normal Form)** 70](#_Toc377922335)

[**SORU-29: YACC** 72](#_Toc377922336)

[**SORU-30: LEX** 74](#_Toc377922337)

[**SORU-31: Parçalama Ağacı (Parse Tree)** 75](#_Toc377922338)

[**SORU-32: Backus Normal Form (BNF)** 76](#_Toc377922339)

[**SORU-33: NFA'den DFA'e çevirim (Converting NFA to DFA)** 77](#_Toc377922340)

[**SORU-34: Belirsiz Sonlu Otomat (Nondeterministic Finite Automat, NFA)** 82](#_Toc377922341)

[**SORU-35: Belirli Sonlu Otomat (Deterministic Finite Automat, DFA)** 87](#_Toc377922342)

[**SORU-36: Kuantum İşleme (Quantum Computing)** 89](#_Toc377922343)

[**SORU-37: Nöbetçi (Sentinel)** 90](#_Toc377922344)

[**SORU-38: Sonlu Ototmatlar (Finite Automaton)** 90](#_Toc377922345)

[**SORU-39: Bağımsız düğümler (Anti Clique, Independent Set)** 91](#_Toc377922346)

[**SORU-40: Klik (clique)** 91](#_Toc377922347)

[**SORU-41: İstikra ile ispat (Tüme varım, Proof by Induction)** 92](#_Toc377922348)

[**SORU-42: Binaen Burhan (İnşâa ile İspat , Proof by Construction, Binaenaleyh)** 93](#_Toc377922349)

[**SORU-43: k-düzenli graf ( k-regular graph)** 94](#_Toc377922350)

[**SORU-44: Nazariye (Teori, Kuram, Theorem)** 94](#_Toc377922351)

[**SORU-45: Alt Dizgi (Substring)** 94](#_Toc377922352)

[**SORU-46: Dizgi (String)** 94](#_Toc377922353)

[**SORU-47: Alfabe (Abece, Alphabet)** 98](#_Toc377922354)

[**SORU-48: Sembol (Harf, İşaret, Symbol)** 98](#_Toc377922355)

[**SORU-49: Güçlü Bağlı Graf (Strongly Connected Graph)** 99](#_Toc377922356)

[**SORU-50: Basit Döngü (Simple Cycle)** 99](#_Toc377922357)

[**SORU-51: Bağlı graf (conected graph)** 99](#_Toc377922358)

[**SORU-52: Döngü (Cycle)** 99](#_Toc377922359)

[**SORU-53: Altgraf (Subgraph)** 99](#_Toc377922360)

[**SORU-54: Yol (Path)** 100](#_Toc377922361)

[**SORU-55: Yönlü Graflar (Directed Graphs)** 100](#_Toc377922362)

[**SORU-56: Yönsüz graflar (undirected graphs)** 100](#_Toc377922363)

[**SORU-57: Graf (Şekil, Graph)** 101](#_Toc377922364)

[**SORU-58: Kenar (Edge)** 101](#_Toc377922365)

[**SORU-59: Düğüm (Node)** 101](#_Toc377922366)

[**SORU-60: DTD (Document Type Definition, Döküman Tip Tanımı)** 101](#_Toc377922367)

[**SORU-61: Özetleme Fonksiyonları (Hash Function)** 104](#_Toc377922368)

[**SORU-62: Ağaçlarda Dengeleme (Rotation, Balancing)** 104](#_Toc377922369)

[**SORU-63: Trie (Metin Ağacı)** 107](#_Toc377922370)

[**SORU-64: İkili Ağaç (Binary Tree)** 108](#_Toc377922371)

[**SORU-65: Ağaçlar (tree)** 110](#_Toc377922372)

[**SORU-66: OWL Time (OWL Zaman, Web Varlıkbilim Dili Zaman)** 110](#_Toc377922373)

[**SORU-67: TTML (Time Tabling Markup Language, Zaman Çizelgeleme İşaretleme Dili)** 116](#_Toc377922374)

[**SORU-68: Arc Constraint (Kiriş Şartı)** 116](#_Toc377922375)

[**SORU-69: 4 vezir problemi (4 queen problem)** 117](#_Toc377922376)

[**SORU-70: Ortak Bölenlerin En Büyüğü (OBEB, GCD, Greatest Common Divisor)** 117](#_Toc377922377)

[**SORU-71: Belirsiz Çokterimli Tam (NP-Complete, Nondeterministic Polynomial Complete)** 119](#_Toc377922378)

[**SORU-72: Sonlu Durum Makinası (Finite State Machine, Finite State Automaton)** 119](#_Toc377922379)

[**SORU-73: Kruskal Asgari Tarama Ağacı Algoritması** 120](#_Toc377922380)

[**SORU-74: Prim asgari tarama ağacı Algoritması** 128](#_Toc377922381)

[**SORU-75: asgari tarama ağacı (en kısa örten ağaç, minimum spanning tree)** 136](#_Toc377922382)

[**SORU-76: fonksiyon göstericileri (function pointer)** 136](#_Toc377922383)

[**SORU-77: otomat yönelimli programlama (automata based programming)** 138](#_Toc377922384)

[**SORU-78: fonksiyonel programlama (functional programming)** 139](#_Toc377922385)

[**SORU-79: En uzun Ortak Küme (longest common subsequence, Lcs)** 140](#_Toc377922386)

[**SORU-80: Dinamik Programlama (Dynamic programming)** 141](#_Toc377922387)

[**SORU-81: parçala fethet yöntemi (divide and conquer)** 142](#_Toc377922388)

[**SORU-82: Alt küme toplamı problemi (subset sum problem)** 142](#_Toc377922389)

[**SORU-83: Pointer (Gösterici) ve Diziler (Arrays)** 142](#_Toc377922390)

**AUTOMATA (OTOMATLAR, ÖZDEVİNİRLER)**

**SORU-1: Emil Post Makinesi hakkında bilgi veriniz.**

Bilgisayar bilimlerinde özellikle özdevinirliler 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 işaretsiz kabul edilmektedir. Yani başlangıç durumunda makinemizin hafızasında bir bilgi olmasını istiyorsak, bu bilgiyi kutulara işaretliyoruz. Geri kalan kutular ise işaretsiz 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 işaretsiz 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 Oi [3] şeklindeki yazım, i. adımda makinemizin sağdaki kutuya geçeceği şeklinde okunabilir. Ayrıca makinemizin yukarıdaki işlemlerden 5ncisini işletmesi durumunda sonraki adımlarda çatallanma olur. Yani okunan değerin işaretli veya işaretsiz 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-2: Post Karşılık Problemi (Post Correspondence Problem) hakkında bilgi veriniz.**

Emil Post tarafından 1946 yılında ortaya atılan ve belirsiz karar problemi olarak sınıflandırılabilecek olan (undecidable decision problem) problemin ismidir. Literatürde kısaca PCP olarak da geçmektedir. Bu problem, yine Emil Post tarafından geliştirilen, Post Makinesi (post machine) olarak bilinen ve Turing makinesinin (Turing Machine) bir benzeri olan makinenin geliştirilmesini sağlamıştır. Problem, durma problemine göre (halting problem) daha basit bir yapıdadır bu yüzden belirsiz karar problemlerinin ispatlanmasında daha çok tercih edilir.

Problem aşağıdaki şekilde tanımlanabilir.

A bir alfabe olmak üzere, bu alfabeden tanımlı olan n elemanlı iki küme ele alalım:

X = {x1,x2, … xn }  
Y = {y1,y2, … yn}

Bu iki küme üzerinden üretilen iki kelimenin eşit olması durumunu arıyoruz.

Örneğin aşağıdaki kümeleri ele alalım:

X = { sad , sek , er, i}  
Y = { sa , se, ker , di }

Yukarıdaki örnek kümeler için böyle bir eşitlik bulunabilir:

sad + i + sek + er = sadiseker = sa + di + se + ker

Yukarıdaki eşitliğin sol tarafı X kümesinde, sağ tarafı ise Y kümesinden üretilmiştir.

Bazı problemler çözümsüz, bazı problemler ise birden fazla çözüme sahip olabilir. Örneğin aşağıdaki kümeleri ele alalım:

X= {a,ab,bba}  
Y= { baa, aa, bb}

Çözüm aşağıdaki şekilde olabilir:

bba + ab + bba + a = bbaabbbaa = bb + aa + bb + baa

Ancak bu tek çözüm değildir. Örneğin aşağıdaki de bir çözüm üretebilir:

bba + ab + bba + a + bba + ab + bba + a = bbaabbbaabbaabbbaa = bb + aa + bb + baa + bb + aa + bb + baa

Yukarıdaki ikinci çözüm, bir önceki çözümün tekrarı niteliğindedir. Yani üretilen eşitlik dizilimi bir önceki çözümün iki kere tekrarlanması ile oluşturulmuştur. Bu işlem istenildiği kadar tekrar edilebilir.

Örneğin yukarıdaki kümelerin ilk elemanları çıkarılarak aşağıdaki kümeler oluşturulsaydı

X= {ab,bba}  
Y= { aa, bb}

Bu kümelerden elde edilen bir çözüm bulunamazdı.

**SORU-3: Mealy ve Moore Makineleri (Mealy and Moore Machines) hakkında bilgi veriniz.**

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.

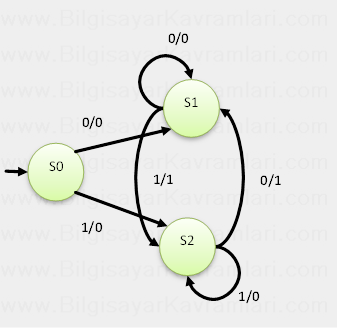
Mealy makinelerinde ise giriş ve çıkış değerleri kenarlar (edges) üzerinde aralarına bir taksim işareti (slash) konularak gösterilir. Örneğin 1/0 gösterimi, girişin 1 ve çıktının 0 olduğunu ifade eder.

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 |

Tablo-2.1: Girdi-çıktı

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 aldığında beklenen çıktıyı vermeli. Ayrıca makine sürekli olarak çalışmaya devam edecek. Örneğin 101011 şeklinde bir veri akışı sağlanması durumunda, sonuç olarak 101011 değerini hesaplamasını isteriz.



Şekil-2.1: Mealy makinesi

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.

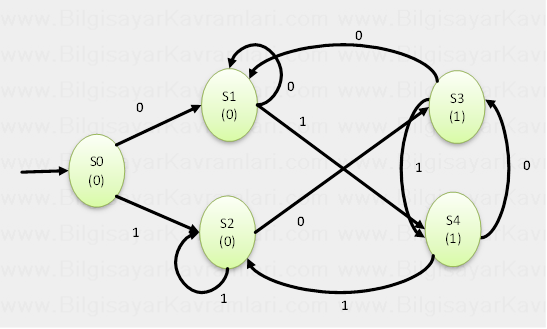
İşlemlere devam edelim ve durumları yazmaya çalışalım:

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

Tablo-2.2: Durumlar

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

Aynı örneği moore makinesi olarak tasarlayacak olsaydık:



Şekil-2.2: Moore makinesi

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 |

Tablo-2.3: Durumlar

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

**SORU-4: Eşit a ve b bulunduran Turing makinesi örneği veriniz.**

Şimdi bu soru iki türlü anlaşılabilir. Kolay olanı anbn şeklinde a ve b harflerinin sırayla gelmesi. yani n tane a harfinin n tane b harfinden önce gelmesi. Tabi bir de eşit sayıda a ve b olması ve bunların karışık dizilimde olması mümkün.

Yani ilk durumda aaabbb şeklindeki kelimeleri kabul ederken ikinci durumda aabbab şeklindeki kelimeleri de kabul etmesini bekleyebiliriz.

ilk durum için basitçe kelimenin başından bir a sildiğimizde, kelimenin sonundan da bir b silicez. Dolayısıyla a ve b harfleri aynı anda biterse eşitler, şayet birisi önce biterse eşit değiller diyeceğiz.

ikinci durum ise biraz daha zor, kelimenin başından bir harf silip (örneğin a) sonra tersi olan harfi (bu durumda b) kelimenin yine başından ilk gördüğümüz durumda silmemiz gerekiyor. Ancak bu silme işlemini kaydırarak yapmamız gerekiyor.

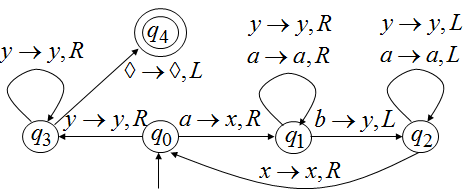
Örnek üzerinden anlatacak olursam :  
-abaaaabbbb- şeklinde bir kelime olsun:  
a ve hemen sonra kelimede gelen ilk b siliniyor  
—aaaabbbb- halini alıyor ve devam ediyoruz:  
—-aaabbbb- şimdi bir a sildik ve bir de b silmemiz gerek. Makinemiz ilk b’yi bulup silecek:  
—-aaa-bbb- fakat görüldüğü üzere kelimenin içinde boşluk oldu bunu düzeltmek için kaydırıyoruz:  
—-aaabbb– ve böylece devam ediyor.

Burada boşluk oluşması ve kaydırma probleminin çözümü için ilave bir sembol kullanılabilir. Örneğin c sembolü kullanacak olalım. ilk silmeden sonra  
-ccaaaabbbb- şeklinde ilk a ve b silindiler  
-cccaaacbbb- şeklinde ikinci silme gerçekleşti ve bu şekilde devam ediyoruz. Ta ki bütün kelime c olana kadar. Şayet b veya a önce biterse o zaman eşit değil diyebiliriz.

Yukarıda, iki problemin de çözümünü anlattım ama kolay olanının makinesini çizip adım adım yukarıdaki yazıya ekleyeceğim.

**Çözüm:**

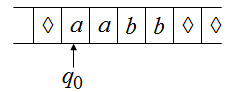
Örnek kelimemiz aabb olsun. Bu örneği çözerken sırasıyla bir a harfinin yerine x bir de b harfinin yerine y koyacağız. En sonunda, bütün a’lar x ve bütün b’ler y ile değişmiş olacak ve makinemizde hiç a ve b kalmadıysa kabul edeceğiz. Şayet a sayısı b’den fazlaysa, o zaman hala x harfine dönmeyen a’lar bulunacak veya benzer şekilde b harfi a’dan fazla ise bu durumda da hala y harfine dönmeyen b’ler bulunacaktır. Bu durumda bantta a veya b harfi kalmışsa, makinemiz bu kelimeyi kabul etmeyecektir.



Şekil-2.3: Turing makinesi

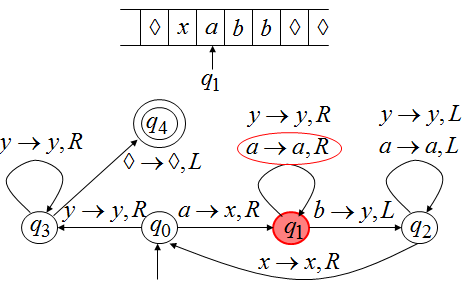
Yukardaki Turing makinesi tasarımı, problemimizi çözer. Bu çözümü bir örnek üzerinden adım adım görelim.

Örneğimiz aabb kelimesi olsun ve bakalım makinemiz bu kelimeyi kabul edecek mi.



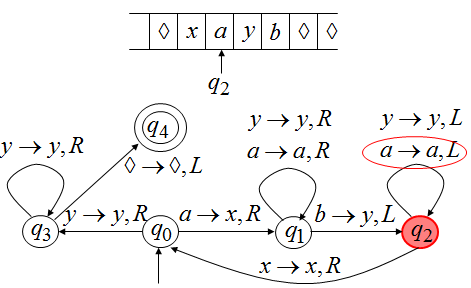
Şekil-2.4: Turing makinesi

Kelimenin ilk harfinde, makinemiz q0 durumundadır (state). Bu durumda banttan a okuduğu için q1 durumuna geçer ve banda x koyar.



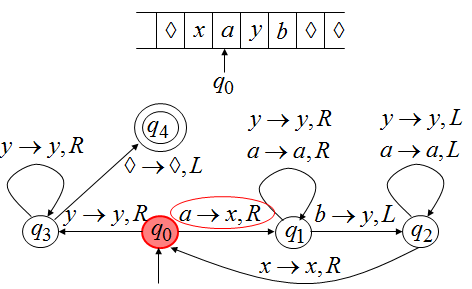
Şekil-2.5: Turing makinesi

Makinemiz, a gördüğü üzere bandı sağa (R) sarmaya devam edecek ve b görünce q2′ye geçecektir.



Şekil-2.6: Turing makinesi

Şimdi makinemiz b gördüğü durum olan q2 geldi ve geri sarmaya başlıyor. Bu işlem x görene kadar devam edecektir ve x görünce q0 durumuna dönecektir.

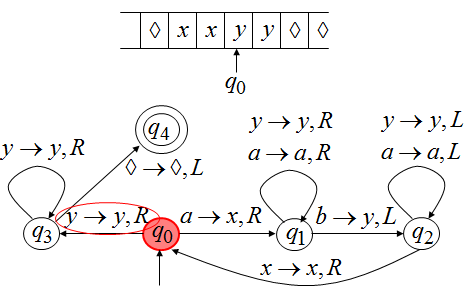


Şekil-2.7: Turing makinesi

X harfini görünce sağa sardık ve gördüğümüz ilk a harfini x ile değiştirerek makinenin şimdiye kadar yaptığı işlemleri bir kere daha tekrar etmeye başladık.

Sonuçta makinemiz, bir a harfini x ile değiştirmiştir ve bir b harfini de az önce olduğu gibi y harfi ile değiştirecektir.

İlgili değiştirme işlemleri yapıldıktan sonra makinemiz ve bandın durumu aşağıdaki şekilde olacaktır:



Şekil-2.8: Turing makinesi

Görüldüğü üzere makine, q0 durumuna geri gelmiş ve bant ilk gördüğü x harfi için bir sağa sarılmıştır.

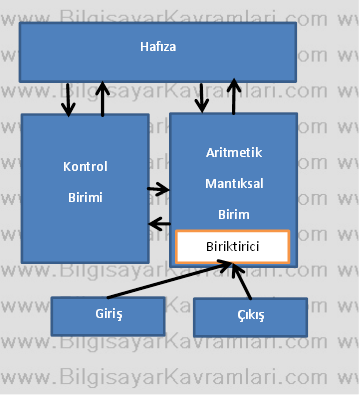
q0 durumunda, daha fazla a harfi kalmadığı için q3 durumuna geçiyoruz. Bu duruma geçme koşulumuz y bulunması hali.

q3 durumunda ise, bandın sonuna kadar y okudukça ilerliyoruz. Buradaki kritik nokta, şayet a ve b harflerinin sayısı eşitse, boşluk sembolüne kadar ilerleyeceğimizdir. Şayet b harfleri, a harflerinden fazla olursa q3 durumunda kalırız çünkü henüz hala y ile değiştirilmeyen b harfleri olacak ve makine q3 durumunda kalacaktır.

Şayet a harfinin sayısı b’den fazla olursa, bu durumda q1′de kalacağız. Çünkü bir a harfi x ile değiştikten sonra bir de b harfinin y ile değişmesi gerekecek. Bunu yapabileceğimiz bir b harfi bulunmayacak çünkü bütün b’ler zaten y ile değişmiş olacak ve makinemiz bu durumu kabul etmeyecek.

## **SORU-5: Von Neumann Makinesi hakkında bilgi veriniz.**

Bilgisayar bilimleri açısından, von neumann makineleri, günümüz klasik bilgisayarlarının temellerini oluşturur. Bu makine tanımında basitçe bir işlemci, bir hafıza ve giriş / çıkış sistemleri modellenmiştir.



Şekil-2.9: Von Neumann Makinesi

Yukarıdaki şekilde görüldüğü üzere, hafıza (memory) ile bir kontrol birimi (control unit) ve bir de aritmetik mantıksal birim (artihmetic logical unit) arasında iletişim kurulmaktadır. Ayrıca aritmetik mantıksal birim içerisinde bir adet biriktirici bulunmaktadır.

Von neumann makinesi, tasarım olarak, evrensel Turing makinesi (universal Turing Machine) problemlerini çözebilmektedir.

Von Neumann makineleri, teoride ” kayıtlı program ” olarak geçen (stored programm) problemi çözebilmeleri açısından, “kayıtlı program makinesi” (stored programm machine) olarak geçmektedir. Bunun anlamı, program tarafından kontrol edilebilen bilgisayarların bir üst seviyesi olarak, programın ve giriş çıkışların kayıt edilebilmesidir.

Von Neumann makinelerinin ayrıca bir üst seviyesi olarak, yeniden programlanabilir makineler (reprogramming) olarak kullanılabilmesidir. Örneğin bir hesap makinesi için geliştirilen cihaz, üzerinde bir program bulundurur. Bu program, hesap işlemlerinin yapıldığı temel özellikleri içerir ve donanımla doğrudan ilişkilidir. Von Neumann makineleri, bu kavramı yıkarak aynı cihaz üzerinden, hesaplama işlemlerinin yapıldığı bir programı tamamen farklı amaçlara hizmet eden bir programla değiştirmeye izin verir. Örneğin bir oyun veya bir tablolama yazılımına geçilmesi cihaz üzerinde ciddi yenileme işlemleri gerektirir.

Yukarıda anlatılan bu yeniden programlama özelliğinin desteklenebildiği ilk bilgisayar tasarımı Von Neumann bilgisayarlarıdır. Bu bilgisayarlar üzerinde hafızada kayıtlı bir program amaca yönelik olarak yüklenebilmektedir. Elbette aynı anda birden fazla programın çalışabilmesi (multi processing) çok daha sonradan geliştirilmiştir.

## **SORU-6: Ve/Veya bağlacı normal şeklleri (Conjunction / Disjunction Normal Form)**

Bu gösterim, bool cebirinde (boolean algebra) kullanılan ve kaziyeleri (önerme, proposition) [ve (and)](http://www.bilgisayarkavramlari.com/2007/11/29/ve-kapisi-and-gate/) bağlacı ile bağlamanın özel bir şeklidir. Kısaca CNF (conjuction normal form) olarak ifade edilir. [Diğer bir normal şekil olan Chomsky Normal Form (CNF) ile ilgili bilgi arıyorsanız buradan ulaşabilirsiniz.](http://www.bilgisayarkavramlari.com/2009/06/22/chomsky-normal-sekili-chomsky-normal-form/)

Bu özel şeklin taşıdığı kuralları aşağıdaki şekilde sıralayabiliriz:

Ters (not) işlemi, sadece önermelerin lafzi gösterimlerin (literal) başında bulunacaktır. Bir [kaziyenin (önerme, propositon)](http://www.bilgisayarkavramlari.com/2007/11/29/onermeler-mantigi-propositional-logic/) başında bulunmayacaktır.

Örneğin ¬A şeklindeki tek lafzi gösterim üzerinde olan ters işlemi kabul edilirken ¬(A V B) şekli kabul edilemez.

Ve (and) işlemi, iki operand almaktadır. Yani a ve b şeklindeki bir işlemde a ve b adında iki lafzi gösterim (literal) bulunmaktadır. Bu işlemin herhangi bir operandı, basit bir lafzi gösterim (literal) veya [disjunction (veya operatörü (or opeartor)](http://www.bilgisayarkavramlari.com/2007/11/29/veya-kapisi-or-gate/) ile oluşturulmuş bir) kaziye (önerme, proposition) olmalıdır.

Örneğin aşağıdaki gösterimler bu kurala uymaktadır:

A Λ B

(A V B ) Λ (C V D)

A Λ (C V D V F)

Ancak aşağıdaki gösterimler bu kurala uymaz

A Λ (B Λ C)

A Λ (B V C Λ D)

(A Λ B) Λ C

Ancak yukarıdaki gösterimler CNF (conjuction normal form) yani ve bağlacı normal şekline çevrilebilir. Bu çevirimler aşağıda gösterilmiştir (yukarıdaki her satır için sırasıyla)

A Λ B Λ C

A Λ ( B V C ) Λ ( B V D)

A Λ B Λ C

Görüldüğü üzere yukarıdaki yeni hallerinde, verilen gösterimler ve bağlacı üzerinde basitleştirilmiştir.

Disjunction Normal Form (DNF, Veya bağlacı normal şekli) ise yukarıdaki ve bağlacı için söylenen her şeyin, veya bağlacı için geçerli olması durumudur.

Kısaca bu şekilde kabul edilemeyecek gösterimleri ve çevrilmiş şekillerini aşağıda verelim:

|  |  |
| --- | --- |
| DNF şeklinde olmayanlar | DNF karşılığı |
| (A V B) Λ C | (A Λ C) V (B Λ C) |
| A Λ (B V C Λ D) | (A Λ B) V (A Λ (C Λ D)) |
| ¬(A Λ B) | ¬A V ¬B |

İki gösterimin doğruluğunun tespiti. Yani verilen bir boole cebiri diziliminin gerçekten CNF veya DNF olduğunun sınanması, veya verilen bir denklemin sonucunun doğru (true) veya yanlış (false) olarak bulunabilmesi, bu denklemdeki lafzi terim (literal) sayısına bağlıdır. Örneğin 3CNF bir denklemin, yani sadece 3 terimin conjunction (ve bağlacı ile bağlanmasından) oluşan bir denklemin karmaşıklığı NP-Complete olurken, bu terim sayısı 2′ye indirildiğinde yani 2CNF bir denklem söz konusu olduğunda, karmaşıklık P zamanda (polinom zamanda) çözülebilmektedir.  
Daha detaylı bilgi için ikili tatmin problemleri (boolean satisfaction problem) başlıklı yazıyı okuyabilirsiniz.

## **SORU-7: CYK Parçalama**

CYK parçalama algoritması (parse algorithm), verilen girdinin (input), [bir içerikten bağımsız dil (context free language)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-dil-context-free-language/) için nasıl parçalanabileceğini gösterir. CYK algoritmasının ismi, algoritmayı bulan kişilerin baş harflerinden oluşur: Cocke–Younger–Kasami.

Algoritmadaki amaç, içerikten bağımsız dilin üretebileceği parçalama ağaçlarını veya alternatif parçalama yöntemlerini gösterebilmektir.

[flashvideo file=http://www.bilgisayarkavramlari.com/wp-content/uploads/cyk.flv /]

Bu durumu bir örnek üzerinden anlatmaya çalışalım:

Örnek olarak aşağıdaki içerikten bağımsız dili ele alalım:

S -> AB | SA

A -> BA | CB | a

B -> CC | b

C -> SS | AC|a

Yukarıdaki bu dil için örnek girdimiz (input) : “ababa” olsun ve biz bu girdiyi CYK algoritmasına göre parçalamaya çalışalım:

Bu durumu bir tablo yardımıyla yapıyoruz.

Tablomuz, girdimizin uzunluğu 5 olduğu için, 5×5 boyutlarında oluyor:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| a | b | a | b | a |

Yukarıda görüldüğü üzere bir tablo oluşturduk ve bu tablonun her sütunun altına, girdimizdeki bir harfi yerleştirdik.

CYK algoritması, bu tablo üzerinde bir merdivenin basamaklarını tırmanır gibi sırasıyla her seviyede, altında kalan kelimeyi bulacaktır.

Bu durum aşağıdaki tabloda gösterilmiştir:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ababa |  |  |  |  |
| abab | baba |  |  |  |
| aba | bab | aba |  |  |
| ab | ba | ab | ba |  |
| a | b | a | b | A |
| a | b | a | b | a |

Görüldüğü üzere, tablodaki en üst satırda bütün girdinin parçalanmış hali yer alacaktır. Bunun altında girdi boyutunun bir eksiği, onun altında iki eksiği … nihayet en altta tek boyutlu girdiler yer alacaktır.

CYK algoritmasının amacı bu tek boyutlu girdilerden yukarı doğru girdinin tamamını oluşturmaktır.

Şimdi CYK algoritması ile çalışmaya hazırız, önce her sonlu (terminal) için dilimizde tanımlı devamlıları (non-terminal) yerleştiriyoruz.

Örneğin a harfi için A ve b harfi için B devamlılarını yazabiliriz:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| A,C | B | A,C | B | A,C |
| a | b | a | b | a |

Ardından iki uzunluğundaki kelimelere geçiyoruz.

Örneğin aşağıdaki tabloda gösterilen renkli hücreye gelen değer, bu hücrenin hemen altındaki iki hücrede bulunan terminallerden üretilen değerdir.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| A,C | B | A,C | B | A,C |
| a | b | a | b | a |

Bu hücreye, altında kalan ab girdisinin değeri yazılacaktır. Bu girdiyi belirleyen değer ise a girdisi ve b girdisi için yazılan bir alt satırdaki değerlerdir.

O halde ab girdisi için CFL üzerinde AB, CB aranacak. Bu değerlerden, AB’ye S, CB’ye A’dan ulaşılabiliyor o halde bu hücreye S,A yazıyoruz.

Diğer hücreler de benzer şekilde hesaplanır:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| S,A | A | S,A | A |  |
| A,C | B | A,C | B | A,C |
| a | b | a | b | a |

Bir üst satırı da benzer şekilde hesaplayalım. Burada ilk sütuna aba girdisinin sonucu yazılacak. Bu girdi ab + a veya a+ab şeklinde yazılabilir. O halde bir önceki satırlardan hesapladığımız değerleri bu denkleme yazacak olursak:

ab -> S,A

a -> A,C

ba -> A

b -> B

ihtimaller şunlar olacaktır:

SA,SC,AA,AC,AB

Bu ihtimalleri içeren devamlılar sırasıyla aşağıda verilmiştir:

S , X, X, C ,S

Yukarıdaki satırda bulunan X değerlerine geçiş olmadığı gösterilir. Yani CFL üzerinde SC veya AA sonucuna ulaşılan bir devamlı yoktur.

Diğer hücreleri de aşağıdaki şekilde doldurabiliriz:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
| S,C | S,A | S,C |  |  |
| S,A | A | S,A | A |  |
| A,C | B | A,C | B | A,C |
| a | b | a | b | a |

Bir üst satırda ilk hücre için bakılacak alternatifler aşağıdadır:

aba+b veya a + bab

bu alternatiflerden aba + b için : S,C + B -> SB, CB

diğer alternatif olan a +bab için A,C + S,A -> AS, CS, AA, CA

Toplamda alternatiflerimiz : SB, CB, AS, CS, AA, CA

Bu alternatiflerin dilimizdeki tanımlarına bakarsak : X, A, X, X, X, X

Sadece A sonucu çıkar.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| A |  |  |  |  |
| S,C | S,A | S,C |  |  |
| S,A | A | S,A | A |  |
| A,C | B | A,C | B | A,C |
| a | b | a | b | a |

Bir yanındaki sütun için alternatiflerimiz : BS, BC, SA, SC, AA, AC

Bu alternatiflerin dilimizdeki tanımları: X, X, S, X, X, C

Sonuç olarak S,C bulunur:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| A | S,C |  |  |  |
| S,C | S,A | S,C |  |  |
| S,A | A | S,A | A |  |
| A,C | B | A,C | B | A,C |
| a | b | a | b | a |

Son satırımızdaki ihtimaller ise:

a+baba : AS, AC, CS, CC

abab+a : AA, AC

Bütün ihtimaller bu durumda: AS, AC, CS, CC, AA

Bu ihtimallerin dilde tanımları : X, C, X, B, X

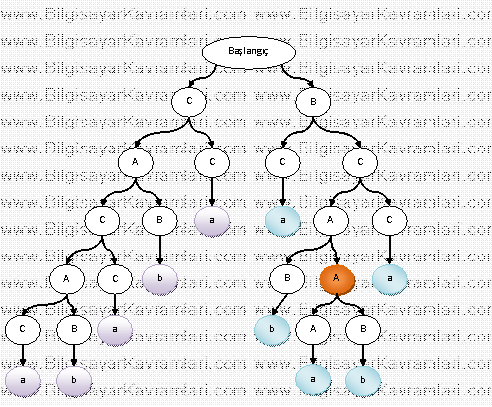
Dolayısıyla bu hücrenin değeri CB olarak bulunur.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C,B |  |  |  |  |
| A | S,C |  |  |  |
| S,C | S,A | S,C |  |  |
| S,A | A | S,A | A |  |
| A,C | B | A,C | B | A,C |
| a | b | a | b | a |

Yukarıdaki bu tablo bize, örnek girdinin parçalanması için alternatif yolları göstermektedir.

Örneğin girdimiz ababa ise, bu girdi için C veya B’den başlanarak parçalama yapılmalıdır. Örneğin dilimiz, S ile başlamak zorundaysa bu girdi parçalanamaz.

Yukarıdaki bu tablonun bir [parçalama ağacı (parse tree)](http://www.bilgisayarkavramlari.com/2008/12/09/parcalama-agaci-parse-tree/) şeklinde çizimi aşağıdadır:



Yukarıdaki şekilde C veya B ile başlayan iki ayrı [parçalama ağacı (parse tree)](http://www.bilgisayarkavramlari.com/2008/12/09/parcalama-agaci-parse-tree/) örneği gösterilmiştir.

İki taraftan da ababa sonucuna ulaşıldığı görülebilir. Elbette bu parçalama işlemi sırasında farklı alt dallardan da dolaşılması söz konusu olabilir. Örneğin turuncu renke gösterilen A düğümünü ele alalım. Bu düğüm A->AB şeklinde açılmıştır. Oysaki aynı düğümün CB şeklinde açılıp buradan da “ab” girdisine indirilmesi mümkün olabilirdi.

## **SORU-8: DFA Metin Arama Algoritması (DFA Text Search)**

[1. Otomatın İnşası](http://bilgisayarkavramlari.sadievrenseker.com/2009/11/24/dfa-metin-arama-algoritmasi-dfa-text-search/#1)  
[2. Algoritmanın arama aşaması](http://bilgisayarkavramlari.sadievrenseker.com/2009/11/24/dfa-metin-arama-algoritmasi-dfa-text-search/#2)  
[3. Algoritmanın çalışması](http://bilgisayarkavramlari.sadievrenseker.com/2009/11/24/dfa-metin-arama-algoritmasi-dfa-text-search/#3)  
[4. Algoritmanın kodlanması](http://bilgisayarkavramlari.sadievrenseker.com/2009/11/24/dfa-metin-arama-algoritmasi-dfa-text-search/#4)

Bilgisayar bilimlerinde, bir metnin içerisinde farklı bir metnin veya bir kelimenin aranması sırasında kullanılan algoritmalardan birisidir. Algoritma, aranan kelime için bir [otomat (automaton)](http://www.bilgisayarkavramlari.com/2008/08/02/sonlu-ototmatlar-finite-automaton/) oluşturur ve hedef metin içerisinde bu otomata göre arama işlemi yapar.

Oluşturulana otomatın [DFA (deterministic finite automaton, belirli sonlu otomat)](http://www.bilgisayarkavramlari.com/2008/11/11/belirli-sonlu-otomat-deterministic-finite-automat/) olması gerekmektedir.

Algoritmanın çalışması iki aşamada incelenebilir:

* Aranan kelimeye özgü bir otomatın inşa edilmesi
* İnşa edilen bu otomatın bir metin içerisinde arama işlemi için kullanılması

Yukarıdaki bu adımları sırasıyla anlatmaya çalışalım ve ayrıca performanslarını inceleyelim.

**Otomatın inşası**

Bilindiği üzere bir DFA ifade edilirken şu dört terimden faydalanılır: **A**(x) =(**Q**, q0, **T**, **E**). Bu tanımın üzerinde arama yapacağı dili x harfleri kümesinden oluşan [bir dil (language)](http://www.bilgisayarkavramlari.com/2008/08/02/alfabe-abece-alphabet/) (http://www.bilgisayarkavramlari.com/wp-content/uploads/112409_0429_DFAMetinAra1.gif\*x) olarak tanımlarsak, otomatı oluşturan bu terimlerin metin arama işlemi için uyarlanmış hallerini şöyle açıklayabiliriz.

* Q kümesi x harflerinden oluşan ve bütün olasılıkları ifade eden kümedir ve **Q**={http://www.bilgisayarkavramlari.com/wp-content/uploads/112409_0429_DFAMetinAra2.gif, x[0], x[0 .. 1], … , x[0 .. m-2], x} olarak gösterilebilir.
* q0, otomatın başlangıç terimidir ve başlangıçta otomatta boş küme olduğunu düşünürsek q0=http://www.bilgisayarkavramlari.com/wp-content/uploads/112409_0429_DFAMetinAra3.gif; olarak gösterilebilir.
* T, otomatın üzerinde tanımlı olduğu dili oluşturan sembolleri gösterir. Bu örnekte dilimizin http://www.bilgisayarkavramlari.com/wp-content/uploads/112409_0429_DFAMetinAra4.gif\*x şeklinde tanımlı olmasından dolayı T = {x} olarak gösterilebilir.
* Son olarak E teriminin tanımını Q kümesinde tanımlı bütün “q” değerleri ve http://www.bilgisayarkavramlari.com/wp-content/uploads/112409_0429_DFAMetinAra5.gifdilinde tanımlı herhangi bir “a” terimi için (q, a, qa) şeklinde tanımlayabiliriz. Bu tanımın doğruluğu, “a” değerinin “x” değerinin bir ön eki olması şartına bağlayabiliriz. Şayet bir ön ek olarak kabul edilmezse, yine (q,a,p) üçlüsünün E kümesinde olması şartıyla, p değeri x’in bir ön eki olan qa için en uzun ek olur.

