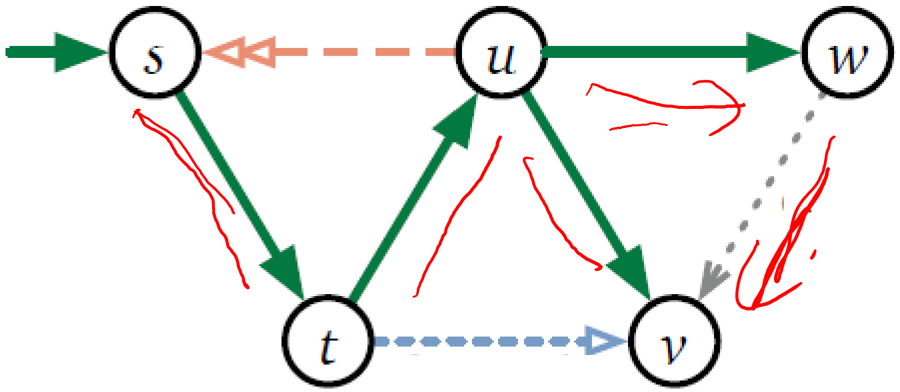
DFS

Sridhar Alagar

DFS in a directed Graph

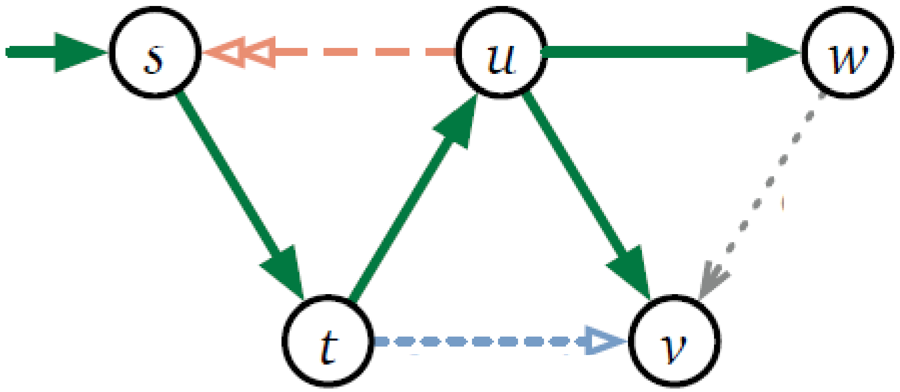


Use stack for DFS

```
wfs(s){
    put s into bag
    while bag not empty
          take v from bag
          if v is unmarked
             mark v
             for each (v, w) in G do
                put w in bag
}
```

```
dfs(v){
    if v is unmarked
       mark v
       for each (v, w) in G do
             dfs(w)
}
```

DFS



Wrapper method to visit all nodes

```
dfsAll(G){
    initialize(G)
    for each v in G do
      unmark v
    for each v in G do
      if v is unmarked
          dfs(v)
}
```

```
dfs(v){
  mark v
  pre(v)
  for each (v, w) in G do
      if w is unmarked
            w.parent = v
            dfs(w)
  post(v)
}
```

Preorder and postorder traversal of DFS Forest

Record when a node entered and left stack

```
dfsAll(G){
   initialize(G)
   for each v in G do
     unmark v
   for each v in G d0
     if v is unmarked
         dfs(v)
}
```

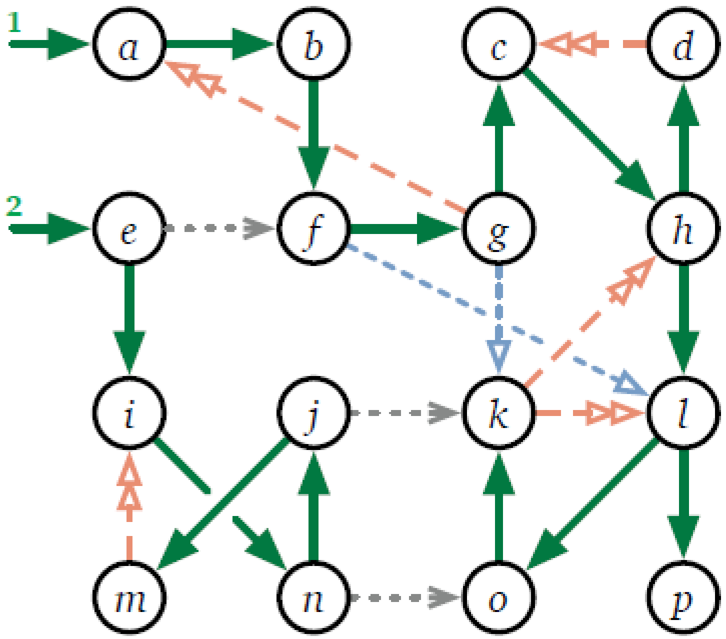
```
dfs(v){
  mark v
  pre(v)
  for each (v, w) in G do
      if v is unmarked
            w.parent = v
            dfs(w)
  post(v)
}
```

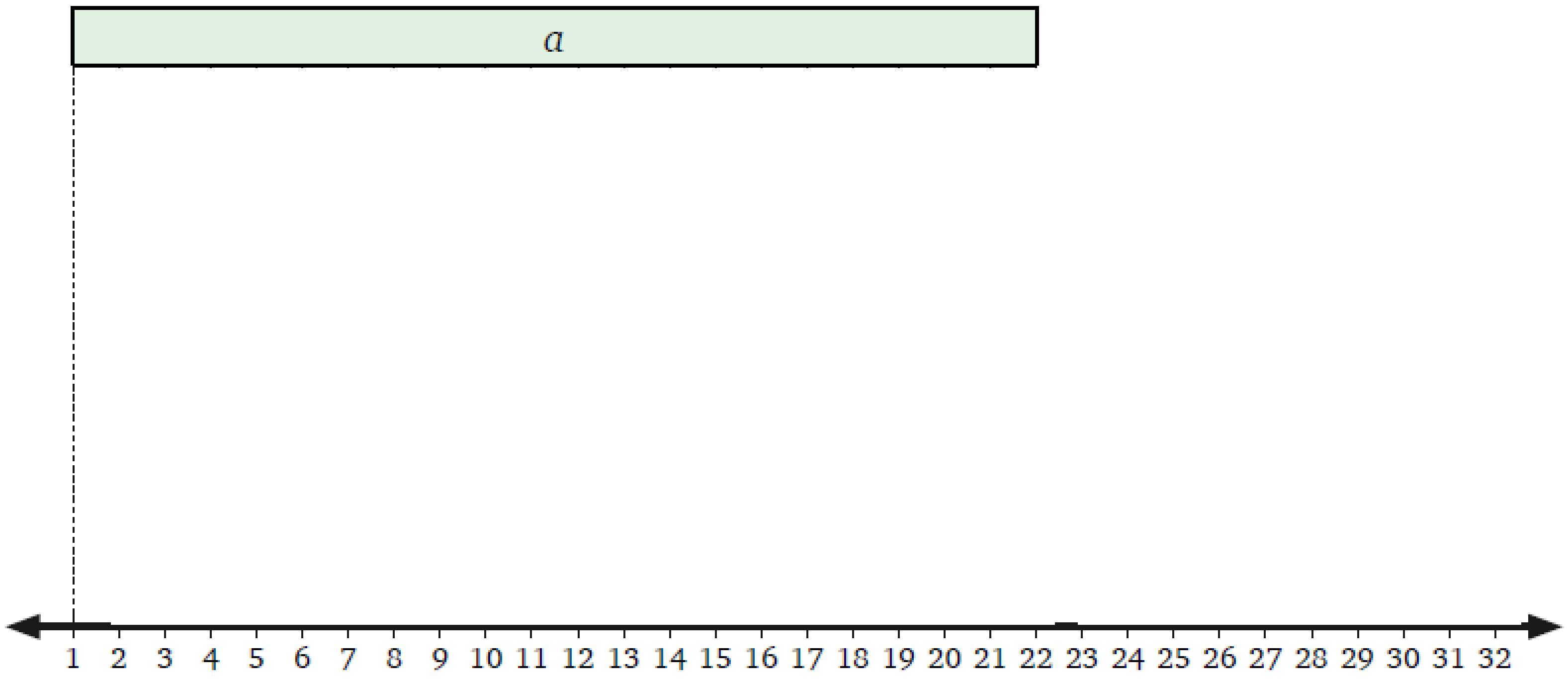
```
initialize(G)
  clock = 0
```

```
pre(v)
  clock++
  v.start = clock
```

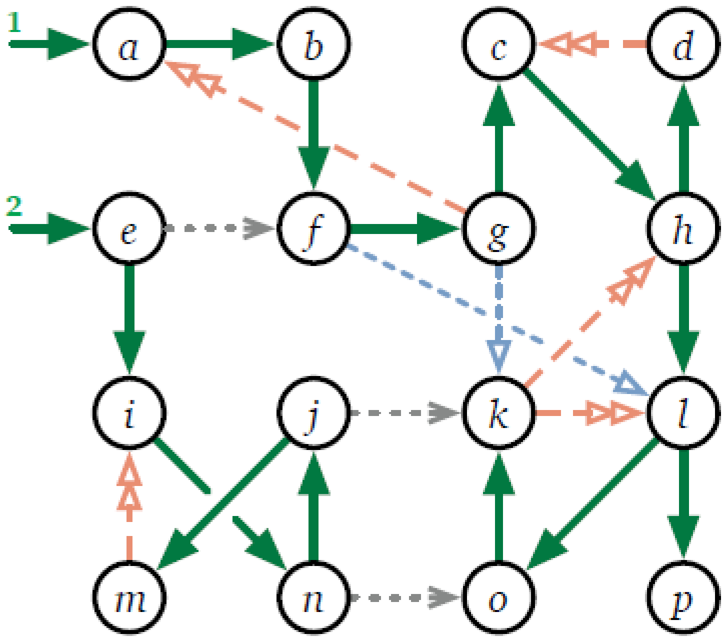
```
post(v)
  clock++
  v.finish = clock
```

