Depth-First Search

Letícia Rodrigues Bueno

Federal University of ABC (UFABC)

Problem 1: Topological Sorting

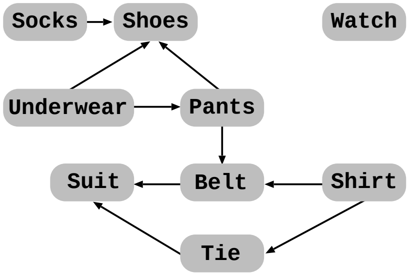
- **acyclic directed graphs:** used to indicate precedence of events;

Problem 1: Topological Sorting

- **acyclic directed graphs:** used to indicate precedence of events;
- **topological sorting:** linear sorting such that directed edges follow from left to right. Example:

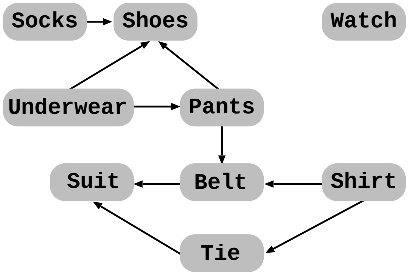
Problem 1: Topological Sorting

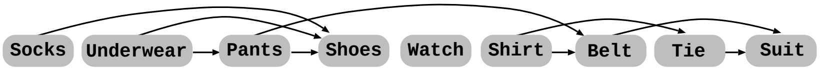
- **acyclic directed graphs:** used to indicate precedence of events;
- **topological sorting:** linear sorting such that directed edges follow from left to right. Example:



Problem 1: Topological Sorting

- **acyclic directed graphs:** used to indicate precedence of events;
- **topological sorting:** linear sorting such that directed edges follow from left to right. Example:





Depth-first search (DFS)

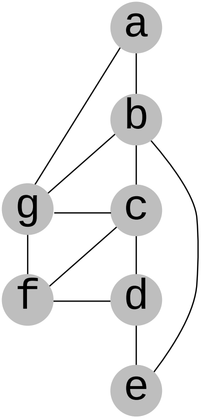
How to find a topological sorting in an acyclic directed graph?

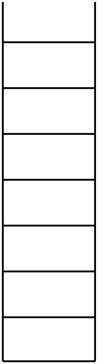
Depth-first search (DFS)

How to find a topological sorting in an acyclic directed graph?

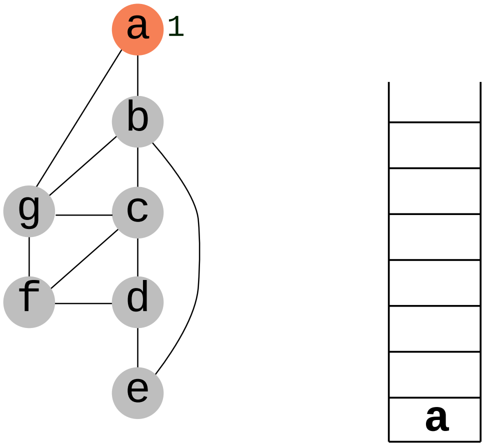
We can use a Depth-first search!

Depth-first search (DFS)

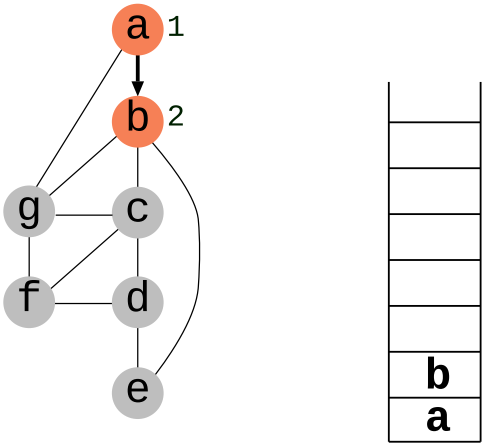




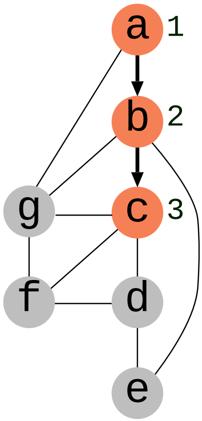
Depth-first search (DFS)

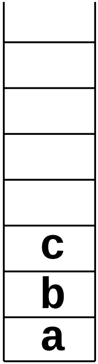


Depth-first search (DFS)

