

# SoPra 2015

# KI - Strategien

Ihre Fragen, unsere Antworten

# Bewertungsfunktion

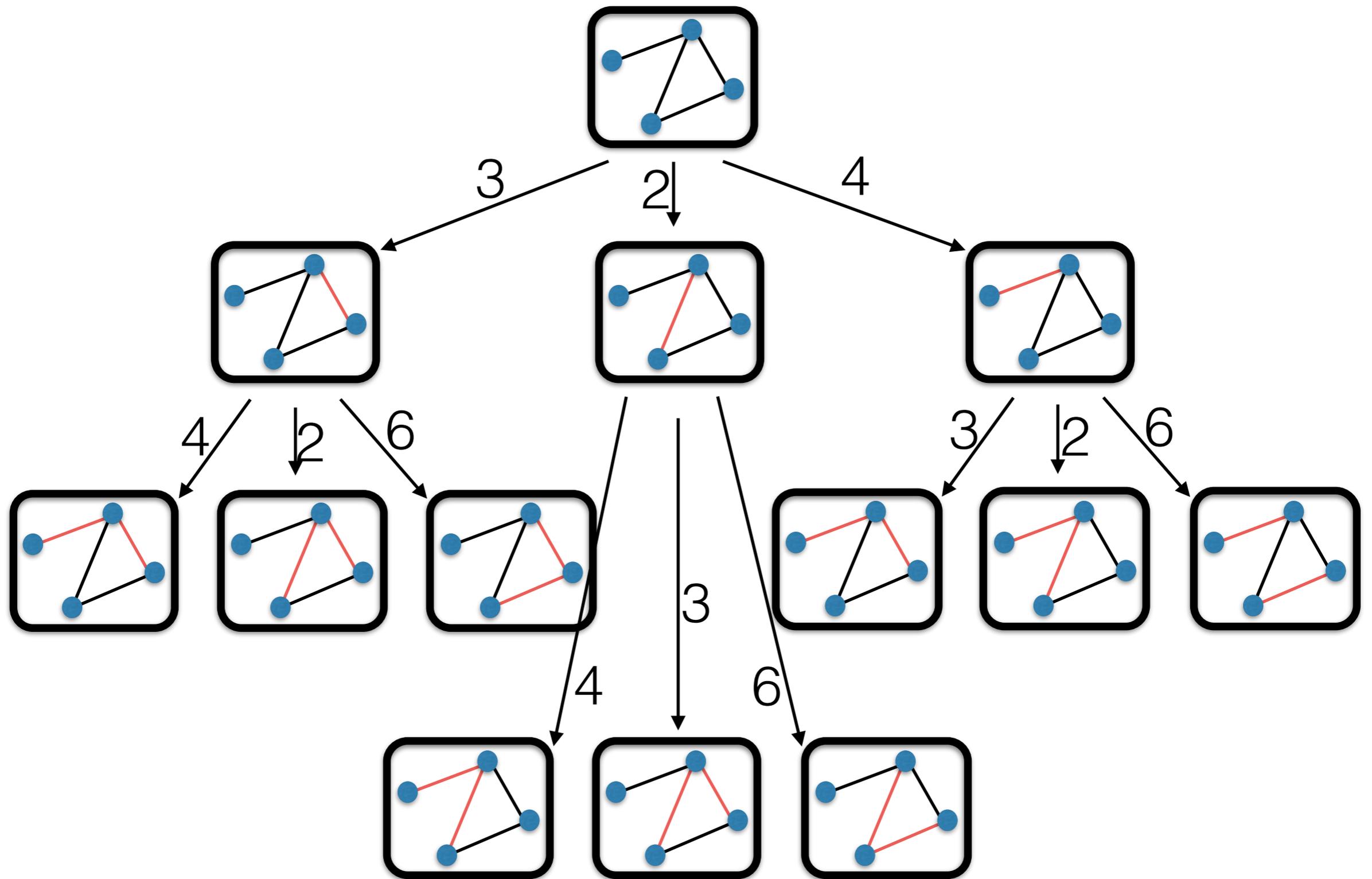
$$f(\text{Map}) = \text{Qualität}$$
The image shows a board game map of the United States, likely from the game 'Ticket to Ride'. The map is a grid of cities connected by colored lines representing routes. Cities labeled include Vancouver, Seattle, Portland, Salt Lake City, San Francisco, Los Angeles, El Paso, Phoenix, Dallas, Houston, New Orleans, Little Rock, Nashville, Atlanta, Charlotte, Memphis, St. Louis, Kansas City, Omaha, Denver, Chicago, St. Paul, Minneapolis, Toronto, Montreal, and New York. The map is surrounded by various game components: a deck of cards at the top, a row of numbered train tokens at the bottom, and several individual train tokens and cards scattered around the map. The entire scene is set against a white background.

# Planen

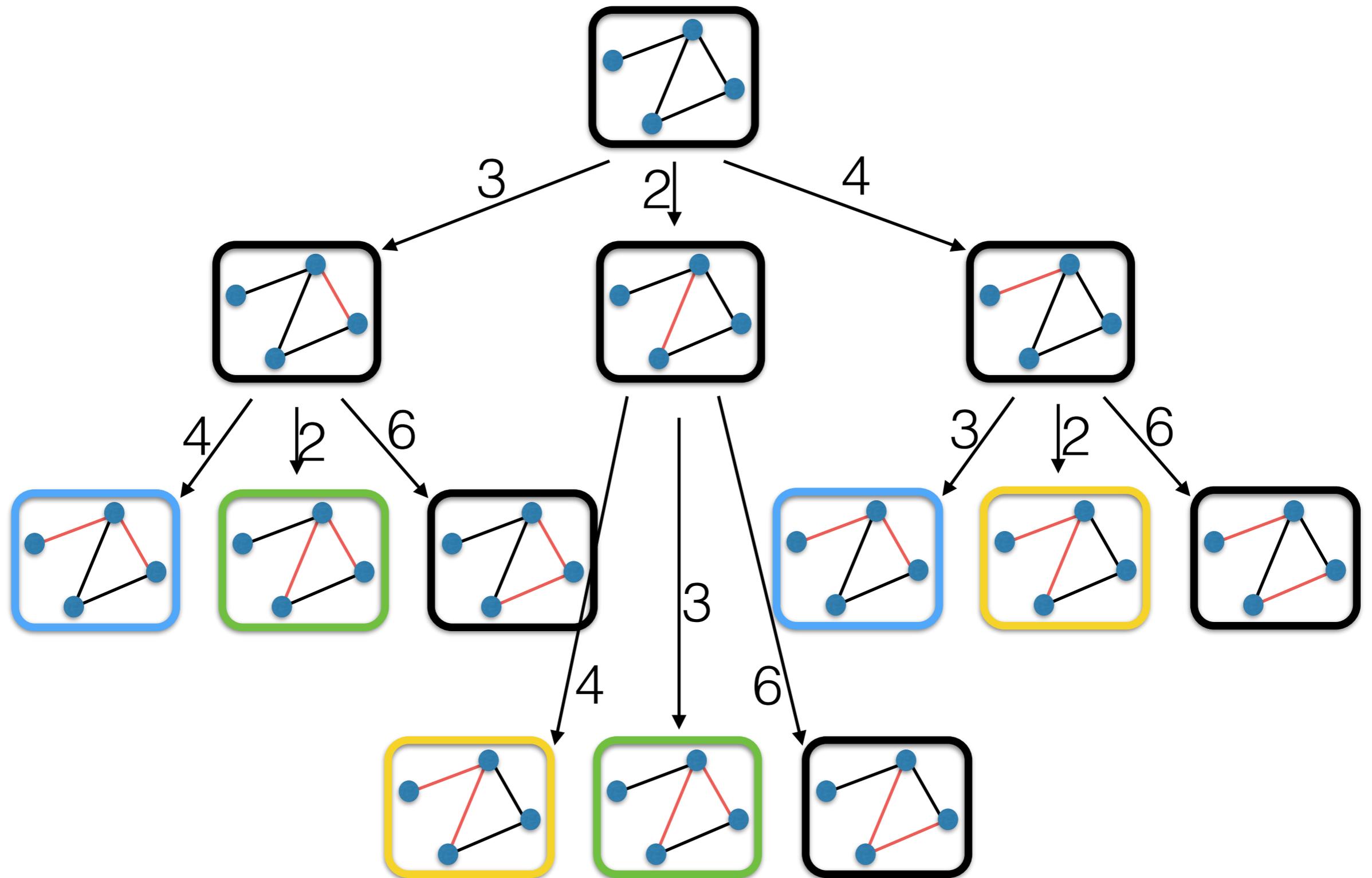
- TakeCard RED
- TakeCard BLUE
- BuildTrack 5
- TakeSecretCard
- TakeCard GREEN
- BuildTrack 13

