CS250: Discrete Math for Computer Science

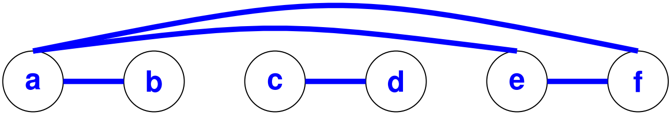
L29: DFS on Directed Graphs

Depth First Search (undirected graphs)

```
DFSmain(G)
 for each u in V :
   color[u] = white
   parent[u] = NULL
 time=0
 for each u in V :
   if (color[u] == white):
     DFSVisit(u)
```

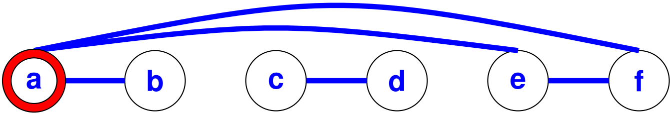
```
DFSVisit(u)
  color[u] = red               // in process
  d[u] = ++time                // discover
  for each v in Adj(u) :
    if (color[v] == white) :   // unseen
      parent[v] = u            // tree edge
      DFSVisit(v)
  color[u] = black             // done
  f[u] = ++time                // finish
```

black Tree edge brown Back edge

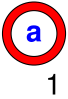



```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```

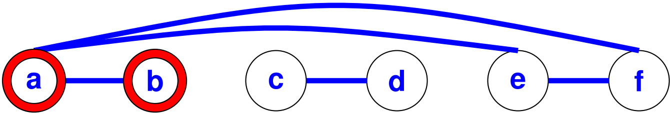
black Tree edge brown Back edge



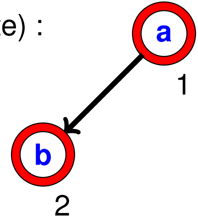

```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```



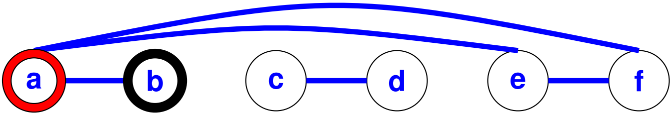
black Tree edge brown Back edge



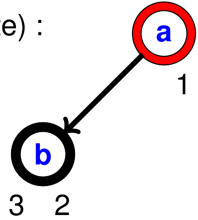

```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```



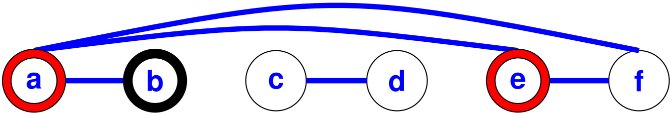
black Tree edge brown Back edge



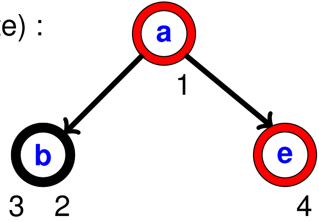

```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```



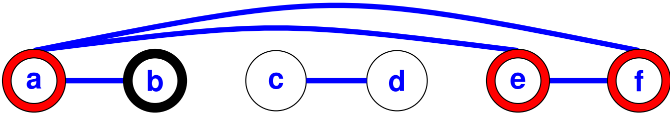
black Tree edge   brown Back edge




```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```



black Tree edge   brown Back edge



DFSVisit(u)

color[u] = **red**

d[u] = ++time

for each v in Adj(u) :

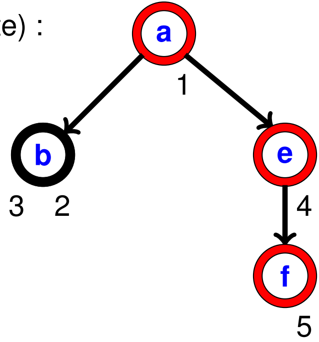
if (color[v] == white) :

parent[v] = u

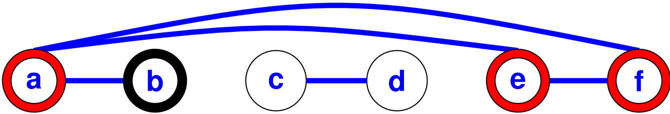
DFSVisit(v)

color[u] = **black**

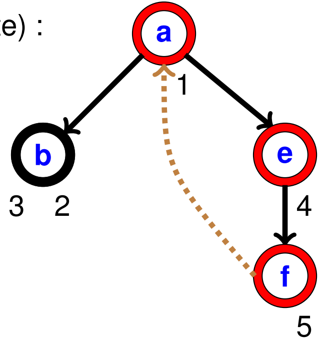
f[u] = ++time



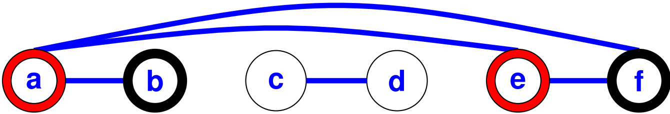
black Tree edge brown Back edge




```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```



black Tree edge   brown Back edge



DFSVisit(u)

color[u] = **red**

d[u] = ++time

for each v in Adj(u) :

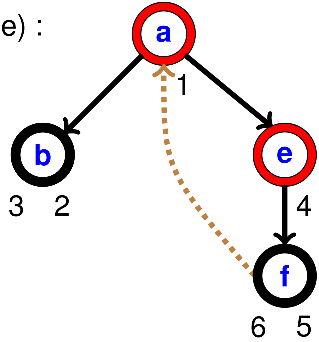
if (color[v] == white) :

parent[v] = u

DFSVisit(v)

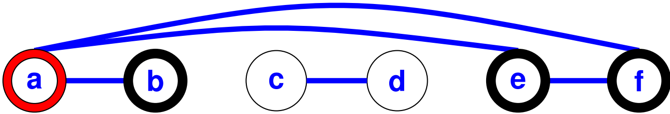
color[u] = **black**

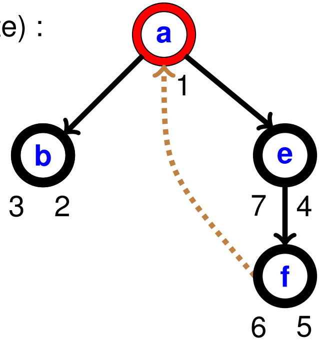
f[u] = ++time



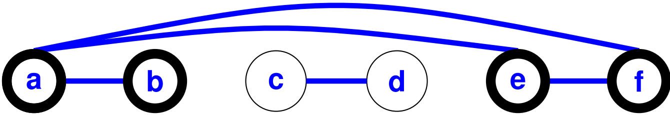
black Tree edge brown Back edge

```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```

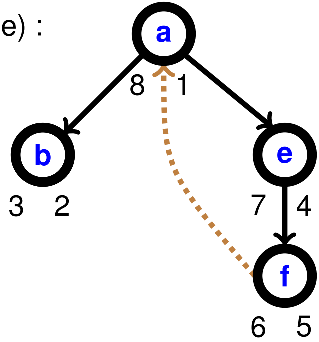




black Tree edge   brown Back edge

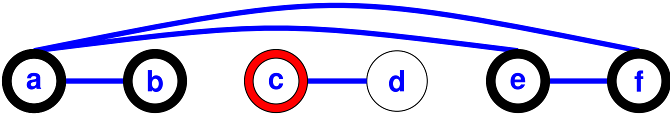


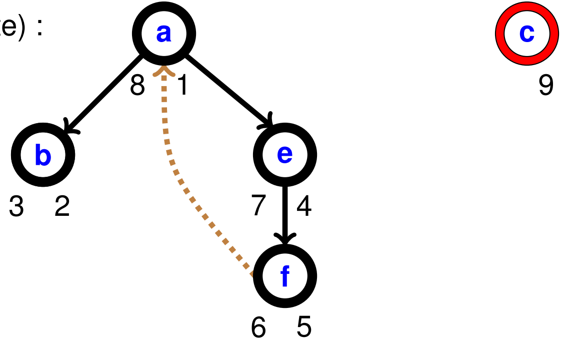

```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```



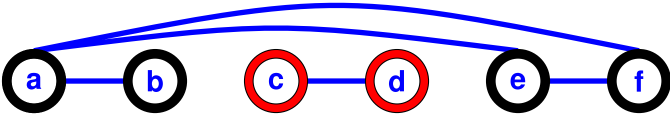
black Tree edge   brown Back edge

```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```

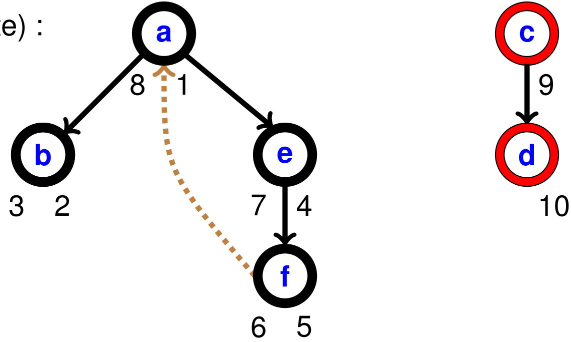




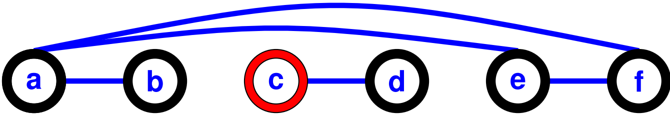
black Tree edge brown Back edge



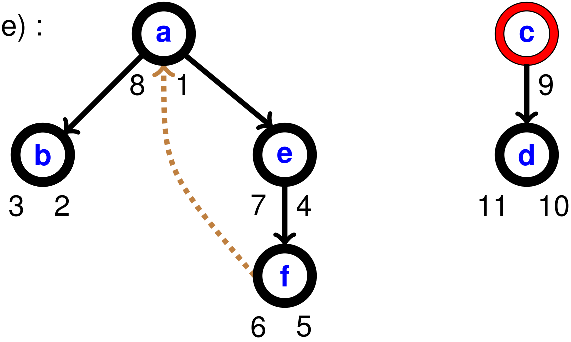

