1 Basics and Eulerian Graphs

1.1 The Seven Bridges of Königsberg

- In 1736 Leonard Euler was confronted in Königsberg with the "seven Bridges of Königsberg" problem:
- Does a Sunday stroll exist in Königsberg that crosses every bridge exactly once?
- Considering this problem, Euler invented graph theory

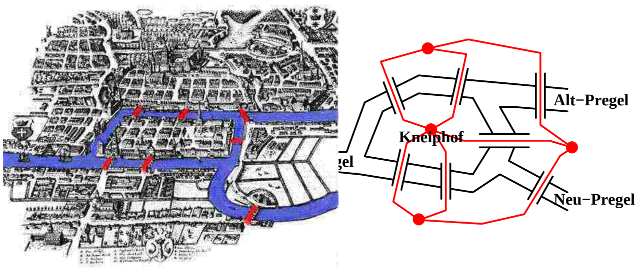


Fig. 1.1: Map of Königsberg in 1736 and an abstract version as a graph. (No. 516)

- Euler's idea of mathematical modeling:
  - Islands correspond to vertices
  - Bridges correspond to edges
  - Stroll corresponds to a sequence of edges
  - "Sunday Stroll" is a stroll that contains all edges exactly once and ends in the same vertex as it starts
- We start with formal definitions of a graph and the other components of the problem

Definition 1 (Graph)**.** An *undirected graph* $G = (V, E)$ consists of a set of vertices V and a multiset of edges $E \subseteq \{\{u, v\} \mid u, v \in V\}$. We denote the number of vertices and edges as follows: $|V| = n$, $|E| = m$.

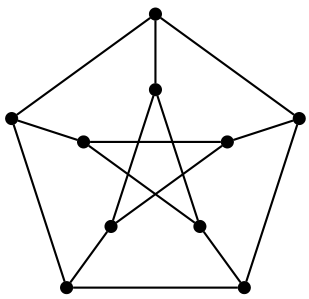


Fig. 1.2: Petersen graph (No. 525)

- **Note**: We almost always consider the graphical visualization of a graph
- More general: a graph models relations (edges) between entities (vertices)
- Let $e = uv$ be an edge, the vertices u and v are called *end vertices* of e
- Alternative notation: $e = \{u, v\}$
- An edge $e = uv$ is *incident* to u and v, we denote by $\delta(v) := \{wu \in E \mid w = v \text{ and } u \in V\}$ all incident edges of $v \in V$
- Vertices u and v are *adjacent* if an edge uv exists
- Special edges:
  - *Loops*: start and end vertex are the same, i.e., $e = vv$, $v \in V$
  - *Parallel edges*: various edges have the same end vertices

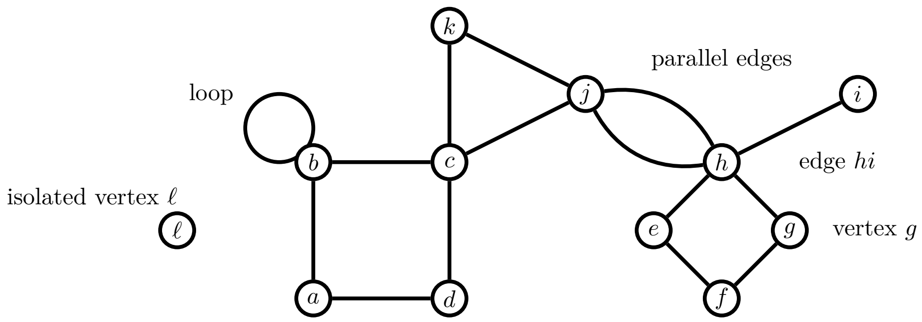


Fig. 1.3: Representation of a graph (No. 515)

Definition 2 (Simple Graph)**.** A *simple graph* has neither loops nor parallel edges.

- **Note**: If not stated otherwise, we consider simple graphs

Definition 3 (Degrees of vertices)**.** The *degree* $d(v) \in \mathbb{N}_0$ of a vertex $v \in V$ denotes the number of edges which are incident to v.

- Special vertices: Vertices v with $d(v) = 0$ are called *isolated* vertices
- **Note**: If not stated otherwise, we consider graphs without isolated vertices
- Let's start with the most basic property of a graph

Lemma 4 (Handshaking-Lemma)**.** *Let $G = (V, E)$ be a graph. Then*

$$\sum_{v \in V} d(v) = 2|E|.$$

Proof. Counting argument

- Every edge is counted twice, since every edge is incident to two vertices
- $\Rightarrow$
$$\sum_{v \in V} d(v) = 2|E|$$

$\square$

- A nice and often used property is the following:

Corollary 5. *Let $G = (V, E)$ be a graph. Then the number of vertices with odd degree is even.*

Proof. Handshaking-Lemma

- We consider all vertices with even and odd degrees separately:
$$2|E| \overset{(1)}{=} \sum_{v \in V} d(v) = \sum_{\substack{v \in V \\ d(v) \text{ even}}} d(v) + \sum_{\substack{v \in V \\ d(v) \text{ odd}}} d(v)$$
(1): Handshaking Lemma
- $\Rightarrow \sum_{\substack{v \in V \\ d(v) \text{ odd}}} d(v)$ is even, but all summands are odd
- $\Rightarrow$ the number of vertices with odd degree is even

