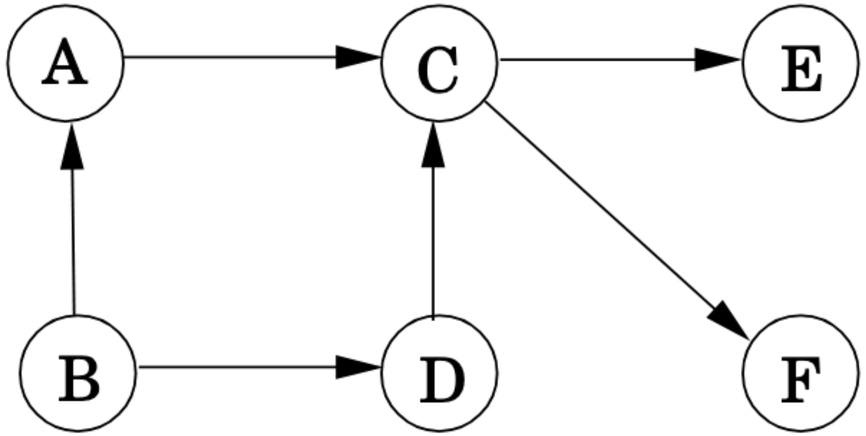
CS62 Class 23: Directed Acyclic Graphs

Graphs



Agenda

- Directed acyclic graphs
- Topological sorts
- Shortest paths on DAGs
- Longest paths
- DAGs in the wild (a research project example)

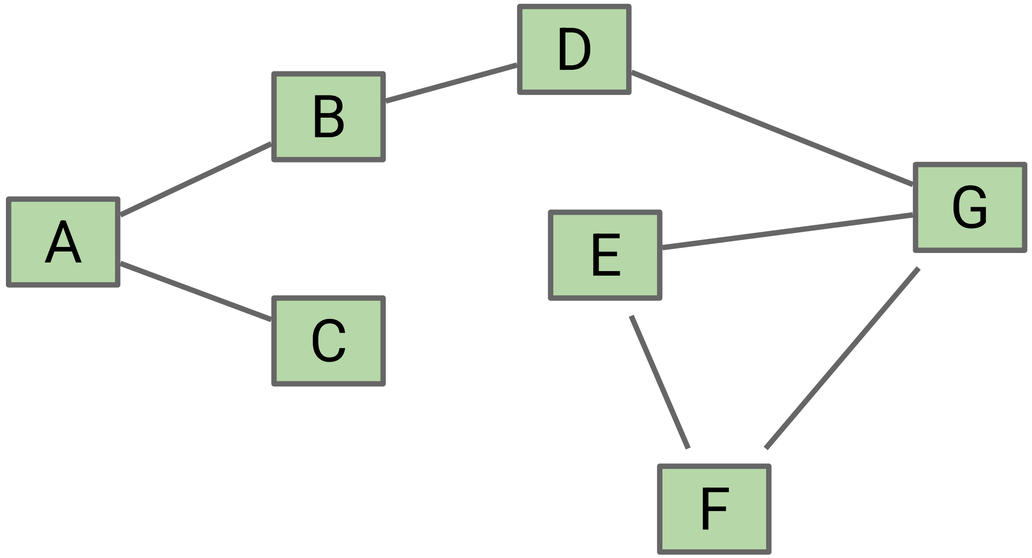
Reminder: graph algos so far

Problem	Problem Description	Solution	Efficiency
paths	Find a path from s to every reachable vertex.	DFS	O(V+E) time Θ(V) space
shortest paths	Find the shortest path from s to every reachable vertex.	BFS	O(V+E) time Θ(V) space
shortest weighted paths	Find the shortest path, considering weights, from s to every reachable vertex.	Dijkstra's	O(E log V) time Θ(V) space
minimum spanning tree	Find the minimum spanning tree.	Prim's	O(E log V) time Θ(V) space
minimum spanning tree	Find the minimum spanning tree.	Kruskal's	O(E log E) time Θ(E) space

Directed Acyclic Graphs

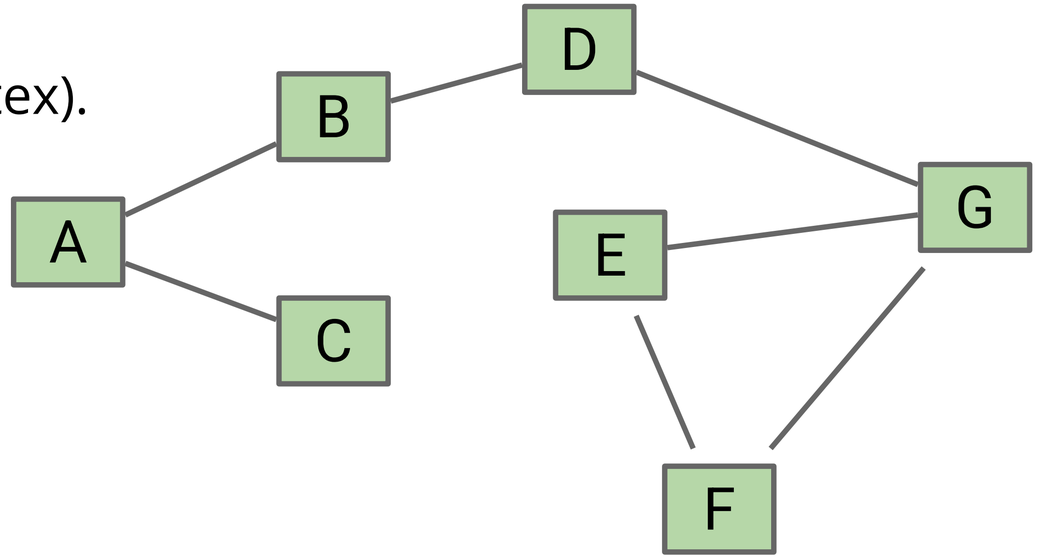
How to determine if a graph has cycles?

- Given a undirected graph, determine if it contains any cycles. What is its runtime?
  - May use any data structure or algorithm from the course so far.



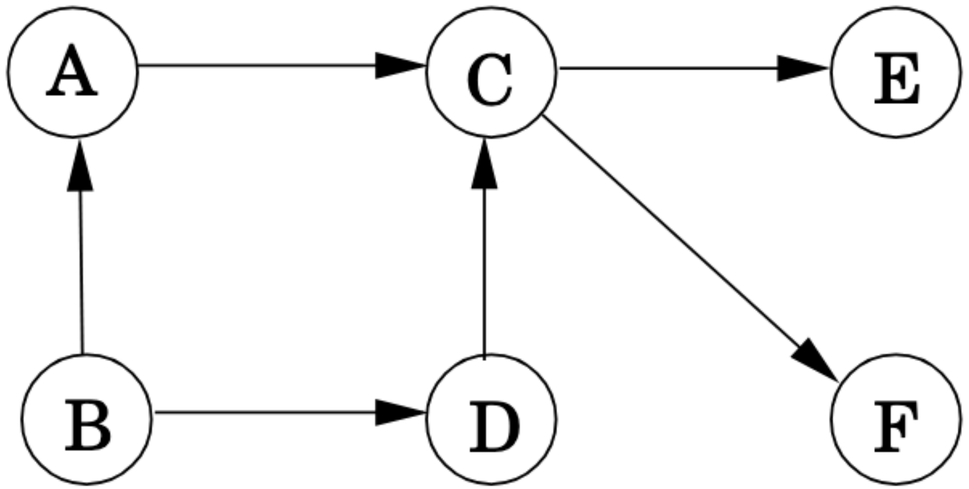
How to determine if a graph has cycles?

- Given a undirected graph, determine if it contains any cycles. What is its runtime?
  - May use any data structure or algorithm from the course so far.



- One possible approach: Do **DFS** from A (arbitrary vertex).
  - Keep going until you see a marked vertex.
  - Potential danger:
    - B looks back at A and sees marked.
  - Solution: Just don't count the node you came from.
- Worst case runtime: O(V + E).
  - With some cleverness, can give a tighter bound of O(V) (the number of edges we check is at most V, so O(V+E) = O(V))

Directed acyclic graph

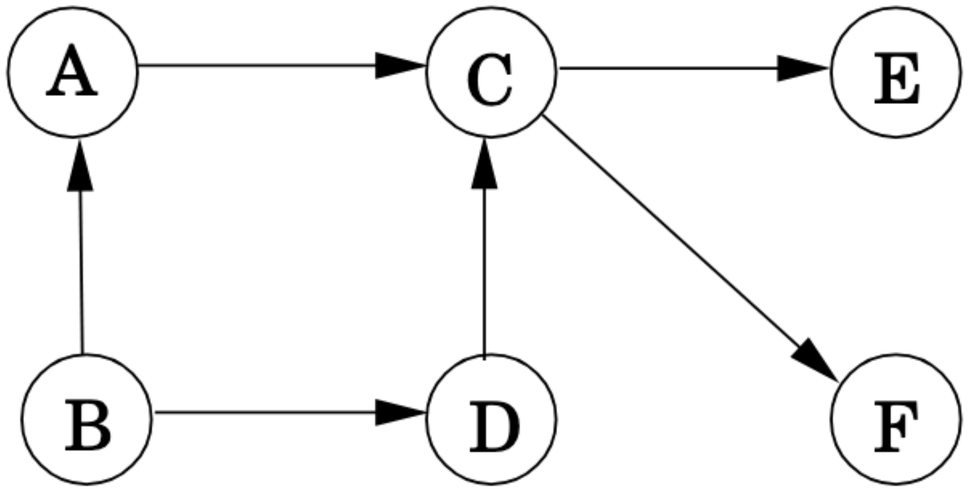


Q: This DAG has...

- how many sources?
- how many sinks?
- how many possible linearizations?
  hint: 1 valid one is B,A,D,C,E,F

- **Property:** A directed graph has a cycle if and only if its depth-first search reveals a back edge.
- A graph without cycles is **acyclic**.
- DAGs are good for modeling dependencies: before you do C, you have to do A; before you do A, you have to do B.
- A **linearization** or a **topological sort** of a DAG is an ordering of its vertices so that for every directed edge from vertex u to vertex v, u comes before v in the ordering.

Directed acyclic graph



Q: This DAG has...

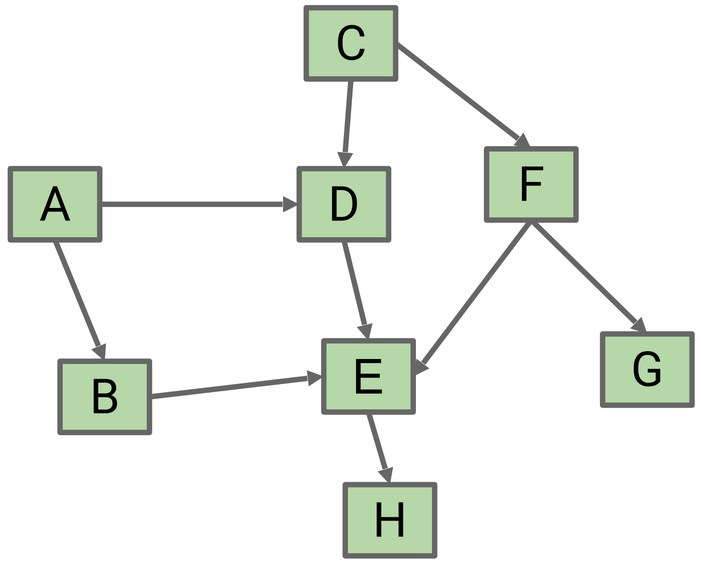
- how many sources? 1 (A)
- how many sinks? 2 (E, F)
- how many possible linearizations?
  hint: 1 valid one is B,A,D,C,E,F

  4. other 3: B,A,D,C,**F,E**; B,**D,A**,C,F,E; B,D,A,C,E,F

- **Property:** A directed graph has a cycle if and only if its depth-first search reveals a back edge.
- A graph without cycles is **acyclic**.
- DAGs are good for modeling dependencies: before you do C, you have to do A; before you do A, you have to do B.
- A **linearization** or a **topological sort** of a DAG is an ordering of its vertices so that for every directed edge from vertex u to vertex v, u comes before v in the ordering.

Topological sorts

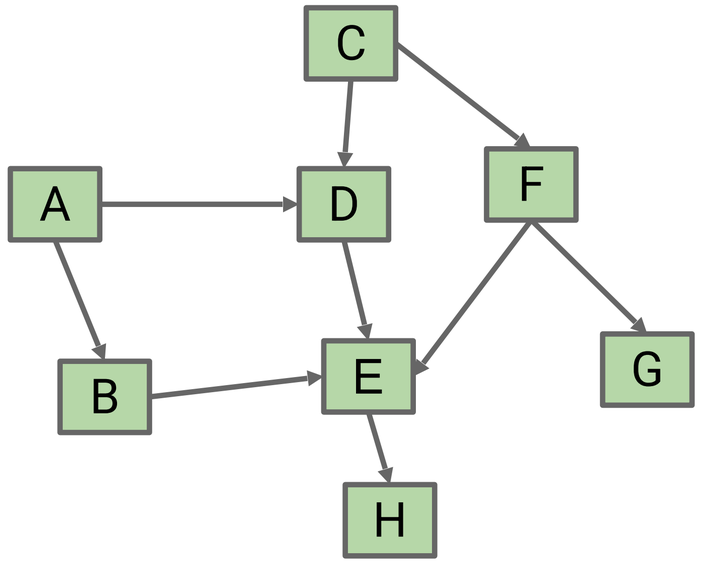
Topological Sort



Suppose we have tasks A through H, where an arrow from v to u indicates that v must happen before u.

- What algorithm do we use to find a valid ordering for these tasks?
- Valid orderings include: [A, C, B, D, F, E, H, G], [C, A, D, F, B, E, G, H], ...

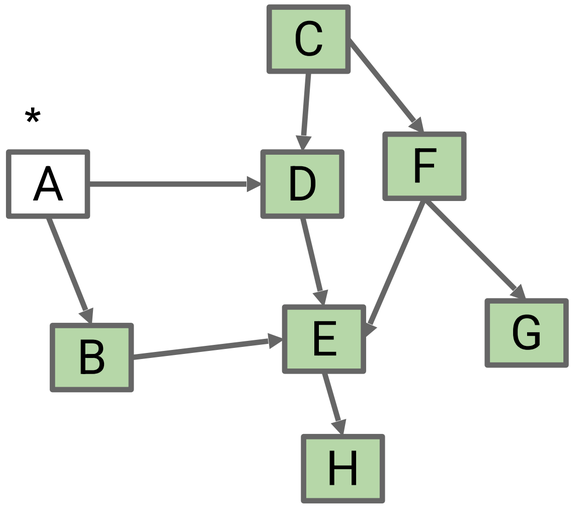
Solution



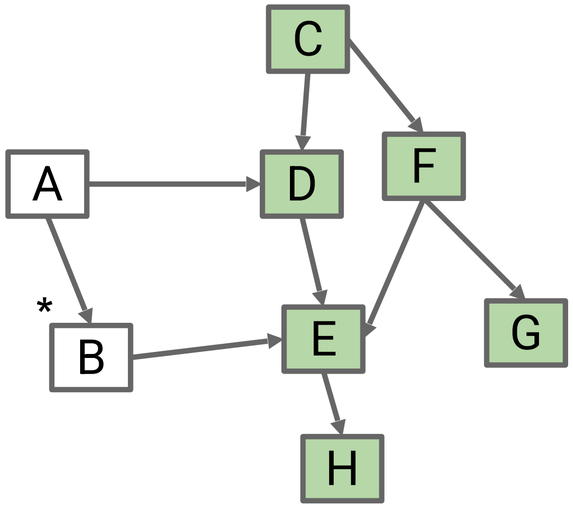
Perform a DFS traversal from every source vertex. Do NOT clear markings in between traversals.

- Record DFS postorder (left, right, root) in a list.
- Topological ordering is given by the reverse of that list (reverse postorder).

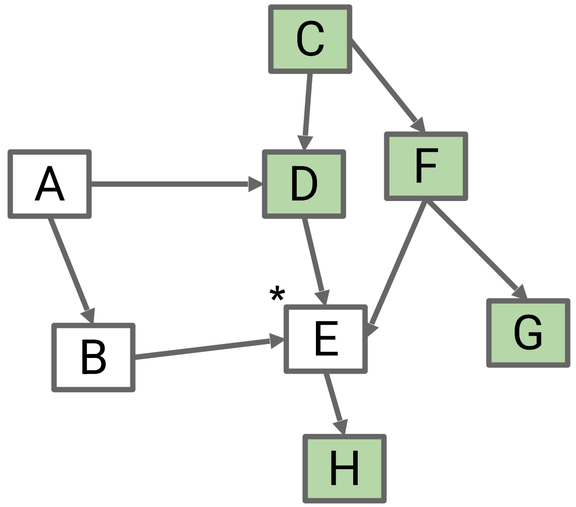
Topological Sort (Demo 1/2)



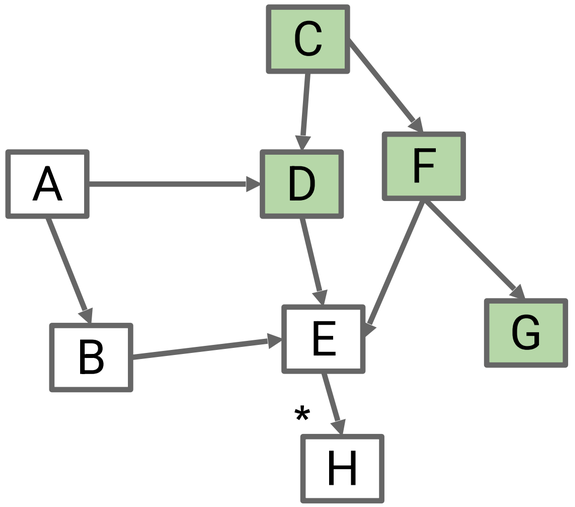
Postorder: []
Call stack: A



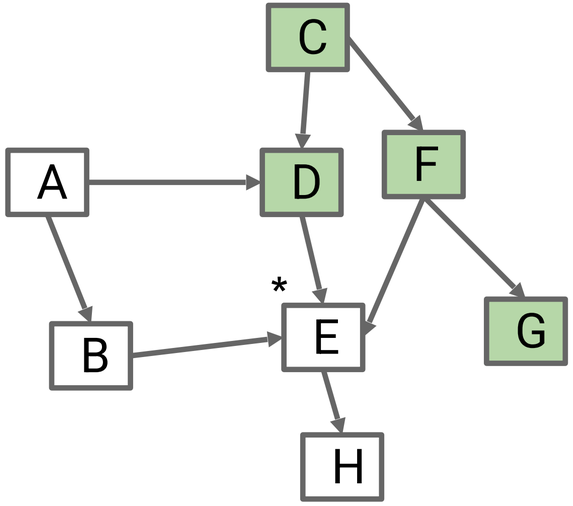
Postorder: []
Call stack: A→B



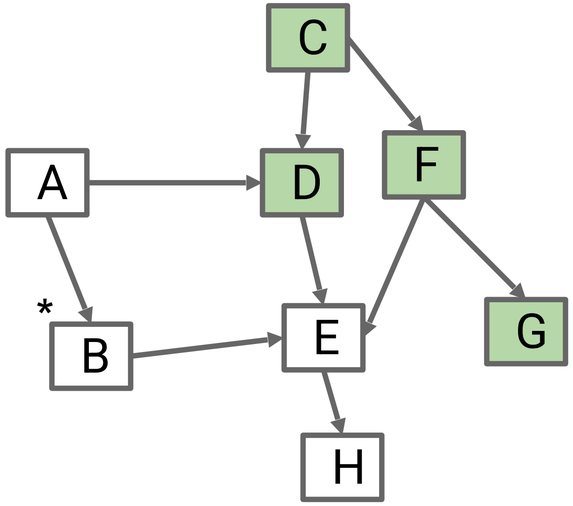
Postorder: []
Call stack: A→B→E



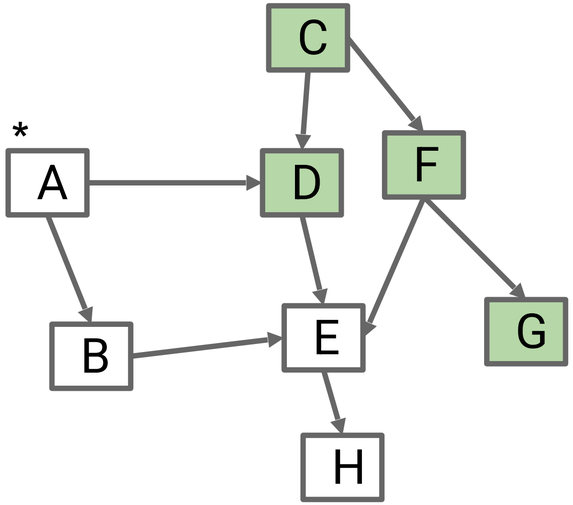
Postorder: [H]
Call stack: A→B→E→H



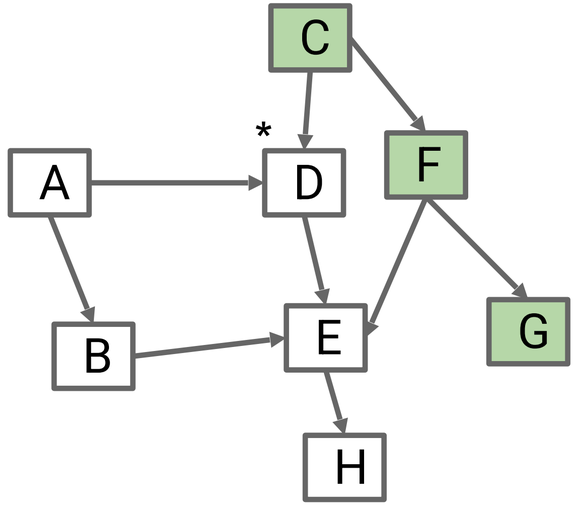
Postorder: [H, E]
Call stack: A→B→E



Postorder: [H, E, B]
Call stack: A→B

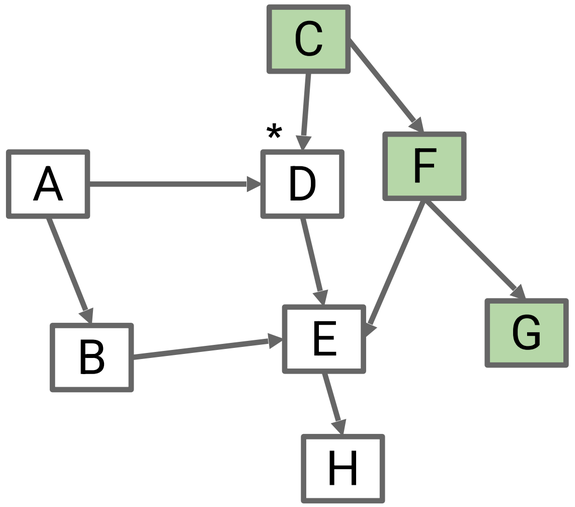


Postorder: [H, E, B]
Call stack: A

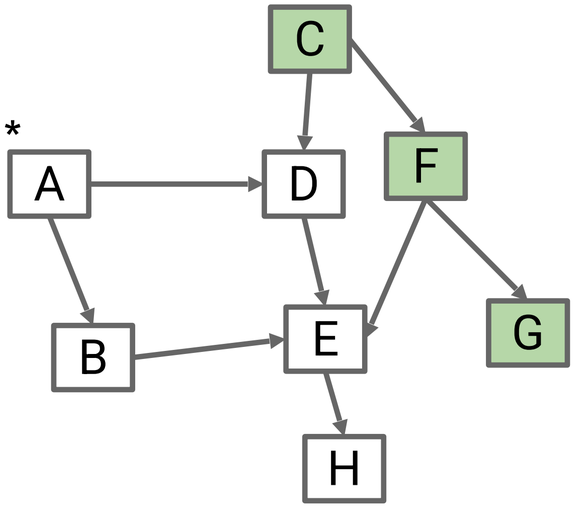


Postorder: [H, E, B, D]
Call stack: A→D

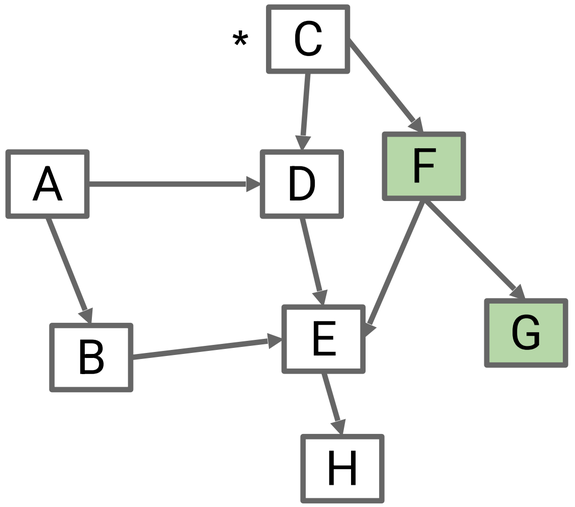
Topological Sort (Demo 2/2)



