INF421, Lecture 4
Trees and DFS

Leo Liberti

LIX, École Polytechnique, France

Course

- **Objective**: teach notions AND develop intelligence
- **Evaluation**: TP noté en salle info, Contrôle à la fin. Note: $\max(CC, \frac{3}{4}CC + \frac{1}{4}TP)$
- **Organization**: fri 31/8, 7/9, 14/9, 21/9, 28/9, 5/10, 12/10, 19/10, 26/10, amphi 1030-12 (Arago), TD 1330-1530, 1545-1745 (SI:30-34)
- **Books**:
  1. K. Mehlhorn & P. Sanders, *Algorithms and Data Structures*, Springer, 2008
  2. D. Knuth, *The Art of Computer Programming*, Addison-Wesley, 1997
  3. G. Dowek, *Les principes des langages de programmation*, Editions de l'X, 2008
  4. Ph. Baptiste & L. Maranget, *Programmation et Algorithmique*, Ecole Polytechnique (Polycopié), 2006
- **Website**: `www.enseignement.polytechnique.fr/informatique/INF421`
- **Blog**: `inf421.wordpress.com`
- **Contact**: `liberti@lix.polytechnique.fr` (e-mail subject: INF421)

Lecture summary

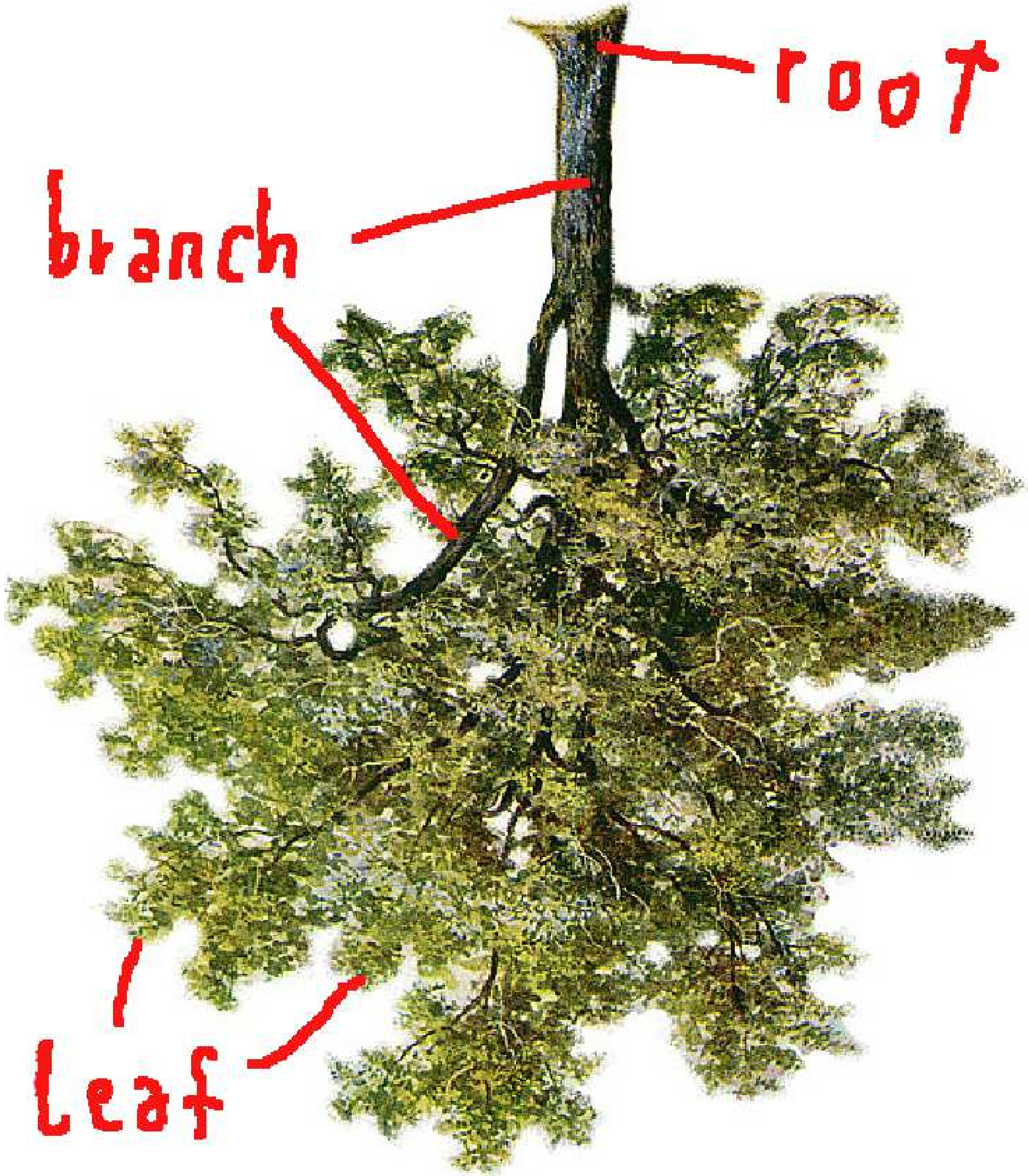
- Introduction and reminders
- Spanning trees
- Chemical trees
- Grammars and languages
- Depth-First Search (DFS)

Introduction and reminders

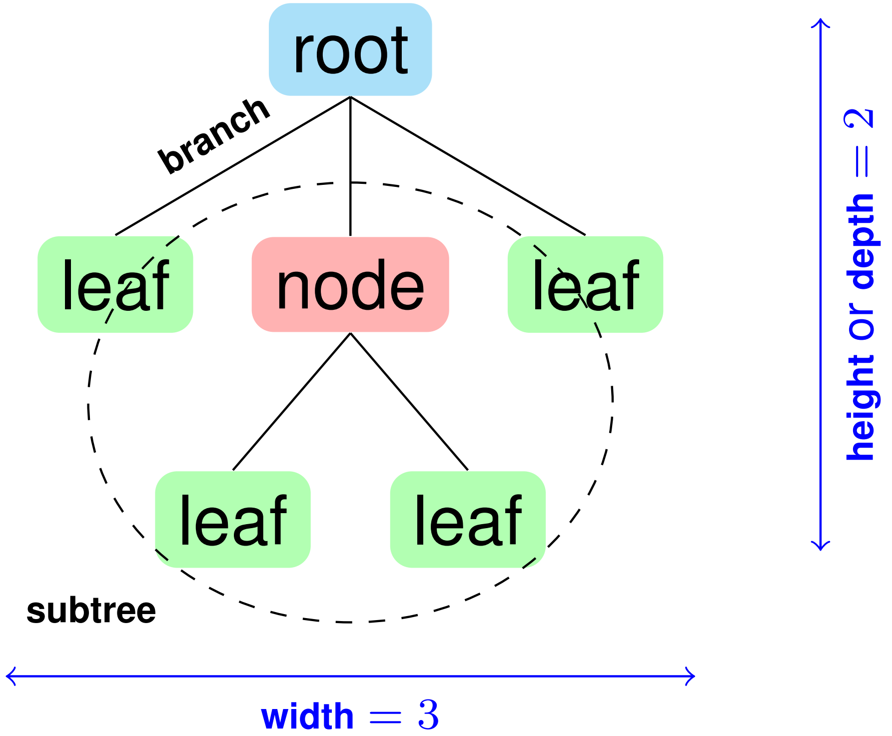
Trees

How we draw them

Nomenclature



Graphical representation



$\boxed{\text{height/depth} = \text{length (\#branches) of longest walk [root} \rightarrow \text{leaf]}}$

$\boxed{\text{width} = \text{max (\#nodes) with same depth}}$

Mathematical definition of a tree

Tree: a connected graph $G = (V, E)$ without cycles

- **root** node $\Rightarrow$ **rooted** tree
- If v has $|N(v)| = 1$, v is a **dangling node**

$$v_1 - v_2 \qquad v_3 - v_4$$

v_1, v_3: dangling nodes

- **Leaf**: non-root dangling node
- **Branch**: edge of a rooted tree

Orientations

- **Orientation** of a tree $T = (V, E)$ with root $r \in V$: digraph $U = (V, A)$ s.t.:

$$\forall \{u, v\} \in E \qquad (u, v) \in A \text{ XOR } (v, u) \in A$$

- **Outward orient.**: an orientation s.t.:

$$\forall \ell \in V \qquad \text{leaf}(\ell) \to \exists \text{ path in } U : r \to \ell$$

$$\begin{tikzcd} v_1 \arrow[r, dash] & {\color{red}r} \arrow[r, dash, bend left] & v_3 \arrow[r, dash] & v_4 \end{tikzcd} \quad \rightarrow \quad \begin{tikzcd} v_1 & {\color{red}r} \arrow[l] \arrow[r, bend right] & v_3 & v_4 \arrow[l] \end{tikzcd}$$

- **Inward orient.**: an orientation s.t.:

$$\forall \ell \in V \qquad \text{leaf}(\ell) \to \exists \text{ path in } U : \ell \to r$$

$$\begin{tikzcd} v_1 \arrow[r, dash] & {\color{red}r} \arrow[r, dash, bend left] & v_3 \arrow[r, dash] & v_4 \end{tikzcd} \quad \rightarrow \quad \begin{tikzcd} v_1 \arrow[r] & {\color{red}r} & v_3 \arrow[l, bend left] \arrow[r] & v_4 \end{tikzcd}$$

A tree has $|V| - 1$ edges

Thm.

A tree T on a set V has $|V| - 1$ edges

Proof

Let $m(T)$ be the number of edges in T

Show $m(T) = |V| - 1$ by induction on $|V|$

If $|V| = 2$, a spanning tree has one edge

Induction hypothesis: Suppose $m(T) = |V| - 2$ for all trees T on $|V| - 1$ nodes

Let T be any tree on V

Any tree must have at least one leaf node ℓ (why?)

Because ℓ is a leaf, it is incident to only one edge e

Consider the tree $T' = T \smallsetminus \{e\}$ on $V' = V \smallsetminus \{\ell\}$

Because $|V'| = |V| - 1$, $m(T') = |V| - 2$ by the induction hypothesis

Thus, T has exactly $m(T) = m(T \cup \{e\}) = m(T) + 1 = |V| - 1$ edges

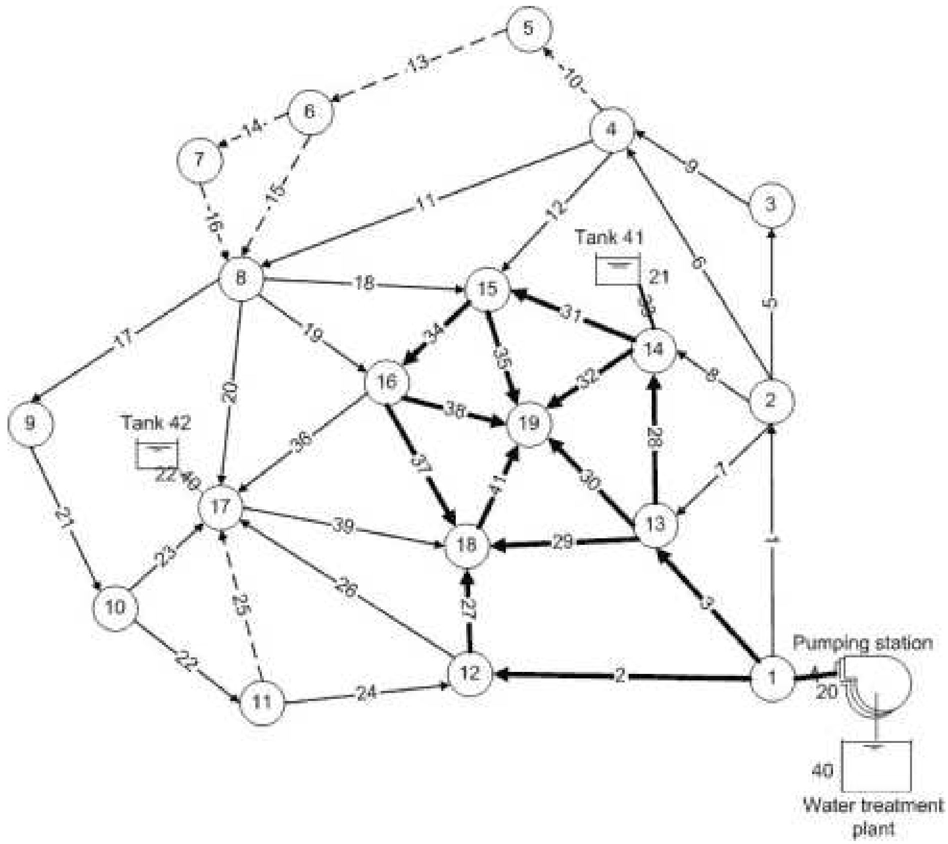
Spanning trees

Distribution networks

- A **network** is another term for a digraph (V, A), when used to model a distribution process
  - E.g. V: production sites, customer sites
  - Arc between two sites: transfer of material
  - Arc between two production sites: transfer of raw material
  - Arc between production and customer: transfer of finished material
- Main cost of distribution: *transportation*
- How do you guarantee that each site has access to the material?

Electricity/water distribution

- Raw and finished material is the same
- Blurred distinction between production and customer sites
- Cable/duct reaches customer γ_1, it is then extended to customer γ_2 (γ_1 *is both production and customer*)
- The main cost is laying the cables/ducts



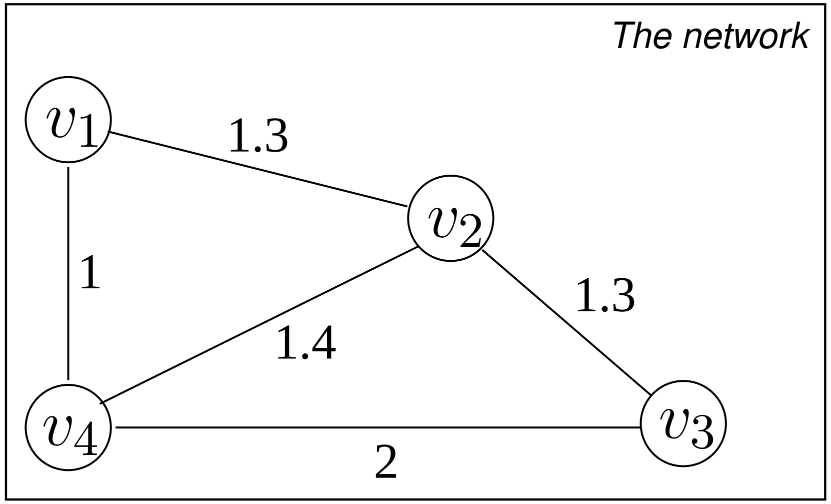
Spanning trees

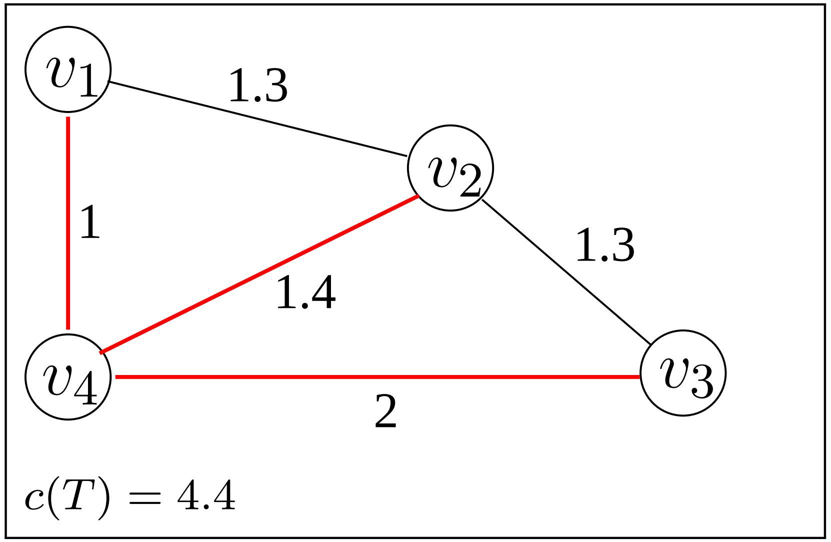
- Cost is optimized if material can be distributed to all sites using as few cables/duct as possible
- Recall: tree on $U \subseteq V$ is **spanning** if $U = V$
- If each edge e has weight/cost c_e, weight/cost of T is

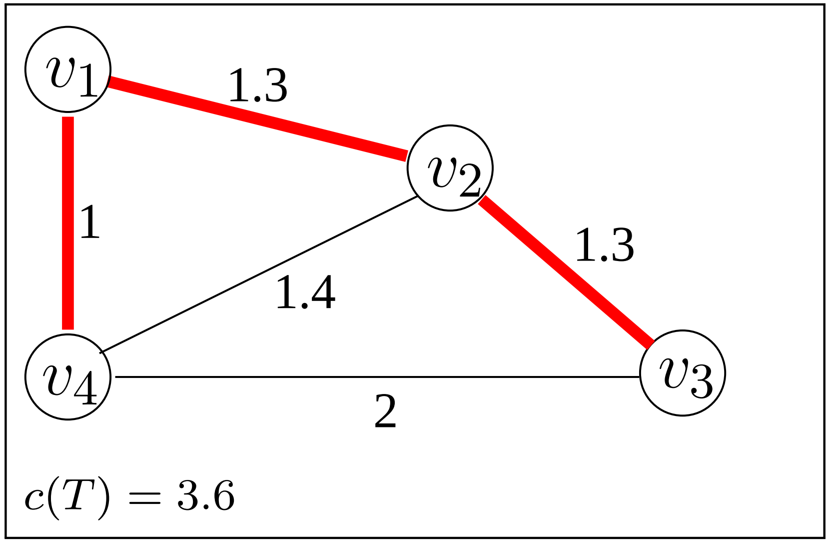
$$c(T) = \sum_{e \in E(T)} c_e$$

MINIMUM SPANNING TREE (MST): Given an undirected weighted graph $G = (V, E)$, find a spanning tree of minimum weight in G

Example


The network
v_1
v_2
v_3
v_4
1.3
1
1.4
1.3
2


v_1
v_2
v_3
v_4
1.3
1
1.4
1.3
2
$c(T) = 4.4$


v_1
v_2
v_3
v_4
1.3
1
1.4
1.3
2
$c(T) = 3.6$

Prim's algorithm

- **Idea**: grow connected subgraph from vertex s, min. cost at each step
- Data structures:
  - R: set of reached vertices (vertices in the tree)
  - F: set of edges in the tree
  - u: best next vertex
  - $\zeta : V \smallsetminus R \to \mathbb{R}$: cost of reaching from R a vertex outside R
  - $\pi : V \smallsetminus R \to R$: immediate predecessor in T to a vertex outside R.

