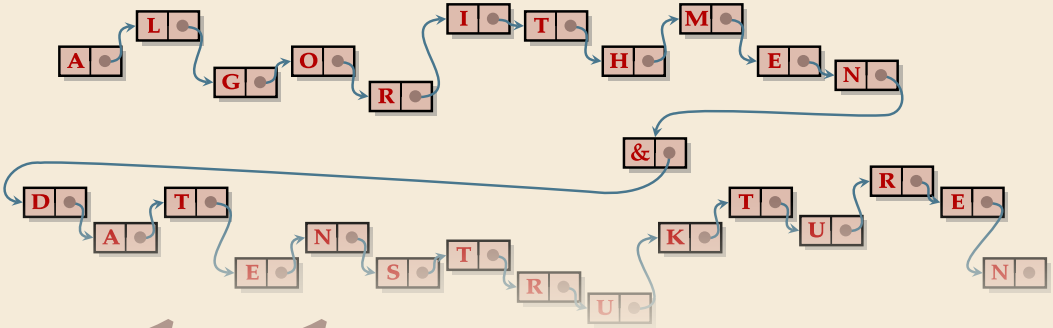


11 Graphen

Algorithmen & Datenstrukturen · Sommersemester 2026

Prof. Dr. Sebastian Wild

CS210 (Sommersemester 2026)
Philipps-Universität Marburg
version `2026-06-29 10:10`

Outline

11 Graphen

11.1 Wo(zu er)finden wir Graphen?

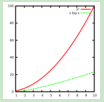
Clicker Question

Geben Sie alle zusammenpassenden Paare an:

(A) Graph

(B) Graf

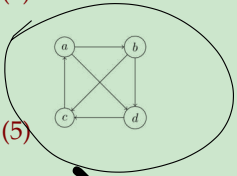
(C) Grave

(1) 

(2) 

(3) 

(4) 

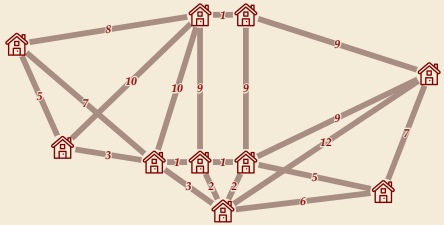
(5) 

(6) à

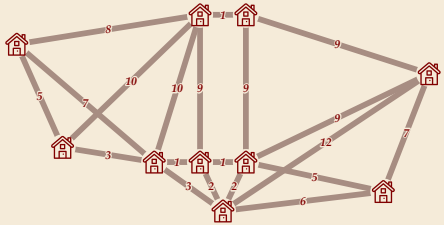
→ *sli.do/cs210*

Graphen

- Ein ***Graph*** ist eine Abstraktion für Daten, bei denen es (primär) um ***Entitäten*** und ihre paarweisen ***Beziehungen*** geht

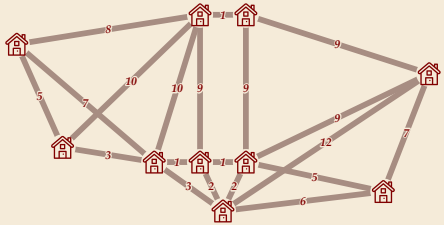


Graphen



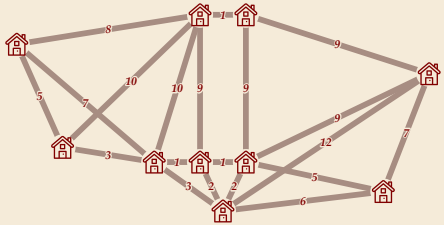
- Ein ***Graph*** ist eine Abstraktion für Daten, bei denen es (primär) um ***Entitäten*** und ihre paarweisen ***Beziehungen*** geht
- finden sich in vielfältiger Form
- In Anwendungen auch **Netzwerke** genannt (für uns synonym zu „Graph“)

Graphen



- Ein ***Graph*** ist eine Abstraktion für Daten, bei denen es (primär) um ***Entitäten*** und ihre paarweisen ***Beziehungen*** geht
- finden sich in vielfältiger Form
- In Anwendungen auch **Netzwerke** genannt (für uns synonym zu „Graph“)
- **Beispiele** für Graphen/Netwerke
  - **soziale Netzwerke:** z. B. Personen und Freundschaftsbeziehungen, . . .
  - **physische Netzwerke:** Straßen, Stromleitungen, das Internet (Computernetzwerk), . . .
  - **Informationsnetzwerke:** das World Wide Web, Ontologien, Protein-Interaktionen, . . .

