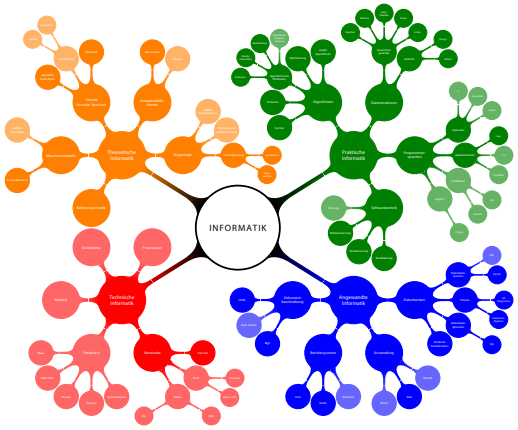
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 30

Graphen

Busnetze, Molekülverbindungen, Genecluster

Vorlesung Einführung in die Informatik 2 vom 8. April 2014 von Till Tantau



Lernziele von Kapitel 30

1. Arten von Graphen kennen
2. Daten und ihre Struktur mittels Graphen modellieren können
3. Die Datenstruktur des Graphen verstehen
4. Konzept des Graphproblems verstehen

Gliederung von Kapitel 30

Graphen: Knoten und Kanten

Definition (Graph)

Ein *Graph* besteht aus *Knoten*, die durch *Kanten* verbunden sind.

Die *Idee* ist, dass

- die Knoten *Dinge* modellieren und
- die Kanten *Beziehungen* modellieren.

Beispiel: Verkehrsnetze

Wir können *Verkehrsnetze* mittels Graphen wie folgt modellieren:

- Die *Knoten* sind Orte oder Straßenkreuzungen.
- Die *Kanten* sind Straßen.

Zur Diskussion

Wenn wir ein Verkehrsnetz als Graph modellieren, welche Informationen sollten wir dann speichern betreffend

- die Knoten (die Orte) und
- die Kanten (die Straßen)?

Beispiele aus der Molekularbiologie

Szenario: Moleküle

Wir können *Moleküle* mittels Graphen wie folgt modellieren:

- Die *Knoten* sind die Atome.
- Die *Kanten* sind die Bindungen.

Szenario: Genregulation

Wir können *Genregulation* mittels Graphen wie folgt modellieren:

- Die *Knoten* sind Gene.
- Die *Kanten* sind die Abhängigkeiten zwischen Genen; Kanten geben also an, wie Gene einander beeinflussen.

Es gibt vieles, was man bei Graphen festlegen kann.

- Welche Art von Knoten gibt es?
- Sind Beziehungen gerichtet oder ungerichtet?
- Sind die Knoten beschriftet?
- Sind die Kanten beschriftet?
- . . .

Der mathematische Begriff des Graphen.

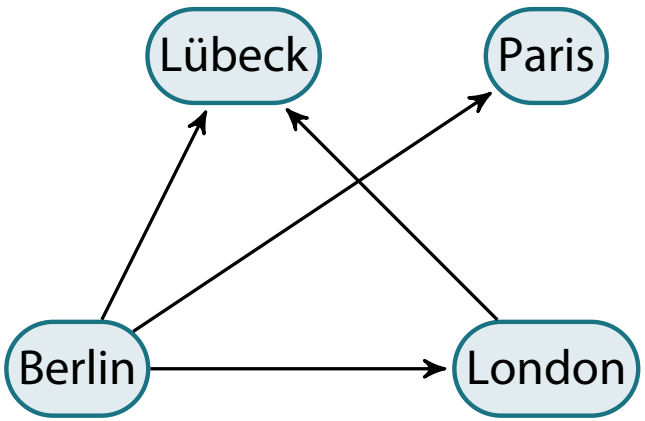
Definition (Graph)

Ein *Graph* besteht aus einer Knotenmenge V und einer Kantenmenge $E \subseteq V \times V$.

Solche Graphen nennen wir auch *gerichtete Graphen*.

