**T.C.**

**KOCAELİ ÜNİVERSİTESİ**

**Mühendislik Fakültesi**

**Bilgisayar Mühendisliği Bölümü**

****

**PROGRAMLAMA LABORATUVARI-II**

***140201035 Fatmanur KÜÇÜKAYVAZ***

***140201054 Hilâl ARSLAN***

***140201020 Ayşegül VEYİSOĞLU***

**Kocaeli-2016**

GRAFLARDA EN KISA YOL AĞACININ OLUŞTURULMASI

Fatmanur Küçükayvaz, Hilal Arslan, Ayşegül Veyisoğlu

Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi

fatmanur.kucukayvaz@gmail.com, hilalarslaan@gmail.com, aysegulveyisoglu@gmail.com

Özet

Elimizde 400 düğümlük bir matris bulunmakta kullanıcıdan ne kadar düğüm girmek istediği bilgisi alınır. En fazla girebileceği düğüm sayısı 400 dür . Daha fazla düğüm sayısı girdiğinde kullanıcıdan tekrar düğüm sayısı girmesi istenir. Daha sonra girdiği düğümlerin koordinatlarına kullanıcı karar verir. Kullanıcıdan koordinatlar istenir. Kullanıcı aynı koordinatı birden fazla kez girerse kullanıcıdan tekrar koordinat girişi istenir. Bu bölümde de kullanıcı koordinat olarak en fazla 19 girebilir. Çünkü matris 0'a 0'dan başlamaktadır. Eğer kullanıcı daha fazla bir sayı girerse kullanıcıyı uyarıcı bir mesajla tekrar giriş istenir. Kullanıcının girdiği koordinatlara göre düğümler yerleştirilir. Düğümler rakamlar ile belirtilir. Düğümlerin koordinatları bir dizide tutulur. Daha sonra düğümler arasındaki mesafe x ve y koordinatları arasındaki farklar toplanarak bulunur. Bu yollar ekranda çizilir. Ekrana çizme işlemi allegro grafik kütüphanesi ile yaptırılıır. Bunlar küçükten büyüğe göre sıralanır. Bunun için bubble sort algoritması kullanılır. En küçük yoldan başlanarak sırayla diğer düğümlere ulaşılmaya çalışılır. Ancak çevrim olduğu yollar kullanılmaz. En kısa yol bulunurken kruskal algoritması kullanılmıştır. Böylece düğümler arasındaki en kısa yol hesaplanır. Bulunan yollar tekrar matriste çizilir. En son olarakta toplam yol hesaplanır.

# 1.Giriş

400 düğümlük bir matrise yerleştirilecek düğümler arasındaki yol çevrimsiz bir şekilde hesaplanıp hem görsel hem de kullanıcı dostu olacak şekilde sunulmalıdır. En kısa yol algoritması bulunduktan sonra ekranda en kısa yol ağacı çizilmelidir. Maliyet hesaplanmalıdır.

# 2.Temel Bilgiler

Projede C++ kullandık. C++ nesneye yönelik programlama dilidir. Nesneye yönelik programlama dilinin en önemli avantajı bize sınıfları kullanabilir imkânı sunmasıdır.

Code Blocks, özgür açık kaynak kodlu bir C++ tümleşik geliştirme ortamıdır. wxWidgets  tabanlı tamamen özelleştirilebilir arabirimiyle, GNU/Linux, Microsoft Windows, MacOS platformlarında sorunsuzca kullanılabilmektedir. Gelişmiş plugin desteğiyle kod yazımı esnasında ihtiyaç duyabileceğiniz birçok yardımcı fonksiyon sunar ve kod üzerinde tam bir hâkimiyet kurmanıza olanak tanır.

