Dijkstra’s algorithm

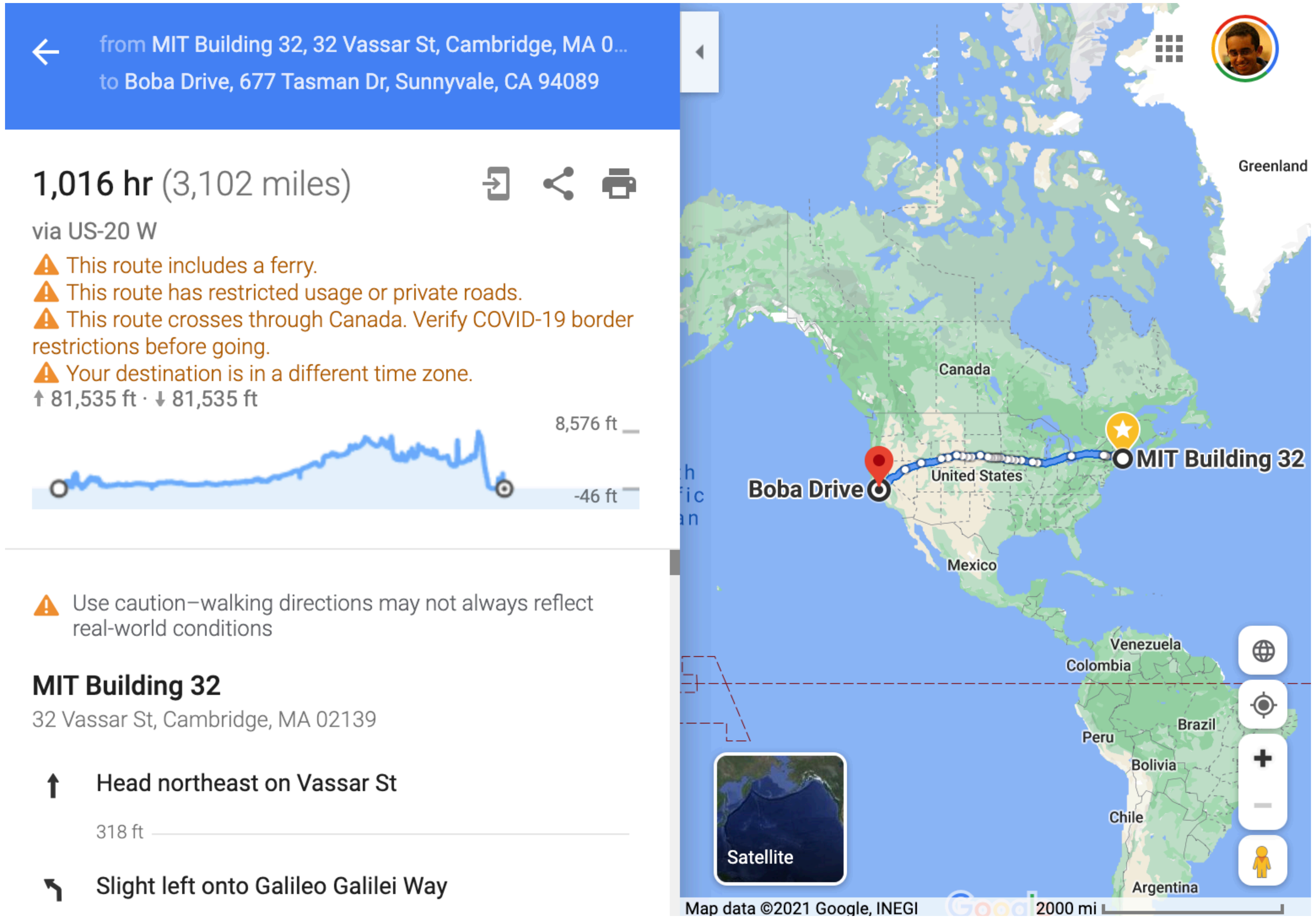
Anand Natarajan, 4/6/2021

Weighted graphs and paths

- Associate each (directed) edge (u,v) with a **weight w(u,v)** which is a real number
    - **For today: w(u,v) ≥ 0**
- A **path** from u_1 to u_n is a sequence of directed edges $(u_1, u_2), (u_2, u_3), (u_3, u_4)\ldots(u_{n-1}, u_n)$
- Total **cost/length** of a path is the **sum of the weights of the edges.**
- **Example: Google Maps** (edge weights are travel times)

Shortest paths

- Problem: given vertices u, v, find a **shortest (min cost) path** from u to v
    - Need not be unique!
- WLOG assume directed graph
- Actually, will solve **single source shortest paths (SSSP):** given u, find a shortest path from u to **every other vertex** in the graph
- **Define δ[v]**
  = length of shortest path from u to v
  = "distance from u to v"

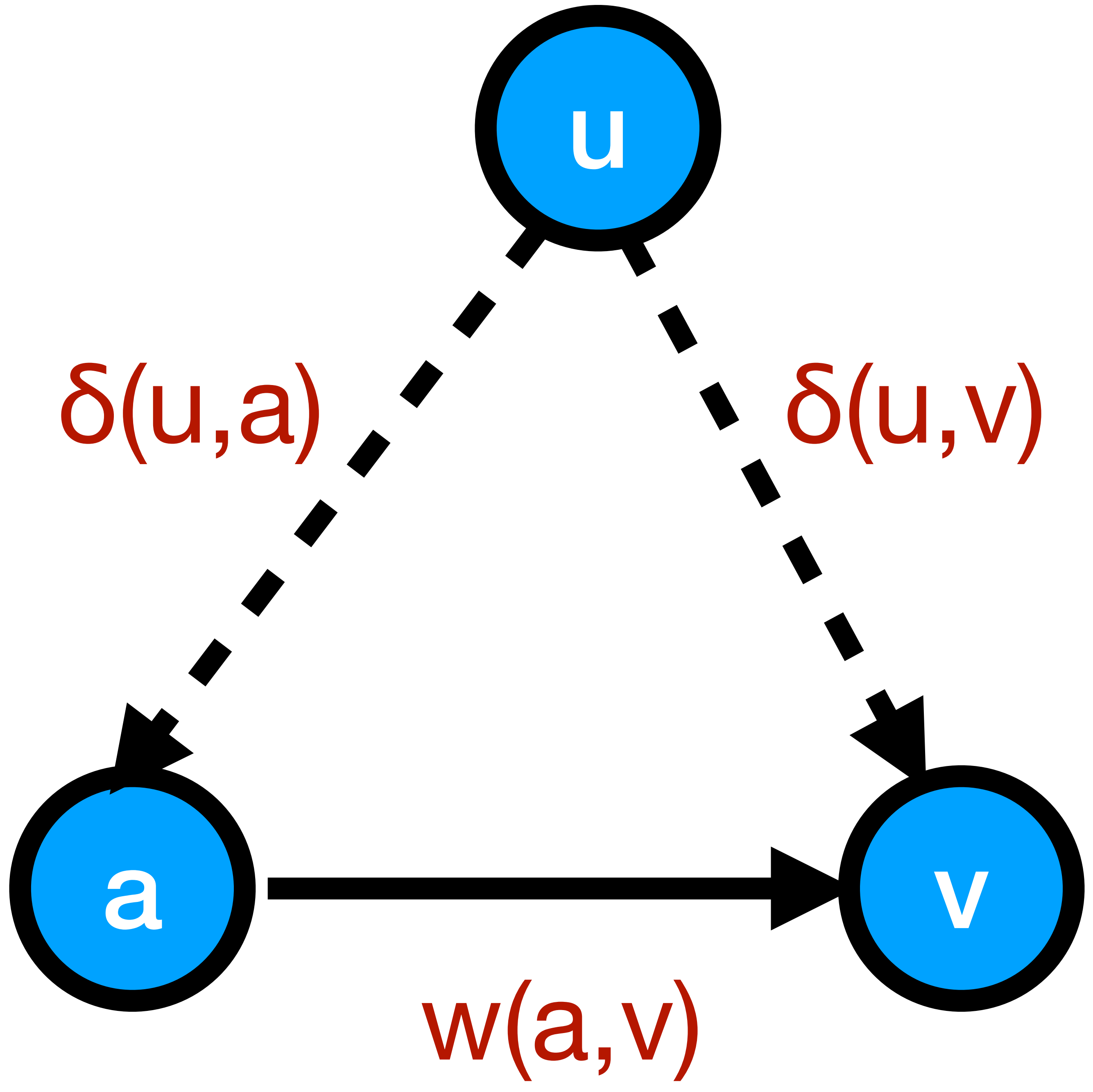


SP facts: optimal subproblems

- **Fact:** if a shortest path from u to v goes through a, then it contains a shortest path from u to a
- **Proof:** Suppose not! Then replace the u→a segment with the shortest path!
- **Conclusion:** let's try finding shortest paths to **intermediate vertices**
  Define:
  - **d[v] = our best guess of distance from u to v (≥ δ[v])**
  - **parent[v] = parent of v**
  - **Reconstruct shortest path by following parents backwards**

