近道をみつける

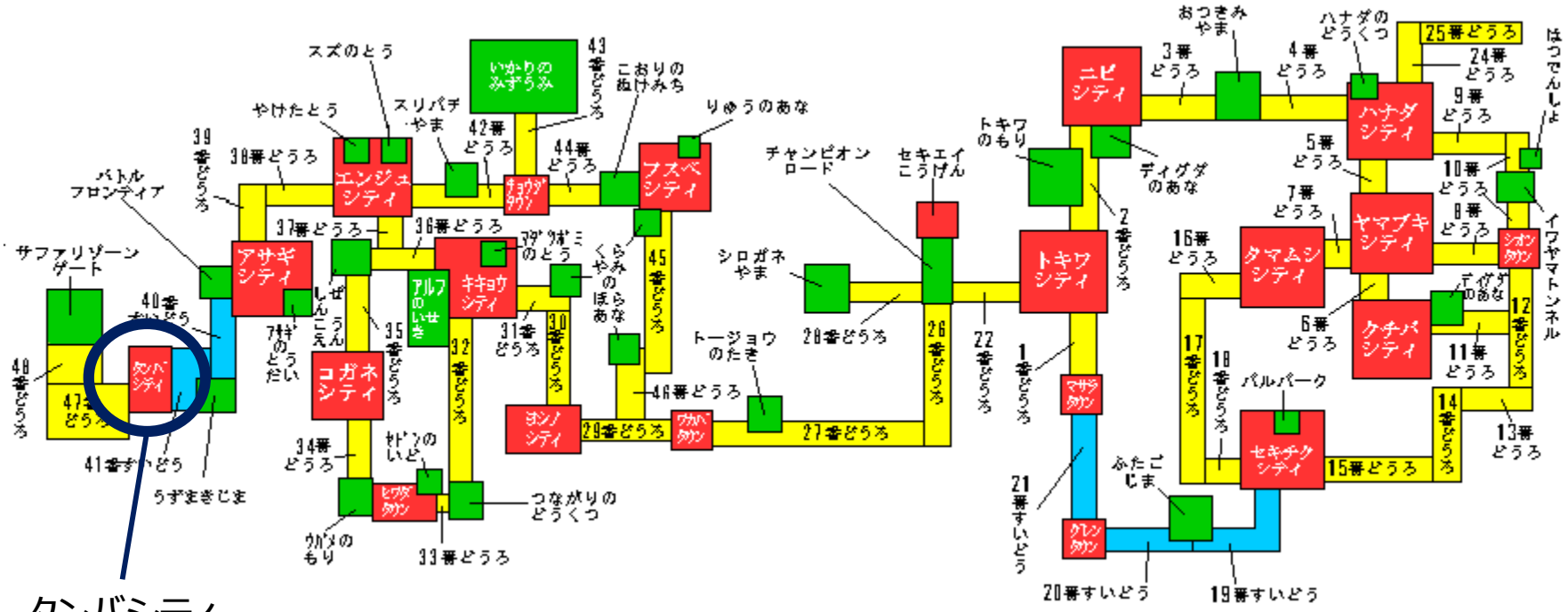
経路探索問題を解く

カーナビゲーションシステム

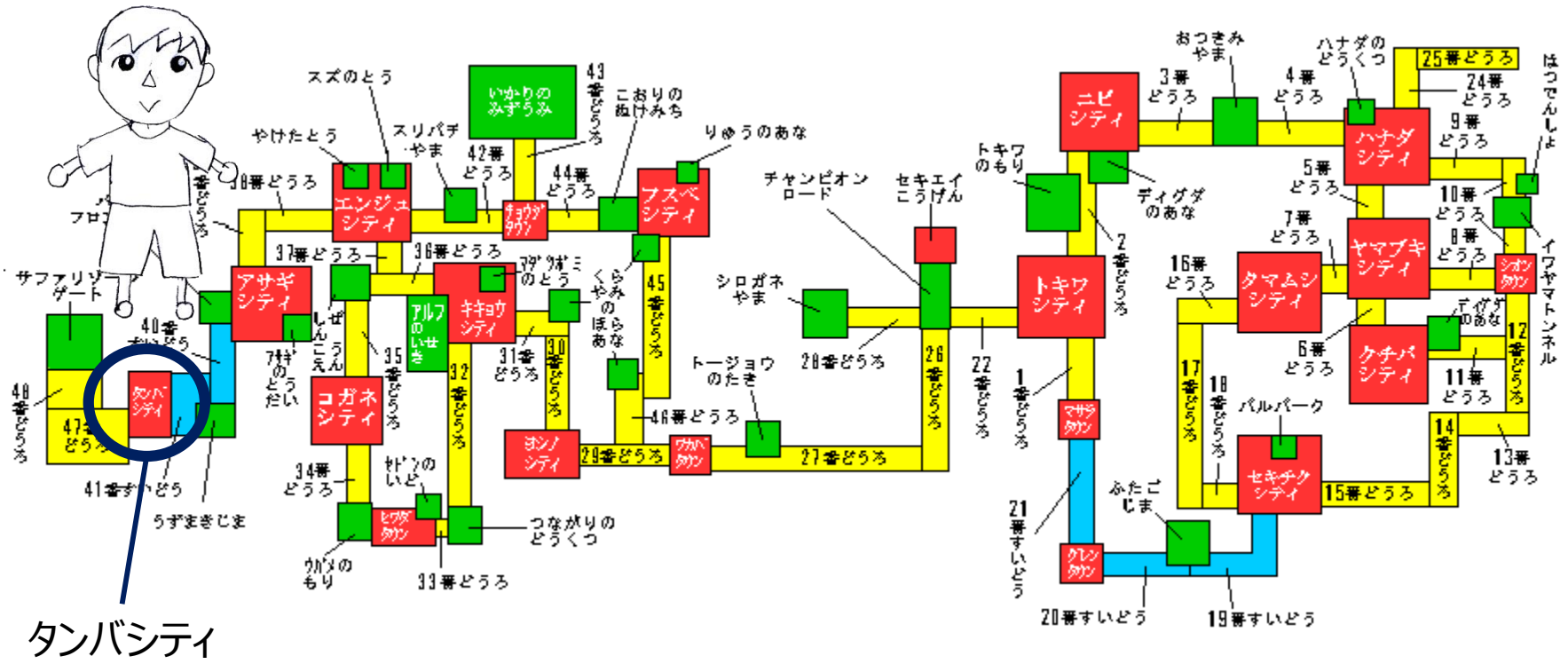
- 目的地をセット

- 時間や距離を考慮して最適なルートの選択
  →コンピュータで算出

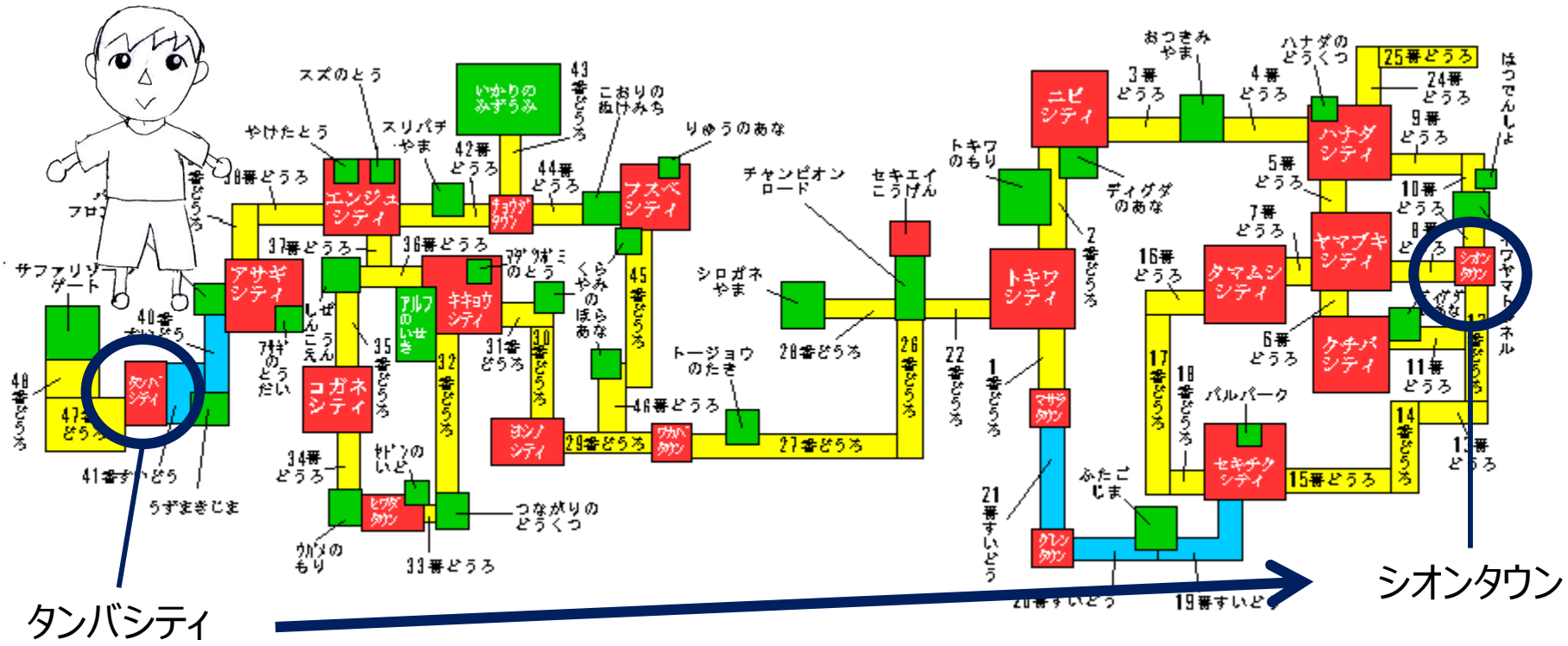
お墓参りに行きたい！



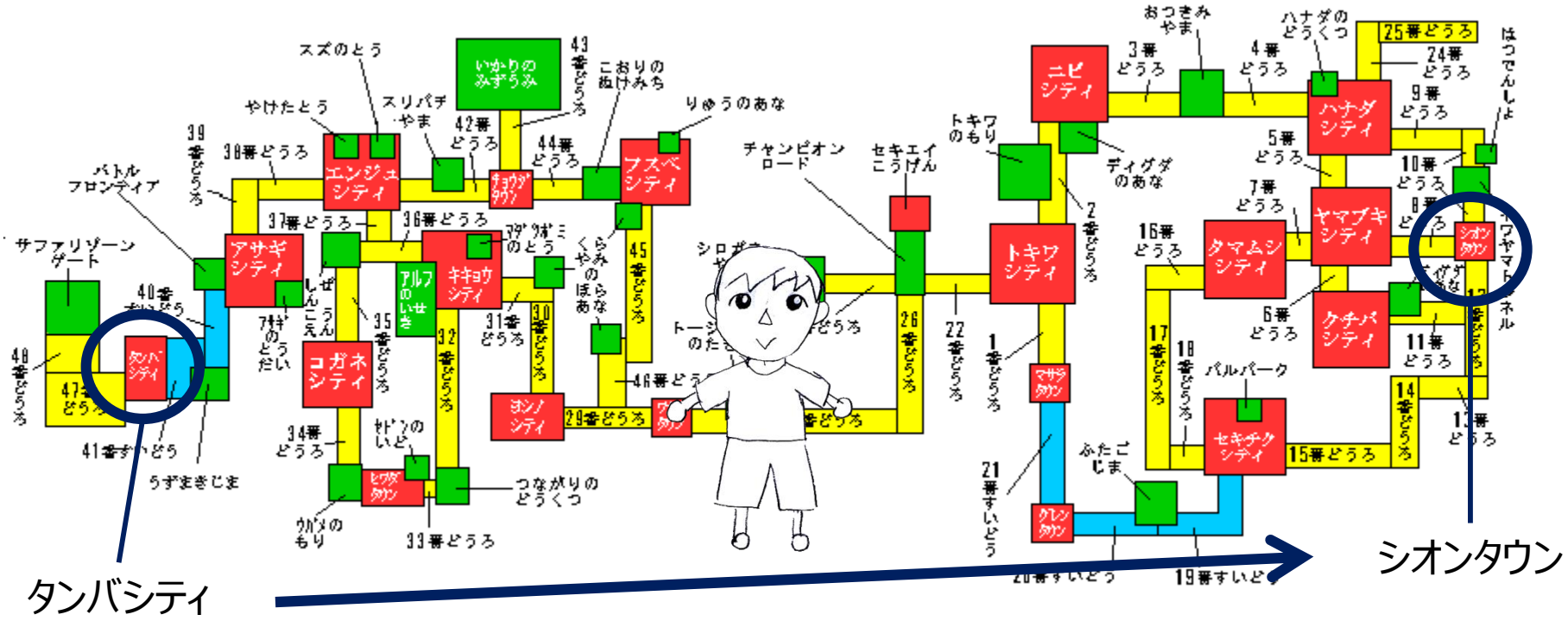
お墓参りに行きたい！



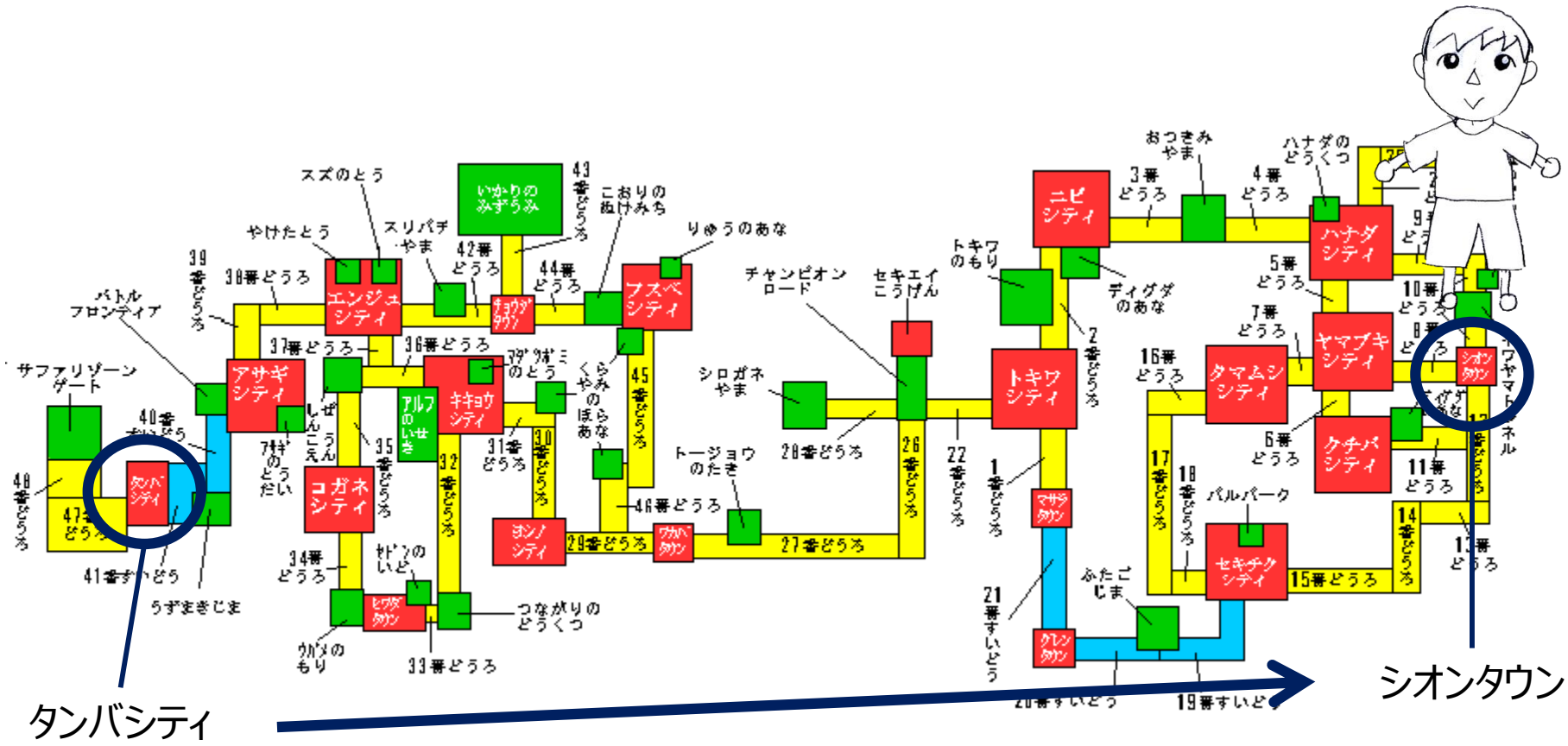
お墓参りに行きたい！


タンバシティ
シオンタウン
ニビシティ
トキワシティ
ハナダシティ
ヤマブキシティ
タマムシシティ
クチバシティ
セキチクシティ
マサラタウン
シオンタウン
フスベシティ
エンジュシティ
キキョウシティ
アサギシティ
コガネシティ
ヨシノシティ
いかりのみずうみ
チョウジタウン
ワカバタウン
ヒワダタウン
クレンタウン
パルパーク
ディグダのあな
トージョウのたき
シロガネやま
チャンピオンロード
セキエイこうげん
りゅうのあな
こおりのぬけみち
スリバチやま
やけたとう
スズのとう
おつきみやま
ハナダのどうくつ
はつでんしょ
ふたごじま
つながりのどうくつ
うずまきじま
ウバメのもり
サファリゾーン

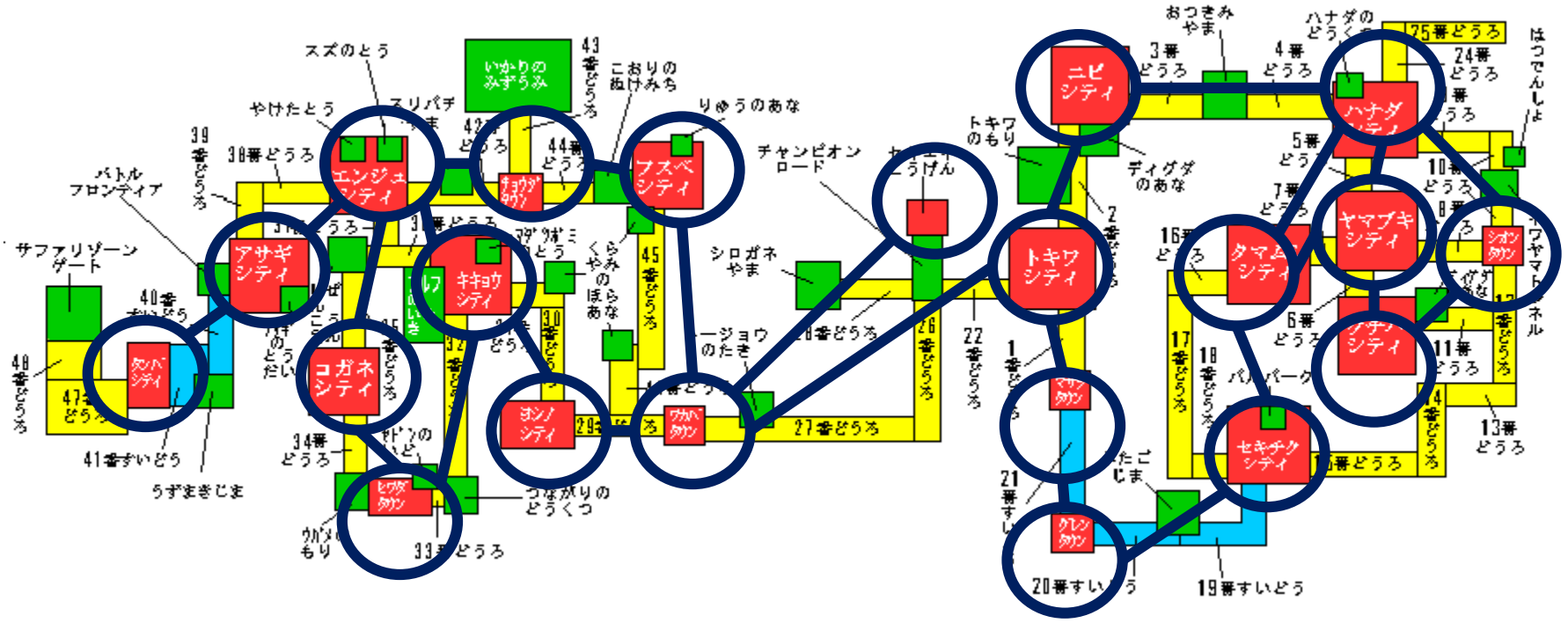
お墓参りに行きたい！



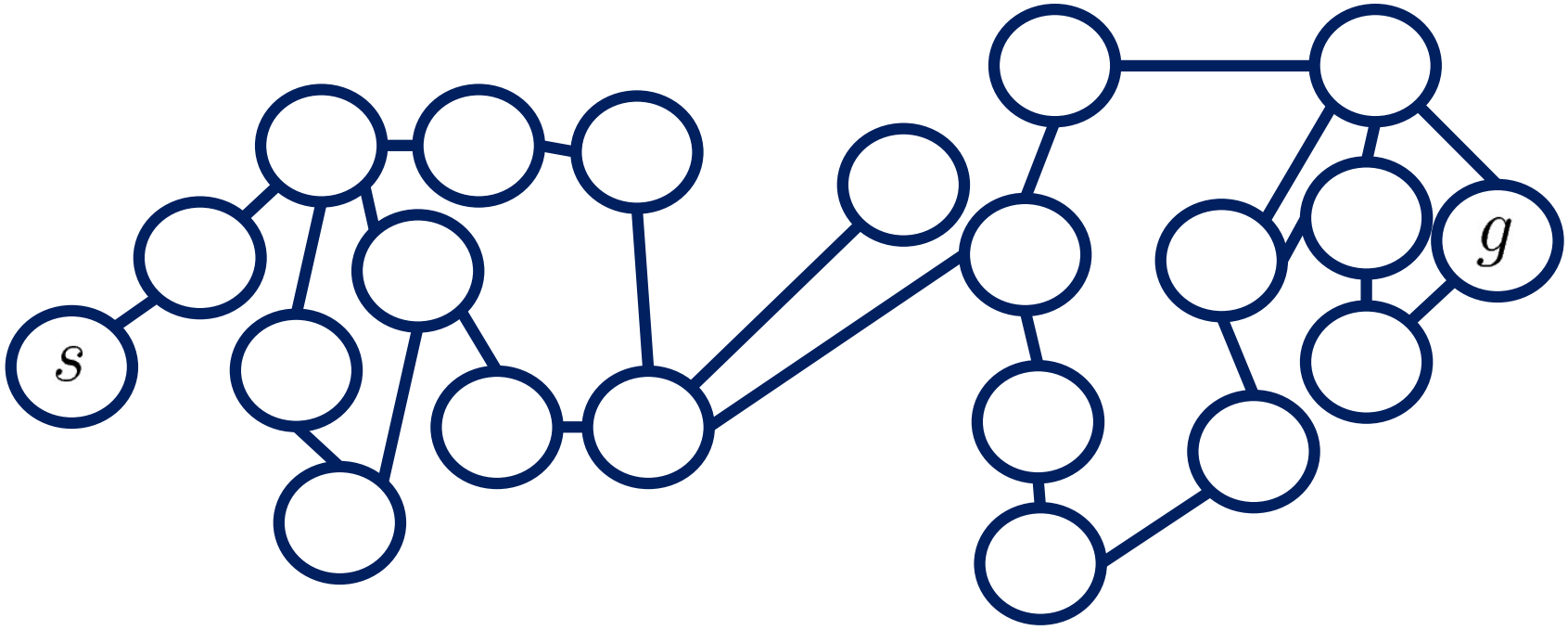
お墓参りに行きたい！



地図から「グラフ」へ


ニビシティ
トキワシティ
ハナダシティ
ヤマブキシティ
タマムシシティ
セキチクシティ
フスベシティ
エンジュシティ
キキョウシティ
アサギシティ
コガネシティ
タンバシティ
ヨシノシティ
ワカバタウン
マサラタウン
クチバタウン
ヒワダタウン
シオンタウン
いかりのみずうみ
スズのとう
やけたとう
こおりのぬけみち
りゅうのあな
チャンピオンロード
シロガネやま
おつきみやま
ディグダのあな
トキワのもり
ハナダのどうくつ
はつでんしょ
バトルフロンティア
サファリゾーンゲート
パークバトル
つながりのどうくつ
うずまきじま
たごじま
ウバメのもり
1番どうろ
2番どうろ
3番どうろ
4番どうろ
5番どうろ
6番どうろ
7番どうろ
8番どうろ
9番どうろ
10番どうろ
11番どうろ
12番どうろ
13番どうろ
14番どうろ
15番どうろ
16番どうろ
17番どうろ
18番どうろ
19番すいどう
20番すいどう
21番すいどう
22番どうろ
24番どうろ
25番どうろ
26番どうろ
27番どうろ
28番どうろ
29番どうろ
30番どうろ
31番どうろ
32番どうろ
33番どうろ
34番どうろ
35番どうろ
36番どうろ
37番どうろ
38番どうろ
39番どうろ
40番どうろ
41番すいどう
42番どうろ
43番どうろ
44番どうろ
45番どうろ
46番どうろ
47番どうろ
48番どうろ

地図から「グラフ」へ

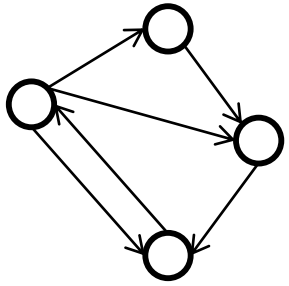


ところで、「グラフ」とは・・・？

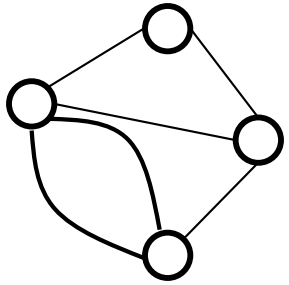
- 頂点（ノード・節点）と辺（エッジ・枝）からなる

  →辺の形や、頂点以外の場所での交差は気にしないことが多い

- 辺に向きが**ある**グラフと**ない**グラフがある



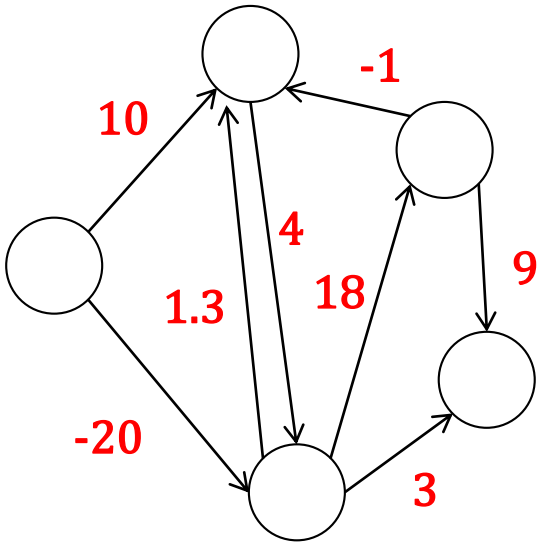
有向グラフ



無向グラフ

- 辺に**「重み」**がつくことがある

重みつき有向グラフの例



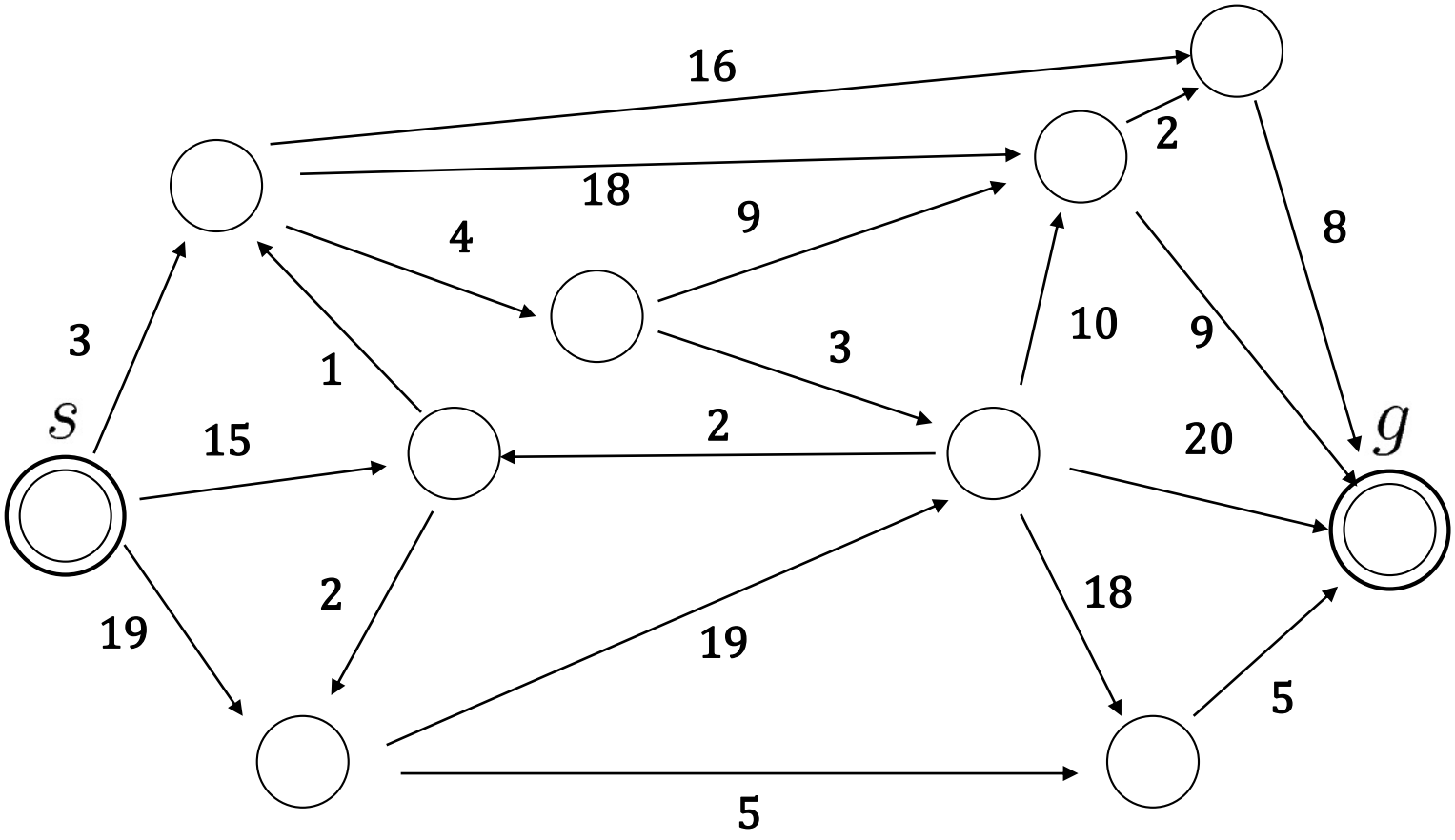
「重み」とは・・・

距離
運賃
材料の量
時間
利益　　etc.

