Graph App.

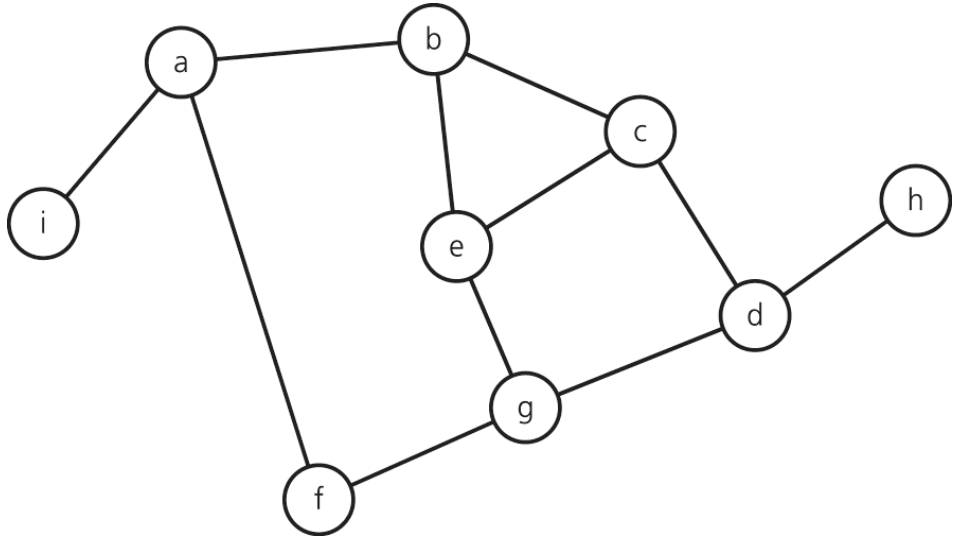
- Topological Sort
- Spanning Tree
- Shortest Paths

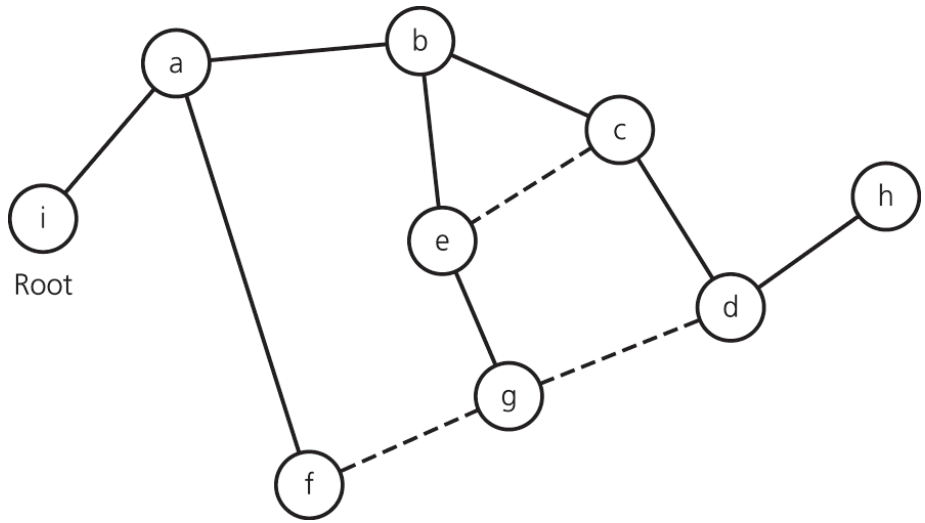
Spanning Tree: *Definition*

- **A tree is an undirected connected graph without cycles (*acyclic*)**
- **A spanning tree of a connected undirected graph *G* is**
  - A subgraph of *G* that contains all of *G*'s vertices and enough of its edges to form a tree
  - Application example: *communication network*

Spanning Tree: *Definition*

- **To obtain a spanning tree from a connected undirected graph with cycles**
  - Remove edges until there are no cycles



