CoNe Freiburg

IIF INSTITUT FÜR INFORMATIK FREIBURG

# Systeme II

## 6. Die Vermittlungsschicht

Thomas Janson°, Kristof Van Laerhoven*, Christian Ortolf°

Folien: Christian Schindelhauer°

Technische Fakultät

°: Rechnernetze und Telematik, *: Eingebettete Systeme

Albert-Ludwigs-Universität Freiburg

Version 24.04.2015

UNI FREIBURG

# Circuit Switching oder Packet Switching

- Circuit Switching
  - Etablierung einer Verbindung zwischen lokalen Benutzern durch Schaltstellen
    - mit expliziter Zuordnung von realen Schaltkreisen
    - oder expliziter Zuordnung von virtuellen Ressourcen, z.B. Slots
  - Quality of Service einfach, außer bei
    - Leitungsaufbau
    - Leitungsdauer
  - Problem
    - Statische Zuordnung
    - Ineffiziente Ausnutzung des Kommunikationsmedium bei dynamischer Last
  - Anwendung
    - Telefon
    - Telegraf
    - Funkverbindung

# Circuit Switching oder Packet Switching

- Packet Switching
  - Grundprinzip von IP
    - Daten werden in Pakete aufgeteilt und mit Absender/Ziel-Information unabhängig versandt
  - Problem: Quality of Service
    - Die Qualität der Verbindung hängt von einzelnen Paketen ab
    - Entweder Zwischenspeichern oder Paketverlust
  - Vorteil:
    - Effiziente Ausnutzung des Mediums bei dynamischer Last
- Resümee
  - Packet Switching hat Circuit Switching in praktisch allen Anwendungen abgelöst
  - Grund:
    - Effiziente Ausnutzung des Mediums

# Taktik der Schichten

- Transport
  - muss gewisse Flusskontrolle gewährleisten
  - z.B. Fairness zwischen gleichzeiten Datenströmen
- Vermittlung
  - Quality of Service (virtuelles Circuit Switching)
- Sicherung
  - Flusskontrolle zur Auslastung des Kanals

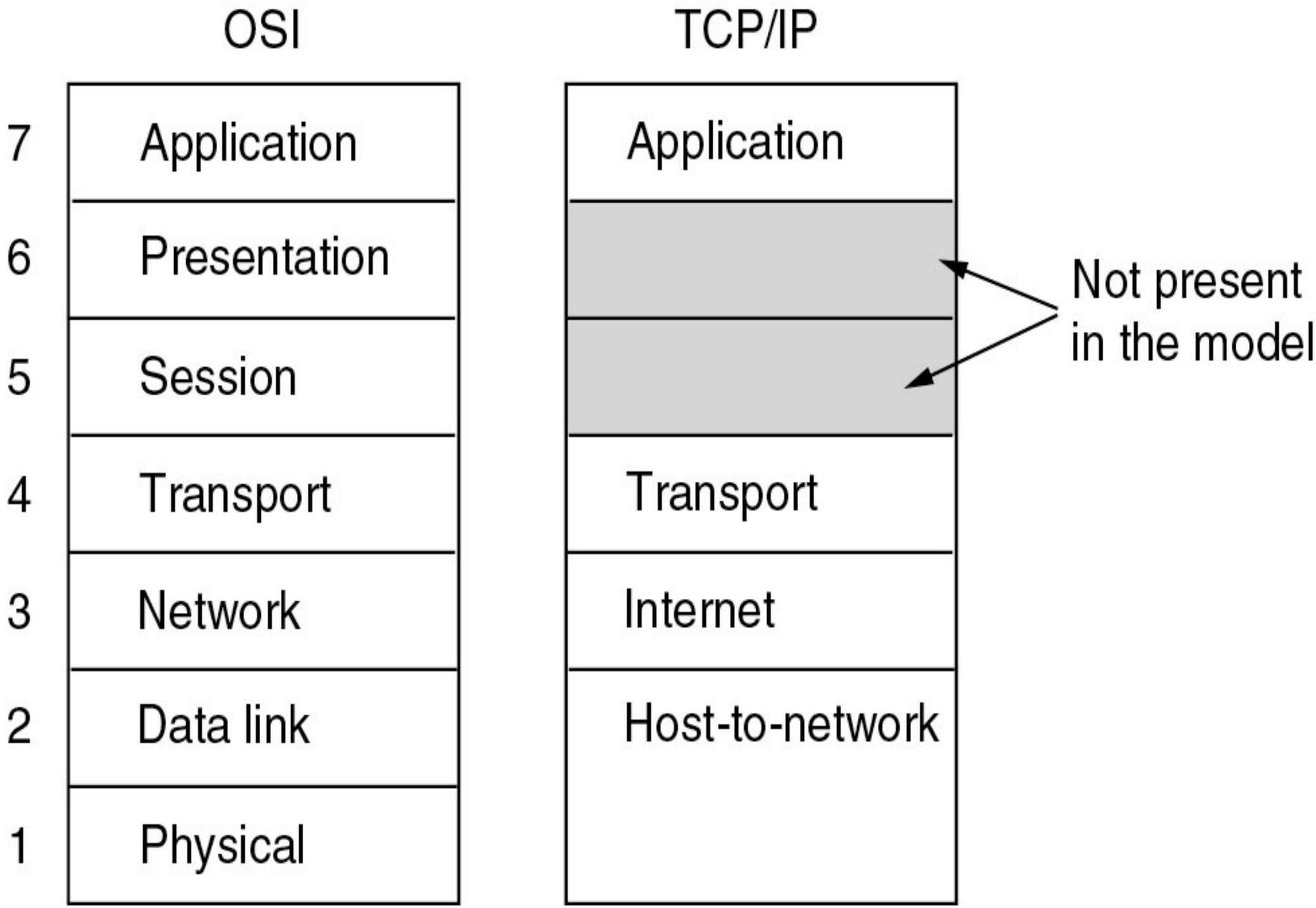
| Layer | Policies |
|---|---|
| Transport | • Retransmission policy<br>• Out-of-order caching policy<br>• Acknowledgement policy<br>• Flow control policy<br>• Timeout determination |
| Network | • Virtual circuits versus datagram inside the subnet<br>• Packet queueing and service policy<br>• Packet discard policy<br>• Routing algorithm<br>• Packet lifetime management |
| Data link | • Retransmission policy<br>• Out-of-order caching policy<br>• Acknowledgement policy<br>• Flow control policy |

# Die Schichtung des Internets - TCP/IP-Layer

| | | |
|---|---|---|
| Anwendung | Application | Telnet, FTP, HTTP, SMTP (E-Mail), ... |
| Transport | Transport | TCP (Transmission Control Protocol)<br>UDP (User Datagram Protocol) |
| Vermittlung | Network | IP (Internet Protocol)<br>+ ICMP (Internet Control Message Protocol)<br>+ IGMP (Internet Group Management Protocol) |
| Verbindung | Host-to-network | LAN (z.B. Ethernet, Token Ring etc.) |

CoNe Freiburg

# OSI versus TCP/IP






UNI FREIBURG

CoNe Freiburg

# Warum eine Vermittlungsschicht

- Lokale Netzwerke können nicht nur über Hubs, Switches oder Bridges verknüpft werden
  - Hubs: Kollisionen nehmen überhand
  - Switches:
    - Routen-Information durch Beobachtung der Daten ineffizient
    - Broadcast aller Nachrichten schafft Probleme
  - Es gibt über 100 Mio. lokale Netzwerke im Internet...
- Zur Beförderung von Paketen in großen Netzwerken braucht man Routeninformationen
  - Wie baut man diese auf?
  - Wie leitet man Pakete weiter?
- Das Internet-Protokoll ist im wesentlich ein Vermittlungsschichtprotokoll

CoNe Freiburg

# Routing-Tabelle und Paket-Weiterleitung

- IP-Routing-Tabelle
  - enthält für Ziel (Destination) die Adresse des nächsten Rechners (Gateway)
  - Destination kann einen Rechner oder ganze Sub-nets beschreiben
  - Zusätzlich wird ein Default-Gateway angegeben
- Packet Forwarding
  - früher Packet Routing genannt
  - IP-Paket (datagram) enthält Start-IP-Adresse und Ziel-IP-Adresse
    - Ist Ziel-IP-Adresse = eigene Rechneradresse dann Nachricht ausgeliefert
    - Ist Ziel-IP-Adresse in Routing-Tabelle dann leite Paket zum angegeben Gateway
    - Ist Ziel-IP-Subnetz in Routing-Tabelle dann leite Paket zum angegeben Gateway
    - Ansonsten leite zum Default-Gateway

CoNe Freiburg

# Paket-Weiterleitung im Internet Protokoll

- IP-Paket (datagram) enthält unter anderen
  - TTL (Time-to-Live): Anzahl der Hops
  - Start-IP-Adresse
  - Ziel-IP-Adresse
- Behandlung eines Pakets
  - Verringere TTL (Time to Live) um 1
  - Falls TTL ≠ 0  dann Packet-Forwarding aufgrund der Routing-Tabelle
  - Falls TTL = 0 oder bei Problemen in Packet-Forwarding:
    - Lösche Paket
    - Falls Paket ist kein ICMP-Paket dann
      - Sende ICMP-Paket mit
        - Start= aktuelle IP-Adresse und
        - Ziel = alte Start-IP-Adresse

CoNe Freiburg

# Statisches und Dynamisches Routing

- Forwarding:
  - Weiterleiten von Paketen
- Routing:
  - Erstellen Routen, d.h.
    - Erstellen der Routing-Tabelle
- Statisches Routing
  - Tabelle wird manuell erstellt
  - sinnvoll für kleine und stabile LANs
- Dynamisches Routing
  - Tabellen werden durch Routing-Algorithmus erstellt
  - Zentraler Algorithmus, z.B. Link State
    - Einer/jeder kennt alle Information, muss diese erfahren
  - Dezentraler Algorithmus, z.B. Distance Vector
    - arbeitet lokal in jedem Router
    - verbreitet lokale Information im Netzwerk

CoNe Freiburg

# Das Kürzeste-Wege-Problem

- Gegeben:
  - Ein gerichteter Graph G=(V,E)
  - Startknoten
  - mit Kantengewichtungen $w : E \rightarrow \mathbb{R}$
- Definiere Gewicht des kürzesten Pfades
  - δ(u,v) = minimales Gewicht w(p) eines Pfades p von u nach v
  - w(p) = Summe aller Kantengewichte w(e) der Kanten e des Pfades
- Gesucht:
  - Die kürzesten Wege vom Startknoten s zu allen Knoten in G
    - also jeweils ein Pfad mit dem geringsten Gewicht zu jedem anderen Knoten
- Lösungsmenge:
  - wird beschrieben durch einen Baum mit Wurzel s
  - Jeder Knoten zeigt in Richtung der Wurzel

CoNe Freiburg

# Kürzeste Wege mit Edsger Wybe Dijkstra

- Dijkstras Kürzeste-Wege-Algorithmus kann mit Laufzeit Θ(|E| + |V| log |V|) implementiert werden.

