Lecture 14

Dynamic programming IV: The Bellman-Ford algorithm

Chinmay Nirkhe | CSE 421 Spring 2025

Midterm

- Midterm during class on May 5th in the usual lecture hall
- For with registered services with DRS for alternate testing
  - Glerum Room CSE2 345 starting at 3:30
  - Your responsibility to convey to the proctor your specific alterations
- You are allowed to bring one page of notes with you. Pen and paper exam.
- Midterm will be 1 hour and starts promptly.
- Reducted to **50 points**.

Midterm

- Covers subjects up through the dynamic programming except Bellman-Ford
- Sample midterm for practice problems and length is posted
- Section this week will review problems and strategy
- I'll host a Q&A section about the subject on Thursday May 2nd.

Academic misconduct

- This week, me and my TAs identified a significant number of cases of academic misconduct.
- Most stemmed from using ChatGPT or other LLMs to solve problems.
- **Clear policy: You are not allowed to use ChatGPT or other LLMs.**
- We are now looking through solutions more carefully for academic misconduct and finding many more.
  - You have an obligation to come forward and declare any use.
  - If you do declare it, before I am inclined to not submit a case to CSSC.

Academic misconduct

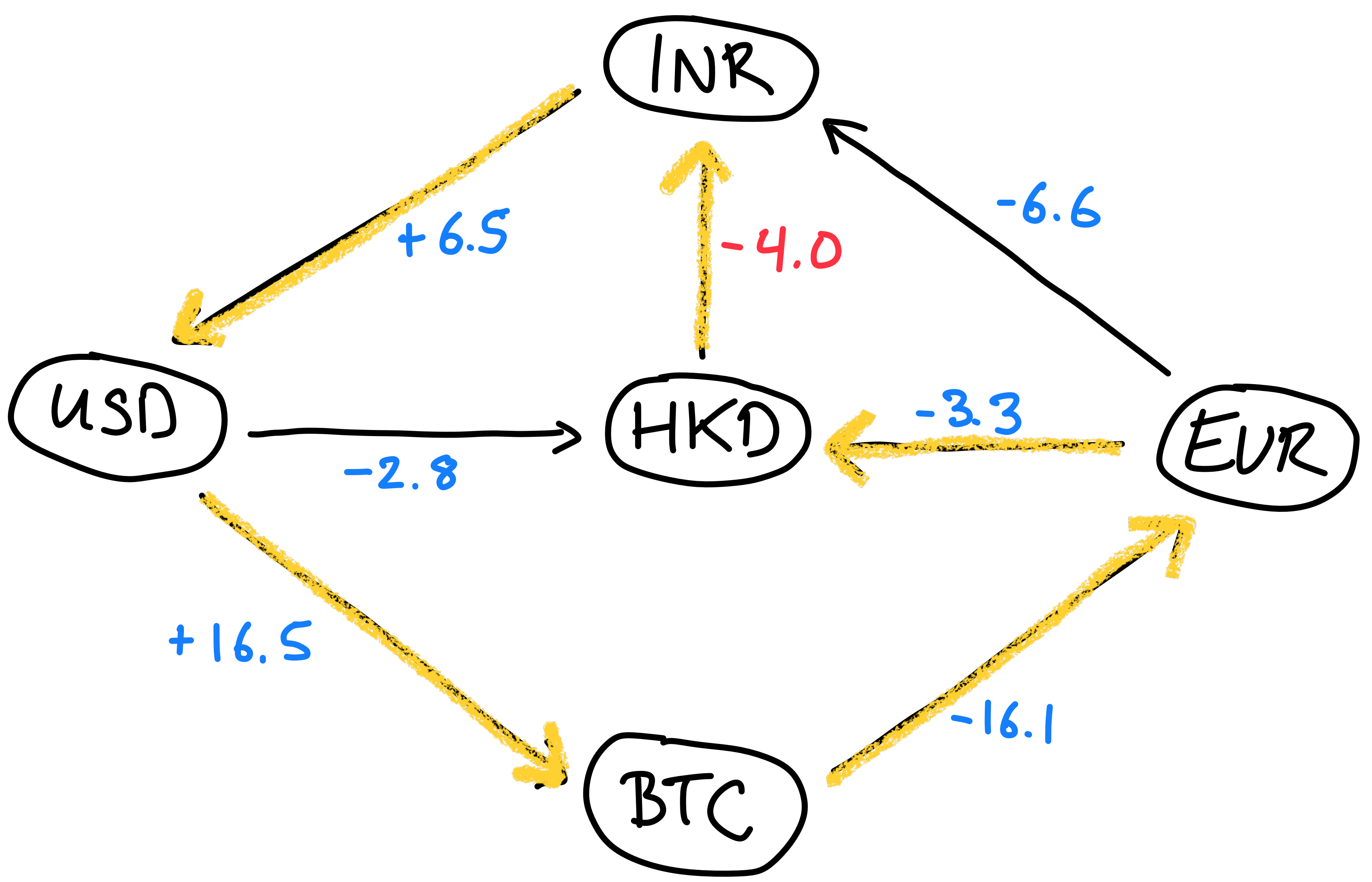
- On problem set 3, we solved a new problem about independent sets. But I didn't explicitly state in the problem statement that d was the avg. degree.
  - That was contextually implied as it was a continuation problem.
  - Many of you understood that it meant avg. degree.
  - ChatGPT didn't. It thought d was max degree.
  - If you used ChatGPT for this, you will likely be getting an email from me real soon. It's best you email me (right now!) admitting your use before I email you.
- This is not the only example. We have others where we have really easily detected use of ChatGPT.

Previously in CSE 421…

Currency exchange

Set edge weight to $\log_2(1/r) = -\log_2(r)$

- Consider the highlighted path from USD to USD:
- Converts 1 USD to $2^{0.8} > 1$ USD
- Constitutes a **negative cycle** in the graph
- In the currency exchange problem, negative cycles represent **arbitrage**
- Since there is a negative cycle, any currency can be converted into any other for arbitrarily cheap as the graph is strongly connected



The Bellman-Ford algorithm

- Dijkstra's is a **greedy** algorithm and suffices to calculate shortest/lightest paths when all weights are non-negative
  - Distances will never need to be recalculated once set
- Bellman-Ford is a **dynamic programming** algorithm for computing shortest path in directed graphs
  - Will run slower than Dijkstra's: $O(mn)$ time versus $O(n + m)$ time
  - Will involve "resetting" distances as the algorithm goes along
  - Bellman-Ford will detect **negative cycles** as shortest paths are undefined if there are negative cycles

Today

Failed attempt #1

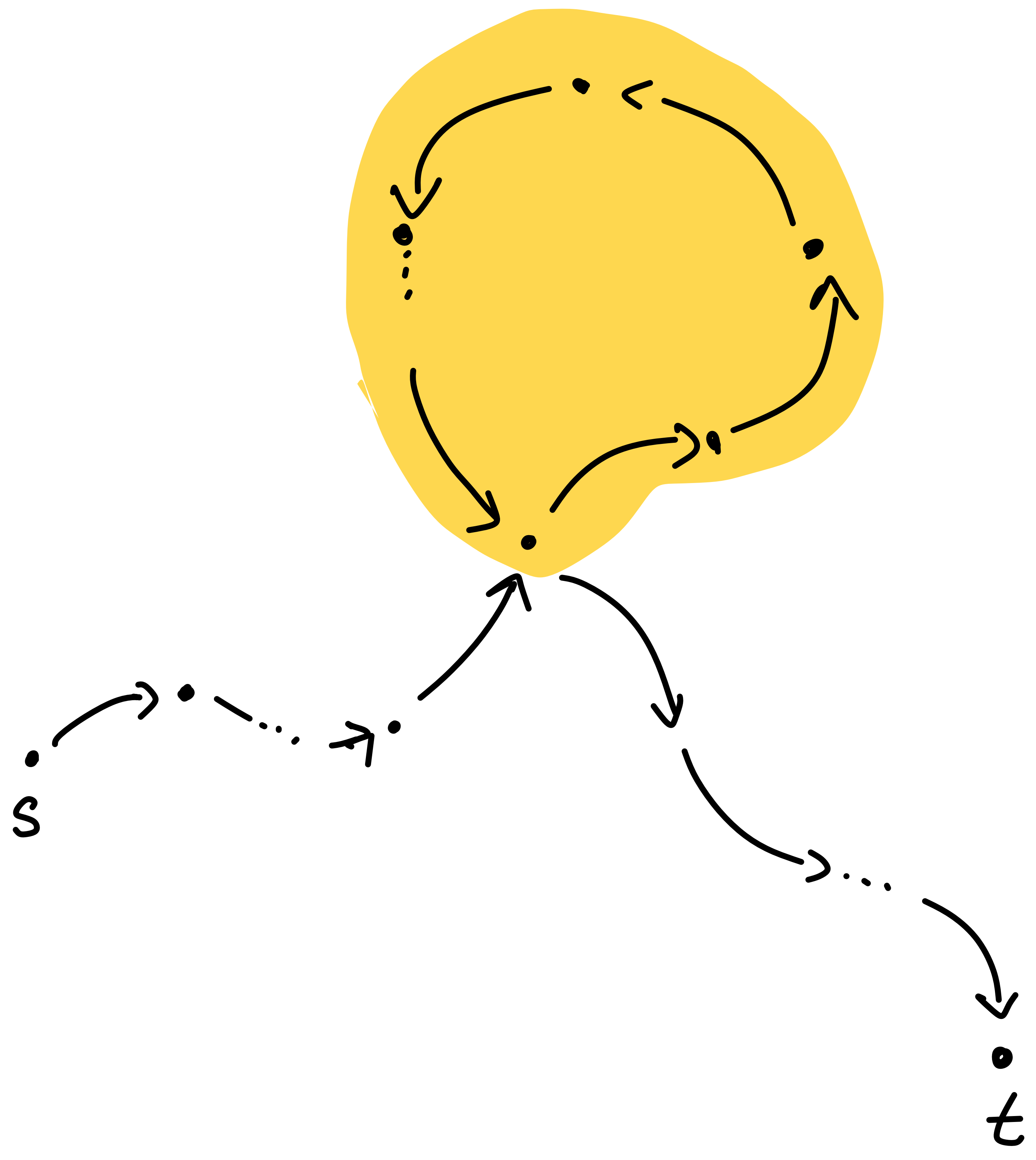
- If a graph has negative weights, let $w_{\min} = \min_{e \in E} w(e)$
- What if we adjusted every edge weight to $w'(e) = w(e) - w_{\min} \geq 0$?
- Can we just run standard Dijkstra's on the adjusted graph?
- **No**. Path weights adjust variably.
  - $w'(p) = w(p) - w_{\min} \cdot |\#$ of edges in $p|$
- Why can we run MST algorithms with negative weights?

Negative weight shortest path

- **Input:** Directed graph $G = (V, E)$ and weights $w : E \to \mathbb{R}$ and a vertex t
- **Output:** For all vertices s, the weight of the shortest path $d(s, t)$
- Note, we are considering shortest paths with respect to the endpoint t
- Its easy enough to convert it to an algorithm for shortest paths with respect to the source

Negative weight shortest path

- **Input:** Directed graph $G = (V, E)$ and weights $w : E \to \mathbb{R}$ and a vertex t
- **Output:** For all vertices s, the weight of the shortest path $d(s, t)$
- **Observation:** If a path $s \rightsquigarrow t$ contains a negative weight cycle, then a shortest path doesn't exist.
- **Observation:** If G has no negative cycles then the shortest path $s \rightsquigarrow t$ is of length $\leq n - 1$.
- **Proof:** A path of length $\geq n$ exists, it has a repeated vertex (i.e. a cycle). That cycle has weight ≥ 0, so removing it only decreases weight. Repeat till path is of length $\leq n - 1$.



Dynamic programming algorithm

- **Definition.** For $i \in \{0, \ldots, n-1\}$, $s \in V$, let $d(i, s)$ be the length of the *shortest path* $s \rightsquigarrow t$ consisting of *at most* i edges
  - Case 1: The shortest path uses $\leq i - 1$ edges. Then

    $$d(i, s) = d(i-1, s)$$

  - Case 2: The shortest path uses exactly i edges. Let u be the first vertex on the path. Then

    $$d(i, s) = w(s, u) + d(i-1, u)$$

Dynamic programming algorithm

- **Definition.** For $i \in \{0, \ldots, n-1\}$, $s \in V$, let $d(i, s)$ be the length of the *shortest path* $s \rightsquigarrow t$ consisting of *at most* i edges
- **DP recursive definition**:

$$d(i,s) = \begin{cases} 0 & \text{if } i = 0 \text{ and } s = t \\ \infty & \text{if } i = 0 \text{ and } s \neq t \\ \min \left\{ d(i-1,s),\ \min\limits_{u: s \to u} w(s,u) + d(i-1,u) \right\} & \text{otherwise} \end{cases}$$

Dynamic programming implementation

(Assuming no negative cycles)

- **Table generation**:
  - Generate table d of size $(n-1) \times n$ and table next of size n
  - Set $d(0,s) \leftarrow \infty$ for $s \neq t$ and $d(0,t) \leftarrow 0$
  - For $i \leftarrow 1$ to n
    - Set $d(i,s) \leftarrow d(i-1,s)$.
    - For each edge $(s \to u) \in E$
      - If $w(s,u) + d(i-1,u) < d(i-1,s)$,
        - Set $d(i,s) \leftarrow w(s,u) + d(i-1,u)$ and $\mathrm{next}(s) \leftarrow u$
- **Path recovery**: Follow $\mathrm{next}(\cdot)$ from s until it reaches t.

Space saving techniques

- The end result is a DAG mapping paths from every vertex s to the sink t
- The entries of $\mathrm{next}(\cdot)$ list the edges in the path
- $d(i, s)$ only depends on entries $d(i-1,\cdot)$. Rows $i-2,\ldots,1$ can be discarded.

Better DP implementation
(Assuming no negative cycles)

- **Table generation:**
  - Generate table d of size n and table next of size n
  - Set $d(s) \leftarrow \infty$ for $s \neq t$ and $d(t) \leftarrow 0$
  - For $i \leftarrow 1$ to n and edge $(s \to u) \in E$
    - If $w(s,u) + d(u) < d(s)$,
      - Set $d(s) \leftarrow w(s,u) + d(u)$ and $\mathrm{next}(s) \leftarrow u$
- **Path recovery:** Follow $\mathrm{next}(\cdot)$ from s until it reaches t.

Even more trimming

- If $d(u)$ doesn't decrease in round i, then we don't need to consider any edges $s \to u$ in round $i+1$ as the best paths through u have already been considered
- Keep a list Q of vertices updated in the previous round and only update edge $s \to u$ if u was in Q

Even better DP implementation
(Assuming no negative cycles)

- Compute the reverse adjacency list: For every $u \in V$, $\mathrm{pre}(u) = \{s : s \to u\}$.
- Generate tables d, next of size n with $d(s) \leftarrow \infty \ \forall \ s \neq t$ and $d(t) \leftarrow 0$
- Initialize counter $i \leftarrow 0$ and generate a queue $Q \leftarrow \{t, \perp\}$.
- While $i < n$
  - Pop u off the queue Q.
  - If $u = \perp$, increment $i \leftarrow i + 1$ and push $\perp$ to Q.
  - Else, for each $s \in \mathrm{pre}(u)$,
    - If $w(s, u) + d(u) < d(s)$, set $d(s) \leftarrow w(s, u) + d(u)$ and $\mathrm{next}(s) \leftarrow u$
    - Push s into queue Q.

