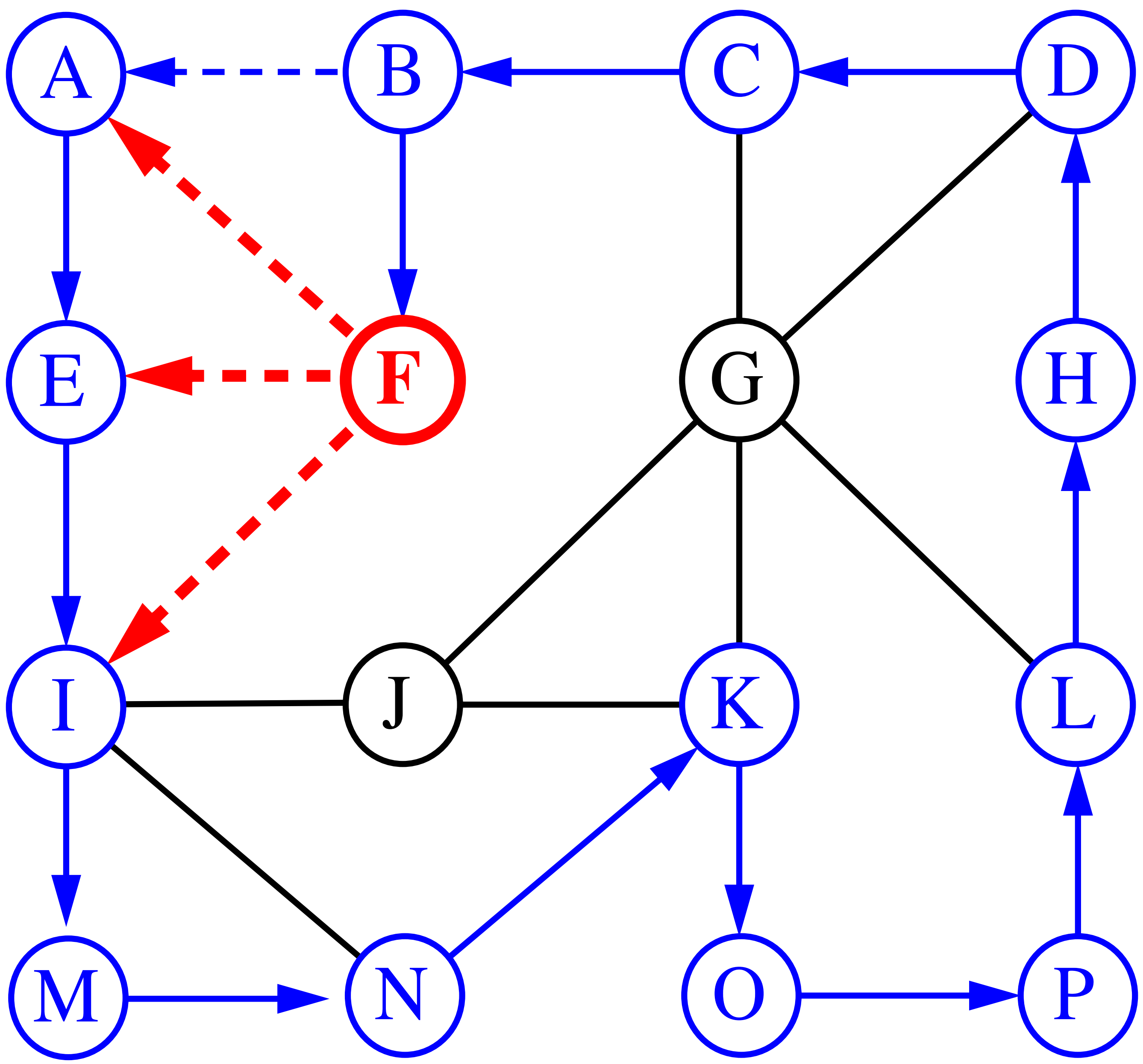
Winter 2016
COMP-250: Introduction to Computer Science

Lecture 18, March 17, 2016

DEPTH-FIRST SEARCH

- Graph Traversals
- Depth-First Search



Exploring a Labyrinth Without Getting Lost

- A **depth-first search (DFS)** in an undirected graph G is like wandering in a labyrinth with a string and a can of red paint without getting lost.
- We start at vertex s, tying the end of our string to the point and painting s "visited". Next we label s as our current vertex called u.
- Now we travel along an arbitrary edge (u,v).
- If edge (u,v) leads us to an already visited vertex v we return to u.

Exploring a Labyrinth Without Getting Lost

- If vertex v is unvisited, we unroll our string and move to v, paint v “visited”, set v as our current vertex, and repeat the previous steps.
- Eventually, we will get to a point where all incident edges on u lead to visited vertices. We then backtrack by unrolling our string to a previously visited vertex v. Then v becomes our current vertex and we repeat the previous steps.

Exploring a Labyrinth Without Getting Lost (cont.)

- Then, if all incident edges on v lead to visited vertices, we backtrack as we did before. We continue to backtrack along the path we have traveled, finding and exploring unexplored edges, and repeating the procedure.
- When we backtrack to vertex *s* and there are no more unexplored edges incident on *s*, we have finished our DFS search.

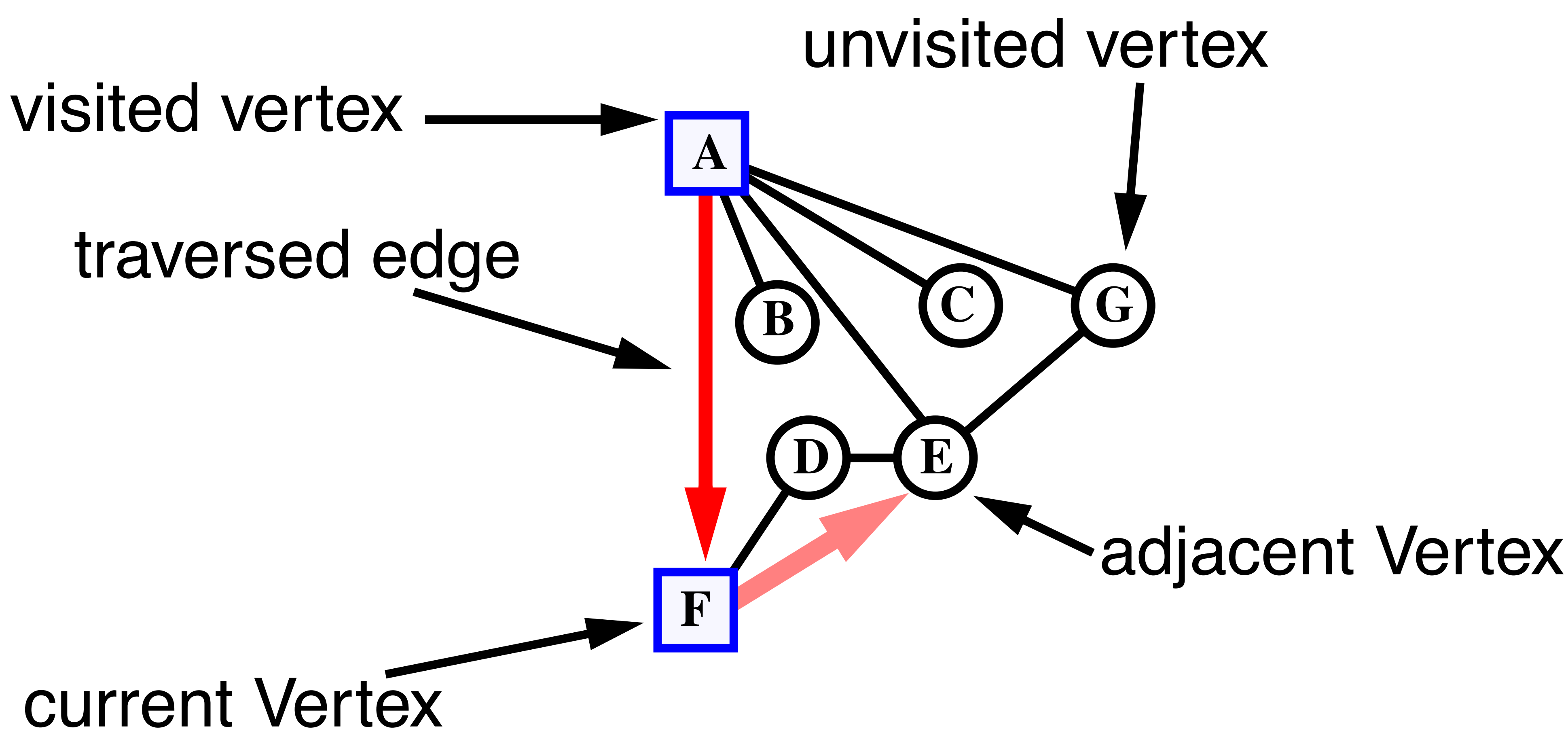
Depth-First Search

Algorithm DFS(v);
Input: A vertex v in a graph
Output: A labeling of the edges as "discovery" edges and "backedges"

for each edge e incident on v **do**
  if edge e is unexplored **then**
    let w be the other endpoint of e
    if vertex w is unexplored **then**
      label e as a discovery edge
      recursively call **DFS**(w)
    else
    label e as a backedge

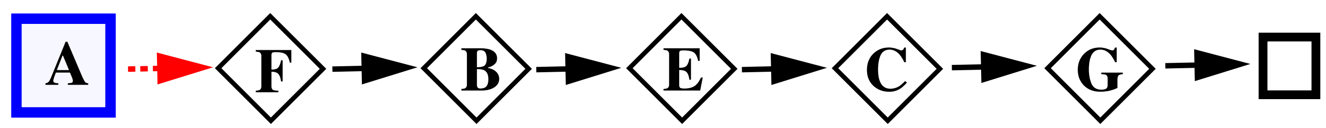
Depth-First Search

Algorithm DFS(v);
  Input: A vertex v in a graph
  Output: A labeling of the edges as "discovery" edges
    and "backedges"
  for each edge e incident on v **do**
    if edge e is unexplored **then**
      let w be the other endpoint of e
      if vertex w is unexplored **then**
        label e as a discovery edge
        recursively call **DFS**(w)
      else
        label e as a backedge



Determining Incident Edges

- DFS depends on how you obtain the incident edges.
- If we start at A and we examine the edge to F, then to B, then E, C, and finally G

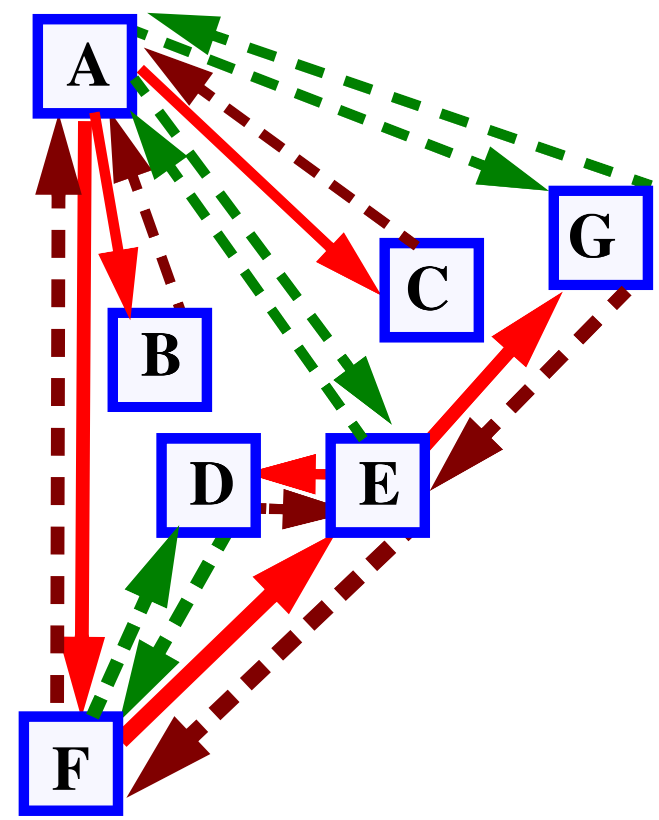


The resulting graph is:

discoveryEdge

backEdge

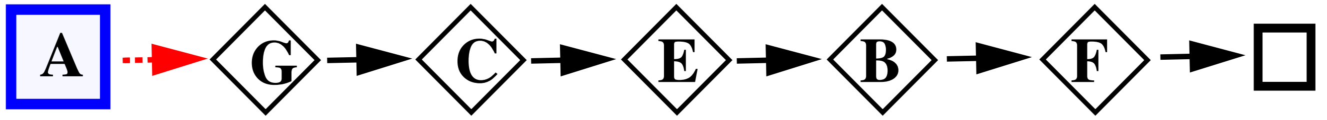
return from dead end



Determining Incident Edges

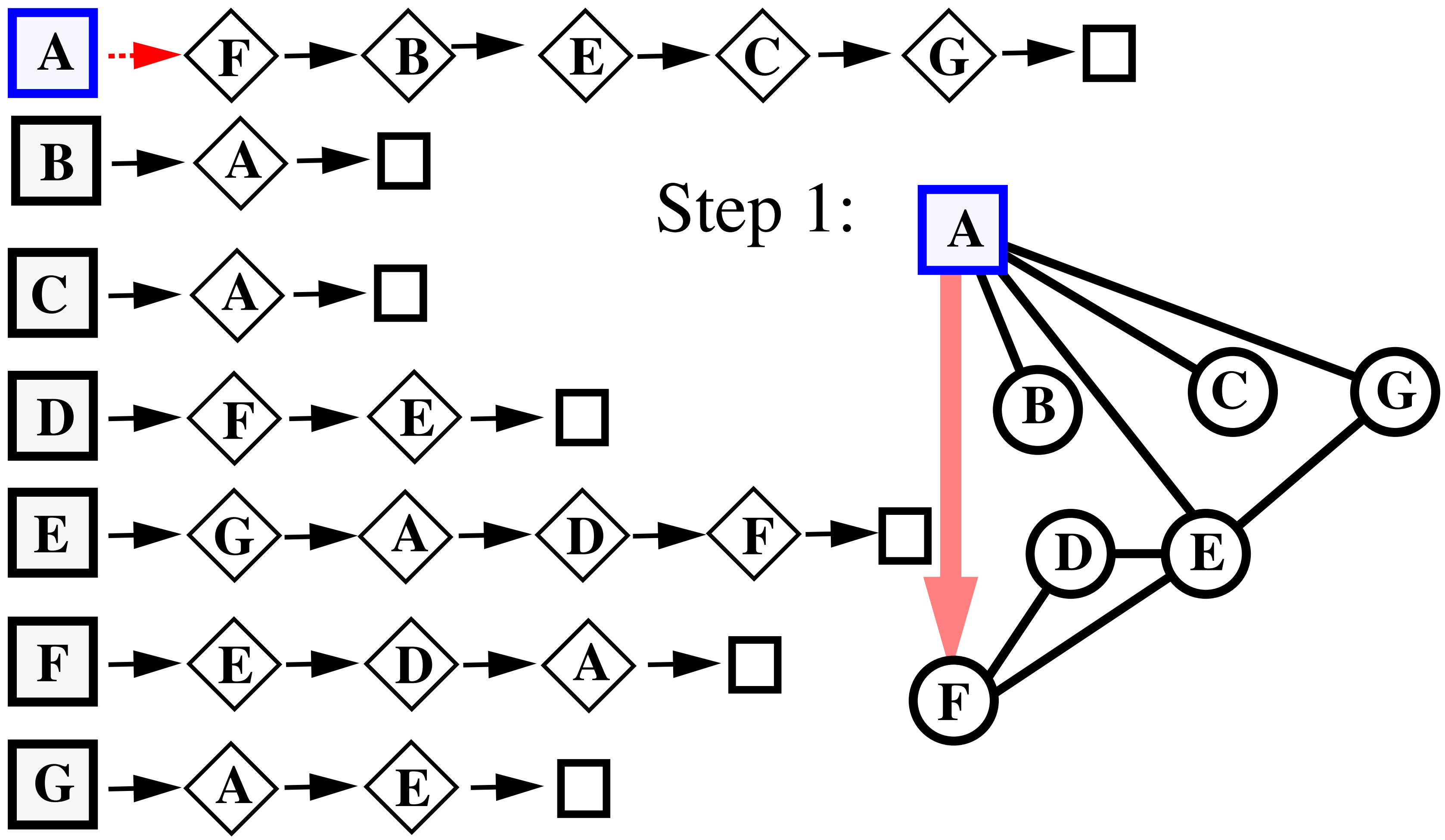
- DFS depends on how you obtain the incident edges.