SP facts: the triangle inequality

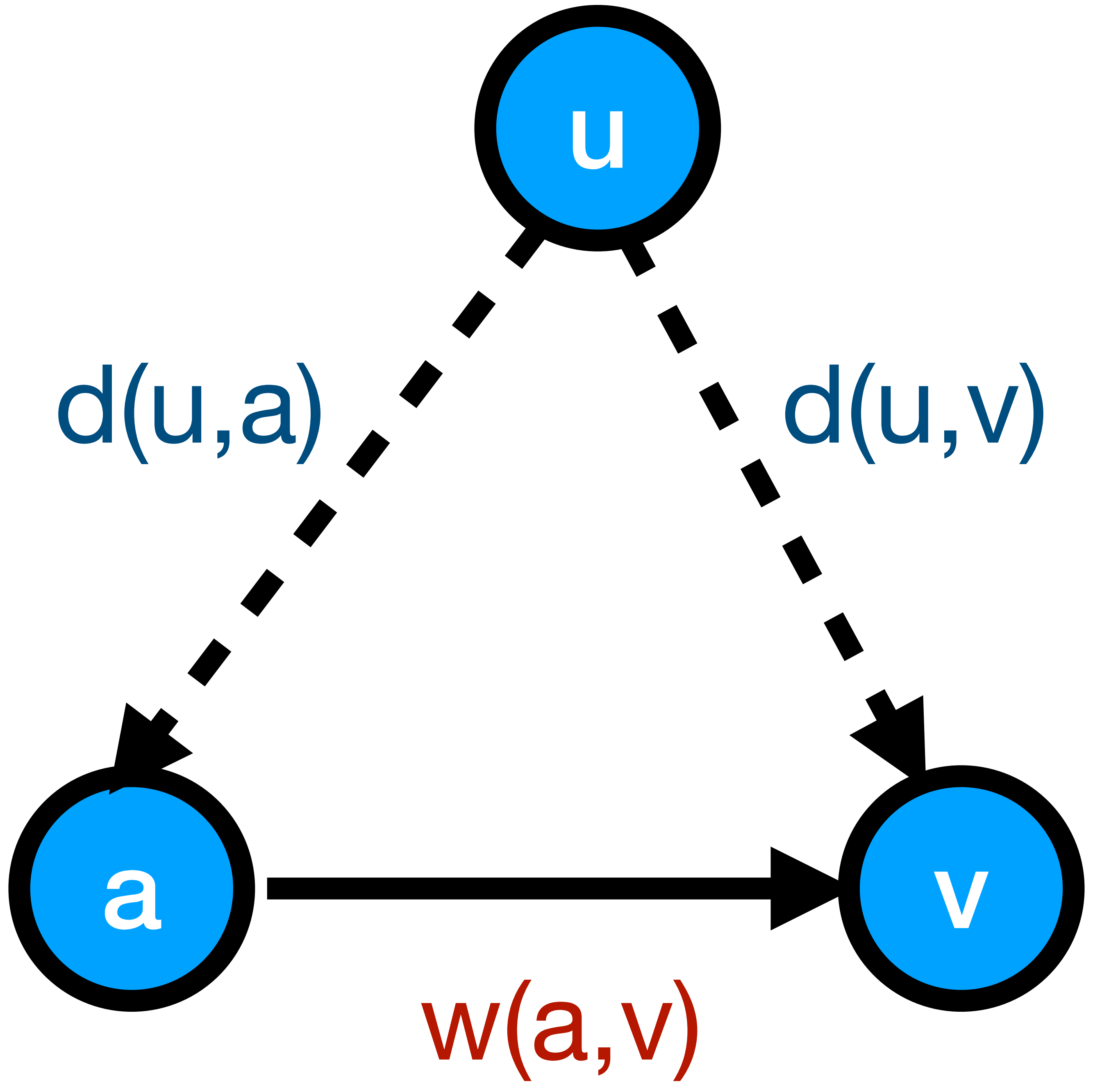
- For **any** u,v, a, $\delta[u,v] \leq \delta[u,a] + \delta[a,v]$
- **In particular** for any u, v, a, **if (a,v)** $\in$ **E**, then $\delta[u,v] \leq \delta[u,a] + w(a,v)$
- **Proof:** Suppose not! Then replace shortest path with path u → .. → a→ v
- **Conclusion:** we will use this to **improve** our guesses d[v]



Relaxation

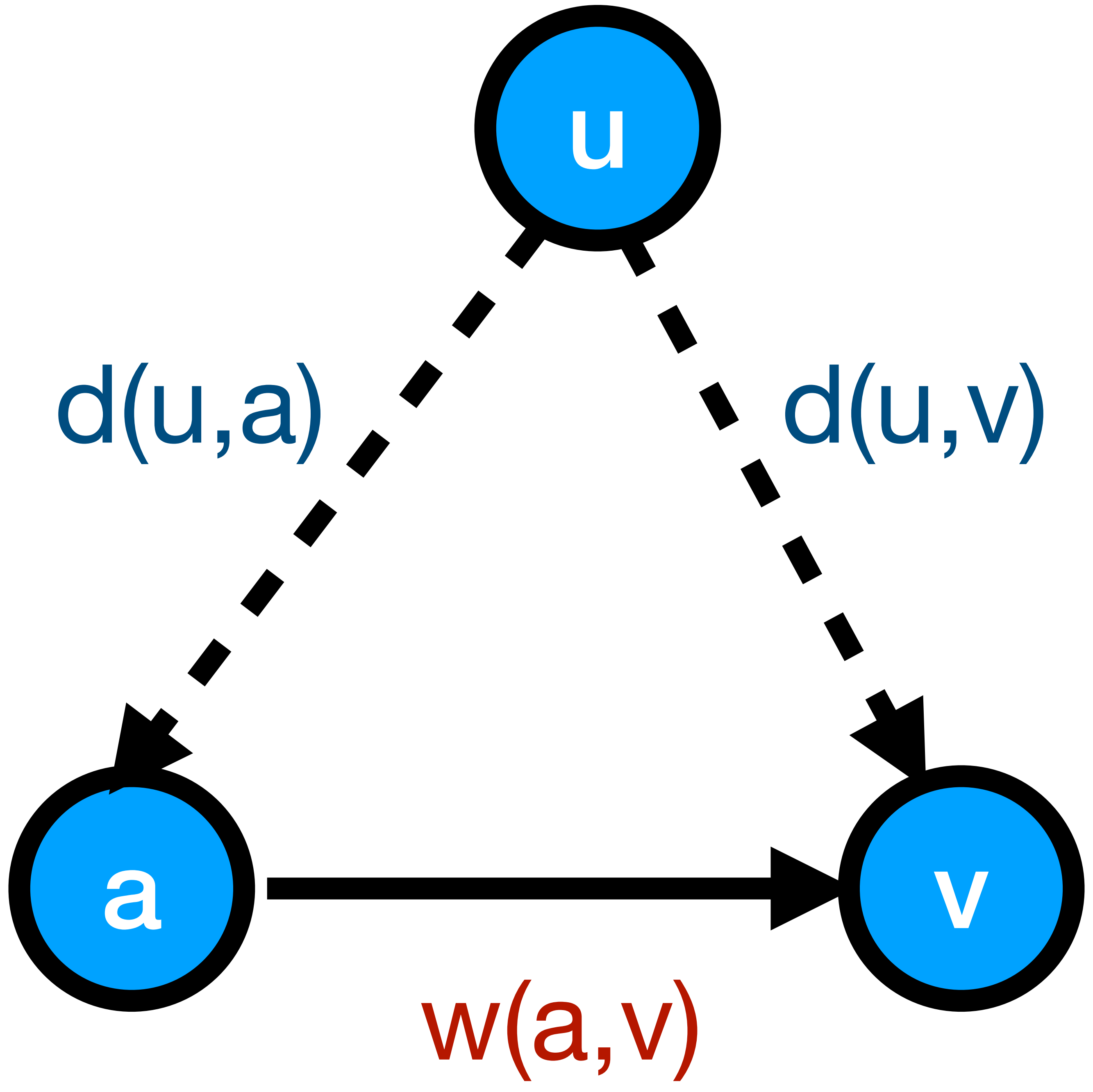
- Whenever d[a] violates the triangle inequality, we **feel stressed!!!**
- To **relax,** fix our estimate of d:
-

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



Relaxation is safe!

- ```
  def try_to_relax(a,v):
      if d[v] > d[a] + w(a,v):
          d[v] = d[a] + w(a,v)
          parent[v] = a
  ```
- **"Safety lemma:" relaxation preserves the invariant $\delta[v] \leq d[v]$**
- **Proof:** from triangle inequality


  ```

Relaxation algorithms

- A framework for a shortest path alg: **keep on relaxing until we can't anymore!**
- ```
  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
  ```
  ```

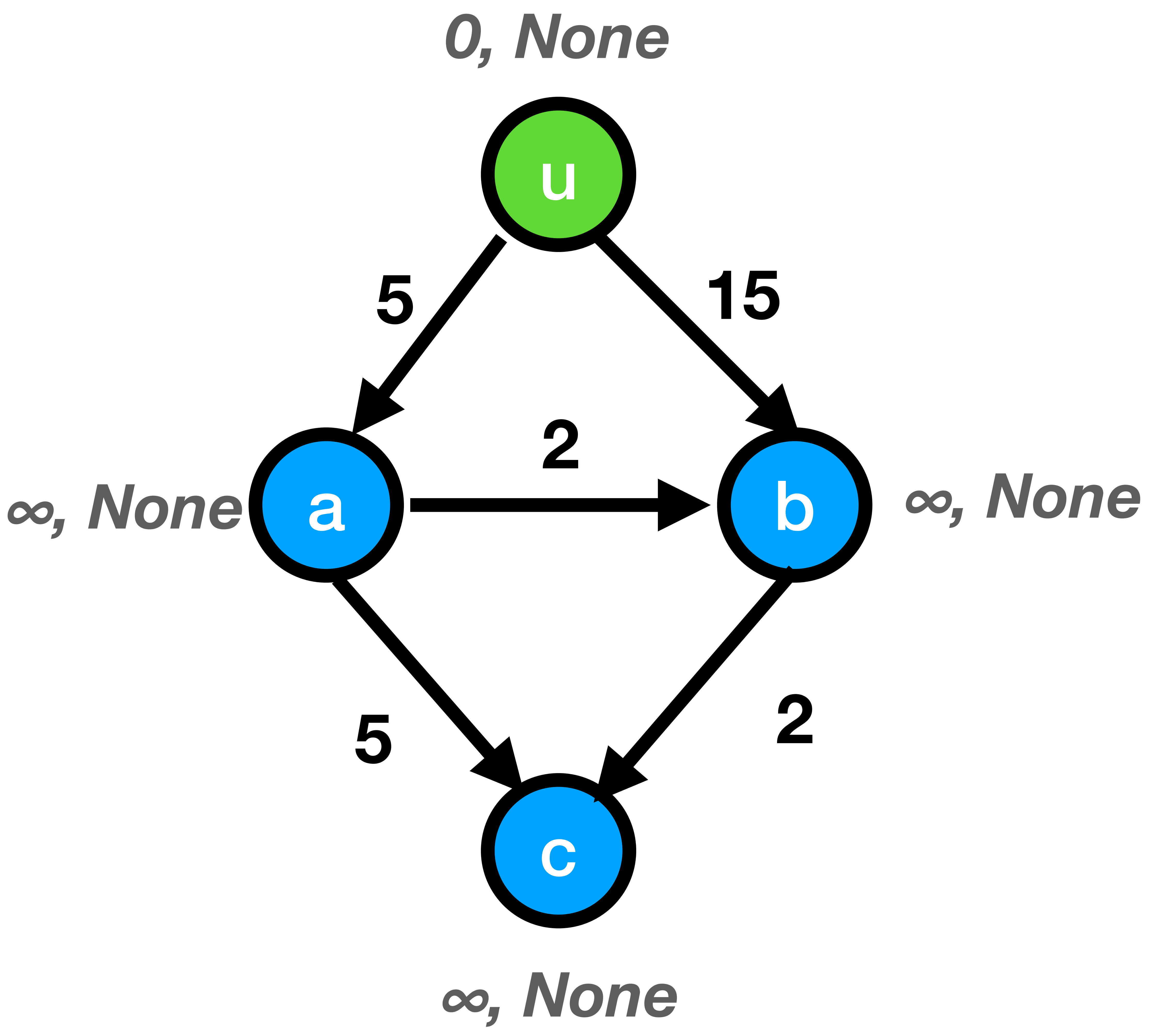
Dijkstra’s algorithm

- Build up a set **S** of vertices **with correct distances** by starting from u and **greedily** relaxing edges

- ```
  def dijkstra(G, u):
      for v in V:
          d[v] = ∞; parent[v] = None
      d[u] = 0; S = {u}
      pq = PriorityQueue.build(d)
      while pq:
          v1 = pq.pop_min(); S.add(v1)
          for (v1,v2) in E:
              try_to_relax(v1, v2)
              pq.update_key(v2)
      return d, parent
  ```
  ```

Dijkstra’s algorithm

- Build up a set **S** of vertices **with correct distances** by starting from u and **greedily** relaxing edges

