ÜYELİK YAPILARI

5. Bölümde ekleme, silme, üyelik testi ve yinelemeyi destekleyen veri yapılarını ele aldık. Bazı uygulamalar için yalnızca üyelik testi yeterli olabilir. Yineleme ve silme işlemleri gerekli olmayabilir. Bunun klasik bir örneği yazım denetleyicisidir.

Bir yazım denetleyicisinin işleyişini düşünelim. Basit bir yazım denetleyicisi sadece yazım hatalarını tespit edebilirken, daha gelişmiş bir yazım denetleyicisi doğru yazılmış kelime alternatifleri de önerebilir.

Açıkça görülüyor ki, bir yazım denetleyicisi geniş bir kelime sözlüğü ile çalışır. Yazım denetleyicisi, kelime listesini kullanarak elinizdeki kelimenin sözlükte olup olmadığını belirler ve böylece doğru bir kelime olup olmadığına karar verir. Eğer kelime sözlükte yer almıyorsa, kelime işlemci veya metin düzenleyici, kelimenin yanlış yazılmış olabileceğini göstermek için altını çizebilir.

Bazı durumlarda, kelime işlemci alternatif, doğru yazılmış bir kelime önerebilir. Bazı durumlarda ise yanlış yazımı doğrudan düzeltebilir. Peki, bu yazım denetleyicileri/düzelticileri nasıl çalışır? Hangi veri yapılarını kullanırlar?

8.1 Bölümün Hedefi

İlk bakışta, bir **hash set** (yani bir **Python sözlüğü**) yazım denetleme için uygun bir veri yapısı gibi görünebilir. Çünkü bir öğeyi hash set içinde aramak **O(1)** zaman karmaşıklığıyla gerçekleştirilebilir. Ancak, burada bir **değiş tokuş** söz konusudur: Bu hash tablosunun boyutu oldukça büyük olabilir.

Tipik bir İngilizce sözlüğü 100.000’den fazla kelime içerebilir. Bu kadar çok kelimeyi saklamak için gereken alan oldukça büyük olacaktır.

Bu bölümde, bir küme içinde üyelik testi yapmak için tasarlanmış iki veri yapısını ele alacağız:

* **Bloom filtresi**, çok daha az yer kaplar ve **hızlı üyelik testi** sağlar.
* **Trie** (okunuşu "try") veri yapısı ise hash set ile elde edilemeyen bazı ek özelliklere sahiptir ve hash setten **daha az yer kaplayabilir**.

8.2 Bloom Filtreleri

Bloom filtreleri, adını yaratıcısı **Burton Howard Bloom**’dan alır ve ilk olarak 1970 yılında önerilmiştir. O zamandan beri, Alan Tharp [7] dahil birçok yazar Bloom filtrelerinin uygulamalarını ele almıştır. Wikipedia her zaman en yetkili kaynak olmasa da, Bloom filtreleri hakkında oldukça iyi bir tartışma sunmaktadır [8].

Bir Bloom filtresi, hash setlerle bazı fikirleri paylaşırken, çok daha az alan kullanır. Bloom filtresi, bir öğenin bir değer kümesine ait olup olmadığını belirlemek için **istatistiksel olasılık** kullanan bir veri yapısıdır. Bloom filtreleri **%100 doğru değildir**.

* Bloom filtresi, hiçbir zaman yanlış negatif sonuç vermez. Yani, bir öğe gerçekten kümenin içindeyse, asla dışında olduğunu bildirmez.
* Ancak, bazen yanlış pozitif sonuç verebilir. Yani, bir öğe kümede olmadığı hâlde varmış gibi görünebilir.

**Yazım Denetimi Örneği**

Bir yazım denetleyicisinin, girilen kelimenin doğru olup olmadığını belirlemek için sözlüğe bakması gerekir. Bloom filtresi, girilen kelimenin doğru olup olmadığını belirlemek için kullanılabilir. Ancak, bazen yanlış bir kelimenin doğru olarak algılanması mümkündür.

**Bloom Filtresi Yapısı**

Bir **Bloom filtresi**, **bit dizisi** ve **bir dizi hash fonksiyonundan** oluşur.

* Bloom filtresindeki bit sayısı ve kullanılan hash fonksiyonlarının sayısı, doğruluk oranını etkiler.
* Bu sayıların tam olarak ne olması gerektiği ilerleyen bölümlerde tartışılacaktır.

Örneğin, **20 bitlik** bir Bloom filtresi ve **3 bağımsız hash fonksiyonu** düşünelim. Başlangıçta, filtredeki tüm bitler **0** olarak ayarlanır (**Şekil 8.1**).

