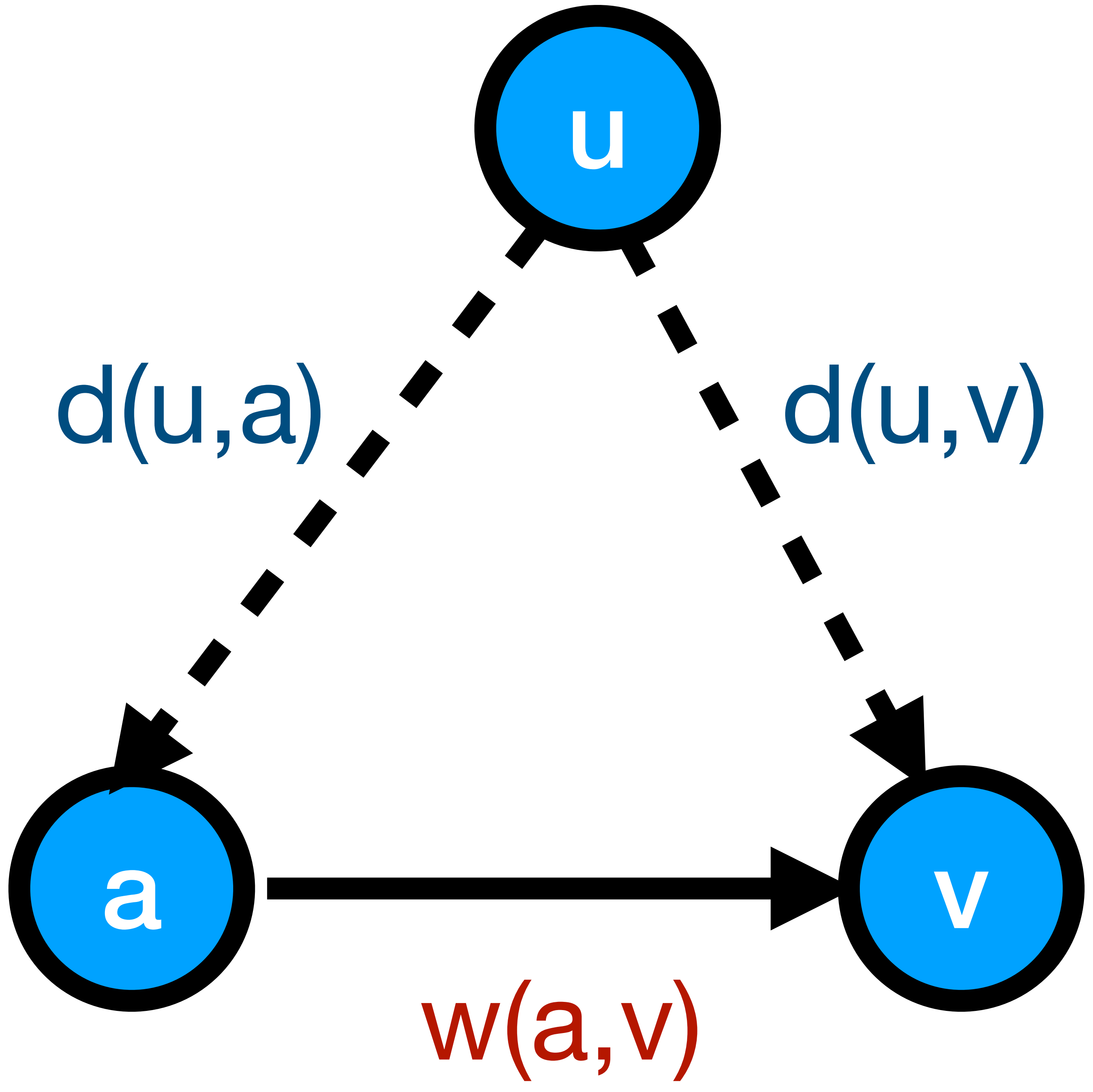
The Bellman-Ford Algorithm

Anand Natarajan, 4/8/2021

Reminder: relaxation

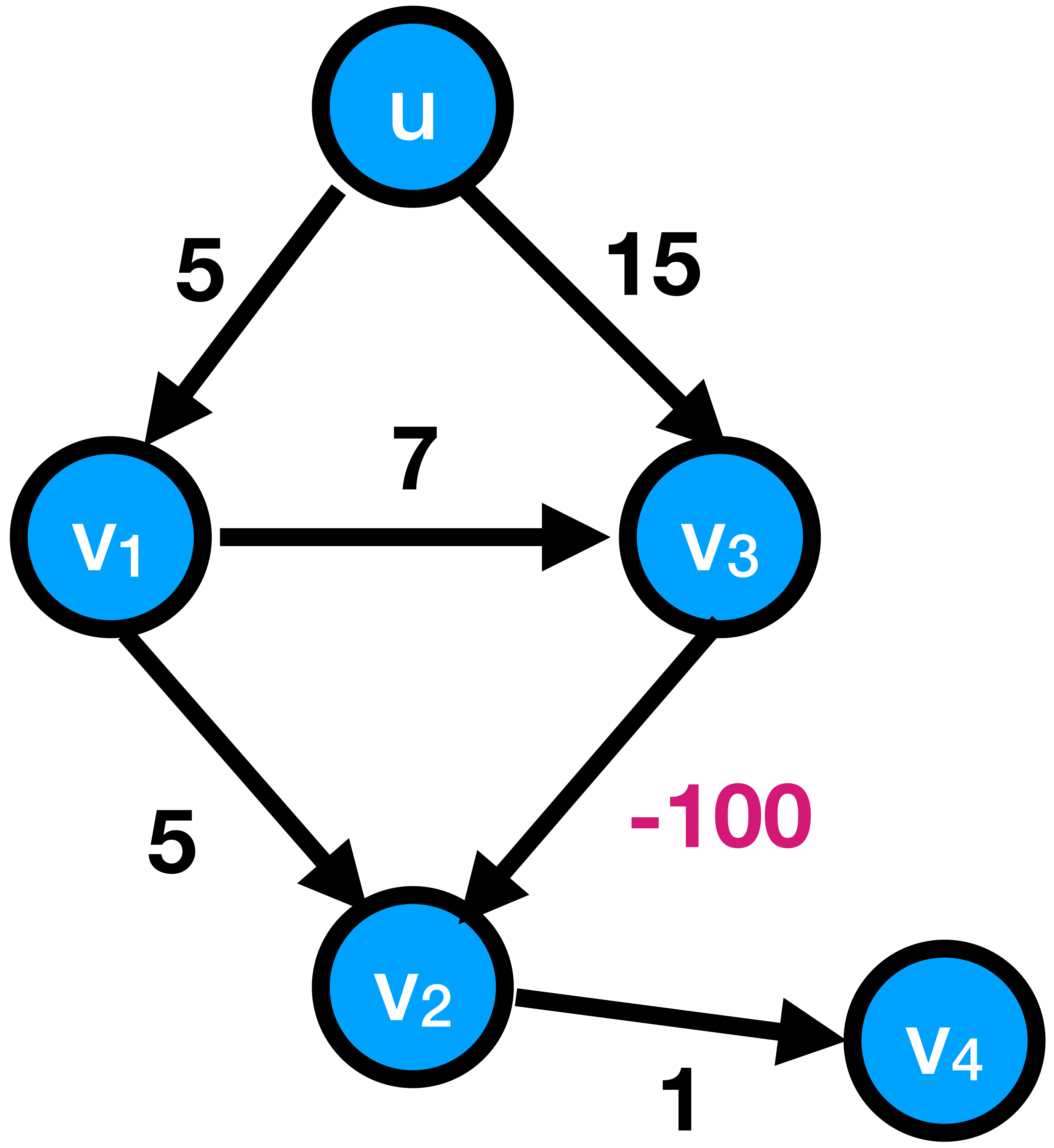
- **Relaxation** fixes violations of the triangle inequality:

- ```
  def try_to_relax(a,v):
      if d[v] > d[a] + w(a,v):
          d[v] = d[a] + w(a,v)
          parent[v] = a
  ```


  ```

What about negative weights?

- Ex: weights measure energy gain or loss
- **Dijkstra fails:**
  $d[v_4] = 11$
  $\delta[v_4] = -87$