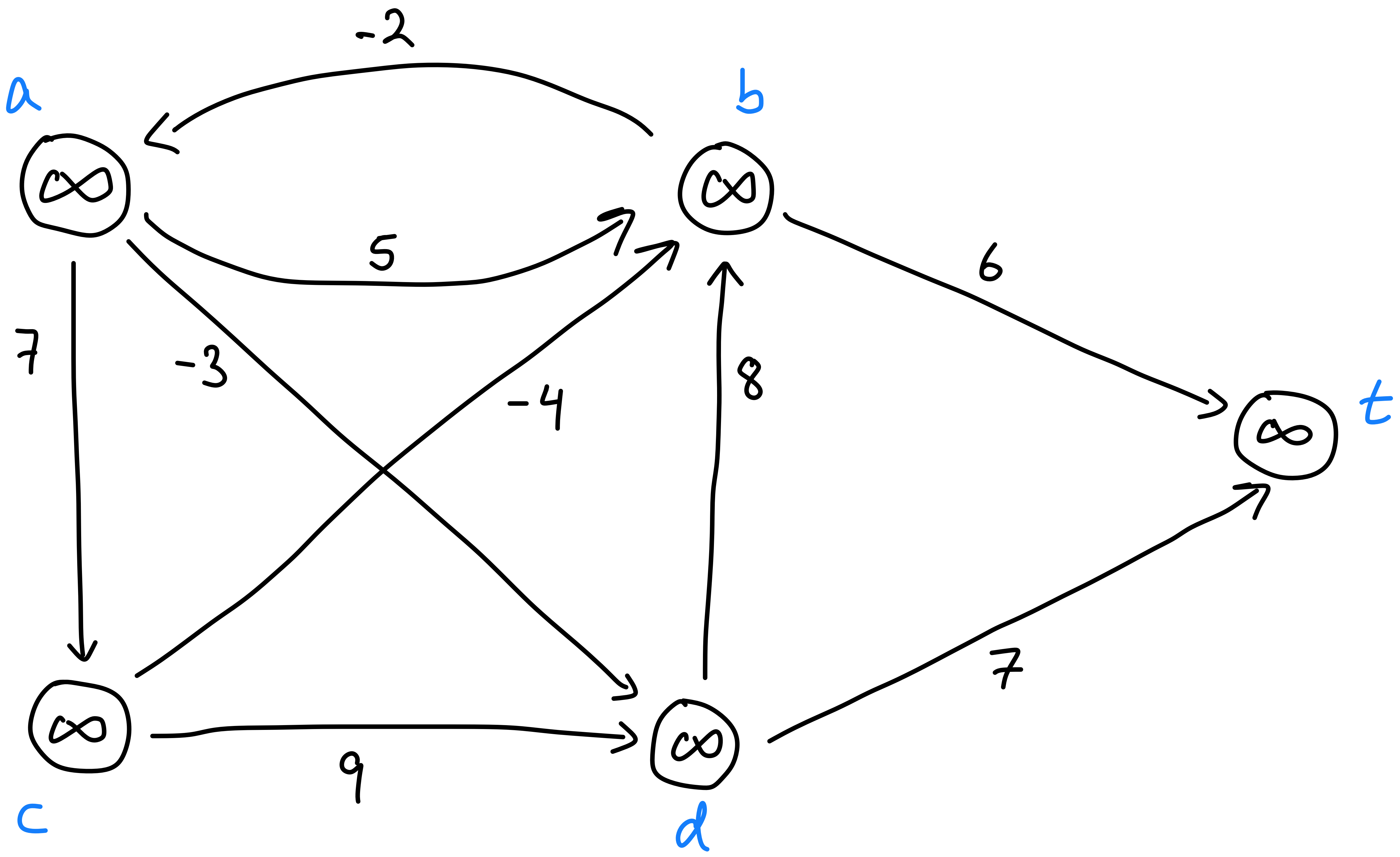
everytime $\perp$ is seen in queue,
we've done one iteration of BF.
We need to do $n-1$.

Bellman-Ford properties

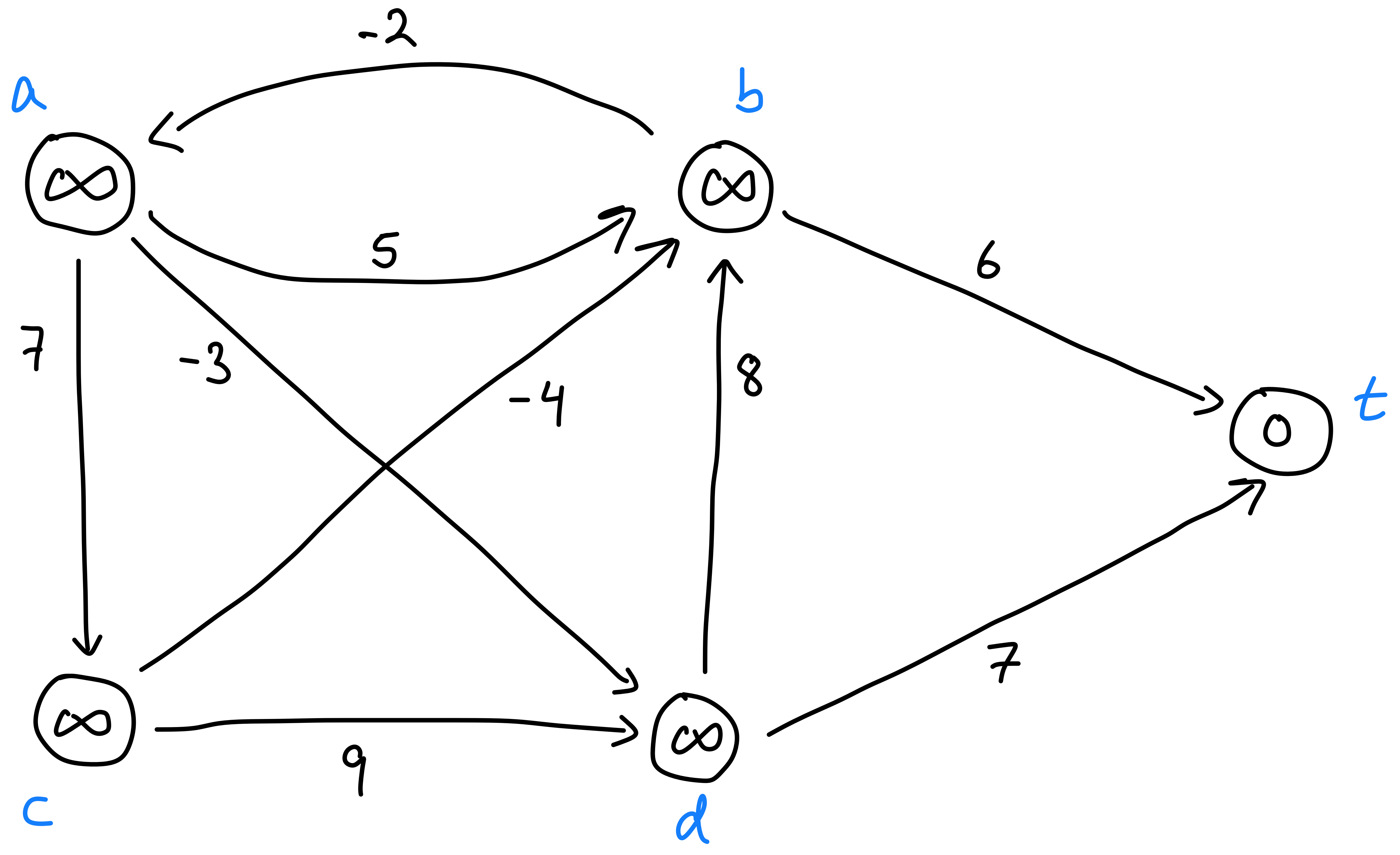
- **Theorem**: Throughout the algorithm, $d(s)$ is the length of some path and that path has weight less than the lightest path of $\leq i$ edges after i rounds of updates

- **Impact:** Space decreases to $O(n+m)$ but runtime is still $O(nm)$ in the worst case. In practice, the runtime is much faster!

