Discrete mathematics

Petr Kovář & Tereza Kovářová
petr.kovar@vsb.cz

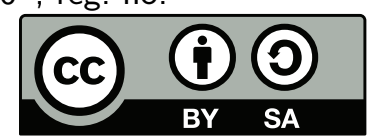
VŠB – Technical University of Ostrava

Winter Term 2022/2023
DiM 470-2301/02, 470-2301/04, 470-2301/06

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION,
YOUTH AND SPORTS

The translation was co-financed by the European Union and the Ministry of Education, Youth and Sports from the Operational Programme Research, Development and Education, project "Technology for the Future 2.0", reg. no. CZ.02.2.69/0.0/0.0/18_058/0010212.



About this file

This file is meant to be a guideline for the lecturer. Many important pieces of information are not in this file, they are to be delivered in the lecture: said, shown or drawn on board. The file is made available with the hope students will easier catch up with lectures they missed.

For study the following resources are better suitable:

- Meyer: Lecture notes and readings for an http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-fall-2005/readings/" (weeks 1-5, 8-10, 12-13), MIT, 2005.
- Diestel: Graph theory http://diestel-graph-theory.com/ (chapters 1-6), Springer, 2010.

See also http://homel.vsb.cz/~kov16/predmety_dm.php

Lecture overview

Chapter Connectivity of graphs

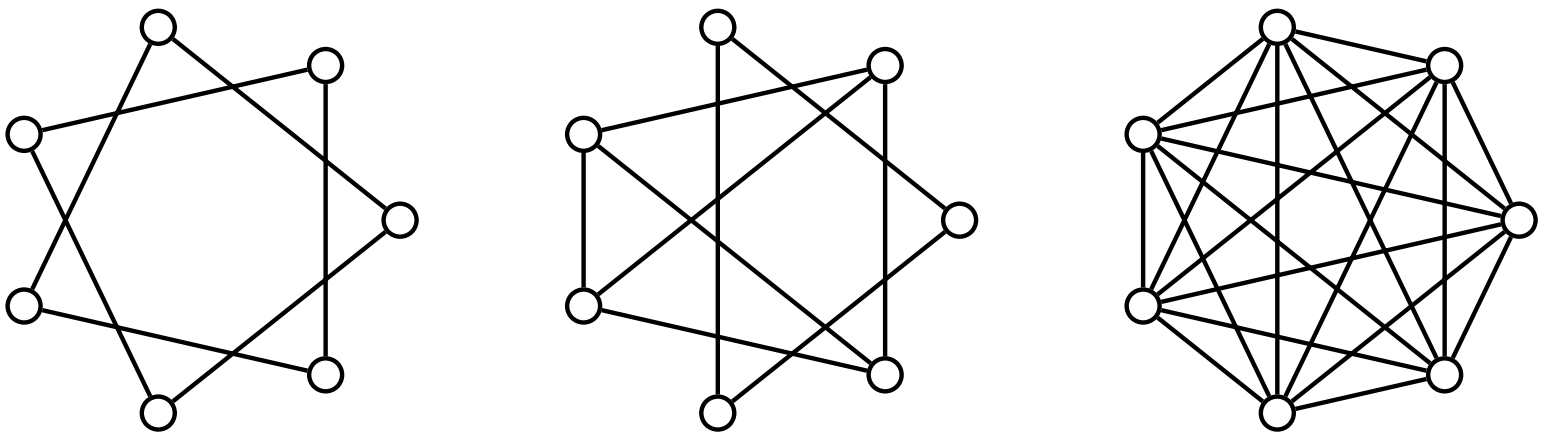
- motivation
- connectivity and components of a graph
- searching through a graph
- higher degrees of connectivity

Connectivity of graphs

If a graph represents a computer network or a road network, it is natural to examine, whether one can transmit a signal or send goods from vertex u to vertex v. This leads us to the notion of *connectivity of graphs*.
For similar reasons one can examine the robustness against local failures:

- vertex redundancy
- connectivity even in the case where several edges are cut

Hence we arrive at the notion of the *degree of vertex- and edge-connectivity*.



Connected and not connected graphs.

Connections in graphs, components

Informally: A graph is connected, if there exists a "connection" between every two vertices (not necessarily an edge).

Formally: we introduce the concept of a *walk*, *trail*, and *path* in s graph.

Definition

A v_0v_n walk in a graph G is such a sequence of vertices and edges

$$(v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n),$$

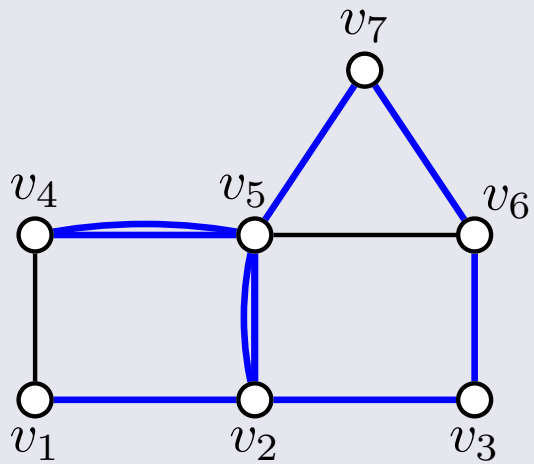
where v_i are vertices and e_i are edges of G such, that v_{i-1} and v_i are incident with e_i. The number of edges n is the length of the v_0v_n walk. v_0 is the starting vertex and v_n the end-vertex of the walk.

If there are no multiple edges, we can describe the walk by listing just a sequence of vertices.

$$(v_0, v_1, v_2, \ldots, v_n)$$

Alternatively we can omit the parentheses: $v_0, v_1, v_2, \ldots, v_n$.