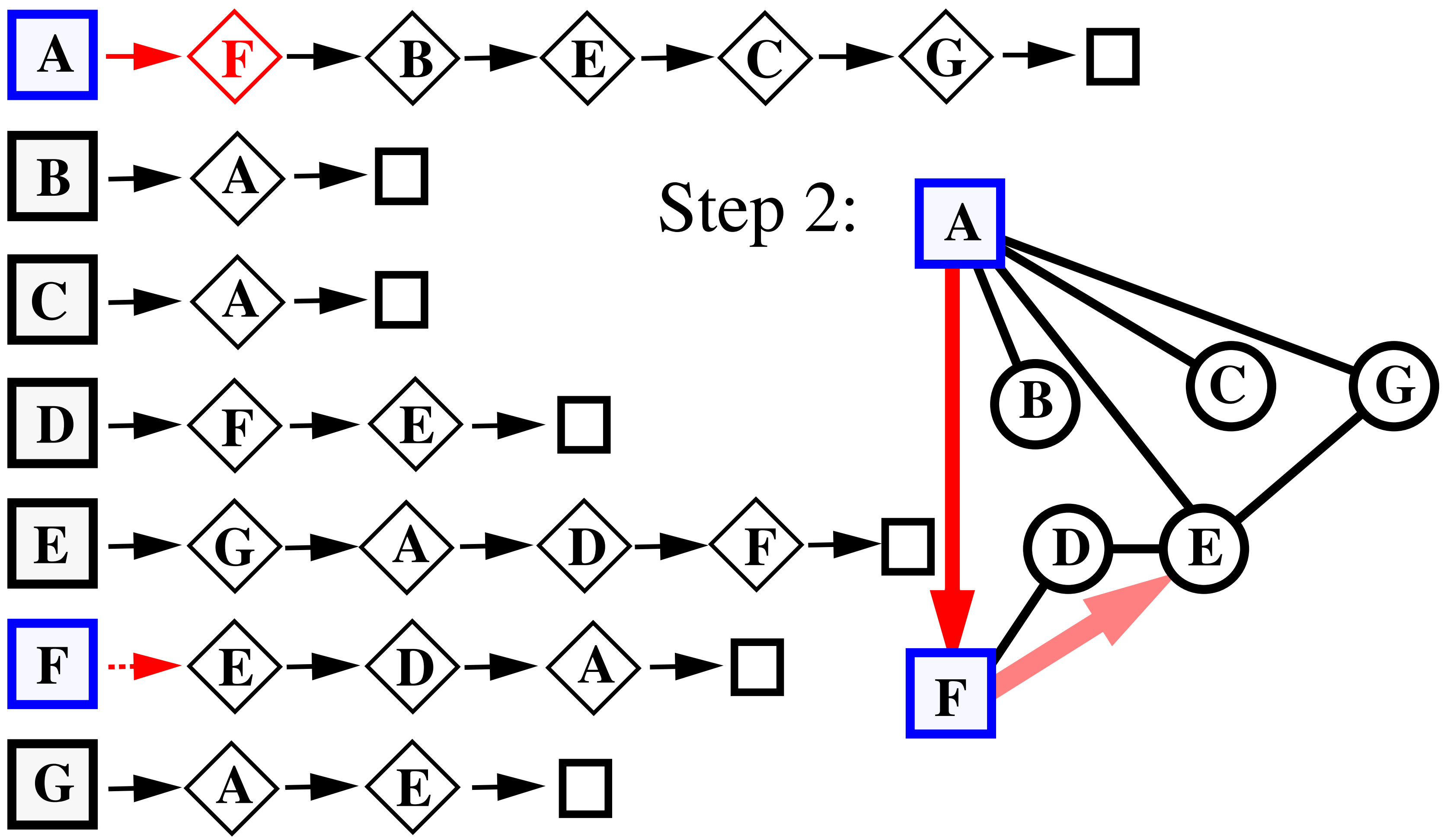
  If we instead examine the tree starting at A and looking at G, the C, then E, B, and finally F,

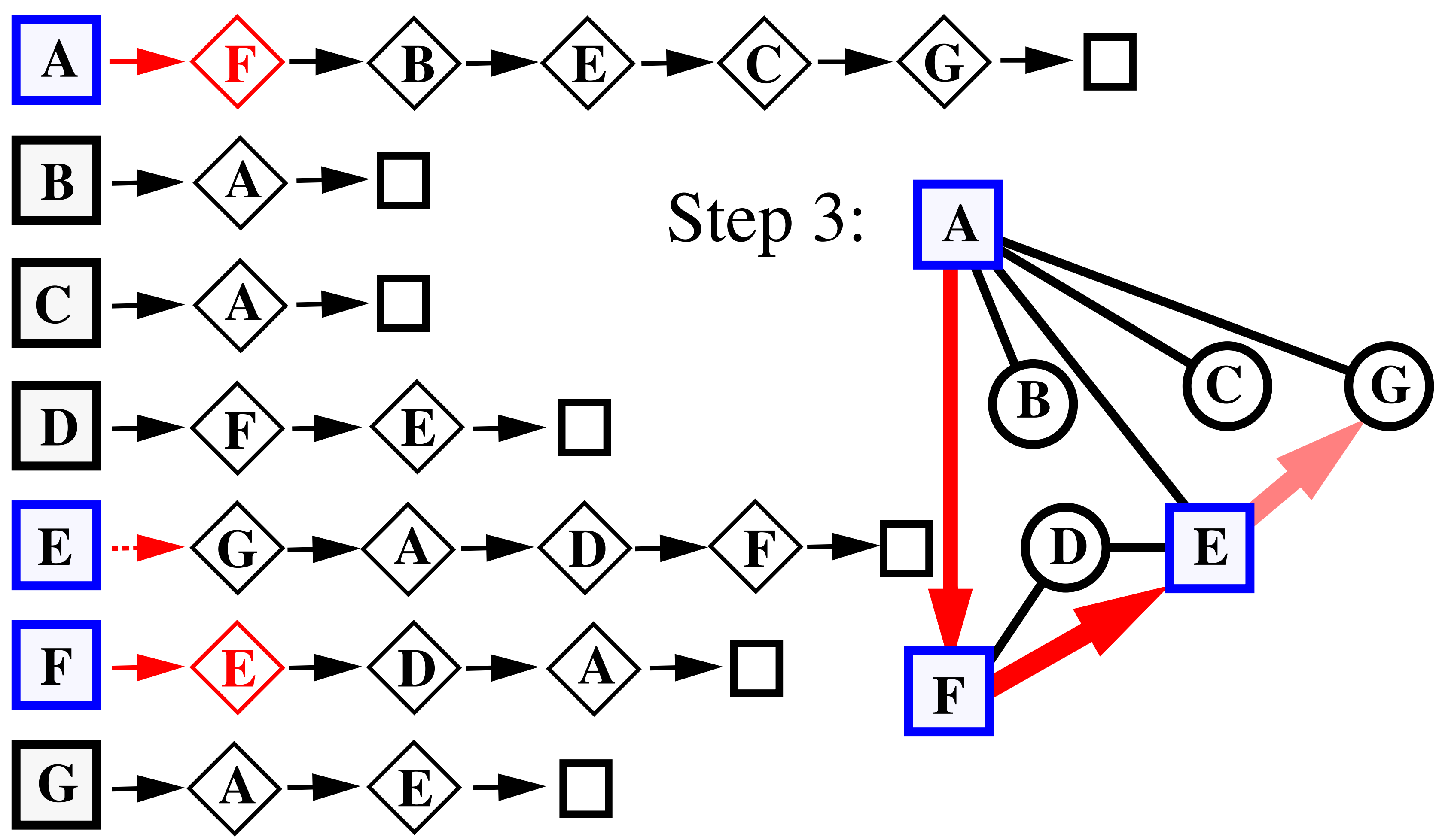
  

  the resulting set of backEdges, discoveryEdges and recursion points is different.

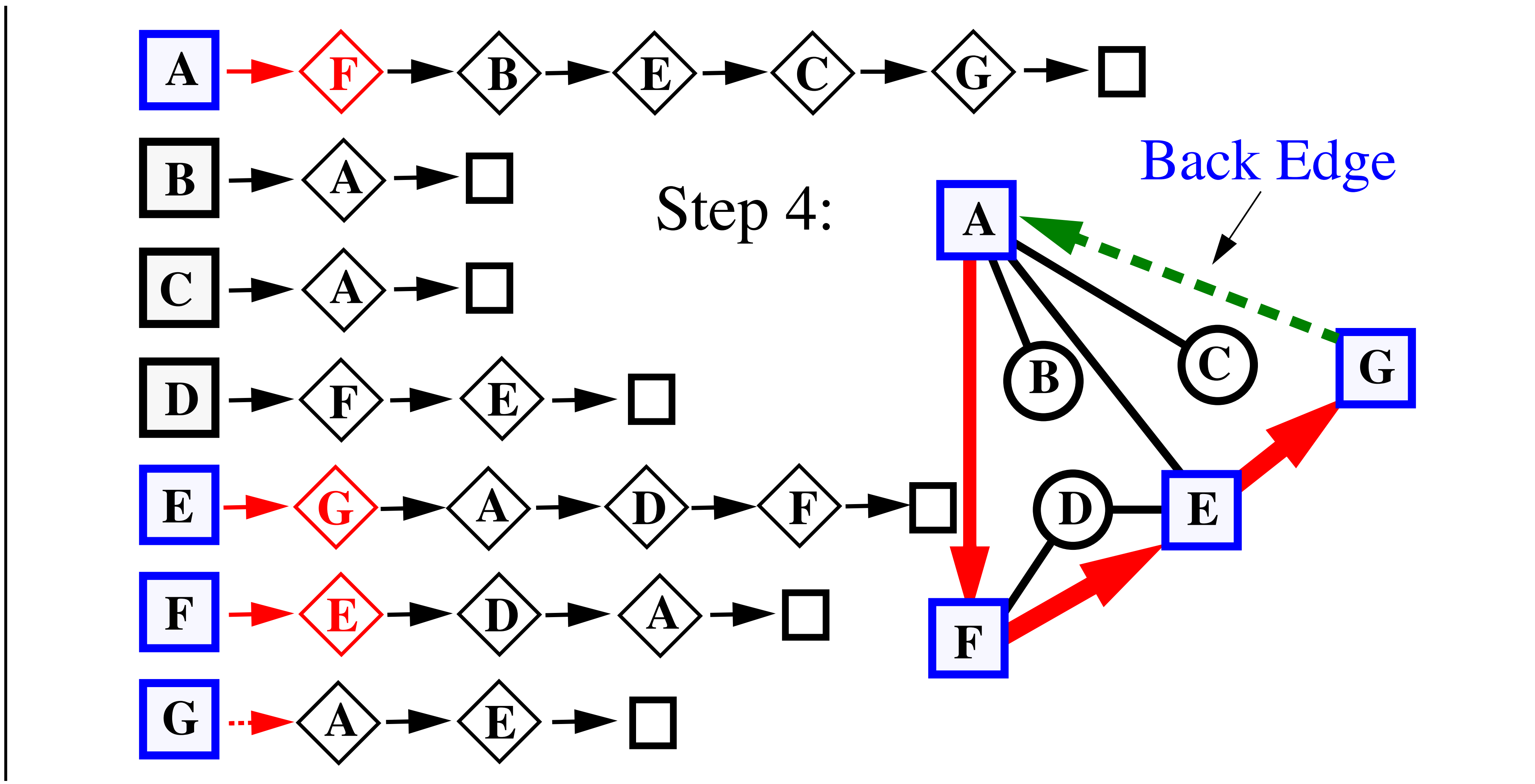
- Now an example of a DFS.


