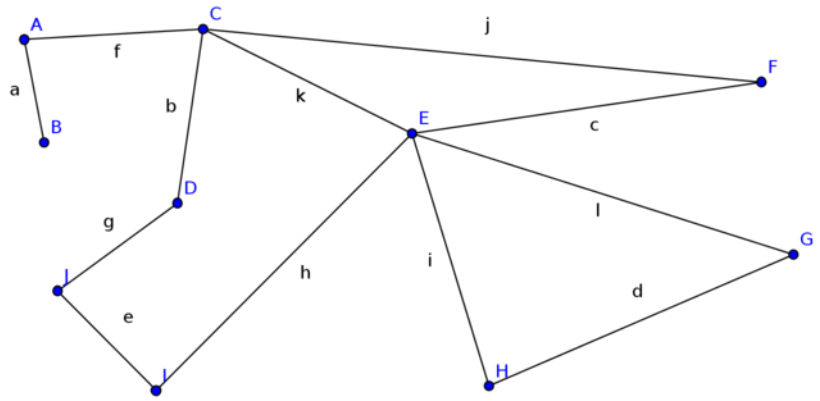
Inhaltsverzeichnis

1 Wiederholung

Übernehmen Sie die in der Vorlesung vorgestellte Implementierung für den Greedy-Matching-Algorithmus und testen Sie sie aus. Alternativ können Sie sich eine eigene Version überlegen/Graphenimplementierungen aus vorangegangenen Vorlesungen nutzen. Den vorgestellten Code finden Sie am Ende des Dokumentes.

2 Greedy-Matching-Algorithmus

2.1 Auf dem Papier



Führen Sie sich das Protokoll für den Greedy-Matching-Algorithmus zu Gemüte und arbeiten Sie es exemplarisch für den angegebenen Graphen auf dem Papier durch, bis ein Maximales Matching gefunden ist.

2.2 Implementierung

Lassen Sie sich anschließend von ihrem Programm ein maximales Matching ausgeben. Machen Sie eine empirische Laufzeitanalyse. Testen Sie den Algorithmus für riesige Graphen.

3 Verbessernde Wege

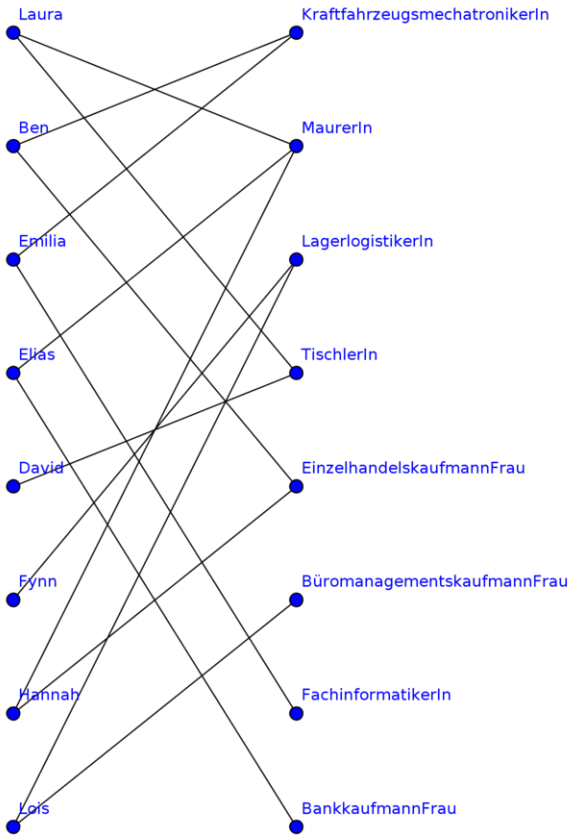
3.1 Auf dem Papier

Finden Sie durch genaues Hinschauen oder dem Protokoll folgend einen verbessernden Weg für das Matching aus der vorherigen Aufgabe.

3.2 Implementierung (optional)

Fügen Sie Ihrer Implementierung aus Aufgabe 1 den Verbessernde-Wege-Algorithmus hinzu und testen Sie ihn.

4 Berufe



Finden Sie für jede Person einen passenden Beruf (perfektes Matching).

5 Code

Edge-Klasse

```
public class Edge {
    String v1;
    String v2;
    Edge(String v1, String v2) {
        this.v1 = v1;
        this.v2 = v2;
    }
    public boolean isNeighbour(Edge e) {
        if (e.v1 == this.v1 || e.v1 == this.v2 ||
```

```
         e.v2 == this.v1 || e.v2 == this.v2) {
            return true;
        }
        return false;
    }
}
```

Graph-Klasse

```
import java.util.ArrayList;
import java.util.List;

public class Graph {
    private List<String> vertices = new ArrayList<>();
    private List<Edge> edges = new ArrayList<>();

    public void setEdges(Edge[] edges) {
        this.edges.clear();
        for (Edge e : edges) {
            this.edges.add(e);
        }
    }

    public void setEdges(String[] edges) {
        if (edges.length % 2 != 0) {
            System.out.println("wrong Array length");
        } else {
            for (int i = 0; i < edges.length; i++) {
                this.addEdge(new Edge(edges[i], edges[i+1]));
                i++;
            }
        }
    }

    public void addEdge(Edge e) {
        this.edges.add(e);
        this.vertices.add(e.v1);
        this.vertices.add(e.v2);
    }

    public List<String> getVertices() {
```

```
        return this.vertices;
    }

    public List<Edge> getEdges() {
        return this.edges;
    }

    public List<Edge> getMaximum(Graph g) {
        List<Edge> e = g.edges;
        List<Edge> eNew = new ArrayList<>();
        List<Edge> m = new ArrayList<>();
        int index = e.size()-1;
        while (!e.isEmpty()) {

            Edge element = e.get(index);
            m.add(element);
            e.remove(index);
            for (Edge elem : e) {
                if (element.isNeighbour(elem)) {
                    eNew.add(elem);
                }

            }
            e.removeAll(eNew);
            index = e.size()-1;

        }
        return m;
    }
}
```

Main-Klasse

```
import java.util.List;

public class Main {
    public static void main(String[] args) {
        Graph g1 = new Graph();
        String[] edges =
                new String[]
                {"a","b","a","c","c","d","c","f","c","e","d","j","j","i",
                "i","e","e","h","e","f","h","g","e","g"};
```

```
        g1.setEdges(edges);
        List<Edge> maximum = g1.getMaximum(g1);
        for (Edge e : maximum) {
            System.out.println(e.v1+","+e.v2);
        }
    }
}
```