

Programlama Laboratuvarı

Sıkıştırma Algoritmalarının Karşılaştırılması

Özge POYRAZ
Kocaeli Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği
180202025@kocaeli.edu.tr

Burak Can TEMİZEL
Kocaeli Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği
180202024@kocaeli.edu.tr

Özet—Bu çalışmada çeşitli sıkıştırma algoritmaları yeniden gerçekleştirilmiş, farklı uzunluk ve yapıdaki metinler üzerinde işleme tabi tutulup sıkıştırma oranları, sıkıştırma süreleri gibi veriler birbiri ile kıyaslanmıştır. LZ77 ve varyasyonları, Huffman kodlaması ile birlikte kendi Deflate implementasyonumuz da oluşturulmuştur.

Anahtar kelimeler— Veri, Sıkıştırma, LZ77, LZSS, Huffman Kodlaması, Deflate, Sıkıştırma Oranı, Karşılaştırma

I. GİRİŞ

Gerçekleştirdiğimiz proje, büyük verilerin çeşitli şekillerde sıkıştırılarak daha küçük veri setleri haline getirilmesiyle alakalıydı. Proje kapsamında bizden halihazırda var olan birtakım sıkıştırma algoritmalarının yeniden gerçekleştirilmesi ve kıyaslanması isteniyordu. Bu algoritmalar LZ77, Huffman kodlaması ve Deflate sıkıştırmasıydı.

İlk etapta veri sıkıştırmanın mantığını anlamak ve algoritmaların işleyişini incelemek çok önemliydi. Veri sıkıştırmanın çok farklı yolları bulunmaktaydı. Biz kayıpsız veri sıkıştırma gerçekleştirecektik. Algoritmaları gerçeklemeye geçmeden önce tek tek çalışma mantıklarını incelemeye ve öğrenmeye başladık. Temel olarak LZ77 ve varyasyonları bir metin üzerinde tekrar eden kısımlara geriye dönük referanslar verip sıkıştırma yapmayı hedefliyordu. Genel olarak bu mantık üzerinde farklı optimizasyonlar bulundurabiliyorlardı. Bizden direkt olarak LZ77'nin kendisi istendiği için temel konsepti gerçekleştirmeliydik. Huffman kodlaması ise bir metinde her karakterin tekrar adetine göre bir frekans tablosu oluşturmaya ve çok sayıda geçen karaktere daha kısa az sayıda geçen karaktere ise daha uzun kodlar vererek sıkıştırma gerçekleştirmeyi hedeflemekteydi. Deflate sıkıştırma ise 32 bit arama tamponu kullanan LZ77 türevidir. LZSS algoritması ile birlikte özel kurallara sahip bir Huffman kodlamasını sıkıştırılacak dosya üzerinde çeşitli veri bloklarıyla çok sayıda standarda göre kompleks bir biçimde uyguluyordu. Bu standartlar RFC1951[1] ismiyle Peter Deutsch tarafından 1996 yılında yayınlanmıştır. Algoritmaların gerçekleşmesi ve teknik detaylara Yöntemler ve Program Mimarisinde daha detaylı bir şekilde değinilecektir. Algoritmaların Karşılaştırılması kısmında çeşitli metinler üzerinde tek tek farklı algoritmalar ile sıkıştırılma yapıp elde edilen sonuçlar çeşitli grafikler ile sunulacaktır.

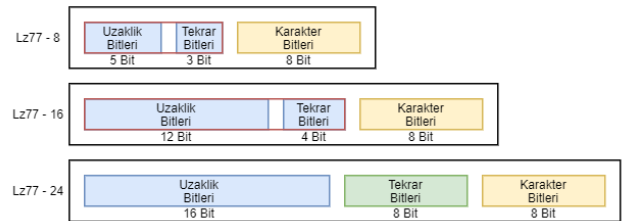
II. YÖNTEMLER VE PROGRAM MİMARİSİ

Bu kısımda programın farklı özelliklerini oluşturmak için kullandığımız araçlar ve yöntemler üzerinde durularak

ayrıntılı olarak bilgi verilecektir. Program mimarisi daha detaylı bir şekilde açıklanacaktır.

A. LZ77 Metin Sıkıştırma Algoritması

LZ77 algoritması metin üzerinde tekrar eden kısımları metnin geriye dönük kısmında işaret ederek sıkırtmayı gerçekleştirir. Bir kelime üzerinden somutlaştıracak olursak örneğin, “yaya” kelimesi eğer LZ77 ile sıkıştırılarak gösterilmek istenirse mecazi olarak şöyle bir gösterim yapılabilir. Sıkıştırma birimimiz <uzaklık, tekrar, karakter> olsun. Uzaklık her karakter için metnin daha önceki kısmında kendisiyle eşleşen ilk karakterin uzaklığını verirken, tekrar kendisinden sonra gelen diğer kaç karakterin eşleştiğini versin, karakter ise bu eşleşmeden hemen sonra gelen karakteri gösterebilir. Bu şekilde sıkıştırılmış gösterimimiz <0,0,y>, <0,0,a>, <2,2,\0> şeklinde olur. Fakat bu gösterim daha önce de bahsettiğimiz gibi mecazidir ve sıkıştırma yapmak için elverişsizdir. Sıkıştırma işlemini verimli hale getirmek için bit düzeyinde işlemler ile uzaklık ve tekrar değerlerini minimum veri yapıları üzerinde kodlayabilmemiz gerekli. LZ77 algoritması metin üzerindeki her karakter için bir gezinme işlemi yapar ve burada iki adet tampon verisi kullanır. Bunlar yukarıda bahsettiğimiz gibi uzaklık için arama tamponu ve tekrar için uzunluk tamponudur. Bu tamponların büyüklüğü ne kadar geriden ve ne kadar uzunlukta bir eşleşme saptayabileceğimizi belirler. Eğer sabit boyutlu bir sıkıştırma birimi tasarlırsak bu tamponların büyüklüğü direkt olarak sıkıştırma verimimizi etkileyecektir. Biz LZ77 algoritmasının implementasyonunu gerçekleştirirken kendimiz için 3 farklı varyasyon oluşturduk. LZ77-8, LZ77-16 ve LZ77-24 adını verdiğimiz sıkıştırma algoritmaları farklı büyüklüklerde sıkıştırma birimleri kullanmaktadırlar ve isimlerini uzaklık ve tekrar bitlerinin toplamından almaktadırlar.



Sıkıştırma Birimlerinin Yapısı

Bu şekilde her birimin boyutu ve sıkıştırma kapasitesi birbirinden farklılaşmış oldu. Örneğin eğer bir cümle üzerinde sıkıştırma yaparsanız LZ77-8 5 bitlik bir arama tamponu ile sizi 32 karakter geriye götürebilir ve 3 bitlik bir uzunluk tamponu ile 8 karakter eşleşmeyi tutabilir. Böylece karakter bitleriyle birlikte elinizde 2 baytlık bir sıkıştırma

