Genel Bakış

Bilişimsel düşünme, bilgisayar bilimlerindeki kavram ve fikirleri kullanarak problemleri çözmeye ve bu problemlere bir bilgisayarda çalıştırılabilmesi için çözümlerini ifade etmeye yönelik bir yaklaşımdır. Bilgisayar, modern toplumun her alanında -sadece yazılım geliştirme ve mühendislikte değil, iş hayatında, beşeri bilimlerde ve hatta günlük yaşamda- gitgide daha yaygın hale geldikçe, gerçek dünya problemlerini çözmek için bilişimsel düşünmenin nasıl kullanılacağını anlamak, 21. yüzyılın en önemli becerisi. Hesaplamalı düşünme dört temel üzerine kuruludur: ayrıştırma, örüntü tanıma, soyutlama ve algoritma tasarımı. Bu kurs size bilgi işlemsel düşünmenin dört sütununu tanıtır ve problem çözme sürecinin bir parçası olarak nasıl uygulanabileceklerini gösterir.

Bilişimsel Düşünmeye Giriş

Bilgisayarlar bir sorunu çözmek için kullanılmadan önce, sorunun kendisi ve nasıl çözülebileceği anlaşılmalıdır. Bilişimsel düşünme teknikleri bu görevlerde yardımcı olur.

Bilişimsel düşünme nedir? -1

Bu soruyu cevaplamak aslında oldukça zor. Bilişimsel düşünmenin (CT) savunucuları, çok yakın zamana kadar, onu nasıl tanımlayacaklarını tartışmak için çok zaman harcadılar.

BD aynı zamanda hem yeni hem de eski bir fikirdir. Konu 2006'da Wing'in konuşmasının ardından aniden hararetle tartışılan bir konu haline geldi (Wing, 2006). Bununla birlikte, temel fikirlerinin birçoğu onlarca yıldır tartışılıyor ve bu süreçte insanlar onları farklı şekillerde değerlendirdiler. Örneğin, daha 1980 yılında, Massachusetts Institute of Technology'den Seymour Papert, 'prosedürel düşünme' adını verdiği bir tekniğe öncülük etti (Papert, 1980). Şimdi CT olarak düşündüğümüz şeyle birçok fikir ortaktı. Prosedürel düşünmeyi kullanan Papert, öğrencilere bilgisayarları araç olarak kullanarak problem çözme yöntemini vermeyi amaçladı. Buradaki fikir, öğrencilerin bir bilgisayarın daha sonra gerçekleştirebileceği algoritmik çözümleri nasıl oluşturacaklarını öğrenecekleriydi; Bunun için Logo programlama dilini kullandı. Papert'in yazıları CT'ye çok ilham verdi, ancak CT bu orijinal fikirden bazı açılardan ayrıldı.

Bununla birlikte, Wing'in konuşmasını takip eden 10 yıl boyunca, bir dizi özlü tanım denendi. Aşağıda bunlardan bazılarını göreceksiniz.

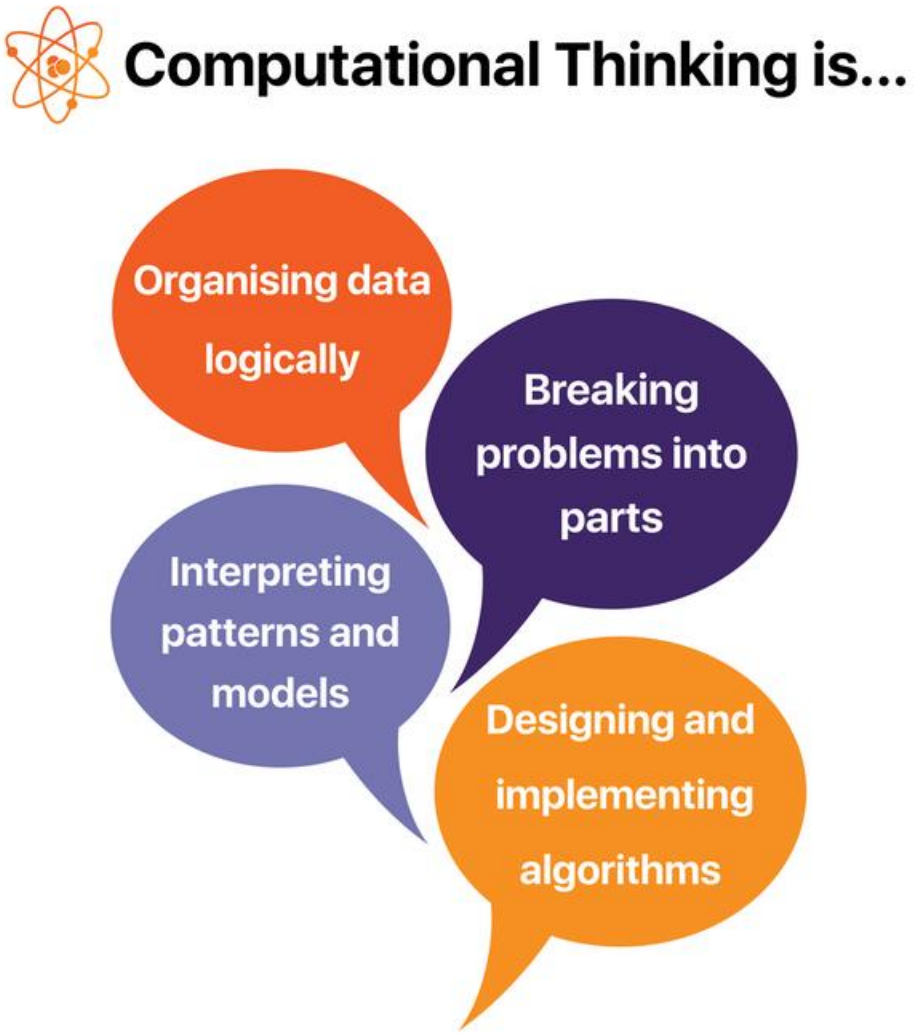
Bilişimsel düşünme, bir problemin formüle edilmesi ve çözüm(ler)inin bir bilgisayarın -insan veya makinenin- etkili bir şekilde gerçekleştirebileceği şekilde ifade edilmesiyle ilgili düşünce süreçleridir. (Kanat, 2014)

Problemleri soyutlamak ve otomatikleştirilebilecek çözümleri formüle etmek için zihinsel aktivite. (Yadav ve diğerleri, 2012)

Bizi çevreleyen dünyadaki hesaplamanın özelliklerini tanıma ve hem doğal hem de yapay sistemler ve süreçleri anlamak ve bunlar hakkında akıl yürütmek için Bilgisayar Bilimlerinden araç ve teknikleri uygulama süreci. (Ferber, 2012)

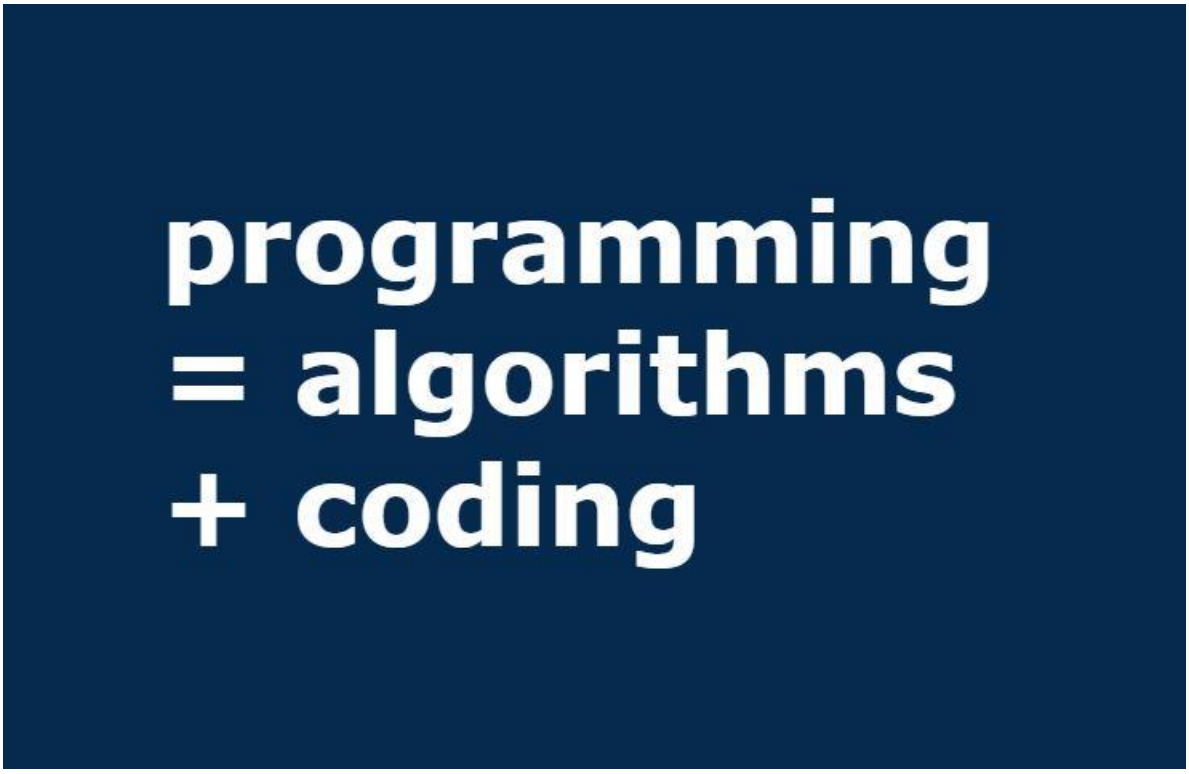
Problemleri bazı girdilerin çıktıya dönüştürülmesi olarak formüle etmeye ve dönüştürmeleri gerçekleştirmek için algoritmalar aramaya yönelik zihinsel bir yönelim. Bugün bu terim, birçok soyutlama düzeyiyle düşünmeyi, algoritmalar geliştirmek için matematiğin kullanımını ve bir çözümün farklı boyutlardaki problemler arasında ne kadar iyi ölçeklendiğini incelemeyi içerecek şekilde genişletildi. (Denning, 2009)

[CT'yi öğretmek] bir ekonomist, bir fizikçi, bir sanatçı gibi düşünmeyi ve onların problemlerini çözmek, yaratmak ve verimli bir şekilde keşfedilebilecek yeni sorular keşfetmek için hesaplamayı nasıl kullanacaklarını anlamaktır. (Hemendinger, 2010)



Bilişimsel düşünme nedir? -2

- Bilişimsel Düşünme, bir sorunla karşılaşmak ile bu soruna bir çözüm bulmak arasındaki zorunlu adımdır. Bilişimsel Düşünme, nihai amacın bir bilgisayara programlanmaya hazır olduğu anlamına gelen bir çözüm sağlamak olduğu bir problem çözme yaklaşımını öğretir.