Yukarıdaki otomat inşa edilirken m uzunluğunda bir kelime olduğu kabul edilirse (ki yukarıdaki Q kümesinin elemanları bu uzunlukta verilmiştir) inşa için gereken süre O( m + r) ve inşa için gereken hafıza ise O(mr) olarak hesaplanabilir.

**Algoritmanın araması**

Algoritma, yukarıdaki şekilde bir DFA inşa ettikten sonra bu DFA’i kullanarak hedef metin içerisinde arama yapar. Bu arama işlemi n boyutundaki bir metin için O(n) zamanda yapılabilir.

Bu arama işleminin çalışmasını bir örnek üzerinden inceleyelim.

**Algoritmanın çalışması**

Örnek olarak “bilgi” kelimesini “bilbilgisayarkavramlari” metninin içerisinde aramaya çalışalım.

Öncelikle kelimemiz için bir [belirli otomat (DFA)](http://www.bilgisayarkavramlari.com/2008/11/11/belirli-sonlu-otomat-deterministic-finite-automat/) oluşturmamız gerekiyor.

Dilimizdeki (http://www.bilgisayarkavramlari.com/wp-content/uploads/112409_0429_DFAMetinAra6.gif\*x) alfabemizi tanımlayacak olursak:

{a,b,g,i,k,l,m,r,s,v,y} harflerinden oluşan bir dil olarak tanımlanabilir.

Bu dil üzerinde belirli bir sonlu otomat “bilgi” kelimesi için aşağıdaki şekilde inşa edilebilir:

d 0 { a -> 0 b -> 1 g -> 0 i -> 0 k -> 0 l -> 0 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

d 1 { a -> 0 b -> 1 g -> 0 i -> 2 k -> 0 l -> 0 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

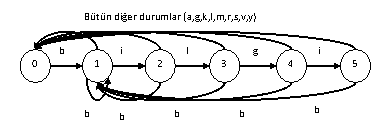
d 2 { a -> 0 b -> 1 g -> 0 i -> 0 k -> 0 l -> 3 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

d 3 { a -> 0 b -> 1 g -> 4 i -> 0 k -> 0 l -> 0 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

d 4 { a -> 0 b -> 1 g -> 0 i -> 5 k -> 0 l -> 0 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

d 5 { a -> 0 b -> 1 g -> 0 i -> 0 k -> 0 l -> 0 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

Yukarıda verilen durumlar için gidilecek durumlar listelenmiştir. Bu otomatı daha iyi anlayabilmek için aşağıdaki şekilde çizebiliriz:



Yukarıdaki otomatta, 0. Durum başlangıç durumu olmaktadır. Ayrıca 5. Duruma her ulaşılmasında aranan metin bulunmuş demektir.

Yukarıdaki otomatın örnek metin üzerindeki çalışmasını adım adım açıklamaya çalışalım:

bilbilgisayarkavramlari

|

İlk olarak hedef metinden b harfi alınıyor. Bu harf otomatımızın ilk geçişine tekabül ediyor ve dolayısıyla 0. Durumdan 1. Duruma geçiyoruz.

Geçiş: 0.b -> b

1. durumdayken ikinci harfi hedef metinden okuyoruz:

Bilbilgisayarkavramlari

|

Şu anda 1. Durumdan 2. Duruma geçmek için şartımız oluştu

Geçiş: b.i -> bi

Bu işlem aşağıdaki şekilde devam ettirilir:

BIlbilgisayarkavramlari

|

Geçiş: bi.l -> bil

BILbilgisayarkavramlari

|

Geçiş: bil.b -> b

Yeni gelen harf, otomattaki 3. Durumdan 1. Duruma dönmeyi gerektiren b harfidir. Dolayısıyla aradığımız kelimenin ilk 3 harfi bulunmasına karşılık kelimenin tamamı bulunamamış ve başa dönülmüştür.

bilBilgisayarkavramlari

|

Geçiş: b.i -> bi

bilBIlgisayarkavramlari

|

Geçiş: bi.l -> bil

bilBILgisayarkavramlari

|

Geçiş: bil.g -> bilg

bilBILGisayarkavramlari

|

Geçiş: bilg.i -> bilgi

bilBILGIsayarkavramlari

|

Geçiş: bilgi.s -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.a -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.y -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.a -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.r -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.k -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.a -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.v -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.r -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.a -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.m -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.l -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.a -> 0

bilBILGIsayarkavramlari

|

Geçiş: 0.r -> 0

bilBILGIsayarkavramlari

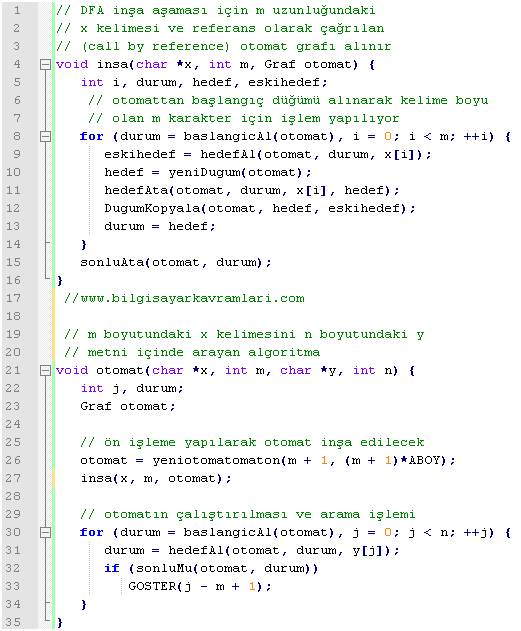
|

Geçiş: 0.i -> 0

Sonuç olarak, yukarıdaki büyük harfle gösterilen alanda, aranan kelime bulunmuştur.

**Algoritmanın kodlanması**

Algoritmanın kodu C dili için aşağıdaki şekilde yazılabilir.



Yukarıdaki kodda, arama işlemini yapan ana fonksiyon, otomat fonksiyonudur. Arama işlemi sırasında bu fonksiyon çağrılarak işlem başlatılır. Bu fonksiyonun otomatı inşa etmek için kullandığı inşa fonksiyonu, kodun 27. Satırında çağrılmıştır. Bu çağırma işleminden önce bir otomaton oluşturulmuş ve [atıf ile çağırma (call by reference)](http://www.bilgisayarkavramlari.com/2009/01/12/atif-ile-cagirma-call-by-reference/) kullanılarak inşa fonksiyonuna geçirilmiştir.

Kod, 8 ile 14. Satırlar arasında bir DFA inşa etmekte ve 30 ile 33. Satırlar arasında ise bu otomatı kullanarak hedef metin içerisinde arama yapmaktadır.

## **SORU-9: Push Down Automata**

Aşağı sürüklemeli otomatlar (push down automaton) yapı olarak birer otomat makineleridir. ~~Normal bir sonlu otomattan farkı, belirli (deterministic) olması ve ilave bir yığın (stack) bulundurmasıdır. Yani makinemiz basitçe her adımda ne yapacağından tam olarak emindir (belirli ,deterministict) ve veri depolamak için hafızada bulunan bir yığından (stack) istifade edebilir.~~ Düzeltme (Tarık Bey’e teşekkürler): PDA’ler için kullanılan otomata (automaton) göre belirli (deterministic) veya belirsiz (nondeterministic) olma ihtimali vardır. Yani kullanılan [otomat belirliyse (Deterministic)](http://www.bilgisayarkavramlari.com/2008/11/11/belirli-sonlu-otomat-deterministic-finite-automat/) bu PDA  de belirli aşağı sürüklemeli otomat ( Deterministic Push Down Automaton DPDA) olarak isimlendirilir. Şayet tersine kullanılan [otomat belirsizse (non deterministic)](http://www.bilgisayarkavramlari.com/2008/11/11/belirsiz-sonlu-otomat-nondeterministic-finite-automat-nfa/) bu durumda otomat, belirsiz aşağı sürüklemeli otomat (NonDeterministic PushDownAutomata NPDA) olarak isimlendirilir. Bir sonlu otomattan (finite state machine) farkı ise [yığın (stack)](http://www.bilgisayarkavramlari.com/2007/05/04/stack-yigin/) kullanılmasıdır.

Bilindiği üzere [yığın (stack)](http://www.bilgisayarkavramlari.com/2007/05/04/stack-yigin/) yapısının temel iki fonksiyonu bulunur. Koyma (push) ve alma (pop) işlemleri isimlerinden de anlaşılacağı üzere yığına koyma ve yığından bir veriyi alma işlemini gerçekleştirir. PDA içerisinde bu fonksiyonlar aynen bulunur. Buna ilave olarak bir pda’i açıkça tanımlayabilmek için 6 bilgi gerekir. Bu bilgiler aşağıdaki şekilde gösterilebilir:

(Q,,,  ,q0,F)

Yukarıdaki satır bir aşağı sürüklemeli otomat için kabul edilen en standart gösterim şeklidir. Kaynaklarda aynı bilgiyi ifade etmek için farklı semboller kullanılmaktadır ancak semboller değişse de pda için bu bilgiler gerekir:

Q: [makinemizde (otomatımızda ,automaton) bulunan durumları (states)](http://www.bilgisayarkavramlari.com/2008/08/02/sonlu-ototmatlar-finite-automaton/) gösterir.

: makinemizin kabul ettiği girişte kullanılabilecek alfabenin (alphabet) kümesidir.

: [yığında (stack)](http://www.bilgisayarkavramlari.com/2007/05/04/stack-yigin/) kullanılabilecek [alfabenin (alphabet)](http://www.bilgisayarkavramlari.com/2008/08/02/alfabe-abece-alphabet/) kümesidir.

: Q ile gösterilen durumlar (states) arasındaki geçişlerin kümesidir.

q0: başlangıç durumudur ( initial state)

F: Bitiş durumudur (final state)

Yukarıdaki bu akademik gösterim ile neyin kastedildiğini bir örnek üzerinden anlamaya çalışalım. Örnek olarak çok klasik bir makine olan 0n1 n probleminin çözüm makinesini pda olarak göstermek isteyelim. Diğer bir deyişle makinemiz bir giriş kelimesini alacak ve bu kelimedeki 0′ların sayısı 1′lerin sayısına eşitse ve o’lar 1′lerden önce geliyorsa bu kelimeyi kabul edecek, şayet 0′ların sayısı ve 1′lerin sayısı eşit değil veya sıralamada bir hata varsa bu girdiyi kabul etmeyecek.

Çözmeye çalıştığımız problemi daha iyi anlayabilmek için bir iki girdinin kabul edilip edilmeyeceğini inceleyelim:

 : kabul, n= 0 için doğru (burada  sembolü ile boş girdi kastedilmiştir)

01 : kabul , n = 1 için doğru

10 : ret , sıralama hatası, 0′lar 1′lerin önünde olmalı

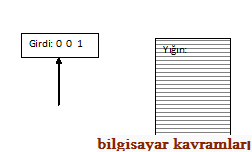
0011: kabul n = 2 için doğru

0101 : ret, sıralama hatası , bütün 0′lar, 1′lerin önünde olmalı

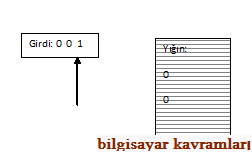
00011: ret, 0′ların sayısı ile 1′lerin sayısı tutmuyor

Şimdi yukarıdaki problemin çözümü olan aşağı sürüklemeli otomatımızı tasarlayalım. Yukarıdaki tanımda 6 unsurun bulunması gerektiğinden bahsetmiştik. Sırayla bunlara cevap arayalım. Önce makinemizi tasarlayarak başlayalım. Makinenin tasarımı için bir [yığın (stack)](http://www.bilgisayarkavramlari.com/2007/05/04/stack-yigin/) kullanacağız. Biliyoruz ki şayet makineye gelen 0′ları sırasıyla koyarsak (push) ve 0′lar bittikten sonra gelen her 1 için yığından bir eleman alırsak (pop) bu durumda makine girdi kelimesi bittiğinde yığını boş bulursa (yığındaki harfler ile girdideki harfler aynı anda biterse) kelimeyi kabul edecek aksi halde reddedecektir.

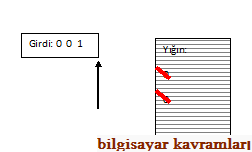
Bu tasarımı bir iki örnek ile anlamaya çalışalım. Örneğin girdimiz 0011 olsun.



İlk durumda makinemiz girdinin ilk harfi olan 0′ı okuyacak ve 0 gördükçe yığına koyacak (push)



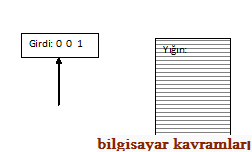
Yığına 0′ları koyduktan sonra her gördüğü 1 için yığından bir 0 çıkaracak (pop):



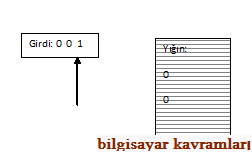
Görüldüğü üzere 2 adet 1 harfi için 2 adet 0 harfi yığından çıkarılmıştır (pop) ve sonuçta girdi kelimenin sonuna ulaşılmış ve yığında aynı anda boşalmıştır. Dolayısıyla 0011 kelimesini kabul ederiz.

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.

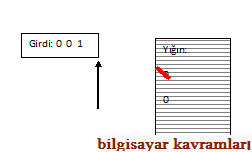
Benzer makinemiz için bu sefer reddedilecek bir girdiyi tecrübe edelim. Misal girdimiz 001 olsaydı ne olurdu?



İlk durumda makinemiz girdinin ilk harfi olan 0′ı okuyacak ve 0 gördükçe yığına koyacak (push)

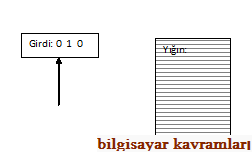


Yığına 0′ları koyduktan sonra her gördüğü 1 için yığından bir 0 çıkaracak (pop):

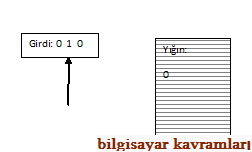


Görüldüğü üzere girdi kelimesinin sonuna gelindiğinde hâlâ yığında bir harf bulunmakta bu durumda girdiyi reddetmek zorundayız.

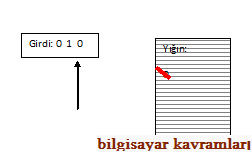
Farklı bir örnek olarak 0101 girdisini tecrübe edelim:



İlk durumda makinemiz girdinin ilk harfi olan 0′ı okuyacak ve 0 gördükçe yığına koyacak (push)

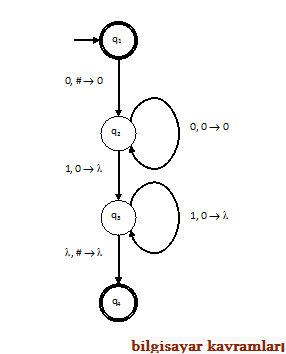


Yığına 0′ları koyduktan sonra her gördüğü 1 için yığından bir 0 çıkaracak (pop):



Bu defa sıfırları yığına koyup ardından her 1 için bir çıkarma (pop) işlemi yaptıktan sonra hâlâ girdi kelimesinin bitmediğini görüyoruz. Bu durumda girdi kelimemizi kabul etmiyoruz.

Tasarımını ve testini yaptığımız yukarıdaki makinemizi bir sonlu durum makinesi olarak çizmek istersek aşağıdaki şekli elde ederiz:



Yukarıdaki makine tasarımımızı kısaca gözden geçirecek olursak. Kabul edilen durumlar q1 ve q4 durumlarıdır. Yani  (boş kelime) durumunu kabul için q1 diğer kabul edilir durumlar için de q4 durumu (state) tasarlanmıştır. Tasarımda bulunan q2 durumu geçiş durumudur. Yani q2 durumunda bulunduğu sürece makine girdiden bir harf okumakta ve bu harf 0 olmaktadır. Okunan harf 0 olduğu sürece de bu değer [yığına (stack)](http://www.bilgisayarkavramlari.com/2007/05/04/stack-yigin/) konulmaktadır (push). Bu durum (yani q2 durumu) girdiden bir harf olarak 1 gelmesinde bozulur. Şayet girdiden 1 harfi okunursa bu defa durum değiştirilerek q3 durumuna geçilir ve bu q3 durumunda da girdiden 1 harfi okundukça yığından 0 harfi çıkarılır (pop).

Son durumda şayet girdi boşsa ve yığın da boşsa q4 durumuna yani kabul durumuna geçilir. Bunun dışındaki ihtimallerde q2 yada q3 gibi kabul edilmeyen bir duruma takılır ve makinemiz girdiyi reddeder.

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 makinemiz görüldüğü üzere başarılı bir şekilde çalışıyor. Bu makineyi bir aşağı sürüklemeli otomat (push down automaton) olarak yazacak olursak aşağıdaki tanımları yapmamız gerekir:

={0,1} olacaktır çünkü girişte ya 0 ya da 1 gelebilmektedir. Bunun dışında bir harfin gelmesi söz konusu değildir.

={0,#} olacaktır. Burada yığında olabilecek harfler kümesi gösterilirken dikkat edilirse yığına sadece 0 harfini koyuyoruz (istenildiği kadar). Dolayısıyla 1 harfi bu kümede bulunmaz. Bu kümede bulunan # sembolü ise yığının boş olduğunu ifade için konulmuştur. Yani yığın boş olduğunda bunu da bir şekilde göstermemiz gerekir. Bu boş durumu temsilen # sembolü kullanılmıştır. Yine farklı kaynaklarda farklı geçen bir semboldür ancak anlamsal olarak bütün PDA gösterimlerinde böyle bir sembol bulunmalıdır.

Q= { q1, q2, q3, q4 } olarak yazılabilir çünkü yukarıdaki makine tasarımında da görüldüğü üzere 4 durum bulunmaktadır.

F = { q1, q4 } olarak yazılabilir çünkü yukarıdaki makinede kabul edilen 2 durum vardır ve bunlar da q1 ve q4 durumlarıdır.

q0 = { q1 }olarak yazılabilir. Burada şekilden de anlaşılacağı üzere başlangıç durumumuz q1 durumudur.

Son olarak [sonlu durum makinemizin (finite state machine)](http://www.bilgisayarkavramlari.com/2008/08/02/sonlu-ototmatlar-finite-automaton/) tasarımını göstermemiz gerekmektedir. Bu gösterim için yine farklı kaynaklarda farklı yöntemler kullanılmaktadır. Örneğin yukarıdaki şekilde bir sonlu durum makinesi çizilmesi yeterli görülebilirken bazı kaynaklarda aşağıdaki gösterim kullanılmıştır. Hangi gösterim kullanılırsa kullanılsın sonuçta bu adımda anlatılan şey bir sonlu durum makinesidir ve bu makinede bulunan ve makineyi bir PDA yapan ise bir [yığın (stack)](http://www.bilgisayarkavramlari.com/2007/05/04/stack-yigin/) kullanılmasıdır.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Giriş  Yığını | 0 | | | 1 | | |  | | |
| 0 | # |  | 0 | # |  | 0 | # |  |
| q1 |  |  | q2,0 |  |  |  |  |  |  |
| q2 |  |  |  | q3,  |  |  |  |  |  |
| q3 |  |  |  | q3,  |  |  |  | q4,  |  |
| q4 |  |  |  |  |  |  |  |  |  |

Yukarıdaki gösterim aslında şekil olarak çizilen sonlu durum makinesini tablo olarak göstermekten başka bir işe yaramaz. Bu tabloda 9 sütün bulunmaktadır. Bunlar sırasıyla 0,1 ve  durumlarından yine 0,1 ve  durumlarına geçişleri gösterir. Örneğin tablonun ilk satırı 0′dan 0 durumuna geçişi son sütünü ise ‘dan  durumuna geçişi gösterir.

Tablodaki 4 satır ise makinemizdeki durumları gösterir. Makinenin anlamlı olabilmesi için başlangıç durumunun { q1 } olduğunu ve kabul edilir bitiş durumlarının { q1, q4 } olduğunu bilmek gerekir (ki bu bilgiyi zaten veriyoruz).

Yukarıdaki tablonun okunmasını daha net anlamak için bir örnek durumu inceleyelim. Örneğin makinemiz girdide bulunan bütün 0′ları okumuş ve yeni bir harf olarak 1 ile karşılaşmış olsun. Bu durumda q2 durumundan q3 durumuna geçiş yapması gerekir. Şekilde görüldüğü üzere bunu yapacağı tek durum bir 1 okunması halinde q3 geçmesini söyleyen satırdır.

## **SORU-10: Özyineli Geçiş Ağları (Reursive Transition Networks)**

Veri modellemede kullanılan bir ağ şeklidir. Esas itibariyle [içerikten bağımsız dillerin (context free grammers)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) görsel gösterimi için kullanılabilirler. Ağların yapısı [uzatılmış geçiş ağlarına (augmented transition network)](http://www.bilgisayarkavramlari.com/2009/06/15/augmented-transition-network-atn-uzatilmis-gecis-agi/) benzemekle birlikte en büyük farkı ve isminin özyineli olmasının da sebebi ağın kendini tekrarlama özelliğidir.

Daha basitçe bir [içerikten bağımsız dil (CFG)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) S devamlısı (nonterminal) ile başlayan bir kurallar listesidir. Bu listedeki her kural bir devamlıdan (nonterminal) bir sonluya (terminal) doğru yapılan bir açıklamadır. Özyineli geçiş ağlarında bu açıklamada (yani [CFG](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) kurallarının sağ tarafında) da S devamlısı (nonterminal) bulunabilmektedir.

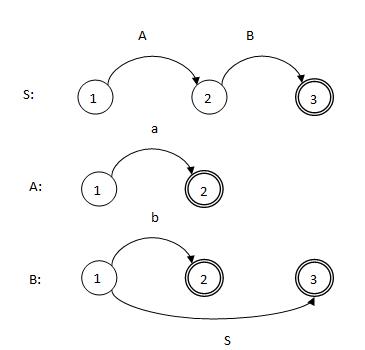
Örneğin aşağıdaki kuralları ele alalım:

S → A B

A → a

B → b | S

Yukarıdaki son kuralda S bağlangıç devamlısına bir bağlantı kurulmuştur ve aşağıdaki şekilde görselleştirilebilir:



Yukarıdaki ağ yapısı verilen CFG örneğinin çizilmiş halidir. Buradaki çizimden de görüleceği üzere ağ yeniden başa döneme özelliğine sahiptir.

Doğal dil olarak yorumlanacak olursa bir cümlenin içinde alt cümlelerin bulunması bu ağ ile gösterilebilir. Örneğin aşağıdaki örnek cümleyi ele alalım:

Ali geldiğini söylemeyi unuttu.

Yukarıdaki örnekte 3 cümle iç içedir.

Ali gelmek

Ali söylemek

Ali unutmak

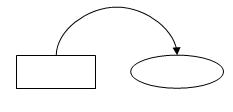
Basit bir yaklaşımla bir cümleyi özne – yüklem olarak tanımlayacak olursak yukarıdaki örnek cümlenin uyduğu yapı aşağıdaki şekilde olabilir:

C: Ö Y

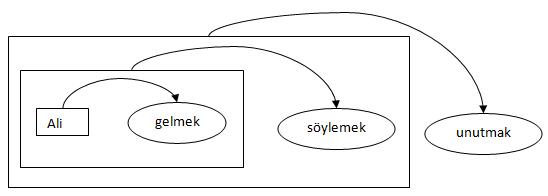
Ö : “Ali”

Y: gelmek | söylemek | unutmak

Yukarıdaki CFG yapısını şayet özyineli geçiş ağımız (recursive transition network) ile gösterecek olursak:



Örneğin basit bir Özne-Yüklem ilişkisini yukarıdaki şekilde gösterelim. Yani özneleri dikdörtgen ve yüklemleri yuvarlak ile gösterelim. Bu durumda örneğimiz:



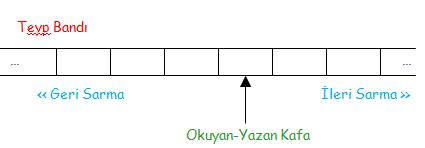
Görüldüğü üzere özyineli ağımızla gösterilen örnekte unutmak fiilinin başındaki isim kelime grubu “ali geldiğini söylemeyi” yine bir cümledir ve yapısı itibariyle bir önceki şekilde gösterilen şablonu içermektedir. Benzer şekilde söylemek fiilinin başındaki isim grubu da bir geçiş ağı (transition network) özelliği taşımaktadır

## **SORU-11: Turing Makinesi (Turing Machine)**

Bilgisayar bilimlerinin önemli bir kısmını oluşturan [otomatlar (Automata)](http://www.bilgisayarkavramlari.com/category/automata-otomatlar/) ve [Algoritma Analizi (Algorithm analysis)](http://www.bilgisayarkavramlari.com/category/algoritma-analizi-teory-of-algorithms/) 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)](http://www.bilgisayarkavramlari.com/2009/06/27/chomsky-hiyerarsisi-chomsky-hierarchy/) için 1. seviye (type-1) olan ve Turing makinesi ile kabul edilebilen diller bütün tip-2 ve tip-3 dilleri yani içerk bağımsız dilleri ve düzenli dilleri kapsamaktadır. Ayrıca ilave olarak içerik bağımsız dillerin işleyemediği (üretemediği veya parçalayamadığı (parse) ) anbncn şeklindeki kelimeleri de işleyebilmektedir.  Düzenli ifadelerin işleyememesi konusunda bilgi için [düzenli ifadelerde pompalama savı (pumping lemma in regular expressions)](http://www.bilgisayarkavramlari.com/2009/03/22/duzenli-ifadelerde-pompalama-onsavi-pumping-lemma-for-regular-expressions/) ve [içerik bağımsız dillerin işlemeyemesi için de içerik bağımsız dillerde pompalama savı (pumping lemma for CFG)](http://www.bilgisayarkavramlari.com/2009/03/22/icerik-bagimsiz-gramerler-icin-pompalama-onsavi-pumping-lemma-for-context-free-grammers/) başlıklı yazıları okuyabilirsiniz.

**Turing Makinesinin Akademik Tanımı**

Turing makineleri literatürde akademik olarak aşağıdaki şekilde tanımlanır:

turing_akademik

Burada M ile gösterilen makinenin parçaları aşağıda listelenmiştir:

Q sembolü sonlu sayıdaki durumların [kümesidir](http://www.bilgisayarkavramlari.com/2009/06/24/kume-teorisi-set-theory/). Yani makinenin işleme sırasında aldığı durumardır.

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

Σ sembolü ile makineye verilecek girdiler (input) [kümesi](http://www.bilgisayarkavramlari.com/2009/06/24/kume-teorisi-set-theory/) gösterilir. Girdi [kümesi](http://www.bilgisayarkavramlari.com/2009/06/24/kume-teorisi-set-theory/) dildeki harfler dışında bir sembol taşıyamayacağı için Σ ⊆ Γ demek doğru olur.

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

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

q0 sembolü makinenin başlangıç durumunu (state) tutmaktadır ve dolayısıyla q0 ⊆ Q olmak zorundadır.

F sembolü makinenin bitiş durumunu (state) tutmaktadır ve yine F ⊆ 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)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) Turing makinesi ile gösterelim ve bize verilen aaa şeklindeki 3 a yı makinemizin kabul edip etmediğine bakalım.

Tanım itibariyle makinemizi aşağıdaki şekilde tanımlayalım:

M = { {q0,q1} , { a } , { a,x } , { q0 a→a R q0 , q0 x→x L q1} , q0 , x , q1 }

Yukarıdaki bu makineyi yorumlayacak olursak:

Q değeri olarak   {q0,q1} 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.

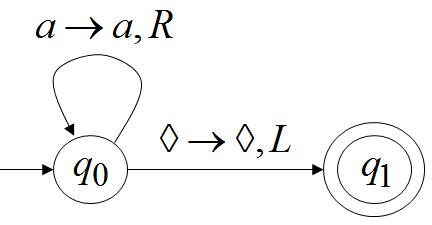
δ değeri olarak iki geçiş verilmiştir { q0 a→a R q0 , q0 x→x L q1} buraadki R sağa sarma L ise sola sarmadır ve görüleceği üzere Q değerindeki durumlar arasındaki geçişleri tutmaktadır.

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

q0 ile makinenin başlangıç durumundaki hali belirtilmiştir.

F değeri olarak q1 değeri verilmiştir. Demek ki makinemiz q1 durumuna geldiğinde bitmektedir (halt) ve bu duruma gelmesi halinde bu duruma kadar olan girdileri kabul etmiş olur.

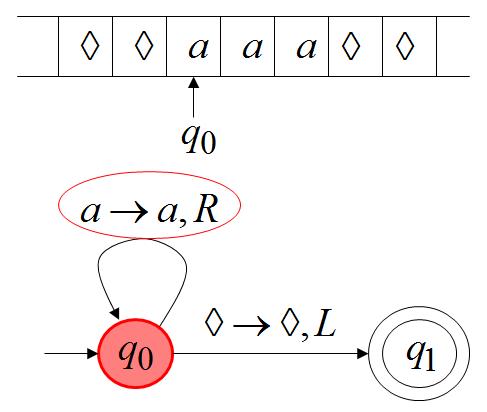
Yukarıdaki bu tanımı görsel olarak göstermek de mümkündür:



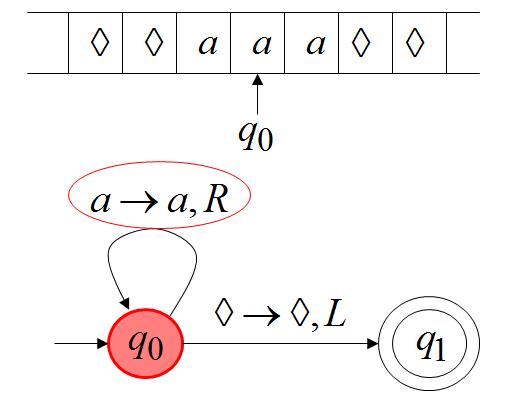
Yukarıdaki bu temsili resimde verilen turing makinesi çizilmiştir.

Makinemizin örnek çalışmasını ve bant durumunu adım adım inceleyelim.

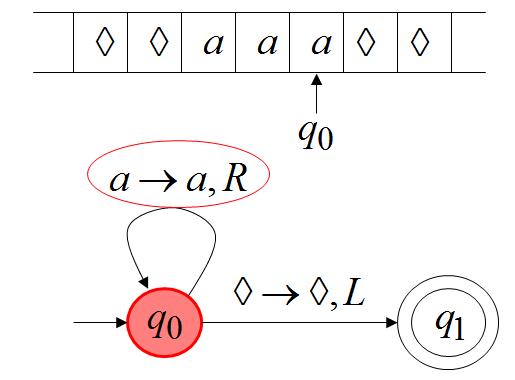
Birinci adımda bandımızda aaa (3 adet a) yazılı olduğunu kabul edelim ve makinemizin bu aaa değerini kabul edip etmeyeceğini adım adım görelim. Zaten istediğimiz de aaa değerini kabul eden bir makine yapabilmekti.



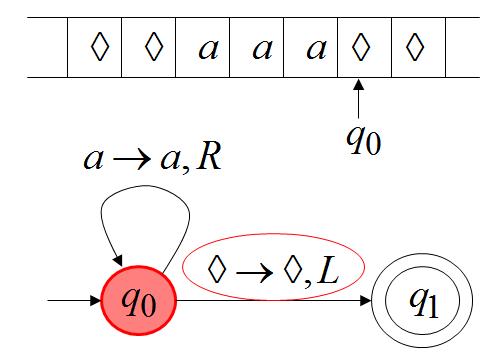
Yukarıdaki ilk durumda bant üzerinde beklenen ve kabul edilip edilmeyeceği merak edilen değerimiz bulunuyor. Makinemizin kafasının okuduğu değer a sembolü. Makinemizin geçiş tasarımına göre q0 halinde başlıyoruz ve a geldiğinde teybi sağa sarıp yine q0 durumunda kalmamız gerekiyor.



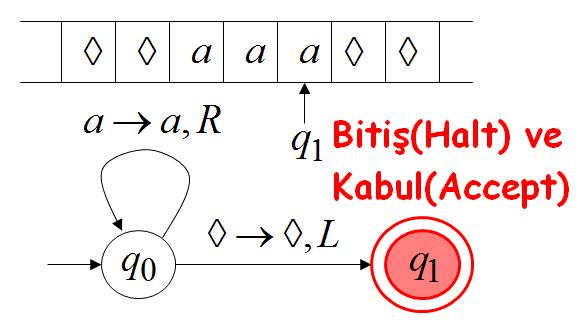
Yeni durumda kafamızın okuduğu değer banttaki 2. a harfi ve bu durumda yine q0 durumundayken teybi sağa sarıp yine q0 durumunda kalmamız tasarlanmış



3. durumda kafamızın okuduğu değer yine a sembolü olmakta ve daha önceki 2 duruma benzer şekilde q0 durumundayken a sembolü okumanın sonucu olarak teybi sağa sarıp q0 durumunda sabit kalıyoruz.



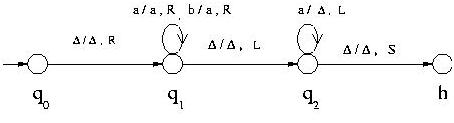
4. adımda teypten okuduğumuz değer boşluk sembolü x oluyor. Bu değer makinemizin tasarımında q1 durumuna gitmemiz olarak tasarlanmış ve teybe sola sarma emri veriyoruz.



Makinenin son durumunda q1 durumu makinenin kabul ve bitiş durumu olarak tasarlanmıştı ( makinenin tasarımındaki F kümesi) dolayısıyla çalışmamız burada sonlanmış ve giriş olarak aaa girdisini kabul etmiş oluyoruz.

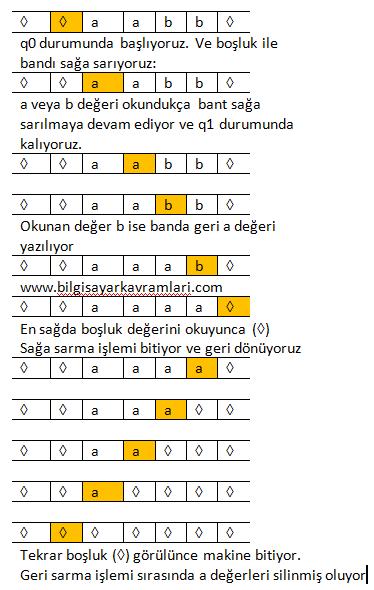
**2. Örnek**

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 bantta 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 bantta 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.



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-12: Özyineli Sayılabilir Diller (Recursively Enumerable Languages)**

[Muntazam dillerden (formal languages)](http://www.bilgisayarkavramlari.com/2009/06/25/muntazam-diller-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)](http://www.bilgisayarkavramlari.com/2009/06/27/chomsky-hiyerarsisi-chomsky-hierarchy/) 0. seviye olan (Type 0) bu dile uygun bütün diller birer düzenli ifade (regular expression) ile gösterilebilir.

[Muntazam dil (formal language)](http://www.bilgisayarkavramlari.com/2009/06/25/muntazam-diller-formal-languages/) olması dolayısıyla dilde üretilen her [özyineli sayılabilir kelime kümesi (recursively enumerable set)](http://www.bilgisayarkavramlari.com/2009/06/25/ozyineli-sayilabilir-kume-recursively-enumerable-sets/) bütün kelimelerden çıkarılabilecek [güç kümesinin (power set)](http://www.bilgisayarkavramlari.com/2009/06/24/guc-kumesi-kuvvet-kumesi-power-set/) bir alt kümesi olmak zorundadır. Bu anlamda bir özyineli sayılabilir dili aşağıdaki üç farklı tanımla tanımlamak mümkündür:

Bir dilde bulunan alfabeden üretilebilen bütük kelimeleri kabul eden dil yani Σ sembolü ile dilimizdeki alfabeyi yani harfler kümesini gösterecek olursak L ile gösterilen dilimiz Σ\* ile üretilebilen kelimeler kümesidir.