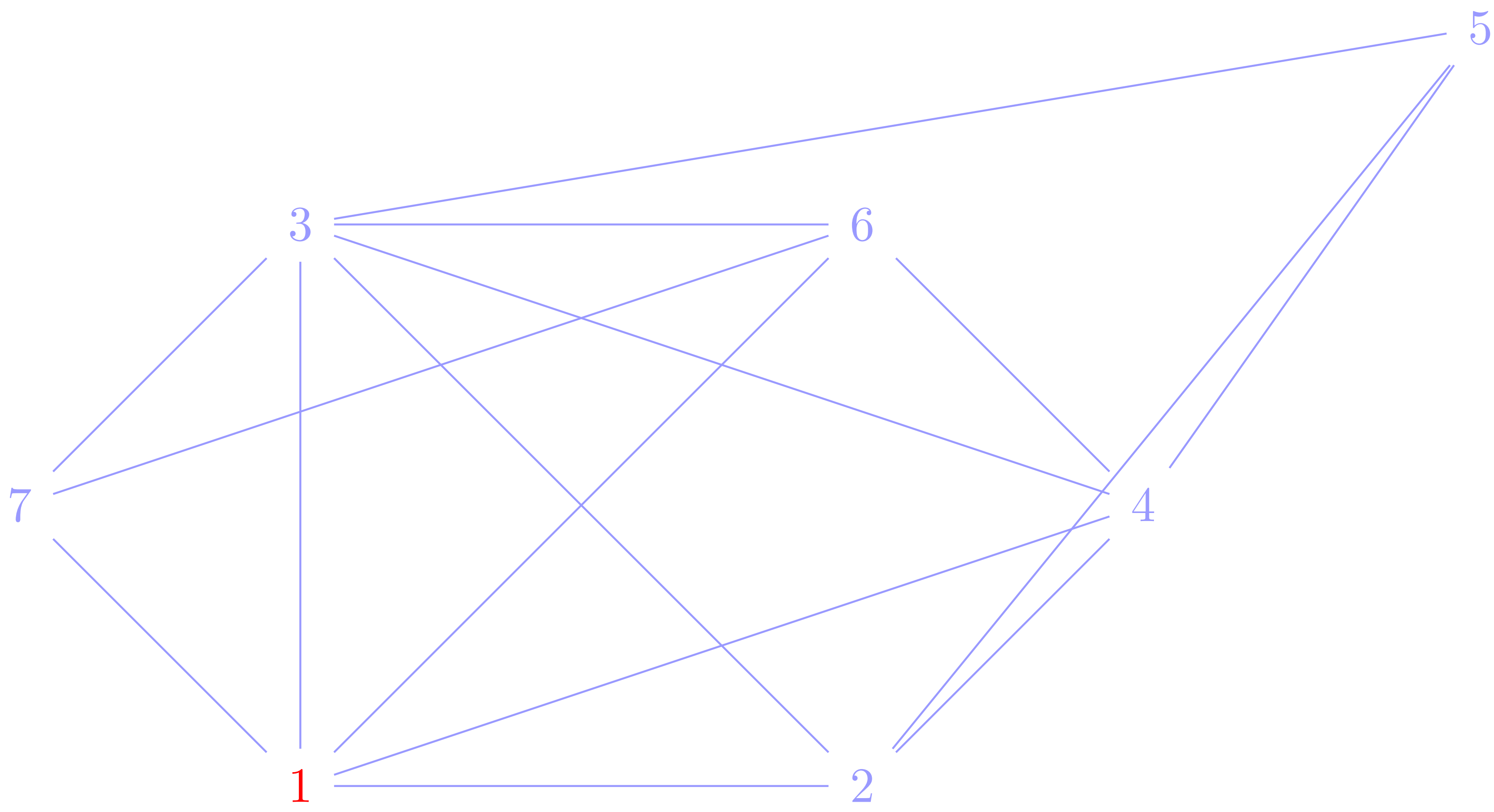
Prim's algorithm

- **Idea**: grow connected subgraph from vertex s, min. cost at each step
- Data structures:
  - R: set of reached vertices (vertices in the tree)
  - F: set of edges in the tree
  - u: best next vertex
  - $\zeta : V \smallsetminus R \to \mathbb{R}$: cost of reaching from R a vertex outside R
  - $\pi : V \smallsetminus R \to R$: immediate predecessor in T to a vertex outside R.

$R = \{s\}$, $F = \varnothing$, $\forall v \in V$ set $\zeta(v) = \infty$, $\pi(v) = s$
for $w \in N(s)$ **do**
    $\zeta(w) = c_{sw}$
end for
while $R \neq V$ **do**
    let $u \in V \smallsetminus R$ such that $\zeta(u)$ is minimum
    mark u as reached by adding it to R
    add the edge $\{\pi(u), u\}$ to T
    update ζ, π: $\forall v \in N(u)$ s.t. $\zeta(v) > c_{uv}$, let $\zeta(v) = c_{uv}$ and $\pi(v) = u$.
end while