birimi olmuş olur. Fakat LZ77-24 ile aynı sıkıştırma işlemini gerçekleştirecek olursanız 15 bit arama tamponu ile 32768 karakter geriye gidebilir ve 8 bit uzunluk tamponu ile 256 karakter eşleşmeyi tutabilirsiniz. Fakat bunun maliyeti her sıkıştırma birimi başına 4 bayt olmaktadır. Bu durum uzun ve tekrarın çok olduğu metinlerde daha yüksek tamponlu sıkıştırma birimleri kullanmayı mantıklı hale getirmektedir. Biz de deneysel olarak 3 farklı sıkıştırma birimi ile bu durumu gözlemlemeye karar verdik. İlerleyen kısımlarda algoritmaların kıyaslanması bu durumu daha detaylı bir şekilde inceleyeceğiz. LZ77 ile Sıkıştırmayı teknik olarak daha detaylı inceleyecek olursak işlem bahsettiğimiz bu birimlerin dinamik bir dizi üzerinden işlenmesiyle gerçekleşiyor. Sıkıştırılma yapılacak metin alınıyor. Burada karakter bitlerimiz 8 bit olduğu için metin üzerinde 8 bitlik veriler sıkıştırılmaktadır. Örneğin Unicode karakterler 1 bayttan fazla olduğu için 8 bitlik bloklar halinde farklı karakterlere indirgenerek sıkıştırılma işlemi gerçekleştirilmektedir. 8 bit olacak şekilde karakter okumaları yapılıyor ve her karakter için bir arama tamponu maksimum kapasitesi kadar geriden başlayarak eşleşme arıyor. Burada metnin dışına çıkması engelleniyor. Daha sonra bir eşleşme tespit edildiğinde uzunluk tamponu o noktadan başlayarak ardışık gelen karakterler üzerinde eşit mi kontrolü gerçekleştiriyor ve bu şekilde en uzun eşleşme tespit ediliyor. Bu işlem arama tamponu o anki karaktere gelene kadar devam ediyor ve bulunan en uzun eşleşme bir sıkıştırma birimine kaydedildikten sonra dizi dinamik olarak boyutlandırılıyor ardından bu birim dinamik diziye ekleniyor. Ekleme işlemi gerçekleştirilirken bazı sıkıştırma birimlerinde bit işlemleri gerçekleştiriliyor örnek olarak LZ77-16 da 16 bitlik bir değişken üzerinde uzaklık ve tekrar değerleri 12 ve 4 bit olmak üzere değişkeni paylaşmaktadırlar. Burada bit kaydırma ve maskeleye işlemleri ile değerler tek bir değişken üzerine kaydedilerek verimlilik artırılıyor. Bu işlem girdi metnindeki tüm karakterler için tekrarlanıyor. LZ77 sıkıştırma temel olarak bu şekilde gerçekleştiriliyor. Geri çözümleme işleminde ise aynı şekilde sabit genişlikli sıkıştırma birimleri okunarak bit işlemleriyle değişkenler üzerindeki uzaklık ve tekrar değerleri ayrıştırılarak metin eski haline getiriliyor. Bu kısım proje kapsamında olmadığı için raporda çok fazla detaylandırılmamıştır fakat programda mevcuttur. Geri çözümlenmiş çıktılar da sunulmaktadır.

B. Huffman Kodlaması

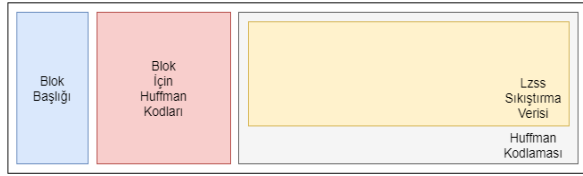
Projede gerçekleştirmemiz istenen Deflate sıkıştırma için LZ77 ile birlikte Huffman kodlamasını kullanmamız gerekliydi. Biz de daha deneysel bir çalışma için Huffman implementasyonunu da bağımsız bir şekilde programa ekledik. Tek blok halinde verilere Huffman kodlaması uygulayıp algoritmaların karşılaştırılması bölümünde bunu da kıyaslamaya katmak istedik. Öncelikle Huffman kodlamasının çalışma mantığından bahsedelim. Girdi metnimiz üzerindeki karakterler (Biz 8 bitlik blokları kullanmaktayız) metin üzerinde bulunma sıklıklarına göre bir frekans tablosuna yerleştirilir bu frekans tablosundan faydalanılarak bir Huffman ağacı oluşturulur ve bu ağaç üzerinden her karakter için yeni bir kod oluşturulur. Bu kodlar ile orijinal metin yeniden oluşturularak Huffman kodlaması yapılır. Daha detaylı teorik bilgi bizim de referans aldığımız Duke Üniversitesinin Huffman Coding: A CS2 Assignment[2] adlı yazısında bulunmaktadır. Huffman ağacının ve kodlarının oluşturulmasının teorik bilgisi üzerinde daha fazla durmayarak kullandığımız yöntemi

detaylı bir şekilde açıklayacağız. Huffman kodlamasını gerçekleştiren çeşitli veri yapılarını kullandık. Bir BST ağacı ile bir adet dinamik dizi kullanmamız gerekti. Öncelikle biz 8 bitlik verileri kodlayacağımız için 256 karakter kodu için bir frekans tablosu oluşturuyoruz. Bu tabloyu bir dizi ile ifade ediyoruz. Daha sonra metin üzerinden 8 bitlik karakter kodlarını okuyarak bu frekans tablosunda tekrarlanma adetlerini tespit ediyoruz ve tablo üzerinde güncelliyoruz. Bu karakterleri ve frekanslarını kullanarak Huffman düğümlerini oluşturuyoruz ve bu düğümleri dinamik dizimize atıyoruz. Diziyi frekanslara göre küçükten büyüğe sıralıyoruz. Diziden iki düğümü alıp üçüncü bir üst düğüm oluşturup frekans toplamalarını veriyoruz. Bunu da dinamik diziye ekliyoruz. Döngü bu şekilde devam ederek Huffman ağacını oluşturuyor. Bundan sonra girdi metin üzerinde karşılık gelen karakteri ağaç üzerinde bularak kodunu öğrenmek kalıyor. Bunun için de özyinelemeli bir arama fonksiyonu ile karşılık gelen karakteri kök düğümünden başlayarak ağaç içinde arıyoruz. Arama için bir kod tamponu kullanıyoruz. Karakteri bulana kadar ağaç içinde sola gittiğimizde bu tampona 0 sağa gittiğimizde 1 ekliyoruz. Bu şekilde kodumuz da oluşmuş oluyor. Girdi metnimizde tüm karakterlere karşılık gelen kodlar ile 0 ve 1'lerden oluşan yazı dizisi oluşturuyoruz. Fakat Deflate algoritmasında kullanılan Huffman kodlamasında bazı standartlar var. Bunun adı Cononical Huffman Code[3] olarak geçiyor. İkinci bir Huffman kodlaması implementasyonunda bu standartları da kodlamaya katarak daha verimli ve hızlı bir sıkıştırma yapabildiğimizi keşfettik. Biz de bu kodlama biçimini standart hale getirdik. Buradaki mantık ise kodların bit sayılarına göre sıralandıktan sonra yukarıdaki referans kaynaktaki gibi çeşitli bit işlemleri ile yeniden şekillendirilmesine dayanıyor. Bu işlemden sonra elde ettiğimiz kodlara biz standart kodlar adını verdik. Nihayetinde Huffman kodlamasından her karakter için 2 türlü kod elde ettik ve bunlar ile girdi metnini yeniden kodladık fakat elimizde 0 ve 1'lerden oluşan bir yazı dizisi vardı. Biz sıkıştırma yapmak istediğimiz için burada bulunan diziyi binary olarak kodlamamız gerekiyordu. Bunu gerçekleştiren dizi üzerindeki her 8 elemanı ikilik bir sayıymış gibi alıp onluk tabana çevirdik ve bir karakter değişkeninde depoladık daha sonra dosyaya yazdırma sırasında ise bu karakterleri yazdırarak sıkıştırma işlemini tamamladık. Bu işlemi gerçekleştiren fonksiyonun algoritması raporda bulunmaktadır.