$\square$

- Let's keep approaching the seven bridge problem

Definition 6 (Path)**.** Let $s, t \in V$ be two vertices. A *path* from s to t, also denoted as (s, t)-path, is an edge sequence $e_1 e_2 \dots e_k$ starting in s and ending in t, where for every edge e_i, $i = 1, \dots, k$, the end vertex of an edge corresponds to the start vertex of the successor edge, i.e., $e_i = v_i v_{i+1} \in E$, $i = 1, \dots, k$ with $v_1 = s$ and $v_{k+1} = t$.

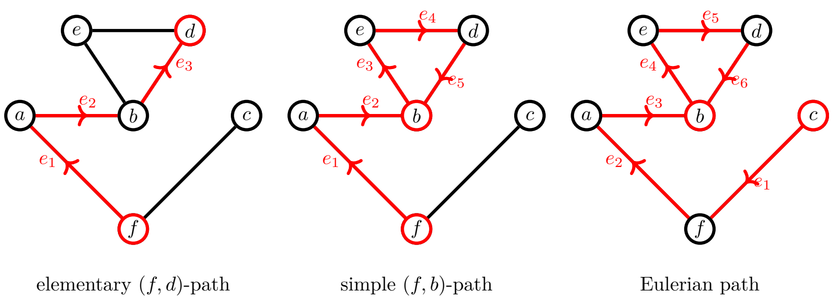


Fig. 1.4: Special paths (No. 517)

- Often, we denote a path p by a sequence of vertices, i.e., $p = v_1 v_2 \dots v_k v_{k+1}$
- The vertex v_i is called *predecessor* of v_{i+1} and v_i is the *successor* of the vertex v_{i-1}

- We denote with $V(p)$ and $E(p)$ the set of vertices and edges of a path p in a considered graph $G = (V, E)$
- Special paths:
  - *Elementary path* is a path without vertex repetition
  - *Simple path* is a path without edge repetition
  - *Eulerian path* is a path which visits every edge of the considered graph exactly once
- A path whose start vertex is the same as the end vertex is called a *cycle*
- A cycle that visits every edge exactly once is called *Eulerian cycle* or *Eulerian tour*

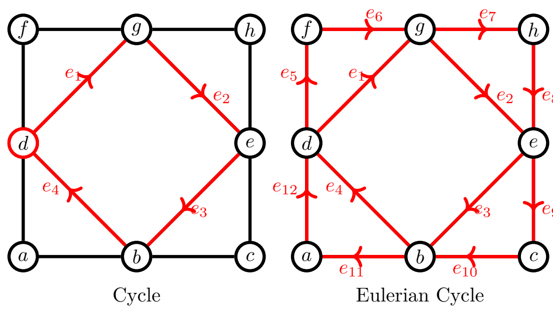


Fig. 1.5: Cycle vs. Eulerian cycle (No. 518)

Definition 7. Eulerian cycle problem

Given: Undirected graph $G = (V, E)$ without isolated nodes

Find: An Eulerian tour if one exists

- When does an Eulerian tour exist?

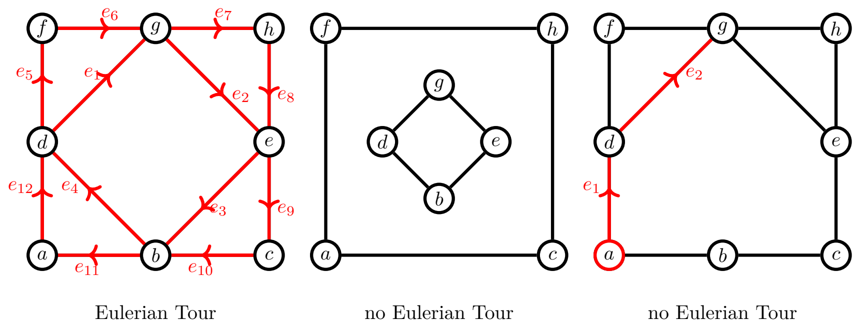


Fig. 1.6: Eulerian tour? (No. 519)

- First necessary condition: we need to get from any vertex to any other

Definition 8 (Subgraph and connected component)**.** Let $G = (V, E)$ be an undirected graph.

1. A *subgraph* of G is a graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$. We often write $G' \subseteq G$.
2. Let $V' \subseteq V$, then $G' := (V', E')$ with $E' := \{uv \in E \mid u, v \in V'\}$ is the subgraph *induces* by V'.
3. Graph $G = (V, E)$ is called *connected* if there is a (u, v)-path for all $u, v \in V$. A maximal connected, induced subgraph G' of G is called *connected component.* Maximal means that adding any vertex yields a disconnected subgraph.