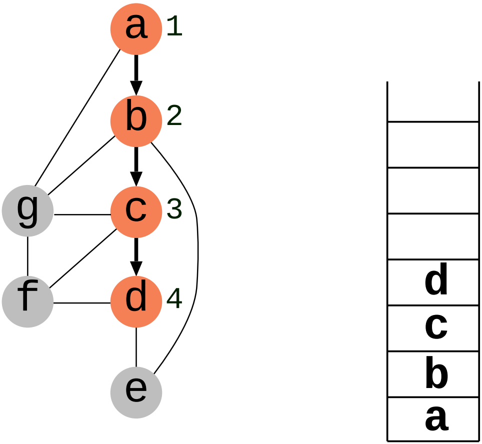


Depth-first search (DFS)

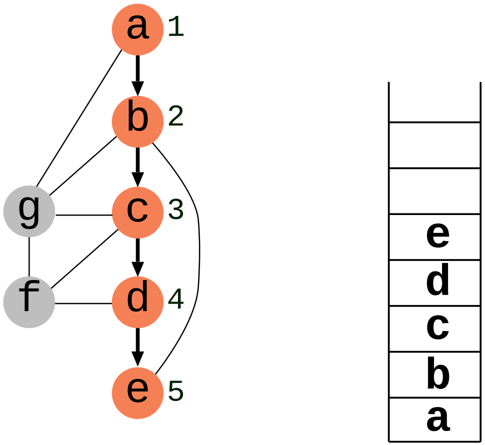




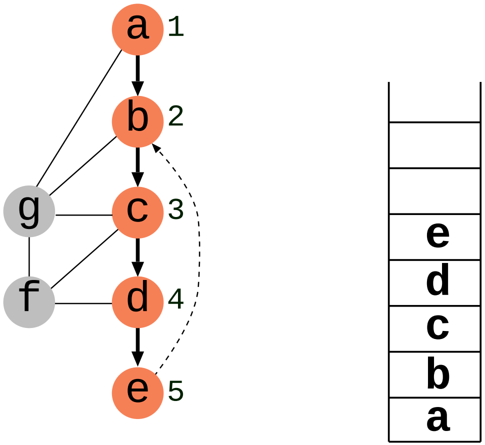
Depth-first search (DFS)



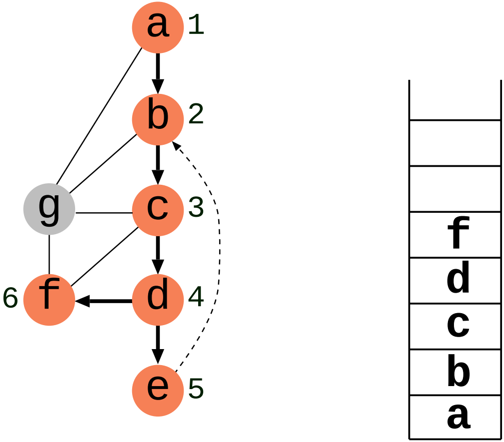
Depth-first search (DFS)



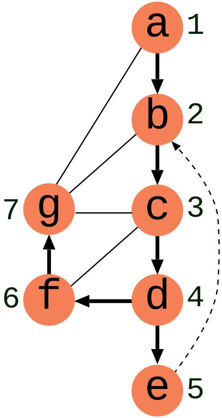
Depth-first search (DFS)

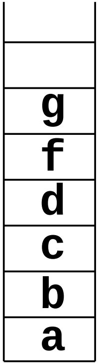


Depth-first search (DFS)

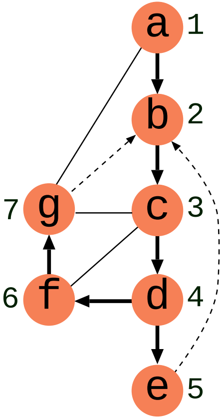


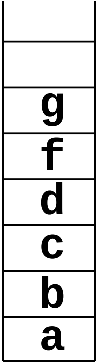
Depth-first search (DFS)



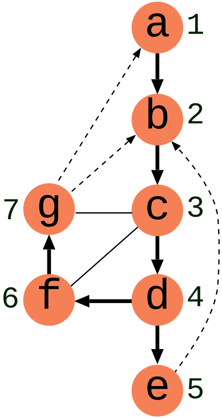


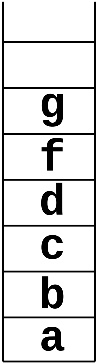
Depth-first search (DFS)



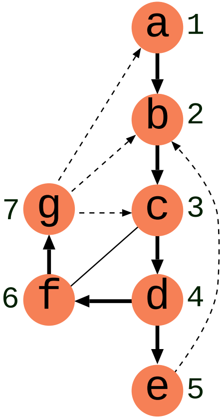


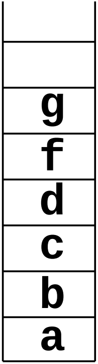
Depth-first search (DFS)



