HAVUZ PROBLEMİ PROJESİ
(Max-Flow ve Min-Cut Algoritmaları)

Mahmut Yüncü, Berk Hayri Dağ

Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
170202115@kou.edu.tr, 18020203@kou.edu.tr

Özet

Bu projenin amacı bir musluk sistemi ile yönetilen havuzun dolumu için kullanılan borularından geçebilecek maximum kapasiteleri bulabilmek. Kullanıcı kişi sisteme musluk sayısını ve musluktan çıkan bağlantıları ekleyebilir ve bu borular üzerinden geçebileck kapasiteleri belirleyebilir. Bu işlemlerden sonra graf oluşturarak başka bir pencere de açılan grafı görüntüleyebilir. Bu graf üzerinde max flow ve min-cut algoritmaları uygulandıktan sonra hangi kenarlardan ve musluklardan ne kadarlık bir işlem gerçekleştiği nasıl bir sırayla gerçekleştiği yanlarında verilen text-boxlar da belirtilmiştir.

Giriş

Proje için verilen isterler için önce basit bir ister arayüzü tasarladık. Burada kullanıcı musluk sistemini oluşturmak için musluk sayısını ve musluk özelliklerini girmesi beklenmektedir. Girilen özellikler ve musluk sayısına göre iki boyutlu bir matris oluşturulmaktadır. Bu oluşan matris ile graf görselliştirilmesi ve algoritmaların çalıştırılası için kullanılmıştır.

Graf oluşturulduktan sonra kullanıcının olşan grafı eliyle düzeltmesi beklenmektedir. Elde edilen grafın matrisi output da belirtilmiştir.

Yöntem

Proje tamamen java programlama dili ile yazılmıştır. Java Swing ile projenin arayüz tasarımı yapılmıştır. Proje NetBeans IDE üzerinden geliştirilmiştir. Arayüz oluşturulurken sadece javaSwing ve javaAWT bileşenleri kullanılmıştır başka bir kaynaktan yararlanılmamıştır. Algoritmaların temel mantığı ve kullanımı için GeeksForGeeks.com sitesinden yararlanılmıştır. Algoritmaların çalışabilmesi için kullanıcıdan alınan bilgiler bir String üzerinde toplanıp bu bilgiler iki boyutlu bir matris oluşturacak şekilde parçalanmıştır. Bu matris daha sonrasında graf görselliştirilmesi ve algoritmaların çalıştırılası için kullanılmıştır. Grafın görselleştirilmesin de JgraphT kütüphanesi kullanılmıştır.

1.1. Kullanılan Bazı Metotlar

```
//find graph from given string
public static int[][] findGraph (String string)
{
    int[][] temp;
    String info = string;

    String nodes[] = info.split("\\*");
    String[][] lines = new String[nodes.length][];
    for(int i = 0; i< nodes.length; i++) {
        nodes[i] = nodes[i] .replaceAll("(?m)^[ \t]*\r?\n", "");
        lines[i] = nodes[i].split("\\n");
    }

    for(int i = 0; i< nodes.length ; i++) {
        for(int j = 0; j< lines[i].length; j++) {
            System.out.print(lines[i][j]);
        }
    }

    temp = new int[nodes.length -1][nodes.length -1];
        // Fill each row with 0
        for (int[] temp1 : temp) {
            for (int j = 0; j < temp1.length; j++) {
                temp1[j] = 0;
            }
        }

    //kaynak m diye kontrol et
    //de ilse ba lant lar n al
    boolean isSrc = false;
    boolean isSnk = false;

    System.out.println("temp boyutu : "+ (nodes.length -1));
    int[] tempAry = new int[nodes.length -1];
    Arrays.fill(tempAry, new Integer(0));
```

- Girilen verilere göre matris oluşturur.

```
// Returns tne maximum flow from s to t in the given graph
int fordFulkerson(int graph[][], int s, int t, int V)
{
        int u, v;


        int rGraph[][] = new int[V][V];

        for (u = 0; u < V; u++)
                for (v = 0; v < V; v++)
                        rGraph[u][v] = graph[u][v];

        // This array is filled by BFS and to store path
        int parent[] = new int[V];

        int max_flow = 0; // There is no flow initially
        // Augment the flow while tere is path from source
        // to sink
        while (bfs(rGraph, s, t, parent))
        {
                // Find minimum residual capacity of the edhes
                // along the path filled by BFS. Or we can say
                // find the maximum flow through the path found.
                int path_flow = Integer.MAX_VALUE;
                for (v=t; v!=s; v=parent[v])
                {
                        u = parent[v];
                        path_flow = Math.min(path_flow, rGraph[u][v]);
                        sb.append("Musluk"+(char)(65+v) +" --" +"-- Musluk"
                        System.out.println("Musluk"+(char)(65+v) +" --"+ rG

                }

                // update residual capacities of the edges and
                // reverse edges along the path
```