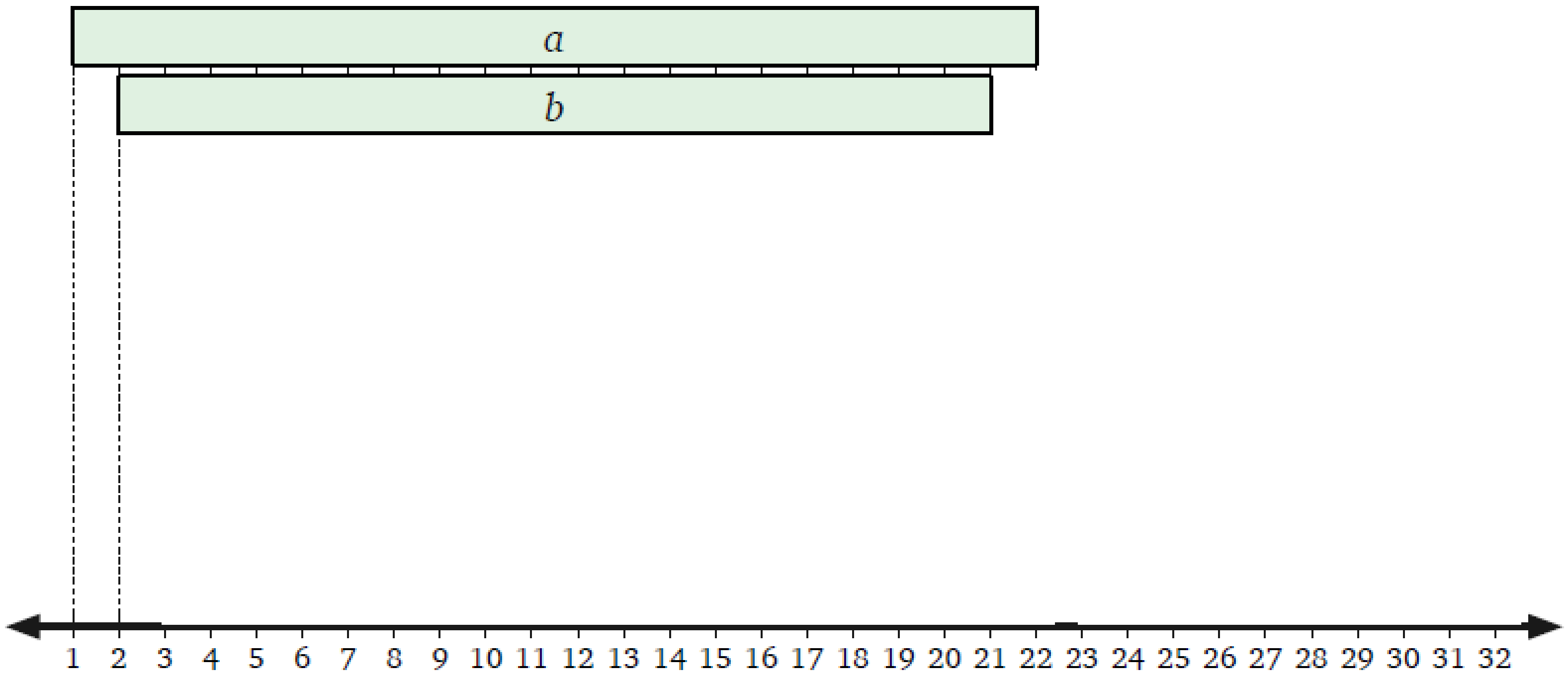
Recursion Stack - Time diagram



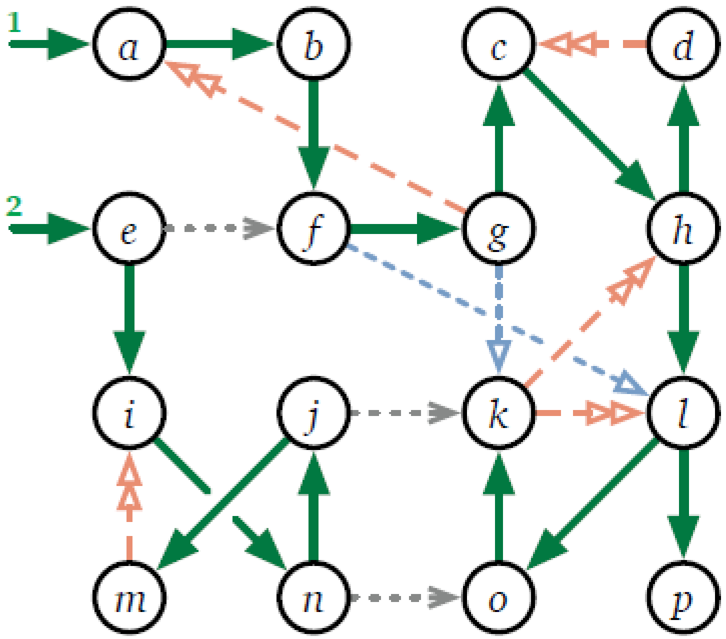


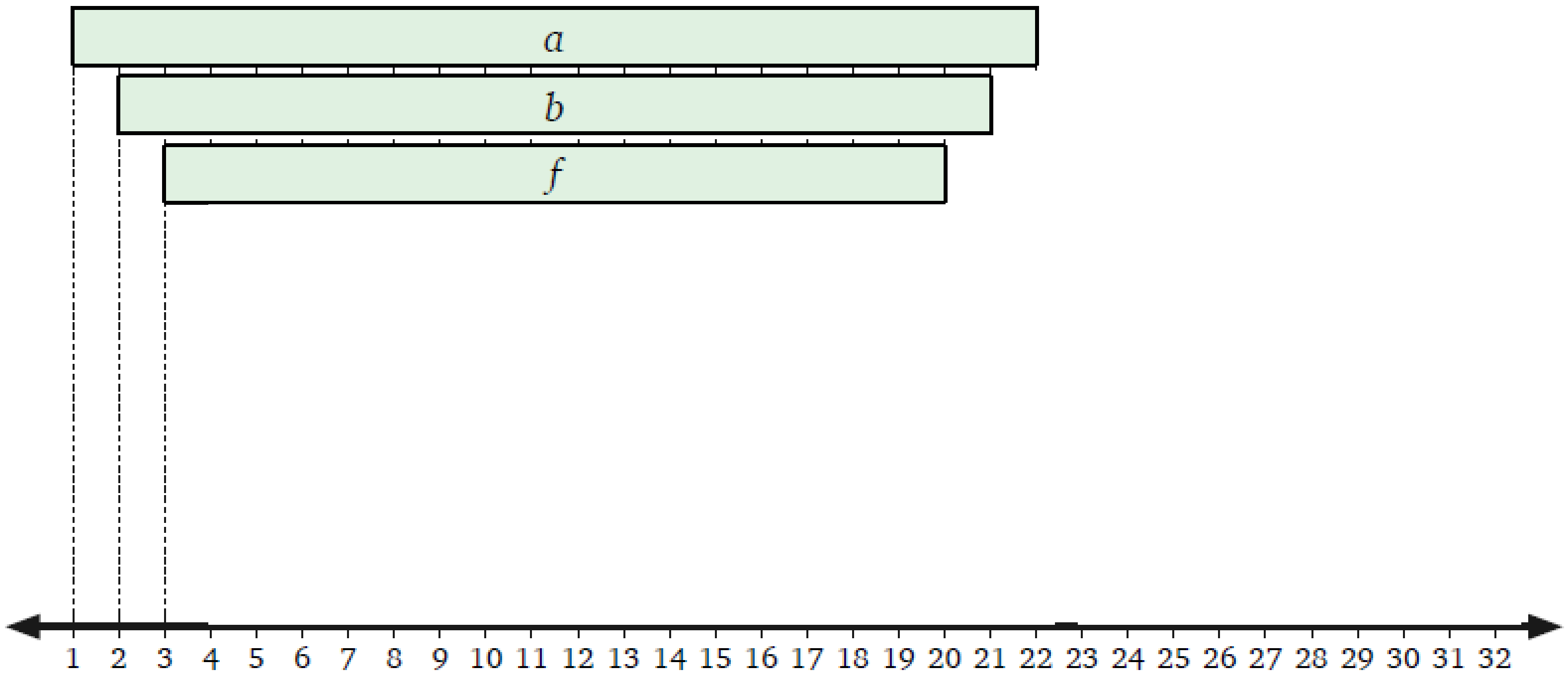
Stack - Time diagram



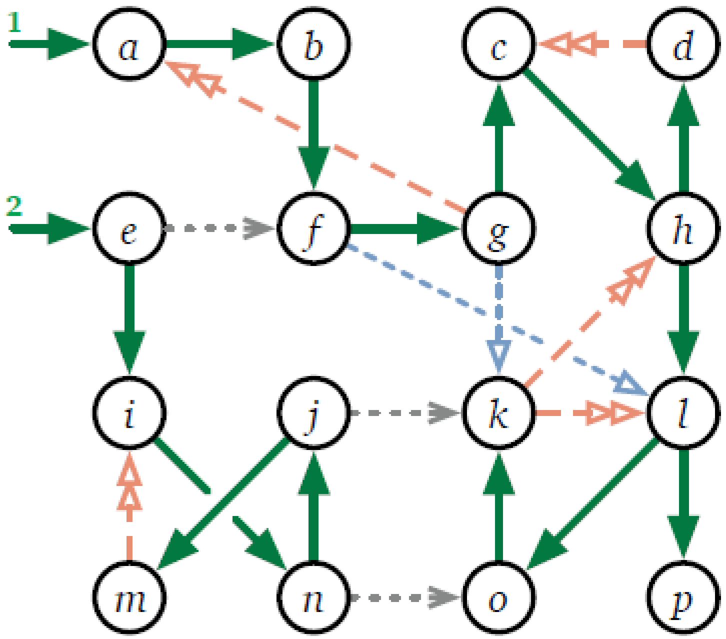


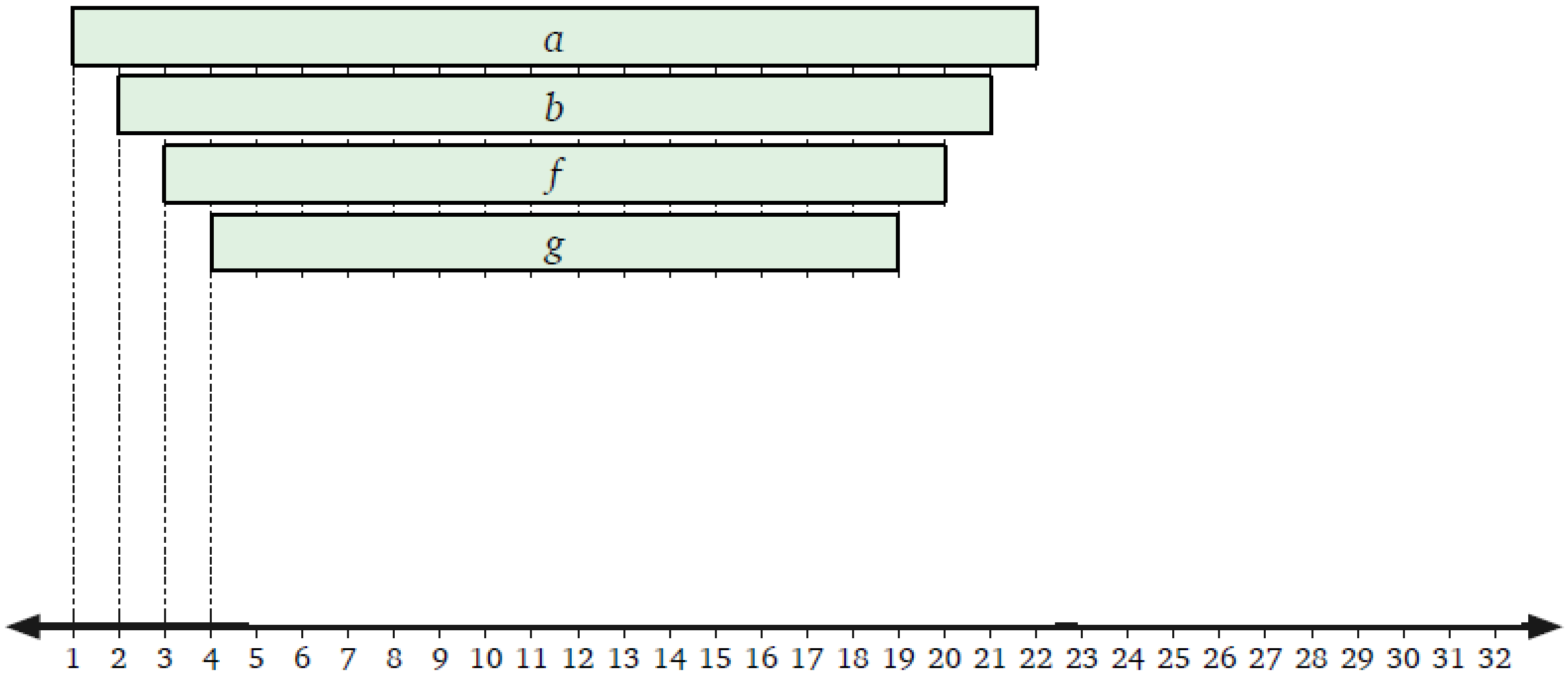
Stack - Time diagram



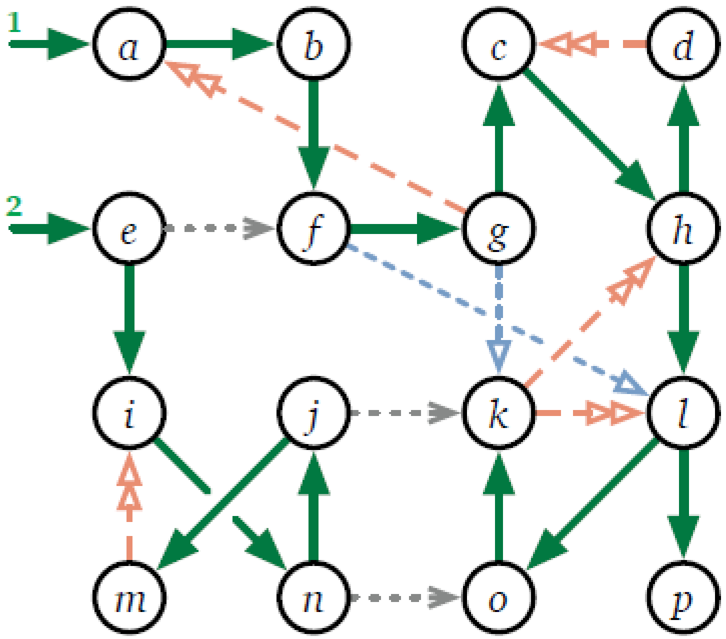


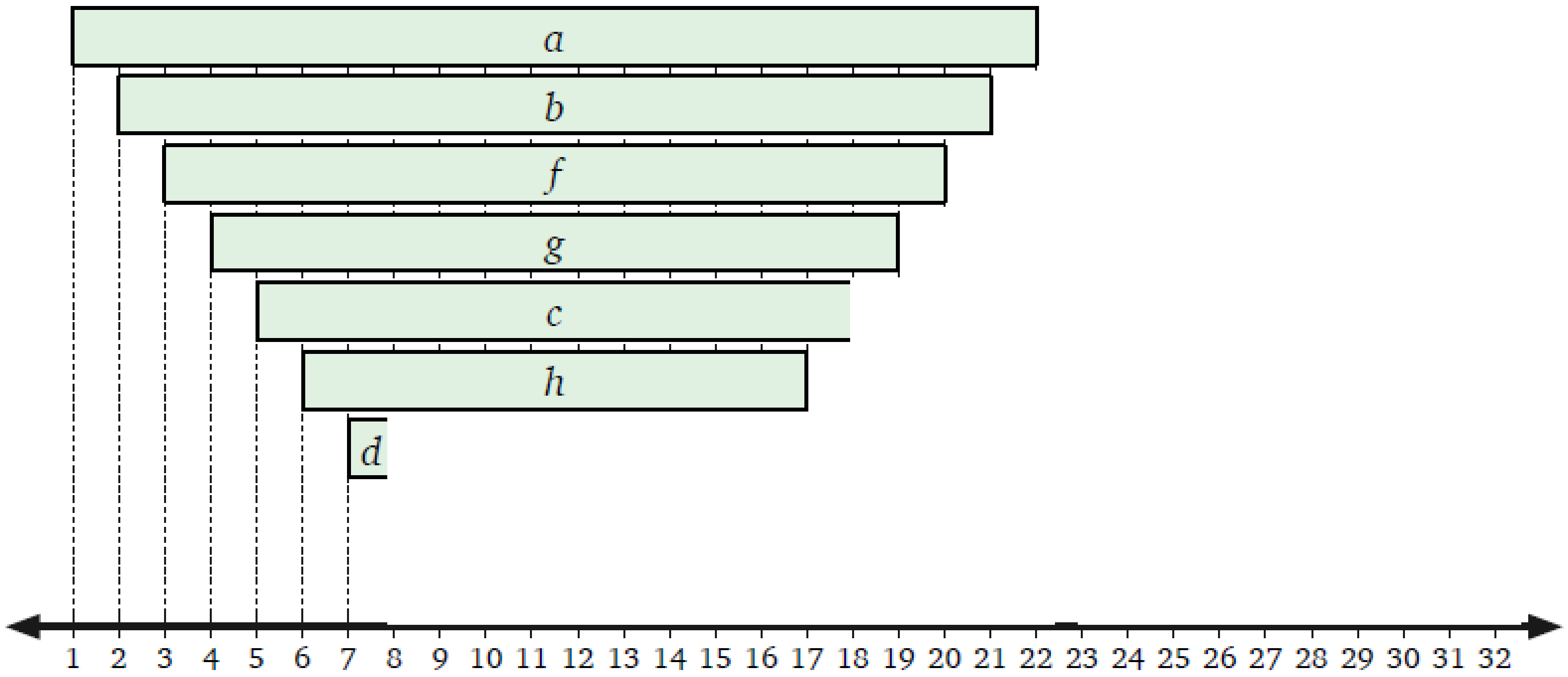
Stack - Time diagram



