# Graphentheorie III

Kantonsschule Sursee
Dr. Philipp Hurni

# Agenda

- Die Suche nach dem kürzesten Weg
- Der Algorithmus von Dijkstra
- Übungen zu Dijkstra

- Lernziele für die Prüfung
- Aufgaben aus Prüfung EIN 2023

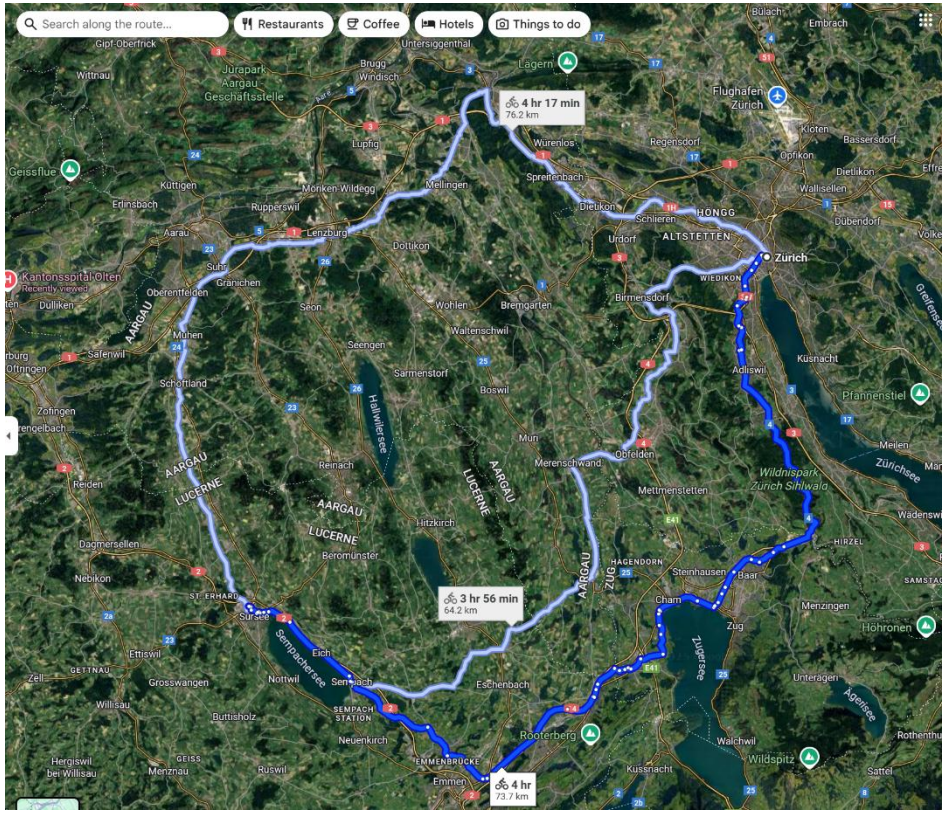
# Die Suche nach dem kürzesten Weg

Wie komme ich am schnellsten nach Zürich?

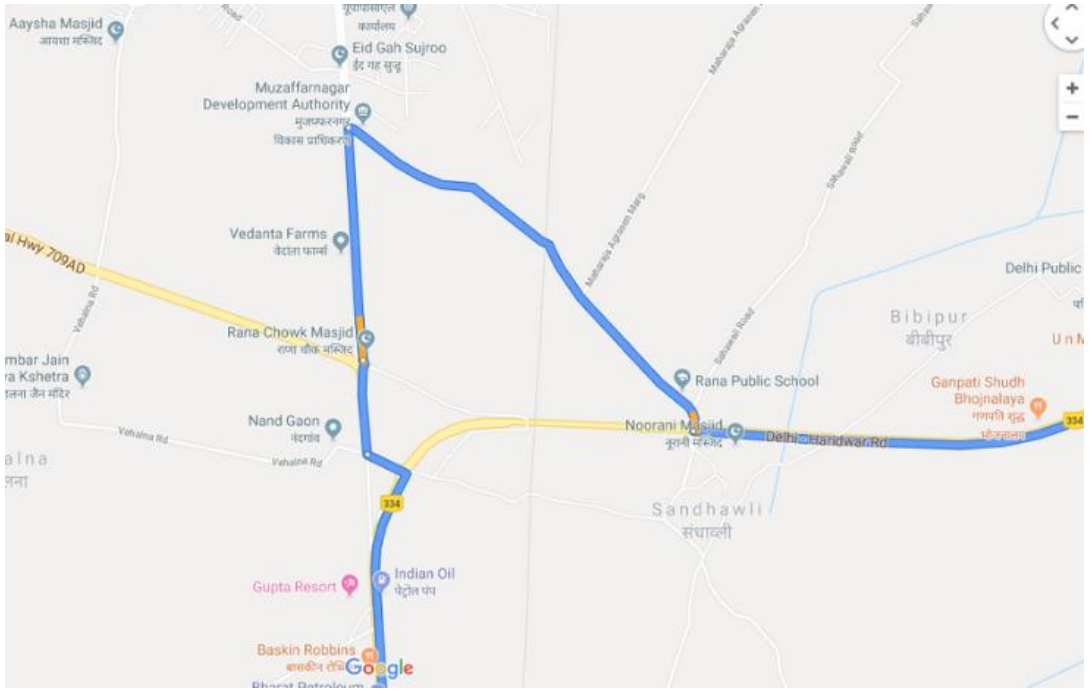
Mit dem Auto?

Mit dem Zug?

Mit dem Fahrrad?



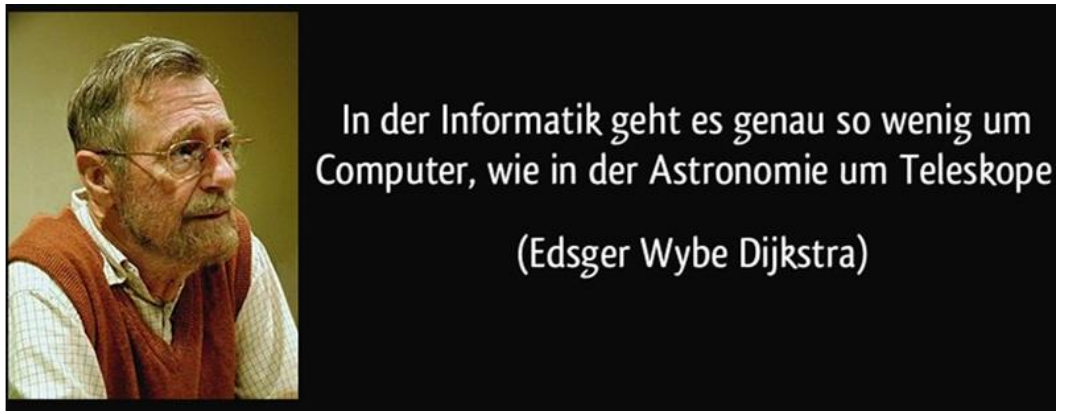
# Navi Fails


Aaysha Masjid
Eid Gah Sujroo
Muzaffarnagar Development Authority
Vedanta Farms
Rana Chowk Masjid
Rana Public School
Nand Gaon
Noorani Masjid
Delhi - Haridwar Rd
Ganpati Shudh Bhojnalaya
Bibipur
Delhi Public
Sandhawli
Gupta Resort
Indian Oil
Baskin Robbins
Google
334

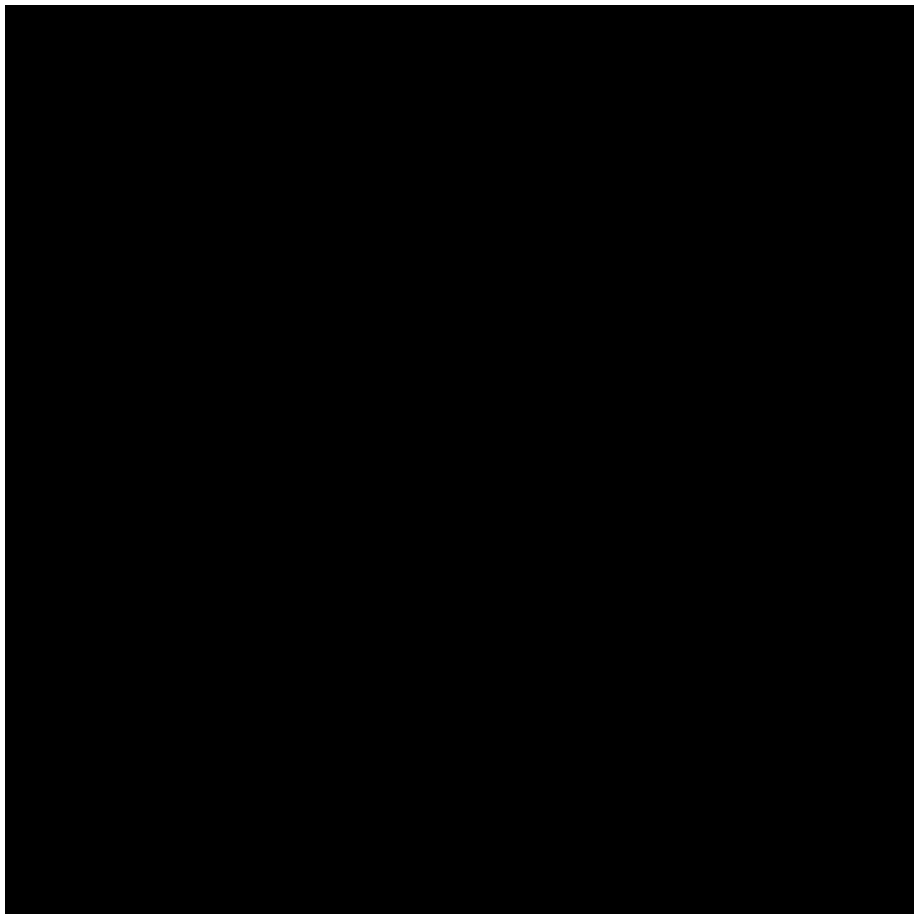
# Navi Fails

# Der Algorithmus von Dijkstra

- Edsger Dijkstra (1930-2002) – einer der bedeutendsten Computerwissenschaftler
- Algorithmus von Dijkstra
  - Gewichteter Graph
  - Startknoten A
  - Auftrag: Finde den kürzesten Weg zu einem anderen Knoten B
  - Findet immer einen optimalen Weg



# Konzept: "Abtasten" bis zum Ziel



# Ablauf des Algorithmus

**Schritt 1:** Initialisierung

Du startest am Startknoten (E) und legst seine Distanz auf 0 fest. Alle anderen Knoten werden auf eine Distanz von unendlich gesetzt, da wir sie noch nicht erreicht haben.

**Schritt 2:** Besuchte Knoten

Erstelle eine Liste der besuchten Knoten. Am Anfang ist diese Liste leer. Wähle den Knoten mit der kleinsten Distanz, die noch nicht in der Liste der besuchten Knoten enthalten ist, und nimm ihn auf in die Liste Besucht. Zuerst ist dies unser Startknoten (E).

**Schritt 3:** Nachbarknoten untersuchen

Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

# Ablauf des Algorithmus

**Schritt 4:** Distanzen aktualisieren

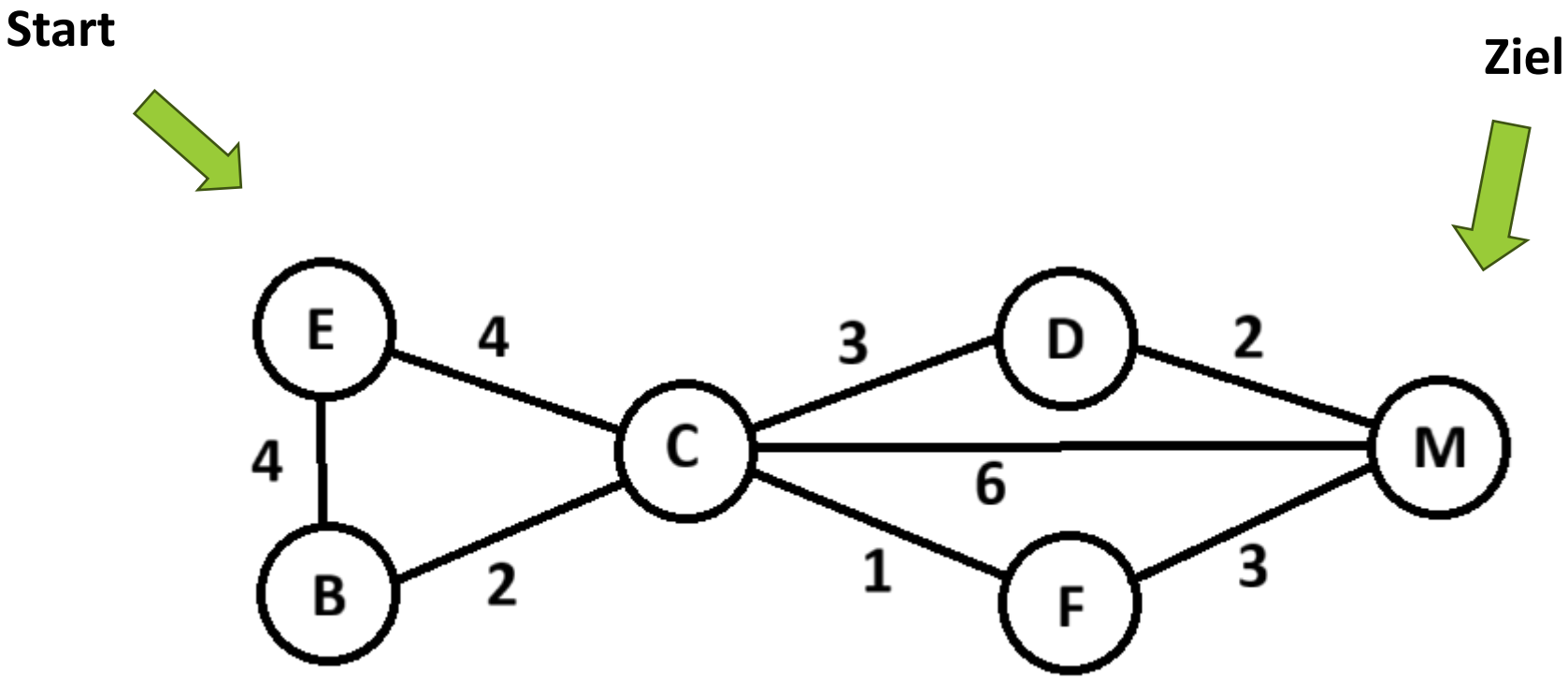
Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn. Du speicherst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

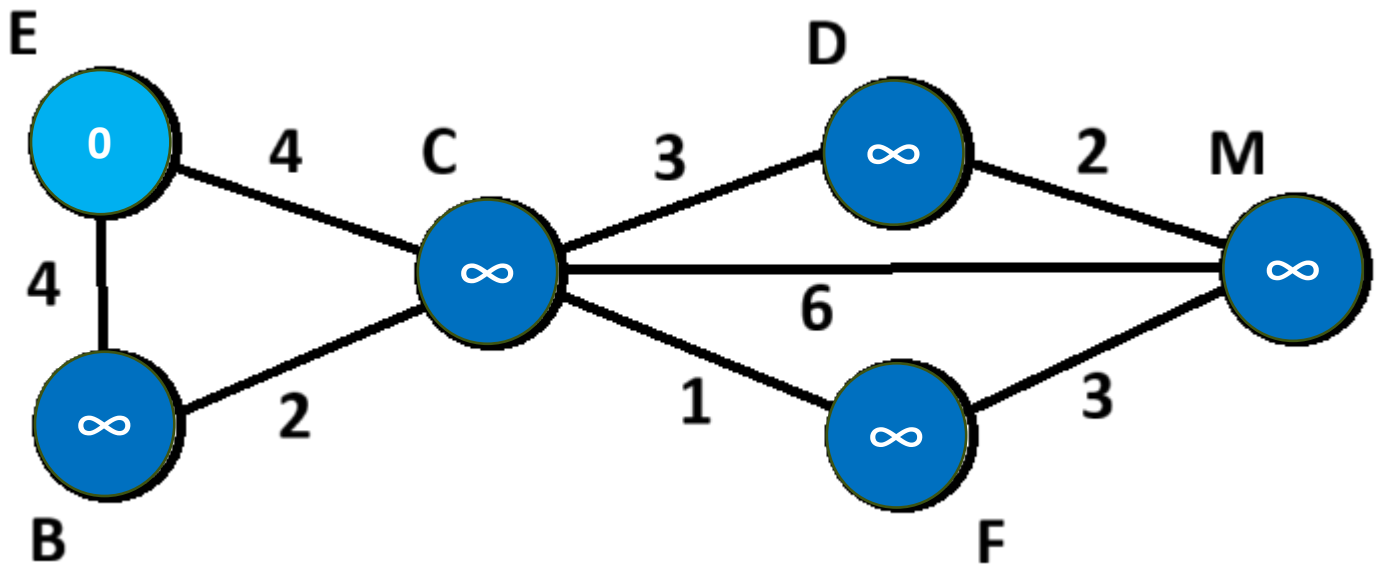
Das machst du so lange, bis du den Zielknoten (M) erreicht hast oder keine Knoten mehr zu besuchen sind.

# Ablauf des Algorithmus




Beginne mit dem (üblicherweise) gewichteten Graph

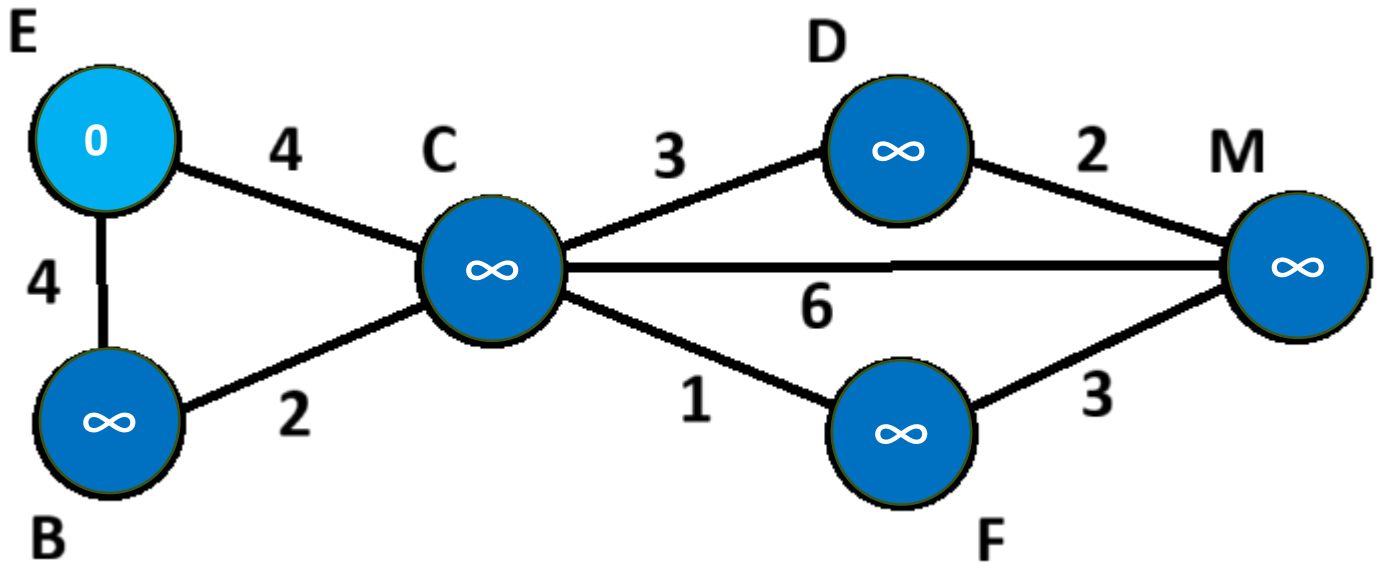
# Schritt 1: Initialisierung




Du startest am Startknoten (hier E) und legst seine Distanz auf 0 fest. Alle anderen Knoten werden auf eine Distanz von unendlich gesetzt, da wir sie noch nicht erreicht haben.

# Schritt 2: Besuchte Knoten

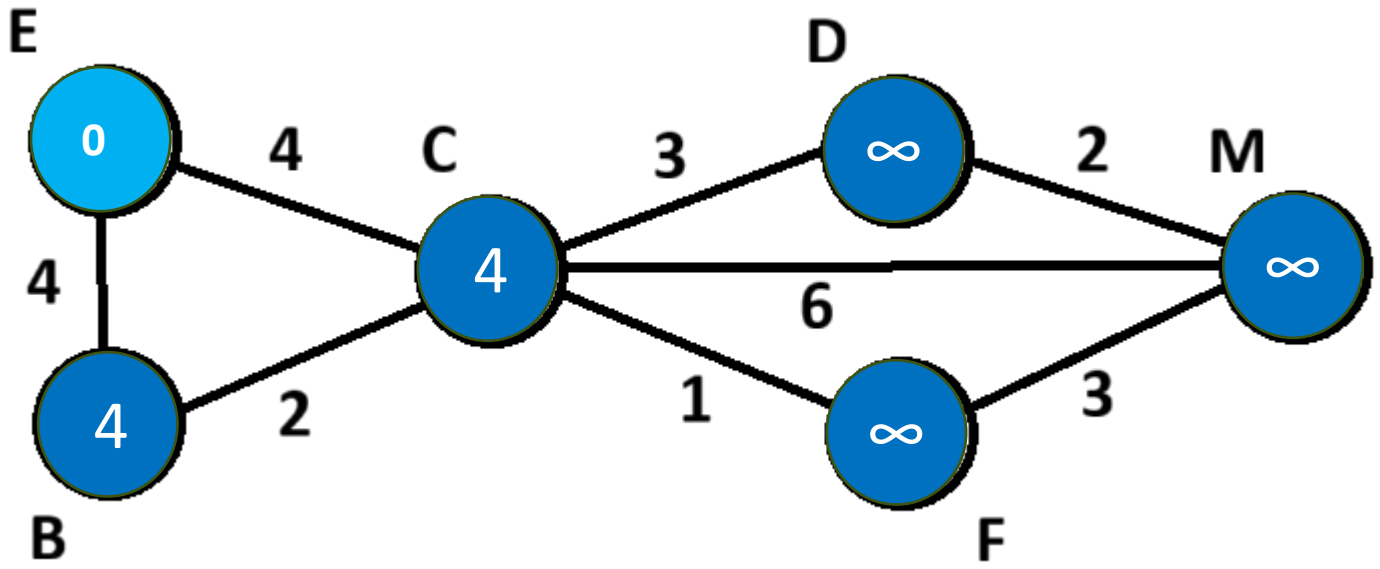
Besucht = [E]




Erstelle eine Liste der besuchten Knoten. Am Anfang ist diese Liste leer. Wähle den Knoten mit der kleinsten Distanz, die noch nicht in der Liste der besuchten Knoten enthalten ist. Zuerst ist dies unser Startknoten (E).

# Schritt 3: Nachbarknoten untersuchen

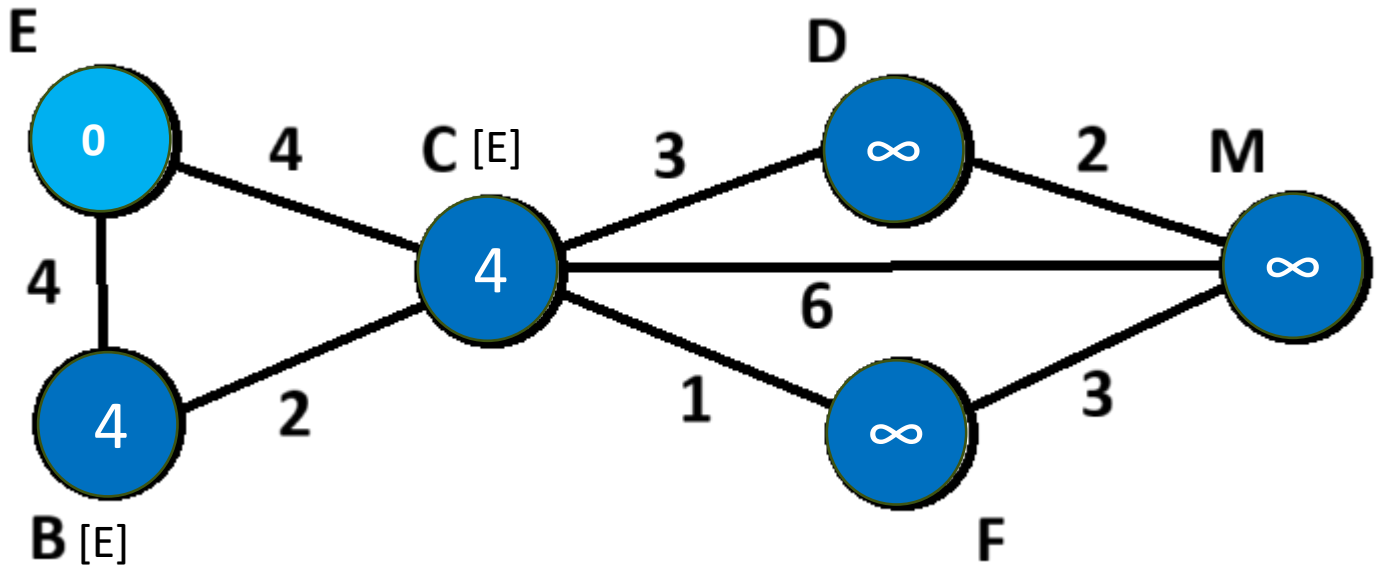
Besucht = [E]




Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

# Schritt 4: Distanzen aktualisieren

Besucht = [E]

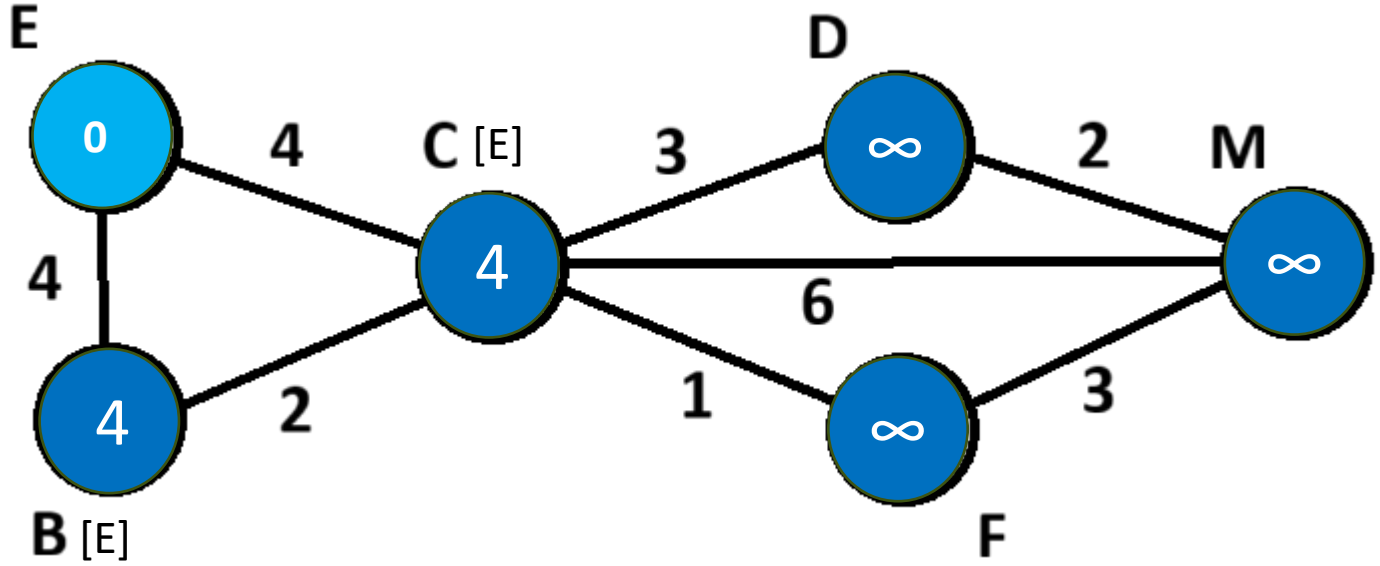



Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn (Hier: nichts weiteres tun)

Du speicherst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

# Schritt 5: Wiederholen

Besucht = [E]



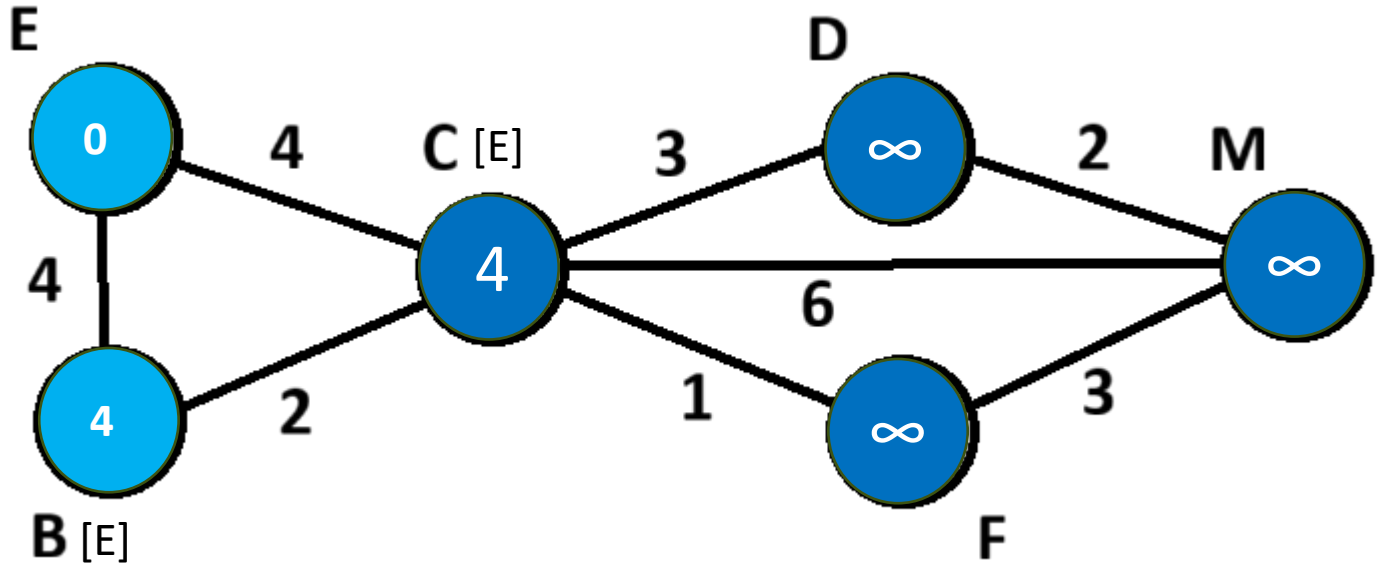

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das machst du so lange, bis du den Zielknoten (M) erreicht hast oder keine Knoten mehr zu besuchen sind.

Hier: wähle B oder C (beide 4). Wir wählen B

# Schritt 2: Besuchte Knoten

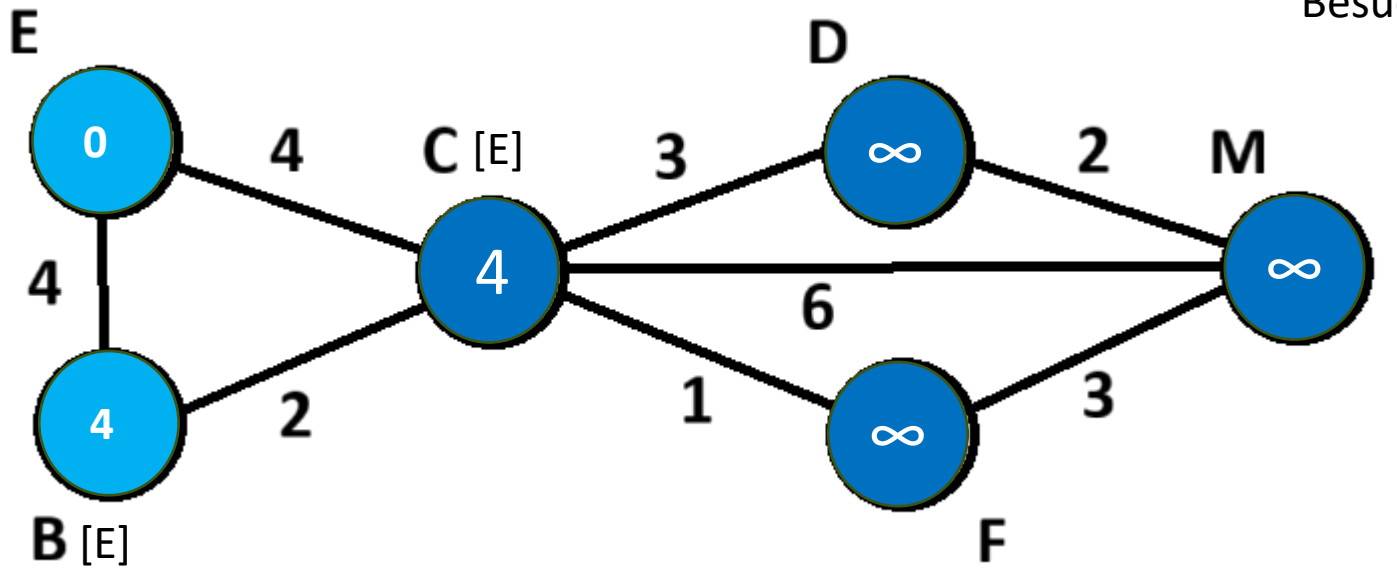
Besucht = [E, B]




Wähle den Knoten mit der kleinsten Distanz, die noch nicht in der Liste der besuchten Knoten enthalten ist. Hier B.