といったものを場合に応じて
色々なものを表すことがある

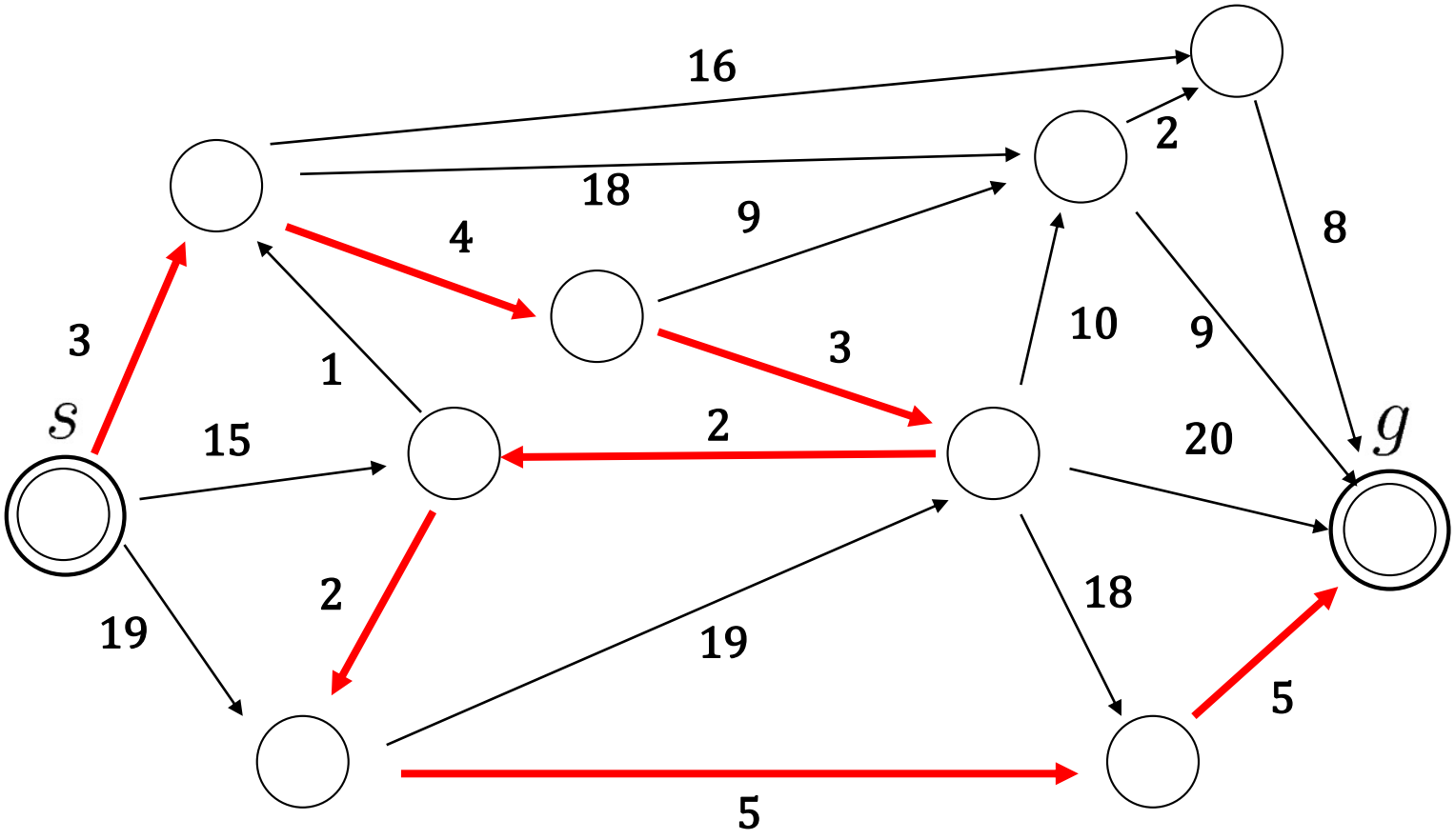
問題を解こう！

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



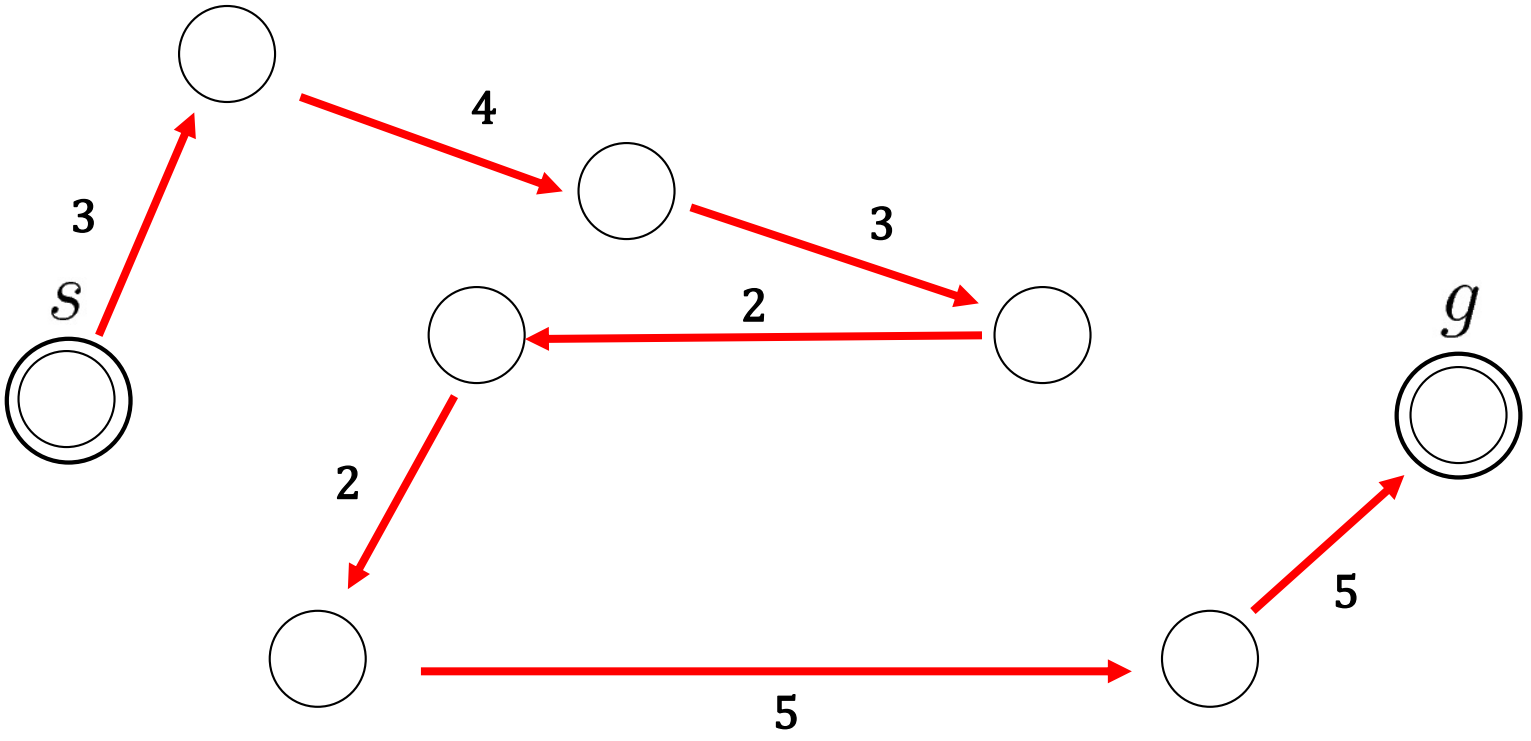
問題を解こう！

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ．
（辺の重みは頂点間の距離を表す．）



問題を解こう！

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）

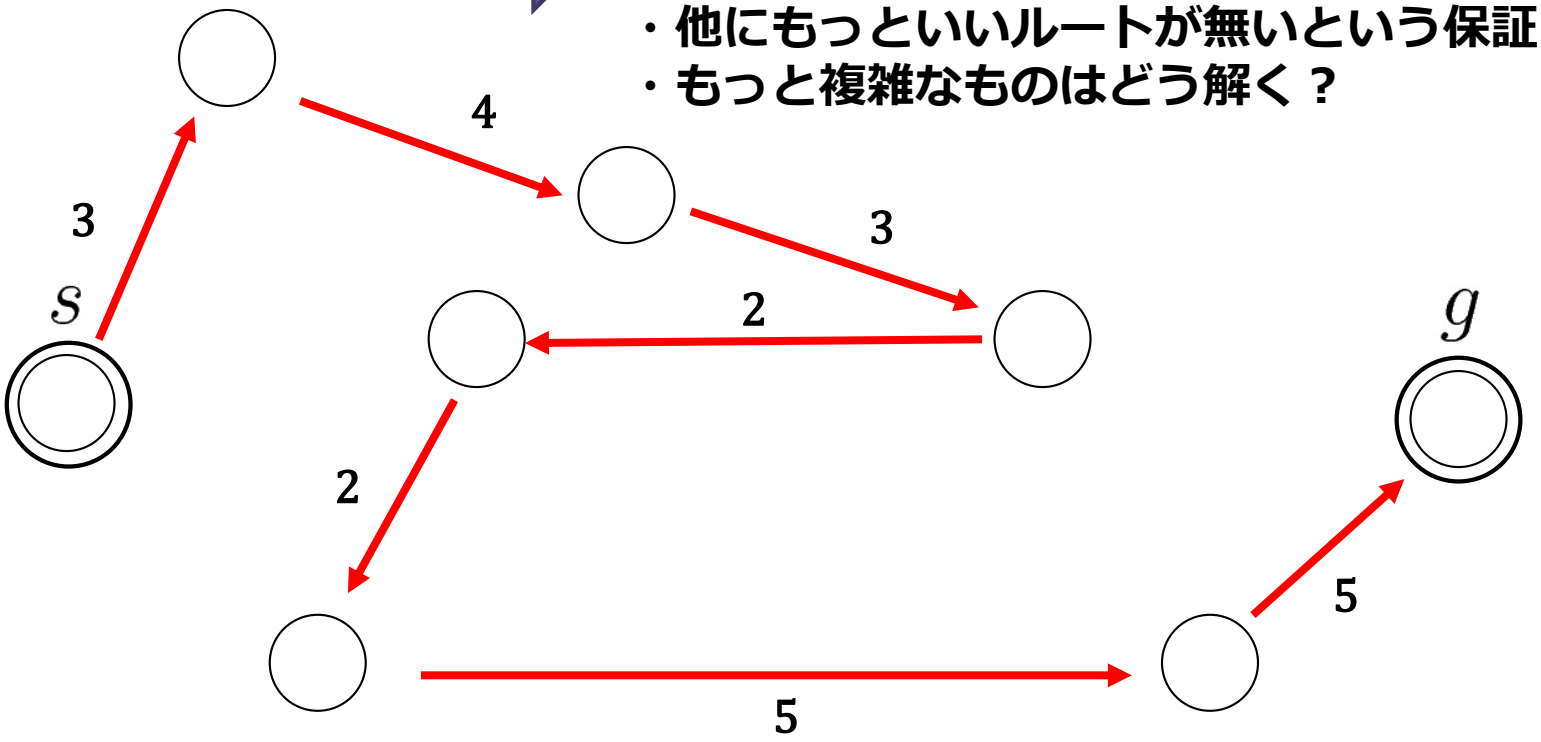


問題を解こう！

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）

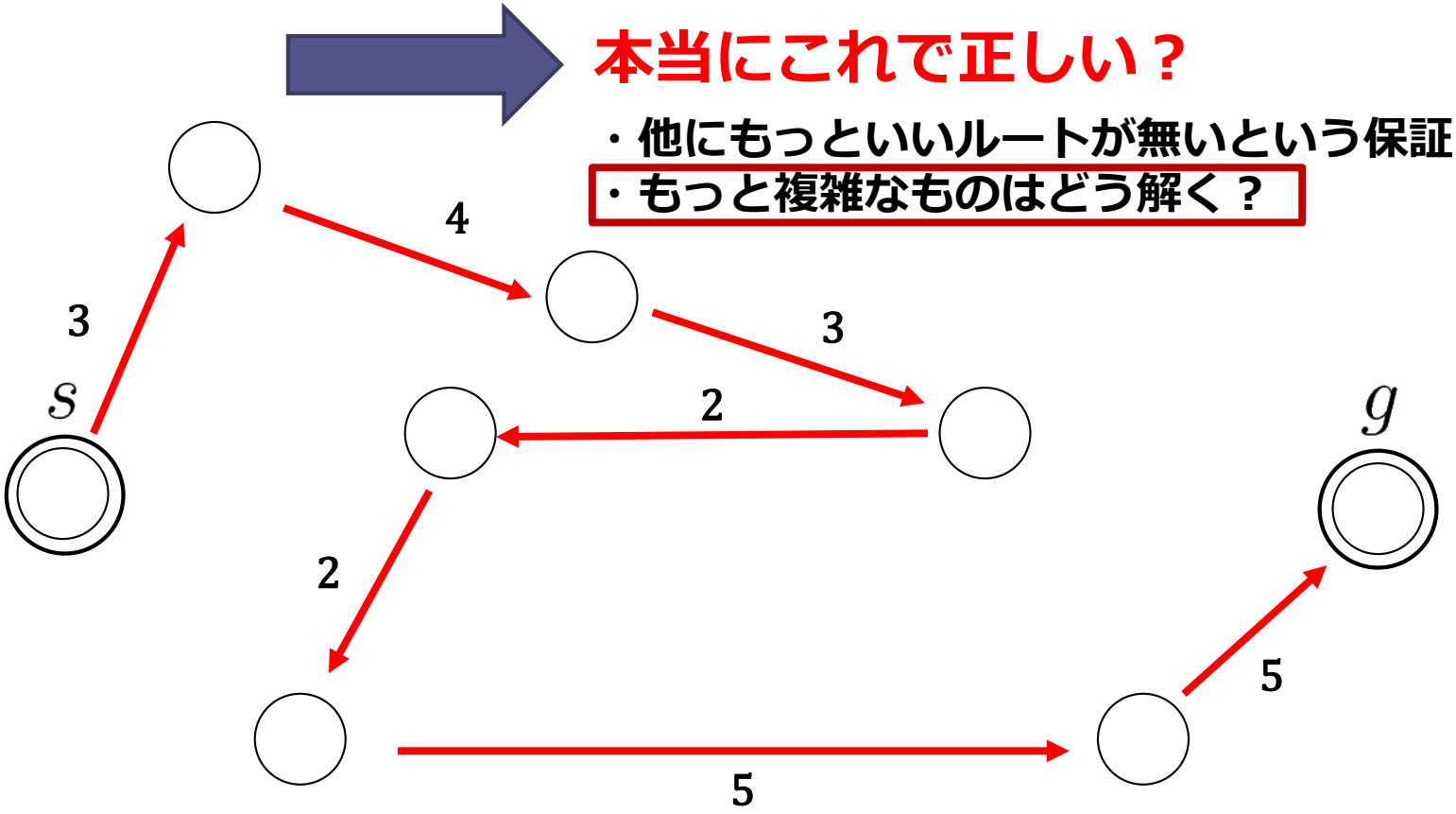
本当にこれで正しい？

・他にもっといいルートが無いという保証は？
・もっと複雑なものはどう解く？



問題を解こう！

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



問題を解くということ

- 「最適解である」という保証
  →しらみ潰しに調べればとりあえず解決
  →手計算では**限界**がある

- 複雑な問題（辺や頂点がとても多い、等）
  →これも手計算では**限界**がある

問題を解くということ

- 「最適解である」という保証
  →しらみ潰しに調べればとりあえず解決
  →手計算では**限界**がある

- 複雑な問題（辺や頂点がとても多い、等）
  →これも手計算では**限界**がある

コンピュータで解こう！

問題を解くということ

- 「最適解である」という保証
  →しらみ潰しに調べればとりあえず解決
  →手計算では**限界**がある

- 複雑な問題（辺や頂点がとても多い、等）
  →これも手計算では**限界**がある

コンピュータで解こう！

→それでもやっぱり**限界**はある
（しらみ潰しは莫大な時間がかかる）

コンピュータに解かせるために

- 解くための手順（アルゴリズム）を考える
  →手順がわからなきゃ解かせようがない

手順は実行の効率や**計算量**に影響

- コンピュータに指令をだす
  →コンピュータがわかる言葉（プログラム言語）で指令しなければならない　＝　この指令が「プログラム」

指令の出し方は実行の効率や**速度**に影響

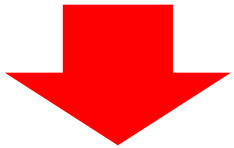
コンピュータに解かせるために

- 解くための手順（アルゴリズム）を考える
  →手順がわからなきゃ解かせようがない

手順は実行の効率や**計算量**に影響

- コンピュータに指令をだす
  →コンピュータがわかる言葉（プログラム言語）で指令しなければならない　＝　この指令が「プログラム」

指令の出し方は実行の効率や**速度**に影響



できるだけ**速く**て（時間は限られている）、
使用メモリを**少なく**（作業机の広さは限られている）する。

コンピュータに解かせるために

- 解くための手順（アルゴリズム）を考える
  →手順がわからなきゃ解かせようがない

  → 手順は実行の効率や**計算量**に影響

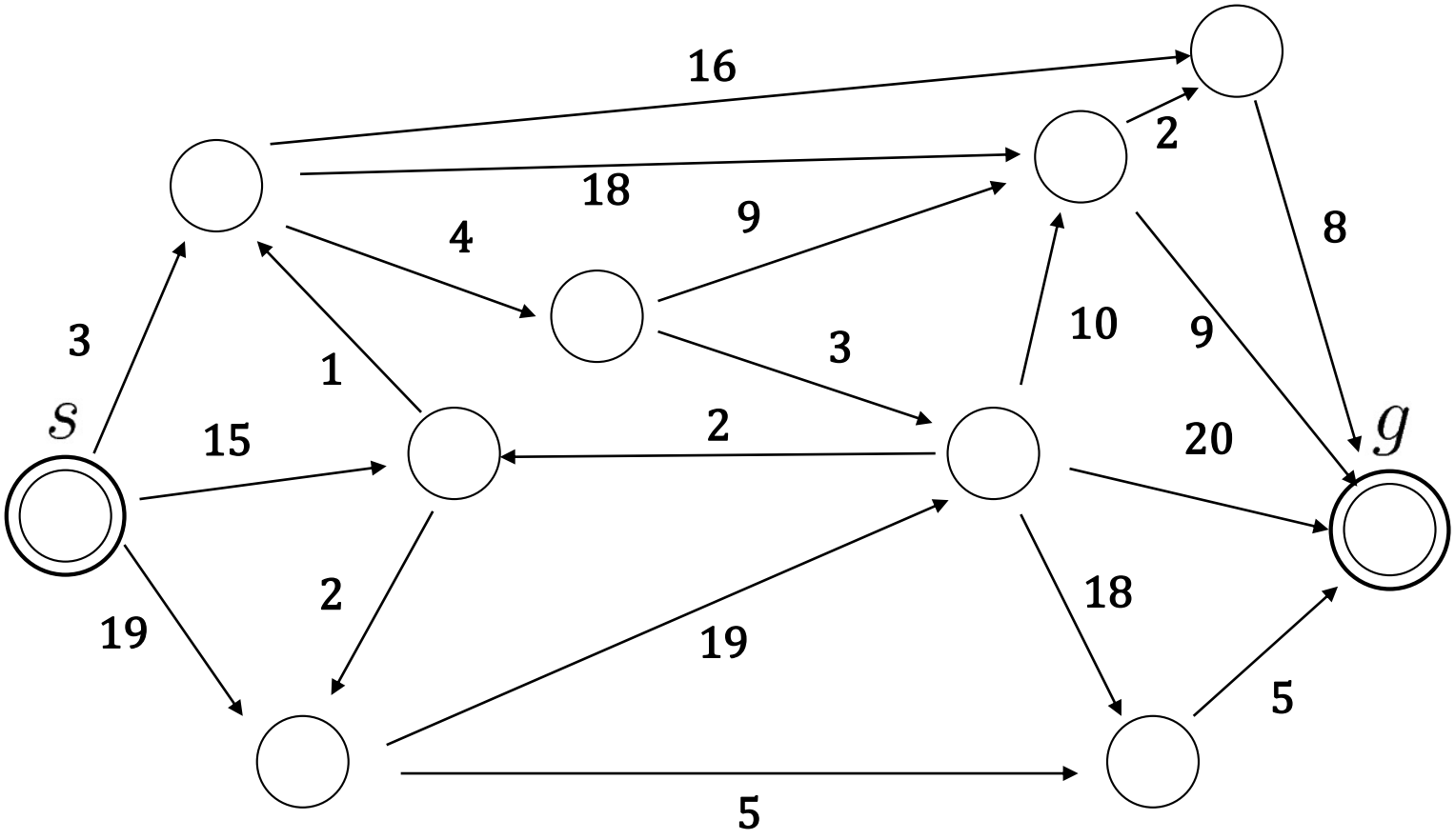
- コンピュータに指令をだす
  →コンピュータがわかる言葉（プログラム言語）で指令しなければならない　＝　この指令が「プログラム」

  → 指令の出し方は実行の効率や**速度**に影響

できるだけ**速く**て（時間は限られている）、
使用メモリを**少なく**（作業机の広さは限られている）する。

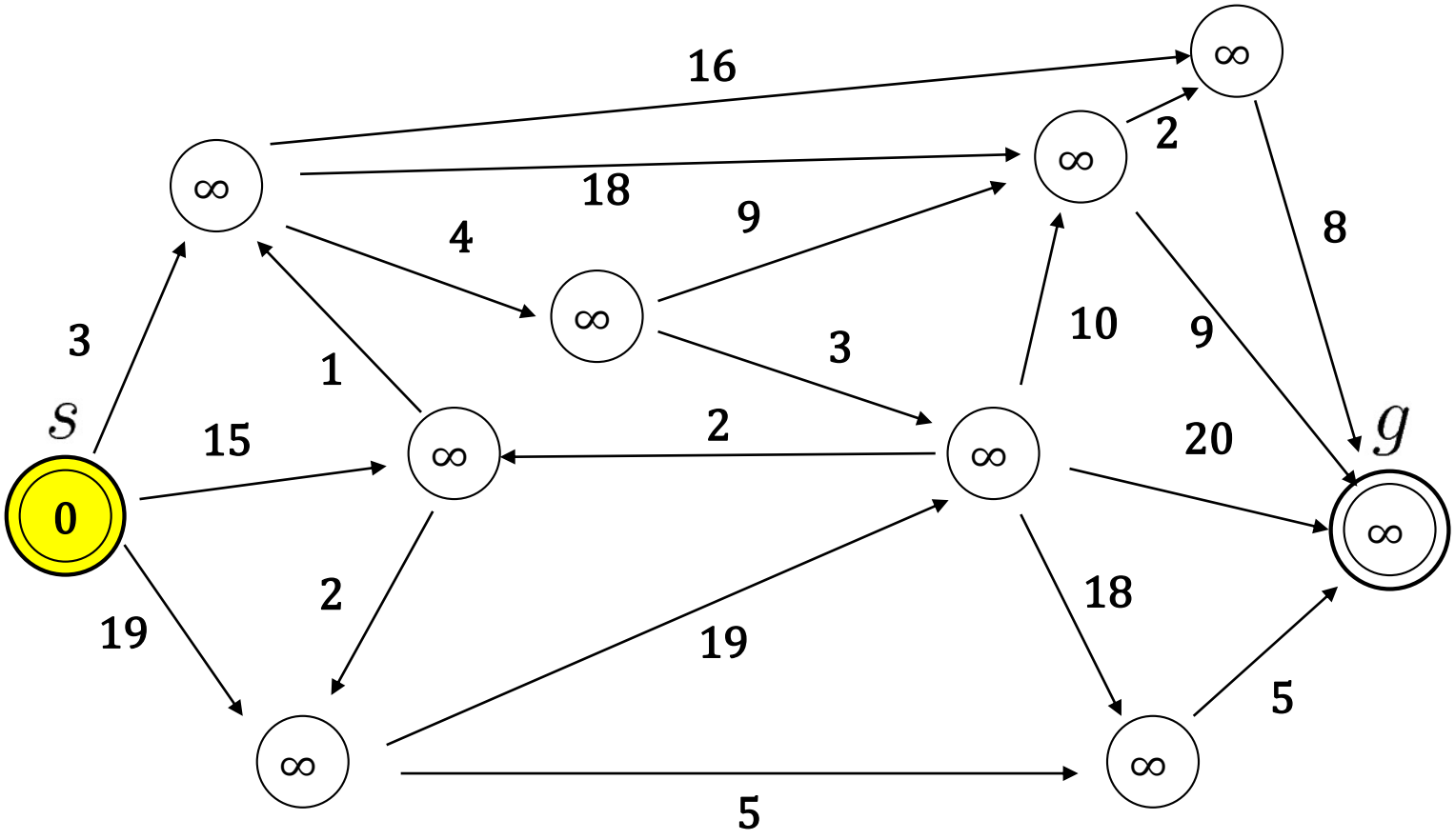
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