Beispiel (Gerichteter Graph)

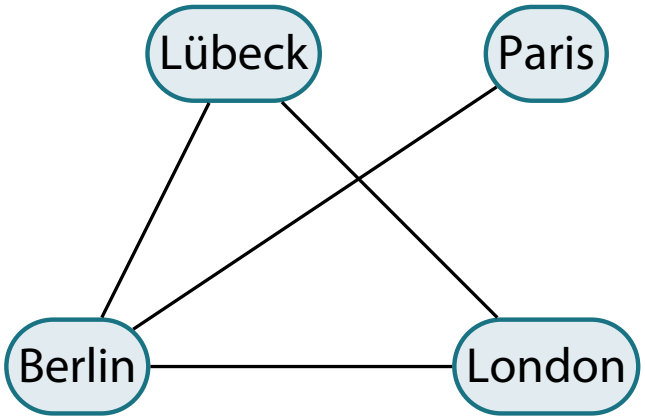
$V = \{\text{Berlin}, \text{London}, \text{Paris}, \text{Lübeck}\}$.
$E = \{(\text{Berlin}, \text{London}), (\text{Berlin}, \text{Paris}), (\text{Berlin}, \text{Lübeck}), (\text{London}, \text{Lübeck})\}$



Spezielle Graphen: Ungerichtete Graphen.

- Bei *ungerichteten* Graphen haben die Kanten keine Richtung.
- Dies kann man mathematisch unterschiedlich modellieren. Typisch ist, dass man annimmt, dass E symmetrisch ist. (Das heißt, falls $(u, v) \in E$, so auch $(v, u) \in E$.)

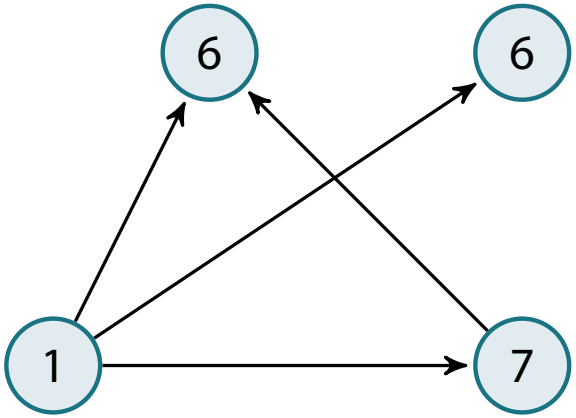
Beispiel (Ungerichteter Graph)



Spezielle Graphen: Knotengewichtete Graphen.

- Bei *knotengewichteten* Graphen wird jedem Knoten noch eine Zahl zugeordnet, genannt *Gewicht* des Knotens.

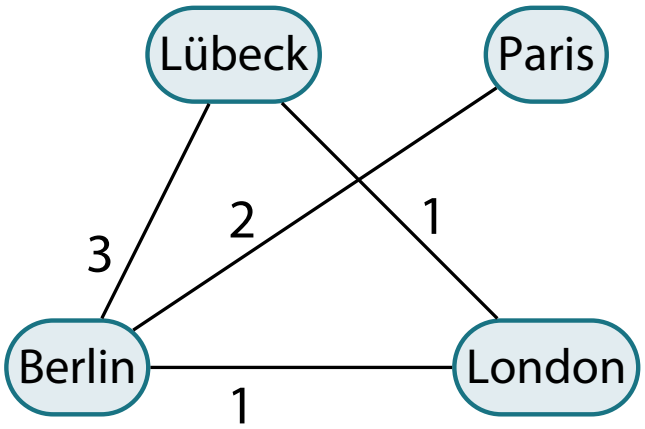
Beispiel (Knotengewichteter Graph)



Spezielle Graphen: Kantengewichtete Graphen.

- Bei *kantengewichteten* Graphen wird jeder Kante noch eine Zahl zugeordnet, genannt *Gewicht* der Kante.

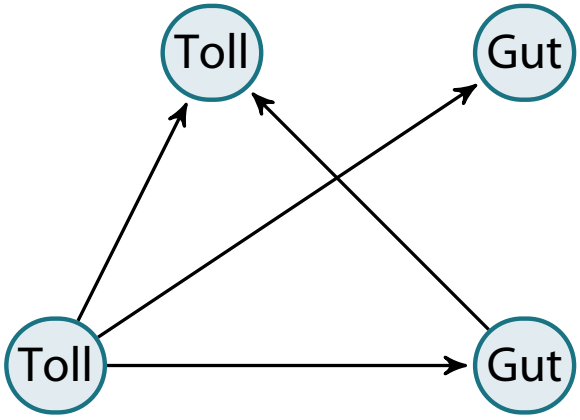
Beispiel (Kantengewichteter Graph)



Spezielle Graphen: Gelabelte Graphen.