```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```



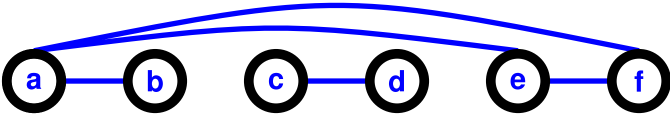
black Tree edge   brown Back edge



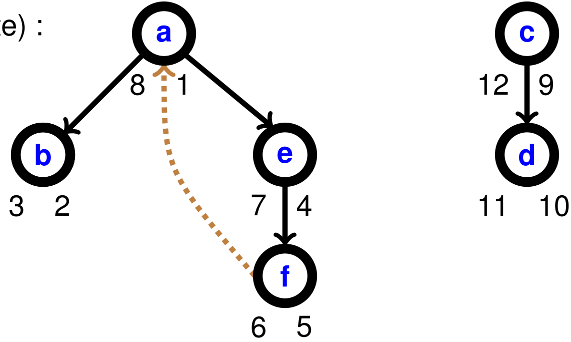

```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```



black Tree edge brown Back edge




```
DFSVisit(u)
  color[u] = red
  d[u] = ++time
  for each v in Adj(u) :
    if (color[v] == white) :
      parent[v] = u
      DFSVisit(v)
  color[u] = black
  f[u] = ++time
```



Depth First Search (undirected graphs)

```
DFSmain(G)
 for each u in V :
   color[u] = white
   parent[u] = NULL
 time=0
 for each u in V :
   if (color[u] == white):
     DFSVisit(u)
```

