ÜYELİK YAPILARI

Bölüm 5'te ekleme, silme, üyelik testi ve yinelemeyi destekleyen veri yapılarını ele aldık. Bazı uygulamalar için üyelik testi yeterli olabilir. Yineleme ve silme gerekli olmayabilir.

Klasik örnek bir yazım denetleyicisidir. Bir yazım denetleyicisinin işini düşünün. Basit bir yazım denetleyicisi yazım hatalarını tespit edebilirken, daha gelişmiş bir yazım denetleyicisi doğru yazılmış kelimelerin alternatiflerini önerebilir.

Açıkça görüldüğü gibi yazım denetleyicisi geniş bir sözcük sözlüğü ile birlikte sunulur. Yazım denetleyicisi, sözcük listesini kullanarak bir sözcüğün sözlükte olup olmadığını ve dolayısıyla doğru bir sözcük olup olmadığını belirler. Eğer kelime sözlükte yer almıyorsa, kelime işlemci ya da editör kelimenin altını çizerek yanlış yazılmış olabileceğini belirtir. Bazı durumlarda, kelime işlemci doğru yazılmış alternatif bir kelime önerebilir. Bazı durumlarda ise kelime işlemci sadece yanlış yazımı düzeltebilir. Bu yazım denetleyicileri/düzelticileri nasıl çalışır? Ne tür veri yapıları kullanırlar?

Bölüm Hedefleri

İlk bakışta, bir hash kümesi (yani bir Python sözlüğü) yazım denetimi için uygun bir veri yapısı gibi görünebilir. Küme içinde arama süresi O(1) zamanında yapılabilir. Ancak, bu hash haritasının boyutunda bir değiş tokuş söz konusudur. Tipik bir İngilizce sözlük 100.000'den fazla kelime içerebilir. Bu kadar çok kelimeyi saklamak için gereken alan miktarı oldukça büyük olacaktır.

Bu bölümde, bir küme içindeki üyeliği test etmek için tasarlanmış iki veri yapısını ele alacağız. Birincisi, bir bloom filtresi, önemli ölçüde daha az alan gereksinimine sahiptir ve çok hızlı bir üyelik testi sağlar. Diğeri ise bir trie (try olarak okunur) veri yapısıdır ve bir hash kümesi uygulamasında kolaylıkla kullanılamayacak özelliklere sahiptir ve bir hash kümesinden daha az yer kaplayabilir.

Bloom Filtreleri

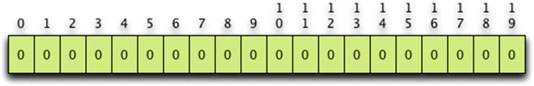
Bloom filtreleri, bu fikri ilk olarak 1970 yılında ortaya atan yaratıcıları Burton Howard Bloom'un adıyla anılmaktadır. O zamandan beri Alan Tharp [7] da dahil olmak üzere birçok yazar bloom filtrelerinin uygulamalarını ele almıştır. Wikipedia, her zaman yetkili bir kaynak olmasa da, bloom filtreleri hakkında çok iyi bir tartışmaya sahiptir [8].

Bir bloom filtresi, çok daha az alan kullanırken hash kümeleriyle bazı fikirleri paylaşır. Bir bloom filtresi, bir öğenin bir değerler kümesinin üyesi olup olmadığını belirlemek için istatistiksel olasılık kullanan bir veri yapısıdır. Bloom filtreleri %100 doğru değildir. Bir bloom filtresi, küme üyeliği için asla yanlış negatif raporlamaz, yani bir öğenin gerçekte bir kümeye ait olduğu halde olmadığını asla raporlamaz. Ancak, bir bloom filtresi bazen yanlış pozitif raporlayacaktır. Bir öğenin gerçekte bir kümede olmadığı halde kümede olduğunu bildirebilir.

Yazım denetimi sorununu düşünün. Bir yazım denetleyicisinin, yazılan bir kelimenin sözlükte aratılarak doğru yazılıp yazılmadığını bilmesi gerekir. Bir bloom filtresi ile, yazılan kelime, doğru yazılmış bir kelime olduğunu veya olmadığını bildirecek olan bloom filtresine verilebilir. Bazı durumlarda, doğru olmadığı halde bir kelimenin doğru olduğunu bildirebilir.

