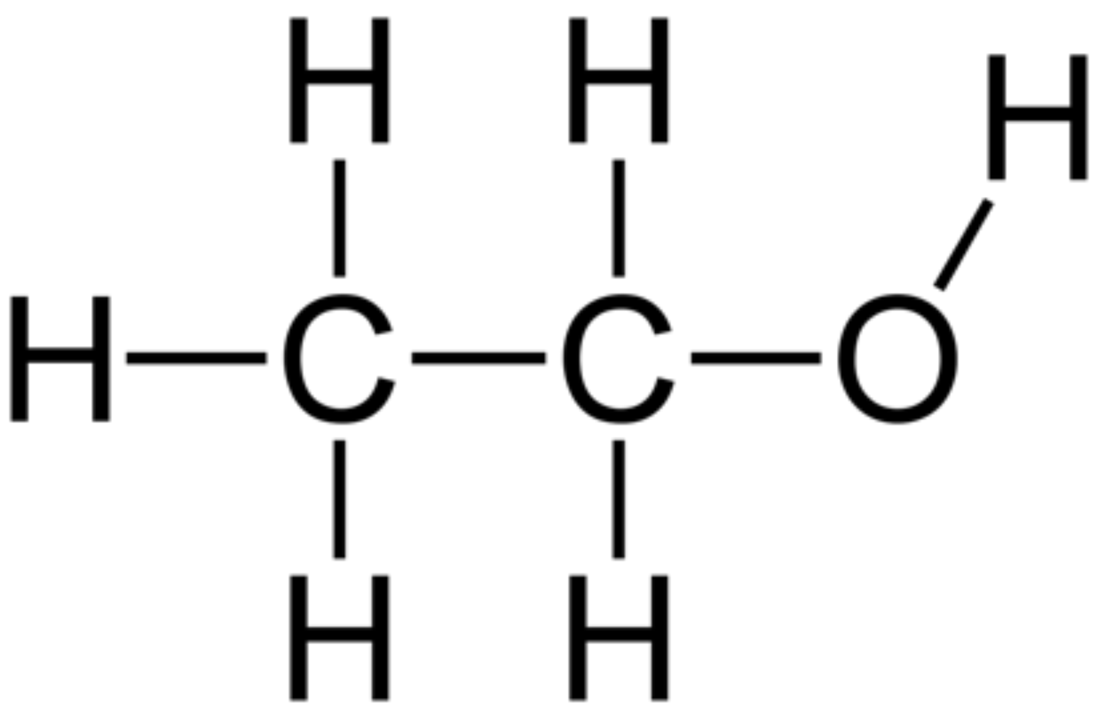
Topic II.1: Frequent Subgraph Mining

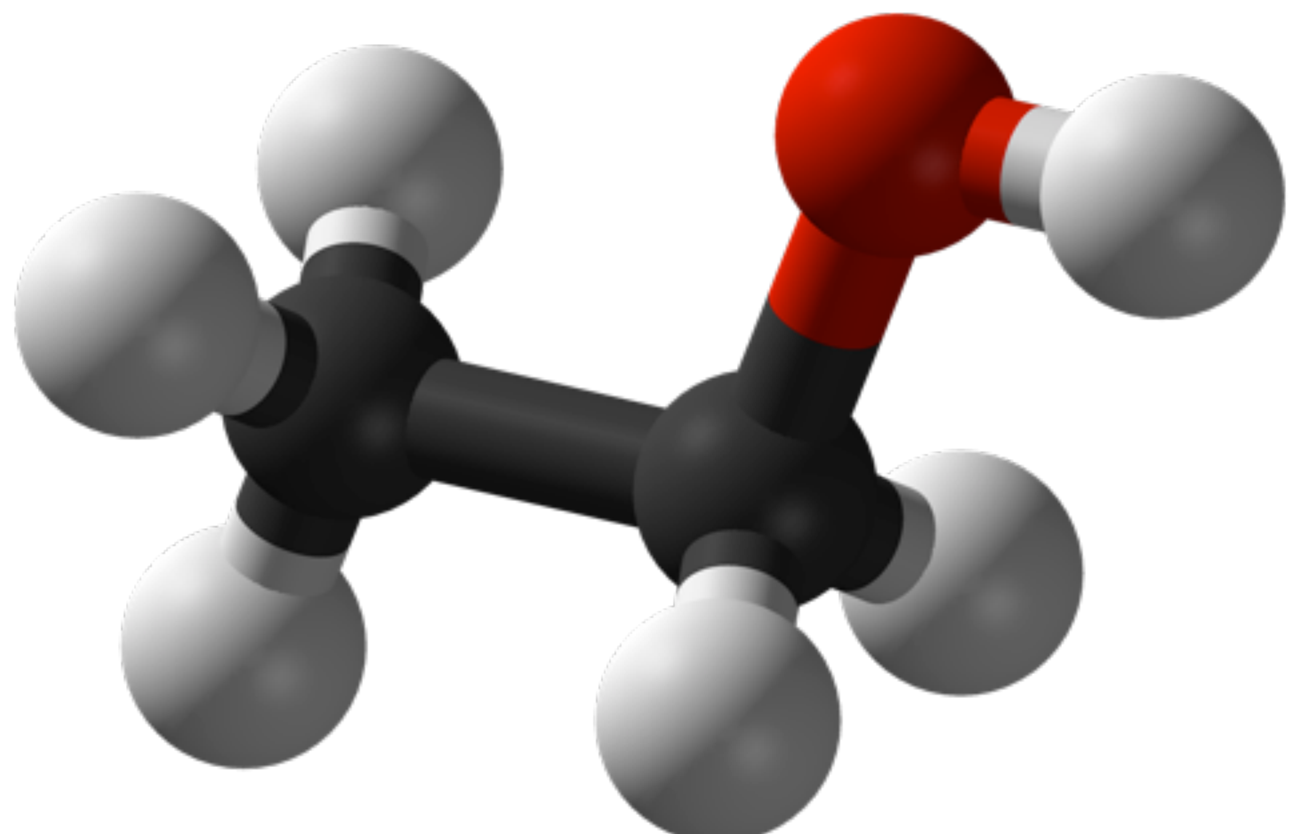
Discrete Topics in Data Mining
Universität des Saarlandes, Saarbrücken
Winter Semester 2012/13

TII.1: Frequent Subgraph Mining

Definitions and Problems

- The data is a set of graphs $D = \{G_1, G_2, \ldots, G_n\}$
  - Directed or undirected
- The graphs G_i are *labelled*
  - Each vertex v has a label $L(v)$
  - Each edge $e = (u, v)$ has a label $L(u, v)$
- Data can be e.g. molecule structures





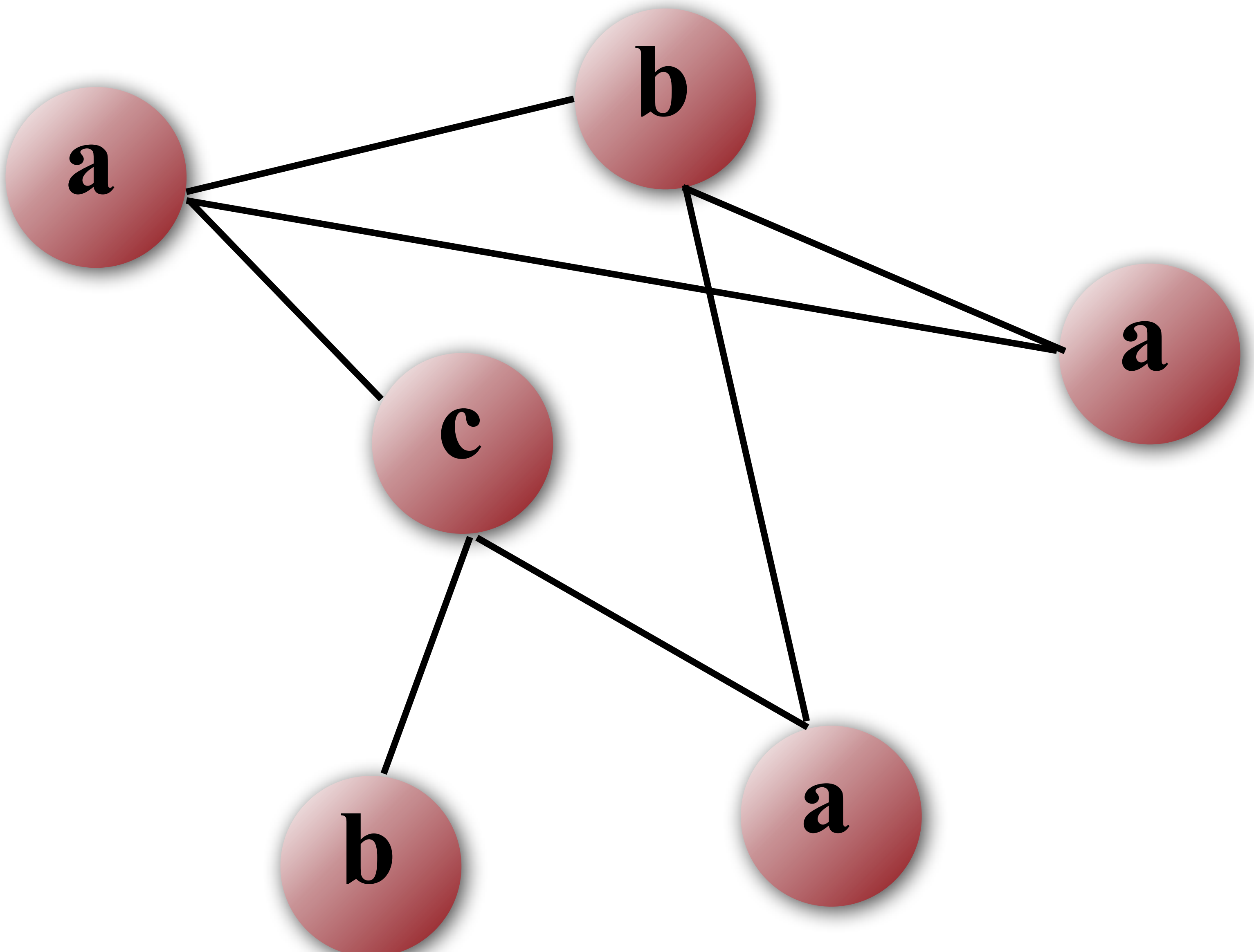
Graph Isomorphism

- Graphs $G = (V, E)$ and $G' = (V', E')$ are **isomorphic** if there exists a bijective function $\varphi: V \to V'$ such that
  - $(u, v) \in E$ if and only if $(\varphi(u), \varphi(v)) \in E'$
  - $L(v) = L(\varphi(v))$ for all $v \in V$
  - $L(u, v) = L(\varphi(u), \varphi(v))$ for all $(u, v) \in E$
- Graph G' is *subgraph isomorphic* to G if there exists a subgraph of G which is isomorphic to G'
- No polynomial-time algorithm is known for determining if G and G' are isomorphic
- Determining if G' is subgraph isomorphic to G is NP-hard

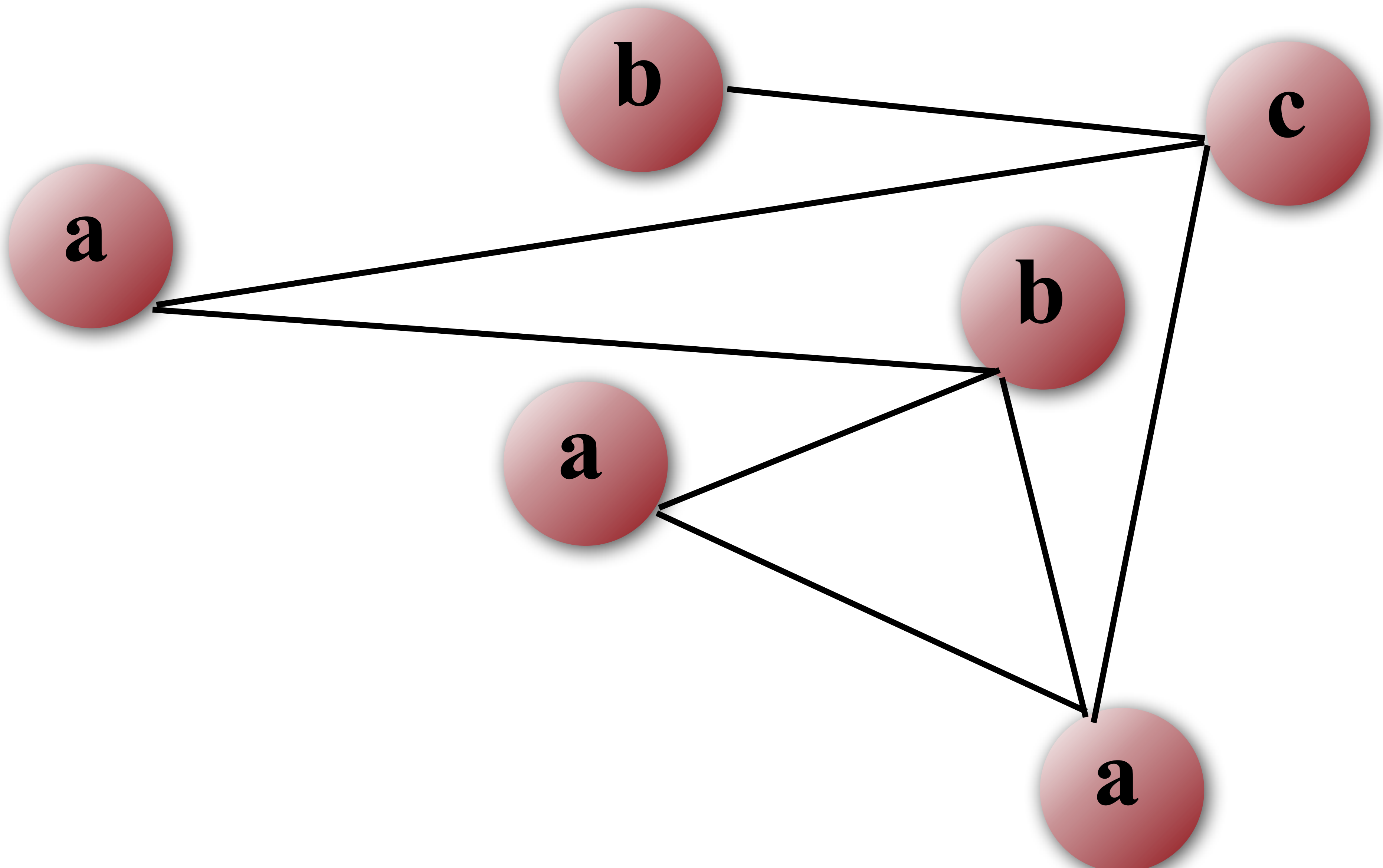
Equivalence and Canonical Graphs

- Isomorphism defines an equivalence class
  - id: $V \to V$, $\mathrm{id}(v) = v$ shows G is isomorphic to itself
  - If G is isomorphic to G' via φ, then G' is isomorphic to G via φ^{-1}
  - If G is isomorphic to H via φ and H to I via χ, then G is isomorphic to I via $\varphi \circ \chi$
- A **canonization** of a graph G, $canon(G)$ produces another graph C such that if H is a graph that is isomorphic to G, $canon(G) = canon(H)$
  - Two graphs are isomorphic if and only if their canonical versions are the same