C. Deflate

Daha önce de bahsettiğimiz gibi Deflate algoritması bir LZ77 Türevi olan LZSS algoritması ve Huffman kodlamasının bir birleşiminden oluşmaktaydı. Fakat RFC1951 Standardına baktığımızda çok kompleks bir yapıda olduğunu gördük. Deflate algoritması png dosyalarından jar dosyalara hatta web sitelerinin bile sıkıştırılmasına kadar çok geniş bir alanda kullanılmaktaydı. Java gibi yüksek seviye dillerin kendi Deflate sınıfının olduğunu ve çoğu dil için kütüphanelerinin yazıldığını gördük. Zlib içindeki deflate.c kaynak kodunu incelediğimizde aynı şekilde implementasyonu gerçekleştiremeyeceğimizi anladık. Projedeki ilk sorumuz da burada başladı. Bu kısma Geliştirme Aşamasında Karşılaşılan Problemler Kısımında tekrar değinilecektir. Bu noktadan sonra Deflate algoritmasını oldukça basite indirgeyerek oluşturduğumuz diğer algoritmaları da kullanarak gerçeklemeye karar verdik. Gerçek standartlara göre sadık kaldığımız ve değiştirdiğimiz

bazı noktalar oldu. Öncelikle Deflate sıkıştırmanın mantığından bahsedelim. Sıkıştırılacak veri bloklar halinde bulunur her bloğun bir başlığı vardır. Bu başlık bloğun ardılarının devamıyla ve blok içindeki sıkıştırma modu ile ilgili bilgiler verir. Deflate'de bir blokta yalnızca tek bir mod bulunabilir. Toplamda üç adet mod bulunmaktadır. Mod 1 blok içerisinde sıkıştırma olmadığını belli eder bazı durumlarda veriyi sıkıştırmak maliyeti artırabileceği için bu bloklar mevcuttur ve 64K uzunluk ile sınırlandırılmışlardır. Diğer bloklarda herhangi bir uzunluk limiti bulunmamaktadır. mod 2 blok içerisinde LZSS ile sıkıştırılmış olan veri statik Huffman kodlaması ile sıkıştırılır. Mod 3 blok içerisinde ise mod 2 den farklı olarak Dinamik Huffman kodlaması kullanılır. Biz kendi basit Deflate implementasyonumuzu sadece mod 2'yi barındıracak şekilde tek bloklu 32K arama tamponlu LZSS, tek bloklu 12 bit arama tamponlu ve çoklu ama sabit bloklu 32K arama tamponlu LZSS barındıracak şekillerde tasarladık. Fakat verimsizlik ve hatalar sebebiyle iki tanesini devre dışı bırakıp implementasyona tek blok , 32K LZSS barındıran Deflate üzerinden devam ettik.



Deflate Bloğu

Deflate bloğumuz yukarıda verildiği gibi ve gerçekte olduğu gibi bir başlık barındırmakta. Fakat tek blok olduğu için başlık sabit olarak blok sonu kodunu işaret etmektedir. Ayrıca yeniden bir LZSS implementasyonu yapmayarak LZ77 üzerinde gerekli çıktı optimizasyonunu gerçekleştirdik. Gerçek Deflate algoritmasında olduğu gibi 15 bit arama tamponuna ve 8 bit tekrar tamponuna sahip bir LZSS algoritması kullandık. Cononical Huffman Kodları ile sıkıştırılmış LZSS verisini yeniden kodladık. Sıkıştırılmış veriden önce Huffman Kodlarını yerleştirdik. Fakat gerçek Deflate algoritmasından farklı olarak bu veriler de Huffman kodlamasıyla tekrardan sıkıştırılmadı.

III. ALGORİTMALARIN KARSILASTIRILMASI

Bu kısımda algoritmalar çeşitli metinler ile karşılaştırılacaktır. Kullanılan metinler yerine göre değişebilmekle birlikte genel olarak Project Gutenberg Top 100[4] kitaplarından oluşmaktadır.

	LZ77-4	LZ77-16	LZ77-24
Tekerleme – Gerçek Boyut: 122 Bayt			
Sıkışmış Boyut	104 Bayt	196 Bayt	196 Bayt
Sıkıştırma Oranı	0,852459	1,606557	1,606557
Geçen Zaman	0,013000 saniye	0,010000 saniye	0,011000 saniye
Kitap, Pride And Prejudice by Jane Austen – Gerçek Boyut: 799738 Bayt / 781 KB			
Sıkışmış Boyut	806190 Bayt	579088 Bayt	419372 Bayt
Sıkıştırma Oranı	1,008068	0,724097	0,524387
Geçen Zaman	0,045000 saniye	0,815000 saniye	4,349000 saniye

	LZ77-4	LZ77-16	LZ77-24
Shakespeare'in Tüm Eserleri, Gerçek Boyut: 5458199 Bayt/ 5331 KB			
Sıkışmış Boyut	5646542 Bayt	4187520 Bayt	3099944 Bayt
Sıkıştırma Oranı	1,034506	0,767198	0,567943
Geçen Zaman	0,237000 saniye	6,286000 saniye	33,474000 saniye

Tabloyu inceleyecek olursak LZ77 kısmında bahsettiğimiz gibi farklı varyasyonların farklı yerlerde daha iyi sıkıştırma yapabildiğini net gözlemleyebiliyoruz. Örneğin kısa ve tekrarların çok olduğu bir tekerlemede LZ77-4 düşük birim boyutu ile maliyeti azaltırken boyutun arttığı yerlerde sıkıştırma sağlayamıyor. Fakat yüksek bitlerle ifade edilen tamponlara sahip LZ77-24 ise uzun metinlerde oldukça yüksek sıkıştırmalar gerçekleştiriyor. Fakat arama tamponunun boyutu arttığı için algoritmanın çalışma süresi de artıyor.

	LZ77-24	Huffman	Deflate
Kitap, Alice's Adventures in Wonderland– Gerçek Boyut: 174481 Bayt /170 KB			
Sıkışmış Boyut	96616 Bayt	104182 Bayt	84052 Bayt
Sıkıştırma Oranı	0,553734	0,597097	0,481726
Geçen Zaman	0,843000 saniye	18,640000 saniye	6,560000 saniye
Kitap, Peter Pan – Gerçek Boyut: 290755 Bayt / 284 KB			
Sıkışmış Boyut	171584 Bayt	170777 Bayt	147703 Bayt
Sıkıştırma Oranı	0,590133	0,587357	0,507998
Sıkıştırma Zamanı	1,694000 saniye	35,403000 saniye	19,944000 saniye
Kitap, The Adventures of Tom Sawyer – Gerçek Boyut: 428104 Bayt / 418 KB			
Sıkışmış Boyut	261700 Bayt	252147 Bayt	223976 Bayt
Sıkıştırma Oranı	0,611300	0,588985	0,523181
Geçen Zaman	2,691000 saniye	56,815000 saniye	45,097000 saniye

Burada ise farklı algoritmalarla sıkıştırmalar gerçekleştirdik. Deflate implementasyonumuz çok yüksek sıkıştırma oranları sağlamasa da kendini oluşturan algoritmalarından daha iyi bir sıkıştırma gerçekleştirdi. Burada Huffman kodlaması kendi başına tüm metine tek bir seferde uygulandığı için çalışma süresi oldukça hızlı artmaktadır. Fakat Deflate içinde kullanımda zaten halihazırda LZSS ile sıkıştırılmış veriyi uygulandığı için çalışma süresi Deflate için daha az olmaktadır. Bu tablolardan algoritmalar ile ilgili çok sayıda bilgi edinilebilmektedir.

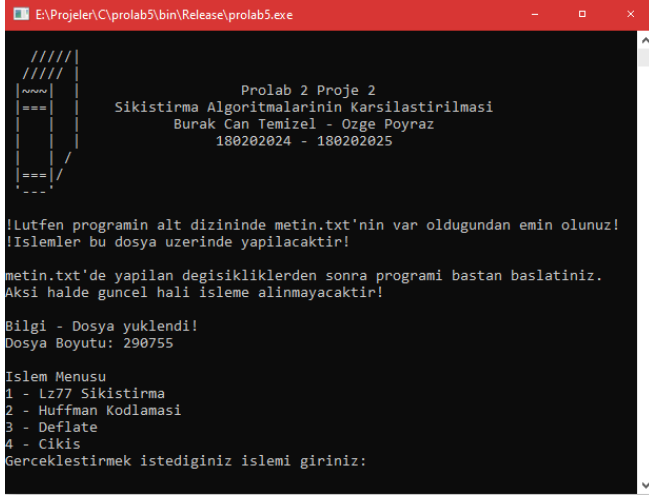
IV. GELİŞTİRME ORTAMI

Projeyi C Programlama dilinde Windows işletim sistemi üzerinde gerçekleştirirken, repl.it bulut geliştirme ortamını kullandık. Son derleme işlemini Codeblocks

idesinde yerleşik Mingw ile yaptık. Son proje dosya yapısı Codeblocks projesi formatındadır. Derlenirken optimizasyon parametreleri verilmiştir.