# Schritt 3: Nachbarknoten untersuchen

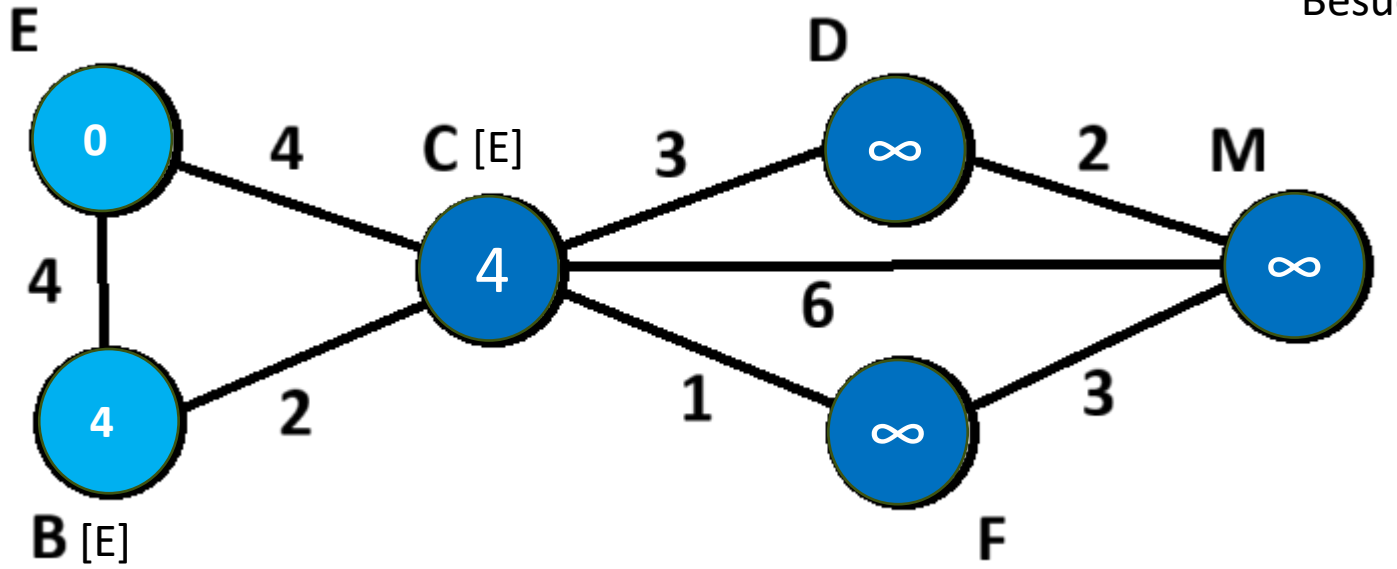
Besucht = [E, B]




Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

# Schritt 4: Distanzen aktualisieren

Besucht = [E, B]

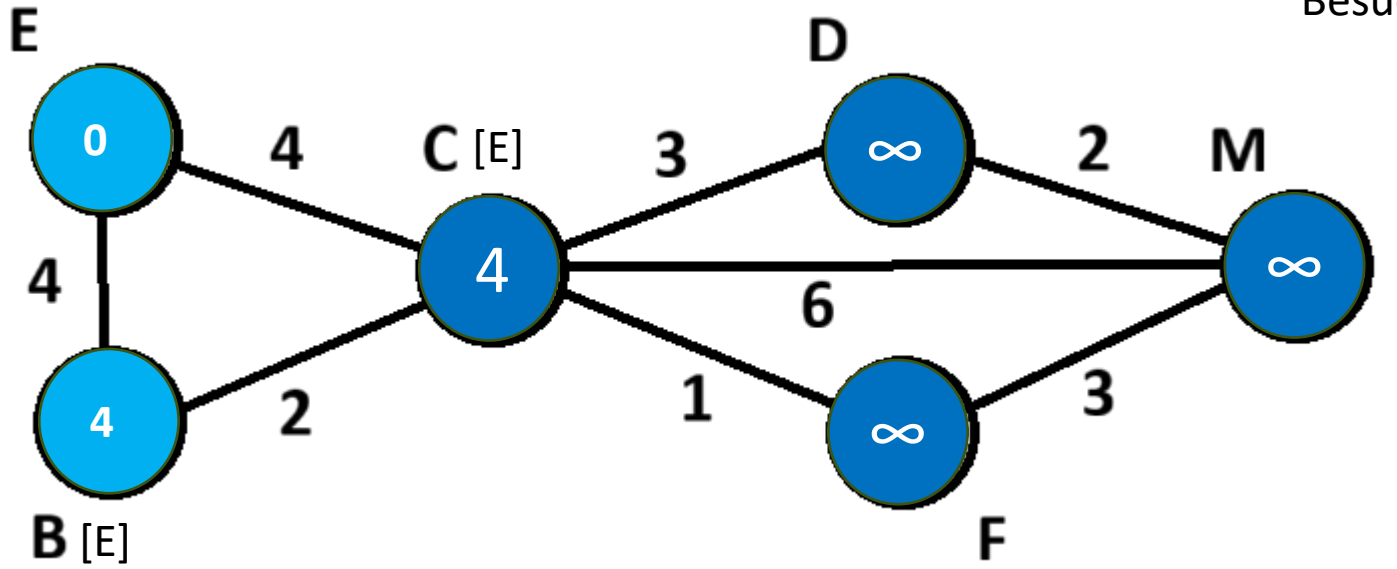



Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn (Hier: nichts weiteres tun, weil 4 < 6)

Du speicherst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

# Schritt 5: Wiederholen

Besucht = [E, B]



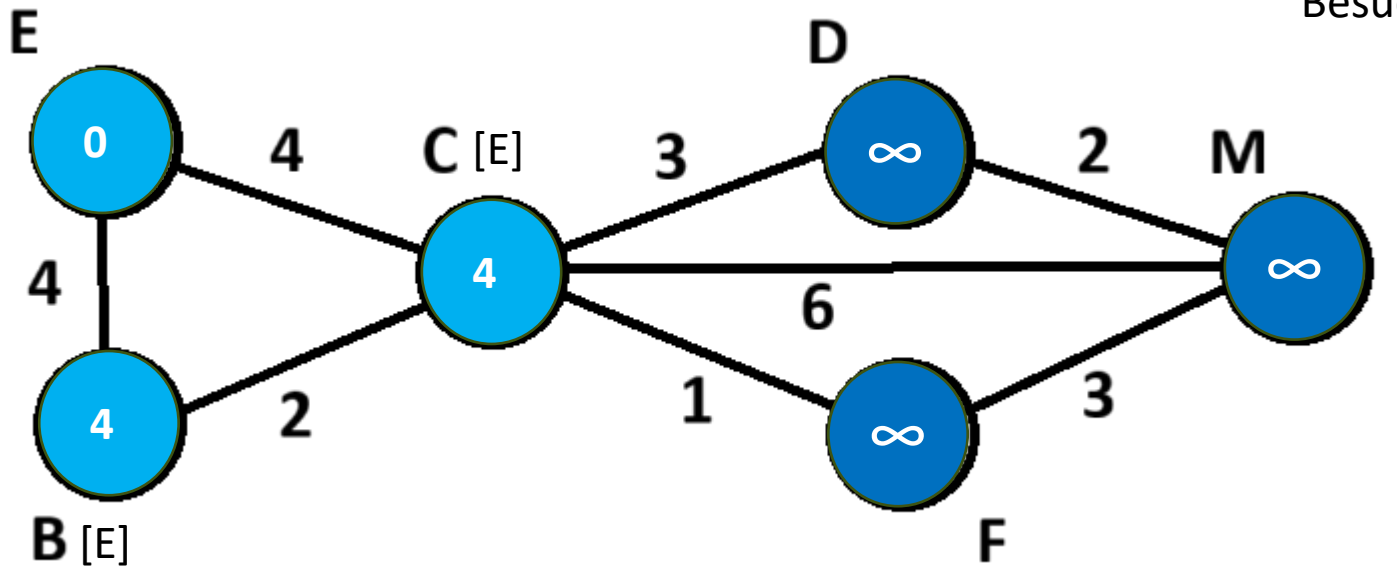

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das machst du so lange, bis du den Zielknoten (M) erreicht hast oder keine Knoten mehr zu besuchen sind.

Hier: wähle C

# Schritt 2: Besuchte Knoten

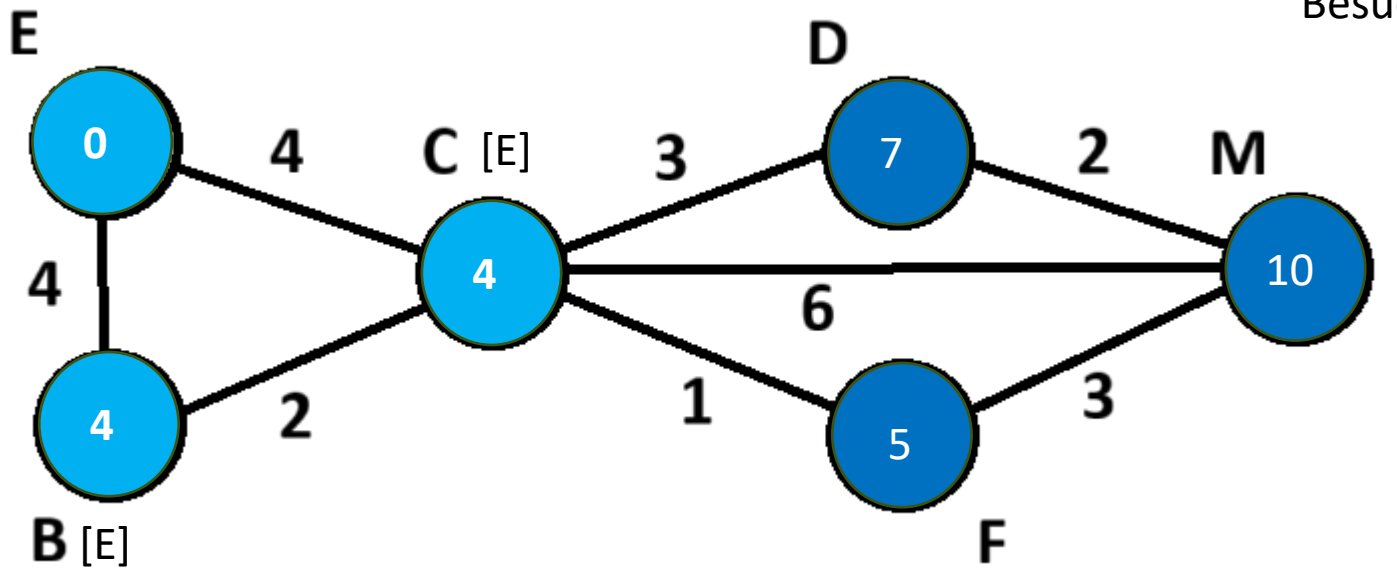
Besucht = [E, B, C]




Wähle den Knoten mit der kleinsten Distanz, die noch nicht in der Liste der besuchten Knoten enthalten ist. Hier C.

# Schritt 3: Nachbarknoten untersuchen

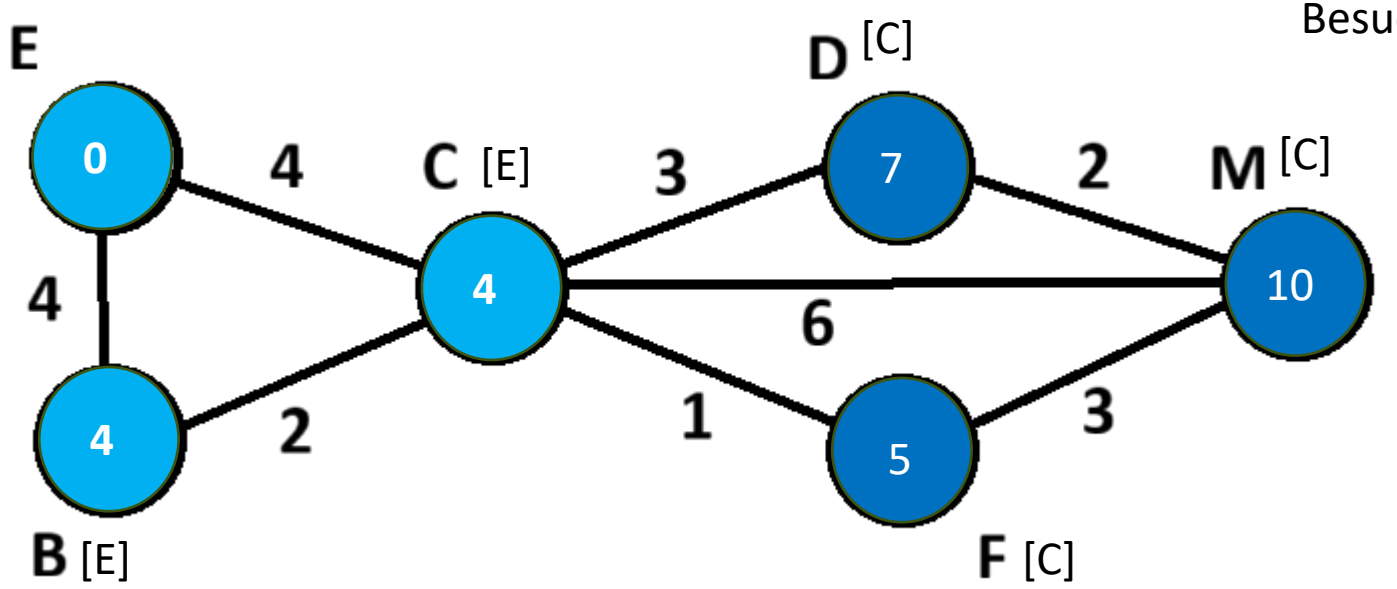
Besucht = [E, B, C]




Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

# Schritt 4: Distanzen aktualisieren

Besucht = [E, B, C]

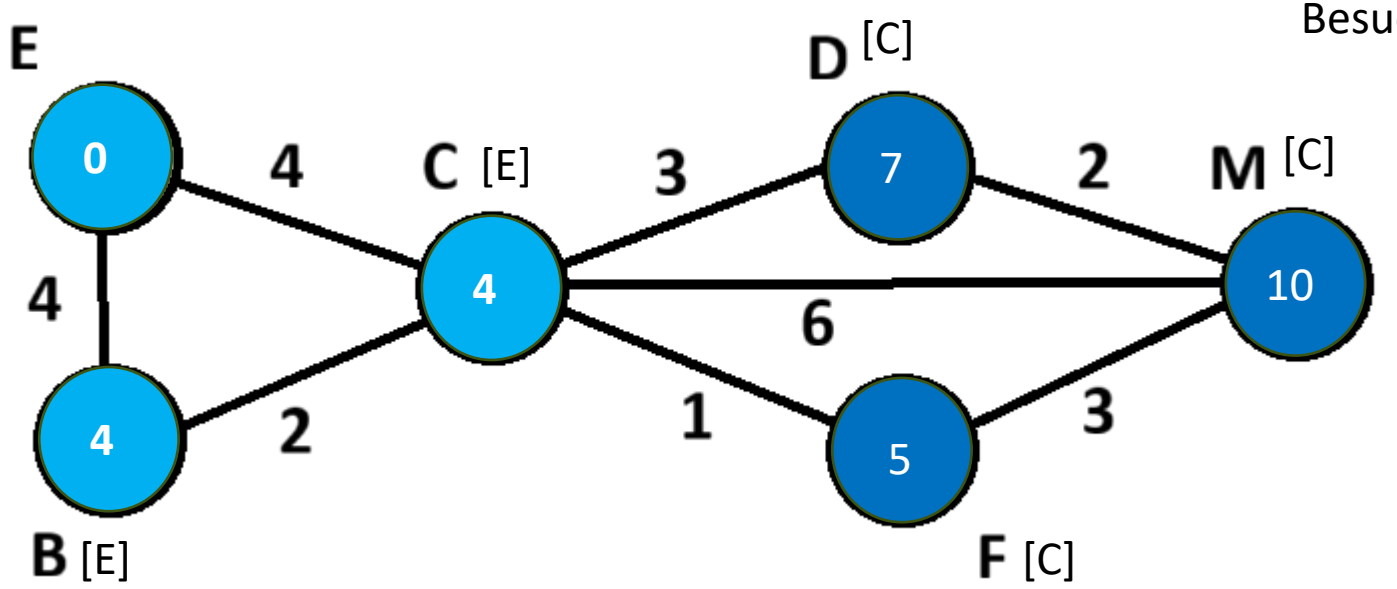



Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn (Hier: nichts weiteres tun)

Du speicherst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

# Schritt 5: Wiederholen

Besucht = [E, B, C]



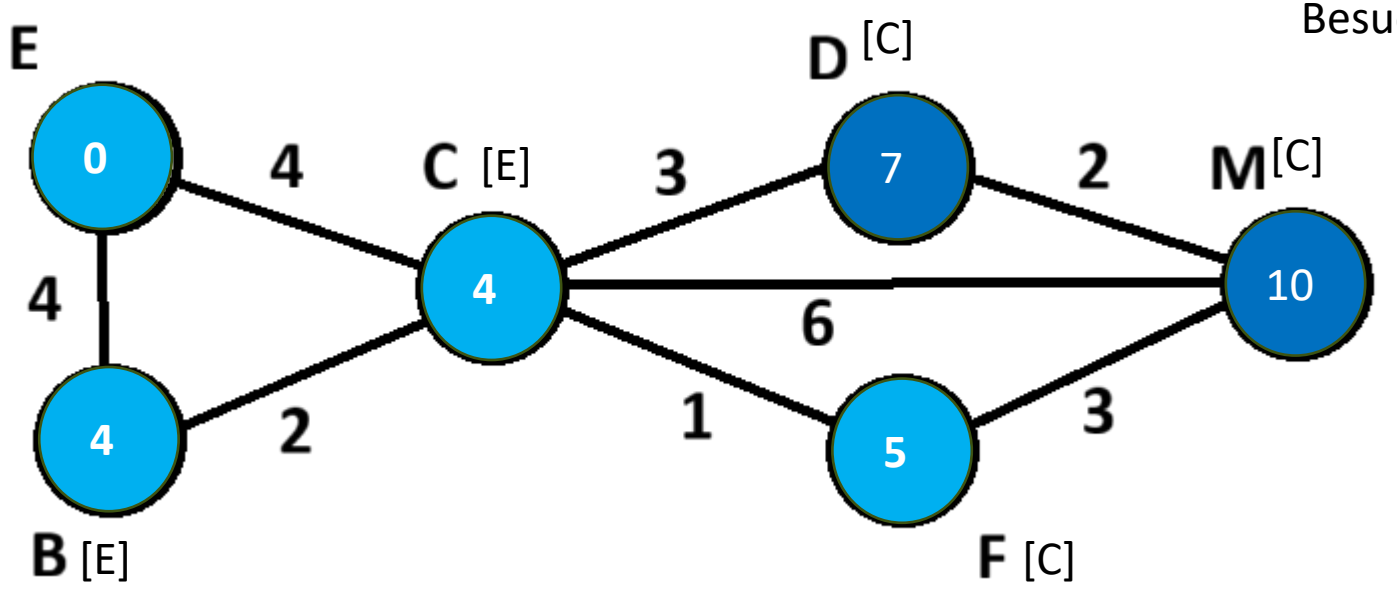

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das machst du so lange, bis du den Zielknoten (M) erreicht hast oder keine Knoten mehr zu besuchen sind.

Hier: wähle F

# Schritt 2: Besuchte Knoten

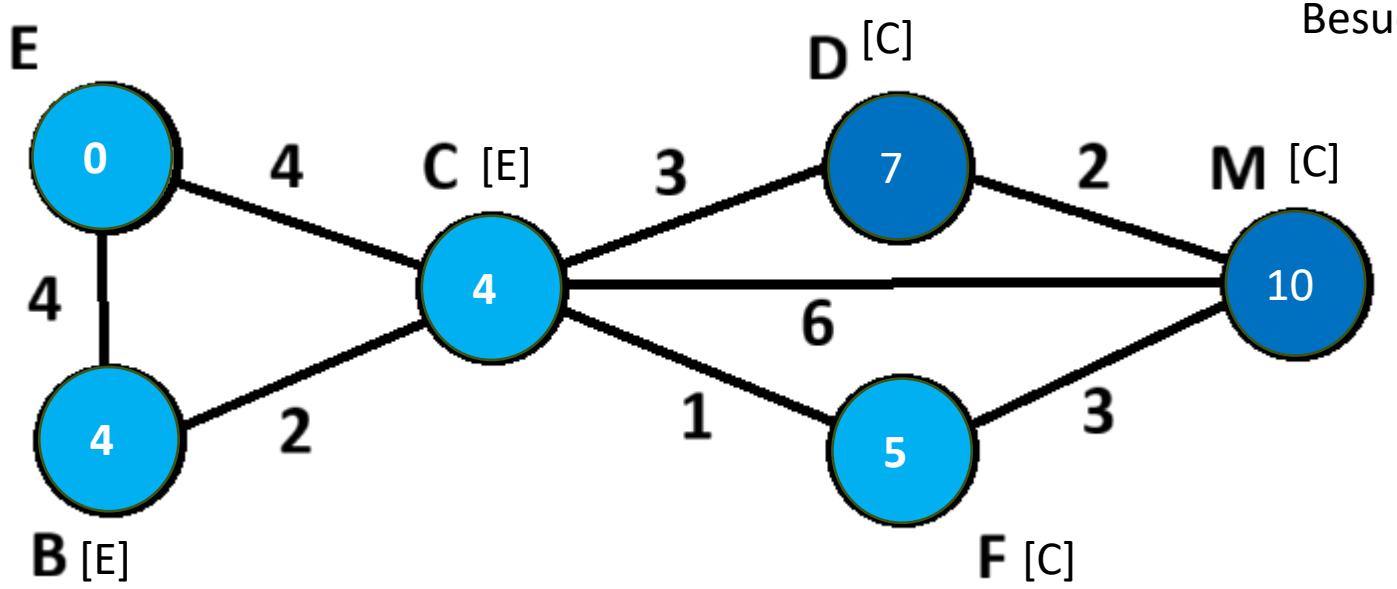
Besucht = [E, B, C, F]




Wähle den Knoten mit der kleinsten Distanz, die noch nicht in der Liste der besuchten Knoten enthalten ist. Hier F.

# Schritt 3: Nachbarknoten untersuchen

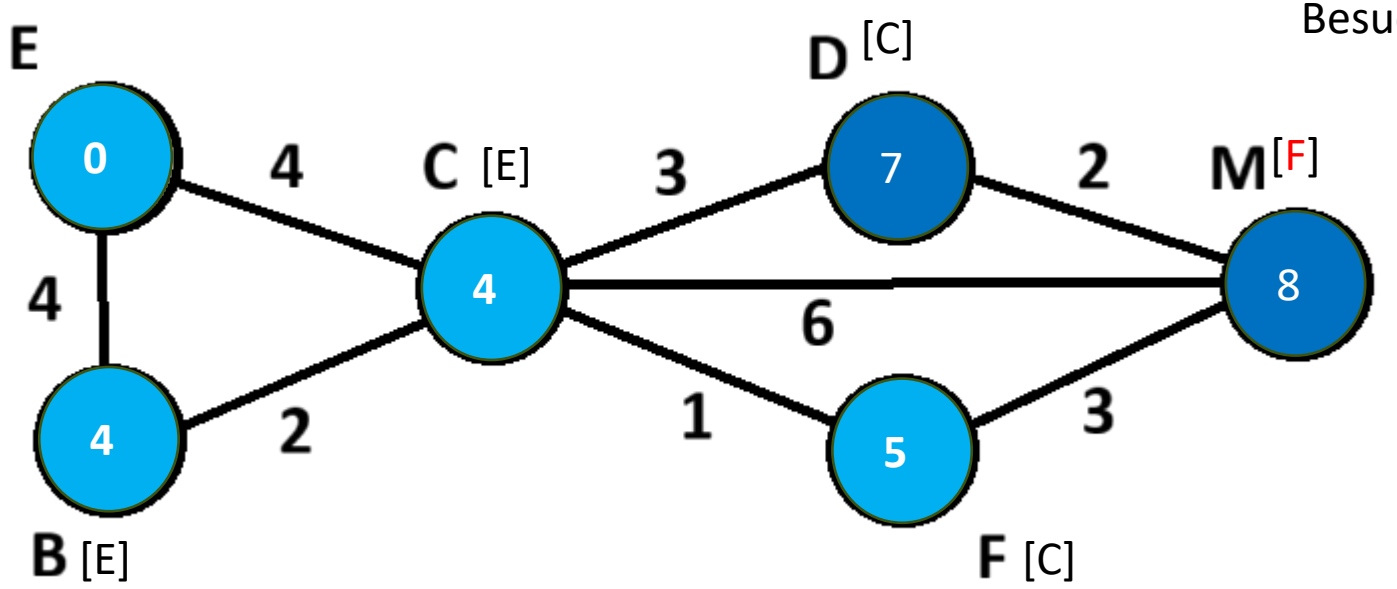
Besucht = [E, B, C, F]




Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

# Schritt 4: Distanzen aktualisieren

Besucht = [E, B, C, F]

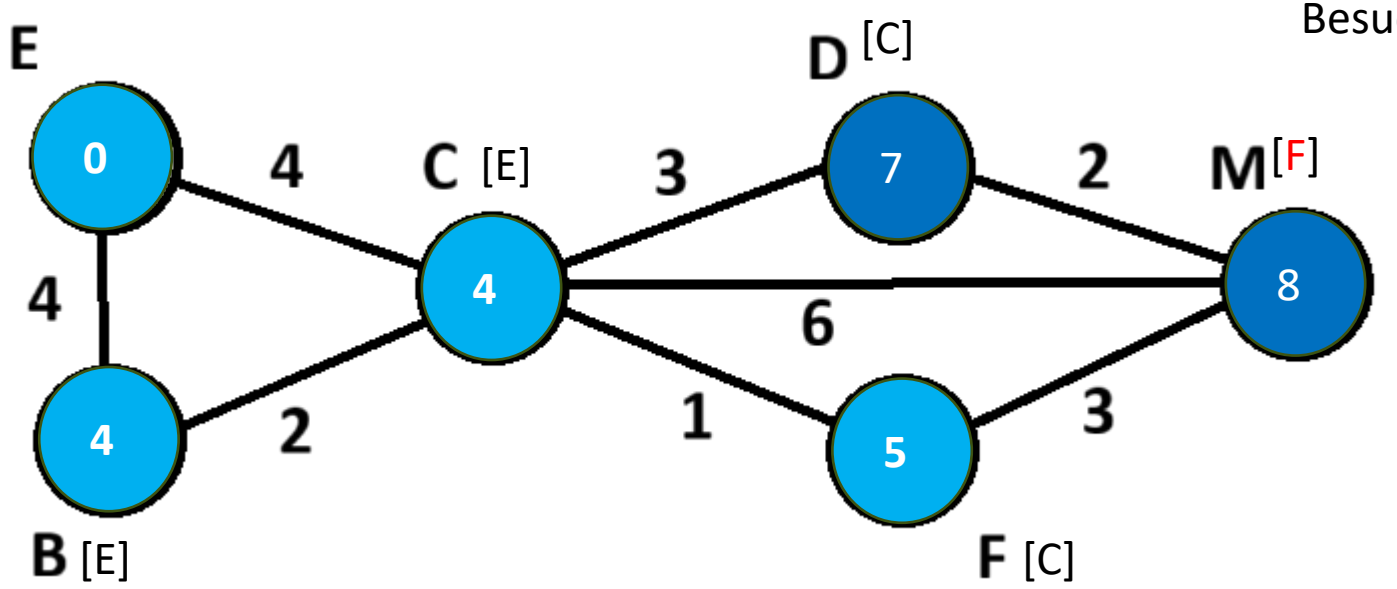



Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn (Hier: aktualisiere M: 8 ist kleiner als 10)

Du speicherst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

# Schritt 5: Wiederholen

Besucht = [E, B, C, F]



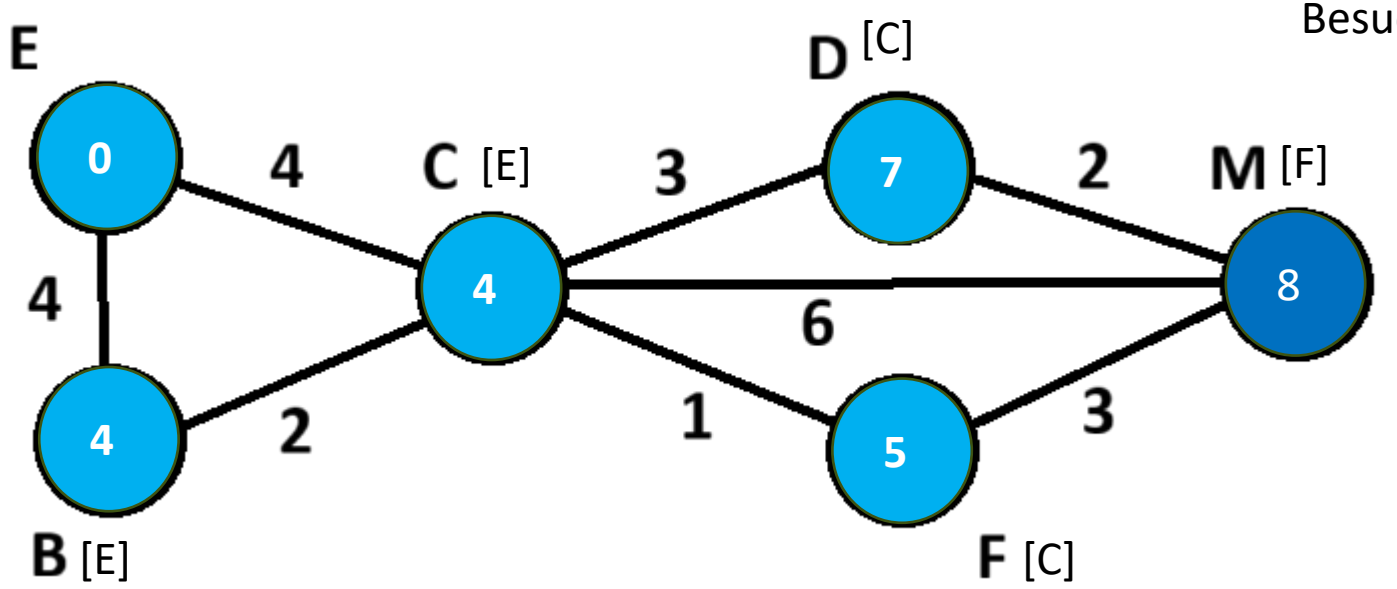

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das machst du so lange, bis du den Zielknoten (M) erreicht hast oder keine Knoten mehr zu besuchen sind.

Hier: wähle D

# Schritt 2: Besuchte Knoten

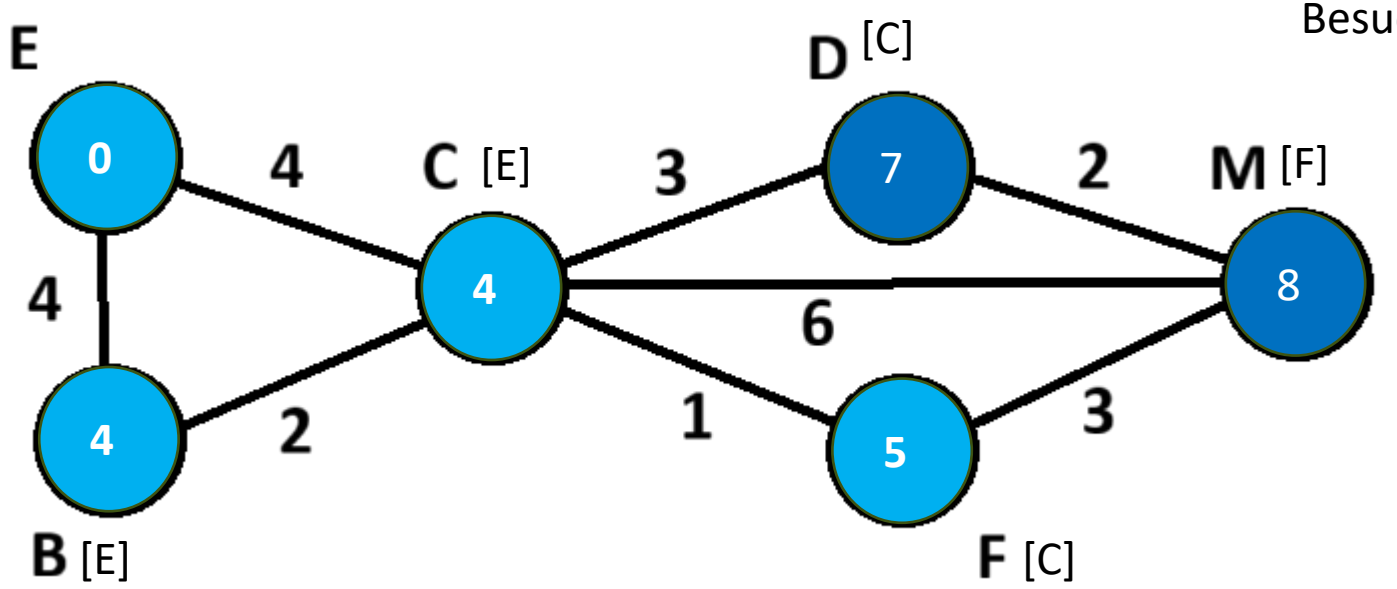
Besucht = [E, B, C, F, D]




Wähle den Knoten mit der kleinsten Distanz, die noch nicht in der Liste der besuchten Knoten enthalten ist. Hier D.

# Schritt 3: Nachbarknoten untersuchen

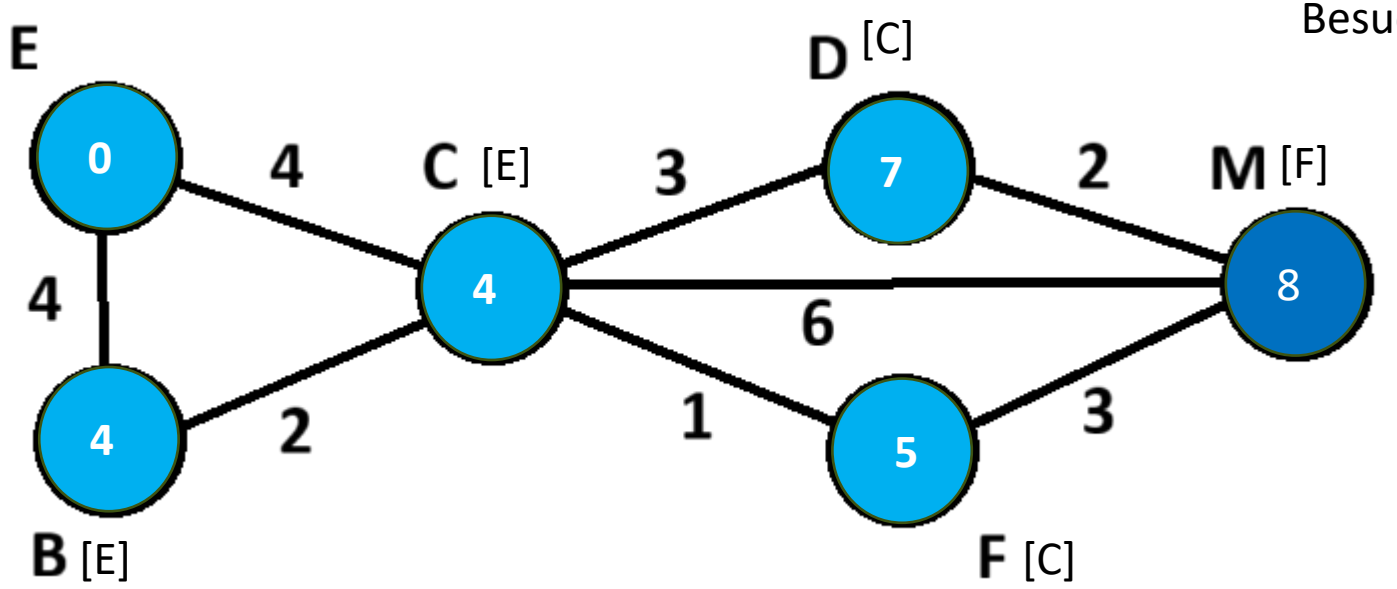
Besucht = [E, B, C, F, D]




Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

# Schritt 4: Distanzen aktualisieren

Besucht = [E, B, C, F, D]

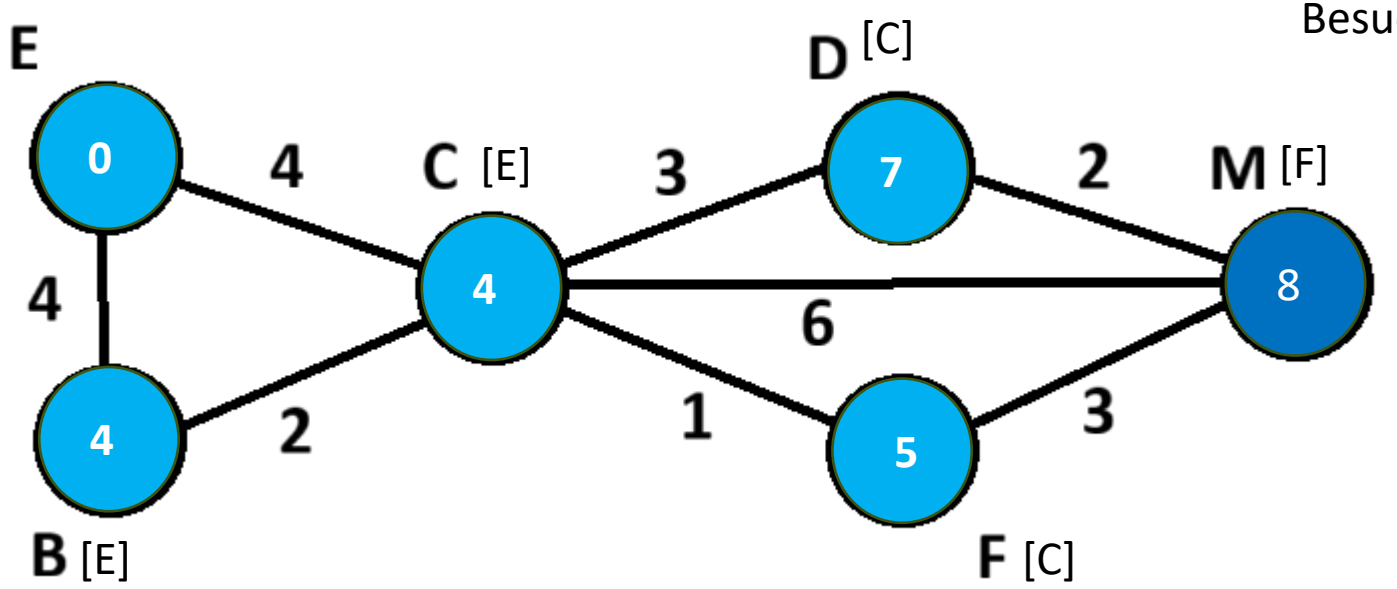



Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn (Hier: nichts tun: 8 ist kleiner als 9)

Du speicherst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

# Schritt 4: Distanzen aktualisieren

Besucht = [E, B, C, F, D]



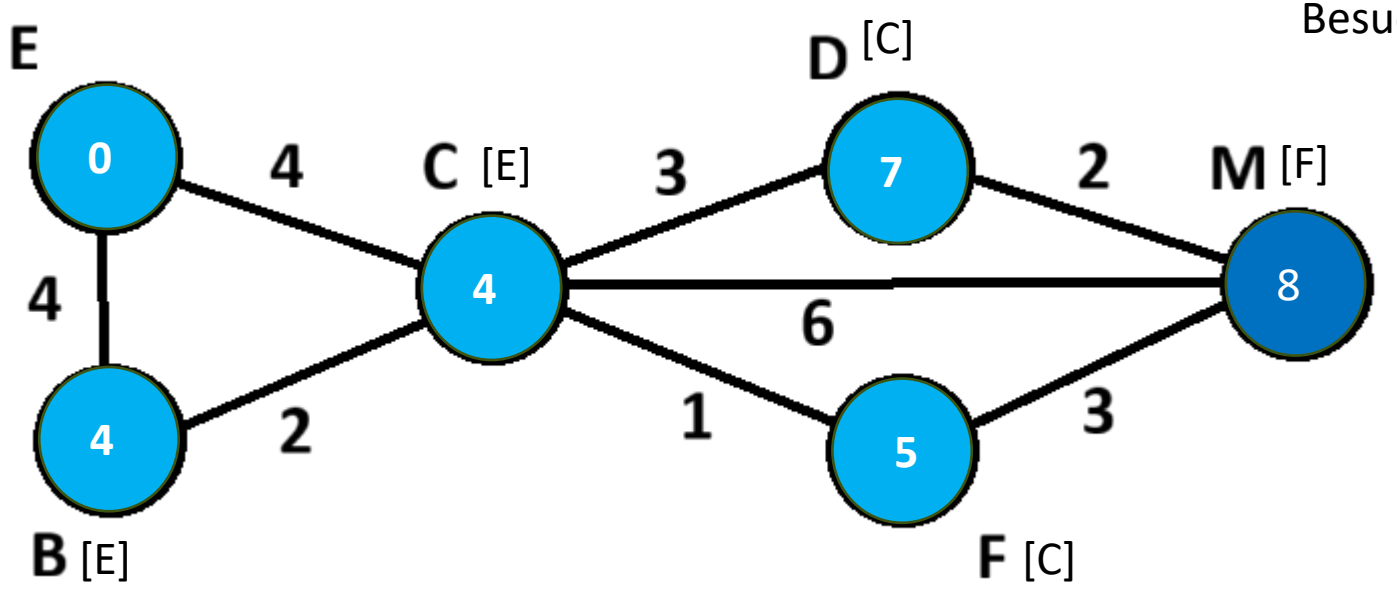

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das machst du so lange, bis du den Zielknoten (M) erreicht hast oder keine Knoten mehr zu besuchen sind.