[Turing makinesi (Turing machine)](http://www.bilgisayarkavramlari.com/2009/06/27/turing-makinesi-turing-machine/) ile işlenebilen veya [hesaplanabilir bir fonksiyon (computable function)](http://www.bilgisayarkavramlari.com/2009/06/25/hesaplanabilir-fonksiyon-computable-function/) bulunabilen dildir. Burada dikkat edilecek nokta dilin sonsuz olabileceğidir. Yani dilimizde sonsuz sayıda tekrar bulunabilir. [Turing makinesi](http://www.bilgisayarkavramlari.com/2009/06/27/turing-makinesi-turing-machine/) ve [hesaplanabilirlik teorisi (computability theory)](http://www.bilgisayarkavramlari.com/2009/06/25/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.

Şayt bir dilde üretilen bir [dizgi (string)](http://www.bilgisayarkavramlari.com/2008/08/02/dizgi-string/) için bir [Turing makinesi (turing machine)](http://www.bilgisayarkavramlari.com/2009/06/27/turing-makinesi-turing-machine/) veya [hesaplanabilir bir fonksiyon (computable function)](http://www.bilgisayarkavramlari.com/2009/06/25/hesaplanabilir-fonksiyon-computable-function/) bulunuyorsa, diğer bir ifadeyle ürettiğimiz [turing makinesi](http://www.bilgisayarkavramlari.com/2009/06/27/turing-makinesi-turing-machine/) şu üç durumdan birisini yapıyorsa:

* [bitmek (halt)](http://www.bilgisayarkavramlari.com/2008/11/12/durma-problemi-halting-problem/)
* [döngü (loop)](http://www.bilgisayarkavramlari.com/2008/09/08/dongu-loop/)
* red etmek (reject)

bu durumda bu dil özyineli sayılabilir dildir denilebilir.

Özyineli sayılabilir diller temel olarak [chomsky hiyerarşisinde (chomsky hierarchy)](http://www.bilgisayarkavramlari.com/2009/06/27/chomsky-hiyerarsisi-chomsky-hierarchy/) bulunan bütün diğer dilleri kapsar. Bu anlamda hiyerarşinin en alt seviye dilidir ve diğer dillere göre çok daha esnektir.

## **SORU-13: Chomsky Hiyerarşisi ( Chomsky Hierarchy )**

Bilgisayar bilimlerinin özellikle dil alanında yapılan çalışmalarında [muntazam dilleri (formal languages)](http://www.bilgisayarkavramlari.com/2009/06/25/muntazam-diller-formal-languages/) tasnif etmek için kullanılan bir yapıdır. Literatürde Chomsky–Schützenberger hiyerarşisi olarak da geçmektedir.

Bilindiği üzere ( [muntazam diller (formal langauges)](http://www.bilgisayarkavramlari.com/2009/06/25/muntazam-diller-formal-languages/) veya [CFG](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) yazısından da okunabileceği üzere) [muntazam dillerin](http://www.bilgisayarkavramlari.com/2009/06/25/muntazam-diller-formal-languages/) dört özelliği bulunur. Bunlar özellikle [içerikten bağımsız dillerin (context free languages)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-dil-context-free-language/) da temelini oluşturmuştur.

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

Örneğin aşağıdaki [içerikten bağımsız dilbilgisini (context free grammer)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) ele alalım

S → AB

A → Aa | ε

B → b

Yukarıdaki bu [CFG](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) tanımındaki sonlular (terminals) {a,b,ε } ,  devamlılar (nonterminals) { S,A,B } olarak tanımlanır. üretim kuralı olarak (production rules) S,A,B’nin açılımlarını gösteren ve → 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](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) şayet [düzenli ifadeye (regular expression)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) çevrilirse a\*b şeklinde yazılabilir. Aslında burada anlatılan değer anb 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)](http://www.bilgisayarkavramlari.com/2009/06/25/muntazam-diller-formal-languages/) için bir sınıflandırmaya gitmiş ve 4 seviye belirlemiştir.

* Type-0 ( tip 0) Sınırlandırılmamış diller (unrestricted grammers)
* Type-1 ( tip 1) İçerik duyarlı diller (context-sensitive grammers)
* Type-2 ( tip 2) [İçerikten bağımsız diller (context free languages)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-dil-context-free-language/)
* Type-3 ( tip 3) [Düzenli diller (regular grammers)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/)

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 bastiçe [turing makinesi (turing machine)](http://www.bilgisayarkavramlari.com/2009/06/27/turing-makinesi-turing-machine/) tarafından çalıştırılabilen ve bir şekilde bitecek olan dillerdir ([hesaplanabilir diller (computable languages)](http://www.bilgisayarkavramlari.com/2009/06/25/hesaplanabilirlik-teorisi-computability-theory/). Örneğin [özyineli sayılabilir diller (recusive enumerable languages)](http://www.bilgisayarkavramlari.com/2009/06/27/ozyineli-sayilabilir-diller-recursively-enumerable-languages/) bu seviyeden kabul edilir.

Tip-1 dilleri için içerk duyarlı diller örnek gösterilebilir. Basitçe αAβ → αγβ 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 → ε

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](http://www.bilgisayarkavramlari.com/2009/06/27/turing-makinesi-turing-machine/) özel bir halidir ve [Turing makinesinde](http://www.bilgisayarkavramlari.com/2009/06/27/turing-makinesi-turing-machine/) bulunan bantın sabit uzunlukta olduğu (çalışmanın sabit zaman sonra sona ereceği) kabul edilir.

Tip-2  diller ise [CFG](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) ile ifade edilebilen [içerikten bağımsız dillerdir (context free languages)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-dil-context-free-language/). Bu dillerin özelliği A → γ şeklinde gösterilmeleridir. Buradaki γ değeri sonlular (terminals) ve devamlılar (nonterminals) olabilmektedir. Bu diller [aşağı sürüklemeli otomatlar (push down automata PDA)](http://www.bilgisayarkavramlari.com/2007/05/04/pda-push-down-automata-asagi-suruklemeli-otomat/) tarafından kabul edilen dillerdir ve hemen hemen bütün programlama dillerinin temelini oluşturmaktadırlar. (programlama dillerinin neredeyse tamamı bu seviye kurallarına uymaktadırlar)

Tip-3 diller ise [düzeli ifadeler (regular expressions)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) ile ifade edilebilen (veya üretilebilen veya parçalanabilen (parse) ) dillerdir. Bu dillerin farkı

A → α ve

A → αB

şeklindeki kurallar ile göterilebilmeleridir.

Yukarıdaki bu tanımları bir tabloda toplamak gerekirse:

|  |  |  |  |
| --- | --- | --- | --- |
| **Seviye** | **Dil Örneği** | **Otomat Uygulaması** | **Kuralları** |
| Type-0 | [Recursively enumerable](http://www.bilgisayarkavramlari.com/2009/06/27/ozyineli-sayilabilir-diller-recursively-enumerable-languages/) | [Turing machine](http://www.bilgisayarkavramlari.com/2009/06/27/turing-makinesi-turing-machine/) | α → β |
| Type-1 | [Context-sensitive](http://en.wikipedia.org/wiki/Context-sensitive_grammar) | Linear-bounded non-deterministic Turing machine | αAβ → αγβ |
| Type-2 | [Context-free](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) | Non-deterministic [pushdown automaton](http://www.bilgisayarkavramlari.com/2007/05/04/pda-push-down-automata-asagi-suruklemeli-otomat/) | A → γ |
| Type-3 | [Regular](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) | Finite state automaton | A → α ve A → αB |

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

## **SORU-14: Muntazam Diller (Formal Languages)**

Kısaca istisnası olmayan dillere muntazam dil diyebiliriz.

Muntazam diller bilgisayar bilimlerinde, mantıkta ve dilbilim (linguistic) çalışmalarında kullanılan bir dil ailesidir. Dilde bulunan bütün öğeler ve dilin ulaşabileceği sınırlar belirli kurallar dahilinde tanımlanabiliyorsa bu dillere muntazam dil ismi verilir.

Bu anlamda bilgisayar bilimlerinde bulunan bütün programlama dillerini bu ailede düşünmek mümkündür.

Temel olarak bir dildeki harfleri bu harflerden oluşabilecek [kelimelerin oluşma kurallarını (morphology)](http://www.bilgisayarkavramlari.com/2008/03/19/kelime-bilim-lexicology-vocabulary/) ve bu [kelimelerin dizilimini (syntax)](http://www.bilgisayarkavramlari.com/2009/06/17/sozdizim-syntax/) muntazam bir şekilde belirleyebilen kurallarımız varsa. Ve bu kuralların istisnası yoksa bu dilin bir muntazam dil olduğu söylenebilir.

Muntazam dilimizi tanımlamak için öncelikle dilin en ufak öğesi olan harflerden başlamak gerekir.

**Alfabe ve Harf tanımı**

Bir dilde kullanılan sembollerin her birisine harf denilir. Bu harflerin listesine alfabe (bazı kaynaklarda abece) denilir.

Örneğin [haber mantığını (predicate logic)](http://www.bilgisayarkavramlari.com/2009/05/07/haber-predicate/) ele alacak olursak bu mantığın ifade ettiği dil için ∧, ¬, ∀ , ) , ( sembolleri kullanılır. Ayrıca mantığımızdaki değişkenleri ve önermeleri gösteren  x0, x1, x2, … şeklinde sonsuz harf bulunmaktadır.

**Kelime (word) tanımı**

Bir alfabedeki sembollerin (harflerin) o dildeki kelimebilim(morphology) kurallarına uygun olarak oluşturduğu dizilime kelime ismi verilir. Kelimeler dildeki anlamlı en küçük birimlerdir ve tek başlarına sadece bir birimlik bilgi ifade ederler. Daha karmaşık bilgi ifadeleri için cümlelere ihtiyaç duyulur. Ne yazık ki dilbilimde bir kelimenin anlattığı bilgi ile bir cümlenin anlattığı bilgi arasında kesin bir çizgi bulunmamaktadır. Örneğin bazı doğal dillerde tek kelime ile anlatılan bir duygu veya olay, başka dillere bir kelime grubu hatta bazan bir cümle ile çevrilebilmektedir.

Muntazam diller için de aynı durum söz konusudur. Dilin tasarımında hangi bilginin yeterli varlık oluşturacağı ve ne kadar detaya inileceği kararlaştırılır.

Hatta bir dilin var olması için sözdizim (syntax) seviyesinin bulunması gerekmez. Yani sadece kelimeler olan ve kelimelerin dizilmesi gerekmeyen diller de bulunabilir.

Bir dilde üretilebilecek kelimelerin belirli kurallarla tanımlanması ve bu kuralların dışına çıkan istisna bulunmaması durumunda bu dile muntazam dil denilebilir.

Örneğin [düzenli ifadeleri (regular expressions)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) ele alırsak, bu ifadelerin kullanılması ile dilde oluşabilecek sonsuz sayıdaki kelime belirlenebilir.

(a + b)\* şeklindeki bir düzenli ifadeden a ve b harfleri kullanılarak (istenilen sıra ve istenilen sayıda) kleene yıldızı sayesinde boş kümeyi de kapsayan bütün kelimeler üretilebilir.

**Muntazam Dil tanımı**

Yukarıdaki dilin öğelerini tanımladıktan sonra dili tanımlamak mümkündür.

Σ sembolü ile dilimizdeki alfabeyi yani harfler kümesini gösterecek olursak L ile gösterilen dilimiz Σ\* ile üretilebilen kelimeler kümesidir.

Bu tanıma ilave olarak muntazam dil kavramının matematik ve bilgisayar bilimlerinde geçen pek çok “dil” kelimesini karşıladığını söyleyebiliriz. Yani aslında matematik ve bilgisayar bilimlerinde geçen “dil” kelimesi aslında tam olarak “muntazam dile” karşılık gelmektedir.

Örneğin [içerikten bağımsız dil (context free language)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-dil-context-free-language/) terimi aslında içerikten bağımsız muntazam dil anlamında kullanılmaktadır. Benzer şekilde düzenli diller (regular languages) terimi aslında düzenli muntazam dil (regular formal languages) anlamında kullanılmaktadır. Dolayısıyla matematik ve bilgisayar bilimlerinin uğraştığı bütün dilleri ve kullandıkları bütün dilleri muntazam dil (formal language) olarak tanımlamak mümkündür. Bunun tek istisnası bilgisayar bilimlerindeki yapay zeka konusunun altında geçen doğal dil işleme konusudur.

**Muntazam dil çalışmaları**

Muntazam diller üzerinde yapılan çalışmalardan birincisi bir muntazam dilin nasıl tanımlanacağıdır. Bu alanda yapılan bazı çalışmalar sonucunda elde edilen yaklaşımları aşağıdaki şekilde sayabiliriz:

* Muntazam bir dilbilgisi (formal grammer) tarafından üretilen dillere Muntazam dil (formal language) ismi verilir. Bu tanım [Chomsky Hiyerarşisi](http://www.bilgisayarkavramlari.com/2009/06/27/chomsky-hiyerarsisi-chomsky-hierarchy/) tarafından sınıflandırılan dilbilgileri ile gösterilebilir.
* [Düzenli ifadelerin (Regular expressions)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) bir örneği ile üretilebilen dillere muntazam dil ismi verilir.
* Bir otomat tarafından üretilen ([sonlu durum otomatı (finite state automat)](http://www.bilgisayarkavramlari.com/2008/08/02/sonlu-ototmatlar-finite-automaton/) veya [Turing Makinesi (Turing Machine)](http://www.bilgisayarkavramlari.com/2009/06/27/turing-makinesi-turing-machine/) gibi) dillere muntazam dil ismi verilir.

Yukarıdaki bu tanımlama çalışmalarının yanında muntazam diller ile ilgili aşağıdaki sorulara da cevap aranmaktadır.

* Tanımlama yeteneği. Bir dilin tanımlayabilme gücü ve tanımlayabildiği dilin büyüklüğü nedir? Aynı güce ve yeteneğe sahip ikinci bir dil farklı bir şekilde olabilir  mi? Bütün dilleri gösteren bir dil yapılabilir mi? şeklindeki soruları bu grup altında toplayabiliriz.
* Algılama yeteneği. Verilen bir kelimenin verilen bir dile ait olup olmadığının algılanması yeteneğidir. Elimizde muntazam bir dilbilgisi (grammer) olduğunu kabul edelim. Bu dilden olan veya olmayan bir kelime verildiğinde bu kelimenin analiz edilmesi ve dile ait olup olmadığının algılanması en verimli nasıl yapılabilir? sorusuna cevap arayan çalışmalardır.
* Karşılaştırma yeteneği. Elimizde iki dil bulunsun. Bu iki dilin birbirine göre farklarının algılanılması, bir dilde olup diğer dilde olmayan veya iki dilde ortak olan kelime ve cümlelerin tespiti gibi konularda çalışılan çalışma grubudur.

Yukarıdaki bu problemler genel olarak algoritma analizi (analysis of algorithms) altındaki karmaşıklık teorisi (complexity theory) alanın çalışma konularına girmektedir.

**Diller üzerinde tanımlı işlemler**

Diller üzerinde de farklı işlemlerin yapılması mümkündür. Aşağıda bu işlemler ve tanımları verilmiştir:

Dillerin kesişimi (intersection): Temel olarak bir dili o dildeki kurallar ve alfabe marifetiyle üretilebilecek kelimeler kümesi olarak tanımlarsak, iki dilin kesişimi aslında iki dilde üretilebilecek kelimelerin kesişimi olmuş olur.L1 ∩ L2 şeklinde gösterilir.

Üleştirme işlemi (concetanation): İki dilden üleştirme yapmak için birinci dilden bir kelime ile ikinci dilden bir kelimenin arka arkaya eklenmesi ile yeni bir kelime oluşturulması kastedilir. Yani L1 dilinden k kelimesi ile L2 dilinden l kelimesi alınıp kl veya lk şeklinde kelime üretme işlemidir. L1L2 şeklinde gösterilir.

Tümleyen işlemi (Complement): Bir dildeki [güç kümesinden (power set)](http://www.bilgisayarkavramlari.com/2009/06/24/guc-kumesi-kuvvet-kumesi-power-set/) o dildeki dilbilgisine (grammer) göre çıkarılabilen kelimelerin farkıdır. Yani alfabede olan harflerle yazılabilecek bütün kelimelerin, dilde izin verilen kelimelerle [fark kümesidir](http://www.bilgisayarkavramlari.com/2009/06/24/kume-teorisi-set-theory/). ¬L şeklinde veya L şeklinde gösterilir.

Kleene Yıldızı (Kleene Star): Bir dildeki alfabe üzerinde boş küme dahil olmak üzere üretilebilecek bütün kelimeler kümesidir. Bir anlamda güç kümesi (power set) ismi de verilebilir. Kullanımı aslında [düzenli ifadelerden (regular expressions)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) alınmıştır.

Ters işlemi (reverse): Bir dildeki herhangi bir kelimenin yazılışının terse çevrilmiş halidir. Örneğin “abc”‘nin tersi “cba” şeklindedir. Daha akademik olarak iki kuralla tanımlanabilir.

* Şayet dilimizdeki boş kelimeyi e sembolü ile gösterirsek eR = e olur yani boş kelimenin tersi yine boş kelimedir.
* Boş kelime dışındaki herhangi bir kelime için  w = x1…xn şeklinde n adet harf ile yazıldığını düşünelim, bu durumda tersi  wR = xn…x1, olur

Bu durumda muntazam bir dil  L için , LR = {wR | w ∈ L} olarak tanım yapılabilir.

## **SORU-15: Chomsky Normal Şekili (Chomsky Normal Form)**

Bilgisayar bilimlerinde özellikle otomatlar (automata) ve dil tasarımında (compiler design) oldukça sık kullanılan konulardan birisi de [içerikten bağımsız dilbilgisidir. (Context Free Grammer)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/)

[İçerikten bağımsız dil (Context Free Language)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-dil-context-free-language/) konusunda yapılan çalışamlar gelişen ihtiyaçlar ilave bazı kurallar konulmasını gerektirmiştir. Bu konuda çalışan Naom Chomsky tarafından konulan kurallara CNF veya Chomsky normal şekli ismi verilir ve aşağıda açıklanmıştır.

CNF’in tanımı ve kuralları

Şayet bir CFG aşağıdaki kurallara uyuyorsa bu dilbilgisine CNF’e uygun denilebilir. Örneğin bir dilde

Arightarrow, BC veya

Arightarrow, α veya

Srightarrow, λ

şeklinde kurallara uyuyorsa. (Burada ABC birer devamlı (nonterminal) α bir sonlu (terminal) ve λ ise boş dizgi (empty string) gösteren sembollerdir). Burada B ve C sembolleri başlangıç sembolü  olamaz. Yani

~~B rightarrow, α~~

gibi bir kural dilbilgisinde bulunamaz. Benzer şekilde

~~Arightarrow,  BCC~~

şeklinde bir gösterim de CNF kurallarına aykırıdır (en fazla 2 devamlı (nonterminal) bulunabilir).

Yukarıdaki tanımdan anlaşılacağı üzere CNF kulllarına uygun olan her dil zaten CFG’dir. Ayrıca yine yukarıdaki tanıma dayanarak bir dildeki bir cümlenin parçalanması (parse) işlemi sırasındaki her adım, kendinden bir önceki adıma göre en fazla bir harf uzun olabilir. Bunun sebebi yukarıdaki şekilde yazılan bir dilde en fazla 2 devamlı (nonterminal) bulunacağı ve dolayısıyla en fazla 1 harf ilave edileceğidir.

Diğer bir ifadeyle CNF’e uygun bir dil parçalaranar bir parçalama ağacı (parse tree) oluşturulduğunda ortaya bir [ikili ağaç (binary tree)](http://www.bilgisayarkavramlari.com/2008/05/07/ikili-agac-binary-tree/) çıkar ve bu ağacın en derin olma durumunda parçalanan kelimenin boyutundan 1 eksik (ilk devamlı açılımının (nonterminal) 2 elemanlı olduğu düşünülürse) ve en sığ olma durumunda da yine kelime boyutundan 1 eksik olacaktır. Daha kesin bir ifadeyle CNF’e uygun bir dil ile parçalanan bir kelime ve oluşan [parçalama ağacında (parse tree)](http://www.bilgisayarkavramlari.com/2008/12/09/parcalama-agaci-parse-tree/) belirsizlik (ambiguity) bulunamaz ve her zaman tek bir ağaç çıkar.

**Bir CFG’nin CNF’e çevrimi**

Yukarıdaki açıklamalar ışığında her CFG’nin CNF yapısında olmadığını öğrendik. Ayrıca her CFG’nin CNF yapısına çevrilebileceği de bir gerçektir. Bu durumda herhangi bir CFG’nin CNF’e çevrimi için aşağıdaki 5 farklı durum ve bu 5 farklı durumun çözümü verilmiştir.

**1. Durumda** başlangıcın sağ tarafta bulunmasına karşılık ilave bir başlangıç devamlısı (nonterminal) eklenir.

Örneğin dilimiz aşağıdaki şekilde olsun:

Srightarrow, ASA | aB

Arightarrow, B | S

Brightarrow, b | λ

Yukarıdaki dilbilgisi tanımında görüleceği üzere iki noktada S yani başlangıç devamlısı (nonterminal) sağ tarafta bulunmaktadır. Çözüm olarak ilave bir devamlı eklenerek aşağıdaki hale getirilebilir:

S0rightarrow,  S

Srightarrow, ASA | aB

Arightarrow, B | S

Brightarrow, b | λ

Yukarıdaki bu yeni haliyle başlangıç geçişlisi artık S0 olmuştur ve artık sağ tarafta istenmeyen bu terim bulunmamaktadır.

**2. durumda** lambda (λ) terimlerinin temizlenmesi gerekir. Bu problemin çözümü için devamlılardan (nonterminal) başlanarak alternatif sonuçlar sıralanır. Örneğin bir önceki adımda oluşturduğumuz grameri ele alacak olursak:

S0rightarrow,  S

Srightarrow, ASA | aB

Arightarrow, B | S

Brightarrow, b | λ

Yukarıdaki gramerde B -> λ terimi bulunmaktadır. Bunu kaldırmak için B devamlısının (nonterminal) kullanıldığı yerlerde değişiklik yapmak gerekir.

S0rightarrow,  S

Srightarrow, ASA | aB | AS | SA | S | a

Arightarrow, B | S

Brightarrow, b

Yukarıdaki S-> AS | SA | S | a terimleri yeni eklenmiştir. Bu terimler B->λ terimi ile üretilebilecek bütün alternatifleri kapsar.

**3. Durumda** tek terimlerin kaldırılması için tek terimlilerde ulaşılabilen terimler tekrarlanır. Örneğin yukarıdaki gramerin son halinden devam edecek olursak:

S0rightarrow,  S

Srightarrow, ASA | aB | AS | SA | S | a

Arightarrow, B | S

Brightarrow, b

Gramerinde tek terili olarak S0rightarrow,  S , Arightarrow, B ve Arightarrow, S durumları bulunmaktadır. Bu durumların kaldırılması için terim değerleri tekrarlanır :

S0rightarrow,  ASA | aB | AS | SA | a  
Srightarrow,  ASA | aB | AS | SA | a  
A rightarrow, b | ASA | aB | AS | SA | a  
B rightarrow, b

Yukarıdaki son halinde S ve B değeri (okun sağ tarafında olan değerler) olduğu gibi kopyalanmıştır.

**4. durumda** şayet bir devamlının (nonterminal) tanımında karışık bir durum varsa (yani bir sonluyu (terminal) ifade eden terim karışıksa) basitletştirmek için ilave bir devamlı (nonterminal) eklenir. Yine yukarıdaki dilden devam edecek olursak:

S0rightarrow,  ASA | aB | AS | SA | a  
Srightarrow,  ASA | aB | AS | SA | a  
A rightarrow, b | ASA | aB | AS | SA | a  
B rightarrow, b

Gramerinde “a” sonlusunun (terminal) değerinin B devamlısı (nonterminal) ile yanyana durduğu aB terimini görüyoruz. Bu karışık bir durumdur ve çözümü için yeni bir devamlı (nonterminal) ilave edilir.

S0rightarrow,  ASA | UB | AS | SA | a  
Srightarrow,  ASA | UB | AS | SA | a  
A rightarrow, b | ASA | UB | AS | SA | a  
B rightarrow, b

U rightarrow, a

Yukarıdaki şekilde karışık olan bütün aB terimleri düzeltilmiştir.

**5. durumda** ise uzun terimlerin kısaltılması söz konusudur. Yani okun sağ tarafında en fazla iki devamlı (nonterminal) bir terimde bulunabilir. Örneğin ASA gibi üç devamlı (nonterminal) bulunduğu durumlar CNF için uygun değildir.

S0rightarrow,  ASA | UB | AS | SA | a  
Srightarrow,  ASA | UB | AS | SA | a  
A rightarrow, b | ASA | UB | AS | SA | a  
B rightarrow, b

U rightarrow, a

dilini ele alırsak

S0 rightarrow, AA1 | UB | AS | SA | a  
S rightarrow, AA1 | UB | AS | SA | a  
Arightarrow,  b | AA1 | UB | AS | SA | a  
B rightarrow, b  
U rightarrow, a  
A1 rightarrow, SA

Tek problemli olan ASA terimi yerine ilave bir devamlı (nonterminal) eklenerek yukarıdaki şekilde düzeltilebilir.

## **SORU-16: Sözdizim (Syntax)**

Temel olarak bir dilde (language) tanımlı olan öğelerin (kelime, işlem, sembol yada değerlerin) anlamlı bir dizilim oluşturmasıyla ilgilenen bilimdir.

**Örnekler**

Örneğin Türkçe için aşağıdaki cümle anlamlı bir cümledir:

"Ali okula geldi"

yukarıdaki cümlede herhangi bir yazım hatası ve dizilim sorunu bulunmamaktadır. Bununla birlikte:

"okula Ali geldi"

cümlesi de Türkçe açısından uygun bir cümle iken

"geldi okula Ali"

cümlesi hem devrik hem de anlamını yitirmiş bir cümledir. Benzer durum matematiksel işlemler için de aşağıdaki şekilde gösterilebilir:

( 3 + 4 )

Yukarıdaki matematiksel işlem dizilim açısından sorunsuzken:

( 3 4 + )

Yukarıdaki halde de yazılabilir (postfix notation).

3 ) ( 4 +

Ancak yukarıdaki yazılış dizilim açısından hatalıdır.

Yukarıdaki örneklerde görüldüğü üzere her dilin kendisine göre bir dizilimi ve dizilime göre farklı anlamları bulunmaktadır. İşte sözdizim (syntax) bu konuda çalışmaktadır.

**Anlam çıkarımı**

Sözdizim çalışmalarının önemli bir kısmı da dilin işlendiği bilgisayar tarafından girilen cümleye (Statement) anlam verilmesidir.  Cümlelerin anlamlarını çalışan [anlambilim (Semantics)](http://www.bilgisayarkavramlari.com/2009/06/17/anlambilimsel-tertip-semantic-composition/) açısından genelde cümlenin bir anlamını gösteren model bulunmakta ve anlam çıkarımında hedeflenen amaç girilen cümleyi bu modele çevirmke olmaktadır.

Bu çevirim için çeşitli yaklaşımlar mevcuttur. Örneği [yacc](http://www.bilgisayarkavramlari.com/2008/12/12/yacc/) gibi sözdizim (syntax) analizi yapan ve makine dilleri (machine languages) için başarılı bir şekilde çalışan ve yapısında [içerik bağımsız dil (context free language)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) barındıran, kod üretme  programları bulunmaktadır. [Yacc](http://www.bilgisayarkavramlari.com/2008/12/12/yacc/) ve benzeri kod üreticilerinede (code builder) hedeflenen dil genelde [CFG](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) ile gösterilebilen bir makine dilidir ve başarı oranı oldukça yüksektir.

Buna karşılık doğal dillerin analizi ve [semantik gösterimi (semantic representation)](http://www.bilgisayarkavramlari.com/2009/06/17/anlambilimsel-tertip-semantic-composition/) için tam başarı ne yazık ki neredeyse imkansızdır.  Ayrıca doğal dilin analizi sonucunda elde edilen sonucun gösteriminde de ciddi problemler bulunmaktadır.

**Cümle üretimi (Syntax Generation)**

Anlam çıkarımına benzer şekilde, ifade edilmek istenen anlamın üretilmesi de bir problemdir. Örneğin 4 sayısını toplama işlemi ve pozitif tamsayılar kümesi ile göstermek istersek:

1+3

2+2

3+1

gösterilmerinin tamamı üretmek istediğimiz değer olabilir. Benzer şekilde doğal dil üretimi (natural language generation) yukarıdaki örneğe göre oldukça karmaşık bir üretim yapısını desteklemek zorundadır. (Yukarıdaki örnekte yaşanan [karmaşıklık (ambiguity)](http://www.bilgisayarkavramlari.com/2008/03/16/belirsizlik-ambiguity-muglaklik-ikircimlik/) çözümü için belirli sonlu otomat (deterministic finite automata) kullanılabilir, burada sadece probleme örnek verilmiştir)

Örneğin Ali’nin okula gelmesinden, ders dinlemesinden ve eve dönmesinden bahsetmek isteyelim. Muhtemel bir bilgisayar çıktısı:

Ali okula geldi. Ali ders dinledi. Ali eve döndü.

Şeklinde olabilir. Ancak dikkat edileceği üzere insanlar konuşurken bu tip cümleler sıralamazlar. Örneğin

Ali okula geldi. O ders dinledi. O eve döndü.

Şeklindeki cümleler daha insana yakındır. ([dönüştü (anaphora resolution)](http://www.bilgisayarkavramlari.com/2008/03/16/donustu-anaphora/) kullanılmıştır)

Benzer şekilde:

Ali okula geldi, ders dinledi ve eve döndü.

Cümlesi insan kullanımına çok daha yakındır ([ellipsis (hazf)](http://www.bilgisayarkavramlari.com/2009/02/27/hazf-eksilti-ellipsis/) kullanılmıştır)

Yukarıda görüldüğü üzere doğal dil üretimi sırasında da söz dizimden faydalanılmaktadır.

**Şablon Yapısı (templates)**

Bu sözdizim çalışması yaklaşımına göre dilde bulunan cümlelerin anlamlarının anlaşılması (veya bu dilde bir cümle üretimi) için çeşitli şablonlar kullanılır. Bir söz dizim çalışmasının en temelini oluşturan şablon kullanımı (çeşitli kaynaklarda çerçeve (Frame) olarak da isimlendirilmektedir). Oldukça katı(rigid) bir yapıya sahiptir. Anlamak için şablona mutlak uyum şarttır. Ürettiği sonuçlar ise ancak şablon yapısında olmaktadır.

**Kategorik Dilbilgisi (Categorical Grammer)**

Sözdizim çalışmalarında dilin modellenmesi sırasında kullanılan bir dilbilgisi örneğdir. Bu yapıda dilde bulunan öğeler anlamsal olarak gruplanır ve modellenir. Örneğin “bahçeli okul” şeklindeki bir tamlamada aslında iki farklı kelime vardır. Kategorik dilbilgisinde bu iki kelimenin oluşturduğu gruba dilbilgisindeki bir kategori ismi verilir.

Genelde çok sık kullanılan kategoriler NP ve VP olarak da geçen isim kelime grubu (noun phrase) ve fiil kelime grubudur. (verb phrase)

**Bağlılık Dilbilgisi (Dependency Grammer)**

Dilbilgisindeki kelime veya kelime gruplarının tanımlanması ve sıralarının belirlenmesine ilave olarak şayet bu kategoriler arasında bir bağlantı tanımı yapılıyorsa bu dilbilgisi çeşidine bağlılık dilbilgisi ismi verilir.

Basitçe bir öznenin, bir fiilin faili (agent) olması durumu bu grup gramer tanımındandır.

Örneğin cebirsel sözdizimler (algebraic syntax) bu gruptan sayılabilir.

**Fonksiyonel Dilbilgisi (Functionalist Grammer)**

Bu dilbilgisi grubunda ise dilde bulunan kelimelerin ve kelime gruplarının kullanım amaçları ve üstlendikleri görevler değerlendirilir.

Örneğin “At, tay doğurdu” cümlesindeki doğurmak fiili üretken bir fiildir ve modellenen dünyaya yeni bir varlık ekler. Buna karşılık “Ali peynir yedi” fiili tüketken bir fiildir ve modellenen dünyadan bir varlık eksilir. Bu durumda iki fiili iki ayrı grupta ele almak mümkündür.

**Stokastik Dizilimler (Stochastic Grammers)**

[İstatistiksel dilbilimin (Probabilistic Linguistic)](http://www.bilgisayarkavramlari.com/2009/06/16/istatistiksel-dilbilim-probabilistic-linguistic/) çalışma alanı olan bu sözdizim çeşidinde çeşitli olasılık ağları (örneğin [markof modelleri (markov chain)](http://www.bilgisayarkavramlari.com/2009/06/17/markof-modeli-markov-model/) ) vasıtasıyla dizilim modellemesi yapılır.

## **SORU-17: Markof Modeli (Markov Model)**

Bilgisayar bilimleri de dahil olmak üzere pekçok bilim ve mühendislik alanında kullanılan markof modelleri aslında graf teorisinin (graph theory) bir uygulamasıdır.

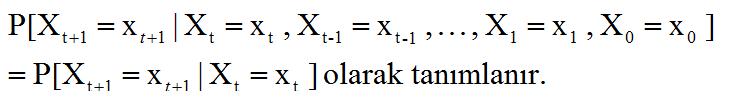
Basitçe [düğümleri (nodes)](http://www.bilgisayarkavramlari.com/2008/08/01/dugum-node/) durumlardan oluşan ve bu durumlar arasında istatistiksel geçişi modelleyen [kenarları (edges)](http://www.bilgisayarkavramlari.com/2008/08/01/kenar-edge/) bulunan graflardır.

Markof modellerine (markof zinciri (markov chain) ismi de kullanılmaktadır) göre bir durum belirli bir istatistiksel değere göre değişir veya değişmeden aynı kalır. Ayrıca geçmiş durumların mevcut durum üzerinde bir etkisi söz konusu değildir. Ancak şimdiki durum gelecek durumları etkileyebilir.

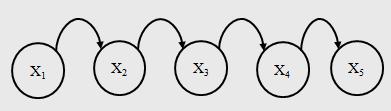
**Markof modelinin tanımı**

Markof modellerinin istatistiksel olma özelliğinden dolayı her bir [stokastik olayın (Stochastic Event)](http://www.bilgisayarkavramlari.com/2009/06/17/stokastik-surec-stochastic-process/) olasılık değerini modelleyen bir gösterimi mümkündür.

Bu olasılıkların gösterildiği formül



Yukarıdaki formülde t+1 zamanındaki olayların t zamanına bağlı olması söz konusudur. Yukarıdaki bu zaman bağlamında bağımlılıktan dolayı markov modellerinin yönlü graf olmaları gerekir.



Yukarıdaki şekilde bu olaylar arası bağlantı [yönlü graf (directed graph)](http://www.bilgisayarkavramlari.com/2008/08/01/kenar-edge/) olarak görülmektedir. Elbette yukarıdaki graf sadece bir örnek olup olayların doğrusal olarak bağlanmaları gerekemez ancak yukarıdaki graftan anlaşılması gereken her olayın bir sonraki olaya belirli bir olasılık değeriyle devam edebileceği veya aynı kalacağıdır. Örneğin t anında X1 olayı oluyor olsun. t+1 zamanında X1 devam edebilir veya X2 olayına geçilebilir.

Yukarıdaki bu gösterimlere ilave olarak markov zincileri birer masfuf (matrix) ile de gösterilebilir.



Örneğin yukarıdaki matriste 4 olay arasındaki geçiş değerleri gösterilmiştir. Matrisin tek yönlü olmasına dikkat edilebilir. Yani köşegen simetriği (diagonal symmetry) mutlaka 0 olmalıdır. Örneğin C’den B’ye olasılık değeri 0.2 iken B’den C’ye 0 olmaktadır. Ayrıca satır bazında ihtimallerin toplamları 1 olmalıdır. Yani satırın ifade ettiği [stokastik olaydan](http://www.bilgisayarkavramlari.com/2009/06/17/stokastik-surec-stochastic-process/) farklı bir stokastik olaya geçişin veya aynı olayda kalmanın ihtimalleri toplamı 1 olmalıdır.

**Markof zincirlerine hava durumu örneği**

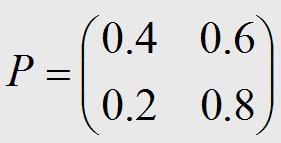
Markof zincirleri tahmin (forecasting) için oldukça kullanışlıdırlar. Örneğin hava durumu tahmini için markof zinciri kullanmak isteyelim ve iki olayımız olsun:

* bugünkü hava
* yarınki hava

şimdi elimizde bugünkü havanın durumu bulunmakta. Bu olaydan yola çıkarak yarınki hava olayını (tahminini) yapmaya çalışalım ve olayı markof modeli ile modelleyelim.