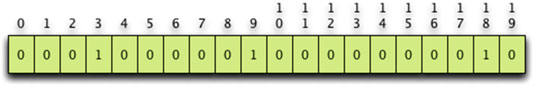
Bir bloom filtresi, bir dizi hashing fonksiyonuyla birlikte bir bit dizisidir. Filtredeki bit sayısı ve hash fonksiyonlarının sayısı bloom filtresinin doğruluğunu etkiler. Bitlerin ve hash fonksiyonlarının tam sayısı daha sonra tartışılacaktır. .

Bir bloom filtresini 20 bit ve 3 bağımsız hash fonksiyonu ile düşünün. Başlangıçta filtredeki tüm bitler Şekil 8.1'de gösterildiği gibi 0 olarak ayarlanır.



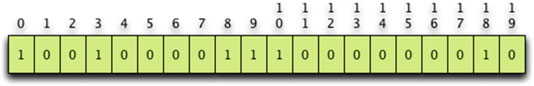


Bloom filtresine cow kelimesini eklediğinizi düşünün. Üç bağımsız hash fonksiyonunun cow kelimesini modulo 20 ile sırasıyla 18, 9 ve 3'e hashlediğini varsayalım. Şekil 8.2'de gösterildiği gibi cow’un filtreye eklendiğini hatırlamak için 18, 9 ve 3 indislerindeki bitler 1 olarak ayarlanır.





Şimdi aynı filtreye cat kelimesini eklemeyi düşünün. Üç hash fonksiyonundan elde edilen hash değerlerinin, modulo 20 ile 0, 3 ve 9 olduğunu varsayalım. Filtreye cat eklendiğinde 0 indeksindeki bit 1 olarak ayarlanır. Diğer ikisi zaten cow eklendiğinde ayarlanmıştı. Son olarak, filtreye dog eklendiğinde Şekil 8.3'tegösterilen bloom filtresi elde edilir. Dog için hash değerleri 10, 9 ve 8'dir.





Bir bloom filtresinde bir öğeyi aramak, bit dizisindeki indisleri oluşturan aynı hash fonksiyonlarıyla değeri tekrar hashlemeyi gerektirir. Bit dizisindeki tüm indislerdeki değer bir ise, arama işlevi başarılı, aksi takdirde başarısız olduğunu bildirir. Şekil 8. 3'teki Bloom filtresinde olmayan bir değeri aradığımızı düşünelim. Fox’u ararsak, üç hash fonksiyonu çağrısı 3, 12 ve 18 değerlerini döndürür. Indeks 3 ve 18'deki rakam 1'dir. Ancak, indeks 12'deki rakam O'dır ve arama fonksiyonu şunu rapor eder Fox ,Bloom filtresinde değil.

Aynı bloom filtresinde rabbit değerini aradığınızı düşünün. Üç hash fonksiyonuyla rabbit değerinin karması 8, 9 ve 18 değerlerini verir. Bloom filtresi içindeki bu konumlarda bulunan rakamların üçü de 1 içerir ve bloom filtresi rabbit’in filtreye eklendiğini yanlış bir şekilde bildirir. Bu bir yanlış pozitiftir ve arzu edilmese de bir bloom filtresi kullanılacaksa kabul edilebilir olmalıdır.

Bir çiçek filtresinin faydalı olabilmesi için asla yanlış negatif raporlamaması gerekir. Bu örneklerden yanlış negatiflerin imkansız olduğu açıkça anlaşılmalıdır. Yanlış pozitifler minimum düzeyde tutulmalıdır. Aslında, bir bloom filtresinin ortalama olarak ne sıklıkla yanlış pozitif raporlayacağını belirlemek mümkündür. Olasılık hesaplaması üç faktöre bağlıdır: hashing fonksiyonları, bloom filtresine eklenen öğe sayısı ve bloom filtresinde kullanılan bit sayısı. Bu faktörlerin analizi sonraki bölümlerde ele alınmaktadır.