Allegro grafik kütüphanesi yazdığımız koda görsellik katmamızı sağlar. Genelde oyun yazarken kullanılır. Şekillerin, resimlerin hareket ettirilmesi ve grafik tarzı şekillerin konsol ekrana çizimini sağlar. Biz bu projede Allegro grafik kütüphanesini yolların görsel gösterimi için kullandık. *[1]*

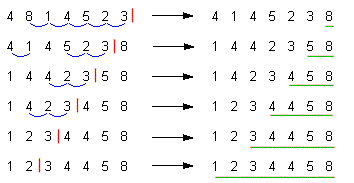
# 3.Geliştirilen Mimari

**3.1 Kullanılan sınıfların ve metotlarının kısa açıklamaları**

Projede iki tane sınıf kullandık. İlk sınıfımız koordinatlar sınıfıdır. Bu sınıf içinde x ve y değişkenleri vardır. Bu değişkenler düğümlerin koordinatlarını tutmak için kullanılır.

İkinci sınıfımız düğümler arasındaki ilişkileri belirten sınıftır. Bu sınıf içinde ise ilk düğümü ikinci düğümü ve bu düğümler arasındaki uzaklığı tutan değişkenler bulunur. Bu sınıflardan koordinatların tutulduğu dizi adlı bir dizi ve ilişki adında bir dizi oluşturduk.

İlk kullandığımız metot düğümler arasındaki uzaklıkları karşılaştırıp küçükten büyüğe doğru sıralamamızı sağlayan bubble sort algoritmasıdır (kabarcık sıralaması). Kabarcık sıralaması, [bilgisayar bilimlerinde](https://tr.wikipedia.org/wiki/Bilgisayar_bilimleri) kullanılan yalın bir [sıralama algoritmasıdır](https://tr.wikipedia.org/wiki/S%C4%B1ralama_algoritmas%C4%B1). Sıralanacak dizinin üzerinde sürekli ilerlerken her defasında iki öğenin birbiriyle karşılaştırılıp, karşılaştırılan öğelerin yanlış sırada olmaları durumunda yerlerinin değiştirilmesi mantığına dayanır. Algoritma, herhangi bir değişiklik yapılmayıncaya kadar dizinin başına dönerek kendisini yineler. Adına kabarcıksıralaması denmesinin nedeni büyük olan sayıların aynı suyun altındaki bir [kabarcık](https://tr.wikipedia.org/w/index.php?title=Kabarc%C4%B1k&action=edit&redlink=1) gibi dizinin üstüne doğru ilerlemesidir. Kabarcık sıralama algoritmasının ortalama ve en kötü durumdaki karmaşıklığı O(n2)'dir. Algoritma ortalama ve en kötü durumda (n2/2) adet karşılaştırma ve yer değiştirme gerçekleştirir.



*[8]Şekil 1: Bubble sort gösterimi*

En kısa yol algoritmasını belirlemek için birçok algoritma geliştirilmiştir.

**Kruskal Algoritması:** Daha az maliyetli kenarları tek tek değerlendirerek yol ağacını bulmaya çalışır. Ana işlemler birden çok ağaç oluşturabilir.

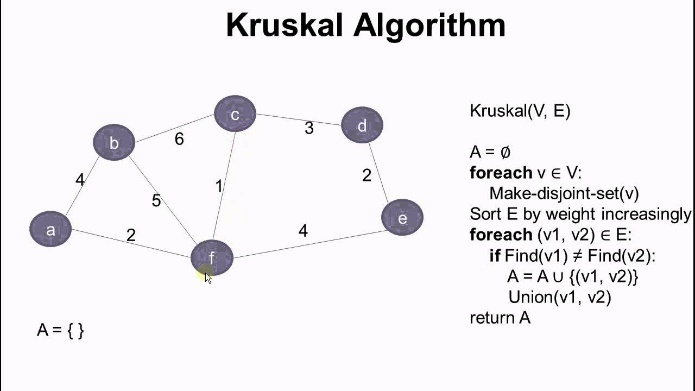
**Kruskal Algoritmasının Adımları:**

**1.Adım:** Graf üzerindeki düğümler, aralarında bağlantı olmayan N tane bağımsız küme gibi düşünülür.

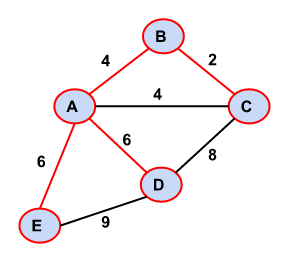
**2.Adım:** Daha sonra bu kümeler maliyeti en az olan kenarlarla birleştirilir(çevrim oluşturmayacak şekilde).

