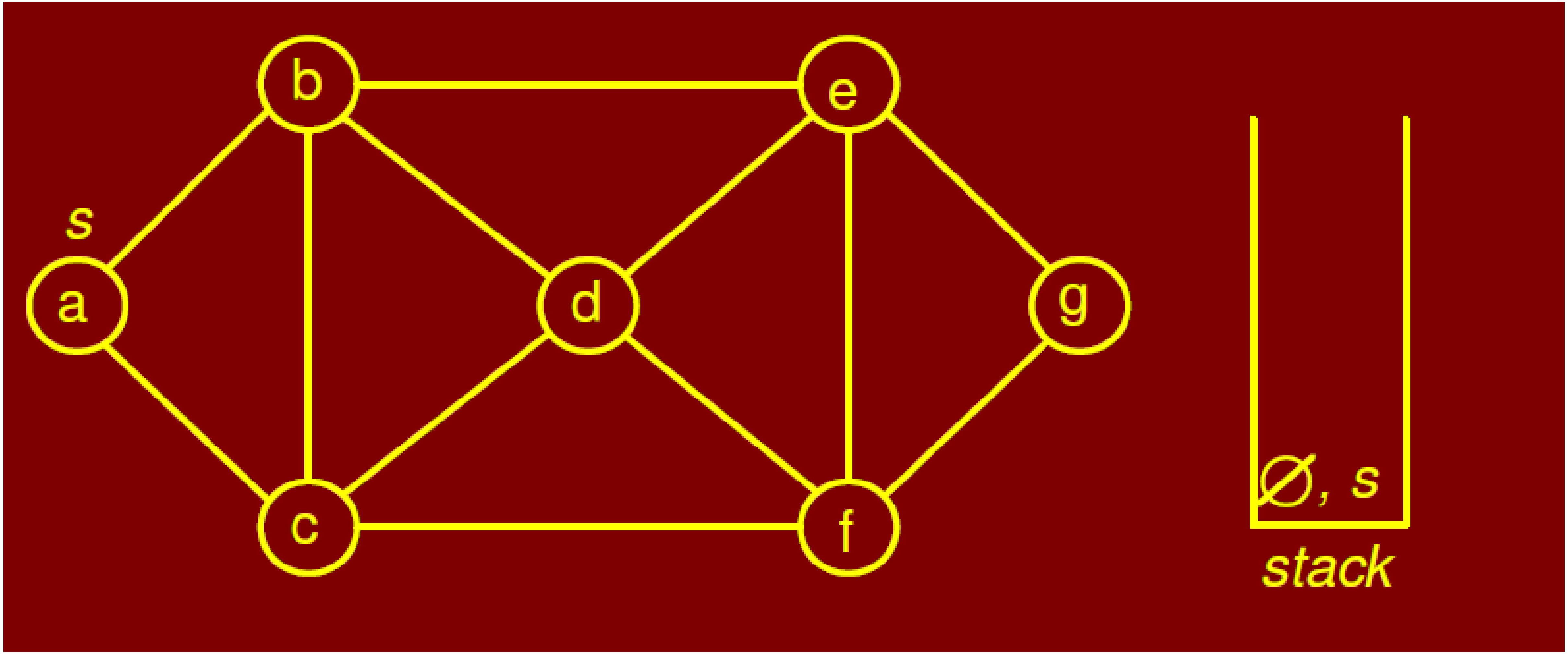
Graphs
Generic Traversal Algorithms

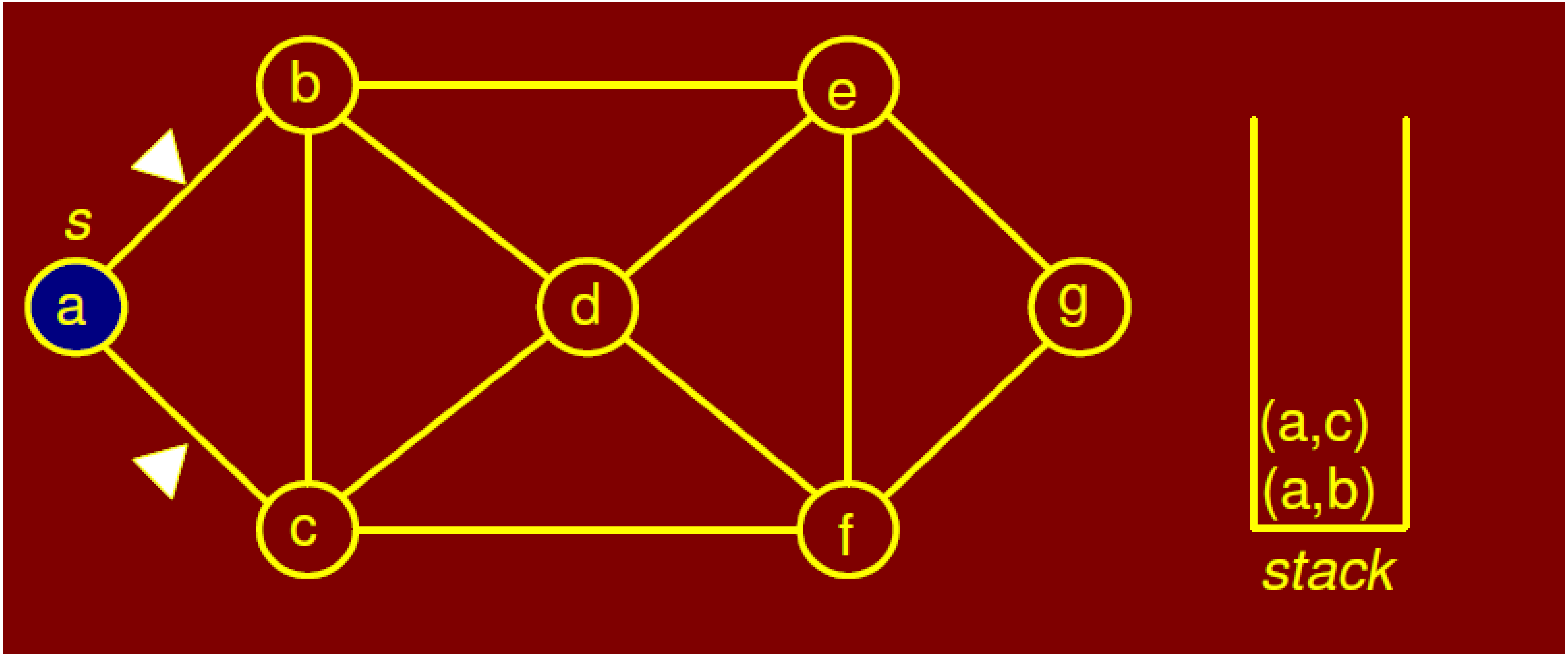