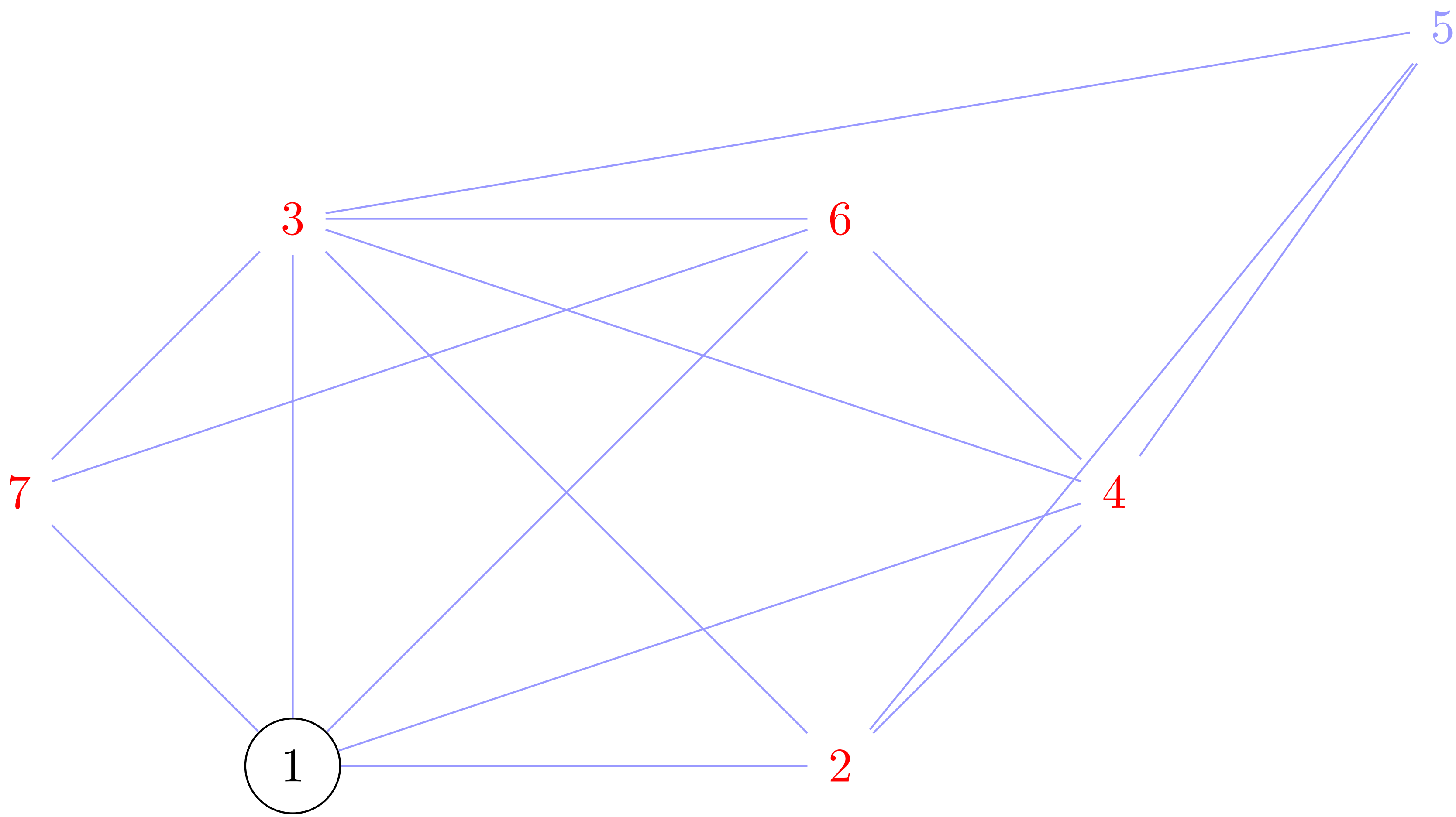
Prim's algorithm

$s = 1$



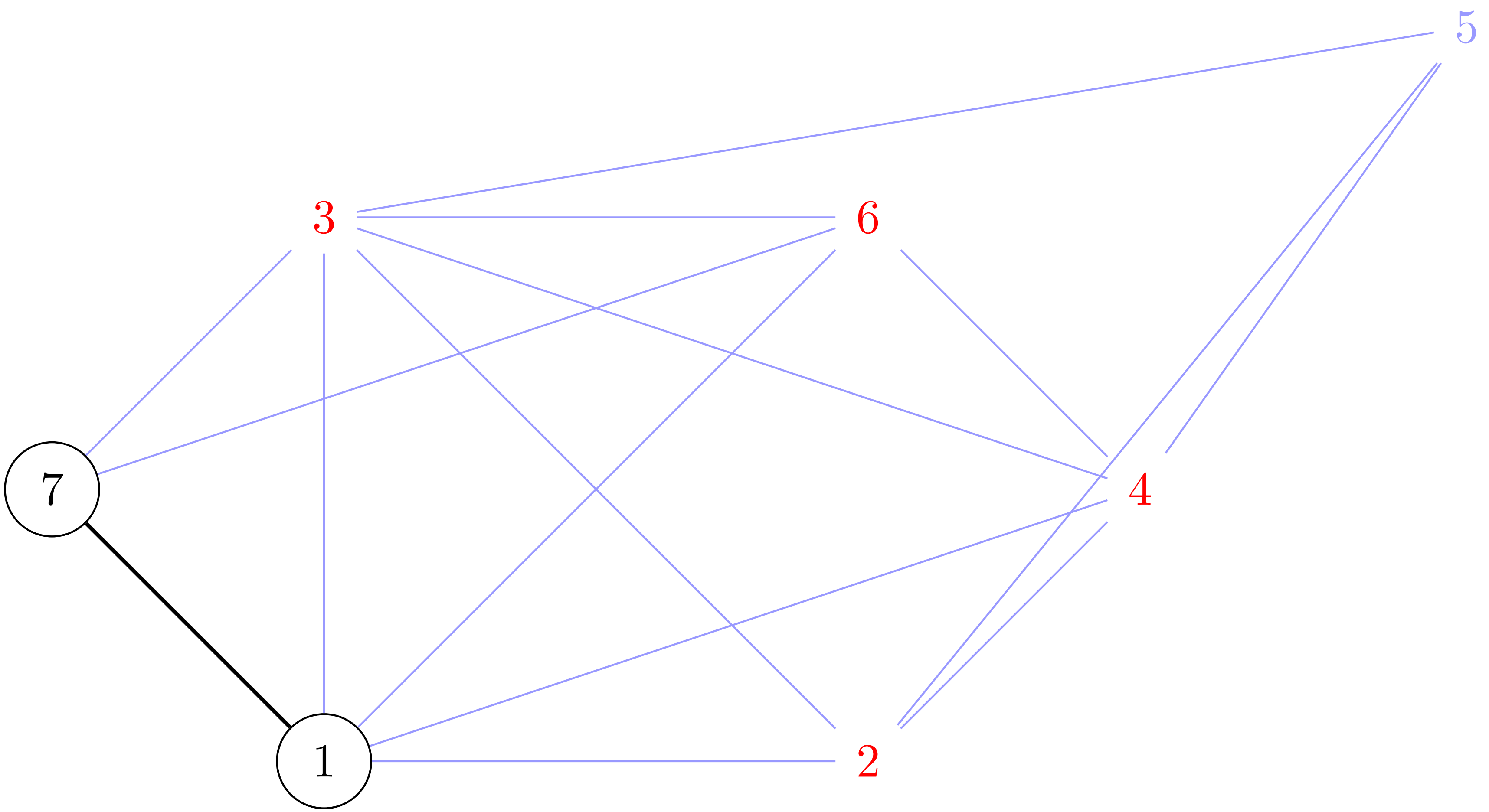
Prim's algorithm

$s = 1$



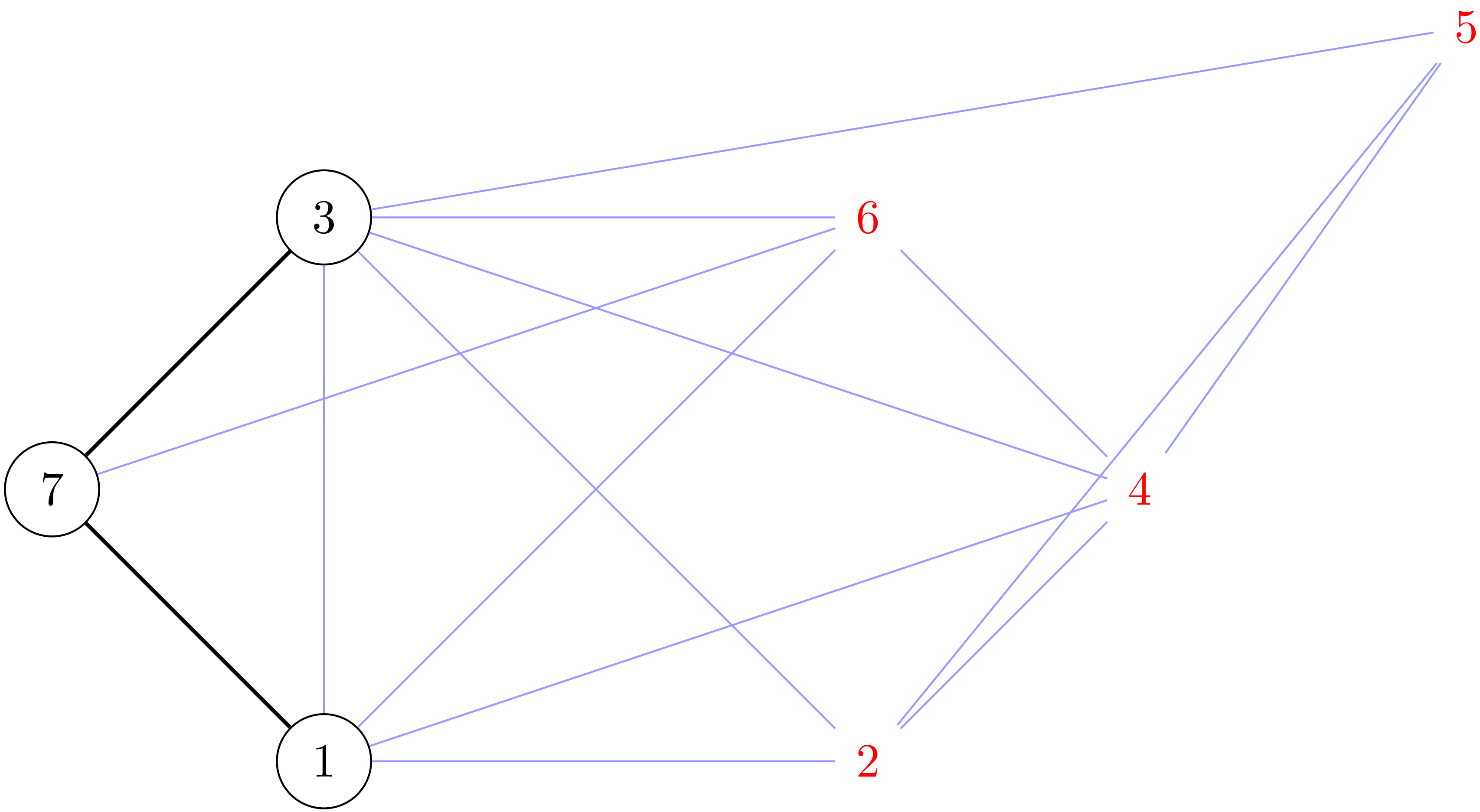
Prim's algorithm

$s = 1$



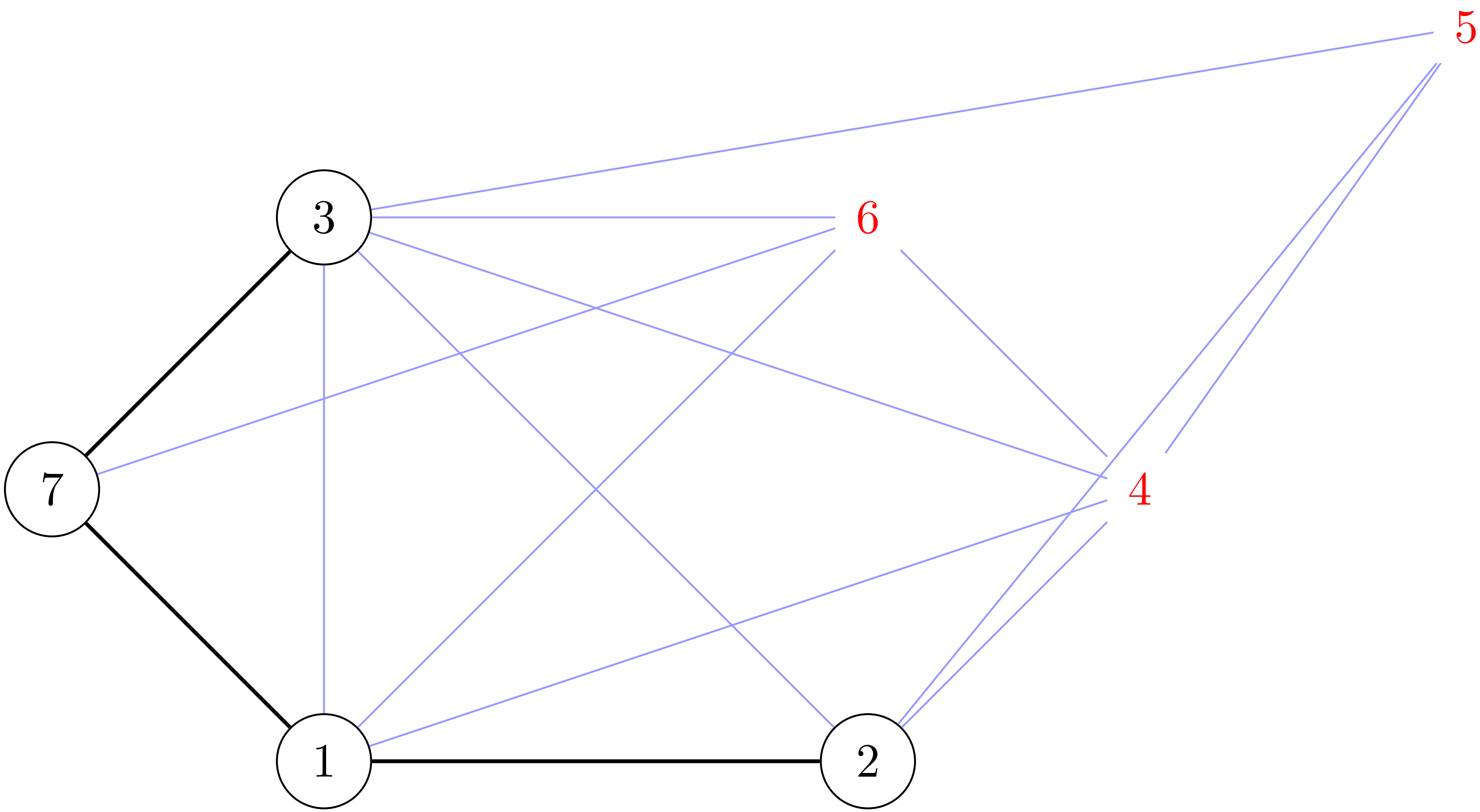
Prim's algorithm

$s = 1$



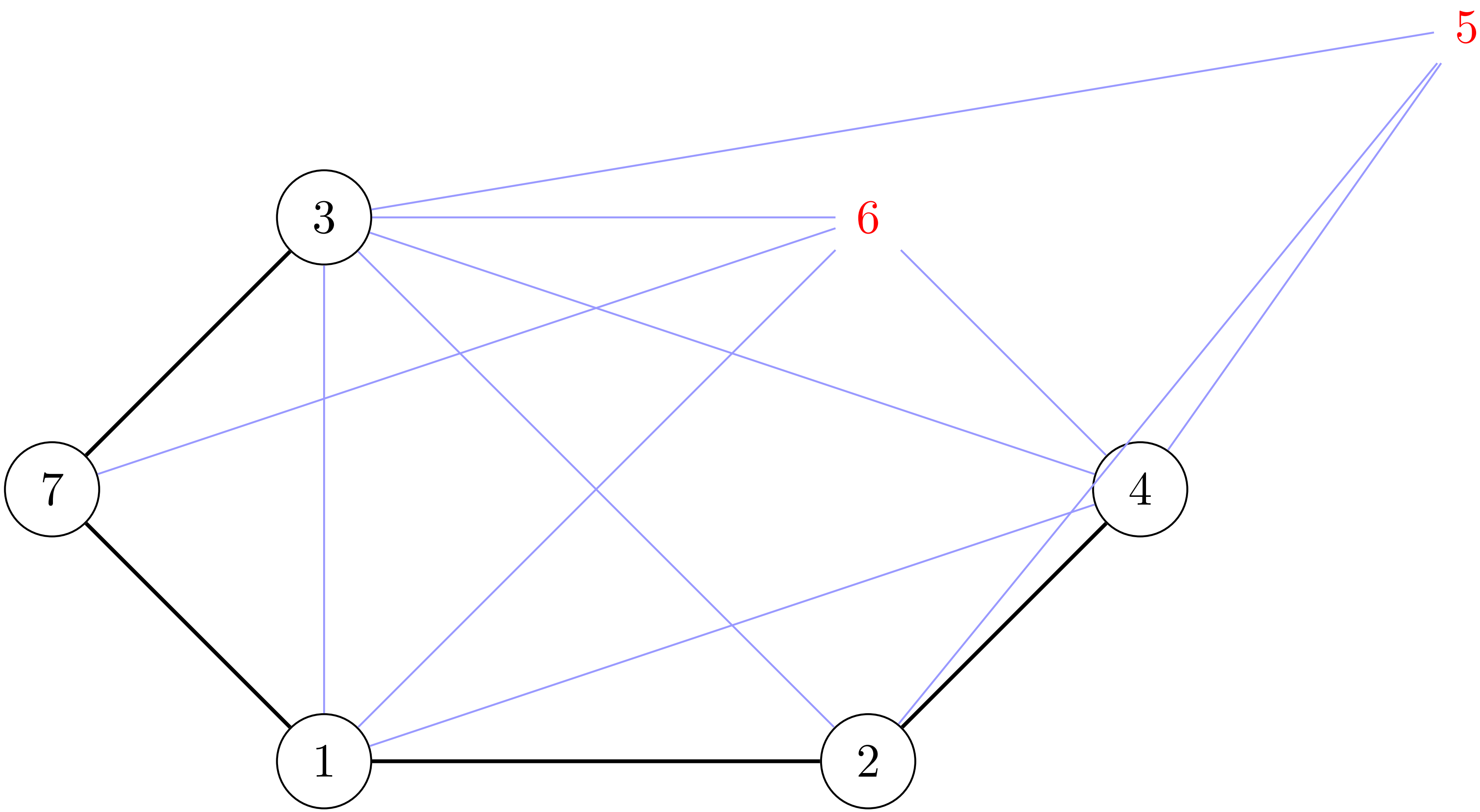
Prim's algorithm

$s = 1$



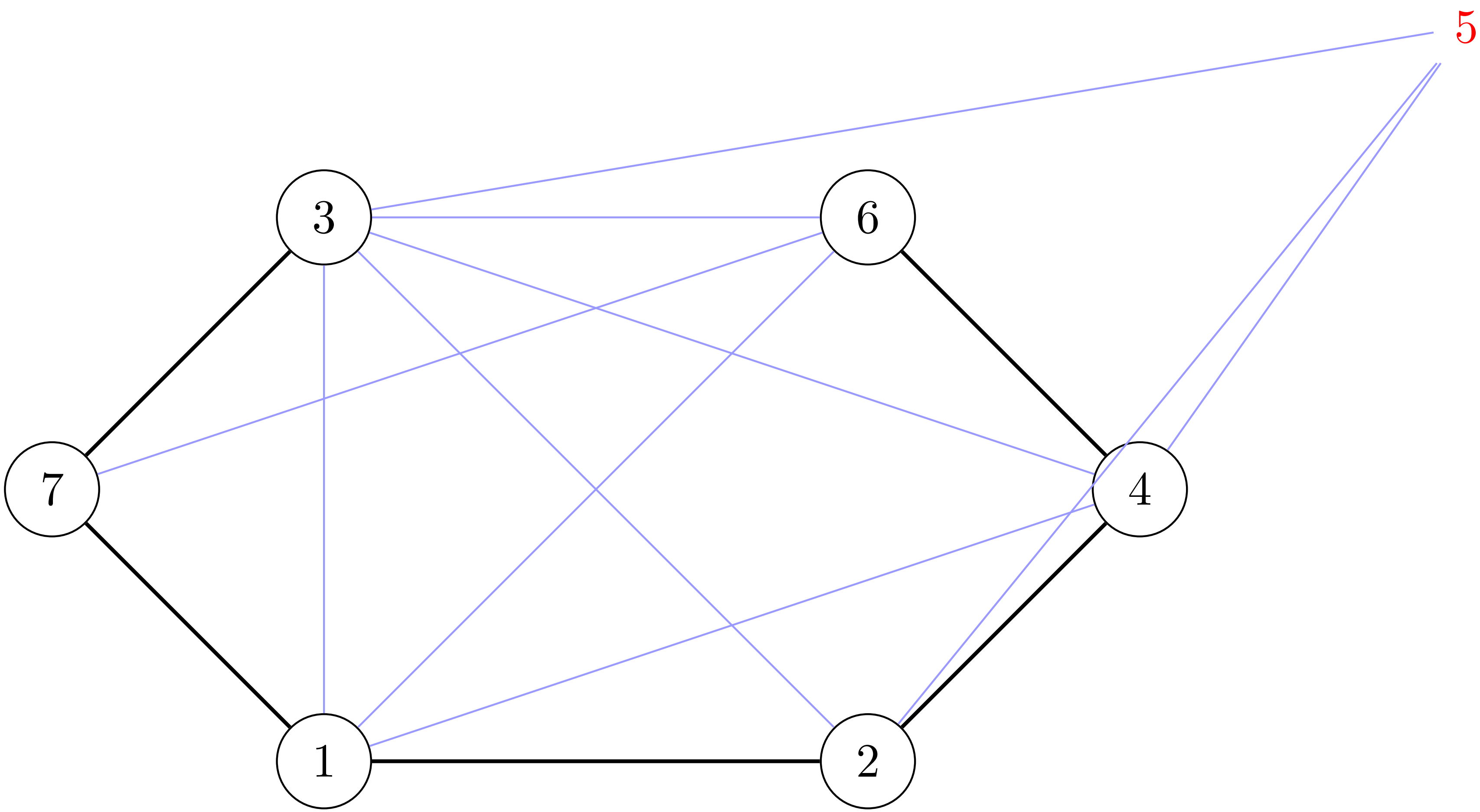
Prim's algorithm

$s = 1$



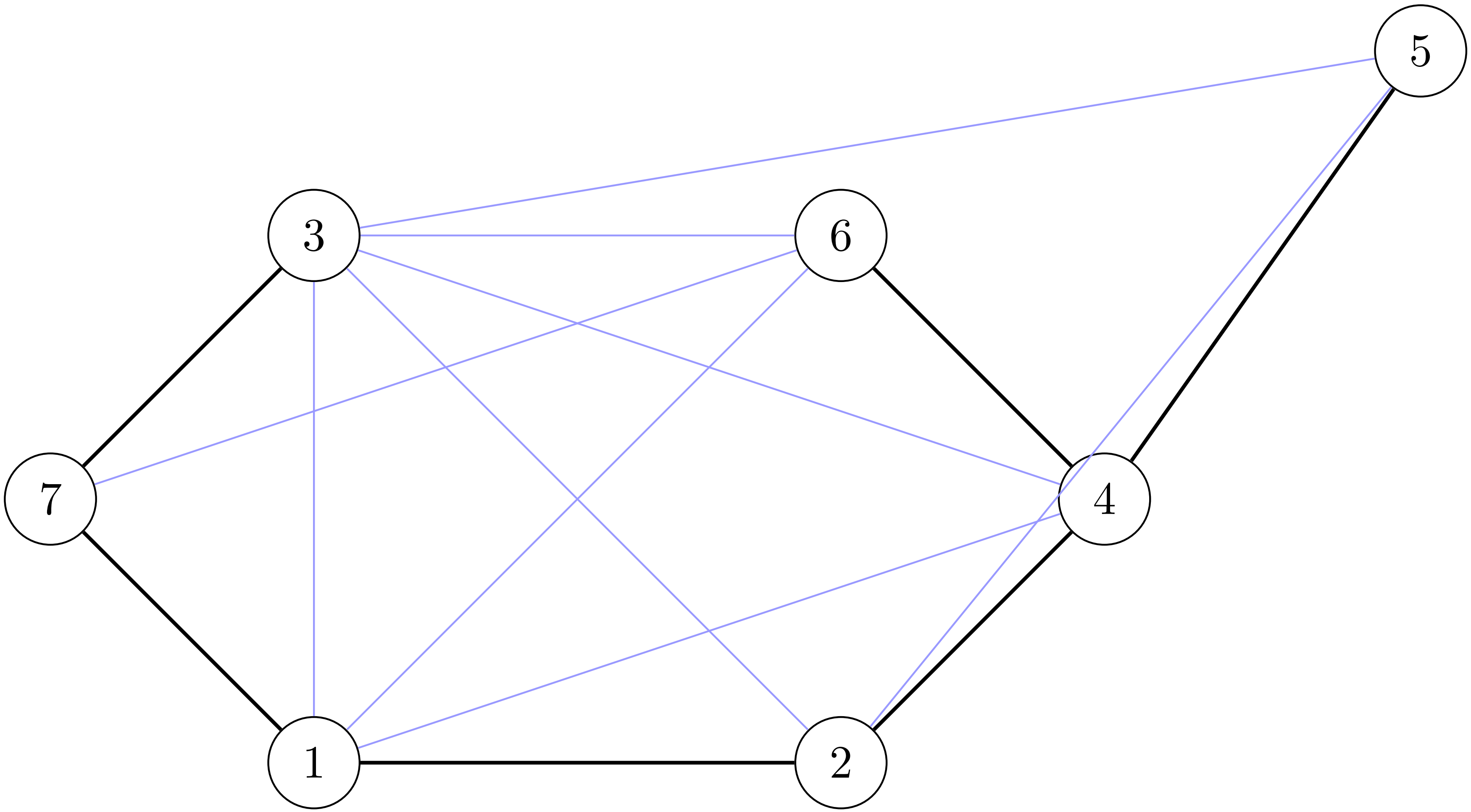
Prim's algorithm

$s = 1$



Prim's algorithm

$s = 1$



A local property with global scope

1. Local choice : at each step, choose best edge in cutset
2. Global optimum : end up with the globally optimal spanning tree

The reason

Thm.

Let T be a spanning tree of G, and $c : E(G) \to \mathbb{R}_+$
T has minimum cost $\Leftrightarrow$
$\forall \varnothing \subsetneq U \subsetneq V(G)\ \exists e \in E(T)\ (\,\delta(U) \cap E(T) = \{e\} \wedge c(e) = \min c(\delta(U))\,)$

Proof

($\Rightarrow$) By contradiction, if $\exists U$ with $f \in \delta(U)$ having $c_f < c_e$, then $T' = T \smallsetminus \{e\} \cup \{f\}$ has lower cost than T
($\Leftarrow$) Consider T' with $c(T') < c(T)$ and $T \cap T'$ as large as possible, take $f \in T \smallsetminus T'$, removing f from T determines $U \subsetneq V(G)$, consider unique $g \in \delta(U) \cap T'$, if $c(f) = c(g)$ then $T \cap T'$ as large as possible implies $f = g$, but $f \in T \smallsetminus T'$ yields a contradiction; otherwise $c(f) < c(g)$ by hypothesis, then $T'' = T' \smallsetminus \{g\} \cup \{f\}$ has $c(T'') < c(T)$, contradiction, hence $T \smallsetminus T' = \varnothing$, i.e. $T = T'$

Complexity

Worst-case complexity of Prim's algorithm: $O(n^2)$

Kruskal's algorithm: a sketch

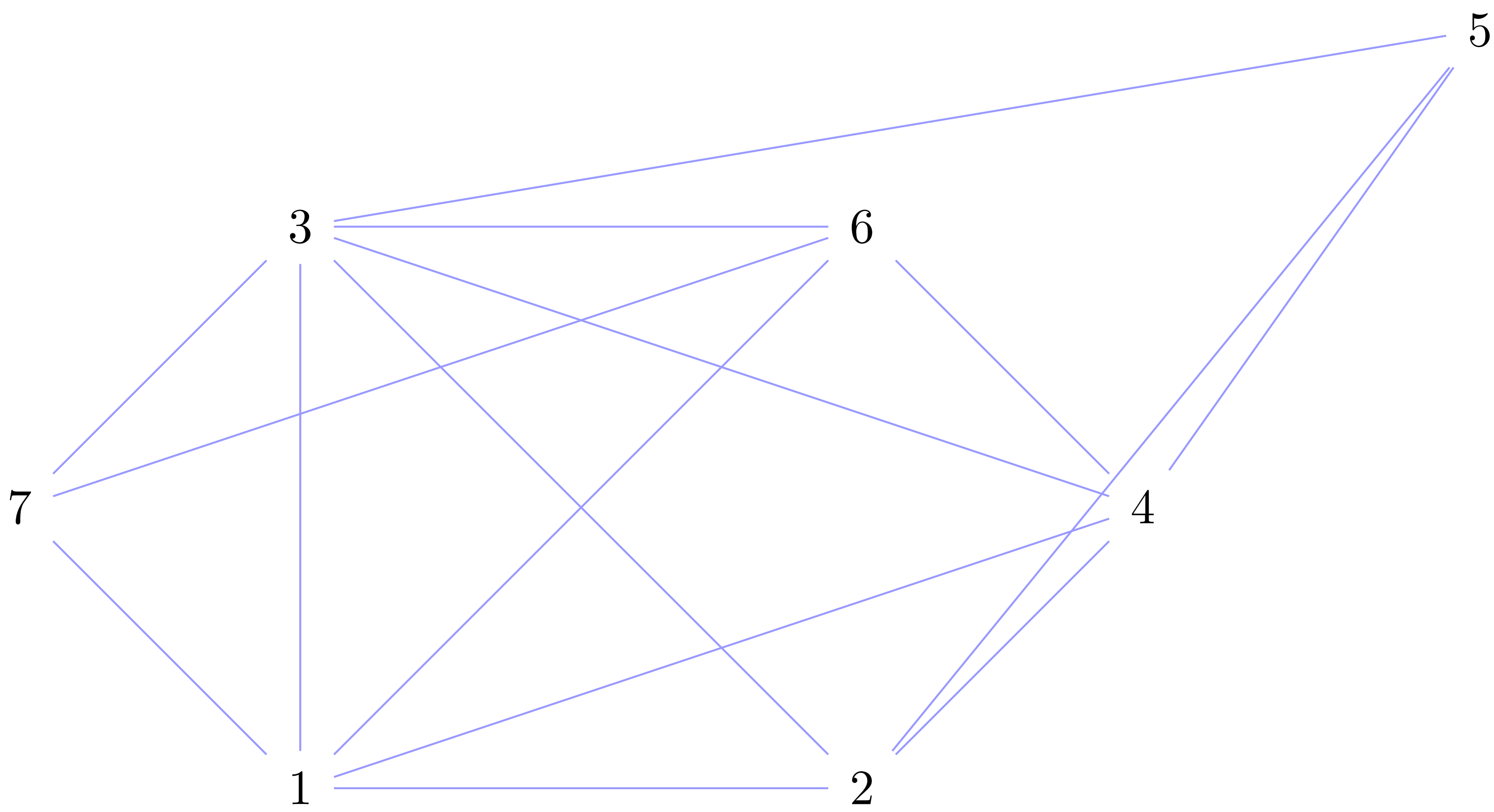
- **Idea**: grow subgraph keeping minimum cost, connect at termination
- *Implementation in INF431: requires union-find data structure*

  $T = \varnothing$
  while $|T| < |V| - 1$ **do**
      let $e =$ arg min$\{c_e \mid e \in E\}$
      if $T \cup \{e\}$ has no cycle **then**
          $T \leftarrow T \cup \{e\}$;
      end if
      $E \leftarrow E \smallsetminus \{e\}$;
  end while

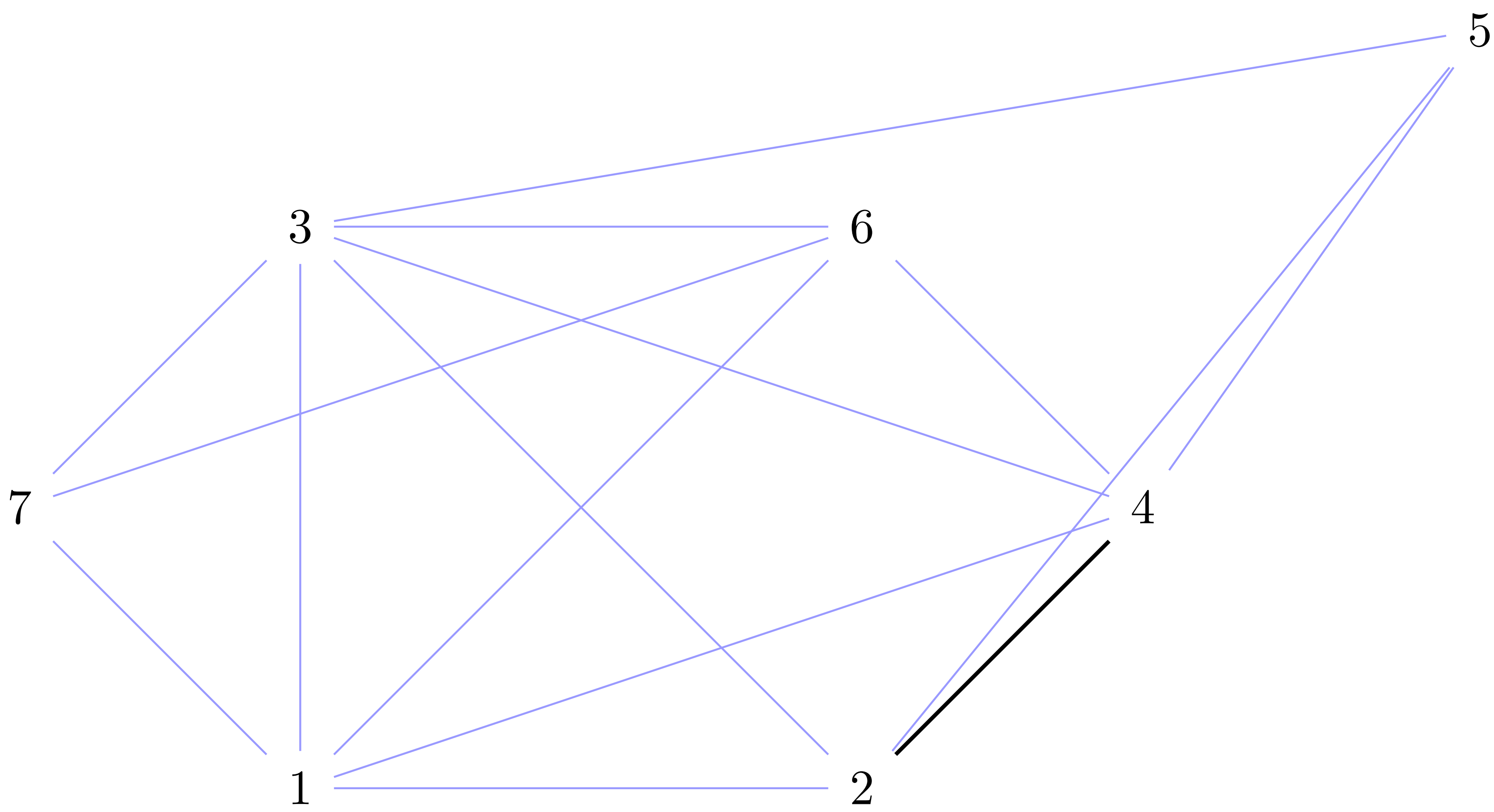
- At termination, T has $|V| - 1$ edges and no cycle
- $\Rightarrow$ A tree by definition