- **Relaxation order:**
  (u, v1), (u,v3)
  (v1, v2), (v1,v3)
  (v2,v4)
  (v3,v2)



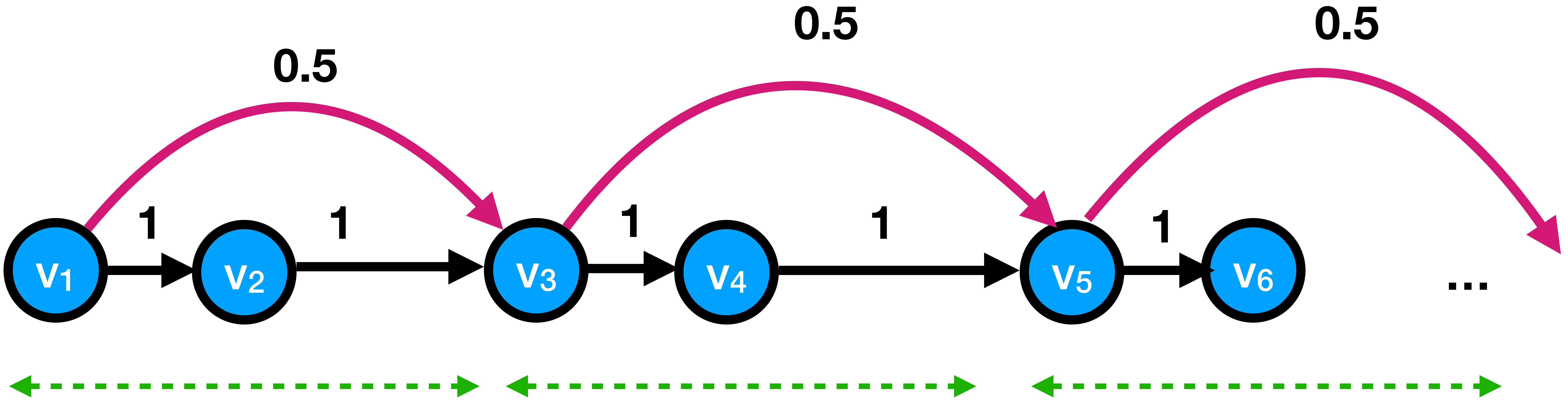
Back to the drawing board…

```
def sssp(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0
    while we're not done:
        (v1, v2) = pick an edge somehow
        try_to_relax(v1, v2)
    return d, parent
```

- Dijkstra chooses edges **greedily** and relaxes each edge **exactly once**
- What if we try **repeatedly relaxing edges?** Will we converge?