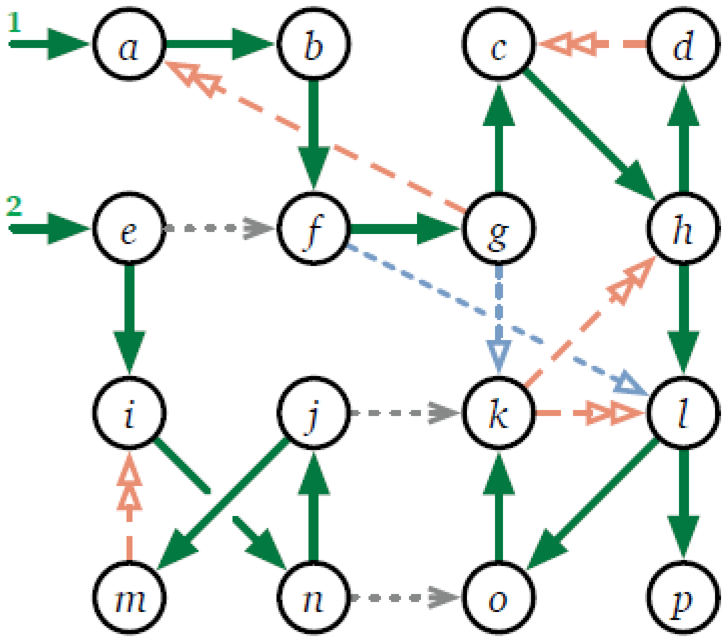


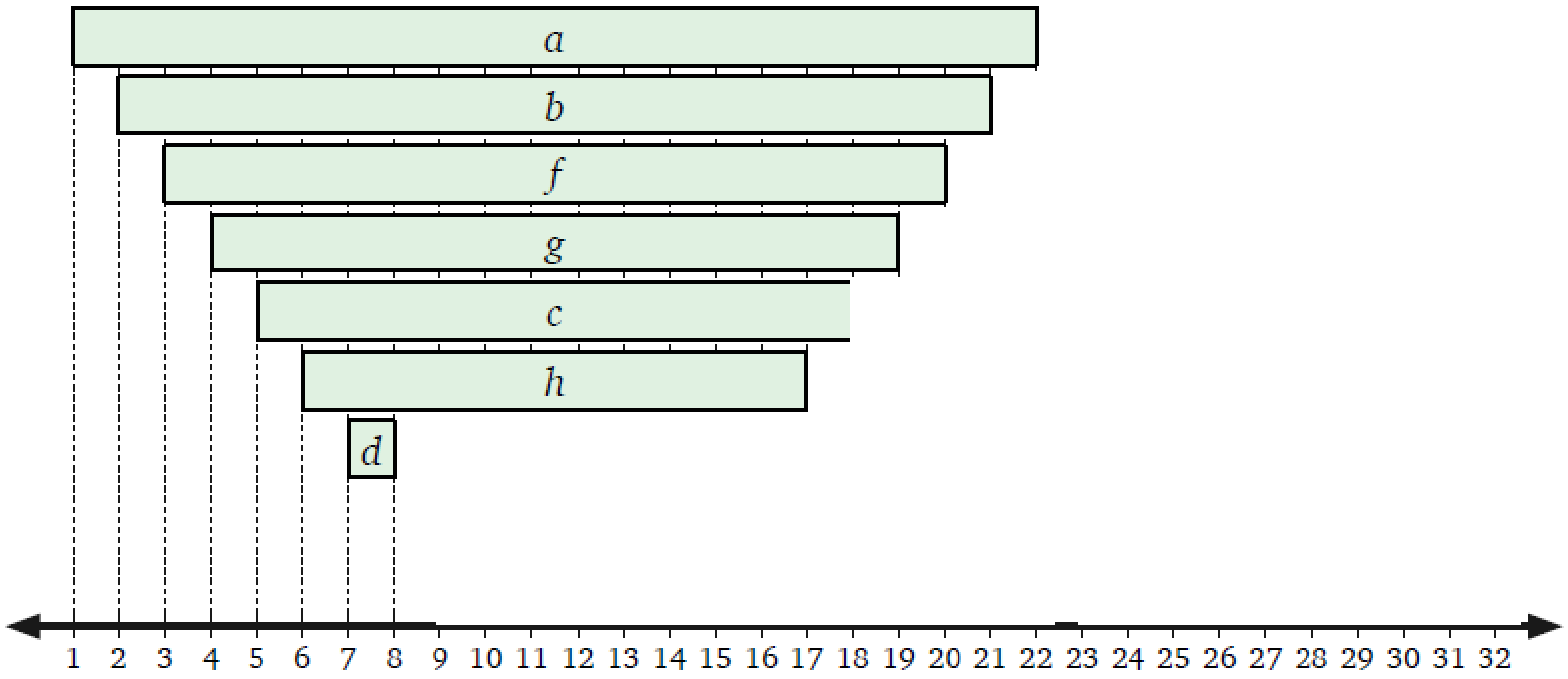
Stack - Time diagram



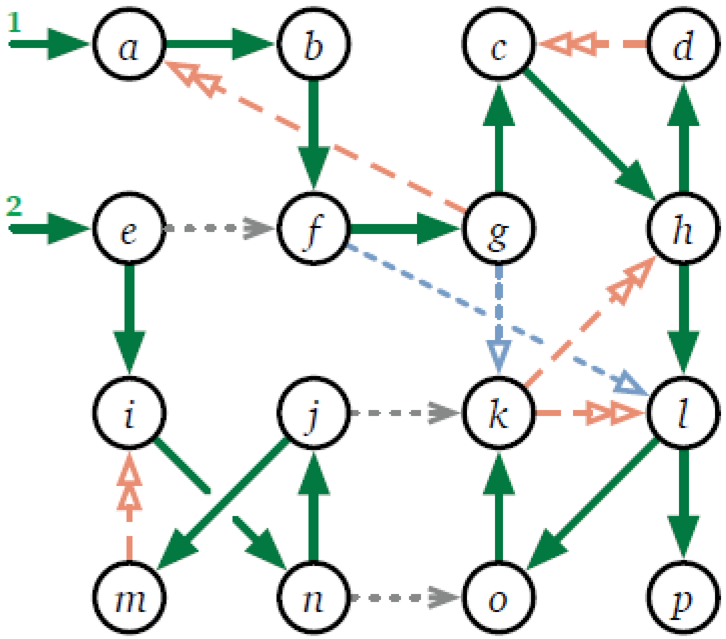


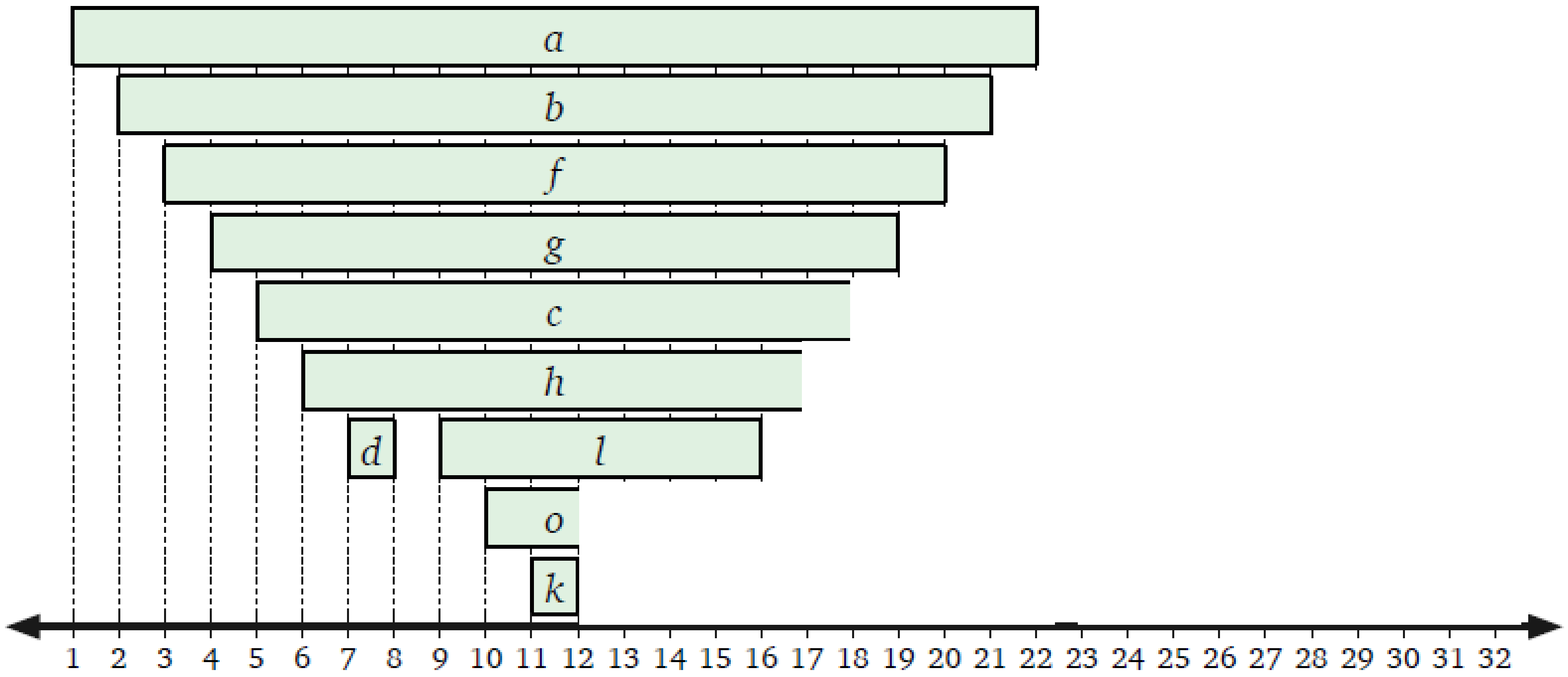
Stack - Time diagram



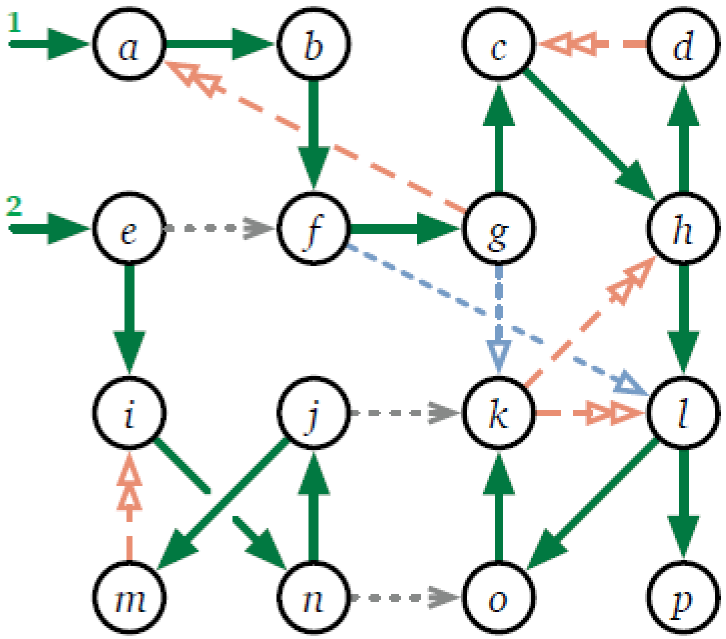


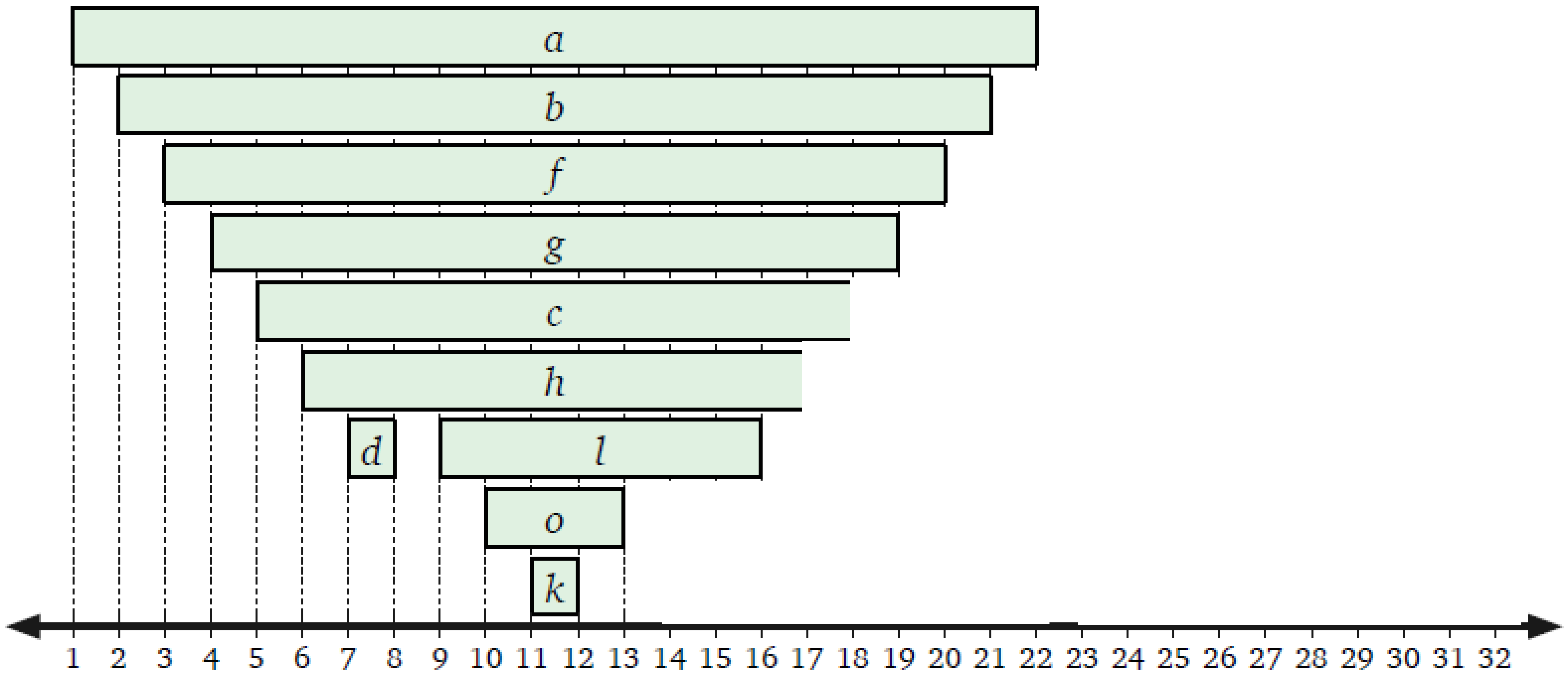
Stack - Time diagram



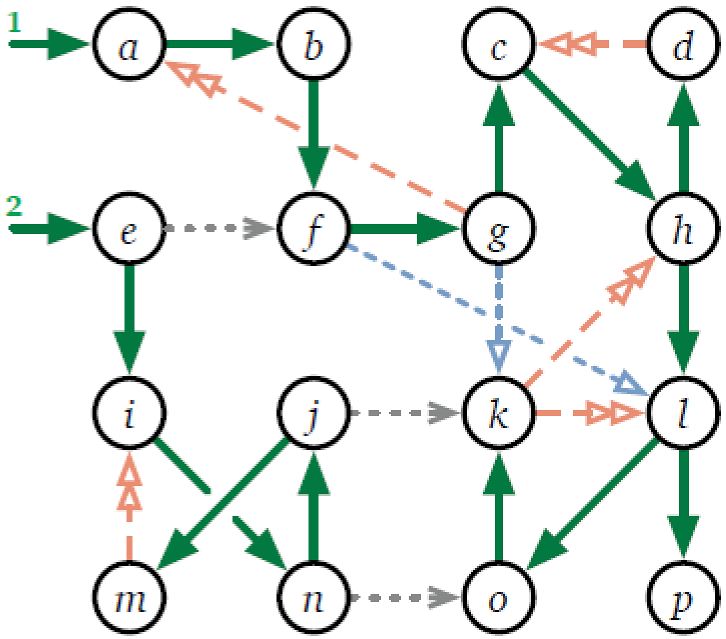


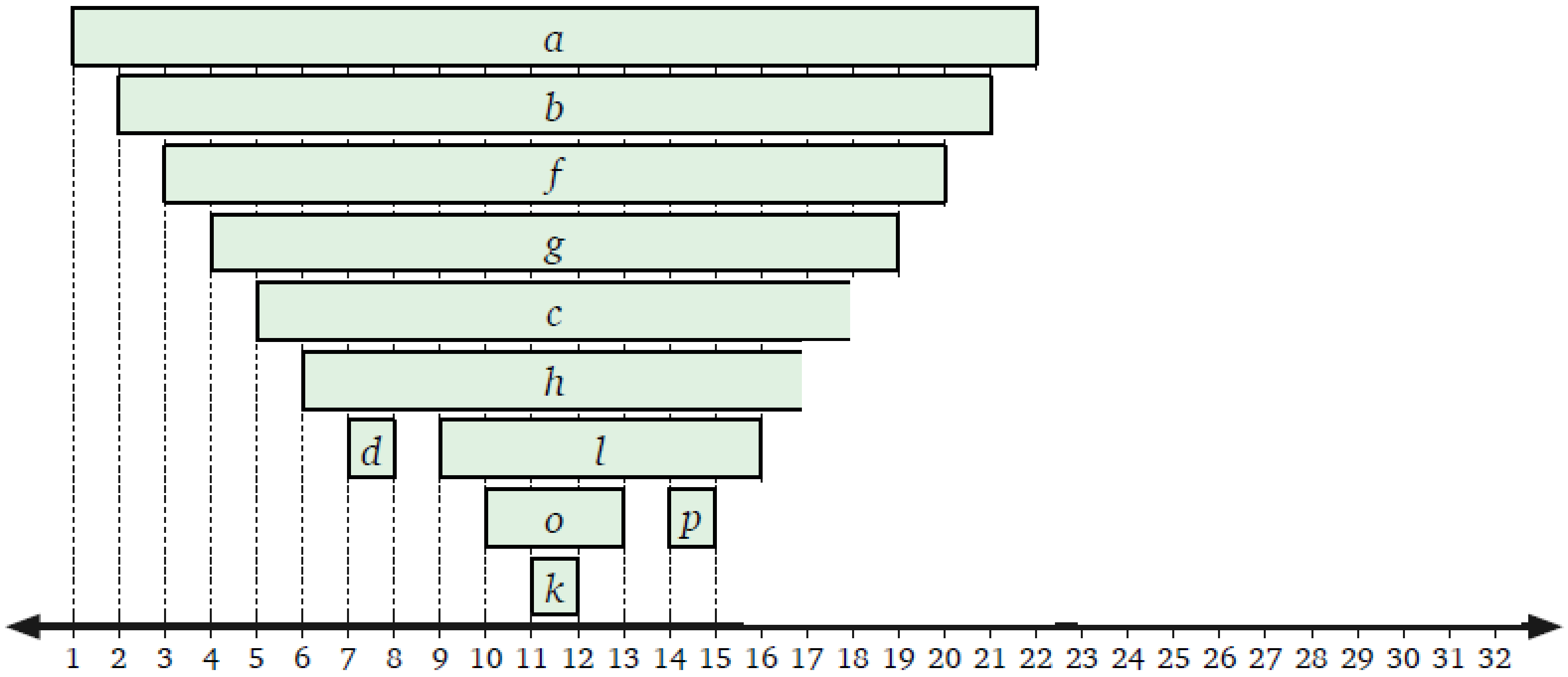