A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 1:
A B C G D E F

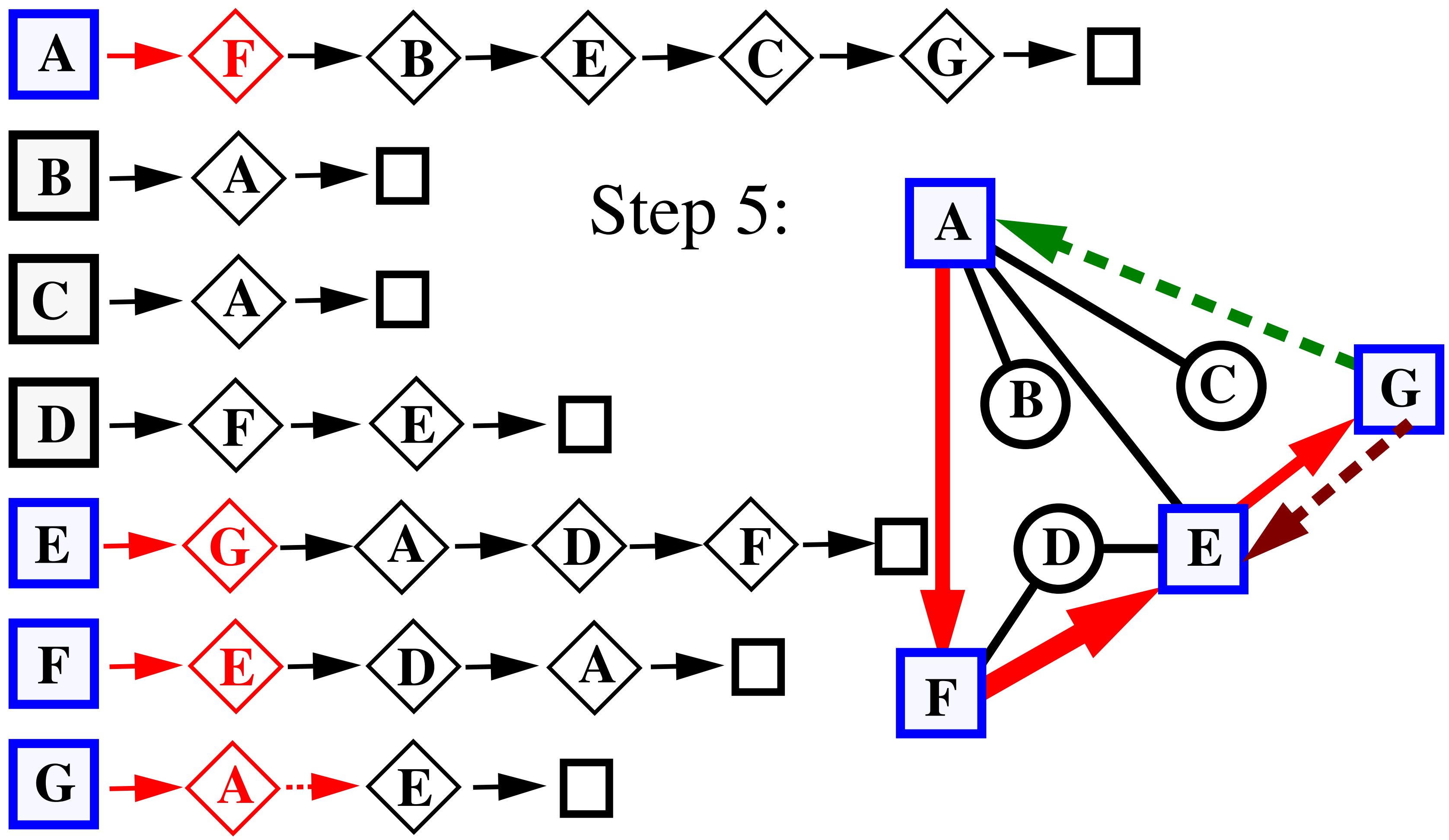

A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 2:
A B C G D E F

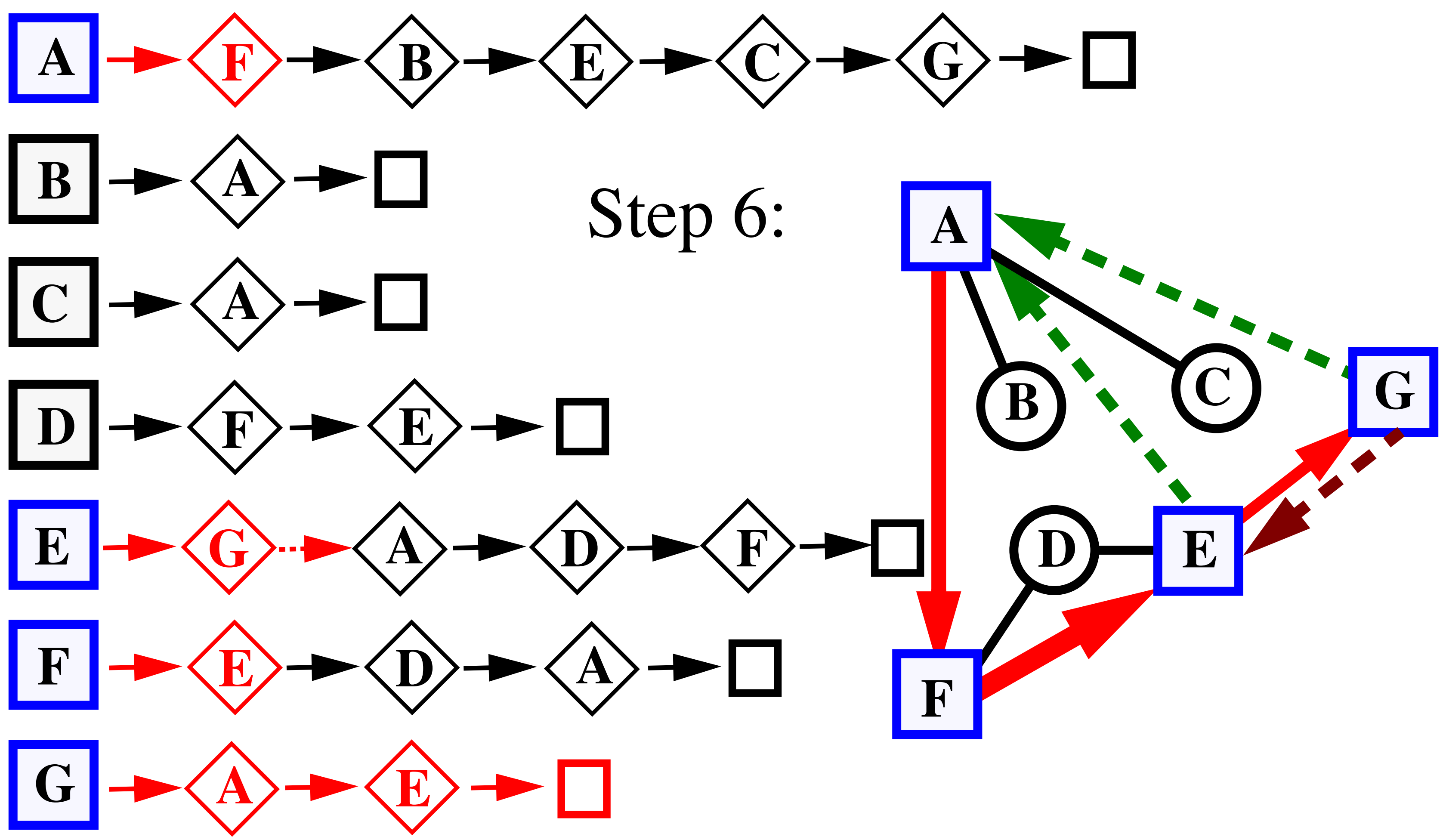


A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 3:
A B C G D E F

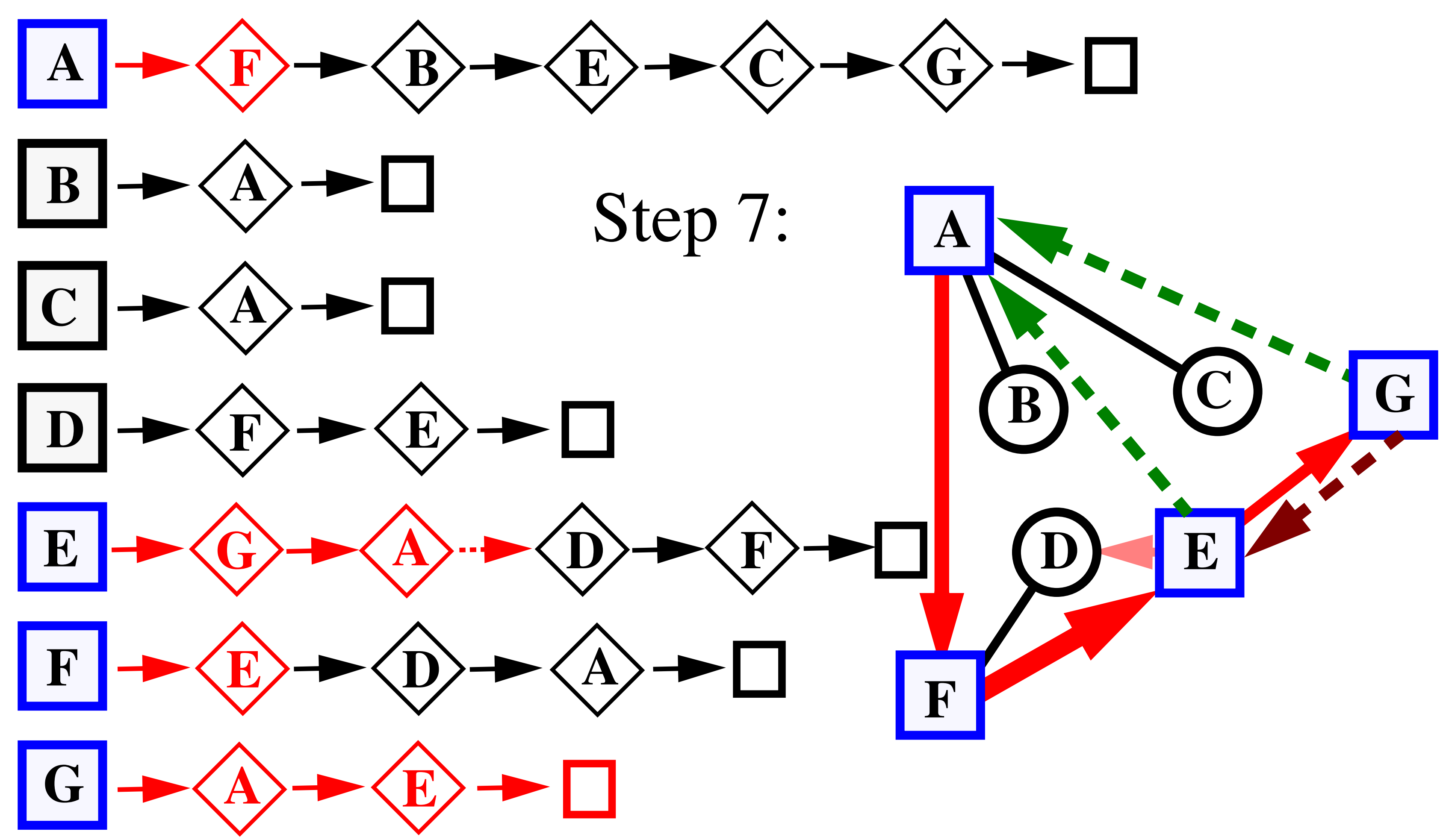


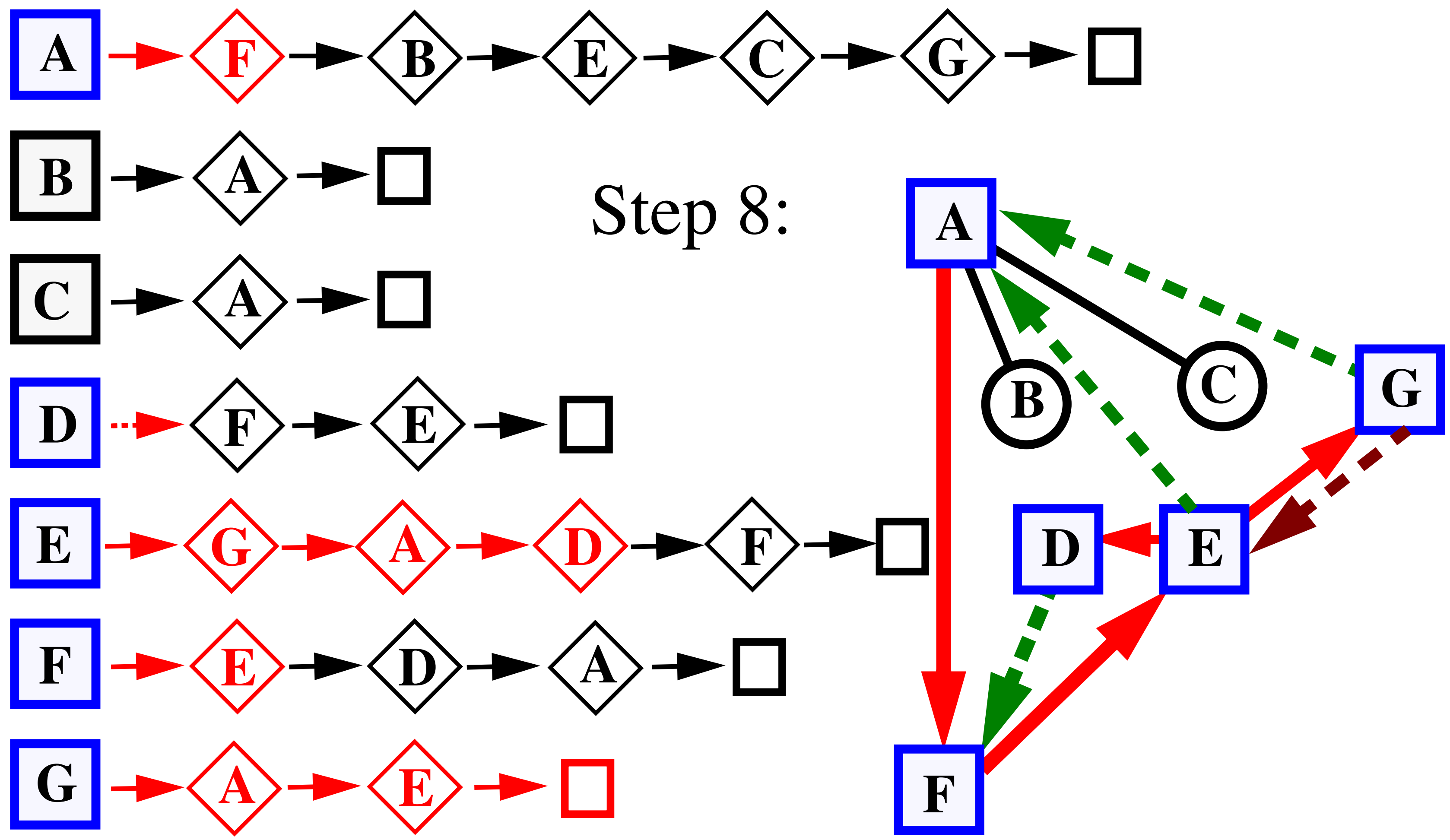
A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 4:
Back Edge
A B C G D E F