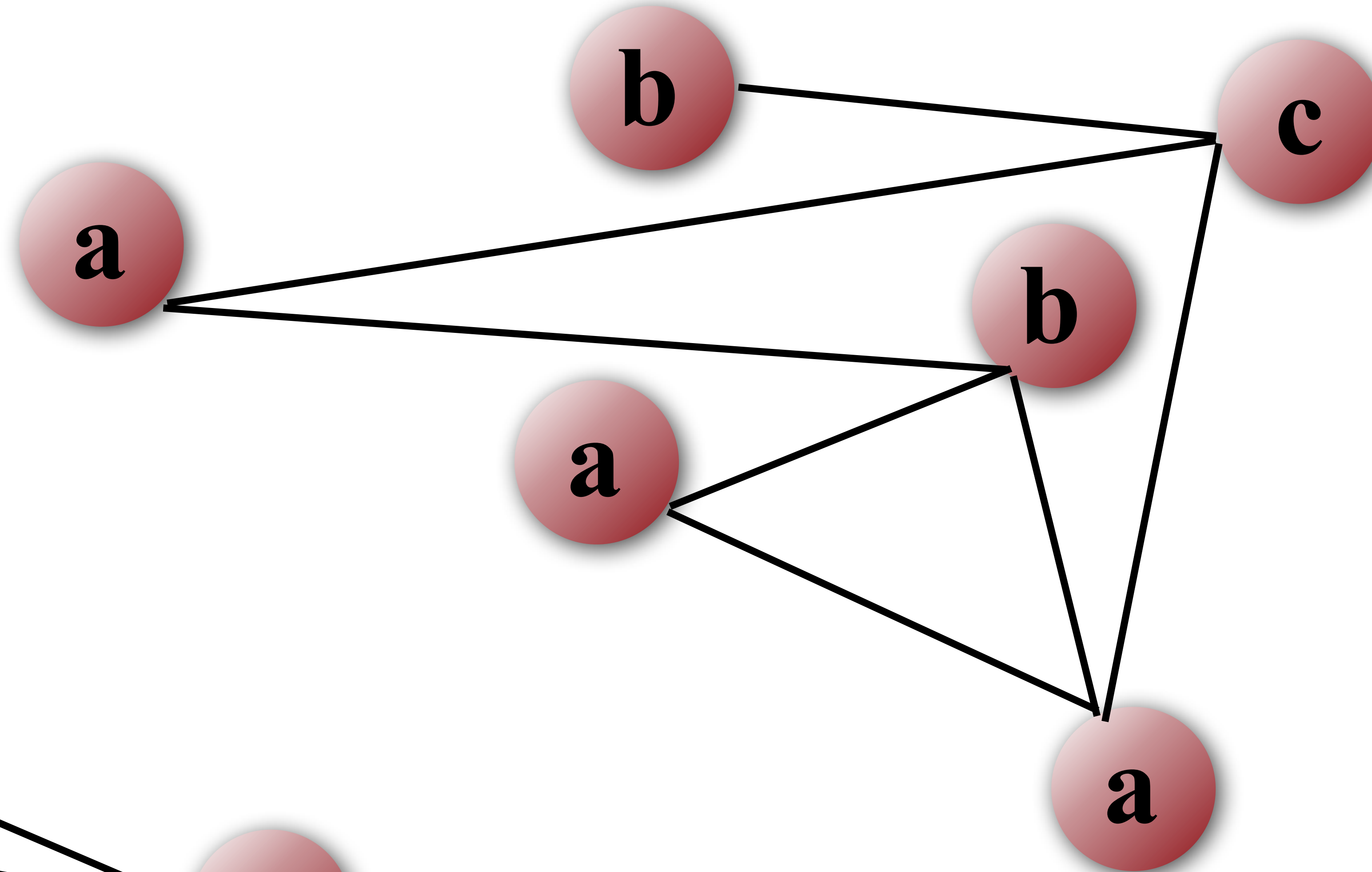
An Example of Isomorphic Graphs

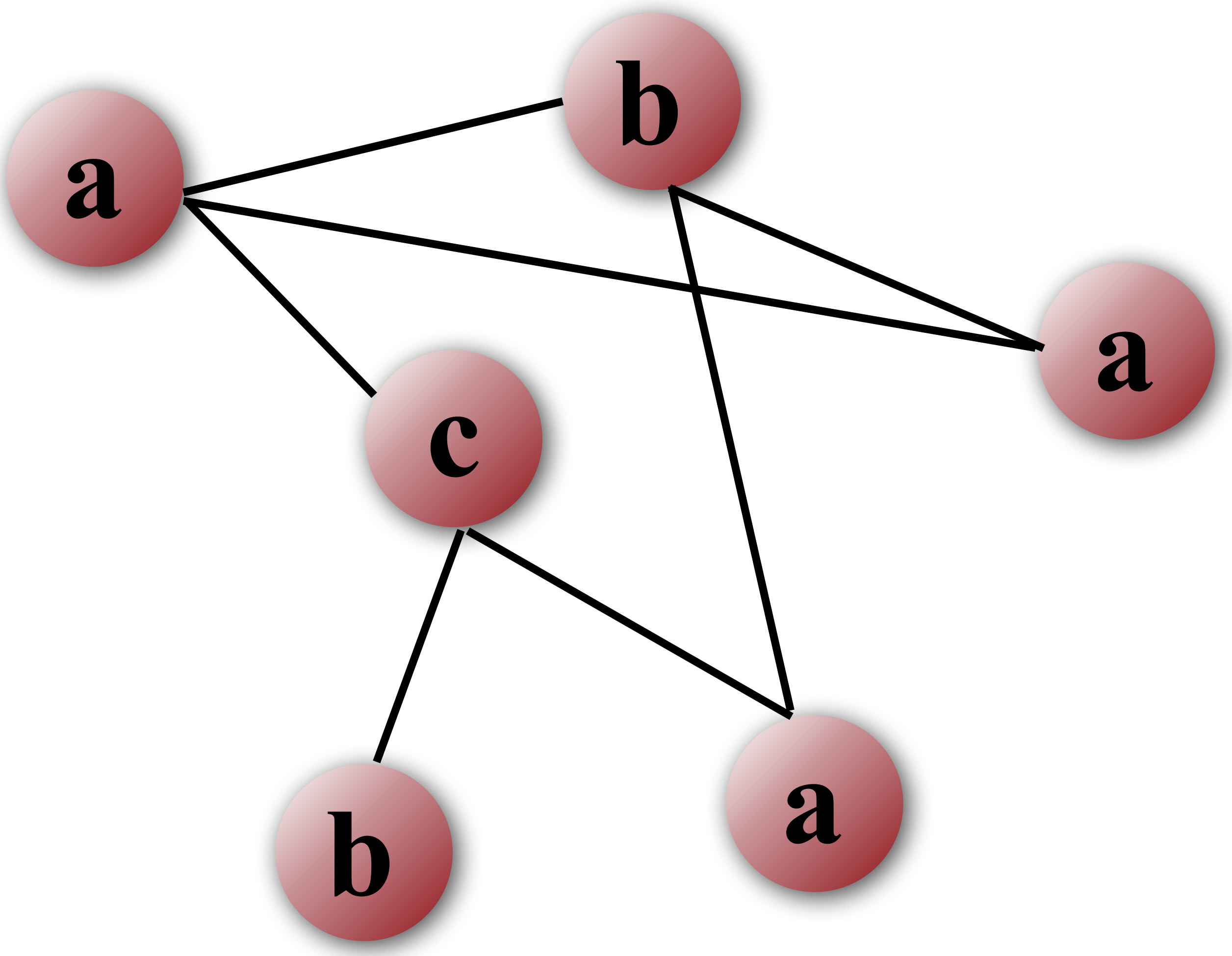


An Example of Isomorphic Graphs

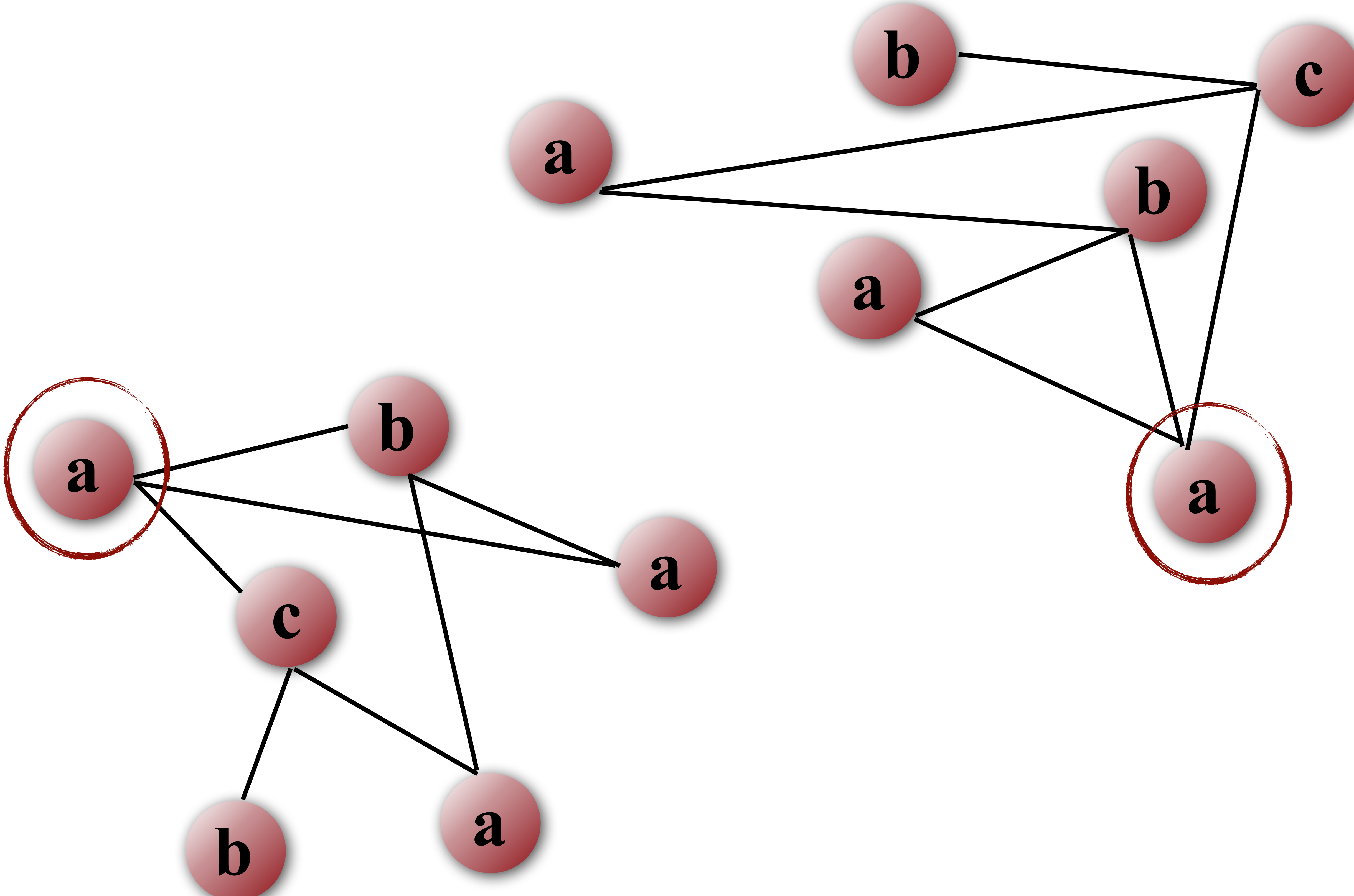


An Example of Isomorphic Graphs

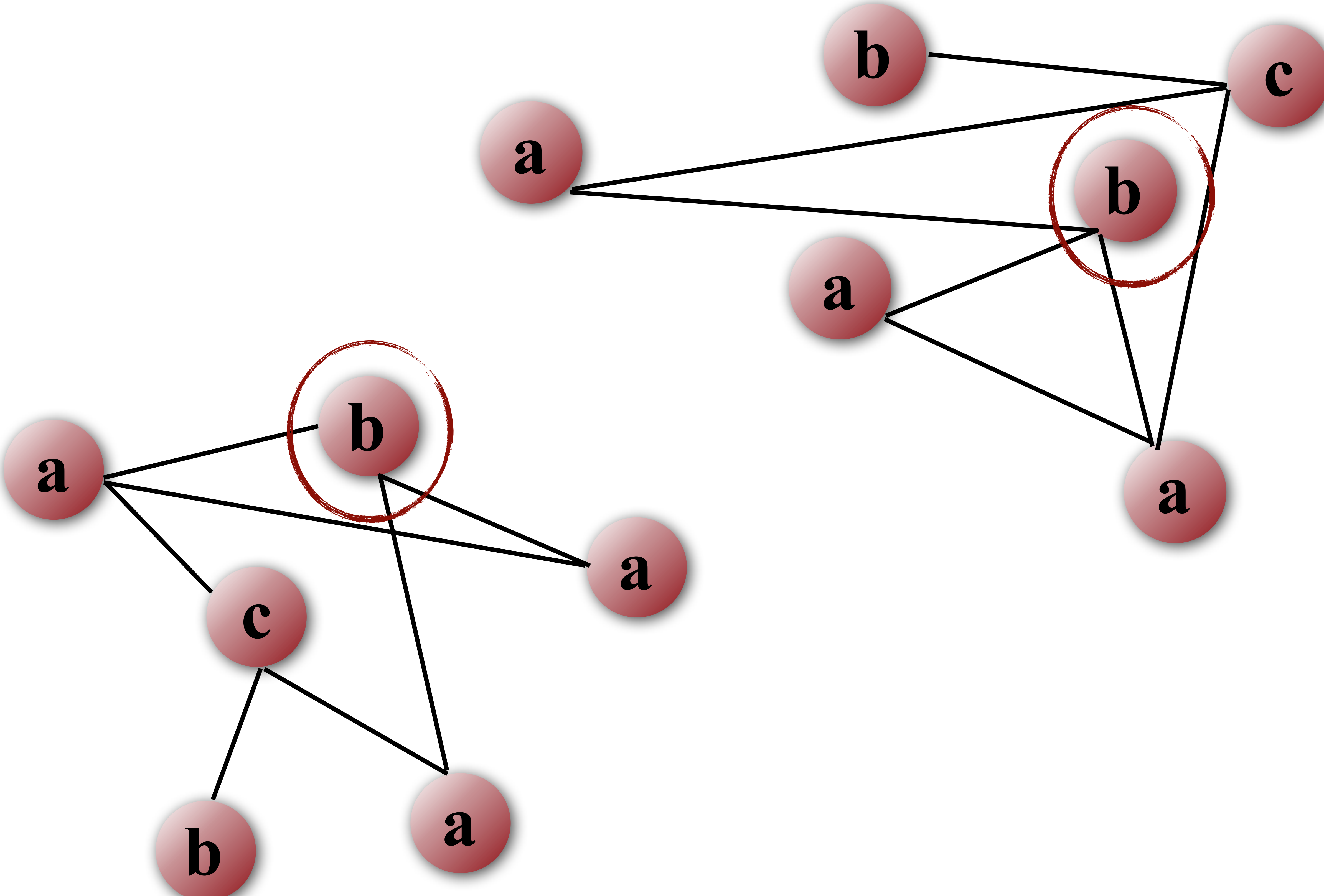




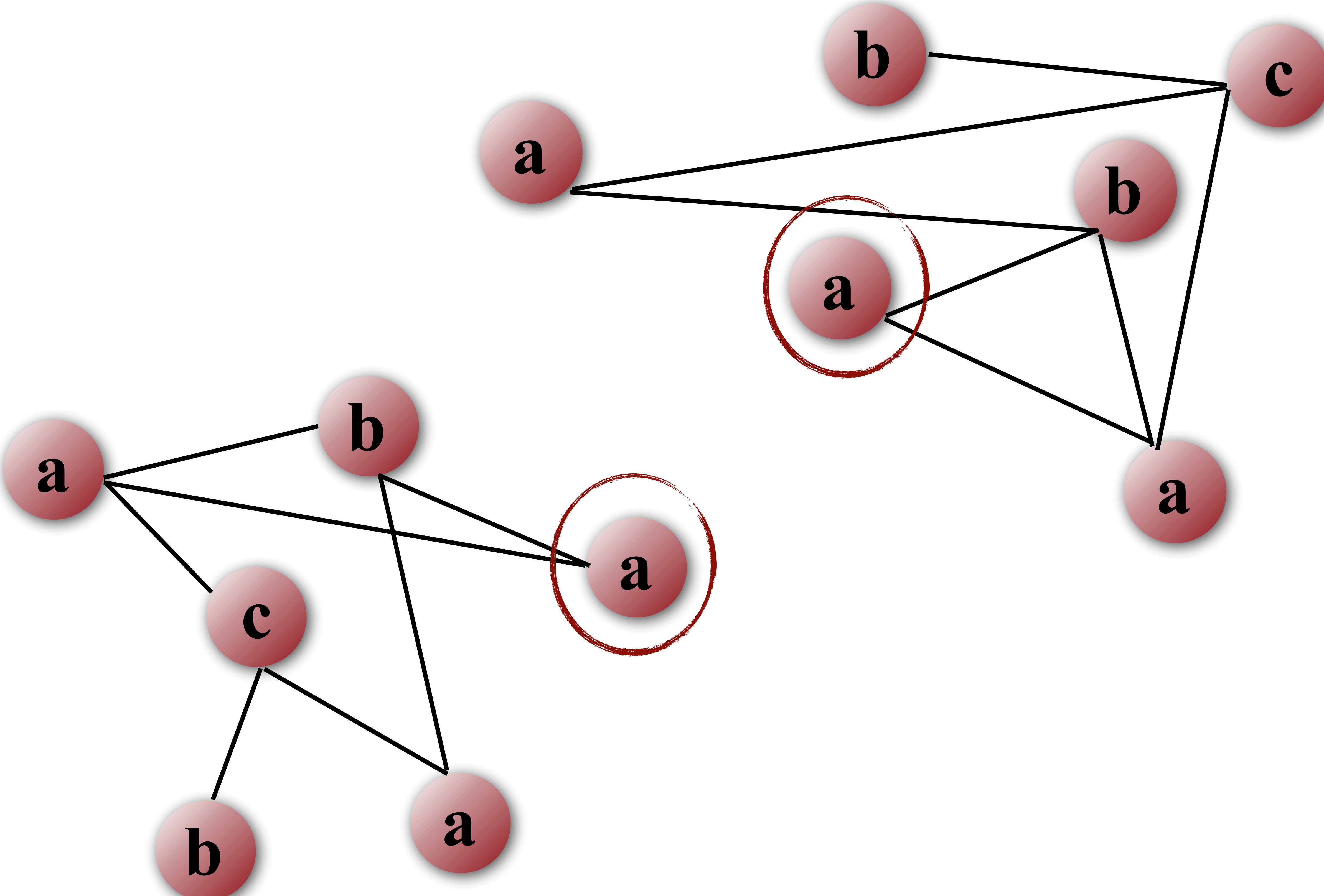
An Example of Isomorphic Graphs



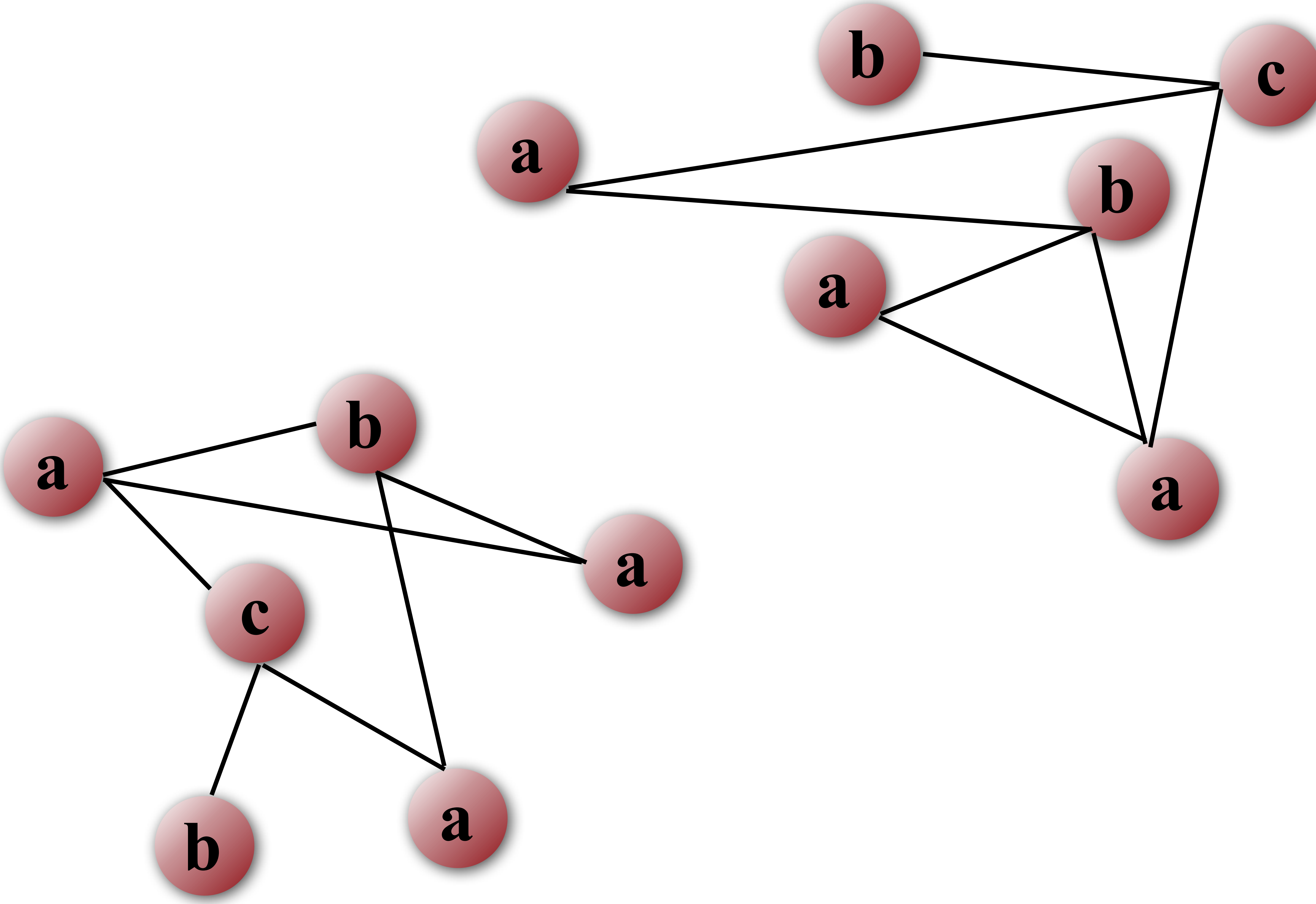
An Example of Isomorphic Graphs



An Example of Isomorphic Graphs



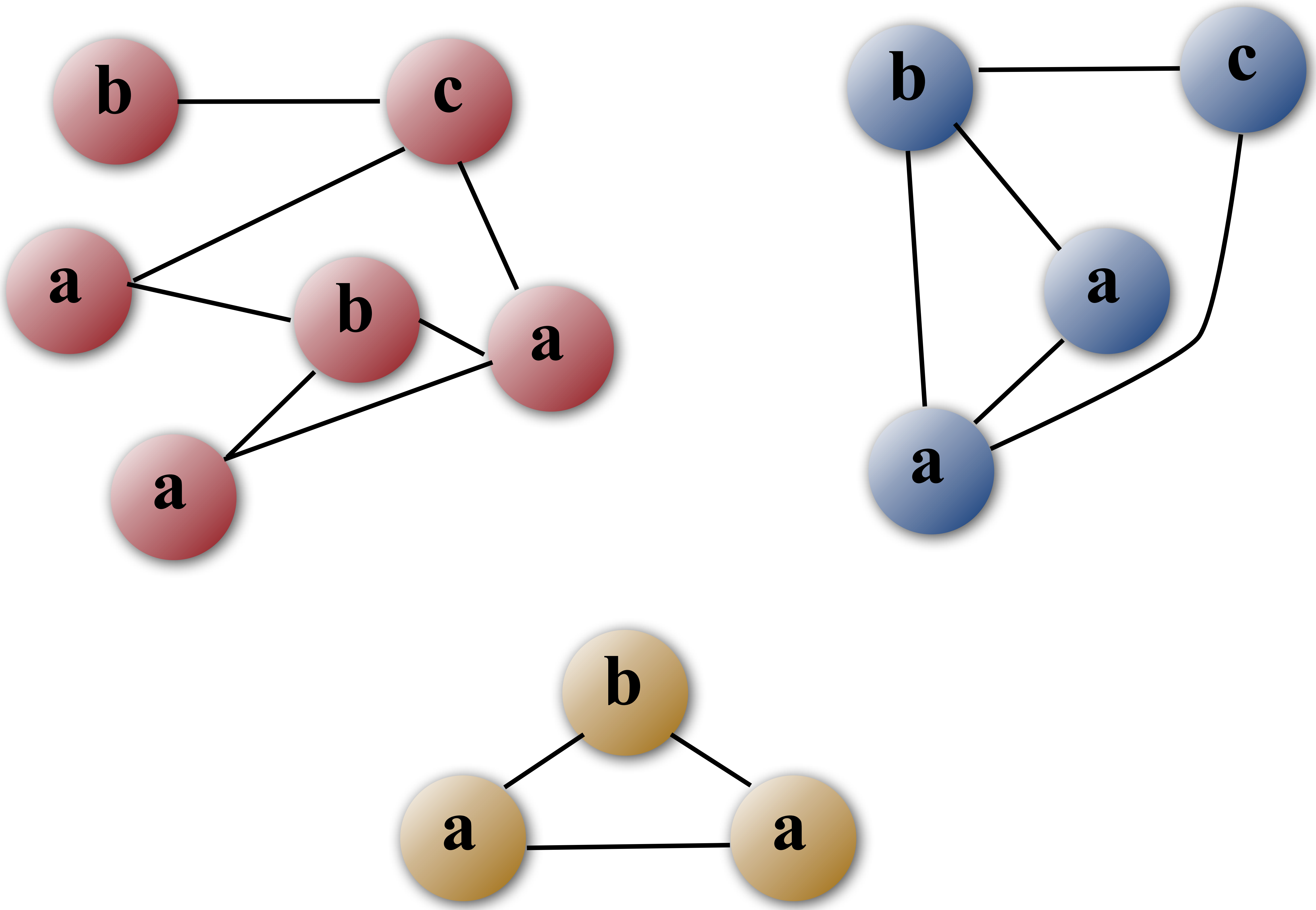
An Example of Isomorphic Graphs



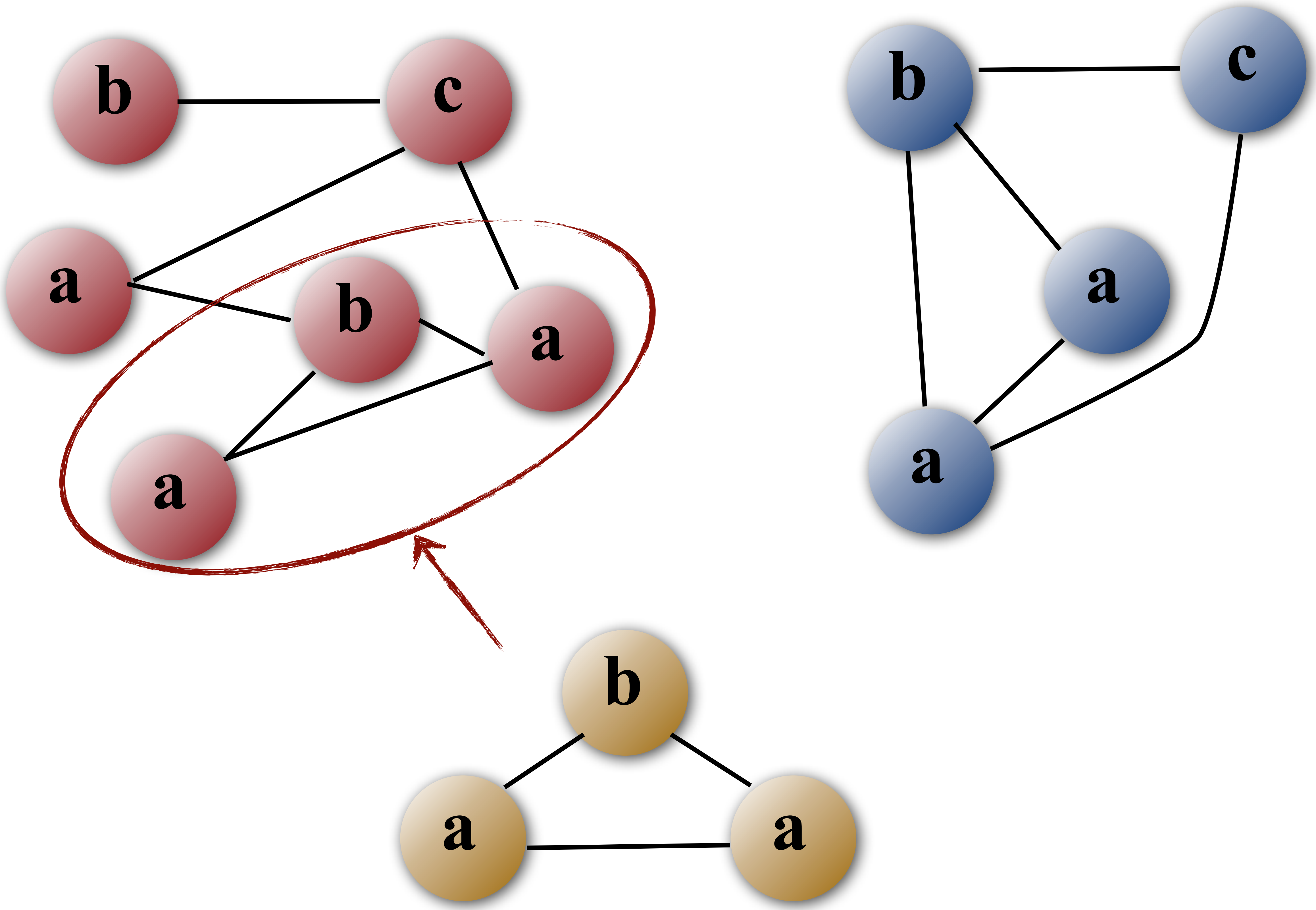
Frequent Subgraph Mining

- Given a set D of n graphs and a minimum support parameter *minsup*, find all connected graphs that are subgraph isomorphic to at least *minsup* graphs in D
  - Enormously complex problem
  - For graphs that have m vertices there are
    - $2^{O(m^2)}$ subgraphs (not all are connected)
  - If we have s labels for vertices and edges we have
    - $O\left((2s)^{O(m^2)}\right)$ labelings of the different graphs