- ```
  def dijkstra(G, u):
      for v in V:
          d[v] = ∞; parent[v] = None
      d[u] = 0; S = {u}
      pq = PriorityQueue.build(d)
      while pq:
          v1 = pq.pop_min(); S.add(v1)
          for (v1,v2) in E:
              try_to_relax(v1, v2)
              pq.update_key(v2)
      return d, parent
  ```
  ```

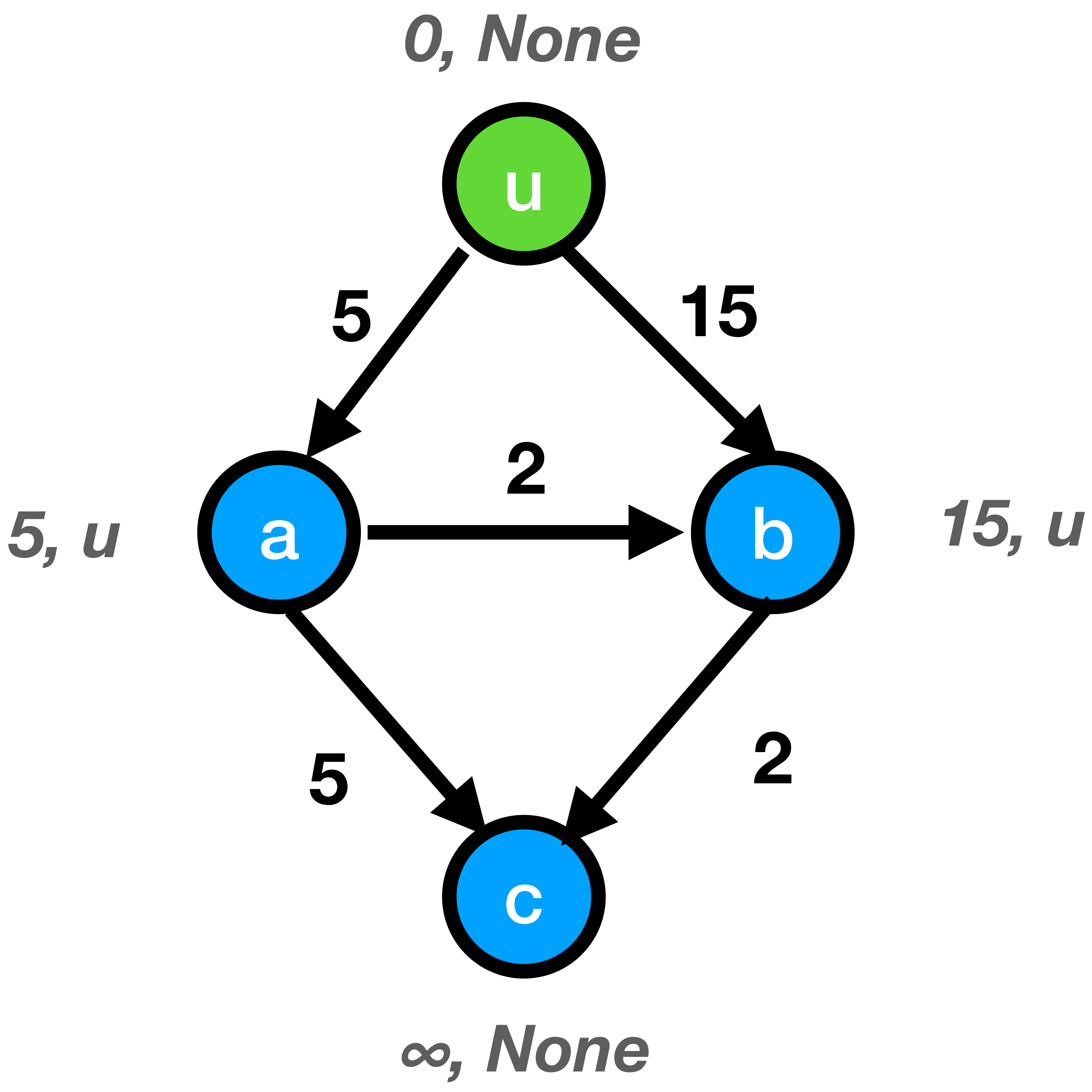
An example

```
def dijkstra(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0; S = {u}
    pq = PriorityQueue.build(d)
    while pq:
        v1 = pq.pop_min(); S.add(v1)
        for (v1,v2) in E:
            try_to_relax(v1, v2)
            pq.update_key(v2)
    return d, parent
```



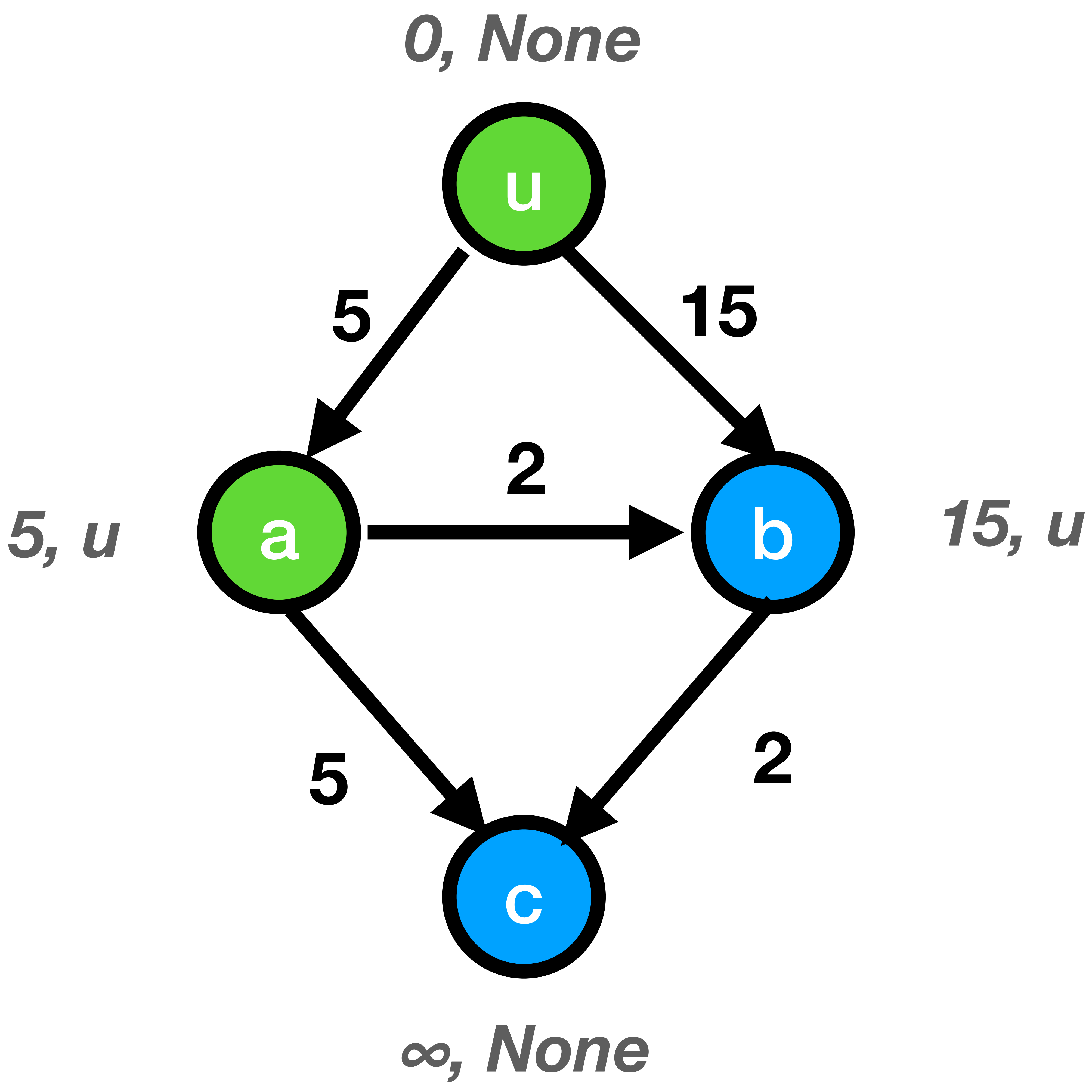
An example

```
def dijkstra(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0; S = {u}
    pq = PriorityQueue.build(d)
    while pq:
        v1 = pq.pop_min(); S.add(v1)
        for (v1,v2) in E:
            try_to_relax(v1, v2)
            pq.update_key(v2)
    return d, parent
```



