Hakkında

Program BitBucket Git sistemi üzerinde yazılmış ve toplamda 8 sürümden meydana gelmiştir. Ödev dokümanında istenilen veri yapılarını, dosyalama ve sıralama işlemlerini gerçekleştirmektedir. Görsel ara yüze, renklendirme ve dosya isimlerinin yazılması gibi birkaç küçük detay eklenmiştir.

Giriş

Programda 9 farklı sınıf bulunmaktadır; *BagilListe, Dosya, Dugum, Gezici, Hata, IkiliHeap, Islem, Konsol, Sayi.*

Sınıf sayısının bu kadar fazla olmasının nedeni daha önceki ödevlerde oluşturulan sınıfların bu ödeve dahil edilmesi ve tamamlayıcı bir etki oluşturulmasının planlanmasıdır. Bu veri yapılarının hiçbirisi boşuna projeye dahil edilmemiştir. Aynı zamanda bu ödevde kullanılması istenilen ikili heap ağacı yapısı da gerektiği şekilde kullanışmış ve ödevde yapılması istenilen temel kıstaslara ek bir müdahale yapılmamıştır.

Önceki ödevlerde kullanılmış olan veri yapılarının yazlınca bu ödevde neden kullanıldıkları açıklanmıştır, işlevleri hakkında detaylı bilgi verilmemiştir.

Sayı

Bu ödevde de kullanılması gereken bir veri yapısıdır, ikinci ödevde kullanılan sınıf projeye dahil edilmiştir fakat tasarımı gereği bu sayı sınıfı elemanlarını bir bağıl listede barındırır dolayısıyla bu sınıfın eklenmesi bağıl liste sınıfının da projeye dahil edilmesini zorunlu kılmıştır.

BağılListe

Bağıl liste sınıfının bu ödevde tek bir işlevi vardır, sayı sınıfı rakamlarını kendine içsel bağıl liste nesnesinde tutar. Bağıl liste ile beraber bu sınıfın iç mekanizmasında kullanılacak olan düğüm ve gezici yapıları da ödeve dahil edilmiştir.

Düğüm

Bağıl liste içerisinde kullanılır, her bir düğümde bir rakam yani bir karakter saklanır.

Gezici

Bağıl listede düğümler arasında gezme işleminin kolaylaştırılması için kullanılan veri yapısıdır.

İşlem

Sayı üzerinde yapılacak olan işlemleri düzenli tutmak için oluşturulmuş olan bir sınıftır.

Dosyalama İşlemleri

Bir önceki ödevde oluşturulan *Dosya* sınıfı bu ödevde de kullanılmış, üzerinde bazı değişiklikler yapılmıştır. Bu sınıfın kullanılmasının amacı dosyadan alınan ve dosyaya yazılacak olan içeriğin yönetilmesi ve dosya üzerinde doğrudan yapılan işlemlerin kolaylaştırılmasıdır. Sınıf yapısında yapılan en büyük değişiklik yazma işlemlerinin de sınıfa eklenmesidir. Okuma işlemindekine benzer bir şekilde *satirEkle* metodu ile eklenecek olan içeriği tek tek alır ve kendi içinde yer alan *String* tipinde bir değişkende saklar. Daha sonra *dosyayaKaydet* metodu ile bu değişken içeriği dosyaya aktarılır.

Dosyadan Okunan Sayıların Sayı Nesnesine Aktarılması

Oluşturulan dosya nesnesi üzerinden *dosyayiOku* metodu çağrılarak dosya içeriği dosya nesnesinin kendi içerisindeki *String* değişkenine aktarılır. İçeriğin Sayı nesnesine aktarılabilmesi için öncelikle içerik dosyadan okunmuş olmalıdır. Sonrasında bu *String* değişkenindeki tüm karakterler satır sonu karakterlerine göre ayrım yapılarak okunur ve her bir satırın karakterleri ayrı bir Sayı nesnesine rakamlar olarak aktarılır. Bu işlem *Dosya* sınıfının *sayiyaAktar* metodu üzerinden gerçekleştirilir. Gönderilecek olan parametre bir *Sayi* dizisi olmalıdır. Dizinin boyutunun yeterli olduğu kabulü yapılır ve ek bir denetime tabi tutulmaz.