Ein Plan ist eine  
Sequenz von Aktionen

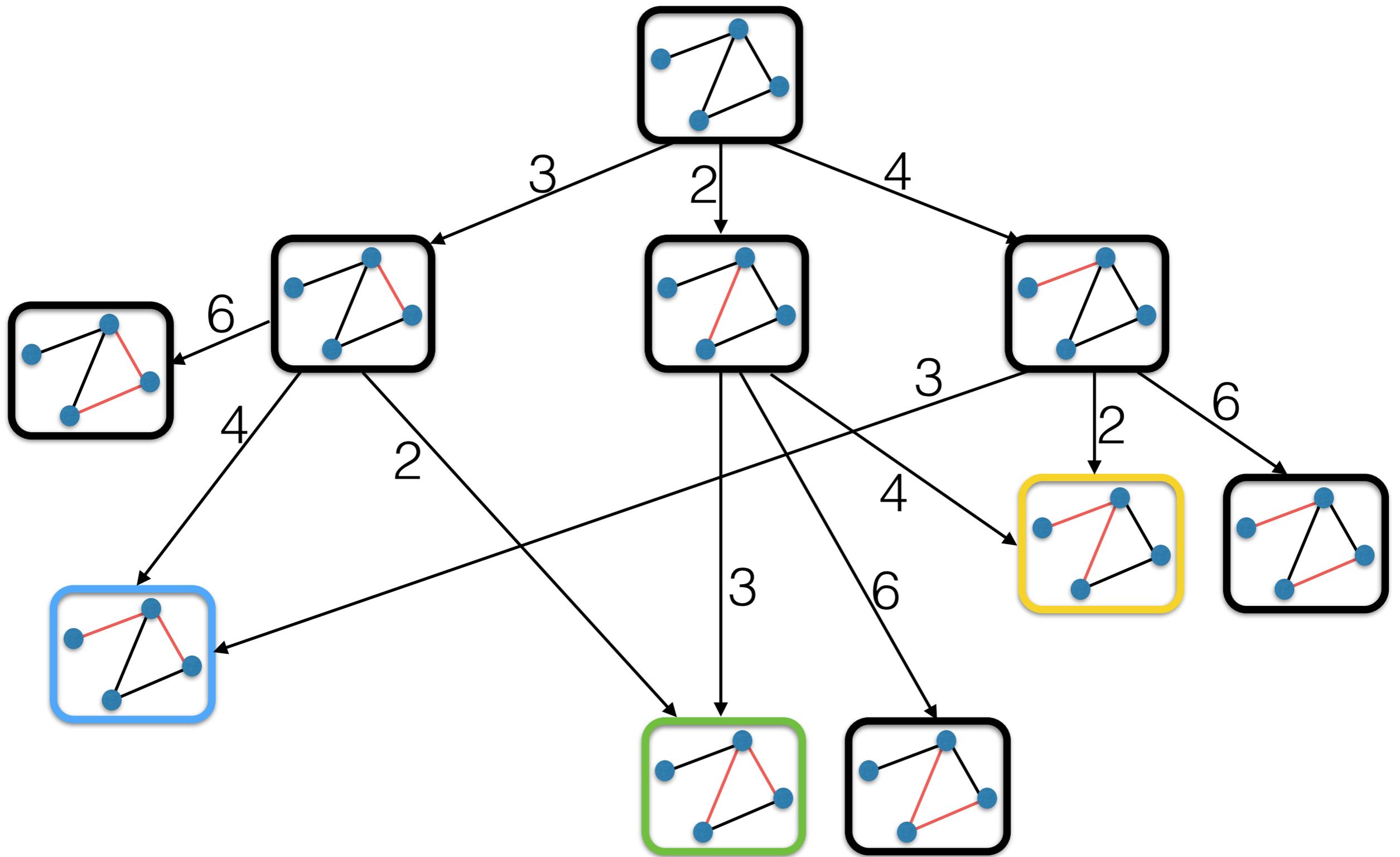
# Planning tree



# Planning tree



# Planning graph

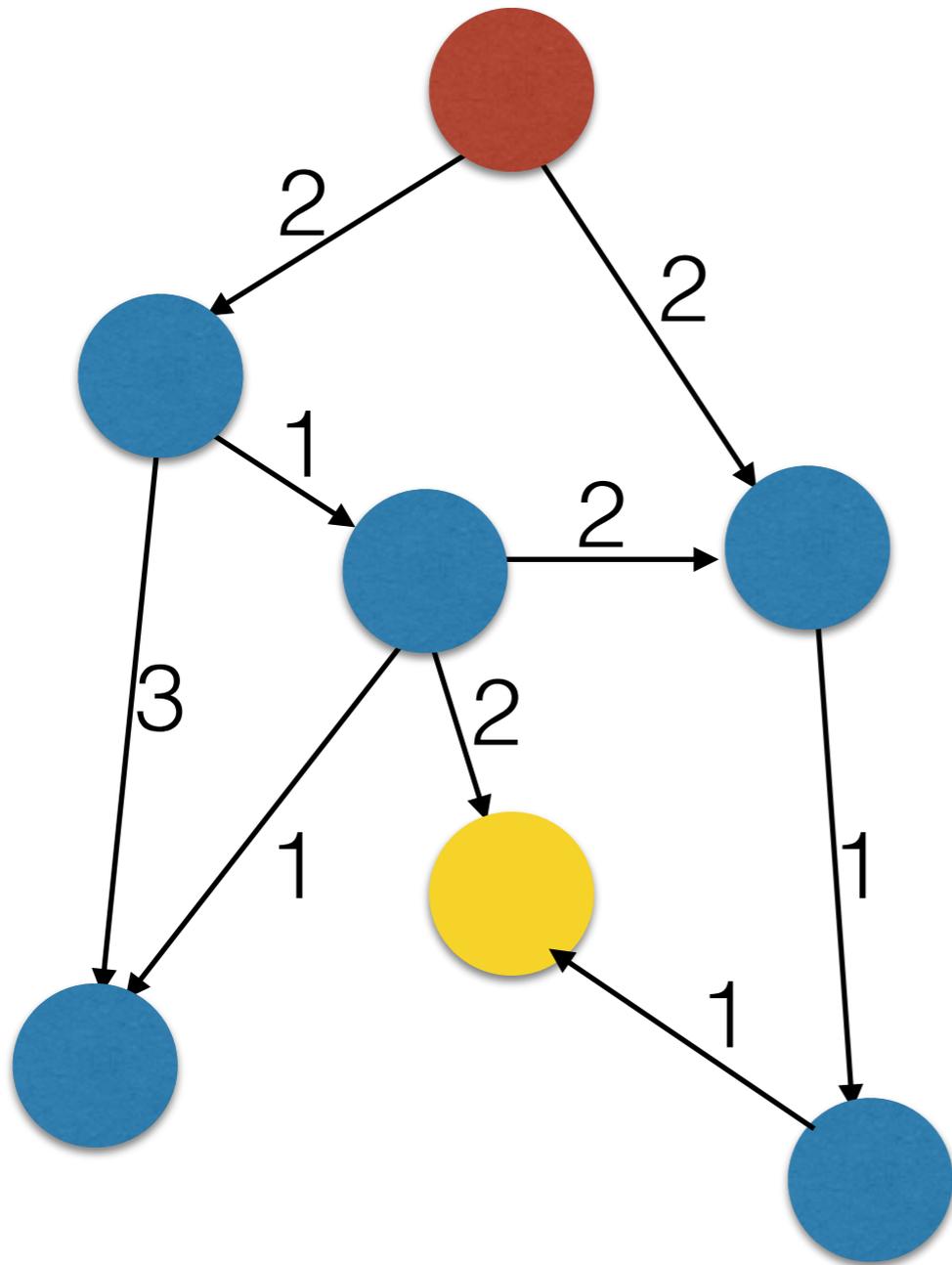


# Suche den optimalen Plan

- Der Startzustand ist der aktuelle Zustand
- Endzustände im Planning Graph sind Zustände, in denen ich gewonnen habe