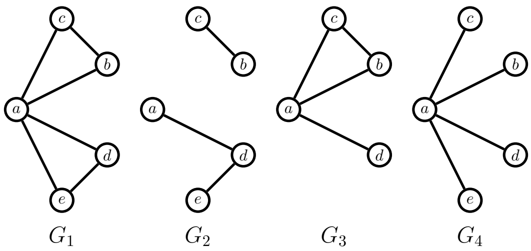


Fig. 1.7: Subgraph (No. 524)

- Connectivity is not sufficient
- Consider the degree of each vertex: if a tour exists the degree is even

Theorem 9 (Euler 1736, Hierholzer 1873)**.** *A graph G without isolated nodes has an Eulerian tour if and only if G is connected and the degree of each vertex is even.*

- In order to prove Theorem 9, we need the following property:

Lemma 10 (Closed-walk)**.** *Let $G = (V, E)$ be a graph such that the degree of every vertex $v \in V$ is even. Let p be a simple path which starts in v_0 and ends in v_k. If every incident edge of v_k is also a part of p, then $v_0 = v_k$ is true. Therefore the path p is a cycle.*

Proof. Counting the edges

- Let $p = v_0 v_1 \dots v_k$ be a simple path such that all incident edges of v_k are a part of p
- $\Rightarrow$ p cannot be extended
- Assume $v_0 \neq v_k$
- Since v_k is the end vertex, an odd number of incident edges of v_k are a part of p
- By assumption there exists an edge $e' \in \delta(v_k)$ which is not a part of p
- $\Rightarrow$ We can extend path p by e' without using an edge twice $\Rightarrow$ contradiction

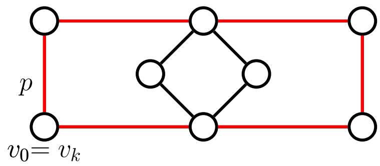


Fig. 1.8: Path p isn't extendable in vertex v_k, however this doesn't mean that p is an Eulerian tour (No. 531)

□

- We now proceed with proof of Euler's Theorem

Proof. Use the Closed-walk Lemma 10

- "⇒": Let T be an Eulerian tour ⇒ the graph is connected
- Let v be a vertex which is crossed k times by T
- Every time the tour crosses v two incident edges are visited
- ⇒ $2k$ edges are incident to v, i.e., $|\delta(v)| = 2k$
- ⇒ All vertices have an even degree

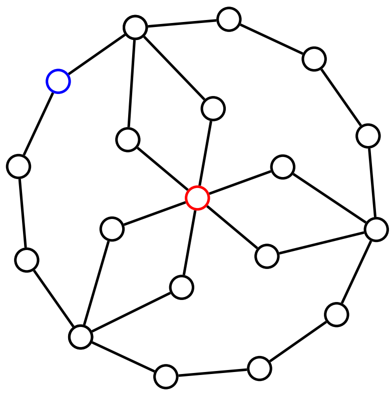


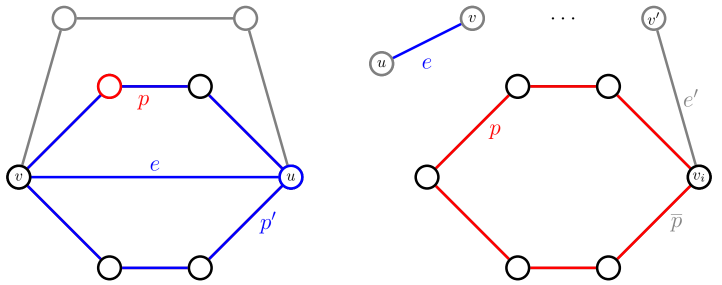
Fig. 1.9: Every vertex has an even degree (No. 653)

- "⇐": Let G be connected and $d(v)$ even $\forall v \in V$
- Let $p = v_0 v_1 \dots v_k$ be a simple path in G with maximum length
- ⇒ p cannot be extended, i.e, all incident edges of v_k are part of p
- ⇒ p is a cycle, i.e., $v_0 = v_k$ (Closed-walk Lemma)
- Case 1: p uses every edge of G ⇒ p is an Eulerian tour
- Case 2: assume there exists an edge $e = uv \notin E(p)$
  - Case 2.1: e is incident to a vertex of p, i.e. $v = v_i \in V(p)$ for an $i = 0, \dots, k$
    - We can define a path

      $$p' = uv_i \dots v_k v_1 \dots e_{i-1} v_i$$

      with larger length ⇒ contradiction
  - Case 2.2: e is not incident to a vertex of p
    - As G is connected, there exists a path $\overline{p}$ from v to a vertex $v_i \in V(p)$ with $E(\overline{p}) \cap E(p) = \emptyset$

- Consider the edge $e' \in E(\overline{p})$ which is incident to v_i, i.e. $e' = v'v_i$
- $\Rightarrow$ setting as in Case 2.1 $\Rightarrow$ contradiction



Case 2.1 Case 2.2

Fig. 1.10: Proof Eulerian tour (No. 526)

$\square$

- **Note:** A graph which contains an Eulerian tour is called *Eulerian graph*
- Euler only provided a proof for the necessary conditions
- With this, the seven bridges of Königsberg problem was solved
- Hierholzer developed the first algorithm to construct an Eulerian tour in 1871