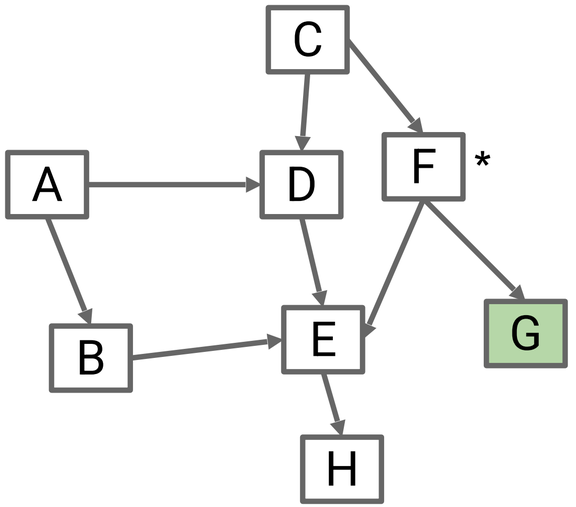
Postorder: [H, E, B, D]
Call stack: A→D



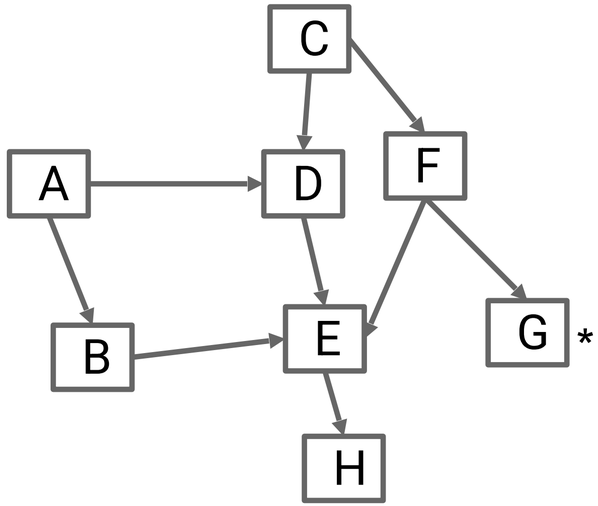
Postorder: [H, E, B, D, A]
Call stack: A



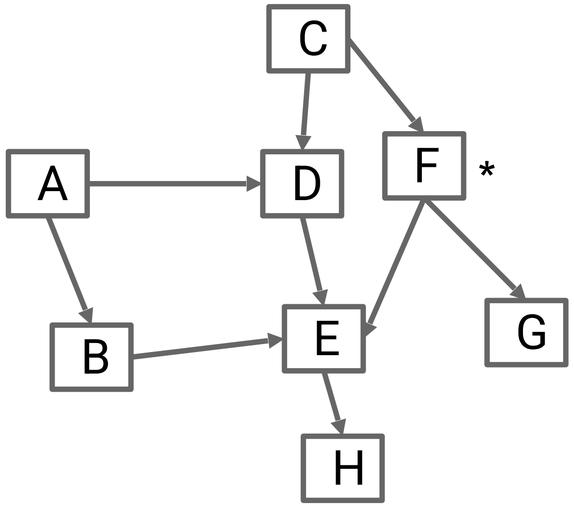
Postorder: [H, E, B, D, A]
Call stack: C



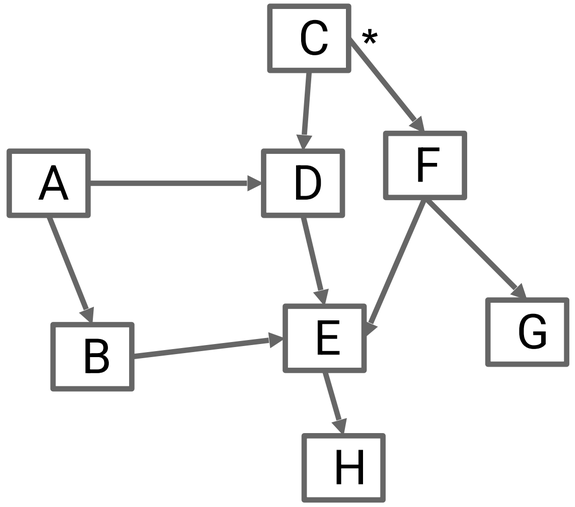
Postorder: [H, E, B, D, A]
Call stack: C→F



Postorder: [H, E, B, D, A, G]
Call stack: C→F→G

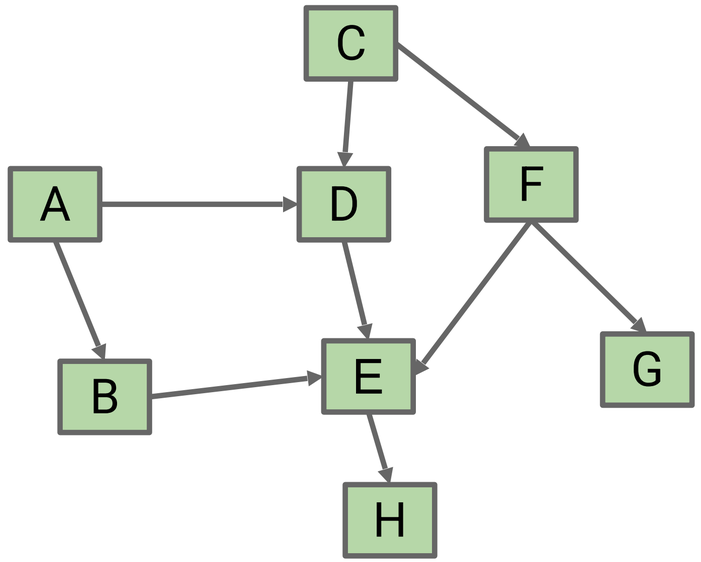


Postorder: [H, E, B, D, A, G, F]
Call stack: C→F



Postorder: [H, E, B, D, A, G, F, C]
Call stack: C

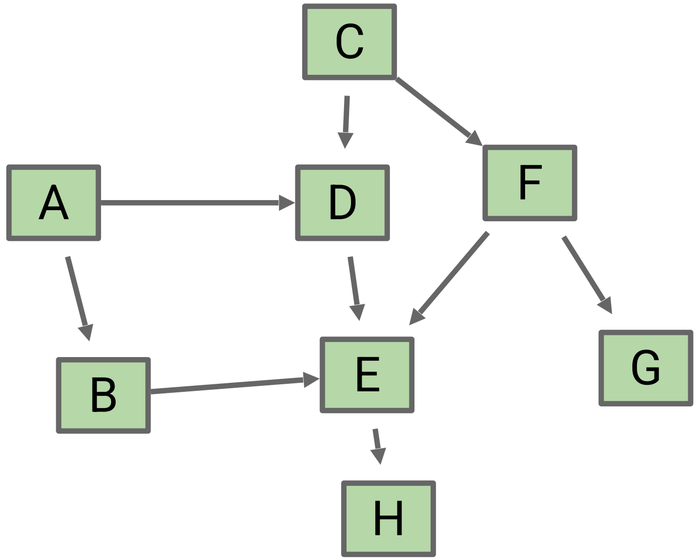
Solution

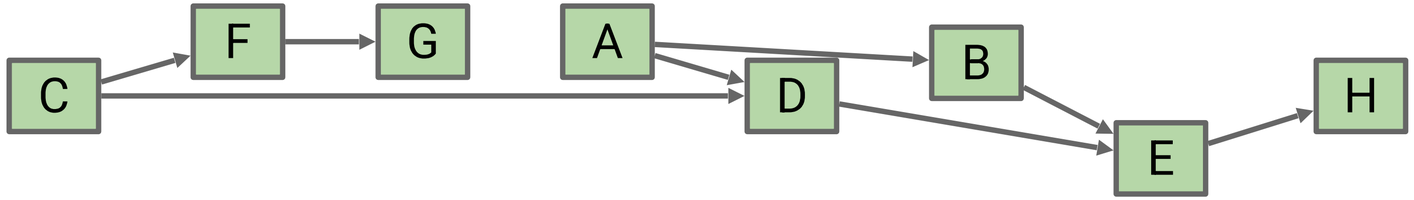


Perform a DFS traversal from every vertex with indegree 0, NOT clearing markings in between traversals.

- Record DFS postorder in a list: [H, E, B, D, A, G, F, C]
- Topological ordering is given by the reverse of that list (reverse postorder):
  - [C, F, G, A, D, B, E, H]

Topological Sort

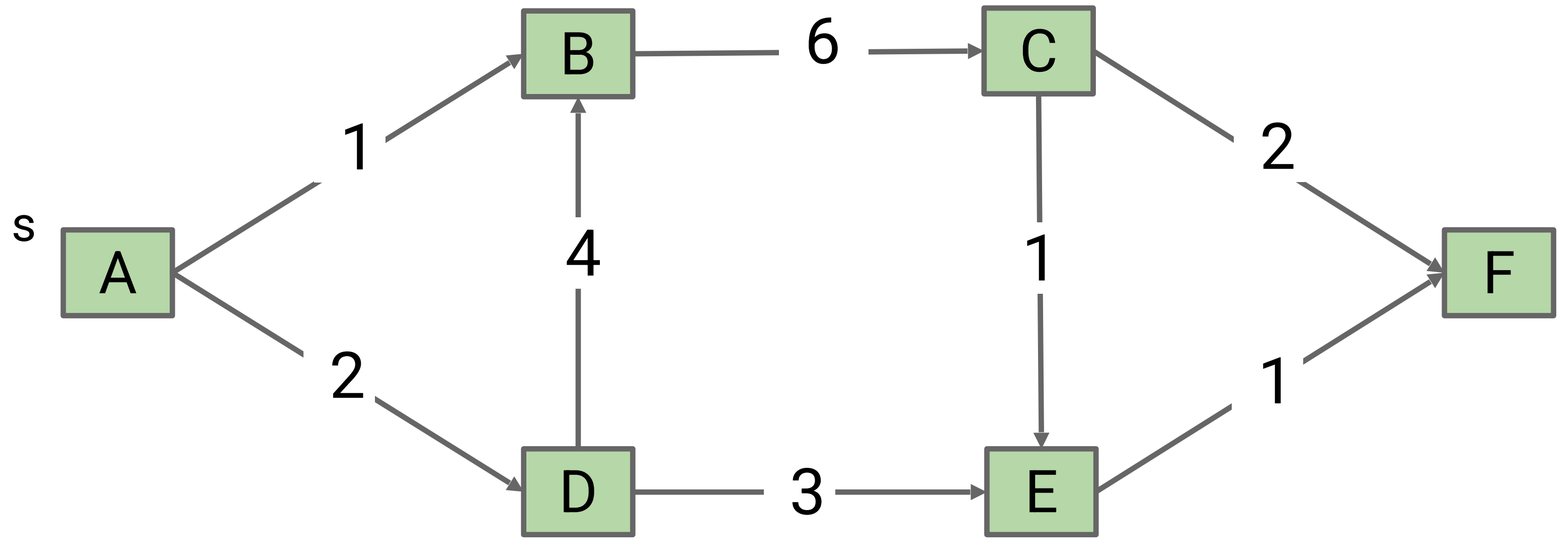




- Topological ordering: [C, F, G, A, D, B, E, H]
- When nodes are topologically sorted in a diagram, arrows all point rightwards.
- Be aware, that when people say “Depth First Search”, they sometimes mean with restarts, and they sometimes mean without.
- For example, when we did DFS for reachability, we did not restart.
- For Topological Sort, we restarted from every vertex with indegree 0 (source).

Worksheet time!

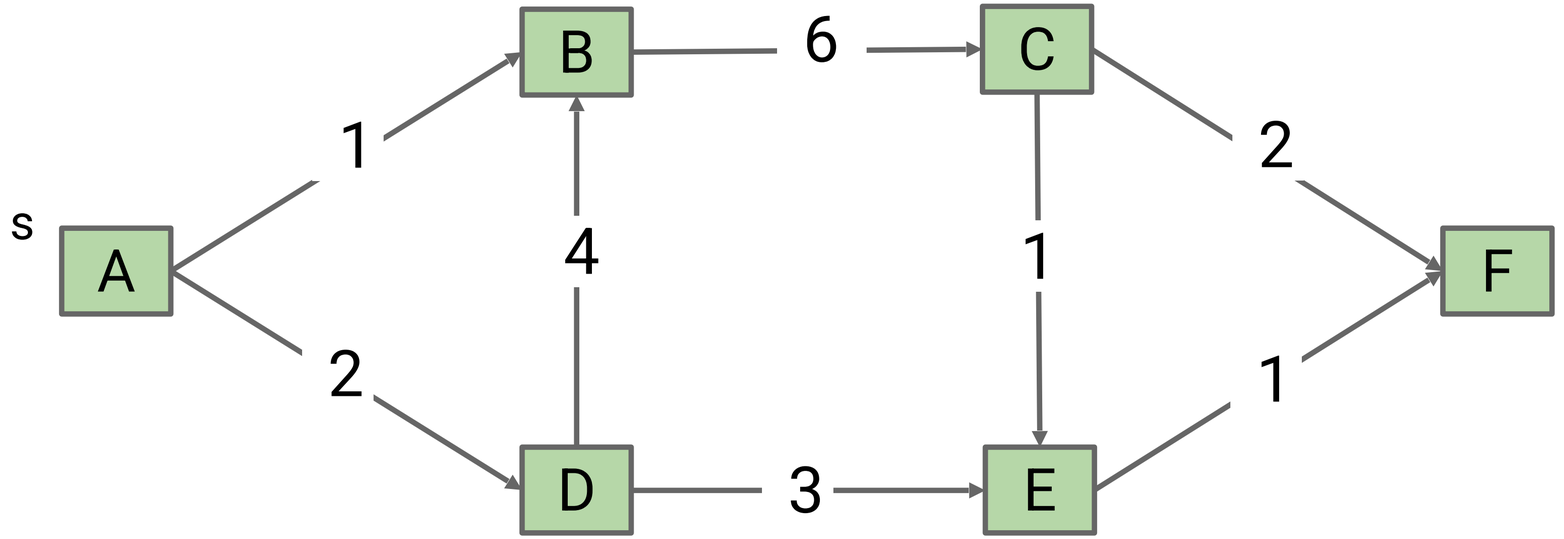
Give a topological ordering for the DAG below (a.k.a. topological sort). When you can choose from multiple vertices, take the one with lower edge weight.

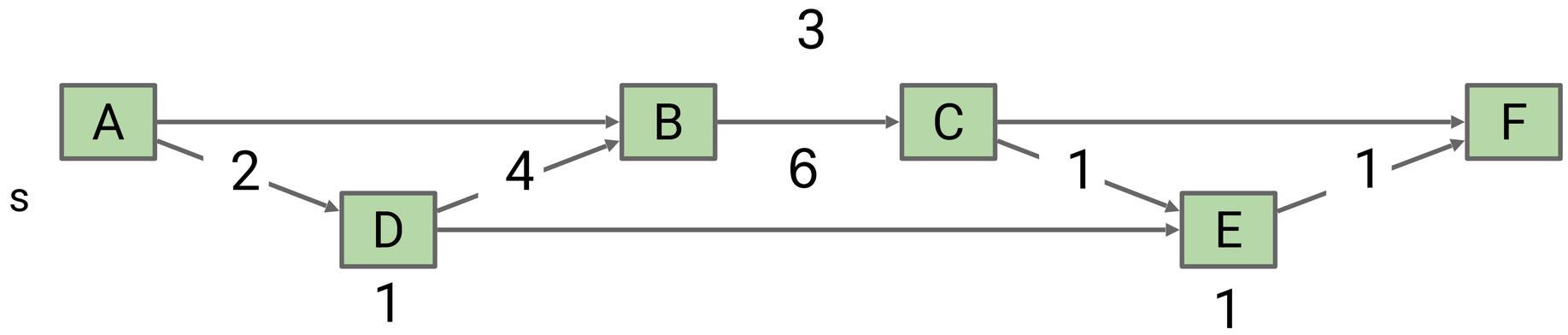


Worksheet answer

Give a topological ordering for the DAG below (a.k.a. topological sort)

- A, D, B, C, E, F (because DFS postorder was FECBDA)

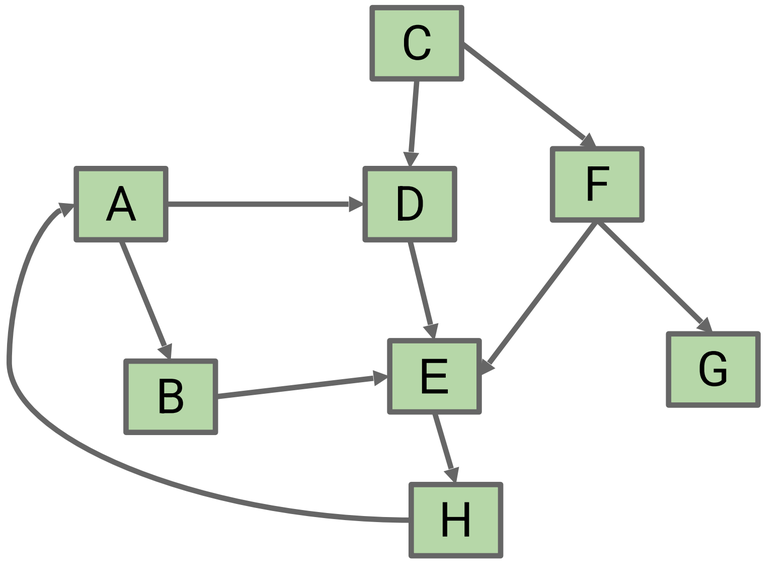




Topological sorts can only work on DAGs

A topological sort only exists if the graph is a directed acyclic graph (DAG).

- For the graph below, there is NO possible ordering where all arrows are respected.



DAGs appear in many real world applications (causalities, hierarchies, temporal dependencies), and there are many graph algorithms that only work on DAGs.

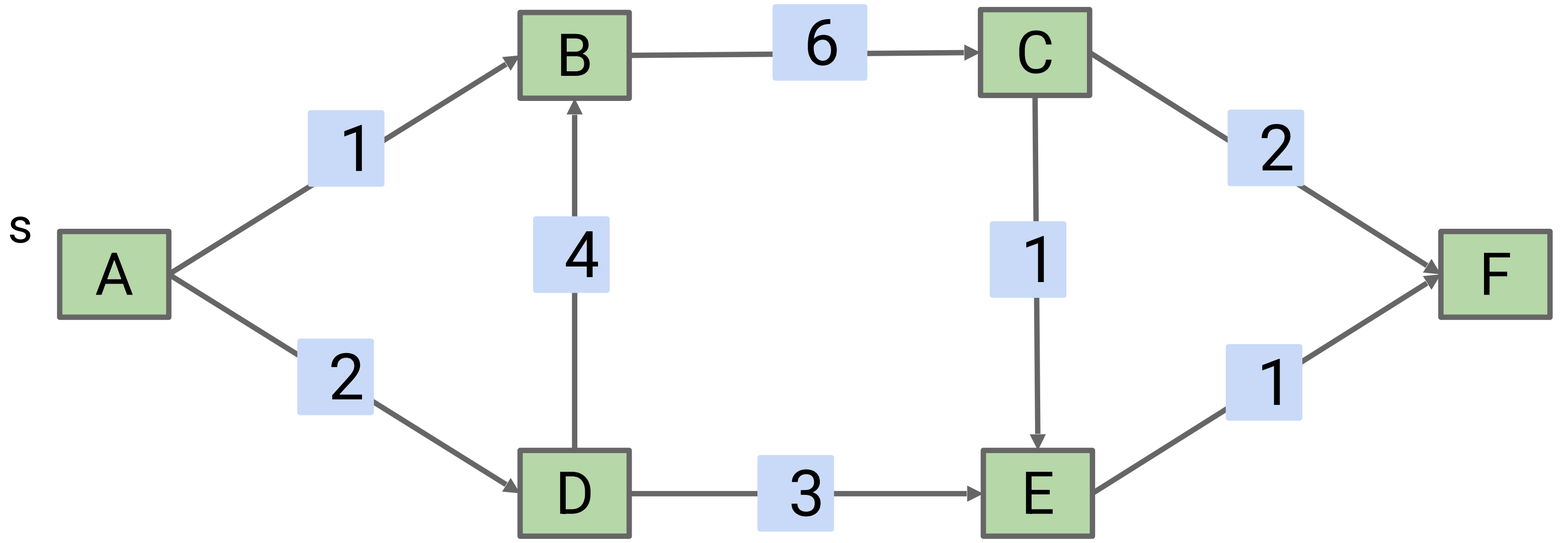
Graph algos so far

Problem	Problem Description	Solution	Efficiency
paths	Find a path from s to every reachable vertex.	DFS	O(V+E) time Θ(V) space
shortest paths	Find the shortest path from s to every reachable vertex.	BFS	O(V+E) time Θ(V) space
shortest weighted paths	Find the shortest path, considering weights, from s to every reachable vertex.	Dijkstra's	O(E log V) time Θ(V) space
minimum spanning tree	Find the minimum spanning tree.	Prim's	O(E log V) time Θ(V) space
minimum spanning tree	Find the minimum spanning tree.	Kruskal's	O(E log E) time Θ(E) space
topological sort	Find an ordering of vertices that respects edges of our DAG.	DFS from source nodes	O(V+E) time Θ(V) space Why? It's just DFS.

Shortest Paths on DAGs

Shortest Paths Warmup

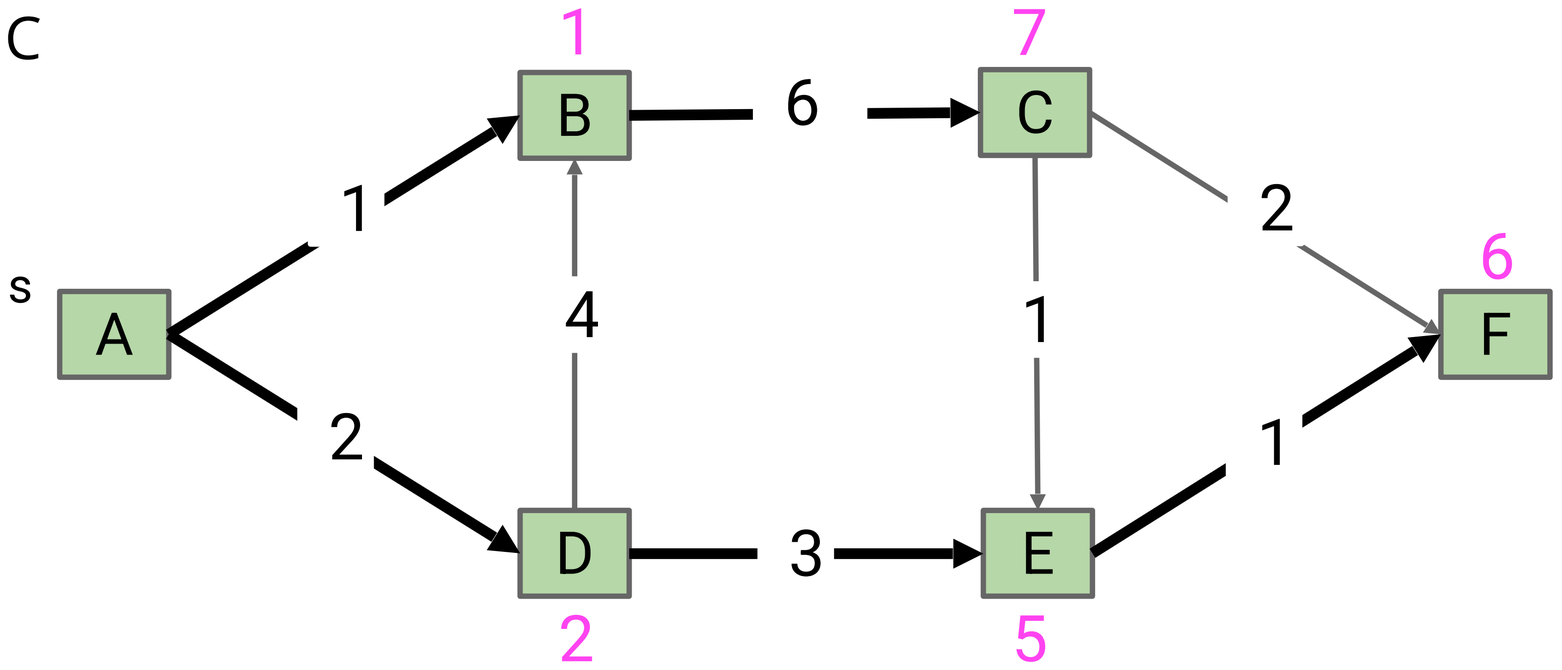
What is the shortest paths tree for the graph below, using s as the source?
In what order will Dijkstra's algorithm visit the vertices?



Shortest Paths Warmup

What is the shortest paths tree for the graph below, using s as the source?
In what order will Dijkstra's algorithm visit the vertices?

