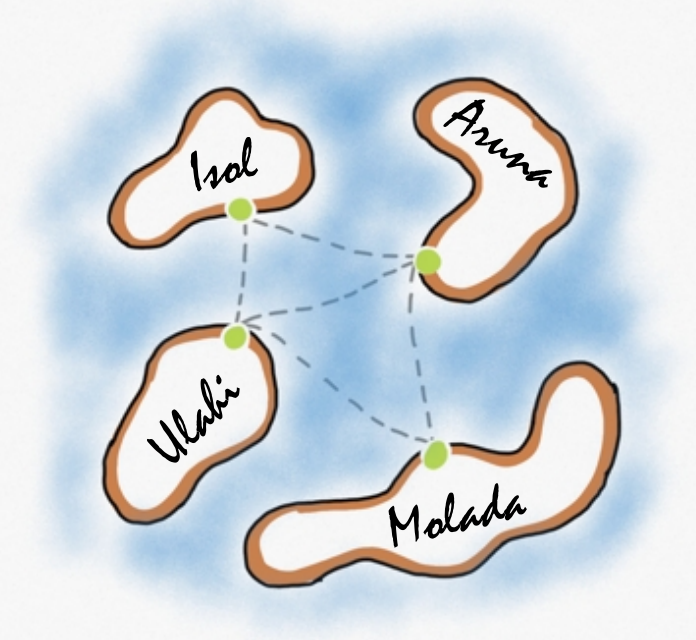
Problemstellung

Die Regionalregierung eines Archipels hat eine Fähre angeschafft, die die vier Inseln des Archipels verbinden soll. Dabei sollen die abgebildeten Fährrouten bedient werden. Idealerweise sollte das Schiff immer wieder eine Rundtour fahren, bei der jede Fährverbindung einmal bedient wird.

1. *Entscheide, ob es eine derartige Rundtour gibt. Gib die Rundtour gegebenenfalls an.*
2. *Entscheide, ob es möglich ist, eine einzige Strecke zu fahren, bei der jede Route genau einmal bedient wird. Gib an, von welchen Häfen aus dies möglich ist.*
3. *** Analysiere, unter welchen Voraussetzungen es eine Rundtour (Starthafen = Zielhafen) gibt, die alle Routen genau einmal abfährt.*



Fährverbindungen (eigenes Werk)

Modellierung

Die Ausgangssituation soll nun als Graph modelliert werden.

4. *Entscheide, welche der folgenden Informationen wichtig für die Suche nach Rundtouren sind:*
   - Name der Inseln
   - Größe der Inseln
   - Entfernung zwischen den Häfen
   - Welche Inseln mit Fährrouten verbunden?
   - genauer Verlauf der Fahrtroute

Modellierung

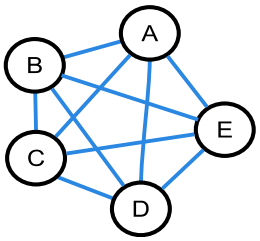
Knoten:

Kanten:

Euler-Zug

Weiterführende Fragen

Ein vollständiger Graph hat eine Verbindung von jedem Knoten zu jedem anderen.



vollständiger Graph mit 5 Knoten (eigenes Werk)

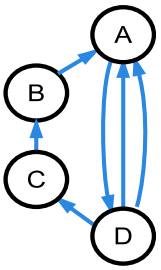
Aufgaben

5. *Zeichne einen vollständigen Graphen mit drei und einen mit vier Knoten.*
6. *Entscheide, ob die Graphen mit drei, vier oder fünf Knoten einen geschlossenen Euler-Zug haben.*

   Zug
7. *Gib eine allgemeine Regel an, wann ein vollständiger Graph einen geschlossenen Euler-Zug hat.*

Für viele Anwendungen verwendet man gerichtete Graphen, d.h. die Kanten haben eine Richtung (z.B. bei einem Stadtplan mit Einbahnstraßen). Beim Euler-Zug darf man die Kanten dann nur in der vorgegebenen Richtung durchlaufen.