Example

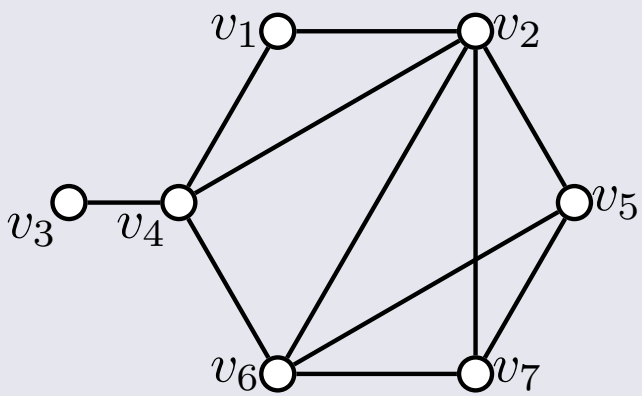


Walk $v_1, v_1v_2, v_2, v_2v_5, v_5, v_5v_7, v_7, v_7v_6, v_6, v_6v_3, v_3, v_3v_2, v_2, v_2v_5, v_5, v_5v_4, v_4, v_4v_5, v_5$ is highlighted in blue.

Briefly:

$v_1, v_2, v_5, v_7, v_6, v_3, v_2, v_5, v_4, v_5$.

Example



$v_1, v_2, v_6, v_7, v_2, v_1, v_2, v_3$ is not a walk

walk $v_1, v_2, v_6, v_7, v_2, v_1, v_2, v_4$

walk $v_1, v_2, v_7, v_5, v_6, v_4, v_3$

(trivial) walk v_4

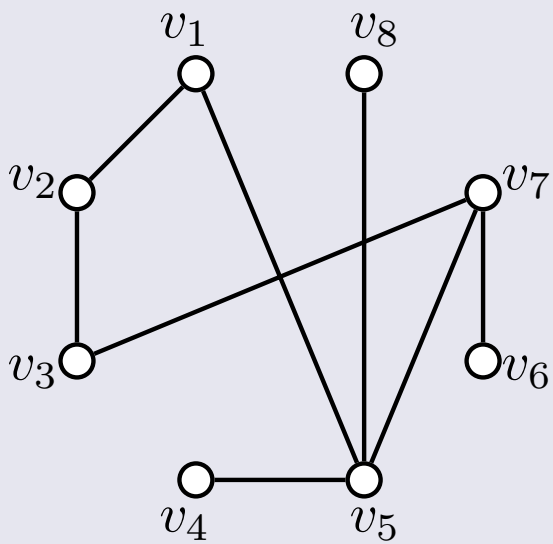
The notion of "connectivity" is based on the term "walk".

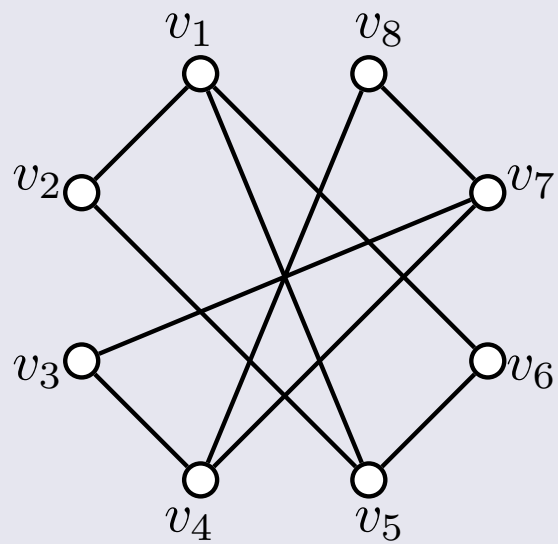
Definition

We say vertex v can be reached from vertex u, if there exists a uv walk in the given graph.
We say a graph is connected if for every pair of vertices u, v is vertex v reachable from vertex u. Otherwise the graph is not connected.

Example

Is each of the two graphs connected?





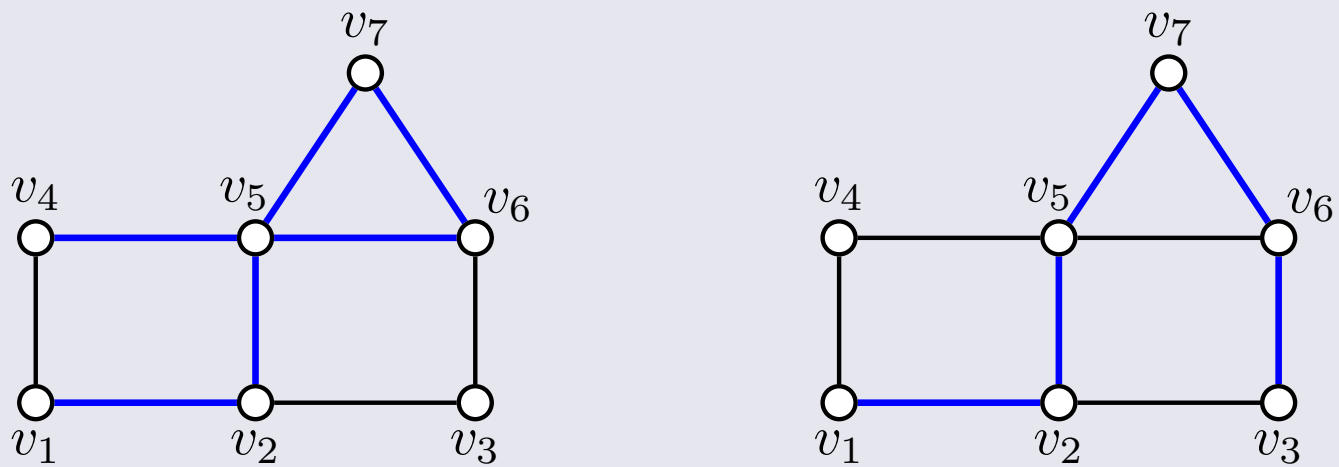
In some application the repetitions of edges or vertices is *not allowed* (pipes, traffic networks, electrical networks, . . .).

Definition

Trail is a walk with no repeated edges.
Path is a walk with no repeated vertices.

Terminology: we travel along trails, we draw "in one stroke".
Vertices and edges of a path in a graph form a subgraph that is a path.

Example



Trail $v_1, v_2, v_5, v_7, v_6, v_5, v_4$ *and a path* $v_1, v_2, v_5, v_7, v_6, v_3$.