Bad relaxation orders

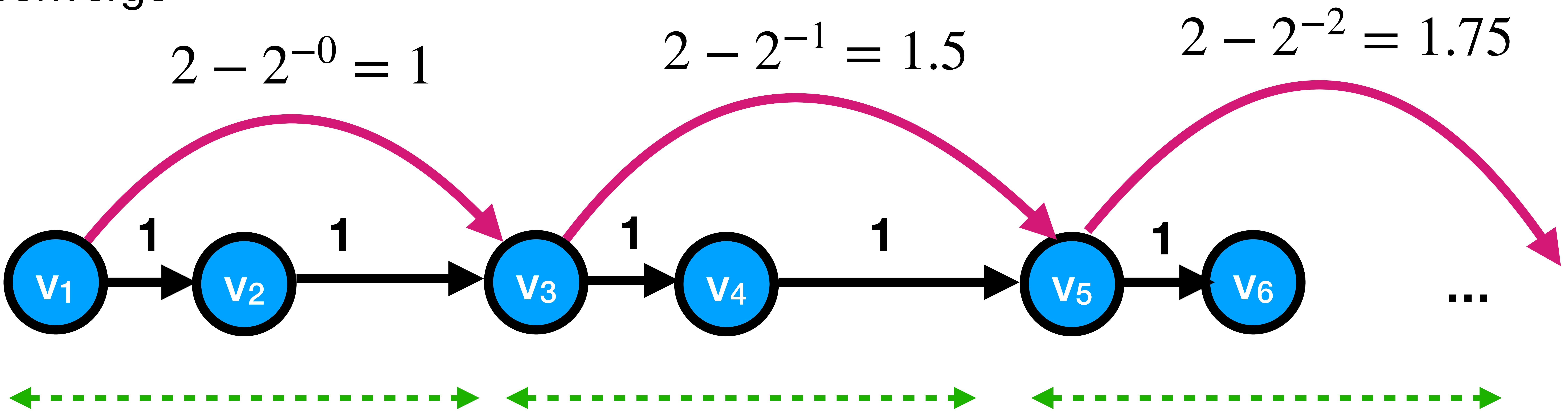
There is an order that takes **Θ(n²)** steps to converge



Relax **black** edges, then recursively relax right subgraph, then relax **purple** edge, then relax right subgraph again

Bad relaxation orders

There is an order that takes **$\Theta(2^{n/2})$** steps to converge



"Doubling gadget"

Relax **black** edges, then recursively relax right subgraph, then relax **purple** edge, then relax right subgraph again

Path relaxation lemma

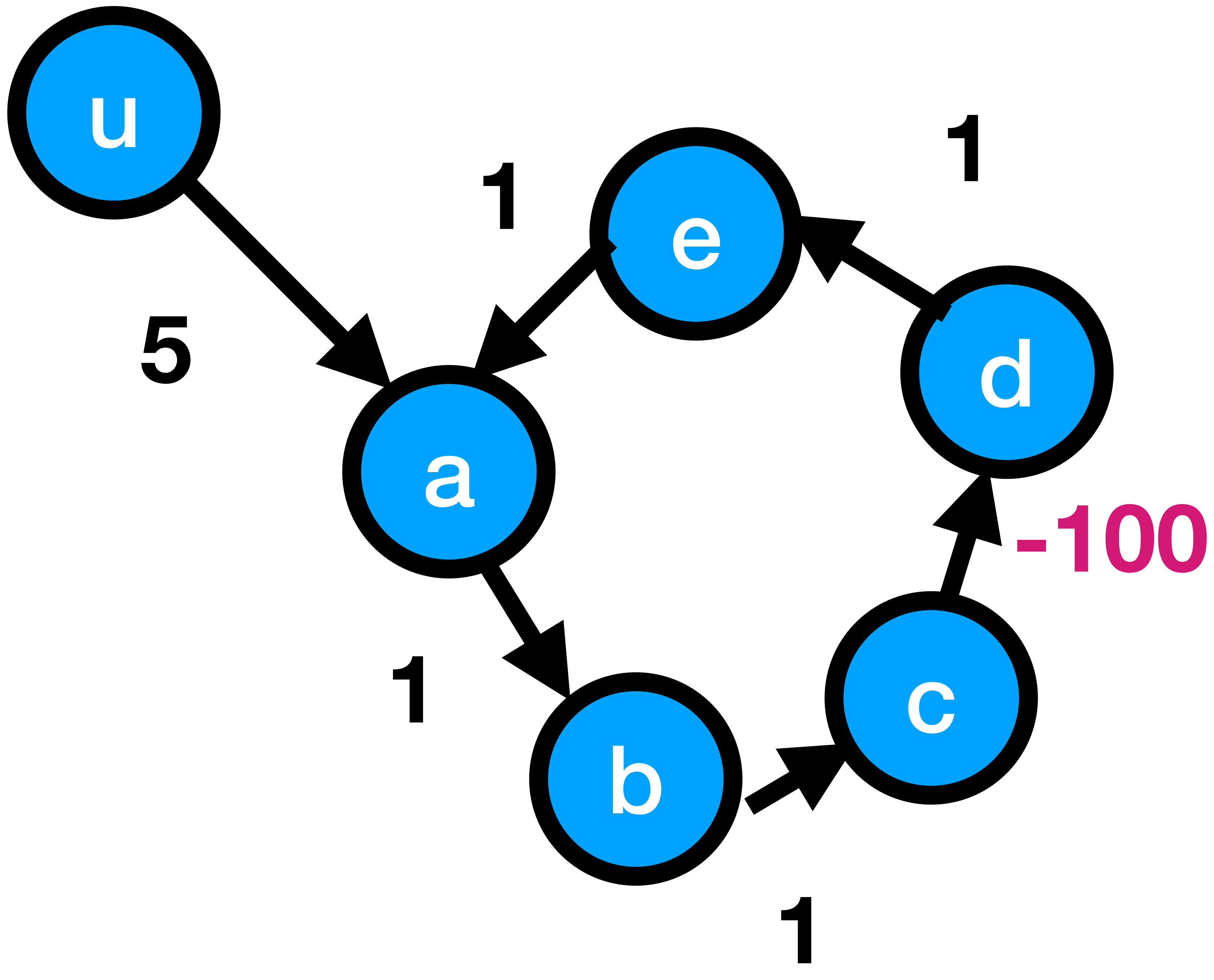
- **Lemma:** if (u, u_1) (u_1, u_2), …, (u_{k-1}, u_k) is a s.p., and we relax these edges **in this order** (possibly with other relaxations in between), then $\boldsymbol{\delta[u_i] = d[u_i]}$ for all i along the path, **no matter** what other relaxations we do
- **Proof:** By induction on path length!