  - Counting the support means solving multiple NP-hard problems

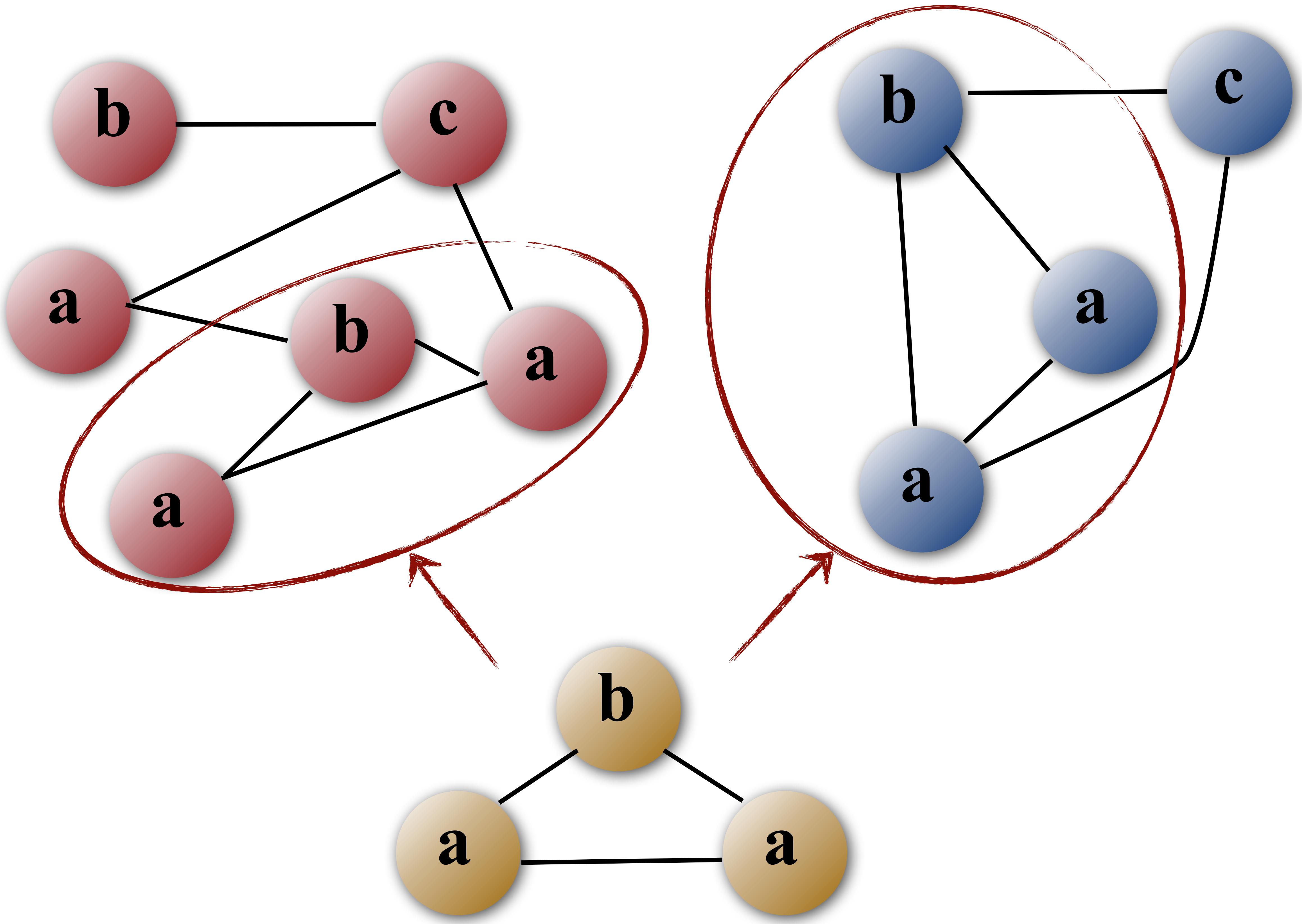
An Example



An Example



An Example



Apriori-Based Graph Mining (AGM)

- Subgraph frequency follows downwards closedness property
  - A supergraph cannot be frequent unless its subgraph is
- Idea: generate all k-vertex graphs that are supergraphs of k–1 vertex frequent graphs and check frequency
- Two problems:
  - How to generate the graphs
  - How to check the frequency
- Idea: do the generation based on adjacency matrices

Inokuchi, Washio & Motoda 2000

Matrices and Codes

- In *labelled adjacency matrix* we have
  - Vertex labels in the diagonal
  - Edge labels in off-diagonal (or 0 if no edges)
- The *code* of the the adjacency matrix $\boldsymbol{X}$ is the lower-left triangular submatrix listed in row-major order
  - $x_{1,1}x_{2,1}x_{2,2}x_{3,1}\ldots x_{k,1}\ldots x_{k,k}\ldots x_{n,n}$
- The adjacency matrices can be sorted using the standard lexicographical order in their codes