- Manchmal möchte man den Knoten und/oder den Kanten noch ein *Label* (einen kleinen Notizzettel) ankleben.

Beispiel (Knotengelabelter Graph)



Was sind Graphprobleme?

Definition

Graphprobleme sind alle Problemstellungen, bei denen die Eingaben (kodierte) Graphen sind.

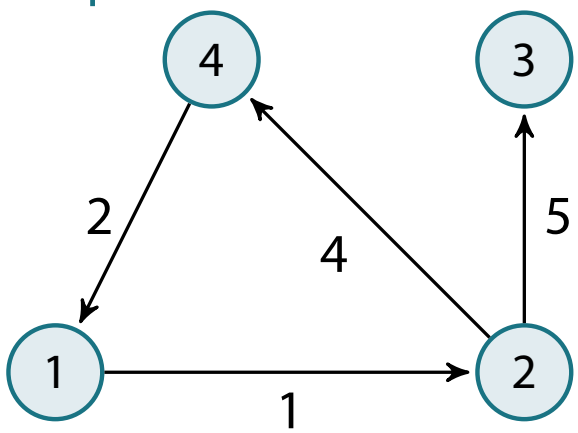
Insbesondere gibt es Graphprobleme als

1. Entscheidungsprobleme, wobei dann die Frage ist, ob ein gegebener Graph eine bestimmte Eigenschaft hat wie »Gibt es einen Weg vom ersten zum letzten Knoten?«; sowie als
2. Optimierungsprobleme, wobei dann die Frage ist, wie eine optimale Lösung aussieht (»Wie sieht der kürzeste Weg vom ersten zum letzten Knoten aus?«).

Was ist eine Adjazenzmatrix?

- Ein Graph habe n Knoten.
- Bilde nun eine Tabelle, die n Zeilen und n Spalten hat.
- Trage in die i-te Zeile und die j-te Spalte das Gewicht der Kante vom i-ten Knoten zum j-ten Knoten ein.
  Trage dort eine 0 ein, wenn es keine Kante gibt.

Graph



Adjazenzmatrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{pmatrix}$$

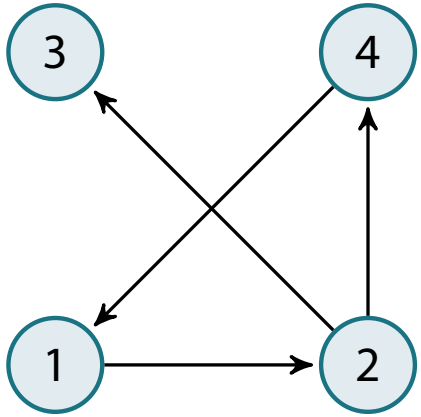
Graphen als Adjazenzmatrizen in Java

```
class GraphAsAdjacencyMatrix {
  int [][] weights;

  GraphAsAdjacencyMatrix (int numberOfVertices) {
    this.weights =
      new int[numberOfVertices][numberOfVertices];
  }

  void addEdge(int u, int v, int weight) {
    this.weights[u][v] = weight;
  }

  void removeEdge(int u, int v) {
    this.weights[u][v] = 0;
  }

  boolean connected(int u, int v) {
    return (this.weights[u][v] != 0); // Boole'scher
        Ausdruck!
  }

  int weight(int u, int v) {
    return this.weights[u][v];
  }
}
```

Was ist eine Adjazenzliste?

- Ein Graph habe n Knoten.
- Für jeden Knoten wird eine Liste gespeichert,
- in der alle Knoten gespeichert sind, zu denen er eine Kante hat.