**3.Adım:** Düğümler arasında bağlantı olan tek bir küme oluşturulmaya çalışılır.

**4.Adım:** Küme birleştirme işleminde en az maliyetli olan kenardan başlanılır; daha sonra kalan kenarlar arasından en az maliyetli olanlar seçilir.



*[7]Şekil 2: Kruskal algoritması sözde kod*



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Kenar | B-C | A-B | A-C | A-D | A-E | C-D | D-E |
| Ağırlık | 2 | 4 | 4 | 6 | 6 | 8 | 9 |

**Prim Algoritması:** En az maliyetli kenardan başlayıp onun uçlarından en az maliyetle genişleyecek kenarın seçilmesine dayanır. Bir tane ağaç oluşur.

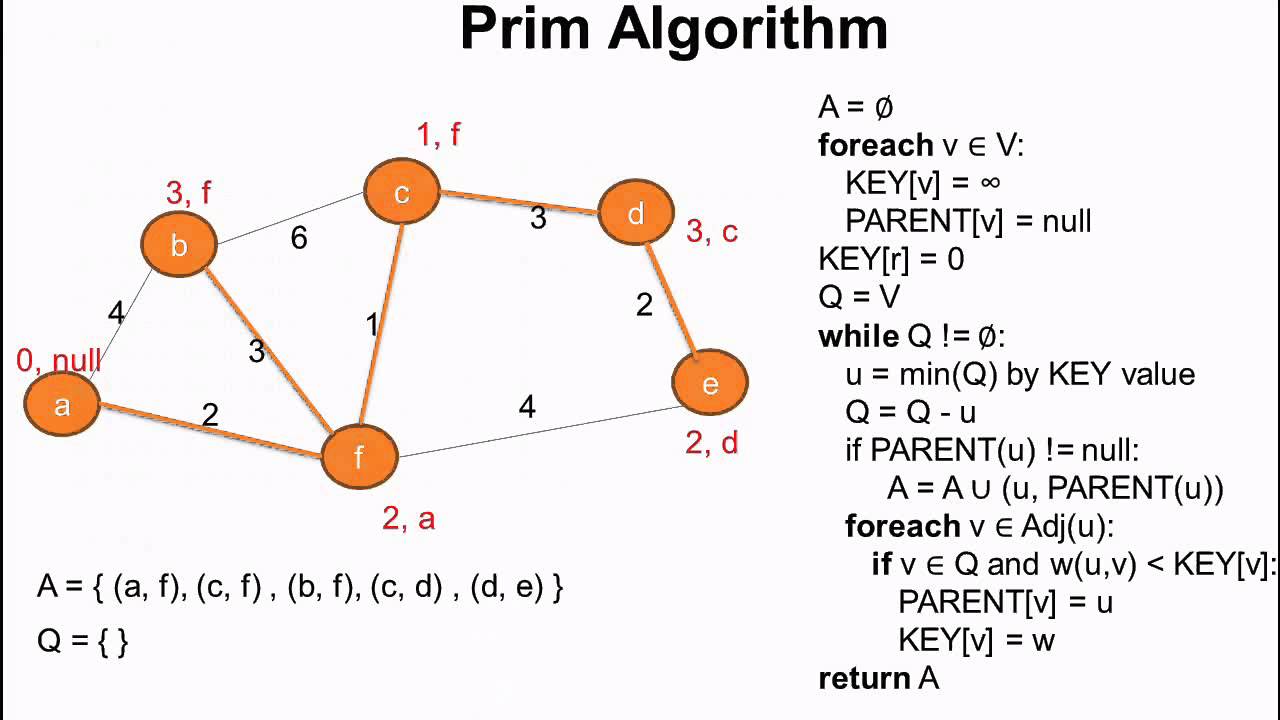
**Prim Algoritmasının Adımları:**

**1.Adım:** Başlangıçta, herhangi bir noktayı ağacı oluşturmak için seç.

**2.Adım:** Oluşturulan ağaca eklemek için, şu ana kadar oluşturulmuş ağaç üzerinden erişilebilen ve daha önceden ağaca katılmamış olan en küçük ağırlıklı kenarı seç.