Joining Two Subgraphs

- Assume we have two frequent subgraphs of *k* vertices whose adjacency matrices agree on the first *k–1* edges

$$X_k = \begin{pmatrix} X_{k-1} & \boldsymbol{x}_1 \\ \boldsymbol{x}_2^T & x_{kk} \end{pmatrix}, Y_k = \begin{pmatrix} X_{k-1} & \boldsymbol{y}_1 \\ \boldsymbol{y}_2^T & y_{kk} \end{pmatrix}$$

- We can do the join as follows

$$Z_{k+1} = \begin{pmatrix} X_{k-1} & \boldsymbol{x}_1 & \boldsymbol{y}_1 \\ \boldsymbol{x}_2^T & x_{kk} & z_{k,k+1} \\ \boldsymbol{y}_2^T & z_{k+1,k} & y_{kk} \end{pmatrix} = \left(\begin{array}{cc|c} \multicolumn{2}{c|}{X_k} & \begin{array}{c}\boldsymbol{y}_1 \\ z_{k,k+1}\end{array} \\ \hline \boldsymbol{y}_2^T & z_{k+1,k} & y_{kk} \end{array}\right)$$

  - $z_{k+1,k} = z_{k,k+1}$ assumes all possible edge labels
    - One matrix for each possibility

Avoiding Redundancy

- The two adjacency matrices are joined only if code($\boldsymbol{X}_k$) ≤ code($\boldsymbol{Y}_k$) ("normal order")
- We need to confirm that all subgraphs of the resulting (k +1)-vertex matrix are frequent
  - We need to consider the normal-order generated k-vertex subgraphs
    - The algorithm only stores normal-order generated graphs
  - They are generated by re-generating the k-vertex subgraph from singletons in normal order
    - Process is called *normalization* and can compute the normal forms of all subgraphs
  - Normalization can be expressed as a row and column permutations: $\boldsymbol{X}_n = \boldsymbol{P}^T\boldsymbol{X}\boldsymbol{P}$

Canonical Forms

- Isomorphic graphs can have many different normal forms
- Given a set $NF(G)$ of all normal forms representing graphs isomorphic to G, the *canonical form* of G is the adjacency matrix $\boldsymbol{X}_c$ that has the minimum code in $NF(G)$

$$\boldsymbol{X}_c = \arg\min \{code(X) : X \in NF(G)\}$$

- Given an adjacency matrix $\boldsymbol{X}$, its normal form is $\boldsymbol{X}_n = \boldsymbol{P}^T\boldsymbol{X}\boldsymbol{P}$ for some permutation matrix $\boldsymbol{P}$, and its canonical form $\boldsymbol{X}_c$ is $\boldsymbol{Q}^T\boldsymbol{P}^T\boldsymbol{X}\boldsymbol{P}\boldsymbol{Q}$ for some permutation matrix $\boldsymbol{Q}$

Finding Canonical Forms

- Let $\boldsymbol{X}$ be an adjacency matrix of $k+1$ vertices
  - Let $\boldsymbol{Y}$ be $\boldsymbol{X}$ with vertex m removed
  - Let $\boldsymbol{P}$ be the permutation of $\boldsymbol{Y}$ to its normal form and $\boldsymbol{Q}$ the permutation of $\boldsymbol{P}^T\boldsymbol{Y}\boldsymbol{P}$ to the canonical form
    - We assume we have already computed them
  - We compute candidate $\boldsymbol{P}'$ and $\boldsymbol{Q}'$ for $\boldsymbol{X}$ by
    - $\boldsymbol{Q}'$ is like $\boldsymbol{Q}$ but bottom-right corner is 1
    - p'_{ij} is
      - p_{ij} if $i < m$ and $j \neq k$
      - $p_{i-1,j}$ if $i > m$ and $j \neq k$
      - 1 if $i = m$ and $j = k$
      - 0 otherwise
  - Final $\boldsymbol{P}'$ and $\boldsymbol{Q}'$ are found by trying all candidates and selecting the ones that give the lowest code

The Algorithm

- Start with frequent graphs of 1 vertex
- **while** there are frequent graphs left
  - Join two frequent (k–1)-vertex graphs
  - Check the resulting graphs subgraphs are frequent
    - If not, **continue**
  - Compute the canonical form of the graph
    - If this canonical form has already been studied, **continue**
  - Compare the canonical form with the canonical forms of the k-vertex subgraphs of the graphs in D
    - If the graph is frequent, keep, otherwise discard
- **return** all frequent subgraphs

The gSpan Algorithm

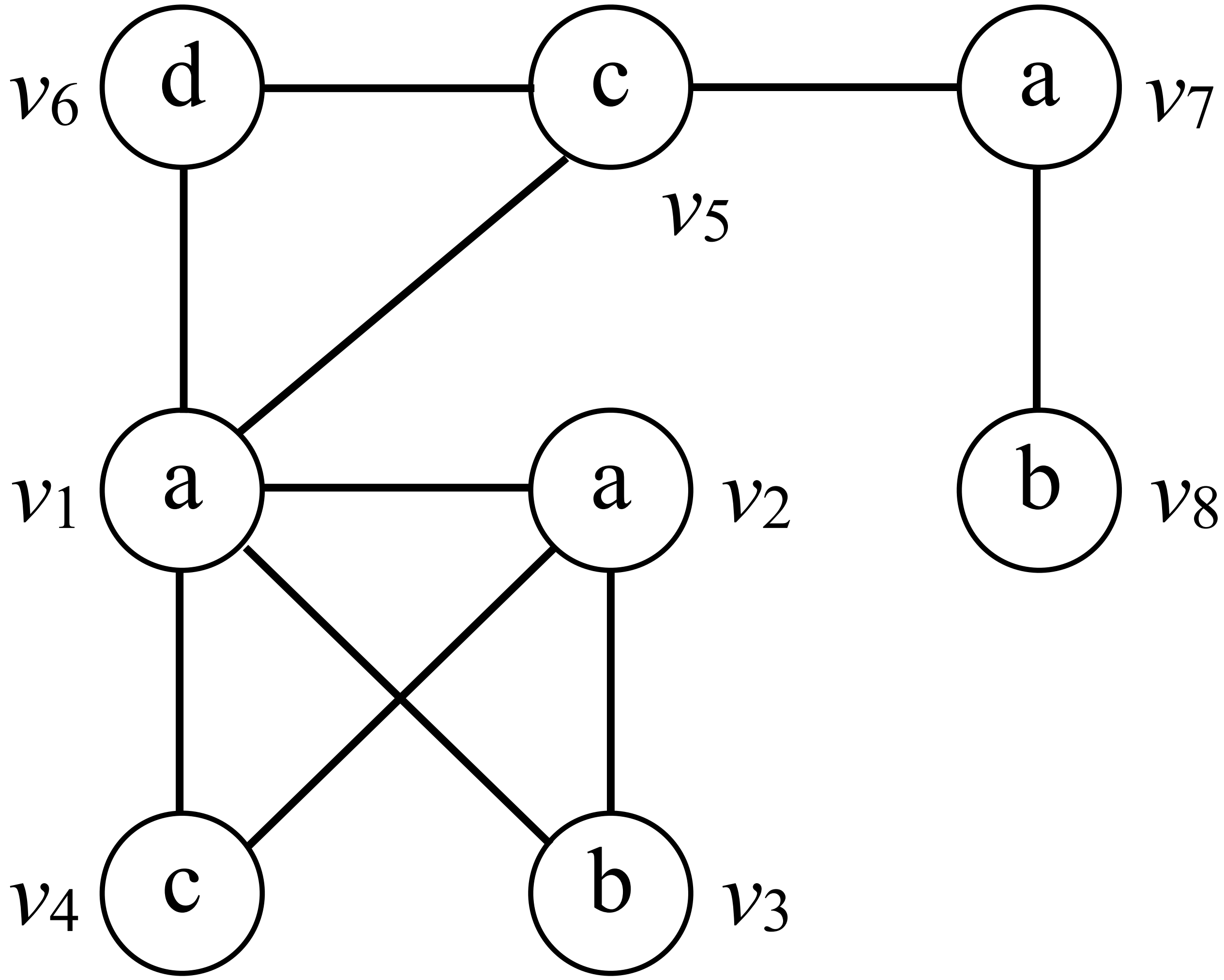
- We can improve the running time of frequent subgraph mining by either
  - Making the frequency check faster
    - Lots of efforts in faster isomorphism checking but only little progress
  - Creating less candidates that need to be checked
    - Level-wise algorithms (like AGM) generate huge numbers of candidates
    - Each must be checked with for isomorphism with others
- The gSpan (graph-based Substructure pattern mining) algorithm replaces the level-wise approach with a depth-first approach

Yan & Han 2002; Z&M Ch. 11

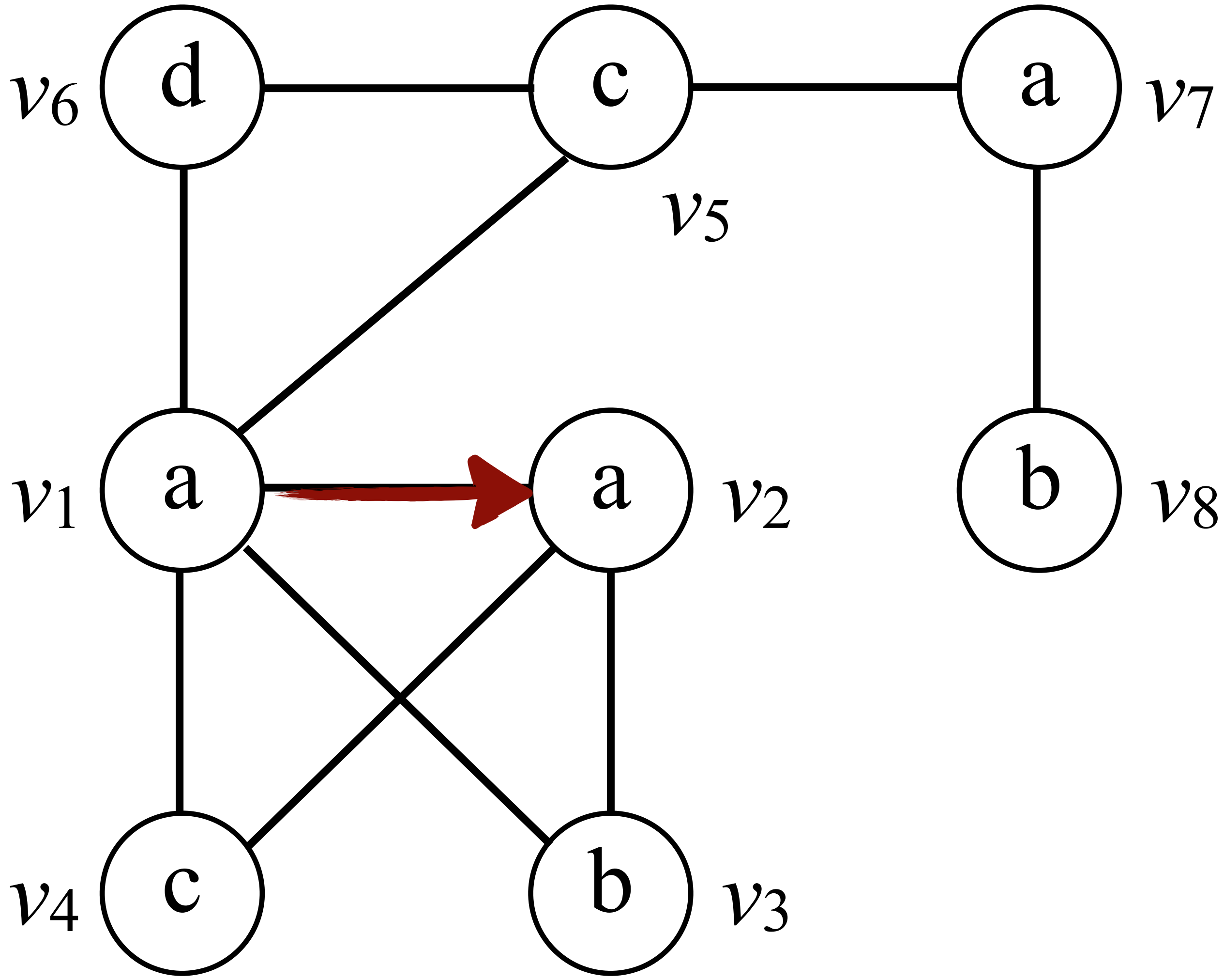
Depth-First Spanning Tree

- A dept-first spanning (DFS) tree of a graph *G*
  - Is a connected tree
  - Contains all the vertices of *G*
  - Is build in depth-first order
    - Selection between the siblings is e.g. based on the vertex index
- Edges of the DFS tree are *forward edges*
- Edges not in the DFS tree are *backward edges*
- A *rightmost path* in the DFS tree is the path travels from the root to the *rightmost vertex* by always taking the rightmost child (last-added)