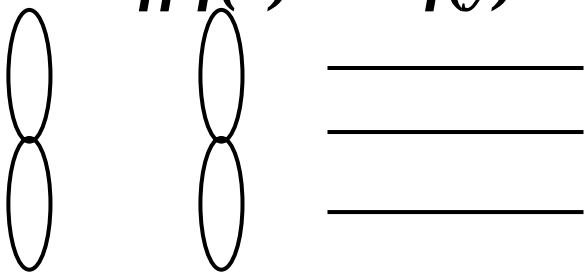


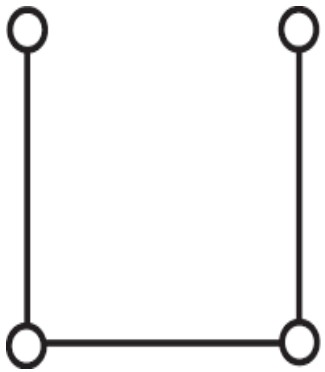
Spanning Tree: *Properties*

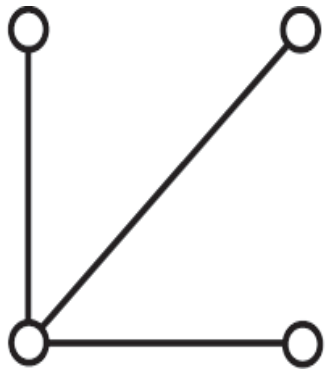
- **Detecting a cycle in an undirected connected graph**
  - DFS or BFS?
  - A connected undirected graph that has n vertices must have at least $n - 1$ edges
  - A connected undirected graph that has n vertices and exactly $n - 1$ edges cannot contain a cycle
  - A connected undirected graph that has n vertices and more than $n - 1$ edges must contain at least one cycle

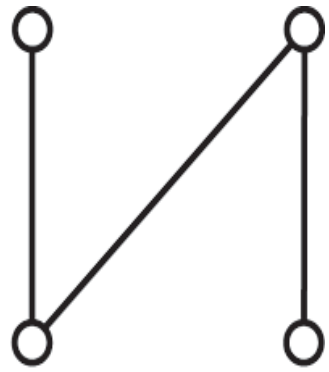
Practice 4: Number of *Spanning Trees*

- **How many different spanning trees?**
  - Two graphs G and H are isomorphic if and only if there is a bijection *f* between their vertex sets
    - ***For any vertices x and y in G, they are adjacent in G iff f(x) and f(y) are adjacent in H***