* Bugün yağmur yağıyorsa -> yarın yağmur yağma ihtimali = 0.4
* Bugün yağmur yağıyorsa -> yarın yağmur yağmama ihtimali = 0.6
* Bugün yağmur yağmıyorsa -> yarın yağmur yağma ihtimali = 0.2
* Bugün yağmur yağmıyorsa -> yarın yağmur yağmama ihtimali = 0.8

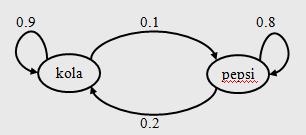
olarak verilmiş olsun. Bu bilgiyi geçmiş tecrübelerden edindiğimizi ve markof modeli ile modellemek istediğimizi düşünelim.



Sonuç olarak yukarıdaki şekilde bir olasılık matrisi elde edilir. Bu matrise, stokastik matris (stokhastic matrix) ismi de verilmektedir.

**Markof zincirleri ile Kola ve Pepsi örneği**

Markof zincirlerinin anlatımı sırasında kullanılan meşhur örneklerden birisi de kola ve pepsi örneğidir. Bu örnekte bir kişinin en son aldığı içeceğin kola ( coca cola ) olması veya pepsi olması durumuna göre bir sonraki içeceğinin tahmin edilmesine çalışılır.

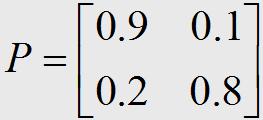


Örneğin bu iki içecek arasındaki ilişki ve karar olasılıkları yukarıdaki şekilde verilmiş olsun. Yani pepsi alan bir kişinin bir sonraki içceğinin yine pepsi olma olasılığı 0.8 ve kola olma olasılığı 0.2, benzer şekilde kola içen birisinin bir sonraki içeceğinin yine kola olma olasılığı 0.9 ve pepsi olma olasılığı 0.1 olarak verilsin.

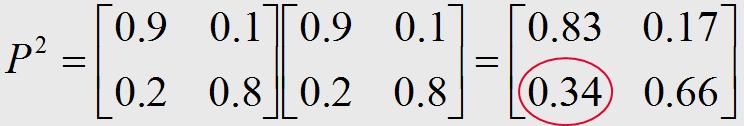
Şimdi sorumuzu soralım:

Pepsi içen bir kişinin ikinci alış verişinde kola alma olsaılığı nedir?

Bu sorunun cevabı için stokastik matrise başvurarak basit bir matris çarpım işlemi yapabiliriz:



Önce yukarıdaki şekilde stokastik matrisimizi çıkaralım ve olasılıkları yerleştirelim. Şimdi sorumuza dönecek olursak bize ikinci alış verişteki ihtimal sorulmuşt. Bu durumda matrisin karesini alalım (kendisi ile çarpalım)

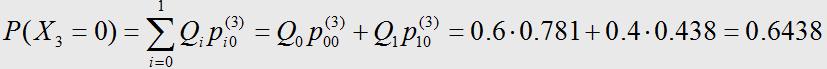


Yukarıdaki p2 matrisinden anlaşılacağı üzere markof modelimizdeki bir olayın tekrar etme olasılığını bulmuş olduk. Kişinin ikinci alışverişinde pepsiden kolaya geçme olasılığı yukarıdaki şekilde 0.34 olarak bulunmuş olur.

Örneğin yukarıdaki bu bilgiler ışığında bize şöyle bir soru da sorulabilir di:

Mevcut durumda insanların %60′nın kola ve %40′ının pepsi içtiklerini düşünelim. Bu insanların haftalık olarak içecek aldıklarını düşünürsek üç hafta sonra insanların ne kadarı kola içiyor olacaktır?

Şimdi bu soruyu çözerken 3 satın alma işlemi yani 3 kere markof modelde değişim işlemi yapılacağını hatırlayalım. Ardından mevcut durumun etkisini de ekleyerek hesabımızı aşağıdaki şekilde yapalım:



Yukarıdaki hesapta öncelikle 3. satın alma işlemi sırasında stokastik matrisin değeri hesaplanmıştır. Yani P3 için matris 3 kere kendisi ile çarpılmıştır. Ayrıca matristen elde edilen katsayılar 0.6 ve 0.4 mevcut durum oranıyla çarpılmıştır. Sonuçta 0.6438 oranında kişinin kola içeceği bulunmuş olunur.

## **SORU-18: Hamilton Yolu (Hamiltonian Path,hamiltonian circuit)**

Bilgisayar bilimlerinde veri modellemede kullanılan graf teorisinde (graph theory) kullanılan bir yol (path) biçimidir. Tanımı oldukça basittir. Buna göre bir [yolun (path)](http://www.bilgisayarkavramlari.com/2008/08/01/yol-path/) Hamilton yolu olabilmesi için bir kere geçilen [kenardan (edge)](http://www.bilgisayarkavramlari.com/2008/08/01/kenar-edge/) tekrar geçilmemesi gerekir ve ayrıca yolun bütün [düğümleri (nodes)](http://www.bilgisayarkavramlari.com/2008/08/01/dugum-node/) birer kere ziyaret etmesi gerekir.

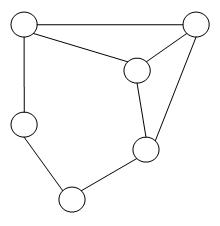
Bir hamilton yolu başladığı yerde bitiyorsa yani tam bir [döngü (cycle)](http://www.bilgisayarkavramlari.com/2008/08/01/dongu-cycle/) tamamlıyorsa (yoldaki son düğümden ilk düğüme gidilmesi mümkünse) bu yollara hamilton döngüsü (hamiltonian cycle) ismi de verilebilir.

Hamiltonian yolları [yönsüz graflarda (undirected graph)](http://www.bilgisayarkavramlari.com/2008/08/01/yonsuz-graflar-undirected-graphs/) tanımlıdır ancak hamilton yollarının benzerleri [yönlü graflar (directed graphs)](http://www.bilgisayarkavramlari.com/2008/08/01/yonlu-graflar-directed-graphs/) için de uyarlanabilir.

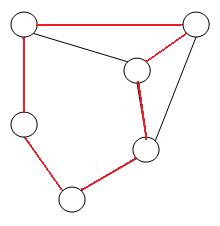
Bir graftaki hamilton yollarının bulunması işlemi [NP-Complete](http://www.bilgisayarkavramlari.com/2008/03/24/belirsiz-cokterimli-tam-np-complete-nondeterministic-polynomial-complete/) bir işlemdir.

**Örnek**

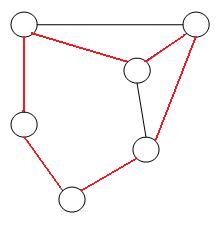
Örneğin aşağıdaki şekli (graph) ele alalım:



Yukarıdaki şekil için tanımımıza uygun bir yol (path) elde etmek istersek:



Yukarıdaki şekilde kırmızı olarak gösterilen yol bir hamiltonian yoludur. Ancak bu şekilde çıkarılabilecek tek yol yukarıdaki yol değildir.



Yukarıdaki bu yeni yol da tanıma uygun bir yoldur ve ilk çıkan yoldan farklıdır. Görüldüğü üzere bir şekilde tek bir hamilton yolu bulunmak zorunda değildir.

**Hamilton yollarının özellikleri**

Herhangi bir hamilton döngüsü (hamiltonian cycle) tek bir [kenarın (edge)](http://www.bilgisayarkavramlari.com/2008/08/01/kenar-edge/) çıkarılmasıyla bir hamilton yoluna (hamiltonian path) dönüştürülebilir.

2 den fazla [düğüme (node)](http://www.bilgisayarkavramlari.com/2008/08/01/dugum-node/) sahip ve [tam bağlı (strongly connected)](http://www.bilgisayarkavramlari.com/2008/08/02/guclu-bagli-graf-strongly-connected-graph/) turnuva çizimleri birer hamilton yoludur (hamiltonian path)

[Tam bağlı graflar (strongly connected graphs)](http://www.bilgisayarkavramlari.com/2008/08/02/guclu-bagli-graf-strongly-connected-graph/) için birbirinden farlı hamilton yollarının (hamiltonian path) sayısı (n-1)!/2 tanedir.

**Hamiltonian yollarının kullanım alanları**

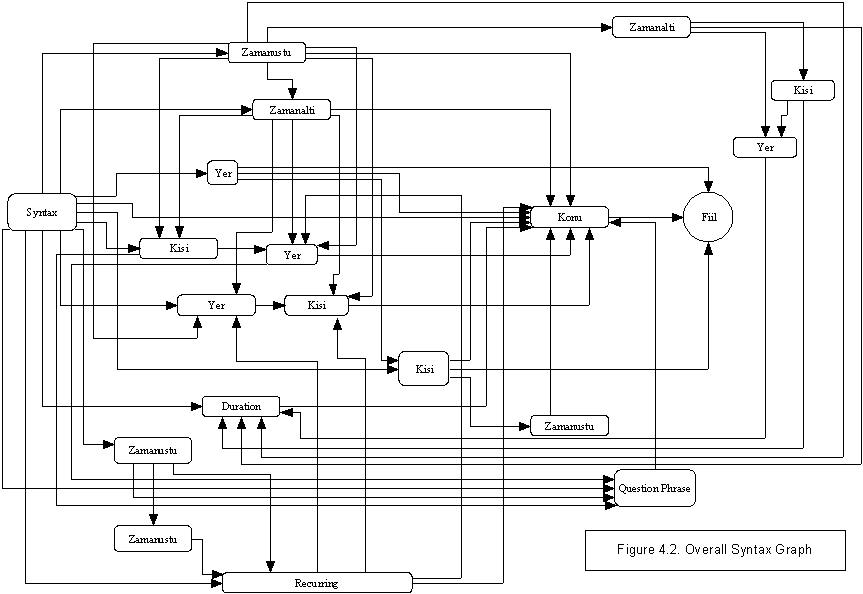
Seyyar satıcı problemi (travelling salesman problem) gibi pekçok graf teori problemin çözümünde kullanılabilirler.

Sıfır bilgi ispatı (zero-knowledge proof) gibi veri güvenliği (cryptography) problemlerinde kullanılabilirler.

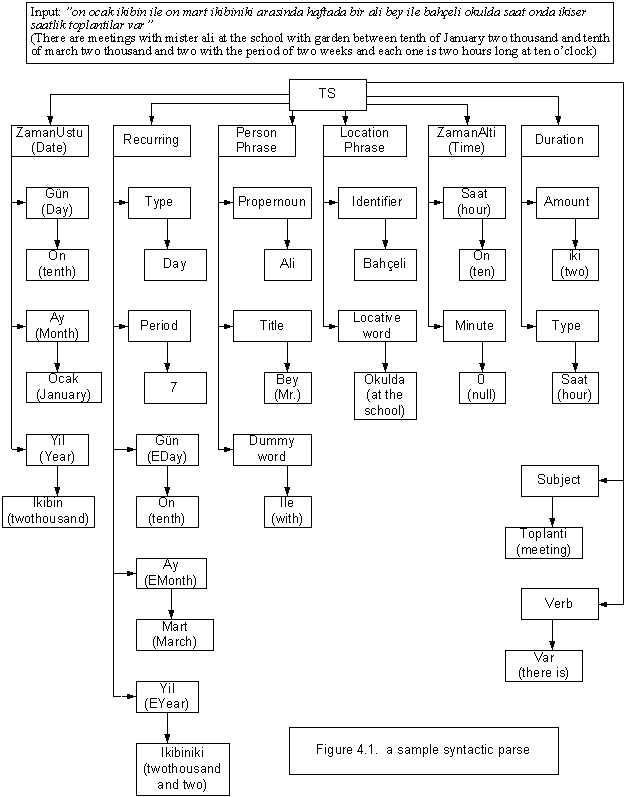
## **SORU-19: Augmented Transition Network (ATN, Uzatılmış Geçiş Ağı)**

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 topalanana kadar tehir etmektir (geciktirmektir, augment). Ağın ismi de buradan gelmektedir.

Temel olarak sonlu otomatları (Finite state automats) baz alan ağlarda her [düğüm (node)](http://www.bilgisayarkavramlari.com/2008/08/01/dugum-node/) bir dilbilimsel (linguistic) bilgiyi içermektedir. [Kirişler (edges, arcs)](http://www.bilgisayarkavramlari.com/2008/08/01/kenar-edge/comment-page-1/) ise bu dilbilimsel üniteler arasındaki geçiş imkanlarını gösterir.



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



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)](http://www.bilgisayarkavramlari.com/2008/12/09/parcalama-agaci-parse-tree/) gösterilmiştir.

## **SORU-20: Knuth Morris Prat Algoritması (KMP Algorithm)**

Knuth-Morris-Prat algoritması bir kelimenin (yada bir metin parçasının) bir metin içerisinde aranmasını sağlayan algoritmadır. Basitçe bu algoritmada bir kelimenin aranan metinde bakılması ve bakıldığı yerde bulunamaması durumunda nerede olabileceği ile ilgili bir bilginin elde edilmesi hedeflenir.

Algoritma aranan kelimenin, aranan metinde bulunmaması durumunda, kelimenin içerisindeki harflerden yola çıkarak birden fazla ihtimali elemektedir.

Klasik bir metin arama işleminde aranan kelime metindeki bütün ihtimallerde denenir. (örneğin [doğrusal arama (linear search)](http://www.bilgisayarkavramlari.com/2008/11/09/dogrusal-arama-linear-search/) bu şekilde çalışır).  KMP algoritmasında ise aranan metindeki bütün ihtimaller denenmez. Bu durumu anlamak için bir örnek üzerinden algoritmayı inceleyelim:

KMP algoritmasının çalışması.

Örneğin metin olarak:

ŞABCDŞADEFABŞADI

Aranan kelime olarak da :

ŞADI

için KMP algoritmasının nasıl çalıştığını inceleyelim.

Öncelikle metin ve aranan kelime aşağıdaki şekilde ilk harflerinden karşılaştırılmaya başlanır:

ŞABCDŞADEFABŞADI

ŞADI

ilk iki harfin tutmasına karşılık 3. ve 4. harfler tutmamaktadır. Algoritma benzemeyen ilk harf olan 3. harfi bulunca geri kalanını benzetmeye çalışmaktan vaz geçer. Ayrıca bakmış olduğu harflerden hiçbirisi aradığı kelime olmadığı için metinde aradığı kelime boyu kadar kayarak yeniden arama yapar.

ŞABCDŞADEFABŞADI

    ŞADI

Yukarıdaki şekilde 4 harf kayarak D harfinden itibaren arama yapmaya başlar çünkü bir önceki aramada, ilk 4 harf arasında aranan kelimenin ilk harfi olan Ş harfinin bulunmadığını anlamış ve artık bu harflere bakmak gerekmediği sonucuna varmıştır.

4. harf olan D harfi ile başlayan metinde, aranan kelime olan ŞADI karşılaştırılmış ve görülmüştür ki metin uymmamaktadır. Ancak aranan metnin bir parçası olan ŞAD kısmı buradaki aranan harfler arasındadır. dolayısıyla algoritma bu sefer 1 harf kaydırarak ve Ş harflerini alt alta getirerek deneme yapar.

ŞABCDŞADEFABŞADI

     ŞADI

Karşılaştırma işlemi sonucunda ilk 3 harf tutmasına karşılık son harfte problem vardır ve karşılaştırılan harflerden hiç birisi aradığımız kelimenin baş harfi olan Ş ile başlamamaktadır. Bu durumda metinde 4 harf daha hareket edilerek bir sonraki karşılaştırma işlemi yapılır:

ŞABCDŞADEFABŞADI

         ŞADI

Karşılaştırmada başarı olmamasına karşılık son harf Ş harfidir ve dolayısıyla bu harfi yakalamak için 3 harf daha kaydırılır:

ŞABCDŞADEFABŞADI

            ŞADI

Bu kez başarı elde edilmiş ve aranan kelime bulunmuştur.

Yukarıdaki örnekte dikkat edilirse toplam 5 karşılaştırma işlemi yapılmıştır. Bu işlem [doğrusal arama algoritması](http://www.bilgisayarkavramlari.com/2008/11/09/dogrusal-arama-linear-search/) ile yapılsaydı 12 karşılaştırma gerekcekti. Bu anlamda KMP algoritmasının daha başarılı olduğu söylenebilir.

Algoritmanın performansı olarak O(n) değeri bulunabilir (n metnin boyutu olarak düşünülürse). Doğrusal arama ile aynı sonucun alındığı algoritma performansı için en kötü durum analizi yapıldığı ve en kötü durumda doğrusal arama ile aynı olacağı unutulmamalıdır.

Daha fazla bilgi için bir benzer algoritma olan [boyer moore algoritmasına](http://www.bilgisayarkavramlari.com/2009/05/19/boyer-moore-dizgi-arama-algoritmasi-boyer-moore-string-search/) bakabilirsiniz.

## **SORU-21: 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 hesapa 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 bölünemezlik (atomicity) geliştirilen ortamda daha düşük seviyeli kontroller ile sağlanır. Örneğin işletim sistemlerinde kullanılan [semafor’lar (semaphores)](http://www.bilgisayarkavramlari.com/2009/03/30/semafor-semaphore-flama-isaret/), 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-22: Tehlike (Hazard)**

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

* Sabit Tehlikeler (Static Hazards)
* Müteharrik Tehlikeler (Dinamik Tehlikeler, Dynamic Hazards)
* Fonksiyonel Tehlikeler (Functional Hazards)

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

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

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

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

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

## **SORU-23: İçerik Bağımsız Gramerler için Pompalama Önsavı (Pumping Lemma for Context Free Grammers)**

Bilgisayar bilimlerinde bir dilin, [içerik bağımsız gramer (context free grammer, CFG)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) ile gösterilemeyeceğini ispatlamaya yarar. Yani pompalama ön savı sayesinde bir dilin CFG olmadığı ispatlanabilir ancak olduğu ispatlanamaz. Şayet pompalama önsavını geçemyorsa CFG değildir denilebilir ancak geçmesi olmasını gerektirmez.

[Pomplama önsavı (pumping lemma)](http://www.bilgisayarkavramlari.com/2009/03/21/pompalama-onsavi-pumping-lemma/) kısaca bir dili aşağıdaki gramere uydurmaya çalışır:

s = uvxyz

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

s  = uv ixy iz

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

**Örnek:**

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

L = {aibici | i > 0}

Yukarıdaki dili, tanımımızdaki

s = uvxyz

şekline getirmeye çalışalım ve bu sırada |vxy| ≤ p, |vy| ≥ 1 şartlarını sağlamaya çalışalım. Bu durumda ilk kelimemiz:

s = a1b1c1

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

v= a

x= b

y= c

olarak yazılabilir. Bunun dışındaki şartların hiç birisi s = uvxyz şartını ve s = aibici | i > 0 şartını aynı anda sağlamaz.

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

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

Böyle bir değer bulunamaz çünkü

s = a2b2c2

değerini ancak

s  = uv ixy iz

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

Benzer şekilde

u= a

v=b

y=c

olması durumunda u,

v=a

y=b

z=c

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

Yukarıdaki durumda L = {aibici | i > 0} dilinin bir CFG gösterimi ile gösterilemeyeceği ispatlanmış olur.

## **SORU-24: Düzenli İfadelerde Pompalama Önsavı (Pumping Lemma for Regular Expressions)**

Bir dilin [Düzenli ifadele (Regular expression)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) olup olmadığının belirlenmesi için kullanılan [pomplama önsavıdı (pumping lemma)](http://www.bilgisayarkavramlari.com/2009/03/21/pompalama-onsavi-pumping-lemma/). Basitçe düzenli ifadede olup olmadığı sınanacak bir w dili için (yani L = w için)

w= xyz

şeklinde bir açılım sınanır. Buradaki sınama sırasında aşağıdaki koşulların sağlanması beklenir:

1. |y| ≥ 1
2. |xy| ≤ p
3. bütün i ≥ 0 için, xyiz ∈ L

Yukarıdaki üç şartı da sağlayan bir dil için düzenli ifadedir denilebilir. Ayrıca bu dilin düzenli ifade olabilmesi için yukarıdaki y teriminin üstü olan i değeri istenildiği kadar arttırılabilmeli ve elde edilen sonuç yine düzenli ifade (Regular expression) olmalıdır.

Daha basit bir ifadeyle, bir dilden üretilen bir terim üç parçaya bölündüğünde ortasındaki parça boş olmamalı, istenildiği kadar tekrarlanabilmeli ve çıkan bütün sonuçlar yine aynı dilden olmalıdır.

**Örnek:**

Pompalama önsavı konusunda en çok verilen örneklerden birisi L = {anbn : n ≥ 0} dilinin bir düzenli ifade olmadığının ispatıdır. Bu örneği beraberce ispatlayalım:

Dilimizi pompalama önsavındaki kalıp olan w = xyz şekline benzetmeye çalışalım. Bu durumda dilimizdeki üretilebilen en kısa terim için w = apbp olduğu söylenebilir ve bu terimi kalıba benzetirken xy için a ve z için b olduğunu söyleyebiliriz. Çünkü y’nin sıfır boyutunda olamayacağını biliyoruz, bu durumda y değeri ya a ya da b olmalıdır. İki durumda birbirinin aynısıdır. Bu durumda

y=a

z=b

olarak kabul edelim.

Şimdi sonucumuzu pompalayalım ve bir sonraki beklentimiz olan xy2z terimini üretelim. Bu durumda çıktımız w = a2b olacaktır. Dikkat edilirse bu yeni terim, dil tanımımız olan L = {anbn : n ≥ 0} tanımı ile çelişmektedir ve bu dilin bir üyesi değildir.

Dolayısıyla L = {anbn : n ≥ 0} dilinin bir düzenli ifade olmadığı veya düzenli ifade olarak yazılamayacağı ispatlanmış olur.

**Örnek Çözüm :**

Mesut Aydın bey efendinin sorusuna istinaden aşağıdaki çözümü yazıyorum:

Sorunun tam metnini kendileri yorum olarak bu sayfada yazmışlar ancak bir kerede buraya alıntılıyorum:

L dili için (x ∈ (0,1)\* ve W=XX^R) Pumping lemma ile regülerliğini araştırın.  
Örneğin : x = 011 olsun x^R =110 olur ve w=011110 olur.

Zannediyorum x^R ile ifade edilmek istenen, x teriminin tersi oluyor.

Şimdi bakın böyle bir soruda doğrudan cevap olarak RE (Regular Expression) olamayacağını söyleyebilirim (biraz tecrübe ile biraz da olaya şu şekilde bakmanız gerekiyor) Bunun sebebi, basitçe herhangi bir RE için sadece bir pompalama noktasına izin verilmesidir. Diyelim ki x için değişim belirttiniz, bu durumda x^R ‘nin buna bağlı değişmesi soruda bir şart olarak tanımlanmış, oysaki biz sadece tek noktada pompalama yapabiliyoruz ve dolayısıyla x değişirken x^R şeklinde verilen ikinci kısma müdahale edemiyoruz.

Daha akademik olarak durumu ifade edecek olursak:  
Yukarıda verilen W non-regular (düzenli olmayan bir ifadedir).

**İspatı** : L dilini (L tanımıyla üretilebilecek kelimeler kümesini) öncelikle düzenli ifade kabul edelim (regular expression)  
Bu kabul doğruysa, n>1 uzunluğunda bir pompalama uzunluğu bulunmalıdır ki W∈L olsun ve her |w| ≥ n  için x = klm yazılabilsin. (Burada klm üç harfini kullandım bunun sebebi aslında yukarıda anlatılan ve yazıda geçen xyz kullanırsak x harfinin sorudaki x ile karışması endişesidir. Soruda x harfi kullanıldığı için (L’den üretilebilen herhangi bir kelime olarak) bu harfi tekrar kullanmadım). Ayrıca bu klm üç harfi için aşağıdaki şartlar da sağlanmalıdır:

* |kl|≤n
* |l|>0
* son olarak klim ∈ L bütün i ≥ 0 değerleri için sağlanmalıdır.

Şimdi yukarıda yazdığımız ve pompalama önsavının tanımındana gelen durumları çürüten bir örnek bulursak, bu L dilinin düzenli ifade olmadığını, [olmayana ergi (burhan-ı mütanakis, proof by contradiction)](http://www.bilgisayarkavramlari.com/2008/08/02/burhan-i-mutenakis-proof-by-contradiction-olmayana-ergi/) yöntemi ile ispat etmiş olacağız:

Örneğin X = 0n olarak verilmiş olsun. Bu durumda W = 0n 0n olacaktır çünkü 0n teriminin tersi yine kendisidir.

Yukarıdaki örnek W için W  ∈ L rahatlıkla denilebilir. Çünkü soruda verilen tanımı sağlamaktadır. Ayrıca dikkat ediniz W terimi X ve X^R terimlerinden mürekkep olduğu için bu terimin boyu her zaman çifttir yani |W| her zaman çift bir sayıdır diyeibliriz. Çünkü X ile X^R kelimelerinin boyu eşittir. Yani bir kelimenin boyu ile tersinin boyu eşit olduğu için bu iki kelimeden oluşan W’nun boyutu tabii olarak çift sayı olacaktır.

Pompalama önsavından öğrendiğimiz üzere, W terimi n adet 0 ile başladığına göre, herhangi bir j sayısı, 0 ≤ j ≤ n şartını sağlamak üzere, pompalama önsavından gelen tanım itibariyle l = oj olur denilebilir. Yani diğer bir deyişle, klm dizilimindeki l teriminin pompalanacağını biliyoruz ve bu terim soruda verilen W ifadesini gösterebilmek için oj olarak seçilmeli ki pompalama neticesinde 0n terimine ulaşılsın (veya 0n terimine ulaşılana kadar pompalansın).

Örneğin klim ifadesi için i = 0 kabul edilirse kl0m = km = 0n-j 0n

olacaktır. Ayrıca bu terimin de bir W terimi olduğunu ve W’nun tanımında bulunan şartları sağladığını söyleyebilmeliyiz.

Ancak, n> n-j (j>0 olduğu için) olduğuna göre ve ayrıca j teriminin tek veya çift olması ile ilgili herhangi bir bilgi olmadığına göre. j teriminin tek olması halinde km teriminin eleman sayısının tek sayı olduğu görülür. Bu durumda tanım itibariyle çift sayıda eleman içermesi gereken W teriminin tek sayıda terim içerebileceği de görülmüş olur. (Bu durumda bir çelişki oluşuyor ve ispatını üzerine kurduğumuz üzere L dilinin RE olması kabulu çökmüş oluyor (contradiction)

Demek ki soruda verilen W terimi pompalama önsavına göre tek sayıda terim içerebilmektedir. Oysaki tanım itibariyle W bu şekilde bir terim asla olamaz.

Soruya bu şekilde cevap verdikten sonra, daha iyi anlaşılması açısından durumu şu şekilde izah edeyim:

Düzenli ifadelerde kullanabileceğimiz ve tekrar ifade eden tek işlemimiz (operator) ne yazık ki [kleene yıldızı (kleene star)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) ismi verilen ve \* sembolü ile gösterilen semboldür. Bu sembolün kaç kere tekrar edeceğini bilemeyiz.

Yukarıdaki soruda X için \* kullanılması halinde X^R için yıldız kullanılamaz. İkisi için de kullanılırsa farklı miktarlarda tekrarlar olabilir. Bunun için iki veya daha fazla sayıdaki tekrar eden terimin birbirine eşit olması istenen durumlarda (ki bu örnek ve daha önce yazıda verdiğim örnekte de aynı durum söz konusu) RE olarak yazılamayacağını söyleyebiliriz çünkü kleen star kontrolsüz bir operatördür.

## **SORU-25: Pompalama Önsavı (Pumping Lemma)**

Bilgisayar bilimlerinde dil tasarımı (language design, compiler design) konusunda önemli araçlardan birisidir. Bu önsava (lemma) göre şayet bir dil, bir herhangi bir gruba ( [içerik bağımsız dil (context free language)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-gramer-context-free-grammer-cfg/) veya [düzenli ifadeler (Regular expression)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) yada farklı bir dil grubu ) dahil olarak kabul ediliyorsa, bu dil ne kadar pompalanırsa pompalansın yine aynı dil grubuna dahil olmalıdır.

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

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

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

Pompalama önsavının farklı dilbilimlerde kulanımına örnekler aşağıda verilmiştir:

* [İçerik Bağımsız Diller için Pompalama Önsavı (Pumping Lemma for Context Free Languages)](http://www.bilgisayarkavramlari.com/2009/03/22/icerik-bagimsiz-gramerler-icin-pompalama-onsavi-pumping-lemma-for-context-free-grammers/)
* [Düzenli ifadeler için Pompalama Önsavı (Pumping Leamma for Regular Expressions)](http://www.bilgisayarkavramlari.com/2009/03/22/duzenli-ifadelerde-pompalama-onsavi-pumping-lemma-for-regular-expressions/)

## **SORU-26: 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 -> 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, -> ok işareti ise, işaretin solundaki devamlının (nonterminal), sağındaki sembolle gösterilebileceğini ifade etmektedir.

Bir [dili içerikten bağımsız (context-free)](http://www.bilgisayarkavramlari.com/2009/03/20/icerikten-bagimsiz-dil-context-free-language/) yapan, o dilin bir [belirsiz aşağı sürüklememli otomat (non deterministic pushdown automata)](http://www.bilgisayarkavramlari.com/2007/05/04/pda-push-down-automata-asagi-suruklemeli-otomat/) 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 grammer).

Bu anlamda, [YACC](http://www.bilgisayarkavramlari.com/2008/12/12/yacc/) gibi programlama ortamlarında, bir dil tasarlamak ve içerikten bağımsız kurallar yazark 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)](http://www.bilgisayarkavramlari.com/2008/03/19/kelime-bilim-lexicology-vocabulary/) bir parçası olarak sıkça kullanılan [uyum (agreement)](http://www.bilgisayarkavramlari.com/2009/03/21/uyum-agreement-kabul-bagit-mutabakat/) 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 sembolu (starting symbol) olarak sıralanabilir. Bir gramerin tanımı sırasında kullanılan bu kümeler aşağıdaki şekilde yazılabilir:

G = ( V , ∑ , R , S)

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

Örneğin

S -> aSb | ab

şeklinde tanımlanan bir dilde:

G= ( {S} , {a,b} , {S -> aSb | ab} , S )

gösterimi kullanılabilir.

## **SORU-27: 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)](http://www.bilgisayarkavramlari.com/2007/05/04/pda-push-down-automata-asagi-suruklemeli-otomat/) tarafından kabul edilen dil çeşididir. Bazı kaynaklarda bağlamdan bağımsız dil olarak da geçmektedir.

Örneğin çok meşhur L= {anbn , 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 çıkaraibiliyoruz:

δ(q0,a,z) = (q0,a)  
δ(q0,a,a) = (q0,a)  
δ(q0,b,a) = (q1,x)  
δ(q1,b,a) = (q1,x)  
δ(q1,b,z) = (qf,z)

δ(state1,read,pop) = (state2,push)

Yukarıdaki [PDA (push down automaton)](http://www.bilgisayarkavramlari.com/2007/05/04/pda-push-down-automata-asagi-suruklemeli-otomat/) tasarıımnda 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= {anbn , n>0} ) CFG (context free grammer, içerik bağımsız gramer)  karşılığını aşağıda yazalım:

S -> aSb | 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 oluşsan dil tasarlanabilir ve üretilen bütün dillerde a’nın sayısı ile b’nin sayısı eşittir.

## **SORU-28: EBNF (Uzatılmış BNF, Extended Backus Normal Form)**

Bilgisayar bilimlerinde dil tasarımı konusunda kullanılan [backus normal şeklinin (backus normal form)](http://www.bilgisayarkavramlari.com/2008/12/09/backus-normal-form-bnf/) 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:

<EGER> ::= if( <KOSUL>) | if( <KOSUL>) else

şeklinde bir satırımız 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:

<EGER> ::= if( <KOSUL>) [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** | **-** |

Yukarıdakitabloda 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)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) 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.

EBNF’in belki de BNF’e göre en büyük eklentisi, hariç (fark) işlemidir. Yani bir bilgi grubundan başka bir bilgi grubunun çıkarılması durumudur.

Mesela bir [dizgi (String)](http://www.bilgisayarkavramlari.com/2008/08/02/dizgi-string/) tanımı sırasında çift tırnaklar arasında herhangi bir yazı yazılabilir. Ancak bu yazının içerisinde çift tırnak bulunamaz çünkü bu durumda dizginin bittiğini belirten çift tırnak ile karışıklık oluşur. Bunu ifade için aşağıdaki EBNF gösterimini inceleyelim:

<DIZGI> ::= ‘ ” ‘ , { ?bütün karakterler? – ‘ ” ‘ } ,  ‘ ” ‘

Yukarıdaki yeni kuralda, bütün karakterlerden, çift tırnak karakteri ayrı tutulmuştur. Yine yukarıdaki gösterimde ?bütün karakterler? tanımı, özel bir dizilim görüntüsüdür.

## **SORU-29: YACC**

YACC, bilgisayara bilimlerinin önemli dallarından birisi olan dil tasarımı ve dil geliştirilmesi sırasında (compiler teory) sıkça kullanılan bir kod üretici programdır. YACC basitçe dildeki sözdizim (syntax) tasarımı için kullanılır ve tasarladığımız dildeki kelimelerin sıralamasının istediğimiz şekilde girilip girilmediğini kontrol eder. Aynı zamanda sıralamadaki her kelimenin anlamını da yacc marifetiyle belirleyebiliriz.

YACC temel olarak [BNF (Backus Normal Form)](http://www.bilgisayarkavramlari.com/2008/12/09/backus-normal-form-bnf/) kullanarak cümle dizimini belirtmektedir.

[LEX](http://www.bilgisayarkavramlari.com/2008/12/12/lex/) ile birlikte kullanıldığıdan bir dil tasarımının neredeyse yarısı olan lexical (kelime) ve syntax (cümle) analizi tamamlanmış olur. Bundan sonra dildeki her kelime ve cümle diziliminin anlamını (semantic) kodlamak kalır.

YACC neredeyse tamamen aynı olan açık kaynak kodlu bison isimli paket ile de kurulabilir.

Aşağıda örnek bir yacc dosyası verilmiştir. Bu dosyayı inceleyerek yacc’ın kullanımını anlamaya çalışalım:

%{ // tanımlamam yapılan ilk bölüm

    #include <stdio.h>

    int yylex(void);

    void yyerror(char \*);

%}

%token INTEGER

%% // BNF yapısındaki ikinci bölüm

program:

        program expr 'n'         { printf("%dn", $2); }

        |

        ;

expr:

        INTEGER

        | expr '+' expr           { $$ = $1 + $3; }

        | expr '-' expr           { $$ = $1 - $3; }

        ;

%%

// fonksiyon içeriklerinin bulunduğu son bölüm

void yyerror(char \*s) {

    fprintf(stderr, "%sn", s);

}

int main(void) {

    yyparse();

    return 0;

}

Dosyamız LEX’e benzer şekilde 3 bölümden oluşmaktadır. İlk bölümde kodumuza dahil edilecek kütüphaneleri ve fonksiyon prototiplerini tanımlıyoruz.

ikinci bölümde BNF yapısı benzeri bir şekilde dilimizdeki sözdizim tanımlanıyor.

üçüncü ve son bölümde fonksiyon tanımları ve main fonksiyonu bulunmaktadır.

Yukarıdaki dosyada yapılan işlemleri aşağıda anlatalım:

YACC dosyasında %token ve %type olarak iki farklı tanım yapılabilmektedir. %token tanımı lex’den gelen bir sabit bilgiyi taşımak için kullanılır. %type ise yacc içerisinde tanımlanan devamlıları (nonterminal) ifade etmek için kullanılır.

Genelde bir kural olmamasına karşılık BNF yapısındaki sonlular (terminal) büyük harfle ve devamlılar (nonterminal) küçük harf ile yazılırlar. (dosyamızın C kodu üreteceğini ve gerek yacc gerek lex gerek se C dilinin büyükküçük harfe duyarlı (case sensitive) olduğunu unutmayınız)

yyerror fonksiyonu yacc tarafından girilen değeri kontrol ederken bir dizilim hatası bulunması (syntax error) çağrılan fonksiyondur.

main fonksiyonu aynen üretilen koda kopyalanmakta ve ilk çalışan C fonksiyonumuzdur.

BNF yapısının yanında bulunan ve küme parantezleri arasında (curved brackets) bulunan kodda ise aşağıdaki terimler kullanılabilir:

$$ -> bu devamlıdan (nonterminal) dönecek olan değerdir.