Bellman-Ford example



Bellman-Ford example

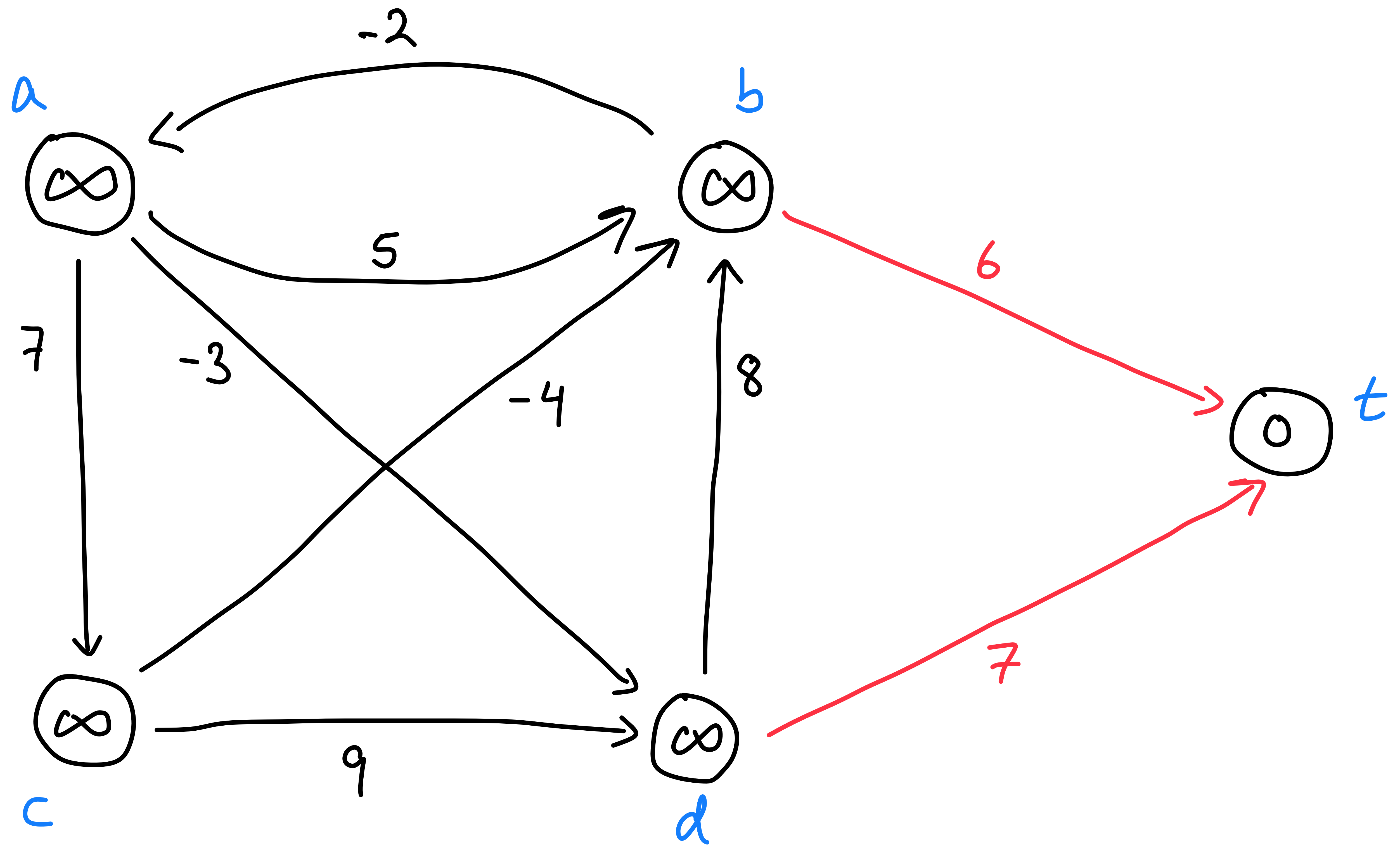


Queue

t

$\perp$

Bellman-Ford example

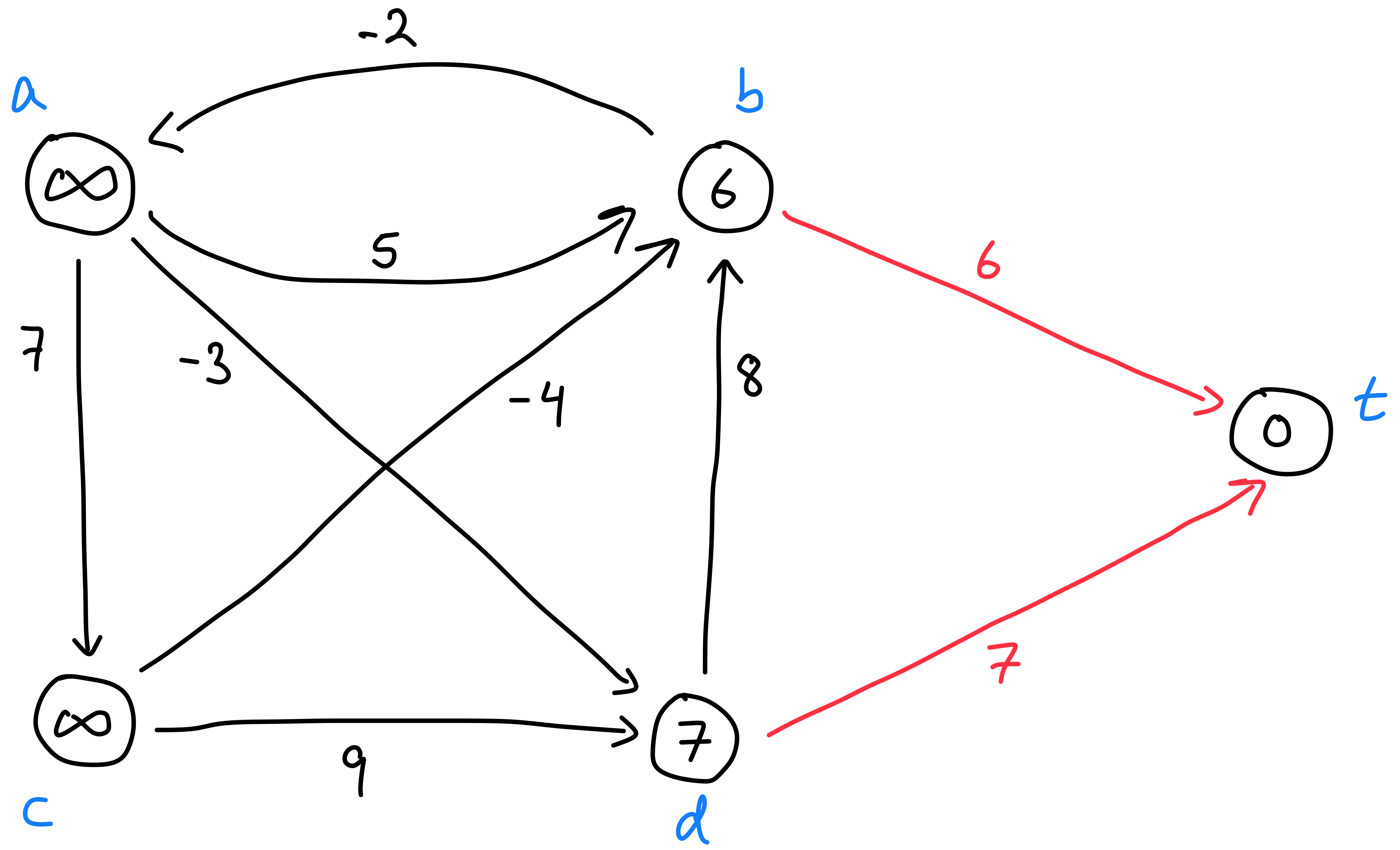


Queue

t

$\perp$

Bellman-Ford example



Queue

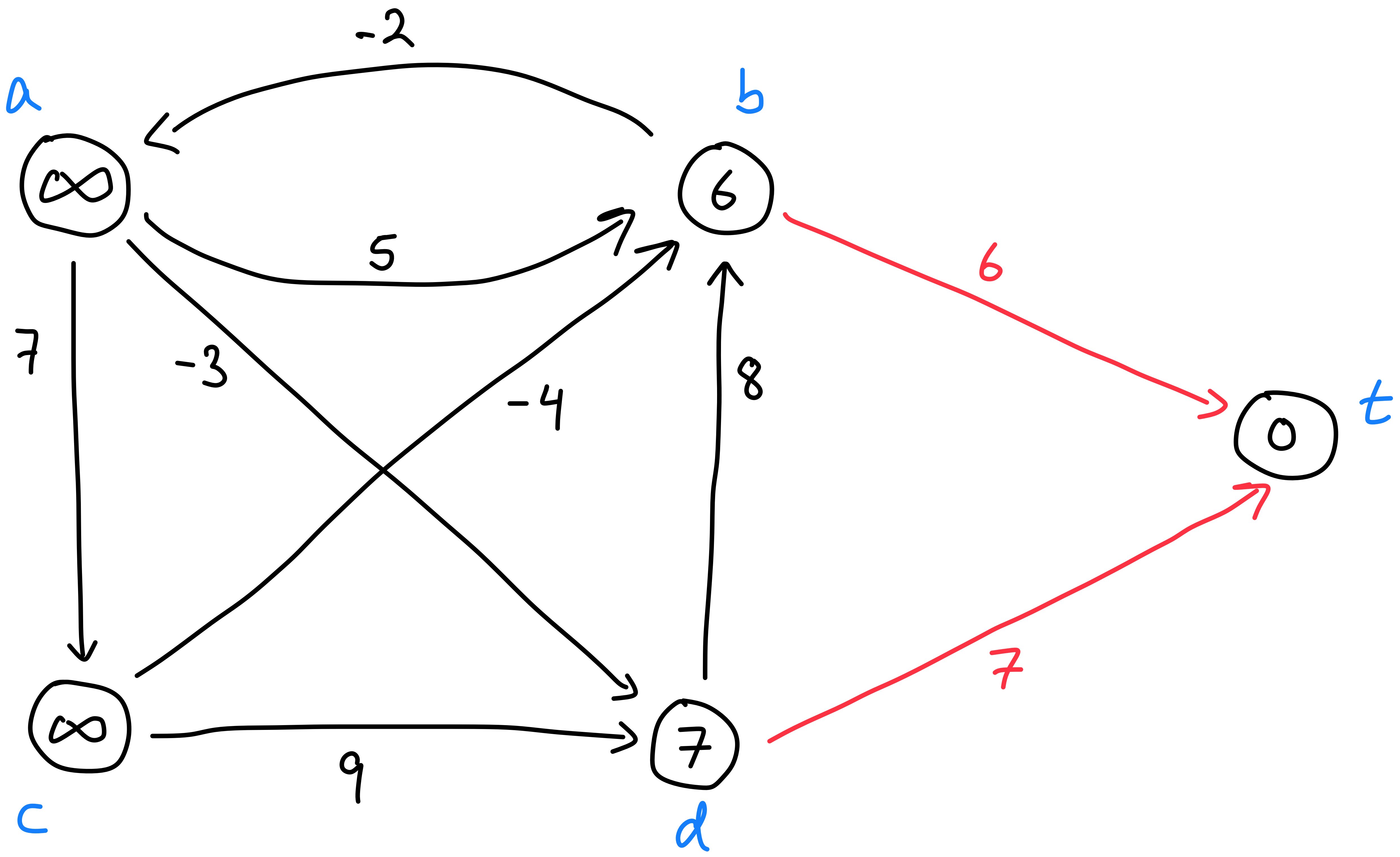
~~t~~

⊥

b

d

Bellman-Ford example



Queue

~~t~~

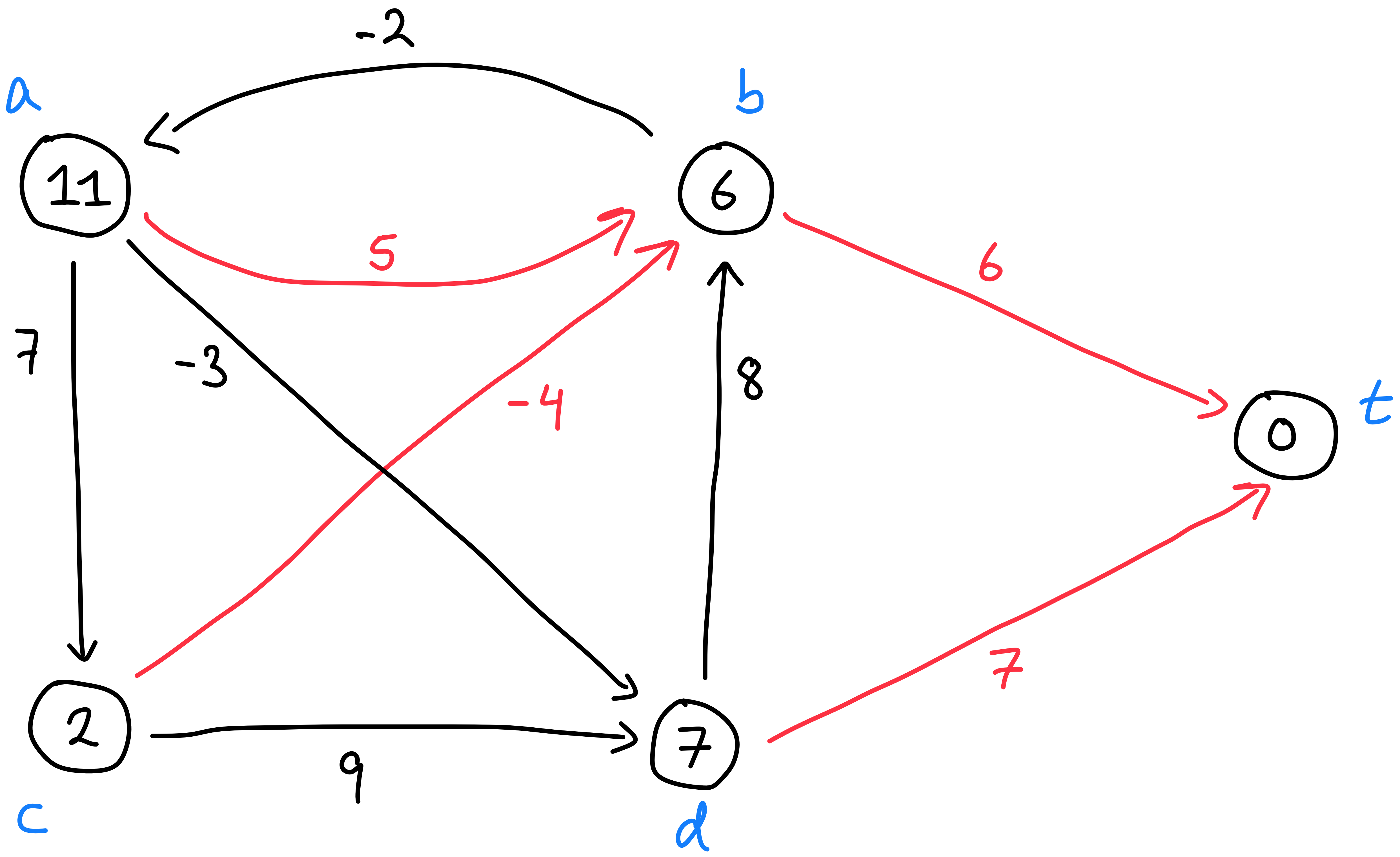
~~$\perp$~~

b

d

$\perp$

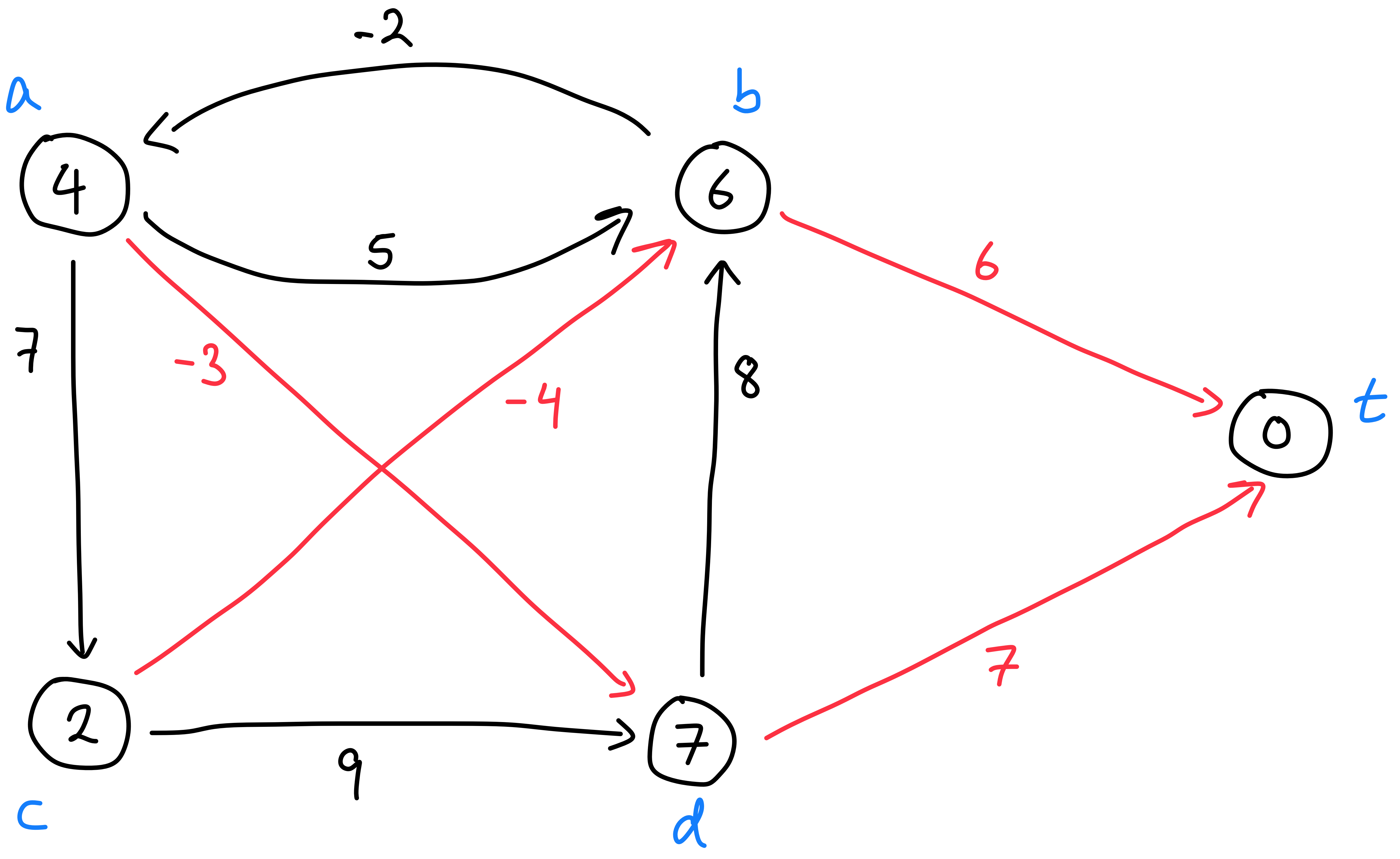
Bellman-Ford example



Queue

~~t~~
~~$\perp$~~
~~b~~
d
$\perp$
a
c

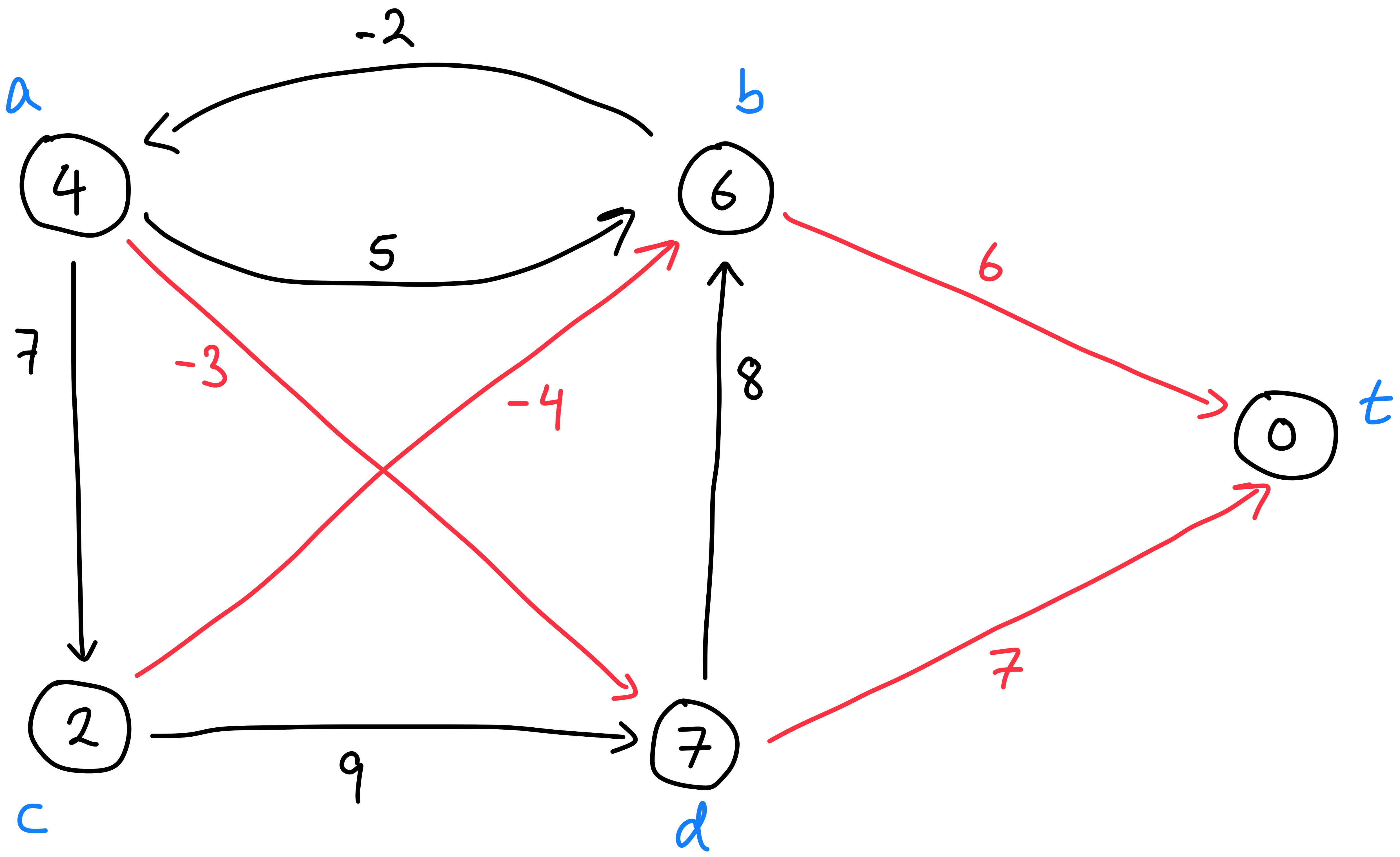
Bellman-Ford example



Queue

~~t~~
~~⊥~~
~~b~~
~~d~~
⊥
a
c

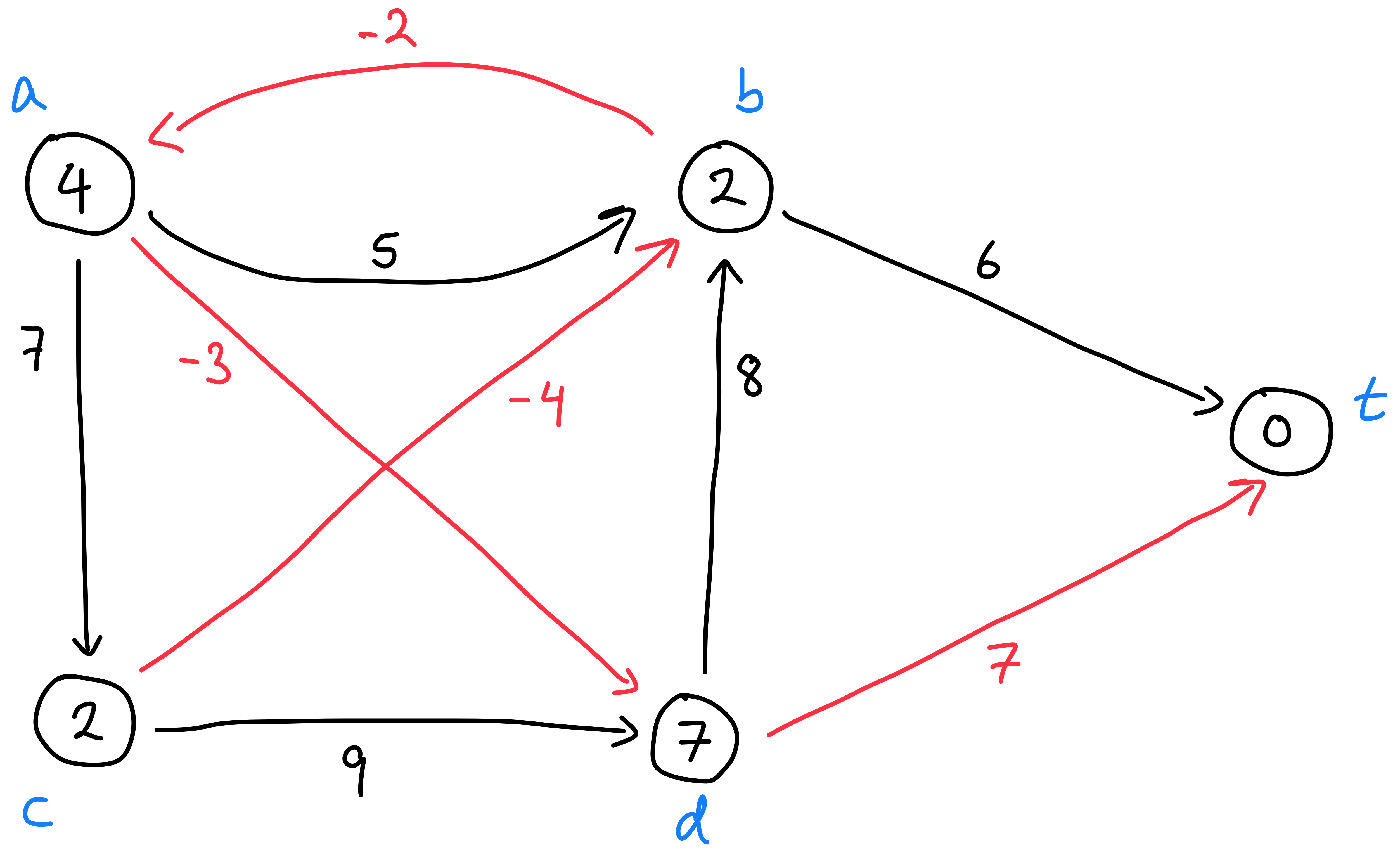
Bellman-Ford example



Queue