Graphen

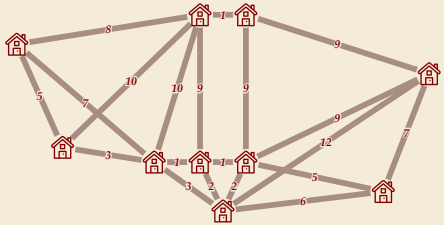


- Ein ***Graph*** ist eine Abstraktion für Daten, bei denen es (primär) um ***Entitäten*** und ihre paarweisen ***Beziehungen*** geht
- finden sich in vielfältiger Form
- In Anwendungen auch **Netzwerke** genannt (für uns synonym zu „Graph“)
- **Beispiele** für Graphen/Netwerke
  - **soziale Netzwerke:** z. B. Personen und Freundschaftsbeziehungen, . . .
  - **physische Netzwerke:** Straßen, Stromleitungen, das Internet (Computernetzwerk), . . .
  - **Informationsnetzwerke:** das World Wide Web, Ontologien, Protein-Interaktionen, . . .

OK, alles Entitäten und Verbindungen . . . aber das sind so unterschiedlichen Bereiche, haben die denn überhaupt irgendwas gemeinsam?

(Ergibt es Sinn, die alle über einen Kamm zu scheren?)

Graphen



- Ein ***Graph*** ist eine Abstraktion für Daten, bei denen es (primär) um ***Entitäten*** und ihre paarweisen ***Beziehungen*** geht
- finden sich in vielfältiger Form
- In Anwendungen auch **Netzwerke** genannt (für uns synonym zu „Graph“)
- **Beispiele** für Graphen/Netwerke
  - **soziale Netzwerke:** z. B. Personen und Freundschaftsbeziehungen, . . .
  - **physische Netzwerke:** Straßen, Stromleitungen, das Internet (Computernetzwerk), . . .
  - **Informationsnetzwerke:** das World Wide Web, Ontologien, Protein-Interaktionen, . . .

OK, alles Entitäten und Verbindungen . . . aber das sind so unterschiedlichen Bereiche, haben die denn überhaupt irgendwas gemeinsam?

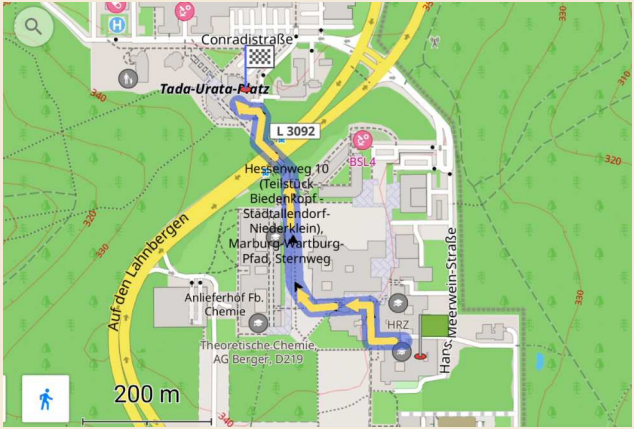
(Ergibt es Sinn, die alle über einen Kamm zu scheren?)

Ja! Abstraktion offenbart wiederkehrenden Fragestellungen aus verschiedenen Bereichen
⇝ **Graph-Algorithmen liefern wiederverwendbare Lösungen!**

Anwendung 1: Routenplanung

Ziel Finde kürzesten Weg von Start zu Ziel
Gegeben: Straßenkarte, Startpunkt, Zielpunkt

- Entität = Straßenkreuzung
- Beziehung = Straße
- i. d. R. bidirektional (außer Einbahnstraßen!)
- Beziehung ist gewichtet z. B. Fahrtzeit



Anwendung 2: Page Rank

1990s Suchmaschinen: erstes Suchergebnis $\approx$ Seite, die Suchbegriff am häufigsten erwähnt

Da fällt mir doch eine „Optimierung“ für meine (100% legitime) Website ein . . .

Anwendung 2: Page Rank

1990s Suchmaschinen: erstes Suchergebnis $\approx$ Seite, die Suchbegriff am häufigsten erwähnt

Da fällt mir doch eine „Optimierung“ für meine (100% legitime) Website ein . . .

Google's erste Generation von Ranking: ***Page Rank***

- Idee wichtige / relevante / vertrauenswürdige Seite $\rightsquigarrow$ viele Links **zu** dieser Seite

Anwendung 2: Page Rank

1990s Suchmaschinen: erstes Suchergebnis ≈ Seite, die Suchbegriff am häufigsten erwähnt

Da fällt mir doch eine „Optimierung“ für meine (100% legitime) Website ein . . .