Jeder Knoten hat dann einen Ausgangsgrad (wie viele Kanten gehen von einem Knoten aus) und einen Eingangsgrad (wie viele Kanten führen zu einem Knoten hin).



gerichteter Graph (eigenes Werk)

Aufgaben

8. *Entscheide, ob der abgebildete Graph einen geschlossenen Euler-Zug hat.*
9. *Gib eine allgemeine Regel an, wann ein gerichteter Graph einen geschlossenen Euler-Zug hat.*

Bisher wurde nur die Existenz eines geschlossenen Euler-Zuges nachgewiesen. Damit ist noch nicht klar, wie dieser gefunden werden kann. Die Algorithmen von Hierholzer oder Fleury lösen dieses Problem.

- Der Algorithmus von Hierholzer (Eulerkreisproblem[1])
  https://www.youtube.com/watch?v=aAKDDW4gUGs (Sep. 2020)
- Der Algorithmus von Fleury (Eulerkreisproblem)
  https://www.youtube.com/watch?v=I_6ODtkaDwo (Sep. 2020)

1 Im allgemeinen Sprachgebrauch wird das Problem des geschlossenen Euler-Zuges oft als Euler-Kreis bezeichnet, obwohl ein Kreis jeden Knoten nur einmal enthalten darf und in einem Euler-Zug im Allgemeinen die Knoten mehrfach besucht werden.

Beschreibung des Algorithmus, um den Zusammenhang zu testen

Man startet bei einem beliebigen Knoten. Diesem gibt man die Nummer 1 und markiert ihn als fertig bearbeitet. Alle seine Nachbarknoten fügt man einer ToDo-Liste hinzu und kennzeichnet sie als besucht, um auszudrücken, dass sie der ToDo-Liste schon hinzugefügt, aber noch nicht fertig bearbeitet sind.

Dann nimmt man den ersten Knoten aus der ToDo-Liste, nummeriert ihn und markiert ihn als fertig bearbeitet. Alle seine nicht als besucht oder fertig bearbeitet gekennzeichneten Nachbarn fügt man der ToDo-Liste hinzu und kennzeichnet sie als besucht.

Diese Schritte werden so lange wiederholt, bis die ToDo-Liste leer ist.

Entspricht die Nummer des zuletzt nummerierten Knotens der Anzahl der Knoten im Graph, ist der Graph zusammenhängend. Ist sie kleiner, hat man nicht alle Knoten erreicht und der Graph besteht aus mehreren Zusammenhangskomponenten.

Varianten:

Beim Einfügen in die ToDo-Liste können neue Knoten entweder am Anfang der Liste oder am Ende der Liste eingefügt werden.

Pseudocode des Algorithmus, um den Zusammenhang zu testen

Der hier beschriebene Algorithmus bestimmt die Anzahl der vom Startknoten aus erreichbaren Knoten und vergleicht sie mit der Gesamtzahl der Knoten.

```
Zusammenhang des Graphen:
Markiere den Startknoten als besucht
Erzeuge eine leere ToDo-Liste und füge den Startknoten hinzu
Setze nr auf 0

Wiederhole solange die ToDo-Liste nicht leer ist
  Erhöhe nr um 1
  Nimm den ersten Knoten aus der ToDo-Liste heraus
  Markiere ihn und gib ihm die Nummer nr

  Wiederhole für jeden seiner Nachbarn
    Falls der Nachbarknoten noch nicht besucht ist
      Kennzeichne ihn als "besucht"
      Füge ihn am Ende der ToDo-Liste hinzu
    Ende-Falls
  Ende-Wiederhole
Ende-Wiederhole

Falls nr = Anzahl der Knoten im Graph
  Der Graph ist zusammenhängend
Sonst
  Es gibt mehrere Zusammenhangskomponenten
Ende-Falls
```

Variante: Statt am Ende der ToDo-Liste können die Nachbarn auch am Anfang der ToDo-Liste eingefügt werden.