$1 -> BNF yapımızdaki ilk parametredir. Örneğin aşağıdaki satırı ele alalım:

| expr '+' expr           { $$ = $1 + $3; }

Bu satırda ilk sembol veya anlamında | sembolüdür. Bu sembolün amacı bir expr tanımlanırken ihtimallerden birisi olmasıdır. Ardından expr sembolü ilk sembol ‘+’ ikinci sembol ve ikinci expr sembolü ise 3. semboldür. Dolayısıyla sırasıyla :

$1= expr

$2= ‘+’

$3 = expr

anlamına gelmektedir.

Bu satırın bulunması durumunda yapılacak işlem ilk expr karşılık gelen değer ile ikinci expr karşılık gelen değerin sayısal olarak toplanması ve sonuç olarak $$ marifeti ile döndürülmesidir.

## **SORU-30: LEX**

Bilgisayar bilimlerinde programlama dillerinin tasarımı ve geliştirilmesi sırasında kullanılan ve dildeki kelimelerin analizine (lexical analysis) yarayan kod üretme programıdır. Yani lex için hazırlanmış bir dosyayı lex programından geçirdikten sonra size C dilinde bir kod çıkar. Bu kodu C dilinde [derledikten (compile)](http://www.bilgisayarkavramlari.com/2008/01/03/derleyici-compiler/) sonra çalışan bir programınız olur. Veya tercihen bu çıktıyı [yacc programına](http://www.bilgisayarkavramlari.com/2008/12/12/yacc/) alt yapı oluşturmak için de kullanabilirsiniz.

LEX programının ismi inigilizcedeki lexical analyzer kelimesinden gelir. Kelime bilimi anlamına gelen [Lexical kelimesinin](http://www.bilgisayarkavramlari.com/2009/05/04/kelimebilimsel-bakis-lexical-aspect/) ilk 3 harfinden kısaltılmıştır. LEX programının linux üzerinde çalışan ve yaygın bir sürümü flex ismindedir. flex programı da lex gibi çalışmaktadır ve hemen hemen aynı parametre ve özelliklerle kullanılabilir.

LEX dosyalarının 3 ana bölümü bulunur. İlk bölümde fonksiyon ve değişke tanımlamaları ve projeye dahil edilecek (include) kütüphaneler tanımlanır.

ikinci bölümde LEX dosyamızın omurgasını oluşturan [düzenli deyimler ( regular expression )](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) kısmı yer alır. Burada her ihtimal için ayrı bir regular expression tanımlanarak ilgili regular expression’a girilmesi durumunda ne yapılacağı kodlanır.

Son bölümde ise fonksiyon içerikleri yer alır.

Aşağıda basit bir lex dosyasının içeriği verilmiştir:

%{ // yukarıda anlatılan ilk bölüm tanımlamalar yapılıyor

    #include "y.tab.h"

    #include <stdlib.h>

    void yyerror(char \*);

%}

// ikinci bölüm regular expressionlar

%%

[0-9]+      {

                yylval = atoi(yytext);

                return INTEGER;

            }

[-+n]      { return \*yytext; }

[ t]       ;       /\* skip whitespace \*/

.           yyerror("Unknown character");

%%

// son bölüm fonksiyon içerikleri

int yywrap(void) {

    return 1;

}

Yukarıdaki bu dosya daha önce de anlatıldığı üzere 3 parçadan oluşmaktadır. Koddaki [düzenli deyimler( regular expression)](http://www.bilgisayarkavramlari.com/2007/04/14/regular-expression-regexp-duzenli-deyimler-ifadeler/) kısaca aşağıda açıklanmıştır:

* [0-9]+ 0 ile 9 arasındaki sayılar (yani 0 1 2 3 4 5 6 7 8 9 sayılarından en az bir yada istenildiği kadar çok
* [-+n] sembol olarak “+”, “-” veya “n” sembolleri (yani artı eksi veya satır sonu (enter tuşu))
* [ t] sembol olara ” ” veya “t” yani boşluk karakteri (space) veya tab karakteri
* . geri kalan bütün karakterler. LEX dosyasında üstte yazılı olan düzenli deyimlerin önceliği var ve . sembolü herşeyi kapsar dolayısıyla yukarıdakilerden birisine girmezse buraya girer.

Yukarıdaki düzenli deyimler için yapılan işlemler küme parantezleri arasında (curved brackets) yazılmaktadır. Bu alanlarda yazılanları da tanıyalım:

yylval değişkeni lex tarafından döndürülen değerdir. Bu değer dahil edilen y.tab.h dosyasında tanımlanır ve bu değişkene değer atanarak lex’den örneğin yacc’a veri geçişi sağlanır.

yytext değişkeni ilgili düzenli deyime (Regular expression) giren terimleri alır. Örneğin lex çalışırken 123 gibi bir girdi aldı diyelim. Bu girdi ilk düzenli deyime girecek (çünkü 123 sayısal değerleri [0-9] aralığına uyan 3 sembolden oluşuyor ve bu sembollerin 3 kere tekrarlanması bu düzenli deyim tarafından algılanır). Bu deyim girdikten sonra yapılan iş gelen metin değerini (yytext ki burada “123″ dizgisi (String) oluyor) atoi fonksiyonu ile integer tipine çevirir. Bu çevrilen değer yylval değişkenine atanarak yacc’a geçirilmektedir.

## **SORU-31: Parçalama Ağacı (Parse Tree)**

Parçalam işlemi  (parsing) bilgisayar bilimlerinde çeşitli amaçlar için kullanılmaktadır. Özellikle de dil ile ilgili işlemlerin hemen hepsinde ihtiyaç duyulan bir işlemdir. Örneğin bir programlama dilinde yazılan komutların algılanması için öncelikle kelimeleirn parçalanması (parse) gerekir. Benzer şekilde dopal 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 (grammer) göre verilen cümlenin (veya kelimenin) nasıl parçalandığını şekilsel olarak gösteren ağaçlardır:

Örneğin aşağıda [BNF yapısında](http://www.bilgisayarkavramlari.com/2008/12/09/backus-normal-form-bnf/) verilmiş dili ele alalım:

<dil>::=<işlem>

<işlem> ::= <işlem> + <terim> | <işlem> – <terim> | <terim>

<terim> ::= <terim> \* <unsur> | <terim> / <unsur> | <unsur>

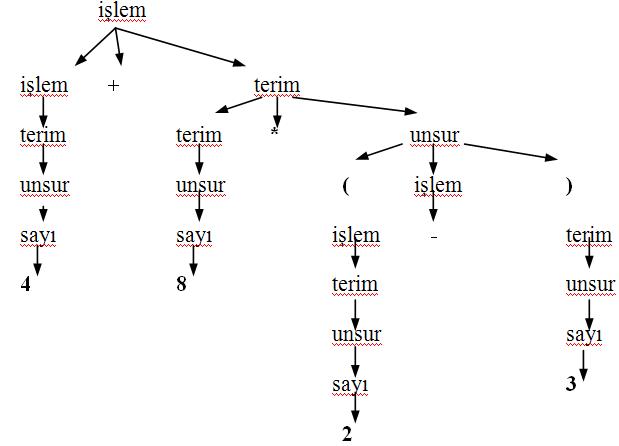
<unsur> ::= **sayı** | **(** <işlem> **)**

<sayı> ::= 1|2|3…|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-32: Backus Normal Form (BNF)**

Bilgisayar bilimlerilnde genellikle bir dil tanımlamada ve bu dilin gramerini (Dil bilgisini) belirlemekte kullanılan gösterim biçimidir.

Basitçe dil bir dil tanımında başlayarak Terminal (sonuncu) ve Non-Terminal (Devamlı) terimler kullanarak tanılmanmaktadır.

Örneğin aşağıda basit bir örneği verilmiştir:

<dil> ::= <harf>|<imla>

<harf> ::= a|b|…|z

<imla> ::= .| |,|?

Yukarıda bir dil tanımı yapılmış ve bu dilde harf veya imlalar bulunduğu söylenmiştir. Buna göre dilimizdeki harfler a’dan z’ye kadar olup imla işaretleri de “. ,?” işaretleridir.  Bir ifadenin (nonterminal veya terminal) alabileceği alternatifler | işareti ile ayrılır. Yani harf ya “a”, ya “b” ya… şeklinde sayılan alternatiflerden birisi olabilir demektir. (aralarında [veya (or)](http://www.bilgisayarkavramlari.com/2007/11/29/veya-kapisi-or-gate/) bağlantısı varmış gibi düşünülebilir)

Yukarıda da görüldüğü üzere BNF gösteriminde nonterminal’ler (devamlılar) <> işaretleri arasında belirtilmektedir.

Yukarıdaki dilimize aşağıdakine benzer bir ek yapılırsa:

<dil> ::= **<harf><dil>** |<harf>|<imla>

<harf> ::= a|b|…|z

<imla> ::= .| |,|?

Yukarıda fark edilsin diye koyu gösterilen ek marifetiyle dilimizde bir dögü elde edilmiş ve bu sayede dilimizdeki harfler istenildiği kadar tekrar edilmiştir. Yani ilk örnekte dilimizde sadece tek bir harf veya tek bir imla işareti bulunabilirken yeni eklentimiz ile dilimizde istediğimiz kadar harf bulunudurabiliriz. Elbette bu durumun bir harf veya imla ile bitmesi gerekmektedir.

BNF ayrıca döngü miktarını belirlemek için de {}ij sembollerini kullanır. Bu gösterim i’den j’ye kadar tekralanacağı anlamındadır.

**SORU-33: NFA'den DFA'e çevirim (Converting NFA to DFA)**

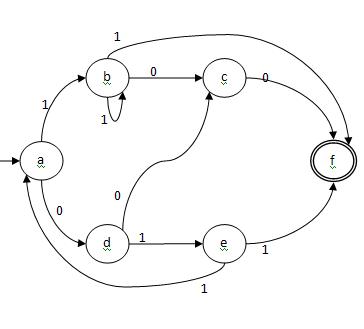
Bu yazıda belirsiz sonlu otomattan(NFA) Belirli sonlu otomata (gerekirci sonlu otomat, nedensel sonlu otomat, deterministic finite automata) dönüştürmenin nasıl yapıldığı anlatılmaktadır.

Basitçe bir iki adımlık işlemler izlenerek bu dönüşüm gerçekleştirilebilir:

1. Öncelikle gerekircilik (determinism) açısından birbiri ile özdeş olan kümler çıkarılmalıdır (subset construction)
2. Bu kümelerin diğer kümler ile olan ilişkisi çıkarılmalıdır.

Sonuçta elde eden kümelerin birer durum göstermesi ve bu durumlar arası bağlantıların yukarıdaki ikinci adımda çıkarılmış ilişkisi belirli bir sonlu otomat (DFA) üretmiş olur.

Bu işlemi aşağıdaki örnekte inceleyelim:



Yukarıda belirsiz bir sonlu otomaton gösterilmiştir. Bu yapıyı belirli sonlu otomata adım adım çevirelim.

Öncelikle lambda ( ε , λ ) ile ulaşılabilen durumları sıralayalım :

{a} : A , tek elemanlı bu küme başlangıç elemanını içerir, bunun dışında yukarıdaki otomatonda herhangi bir elemana lambda ile ulaşılması söz konusu değildir.

Şimdi bir önceki adımda ürettiğimi A kümesinden gidilebilecek kelime ihtimallerini yazalım

A1 (A kümesinden 1 değeri ile ulaşılabilecek olan durumlar) : { b } : B (bu kümeye B ismini verelim)

A0 (A kümesinden 0 değeri ile ulaşılabilecek olan durumlar): { d } : C ( bu kümeye C ismini verelim)

B1 (B kümesinden 1 değeri ile ulaşılabilecek olan durumlar): {b,f} : D (bu kümeye D ismini verelim)

B0 (B kümesinden 0 değeri ile ulaşılabilecek olan durumlar): {c} : E (bu kümeye E ismini verelim)

C1 (C kümesinden 1 değeri ile ulaşılabilecek olan durumlar): {e} : F (bu kümeye F ismini verelim)

C0 (C kümesinden 0 değeri ile ulaşılabilecek olan durumlar): {c} : E (bu kümeye zaten E ismini vermiştik)

D0 (D kümesinden 0 değeri ile ulaşılabilecek olan durumlar): {c} : E (bu kümeye zaten E ismini vermiştik)

D1 (D kümesinden 1 değeri ile ulaşılabilecek olan durumlar): {e} : F (bu kümeye zaten F ismini vermiştik)

E0 (E kümesinden 0 değeri ile ulaşılabilecek olan durumlar): {f} : G (bu kümeye G ismini verelim)

E1 bu ihtimal boş kümedir dolayısıyla işlem yapılmıyor

F0 bu ihtimal boş kümedir dolayısıyla işlem yapılmıyor

F1 (F kümesinden 1 değeri ile ulaşılabilecek olan durumlar): {a,f} : H (bu kümeye H ismini verelim)

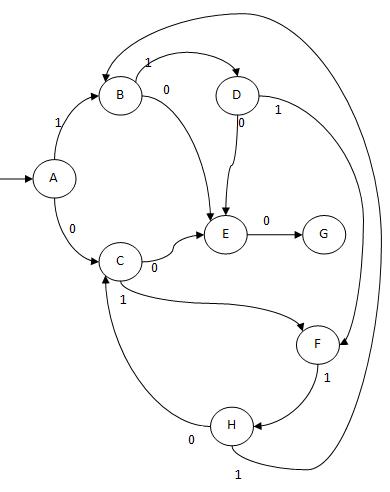
G0 bu ihtimal boş kümedir dolayısıyla işlem yapılmıyor

G1 bu ihtimal boş kümedir dolayısıyla işlem yapılmıyor

H0 (H kümesinden 0 değeri ile ulaşılabilecek olan durumlar): {d} : C (bu kümeye zaten C ismini vermiştik)

H1 (H kümesinden 1 değeri ile ulaşılabilecek olan durumlar): {b} : B (bu kümeye zaten B ismini vermiştik)

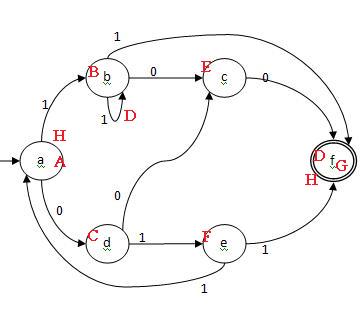
Yukarıda bulunan bütün alt kümelerin inşası tamamlandı ve bu kümelerin diğer kümelere gidiş koşulları belirlendi şimdi bu kümeler arasında yukarıda da listelenmiş olan geçişleri çizebiliriz:



Yukarıdaki yeni otomatımızın belirli (nedensel, gerekirci) olduğunu söyleyebiliriz çünkü her durumda gelen kelimeye göre gidilebilecek tek ihtimal bulunmaktadır.

**Kümelerin NFA üzerinde gösterimi**

Bu kısmı Tarık Bey’in sorusu üzerine ekliyorum. Yukarıdaki kümeleri graf üzerinde kırmızı büyük harf ile gösterdim. Bu sayede kümelerin takibi sanırım daha kolay olacaktır.



Yukarıdaki şekilde şaşılmaması gereken bir husus, aynı düğüme birden fazla küme ismi geliyor olmasıdır. Bu normaldir ve doğrudur ancak kümelerin birden fazla düğüm içerebildiğine dikkat etmek gerekir. Örneğin f düğümü, D,H ve G kümelerinin üyesidir. Ancak tek başına sadece G kümesine üyedir. H kümesine a düğümü ile ve D kümesine b düğümü ile üyedir.

**Örnek  1**

Bir düzenli ifade (regular expression) olarak aşağıdaki örnek verilmiş olsun. Bu örneği sonlu otomat (finite state automata) olarak göstermeye çalışalım:

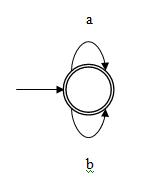
a\*b\*

Yukarıdaki düzenli ifadeyi tanımak için öncelikle bu ifadeden üretilebilecek örnekleri görelim:

Üretilebilecek en kısa dizgi (String): ε (yani boş küme olacaktır, bu bazı kitaplarda λ sembolü ile de gösterilir). Çünkü kleene yıldızının üreteceği sonuçlar arasında boş küme de bulunmaktadır.

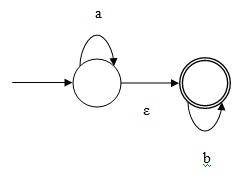
İkinci dizgimiz (String) : ab olacaktır ve bu üretme işlemi aab , abb, aabb, aaab, abbb şeklinde devam edecektir.

Yukarıdaki bu düzenli ifadeyi göstermek için aşağıdaki sonlu otomatı çizebiliriz (genelde sık yapılan bir hata olduğu için aşağıdaki **hatalı örneği** çizmek istiyorum):



Yukarıdaki şekilde anlatılmak istenen tek bir durum (State) oluşudur ve bu durum hem başlangıç (okla belirtilmiştir) hem de bitiş durumu (çift çizgi ile belirtilmiştir) olduğudur. Bu durum üzerinde hem a hem de b harfleri ile istenildiği kadar dönülebilir.

Yukarıdaki hata, yukarıdai FSM’in örneğin ba gibi bir kelimeyi de kabul etmesidir. Oysaki düzenli ifademiz buna izin vermemektedir. **Doğrusu** aşağıdaki şekilde olmalıdır.



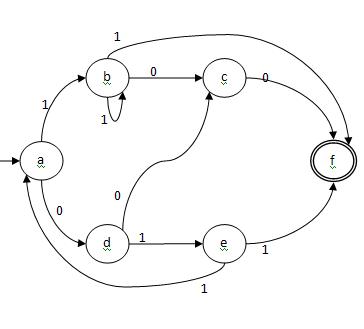
Yukarıdaki çizim doğru çizimdir. Görüldüğü üzere otomatımız hem boş kelimeyi hem de a ve b ile üretilebilecek (Sırası değişmeden) bütün kelimeleri desteklemektedir.

**SORU-34: Belirsiz Sonlu Otomat (Nondeterministic Finite Automat, NFA)**

[DFA (deterministic finite automat)](http://www.bilgisayarkavramlari.com/2008/11/11/belirli-sonlu-otomat-deterministic-finite-automat/) belirli sonlu otomatların (özdevinirlerin) tersine her durumdan gidişin karışık olduğu ve her durum için bir sonraki kelimede nereye gidileceğinin belirli olmadığı otomatlardır.

Basitçe DFA kurallarına uymayan bütün otomatlar NFA olarak adlandırılabilir.

Aşağıda bir örnek üzerinde durumu incleyelim:



yukarıdaki örnekte belirsiz durumlar bulunmaktadır. Örneğin b durumunda iken 1 kelimesi ile hem f durumuna hem de yine b durumuna gitmek mümkündür. Veya e durumundaki ike 1 kelimesi ile f veya a durumlarına geçmek de mümkündür. Bu belirsizlik yüzünden NFA’in bilgisayarlar tarafından algılanması ve kullanılması zor olmaktadır. Ancak insanlar tarafından NFA daha kolay anlaşılır ve kullanılır yapılardır.

**Örnek  1** (Vildan hn.’ın isteği üzerine bu iki örneği ekliyorum umarım yardımcı olur)

Bir düzenli ifade (regular expression) olarak aşağıdaki örnek verilmiş olsun. Bu örneği sonlu otomat (finite state automata, Sonlu Özdevinirler) olarak göstermeye çalışalım:

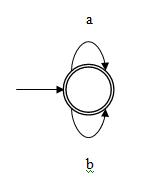
a\*b\*

Yukarıdaki düzenli ifadeyi tanımak için öncelikle bu ifadeden üretilebilecek örnekleri görelim:

Üretilebilecek en kısa dizgi (String): ε (yani boş küme olacaktır, bu bazı kitaplarda λ sembolü ile de gösterilir). Çünkü kleene yıldızının üreteceği sonuçlar arasında boş küme de bulunmaktadır.

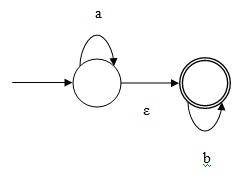
İkinci dizgimiz (String) : ab olacaktır ve bu üretme işlemi aab , abb, aabb, aaab, abbb şeklinde devam edecektir.

Yukarıdaki bu düzenli ifadeyi göstermek için aşağıdaki sonlu otomatı çizebiliriz (genelde sık yapılan bir hata olduğu için aşağıdaki **hatalı örneği** çizmek istiyorum):



Yukarıdaki şekilde anlatılmak istenen tek bir durum (State) oluşudur ve bu durum hem başlangıç (okla belirtilmiştir) hem de bitiş durumu (çift çizgi ile belirtilmiştir) olduğudur. Bu durum üzerinde hem a hem de b harfleri ile istenildiği kadar dönülebilir.

Yukarıdaki hata, yukarıdai FSM’in örneğin ba gibi bir kelimeyi de kabul etmesidir. Oysaki düzenli ifademiz buna izin vermemektedir. **Doğrusu** aşağıdaki şekilde olmalıdır.



Yukarıdaki çizim doğru çizimdir. Görüldüğü üzere otomatımız hem boş kelimeyi hem de a ve b ile üretilebilecek (Sırası değişmeden) bütün kelimeleri desteklemektedir.

**Örnek 2:**

Biraz daha karmaşık kabul edebileceğimiz bir örneği çözmeye çalışalım:

(a+b)\*ab+c

Yukarıdaki ifadeyi analiz edip anlamaya çalışalım. Dilin üreteceği en küçük dizgiden (String) başlayalım ve uzatarak ihtimalleri deneyelim:

c

ab

aab

abab

aaab

bbab

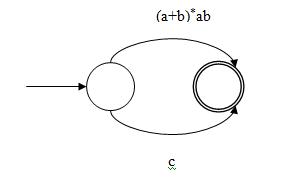
şeklinde giden üretim listemiz bulunuyor. Yukarıdaki düzenli ifadeye bakıldığında görüleceği üzere + (ikinci + sembolü) ifadeyi ikiye bölmüştür. Yani ifademizi:

(a+b)\*ab

veya

c

ifadelerinden birisini üretecek gibi düşünebiliriz.Bu durumu FSM ile gösterecek olursak:



şeklinde düşünülebilir. Elbette yukarıdaki çizim tam bir FSM değildir. Yani kollardan üstteki kolu açarak çizmemiz gerekir ancak fikir vermesi açısından + sembolleri (veya anlamındaki semboller) iki kol olarak düşünülebilir.

Yukarıdaki bu ifadeyi daha açık halde yazacak olursak ve üst kolu analiz edersek

(a+b)\*ab

ifadesini de ikiye bölmek mümkündür:

(a+b)\*

ve

ab

olarak bölünebilir.

Yukarıda bu üç ifade üleştirilmiştir (concatenate) yani

ABC

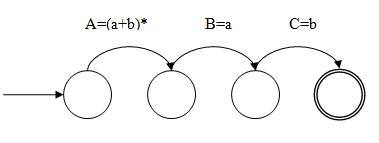
üleştirmesi olarak düşünülürse

A= (a+b)\*

B= a

C= b

olarak düşünülebilir. Bu durumdaki üleştirme işlemleri aşağıdaki şekilde çizilebilir:

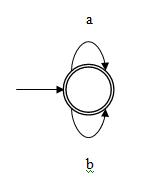


Yukarıdaki bu yeni çizimde üleştirme işlemi görülmüştür.

Yukarıdaki üleştirme işleminde de (a+b)\*ifadesi kapalı olarak bırakılmıştır. Son olarak bu ifadeyi nasıl göstereceğimizi inceleyelim:

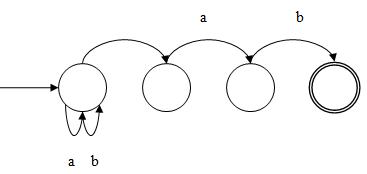
(a+b)\*

ifadesi de görüldüğü üzere aslında a+b ifadesinin kleene yıldızının (kleene star) uygulanmış halidir. Dolayısıyla aşağıdaki şekilde gösterilebilir:

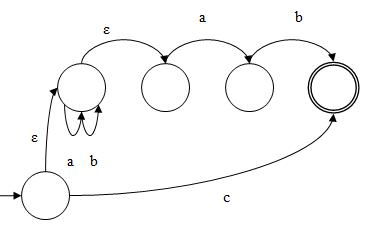


yukarıdaki şekilde görüleceği üzere bütün a ve b ile üretilebilecek (ve boş küme dahil) ihtimaller kapsanmaktadır.

Son aşamada bütün bu alt FSM çizimlerimizi birleştiriyoruz. Önce sondan başalayarak son iki çizimi birleştirelim:



Yukarıdaki ifade (a+b)\*ab düzenli ifadesinin sonlu otomatı olmaktadır. Buna ilk otomatımızı da eklersek sonucu buluruz:



Yukarıda FSM’ini bulmak istediğimiz düzenli ifade olan (a+b)\*ab+c ifadesinin son hali görülmektedir.

Yukarıdaki soru çözümü sırasında izelenen yöntem parçala fethet yöntemidir. Problem cevabını bildiğimiz basit parçalara bölünmüş ve sonra birleştirilmiştir. Problemde özel olarak karşılaşılabilecek en temel 3 durum olan veya (+), üleştirme (concetanation) ve kleene yıldızı durumlarını içeren bir örnek seçtim. Düzenli ifadeleri çevirirken istisnalar hariç bu üç duruma indirgeyerek çizim yapabilirsiniz.

**SORU-35: Belirli Sonlu Otomat (Deterministic Finite Automat, DFA)**

[Sonlu otomatların](http://www.bilgisayarkavramlari.com/2007/12/25/sonlu-durum-makinasi-finite-state-machine-finite-state-automaton/) özel bir halidir. Bu özel hal aşağıdaki 3 durumu içermelidir:

1. Her durumdan (State) gidilecek koşulun tek bir durum göstermesi. Yani bir durumda başka duruma geçerken bir kelime ile sadece bir duruma gidilebilmesi
2. Tek bitiş durumunun bulunması (final state)
3. Lambda (veya epsilon) kelimesinin durumlar arası geçişte yer almaması

Örneğin aşağıdaki sonlu otomatı (aynı zamanda [sonlu durum makinesi (finite state machine](http://www.bilgisayarkavramlari.com/2007/12/25/sonlu-durum-makinasi-finite-state-machine-finite-state-automaton/)) de denilmektedir) inceleyelim:

Yukardaki otomatta, başlangıç soldan gelen ok ile gösterilmiş q1 ve bitiş çift halka içerisine alınmış durum (düğüm, node, state) ile gösterilmiş olan q2′dir.

Yukarıdaki otomatın özellikleri kontrol edildiğinde belirli sonlu otomat (deterministic finite automaton) olmasını gerektiren koşullar şöyledir:

1. adımdaki şartı sağlar çünkü herhangi bir durumda diğerine giderken belirsizlik söz konusu değildir. Örneğin q1 durumundayken 0 gelince tek bir yere (yine q1) ve 1 gelince tek bir yere (q2) gidilmektedir. Şayet herhangi birisi için birden fazla alternatif bulunsaydı bu durumda [belirsiz sonlu otomat (nondeterministic finite automat, nfa)](http://www.bilgisayarkavramlari.com/2008/11/11/belirsiz-sonlu-otomat-nondeterministic-finite-automat-nfa/) denilebilirdi.

2. adımdaki şartı da sağlamaktadır çünkü otomatta hiç lambda geçişi (boş geçiş) yoktur. bütün geçişler bir değerle (kelime) yapılmaktadır.

3. adımı da sağlamaktadır çünkü tek bitiş durumu söz konusudur (bu bitiş de q2 durumudur)

dolayısıyla yukarıdaki otomatın belirli (Deterministic) olduğu söylenebilir

**Sonlu Durum Makinelerinin Gösterimi**

Bazı kaynaklarda, sonlu durum makinelerinin ifadesi için, yukarıdaki yazıda yer alan görsel gösterimin yerine küme gösterimi veya tablo gösterimi de kullanılmaktadır.

Buna göre durum makinesinin geçişlerini ve durumlarını gösteren bir şekil tablo hazırlanabilir. Yukarıdaki makine için aşağıdaki tablo aynı makineyi tutmaktadır:

|  |  |  |  |
| --- | --- | --- | --- |
|  | q1 | q2 | q3 |
| q1 | 0 | 1 |  |
| q2 |  | 1 | 0 |
| q3 |  | 1,0 |  |

Yukarıdaki gösterime göre geçişlerin tutulduğu bir matris oluşmuştur. Bu matris, yönlü bir şekil (graph) için kullanılmasından mütevellit asimetrik yapıdadır.

Aynı sonlu durum makinesi aşağıdaki tablo ile de tutulabilir:

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| q1 | q1 | q2 |
| q2 | q3 | q2 |
| q3 | q2 | q2 |

Yukarıdaki yeni tabloda, kolon başlıkları gelen parametreyi (geçiş değerini) ve tablo içeriği de geçilecek olan durumu(state) tutmaktadır.

Bu tabloda da bir hücrede birden fazla durum(state) bulunması halinde NFA, her hücrede tek durum (state) bulunması halinde DFA yorumu yapılabilir.

Yukarıdaki tablo gösterimlerine ilave olarak küme halinde tutulabilir. En çok kullanılan yazım aşağıdaki şekildedir:

(Σ, S, S0 , δ , F )

Buna göre Σ sembolü bir [alfabeyi (alphabet)](http://www.bilgisayarkavramlari.com/2008/08/02/alfabe-abece-alphabet/) ifade etmektedir. Yani dilimizde kabul edilen girdilerin her birisini gösteren küme. Örneğin yazı boyunca kullandığımız DFA için {0,1 }kümesidir denilebilir.

S kümesi, otomattaki durumların kümesidir. Yani örnek otomatımızda {q1,q2,q3} durumları bulunmaktadır

S0 ise başlangıç durumudur (state). Örnek otomatımızda q1 başlangıç durumudur.

δ kümesi ise geçişlerin tutulduğu kümedir. Örnek otomatımızda geçişler için şöyle bir küme yazabiliriz δ = { q1 -> q1, 0 ; q1 -> q2 ,1 ; q2 -> q3 , 0 ;q2 -> q2 ,1 ;q3->q2, 0; q3 -> q2, 1   } olarak yazılabilir.

F ise bitiş durumunu gösterir. Yani bir gidi (input) bu durumda bitiriyorsa kabul edilir (accept). Diğer durumlarda bitmesi veya gidilemeyen bir devam olması durumunda ise red edilir (reject). Örnek otomatımızda q2 bitiş durumudur.

Kısaca yukarıdaki özellikler tek bir kümeler kümesinde toplanırsa, aşağıdaki gibi bir gösterim ortaya çıkar:

{ {0,1 } ,  {q1,q2,q3} , q1 , { q1 -> q1, 0 ; q1 -> q2 ,1 ; q2 -> q3 , 0 ;q2 -> q2 ,1 ;q3->q2, 0; q3 -> q2, 1   } , q2 }

Yukarıdaki bu küme en başta çizilen durum makinesini göstermek için yeterlidir.

**SORU-36: Kuantum İşleme (Quantum Computing)**

Bu yazının amacı kuantum bilgisayarları ve kuantum işleme (Quantum Computing) konusunda fikir vermek ve yapılan çalışmaların arkasındaki felsefeyi aktarmaktır.

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](http://www.bilgisayarkavramlari.com/2010/10/05/von-neumann-makinesi/) 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](http://www.bilgisayarkavramlari.com/kubit-qubit/) daha detaylı okuyabilirsiniz.

Kuantum hesaplamalarının en büyük farklılığı Kuantum paralelliğidir. Kuantum paralelliği (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 sağalandığı 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 paralelliği).

**SORU-37: Nöbetçi (Sentinel)**

Veri işlenmesi sırasında çeşitli durumlarda işlemin istenmeyen yerlere gitmesini engelleyen veri tipidir. Örneğin boyutu bilinmeyen bir dizi işlenirken dizinin sonuna gelindiğini anlamak için dizinin sonunda programın algılamasını sağlayan ve dizideki sayılardan ayırt edilen bir değer girmek gibi. Mesela sadece pozitif tam sayılardan oluşan bir dizide bitişi belirtmek için -1 yazılması gibi.

Veya klavyeden sayılar okunurken -1 okununca işlemin sona ereceğini algılamak gibi.

**SORU-38: Sonlu Ototmatlar (Finite Automaton)**

[Bir sonlu durum makinesinin](http://bilgisayarkavramlari.com/2007/12/25/sonlu-durum-makinasi-finite-state-machine-finite-state-automaton/) formal şekilde gösterilmiş halidir. Buna göre bir otomat’ı oluşturan 5 farklı unsur bulunur. Bunlar o otomatta  bulunan durumlar (states) o otomatın durumları arası geçişlerin alabileceği [semboller](http://bilgisayarkavramlari.com/2008/08/02/sembol-harf-isaret-symbol/) kümesi olan [alfabe](http://bilgisayarkavramlari.com/2008/08/02/alfabe-abece-alphabet/), o otomattaki durumlar ve [alfabeler](http://bilgisayarkavramlari.com/2008/08/02/alfabe-abece-alphabet/) arasındaki geçişi gösteren fonksiyon (transition function) ve son olarak otomattaki bitiş durumlarını gösteren küme (set of accept sets) olarak sıralanabilir.

Bir sonlu otomat aşağdaki şekilde gösterilebilir:

(Q,Σ,δ,q0,F)

Q: durumlar kümesidir (states)

Σ: dilin tanımlı olduğu semboller kümesi (alfabe)

δ: geçiş fonksiyonu Q x Σ → Q

q0 ∈ Q , Q kümesinin bir elemanıdır ve başlangış durumunu (state) gösterir

F ⊆ Q , Q kümesinin bir alt kümesi olan F kümesi, otomatın kabul ettiği bitiş durumlarını gösterir.

Örneğin aşağıda verilmiş olan sonlu durum makinesini sonlu otomat şeklinde yazalım:

Q: { q1,q2,q3 }

Σ: {0,1}

δ:

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |

q1 başlangıç durumudur.

F = { q2 }

olarak atanır.

**SORU-39: Bağımsız düğümler (Anti Clique, Independent Set)**

[Klik yapısının](http://bilgisayarkavramlari.com/2008/08/02/klik-clique/) tersi olarak düşünülebilir. Basitçe bir [grafta](http://bilgisayarkavramlari.com/2008/08/01/graf-sekil-graph/) birbiri ile doğrudan bağlantısı olmayan [düğümlerin](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) oluşturduğu [alt graftır](http://bilgisayarkavramlari.com/2008/08/01/altgraf-subgraph/).

Yukarıdaki tasvirde iki adet graf verilmiştir. Üstte bütün graf görülmekte altta ise bu grafın bir [alt grafı](http://bilgisayarkavramlari.com/2008/08/01/altgraf-subgraph/) görülmketedir. Dikkat edilirse sadece altta bulunan {A,E,F} [düğümleri](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) alındığında aşağıdaki graf elde edilir ve bu grafta bulunan düğümleri birbirlerine bağlayan [kenar](http://bilgisayarkavramlari.com/2008/08/01/kenar-edge/) bulunmamaktadır.

**SORU-40: Klik (clique)**

[Graf](http://bilgisayarkavramlari.com/2008/08/01/graf-sekil-graph/) teorisinde her iki düğümü birbirine bir kenar ile bağlanmış alt graflara verilen isimdir. Örneğin aşağıdaki grafikte bir klik kırmızı çizgiler ile işaretlenmiştir. Buna göre {A,B,C,D} alt grafı bir kliktir.

Sosyal bilimlerde de aynı kelime(klik) bir toplumun en alt birimine verilen isimdir. Bunun sebebi doğrudan bağlantısı olan ve komşuluğu bulunan bireylerden oluşması olarak yorumlanabilir.

Klik yapısının tersi için [bağımsız düğümler veya anti clique veya independent set](http://bilgisayarkavramlari.com/2008/08/02/bagimsiz-dugumler-anti-clique-independent-set/) terimleri kullanılır.

**SORU-41: İstikra ile ispat (Tüme varım, Proof by Induction)**

Bir [kaziyeyi (önerme)](http://bilgisayarkavramlari.com/2007/11/29/onermeler-mantigi-propositional-logic/) ispat ederek [nazariye (teorem)](http://bilgisayarkavramlari.com/2008/08/02/nazariye-teori-kuram-theorem/) elde etme yöntemidir. [İstikra](http://bilgisayarkavramlari.com/2008/08/02/istikratume-varim-induction/) cüz’îler (tikeller) den küllî  
(tümel) ye gitme yöntemidir dolayısıyla örneklerden yola çıkarak her zaman için geçerli bir sonuç elde ederek ispat yapılır.

Her istikra için bir esas(basis) bir de istikra(induction) safhası(step) bulunur.  Bu iki safhanın ispatı bütün durumların ispatı demektir. Çünkü isikra edilen herşeyin dayanacağı bir esas bulunmalıdır.