~~t~~
~~⊥~~
~~b~~
~~d~~
~~⊥~~
a
c
⊥

Bellman-Ford example



Queue

~~t~~

~~⊥~~

~~b~~

~~d~~

~~⊥~~

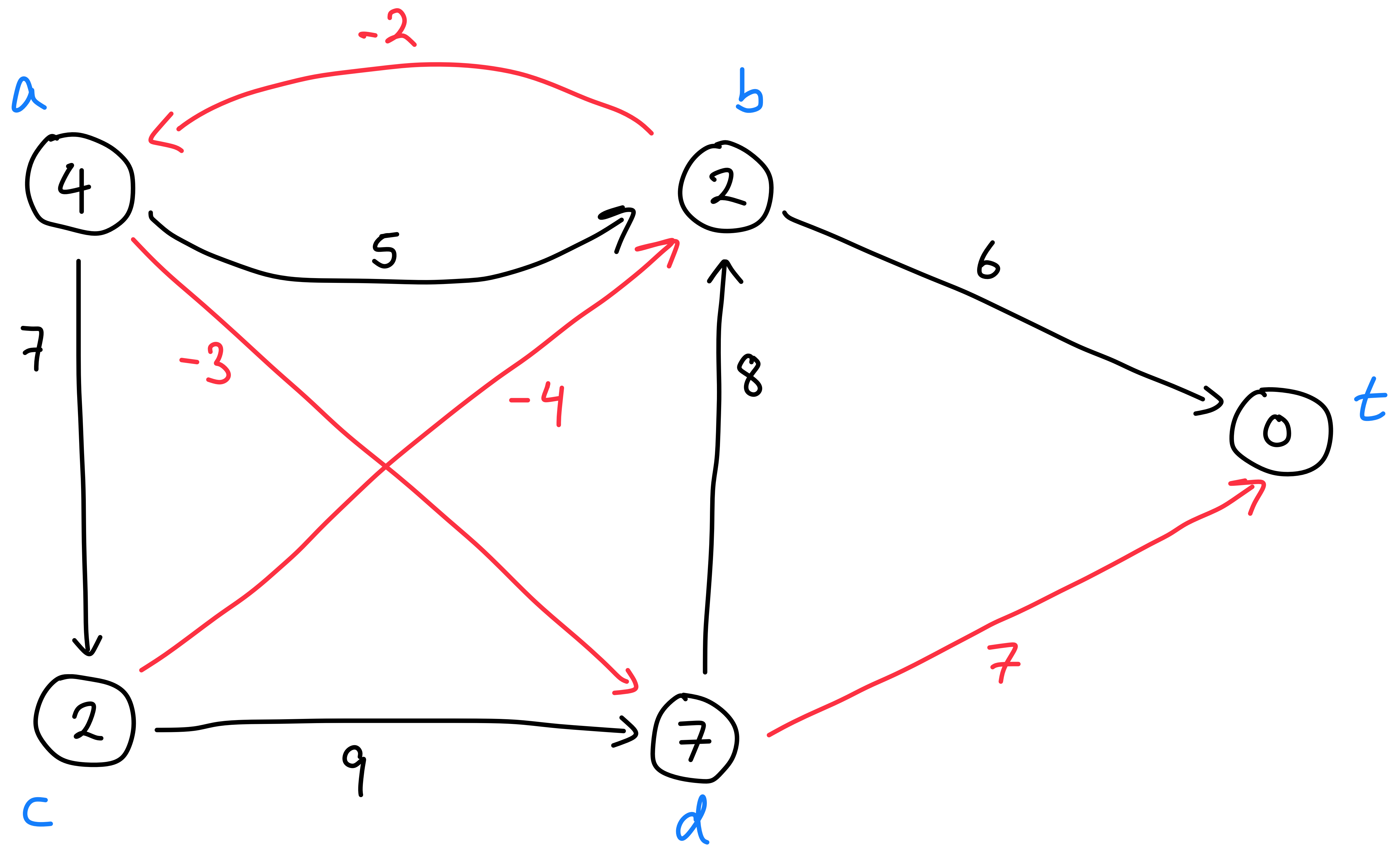
~~a~~

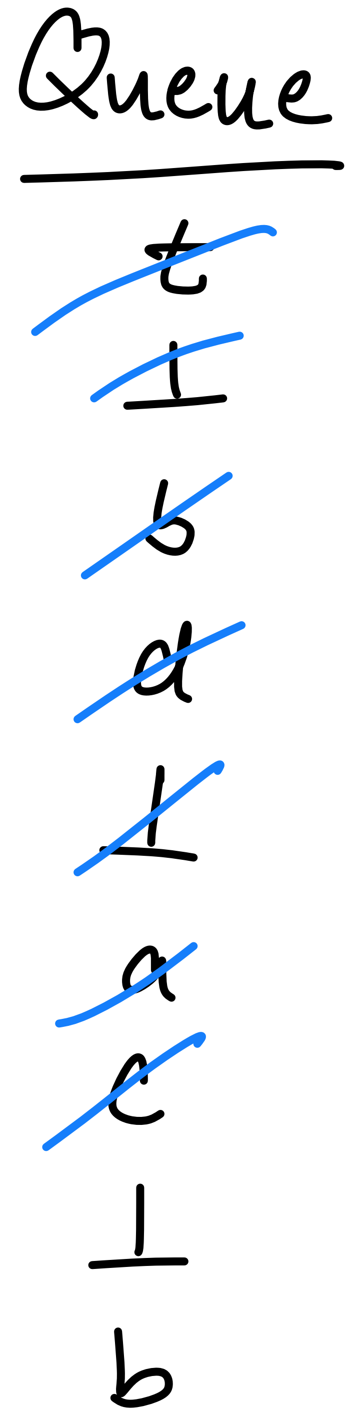
c

⊥

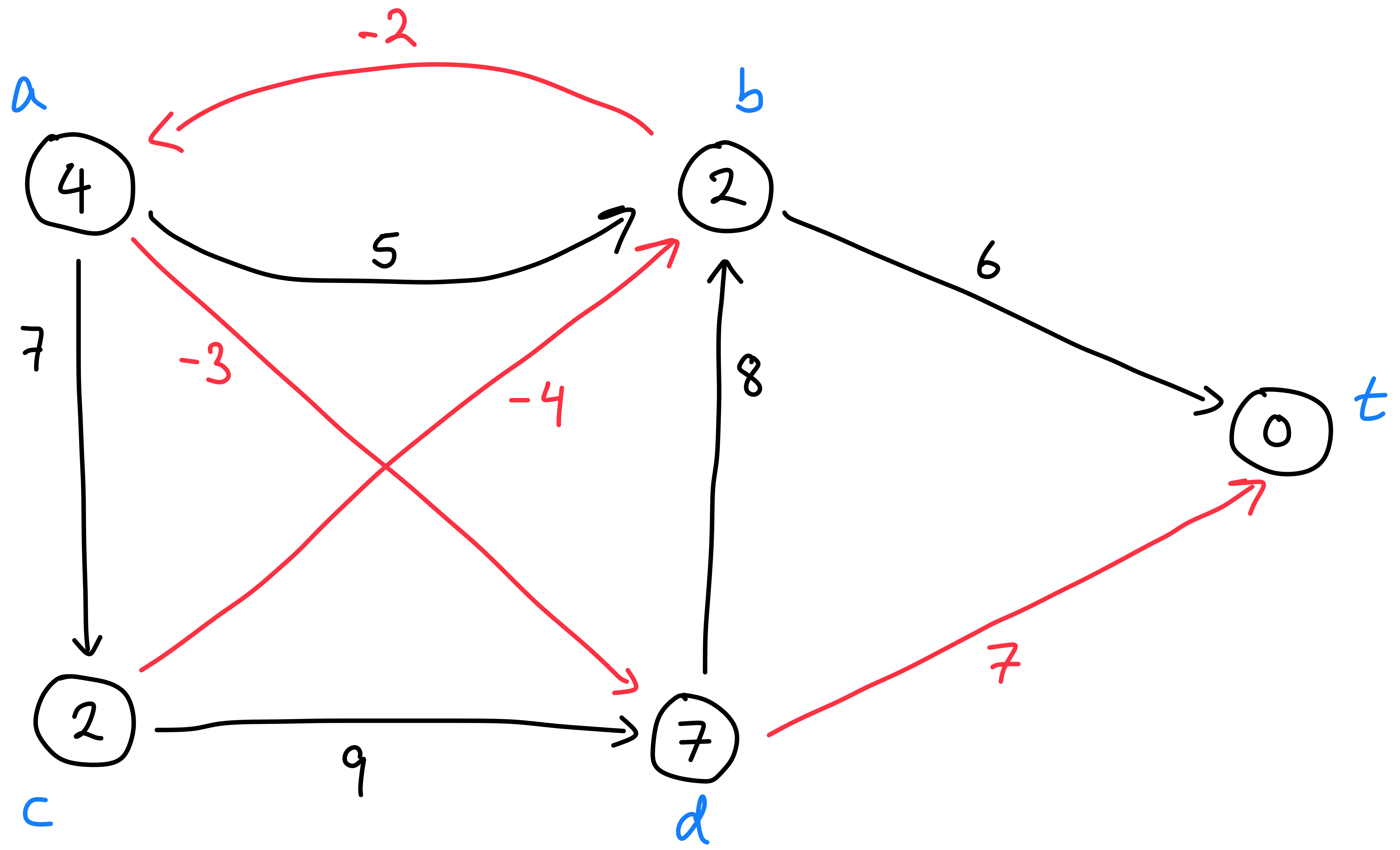
b

Bellman-Ford example





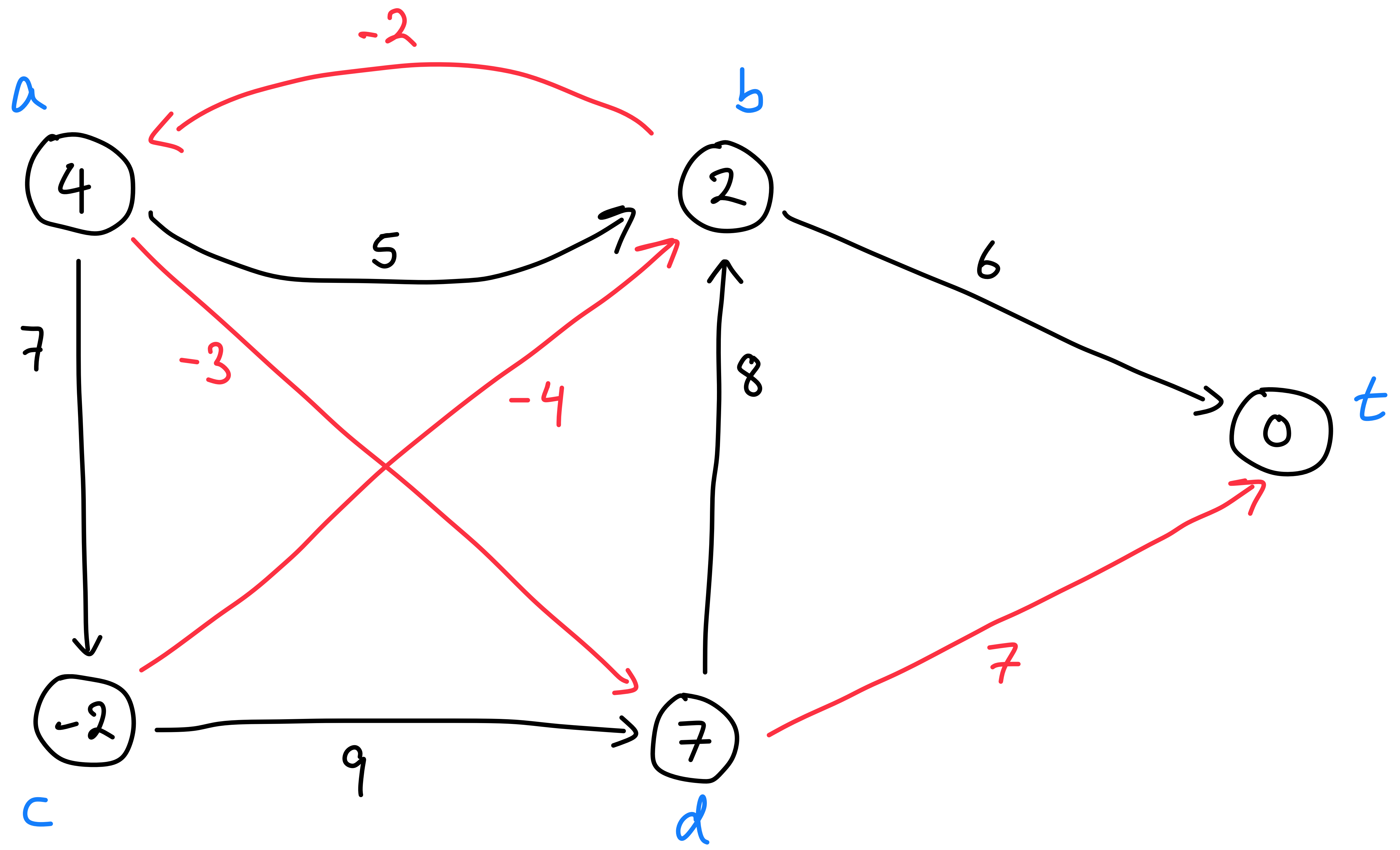
Bellman-Ford example



Queue

~~t~~
~~⊥~~
~~b~~
~~d~~
~~⊥~~
~~a~~
~~c~~
~~⊥~~
b
⊥

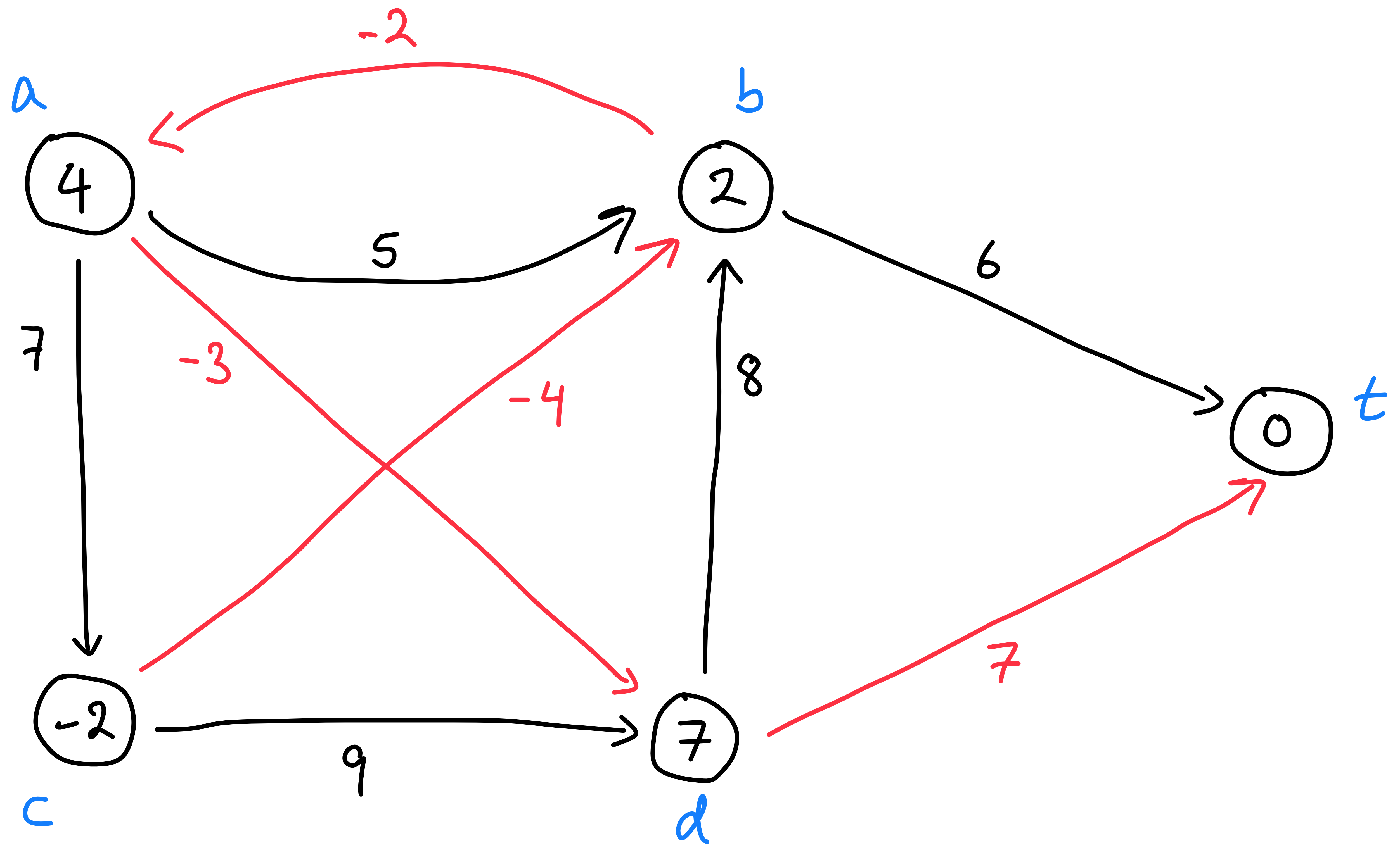
Bellman-Ford example



Queue

~~t~~
~~⊥~~
~~b~~
~~d~~
~~⊥~~
~~a~~
~~c~~
~~⊥~~
~~b~~
⊥
c

Bellman-Ford example



Queue

~~t~~
~~$\bot$~~ 1
~~b~~
~~d~~
~~$\bot$~~ 2
~~a~~
~~c~~
~~$\bot$~~ 3
~~b~~
~~$\bot$~~ 4
c
$\bot$

n-1 iterations.