```
DFSVisit(u)
 color[u] = red                    // in process
 d[u] = ++time                     // discover
 for each v in Adj(u) :
   if (color[v] == white) :        // unseen
     parent[v] = u                 // tree edge
     DFSVisit(v)
 color[u] = black                  // done
 f[u] = ++time                     // finish
```

Thm. (Properties of DFS on Undirected Graphs)
Let G be an undirected graph with n vertices and m edges.
Then:

Thm. (Properties of DFS on Undirected Graphs)
Let G be an undirected graph with n vertices and m edges.
Then:

1. DFS(G) runs in **linear time**, i.e., $O(n + m)$.

Thm. **(Properties of DFS on Undirected Graphs)**
Let G be an undirected graph with n vertices and m edges.
Then:

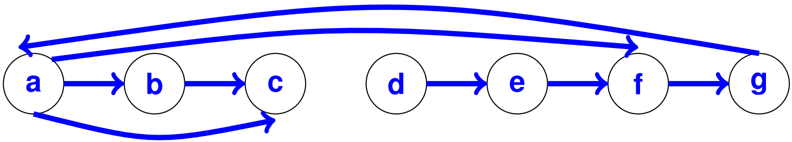
1. DFS(G) runs in **linear time**, i.e., $O(n + m)$.

2. DFS computes **connected components** of G.

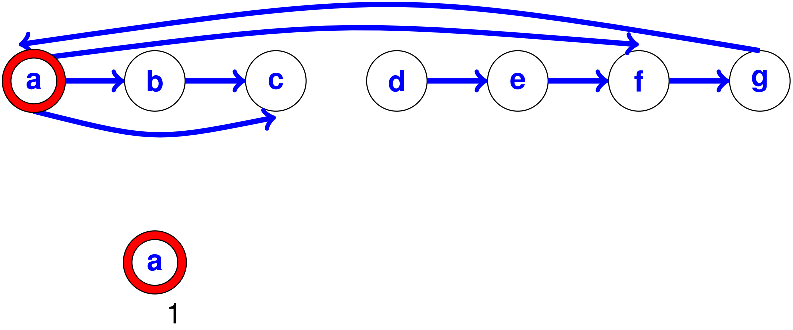
Thm. (Properties of DFS on Undirected Graphs)
Let G be an undirected graph with n vertices and m edges.
Then:

1. DFS(G) runs in **linear time**, i.e., $O(n+m)$.
2. DFS computes **connected components** of G.
3. DFS determines which of these components is cyclic: a component is **cyclic** iff it has a **backedge**.

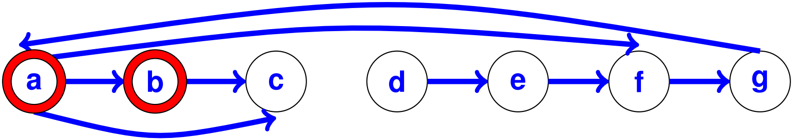
black	brown	cyan	fuchsia
tree edge	back edge	cross edge	forward edge

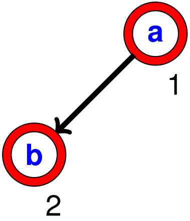


black	brown	cyan	fuchsia
tree edge	back edge	cross edge	forward edge

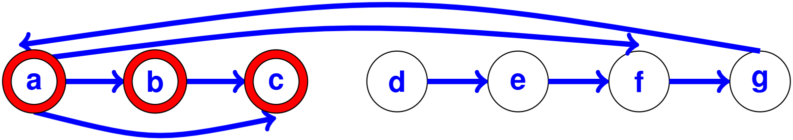


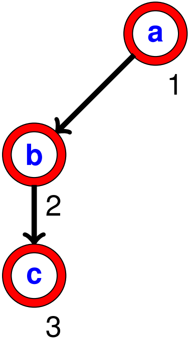
black tree edge | brown back edge | cyan cross edge | fuchsia forward edge





black	brown	cyan	fuchsia
tree edge	back edge	cross edge	forward edge



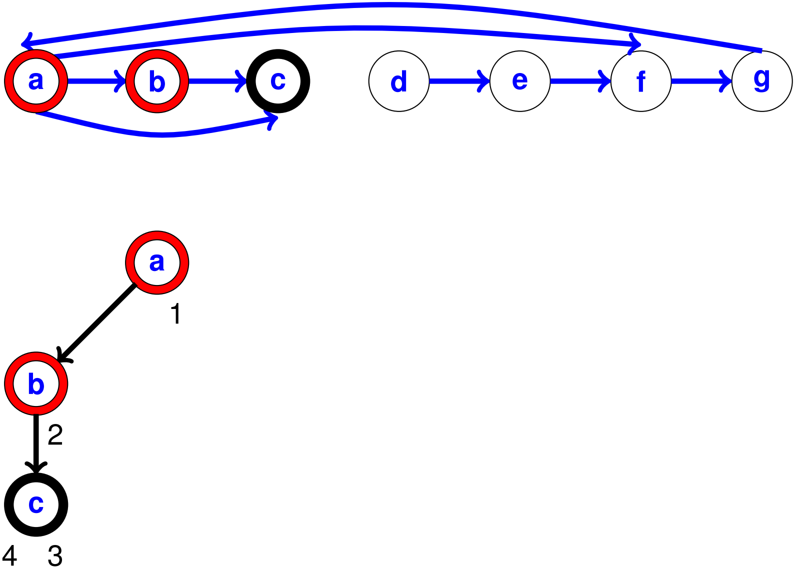


black
tree edge

brown
back edge

cyan
cross edge

fuchsia
forward edge

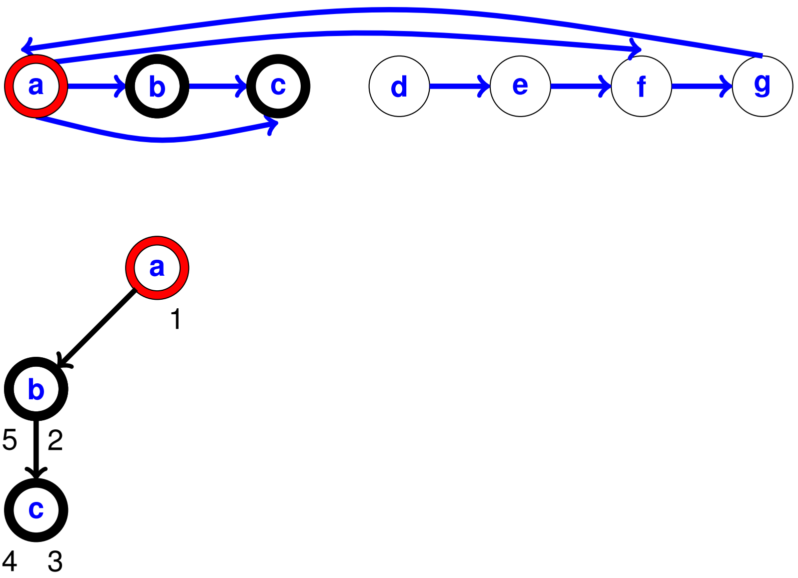


black
tree edge

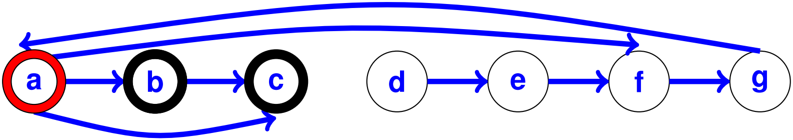
brown