V. PROGRAMIN GENEL YAPISI VE TASARIMI

Program bir konsol uygulaması olarak tasarlandı. Varsayılan olarak "metin.txt" klasörünü okuyor ve karşınıza çeşitli menüler getiriyor. Menülerde seçimler yaparak çeşitli sıkıştırma işlemlerini gerçekleştirebiliyorsunuz ve gerekli çıktıları ana dizinde oluşturuyor.



```
E:\Projeler\C\prolab5\bin\Release\prolab5.exe

Prolab 2 Proje 2
Sıkıştırma Algoritmalarının Karşılaştırılması
Burak Can Temizel - Ozge Poyraz
180202024 - 180202025

!Lutfen programın alt dizininde metin.txt'nin var olduğundan emin olunuz!
!İşlemler bu dosya üzerinde yapılacaktır!

metin.txt'de yapılan değişikliklerden sonra programı baştan başlatınız.
Aksi halde güncel hali işleme alınmayacaktır!

Bilgi - Dosya yüklendi!
Dosya Boyutu: 290755

İşlem Menuşu
1 - Lz77 Sıkıştırma
2 - Huffman Kodlaması
3 - Deflate
4 - Çıkış
Gerçekleştirmek istediğiniz işlemi giriniz:
```

VI. PROGRAMIN ÇALIŞTIRILMASI VE KULLANILMASI

Detaylı görseller ile adım adım kullanım raporun en sonunda mevcuttur.

Program 2_180202024_180202025 içerisindeki Prolab5SıkıştırmaAlgoritmaları.exe ya da aynı klasör içerisindeki bin klasörünün altındaki release klasörü içerisinde bulunan prolab5.exe üzerinden direkt olarak çalıştırılabilir. Bu exe dosyalarını herhangi bir yere taşıyarak da programı çalıştırabilirsiniz. Program metin.txt dosyasından girdi okur. İşlevsel hale gelmesi için yanında metin.txt'nin bulunması gerekir aksi halde size uyarı verecektir.

Proje üzerinden derleme yapılmak istenirse Projenin dosya yapısı Codeblocks projesi formatındadır. Prolab5.cbp dosyası ile projeyi codeblocks ortamında açabilirsiniz. Mingw entegreli bir codeblocksta tüm ayarları yapılabildiği bir halde direkt olarak derleyebilirsiniz.

Direkt olarak mevcut exe dosyası üzerinden ya da kendi derlediğiniz exe dosyası üzerinden programı çalıştırınız. Eğer program dizininde metin.txt yok ise size uyarı verecektir. Bu dosyanın olduğundan emin olunuz. Ayrıca metin.txt'de değişiklik yaptığınızda program yeniden başlatınız. Program içerisinde tüm işlemler detaylı bir şekilde açıklamalarıyla birlikte listelenmektedir. Yapmak istediğiniz işlemde önce açıklamayı okuyup daha sonra işlemin numarasını giriniz ve işlemin gerçekleşmesini bekleyiniz. İşlem sonucunda exe ile aynı dizinde çıktılar oluşacaktır.

VII. GELİŞTİRME AŞAMASINDA KARŞILAŞILAN PROBLEMLER

Proje boyunca karşılaştığımız en büyük problem, Deflate sıkıştırmanın yapısal olarak oldukça kompleks olması ve direkt implementasyonunu yapamamıştık. Bu problemi aşabilmek için çok sayıda dokümantasyon inceledik. Çeşitli kaynak kodlarını inceledik fakat nihai olarak basite indirgeyerek bu sorunu aşabildik. Çalışma mantığını öğrenmek bizim için daha önemli bir hale geldi. İkinci yaşadığımız problem ise proje kapsamında olmamasına rağmen sonuçlarını merak ettiğimiz için yalnız başına uyguladığımız Huffman kodlamasının içerdiği özyinelemeli fonksiyonlar yüzünden çok uzun süren işlemler gerçekleştirmesi idi. Deflate içindeki kodlamadan farklı olarak sıkıştırılmış bir metne değil de direkt girdinin kendisine uygulanması çalışma süresini daha da uzatıyordu. Özellikle stringler üzerinde yaptığımız stringe ekleme işleminin ve ağaç içerisinde özyinelemeli kod bulma işleminin çok zaman aldığını fark ettik. Kodlar için sabit bir dizi oluşturduk ve string eklemesini memcpy() fonksiyonu ve pointer aritmetiğiyle yeniden oluşturarak çalışma zamanını hızlandırdık.

VIII. DENEYSEL SONUÇLAR

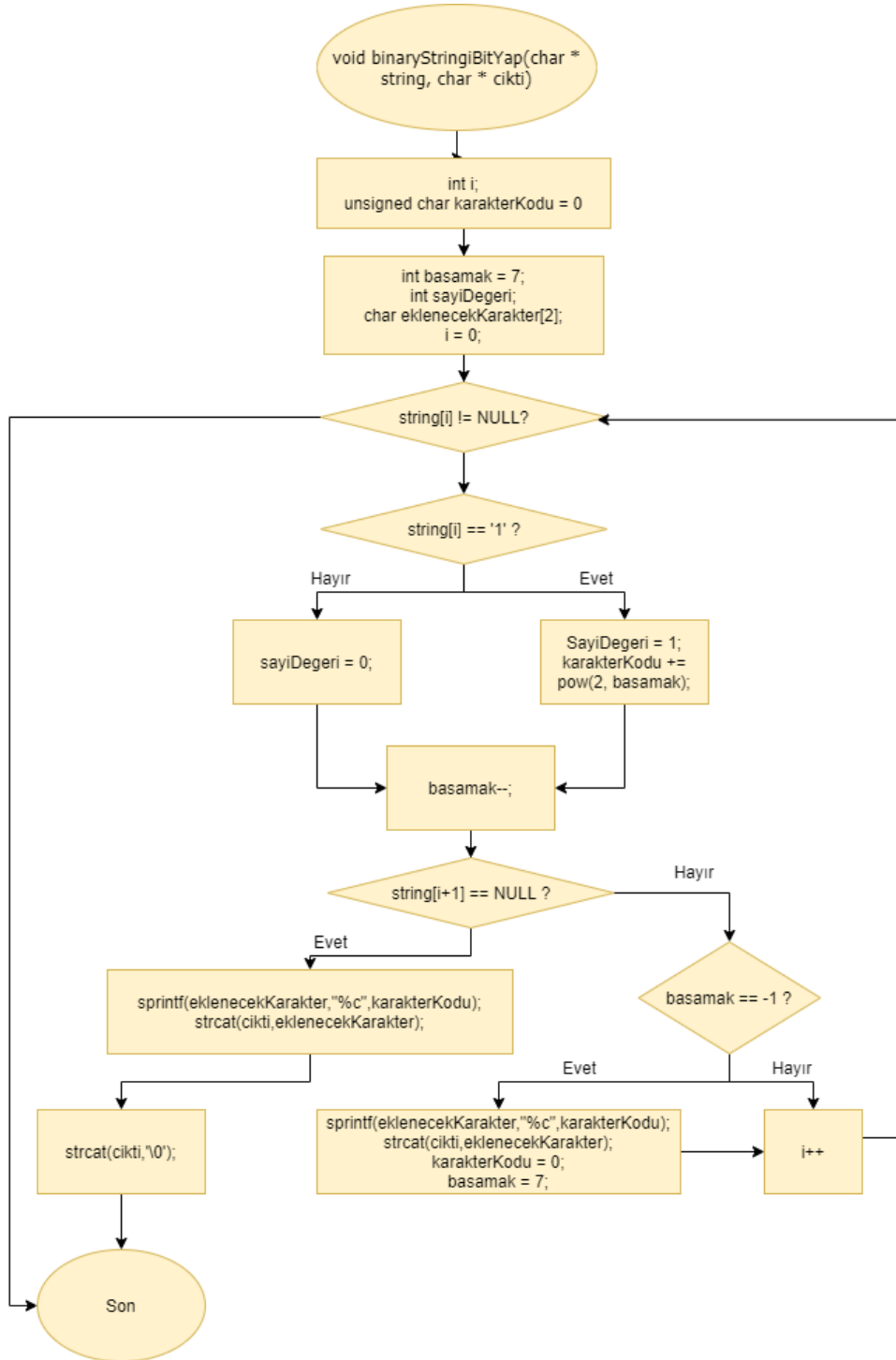
Projenin gerçekleşmesiyle birlikte çok çeşitli sıkıştırma algoritmalarının çalışma prensiplerini de öğrenmiş olduk. Ayrıca kayıplı ve kayıpsız veri sıkıştırma gibi çok farklı konularda bilgiler edindik. Bunlardan yola çıkarak oluşturduğumuz implementasyonları sürekli test ettik ve birbiri ile kıyasladık. Bu bize çeşitli grafikler ve sonuçlar verdi. Ayrıca proje kapsamında olmayan LZ77 geri çözümleme, tek başına Huffman kodlama gibi uygulamaları yaparak sonuçlarını da gözlemledik. Deflate implementasyonunu gerçekleştirirken farklı yapılar denedik ve farklı hatalar alarak sonuçlarını gözlemledik.