1.2 Hierholzer's Algorithm

Algorithms

- *Algorithms* are finite descriptions of steps to hopefully get a solution to a considered problem
- Examples:
  - Lego construction manual
  - Recipes
  - IKEA instructions

Reading Material

- Important components of our algorithms:

Input	Given instances to perform an algorithm
	Example: Graph $G = (V, E)$, edge weight $c(e) \in \mathbb{R}$, $\forall e \in E$
Output	Solution which is generated by the algorithm
	Example: (s, t)-path
$\leftarrow / =$	Assignment
	Example: $x \leftarrow 3$ means that variable x is set to value 3
//	Comments for the readers
While-Loop	**While** a statement is true **do** the following
	Example: **While** $\exists$ a vertex $v \in V$ with $d(v) \geq 3$ and v is not red **do**
	• Color v red

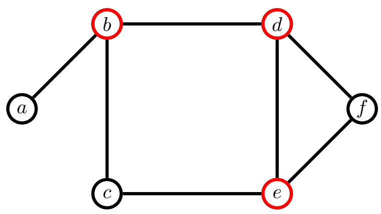


Fig. 1.11: The algorithm colors vertices with $d(v) \geq 3$ red. (No. 521)

If-Query **If** a statement is true **then** do the following **Else** do the following

Example: Choose $v \in V$

If $d(v) \geq 3$ **then**

- Color v red

Else

- Color v blue

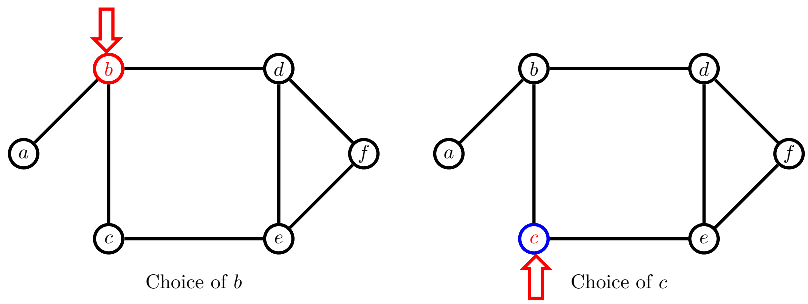


Fig. 1.12: Depending on the choice of the vertex it will be colored red or blue. (No. 522)

For-Loop **For** a set **do** the following

Example: Choose $v \in V$

For $(u, v) \in E$ **do**

- Color (u, v) blue

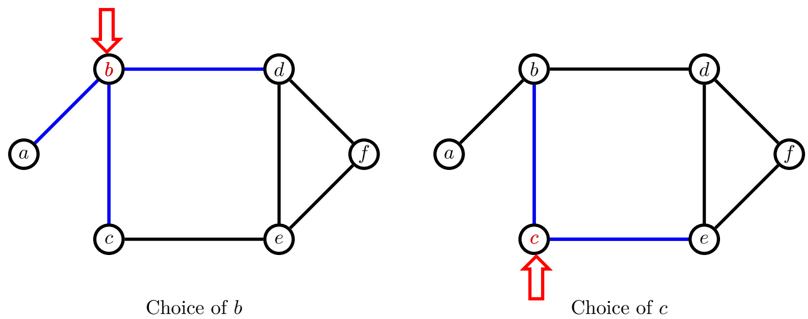


Fig. 1.13: The edges which are incident to a selected vertex will be colored blue. (No. 523)

Eulerian Cycle Problem

- Carl Hierholzer was a German mathematician who habilitated in Königsberg
- After his death in 1871 two colleges published in 1873 his algorithm that solves the

Eulerian cycle problem

- Idea of Hierholzer's algorithm
  - Choose a vertex v_0
  - Construct a cycle by choosing unused edges and mark the edges as visited
  - If the cycle doesn't contain all edges, choose a vertex on the cycle which is incident to an unused edge and construct a new cycle
  - Merge both cycles

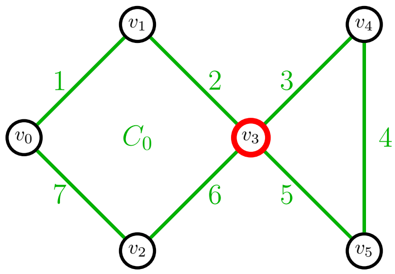


Fig. 1.14: Hierholzer's algorithm (No. 527)

Algo. 1.1 Hierholzer's algorithm

Input: Connected graph $G = (V, E)$ with $d(v)$ even $\forall v \in V$

Output: Eulerian tour K

Method:

Step 1
- Choose a vertex v_0
- Successively select unused edges until we obtain a cycle K

Step 2 **If** K is an Eulerian tour **then**
Stop and **Return** K

Step 3
- Set $K' = K$
- Choose a vertex $v_i \in V(K')$ which is incident to an unused edge
- As in Step 1, construct a cycle K'' starting from v_i with $E(K'') \cap E(K') = \emptyset$
- Merge K' and K'' to a new cycle K as follows: pass all vertices from v_0 to v_i, pass through K'' and then pass the rest of K'
- Go to Step 2