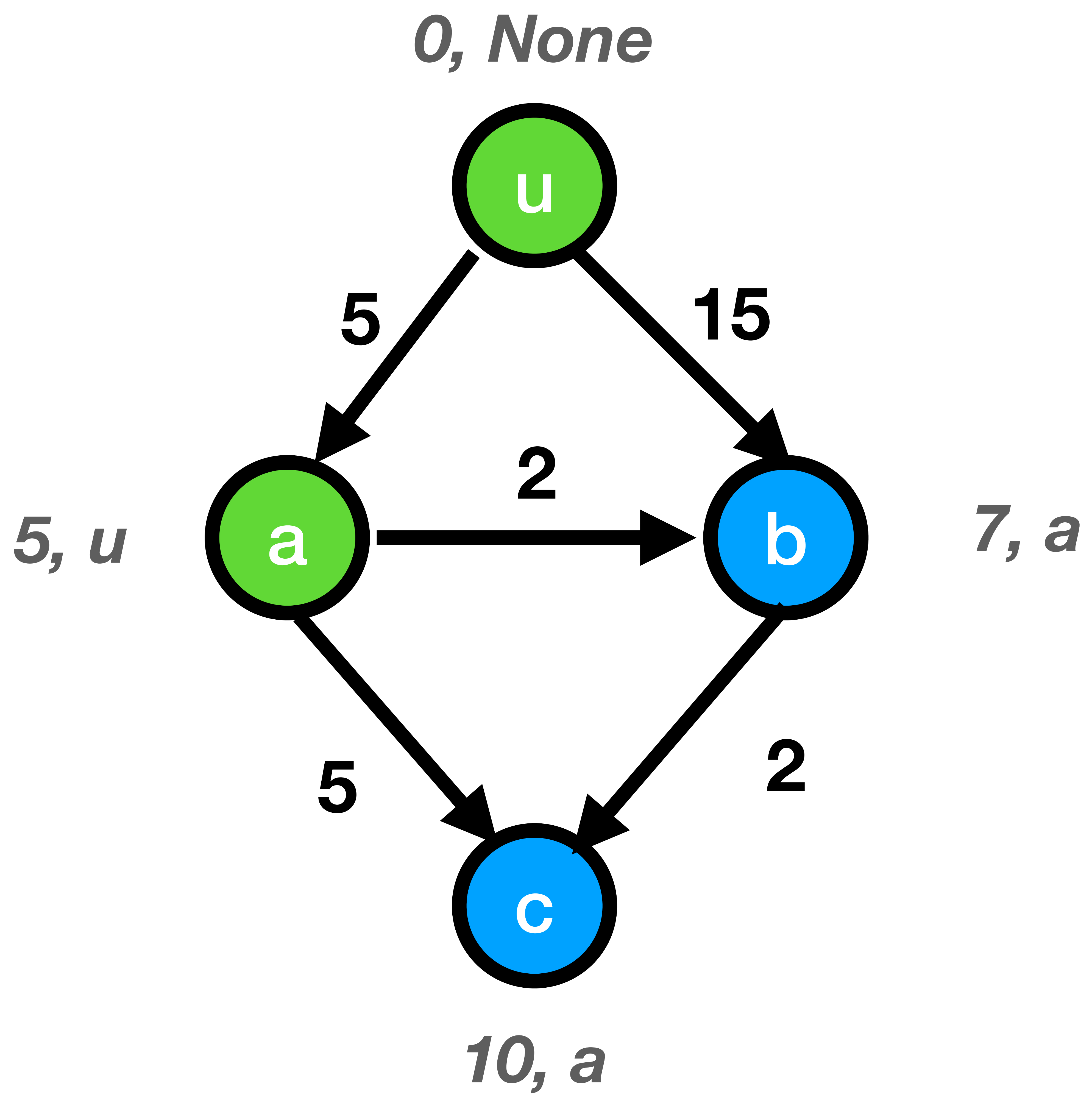
An example

```
def dijkstra(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0; S = {u}
    pq = PriorityQueue.build(d)
    while pq:
        v1 = pq.pop_min(); S.add(v1)
        for (v1,v2) in E:
            try_to_relax(v1, v2)
            pq.update_key(v2)
    return d, parent
```



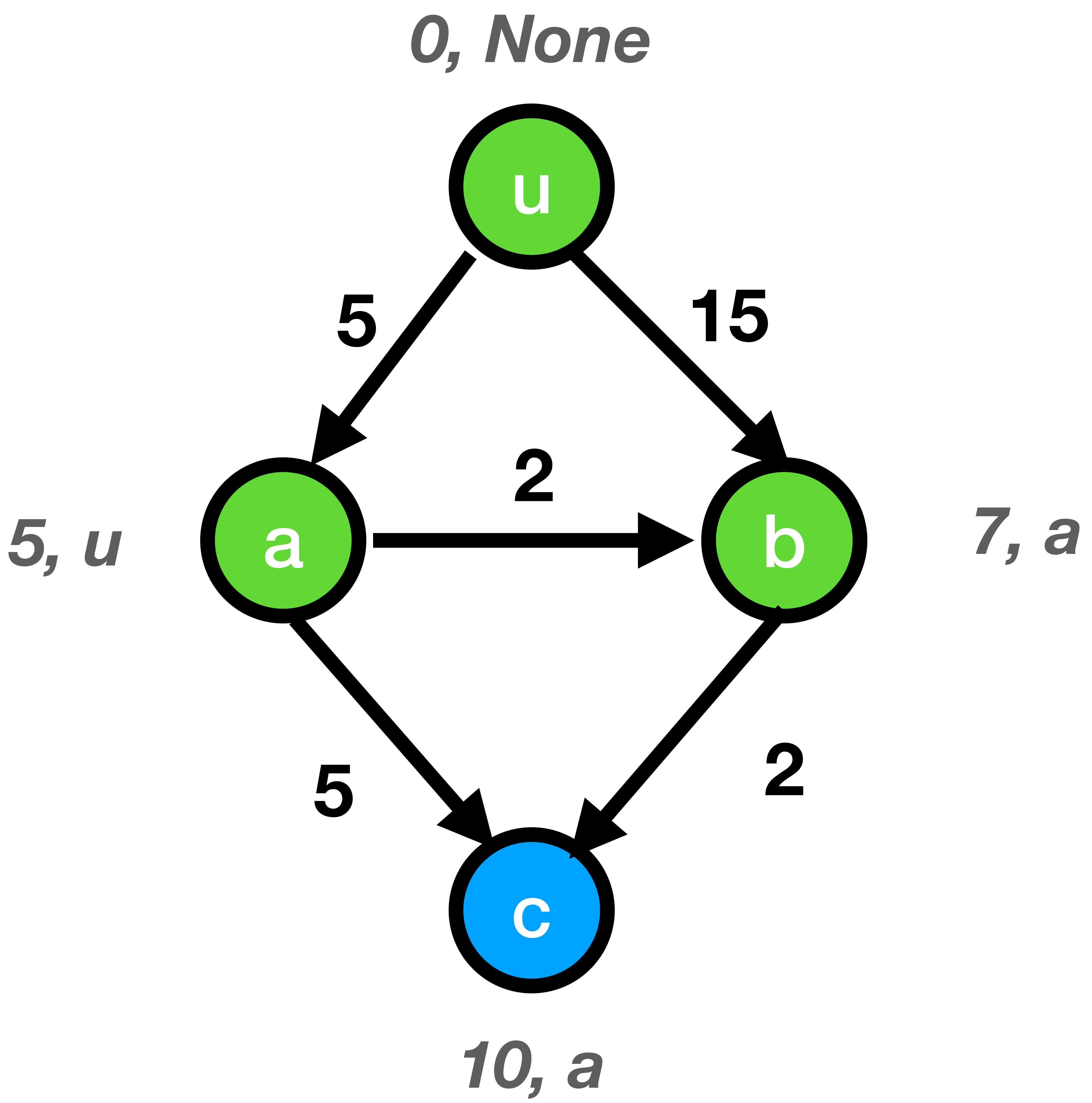
An example

```
def dijkstra(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0; S = {u}
    pq = PriorityQueue.build(d)
    while pq:
        v1 = pq.pop_min(); S.add(v1)
        for (v1,v2) in E:
            try_to_relax(v1, v2)
            pq.update_key(v2)
    return d, parent
```



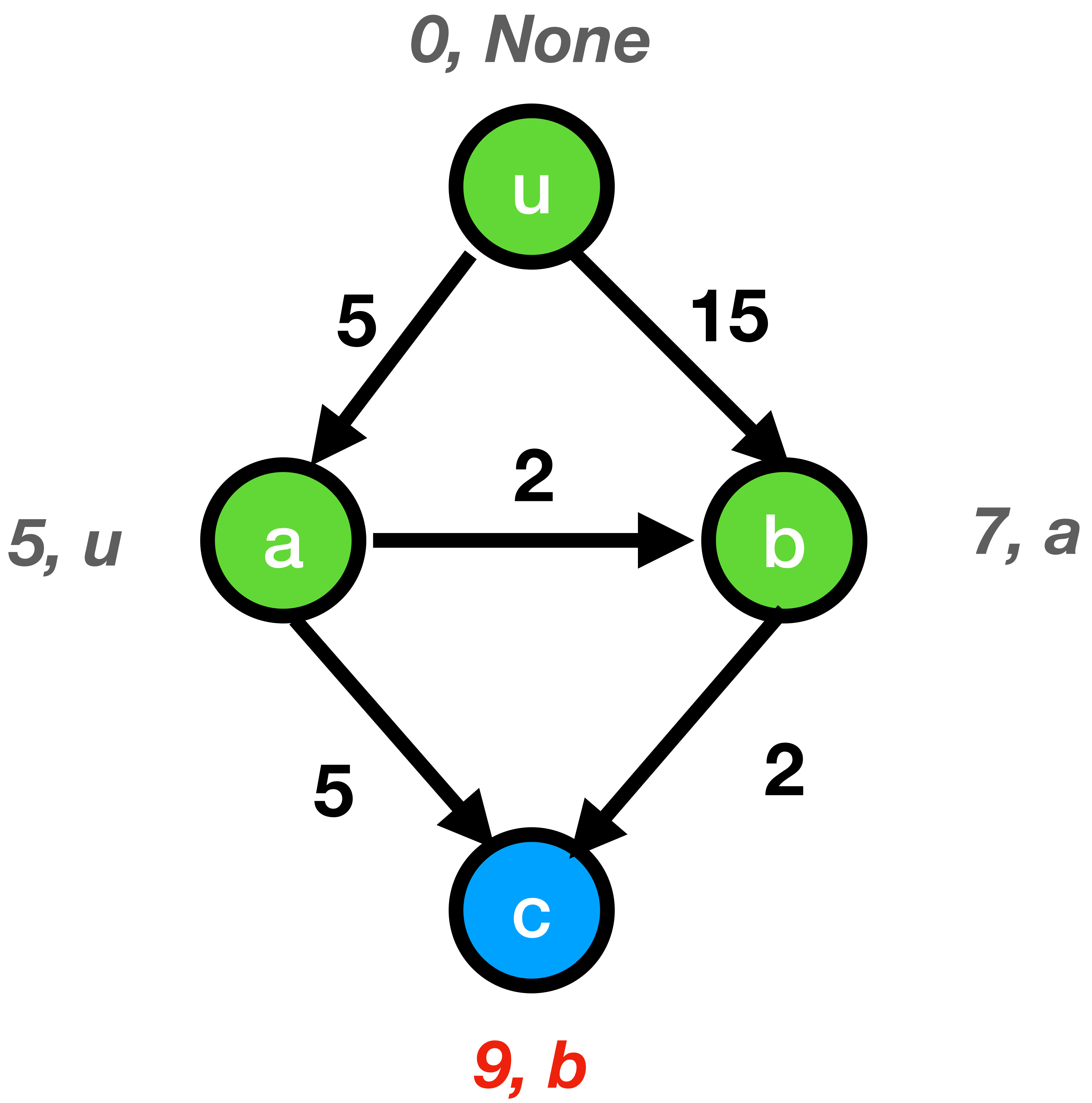
An example

```
def dijkstra(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0; S = {u}
    pq = PriorityQueue.build(d)
    while pq:
        v1 = pq.pop_min(); S.add(v1)
        for (v1,v2) in E:
            try_to_relax(v1, v2)
            pq.update_key(v2)
    return d, parent
```



