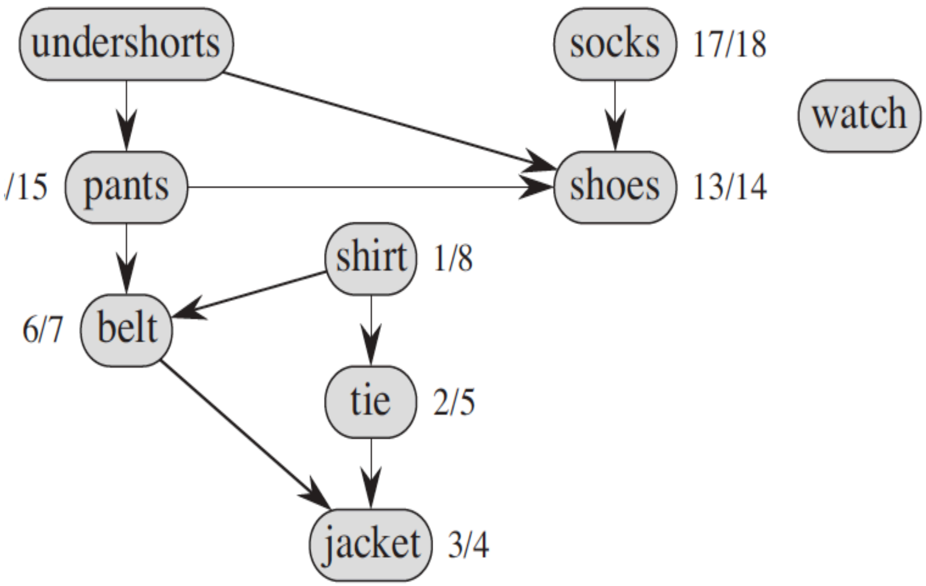
Topological Sort

Given directed acyclic graph (DAG) G=(V,E). List the vertices in V such that for each edge (i,j) in E, vertex i is listed before vertex j



repeat

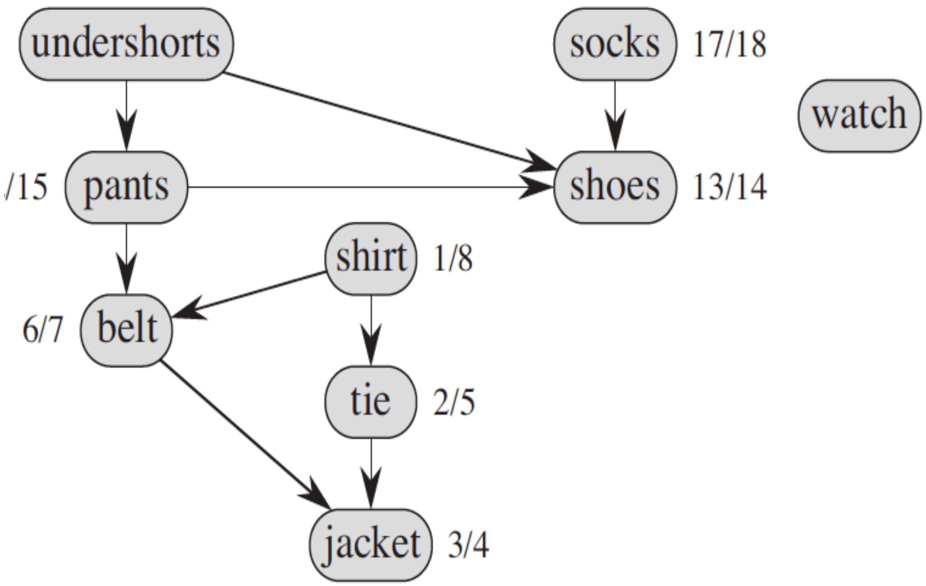
- Write out a vertex with no incoming edges
- Remove this vertex and all its edges

until done

- Need not be unique…
- Also a cycle-detection algorithm
- Indegree of a vertex: number of incoming edges

Topological Sort

Given directed acyclic graph (DAG) G=(V,E). List the vertices in V such that for each edge (i,j) in E, vertex i is listed before vertex j



Compute the indegrees of all vertices

repeat

- Write out a vertex with indegree zero
- Decrement indegrees of all vertices to which this vertex has edges

until done

- Need not be unique…
- Also a cycle-detection algorithm
- Indegree of a vertex: number of incoming edges

Topological Sort -- pseudocode

```
for (int k=0; k<|V|; k++)
             indegree[k] = 0;

for each vertex i
     for each edge (i,j)
            indegree[j] += 1;

for (int k=0; k<|V|; k++) {
     let i be a vertex with (indegree[i]==0); print i;
     if no such node, return failure; // cycle !!
     for each edge (i,j)
            indegree[j] -= 1;
 }
```

Analysis – adjacency matrix

```
for (int k=0; k<|V|; k++)
            indegree[k] = 0;

for each vertex i
     for each edge (i,j)
           indegree[j] += 1;

for (int k=0; k<|V|; k++) {
     let i be a vertex with (indegree[i]==0); print i;
     if no such node, return failure; // cycle !!
     for each edge (i,j)
           indegree[j] -= 1;
 }
```

$O(|V|^2)$ to compute the indegrees

In each iteration:

- $O(|V|)$ to find i
- $O(|V|)$ to update indegrees

Total run-time: $O(|V|^2) + O(|V|^2) = O(|V|^2)$

Analysis – adjacency list

$O(|V| + |E|)$ to compute the indegrees

In each iteration:

  $O(|V|)$ to find i

  $O(|E|)$ across all iterations to update indegrees

Total run-time: $O(|V|+|E|) + O(|V|^2) + O(|E|) = O(|V|^2)$

```
for (int k=0; k<|V|; k++)
            indegree[k] = 0;

for each vertex i
     for each edge (i,j)
           indegree[j] += 1;

for (int k=0; k<|V|; k++) {
     let i be a vertex with (indegree[i]==0); print i;
     if no such node, return failure; // cycle !!
     for each edge (i,j)
           indegree[j] -= 1;
 }
```

Analysis – adjacency list [Improved implementation]

Place zero-indegree vertices in a bag

- O(1)-time insert and remove
- E.g., a queue

```
for (int k=0; k<|V|; k++)
          indegree[k] = 0;

for each vertex i
    for each edge (i,j)
          indegree[j] += 1;
for (int k=0; k<|V|; k++) if (indegree[k]==0) Q.enqueue(k);
for (int k=0; k<|V|; k++) {
    if (Q.isEmpty()) cycle detected; else i = Q.dequeue();

    for each edge (i,j)
          indegree[j] -= 1;
 }        if (indegree[j] == 0) Q.enqueue(j);
```

Analysis – adjacency list [Improved implementation]

O(|V| + |E|) to compute the indegrees and initialize Q

In each iteration:

O(|1|) to find i

O(|E|) across all iterations to update indegrees and enqueue vertices

Total run-time: O(|V|+|E|) + O(|V|) + O(|E|) = O(|V|+|E|)

Linear – optimal

```
for (int k=0; k<|V|; k++)
              indegree[k] = 0;

for each vertex i
     for each edge (i,j)
             indegree[j] += 1;
for (int k=0; k<|V|; k++) if (indegree[k]==0) Q.enqueue(k);

for (int k=0; k<|V|; k++) {
     if (Q.isEmpty()) cycle detected; else i = Q.dequeue();
     for each edge (i,j){
             indegree[j] -= 1;
             if (indegree[j] == 0) Q.enqueue(j);
    }
 }
```