Theorem 11. *Let G be an undirected, connected graph such that all vertices have an even degree. Then, Hierholzer's algorithm constructs an Eulerian tour.*

Proof. Feasibility of each step

- Step 1: Construction of a path which doesn't use an edge twice and which either is a cycle or isn't extendable $\Rightarrow$ we get a cycle (Closed-walk Lemma)
- Step 2: Algorithm terminates if an Eulerian tour is found
- Step 3: is valid due to the following two claims

- *Claim 1*: If K is not an Eulerian Tour, then $\exists\, v_i \in V(K')$ which is incident to $e \in E \backslash E(K)$.
  Proof of Claim:
  - K' isn't an Eulerian tour, i.e. $\exists\, e = uv \in E(G) \backslash E(K')$
  - Case 1 : $u \in V(K')$ or $v \in V(K') \Rightarrow$ claim holds true
  - Case 2: $u \notin V(K')$ and $v \notin V(K')$
    - G is a connected graph
    - Find a path p from v to a vertex of K'
    - The end vertex of p corresponds to a vertex is incident to an edge $e \in E \backslash E(K)$ □C1

- *Claim 2*: Combining K' and K'' leads to a new cycle.
  Proof of Claim
  - The graph $G' = (V, E')$ with $E' = E \backslash E(K')$ has an even degree for all vertices
  - Thus, G' satisfies the property of the Closed-walk Lemma
  - By construction, K'' is a cycle (Closed-walk Lemma)
  - Let $K' = v_0 v_1 \dots v_i \dots v_0$ and $K'' = w_0 w_1 \dots w_j w_0$ be two cycles with $w_0 = v_i$
  - $\Rightarrow$
  $$K := v_0 \dots v_i w_1 \dots w_j v_i \dots v_0$$
  is a cycle □C2

- Since Step 3 is executed at most $\frac{1}{2}|E|$ times, the algorithm terminates with Step 2

□

- Fleury introduced in 1883 another important algorithm to solve the Eulerian cycle problem
- Here, the Eulerian tour is constructed in one run by choosing the next edge wisely

1.3 Special Graph Classes

- Often graphs have special structures
- According to these structures
  - algorithms may behave differently
  - problems become more easy to solve
  - or optimal solutions may have certain properties

Trees

- Trees are the basic structure of a connected graph
- They combine two opposite concepts:
  - Connection $\Rightarrow$ (many) edges are necessary
  - No cycles $\Rightarrow$ not too many edges are allowed

Definition 12 (Tree)**.** A connected graph $G = (V, E)$ is called a *tree* if contains no cycle. The *leaves* of a tree are the vertices with degree one.

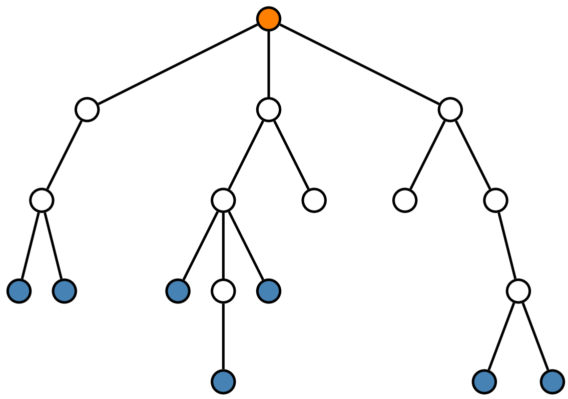


Fig. 1.15: Tree with one root (orange) and seven leaves (blue) (No. 534)

- Often, we select one arbitrary vertex and call it the *root* of the tree
- By means of a root r, we can address other vertices or leaves v with their *distances* to r, i.e., the number of edges which are between root r and vertex v in tree T. We write $\text{dist}_T(r, v)$

Lemma 13. *Let G be a tree with at least two vertices, then G has at least one leaf.*

Proof. Exercise □

- An important property, we often use for trees, is the following

Lemma 14. *Let G be a graph. Then G is a tree if and only if there exists a unique path between any two vertices in G.*

Proof. Exercise □

- A slight extension of a tree is a forest
- A *forest* is a graph whose connected components are trees.

Lemma 15. *Let G be a forest on n vertices with m edges and p connected components. Then $n = m + p$.*

- With $G + e$ and $G - e$ we denote the graph $G = (V, E)$ with an additional edge $e \notin E$ and without the edge $e \in E$, respectively, i.e. $G + e = (V, E \cup \{e\})$ and $G - e = (V, E \backslash \{e\})$

Proof. Induction on the number of edges m

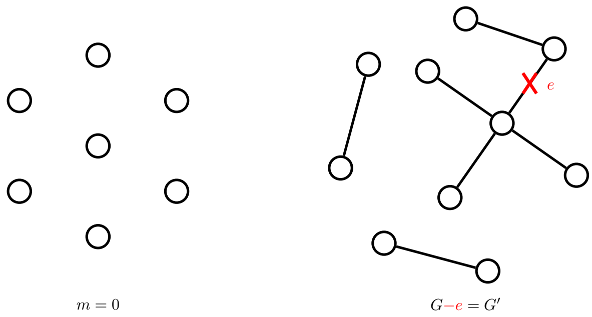


Fig. 1.16: In forests: $n = m + p$ (No. 654)