(Class 30)

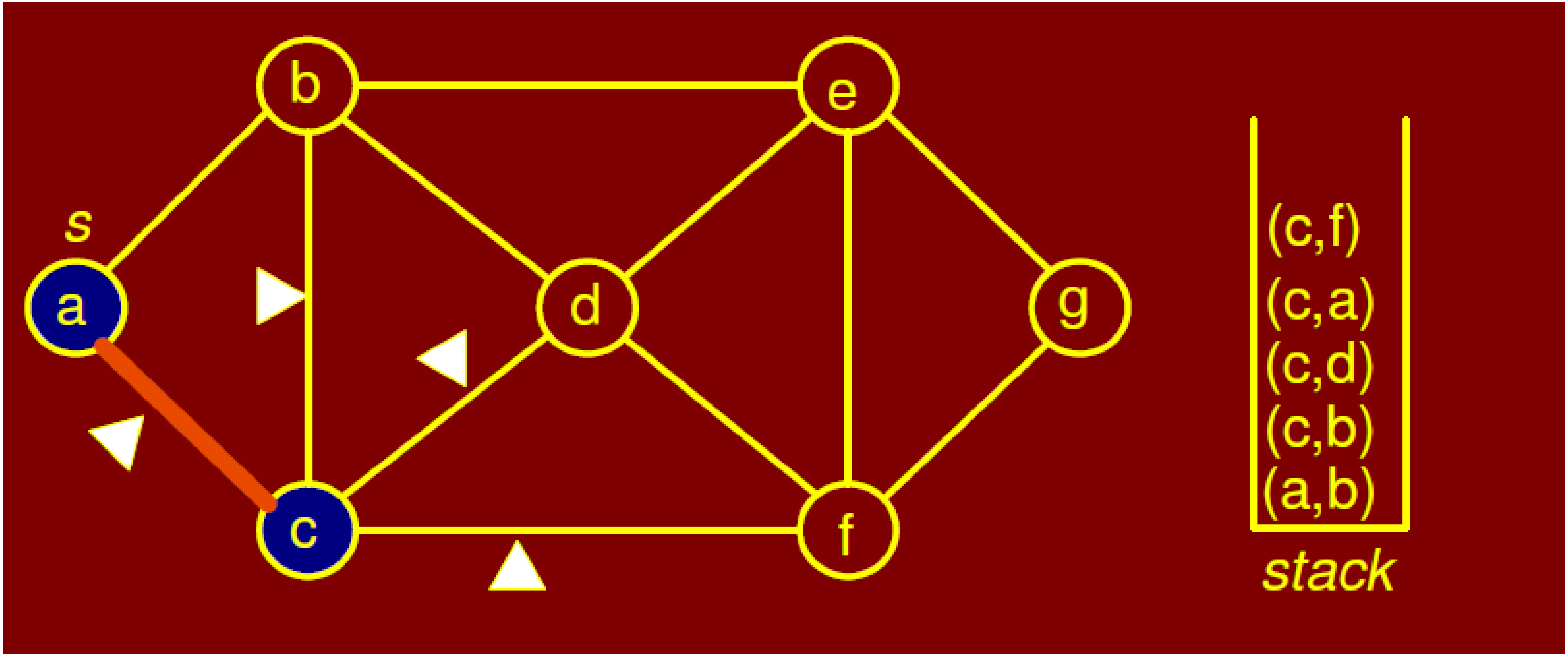
Depth-First Search using Generic Traversal Algorithm