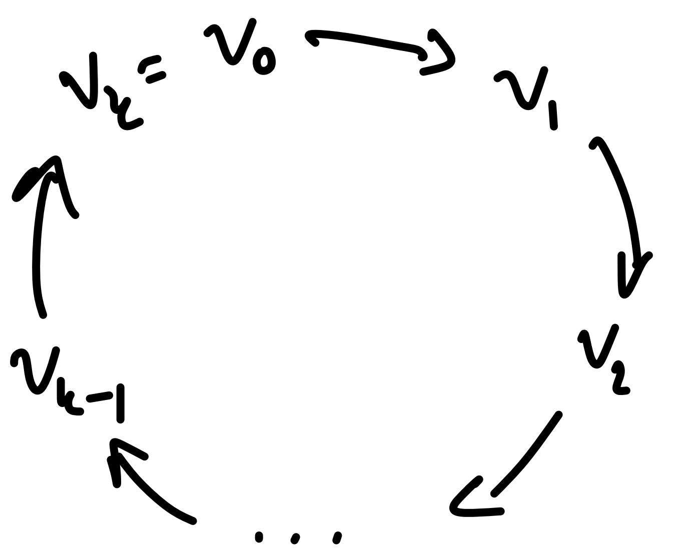
Alg concludes.

Detecting negative cycles

- **Lemma:** If every vertex s can reach t, and G has a negative cycle, then there is some edge $u \to v$ so that $d(n-1,u) > d(n-1,v) + w(u,v)$. If G has no negative cycles, then output of Bellman-Ford is correct on final iteration.

- **Proof:** By contradiction.

Let G have a negative cycle.



$$\sum_{i=0}^{k-1} w(v_i, v_{i+1}) < 0. \quad (1)$$

Assume (for $\perp$) that $\forall$ edges $u \to v$, $d(n-1,u) \leq d(n-1,v) + w(u,v)$.

Adding up these equations for the cycle,

$$\underbrace{\sum_{i=0}^{k-1} d(n-1, v_i)}_{\text{same term}} \leq \underbrace{\sum_{i=0}^{k-1} d(n-1, v_{i+1})}_{\text{same term}} + \sum_{i=0}^{k-1} w(v_i, v_{i+1})$$

$$\Rightarrow \quad 0 \leq \sum_{i=0}^{k-1} w(v_i, v_{i+1}) \quad (2)$$

(1) and (2) are inconsistent, proving the contradiction.

Detecting negative cycles

- **Lemma:** If every vertex s can reach t, and G has a negative cycle, then there is some edge $u \to v$ so that $d(n-1,u) > d(n-1,v) + w(u,v)$. If G has no negative cycles, then output of Bellman-Ford is correct on final iteration.

- **Proof:** The previous slide proves the first part of the statement.

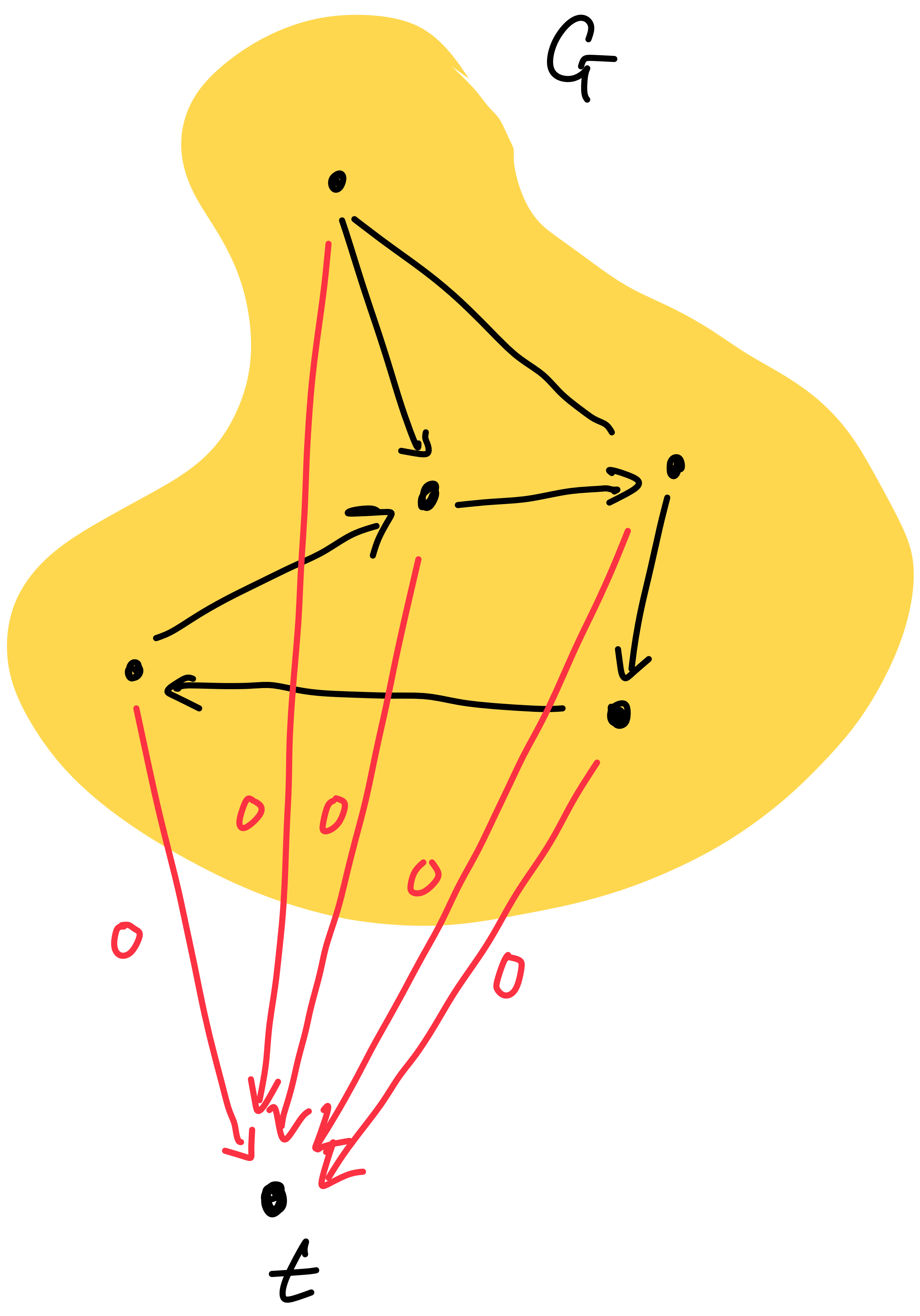
    - If there are no negative cycles, the shortest path $s \rightsquigarrow t$ consists of unique vertices and has length $\leq n-1$.

    - We previously proved that $d(i,s)$ was optimal length of path $s \rightsquigarrow t$ of length $\leq i$.

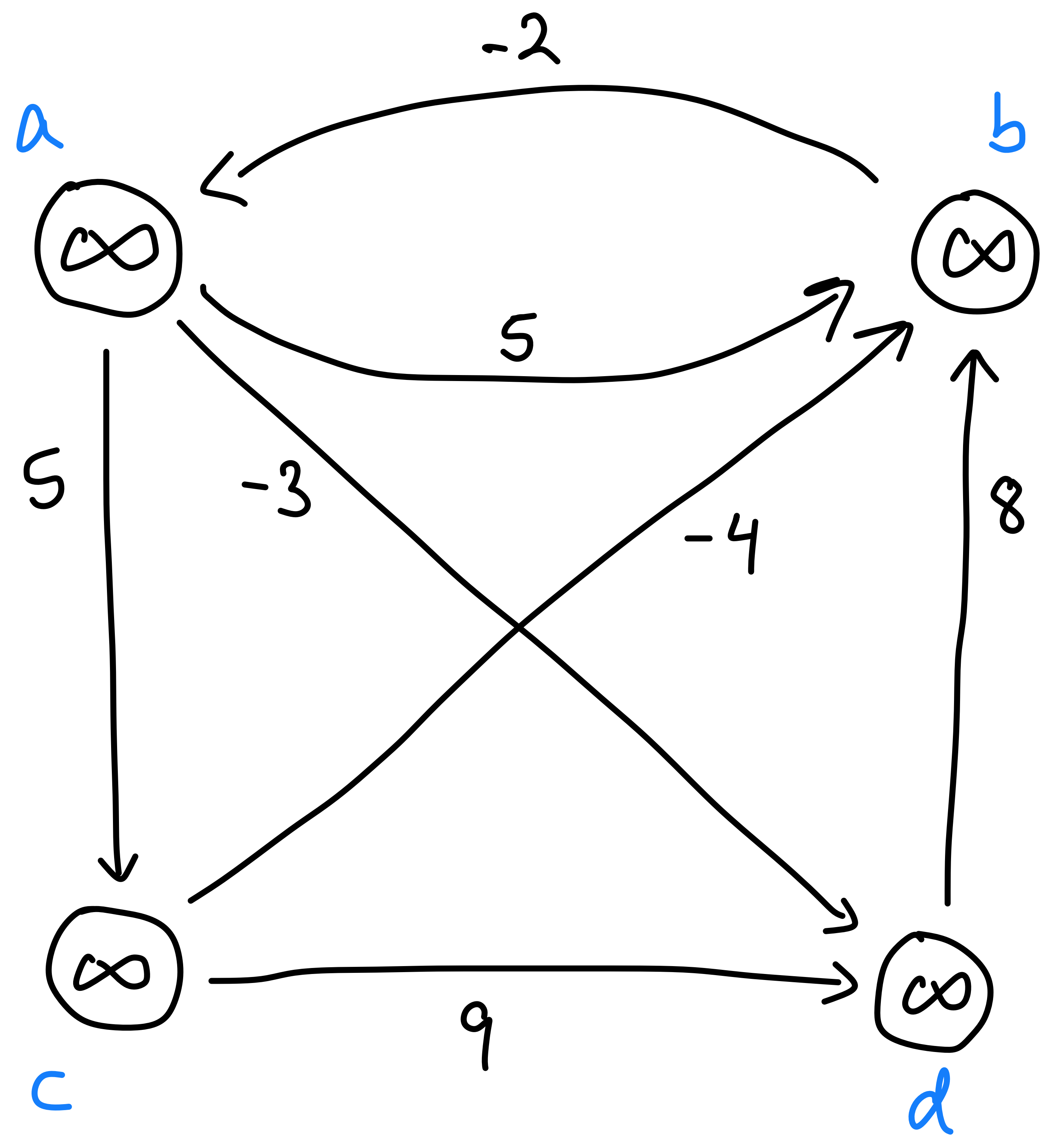
    - Together, concludes proof.

Negative cycle detection

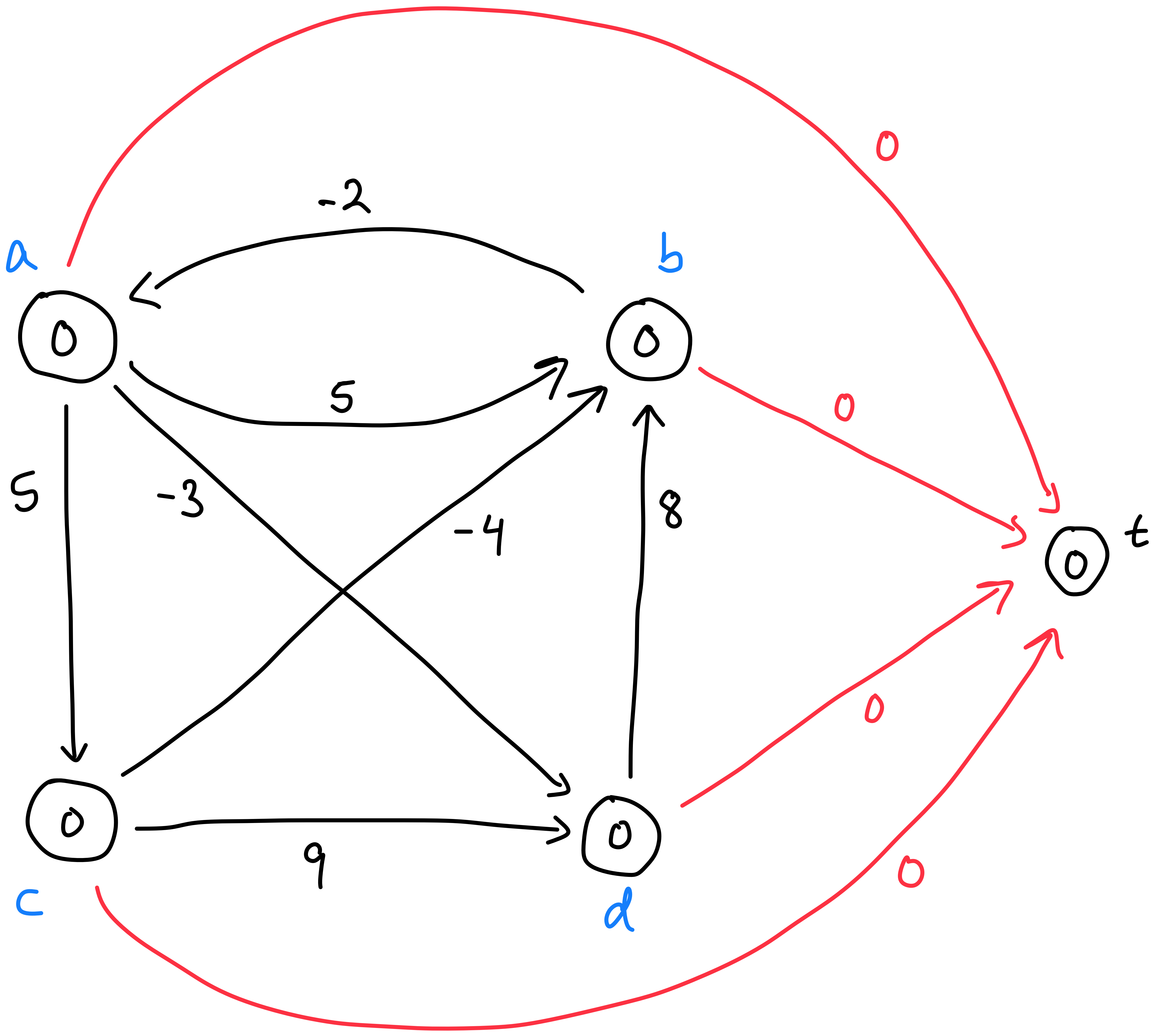
- **Negative cycle detection algorithm:**
  - Run Bellman-Ford assuming there are no negative cycles
  - For each edge $u \to v$, verify that $d(u) \leq d(v) + w(u, v)$. Else, report “negative cycle detected”.
- This will only detective negative cycles amongst vertices that have paths to t. Will not detect negative cycles in the entire graph for a poorly connected choice of t.
- Solution: Add a new “sink” t to the graph and add edge $v \to t$ of weight 0 for all vertices. Run detection algorithm w.r.t this sink.



