

Eğer bir non-None olmayan matris döndürülürse, o zaman bulmaca çözülmüş demektir ve çözüm yazdırılabilir. Bu, hiçbir zaman tree oluşturulmadığı bir örnektir, ancak arama alanı bir tree gibi şekillenir ve derinlik öncelikli arama, problem alanını aramak için kullanılabilir.



**6.7 Chapter Summary (Bölüm Özeti)**

Tree benzeri yapılar, Bilgisayar Bilimleri'ndeki birçok problemde karşımıza çıkar. Bir tree veri tipi, bilgi tutabilir ve hızlı bir şekilde ekleme, silme ve arama işlemleri yapılmasına olanak tanır. İkili arama trees pratikte kullanılmasa da, bunlara hakim olan ilkeler, B-trees, AVL-trees ve Splay trees gibi birçok ileri düzey veri yapısında kullanılmaktadır. Nesnelere referansların nasıl işaret ettiğini ve bunun bir tree gibi bir veri tipi oluşturmak için nasıl kullanılabileceğini anlamak, bilgisayar programcılarının kavraması gereken önemli bir konsepttir.

Arama alanları, genellikle birden fazla seçenek arasında karar vermek, başka bir karara yol açtığında tree benzeri bir yapı oluşturur. Bir arama alanı bir veri tipi değildir, bu yüzden bu durumda bir tree oluşturulmaz. Bununla birlikte, aranan alan tree benzeri bir yapıya sahiptir. Bir alanın derinlik öncelikli aramasını yapmanın anahtarı, nerede olduğunuzu hatırlamaktır, böylece bir seçim çıkmaza yol açarsa geri dönebilirsiniz. Geri izleme genellikle özyineleme (recursion) kullanılarak yapılır.

Tree ile ilgili birçok algoritma doğası gereği özyinelemeli (recursion). Derinlik öncelikli arama, tree gezintileri, ayrıştırma ve soyut sözdizimi değerlendirmesi hepsi özyinelemeli (recursion) olarak uygulanabilir. Özyineleme (recursion), problem çözme konusunda araç kutunuzda bulundurmanız gereken güçlü bir mekanizmadır.



**6.8 Rewiew Questions (Gözden Geçirme Soruları)**

Bu kısa cevap, çoktan seçmeli ve doğru/yanlış sorularını yanıtlayarak bölümü ne kadar iyi kavradığınızı test edin.

1. Bilgisayar Bilimleri'nde bir ağacın kökü ağaçta (tree) üstte mi yoksa altta mı yer alır?
2. Bir ağacın (tree) kaç kökü (roots) olabilir?
3. Tam bir ikili ağaç (binary tree), her seviyede tam olan bir ağaçtır (Tree), yani ağaçta (tree) herhangi bir seviyede yapraklar dışında başka bir düğüm için yer yoktur. Üç seviyeli tam bir ikili ağaçta kaç düğüm vardır? Peki ya 4 seviyeli? 5 seviyeli?
4. Tam bir ikili ağaçta (binary tree), ağacın yaprak sayısı ile toplam düğüm sayası arasında nasıl bir ilişki vardır?
5. Bir ağaç (tree) inşa ederken, ağaç yapısının alt-üst (bottom-up) mü yoksa üst-alt(top-down) bir şekilde yapılması daha kolaydır?
6. Bir ağaçta bir değeri ararken yanlış bir seçim yapıldığında, başka bir seçim yapılması gerektiğinde hangi terim kullanılır?
7. Bir arama alanı (search spaces) ile ağaç (tree) veri tipi arasındaki fark nedir?
8. Bir ağacın inorder gezintisini yapmak için özyinelemeli (recursion) olmayan bir algoritma açıklayın. İpucu: Bu işlemi gerçekleştirmek için bir yığına (stack) ihtiyacınız olacak.
9. 5 \* 4 + 3 \* 2 infiks ifadesi için bir ağaç oluşturmak için bazı kodlar yazın. Operatörlerin önceliğini ve ağacınızdaki yerlerini takip ettiğinizden emin olun. PlusNode ve TimesNode sınıflarının bölümden zaten tanımlandığını varsayabilirsiniz.
10. 5 \* 4 + 3 \* 2 ifadesinin prefix ve postfix formlarını verin.