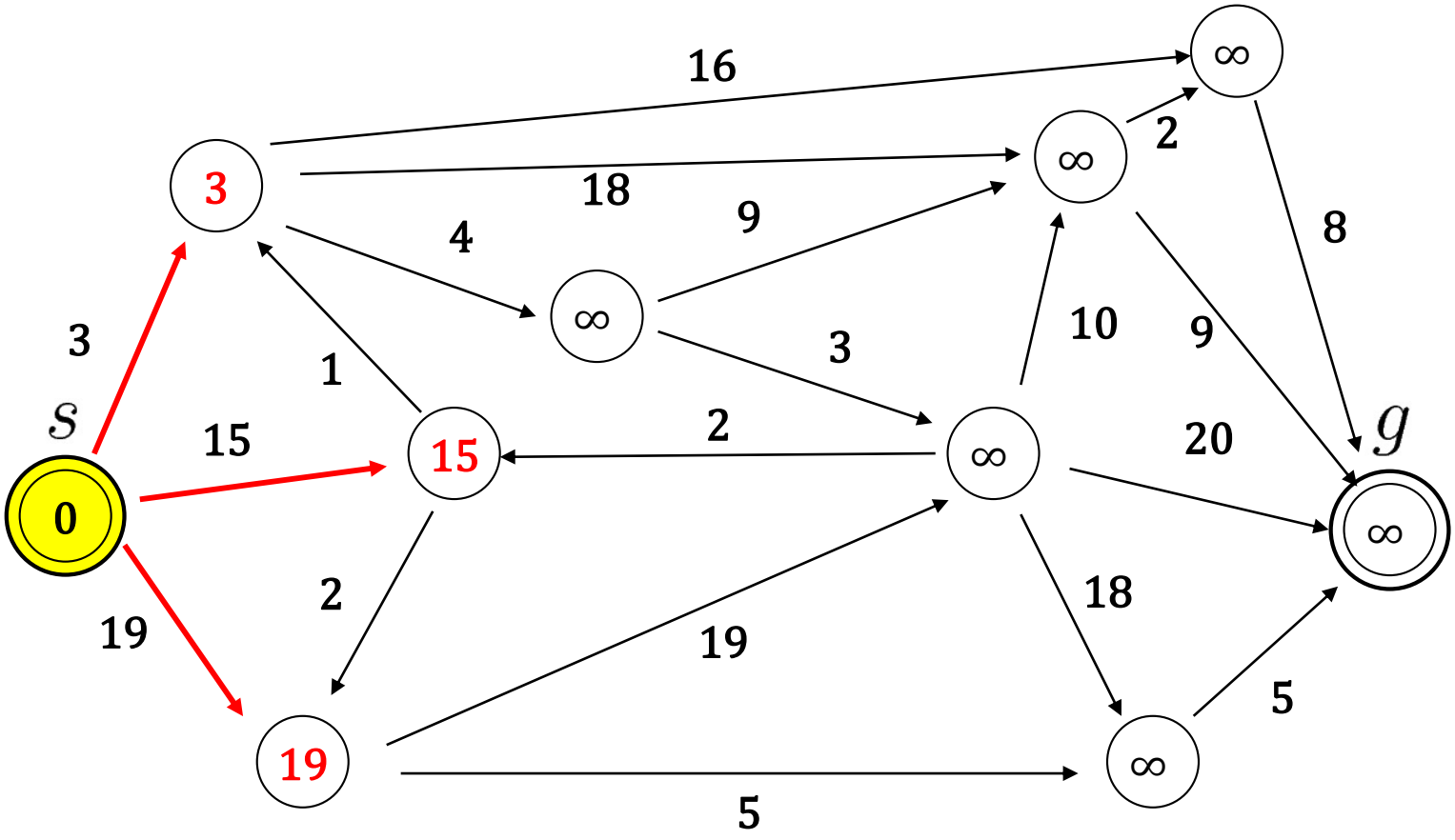
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



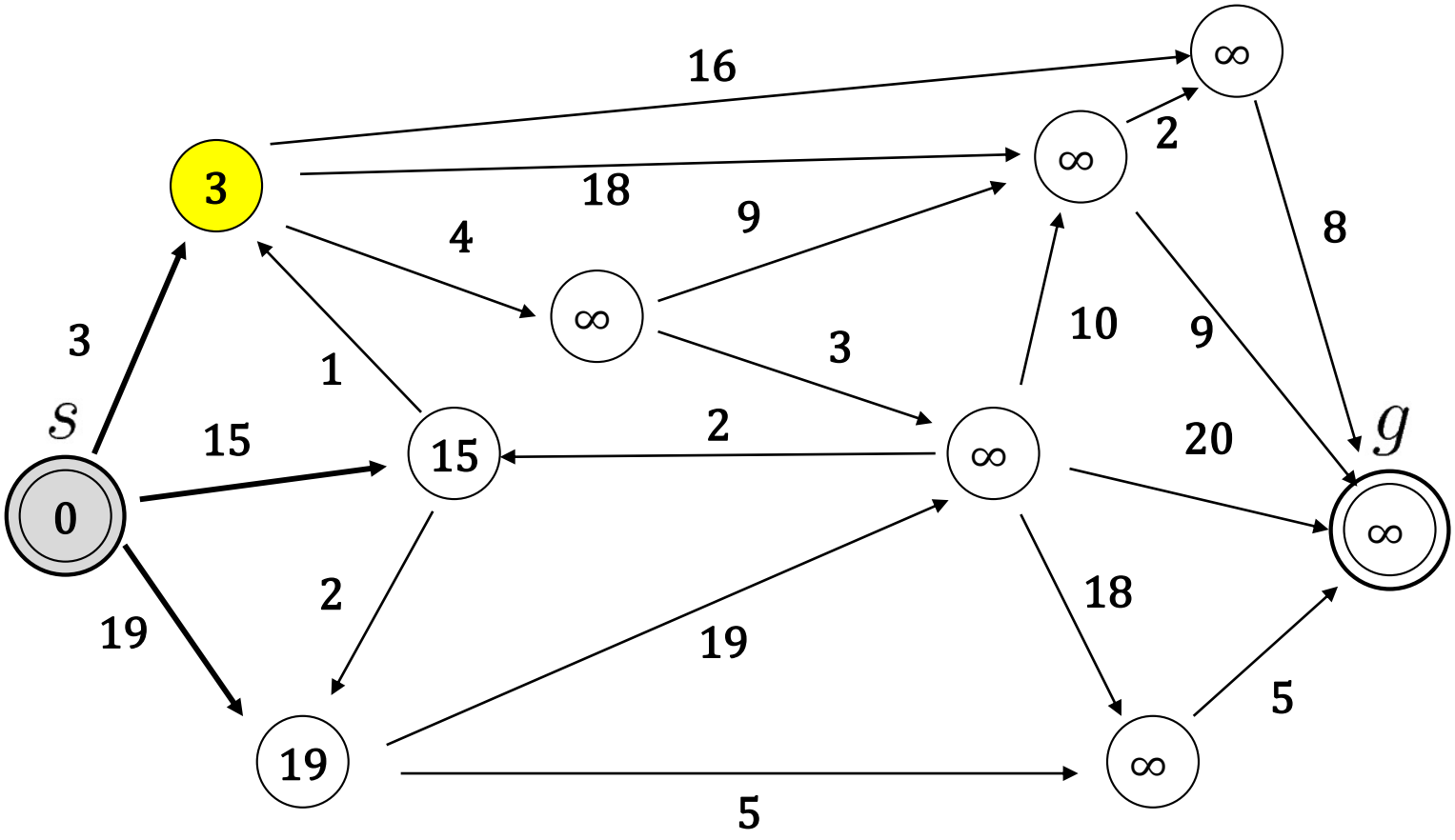
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



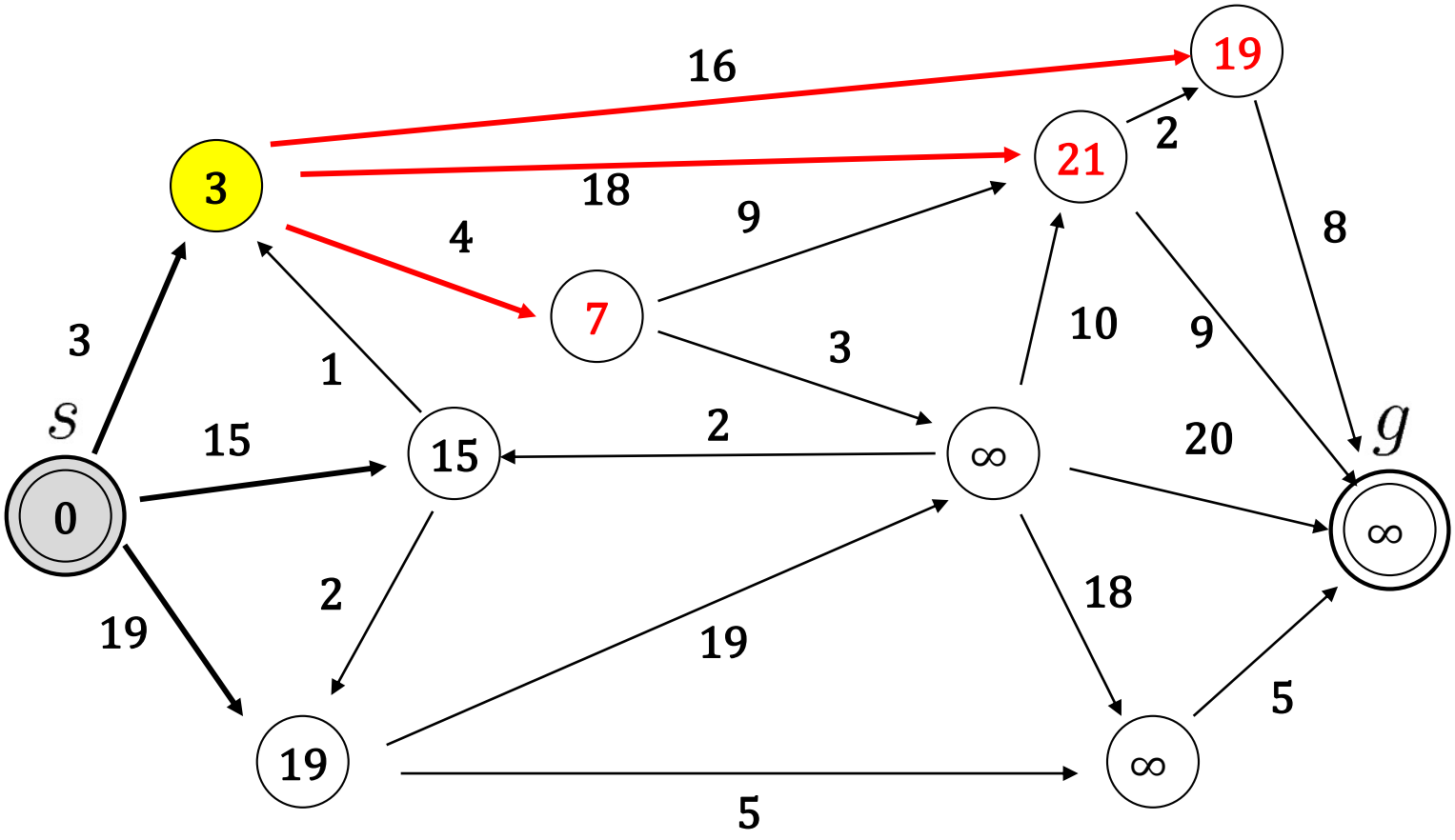
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



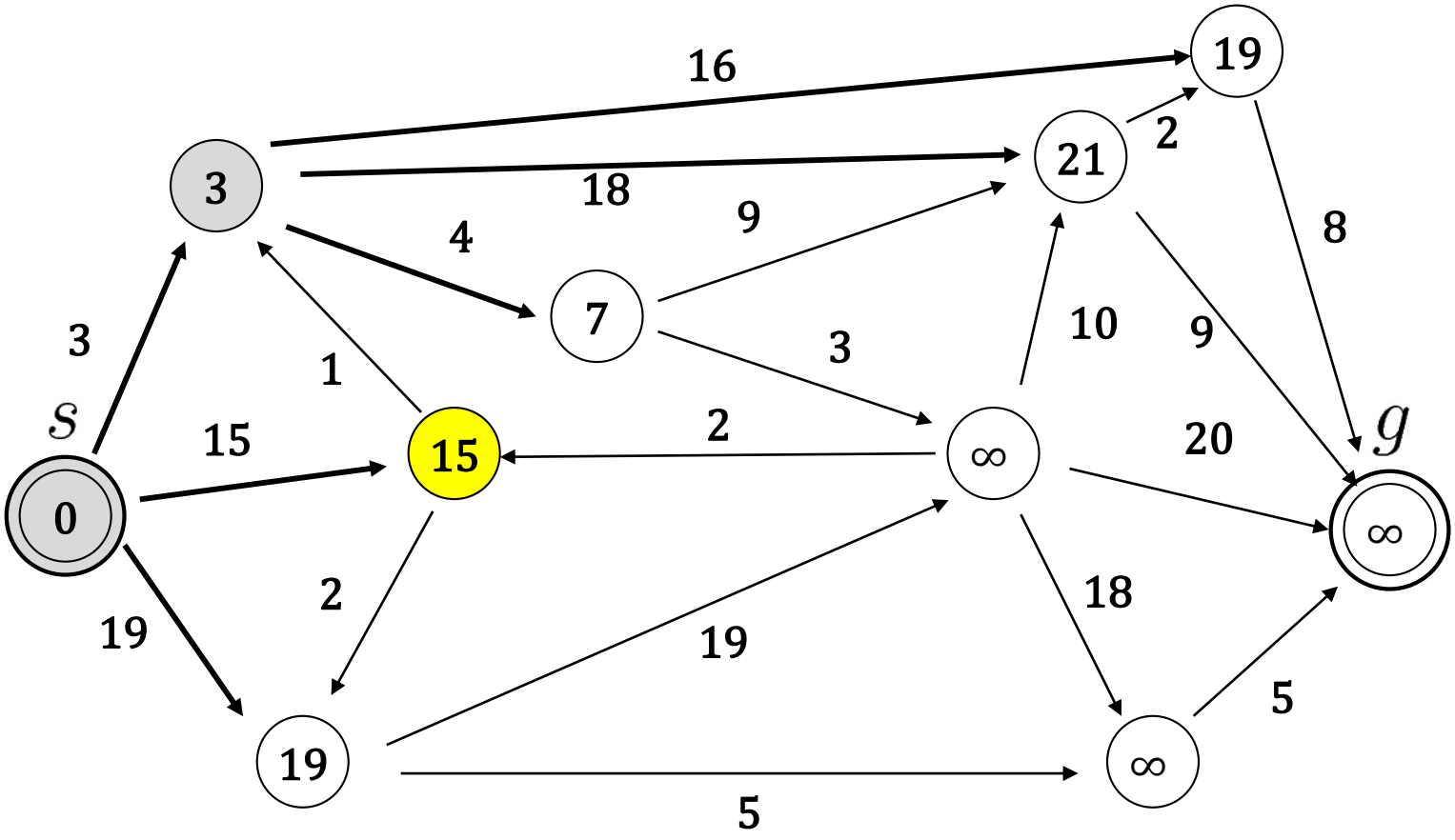
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



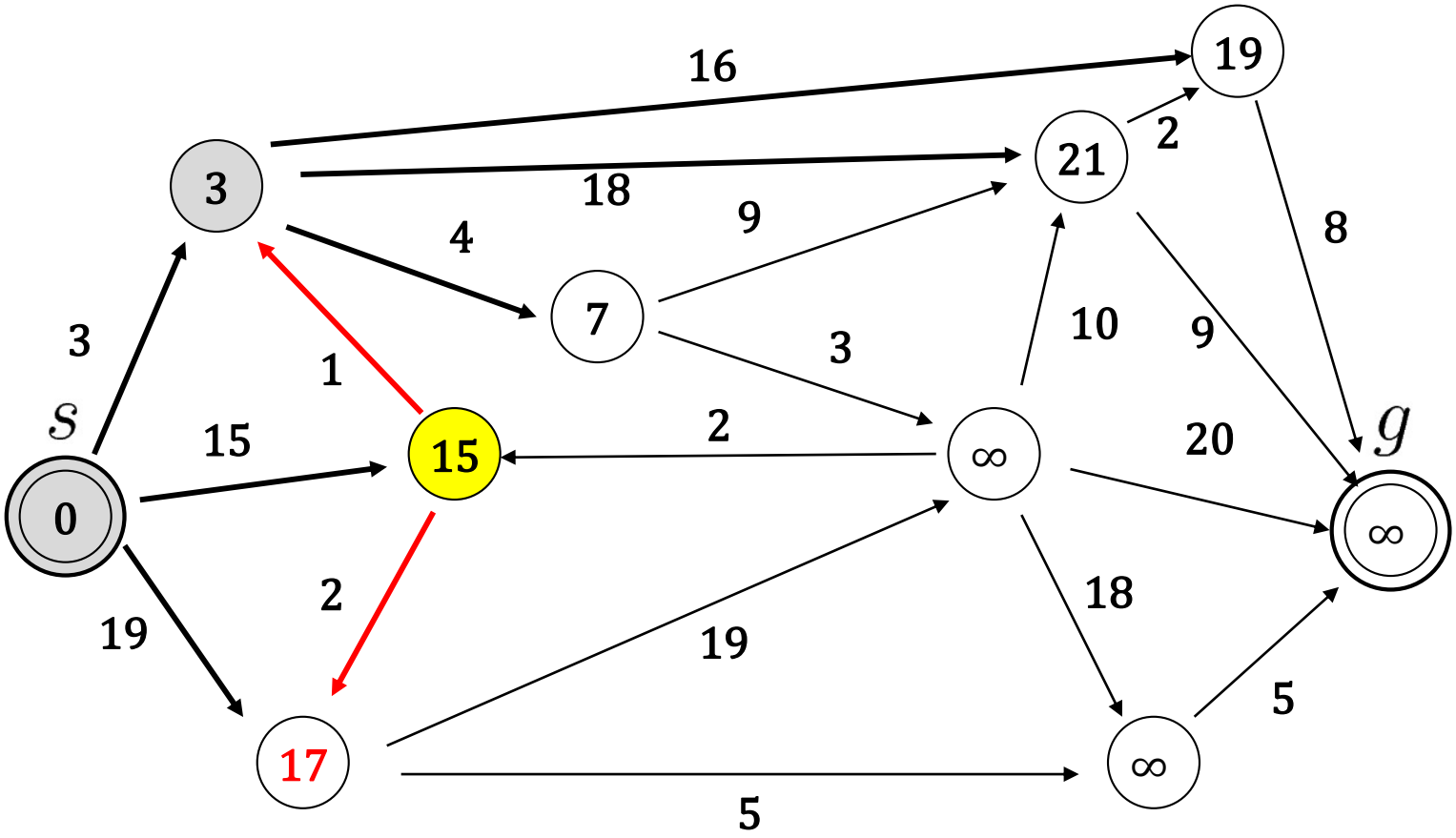
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



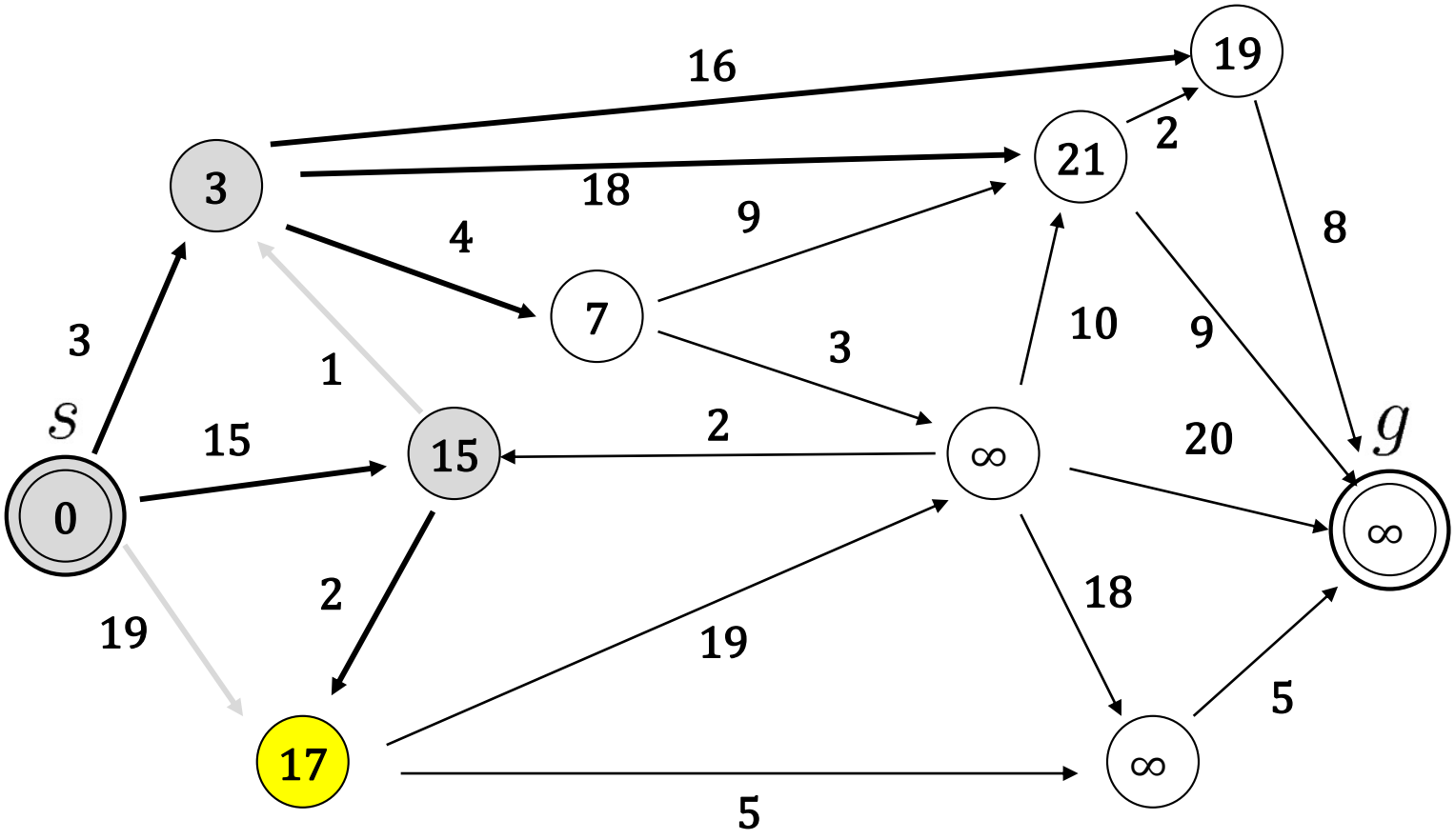
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



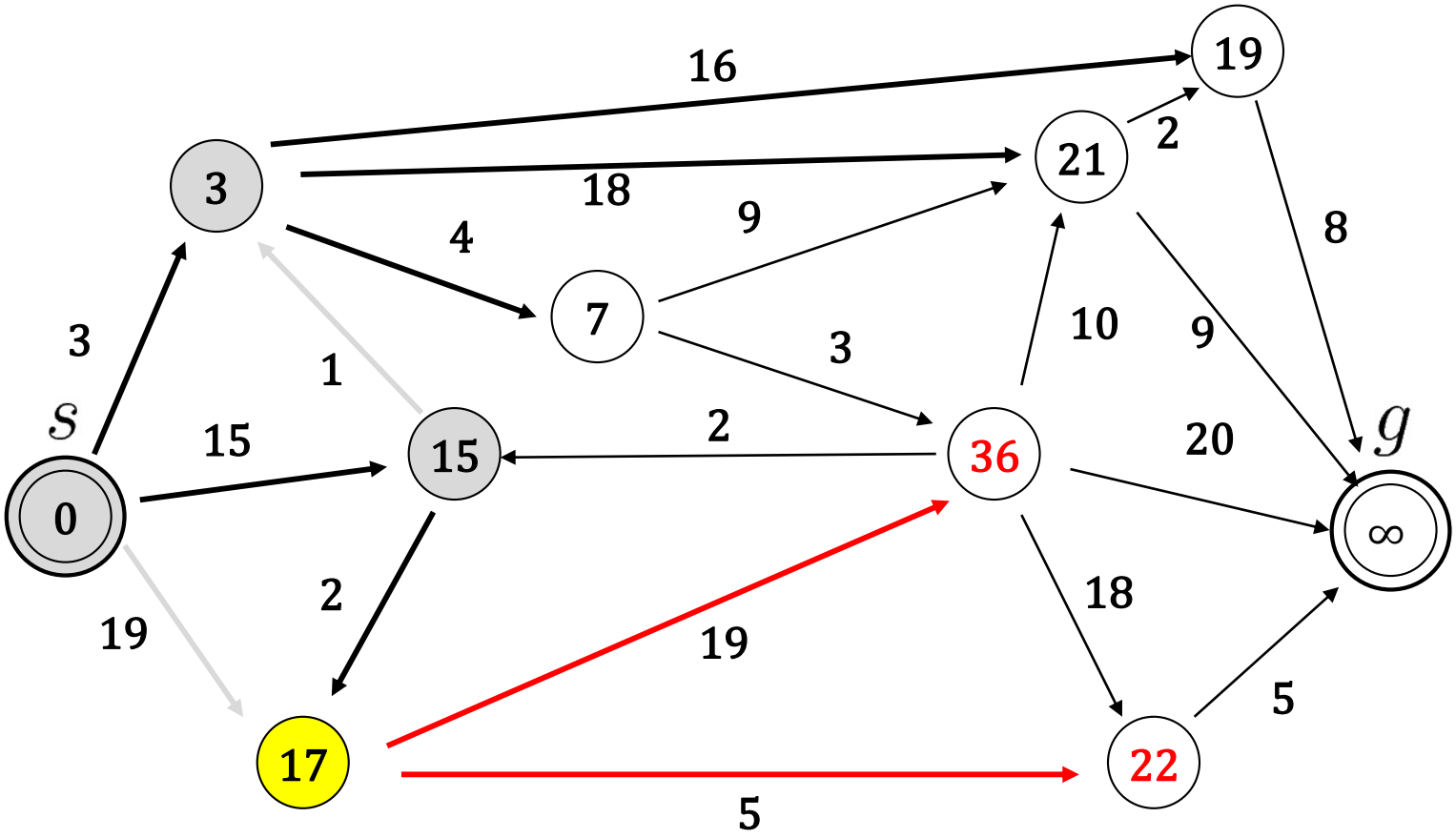
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



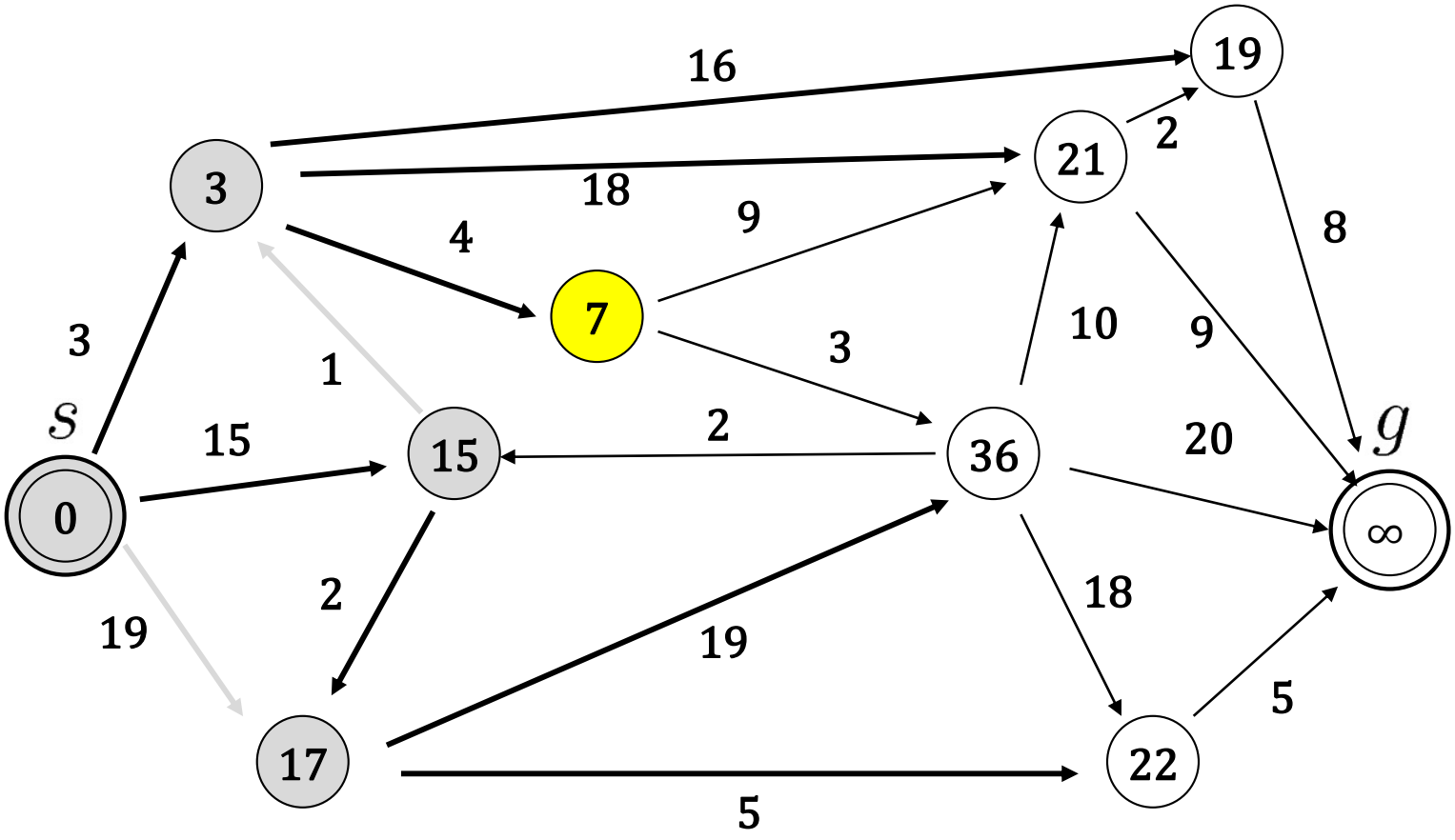
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