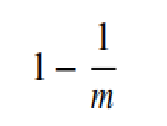
Hashing İşlevleri

Bir bloom filtresine eklenen her öğe, birbirinden tamamen bağımsız olan bir dizi hash fonksiyonu ile hashlenmelidir. Her bir hash fonksiyonu ayrıca bit dizisindeki bit indeksleri aralığına eşit olarak dağıtılmalıdır. Bu ikinci gereklilik, hash kümeleri ve hash tabloları için hash fonksiyonları için de geçerlidir. Eşit dağılım Python'un ve diğer birçok dilin yerleşik hash fonksiyonları tarafından garanti edilir. Yukarıdakı örneklerde üç hash fonksiyonu gereklıydı. Bazen eklenen öğe sayısına ve bit dizisindeki bit sayısına bağlı olarak gerekli hash fonksiyonu sayısı çok daha fazla olabilir. Gerekli sayıda bağımsız, eşit dağılımlı hashing fonksiyonu oluşturmak göz korkutucu bir sorun gibi görünebılır, ancak en azından bırkaç yolla çözülebilir. Bazı hashing fonksiyonları bir çekirdek değerinin verilmesine izin verir. Bu durumda, farklı çekirdek değerleri farklı hashing fonksiyonları oluşturmak için kullanılabilir.

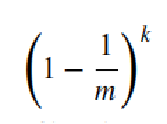
Bağımsız hash fonksiyonları oluşturmanın eşit derecede etkili bir başka yolu da hashlenmeden önce her öğenin sonuna bilinen bir değer eklemektir. Örneğin, ilk hash fonksiyonunu elde etmek için hashlemeden önce öğeye bir 0 eklenebilir. İkinci hash fonksiyonunu elde etmek için hash işleminden önce öğeye 1 eklenebilir. Benzer şekilde, üçüncü hash fonksiyonu değerini elde etmek için öğeye 2 eklenebilir. Dolayısıyla, Bloom filtresinde rabbit’i aramak, önce aynı hash işlevine sahip rabbit0, rabbit1 ve rabbit2'nin hash haline getirerek gerçekleştirilir .Hash işlevi eşit olarak dağıtıldığından, üç hash değerinin döndürdüğü değerler birbirinden bağımsız olacaktır. Ve 0 eklenmiş tüm öğelerin kendileri de eşit dağılımlı olacaktır. Aynı şekilde 1 ve 2 eklenmiş öğeler için de olacaktır.

Bloom Filtre Boyutu

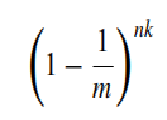
Eklenecek öğe sayısı ve istenen yanlış pozitif olasılığı göz önüne alındığında gerekli bloom filtresi boyutunu bulmak mümkündür. Bir öğe eklenirken bloom filtresi içindeki herhangi bir konumun bir hash fonksiyonu tarafından ayarlanmama olasılığı, filtrenin m bitten oluştuğu aşağıdaki formülle verilir.



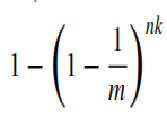
Bloom filtresi k adet hash fonksiyon kullanıyorsa, bit dizisindeki bir bitin bir öğe eklemek için gereken hash fonksiyonlardan herhangi biri tarafından ayarlanmamış olma olasılığı bu formülle verilir.



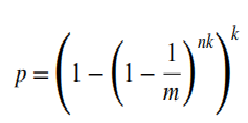
Bloom filtresine n öğe eklenirse, bu formülü n'e yükseltmek, bloom filtresinin bit dizisindeki bir bitin tüm n öğe eklendikten sonra hala sıfır olma olasılığını sağlayacaktır.



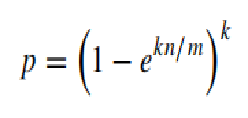
Dolayısıyla, k hashing fonksiyonu kullanılırken n öğe eklendikten sonra bloom filtresindeki bir bitin 1 olma olasılığı bu formülle verilir.



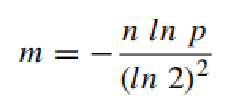
Şimdi bloom filtresine eklenmemiş bir öğeyi aradığınızı düşünün. Yanlış pozitif raporlama olasılığı, bloom filtresi içindeki her konumun tüm k hash işlevleri için 1 olma olasılığı hesaplanarak bulunabilir. Bu aşağıdaki gibi ifade edilir.