Graph



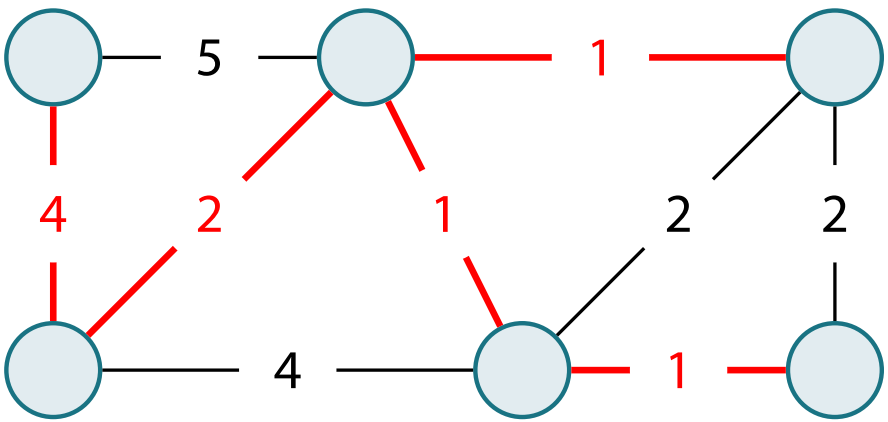
Adjazenzlisten

Liste von Knoten 1: 2
Liste von Knoten 2: 3, 4
Liste von Knoten 3: leer
Liste von Knoten 4: 1

Einführung zu Gerüsten

Definition (Gerüst)

Ein *Gerüst* (englisch *spanning tree*) ist ein Teilbaum des Graphen, der jeden Knoten berührt.
Ein *minimales Gerüst* ist ein Gerüst, so dass die Summe der Kantengewichte minimal ist.



Das Gerüstproblem als Optimierungsproblem

Das Minimale-Gerüst-Problem

Eingaben Ein kantengewichteter Graph.

Ausgabe Ein Gerüst, so dass die Summe der Gewichte im Baum minimal ist.

Eine Anwendung des Gerüstproblems.

Y. Inbar, H. Benyamini, R. Nussinov und H. Wolfson
Protein structure prediction via combinatorial assembly of sub-structural units,
Bioinformatics, 19(1) i158–i168, 2003.

Vorgehen in obiger Arbeit:

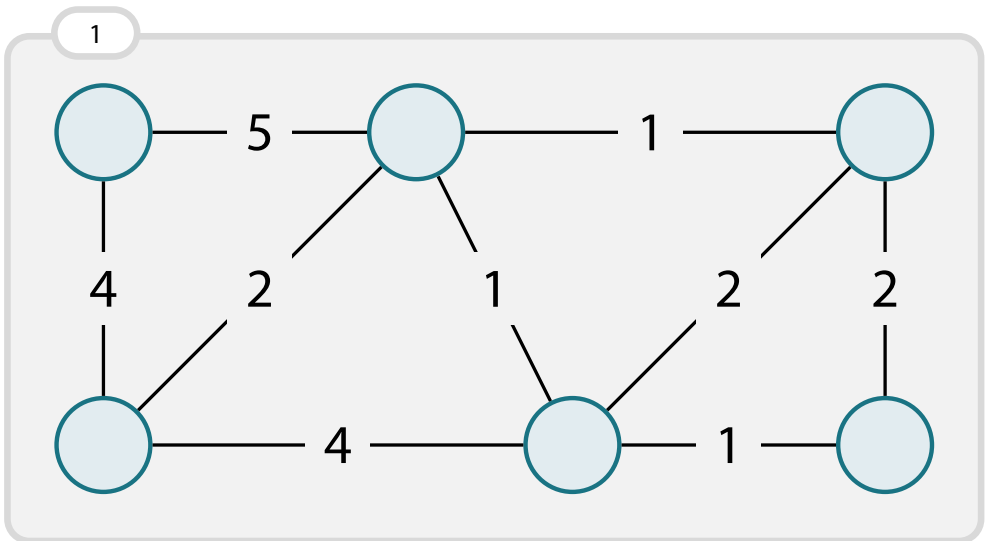
1. Teile die Aminosäuresequenz in kleine Stücke auf.
2. Berechne für die Stücke, welcher bekannten Aminosäuresequenz sie am ähnlichsten sind. Weise den Stücken die Raumstruktur dieser ähnlichsten Sequenzen zu.
3. Berechne für je zwei Stücke, wie gut sie zusammenpassen (wie Legosteine).
4. Bilde daraus einen Graphen, dessen Knoten die Stücke sind und dessen Kanten für je zwei Stücke angeben, wie gut sie zusammenpassen.
5. Berechne ein minimales Gerüst in dem Graphen.
6. Puzzle die Stücke entsprechend dem Gerüst zusammen.