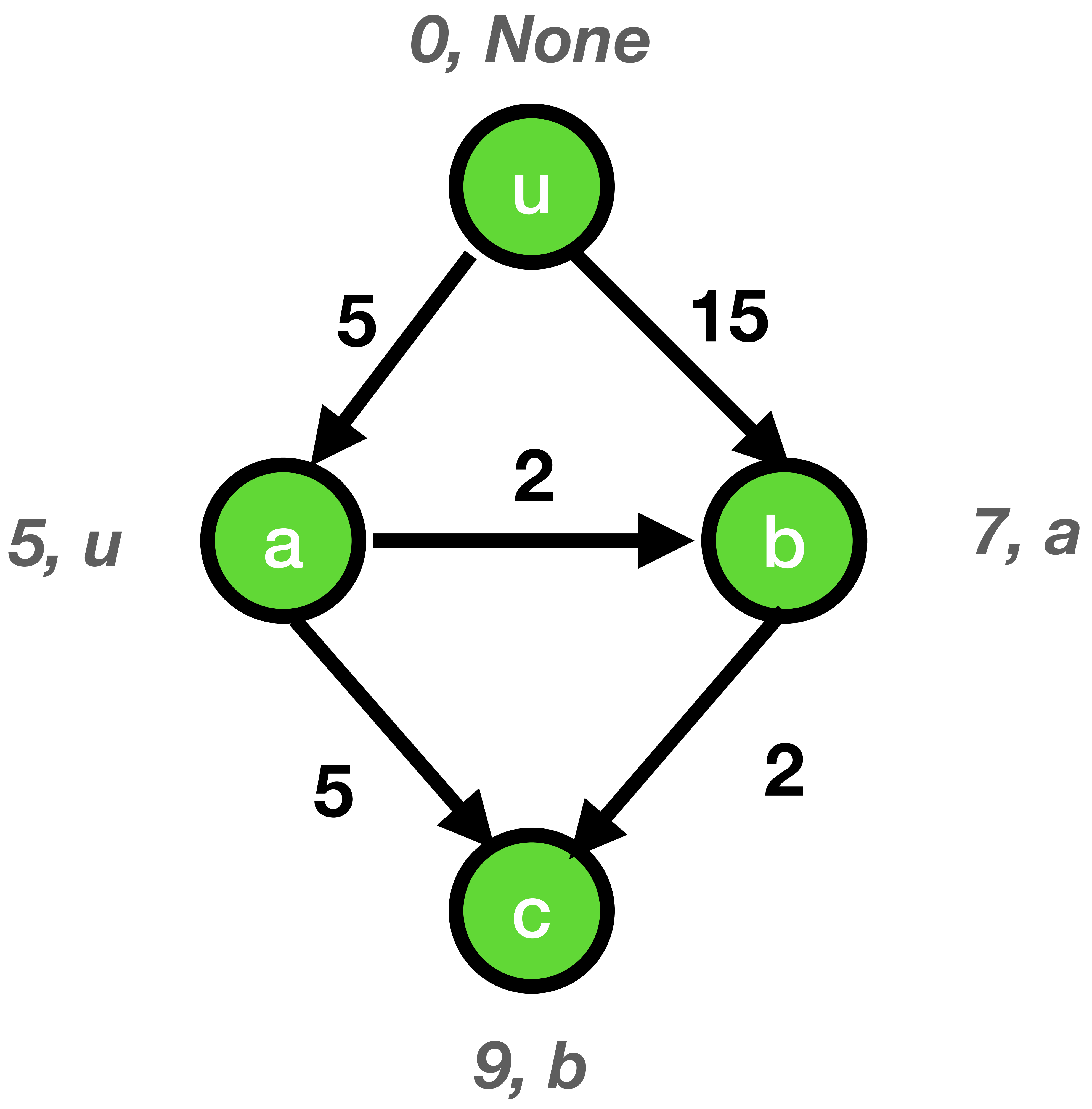
An example

```
def dijkstra(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0; S = {u}
    pq = PriorityQueue.build(d)
    while pq:
        v1 = pq.pop_min(); S.add(v1)
        for (v1,v2) in E:
            try_to_relax(v1, v2)
            pq.update_key(v2)
    return d, parent
```



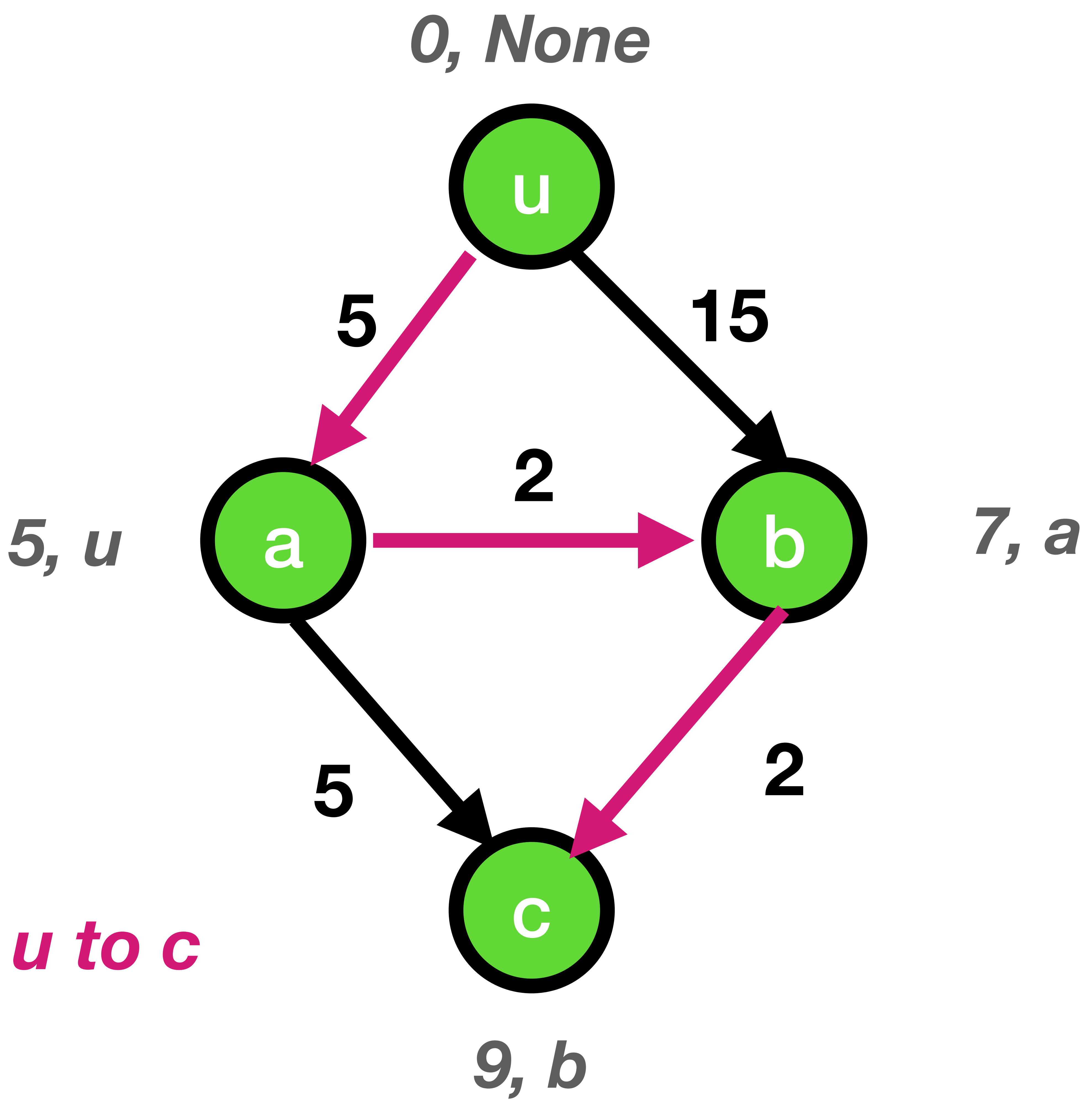
An example

```
def dijkstra(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0; S = {u}
    pq = PriorityQueue.build(d)
    while pq:
        v1 = pq.pop_min(); S.add(v1)
        for (v1,v2) in E:
            try_to_relax(v1, v2)
            pq.update_key(v2)
    return d, parent
```



An example

```
def dijkstra(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0; S = {u}
    pq = PriorityQueue.build(d)
    while pq:
        v1 = pq.pop_min(); S.add(v1)
        for (v1,v2) in E:
            try_to_relax(v1, v2)
            pq.update_key(v2)
    return d, parent
```



Shortest path from u to c

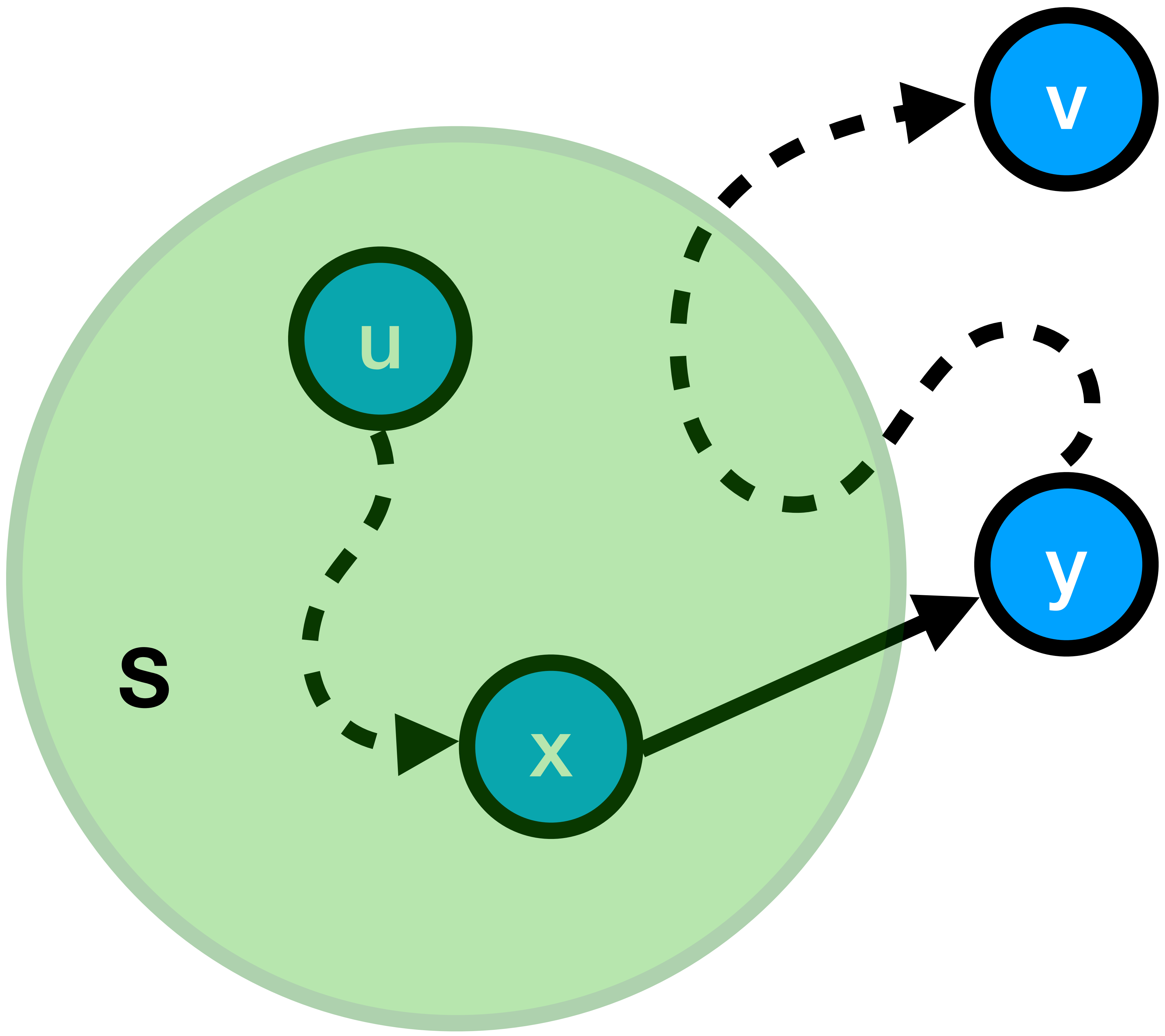
Some observations

```
def dijkstra(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0; S = {u}
    pq = PriorityQueue.build(d)
    while pq:
        v1 = pq.pop_min(); S.add(v1)
        for (v1,v2) in E:
            try_to_relax(v1, v2)
            pq.update_key(v2)
    return d, parent
```

