MORE GRAPHS

David Kauchak
CS 140 – Fall 2022

1

Admin

Checkpoint

Assignment 7

Assignment 8

2

Heaps

What's an abstract data type?

How can we implement a heap?

- Build-Heap
- Extract-Max
- Insert

3

Proofs on trees

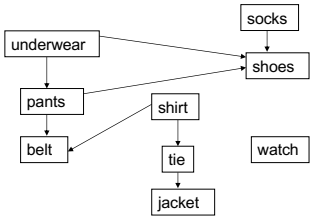
4

Hotels!

5

DAGs

Can represent dependency graphs

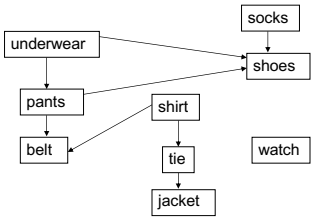


6

Topological sort

A linear ordering of all the vertices such that for all edges (u,v) ∈ E, u appears before v in the ordering

An ordering of the nodes that "obeys" the dependencies, i.e. an activity can't happen until it's dependent activities have happened



watch
underwear
pants
shirt
belt
tie
socks
shoes
jacket

7

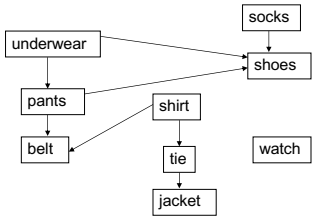
Topological sort

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)

8

Topological sort

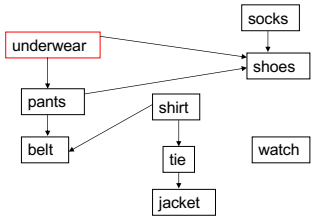
TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)



9

Topological sort

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)

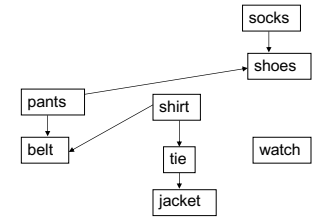


10

Topological sort

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)

underwear

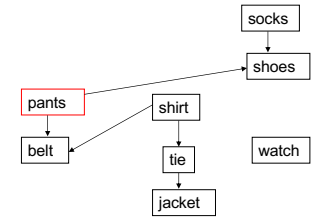


11

Topological sort

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)

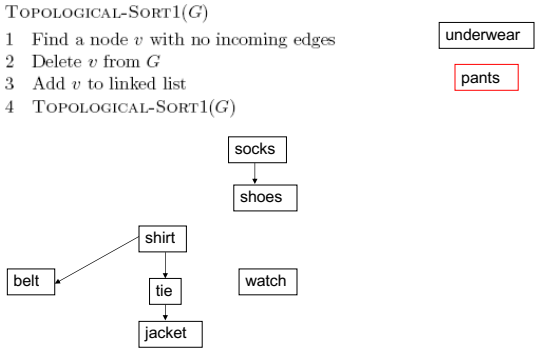
underwear



12

Topological sort

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)



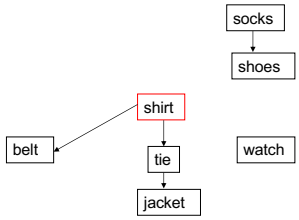
underwear

pants

13

Topological sort

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)



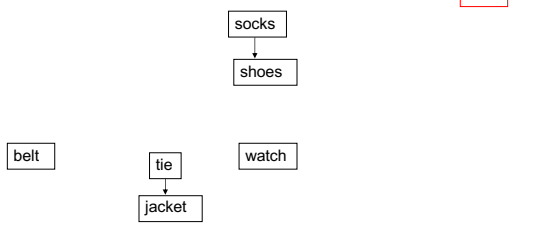
underwear

pants

14

Topological sort

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)



underwear

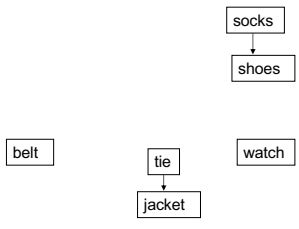
pants

shirt

15

Topological sort

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)



underwear

pants

shirt

...

16

Running time?

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)

17

Running time?

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)

O(|V|+|E|)

18

Running time?

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)

O(E) overall

19

Running time?

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)

How many calls? |V|

20

Running time?

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)

Overall running time?

$O(|V|^2+|V|\ |E|)$

21

Can we do better?