**CEVAPLAR**

1. Kök en üstte bulunur ve diğer tüm düğümler ona bağlıdır.
2. Bir ağacın bir kökü olabilir.
3. Her seviyedeki düğüm sayısı, 2'nin o seviyedeki numaralı kuvvetinin 1 eksiğine eşittir.
4. Bir tam ikili ağacın yaprakları, her seviye dolu olduğunda ve her düğümün iki çocuğu olduğunda, yaprak sayısı, toplam düğüm sayısının yarısından bir fazla olur.
5. Bir ağaç inşa ederken, genellikle **üst-alt (top-down)** bir şekilde yapmak daha kolaydır.
6. Geri izleme (Backtracking), yanlış bir yol seçildiğinde, önceki adımlara geri dönüp doğru yola yönelmek için kullanılan bir tekniktir. Bu, özellikle derinlik öncelikli aramalarda yaygın olarak kullanılır.
7. Arama alanı, belirli bir problemi çözme sürecindeki tüm olasılıkları temsil eden soyut bir yapıdır, oysa ağaç veri tipi, verileri düzenlemek ve işlem yapmak için kullanılan somut bir veri yapısıdır.
8. Yığın kullanarak, ağacın tüm düğümleri gezilir. İnorder gezinti sırası, her düğümü sol alt ağaçtan başlayarak, kök düğümü ve sağ alt ağaç şeklinde işler.

Özyinelemeli olmayan bu yaklaşım, yığın yardımıyla her düğüm için hangi işlemin yapılacağını takip eder.

9. Pythondan yapıldı

10.  **Prefix (Önceden Operatör)**: + \* 5 4 \* 3 2

**Postfix (Sonra Operatör)**: 5 4 \* 3 2 \* +

Bu formasyonlar, operatörlerin önceliğini ve işlem sırasını doğru şekilde yansıtır.



* 1. **Programming Problems (Programlama Problemi)**

1. Kullanıcıdan bir prefix ifadesi girmesini isteyen bir program yazın. Ardından, program bu ifadenin infiks ve postfix formlarını yazdırmalıdır. Son olarak, ifadenin değerlendirilmesinin sonucunu yazdırmalıdır. Programla etkileşim aşağıdaki gibi olmalıdır.

metin, yazı tipi, makbuz, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Eğer prefix ifadesi hatalıysa, program, ifadenin hatalı olduğunu yazdırmalı ve çıkmalıdır. Bu durumda program, ifadenin infiks veya postfix formlarını yazdırmamalıdır.

1. Kullanıcıdan bir dizi sayı okuyan ve kullanıcının ağaçta değerleri eklemesine, silmesine ve aramasına olanak tanıyan bir program yazın. Program, ikili arama ağacına veri ekleme, arama yapma ve silme işlemleri için bir menü sunmalıdır. Ağaç içine veri ekleme işlemi, aşağıdaki gibi birden fazla eklemeye izin vermelidir.

metin, ekran görüntüsü, yazı tipi, doküman, belge içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

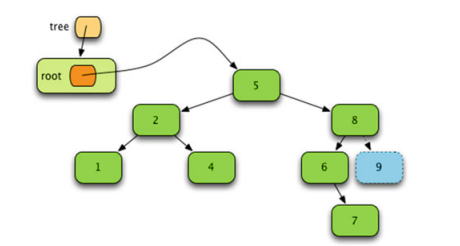
**6.9 Programming Problems (Programlama Problemleri) 181**

metin, ekran görüntüsü, yazı tipi, cebir içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Bu programın en zor kısmı, ağaçtan (tree) silme işlemidir. Bir değeri silmek için özyinelemeli (recursion) bir fonksiyon yazabilirsiniz. Bazı açılardan, ağaçtan (tree) silme fonksiyonu, bölümde verilen ekleme fonksiyonu gibi çalışır. İki fonksiyon yazmanız gerekecek: biri, bir ikili arama ağacında (tree) bir değeri silmek için çağrılacak bir yöntem, diğeri ise gizli bir özyinelemeli "ağaçtan silme" fonksiyonu olacaktır. Özyinelemeli (recursion) fonksiyon, bir ağaç (tree) ve silinecek bir değer almalıdır. Değeri ağaçtan sildikten sonra, ağacı geri döndürmelidir. Özyinelemeli (recursion) silme fonksiyonu üç farklı durumla ele alınmalıdır.

• Durum 1. Silinecek değer, child olmayan bir düğümde bulunmaktadır. Bu durumda, özyinelemeli fonksiyon, değeri ağaçtan sildikten sonra boş bir ağaç (yani None) döndürebilir. Bu, 9'un Şekil 6.12'deki ikili arama ağacından silindiği durum olur. Şekil 6.13'te, 8'i içeren düğümün sağ alt ağacı artık None'dur ve bu nedenle 9'u içeren düğüm ağaçtan silinmiştir.