Hier: wähle M

# Schritt 5: Wiederholen

Besucht = [E, B, C, F, D]

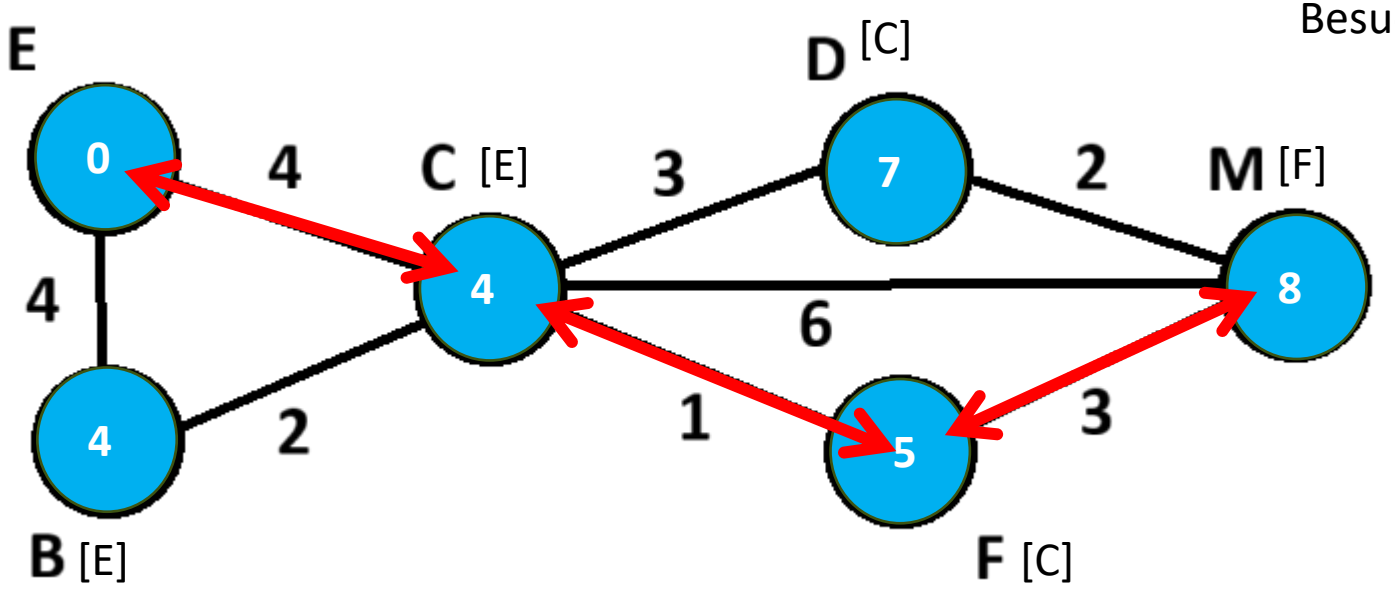



Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das machst du so lange, bis du den Zielknoten (M) erreicht hast oder keine Knoten mehr zu besuchen sind.

Hier: M wird erreicht! Wir sind fertig!

# Kürzester Weg



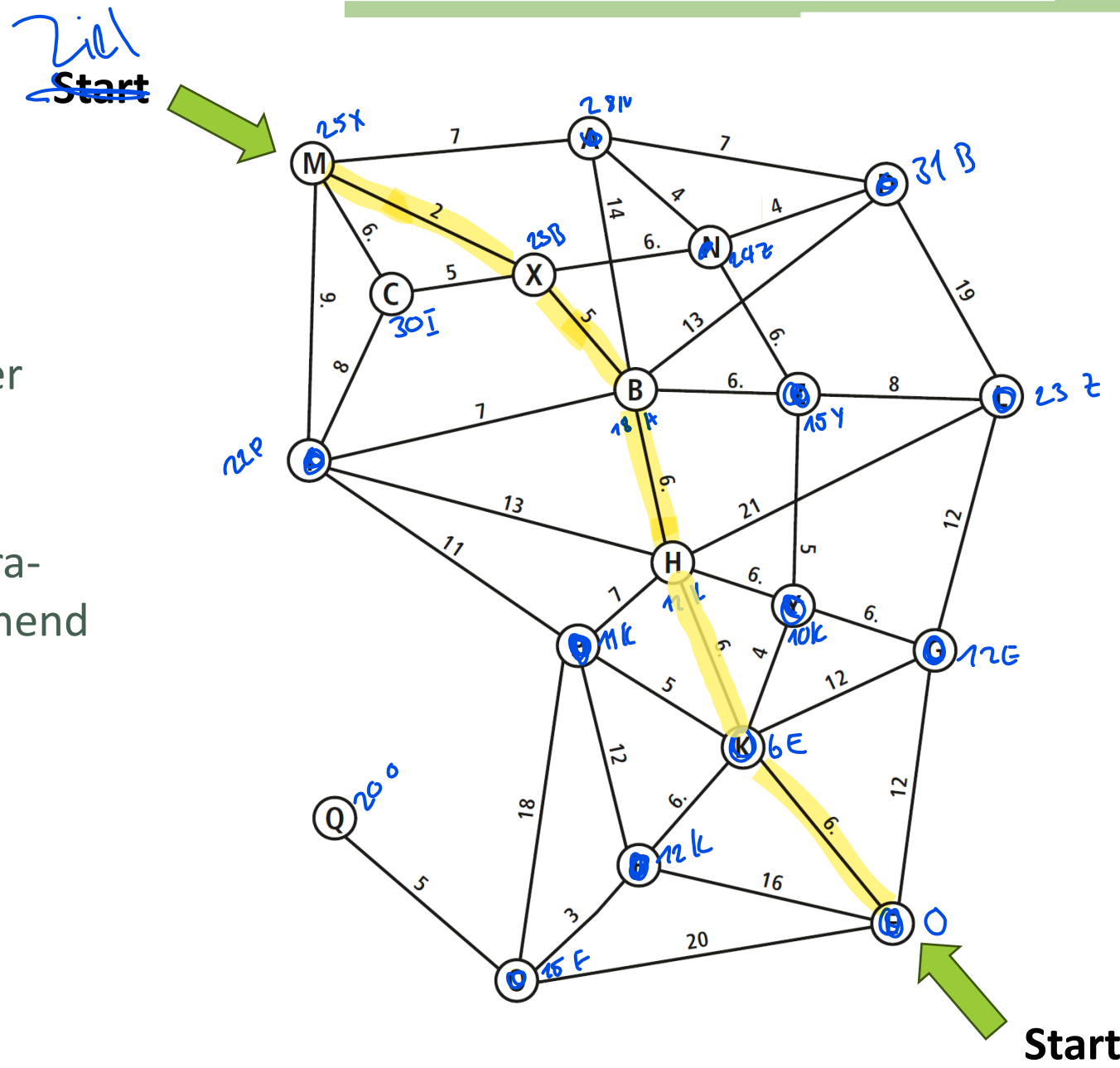

Besucht = [E, B, C, F, D, M]

# Übungen zu Kürzester Weg / Dijkstra

# Übung A - Dijkstra

Gegeben ist der folgende Graph, welcher Städte in Deutschland symbolisiert.

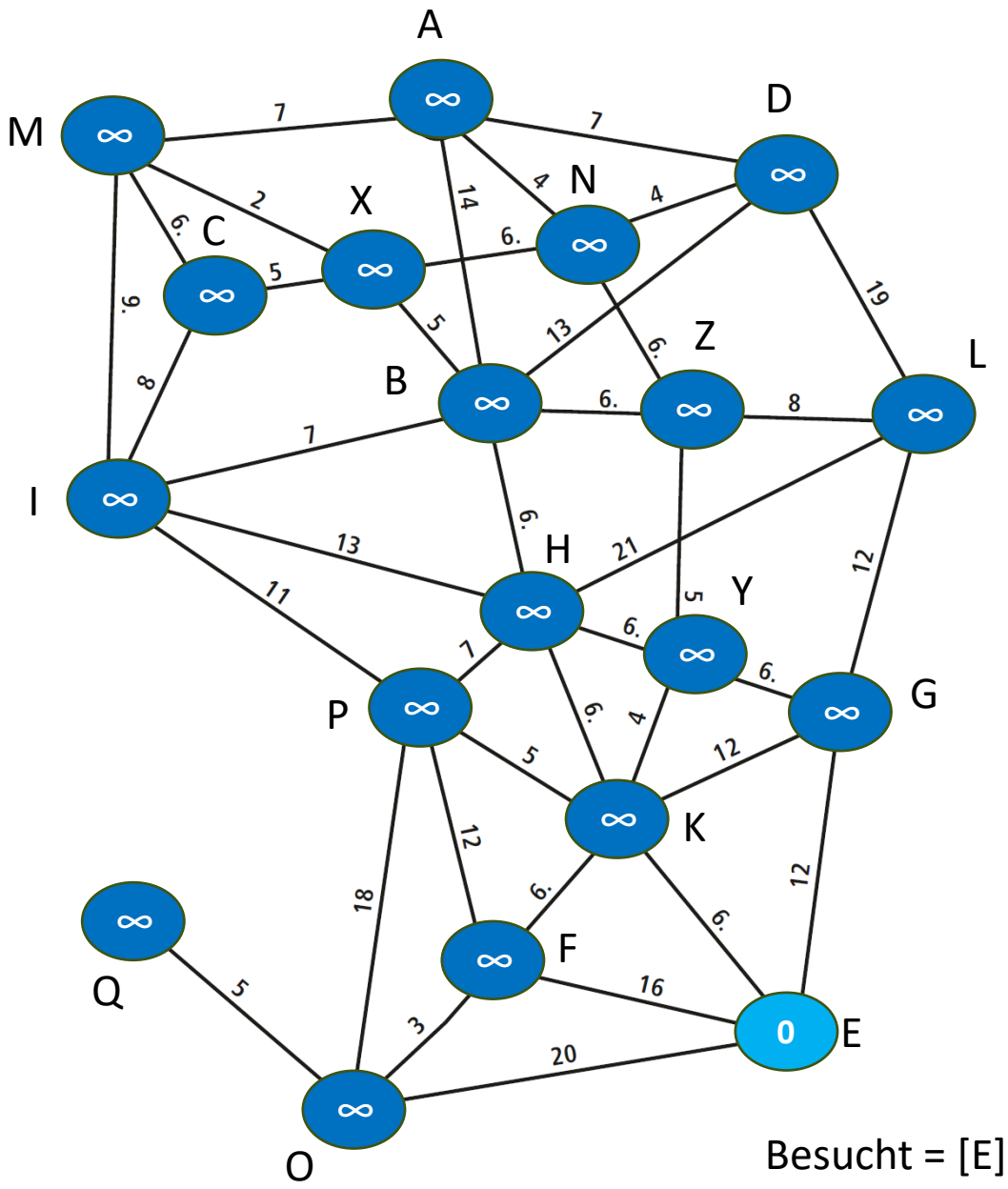
Bestimme durch Anwendung des Dijkstra-Algorithmus alle kürzesten Wege ausgehend von Knoten E zu M



# Lösung zu A

**Schritt 1:** Initialisierung

Du startest am Startknoten (E) und legst seine Distanz auf 0 fest. Alle anderen Knoten werden auf eine Distanz von unendlich gesetzt, da wir sie noch nicht erreicht haben.

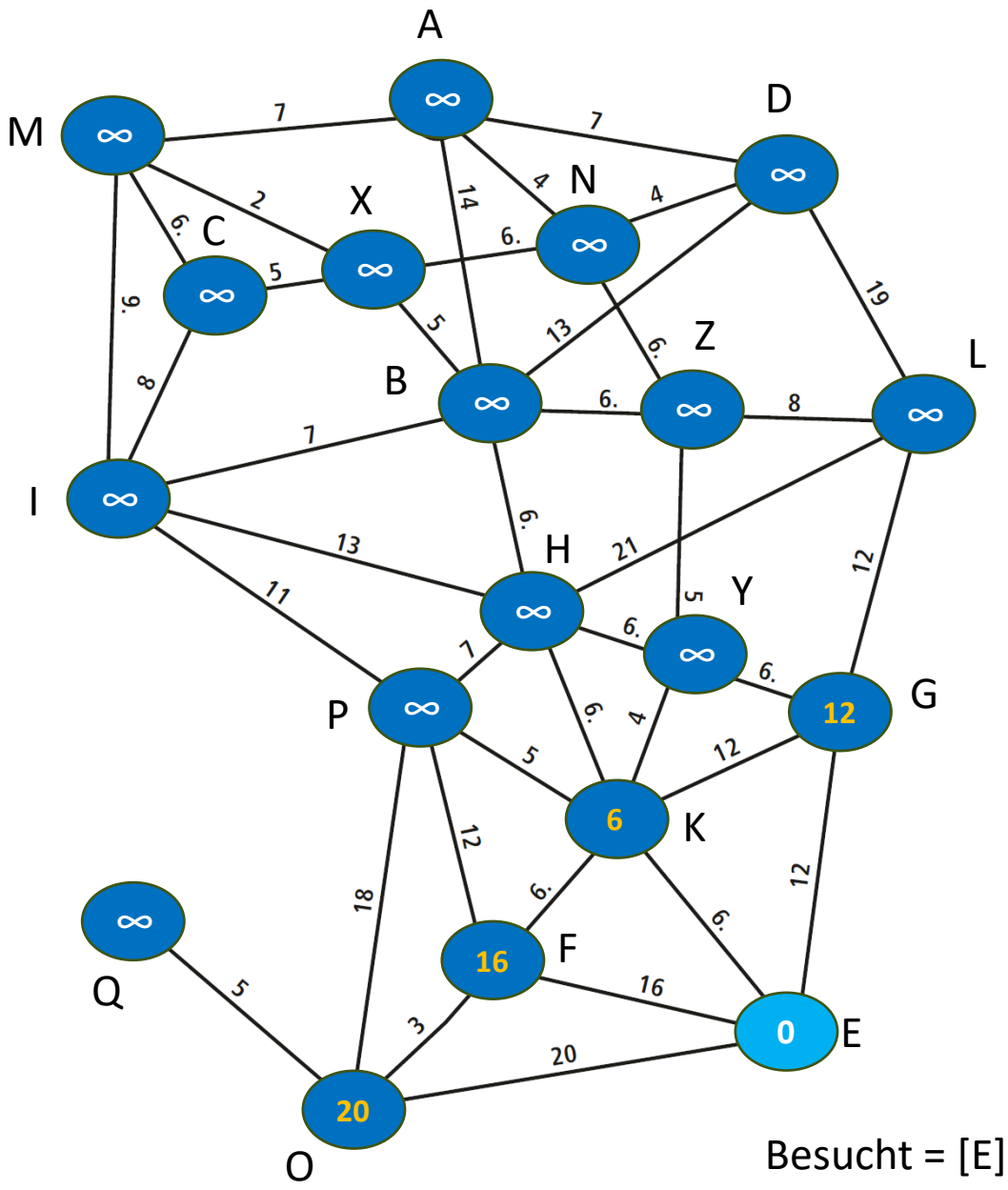


**Schritt 2:** Besuchte Knoten
Erstelle eine Liste der besuchten Knoten. Am Anfang ist diese Liste leer.

Zuerst ist dies unser Startknoten – also der unten rechts mit 0.

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Merke: aktuelle Distanz ist hier 0

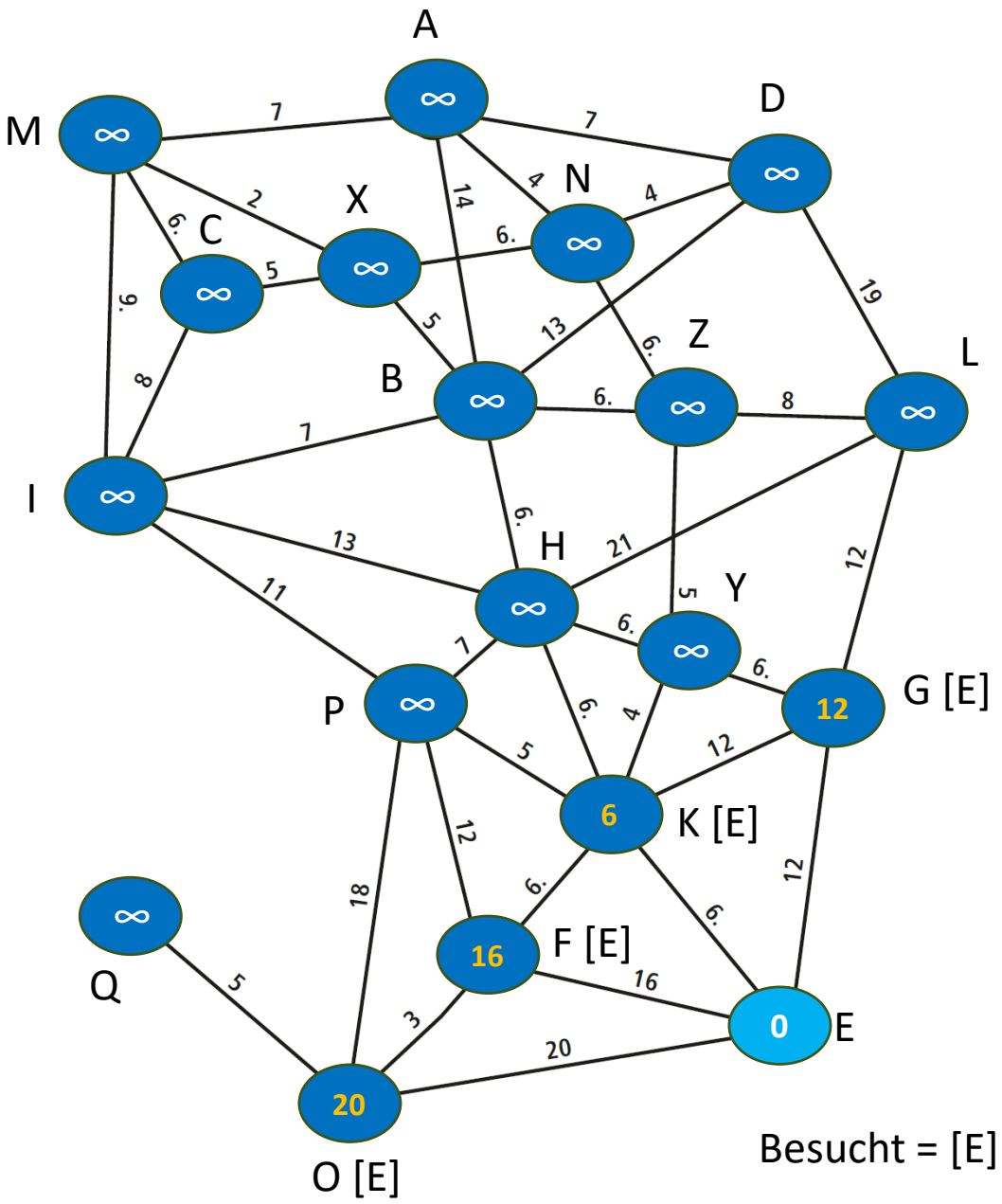


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

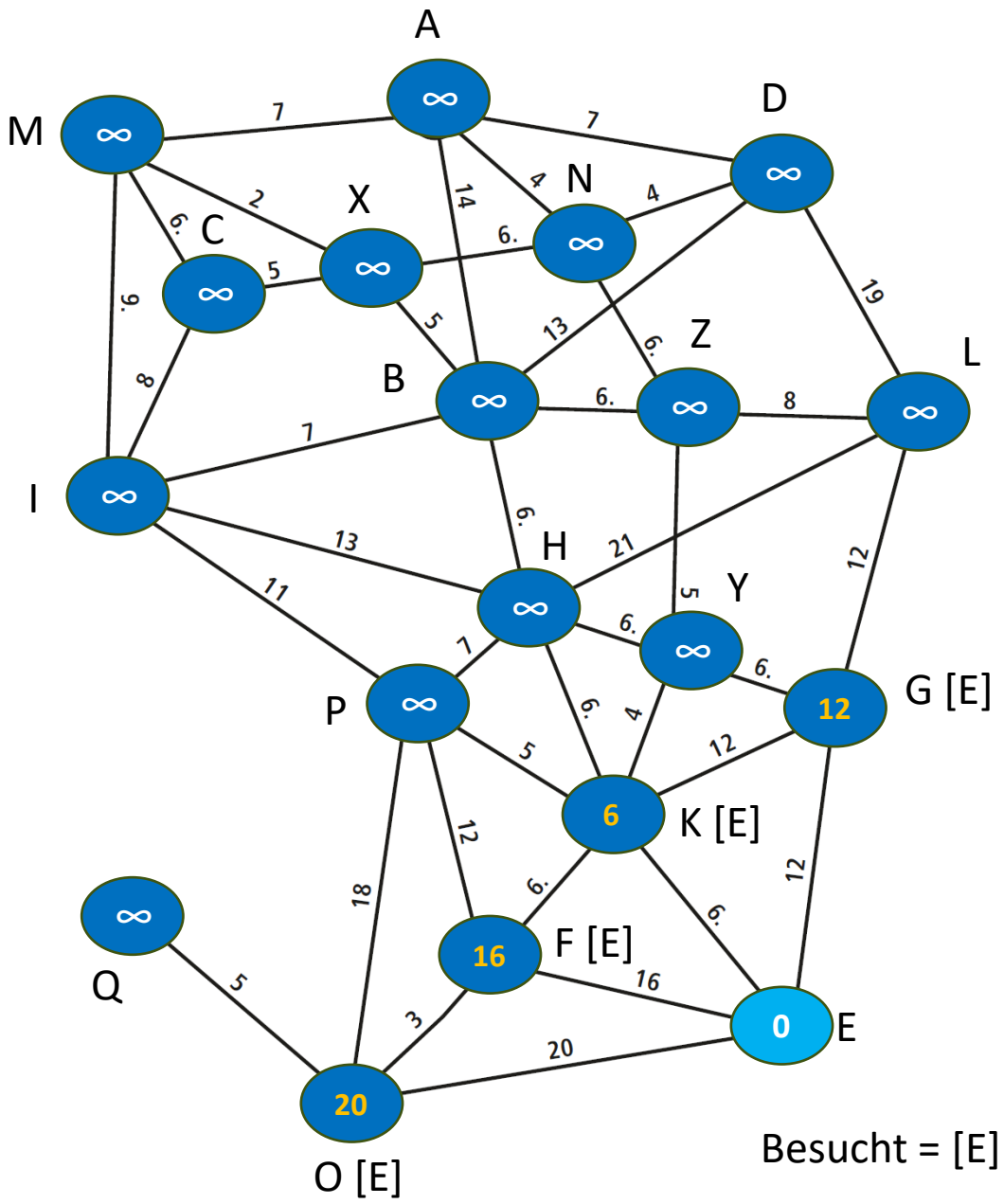
O -> O [E]
F -> F [E]
K -> K [E]
G -> G [E]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das machst du so lange, bis du den Zielknoten (M) erreicht hast oder keine Knoten mehr zu besuchen sind.

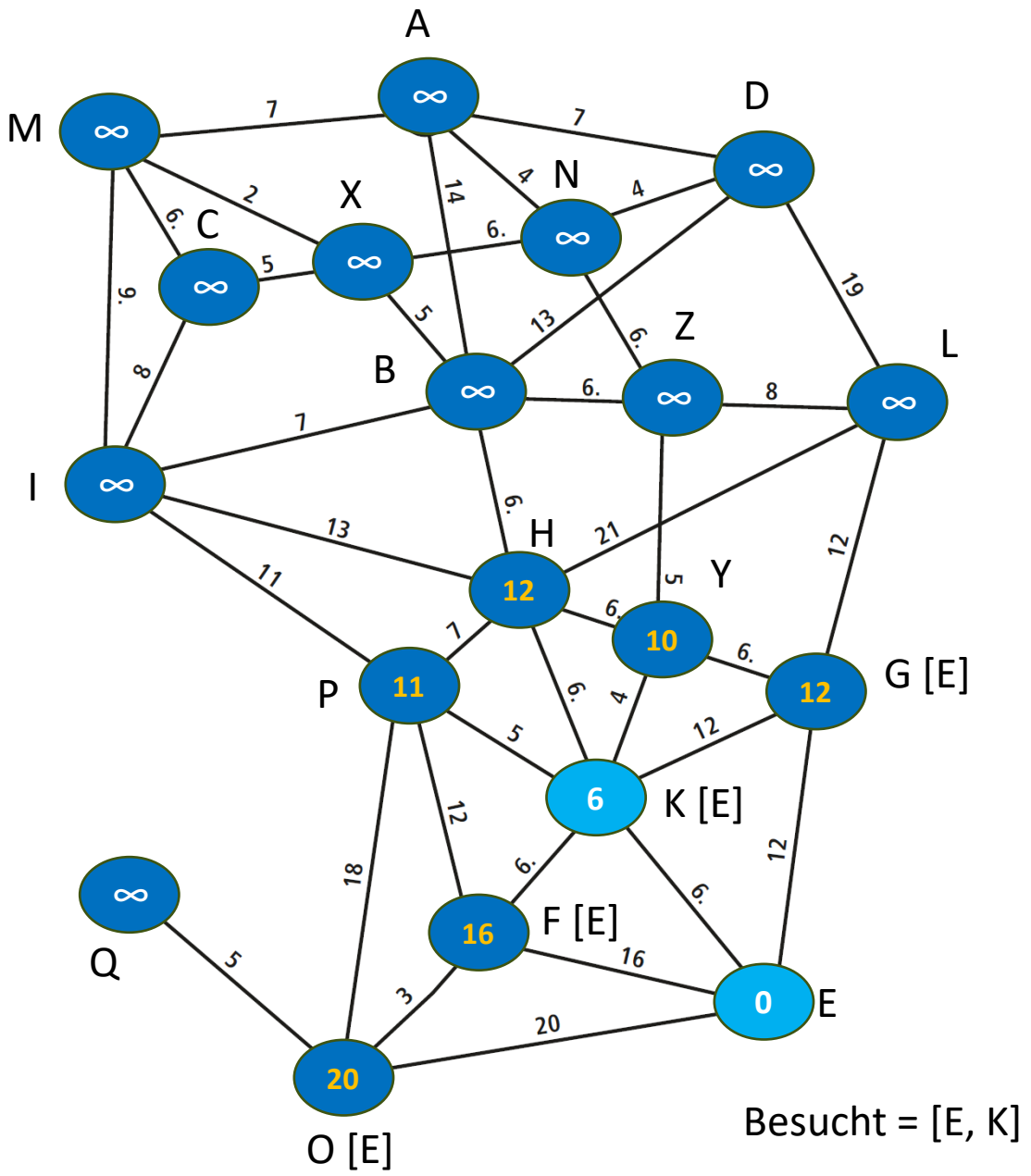


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge K zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Merke: das betrifft hier Y, H, P

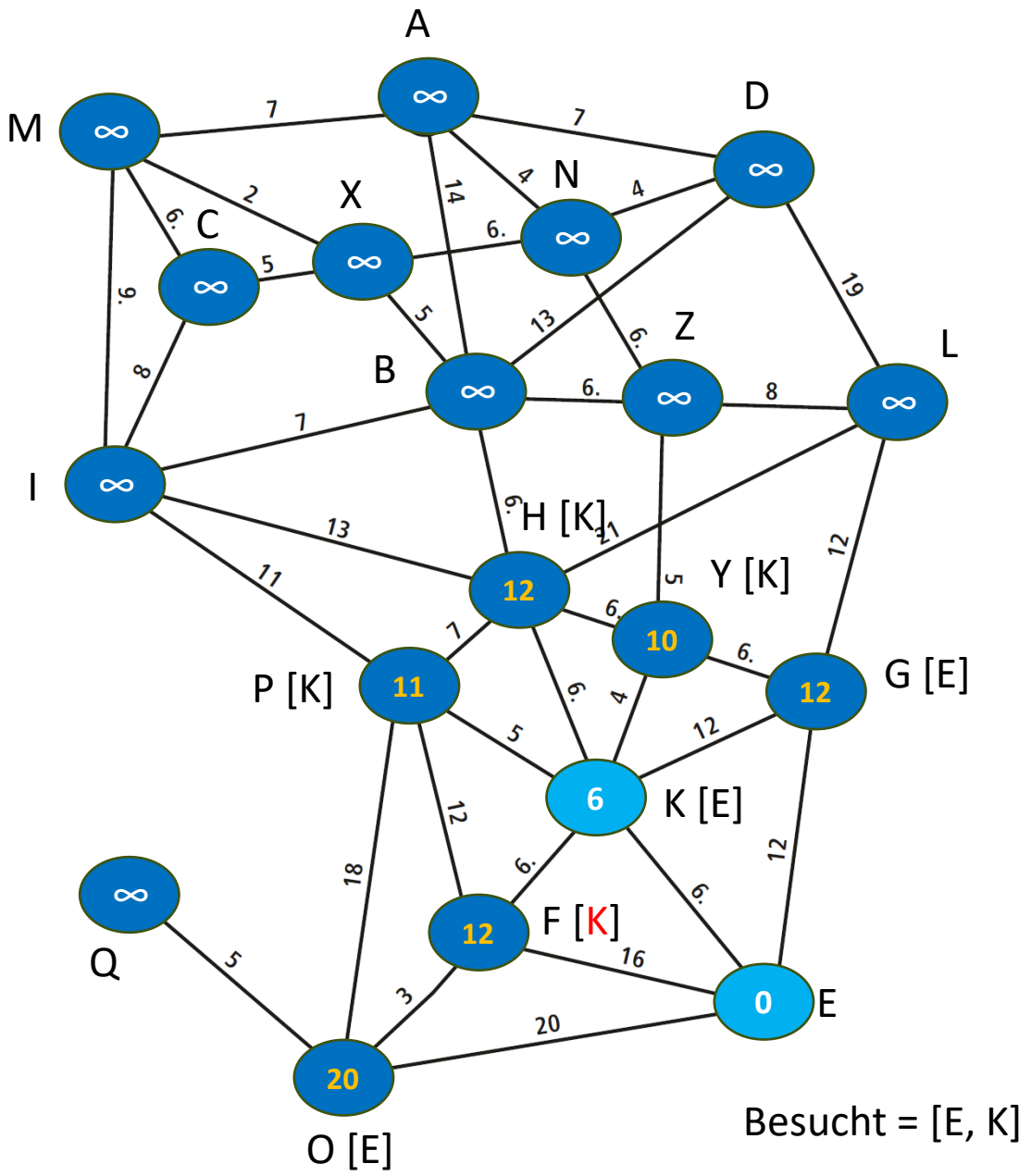


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

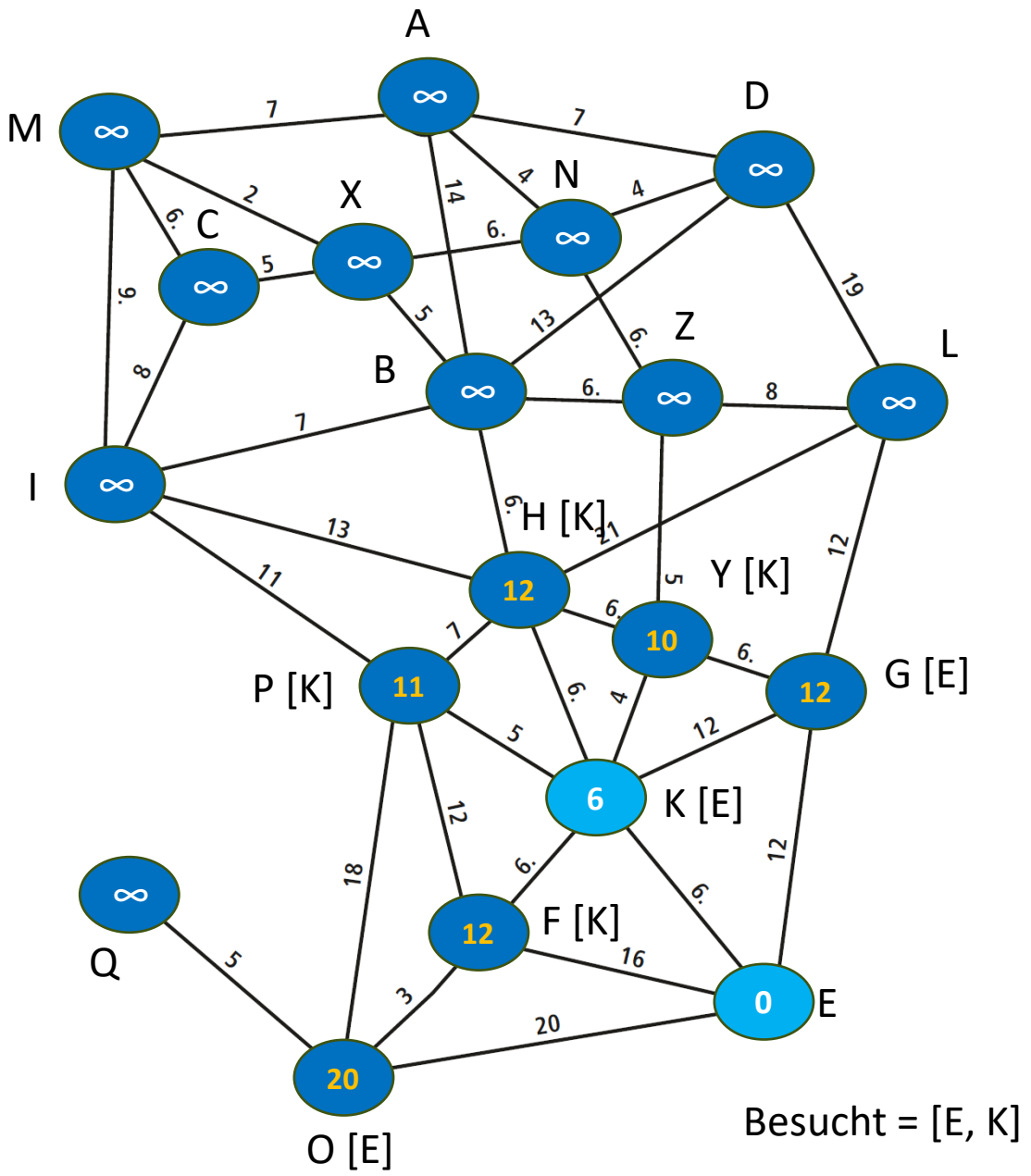
Y -> Y [K]
H -> H [K]
P -> P [K]
F [E] -> F [K]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das ist hier Knoten Y