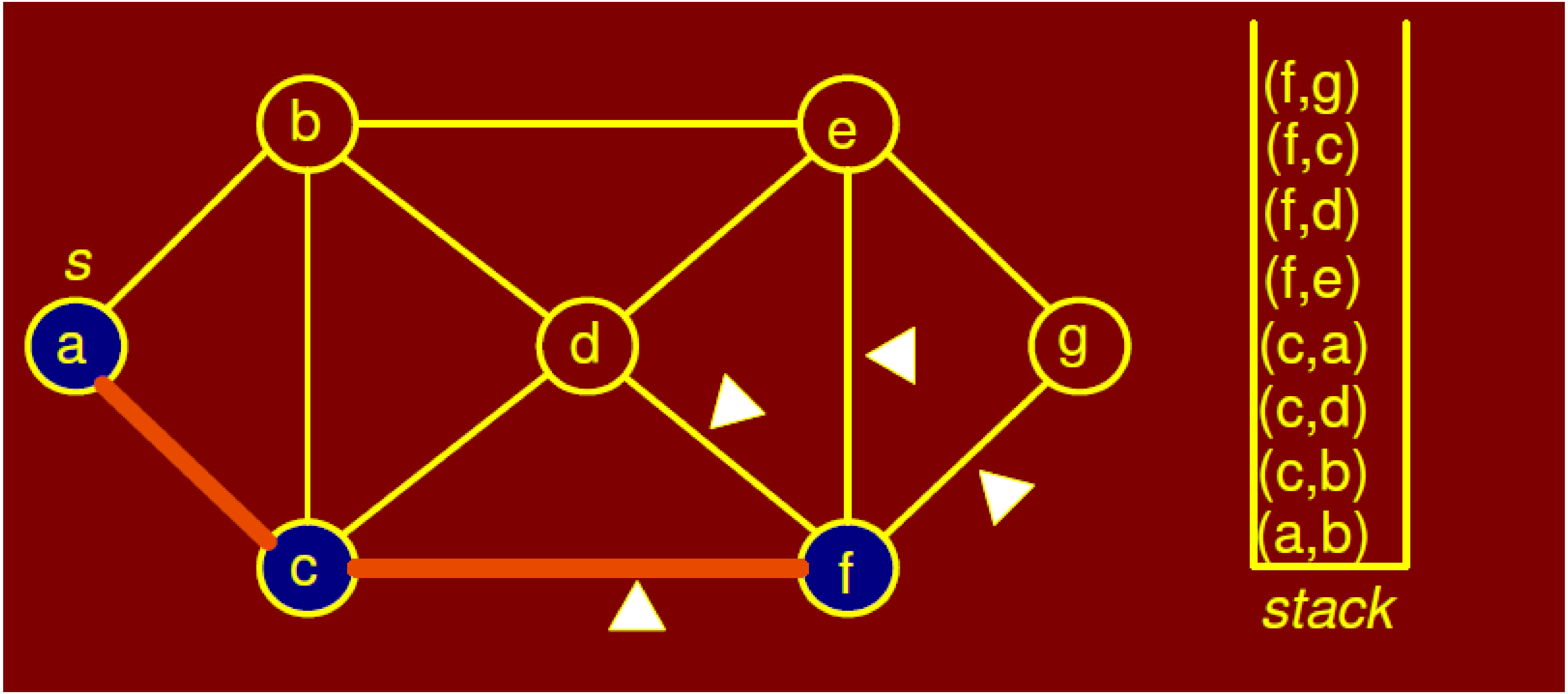
```
TRAVERSE(s)
push(∅, s)
while stack not empty
  pop(p, v)
    if (v is unmarked)
      mark v
      parent (v) ← p
      for each edge (v,w)
        push(v,w)
```

