

# Design and Analysis of Algorithms

## Part IV: Graph Algorithms

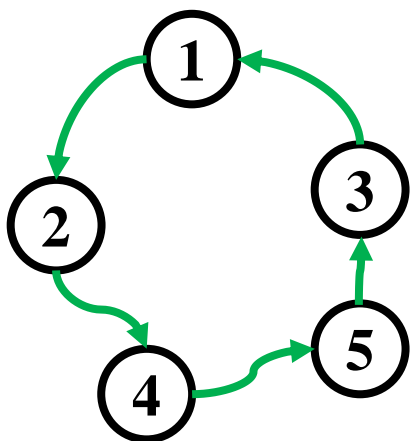
### Lecture 25: Topological Sort

---

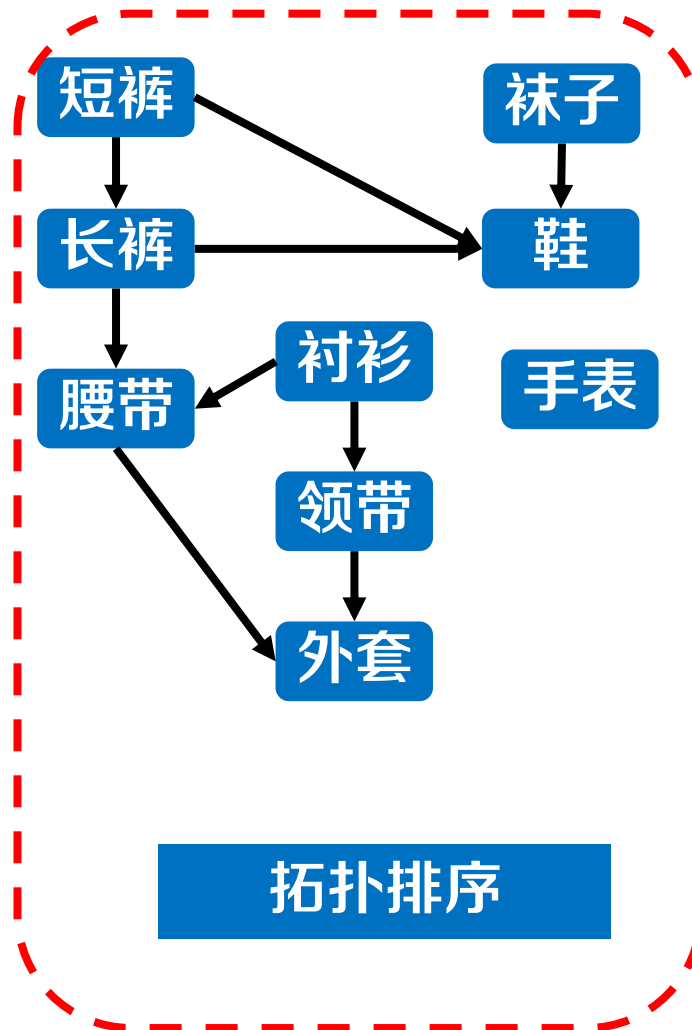
童咏昕

北京航空航天大学  
计算机学院

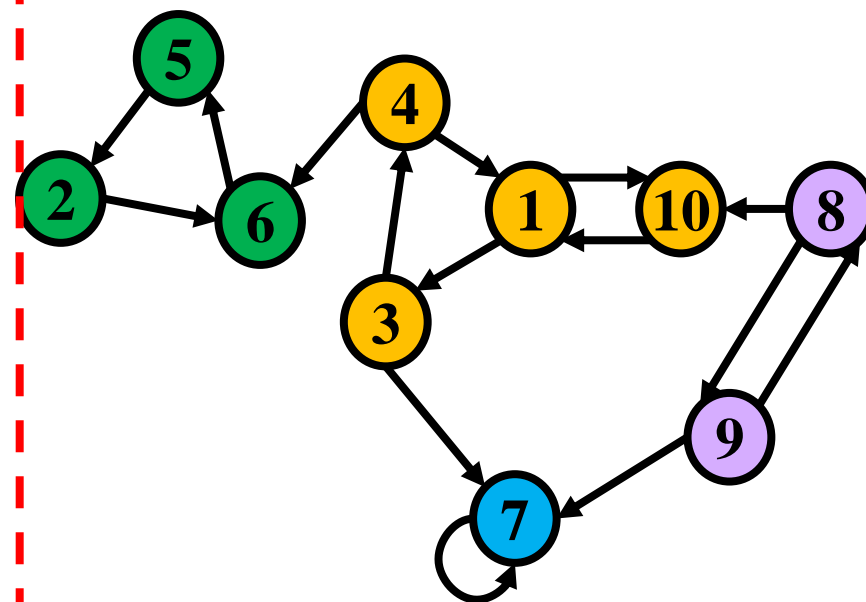
- 在算法课程第四部分“图算法”主题中，我们将主要聚焦于如下经典问题：
  - Basic Concepts in Graph Algorithms (图算法的基本概念)
  - Breadth-First Search (BFS, 广度优先搜索)
  - Depth-First Search (DFS, 深度优先搜索)
  - Cycle Detection (环路检测)
  - **Topological Sort (拓扑排序)**
  - Strongly Connected Components (强连通分量)
  - Minimum Spanning Trees (最小生成树)
  - Single Source Shortest Path (单源最短路径)
  - All-Pairs Shortest Paths (所有点对最短路径)
  - Bipartite Graph Matching (二分图匹配)
  - Maximum/Network Flows (最大流/网络流)



环路的存在性判断



拓扑排序



强连通分量

问题定义

广度优先策略

深度优先策略

算法分析

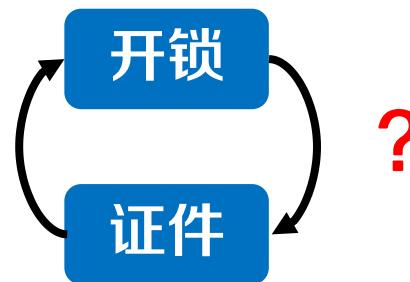
- 穿衣步骤

- 有向无环图（Directed Acyclic Graph, DAG）：表示事件发生的先后顺序

有向：规定先后顺序

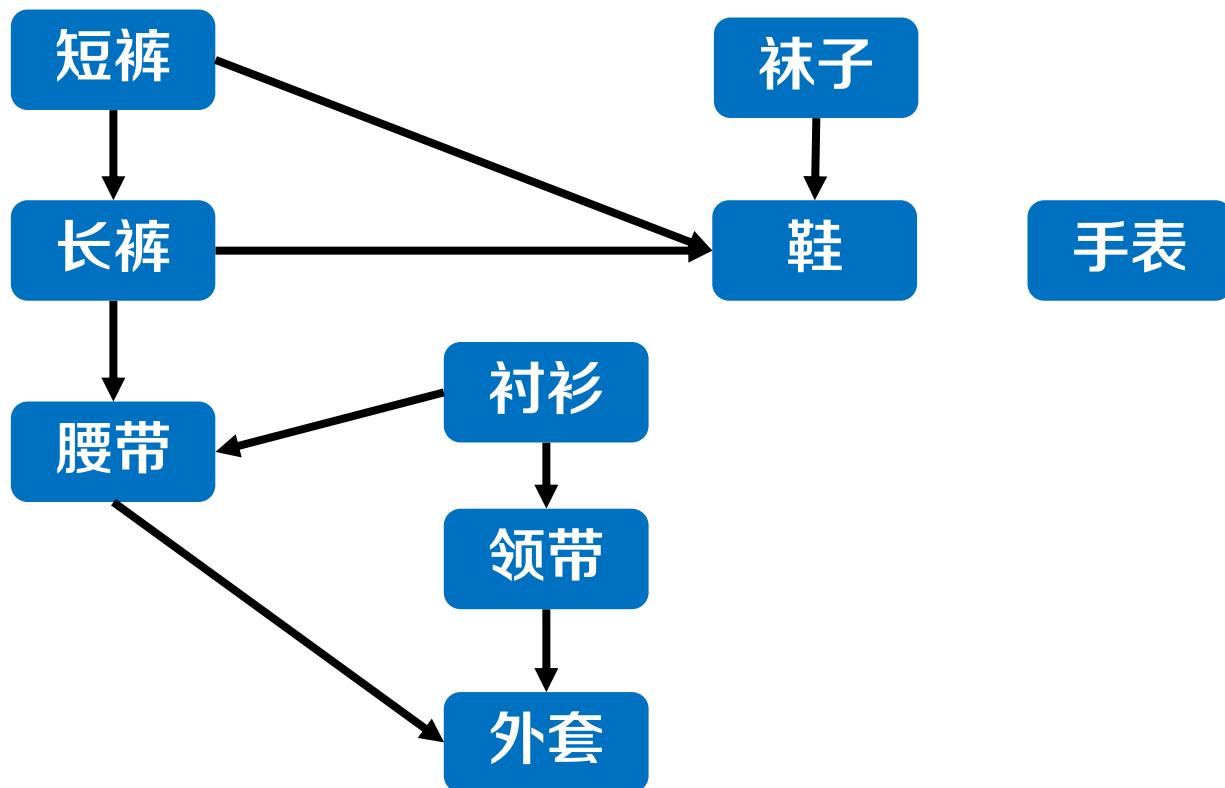


无环：避免相互依赖



- 穿衣步骤

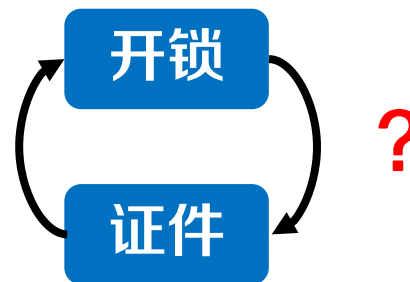
- 有向无环图（Directed Acyclic Graph, DAG）：表示事件发生的先后顺序



有向：规定先后顺序

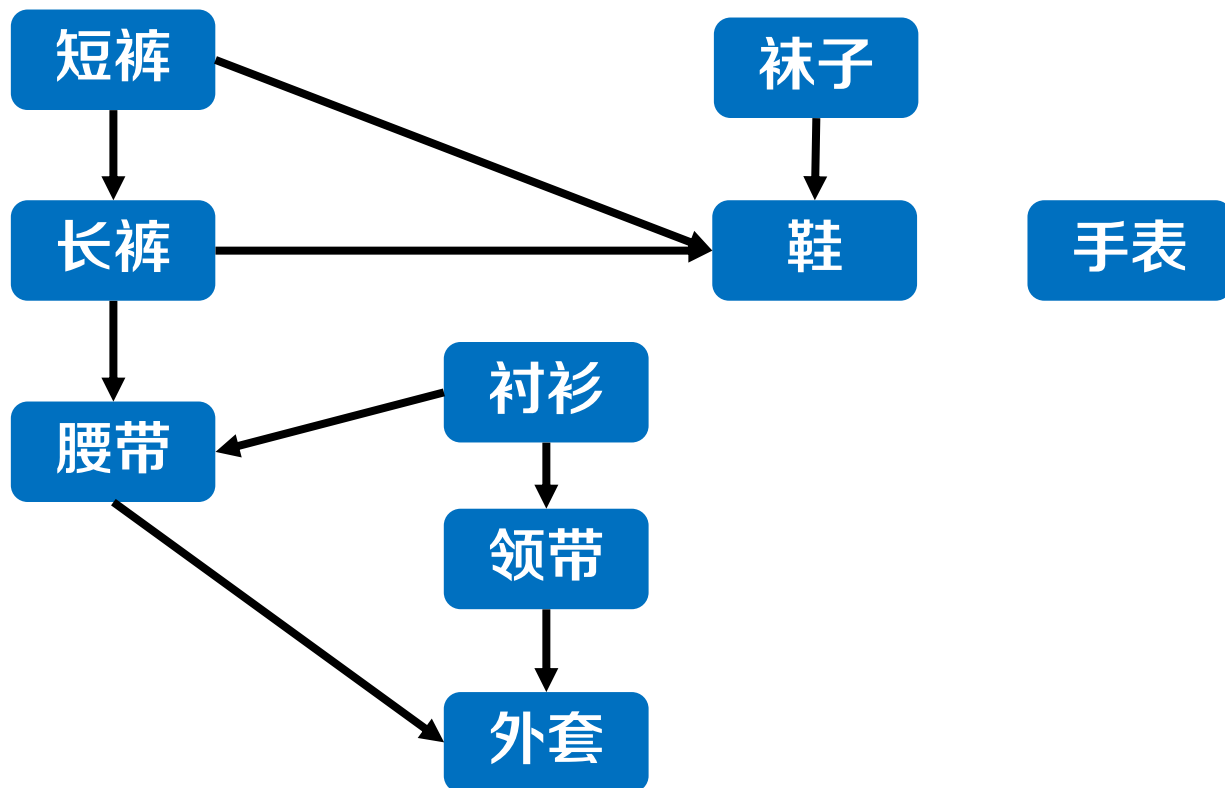


无环：避免相互依赖



- 穿衣步骤

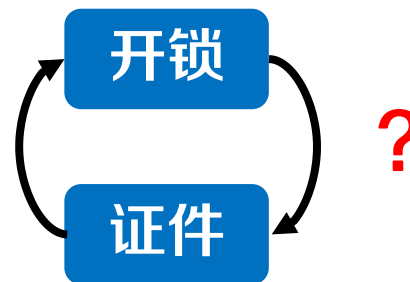
- 有向无环图（Directed Acyclic Graph, DAG）：表示事件发生的先后顺序



