Veri yapıları Ödev 4 Raporu

Yazarlar: İsmet GÜZELGÜN(b121210025@sakarya.edu.tr)
İbrahim AKDAĞ(b121210027@sakarya.edu.tr)
Taşkın TEMİZ(g101210009@sakarya.edu.tr)

Özet

Ödevde C++ programlama diliyle, pointerlar yardımıyla nesneye dayalı programlama paradigmasına uygun bir şekilde altı adet sınıf oluşturmamız istendi. Oluşturacağımız bu sınıflar dosyadan okunacak olan değerleri ikili arama agacina ve heap agacina yerleştirecek olup sonrasında bunları inorder sıralamalıydı.Ödevde bizden istenilen her şeyi gerçekleştirdiğimize inanıyoruz.Detaylar aşağıdadır.

HeapAgaci.cpp

HeapAgaci sınıfı Sehir sınıfı için gereken bir sınıf yapısıydı.İçerisinde string cinsinden değerler tutup istenildiğinde yapısı gereği bu değerleri küçükten büyüğe doğru sıralar.

Sehir.cpp

Sehir sınıfı içerisinde HeapAgaci cinsinden bir nesne tutar.Bu nesne sayesinde dosyadan okuduğumuz şehir değerlerini önce ulke sınıfına oradanda içerisinde tuttuğu şehir nesnesi sayesinde şehir sınıfına ve dolayısıyla heapagacina atabilmekteyiz.Sehir sınıfı içerisinde bir parametresiz kurucu, yıkıcı , heap agacina attığı değerleri yazdırmak istediğimizde kullanılmak üzere çağırılabilecek Yaz() fonksiyonu ve parametreli agacDondur(string) fonksiyonu bulundurmaktadır.agacDondur(string) fonksiyonu UlkeDugum içerisinde Ulke sınıfına taşınan Sehir nesnesi yardımıyla heapagacina değer eklenmek istenildiğinde,Sehir sınıfı içerisinde private olarak tanımlanmış heapagaci nesnesine ulaşıp istenilen değeri eklemek için kullanılacaktır.

UlkeDugum.cpp

Aslında ikili arama ağaçları için kullandığımız Dugum yapısından hiçbir farkı olmayan bu sınıf ekstra olarak içerisinde bir adet Sehir nesnesi tutar.Dugum sınıfından kalıtım alır.Bu sayede her dugum oluşturulduğunda bir tane de Sehir nesnesi oluşturulur bu da o dugum için bir adet heapagaci oluşturulmasını sağlar.Ulke sınıfı içerisinde her dugum oluşturulduğunda bu dugumun içerisinde tutabileceği bir heapagacida hali hazırda oluşturulmuş olur.Bunun dışında içerisinde tuttuğu string cinsinden data değişkeni ile dosyadan okunan ulkeleri tutar.Yine içerisinde tuttuğu sag,sol dugumleri ile ise ikili arama agacina değer yerleştirilirken gereken kontrol işlemlerini ve yönlendirilmeleri yapmayı sağlar.

Ulke.cpp

İkiliaramaagaci sınıfından kalıtım alan bu sınıf temelde bir ikili arama agacidir.İkili arama agaci sınıfında Ulke sınıfında bir takım değişikilikler yapmak gerekebileceği için gerekli olan fonksiyonlar virtual olarak tanımlanmıştı.Bu gerekli değişiklikler yapıldı.Özellikle yerlestir fonksiyonu için Şehir sınıfına ve doğal olarak heap agacina ekleme yapabilmek üzere değişiklikler yapıldı.Bu bahsi geçen değişiklikleri 3.ödev içinde yapmaya çalışmış fakat başarılı olamamıştık.Bu ödev içinse en çok Ulke sınıfının yerleştir fonksiyonunu yazmakta güçlük çektik.Aslında mantık çok basitti.test.cpp dosyasından okunan ve parse edilen değerleri ulke

ve sehir string ifadelerine atayacaktı.Atanan bu değerlerden ulke bir UlkeDugum değişkeni içerisinde Ulke ikili arama agacinin ilk dugumu yani root olmak üzere ve sonrasında okunan diğer ulkelerde bu root un sağına ya da soluna atanmak üzere yerleştirilmekte, sehir stringi ise heap agacina yerleştirilmek üzere yerleştir fonksiyonuna parametre olarak verilmekte.Şimdi burada tüm bu işleri ayrı ayrı yapmakta sorun yok ama eğer okunan Ulke lerden başka bir tane daha varsa o ülkenin heap agacina # işaretinden sonra okunan şehir değerini atamak sıkıntı oluşturmaktaydı.Çünkü eğer o ulke için daha önce bir dugum oluşturulduysa ikili arama agacinda bu düğümün bulunması ve bu düğüm içerisindeki heap agacina gerekli şehrin yerleştirilmesi gerekmekteydi.Bunun için 2 adet ek fonksiyon yazmamız gerekti.Bunlar dugumKontrol(string) ve dugumAra(string) fonksiyonları oldu.Bu fonksiyonlardan dugumKontrol daha önce dugum oluşturulmuş mu oluşturulmamış mı diye kontrol edip bool cinsinden bir değer döndürmekte ve eğer bu değer true ise dugumAra fonksiyonu devreye girip hangi dugum için true döndüyse o düğümü döndürmekte dönen bu düğümün içerisindeki heap agacina şehir değeri eklenmektedir.Tüm bunları yazmak her ne kadar kolay olsa da kurduğumuz bu algoritmanın hatasız çalıştırılması gerçekten bizim için zor oldu.Görünürde hiçbir hata olmamasına rağmen yerlestir fonksiyonundaki bir satırda yaptığımız bir yanlış karşılaştırma sebebiyle inOrder yazdırmak istediğimizde yalnızca root düğümünün içerisindeki heapAgacinin değerlerini yazdırabiliyorduk.yerlestir fonksiyonundaki if-else döngülerinden yalnızca if sadece root için çalışıyordu.Bir önceki ödevde de bu sıkıntının aynısını yaşamıştık.Sorunu bulmamız tüm günümüze mal olsa da sonunda durumu

```
if (kontrol==true)
{
    dd = dugumAra(dg->data);
    dd->sehirDondur(dg2);
}
else
{
    while(true)
    {
        if (dg->data.compare(gecici->data) < 0)
        {
```

düzeltebildik.Bu kontrol işlemi gecici->data.compare(dg->data) iken bahsettiğimiz gibi yalnızca root için heapagaci yazdırabiliyorduk.Burada kıyasladığımız şeylerin yeri gerçekten önemliydi.

Sonuç

Üçüncü ödeve nazaran daha iyi başladığımız bu ödevde doğru kurduğumuzu düşündüğümüz algoritma yine az kalsın yukarıda bahsi geçen satırda bulunan hata sebebiyle eksik çalışıyor olacaktı. Sınıf tasarımlarında dugum ve ikili arama agaci için iki adet sınıf tasarlayıp bunları ulkeDugum ve Ulke sınıflarına kalıtım verdirdik.Ödev boyunca bu hatayı bulmamız uzun sürdüğü için tüm fonksiyonları ve hatta satırları gereken durumlarda debug dahi ederek taradığımız için ağaç veri yapılarını gerçek anlamda öğrendiğimizi düşünüyoruz.Bu anlamda bizim için oldukça verimli geçen bir süreç oldu.