Google's erste Generation von Ranking: ***Page Rank***

- Idee wichtige / relevante / vertrauenswürdige Seite ⇝ viele Links **zu** dieser Seite

⇝ *Verwende Links zwischen Seiten selbst als Hauptkriterium für Relevanz!*

Anwendung 2: Page Rank

1990s Suchmaschinen: erstes Suchergebnis ≈ Seite, die Suchbegriff am häufigsten erwähnt

Da fällt mir doch eine „Optimierung“ für meine (100% legitime) Website ein . . .

Google’s erste Generation von Ranking: ***Page Rank***

- Idee wichtige / relevante / vertrauenswürdige Seite ⇝ viele Links **zu** dieser Seite

⇝ *Verwende Links zwischen Seiten selbst als Hauptkriterium für Relevanz!*

- ***Random-Surfer-Metapher***
  - Random Surfer klickt stets auf einen zufällig gewählten Link der aktuellen Seite (oder startet mit kleiner Wahrscheinlichkeit neu auf zufälliger Seite)
  - Relevanz = Anteil der Zeit auf dieser Seite

Anwendung 2: Page Rank

1990s Suchmaschinen: erstes Suchergebnis ≈ Seite, die Suchbegriff am häufigsten erwähnt

Da fällt mir doch eine „Optimierung“ für meine (100% legitime) Website ein . . .

Google’s erste Generation von Ranking: ***Page Rank***

- Idee wichtige / relevante / vertrauenswürdige Seite ⇝ viele Links **zu** dieser Seite

⇝ *Verwende Links zwischen Seiten selbst als Hauptkriterium für Relevanz!*

- ***Random-Surfer-Metapher***
  - Random Surfer klickt stets auf einen zufällig gewählten Link der aktuellen Seite (oder startet mit kleiner Wahrscheinlichkeit neu auf zufälliger Seite)
  - Relevanz = Anteil der Zeit auf dieser Seite
  - ⇝ cleverer Algorithmus berechnet das als *steady state* einer *Markovkette*
  - ⇝ clevere Implementierung schafft das verteilt im Cluster für Millarden von Seiten

Anwendung 2: Page Rank

1990s Suchmaschinen: erstes Suchergebnis ≈ Seite, die Suchbegriff am häufigsten erwähnt

Da fällt mir doch eine „Optimierung“ für meine (100% legitime) Website ein . . .

Google's erste Generation von Ranking: ***Page Rank***

- Idee wichtige / relevante / vertrauenswürdige Seite ⇝ viele Links **zu** dieser Seite

⇝ *Verwende Links zwischen Seiten selbst als Hauptkriterium für Relevanz!*

- ***Random-Surfer-Metapher***
  - Random Surfer klickt stets auf einen zufällig gewählten Link der aktuellen Seite (oder startet mit kleiner Wahrscheinlichkeit neu auf zufälliger Seite)
  - Relevanz = Anteil der Zeit auf dieser Seite
  - ⇝ cleverer Algorithmus berechnet das als *steady state* einer *Markovkette*
  - ⇝ clevere Implementierung schafft das verteilt im Cluster für Millarden von Seiten

⇝ *Google's initialer Erfolg beruht auf der Abstraktion des WWW zu einem Graphen!*

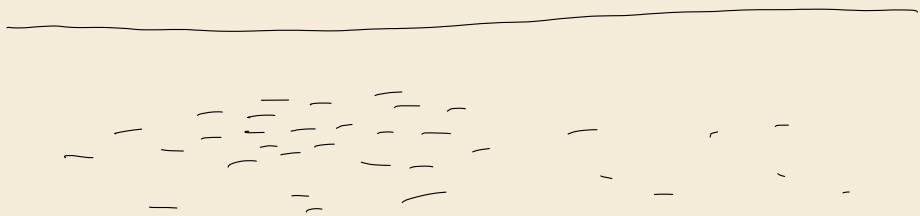
- Entität = Website; Beziehung = Hyperlinks
- Beziehungen haben feste Richtung (asymmetrisch), aber keine Gewichtung

Anwendung 3: de-Bruijn-Graphs, Overlap-Graphs

- Das **Genom** (eines Individuums) zu **sequenzieren** ist mittlerweile kommerziell möglich

Anwendung 3: de-Bruijn-Graphs, Overlap-Graphs

- Das **Genom** (eines Individuums) zu **sequenzieren** ist mittlerweile kommerziell möglich
- Labor-Techniken liefern aber nur kurze „Schnipsel“ des Genoms
  - 100 – 5000 Basenpaare (Zeichen) pro Read (je nach Technologie)
  - Menschliches Genom hat etwa 3 000 000 000 Basenpaare! (für einen Chromosomensatz!)
  - bekommen Tausende Kopien, die an zufälligen Stellen zerbrochen wurden