有向：规定先后顺序



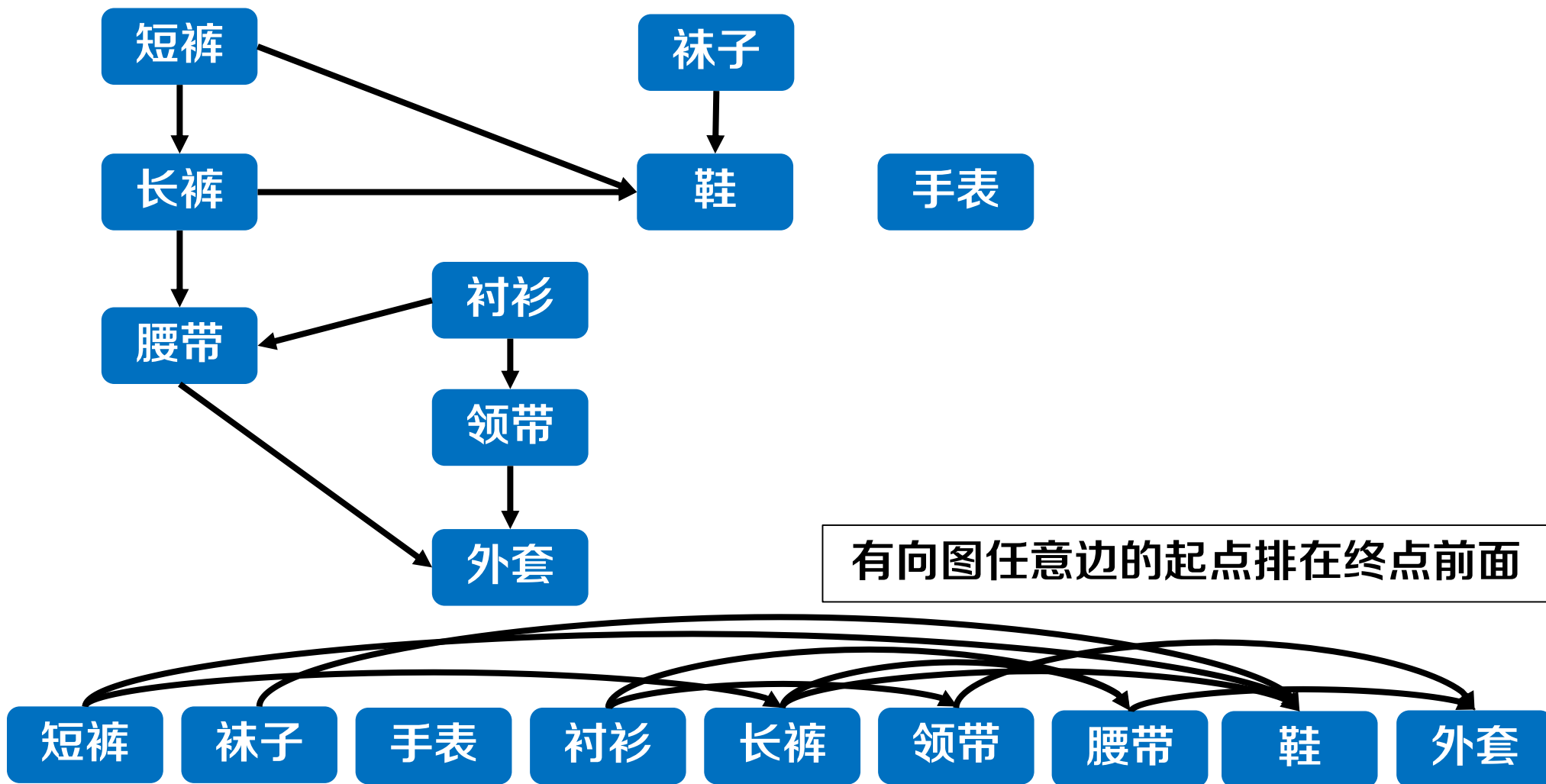
无环：避免相互依赖



问题：如何确定一个可行的穿衣顺序？

- 穿衣步骤

- 有向无环图（Directed Acyclic Graph, DAG）：表示事件发生的先后顺序





## 拓扑排序

### Topological Sort

#### 输入

- 有向无环图  $G = \langle V, E \rangle$

#### 输出

- 图顶点 $V$ 的拓扑序 $S$ ，满足：  
对任意有向边 $(u, v)$ ，排序后 $u$ 在 $v$ 之前

## 拓扑排序

### Topological Sort

#### 输入

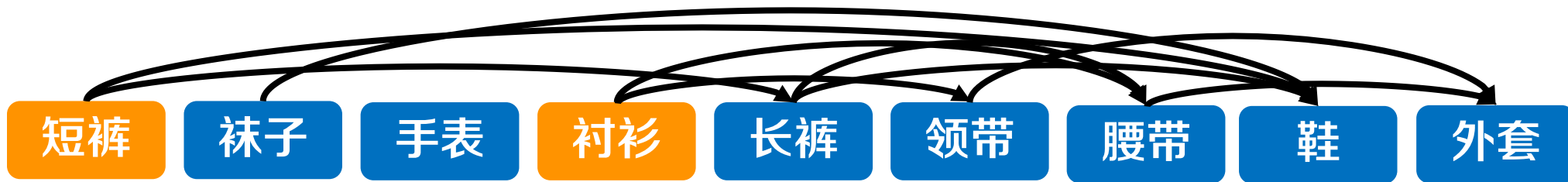
- 有向无环图  $G = \langle V, E \rangle$

#### 输出

- 图顶点  $V$  的拓扑序  $S$ ，满足：  
对任意有向边  $(u, v)$ ，排序后  $u$  在  $v$  之前

- 拓扑序不唯一

有向图任意边的起点排在终点前面



## 拓扑排序

### Topological Sort

#### 输入

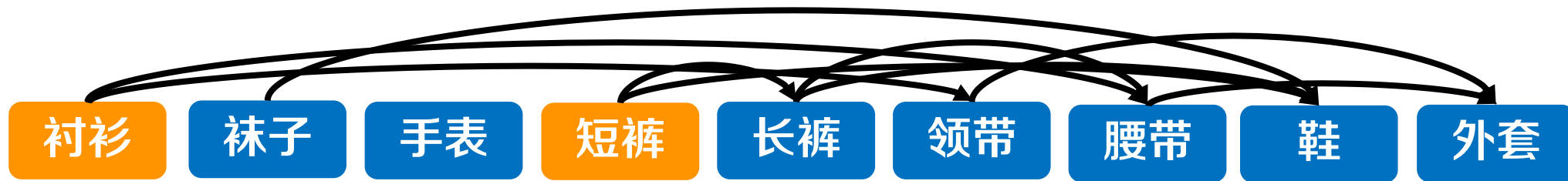
- 有向无环图  $G = \langle V, E \rangle$

#### 输出

- 图顶点  $V$  的拓扑序  $S$ ，满足：  
对任意有向边  $(u, v)$ ，排序后  $u$  在  $v$  之前

- 拓扑序不唯一

有向图任意边的起点排在终点前面



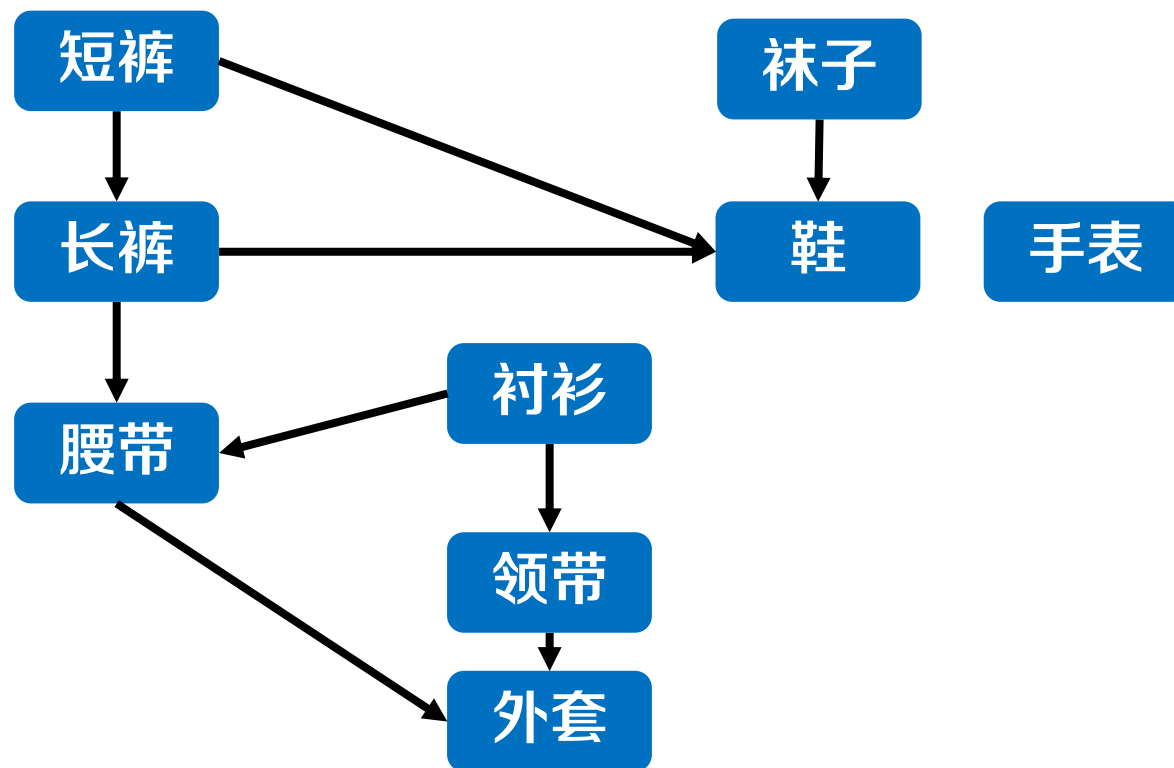
问题定义

广度优先策略

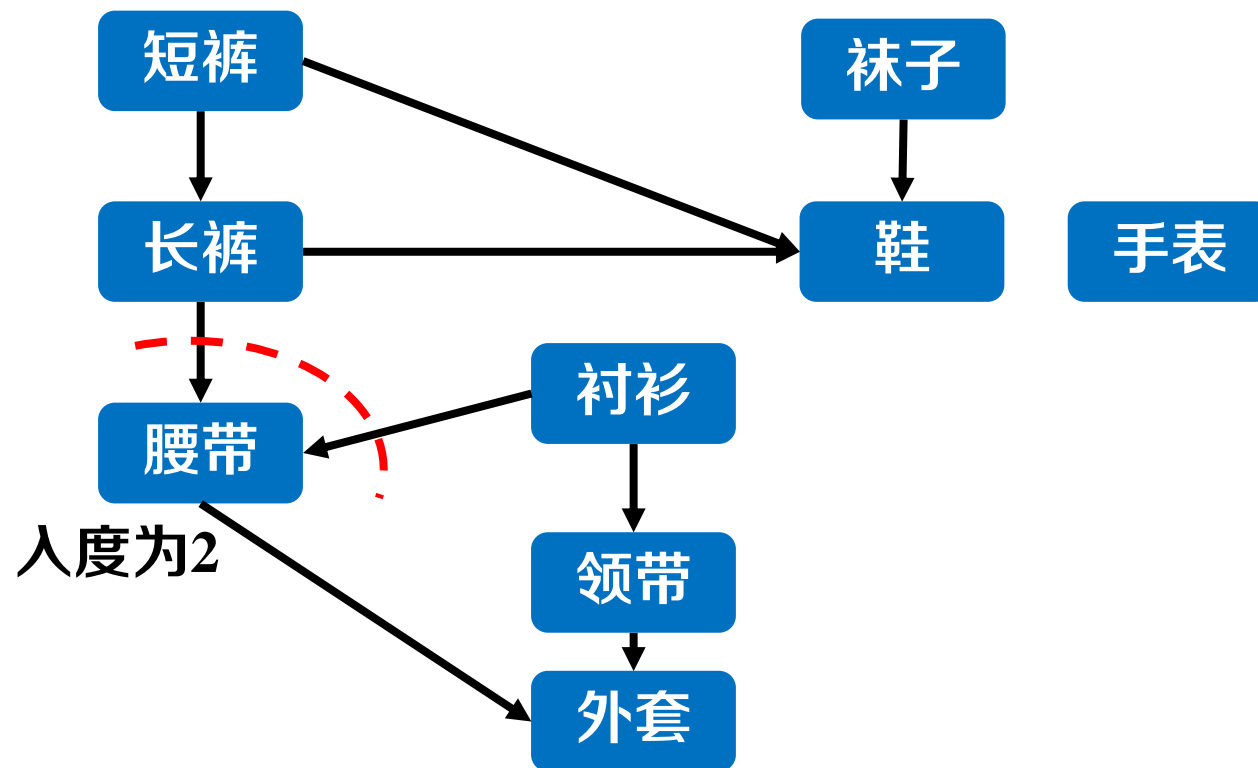
深度优先策略

算法分析

- 有向图顶点的度分为入度和出度

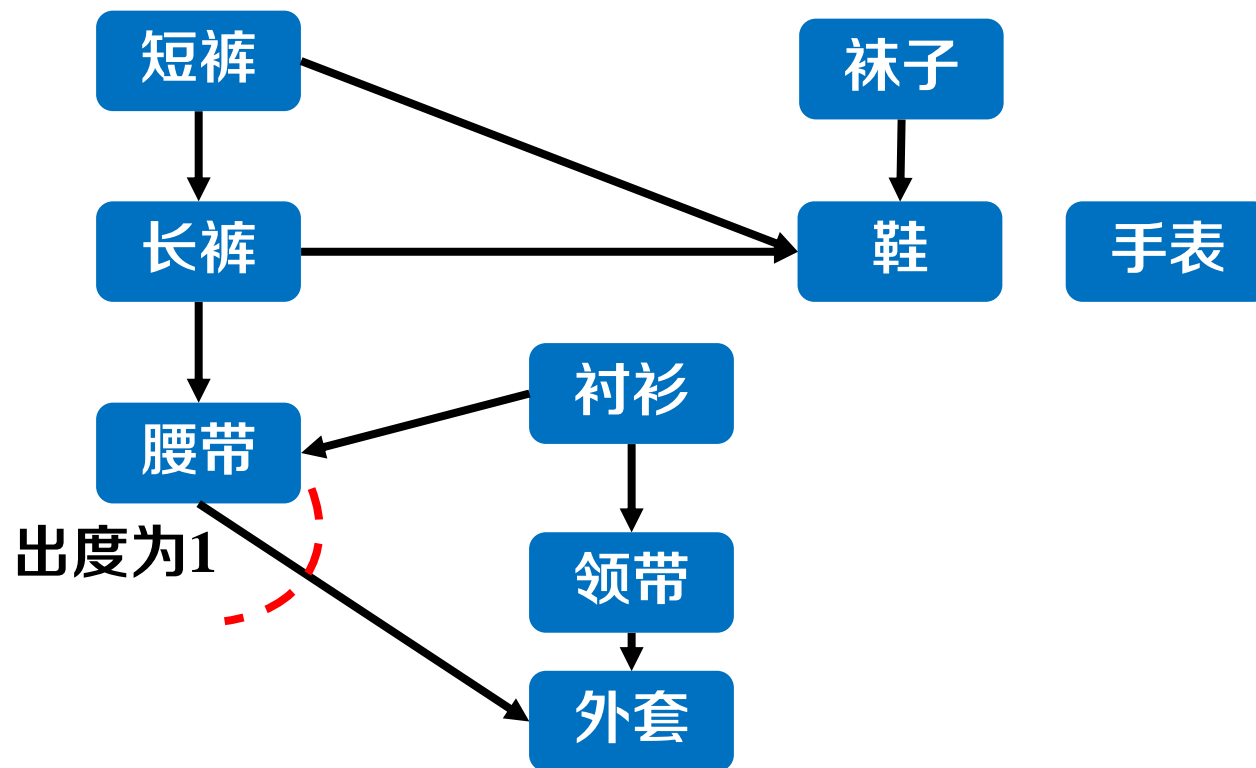


- 有向图顶点的度分为入度和出度
  - 顶点 $u$ 的入度：起点为 $u$ 的边数



- 有向图顶点的度分为入度和出度

- 顶点 $u$ 的入度：起点为 $u$ 的边数
- 顶点 $u$ 的出度：终点为 $u$ 的边数

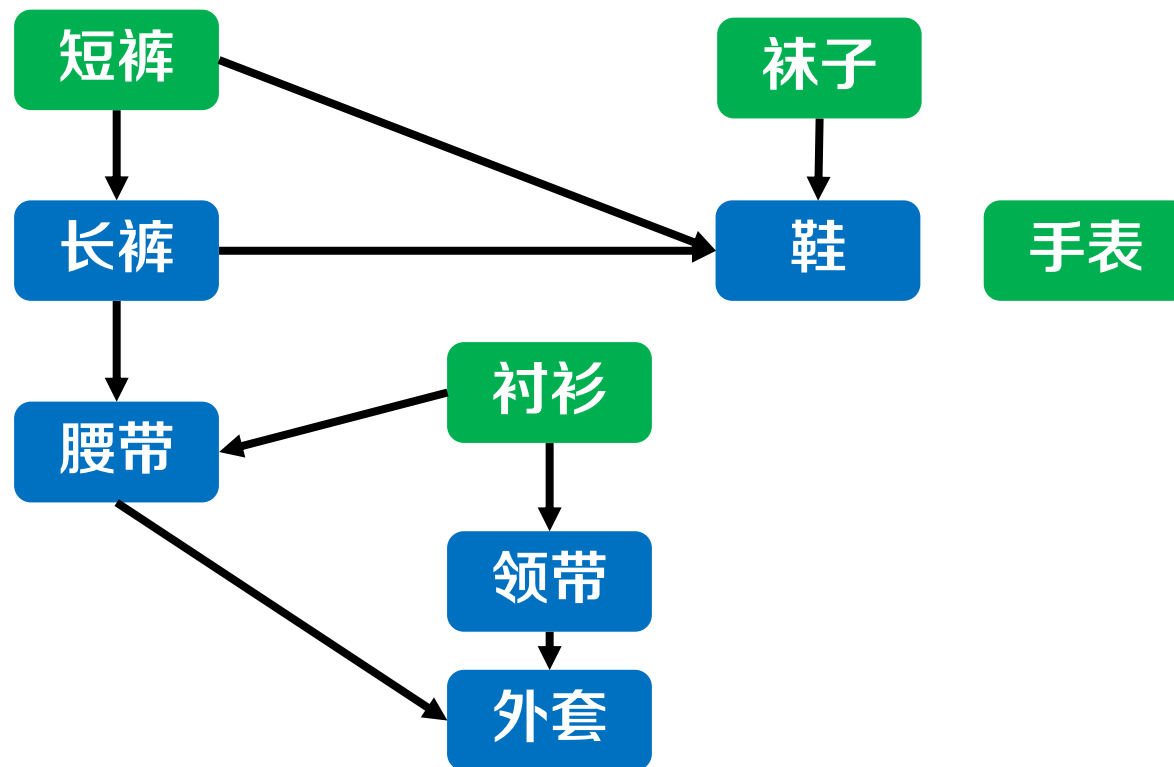


- 有向图顶点的度分为入度和出度

- 顶点 $u$ 的入度：起点为 $u$ 的边数
- 顶点 $u$ 的出度：终点为 $u$ 的边数

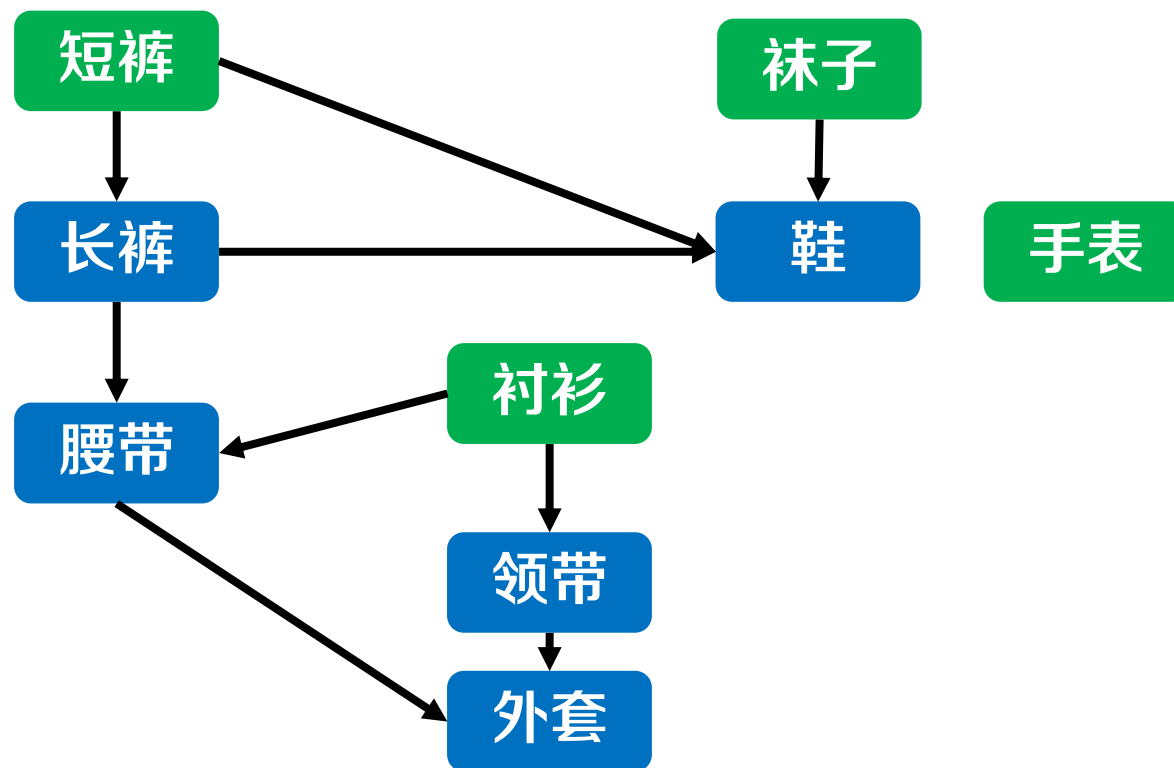
- 若顶点入度为0

- 所对应事件无制约，可直接完成

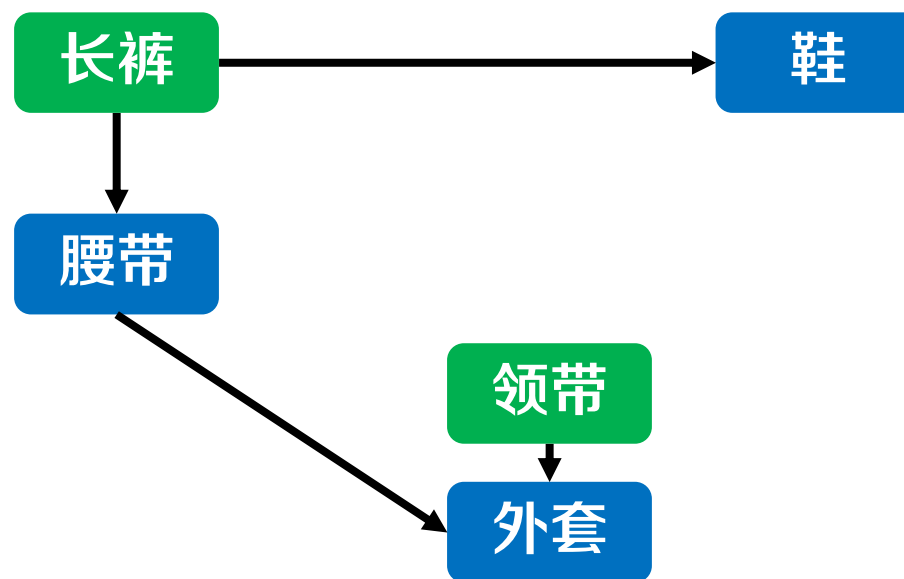




- 完成入度为0点对应的事件



- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成



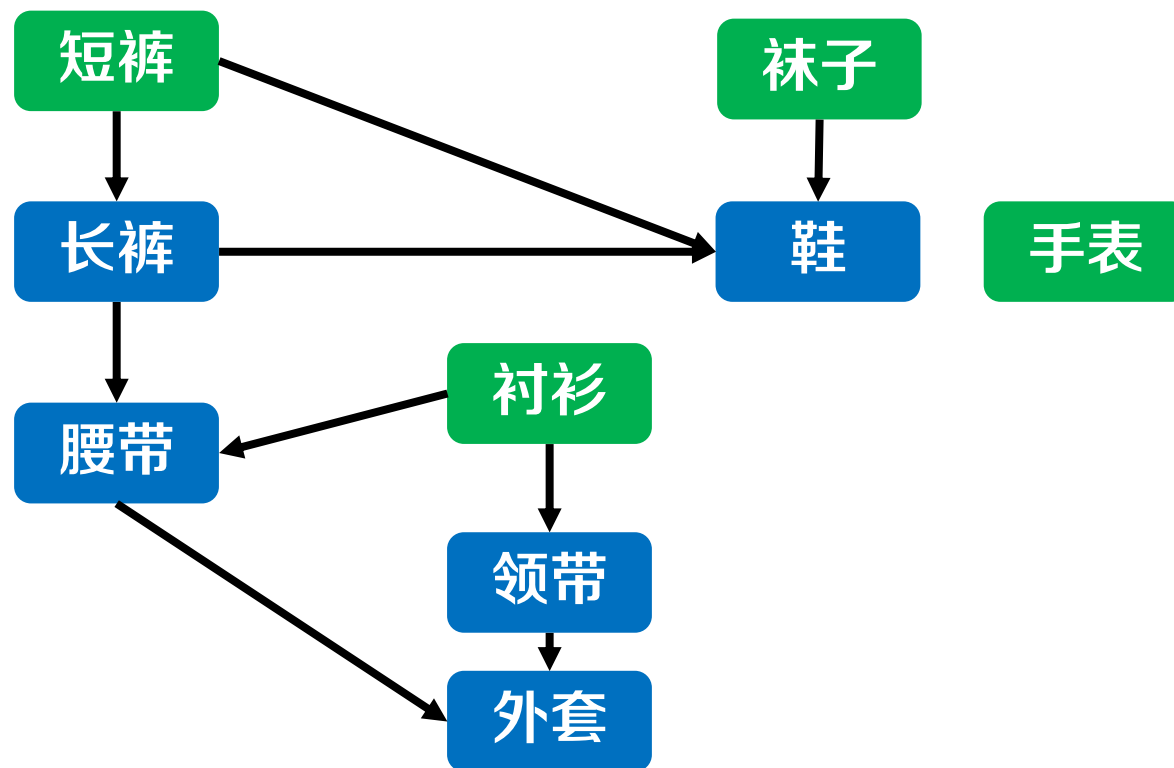
- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点



拓扑序 记录已完成事件



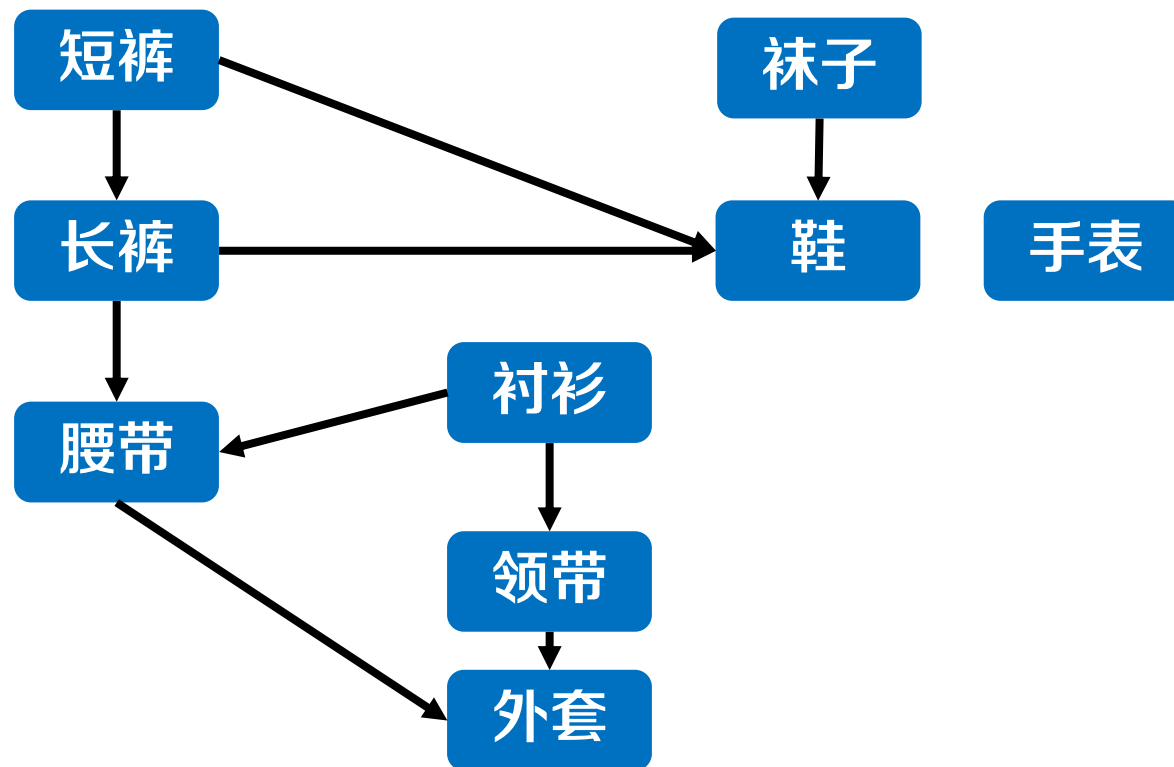
- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点