Bellman-Ford with negative cycles example

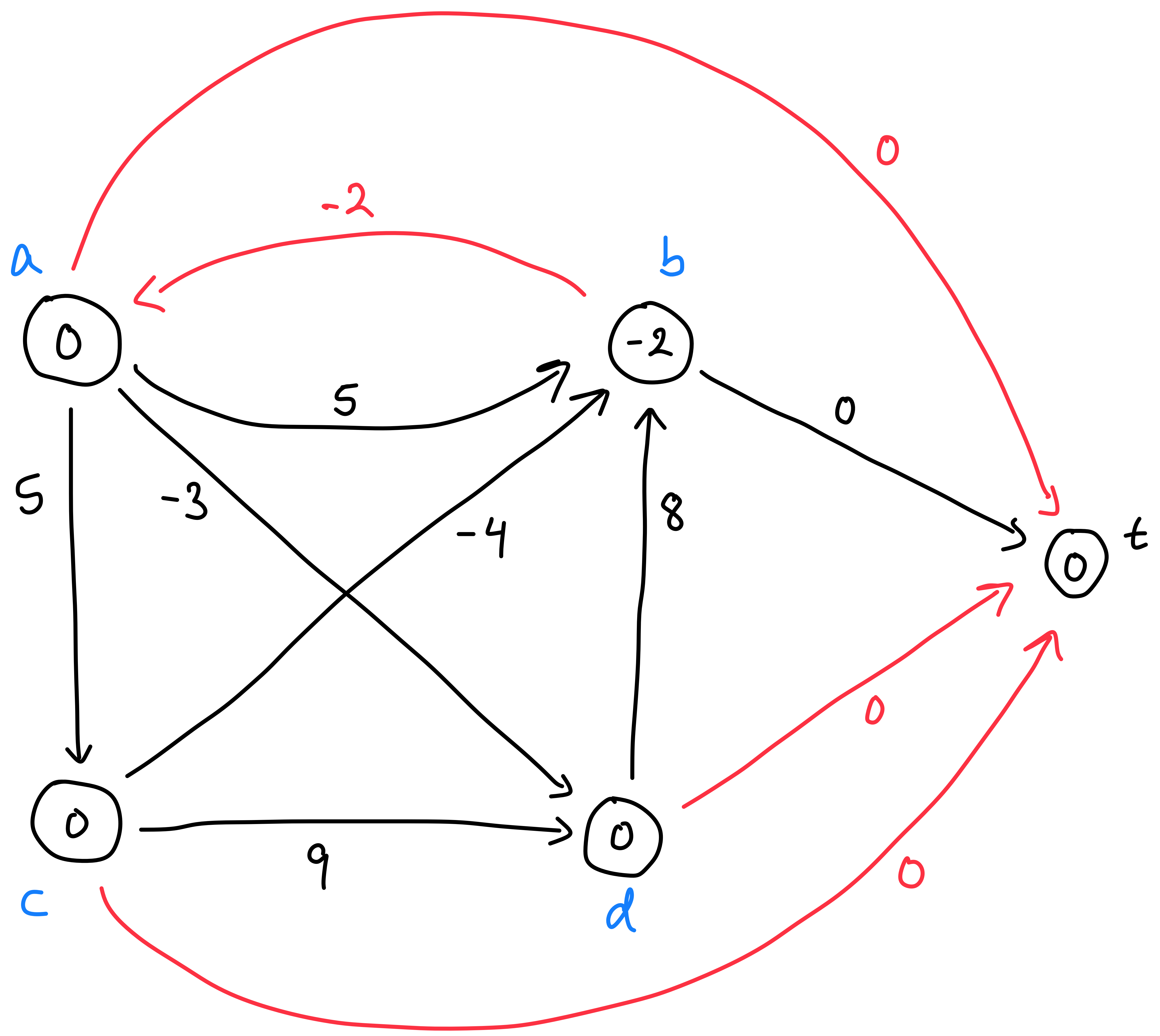


Bellman-Ford with negative cycles example



Queue
~~t~~
~~⊥~~
a
b
c
d
⊥

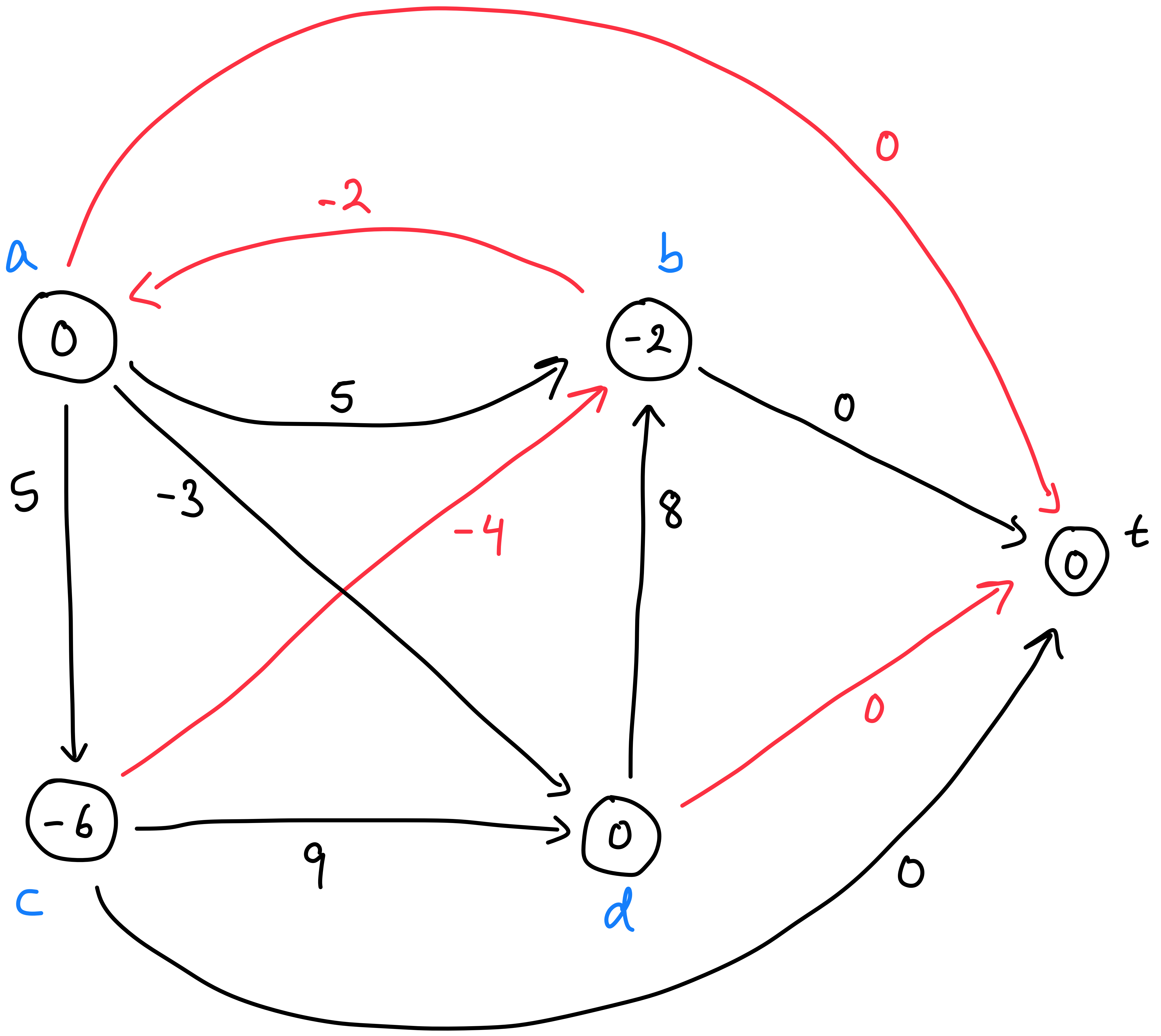
Bellman-Ford with negative cycles example



Queue

~~t~~
~~⊥~~
~~a~~
b
c
d
⊥
b

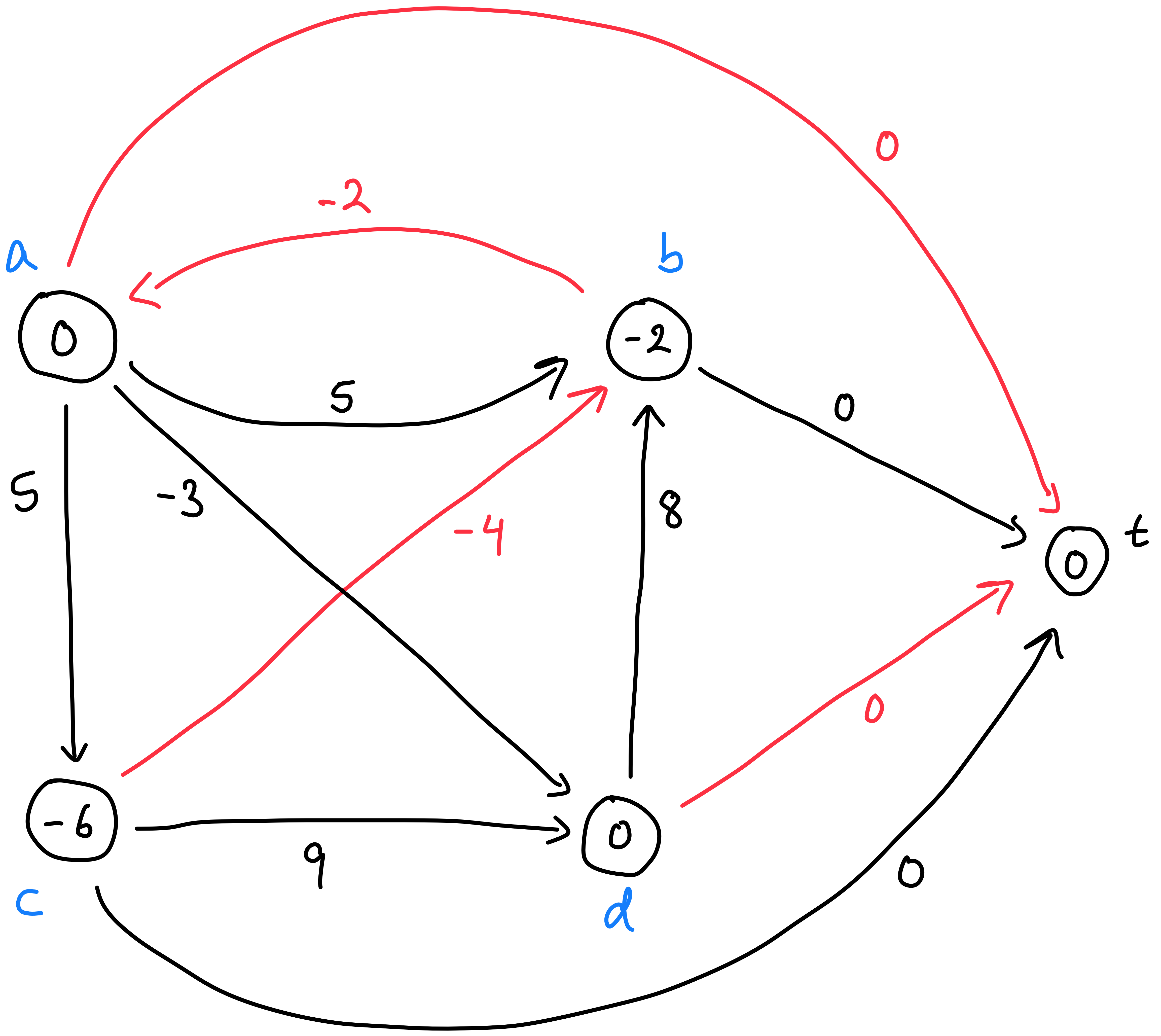
Bellman-Ford with negative cycles example



Queue

~~t~~
~~⊥~~
~~a~~
~~b~~
c
d
⊥
b
c

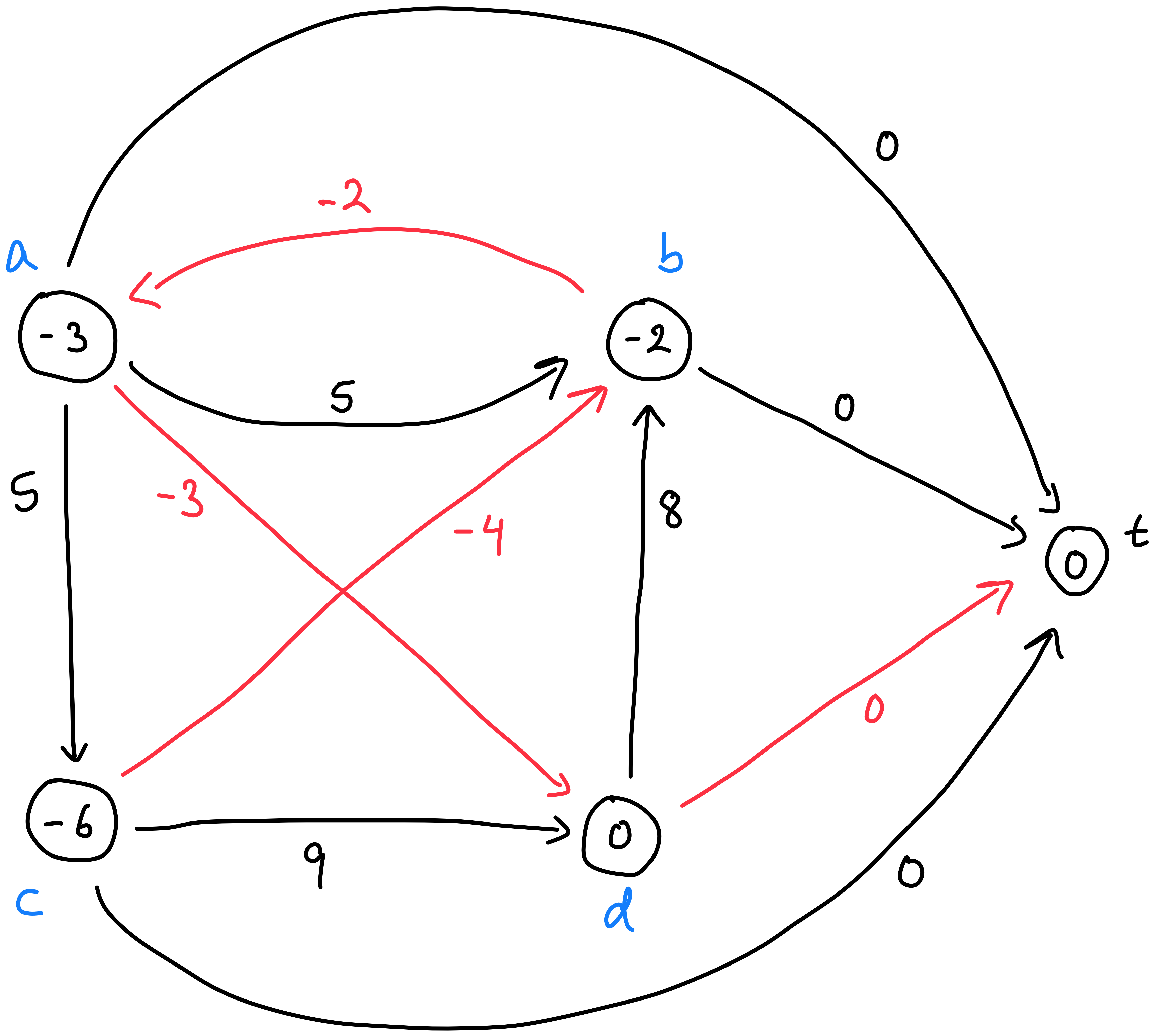
Bellman-Ford with negative cycles example