Ein effizienter Algorithmus für das Gerüstproblem.

Algorithmus

Eingabe ist ein kantengewichteter Graph.

1. Finde eine Kante minimalen Gewichts. Diese Kante bildet unser initiales Teilgerüst.
2. Wiederhole folgendes, solange noch kein Gerüst vorliegt:
   - 2.1 Finde die Kante minimalen Gewichts, die das Teilgerüst mit einem noch nicht erreichten Knoten verbindet.
   - 2.2 Füge die Kante zum Teilgerüst hinzu.

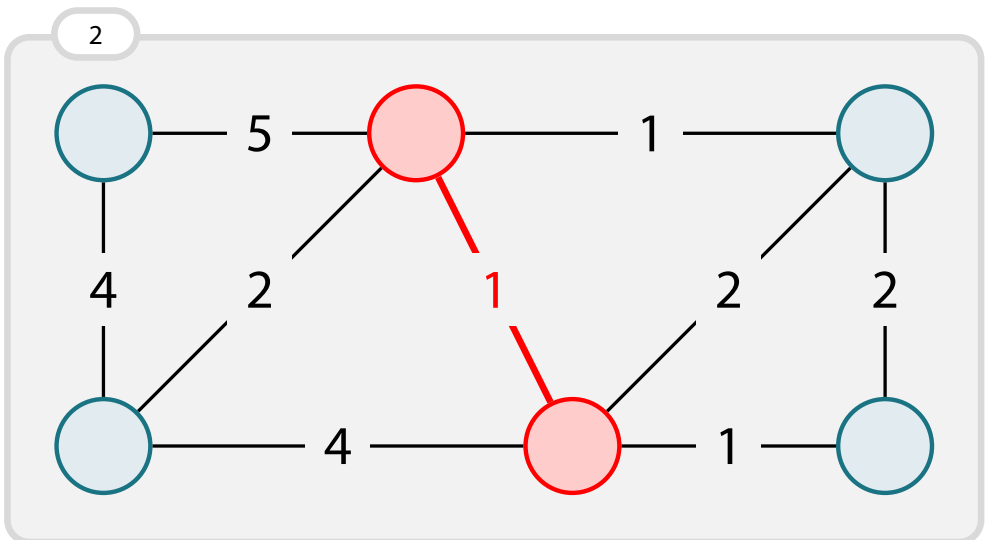


Ein effizienter Algorithmus für das Gerüstproblem.

Algorithmus

Eingabe ist ein kantengewichteter Graph.

1. Finde eine Kante minimalen Gewichts. Diese Kante bildet unser initiales Teilgerüst.
2. Wiederhole folgendes, solange noch kein Gerüst vorliegt:
   - 2.1 Finde die Kante minimalen Gewichts, die das Teilgerüst mit einem noch nicht erreichten Knoten verbindet.
   - 2.2 Füge die Kante zum Teilgerüst hinzu.