- Der optimale Plan ist der kürzeste Pfad vom Startzustand zu einem Endzustand



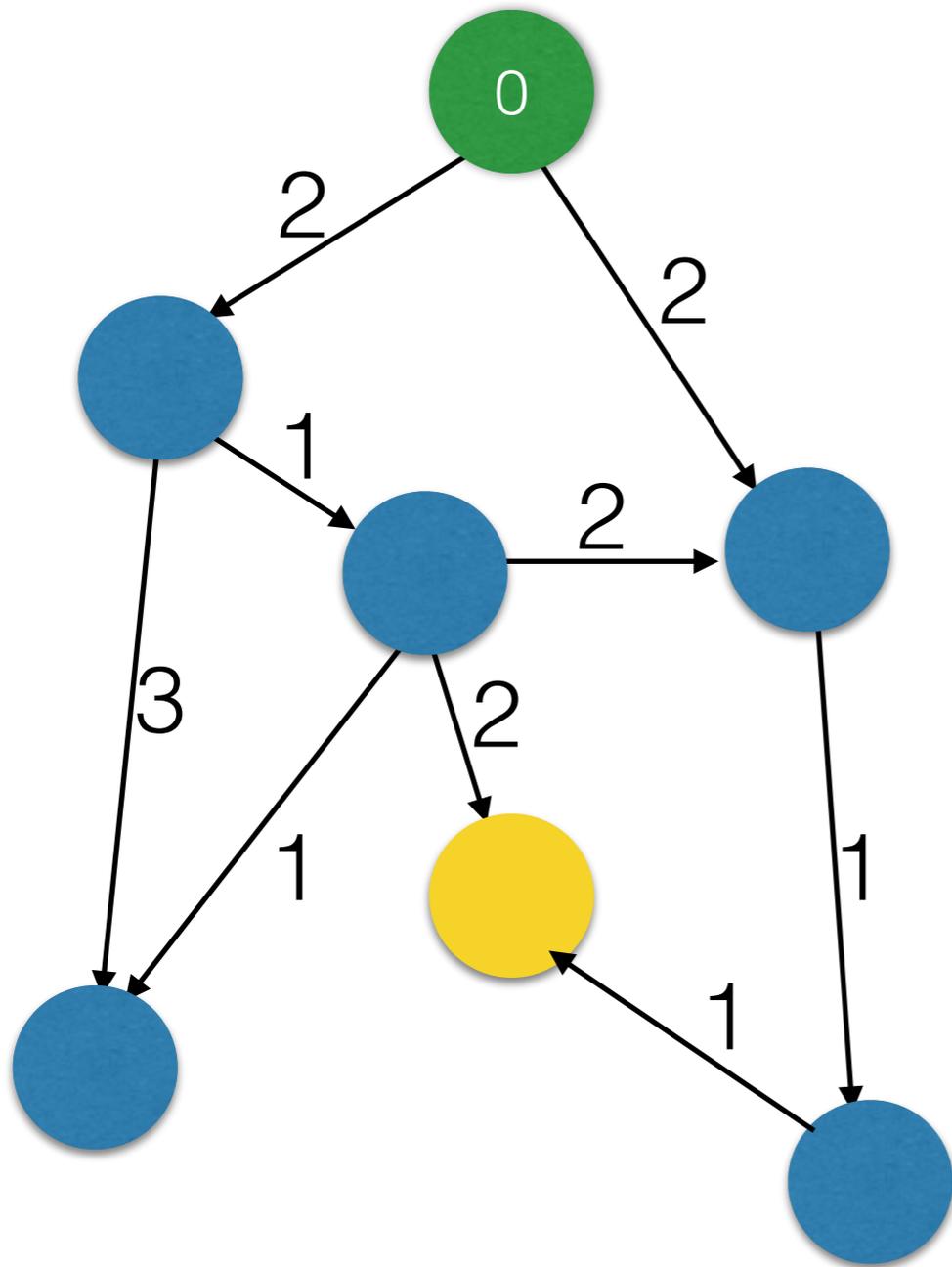
# - Algorithmus



- `q = new PriorityQueue()`
- Solange wie `q` nicht leer ist:
  - hole den besten Zustand `z` aus `q`
  - erzeuge alle Zustände `z'` aus `z`
  - füge alle `z'` zu `q` hinzu