Theorem

If there exists a uv walk in G, then there exists also a uv path in G.

Proof Let W be a uv walk $u = v_0, e_1, v_1, \ldots, e_n, v_n = v$ of length n in G. We want to find a uv walk P with no repeated vertex. If no vertex in W is repeated, then $P = W$ is the wanted path.
If a certain vertex v_i is repeated, we can omit the entire part of W between its first occurrence v_i and its last occurrence of, say v_k. We obtain a uv walk W', in which the vertex v_i occurs only once.
Now if no other vertex is repeated in W' we take $P = W'$. Otherwise repeat the process for the next repeated vertex.
The algorithm is deterministic, since there are only finitely many vertices in G. □

If there exists a uv walk in G, we can obtain a uv path by the proof of the theorem. We say that vertices u and v are joined by a path in G.

On the set of vertices of a given graph G we introduce the relation $\sim$. Two vertices $u, v \in V(G)$ are related in $\sim$ (we write $u \sim v$) if and only if there is uv walk in G.

We call $\sim$ a "relation of being reachable."

Lemma

The relation $\sim$ is an equivalence relation.

Proof

- Reflexivity follows from the existence of a trivial walk uu of length 0. For each vertex $u \in V(G)$ is $u \sim u$.
- Symmetry is obvious, since for each uv walk in G we can easily construct the vu walk in G by "reversing" the sequence of vertices and edges (in an unoriented graph). $\forall u, v \in V(G)$ is $u \sim v \Leftrightarrow v \sim u$.
- Transitivity follows from the fact, that by joining the walks $u, \ldots, v$ and $v, \ldots, w$ we obtain the walk $u, \ldots, w$. $\forall u, v, w \in V(G)$ is $u \sim v \wedge v \sim w \Rightarrow u \sim w$.

□

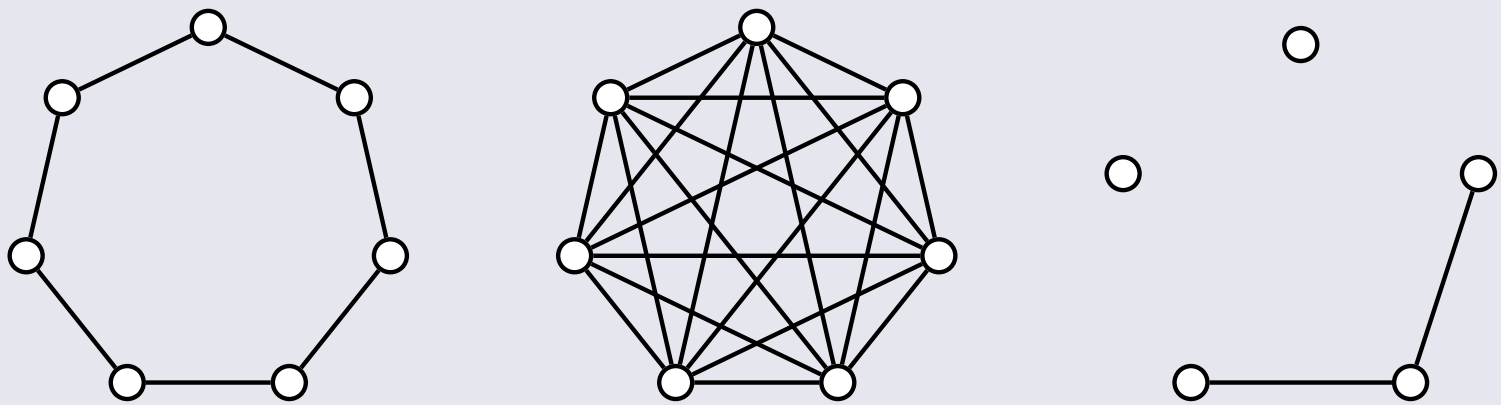
The relation $\sim$ from the Lemma above defines a partitioning of $V(G)$.

Now we can define the following:

Definition

Every maximal connected subgraph of graph G is a component of G.

Example



Examples of graphs with one and more components.

Two alternative definitions of connectivity follow.

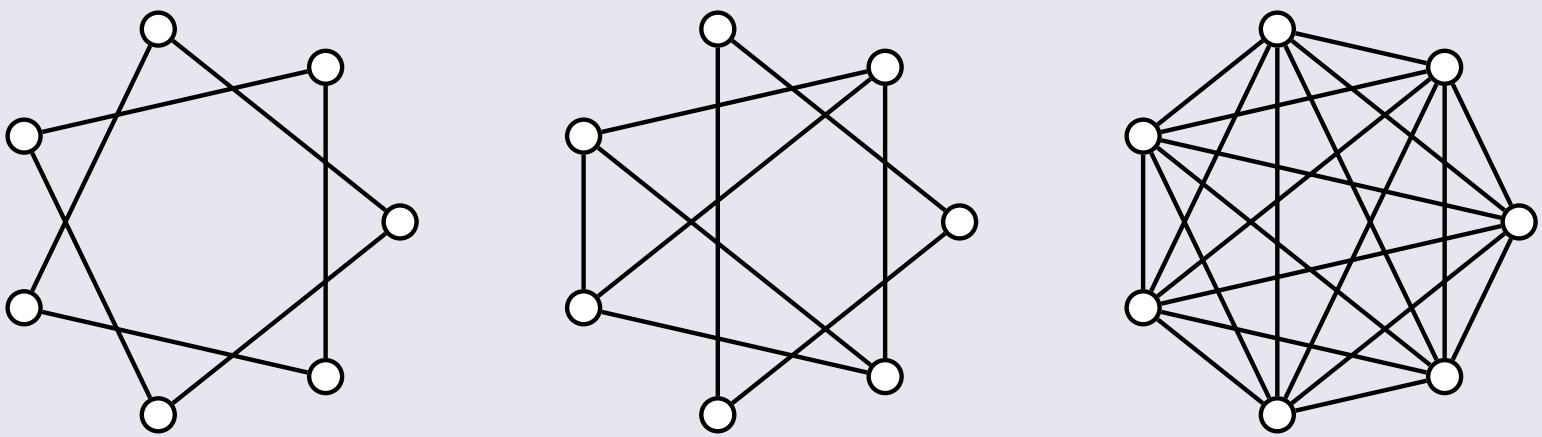
Definition

We say that a graph G is connected if it has only one component.