Stack - Time diagram



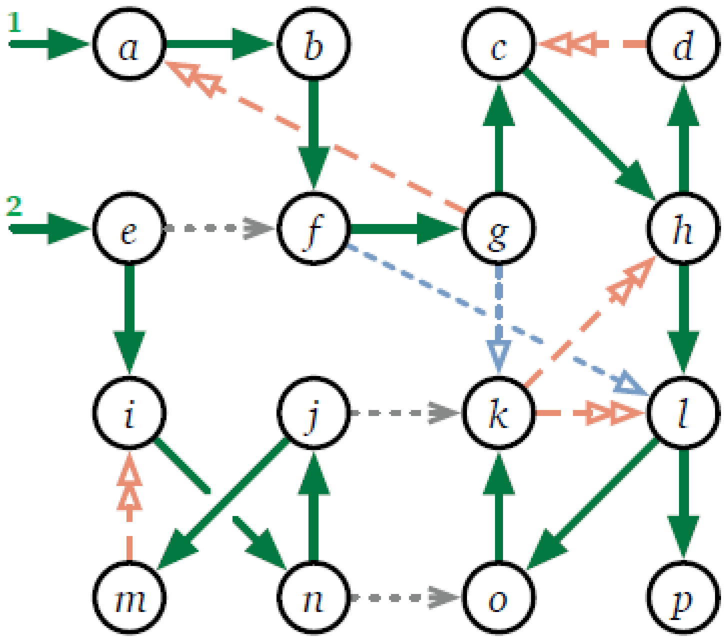


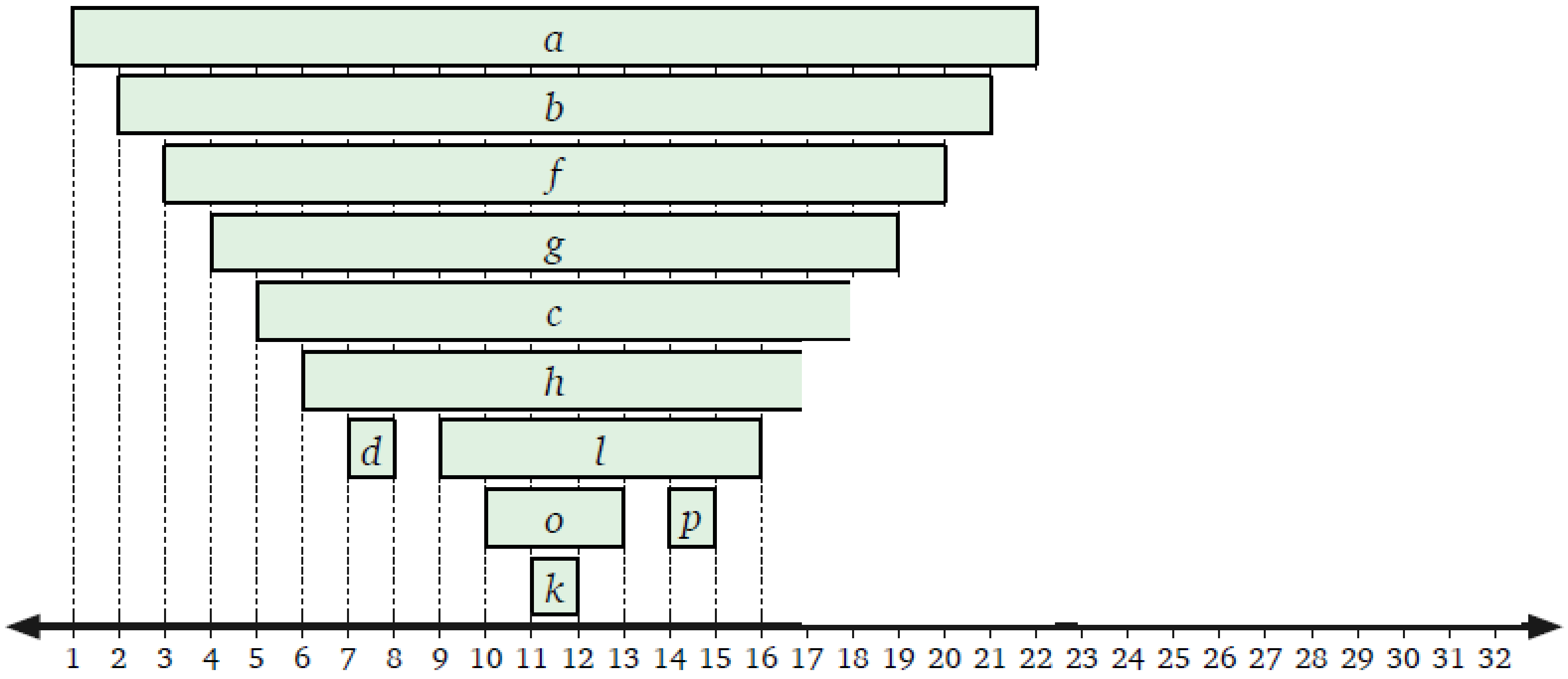
Stack - Time diagram



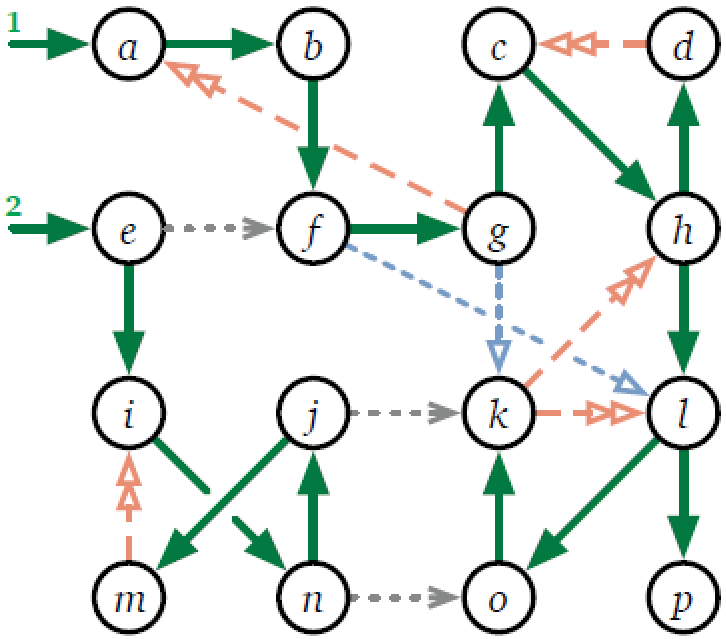


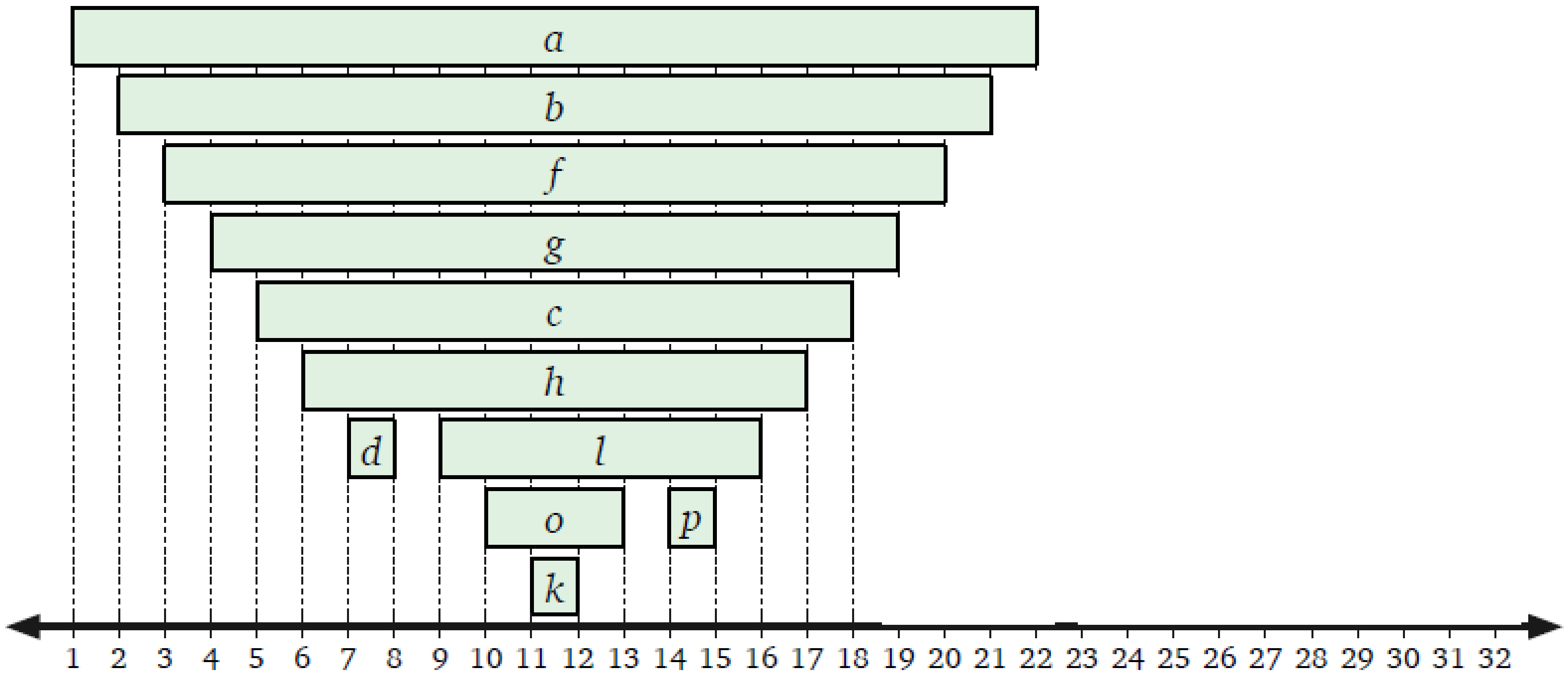
Stack - Time diagram



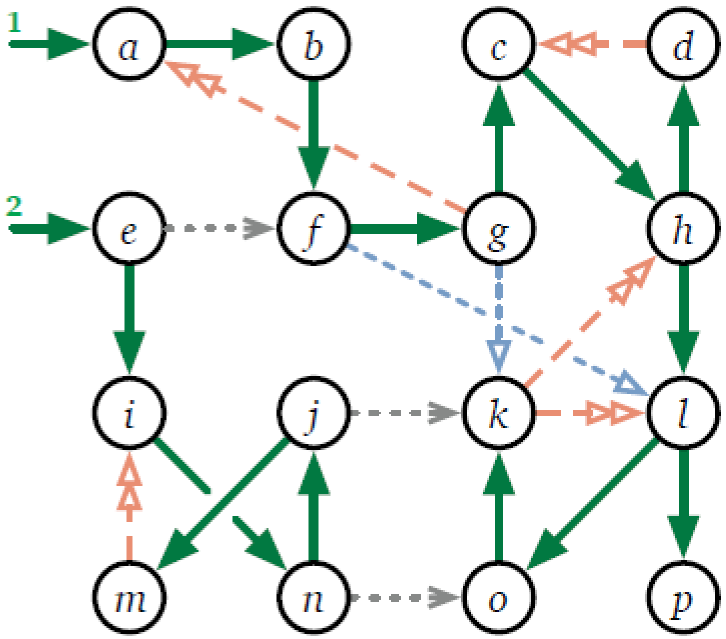


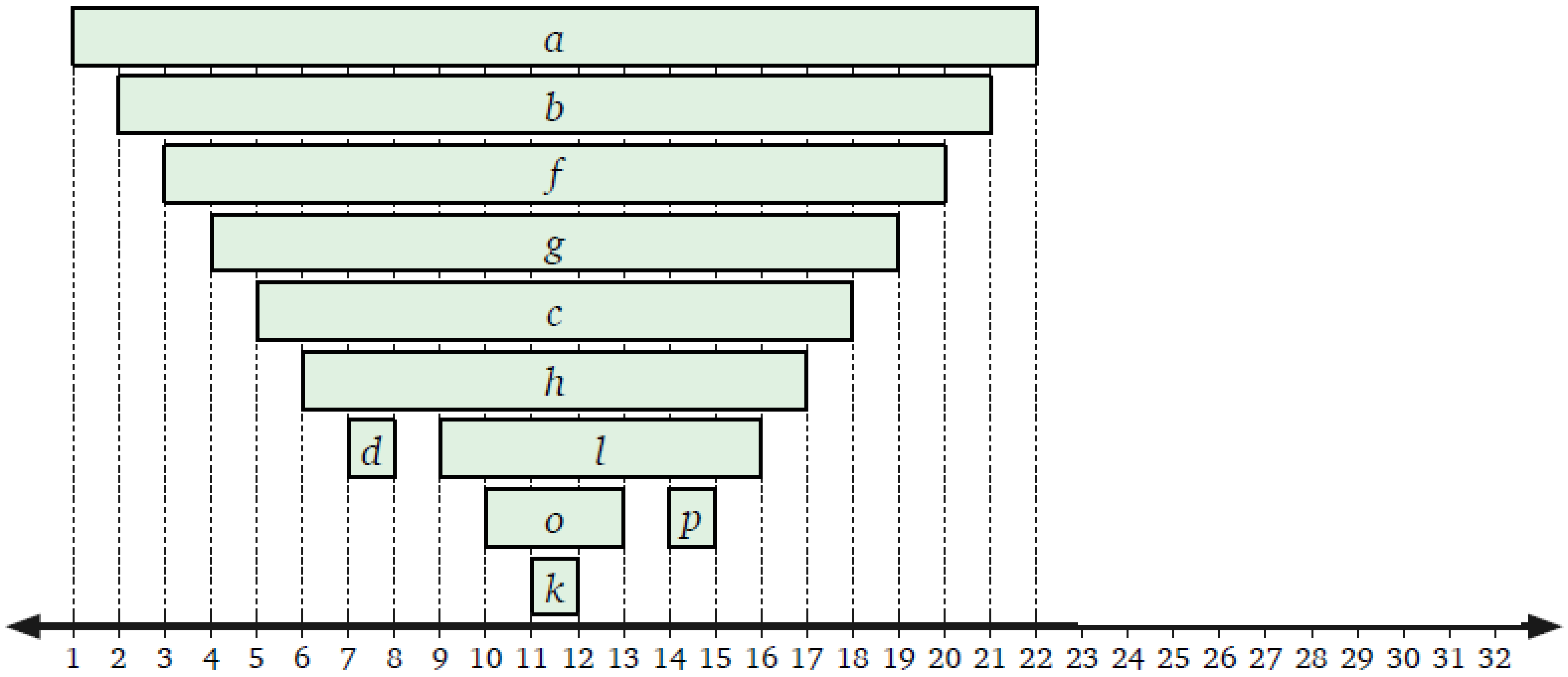
Stack - Time diagram



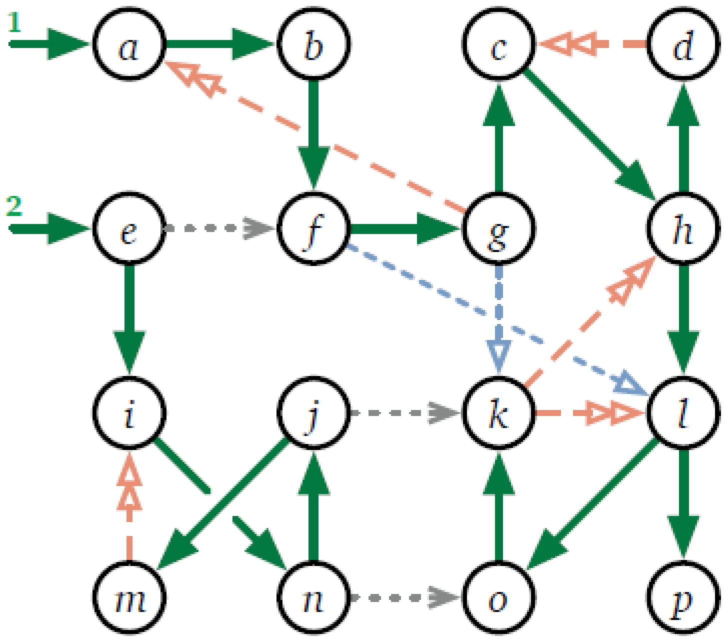