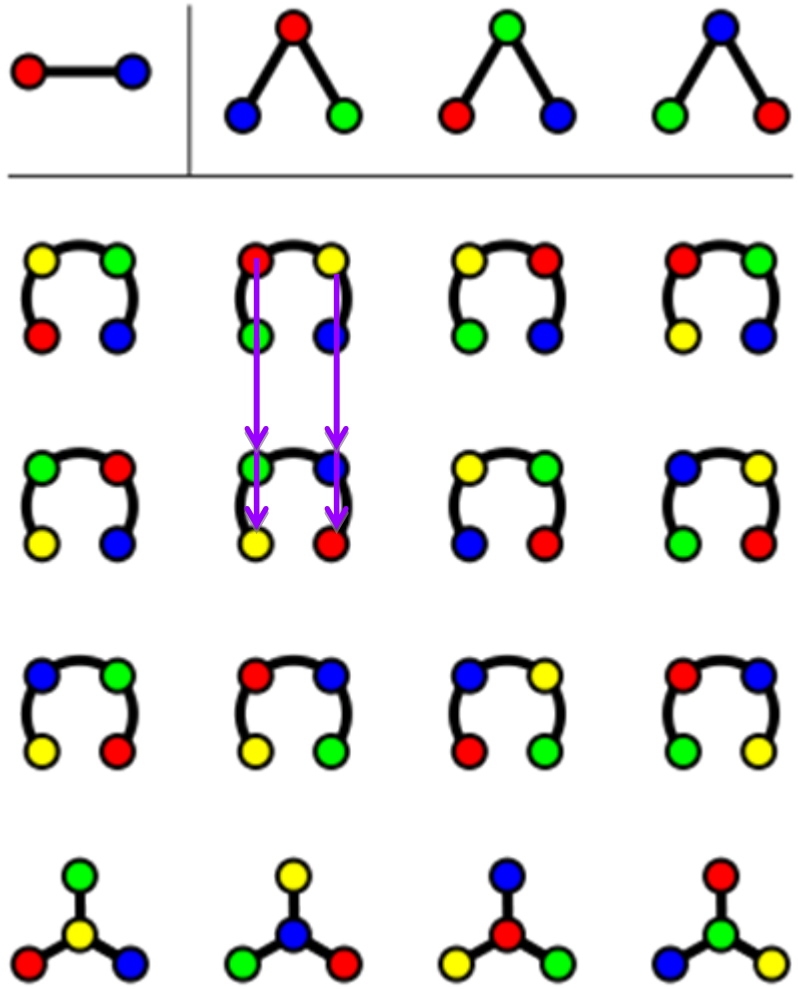




Counting Spanning Trees

Various vertex labeling: n^{n-2}

- **2 nodes ➔ $2^{2-2}=1$**
- **3 nodes ➔ $3^{3-2}=3$**
- **4 nodes ➔ $4^{4-2}=16$**
- **Why n^{n-2}?**
  - One proof is based on Prüfer sequence
- **Graph isomorphism**
  - One of the NP problems



Wikipedia ©

Counting Spanning Trees

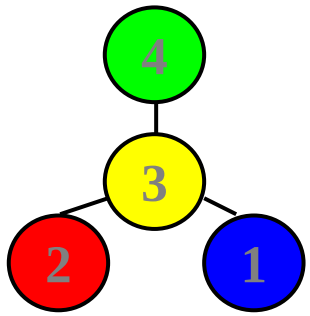
- **Prüfer sequence** **[Heinz Prüfer, 1918]**

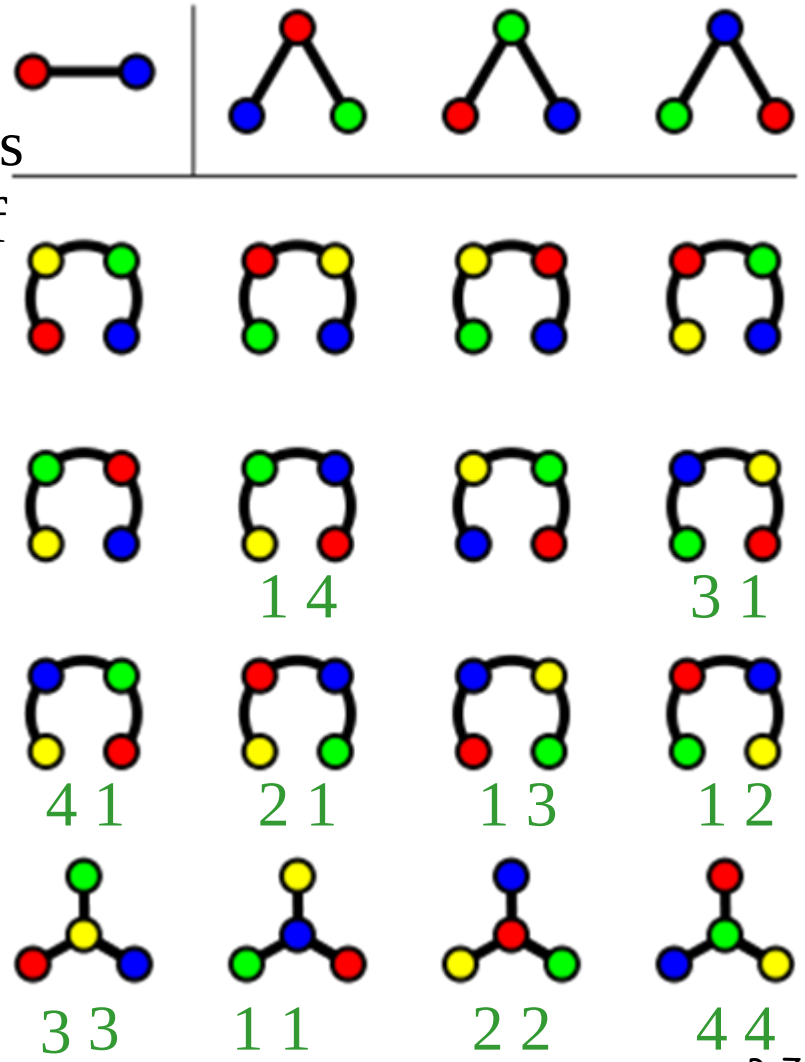
1. Each labeled tree with n vertices has a unique Prüfer sequence of length n-2