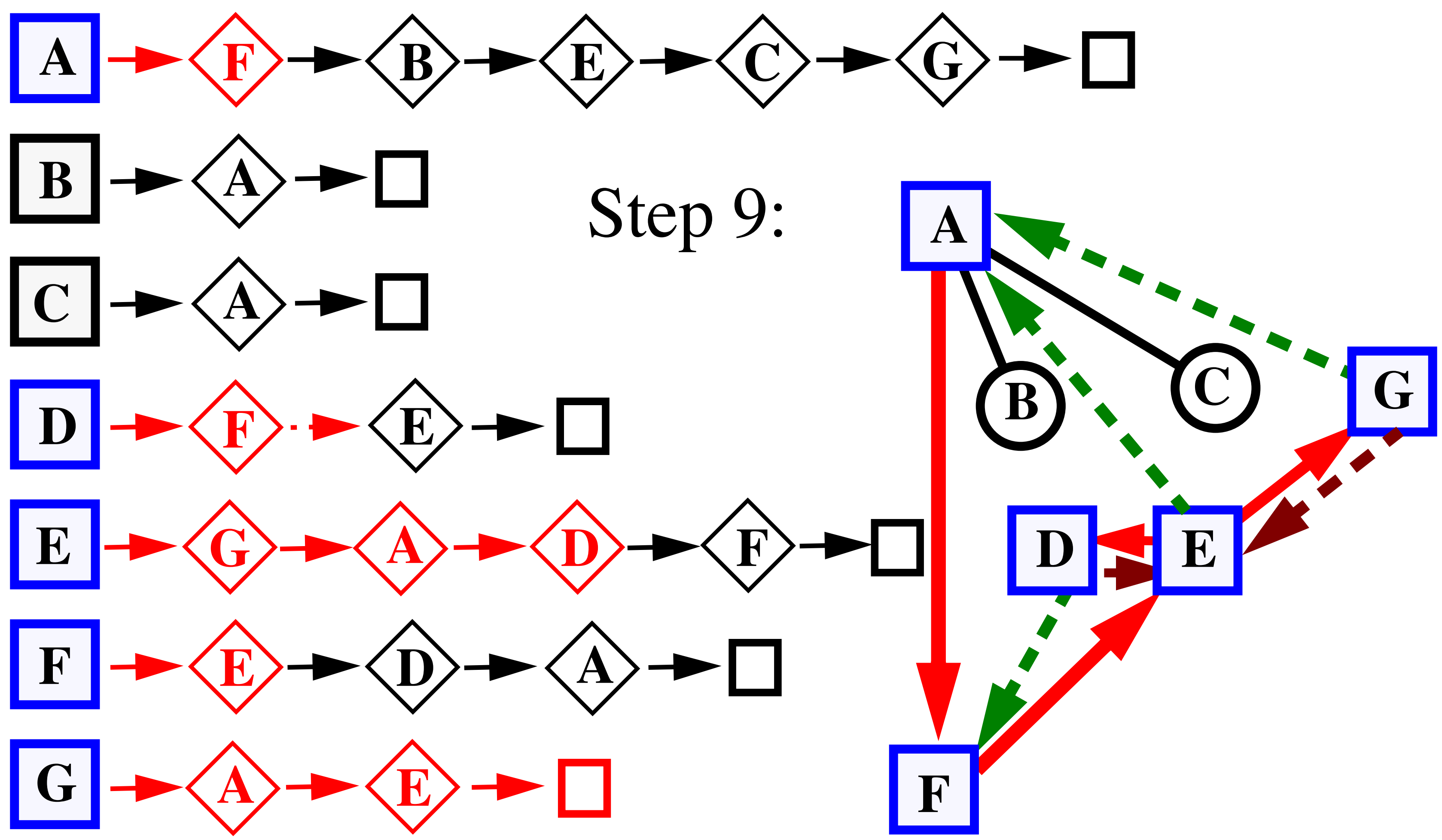

Step 5:
A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
A B C G D E F

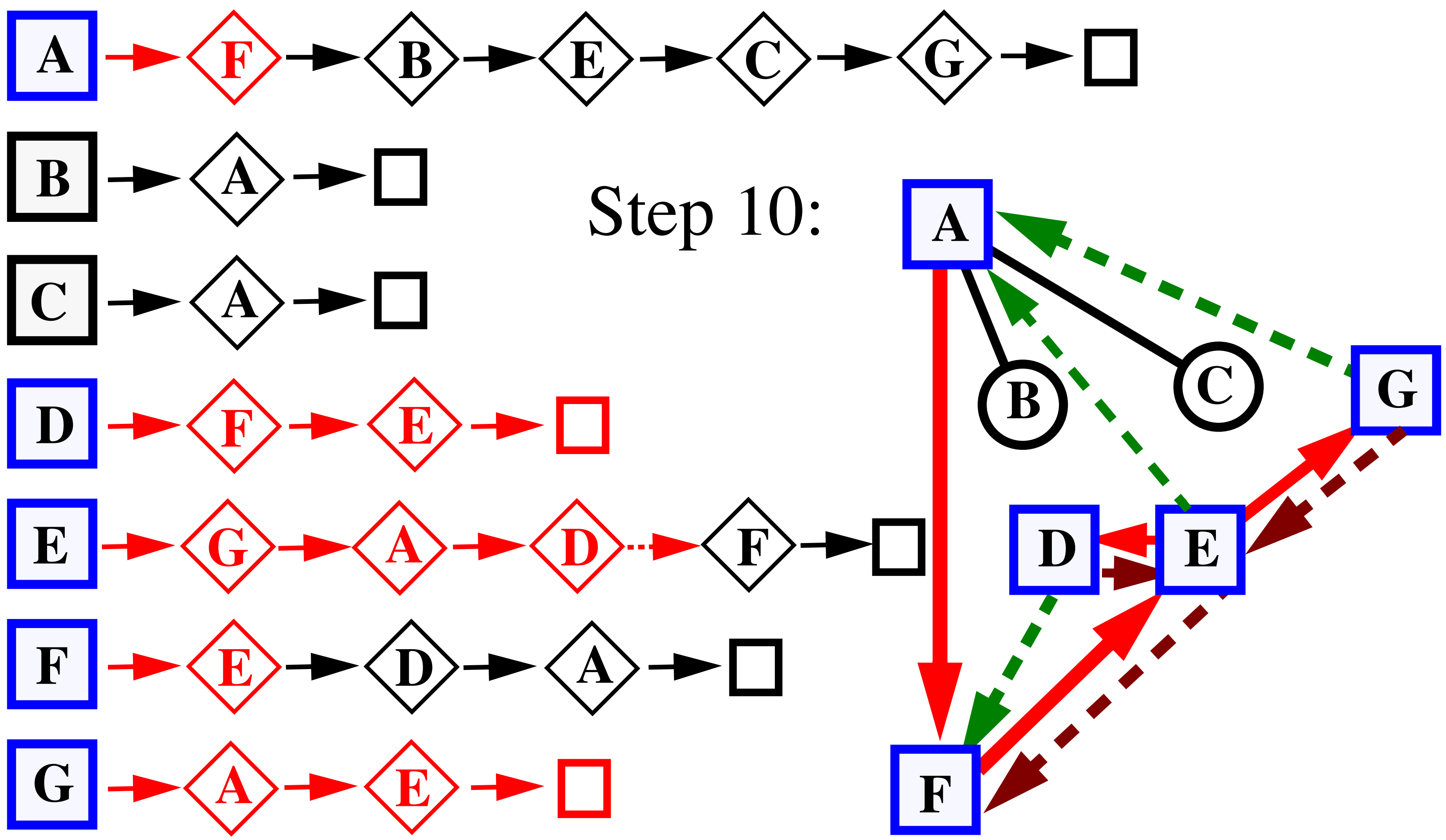


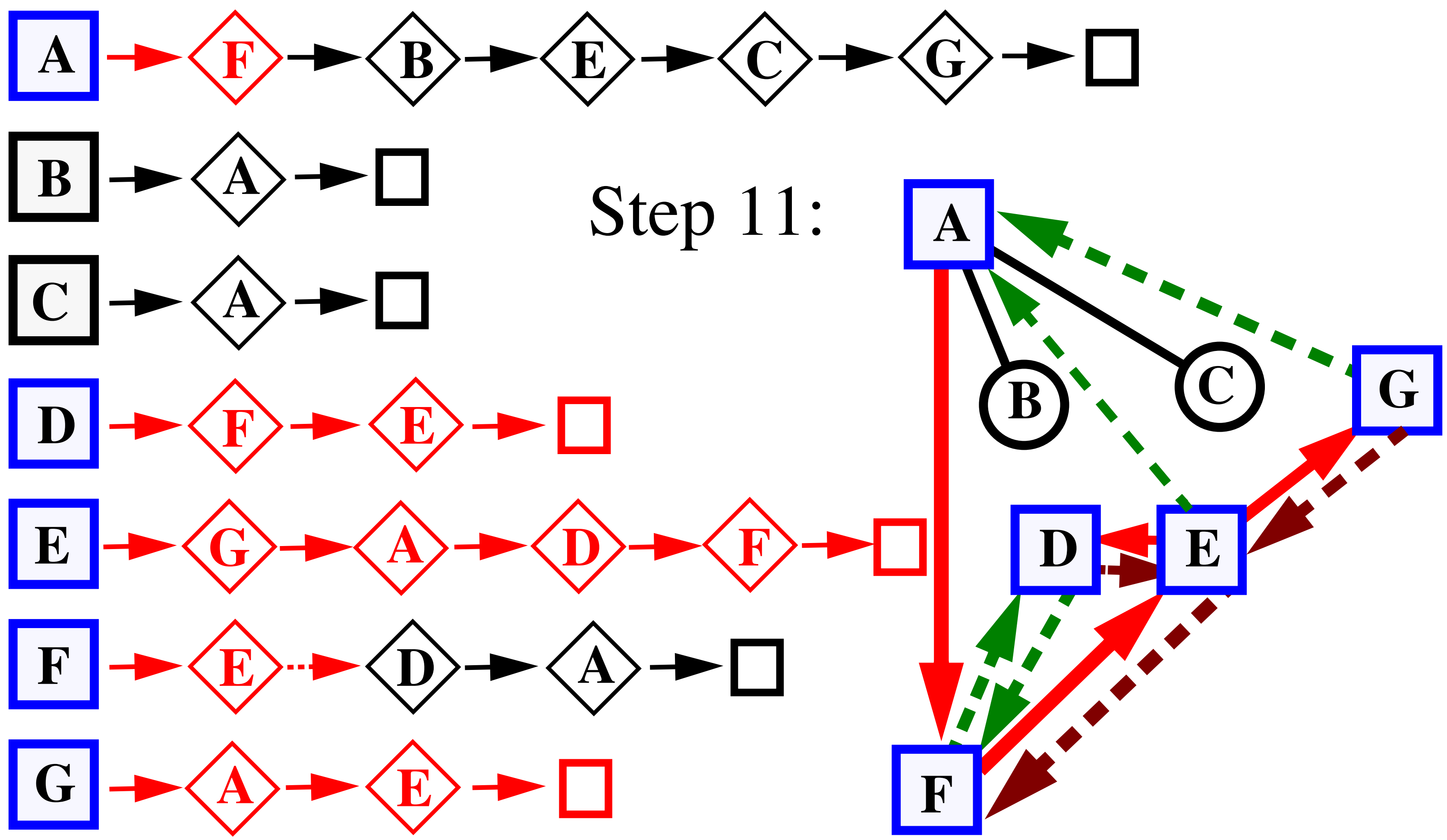
A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 6:
A B C G D E F


