**HAVUZ PROBLEMİ**

Fatma Sıla SEÇGİN

Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi

[silasecgin@gmail.com](mailto:silasecgin@gmail.com)

170201087

Özet

*Bu projeyi geliştirirken java dilini ve özelliklerini kullandım. Projenin amacı literatürde azami akış (maximum flow) olarak geçen ve düğümler (nodes) arasında akış kapasiteleri belirli bir şekilde (graph) bir başlangıçtan bir hedefe en fazla akışın sağlandığı problemleri ve akışın sistemden geçmemesi için literatürde min-cut olarak geçen yöntemi uyguladık . Bu uygulamada kullanıcının oluşturduğu musluk sayısını (node/düğüm), musluk arasında bağlantı bilgisini veren boru hattı (edge/kenar) ve boru hatlarının kapasitelerini girerek boru hattını oluşturmuş olacaktır. Daha sonrasında uygulama bize maximum flow adımlarını sırayla graph olarak gösterecek ve en son graph ağacının tamamını göstererek bize maximum flow ve min cutları gösterecektir. Uygulama dinamik olarak çalışmaktadır.*

# Genel Yapı

Kullanıcı tarafından oluşturulacak boru sistemi hakkında yapmış olduğumuz uygulama max flow akışını ve min cut adımlarını söyleyecektir. Kullanıcı oluşturacak olduğu musluk sayısını girerek uygulamaya başlar. Uygulama musluk sayısına göre bir matris oluşturur. Kullanıcı bu matrisleri doldurarak bağlantıları ve bu bağlantılar arasındaki kapasiteleri girerek bir boru hattı oluşturur. Uygulama bu adımları nodes.dgs klasörüne kaydederek kendini çalıştırmaya başlar.

Kaydedilen node, edge ve weight leri alarak arama işlemine tabi tutuyoruz. Arama işlemi Ford Fulkerson algoritmasına göre max flow adımlarını buluyor. Max flow adımlarını bulurken gittiği her f1,f2 adımlarını tutarak kullanıcıya bu yollardan geçebilecek kapasiteleri gösteriyor. Girişten gelen kapasite o f1 in toplam kapasitesi oluyor. Uygulama bu f leri toplayarak bize max flow u bulmuş oluyor. Ardından min cut için tabi tuttuğumuz algoritma sayesinde kesilmesi gereken adımları kullanıcıya gösteriyoruz.

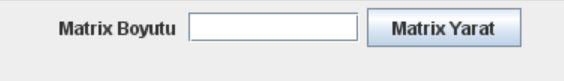
Son adım olarak bütün bir graph göstererek graph üzerinde kesilmesi gereken (min-cut) adımları yeşil olarak görülmektedir. İlk çalışan frame üzerinde de bu adımlar belirtilmektedir.

# 2.2 Yapılan Araştırmalar

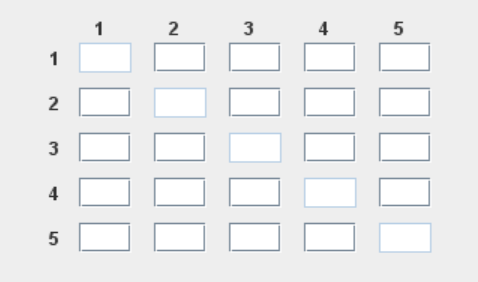
Projeme başlamadan önce genel mantık yapısını kurdum. Sonra max-flow ve min-cut ile ilgili araştırmalar yapıp bu işlemlerin nasıl yapıldığını öğrendim. Graph ile ilgili graphStream kütüphanesini kullanmaya karar verdim. Graph stream dökümanlarını okudum. Uygulama desktop uygulama olacağı için java swing kütüphanesi hakkında makaleler okudum. Ve uygulamayı geliştirmeye başladım.

* 1. **Tasarım**

Kullanıcı oluşturmak istediği musluk sayısını burada girecektir. Eğer boş geçerse hata mesajı (“Matris boyutu boş geçilemez”) ile karşılaşacaktır.



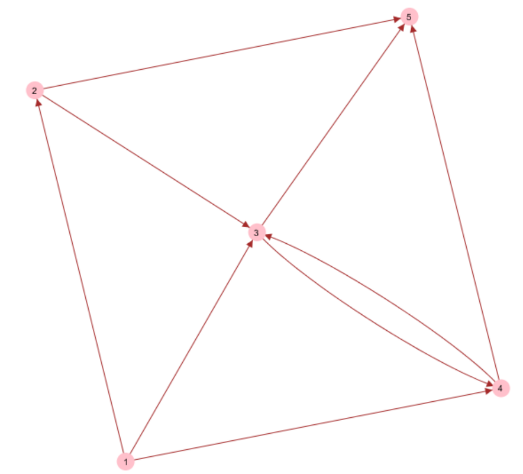
Matris boyutu örneğin 5 girildikten sonra oluşan matris aşağıda gösterilmiştir.