**3.Adım:** Eğer bu kenarın ağaca katılması, bir çember oluşmasına sebep olmuyorsa, ağaca ekle.

**4.Adım:** Ağaçtaki kenar sayısı (N-1)’e ulaşana kadar ikinci adıma geri dön.



*[6] Şekil 3: Prim algoritması sözde kod*

En kısa yol algoritması olarak kruskal algoritmasını kullandık. Projedeki düğümleri ve yolları bir graf olarak düşünebiliriz. Graf, düğümler ve bu düğümleri birbirine bağlayan kenarlardan oluşan bir tür ağ yapısıdır. Düğümlerimiz girilen koordinatlara yerleştirilen rakamlardır. Kenarlarda düğümden düğüme çizilen yollardır.

**3.2 Kullanılan veri yapılarının küçük örneklerle anlatımı ve kısa açıklamaları ve karşılaşılan problemler ile çözüm yaklaşımları**

En başta oluşturduğumuz iki tane sınıftan dizi şeklinde nesneler oluşturduk. Kullanıcıdan girdi istemeden önce boş matrise '.' elemanını her koordinatına atadık. Daha sonra girdi istedik. Kullanıcıdan girdi isterken aynı zamanda bazı kontroller yapmamız gerekliydi. Örneğin kullanıcının maksimum 400 düğüm girebilmeliydi. Kullanıcı daha fazla düğüm sayısı girdiğinde kullanıcıları uyarıp tekrar giriş yapmalarını istedik. Bunu while döngüsünün içinde yaptık. Geçerli bir sayı girdiği taktirde break deyimiyle döngüyü bitirdik. Düğüm sayısından sonra kullanıcının bu düğümleri matris içinde nerelere yerleştirmesi gerektiği bilgisi alınmalıydı. Bunun içinde yine bir while döngüsü içinde kullanıcıdan koordinatları istedik. Burada da bir koordinat kontrolü yapmamız gerekliydi. Kullanıcı 19 dan büyük bir koordinat girerse program kullanıcıdan tekrar giriş istemekte. doğru girdiği taktirde yine break deyimiyle döngüyü bitirdik. Matrisimizi char tipinde tanımladığımız için ascii kodlarına göre düğümümüzün ekranda 0'dan başlaması için düğüme 48 eklememiz gerekti. Düğüm sayısı arttıkça sayısal bir şekilde 9 düğüme kadar yazdı. 9. düğümden sonra ascii kodlarına göre farklı şekiller ile düğüm gösterimleri yapıldı. Allegro kütüphanesinde ekrana yazdırırken koordinatların tam ters bir şekilde kullanarak yazıldığını gördük. Bunu ortadan kaldırmak için koordinatları matrise ters bir şekilde okuttuk. Yani x yazılması gereken yere y, y'nin yazılması gerek yere ise x yazdık ve bu şekilde ekrana verilen koordinatlarda düğümleri yazdırdık. Bu şekilde düğümü matristeki koordinatlara atadık. Elimizdeki koordinatları kaybetmemek için tanımladığımız dizi adındaki dizinin x ve y değişkenlerine düğümlerin koordinatları atadık. Dizinin indisi düğüm sayısını, içindeki x ve y ler de o numaralı düğümün koordinatlarını tutuyor oldu.

Kullanıcıdan almamız gerek her şeyi aldıktan sonra ikinci ekran olan allegro ekranını başlatmak için bazı allegro komutları gerekti. allegro\_init ile allegro kütüphanesini başlattık. install\_keyboard ile klavyeden giriş yapmayı sağladık. Ekranın rengini clear\_to\_color fonksiyonuyla boyutunu set\_gfx\_mode fonksiyonuyla sağladık. Bu fonksiyonlardan sonra allegro ekranına oluşturulan matrisi textprintf\_ex ile yazdırdık. Yazdırma fonksiyonu için koordinatlara ihtiyacımız vardı. Her defasında başa dönmesi için x1 koordinatını 0'a eşitlesik. y koordinatını ise 10'ar 10'ar arttırdık.