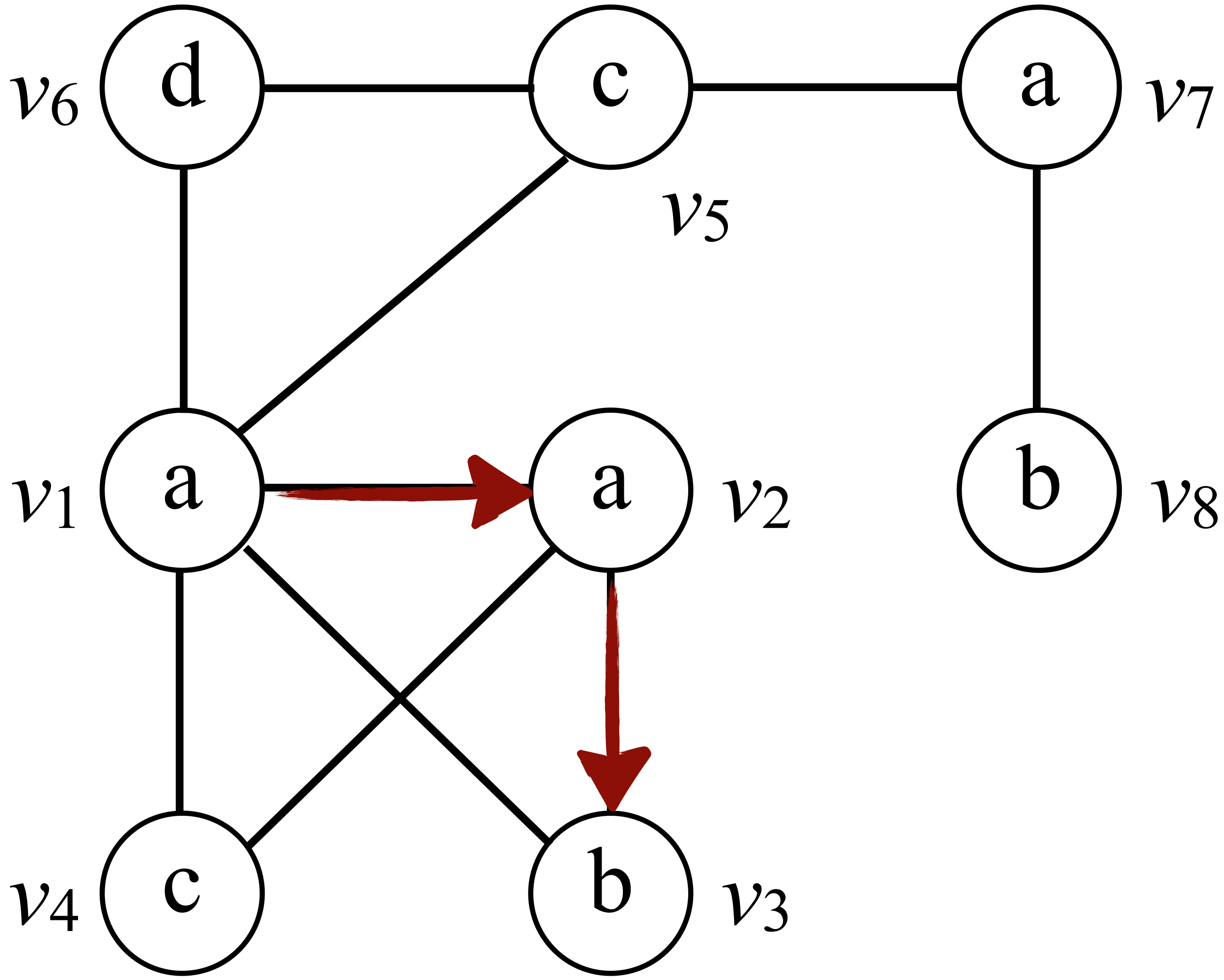
An Example



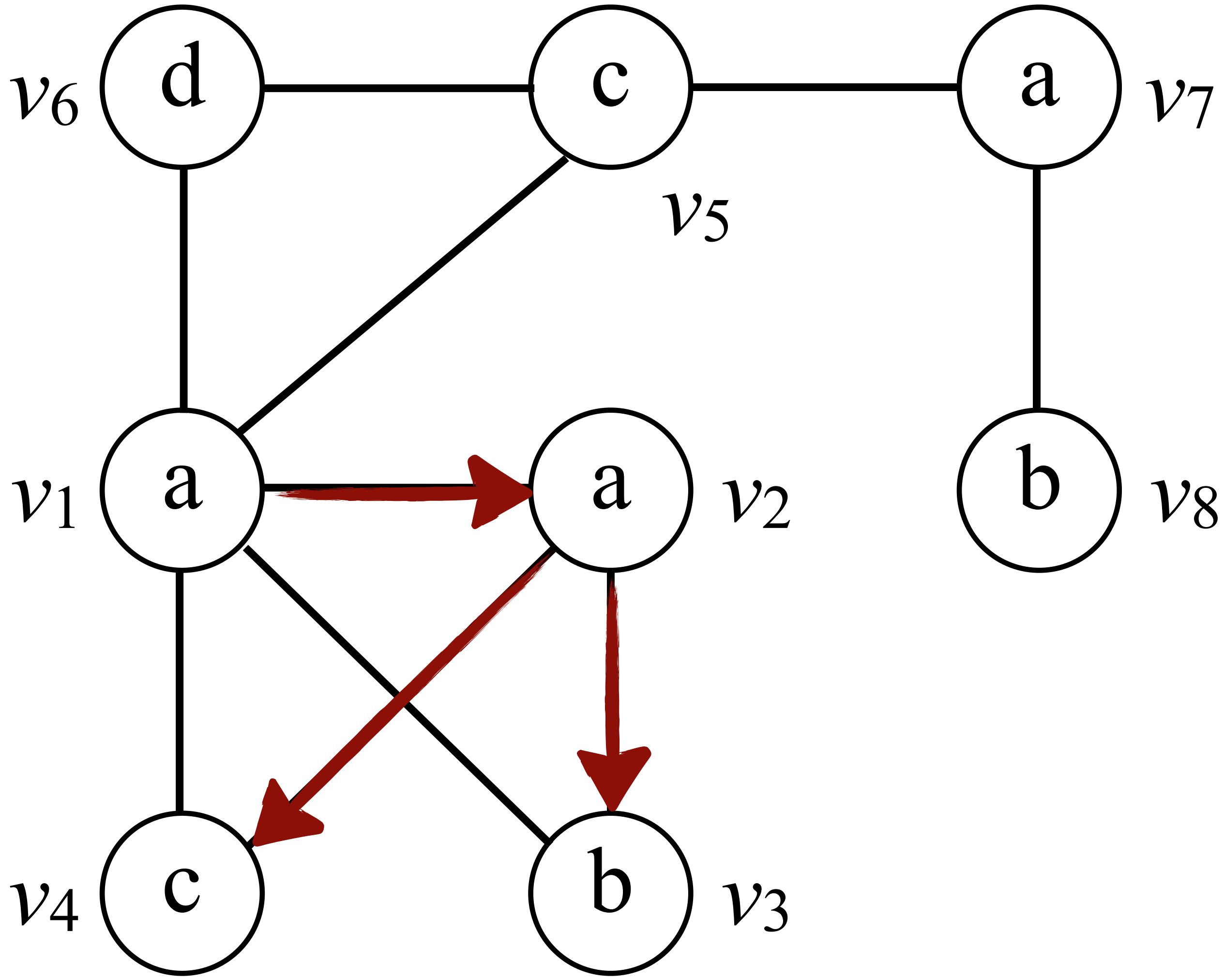
An Example



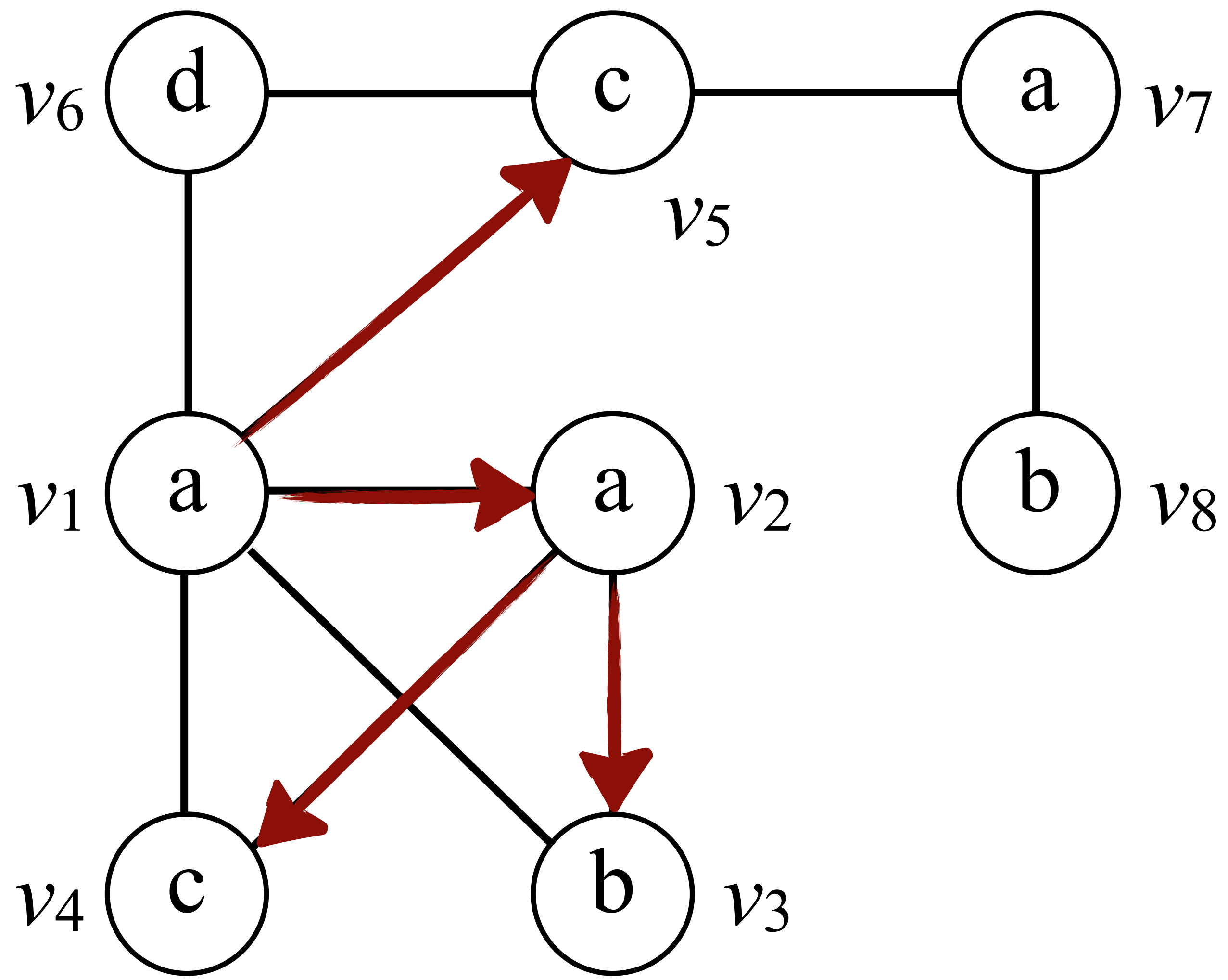
An Example



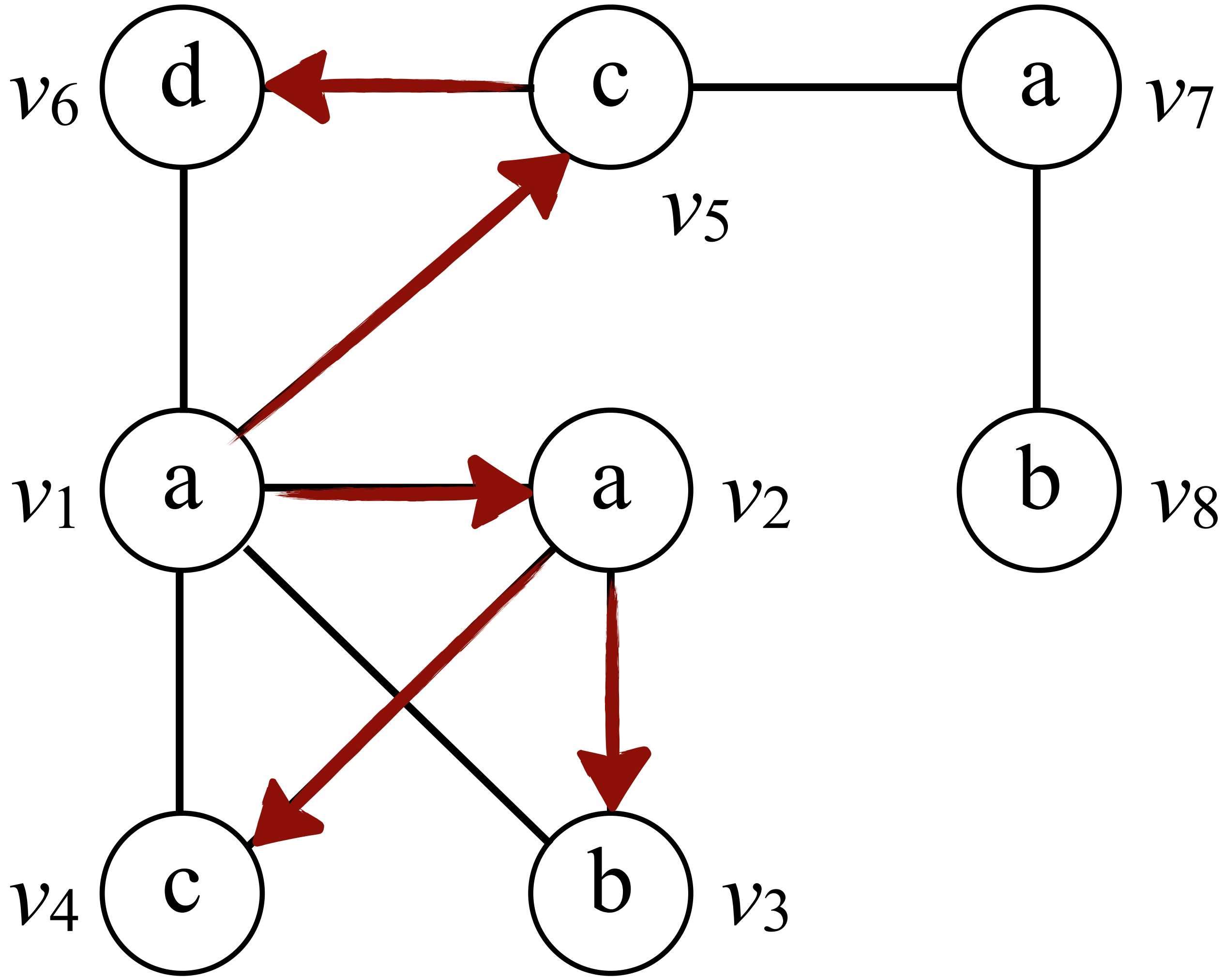
An Example



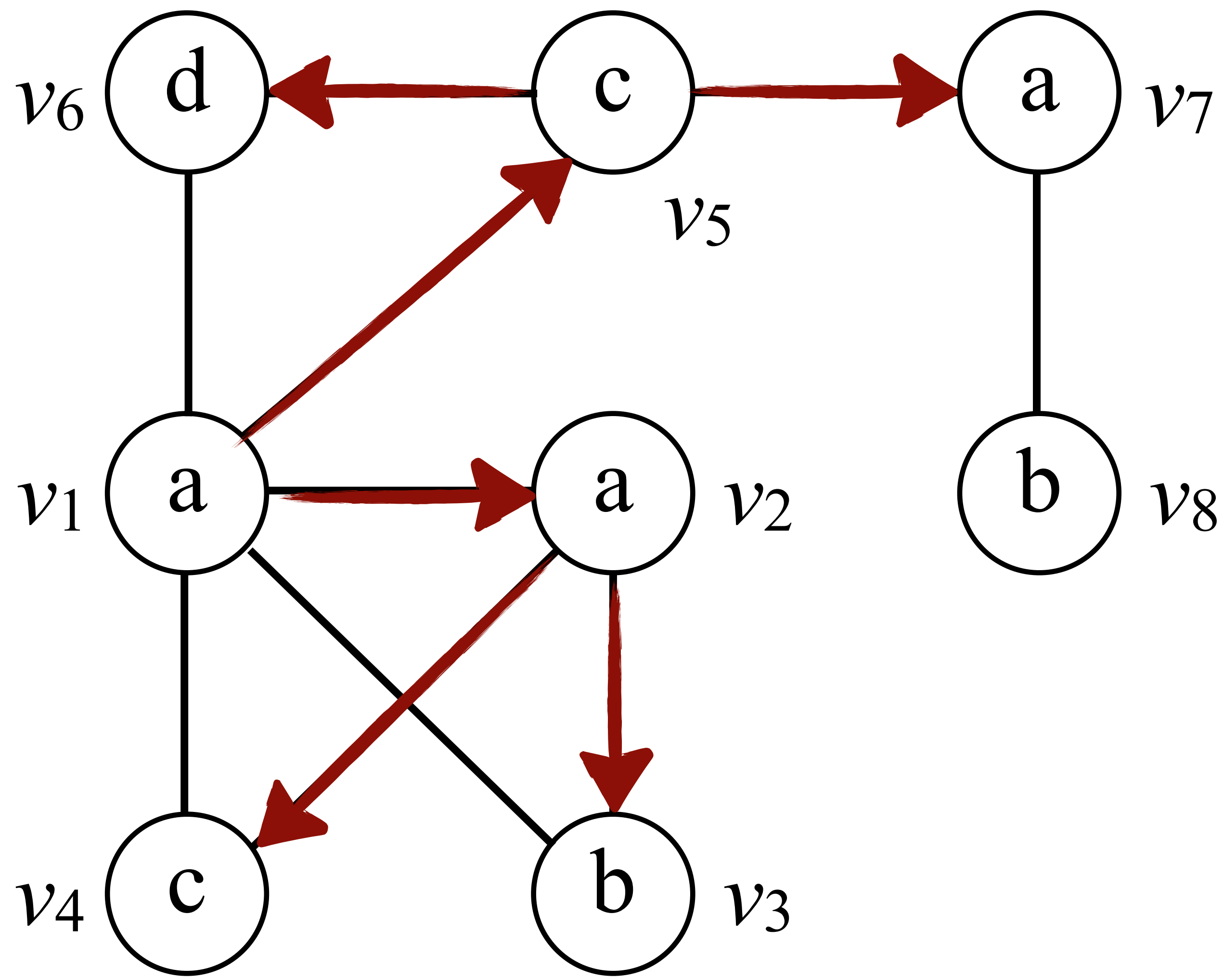
An Example



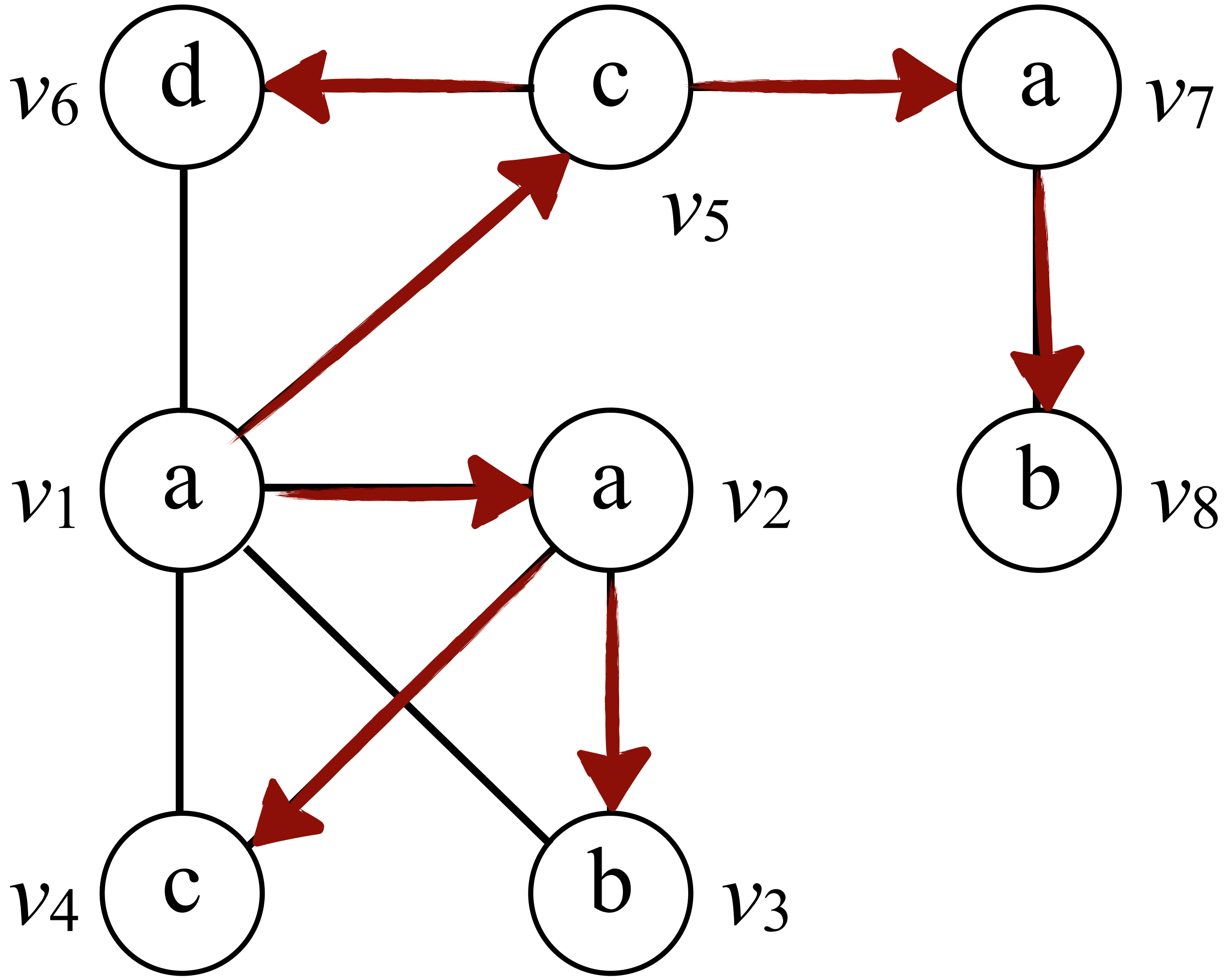
An Example



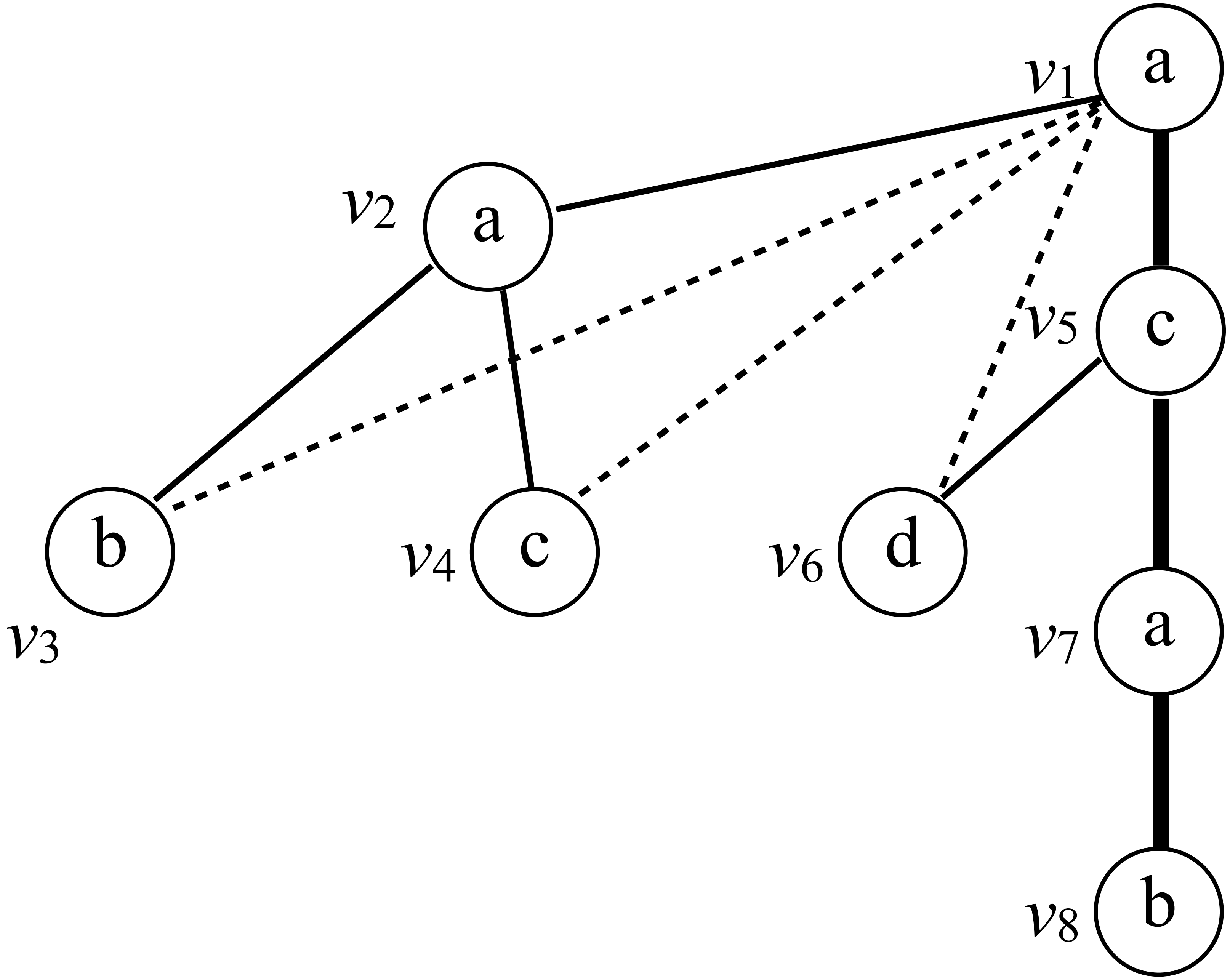
An Example



An Example



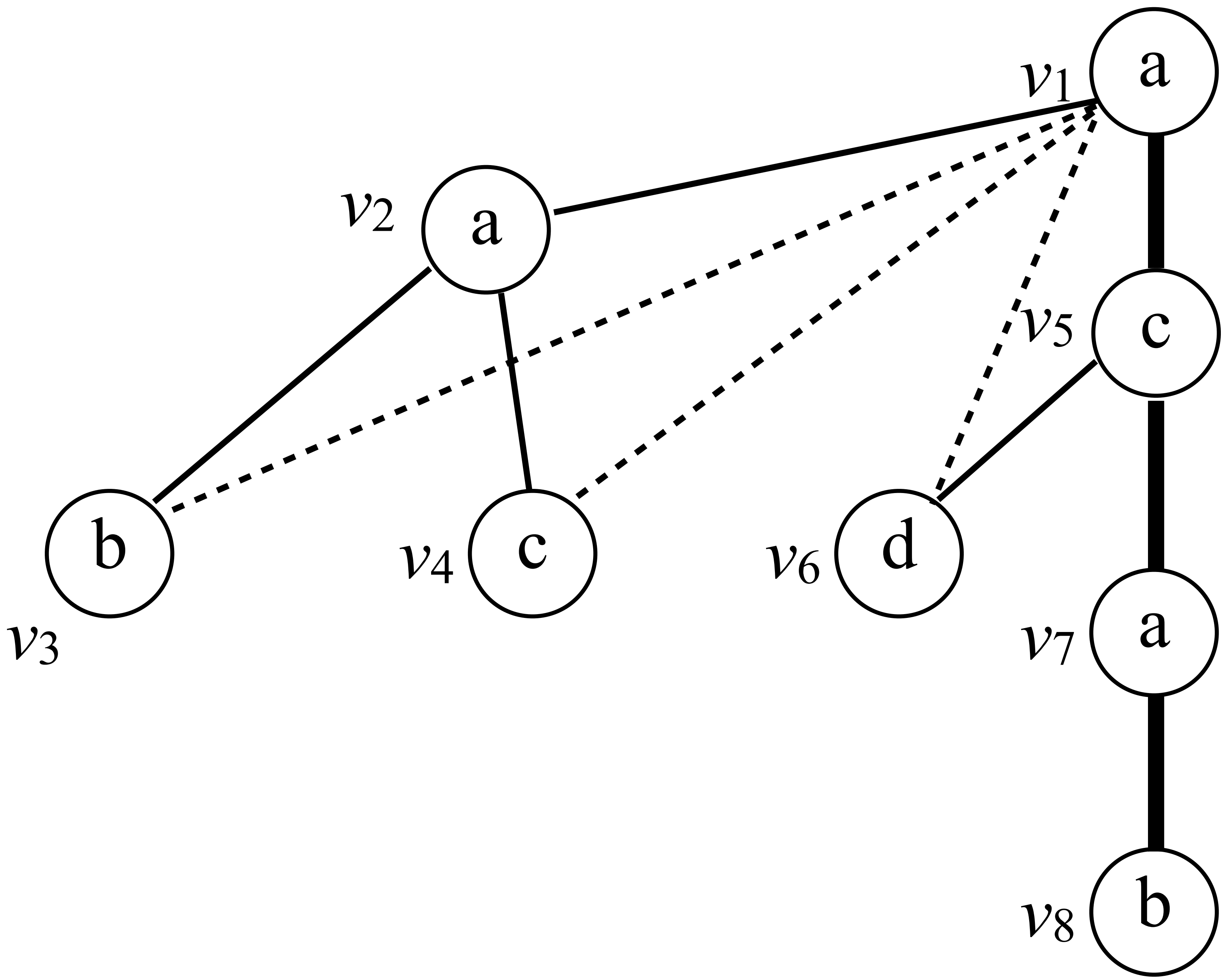
The DFS Tree



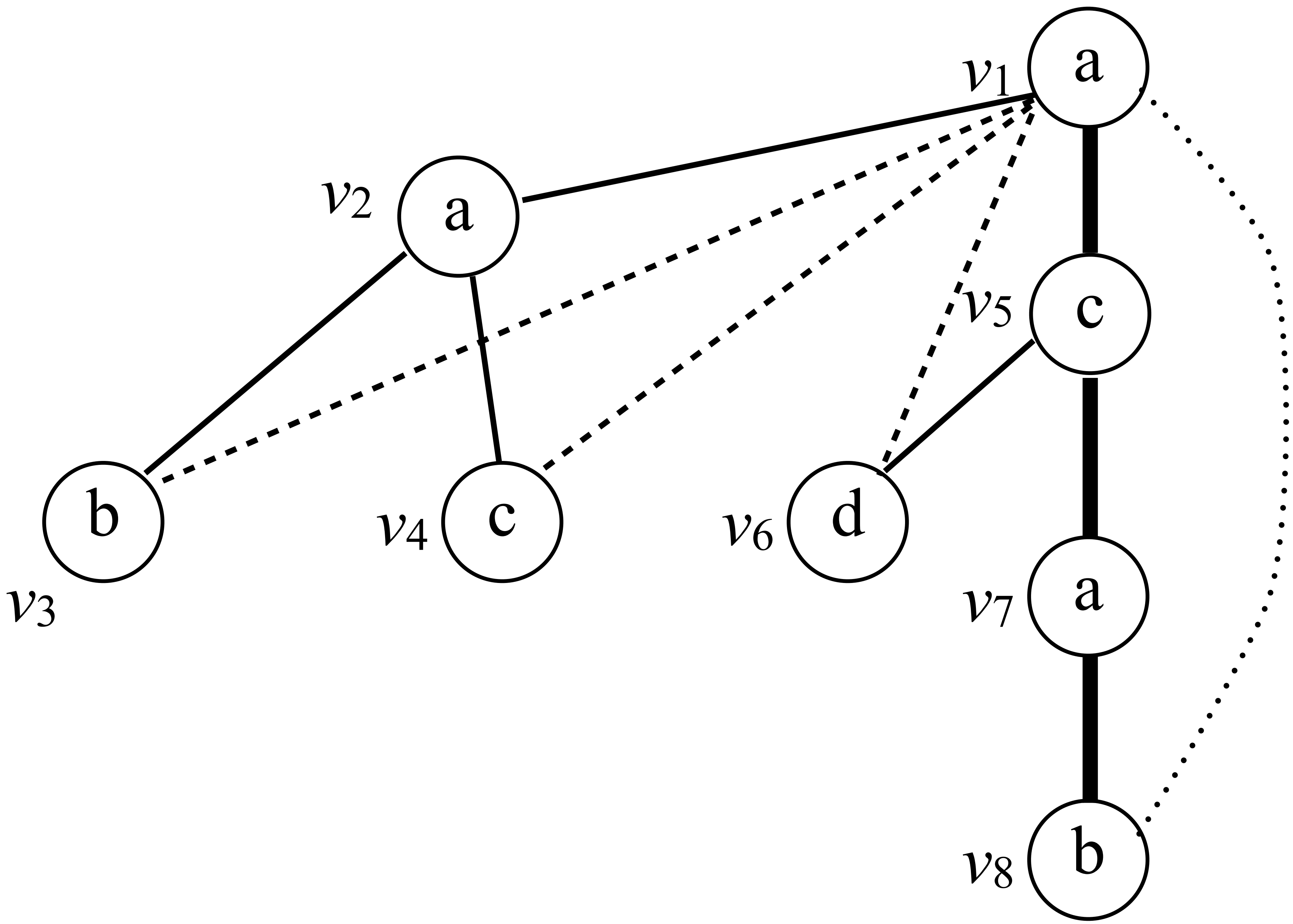
Generating Candidates from DFS Tree

- Given graph G, we extend it *only* from the vertices in the rightmost path
  - We can add backwards edges from the rightmost vertex to some other vertex in the rightmost path
  - We can add a forward edge from any vertex in the rightmost path
    - This increases the number of vertices by 1
- The order of generating the candidates is
  - First backward extensions
    - First to root, then to root's child, …
  - Then forward extensions
    - First from the leaf, then from leaf's father, …