back edge

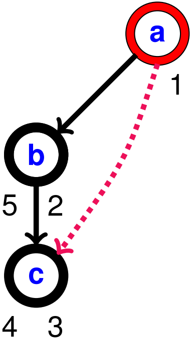
cyan
cross edge

fuchsia
forward edge



black	brown	cyan	fuchsia
tree edge	back edge	cross edge	forward edge



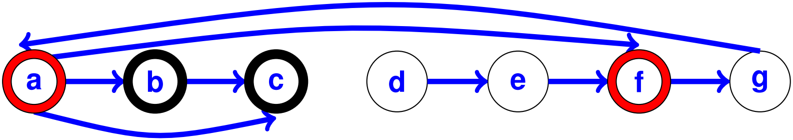


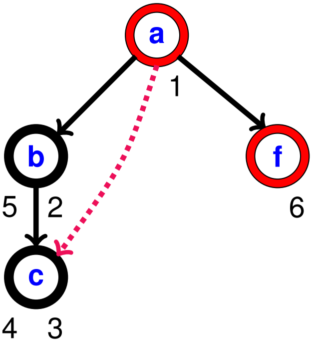
black
tree edge

brown
back edge

cyan
cross edge

fuchsia
forward edge



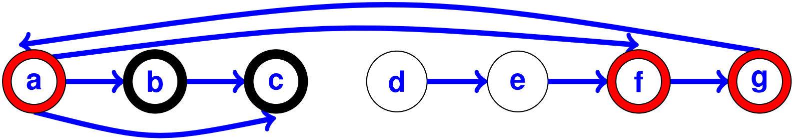


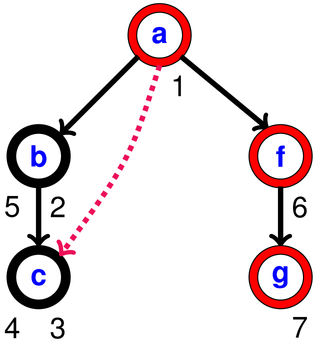
black tree edge

brown back edge

cyan cross edge

fuchsia forward edge



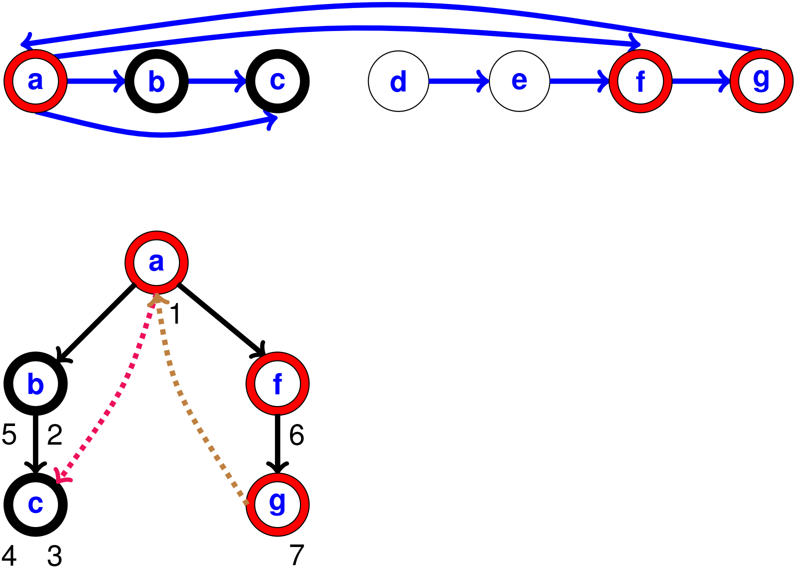


black
tree edge

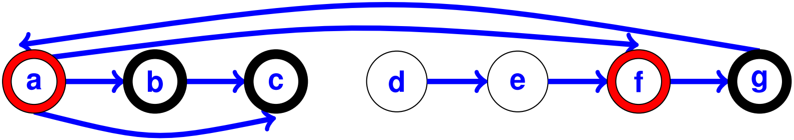
brown
back edge

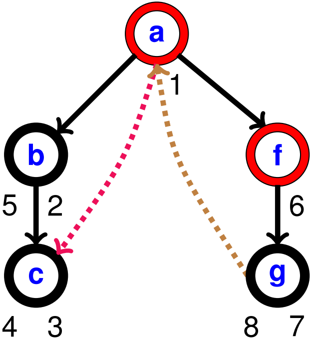
cyan
cross edge

fuchsia
forward edge

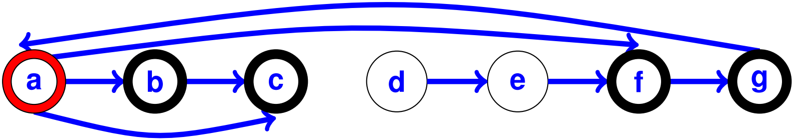


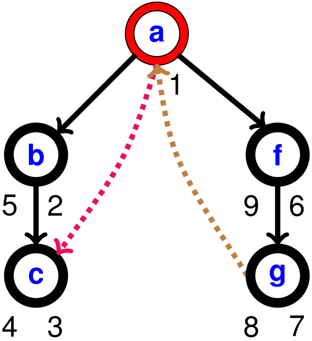
black	brown	cyan	fuchsia
tree edge	back edge	cross edge	forward edge



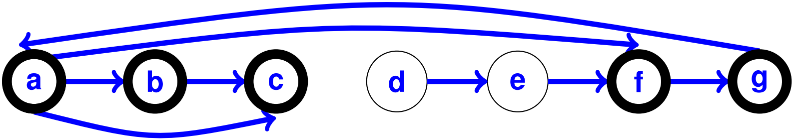


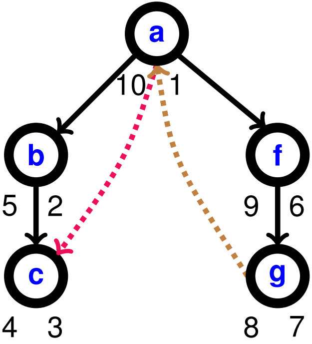
black tree edge | brown back edge | cyan cross edge | fuchsia forward edge

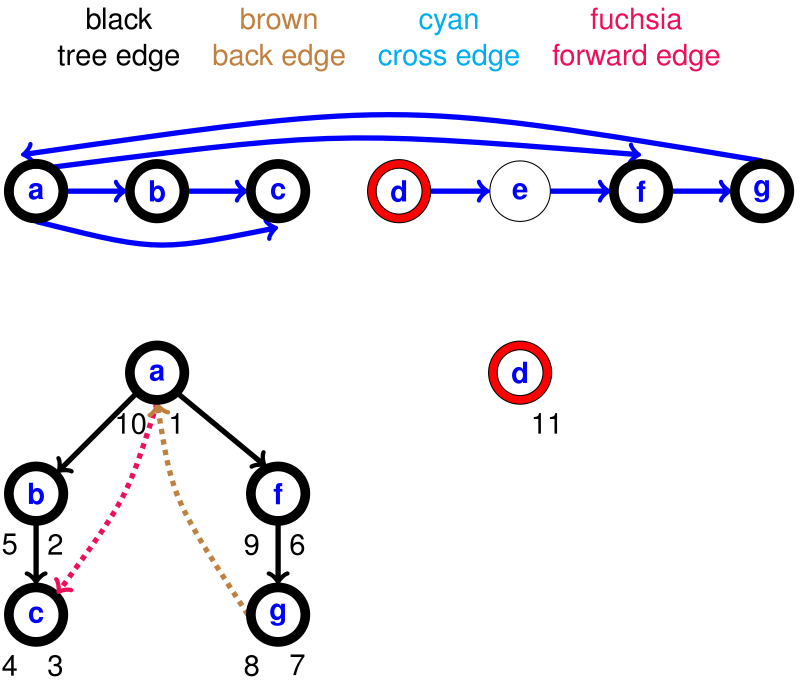




black tree edge | brown back edge | cyan cross edge | fuchsia forward edge





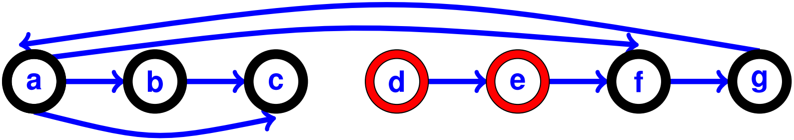

black
tree edge
brown
back edge
cyan
cross edge
fuchsia
forward edge
a
b
c
d
e
f
g
10
1
11
5
2
9
6
4
3
8
7

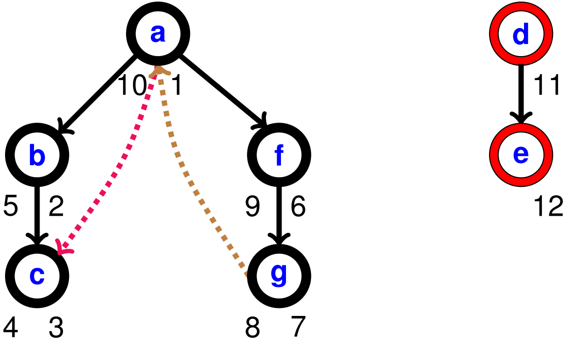
black
tree edge

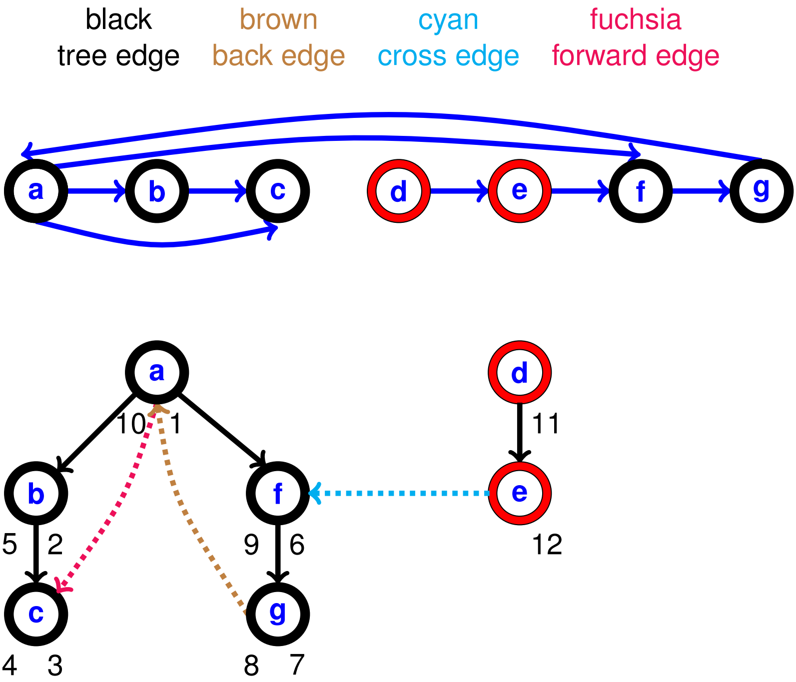
brown
back edge

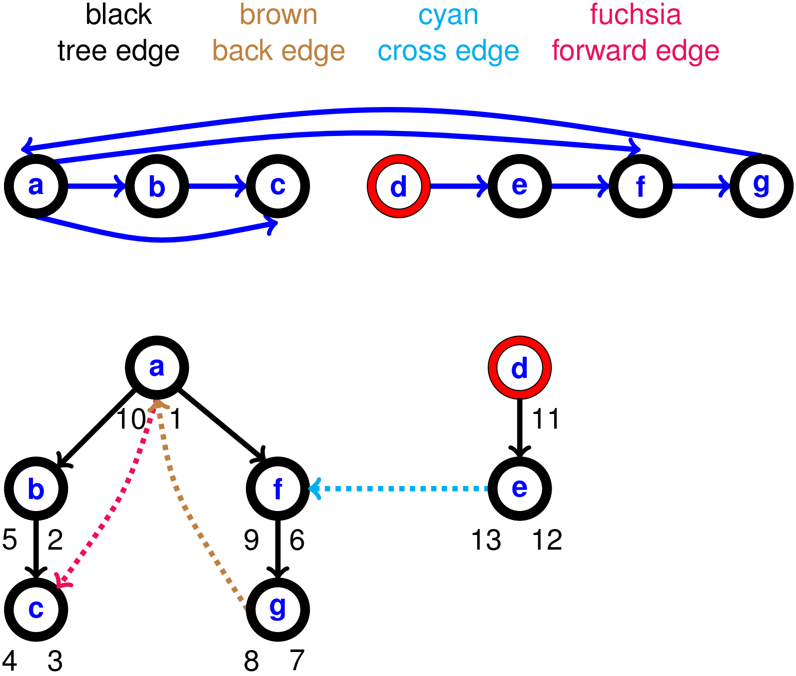
cyan
cross edge

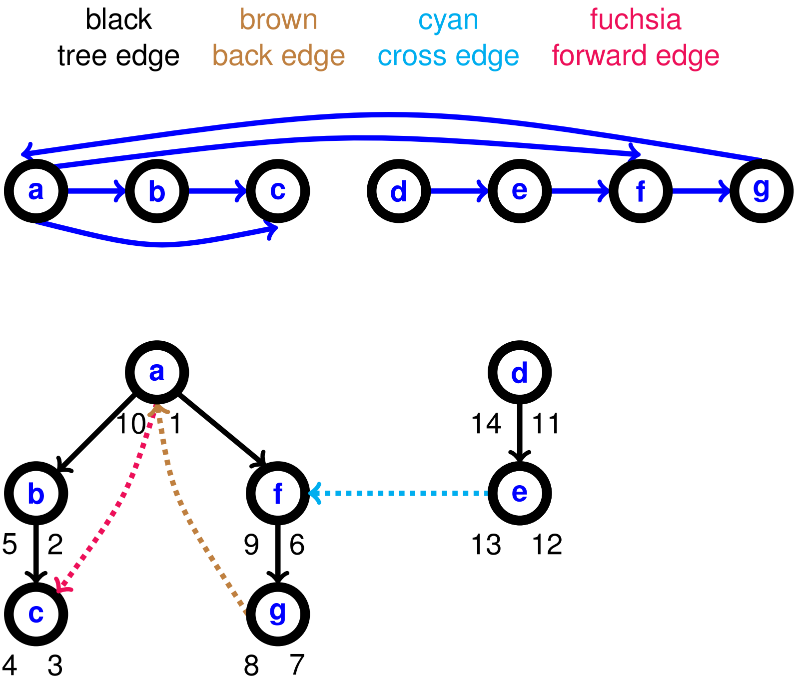
fuchsia
forward edge






black
tree edge
brown
back edge
cyan
cross edge
fuchsia
forward edge
a
b
c
d
e
f
g
a
10
1
b
5
2
c
4
3
f
9
6
g
8
7
d
11
e
12


black
tree edge
brown
back edge
cyan
cross edge
fuchsia
forward edge
a
b
c
d
e
f
g
a
10
1
b
5
2
c
4
3
f
9
6
g
8
7
d
11
e
13
12


black
tree edge
brown
back edge
cyan
cross edge
fuchsia
forward edge
a
b
c
d
e
f
g
a
10 1
b
5 2
c
4 3
f
9 6
g
8 7
d
14 11
e
13 12

Depth First Search

