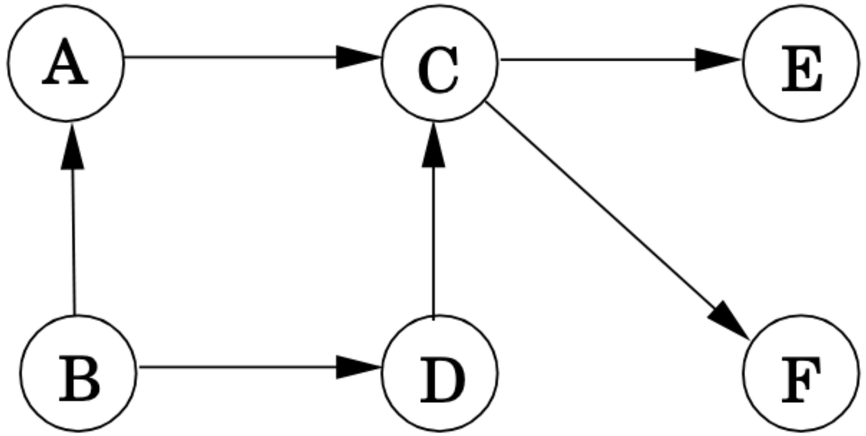
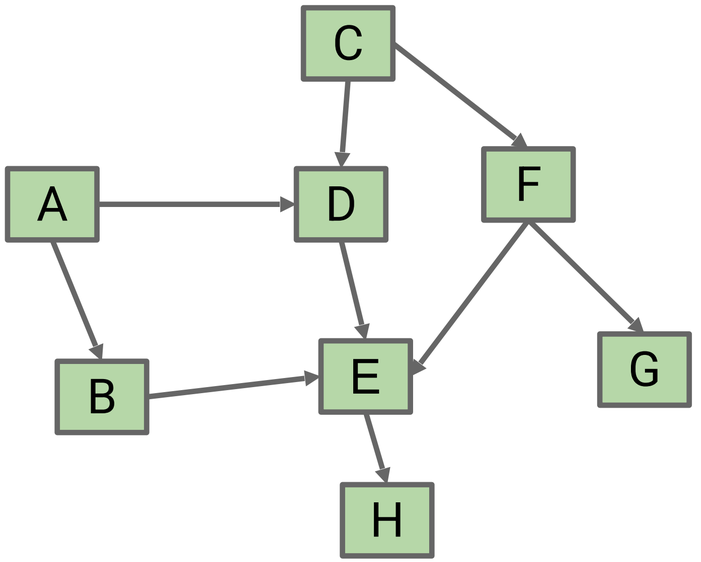
CS62 Class 24: More DAGs

Graphs





Agenda

- Wrapping up DAGs
  - Shortest paths on DAGs
  - Longest paths
- For “fun”
  - One DAG application in Prof Li’s research
  - Semester personal reflections
  - AMA

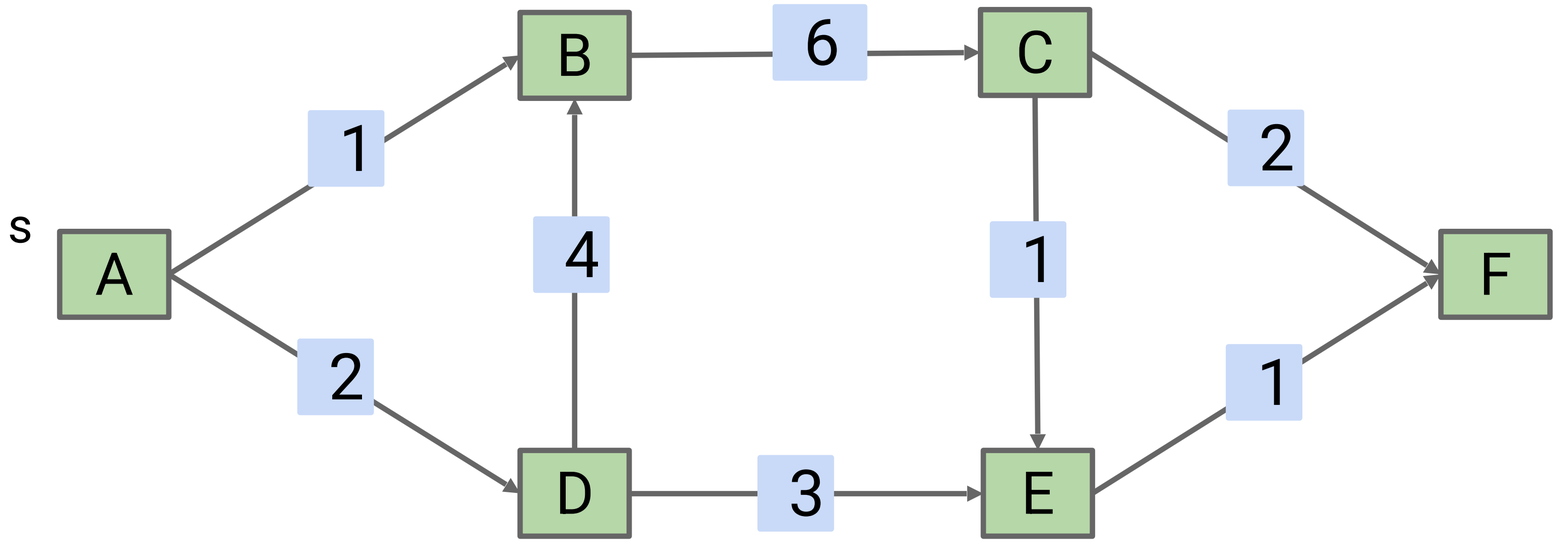
Last time: Graph algos so far

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

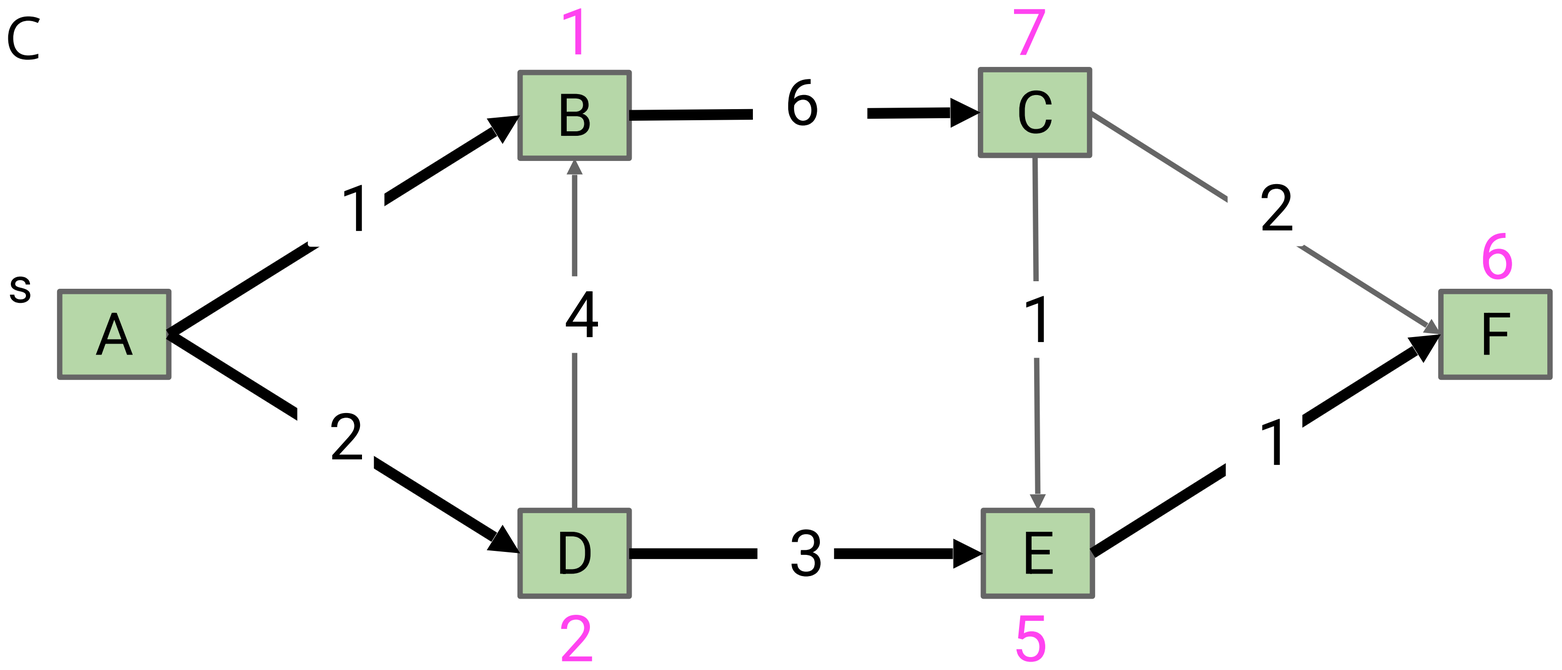
What is the shortest paths tree for the graph below, using s as the source?
In what order will Dijkstra's algorithm visit the vertices?



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?

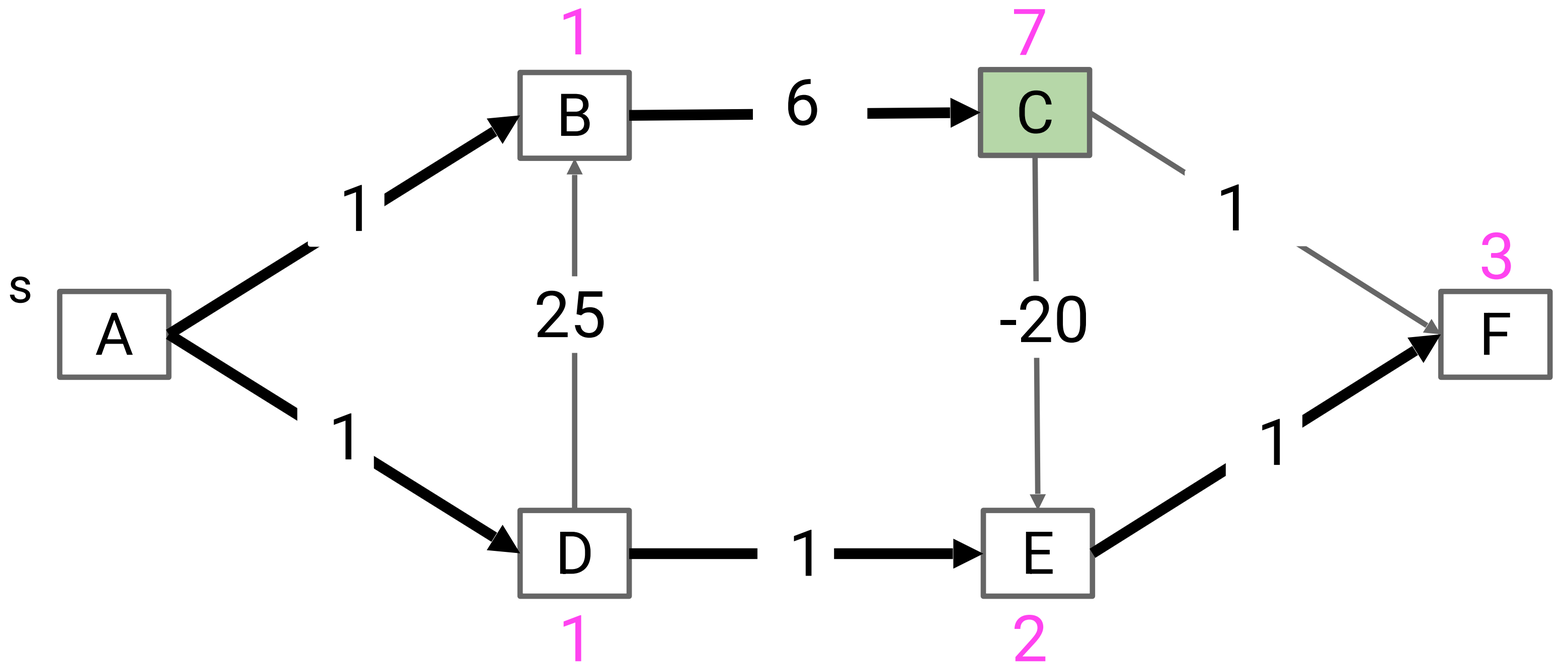
- A, B, D, E, F, C



Shortest Paths with negative edges

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

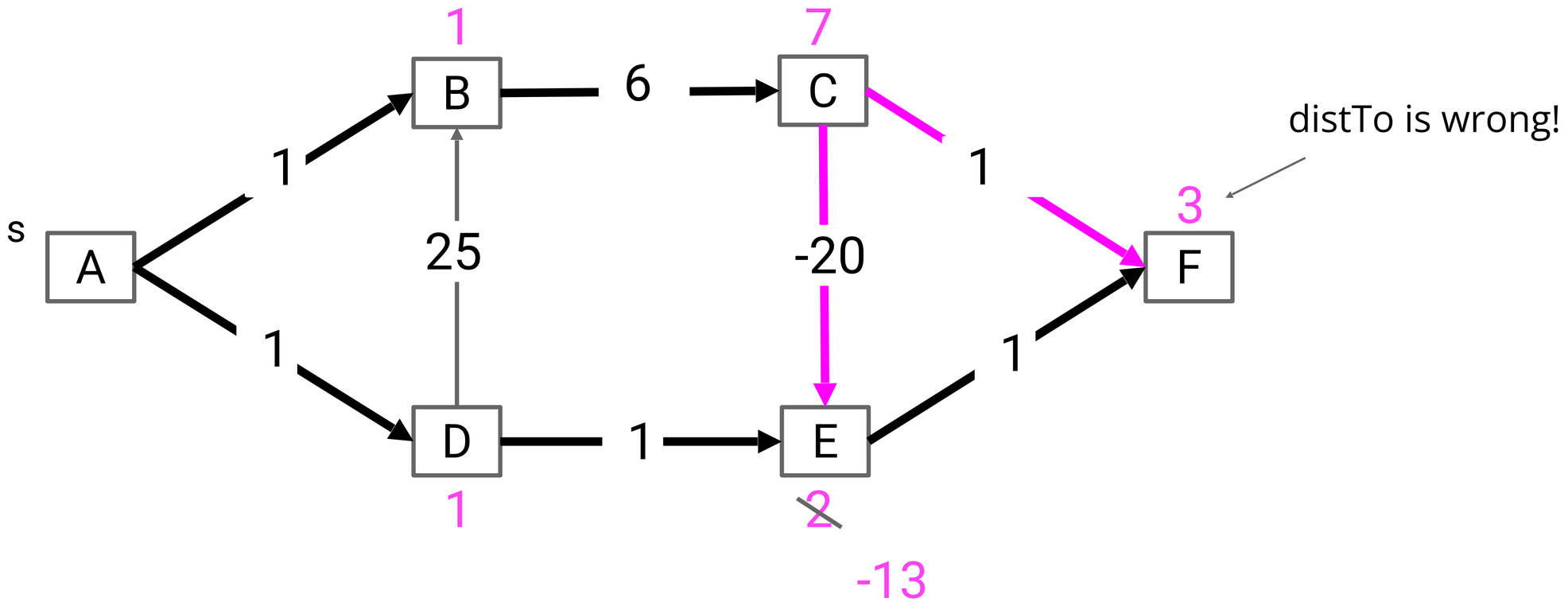
- For example, below we see Dijkstra's just before vertex C is visited.



Shortest Paths with negative edges

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

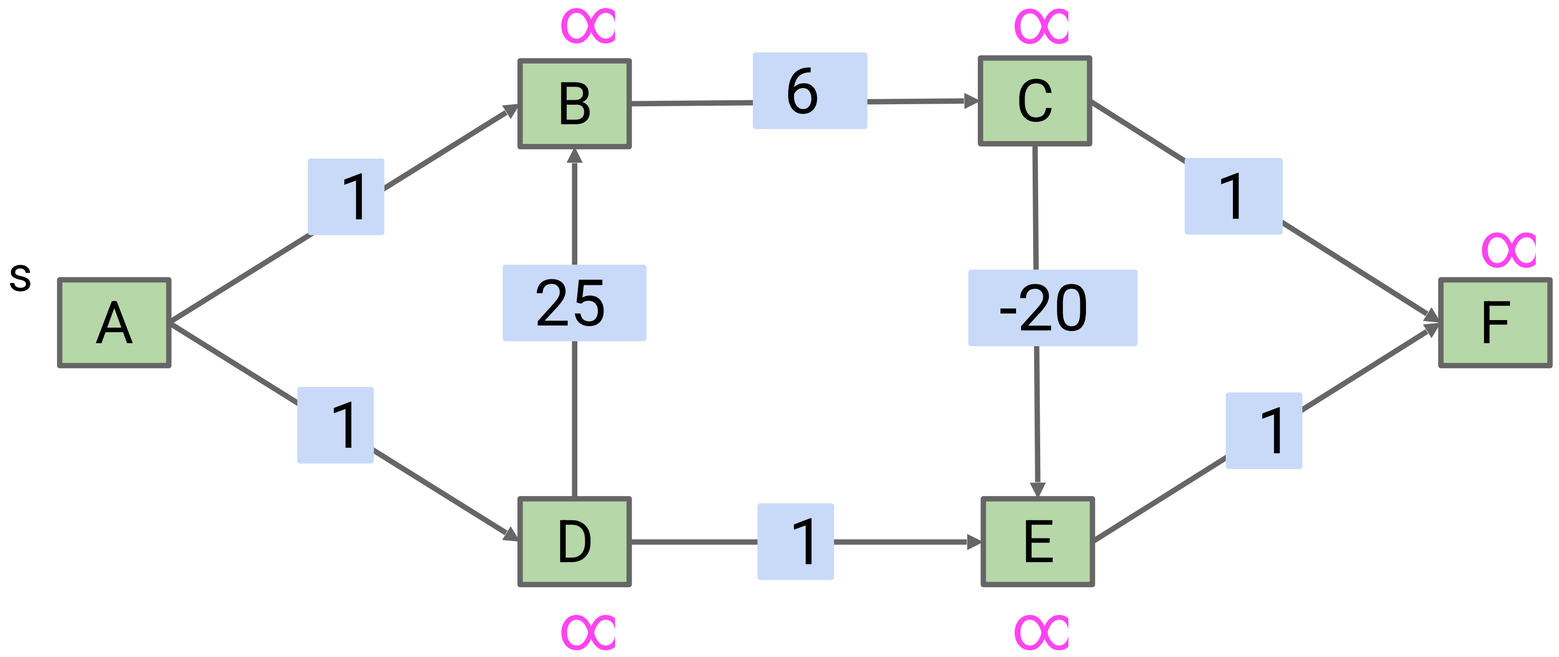
- For example, below we see Dijkstra's just before vertex C is visited.
- Relaxation on E succeeds, but distance to F (now should be -10) will never be updated.



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.

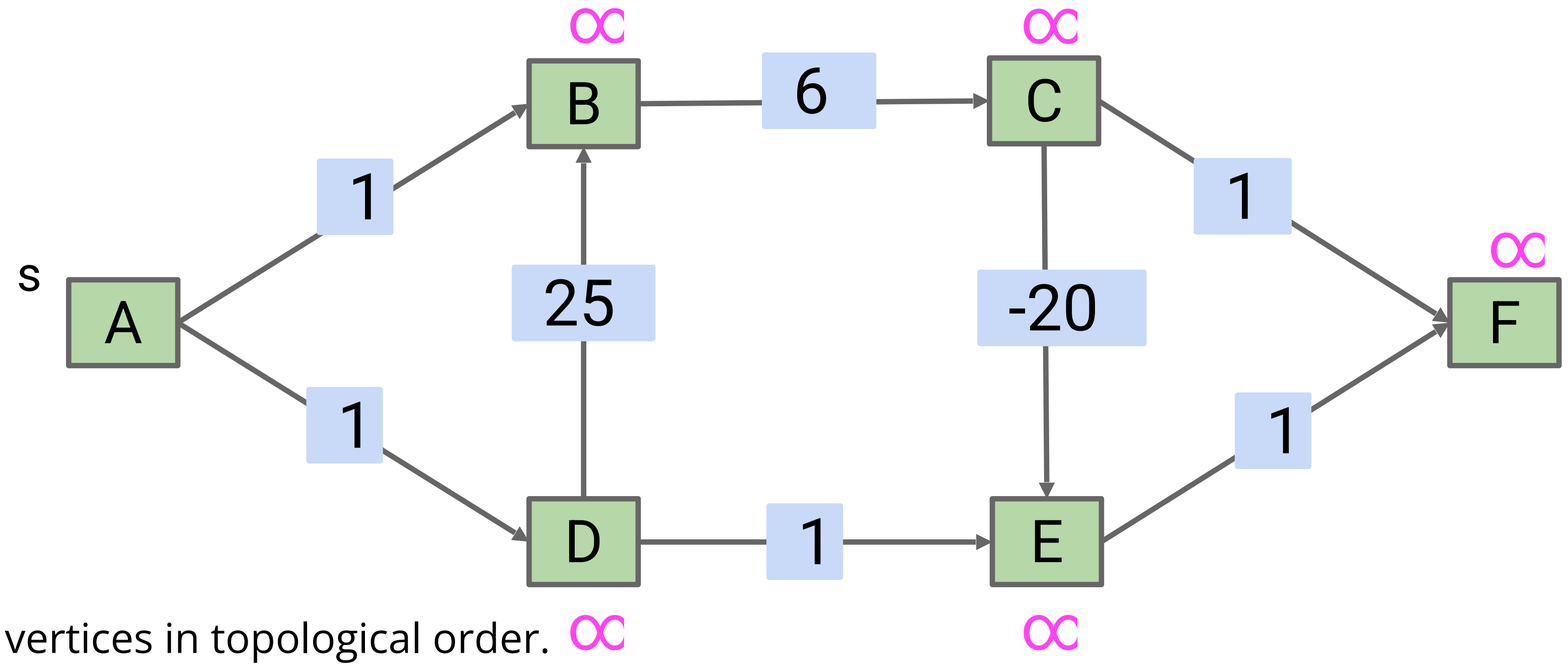
- Hint: You should still use the “relax” operation as a basic building block.