Anwendung 3: de-Bruijn-Graphs, Overlap-Graphs

- Das **Genom** (eines Individuums) zu **sequenzieren** ist mittlerweile kommerziell möglich
- Labor-Techniken liefern aber nur kurze „Schnipsel“ des Genoms
  - 100 – 5000 Basenpaare (Zeichen) pro Read (je nach Technologie)
  - Menschliches Genom hat etwa 3 000 000 000 Basenpaare! (für einen Chromosomensatz!)
  - bekommen Tausende Kopien, die an zufälligen Stellen zerbrochen wurden

⇝ benötigen Algorithmen, die daraus das Genom **rekonstruierten** (*genome assembly*)

Anwendung 3: de-Bruijn-Graphs, Overlap-Graphs

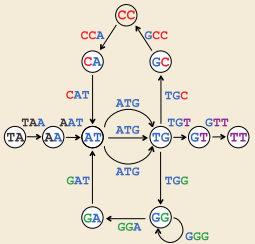
- Das **Genom** (eines Individuums) zu **sequenzieren** ist mittlerweile kommerziell möglich
- Labor-Techniken liefern aber nur kurze „Schnipsel“ des Genoms
  - 100 – 5000 Basenpaare (Zeichen) pro Read (je nach Technologie)
  - Menschliches Genom hat etwa 3 000 000 000 Basenpaare! (für einen Chromosomensatz!)
  - bekommen Tausende Kopien, die an zufälligen Stellen zerbrochen wurden

⇝ benötigen Algorithmen, die daraus das Genom **rekonstruierten** (*genome assembly*)

- Kern-Algorithmen modellieren das Genome-Assembly-Problem als Graph-Problem
  - Entitäten = Teilstrings

    (für jeden Read einmal ohne erstes Zeichen, einmal ohne letztes Zeichen)
  - Beziehung = Read der Teilstrings verbindet
  - Beziehung hat Richtung, aber keine Gewichtung
  - Ziel: eindeutiger Weg durch Graph mit allen Reads

DeBruijn$_3$(**TAATGCCATGGGATGTT**)



Compeau & Pevzner, *Bioinformatics Algorithms*, Fig. 4.1

https://cogniterra.org/lesson/29910/step/2?unit=22007

11.2 Terminologie

Kernbegriffe

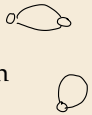
- **Knoten** (***vertex***) = Entitäten im Graph
  - Synonyme: Ecke, Punkt; *node*, *point*
  - Formelsprache: Knoten $v \in V$ (Menge aller Knoten eines Graphen)
- **Kante** (***edge***) = Beziehung zwischen zwei Knoten
  - Synonyme: Pfeil, Linie, Relation; *arc*
  - Formelsprache: Kante $e \in E$ (Menge aller Kanten eines Graphen)
- Graph = Knoten und Kanten
  - Synonym: Netzwerk
  - Formelsprache: $G = (V, E)$

Graphen in allen „Geschmacksrichtungen“

- Graphen modellieren sehr diverse Entitäten und Beziehungen

⇝ mehrere verschiedene (aber wiederkehrende!) Varianten

Eigenschaft	**erfüllt**	**nicht erfüllt**
Kanten sind Einbahnstraßen	*gerichteter* Graph (*directed graph*, *Digraph*)	*ungerichteter* Graph
≤ 1 Kante zwischen u und v	*einfacher* Graph	*Multigraph* G. mit *parallelen* Kanten
Kanten von v zu v	mit *Schleifen* (*loops*)	(schleifenfrei)
Gewichtung auf Kanten	*(Kanten-) gewichteter* G.	*ungewichteter* Graph



🙃 jegliche Kombination kann Sinn ergeben . . .

Graphen in allen „Geschmacksrichtungen“

- Graphen modellieren sehr diverse Entitäten und Beziehungen

⇝ mehrere verschiedene (aber wiederkehrende!) Varianten

Eigenschaft	**erfüllt**	**nicht erfüllt**
Kanten sind Einbahnstraßen	*gerichteter* Graph (*directed graph*, *Digraph*)	*ungerichteter* Graph
≤ 1 Kante zwischen u und v	*einfacher* Graph	*Multigraph* G. mit *parallelen* Kanten
Kanten von v zu v	mit *Schleifen* (*loops*)	(schleifenfrei)
Gewichtung auf Kanten	*(Kanten-) gewichteter* G.	*ungewichteter* Graph

🙃 jegliche Kombination kann Sinn ergeben . . .

⇝ Man muss angeben, welche Art von Graph gemeint ist!