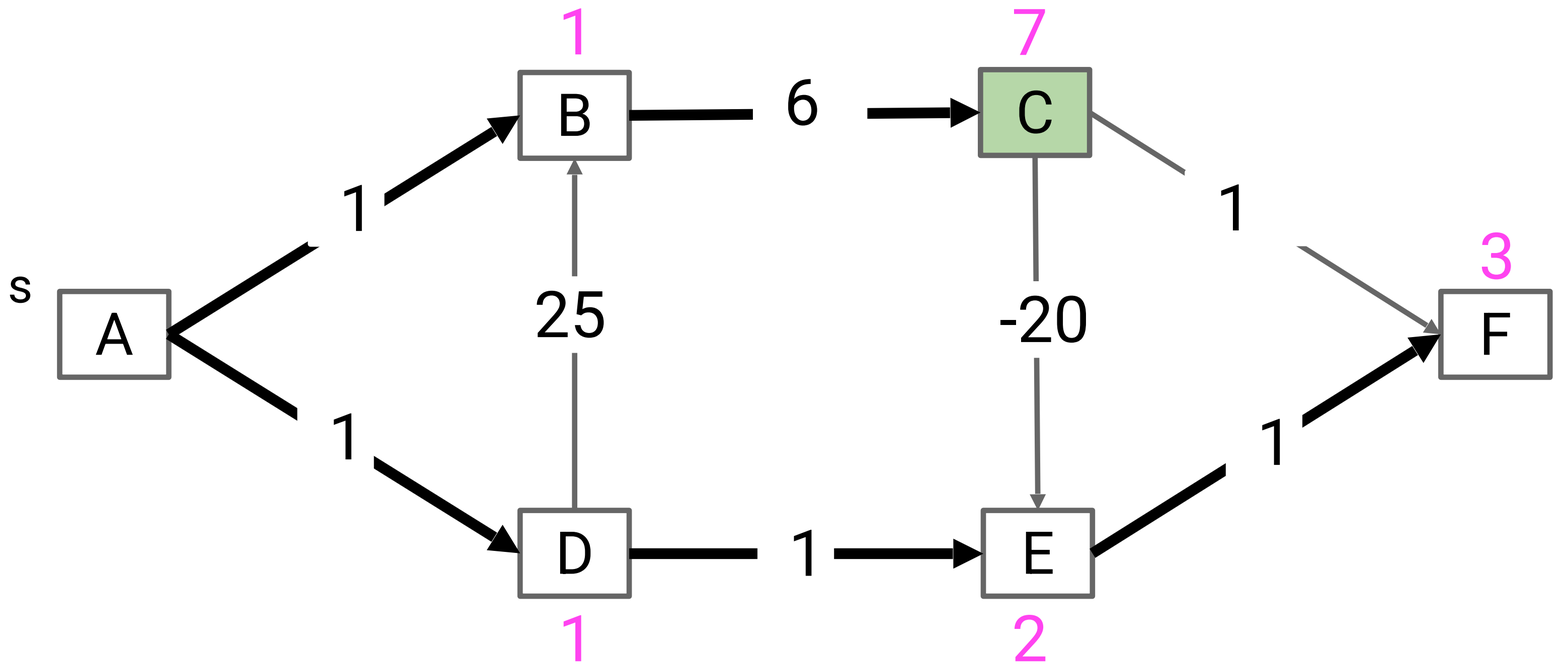
- A, B, D, E, F, C



Shortest Paths with negative edges

If we allow negative edges, Dijkstra's algorithm can fail.

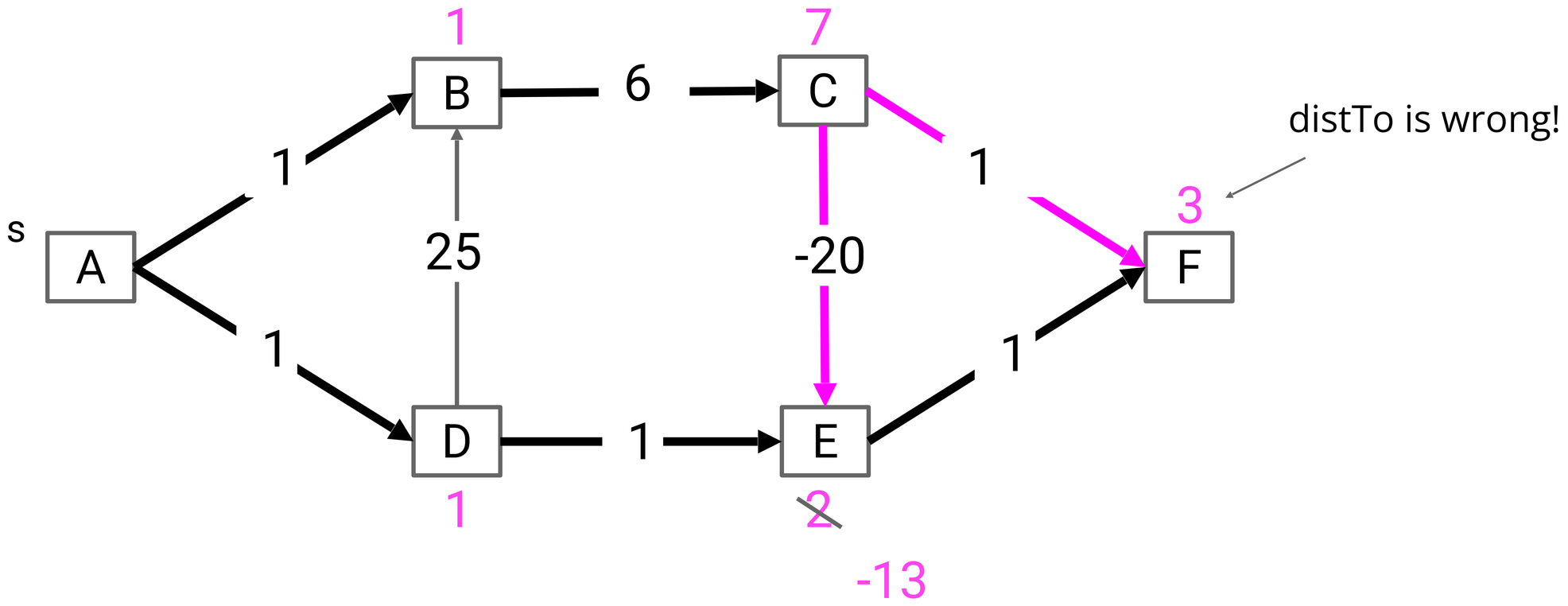
- For example, below we see Dijkstra's just before vertex C is visited.



Shortest Paths with negative edges

If we allow negative edges, Dijkstra's algorithm can fail.

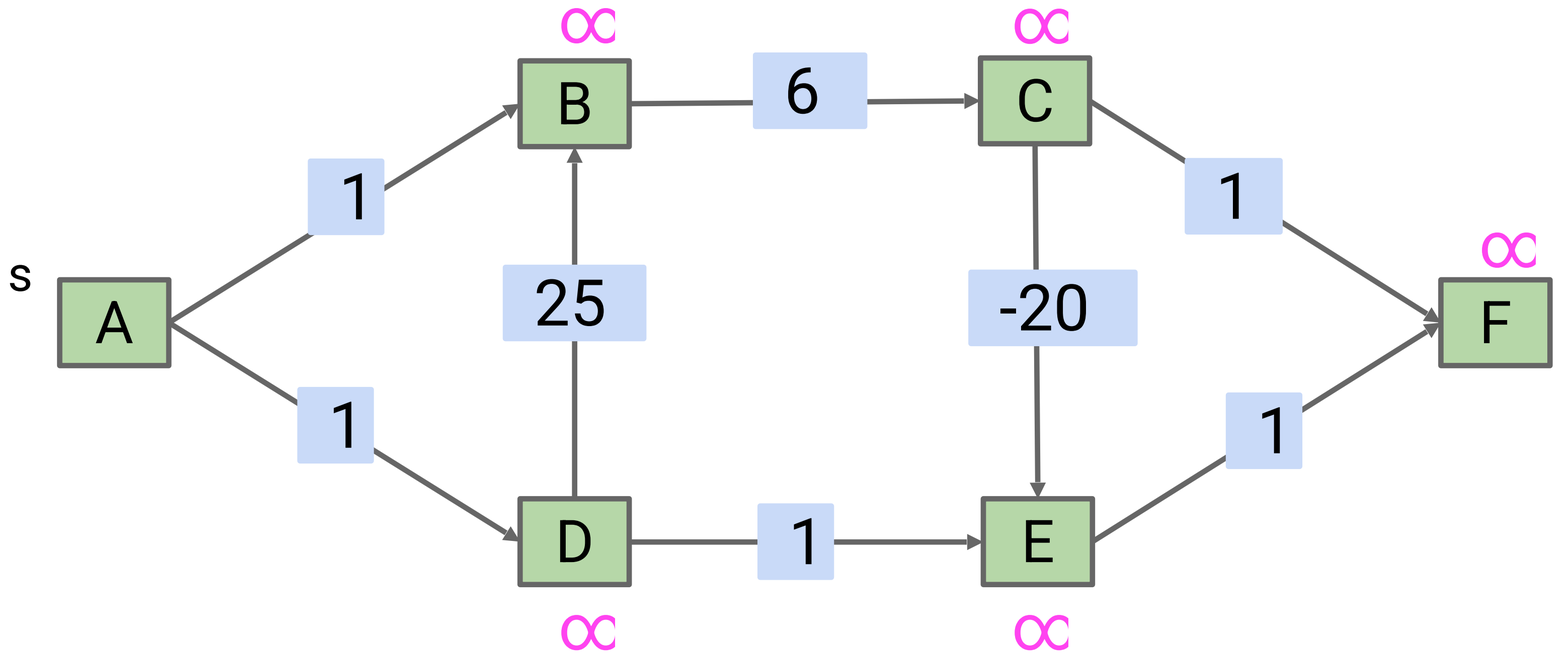
- For example, below we see Dijkstra's just before vertex C is visited.
- Relaxation on E succeeds, but distance to F (now should be -10) will never be updated.



But, shortest paths on DAGs with negative weights will work

Try to come up with an algorithm for shortest paths on a DAG that works even if there are negative edges.

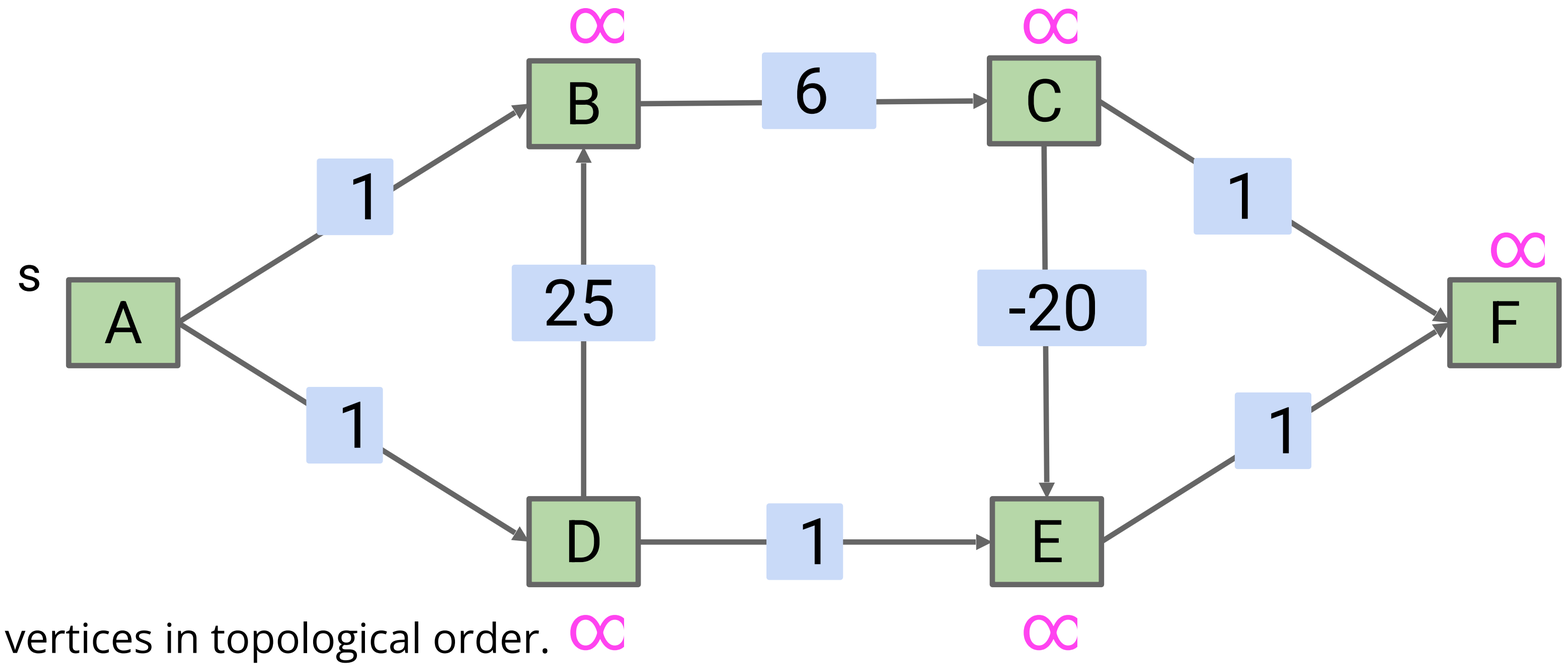
- Hint: You should still use the “relax” operation as a basic building block.



But, shortest paths on DAGs with negative weights will work

Try to come up with an algorithm for shortest paths on a DAG that works even if there are negative edges.

- Hint: You should still use the “relax” operation as a basic building block.

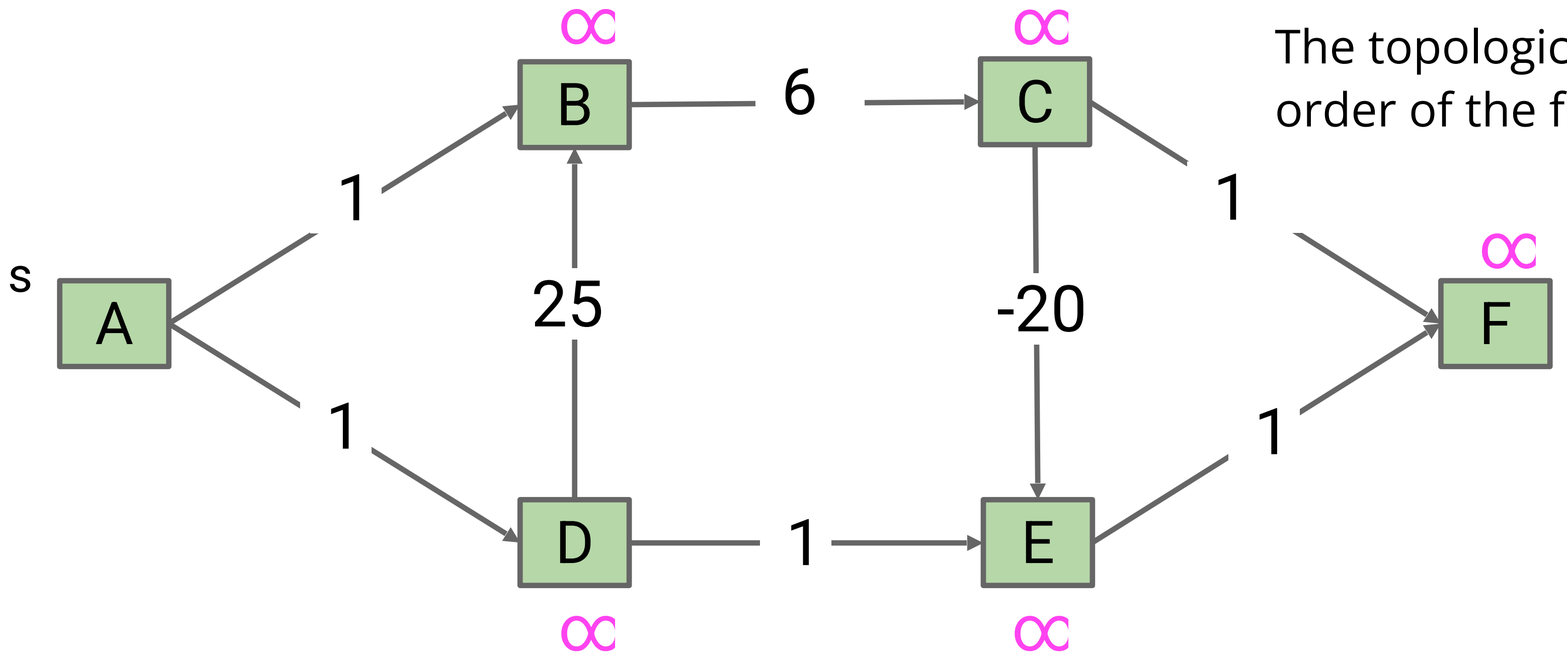


One simple idea: Visit vertices in topological order.

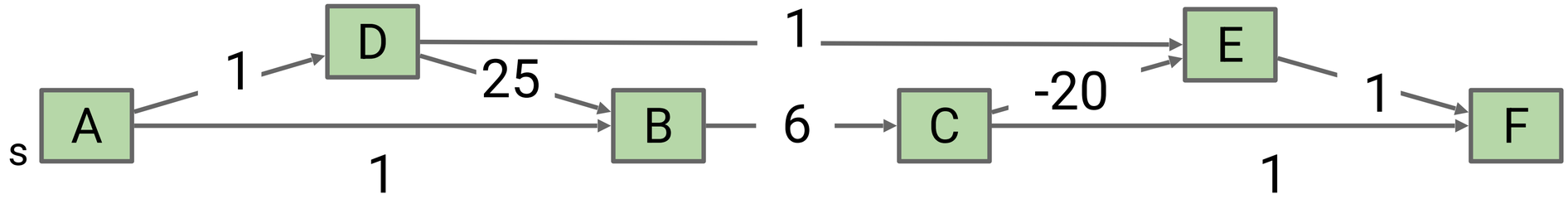
- On each visit, relax all outgoing edges.
- Each vertex is visited only when all possible info about it has been used!

The DAG SPT Algorithm: Relax in Topological Order

First: We have to find a topological order, e.g. ADBCEF. Runtime is O(V + E).

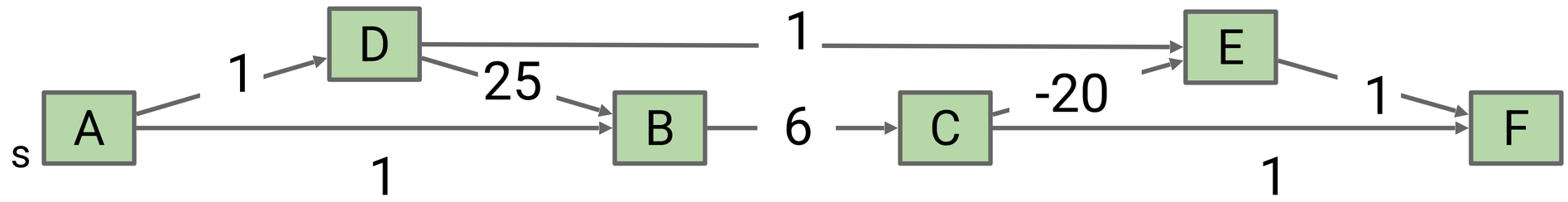


The topological order is just the order of the fringe priority queue!



The DAG SPT Algorithm: Relax in Topological Order

Second: We have to visit all the vertices in topological order, relaxing all edges as we go. Let's see a demo.

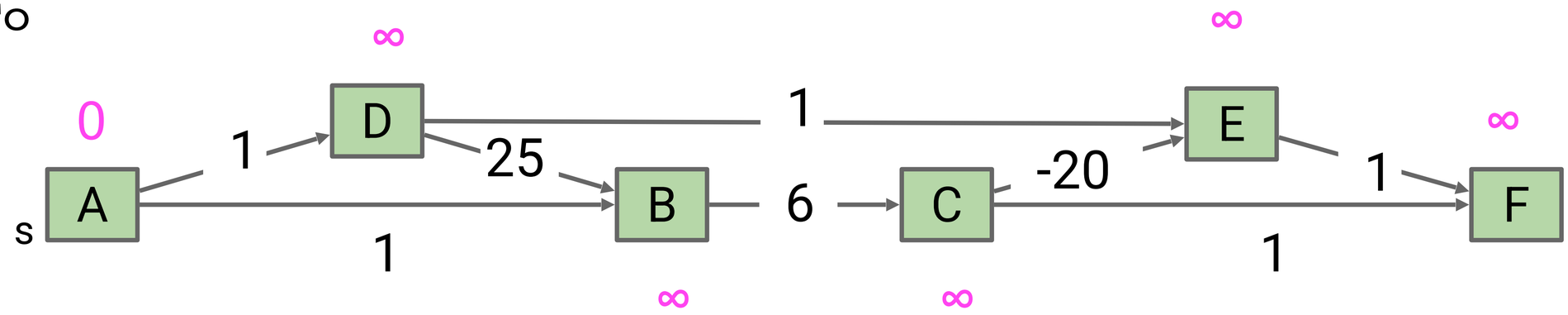


DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax all of its going edges.

	distTo	edgeTo
A	0	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-
F	∞	-



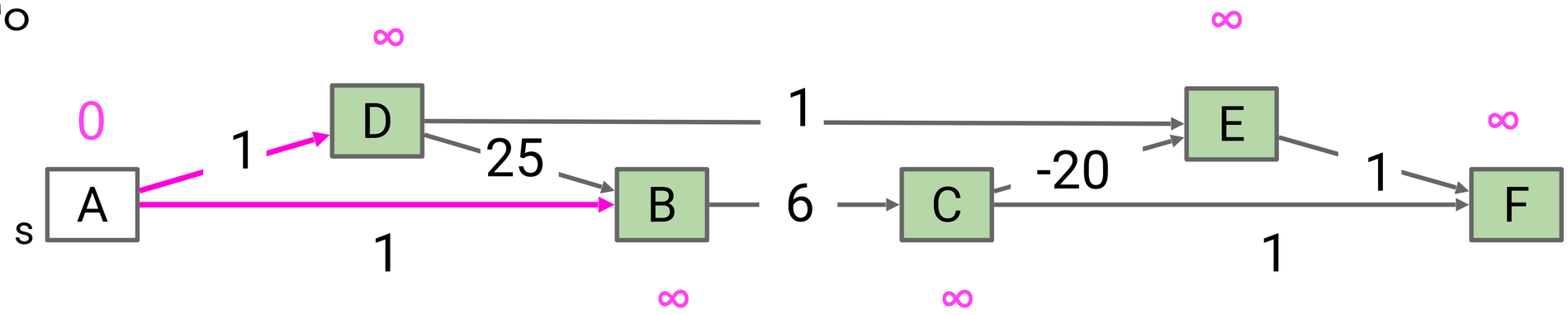
Fringe: [A, D, B, C, E, F]

DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax **all** of its going edges.

	distTo	edgeTo
A	0	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-
F	∞	-



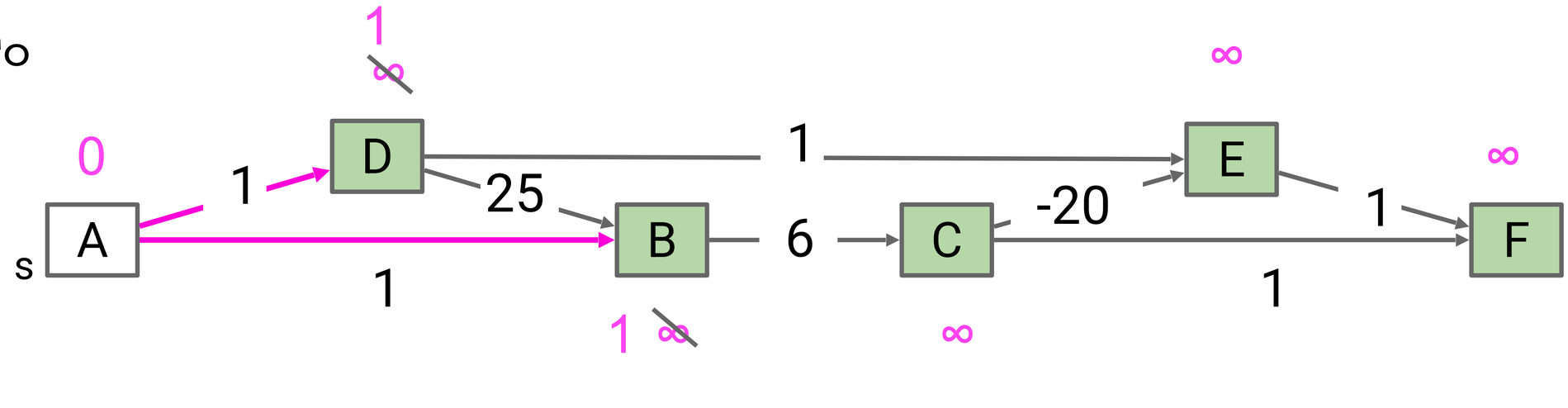
Fringe: [A, D, B, C, E, F]

DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax all of its going edges.