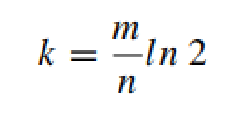
Bu formül, doğal log [8] kullanılarak yaklaşık olarak tahmin edilebilen bir dizi içerir.



Bu formülü kullanarak, bir n ve istenen yanlış pozitif olasılığı, p, verildiğinde m'yiçözmek mümkündür. Formül aşağıdaki gibidir.



Son olarak, yukarıdaki k değerini çözdüğümüzde aşağıdaki formül elde edilir.



Bu iki formül bize, belirlenen maksimum yanlış pozitif oranını garanti etmek için filtremizde kaç bit gerektiğini söyler. Ayrıca gerekli hash fonksiyonu sayısını da hesaplayabiliriz. Örneğin, 109.583 kelime içeren bir İngilizce sözlük ve %l'den fazla olmayan istenen yanlış pozitif yüzdesi (formülde 0,01 olarak ifade edilir) için 1.050.360 bitlik bir bit dizisi ve yedi hash fonksiyonu gerekir.

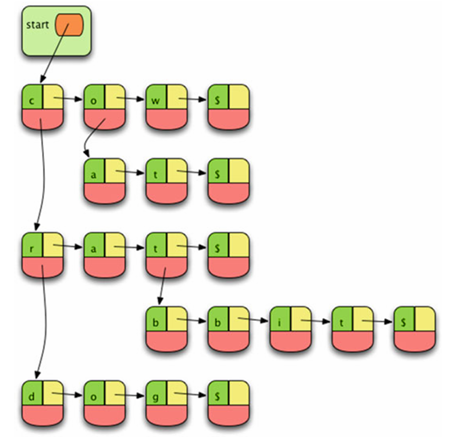
Bu örnekteki bit sayısı aşırı görünebilir. Ancak, bunların bit olduğunu hatırlayın. Verimli bir uygulama yaklaşık 128 KB depolama alanı gerektirir. Bu örneklerde kullanılan İngilizce sözlükteki karakter sayısı toplam 935.171'dir. Karakter başına 1 bayt olduğu varsayılırsa, tüm bu kelimelerin saklanması için en az 914 KB gerekir. Bloom filtresi oldukça büyük bir alan tasarrufu sağlamaktadır. Buna ek olarak, deneyler sırasında çiçek filtresi kullanılarak yapılan arama süresi hiçbir zaman 160 µs'den uzun sürmemiştir. Arama süresi, istenen değerleri hesaplamak için kullanılan hash fonksiyonlarının sayısı ve verimliliği ile sınırlıdır. Hash fonksiyonlarının hashlenen dizenin uzunluğuna bağlı olduğu varsayılırsa, arama süresi O(1k) olur, burada 1, aranan öğenin uzunluğu ve k da hash fonksiyonlarının sayısı ile verilir.

**Bloom Filtresinin Dezavantajları**

Bariz yanlış pozitif potansiyelinin yanı sıra, çiçek filtresi yalnızca evet veya hayır olarak rapor verebilir. Doğru yazılmaya yakın olabilecek öğeler için alternatifler öneremez. Bir çiçek filtresinin hangi bitlerin hangi öğeler tarafından ayarlandığına dair bir hafızası yoktur, bu nedenle evet veya hayır cevabı, bazı durumlarda doğru olmayan bir evet cevabıyla bile alabileceğimiz en iyi cevaptır. Bir sonraki bölümde, yanlış pozitifleri rapor etmeyecek ve yanlış yazılan kelimelerin alternatiflerini bulmak için kullanılabilecek bir Trie veri yapısı sunulmaktadır.

**Trie Veri Türü**

Trie, reTRIEval için tasarlanmış bir veri yapısıdır. Veri yapısı try kelimesi gibi isimlendirilir. Bir trie, bir veri yapısından değerlerin silinmesi gerektiğinde kullanılmak üzere tasarlanmamıştır. Yalnızca bir anahtar değere dayalı olarak öğelerin alınması içindir.