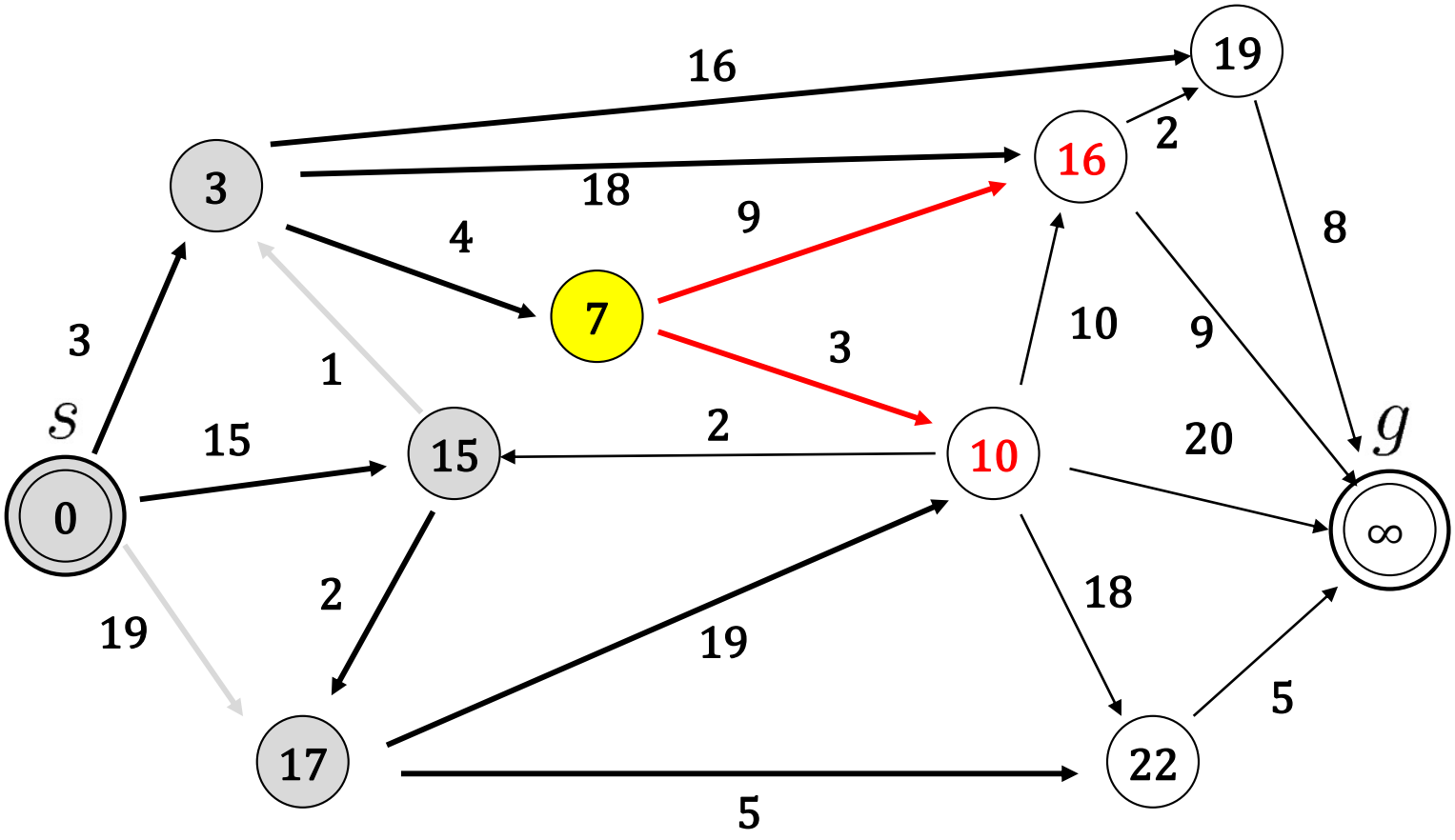
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



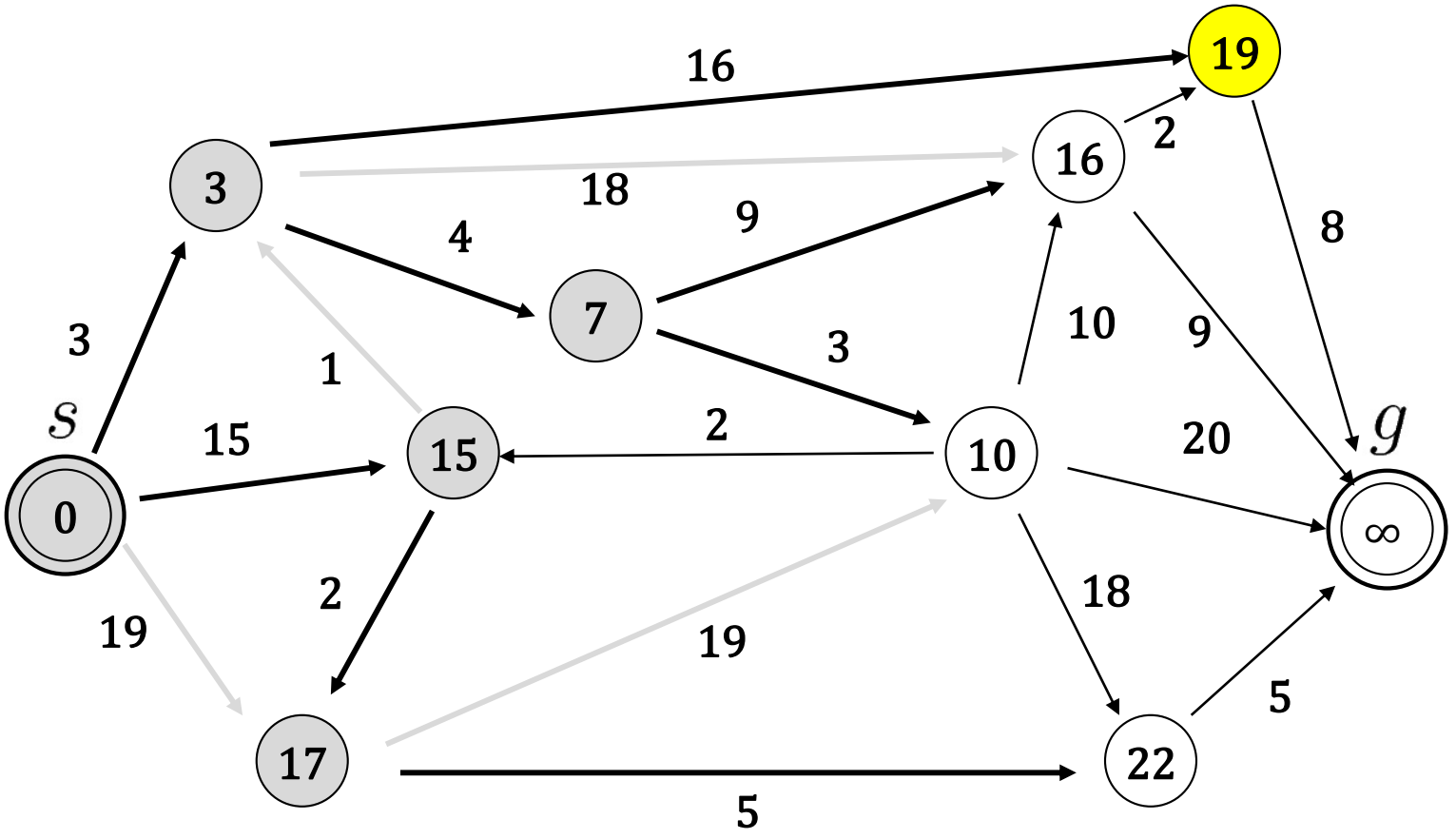
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



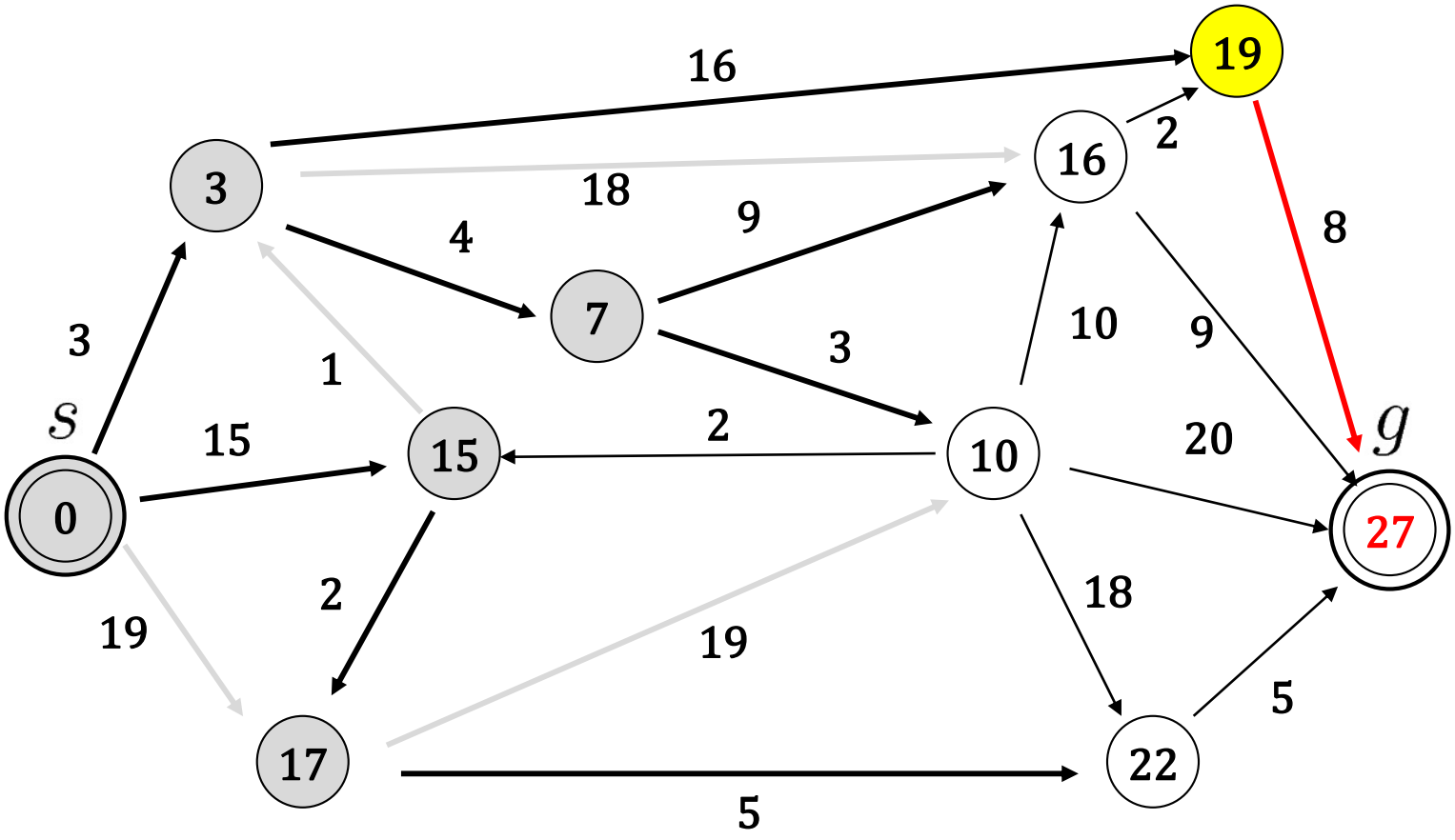
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



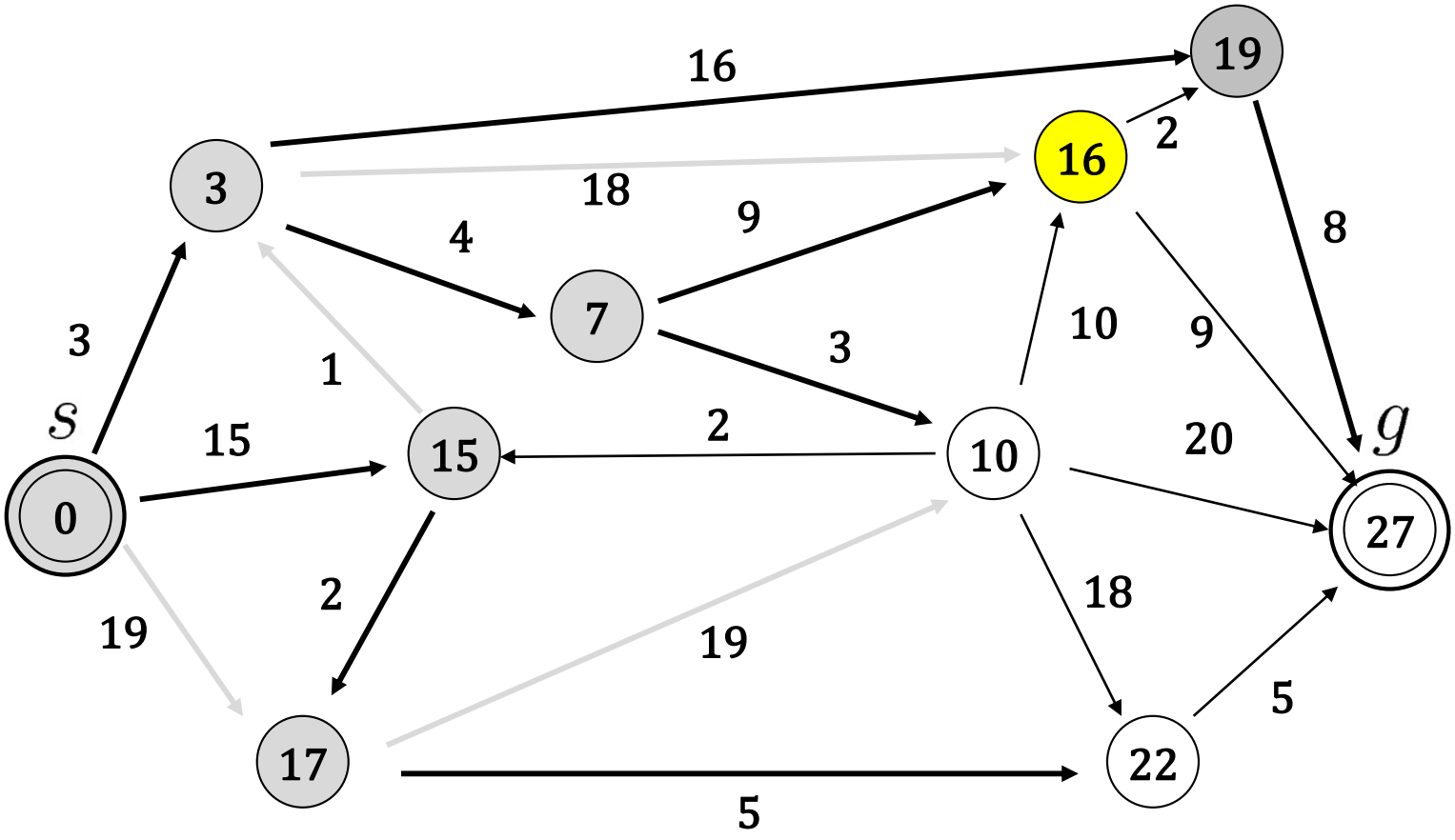
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



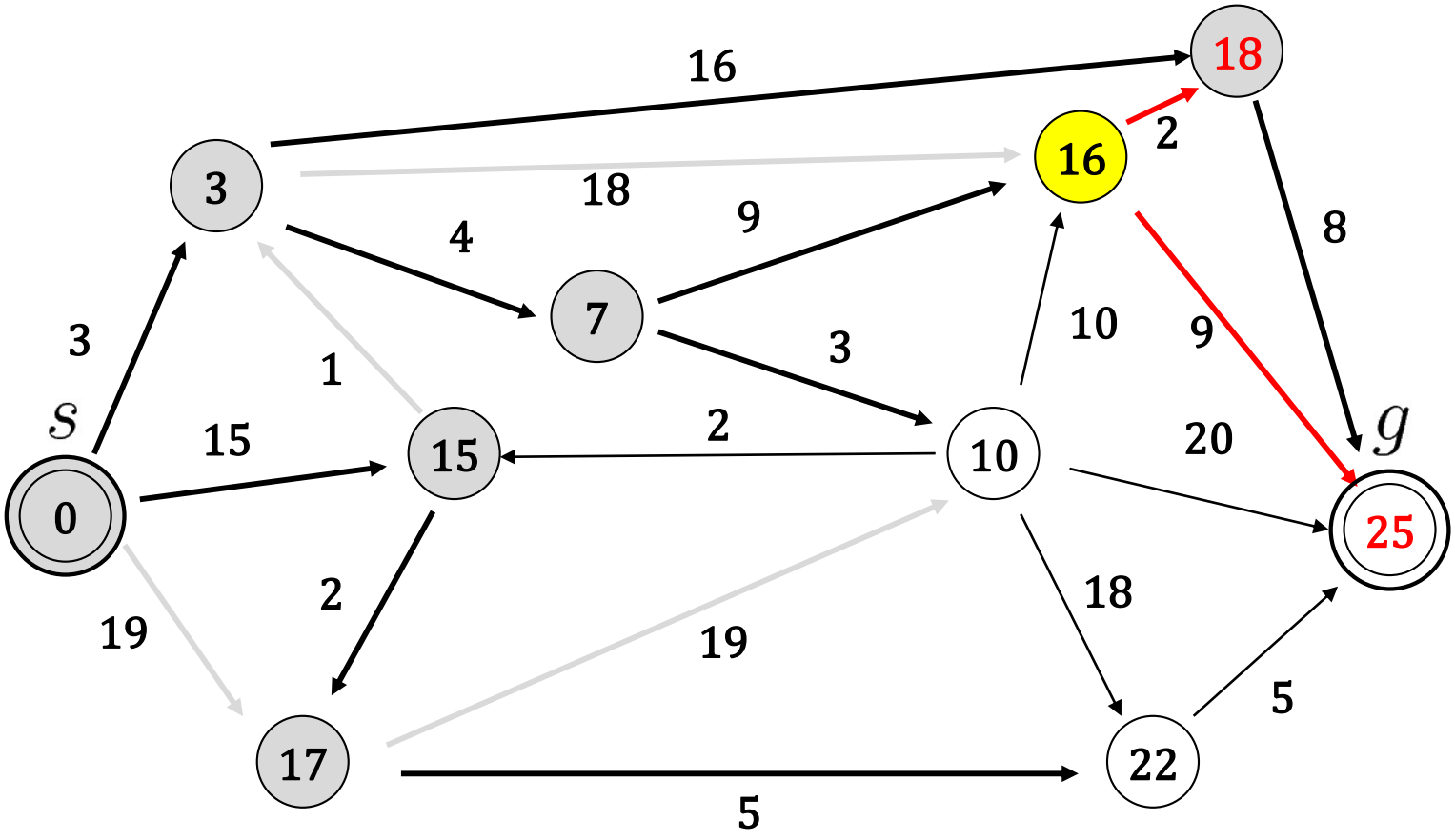
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



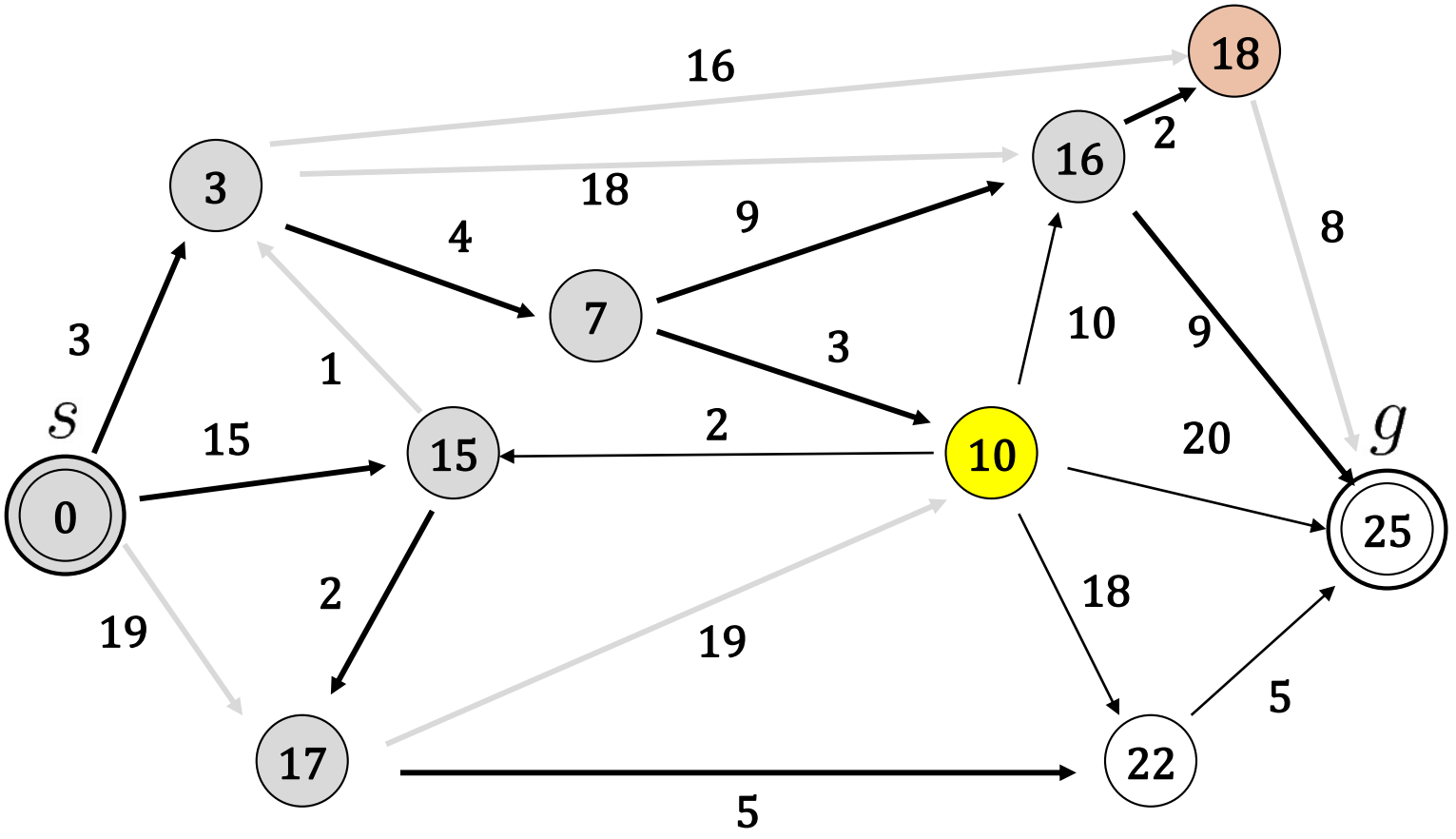
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



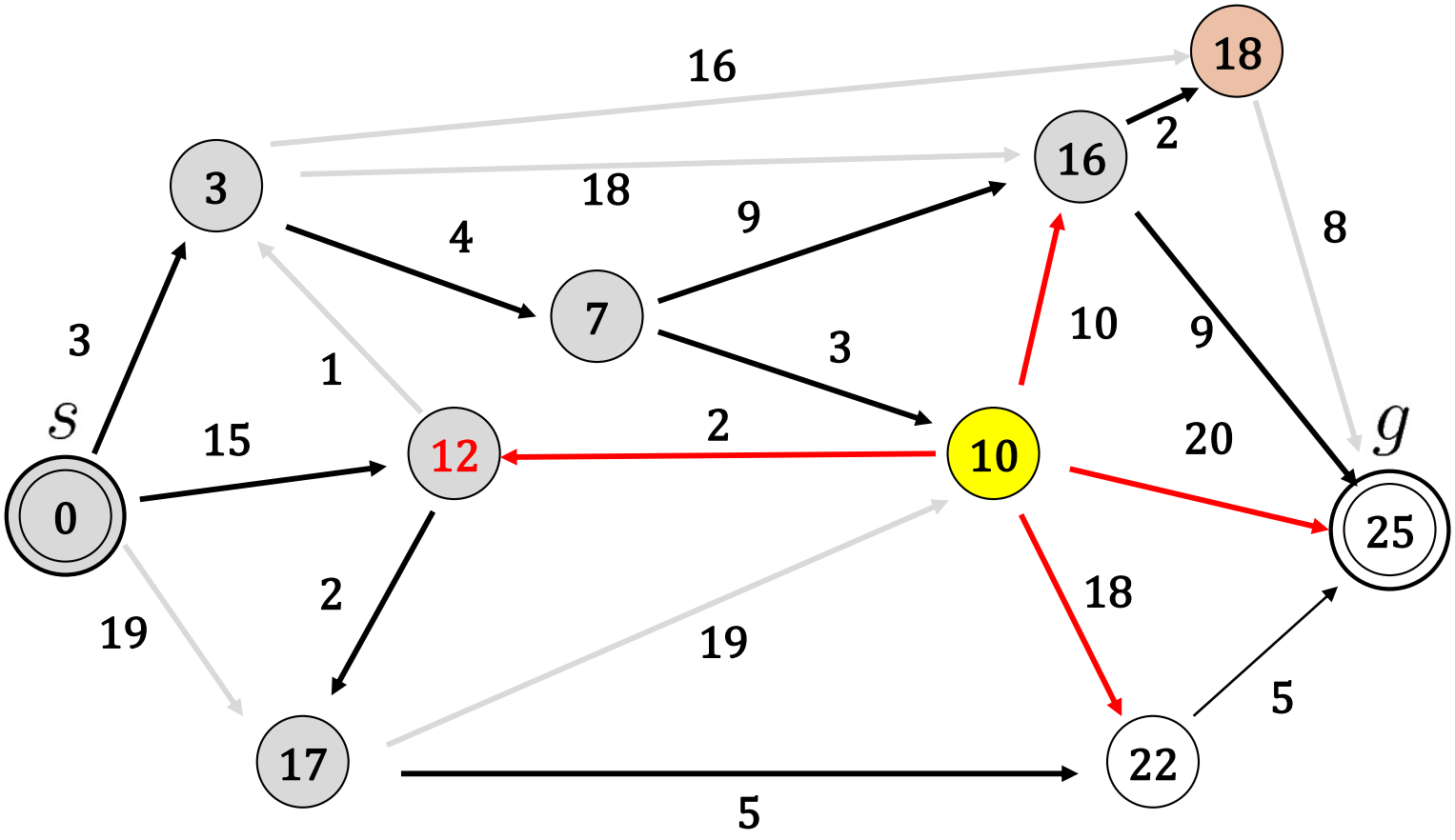
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