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.

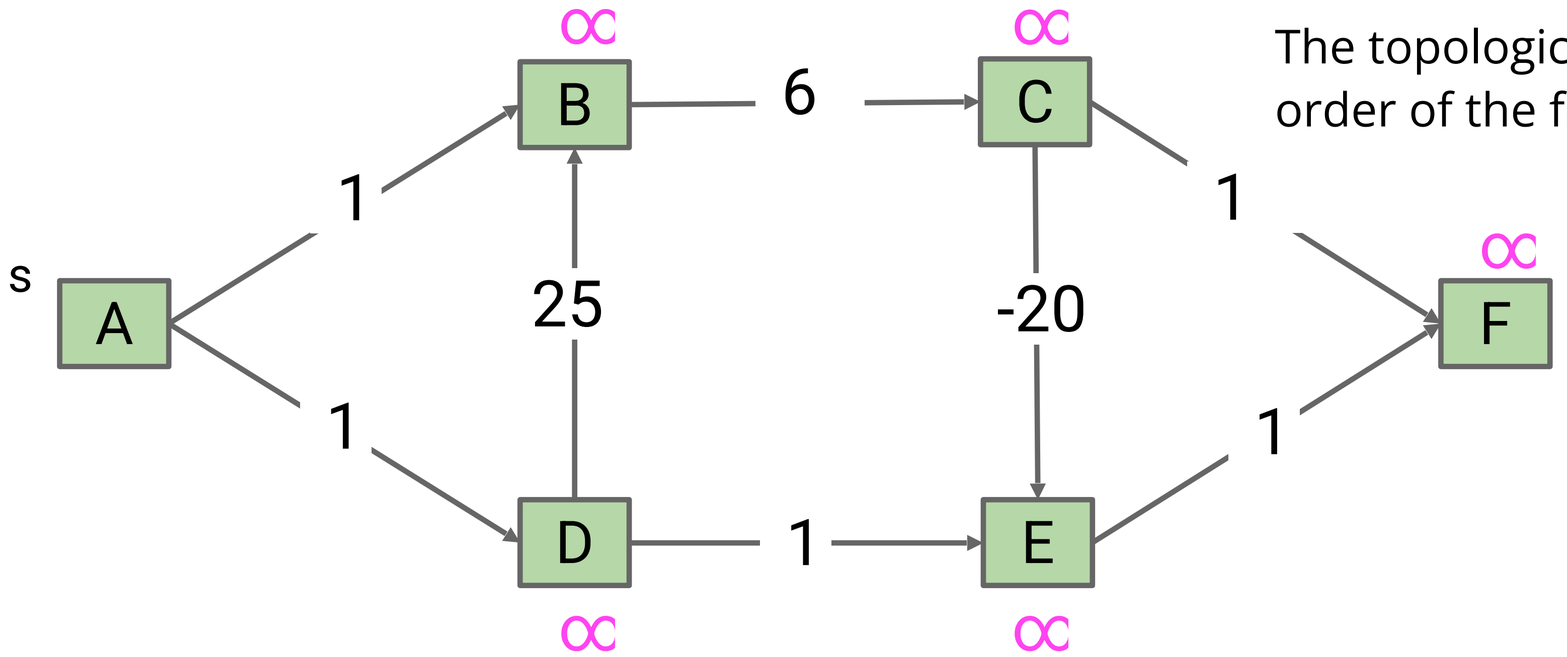


One simple idea: Visit vertices in topological order.

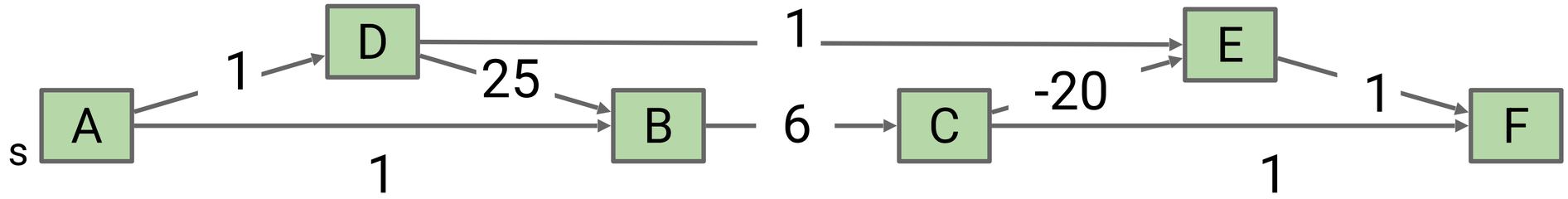
- On each visit, relax all outgoing edges.
- Each vertex is visited only when all possible info about it has been used!

The DAG SPT Algorithm: Relax in Topological Order

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

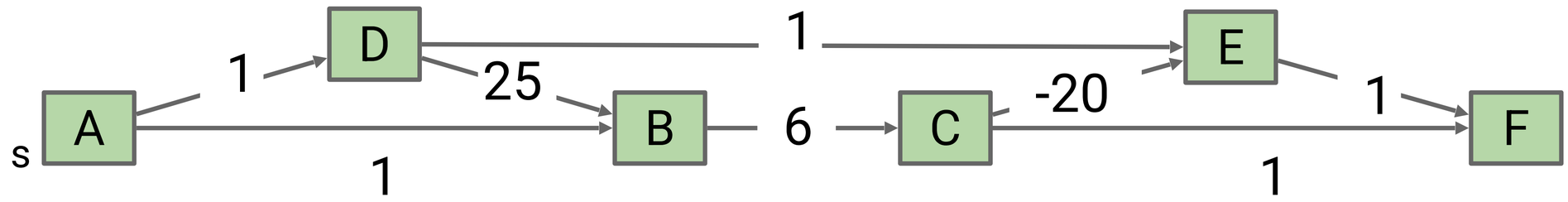


The topological order is just the order of the fringe priority queue!



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.

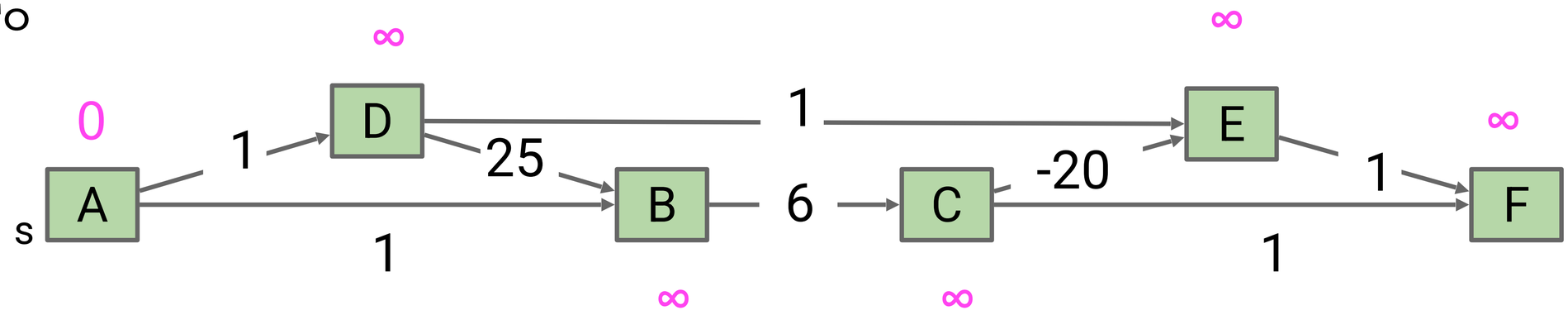


DAG SPT Algorithm

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	∞	-



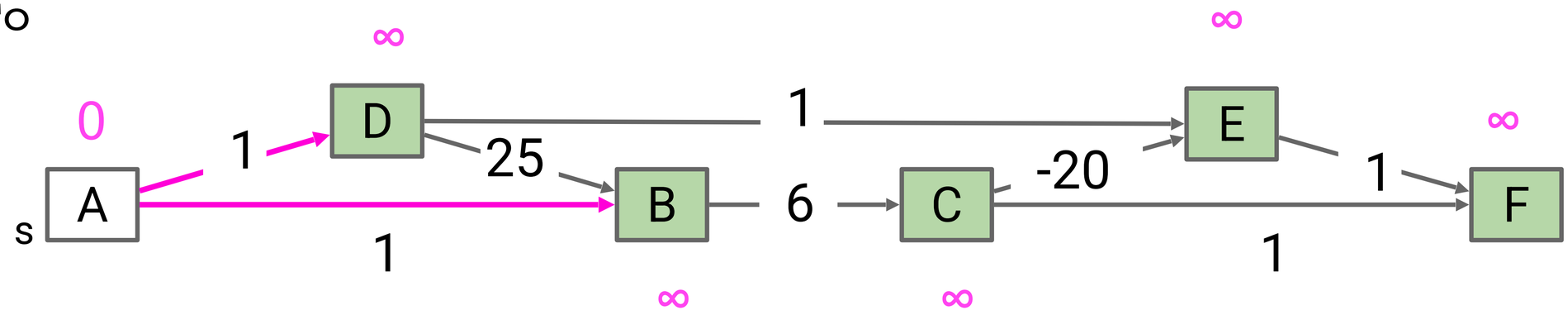
Fringe: [A, D, B, C, E, F]

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	∞	-



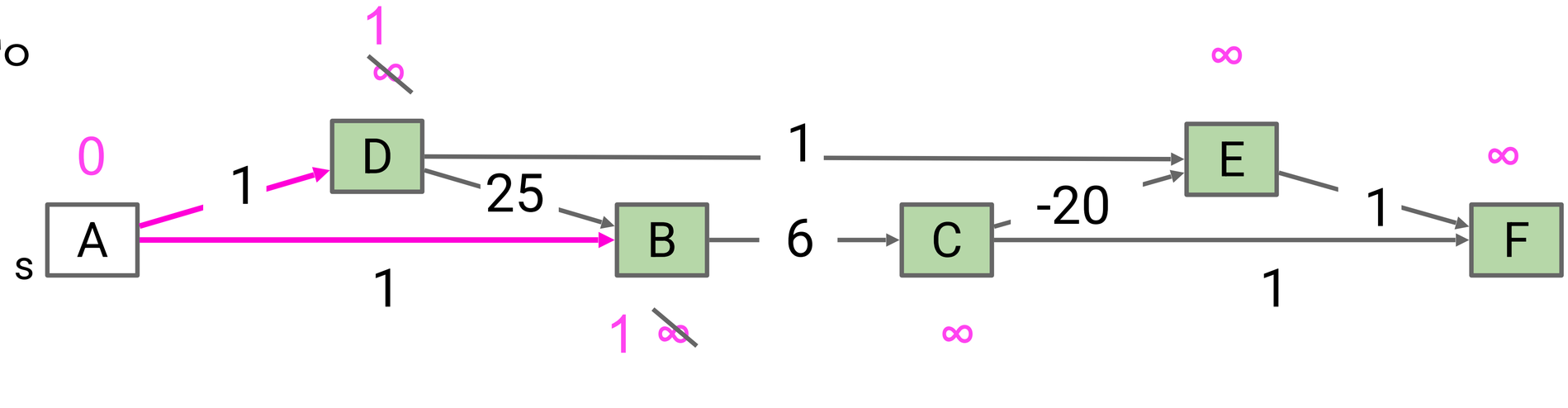
Fringe: [A, D, B, C, E, F]

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	∞	-



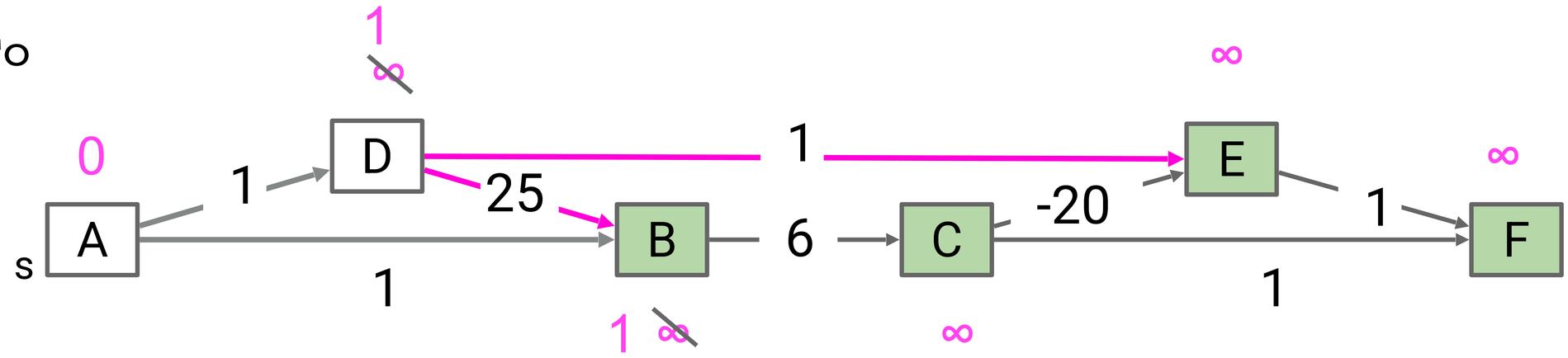
Fringe: [A, D, B, C, E, F]

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	∞	-



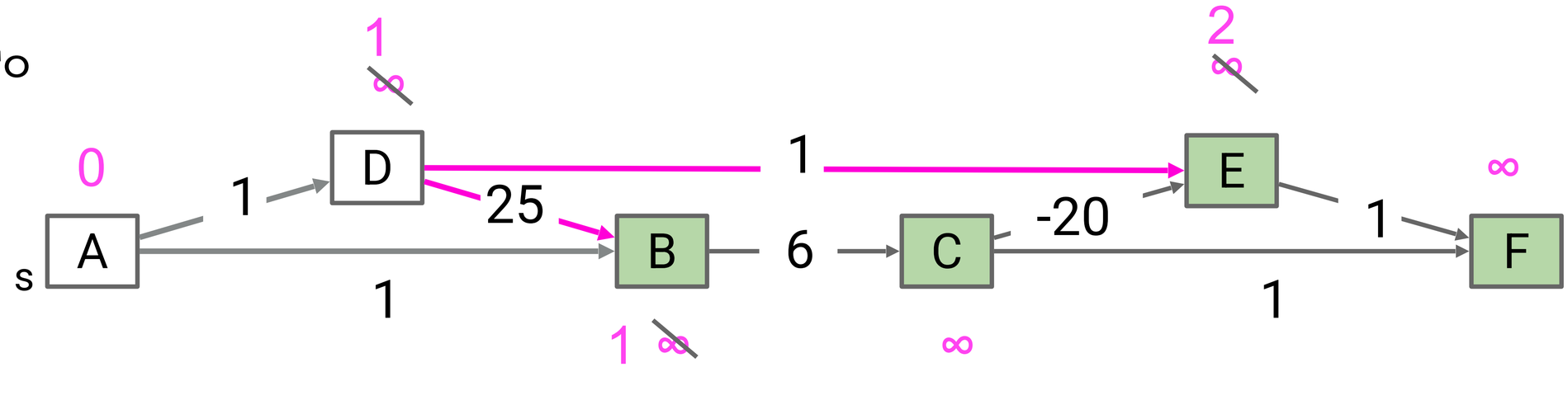
Fringe: [~~A~~, ~~D~~, B, C, E, F]

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	∞	-



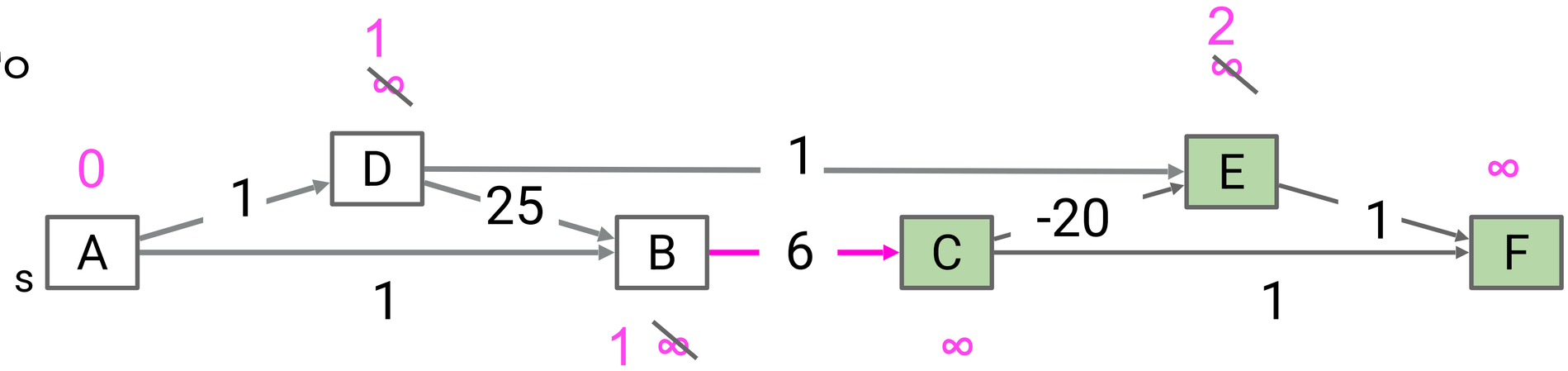
Fringe: [~~A~~, ~~D~~, B, C, E, F]

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	∞	-



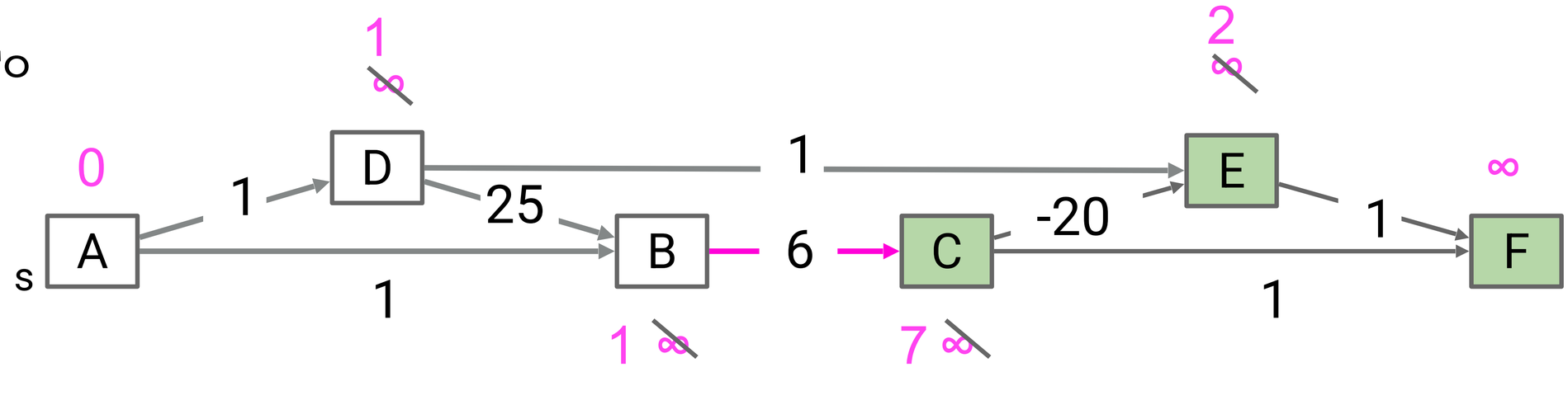
Fringe: [~~A~~, ~~D~~, ~~B~~, C, E, F]