```
DFSmain(G)
 for each u in V :
   color[u] = white
   parent[u] = NULL
 time=0
 for each u in V :
   if (color[u] == white):
     DFSVisit(u)
```

```
DFSVisit(u)
 color[u] = red                  // in process
 d[u] = ++time                   // discover
 for each v in Adj(u) :
   if (color[v] == white) :      // unseen
     parent[v] = u               // tree edge
     DFSVisit(v)
 color[u] = black                // done
 f[u] = ++time                   // finish
```

Thm. (Properties of DFS on directed Graphs)

Let G be an directed graph with n vertices and m edges. Then:

Thm. (Properties of DFS on directed Graphs)

Let G be an directed graph with n vertices and m edges. Then:

1. DFS(G) runs in **linear time**, i.e., $O(n + m)$.

Thm. (Properties of DFS on directed Graphs)

Let G be an directed graph with n vertices and m edges. Then:

1. DFS(G) runs in **linear time**, i.e., $O(n + m)$.
2. When a DFS tree T is completed all the vertices reachable from any vertex in T have been visited either in T, or in previous trees.

Thm. (Properties of DFS on directed Graphs)

Let G be an directed graph with n vertices and m edges. Then:

1. DFS(G) runs in **linear time**, i.e., $O(n + m)$.

2. When a DFS tree T is completed all the vertices reachable from any vertex in T have been visited either in T, or in previous trees.

3. G is **cyclic** iff DFS(G) finds **backedge**.

Thm. (Properties of DFS on directed Graphs)

Let G be an directed graph with n vertices and m edges. Then:

1. DFS(G) runs in **linear time**, i.e., $O(n + m)$.
2. When a DFS tree T is completed all the vertices reachable from any vertex in T have been visited either in T, or in previous trees.
3. G is **cyclic** iff DFS(G) finds **backedge**.
4. If G is acyclic, then the **reverse finish times** provide a **topological ordering** of G.

Thm. (Properties of DFS on directed Graphs)