Bu durum şöyle bir örnek ile de ifade edilebilir örneğin iki ayaklı bir sandalyenin ayakta durabilmesi için başka bir iki ayaklı sandalyeye dayanması gerekir, onunda ayakta durması için bir diğerine ve böylece bütün sandalyelerin bir diğerine dayanaması gerekir. En başta ise bir adet 4 ayaklı sandalyeye ihtiyaç vardır ki bütün birbirine dayanan sandalyeler gelip bu sandalyeye dayansın. İşte şayet en başta bulunan bu 4 ayaklı sandalye yanı esas safhası (basis step) ispat edilirse ve her sandalyeninde birbirine dayanarak ayakta duracağı yani istikra safhası (induction step) ispat edilirse istenilen sayıdaki sandalyenin ayakta durabileceği ispatlanmış olur.

Daha sayısal bir örnek için 0′dan n’e kadar olan sayıların toplamının (n \* (n + 1) ) / 2 olduğunu ıspatlayalım.

bu ispattaki esas safhamız (basis step)  n = 1 durumu olsun ve bu durumda (1 \* ( 1 + 1) ) / 2 = 2/ 2 = 1 olarak denklemimizin doğru çalıştığını gösterebiliriz.

istikra safhamız (induction step) ise n + 1 için olan durum olmalıdır. Bu durum da  ( ( n + 1) \* ( (n  + 1) + 1)) / 2 olur. daha sade bir şekilde ( ( n + 1 ) \* ( n + 2) )  / 2 yazılabilir.

şayet istikra safhamızı ispatlayabilirsek esas safhamızdaki ispata dayandırarak istikra edebiliriz ( bütün durumlar için genelleme yapabiliriz).

şayet n sayının toplamı (n \* (n + 1) ) / 2 ise n+1 sayının toplamı da  (n \* (n + 1) ) / 2 + ( n + 1) olmalıdır.

isitkra safhasında elde ettiğimiz sonuç da n+1 sayının toplamını vermektedir o halde :

( ( n + 1 ) \* ( n + 2) )  / 2 =  (n \* (n + 1) ) / 2 + ( n + 1)

eşitliği doğru olmalıdır denilebilir. Bu eşitlik çözülecek olursa sonucun 1= 1 doğruluğu ile ispat edildiği görülür.

Dolayısıyla bir esas alınmış ve ispat edilmiştir. Ayrıca bu esasa bina edilen istikra adımları da ispat edilmiştir. Dolayısıyla bütün pozitif sayılar için doğru sonuç veren bir istikra yapılmıştır.  Bu duruma [tam istikra](http://bilgisayarkavramlari.com/2008/08/02/istikratume-varim-induction/) da denilebilir.

**SORU-42: Binaen Burhan (İnşâa ile İspat , Proof by Construction, Binaenaleyh)**

Bilgisayar bilimlerinde kullanılan ispat yöntemlerinden birisidir. Bu yöntemde bir varlığın oluşmasının gösterilmesi hedeflenir.  Örneğin aşağıdaki teoriyi inşaa yöntemi ile ispat edelim:

“2′den büyük her çift n sayısı için n düğüm içeren 3-düzenli graf bulunur”

Öncelikle k-düzenli graf tanımını hatırlayalım:

Bir graf üzerindeki her [düğümün](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) “k” kadar komşusu bulunması durumuna k-düzenli graf denilir. Örneğin aşağıdaki graf 2-düzenli bir graftır çünkü her [düğümün](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) derecesi 2′dir.

Dolayısıyla ispatlanmak istenen [nazariyede](http://bilgisayarkavramlari.com/2008/08/02/nazariye-teori-kuram-theorem/) bize düğüm derecelerinin en az 3 ve daha fazla olabildiği bildirilmiş bir k-düzenli graf isteniyor.

Bir graftaki kenarları aşağıdaki küme ile ifade etmek mümkündür:

K =

{ {i,i+1} | 0 ≤ i  ≤ n-2}  U { n – 1 , 0 } }

U { {i, i+ n/2 } | 0 ≤ i ≤ n/2 – 1 }

Yukarıda iki satır olarak tanımlanan kenarlar kümesindeki ilk satırı komşu düğümleri gösteren satır ve ikinci satırı karşı düğümleri gösteren satır olarak düşünebiliriz. Buna göre n adet düğüm içeren bir grafta her düğüm içi kendisinden önce ve kendisinden sonraki düğümlere birer kenar olduğu kabul edilirse (ilk satır) ayrıca karşısındaki düğüme de bir kenar olduğu kabul edilirse (alttaki satır) bu durumda her düğümün 3. derece olması mümkündür denilebilir.

Yukarıdaki grafta bu durum tasvir edilmiştir. Grafın bir çember olmasını sonsuz sayıda düğüm içermesi olara düğünebiliriz. Bu çeber üzerindeki bir noktayı komşusu olan 2 noktaya bağlayan birer kenar (mavi ile gösterilmiştir) ve tam karşısında buluna noktaya bağlayan(kırmızı ile gösterilmiştir) birer kenar çizilmesi durumunda n > 2 düğümü bulunan herhangi bir graf için 3-düzenli graf bulunabileceğini tasvir etmiş oluruz.

Yukarıdaki örnekte kullanılan ispat yöntemine bakıldığında bir varlığı oluşturan diğer varlıkların ispatta kullanıldığını görürüz. Örneğin 3-düzenli graf kavramını oluşturan bir düğüm ve 3 kenar varlıklarının ayrı ayrı gösterilmesi ve bu varlıkların üzerine ispatın inşası mümkün olmuştur.

**SORU-43: k-düzenli graf ( k-regular graph)**

Bir graf üzerindeki her [düğümün](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) “k” kadar komşusu bulunması durumuna k-düzenli graf denilir. Örneğin aşağıdaki graf 2-düzenli bir graftır çünkü her [düğümün](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) derecesi 2′dir.

**SORU-44: Nazariye (Teori, Kuram, Theorem)**

Bilgisayar bilimleri açısından matematiksel olarak ispat edilmişi kaziyeler (önermeler, statements) birer nazariyedir. Bazı kaziyelerin(önermelerin) doğruluğu ise sırf  farklı teorilerin ıspatına yardımcı oluyor diye ıspatlanır. Bu tip kaziyelere (önermelere) ise önkuram (önsav, lemma) adı verilmektedir. Bazı durumlarda ise bir nazariyenin ispatı bize bazı başka neticelerin doğru olduğunu gösterir. Bu tarz kendiliğinden doğruluğu anlaşılan kaziyelere(önermelere) ise tabiî sonuç anlamına gelen  sonurgu (corollary) denilmektedir.

**SORU-45: Alt Dizgi (Substring)**

Bir dilde tanımlı olan ve o dildeki [alfabenin](http://bilgisayarkavramlari.com/2008/08/02/alfabe-abece-alphabet/) üyesi olan [semboller](http://bilgisayarkavramlari.com/2008/08/02/sembol-harf-isaret-symbol/) ile üretilmiş her [dizginin](http://bilgisayarkavramlari.com/2008/08/02/dizgi-string/) alt dizgisi olabilir. Alt dizgi o [dizginin](http://bilgisayarkavramlari.com/2008/08/02/dizgi-string/) belirli bir kısmına verilen isimdir. Buna göre örneğin boş dizgi her dizginin alt dizgisidir.

Örneğin bir dildeki alfabe aşağıdaki şekilde tanımlı olsun:

∑1 = {0,1}

Buna göre dilimizde sadece “0″ ve “1″ [sembolleri](http://bilgisayarkavramlari.com/2008/08/02/sembol-harf-isaret-symbol/) tanımlı demektir. Bu dilde örneğin w1=0 veya w2=10101011010 gibi bir dizgi elde etmek mümkündür.

Bir dizginin belirli bir kımını içeren dizgiye ise alt dizgi adı verilir. Örneğin w3=1011 dizgisi w2 dizgisinin bir altdizgisidir.

**SORU-46: 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:

∑1 = {0,1}

Buna göre dilimizde sadece “0″ ve “1″ sembolleri tanımlı demektir. Bu dilde örneğin w1=0 veya w2=10101011010 gibi bir dizgi elde etmek mümkündür.

Bir dizginin belirli bir kımını içeren dizgiye ise alt dizgi adı verilir. Örneğin w3=1011 dizgisi w2 dizgisinin bir altdizgisidir.

Ayrıca iki dizginin arka arkaya eklenmesine de üleştirme(concatenation) denilir.Örneğin w1 ile w3 dizgilerinin üleştirilmiş hali w4=01011 olur.

Dizgiler ile ilgili diğer yazılar:

* [Dizgi Parçalayıcısı (String Tokenizer)](http://www.bilgisayarkavramlari.com/2008/12/20/string-tokenizer-dizgi-parcalayici/)
* [Dizgi Karşılaştırma (String Comparison)](http://www.bilgisayarkavramlari.com/2008/12/27/dizgi-karsilastirma-string-comparison/)
* [Veri Tabanı Dizgi işlemleri (Database String manipulations)](http://www.bilgisayarkavramlari.com/2008/12/30/veritabani-dizgi-islemleri-string-manipulations-on-database/)
* [En uzun ortak küme (longest common subsequence) problemi](http://www.bilgisayarkavramlari.com/2007/12/04/en-uzun-ortak-kume-longest-common-subsequence-lcs/)
* [Dizgi Eş şekilliliği (String Isomorphism)](http://www.bilgisayarkavramlari.com/2009/06/25/dizgi-es-sekilliligi-string-homomorphism/)
* [C dilinde dizgi kopyalama (strcpy)](http://www.bilgisayarkavramlari.com/2008/12/01/strcpy-string-copy-dizgi-kopyalama/)
* [Alt Dizgi (Substring)](http://www.bilgisayarkavramlari.com/2008/08/02/alt-dizgi-substring/)
* [Dizgi Hizalama (String Alignment) Problemi](http://www.bilgisayarkavramlari.com/2012/06/10/dizgi-hizalama-string-alignment/)

Dizgi Karşılaştırma/Arama/Mesafe algoritmaları:

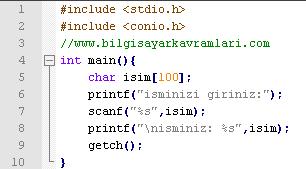
* [Boyer Moore Dizgi Karşılaştırması (Boyer Moore String Comparison Algorithm)](http://www.bilgisayarkavramlari.com/2009/05/19/boyer-moore-dizgi-arama-algoritmasi-boyer-moore-string-search/)
* [Knuth Moris Prat Dizgi Arama Algoritması (Knuth Morris Prat Algorithm)](http://www.bilgisayarkavramlari.com/2009/04/11/knuth-morris-prat-algoritmasi-kmp-algorithm/)
* [Dizgi Eş şekilliliği (String Isomorphism)](http://www.bilgisayarkavramlari.com/2009/06/25/dizgi-es-sekilliligi-string-homomorphism/)
* [Needleman Wunsch Yaslama Algoritması (String Alignment)](http://www.bilgisayarkavramlari.com/2012/06/05/needleman-wunsch-algoritmasi/)
* [Smith Waterman Yaslama Algoritması (String Alignment)](http://www.bilgisayarkavramlari.com/2012/06/06/smith-waterman-dizgi-yaslama-string-alignment-algoritmasi/)
* [Hunt Macllor Yaslama Algoritması (String Alignment)](http://www.bilgisayarkavramlari.com/2012/06/10/hunt-macllory-algoritmasi/)
* [Horspool Algoritması (Dizgi arama algoritması)](http://www.bilgisayarkavramlari.com/2012/01/16/horspool-algoritmasi/)
* [Levenstein Mesafesi (Levenshtein Distance)](http://www.bilgisayarkavramlari.com/2010/12/30/levenshtein-mesafe-algoritmasi-levenshtein-distance/)
* [Hamming Mesafesi (Hamming Distance)](http://www.bilgisayarkavramlari.com/2008/08/05/hamming-mesafesi-hamming-distance/)
* [Diff komutu](http://www.bilgisayarkavramlari.com/2012/06/10/diff-komutu/)
* [Jaccard Indeksi, mesafesi ve katsayısı (Jaccard Index)](http://www.bilgisayarkavramlari.com/2013/07/01/jaccard-indeksi-jaccard-index/)
* [Dice Sorensen Benzerliği (Dice Sorensen Similarity)](http://www.bilgisayarkavramlari.com/2013/07/01/sorenson-dice-katsayisi-dice-sorensen-coefficient/)

Dizgi Parçalama (Parsing) algoritmaları:

* [LL(1) parçalama algoritması](http://www.bilgisayarkavramlari.com/2012/04/08/ll1-parcalama-algoritmasi/)
* [SLR(1) parçalama algoritması](http://www.bilgisayarkavramlari.com/2012/05/31/slr1/)
* [Earley Parçalama Algoritması](http://www.bilgisayarkavramlari.com/2012/05/31/earley-parcalama-algoritmasi-earley-parsing-algorithm/)

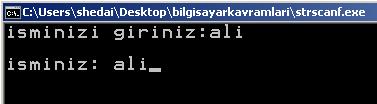
**C ile dizgi okuma**

C dilinde klavyeden dizgi (String) okumak için kullanılan en basit fonksiyon scanf fonksiyonudur. Bu fonksiyonu basit bir uygulamada aşağıdaki şekilde kullanabiliriz:

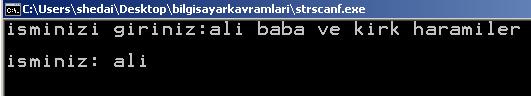


Yukarıdaki kodda, 5. satırda tanımlalan karakter dizisi (string) içerisine 7. satırda %s parametresi ile scanf fonksiyonu kullanılarak bir dizgi okunmuştur. Bu dizginin içeriği kodun 8. satırında ekrana basılmıştır.

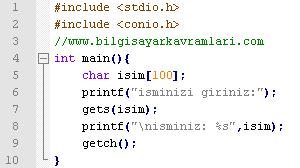
Örneğin yukarıdaki kod aşağıdaki şekilde çalıştırılabilir:



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:



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 eşitlerse 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-47: Alfabe (Abece, Alphabet)**

Bilgisayar bilimlerinde kullanılan ve yazıları ifade etmeye yarayan [sembollerden](http://bilgisayarkavramlari.com/2008/08/02/sembol-harf-isaret-symbol/) oluşmuş kümelere verilen isimdir. Buna göre bir dildeki olası bütün [semboller](http://bilgisayarkavramlari.com/2008/08/02/sembol-harf-isaret-symbol/) kullanılarak oluşturulan alfabeler kullanılarak metinlerin elde edilmesi mümkündür.

Bilgisayar bilimlerindeki alfabelerde bulunan [semboller](http://bilgisayarkavramlari.com/2008/08/02/sembol-harf-isaret-symbol/) sınırlı sayıda kabul edilmiştir. Örneğin aşağıda çeşitli semboller içeren alfabe örnekleri verilmiştir:

∑1 = {0,1}

∑2 = {a,b,c,d,e,f}

Örneğin yukarıdaki ∑1 alfabesi “0″ ve “1″ sayılarını birer [sembol](http://bilgisayarkavramlari.com/2008/08/02/sembol-harf-isaret-symbol/) 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=”debdebe”dizgisi ∑2 üzerinde tanımlı bir [dizgidir](http://bilgisayarkavramlari.com/2008/08/02/dizgi-string/).

Bir dizginin boyutu |w| işareti ile gösterilir ve o dizgideki harf sayısına eşittir. Örneğin yukarıda verilen w=”debdebe” [dizgisi](http://bilgisayarkavramlari.com/2008/08/02/dizgi-string/) için |w| = 7 olarak kabul edilir. Bir [dizginin](http://bilgisayarkavramlari.com/2008/08/02/dizgi-string/) içeriği boş olması durumunda ise |w|=0 olarak kabul edilir.

**SORU-48: 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:

∑1 = {0,1}

∑2 = {a,b,c,d,e,f}

Örneğin yukarıdaki ∑1 alfabesi “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=”debdebe”dizgisi ∑2 üzerinde tanımlı bir dizgidir.

Bir dizginin boyutu |w| işareti ile gösterilir ve o dizgideki harf sayısına eşittir. Örneğin yukarıda verilen w=”debdebe” dizgisi için |w| = 7 olarak kabul edilir. Bir dizginin içeriği boş olması durumunda ise |w|=0 olarak kabul edilir.

**SORU-49: Güçlü Bağlı Graf (Strongly Connected Graph)**

Bir grafta bulunan bütün düğümleri diğer bütün düğümlere bağlayan birer kenar bulunuyorsa bu grafa güçlü bağlı graf adı verilir.

**SORU-50: Basit Döngü (Simple Cycle)**

Bir [graftaki](http://bilgisayarkavramlari.com/2008/08/01/graf-sekil-graph/) bir [döngünün](http://bilgisayarkavramlari.com/2008/08/01/dongu-cycle/) başlangıç ve bitiş [düğümleri](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) olan düğümü dışındaki bütün düğümlerin, bu döngü içerisinde sadece bir kere geçmesi durumunda bu döngüye basit döngü adı verilir.

**SORU-51: Bağlı graf (conected graph)**

Bir [graftaki](http://bilgisayarkavramlari.com/2008/08/01/graf-sekil-graph/) bütün [düğümleri](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) diğer bütün düğümlere bağlayan bir [yol](http://bilgisayarkavramlari.com/2008/08/01/yol-path/) bulunuyorsa bu graflara bağlı graf denilir.

**SORU-52: Döngü (Cycle)**

Graf teorisinde bir [düğümden](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) başlayıp aynı [düğümde](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) biten [yola](http://bilgisayarkavramlari.com/2008/08/01/yol-path/) döngü adı verilir

Örneğin yukarıdaki [grafta](http://bilgisayarkavramlari.com/2008/08/01/graf-sekil-graph/) A düğümünden başlayarak gene bu [düğümde](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) biten {A,C,D}  döngüsü tasvir edilmiştir.

**SORU-53: Altgraf (Subgraph)**

Bir [grafikte](http://bilgisayarkavramlari.com/2008/08/01/graf-sekil-graph/) bulunan [düğüm](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) ve [kenarlardan](http://bilgisayarkavramlari.com/2008/08/01/kenar-edge/) sadece bir kısmını içeren grafa verilen isimdir. Her altgraf da bir graftır. Ayrıca grafın kendisi de altgraflarından bir tanesidir.

Örneğin yukarıdaki şekilde bir graf ve bir alt grafı yanyana gösterilmiştir.

**SORU-54: Yol (Path)**

Bir [graf](http://bilgisayarkavramlari.com/2008/08/01/graf-sekil-graph/) üzerinde bir veya daha fazla [düğümden](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) ve [kenardan](http://bilgisayarkavramlari.com/2008/08/01/kenar-edge/) geçen rotaya verilen isimdir. Örneğin aşağıdaki graf üzerinde bir yol gösterilmiştir.

Yolların yazılışı ise geçtikleri düğümlerin sırasıyla yazılması ile elde edilir. Örneğin yukarıdaki yolu {A,C,D} olarak göstermek mümkündür.

**SORU-55: Yönlü Graflar (Directed Graphs)**

Bir [grafın](http://bilgisayarkavramlari.com/2008/08/01/graf-sekil-graph/) [kenarlarının](http://bilgisayarkavramlari.com/2008/08/01/kenar-edge/) yön belirtmesi durumunda bu [grafa](http://bilgisayarkavramlari.com/2008/08/01/graf-sekil-graph/) yönlü graf adı verilir.

Bir [kenar](http://bilgisayarkavramlari.com/2008/08/01/kenar-edge/) iki [düğümü](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) birleştirmektedir. Yönlü bir [kenar](http://bilgisayarkavramlari.com/2008/08/01/kenar-edge/) ise bir düğümden diğer [düğüme](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) gidilebilen yönü göstermektedir. Bu [kenarın](http://bilgisayarkavramlari.com/2008/08/01/kenar-edge/) gösterdiği yönün tersine doğru da hareket edilebilmesi durumunda bu ikinci bir [kenar](http://bilgisayarkavramlari.com/2008/08/01/kenar-edge/) ile ifade edilir.

Yukarıda A ile B [düğümleri](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) arasında her iki yönde de hareket edilebildiğini gösteren iki adet [kenar](http://bilgisayarkavramlari.com/2008/08/01/kenar-edge/) bulunmaktadır. Bu kenarlardan birsinin bulunmaması durumunda;

tek yönlü hareket etmek mümkün olurken tersi yönde hareket mümkün değildir. Örneğin yukarıdaki şekilde A düğümünden B düğümüne geçiş mümkün iken tersi olan B düğümünden A düğümüne hareket edilememektedir.

**SORU-56: Yönsüz graflar (undirected graphs)**

Bir grafta bulunan kenarların yön bildirmemesi durumunda bu grafa yönsüz graf denilir. Bu durumda iki düğüm arasında bulunan kenar, her iki yönlü de hareket edilebileceğini ifade eder.

graf11.jpg

Örneğin yukarıdaki graf yönsüzdür. Bu grafta A ile B düğümleri arasında bir a kenarı bulunmaktadır. Bu kenar yönsüz olduğu için hem A’dan B düğümüne hem de B’den A düğümüne hareket etmenin mümkün olduğunu gösterir.

**SORU-57: Graf (Şekil, Graph)**

Bilgisayar dünyasında bulunan ve gerçek hayatta çeşitli sebeplerle karşılaşılan yapıları temsil amacıyla kullanılan şekillerdir.

Örneğin bir bilgisayar ağını, karakenarları haritasını veya bir karar ağacını graflar kullanarak temsil etmek mümkündür.

Bilgisayar bilimleri çeşitli uygulamalarda karşılaşılan bu yapıları ifade etmek için çeşitli matematiksel ve görsel yöntemlerden faydalanır.

Buna göre bir grafta bulunan varlıklar [düğümler](http://bilgisayarkavramlari.com/2008/08/01/dugum-node/) ile ifade edilmekte, bu varlıklar arasındaki ilişkiler ise graftaki [kenarlar](http://bilgisayarkavramlari.com/2008/08/01/kenar-edge/) ile ifade edilmektedir.

Grafları [kenarların](http://bilgisayarkavramlari.com/2008/08/01/kenar-edge/) yönlü olup olmamasına göre, yönlü graflar ve yönsüz graflar olarak ikiye ayırmak mümkündür. Ayrıca kenarların değer almasına göre değerli graflar veya değersiz graflar isimleri verilebilir.

 Örneğin yukarıdaki grafta 4 düğümden ve 4 kenardan oluşan bir graf gösterilmektedir. Bu grafı G= ({A,B,C,D} , {(A,B),(A,C),(C,D),(A,D)}) şeklinde ifade etmek mümkündür. Dolayısıyla graflar G= (V,E) şeklinde yanı düğümler ve kenarlar şeklinde yazılmaktadır.

**SORU-58: Kenar (Edge)**

Bir graf üzerindeki her çizgiye kenar adı verilir. kenarlar düğümleri birleştirdikleri için bu ismi almışlardır. Graf teorisinde bir kenaru ifade etmek için birleştirdiği düğümlerin isimleri yazılır. Örneğin aşağıdaki şekildeki “a” kenarunu ifade etmek için (A,B) gösterimi kullanılır.

**SORU-59: Düğüm (Node)**

Bir graf üzerindeki her noktaya düğüm adı verilir. Düğümler, kenarlar kendi üzerlerinde birleştiği için bu ismi almışlardır.

Graf teorisine göre bir düğümün derecesi o düğümde bulunan kenar sayısıdır. Örnepin aşağıdaki grafta A düğümünün derecesi 3′tür.

**SORU-60: DTD (Document Type Definition, Döküman Tip Tanımı)**

[XML](http://bilgisayarkavramlari.com/2008/01/07/xml-extensible-markup-language-genisletilebilir-isaretleme-dili/) veya [HTML dosyalarındaki](http://www.bilgisayarkavramlari.com/2008/12/10/html-hyper-text-markup-language/) kayıtların uyması gereken koşulları belirlemeye yarar. Buna göre bir [XML dökümanı](http://bilgisayarkavramlari.com/2008/01/07/xml-extensible-markup-language-genisletilebilir-isaretleme-dili/) istenilen bir DTD kuralına tabi tutularak bu kurala uyup uymadığı denetlenebilir. DTD’nin kendisi de bir dil olarak tanımlanabilir ve bu dilin de kendisine özgü kuralları bulunmaktadır.

Temel olarak bir dökümanda kullanılacak olan DTD’nin belirlenmesi iki türlü mümkündür. Birinci yöntemde dökümanda kullanılacak olan DTD tanımı harici olarak yapılmıştır ve dökümana bu harici tanım belirtilir. İkinci yöntemde ise DTD tanımı doğrudan dökümanın içinde yapılır.

İlk gruba örnek olarak aşağıdaki yapıyı alabiliriz:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

Yukarıdaki örnek kodda bir HTML tamını içerisinde kullanılacak olan DTD dosyasının yolu verilmiş ve HTML kodlarımızın bu DTD dosyasındaki kurallara uygun olacağı tanımlanmıştır.

İkinci gruba yani döküman içerisinde DTD tanımının yapıldığı duruma örnek olarak aşağıdaki kodu inceleyelim:

<?xml version=”1.0”?>

<!DOCTYPE SIMPLE

[

<!ELEMENT SIMPLE ANY>

]

>

<SIMPLE> Bu basit bir dtd'li xml ornegidir. </SIMPLE>

Yukarıdaki örnekte basit bir [XML dosyas](http://bilgisayarkavramlari.com/2008/01/07/xml-extensible-markup-language-genisletilebilir-isaretleme-dili/)ı ve dosyanın içeriğinde bir DTD tanımı yapılmıştır. Bu DTD tanımı <!DOCTYPE ile başlayan ve > işareti ile biten aralıktır. DTD alanında bir SIMPLE etiketi(tag) tanımlanmış ve bu etiketin içerisine ANY yapısına uygun yani XML kurallarına uygun herhangi bir bilgi girilebileceği belirtilmiş. Dolayısıyla [XML dökümanının](http://bilgisayarkavramlari.com/2008/01/07/xml-extensible-markup-language-genisletilebilir-isaretleme-dili/) içerik kısmında da benzer şekilde SIMPLE etiketi kullanılmış ve içerisine serbest olarak bir metin girilmiştir.

Biraz daha detaylı tanımların yapıldığı aşağıdaki DTD örneğini inceleyelim:

<!DOCTYPE INSAN

[

<!ELEMENT INSAN(ISIM,SOYISIM,MEMLEKET)>

<!ELEMENT ISIM(#PCDATA)>

<!ELEMENT SOYISIM(#PCDATA)>

<!ELEMENT MEMLEKET(#PCDATA)>

]

>

Yukarıdaki örnekte INSAN dökümanı içerisinde ISIM, SOYISIM ve MEMLEKET etiketleri bulunacağı belirtilmiş ve bu etiketlerin tipi olarak PCDATA tanımlanmış. PCDATA alanlar yanlızca karakter içerikli veriler bulundurabilir. Örneğin aşağıdaki XML kodu yukarıdaki DTD tanımına uygundur:

<INSAN>

<ISIM>ŞADİ</ISIM>

<SOYISIM>ŞEKER</SOYISIM>

<MEMLEKET>İSTANBUL</MEMLEKET>

</INSAN>

Ancak aşağıdaki kod yukarıdaki DTD tanımı dahilinde çalışmaz:

~~<INSAN>~~

~~<SOYISIM>VELI</SOYISIM>~~

~~<ISIM>ALI</ISIM>~~

~~<MEMLEKET>GIRESUN</SOYISIM>~~

~~</INSAN>~~

DTD tanımlamaları içinde ayrıca mantıksal operatörler kullanılması da mümkündür. Örneğin yukarıdaki örnekte bir insan etiketi için mutlaka isim, soyisim ve memleket tanımı yapılması beklenmektedir ancak meslek için farklı bir durum söz konusudur bir insan postac, mühendis yada hemşire olabilir ama hepsini birden olması beklenemez. İşte böyle bir DTD tanımı aşağıda verilmiştir:

<!DOCTYPE INSAN

[

<!ELEMENT INSAN(POSTACI | MUHENDIS | HEMSIRE)>

<!ELEMENT POSTACI(#PCDATA)>

<!ELEMENT MUHENDIS(#PCDATA)>

<!ELEMENT HEMSIRE(#PCDATA)>

]

>

Yukarıdaki DTD örneğinde dikkat edilecek nokta etiketler arasına | (pipe) işareti konmuş olmasıdır. Bu işaretin anlamı [mantıksal olarak veya’dır.](http://bilgisayarkavramlari.com/2007/11/29/veya-kapisi-or-gate/)

Yukarıdaki DTD tanımına uygun bir XML dökümanı aşağıdaki şekilde olabilir:

<INSAN>

<POSTACI>ALI</POSTACI>

</INSAN>

veya

<INSAN>

<MUHENDIS>ALI</HEMSIRE>

</INSAN>

ama asagidaki dokuman birden fazla komutu ayni anda icerdigi icin gecersiz kabul edilir.

~~<INSAN>~~

~~<MUHENDIS>ALI</HEMSIRE>~~

~~<POSTACI>ALI</POSTACI>~~

~~</INSAN>~~

Ayrıca DTD tanımı içerisinde bir etiketin ne kadar olması gerektiği de tanımlanabilir bunu için  ?, +, \* sembolleri kullanılır

automata theorydeki regular expressionlardan esinlenen bu semboller kisaca:

* ? : sifir yada bir
* +: bi yada daha cok
* \*: sifir yada daha cok

demektir.

**SORU-61: 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 yansımasıdır.

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

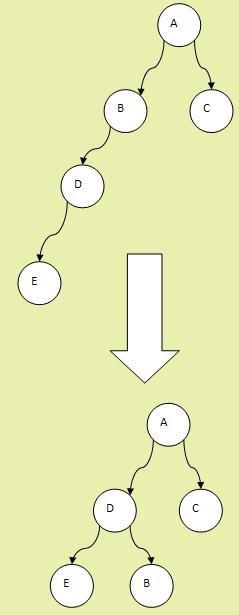
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Demet (Buket, Bucket) | Sayılar | | | | | |
| 0 |  |  |  |  |  |  |
| 1 | 81 |  |  |  |  |  |
| 2 | 12 | 432 | 62 |  |  |  |
| 3 | 3 |  |  |  |  |  |
| 4 | 4 | 34 | 344 | 54 | 94 | 44 |
| 5 | 95 |  |  |  |  |  |
| 6 |  |  |  |  |  |  |
| 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.

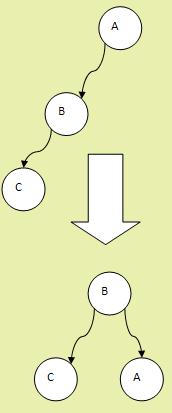
**SORU-62: Ağaçlarda Dengeleme (Rotation, Balancing)**

En çok karşılaşılan durum, ikili arama ağaçlarında bir düğüm için çocuklarının derinliklerinin 2 olması durumudu. Bu durum aşağıdaki örnekte gösterilmiştir:

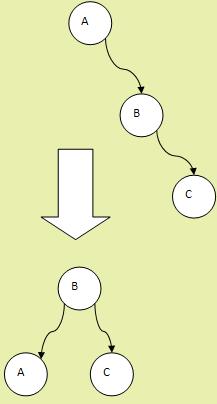
[](http://bilgisayarkavramlari.com/wp-content/uploads/2008/05/rotate.jpg)

Yukarıdaki tasvirde ayrıca bu ağacın dengelenmiş hale nasıl dönüştürüldüğü de gösterilmiştir. Buna göre ağaç sağa dengelenmiş ve ikili arama ağacı özelliği bozulmamıştır. Yani dengelendikten sonra da ağacın sağ kolundaki değerler, sol kolundaki değerlerden büyüktür.

Yukarıdaki şekilde de gösterildiği üzere ağacın kolları arasındaki derinlik farkı 2 olduğunda dengeleme yapılabilir. Bu durum iki türlü olabilir ya sağ kolu daha uzun ya da sol kolu daha uzun olacaktır. Her iki durum içinde yapılan işlemler sola dengele veya sağa dengele olarak sınıflandırılabilir.

[](http://bilgisayarkavramlari.com/wp-content/uploads/2008/05/soladengele.jpg)

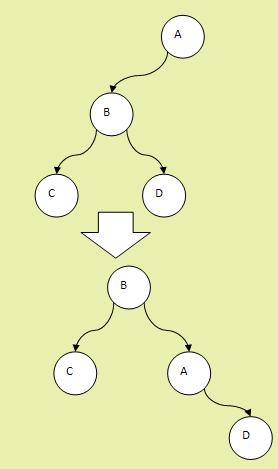
Örneğin yukarıda, bir ağacın kolları arasındaki derinlik farkı 2′yi geçmiş ve bu dengesizlikde derin olan taraf sol kolu olmuştur. Buna göre ağacın dengelenmiş halinde, ilk halinde kökte bulunan düğüm sağ kola yerleştirilir.

[](http://bilgisayarkavramlari.com/wp-content/uploads/2008/05/sagadengele.jpg)

Benzer bir durumda ağacın sağ kolunun daha derin olamsı halidir. Bu durum da da ağacın kökünde bulunan düğüm sol kola yerleştirilir.

Ağaçlarda dengesiz olan bütün durumlar yukarıda anlatılan örnekte olduğu gibi dengelenebilir. Basitçe şayet ağacımız dengeli yapılmak isteniyorsa ve her ekleme veya silme işleminden sonra bu dengeleme kontrolü çalışıyorsa ağaç dengeli bir halde kalacaktır. Ayrıca ikili ağaçların iki kolu bulunduğu için yukarıdaki durumlar dışında bir durum ile karşılaşılamaz.

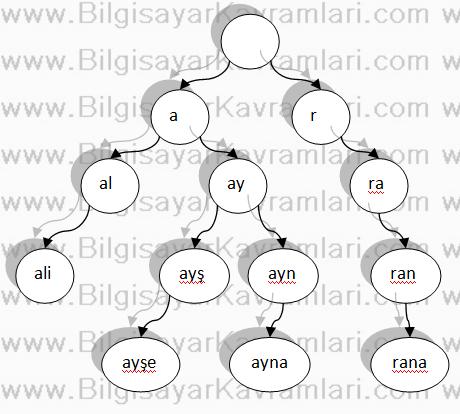
Aşağıda ise sadece silme işleminden sonra karşılaşılabilecek bir durum verilmiştir:

[](http://bilgisayarkavramlari.com/wp-content/uploads/2008/05/soladengele2.jpg)

Yukarıdaki bu durum ekleme işlemleri sırasında karşılaşılamaz çünkü eklenen her adımdan sonra ağaç dengeli hale getirileceği için C eklenmesi durumunda D, D eklenmesi durumunda ise C dengeli bir ağaca eklenecektir. Ancak silme durumlarında yukarıdaki şekilde dengeleme işlemi yapılabilir. Ağacın sola dengelenmesi gerektiği durum ise yukarıda tasvir edilen halin simetriğidir.

**SORU-63: Trie (Metin Ağacı)**

Metin ağaçları, her düğümün kendisinden sonra gelen harfi işaret ettiği [ağaçlardır](http://bilgisayarkavramlari.com/2008/05/07/agaclar-tree/). 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 kollardaki harflerin birleştirilmiş halidir. Bir düğümden bir harf taşıyan sadece bir kol çıkabilir.

Trie ağacının ismi re**trie**val kelimesininin ortasındaki 4 harften gelmektedir.

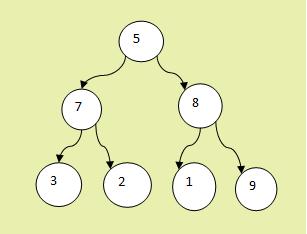
Metin ağaçlarının (trie), [ikili arama ağaçlarına](http://bilgisayarkavramlari.com/2008/05/07/ikili-arama-agaci-binary-search-tree/) göre en önemli avantajları bir metni aramanın, metin boyutu kadar işlem gerektirmesidir. [İkili arama ağaçlarında](http://bilgisayarkavramlari.com/2008/05/07/ikili-arama-agaci-binary-search-tree/) ise bu süre log n kadar varkit almaktadır. Buradaki n, ağaçtaki düğüm sayısıdır dolayısıyla [ikili arama ağaçları](http://bilgisayarkavramlari.com/2008/05/07/ikili-arama-agaci-binary-search-tree/), ağaçtaki bilgiye göre hızlı veya yavaş çalışırken, metin ağaçları, ağaçta ne kadar bilgi bulunduğundan bağımsız olarak çalışırlar.