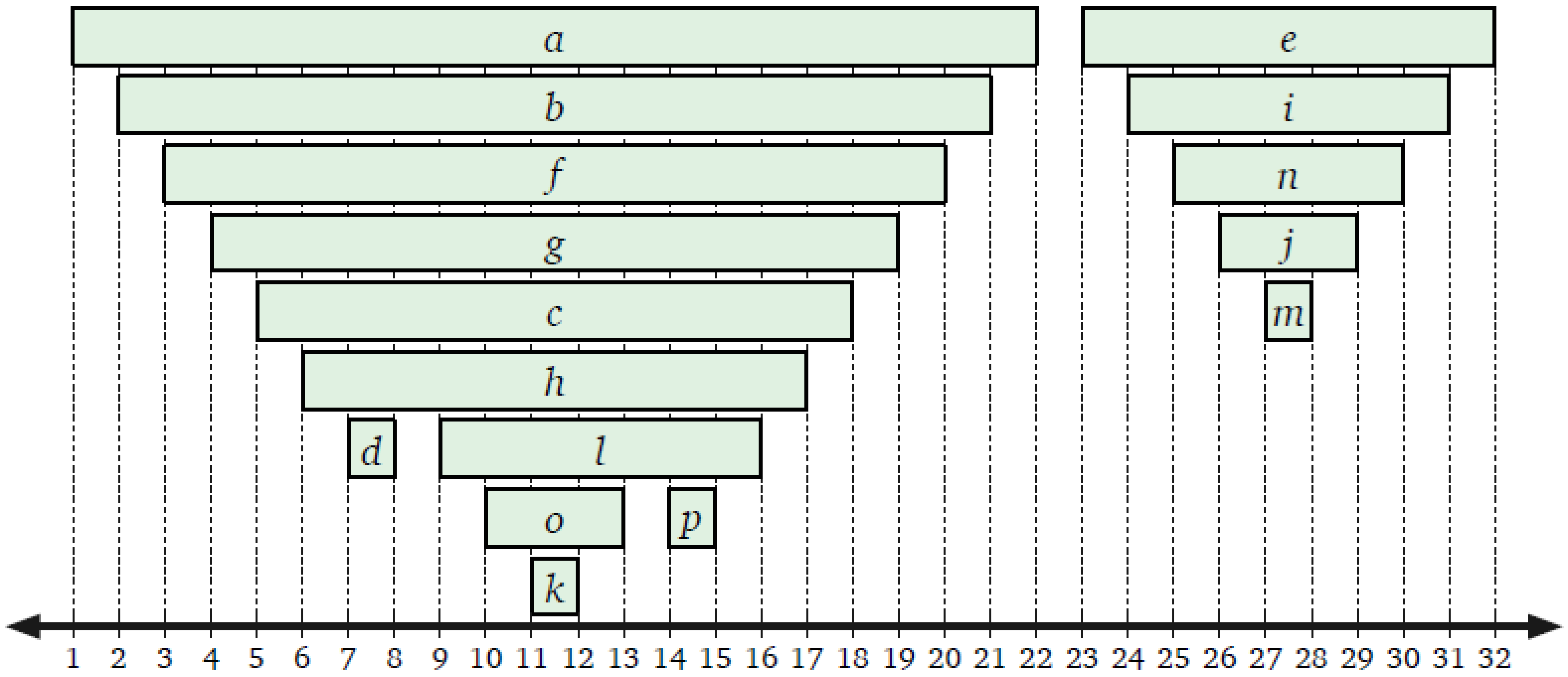
Stack – Time diagram



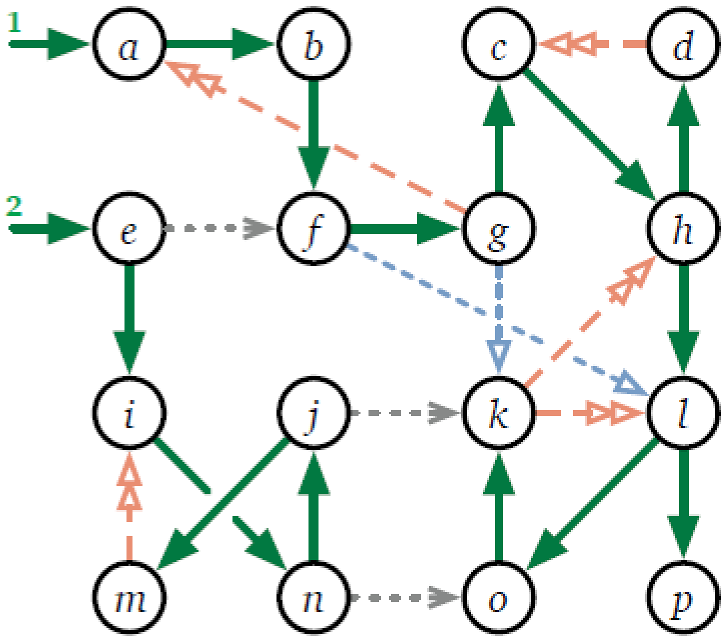


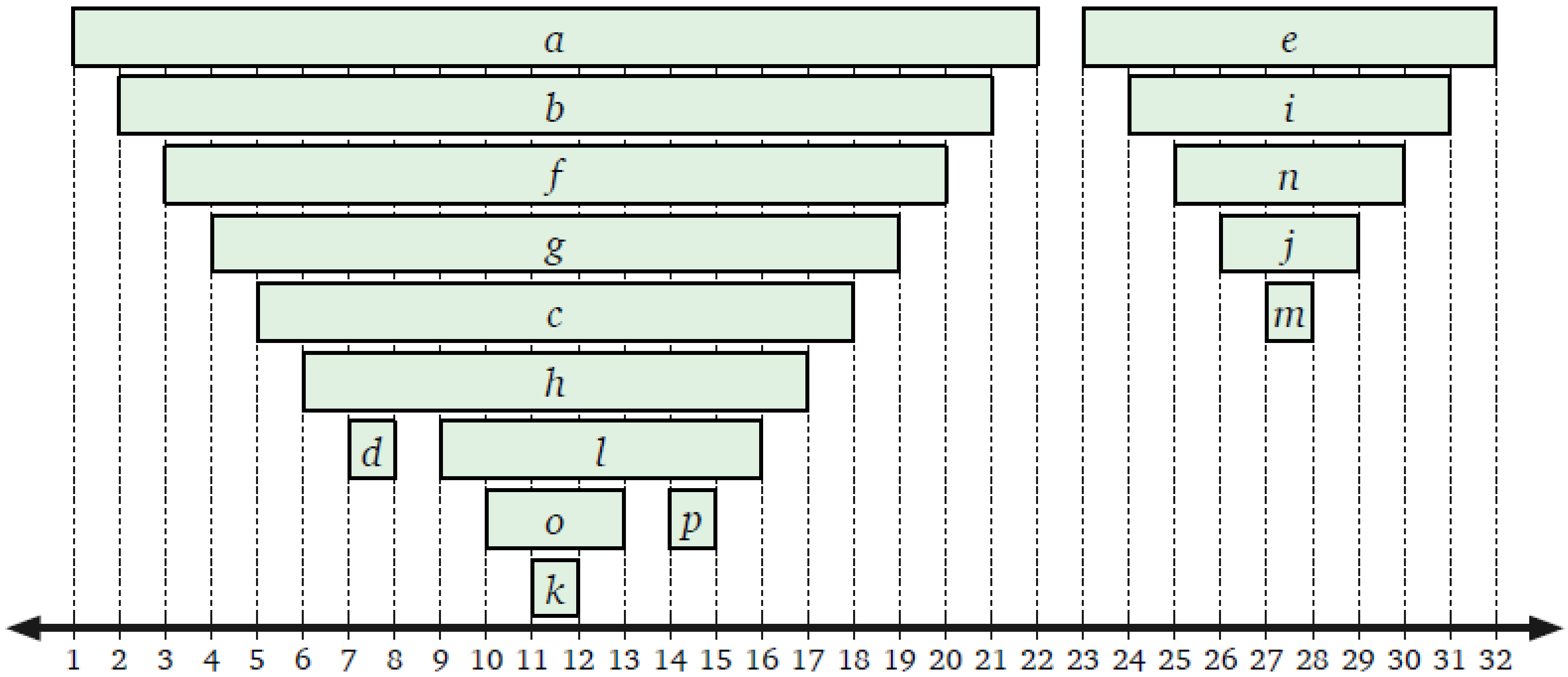
Stack - Time diagram





Preorder and postorder traversal of DFS Forest





Preorder a b f g c h d l o k p e i n j m

pre(v)

```
startList.add(v)
```

post(v)

```
finishList.add(v)
```

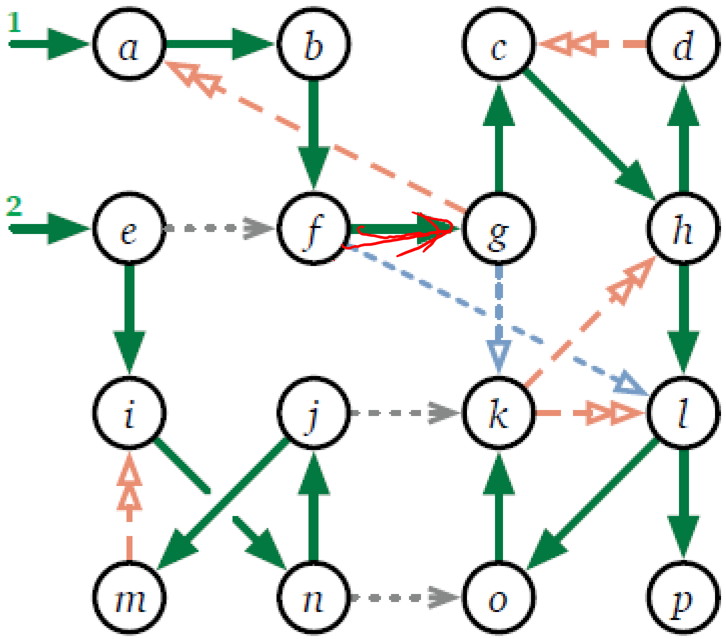
postorder d o p l