Let G be an directed graph with n vertices and m edges. Then:

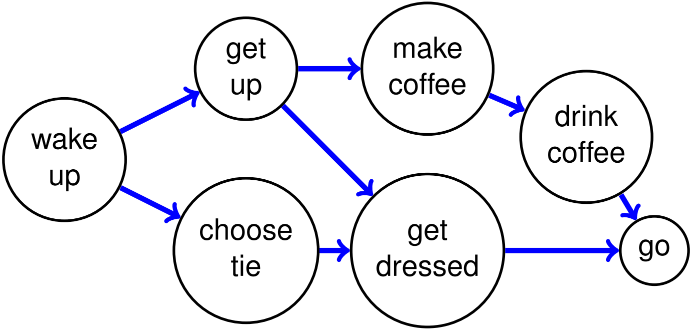
1. DFS(G) runs in **linear time**, i.e., $O(n + m)$.
2. When a DFS tree T is completed all the vertices reachable from any vertex in T have been visited either in T, or in previous trees.
3. G is **cyclic** iff DFS(G) finds **backedge**.
4. If G is acyclic, then the **reverse finish times** provide a **topological ordering** of G.
5. A second run of DFS on G^R – with roots chosen in **reverse finish time** of the first search – computes the **strongly connected components** of G.

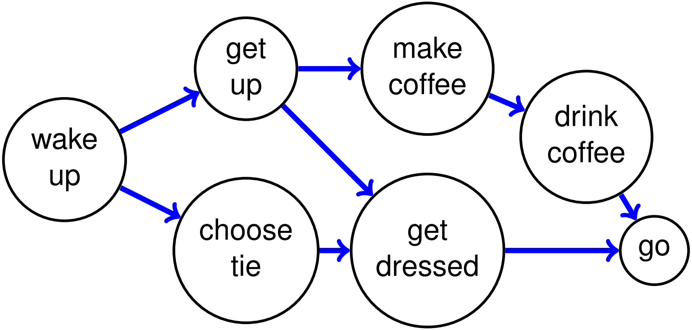
Thm. **(Properties of DFS on directed Graphs)**

Let G be an directed graph with n vertices and m edges. Then:

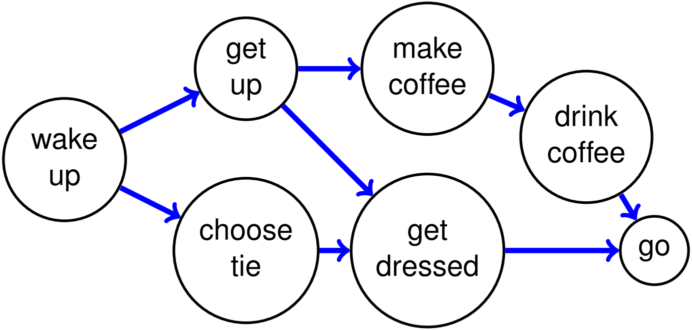
1. DFS(G) runs in **linear time**, i.e., $O(n+m)$.
2. When a DFS tree T is completed all the vertices reachable from any vertex in T have been visited either in T, or in previous trees.
3. G is **cyclic** iff DFS(G) finds **backedge**.
4. If G is acyclic, then the **reverse finish times** provide a **topological ordering** of G.
5. A second run of DFS on G^R – with roots chosen in **reverse finish time** of the first search – computes the **strongly connected components** of G.

Proof: 1. 2., and 3. are similar as for undirected graphs. ✓


wake up
get up
choose tie
make coffee
get dressed
drink coffee
go

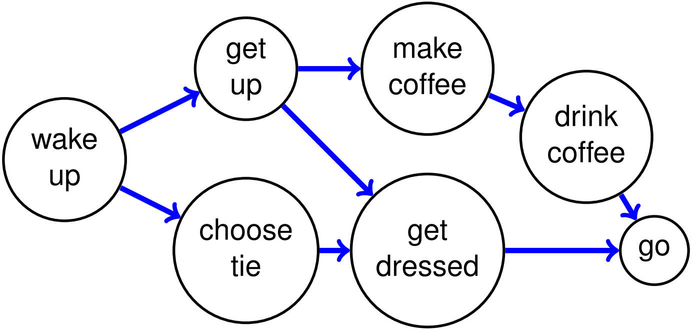


Def. A **topological ordering** of a DAG G, is an ordering of $V^G = \{v_1, \ldots, v_n\}$ such that for all $(v_i, v_j) \in E^G$, $i < j$.



Def. A **topological ordering** of a DAG G, is an ordering of $V^G = \{v_1, \ldots, v_n\}$ such that for all $(v_i, v_j) \in E^G$, $i < j$.

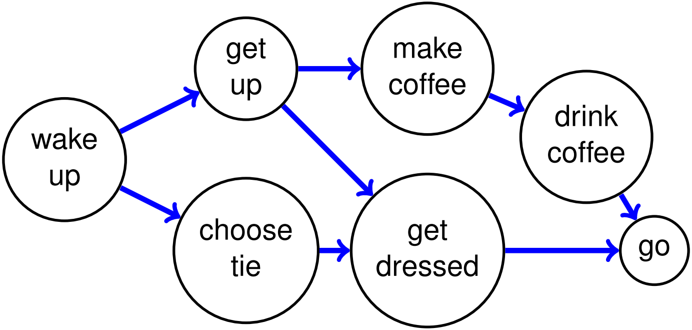
A **topological ordering** of a pert chart, is a valid ordering for doing the tasks such that no task is done before any of its prerequisites.