- Programlama tuğla örmeye çok benzer: Bir şey inşa edebilmek için onun hakkında biraz bilgi sahibi olmanız gerekir ve usta zanaatkarlar, işaretleme ve balıksırtı düzenlerindeki varyasyonların göreceli avantajları hakkında uzun ve ayrıntılı konuşmalar yapabilir, ancak temelde duvar bir duvardır.

Architects or Bricklayers

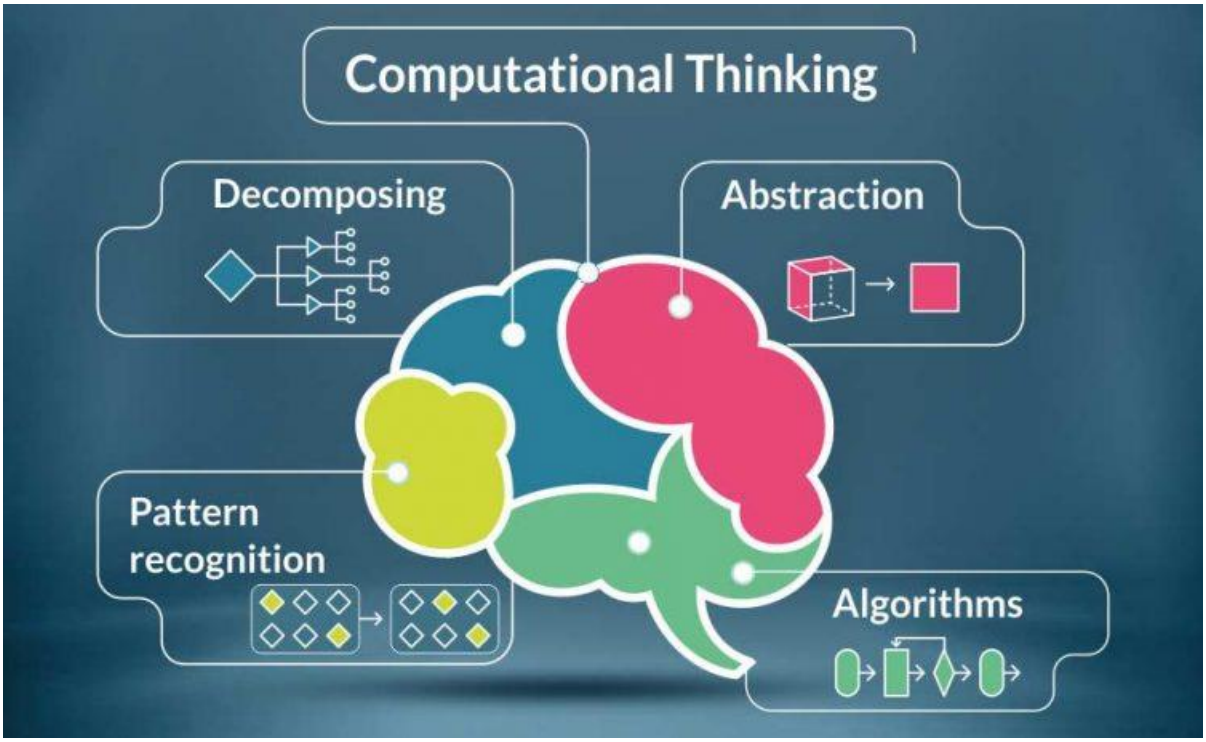
- Duvar örmek o kadar da ilginç değil: mimari ilginç. Mimarlığın insanların gereksinimlerini anlamak ve özellikle şekillendirilmiş bir tuğla yığınının onlara nasıl hitap edebileceğini görmekle ilgili olması gibi, bilişimsel düşünme de bir sorunu anlamak ve özellikle şekillendirilmiş bir program ifadeleri yığınının ona nasıl hitap edebileceğini görmekle ilgilidir.

Architects or Bricklayers

Pillars of Computational Thinking

Bilişimsel Düşünme dört bölümden oluşur:

1. **Ayrıştırma:** Verileri, süreçleri veya sorunları daha küçük, yönetilebilir parçalara ayırma
2. **Örüntü Tanıma:** Verilerdeki desenleri, eğilimleri ve düzenlilikleri gözlemleme
3. **Soyutlama:** Sadece önemli kısımlara odaklanmak, alakasız detayları görmezden gelmek
4. **Algoritma Tasarımı:** Bu ve benzeri problemlerin çözümü için adım adım yönergelerin geliştirilmesi



Aşağıdaki derslerde her adımın ayrıntılarını öğreneceksiniz.

Pratikte bilişimsel düşünme

Karmaşık bir problem, ilk bakışta nasıl kolayca çözeceğimizi veya anlayacağımızı bilmediğimiz bir problemdir.

CT, bu karmaşık sorunu alıp onu bir dizi küçük, daha yönetilebilir soruna (ayrışma) ayırmayı içerir. Daha sonra bu küçük problemlerin her birine, benzer problemlerin daha önce nasıl çözüldüğü dikkate alınarak (kalıp tanıma) ve alakasız bilgileri görmezden gelirken (soyutlama) sadece önemli ayrıntılara odaklanarak ayrı ayrı bakılabilir. Daha sonra, daha küçük problemlerin her birini çözmek için basit adımlar veya kurallar tasarlanabilir (algoritma tasarımı).

Son olarak, bu basit adımlar veya kurallar, bir bilgisayarı karmaşık sorunu en iyi şekilde çözmeye yardımcı olacak şekilde programlamak için kullanılır.

CT programlama değildir. Bilgisayarların düşünmediği ve düşünemeyeceği gibi, bilgisayar gibi düşünmek bile değildir.

Basitçe söylemek gerekirse, programlama bir bilgisayara ne yapacağını ve nasıl yapacağını söyler. Bilişimsel düşünme, bilgisayara tam olarak ne yapmasını söyleyeceğinizi bulmanızı sağlar.

Örneğin, daha önce hiç bulunmadığınız bir yerde arkadaşlarınızla buluşmayı kabul ederseniz, muhtemelen evinizden çıkmadan önce rotanızı planlarsınız. Mevcut rotaları ve hangi rotanın "en iyi" olduğunu düşünebilirsiniz - bu en kısa, en hızlı veya yolda en sevdiğiniz mağazanın yanından geçen rota olabilir. Daha sonra oraya ulaşmak için adım adım yönergeleri izlersiniz. Bu durumda, planlama kısmı CT gibidir ve yönergeleri takip etmek programlama gibidir.

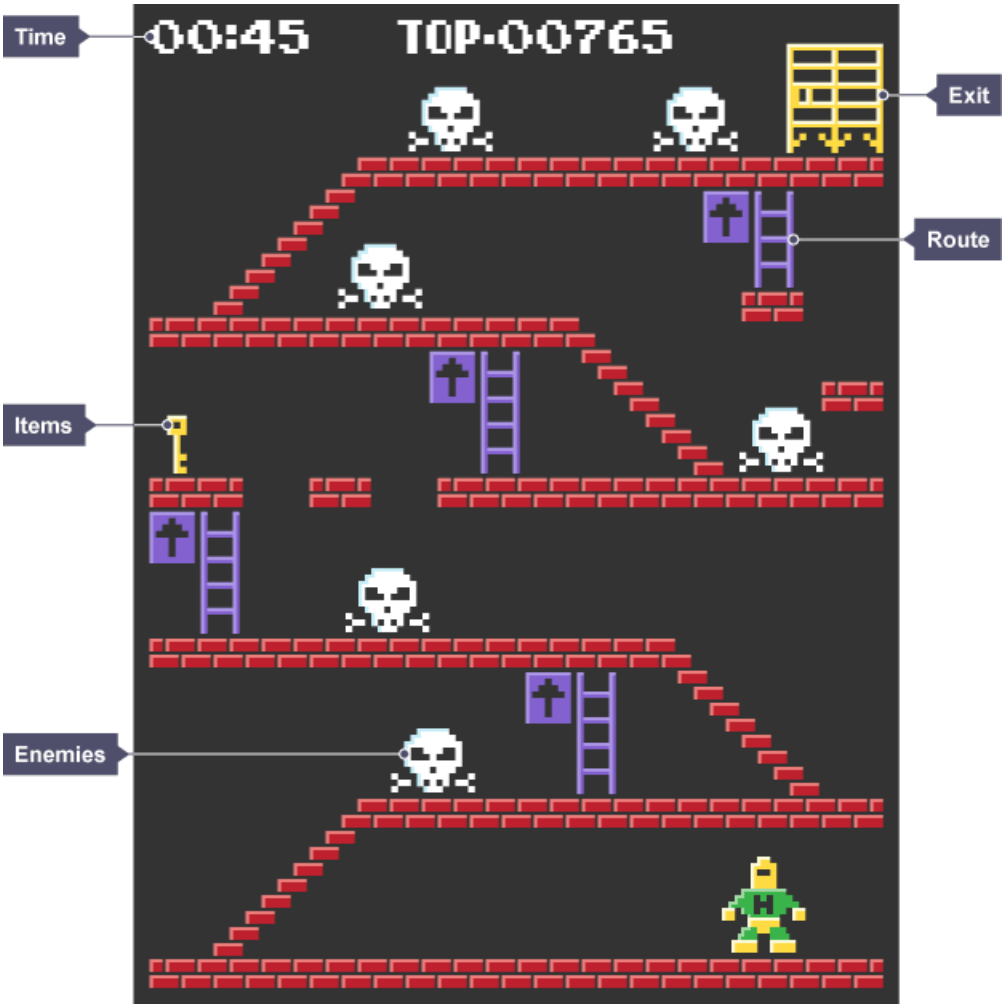
Karmaşık bir problemi kolayca anlayabileceğimiz bir problem haline getirebilmek son derece faydalı bir beceridir. Aslında, zaten sahip olduğunuz ve muhtemelen her gün kullandığınız bir beceridir.

Bir video oyunu oynarken başka bir örnek ortaya çıkabilir. Oyuna bağlı olarak, bir seviyeyi tamamlamak için şunları bilmeniz gerekir:

- hangi eşyaları toplamanız gerekiyor, bunları nasıl toplayabilirsiniz ve bunları toplamak için ne kadar süreniz var?
- çıkışın nerede olduğu ve ona en kısa sürede ulaşmak için en iyi rota
- ne tür düşmanlar var ve zayıf noktaları

Bu ayrıntılardan, seviyeyi en verimli şekilde tamamlamak için bir strateji geliştirebilirsiniz.

Kendi bilgisayar oyununuzu yaratacak olsaydınız, bunlar tam olarak oyununuzu programlamadan önce düşünmeniz ve cevaplamanız gereken soru türleridir.