  - Assume it's true up to k-1: $\delta[u_{k-1}] = d[u_{k-1}]$
  - **After relaxation of (u_{k-1}, u_k),**
    $$\begin{aligned} d[u_k] &\leq d[u_{k-1}] + w(u_{k-1}, u_k) && \textit{By def. of relaxation} \\ &= \delta[u_{k-1}] + w(u_{k-1}, u_k) && \textit{By inductive hypothesis} \\ &= \delta[u_k] && \textit{By optimal subproblems} \end{aligned}$$

Some observations

- Would suffice if for every s.p., we relax the edges on the path **in order**—but we **don't know** order in advance!
- On a graph with V vertices, a s.p. has **at most V - 1** edges
- **Proposal (Bellman-Ford): number the edges from 1 to E. Relax them in this order V-1 times!**

Negative-weight cycles

- If there is a **negative weight cycle** reachable from the source, then there are **no shortest paths** to vertices on the cycle!
  - For any path, I can find a "shorter" one by going around the cycle more times!
- **New goal:** Given graph G, source u, output whether there exists a negative-weight cycle. **If not**, compute shortest paths from u to all other vertices.



Bellman-Ford

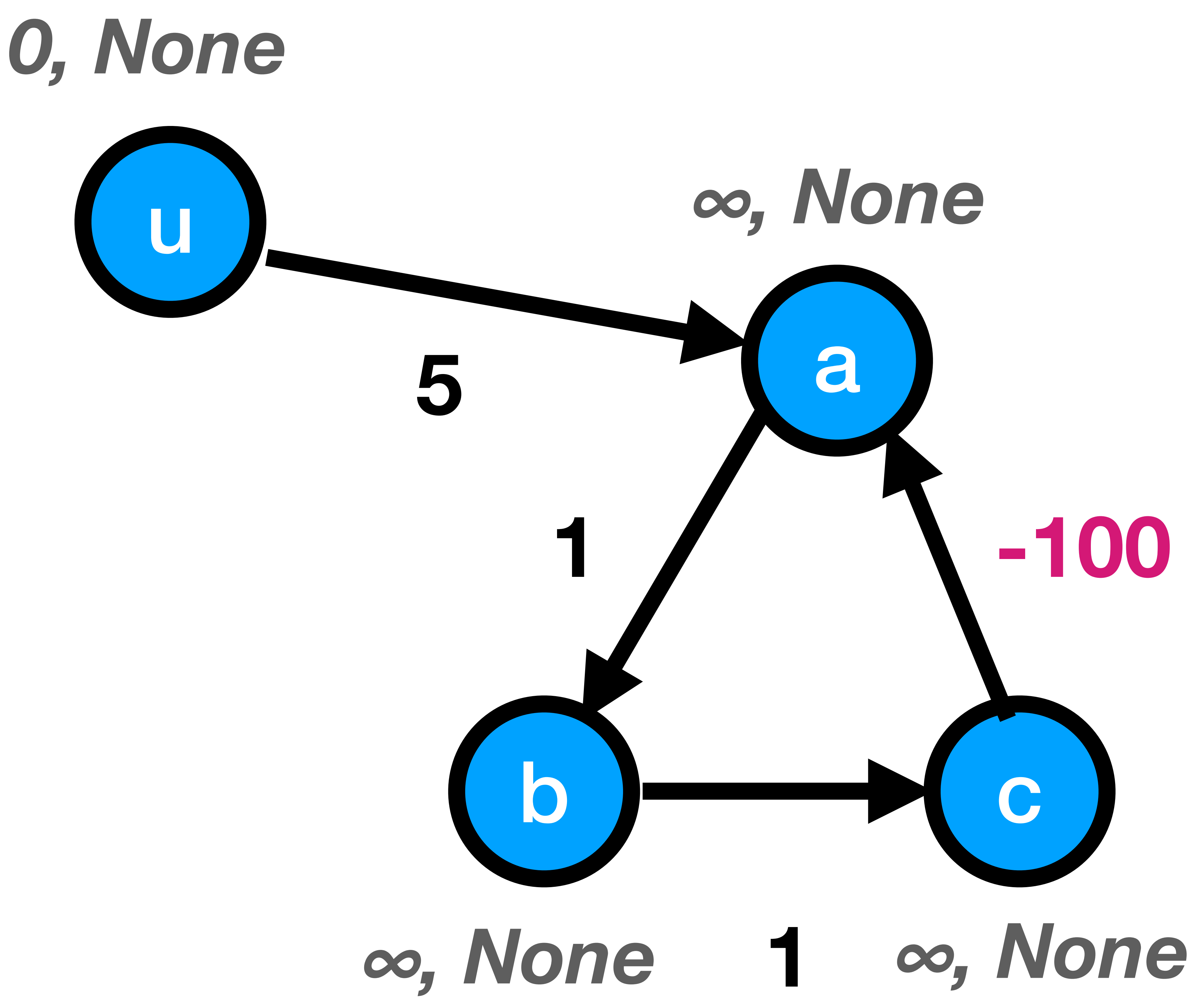
```
def BellmanFord(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0
    for i in range(|V| - 1):
        for (v1, v2) in E:
            try_to_relax(v1, v2)
    # check for negative cycles
    for (v1, v2) in E:
        if d[v2] > d[v1] + w(v1, v2):
            return False
    return True, d, parent
```

Remember: *we don't reinitialize d, parent between passes!*

Runtime: O(V E)

Example: if negative cycle

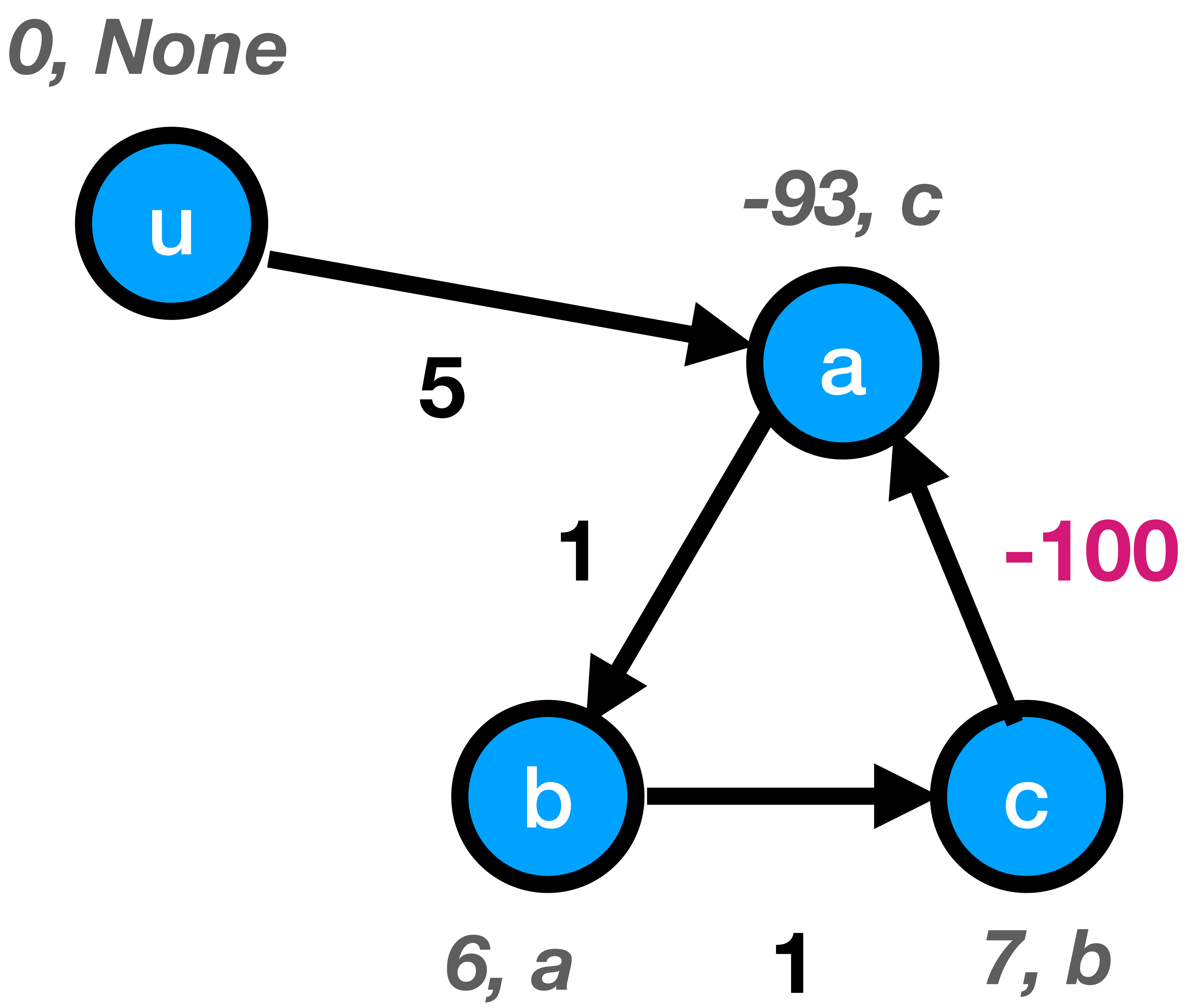
```
def BellmanFord(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0
    for i in range(|V| - 1):
        for (v1, v2) in E:
            try_to_relax(v1, v2)
    # check for negative cycles
    for (v1, v2) in E:
        if d[v2] > d[v1] + w(v1, v2):
            return False