# Dijkstra - Algorithmus

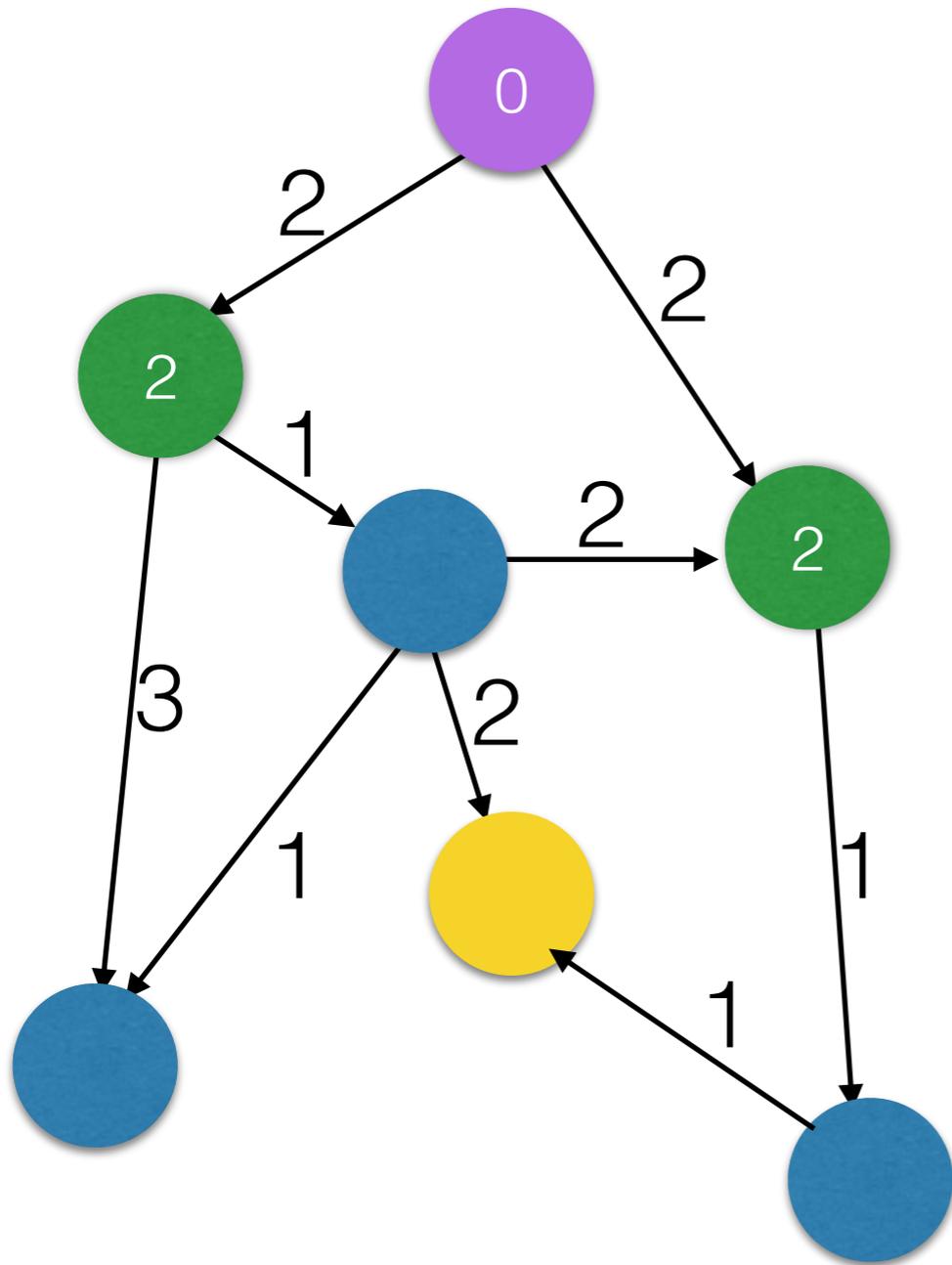
Sortiere die Queue nach Pfadkosten



- `q` = new  
PriorityQueue()
- Solange wie `q` nicht leer ist:
  - hole den besten Zustand `z` aus `q`
  - erzeuge alle Zustände `z'` aus `z`
  - füge alle `z'` zu `q` hinzu

