

**Algoritma Analizi 3. Ödevi**

**Muhammet Kayra Bulut**

**20011901**

**Yöntem**

Hash tablomu bir dizi olarak tuttum ve dizimin elemanlarını struct olarak tuttum. Structumun içeriğiyse flag, key, link strcut’ı ve deneme sayısı. Deneme sayısı, kaç denemede yerine yerleştiğini tutuyor. Key değeri unique olarak key değerini tutuyor ve arama işlemlerinde bu değer kullanılıyor. Flag değeri olası silme işlemleri için tutuluyor. Links struct pointer’ı da alakalı bağlantıları daha kolay takip edebilmek için tutuluyor.

Problemi çözmek için öncelikle hash fonksiyonu yazdım ve aynı zamanda asal sayı bulma fonksiyonu yazdım. Sonrasında hash tablosuna ekleme çıkarma işlemleriyle alakalı fonksiyonları yazdım. Dosyadan okumak için de bir struct tanımladım ve tuttum. Aynı zamanda key değerlerini bir linkli listede sakladım. Bu şekilde kodum daha anlaşılır ve aynı zamanda daha kolay implemente edilebilir bir hal aldı. Sonrasında eleman bulma işlemleri için arama fonksiyonları ve yazdırma fonksiyonları yazdım ve switch-case yapısıyla programı sürekli ayakta tuttum.

**Uygulama**

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Sonuç**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. **Değeri İçin Load Factor**

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**0.5 Değeri İçin Load Factor**

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu0.1 Değeri İçin Load Factor**

Load Factor yükseldikçe, deneme sayısında yükselme gözlemleniyor ancak deneme sayısı 0.5’den daha düşük olduğunda harcanan alana göre kazanç çok da yüksek olmadığından, load factor %50 civarında tutulursa daha mantıklı olur.

Hash tablosunun uzunluğunu bulurken “((float)100/loadFactor)\*keys->elemanSayisi)” şeklinde sabit bir işlem yatıktan sonra, findPrimeNumber isimli fonksiyonu çağırıyorum. Bu fonksiyon, o sayıdan daha büyük olan en küçük asal sayıyı buluyor. Bunun karmaşıklığıysa yaklaşık olarak n. Çünkü asal sayı bulmak zor bir iş. Ama bir defa tablo oluşurken yapıldığı için göz ardı edilebilir.

Arama motoru algoritmamın karmaşıklığı en kötü durumda O(n). Ama bu durum istatistiksel olarak neredeyse imkansız. Arama motoru algoritmam hash değerini alıp indise bakıyor ve sonra o indiste bulunamazsa tekrar hash değeri hesaplıyor. Ortalama olaraksa load factor değerine bağlı olarak 2.5 çalışmada buluyor. Hashing yerine binary search’ü kullanırsak karmaşıklık O(n\*log(n)) olur. Bu karmaşıklık sabit sayıda işleme göre hayli yüksek.