Matristeki düğümler arasında çizler çizdirmak için vline dikey, hline yatay çizgi çizmeye yarayan fonksiyoları kullandık. Bunun için iki düğümün koordinatı gerekiyordu. İki for döngüsü içinde bütün düğümlerin birbirleriyle olan yolları bulabilmek için ilk for’daki i'yi 0'dan ikinci for’daki j'yi i+1'den başlattık. Matrisi yazdırırken 10'ar 10'ar ilerlediğimiz için koordinatları da kullanırken 10'la çarptık. Fakat öyle olunca çizgiler biraz kaydı bizde önce 5 sonrada 4 eklemeyi denedik. Ve 4 ekleyen çizgileri çizdirdik.

Yolların uzunluklarına bulmak için düğümlerin x ve y koordinatlarını birbirlerinden çıkarıp toplayarak ilişki adındaki diziye atamamız gerekiyordu. İlişki dizisinin maksimum eleman sayısını bulduk. İlişki dizisindeki ilk düğüme ve ikinci düğüme hangi düğümler arasındaki uzaklığı buluyorsak onları uzaklık değişkenine de bulduğumuz sayıyı atadık. Uzaklığı bulurken mutlak değer aldı. Böylece hangi düğümün koordinatı daha büyük diye bakmamıza gerek kalmadı. Hem kaç tane ilişki olduğunu hem de ilişki dizisinin indislerini yazmak için q değişkenini kullandık.

İlişki dizisindeki uzaklıklar herhangi bir sıralamaya göre dizilmediği için bunları küçükten büyüğe sıralamamız gerekliydi. Bunun için bubble sort algoritmasını kullanarak bir kod yazdık. Bir değişkenle iki tanesi arasında kıyaslama yapıp gerektiğinde yer değiştirdik. Böylece yolları küçükten büyüğe sıraladık.

Bu adımdan sonra en kısa yolu bulmamız gerekli. Bunun için en kısa yoldan başlayıp çevrim olup olmadığını kontrol ettirmemiz gerekiyor. Çevrimi renklendirme ile kontrol ettirdik. Renkler adında bir dizi oluşturduk. Her elemanın indisini kendine atadık. En kısa yoldan başlayarak ilişki dizisindeki birinci ve ikinci düğümün renklerini aynı yaptık. Daha sonra ondan büyük olan yolu ekledik. Ekleme işleminden önce düğümlerin renklerini kontrol ettirdik. Eğer renkler aynıysa bu çevrim olduğu anlamına geldiğinden o yolu en kısa yol dizisine eklemedik. Bir sonrakş düğümden devam ettik. Böylece bütün düğümleri taradık.

Bu işlemlerden sonra en kısa yol bulunmuş oldu. Toplam yol için bulduğumuz yolları topladık.

Ekrana çizdirme işleminde aynı şekilde matrisi yazdırdık. Yolları çizerken en kısa yol dizisinde birinci düğüm ve ikinci düğümün koordinatlarına ihtiyacımız var. Bunun için ik tane değişken tutuk. Bu değişkenlerden birinin ilk düğüme diğerinin ikinci düğüme eşit olması gereklidir. Düğüm eşitliğini bulamazsak birinci ve ikinci düğüm değişkenlerini arttırdık. Eşit olduğunda ilk düğümün ve ikinci düğümün koordinatlarını 10 ile çarpıp koordinatlar yazdırma işlemlerinden sonra nerede kaldıysa topladık. Ekrandaki çizgileri hline ve vline fonksiyonlarıyla düğümlerden düğümlere çizdirdik. Daha sonra yazdığımız son matrisin koordinatları taşıyıp yukarı yazdırmak için y koordinatını 0'a çektik. X koordinatı ise 500 ekledik ki yana kaysın. Matrisi yazdırdıktan sonra çizgileri(yolları) düzgün çizmesi için y koordinatından matrisin getirdiği koordinat kadar çıkardık ve çizgileri çizdirdik.