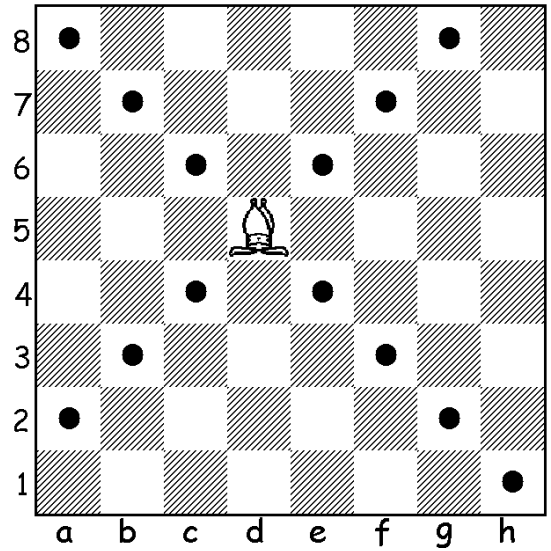
Equivalent definition

We say that G is connected if the $\sim$ relation on $V(G)$ is total.

Example



Examples of connected graphs and a not connected graph.



How bishop moves.

Example

Take the graph S, which describes all possible movements of a bishop on a chess board. (Recall that a bishop moves any number of vacant squares diagonally.)

Vertices correspond to squares and an edges joins two squares if and only if there is legal move of a bishop between them.

Graph S has 64 vertices, many edges and it is not connected.

Example

Loyds' fifteen puzzle is a classic. The task is to shuffle the pieces numbered 1 through 15 so that they form an arithmetic progression 1 through 15 by rows.

We construct a graph of states: vertices are all possible arrangements of the pieces and an edge joined two arrangements if there is a single valid move from one to the other. One can show that such graph is not connected and thus there is no solution to the puzzle!

Towers of Hanoi

We have three pegs and a set of discs of different sizes. All discs are on one peg arranged according their size. The task is to move all discs to another peg while

- always one discs is moved,
- never a larger disc can be on top of a smaller one.

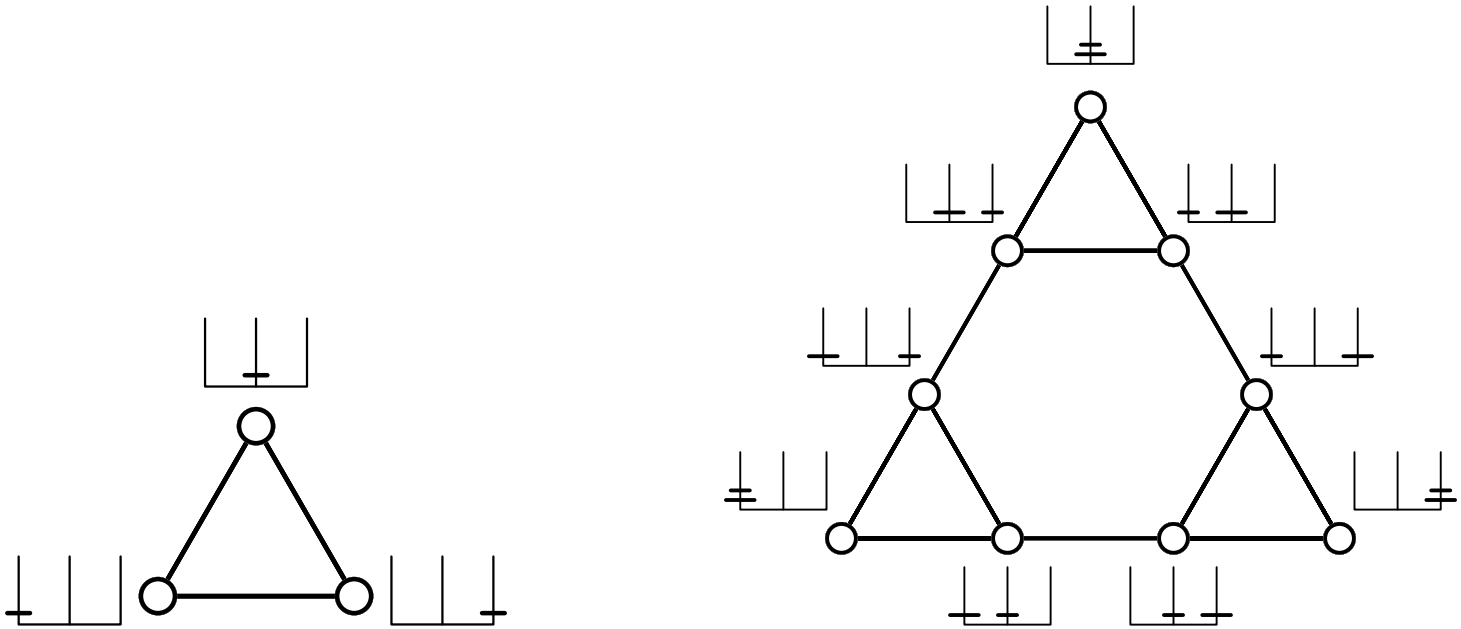
Is it possible? What is the least number of moves required?

In chapter on recurrence relations we found the minimum number of moves for n discs.

Graph formulation – state graph

For the solution we set up a state graph:

- vertices – each valid distribution of discs,
- edges – join two states with a valid move in between.