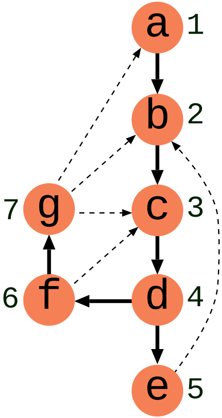


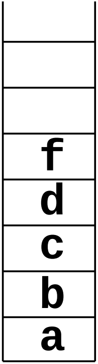
Depth-first search (DFS)



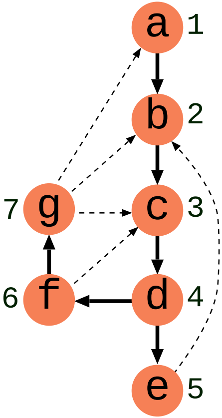


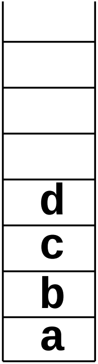
Depth-first search (DFS)



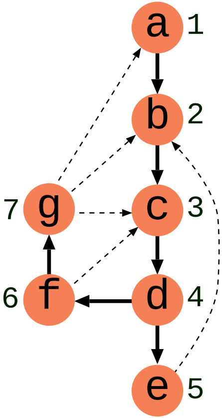


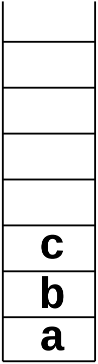
Depth-first search (DFS)



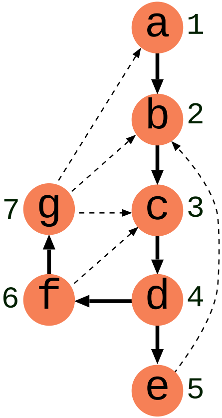


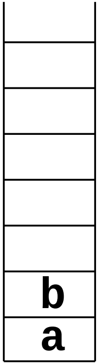
Depth-first search (DFS)



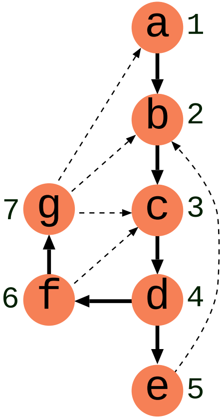


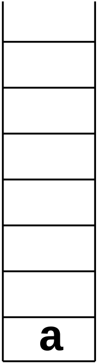
Depth-first search (DFS)



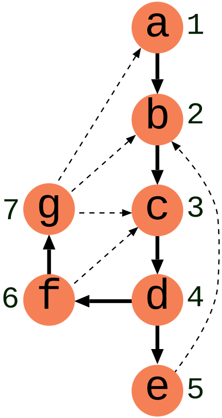


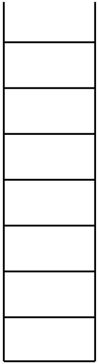
Depth-first search (DFS)





Depth-first search (DFS)





DFS Algorithm

```
dfs(G, u, cont):
    u.visited = True
    u.d = cont
    for v in adj(u) do
        if not v.visited then
            v.p = u
            dfs(G, v, cont+1)


for u in V(G) do
     u.visited = False
     u.d = ∞
     u.p = None
cont = 1
dfs(G, u, cont)
```

DFS Algorithm

```
dfs(G, u, cont):
    u.visited = True
    u.d = cont
    for v in adj(u) do
        if not v.visited then
            v.p = u
            dfs(G, v, cont+1)


for u in V(G) do
     u.visited = False
     u.d = ∞
     u.p = None
cont = 1
dfs(G, u, cont)
```

Analysis of Complexity:

DFS Algorithm

```
1  dfs(G, u, cont):
2      u.visited = True
3      u.d = cont
4      for v in adj(u) do
5          if not v.visited then
6              v.p = u
7              dfs(G, v, cont+1)

8  for u in V(G) do
9      u.visited = False
10     u.d = ∞
11     u.p = None
12 cont = 1
13 dfs(G, u, cont)
```

Analysis of Complexity:

- Each vertex is passed to the method dfs (line 7 and 13): $O(n)$;

DFS Algorithm

```
1   dfs(G, u, cont):
2       u.visited = True
3       u.d = cont
4       for v in adj(u) do
5           if not v.visited then
6               v.p = u
7               dfs(G, v, cont+1)

8   for u in V(G) do
9       u.visited = False
10      u.d = ∞
11      u.p = None
12  cont = 1
13  dfs(G, u, cont)
```

Analysis of Complexity:

- Each vertex is passed to the method dfs (line 7 and 13): $O(n)$;
- Adjacency list of each vertex is traversed once (line 4): $O(m)$;

DFS Algorithm

```
1  dfs(G, u, cont):
2      u.visited = True
3      u.d = cont
4      for v in adj(u) do
5          if not v.visited then
6              v.p = u
7              dfs(G, v, cont+1)

8  for u in V(G) do
9      u.visited = False
10     u.d = ∞
11     u.p = None
12 cont = 1
13 dfs(G, u, cont)
```

Analysis of Complexity:

- Each vertex is passed to the method dfs (line 7 and 13): $O(n)$;
- Adjacency list of each vertex is traversed once (line 4): $O(m)$;
- Total complexity: $O(n + m)$.

Topological Sorting

Coming back to the topological sorting problem:

Topological Sorting

Coming back to the topological sorting problem:

How to use DFS to solve it?

Coming back to the topological sorting problem:

How to use DFS to solve it?

Lemma

An acyclic directed graph always has a vertex with indegree 0.

Topological Sorting

Coming back to the topological sorting problem:

How to use DFS to solve it?

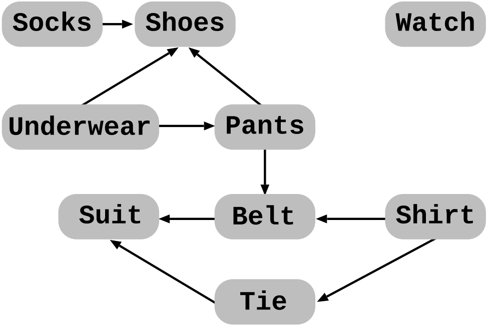
Lemma

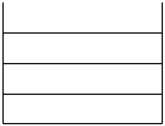
An acyclic directed graph always has a vertex with indegree 0*.*

Proof.

If every vertex has indegree greater than 0, we can go backwards through the edges without stopping. Since there is a finite number of vertices, this is only possible in a cycle, but acyclic graphs do not have cycles. □

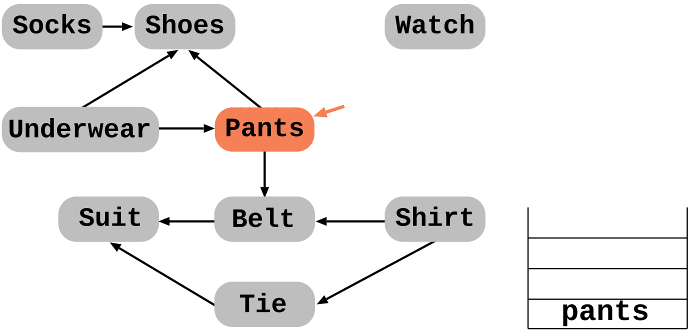
Depth-first search (DFS)





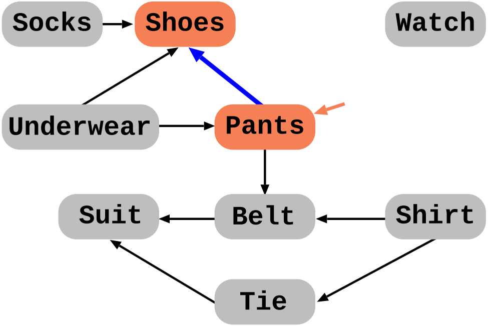
pt → λ

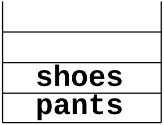
Depth-first search (DFS)



pt → λ

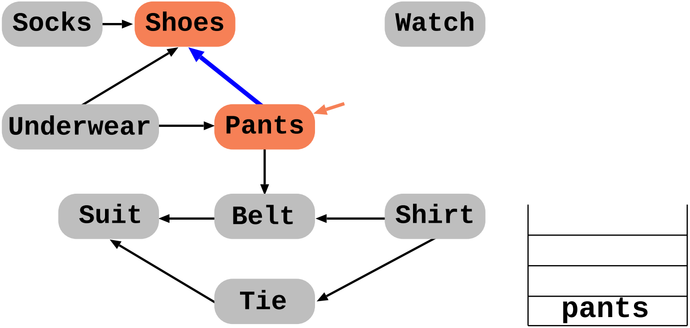
Depth-first search (DFS)