- Oft sind aber die Algorithmen ähnlich
- kann verwirren, aber macht Methoden wiederverwertbar

Ungerichtete Graphen – Formale Definition

- Standard-Annahme: Graph ist ungewichtet, ungerichtet, schleifenfrei & einfach
- *Graph* $G = (V, E)$ mit
  - V endliche Menge von *Knoten*
  - $E \subseteq [V]^2$ eine Menge von *Kanten*, mit $[V]^2 = \{e : e \subseteq V \wedge |e| = 2\}$

2-elementige Teilmengen

Ungerichtete Graphen – Formale Definition

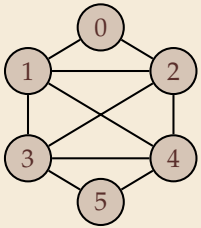
- Standard-Annahme: Graph ist ungewichtet, ungerichtet, schleifenfrei & einfach
- *Graph* $G = (V, E)$ mit
  - V endliche Menge von *Knoten*
  - $E \subseteq [V]^2$ eine Menge von *Kanten*, mit $[V]^2 = \{e : e \subseteq V \wedge |e| = 2\}$

2-elementige Teilmengen

Beispiel

$$V = \{0,1,2,3,4,5\}$$
$$E = \{\{0,1\},\{1,2\},\{1,4\},\{1,3\},\{0,2\},\{2,4\},\{2,3\},\{3,4\},\{3,5\},\{4,5\}\}.$$

Graphische Darstellung



so . . .

Ungerichtete Graphen – Formale Definition

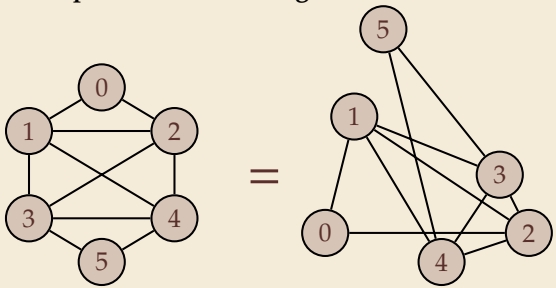
- Standard-Annahme: Graph ist ungewichtet, ungerichtet, schleifenfrei & einfach
- *Graph* $G = (V, E)$ mit
  - V endliche Menge von *Knoten*
  - $E \subseteq [V]^2$ eine Menge von *Kanten*, mit $[V]^2 = \{e : e \subseteq V \wedge |e| = 2\}$

2-elementige Teilmengen

Beispiel

$$
\begin{aligned}
V &= \{0,1,2,3,4,5\} \\
E &= \big\{\{0,1\},\{1,2\},\{1,4\},\{1,3\},\{0,2\}, \\
&\qquad \{2,4\},\{2,3\},\{3,4\},\{3,5\},\{4,5\}\big\}.
\end{aligned}
$$

Graphische Darstellung



so . . .

. . . oder auch so

(gleicher Graph!)

Eine „hilfreiche“ Anordnung der Knoten findet man schönsten dynamisch

`https://csacademy.com/app/graph_editor`

Gerichtete Graphen – Formale Definition

- Standard-Annahme: Digraph ist ungewichtet, schleifenfrei & einfach
- *Digraph / Gerichteter Graph* $G = (V, E)$ mit
  - V endliche Menge von *Knoten*
  - $E \subseteq V^2 \setminus \{(v, v) : v \in V\}$ Menge *(gerichteter) Kanten*, ohne Schleifen
    $V^2 = V \times V = \{(x, y) : x \in V \wedge y \in V\}$ 2-Tuple / (geordnete) Paare aus V

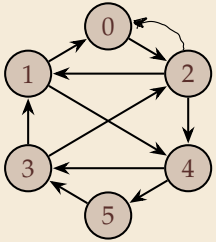
Gerichtete Graphen – Formale Definition

- Standard-Annahme: Digraph ist ungewichtet, schleifenfrei & einfach
- *Digraph / Gerichteter Graph* $G = (V, E)$ mit
  - V endliche Menge von *Knoten*
  - $E \subseteq V^2 \setminus \{(v, v) : v \in V\}$ Menge *(gerichteter) Kanten*,
    $V^2 = V \times V = \{(x, y) : x \in V \wedge y \in V\}$ 2-Tuple / (geordnete) Paare aus V

Beispiel

$$V = \{0, 1, 2, 3, 4, 5\}$$
$$E = \{(0,2), (1,0), (1,4), (2,1), (2,4), (2,0), (3,1), (3,2), (4,3), (4,5), (5,3)\}$$

Graphische Darstellung



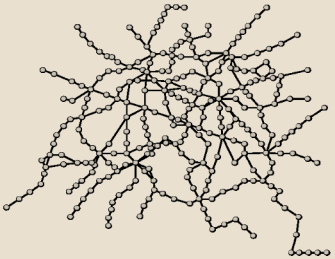
11.3 Ungerichtete Graphen

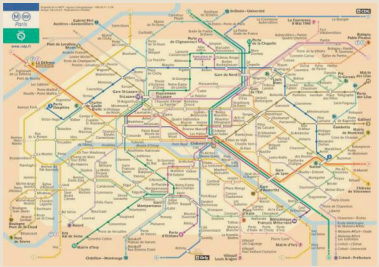
Undirected graphs

Graph. Set of vertices connected pairwise by edges.