Worst-case complexity of best implementation: $O(m \log n)$

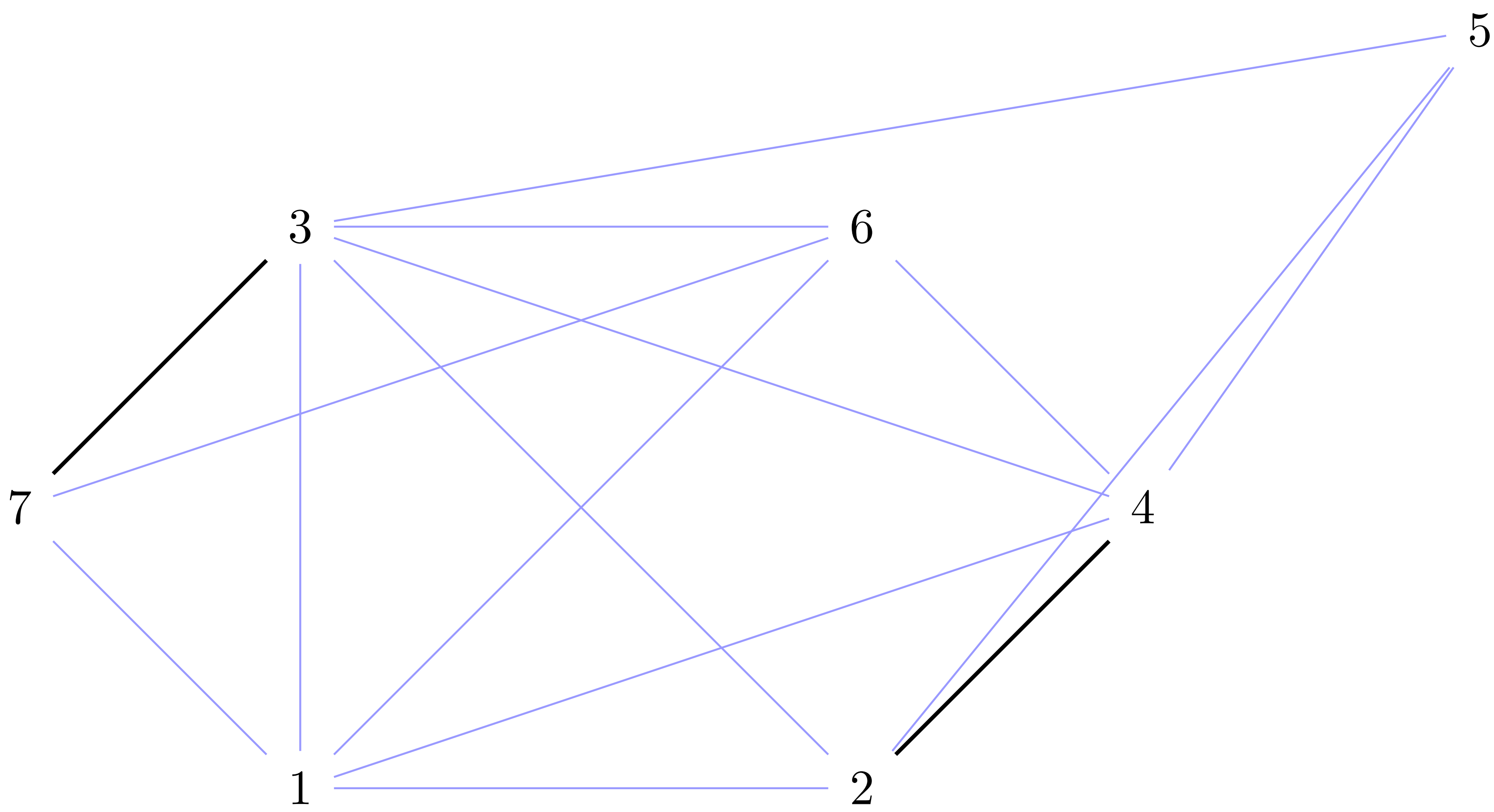
Kruskal's algorithm



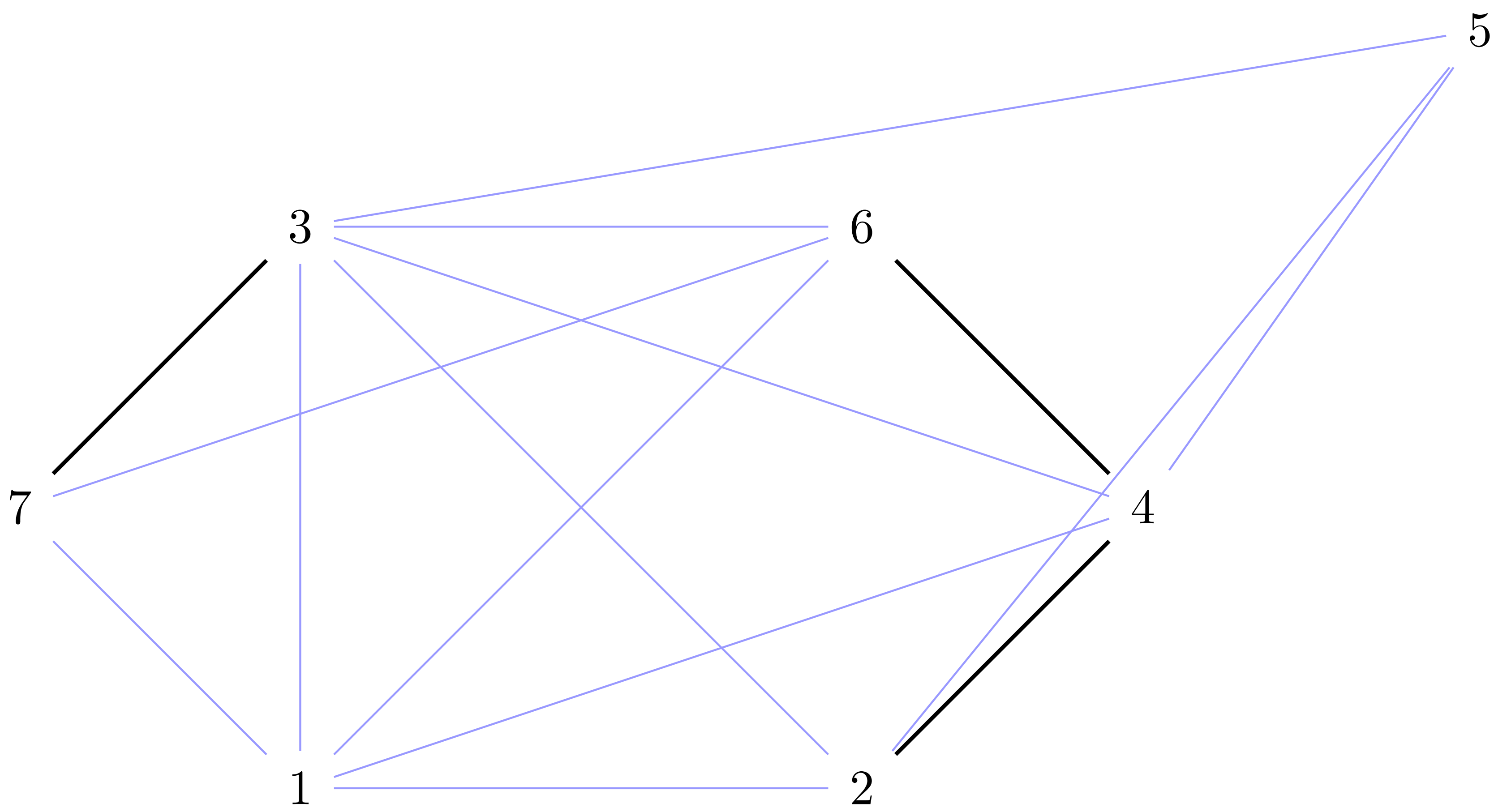
Kruskal's algorithm



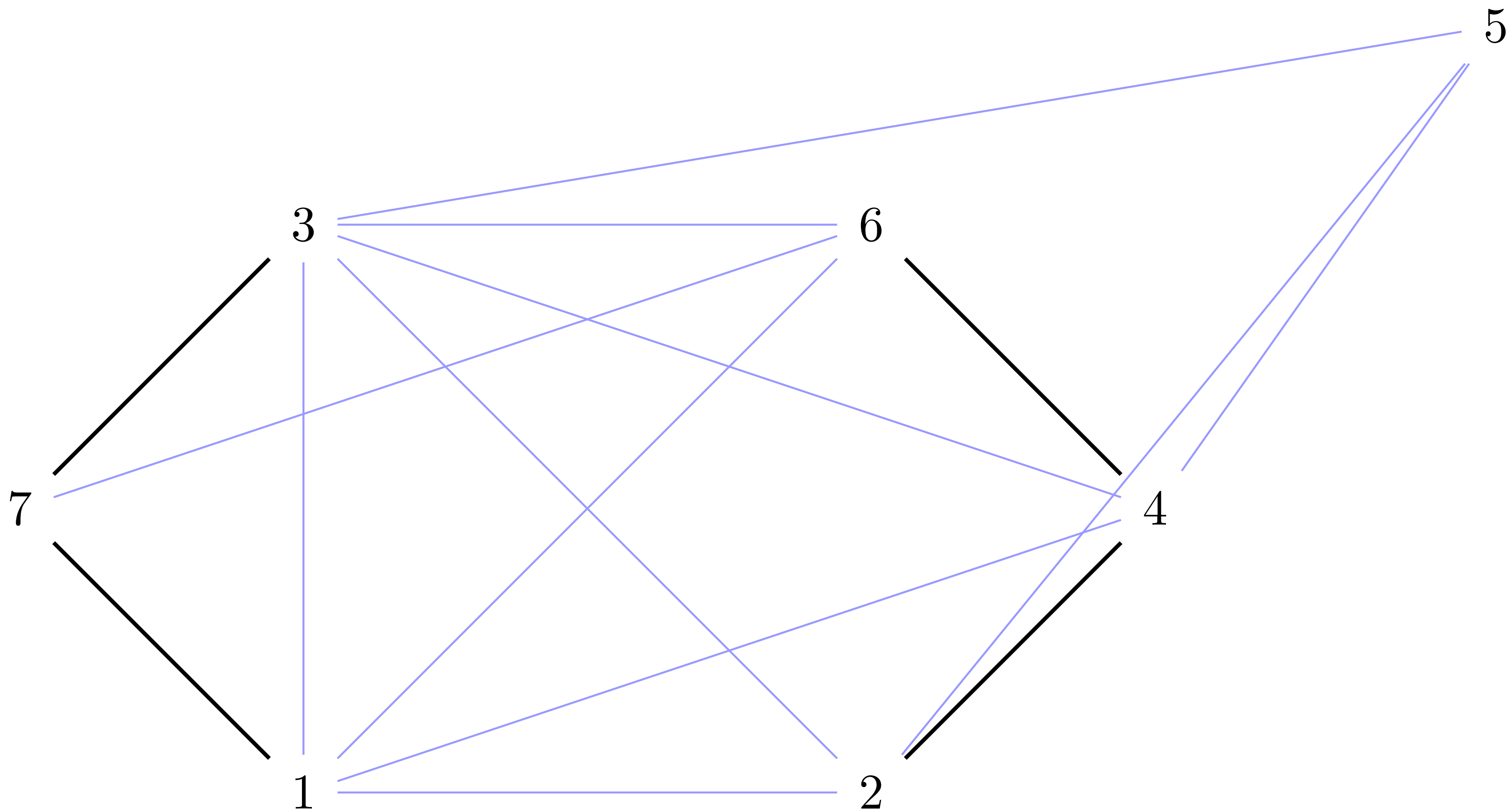
Kruskal's algorithm



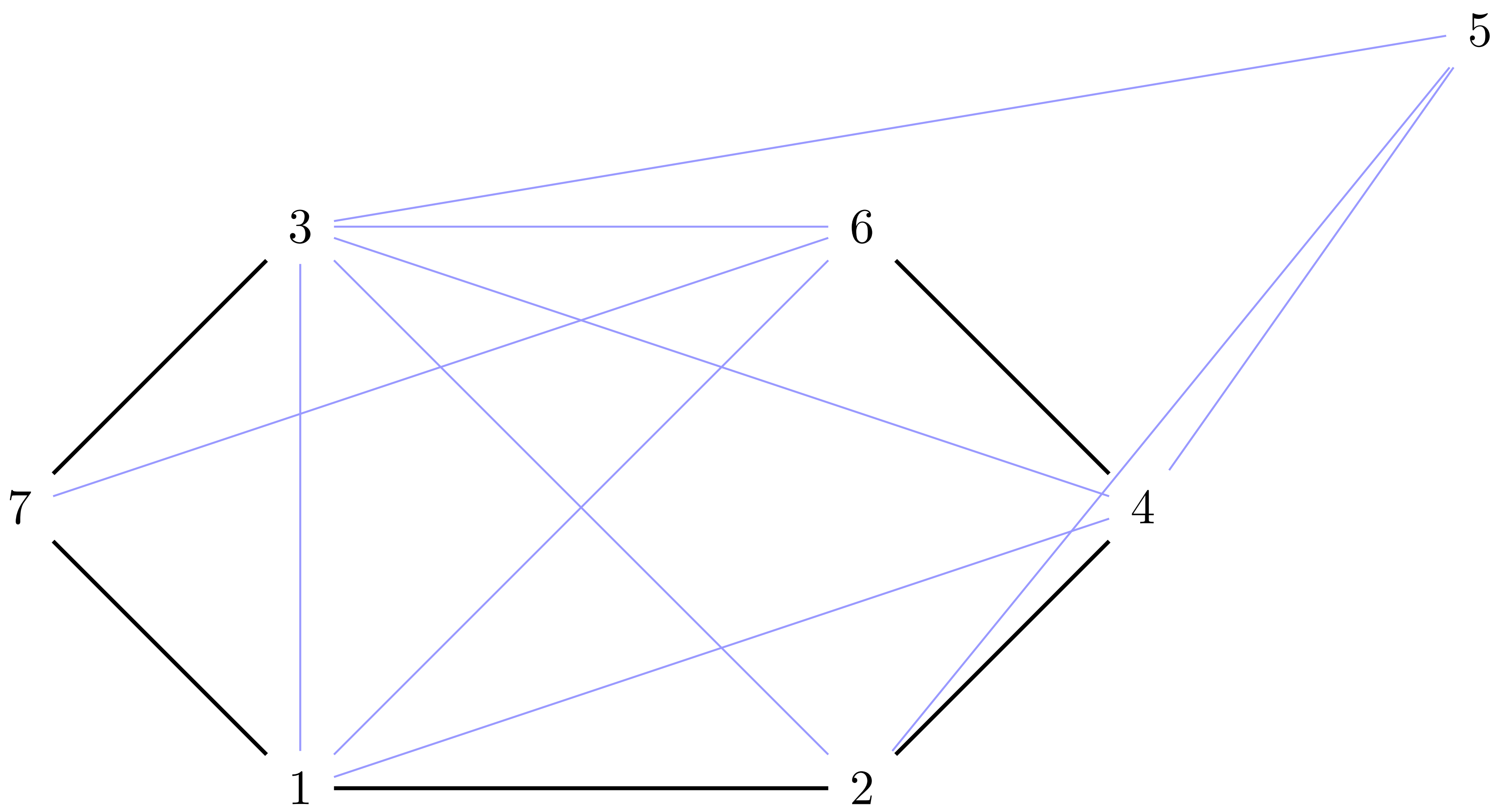
Kruskal's algorithm



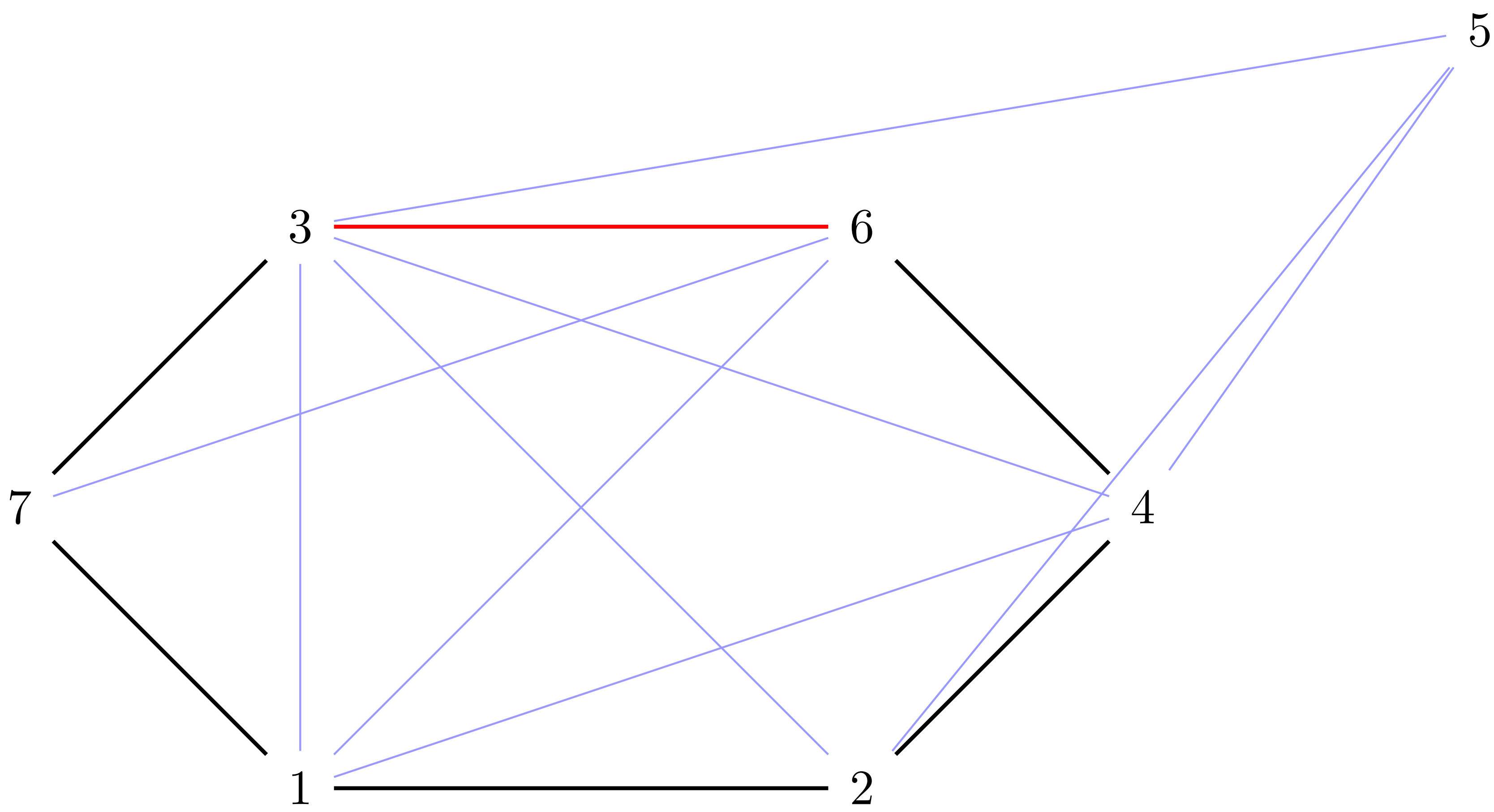
Kruskal's algorithm



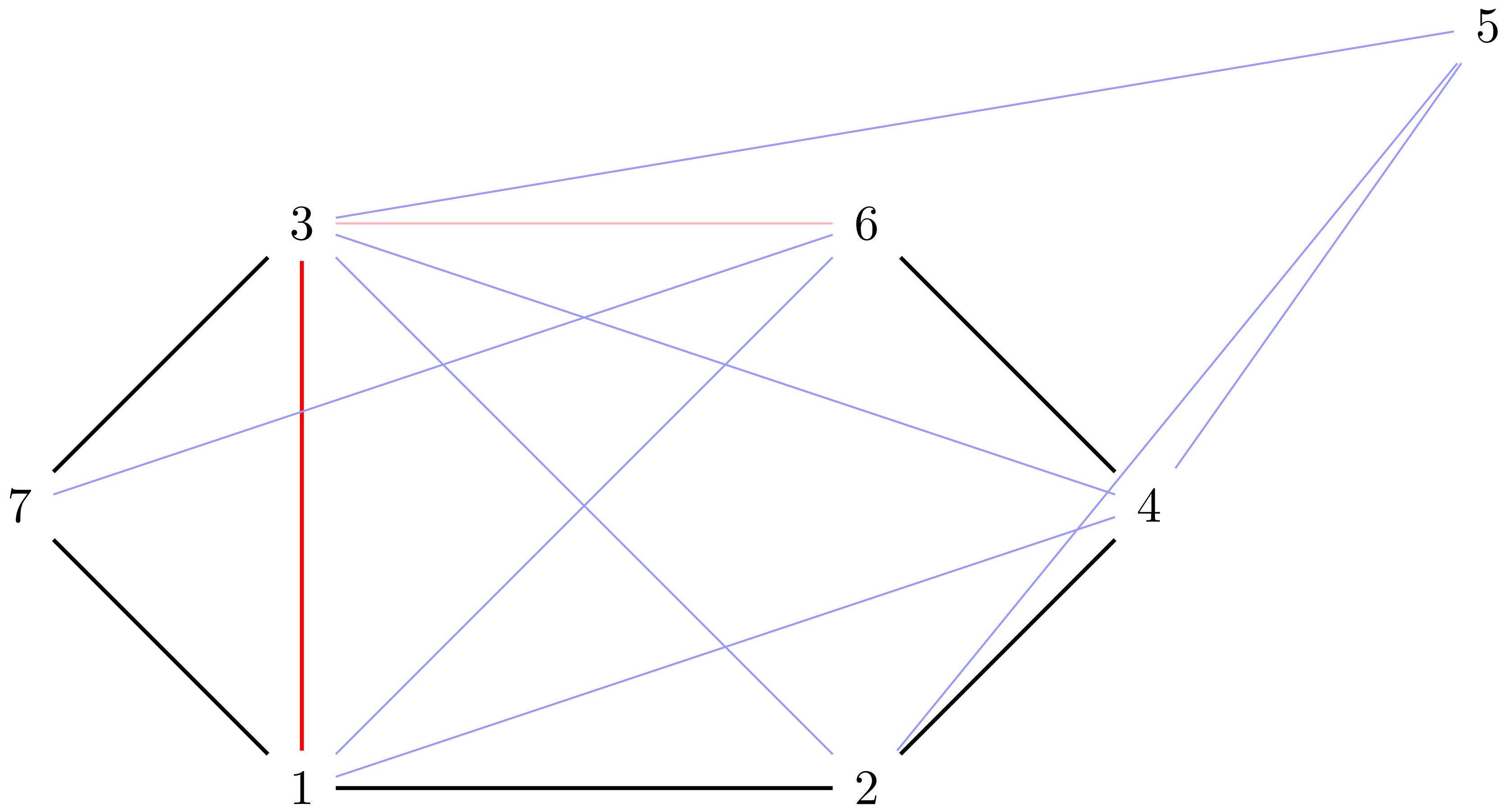
Kruskal's algorithm



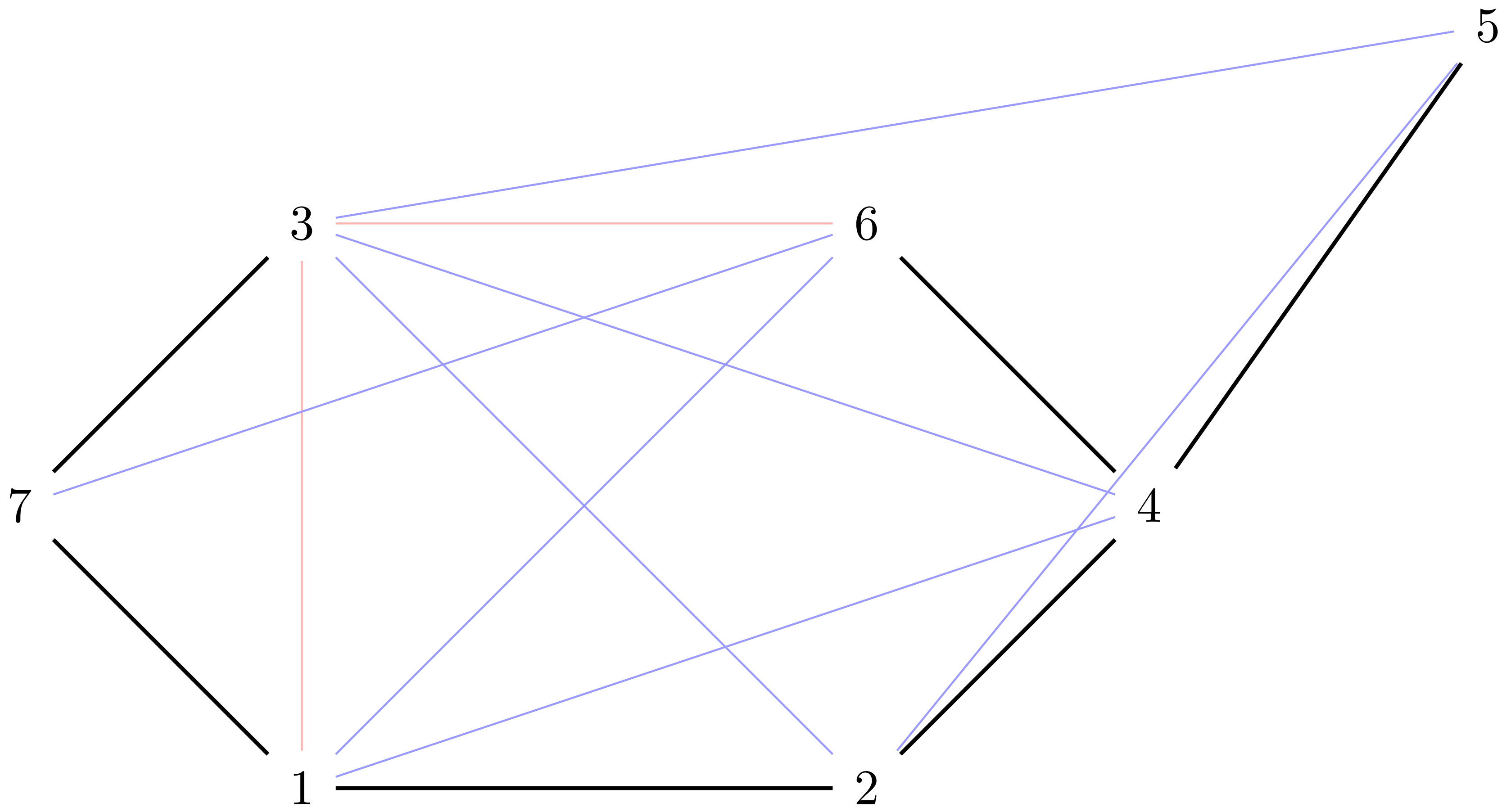
Kruskal's algorithm



Kruskal's algorithm



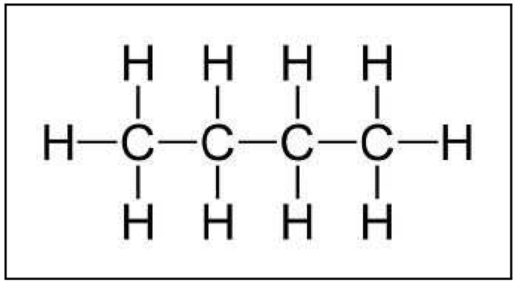
Kruskal's algorithm



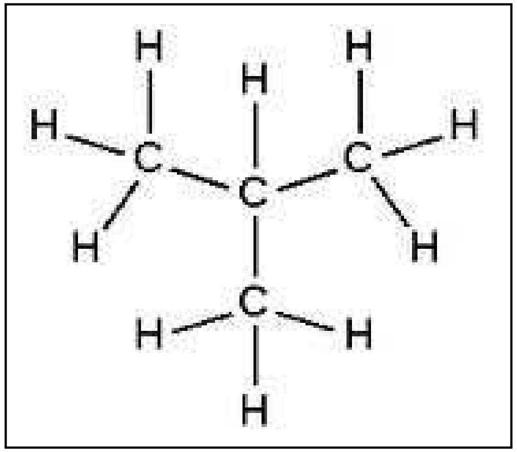
Chemical trees

Molecular descriptions

- Until mid-XIX century: molecules are completely defined by their atomic formula
- E.g. paraffins are C_kH_{2k+2}
- Experiments showed different bond relations give rise to substances with different properties: **isomers**



butane



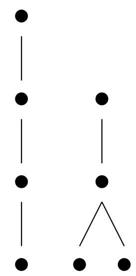
isobutane

Listing isomers

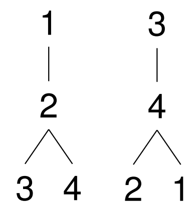
- Carbons have valence 4
- Hydrogens have valence 1
- Paraffins known to have tree-like bond relations
- Finding paraffin isomers in the mid-XIX century:
  - list all trees on $n = 3k + 2$ nodes
  - remove those whose valences does not match the paraffin chemical formula
- How do we list all trees? How many are there?

Listing labelled trees

- Two possible interpretations
- These two are different **unlabelled trees**:



- These two are different **labelled trees**:



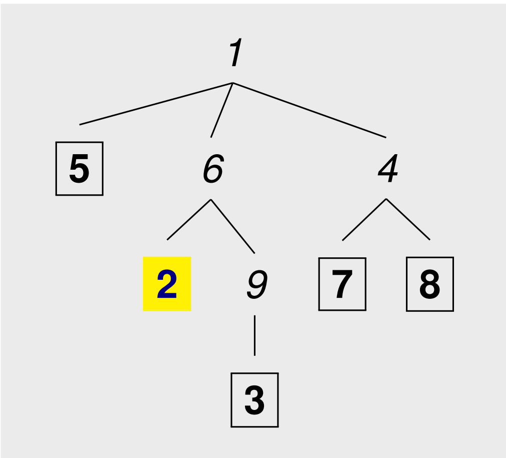
- Listing labelled trees is easier
- $\exists$ more labelled than unlabelled trees

Prüfer sequences

Mapping trees on V to sequences in $V^{|V|-2}$

- For a tree T let $L(T)$ be the set of leaf nodes of T

for $k \in \{1, \ldots, |V|-2\}$ **do**
  $v = \min L(T)$;
  let e be the only edge incident to v;
  let $t_k \neq v$ be the other node incident to e;
  $T \leftarrow T \smallsetminus \{v\}$;
end for
return $t = (t_1, \ldots, t_{|V|-2})$



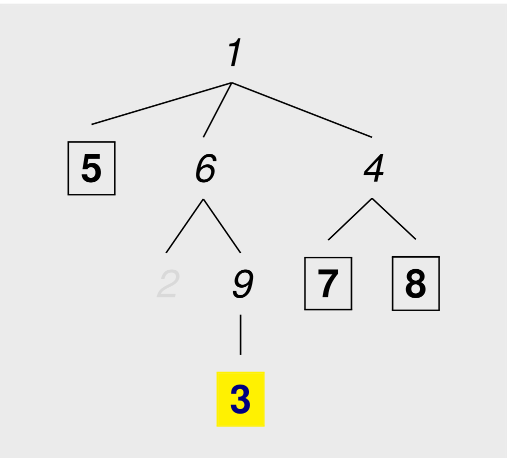
$$\boxed{L(T) = \{5,2,3,7,8\},\ v = 2,\ t = (6)}$$

Prüfer sequences

Mapping trees on V to sequences in $V^{|V|-2}$