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	∞	-



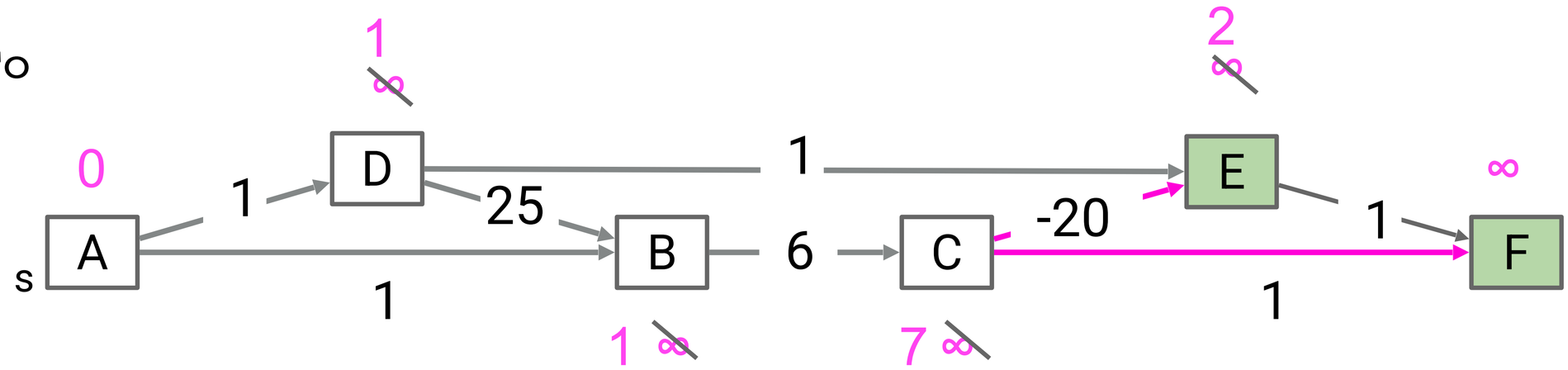
Fringe: [~~A~~, ~~D~~, ~~B~~, C, E, F]

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	∞	-



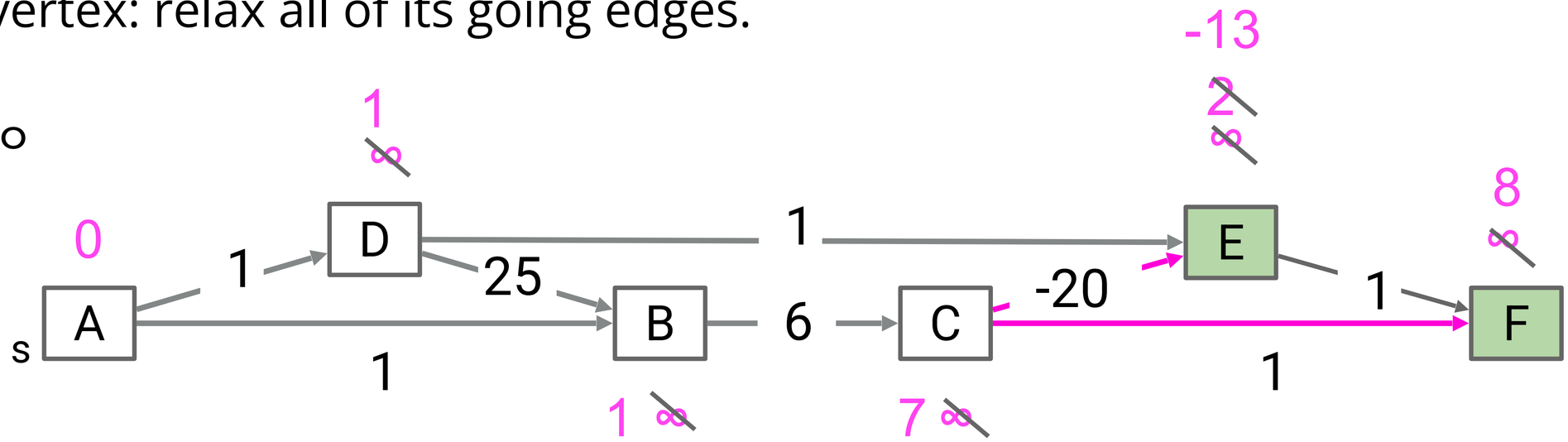
Fringe: [~~A~~, ~~D~~, ~~B~~, ~~C~~, E, F]

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



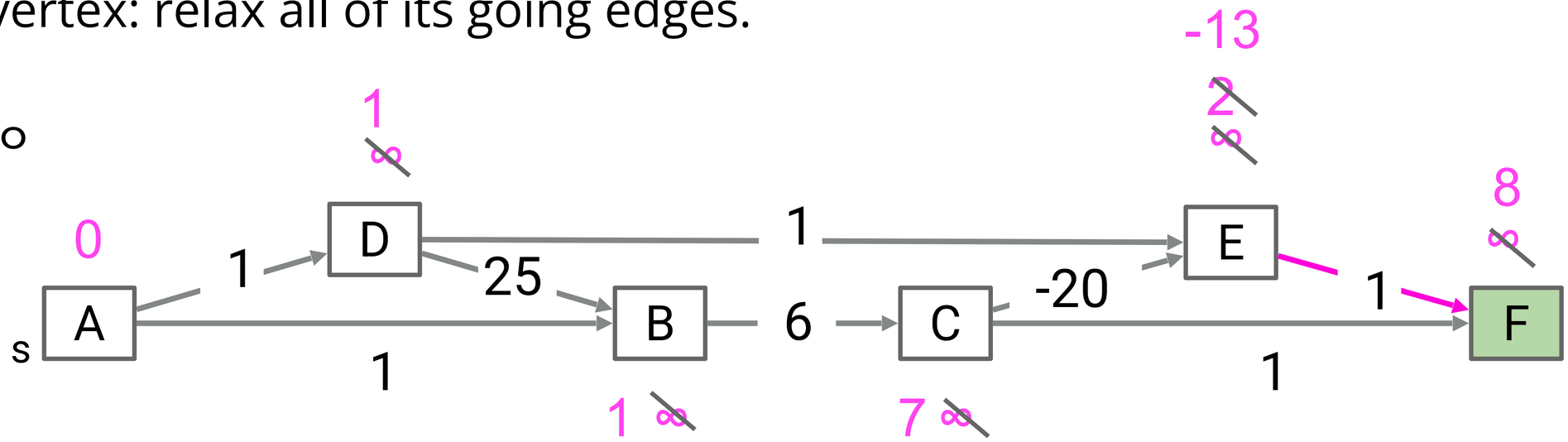
Fringe: [~~A~~, ~~D~~, ~~B~~, ~~C~~, E, F]

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



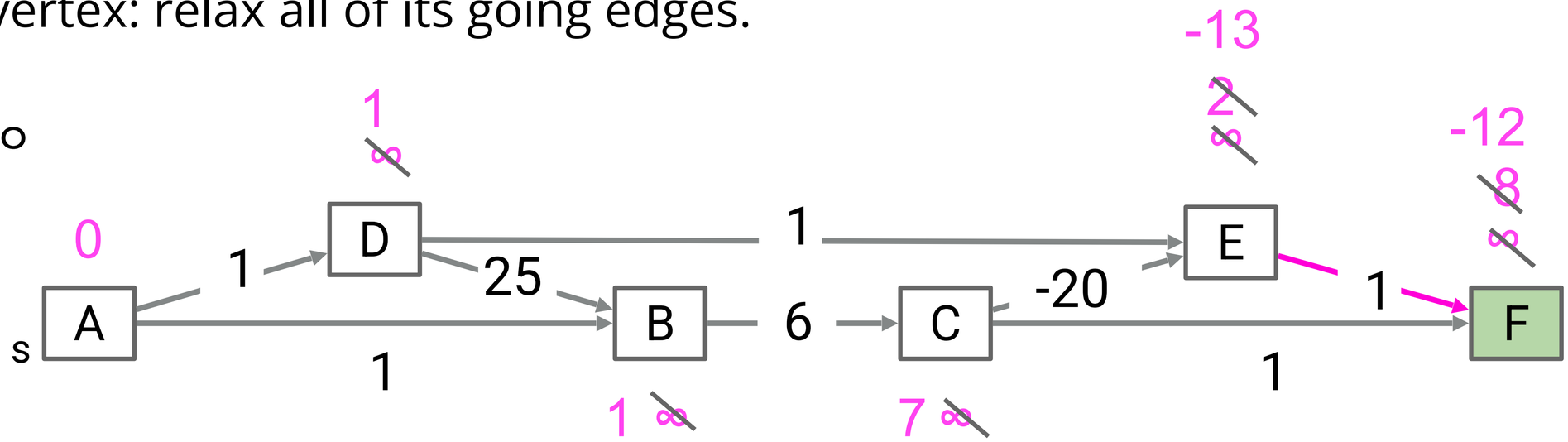
Fringe: [~~A~~, ~~D~~, ~~B~~, ~~C~~, ~~E~~, F]

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



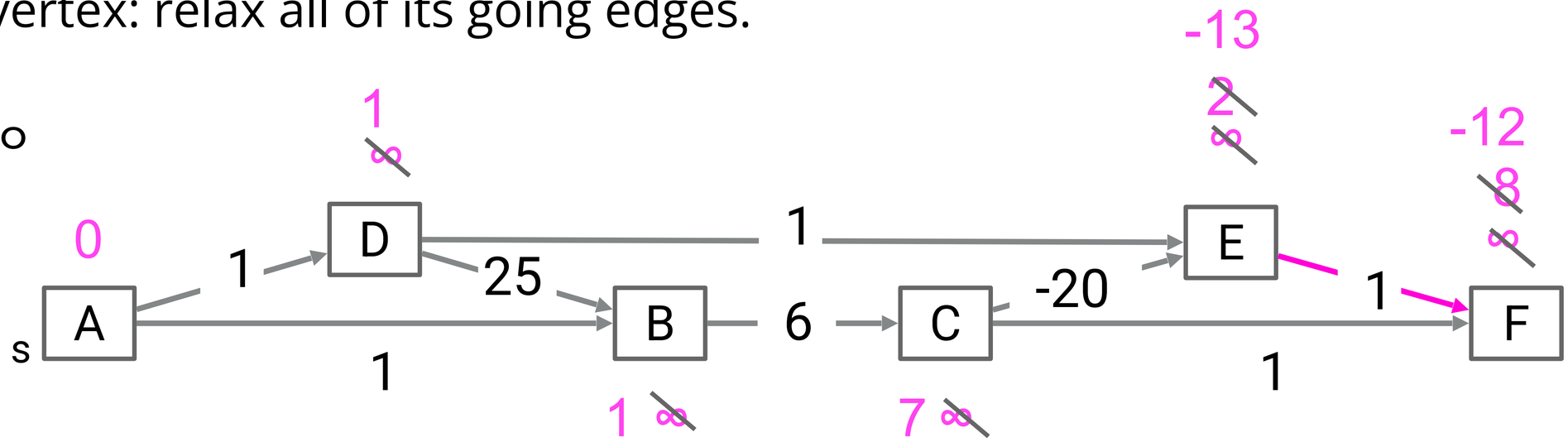
Fringe: [~~A~~, ~~D~~, ~~B~~, ~~C~~, ~~E~~, F]

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.

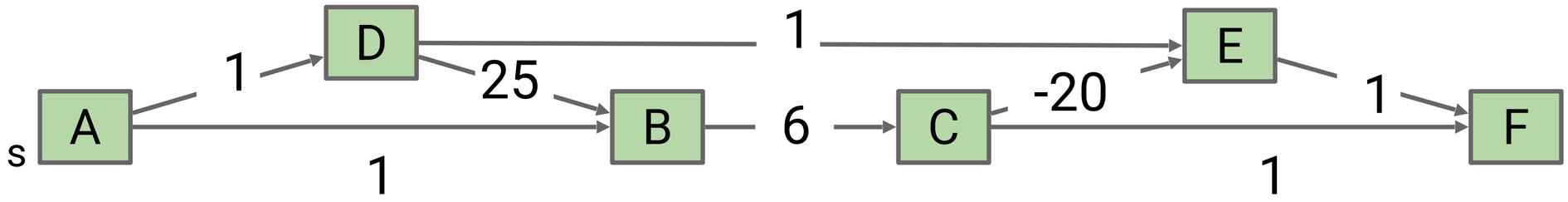
Fringe: [~~A~~, ~~D~~, ~~B~~, ~~C~~, ~~E~~, ~~F~~]

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.

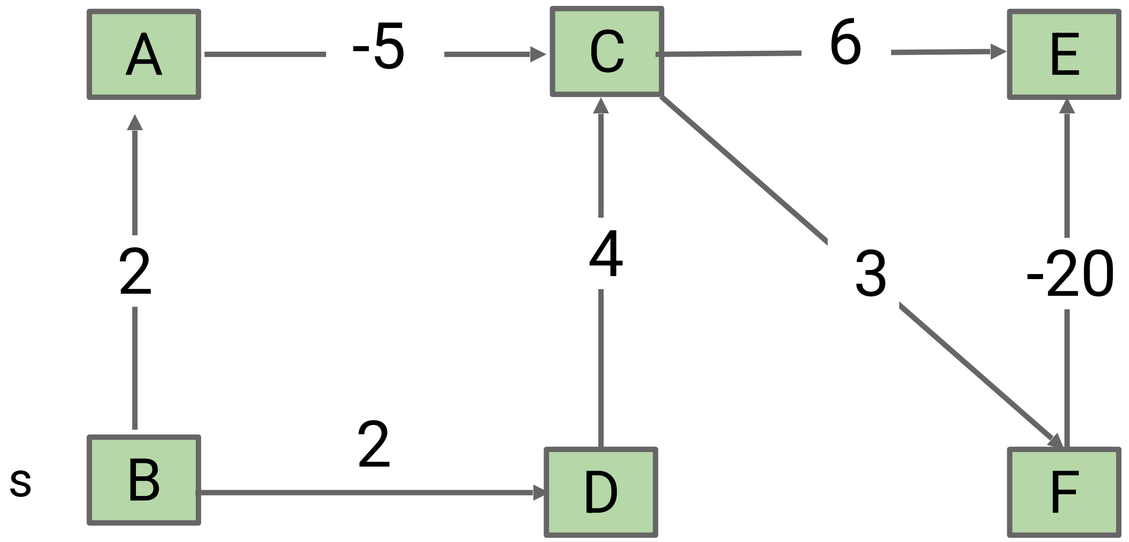


DAG algos

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

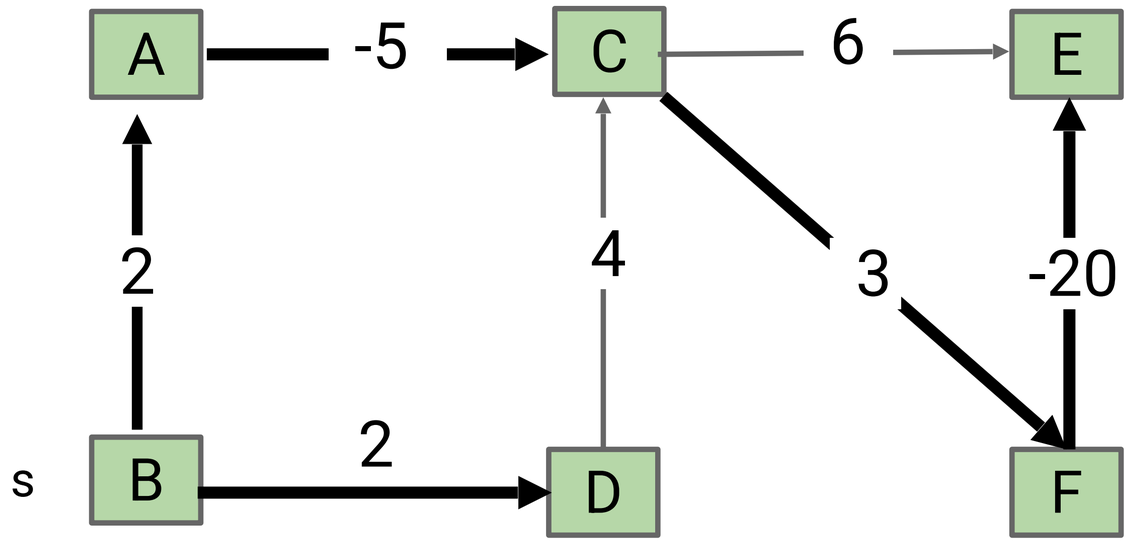
Worksheet time!

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?

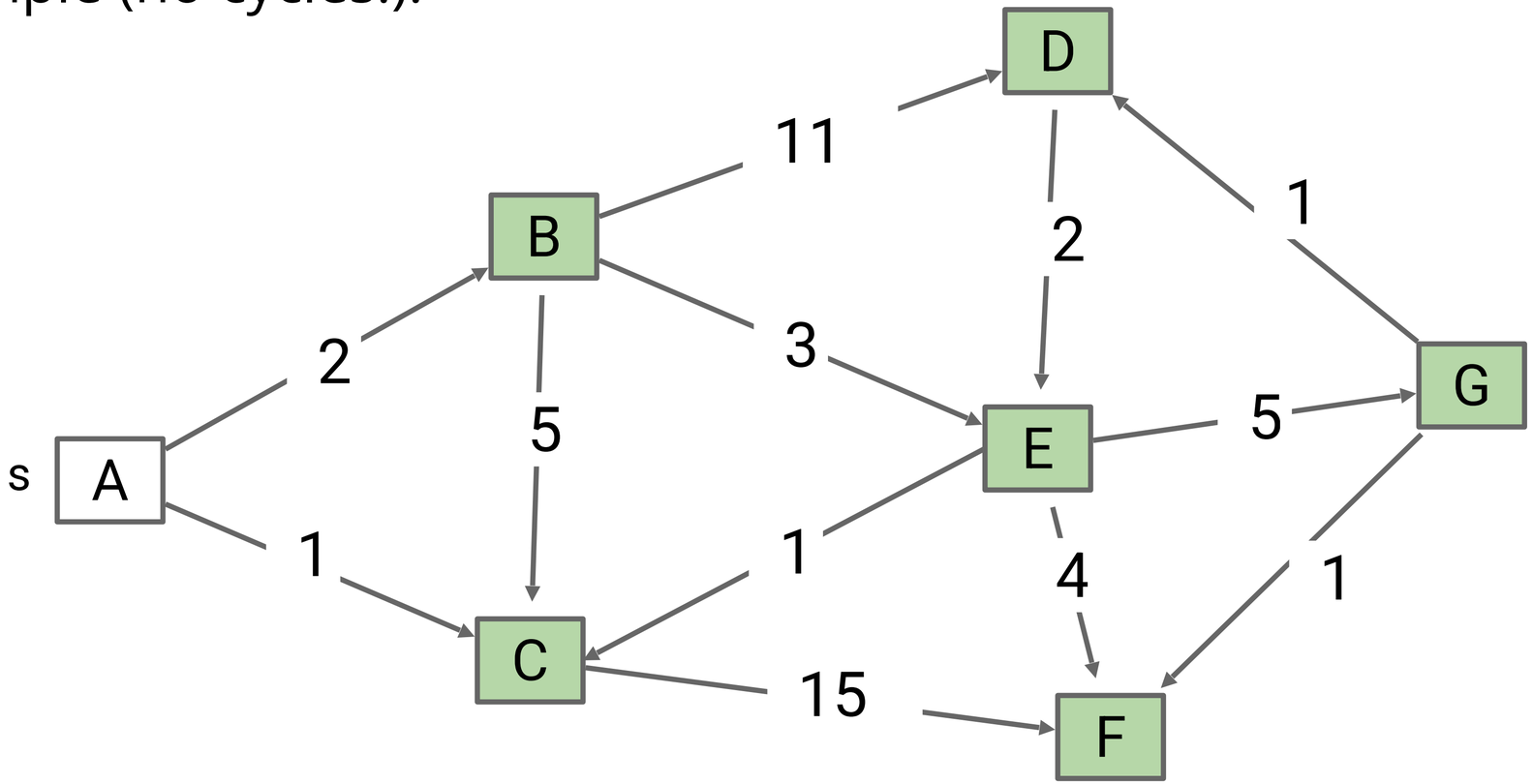


	distTo	edgeTo
A	2	B
B	0	-
C	-3	A
D	2	B
E	-20	F
F	0	C

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!).