• Durum 2. Silinecek değer, bir child olan bir düğümde bulunmaktadır. Bu durumda, özyinelemeli fonksiyon, değeri sildikten sonra child tree olarak döndürebilir. Bu, Şekil 6.13'teki ağaçtan 6'yı silme durumu için geçerli olur. Bu durumda, 6'yı silmek için

**Şekil 6.13**: 9 Silindikten Sonra Ağaç (tree)

Ağaçtan 6'yı içeren düğümü silmek için, basitçe 7'yi içeren düğümü döndürürsünüz, böylece 7, 8'i içeren düğüme bağlanmış olur. Şekil 6.14'te, 6'yı içeren düğüm, 8'i içeren düğümün sol alt ağacının, 6'yı içeren düğümün sağ alt ağacına işaret etmesiyle ortadan kaldırılmıştır

• Durum 3. Bu, uygulaması en zor olan durumdur. Silinecek değer, iki çocuğu olan bir düğümde bulunduğunda, düğümü silmek için başka bir fonksiyon kullanmak isteriz, buna getRightMost diyelim, bu fonksiyon ağacın sağdaki en uç değerini alır. Ardından bu fonksiyonu, silinecek düğümün sol alt ağacındaki sağdaki en uç değeri almak için kullanırsınız. Düğümü silmek yerine, düğümün değerini sol alt ağacın sağdaki en uç değeriyle değiştirirsiniz. Sonrasında, sol alt ağacın sağdaki en uç değerini sol alt ağaçtan silersiniz. Şekil 6.15'te, 5, sol alt ağacın sağdaki en uç değeri olan 4 ile değiştirilerek ortadan kaldırılır. Ardından, 4 sol alt ağaçtan silinir. 

diyagram, kırpıntı çizim içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Şekil 6.15**: 5 Silindikten Sonra Tree

1. **6.9 Programming Problems (Programlama Provlemleri) 183**Bölüm 5'te açıklanan şekilde Sudoku programını tamamlayın ve bunu, Bölüm 6.6.2'deki derinlik öncelikli arama (depth-first search) ile genişletin. Bu, her Sudoku bulmacasını çözebilen bir Sudoku programı oluşturacaktır. Bu program bulmacaları neredeyse anında çözmelidir. Eğer bir bulmaca uzun süre çözülüyorsa, bu büyük ihtimalle reduce fonksiyonunuzun, Bölüm 5'te açıklandığı şekilde bulmacayı yeterince azaltmadığından kaynaklanıyordur. Bu egzersizi tamamlamak için iki fonksiyona ihtiyacınız olacak: solutionOK fonksiyonu ve solutionViable fonksiyonu. solutionViable fonksiyonu, bölümde verilmiştir ve matrisin içindeki hiçbir küme boş değilse True döner. solutionOK fonksiyonu, çözümün geçerli bir çözüm olup olmadığını kontrol eder. Bu çok kolay bir şekilde kontrol edilebilir. Eğer matrisin içindeki kümelerden herhangi biri tam olarak 1 öğe içermiyorsa, çözüm geçerli değildir ve False döndürülmelidir. Eğer bir Sudoku bulmacasındaki herhangi bir grubun birleşimi 9 öğe içermiyorsa, çözüm geçerli değildir ve False döndürülmelidir. Aksi takdirde, çözüm geçerlidir ve True döndürülmelidir. Bu programı tamamladıktan sonra, sudoku7.txt veya sudoku8.txt gibi dosyaları çözebilmelisiniz; bu dosyalar, metnin web sitesinden indirilebilir.
2. Ortalama durumda O(log n) zaman karmaşıklığıyla öğe ekleyebilen, öğe silebilen ve öğe arayabilen bir OrderedTreeSet sınıfı tasarlayın. Bu sınıf üzerinde set içerik kontrolü için in operatörünü de implement edin. Ayrıca, set'teki öğeleri artan sırayla döndüren bir iterator implement edin. Bu set'in tasarımı, lt operatörünü implement eden her türlü öğenin set'e eklenmesine izin vermelidir. Bu OrderedTreeSet sınıfı, orderedtreeset.py adlı bir dosyada yazılmalıdır. Bu modülün ana fonksiyonu, OrderedTreeSet sınıfınızı kapsamlı bir şekilde test eden bir test programı içermelidir. Ana fonksiyon, modülün import edilip edilmediğini veya doğrudan çalıştırıldığını ayırt etmek için standart if ifadesi kullanılarak çağrılmalıdır.
3. OrderedTreeSet sınıfını kullanan bir OrderedTreeMap sınıfı tasarlayın. Bunu doğru şekilde organize etmek için iki modül oluşturmalısınız: orderedtreeset.py ve orderedtreemap.py modülleri. OrderedTreeMap sınıfının, HashSet ve HashMap'in Bölüm 5'teki gibi implement edildiği şekilde, OrderedTreeSet sınıfını kullanacak şekilde tasarlanması gerekmektedir. OrderedTreeMap sınıfınızı kapsamlı bir şekilde test eden test senaryoları tasarlayın.