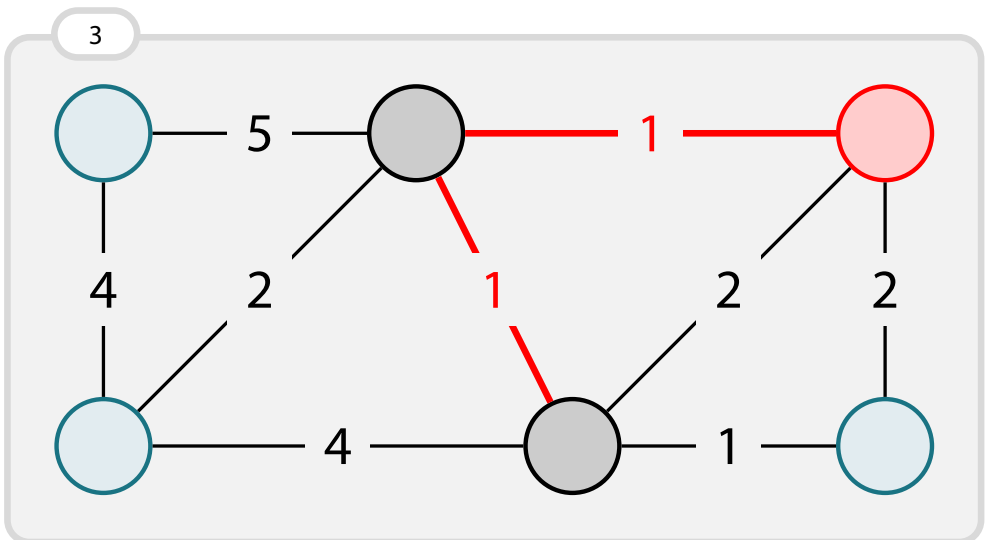


Ein effizienter Algorithmus für das Gerüstproblem.

Algorithmus

Eingabe ist ein kantengewichteter Graph.

1. Finde eine Kante minimalen Gewichts. Diese Kante bildet unser initiales Teilgerüst.
2. Wiederhole folgendes, solange noch kein Gerüst vorliegt:
   - 2.1 Finde die Kante minimalen Gewichts, die das Teilgerüst mit einem noch nicht erreichten Knoten verbindet.
   - 2.2 Füge die Kante zum Teilgerüst hinzu.

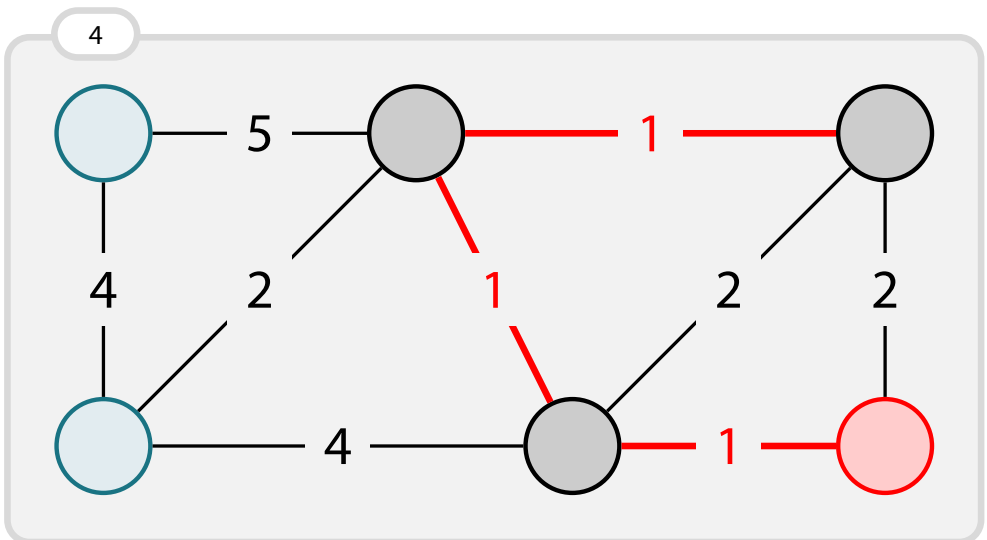


Ein effizienter Algorithmus für das Gerüstproblem.

Algorithmus

Eingabe ist ein kantengewichteter Graph.

1. Finde eine Kante minimalen Gewichts. Diese Kante bildet unser initiales Teilgerüst.
2. Wiederhole folgendes, solange noch kein Gerüst vorliegt:
   - 2.1 Finde die Kante minimalen Gewichts, die das Teilgerüst mit einem noch nicht erreichten Knoten verbindet.
   - 2.2 Füge die Kante zum Teilgerüst hinzu.