Def. A **topological ordering** of a DAG G, is an ordering of $V^G = \{v_1, \ldots, v_n\}$ such that for all $(v_i, v_j) \in E^G$, $i < j$.

A **topological ordering** of a pert chart, is a valid ordering for doing the tasks such that no task is done before any of its prerequisites.

If G has a cycle, then G has not topological ordering.

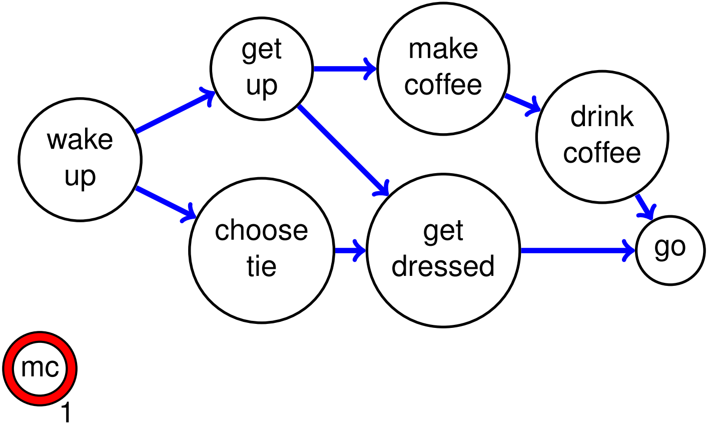


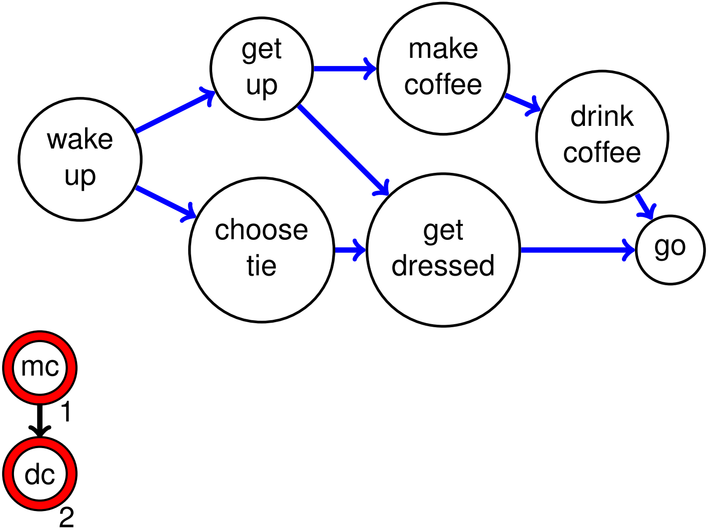
Def. A **topological ordering** of a DAG G, is an ordering of $V^G = \{v_1, \dots, v_n\}$ such that for all $(v_i, v_j) \in E^G$, $i < j$.

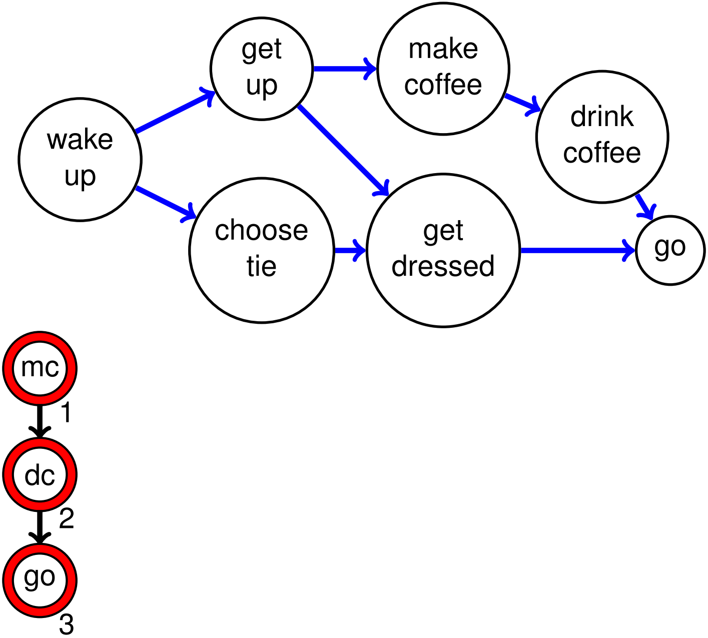
A **topological ordering** of a pert chart, is a valid ordering for doing the tasks such that no task is done before any of its prerequisites.

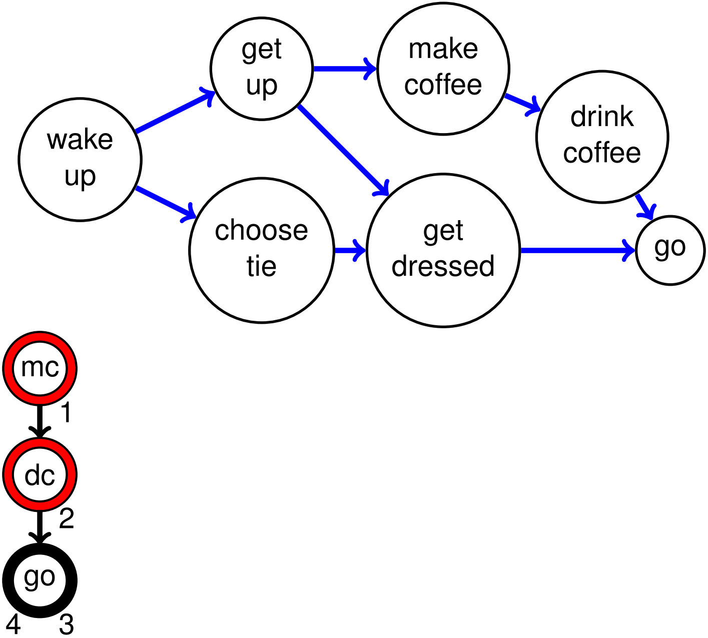
If G has a cycle, then G has not topological ordering.

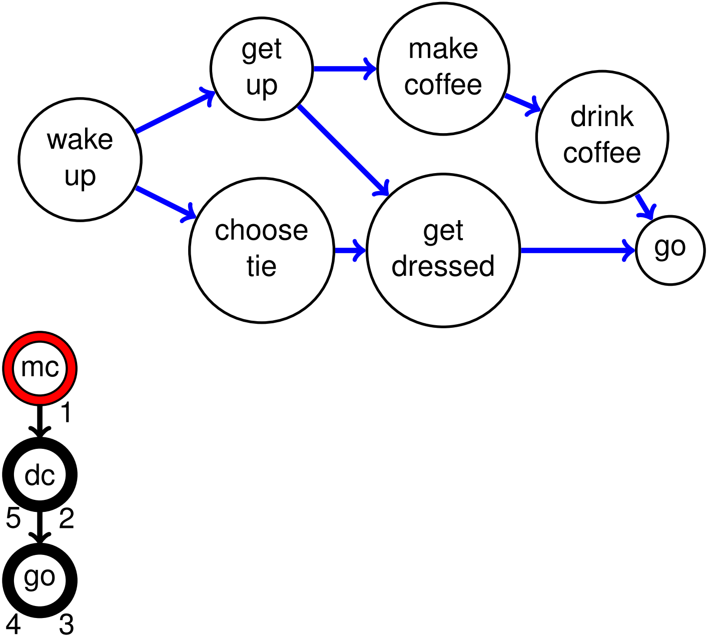
4. If G is acyclic, then the **reverse finish times** provide a **topological ordering** of G.

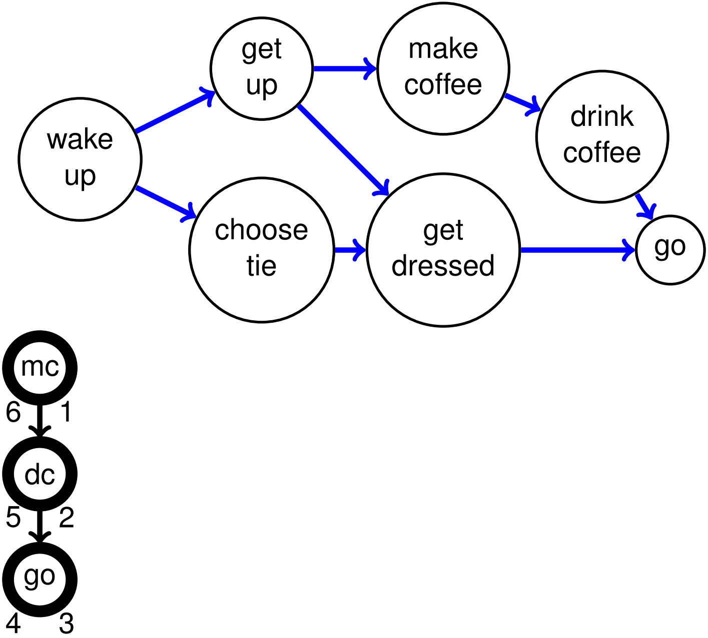

wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
1

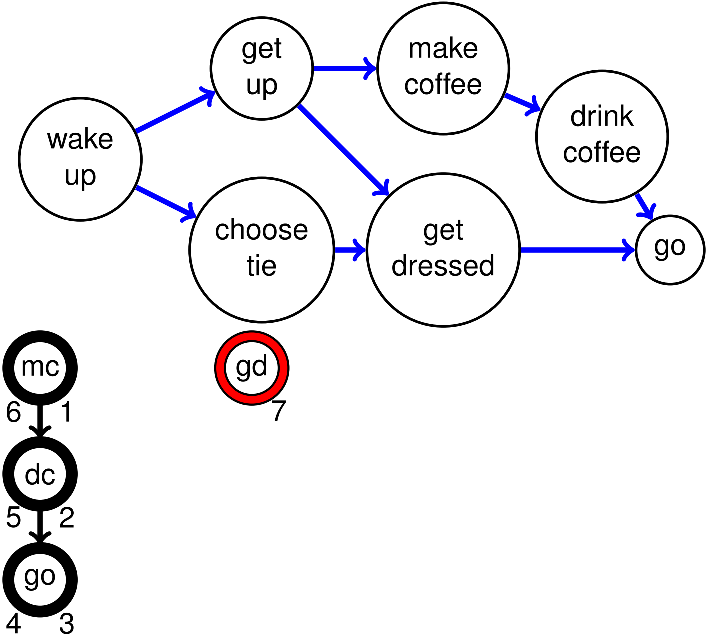

wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
1
dc
2

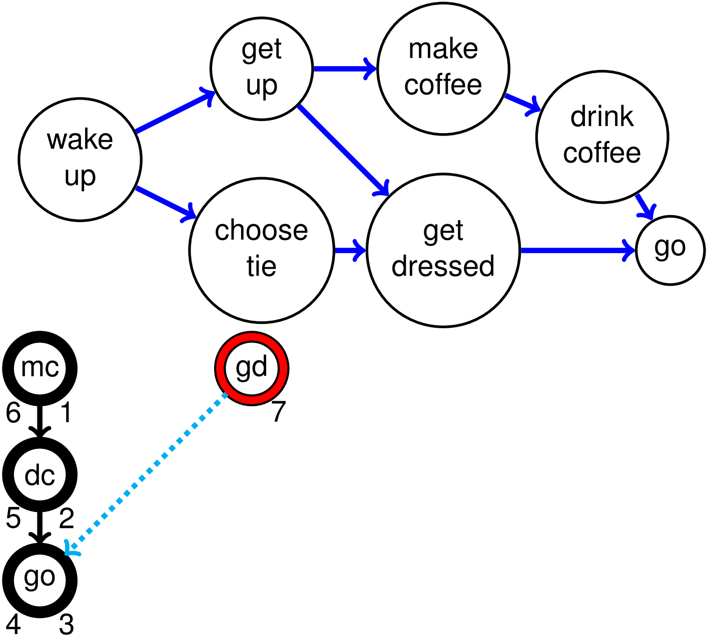

wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
1
dc
2
go
3


wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
1
dc
2
go
4
3

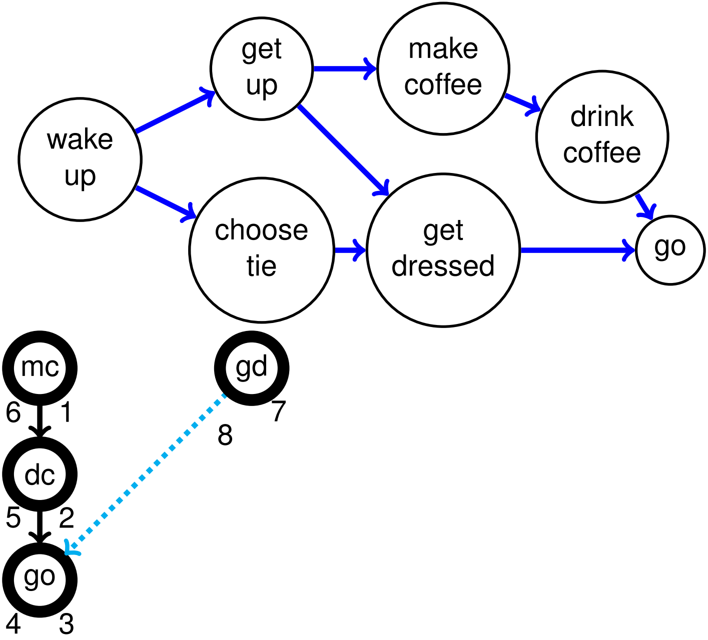

wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
1
dc
5
2
go
4
3


wake up
get up
choose tie
make coffee
get dressed
drink coffee
go
mc
dc
go
6
1
5
2
4
3

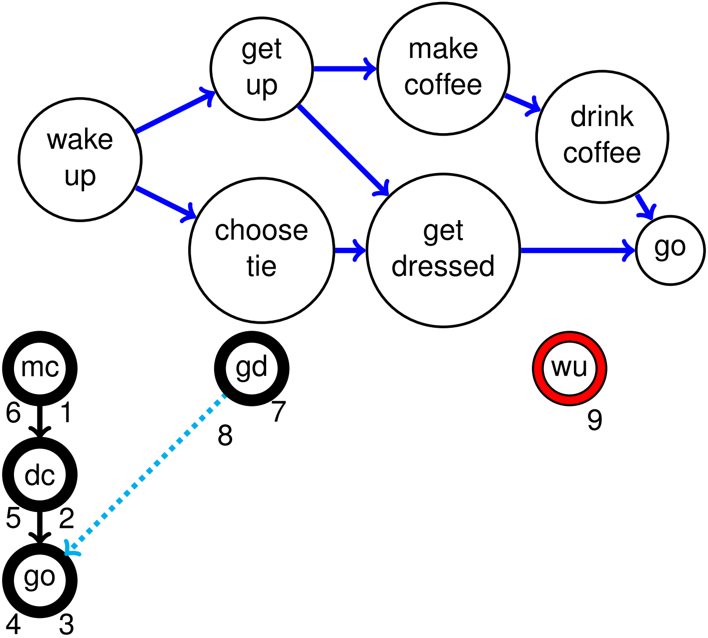

wake up
get up
choose tie
make coffee
get dressed
drink coffee
go
mc
6
1
dc
5
2
go
4
3
gd
7



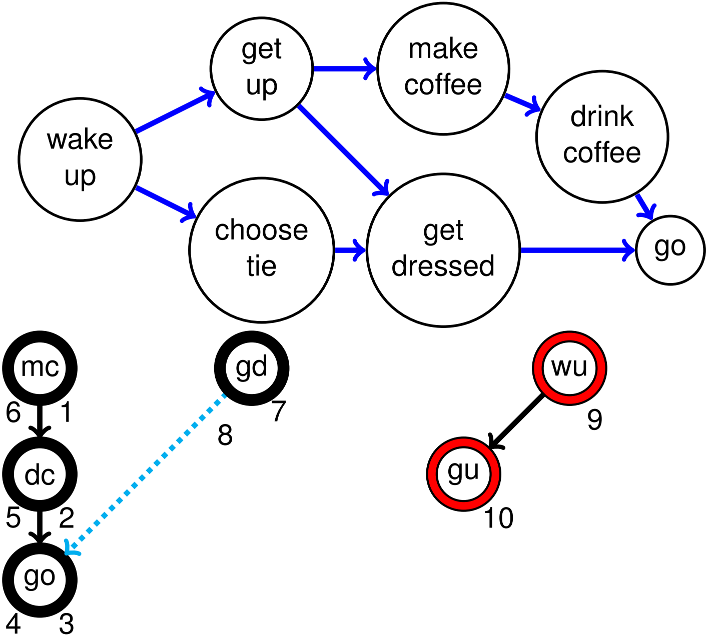
wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
dc
go
gd
6
1
5
2
4
3
7