IX. SONUC

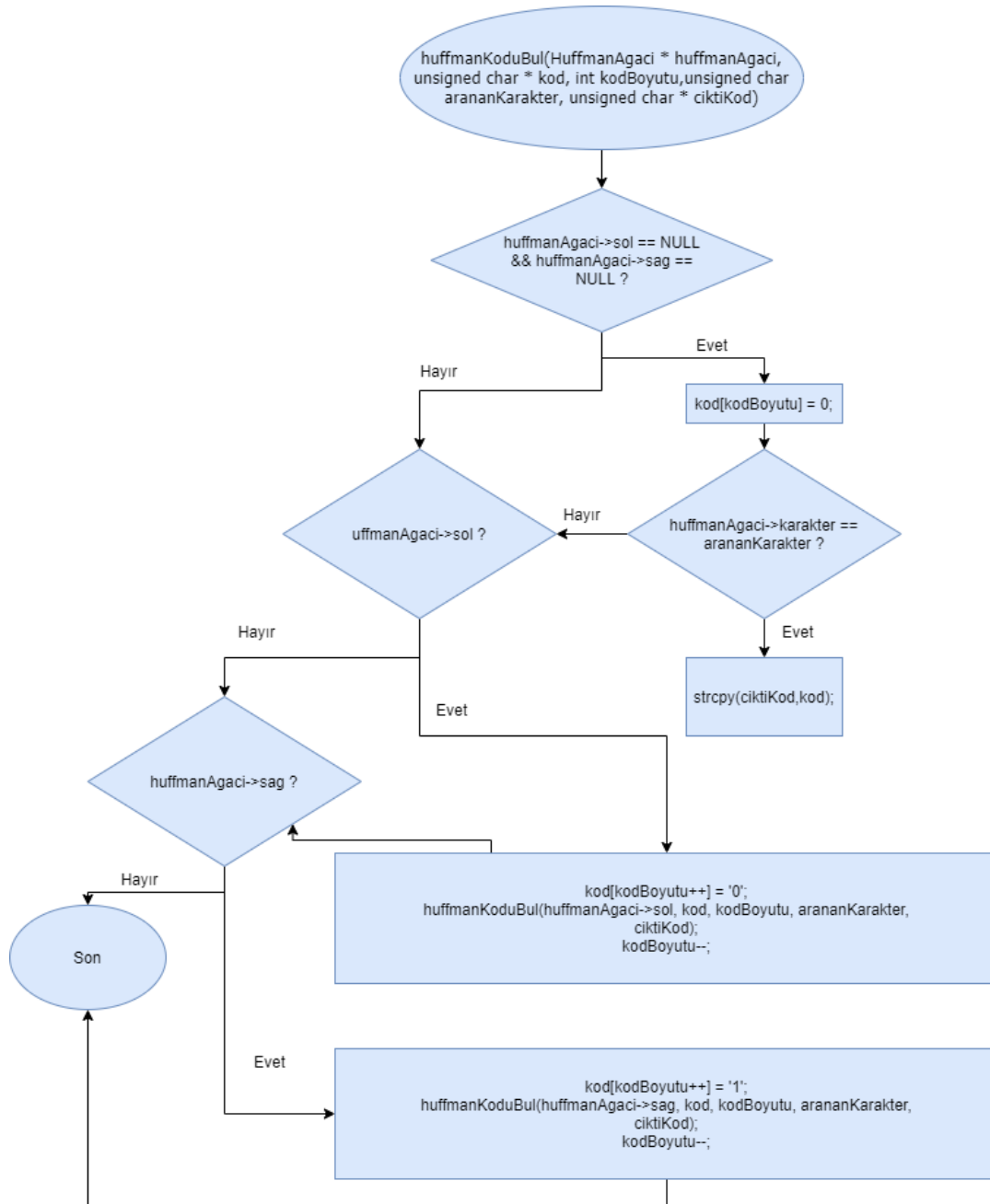
Bu projeyi gerçekleştirerek veri sıkıştırmanın ve çeşitli sıkıştırma algoritmalarının çalışma mantığını detaylıca inceleme, öğrenme ve uygulama fırsatı elde ettik. Bunları oluşturup kendi aralarında karşılaştırarak birtakım veriler elde ettik.

X. KAYNAKLAR

- (1) <https://tools.ietf.org/html/rfc1951>
- (2) <https://www2.cs.duke.edu/csed/poop/huff/info/>
- (3) https://pineight.com/mw/index.php?title=Canonical_Huffman_code
- (4) <https://www.gutenberg.org/browse/scores/top>

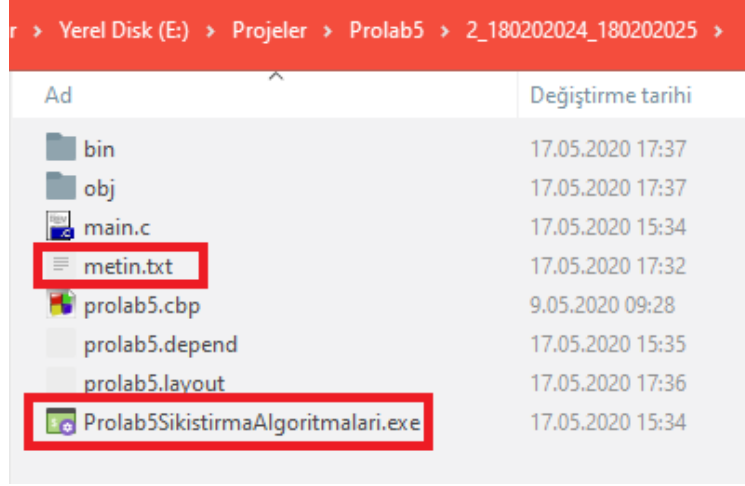


Char binary diziyi 8'li bloklar halinde karaktere çeviren fonksiyonun algoritması