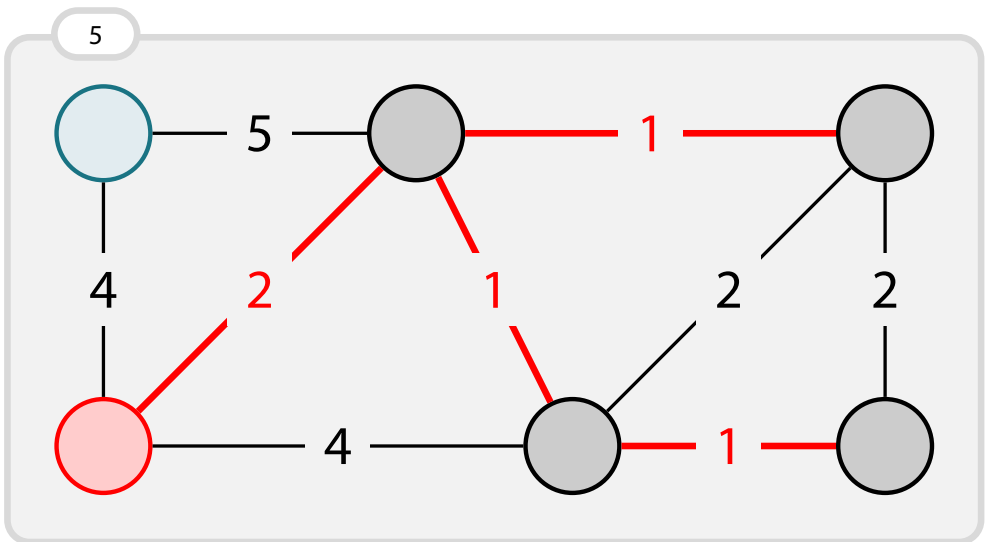


Ein effizienter Algorithmus für das Gerüstproblem.

Algorithmus

Eingabe ist ein kantengewichteter Graph.

1. Finde eine Kante minimalen Gewichts. Diese Kante bildet unser initiales Teilgerüst.
2. Wiederhole folgendes, solange noch kein Gerüst vorliegt:
   - 2.1 Finde die Kante minimalen Gewichts, die das Teilgerüst mit einem noch nicht erreichten Knoten verbindet.
   - 2.2 Füge die Kante zum Teilgerüst hinzu.

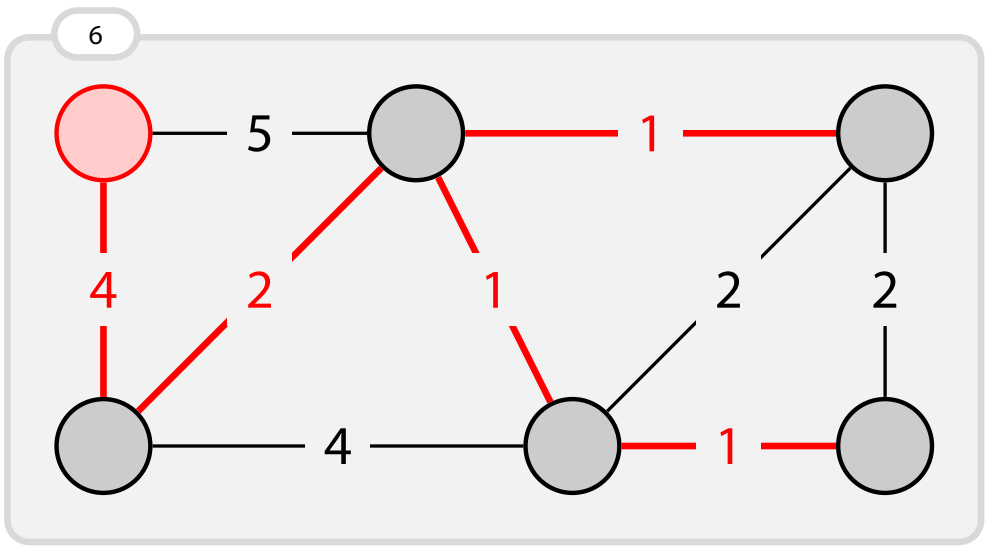


Ein effizienter Algorithmus für das Gerüstproblem.

Algorithmus

Eingabe ist ein kantengewichteter Graph.

1. Finde eine Kante minimalen Gewichts. Diese Kante bildet unser initiales Teilgerüst.
2. Wiederhole folgendes, solange noch kein Gerüst vorliegt:
   - 2.1 Finde die Kante minimalen Gewichts, die das Teilgerüst mit einem noch nicht erreichten Knoten verbindet.
   - 2.2 Füge die Kante zum Teilgerüst hinzu.



Problem des Handlungsreisenden.