拓扑序 记录已完成事件



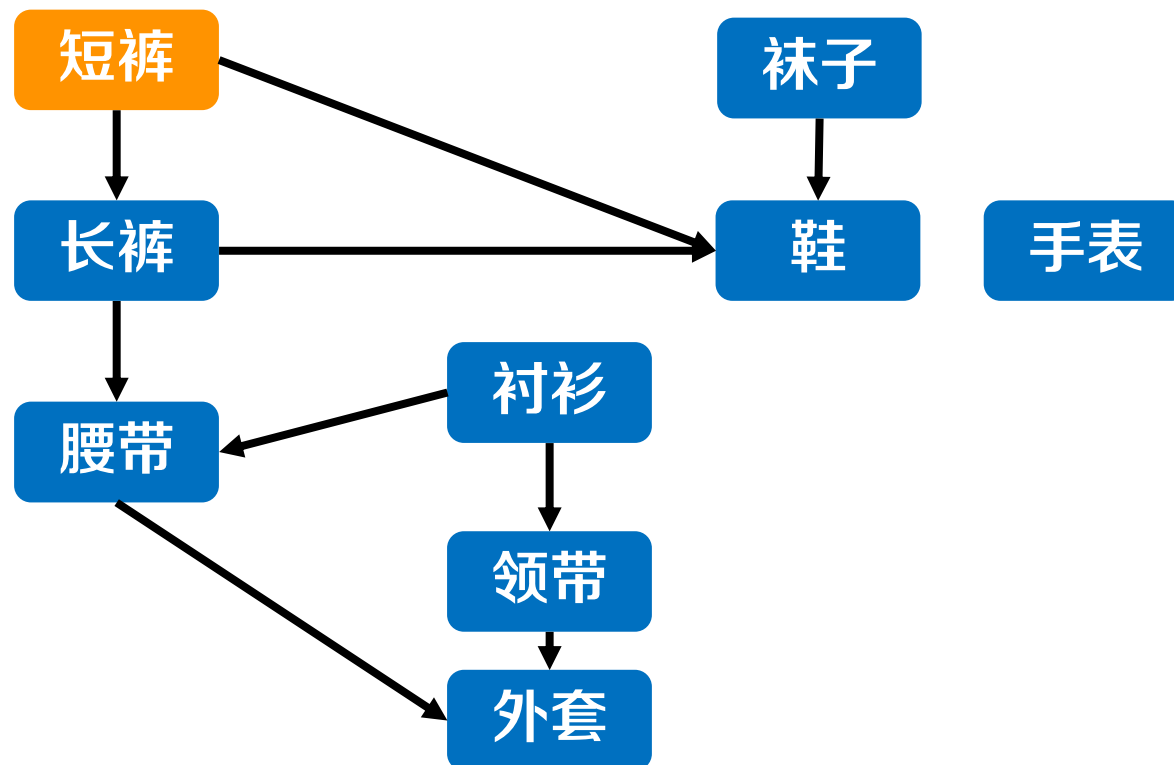
- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点



拓扑序 记录已完成事件



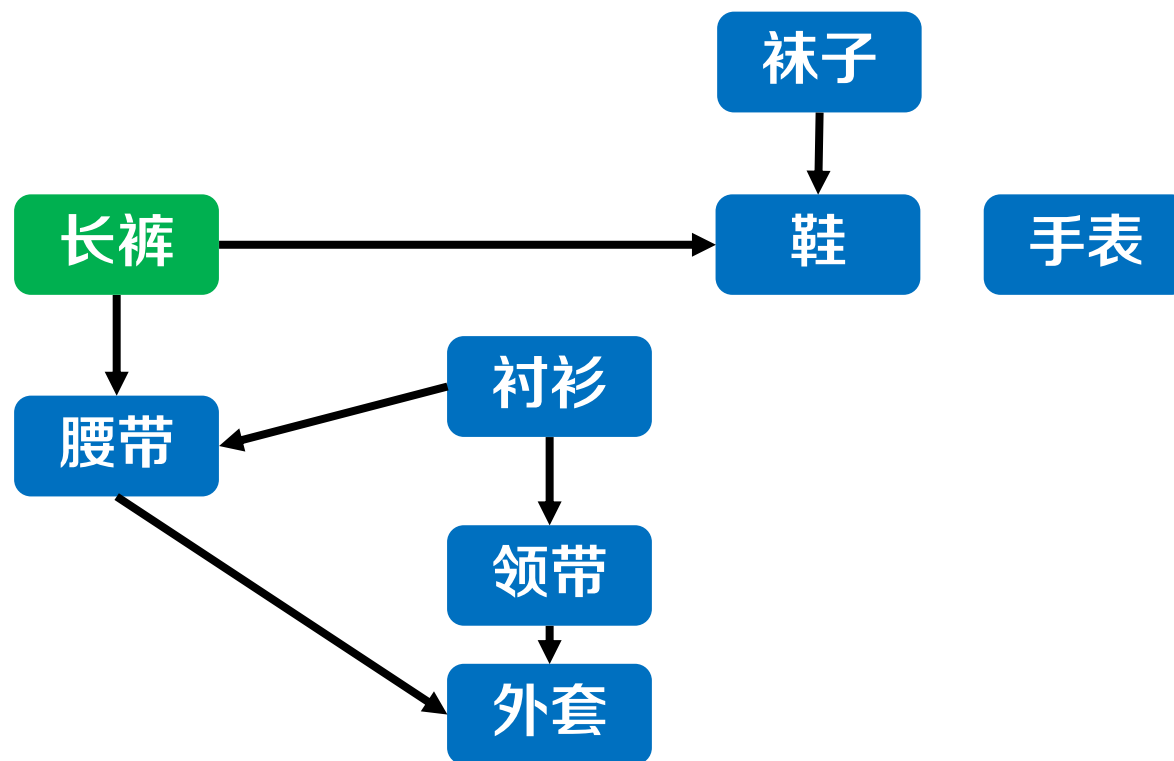
- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点



拓扑序 记录已完成事件



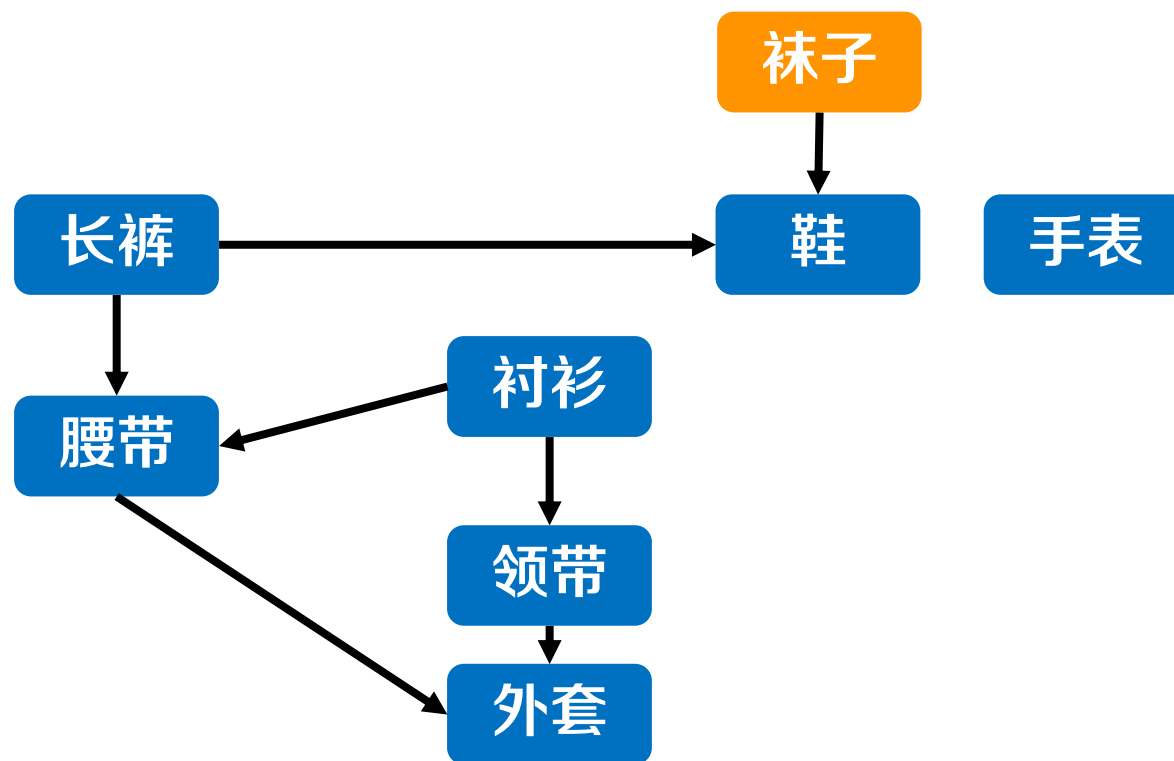
- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点



拓扑序 记录已完成事件



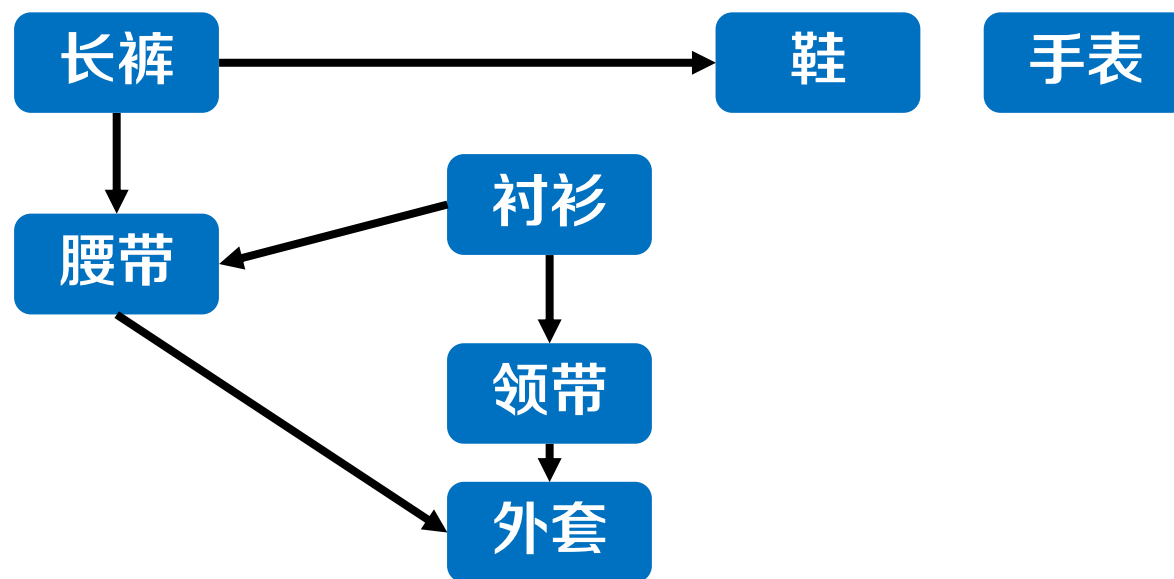
- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点



拓扑序 记录已完成事件





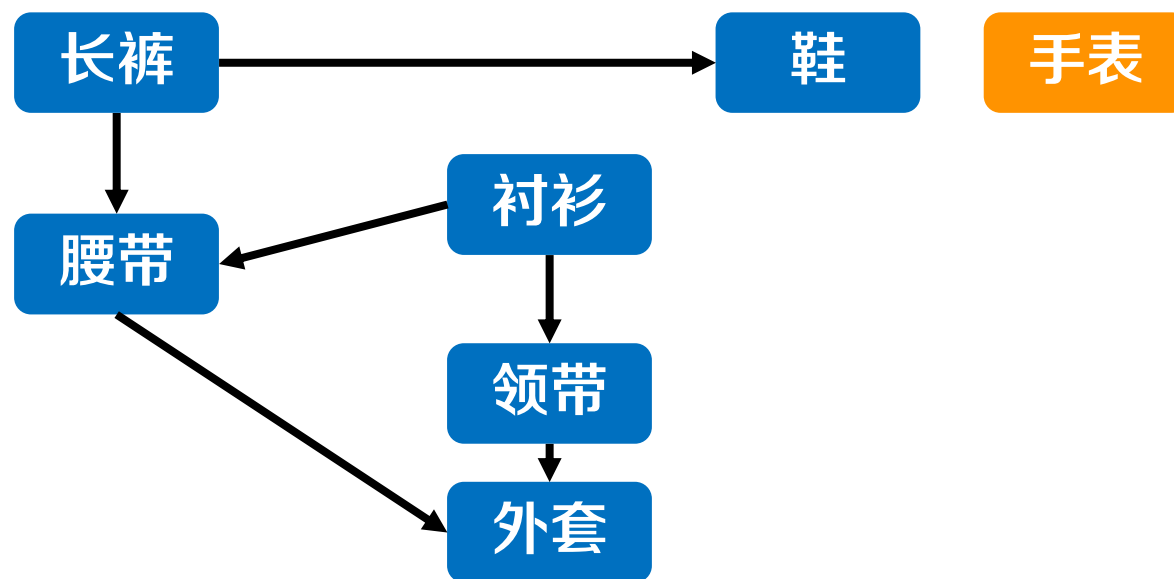
- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点



拓扑序 记录已完成事件



- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点

衬衫

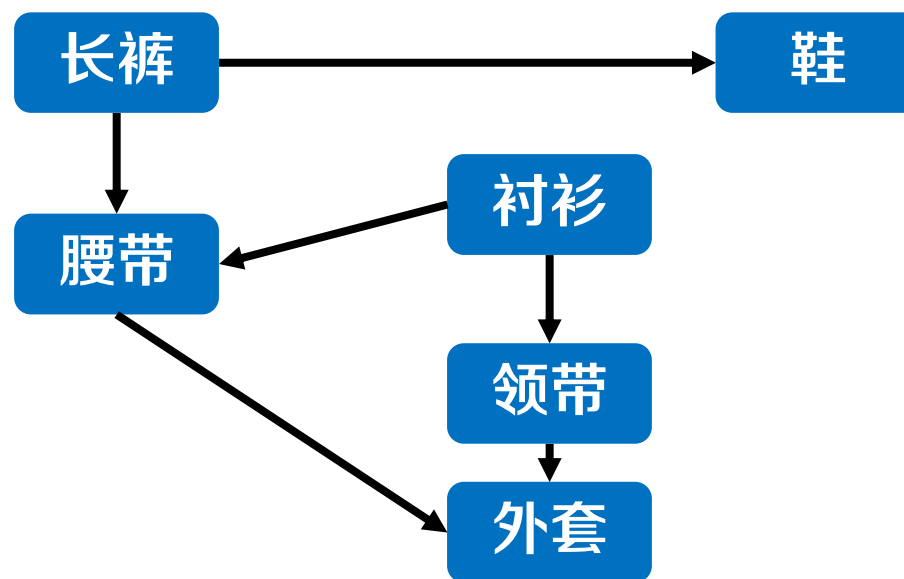
长裤

拓扑序 记录已完成事件

短裤

袜子

手表



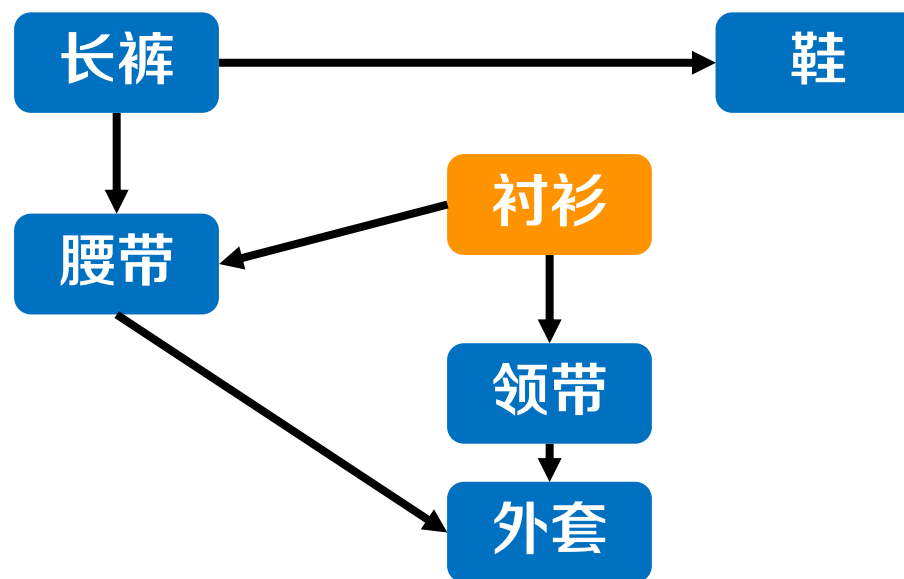
- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点



拓扑序 记录已完成事件



- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点

长裤

领带

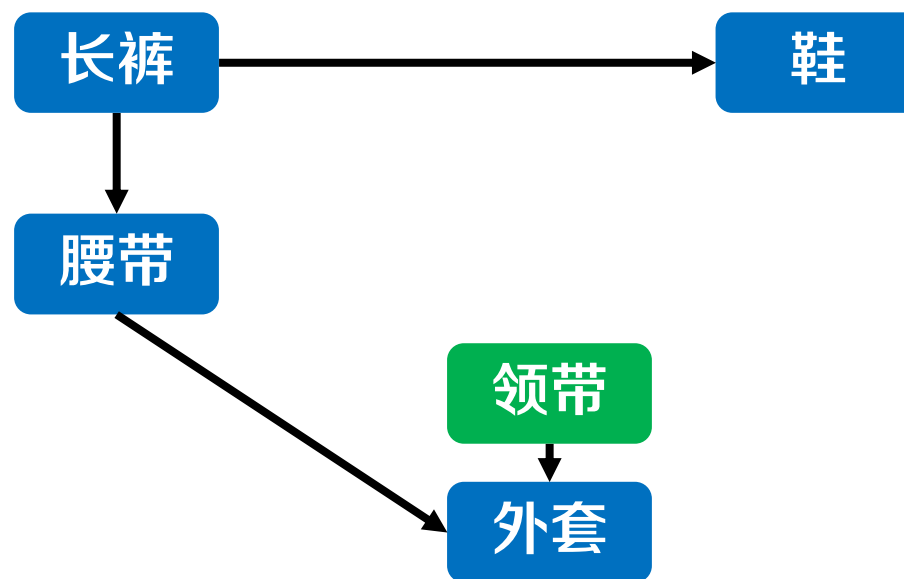
拓扑序 记录已完成事件

短裤

袜子

手表

衬衫



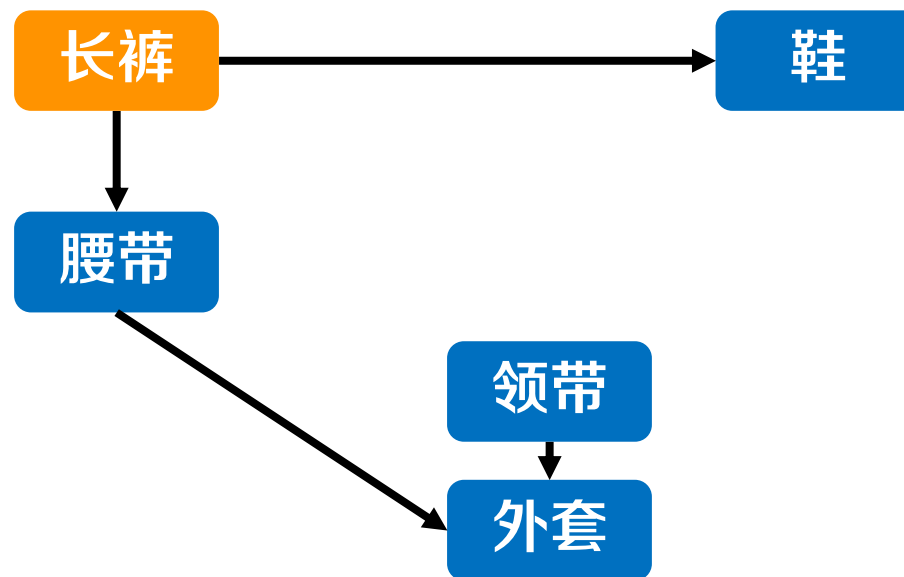
- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点