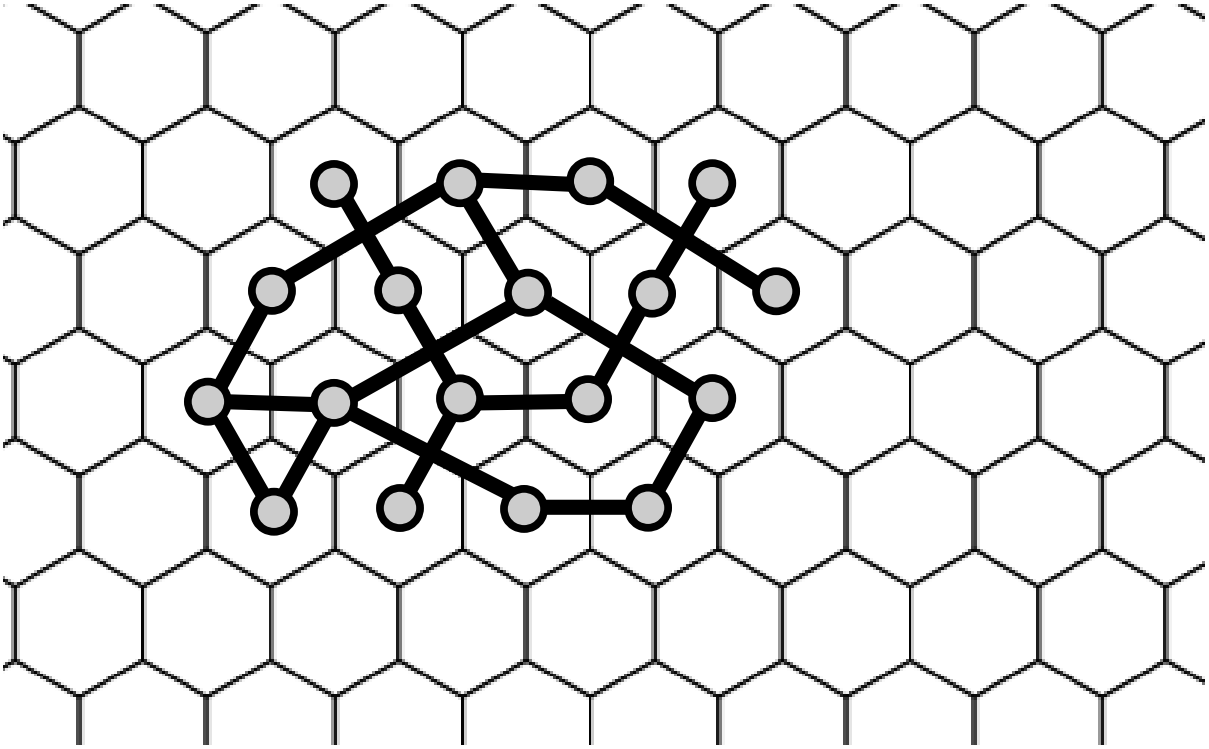
Quelltext des Algorithmus, um Zusammenhang zu testen

```
public boolean istZusammenhaengend() {
    if (g.getAnzahlKnoten()==0) {
        return true;
    }

    ArrayList<Knoten> toDo = new ArrayList<Knoten>();
    getStartKnoten().setBesucht(true);
    toDo.add(getStartKnoten());

    int nr=0;
    while(toDo.size()>0) {
        Knoten k = toDo.remove(0);
        nr++;
        k.setMarkiert(true);
        k.setWert(nr);
        for(Knoten n : g.getNachbarKnoten(k)) {
            if(!n.isBesucht()){
                toDo.add(n);
                g.getKante(k,n).setMarkiert(true);
                n.setBesucht(true);
            }
        }
    }
    if(nr == g.getAnzahlKnoten()) {
      return true;
    } else
    {
      return false;
    }
  }
```

Beispiel 1



Beispiel 2