TOPOLOGICAL-SORT1(G)
1 Find a node v with no incoming edges
2 Delete v from G
3 Add v to linked list
4 TOPOLOGICAL-SORT1(G)

22

Topological sort 2

TOPOLOGICAL-SORT2(G)
1 **for** all edges $(u, v) \in E$
2 $\quad active[v] \leftarrow active[v] + 1$
3 **for** all $v \in V$
4 $\quad$ **if** $active[v] = 0$
5 $\quad\quad$ ENQUEUE(S, v)
6 **while** !EMPTY(S)
7 $\quad u \leftarrow$ DEQUEUE(S)
8 $\quad$ add u to linked list
9 $\quad$ **for** each edge $(u, v) \in E$
10 $\quad\quad active[v] \leftarrow active[v] - 1$
11 $\quad\quad$ **if** $active[v] = 0$
12 $\quad\quad\quad$ ENQUEUE(S, v)

23

Topological sort 2

TOPOLOGICAL-SORT2(G)
1 **for** all edges $(u, v) \in E$
2 $\quad active[v] \leftarrow active[v] + 1$
3 **for** all $v \in V$
4 $\quad$ **if** $active[v] = 0$
5 $\quad\quad$ ENQUEUE(S, v)
6 **while** !EMPTY(S)
7 $\quad u \leftarrow$ DEQUEUE(S)
8 $\quad$ add u to linked list
9 $\quad$ **for** each edge $(u, v) \in E$
10 $\quad\quad active[v] \leftarrow active[v] - 1$
11 $\quad\quad$ **if** $active[v] = 0$
12 $\quad\quad\quad$ ENQUEUE(S, v)

24

Topological sort 2

```
TOPOLOGICAL-SORT2(G)
 1  for all edges (u, v) ∈ E
 2          active[v] ← active[v] + 1
 3  for all v ∈ V
 4          if active[v] = 0
 5                  ENQUEUE(S, v)
 6  while !EMPTY(S)
 7          u ← DEQUEUE(S)
 8          add u to linked list
 9          for each edge (u, v) ∈ E
10                  active[v] ← active[v] − 1
11                  if active[v] = 0
12                          ENQUEUE(S, v)
```

25

Topological sort 2

```
TOPOLOGICAL-SORT2(G)
 1  for all edges (u, v) ∈ E
 2          active[v] ← active[v] + 1
 3  for all v ∈ V
 4          if active[v] = 0
 5                  ENQUEUE(S, v)
 6  while !EMPTY(S)
 7          u ← DEQUEUE(S)
 8          add u to linked list
 9          for each edge (u, v) ∈ E
10                  active[v] ← active[v] − 1
11                  if active[v] = 0
12                          ENQUEUE(S, v)
```

26

Running time?

How many times do we process each node?

How many times do we process each edge?

O(|V| + |E|)

```
TOPOLOGICAL-SORT2(G)
 1  for all edges (u, v) ∈ E
 2          active[v] ← active[v] + 1
 3  for all v ∈ V
 4          if active[v] = 0
 5                  ENQUEUE(S, v)
 6  while !EMPTY(S)
 7          u ← DEQUEUE(S)
 8          add u to linked list
 9          for each edge (u, v) ∈ E
10                  active[v] ← active[v] − 1
11                  if active[v] = 0
12                          ENQUEUE(S, v)
```

27

Detecting cycles

Undirected graph

- BFS or DFS. If we reach a node we've seen already, then we've found a cycle

Directed graph

- Call TopologicalSort
- If the length of the list returned ≠ |V| then a cycle exists

28

Connectedness

Given an undirected graph, for every node $u \in V$, can we reach all other nodes in the graph? Algorithm + running time

Run BFS or DFS-Visit (one pass) and mark nodes as we visit them. If we visit all nodes, return true, otherwise false.

Running time: $O(|V| + |E|)$

29

Strongly connected

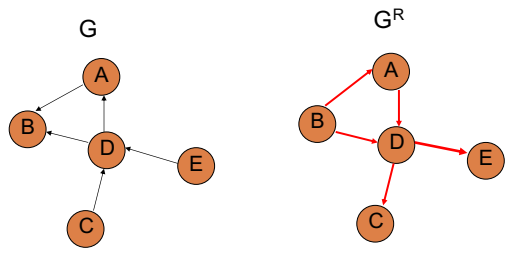
Given a directed graph, can we reach any node v from any other node u?

Can we do the same thing?

30

Transpose of a graph

Given a graph G, we can calculate the transpose of a graph G^R by reversing the direction of all the edges



Running time to calculate G^R? $\theta(|V| + |E|)$

31

Strongly connected

Strongly-Connected(G)
- Run DFS-Visit or BFS from some node u
- If not all nodes are visited: return false
- Create graph G^R
- Run DFS-Visit or BFS on G^R from node u
- If not all nodes are visited: return false
- return true

32

Is it correct?

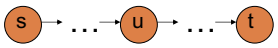
What do we know after the first pass?

- Starting at u, we can reach every node

What do we know after the second pass?

- All nodes can reach u. Why?
- We can get from u to every node in G^R, therefore, if we reverse the edges (i.e. G), then we have a path from every node to u

Which means that any node can reach any other node. Given any two nodes s and t we can create a path through u



33

Runtime?

Strongly-Connected(G)

- Run DFS-Visit or BFS from some node u — O(|V| + |E|)
- If not all nodes are visited: return false — O(|V|)
- Create graph G^R — θ(|V| + |E|)
- Run DFS-Visit or BFS on G^R from node u — O(|V| + |E|)
- If not all nodes are visited: return false — O(|V|)
- return true

O(|V| + |E|)

34

Shortest paths

Dijkstra's

Bellman-Ford

Floyd-Warshall

Johnson's

35