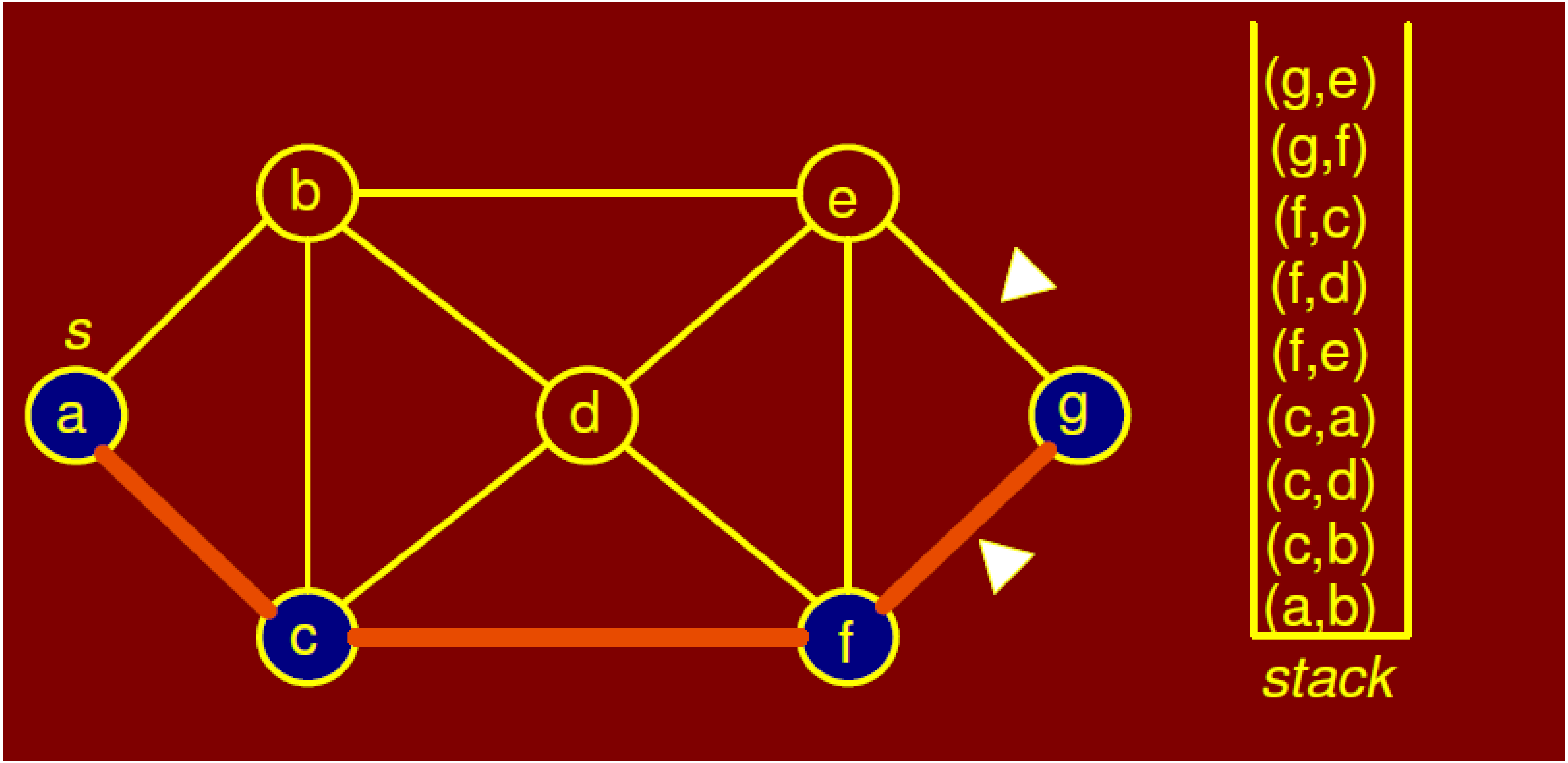
- The following figures show a trace of the DFS algorithm applied to a graph.
- The figures show the content of the stack during the execution of the algorithm.
- The DFS or BFS traversal generates a tree at the end.