- Ford fulkerson algoritmasını gerçekler

```java
public void createNodes2 (int[][] g, String[] vNames){

        //Create graph model
        SimpleDirectedWeightedGraph<String, MyEdge>  graph =
        new SimpleDirectedWeightedGraph<>
        (MyEdge.class);

        //count edges of the graph
        int edgeCount = 0;
        for (int i = 0; i < g.length; i++) {
            for (int j = 0; j < g[i].length; j++) {
                if(g[i][j] != 0){
                    edgeCount++;
                }
            }
        }
        //Create edge array
        MyEdge[] edge = new MyEdge[edgeCount];

        for (int i = 0; i < g.length; i++) {
            String str = "Musluk" + (char)(65 + i);
            graph.addVertex(str);
        }

         int k = 0;
         for (int i = 0; i < g.length; i++) {
            for (int j = 0; j < g[i].length; j++) {
                if(g[i][j] != 0){

                    String from = "Musluk" + (char)(65 + i);
                    String to = "Musluk" + (char)(65 + j);

                    edge[k] = graph.addEdge(from, to);
                    graph.setEdgeWeight(edge[k], g[i][j] );
                    k++;
```

- Verilen matrise göre graph modeli oluşturulur ve görselleştirilmesinin yapıldığı fonksiyondur.

2. Sonuçlar

Kulladğımız algoritmada, kullanıcıdan alınan bilgiler doğrulturultusunda oluşturulan matrisi kullanarak en fazla kapasiteye sahip boruları sırayla seçerek bitiş musluğundan kaynak musluğa kadar giden bir yol bulup bu yol üzerinde ki değerler karşılaştırılarak dar boğaz değeri bulunur. Bu şekilde max flow algoritması gerçeklenir. Bu gerçekleme algoritması iki fonksiyon ile sağlanır bu fonksiyonlar sırasıyla breadth-first search algoritması ve ford-fulkerson algoritmasıdır.

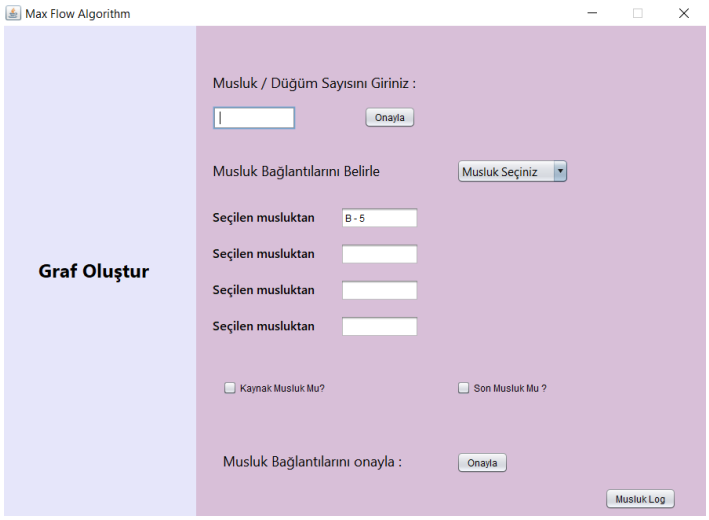
Ford Fulkerson algoritması: Bu algoritmanın amacı, literatürde azami akış (maximum flow) olarak geçen ve düğümler (nodes) arasında akış kapasiteleri belirli bir şekildeki (graph) bir başlangıçtan bir hedefe en fazla akışın sağlandığı problemleri çözmektir.

Minumum Cut algoritması: Bu algoritmanın amacı, graf teorisinde, bir grafiğin minumum kesim; yani bir anlamda minumum keilmesi gereken kenardır. Bir grafiğin köşelerinin iki ayrık alt kümeye bölünmesi.