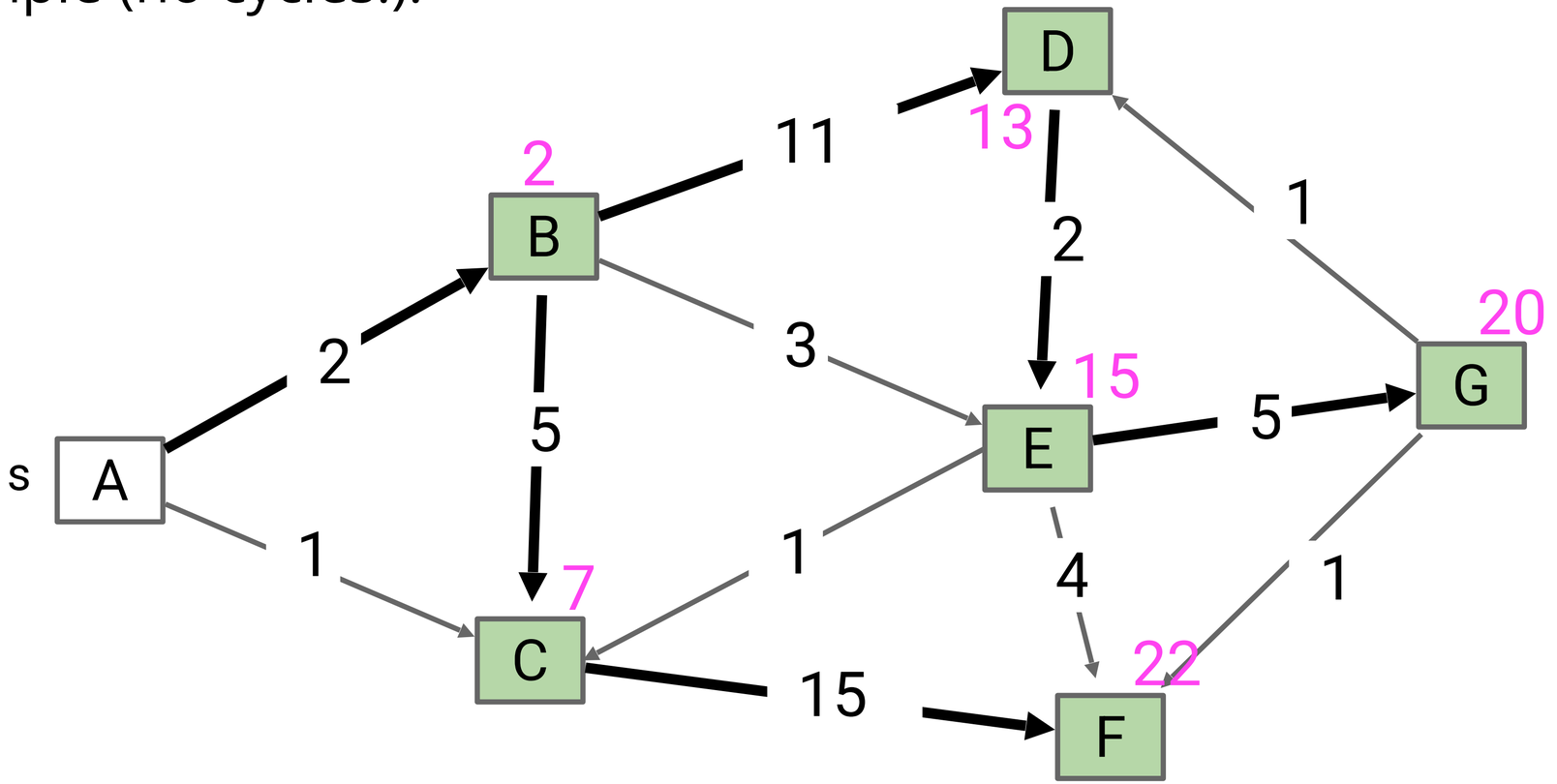


The Longest Paths Problem

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.

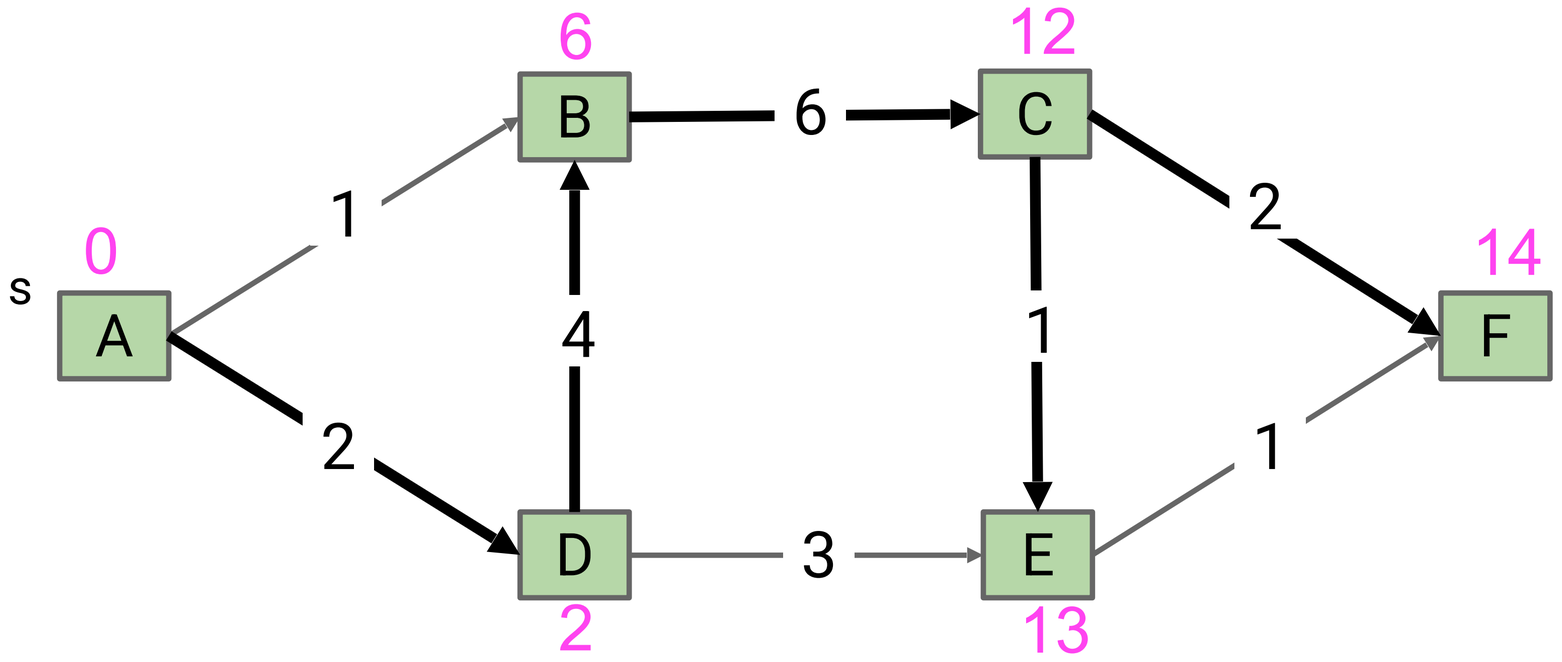


The Longest Paths Problem on DAGs

(Worksheet time!)

Difficult challenge for you:

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



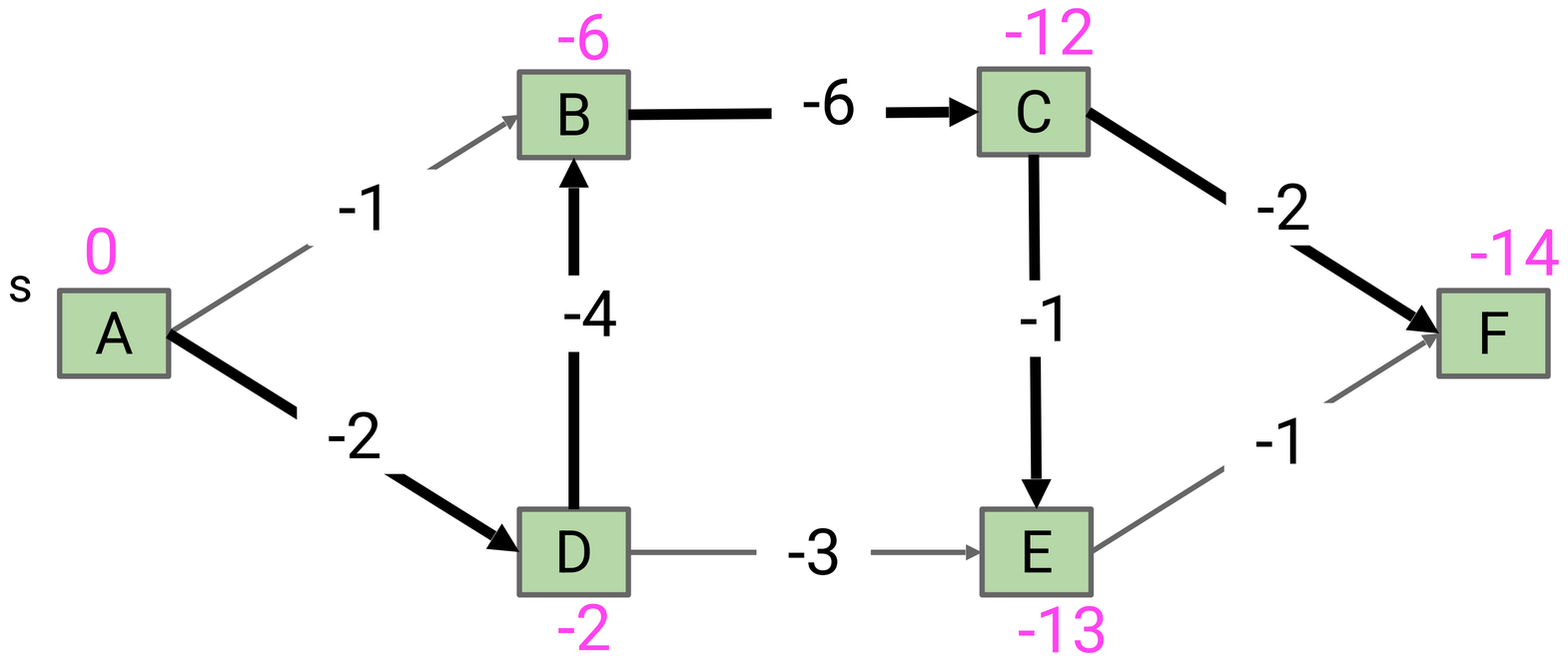
Hint: How can we make this problem look like the problem we just solved?

The Longest Paths Problem on DAGs

(Worksheet answers)

DAG LPT solution for graph G:

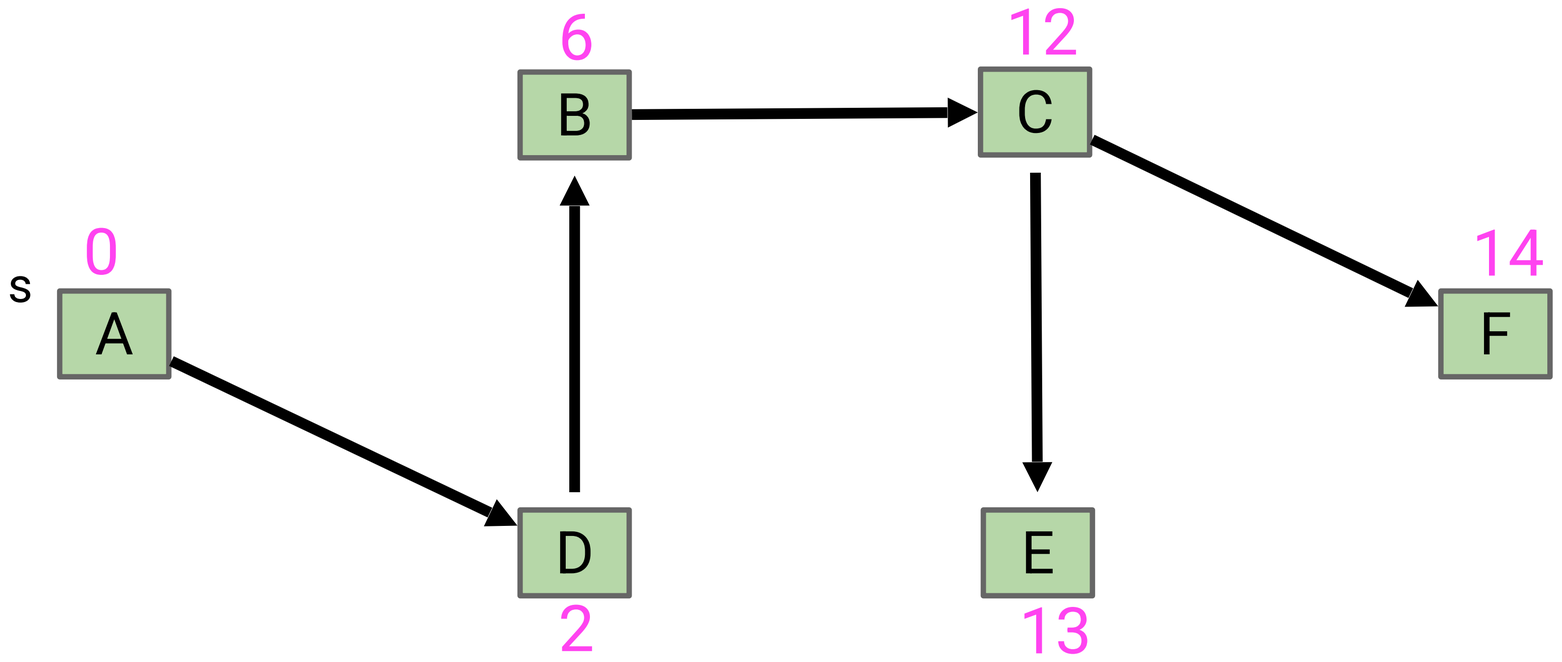
- Form a new copy of the graph G' with signs of all edge weights **flipped**.
- Run DAGSPT on G' yielding result X.
- Flip signs of all values in X.distTo. X.edgeTo is already correct.



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

This slide ends content that will be on checkpoint 3!
Anything after you will not be tested on.

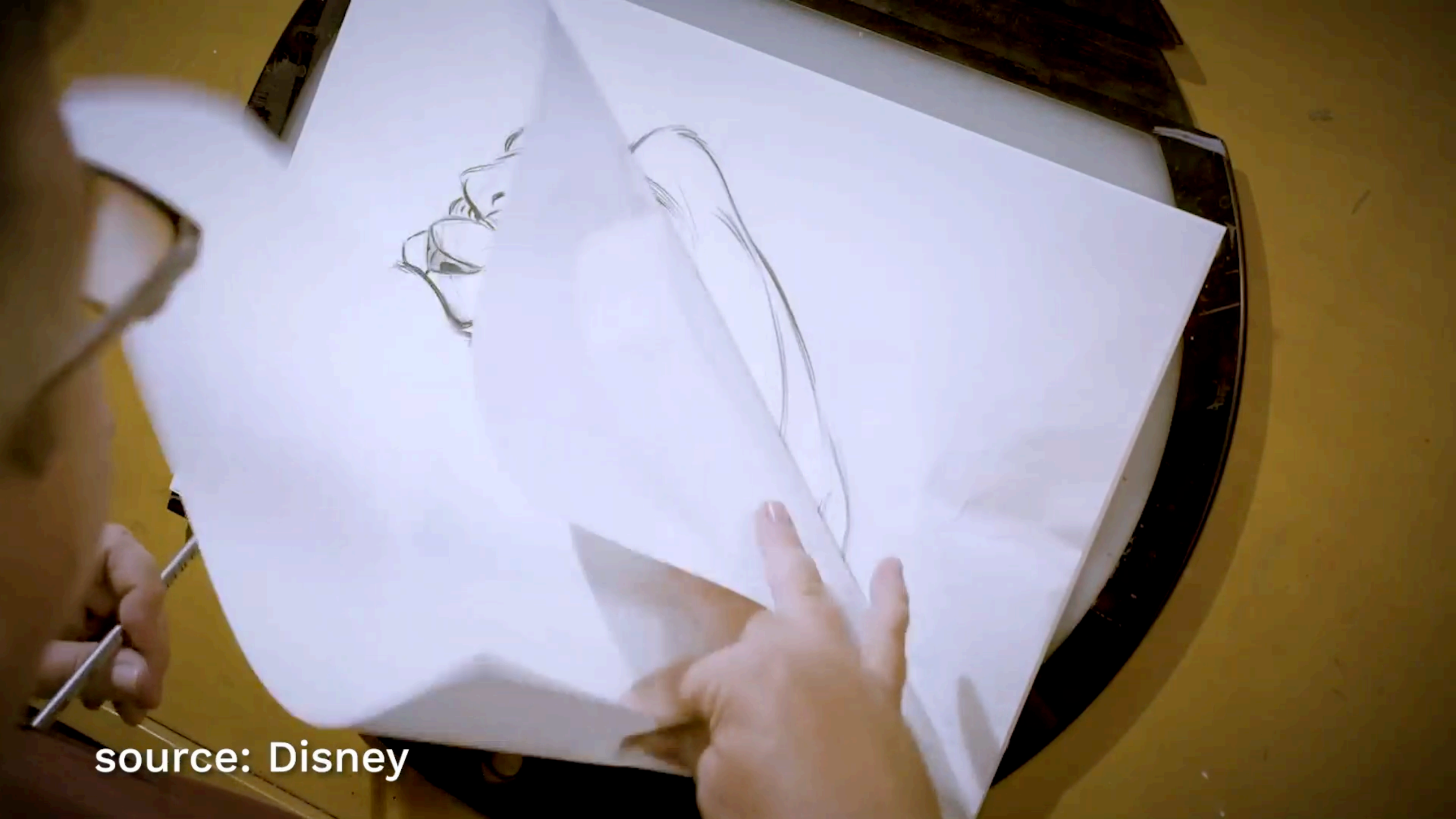
Why? It's just
DAG shortest paths.
Which is just topological sort.
Which is just DFS.

DAGs in my research

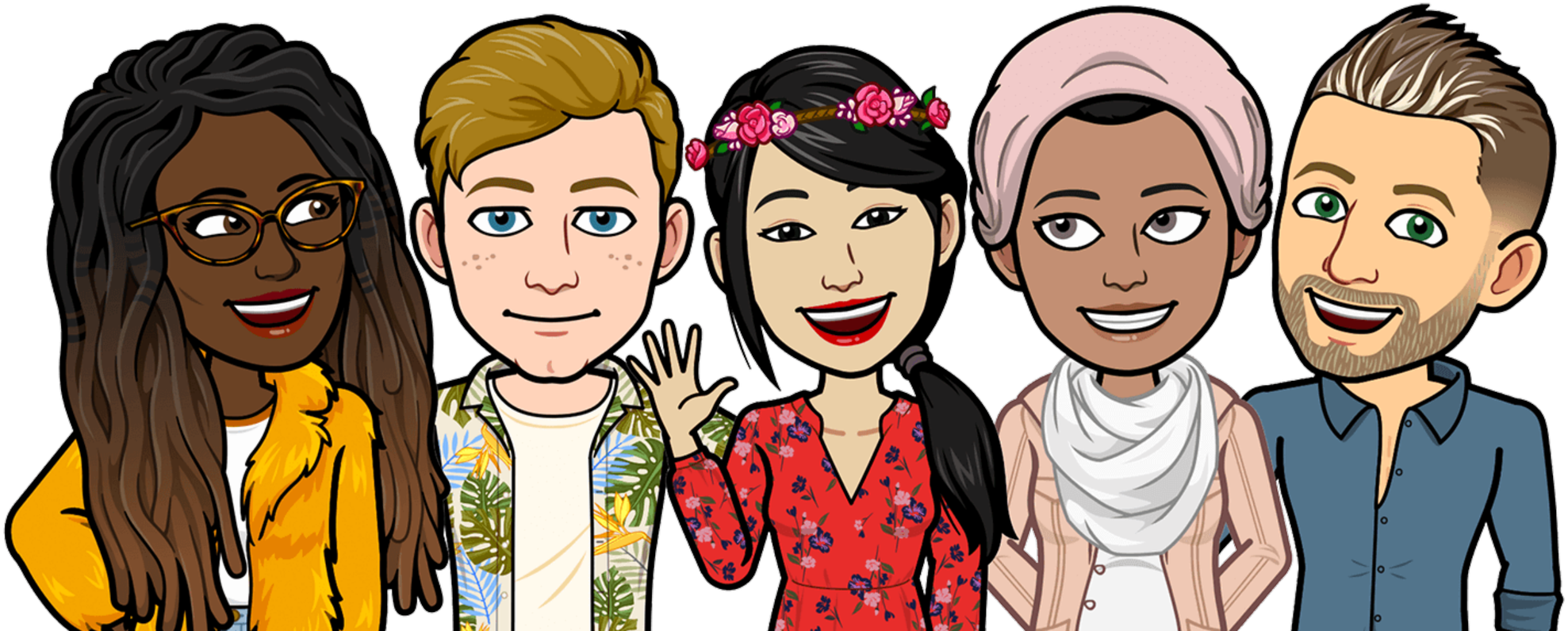
Automated Accessory Rigs for Layered 2D Character Illustrations