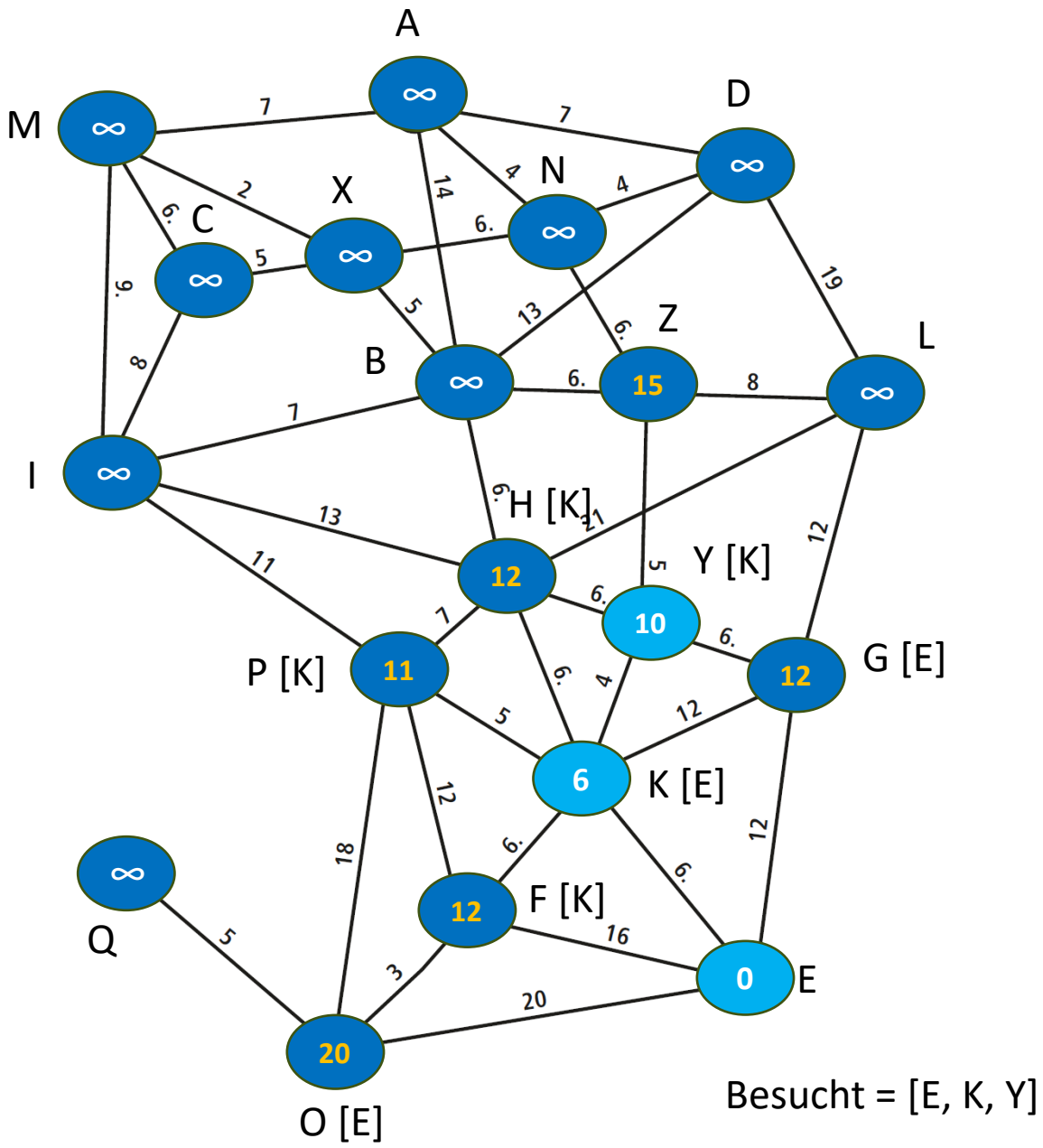


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge Y zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Merke: das betrifft hier Z

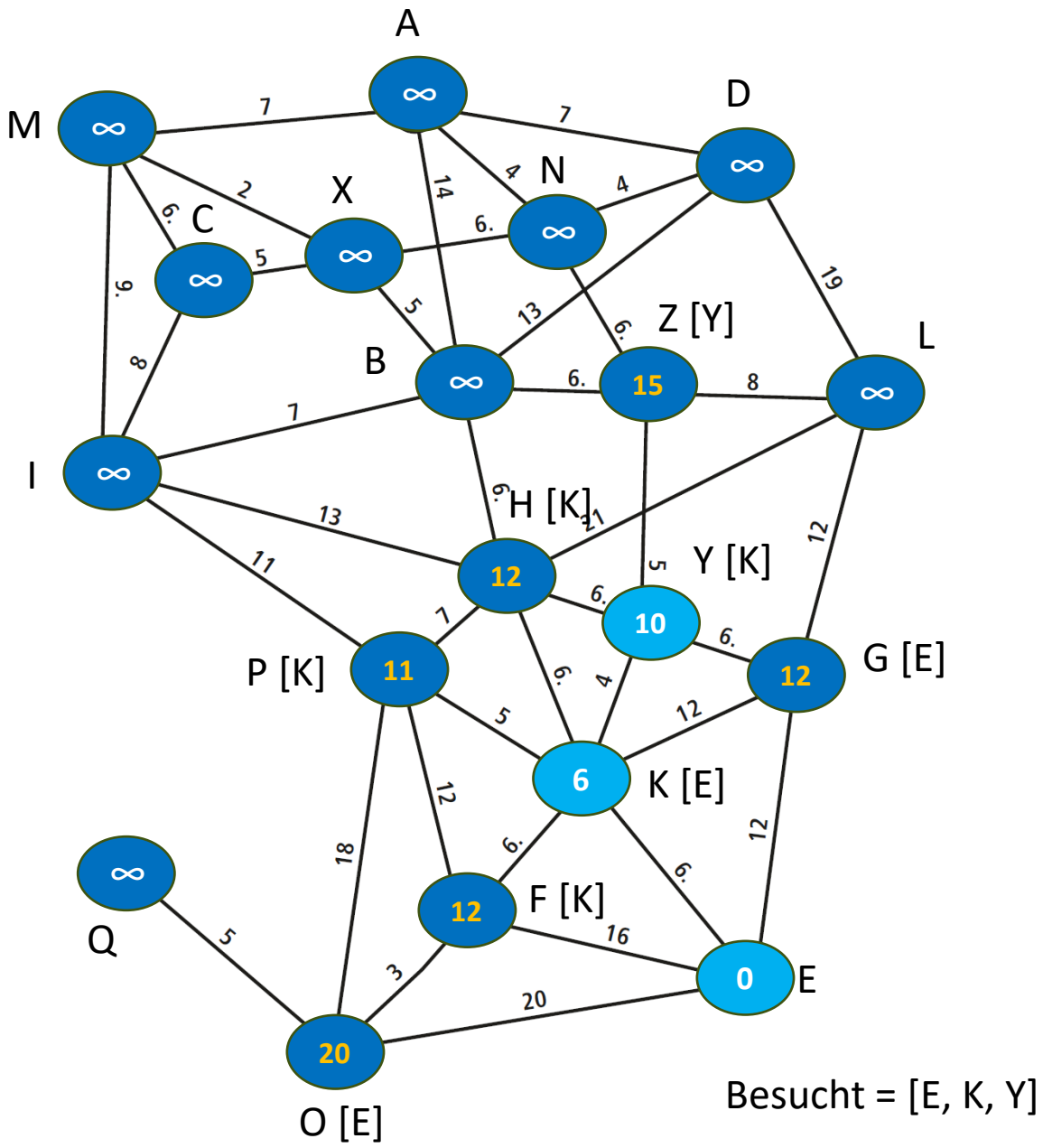


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

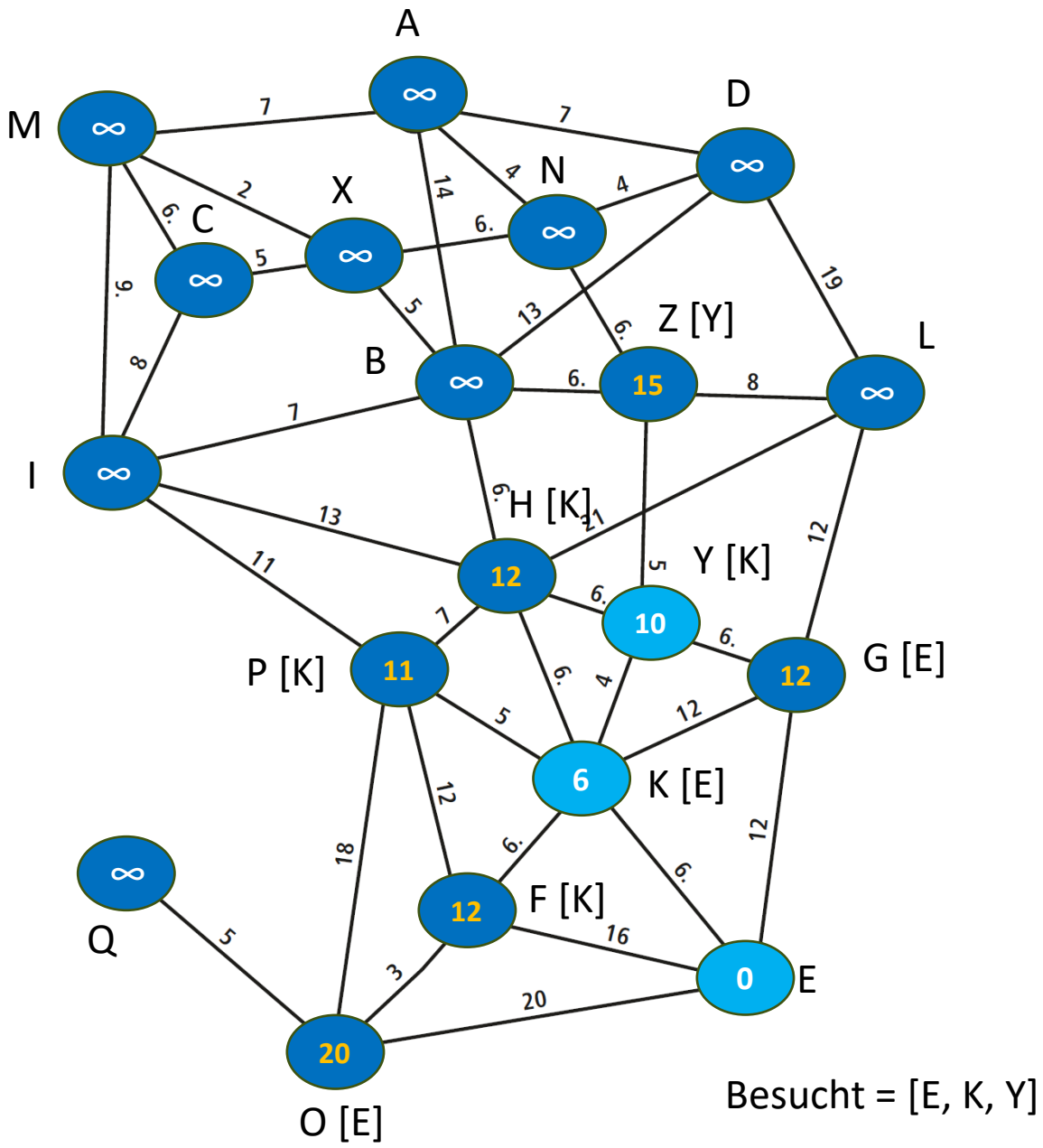
Z -> Z[Y]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das ist hier Knoten P

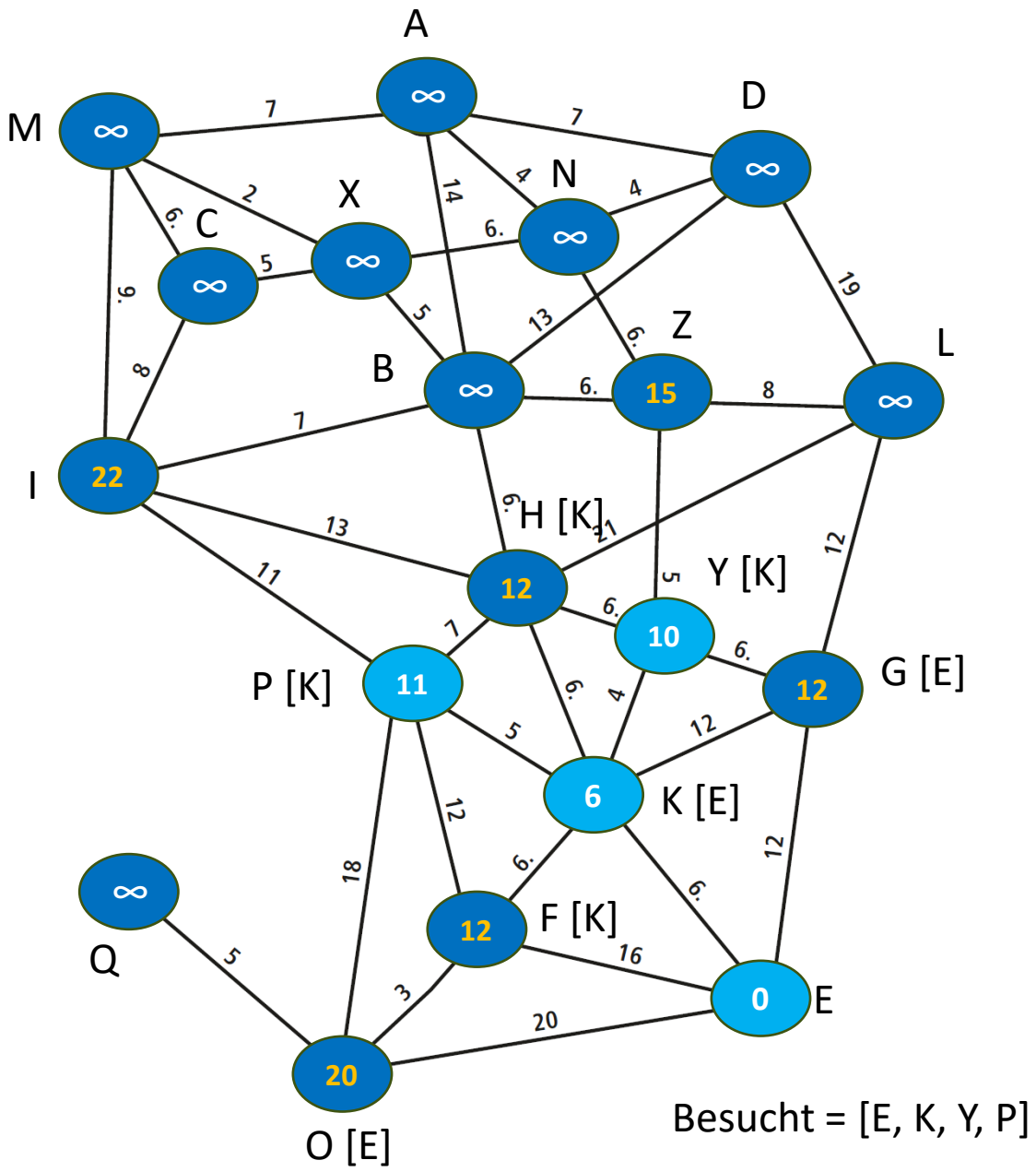


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge P zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Merke: das betrifft hier I

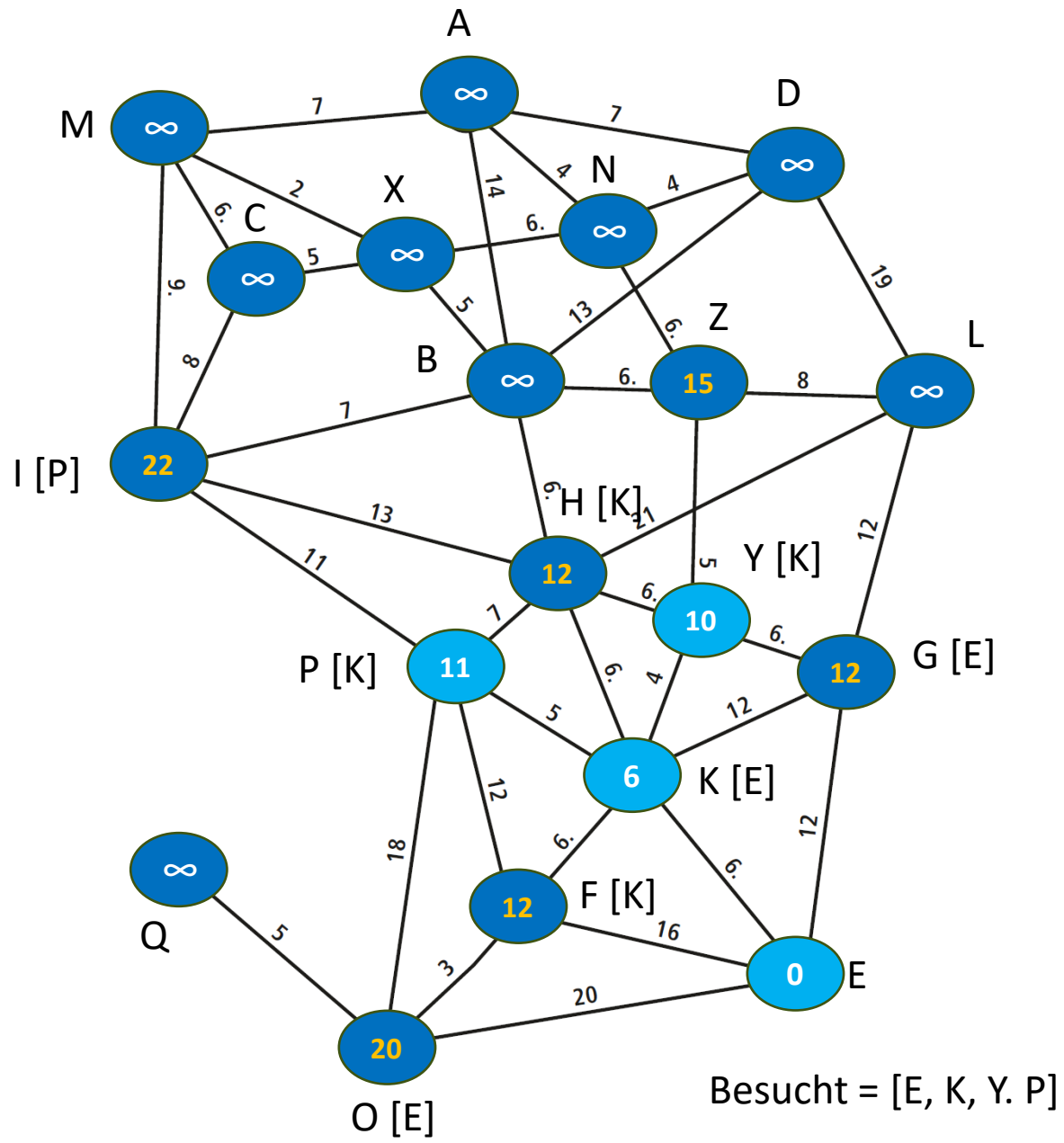


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

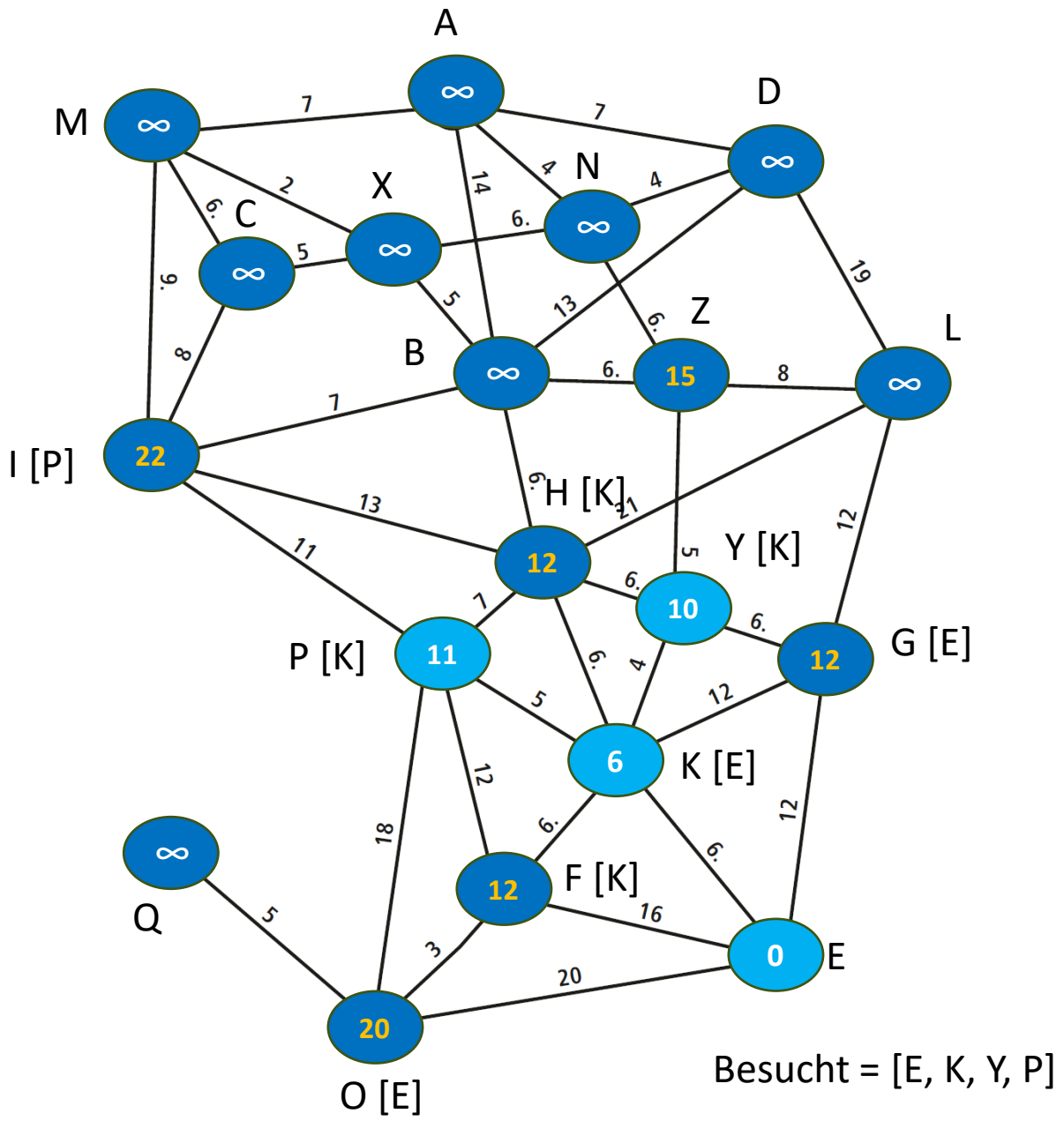
I -> I[P]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

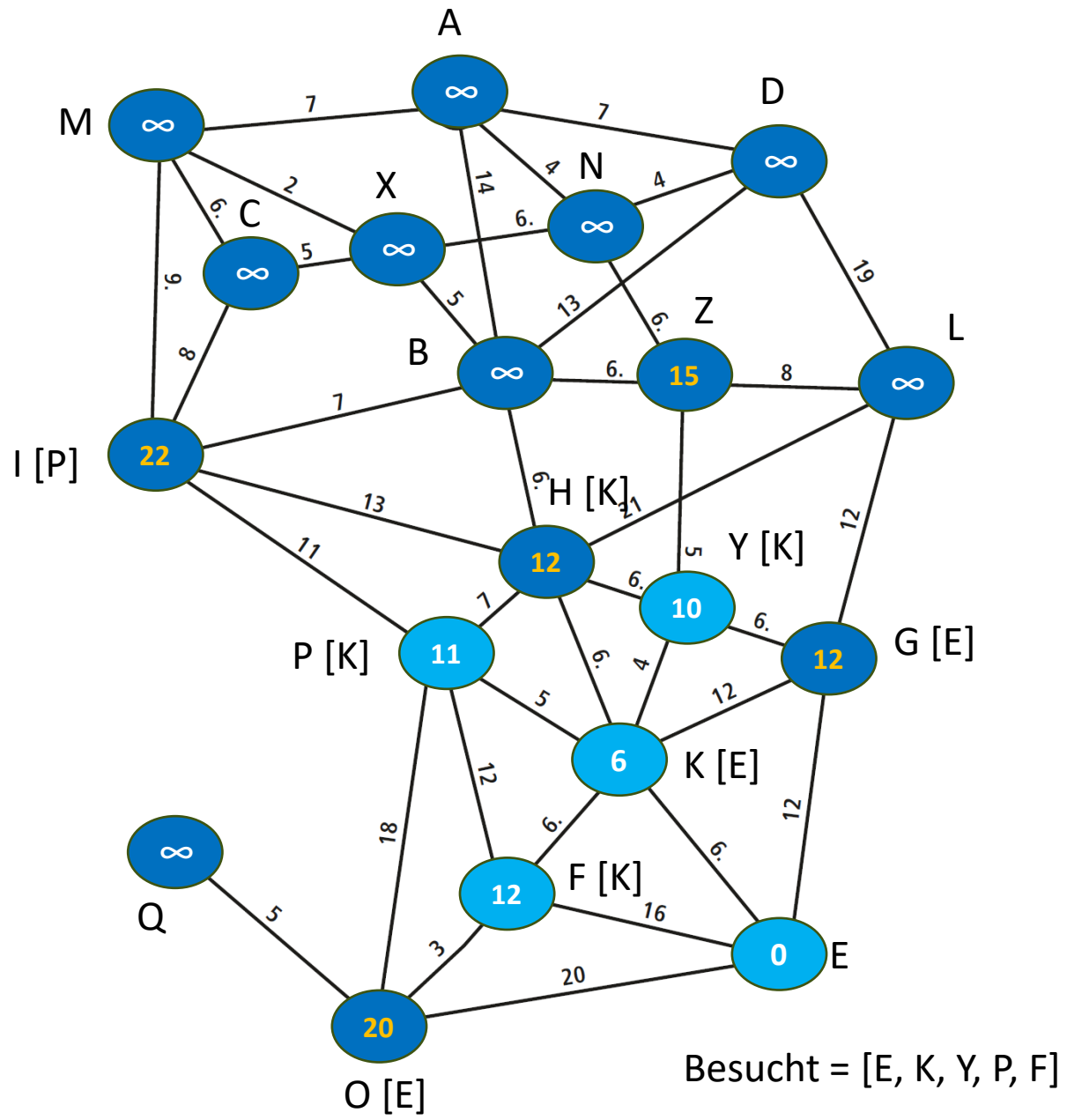
Das ist hier Knoten F (oder G oder H)



**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge F zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

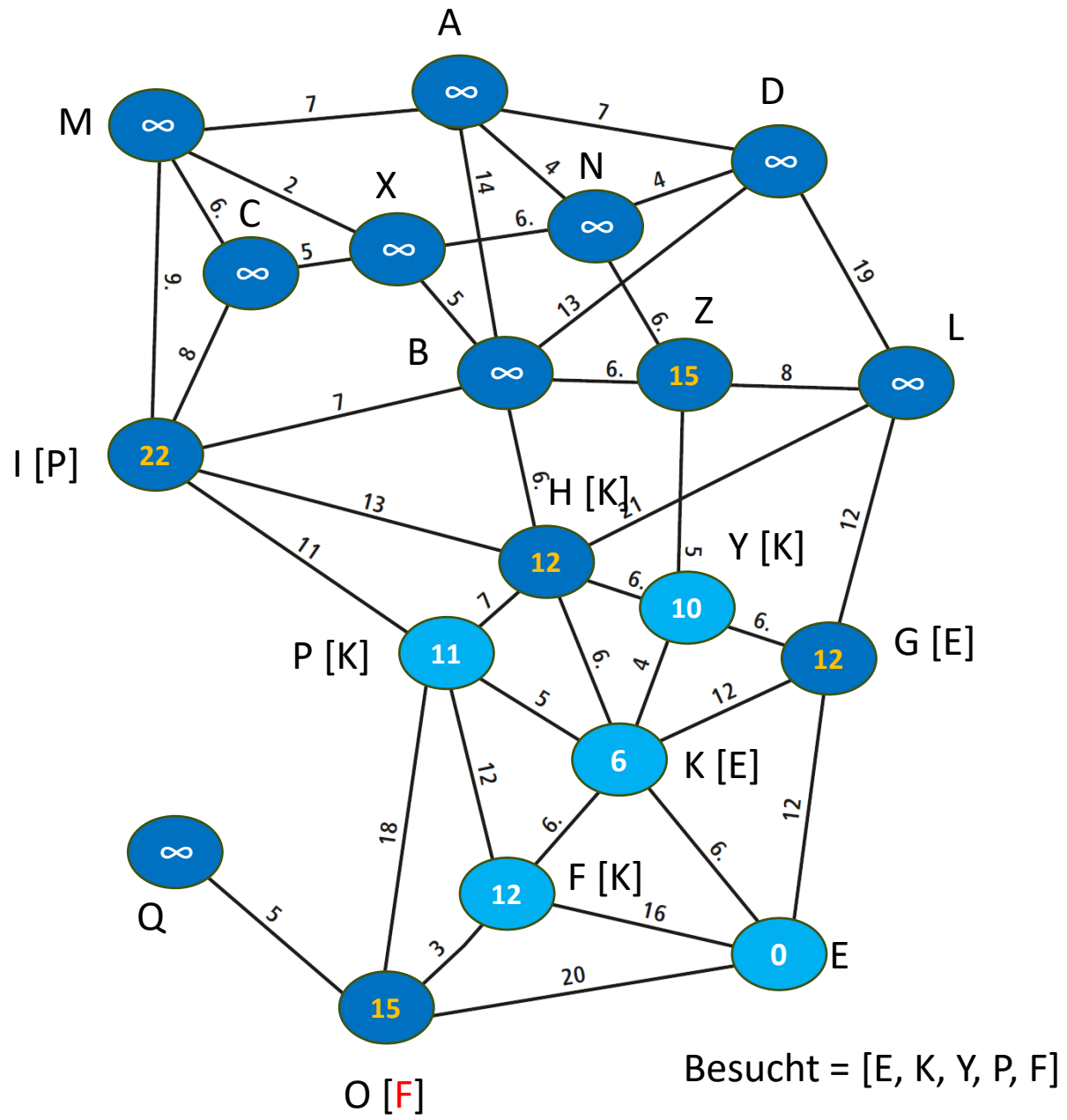


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

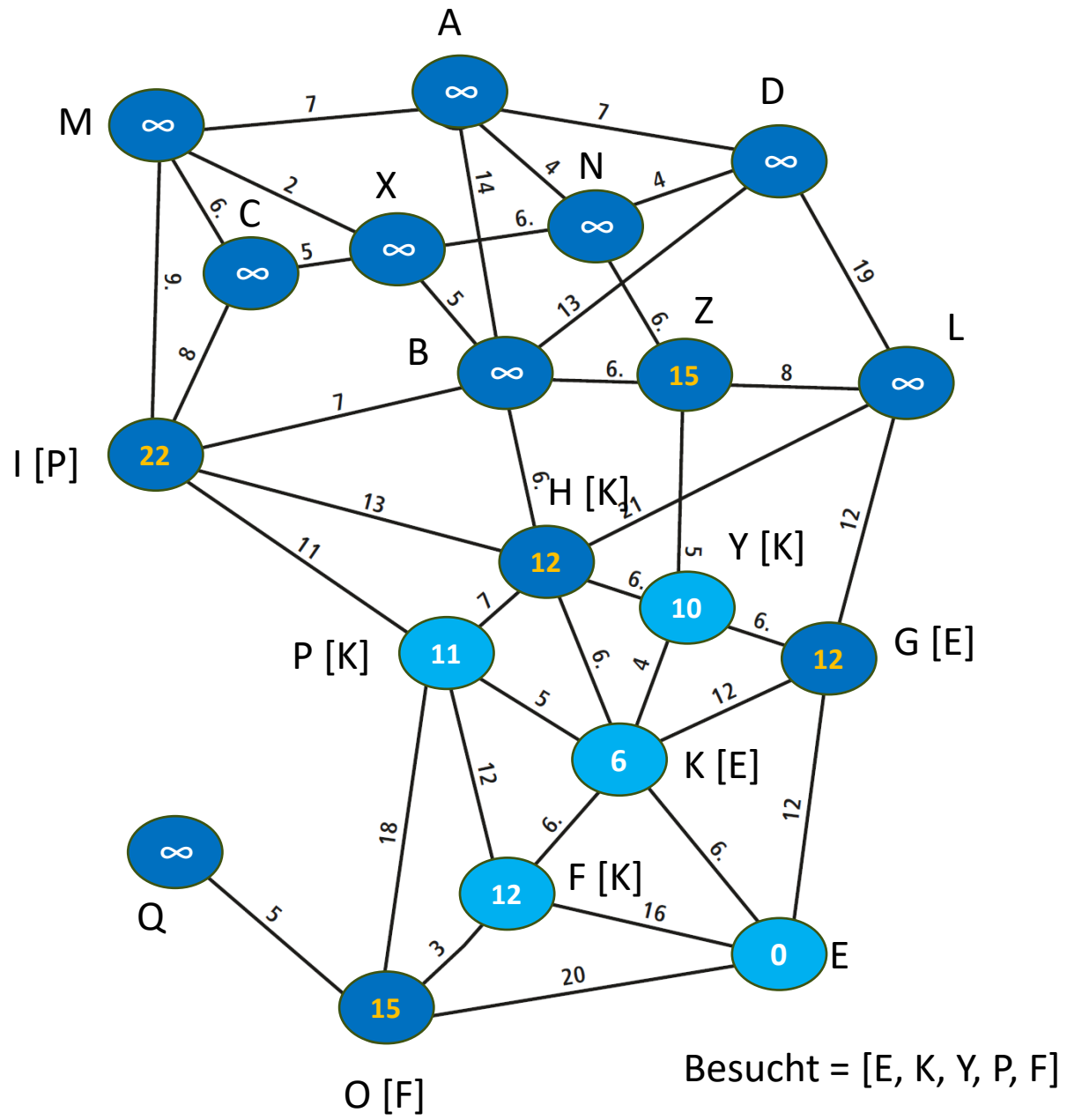
O [E] -> O[F]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das ist hier Knoten G (oder H)