# Dijkstra - Algorithmus

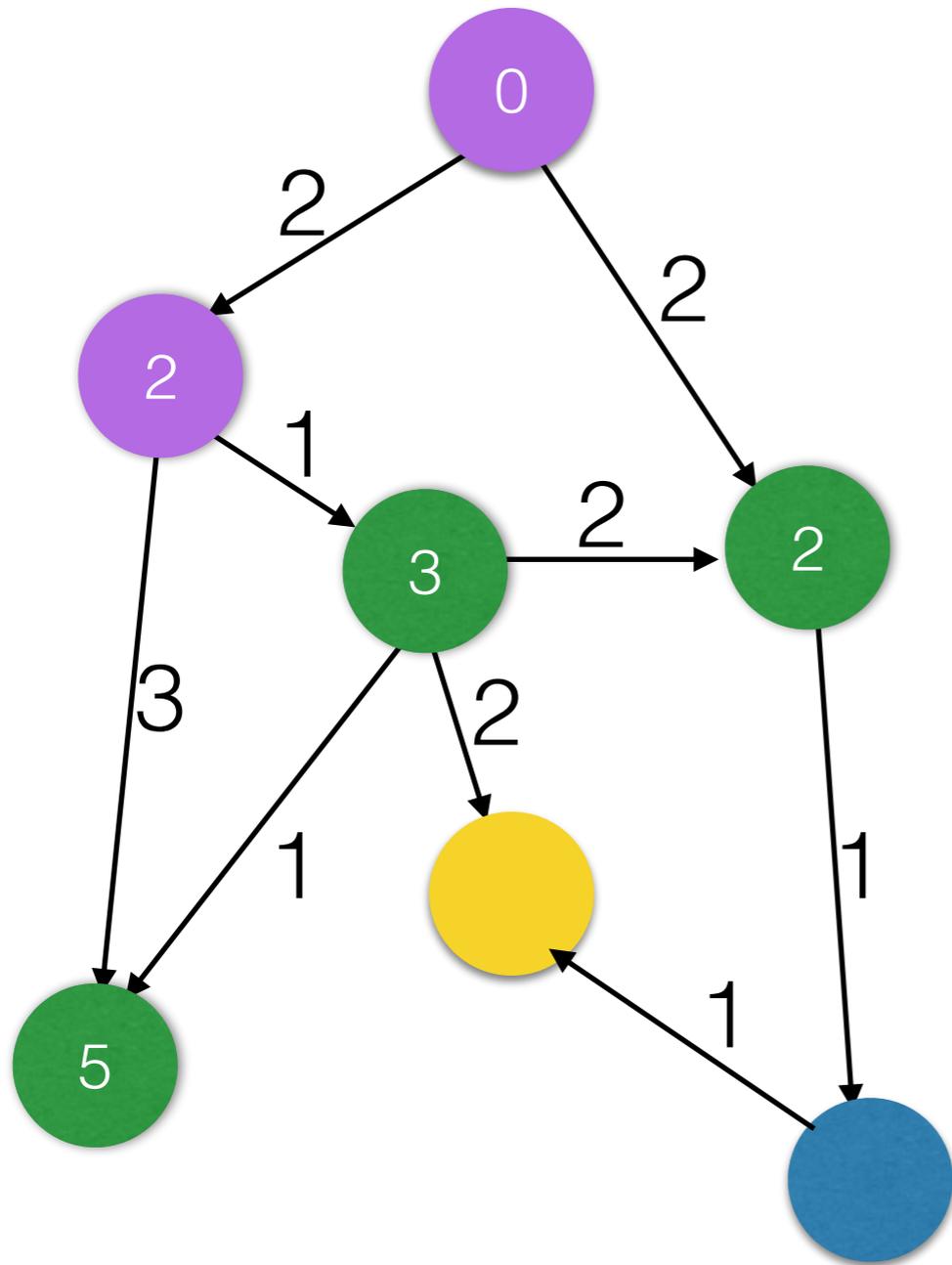
Sortiere die Queue nach Pfadkosten



- `q = new PriorityQueue()`
- Solange wie `q` nicht leer ist:
  - hole den besten Zustand `z` aus `q`
  - erzeuge alle Zustände `z'` aus `z`
  - füge alle `z'` zu `q` hinzu

# Dijkstra - Algorithmus

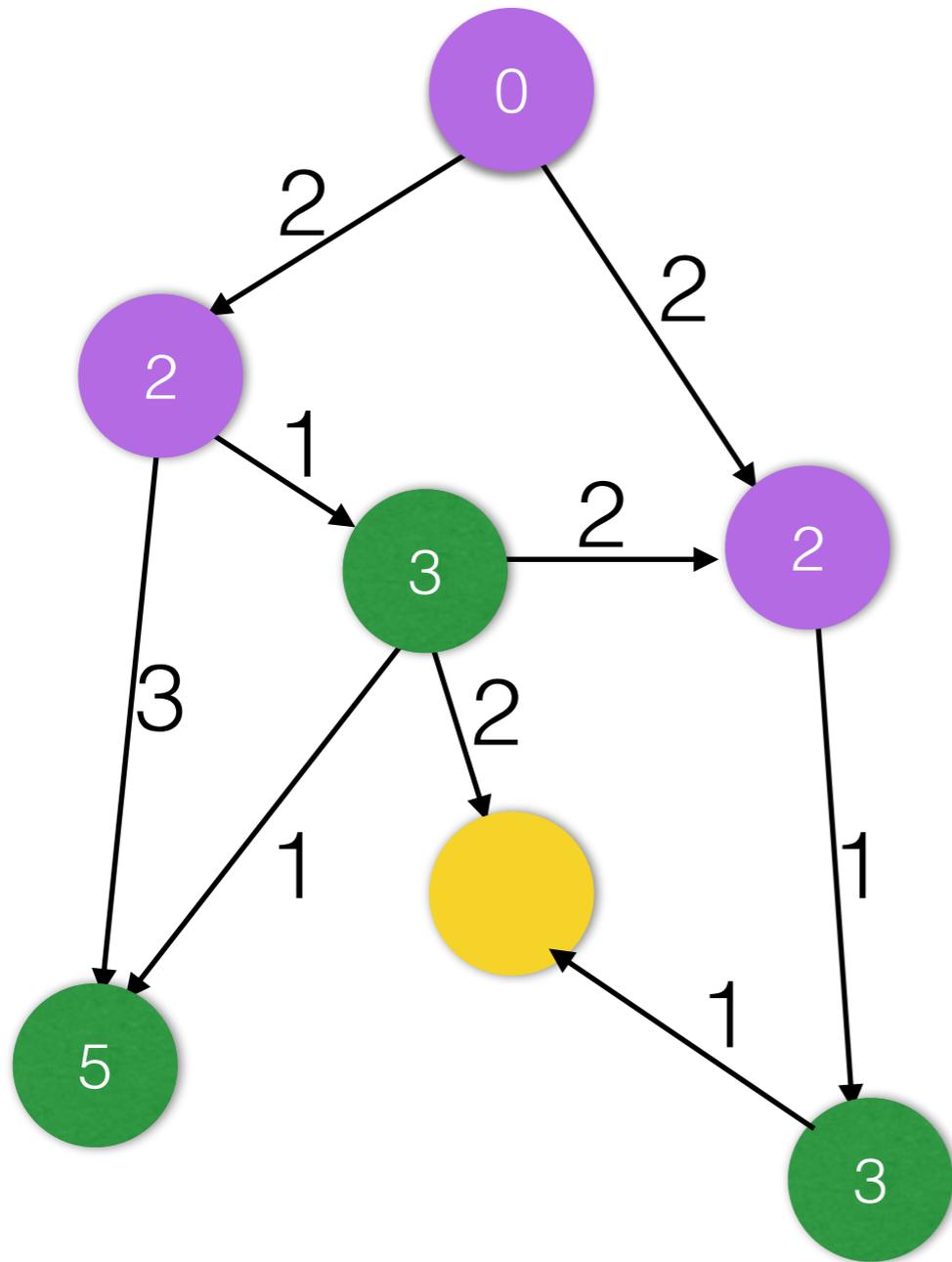
Sortiere die Queue nach Pfadkosten



- `q` = new  
PriorityQueue()
- Solange wie `q` nicht leer ist:
  - hole den besten Zustand `z` aus `q`
  - erzeuge alle Zustände `z'` aus `z`
  - füge alle `z'` zu `q` hinzu