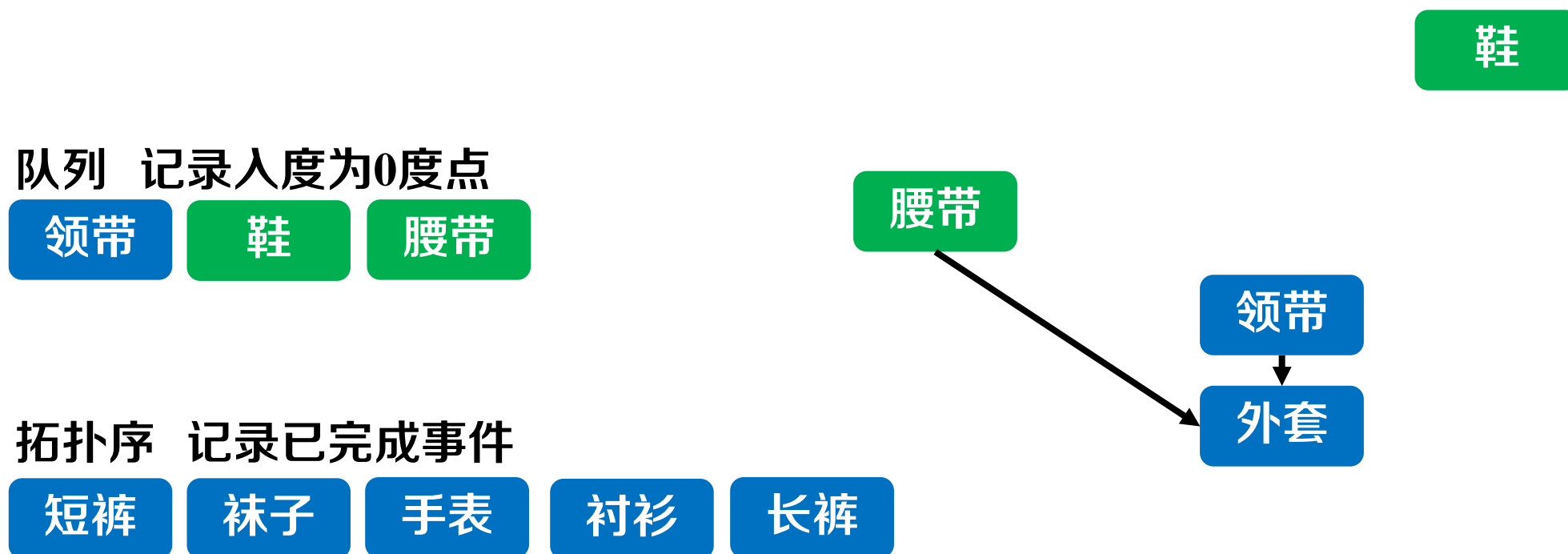


拓扑序 记录已完成事件



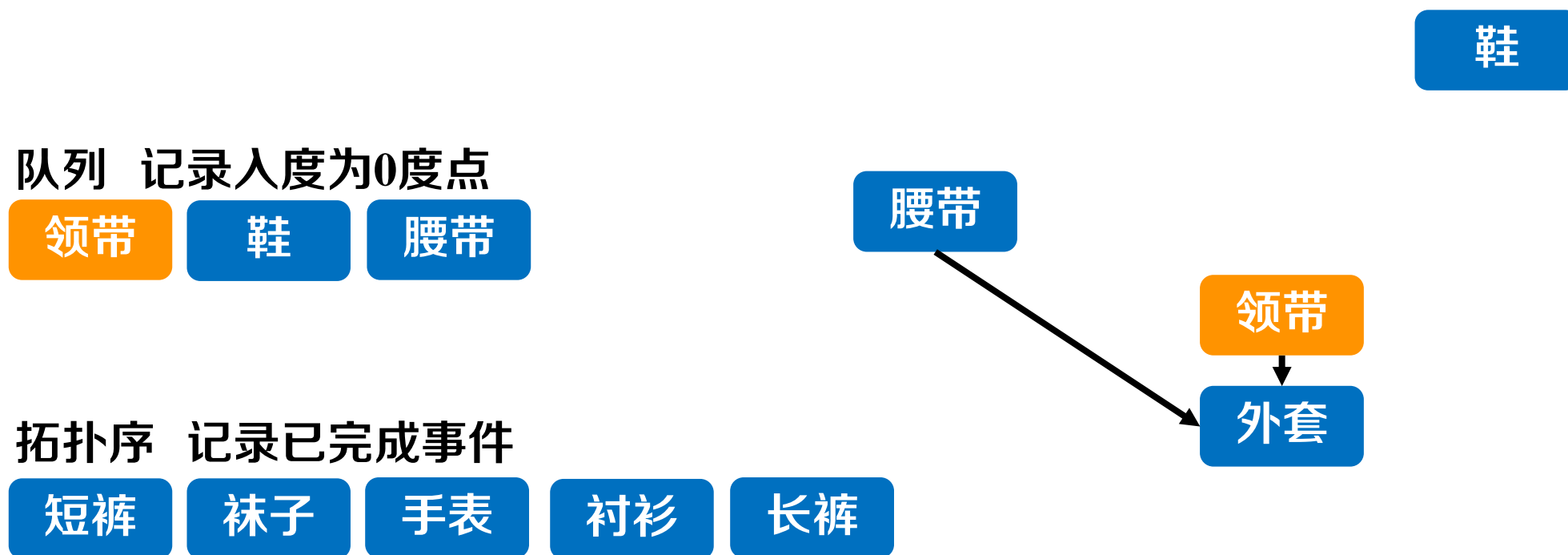
- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成



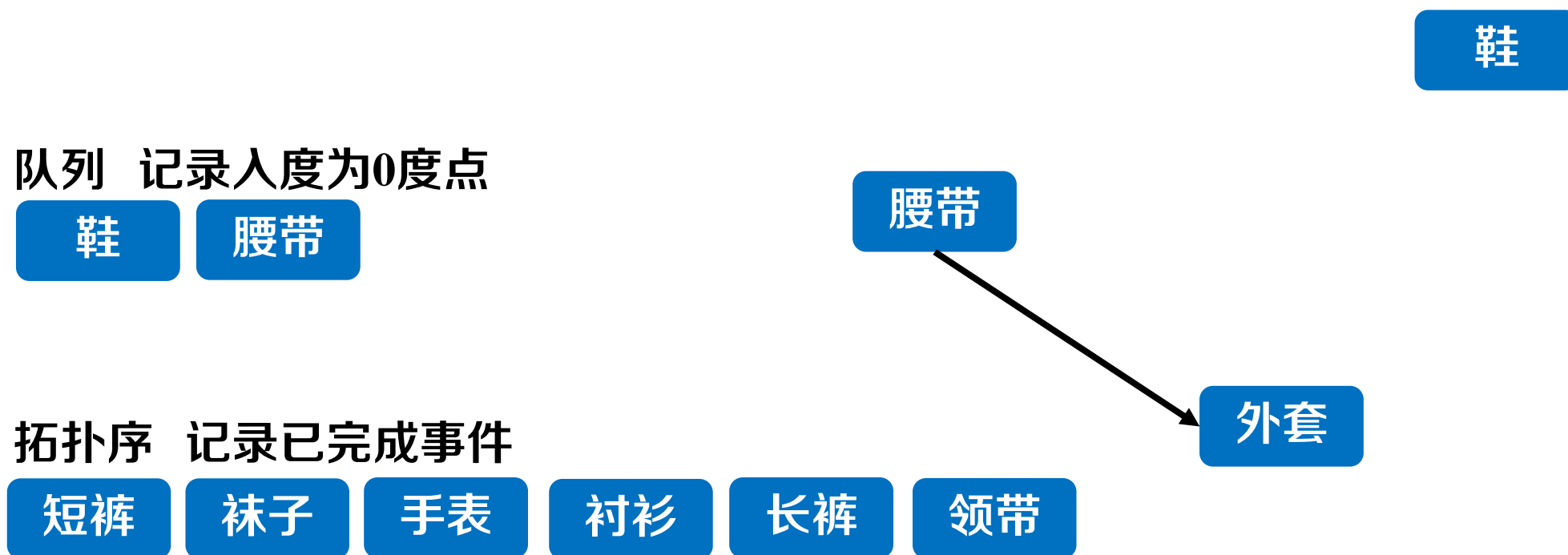
- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成



- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成





- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点



腰带

外套

拓扑序 记录已完成事件



鞋

- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点

腰带

腰带

外套

拓扑序 记录已完成事件

短裤

袜子

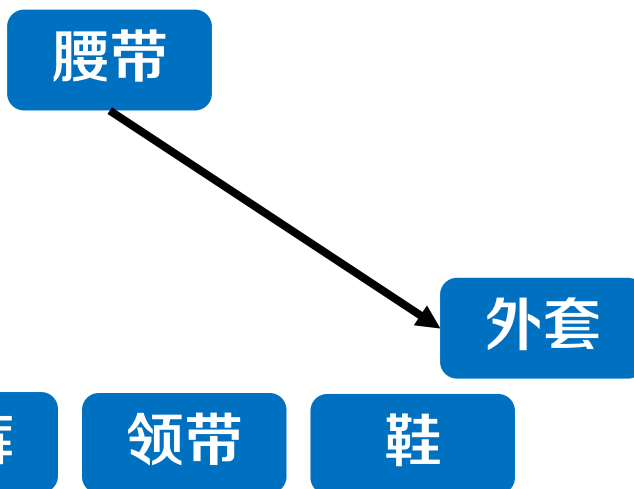
手表

衬衫

长裤

领带

鞋



- 算法思想

- 完成入度为0点对应的事件
- 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点

腰带

腰带

外套

拓扑序 记录已完成事件

短裤

袜子

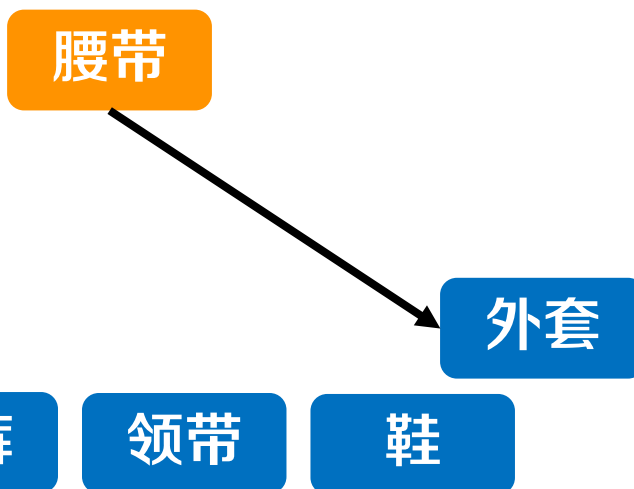
手表

衬衫

长裤

领带

鞋



- 算法思想
  - 完成入度为0点对应的事件
  - 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点

外套

拓扑序 记录已完成事件

短裤

袜子

手表

衬衫

长裤

领带

鞋

腰带

外套

- 算法思想
  - 完成入度为0点对应的事件
  - 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点

外套

拓扑序 记录已完成事件

短裤

袜子

手表

衬衫

长裤

领带

鞋

腰带

外套

- 算法思想
  - 完成入度为0点对应的事件
  - 删除完成事件，产生新的入度为0点，继续完成

队列 记录入度为0度点

拓扑序 记录已完成事件

短裤

袜子

手表

衬衫

长裤

领带

鞋

腰带

外套

- **Topological-Sort-BFS( $G$ )**

输入: 图 $G$

输出: 顶点拓扑序

初始化空队列 $Q$

初始化

```
for  $v \in V$  do
    if  $v.in\_degree = 0$  then
        |  $Q.Enqueue(v)$ 
    end
end
while not  $Q.is\_empty()$  do
     $u \leftarrow Q.Dequeue()$ 
    print  $u$ 
    for  $v \in G.Adj(u)$  do
        |  $v.in\_degree \leftarrow v.in\_degree - 1$ 
        | if  $v.in\_degree = 0$  then
        | |  $Q.Enqueue(v)$ 
        | end
    end
end
end
```

- **Topological-Sort-BFS( $G$ )**

输入: 图 $G$

输出: 顶点拓扑序

初始化空队列 $Q$

for  $v \in V$  do

    if  $v.in\_degree = 0$  then

$Q.Enqueue(v)$

    end

end

while not  $Q.is\_empty()$  do

$u \leftarrow Q.Dequeue()$

    print  $u$

    for  $v \in G.Adj(u)$  do

$v.in\_degree \leftarrow v.in\_degree - 1$

        if  $v.in\_degree = 0$  then

$Q.Enqueue(v)$

        end

    end

end

入度为0, 加入队列



- **Topological-Sort-BFS( $G$ )**

输入: 图 $G$

输出: 顶点拓扑序

初始化空队列 $Q$

for  $v \in V$  do

    if  $v.in\_degree = 0$  then

$Q.Enqueue(v)$

    end

end

while not  $Q.is\_empty()$  do

$u \leftarrow Q.Dequeue()$

    print  $u$

    for  $v \in G.Adj(u)$  do

$v.in\_degree \leftarrow v.in\_degree - 1$

        if  $v.in\_degree = 0$  then

$Q.Enqueue(v)$

        end

    end

end

完成事件

- **Topological-Sort-BFS( $G$ )**

输入: 图 $G$

输出: 顶点拓扑序

初始化空队列 $Q$

for  $v \in V$  do

    if  $v.in\_degree = 0$  then

$Q.Enqueue(v)$

    end

end

while not  $Q.is\_empty()$  do

$u \leftarrow Q.Dequeue()$

    print  $u$

    for  $v \in G.Adj(u)$  do

$v.in\_degree \leftarrow v.in\_degree - 1$

        if  $v.in\_degree = 0$  then

$Q.Enqueue(v)$

        end

    end

end

删除事件，更新入度

- **Topological-Sort-BFS( $G$ )**

输入: 图 $G$

输出: 顶点拓扑序

初始化空队列 $Q$

for  $v \in V$  do

    if  $v.in\_degree = 0$  then

$Q.Enqueue(v)$

    end

end

while not  $Q.is\_empty()$  do

$u \leftarrow Q.Dequeue()$

    print  $u$

    for  $v \in G.Adj(u)$  do

$v.in\_degree \leftarrow v.in\_degree - 1$

        if  $v.in\_degree = 0$  then

$Q.Enqueue(v)$

        end

    end

end

入度为0，加入队列

- Topological-Sort-BFS( $G$ )

输入: 图 $G$

输出: 顶点拓扑序

初始化空队列 $Q$

for  $v \in V$  do

    if  $v.in\_degree = 0$  then

$Q.Enqueue(v)$

    end

end

while not  $Q.is\_empty()$  do

$u \leftarrow Q.Dequeue()$

    print  $u$

    for  $v \in G.Adj(u)$  do

$v.in\_degree \leftarrow v.in\_degree - 1$

        if  $v.in\_degree = 0$  then

$Q.Enqueue(v)$

        end

    end

end

$O(|V|)$

$$\sum_{v \in V} deg(v) = O(|E|)$$

$$O(\sum_{v \in V} (1 + |Adj[v]|)) \\ = O(|V| + |E|)$$

时间复杂度:  $O(|V| + |E|)$

问题定义

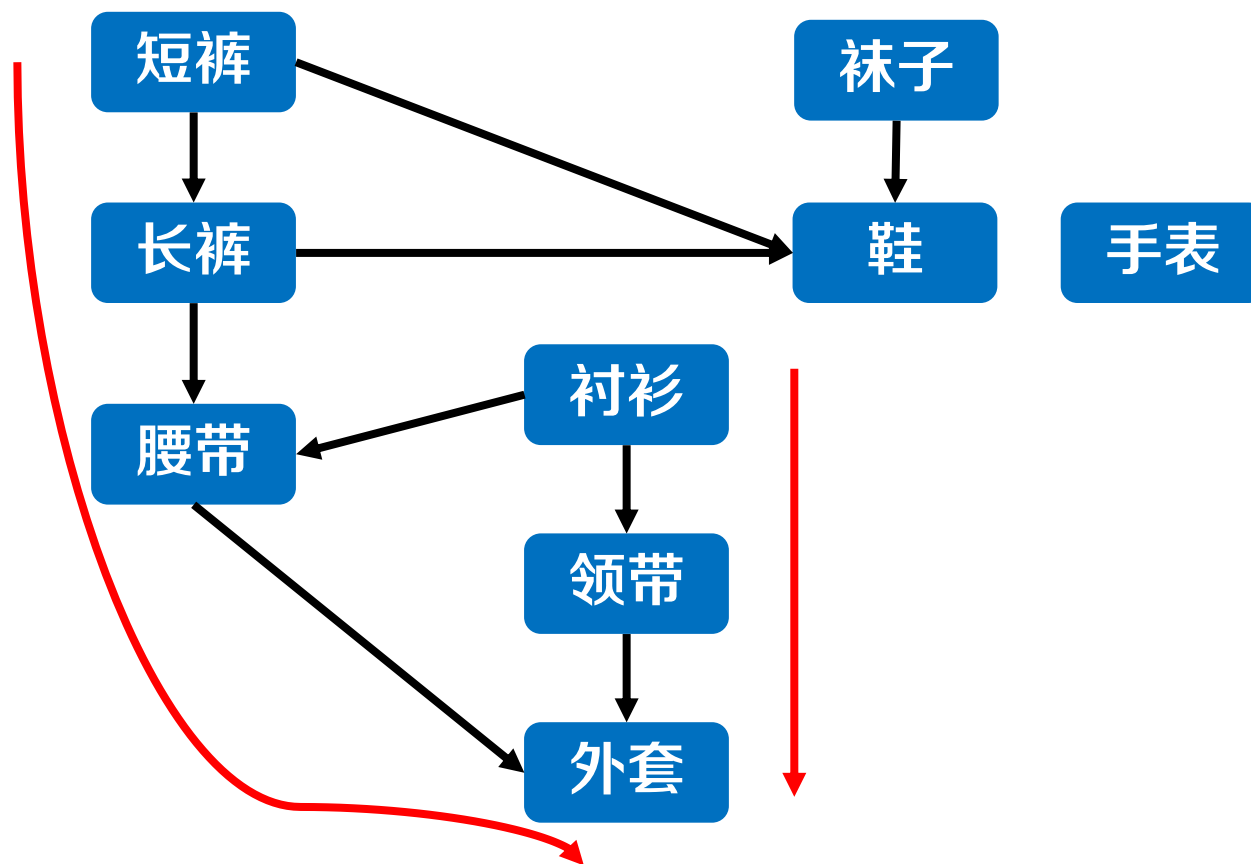
广度优先策略

深度优先策略

算法分析

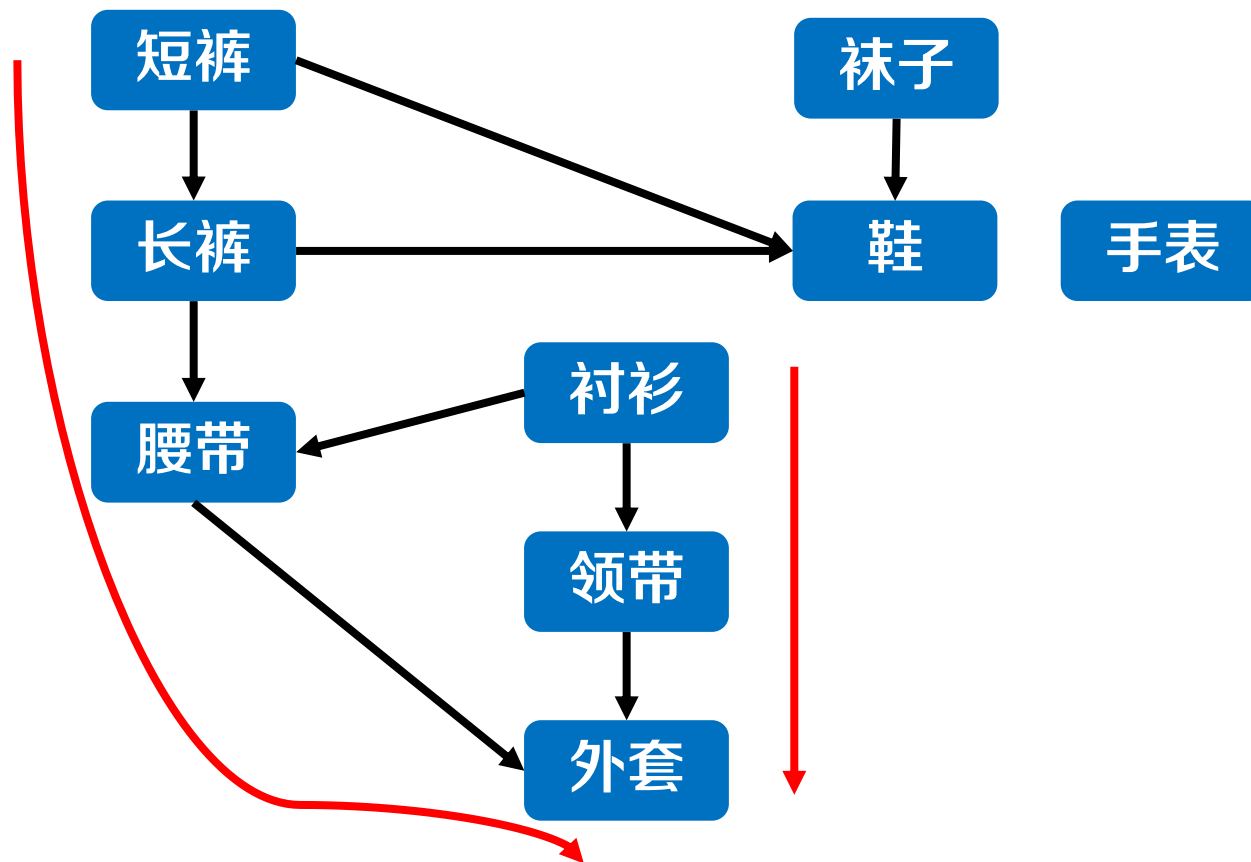
- 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后