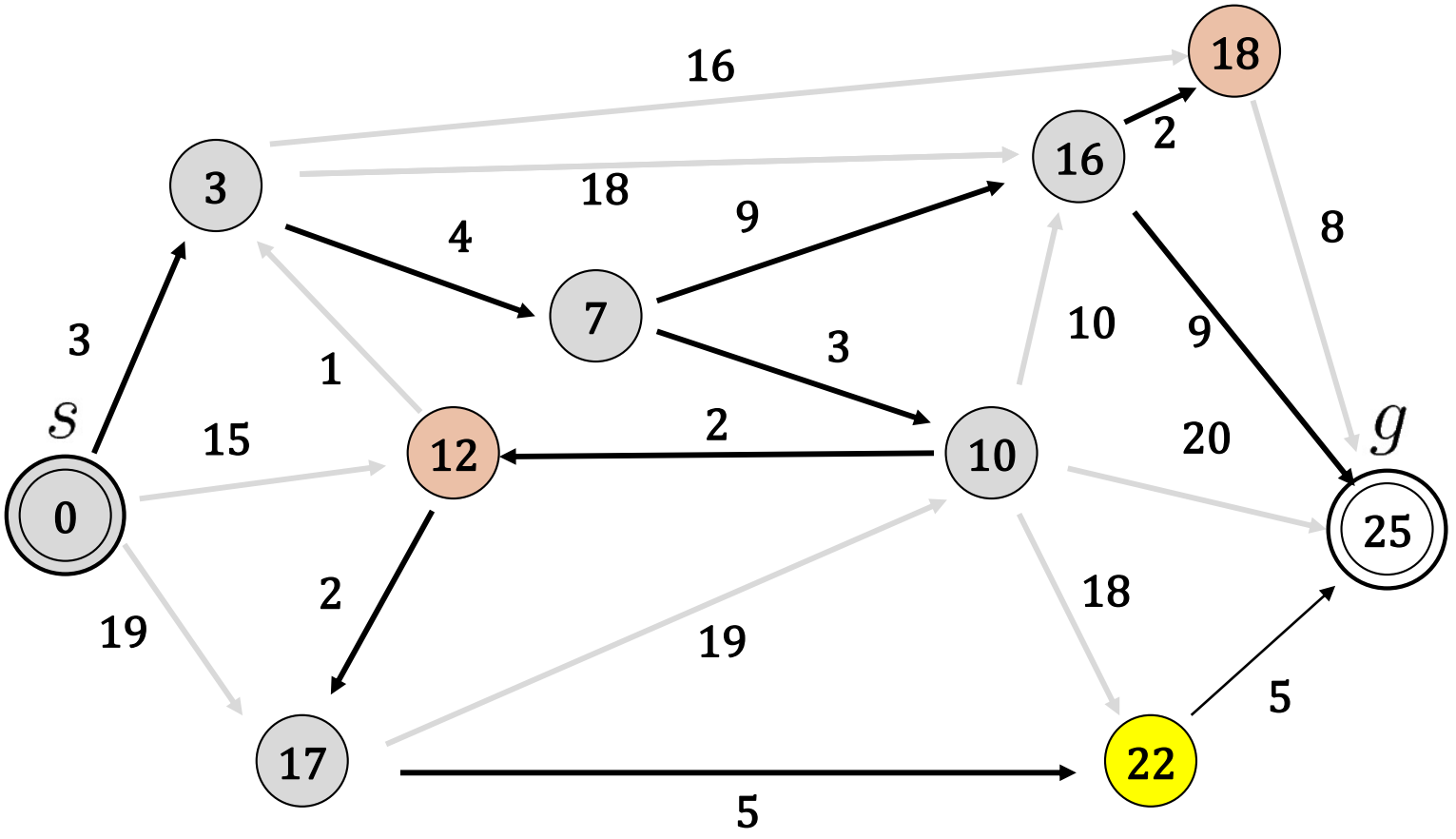
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



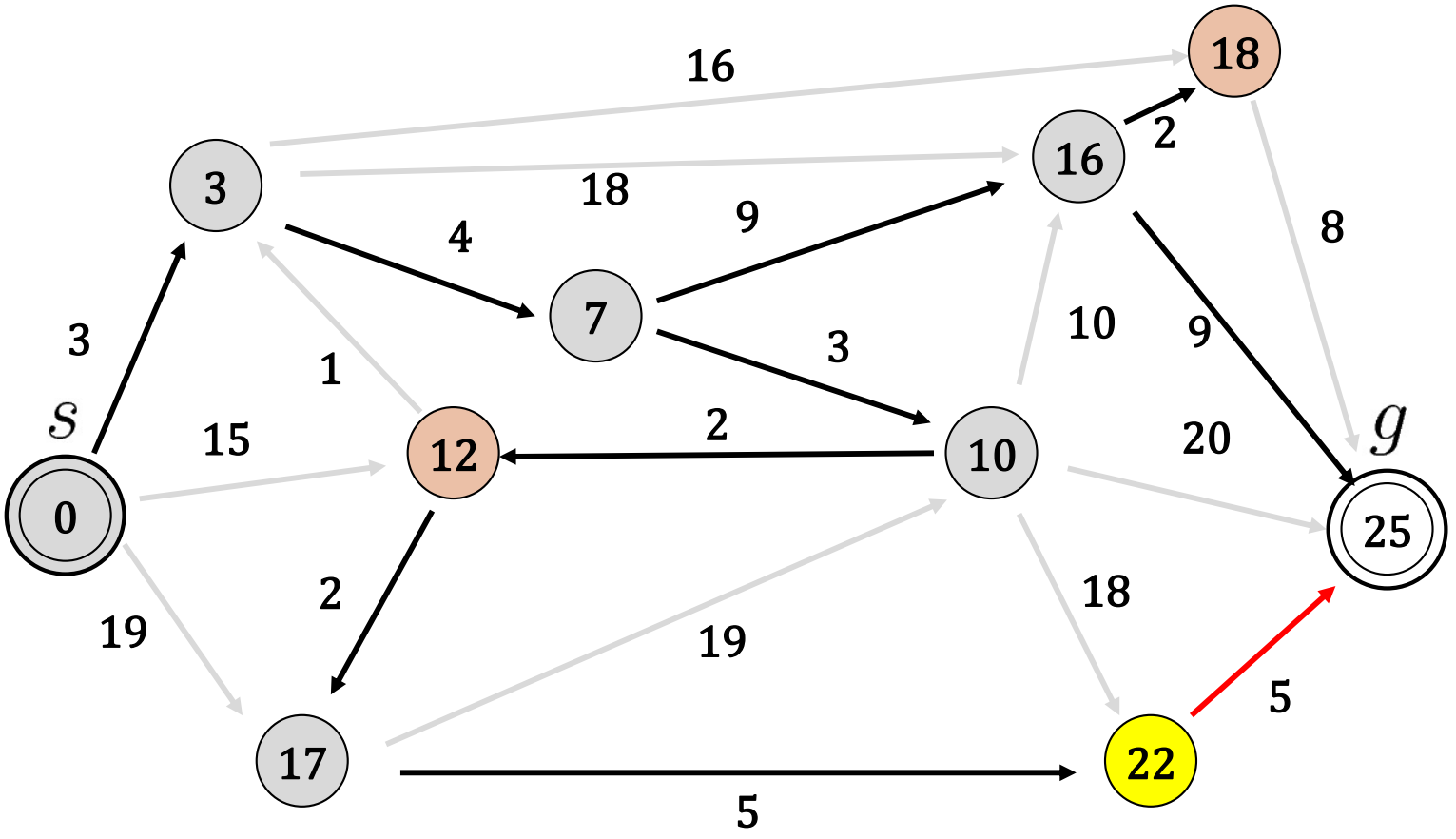
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



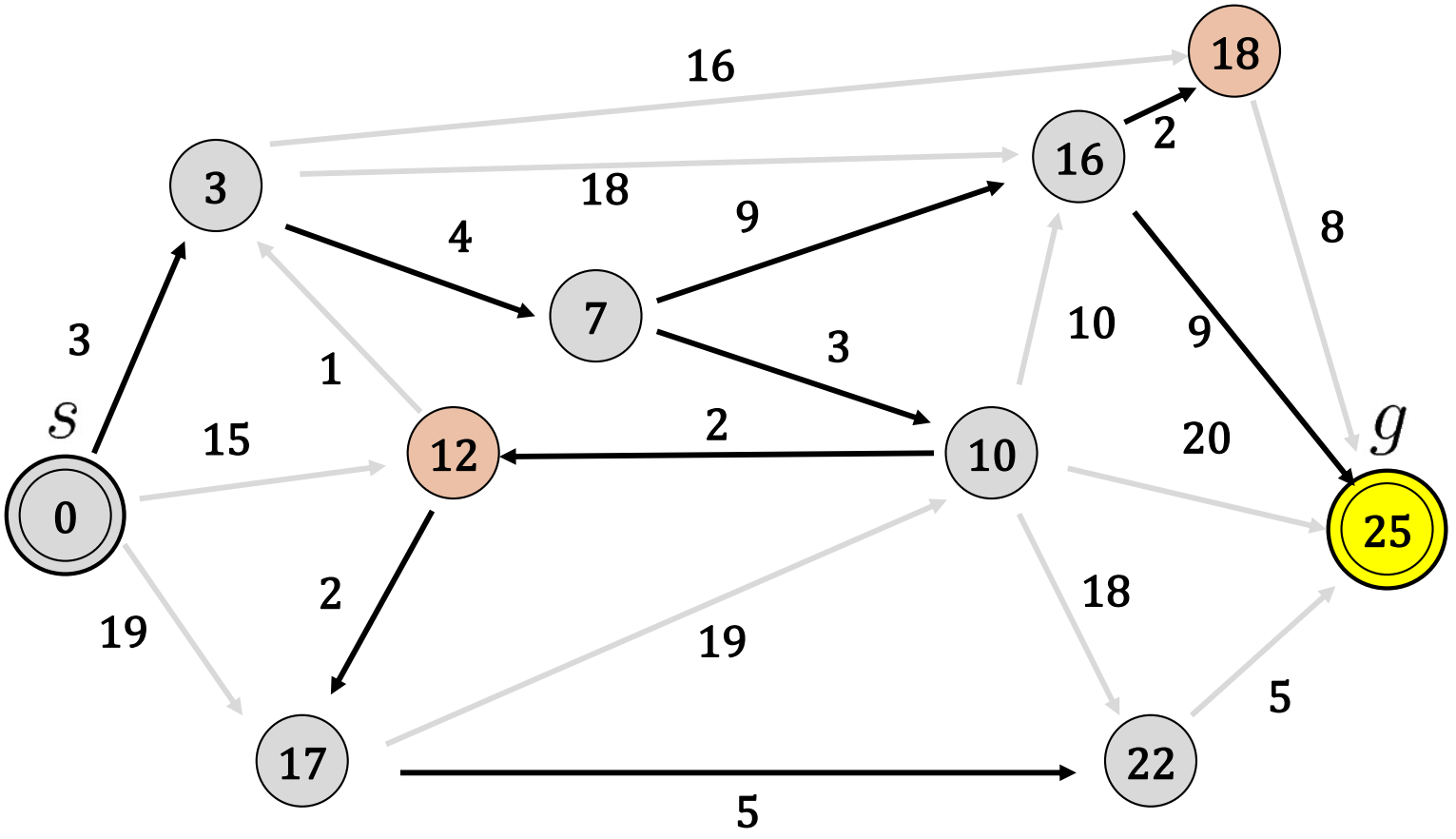
手順を考えよう

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）

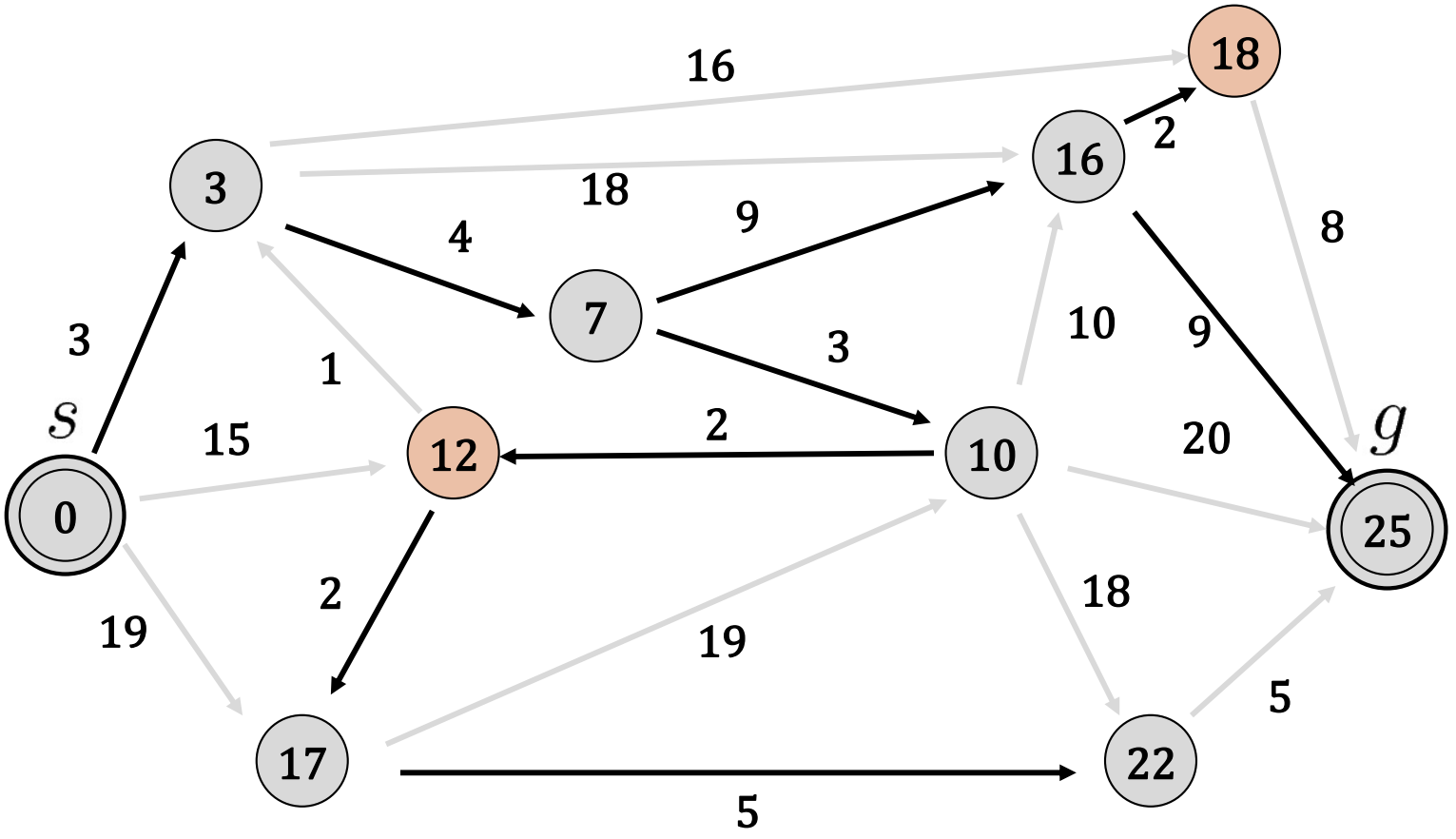


手順を考えよう

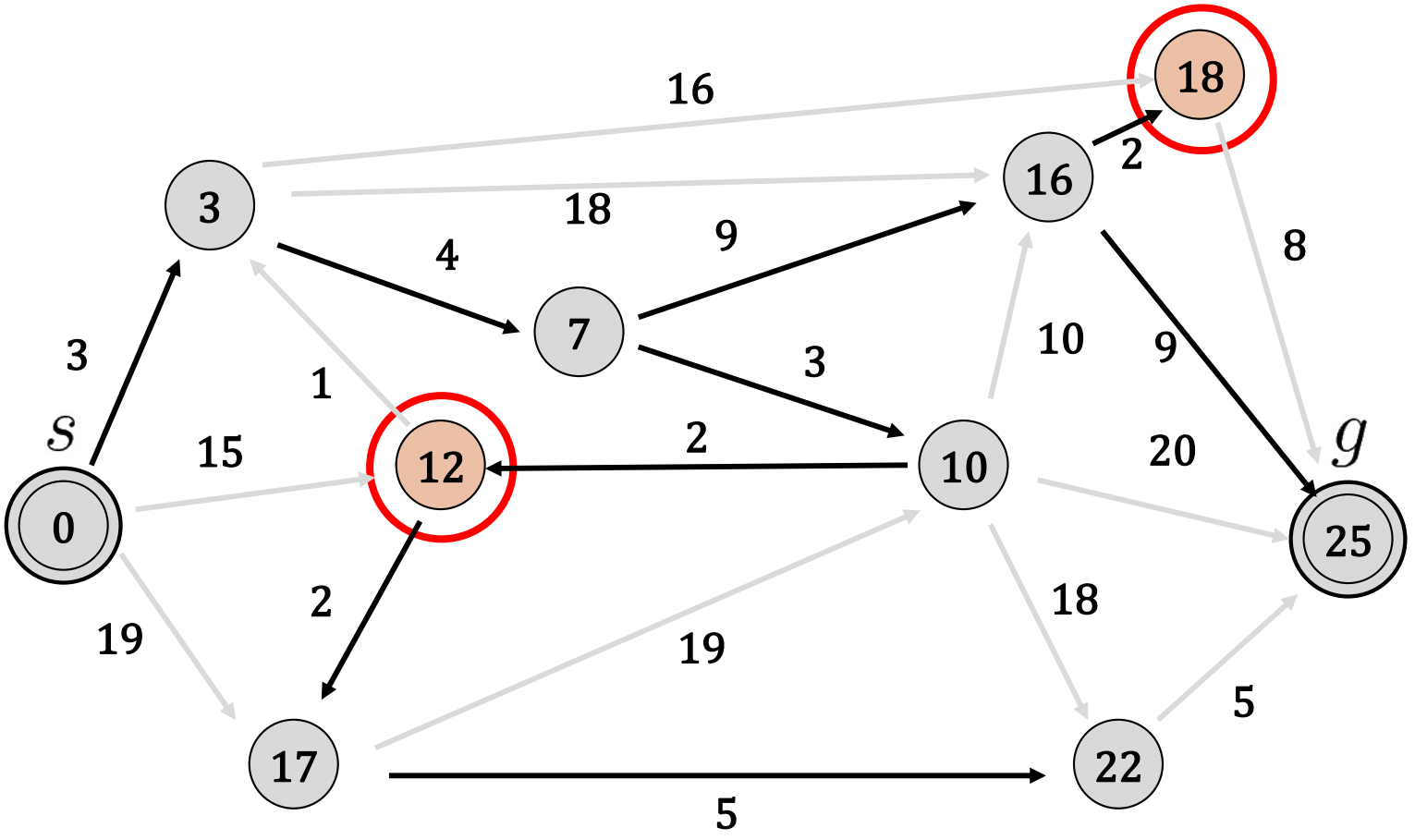
出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