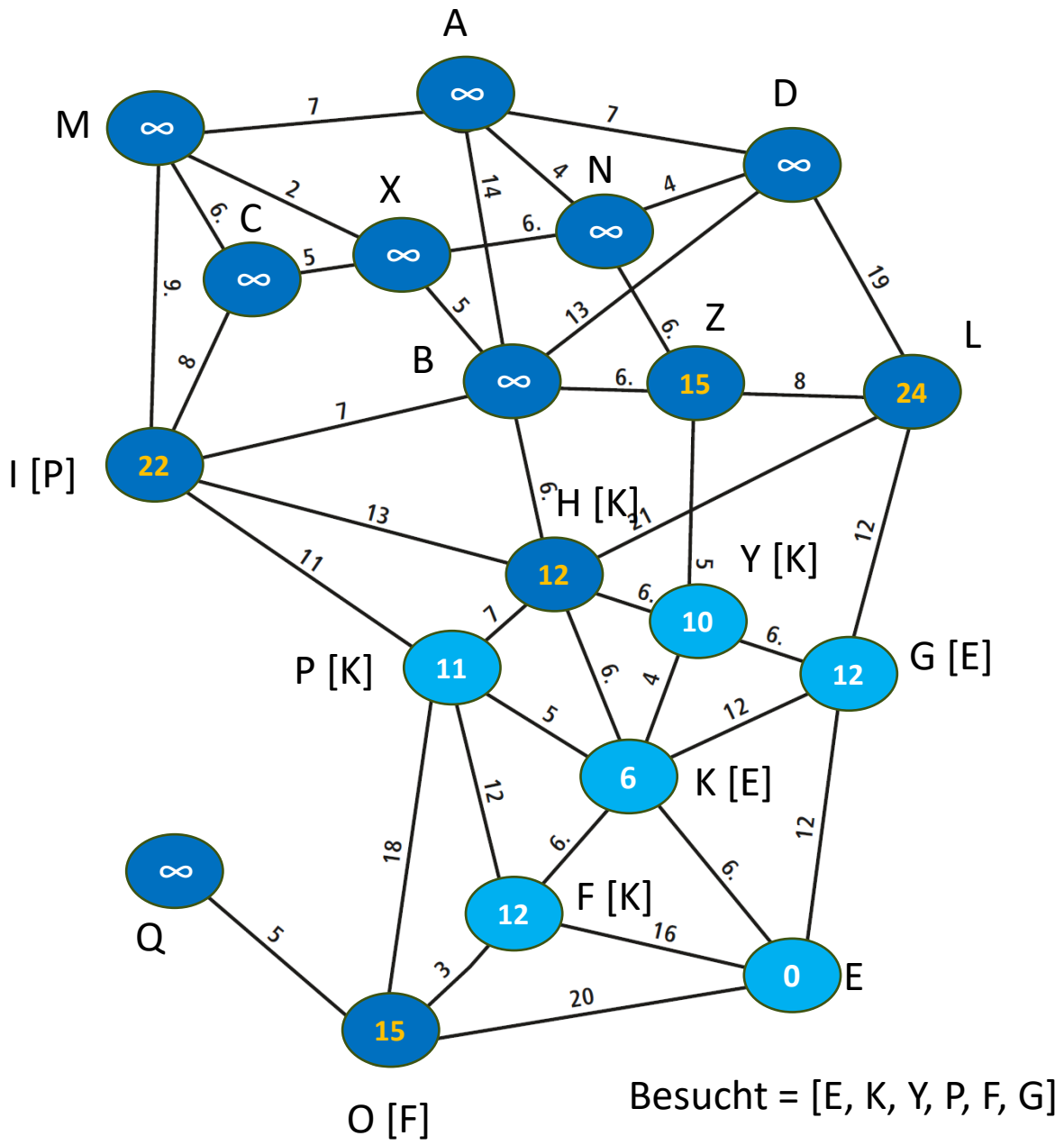


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge G zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Hier: L

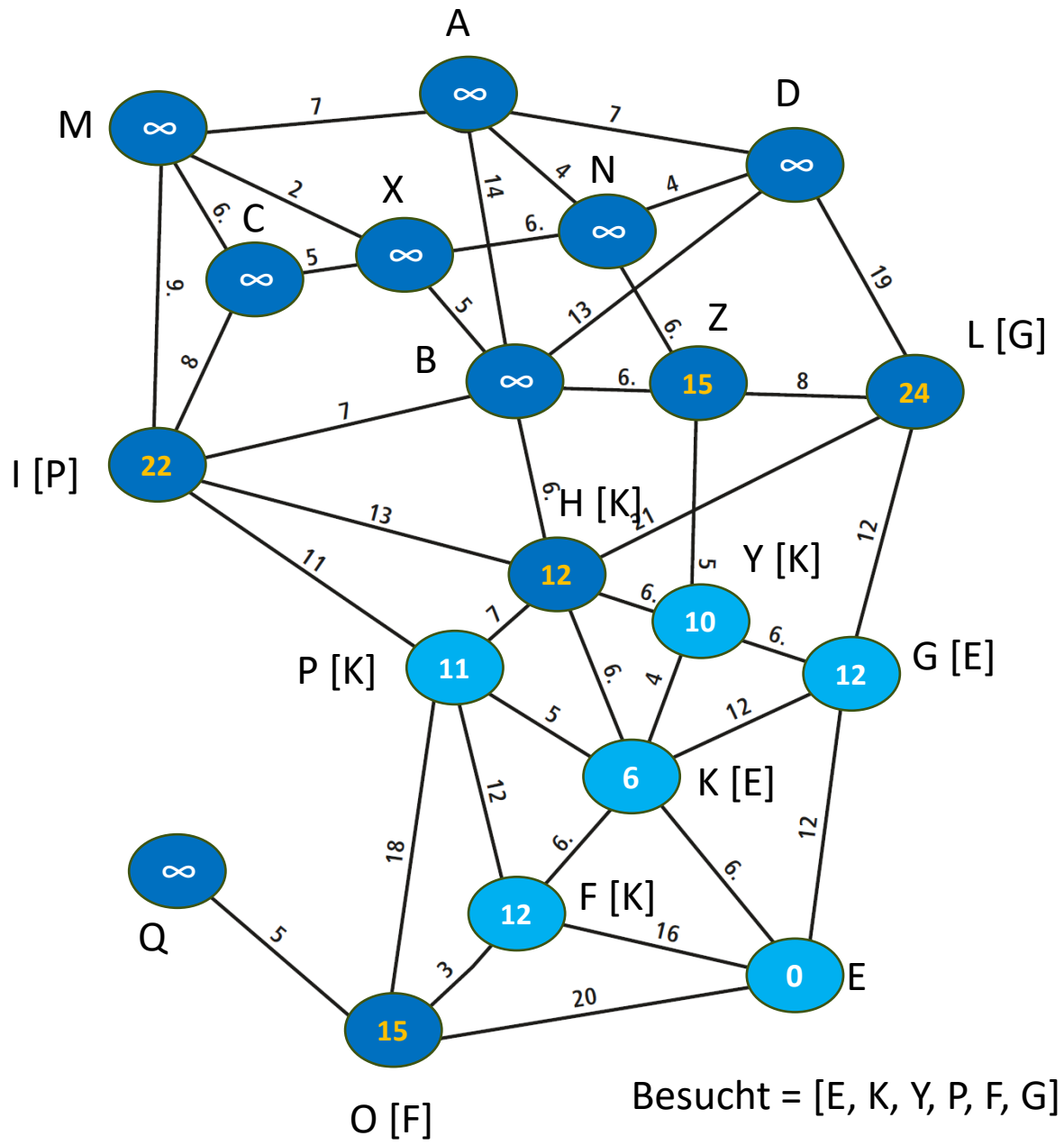


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

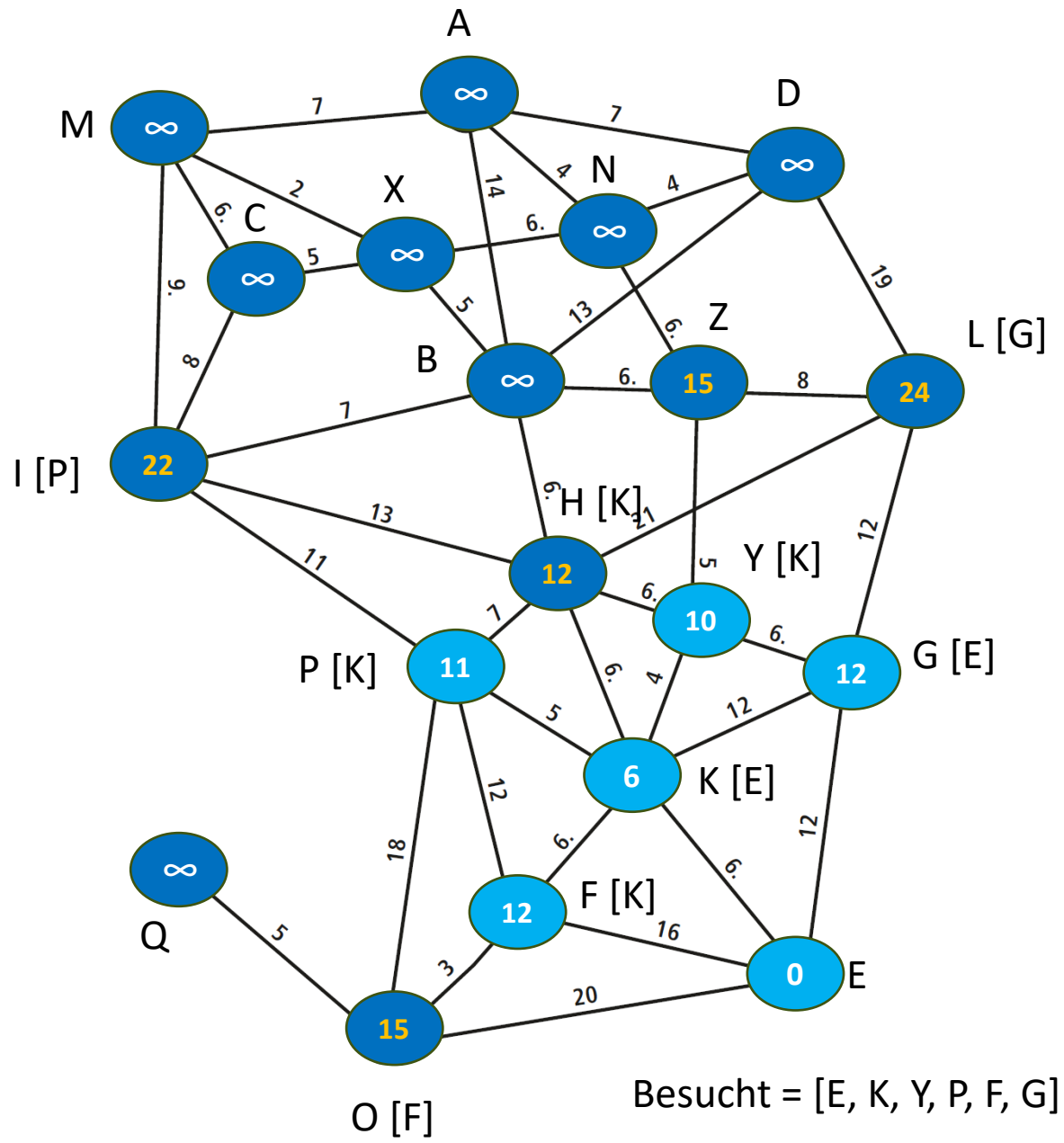
Hier: L -> L [G]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das ist hier Knoten H

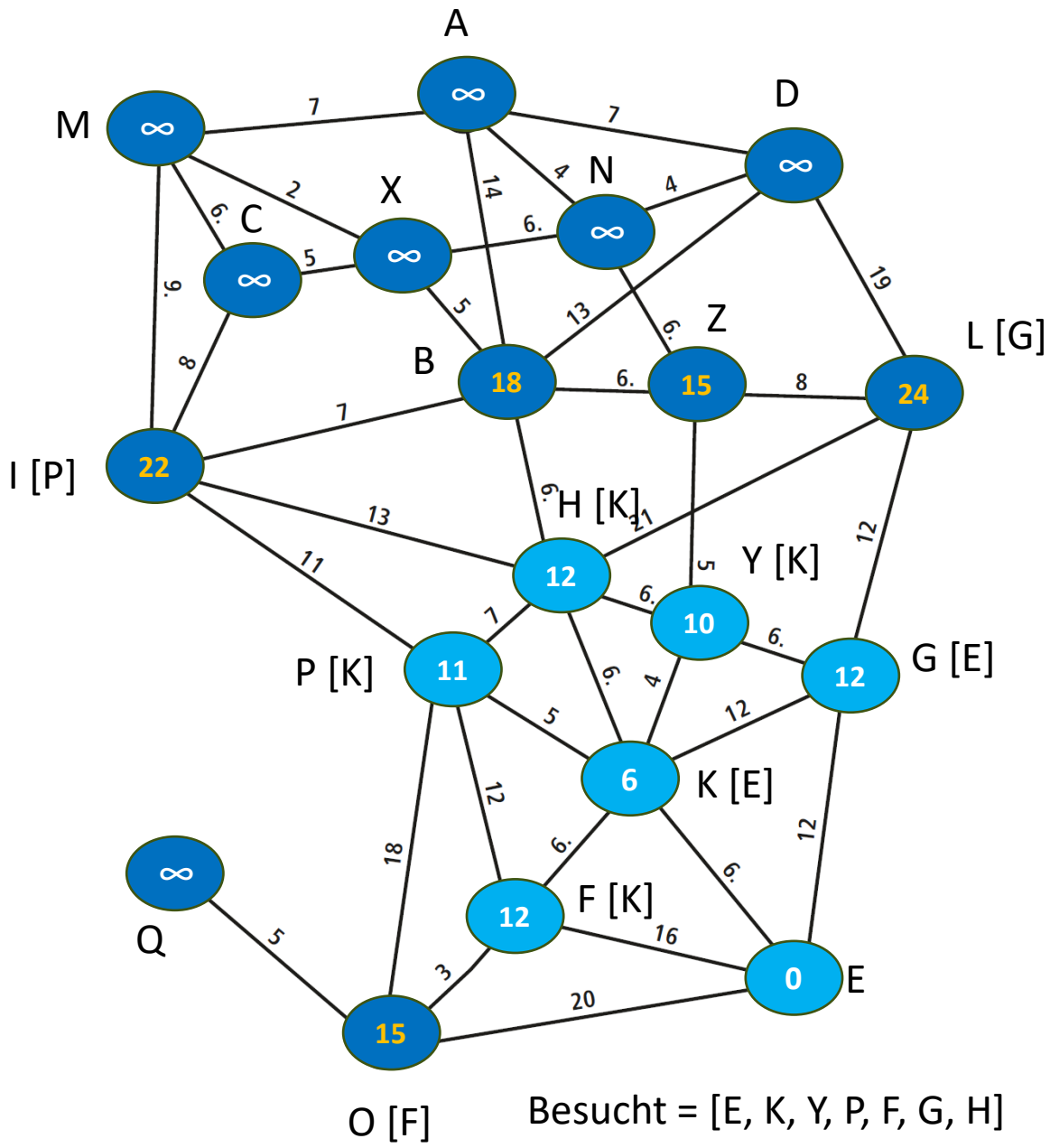


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge H zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Hier: B

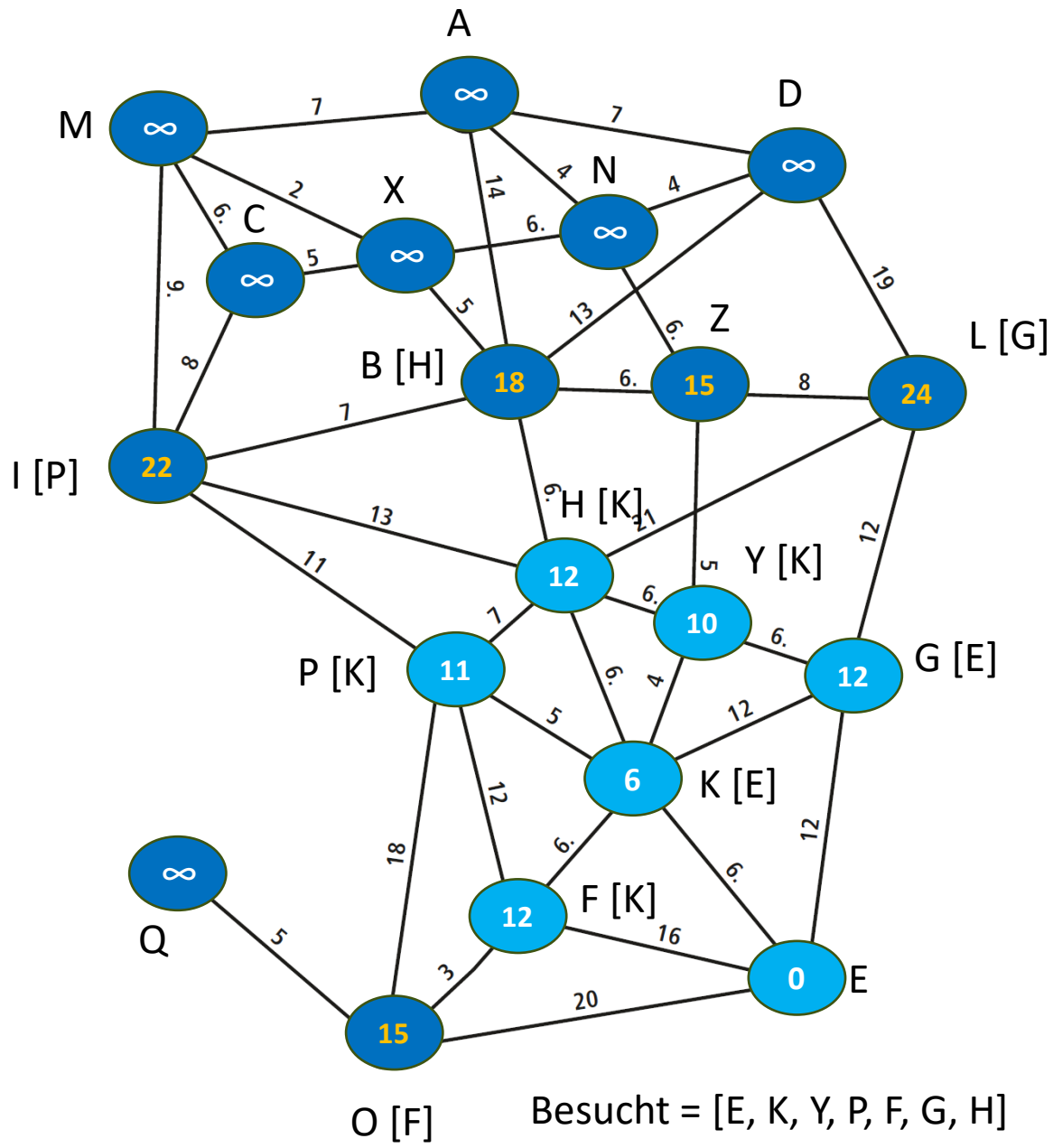


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

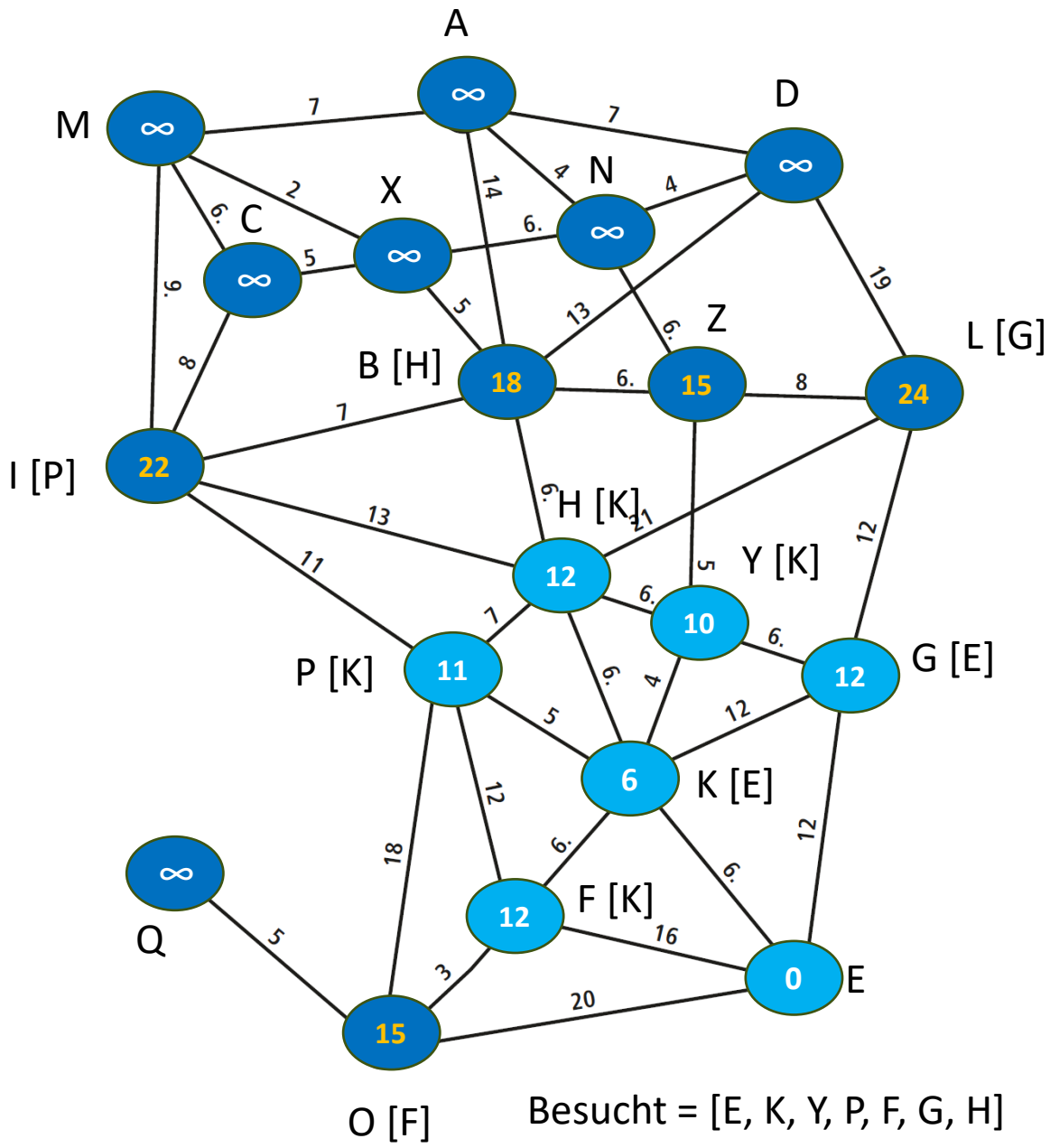
Hier: B -> B [H]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das ist hier Knoten O (oder Z)

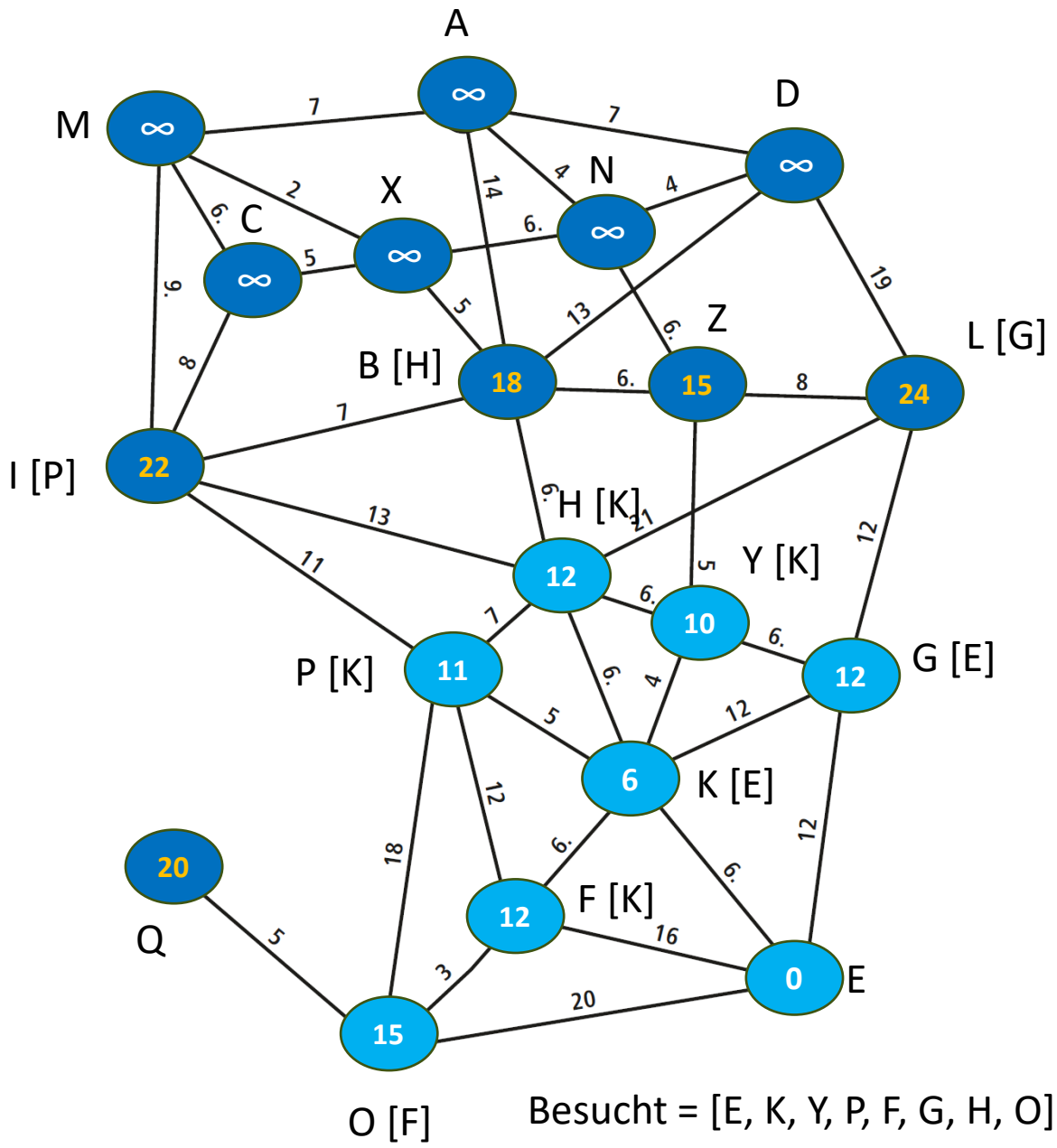


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge O zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Hier: Q

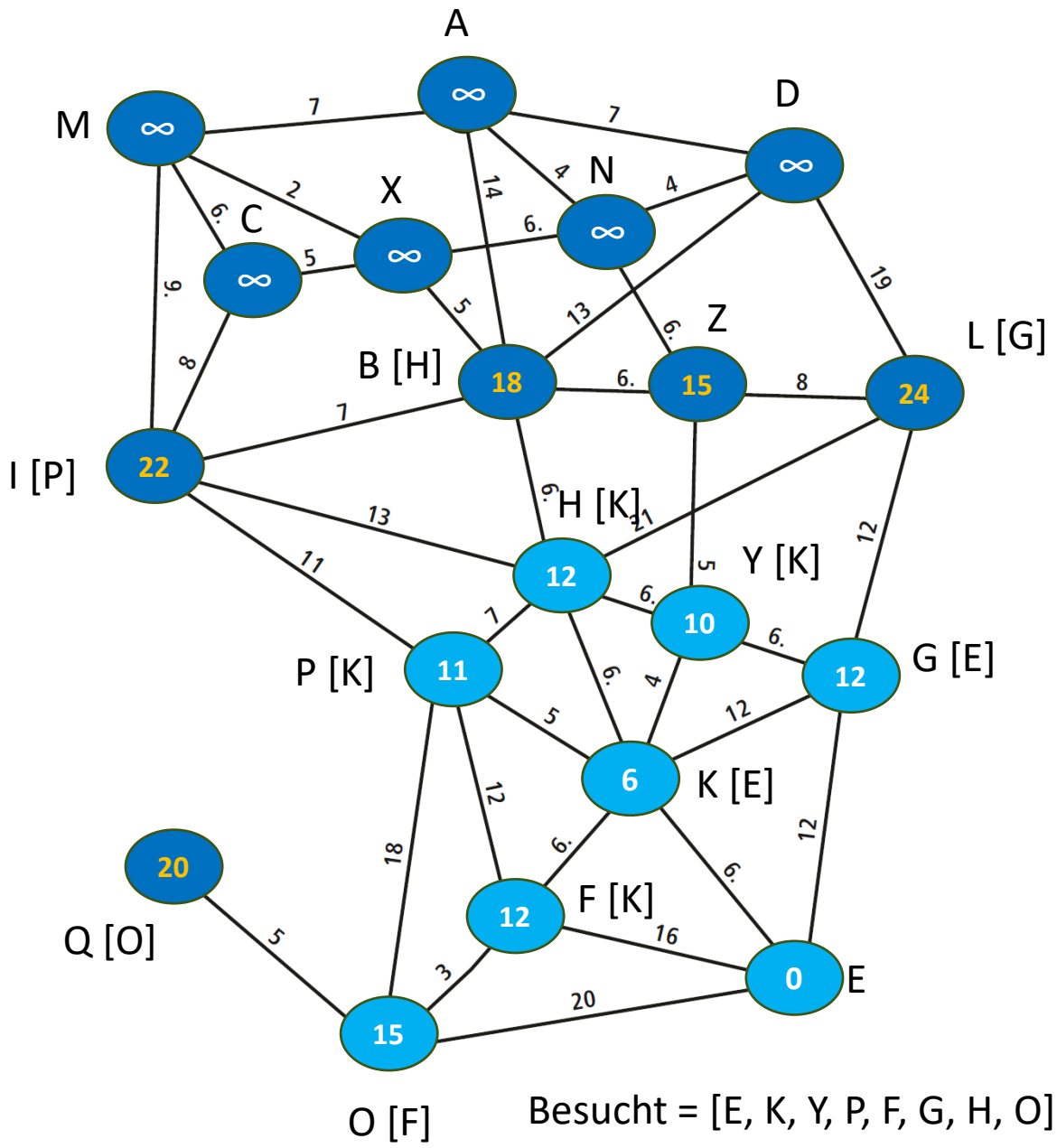


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

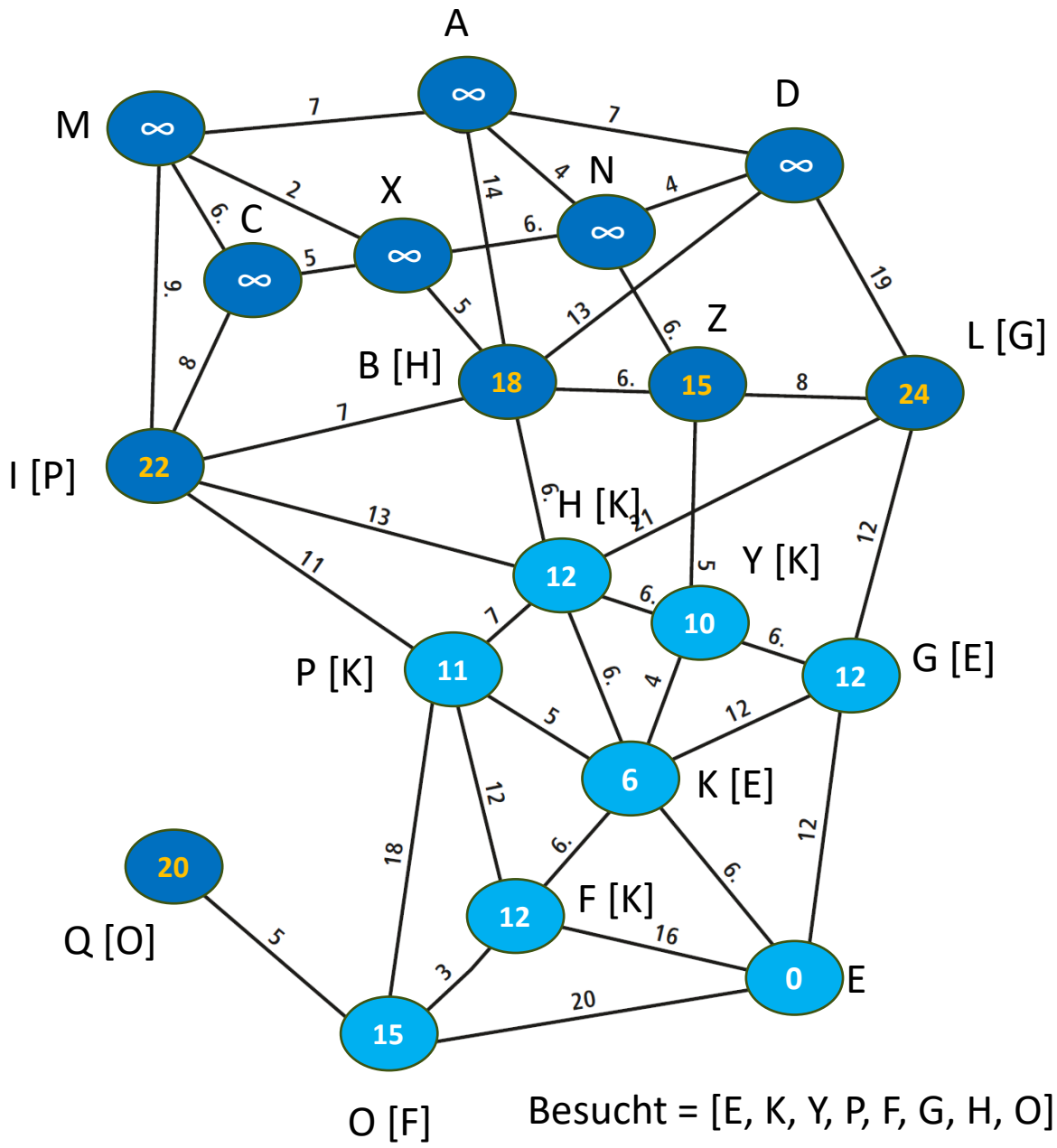
Hier: Q -> Q[O]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das ist hier Knoten Z