Gerçek Hayatta Bilişimsel Düşünmeyi Kullanma Örnekleri

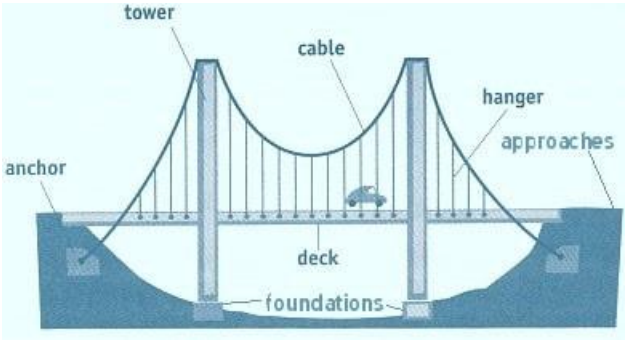
Mezuniyet töreni akışı

Dekan Randy Bryant, mezuniyet törenindeki diploma törenini nasıl hızlandıracağını düşünüyordu. Bireylerin nerede durması gerektiğini dikkatli bir şekilde yerleştirerek, verimli bir akış tasarladı, böylece her bir mezunun adının ve onur derecesinin Dekan Yardımcısı Mark Stehlik tarafından okunmasından sonra, her bir kişi diplomasını alabilir, ardından Mark'tan bir el sıkışabilir veya kucaklayabilir ve ardından onunla fotoğrafı çektirir. Bu tasarım, düzenli bir öğrenci akışının sahne boyunca yürümesine izin verdi (gerçi Mark'a sarılırken mezunun şapkası her düştüğünde akış duraklaması meydana geldi)

Depreme dayanıklı köprüler tasarlamak.

Bir Fen Bilgisi dersinde öğrenciler, depreme dayanıklı köprüler inşa etmek için CT yi, fizik ve mühendislik tasarımına uyguladılar. Ünite, köprülerin işlevini ve farklı türlerini anlamakla başladı. Öğrenciler daha sonra depremleri ve kuvvetlerinin etkisini incelemeye başladılar.

Depreme dayanıklı bir köprü tasarlamak için öğrenciler hem tasarım odaklı düşünmeyi hem de bilişimsel düşünmeyi uygulamışlardı. Bilişimsel düşünme, öğrencilerin yapılarındaki örüntüleri bulmak için çeşitli köprü modellerini analiz etmelerini (desen tanıma) ve bunlardan işlevsel bir tasarımda ihtiyaç duyulan önemli unsurları (soyutlama) soyutlamalarını sağladı. Farklı prototipleri test ederken, bilgi işlemsel düşünme, veri toplamalarına ve yapıyı iyileştirme fırsatları bulmalarına izin verdi.



Sızdıran boru

İşte gerçek hayattaki bir problemde bilgi işlemsel düşünme yaklaşımını kullanan başka bir örnek.

Anna Shipman'ın dairesini sel basmaya devam etti. Çeşitli tesisatçılar gelip gittiler ve bir şeyler yaptılar - bir sonraki sele kadar. Ama sonra, sanki önceki hayatında bir yazılım mühendisi veya programcı gibi biri çıktı. Sızıntının kaynağı olabilecek borulardaki ve oluklardaki her bir elemanı sistematik ve metodik olarak test etti. Başka bir deyişle, ayrıştırma ilkesini uyguluyordu.

Anna'nın vardığı sonuç ilginç: "Her şey biraz daha yazılım geliştirme gibi olsaydı daha iyi olurdu. Zanaatkarlar daha çok yazılım geliştiriciler gibi olmalı."

Alıştırma: En Kısa Yolu Bulma

Seyahat

Siz ve aileniz bölgedeki tüm başkentlerde bir yolculuğa çıkmayı planlıyorsunuz ve yakıt maliyetinden tasarruf etmek için mümkün olan en kısa rotayı bulma göreviniz var.

Şimdi birkaç seçenek üzerinde düşünme şansınız olduğuna göre, şimdi bir düşünün, önerdiğiniz rotayı ailenize nasıl açıklarsınız? Şehirlerin bir listesini verecek misiniz yoksa batıdan başlayıp doğuya doğru takip etmek gibi yol gösterici bir ilke var mı?

İşte dikkate alınması gereken bazı sorular:

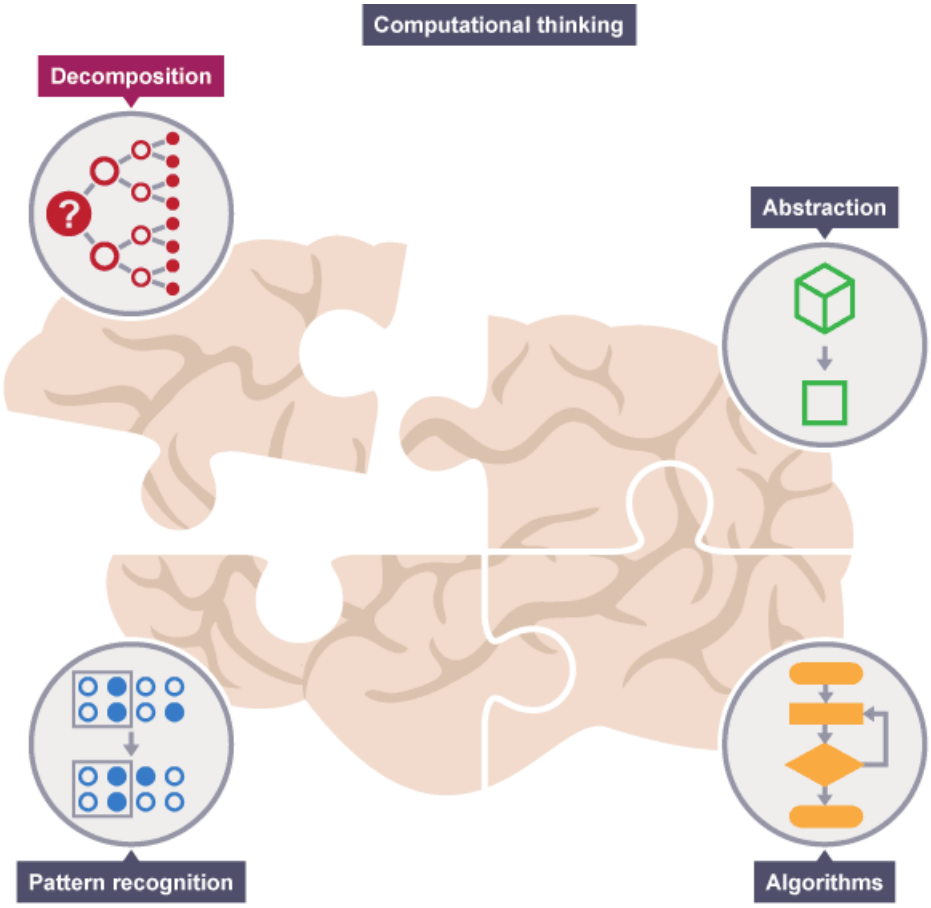
1. Hangi şehirden başladığınız önemli mi?
2. Rotanızı geliştirmek için kullandığınız strateji (boylamsal, enlemsel vb.) herhangi bir ülke için geçerli mi yoksa ülkeye özel mi? Başka bir ülkeye geçmeyi deneyin ve orada da işe yarayıp yaramadığını görün.
3. Bu sorunun zor olmasının bazı nedenleri nelerdir?

Bu tür bir problem geleneksel olarak gezgin satış elemanı problemi (the traveling salesperson problem) olarak bilinir, burada zorluk her şehirden sadece bir kez geçmek ve başlangıç şehre geri dönmektir. Bu tür bir problem topluluk tarafından optimal olarak çözülmemiş olsa da, problemi araştırmak ve onu denemek ve çözmek için algoritmalar geliştirmek (optimal olmayan bir şekilde bile olsa) diğer birçok ilgili problemin verimliliğini artırmak için kullanılmıştır. DNA ve web sayfaları gibi verilerde rota planlama ve arama, benzer iki tür problemdir.

Bilişimsel düşünmede ilk adım: Ayrıştırma

Dekompozisyon nedir?

Ayrıştırma, Bilgisayar Biliminin dört sütunundan biridir. Karmaşık bir sorunu veya sistemi daha yönetilebilir ve anlaşılması daha kolay olan daha küçük parçalara ayırmayı içerir. Daha küçük parçalar daha sonra incelenebilir ve çözülebilir veya üzerinde çalışmak daha kolay olduğu için ayrı ayrı tasarlanabilir.



Örneğin, bir oyun geliştiriyorsak, farklı insanlar farklı seviyeleri bağımsız olarak tasarlayabilir ve oluşturabilirler, ancak kilit hususlarda önceden anlaşmaya varılır. Orijinal görevin ayrıştırılması yoluyla, her bir parça daha sonra süreçte geliştirilebilir ve entegre edilebilir. Basit bir arcade seviyesi, bir karakterin gerçeğe yakın hareketi, arka planı kaydırma ve karakterlerin nasıl etkileşime girdiğine ilişkin kuralları belirleme gibi birkaç bölüme ayrılabilir.

Ayrıştırma neden önemlidir?

Ayrıştırma, sadece proje planlamasından daha fazlası için kullanılır. Belirlenemeyecek kadar büyük veya hantal olan karmaşık kavramları anlamada kullanışlıdır. Bir kavram ayrıştırıldığında, onu oluşturan parçaların incelenmesi, konuya yaklaşmak için yeni yollar sunabilir. Süreçleri optimize edebilir, tasarımları düzene sokabilir ve hatta tamamen yeni fikirler üretebilirsiniz.

Bir pasta düşünün. Birden fazla malzemeden oluşan tek bir gıda maddesidir: yumurta, un, şeker, tereyağı. İçerik listesi bir çeşit ayrıştırmadır. Bir pastanın parçalarını anlamak, pişirmenin temellerini anladıktan sonra tarifi değiştirmenize olanak tanır.

Bir problem ayrıştırılmamışsa, çözülmesi çok daha zordur. Aynı anda birçok farklı aşamayla uğraşmak, bir problemi birkaç küçük probleme bölüp her birini teker teker çözmekten çok daha zordur. Problemi daha küçük parçalara bölmek, her küçük problemin daha detaylı olarak incelenebileceği anlamına gelir.

Benzer şekilde, karmaşık bir sistemin nasıl çalıştığını anlamaya çalışmak, ayrıştırmayı kullanarak daha kolaydır. Örneğin, bir bisikletin nasıl çalıştığını anlamak, tüm bisiklet daha küçük parçalara ayrılırsa ve her parça nasıl çalıştığını daha ayrıntılı görmek için incelenirse daha kolaydır.

Bir uygulama oluşturmayı ayrıştırma

İlk uygulamanızı oluşturmak istediğinizi hayal edin. Bu karmaşık bir sorundur - dikkate alınması gereken pek çok şey vardır.

Soru

Bir uygulama oluşturma görevini nasıl ayrıştırırsınız?

Bu görevi ayrıştırmak için, bir dizi daha küçük sorunun cevabını bilmeniz gerekir:

- ne tür bir uygulama oluşturmak istiyorsun
- uygulamanız nasıl görünecek
- uygulamanızın hedef kitlesi kim
- grafikleriniz nasıl görünecek
- hangi sesi dahil edeceksiniz
- uygulamanızı oluşturmak için hangi yazılımı kullanacaksınız
- kullanıcının uygulamanızda nasıl gezineceği
- uygulamanızı nasıl test edeceksiniz
- uygulamanızı nerede satacaksınız

Bu liste, bir uygulama yaratmanın karmaşık problemini, şimdi çözülebilecek çok daha basit problemlere böldü. Ayrıca, uygulamanın farklı bölümlerinde size yardımcı olacak diğer kişileri de alabilirsiniz. Örneğin, grafikleri oluşturabilecek bir arkadaşınız olabilirken, bir başkası testçiniz olabilir.