    return True, d, parent
```



Example: if negative cycle

```
def BellmanFord(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0
    for i in range(|V| - 1):
        for (v1, v2) in E:
            try_to_relax(v1, v2)
    # check for negative cycles
    for (v1, v2) in E:
        if d[v2] > d[v1] + w(v1, v2):
            return False
    return True, d, parent
```

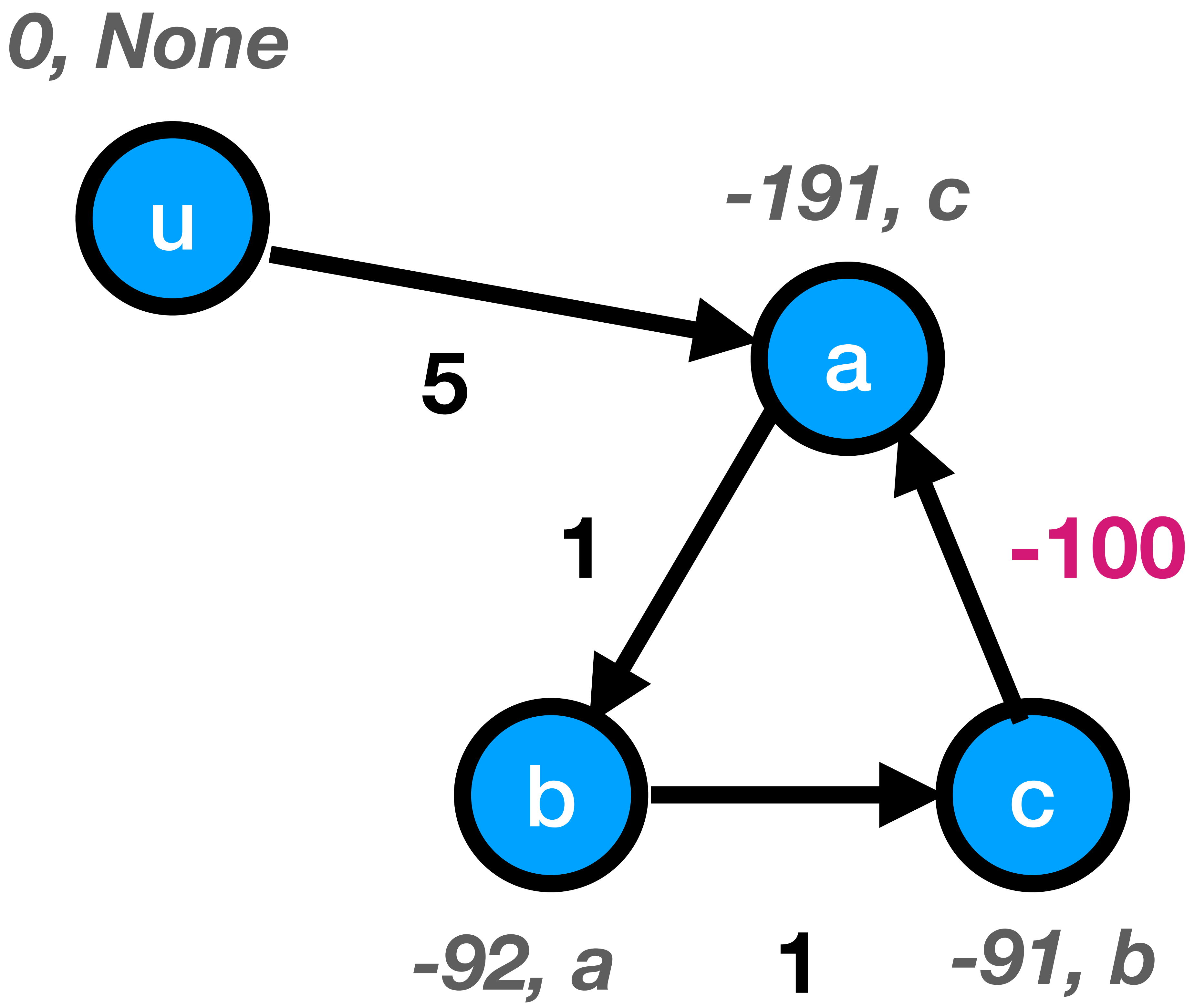
i = 0



Example: if negative cycle

```
def BellmanFord(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0
    for i in range(|V| - 1):
        for (v1, v2) in E:
            try_to_relax(v1, v2)
    # check for negative cycles
    for (v1, v2) in E:
        if d[v2] > d[v1] + w(v1, v2):
            return False
    return True, d, parent
```

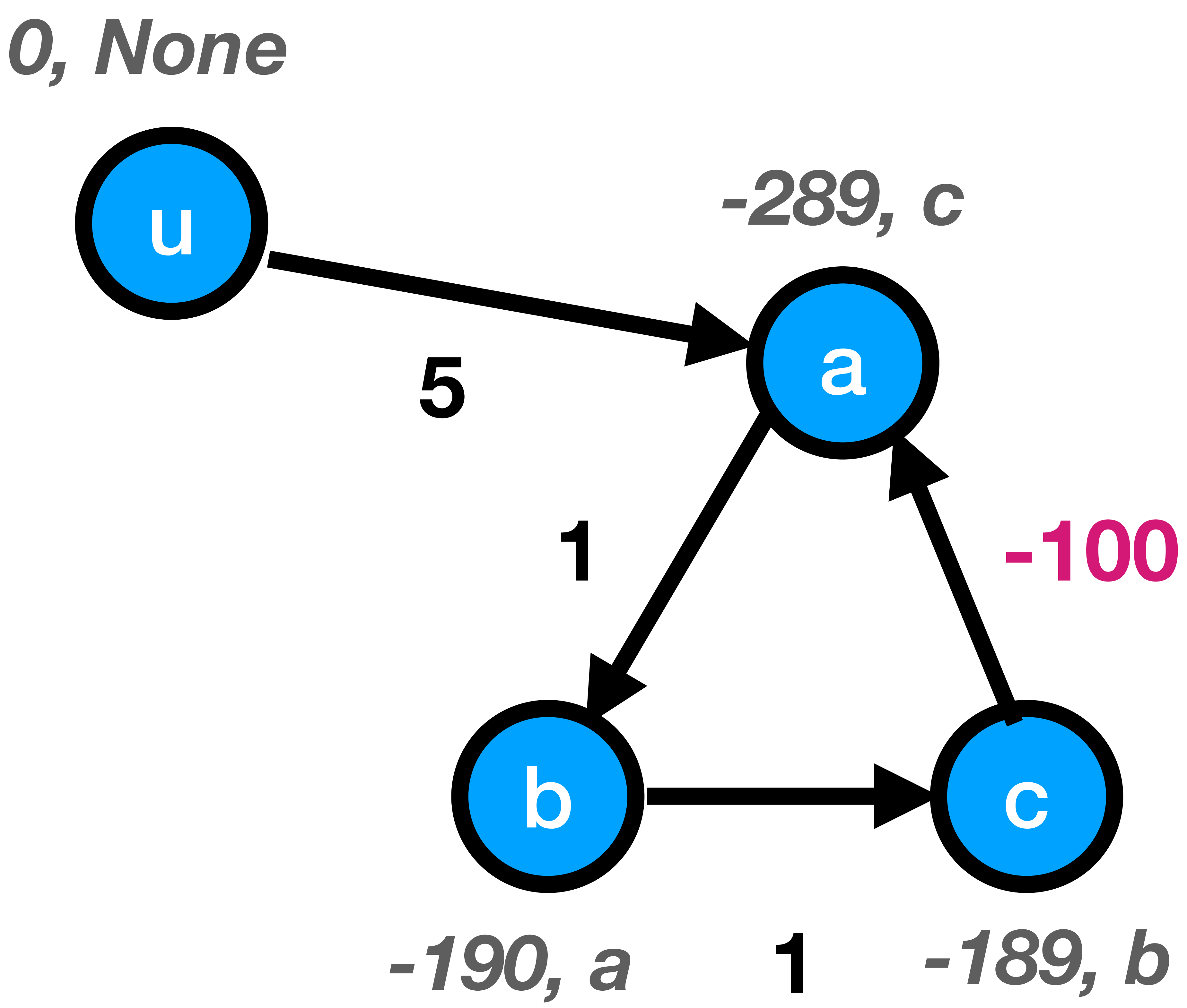
i = 1



Example: if negative cycle

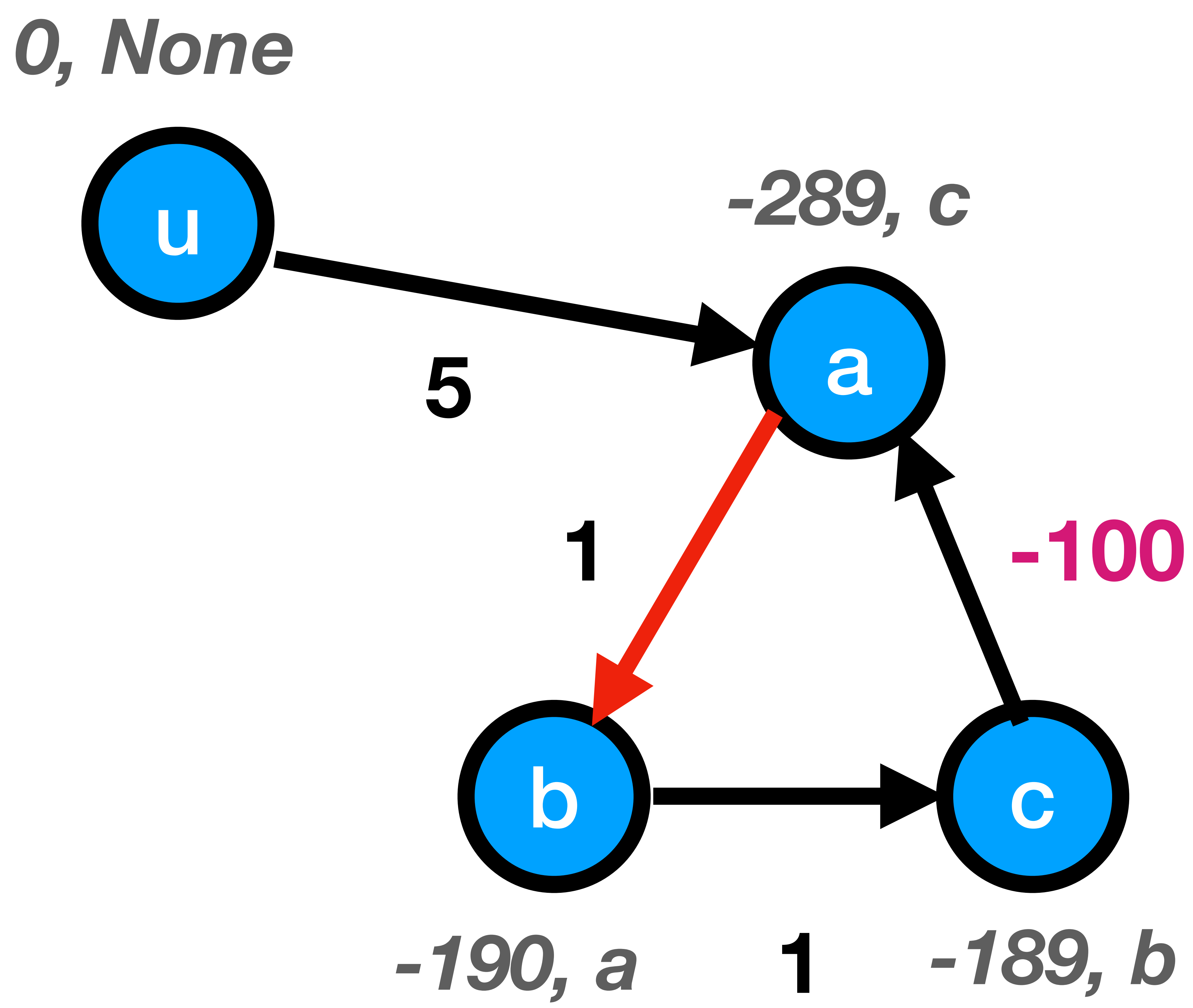
```
def BellmanFord(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0