- For a tree T let $L(T)$ be the set of leaf nodes of T

for $k \in \{1, \ldots, |V|-2\}$ **do**
  $v = \min L(T)$;
  let e be the only edge incident to v;
  let $t_k \neq v$ be the other node incident to e;
  $T \leftarrow T \setminus \{v\}$;
end for
return $t = (t_1, \ldots, t_{|V|-2})$



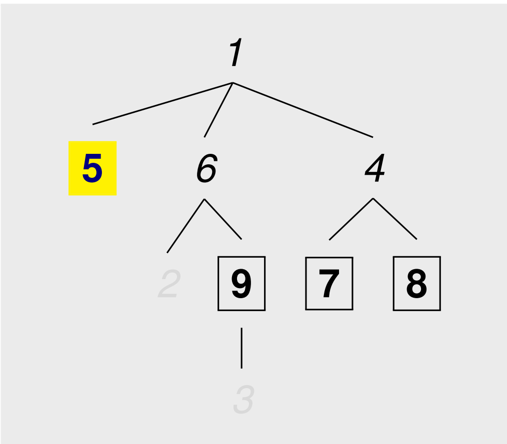
$$L(T) = \{5, 3, 7, 8\},\ v = 3,\ t = (6, 9)$$

Prüfer sequences

Mapping trees on V to sequences in $V^{|V|-2}$

- For a tree T let $L(T)$ be the set of leaf nodes of T

for $k \in \{1, \ldots, |V|-2\}$ **do**
  $v = \min L(T)$;
  let e be the only edge incident to v;
  let $t_k \neq v$ be the other node incident to e;
  $T \leftarrow T \smallsetminus \{v\}$;
end for
return $t = (t_1, \ldots, t_{|V|-2})$



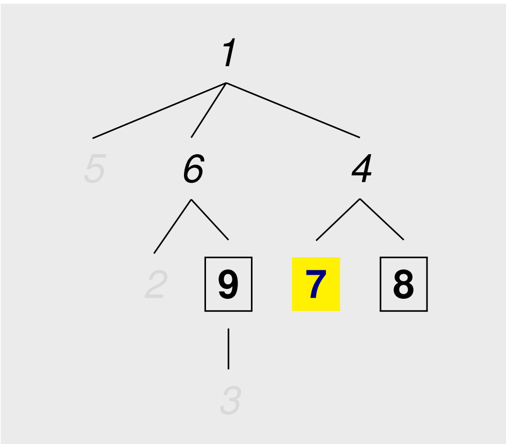
$$L(T) = \{5, 7, 8, 9\},\ v = 5,\ t = (6, 9, 1)$$

Prüfer sequences

Mapping trees on V to sequences in $V^{|V|-2}$

- For a tree T let $L(T)$ be the set of leaf nodes of T

for $k \in \{1, \ldots, |V|-2\}$ **do**
  $v = \min L(T)$;
  let e be the only edge incident to v;
  let $t_k \neq v$ be the other node incident to e;
  $T \leftarrow T \smallsetminus \{v\}$;
end for
return $t = (t_1, \ldots, t_{|V|-2})$



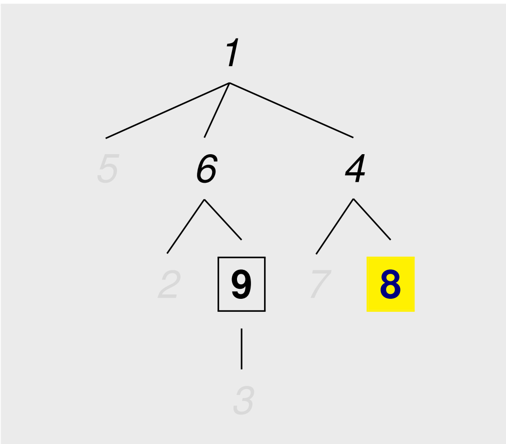
$\boxed{L(T) = \{7, 8, 9\},\ v = 7,\ t = (6, 9, 1, 4)}$

Prüfer sequences

Mapping trees on V to sequences in $V^{|V|-2}$

- For a tree T let $L(T)$ be the set of leaf nodes of T

for $k \in \{1, \ldots, |V|-2\}$ **do**
  $v = \min L(T)$;
  let e be the only edge incident to v;
  let $t_k \neq v$ be the other node incident to e;
  $T \leftarrow T \smallsetminus \{v\}$;
end for
return $t = (t_1, \ldots, t_{|V|-2})$



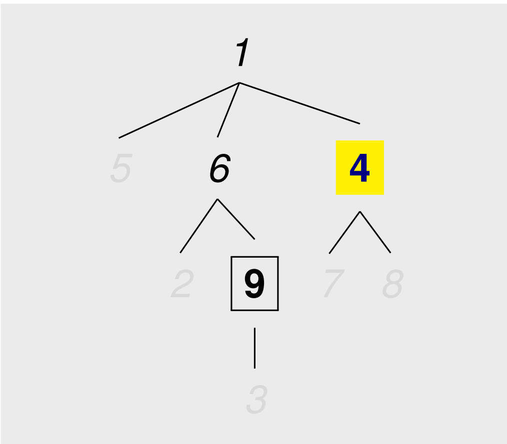
$L(T) = \{8, 9\}$, $v = 8$, $t = (6, 9, 1, 4, 4)$

Prüfer sequences

Mapping trees on V to sequences in $V^{|V|-2}$

- For a tree T let $L(T)$ be the set of leaf nodes of T

for $k \in \{1, \ldots, |V|-2\}$ **do**
  $v = \min L(T)$;
  let e be the only edge incident to v;
  let $t_k \neq v$ be the other node incident to e;
  $T \leftarrow T \setminus \{v\}$;
end for
return $t = (t_1, \ldots, t_{|V|-2})$