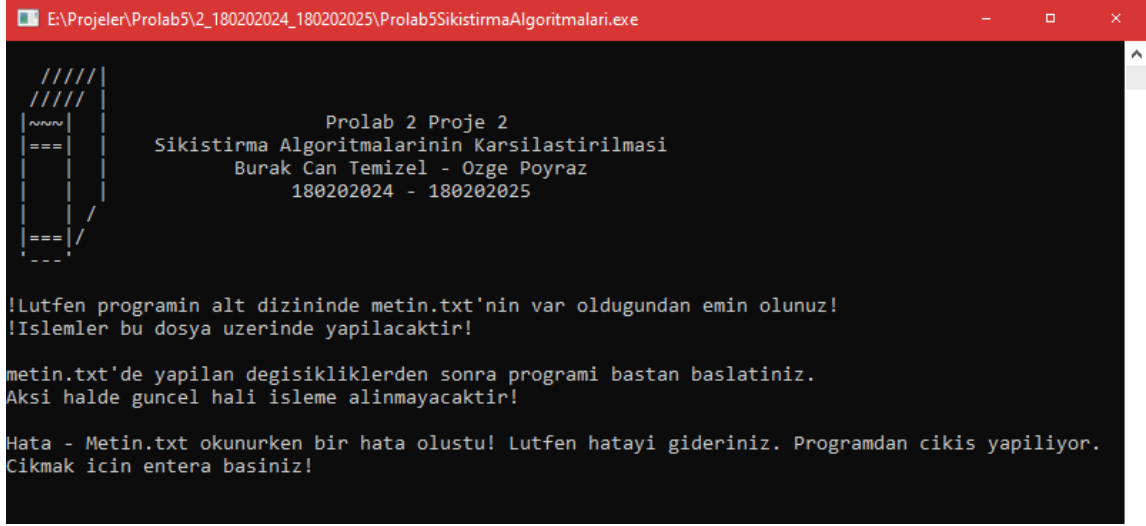
Özyinelemeli olarak Huffman kodunu bulan ve tampona aktaran fonksiyon

PROGRAMIN GÖRSELLERLE DETAYLI ADIM ADIM ÇALIŞTIRILMASI

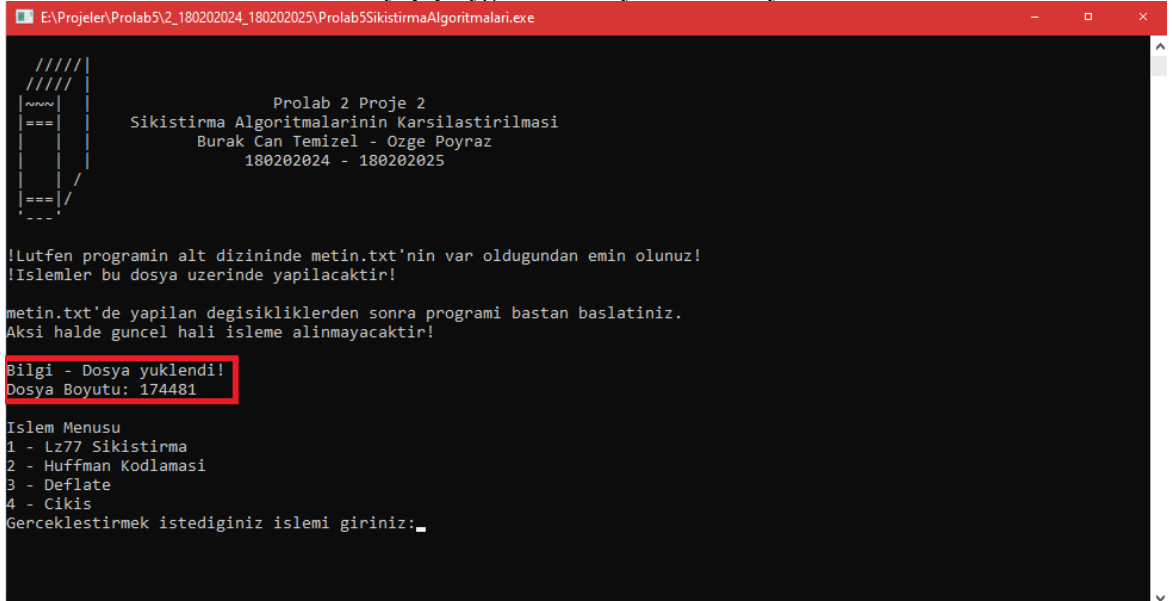


Programı gösterildiği gibi ana dizin içerisinde bulunan exe dosyası ile çalıştırabilirsiniz. (isteğe bağlı olarak bin/releases içerisindeki prolab5.exe' de kullanılabilir.).

Programın alt dizininde metin.txt'nin aynı isim ve uzantıda bulunması gereklidir. Aksi halde size şu şekilde uyarı verir.



Sıkıştırma yapmak istediğiniz veriyi metin.txt'nin içerisine kopyalayınız. Ya da ismini metin.txt haline getirip aynı dizine atınız. Programın işleyişini göstermek için raporda da kullandığımız Project Gutenbergteki Alice in Wonderland txt kitabının adını metin.txt yapıp uygulamaya ile aynı dizine atıyorum.



Bu şekilde dosya yüklenecektir ve size boyut bilgisi verilecektir. Bundan sonra karşınıza işlem menüsü gelmektedir. Burada dikkat edilmesi gereken nokta program çalıştığı andaki metin.txt'yi referans alır o yüzden dosyada bir değişiklik gerçekleşirse programı yeniden başlatmalısınız.

1 yazıp Programı LZ 77 Sıkıştırma moduna alalım.

```
E:\Projeler\Prolab5\2_180202024_180202025\Prolab5SikistirmaAlgoritmaları.exe
Aksi halde guncel hali isleme alınmayacaktır!

Bilgi - Dosya yuklendi!
Dosya Boyutu: 174481

Islem Menu
1 - Lz77 Sikistirma
2 - Huffman Kodlamasi
3 - Deflate
4 - Cikis
Gerceklestirmek istediginiz islemi giriniz:1

Bu programda 3 farkli lz77 konfigurasyonu bulunmaktadır.
Farkli dosyalar icin farkli optimizasyonlara sahiptirler bu yuzden "hepsini" calistirmaniz tavsiye edilir.
Algoritmaların calisma zamanlari 6 mb txt uzerinde yaklasik olarak lz77-8 0.5 sn, lz77-16 8 sn, lz77-24 40 sn dir.
1 - lz77-16 12 uzaklik biti, 4 tekrar biti kullanir.Ortalama sikistirma yapar fakat hizlidir.
2 - lz77-8 5 uzaklik biti, 3 tekrar biti kullanir. Kisa metin ve ufak dosyaların sikistirilmesi icin idealdir.
3 - lz77-24 15 uzaklik biti, 8 tekrar biti kullanir.Yuksek oranda sikistirma yapar.(Varsayilan)
4 - Tum konfigurasyonlari calistir.
Gerceklestirmek istediginiz islemi giriniz:
```