– Conversion algorithms

- **Leaf with the smallest label**
- **Keep the label of its *parent***

2. Each Prüfer sequence of length n-2 has a unique labeled tree with n vertices

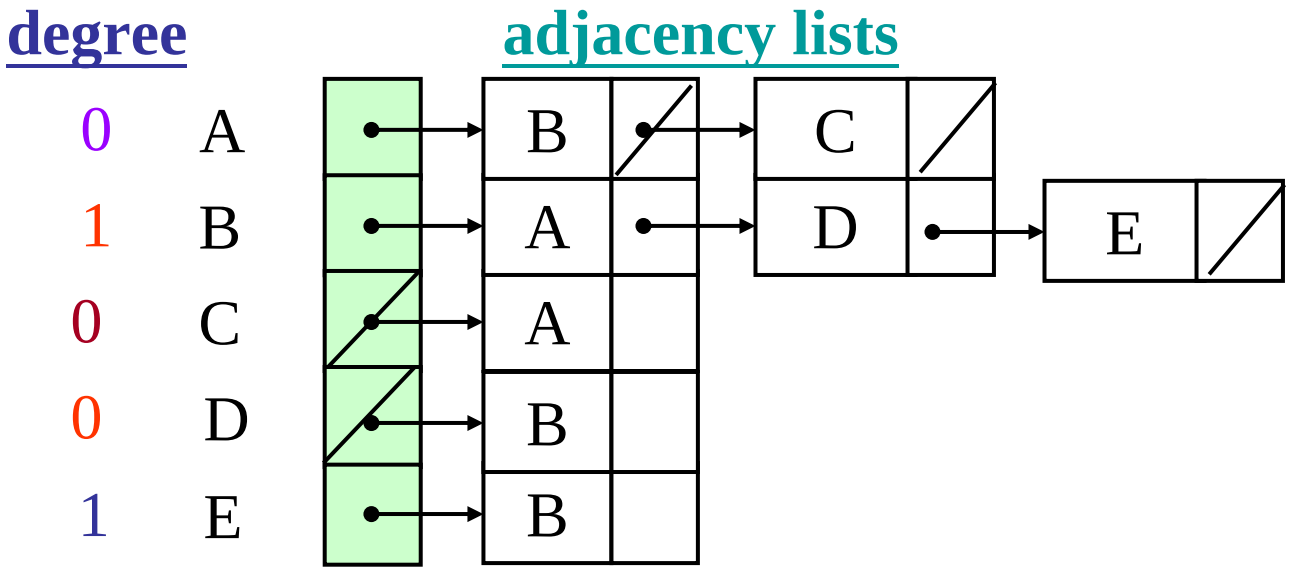


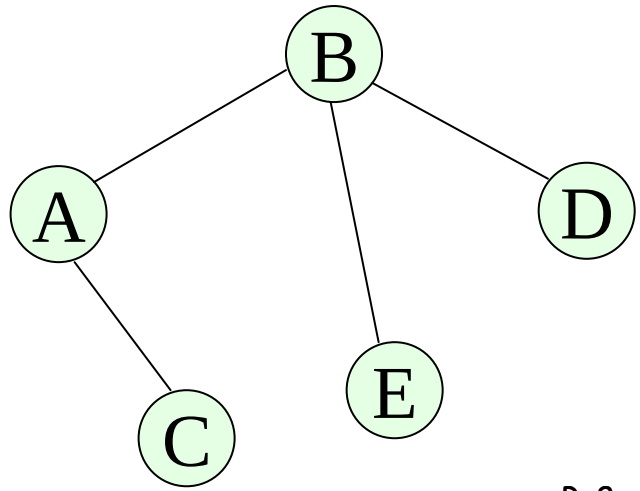


Wikipedia ©

Prüfer Sequence

Prüfer sequence: A B B



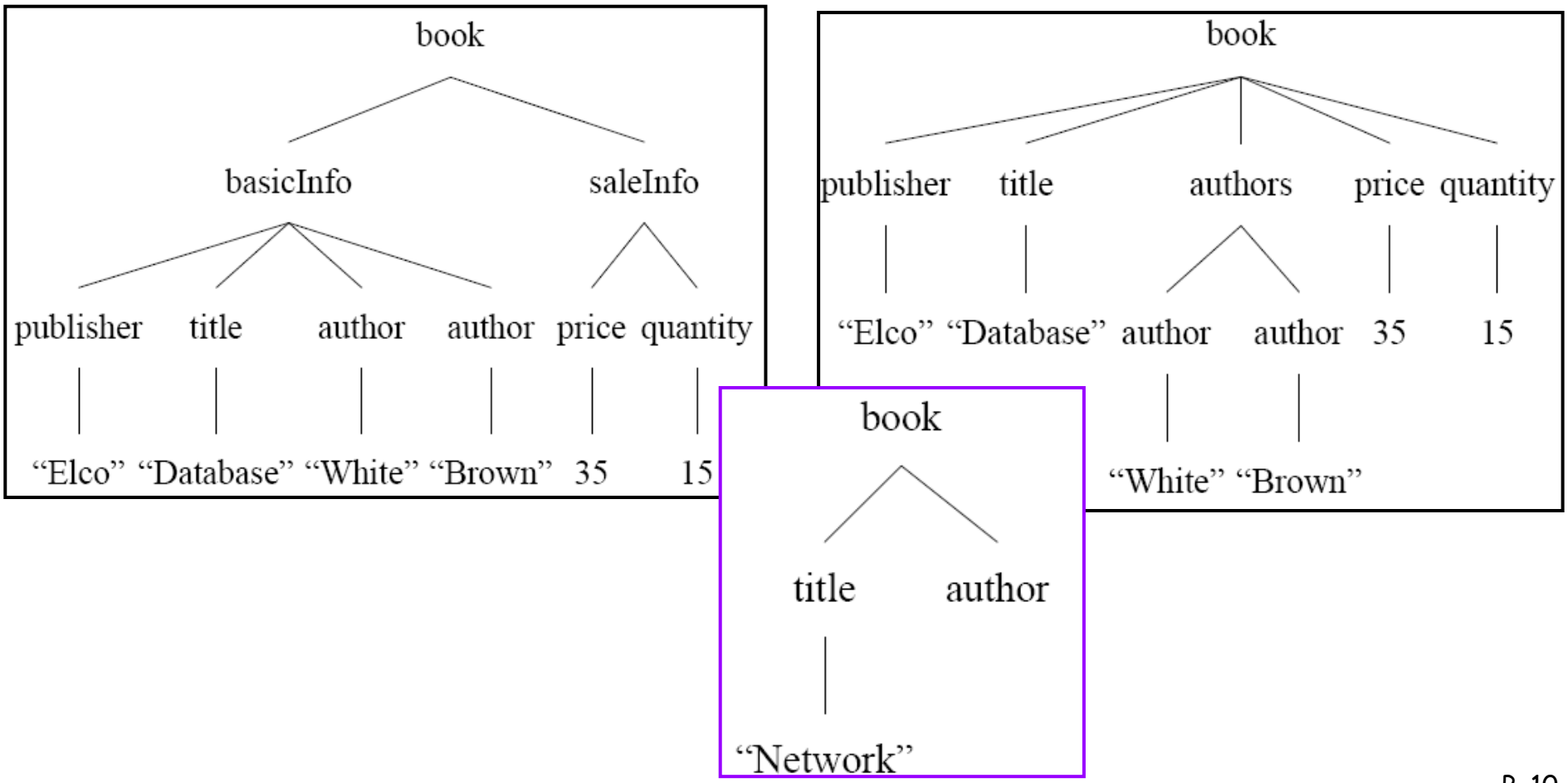


Prüfer Sequence

- **Rationale behind the proof**
  - For n labels, there are n^{n-2} sequences of length $n-2$
  - Each Prüfer sequence can be converted to a unique labeled tree with n vertices (*a spanning tree*)
  - For a *complete* graph with n vertices, there are n^{n-2} spanning trees
- **Other applications**
  - Compact tree representation ➔ Prüfer code
  - XML structural matching ➔ twig query