**Dijkstra**$(G, w, s)$
begin
    **Init-Single-Source**$(G, w)$
    $S \leftarrow \emptyset$
    $Q \leftarrow V$
    while $Q \neq \emptyset$ do
        $u \leftarrow$ Element aus $Q$ mit minimalen Wert $d(u)$
        $S \leftarrow S \cup \{u\}$
        $Q \leftarrow Q \setminus \{u\}$
        for all $v \in$ Adj$(u)$ do
            **Relax**$(u, v)$
        od
    od
end

**Init-Single-Source**$(G, w, s)$
begin
    for all $v \in V$ do
        $d(v) \leftarrow \infty$
        $\pi(v) \leftarrow v$
    od
    $d(s) \leftarrow 0$
end

**Relax**$(u, v)$
begin
    if $d(v) > d(u) + w(u, v)$ then
        $d(v) \leftarrow d(u) + w(u, v)$
        $\pi(v) \leftarrow u$
    fi
end

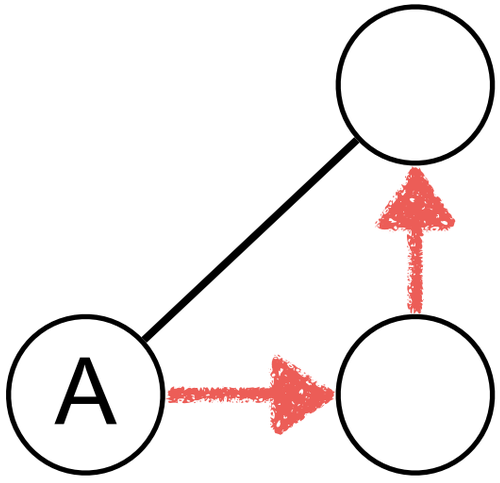
CoNe Freiburg

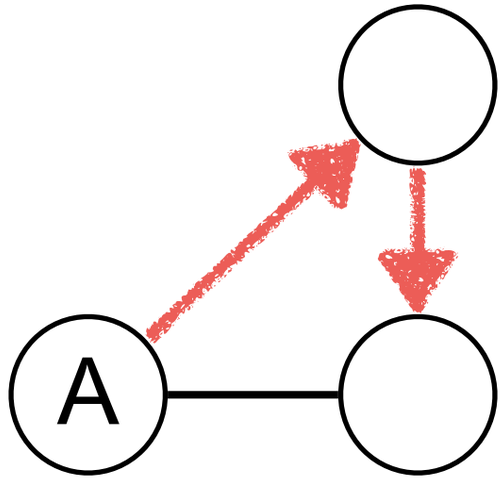
# Brute-Force-Ansatz

Im schlechtesten Fall:

- alle von einem Punkt ausgehenden Wege müssen bestimmt werden
- alle Knoten sind mit allen anderen Knoten verbunden

Für drei Knoten und Startpunkt A: **2 mögliche Wege**

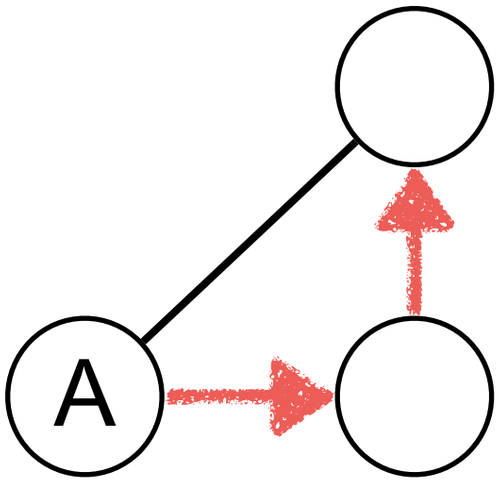


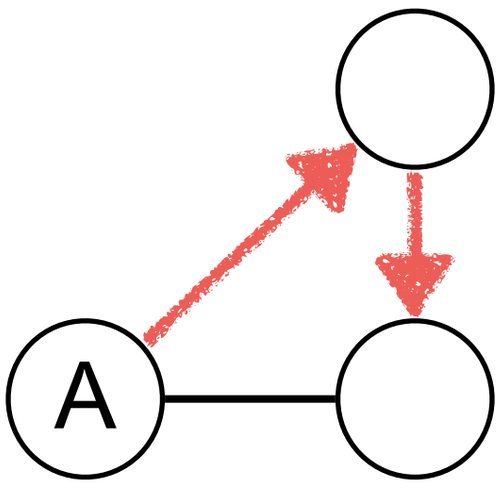




UNI FREIBURG

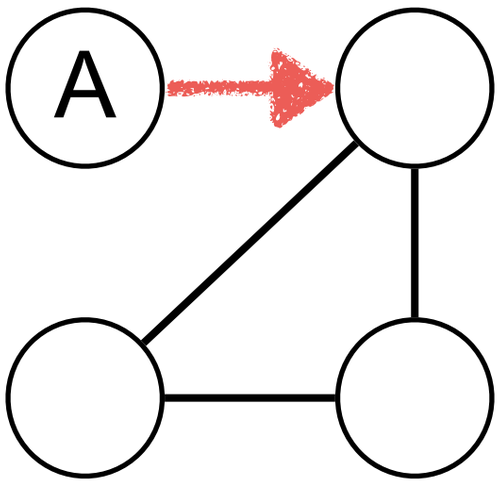
# Brute-Force-Ansatz

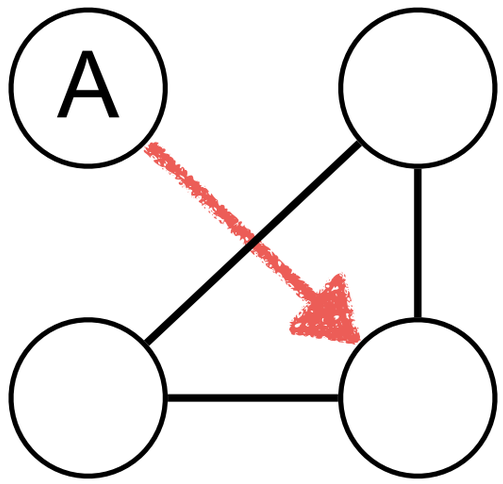
Für drei Knoten und Startpunkt A: **2 mögliche Wege**

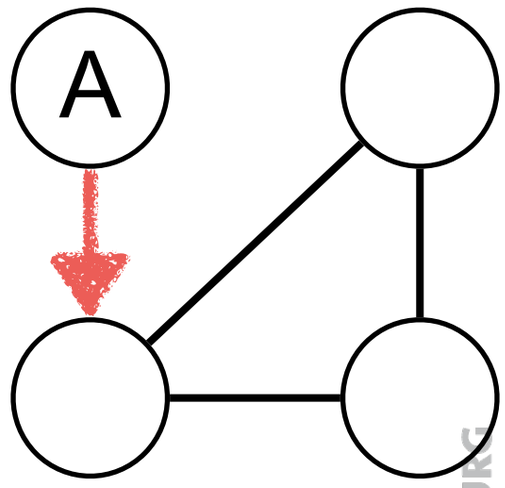



Für vier Knoten (ein Knoten A + ein Graph mit 3 Knoten ):
**3 x 2 mögliche Wege**



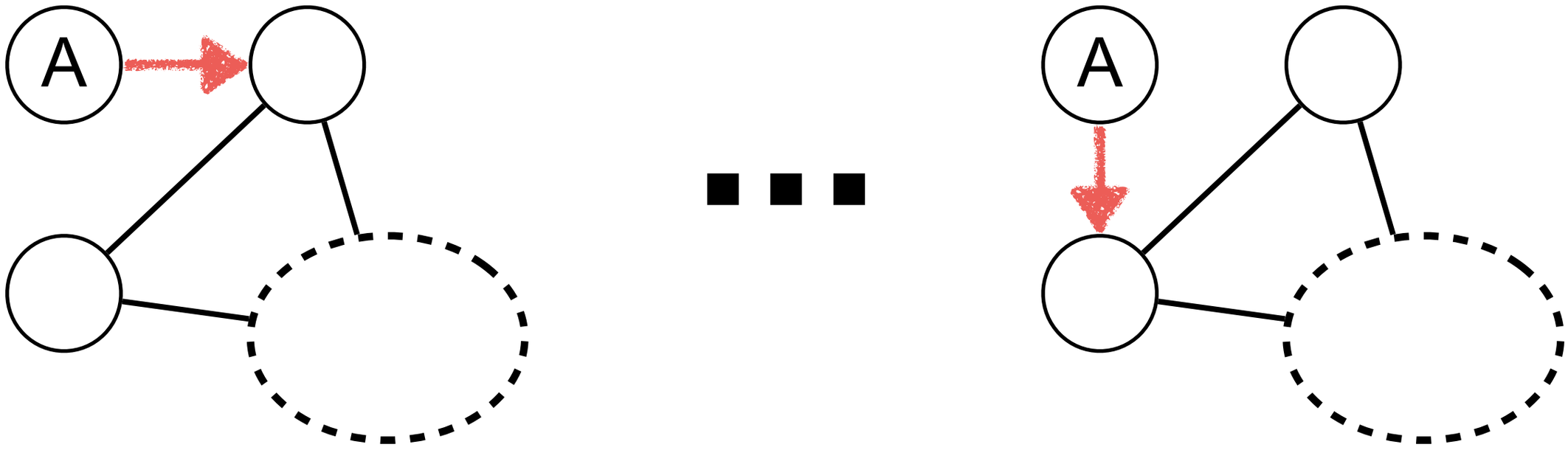



CoNe Freiburg

# Brute-Force-Ansatz

Für *n* Knoten [ein Knoten A + ein Graph mit *(n-1)* Knoten ]:

**(n-1)! mögliche Wege**




Für jeden der Wege: n-1 Streckenabschnitte

=> **(n-1)(n-1)! Berechnungen**

10: 3265920
20: 2311256907767808000
50: 298058113376791104827403560027434734674900887375813017600000000000

# Der Algorithmus von Dijkstra

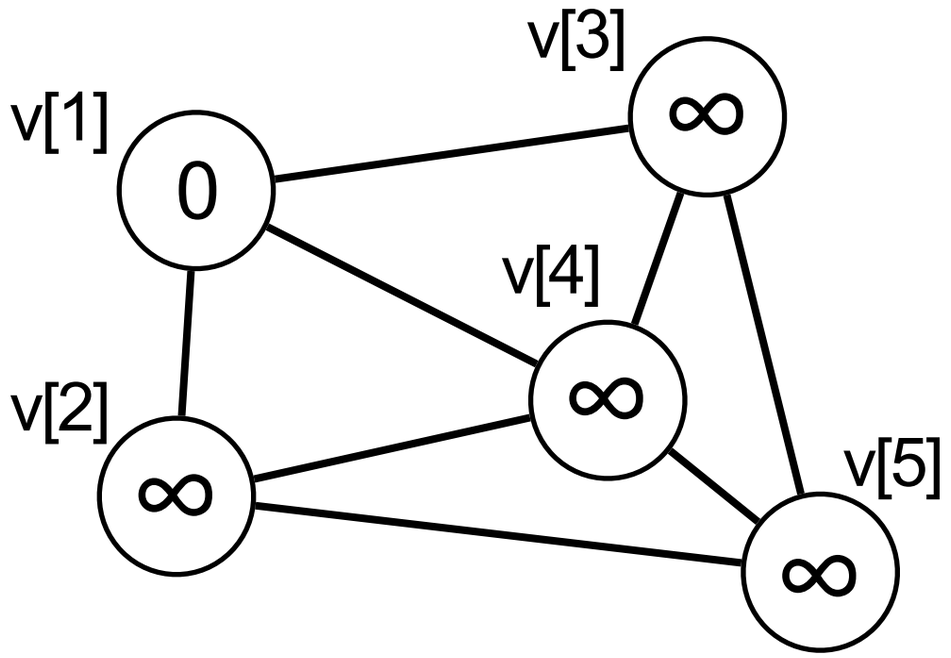
Knoten v[i] hat die Eigenschaften:

- „Distanz“ zu v[1]: **v[i].d**
- „Vorgänger“: **v[i].p**

Warteschlange für noch nicht besuchte Knoten: **queue**

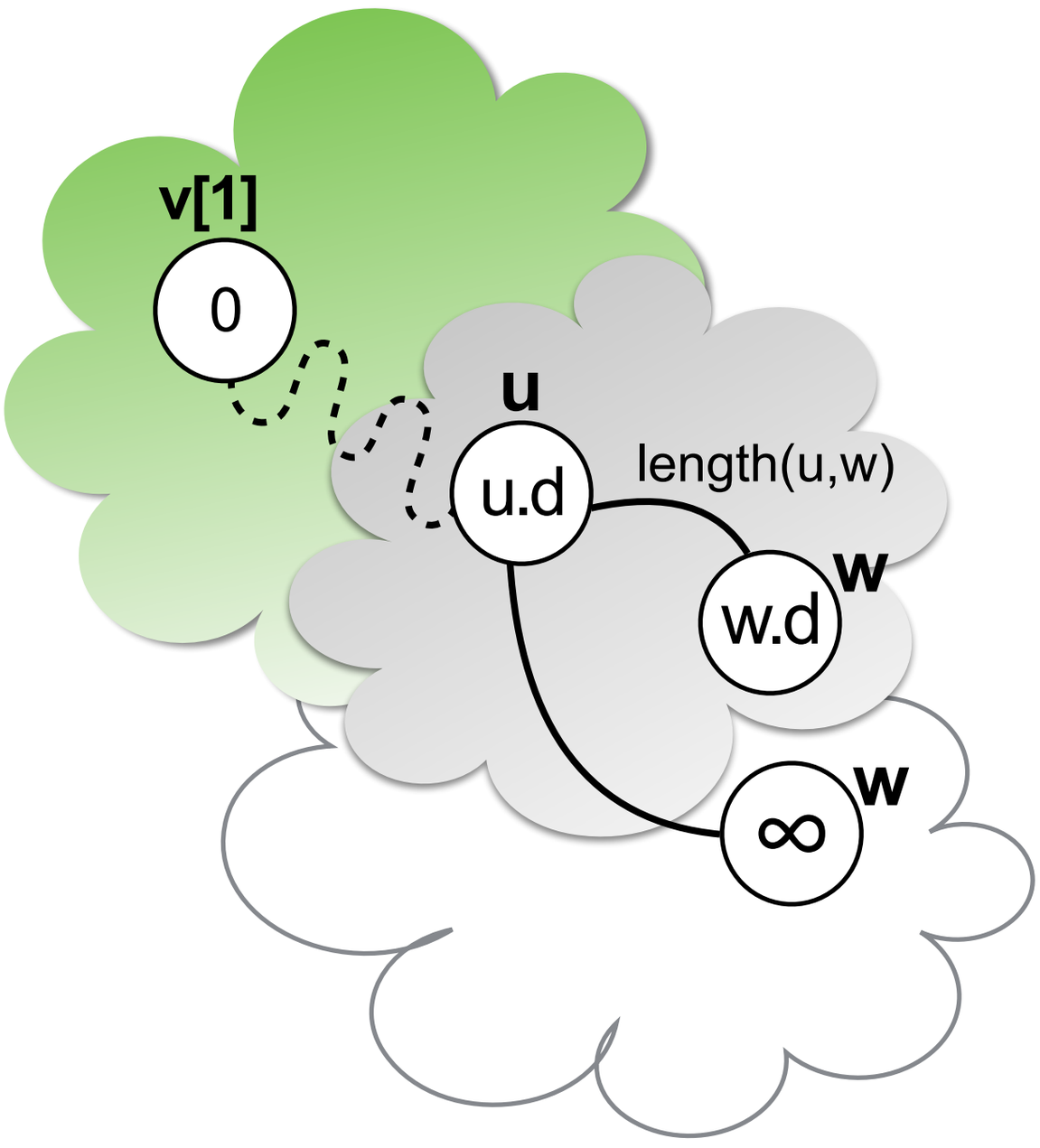
Initialisieren:

```
v[1].d = 0;
for ( i = 2..n )
{
    v[i].d =∞;
    v[i].p = NULL;
}
queue = { v[1] };
```