Şekil 8.4 Trie'ye İnek, Kedi, Sıçan, Tavşan ve Köpek Eklendikten Sonra

Anahtar değerler birden fazla birimden oluştuğunda ve bir anahtarın bağımsız birimleri diğer öğe anahtarlarıyla çakışabildiğinde denemeler uygundur. Aslında, anahtar birimleri ne kadar çok örtüşürse, trie veri yapısı o kadar kompakt olur.

Yazım denetimi probleminde, kelimeler karakterlerden oluşur. Bu karakterler anahtarların bireysel birimleridir. Bir sözlükte a, an ve ant gibi birçok kelime üst üste gelir. Bir trie birkaç farklı şekilde uygulanabilir. Bu metinde, bir matris oluşturan bir dizi bağlantı listesi olan bağlantılı trie üzerinde yoğunlaşacağız. Matris uygulamaları, boş konumlarla çok daha fazla yer kaplayan seyrek nüfuslu dizilere yol açar. Bağlantılı trie'nin işaretçiler için ek yükü vardır, ancak seyrek nüfuslu değildir.

Trie veri yapısı boş bir bağlı liste ile başlar. Bağlı trie listesindeki her düğüm üç değer içerir: anahtarın bir birimi (yazım denetimi örneğinde bu, kelimenin bir karakteridir), listede bir anahtar (yani kelime) içinde aynı konumda görünen başka bir birimi (yani karakter) içeren bir sonraki düğümü işaret eden bir sonraki işaretçi ve aynı anahtar içinde bir sonraki birimi içeren bir düğümü işaret eden bir takip eden işaretçi. Şekil 8.4'te takip eden işaretçisi sarı renkte, sonraki işaretçi alanı ise kırmızı renktedir.

**Trie Veri Tipi**

Öğeler trie'ye eklendiğinde bir sentinel birimi eklenir. Yazım denetleyicisi durumunda, her kelimenin sonuna bir '$' karakteri eklenir. Rat gibi sözcükler ratchet gibi sözcüklerin ön eki olduğu için gönderme birimine ihtiyaç duyulur. Sentinel karakteri olmadan, bir kelimenin sonunun mu yoksa başka bir kelimenin ön eki mi olduğu belirsiz olurdu.

Bir üçlüde, ortak bir öneki olan anahtarlar bu öneki paylaşır ve tekrarlanmaz. Birden fazla olası sonraki karakter mümkün olduğunda sonraki işaretçi kullanılır. Bu, veri yapısında yer tasarrufu sağlar. Bunun karşılığında next ve follows işaretçileri her düğümde fazladan yer kaplar.

**Trie Sınıfı**

1 Trie sınıfı:

2 def insert(node,item):

3 # Bu özyinelemeli ekleme işlevidir.

4

5 def contains(node,item):

6 # Bu özyinelemeli üyelik testidir.

7

8

9 sınıf TrieNode:

10 def init (self,item,next = None, follows = None):

11 self.item = item

12 self.next = next

13 self.takip ediyor = takip ediyor

14

15 def init (self):

16 self.start = Yok

17

18 def insert(self,item):

19 self.start = Trie. insert(self.start,item)

20

21 def contains (self,item):

22 return Trie. contains(self.start,item)

**Trie'ye Ekleme**

Bir trie'ye değer ekleme işlemi bir döngü ile yinelemeli olarak ya da özyinelemeli olarak yapılabilir. Bir trie'ye özyinelemeli olarak değer eklemek için insert yöntemi bir insert fonksiyonunu çağırabilir. Fonksiyona aktarılan düğüm değeri None olabileceğinden, özyinelemeli kodu Trie sınıfının bir metodu olarak değil, bir fonksiyon olarak yazmak daha kolaydır. Trie'ye eklemek için insert fonksiyonu aşağıdaki gibi çalışır.

⦁ Anahtar boşsa (yani anahtarda hiç birim kalmamışsa), boş düğüm olarak None

döndürülür.

⦁ Düğüm Yok ise, anahtarın bir sonraki birimi ile yeni bir düğüm oluşturulur ve

anahtarın geri kalanı eklenir ve aşağıdaki bağlantıya eklenir.