Dosyaya Yazma İşlemi

Dosya nesnesi oluşturulurken kurucu metot parametresi olarak gönderilen dosya adı yazma esnasında oluşturulacak veya üzerine yazılacak olan dosya adını belirtir.

Yazma işleminde yazılacak olan içerik önce *satirEkle* metodu ile alınır. Bu şekilde eklenen her bir karakter dizisi dosya nesnesinin iç *String* değişkenine ayrı bir satır olarak eklenir, yani satır sonu karakteri ile ayrılırlar. İçerik ekleme işlemi tamamlandıktan sonra *dosyayaKaydet* metodu ile bu *String* değişkeninin değeri dosyaya kaydedilir.

İkili Heap Ağacı

Öncelikle bu veri yapısı bu projedeki kullanımı itibariyle herhangi başka bir veri yapısıyla bağdaşık olarak çalışmaz. Fakat veri olarak barındırdığı veri yapısı ***Sayi***'dır.

Gerçekleme **dizi** ile yapılmıştır ve **çift katlı *Sayi* göstericisi** verileri tutmak için kullanılmıştır. İkili heap ağacında *yukarı* ve *aşağı fırlatma* işlemlerinde düğümlerin sıklıkla yer değiştirmesi gerekmektedir. Dizinin her bir hücresinde yer alan nesnelerin atama operatörleri kullanılarak verilerini değiştirmek çok sağlıklı bir çözüm olmayacağından çift katlı bir gösterici, yani bir gösterici dizisi kullanılmıştır. Bu sayede yer değişikliği yapmak için tek yapılması gereken hücrelerdeki adresleri değiş tokuş etmek olacaktır.

Ekleme İşlemi

Öncelikle dizi kapasitesinin yeterli olup olmadığına bakılır, eğer kapasite yetersizse *Genislet* metodu ile dizi kapasitesi iki katına çıkartılır.

İlk olarak parametre olarak gelen veri dizinin en sonuna yerleştirilir. Daha sonra *YukariFirlat* metodu dizinin son indeksi ile çağrılır. Bu metot Heap ağacının standart bir metodudur ve işlevi de ebeveyn düğüm yavru düğümden büyük olduğu sürece yerlerini değiştirmek ve özyinelemeli olarak bu işlemi kök düğüme kadar sürdürmektir. Bu sayede kök düğüme en küçük verinin yerleşmiş olması garanti edilir.

En Küçük Veriyi Getirme İşlemi

Ekleme esnasında en küçük verinin kök düğüme yerleşmiş olması sağlandığından yapılması gereken tek şey dizinin ilk adresinde yer alan verinin getirilmesi olacaktır. Gerçekleme dizi ile yapıldığından 0. İndekste yer alan veri daima en küçük olacaktır.

En Küçük Veriyi Silme İşlemi

Gerçekleme dizi ile yapıldığından silme işlemi için ek bir uygulama yapılması gerekmez fakat en küçük veri dizinin kök düğümünde yer alan veri olduğundan ve silinmiş olarak kabul edileceğinden ağaç yeniden düzenlenmelidir. Bunun için *AsagiFirlat* metodu çağırılır. Bu metodun görevi, kökten başlayarak ve özyinelemeli olarak her bir düğümü yavrularıyla karşılaştırmak ve en küçük olan verinin köke yerleşmesini sağlamaktır. Bir düğümün yavru düğümleri o düğümün dizideki indeksinin iki katının bir ve iki fazlası ile bulunabilir. Bu metoda parametre olarak gönderilen indeksin yavru olarak hesaplanan değerleri uzunluktan küçük olduğu, yani yapraktan farklı bir yavru bulunduğu sürece özyineleme işlemi devam eder.

Sonuç

Dosyanın satır sayısı adedince Sayı nesnesi içeren dizi oluşturulur ve dosyadan okunan değerler buna aktarılır. Sonrasında bunların her biri oluşturulan *İkili Heap Ağacı'*na eklenir. Bir döngü içerisinde her seferinden ağaçtan en küçük eleman alınır, dosyaya yazdırılır ve ağaçtan silinir. Nihayet ağaçtaki tüm elemanlar bittiğinde başta "Sayilar.txt" dosyasından okunan ve *Sayı* nesnelerinde tutulan veriler başka bir dosyaya, "Sirali.txt" dosyasına küçükten büyüğe doğru sıralanmış bir şekilde yazılmış olur.

22.12.2016
Emirhan Aydın
G1412.10018