A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 7:
A B C G D E F

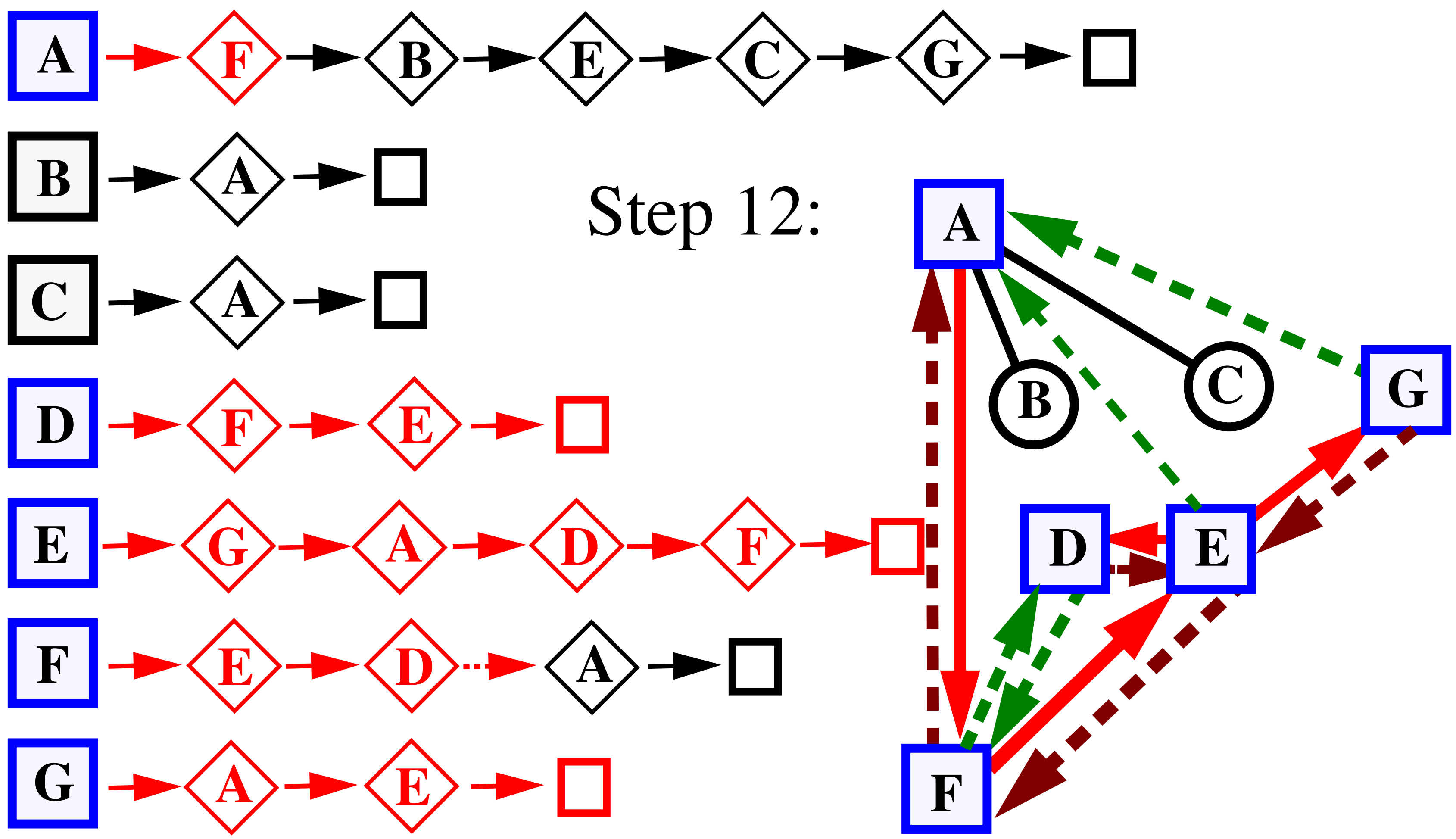

A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 8:
A
B
C
G
D
E
F

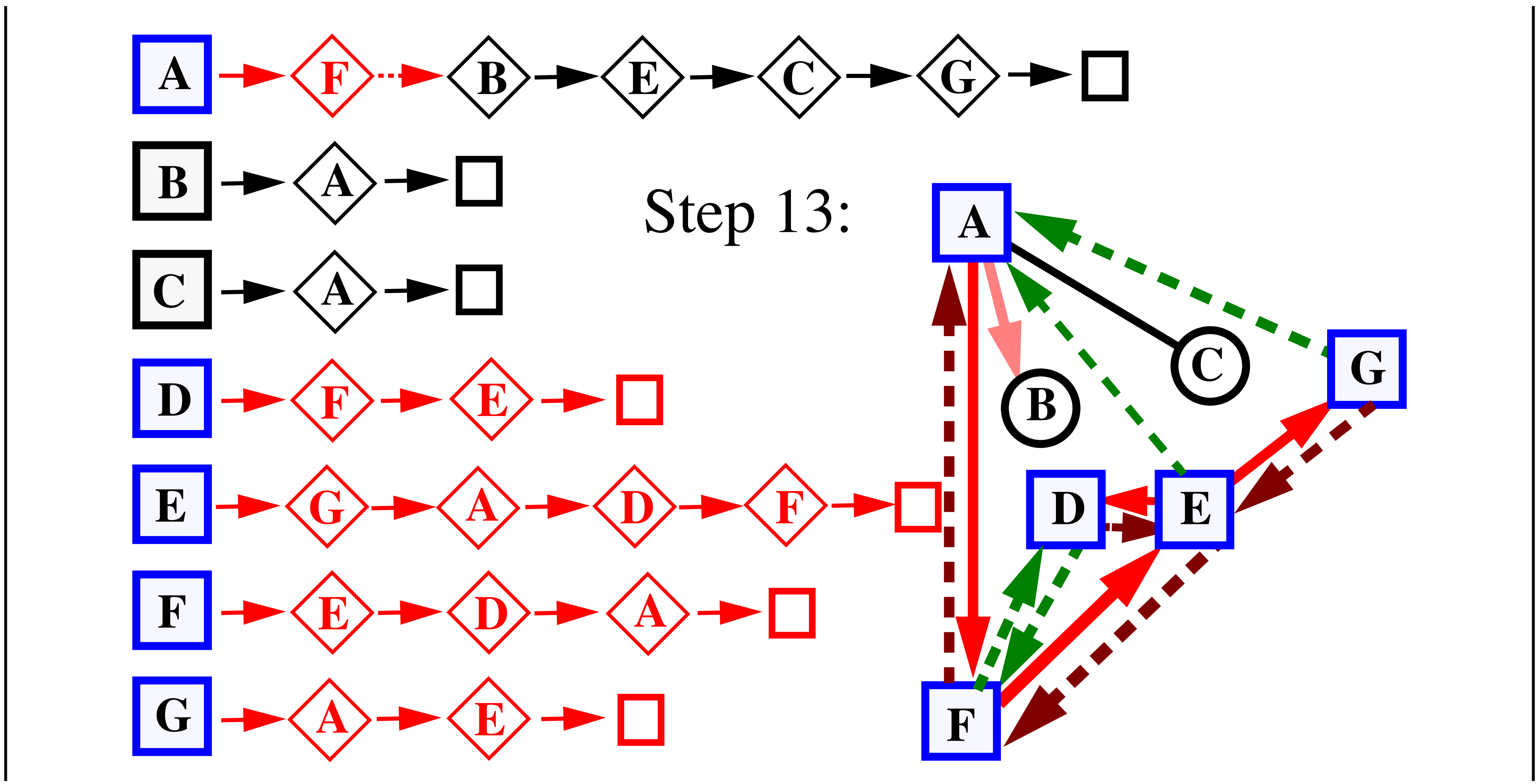

A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 9:
A B C G D E F


A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 10:
A B C G D E F

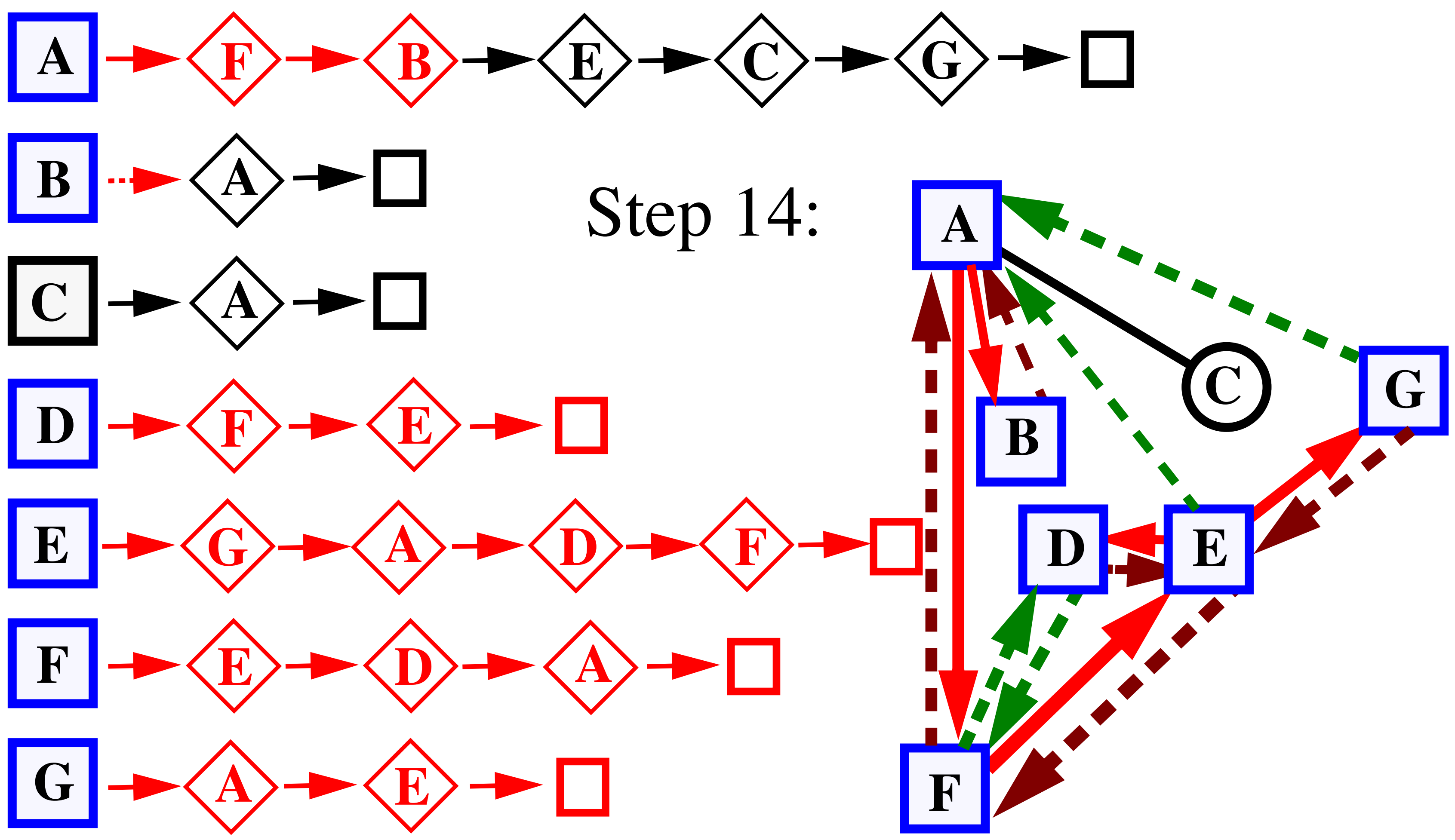


A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 11:
A B C G D E F

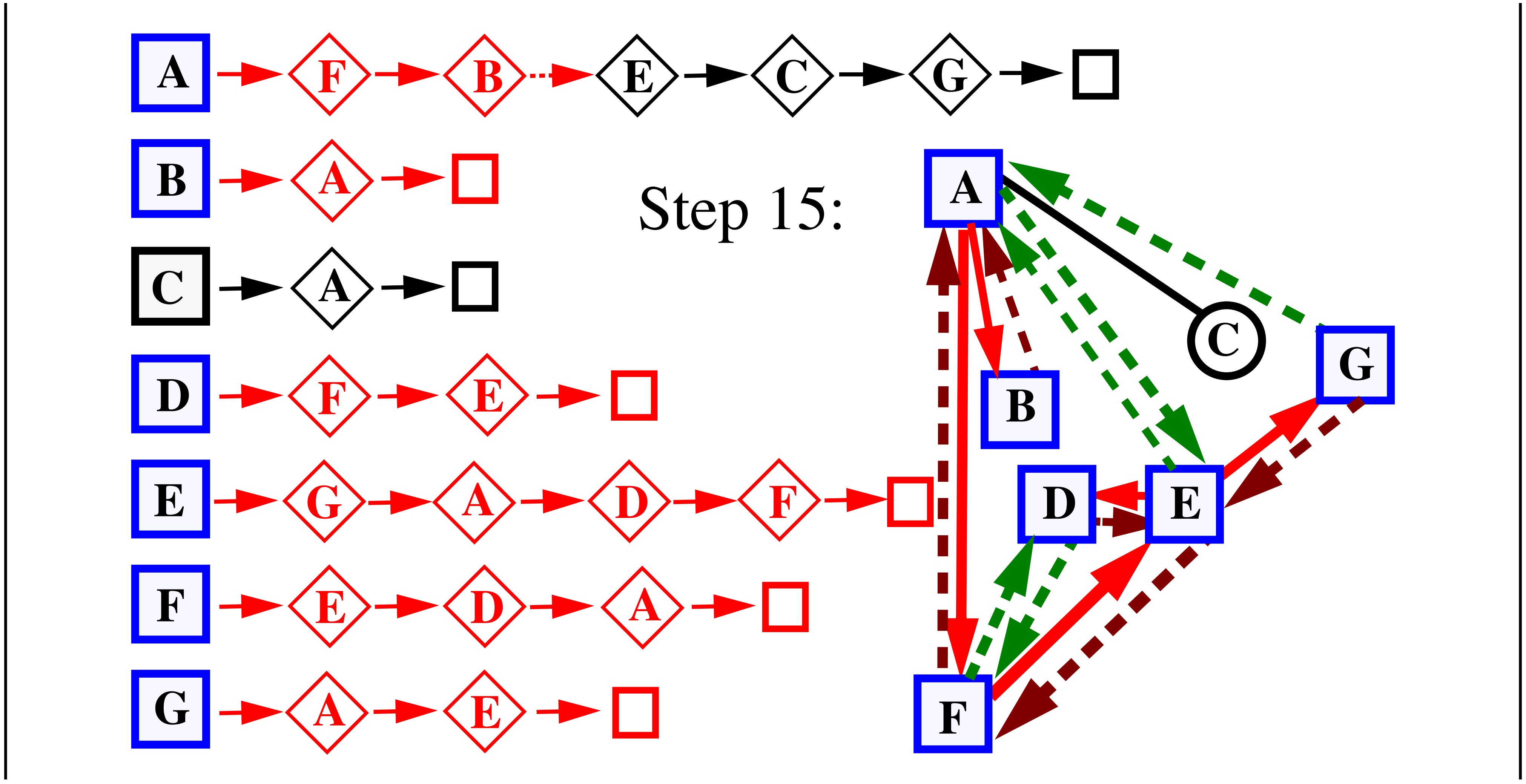

A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 12:
A B C G D E F



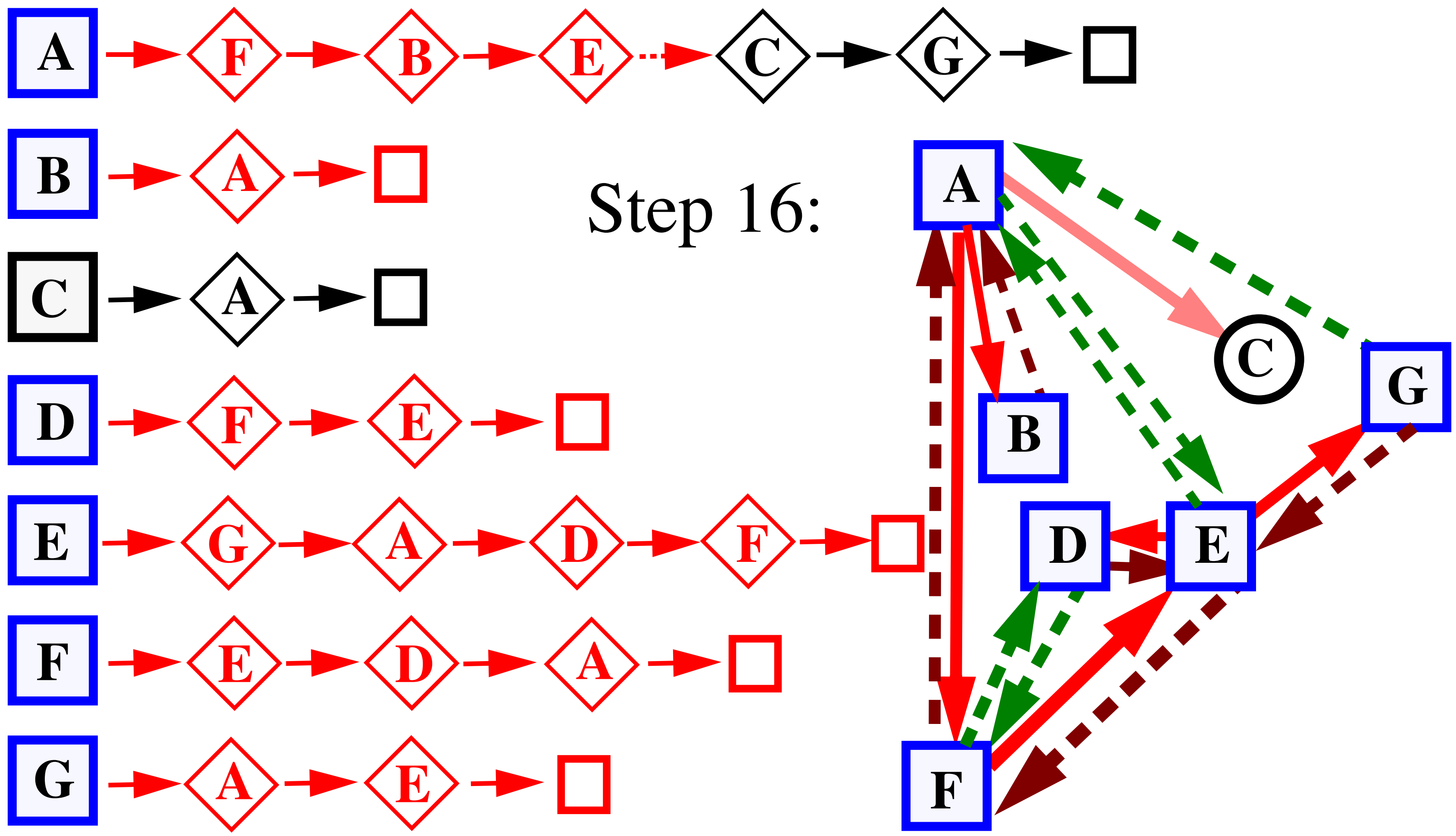
Step 13:
A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
A B C G D E F