- I.B.: $m = 0 \Rightarrow$ Each vertex is a separate connected component, i.e. $n = p$
- I.H.: For every forest on n vertices with m edges, it holds $n = m + p$
- I.S.: Let G be a forest on n vertices with $m + 1$ edges and p connected components
- Delete an arbitrary edge e to obtain $G' = G - e$
- $\Rightarrow$ G' is also a forest on n vertices with m edges and $p + 1$ connected components
- $\Rightarrow$
$$n \overset{I.H.}{=} m + p + 1 = (m + 1) + p$$
- By the principle of induction, $n = m + p$ is true

$\square$

- For trees, we obtain with this a nice characterization via the number of edges

Theorem 16 (Important characteristics of trees)**.** *Let G be a graph on n vertices. Then the following statements are equivalent:*

1. *G is a tree (i.e. is connected and has no cycles).*
2. *G has $n - 1$ edges and no cycles.*
3. *G has $n - 1$ edges and is connected.*

Proof. Number of connected components vs. number of edges

- (1) $\Rightarrow$ (2): Graph G is a tree, i.e. G has no cycles and is connected by definition.
- Prove: $m = n - 1$
  - By Lemma 15, $n = m + p \Leftrightarrow m = n - p$
  - Since $p = 1 \Rightarrow m = n - 1$
- (2) $\Rightarrow$ (3): Graph G has no cycles and $n - 1$ edges.
- Prove: $p = 1$
  - Since G has no cycles, G is a forest
  - Lemma 15: $m = n - p \Rightarrow p = n - m = n - (n - 1) = 1$
  - $\Rightarrow$ G is connected
- (3) $\Rightarrow$ (1): Graph G is connected with $n - 1$ edges.

- Prove: G has no cycles
  - Assume that G has a cycle $\Rightarrow \exists$ edge such that $G - e$ is connected as well
  - Delete edges until the new graph G' has no cycles and is connected

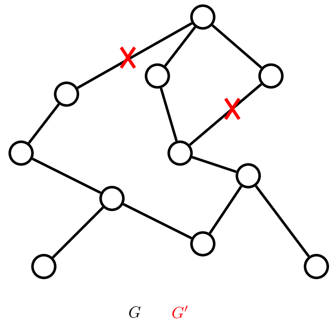


Fig. 1.17: Delete until acyclic (No. 655)

  - $\Rightarrow m' < m$ and $m' = n - 1$ since G' is a tree by construction
  - However, $m = n - 1$, contradiction

□

Spanning trees

- Networks and graphs in real-world applications are quite big, e.g., the representation of the internet as graph or of social networks
- Often, one is looking for a simple view of the graph and a way to understand how to get from one vertex to another, if possible
- Spanning trees represent such a simple view of the graph
- The question is, how to we obtain such a tree
- In other words: how to design an algorithm, that visits all nodes and all edges in an efficient way and remembers a tree structure of the graph

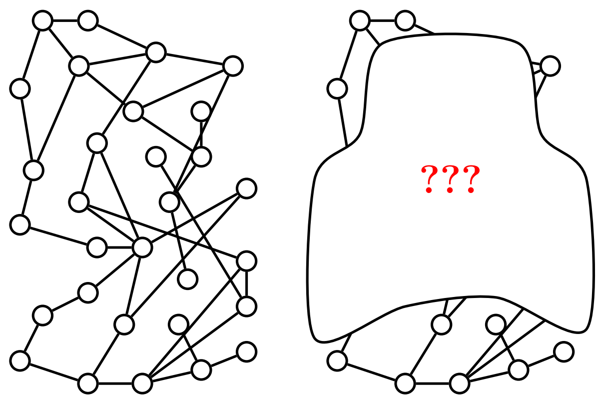


Fig. 1.18: Searching a big graph (No. 667)

- In computer science, this is also known as tree traversal, tree search or walking the tree
- Such trees are classified by the order in which the nodes are visited
- The underlying general principle is the following

Algorithm 1.1 Generic graph search

Input: Graph $G = (V, E)$, root vertex $r \in V$

Output: Search Tree T

Method:

Step 1 Initialization

- Set $R = \{r\}$ as the set of visited nodes
- Set $\text{pred}[r] = 0$ and $\text{pred}[v] = NULL\ \forall v \in V \backslash \{r\}$
- Set $L = \{r\}$ as the list of candidates for a visit

Step 2 Search procedure

While $L \neq \emptyset$ **do**
- chose $v \in L$

If $\exists\ w \in V \backslash R$ with $vw \in E$ **do**
- chose $w \in V \backslash R$ with $vw \in E$
- add w to R, set $\text{pred}[w] = v$, addd w to L

Else delete v from L

Step 3 **Return** Tree $T = (R, E)$ with $E = \{\{u, \text{pred}(u)\} \mid u \in R \backslash \{r\}\}$.