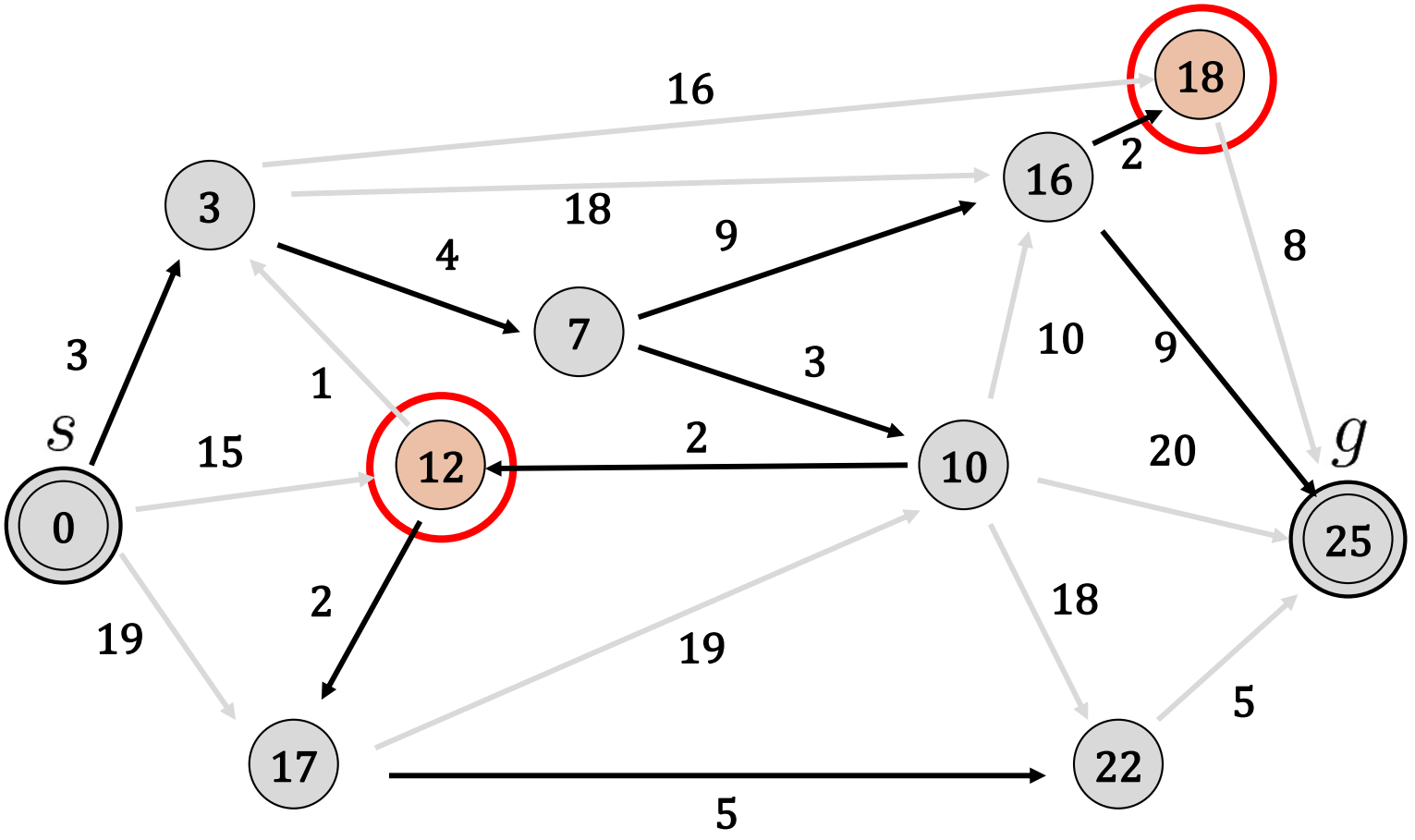
一周目終わり



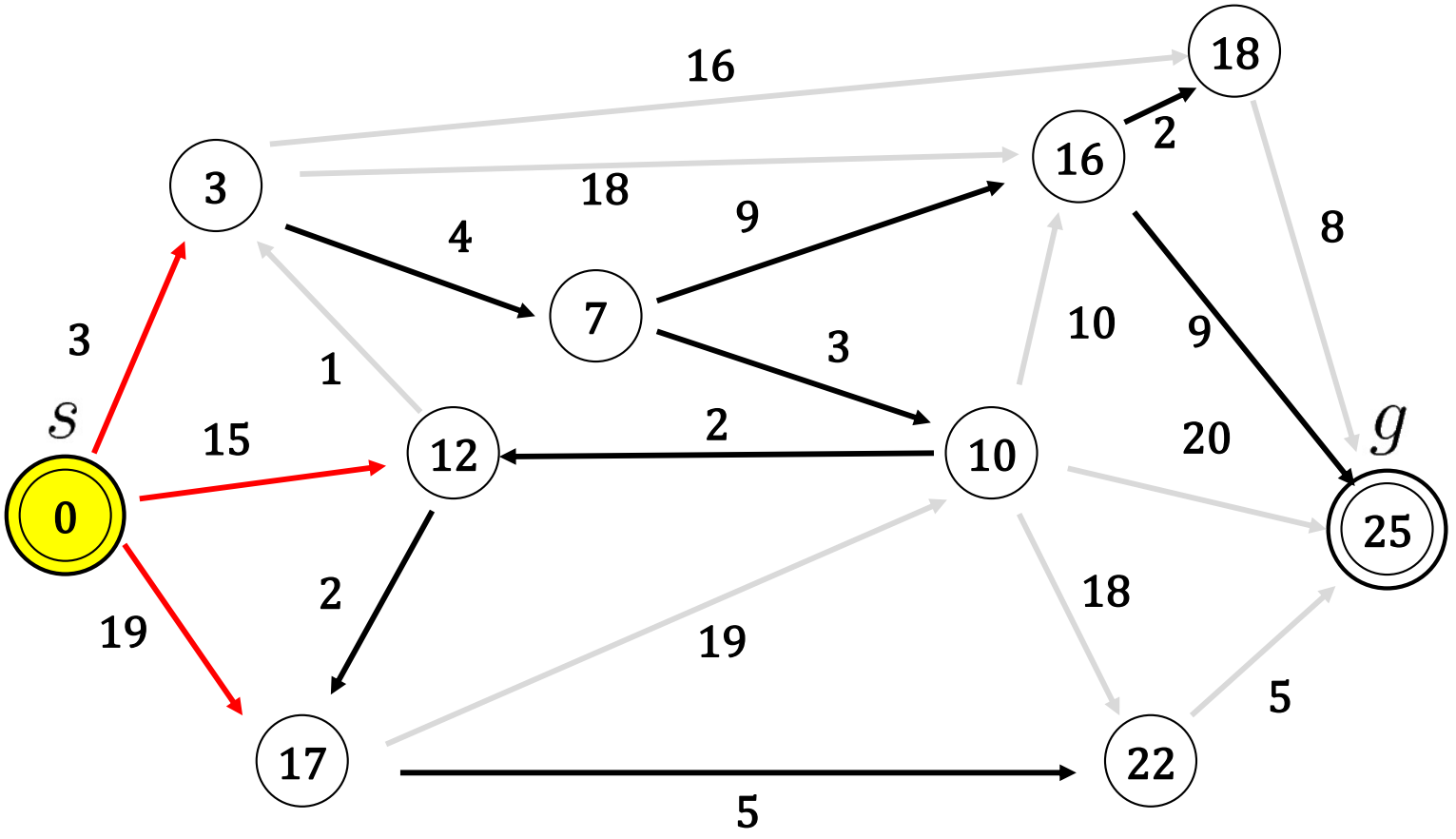
一周目終わり



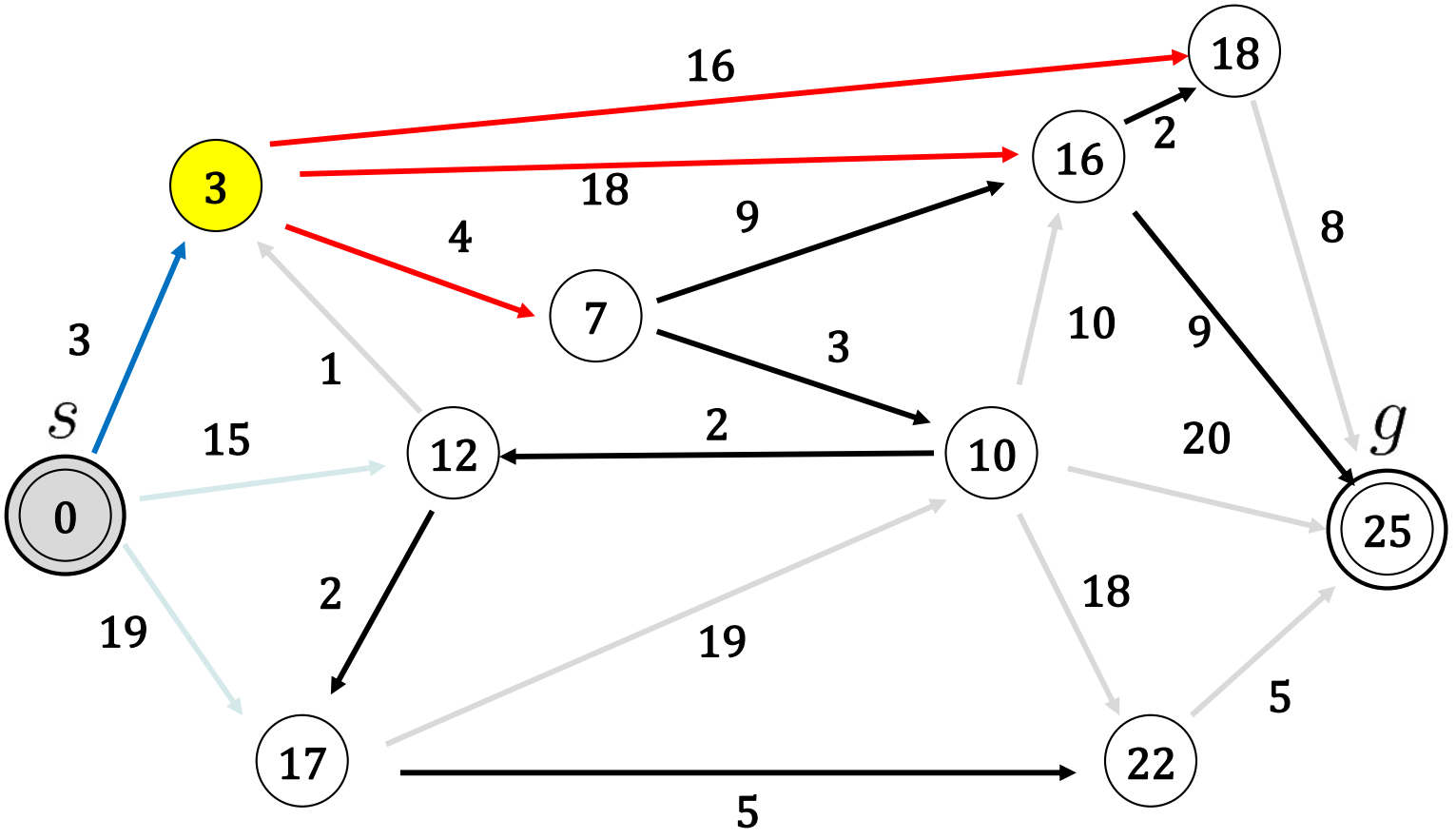
一周目終わり



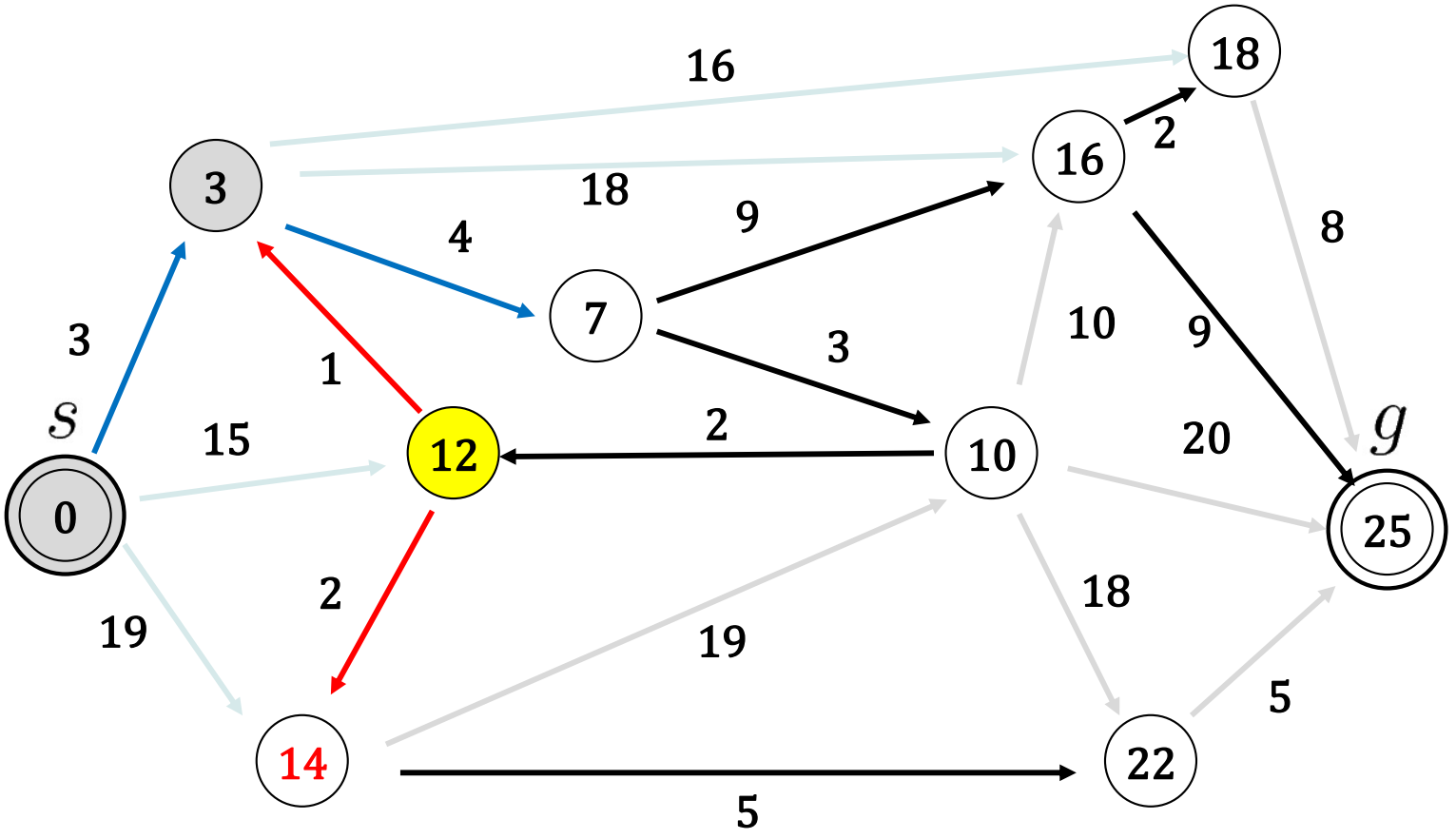
二周目（同じ順番で点をみる）



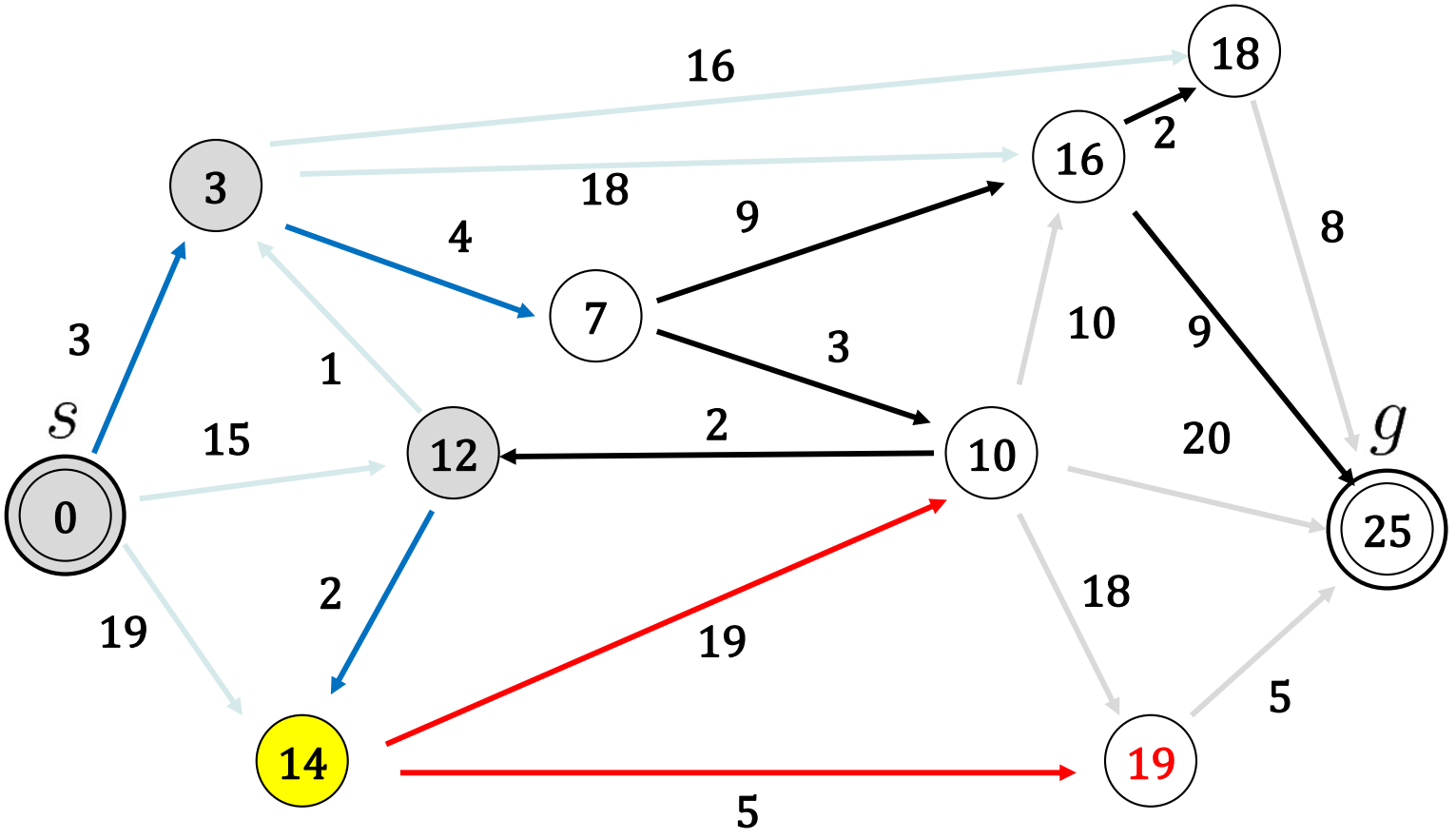
二周目（同じ順番で点をみる）



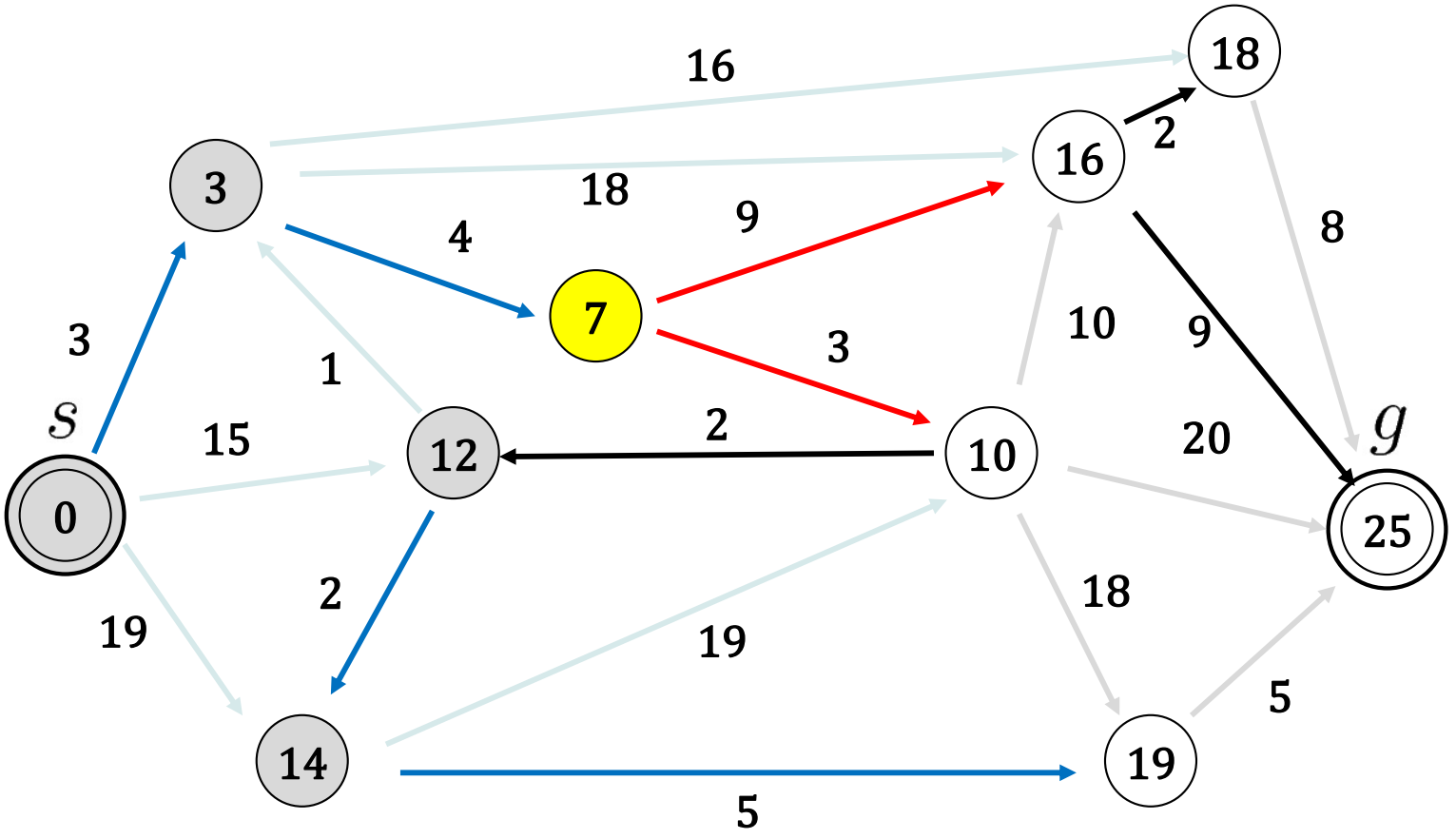
二周目（同じ順番で点をみる）



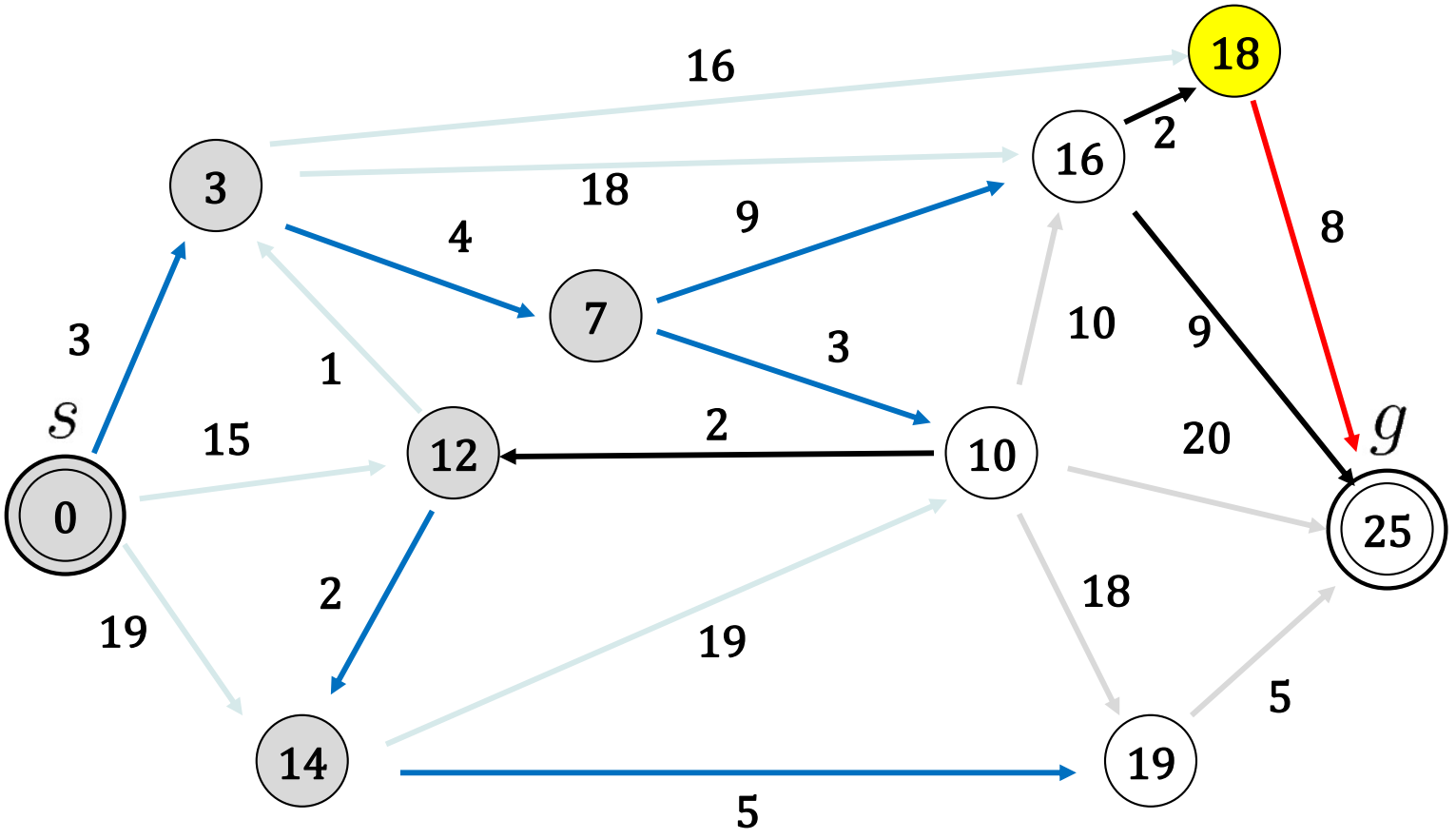
二周目（同じ順番で点をみる）



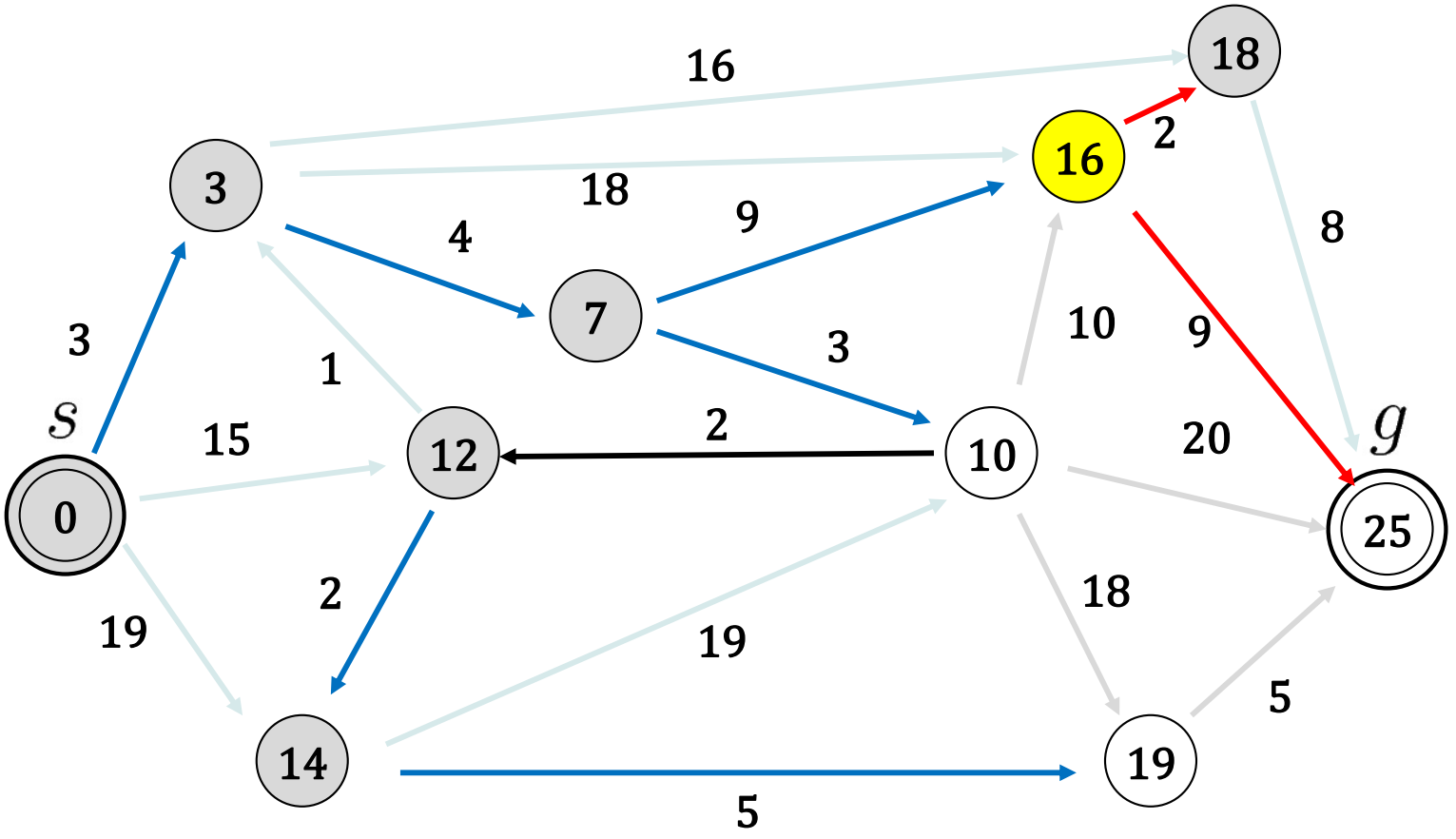
二周目（同じ順番で点をみる）



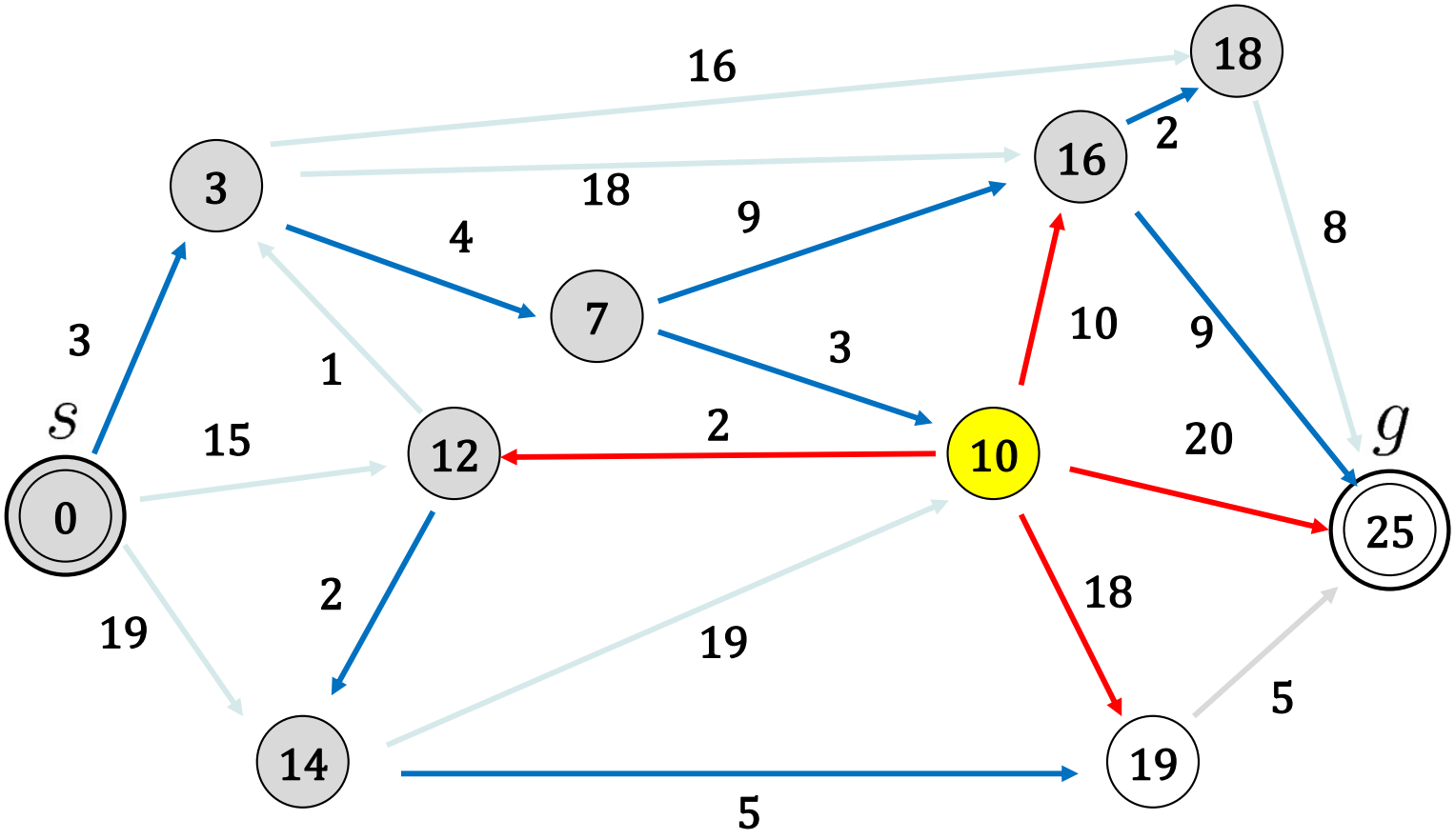
二周目（同じ順番で点をみる）



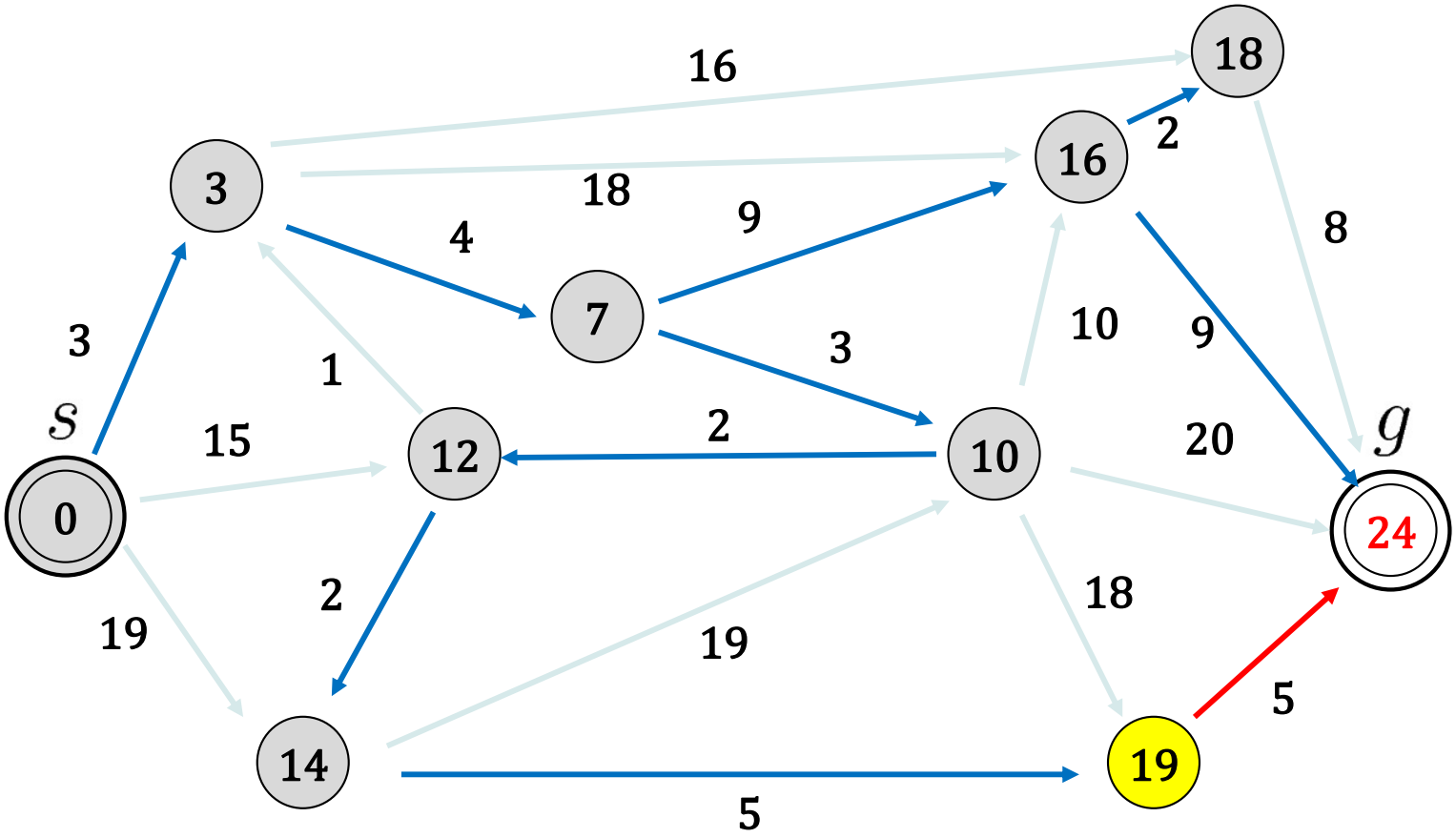
二周目（同じ順番で点をみる）



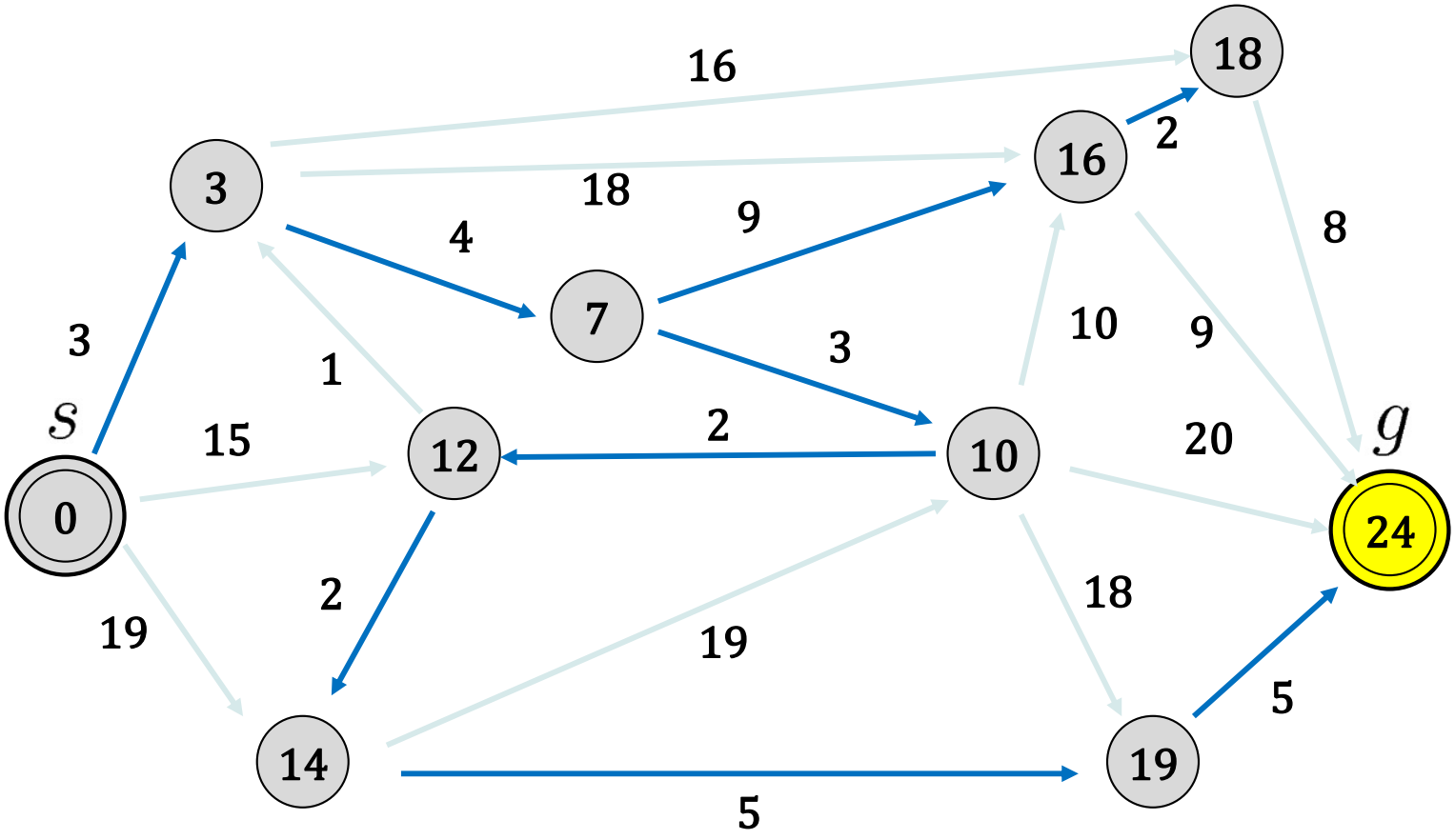
二周目（同じ順番で点をみる）