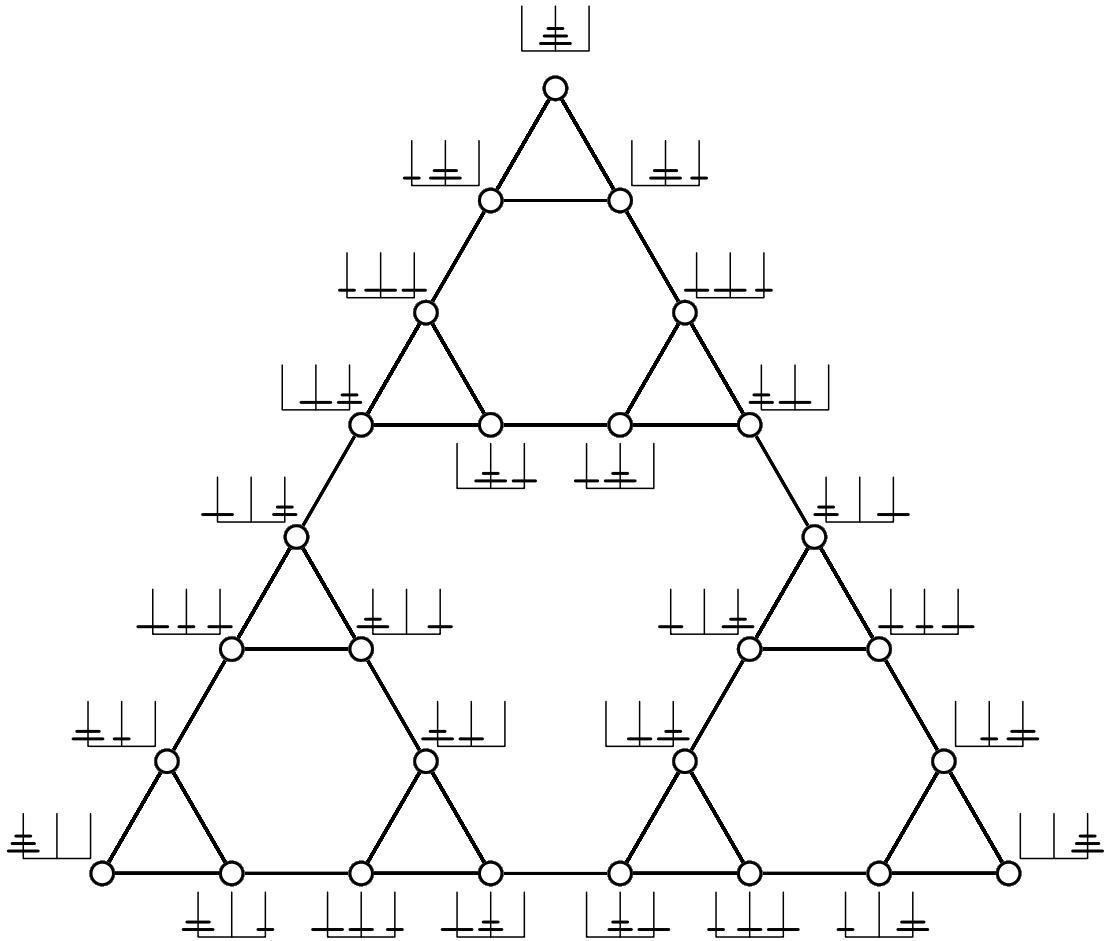


The puzzle with a single disc and with two discs.

For two discs we distinguish three cases where to put the larger discs.

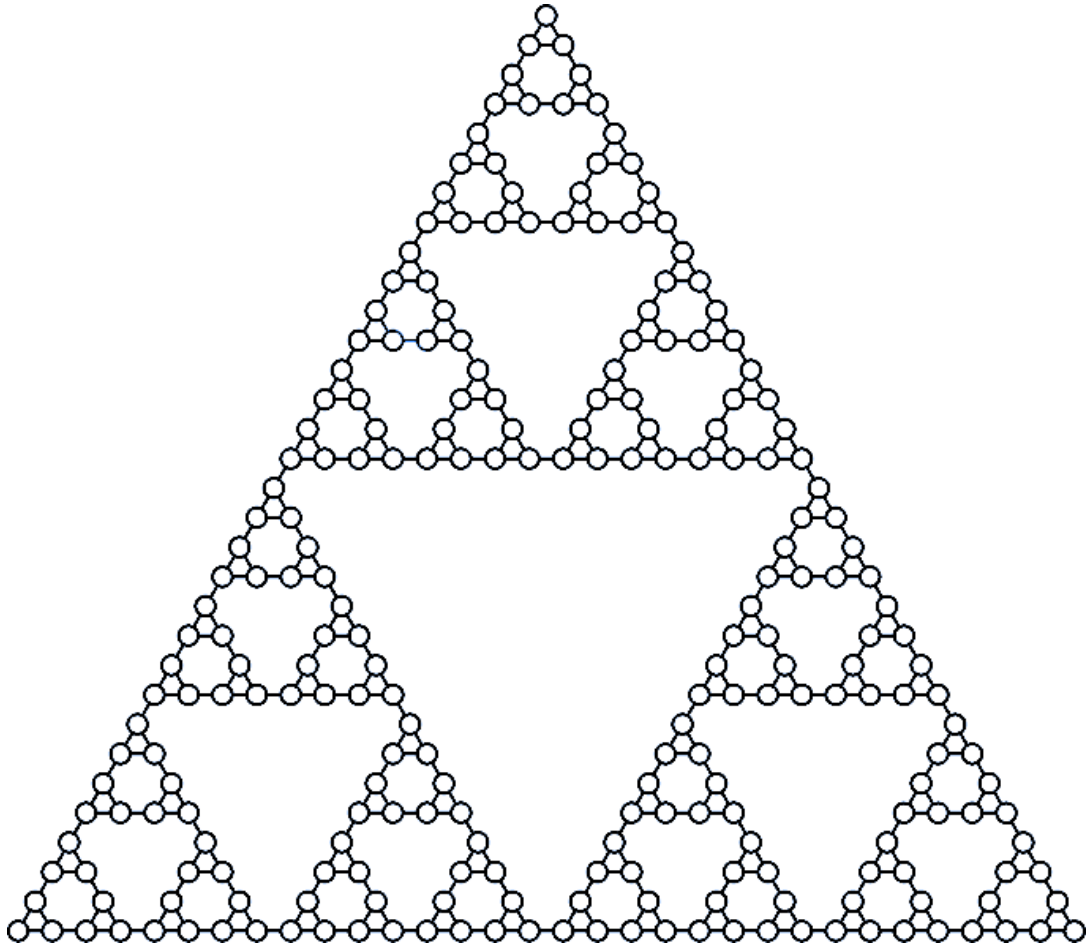
For each we take a copy of the state graph for one disc.
We add edges where the larger discs can be moved.