# Der Algorithmus von Dijkstra

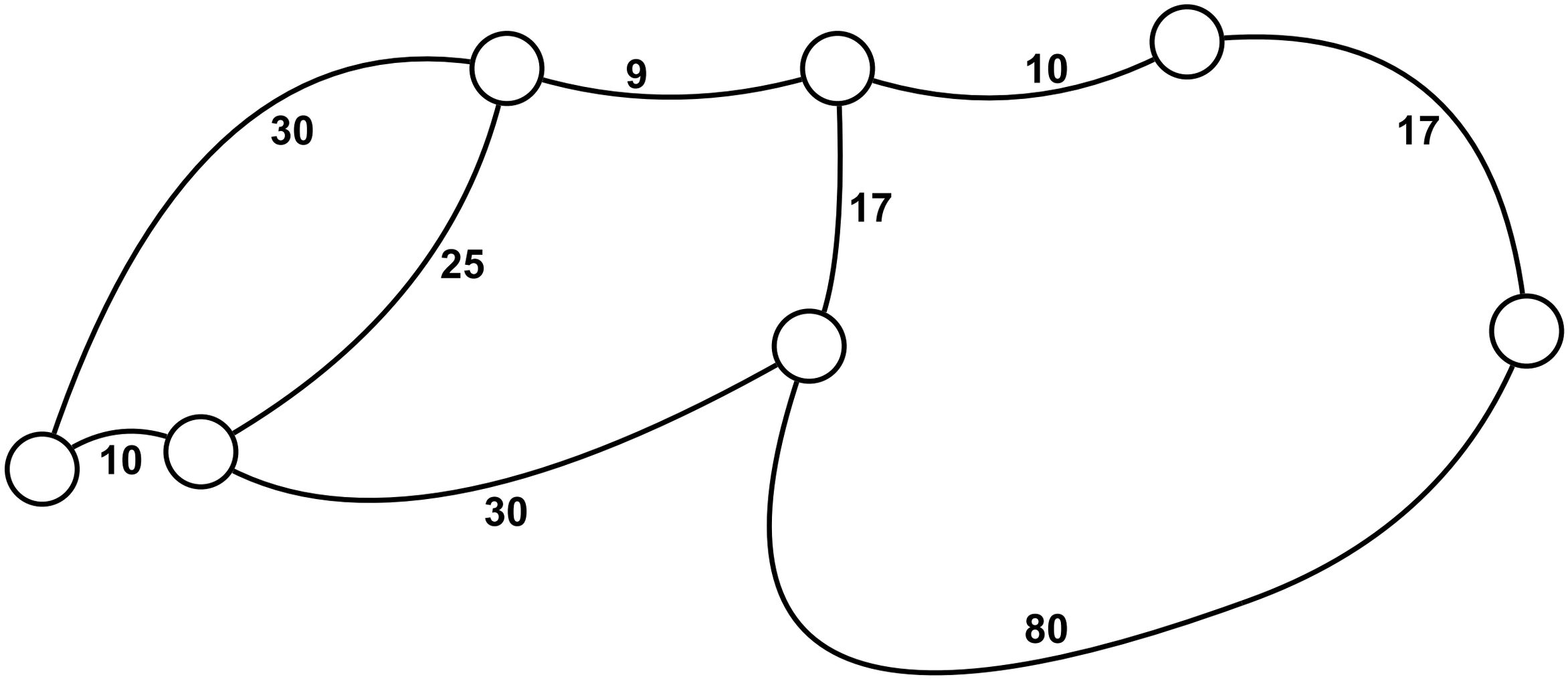
"Relax":




```
while ( queue ≠ ∅ )
{
    u = queue.extractMin();
    for ( (u,w) in E )
    {
       dist = u.d + length(u,w);
       if ( w ∈ queue  and  w.d > dist )
       {
           w.d = dist;  w.p = (u,w);
       }
       else if (w.p == NULL) {
           w.d = dist;  w.p = (u,w);
           queue.insert(w,dist);
       }
}
```

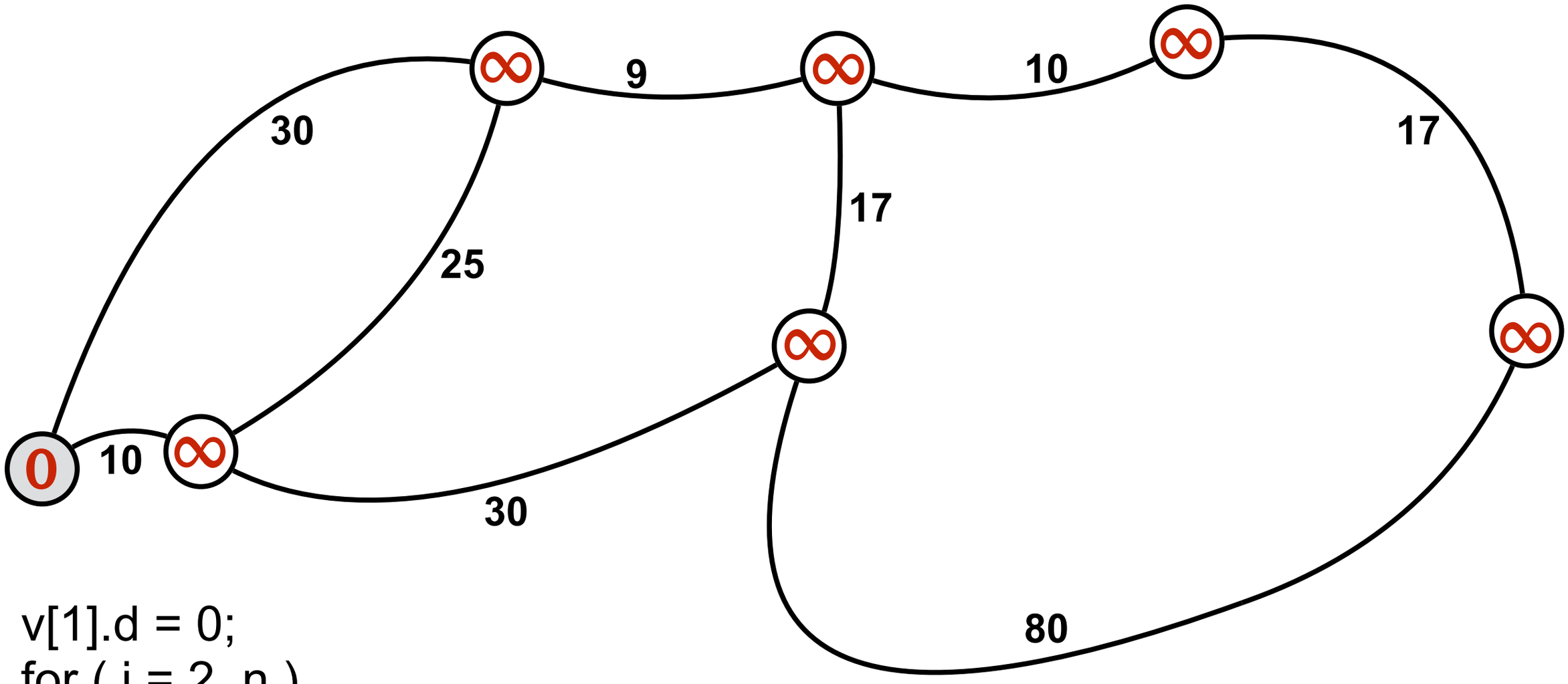
CoNe Freiburg

# Beispiel Dijkstra



CoNe Freiburg

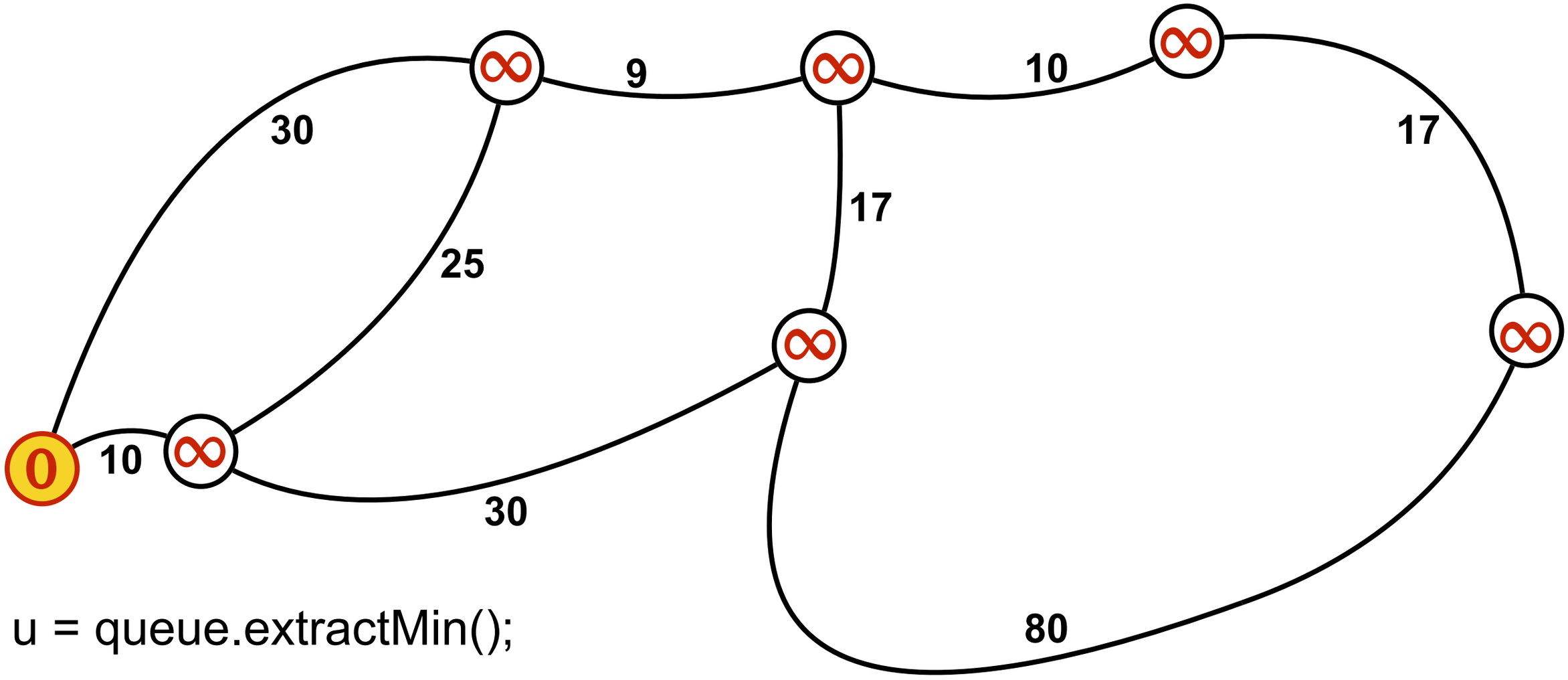
# Beispiel Dijkstra




```
v[1].d = 0;
for ( i = 2..n )
{
    v[i].d =∞;
    v[i].p = NULL;
}
queue = { v[1] };
```