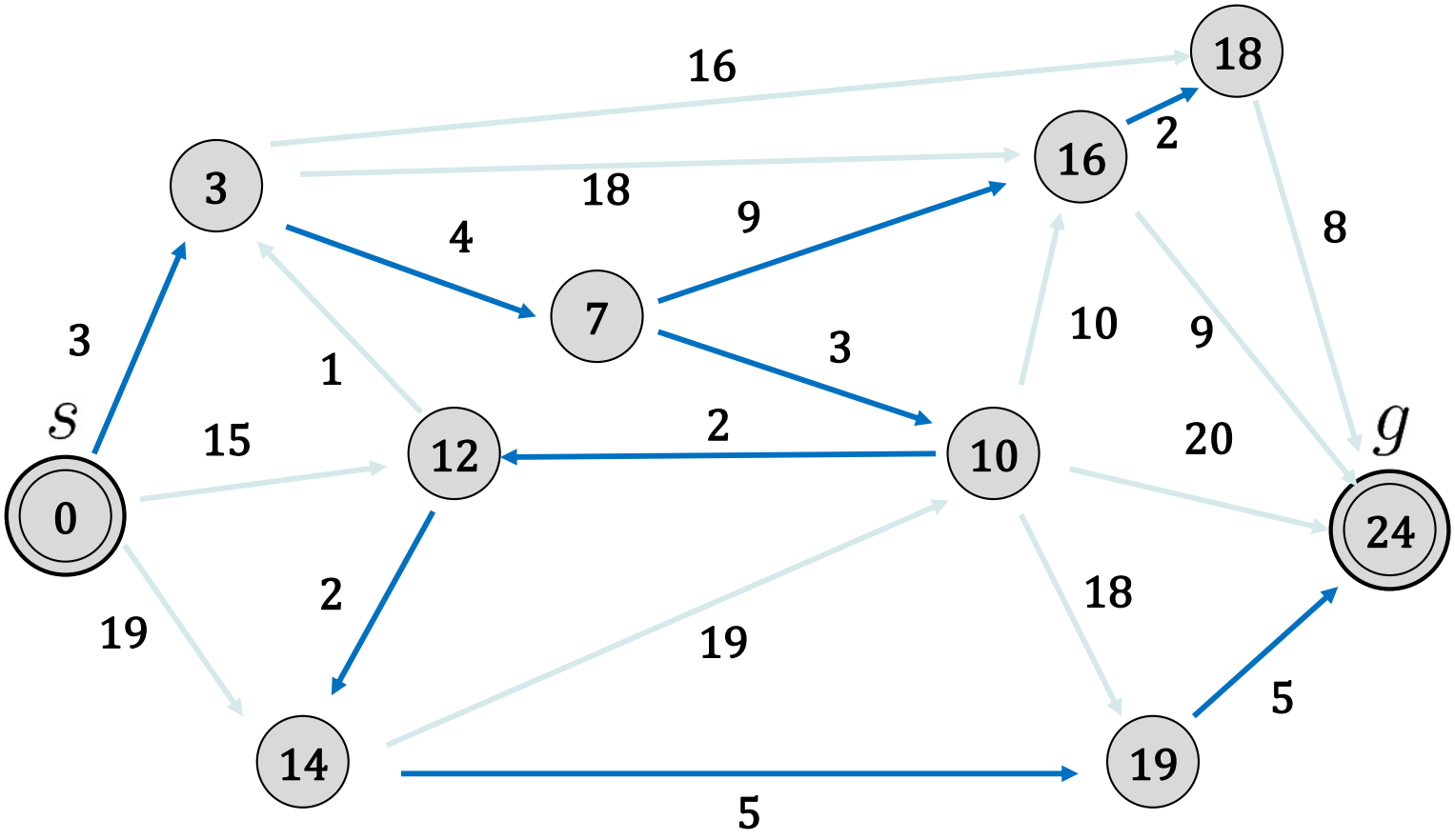
二周目（同じ順番で点をみる）



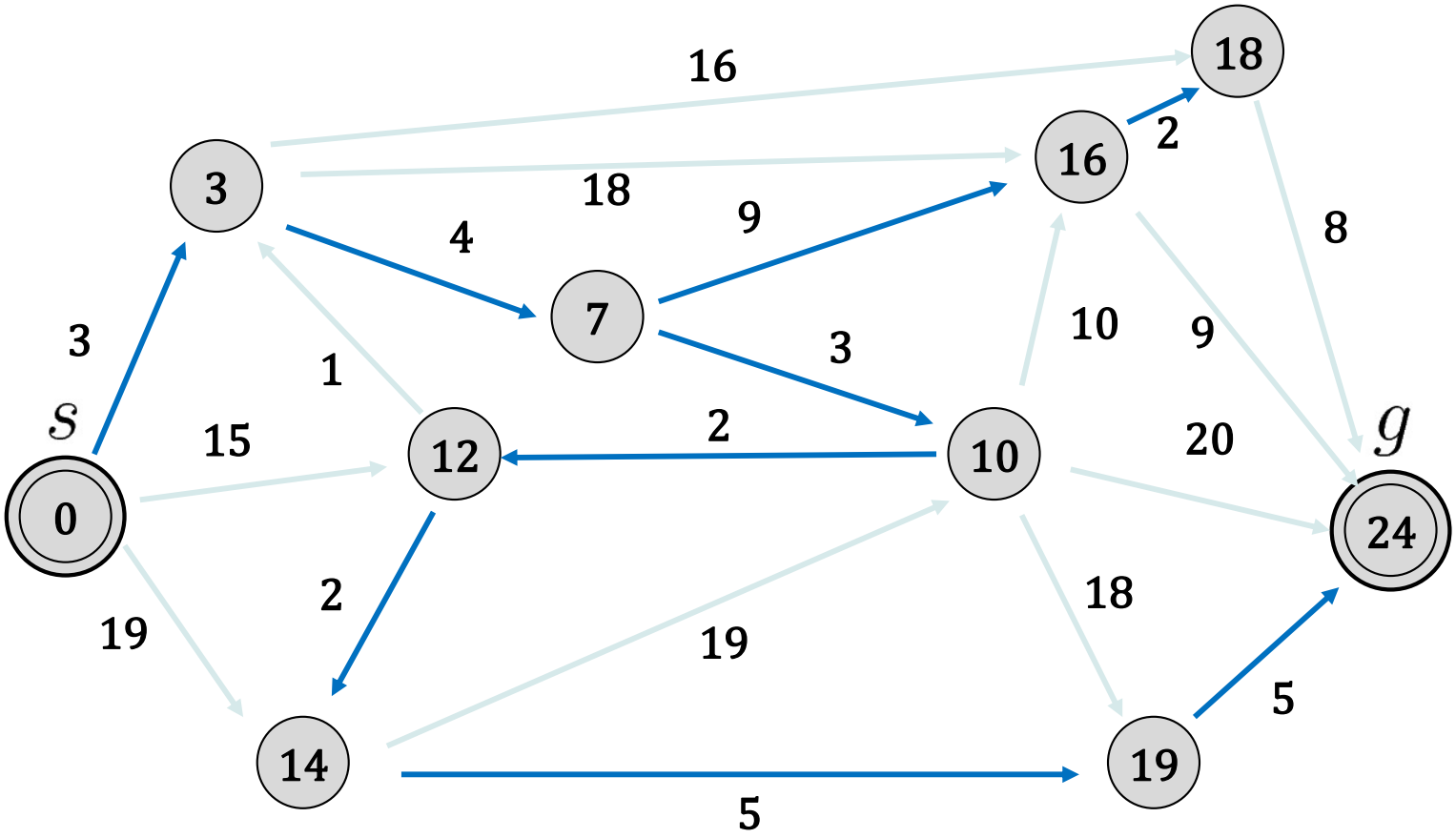
二周目（同じ順番で点をみる）



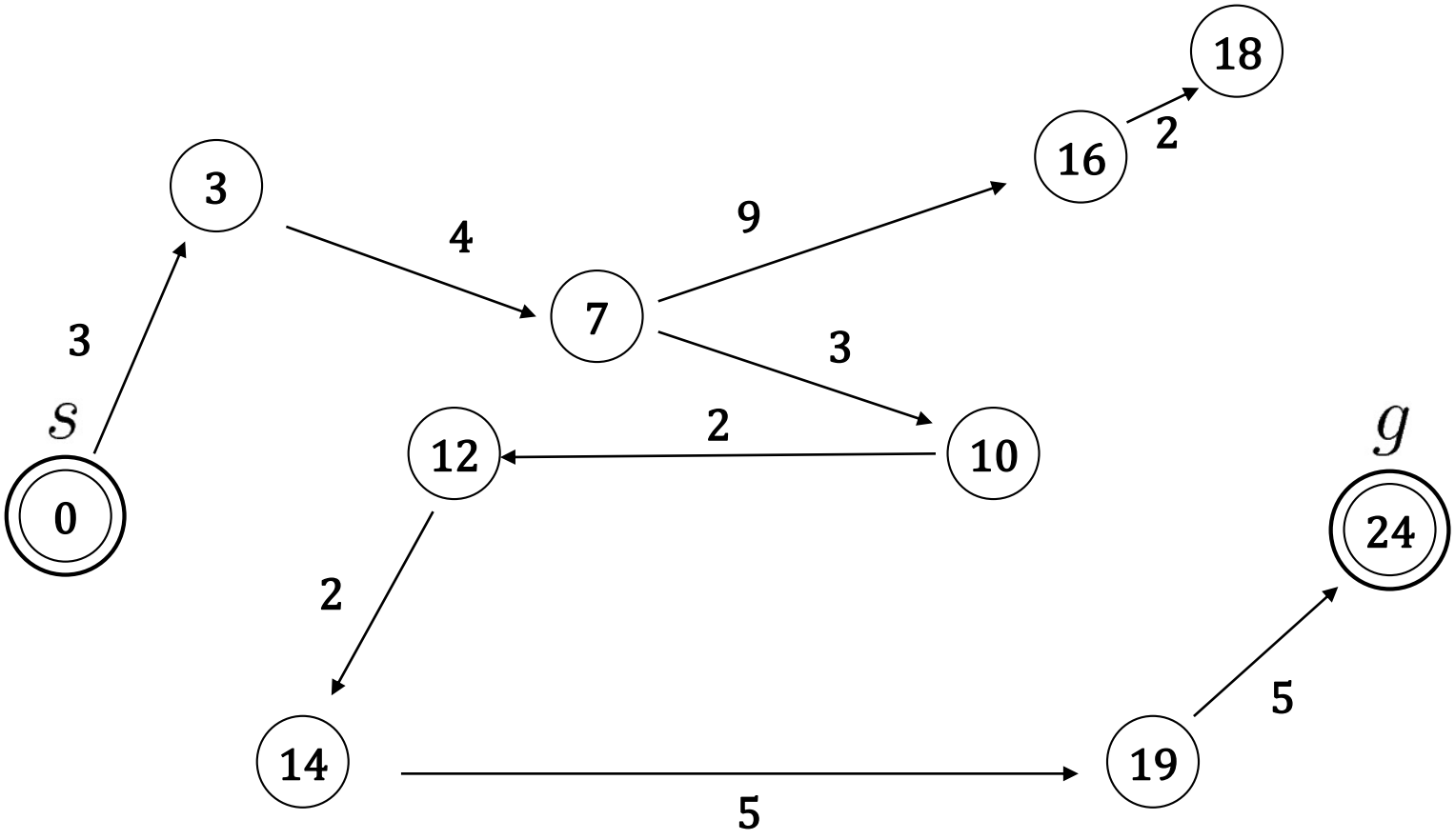
二周目終わり



二周目終わり



解（頂点の数字は出発点からの距離を表す）



ベルマン・フォード法

- いまの手順（アルゴリズム）を
ベルマン・フォード法（Bellman–Ford algorithm）
という。

手順

1. 頂点の順番を決める（出発点は必ず最初）

2. i 番目の頂点から出ている辺とその先の頂点をみる
→距離の更新
（これを最後の点まで繰り返す）

3. 探索後に更新された点があればもう一度2をやり直す。
なければ終了。

ベルマン・フォード法

特徴

- 各頂点からの経路を総当たり的に調べる（しらみ潰しよりはマシ）
- 出発点からすべての点への経路と最小距離が求まる
- 調べた後に値が変更される点が残っている限り、何周でも続いてしまう

ベルマン・フォード法

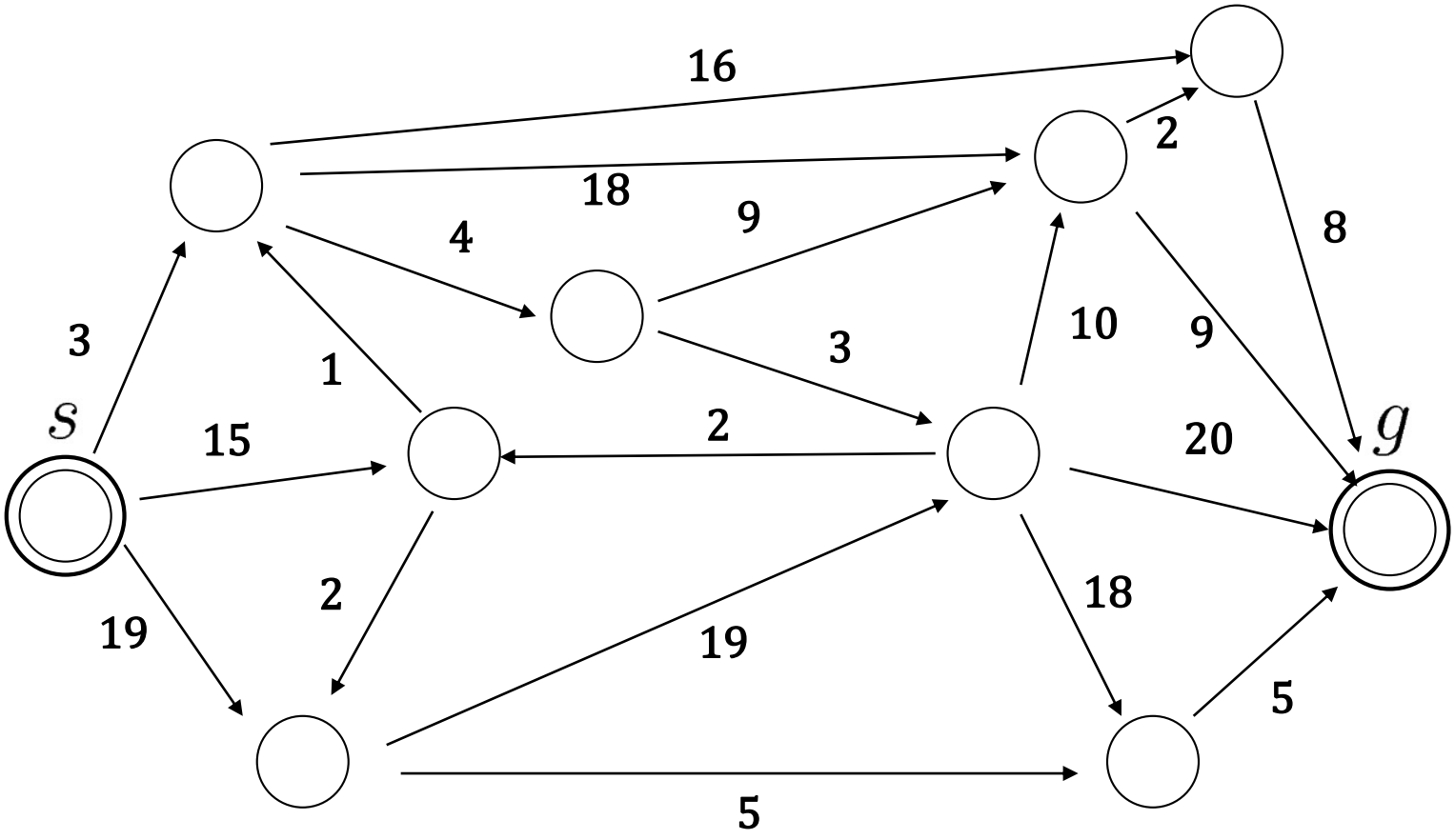
特徴

- 各頂点からの経路を総当たり的に調べる（しらみ潰しよりはマシ）
- 出発点からすべての点への経路と最小距離が求まる
- 調べた後に値が変更される点が残っている限り、何周でも続いてしまう