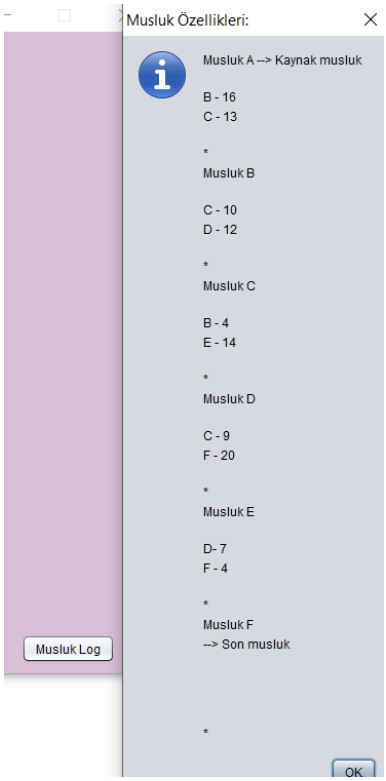
2.1. Kazanımlar

- Graph mantığı öğrenildi.
- Java Swing biçiminde programlamada gelişim sağlandı.
- Max flow algoritmaları öğrenildi.
- Ford Fulkerson algoritması gerçeklendi.

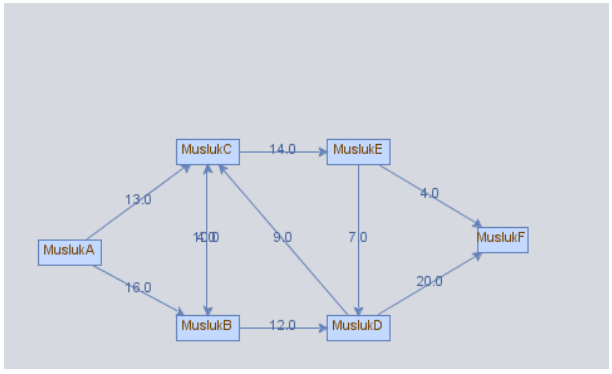
3.1 Örnek Çıktılar

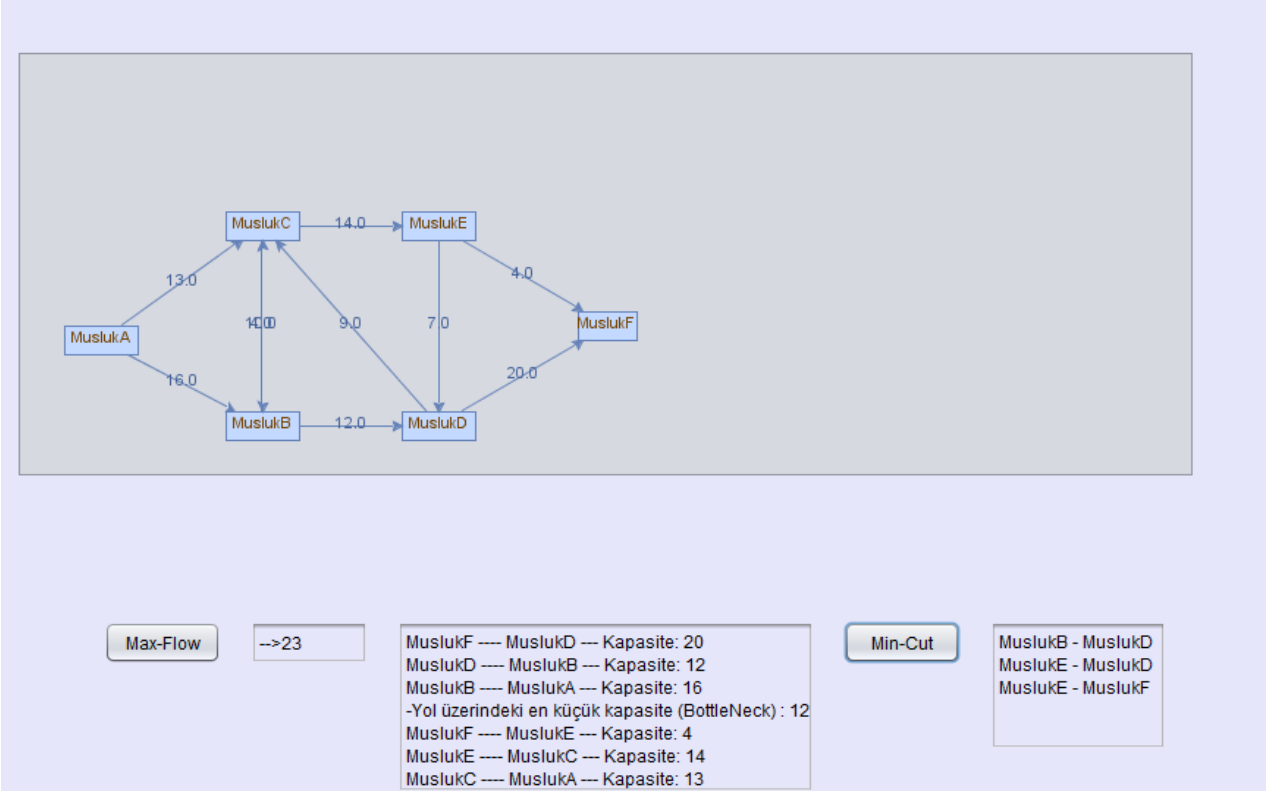


Görsel-1



Görsel-2





Görsel-4

```
 ------------------------------------------------
0       16      13      0       0       0
0       0       10      12      0       0
0       4       0       0       14      0
0       0       9       0       0       20
0       0       0       7       0       4
0       0       0       0       0       0
Shortest path from vertex1 to vertex5:
true
[0, 16, 13, 0, 0, 0]
[0, 0, 10, 12, 0, 0]
[0, 4, 0, 0, 14, 0]
[0, 0, 9, 0, 0, 20]
[0, 0, 0, 7, 0, 4]
[0, 0, 0, 0, 0, 0]
MuslukF --20-- MuslukD
MuslukD --12-- MuslukB
MuslukB --16-- MuslukA
-------------Yol üzerindeki en küçük kapasite (BottleNeck) : 12
MuslukF --4-- MuslukE
MuslukE --14-- MuslukC
MuslukC --13-- MuslukA
-------------Yol üzerindeki en küçük kapasite (BottleNeck) : 4
MuslukF --8-- MuslukD
MuslukD --7-- MuslukE
MuslukE --10-- MuslukC
MuslukC --9-- MuslukA
-------------Yol üzerindeki en küçük kapasite (BottleNeck) : 7
The maximum possible flow is 23
[0, 16, 13, 0, 0, 0]
[0, 0, 10, 12, 0, 0]
[0, 4, 0, 0, 14, 0]
[0, 0, 9, 0, 0, 20]
[0, 0, 0, 7, 0, 4]
[0, 0, 0, 0, 0, 0]
Edges between these Vertexes should be cut for Min - Cut algorithm:
MuslukB - MuslukD
MuslukE - MuslukD
```

Görsel-5

4. Projede Kullanılan Kütüphaneler