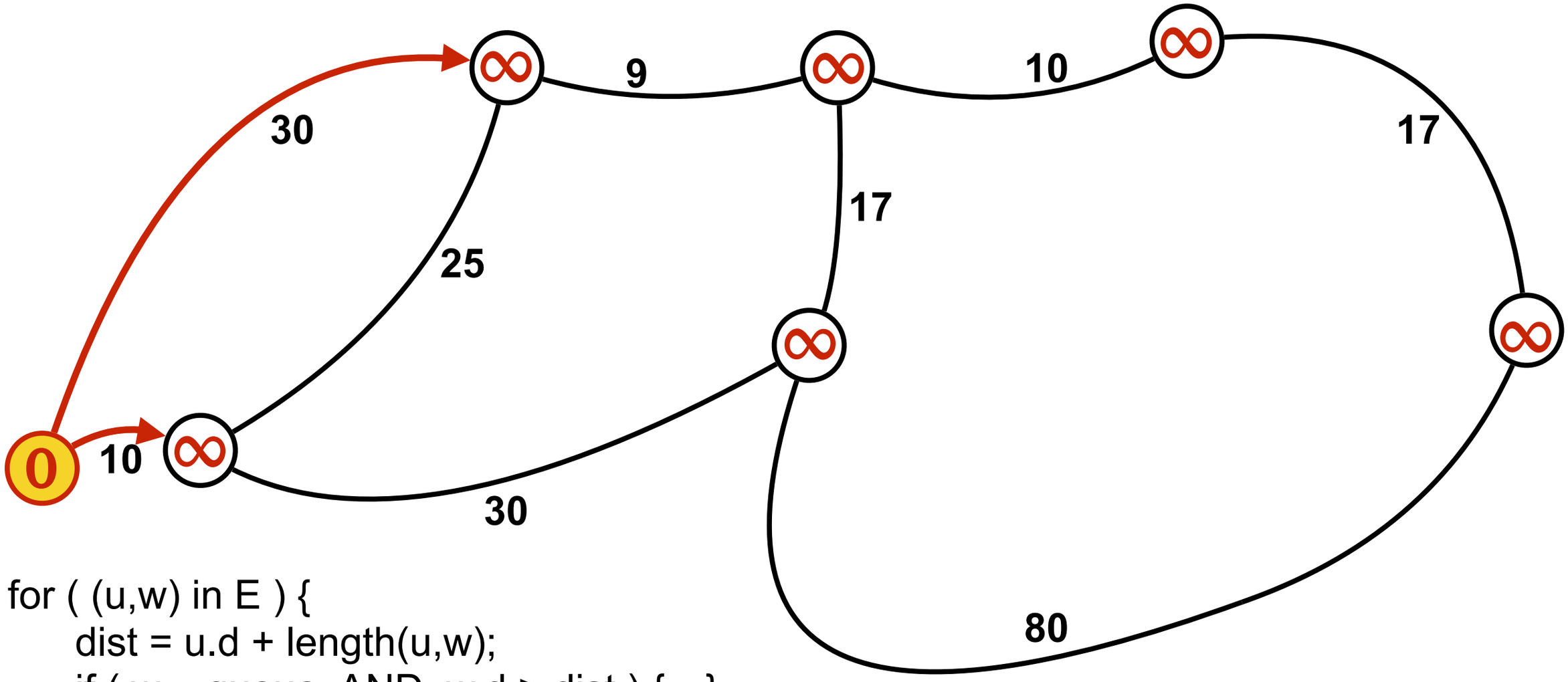
CoNe Freiburg

# Beispiel Dijkstra




u = queue.extractMin();

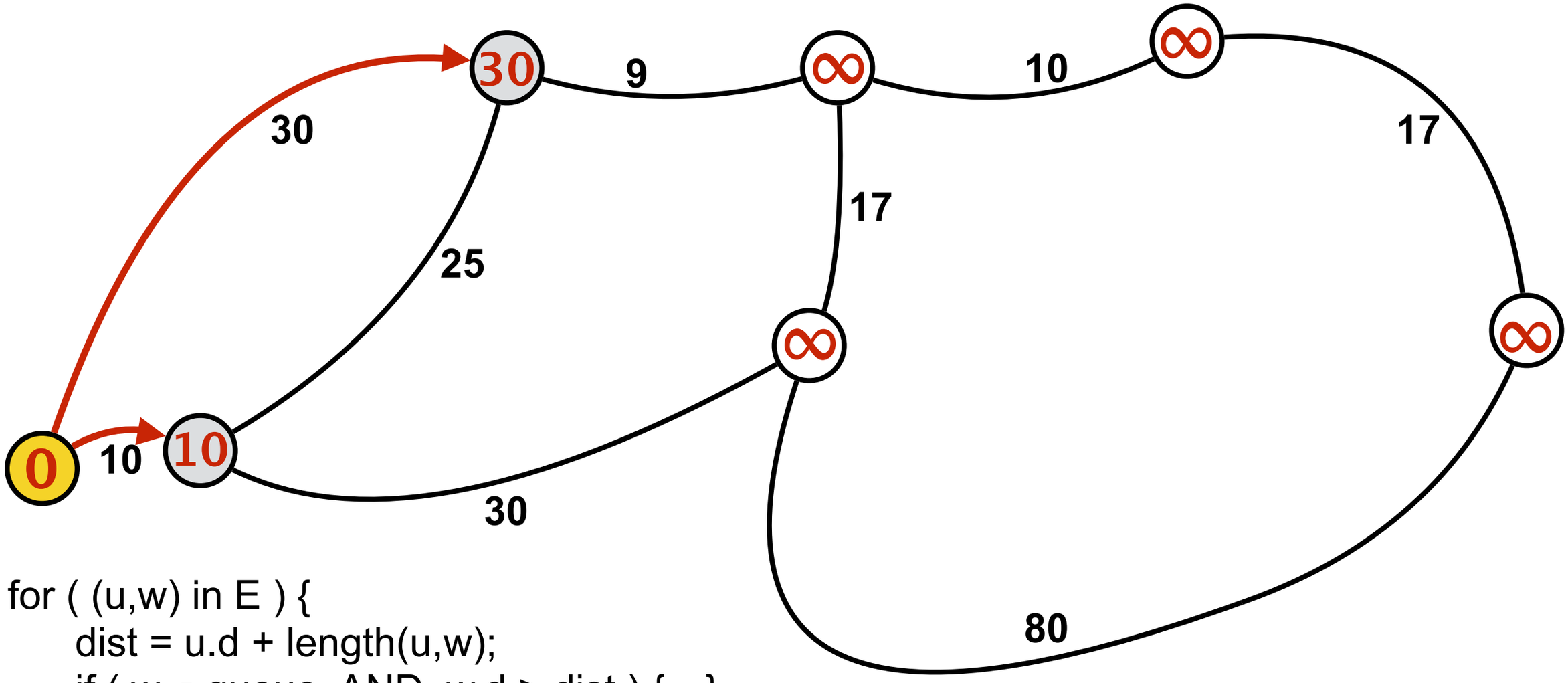
# Beispiel Dijkstra




```
for ( (u,w) in E ) {
      dist = u.d + length(u,w);
      if ( w ∈ queue  AND  w.d > dist ) {…}
      else if (w.p == NULL)
      {
        w.d = dist;
        w.p = (u,w);
        queue.insert(w,dist);
      }
}
```

CoNe Freiburg

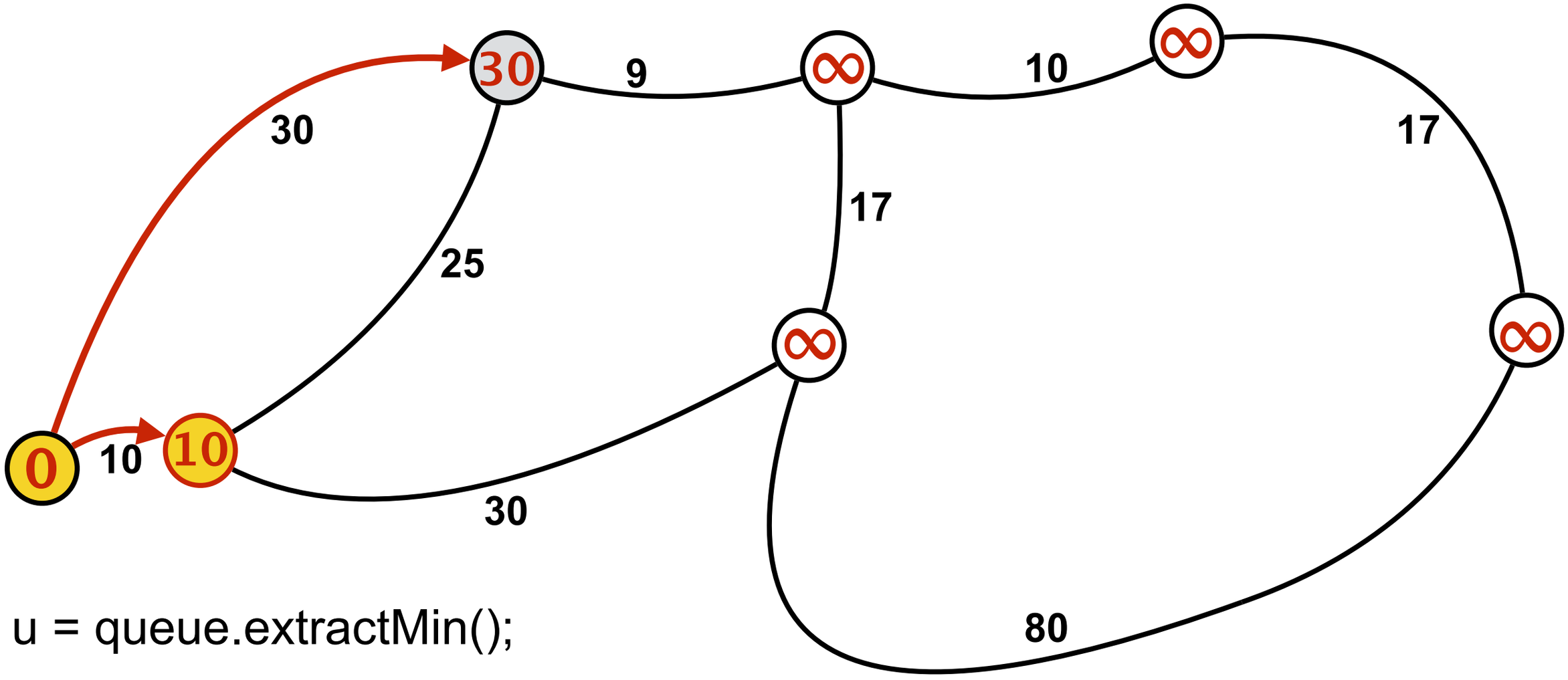
# Beispiel Dijkstra




```
for ( (u,w) in E ) {
      dist = u.d + length(u,w);
      if ( w ∈ queue  AND  w.d > dist ) {…}
      else if (w.p == NULL)
      {
        w.d = dist;
        w.p = (u,w);
        queue.insert(w,dist);
      }
}
```