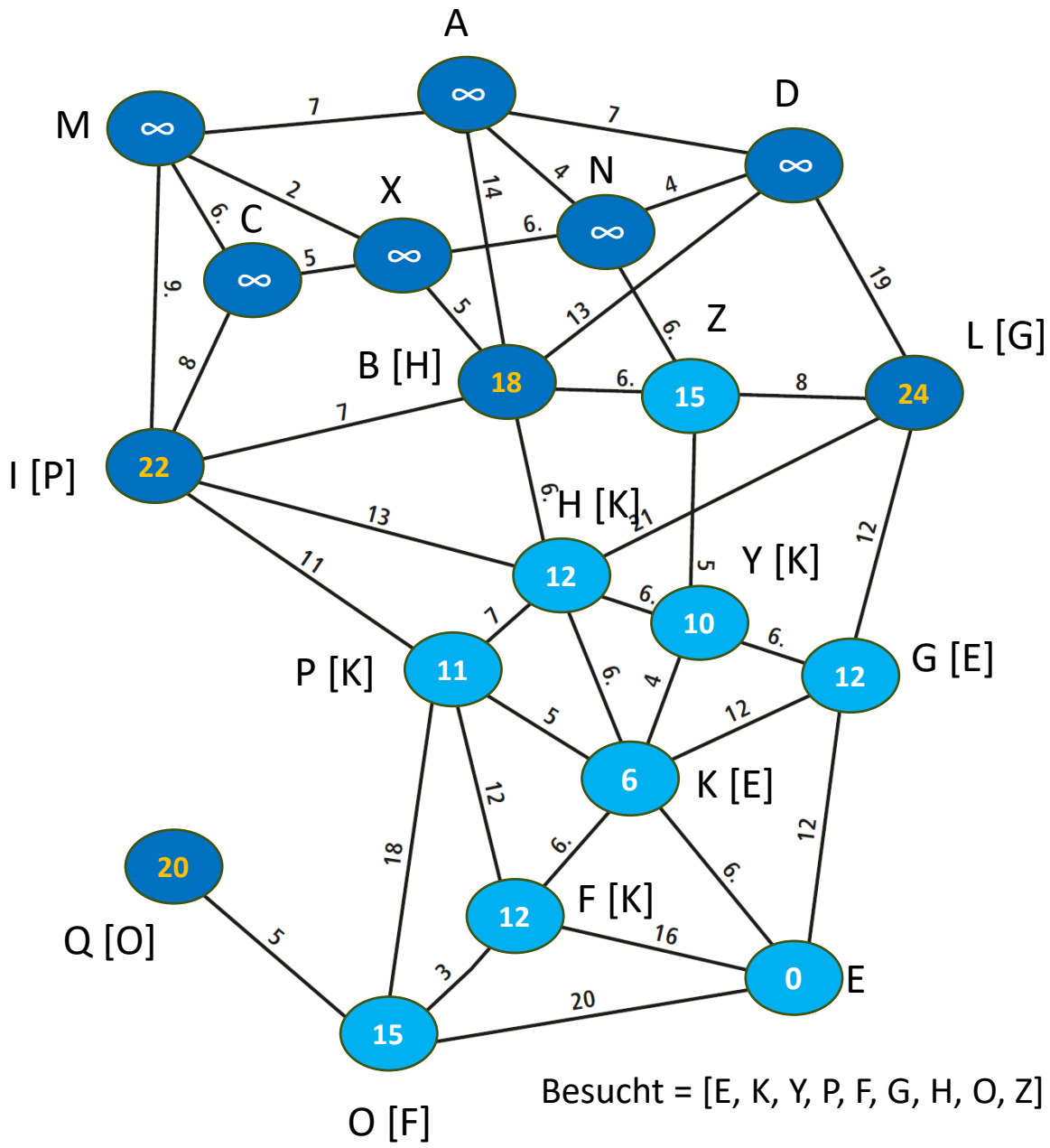


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge Z zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Hier: N

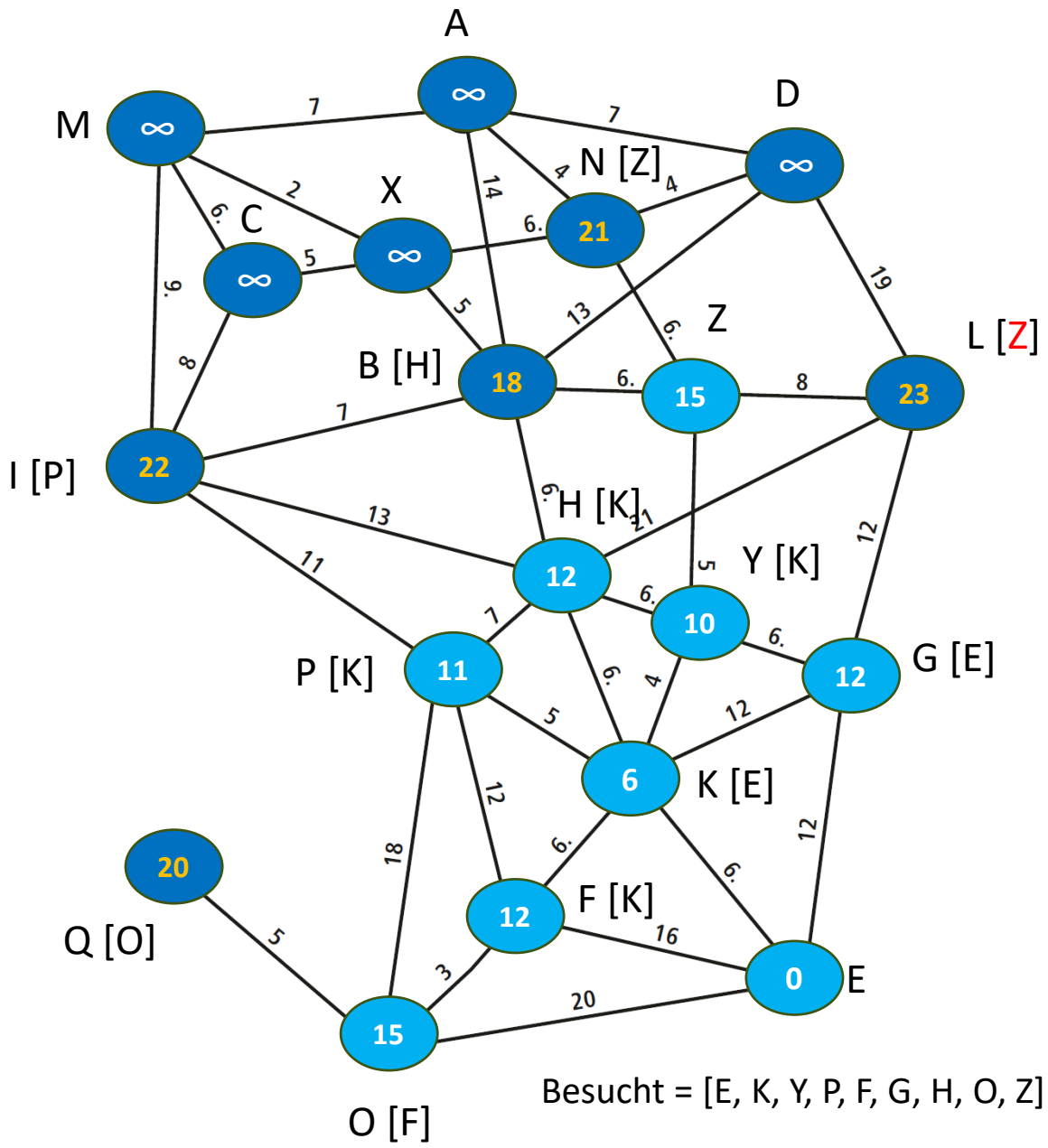


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

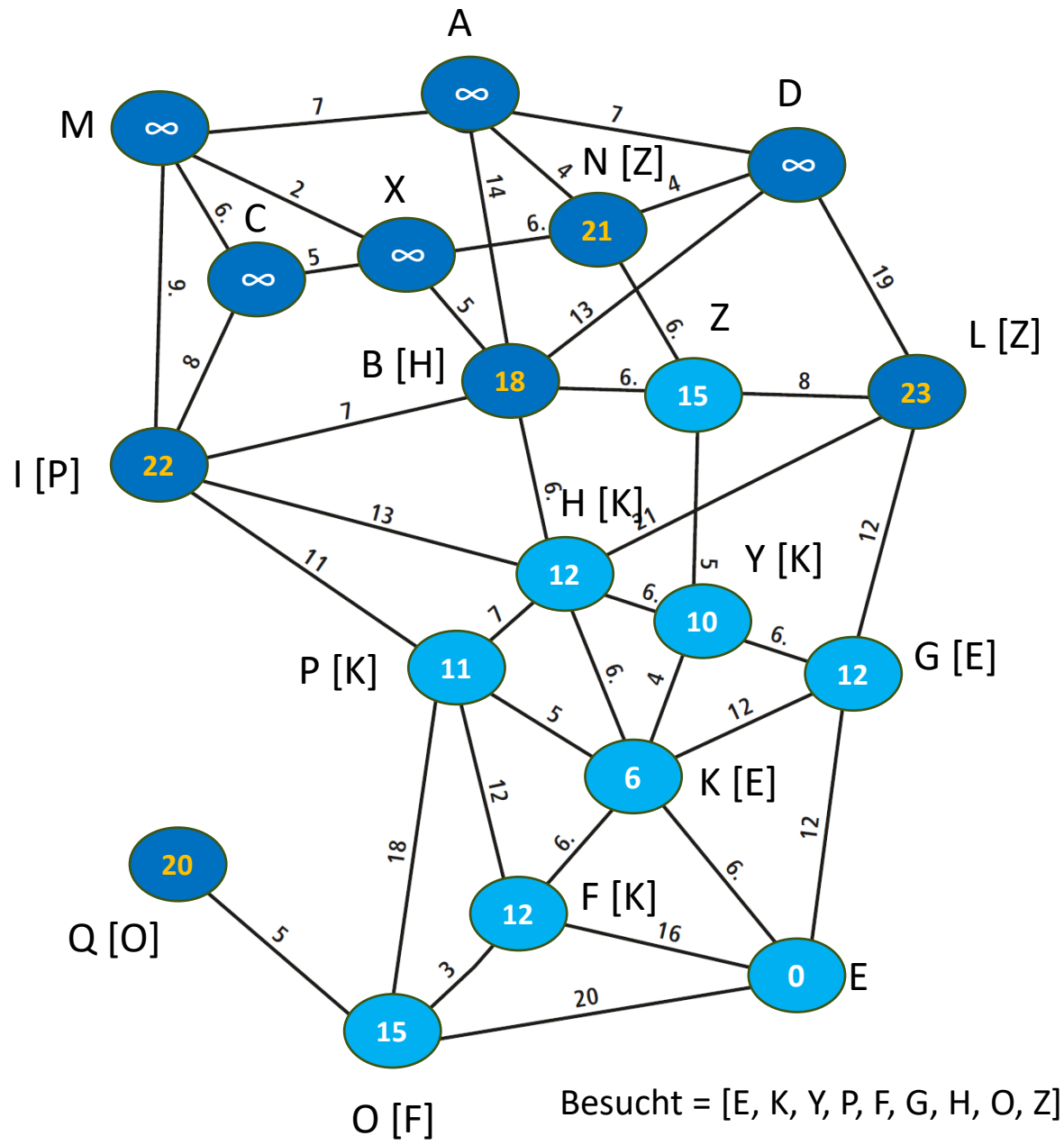
Hier:
N -> N[Z]
L -> L[Z]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das ist hier Knoten B

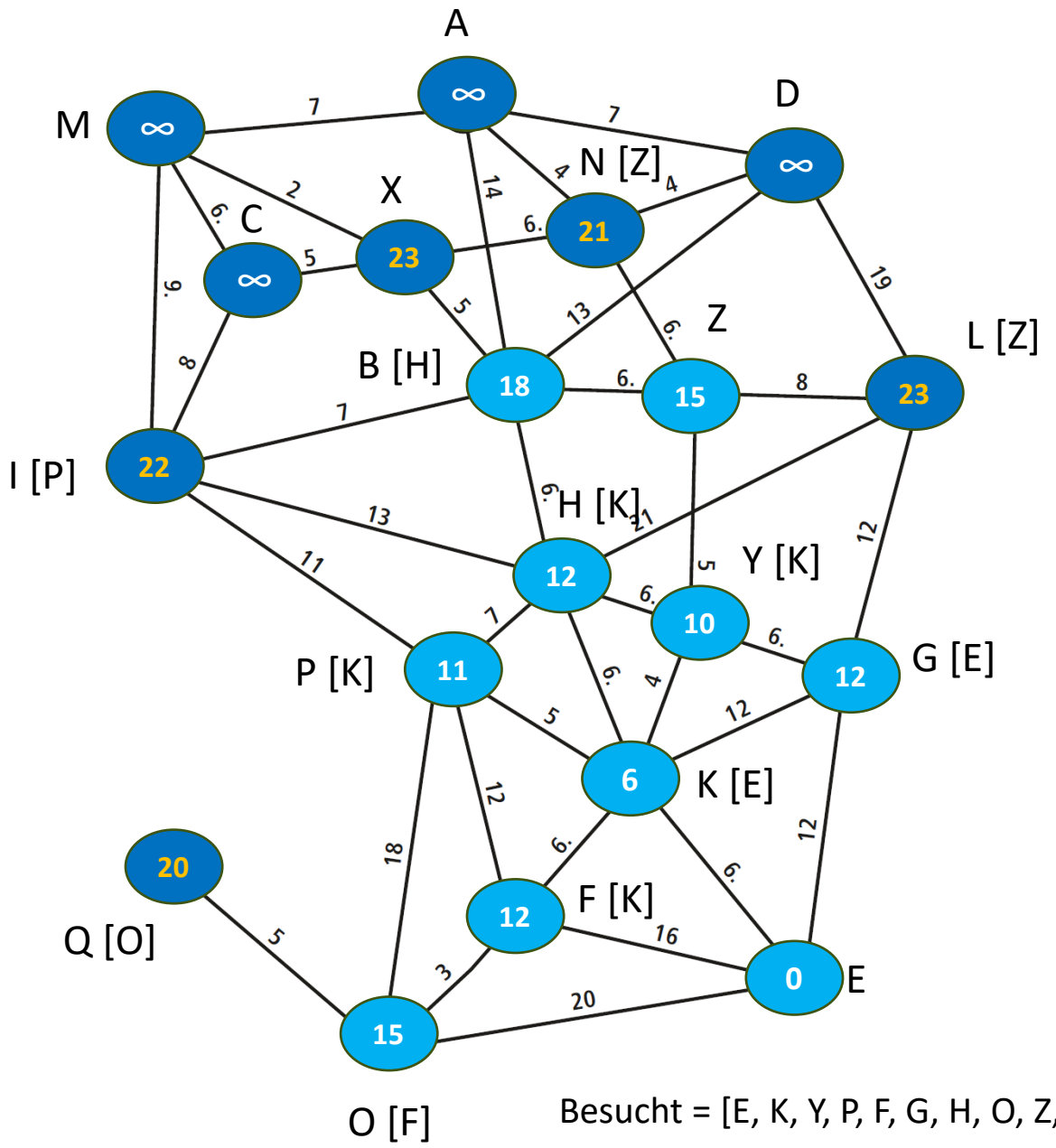


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge B zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Hier: X

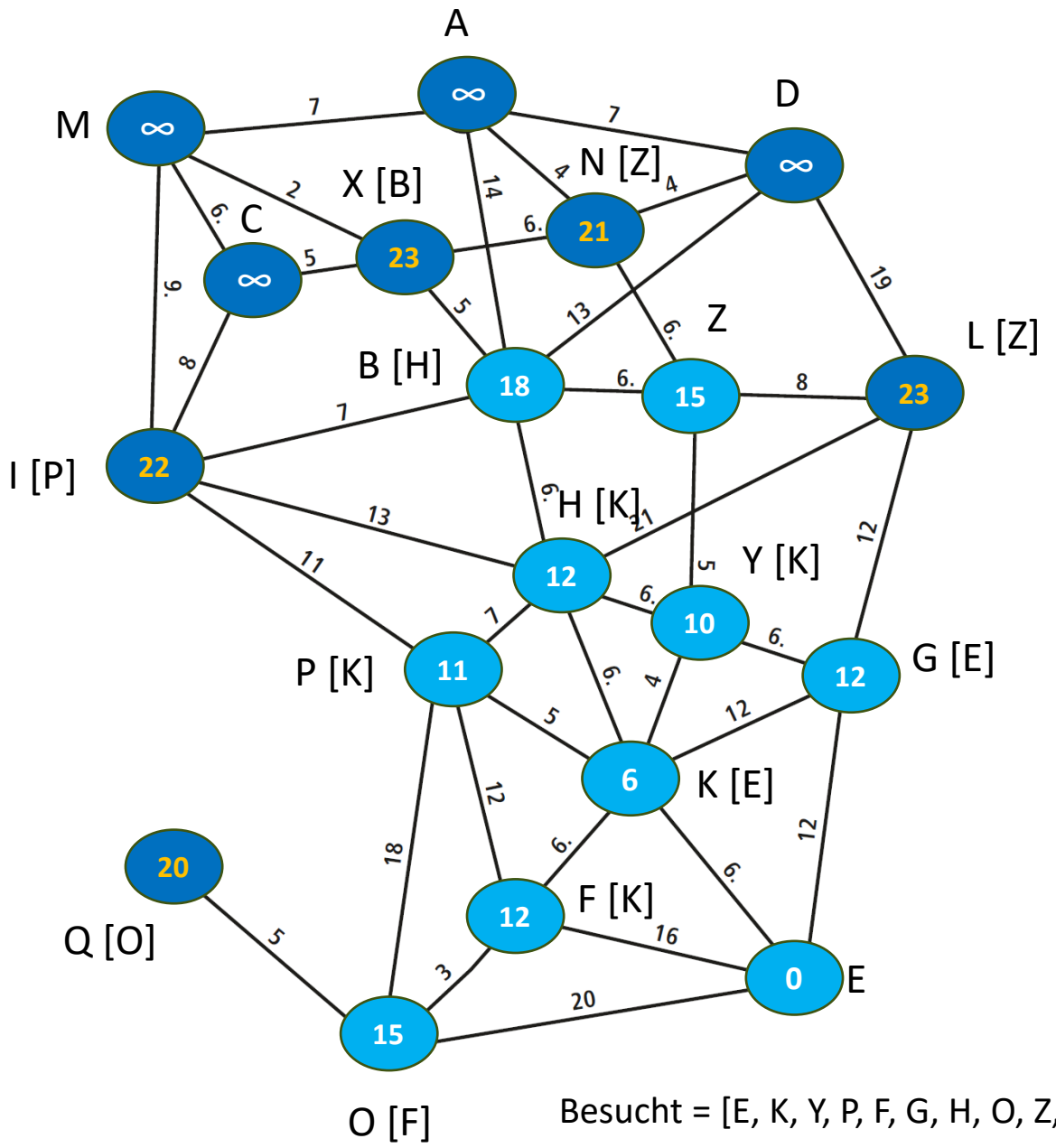


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

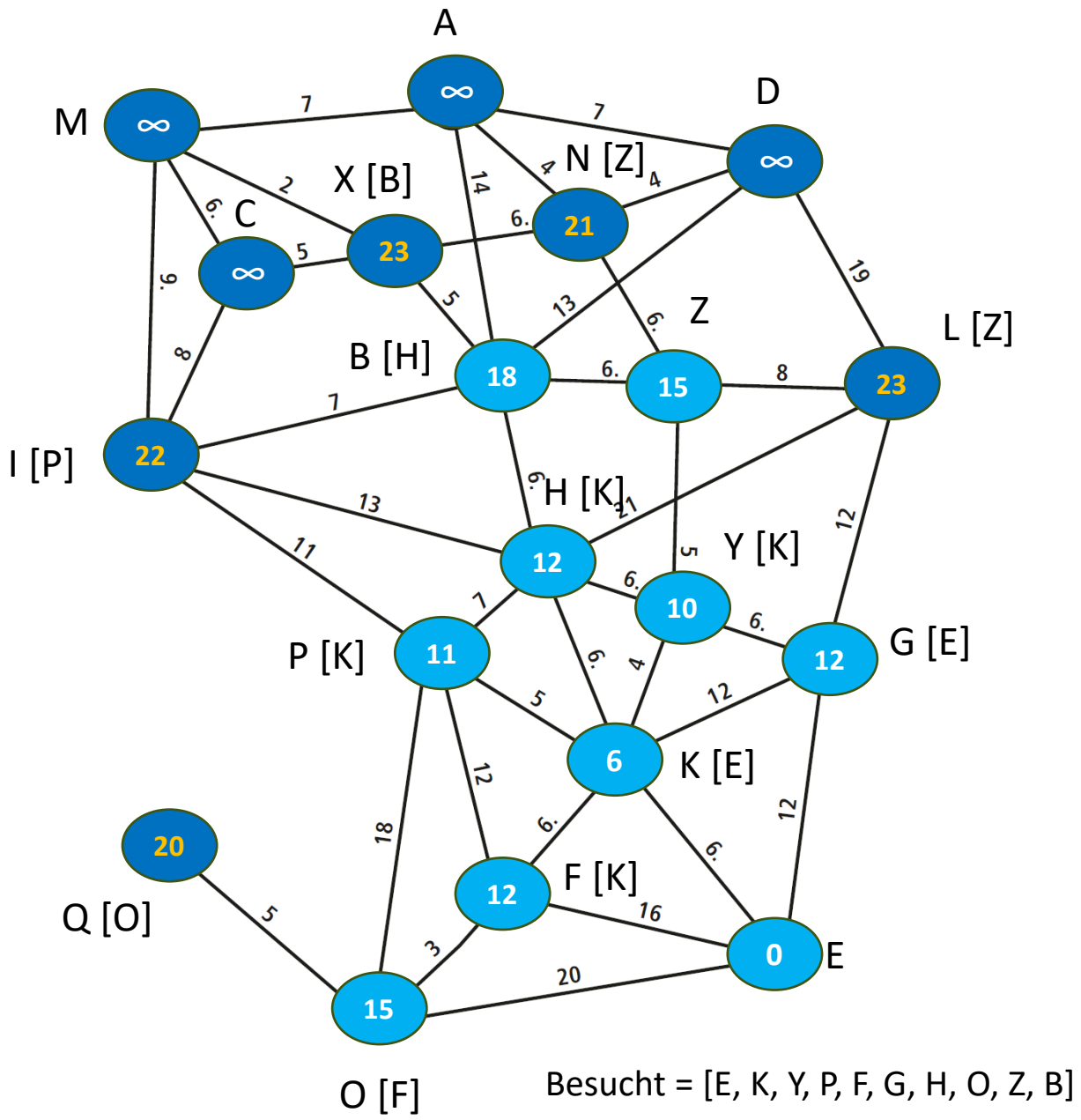
Hier: X -> X[Z]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

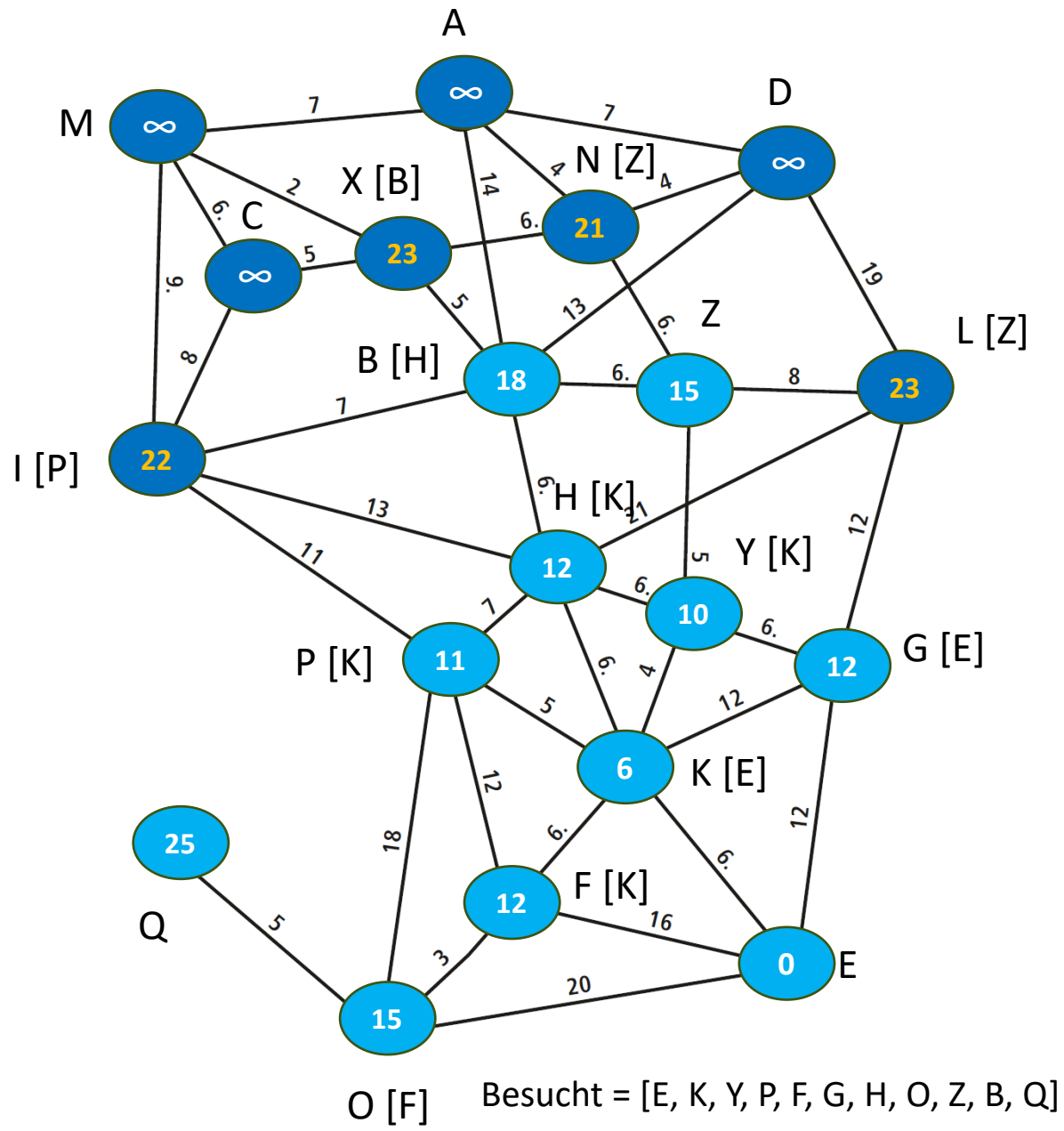
Das ist hier Knoten Q



**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge Q zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

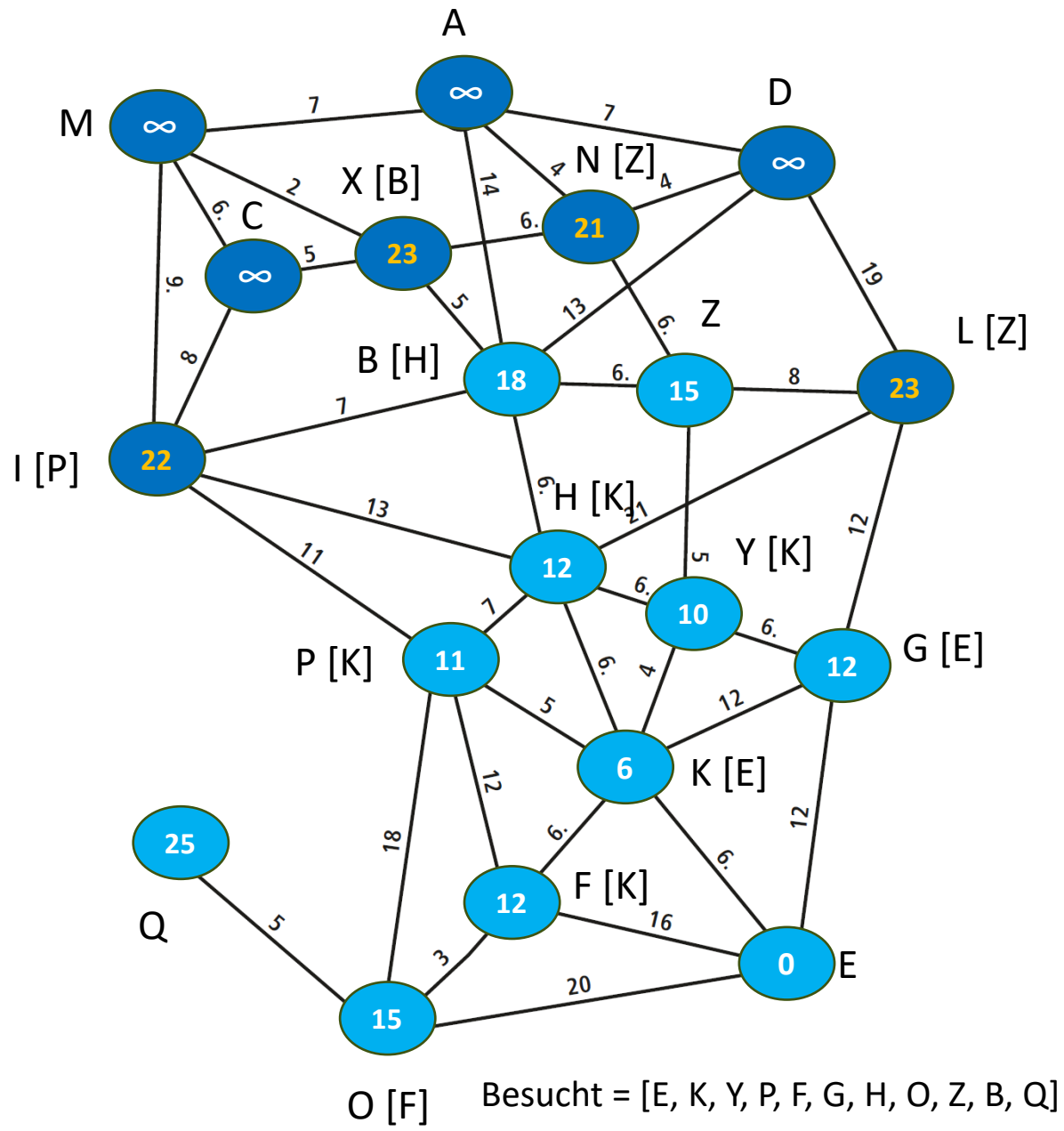


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

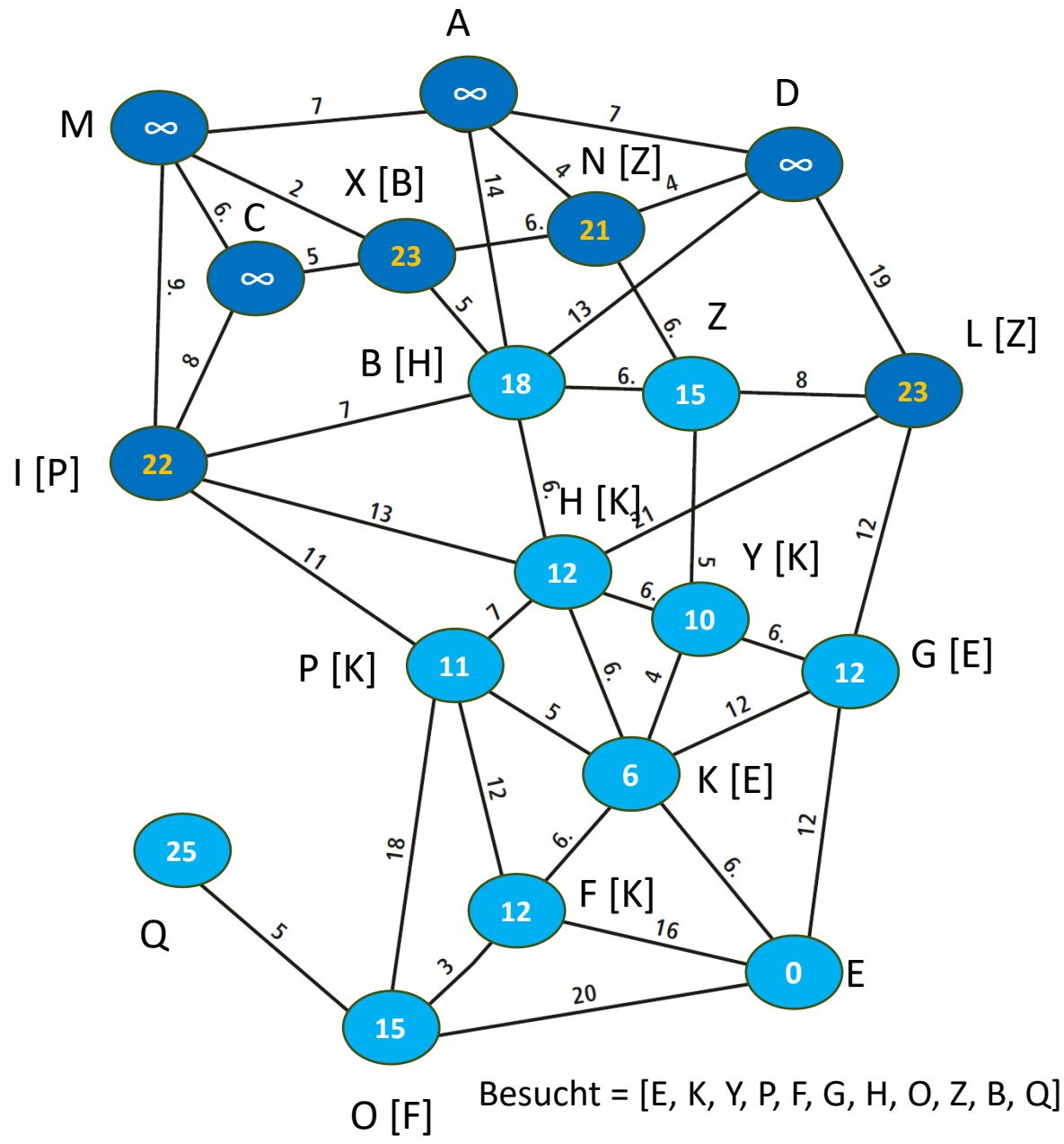
Hier: keine Änderung



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das ist hier Knoten N

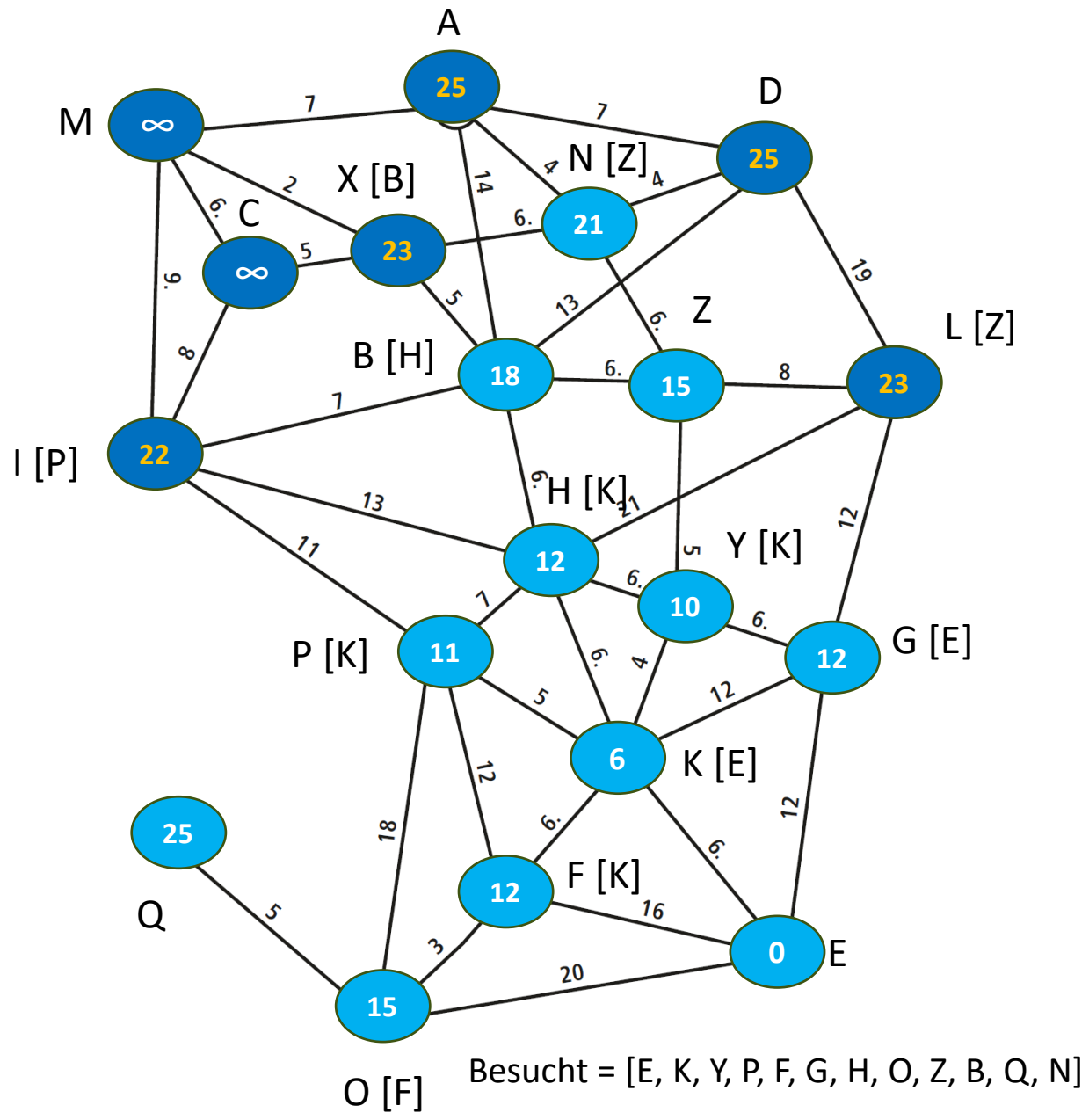


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge N zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Hier: D, A