Why study graph algorithms?

- Thousands of practical applications.
- Hundreds of graph algorithms known.
- Interesting and broadly useful abstraction.
- Challenging branch of computer science and discrete math.



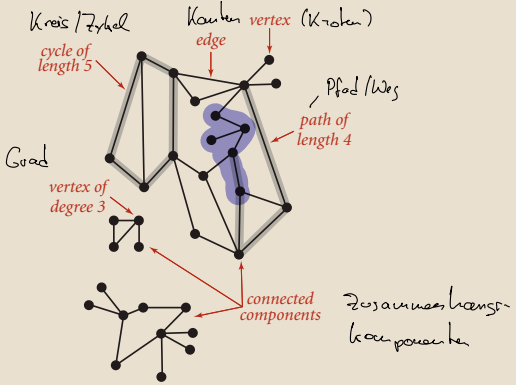


Graph terminology

Path. Sequence of vertices connected by edges.

Cycle. Path whose first and last vertices are the same.

Two vertices are connected if there is a path between them.



Some graph-processing problems

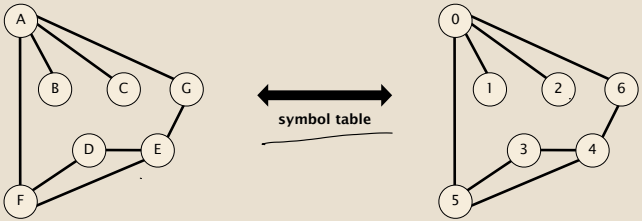
problem	description
s–t path	*Is there a path between s and t ?*
shortest s–t path	*What is the shortest path between s and t ?*
cycle	*Is there a cycle in the graph ?*
Euler cycle	*Is there a cycle that uses each edge exactly once ?*
Hamilton cycle	*Is there a cycle that uses each vertex exactly once ?*
connectivity	*Is there a way to connect all of the vertices ?*
biconnectivity	*Is there a vertex whose removal disconnects the graph ?*
planarity	*Can the graph be drawn in the plane with no crossing edges ?*
graph isomorphism	*Do two adjacency lists represent the same graph ?*

Challenge. Which graph problems are easy? difficult? intractable?

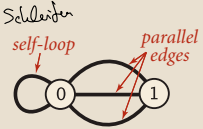
Graph representation

Vertex representation.

- This lecture: use integers between 0 and $V-1$.
- Applications: convert between names and integers with symbol table.



Anomalies.



Graph API

public class Graph	
Graph(int V)	*create an empty graph with V vertices*
Graph(In in)	*create a graph from input stream*
void addEdge(int v, int w)	*add an edge v-w*
Iterable<Integer> adj(int v)	*vertices adjacent to v*
int V()	*number of vertices*
int E()	*number of edges*

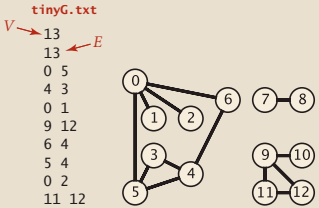
```
In in = new In(args[0]);
Graph G = new Graph(in);

for (int v = 0; v < G.V(); v++)
   for (int w : G.adj(v))
      StdOut.println(v + "-" + w);
```

read graph from input stream

print out each edge (twice)

Graph API: sample client

Graph input format.




```
% java Test tinyG.txt
0-6
0-2
0-1
0-5
1-0
2-0
3-5
3-4
⋮
12-11
12-9
```

```
In in = new In(args[0]);
Graph G = new Graph(in);

for (int v = 0; v < G.V(); v++)
   for (int w : G.adj(v))
      StdOut.println(v + "-" + w);
```

read graph from input stream

print out each edge (twice)

Typical graph-processing code

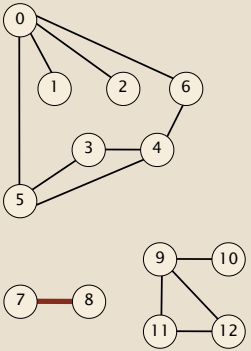
public class Graph		
	Graph(int V)	*create an empty graph with V vertices*
	Graph(In in)	*create a graph from input stream*
void	addEdge(int v, int w)	*add an edge v-w*
Iterable<Integer>	adj(int v)	*vertices adjacent to v*
int	V()	*number of vertices*
int	E()	*number of edges*

```
// degree of vertex v in graph G
public static int degree(Graph G, int v)
{
    int degree = 0;
    for (int w : G.adj(v))
       degree++;
    return degree;
}
```

11.4 Graph Repräsentationen

Set-of-edges graph representation

Maintain a list of the edges (linked list or array).