Ayrıştırmaya Bir Örnek: Böl ve Fethet

Bu video, hayali ama ciddi bir problem kullanarak "böl ve yönet" fikrini tanıtıyor - bir çift kirli çorap, Noel Baba'nın vermek üzere olduğu 1024 hediyeden birine yanlışlıkla sarılmış ve çocukların kötü bir sürprizle karşılaşmaması için hangisinden kaçınacağını bulması gerekiyor. Aşağıdaki videoyu oynatmadan önce birkaç dakika düşünün ve bu soruna çözümünüzü koymaya çalışın.

Hikayedeki çözüm, test edilecek 1024 kutu olduğunda, çoraplar bulunana kadar hepsini açmak yerine, bir seferde bir yarısının ortadan kaldırılabileceğine ve sorunu tekrar tekrar yarıya indirmenin sorunu çok hızlı bir şekilde daralttığına işaret ediyor. bir kutu (sorunun boyutu 1024'te başlıyor, bir tartı ile 512 kutu var, ardından 256, 128, 64, 32, 16, 8, 4, 2 ve 1.) Bu fikir, hızlı bilgisayar algoritmaları tasarımında sıklıkla karşımıza çıkıyor.

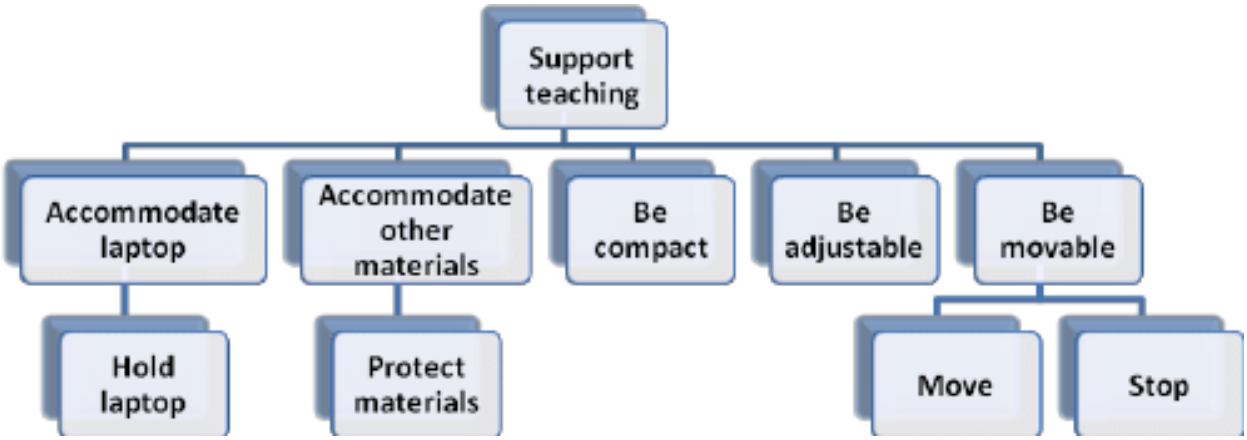
İşte size bir soru: Genç elfin çözümünden daha hızlı başka bir çözüm var mı?

(Cevap: Evet var. Çözmeye çalışın.)

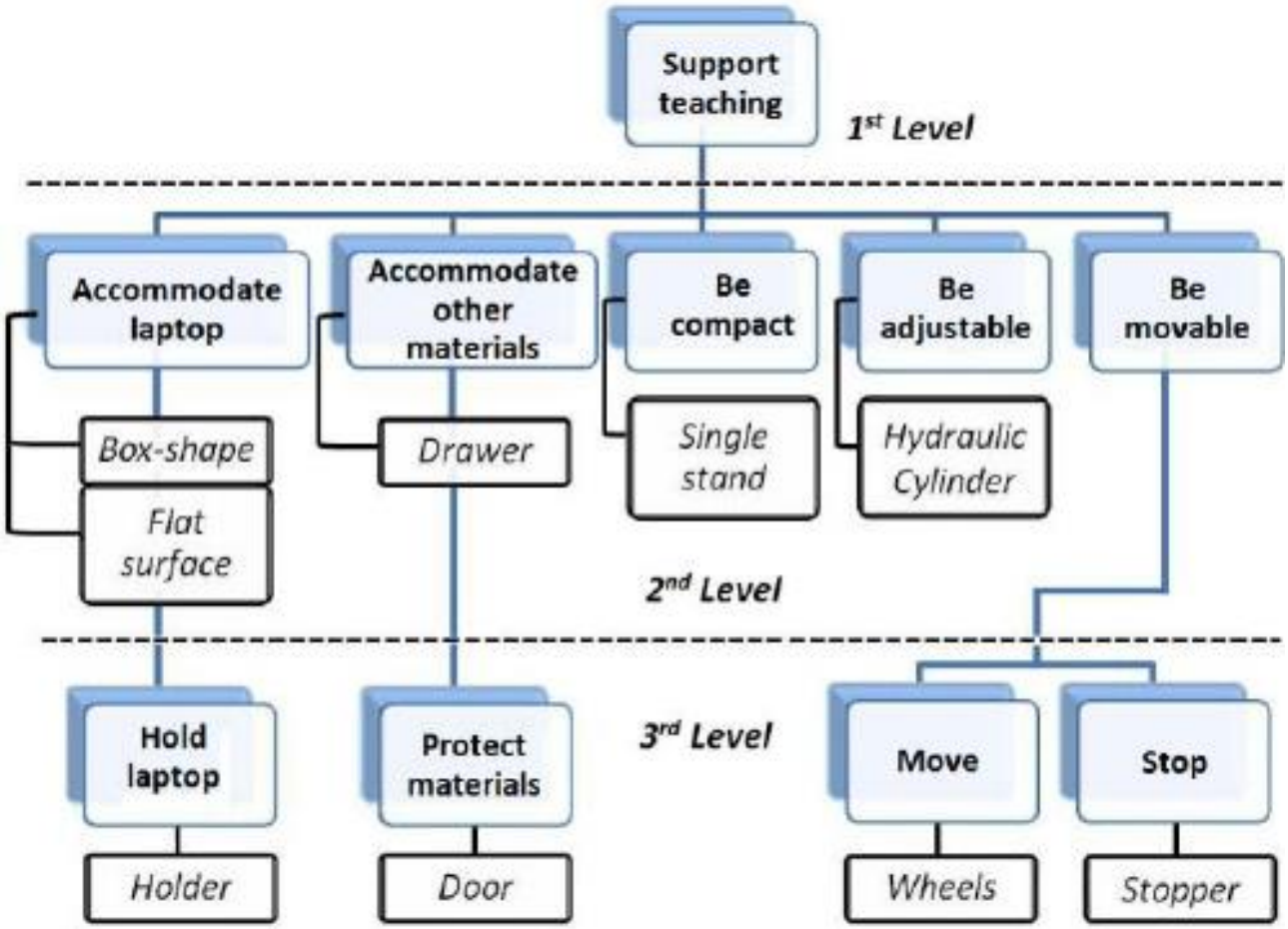
Böl ve yönet yaklaşımı, sorunları daha kolay (ve daha hızlı) çözmeyi sağladığından, neden daha fazla bölmeyelim. 1024 kutuyu iki yerine üç gruba (341-341-342) bölün. Aynı sayıda kutuya sahip iki grubu tartın. Daha sonra daha ağır olan grubu üçe bölün ve eğer ikisi de aynı ağırlıktaysa üçüncü grubu üçe bölün ve çorapları bulana kadar bu şekilde devam edin.

Ayrıştırma: Ağaç Yapıları

Problemin yapısı, dallarla birbirine bağlanan bloklardan oluşan bir fonksiyonel ağaçta fonksiyonlar ve alt fonksiyonlar arasındaki ilişkilerle temsil edilebilir (aşağıda gösterildiği gibi). Bir alt işlev, ana işlevinden daha düşük bir düzeydedir.

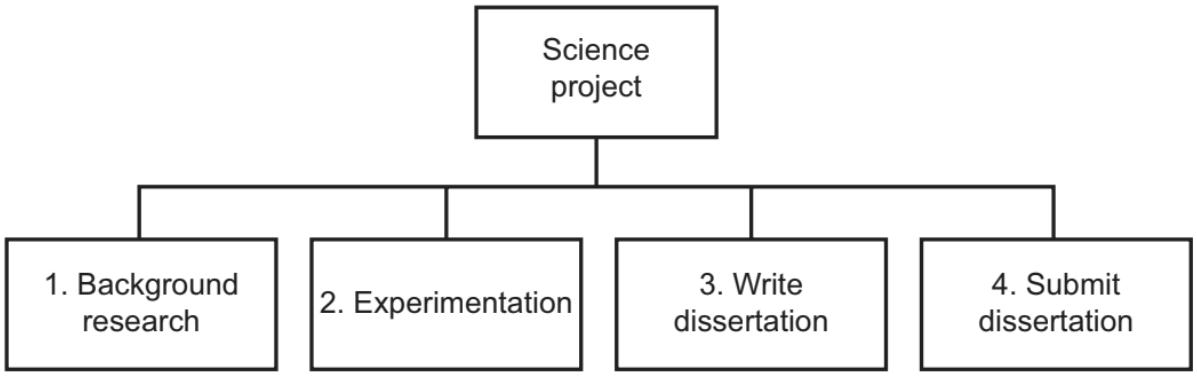


Aşağıdaki şekildeki ağaç yapılı fonksiyon ayrıştırması, problem yapısının ayrıştırılabilirlik ve soyutlama hiyerarşilerinin özelliklerini içerir. İlk olarak, ana fonksiyon birkaç alt fonksiyona ayrıştırılır. İkincisi, ana işlev, alt işlevler ve çözümler hiyerarşik bir yapı içinde birbirine bağlıdır. Bilgi türü (örneğin davranış, işlevler ve yapı) ve tasarım süreçlerinin aşamaları (problem yapılandırmasından kavram oluşturmaya kadar) soyutlama hiyerarşisine dahil edilir.