Karşımıza bu şekilde bir menü geliyor. Raporda da bahsettiğimiz gibi 3 varyasyonda LZ77 algoritması mevcut burdan hangisiyle işlem yapmak istiyorsak onu seçiyoruz. Menü üzerinde hangisinin ne amaçla kullanıldığı yazıyor. Eğer yalnızca biri denenecekse LZ77-24 seçip yani 3 yazarak devam edebilirsiniz. Deneyisel olarak farkı görmek için biz üçünü de devrede bıraktık. Hepsini işleme sokmak için 4 yazıp devam edin. Diğerleri içinde ilgili numarayı kullanabilirsiniz. Bu aşamada ben 4 yazarak devam ediyorum ve tüm LZ77 varyasyonlarını çalıştırıyorum.

```
Bu programda 3 farkli lz77 konfigurasyonu bulunmaktadır.
Farkli dosyalar icin farkli optimizasyonlara sahiptirler bu yuzden "hepsini" calistirmaniz tavsiye edilir.
Algoritmaların calisma zamanlari 6 mb txt uzerinde yaklasik olarak lz77-8 0.5 sn, lz77-16 8 sn, lz77-24 40 sn dir.
1 - lz77-16 12 uzaklik biti, 4 tekrar biti kullanir.Ortalama sikistirma yapar fakat hizlidir.
2 - lz77-8 5 uzaklik biti, 3 tekrar biti kullanir. Kisa metin ve ufak dosyaların sikistirilmesi icin idealdir.
3 - lz77-24 15 uzaklik biti, 8 tekrar biti kullanir.Yuksek oranda sikistirma yapar.(Varsayilan)
4 - Tum konfigurasyonlari calistir.
Gerceklestirmek istediginiz islemi giriniz:4

Bilgi - Lz77-16 ile sikistirma yapiliyor lutfen bekleyiniz...
Bilgi - Lz77-16 ile sikistirma islemi tamamlandi.
Bilgi - Lz77-16 ile geri cozumleme islemi yapiliyor lutfen bekleyiniz...
Bilgi - Lz77-16 ile geri cozumleme islemi tamamlandi.
Bilgi - Lz77-16 ciktilari olusturuldu..

Lz77-16 Sikistirma Sonuclari:
Dosyanin Gercek Boyutu: 174481
Sikistirilmis Boyutu : 127508
Geri Cozumlenmis Boyutu: 174481
Sikistirma Orani: 0,730784
Gecen zaman : 0,536000

Bilgi - Lz77-8 ile sikistirma yapiliyor lutfen bekleyiniz...
Bilgi - Lz77-8 ile sikistirma islemi tamamlandi.
Bilgi - Lz77-8 ile geri cozumleme islemi yapiliyor lutfen bekleyiniz...
Bilgi - Lz77-8 ile geri cozumleme islemi tamamlandi.
Bilgi - Lz77-8 ciktilari olusturuldu..

Lz77-8 Sikistirma Sonuclari:
Dosyanin Gercek Boyutu: 174481
Sikistirilmis Boyutu : 182422
Geri Cozumlenmis Boyutu: 174481
Sikistirma Orani: 1,045512
Gecen zaman : 0,021000

Bilgi - Lz77-24 ile sikistirma yapiliyor lutfen bekleyiniz...
Bilgi - Lz77-24 ile sikistirma islemi tamamlandi.
Bilgi - Lz77-24 ile geri cozumleme islemi yapiliyor lutfen bekleyiniz...
Bilgi - Lz77-24 ile geri cozumleme islemi tamamlandi.
Bilgi - Lz77-24 ciktilari olusturuldu..

Lz77-24 Sikistirma Sonuclari:
```

Program algoritmaların sonuçlarıyla ilgili çeşitli bilgiler veriyor. Bu bilgileri konsol üzerinden alabilirsiniz sıkıştırma oranı, sıkıştırma zamanı gibi bilgiler mevcut. Sıkıştırılmış dosyalar ise exe ile aynı dizinde oluşuyor.

Bu şekilde huffman içinde bir menü gelecek. Burada deflate içerisindeki gibi standartlaştırılmış kodlarla çalışan için 1'i klasik huffman için 2'yi seçmeniz gerekiyor. Standartlaştırılmış hali hem daha hızlı çalıştığı için hem de daha iyi sıkıştırma yaptığı için onu seçmenizi tavsiye ediyoruz. İşlemin bu adımını da 1 numarayı seçerek devam ettiriyoruz. İşlem başlayacak ve bir müddet sürecektir.

```
metin.txt üzerinde direkt olarak yalnızca huffman kodlaması uygulanacaktır.
Lz77 ve huffman için deflate sıkıştırmasını kullanınız.
Dikkat!! - Büyük dosyalar üzerinde huffman kodlaması gerçekleştirmek uzun işlem süreleri alacaktır.
Lütfen küçük dosyalar ile deneyiniz!
Deflate tipi huffman kodlarının kullanılması daha iyi ve hızlı sıkıştırma sonuçları verecektir ilk seçeneği tercih ediniz.
1 - Huffman ağacı oluşturun ve kodları standartlaştırın(deflate tipi). Standart kodlar ile dosyayı sıkıştırın.(Varsayılan)
2 - Huffman ağacı ve kodları oluşturun. Kodlar ile dosyayı sıkıştırın.
Gerçekleştirmek istediğiniz işlemi giriniz:1

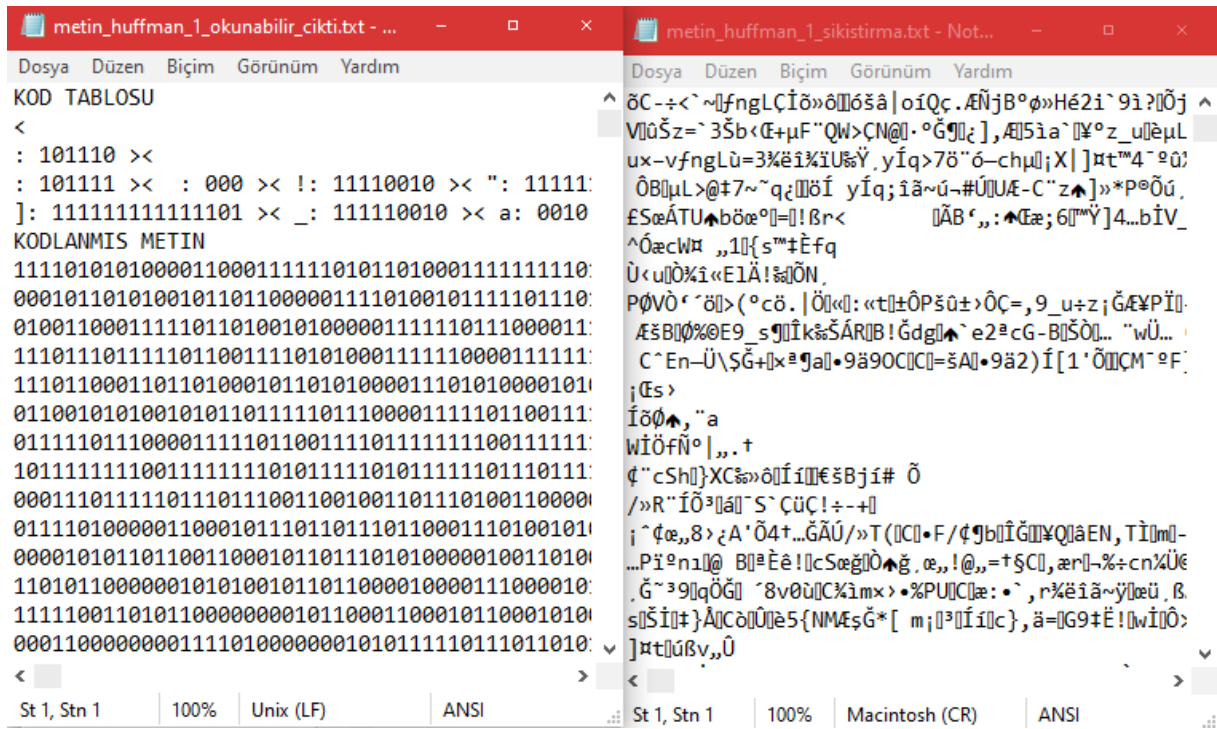
Bilgi - Standart Huffman Kodları Oluşturuluyor...
Bilgi - Metin Huffman Standart Kodlaması Haliye Getiriliyor. Lütfen Bekleyiniz...
Bilgi - Standart Huffman Kodlaması Bit Düzeyinde Çıktıya İseniyor. Lütfen Bekleyiniz...

Standart Huffman Sıkıştırma Sonuçları:
Dosyanın Gerçek Boyutu: 174481
Sıkıştırılmış Boyutu : 104182
Sıkıştırma Oranı: 0,597097
Gecen zaman : 18,868000
```