# Dijkstra - Algorithmus

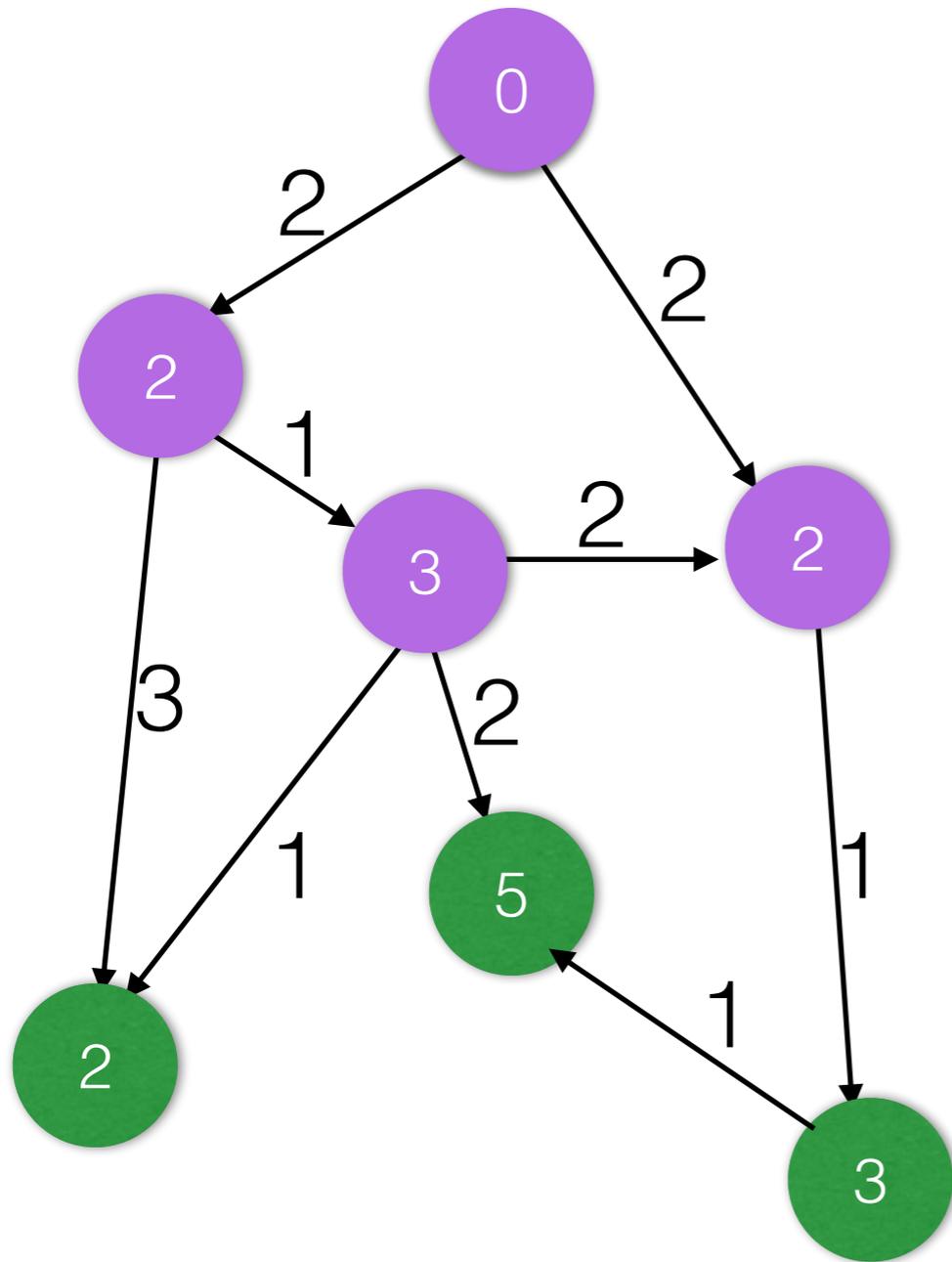
Sortiere die Queue nach Pfadkosten



- `q = new PriorityQueue()`
- Solange wie `q` nicht leer ist:
  - hole den besten Zustand `z` aus `q`
  - erzeuge alle Zustände `z'` aus `z`
  - füge alle `z'` zu `q` hinzu

# Dijkstra - Algorithmus

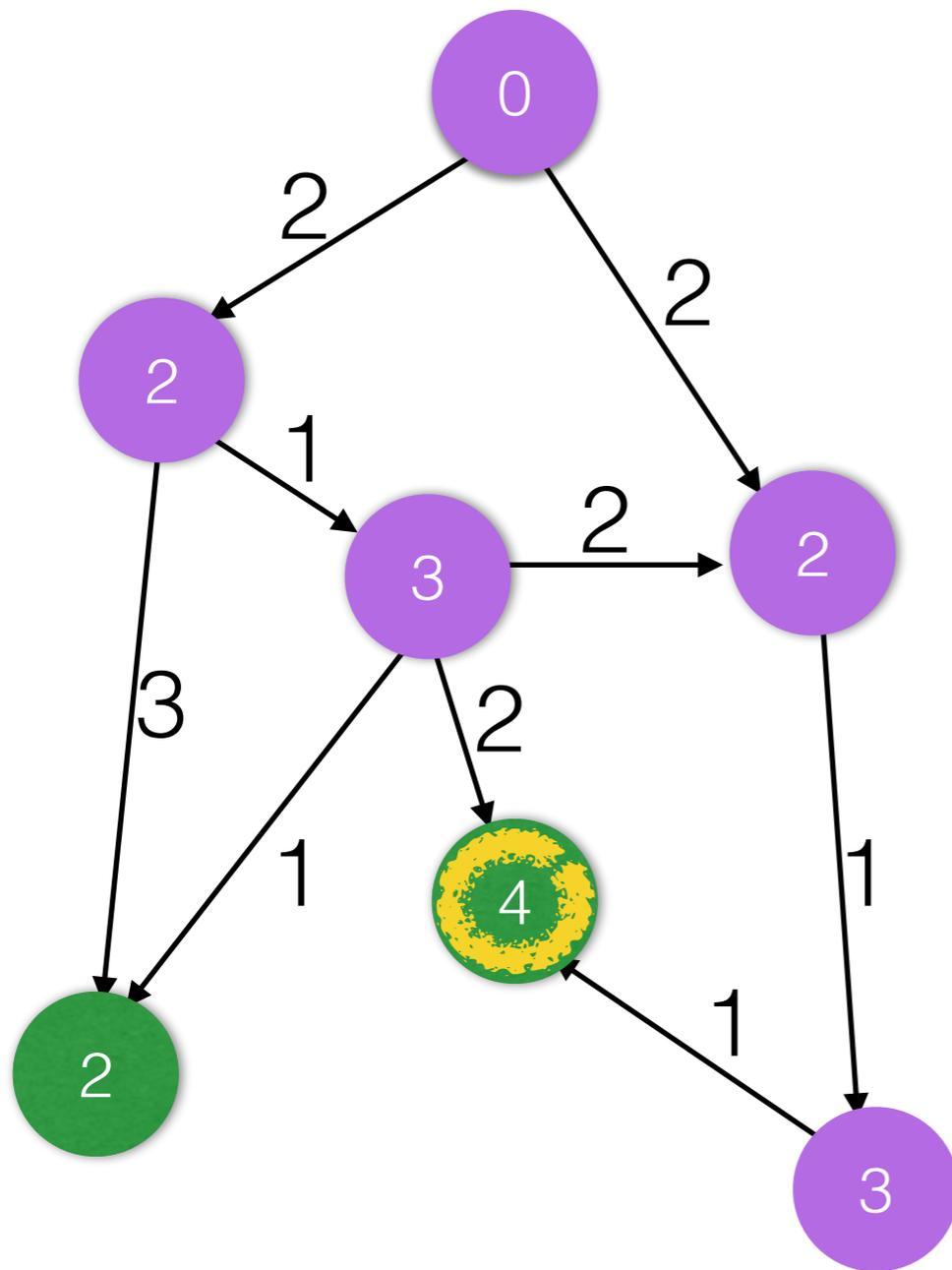
Sortiere die Queue nach Pfadkosten



- `q` = new  
PriorityQueue()
- Solange wie `q` nicht leer ist:
  - hole den besten Zustand `z` aus `q`
  - erzeuge alle Zustände `z'` aus `z`
  - füge alle `z'` zu `q` hinzu