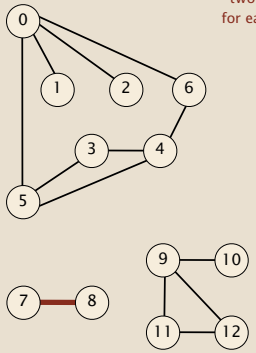
0	1
0	2
0	5
0	6
3	4
3	5
4	5
4	6
7	8
9	10
9	11
9	12
11	12

Q. How long to iterate over vertices adjacent to v ?

Adjazenzmatrix

Adjacency-matrix graph representation

Maintain a two-dimensional V-by-V boolean array;
for each edge v–w in graph: `adj[v][w] = adj[w][v] = true`.



two entries for each edge

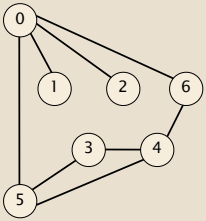
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

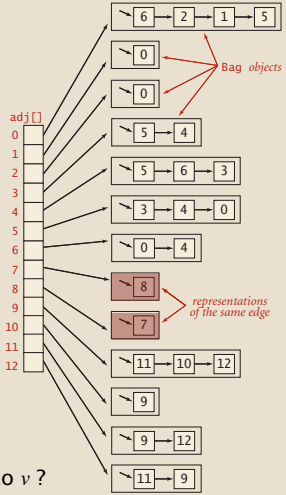
Q. How long to iterate over vertices adjacent to v ?

Adjazenzlisten

Adjacency-list graph representation

Maintain vertex-indexed array of lists.





Q. How long to iterate over vertices adjacent to v ?

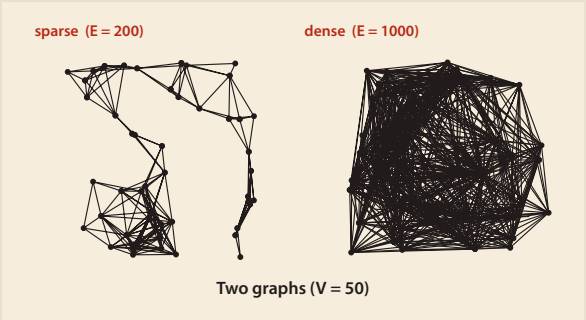
Graph representations

In practice. Use adjacency-lists representation.

- Algorithms based on iterating over vertices adjacent to v.
- Real-world graphs tend to be sparse.

dicht oder dünn besetzt

huge number of vertices,
small average vertex degree



Two graphs (V = 50)

Graph representations

In practice. Use adjacency-lists representation.

- Algorithms based on iterating over vertices adjacent to v.
- Real-world graphs tend to be sparse.

huge number of vertices, small average vertex degree

representation	space	add edge	edge between v and w?	iterate over vertices adjacent to v?
list of edges	E	1	E	E
adjacency matrix	V^2	1 *	1	V
adjacency lists	$E + V$	1	$degree(v)$	$degree(v)$

* disallows parallel edges

Adjacency-list graph representation: Java implementation

```
public class Graph
{
   private final int V;
   private Bag<Integer>[] adj;

   public Graph(int V)
   {
      this.V = V;
      adj = (Bag<Integer>[]) new Bag[V];
      for (int v = 0; v < V; v++)
         adj[v] = new Bag<Integer>();
   }

   public void addEdge(int v, int w)
   {
      adj[v].add(w);
      adj[w].add(v);
   }

   public Iterable<Integer> adj(int v)
   {  return adj[v];  }
}
```

adjacency lists
(using Bag data type)

create empty graph
with V vertices

add edge v-w
(parallel edges and
self-loops allowed)

iterator for vertices adjacent to v

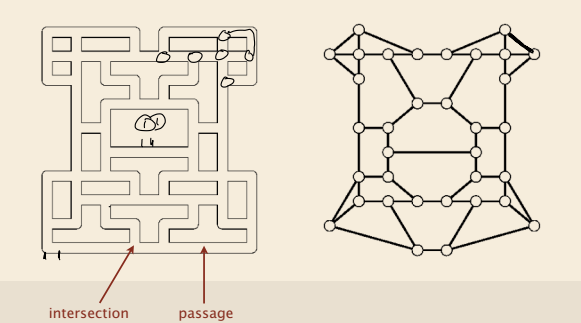
11.5 Tiefensuche

Depth-First Search

Maze exploration

Maze graph.