```
import java.awt.Color;
import java.util.Arrays;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JOptionPane;
import java.util.LinkedList;
import java.util.Queue;

import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;

--JgraphT--

import com.mxgraph.layout.mxCircleLayout;
import com.mxgraph.layout.mxIGraphLayout;
import com.mxgraph.swing.mxGraphComponent;
import com.mxgraph.swing.util.mxSwingConstants;

import org.jgrapht.ext.JGraphXAdapter;
import org.jgrapht.graph.DefaultWeightedEdge;
import org.jgrapht.graph.SimpleDirectedWeightedGraph;
```

5.Algoritma Karmaşıklık Analizi

Ford-Fulkerson algoritmasının karmaşıklık analizi O(max_flow * Edge) 'dir. Kötü durum olarak her döngüde 1 birim akış eklenir. Bu yüzden karmaşıklık O(max_flow*Edge) ‘dir. (E : edge = kenar sayısı).

6.Pseude Kod

1)İlk artık çizgeyi oluştur (residual graph).

2)Residual graph’ ta başlangıç noktasından bitiş noktasına ulaşan yol olduğu sürece
aşağıdaki adımları yap:

a) Bitiş noktasından başlangıç noktasına olan yolu belirle

b) Yol üzerindeki Minumum kapasiteyi (δ) belirle

c) Yol üzerindeki akışı δ değerince azalt

d) Geri dönüş yolundaki akışı δ değerince arttır.

7. Kaynakça

- https://jgrapht.org/
- https://jgrapht.org/visualizations.html
- https://www.geeksforgeeks.org/max-flow-problem-introduction/
- https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/?ref=lbp
- https://www.geeksforgeeks.org/minimum-cut-in-a-directed-graph/
- https://stackoverflow.com/questions/13378519/how-represent-edge-weight-via-jgrapht-visualization