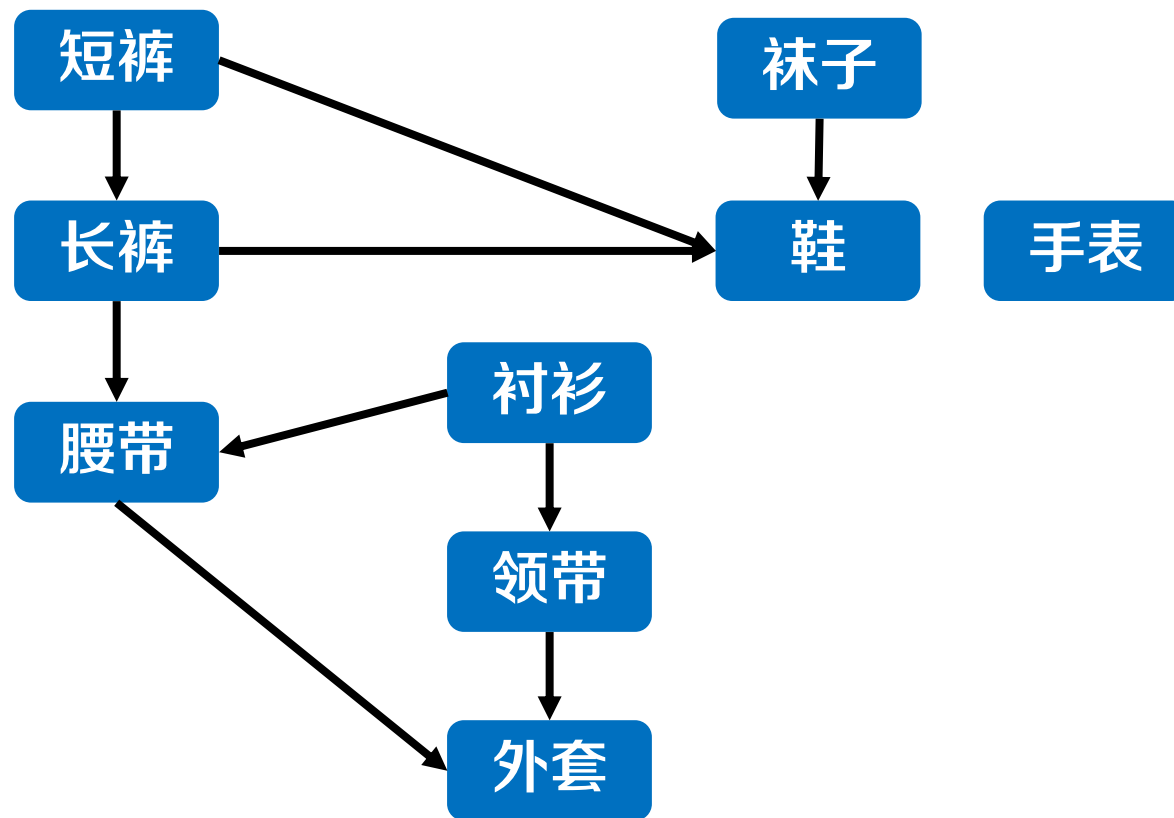
- 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚
  - 完成时刻越早



- 从DFS的视角观察

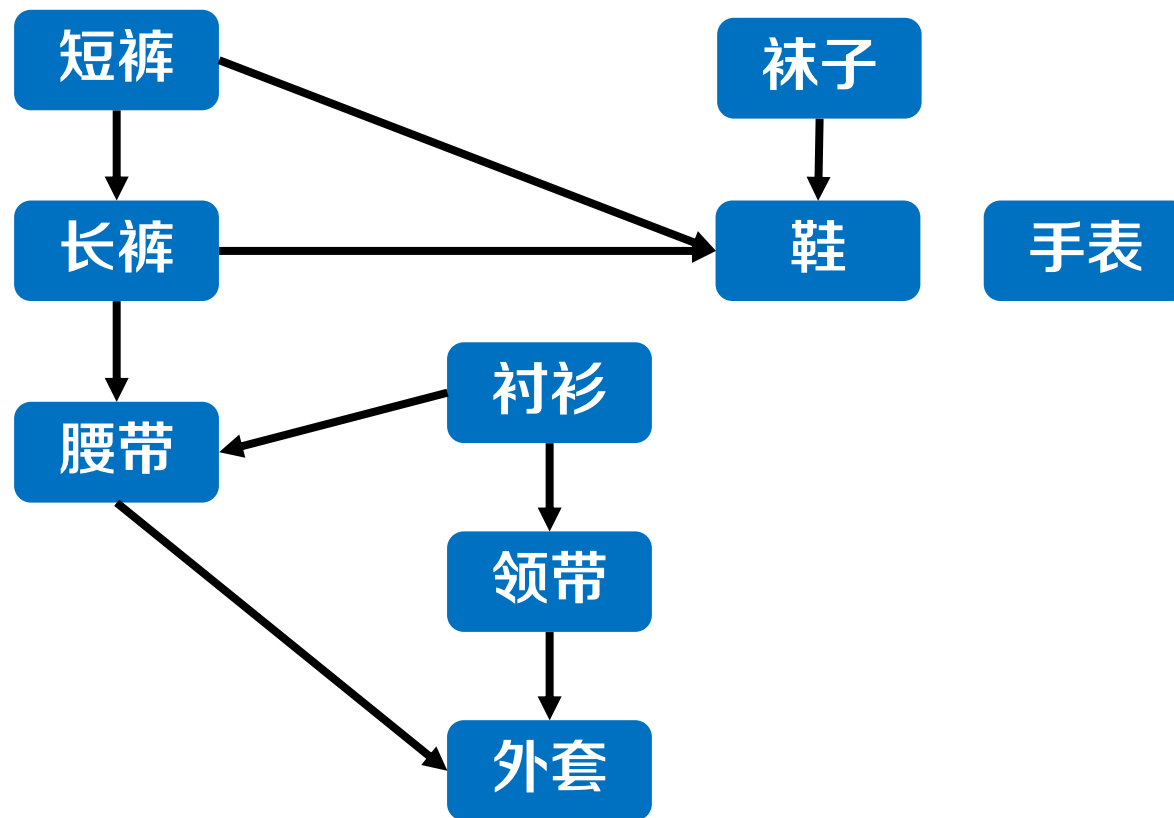
- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序
  - 完成时刻越早：按完成时刻逆序





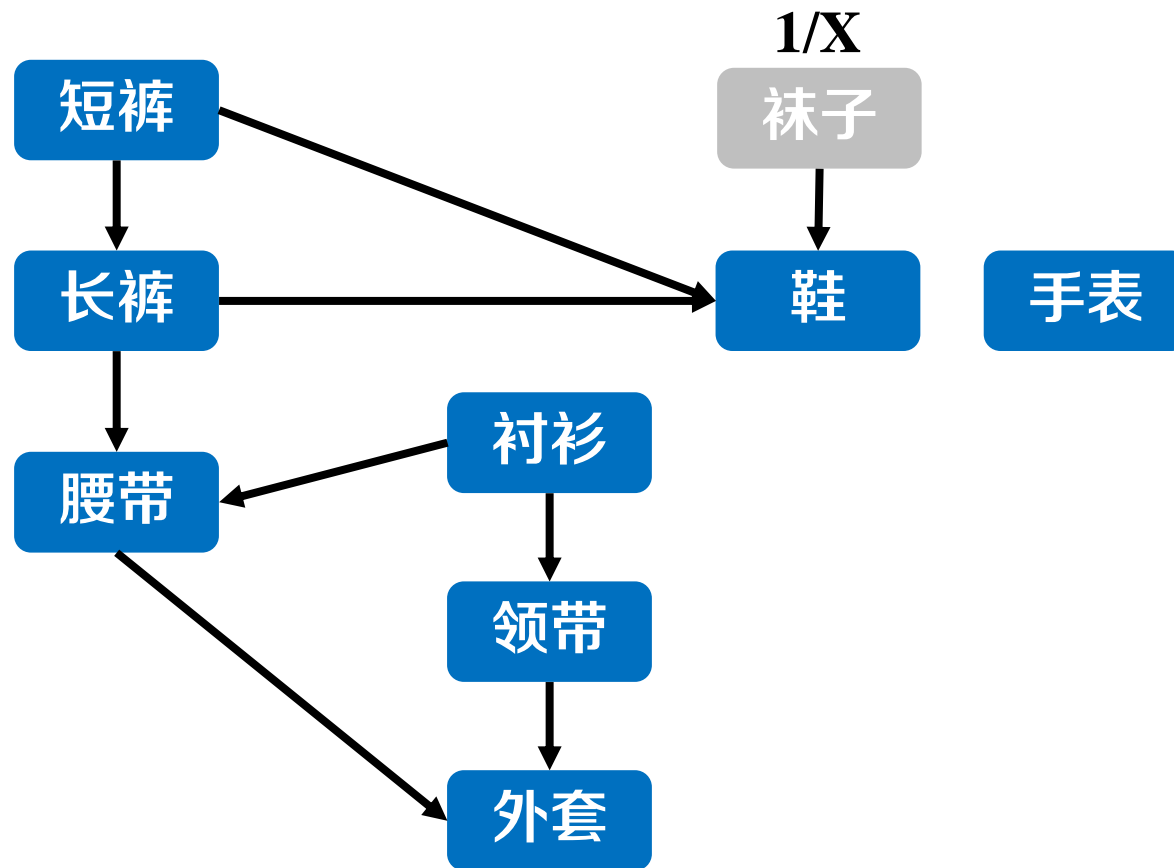
- 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序?
  - 完成时刻越早：按完成时刻逆序




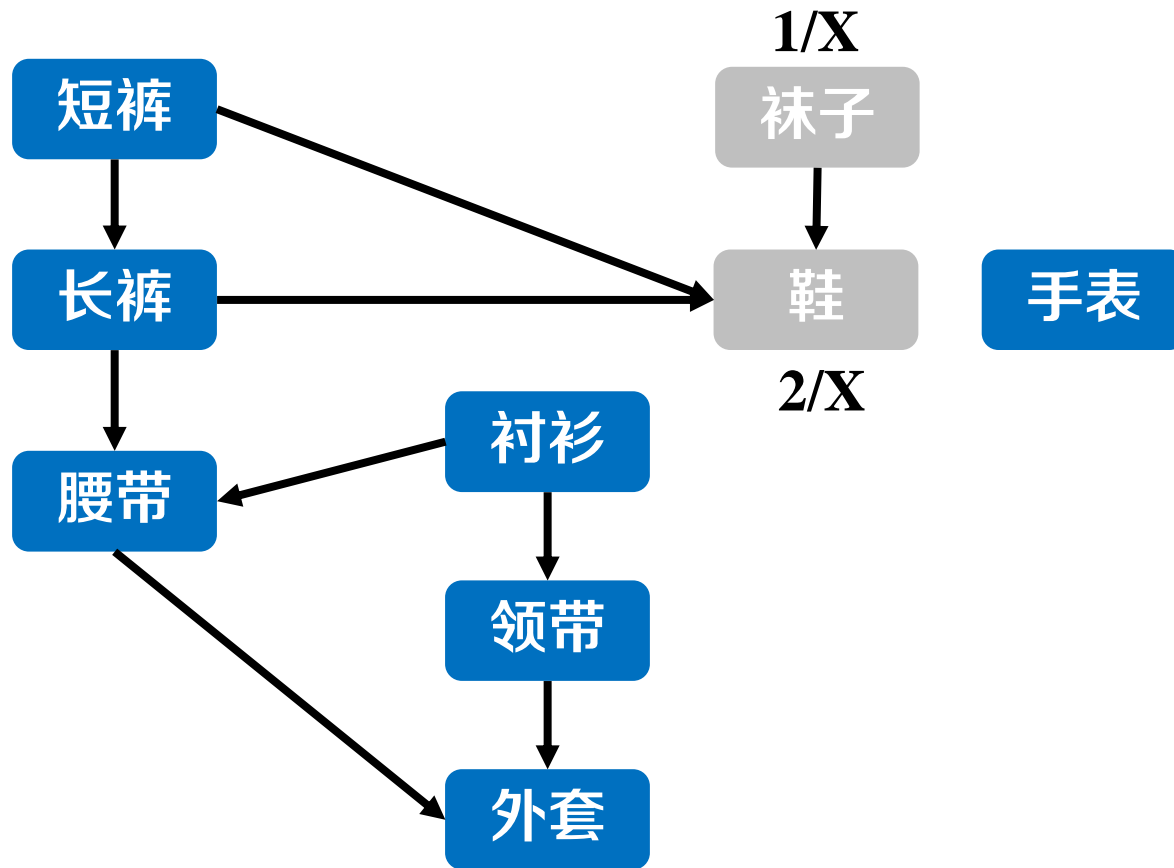
- 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序?
  - 完成时刻越早：按完成时刻逆序



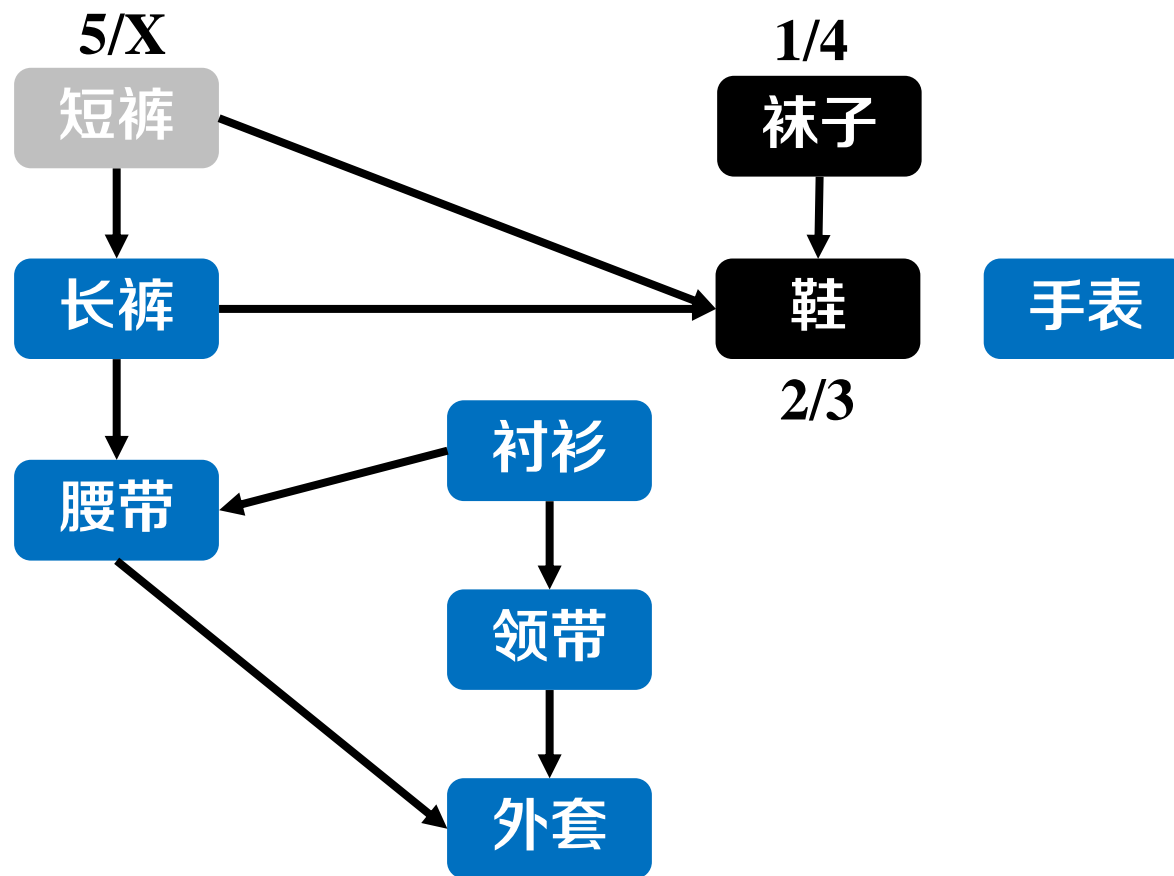
## ● 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
  - 深度越深
    - 发现时刻越晚：按发现时刻顺序?
    - 完成时刻越早：按完成时刻逆序
- 
- ```
graph TD; A[短裤] --> B[袜子]; A --> C[鞋子];
```



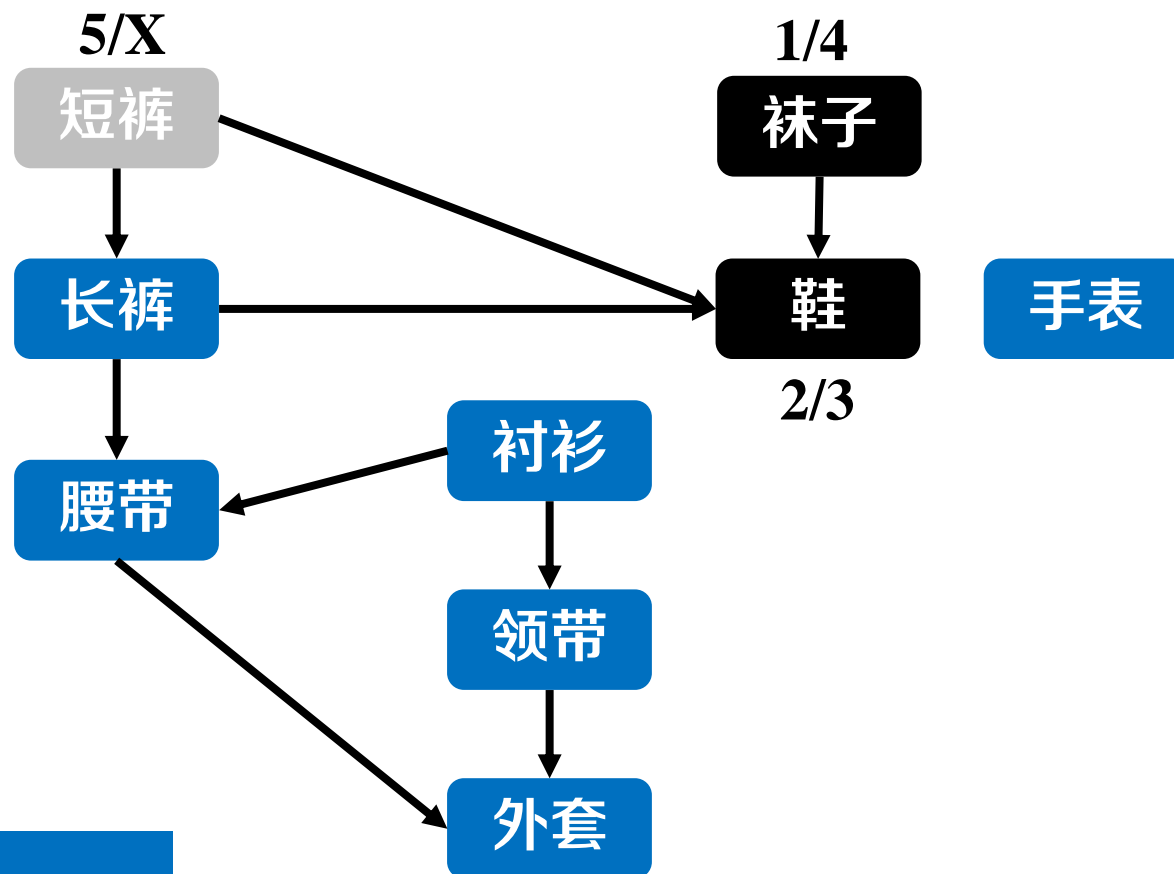
- 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序?
  - 完成时刻越早：按完成时刻逆序