Prüfer Sequence

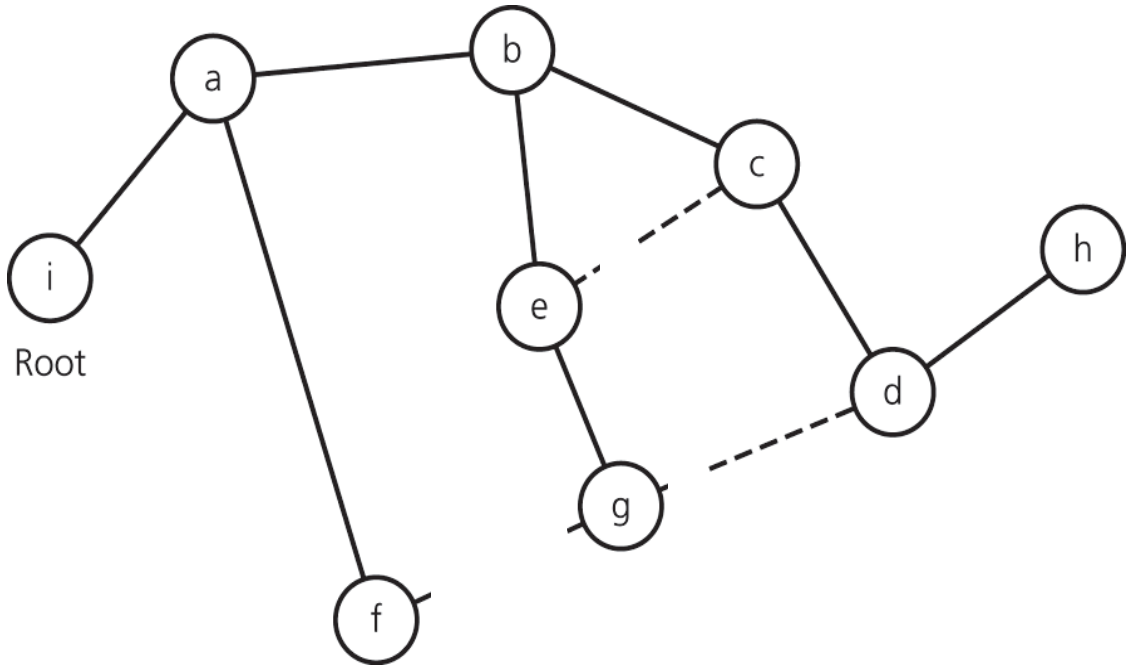
- An example of twig query on XML documents

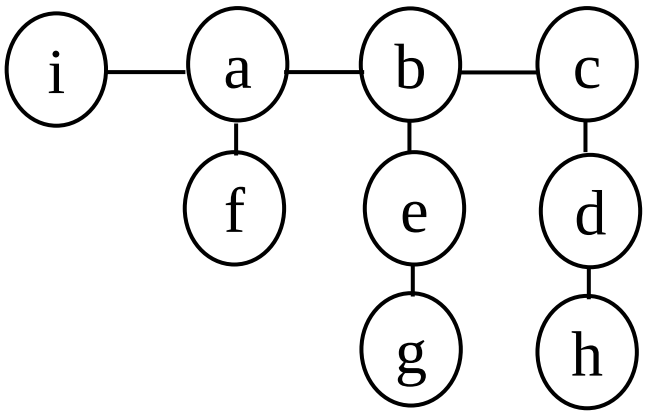


Practice 5: Tree ➔ *Prüfer Sequence*

- **Prüfer sequence?**

a e b d c b a





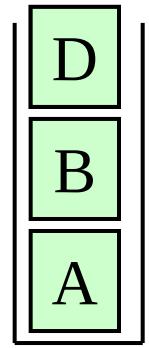
DFS / *BFS* for Spanning Trees

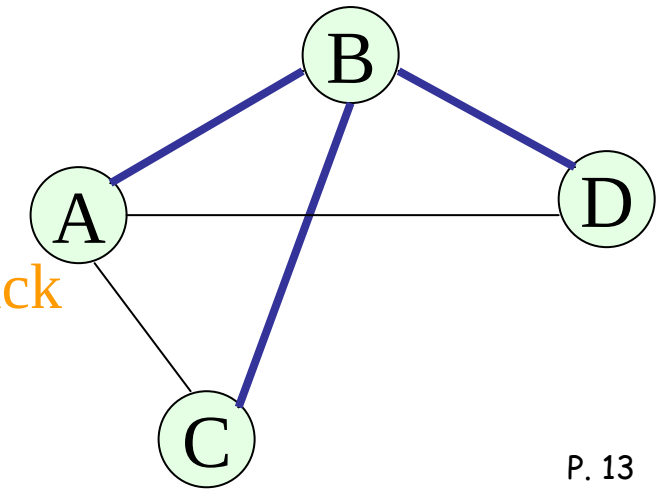
- **To create a spanning tree**
  - Traverse the graph using either depth-first search (DFS) or breadth-first search (BFS) and mark the edges that you follow
  - After the traversal is completed, the graph's vertices and marked edges form a spanning tree