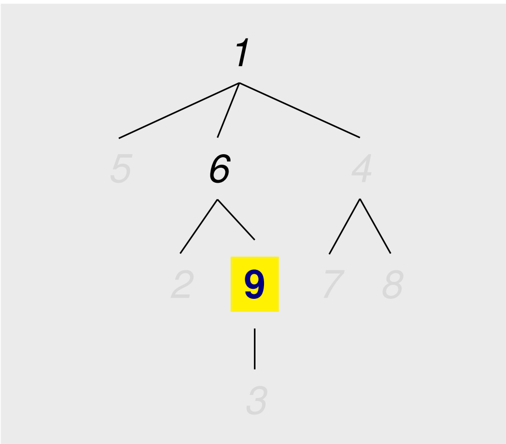
$$\boxed{L(T) = \{9, 4\},\ v = 4,\ t = (6, 9, 1, 4, 4, 1)}$$

Prüfer sequences

Mapping trees on V to sequences in $V^{|V|-2}$

- For a tree T let $L(T)$ be the set of leaf nodes of T

```
for k ∈ {1, ..., |V| − 2} do
    v = min L(T);
    let e be the only edge incident to v;
    let t_k ≠ v be the other node incident to e;
    T ← T ∖ {v};
end for
return t = (t_1, ..., t_{|V|−2})
```



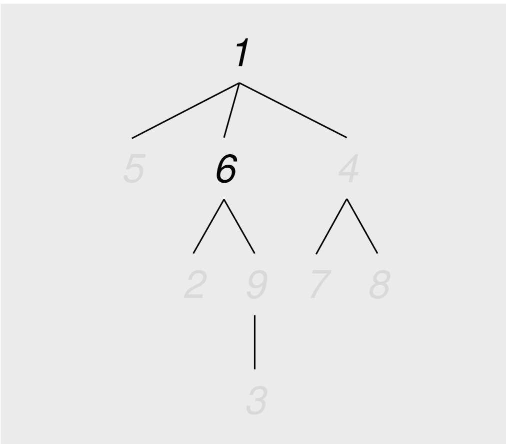
$$\boxed{L(T) = \{9\},\ v = 9,\ t = (6, 9, 1, 4, 4, 1, 6)}$$

Prüfer sequences

Mapping trees on V to sequences in $V^{|V|-2}$

- For a tree T let $L(T)$ be the set of leaf nodes of T

for $k \in \{1, \ldots, |V|-2\}$ **do**
  $v = \min L(T)$;
  let e be the only edge incident to v;
  let $t_k \neq v$ be the other node incident to e;
  $T \leftarrow T \smallsetminus \{v\}$;
end for
return $t = (t_1, \ldots, t_{|V|-2})$



$L(T) = \{6\}$, $t = (6, 9, 1, 4, 4, 1, 6)$, **stop**

Back to the trees

Mapping $V^{|V|-2}$ to trees

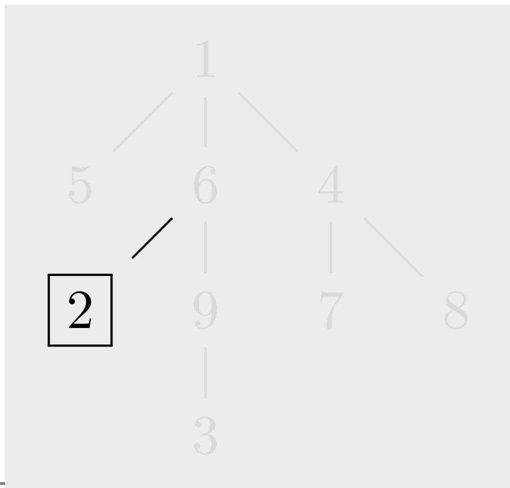
1. Given a Prüfer sequence t on V, e.g. $(6, 9, 1, 4, 4, 1, 6)$
2. Find smallest index ℓ in $V \smallsetminus t = \{2, 3, 5, 7, 8\}$, e.g. 2
3. Add $\{\ell, t_1\}$ to T, e.g. $\{2, 6\} \in E(T)$
4. Remove t_1 from t, e.g. $t = (9, 1, 4, 4, 1, 6)$
5. Remove ℓ from V, e.g. $V \smallsetminus t = \{3, 5, 7, 8\}$
6. Repeat from Step 2 until $t = \varnothing$
7. At this point $|V \smallsetminus t| = 2$ (it is an edge): add it

$$\boxed{\begin{array}{l} V \smallsetminus t = \{2, 3, 5, 7, 8\}, \\ t = (6, 9, 1, 4, 4, 1, 6) \end{array}}$$

Back to the trees

Mapping $V^{|V|-2}$ to trees

1. Given a Prüfer sequence t on V, e.g. $(6, 9, 1, 4, 4, 1, 6)$
2. Find smallest index ℓ in $V \smallsetminus t = \{2, 3, 5, 7, 8\}$, e.g. 2
3. Add $\{\ell, t_1\}$ to T, e.g. $\{2, 6\} \in E(T)$
4. Remove t_1 from t, e.g. $t = (9, 1, 4, 4, 1, 6)$
5. Remove ℓ from V, e.g. $V \smallsetminus t = \{3, 5, 7, 8\}$
6. Repeat from Step 2 until $t = \varnothing$
7. At this point $|V \smallsetminus t| = 2$ (it is an edge): add it

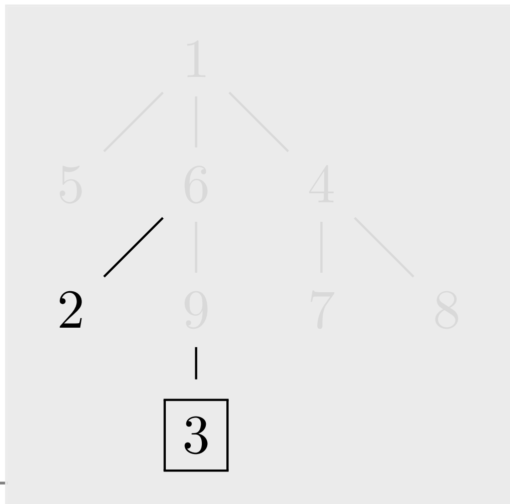


$V \smallsetminus t = \{\boxed{2}, 3, 5, 7, 8\}$, $\ell = 2$,
$t = (6, 9, 1, 4, 4, 1, 6)$, edge $\{2, 6\}$

Back to the trees

Mapping $V^{|V|-2}$ to trees

1. Given a Prüfer sequence t on V, e.g. $(6, 9, 1, 4, 4, 1, 6)$
2. Find smallest index ℓ in $V \smallsetminus t = \{2, 3, 5, 7, 8\}$, e.g. 2
3. Add $\{\ell, t_1\}$ to T, e.g. $\{2, 6\} \in E(T)$
4. Remove t_1 from t, e.g. $t = (9, 1, 4, 4, 1, 6)$
5. Remove ℓ from V, e.g. $V \smallsetminus t = \{3, 5, 7, 8\}$
6. Repeat from Step 2 until $t = \varnothing$
7. At this point $|V \smallsetminus t| = 2$ (it is an edge): add it

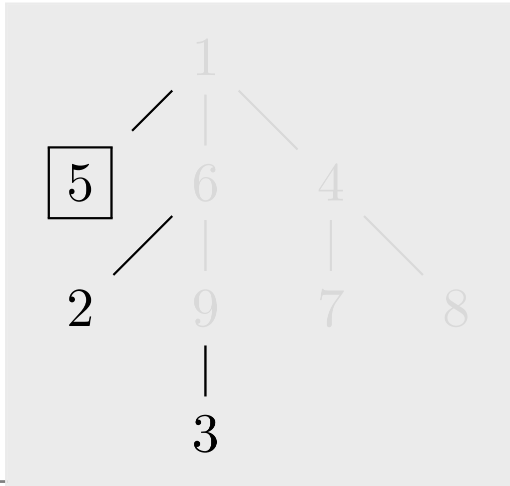


$V \smallsetminus t = \{\boxed{3}, 5, 7, 8\}$, $\ell = 3$,
$t = (9, 1, 4, 4, 1, 6)$, edge $\{3, 9\}$

Back to the trees

Mapping $V^{|V|-2}$ to trees

1. Given a Prüfer sequence t on V, e.g. $(6, 9, 1, 4, 4, 1, 6)$
2. Find smallest index ℓ in $V \setminus t = \{2, 3, 5, 7, 8\}$, e.g. 2
3. Add $\{\ell, t_1\}$ to T, e.g. $\{2, 6\} \in E(T)$
4. Remove t_1 from t, e.g. $t = (9, 1, 4, 4, 1, 6)$
5. Remove ℓ from V, e.g. $V \setminus t = \{3, 5, 7, 8\}$
6. Repeat from Step 2 until $t = \varnothing$
7. At this point $|V \setminus t| = 2$ (it is an edge): add it

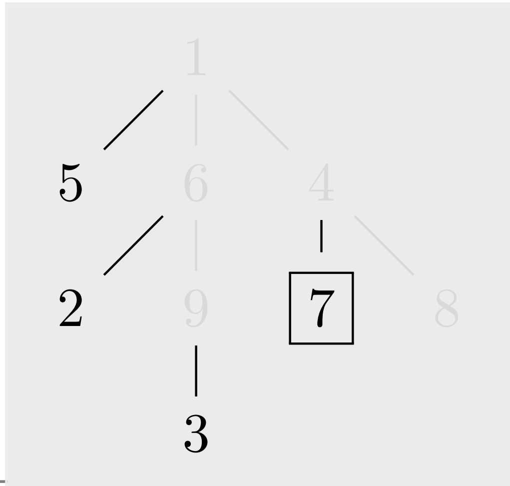


$V \setminus t = \{\boxed{5}, 7, 8, 9\}$, $\ell = 5$, $t = (1, 4, 4, 1, 6)$,
edge $\{5, 1\}$

Back to the trees

Mapping $V^{|V|-2}$ to trees

1. Given a Prüfer sequence t on V, e.g. $(6, 9, 1, 4, 4, 1, 6)$
2. Find smallest index ℓ in $V \smallsetminus t = \{2, 3, 5, 7, 8\}$, e.g. 2
3. Add $\{\ell, t_1\}$ to T, e.g. $\{2, 6\} \in E(T)$
4. Remove t_1 from t, e.g. $t = (9, 1, 4, 4, 1, 6)$
5. Remove ℓ from V, e.g. $V \smallsetminus t = \{3, 5, 7, 8\}$
6. Repeat from Step 2 until $t = \varnothing$
7. At this point $|V \smallsetminus t| = 2$ (it is an edge): add it

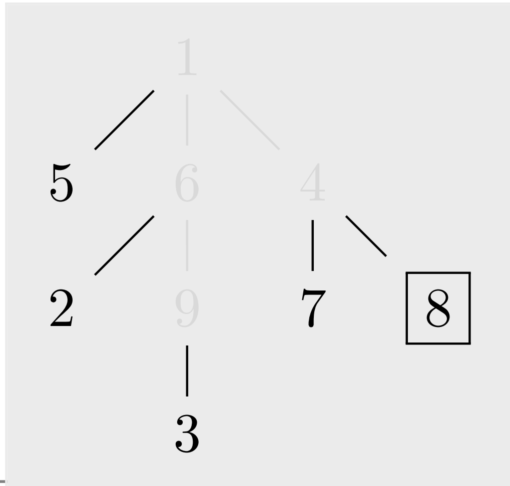


$V \smallsetminus t = \{\boxed{7}, 8, 9\}$, $\ell = 7$, $t = (4, 4, 1, 6)$,
edge $\{7, 4\}$

Back to the trees

Mapping $V^{|V|-2}$ to trees

1. Given a Prüfer sequence t on V, e.g. $(6, 9, 1, 4, 4, 1, 6)$
2. Find smallest index ℓ in $V \smallsetminus t = \{2, 3, 5, 7, 8\}$, e.g. 2
3. Add $\{\ell, t_1\}$ to T, e.g. $\{2, 6\} \in E(T)$
4. Remove t_1 from t, e.g. $t = (9, 1, 4, 4, 1, 6)$
5. Remove ℓ from V, e.g. $V \smallsetminus t = \{3, 5, 7, 8\}$
6. Repeat from Step 2 until $t = \varnothing$
7. At this point $|V \smallsetminus t| = 2$ (it is an edge): add it

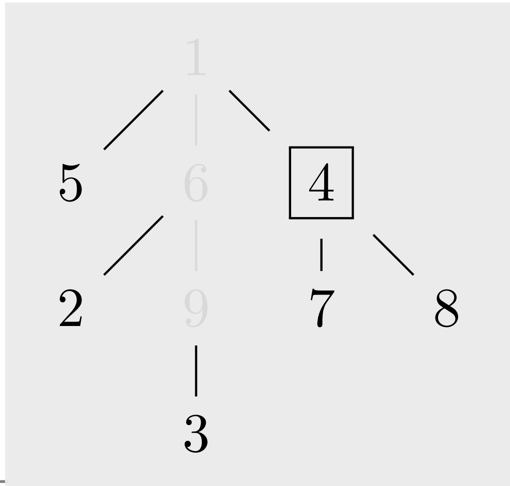


$V \smallsetminus t = \{\boxed{8}, 9\}$, $\ell = 8$, $t = (4, 1, 6)$,
edge $\{8, 4\}$

Back to the trees

Mapping $V^{|V|-2}$ to trees

1. Given a Prüfer sequence t on V, e.g. $(6,9,1,4,4,1,6)$
2. Find smallest index ℓ in $V \smallsetminus t = \{2,3,5,7,8\}$, e.g. 2
3. Add $\{\ell, t_1\}$ to T, e.g. $\{2,6\} \in E(T)$
4. Remove t_1 from t, e.g. $t = (9,1,4,4,1,6)$
5. Remove ℓ from V, e.g. $V \smallsetminus t = \{3,5,7,8\}$
6. Repeat from Step 2 until $t = \varnothing$
7. At this point $|V \smallsetminus t| = 2$ (it is an edge): add it

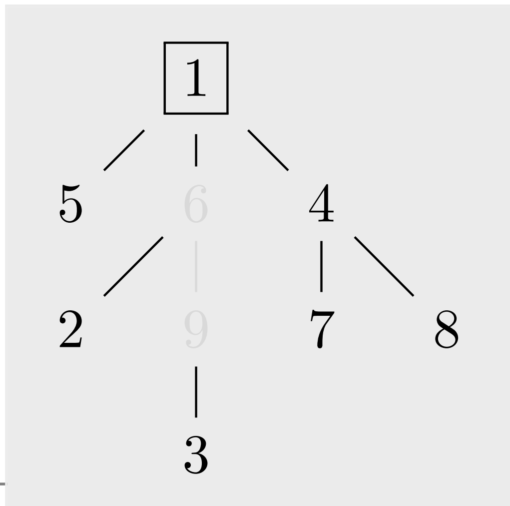


$V \smallsetminus t = \{9, \boxed{4}\}$, $\ell = 4$, $t = (1, 6)$,
edge $\{4,1\}$

Back to the trees

Mapping $V^{|V|-2}$ to trees

1. Given a Prüfer sequence t on V, e.g. $(6, 9, 1, 4, 4, 1, 6)$
2. Find smallest index ℓ in $V \smallsetminus t = \{2, 3, 5, 7, 8\}$, e.g. 2
3. Add $\{\ell, t_1\}$ to T, e.g. $\{2, 6\} \in E(T)$
4. Remove t_1 from t, e.g. $t = (9, 1, 4, 4, 1, 6)$
5. Remove ℓ from V, e.g. $V \smallsetminus t = \{3, 5, 7, 8\}$
6. Repeat from Step 2 until $t = \varnothing$
7. At this point $|V \smallsetminus t| = 2$ (it is an edge): add it

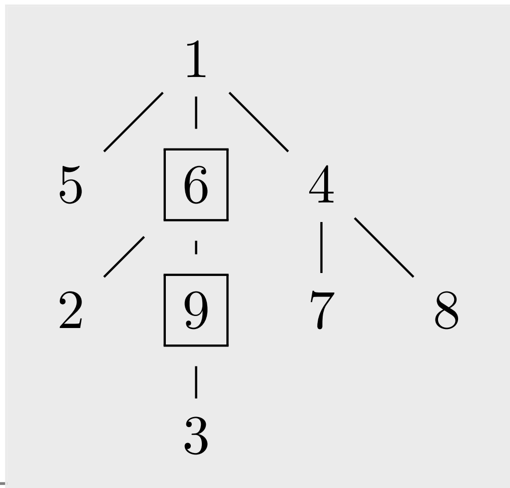


$V \smallsetminus t = \{\boxed{1}, 9\}$, $\ell = 1$, $t = (6)$,
edge $\{1, 6\}$

Back to the trees

Mapping $V^{|V|-2}$ to trees

1. Given a Prüfer sequence t on V, e.g. $(6, 9, 1, 4, 4, 1, 6)$
2. Find smallest index ℓ in $V \smallsetminus t = \{2, 3, 5, 7, 8\}$, e.g. 2
3. Add $\{\ell, t_1\}$ to T, e.g. $\{2, 6\} \in E(T)$
4. Remove t_1 from t, e.g. $t = (9, 1, 4, 4, 1, 6)$
5. Remove ℓ from V, e.g. $V \smallsetminus t = \{3, 5, 7, 8\}$
6. Repeat from Step 2 until $t = \varnothing$
7. At this point $|V \smallsetminus t| = 2$ (it is an edge): add it



$V \smallsetminus t = \{\boxed{6}, 9\}$,
edge $\{6, 9\}$

Bijection

Thm.

There is a bijection between trees on V and sequences in $V^{|V|-2}$

Proof

Essentially follows by two algorithms above

Left to prove: no cycles occur when constructing the tree from the sequence

Claim: no cycles, proceed by contradiction

Notice the mapping trees $\to$ sequences always deletes leaf nodes

By definition, a cycle must have ≥ 3 nodes, and none of these can be a leaf

So the resulting sequence has at most $|V| - 3$ nodes, contradiction (why?)

Thm.