- While loop runs **exactly V times**
- Once a vertex is added to S, we **never revisit it**
- So d[v], parent[v] **must be correct when we add v to S**

Correctness

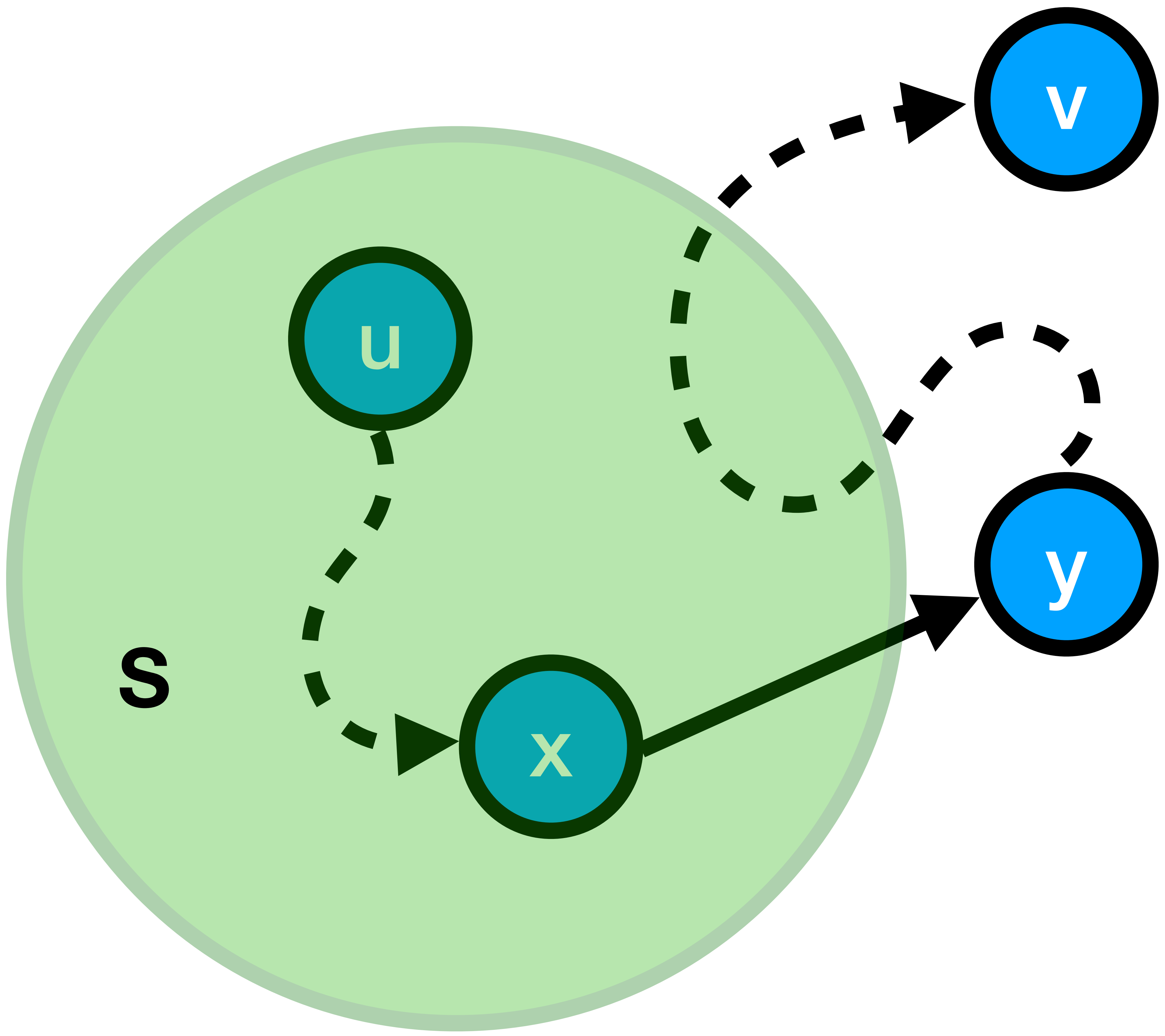
- **Claim: when we add a vertex v to S, d[v] = δ[v]**
- **Proof:** Suppose not! Take **v = first vertex where claim fails**
    - Take a s.p. u → … → x → y → .. v
      y is the first vertex outside of S in this path
    - x already in S, so d[x] = δ[x]



Correctness

$\boxed{d[x] = \delta[x]}$

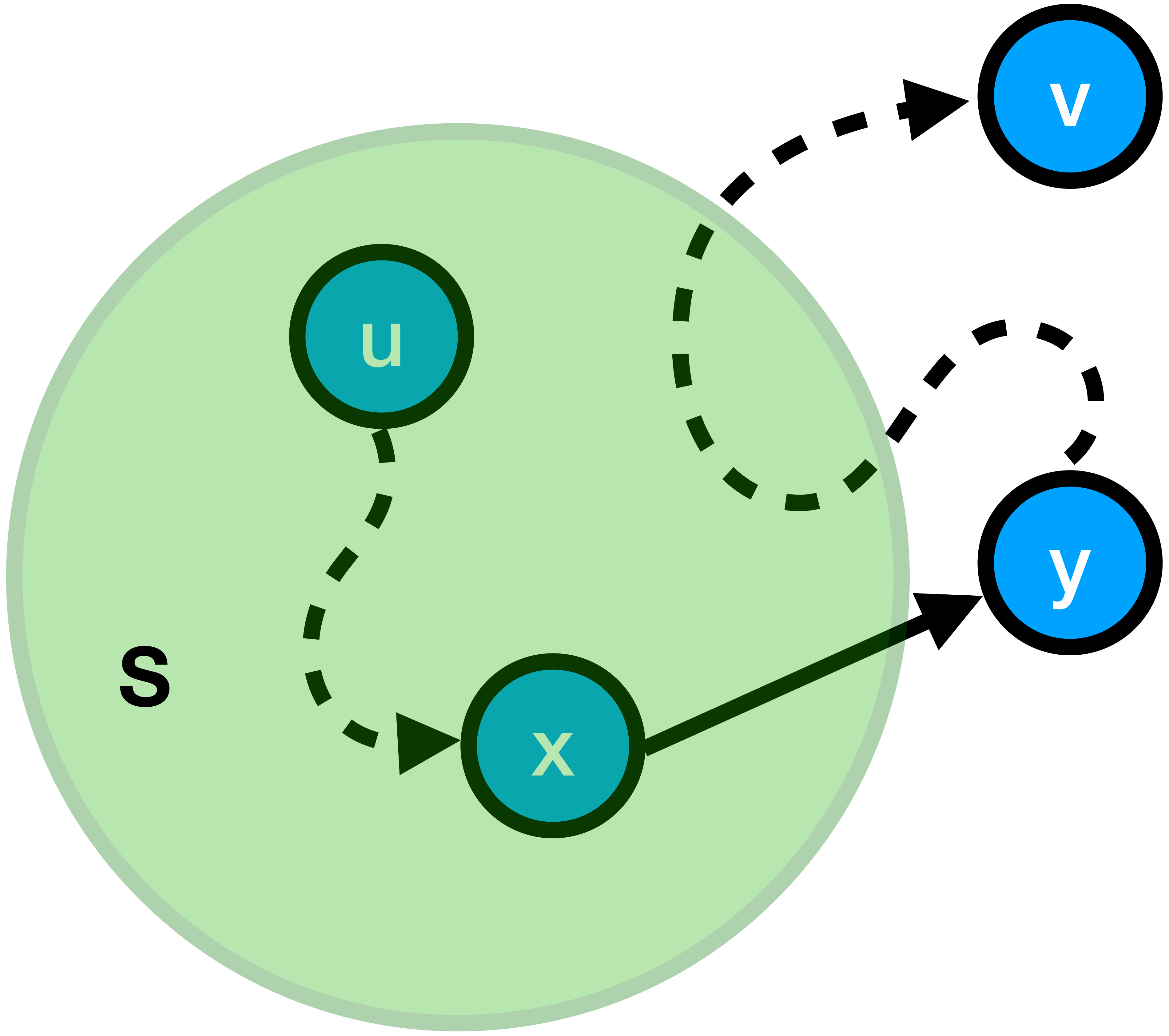
- **Claim: when we add a vertex v to S, d[v] = δ[v]**
- **Proof:** Suppose not! Take **v = first vertex where claim fails**
  - Take a s.p. u → … → x → y → .. v
    y is the first vertex outside of S in this path
  - u → … → x → y is a s.p.
    => δ[y] = δ[x] + w(x,y) = d[x] + w(x,y)