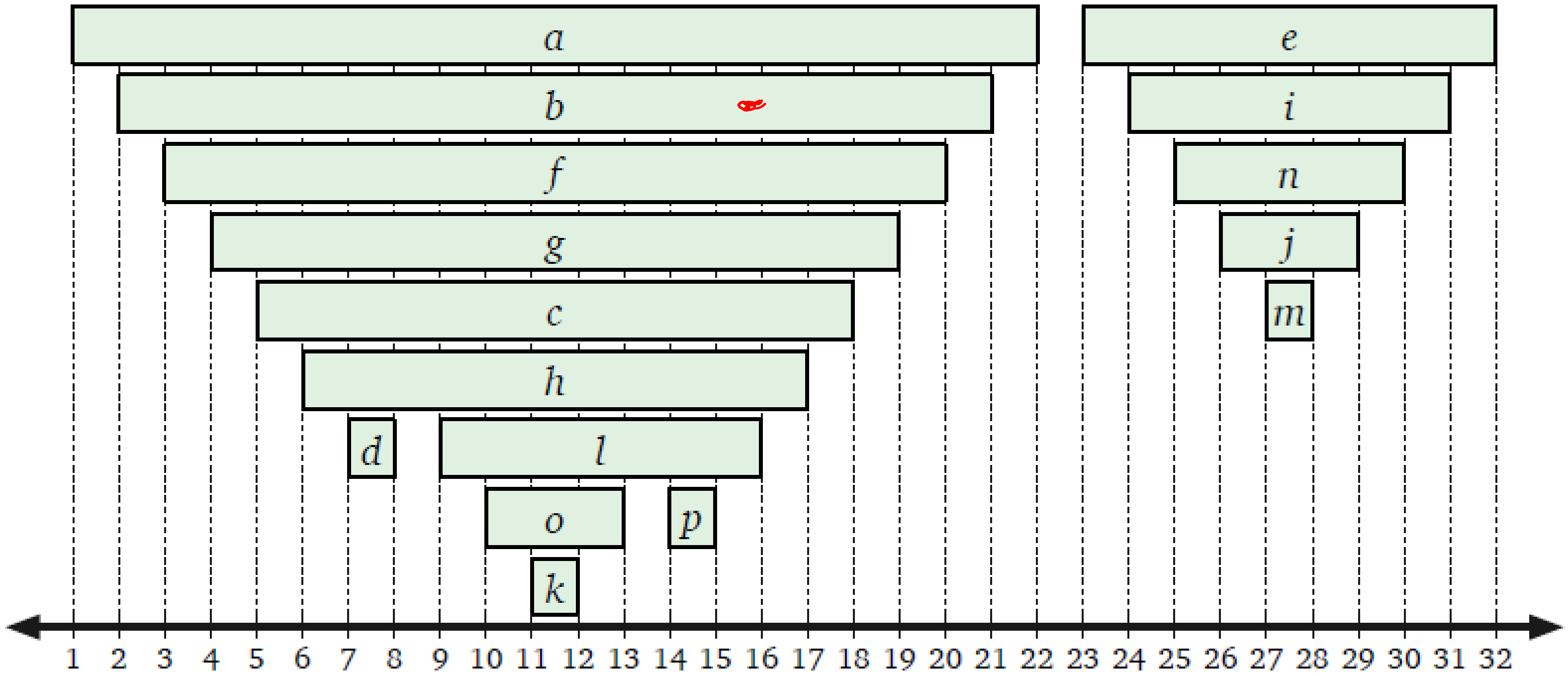
Classifying Vertices

- v is new if DFS(v) is not called
  - clock < v.start
- v is active if DFS(v) is called but not returned
  - v.start <= clock < v.finish
- v is finished if DFS(v) has returned
  - v.finish <= clock

Classify Edges

u, v .





- u.start < v.start < v.finish < u.finish
  - v is reachable from u
  - u is an ancestor of v in DFS tree