DFS in *iterative* form (stack)

iterativeDFS(Vertex v) **DFS traversal sequence**: **AB** BC B**D**

```
s.createStack();      count=0;
s.push(v);
Mark v as visited;
while (!s.isEmpty() && count < |V|-1)
{     u = s.getTop();                    // at top of the stack
      if (unvisited vertex w is adjacent to u)
      {       s.push(w);      count++;
              Mark w as visited;      // (u,w)
      }
      else    s.pop();                   // backtrack
}
```





DFS in *iterative* form (stack)

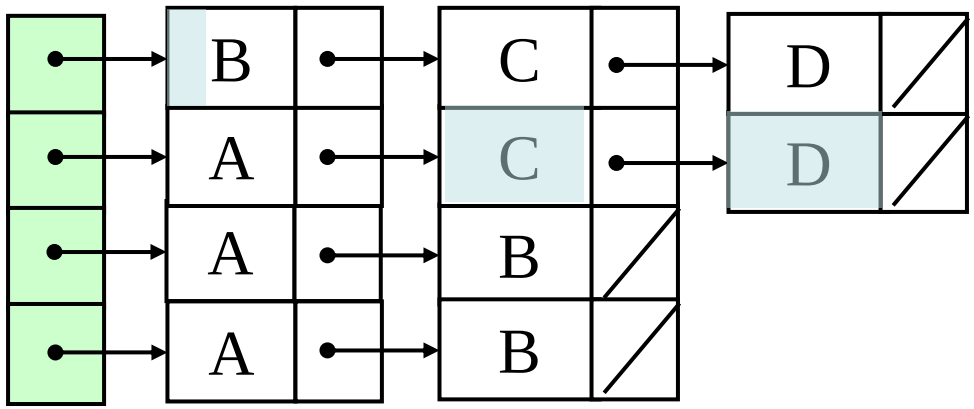
DFS traversal sequence: AB BC BD

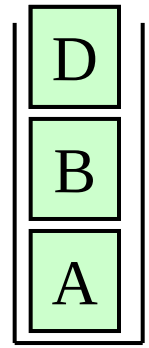
count = 3

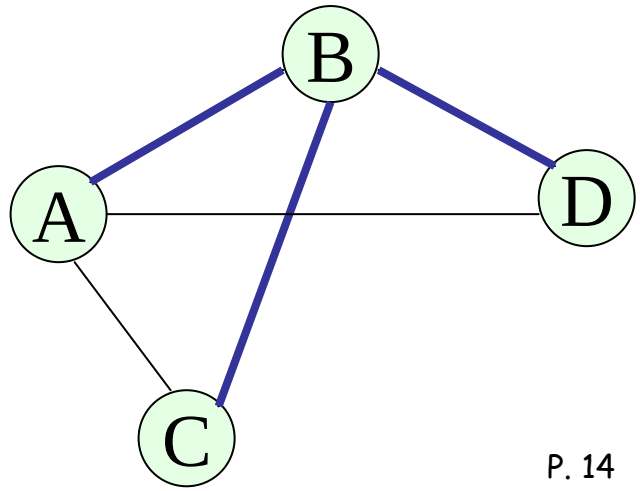
Visited

T	A
T	B
T	C
T	D

adjacency lists



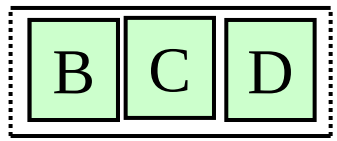


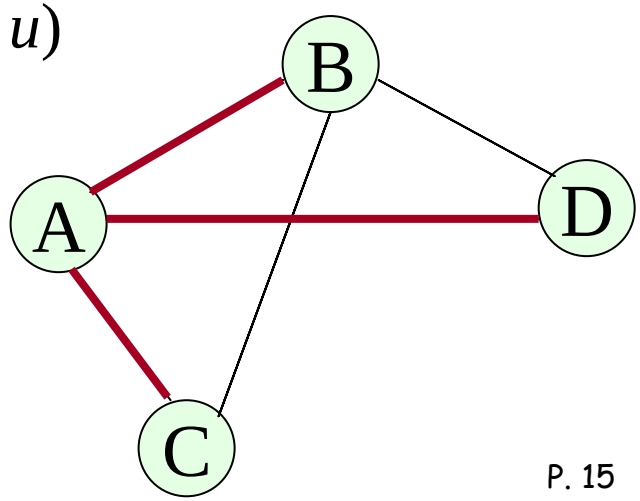


BFS in *iterative* form (queue)