CoNe Freiburg

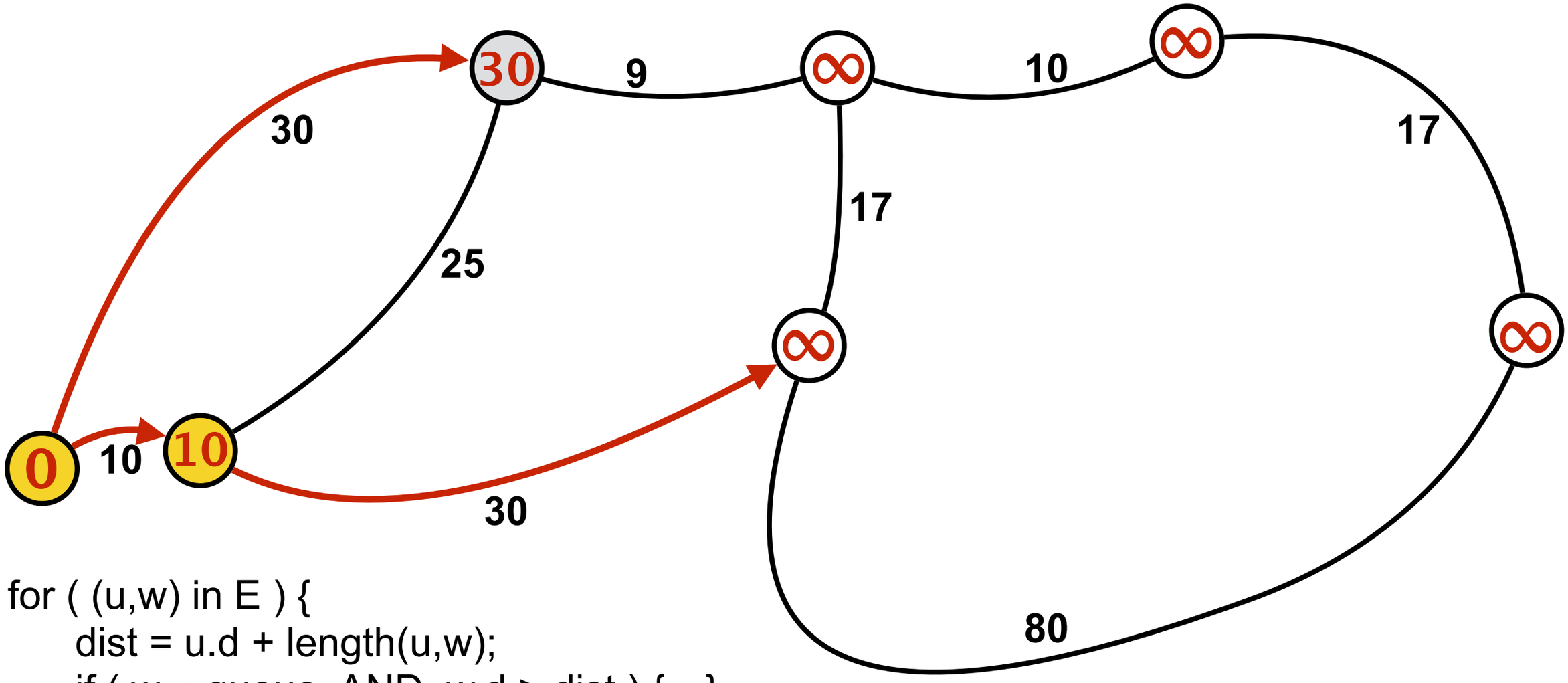
# Beispiel Dijkstra




u = queue.extractMin();

CoNe Freiburg

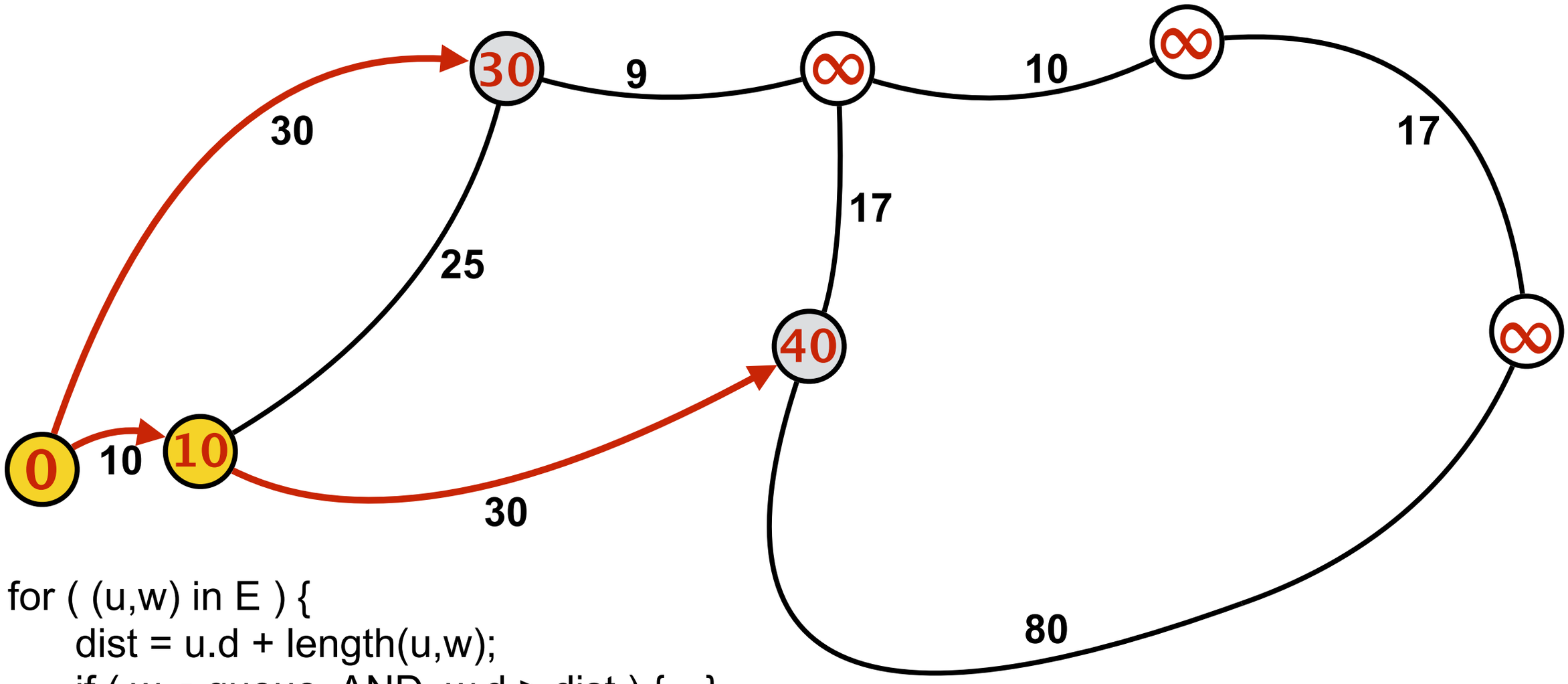
# Beispiel Dijkstra




```
for ( (u,w) in E ) {
      dist = u.d + length(u,w);
      if ( w ∈ queue  AND  w.d > dist ) {…}
      else if (w.p == NULL)
      {
        w.d = dist;
        w.p = (u,w);
        queue.insert(w,dist);
      }
}
```

CoNe Freiburg

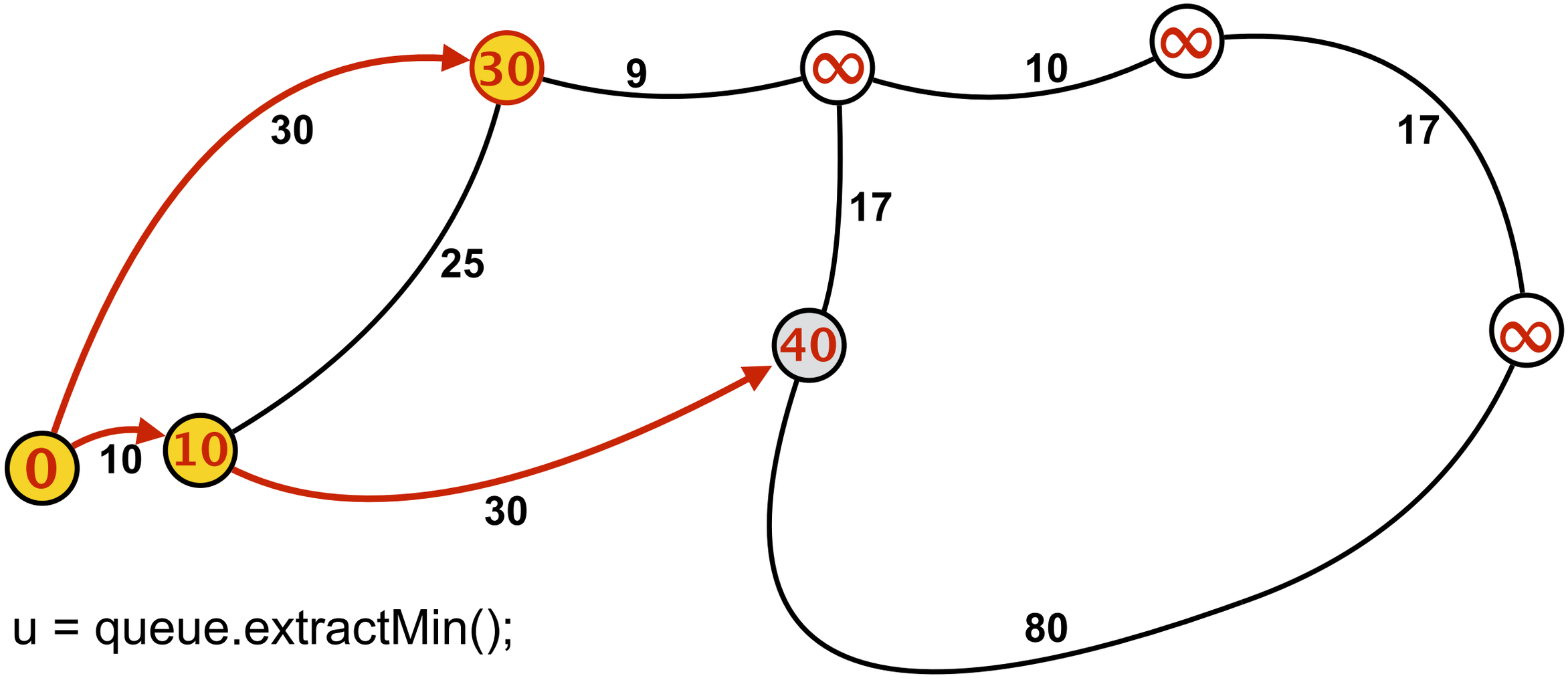
# Beispiel Dijkstra




```
for ( (u,w) in E ) {
      dist = u.d + length(u,w);
      if ( w ∈ queue  AND  w.d > dist ) {…}
      else if (w.p == NULL)
      {
        w.d = dist;
        w.p = (u,w);
        queue.insert(w,dist);
      }
}
```

CoNe Freiburg

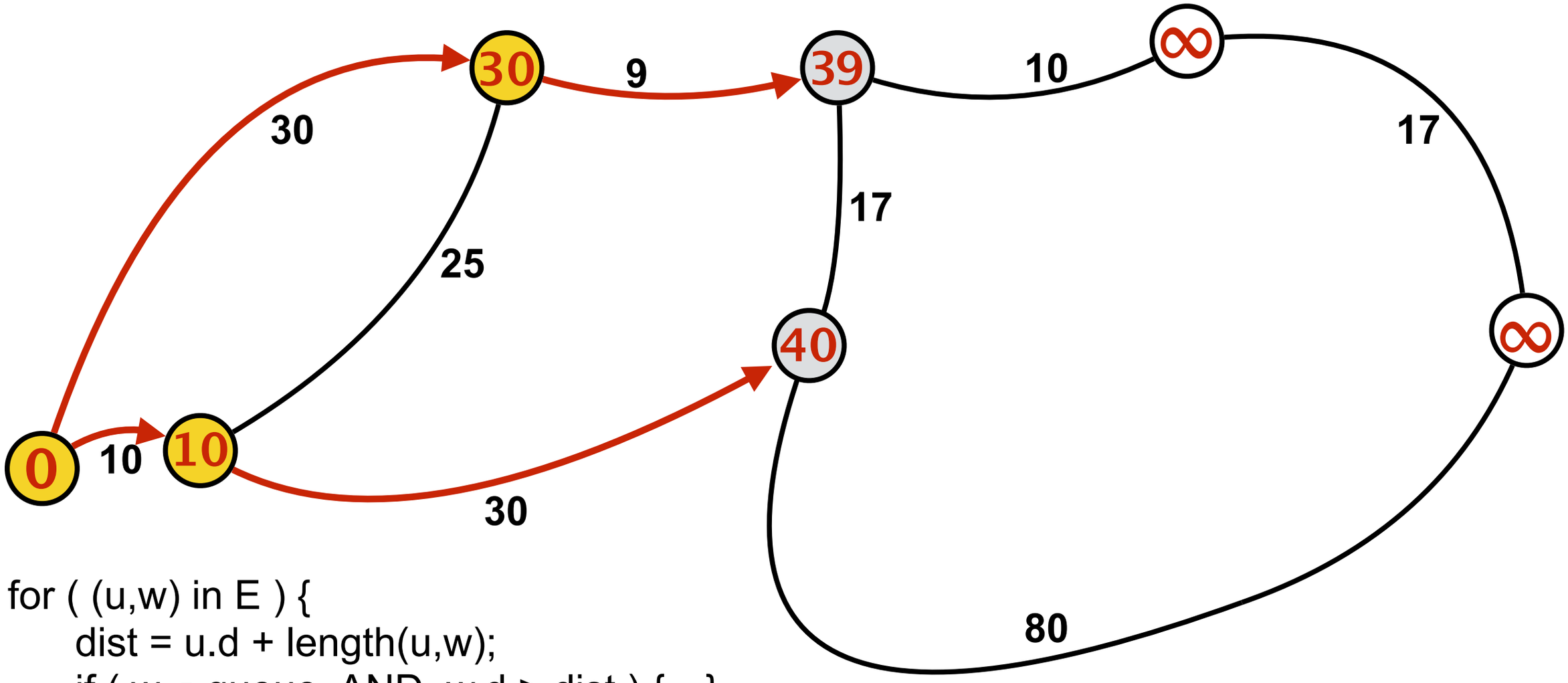
# Beispiel Dijkstra




u = queue.extractMin();