    for i in range(|V| - 1):
        for (v1, v2) in E:
            try_to_relax(v1, v2)
    # check for negative cycles
    for (v1, v2) in E:
        if d[v2] > d[v1] + w(v1, v2):
            return False
    return True, d, parent
```

i = 2



Example: if negative cycle

```
def BellmanFord(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0
    for i in range(|V| - 1):
        for (v1, v2) in E:
            try_to_relax(v1, v2)
    # check for negative cycles
    for (v1, v2) in E:
        if d[v2] > d[v1] + w(v1, v2):
            return False
    return True, d, parent
```

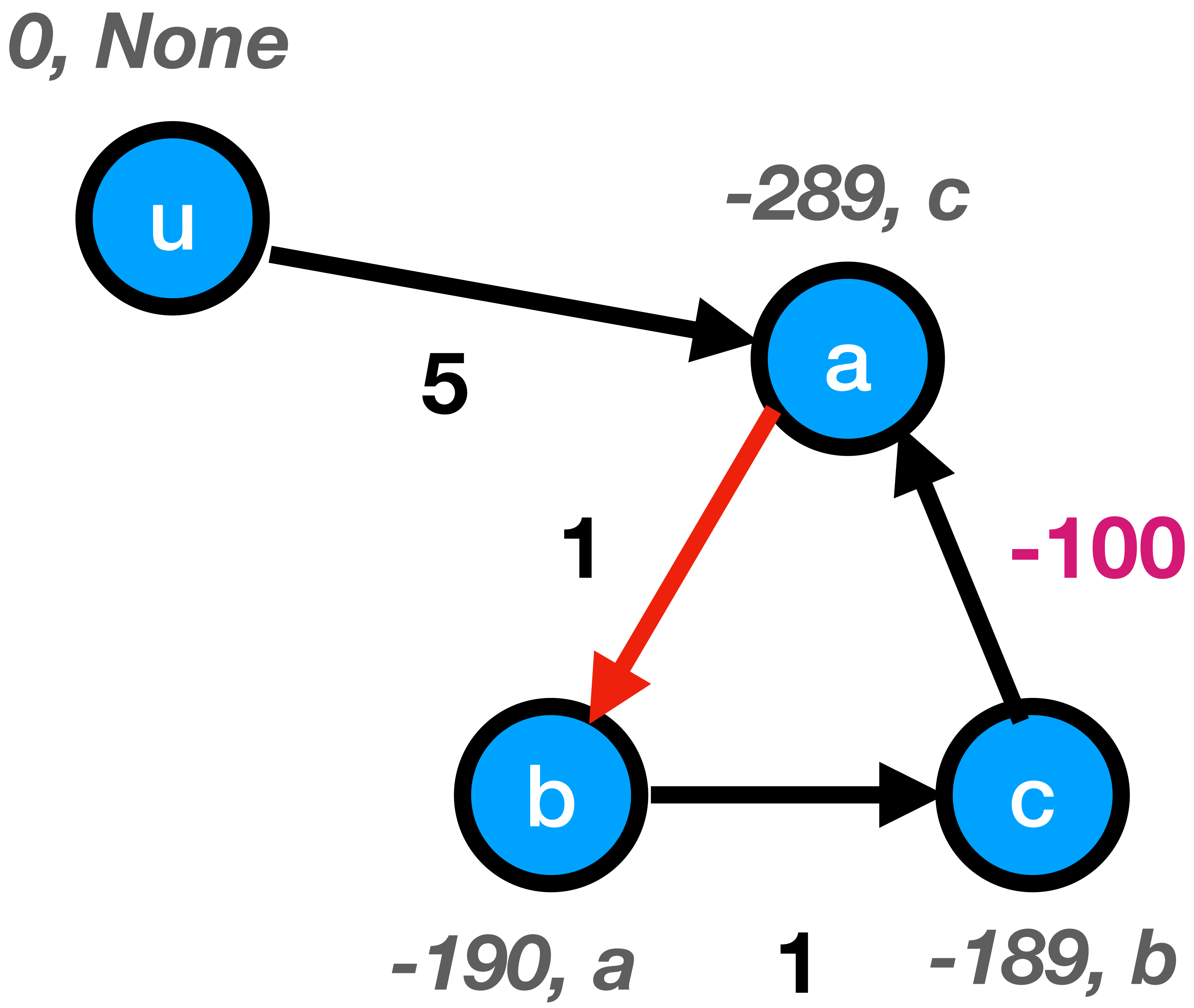


Correctness: if no negative cycle

- **Almost automatic from Path Relaxation Lemma!:**
    - Suppose there is a s.p. $(u, v_1, \ldots, v_k)$, where $k \leq |V| - 1$
        1. In first pass, we **relax (u,v_1)** *(together with all other edges!)*
        2. In second pass, we **relax (v_1,v_2)** *(together with all other edges!)*
        3. …
    - **Therefore** we relax all edges in order on this path, so PRL applies!
- **Bonus:** after iteration k, distances are correct for any v with a s.p. of k edges from u
- **Question:** Does the neg. cycle check return false? Why?

Correctness: if negative cycle

- Along a neg cycle (v_1, …, v_k)
$$\sum_{i=1}^{k} w(v_i, v_{i+1}) < 0$$
- Add to both sides:
$$\sum_{i=1}^{k} (d[v_i] + w(v_i, v_{i+1})) < \sum_{i=1}^{k} d[v_{i+1}]$$
- **Therefore** at least one edge on the cycle must violate check:
$$d[v_i] + w(v_i, v_{i+1}) < d[v_{i+1}]$$



Shortest paths on DAGs

- Directed Acyclic Graph
- **Problem:** Find an **O(V+E) algorithm** using path relaxation lemma + last Thursday's lecture!
- **Solution:**
  - Run topological sort
  - For each v in topological order, relax all outgoing edges (v, v')

Topological order: u, v1, v3, v2, v4