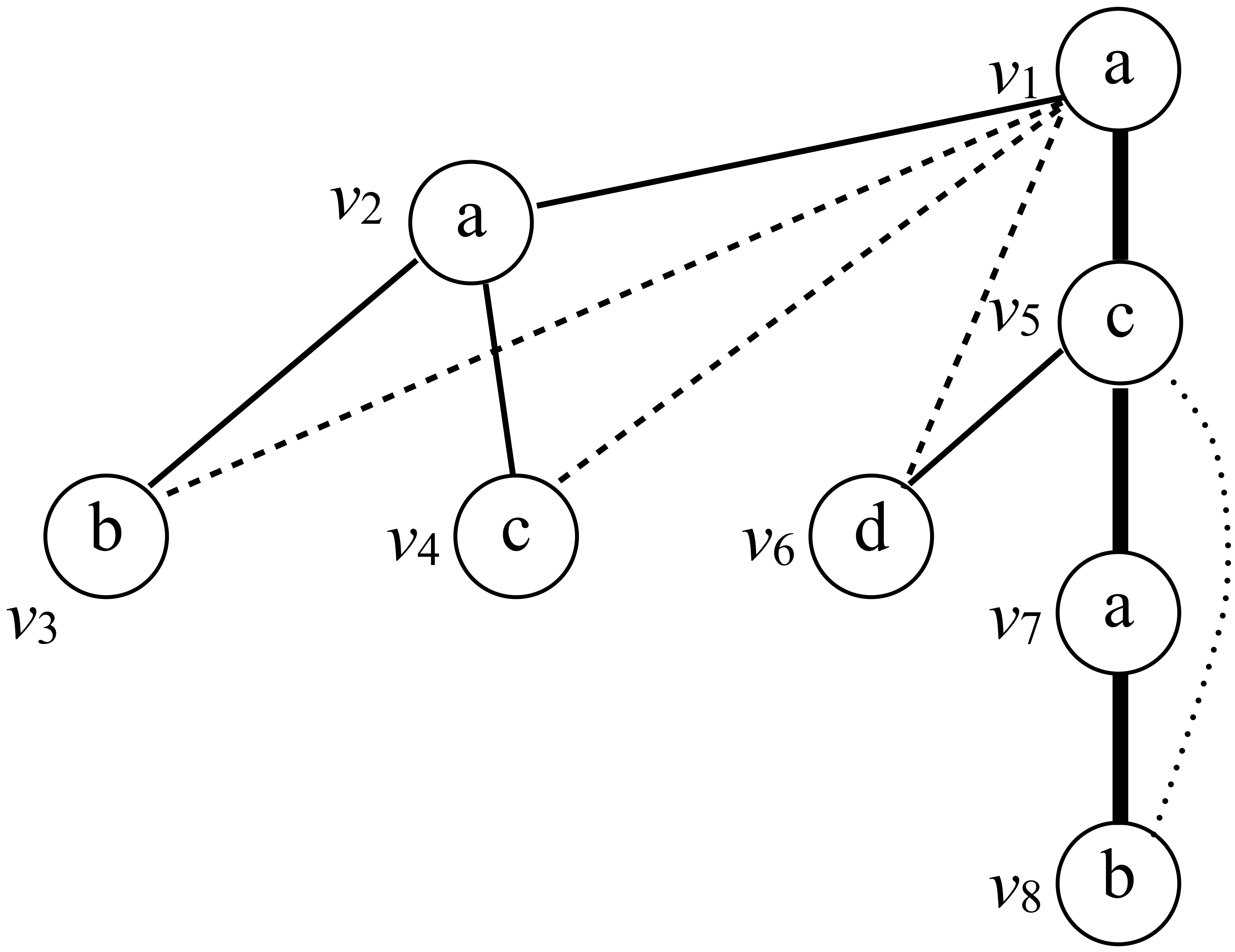
An Example



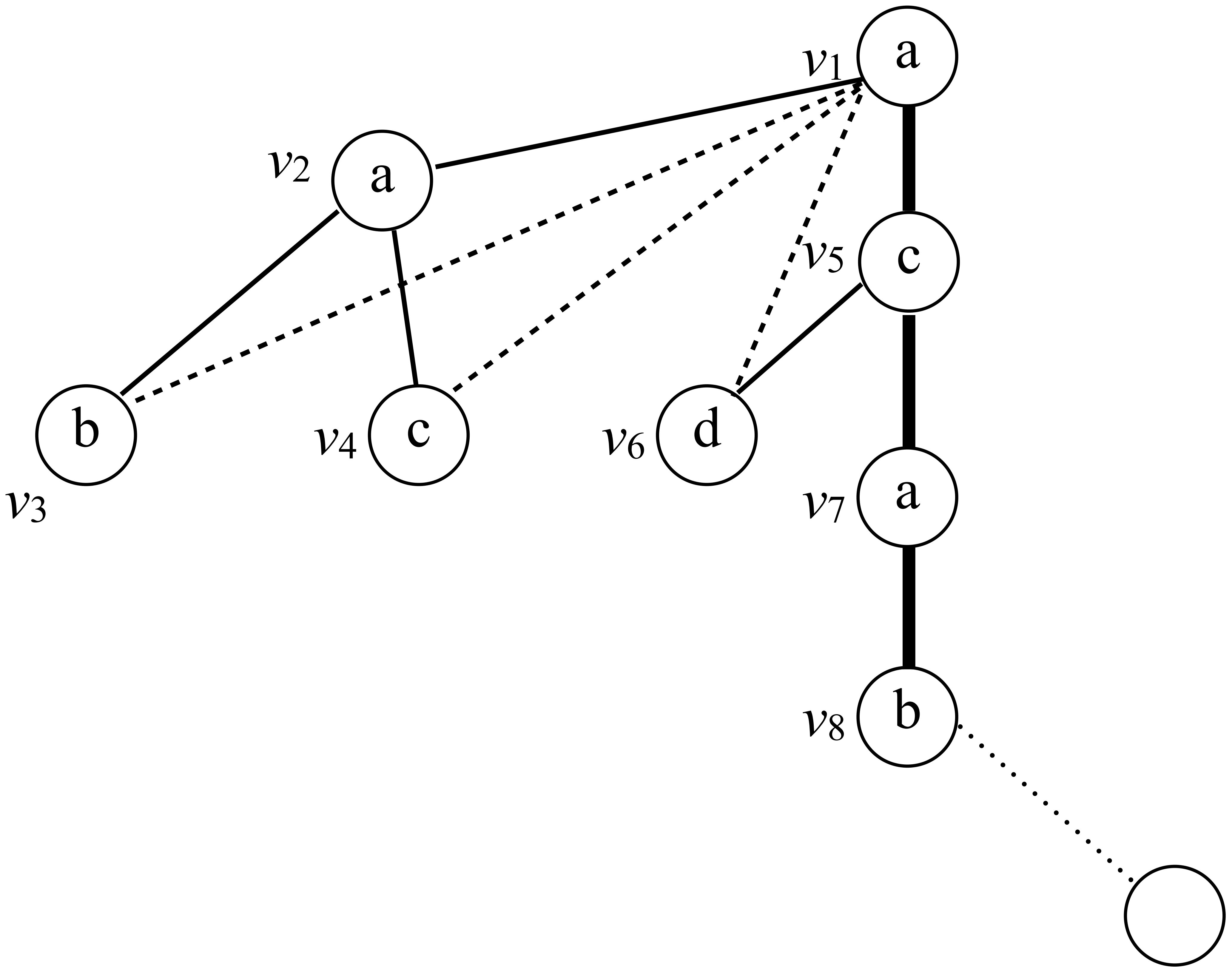
An Example



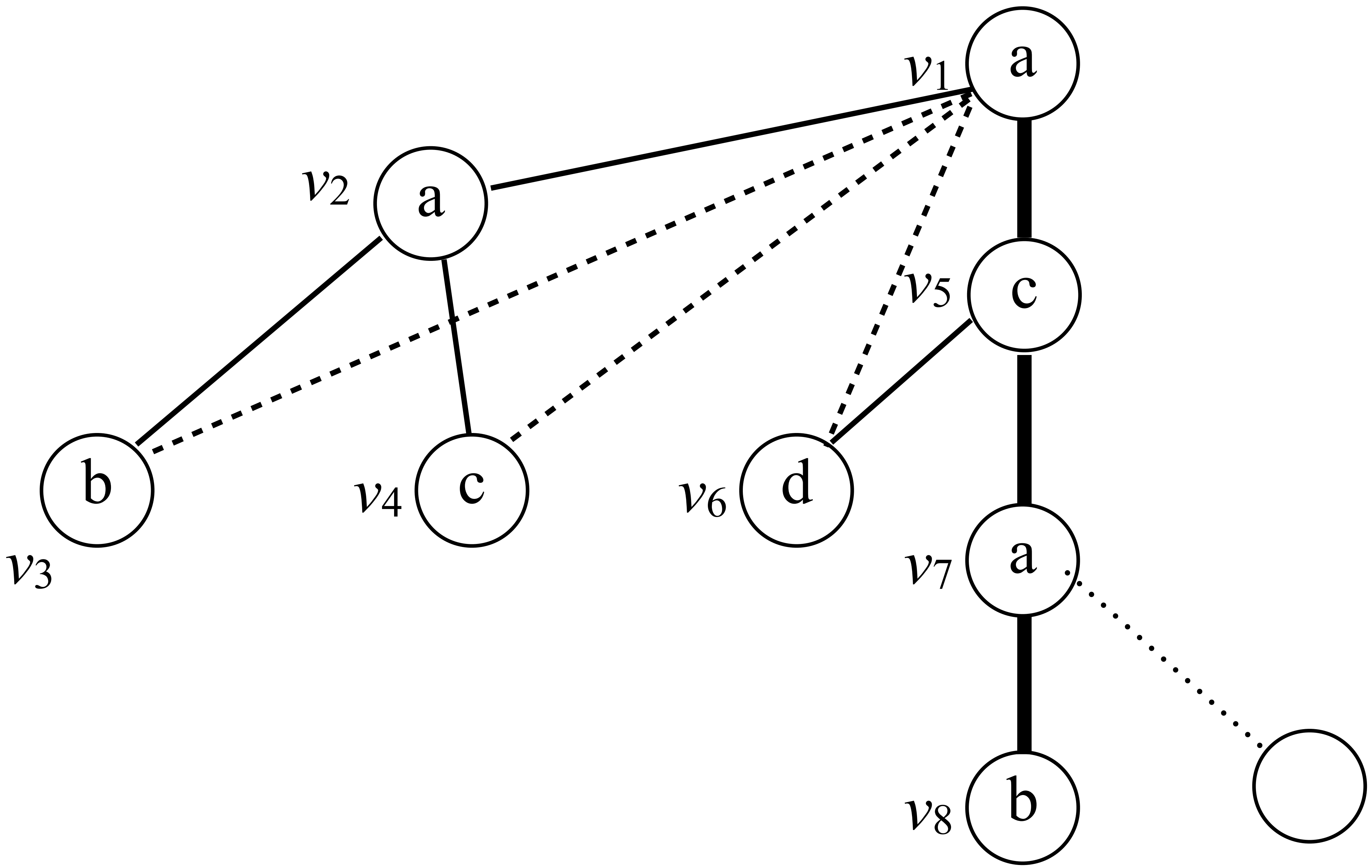
An Example



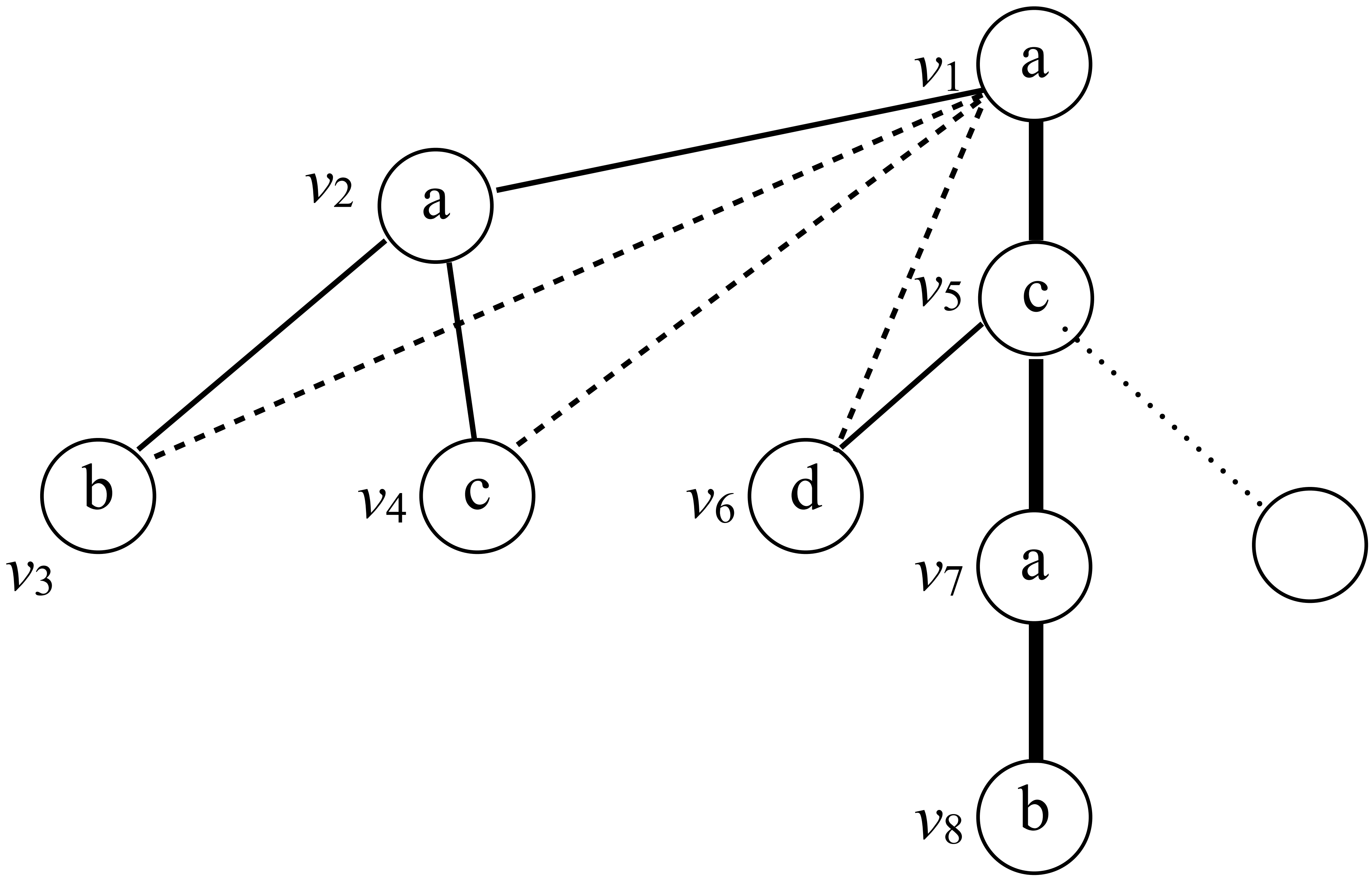
An Example



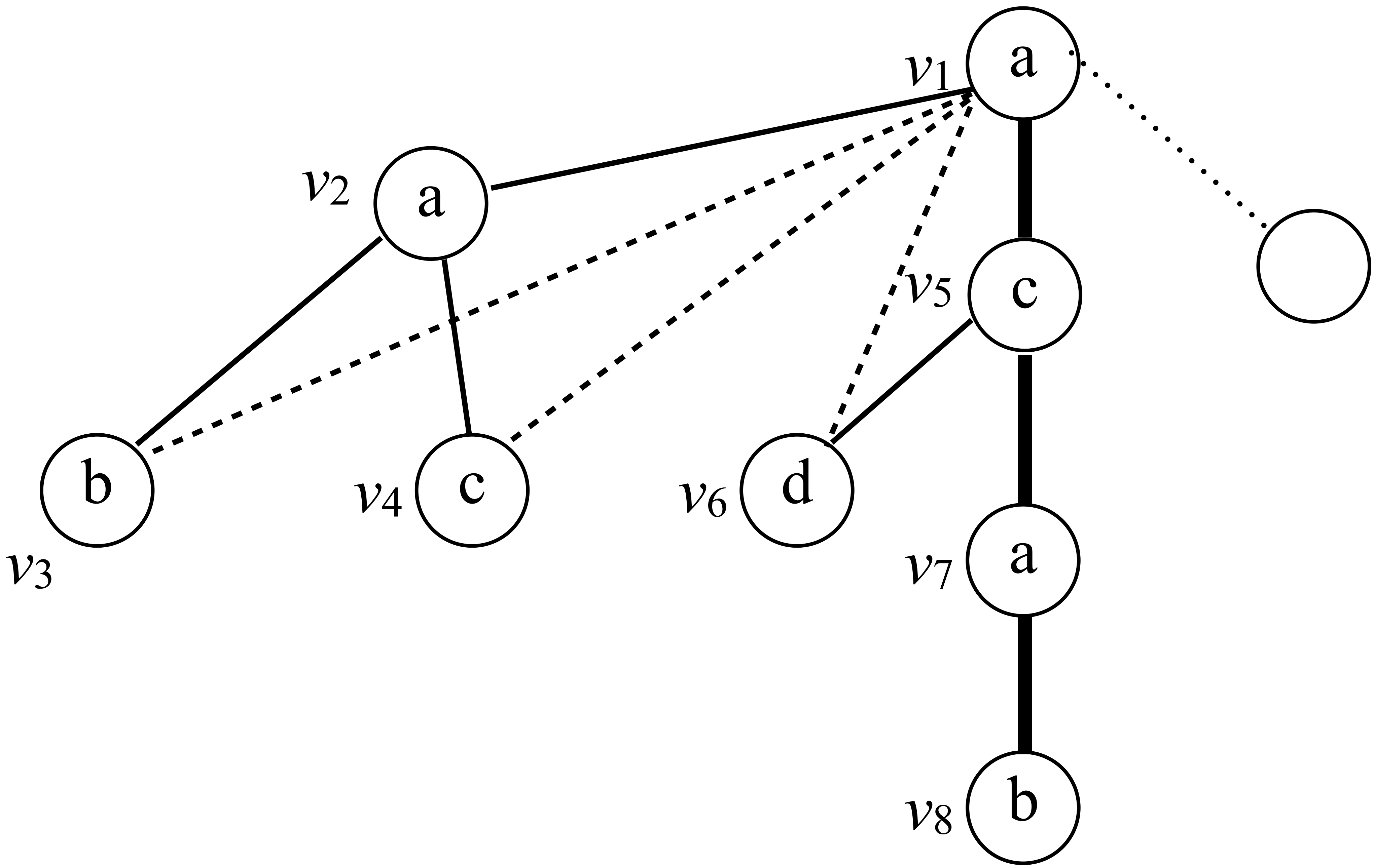
An Example



An Example



An Example



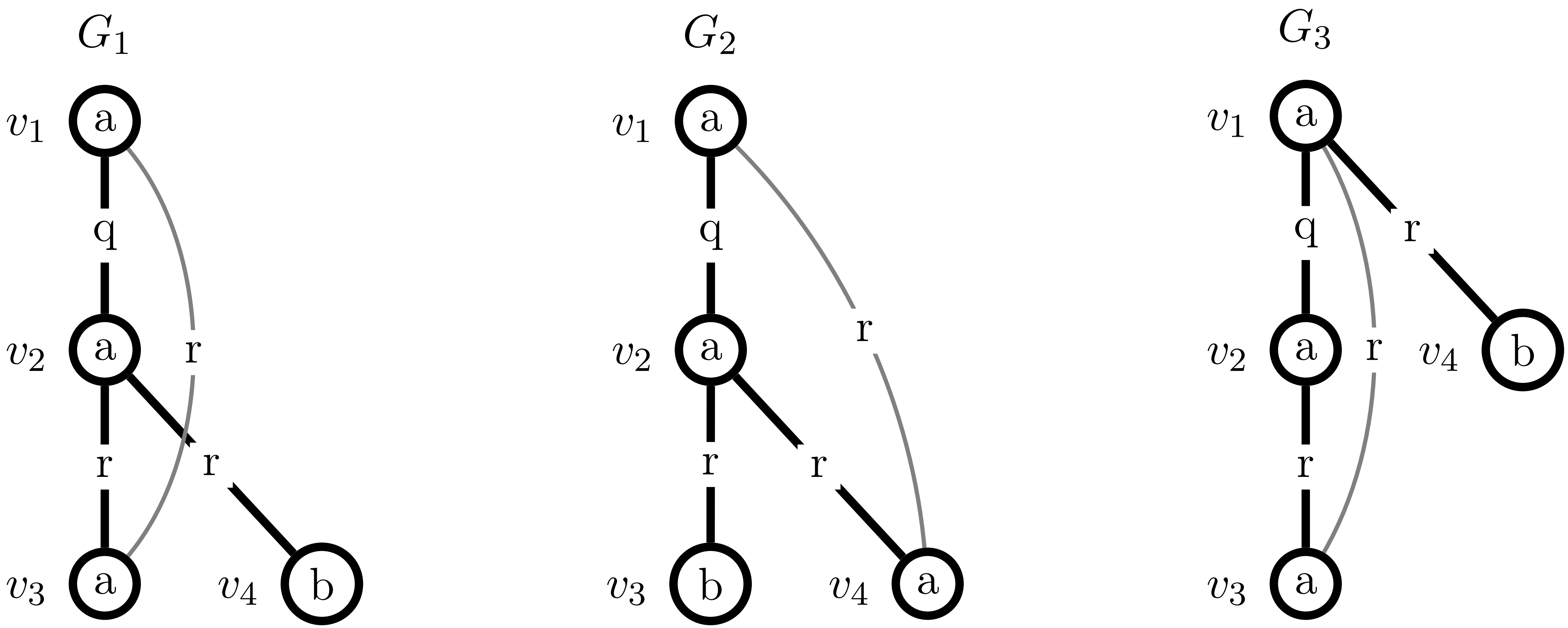
DFS Codes and their Orders

- A *DFS code* is a sequence of tuples of type $\langle v_i, v_j, L(v_i), L(v_j), L(v_i, v_j)\rangle$
  - Tuples are given in DFS order
    - Backwards edges are listed before forward edges
- A DFS code is *canonical* if it is the smallest of the codes in the ordering
  - $\langle v_i, v_j, L(v_i), L(v_j), L(v_i, v_j)\rangle < \langle v_x, v_y, L(v_x), L(v_y), L(v_x, v_y)\rangle$ if
    - $\langle v_i, v_j\rangle <_e \langle v_x, v_y\rangle$; or
    - $\langle v_i, v_j\rangle = \langle v_x, v_y\rangle$ and $\langle L(v_i), L(v_j), L(v_i, v_j)\rangle <_l \langle L(v_x), L(v_y), L(v_x, v_y)\rangle$
  - The ordering of the label tuples is the lexicographical ordering