Graph formulation



For three discs similarly. . .

Graph formulation



. . . and for five discs.

Interpretation of the graph formulation

For n discs we have:

- 3^n different valid states $= 3^n$ vertices in the graph,
- all discs on one peg $=$ "tip" vertices,
- each state is reachable,
- to move all discs – at least $2^n - 1$ moves,
- fastest solution $=$ shortest path (next chapter).

Searching in a graph

For a general concept of "searching" in a graph we need to distinguish for each graph element few different states and one auxiliary structure:

Vertex can have the status ...

- initial – at the beginning,
- found – if it is found as an endpoint of an edge,
- processed – once all outgoing edges are processed.

Edge can have the status ...

- initial – at the beginning,
- processed – once it is processed from one end-point.

Depository as an auxiliary structure (sequence/set, see later),

- we store here all found and unprocessed vertices.

Based on how we pick the vertices in the depository, we obtain different variants of graph searching (depth-first/breadth-first search). For every vertex and edge we can implement an action to be performed – searching and processing a graph.

At the beginning

- pick an arbitrary vertex
- assign initial status to all vertices and edges

Algorithm of traversing all components

Traversing all connected components – we traverse each vertex and each edge.

```
// on the input is the graph G
input < graph G;
status(all vertices and edges of G) = initial;
depository U = arbitrary vertex u of G;
status(u) = found;
```

Now we traverse the graph...

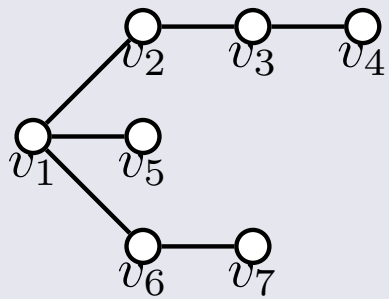
Algorithm of traversing all components (continued...)

```
// processing a component of G
while (U is not empty) {
   pick a vertex v from the depository U: U = U - {v};
   PROCESS(v);
   for (edges e incident with v)        // for all edges
      if (status(e) == initial)  PROCESS(e);
      w = other end-vertex of e = vw;  // known neighbor?
      if (status(w) == initial) {
         status(w) = found;
         add vertex w to depository U: U = U + {w};
      }
      status(e) = processed;
   }
   status(v) = processed;
   // check for additional components of G
   if (U is empty && G has additional vertices)
      U = {vertex u_1 from another component of G};
}
```

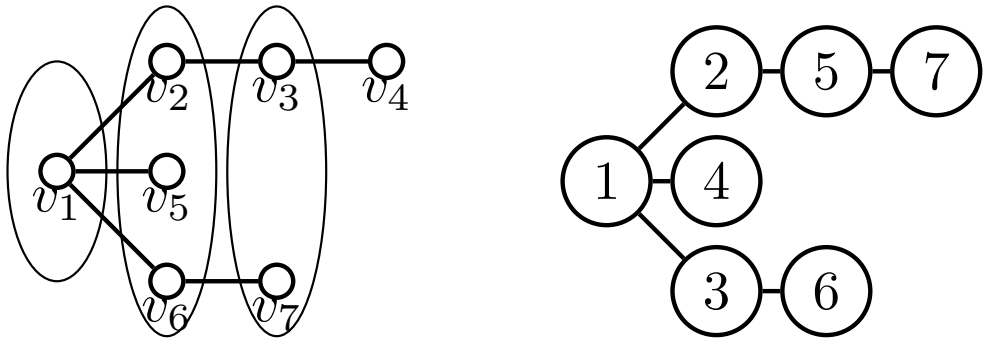
By various implementations of the *depository* we get various algorithms.

- *"Depth-first" search* – depository `U` is implemented as a stack, i.e. next processed vertex is the last found (and unprocessed).
- *"Breadth-first" search* – depository `U` is implemented as a queue, i.e. next processed vertex is the first found (and unprocessed).
- *Dijkstra algorithm* for shortest path – from the depository pick always the vertex closest to the initial vertex `v_0`; (work as breadth-first search when all edges are of "equal length").

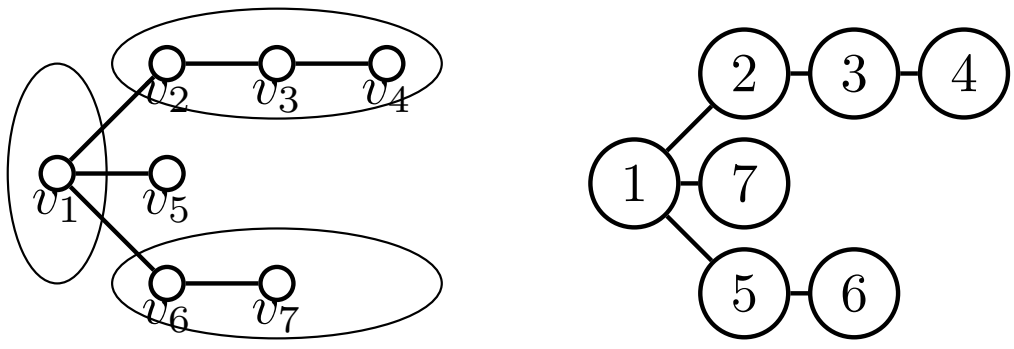
Example



Search through the graph using the depth-first and breadth-first search (starting at the vertex v_1).



Breadth-first search (starting at v_1).



Depth-first search (starting at v_1).

Note

The symbol $O(g(n))$ stands for all functions $f(n)$, for which there exist such positive constants c and n_0, that $\forall n > n_0$ is $0 \leq f(n) \leq c \cdot g(n)$.