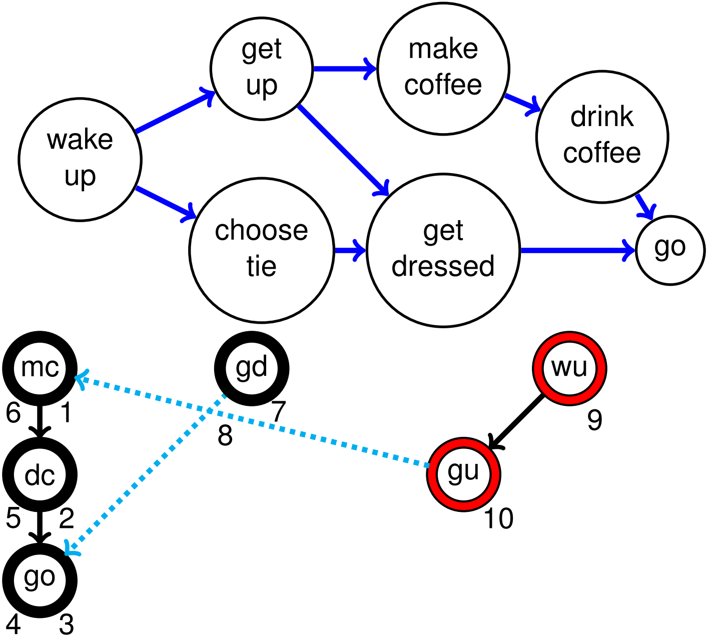
wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
dc
go
gd
6
1
5
2
4
3
7
8

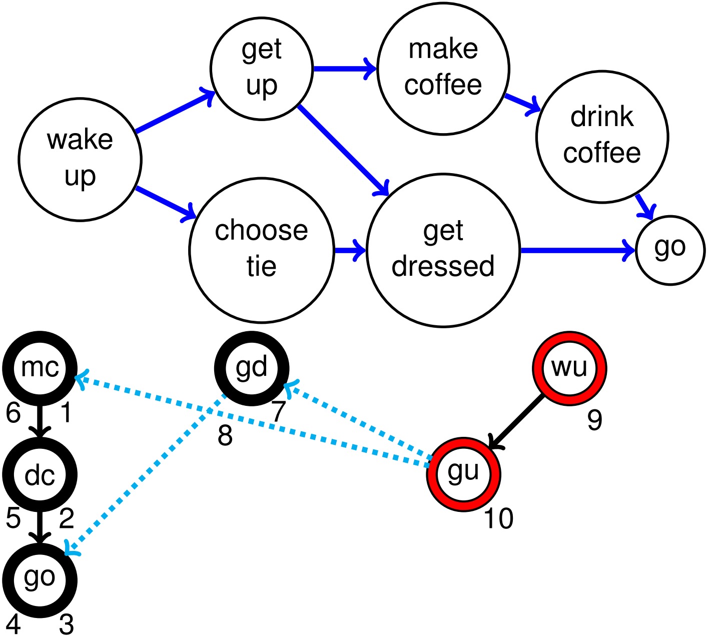


wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
dc
go
gd
wu
6
1
5
2
4
3
7
8
9

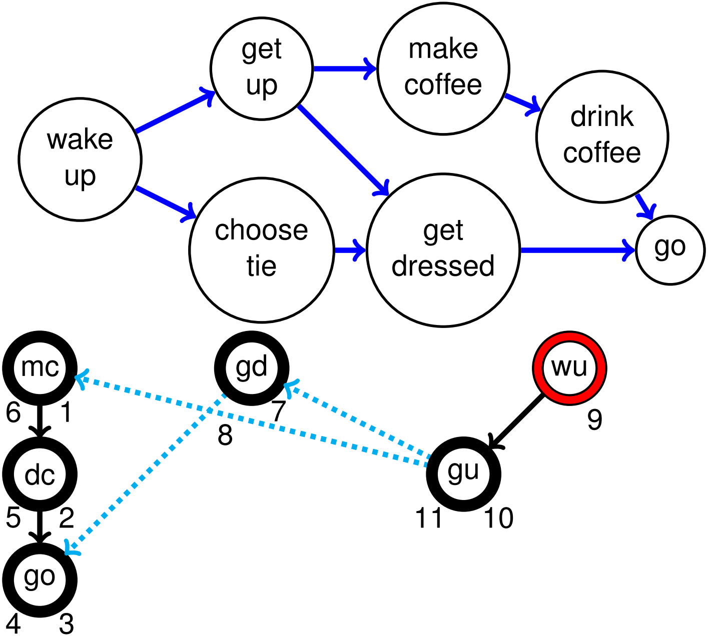


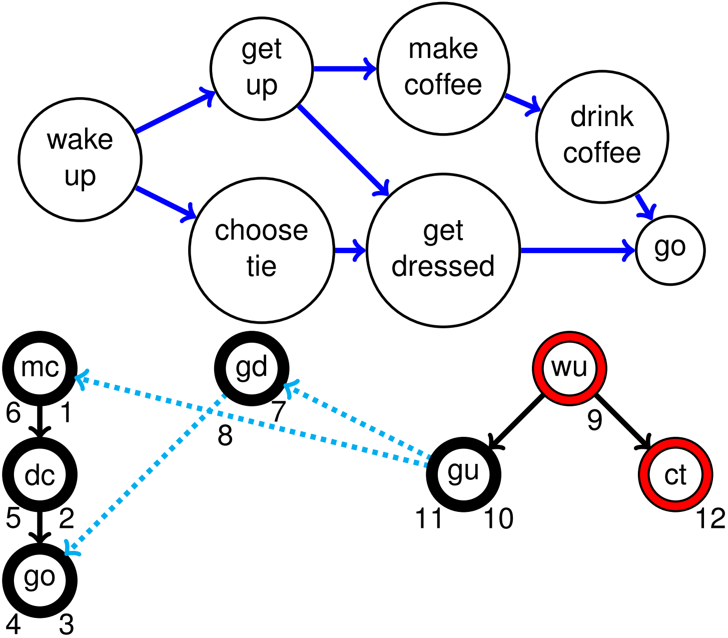
wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
gd
wu
dc
gu
go
6
1
7
8
9
5
2
10
4
3


wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
dc
go
gd
wu
gu
6
1
5
2
4
3
7
8
9
10

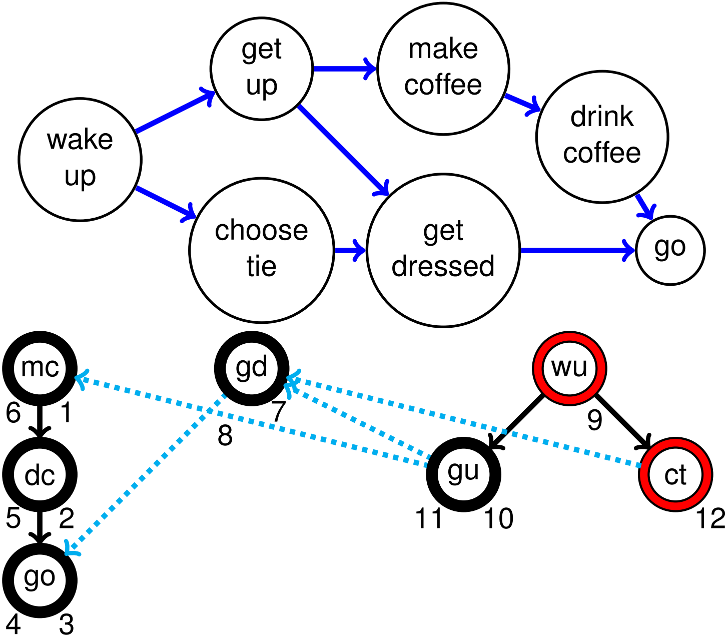


get up
make coffee
wake up
drink coffee
choose tie
get dressed
go
mc
gd
wu
6
1
7
8
9
dc
gu
5
2
10
go
4
3

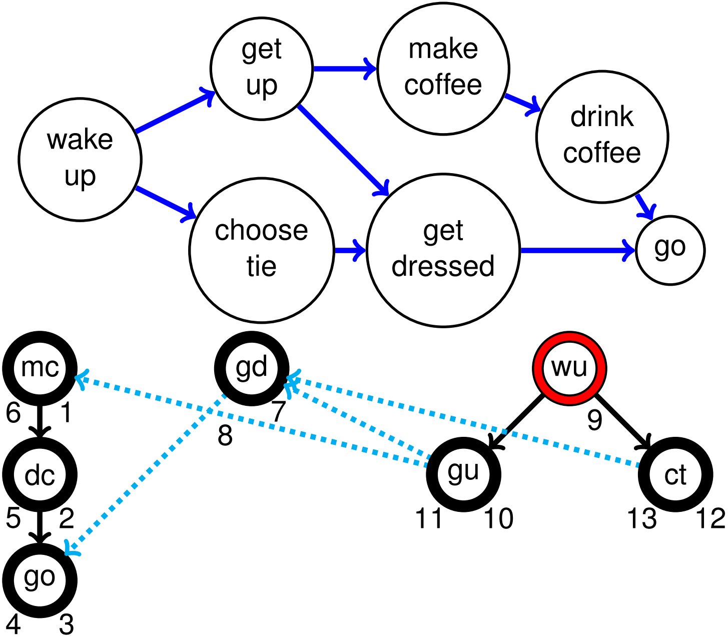

wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
gd
wu
dc
gu
go
6
1
7
8
9
5
2
11
10
4
3



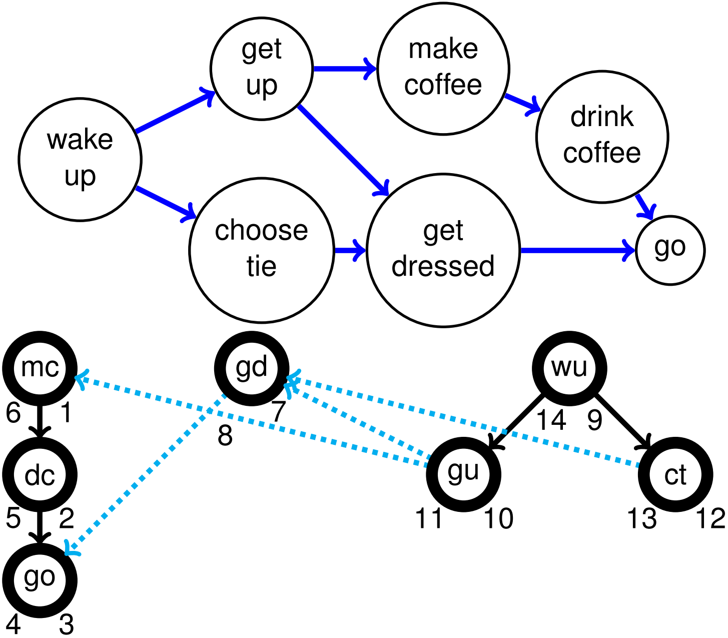
wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
dc
go
gd
wu
gu
ct
6
1
5
2
4
3
7
8
9
11
10
12



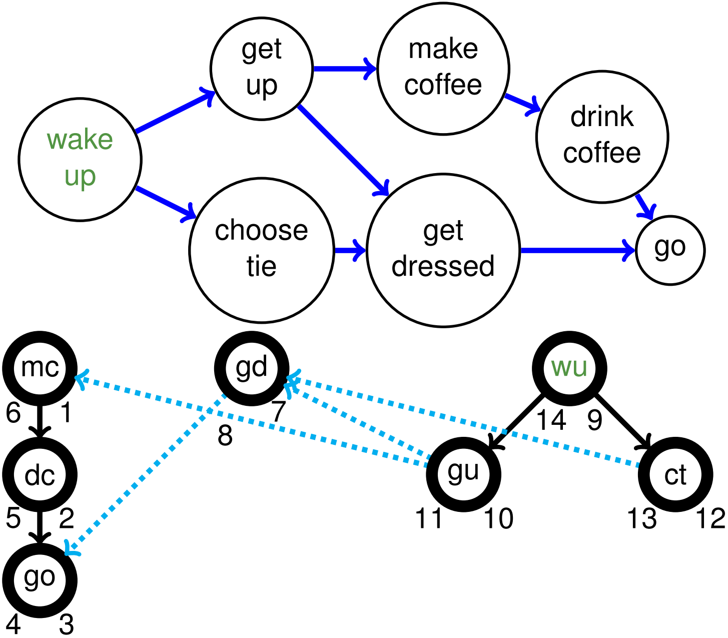
wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
gd
wu
dc
gu
ct
go
6
1
7
8
9
5
2
11
10
12
4
3



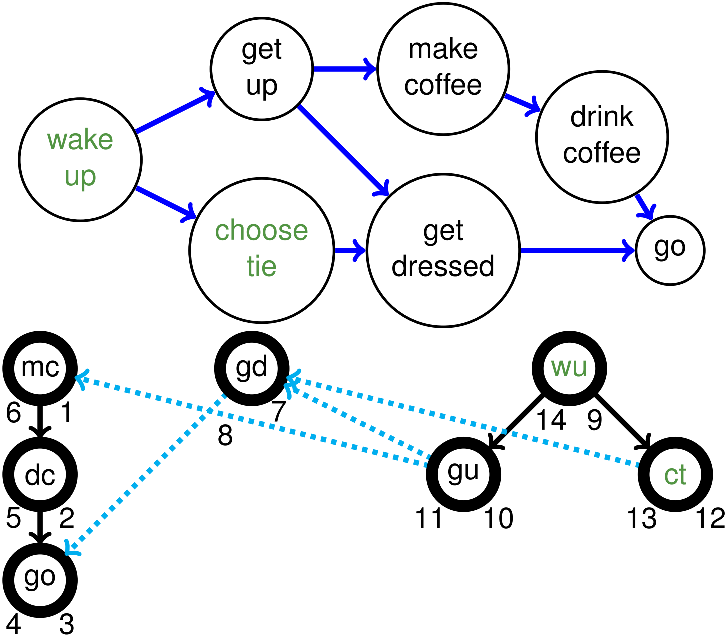
wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
gd
wu
dc
gu
ct
go
6
1
7
8
9
5
2
11
10
13
12
4
3



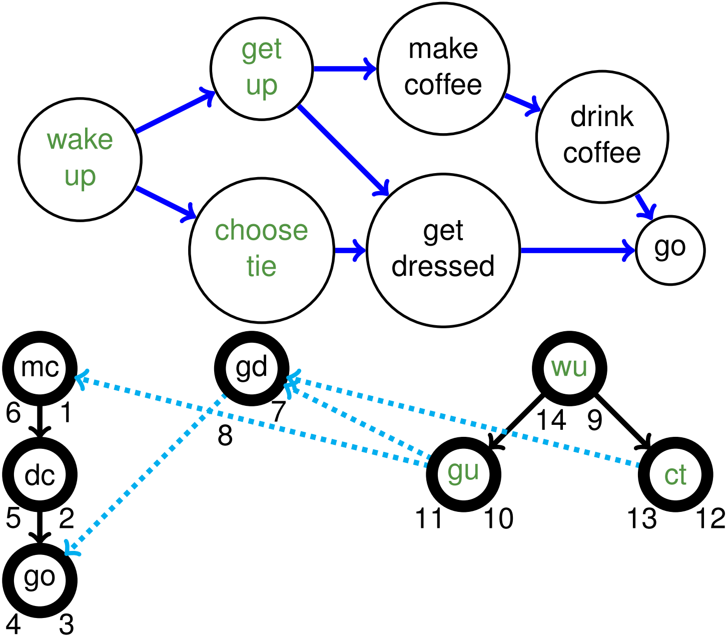
wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
gd
wu
dc
gu
ct
go
6
1
7
8
14
9
5
2
11
10
13
12
4
3

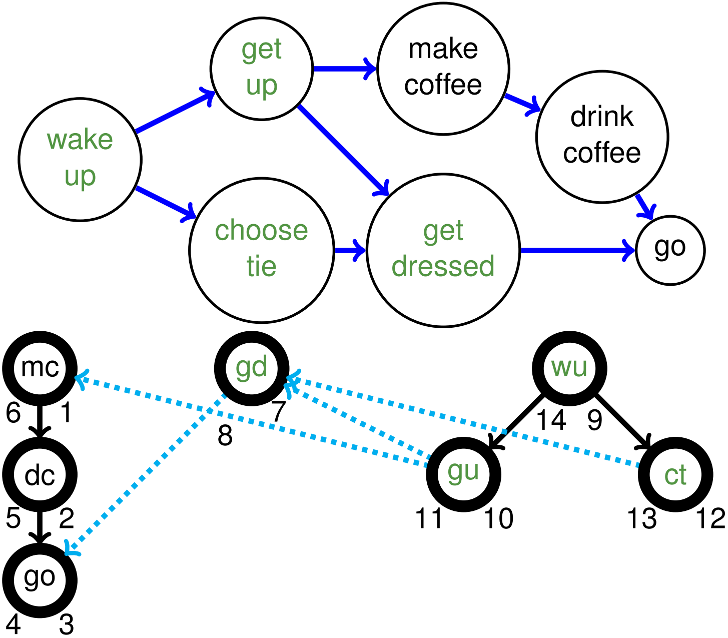


wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
gd
wu
dc
gu
ct
go
6
1
7
8
14
9
5
2
11
10
13
12
4
3



wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
gd
wu
dc
gu
ct
go
6
1
7
8
14
9
5
2
11
10
13
12
4
3


wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
gd
wu
dc
gu
ct
go
6
1
7
8
14
9
5
2
11
10
13
12
4
3



wake up
get up
make coffee
drink coffee
choose tie
get dressed
go
mc
gd
wu
dc
gu
ct
go
6
1
7
8
14
9
5
2
11
10
13
12
4
3