Jingyi Li • Wilmot Li • Sean Follmer • Maneesh Agrawala


source: RossDraws, YouTube


source: Disney

Mix & match character creation



source: Bitmoji

Home / Collections / Open Peeps /

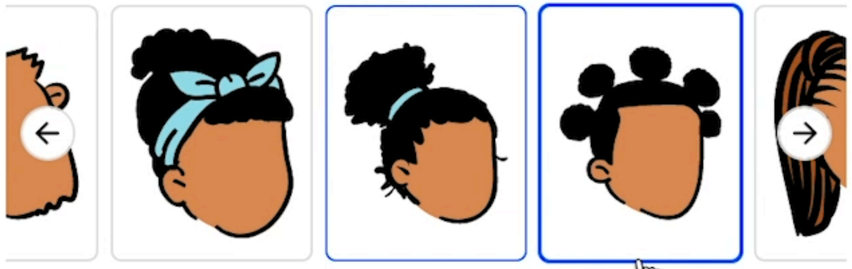
Standing by Pablo Stanley

Save Share Download

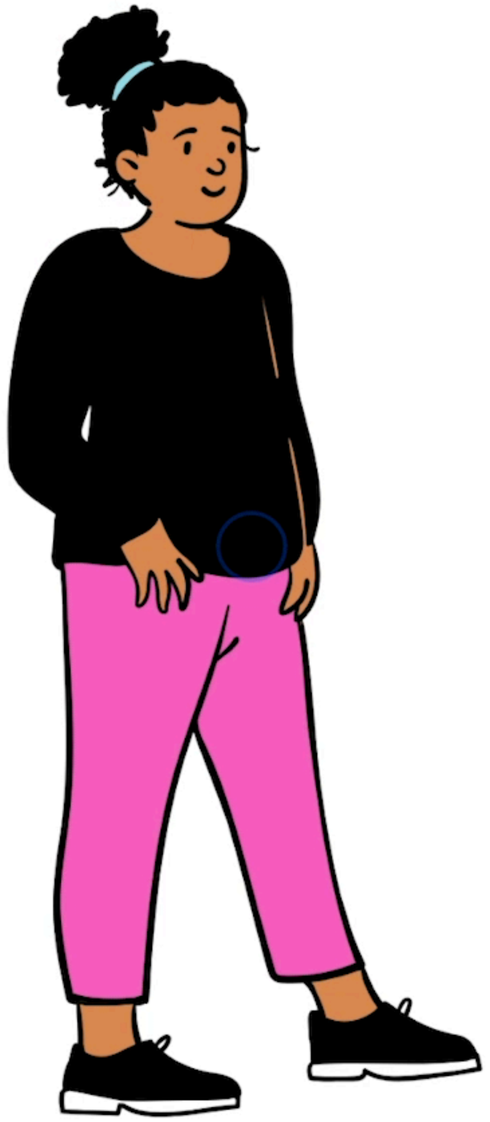
Skin

Standing See All

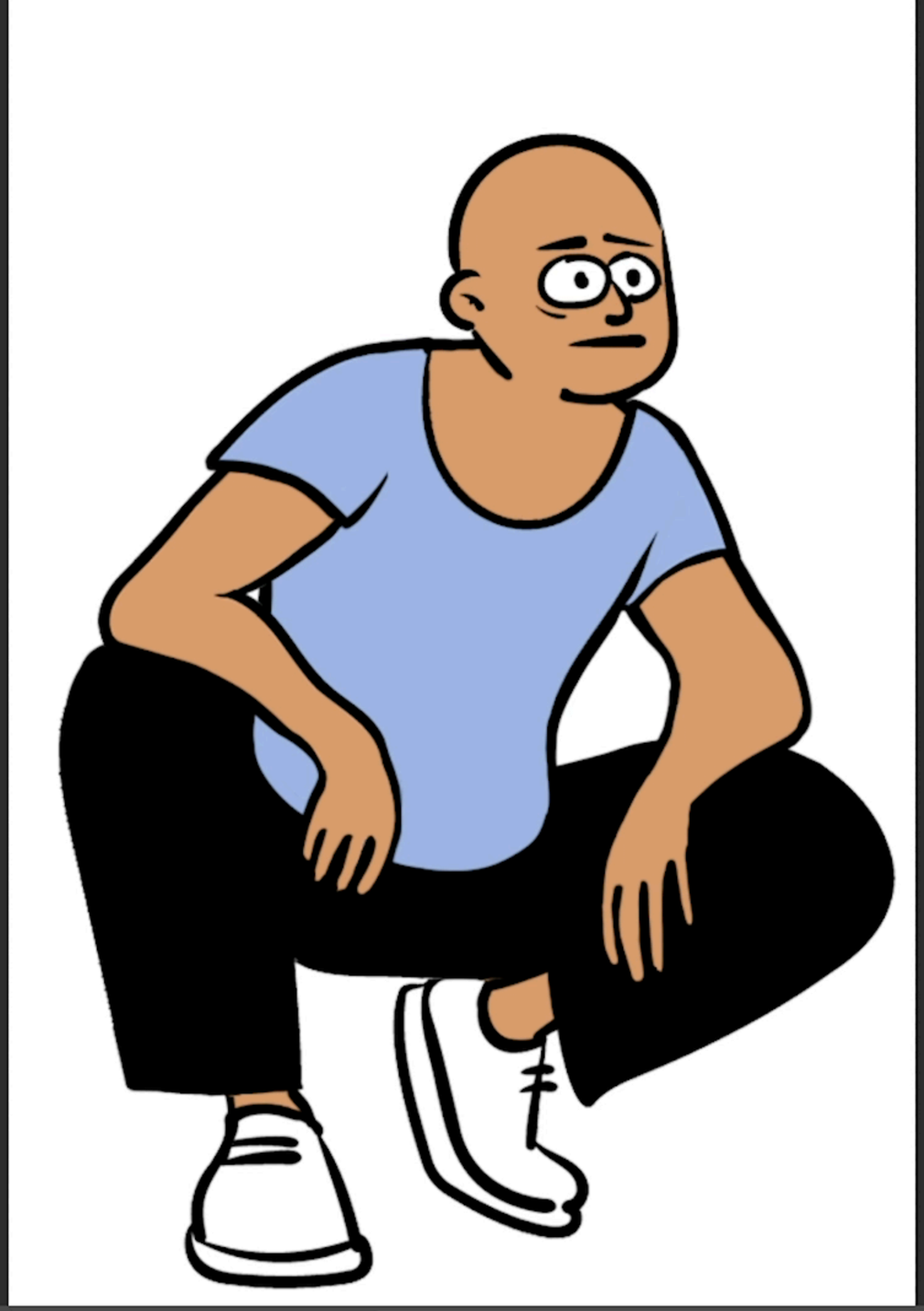
Head See All



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

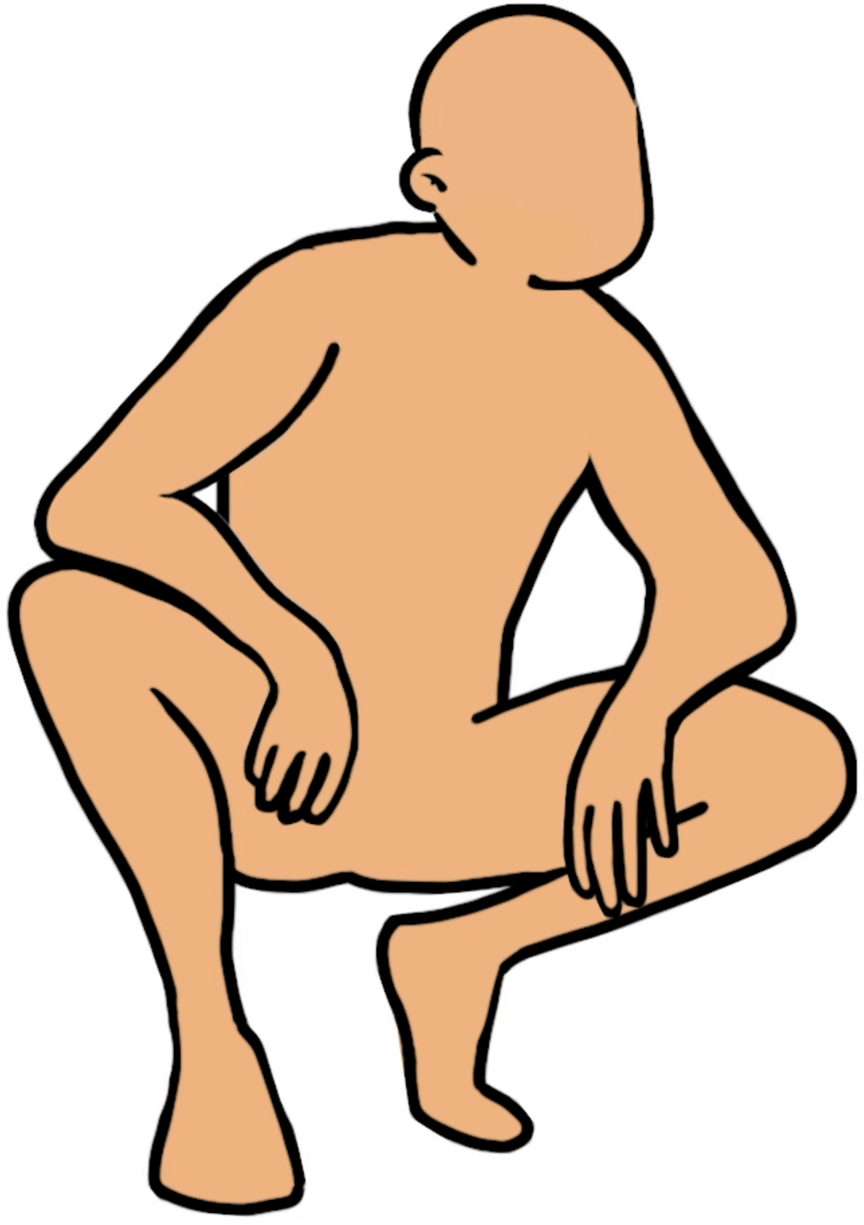
blue-shirt-backarm

polo-shirt-backarm

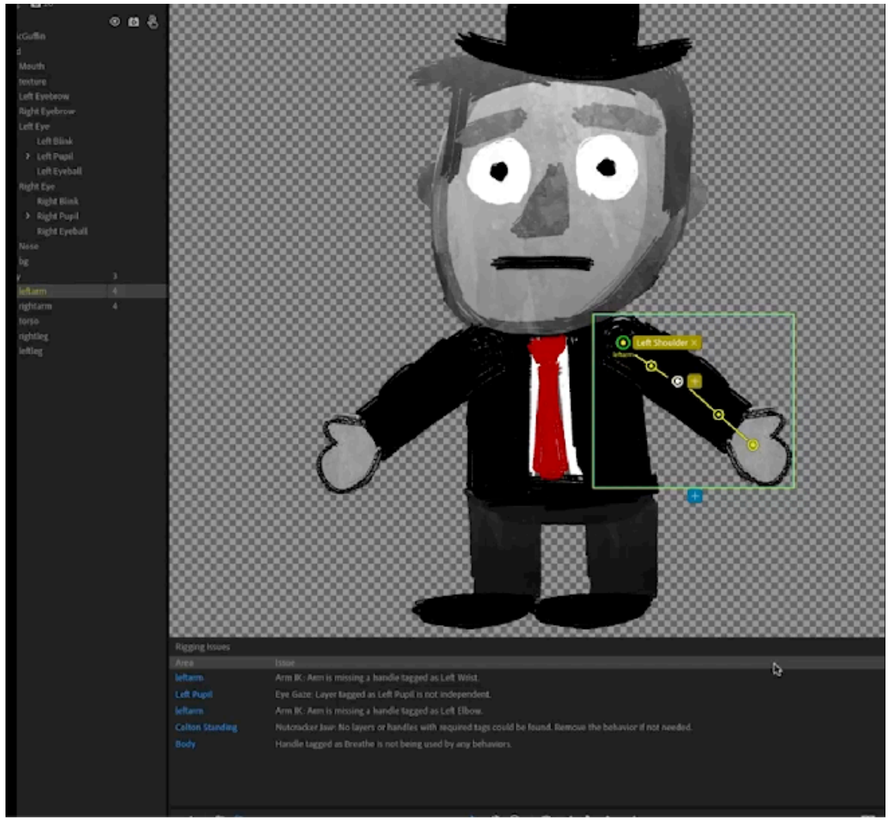
backarm-base

leg-shorts

Doc: 1.54M/88.3M



Related Work

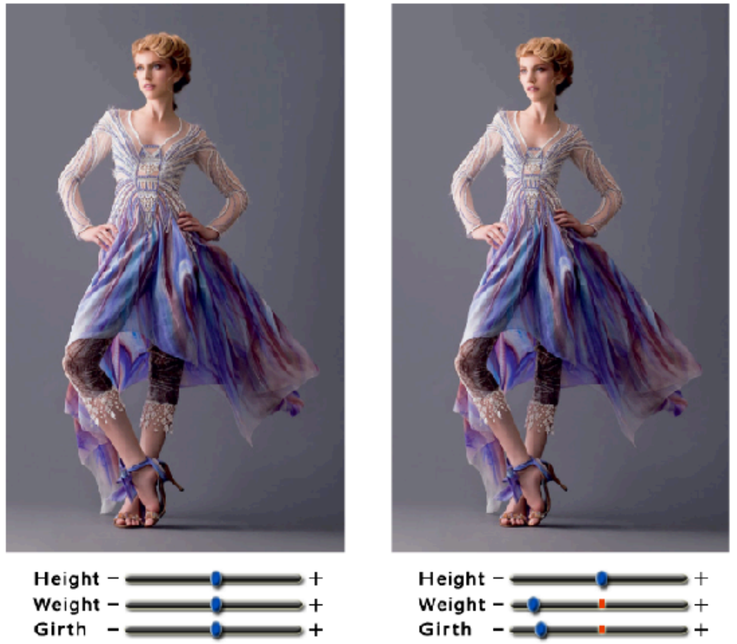


Adobe Character Animator

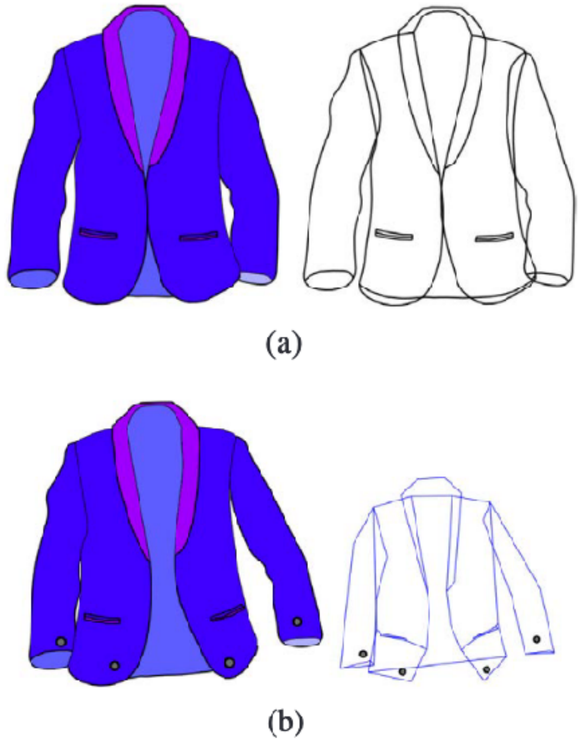
Spline

Related Work

Character animation from 2D pictures and 3D motion data
Hornung et al., 2007



Parametric Reshaping of Human Bodies in Images
Zhou et al., 2010



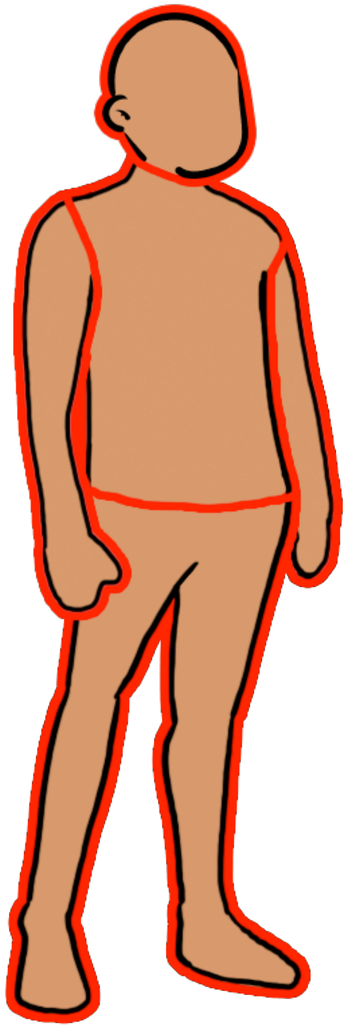
Structure Preserving Manipulation and Interpolation for Multi-element 2D Shapes
Yang et al., 2012