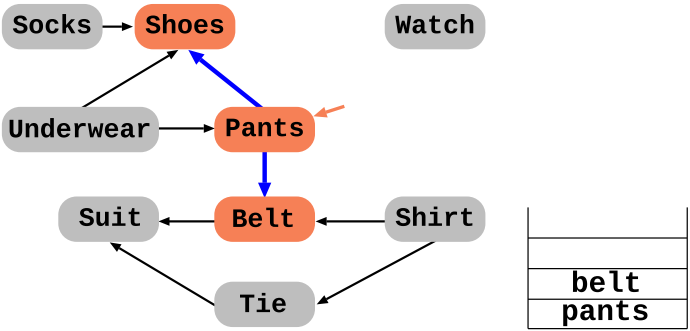
pt → λ

Depth-first search (DFS)



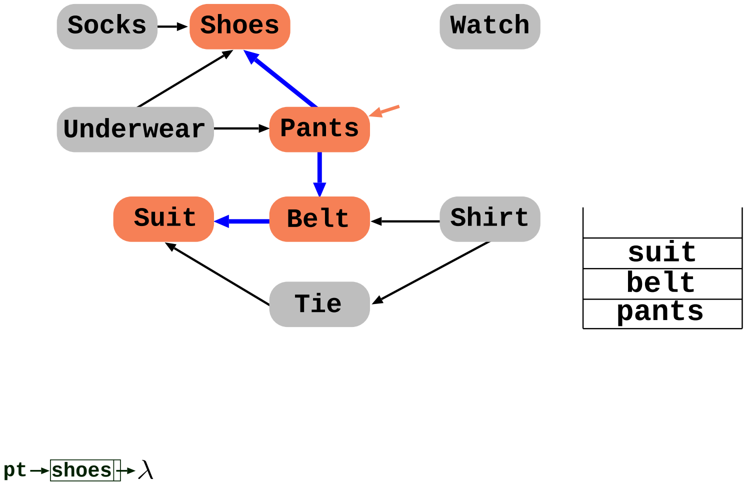
pt → shoes → λ

Depth-first search (DFS)

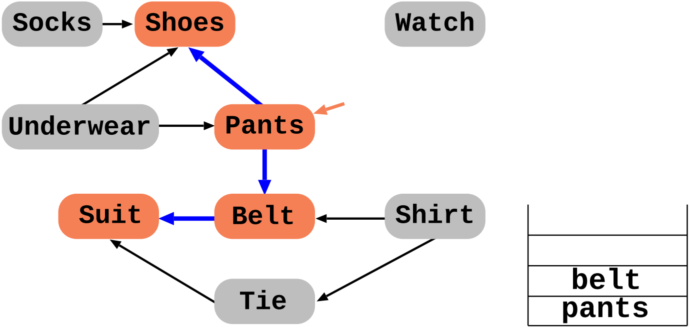


pt → shoes → λ

Depth-first search (DFS)

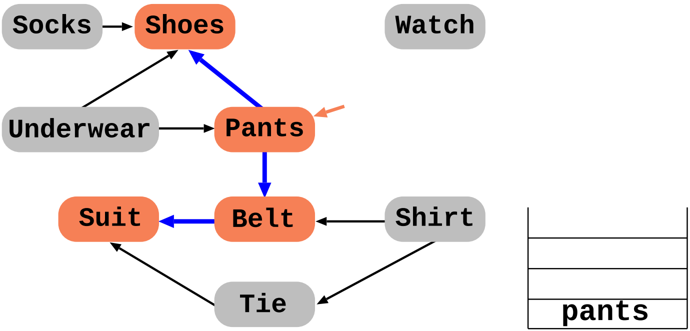


Depth-first search (DFS)



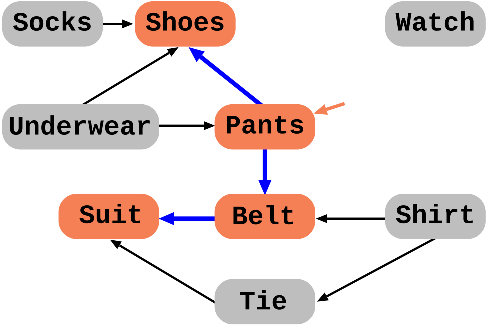
pt → suit → shoes → λ

Depth-first search (DFS)



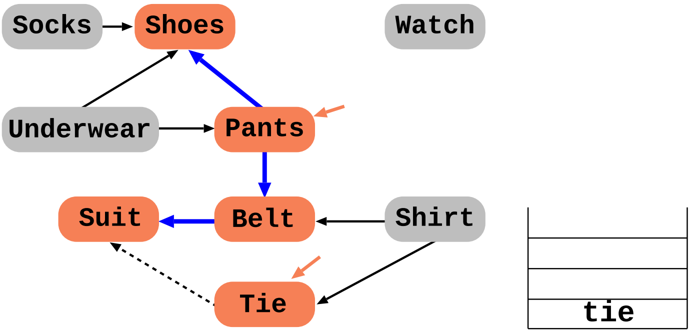
pt → belt → suit → shoes → λ

Depth-first search (DFS)