	distTo	edgeTo
A	0	-
B	1	A
C	∞	-
D	1	A
E	∞	-
F	∞	-



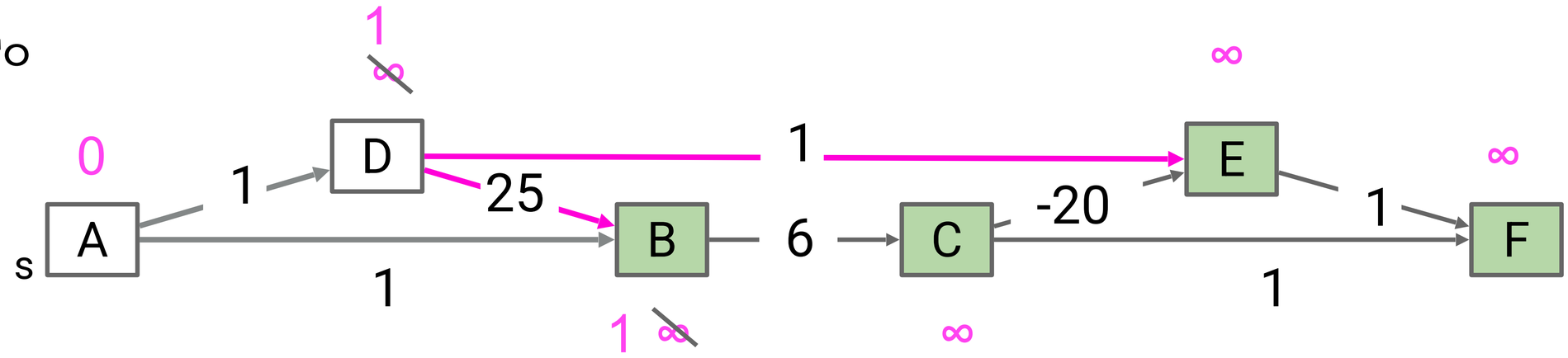
Fringe: [A, D, B, C, E, F]

DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax all of its going edges.

	distTo	edgeTo
A	0	-
B	1	A
C	∞	-
D	1	A
E	∞	-
F	∞	-



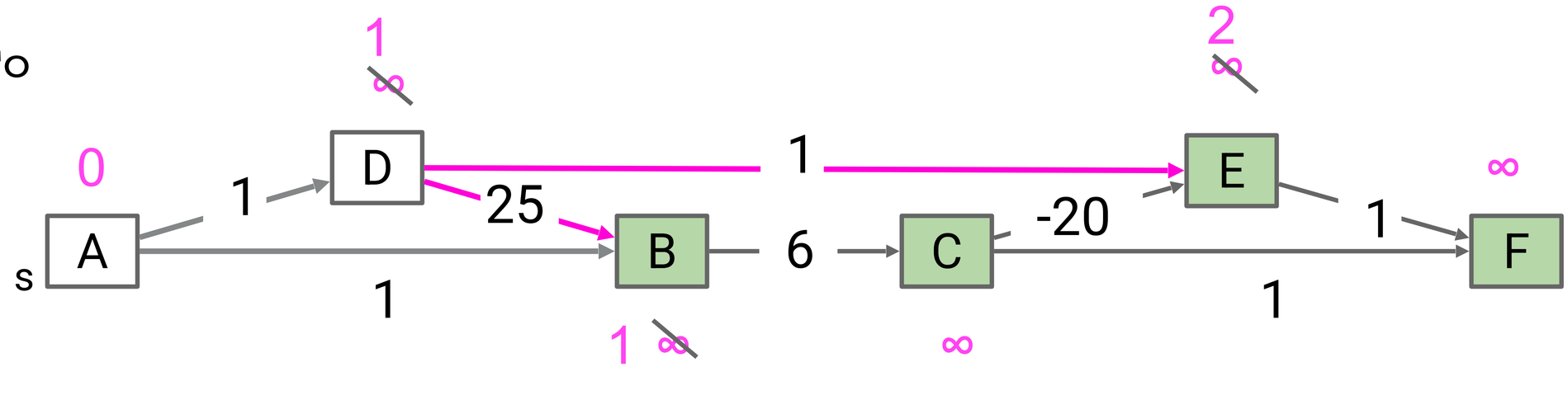
Fringe: [~~A~~, ~~D~~, B, C, E, F]

DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax all of its going edges.

	distTo	edgeTo
A	0	-
B	1	A
C	∞	-
D	1	A
E	2	D
F	∞	-



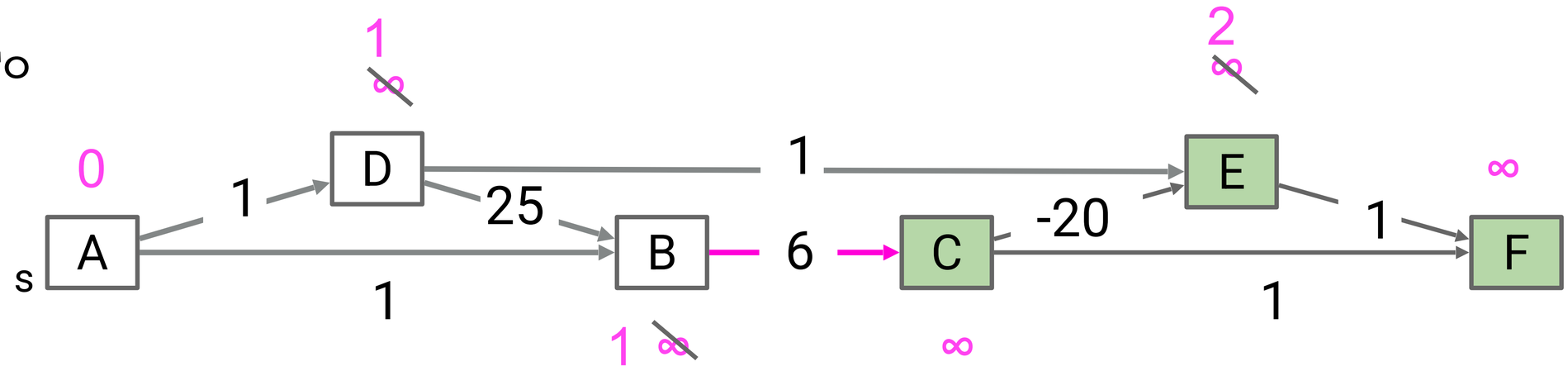
Fringe: [~~A~~, ~~D~~, B, C, E, F]

DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax all of its going edges.

	distTo	edgeTo
A	0	-
B	1	A
C	∞	-
D	1	A
E	2	D
F	∞	-



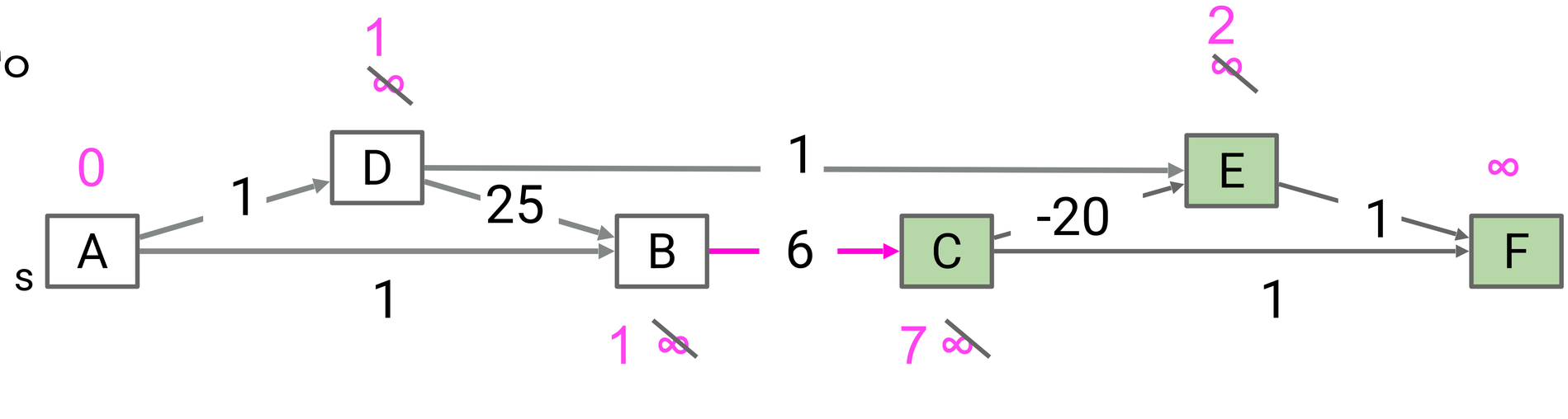
Fringe: [~~A~~, ~~D~~, ~~B~~, C, E, F]

DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax all of its going edges.

	distTo	edgeTo
A	0	-
B	1	A
C	7	B
D	1	A
E	2	D
F	∞	-



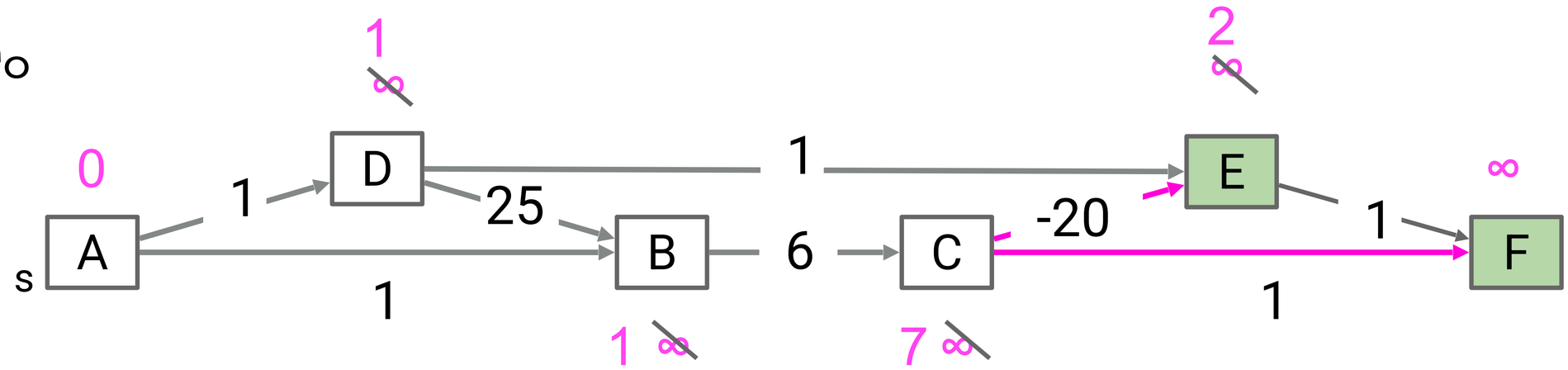
Fringe: [~~A~~, ~~D~~, ~~B~~, C, E, F]

DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax all of its going edges.

	distTo	edgeTo
A	0	-
B	1	A
C	7	B
D	1	A
E	2	D
F	∞	-



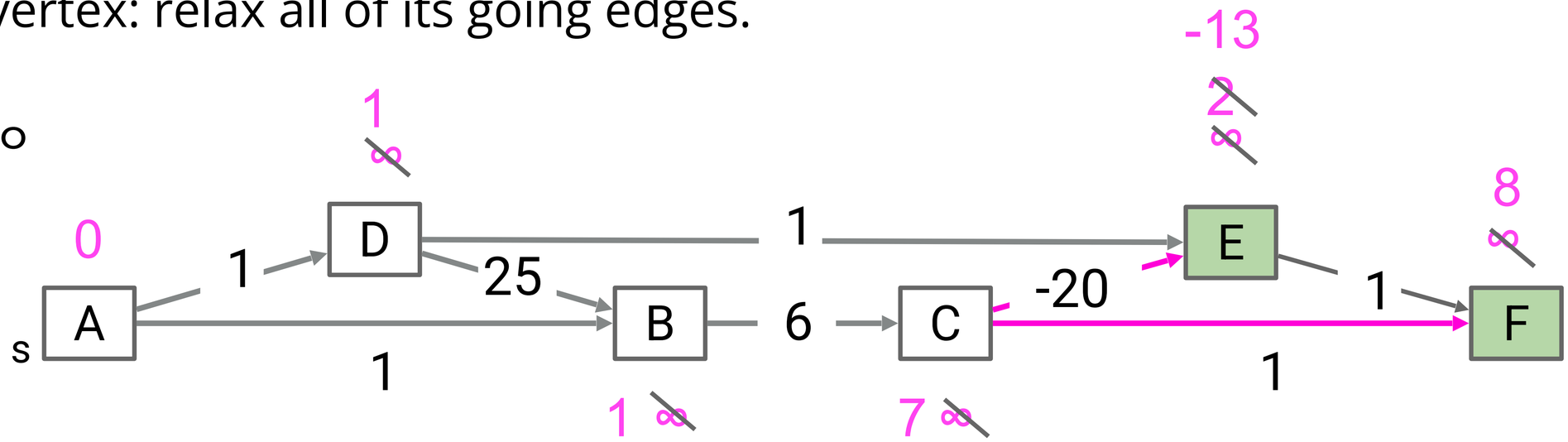
Fringe: [~~A~~, ~~D~~, ~~B~~, ~~C~~, E, F]

DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax all of its going edges.

	distTo	edgeTo
A	0	-
B	1	A
C	7	B
D	1	A
E	-13	C
F	8	C



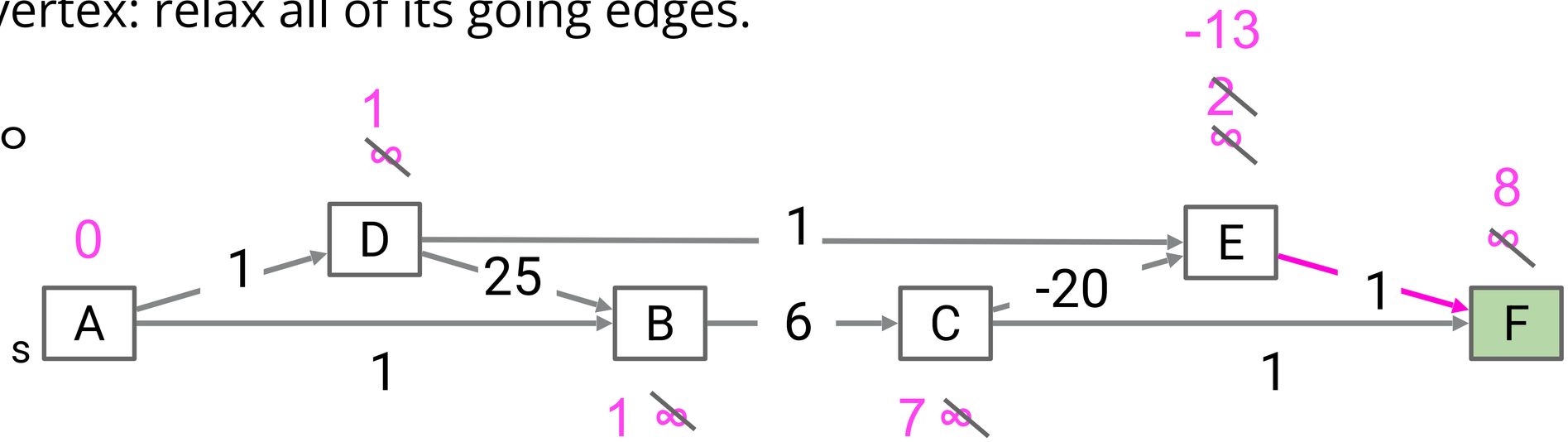
Fringe: [~~A~~, ~~D~~, ~~B~~, ~~C~~, E, F]

DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax all of its going edges.

	distTo	edgeTo
A	0	-
B	1	A
C	7	B
D	1	A
E	-13	C
F	8	C



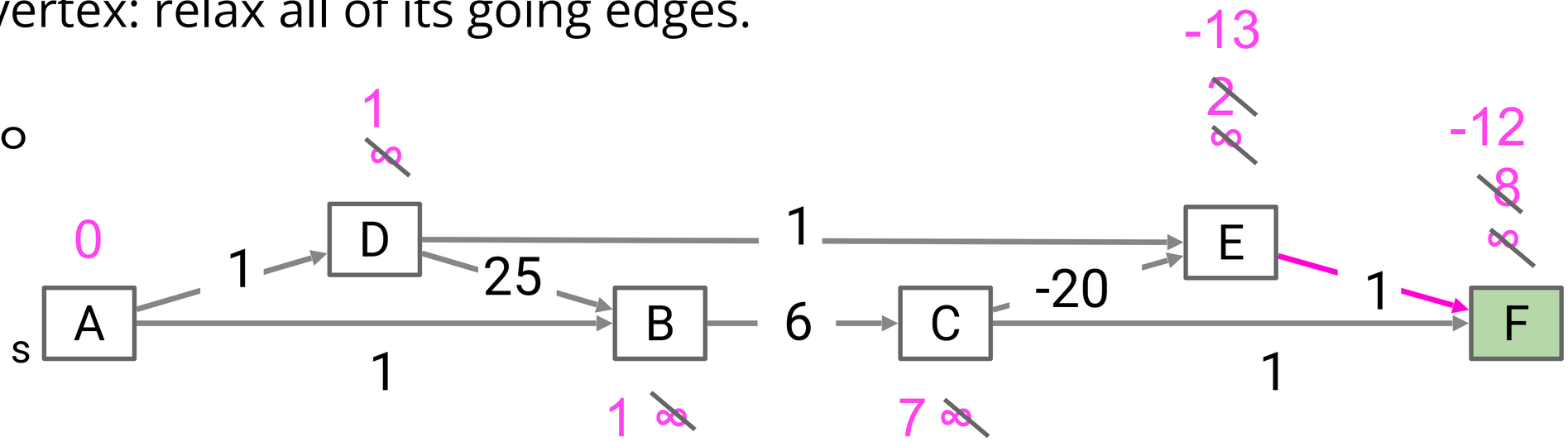
Fringe: [~~A~~, ~~D~~, ~~B~~, ~~C~~, ~~E~~, F]

DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax all of its going edges.

	distTo	edgeTo
A	0	-
B	1	A
C	7	B
D	1	A
E	-13	C
F	-12	E



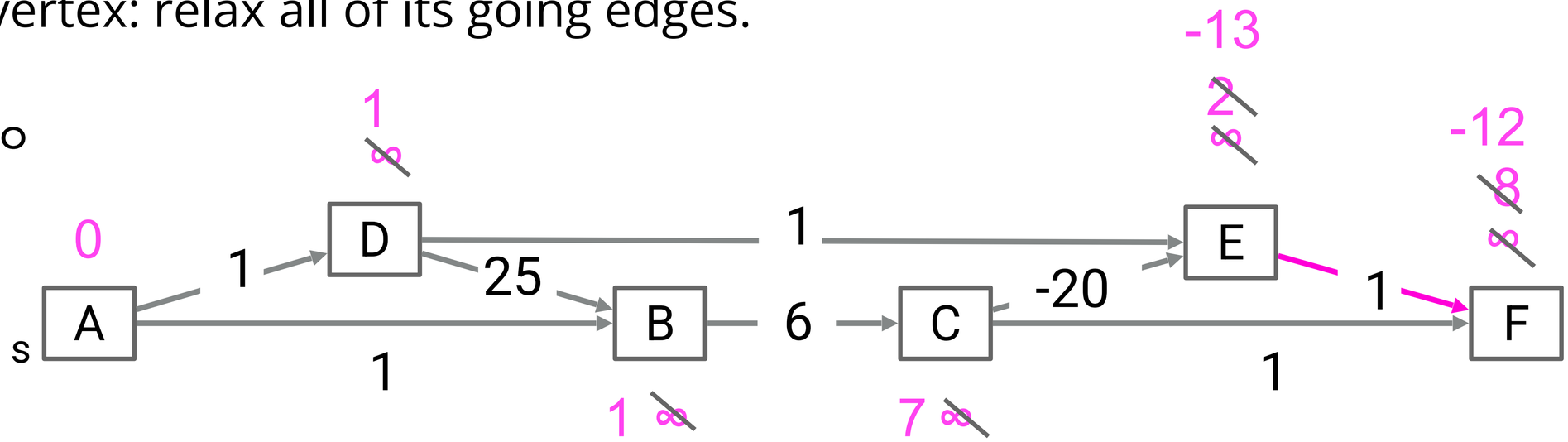
Fringe: [~~A~~, ~~D~~, ~~B~~, ~~C~~, ~~E~~, F]

DAG SPT Algorithm

Visit vertices in topological order.

- When we visit a vertex: relax all of its going edges.

	distTo	edgeTo
A	0	-
B	1	A
C	7	B
D	1	A
E	-13	C
F	-12	E



A DAG always has a guaranteed source and sink.

The last node will always be a sink, so we're done.

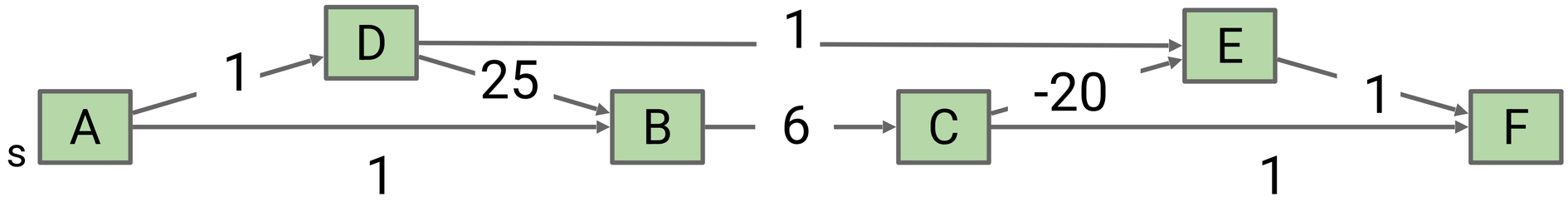
Fringe: [~~A~~, ~~D~~, ~~B~~, ~~C~~, ~~E~~, ~~F~~]

Runtime

- First: We have to find a topological order, e.g. ADBCEF. Runtime is O(V + E).
- Second: We have to visit all the vertices in topological order, relaxing all edges as we go. Runtime for step 2 is also O(V + E).

Occasional question: why isn't it O(V*E)? We're relaxing all edges from each vertex.

- Keep in mind that E is the *total* number of edges in the entire graph, not the number of edges per vertex.
  Example: for the graph below, E = 8.

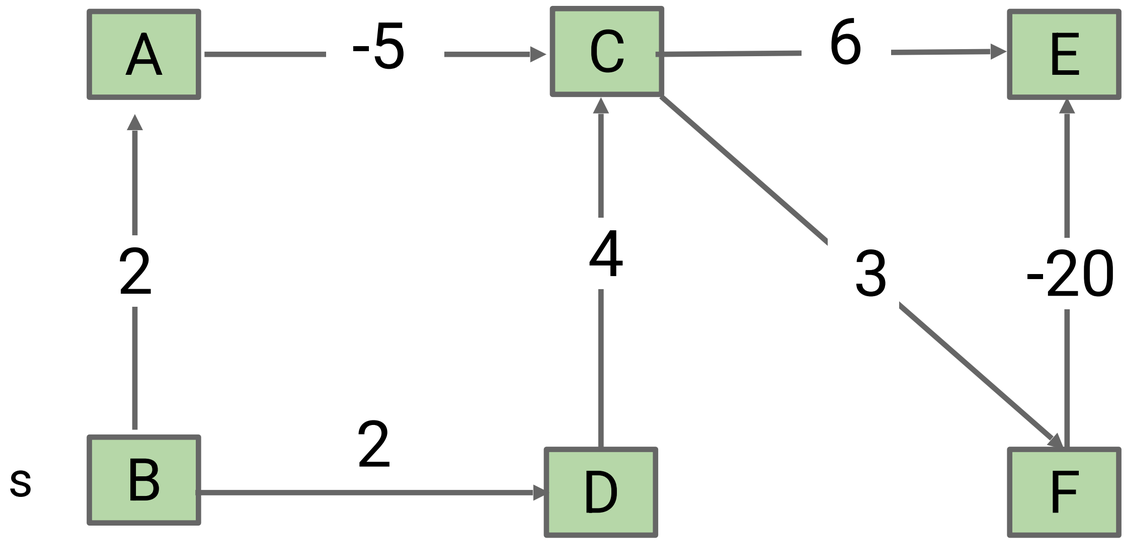


DAG algos

Problem	Problem Description	Solution	Efficiency
topological sort	Find an ordering of vertices that respects edges of our DAG.	DFS from source nodes	O(V+E) time Θ(V) space
DAG shortest paths	Find a SPT on a DAG. Negative weights OK.	topological sort, then visit vertices in that order and relax	O(V+E) time Θ(V) space

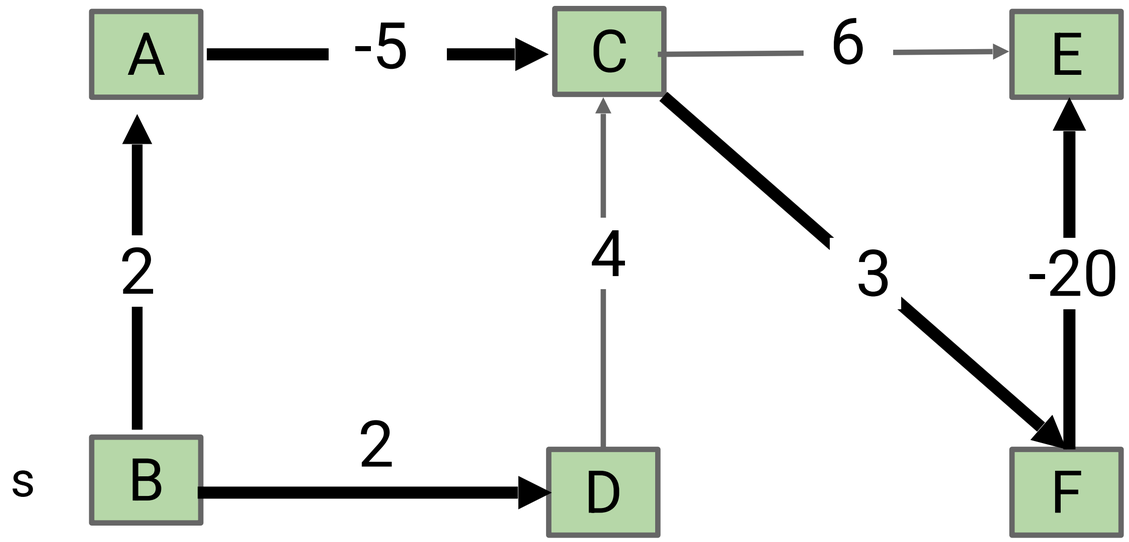
Worksheet time!