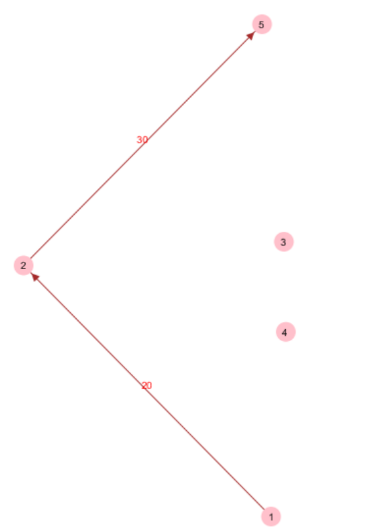
Yukarıdaki matrisi kenar ve ağırlıkları girerek dolduruyoruz.



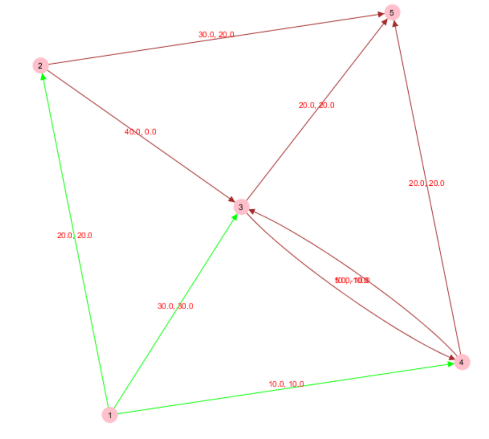
Burada başlangıç musluğunu ve çıkış musluğunu girerek uygulamanın başlamasını sağlıyoruz.



Uygulama ilk olarak bize bütün boru sistemini göstermektedir.



Ardından her f işlemi için sırayla bize f adımını ve o adıma giden borunun kapasitesini göstermektedir. 1. Borudan çıkan kapasite kaynak kapasitesidir. Her f işlemi için adım adım bu işlemler gösterilmektedir.



En sonda açılan graph ise bize boru hattının tamamını ve yeşiller ise kesilmesi gereken min-cut kenarlarını göstermektedir.



Uygulamanın oluşturulduğu alan ise max flow ve min cut sonuçları görülmektedir.