- Ein Handlungsreisender möchte in einer Rundreise eine Reihe von Städten besuchen.
- Dabei soll er jede Stadt genau einmal besuchen.
- Die Benzinkosten für eine Fahrt zwischen zwei Städten hängt linear von deren Entfernung ab.
- Ziel ist es, eine möglichst billige Rundreise zu finden.

Formalisierungen des Handlungsreisenden-Problems

Euklidisches Handlungsreisenden-Problem

Eingaben Ein Menge von Punkten in der Ebene.

Ausgabe Eine Rundreise (Folge der Punkte, die jeden Punkt genau einmal enthält), so dass die Summe der Länge der Strecken entlang der Rundreise minimal ist.

Allgemeines Handlungsreisenden-Problem

Eingaben Ein kantengewichteter Graph.

Lösungen Rundreise (Folge von miteinander verbundenen Knoten, die jeden Knoten genau einmal enthält), so dass die Summe der Gewichte der Kanten entlang der Rundreise minimal ist.

Ein Ziel der Bioinformatik: Gene-Clustering

- Im Rahmen einer Untersuchung wird für eine große Anzahl Gene untersucht, wie stark sie unter verschiedenen Bedingungen exprimiert werden,
- was für jedes Gen einen Eigenschaftsvektor liefert.
- Ziel ist es nun, Cluster von Genen zu bilden, deren Eigenschaftsvektoren »zusammengehören«.
- Solches »Clustering« ist in hochdimensionalen Räumen eine Kunst für sich.

Beispiel aus der Literatur zum Gene-Clustering-Problem.

Zitat aus

S. Climer und W. Zhang.
A traveling salesman's approach to clustering gene expression data,
Technischer Bericht WUCSE-2005-5, *Washington University in St. Louis*, 2005.

»The dataset consists of 2,467 genes in the budding yeast Saccharomyces cerevisiae that were studied during the diauxic shift, mitotic cell division cycle, sporulation, and temperature and reducing shocks, yielding 79 measurements that are used as the features for the genes.«

Vorgehensweise zum Gene-Clustering.

S. Climer und W. Zhang.
A traveling salesman's approach to clustering gene expression data,
Technischer Bericht WUCSE-2005-5, *Washington University in St. Louis*, 2005.

Vorgehen in obiger Arbeit:

1. Messe für jedes Gen sein Verhalten unter verschiedenen Einflüssen.
2. Dies liefert für jedes Gen einen *Eigenschaftsvektor*.
3. Berechne für jedes Genpaar eine *»Distanz«*, basierend auf der »Ähnlichkeit« der Eigenschaftsvektoren.
4. Dies induziert einen so genannte *Metrik* auf den Genen. Vereinfachend stellen wir uns die Gene in der Ebene vor und die Metrik ist durch die Entfernungen gegeben.
5. Löse das Handlungsreisenden-Problem für die Gene.
6. Alle aufeinanderfolgenden Städte mit kleiner Distanz bilden einen Cluster.

Was haben Handlungsreisende mit Gene-Clustering zu tun?

Idee

Eine kürzeste Rundreise wird nicht oft zwischen den Clustern »herumspringen«.

Zusammenfassung

Graph

Graphen bestehen aus *Knoten* und *Kanten* zwischen Paaren von Knoten. Wichtige Arten von Graphen sind:

»gerichtet« Die Kanten haben eine »Richtung«, gehen also »von« einem Knoten »zu« einem Knoten.

»ungerichtet« Die Kanten haben keine »Richtung«.

»gewichtet« Eine »Wichtung« ist eine Zahl, die die »Bedeutung« einer Kante oder einem Knoten angibt. Man kann sowohl »knotengewichtete« wie auch »kantengewichtete« Graphen betrachten.

»gelabelt« Ein »Label« ist ein kleiner Text, der an Kanten oder Knoten »geklebt« ist.

Implementation in Java

Man kann Graphen als *Adjazenmatrizen* implementieren oder mit Hilfe von *Adjazenzlisten*.

Zusammenfassung

Graphprobleme

Ein *Graphproblem* ist ein Berechnungsproblem, bei dem Graphen die Eingaben sind. Drei typische Beispiele sind:

- Das Finden kürzester Wege.
- Das Finden eines minimalen Gerüsts.
- Das Finden einer minimalen Rundreise.