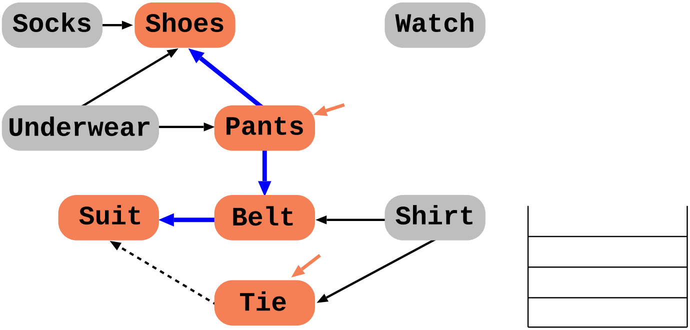
pt → **pants** → **belt** → **suit** → **shoes** → λ

Depth-first search (DFS)



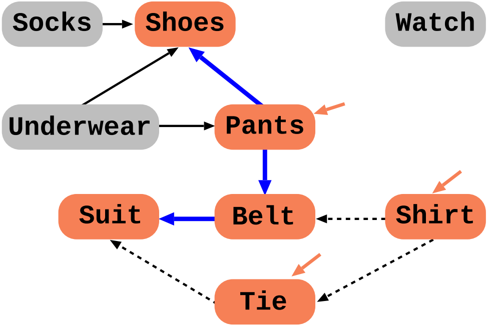
pt→pants→belt→suit→shoes→λ

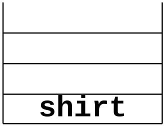
Depth-first search (DFS)



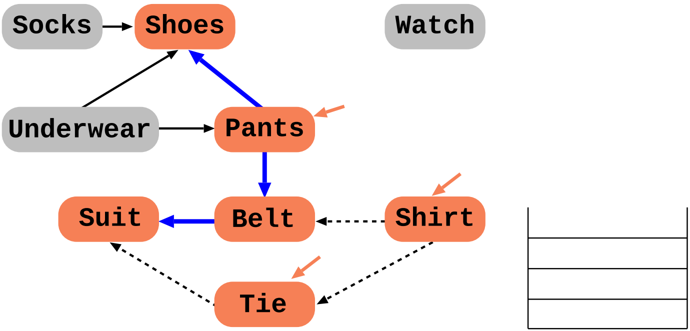
pt → tie → pants → belt → suit → shoes → λ

Depth-first search (DFS)



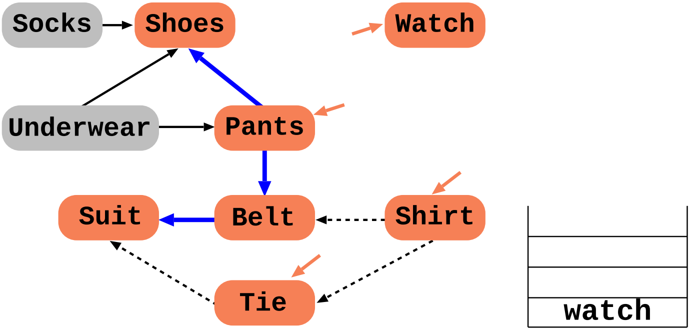


Depth-first search (DFS)



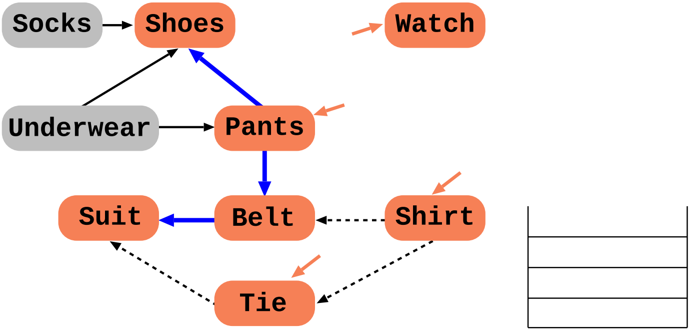
pt → shirt → tie → pants → belt → suit → shoes → λ

Depth-first search (DFS)