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

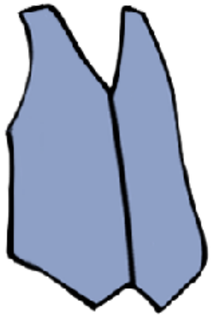
Accessory layers

Top layer

Bowtie

Front sleeve

Shirt collar



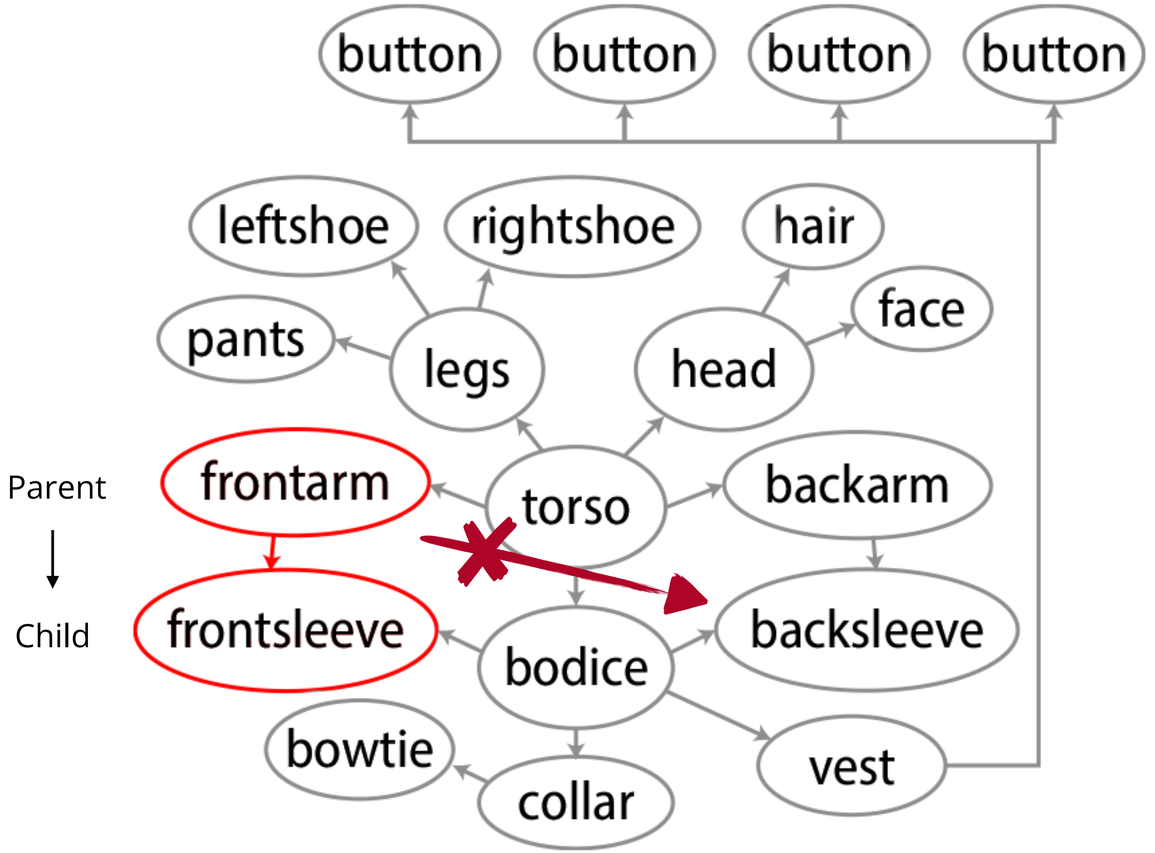
Vest

Shirt bodice

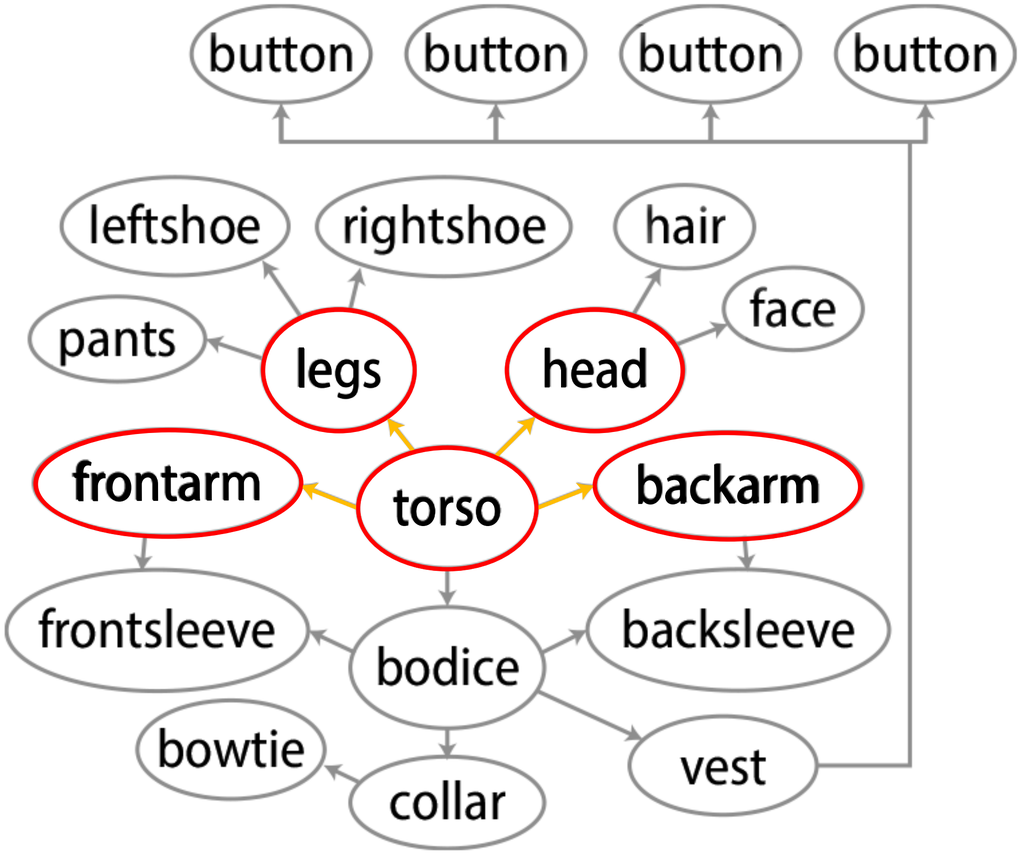
Back sleeve

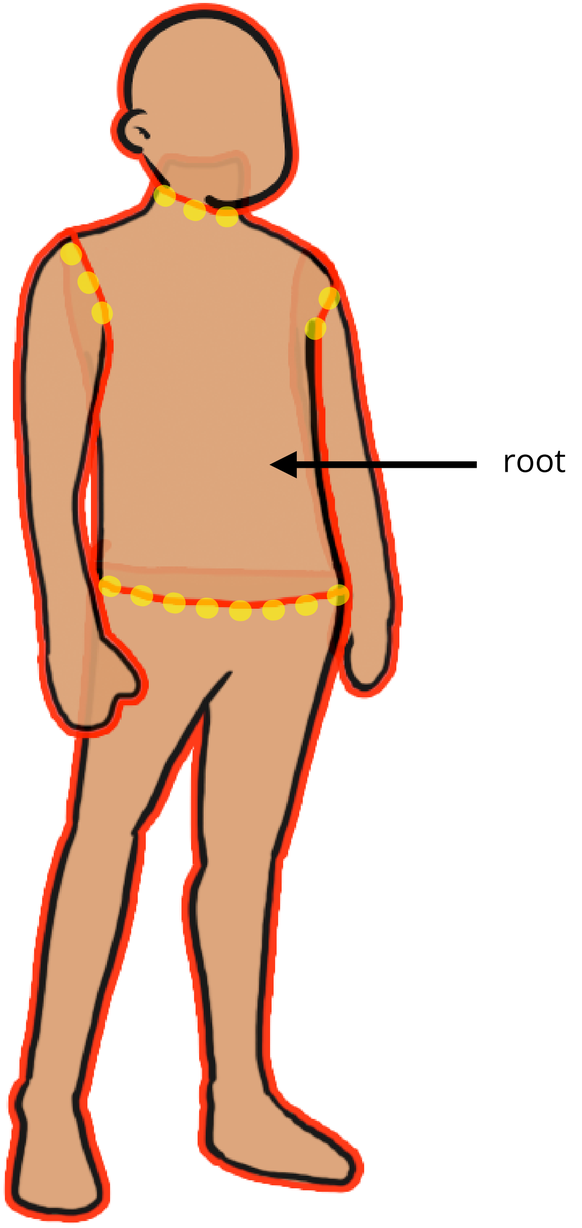
Bottom layer

DAG


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

DAG





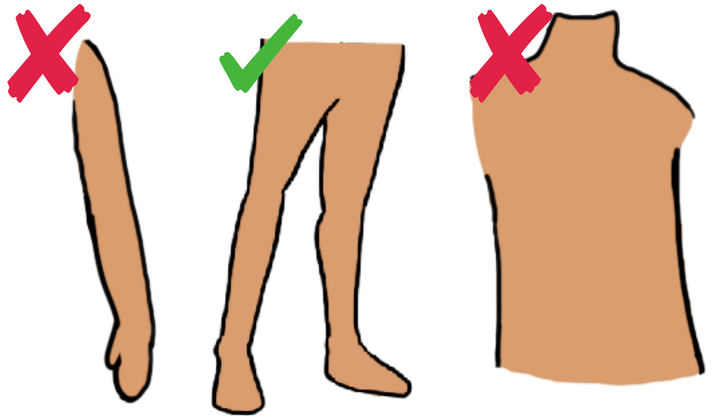
Body layers

For each accessory,

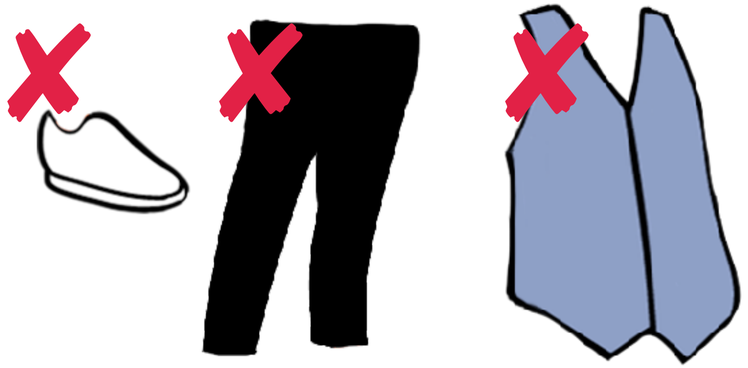
check overlap with every other layer

If just one overlap, make an edge

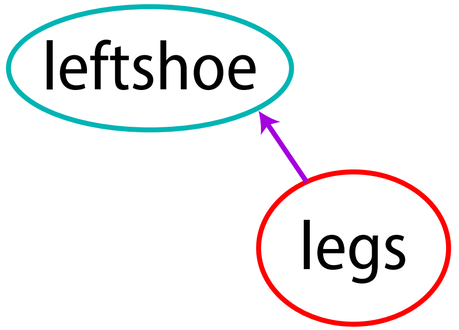
Target accessory (shoe)



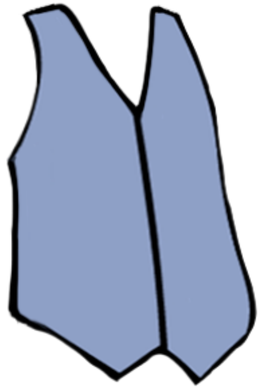
…



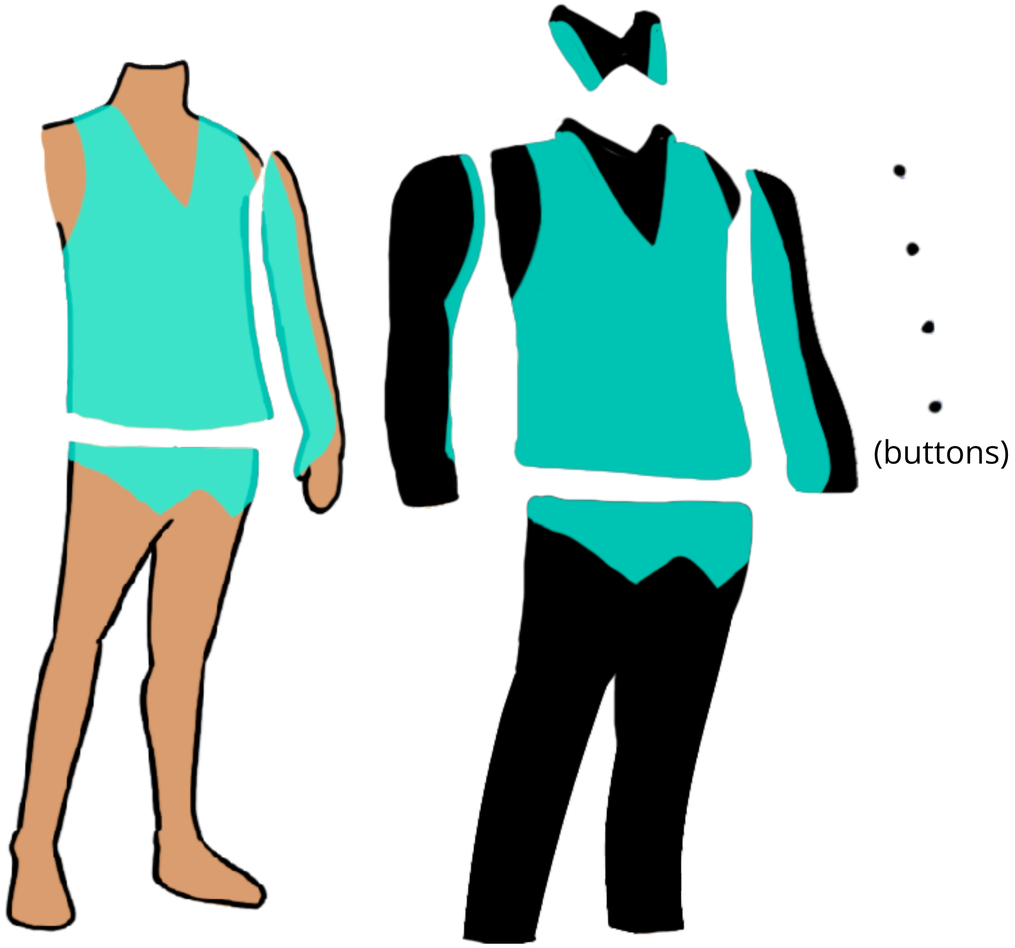
…



For each accessory, multiple overlaps with



Target accessory
(vest)

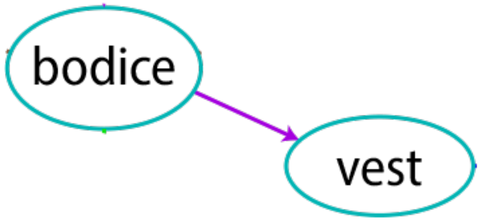


(buttons)