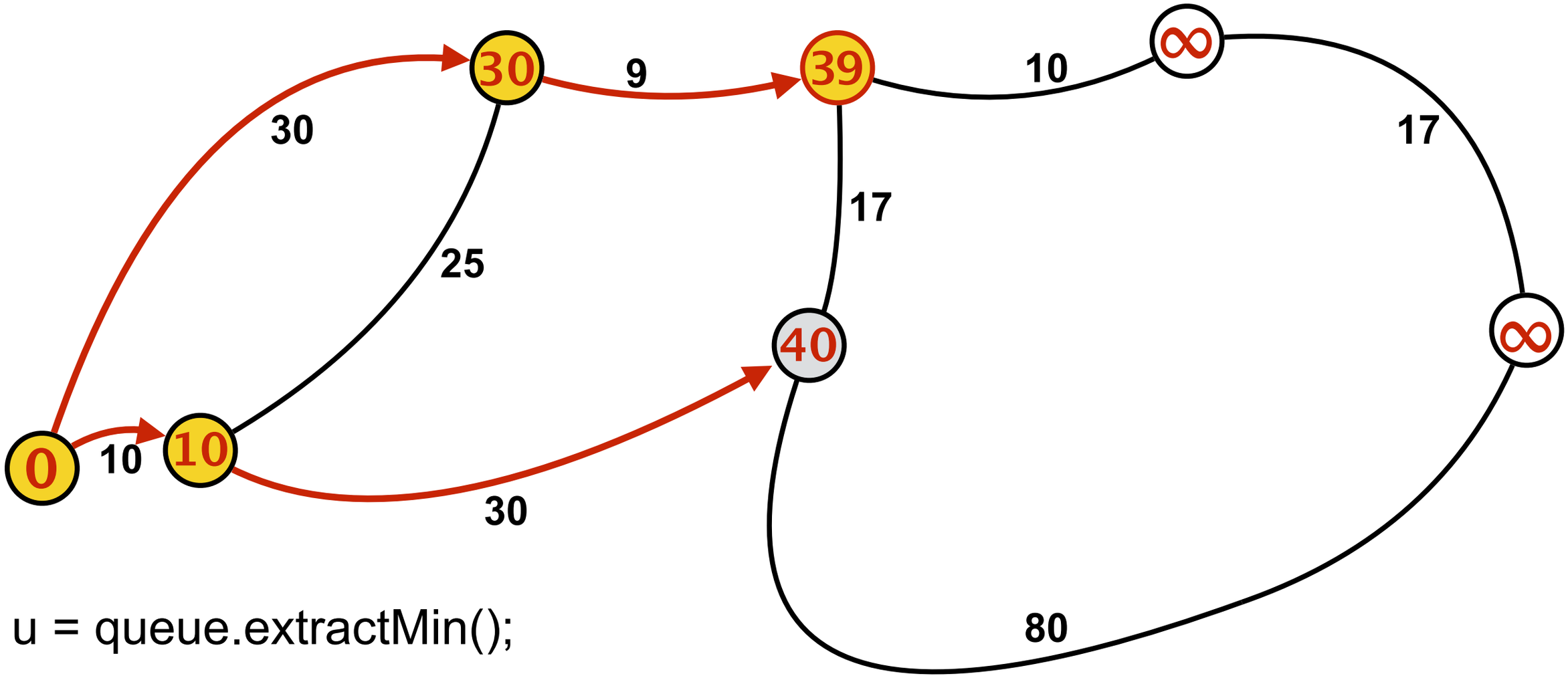
CoNe Freiburg

# Beispiel Dijkstra




```
for ( (u,w) in E ) {
      dist = u.d + length(u,w);
      if ( w ∈ queue  AND  w.d > dist ) {…}
      else if (w.p == NULL)
      {
        w.d = dist;
        w.p = (u,w);
        queue.insert(w,dist);
      }
}
```

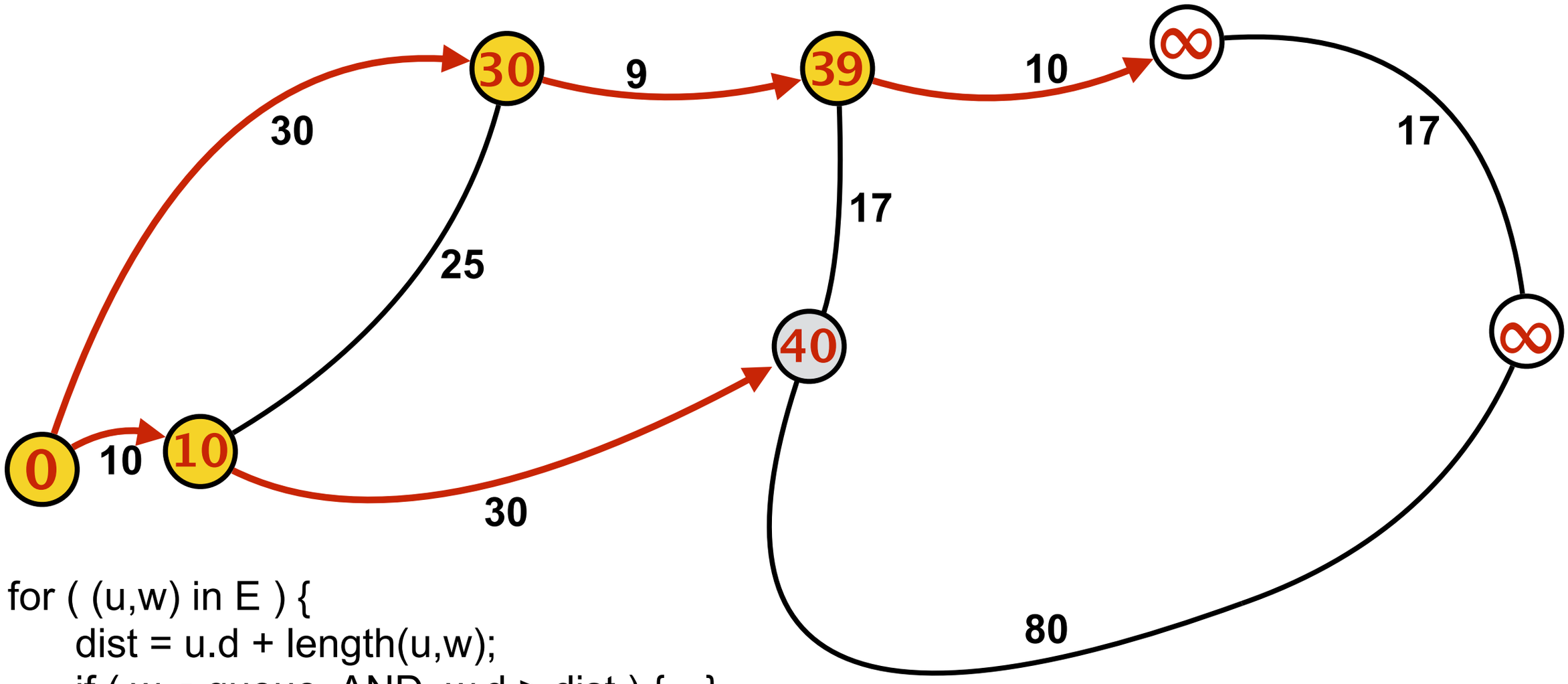
# Beispiel Dijkstra




u = queue.extractMin();

CoNe Freiburg

# Beispiel Dijkstra



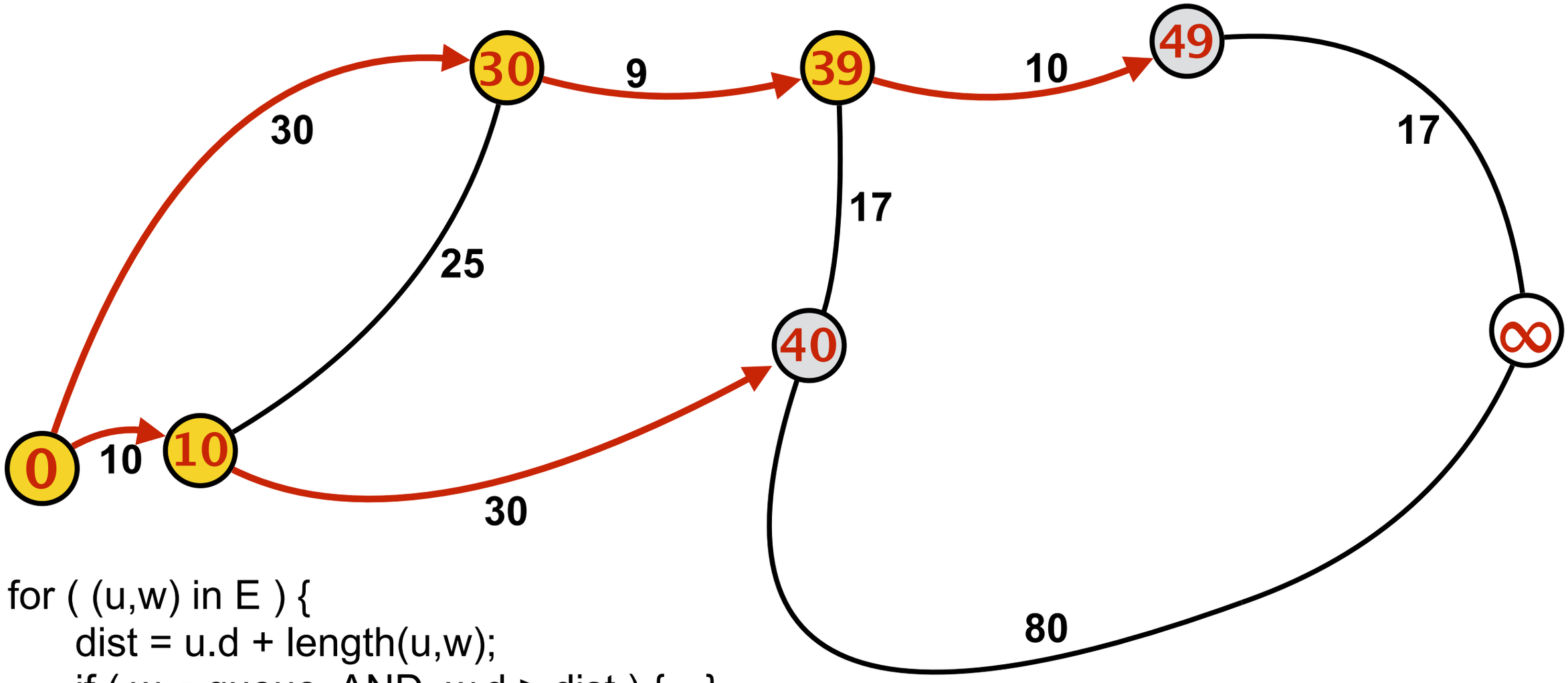

```
for ( (u,w) in E ) {
      dist = u.d + length(u,w);
      if ( w ∈ queue  AND  w.d > dist ) {…}
      else if (w.p == NULL)
      {
        w.d = dist;
        w.p = (u,w);
        queue.insert(w,dist);
      }
}
```