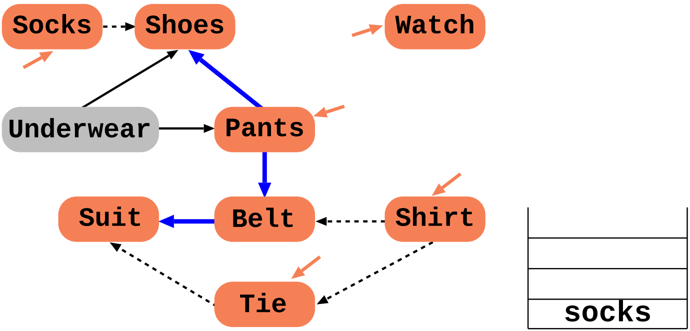
pt → shirt → tie → pants → belt → suit → shoes → λ

Depth-first search (DFS)



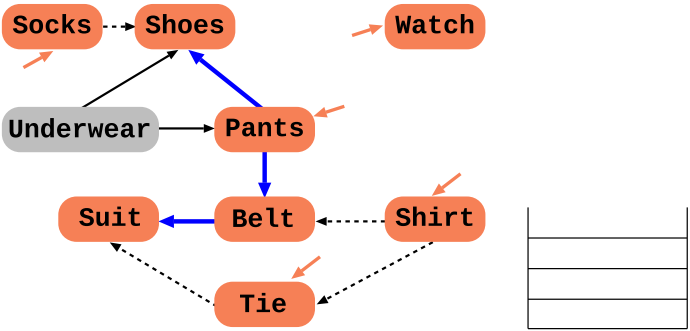
pt → watch → shirt → tie → pants → belt → suit → shoes → λ

Depth-first search (DFS)



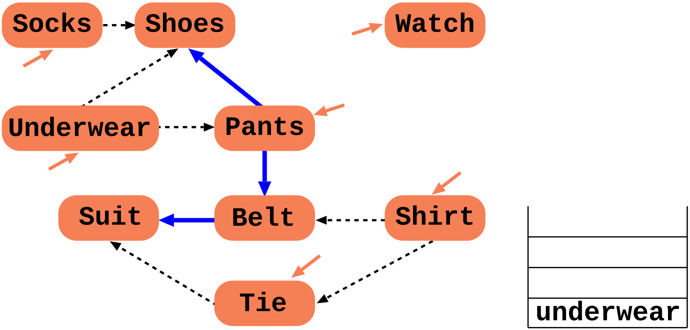
pt → watch → shirt → tie → pants → belt → suit → shoes → λ

Depth-first search (DFS)



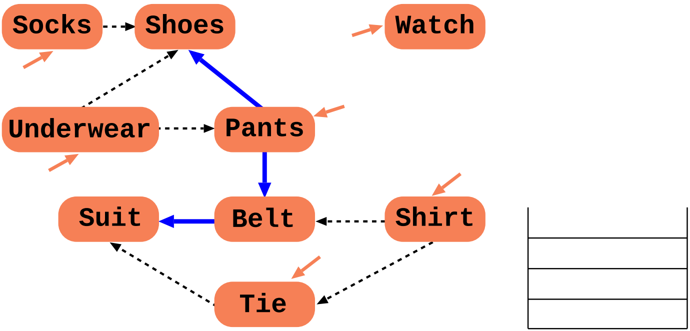
pt→ socks → watch → shirt → tie → pants → belt → suit → shoes → λ

Depth-first search (DFS)



pt → socks → watch → shirt → tie → pants → belt → suit → shoes → λ

Depth-first search (DFS)



pt→ underwear → socks → watch → shirt → tie → pants → belt → suit → shoes → λ

Topological Sorting Algorithm

```
dfs(G, u):
    u.visited = True
    for v in adj(u) do
        if not v.visited then
            dfs(G, v)
    insertFirst(L, u)


for u in V(G) do
     u.visited = False
L = List();
for u in V(G) do
    if not u.visited then
        dfs(G, u)
```

Topological Sorting Algorithm

Analysis of complexity:

```
dfs(G, u):
    u.visited = True
    for v in adj(u) do
        if not v.visited then
            dfs(G, v)
    insertFirst(L, u)


for u in V(G) do
    u.visited = False
L = List();
for u in V(G) do
    if not u.visited then
        dfs(G, u)
```

Topological Sorting Algorithm

```
1  dfs(G, u):
2      u.visited = True
3      for v in adj(u) do
4          if not v.visited then
5              dfs(G, v)
6      insertFirst(L, u)

7  for u in V(G) do
8       u.visited = False
9  L = List();
10 for u in V(G) do
11     if not u.visited then
12         dfs(G, u)
```

Analysis of complexity:

- Each vertex is passed to method dfs once (lines 5 and 12): $O(n)$;