⦁ Anahtarın ilk birimi geçerli düğümün birimiyle eşleşirse, anahtarın geri kalanı düğümün aşağıdaki bağlantısına eklenir.

⦁ Aksi takdirde, anahtar düğümün bir sonraki bağlantısına eklenir.

Trie'yi özyinelemeli olarak oluşturmak basittir. Ancak, yinelemeli bir sürüm de aynı şekilde çalışacaktır. Yinelemeli sürüm, bir döngü ve eklenecek kalan anahtarla birlikte geçerli düğüme bir işaretçi gerektirecektir. Yinelemeli ekleme algoritması yukarıda özetlenen adıma benzer bir şekilde davranacaktır, ancak bağlantıların doğru bir şekilde ayarlanabilmesi için mevcut düğümün yanı sıra önceki düğümü de takip etmesi gerekecektir.

**Üçlü Birliğe Üyelik**

Bir trie'deki üyeliğin kontrol edilmesi özyinelemeli olarak da gerçekleştirilebilir. Adımlar, ilk başta tamamen sezgisel olmayabilecek bir temel durum içerir. Boş anahtar herhangi bir trie'nin üyesi olarak rapor edilir çünkü üyelik kontrol edilirken çalışır. Trie'ye eklenen sentinel birimi ile, boş bir anahtar için True döndürmek tamamen güvenlidir çünkü herhangi bir gerçek anahtar en azından sentinel karakterinden oluşacaktır. Burada özetlenen algoritmada, sentinelin anahtara zaten eklenmiş olduğu varsayılır. Üyelik testi için adımlar aşağıdaki gibidir.

⦁ Anahtarın uzunluğu 0 ise, True döndürerek başarıyı raporlayın.

⦁ Eğer baktığımız düğüm None ise, False döndürerek başarısızlığı bildirir.

⦁ Anahtarın ilk birimi geçerli düğümdeki birimle eşleşiyorsa, aşağıdaki

düğümden başlayarak anahtarın geri kalanının üyeliğini kontrol edin.

⦁ Aksi takdirde, triyedeki bir sonraki düğümden başlayarak anahtarın üyeliğini kontrol edin.

Yine bu kod, mevcut düğümü ve anahtarın geri kalanını takip eden bir while döngüsü ile yinelemeli olarak uygulanabilir. Özyinelemeli veya yinelemeli bir uygulama eşit derecede iyi çalışacaktır.

**Denemeleri ve Bloom Filtrelerini Karşılaştırma**

Bloom filtreleri, üyeliği test etmek için bir trie'den açıkça daha hızlıdır. Bununla birlikte, üçlü kabul edilebilir derecede iyi çalışır. Basit bir deneyde en uzun bloom filtresi arama süresi 160 µs iken, en uzun trie arama süresi 217 µs'dir. Elbette trie daha fazla yer kaplar, ancak ortak önekler bir trie'deki düğümleri paylaşarak her bir kelimeyi bir hash kümesinde olduğu gibi bir veri yapısında ayrı ayrı depolamaya göre biraz yer tasarrufu sağlar.

Yazım denetimi amacıyla bir trie'nin belirgin avantajları vardır, çünkü yazım değişiklikleri kolayca bulunabilir. Yaygın yazım hataları dört kategoriden birine girer.

⦁ Teh gibi karakterlerin the yerine transpoze edilmesi

⦁ Onların yerine thei gibi karakterler bırakıldı

⦁ yerine thre gibi ekstra karakterler

⦁ There yerine thare gibi yanlış karakterler

Bir trie'de arama yapıldığında bir kelime bulunamazsa, alternatif yazımların bir seçimini bulmak için bu alternatifler de aranabilir. Dahası, bu alternatif yazımlar, alternatiflerin bir listesini hızlı bir şekilde bir araya getirmek için bir trie'de paralel olarak aranabilir. Alternatif yazımları bulmak için bir çiçek filtresi kullanılamaz, çünkü bu bilgi filtreye girildikten sonra kaybolur. Elbette bir trie, çiçek filtresinde mümkün olduğu gibi asla yanlış pozitif raporlamayacaktır.