Correctness

$d[x] = \delta[x]$
$\delta[y] = d[x] + w(x,y)$

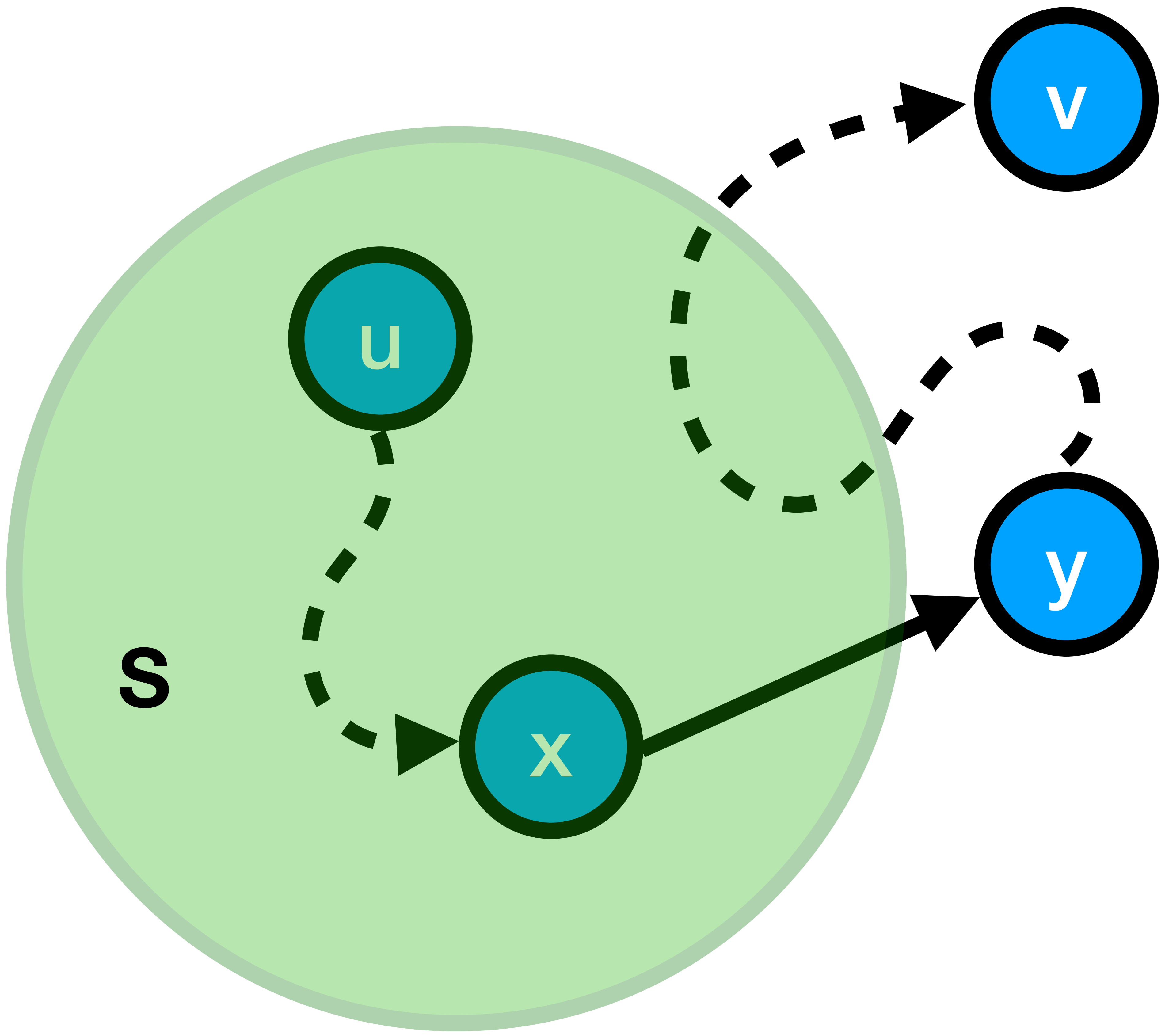
- **Claim: when we add a vertex v to S, d[v] = δ[v]**
- **Proof:** Suppose not! Take **v = first vertex where claim fails**
    - Take a s.p. u → … → x → y → .. v
      y is the first vertex outside of S in this path
    - x ∈ S, so (x,y) has **already** been relaxed
      => $d[y] \leq = d[x] + w(x,y) = \delta[y]$



Correctness

$d[x] = \delta[x]$
$\delta[y] = d[x] + w(x,y)$

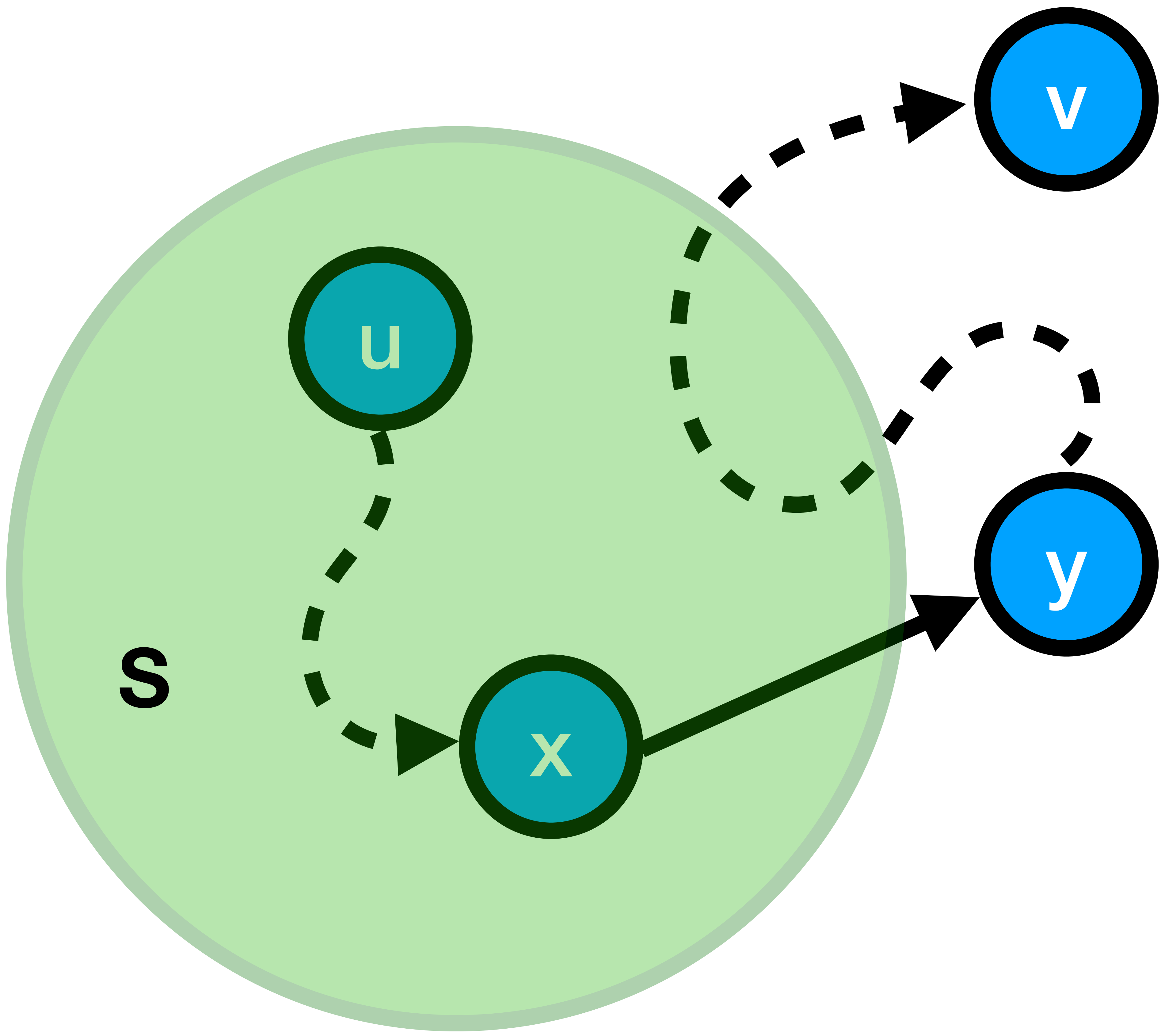
- **Claim: when we add a vertex v to S, $d[v] = \delta[v]$**
- **Proof:** Suppose not! Take **v = first vertex where claim fails**
  - Take a s.p. u → … → x → y → .. v
    y is the first vertex outside of S in this path
  - x ∈ S, so (x,y) has **already** been relaxed
    => $d[y] \leq = d[x] + w(x,y) = \delta[y]$
    => $d[y] = \delta[y]$



Correctness

$\delta[y] = d[y]$

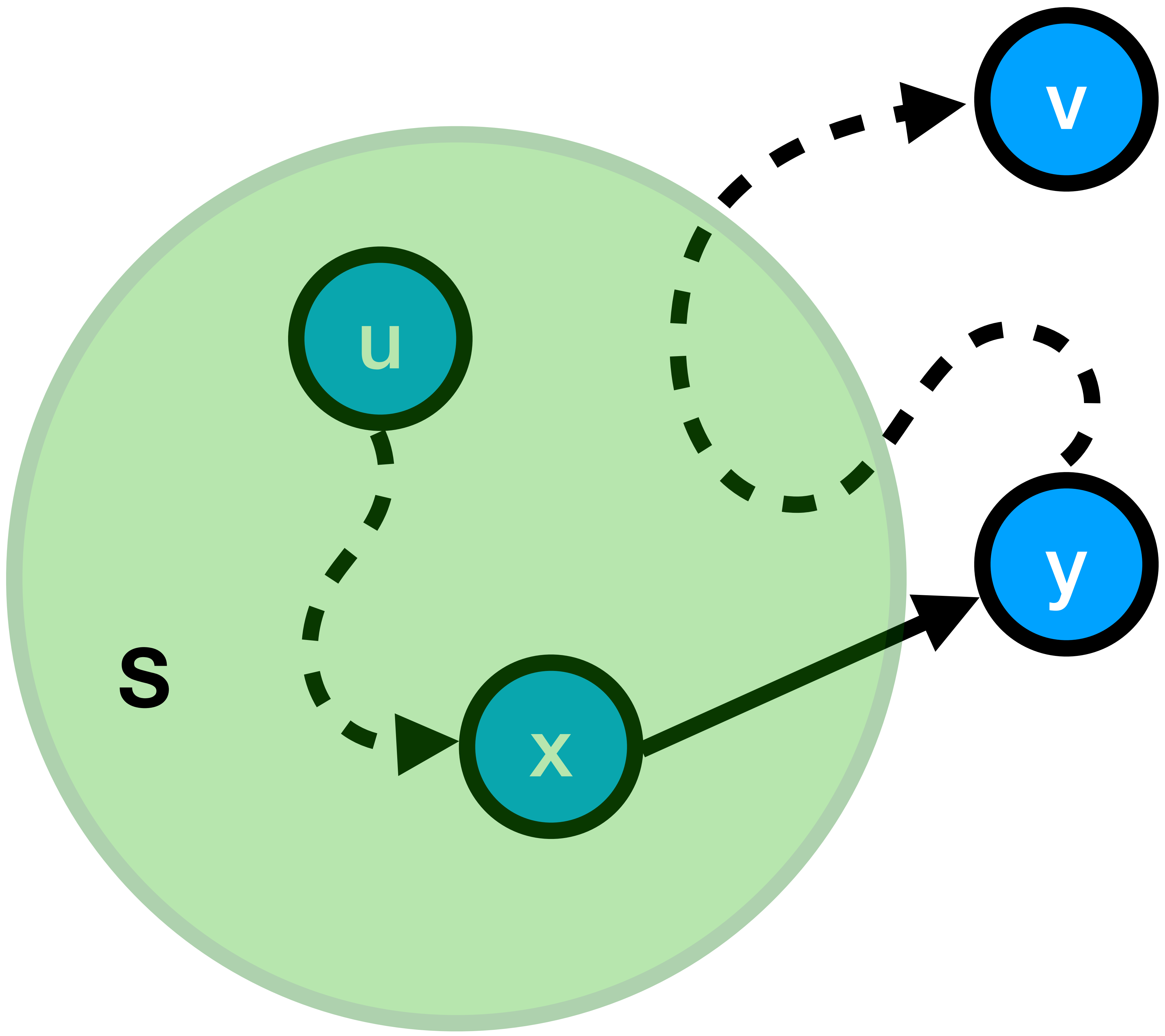
- **Claim: when we add a vertex v to S, $d[v] = \delta[v]$**
- **Proof:** Suppose not! Take **v = first vertex where claim fails**
  - y is in pq, but we popped v instead
    => $d[v] \leq d[y] = \delta[y]$