v is on top of u
on the stack

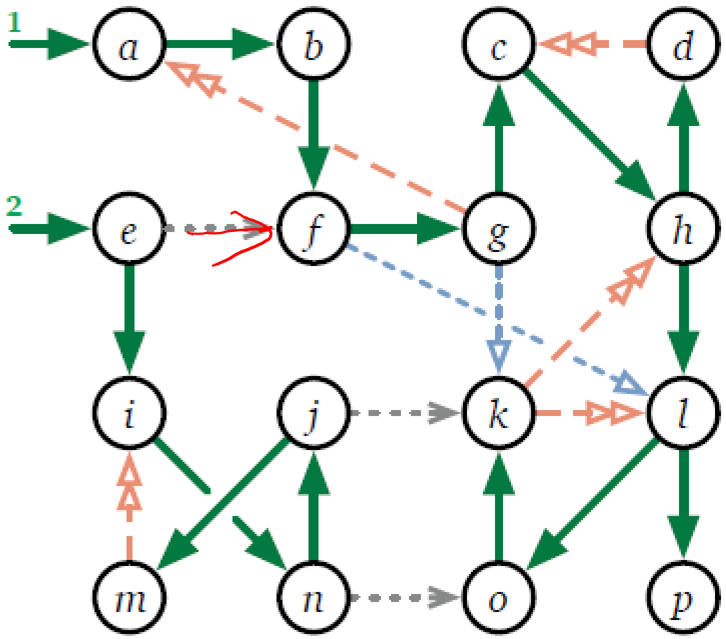
Classify Edges

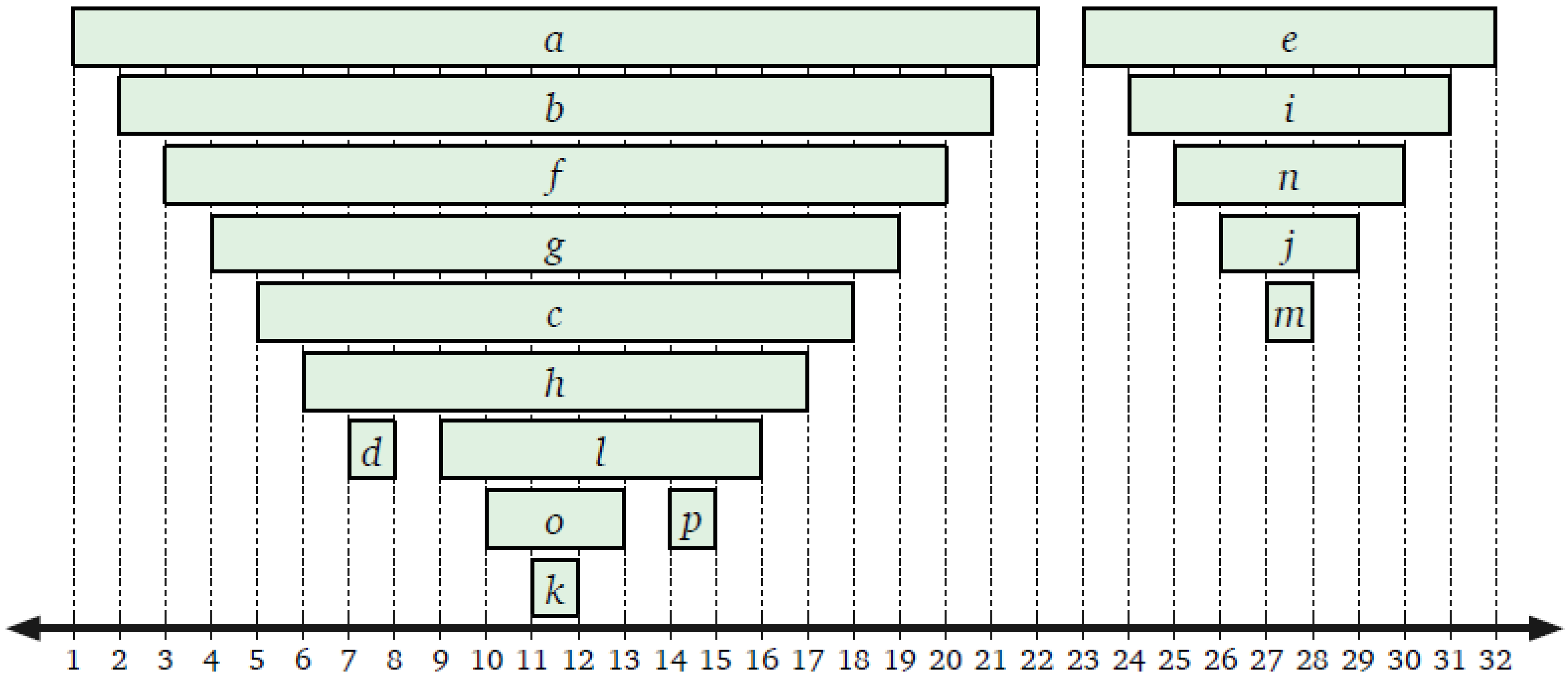
(f, l)   3 < 9 < 16 < 20

- consider u->v and suppose v is new when DFS(u) begins
  - u.start < v.start < v.finish < u.finish
  - If DFS(u) calls DFS(v) directly, then u->v is a tree edge
  - Otherwise u->v is a forward edge

Classify Edges

f
3 < 20 < 23 < 32.