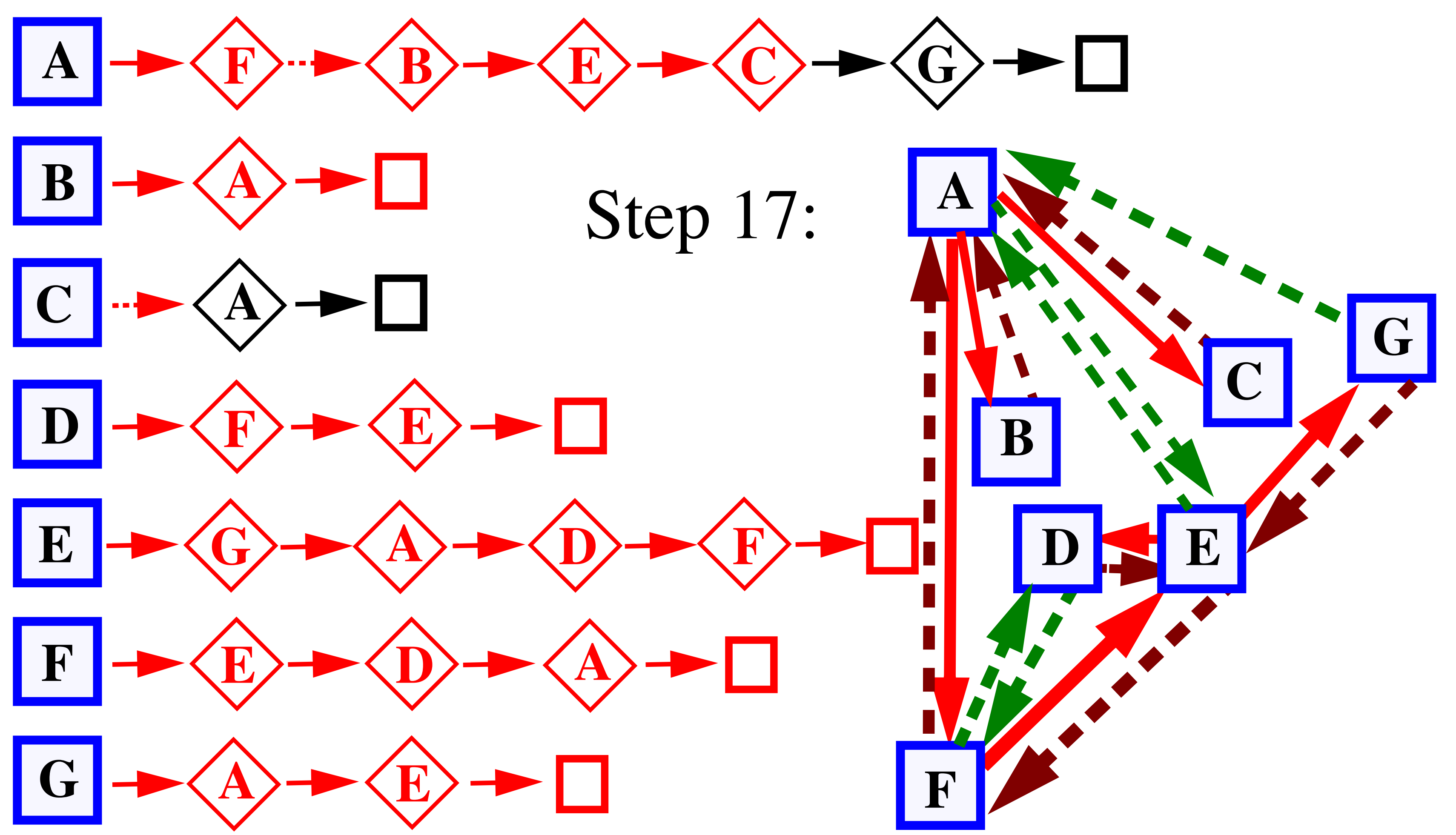


Step 14:
A
B
C
D
E
F
G

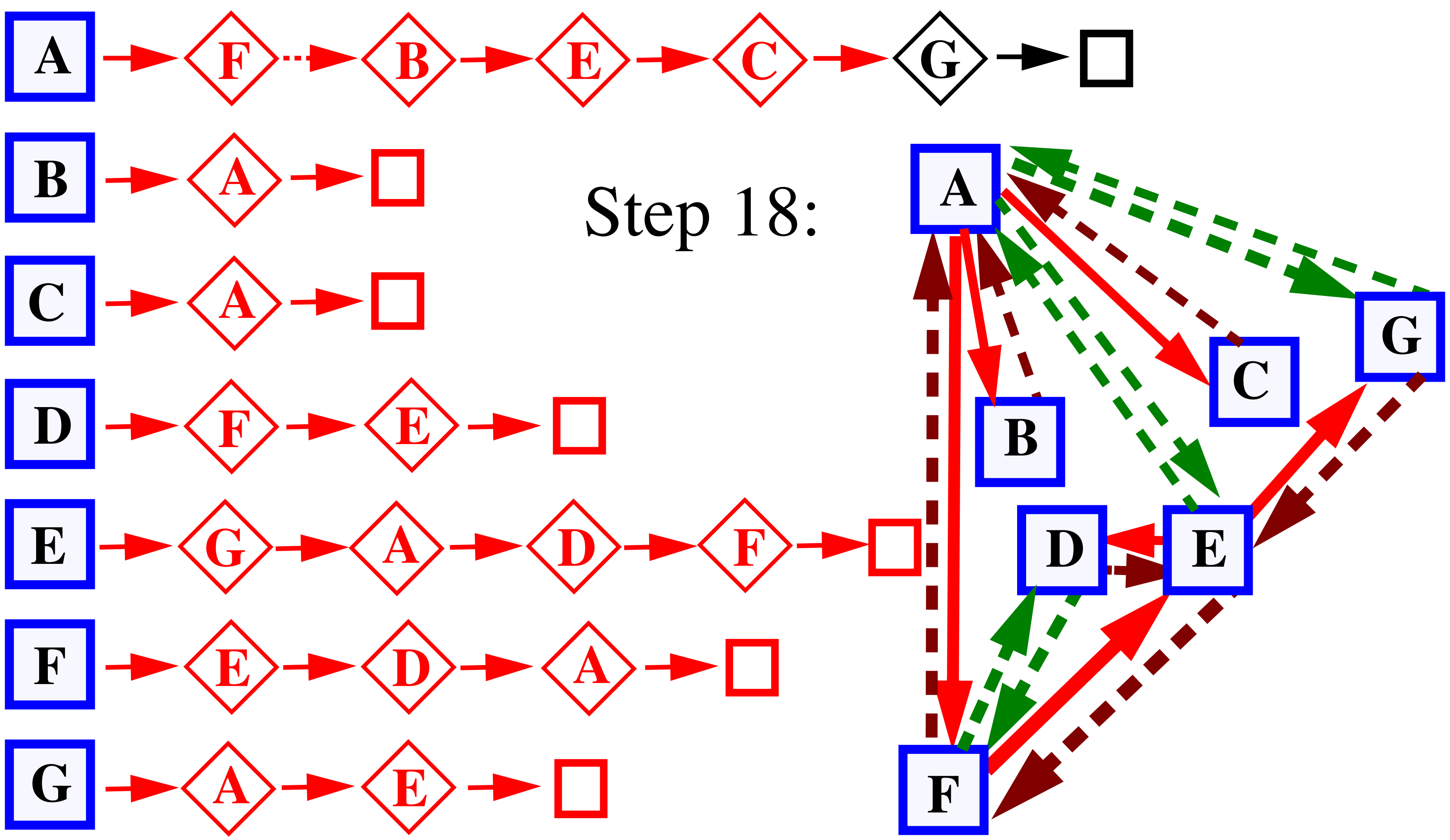


A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 15:
A B C D E F G

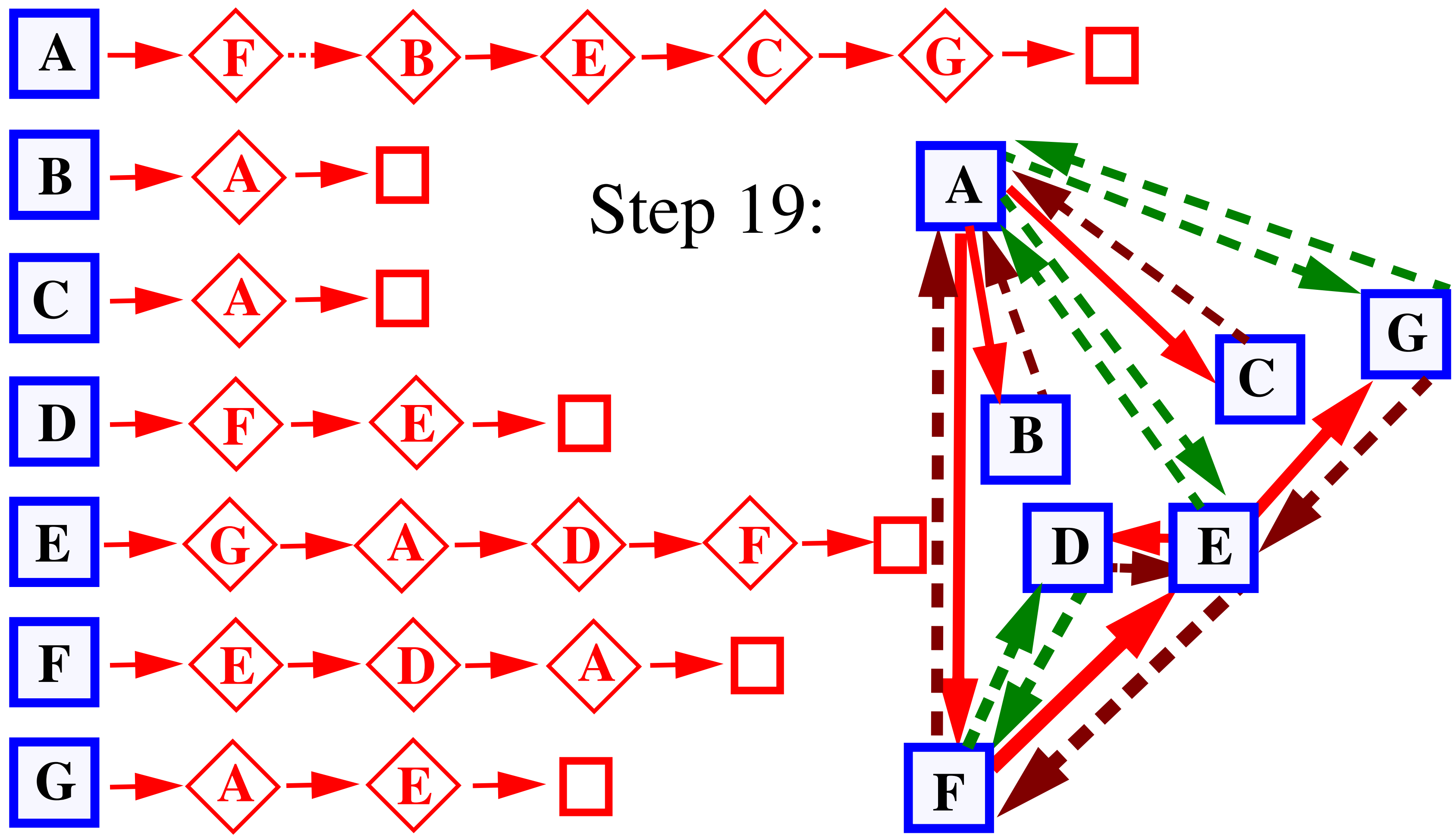

A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 16:
A
B
C
G
D
E
F



A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 17:
A B C D E F G



A F B E C G
B A
C A
D F E
E G A D F
F E D A
G A E
Step 18:
A B C D E F G



A
F
B
E
C
G
B
A
C
A
D
F
E
E
G
A
D
F
F
E
D
A
G
A
E
Step 19:
A
B
C
D
E
F
G

DFS Properties

- Proposition 9.12 : Let G be an undirected graph on which a DFS traversal starting at a vertex *s* has been preformed. Then:
  1) The traversal visits all vertices in the connected component of *s*
  2) The discovery edges form a spanning tree of the connected component of *s*

- Justification of 1):
  - Let's use a contradiction argument: suppose there is at least on vertex *v* not visited and let *w* be the first unvisited vertex on some path from *s* to *v*.
  - Because *w* was the first unvisited vertex on the path, there is a neighbor *u* that has been visited.
  - But when we visited *u* we must have looked at edge(*u*, *w*). Therefore *w* must have been visited.
  - and justification