BFS traversal sequence: **AB AC AD**

```
iterativeBFS(Vertex v)
    q.createQueue();     count=0;
    q.enqueue(v);
    Mark v as visited;
    while (!q.isEmpty() && count < |V|-1)
    {       q.dequeue(u);
            for (each unvisited vertex w adjacent to u)
            {       Mark w as visited;       // (u,w)
                    q.enqueue(w);            count++;
            }
    }
```





BFS in *iterative* form (queue)

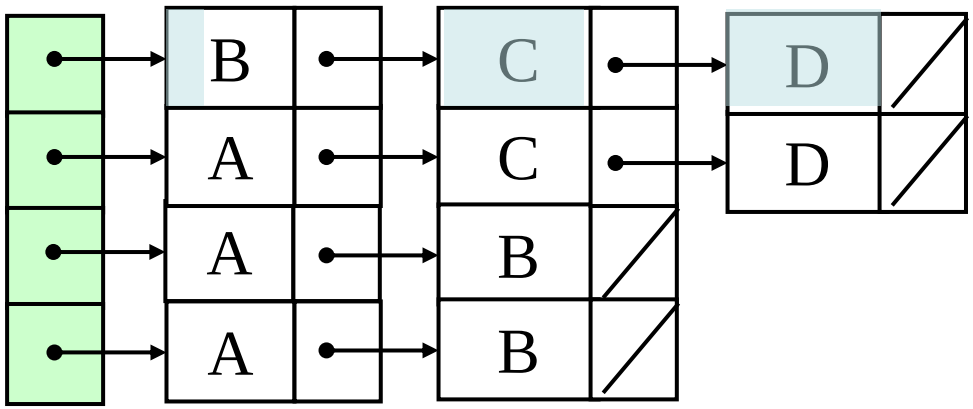
BFS traversal sequence: **AB AC AD**

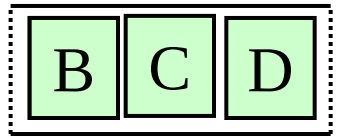
count = 3

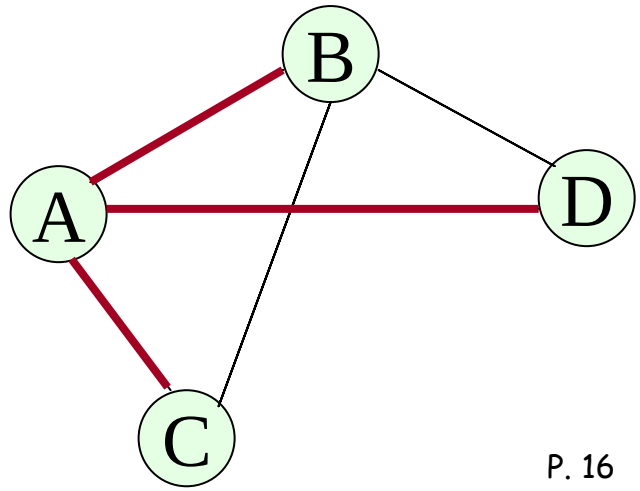
Visited

T	A
T	B
T	C
T	D

adjacency lists







Self-exercise 4

1. Consider the graph to answer the following:

(a) How many **spanning trees** does it have?

(b) Draw all the **spanning trees**

(c) Compute Prüfer sequence of each spanning tree