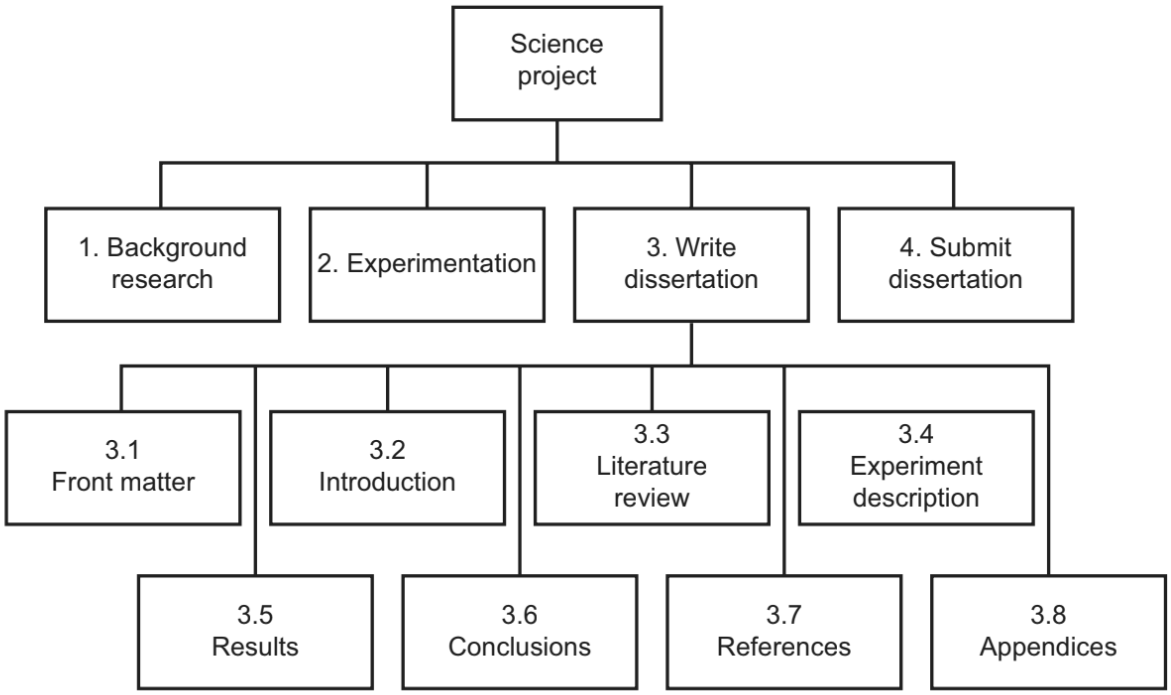


Ayrıştırma uygulayarak, bireysel olarak anlaşılabilecek ve çözülebilecek bir takım alt problemler elde etmeyi hedefliyorsunuz. Bu, işlemi tekrar tekrar uygulamanızı gerektirebilir (yukarıya bakın). Başka bir deyişle, problem, daha basit olsa da hala çok karmaşık olabilen bir dizi daha küçük problem olarak yeniden oluşturulur, bu durumda onların da parçalanması gerekir, vb. Görsel olarak bu, problem tanımına bir ağaç yapısı verir.

Bir örneğe bakalım.



Birçoğumuzun eğitimimiz sırasında karşılaştığı bir görev, bir tez hazırlamaktır. On binlerce kelime uzunluğunda bir akademik çalışma yazmak korku, paniğe ve hatta umutsuzluğa neden olabilir. Tek başına, "bir tez yazmak" başlıklı tek bir görev yönetilemez, ancak sorunu ayrıştırarak, her biri çok daha az ayrıntıya sahip olan daha küçük parçalara indirgersiniz. Daha sonra dikkatinizi her bir parçaya odaklayabilir ve onunla çok daha etkili bir şekilde ilgilenebilirsiniz. Her bir alt problemi çözerek, genel problemi çözmüş olursunuz.



Bu, görevi biraz ayrıştırdı, ancak bazı alt görevler hala çok büyük olabilir. Bu görevler, aynı ayrıştırma işlemi uygulanarak daha da bölünmelidir. Örneğin, bir tezin ön maddesi birkaç bölümden oluşur, örneğin:

3.1.1 bir başlık sayfası yazın;

3.1.2 telif hakkı bölümünü yazın;

3.1.3 Özet yazmak;

3.1.4 içindekileri yaz bölümü;

3.1.5 Şekiller bölümünün listesini yazar.

Bu sefer, ortaya çıkan tüm görevler artık yönetilebilir. Hiçbiri birkaç yüz kelimeden fazlasını içermeyecek ve hatta bazıları kelime işlemci tarafından otomatik olarak yapılabilir.

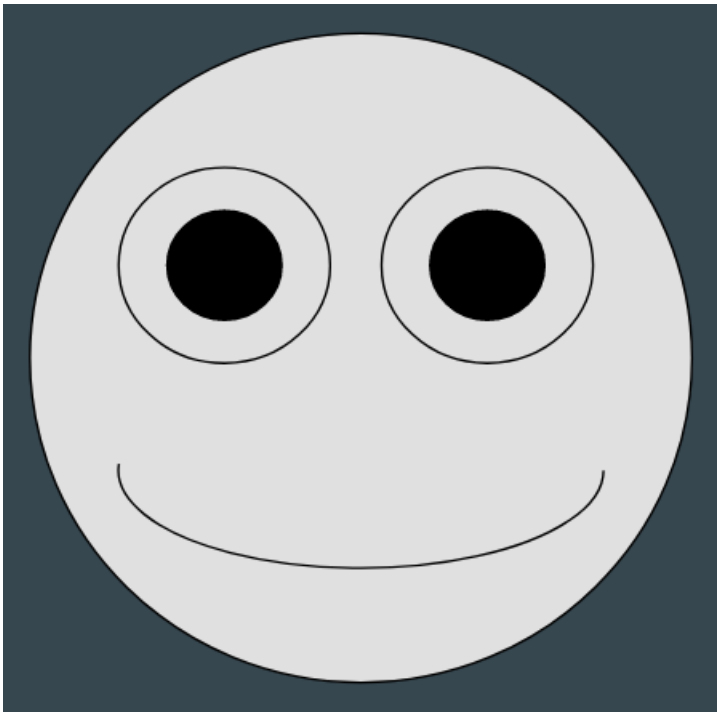
Bu noktada, ağaca geri dönebilir ve daha fazla ayrıştırma gerektiren farklı bir görev bulabilirsiniz. Ortaya çıkan ağacın tüm yaprakları tek tek çözebileceğiniz problemler olana kadar bu işlemin tekrarlanması gerekir.

Ayrıştırma: Gülemseyen Yüz Örneği

Bir bilgisayara karmaşık bir görüntü çizmesi talimatı vermek çok adımlı bir işlemdir, bu nedenle tek seferde çözülemez. Ancak, bir görüntüyü bir dizi basit şekle ayrıştırarak, sorunu ayrı ayrı çözülebilecek bir dizi basit alt soruna bölersiniz.

Basit bir örnek olarak: gülemseyen bir yüz. Bu tek bir görüntüdür, ancak aslında birkaç bileşen şekli içerir:

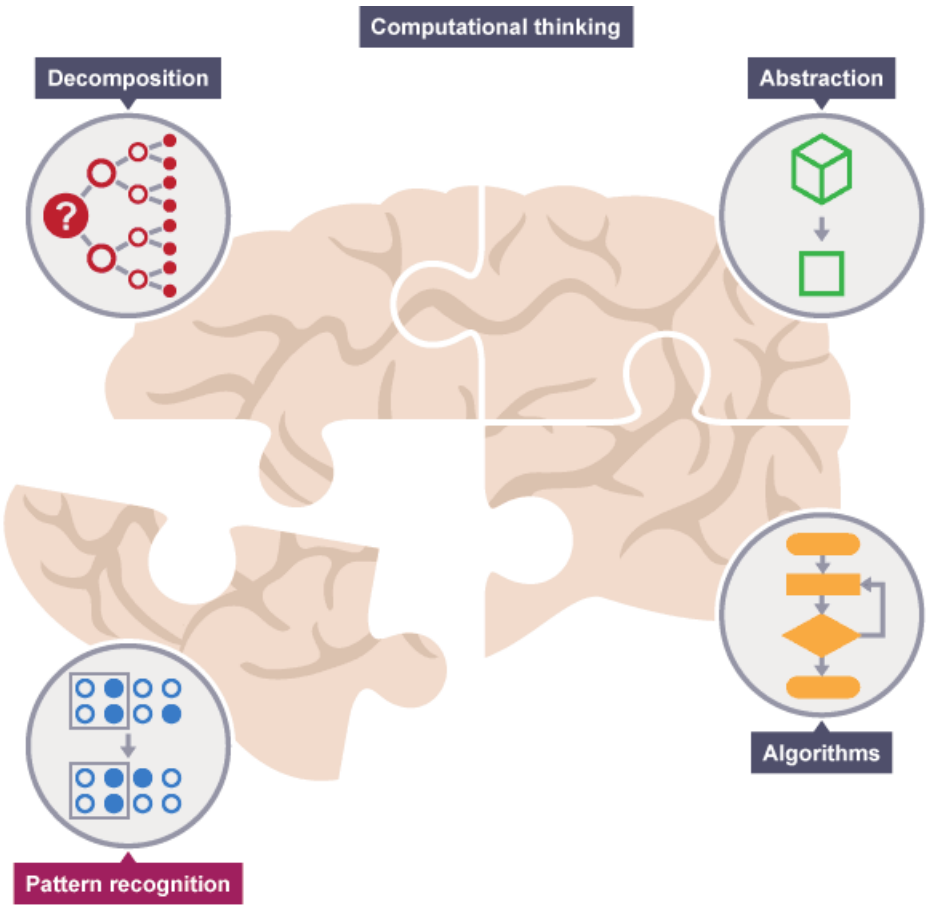
- Yüz için büyük bir daire.
- Dış göz için orta bir daire.
- İç göz için küçük, dolu bir daire.
- Gülümseme için bir yay.



Örüntü Tanıma nedir?

Karmaşık bir problemi ayrıştırdığımızda, genellikle oluşan daha küçük problemler arasında kalıplar buluruz. Kalıplar, bazı problemlerin benzerlikleri veya ortak özellikleridir.

Örüntü tanıma, CT'nin dört sütunundan biridir. Daha karmaşık problemleri daha verimli bir şekilde çözmemize yardımcı olabilecek küçük, ayrıştırılmış problemler arasındaki benzerlikleri veya kalıpları bulmayı içerir.



Desenler nelerdir?

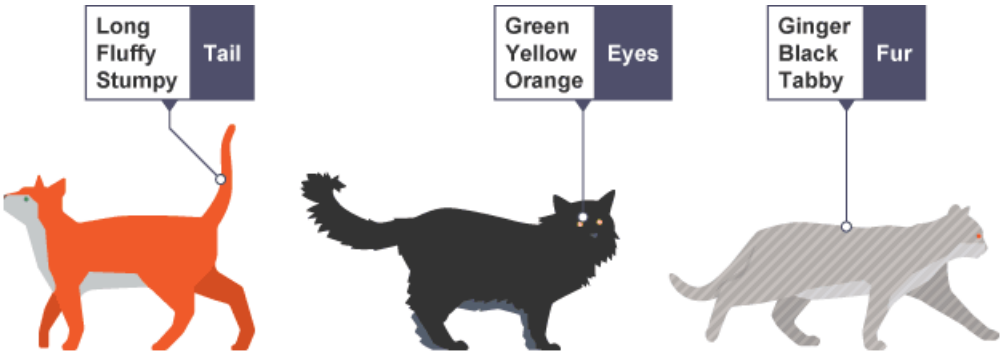
Bir dizi kedi çizmek istediğimizi hayal edin.