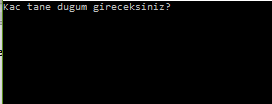
Kodumuzu tamamladıktan sonra denemek için bazı değerler girdik. Fakat 2 düğüm girdiğimizde yolu hesaplamadığını farkettik. İki düğüm arasında sadece bir ilişki vardır. Fakat sıralama algoritması iki nesneyi karşılaştırdığı için ikinci ilişkiye saçma bir değer atanıyordu. Bunu düzelmek için sıralama algoritmasının başına bir kontrol koyduk. Bu sıralama kodunu ilişki sayısı 1 den fazlaysa çalıştırdık. Bunun sonucunda kodu düzelttik.

**3.3 Yazılım Geliştirme İçin Harcanan Süreler (kişi ve saat bazında)**

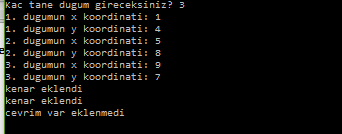
Projenin verildiği tarihten itibaren haftada 3 kez proje için buluşup ortalama olarak haftada 10 saat grup olarak, geriye kalan günlerde ise paylaştığımız görevleri araştırarak bireysel şekilde ortalama günde 1 saat yazılımı geliştirmek için uğraştık. Toplamda grup olarak 40 saat, bireysel olarak 35 saat proje için zaman ayırdık.

# Kullanıcı Kataloğu

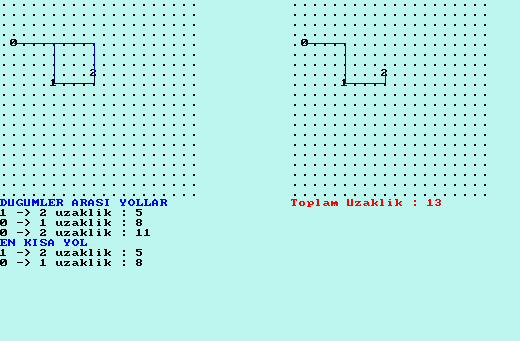
Programı derleyip çalıştırdığımızda ilk olarak konsol ekranında kullanıcıdan gireceği düğüm sayısını istiyoruz.



Kullanıcıdan gireceği düğüm sayısını aldıktan sonra düğüm sayısı kadar düğümlerin x ve y koordinatlarını da kullanıcıdan alıyoruz. Ve düğümlerin çevrim oluşturup oluşturmama durumuna göre en kısa yol dizisine ekliyoruz.



Çevrim oluşturmayan düğümler en kısa yol dizisine eklendikten sonra matris ekrana allegro ile çizdirilir. Daha sonra bulunan yollara göre düğümlerden geçecek şekilde en kısa yol ekrana çizdirilir.



# Sonuçlar

Programı yazma aşamasında karşılaştığımız problemleri çözdük. Şu anda bütün düğümlerin birbirleriyle olan ilişkilerini belirleyip bu ilişkilerin görsel gösterimi yapılmaktadır. Son olarak ise düğümler arasındaki en kısa yol çizilmektedir.

# 3. Denklemler

Koordinatları bulurken

x2=dizi[i]\*10+4

y2=dizi[i]\*10+4

işlemlerini yaparak x ve y koordinatlarını bulduk. Matrisi 10’ar 10’ar ilerleterek yazdırdık.

Allegrodaki koordinatları arttırırken x1 ve y1’leri

x1+=10

y1+=10

olarak 10’ar 10’ar arttırdık.

En kısa yol ağacında çizim yapmak için

x2=dizi[i]\*10+x1+4

y2=dizi[i]\*10+y1+4

denklemlerini kullandık.

# Kaynakça

1. http://kodumbuyazarim.blogspot.com.tr/2015/04/allegro-ve-codeblocks-merhaba.html
2. http://www.programlamadefteri.com/
3. https://tr.wikipedia.org/wiki/Vikipedi
4. http://haktanak.blogcu.com/allegro-kodlari/7851332
5. http://www.esatarslan.com/2009/11/allegro-ornek-program-2.html
6. https://www.youtube.com/watch?v=z1L3rMzG1\_A
7. https://www.youtube.com/watch?v=5xosHRdxqHA
8. http://www.gjszlin.cz/ivt/esf/algoritmizace/bubblesort.php?lang=1
9. http://liballeg.org/stabledocs/en/allegro.html