**Örnek: Kelime Ekleme**

**1. "cow" kelimesinin eklenmesi**

* Üç bağımsız hash fonksiyonu, "cow" kelimesini **mod 20** işlemi ile şu indekslere eşler: **18, 9 ve 3**
* Bu bitler **1** olarak değiştirilir (**Şekil 8.2**).

**2. "cat" kelimesinin eklenmesi**

* "cat" için hash fonksiyonları şu indeksleri üretir: **0, 3 ve 9**
* İndeks **0** daha önce **0** olduğu için **1** yapılır. Ancak, **3 ve 9** zaten "cow" eklenirken **1** olduğu için tekrar değişmez.

**3. "dog" kelimesinin eklenmesi**

* "dog" için hash fonksiyonları şu indeksleri üretir: **10, 9 ve 8**
* Bu indeksler **1** olarak işaretlenir ve sonuçta Bloom filtresinin son hâli elde edilir (**Şekil 8.3**).



Şekil 8.1 Boş Bir Bloom Filtresi



Şekil 8.2 "cow" Kelimesi Bloom Filtresine Eklendikten Sonra



Şekil 8.3 "cow", "cat" ve "dog" Kelimeleri Bloom Filtresine Eklendikten Sonra

Bir Bloom filtresinde bir öğeyi aramak, değerin aynı hash fonksiyonlarıyla yeniden hashlenmesini ve bit dizisi içindeki indekslerin oluşturulmasını gerektirir. Eğer bu indekslerin tamamındaki bitler **1** ise, arama işlemi başarılı olarak raporlanır; aksi takdirde başarısız olur.

Şekil **8.3**'teki Bloom filtresinde **olmayan** bir değeri aramayı düşünelim.

* **"fox"** kelimesini ararsak, üç hash fonksiyonu şu indeksleri döndürür: **3, 12 ve 18**.
* **3 ve 18. indekslerdeki bitler 1**'dir. Ancak, **12. indeksteki bit 0** olduğu için Bloom filtresi "fox"un bulunmadığını bildirir.

Aynı Bloom filtresinde **"rabbit"** kelimesini arayalım.

* "rabbit" için üç hash fonksiyonu şu indeksleri üretir: **8, 9 ve 18**.
* **Bu üç indeksteki bitlerin tümü 1** olduğundan, Bloom filtresi yanlış bir şekilde "rabbit"in eklenmiş olduğunu bildirir.
* **Bu bir yanlış pozitif (false positive) durumudur.** Bu istenmeyen bir durum olsa da , Bloom filtresi kullanılıyorsa kabul edilmesi gerekir.

Eğer bir Bloom filtresinin kullanışlı olması isteniyorsa, **yanlış negatif (false negative) durumu hiçbir zaman oluşmamalıdır**. Yukarıdaki örneklerden de görülebileceği gibi, **yanlış negatiflerin oluşması imkânsızdır**. Ancak, yanlış pozitifler en aza indirilmelidir.

Aslında, **bir Bloom filtresinin ortalama olarak ne sıklıkta yanlış pozitif raporlayacağını belirlemek mümkündür**. Yanlış pozitif olasılığı üç faktöre bağlıdır:

* **Hash fonksiyonlarının sayısı ve kalitesi**
* **Bloom filtresine eklenen öğe sayısı**
* **Bloom filtresinde kullanılan bit sayısı**

Bu faktörlerin analizi, sonraki bölümlerde ele alınacaktır.

8.5 İnceleme Soruları

Bu bölümü ne kadar iyi anladığınızı test etmek için aşağıdaki kısa cevaplı, çoktan seçmeli ve doğru/yanlış sorularını cevaplayın.

1: Hangi veri yapısı yanlış pozitiflere karşı hassastır: trie mi yoksa Bloom filtresi mi?

**Bloom filtresi** yanlış pozitiflere karşı hassastır. Yani, bir öğe gerçekte filtreye eklenmemiş olsa bile, Bloom filtresi bazen o öğenin var olduğunu bildirebilir. **Trie** veri yapısı ise yanlış pozitif üretmez çünkü eklenen öğelerin tamamını açıkça saklar.

2: Bu bağlamda yanlış pozitif (false positive) nedir?

Yanlış pozitif (**false positive**), Bloom filtresinin bir öğenin kümede olduğunu bildirmesine rağmen, aslında o öğenin kümede olmaması durumudur.

3: Bir Bloom filtresi, bir trie'dan daha fazla mı yoksa daha az mı depolama alanı gerektirir

Bir **Bloom filtresi**, bir **trie**'dan **daha az** depolama alanı gerektirir. Çünkü Bloom filtresi yalnızca bir bit dizisi ve birkaç hash fonksiyonu kullanırken, trie her düğümde karakterleri ve bağlantıları saklar, bu da daha fazla bellek tüketir.