[Cayley 1889] Let $|V| = n$. There are n^{n-2} labelled trees on V

Proof

By previous theorem, the number of labelled trees is the same as the number of sequences in $V^{|V|-2}$ (this proof is by Prüfer, 1918)

Grammars and languages

A remark

- Most students *(and not just students!)* find arrays, lists, maps, queues and stacks “easier” than trees

A remark

- Most students *(and not just students!)* find arrays, lists, maps, queues and stacks “easier” than trees
- **Thesis 1**: the graphical representation

  People are used to read sequence-like rather than tree-like text

A remark

- Most students *(and not just students!)* find arrays, lists, maps, queues and stacks “easier” than trees
- **Thesis 1**: the graphical representation

  People are used to read sequence-like rather than tree-like text
- **Thesis 2**: iterative *vs.* recursive
  - *Sequences are models of* **iteration** *and trees models of* **recursion**
  - *Most people think iteratively rather than recursively (?)*

A remark

- Most students *(and not just students!)* find arrays, lists, maps, queues and stacks “easier” than trees
- **Thesis 1**: the graphical representation
  People are used to read sequence-like rather than tree-like text
- **Thesis 2**: iterative *vs.* recursive
  - *Sequences are models of* **iteration** *and trees models of* **recursion**
  - *Most people think iteratively rather than recursively (?)*
- **Thesis 3**: trees require decisions
  - *Every node has* ≤ 1 *next node in a sequence*
    tree nodes might have more than one subnodes
  - $\Rightarrow$ *Scanning a sequence: no decisions to take*
    $\Rightarrow$ *Exploring a tree: which subnode to process next?*

Languages and grammars

- Remember *nouns, adjectives, transitive verbs* from school?
- Sentence analysis: identify and name grammatical components
- Analyze components recursively:

$$
\begin{array}{rcl}
\underline{\textit{sentence}} & \longrightarrow & \textit{names verb} \\
\textit{names} & \longrightarrow & \textit{name names} \\
\textit{name} & \longrightarrow & \textit{noun} \\
 & || & \textit{article noun} \\
 & || & \textit{adjectives noun} \\
 & || & \textit{article adjectives noun} \\
\textit{adjectives} & \longrightarrow & \textit{adjective adjectives} \\
\textit{verb} & \longrightarrow & \ldots
\end{array}
$$

Parse trees

The soft, furry cat purrs

<u>*sentence*</u>	$\longrightarrow$	*names verb*
names	$\longrightarrow$	*name names*
name	$\longrightarrow$	*noun*
	$\|\|$	*article noun*
	$\|\|$	*adjectives noun*
	$\|\|$	*article adjectives noun*
adjectives	$\longrightarrow$	*adjective adjectives*
verb	$\longrightarrow$	...

Parse trees

sentence	$\longrightarrow$	*names verb*
names	$\longrightarrow$	*name names*
name	$\longrightarrow$	*noun*
	$\vert\vert$	*article noun*
	$\vert\vert$	*adjectives noun*
	$\vert\vert$	*article adjectives noun*
adjectives	$\longrightarrow$	*adjective adjectives*
verb	$\longrightarrow$	...

The soft, furry cat purrs

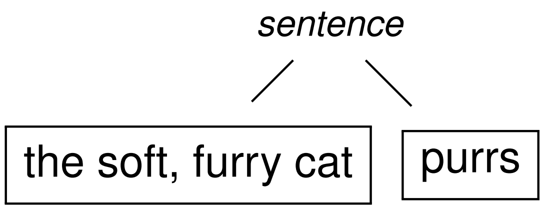
---

the soft, furry cat purrs

Parse trees

sentence	$\longrightarrow$	*names verb*
names	$\longrightarrow$	*name names*
name	$\longrightarrow$	*noun*
	$\|\|$	*article noun*
	$\|\|$	*adjectives noun*
	$\|\|$	*article adjectives noun*
adjectives	$\longrightarrow$	*adjective adjectives*
verb	$\longrightarrow$	...

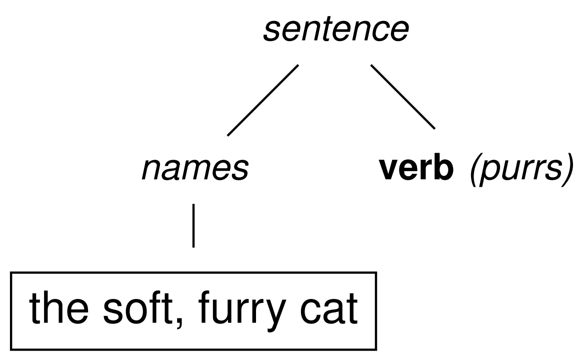
The soft, furry cat purrs



Parse trees

sentence	$\longrightarrow$	*names verb*
names	$\longrightarrow$	*name names*
name	$\longrightarrow$	*noun*
	$\|\|$	*article noun*
	$\|\|$	*adjectives noun*
	$\|\|$	*article adjectives noun*
adjectives	$\longrightarrow$	*adjective adjectives*
verb	$\longrightarrow$	...

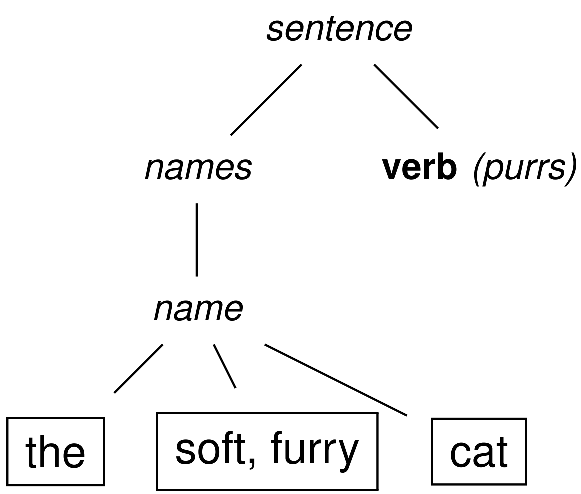
The soft, furry cat purrs



Parse trees

The soft, furry cat purrs

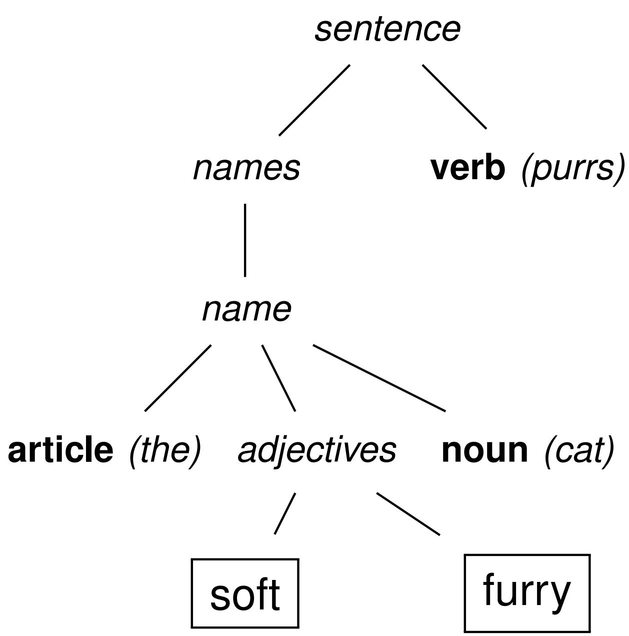
$$\begin{array}{rcl}
\underline{\textit{sentence}} & \longrightarrow & \textit{names verb} \\
\textit{names} & \longrightarrow & \textit{name names} \\
\textit{name} & \longrightarrow & \textit{noun} \\
 & || & \textit{article noun} \\
 & || & \textit{adjectives noun} \\
 & || & \textit{article adjectives noun} \\
\textit{adjectives} & \longrightarrow & \textit{adjective adjectives} \\
\textit{verb} & \longrightarrow & \ldots
\end{array}$$



Parse trees

The soft, furry cat purrs

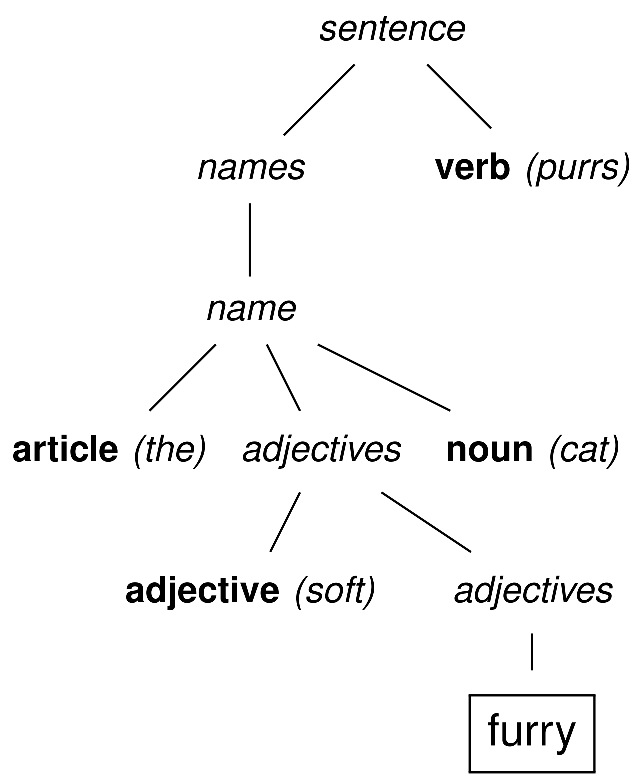
sentence	$\longrightarrow$	*names verb*
names	$\longrightarrow$	*name names*
name	$\longrightarrow$	*noun*
	$\|\|$	*article noun*
	$\|\|$	*adjectives noun*
	$\|\|$	*article adjectives noun*
adjectives	$\longrightarrow$	*adjective adjectives*
verb	$\longrightarrow$	$\ldots$



Parse trees

The soft, furry cat purrs

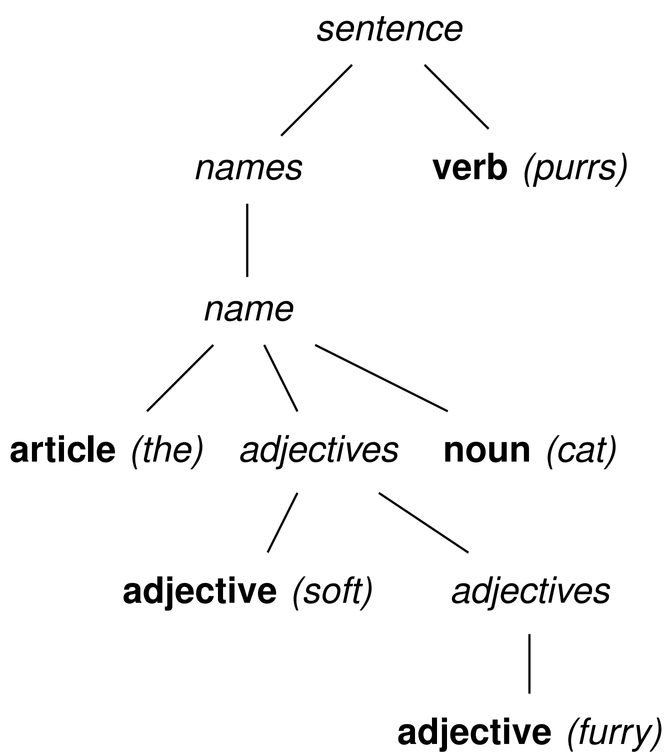
sentence	$\longrightarrow$	*names verb*
names	$\longrightarrow$	*name names*
name	$\longrightarrow$	*noun*
	$\vert\vert$	*article noun*
	$\vert\vert$	*adjectives noun*
	$\vert\vert$	*article adjectives noun*
adjectives	$\longrightarrow$	*adjective adjectives*
verb	$\longrightarrow$	...



Parse trees

sentence	$\longrightarrow$	*names verb*
names	$\longrightarrow$	*name names*
name	$\longrightarrow$	*noun*
	$\|\|$	*article noun*
	$\|\|$	*adjectives noun*
	$\|\|$	*article adjectives noun*
adjectives	$\longrightarrow$	*adjective adjectives*
verb	$\longrightarrow$	...

The soft, furry cat purrs



Formal and natural languages

- More than one parse tree to a given sentence $\Rightarrow$ **ambiguous grammar**
- Different parse trees lead to different meanings $\Rightarrow$ **ambiguous language**
- **Formal languages**: non-ambiguous
  (e.g. formal logic, C/C++, Java,. . .)
- **Natural languages**: ambiguous
  (e.g. common mathematical language, English, French,. . .)
- Richard Montague (1930-1971): grammar based mechanisms to disambiguate subsets of English

Depth-First Search

Tree exploration

- Breadth-First Search (BFS — seen in Lecture 2 on graphs)
  find the way out of a maze in the smallest number of steps
- Depth-First Search (DFS — on trees)
- DFS: recursive call to `dfs`(*node* v):

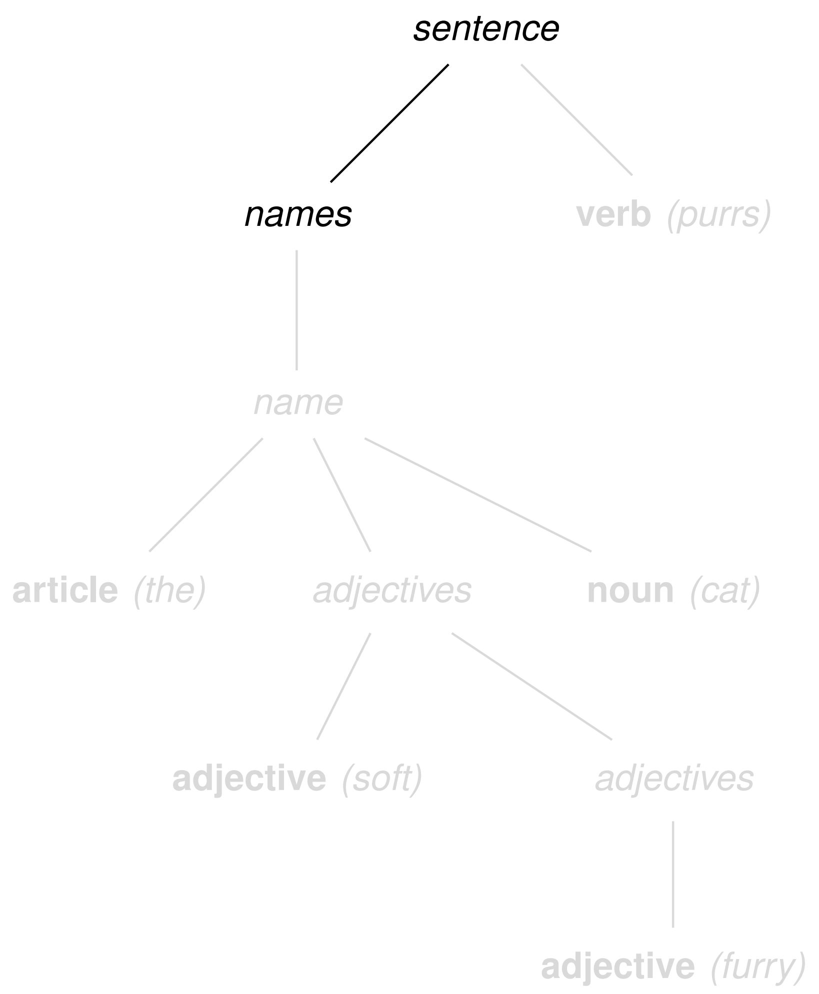
  optionally perform an action on v (*prefix*);
  for all subnodes u of v **do**
    `dfs`(u);
  end for
  optionally perform an action on v (*postfix*);

- DFS on trees is: `dfs`(*root*)