Metin ağaçları [hafızayı](http://www.bilgisayarkavramlari.com/2008/11/07/rastgele-erisilebilir-bellek-random-access-memory-ram/) da verimli kullanırlar çünkü bir metin ağacının en derin noktası, ağaç üzerindeki en uzun metin kadardır. [İkili ağaçlar](http://bilgisayarkavramlari.com/2008/05/07/ikili-arama-agaci-binary-search-tree/)da ise bu derinlik eklenen düğüm sayısına göre en kötü ihtimalle düğüm sayısı kadar olabilmektedir.

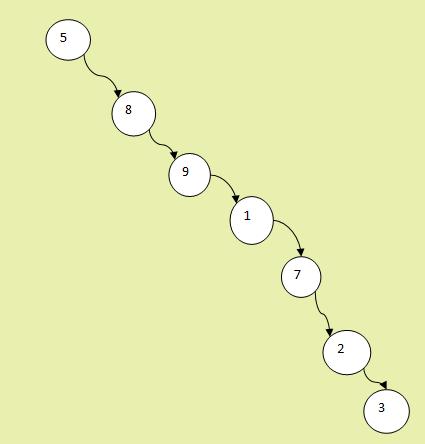
Ayrıca metin ağaçları [en uzun önek eşlemesi (longest prefix matching)](http://bilgisayarkavramlari.com/2007/12/22/en-uzun-onek-eslesmesi-longest-prefix-matching/) gibi problemlerin çözümünde de avantaj sağlar.

**SORU-64: İkili Ağaç (Binary Tree)**

[Ağaçların](http://bilgisayarkavramlari.com/2008/05/07/agaclar-tree/) özel bir hali olan ikili ağaçlarda her düğümün çocuklarının sayısı azami 2 olabilir. Bir düğümün daha az çocuğu bulunması durumunda ( 0 veya 1) ağacın yapısı bozulmaz. Yapraklar hariç bütün düğümlerin ikişer çocuğu bulunması ve yaprakların aynı derinlikte bulunması durumunda bu ağaca dengeli ağaç (balanced tree) denilir. Aşağıda bir dengeli ikili ağaç örneği tasvir edilmiştir:

[](http://bilgisayarkavramlari.com/wp-content/uploads/2008/05/ikiliagac.jpg)

Bu ağacı değişik sıralarda yeniden oluşturabiliriz. Örneğin aşağıdaki ağaç da yukarıdaki verilerin aynılarını taşıyan bir ikili ağaç örneğidir.

[](http://bilgisayarkavramlari.com/wp-content/uploads/2008/05/ikilizincir.jpg)

Yukarıdaki bu ağacın ilk örnekten farkı dengesiz olması ve özel olarak her düğümün çocuk sayısının 1 olmasıdır. Tanım hatırlanacak olursa yukarıdaki bu ağaç da bir ikili ağaç olarak kabul edilebilir.

C dilinde bir ikili ağacı ifade edecek [struct](http://bilgisayarkavramlari.com/2007/11/08/olusum-composition-ve-struct-yapilar/) aşağıdaki şekilde yazılabilir:

typedef struct dugum{

int veri;

dugum \*sol;

dugum \*sag;

};

Yukarıdaki kodda bir düğümün taşıması gereken bilgiler tanımlanmıştır. Buna göre düğümün sağındaki ve solundaki çocukları gösteren birer [gösterici (pointer)](http://bilgisayarkavramlari.com/2007/10/16/pointer-gosterici-2/) ve düğümün içindeki veriyi tutan bir veri değişkeni bulunmaktadır.

Benzer durum java dilinde aşağıdaki şekilde ifade edilebilir:

class dugum{

int veri;

dugum sol;

dugum sag;

};

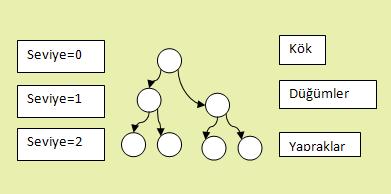
Yukarıdaki kodda ise nesne göstericisi (object referrer) kullanılarak bir nevi [gösterici (pointer)](http://bilgisayarkavramlari.com/2007/10/16/pointer-gosterici-2/) yapısı kullanılmıştır. Buna göre her düğümün sol ve sağında gene düğüm cinsinden birer nesne bulunabilecektir.

Daha fazla bilgi için ikili ağaçların özel bir hali olan [ikili arama ağaçlarına (binary search tree)](http://bilgisayarkavramlari.com/2008/05/07/ikili-arama-agaci-binary-search-tree/) bakabilirsiniz.

**SORU-65: Ağaçlar (tree)**

Bir [graf](http://bilgisayarkavramlari.com/2008/08/01/graf-sekil-graph/) şayet [bağlı grafsa](http://bilgisayarkavramlari.com/2008/08/01/bagli-graf-conected-graph/) ve hiç [döngü](http://bilgisayarkavramlari.com/2008/08/01/dongu-cycle/) 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.

[](http://bilgisayarkavramlari.com/wp-content/uploads/2008/05/agac.jpg)

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.

**SORU-66: OWL Time (OWL Zaman, Web Varlıkbilim Dili Zaman)**

Gelişen zamanlama ihtiyaçları ile birlikte zamanın gösterimi ve formüllenmesi de bir ihtiyaç haline gelmiştir. Örneğin yapılan her siparişte, siparişin zamanının tutulması, basir bir kiralama işleminde veya bilet satış işleminde yapılan işlemin hangi tarih ve saatler için yapıldığının tutulması artık sıradan birer gereksinim haline gelmiştir. Bu amaçla doğmuş olan OWL Time , Web Ontology Language (Baş harflerinin sırası okunmasını kolaylaştırmak için değiştirilmiş ve OWL olarak kısaltılmıştır) altında zaman göstermek amacıyla XML üzerine kurulu bir dil olarak tasarlanmıştır. Projenin daha önceki ismi DAML-Time olarak geçmekteydi.

Dilin yapısında varlıklar TemporalEntity (zaman varlığı veya fânî varlık olarak isimlendirilmektedir) ve her TemporalEntity altında Instant (kesin, ânî ) veya Interval (aralık, zaman aralığı) olarak iki farklı varlık bulunur.

:Instant

a owl:Class ;

rdfs:subClassOf :TemporalEntity .

:Interval

a owl:Class ;

rdfs:subClassOf :TemporalEntity .

:TemporalEntity

a owl:Class ;

rdfs:subClassOf :TemporalThing ;

owl:equivalentClass

[ a owl:Class ;

owl:unionOf (:Instant :Interval)

] .

Bu durum yukarıdaki RDF/XML gösterimi ile temsil edilmiştir. Dikkat edilirse Instant ve Interval sınıfları (Class) , TemporalEntity sınıfının (Class) birer alt sınıfıdır (subclass)

**Sürelerin İfade edilmesi**

OWL-Time kullanılarak sürelerin ifade edilmesi de ayrı bir problemdir. Örneğin bir eylemin süresi 1 gün ve 2 saat veya 26 saat veya 1560 dakika olabilir ve bütün bu zamanlar aynı süreye işaret etmektedir. Dolayısıyla sağlıklı bir zaman gösteriminde Yıl, Ay, Hafta, Gün, Saat, Dakika, Saniye olmak üzere 7 farklı zaman gösterimi gerekmektedir. Ayrıca bu zaman gösterimini belirli eden ilave bir gösterici ile toplam 8lik gösterici kullanılmaktadır. Bu durum aşağıdaki sınıf yapısında gösterilmiştir:

:DurationDescription

a owl:Class ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :seconds

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :minutes

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :hours

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :days

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :weeks

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :months

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :years

] .

Yukarıda gösterilen sınıf DurationDescription yani Süre Tanımlayıcı sınıfının yapısıdır. Bu sınıftan üretilen bir örnek aşağıda verilmiştir:

duration

a :DurationDescription ;

:seconds 20 ;

:hours 5 ;

:days 15 .

Yukarıdaki örnekte 15 gün 5 saat ve 20 saniyelik bir zaman dilimi temsil edilmiştir.

Yukarıda Süre Tanılmayıcısı (DurationDescription) sınıfının tanımına dikkat edilirse her zaman birimi için ayrıca bir de maxCardinality (azami sayısallık) tanımı yapılmış ve bütün zaman birimleri için bu değer 1 atanmıştır. Bunun OWL dili açısından anlamı bu değerlerin sayısal olarak istedikleri gibi atanabilecekleridir. Ancak OWL dili bilindiği üzere varlıkbilimsel bir dildir ve her zamanın bütün bu birimleri ifade etmesi beklenemez. Örneğin üniversite eğitiminin saniyeler dakikalar cinsinden ifadesi beklenmedik bir durumdur, genelde üniversite eğitimi 2,3,4,5,6 gibi yıllar ile ifade edilir. Veya bir yemek süresi dakika ve saatler mertebesindedir bu sürenin yıllar ile ifade edilmesi beklenmedik bir durumdur. İşte OWL-Time bu gibi durumlarda diğer zaman birimlerinin kapatılmasına imkan verir ve yapılması gereken basitçe bu zaman birimlerinin maxCardinality bilgisini 0 yapmaktır:

:Year

a owl:Class ;

rdfs:subClassOf :DurationDescription ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:cardinality 1 ;

owl:onProperty :years

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:cardinality 0 ;

owl:onProperty :months

] ;

...

rdfs:subClassOf

[ a owl:Restriction ;

owl:cardinality 0 ;

owl:onProperty :seconds

] .

Örneğin yukarıda bir yıl sınıfı tanımı yapılmıştır ve yıl dışındaki bütün zaman birimlerinin cardinality (sayısallık) bilgisi 0 yapılmıştır.

**Tarih ve Saat kavramı**  
OWL-Time kullanılarak belirli bir tarihin veya saatin ifade edilmesi de gerekmektedir. Örneğin 10:30 bir saat ifade etmektedir buna karşılık 26 mart 2008 ise bir tarih ifade etmektedir. Dolayısıyla herhangi bir zamanı ifade etmek için öncelikle hangi birimden ifadenin yapılacağına karar verilmelidir. Yani ifade edilecek olan zaman bir dakikayı mı bir günü mü yada bir saati mi ifade etmek amacındadır belirlenmelidir. ( “saat 10″ önermesi bir saat ifade ederken “26 mayıs” bir günü ifade etmektedir). Bu belirlenen birime unitType (birim tipi) denilirse geriye zaman diliminin ifade edilmesi sorunu kalmaktadır. Yani istenilen saat acaba hangi zaman dilimine göre (örneğin GMT, EST, PST ) belirtilmektedir.  
Son olarak tarih ve saati ifade eden zaman birimleri aşağıdaki şekilde tanımlanabilir:

:DateTimeDescription

a owl:Class ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:cardinality 1 ;

owl:onProperty :unitType

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :second

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :minute

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :hour

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :day

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :dayOfWeek

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :dayOfYear

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :week

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :month

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :year

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:maxCardinality 1 ;

owl:onProperty :timeZone

] .

:hasDateTimeDescription

a owl:ObjectProperty ;

rdfs:domain :DateTimeInterval ;

rdfs:range :DateTimeDescription .

Görüldüğü üzere yukarıdaki gösterimde haftanın günleri veya yılın günü gibi kavramlar da ilave edilerek bir tarih/saat ifadesine yer verilmiştir. İlk eklenen alt sınıf (subclass) bir zaman birmini gösterirken son alt sınıf ise zaman dilimini (timeZone) göstermektedir.

Bu bilgiler paralelinde aşağıdaki toplantı varlığını inceleyelim:

:meetingStart

a :Instant ;

:inDateTime

:meetingStartDescription ;

:inXSDDateTime

2006-01-01T10:30:00-5:00 .

:meetingStartDescription

a :DateTimeDescription ;

:unitType :unitMinute ;

:minute 30 ;

:hour 10 ;

:day 1 ;

:dayOfWeek :Sunday ;

:dayOfYear 1 ;

:week 1 ;

:month 1 ;

:timeZone tz-us:EST ;

:year 2006 .

Yukarıda aynı toplantı varlığı için iki farklı gösterim bulunmaktadır. İlk gösterimde XSD zamanı denilen XML Scheme Definition kelimelerinin kısaltılmışı olan ve XML üzerinde tanımlama yapılan dilin biçemine uygun olarak yazılmış gösterime yer verilmiştir. Aynı tarih değeri altta bizim tanımladığımız tarih saat yapısına uygun olarak tekrar gösterilmiştri. Buna göre toplantı dakika cinsinden bir zamana işaret etmekte olup zaman birimi EST’dir ve toplantının zamanı: 1/1/2006 saat 10:30′dur.

**OWL Time kullanarak zaman aralığı gösterimi**  
Bilindiği üzere zaman gösterimlerinde her zaman kesin ve belirli zamanları ifade etmek mümkün değildir. Örneğin bir kargo firmasının teslim süresi olarak 2-3 gün arasında demesi, teslimin en az 2 en çok 3 günde gerçekleşeceğini ifade etmektedir. Bu durum OWL Time kullanılarak aşağıdaki şekilde ifade edilebilir:

:DeliveryDuration

a owl:Class ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:cardinality 1 ;

owl:onProperty :maxDeliveryDuration

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:cardinality 1 ;

owl:onProperty :minDeliveryDuration

] .

:maxDeliveryDuration

a rdf:Property ;

rdfs:domain :DeliveryDuration ;

rdfs:range time:Interval .

:minDeliveryDuration

a rdf:Property ;

rdfs:domain :DeliveryDuration ;

rdfs:range time:Interval .

Yukarıdaki gösterimde bir teslim tarihi için iki alt bilgi girilmiş bunlardan birisi minDelivery (asgari teslim) diğeri ise maxDelivery (azami teslim) süresi olarak belirlenmiştir.  
Bu belirlemenin yanında 2 günlük ve 3 günlük sürelerde aşağıdaki şekilde tanımlanabilir:

:Interval2Days

a owl:Class ;

rdfs:subClassOf time:Interval ;

owl:subClassOf

[ a owl:Restriction ;

owl:hasValue P2D ;

owl:onProperty time:durationDescriptionDataType

] .

:Interval3Days

a owl:Class ;

rdfs:subClassOf time:Interval ;

owl:subClassOf

[ a owl:Restriction ;

owl:hasValue P3D ;

owl:onProperty time:durationDescriptionDataType

] .

Artık bu tanımlamalardan sonra kargo firmasının teslim süresi tanımlanabilir:

:Cargo2-3dayDuration

a owl:Class ;

rdfs:subClassOf :DeliveryDuration ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:allValuesFrom :Interval3Days ;

owl:onProperty :maxDeliveryDuration

] ;

rdfs:subClassOf

[ a owl:Restriction ;

owl:allValuesFrom :Interval2Days ;

owl:onProperty :minDeliveryDuration

] .

yukarıda görüldüğü üzere kargo firmasının teslim süresi 2 günlük zaman diliminde asgari ve 3günlük zaman diliminde azami süreler olarak tanımlanmıştır.

OWL-Time kullanarak ne yazık ki belirsiz iki eylemin göreceli zamanlarını tanımlamak mümkün değildir. Örneğin A olayı B olayından sonra olmuştur önermesi OWL-Time kullanımıyla gösterilemez.

**SORU-67: TTML (Time Tabling Markup Language, Zaman Çizelgeleme İşaretleme Dili)**

Zaman çizelge işareteleme dili (Timetabling Markup Language (TTML)), XML üzerine kurulmuştur. MathML üzerine kurulu zaman çizelgeleme problemlerinin çözümünde kullanılmaktadır.

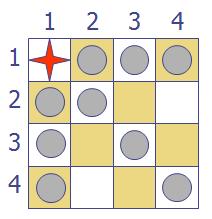
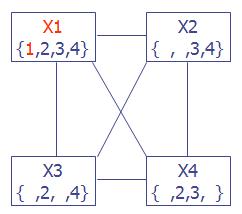
TTML üzerinde kullanılan formülleme dili küme teorisine dayandırılabilir. Örneğin MathML üzerinde kullanıcılara tamamen yeni semboller ile bu sembollerin ifade ettiği yeni fonksiyon ve formülleri tanımlama imkanı sağlanır. Bu durum MathML kullanan TTML dilinin de küme teorisini desteklemesini sağlamaktadır. Bu sayede kullanıcı esnek bir şekilde kendi kıstaslarını belirleyebilmekte ve zaman çizelgelemesini özelleştirebilmektedir.

Bilindiği üzere zaman çizelgelemesi [NP Complete (NP Tam)](http://bilgisayarkavramlari.com/2008/03/24/belirsiz-cokterimli-tam-np-complete-nondeterministic-polynomial-complete/) problem ailesindendir ve zaman çizelgelemesindeki amaç uygun zaman aralıklarına uygun olaylar kümesini yerleştirmektir. Gelişmekte olan şartlara göre zaman çizelgelemede aranan özellikler ve kurallar kümesi de gelişmekte dolayısıyla problemin karmaşıklığı artmaktadır. Bu konuda bir standart bulunmaması ayrıca araştırmacıların karşılaştırma imkanını kaldırmakta ve mukayeseli sonuçlar çıkarılmasını zorlaştırmaktadır. Bu konuda ilk çalışma Andrw Cumming tarafından ICPTAT’95 ‘de yapılmış ve çalışmanın sonucu olan dile SSTL ismi verilmiştir. Beklenildiği üzere SSTL dili de bir standartlaşmayı sağlayamamıştır çünkü elde bulunan problemlerin bu dile çevrilmesi tam anlamıyla mümkün olmamıştır. Günümüzde neredeyse her araştırmacı kendi gösterimini kullanmakta ve bu konuda bir standartlaşma henüz sağlanamamaktadır.

**SORU-68: Arc Constraint (Kiriş Şartı)**

AC3 algoritması olarak bilinen ve kiriş tutarlılıklarını (arc consistency) işlemek için kullanılan algoritma anlatılacaktır. Şart işleme (Constraint Processing) konusunda kullanılan önemli yöntemlerden birisidir. Bu algoritmanın AC1 ve AC2 diye adlandırılan daha eski ancak performans olarak daha kötü çözümleri bulunduğu gibi AC4, AC5 gibi daha başarılı ancak uygulaması daha zor ve algoritması daha karmaşık çözümleri de vardır.  
AC3 algoritmasında amaç birer kiriş listesi olarak verilen şartların çözümüdür. Yani algoritmanın girişi bir grafik gibi düşünülebilir ve grafiğin üzerinde hangi varlıktan hangi varlığa şart bulunuyorsa işaretlidir.  
Bu algoritmanın detaylarına geçmeden önce [4 vezir problemi diye bilinen problemin](http://bilgisayarkavramlari.com/2008/04/26/4-vezir-problemi-4-queen-problem/) tanımına ve modellenmesine göz atabilirsiniz.

**SORU-69: 4 vezir problemi (4 queen problem)**

Satranç oyunundan esinlenerek üretilen bu problemde 4×4′lük bir satranç tahtasına (orjinal satranç tahtası 8×8′lik olmakla beraber problem tahtanın küçültülmüş bir kısmında tasarlanmıştır) 4 adet vezirin birbirini yemeden nasıl yerleştirileceği tartışılır. Bilindiği üzere satranç oyununda vezirler satır sütün veya diyagonal olarak (ileri, geri, sağa, sola veya istenilen bir çapraz yöne) istedikleri kadar hareket edebilmektedirler.  
Dolayısıyla örneğin aşağıdaki kırmızı yıldız ile gösterilen köşeye bir vezir konulması durumunda, daireler ile kapatılan karelere vezir konulması imkânsızdır.  
  
Problemin modellenmesinde [kiriş şartlarından (arc constraints)](http://bilgisayarkavramlari.com/2008/04/26/arc-constraint-kiris-sarti/) faydalanılabilir. Buna göre yukarıdaki resimde gösterilen vezir yerleştirme durumu aşağıdaki kirişleri doğurur:  
[](http://bilgisayarkavramlari.com/wp-content/uploads/2008/04/4vezir_kirisler.jpg)  
yukarıdaki şekilde, tahta üzerine yerleştirilmiş olan vezirlerin 4 değişken (variable) için gösterimi yer almaktadır. Buna göre her satırı bir X değişkeni ile gösterecek olursak, Örneğin 3. satır için yani X3 için vezir konulabilecek sütünlar 2 ve 4 numaralı sütunlardır.  
Dolayısıyla 4 vezir problemi 4 satırı gösteren 4 değişken ve her değişkene karşılık gelen değerler ile gösterilmiştir. Problem ilk başta bütün ihtimalleri kapsamaktadır. Yani 4 satırda da 1,2,3,4 numaralı sütunlara vezir konulması mümkündür. Ancak sol üst köşeye vezir konulduktan sonra yukarıdaki şekilde kirişlerin güncellenmesi gerekir.

**SORU-70: Ortak Bölenlerin En Büyüğü (OBEB, GCD, Greatest Common Divisor)**

İki sayının ortak bölenlerinin en büyüğü ile kastedilen iki sayı çarpanlarına ayrıldığında ortak çarpanlarının en büyüğüdür. Örneğin 27 ve 18 sayılarının en büyük ortak bölenleri 9 dur çünkü 9 iki sayıyı da böler, ve iki sayıyı da bölen daha büyük ortak bir sayı yoktur.

iki sayının ortak böleninin en büyüğünün 1 olması demek, iki sayının aralarında asal olması demektir. Aşağıda ortak bölenler ile ilgili iki problemin ıspatı bulunmaktadır:

**1    gcd(m,n)=1  =>  gcd(m+n,m-n)=1**

 1.       Sorunun cevabı:

K gibi bir sayımız olsun öyle ki k=gcd(m+n,m-n) olsun.

Dolayısıyla m+n = k\*p ve m-n=k\*q denilebilir (şayet ortak bölen k ise her iki sayıda p ve q gibi iki farklı sayının çarpımı olacaktır (p ve q aynı olabileceği gibi 1 de olabilir).

Şayet k sayısı çift sayı ise k\*p ve k\*q sayıları da çifttir. Bu ise imkansızdır çünkü sorunun ilk kısmında gcd(m,n)=1 verilmiş

Şayet k sayısı tek syaı ise o halde p ve q ‘nun da tek olması gerekir çünkü birisi çift olursa sonuçta çift olacaktır bu da ilk şart ile çelişir.

Şayet p ve q tek sayılar ise p –q çift sayı demektir. Öyleyse r=(p-q)/2 gibi bir tam sayı tanımlanabiilr

(m+n) – (m-n) = k\*p – k\*q  
2n = k\*(p-q)  
n = k\*(p-q) / 2  
n = k\*r  
dolayısıyla k, n’in bir çarpanıdır:

(m+n) = k\*p  
m + k\*r = k\*p  
m = k\*p – k\*r  
m = k\*(p-r)

Dolayısıyla k aynı zamanda m’in de bir çarpanıdır denilebilir

Dolayısıyla gcd(m,n) değeri en az k olmalıdır.

Buradaki k değerinin k>1 olması gerekir denilirse bu k=1 değeri ile çelişir. Aksi durum ise zaten ilk eşitliğin ıspatı olmuş olur.

**2  gcd(fn,fn+1)=1**

 .       bu ardışık iki sayının ortak böleninin 1 olduğunu söyler. Basitçe şu şekilde düşünülebilir

P gibi bir tam sayı alalım. Bu sayının katları p değeri kadar artar. Örneğin 2’nin katları 2 4 6 8 gibi 3’ün katları ise 3 6 9 gibi artacaktır. Dolayısıyla pm ile p(m+1) arasındaki fark en az p kadar olaiblir. Ve ancak bu durumda gcd(pm, p(m+1))=p olabilir. Oysaki ardışık sayıların farkı 1 ‘dir dolayısıyla iki ardışık sayının gcd değeri 1 olabilir. Daha resmi olarak:

P ve q gibi iki sayı olsun.

gcd(p,q) = x ise |p-q| = xk olmalıdır.

|Fn-fn+1| =1 olduğuna göre gcd (fn,fn+1) =1 olmalıdır.

Dolayısıyla hem tek hem de çift için ıspatlanan durum (başka bir durum olamayacağına göre) doğrudur.

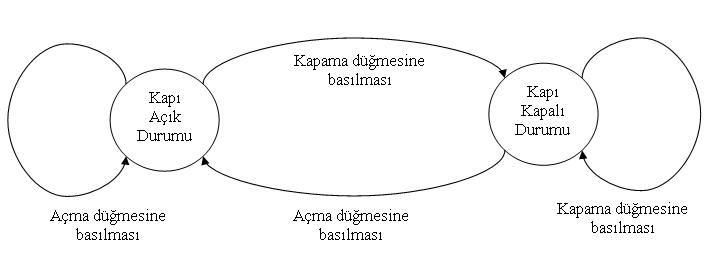
**SORU-71: Belirsiz Çokterimli Tam (NP-Complete, Nondeterministic Polynomial Complete)**

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](http://www.bilgisayarkavramlari.com/2009/06/27/turing-makinesi-turing-machine/) de yararlanılır.

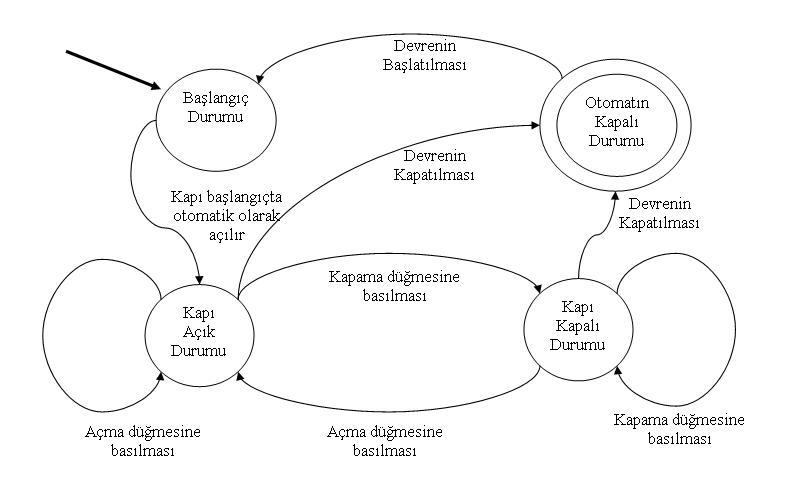
[Turing makineleri](http://www.bilgisayarkavramlari.com/2009/06/27/turing-makinesi-turing-machine/) beliri (deterministic) ve belirsiz (nondeterministic) olarak ikiye ayrılır. Buna göre NP-Complete bir problem Belirsiz Turing Makinesi tarafından belirli zamanda çözülebilmektedir.  
Ayrıca problemleri kümelerken diğer bir küme olan P kümesi yani çokterimli (polynomial) kümesi, NP kümesinin bir alt kümesi olarak kabul edilir.

**SORU-72: Sonlu Durum Makinası (Finite State Machine, Finite State Automaton)**

Sonlu durum makinaları bir çizim şeklidir. Bu çizim şeklinde çeşitli durumlar ve bu durumlar arası geçiş şekilleri gösterilir. Örneğin aşağıda basit bir kapı açma ve kapama makinesi verilmiştir:

[](http://bilgisayarkavramlari.com/2007/12/25/sonlu-durum-makinasi-finite-state-machine-finite-state-automaton/sonlu-durum-makinasi-finite-state-machine-finite-state-automaton-kapi-acma-ve-kapama-otomati/)

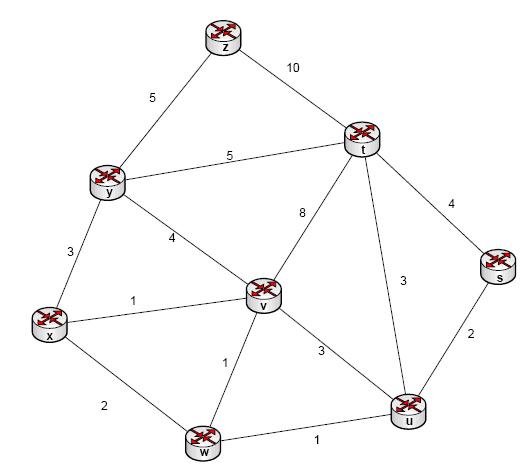
Yukarıdaki şekilde makine, açık durumdan kapalıya geçmek için kapama düğmesine basılmasını bekler. Tersi durumda da kapalıdan açığa geçmek için kapama düğmesinin basılmasını bekler. Durum makinemizde kapalı durumdayken kapama düğmesine ve açık durumdayken açma düğmesine basılması bir durum değişikliği doğurmaz ve istenildiği kadar basılabilir.  
Yukarıdaki şekilde bir başlangıç veya bitiş durumu belirtilmemiştir. Yani makine sonsuza kadar çalışmaktadır ve herhangi bir başlangıç koşulu yoktur. Yukarıdaki bu makinanın bir de açma ve kapama tuşları olduğunu düşünelim. Bu durumda makinenin başlangıcı açma düğmesi ve bitişi de kapama düğmesi ile olacaktır. Aşağıdaki şekilde yeni makinemizin çizimi gösterilmiştir:

[](http://bilgisayarkavramlari.com/2007/12/25/sonlu-durum-makinasi-finite-state-machine-finite-state-automaton/sonlu-durum-makinasi-finite-state-machine-finite-state-automaton-kapi-acma-ve-kapama-otomati-2/)

Yukarıdaki şekilde sonlu durum makinemize ilave olarak başlangıç ve bitiş durumları da eklenmiştir. Buna göre kalın gidi oku başlangıç durumunu gösterir ve makinemiz buradan başlar. Çift çember içindeki durum ise bitiş durumudur ve istenirse makine burada sona erdirilir, veya çalışmaya devam edebilir. Görüldüğü üzere yukarıdaki şekilde kapı otomatı başlatıldıktan sonra istenildiği kadar açma kapama işlemi yapılmakta, gelen bu komutlara göre kapını durumu değişmektedir. Kapı otomatı kapatıldıktan sonra gelen açma ve kapama emirleri doğal olarak icra edilemez.

**SORU-73: Kruskal Asgari Tarama Ağacı Algoritması**

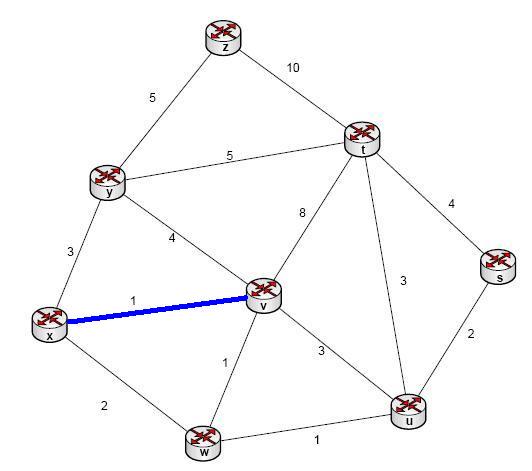
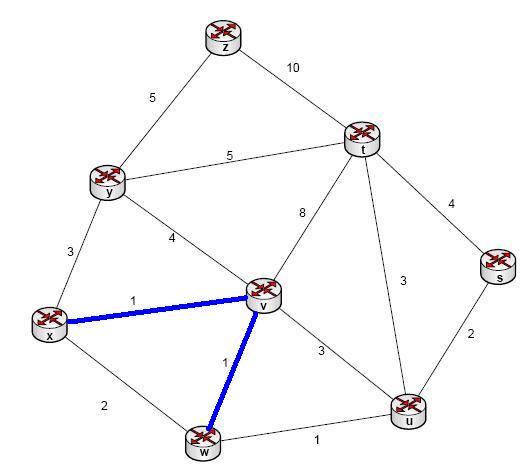
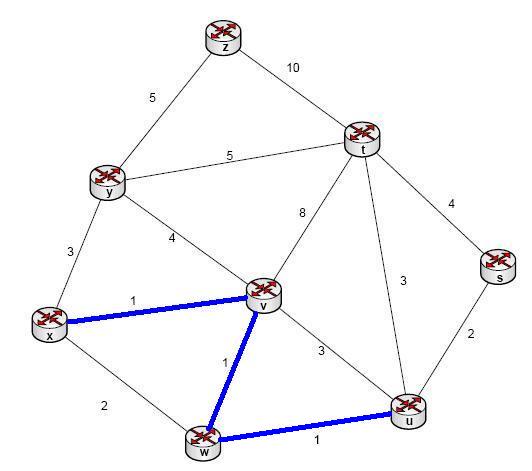
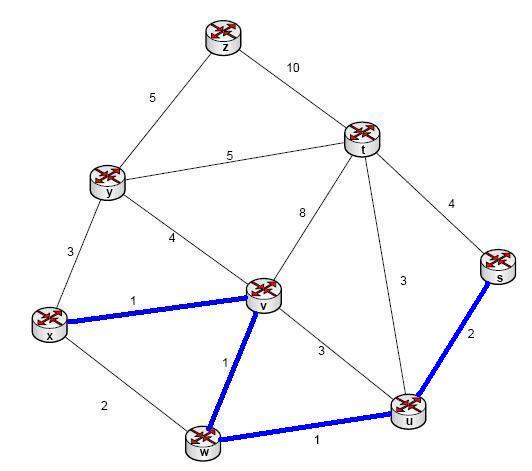
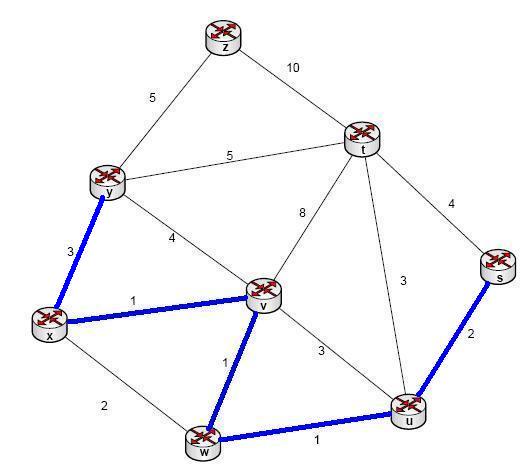
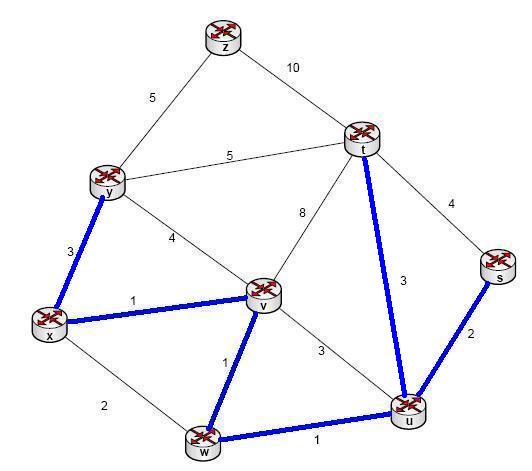
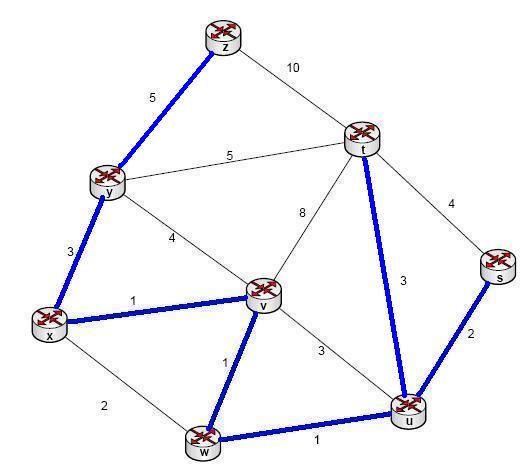
Bir [asgari tarama ağacı (minimum spanning tree)](http://bilgisayarkavramlari.com/2007/12/24/asgari-tarama-agaci-en-kisa-orten-agac-minimum-spanning-tree/) algoritması olan Dijkstra algoritması, işaretlemiş olduğu komşuluklara en yakın düğümü bünyesine katarak ilerler. Buna göre aşağıdaki grafiğin [asgari tarama ağacını](http://bilgisayarkavramlari.com/2007/12/24/asgari-tarama-agaci-en-kisa-orten-agac-minimum-spanning-tree/) çıkaralım:

[](http://bilgisayarkavramlari.com/wp-content/uploads/2007/12/asgari_tarama_agaci.jpg)

Yukarıdaki grafikte her düğüm için bir temsili harf ve her bağlantı için bir ağırlık değeri atanmıştır. Buna göre her düğümden diğerine gitmenin maliyeti belirlenmiştir.