Queue

~~t~~
~~⊥~~
~~a~~
~~b~~
~~c~~
d
⊥
b
c

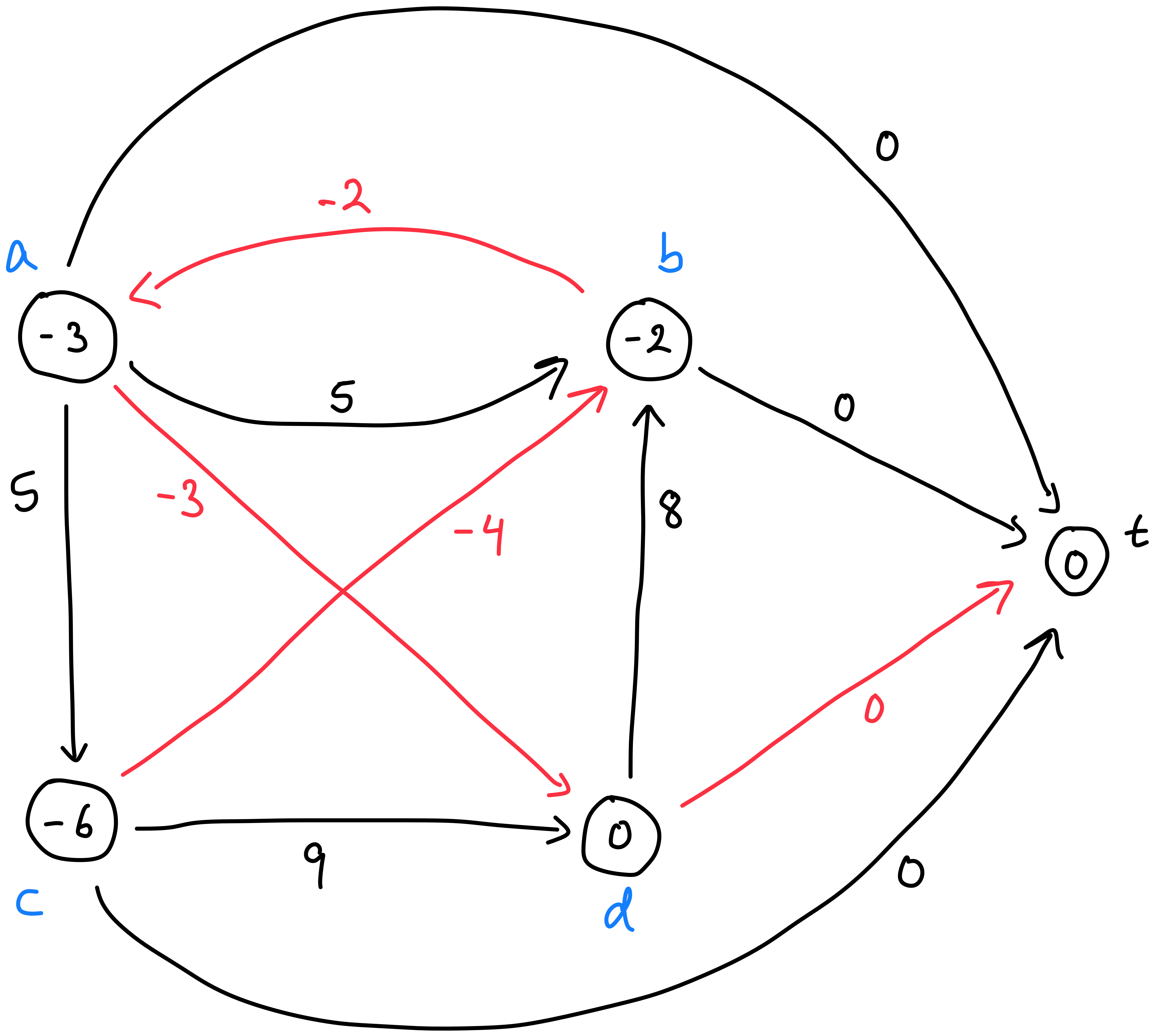
Bellman-Ford with negative cycles example



Queue

~~t~~
~~⊥~~
~~a~~
~~b~~
~~c~~
~~d~~
⊥
b
c
a

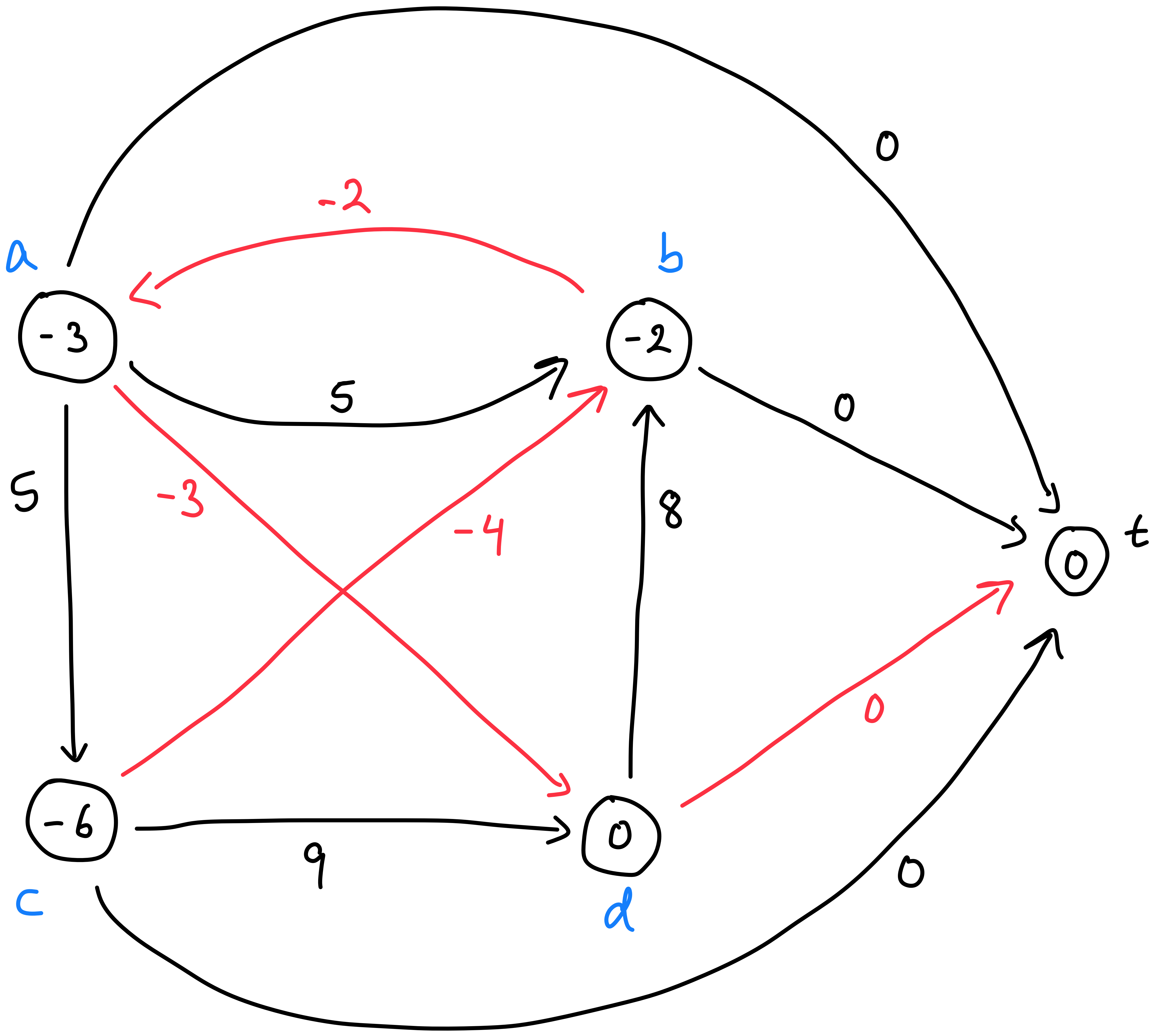
Bellman-Ford with negative cycles example



Queue

~~t~~
~~⊥~~
~~a~~
~~b~~
~~c~~
~~d~~
~~⊥~~
b
c
a
⊥

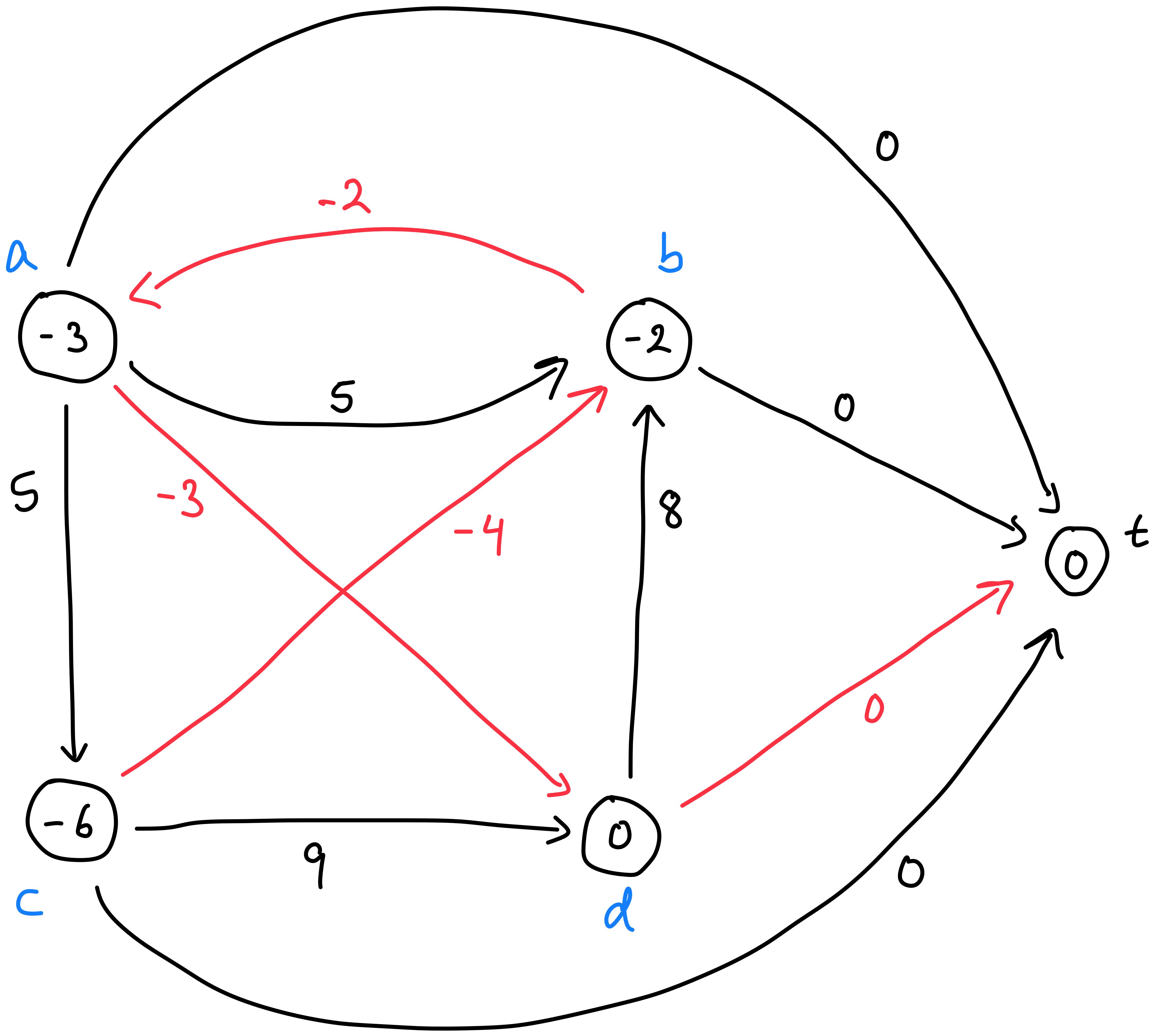
Bellman-Ford with negative cycles example



Queue

~~t~~
~~⊥~~
~~a~~
~~b~~
~~c~~
~~d~~
~~⊥~~
~~b~~
c
a
⊥

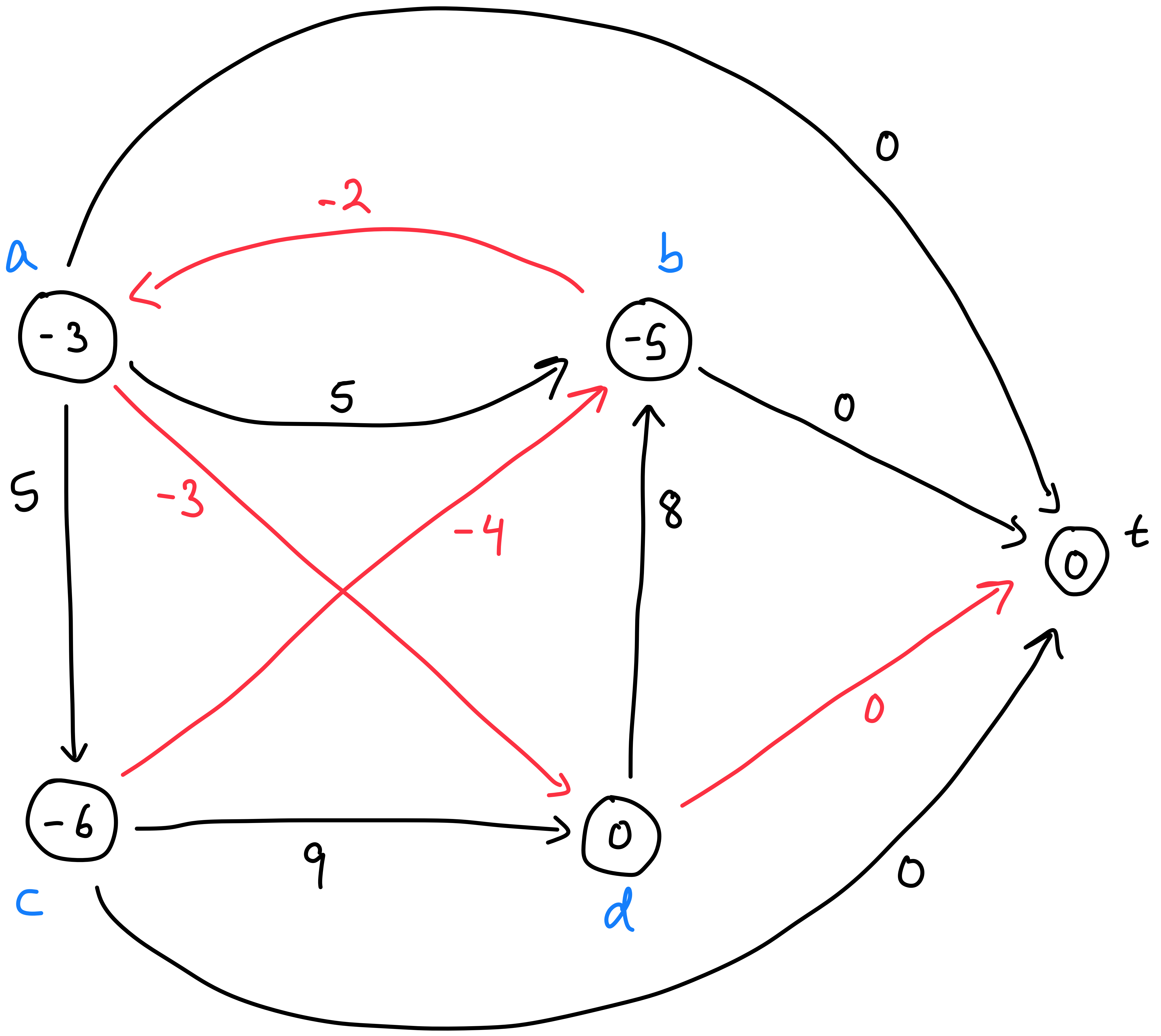
Bellman-Ford with negative cycles example