Run our algorithm to find the SPT of this DAG from s. What is distTo and edgeTo?



Worksheet answers

Run our algorithm to find the SPT of this DAG from s. What is distTo and edgeTo?

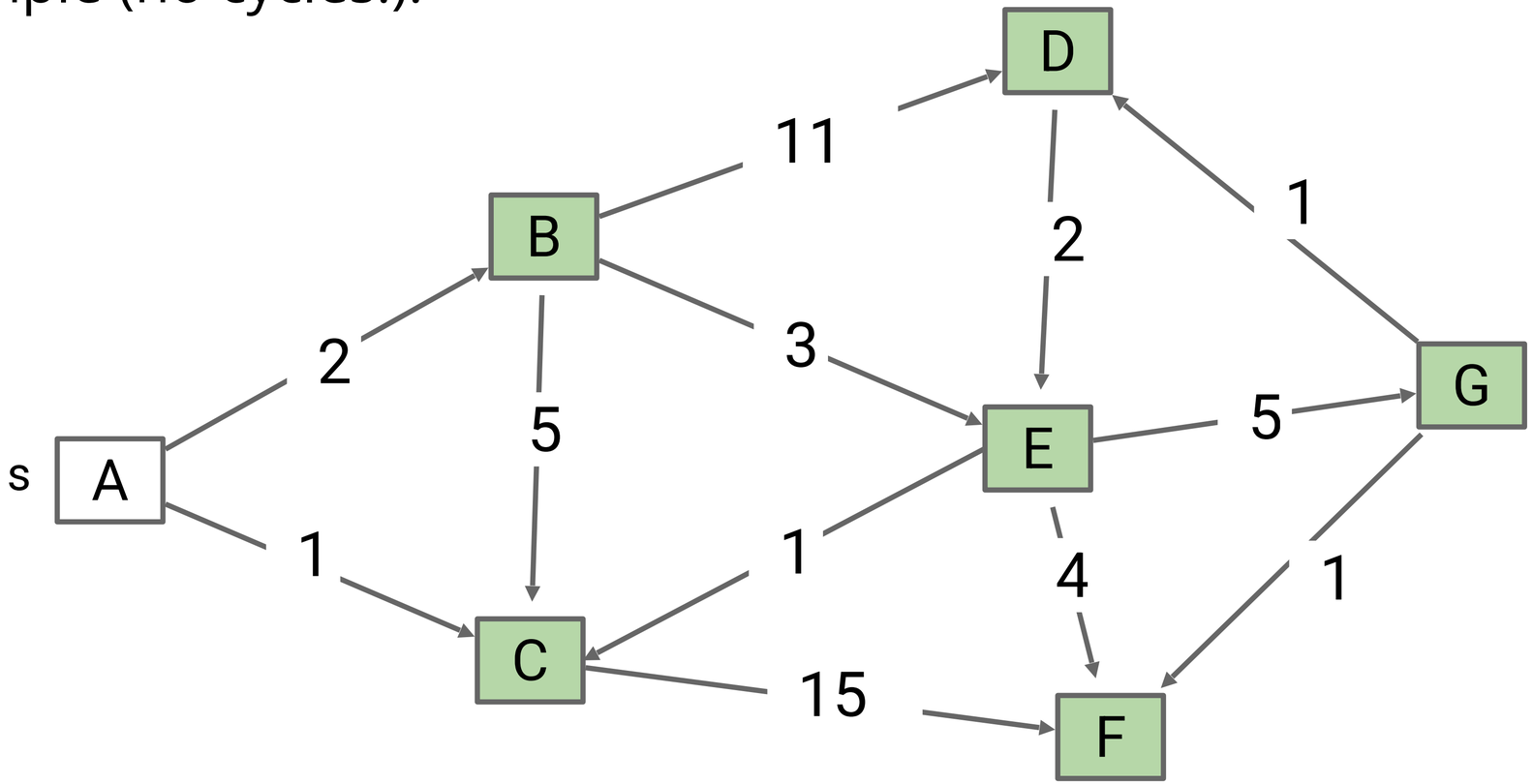


	distTo	edgeTo
A	2	B
B	0	-
C	-3	A
D	2	B
E	-20	F
F	0	C

Longest Paths

The Longest Paths Problem

Consider the problem of finding the longest path tree (LPT) from s to every other vertex. The path must be simple (no cycles!).

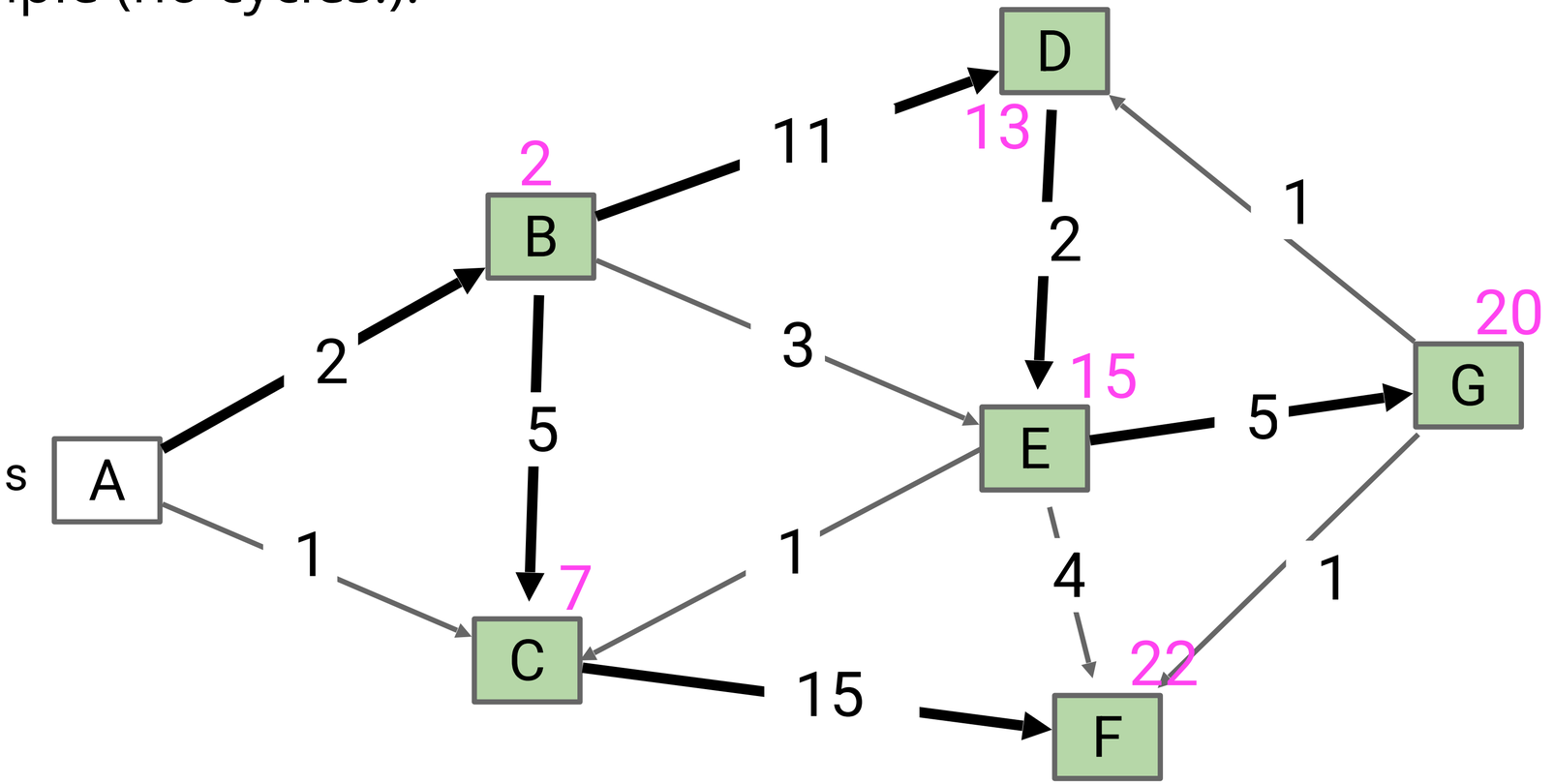


The Longest Paths Problem

Consider the problem of finding the longest path tree (LPT) from s to every other vertex. The path must be simple (no cycles!).

Some surprising facts:

- Best known algorithm is exponential (extremely bad).
- Perhaps the most important unsolved problem in mathematics.

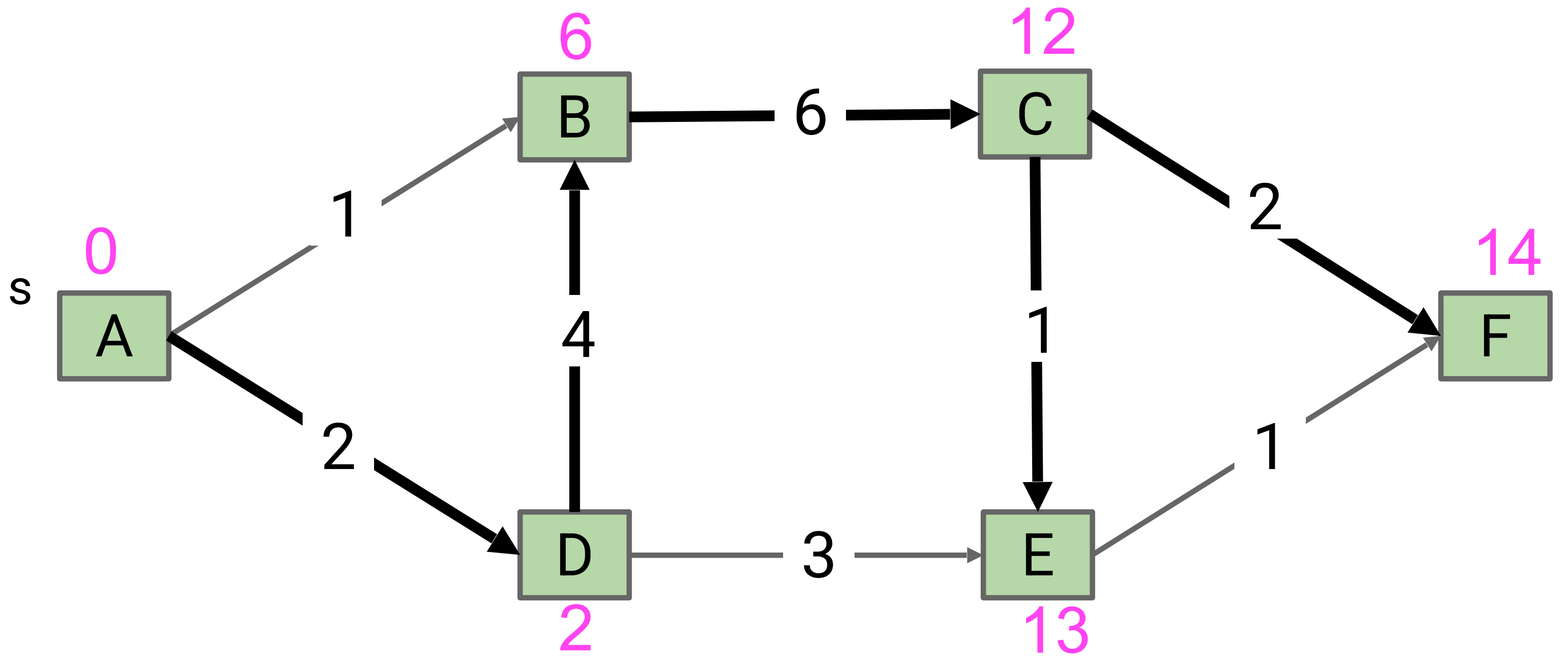


The Longest Paths Problem on DAGs

(Worksheet time!)

Difficult challenge for you:

- Solve the LPT problem on a directed acyclic graph.
- Algorithm must be O(E + V) runtime.



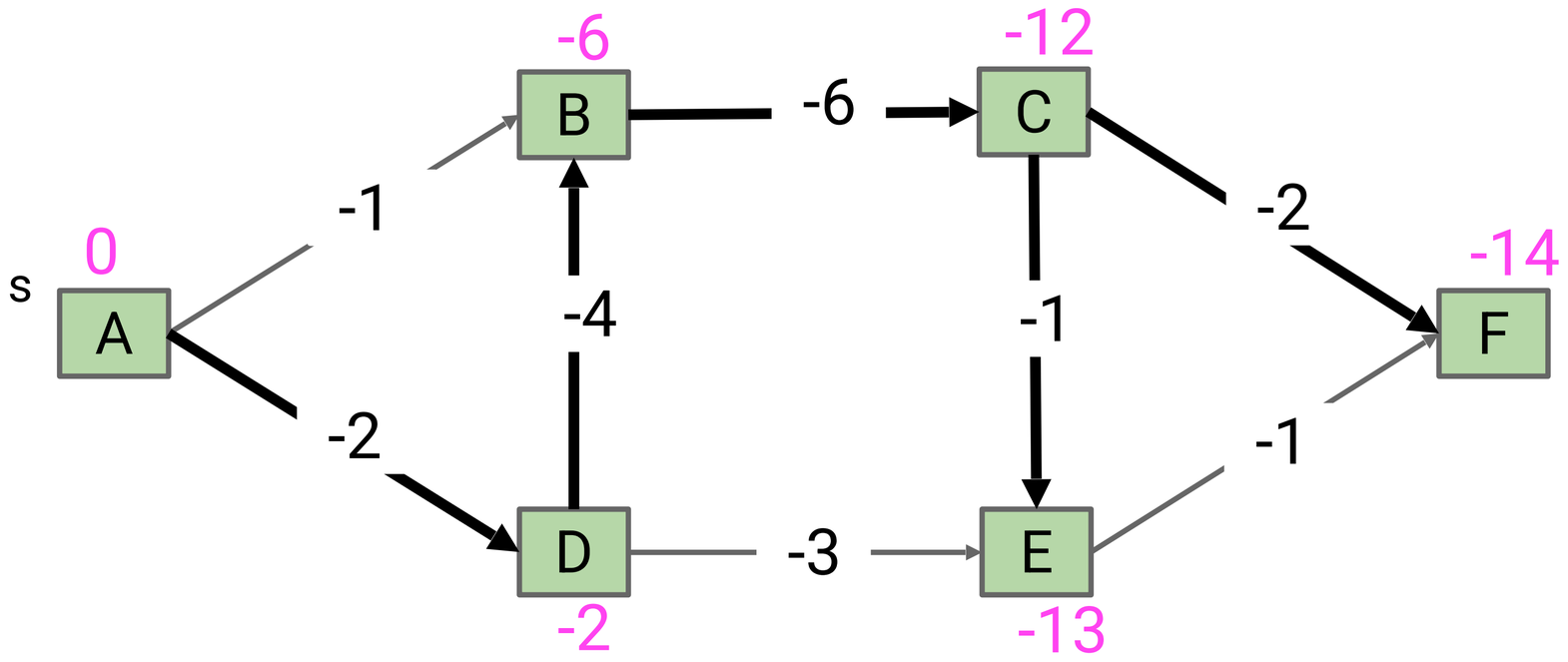
Hint: How can we make this problem look like the problem we just solved?

The Longest Paths Problem on DAGs

(Worksheet answers)

DAG LPT solution for graph G:

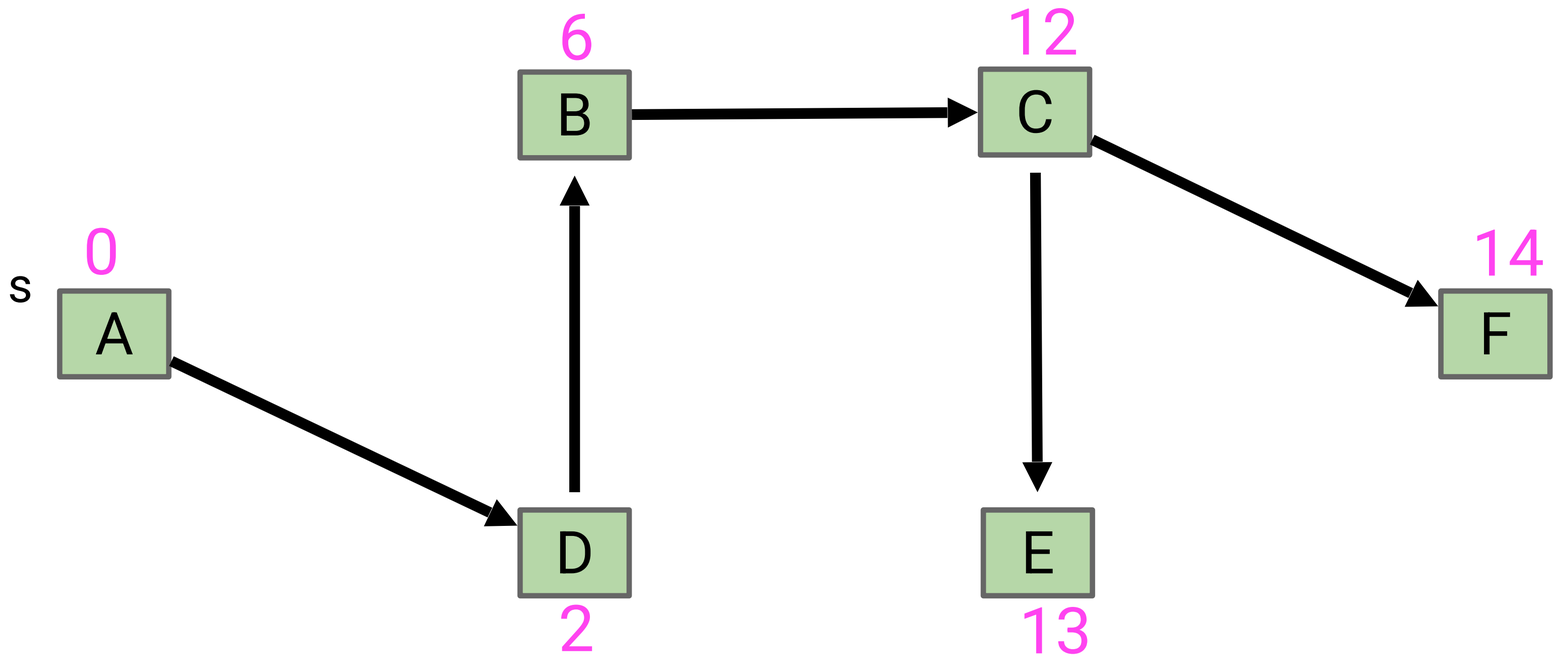
- Form a new copy of the graph G' with signs of all edge weights **flipped**.
- Run DAGSPT on G' yielding result X.
- Flip signs of all values in X.distTo. X.edgeTo is already correct.



The Longest Paths Problem on DAGs

DAG LPT solution for graph G:

- Form a new copy of the graph G’ with signs of all edge weights flipped.
- Run DAGSPT on G’ yielding result X.
- **Flip signs** of all values in X.distTo. X.edgeTo is already correct.



DAG algos

Problem	Problem Description	Solution	Efficiency
topological sort	Find an ordering of vertices that respects edges of our DAG.	DFS from source nodes	O(V+E) time Θ(V) space
DAG shortest paths	Find a SPT on a DAG. Negative weights OK.	topological sort, then visit vertices in that order and relax	O(V+E) time Θ(V) space
longest paths	Find a longest paths tree on a graph.	No good known solution exists.	?
DAG longest paths	Find a longest paths tree on a DAG.	flip signs, then DAG shortest paths, flip signs again	O(V+E) time Θ(V) space

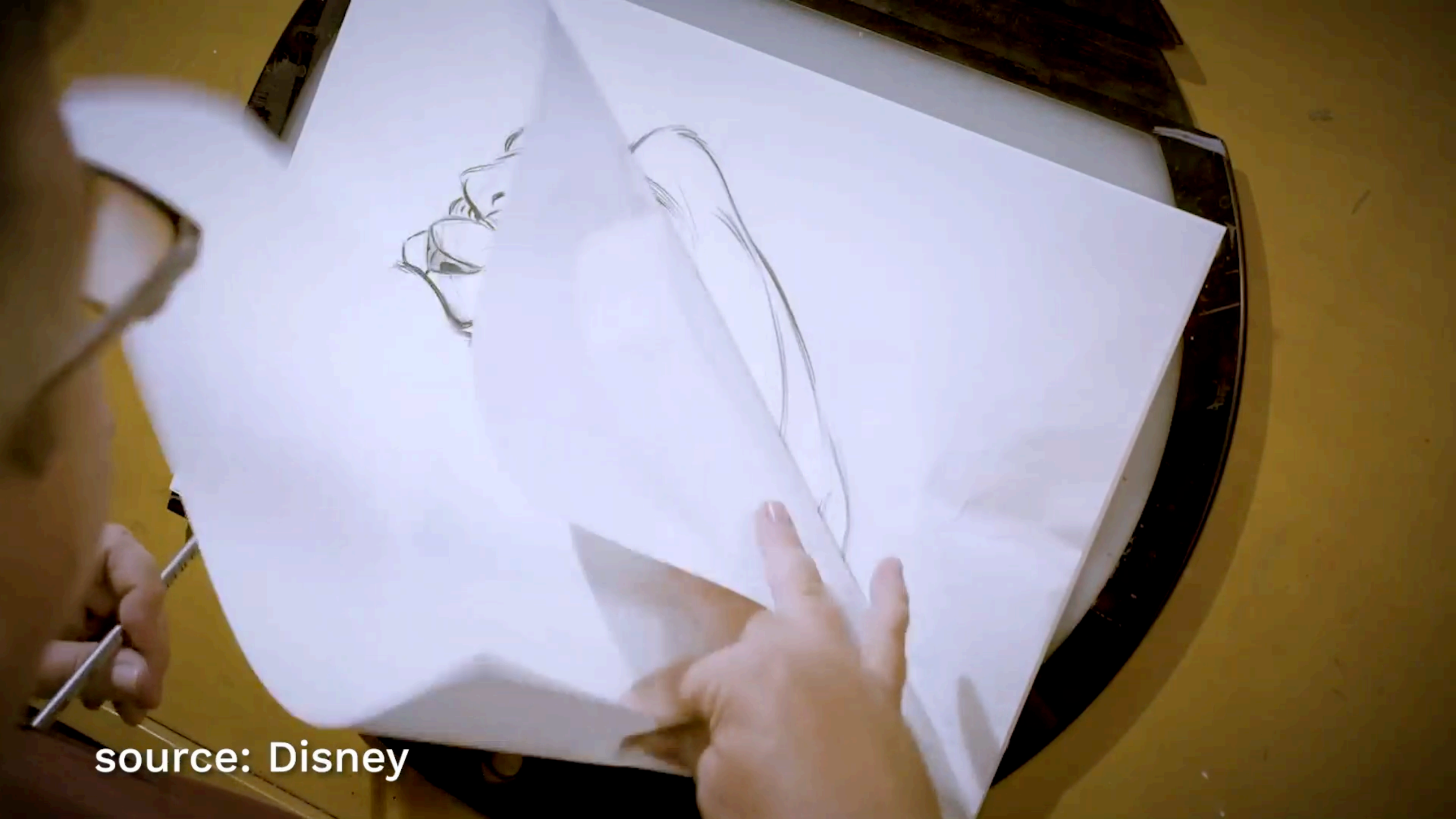
Why? It's just
DAG shortest paths.
Which is just topological sort.
Which is just DFS.