# Dijkstra - Algorithmus

Sortiere die Queue nach Pfadkosten



- `q = new PriorityQueue()`
- Solange wie `q` nicht leer ist:
  - hole den besten Zustand `z` aus `q`
  - erzeuge alle Zustände `z'` aus `z`
  - füge alle `z'` zu `q` hinzu

# Dijkstra - Algorithmus

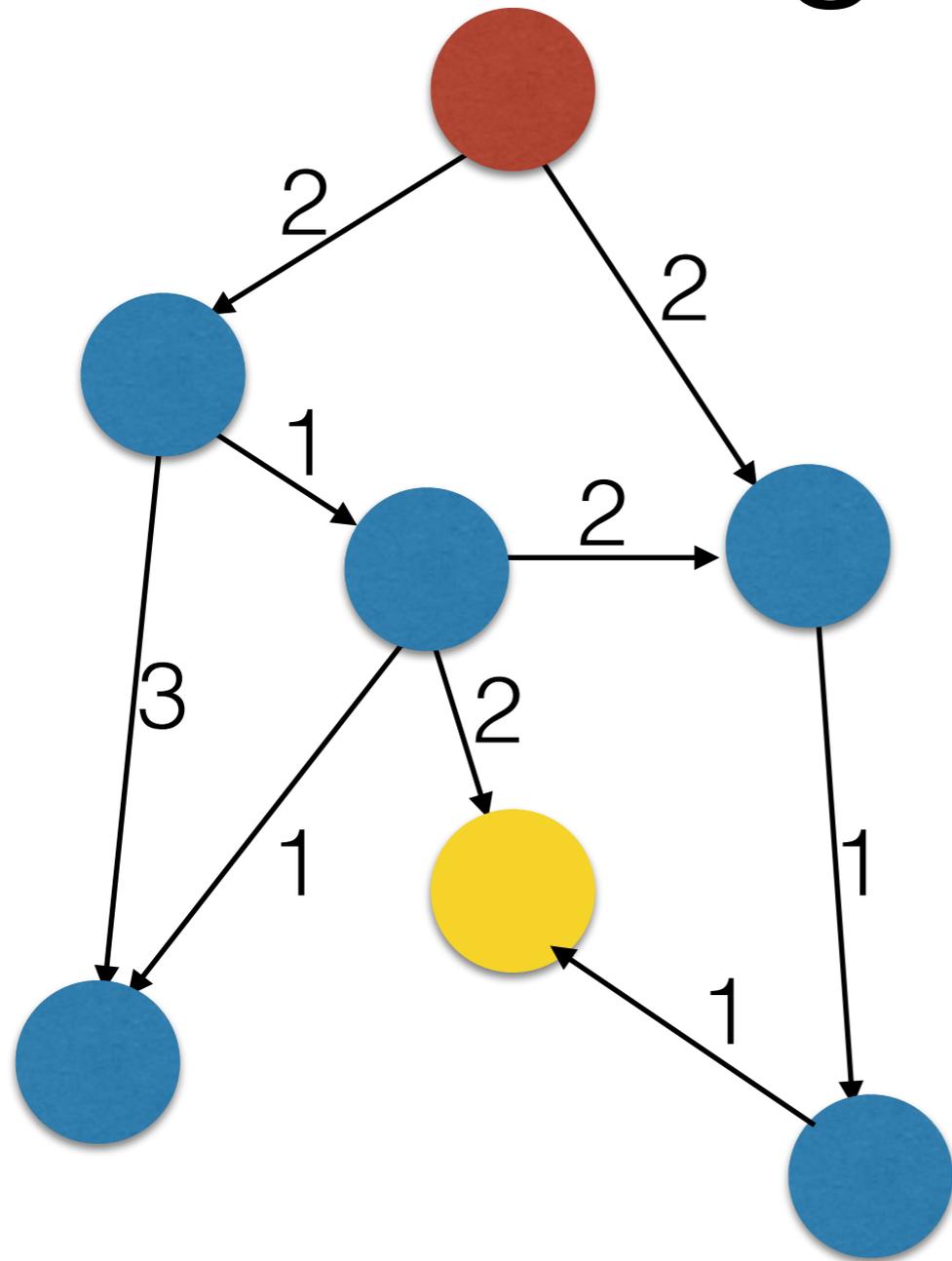
Sortiere die Queue nach  
Pfadkosten

Zum Planen ungeeignet,

1. da der ganze Planungsgraph im Speicher stehen muss.
2. da der Endzustand erreicht werden muss.

# Greedy Best First - Algorithmus

Sortiere die Queue nach Bewertungsfunktion



- $q$  = new PriorityQueue()
- Solange wie  $q$  nicht leer ist:
  - hole den besten Zustand  $z$  aus  $q$
  - wenn  $z$  der Endzustand ist, abbrechen
  - erzeuge alle Zustände  $z'$  aus  $z$
  - füge alle  $z'$  zu  $q$  hinzu

# Greedy Best First - Algorithmus

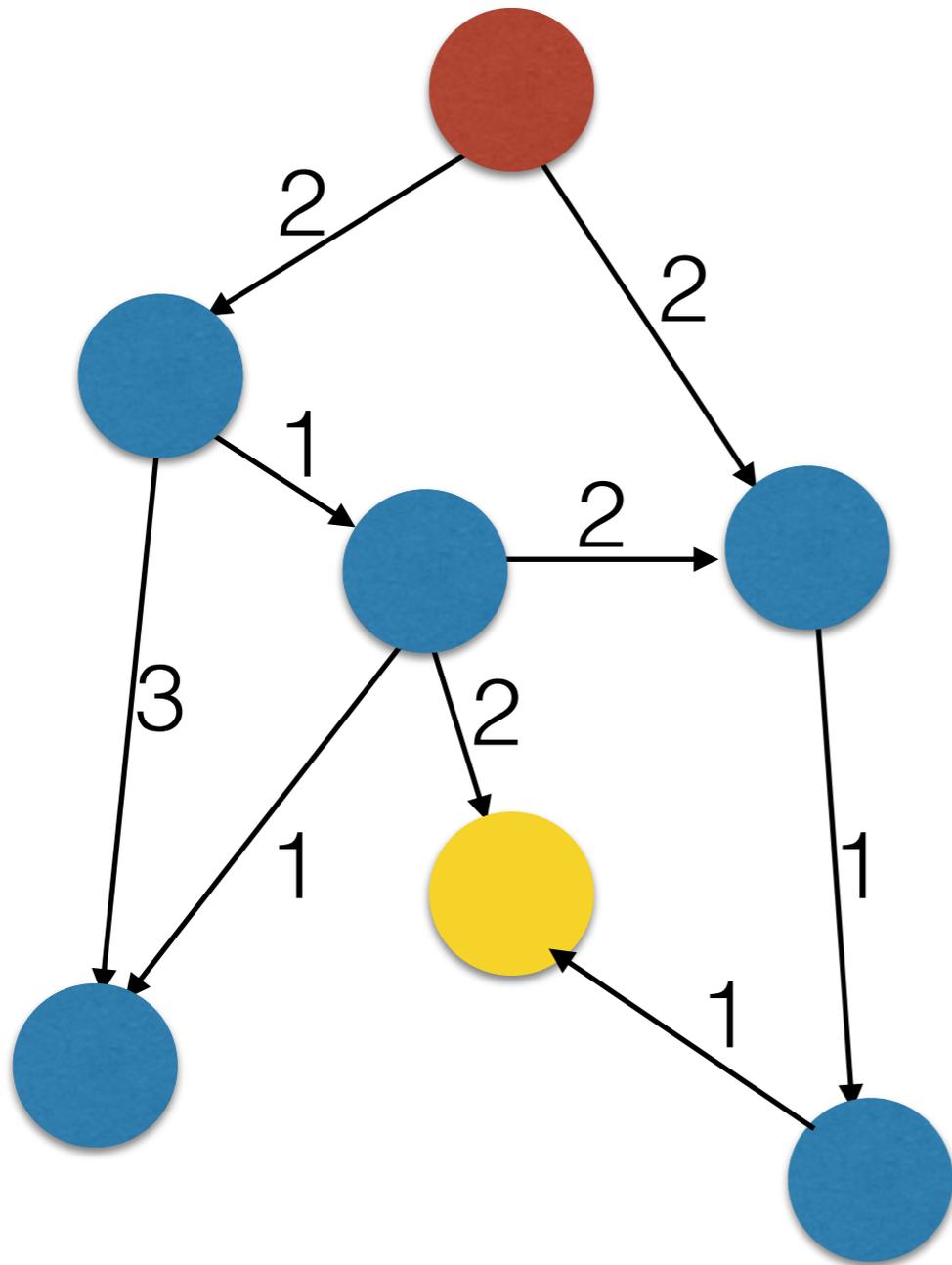
Sortiere die Queue nach  
Bewertungsfunktion