Ordering the Edges

- Let $e_{ij} = \langle v_i, v_j \rangle$ and $e_{xy} = \langle v_x, v_y \rangle$
- $e_{ij} <_e e_{xy}$ if
  - If e_{ij} and e_{xy} are forward edges, then
    - $j < y$; or
    - $j = y$ and $i > x$
  - If e_{ij} and e_{xy} are backward edges, then
    - $i < x$; or
    - $i = x$ and $j < y$
  - If e_{ij} is forward and e_{xy} is backward, then $i < y$
  - If e_{ij} is backward and e_{xy} is forward, then $j \leq x$

Example

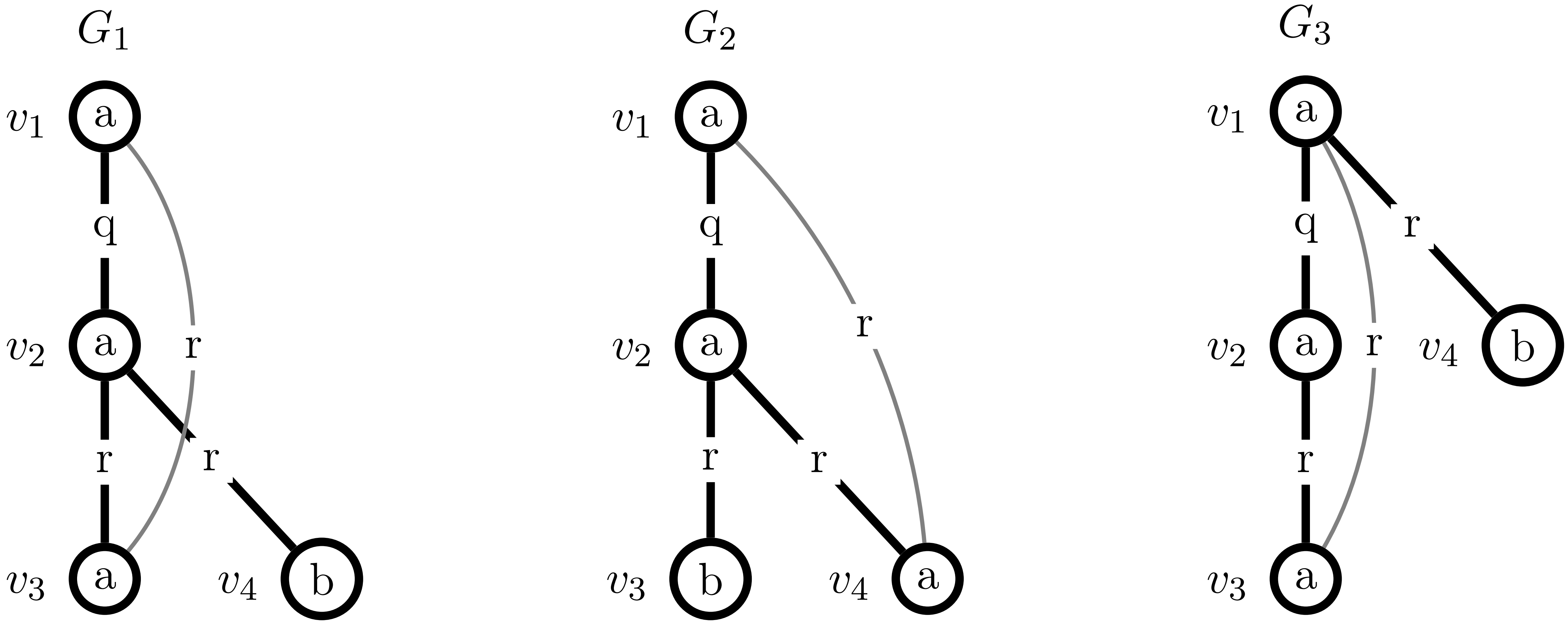


$$
\begin{aligned}
t_{11} &= \langle v_1, v_2, a, a, q \rangle \\
t_{12} &= \langle v_2, v_3, a, a, r \rangle \\
t_{13} &= \langle v_3, v_1, a, a, r \rangle \\
t_{14} &= \langle v_2, v_4, a, b, r \rangle
\end{aligned}
$$

$$
\begin{aligned}
t_{21} &= \langle v_1, v_2, a, a, q \rangle \\
t_{22} &= \langle v_2, v_3, a, b, r \rangle \\
t_{23} &= \langle v_2, v_4, a, a, r \rangle \\
t_{24} &= \langle v_4, v_1, a, a, r \rangle
\end{aligned}
$$

$$
\begin{aligned}
t_{31} &= \langle v_1, v_2, a, a, q \rangle \\
t_{32} &= \langle v_2, v_3, a, a, r \rangle \\
t_{33} &= \langle v_3, v_1, a, a, r \rangle \\
t_{34} &= \langle v_1, v_4, a, b, r \rangle
\end{aligned}
$$

Example



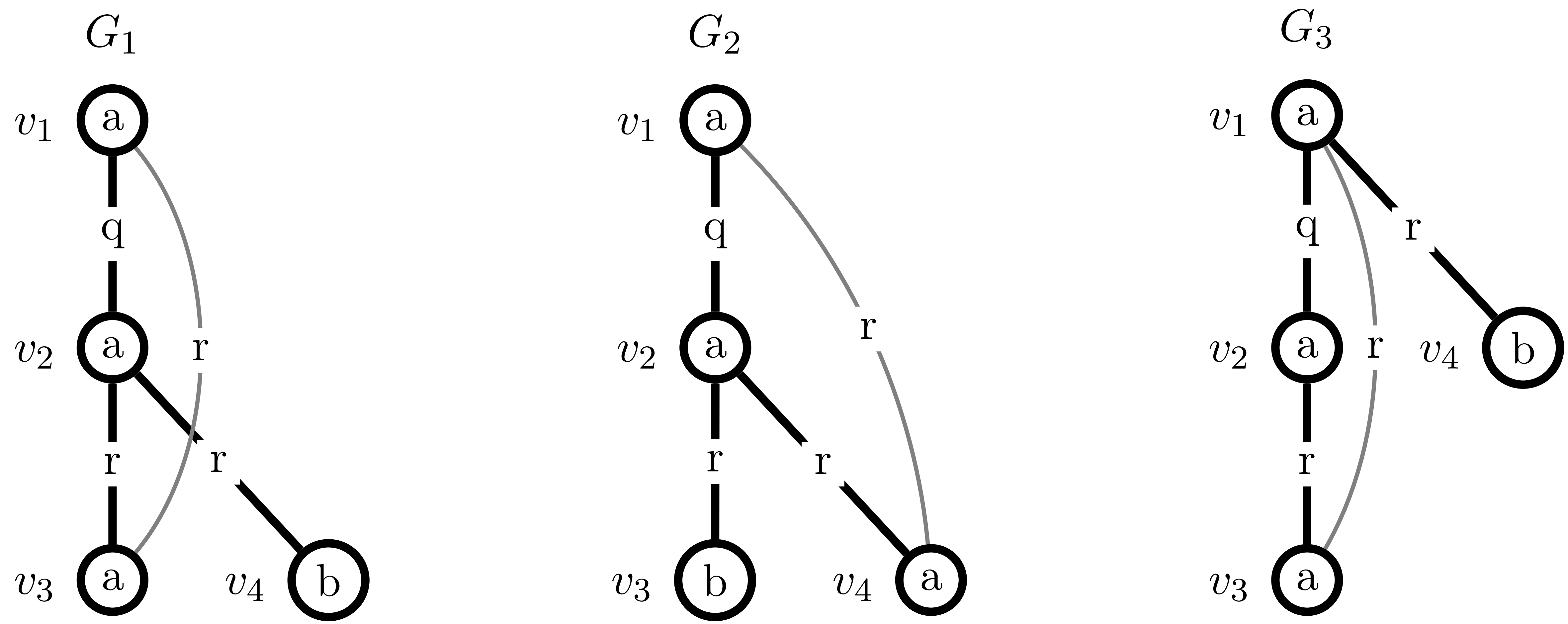
$$\begin{aligned} t_{11} &= \langle v_1, v_2, a, a, q\rangle \\ t_{12} &= \langle v_2, v_3, a, a, r\rangle \\ t_{13} &= \langle v_3, v_1, a, a, r\rangle \\ t_{14} &= \langle v_2, v_4, a, b, r\rangle \end{aligned}$$

$$\begin{aligned} t_{21} &= \langle v_1, v_2, a, a, q\rangle \\ t_{22} &= \langle v_2, v_3, a, b, r\rangle \\ t_{23} &= \langle v_2, v_4, a, a, r\rangle \\ t_{24} &= \langle v_4, v_1, a, a, r\rangle \end{aligned}$$

$$\begin{aligned} t_{31} &= \langle v_1, v_2, a, a, q\rangle \\ t_{32} &= \langle v_2, v_3, a, a, r\rangle \\ t_{33} &= \langle v_3, v_1, a, a, r\rangle \\ t_{34} &= \langle v_1, v_4, a, b, r\rangle \end{aligned}$$

First rows are identical

Example



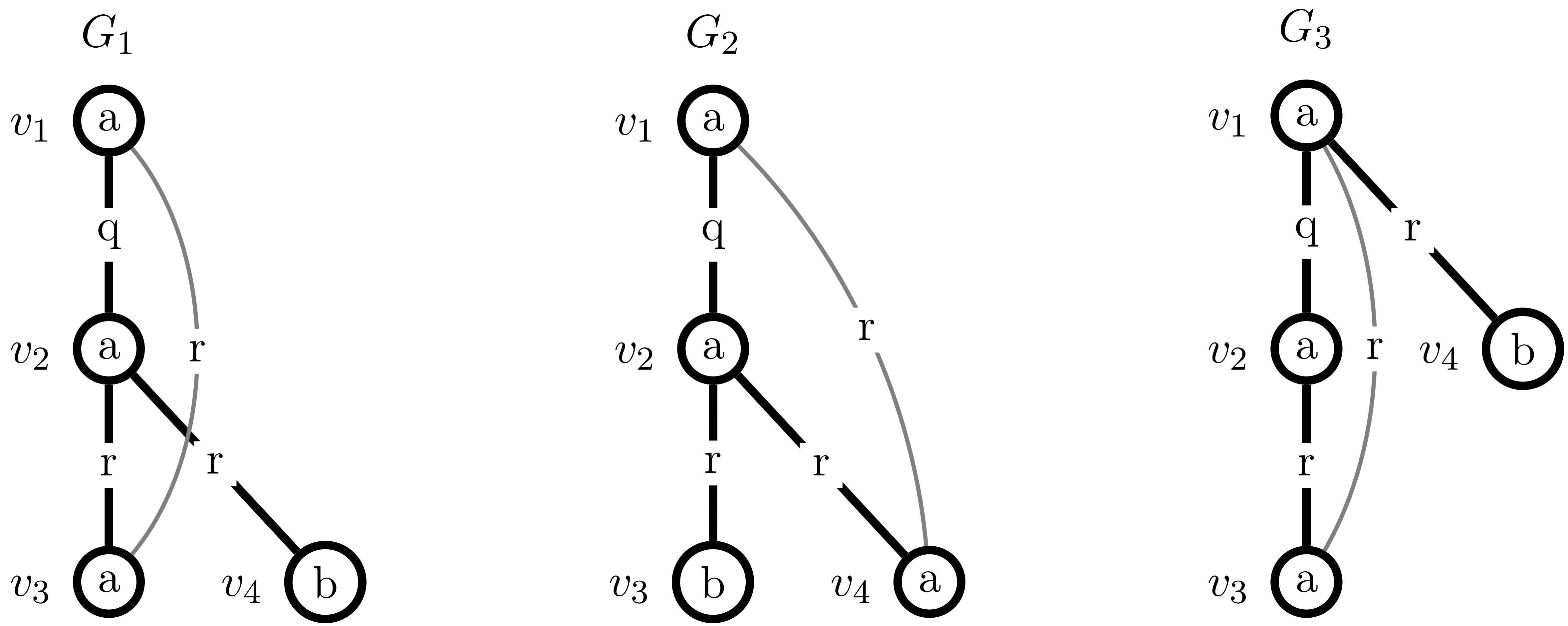
$$t_{11} = \langle v_1, v_2, a, a, q\rangle$$
$$t_{12} = \langle v_2, v_3, a, a, r\rangle$$
$$t_{13} = \langle v_3, v_1, a, a, r\rangle$$
$$t_{14} = \langle v_2, v_4, a, b, r\rangle$$

$$t_{21} = \langle v_1, v_2, a, a, q\rangle$$
$$t_{22} = \langle v_2, v_3, a, b, r\rangle$$
$$t_{23} = \langle v_2, v_4, a, a, r\rangle$$
$$t_{24} = \langle v_4, v_1, a, a, r\rangle$$

$$t_{31} = \langle v_1, v_2, a, a, q\rangle$$
$$t_{32} = \langle v_2, v_3, a, a, r\rangle$$
$$t_{33} = \langle v_3, v_1, a, a, r\rangle$$
$$t_{34} = \langle v_1, v_4, a, b, r\rangle$$

In second row, G_2 is bigger in labels' order

Example



$$\begin{aligned} t_{11} &= \langle v_1, v_2, a, a, q\rangle \\ t_{12} &= \langle v_2, v_3, a, a, r\rangle \\ t_{13} &= \langle v_3, v_1, a, a, r\rangle \\ \cancel{t_{14}} &\rightarrow \langle v_2, v_4, a, b, r\rangle \end{aligned}$$

$$\begin{aligned} t_{21} &= \langle v_1, v_2, a, a, q\rangle \\ t_{22} &= \langle v_2, v_3, a, b, r\rangle \\ t_{23} &= \langle v_2, v_4, a, a, r\rangle \\ t_{24} &= \langle v_4, v_1, a, a, r\rangle \end{aligned}$$

$$\begin{aligned} t_{31} &= \langle v_1, v_2, a, a, q\rangle \\ t_{32} &= \langle v_2, v_3, a, a, r\rangle \\ t_{33} &= \langle v_3, v_1, a, a, r\rangle \\ \cancel{t_{34}} &\rightarrow \langle v_1, v_4, a, b, r\rangle \end{aligned}$$

Last rows are forward edges and 4 = 4 but 2 > 1 $\Rightarrow$ G_1 is smallest

Building the Candidates

- The candidates are build in a *DFS code tree*
  - A DFS code **a** is an *ancestor* of DFS code **b** if **a** is a proper prefix of **b**
  - The siblings in the tree follow the DFS code order
- A graph can be frequent only if all of the graph representing its ancestors in the DFS tree are frequent
- The DFS tree contains all the canonical codes for all the subgraphs of the graphs in the data
  - But not all of the vertices in the code tree correspond to canonical codes
- We will (implicitly) traverse this tree

The Algorithm

- `gSpan`:
  - **for each** frequent 1-edge graphs
    - **call** `subgrm` to grow all nodes in the code tree rooted in this 1-edge graph
    - remove this edge from the graph
- `subgrm`
  - **if** the code is not canonical, return
  - Add this graph to the set of frequent graphs
  - Create each super-graph with one more edge and compute its frequency
  - **call** `subgrm` with each frequent super-graph