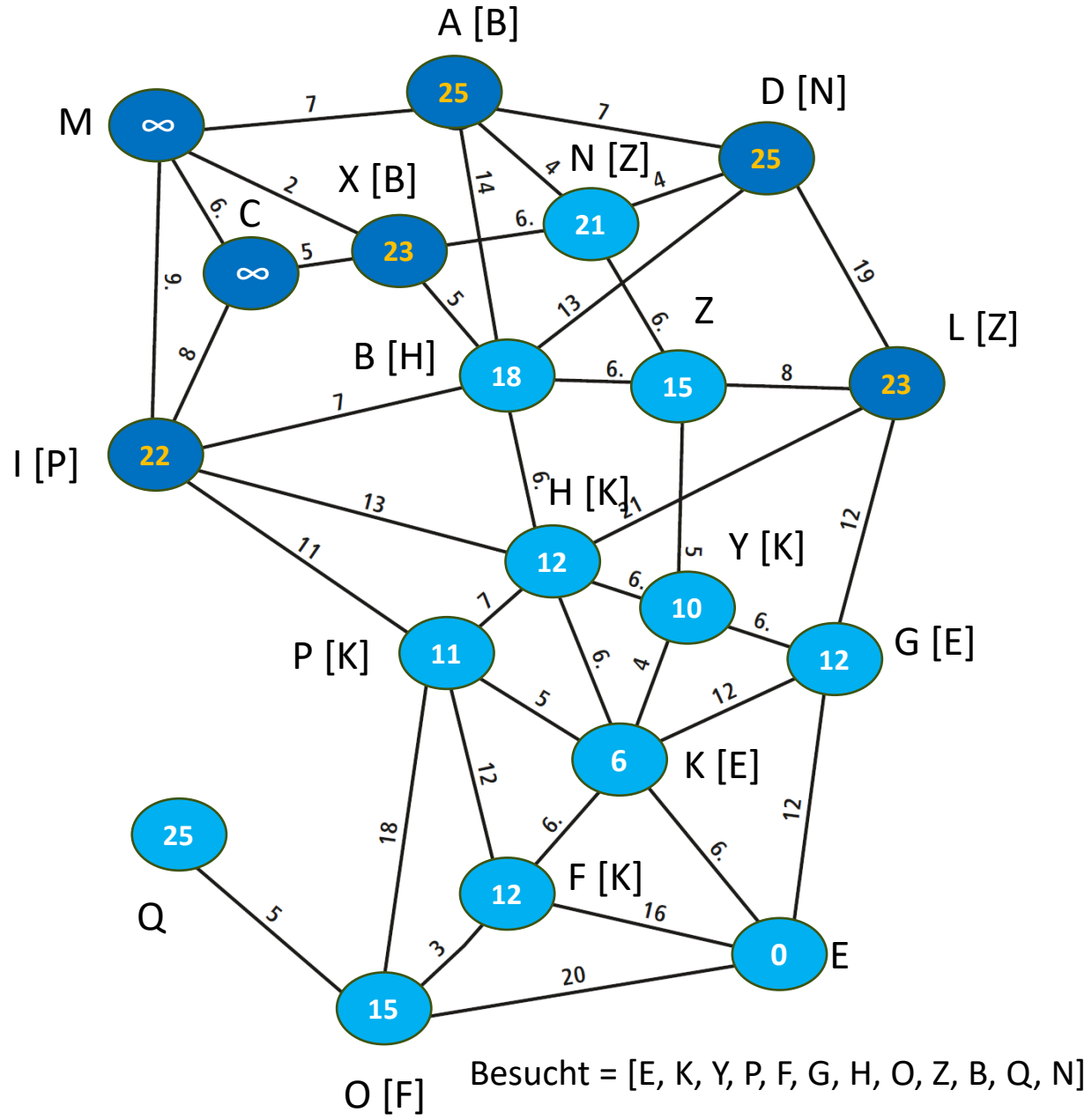


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

Hier:
D -> D[N]
A -> A[B]

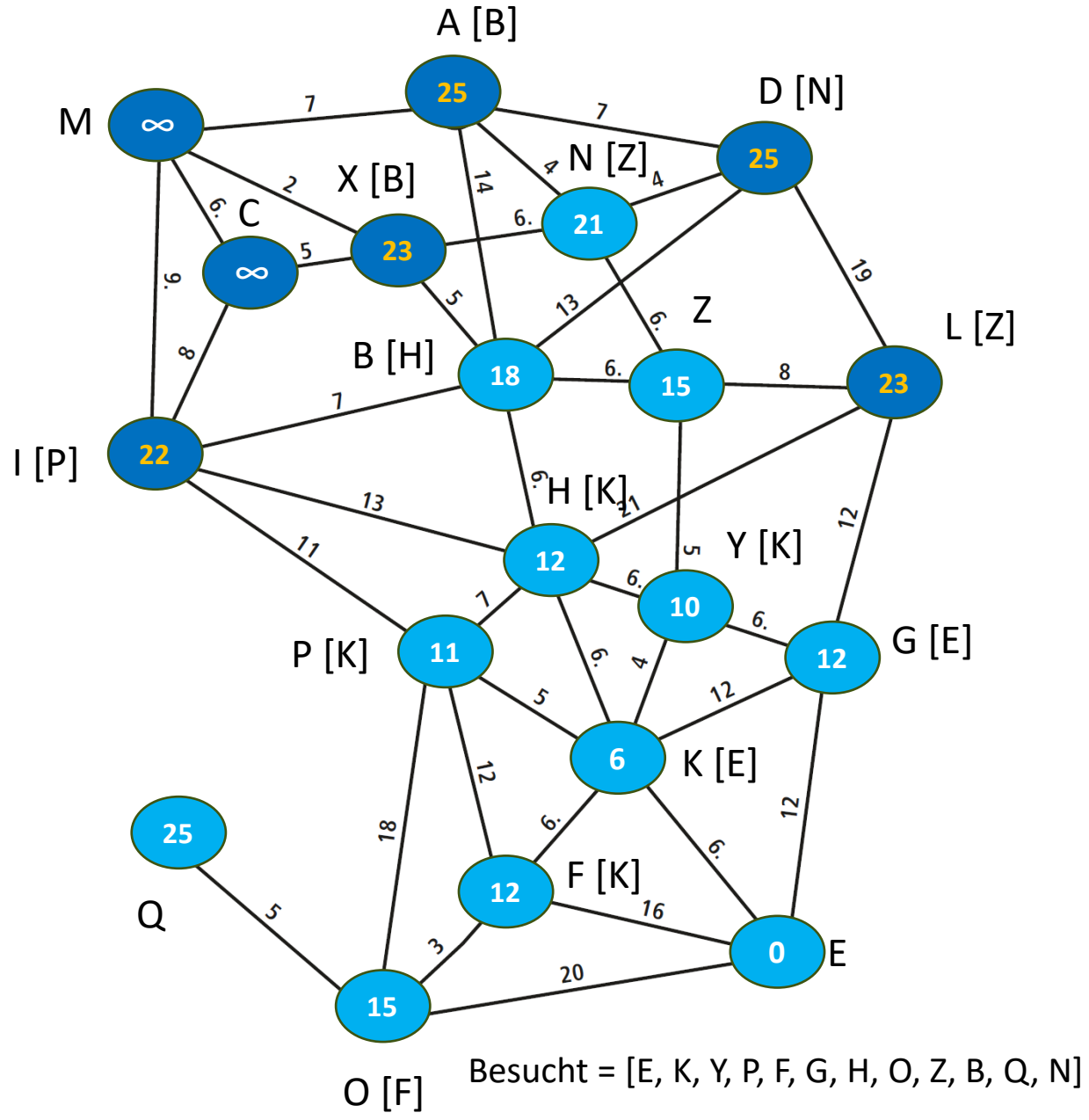



Besucht = [E, K, Y, P, F, G, H, O, Z, B, Q, N]

**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das ist hier Knoten P

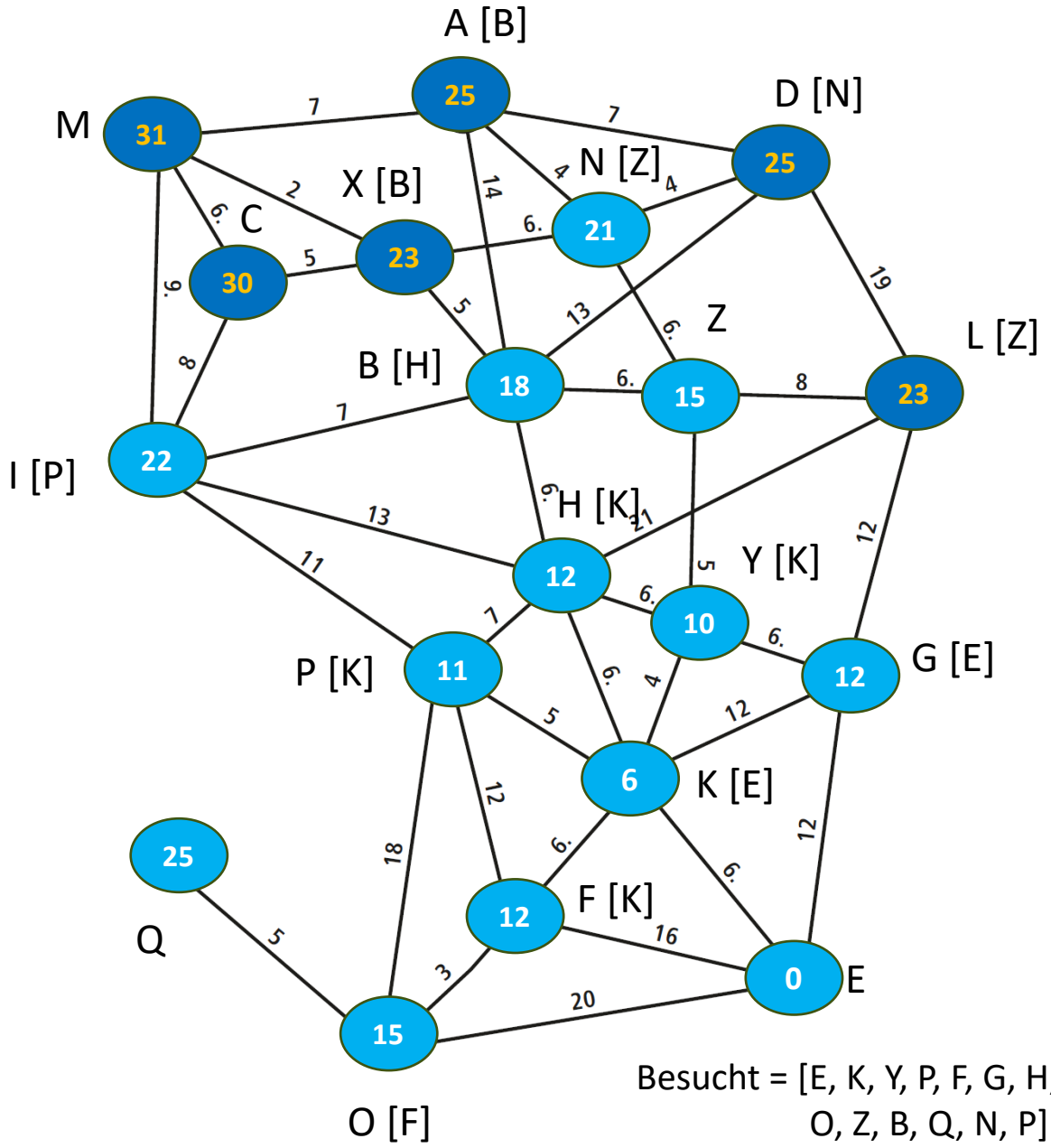


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge P zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Hier: M, C

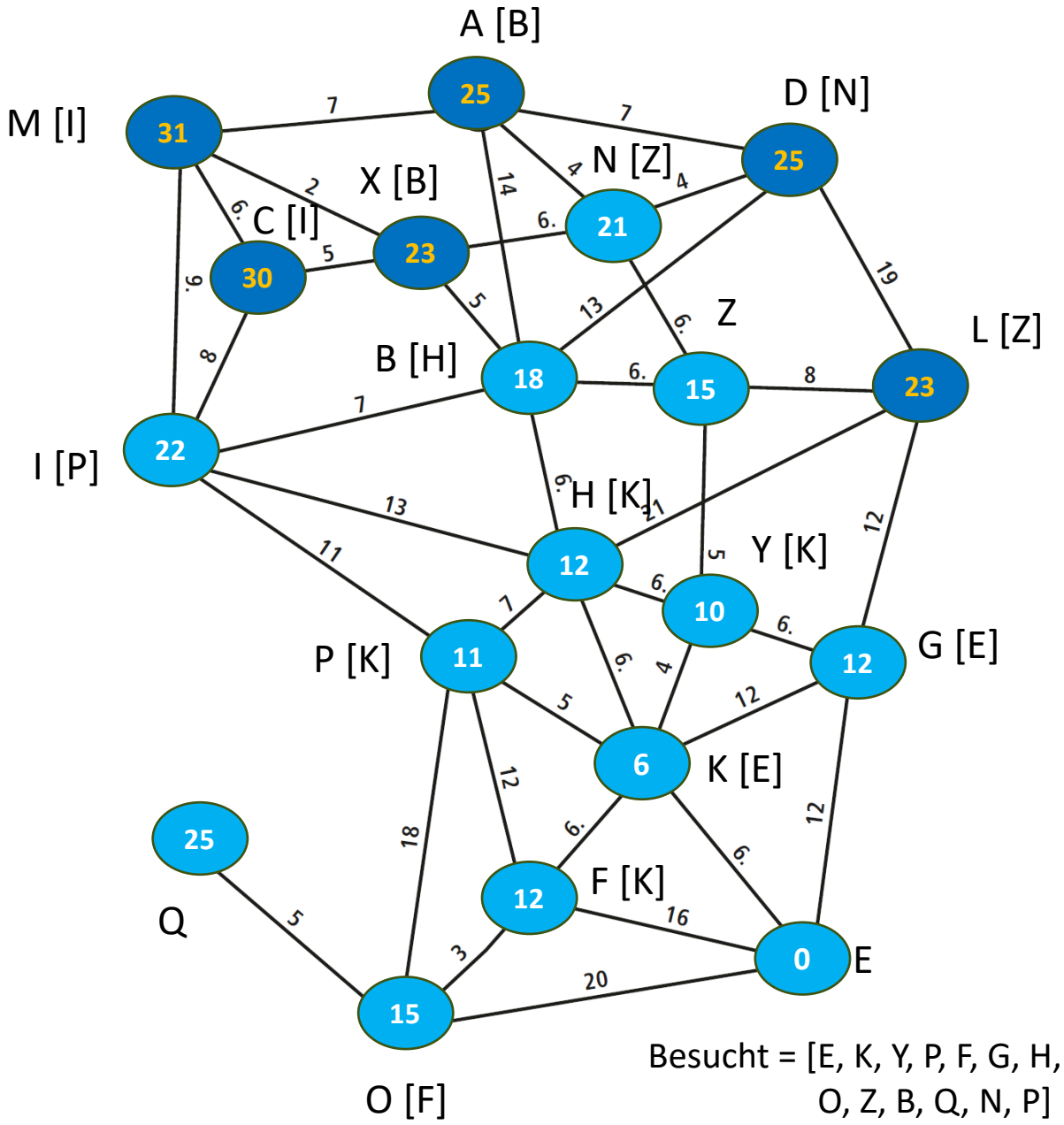


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

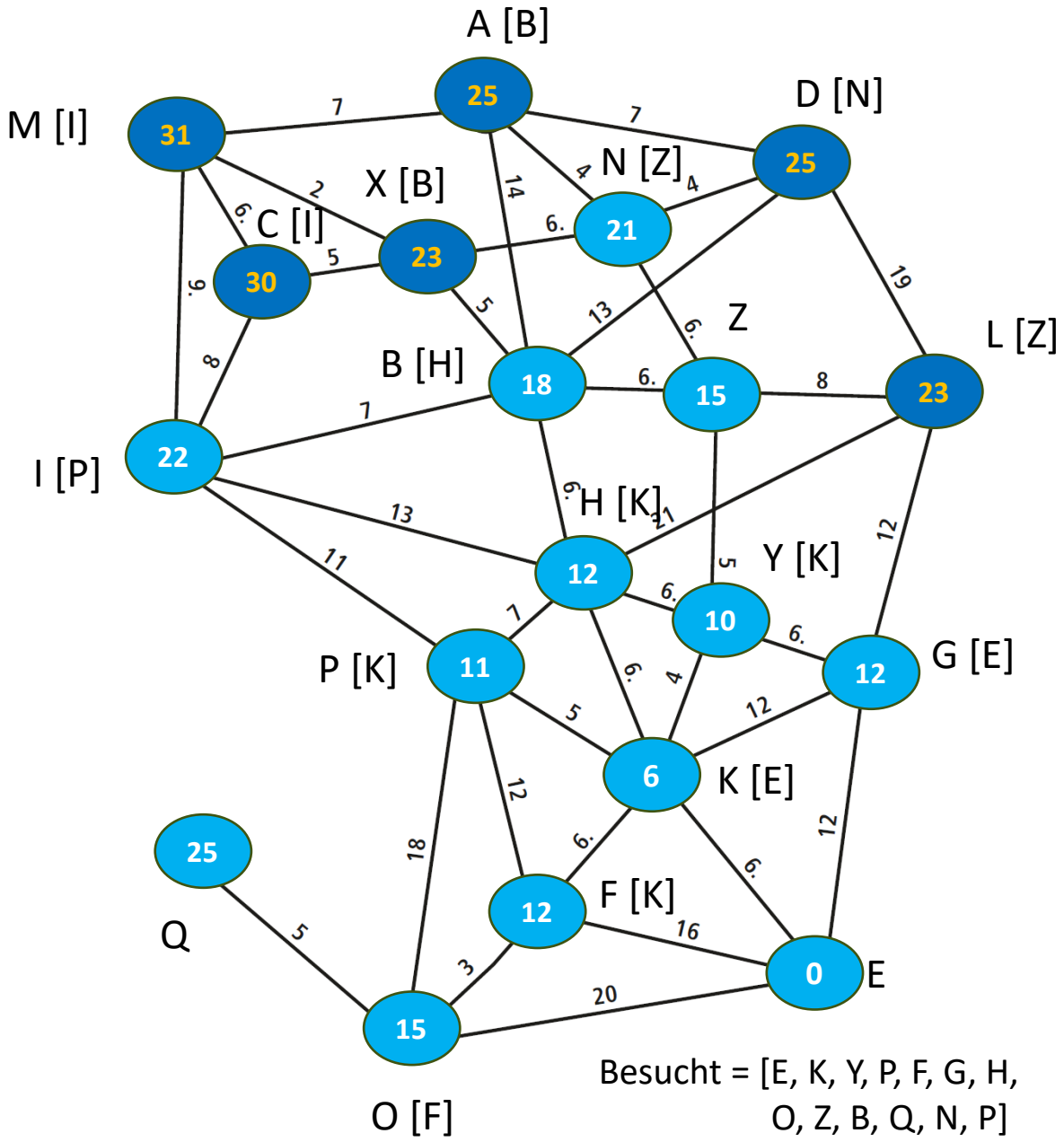
Hier:
C -> C[I]
M -> M[I]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das ist hier Knoten X (oder L)

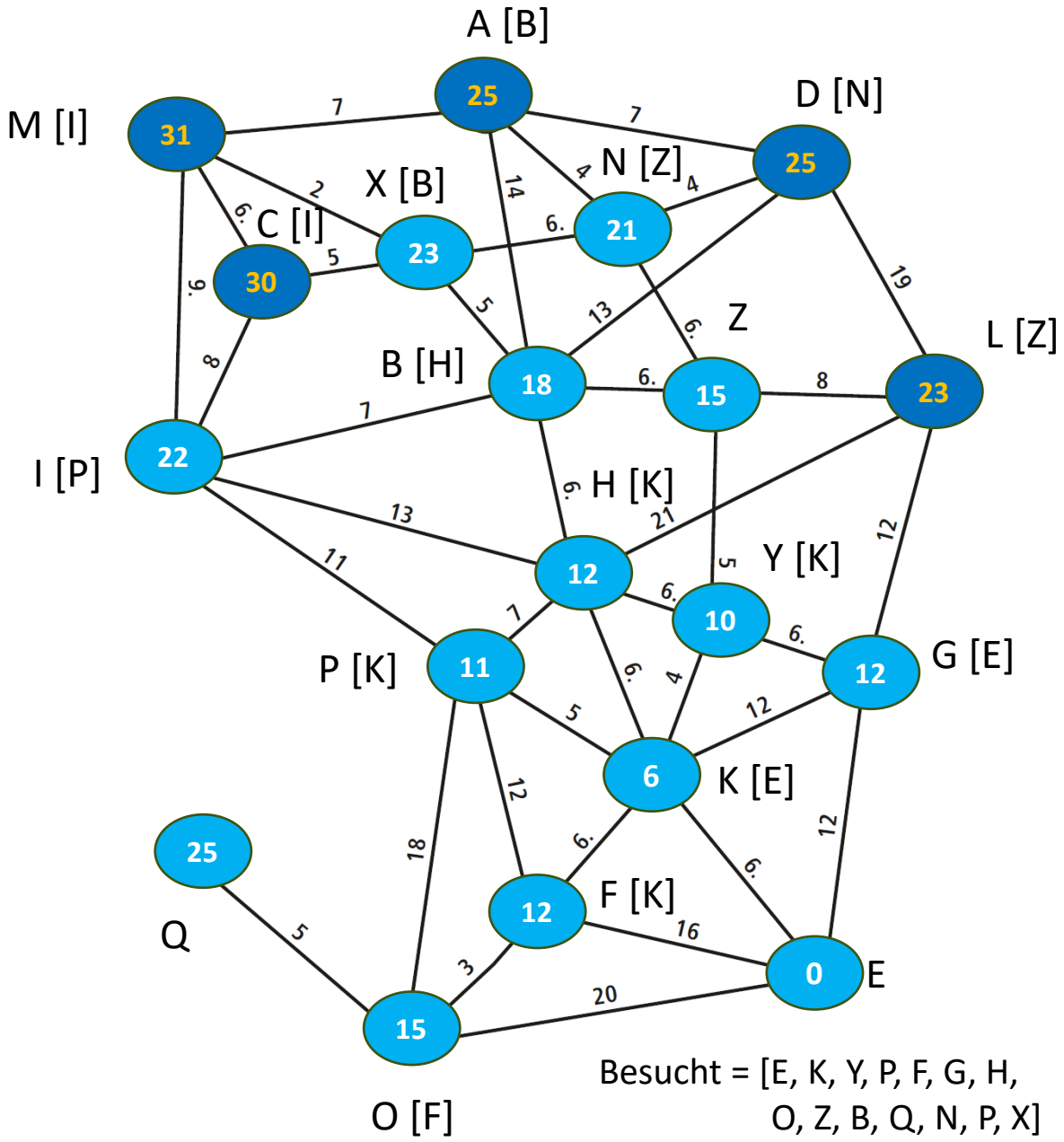


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge X zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Hier: keine Änderung

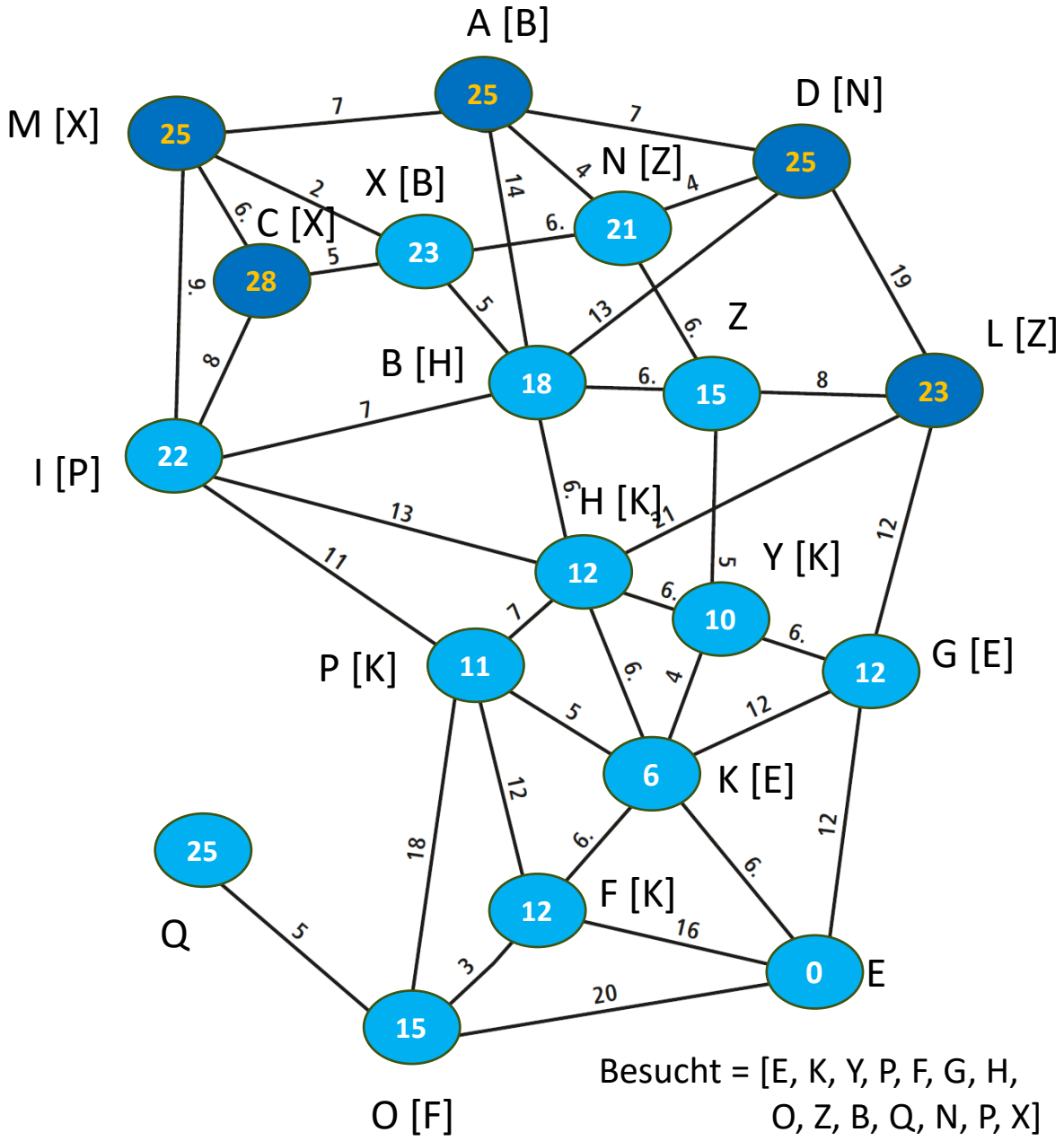


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

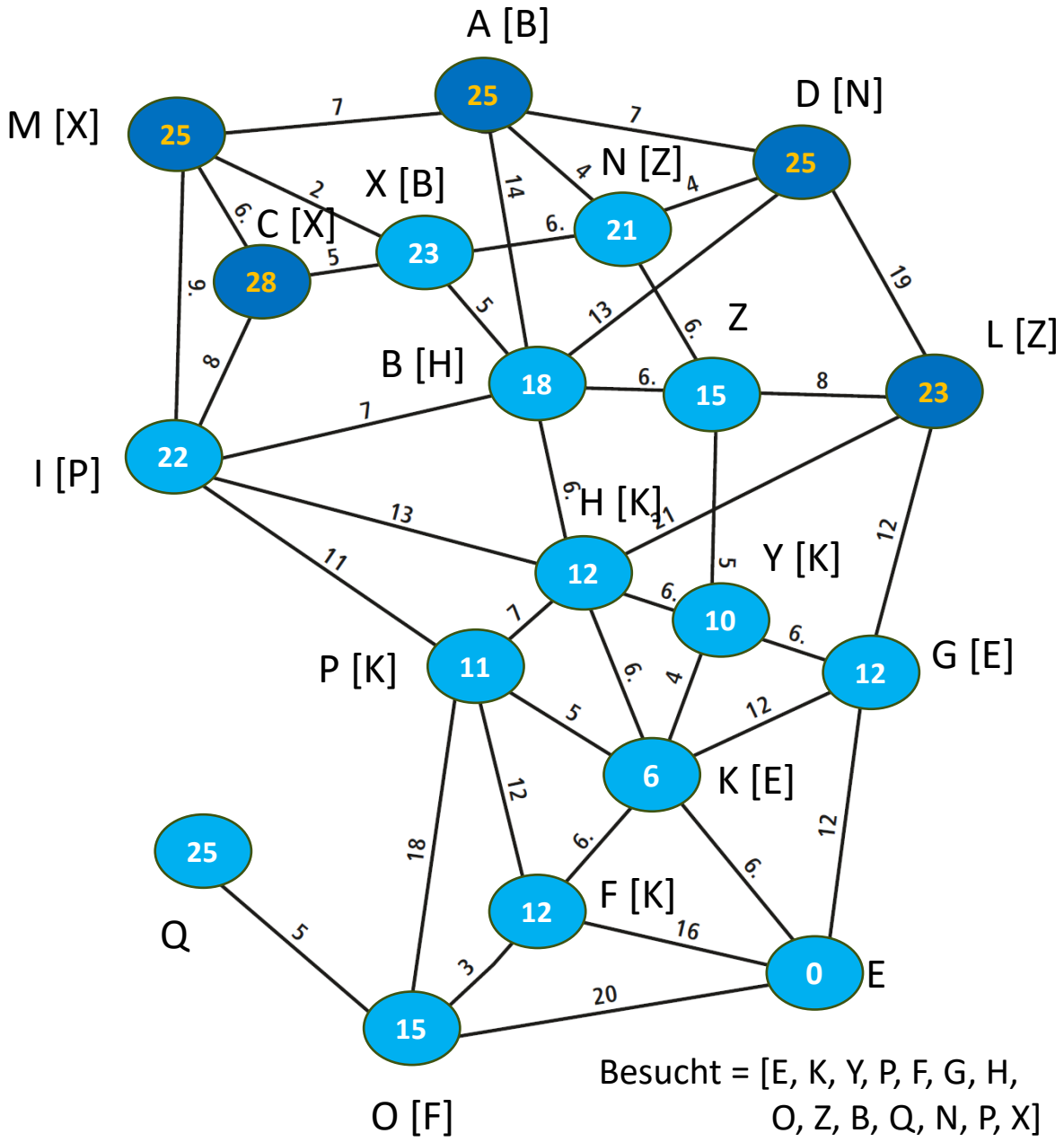
Hier:
C [I] -> C [X]
M[I] -> M[X]



**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

Das ist hier Knoten L

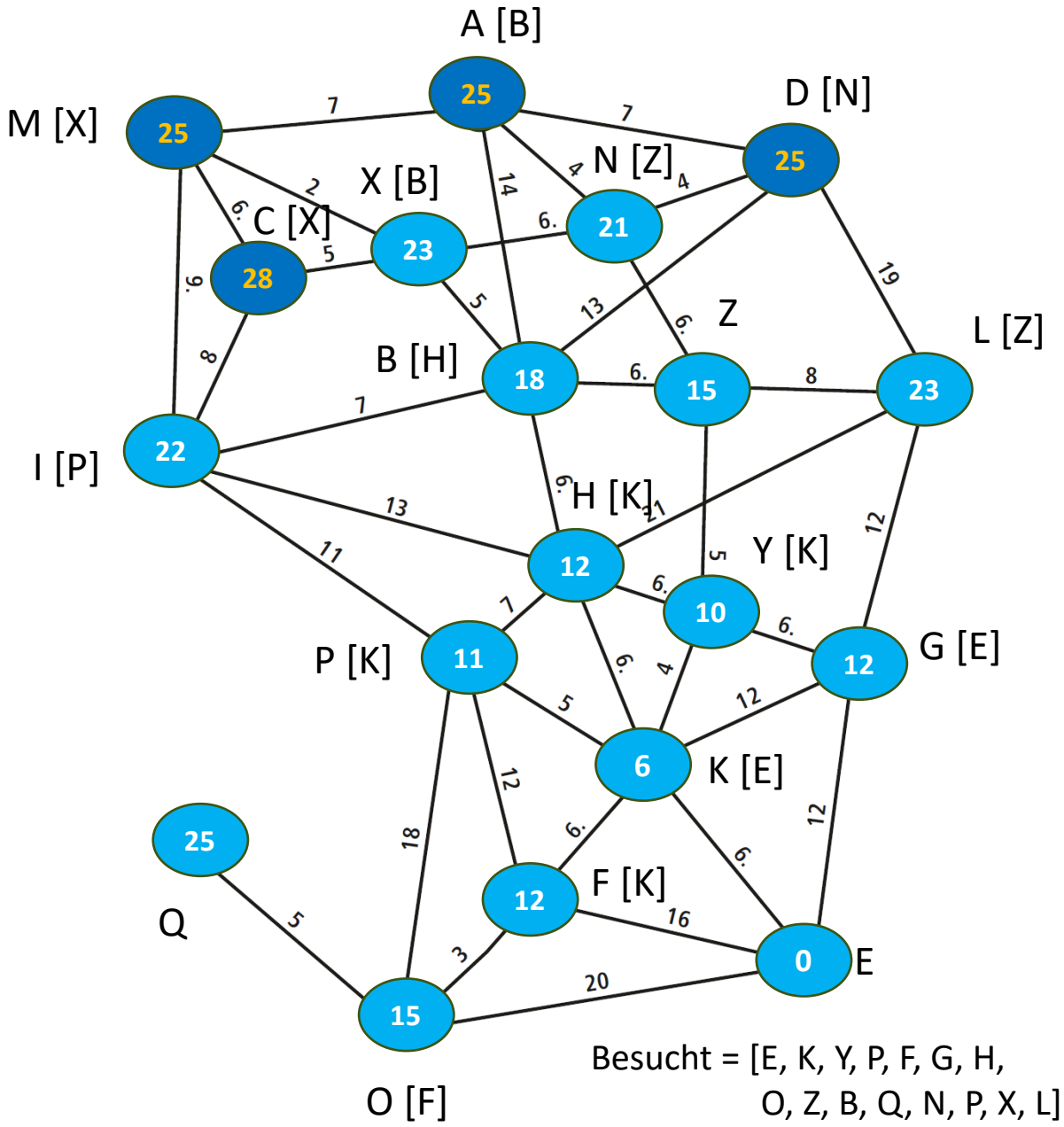


**Schritt 2:** Besuchte Knoten
Erstelle die Liste der besuchten Knoten.

Hier: füge L zur Liste Besucht hinzu

**Schritt 3:** Nachbarknoten untersuchen
Jetzt schaust du dir alle Nachbarknoten des aktuellen Knotens an. Für jeden Nachbarn berechnest du die neue Distanz: (aktuelle Distanz) + (Gewicht der Kante zum Nachbarn).