Correctness

$\delta[y] = d[y]$

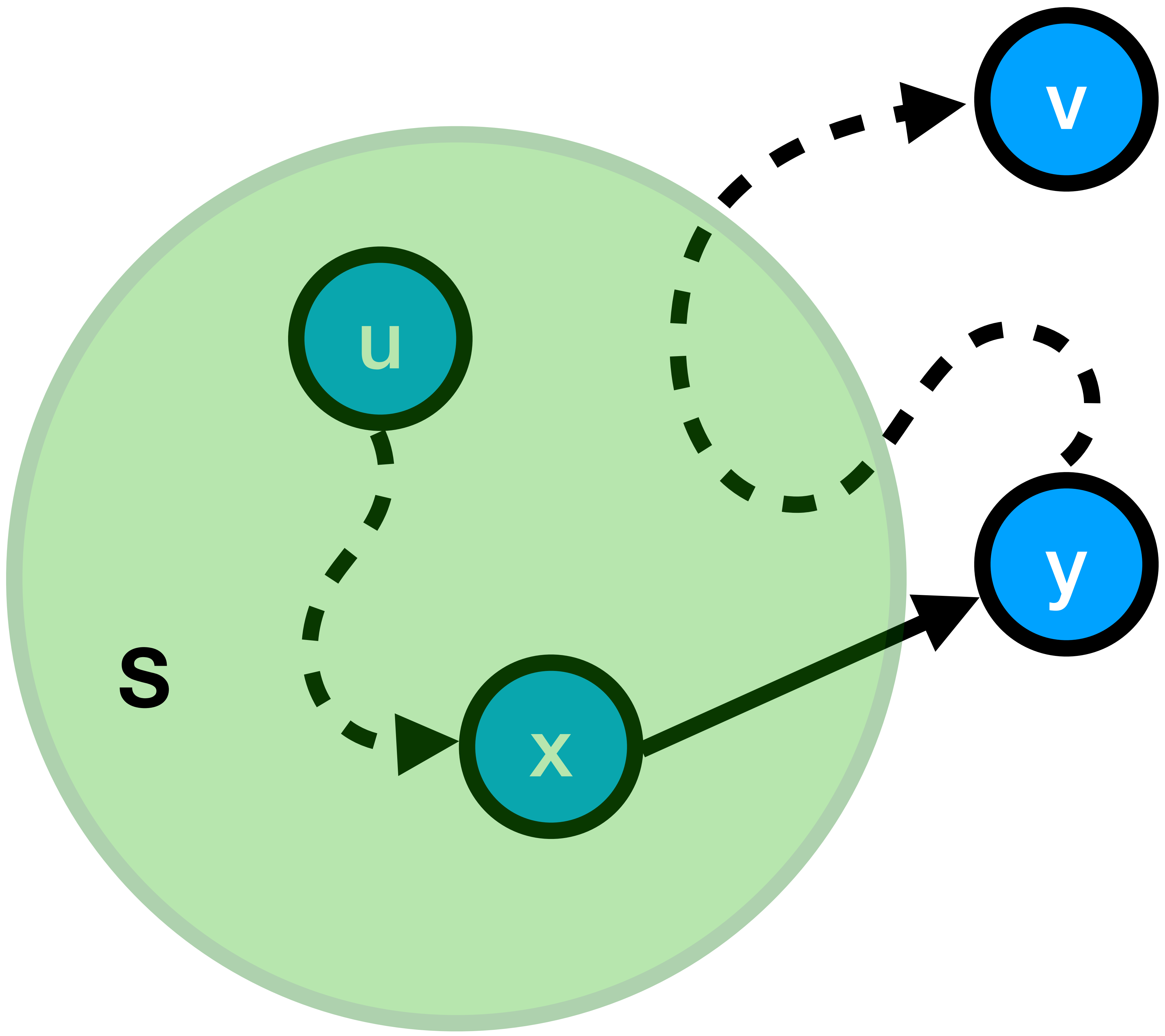
- **Claim: when we add a vertex v to S, d[v] = δ[v]**
- **Proof:** Suppose not! Take **v = first vertex where claim fails**
  - y is in pq, but we popped v instead
    => $d[v] \leq d[y] = \delta[y] \leq \delta[v] \leq d[v]$



Correctness

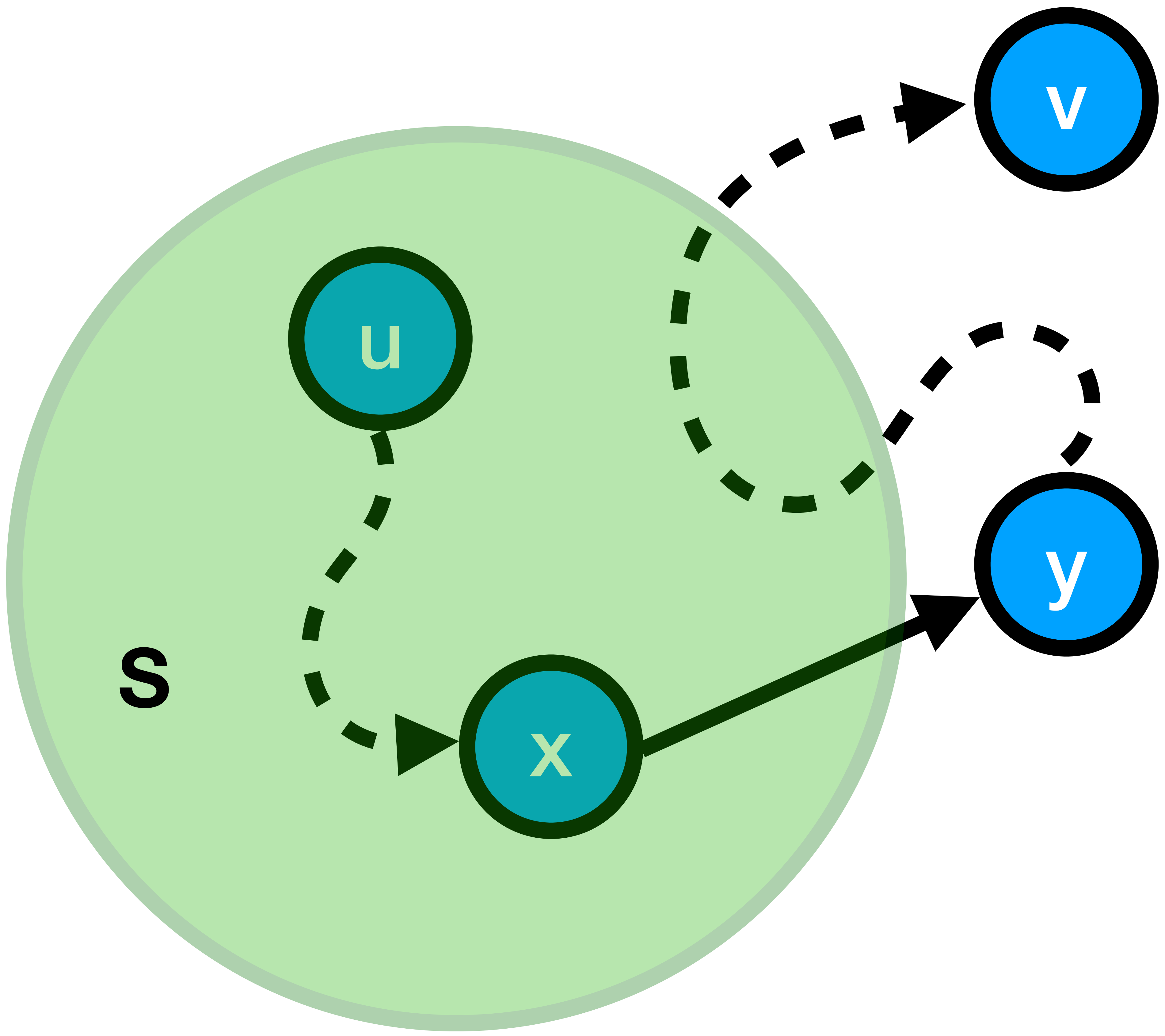
$\delta[y] = d[y]$

- **Claim: when we add a vertex v to S, d[v] = δ[v]**
- **Proof:** Suppose not! Take **v = first vertex where claim fails**
    - y is in pq, but we popped v instead
      => $d[v] \leq d[y] = \delta[y] \leq \delta[v] \leq d[v]$
    - **Therefore, d[v] = δ[v]**
      Contradiction!!!



Correctness

- **Claim: when we add a vertex v to S, d[v] = δ[v]**
- **Corollary:** Say you only care about the shortest path from u to a specific v. Then can **terminate immediately** once v is added to S!



Runtime

```
def dijkstra(G, u):
    for v in V:
        d[v] = ∞; parent[v] = None
    d[u] = 0; S = {u}
    pq = PriorityQueue.build(d)
    while pq:
        v1 = pq.pop_min(); S.add(v1)
        for (v1,v2) in E:
            try_to_relax(v1, v2)
            pq.update_key(v2)
    return d, parent
```

- Runtime is **dominated by priority queue operations**
  - pop_min() is called V times
  - update_key() is called E times
- **DAA:** $O(V^2 + E) = O(V^2)$
- **Heap:** $O((V + E) \log V)$
- **Fibonacci heap*:** $O(V \log V + E)$

  Not taught in 6.006, but you can **cite this runtime on psets/exams*

Shortest paths and AI

- Graphs represent **state spaces**
  - **Vertices = states, edges = actions**
  - **Weights = cost of action**
- An intelligent agent finds the **lowest cost** path to the goal
- Dijkstra works on on **implicitly represented** graphs
  - Like BFS on Noolbs (PS5), add vertices to pq **as you explore them**

What about negative weights?

- Dijkstra doesn't work!
- But a different relaxation algorithm does! **Tune in next time!**