Topological Sorting Algorithm

```
1  dfs(G, u):
2      u.visited = True
3      for v in adj(u) do
4          if not v.visited then
5              dfs(G, v)
6      insertFirst(L, u)

7  for u in V(G) do
8      u.visited = False
9  L = List();
10 for u in V(G) do
11     if not u.visited then
12         dfs(G, u)
```

Analysis of complexity:

- Each vertex is passed to method dfs once (lines 5 and 12): $O(n)$;
- Adjacency list of each vertex is traversed once (line 3): $O(m)$;

Topological Sorting Algorithm

```
1  dfs(G, u):
2      u.visited = True
3      for v in adj(u) do
4          if not v.visited then
5              dfs(G, v)
6      insertFirst(L, u)


7  for u in V(G) do
8      u.visited = False
9  L = List();
10 for u in V(G) do
11     if not u.visited then
12         dfs(G, u)
```

Analysis of complexity:

- Each vertex is passed to method dfs once (lines 5 and 12): $O(n)$;
- Adjacency list of each vertex is traversed once (line 3): $O(m)$;
- insertFirst inserts in the beginning of a linked list: $O(1)$;

Topological Sorting Algorithm

```
1  dfs(G, u):
2      u.visited = True
3      for v in adj(u) do
4          if not v.visited then
5              dfs(G, v)
6      insertFirst(L, u)
```

```
7  for u in V(G) do
8      u.visited = False
9  L = List();
10 for u in V(G) do
11     if not u.visited then
12         dfs(G, u)
```

Analysis of complexity:

- Each vertex is passed to method dfs once (lines 5 and 12): $O(n)$;
- Adjacency list of each vertex is traversed once (line 3): $O(m)$;
- insertFirst inserts in the beginning of a linked list: $O(1)$;
- Total complexity: $O(n + m)$.

Topological Sorting Algorithm

```
dfs(G, u):
    u.visited = True
    for v in adj(u) do
        if not v.visited then
            dfs(G, v)
    insertFirst(L, u)


for u in V(G) do
     u.visited = False
L = List();
for u in V(G) do
    if not u.visited then
        dfs(G, u)
```

Topological Sorting Algorithm

```
dfs(G, u):
    u.visited = True
    for v in adj(u) do
        if not v.visited then
            dfs(G, v)
    insertFirst(L, u)


for u in V(G) do
     u.visited = False
L = List();
for u in V(G) do
    if not u.visited then
        dfs(G, u)
```

Analysis of Correctness:

Topological Sorting Algorithm

```
dfs(G, u):
    u.visited = True
    for v in adj(u) do
        if not v.visited then
            dfs(G, v)
    insertFirst(L, u)


for u in V(G) do
    u.visited = False
L = List();
for u in V(G) do
    if not u.visited then
        dfs(G, u)
```

Analysis of Correctness:

- Vertices are added in the beginning of the list only if they do not have more output edges;

Topological Sorting Algorithm

```
dfs(G, u):
    u.visited = True
    for v in adj(u) do
        if not v.visited then
            dfs(G, v)
    insertFirst(L, u)


for u in V(G) do
    u.visited = False
L = List();
for u in V(G) do
    if not u.visited then
        dfs(G, u)
```

Analysis of Correctness:

- Vertices are added in the beginning of the list only if they do not have more output edges;
- Sinks are added first;

Topological Sorting Algorithm

```
dfs(G, u):
    u.visited = True
    for v in adj(u) do
        if not v.visited then
            dfs(G, v)
    insertFirst(L, u)


for u in V(G) do
     u.visited = False
L = List();
for u in V(G) do
    if not u.visited then
        dfs(G, u)
```

Analysis of Correctness:

- Vertices are added in the beginning of the list only if they do not have more output edges;
- Sinks are added first;
- Then “satisfied” vertices are added;

Topological Sorting Algorithm

```
dfs(G, u):
    u.visited = True
    for v in adj(u) do
        if not v.visited then
            dfs(G, v)
    insertFirst(L, u)


for u in V(G) do
     u.visited = False
L = List();
for u in V(G) do
    if not u.visited then
        dfs(G, u)
```

Analysis of Correctness:

- Vertices are added in the beginning of the list only if they do not have more output edges;
- Sinks are added first;
- Then “satisfied” vertices are added;
- All sources are now added.

Exercises

1. Modify DFS algorithm to verify if a graph is acyclic.
2. Suppose G is connected, how we can use DFS algorithm to get a spanning tree of G?
3. DFS algorithm can be used to verify if a graph is connected?
4. Propose an alternative algorithm to solve the topological sorting problem without using DFS, but with the same complexity of time.

Bibliography

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. and STEIN, C. *Introduction to Algorithms*, 3rd edition, MIT Press, 2009.