s
a
b
c
d
e
f
g
∅, s
stack

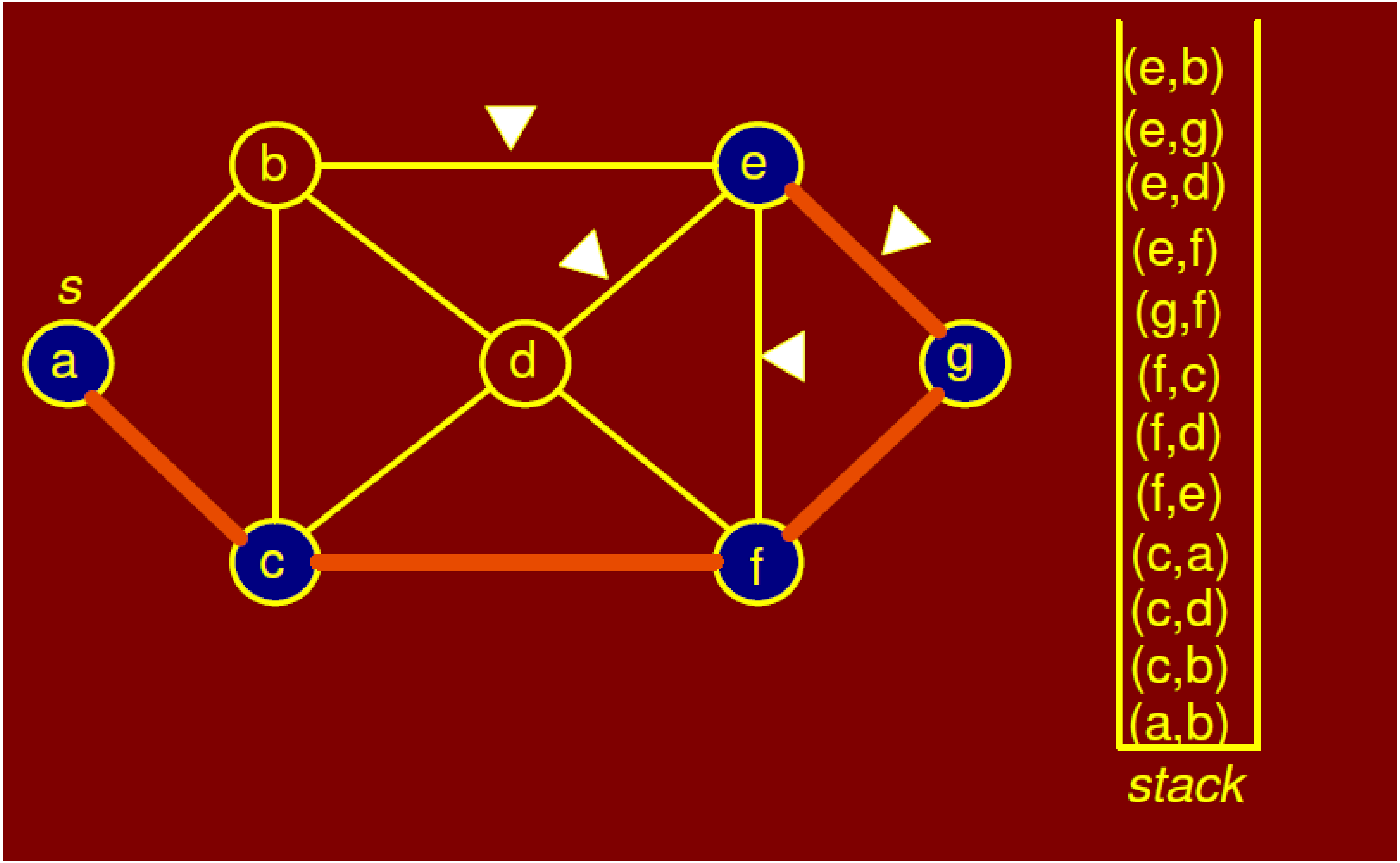

s
a
b
c
d
e
f
g
(a,c)
(a,b)
stack

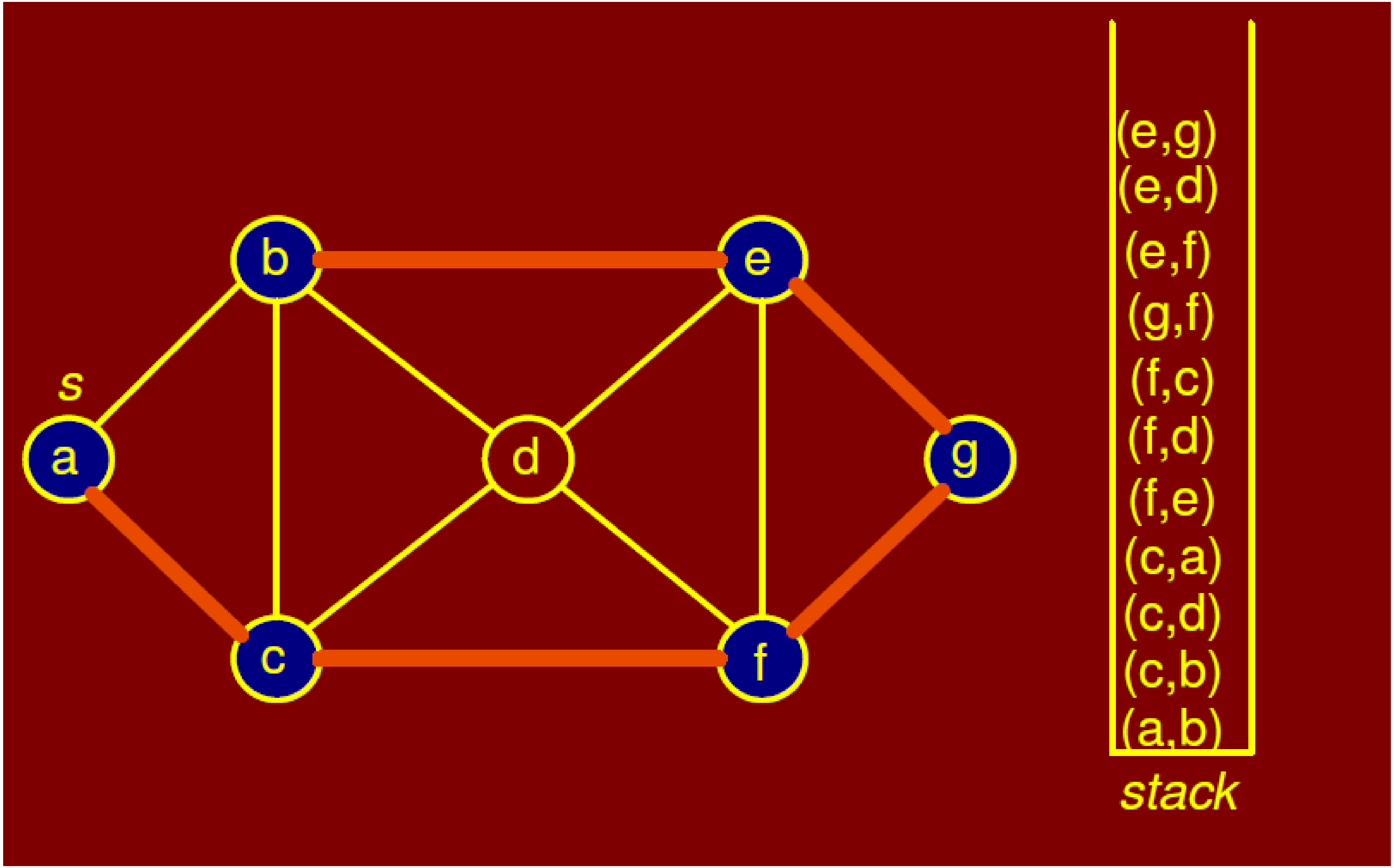

s
a
b
c
d
e
f
g
(c,f)
(c,a)
(c,d)
(c,b)
(a,b)