Queue

~~t~~
~~⊥~~
~~a~~
~~b~~
~~c~~
~~d~~
~~⊥~~
~~b~~
~~c~~
a
⊥

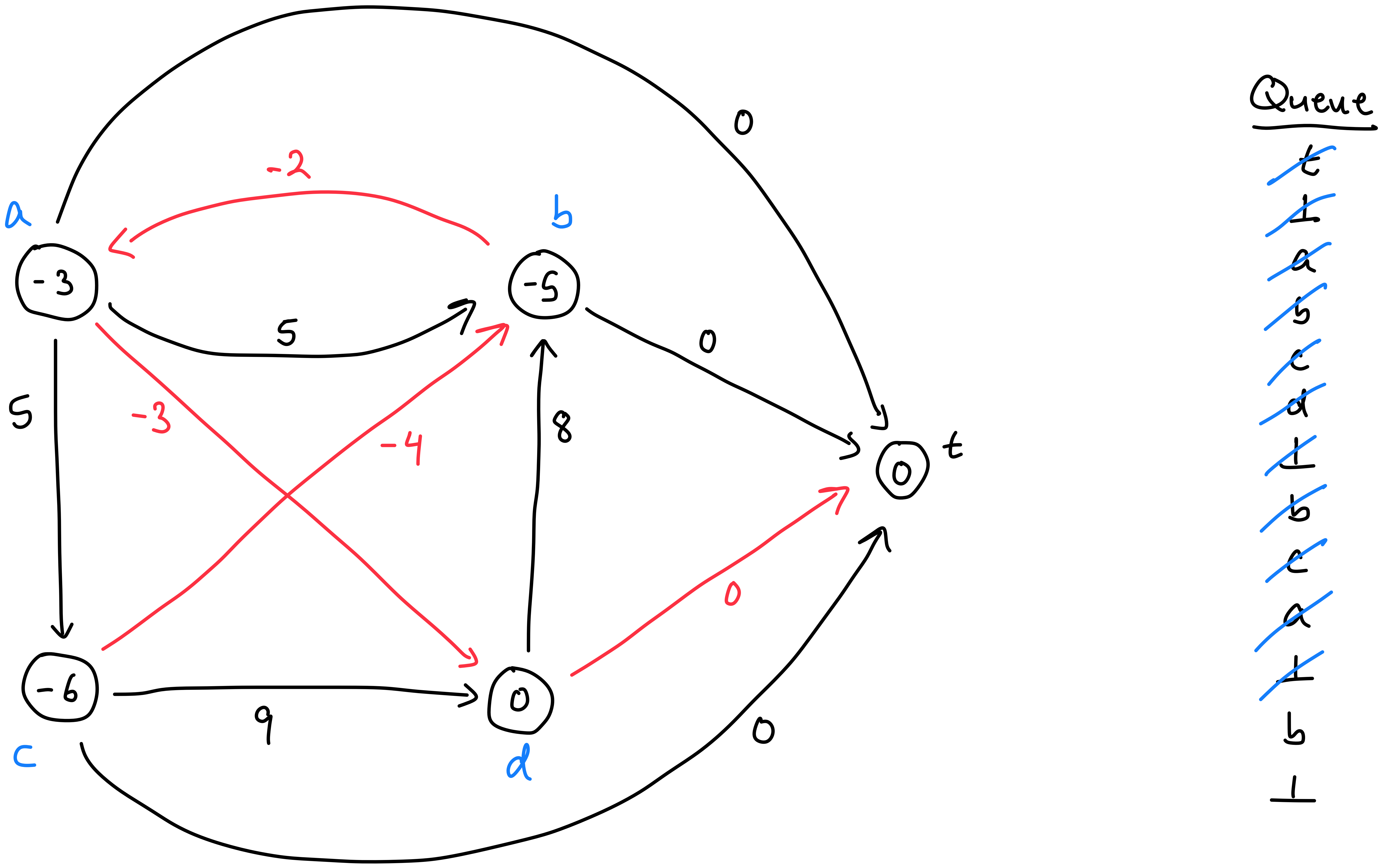
Bellman-Ford with negative cycles example



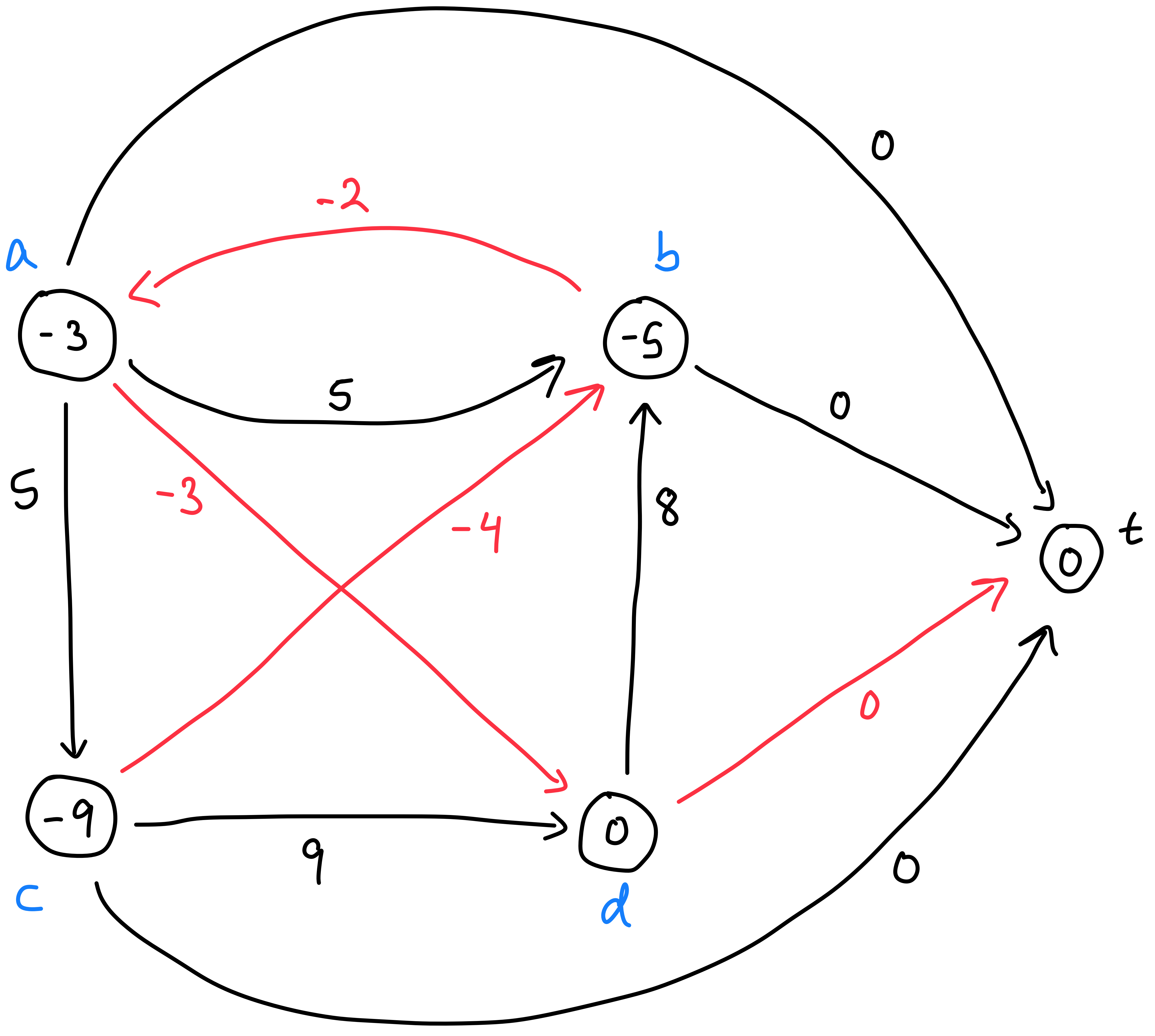
Queue

~~t~~
~~⊥~~
~~a~~
~~b~~
~~c~~
~~d~~
~~⊥~~
~~b~~
~~c~~
~~a~~
⊥
b

Bellman-Ford with negative cycles example



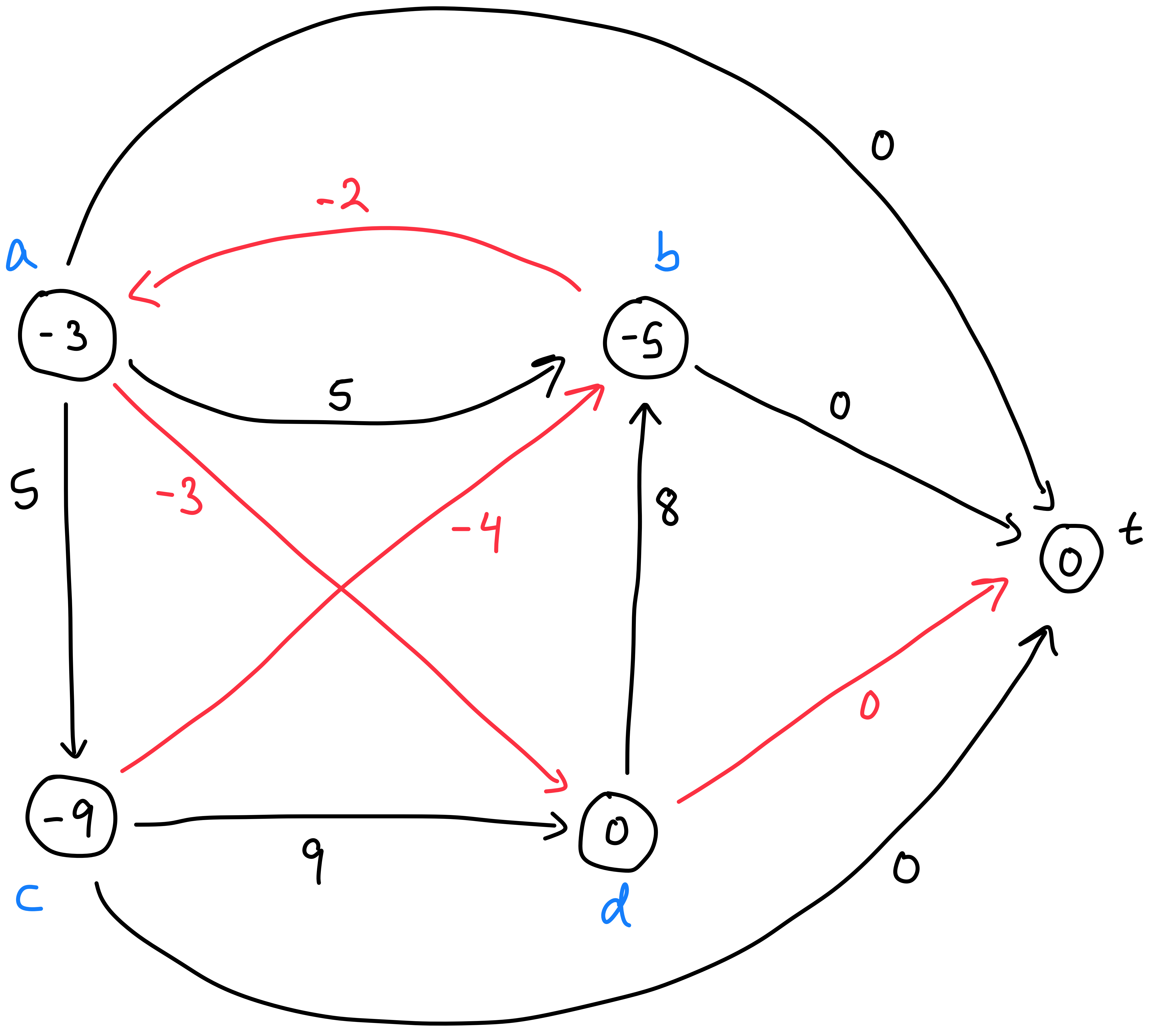
Bellman-Ford with negative cycles example



Queue

- ~~t~~
- ~~⊥~~
- ~~a~~
- ~~b~~
- ~~c~~
- ~~d~~
- ~~⊥~~
- ~~b~~
- ~~c~~
- ~~a~~
- ~~⊥~~
- ~~b~~
- ⊥
- c

Bellman-Ford with negative cycles example



Queue

~~t~~
~~⊥~~ ①
~~a~~
~~b~~
~~c~~
~~d~~
~~⊥~~ ②
~~b~~
~~c~~
~~a~~
~~⊥~~ ③
~~b~~
~~⊥~~ ④
c

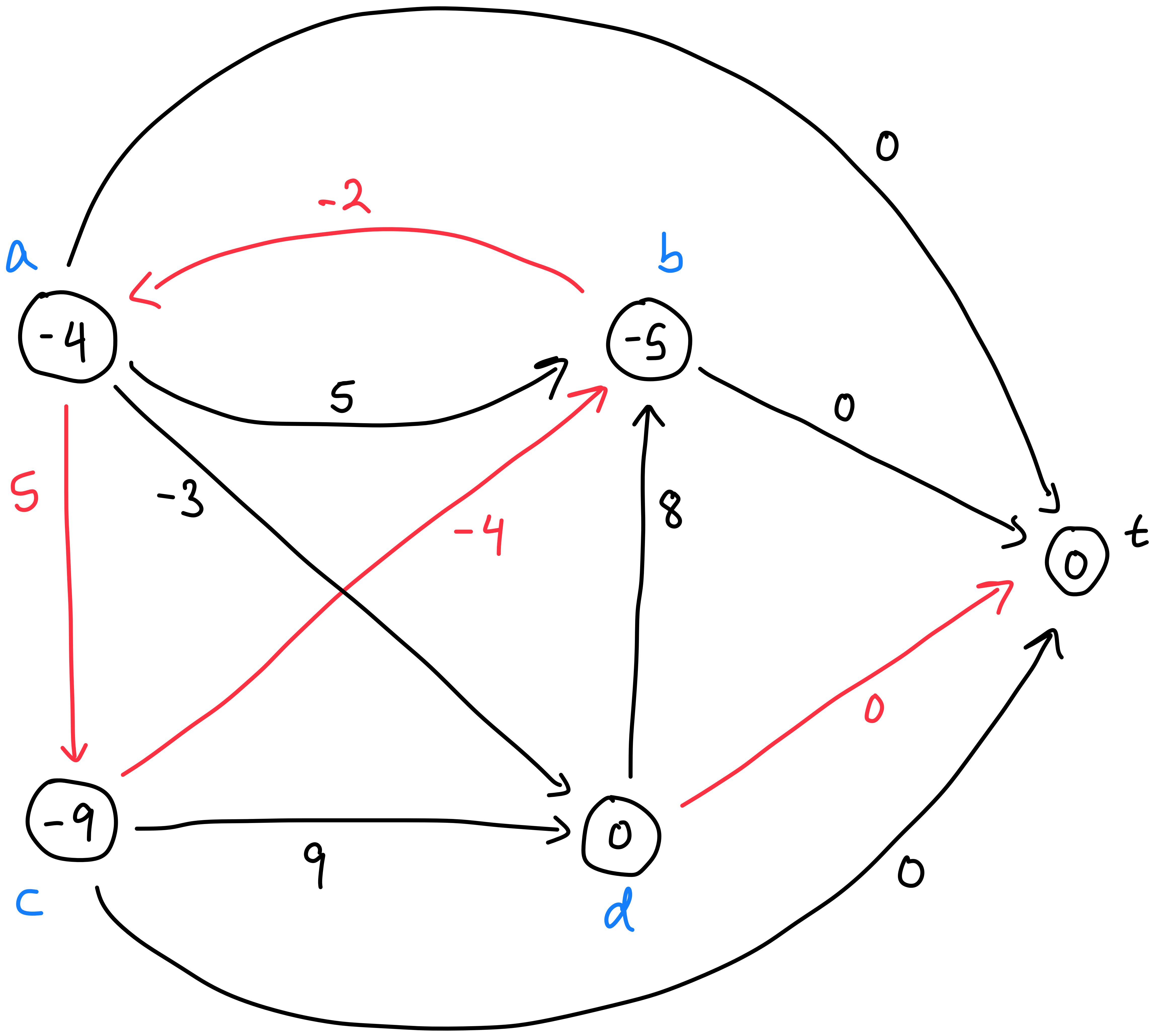
4 iterations completed.

Now checking edges, we notice that

$$\underbrace{d(a)}_{-3} > \underbrace{d(c)}_{-9} + \underbrace{w(a,c)}_{5}$$

So a negative cycle exists $(a \to c \to b)$.

Bellman-Ford with negative cycles example



Queue

~~t~~
~~⊥~~ 1
~~a~~
~~b~~
~~c~~
~~d~~
~~⊥~~ 2
~~b~~
~~c~~
~~a~~
~~⊥~~ 3
~~b~~
~~⊥~~ 4
~~c~~
⊥
a

Observe what would happen if we updated once more.

Shortest paths with negative weights on a DAG

- No cycles by definition
- Under topological sort, edges only go from low to high numbered vertices
- One pass through the vertices in reverse topological order suffices
- **Runtime**: $O(n + m)$