- Vertex = intersection.
- Edge = passage.

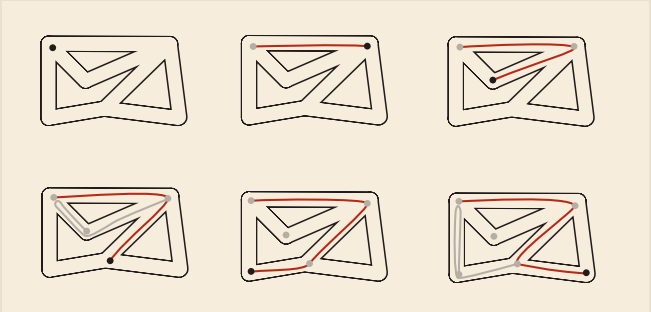


Goal. Explore every intersection in the maze.

Trémaux maze exploration

Algorithm.

- Unroll a ball of string behind you.
- Mark each visited intersection and each visited passage.
- Retrace steps when no unvisited options.



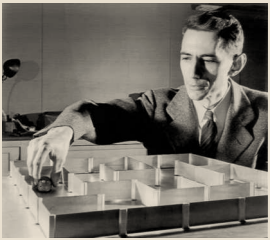
Trémaux maze exploration

Algorithm.

- Unroll a ball of string behind you.
- Mark each visited intersection and each visited passage.
- Retrace steps when no unvisited options.

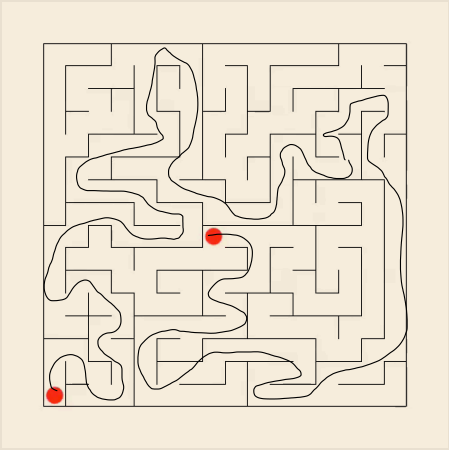
First use? Theseus entered Labyrinth to kill the monstrous Minotaur;
Ariadne instructed Theseus to use a ball of string to find his way back out.

The Labyrinth (with Minotaur)

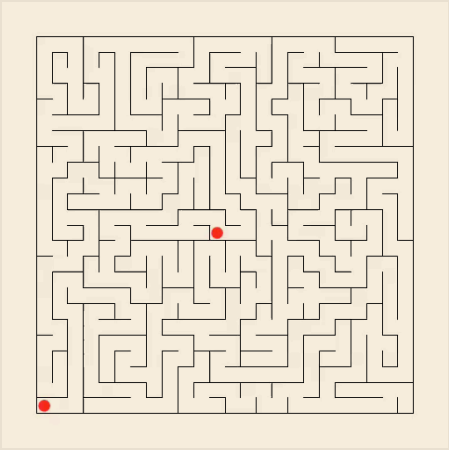


Claude Shannon (with Theseus mouse)

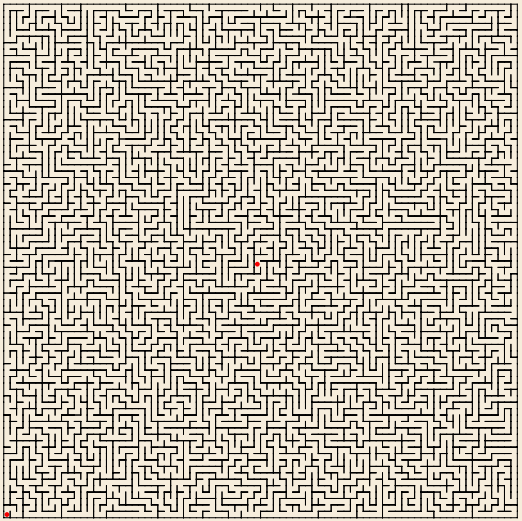
Maze exploration: easy



Maze exploration: medium



Maze exploration: challenge for the bored



Design pattern for graph processing

Design pattern. Decouple graph data type from graph processing.

- Create a `Graph` object.
- Pass the `Graph` to a graph-processing routine.
- Query the graph-processing routine for information.

public class Paths		
	`Paths(Graph G, int s)`	*find paths in G from source s*
`boolean`	`hasPathTo(int v)`	*is there a path from s to v?*
`Iterable<Integer>`	`pathTo(int v)`	*path from s to v; null if no such path*

```
Paths paths = new Paths(G, s);
for (int v = 0; v < G.V(); v++)
   if (paths.hasPathTo(v))
      StdOut.println(v);
```

print all vertices connected to s