Besser, weil statt dem Endzustand der Zustand mit der besten Bewertung gesucht wird.

Aber keine Garantie, dass der kürzeste Pfad zu dem besten Zustand gefunden wird.

# A\* - Algorithmus

Sortiere die Queue nach  
Bewertungsfunktion +  
Pfadkosten



- $q$  = new  
PriorityQueue()
- Solange wie  $q$  nicht  
leer ist:
  - hole den besten  
Zustand  $z$  aus  $q$
  - wenn  $z$  der  
Endzustand ist,  
abbrechen
  - erzeuge alle  
Zustände  $z'$  aus  $z$
  - füge alle  $z'$  zu  $q$   
hinzu

# A\* - Algorithmus

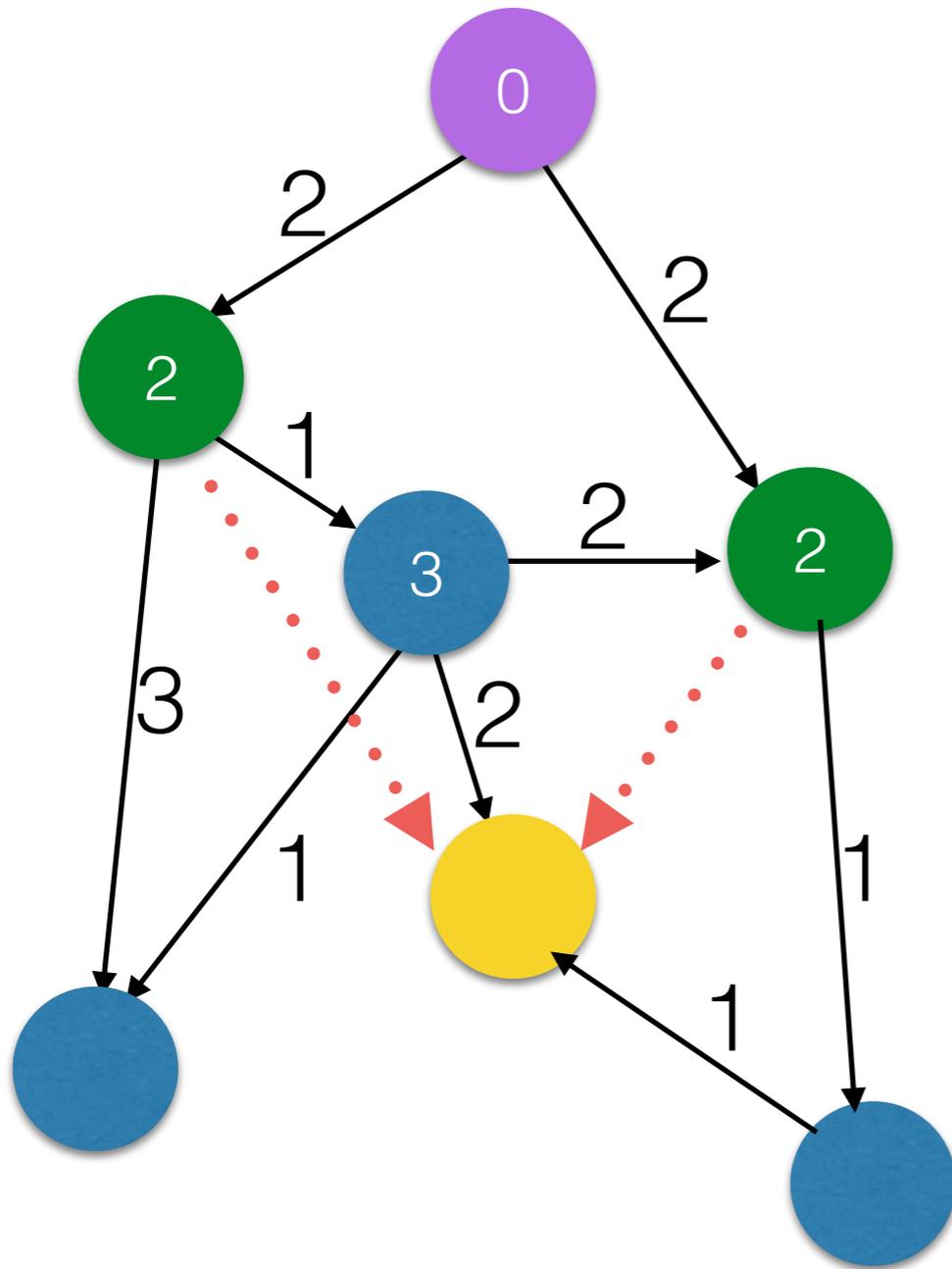
Sortiere die Queue nach  
Bewertungsfunktion +  
Pfadkosten

Besser, weil statt dem Endzustand der Zustand mit der besten Bewertung gesucht wird.

Garantiert, dass der kürzeste Pfad gefunden wird, wenn die Bewertungsfunktion die Pfadkosten überschätzt.

# A\* - Algorithmus

Sortiere die Queue nach  
Bewertungsfunktion +  
Pfadkosten



- `q` = new  
PriorityQueue()
- Solange wie `q` nicht  
leer ist:
  - hole den besten  
Zustand `z` aus `q`
  - wenn `z` der  
Endzustand ist,  
abbrechen
  - erzeuge alle  
Zustände `z'` aus `z`
  - füge alle `z'` zu `q`  
hinzu