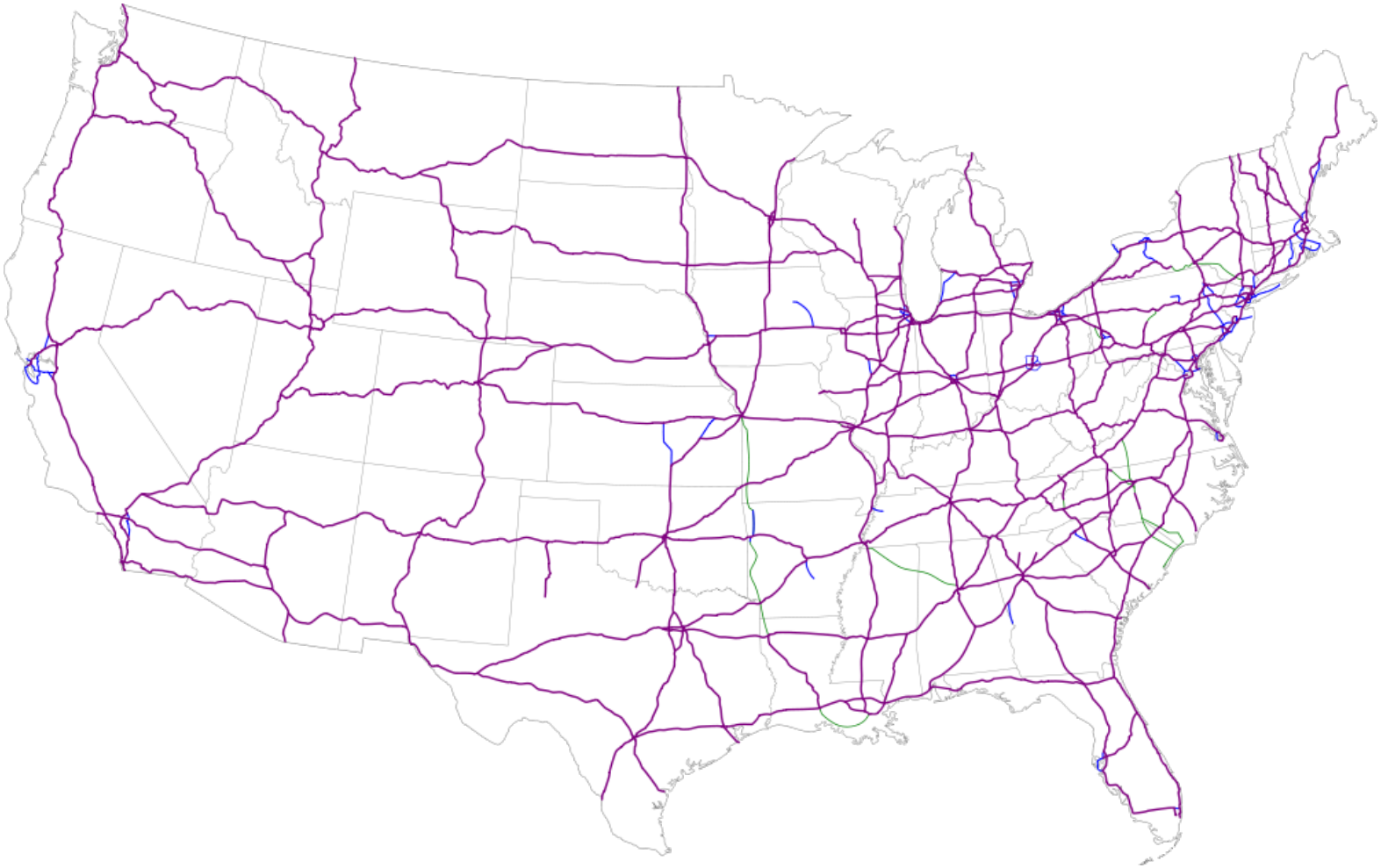
The algorithm described above is both easy and fast. The number of steps grows linearly with the number of vertices plus the number of edges of a given graph, the complexity is $O(n + m)$, where n is the number of vertices and m is the number of edges.

Questions

How to modify the algorithm to list all edges of a given graph?
How to modify the algorithm to check connectivity of a given graph?
How to modify the algorithm to find and distinguish all components of a given graph?

k-connectivity

Often we examine not only if there exists a connection between a two vertices in a graph, but also if there will be a loss of connectivity if the case of local failures (web, roads, electricity network).



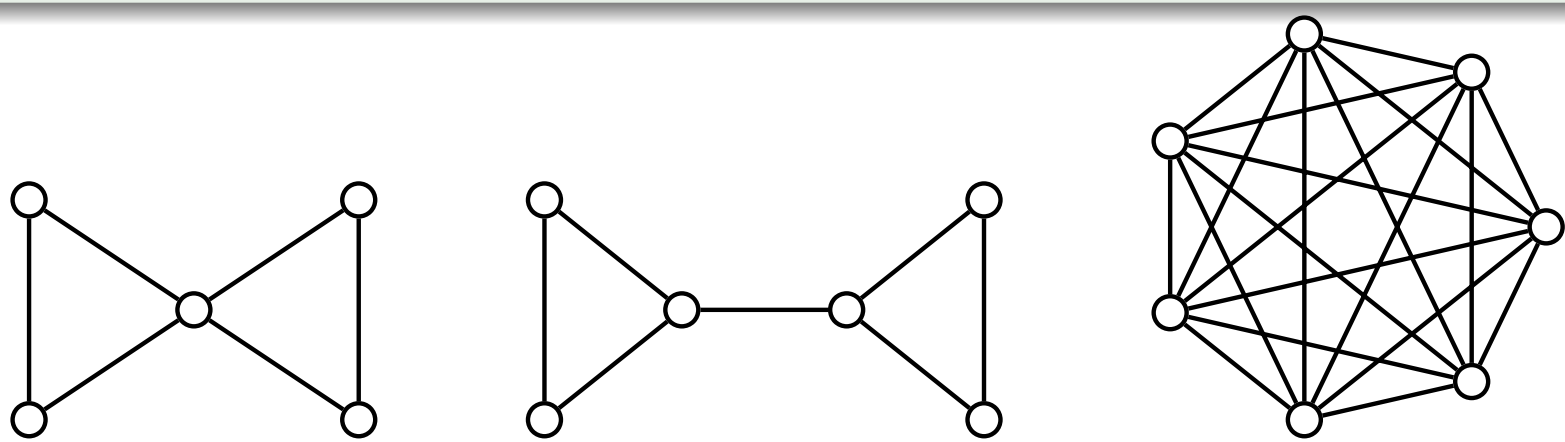
Eisenhower Interstate Highway System in the USA.