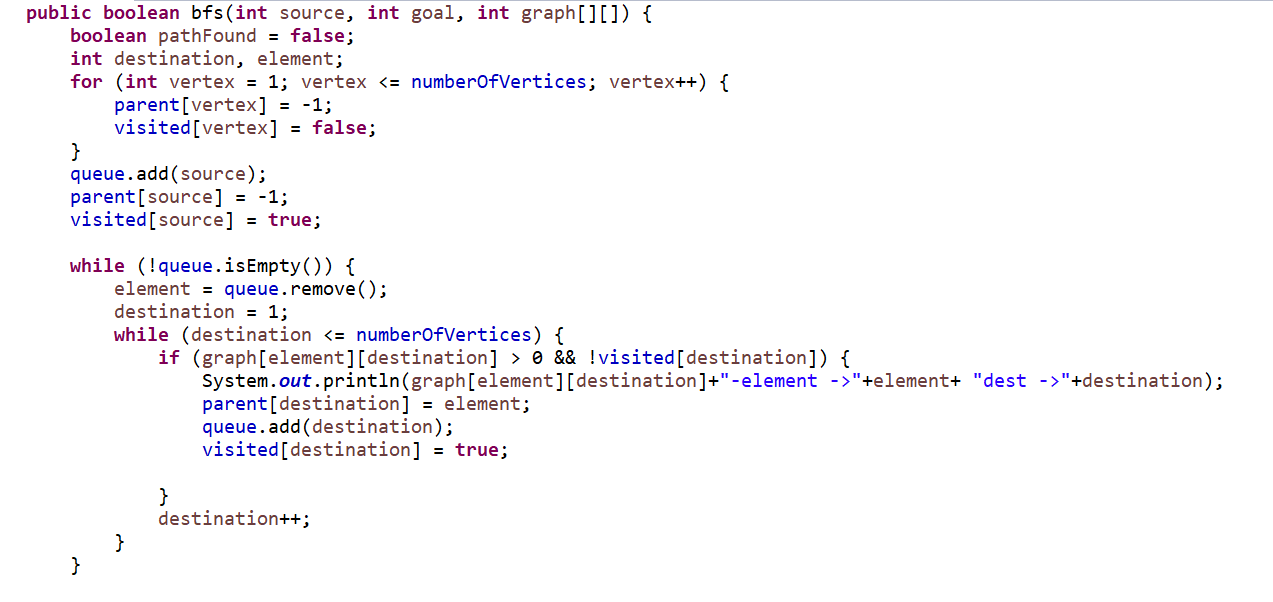
**3.1 Kullanılan Algoritma**

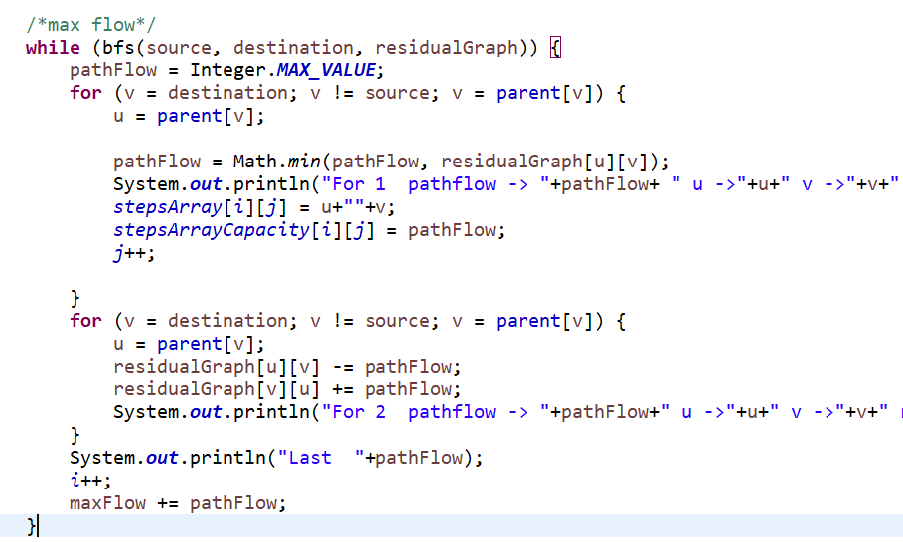
Ford-Fulkerson çözüm için yukarıdaki şekilde öncelikle hedef düğüme giden yolu bulur. Algoritma bu arama işlemi sırasında şayet derin öncelikli arama (depth first search ,DFS) kullanıyorsa ford fulkerson olarak isimlendirilir. Şayet aynı algoritma bu arama işlemi sırasında sığ öncelikli arama (breadth first search, BFS) kullanırsa bu durumda da edmonds karp algoritması olarak isimlendirilir.

Breadth First Search: Bir grafik için Genişlik İlk Geçişi (veya Arama) , bir ağacın Genişliği İlk Geçişi'ne benzer. Buradaki tek yakalama, ağaçların aksine, grafikler döngü içerebilir, bu yüzden aynı düğüme tekrar gelebiliriz. Bir düğümü birden çok kez işlemekten kaçınmak için, bir boole ziyaret dizisi kullanırız. Basit olması için, tüm köşe noktalarına başlangıç ​​noktasından erişilebileceği varsayılmaktadır.

**Big O complexity analysis:**

Her bir node’a yalnızca 1 defa gittiğimiz için karmaşıklık O(n) oluyor.



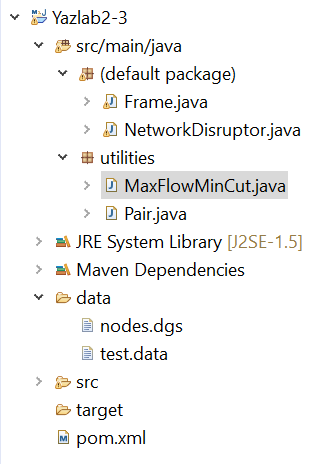


Örnek kaba kodu yukarıda ki fotoğraflarda gösterilmiştir.

* 1. **Yazılım Mimarisi**

Java kullanılarak kodlanmıştır. Spring, graphstream kütüphaneleri kullanılmıştır. Proje maven temellidir. Kullanıcının el ile girdiği datalar frame üzerinde birer paneldir. Graph açılan yerler ise graphstream kütüphanesi üzerinden graph.display();

kodu ile başlatılmaktadır. Pair içerisinde min cut için iteratorler tutulmaktadır. Nodes.dgs içerisinde bütün düğümler düğümlerin bağlı oldukları kenarlar ve kenarların ağırlıkları tutulmaktadır. Uygulama tekrar çalıştırıldığında orası baştan oluşturulmaktadır otomatik olarak. Pom.xml içerisinde ise maven in otomatik olarak indireceği kütüphaneler bulunmaktadır.

****

**3.3 Akış Şeması**

****

* 1. **Referanslar**

1. <http://bilgisayarkavramlari.sadievrenseker.com/2008/11/13/sig-oncelikli-arama-breadth-first-search/> <https://dev.mysql.com/doc/refman/5.6/en/create-table-foreign-keys.html>
2. <https://github.com/nursultanturdaliev/transportation-network-disrupt#fromHistory>
3. <https://www.hackerearth.com/practice/algorithms/graphs/min-cut/tutorial/>
4. <https://www.youtube.com/watch?v=0My2WodLhHc>
5. <https://www.javatpoint.com/java-swing>
6. <https://selimkaratas.com.tr/wp/java-swing-programlama.html>
7. <http://ekremyildiz.net/index.php/2011/09/20/java-frameler-ve-paneller/>
8. <http://javayaz.com/?p=1057>
9. <http://graphstream-project.org/>
10. <http://graphstream-project.org/doc/Tutorials/Getting-Started/>
11. <https://www.geeksforgeeks.org/minimum-cut-in-a-directed-graph/>