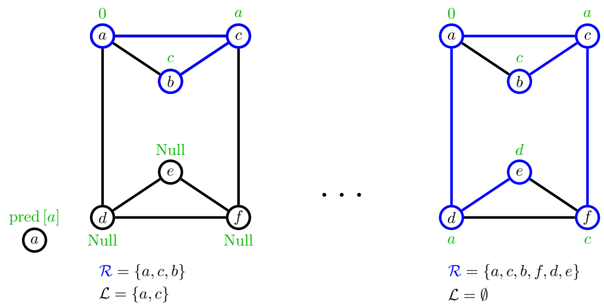


Fig. 1.19: General graph search (No. 672)

- The output is a tree due to the construction
- If all nodes of a graph are part of a tree, we call it a spanning tree

Definition 17. Let G be a connected graph. A subgraph $T \subseteq G$ is called a *spanning tree* of G if T is a tree which covers all vertices of G, i.e. $V(T) = V(G)$.

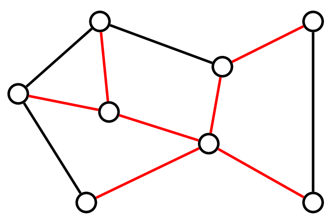


Fig. 1.20: Spanning tree (No. 668)

- If a graph is connected, the generic search algorithm computes a spanning tree

- The output of the generic graph search algorithm depends on the order in which the vertices are chosen from L
- The most famous ones are

  BFS Breadth First Search: the vertices are always added at the end of L and the first vertex is chosen to be visited next, i.e., First-In-First-Out (FIFO)

  DFS Depth First Search: the vertices are added at the beginning of L and the first vertex is chosen to be visited next, i.e., Last-In-First-Out (LIFO)

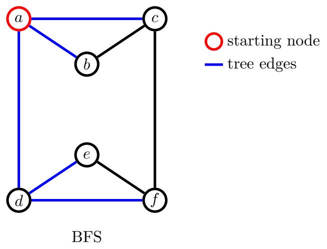


Fig. 1.21: BFS and DFS (No. 673)

- Using BFS or DFS on the same graph, computes spanning trees with different properties

Lemma 18. *Let G be an undirected graph and r the root vertex.*

1. *Let T_{BFS} be a BFS-tree. Then any (r, v)-path is a shortest path with respect to the number of edges.*
2. *Let T_{DFS} be a DFS-tree. Then any edge that is not in T_{DFS} connects nodes along a path starting in r.*

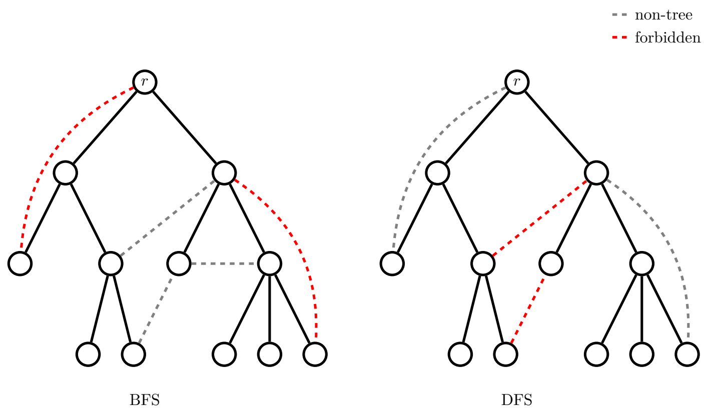


Fig. 1.22: Properties of BFS and DFS (No. 669)

Proof. Construction of the tree

- Property (1): In T_{BFS} any path starting in r is a shortest path.
  - Let $\text{dist}(v)$ be the shortest path length from r to v in G w.r.t. the number of edges

- Let $\text{level}(v)$ be the path length from r to v in T_{BFS}
- *Claim*: $\text{level}(v) = \text{dist}(v)\ \forall v \in V$.
  Proof of Claim:
  - Since T_{BFS} is a subgraph of G, $\text{level}(v) \geq \text{dist}(v)$ for all $v \in V$
  - Assume, $\exists w \in V$ with $\text{dist}(w) < \text{level}(w)$ and minimum $\text{dist}(w)$ value, i.e., the "first" vertex that violates the condition according to $\text{dist}(v)$

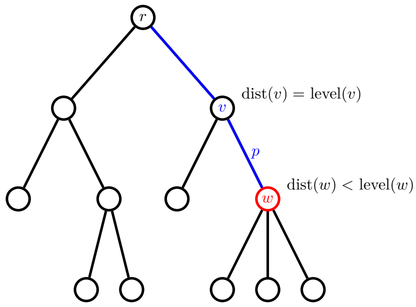


Fig. 1.23: w is smallest criminal (No. 670)

  - Let p be a shortest path in G from s to w
  - Let vw be the last edge on p
  - $\Rightarrow \text{dist}(v) = \text{level}(v)$
  - Then,

$$\begin{aligned} \text{level}(w) > \text{dist}(w) &= \text{dist}(v) + 1 \\ &= \text{level}(v) + 1 \end{aligned}$$