Tüm kedilerin ortak özellikleri vardır. Diğer şeylerin yanı sıra hepsinin gözleri, kuyrukları ve kürkleri var. Ayrıca balık yemeyi ve miyavlama sesleri çıkarmayı severler.

Tüm kedilerin gözleri, kuyrukları ve kürkleri olduğunu bildiğimiz için, sadece bu ortak özellikleri dahil ederek bir kedi çizmeye çalışabiliriz.

Bilişimsel düşünmede bu özellikler kalıplar olarak bilinir. Bir kediyi nasıl tanımlayacağımızı öğrendikten sonra, sadece bu kalıbı takip ederek diğerlerini de tanımlayabiliriz. Farklı olan tek şey özelliklerdir:

- bir kedinin yeşil gözleri, uzun kuyruğu ve siyah kürkü olabilir
- başka bir kedinin sarı gözleri, kısa kuyruğu ve çizgili kürkü olabilir



Neden kalıpları aramamız gerekiyor?

Kalıpları bulmak son derece önemlidir. Kalıplar görevlerimizi kolaylaştırır. Kalıplar ile ortak problemlerin çözülmesi daha kolaydır çünkü kalıbın olduğu her yerde aynı problem çözümünü kullanabiliriz.

Ne kadar çok model bulabilirsek, genel problem çözme görevimiz o kadar kolay ve hızlı olacaktır.

Birkaç kedi çizmek istiyorsak, kedileri genel olarak tanımlayacak bir desen bulmak, örneğin hepsinin gözleri, kuyruğu ve kürkü var, bu işi daha hızlı ve daha kolay hale getirir.

Tüm kedilerin bu modeli takip ettiğini biliyoruz, bu yüzden bunu çözmek için her yeni kedi çizmeye başladığımızda durmak zorunda değiliz. Kedilerin izlediğini bildiğimiz kalıplardan birkaç kediyi hızlıca çizebiliriz.

Kalıpları aramadığımızda ne olur?

Kedilerde kalıp aramadığımızı varsayalım. Ne zaman bir kedi çizmek istesek, durup bir kedinin neye benzediğini bulmamız gerekir. Bu bizi yavaşlatır.

Yine de kedilerimizi çizebilirdik - ve kediler gibi görünürlerdi - ama her kediyi çizmek çok daha uzun sürerdi. Bu çok verimsiz olurdu ve kedi çizme görevini çözmenin kötü bir yolu olurdu.

Ayrıca, desen aramazsak, tüm kedilerin gözleri, kuyrukları ve kürkleri olduğunu fark etmeyebiliriz. Kedilerimiz çizildiğinde kedi gibi görünmeyebilir bile. Bu durumda örüntüyü tanımadığımız için problemi yanlış çözmüş oluruz.

Kalıpları tanıma

Problemlerdeki kalıpları bulmak için her problemde aynı (veya çok benzer) olan şeyleri ararız. Sorunlar arasında ortak bir özelliğin olmadığı ortaya çıkabilir, ancak yine de bakmalıyız.

Kalıplar, farklı problemler arasında ve her problem içinde mevcuttur. İkisini de aramamız gerekiyor.

Farklı problemler arasındaki desenler

Problemler arasındaki örüntüleri bulmak için her problem için aynı (veya çok benzer) şeyleri ararız.

Örneğin, bir pasta pişirme görevini ayrıştırmak, bir dizi daha küçük problemin çözümlerini bilmemiz gerektiğini vurgulayacaktır:

- ne tür bir pasta pişirmek istiyoruz
- hangi malzemelere ihtiyacımız var ve her birinden ne kadar

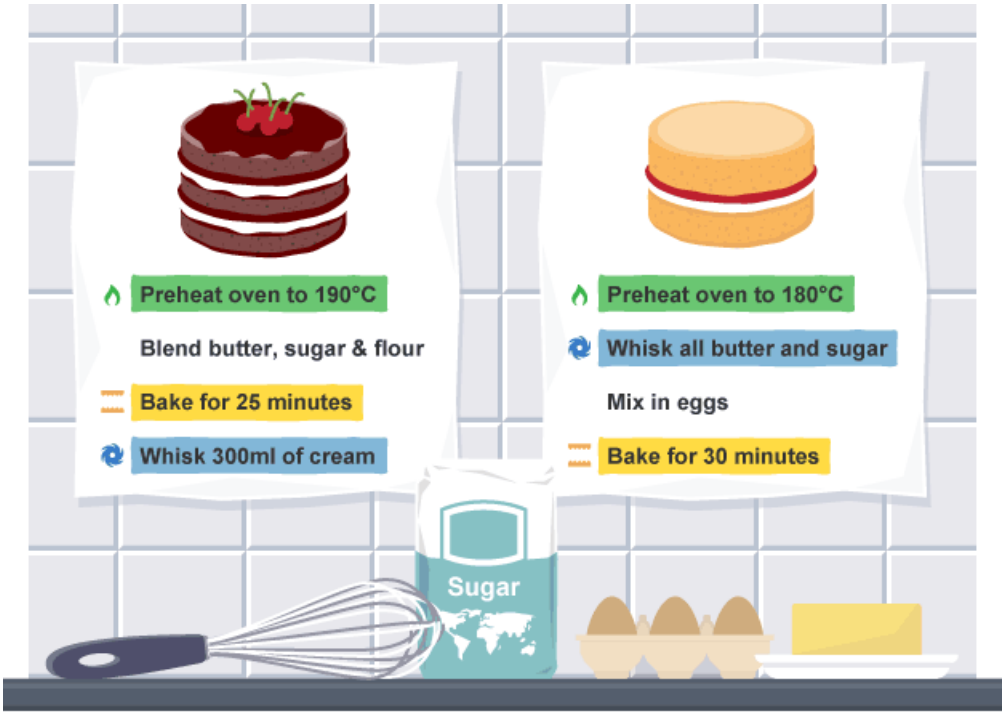
- pastayı kaç kişi için pişirmek istiyoruz
- pastayı ne kadar süre pişirmemiz gerekiyor
- her bir malzemeyi ne zaman eklememiz gerekir
- hangi ekipmana ihtiyacımız var

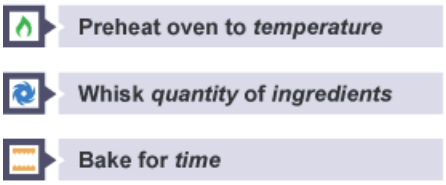
Belirli bir kek türünü nasıl pişireceğimizi öğrendiğimizde, başka bir tür kek pişirmenin o kadar da farklı olmadığını görebiliriz - çünkü kalıplar vardır.

Örneğin:

- her pastanın belirli bir miktarda belirli bileşenlere ihtiyacı olacaktır.
- malzemeler belirli bir zamanda eklenecek
- her kek belirli bir süre için pişecek

Kalıpları belirledikten sonra, problemler arasında ortak çözümler üzerinde çalışabiliriz.





Problemler içindeki desenler

Kalıplar, ayrıştırdığımız daha küçük problemlerde de mevcut olabilir.

Bir pasta pişirmeye bakarsak, daha küçük problemlerin içinde de kalıplar bulabiliriz. Örneğin, 'her pastanın belirli bir miktarda belirli malzemeye ihtiyacı olacaktır' Bunun için, her bir malzemenin aşağıdakilere ihtiyacı vardır:

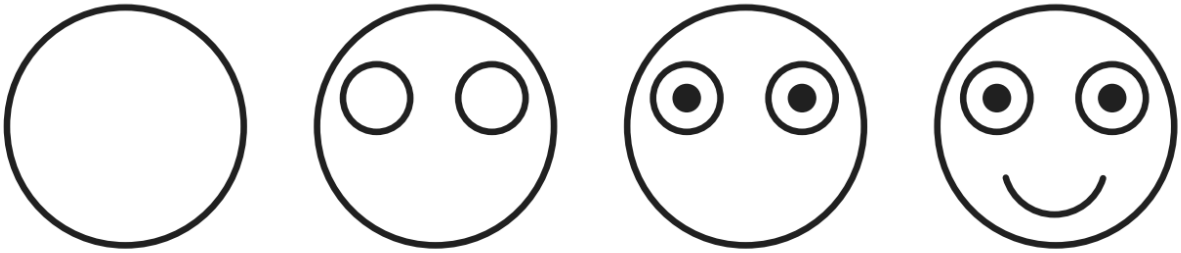
- tanımlama (adlandırma)
- belirli bir ölçüm

Her bir malzemeyi ve miktarını nasıl tanımlayacağımızı öğrendikten sonra, bu kalıbı tüm malzemelere uygulayabiliriz. Yine, değişen tek şey özelliklerdir.

Örnek: Gülümseyen Yüz Çizimindeki Kalıp.

Aşağıda gülümseyen yüz çizmek için bir desen verilmiştir.

1. 50,50 konumunda 30 yarıçaplı daireyi çizgi kalınlığı 2 ile çizin.
2. 40,40 konumunda yarıçapı 6 olan daireyi çizgi kalınlığı 1 ile çizin.
3. 40,40 konumunda siyahla doldurulmuş yarıçapı 3 olan bir daire çizin.
4. 60,40 konumunda yarıçapı 6 olan daireyi çizgi kalınlığı 1 ile çizin.
5. 60,40 konumunda siyah dolgulu 3 yarıçaplı daire çizin.
6. Çizgi kalınlığı 1 ile 30,70 konumundan 70,70 konumuna kırmızı bir çizgi çizin.



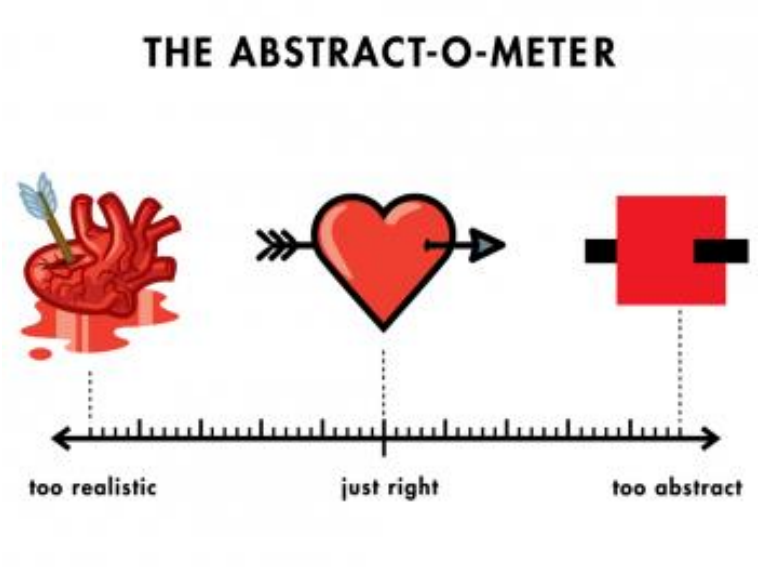
Bu örnekte, talimatlar arasındaki kalıpları tespit etmek oldukça basit olmalıdır. Bir kalıbın ortaya çıkması, iyileştirme için bir fırsat sağlar. Deseni oluşturan parçalar birbirinden koparmak ve ayırmak yerine genellikle bir araya getirilip ortak bir yaklaşımla çözülebilir. Başka bir deyişle, birbirinden ayrı ama benzer kavramları alıp tek bir kavramda genelleştirebilirsiniz. Sonuç olarak, daha az farklı kavram içerdiği için çözümünüz daha basit hale gelir ve başka durumlarda ve çözümlerde yeniden kullanabileceğiniz için daha güçlü hale gelir.