Hier: keine Änderung

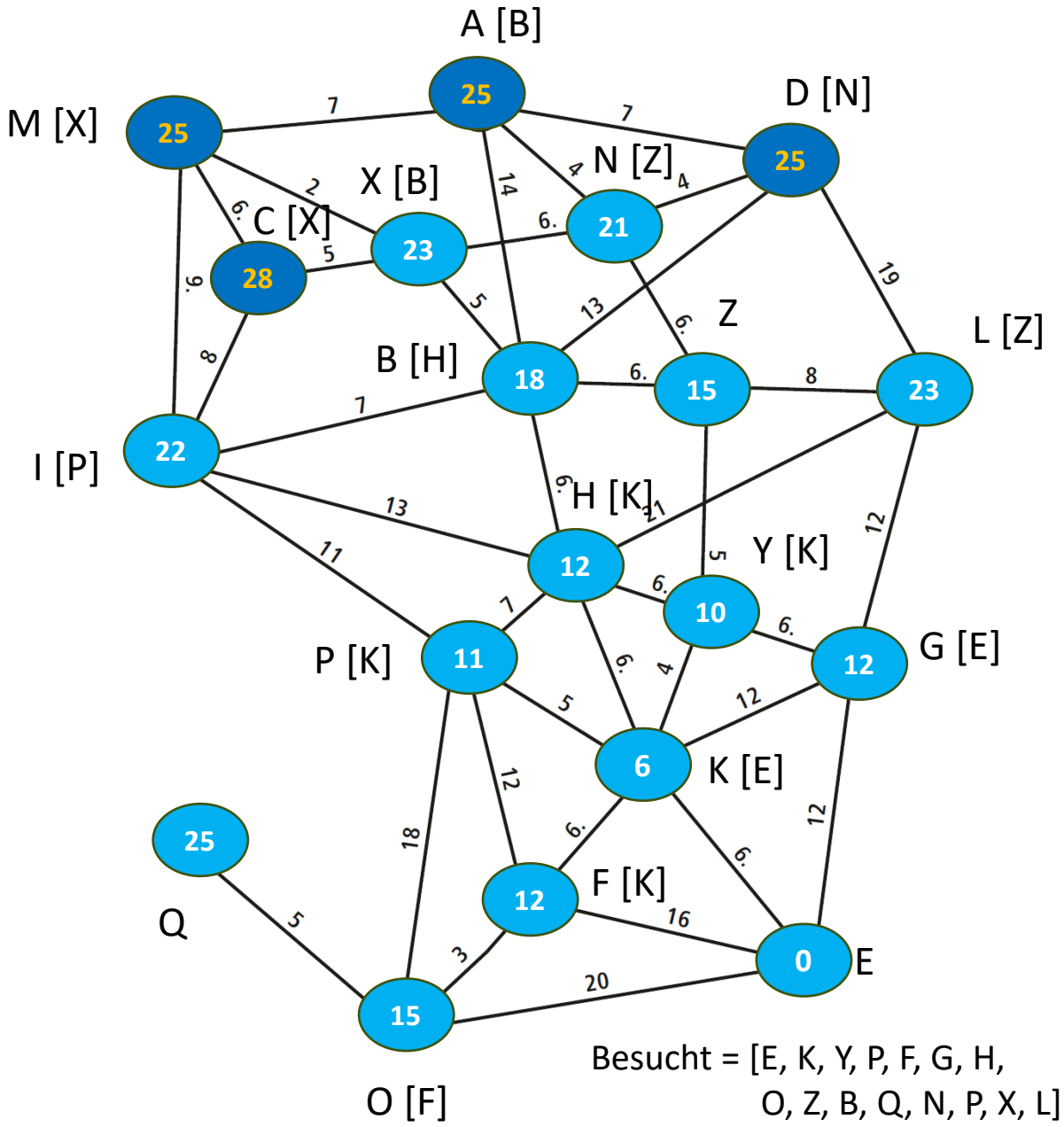


**Schritt 4:** Distanzen aktualisieren

Wenn diese neu berechnete Distanz kleiner ist als die bereits bekannte Distanz des Nachbarn, aktualisierst du die Distanz des Nachbarn.

Du speicherst/aktualisierst auch, von welchem Knoten aus du diesen Nachbarn erreicht hast, um am Ende den kürzesten Weg rekonstruieren zu können.

Hier: keine Änderung



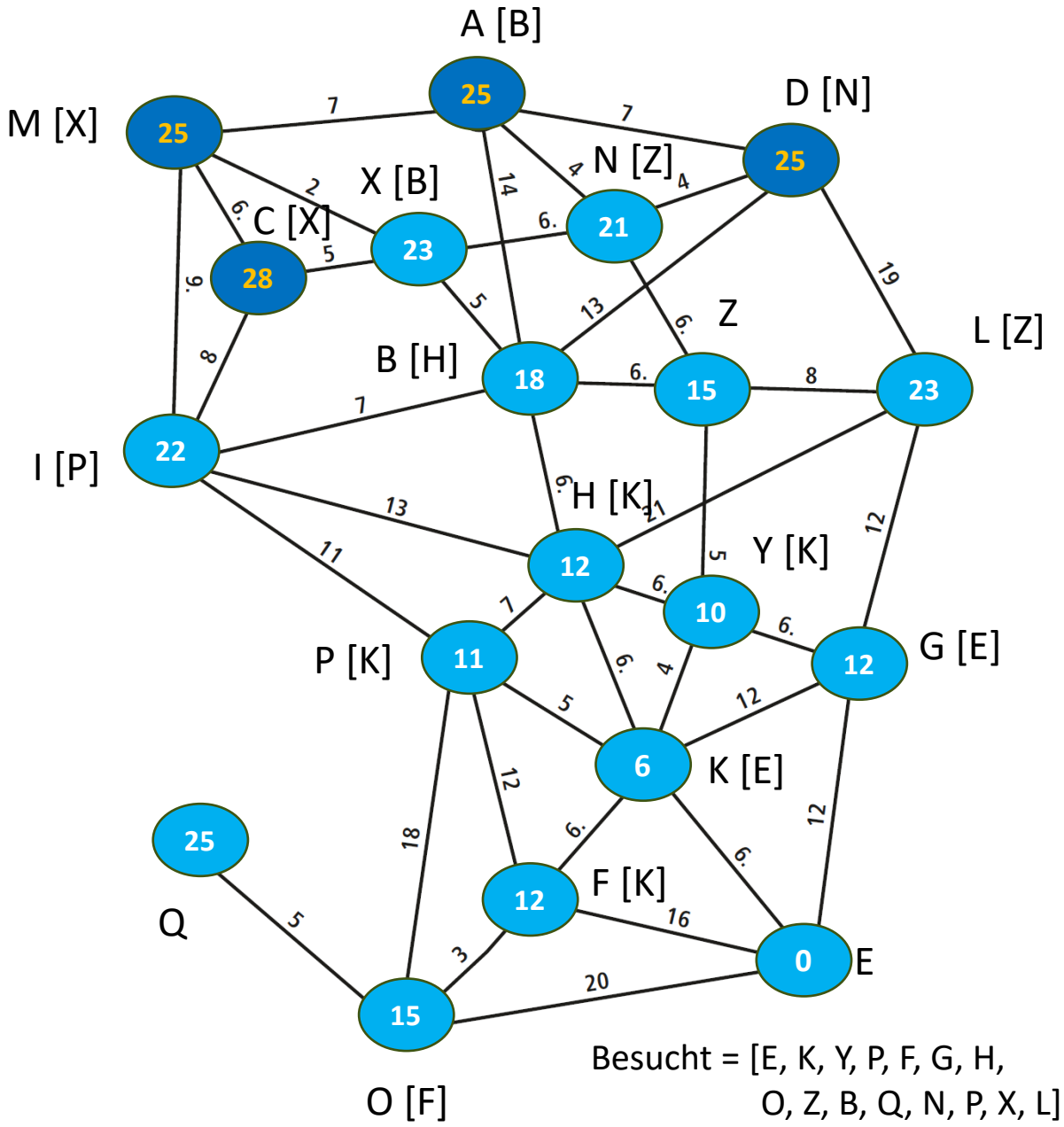
**Schritt 5:** Wiederholen

Wähle den nächsten verbundenen aber unbesuchten Knoten mit der aktuell kleinsten Distanz aus der Liste aus und verarbeite ihn. Wiederhole dafür die Schritte 2 bis 4.

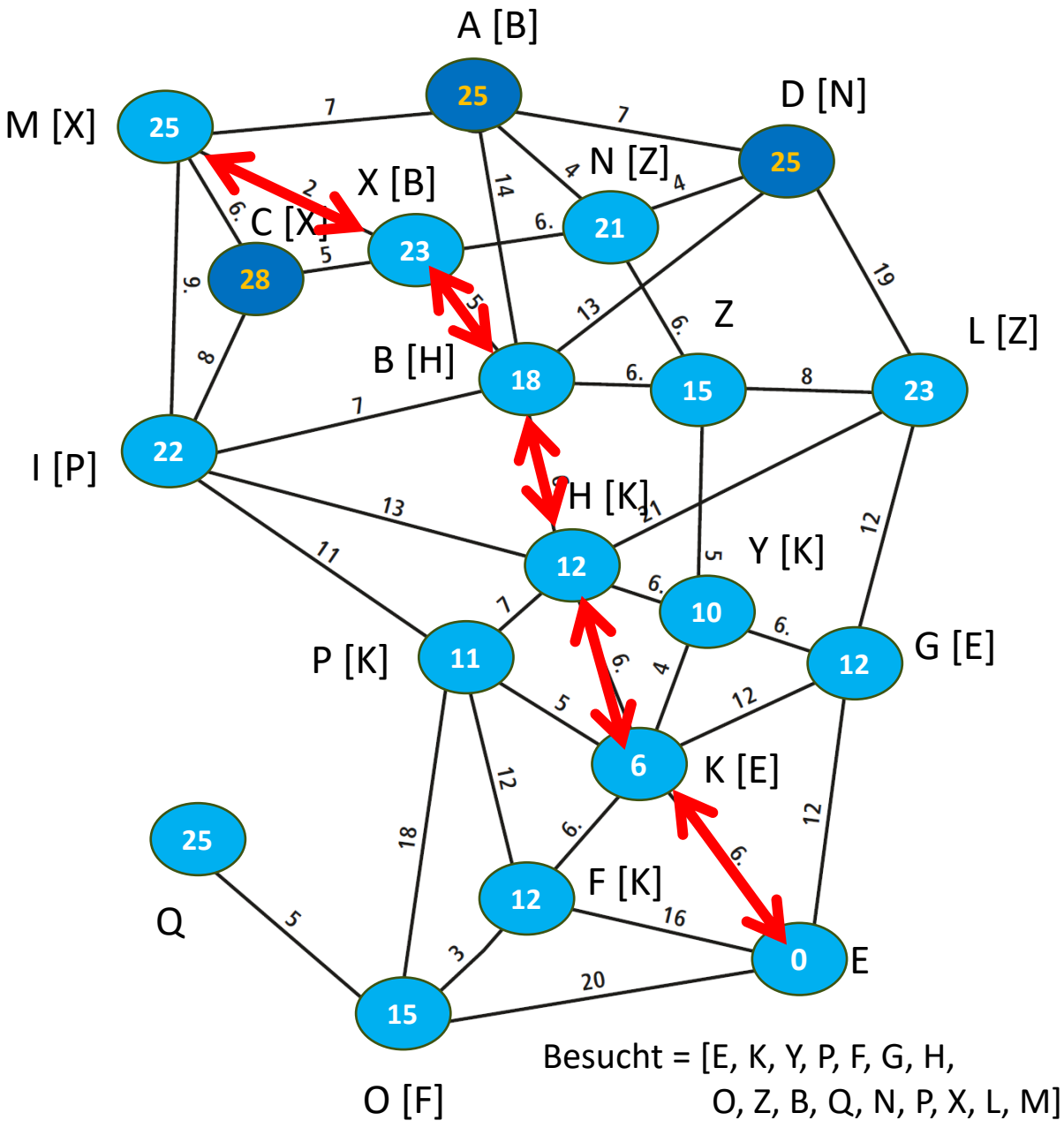
Das ist hier Knoten M, A oder D

Wir nehmen M

"Das machst du so lange, bis du den Zielknoten (M) erreicht hast oder keine Knoten mehr zu besuchen sind."



**Der kürzeste Pfad ist eingezeichnet**



# Lernziele

# Lernziele Prüfung Graphentheorie

- Definition – was ist ein Graph
- Darstellung von Graphen: Verstehen und Anwenden können der Darstellungskonzepte
  - Adjazenzliste
  - Adjazenzmatrix
- Eigenschaften von Graphen:
  - Verstehen der Grapheigenschaften, bspw
    - Multigraph vs. Einfacher Graph
    - Zusammenhängend vs. nicht zusammenhängend
    - Zyklisch vs. azyklisch
    - Gerichtet vs. Ungerichtet
    - Grad eines Knotens
    - etc.
  - Gegebene Grapheigenschaften algorithmisch beschreiben können
- Prim's Algorithmus:
  - Was leistet der Algorithmus?
  - Anwenden können auf Graph, mit Angabe von Zwischenresultaten
- Dijkstra's Algorithmus
  - Was leistet der Algorithmus?
  - Anwenden können auf Graph, mit Angabe von Zwischenresultaten

- Keine Programmieraufgaben
- Gewicht: 2/3 einer "normalen" Prüfung
- 2 Beispielaufgaben aus der letzten Prüfung
- Bitte Exam.net vorbereiten!

# Weitere Anwendungen des Graph Konzeptes

# Sprachkonzept

In der deutschen Sprache besteht ein Satz (S) z.B. aus einer Nominalphrase (NP) und einer Verbalphrase (VP), was man folgendermassen ausdrücken kann:
S → NP VP

Eine Nominalphrase besteht aus einem Nomen (N), einem Nomen mit einem vorangestellten Artikel (A) oder einem Nomen mit einer Präpositionalphrase (PP).
NP → N
NP → A N
NP → NP PP

Eine Verbalphrase wiederum kann aus einem Verb (V), einem Verb mit einer folgenden Nominalphrase, oder einer Verbalphrase mit folgender Präpositionalphrase bestehen.
VP → V
VP → V NP
VP → VP PP

# Sprachkonzept

Eine Präpositionalphrase wiederum besteht aus einer Präposition und einer Nominalphrase
PP → P NP

Wenn nun noch Artikel, Nomen, Präposition und Verb definiert werden, dann kann man Sätze bilden bzw. überprüfen. Die restlichen Regeln bzw. Ableitungen erfolgen durch Auf-zählen.

A → der
A → die
A → das
A → dem
A → den
P → mit
P → in

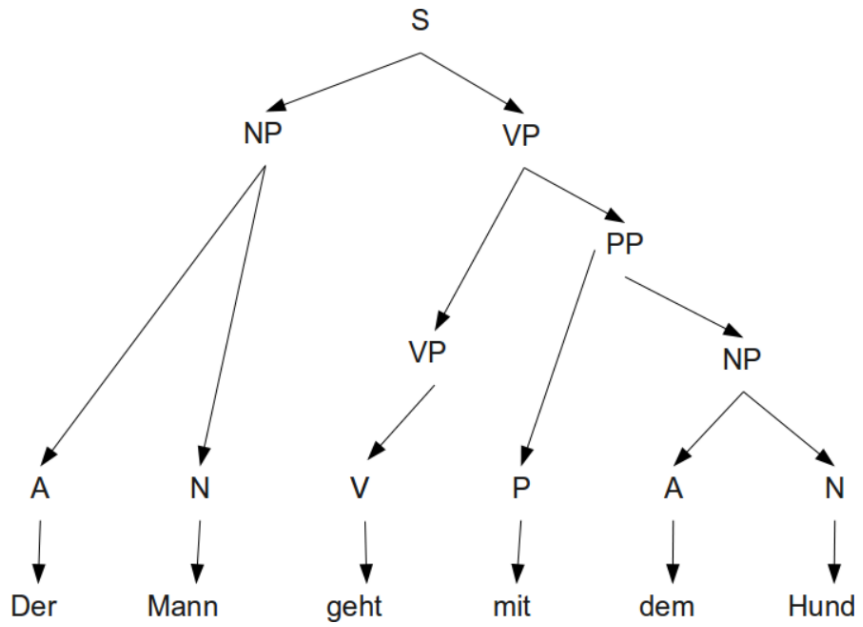
N → Mann
N → Hund
N → Park
N → Klaus
V → bellt
V → geht

# Sprachkonzept

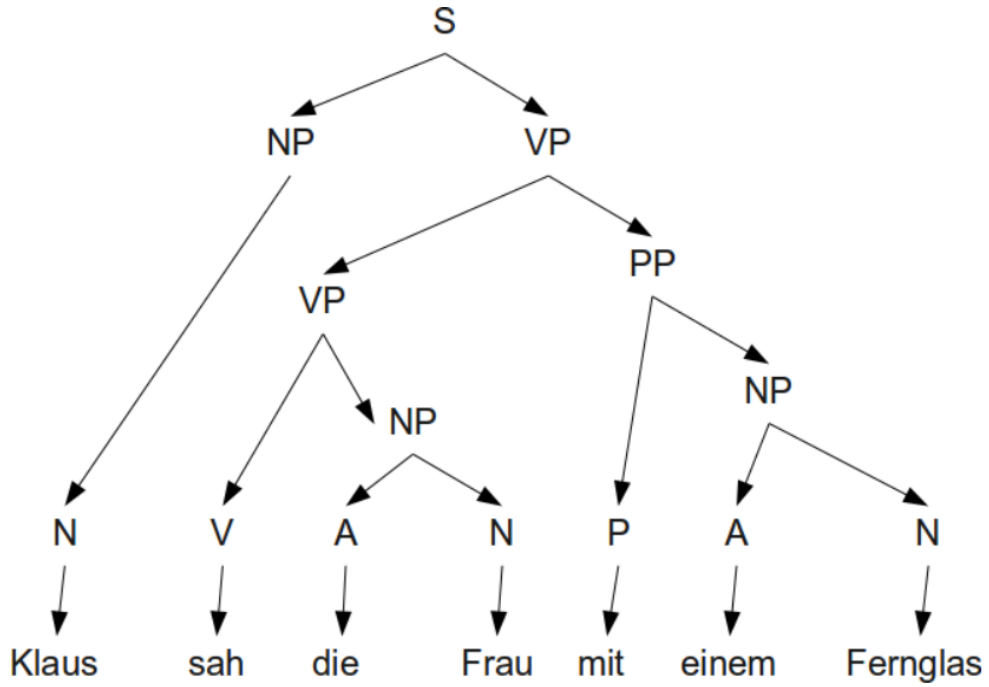
Nun kann man beginnen Sätze in dieser Sprache zu formulieren, bzw. zu überprüfen, ob ein gegebener Satz zu dieser Sprache gehört.

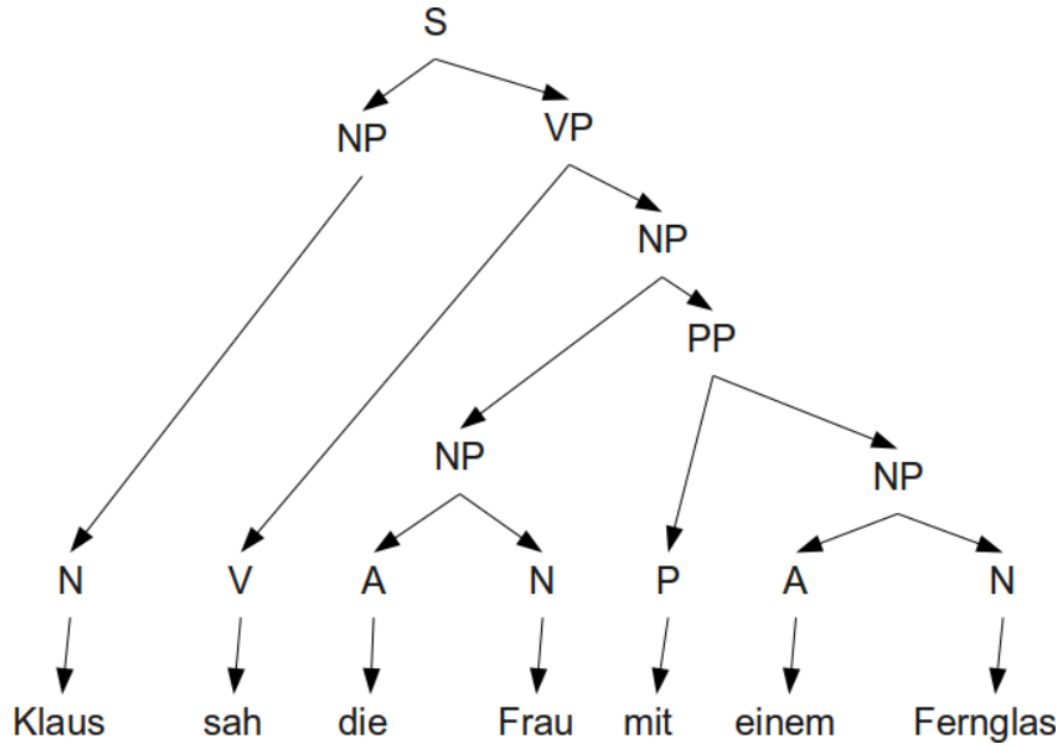
„Der Hund bellt“ lässt sich folgendermassen produzieren:
S → NP VP → A N V → der Hund bellt.

Etwas unübersichtlicher wird die Situation schon bei dem Satz „Der Mann geht mit dem Hund“. Hier hilft ein Baumdiagramm zur Übersicht.



# Zwei Bäume für denselben Satz

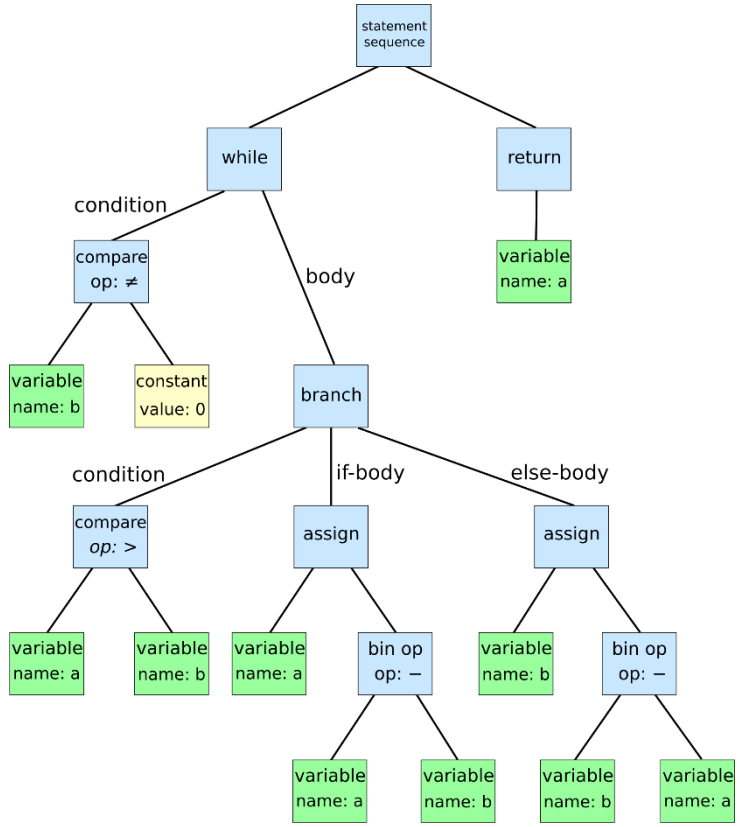


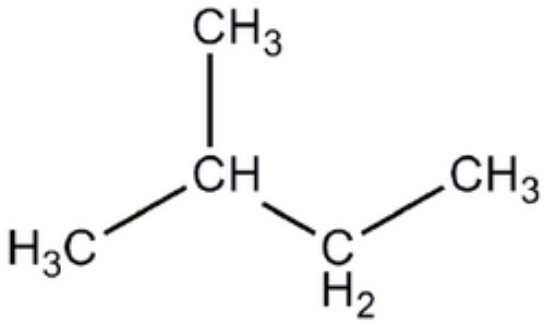
# Programmiersprachen

```
while b ≠ 0:
    if a > b:
        a := a - b
    else:
        b := b - a
return a
```

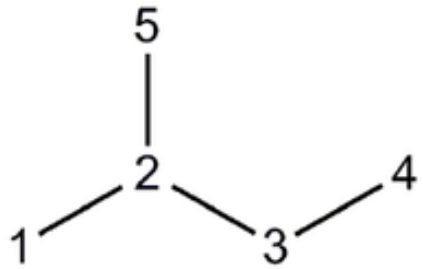
Abstract Syntax Tree (AST)



# Chemie




Molecule

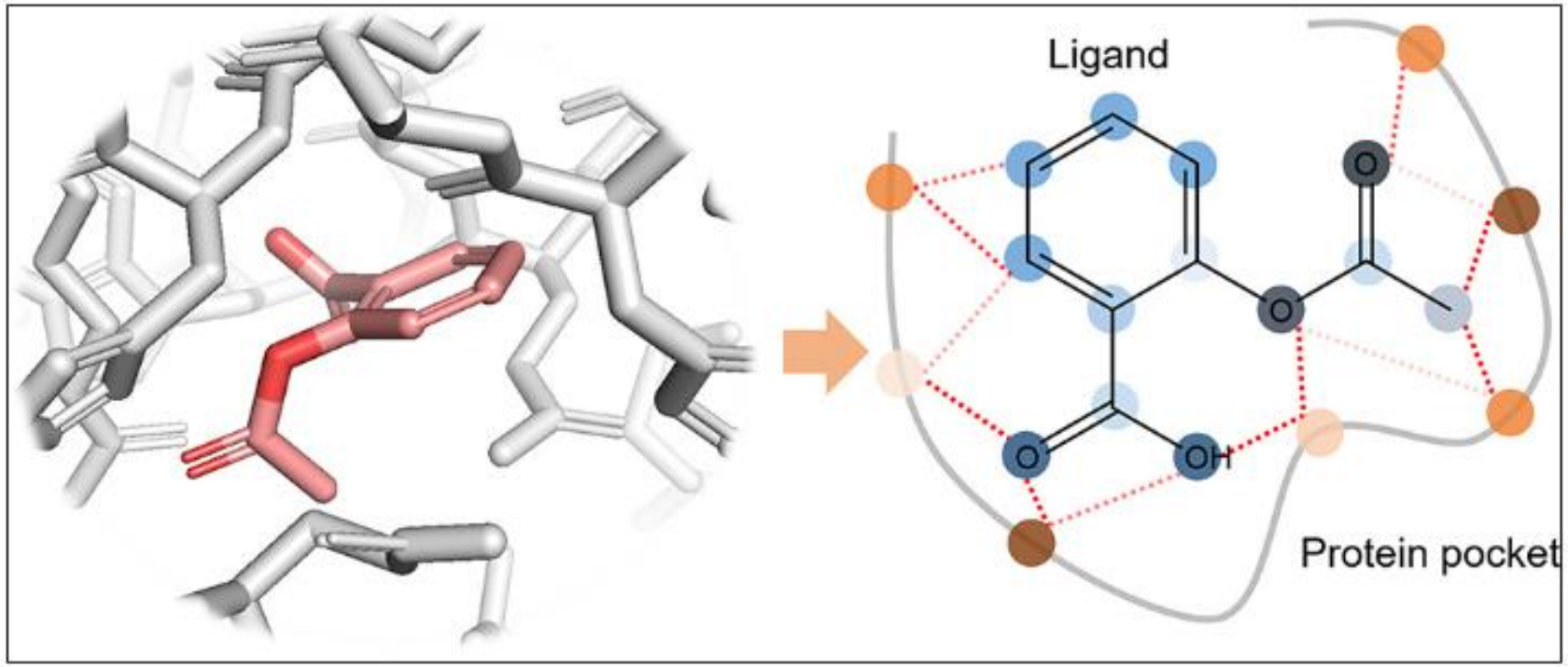



Graph

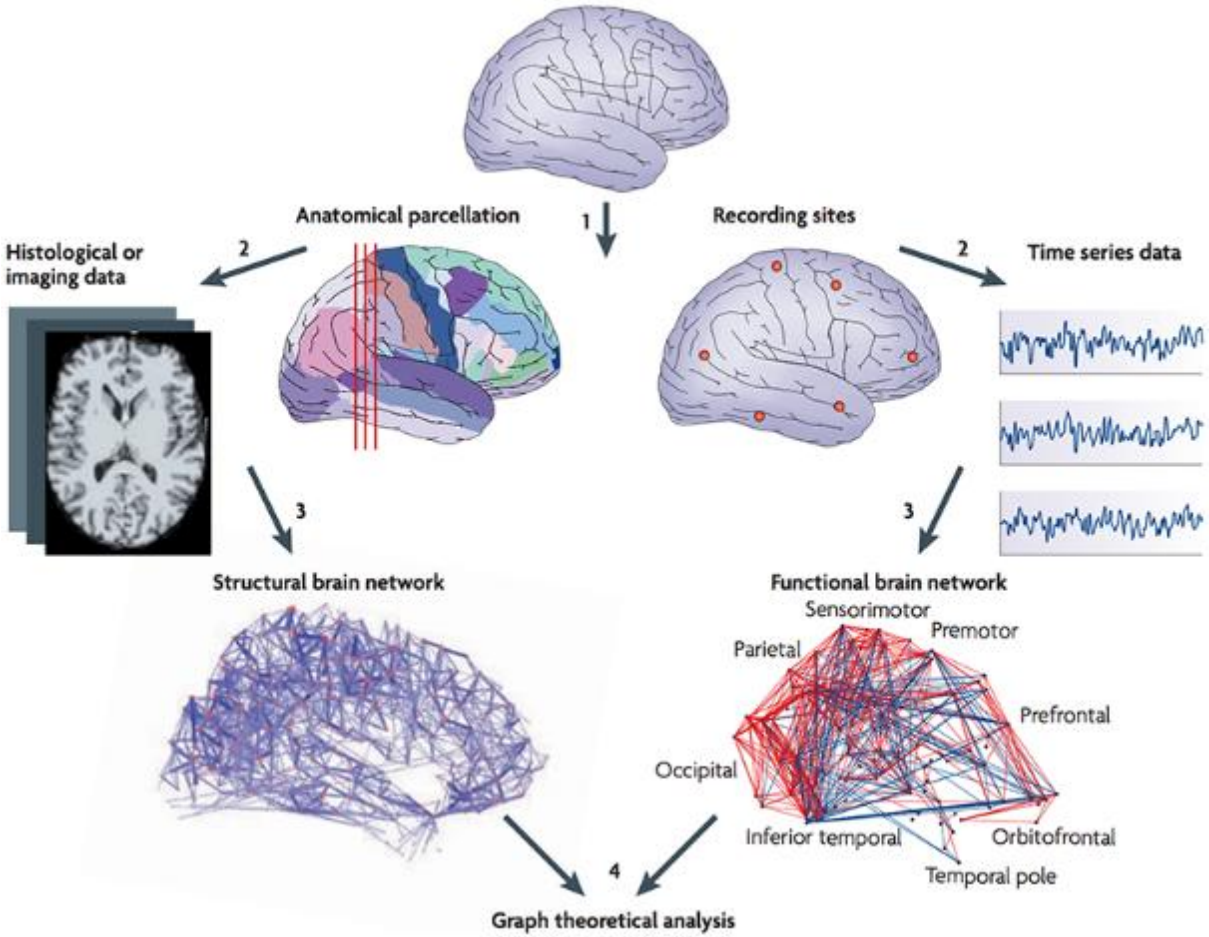
$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

adjacency matrix

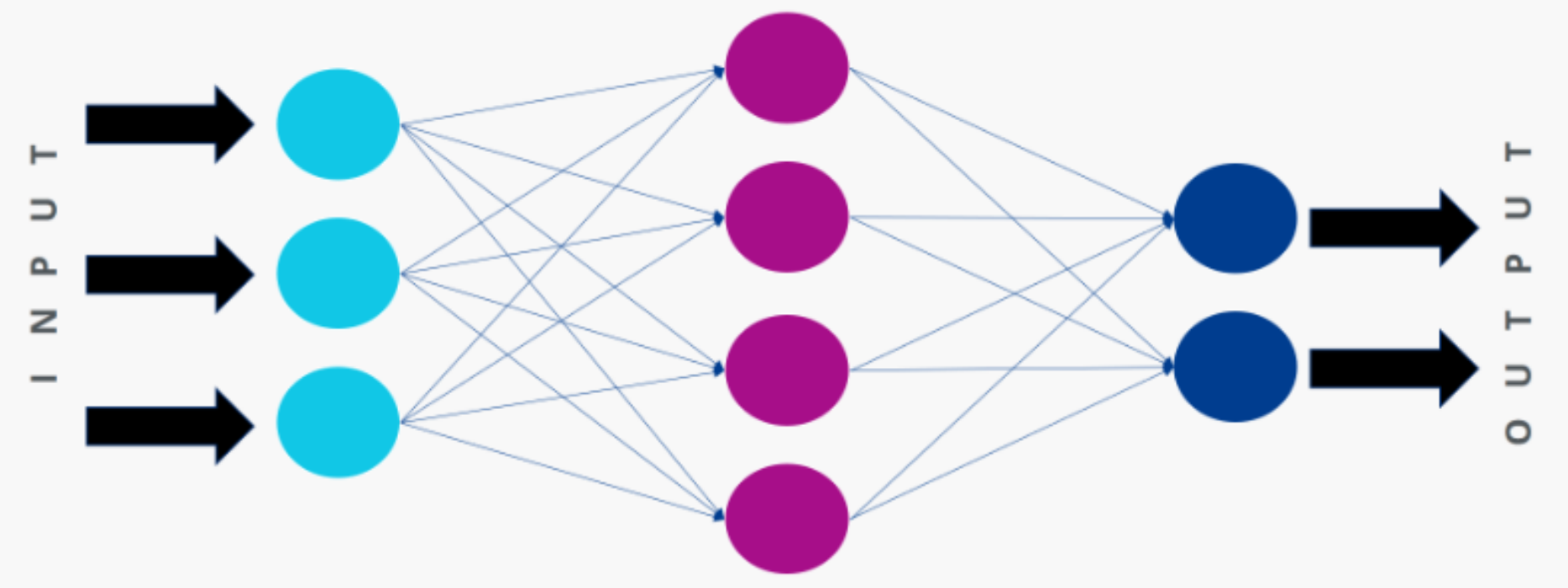
# Biologie



# Biologie (Hirn)



# Neuronale Netze



# Neuronale Netze