UNI FREIBURG

CoNe Freiburg

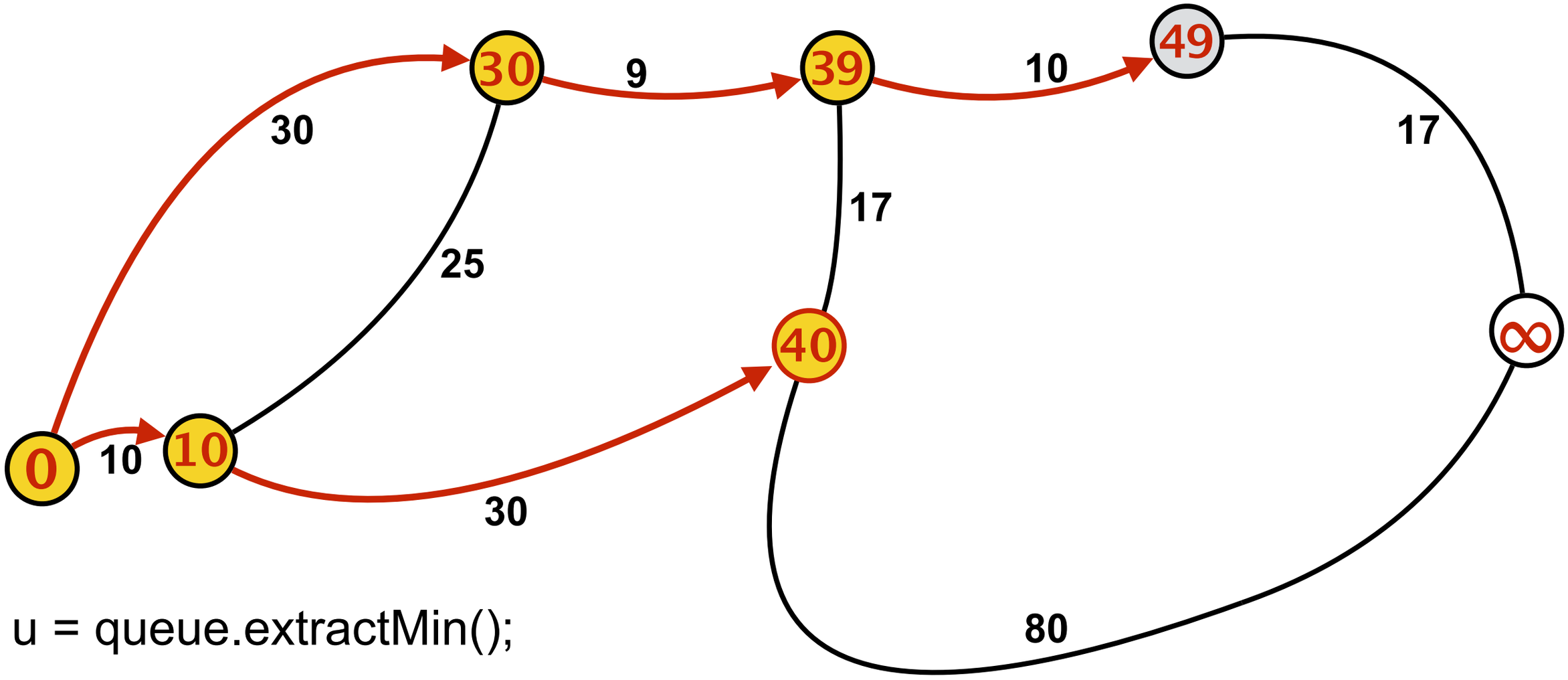
# Beispiel Dijkstra




```
for ( (u,w) in E ) {
      dist = u.d + length(u,w);
      if ( w ∈ queue  AND  w.d > dist ) {…}
      else if (w.p == NULL)
      {
        w.d = dist;
        w.p = (u,w);
        queue.insert(w,dist);
      }
}
```

CoNe Freiburg

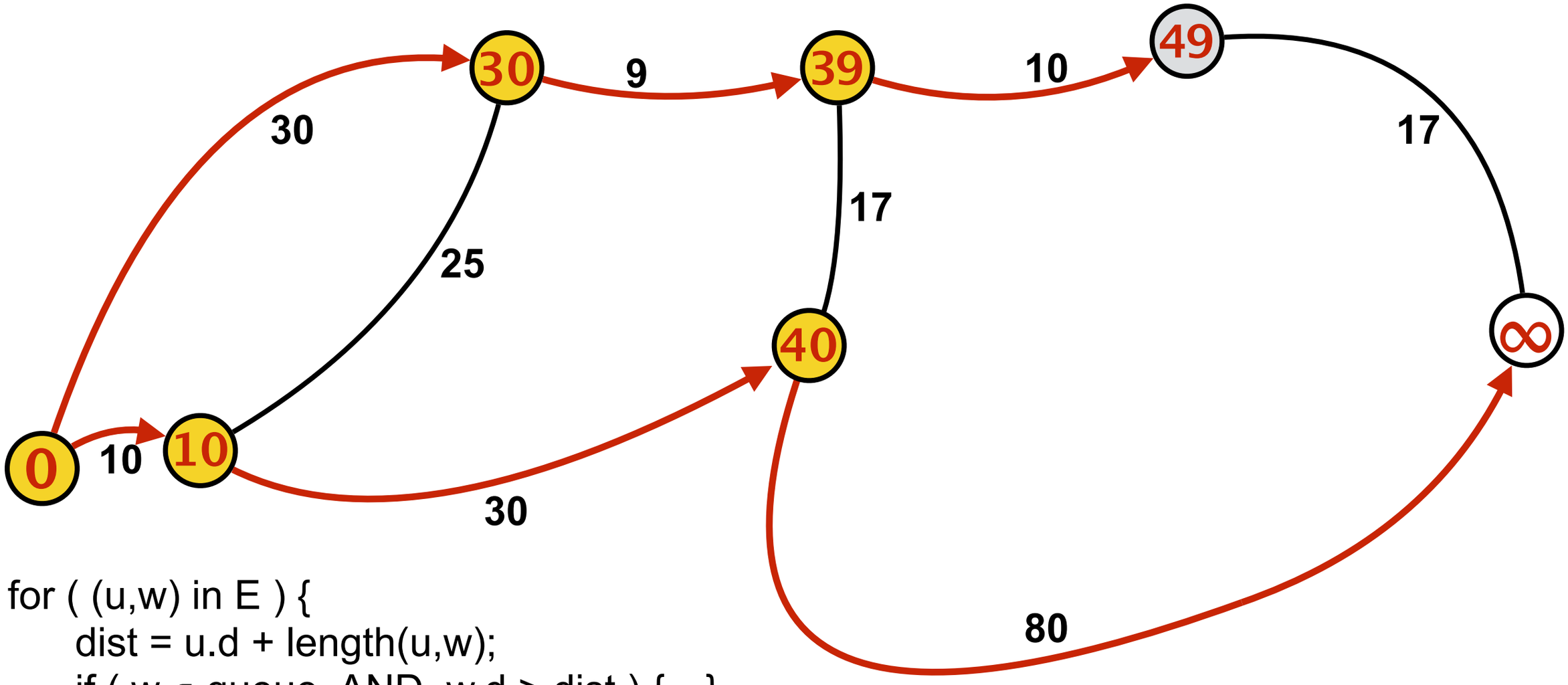
# Beispiel Dijkstra




u = queue.extractMin();

CoNe Freiburg

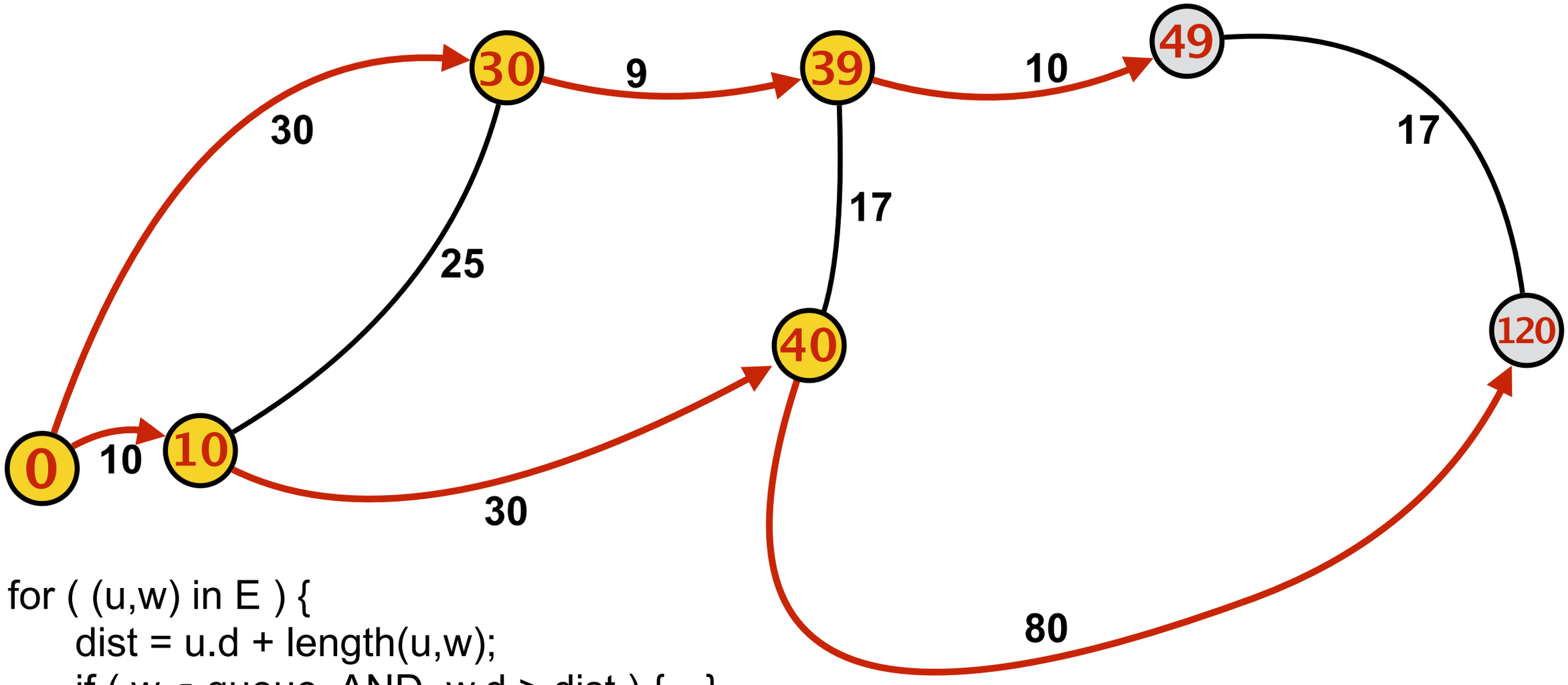
# Beispiel Dijkstra




```
for ( (u,w) in E ) {
      dist = u.d + length(u,w);
      if ( w ∈ queue  AND  w.d > dist ) {…}
      else if (w.p == NULL)
      {
        w.d = dist;
        w.p = (u,w);
        queue.insert(w,dist);
      }
}
```

CoNe Freiburg

# Beispiel Dijkstra



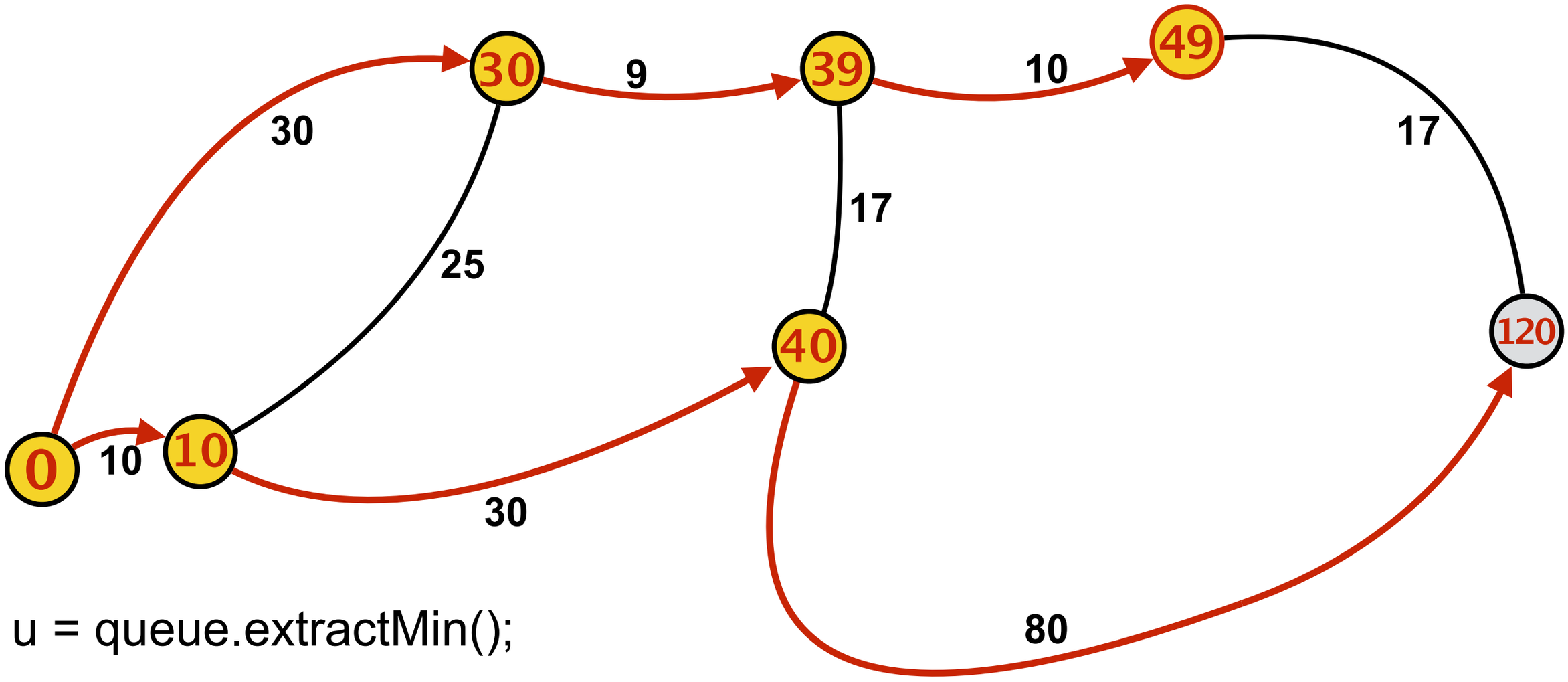

```
for ( (u,w) in E ) {
      dist = u.d + length(u,w);
      if ( w ∈ queue  AND  w.d > dist ) {…}
      else if (w.p == NULL)
      {
        w.d = dist;
        w.p = (u,w);
        queue.insert(w,dist);
      }
}
```