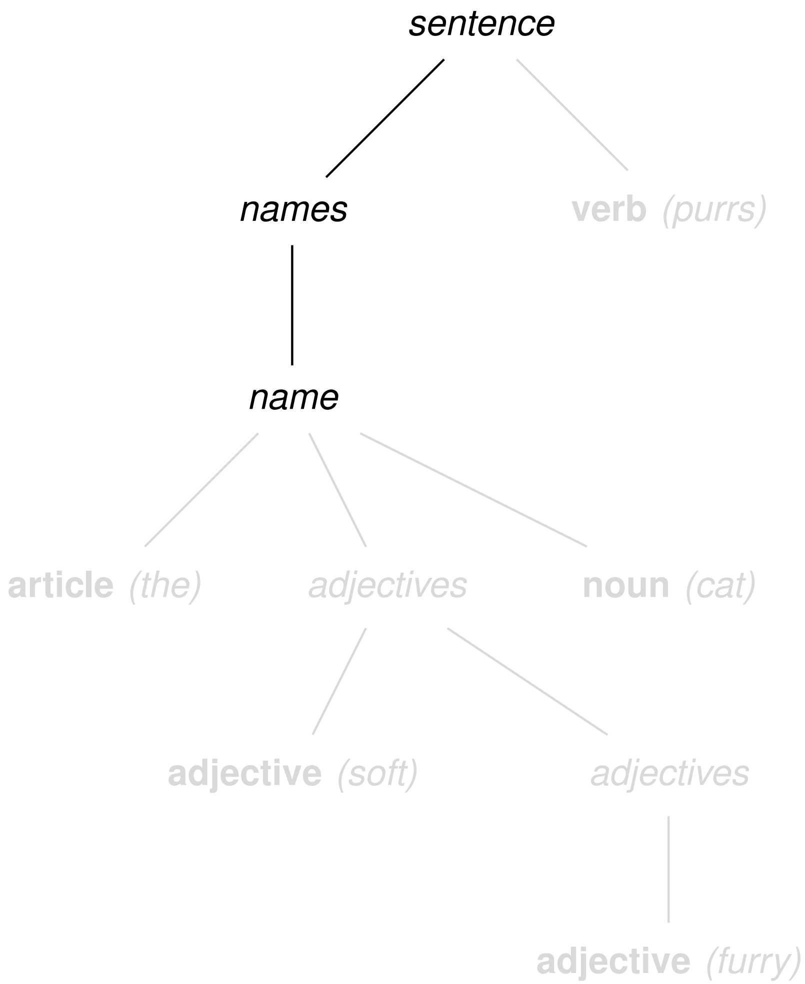
DFS: exploring a parse tree

DFS: exploring a parse tree

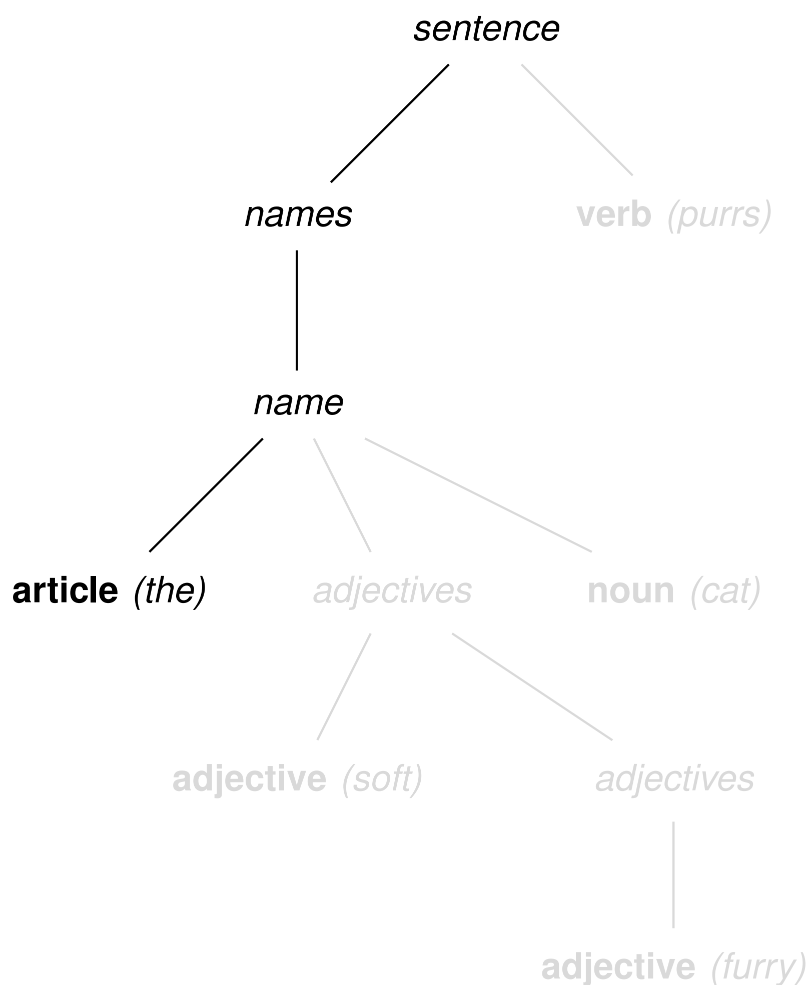
DFS: exploring a parse tree



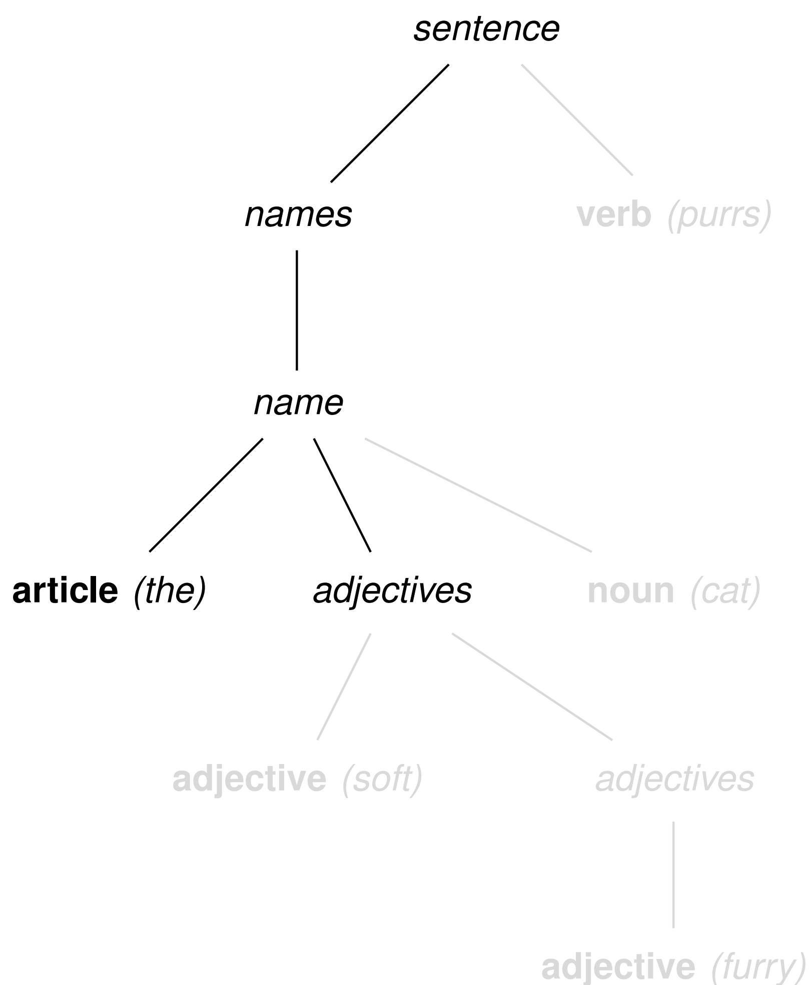
DFS: exploring a parse tree



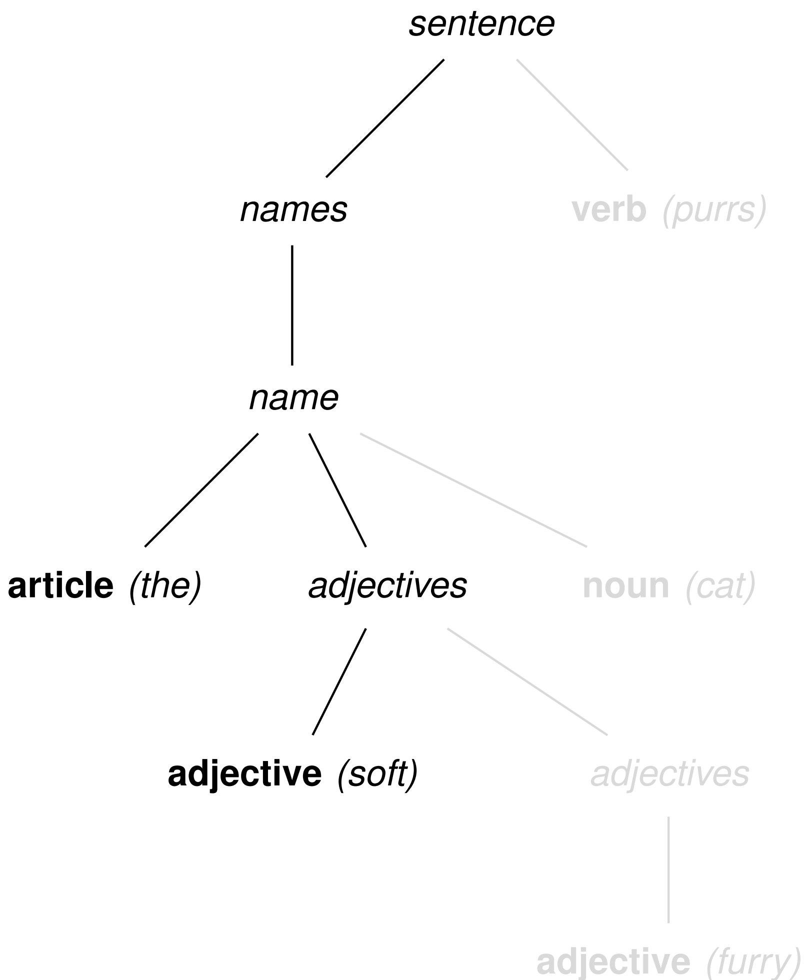
DFS: exploring a parse tree



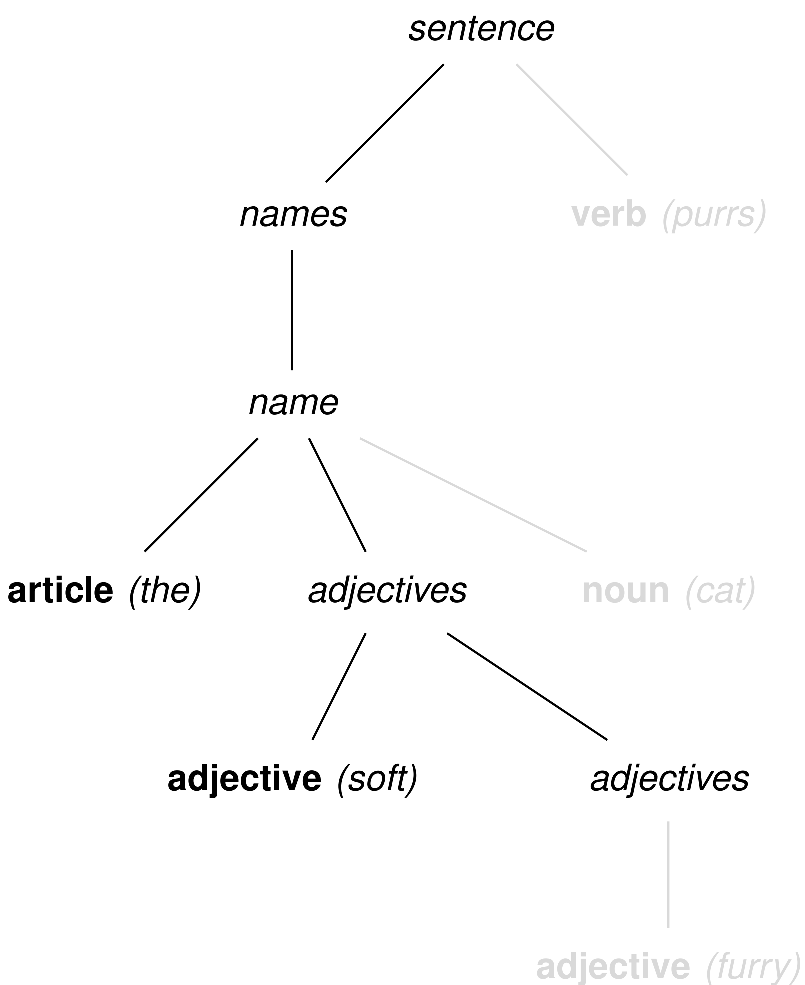
DFS: exploring a parse tree



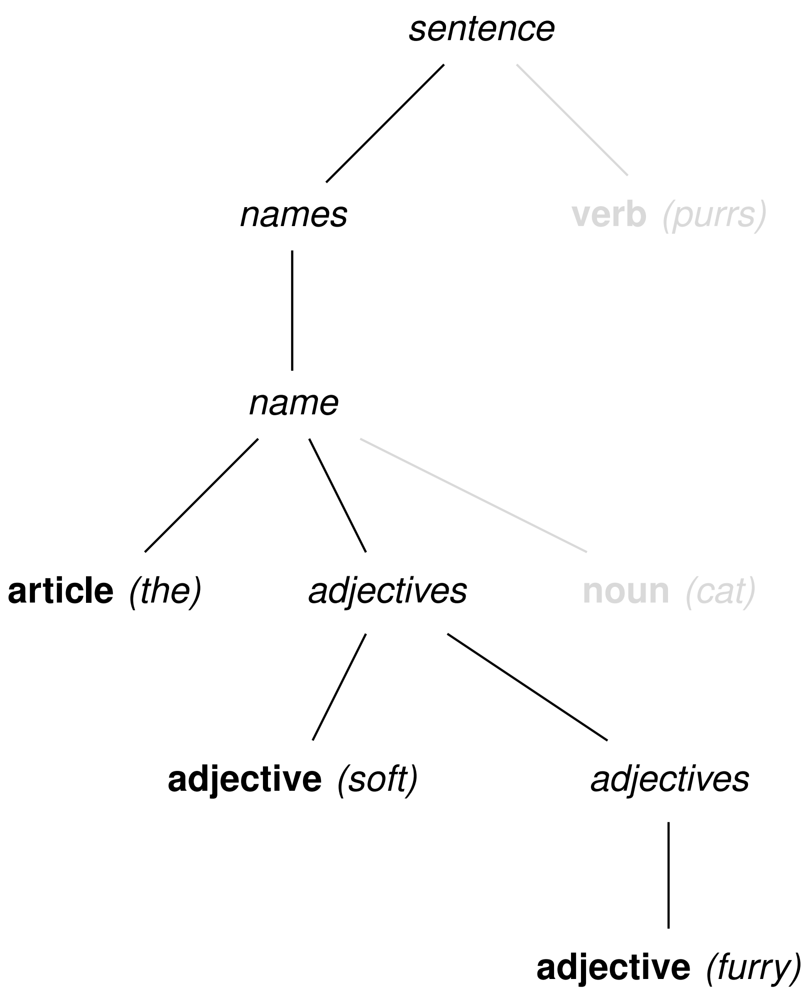
DFS: exploring a parse tree



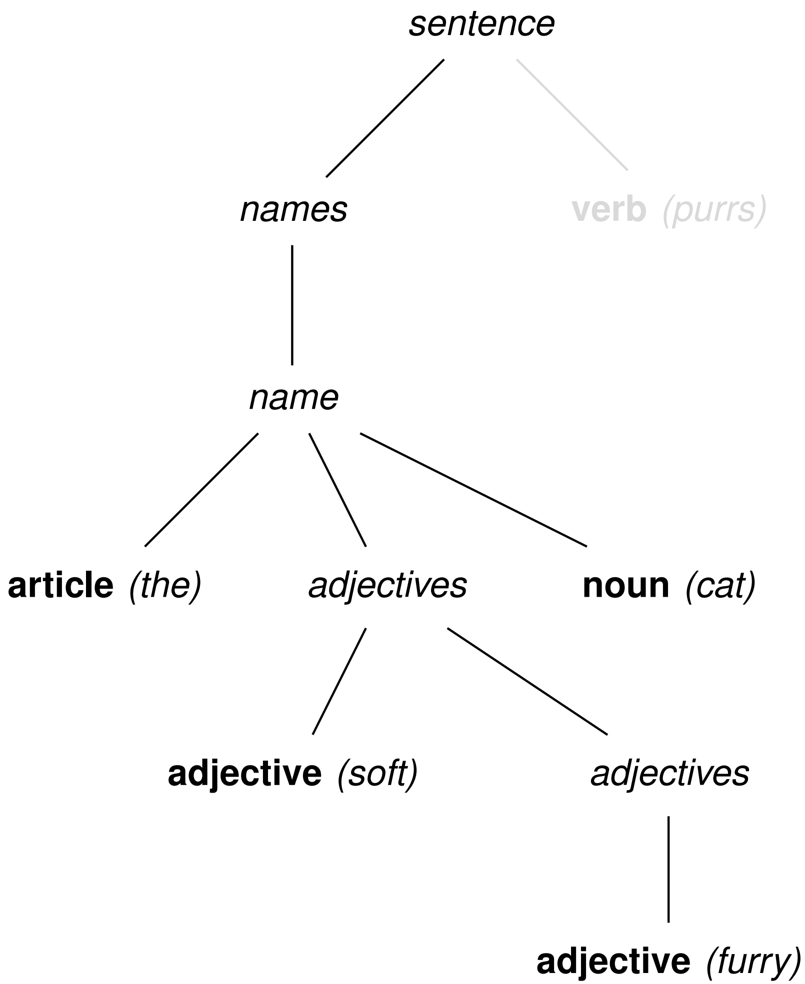
DFS: exploring a parse tree



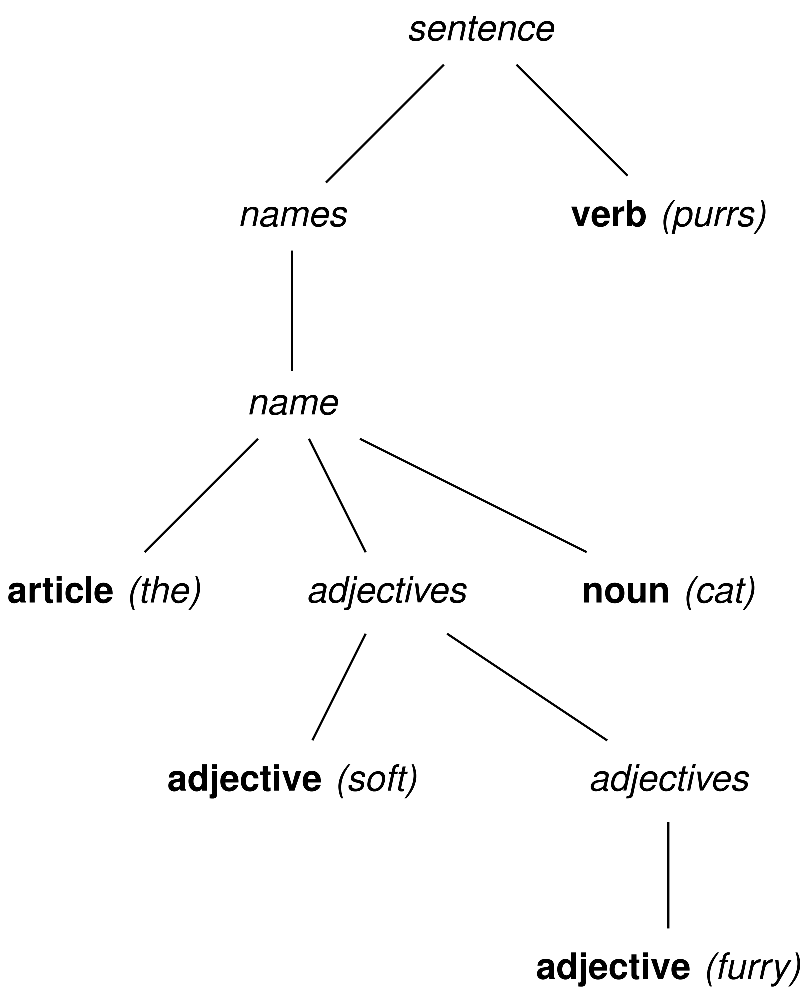
DFS: exploring a parse tree



DFS: exploring a parse tree



DFS: exploring a parse tree



Digraph scanning

- DFS on trees: explore nodes from root, visit each node once
- DFS on digraphs: record visited nodes, don't visit them again

Require: $G = (V, A)$, $s \in V$, $R = \{s\}$, $Q = \{s\}$

```
while Q ≠ ∅ do
   choose v ∈ Q  // v is scanned
   Q ← Q ∖ {v}
   for w ∈ N⁺(v) ∖ R do
      R ← R ∪ {w}
      Q ← Q ∪ {w}
   end for
end while
```

The algorithm is correct

Thm.

If there is an oriented path P from s to $z \in V$, then DIGRAPH SCANNING scans z

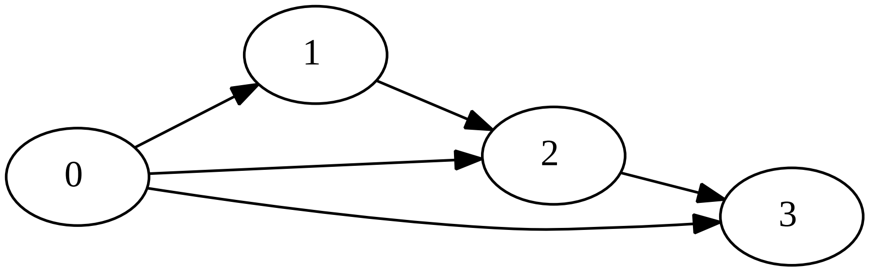
Proof

- Suppose not, then $\exists (x, y) \in P$ with $x \in R$ and $y \notin R$ (for otherwise, by induction on the path length, $z \in R$ by Step 5 and hence in Q by Step 6)
- By Step 6 x was added to Q
- The algorithm does not stop before eliminating x from Q in Step 3 at some iteration
- When this happens, $N^+(x) \subseteq R$ by Steps 4-5
- Hence $y \notin N^+(x)$, which implies $(x, y) \notin P$, which yields a contradiction

Storing a digraph

- Seen in Lecture 1: use the *jagged array* representation (*also called* **adjacency list**)

$$N^+(0) = (1, 2, 3)$$
$$N^+(1) = (2)$$
$$N^+(2) = (3)$$



- Seen in Lecture 2: use the *list of arcs* representation

$$L = ((0, 1), (0, 2), (0, 3), (1, 2), (2, 3))$$

Different efficiency on different algorithms

The algorithm takes $O(n+m)$

Thm.

If the digraph is encoded as adjacency lists, DIGRAPH SCANNING takes CPU time proportional $O(n+m)$ in the worst case

Proof

- **Each node is considered only once:**
  - Whenever a node x is eliminated from Q, it was previously inserted by Step 6, which means that it was also added to R by Step 5
  - By Step 4, x is never re-added to Q
- **Each arc (x, y) is considered only once:**
  - When $x = v$ in Step 2 then $y \in N^+(x)$, so either $y = w$ in Step 4 or it must be verified that $y \in R$
  - In both cases, the relation (x, y) was considered once

The choice of $v \in Q$

- In Step 2, the choice of $v \in Q$ determines the order in which the nodes are scanned
- Can alter this using different data structures for implementing the set Q
- Two data structures are commonly used:
  1. Queues *(lecture 2)*

     BREADTH-FIRST SEARCH: this corresponds to the order being First-In, First-Out (FIFO)

  2. Stacks *(lecture 6)*

     DEPTH-FIRST SEARCH (DFS): this corresponds to the order being Last-In, First-Out (LIFO)

Stacks: a first peek

- Linear data structure
- Accessible from only one end (top)
- Operations:
  - push an item on the top
  - pop an item from the top
  - test whether stack is empty
- Implement using arrays or lists in $O(1)$

DFS on a digraph $\equiv$ GRAPH SCANNING with a stack

End of Lecture 4