**Bölüm Özeti**

Denemeler ve bloom filtreleri üyeliği test etmek için kullanılan iki veri yapısıdır. Bloom filtreleri nispeten küçüktür ve zamanın belirli bir yüzdesinde yanlış pozitifler üretecektir. Denemeler daha büyüktür, yanlış pozitifler üretmez ve aranan anahtara yakın alternatif anahtar değerleri bulmak için kullanılabilir. Her iki veri yapısı da yazım denetimi için işe yarayacak olsa da, yazım düzeltmesine bir deneme yardımcı olurken bir bloom filtresi yardımcı olmaz.

Verimlilik söz konusu olduğunda, bloom filtreleri, bazen üretilen yanlış pozitiflere tabi olarak, küme üyeliğini daha verimli bir şekilde test eder. Bununla birlikte, bir trie de verimli bir şekilde çalışır ve aynı zamanda bir bloom filtresinden daha fazla yer kaplar. Hem bloom filtresi hem de trie, sözlükteki kelimelerin üyeliğini mikrosaniyeler içinde test etmiştir. Bloom filtresinin en kötü süresi 160 µs iken trie'nin en kötü süresi her ikisinde de gerçekleştirilen gayri resmi test için 217 µs'dir.

Boyut gereksinimleri de elbette bir endişe kaynağıdır. Bu bölümde hem bloom filtresinin hem de trie'nin geliştirilmesinde kullanılan örnek sözlük 109.583 kelime içeriyordu. Bu sözcük sözlüğü için çiçek filtresi yaklaşık 128 KB boyutundaydı. Sonraki ve takip eden işaretçilerin her birinin 4 bayt ve anahtar birimlerin (yani kelime karakterlerinin) her birinin 1 bayt aldığı varsayıldığında, trie'nin boyutu kabaca 3,1 MB'tır. Çiçek filtresi trie'den çok daha küçük olsa da, her ikisi de bilgisayarların depolama kapasitesinin sınırları içindedir.

**İnceleme Soruları**

Bu kısa cevaplı, çoktan seçmeli ve doğru/yanlış soruları cevaplayarak bölüme hakimiyetinizi test edin.

⦁ Hangi veri türü, trie veya bloom filtresi, yanlış pozitiflere karşı hassastır?

⦁ Bu bağlamda yanlış pozitif nedir?

⦁ Bir çiçek filtresi, bir trie'den daha fazla mı yoksa daha az mı depolama gerektirir?

⦁ Yazım denetimi yaparken, yazım düzeltmesi için hangi veri türü kullanılabilir?

⦁ Bir çiçek filtresinde kullanılmak üzere birden fazla karma işlevi nasıl oluşturabilirsiniz?

⦁ "a", "an", "ant", "bat" ve "batter" kelimelerini bir trie'ye ekleyiniz. Kelimeleri burada verilen sırayla ekledikten sonra yapısını gösteren trie veri yapısını çizin.

⦁ Bir trie'de neden bir nöbetçiye ihtiyaç duyulur?

⦁ Bir çiçek filtresinde neden bir sentinel gerekli değildir?

⦁ Anahtarların bir trie'de saklanabilmesi için ne doğru olmalıdır?

⦁ Hangi veri türü, trie veya bloom filtresi, alan açısından daha verimlidir? Hız açısından hangisi daha verimlidir?

**Programlama Problemleri**

⦁ Metin web sitesine gidin ve kelime sözlüğünü indirin. Bu kelime listesi için bir bloom filtresi oluşturun ve bağımsızlık bildirgesini yazım denetimi yapmak için kullanın, yanlış yazılan tüm kelimeleri ekrana yazdırın.

⦁ Metin web sitesine gidin ve kelime sözlüğünü indirin. Bu sözcük listesi için bir trie veri türü oluşturun ve bağımsızlık bildirgesini yazım denetimi yapmak için kullanın, yanlış yazılan tüm sözcükleri ekrana yazdırın.

⦁ Önceki alıştırmada olduğu gibi bir trie oluşturun, ancak yanlış yazılan tüm kelimeler için önerilen değişiklikleri de yazdırın. Bu zor bir ödevdir. Önerilen değişiklikler orijinalinden bölümde önerilen yollardan birden fazlasıyla farklı olmamalıdır.