DFS Properties

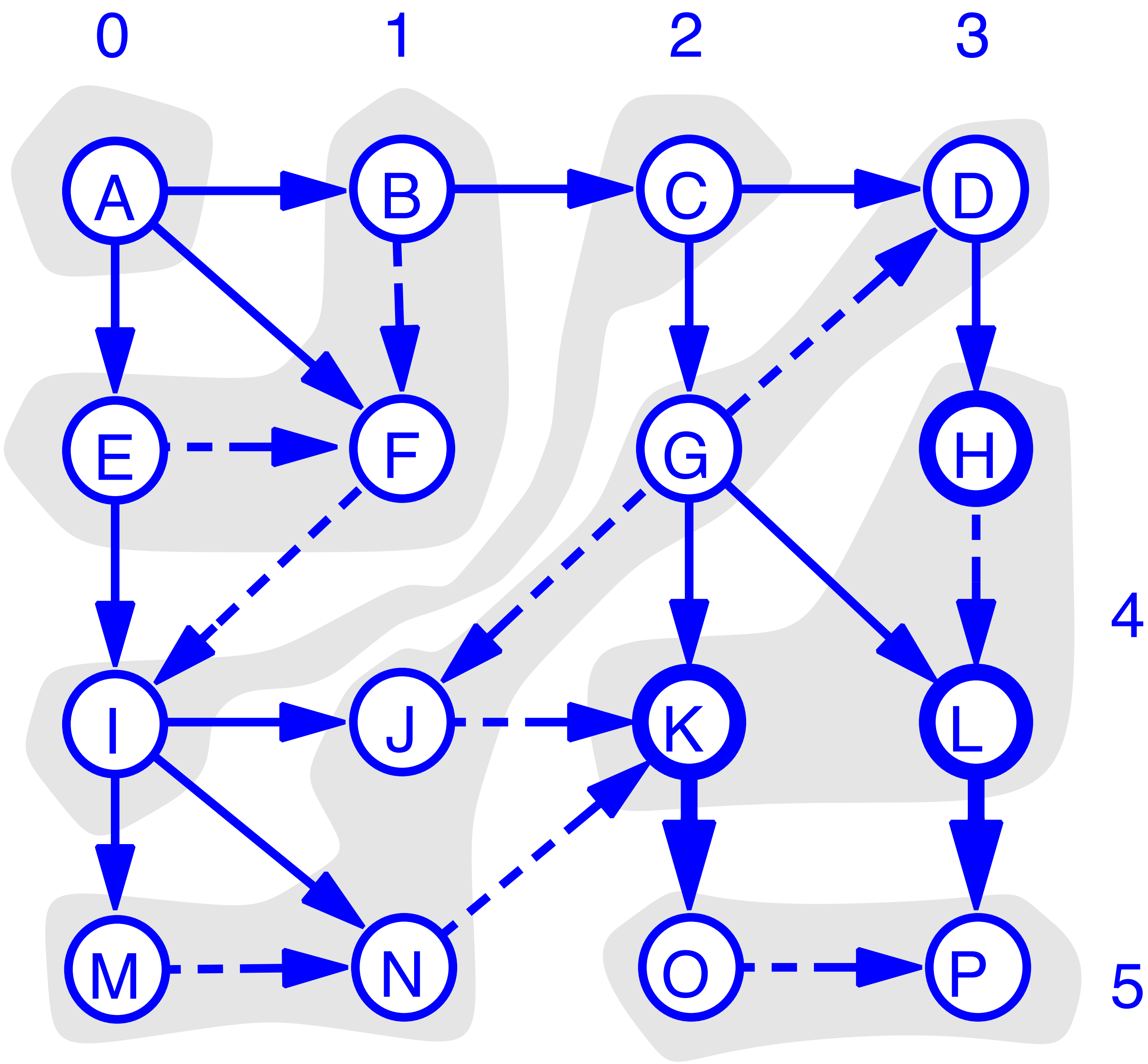
- Proposition 9.12 : Let G be an undirected graph on which a DFS traversal starting at a vertex *s* has been preformed. Then:
  1) The traversal visits all vertices in the connected component of *s*
  2) The discovery edges form a spanning tree of the connected component of *s*

- Justification of 2):
  - We only mark edges from when we go to unvisited vertices. So we never form a cycle of discovery edges, i.e. discovery edges form a tree.
  - This is a spanning tree because DFS visits each vertex in the connected component of *s*

Running Time Analysis

- Remember:
  - DFS is called on each vertex exactly once.
  - Every edge is examined exactly twice, once from each of its vertices
- For n_s vertices and m_s edges in the connected component of the vertex s, a DFS starting at s runs in $O(n_s + m_s)$ time if:
  - The graph is represented in a data structure, like the adjacency list, where vertex and edge methods take constant time
  - Marking a vertex as explored and testing to see if a vertex has been explored takes O(degree)
  - By marking visited nodes, we can systematically consider the edges incident on the current vertex so we do not examine the same edge more than once.

Breadth-First Search



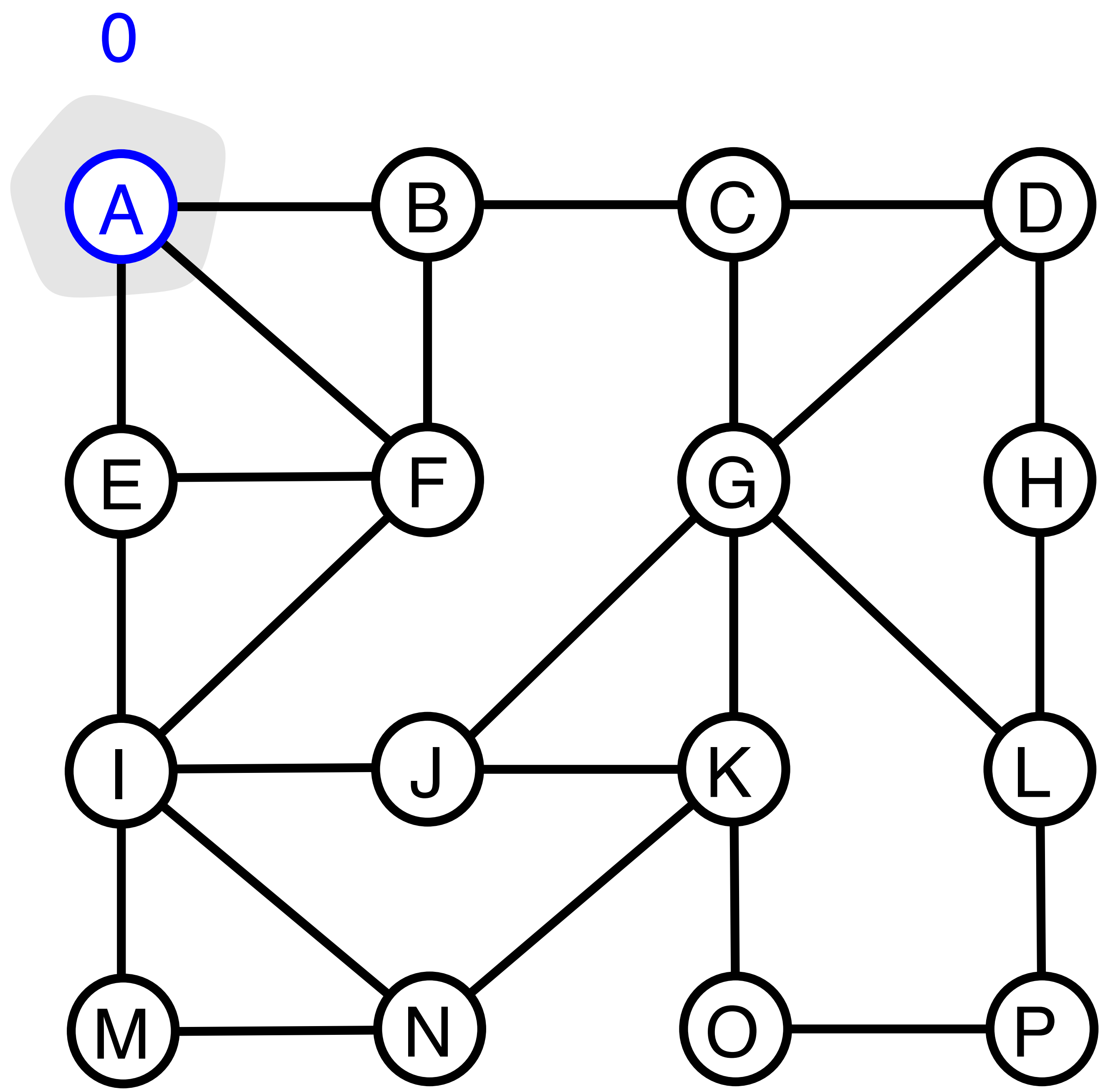
Breadth-First Search

- Like DFS, a Breadth-First Search (BFS) traverses a connected component of a graph, and in doing so definesaspanningtreewithseveralusefulproperties

  - -The starting vertex *s* has level 0, and, as in DFS, defines that point as an "anchor."
  - -In the first round, the string is unrolled the length of one edge, and all of the edges that are only one edge away from the anchor are visited.
  - -These edges are placed into level 1
  - -In the second round, all the new edges that can be reached by unrolling the string 2 edges are visited and placed in level 2.
  - -This continues until every vertex has been assigned a level.
  - -The label of any vertex *v* corresponds to the length of the shortest path from *s* to *v*.

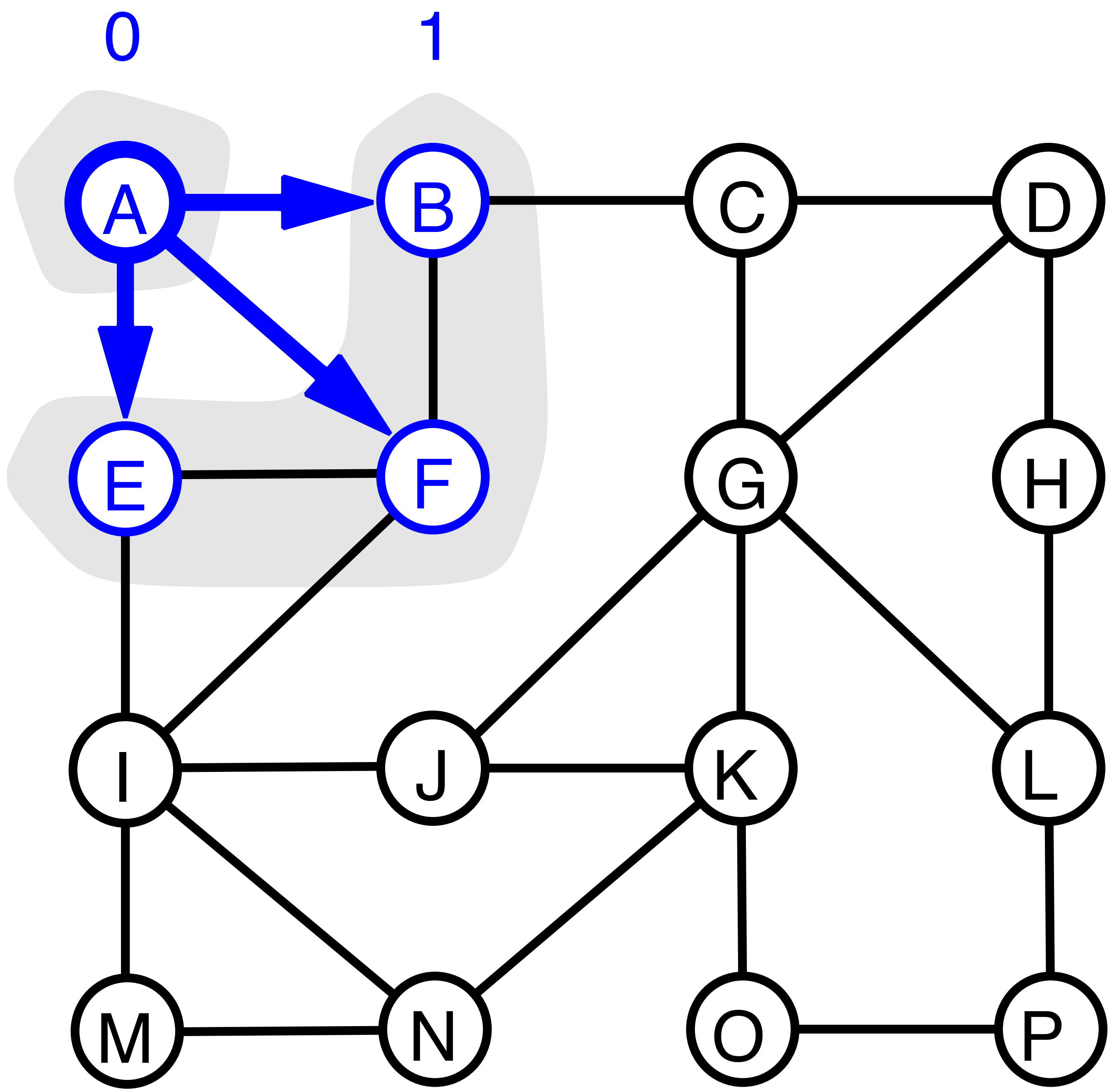
BFS - A Graphical Representation

a)