stack

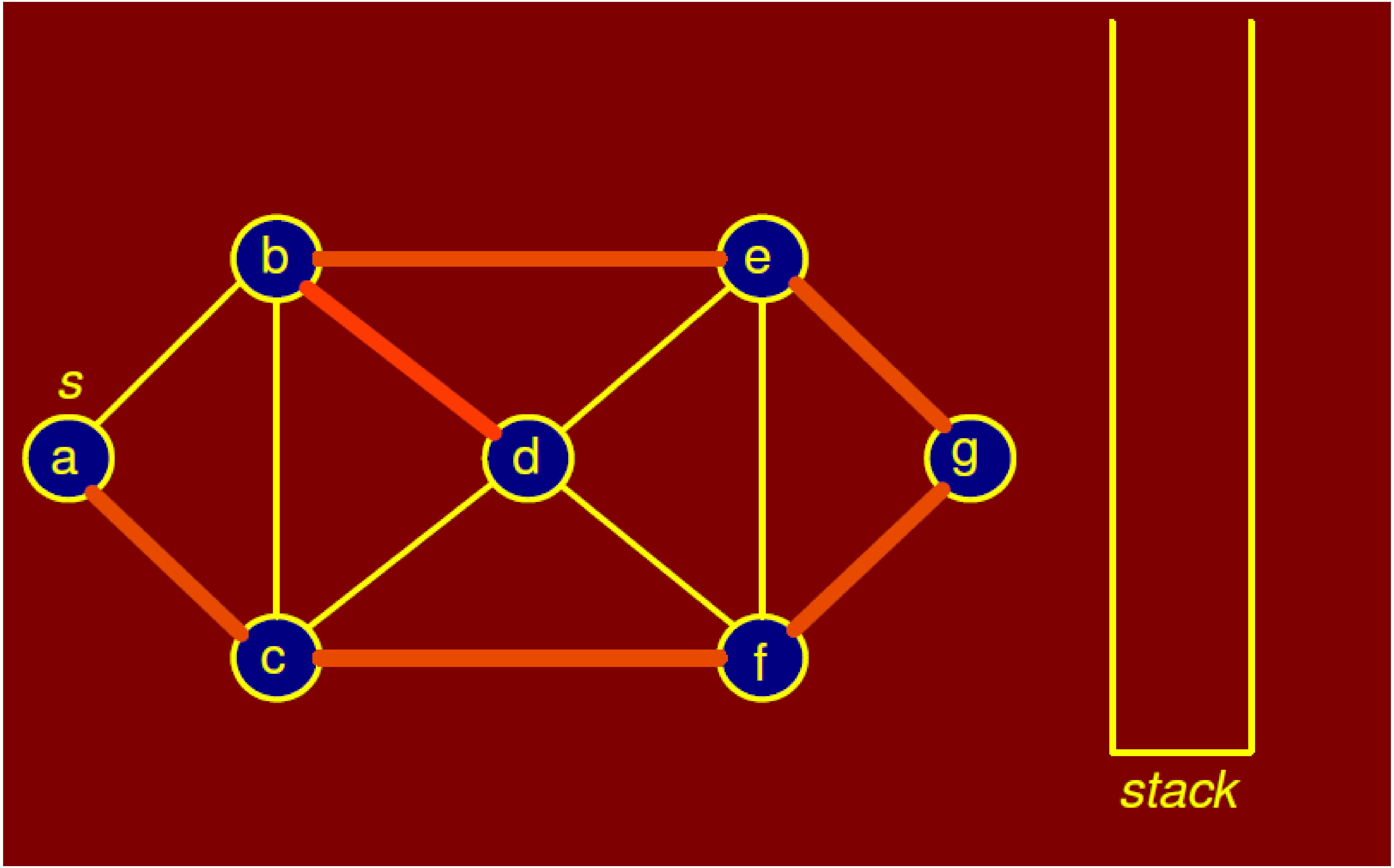

s
a
b
c
d
e
f
g
(f,g)
(f,c)
(f,d)
(f,e)
(c,a)
(c,d)
(c,b)
(a,b)
stack



s
a
b
c
d
e
f
g
(g,e)
(g,f)
(f,c)
(f,d)
(f,e)
(c,a)
(c,d)
(c,b)
(a,b)
stack


s
a
b
c
d
e
f
g
(e,b)
(e,g)
(e,d)
(e,f)
(g,f)
(f,c)
(f,d)
(f,e)
(c,a)
(c,d)
(c,b)
(a,b)
stack


s
a
b
c
d
e
f
g
(e,g)
(e,d)
(e,f)
(g,f)
(f,c)
(f,d)
(f,e)
(c,a)
(c,d)
(c,b)
(a,b)
stack