## • 从DFS的视角观察

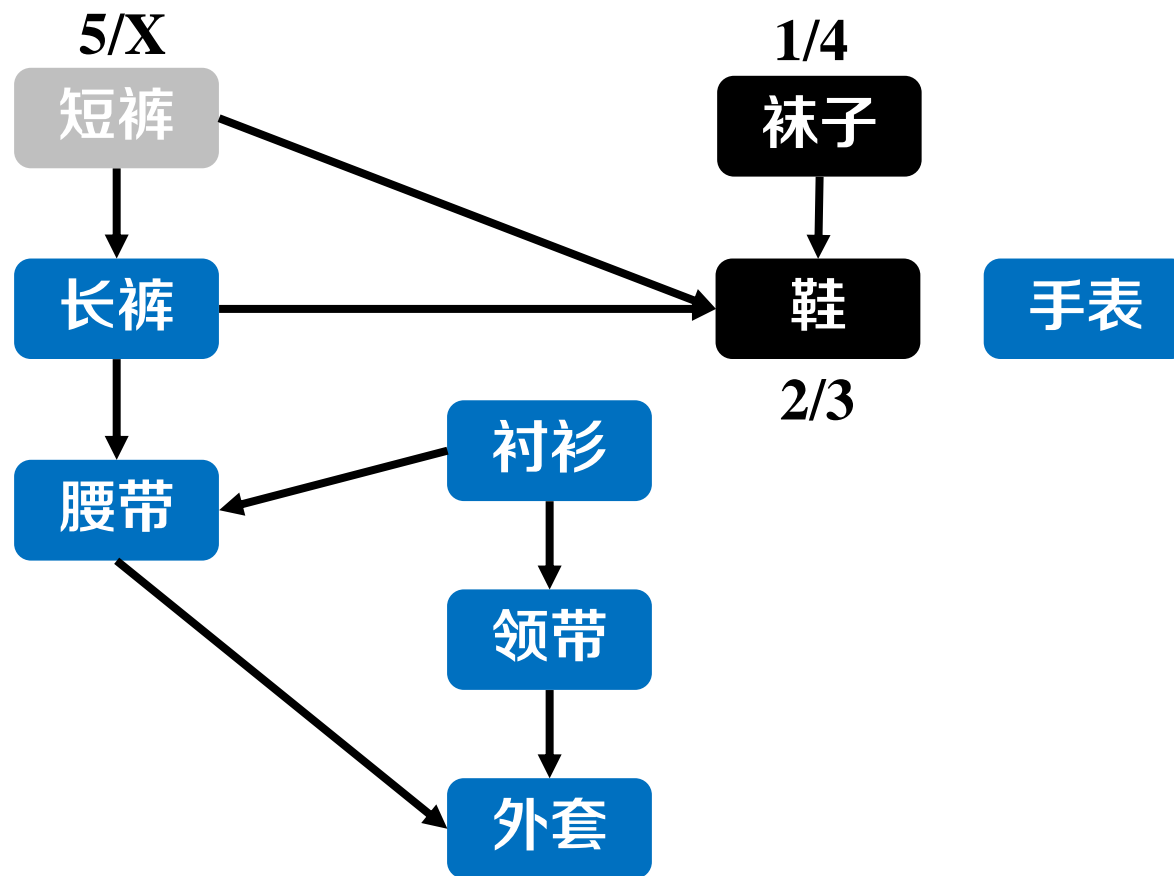
- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序



若按发现时刻顺序执行，会先穿鞋后穿短裤

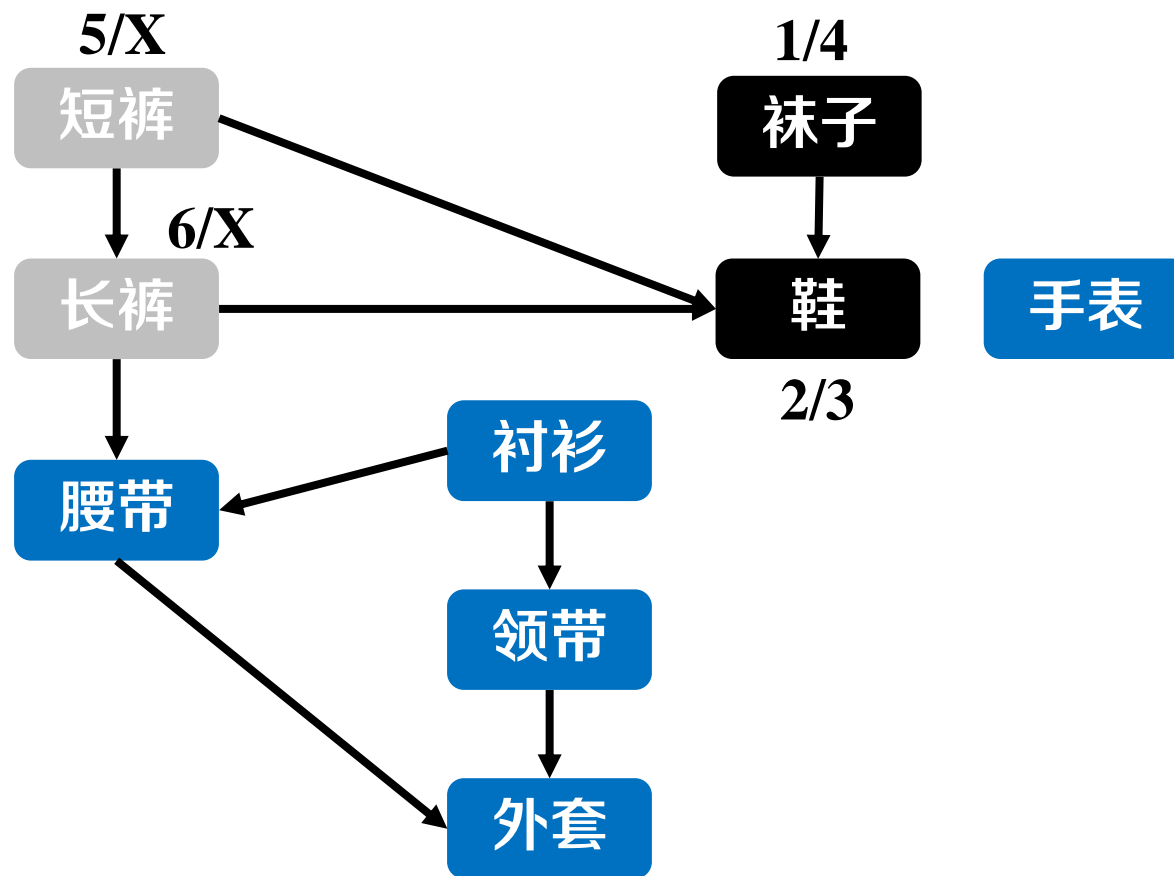
## • 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序？



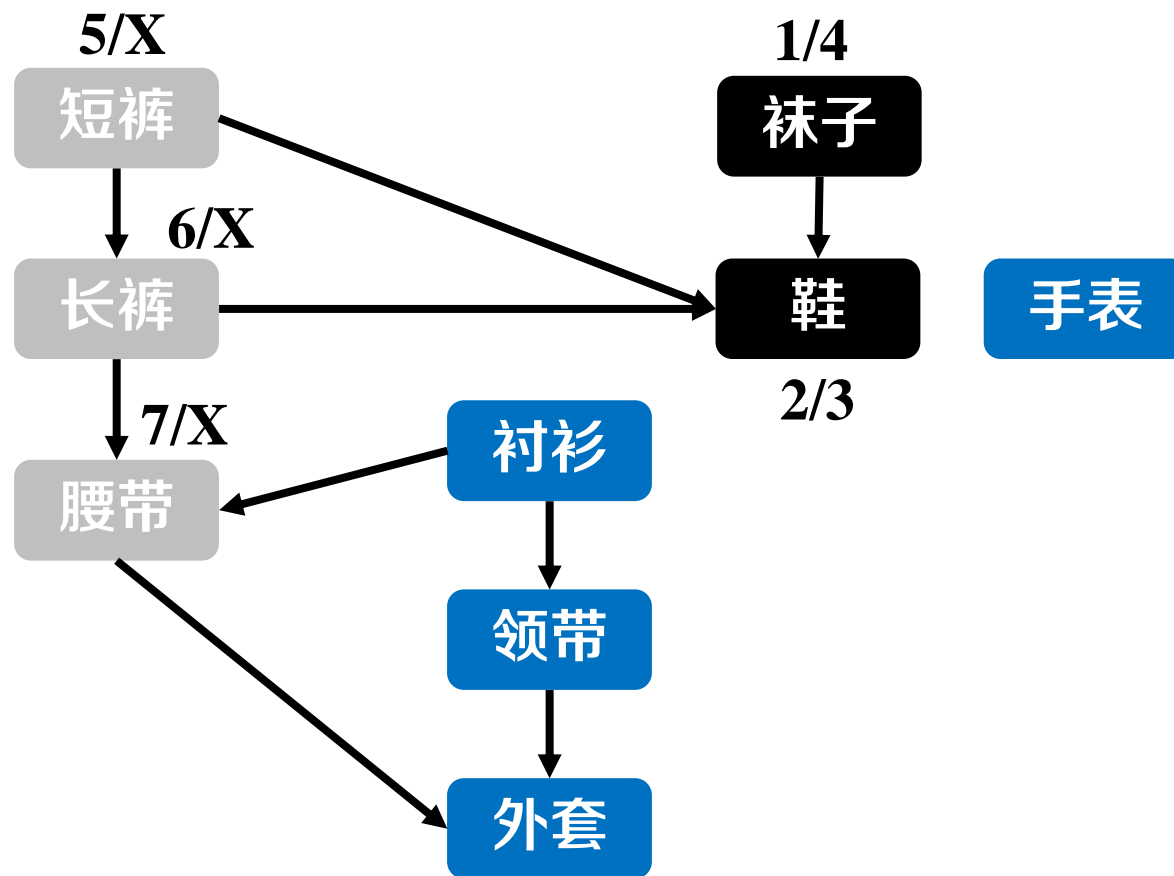
- 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序？



## • 从DFS的视角观察

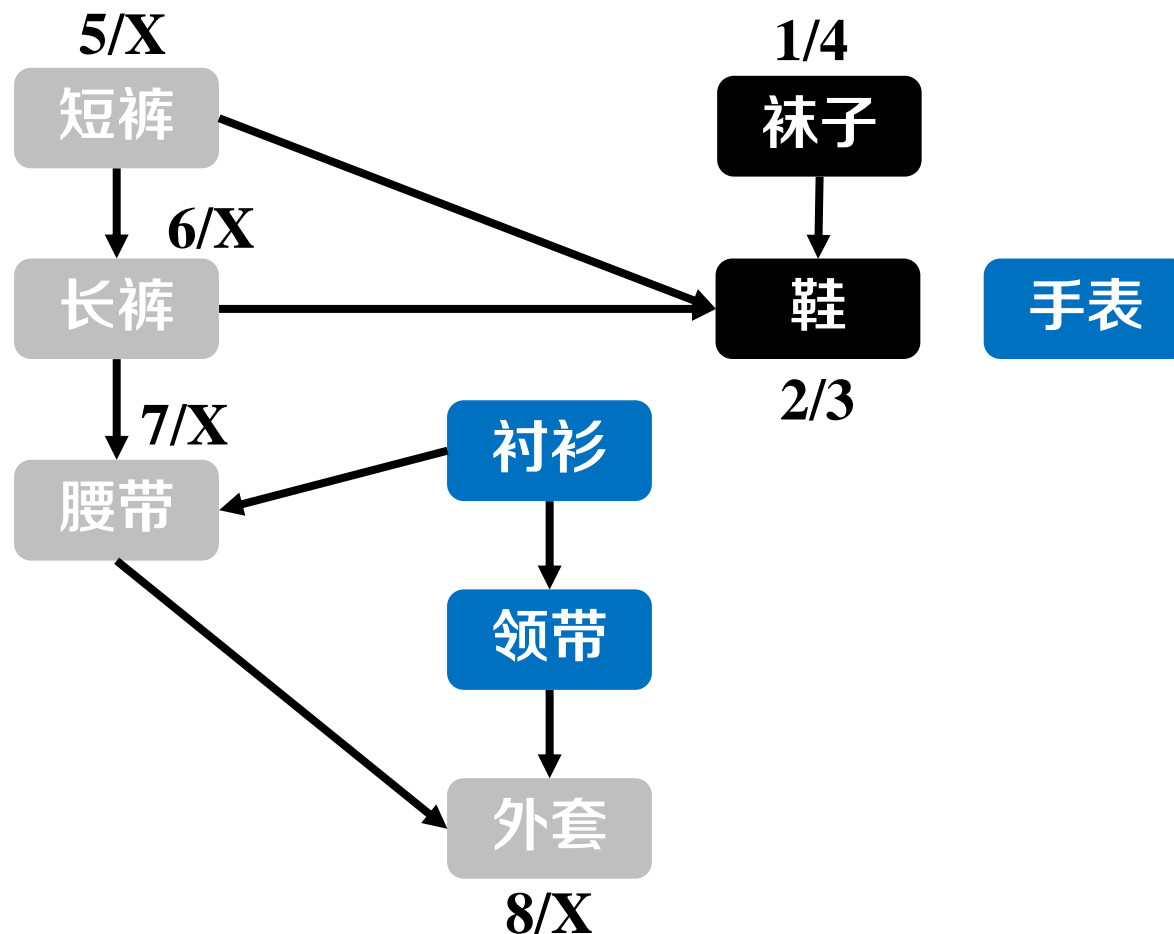
- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序？





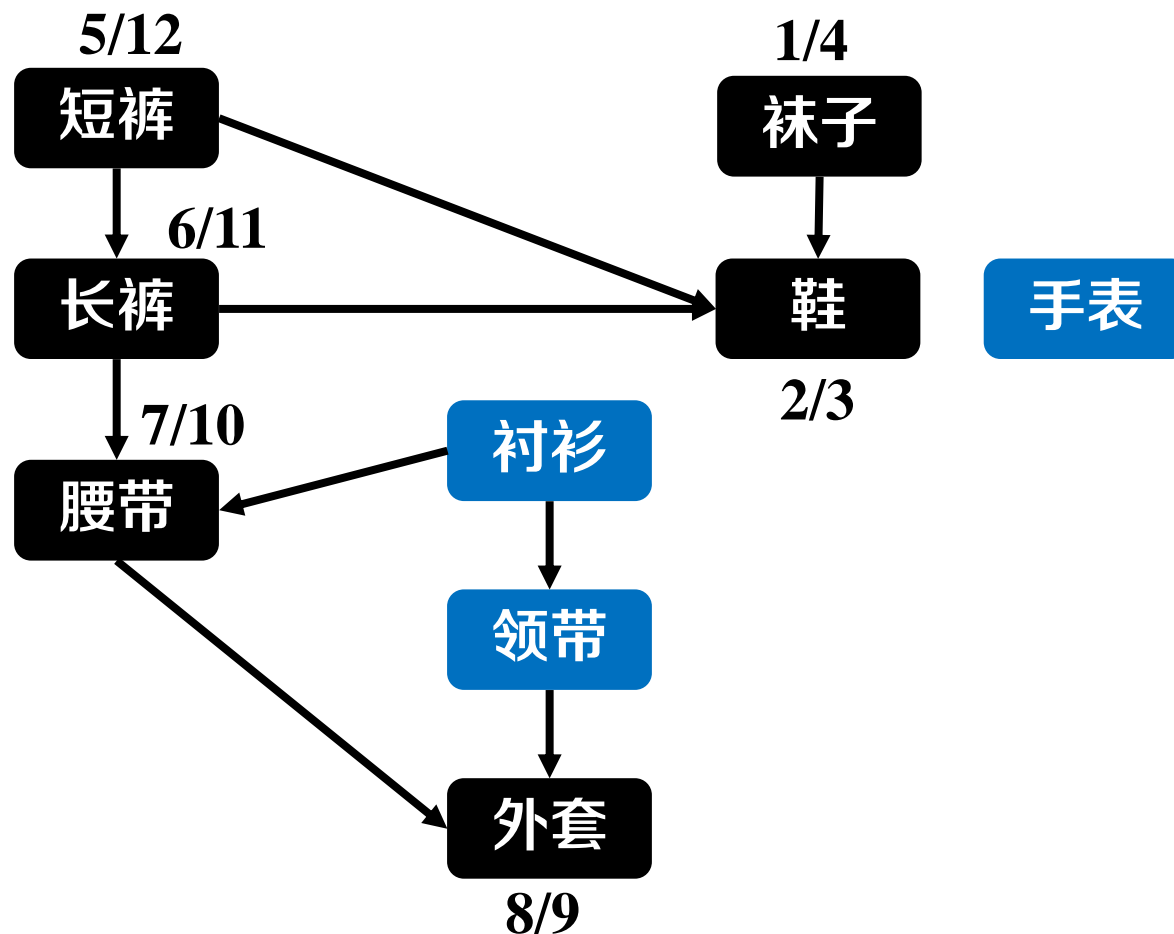
## • 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序？



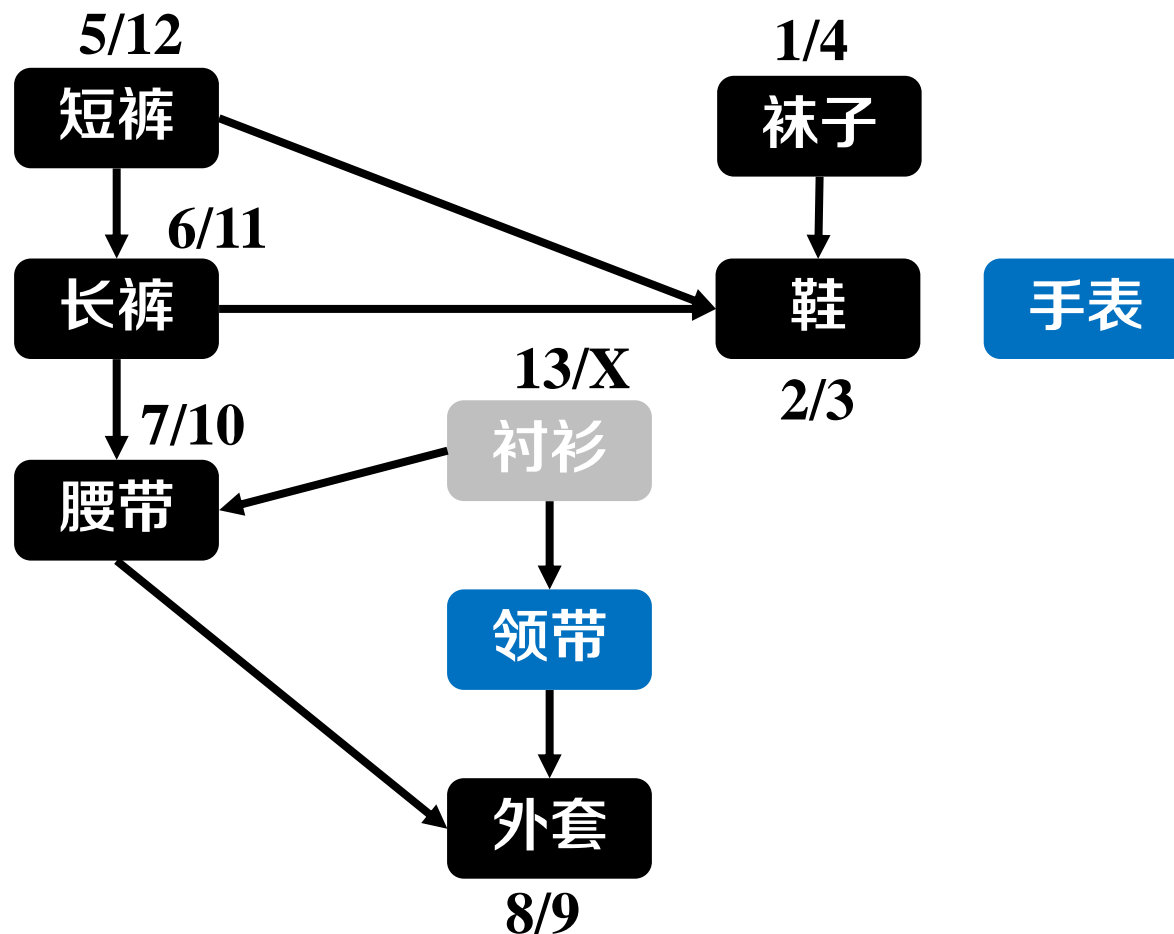
## • 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序？