Soyutlama

Soyutlamanın Kısa Açıklaması

Soyutlama, bir fikri belirli bir bağlamda ifade etmenin ve aynı zamanda bu bağlamda alakasız ayrıntıları yok saymanın bir yoludur. Soyutlama, hem bilgisayar biliminin hem de CT önemli bir özelliğidir. Bazıları bilgisayar bilimini "soyutlamanın otomasyonu" olarak tanımlayacak kadar ileri gitti (Wing, 2014).

Bunun arkasındaki mantık, programcıların ve bilgisayar bilimcilerinin yapmaya çalıştıkları şeyin özüne kadar gider; yani, bilgisayarları kullanarak gerçek dünyadaki sorunları çözer. Gerçek dünyayı sihirli bir şekilde bilgisayara aktaramazlar; bunun yerine, gerçek dünyayı bilgisayara tanımlamaları gerekir. Ama gerçek dünya dağınık, bir sürü gürültü ve sonsuz ayrıntıyla dolu. Dünyayı bütünüyle tarif edemeyiz. Çok fazla bilgi var ve bazen nasıl çalıştığını tam olarak anlamıyoruz. Bunun yerine, gerçek dünyanın modellerini oluştururuz ve sonra bu modeller aracılığıyla problem hakkında akıl yürütürüz. Yeterli bir anlayış düzeyine ulaştığımızda, bilgisayara bu modellerin nasıl kullanılacağını öğretiriz (yani onu programlarız).



Bilişimsel düşünmede, problemleri ayrıştırdığımızda, daha sonra karmaşık problemi oluşturan daha küçük problemler arasında ve içinde örüntüler ararız.

Soyutlama, yaptığımız şeylere konsantre olmak için ihtiyaç duymadığımız kalıpların özelliklerini filtreleme – yok sayma – sürecidir. Aynı zamanda belirli detayların filtrelenmesidir. Bundan, çözmeye çalıştığımız şeyin bir temsilini (fikri) oluştururuz.

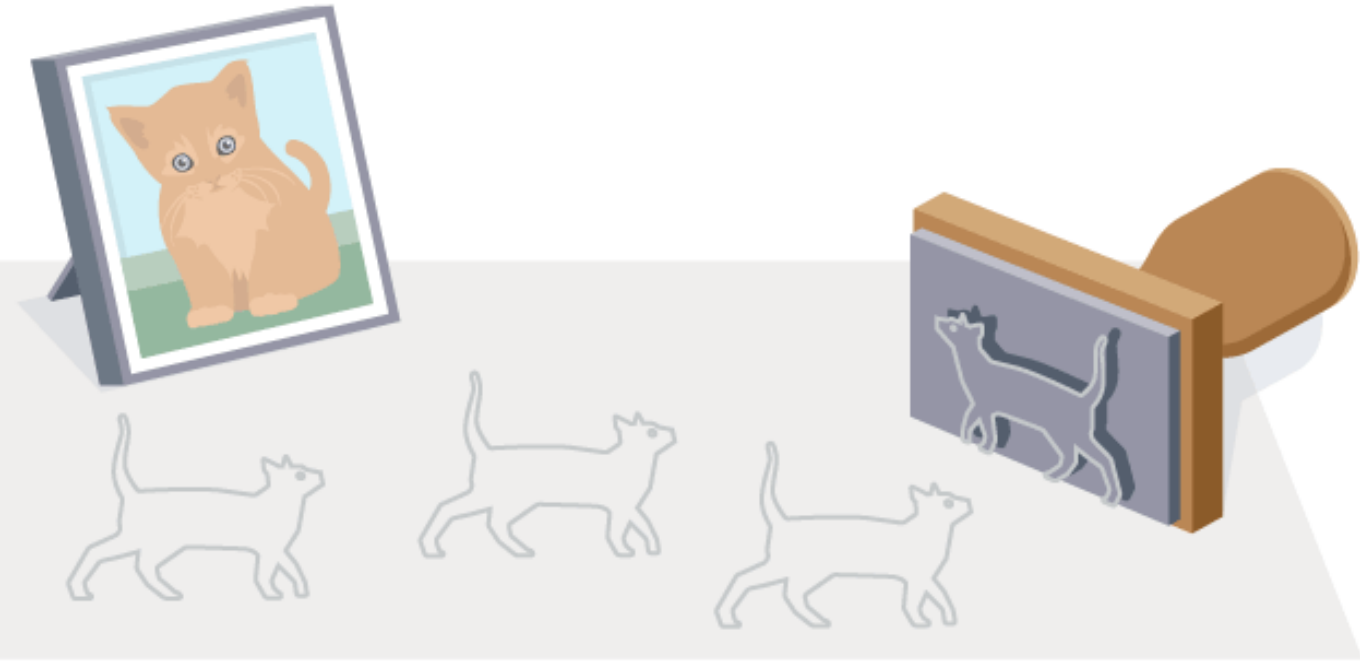
Belirli ayrıntılar veya özellikler nelerdir?

Örüntü tanımada bir dizi kedi çizme sorununa baktık.

Tüm kedilerin, örneğin gözler, kuyruk, kürk, balık sevme ve miyavlama sesleri çıkarma gibi tüm kedilerde ortak olan genel özelliklere sahip olduğunu belirtmiştik. Ayrıca her kedinin siyah kürk, uzun kuyruk, yeşil gözler, somon sevgisi ve yüksek sesle miyavlama gibi kendine has özellikleri vardır. Bu ayrıntılar, özellikler olarak bilinir.

Basit bir kedi çizmek için kuyruğu, kürkü ve gözleri olduğunu bilmemiz gerekir. Bir kedinin hangi sesi çıkardığını veya balıkları sevdiğini bilmemize gerek yok. Bu özellikler ilgisizdir ve filtrelenebilir. Bir kedinin kuyruğu, kürkü ve gözleri olduğunu bilmemiz gerekir, ancak bunların ne büyüklükte ve renkte olduğunu bilmemize gerek yoktur. Bu özellikler filtrelenebilir.

Sahip olduğumuz genel özelliklerden (kuyruk, kürk, gözler) bir kedi hakkında temel bir fikir oluşturabiliriz, yani bir kedinin temel olarak nasıl göründüğü. Bir kedinin neye benzediğini öğrendikten sonra, basit bir kedinin nasıl çizileceğini tanımlayabiliriz.



Soyutlama neden önemlidir?

Soyutlama, sorunun ne olduğu ve nasıl çözüleceği hakkında genel bir fikir oluşturmamızı sağlar. Süreç, sorunumuzu çözmemize yardımcı olmayacak tüm belirli ayrıntıları ve kalıpları kaldırmamızı ister. Bu, sorun hakkındaki fikrimizi oluşturmamıza yardımcı olur. Bu fikir bir 'model' olarak bilinir.

Soyutlama yapmazsak, çözmeye çalıştığımız soruna yanlış bir çözüm bulabiliriz. Kedi örneğimizde, eğer soyutlama yapmazsak, tüm kedilerin uzun kuyrukları ve kısa kürkleri olduğunu düşünebiliriz. Özetle, kedilerin kuyrukları ve kürkleri olmasına rağmen, tüm kuyrukların uzun olmadığını ve tüm kürklerin kısa olmadığını biliyoruz. Bu durumda soyutlama, daha net bir kedi modeli oluşturmamıza yardımcı olur.

Soyutlama Nasıl Yapılır

Soyutlama, ihtiyacımız olan genel özelliklerin toplanması ve ihtiyacımız olmayan detayların ve özelliklerin filtrelenmesidir.

Kek pişirirken kekler arasında bazı genel özellikler vardır.

Örneğin:

- bir pastanın malzemeye ihtiyacı var
- her bileşenin belirli bir miktara ihtiyacı vardır
- bir pastanın zamanlamaya ihtiyacı var

Soyutlama yaparken, belirli ayrıntıları kaldırır ve genel ilgili kalıpları koruruz.

Genel desenler Özel ayrıntılar

Bir pastanın malzemeleri olduğunu bilmeliyiz. Bu malzemelerin ne olduğunu bilmemize gerek yok.

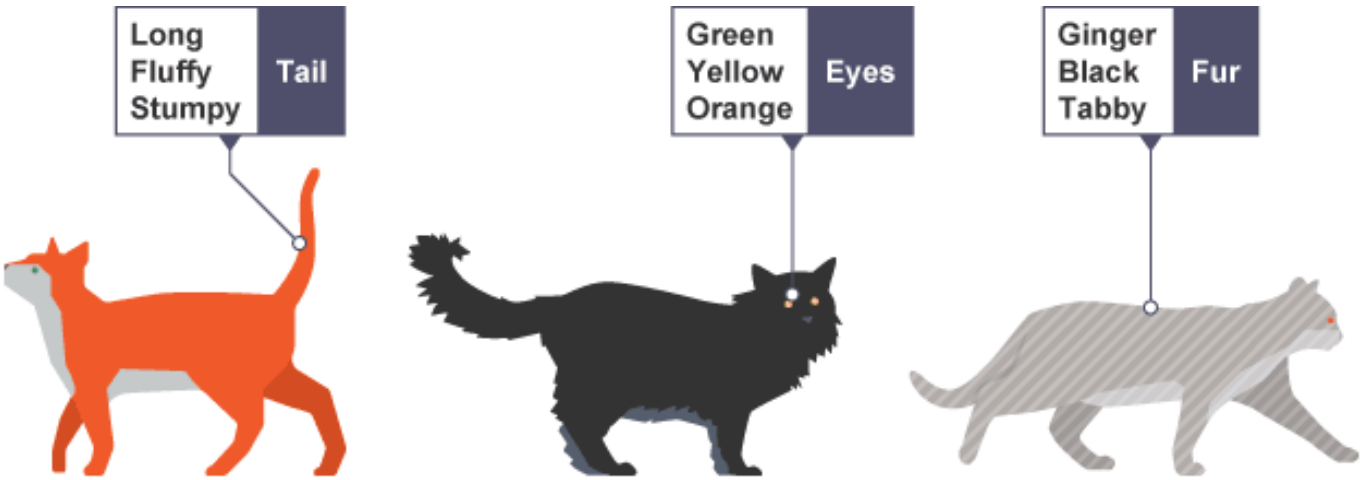
Her bir bileşenin belirli bir miktarı olduğunu bilmemiz gerekir. Bu miktarın ne olduğunu bilmemize gerek yok.