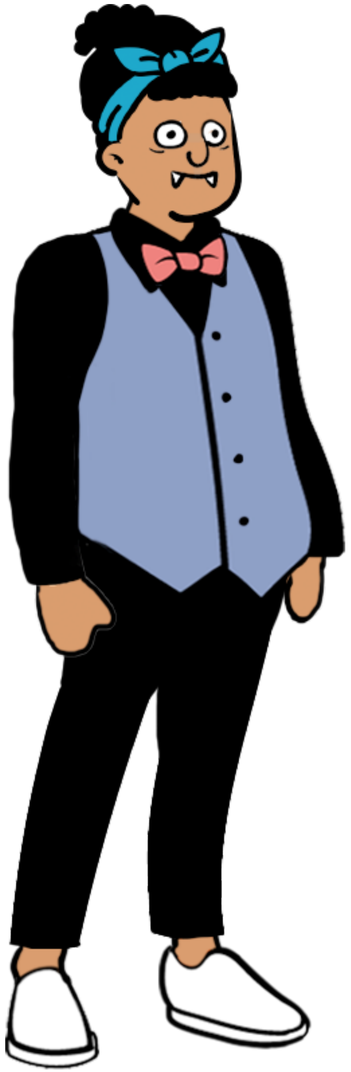
Within 15% of maximum
overlapping area

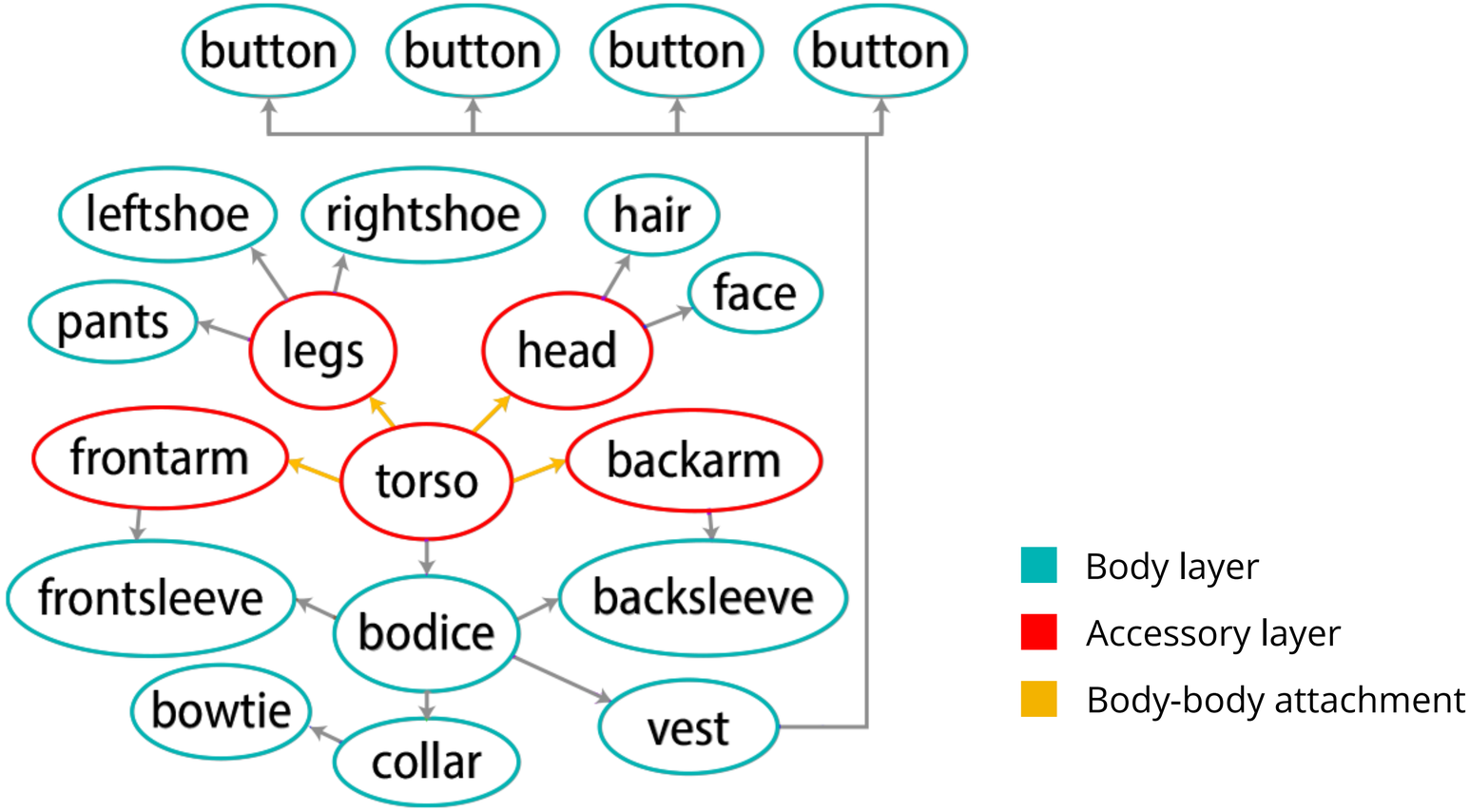
layer order

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

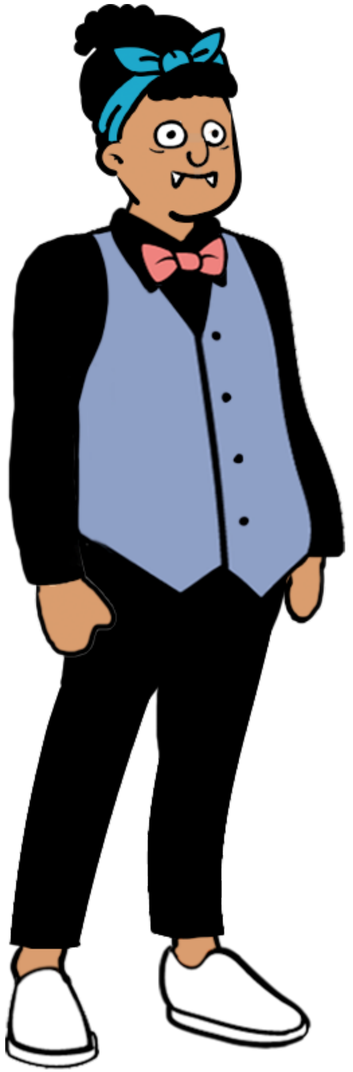


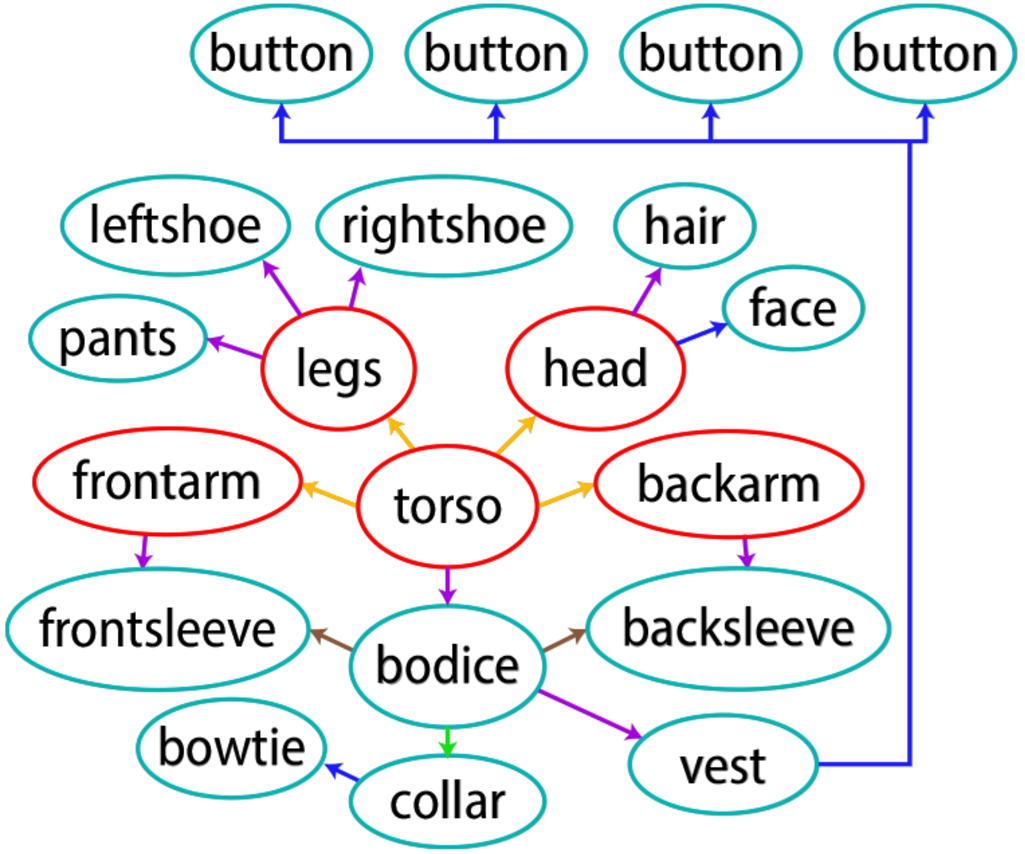
DAG




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



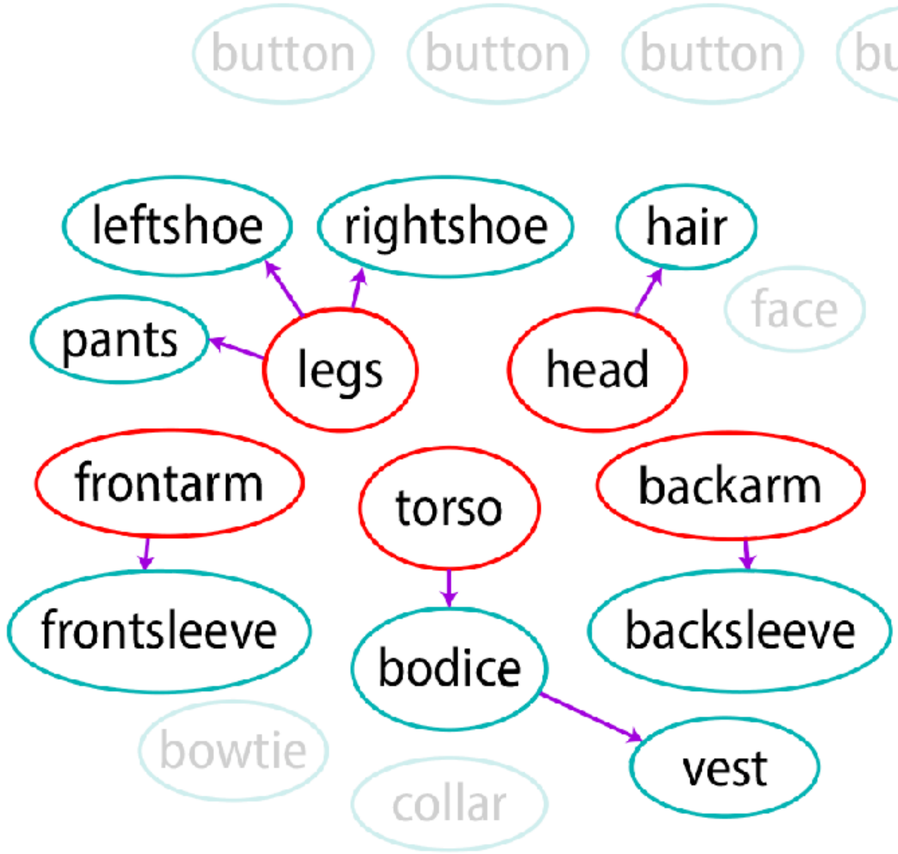


Constraint types

- Occlusion
- Single point
- Boundary
- Region

- Body layer
- Accessory layer
- Body-body attachment

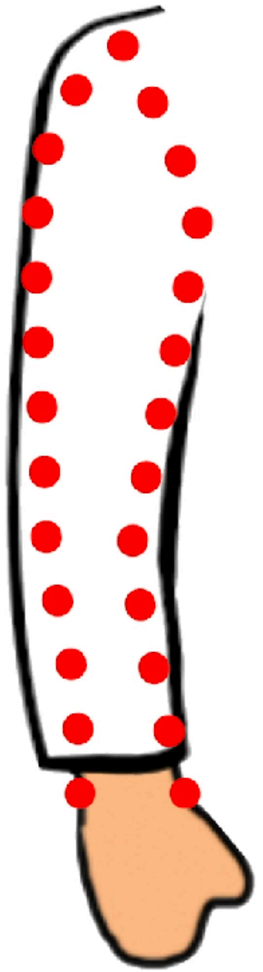
Constraint 1: Occlusion



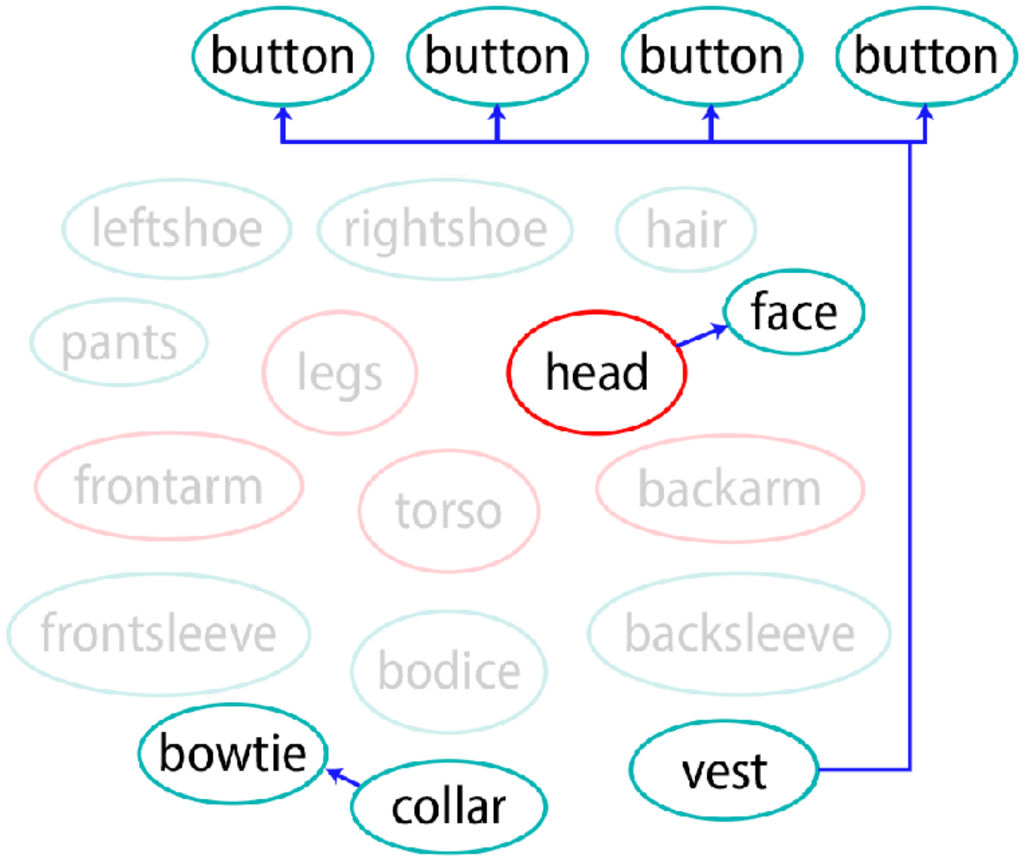
Constraint types

- Occlusion
- Single point
- Boundary
- Region

parent layer boundary

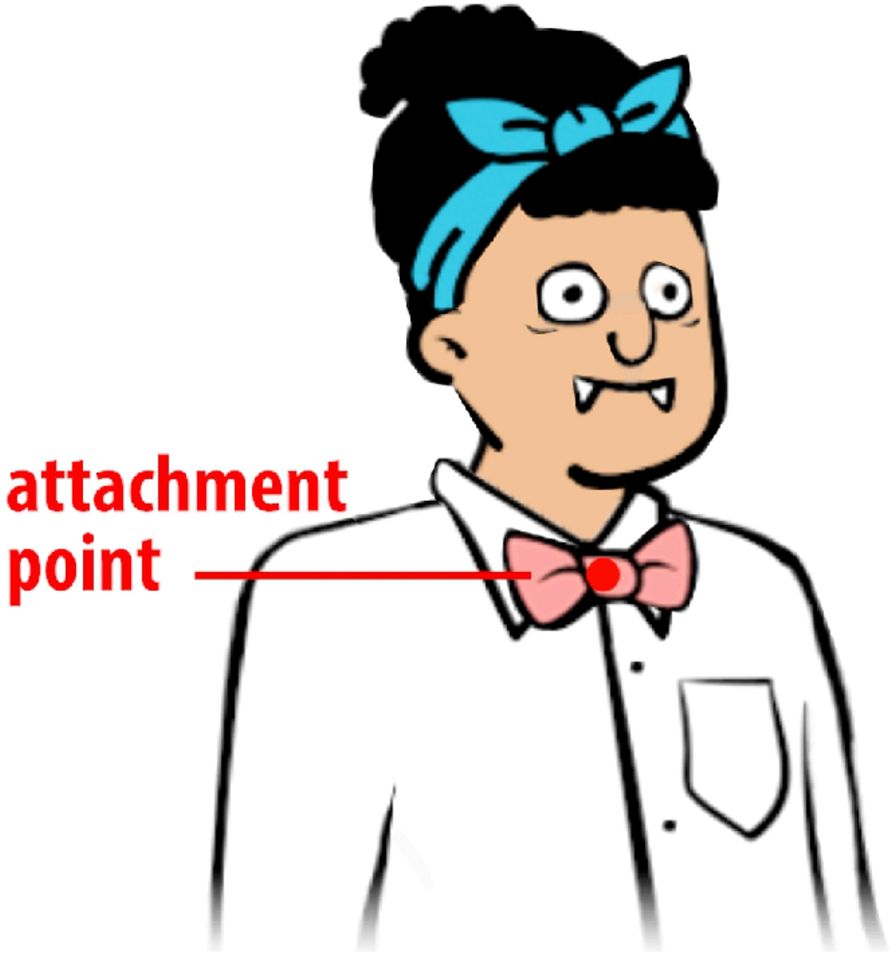


Constraint 2: Single point

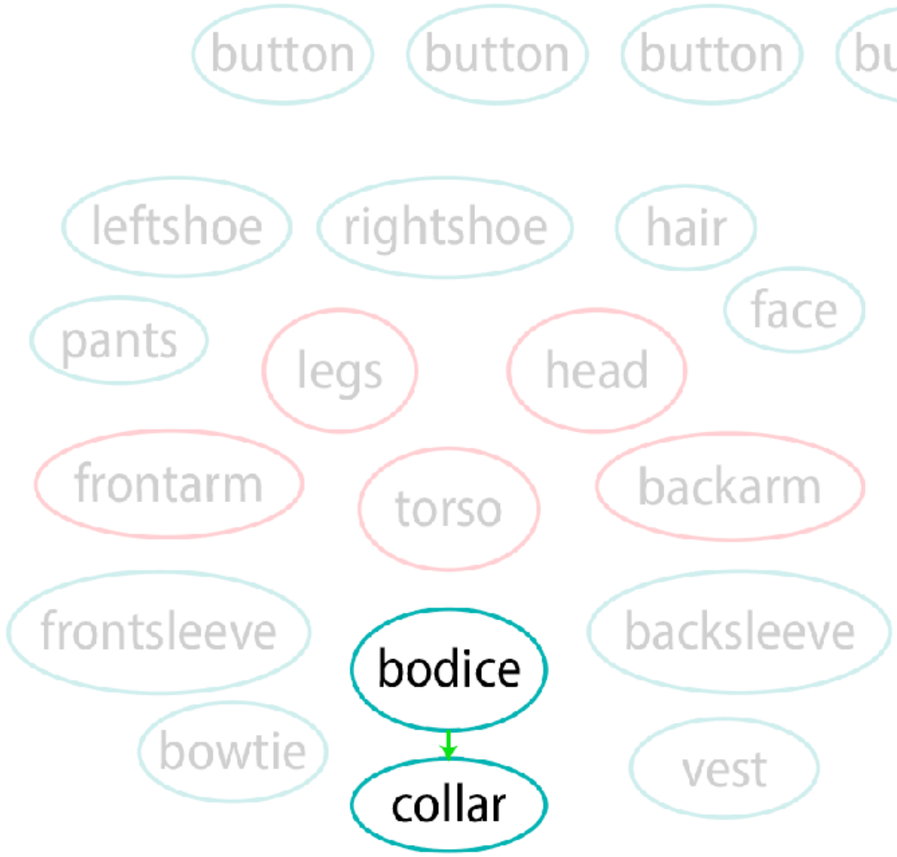


Constraint types

- Occlusion
- Single point
- Boundary
- Region

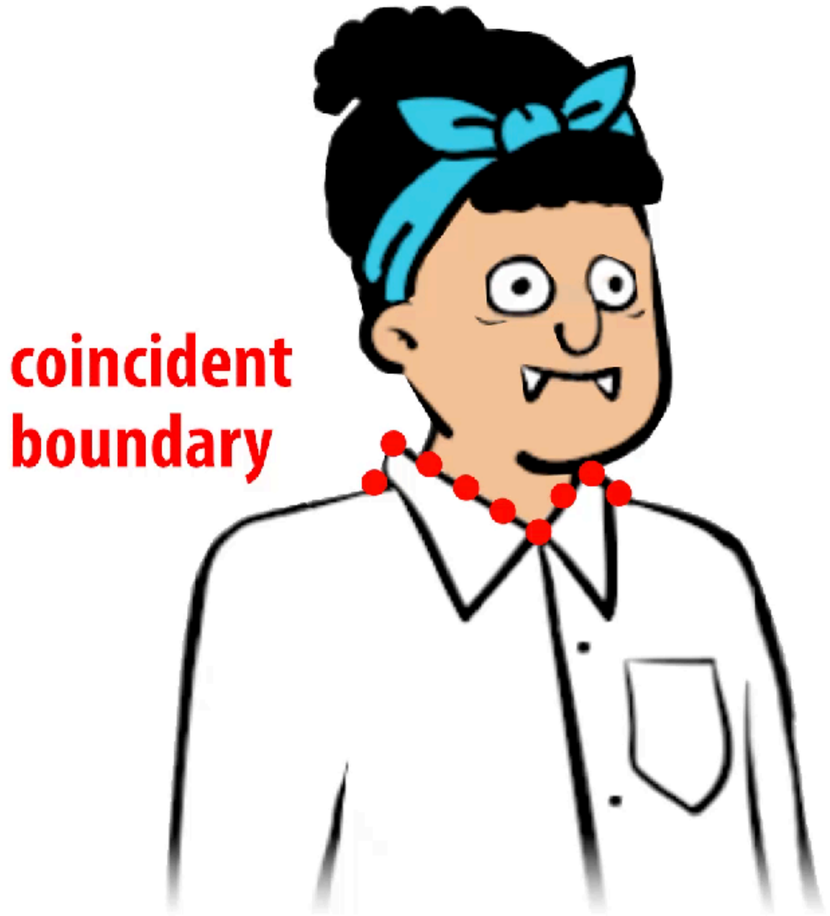


Constraint 3: Boundary

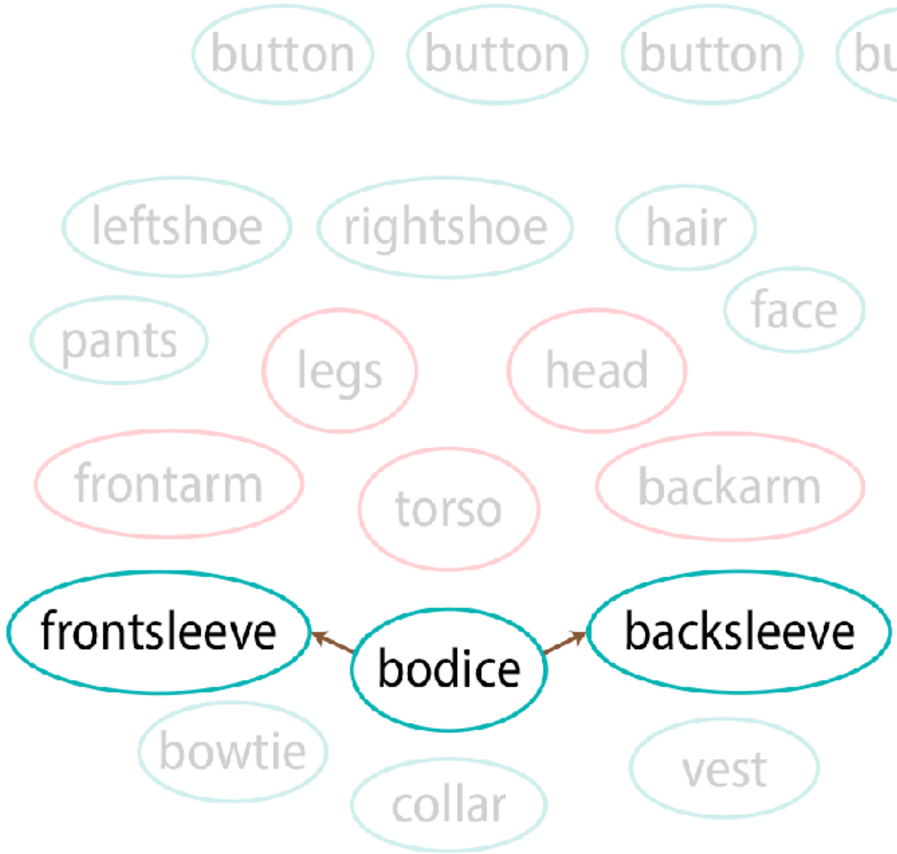


Constraint types

- Occlusion
- Single point
- Boundary
- Region

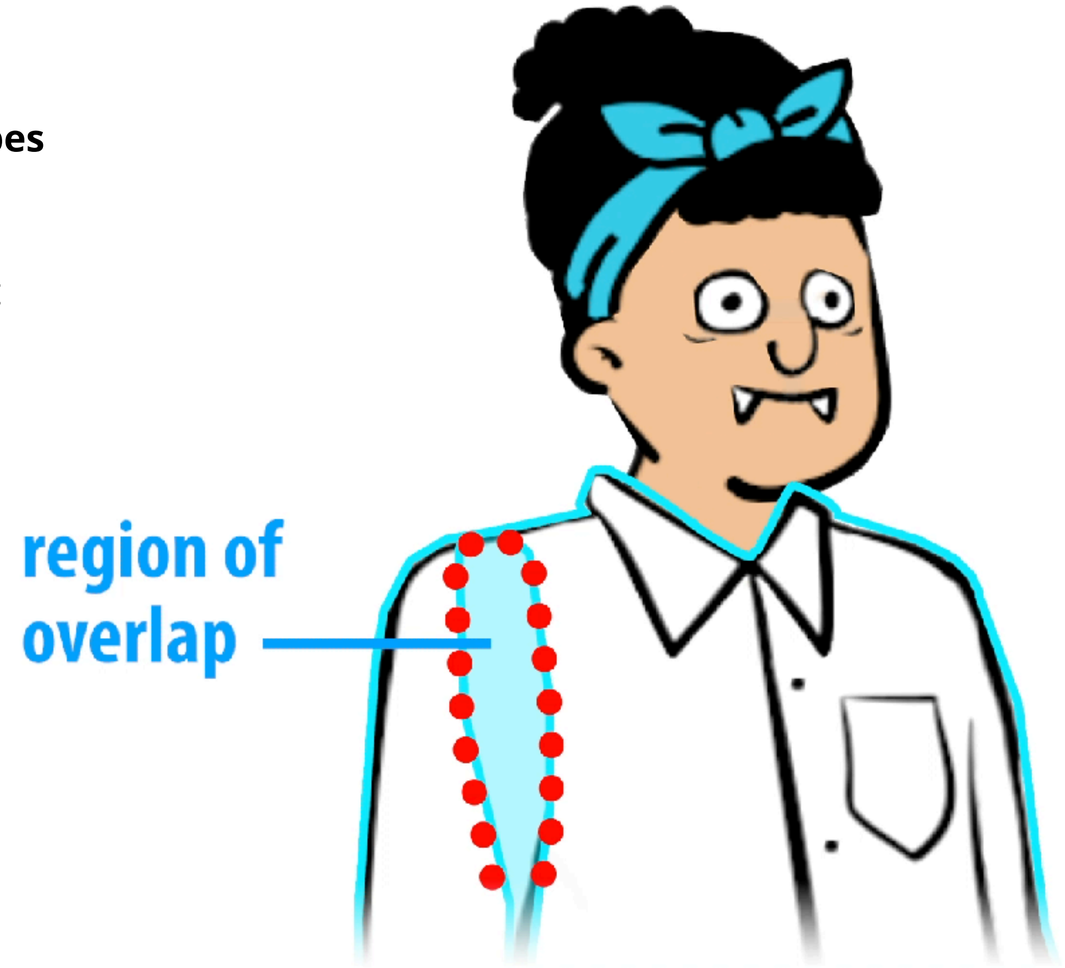


Constraint 4: Overlapping region



Constraint types

- Occlusion
- Single point
- Boundary
- Region

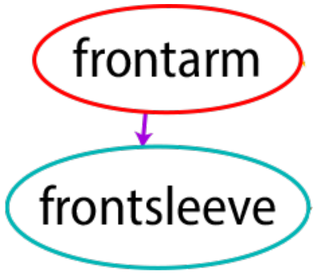


Constraint inference

For each DAG edge,

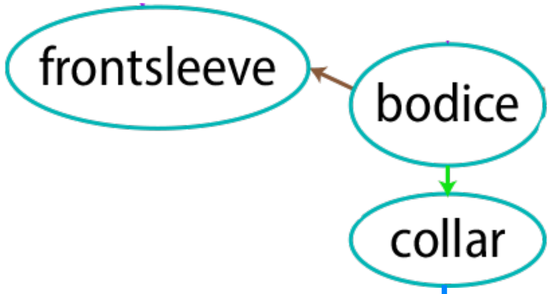
Body-accessory:

- try occlusion

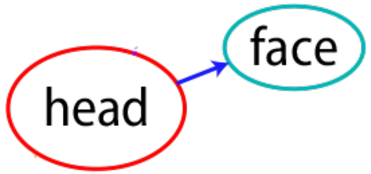


Accessory-accessory:

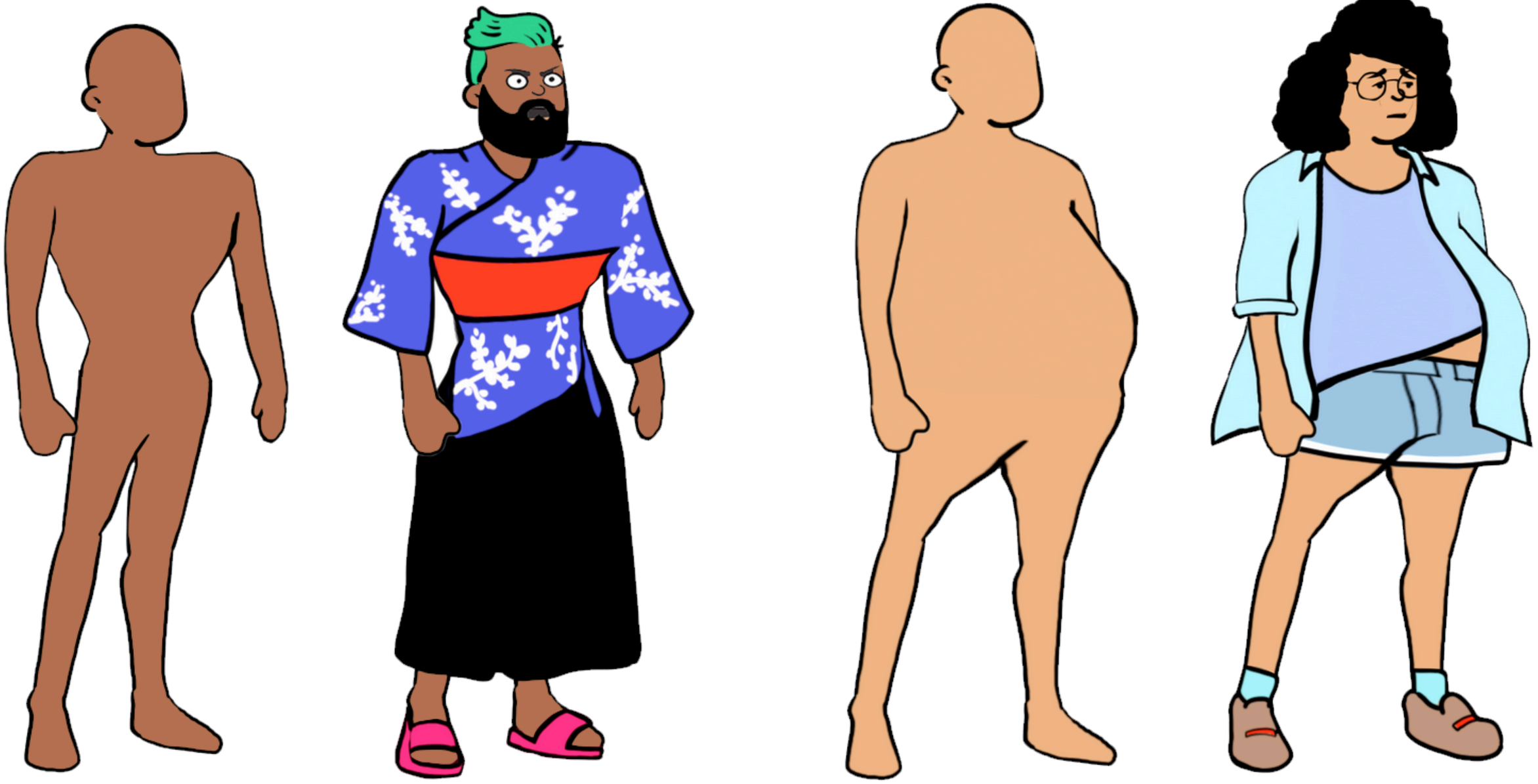
- try boundary & overlap

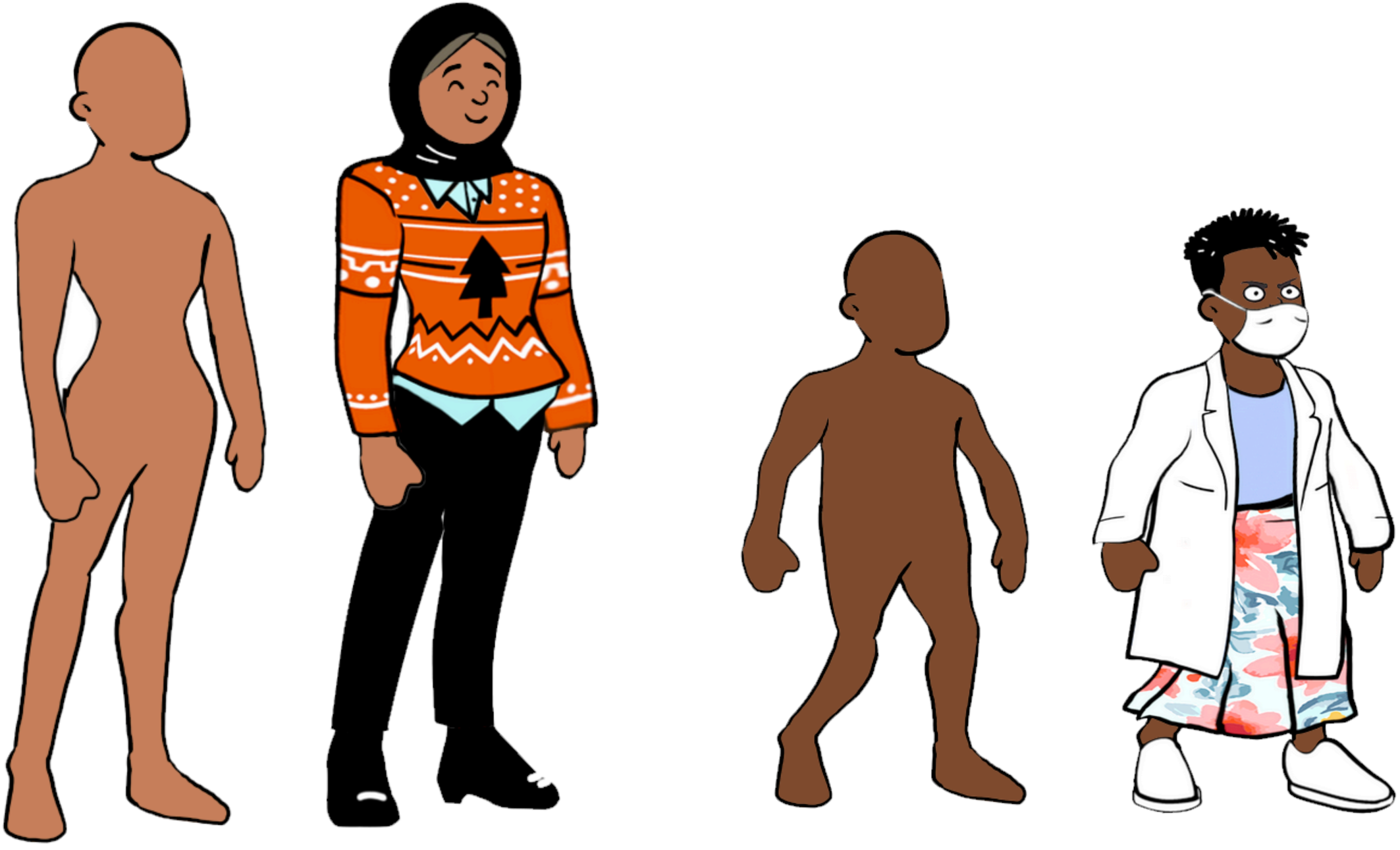


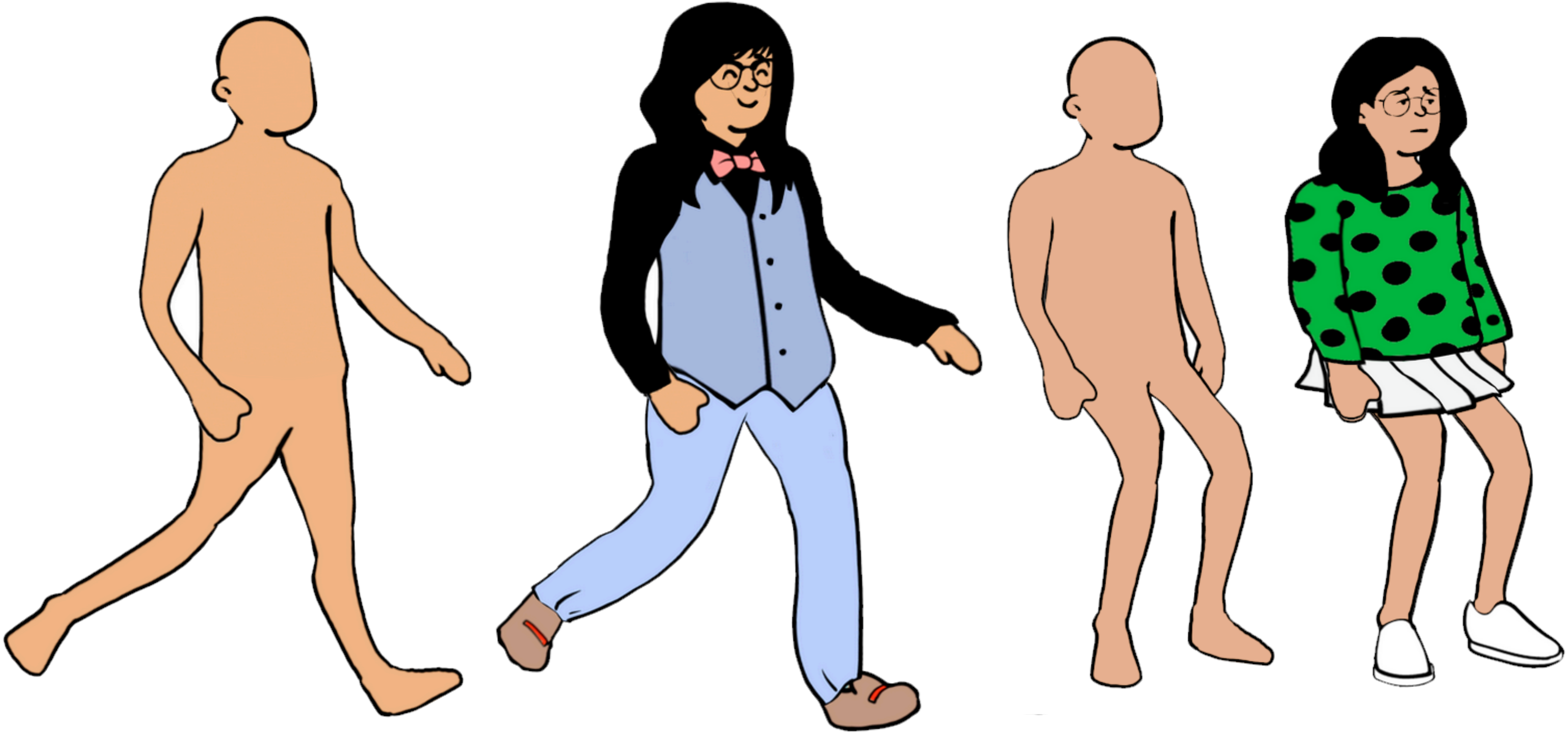
No resultant points: attach at a single point

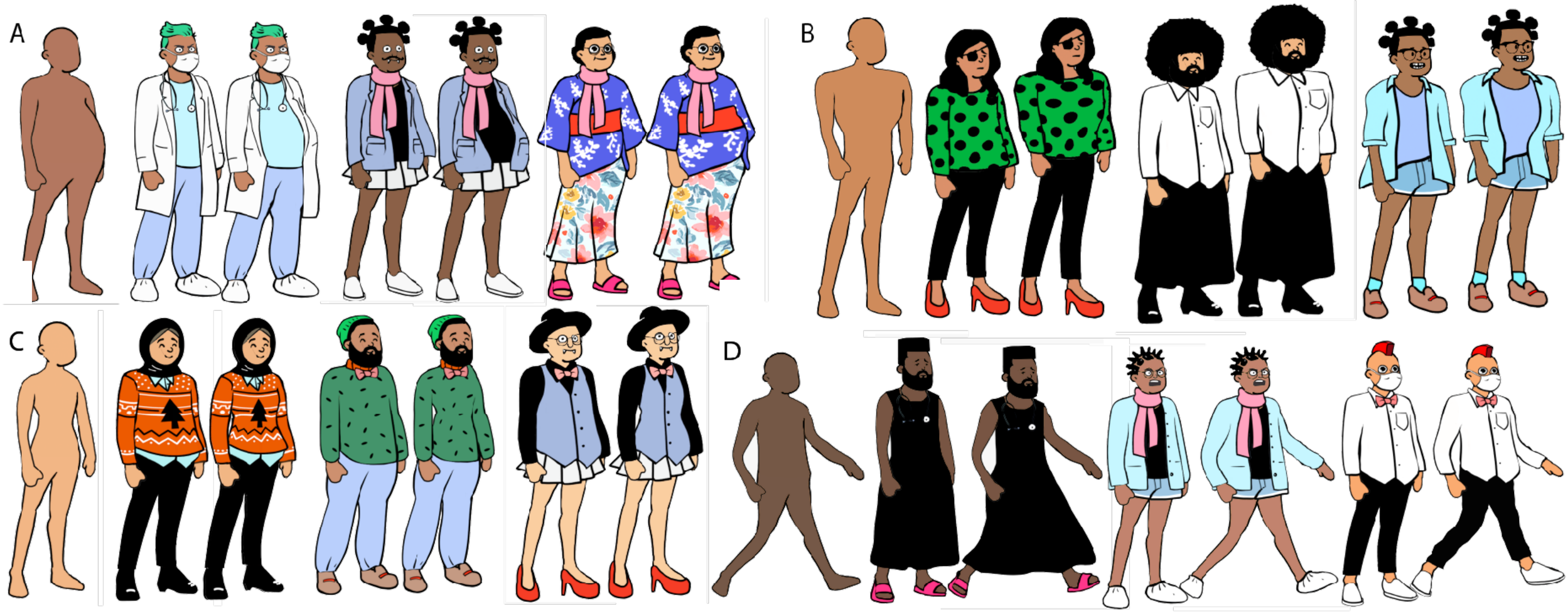


Results








A
B
C
D

Original base layer

Original accessory #1

Original accessory #2

Widened base layer

Widened accessory #1,
single point

Widened accessory #2,
single point

user changes constraint to overlap

Viewer

Workspace

Load Save

Mesh

Load Save

Viewing Options

Draw Options

Overlays

Accessories

Rig filename

cat-1 Head Accessori

blue Hair

necklace Body Accessori

green Shirt

red Eyes

bushy Eyebrows

small.png Nose

open.png Mouth

Layer

Mouth Current Layer

Rig: Contact

Skip accessory contact

Rig: Single Point

Rig: Boundary

Rig: Region

Transform

0 rotation

0 trans x

0 trans y

Sliders

0 shoulder width

0 hip width

0 body height

0 roundness

Widened base layer

Widened accessory #1, single point

Widened accessory #2, single point

Widened base layer

Widened accessory #1, overlapping region

Widened accessory #2, overlapping region

highlights aesthetic differences of constraint choices

Automated Accessory Rigs for Layered 2D Character Illustrations

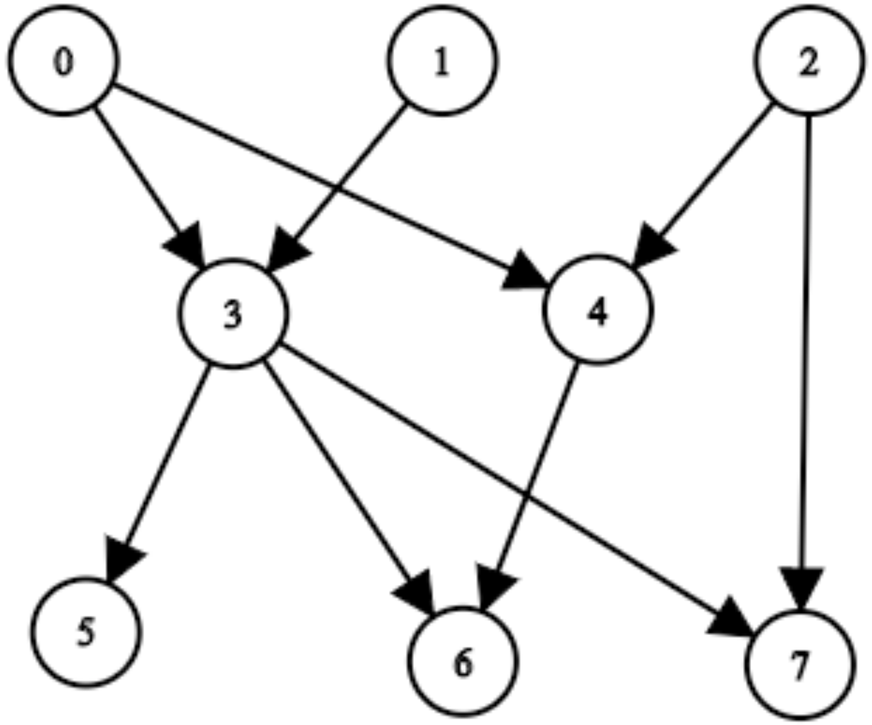
jingyili@cs.stanford.edu
@jingyitweets

Jingyi Li • Wilmot Li • Sean Follmer • Maneesh Agrawala

Leetcode: Find All Acnestors

All ancestors of a node

- Input: DAG with *n* nodes numbered 0 to n-1, and edge adjacency list.
- Output: Return a list `answer`, where `answer[i]` is the list of ancestors of the `i`th node, sorted in ascending order.
- A node u is an *ancestor* of another node v if u can reach v via a set of edges.
- E.g., for this graph, `answer` should be



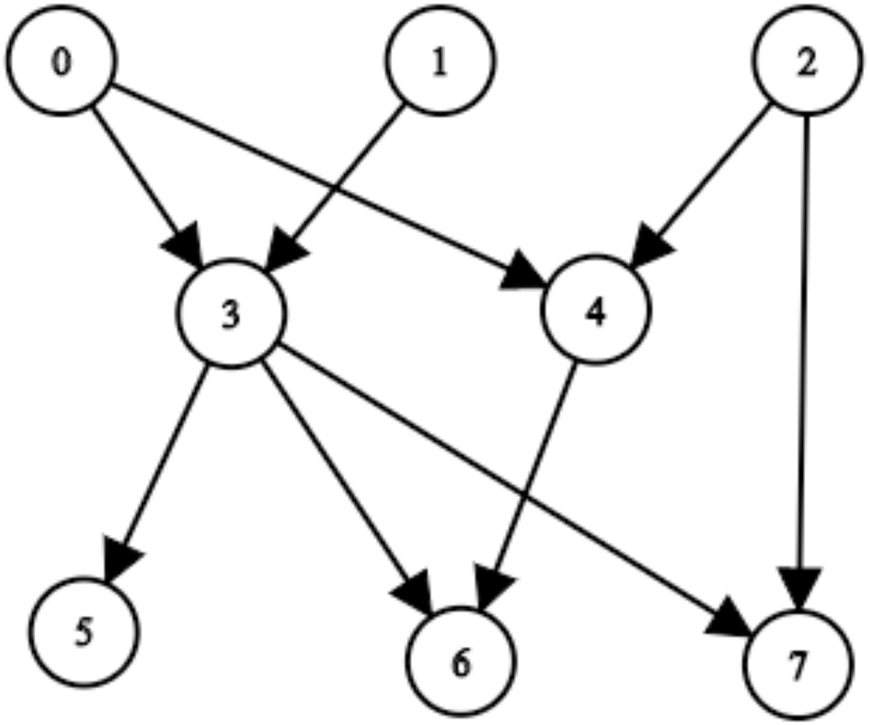

```
[[],[],[],[0,1],[0,2],[0,1,3],[0,1,2,3,4],[0,1,2,3]]
 0    1   2     3       4         5             6               7
```

Write a plain text algorithm.

If you finish that, try writing code.

Naive approach

- For each node, run DFS from every other node to see if it reaches the node. If it does, add to the list.
  - E.g., for 0, run DFS from 1, 2, 3 … we never see 0, so list is empty
  - But for 3, run DFS from 0, which sees 3. DFS from 1, sees 3. DFS from 2, does not see 3…
- Inefficient.
  - For each V vertex, for each potential ancestor U, run DFS O(V+E). O(V) vertices * O(V) ancestors * O(V+E) DFS = O(V^3 + V^2E).




```
[[],[],[],[0,1],[0,2],[0,1,3],[0,1,2,3,4],[0,1,2,3]]
 0   1  2    3      4       5           6          7
```

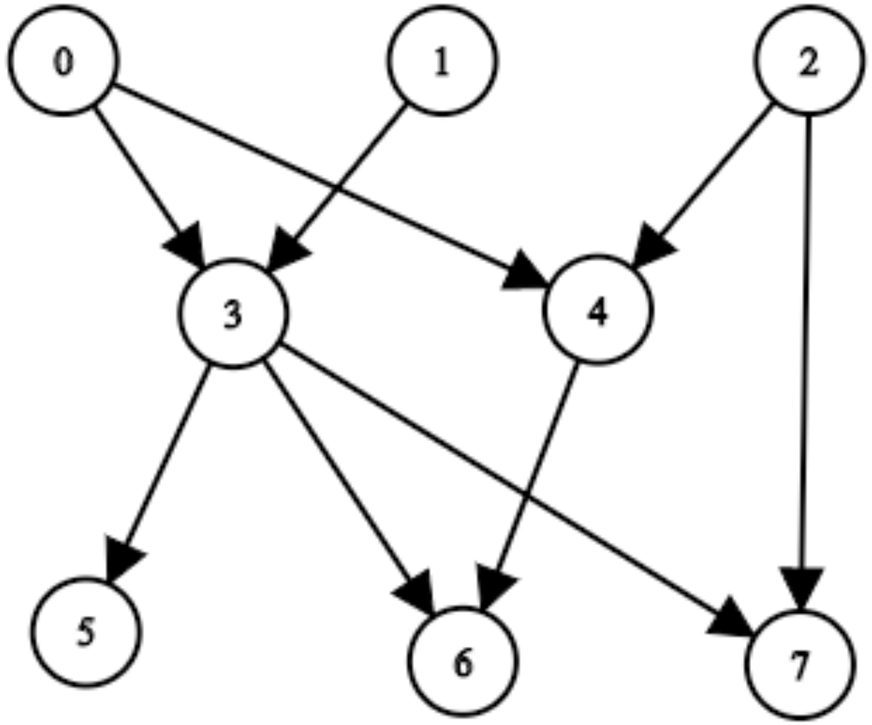
Faster approach

```
DFS post order: 5 6 7 3 4 0 1 2

Topological sort: 2 1 0 4 3 7 6 5
```

- How can we use topological sort? What is the topological sort of this graph given the algorithm we learned last time?
- Process nodes in topological order. Keep track of ancestors for each node. Each node accumulates its ancestors from parent nodes, so no need to recalculate it all.
  - E.g., start at 2. 2 has no ancestors* so answer[2] = []. For 2's children (4, 7), add 2 to its ancestors, and 2's ancestors to its ancestors. answer[4].add(2); answer[7].add(2).
  - Use a queue to pop vertices
  - First time we add an ancestor's ancestors: at node 4, 6 should add 4 as an ancestor, and 4's ancestors (0, 2)
- Run time: O(V+E) for initial topological sort, O(E) for finding source nodes. For each vertex, O(V) to update ancestors. Thus O(V+E) + O(E) + O(V) each vertex * O(V) update ancestors = O(V^2)

* How do we know 2, 1, 0 are source nodes? Go through edges in the adjacency list and keep track of the in degree of each vertex. If 0, it's a source. This takes O(E) time.




```
[[],[],[],[0,1],[0,2],[0,1,3],[0,1,2,3,4],[0,1,2,3]

 0    1  2    3      4       5          6          7
```

Lecture 24 wrap-up

- HW10: On The Road due next Tues 11:59pm
- **Checkpoint 3 in two weeks** (Mon 5/6). This is the last lecture on the checkpoint. If you have SDRC accommodations, please schedule with them.
- Final project midpoint check-in during lab Apr 29: have a demo for at least 1 feature
- Final project due Weds 5/13

Resources

- Conceptual practice problem behind this slide
- Textbook on DAGs (3.3.2): http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf

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.)
- (There are no solutions provided to these conceptual problems)