UNI FREIBURG

CoNe Freiburg

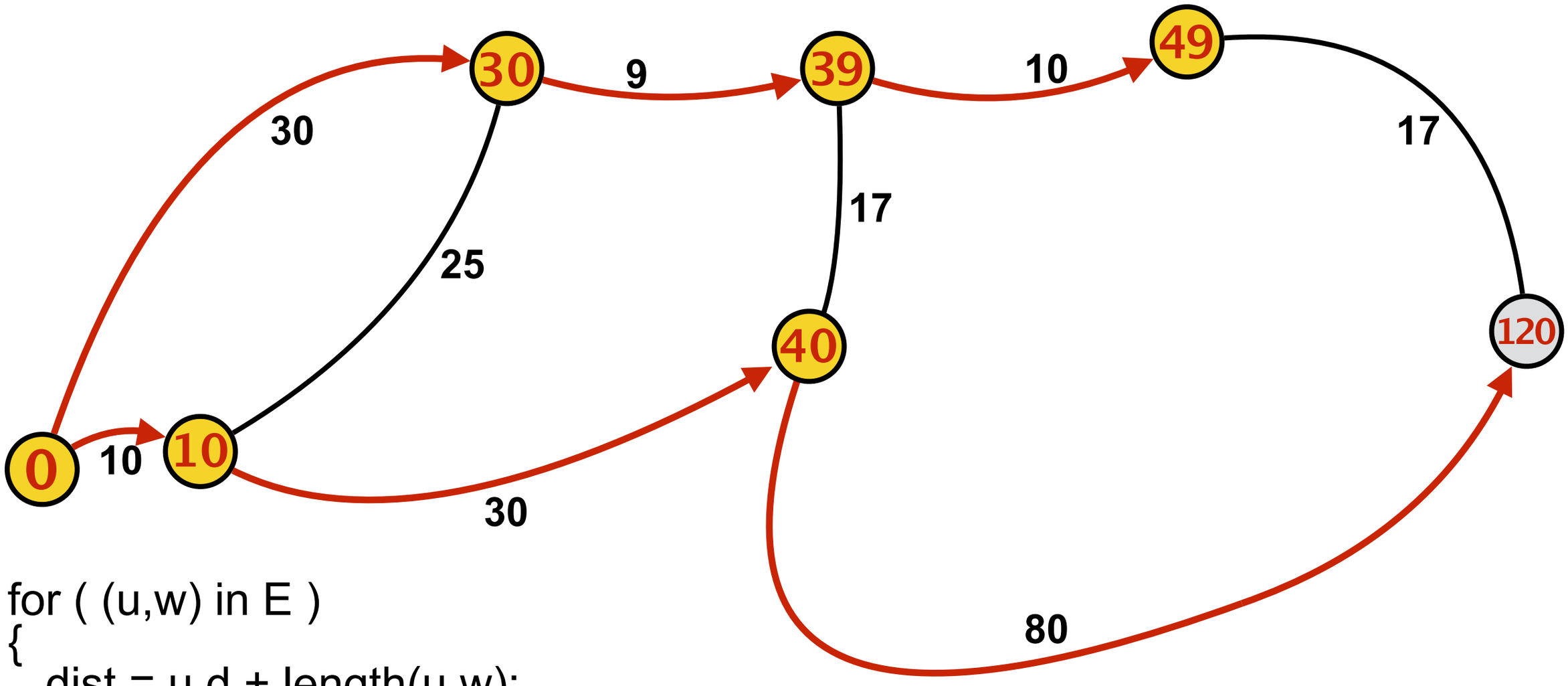
# Beispiel Dijkstra




u = queue.extractMin();

CoNe Freiburg

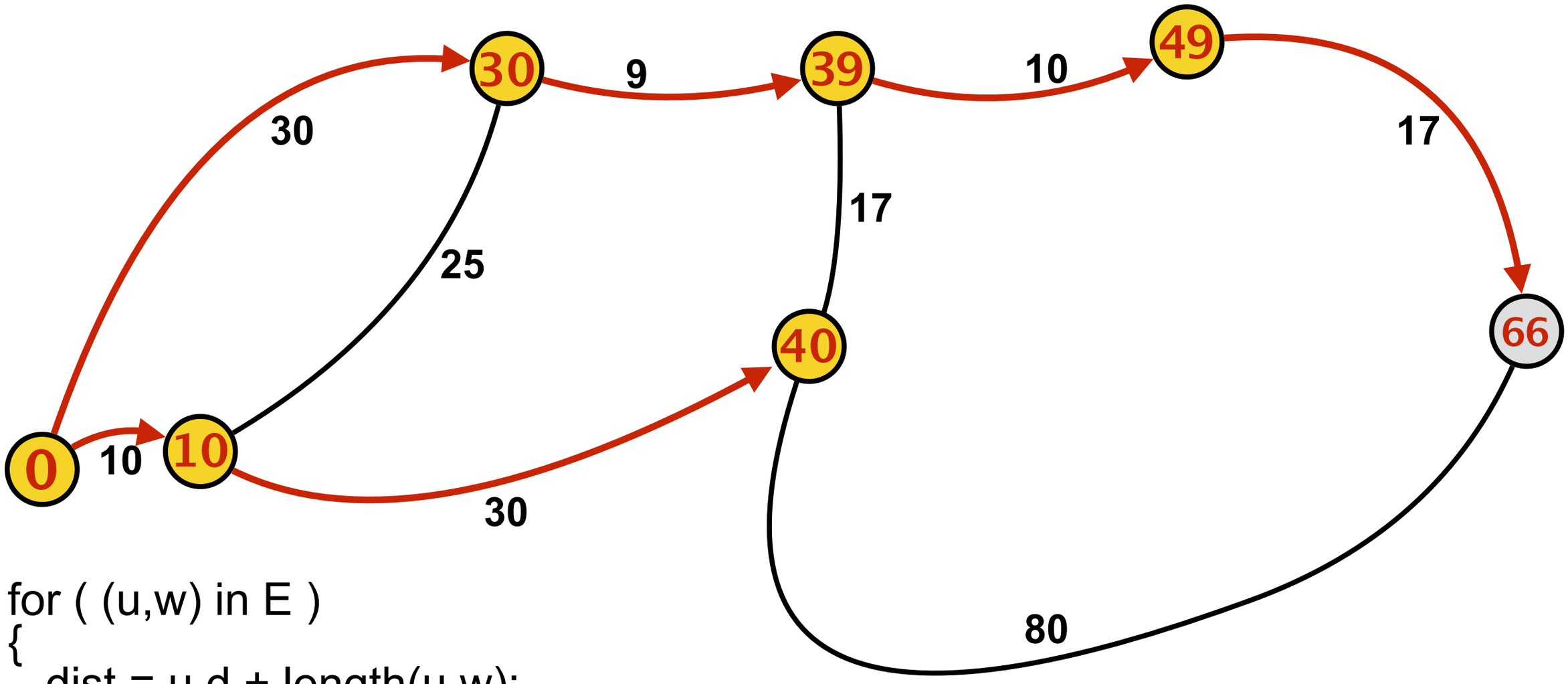
# Beispiel Dijkstra




```
for ( (u,w) in E )
{
   dist = u.d + length(u,w);
   if ( w ∈ queue  AND  w.d > dist )
   {
        w.d = dist;  w.p = (u,w);
   }
   else if (w.p == NULL) { … }
}
```

CoNe Freiburg

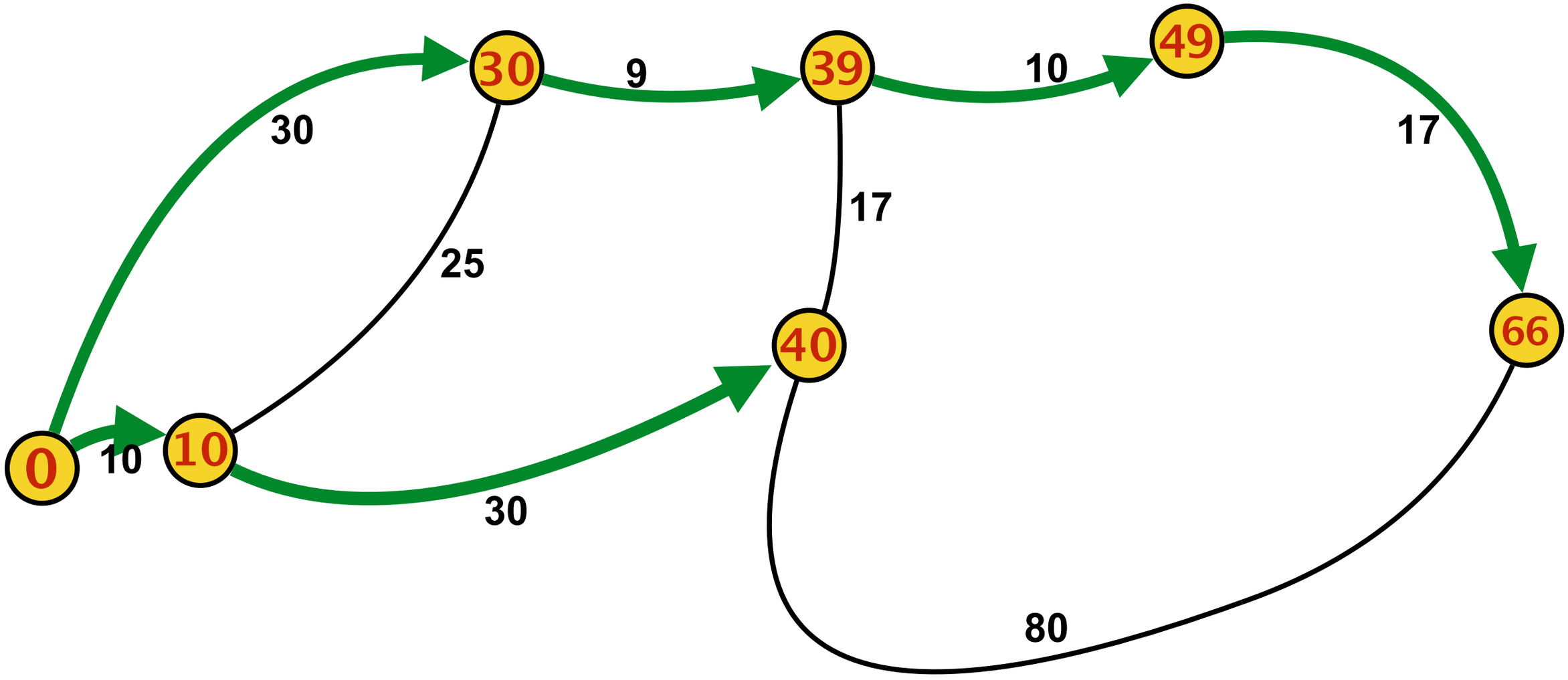
# Beispiel Dijkstra




```
for ( (u,w) in E )
{
   dist = u.d + length(u,w);
   if ( w ∈ queue  AND  w.d > dist )
   {
        w.d = dist;  w.p = (u,w);
   }
   else if (w.p == NULL) { … }
}
```

CoNe Freiburg

# Beispiel Dijkstra



CoNe Freiburg

# Bellman-Ford

- Bei negativen Kantengewichten versagt Dijkstras Algorithmus
- Bellman-Ford
  - löst dies in Laufzeit O(|V| |E|).

## Bellman-Ford$(G, w, s)$

- **Init-Target**$(G, w)$
- loop $|V| - 1$ times:
  - for all $(u, v) \in E$ do
    - **Relax**$(u, v)$
- for all $(u, v) \in E$ do
  - if $d(u) > d(v) + w(u, v)$ then return false

## Init-Target$(G, w, t)$

- **Init-Target**$(G, w)$
- for all $v \in V$ do
  - $d(v) \leftarrow \infty$
  - $\pi(v) \leftarrow v$
- $d(t) \leftarrow 0$

## Relax$(u, v)$

- **Relax**$(u, v)$
- if $d(u) > w(u, v) + d(v)$ then
  - $d(u) \leftarrow w(u, v) + d(v)$
  - $\pi(u) \leftarrow v$

UNI FREIBURG