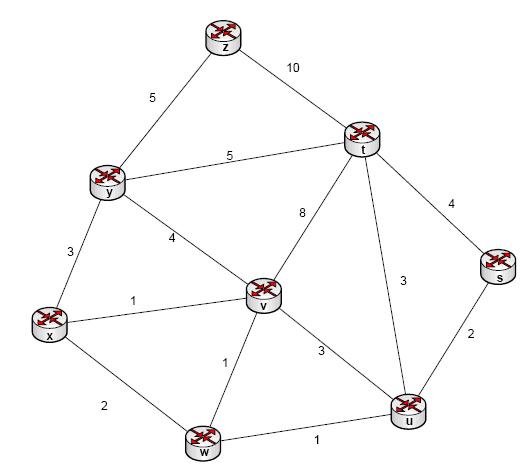
Kruskal algoritmasında bütün yollar listelenip küçükten büyüğe doğru sıralanır. Bu liste yukarıdaki grafik için aşağıda verilmiştir:

x-v:1  
w-v:1  
w-u:1  
x-w:2  
u-s:2  
x-y:3  
t-u:3  
u-v:3  
y-v:4  
s-t:4  
y-z:5  
y-t:5  
z-t:10

Yukarıdaki liste çıkarıldıktan sonra sırasıyla en küçükten en büyüğe doğru komşuluklar işaretlenir. Bu işaretleme sırasında ada grupları ve grupların birbiri ile ilişkisine dikkat edilir. Yani şayet listedeki iki düğüm harfi de aynı adadan ise bu bağlantı atlanır. Aşağıda sırasıyla bu grafikteki adaların oluşması ve asgari tarama ağacının çıkarılması gösterilmiştir:  
x-v:1  
[](http://bilgisayarkavramlari.com/wp-content/uploads/2007/12/asgari_tarama_agaci_kruskal1.jpg)  
w-v:1  
[](http://bilgisayarkavramlari.com/wp-content/uploads/2007/12/asgari_tarama_agaci_kruskal2.jpg)  
w-u:1  
[](http://bilgisayarkavramlari.com/wp-content/uploads/2007/12/asgari_tarama_agaci_kruskal3.jpg)  
x-w bağlantısı atlanır, çünkü iki düğüm de zaten dolaşılmıştır bunun yerine u-s:2 bağlantısına atlanır.  
[](http://bilgisayarkavramlari.com/wp-content/uploads/2007/12/asgari_tarama_agaci_kruskal4.jpg)  
x-y:3  
[](http://bilgisayarkavramlari.com/wp-content/uploads/2007/12/asgari_tarama_agaci_kruskal5.jpg)  
t-u:3  
[](http://bilgisayarkavramlari.com/wp-content/uploads/2007/12/asgari_tarama_agaci_kruskal6.jpg)  
Bu noktadan sonra u-v:3 , y-v:4 , s-t:4 , y-z:5 bağlantılarındaki her iki düğümde aynı adada olduğu için atlanır ve y-t:5 bağlantısına geçilir.  
[](http://bilgisayarkavramlari.com/wp-content/uploads/2007/12/asgari_tarama_agaci_kruskal7.jpg)  
z-t:10 bağlantısı ise iki düğüm de gezildiği için yine gereksizdir.

**SORU-74: Prim asgari tarama ağacı Algoritması**

Bir [asgari tarama ağacı (minimum spanning tree)](http://bilgisayarkavramlari.com/2007/12/24/asgari-tarama-agaci-en-kisa-orten-agac-minimum-spanning-tree/) algoritması olan Prim algoritması, işaretlemiş olduğu komşuluklara en yakın düğümü bünyesine katarak ilerler. Buna göre aşağıdaki grafiğin [asgari tarama ağacını](http://bilgisayarkavramlari.com/2007/12/24/asgari-tarama-agaci-en-kisa-orten-agac-minimum-spanning-tree/) çıkaralım:

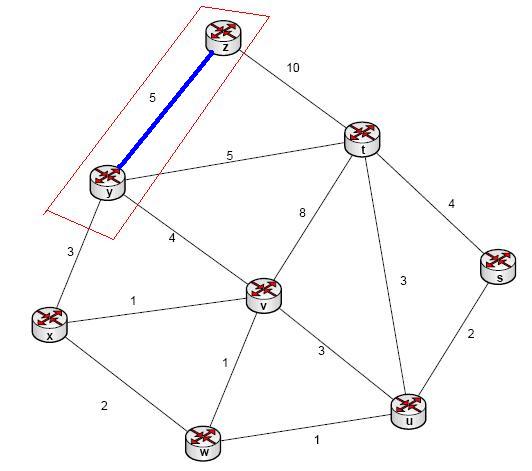
[](http://bilgisayarkavramlari.com/wp-content/uploads/2007/12/asgari_tarama_agaci.jpg)

Yukarıdaki grafikte her düğüm için bir temsili harf ve her bağlantı için bir ağırlık değeri atanmıştır. Buna göre her düğümden diğerine gitmenin maliyeti belirlenmiştir.

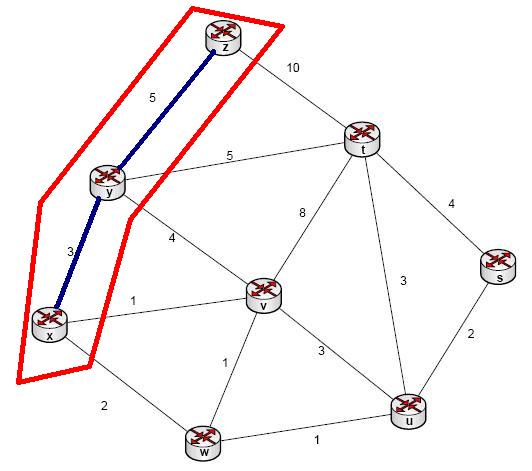
Prim algoritmasında rasgele bir başlangıç noktası seçilir. Örneğin bizim başlangıç noktamız “z” düğümü olsun. Bu durumda ilk inceleyeceğimiz komşuluk, “z” düğümünden gidilebilen düğümler ve maliyetleri olacaktır.

z düğümünden gidilebilen düğümler ve maliyetleri aşağıda listelenmiştir:

y:5  
t:10

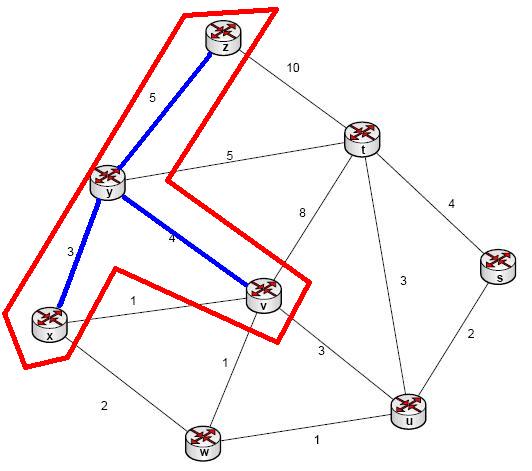
Prim algoritması bu listedeki en küçük maliyetli komşuyu bünyesine dahil eder. Buna göre yeni üyelerimiz {z,y} olacaktır ve gidilen yollar {z-y:5} olacaktır. (ilk üyeler listesinde şimdiye kadar ziyaret edilmiş düğümler bulunur. Bu düğümler listesinde zaten olan bir düğüm listeye eklenemez. yollar listesinde ise hangi düğümden hangi düğüme ne kadar maliyetle gidildiği tutulur.) Dolayısıyla grafiğimizde Prim algoritması tarafından işaretlenen düğümler aşağıda gösterilmiştir:  
[](http://bilgisayarkavramlari.com/wp-content/uploads/2007/12/asgari_tarama_agaci_dijkstra1.jpg)

şimdi üyelerimizin durduğu listedeki bütün düğümlerin komşularını listeleyelim:  
t:5  
t:10  
v:4  
x:3

yukarıdaki listede t düğümüne iki farklı gidiş bulunmaktadır ( hem z hem de y üzerinden). Biz algoritmamıza devam edip en küçük yolu bünyemize dahil edelim. En yakın komşu x:3′tür. Bu durumda üyelerimiz {z,y,x} olacak ve yollarımız {z-y:5,y-x:3} olacaktır. Bu durum aşağıdaki grafikte gösterilmiştir:  
[](http://www.bilgisayarkavramlari.com/2007/12/29/owl-lite/attachment/219/)

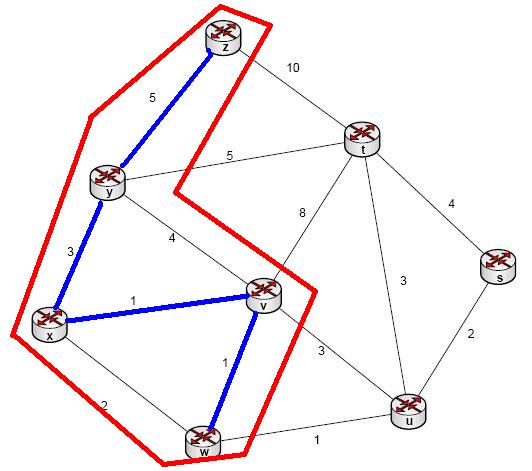
Yeniden komşularımızı listelersek:  
t:5  
v:1  
w:2

yukarıdaki listede bünyemize aldığımız adadan, bir düğüme giden birden fazla yol bulunması durumunda en kısası alınmıştır. Bu durumda listenin en küçük elemanı olan v:1 tercih edilir ve üyelerimiz {z,y,x,v} yollarımız ise {z-y:5,y-x:3,x-v:1} olur. Durum aşağıdaki grafikte gösterilmiştir:

[](http://www.bilgisayarkavramlari.com/2007/12/29/owl-lite/attachment/220/)

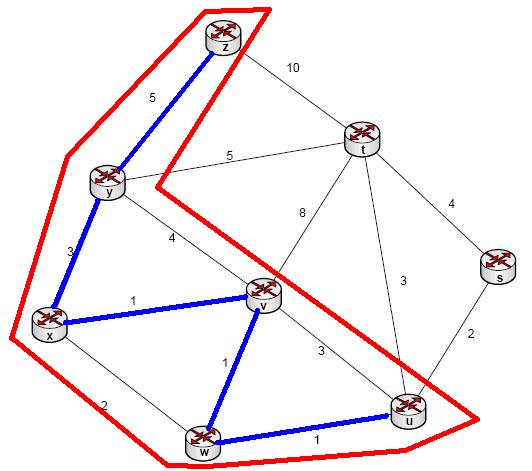
Yeniden komşularımızı listelersek:  
t:5  
u:3  
w:1

yukarıdaki listede bünyemize aldığımız adadan, bir düğüme giden birden fazla yol bulunması durumunda en kısası alınmıştır. Bu durumda listenin en küçük elemanı olan w:1 tercih edilir ve üyelerimiz {z,y,x,v,w} yollarımız ise {z-y:5,y-x:3,x-v:1,v-w:1} olur. Durum aşağıdaki grafikte gösterilmiştir:

[](http://www.bilgisayarkavramlari.com/2007/12/24/dijkstra-algoritmasi/kruskal-asgari-tarama-agaci-algoritmasi/)

Yeniden komşularımızı listelersek:  
t:5  
u:1

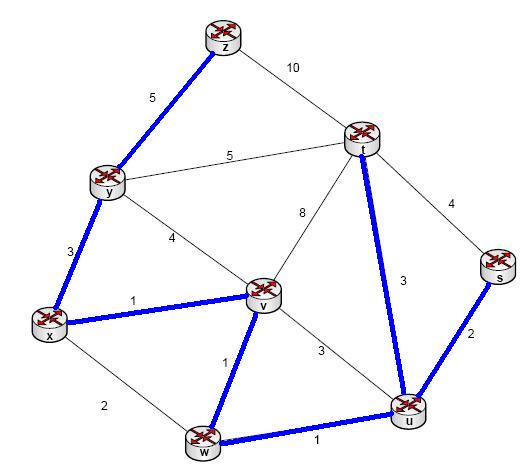
yukarıdaki listede bünyemize aldığımız adadan, bir düğüme giden birden fazla yol bulunması durumunda en kısası alınmıştır. Bu durumda listenin en küçük elemanı olan u:1 tercih edilir ve üyelerimiz {z,y,x,v,w,u} yollarımız ise {z-y:5,y-x:3,x-v:1,v-w:1,w-u:1} olur. Durum aşağıdaki grafikte gösterilmiştir:

[](http://www.bilgisayarkavramlari.com/2007/11/30/de-morgan-kurali-de-morgan-rule/f-ab-ab-esitligi-icin-cozum-haritasi-3/sonlu-durum-makinasi-finite-state-machine-finite-state-automaton-kapi-acma-ve-kapama-otomati/)

Yeniden komşularımızı listelersek:  
t:3  
s:2

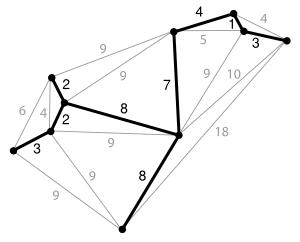
yukarıdaki listede bünyemize aldığımız adadan, bir düğüme giden birden fazla yol bulunması durumunda en kısası alınmıştır. Bu durumda listenin en küçük elemanı olan s:2 tercih edilir ve üyelerimiz {z,y,x,v,w,u,s} yollarımız ise {z-y:5,y-x:3,x-v:1,v-w:1,w-u:1,u-s:2} olur. Durum aşağıdaki grafikte gösterilmiştir:

Son komşumuz olan t için en kısa ulaşım  
t:3 değeridir ve u üzerinden sağlanır. Bu durumda listenin en küçük elemanı olan t:3 tercih edilir ve üyelerimiz {z,y,x,v,w,u,s,t} yollarımız ise {z-y:5,y-x:3,x-v:1,v-w:1,w-u:1,u-s:2,u-t:3} olur. Sonuçta elde edilen [asgari tarama ağacı](http://bilgisayarkavramlari.com/2007/12/24/asgari-tarama-agaci-en-kisa-orten-agac-minimum-spanning-tree/) aşağıda verilmiştir:

[](http://www.bilgisayarkavramlari.com/2007/12/24/dijkstra-algoritmasi/sonlu-durum-makinasi-finite-state-machine-finite-state-automaton/)

**SORU-75: asgari tarama ağacı (en kısa örten ağaç, minimum spanning tree)**

Asgarai tarama ağacı, ağırlıklık bir ağda (weighted graph, yani her düğümü birbirine bağlayan yolların maliyeti (ağırlığı) olması durumu), bütün düğümleri dolaşan en kısa yolu verir. Örneğim aşağıdaki grafikte bütün düğümlere uğrayan en kısa yol işaretlenmiştir:



**SORU-76: fonksiyon göstericileri (function pointer)**

fonksiyon göstericilerinin amacı, programlama dilinde bulunan fonksiyonları gösteren birer referans bilgisi tutmaktır. Bu sayede gösterilmekte olan fonksiyon için [hafızada](http://www.bilgisayarkavramlari.com/2008/11/07/rastgele-erisilebilir-bellek-random-access-memory-ram/) ayrılmış olan yere erişmek ve dolayısıyla örneğin fonksiyonun yerel değişkenlerine ulaşmak mümkündür. Aşağıda C dilinde yazılmış bir fonksiyon göstericisi kullanan kod örneği verilmiştir:

#include

#include

void func(int);

main(){

void (\*fp)(int);

fp = func;

(\*fp)(1);

fp(2);

}

void

func(int arg){

printf("%dn", arg);

}

Yukarıdaki kodda “func” isminde bir fonksiyon tanımlanmıştır. Ayrıca void tipinde dönüş değerine sahip ve int tipinde parametre alan bir fonksiyoö göstericisi “fp” tanımlanmıştır. Dikkat edilirse “func” fonksiyonunun ve “fp” göstericisinin hem parametreleri hem de dönüş değerleri aynıdır. Bu durum bir göstericinin foknsiyonu göstermesi (refer etmesi ) için gereklidir. Bu gösterme işlemi atama satırı olan  
fp=func;  
satırı ile yapılmaktadır. artık bu satırdan sonra “fp” göstericisine verilen her değer “func” fonksiyonuna verilmiş gibi icra edilir. Yani yukarıdaki kod çalışıtırıldığında ekranda önce 1 sonra 2 görülmektedir.

Karıştırılmaması gereken bir nokta:  
void (\*fp)(int);  
void \*fp(int);  
Yukarıdaki iki satır birbirinden farklıdır. İlk satır bir fonksiyon göstericisini, ikinci satır ise gösterici döndüren bir fonksiyonu tanımlarken kullanılmalıdır. Yani yukarıdaki iki satır aynı değildir.

**Fonksiyon göstericileirinin fonksiyonlara parametre olarak verilmesi.**   
bir fonksiyon parametre olarak bir fonksiyon göstericisini alabilir. Aşağıda bunu yapan temsili kod verilmiştir:

#include

#include

int func(int);

void PassPtr(int (\*pt2Func)(int))

{

int sonuc = (\*pt2Func)(12);

printf("%d",sonuc);

}

main(){

int (\*fp)(int);

fp = func;

PassPtr(fp);

}

int

func(int arg){

return ++arg;

}

Yukarıdaki örnek kodda, func ismindeki fonksiyonu gösteren fp isminde bir fonksiyon göstericisi tanımlanmıştır. Bu gösterici PassPtr fonksiyonuna parametre olarak verilmiştir. Bu kod çalıştırıldığında ekranda 13 sayısı görlür çünkü, func fonksiyonunu gösteren fp göstericisi PassPtr fonksiyonunun içinden çağrılmış ve değer olarak 12 parametresi atanmıştır. Fonksiyon incelenirse parametre olarak aldığı sayıyı bir arttırdığı görülür. Bu durumda ekrandaki değer 13 olmaktadır.  
Dikkat edilirse yukarıdaki kod ile, bir fonksiyona başka bir fonksiyon verilebilmektedir. Bu sayede genel amaçlı fonksiyonlar yazılarak ve bu fonksiyonlar parametre olarak geçirilerek programlamada avantaj elde edilebilmektedir.

**Fonskiyon göstericilerinden dizi oluşturmak:**  
Fonksiyon göstericileri de birer gösterici olduğu için normal bir göstericiye yapılan herşey bu göstericilere de yapılabilir. Bunlardan birisi de bir dizi tanımlamaktır.  
Aşağıdaki örnek kodu inceleyelim:

#include

#include

typedef int (\*pt2Function)(int);

int

func(int arg){

return ++arg;

}

int main(){

pt2Function funcArr1[10] = {NULL};

int (\*funcArr2[10])(int) = {NULL};

funcArr1[0] = funcArr2[1] = func;

printf("%dn", funcArr1[1](12));

}

Yukarıdaki örnek kodda, iki adet fonksiyon gösterici dizisi tanımlanmıştır. İlk dizi olan funcArr1 dizisi, typedef marifeti ile tanımlanmıştır ve bu tip tanımı daha önceden yapılmıştır. İkinci dizi olan funcArr2 dizisi ise daha önceden tanımlanmış herhangi bir tip kullanmaksızın tanımlanmıştır. Sonuçta ekranda 13 sayısını gördüğümüz bu yukarıdaki kodda fonksiyon göstericisi, bir dizinin elemanı olarak tutulmakta ve çağrılmaktadır.

**SORU-77: otomat yönelimli programlama (automata based programming)**

otomat yönelimli programlama yaklaşımı, kaynağını otomatlar (automata)’dan alır ve sonlu durum makinaları (finite state machine, FSM) ile tasarlanan bir makinanın kodlanmasını hedefler.

Basitçe C dilindeki switch komutlarının dallanmasına benzer bir şekilde her durumdan bir sonraki duruma geçiş yapan bu programlama yaklaşımında amaç durumlar arası geçişin tasarıma uygun olarak kolay bir şekilde gerçeklenmesidir. Bunun için çeşitli dil çevirici araçlar olduğu gibi günümüz dillerinin pek çoğunda kullanılan eylem bazlı programlama (event based programming) aslında bir otomat yönelimli programlama tipdiri.

Bu eylem bazlı programlama yaklaşımında yapılan, her eylem için bir alt program tanımlayarak, gerçekleşen olaylar sonucunda bu alt programlara yönlendirme yapılmasıdır. Örneğin visual basic, C++ veya JAVA gibi dillerde ekrandaki bir düğmeye (button) tıklanması durumunda bir fonksiyonun çağrılması veya javascript için onclick event, (tıklama eylemi) bu yaklaşıma birer örnektir.

**SORU-78: fonksiyonel programlama (functional programming)**

Programlama yaklaşımlarından birisi olan fonksiyonel programlama günümüz dillerinin neredeyse tamamında kullanılmaktadır. Bu yaklaşımda matematik fonksiyonlarında olduğuna benzer bir şekilde alt programlar tanımlanmakta ve bu alt programların değişik argümanlar ile çalışması sağlanmaktadır. Bu yaklaşım basitçe:

 Kod tekrarını engellemekte ve aynı kodun farklı şartlar için tekrar tekrar çalışmasını sağlamaktadır

 Kodun okunaibilirliğini arttırmakta ve kod analizini daha kolay hale getirmektedir.

 Programın tasarlanması aşamasında tasarımcıya modüler yaklaşım yapmasını sağlamaktadır.

Bir dilin fonksiyonel olması dilde fonksiyon veya prosedüre benzeri özellikler bulundurması ile sağlanır. Bu özellikleri tanımlamak gerekirse:

 Sıfır veya daha fazla argüman ile giriş yapılabilen

 Sıfır veya daha fazla argünan ile çıkış yapabilen

 İç yapısında dilin izin verdiği alt programları barındıran yapıdır.

Yukarıdaki bu maddeleri bir [kara kutu (blackbox)](http://www.bilgisayarkavramlari.com/2009/08/29/kara-kutu-yaklasimi-black-box/) yaklaşımı olarak da düşünebiliriz. Yani alt programların, dış dünya ile (programın geri kalanı ile) olan tek bağlantıları almış oldukları ve geri döndürmüş oldukları argümanlardır (paramatrelerdir).

Aşağıda bir fonksiyon örneği verilmiştir: ([JAVA, C , C++, C# dillerinde kabul edilir koddur):](http://bilgisayarkavramlari.com/2007/04/14/java-programlama-dili/)

int topla ( int a, int b){  
return a + b;  
}

Yukarıdaki kod incelendiğinde, a ve b, fonksiyona verilen argümanlardır (parametrelerdir). Dönüş değeri olarak tam sayı (integer) tipi kullanılmış ve bu durum return komudu ile belirtilmiştir. a+b eylemi ise bir alt programdır. Yani bu fonksiyon çağrıldığında icra edilen program parçasıdır.

Fonksiyonel programlama, [yapısal programlamanın](http://bilgisayarkavramlari.com/2007/12/18/yapisal-programlama-structured-programming/) gerektirdiği bir yaklaşımdır. Buna göre fonksiyonel programlama kullanılan bütün diller [yapısal programlama](http://bilgisayarkavramlari.com/2007/12/18/yapisal-programlama-structured-programming/) yaklaşımına uygundur denilebilir. Ancak tersi doğru değildir. Her ne kadar [yapısal programlamanın](http://bilgisayarkavramlari.com/2007/12/18/yapisal-programlama-structured-programming/) tanımında bir alt programın varlığı zarurî olsa da bu alt program basit bir if bloğu olarak da düşünülebilir. Yani okuyucu kod blokları ile fonksiyonları karıştırmamalıdır.  
 [Nesne yönelimli programlama](http://bilgisayarkavramlari.com/2007/04/14/nesne-yonelimli-programlama-object-oriented-programming/) yaklaşımınlarında fonksiyonel programlama kullanılmaktadır. Dolayısıyla her [nesne yönelimli programlama](http://bilgisayarkavramlari.com/2007/04/14/nesne-yonelimli-programlama-object-oriented-programming/) yaklaşımı, fonksiyonel programlamayı barındırmaktadır. [Nesne yönelimli programlama](http://bilgisayarkavramlari.com/2007/04/14/nesne-yonelimli-programlama-object-oriented-programming/) terminolojisinde, fonksiyonlara metod ismi verilmektedir.

**SORU-79: En uzun Ortak Küme (longest common subsequence, Lcs)**

İki küme arasındaki ortak elamanların (sıralı olmak şartıyla) en uzun ortaklığını arar. Örnek:  
A-> {X,M,J,Y,A,U}  
B-> {M,Z,J,A,W,X,U}  
olarak verilmiş olsun. Bu iki kümenin, sırası bozulmadan ortak olan en uzun alt kümesi:  
LCS -> {M,J,A,U} olarak bulunur.

Bu problem karmaşıklık açısından NP-hard problemlere bir örnektir. Aynı zamanda çözüm için önerilen yöntemler incelendiğinde [dinamik programlamanın](http://bilgisayarkavramlari.com/2007/12/03/dinamik-programlama-dynamic-programming/) anlaşılması için ideal bir örnektir.

Örnek bir çözüm yöntemini inceleyelim:  
Problemin ilk akla gelen ve en basit çözümü kümelerden birisini ana küme seçerek, bu kümedeki elemanları teker teker diğer küme ile sınamak olabilir.  
Örneğin A kümesi seçilmiş olsun. A kümesinin ilk elemanından son elemanına kadar ortak elemanlara bakılacaktır. Bu durumda ilk eleman X, B kümesinde aranacaktır, daha sonra X ile birlikte U aranacaktır. ihtimallerin sonuna gelindiğinde bu ikili bırakılıp yeni bir ihtimal denenecektir:  
Örnek çalışma:  
A kümesinden X seçilir. B kümesinde sınanır. Karşılığı bulununca LCS1 olarak kaydedilir. devamı olan harflerden uygun olan seçilir (bu örnekte U) LCS1′e eklenir.  
LCS1->{X,U}  
LCS2->{M,J,A,U}  
LCS3->{J,A,U}  
LCS4->Y’nin eşi diğer kümede yok  
LCS5->{A,U}  
LCS6->{U}

yukarıdaki yönteme göre A, kümesinin bütün elemanları başlangıç elemanı olarak denenmiştir. Fakat [dinamik programlama](http://bilgisayarkavramlari.com/2007/12/03/dinamik-programlama-dynamic-programming/) yaklaşımı incelenecek olursa bu işlemin çok daha kısa zamanda yapılabileceği görülür:  
Aşağıda bir çözüm yöntemi olarak tablo kullanılması önerilmiştir. Buna göre tablodaki her hücre, ilgili satır ve sütün başlığına kadar olan en uzun ortak kümeyi tutmaktadır:

yukarıdaki şekilde örnek bir tablo verilmiştir. Buna göre verilen problemdeki en uzun ortak kümeyi bulmak hedeflenmektedir. Tablonun sağ alt köşesinde bulunan bu küme sorunun cevabıdır. Sorunun çözümünde tablo oluşturulurken her hücre kendisinin bir üstündeki veya bir solundaki hücreyi aynen kopyalamakta, ve üzerine satır ve sütün ismini şayet ortaksa yazmaktadır. Örneğin son oluşturulan sağ alt köşedeki küme, solundaki ve tepesindeki kümelerin kopyalanması ile oluşur. Bu kopyalama işlemine U elemanı ilave edilir çünkü bu hücrenin bulunduğu sütun ile satır ismi aynıdır.  
Bu işlem yapılırken karşılaşılabilecek bir problem de solundaki bilgi ile tepesindeki bilginin aynı olmamasıdır. Bu durum kırmızı çember içerisinde gösterilmiştir. Bu durumda daha uzun bir küme elde edilince kısa kalan küme bırakılarak, uzun küme ile yola devam edilir. (problemin en uzun kümeyi bulmak olduğunu hatırlayınız).  
Bu işlem sonunda aşağıdaki ortak küme uzunluklarını gösteren tablo elde edilir:

| 0 1 2 3 4 5 6 7

| M Z J A W X U

-----|-----------------

0 | 0 0 0 0 0 0 0 0

1 X | 0 0 0 0 0 0 1 1

2 M | 0 1 1 1 1 1 1 1

3 J | 0 1 1 2 2 2 2 2

4 Y | 0 1 1 2 2 2 2 2

5 A | 0 1 1 2 3 3 3 3

6 U | 0 1 1 2 3 3 3 4

Buna göre her seferinde kendinden önceki elemanlara bakılması önlenmiş olur. Yani örneğin tablonun 4,5 hücresine bakıldığında A harfleri kontrol edilmektedir. Bu kontrol sırasında kendisinden önceki harflerin kontrolüne gerek kalmaz. (tekrar B kümesinin MZJ harflerine bakılmaz), bu sayede ortak yapılan işlemler, [dinamik programlama](http://bilgisayarkavramlari.com/2007/12/03/dinamik-programlama-dynamic-programming/) ilkesine uygun olarak elenmiş olur.

**SORU-80: Dinamik Programlama (Dynamic programming)**

Bir problem tahlil ve çözüm yöntemi olan dinamik programlama yapı olarak [parçala fethet](http://bilgisayarkavramlari.com/2007/12/03/parcala-fethet-yontemi-divide-and-conquer/) yöntemine benzer. Tek farkı problemi parçalara böldükten sonra aynı problemin tekrarı olan parçaları bir kerede çözüp her tekrar için ayrı bir çözüm yapmamasıdır.

Örneğin fibonacci serilerini ele alalım, Bu seriyi üreten örnek kod aşağıda verilmiştir:  
  
int fibonacci(int n)  
{  
if (0 == n) {  
return 0;  
} else if (1 == n) {  
return 1;  
} else {  
return fibonacci(n - 2) + fibonacci(n - 1);  
}  
}

Yukarıda verilen bu recursive (kendi kendini çağıran) koda bakıldığında ve kodun tahlili yapıldığında aşağıdaki fonksiyon iç içe çağırma ağacı (recursion tree ) fark edilir:

fibonacci(4)

+------------------------------

| |

fibonacci(2) fibonacci(3)

+----------------- +---------------

| | | |

fibonacci(0) fibonacci(1) fibonacci(1) fibonacci(2)

+-----------

| |

fibonacci(0) fibonacci(1)

yani yukarıdaki örnekte, fibonacci(4) fonksiyonu için çağırma işlemleri sırasıyla gösterilmiştir. Dikkat edilirse fonksiyonlar açıldğında kendisinden önceki iki sayının toplamını bulmakta, bu işlemi yaparken de ortak elemanlar kullanmaktadır. Örneğin fibonacci(2) fonksiyonu ağacın iki farklı yerinde bulunmaktadır ve iki farklı kere içi hesaplanmıştır. İşte dinamik programlamada amaç bunu kaldırarak bir kerede çözüme ulaşmaktır.

Dinamik programlamada aşağıdaki adımlar takip edilebilir:  
Verimli bir çözüm için problemin yapsınının çıkarılması  
Kendini çağıran bir şekilde (Recursive) verimli çözüme değer atanması  
Verimli çözümün değerini aşağıdan yukarı (bottom-up) olarak hesaplanması  
Hesaplanan bu çözümle daha verimli bir çözüm varsa aranıp üretilmesi

**SORU-81: parçala fethet yöntemi (divide and conquer)**

Bu yöntem algoritma analizinde çok kullanılan, bir algoritmayı tahlil etmek veya yeni bir algoritma oluşturmak için kullanılan yaklaşımlardan birisidir.

Bu yaklaşıma göre problem ufak ve çözülmesi nispeten daha kolay olan parçalara bölünür. Her parça ayrı ayrı çözüldükten sonra sonuçlar birleştirilerek genel problemin çözümü elde edilir.

**SORU-82: Alt küme toplamı problemi (subset sum problem)**

Algoritma teorisinde meşhur problemlerden birisidir. [NP-Complete problemlere](http://bilgisayarkavramlari.com/2008/03/24/belirsiz-cokterimli-tam-np-complete-nondeterministic-polynomial-complete/) iyi bir örnektir. Problemin tanımı aşağıdaki şekildedir:

verilen eksi ve artı tam sayılar kümesinin herhangi bir alt kümesinin toplamının 0 olduğunu bulmak.

Bu problemin kontrol edilmesi oldukça basittir ancak verilen sayı kümesinden yukarıdaki tanımı sağlayan bir alt küme bulmak kolay değildir.

Problemin örnek çözümü:

verilen sayı kümesi : { 1,4,-5,2,7,-4,2,9} olsun. Bu sayı kümesinin öyle bir alt kümesi var mıdır ki toplamı 0 olsun?

evet böyle bir alt küme buluna bilir örnek:  
{-4,4} veya {-4,2,2} veya {1,4,-5} veya {9,-4,-5} … gibi

yukarıdaki örnek için bütün çözümleri verecek bir yöntemin geliştirilmesi mümkün olsa bile bu yöntemin çalışmas süresi problem kümesi büyüdükçe çok çok yüksek değerlere çıkacaktır.

**SORU-83: Pointer (Gösterici) ve Diziler (Arrays)**

Pointerlar ve kullanımları

Yazan : Şadi Evren ŞEKER

Pointer (gösterici) basitçe bir değişkenin bir [hafıza](http://www.bilgisayarkavramlari.com/2008/11/07/rastgele-erisilebilir-bellek-random-access-memory-ram/) alanını göstermesi demektir. C dilinde pointerlar:

Veritipi \*pointeradi;

Şeklinde tanımlanır. Burada veritipi int, char, float gibi değişken tipleridir. Pointer adı ise bir değişken adının taşıması gereken özellikleri taşıyan ve tanımlandığı scope (geçerlilik alanında) yaşayacak olan değişken adıdır.

Bir değişkenin başına & işareti geldiğinde ise bu değişkenin adresini temsil eder. Yani değişken hafızada nereye konulduysa bu konulan yerin adresine bu işaret ile ulaşılabilir.

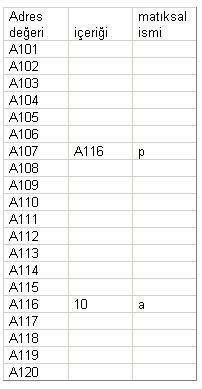
Örneğin,

int \*p;

int a=10;

p=&a;

satırından sonra hafızada integer değer gösteren p isminde bir pointer tanımlanmış olur. Bunu hafızda temsili olarak gösterecek olursak:



Yukarıdaki temsili resimde, ilk sütun adres değerlerini temsil etmektedir, buna göre hafızanın a101 ile a120 numaralı adresleri arasındaki bilgiler gösterilmiştir ( bu değerler hexadecimal olup temsili olarak yazılmıştır)

Kodda bulunan p ve a değerlerinin hafızda hangi bölüme atanacağı çalıştırma sırasında ( execution time ) belirlenir dolayısıyla tam olarak nerede bulunacağı bilinemez ancak p tanımlnadıktan sonra a107 adresine ve a tanımlandıktan sonra a116 adresinin ayrıldığını kabul edelim. Bu durumda p’nin değeri a107 numaralı adreste olacaktır.

Kodda a değerine 10 konulmuştur bu durumda a’nın değerinin durduğu a116 numaralı adreste sayısal olarak 10 yazacaktır.

Kodda a’nın taşıdığı adres değeri, p’nin içine atılmıştır. Bu durumda p’nin değeri a’nın adresi olacaktır.

Aşağıdaki örnek kodu çalıştırınız ve yorumlayınız:

#include

int main(){

int a=10;

int \*p;

p=&a;

printf(“%dn”,\*p); // p’nin gösterdiği yeri basar

printf(“%dn”,p); // p’nin değerini yani, p’nin gösterdiği yerin adresini basar

printf(“%dn”,a); // a’nın değerini basar

printf(“%dn”,&a); //a’nın adresini basar

printf(“%dn”,&p); //p’nin adresini basar

}

C dilinde her dizi bir pointer her pointer da doğal bir dizidir.

char str[80], \*p1;  
p1 = str;

Burada p1, str dizisinin – stringinin – ilk elamaninin adresinin degerini alir. Yani string adi, aslinda o stringin hafizadaki baslangiç adresini = stringin ilk karakterinin adresini tutmaktadir. str dizisinin 5. elemanina erismek için ise;

str[4] veya \*(p1+4)

ifadelerini kullaniriz. Her ikisinin de anlami aynidir.

C’de dizi elemanlarina 2 sekilde ulasilir: pointer kullanimi ile ve indis kullanimi ile. Indis kullanimi gelistirme ve anlama bakimindan bir kolaylik saglasa da, hiz önemli bir konu oldugundan C programcilari genelde dizi erisimini pointer kullanarak yaparlar. Asagida ayni isi yapan iki fonksiyon yeralmaktadir:

void putstr(char \*s)  
{  
int t;  
for (t=0; s[t]; ++t) putchar(s[t]);  
}  
void putstr(char \*s)  
{  
while (\*s) putchar(\*s++);  
}

C dilinde bir pointer’ı array gibi kullanmak için:

int \*p = (int \*) malloc ( sizeof(int)\*10);

şeklinde bir satır yazmak yeterlidir. Burada klasik dizi tanımından farksız bir işlem yapılmış, hafızada 10’luk bir alan açılmış ve tipi integer olarak tanımlanmıştır.

Pointer’ın farkı bu alana erişim şeklindedir. Yani dizi tanımlarında olduğu gibi köşeli parantezle (a[3] gibi) erişmek yerine pointer üzerinde işlem yapmak gereki ancak yöntem bir önceki örnekte olduğu gibi

printf(“%dn”,\*(p+4)) şeklinde işleyebilmektedir.