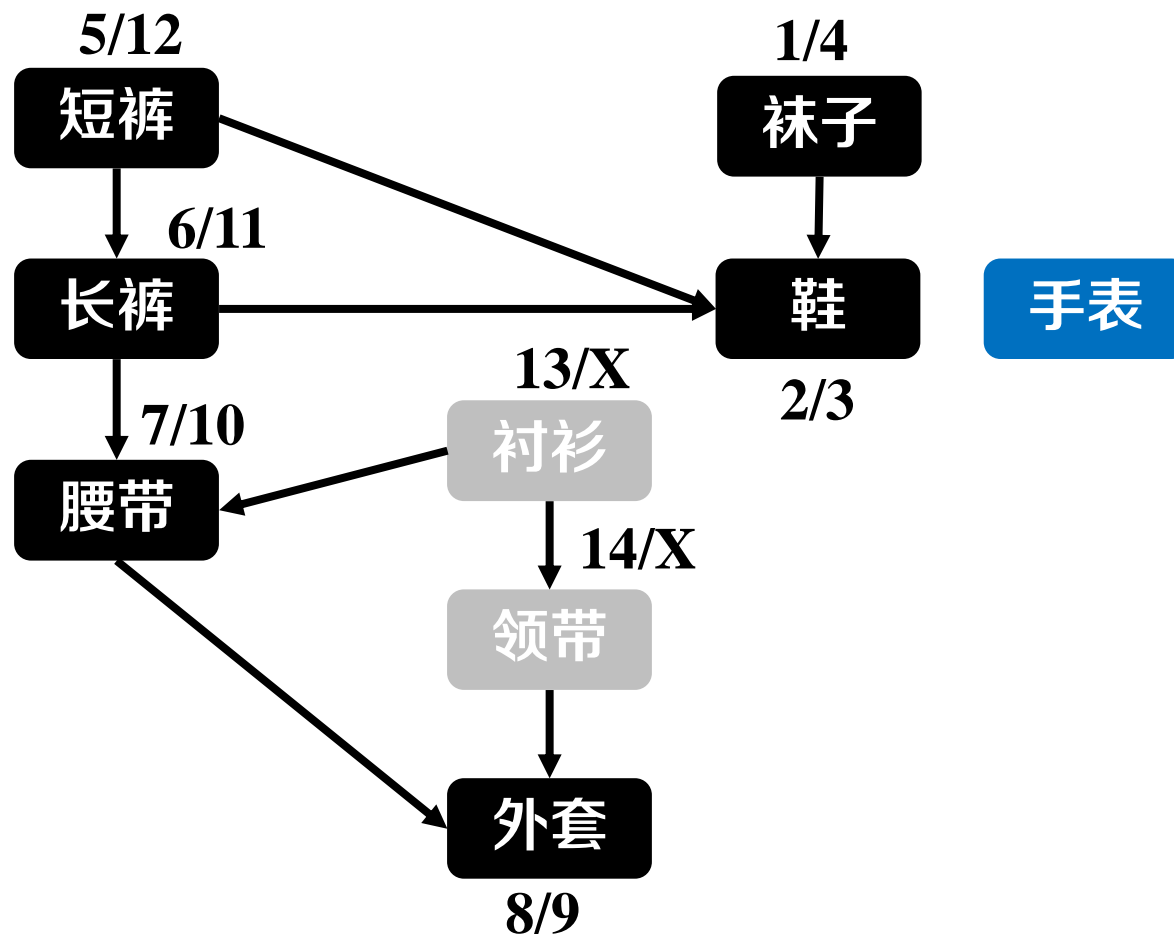
## • 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序？



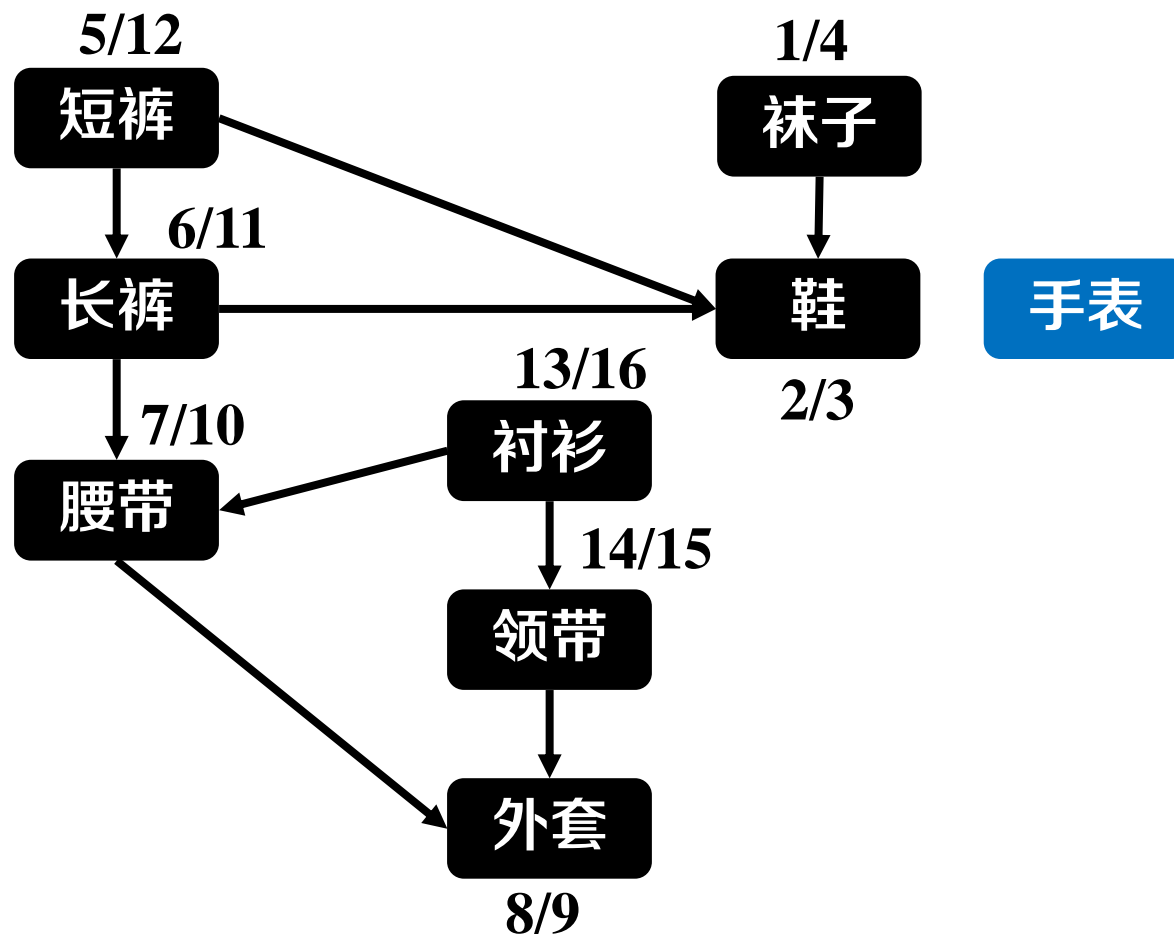
## • 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序？



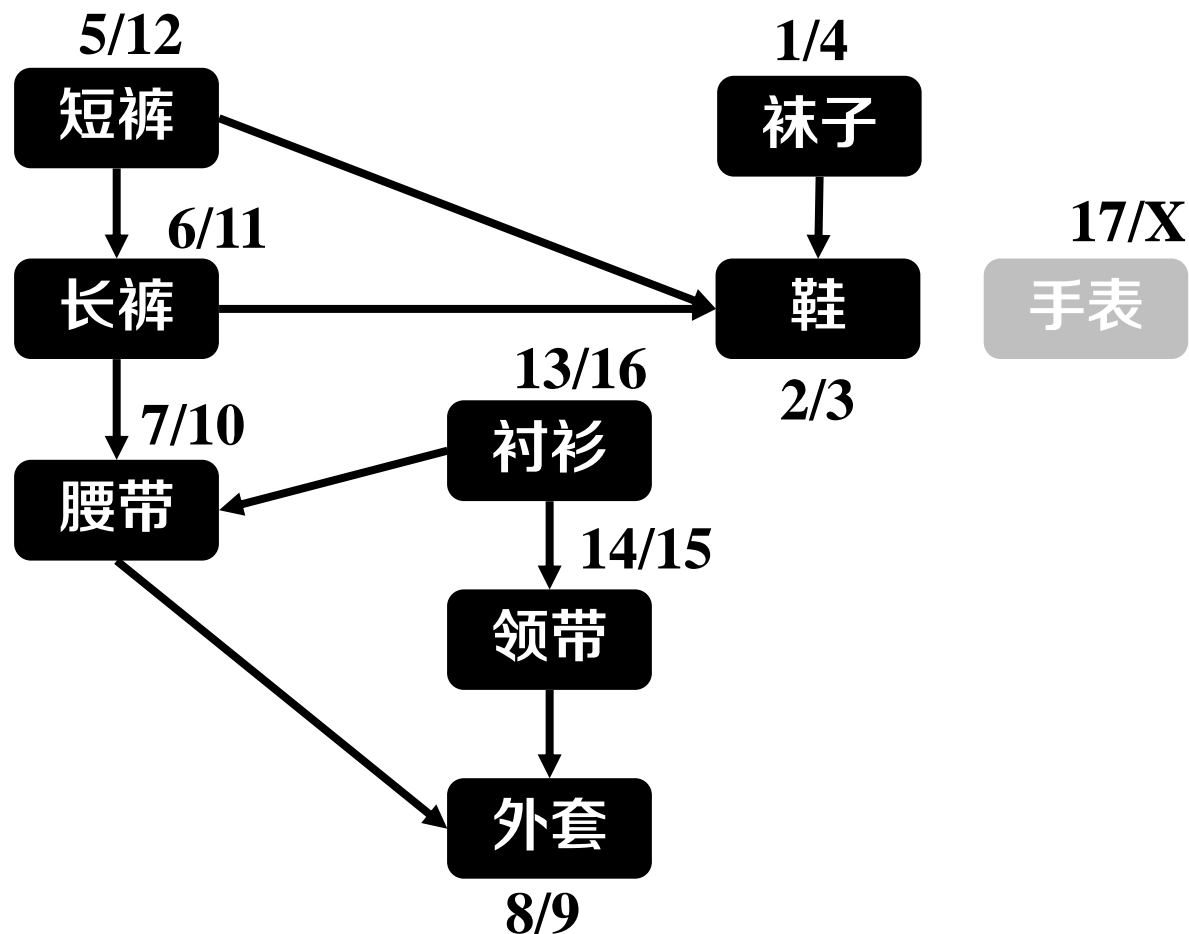
## • 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序？