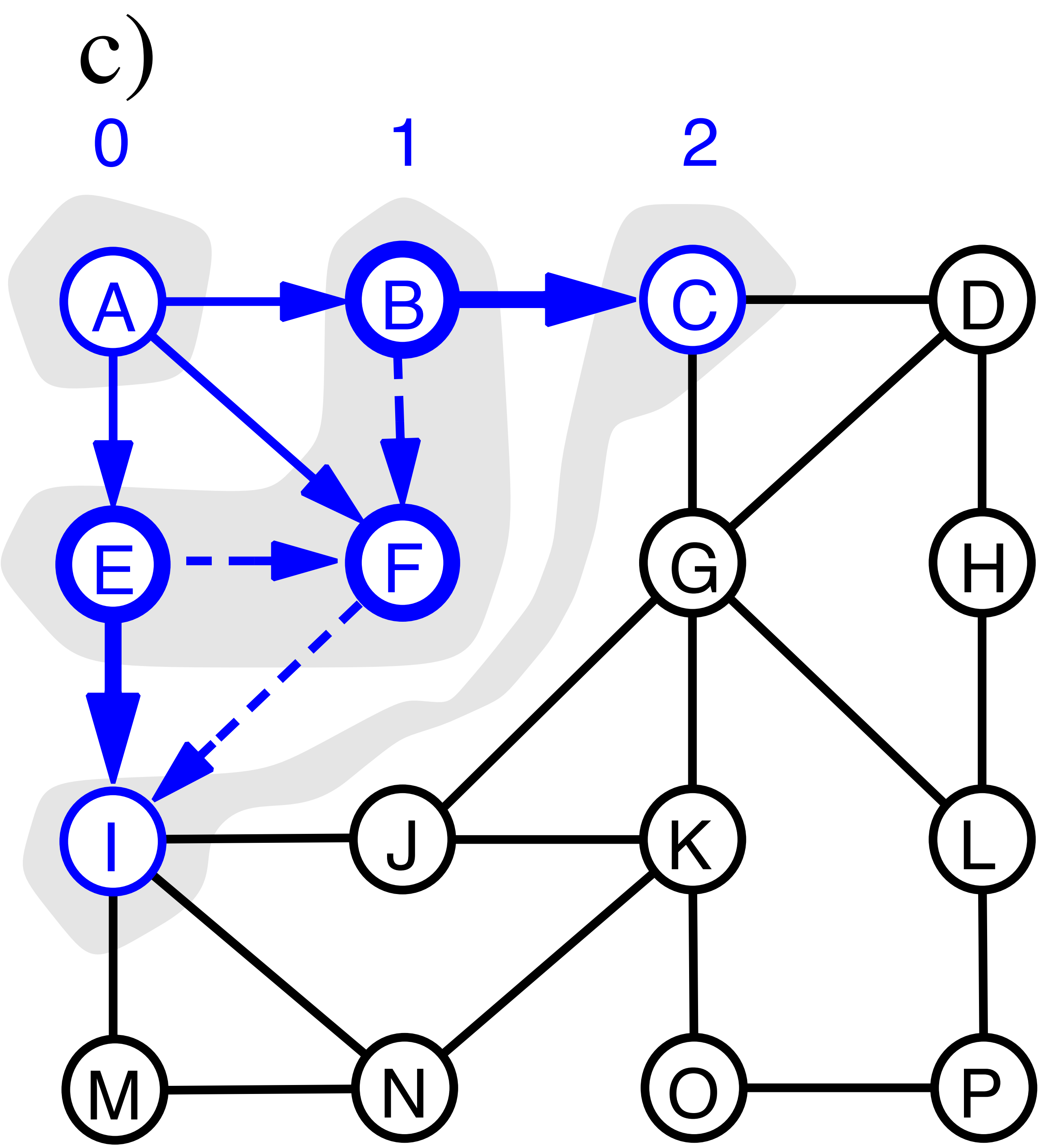


BFS - A Graphical Representation

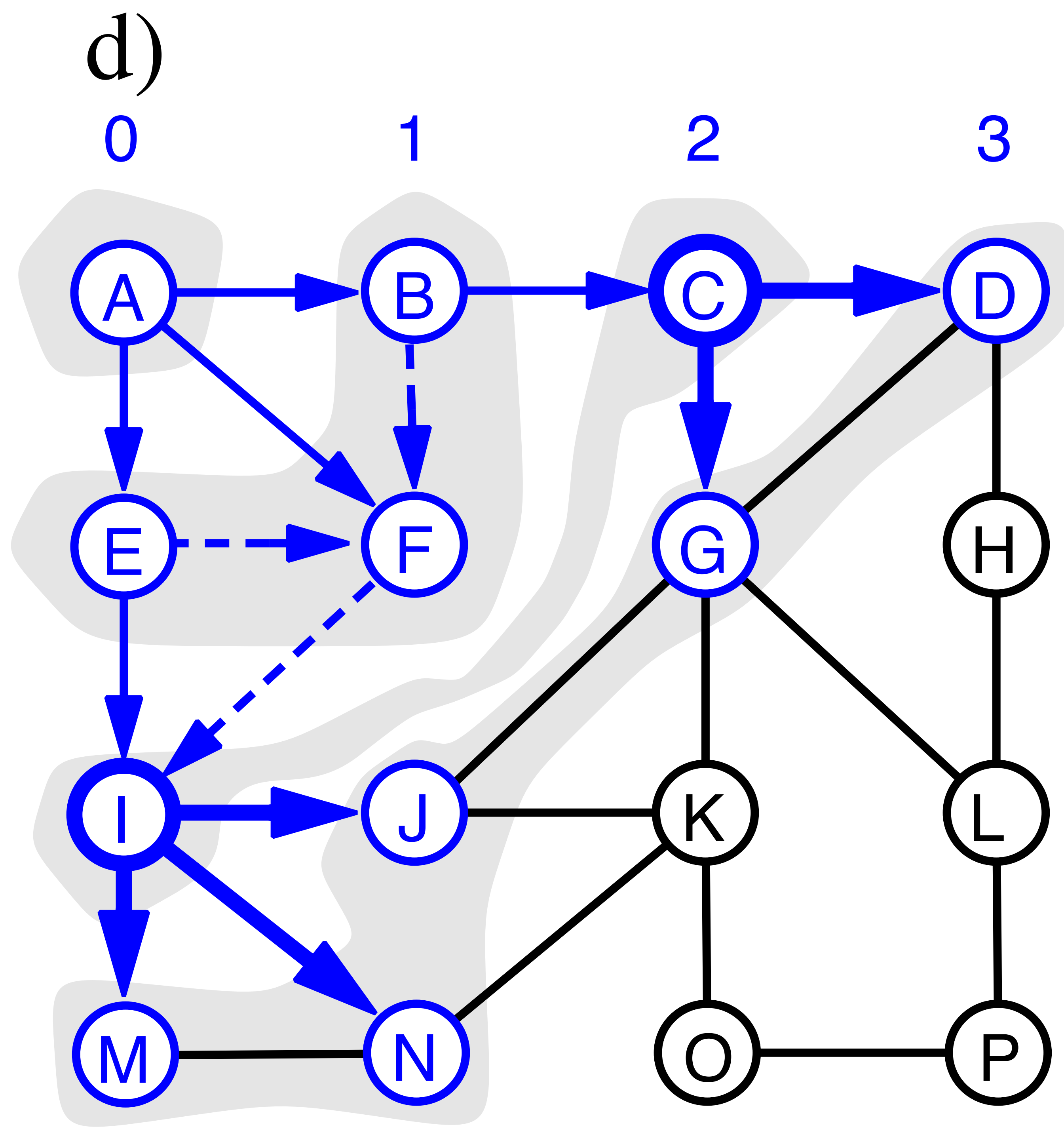
b)



BFS - A Graphical Representation

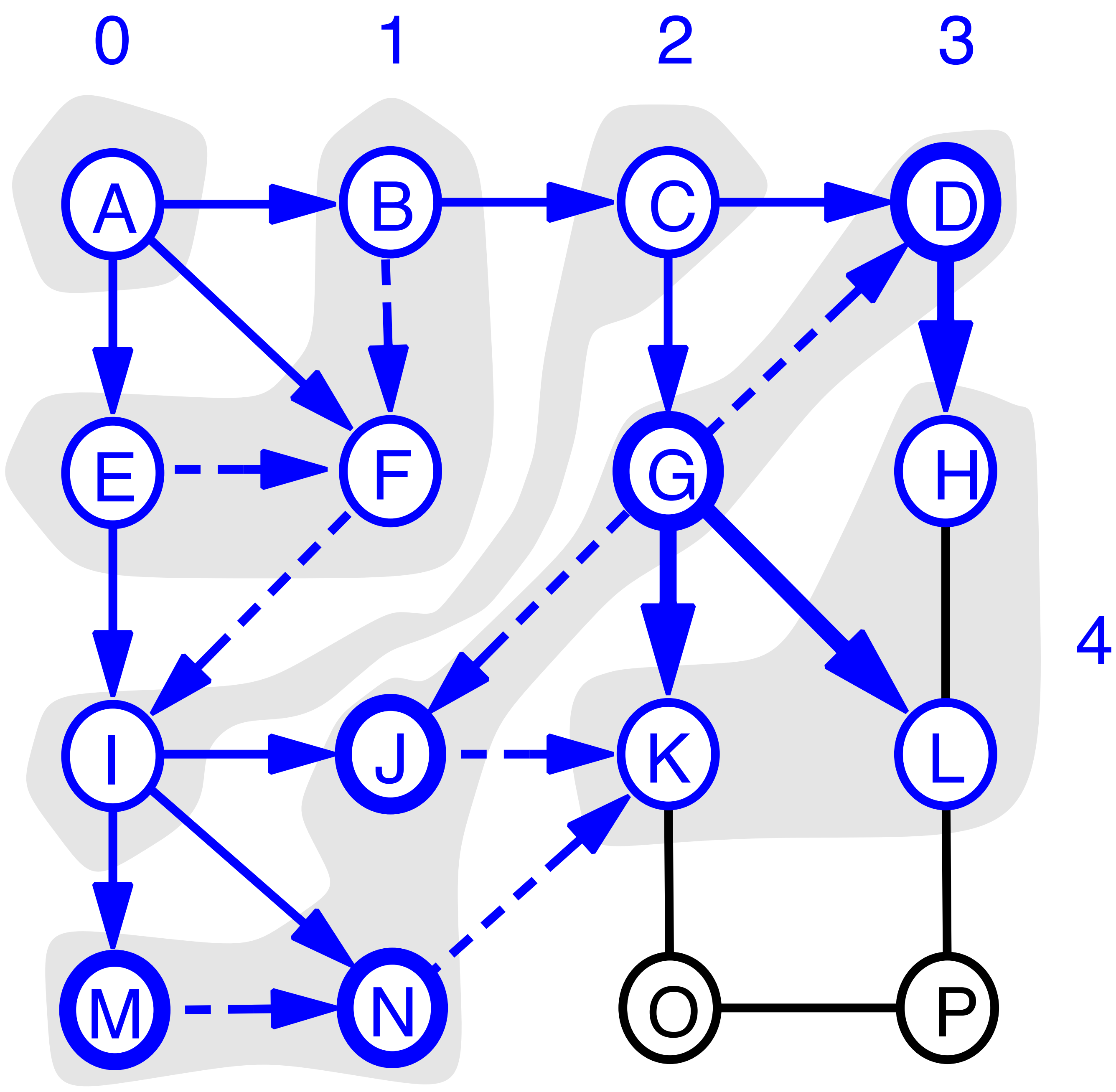


BFS - A Graphical Representation



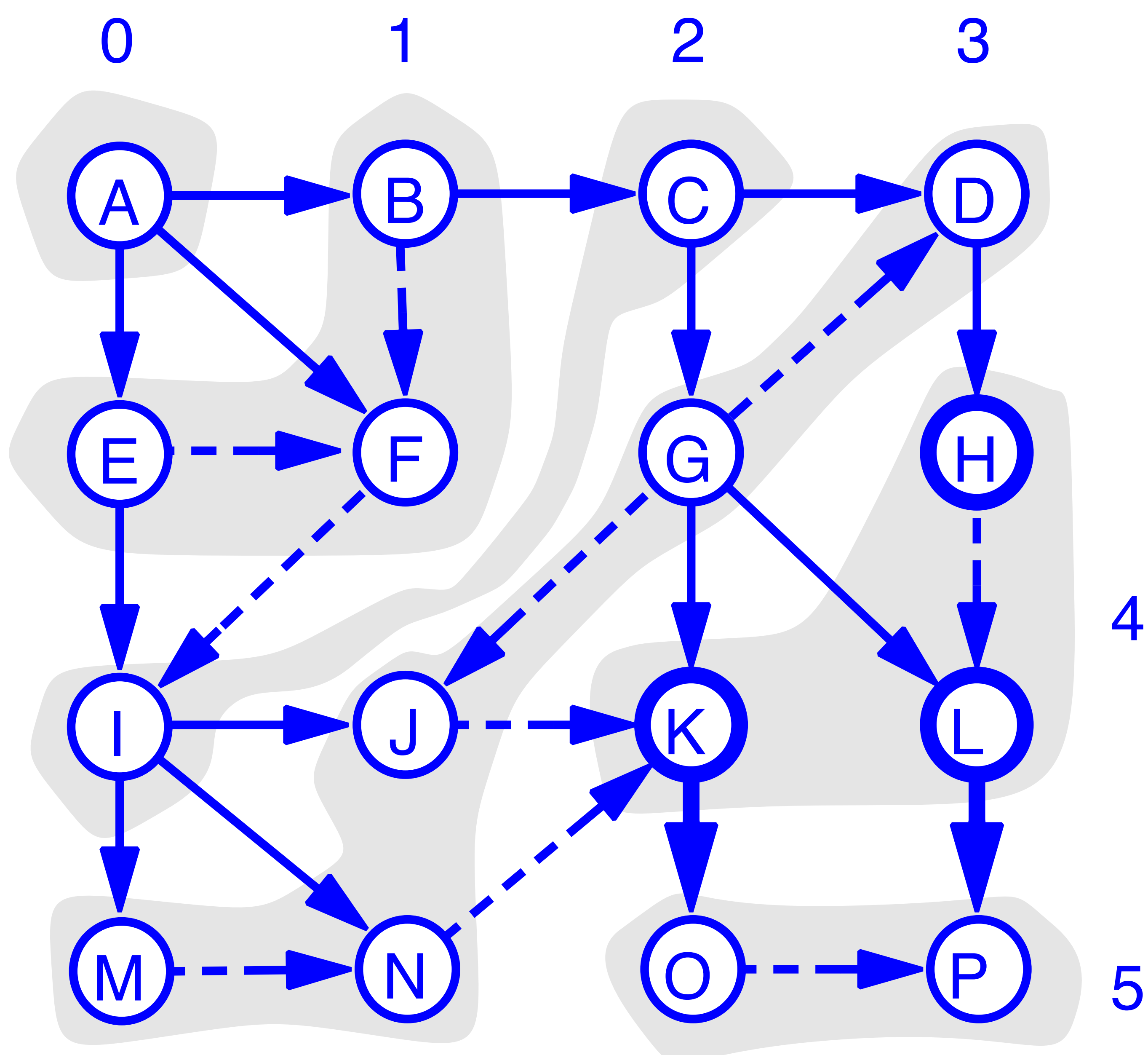
BFS - A Graphical Representation

e)



BFS - A Graphical Representation

f)



BFS Pseudo-Code

```
Algorithm BFS(s):
  Input: A vertex s in a graph
  Output: A labeling of the edges as "discovery" edges
     and "cross edges"
  initialize container L_0 to contain vertex s
  i ← 0
  while L_i is not empty do
     create container L_{i+1} to initially be empty
     for each vertex v in L_i do
        for each edge e incident on v do
           if edge e is unexplored then
              let w be the other endpoint of e
              if vertex w is unexplored then
                 label e as a discovery edge
                 insert w into L_{i+1}
              else
                 label e as a cross edge
     i ← i + 1
```

Properties of BFS

- Proposition:Let G be an undirected graph on which a BFS traversal starting at vertex s has been performed. Then

  -The traversal visits all vertices in the connected component of s.

  -The discovery-edges form a spanning tree T, which we call the BFS tree, of the connected component of s

  -For each vertex v at level i,the path of the BFS tree T between s and v has i edges, and any other path of G between s and v has at least i edges.

  - If (u, v) is an edge that is not in the BFS tree, then the level numbers of u and v differ by at most one.

Properties of BFS

- Proposition: Let G be a graph with n vertices and m edges. A BFS traversal of G takes time $O(n + m)$. Also, there exist $O(n + m)$ time algorithms based on BFS for the following problems:

  -Testing whether G is connected.

  -Computing a spanning tree of G

  -Computing the connected components of G

  -Computing, for every vertex v of G, the minimum number of edges of any path between s and v.

Winter 2016
COMP-250: Introduction to Computer Science

Lecture 18, March 17, 2016