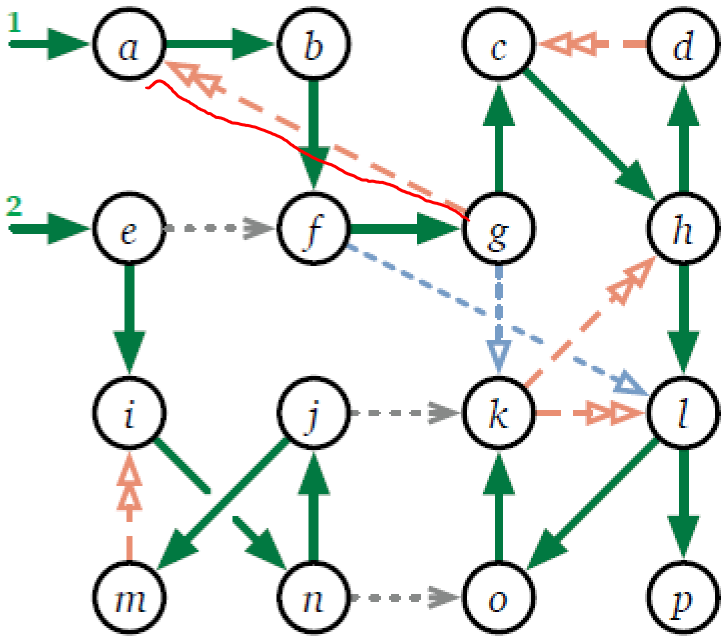


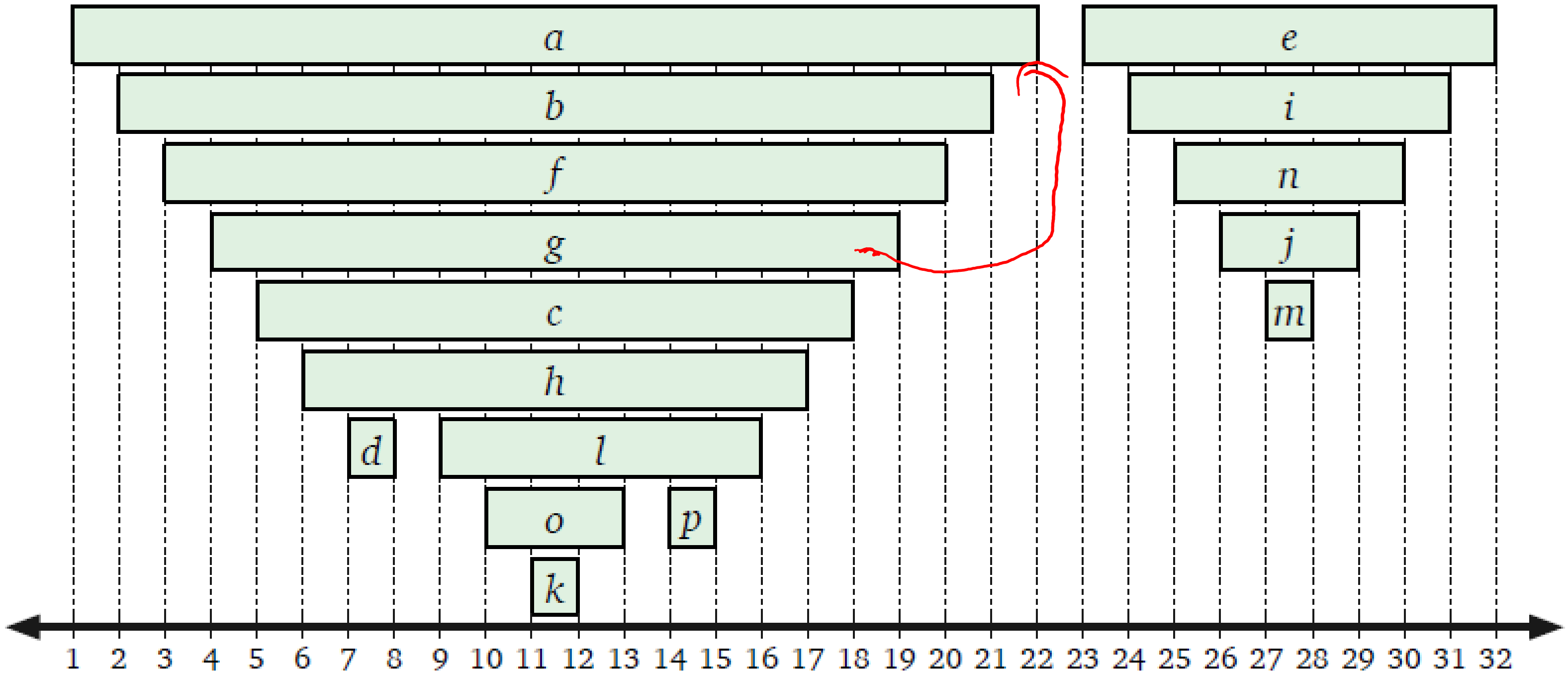


- consider u->v and v is finished when DFS(u) begins
  - v.start < v.finish < u.start < u.finish, then u->v is a cross edge

Classify Edges

1 < 4 < 19 < 22

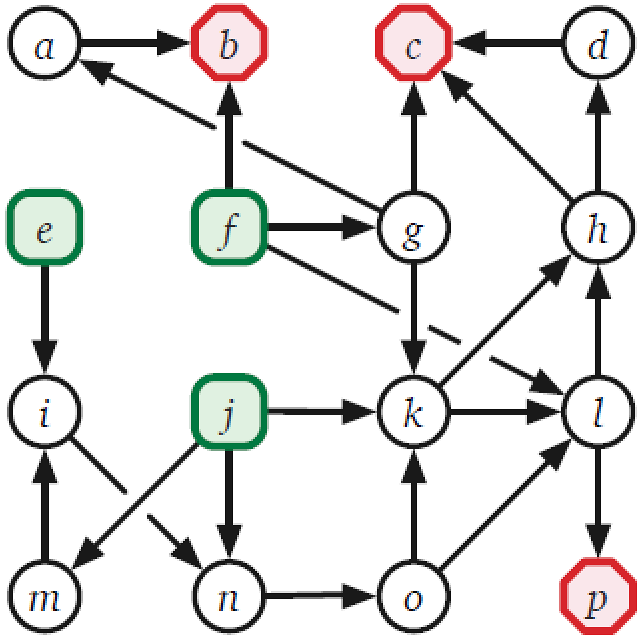




- consider u->v and v is active when DFS(u) begins
  - v.start < u.start < u.finish < v.finish ,
  - u->v is a back edge

Directed Acyclic Graph

- Source vertices have no incoming edges
- Sink vertices have no outgoing vertices
- DAG has at least one source vertex and one sink vertex
- Is G a DAG?



Is G a DAG?

What would you change to the code below?

```
dfsAll(G){
    initialize(G)
    for each v in G do
      unmark v
    for each v in G d0
      if v is unmarked
          dfs(v)
}
```

```
dfs(v){
  mark v
  pre(v)
  for each (v, w) in G do
      if w is unmarked
            w.parent = v
            dfs(w)
  post(v)
}
```

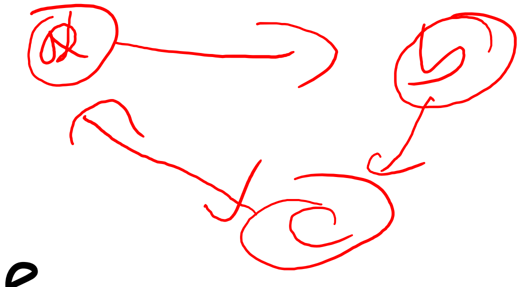
Is G a DAG?

A linear time algorithm:

```
isDAGAll(G){
   for each v in G do
     v.status = new
   for each v in G d0
     if v.status is new
         if(!isDAG(v)) return false
   return true
}
```

```
isDAG(v){
  v.status = active
  for each (v, w) in G do
      if w.status is active
            return false
      else if w.status is new
            if (!isDAG(w))
                  return false
  v.status = finished
  return true
}
```

Topological Ordering

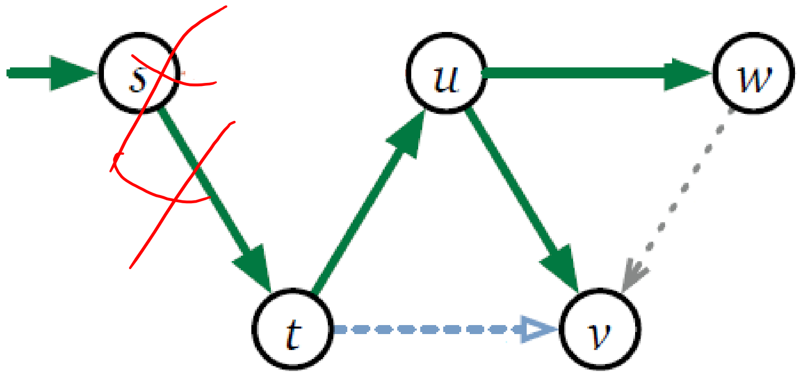


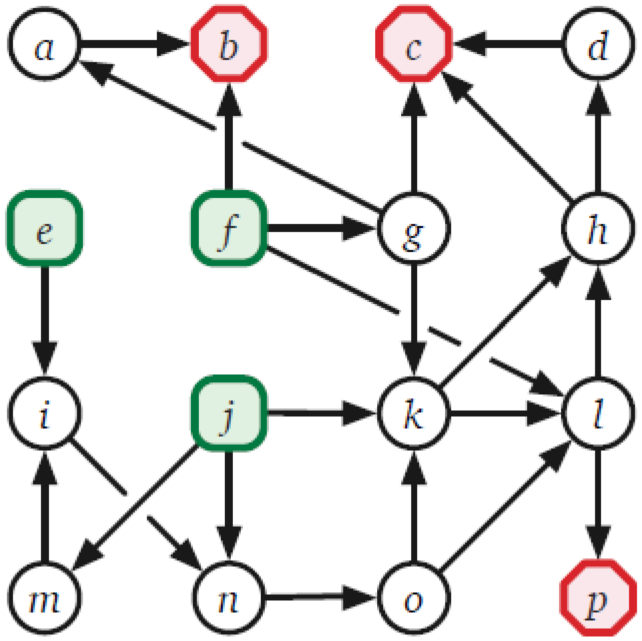
- Informally, place all vertices in a horizontal line such that edges go only from left to right

- Topological ordering of G is a total order <, on vertices such that
  - u < v for every edge u->v

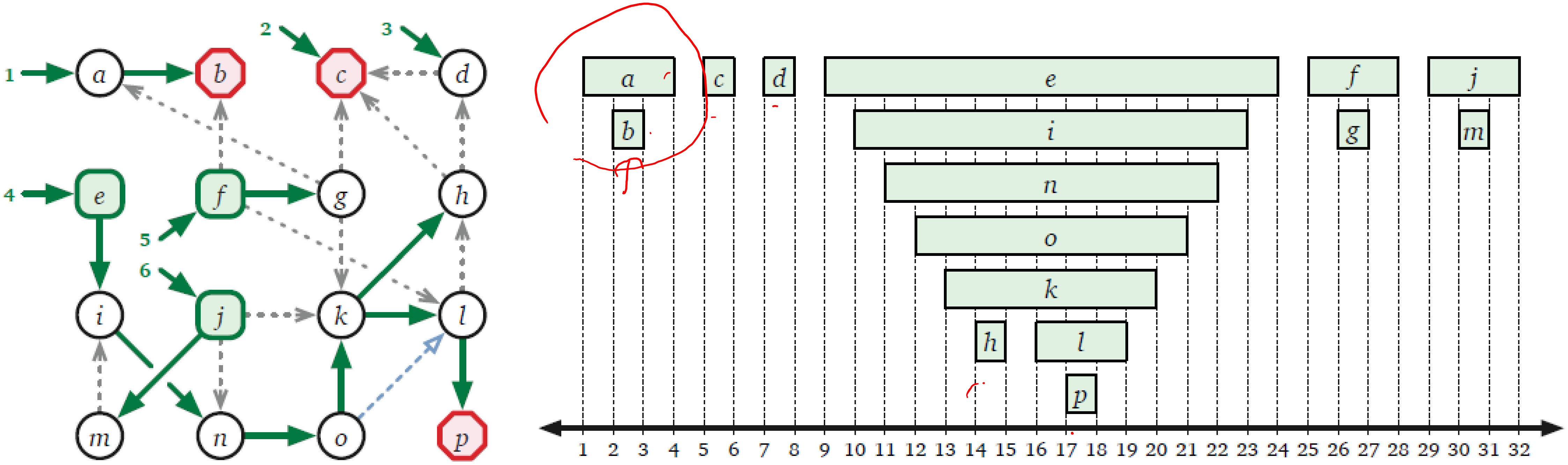
- Topological ordering is only possible if G is a DAG

Topological Sort





Topological Sort



cab

Topological Sort Algorithm

Reverse of post order traversal

```
dfsAll(G){
    initialize(G)
    for each v in G do
      unmark v
    for each v in G d0
      if v is unmarked
          dfs(v)
}
```

```
dfs(v){
  mark v
  pre(v)
  for each (v, w) in G do
      if v is unmarked
            w.parent = v
            dfs(w)
  post(v)
}
```

```
initialize(G)
```

```
pre(v)
```

```
post(v)
  topList.addFirst(v)
```

Problem of the day

Let G be an undirected graph. Suppose we start with two coins on two arbitrarily chosen vertices of G. At every step, each coin ***must*** move to an adjacent vertex. Describe and analyze an efficient algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph $G = (V, E)$ and two vertices $u, v \in V$ (which may or may not be distinct).