s
a
b
c
d
e
f
g
stack

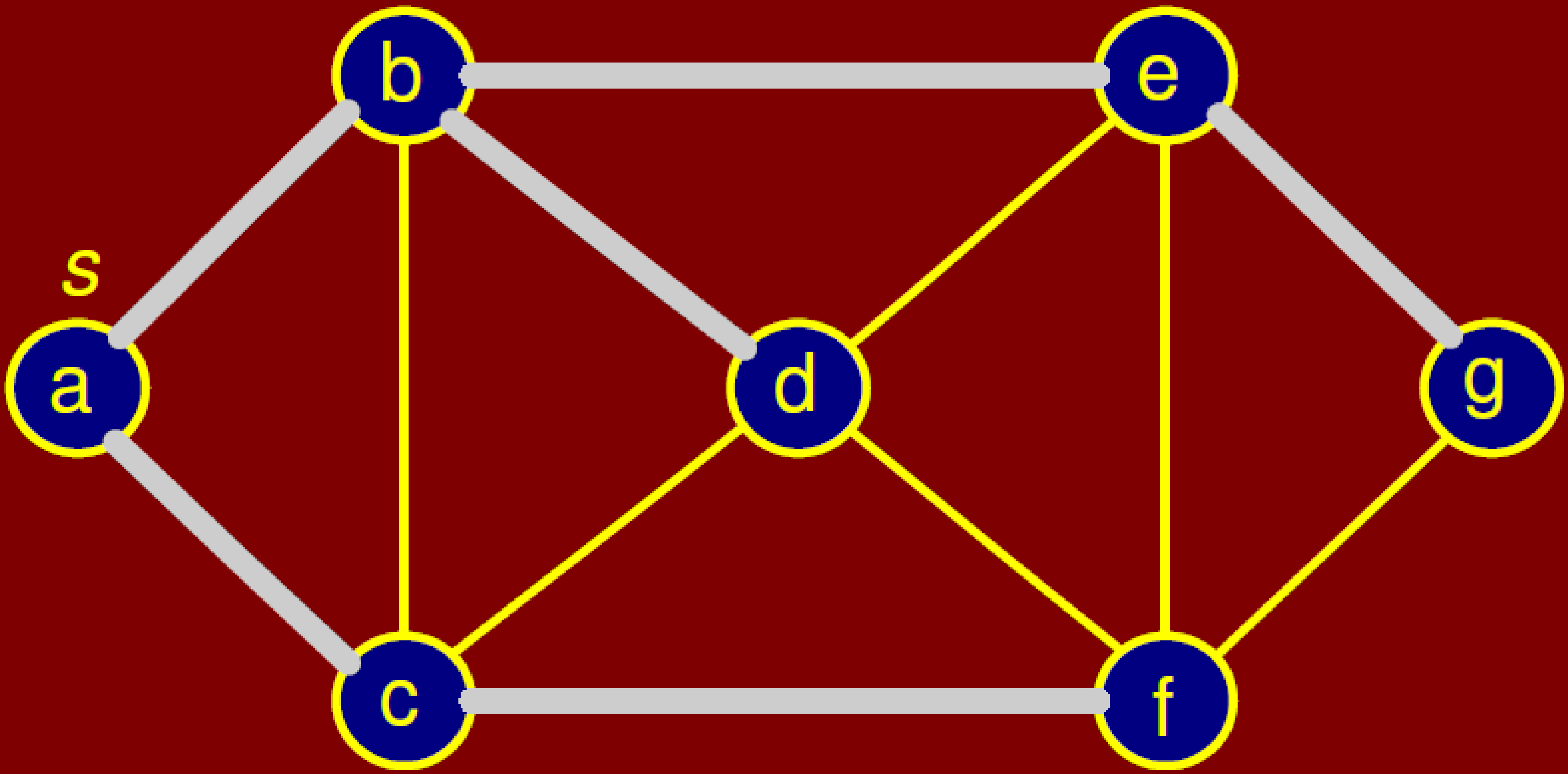
Running Time Analysis

- Each execution of line 3 or line 8 in the TRAVERSE-DFS algorithm takes constant time.
- So, the overall running time is $O(V + E)$.
- Since the graph is connected, $V \leq E + 1$, this is $O(E)$.

Breadth-First Search using Generic Traversal Algorithm

- If we implement the bag by using a *queue*, we have breadth-first search (BFS).
- Each execution of line 3 or line 8 still takes constant time.
- So overall running time will be still $O(E)$.

```
TRAVERSE(s)
enqueue(∅, s)
while queue not empty
  do dequeue(p, v)
  if (v is unmarked)
    mark v
    parent (v) ← p
    for each edge (v,w)
      enqueue(v,w)
```



BFS Tree

- If the graph is represented using an *adjacency matrix*, the finding of all the neighbors of vertex in line 7 takes $O(V)$ time.
- Thus depth-first and breadth-first take $O(V^2)$ time overall.
- Either DFS or BFS yields a spanning tree of the graph.
- The tree visits every vertex in the graph.

- A spanning tree is a tree-like subgraph of a connected, undirected graph that includes all of the vertices of the original graph but contains only a subset of its edges.
- In other words, it is a subgraph that is both connected and acyclic, and it includes all the vertices of the original graph.
- There are many different algorithms for constructing spanning trees, such as Kruskal's algorithm and Prim's algorithm.

- These algorithms are often used in network design and optimization problems, where the goal is to create a connected network with the fewest possible edges.
- Spanning trees are also useful in other areas of computer science, such as graph theory and algorithms.

- The DFS or BFS traversal generates a tree at the end.
- This fact is established by the following lemma:
- **Lemma:**
  - The generic TRAVERSE(S) marks every vertex in any connected graph exactly once and the set of edges $(v, parent(v))$ with $parent(v) \neq \emptyset$ form a spanning tree of the graph.

- **Proof:**
- First, it should be obvious that no vertex is marked more than once.
- Clearly, the algorithm marks s.
- Let $v \neq s$ be a vertex and let $s \to \cdots \to u \to v$ be a path from s to v with the minimum number of edges.

- Since the graph is connected, such a path always exists.
- If the algorithm marks u, then it must put (u, v) into the bag, so it must take (u, v) out of the bag at which point v must be marked.
- Thus, by induction on the shortest-path distance from s, the algorithm marks every vertex in the graph.

- Call an edge $(v, parent(v))$ with $parent(v) \neq \emptyset$, a *parent* edge.
- For any node v, the path of parent edges $v \rightarrow parent(v) \rightarrow parent\big(parent(v)\big)$ … eventually leads back to s.
- So, the set of parent edges form a connected graph.
- Clearly, both end points of every parent edge are marked, and the number of edges is exactly one less than the number of vertices.
- Thus, the parent edges form a spanning tree.