Her pastanın pişmesi için belirli bir süreye ihtiyacı olduğunu bilmeliyiz. Sürenin ne kadar olduğunu bilmemize gerek yok.

Model oluşturma

Model, çözmeye çalıştığımız problemin genel bir fikridir.

Örneğin, bir model kedi herhangi bir kedi olabilir. Uzun kuyruklu ve kısa kürklü belirli bir kedi değil - model tüm kedileri temsil ediyor. Kedi modelimizden, tüm kedilerin paylaştığı kalıpları kullanarak herhangi bir kedinin neye benzediğini öğrenebiliriz.



Benzer şekilde, kek pişirirken, model kek, sünger kek veya meyveli kek gibi belirli bir kek olmaz. Bunun yerine, model tüm kekleri temsil etmelidir. Bu modelden, tüm kekler için geçerli olan kalıpları kullanarak herhangi bir pastayı nasıl yapacağımızı öğrenebiliriz.

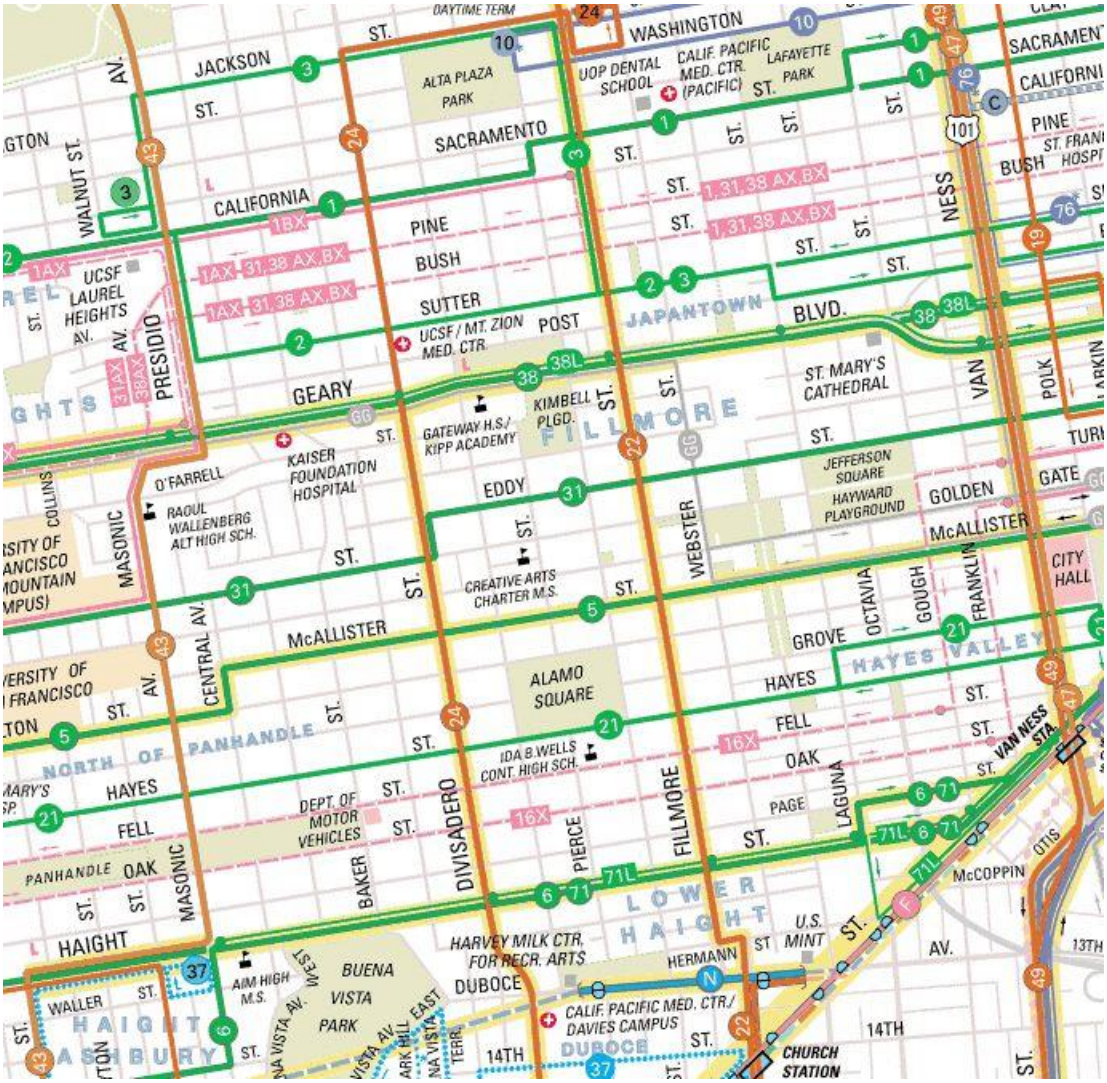
Problemimizin bir modeline sahip olduğumuzda, onu çözmek için bir algoritma tasarlayabiliriz.

Bir haritayı daha soyut bir versiyonuna dönüştürmek (Gerçekçi değil ama anlaşılması daha kolay)

Bir soyutlama örneği.

Transit haritalar coğrafi mi yoksa soyut mu olmalı?

Transit haritaların coğrafi olarak doğru olması gerektiğini söylemeye gerek yok. Birçok ajans, San Francisco Muni şehrinin ayrıntılı bir haritası üzerinde toplu taşıma hatlarını üst üste bindirilmiş olarak takip ediyor:



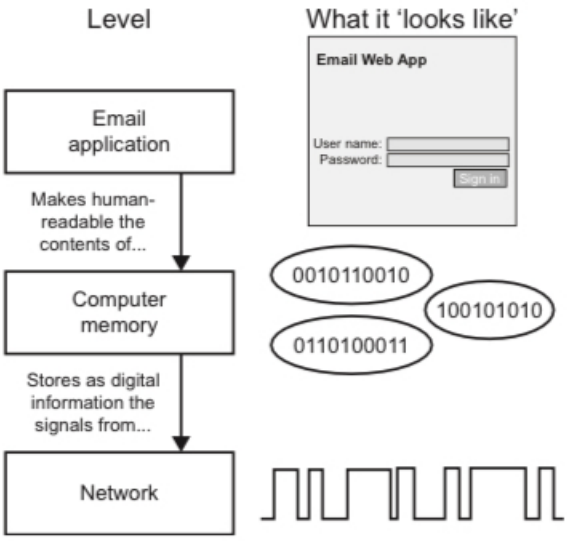
Ancak araştırmalar, ağ yapısını gerçekten görmemiz gerektiğini ve bunun bir dereceye kadar soyutlama gerektirdiğini öne sürüyor:

Araştırmacılar, New York ve Boston metro haritalarının alternatif versiyonlarını bilgisayar modeline koyarak, haritaların soyut versiyonlarının (coğrafi olarak doğru versiyonların aksine) tek bir bakışta kolayca anlaşılmasının daha muhtemel olduğunu gösterdiler.

Soyutlamaya Başka Bir Örnek: E-postalar

Bir bilgisayar bilimcisine soyutlamalar hakkında neyin bu kadar harika olduğunu sorarsanız, size daha teknik bir örnek verebilirler, örneğin bir e-posta. Dünyanın çoğu e-postanın ne olduğunu bilir: Bir bilgisayarda yazılan ve bir ağ aracılığıyla başka bir kullanıcıya iletilen bir mesaj. Bununla birlikte, bu bir soyutlamadır, aynı şekilde 'harf', üzerinde anlaşılır mürekkep işaretleri olan bir kağıt parçası için bir soyutlamadır. Bir e-postanın altında yatan çok daha fazla ayrıntı var. Bir e-posta hayal ediyorsanız, muhtemelen posta istemcinizdeki metni düşünürsünüz.

Ancak bir e-posta, bilgisayarınızın belleğinde birler ve sıfırlardan oluşan paketler olarak düzenlenmiş dijital bilgi olarak bulunur. Dahası, geçiş halindeyken, bir kablodan geçen bir dizi elektriksel akım veya atmosferde hızla yayılan elektromanyetik dalgalar olarak var olur. Bu soyutlama düzeyinde çalışsaydınız, arkadaşınıza, 'İnsan tarafından okunabilir orijinal mesajın kodunu çözmeniz için size elektriksel akımlar yoluyla kodlanmış birler ve sıfırlar dizisini İnternet üzerinden göndereceğim' derdiniz. Kafası karışmış arkadaşınız farkında olmadan, 'Neden onun yerine bana bir e-posta göndermiyorsunuz?



Algoritmanın Kısa Açıklaması.

Algoritma, net başlangıç ve bitiş noktaları ile sonlu bir belirsiz olmayan talimatlar kümesini takip etmek için bir süreci tanımlayan açıkça tanımlanmış adımlar dizisidir. Algoritmalar, çok adımlı bir görevi belirlemenin bir yoludur ve özellikle üçüncü bir tarafa (ister insan ister makine olsun) adımların aşırı hassasiyetle nasıl gerçekleştirileceğini açıklamak istediğimizde kullanışlıdır. Mantıkta olduğu gibi, insanlar zaten sezgisel bir algoritma anlayışına sahiptir. Ancak aynı zamanda zengin ve kesin bir bilim, algoritmaların tam olarak nasıl çalıştığını belirler. Bunu daha derinden anlamak, algoritmik düşüncenizi geliştirecektir. Bu önemlidir, çünkü doğru bir algoritma, herhangi bir bilgisayar tabanlı çözümün nihai temelidir.

Bir algoritmada, her talimat tanımlanır ve gerçekleştirilmeleri gereken sıra planlanır. Algoritmalar genellikle bir bilgisayar programı oluşturmak için bir başlangıç noktası olarak kullanılır ve bazen bir akış şeması (flowchart) veya sözde kod (pseudocode) olarak yazılırlar.

Bir bilgisayara bir şey yapmasını söylemek istiyorsak, bilgisayara adım adım tam olarak ne yapmasını ve nasıl yapmasını istediğimizi söyleyecek bir bilgisayar programı yazmalıyız. Bu adım adım programın planlamaya ihtiyacı olacak ve bunu yapmak için bir algoritma kullanıyoruz.

Bilgisayarlar ancak kendilerine verilen algoritmalar kadar iyidir. Bir bilgisayara zayıf bir algoritma verirseniz, kötü bir sonuç alırsınız - bu nedenle: 'Çöp içeri, çöp dışarı' ifadesi. (‘Garbage in, garbage out.’)

Algoritmalar, hesaplamalar, veri işleme ve otomasyon dahil olmak üzere birçok farklı şey için kullanılır.

Plan yapmak

Doğru olacağından emin olmak için bir sorunun çözümünü planlamak önemlidir. CT ve ayrıştırmayı kullanarak sorunu daha küçük parçalara ayırabiliriz ve sonra sorunu çözmek için bunların uygun bir sırayla nasıl bir araya geleceğini planlayabiliriz.

Bu sıra bir algoritma olarak temsil edilebilir. Bir algoritma açık olmalıdır. Bir başlangıç noktası, bir bitiş noktası ve aralarında bir dizi net talimat olmalıdır.