Definition

Graph G is edge k-connected if $k \geq 1$ and after removing any $k-1$ edges from G remains the resulting factor connected.
The edge-connectivity of G is such a highest number k that G is edge k-connected.

Definition

Graph G is vertex k-connected if $|V(G)| > k \geq 1$ and after removing any $k-1$ vertices from G remains the resulting induced subgraph connected.
The vertex-connectivity of G is such a highest number k that G is vertex k-connected.



Graphs with different edge/vertex k-connectivity.

We say that paths P and P' are:

- edge-disjoint if they share no edge,
- internally-disjoint if they share no internal vertex.

Theorem (Menger's theorem)

Graph G is edge k-connected if and only if there are at least k edge-disjoint paths between any two vertices (the paths can share vertices).
Graph G is vertex k-connected if and only if there are at least k internally-disjoint paths between any two vertices (the paths share only end-vertices).

Proof In the last Chapter. □

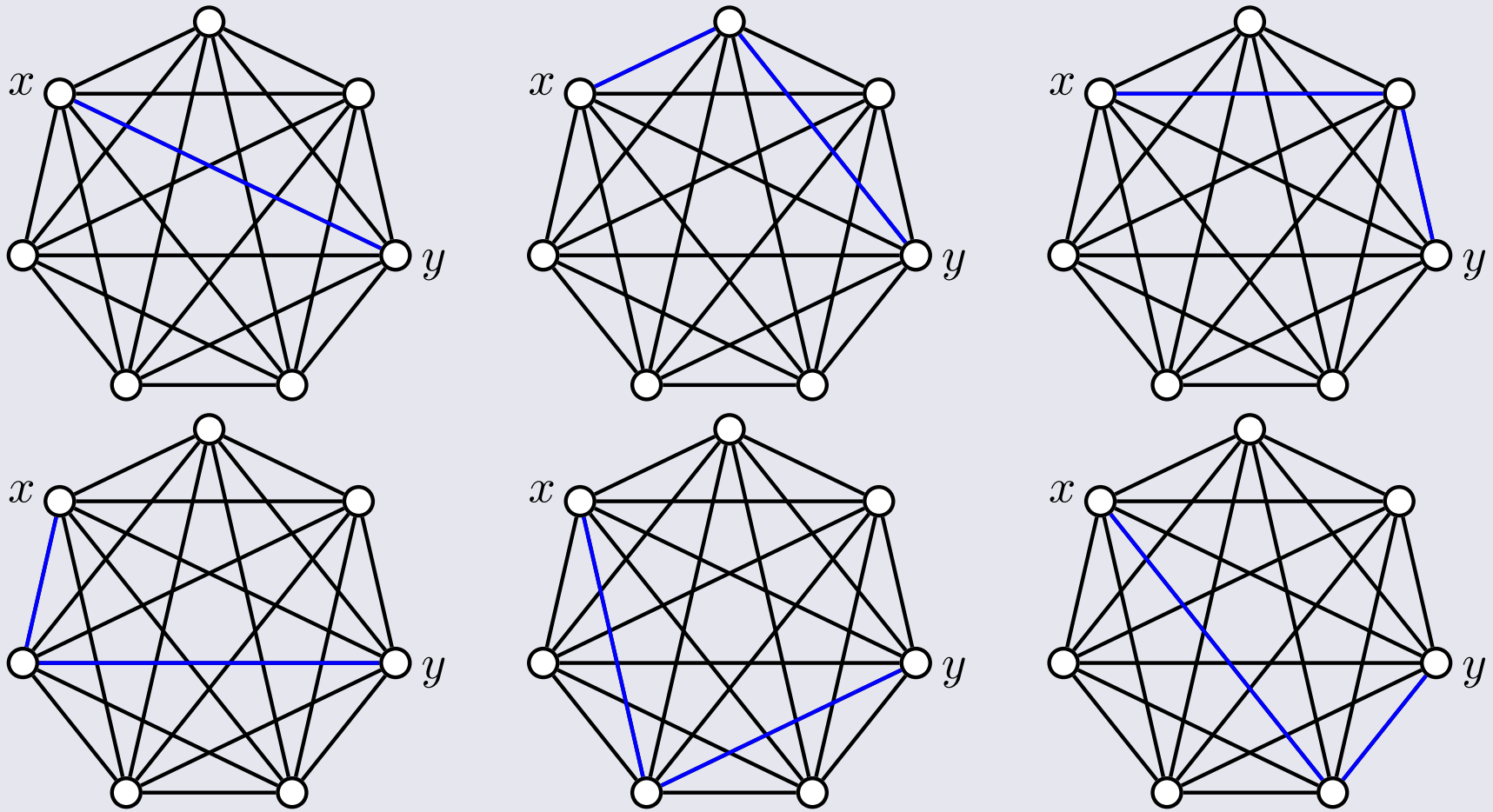
The following important theorem we state without a proof:

Theorem

In any graph G is the vertex-connectivity does not exceed edge-connectivity which then does not exceed the minimum vertex degree $\delta(G)$.

Example

The complete graph K_n is edge and vertex $(n-1)$-connected.



Different edge-/internally-disjoint paths between x and y in K_7.

Next lecture

Chapter Eulerian and hamiltonian graphs

- motivation
- eulerian graphs traversable in "one trail"
- hamiltonian graphs traversable in "one path"