  - v is added to L before w
  - (BFS): w is added to L when scanning v or from another vertex $v' \in L$ (which was added even before)
  - $\Rightarrow \text{level}(w) \leq \text{level}(v) + 1$, contradiction □C

- Property (2): Any non-tree edge $e \in E \backslash E(T_{\text{DFS}})$ connects only nodes along a path starting in s
  - Assume uv connects two different paths, i.e., $v \notin p_{[r,u]}$ and $u \notin p_{[r,v]}$ with $p_{[a,b]}$ being the path in T connecting a and b
  - Let w be last vertex with $w \in p_{[r,v]} \cap p_{[r,u]}$
  - W.l.o.g. u is added to L before v
  - (DFS): u is scanned before w
  - $\Rightarrow v$ is added to L when u is scanned and $\text{pred}(v) = u$
  - $\Rightarrow uv \in T$, contradiction

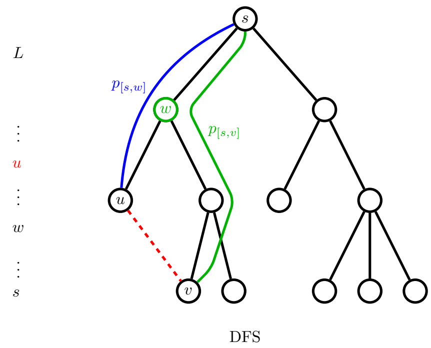


Fig. 1.24: uv does not exist (No. 671)

□

- BFS and DFS are often subroutines in different, more complex algorithms

BFS	DFS
• shortest path computation • computation of maximum flows • choice of shortest cycles	• test of planarity • topological sorting • construction of strong connectivity components

Bipartite Graphs

- In real world applications, vertices often represent groups and edges consist just between different groups
  - worker vs. jobs
  - seats vs. persons/people
- Such situations yield so called bipartite graphs

Definition 19 (Bipartite Graph)**.** A graph $G = (V, E)$ is *bipartite* if there exists a partition of the vertices $V = U \cup W$ such that every edge $e = uw \in E$ has one end vertex in U and one in W.

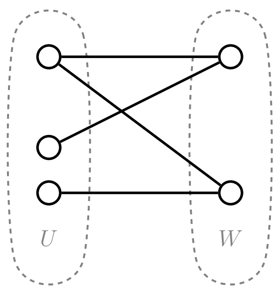


Fig. 1.25: Bipartite graph (No. 528)

Theorem 20. *A graph G is bipartite if and only if G has no cycle of odd length.*

- The length of a cycle equals the number of edges

Proof. Exercise □

- Using the idea in the proof, we can easily derive an algorithm to test whether a graph is bipartite

Complete Graphs

- Complete graphs contain the maximum number of edges

Definition 21. A graph $G = (V, E)$ in which each two vertices are adjacent is called a *complete graph.*

- We denote the complete graph on n vertices with K_n
- K_n has exactly $\binom{n}{2}$ edges

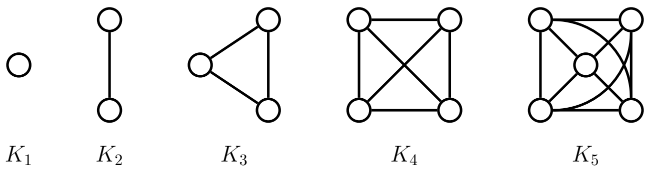


Fig. 1.26: Complete graphs $K_1, \ldots, K_5$ (No. 530)

- For some problems, we assume that a complete graph is given
- The most famous of these problems is the Traveling Salesperson Problem (TSP)

Definition 22. Traveling Salesperson Problem (TSP)

Given: Undirected complete graph $G = (V, E)$ and edge costs $c : E \to \mathbb{Z}$

Find: A cycle $\pi = e_1 e_2 \ldots e_n$ which visits every vertex of graph G exactly once with minimum cost

$$c(\pi) := \sum_{e \in \pi} c(e)$$

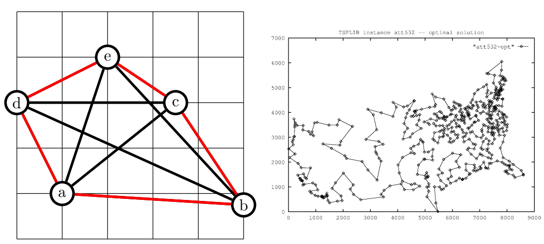


Fig. 1.27: TSP (No. 529)

- Size of search space for possible solutions:
  - Each of n cities can be at each position $\Rightarrow n!$ different ways

  - Cost for tours are the same regardless from which city you start and in which direction you are going
  - $\Rightarrow \frac{n!}{2n} = \frac{(n-1)!}{2}$ possible solutions
- Enumeration of all solutions to chose the best tour is no option

n	$(n-1)!/2$	Comparison
2	1	
6	60	1 min. if per tour 1 sec.
11	1.814.400	$\approx 5 \times$ distance earth to moon (in km)
16	$6,5 \cdot 10^{11}$	Age of Universe $\approx 1,3 \cdot 10^{10}$ (in years)