## • 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序？

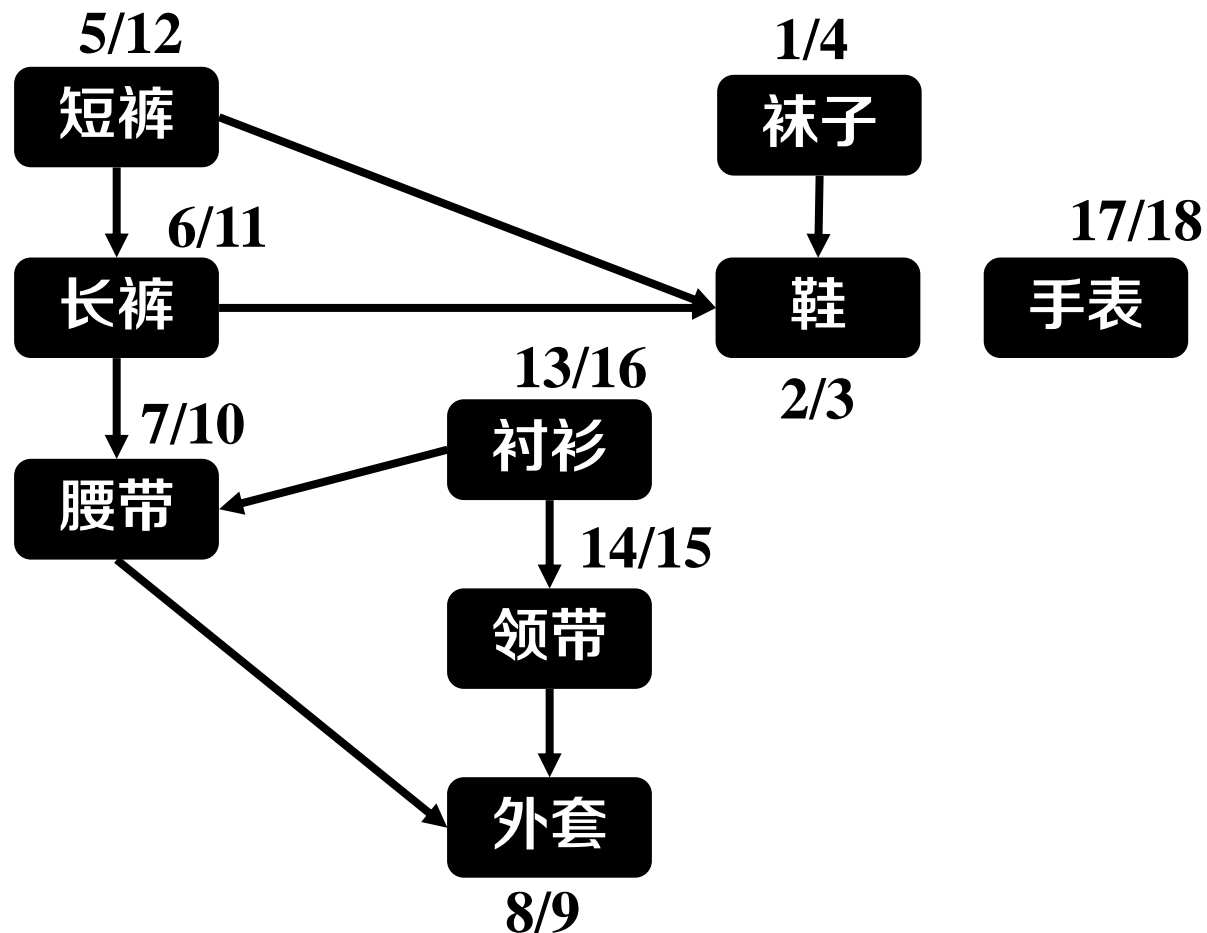


- 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后

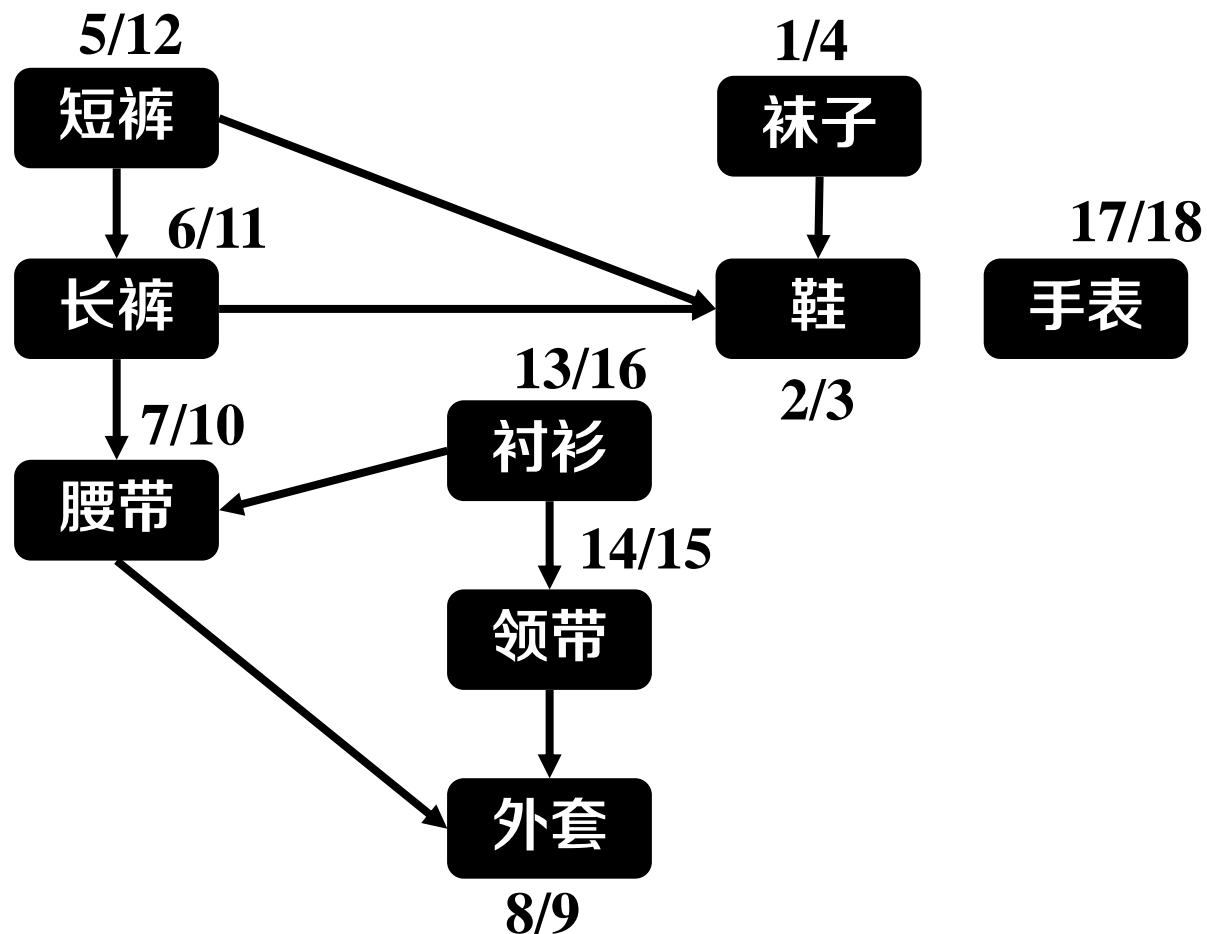
- 深度越深

- 发现时刻越晚：按发现时刻顺序~~✗~~
- 完成时刻越早：按完成时刻逆序？



## • 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序？



完成时刻逆序排列

手表

衬衫

领带

短裤

长裤

腰带

外套

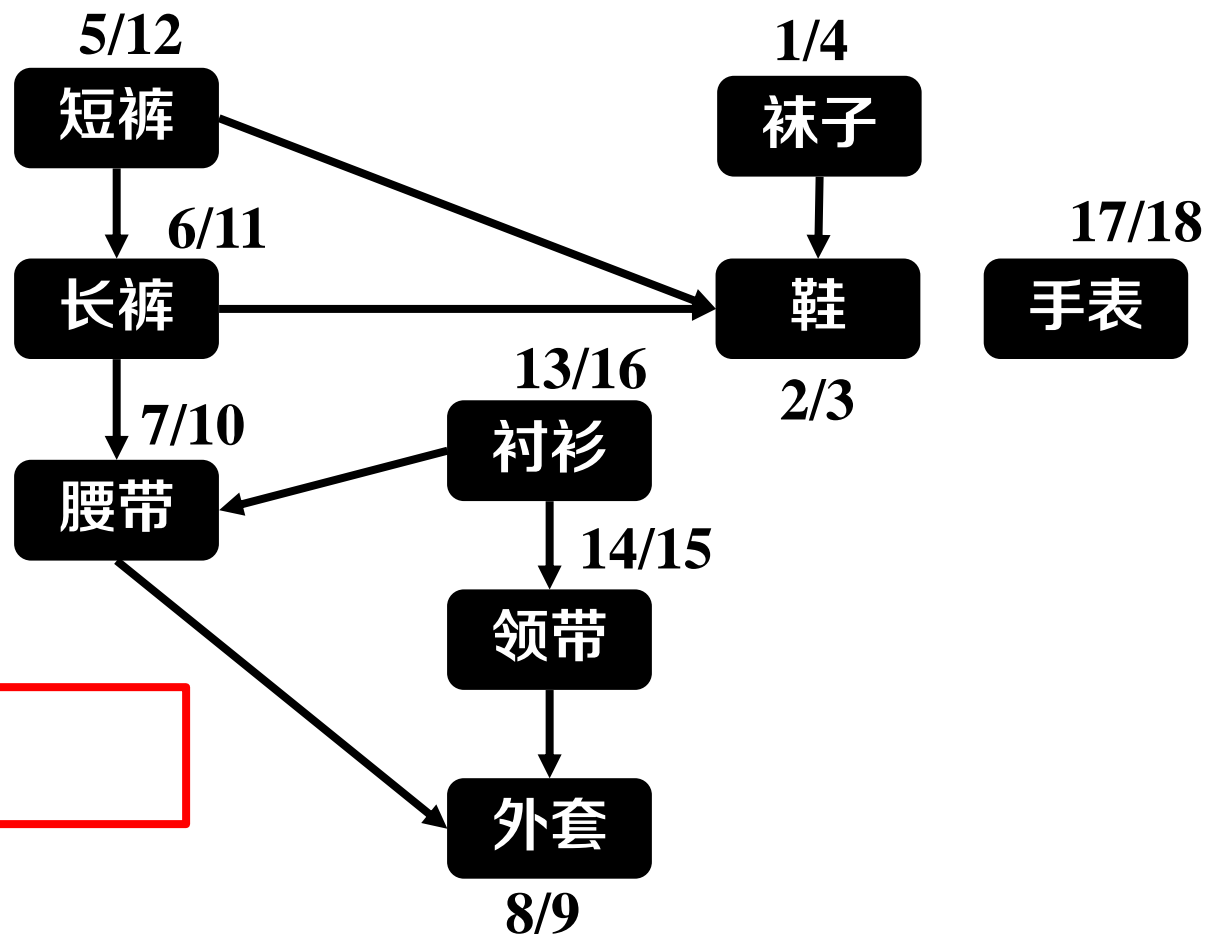
袜子

鞋



## • 从DFS的视角观察

- 穿衣顺序和搜索深度有关：深度越深，顺序越靠后
- 深度越深
  - 发现时刻越晚：按发现时刻顺序~~✗~~
  - 完成时刻越早：按完成时刻逆序？



按完成时刻逆序是否正确？

完成时刻逆序排列

手表

衬衫

领带

短裤

长裤

腰带

外套

袜子

鞋

问题定义

广度优先策略

深度优先策略

算法分析

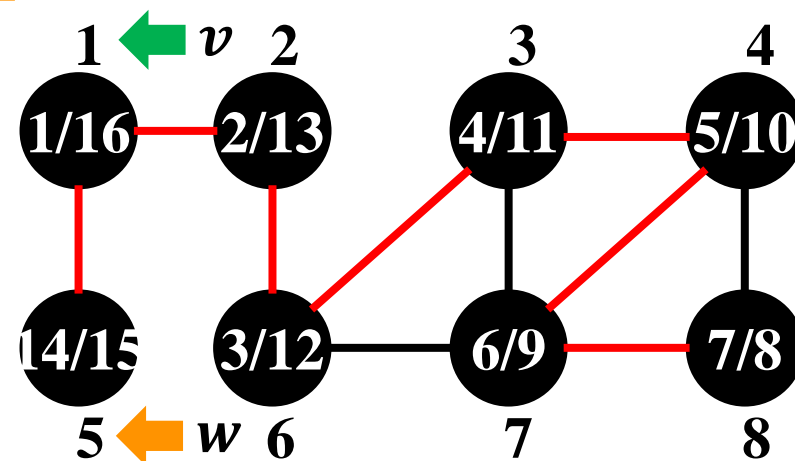
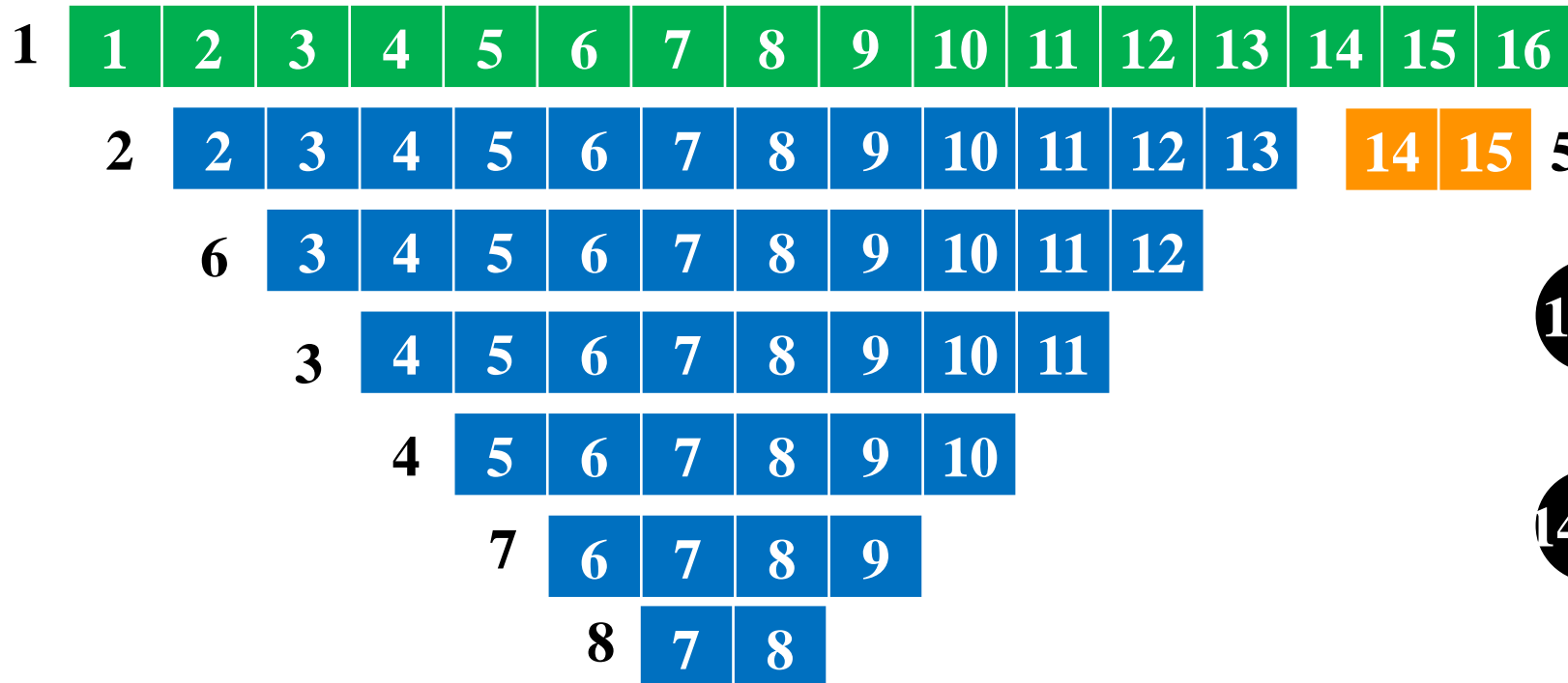
- 深度优先搜索确定的顺序：顶点完成时刻的逆序
- 拓扑序：对任意边 $(u, v)$ ， $u$ 在 $v$ 前面

- 已知深度优先搜索确定的顺序：顶点完成时刻的逆序
- 已知拓扑序：对任意边 $(u, v)$ ， $u$ 在 $v$ 前面
- 对任意边 $(u, v)$ ，完成时刻满足： $f(u) > f(v)$   $\longrightarrow$  算法正确

- 已知深度优先搜索确定的顺序：顶点完成时刻的逆序
- 已知拓扑序：对任意边 $(u, v)$ ， $u$ 在 $v$ 前面
- 对任意边 $(u, v)$ ，完成时刻满足： $f(u) > f(v)$   $\longrightarrow$  算法正确
  - 证明：设当前顶点为 $u$ ，搜索顶点 $v$

# 正确性证明

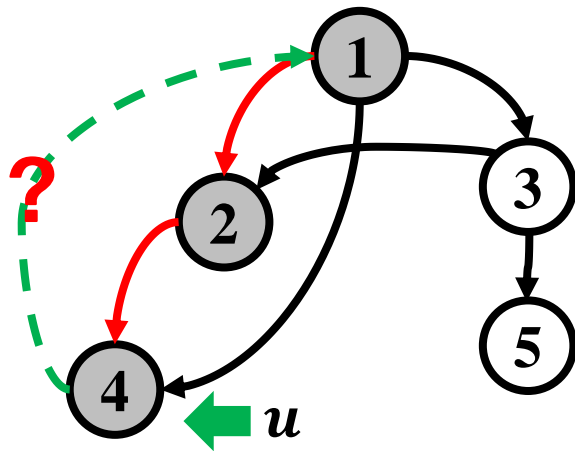
- 已知深度优先搜索确定的顺序：顶点完成时刻的逆序
- 已知拓扑序：对任意边 $(u, v)$ ， $u$ 在 $v$ 前面
- 对任意边 $(u, v)$ ，完成时刻满足： $f(u) > f(v)$   $\longrightarrow$  算法正确
  - 证明：设当前顶点为 $u$ ，搜索顶点 $v$ 
    - 若 $v$ 为白色， $v$ 是 $u$ 的后代， $f(u) > f(v)$ （括号化定理）



- 已知深度优先搜索确定的顺序：顶点完成时刻的逆序
- 已知拓扑序：对任意边 $(u, v)$ ， $u$ 在 $v$ 前面
- 对任意边 $(u, v)$ ，完成时刻满足： $f(u) > f(v)$   $\longrightarrow$  算法正确
  - 证明：设当前顶点为 $u$ ，搜索顶点 $v$ 
    - 若 $v$ 为白色， $v$ 是 $u$ 的后代， $f(u) > f(v)$ （括号化定理）
    - 若 $v$ 为黑色， $v$ 已经完成， $u$ 尚未完成， $f(u) > f(v)$

# 正确性证明

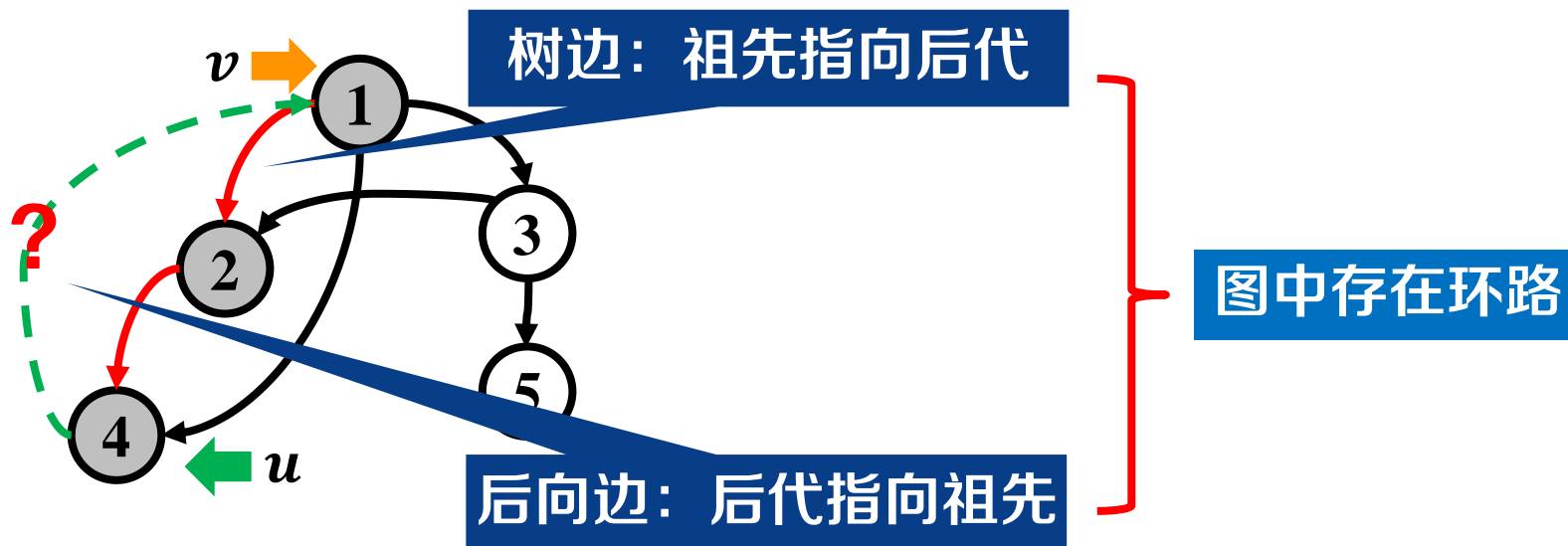
- 已知深度优先搜索确定的顺序：顶点完成时刻的逆序
- 已知拓扑序：对任意边 $(u, v)$ ， $u$ 在 $v$ 前面
- 对任意边 $(u, v)$ ，完成时刻满足： $f(u) > f(v)$   $\longrightarrow$  算法正确
  - 证明：设当前顶点为 $u$ ，搜索顶点 $v$ 
    - 若 $v$ 为白色， $v$ 是 $u$ 的后代， $f(u) > f(v)$ （括号化定理）
    - 若 $v$ 为黑色， $v$ 已经完成， $u$ 尚未完成， $f(u) > f(v)$
    - 若 $v$ 是灰色？






# 正确性证明

- 已知深度优先搜索确定的顺序：顶点完成时刻的逆序
- 已知拓扑序：对任意边 $(u, v)$ ， $u$ 在 $v$ 前面
- 对任意边 $(u, v)$ ，完成时刻满足： $f(u) > f(v)$   $\longrightarrow$  算法正确
  - 证明：设当前顶点为 $u$ ，搜索顶点 $v$ 
    - 若 $v$ 为白色， $v$ 是 $u$ 的后代， $f(u) > f(v)$ （括号化定理）
    - 若 $v$ 为黑色， $v$ 已经完成， $u$ 尚未完成， $f(u) > f(v)$
    - 若 $v$ 是灰色？不可能！因为有向无环图不存在后向边



- 已知深度优先搜索确定的顺序：顶点完成时刻的逆序
- 已知拓扑序：对任意边 $(u, v)$ ， $u$ 在 $v$ 前面
- 对任意边 $(u, v)$ ，完成时刻满足： $f(u) > f(v)$   算法正确
  - 证明：设当前顶点为 $u$ ，搜索顶点 $v$ 
    - 若 $v$ 为白色， $v$ 是 $u$ 的后代， $f(u) > f(v)$ （括号化定理）
    - 若 $v$ 为黑色， $v$ 已经完成， $u$ 尚未完成， $f(u) > f(v)$
    - 若 $v$ 是灰色？不可能！因为有向无环图不存在后向边

- **Topological-Sort-DFS( $G$ )**

输入: 图 $G$

输出: 顶点拓扑序

$L \leftarrow DFS(G)$

**[ return  $L.reverse()$  ]**

数组中元素逆序排列

- **Topological-Sort-DFS( $G$ )**

输入: 图 $G$

输出: 顶点拓扑序

$L \leftarrow DFS(G)$

return  $L.reverse()$

问题: 如何在搜索过程中得到按完成时刻顺序排列的顶点?

- DFS( $G$ )

```
输入: 图  $G$   
新建数组  $color[1..V], L[1..V]$   
for  $v \in V$  do  
    |  $color[v] \leftarrow WHITE$   
end  
for  $v \in V$  do  
    | if  $color[v] = WHITE$  then  
    |     |  $L' \leftarrow \text{DFS-Visit}(G, v)$   
    |     | 向  $L$  结尾追加  $L'$   
    |     end  
    end  
end  
return  $L$ 
```

- DFS-Visit( $G, v$ )

```
输入: 图  $G$ , 顶点  $v$   
输出: 按完成时刻从早到晚排列的顶点  $L$   
 $color[v] \leftarrow GRAY$   
for  $w \in G.Adj[v]$  do  
    | if  $color[w] = WHITE$  then  
    |     |  $L \leftarrow \text{DFS-Visit}(G, w)$   
    |     end  
    end  
end  
 $color[v] \leftarrow BLACK$   
向  $L$  结尾追加顶点  $v$   
return  $L$ 
```

顶点按  
完成时  
刻排列

- **Topological-Sort-DFS( $G$ )**

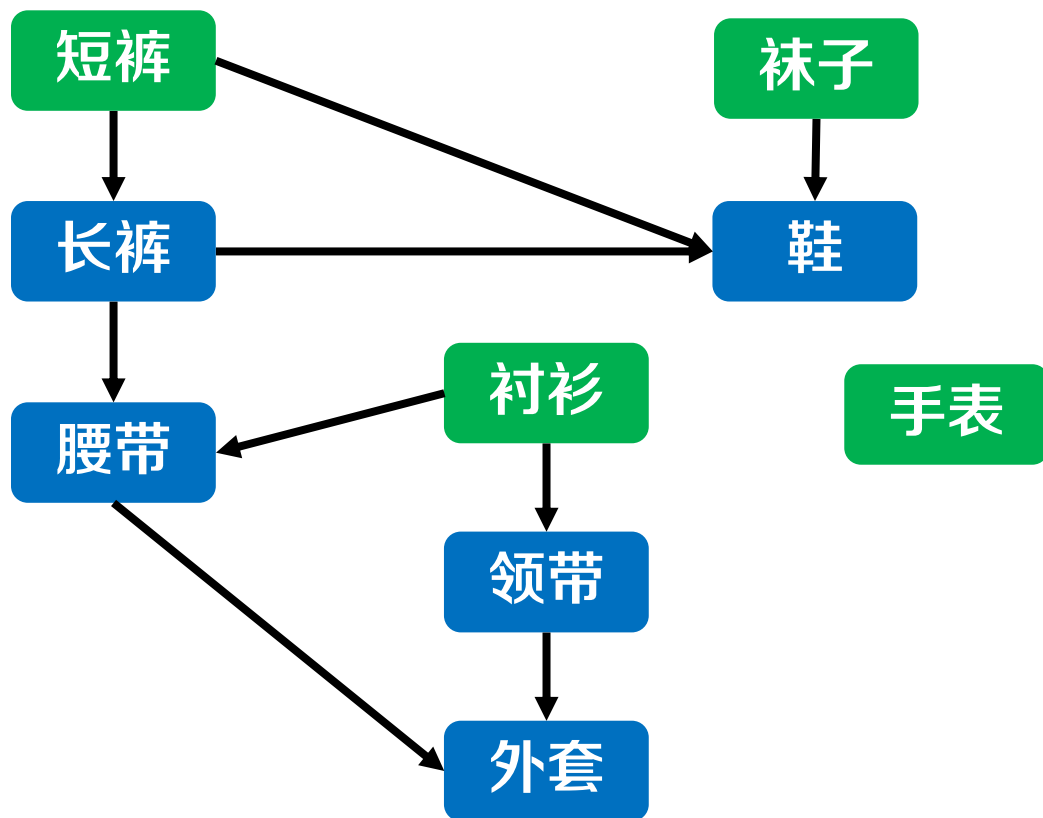
输入: 图 $G$

输出: 顶点拓扑序

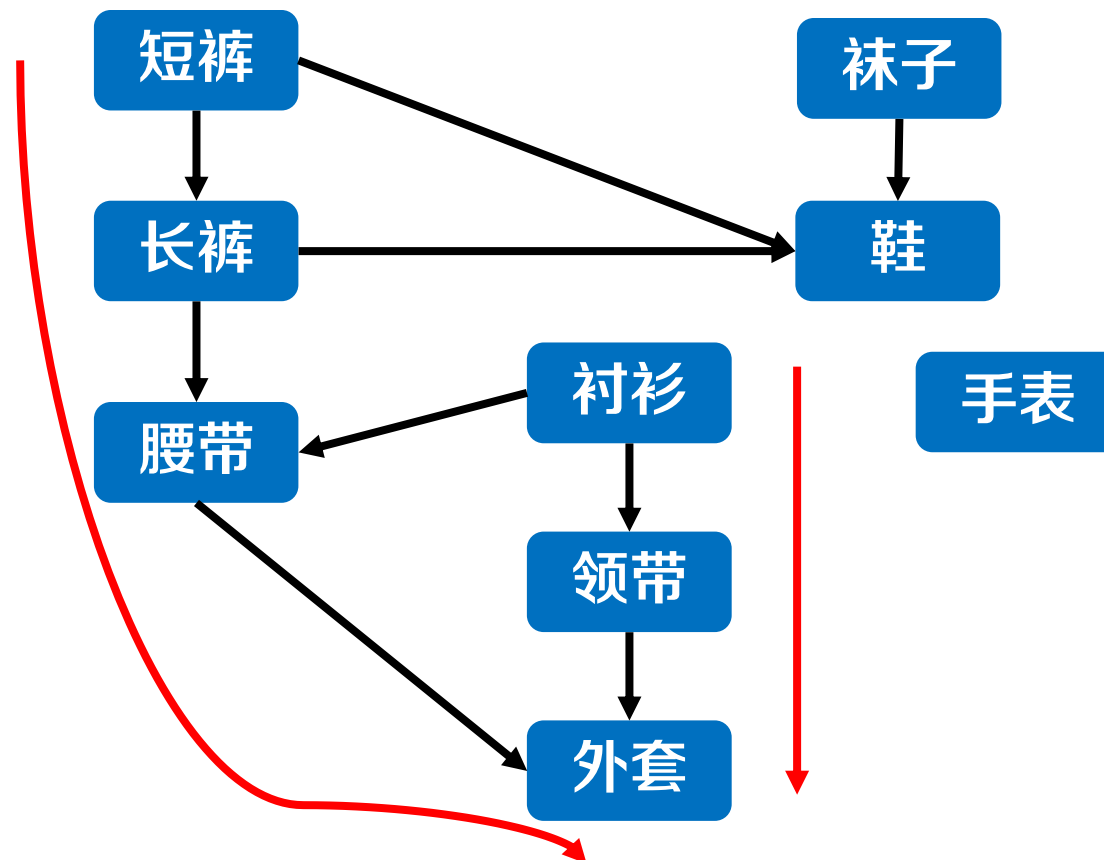
$L \leftarrow DFS(G)$

return  $L.reverse()$

时间复杂度:  $O(|V| + |E|)$



**广度优先**  
顺序思想：把容易完成的事优先完成



**深度优先**  
逆序思想：把不易完成的事放到后面

# 谢谢