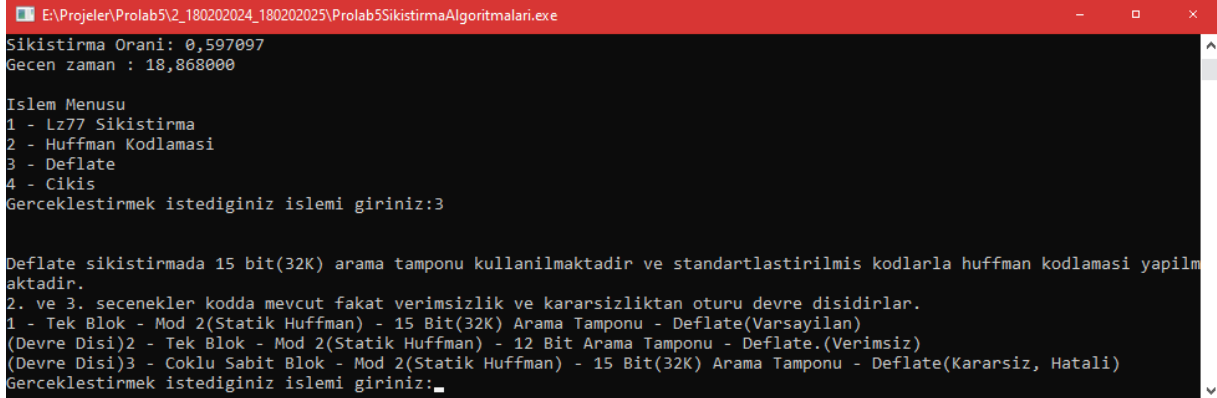
İşlem tamamlandıktan sonra yine aynı şekilde bilgileri ekrandan alabilirsiniz. Ve Çıktı dosyalarına ulaşabilirsiniz.

Ad	Değiştirme tarihi	Tür	Boyut
bin	17.05.2020 17:37	Dosya klasörü	
obj	17.05.2020 17:37	Dosya klasörü	
main.c	17.05.2020 15:34	C Source File	69 KB
metin.txt	17.05.2020 15:48	Metin Belgesi	171 KB
metin_huffman_1_okunabilir_cikti.txt	17.05.2020 18:52	Metin Belgesi	819 KB
metin_huffman_1_sikistirma.txt	17.05.2020 18:52	Metin Belgesi	102 KB
metin_lz77-8_GeriCozumleme.txt	17.05.2020 18:46	Metin Belgesi	171 KB
metin_lz77-8_Sikistirma.txt	17.05.2020 18:46	Metin Belgesi	179 KB
metin_lz77-16_GeriCozumleme.txt	17.05.2020 18:46	Metin Belgesi	171 KB
metin_lz77-16_Sikistirma.txt	17.05.2020 18:46	Metin Belgesi	125 KB
metin_lz77-24_GeriCozumleme.txt	17.05.2020 18:46	Metin Belgesi	171 KB
metin_lz77-24_Sikistirma.txt	17.05.2020 18:46	Metin Belgesi	95 KB
prolab5.cbp	9.05.2020 09:28	project file	2 KB
prolab5.depend	17.05.2020 15:35	DEPEND Dosyası	1 KB
prolab5.layout	17.05.2020 17:36	LAYOUT Dosyası	1 KB
Prolab5SikistirmaAlgoritmaları.exe	17.05.2020 15:34	Uygulama	32 KB

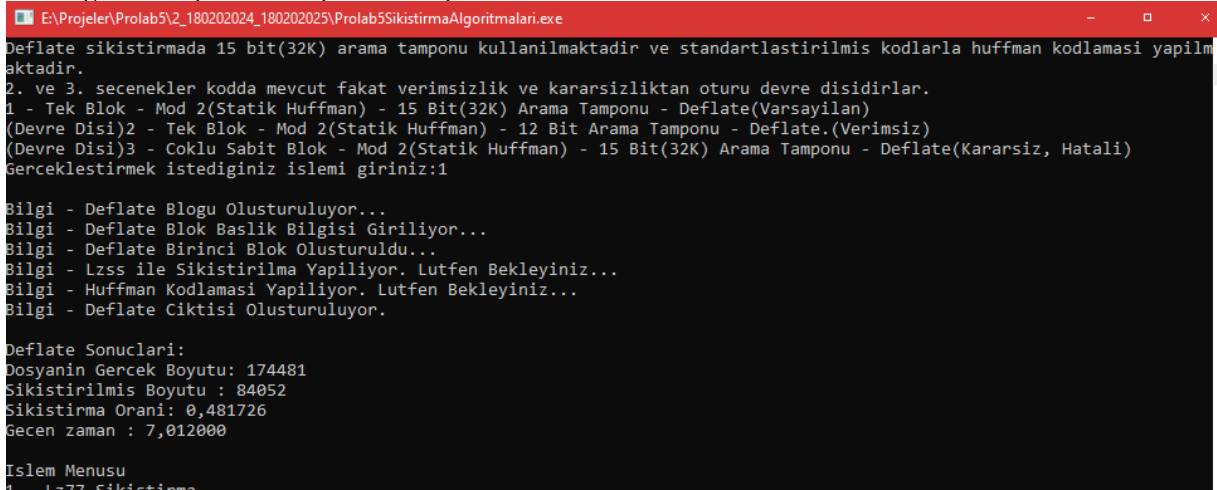
Bu şekilde yeni çıktılar eklenecektir. Huffman kodlamasını anlamlı çıktı olarak ve sıkıştırılmış çıktı olarak görüntüleyebilirsiniz. Anlamlı çıktıyı görebilmek için huffmanı 1. Seçenek ile çalıştırmalısınız.



Daha sonra tekrar menüye yönlendirileceksiniz. Buradan 3. Seçeneği seçerek deflate menüsüne gidiyoruz.



Burada zaten tek mod aktif halde onu 1 ile seçerek devam ediyoruz. İşlemler gerçekleşecek. Bekliyoruz işlem bittiğinde bilgi ekranı çıkacak ve çıktılar oluşacak.



Ad	Değiştirme tarihi	İür	Boyut
bin	17.05.2020 17:37	Dosya klasörü	
obj	17.05.2020 17:37	Dosya klasörü	
main.c	17.05.2020 15:34	C Source File	69 KB
metin.txt	17.05.2020 15:48	Metin Belgesi	171 KB
metin_deflate_sikistirma_1.txt	17.05.2020 18:59	Metin Belgesi	83 KB
metin_huffman_1_okunabilir_cikti.txt	17.05.2020 18:52	Metin Belgesi	819 KB
metin_huffman_1_sikistirma.txt	17.05.2020 18:52	Metin Belgesi	102 KB
metin_lz77-8_GeriCozumleme.txt	17.05.2020 18:46	Metin Belgesi	171 KB
metin_lz77-8_Sikistirma.txt	17.05.2020 18:46	Metin Belgesi	179 KB
metin_lz77-16_GeriCozumleme.txt	17.05.2020 18:46	Metin Belgesi	171 KB
metin_lz77-16_Sikistirma.txt	17.05.2020 18:46	Metin Belgesi	125 KB
metin_lz77-24_GeriCozumleme.txt	17.05.2020 18:46	Metin Belgesi	171 KB
metin_lz77-24_Sikistirma.txt	17.05.2020 18:46	Metin Belgesi	95 KB
prolab5.cbp	9.05.2020 09:28	project file	2 KB
prolab5.depend	17.05.2020 15:35	DEPEND Dosyası	1 KB
prolab5.layout	17.05.2020 17:36	LAYOUT Dosyası	1 KB
Prolab5SikistirmaAlgoritmaları.exe	17.05.2020 15:34	Uygulama	32 KB

Bu şekilde son çıktımızda oluşacaktır.Çıktı dosyaları yalnızca işlem yaptığınız zaman güncellenir. Yeni bir metine geçtiğinizde ilgili işlemi seçtiğinizden emin olunuz yoksa eski bir çıktı orada kalmış olabilir. Proje içerisinde raporda karşılaştırma tablolarında kullandığımız metin belgeleri RapordaKullanilanTestMetinleri klasörü altında bulunmaktadır. Çok sayıda çıktı dosyası olduğu için yeni bir metine geçtiğinizde eskileri silmek faydalı olup karışıklığı önleyebilir.