DAGs in my research

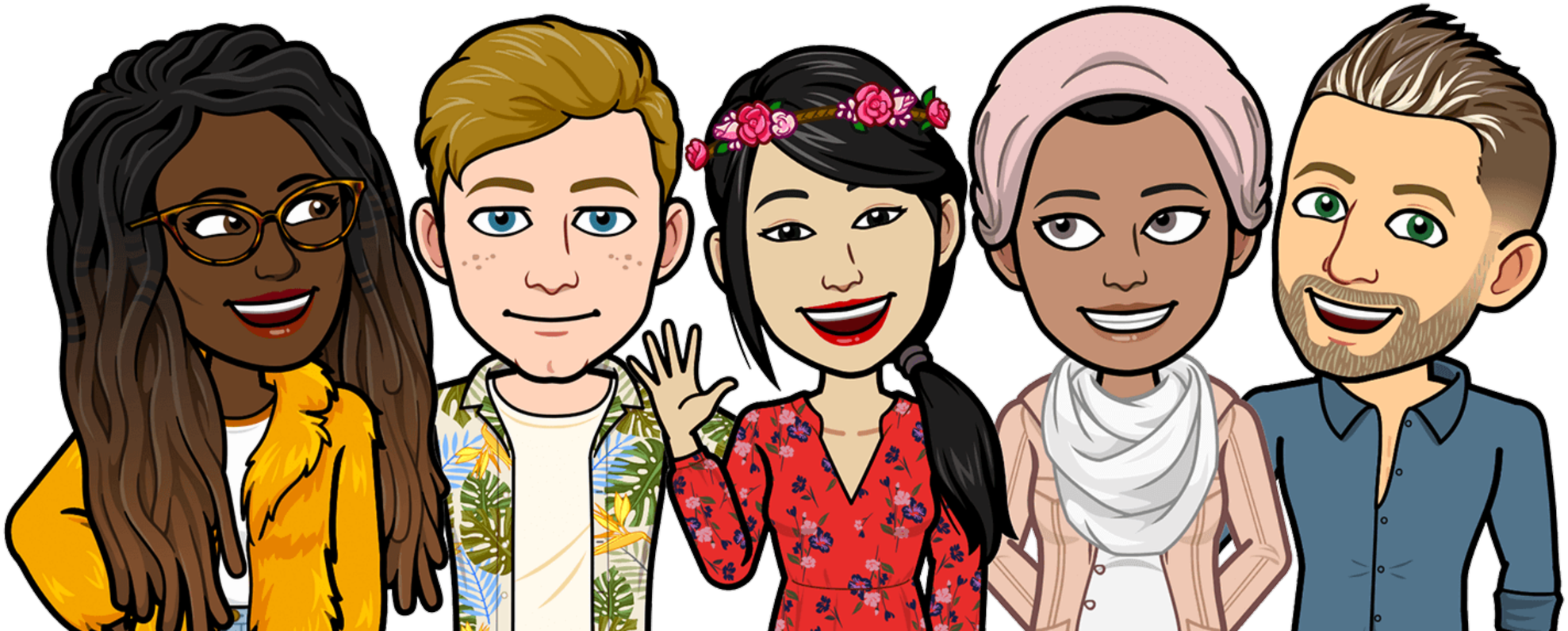
Automated Accessory Rigs for Layered 2D Character Illustrations

Jingyi Li • Wilmot Li • Sean Follmer • Maneesh Agrawala


source: RossDraws, YouTube


source: Disney

Mix & match character creation



source: Bitmoji

Home / Collections / Open Peeps /

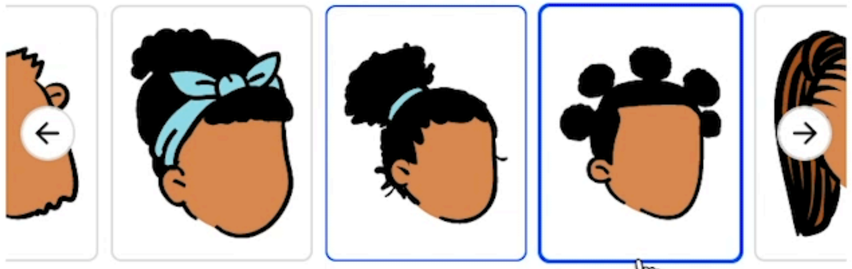
Standing by Pablo Stanley

Save Share Download

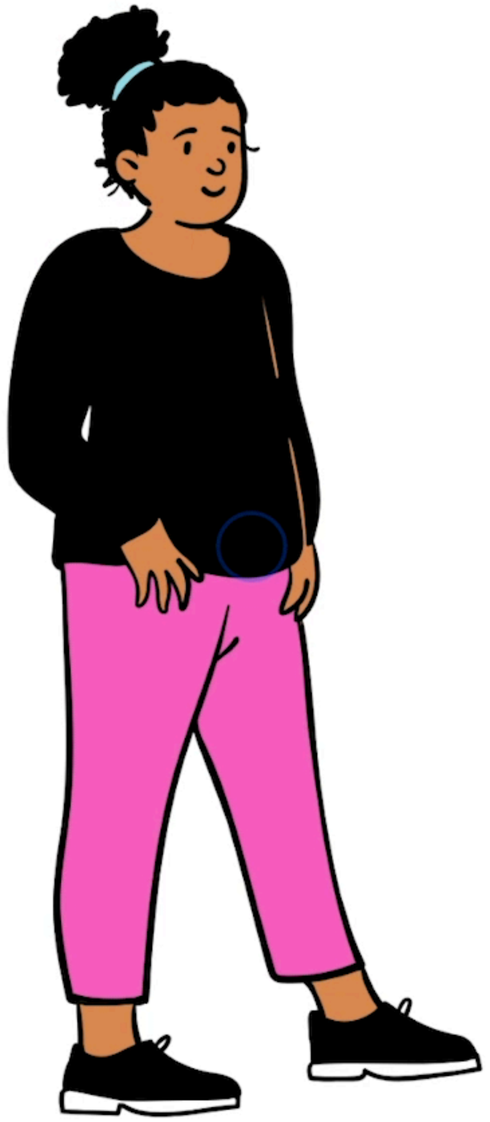
Skin

Standing See All

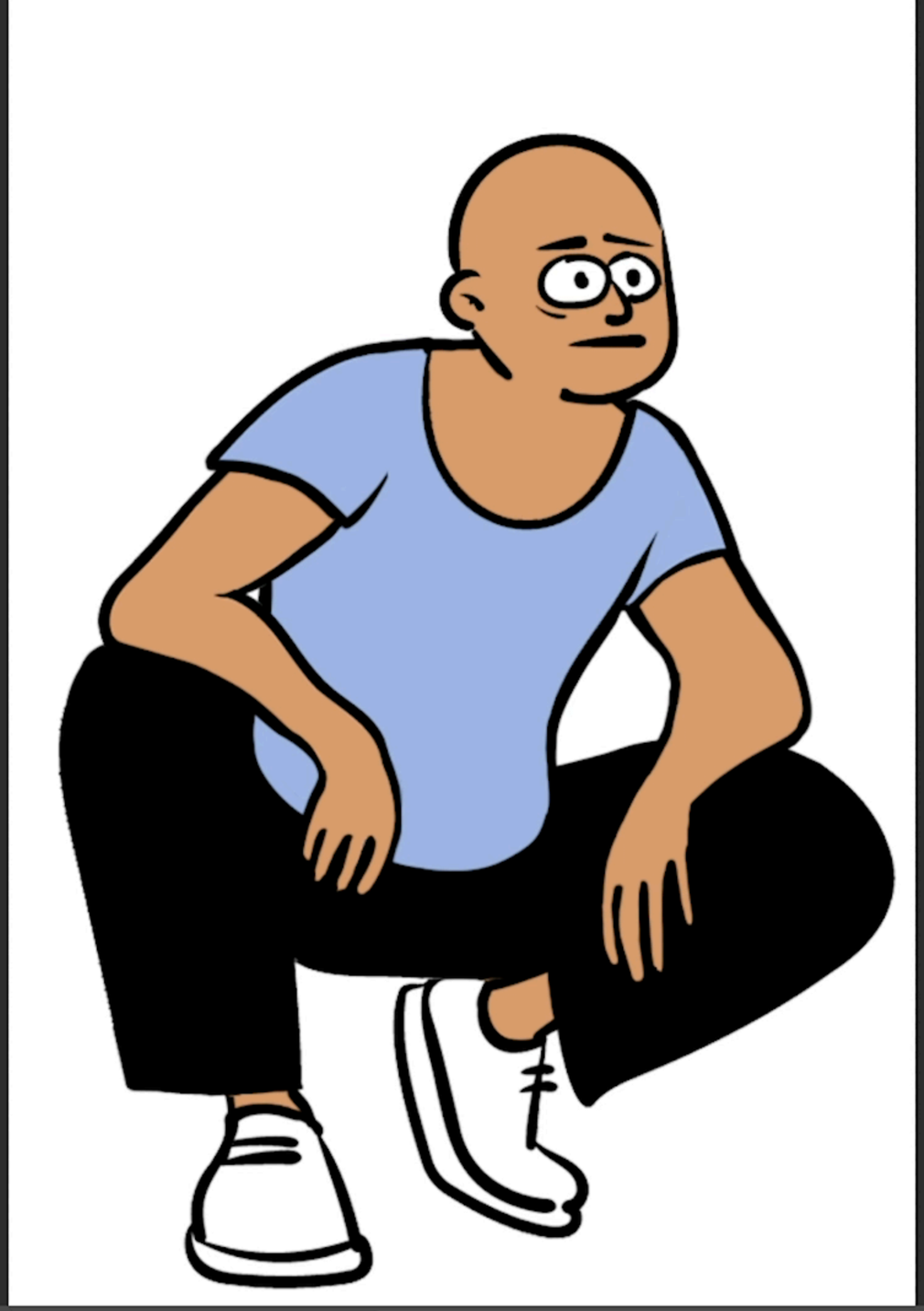
Head See All



Facial Hair See All



Characters are static



Kind

Normal Opacity: 100%

Lock: Fill: 100%

leather-jacket-frontarm

blue-shirt-frontarm

polo-shirt-frontarm

frontarm-base

leather-jacket-back

leather-jacket-front

blue-shirt-base

tank-shirt-base

polo-shirt-feature

polo-shirt-base

leather-jacket-backarm

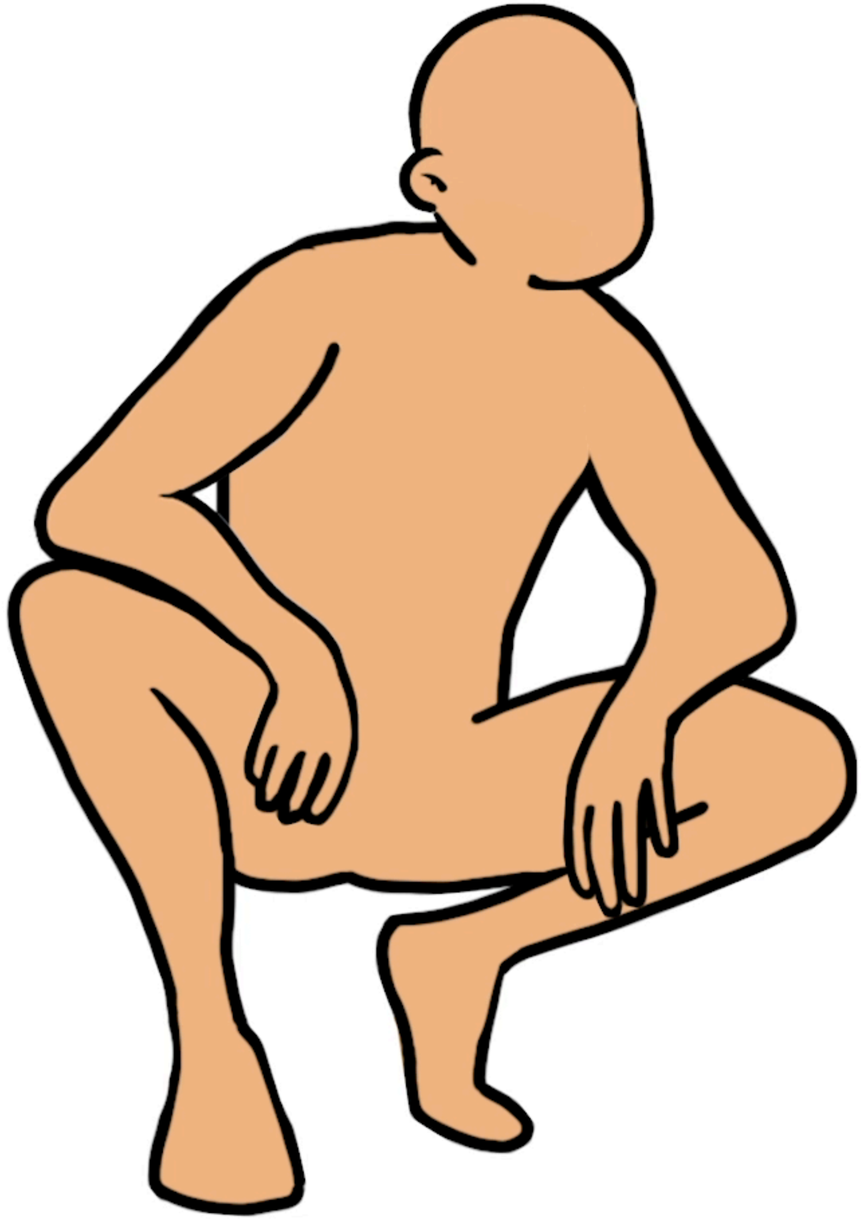
blue-shirt-backarm

polo-shirt-backarm

backarm-base

leg-shorts

Doc: 1.54M/88.3M

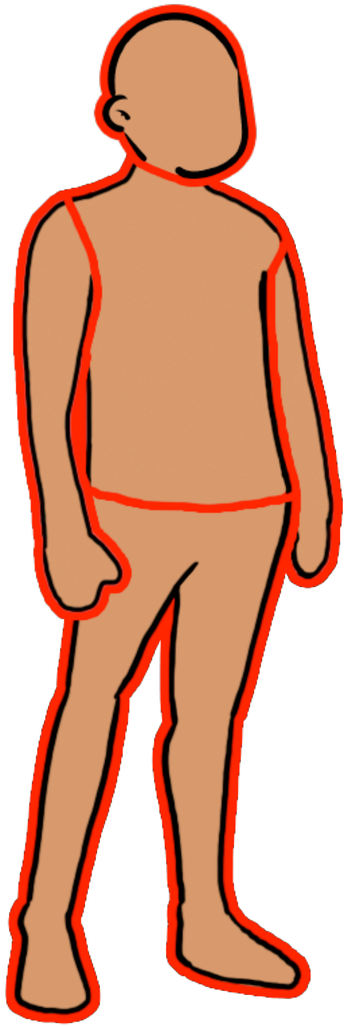


How can we automatically attach accessories to the body such that they deform with the body?

How can we automatically attach accessories to the body such that they deform with the body?

Construct & infer four types of constraints between layers for automated rigs: (1) occlusion, (2) at a single point, (3) along coincident boundaries, and (4) around a region of overlap.

Input



Body layers

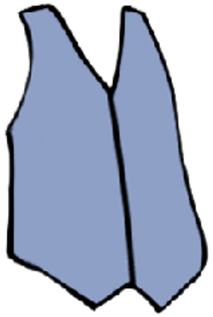
Accessory layers

Top layer

Bowtie

Front sleeve

Shirt collar



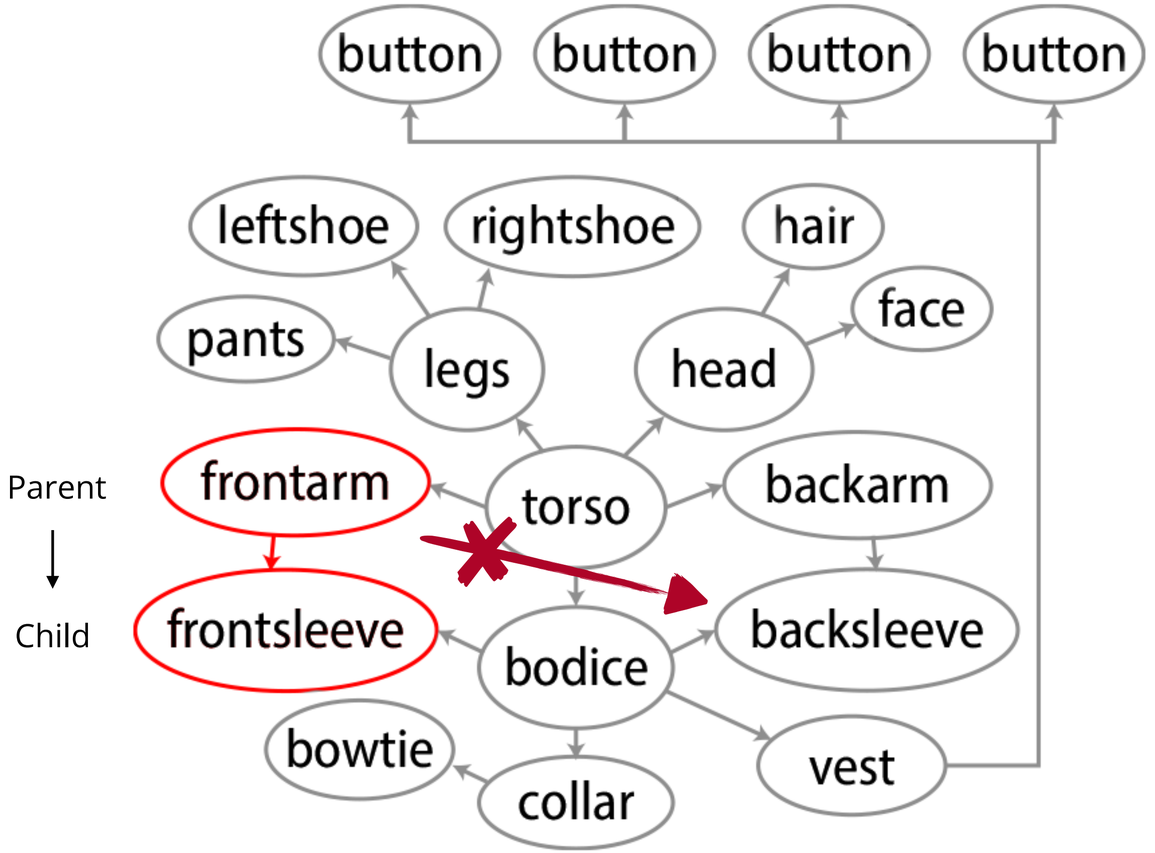
Vest

Shirt bodice

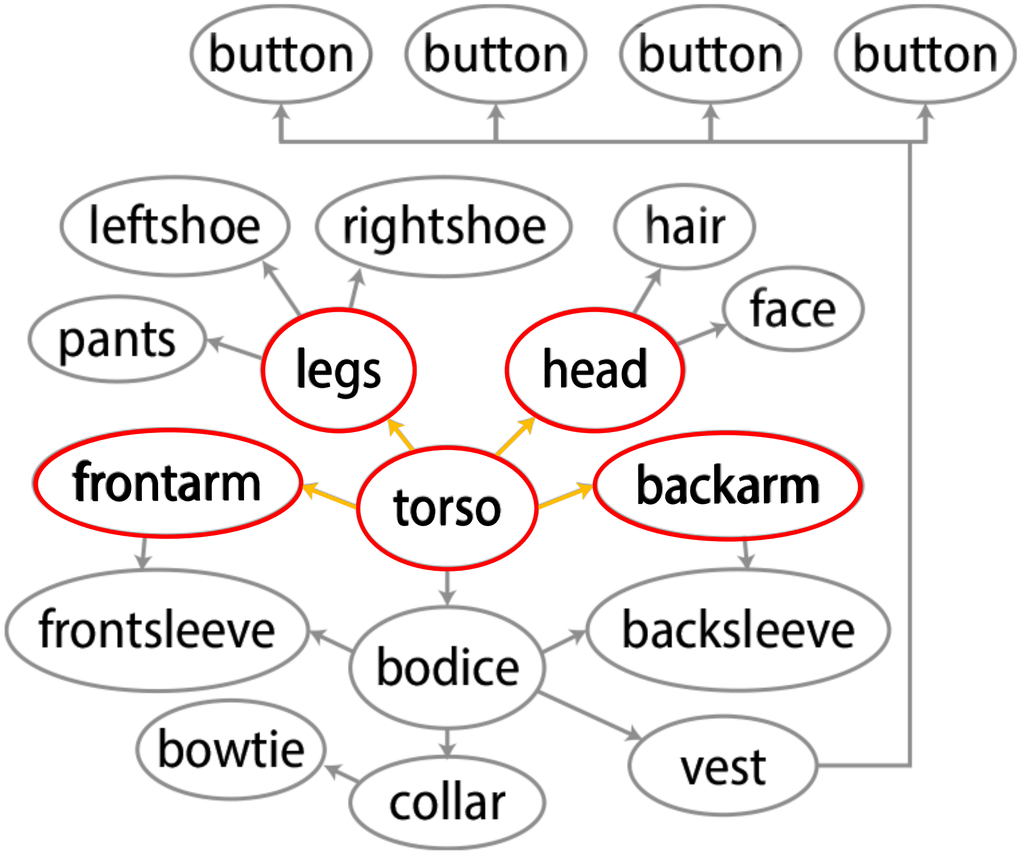
Back sleeve

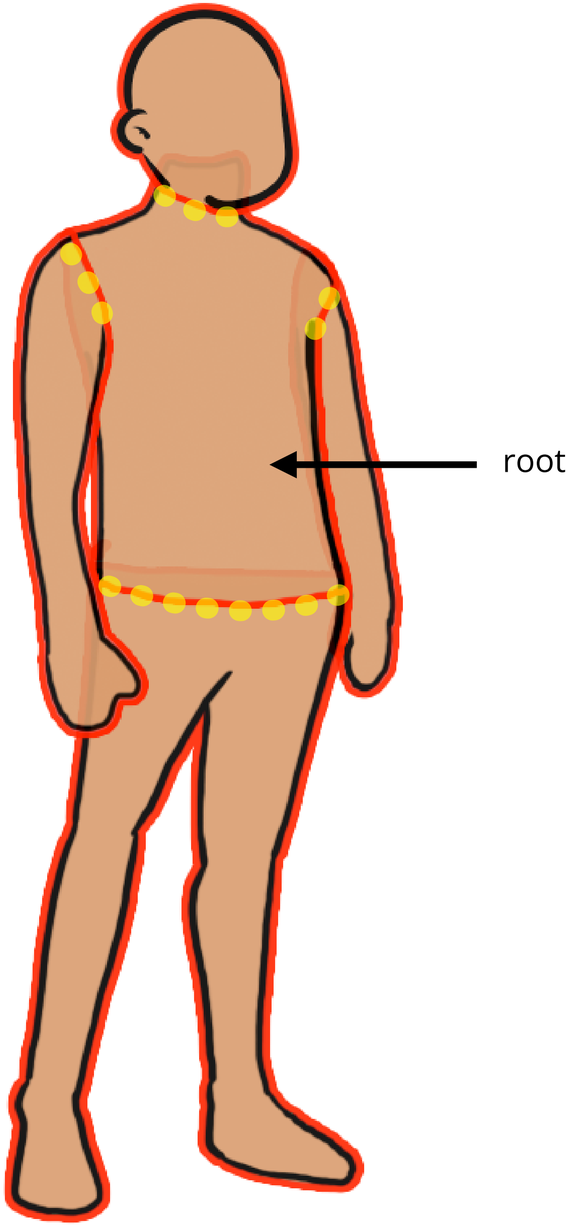
Bottom layer

DAG


button
button
button
button
leftshoe
rightshoe
hair
face
pants
legs
head
Parent
frontarm
torso
backarm
Child
frontsleeve
bodice
backsleeve
bowtie
collar
vest

DAG





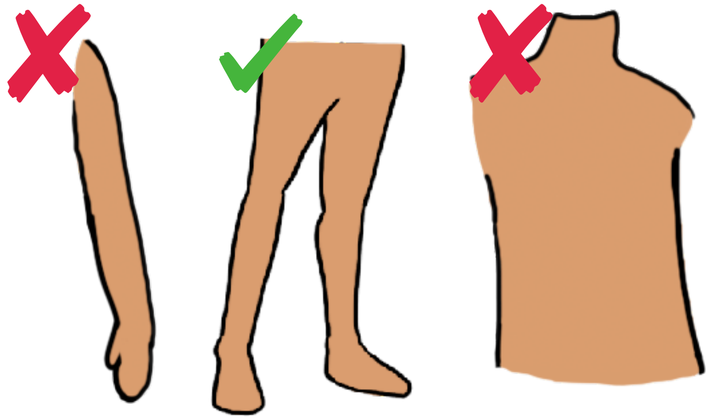
Body layers

For each accessory,

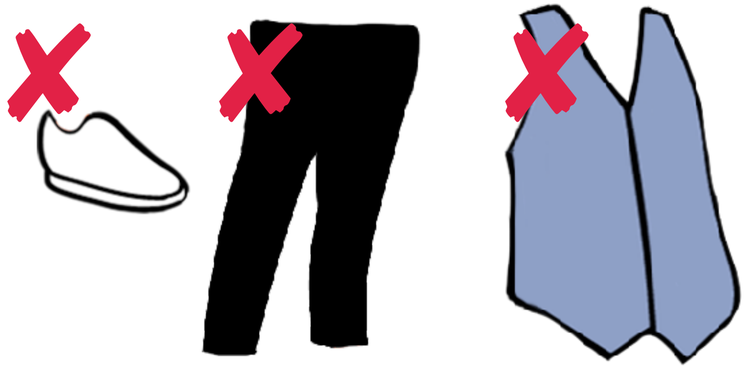
check overlap with every other layer

If just one overlap, make an edge

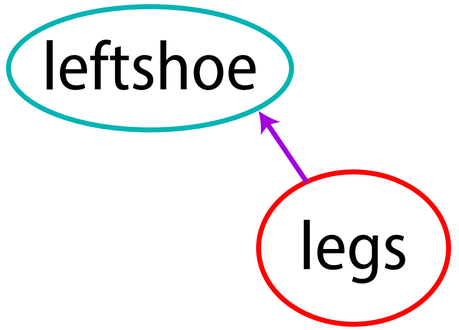
Target accessory (shoe)



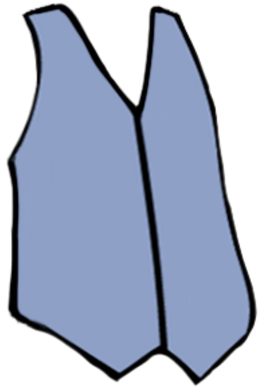
…



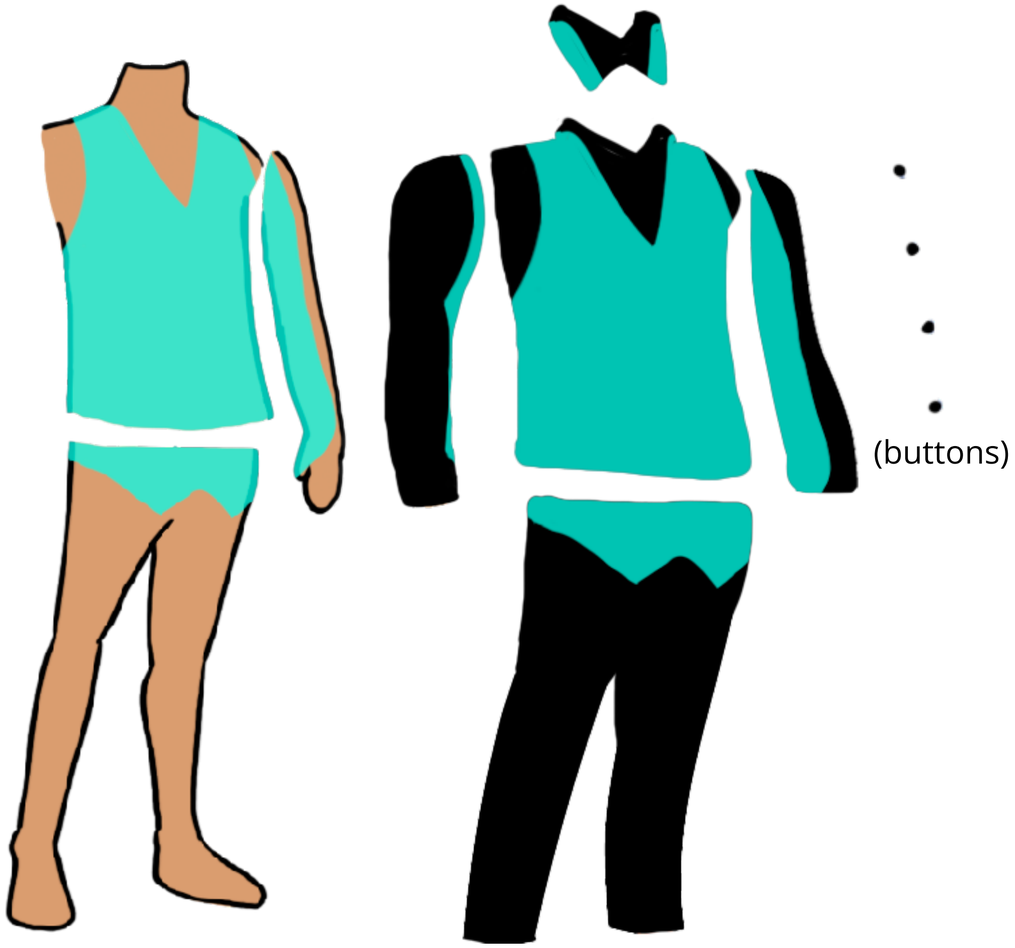
…



For each accessory, multiple overlaps with



Target accessory
(vest)

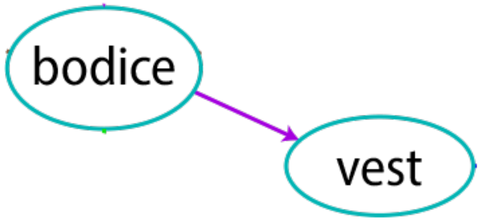


(buttons)

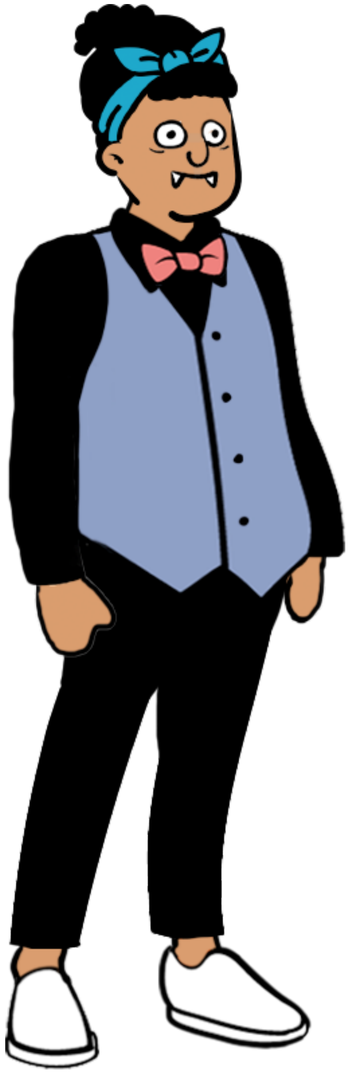
Within 15% of maximum
overlapping area

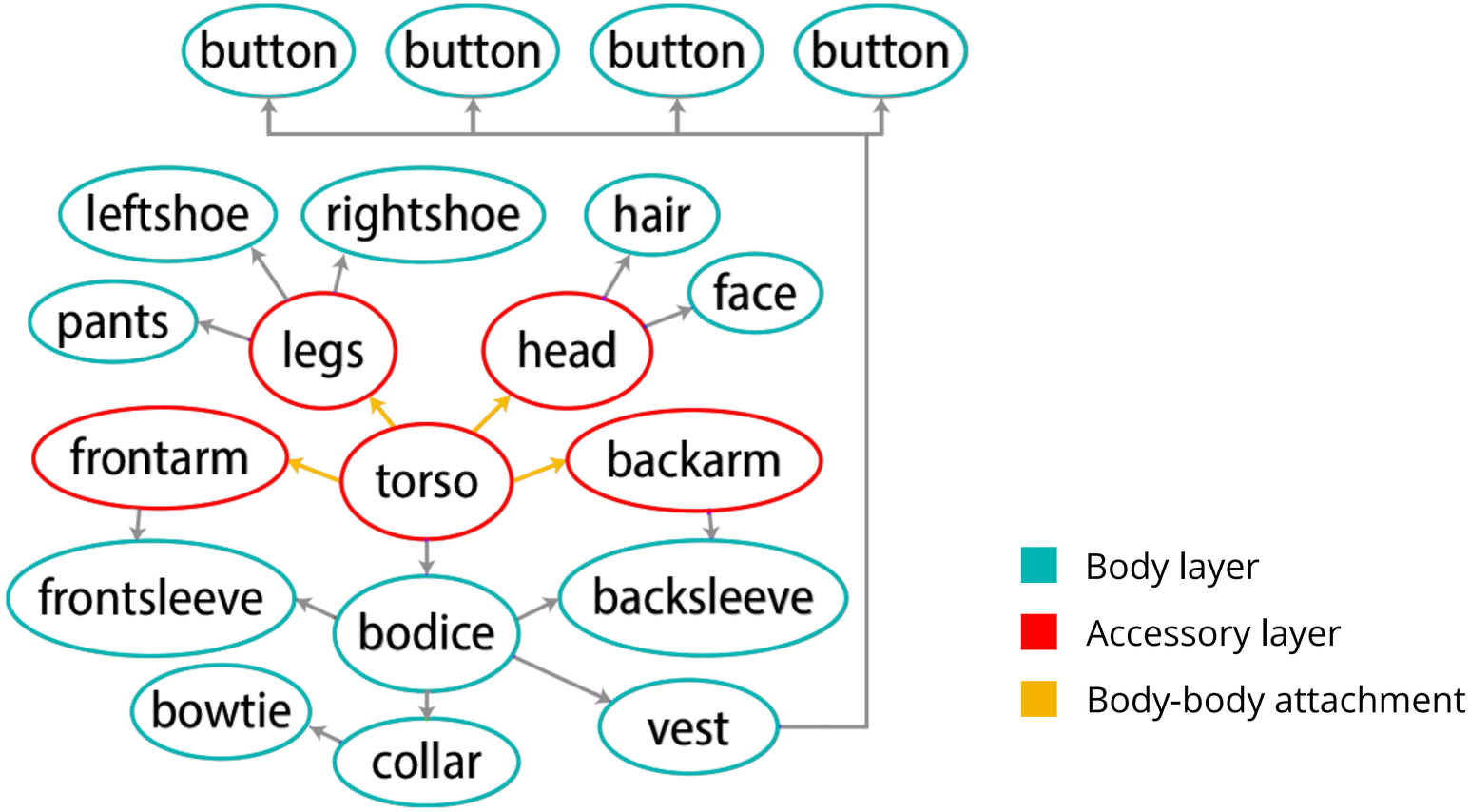
layer order

...
9. button
8. vest
7. shirt bodice ✓
6. shirt backsleeve
5. body torso
4. legs
...

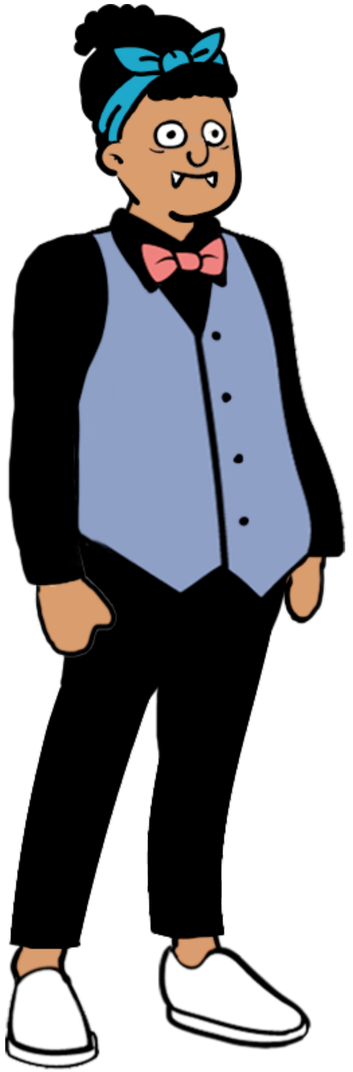


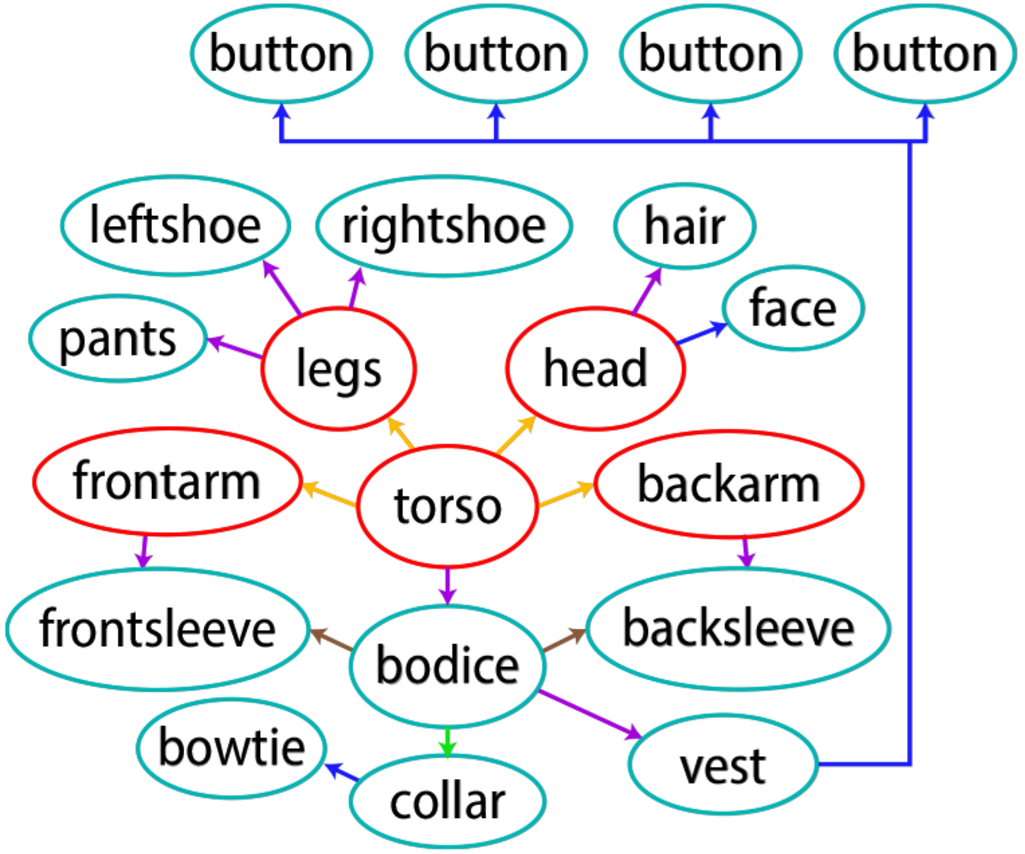
DAG




button
button
button
button
leftshoe
rightshoe
hair
face
pants
legs
head
frontarm
torso
backarm
frontsleeve
bodice
backsleeve
bowtie
collar
vest
Body layer
Accessory layer
Body-body attachment

Constraints



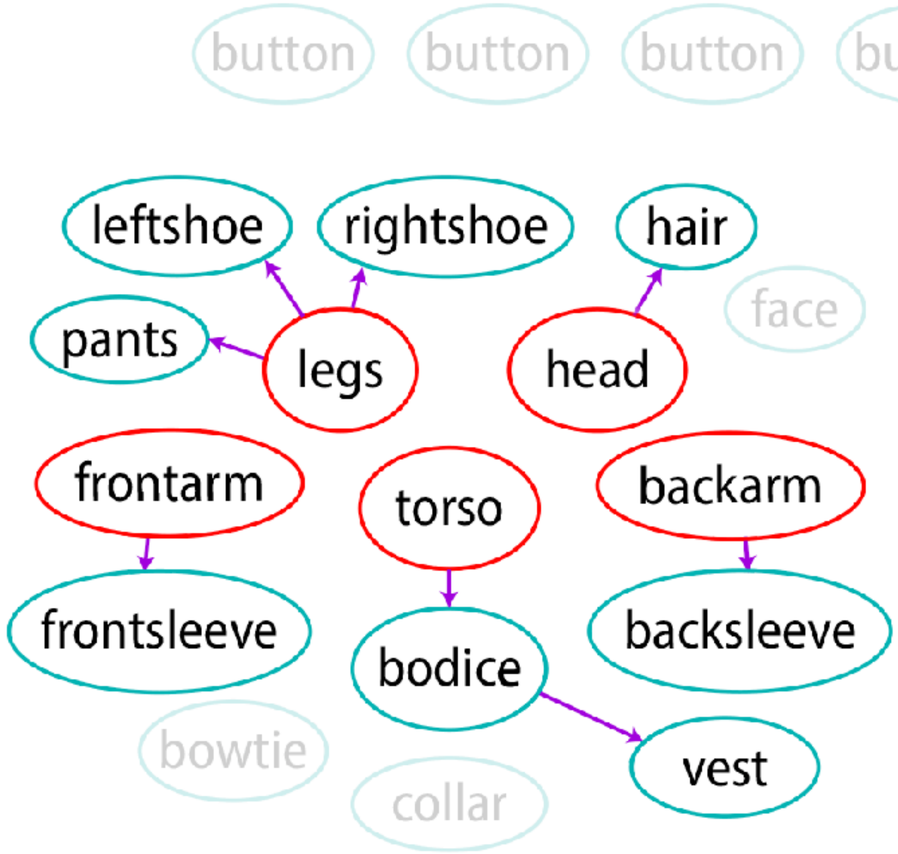


Constraint types

- Occlusion
- Single point
- Boundary
- Region

- Body layer
- Accessory layer
- Body-body attachment

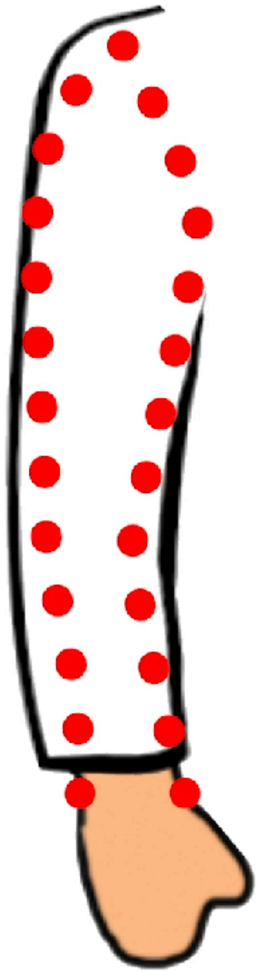
Constraint 1: Occlusion



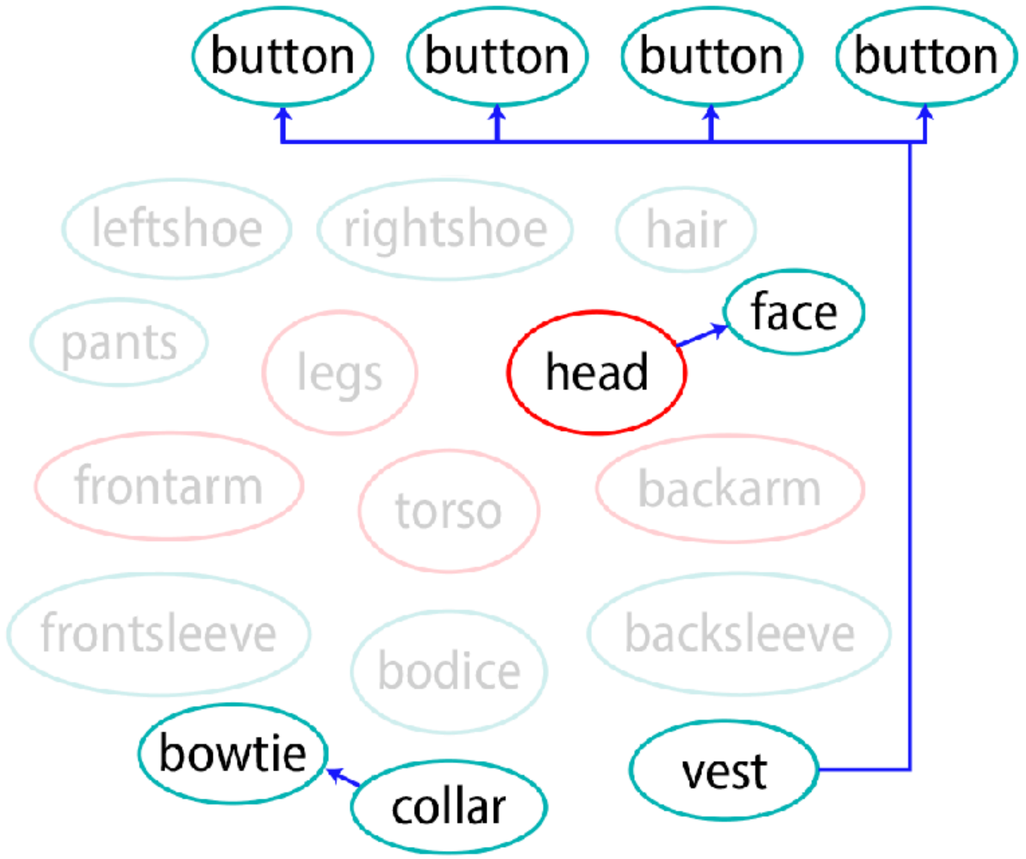
Constraint types

- Occlusion
- Single point
- Boundary
- Region

parent layer boundary

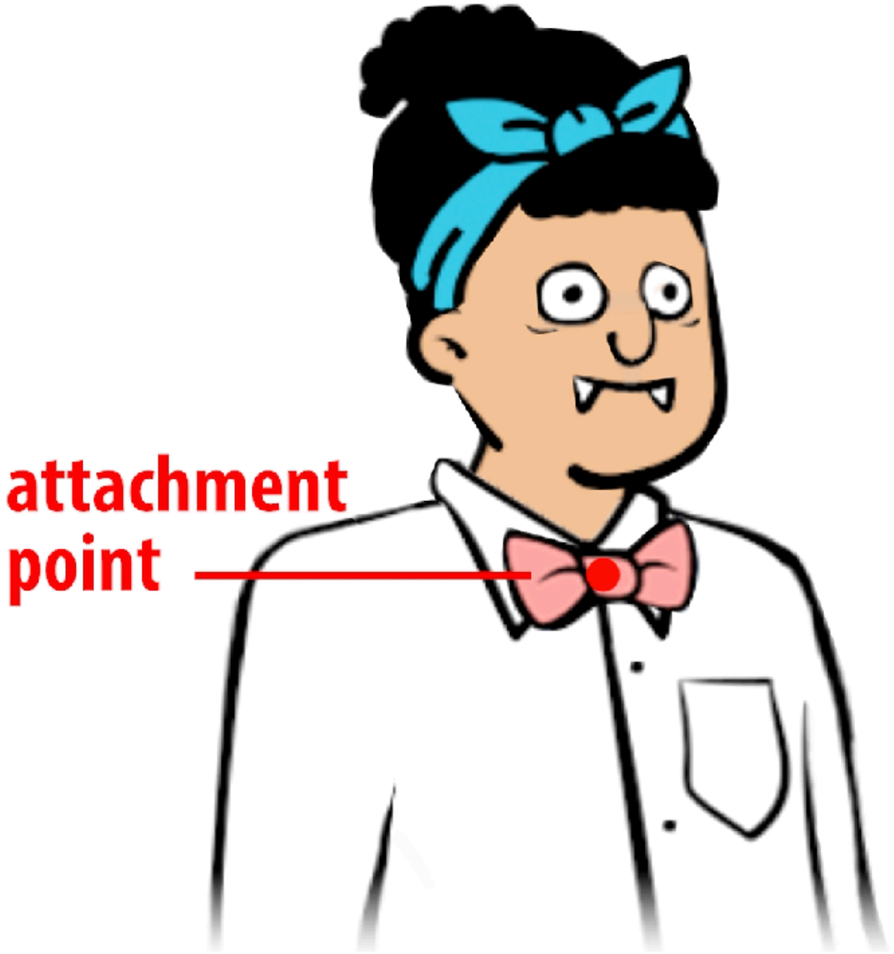


Constraint 2: Single point

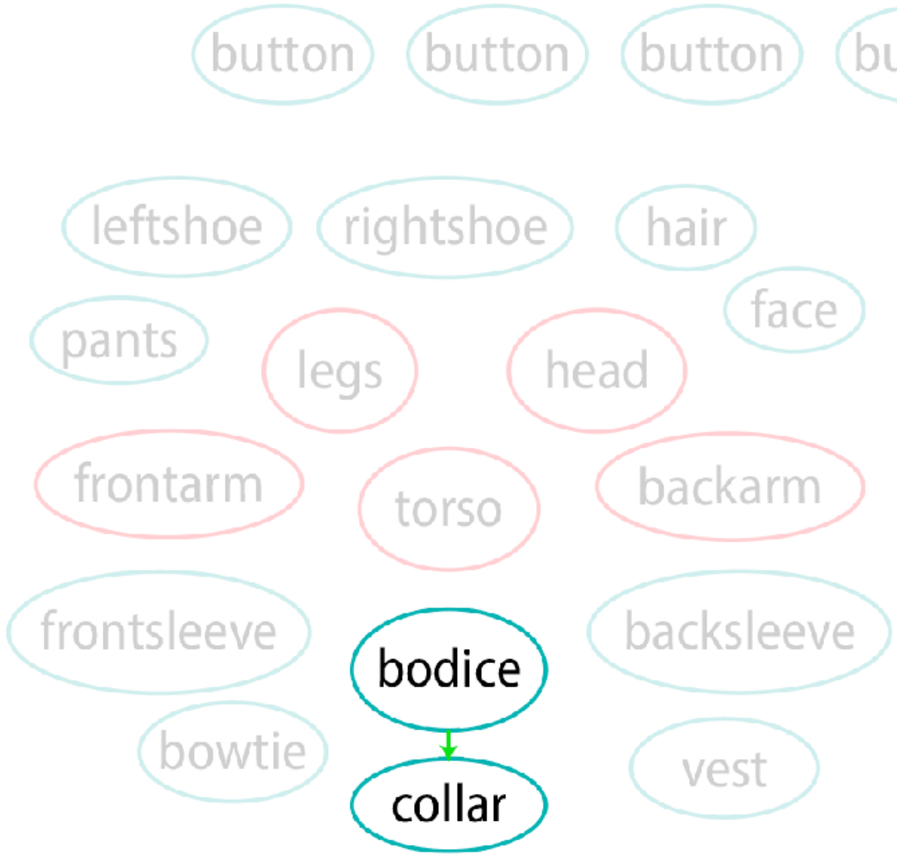


Constraint types

- Occlusion
- Single point
- Boundary
- Region

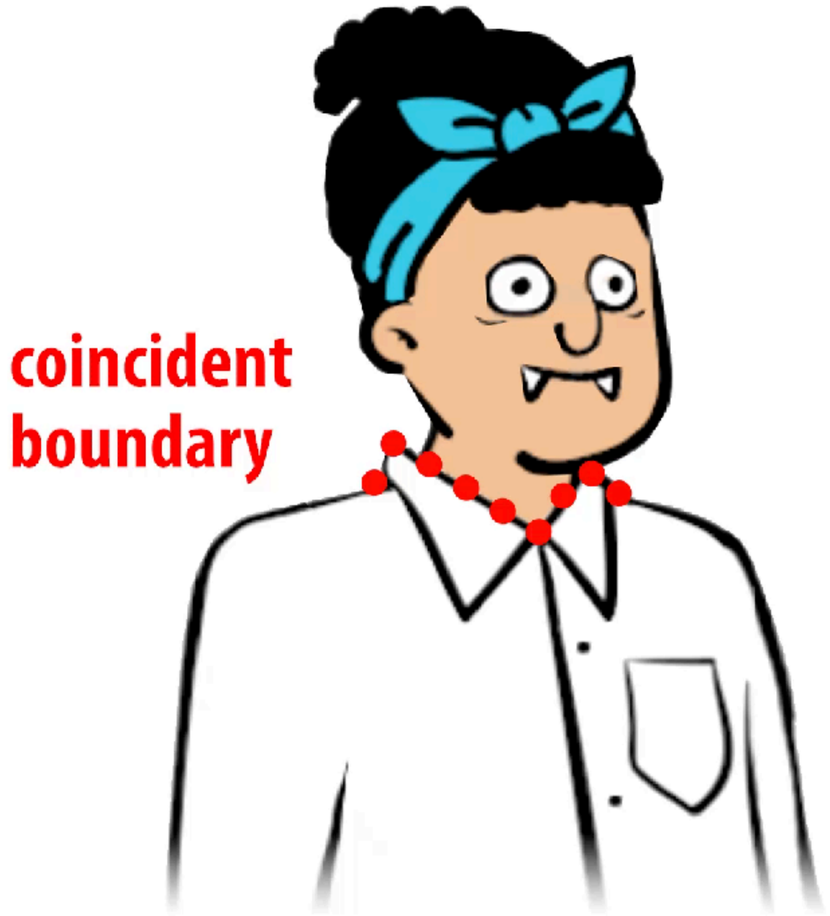


Constraint 3: Boundary

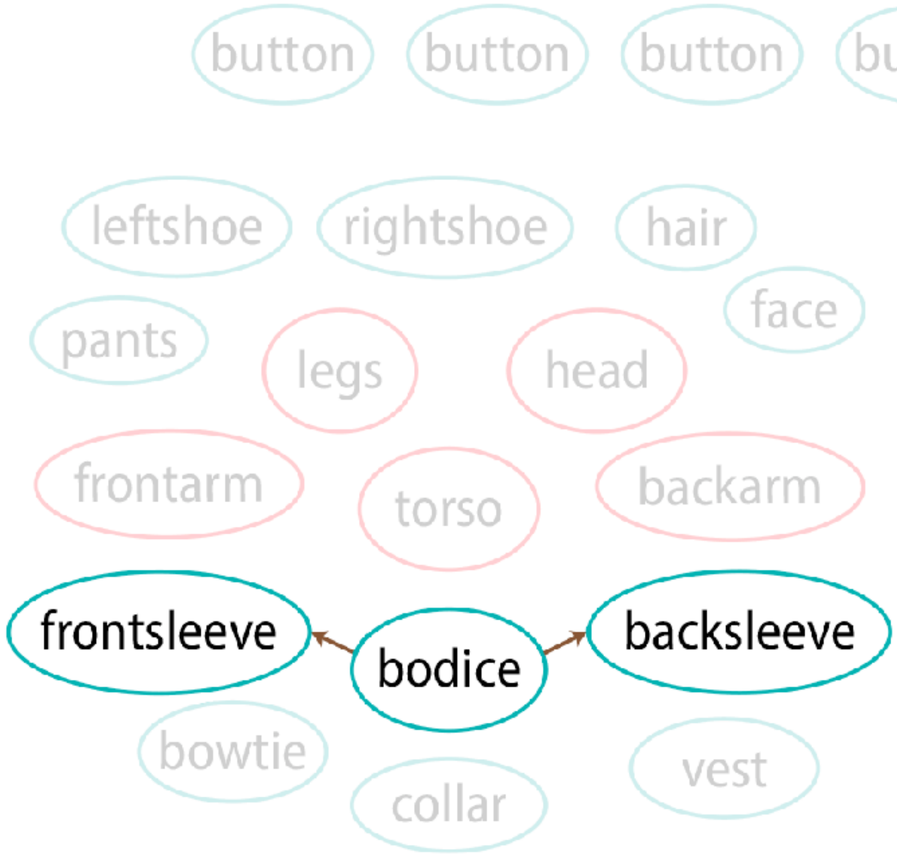


Constraint types

- Occlusion
- Single point
- Boundary
- Region

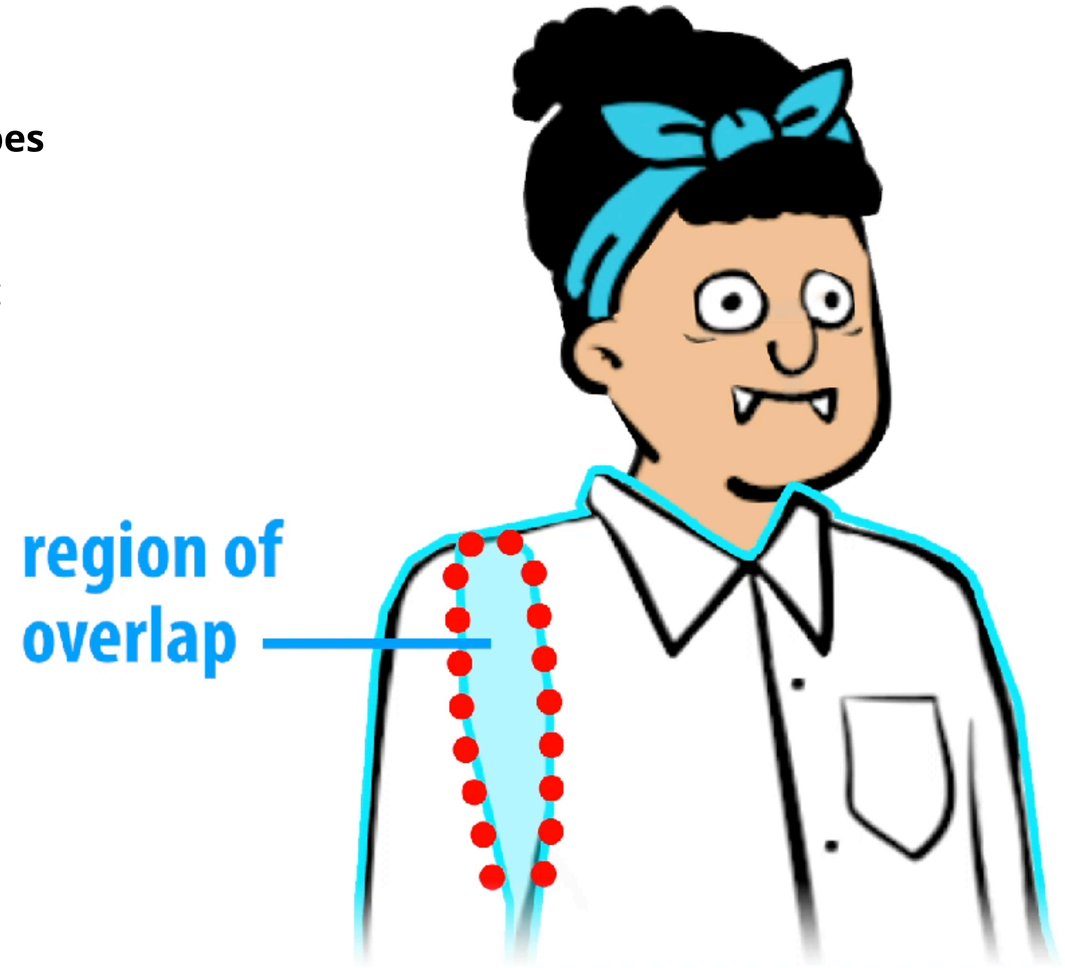


Constraint 4: Overlapping region



Constraint types

- Occlusion
- Single point
- Boundary
- Region

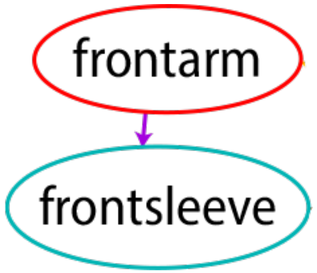


Constraint inference

For each DAG edge,

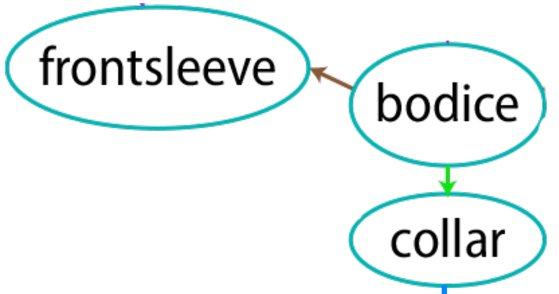
Body-accessory:

- try occlusion

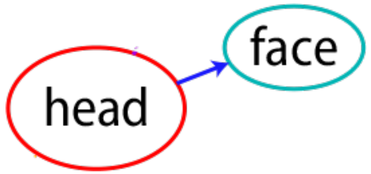


Accessory-accessory:

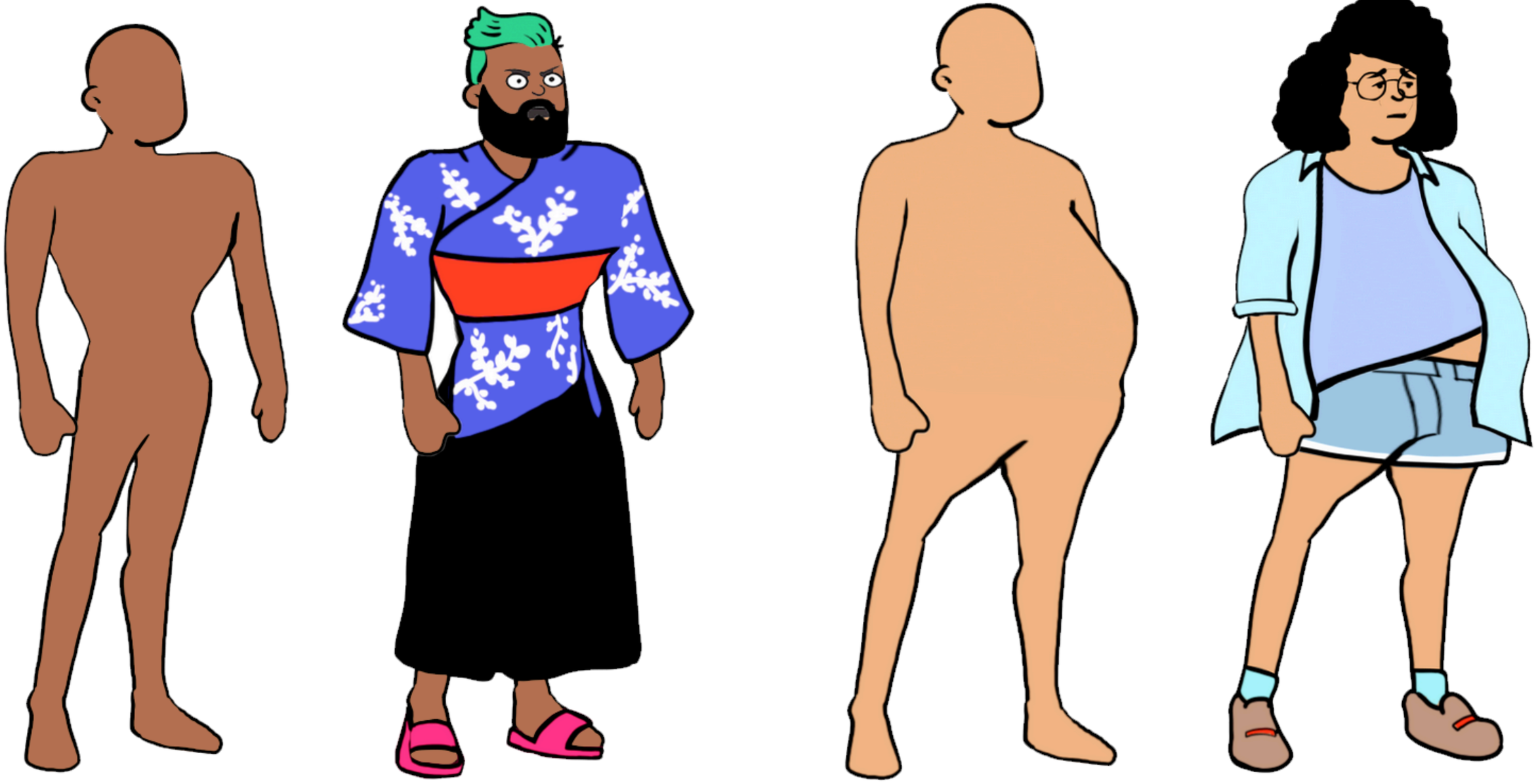
- try boundary & overlap

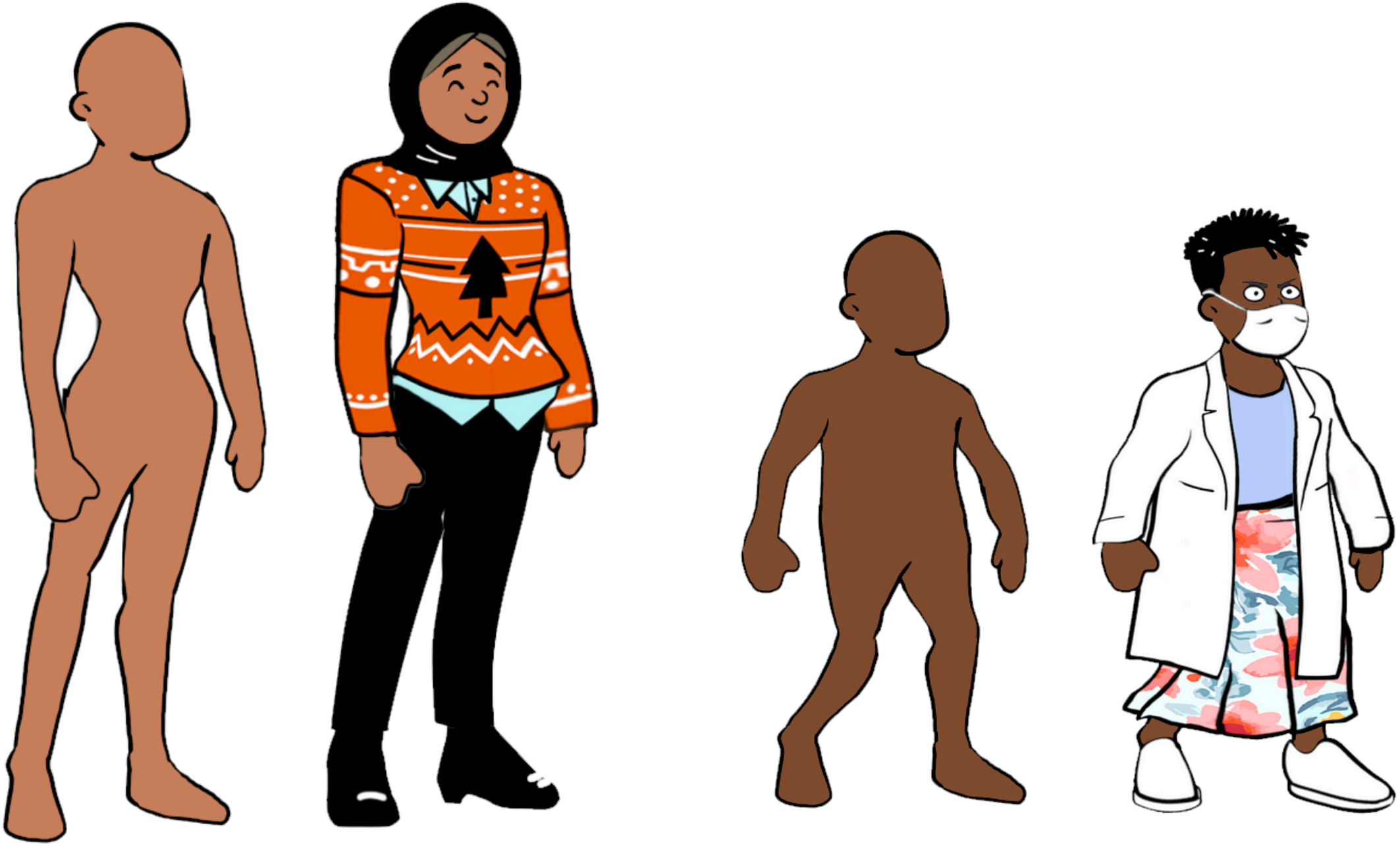


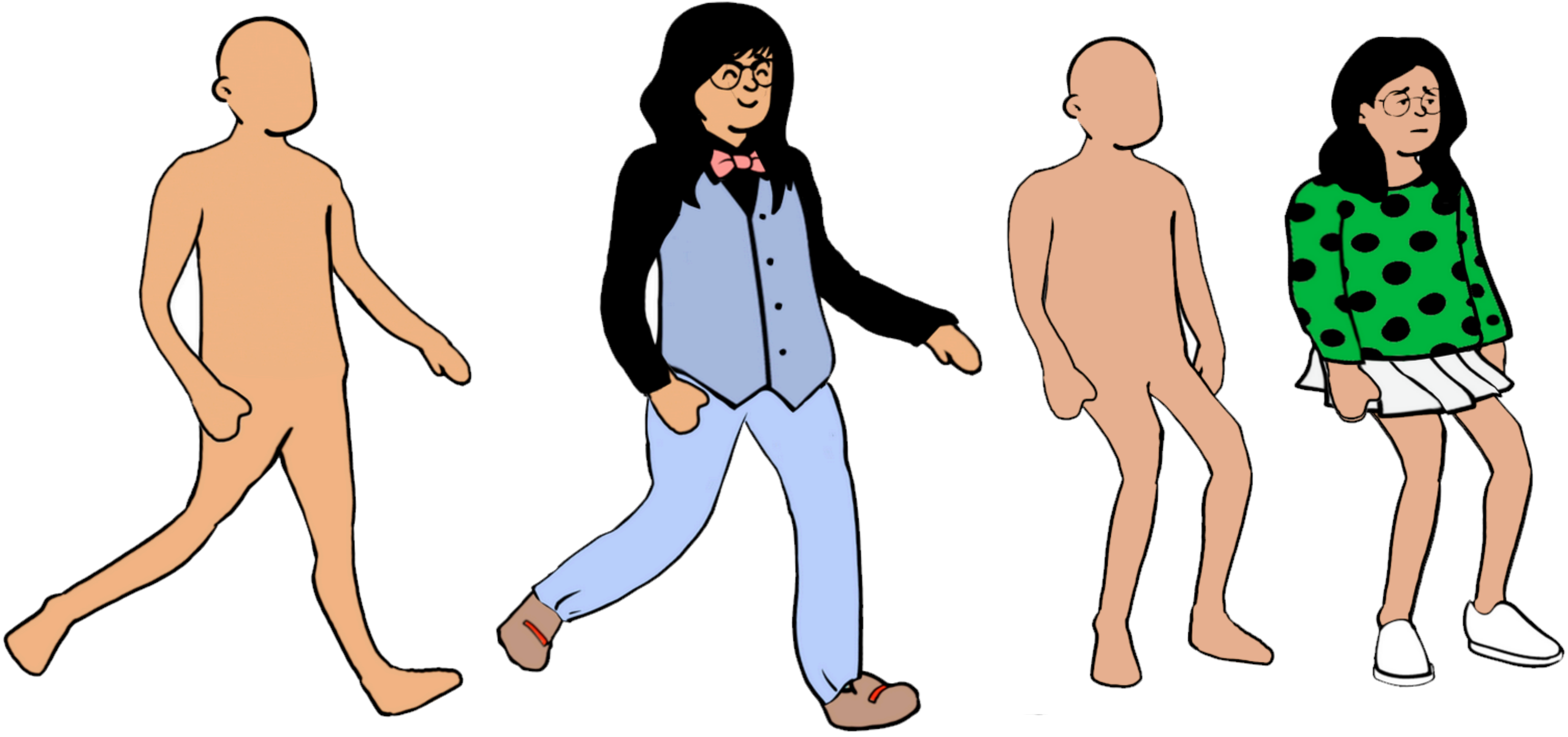
No resultant points: attach at a single point

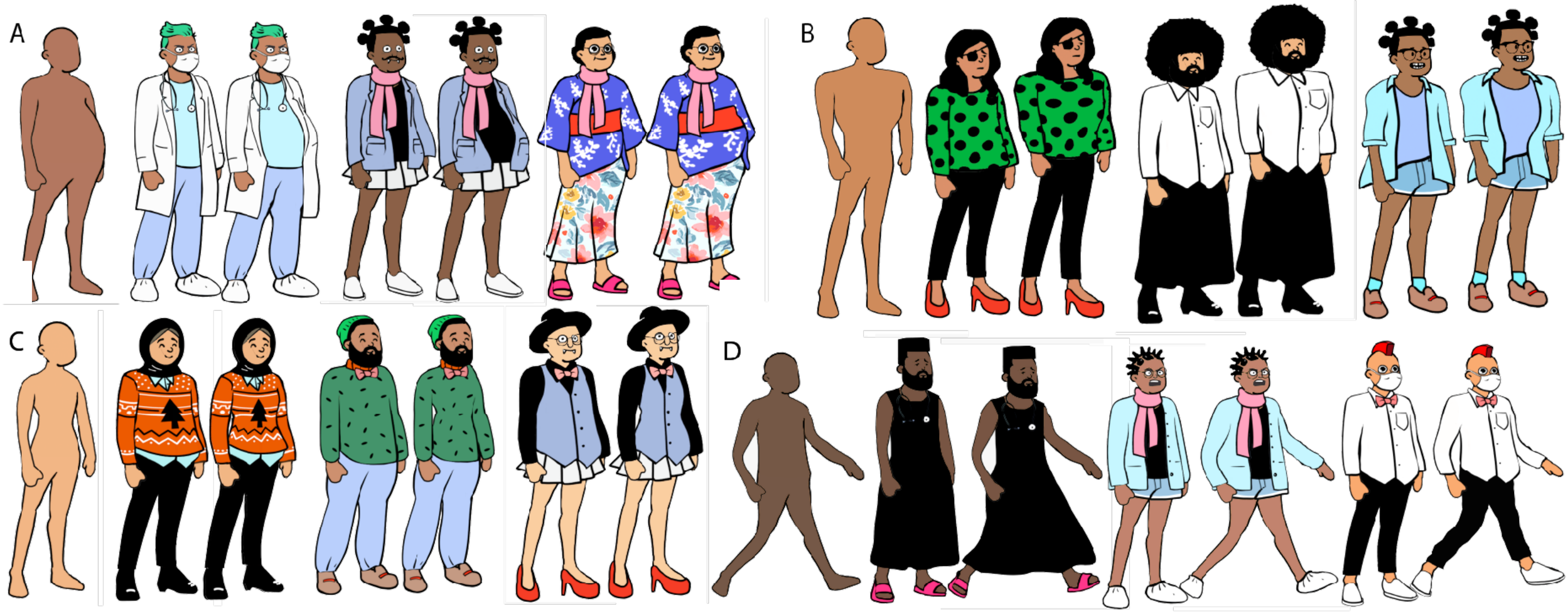


Results








A
B
C
D

Lecture 23 wrap-up

- HW10: On The Road due Tues 11:59pm
- **Checkpoint 3 in two weeks** (12/3). This is the last lecture on the checkpoint (but my rigging research project you saw is not on the checkpoint :))

Resources

- HW10: On The Road due Tues 11:59pm
- **Checkpoint 3 in two weeks** (12/3). This is the last lecture on the checkpoint.
- Textbook on DAGs (3.3.2): **http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf**
- Practice problems (conceptual, no solutions) behind this slide

Longest Path conceptual problems

- Find an example where the obvious algorithm (Dijkstra's but pick the biggest edge first) fails.
- Is the longest path to every other vertex always a tree (i.e. does an LPT exist for all graphs)?
- Why is the longest path problem (general, not just DAGs) hard? Try to develop an intuition for exactly how.
  - (Search the internet for any of these answers if you're having trouble thinking about it.)