もっといいアルゴリズムは？

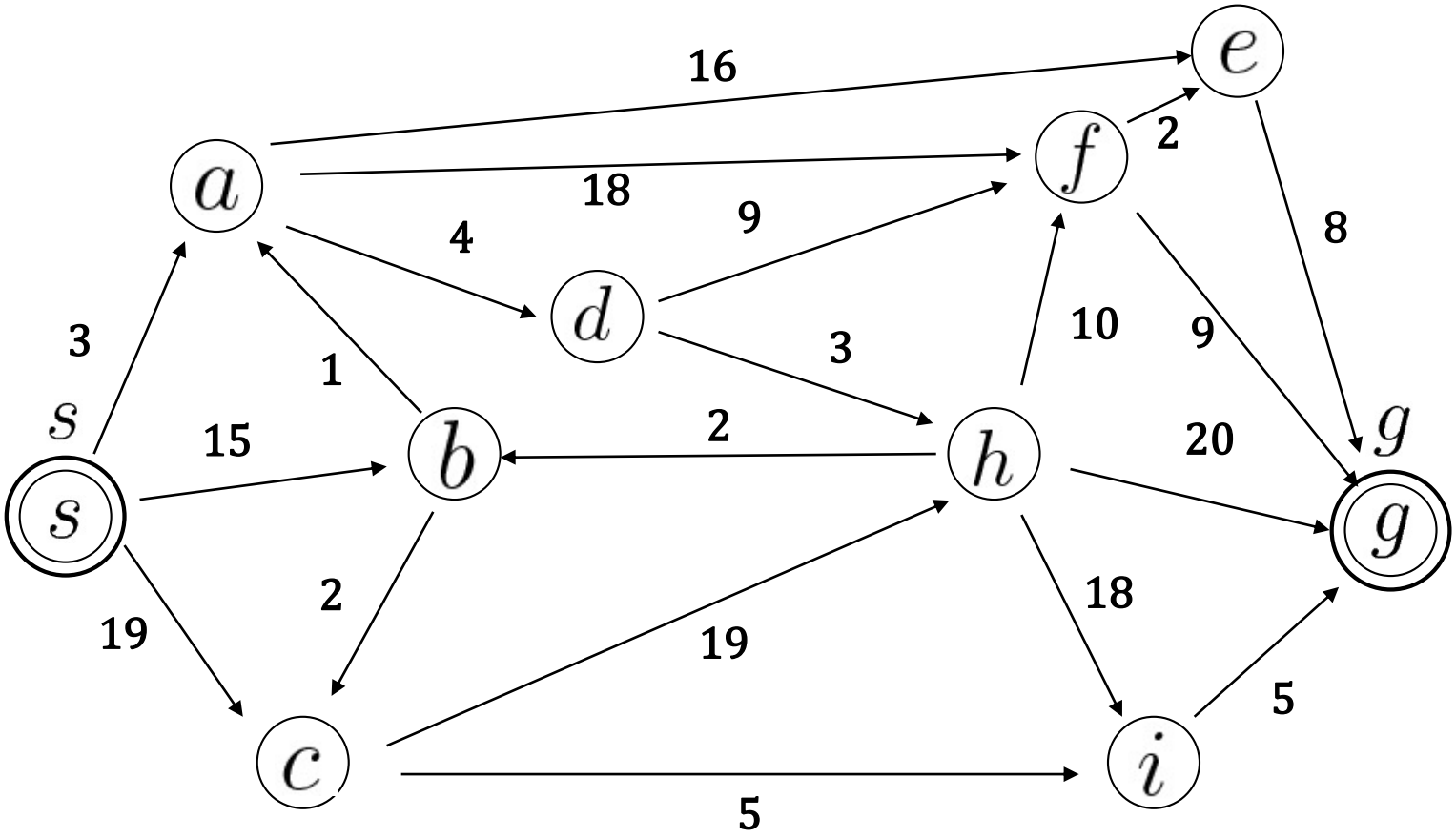
もっといいアルゴリズム

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）

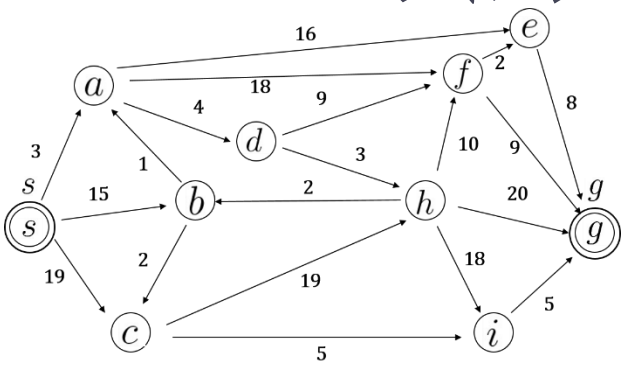


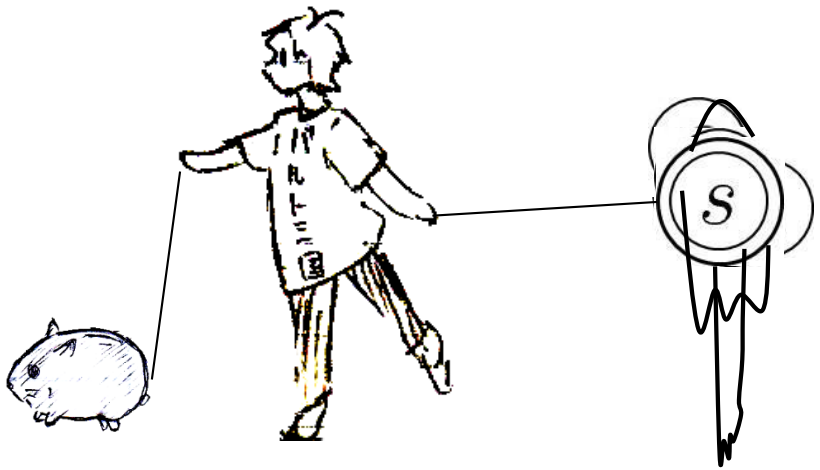
もっといいアルゴリズム

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ．
（辺の重みは頂点間の距離を表す．）

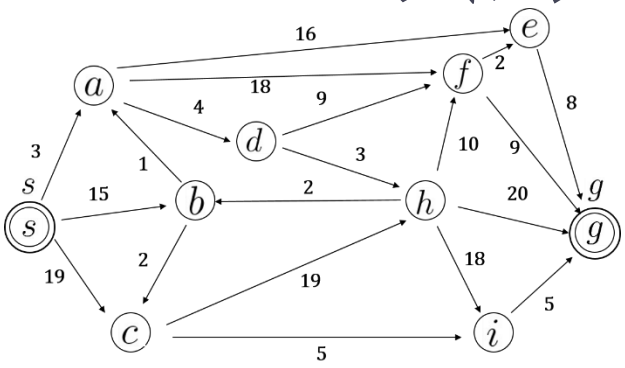


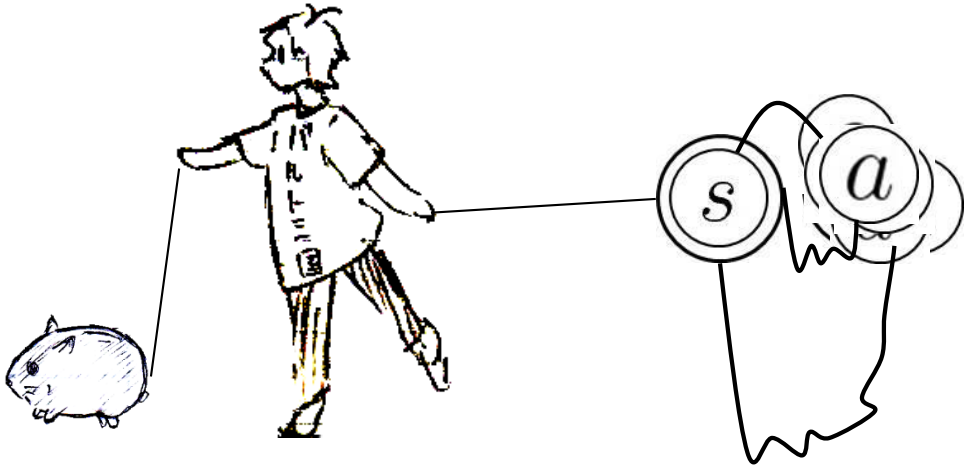
ひっぱり法



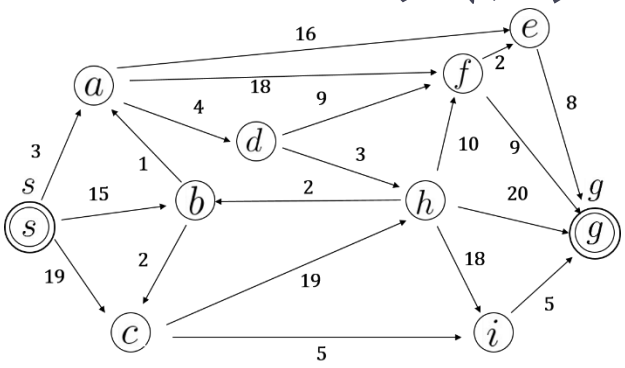


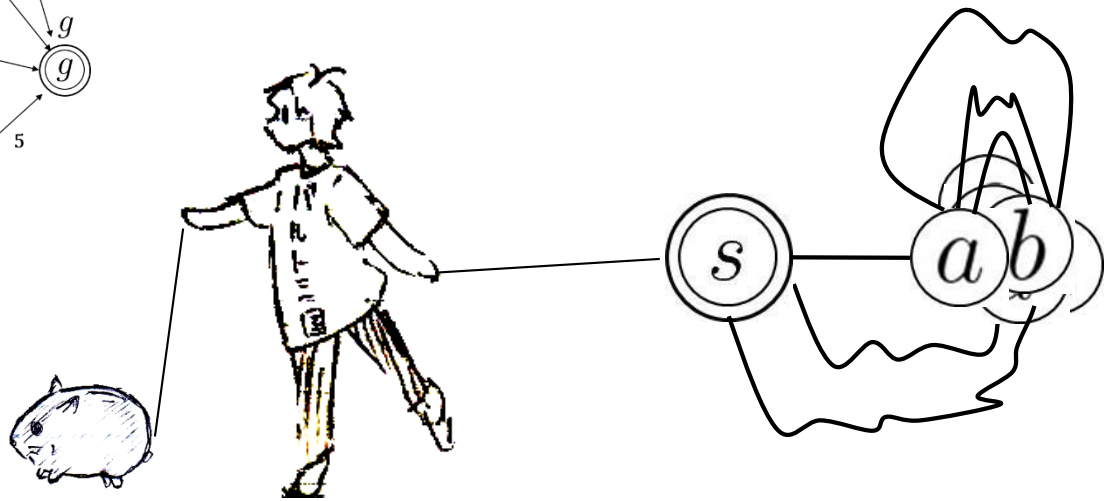
ひっぱり法



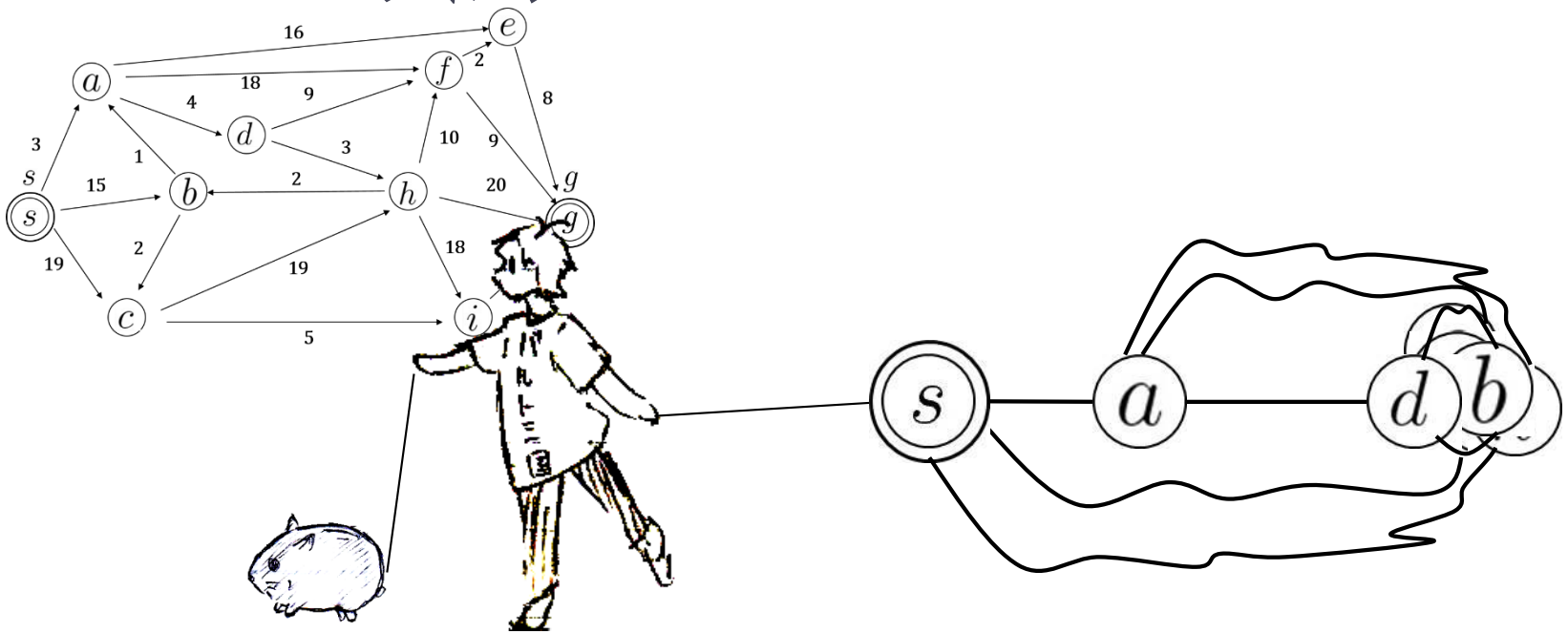


ひっぱり法

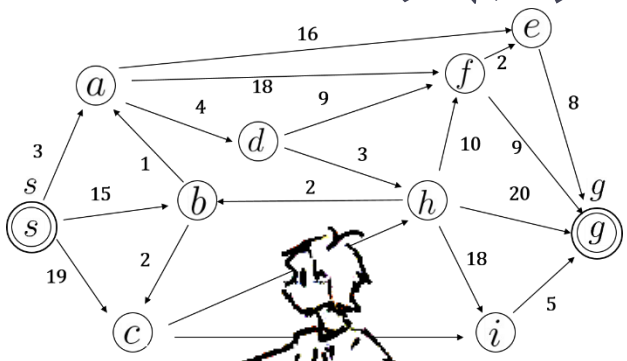


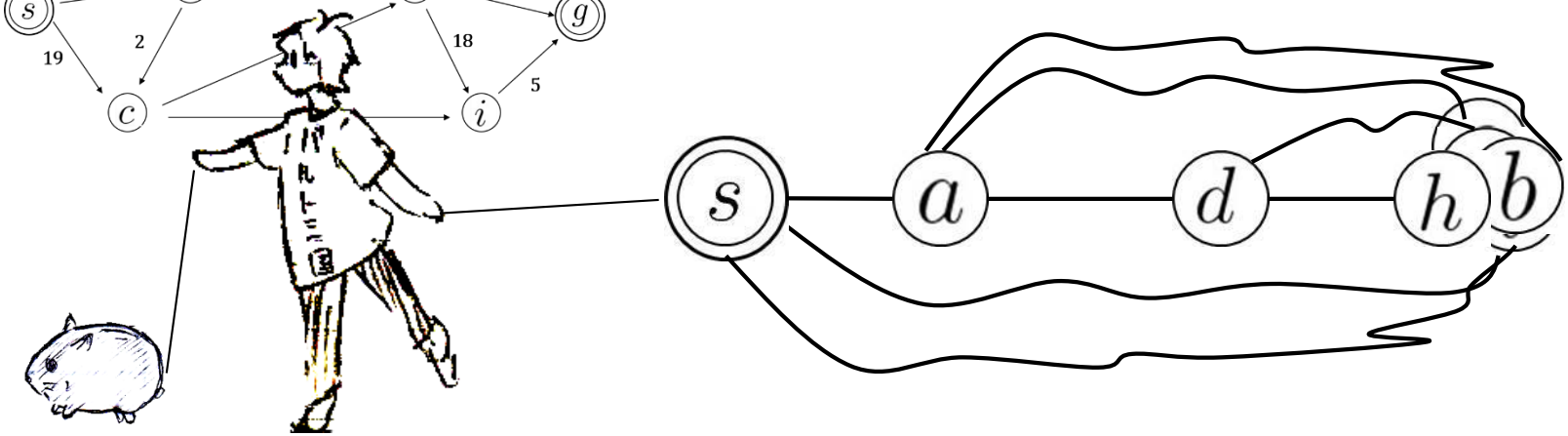


ひっぱり法

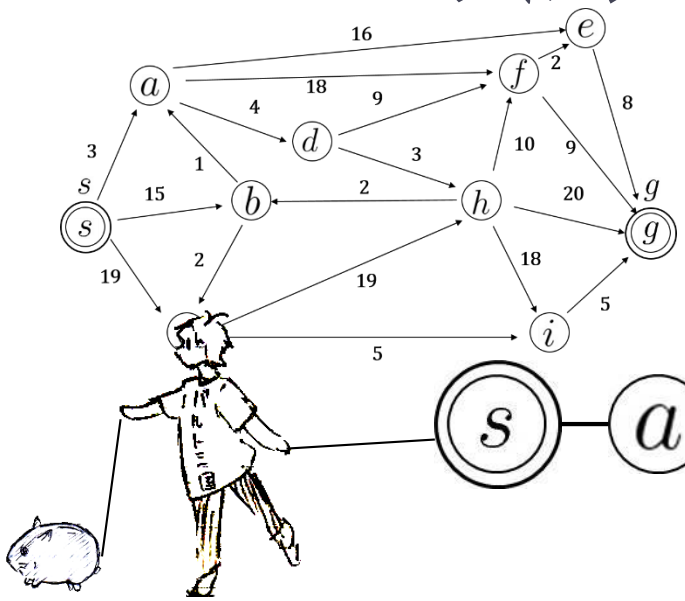


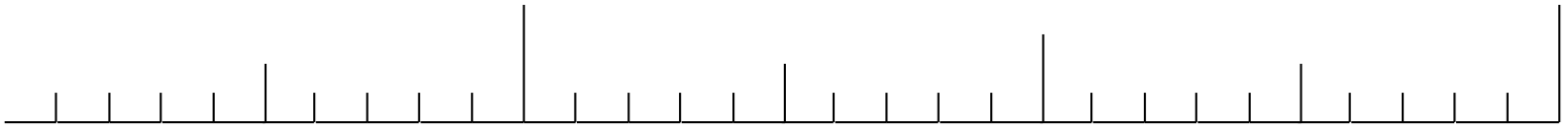
ひっぱり法





ひっぱり法





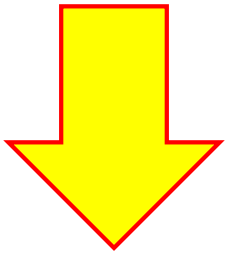
「ひっぱり法」は
結局何をしている？

「ひっぱり法」の本質

- 頂点を始点から**近い順**に見つけてくる
- 一度糸がピンと張ったら**それ以上短くならない**

「ひっぱり法」の本質

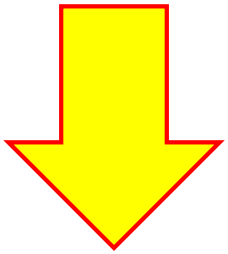
- 頂点を始点から**近い順**に見つけてくる
- 一度糸がピンと張ったら**それ以上短くならない**



頂点を始点から**近い順**に見つけ、
その距離を**確定**してゆくアルゴリズム

「ひっぱり法」の本質

- 頂点を始点から**近い順**に見つけてくる
- 一度糸がピンと張ったら**それ以上短くならない**



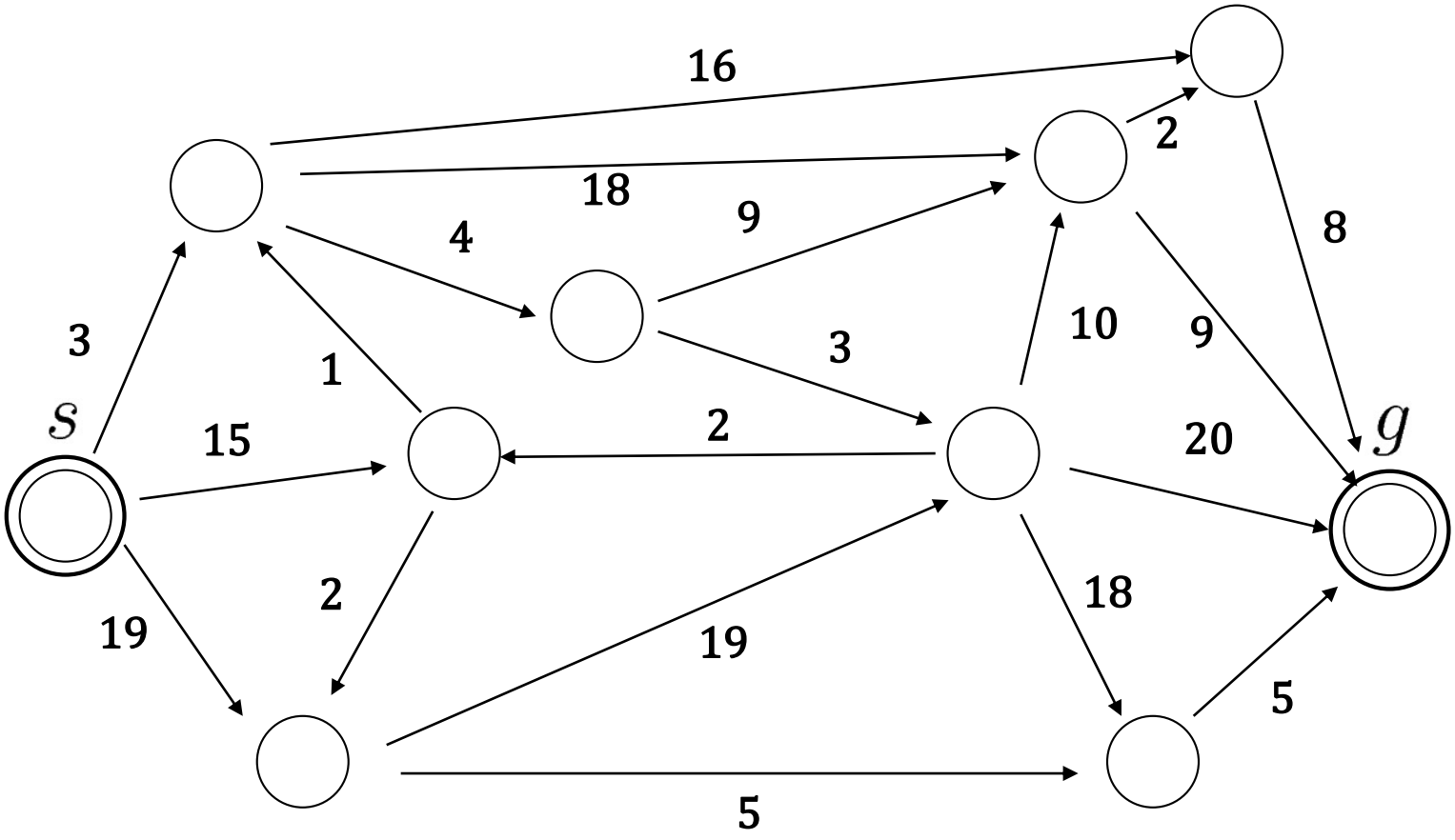
頂点を始点から**近い順**に見つけ、
その距離を**確定**してゆくアルゴリズム

=

ダイクストラ法（Dijkstra’s algorithm）

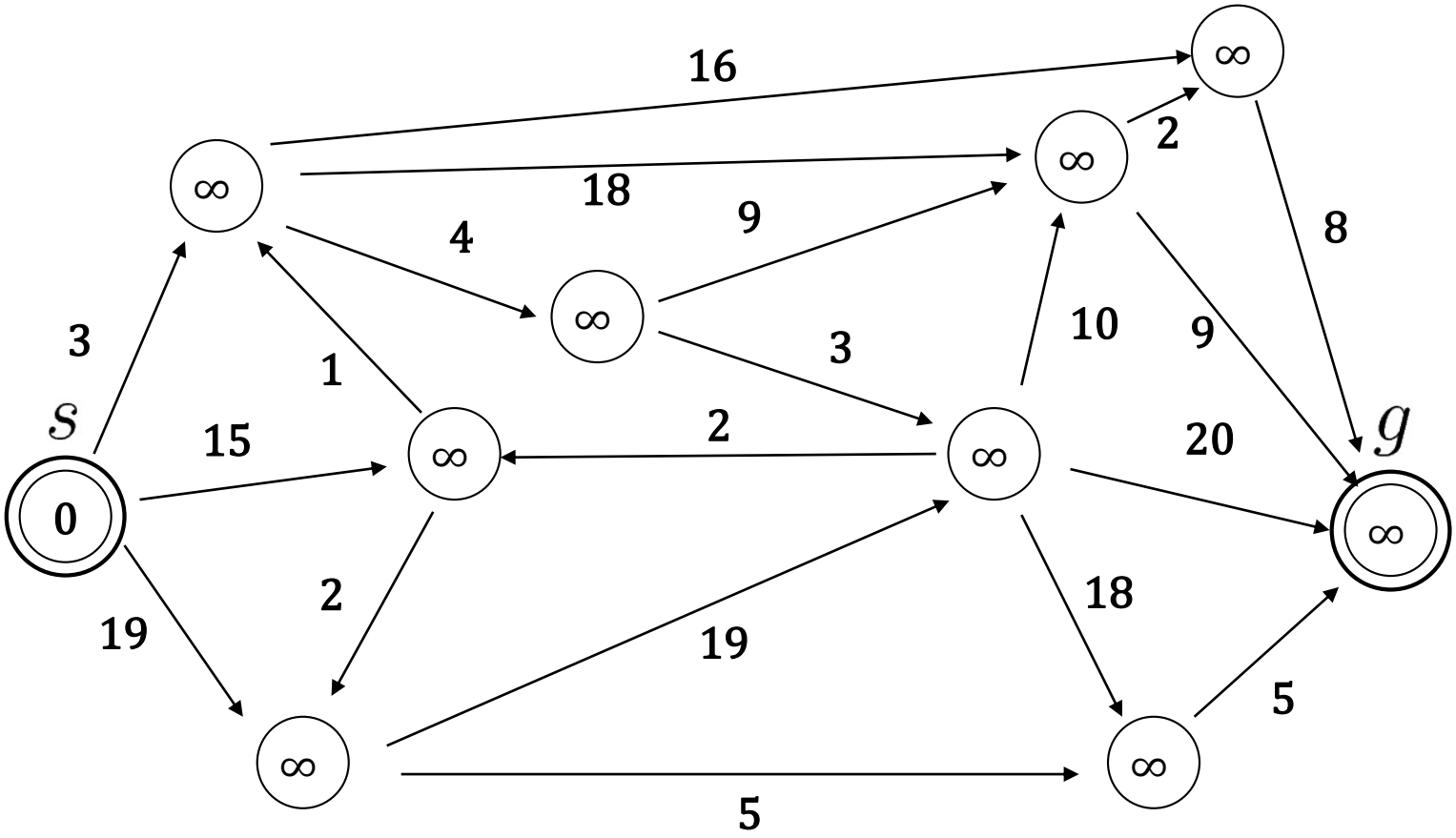
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



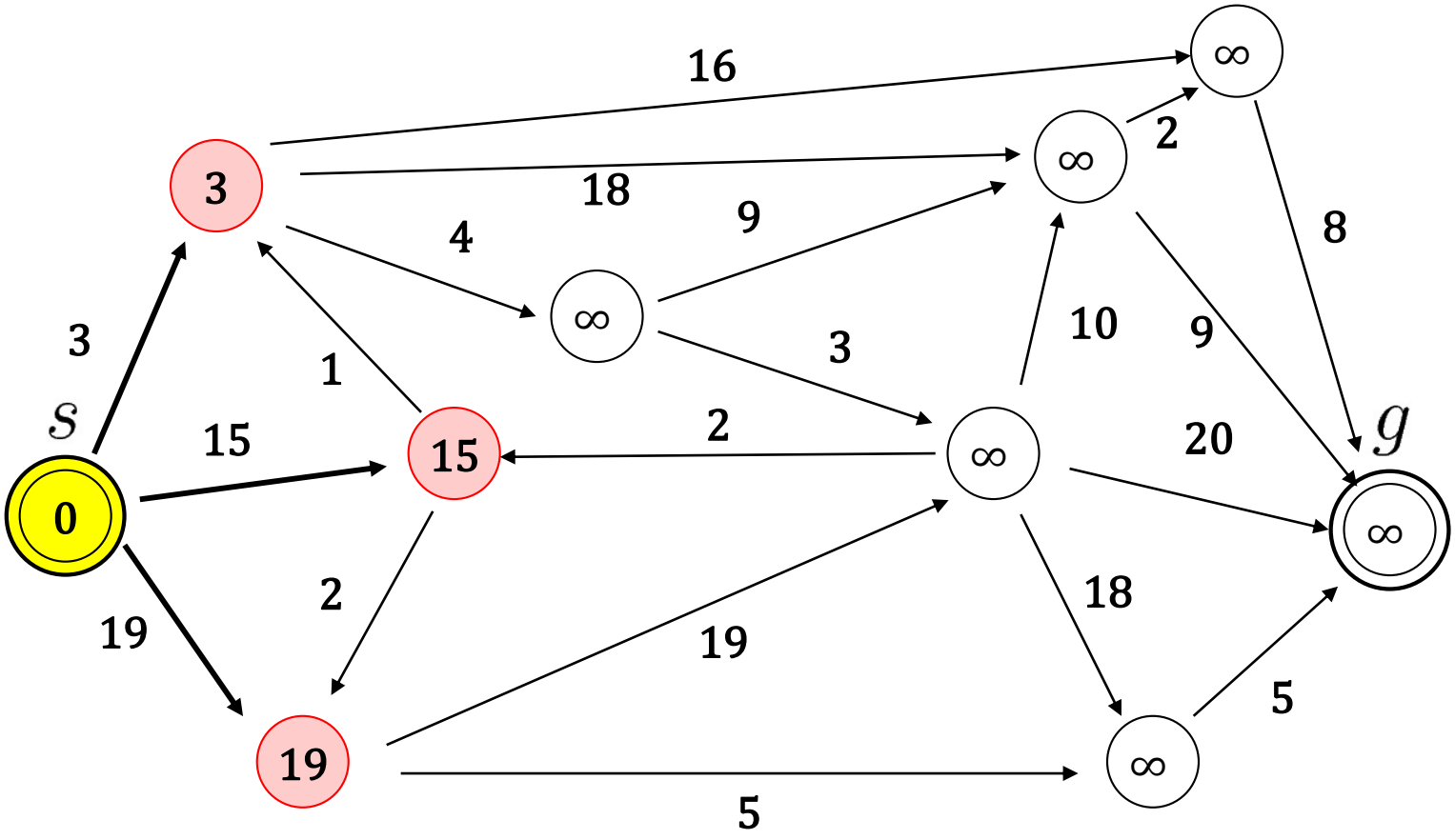
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



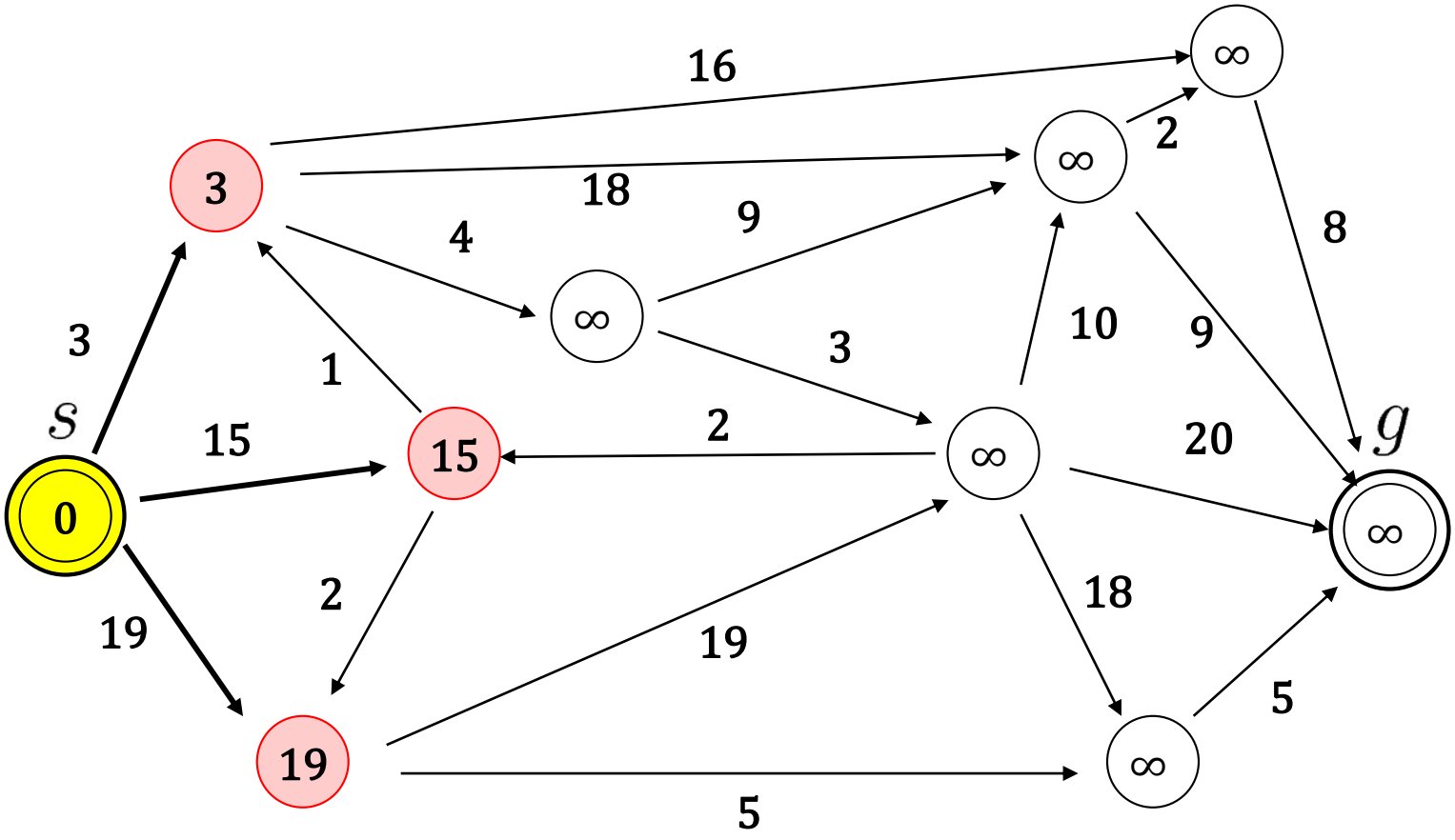
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



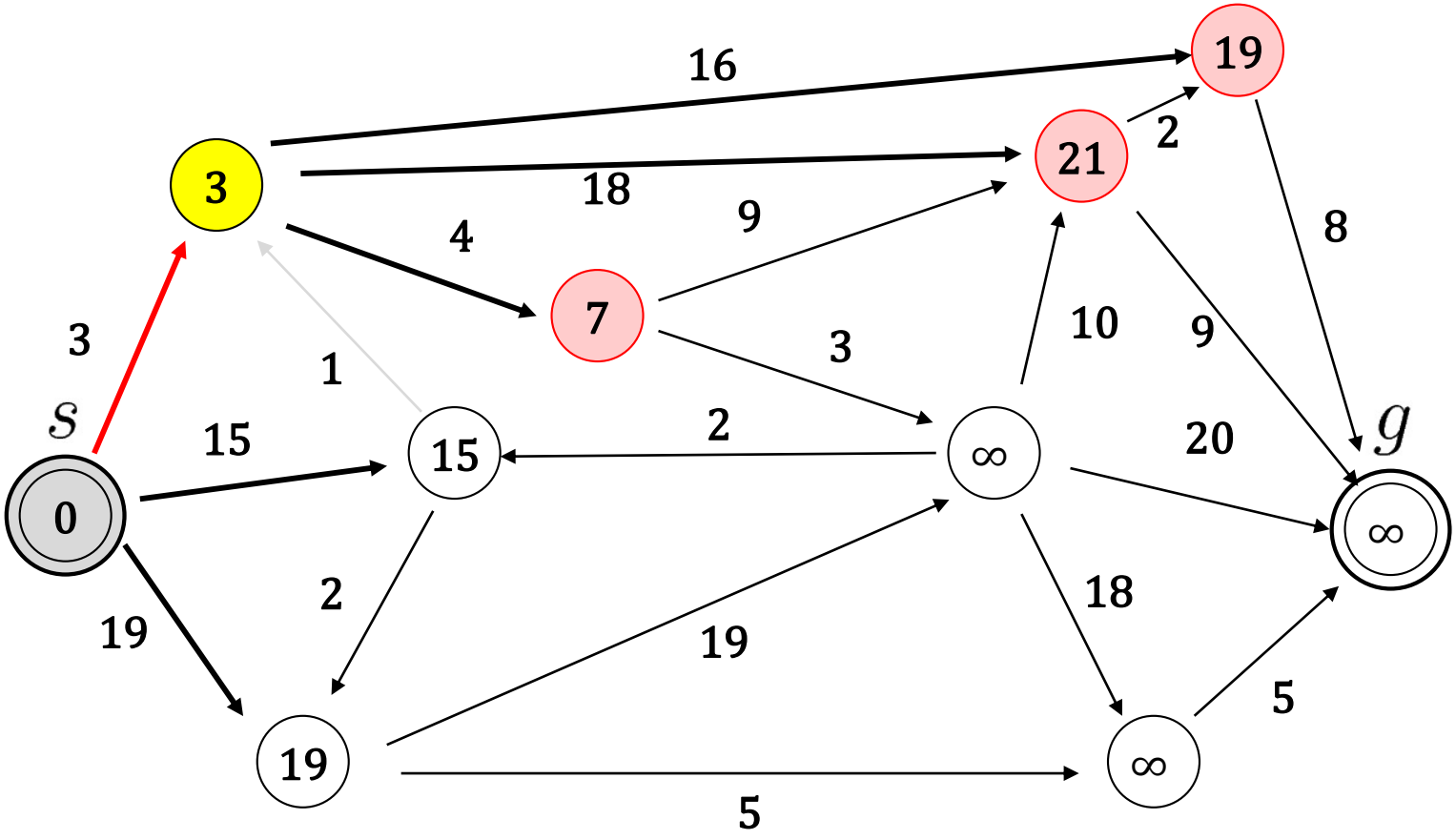
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ．
（辺の重みは頂点間の距離を表す．）



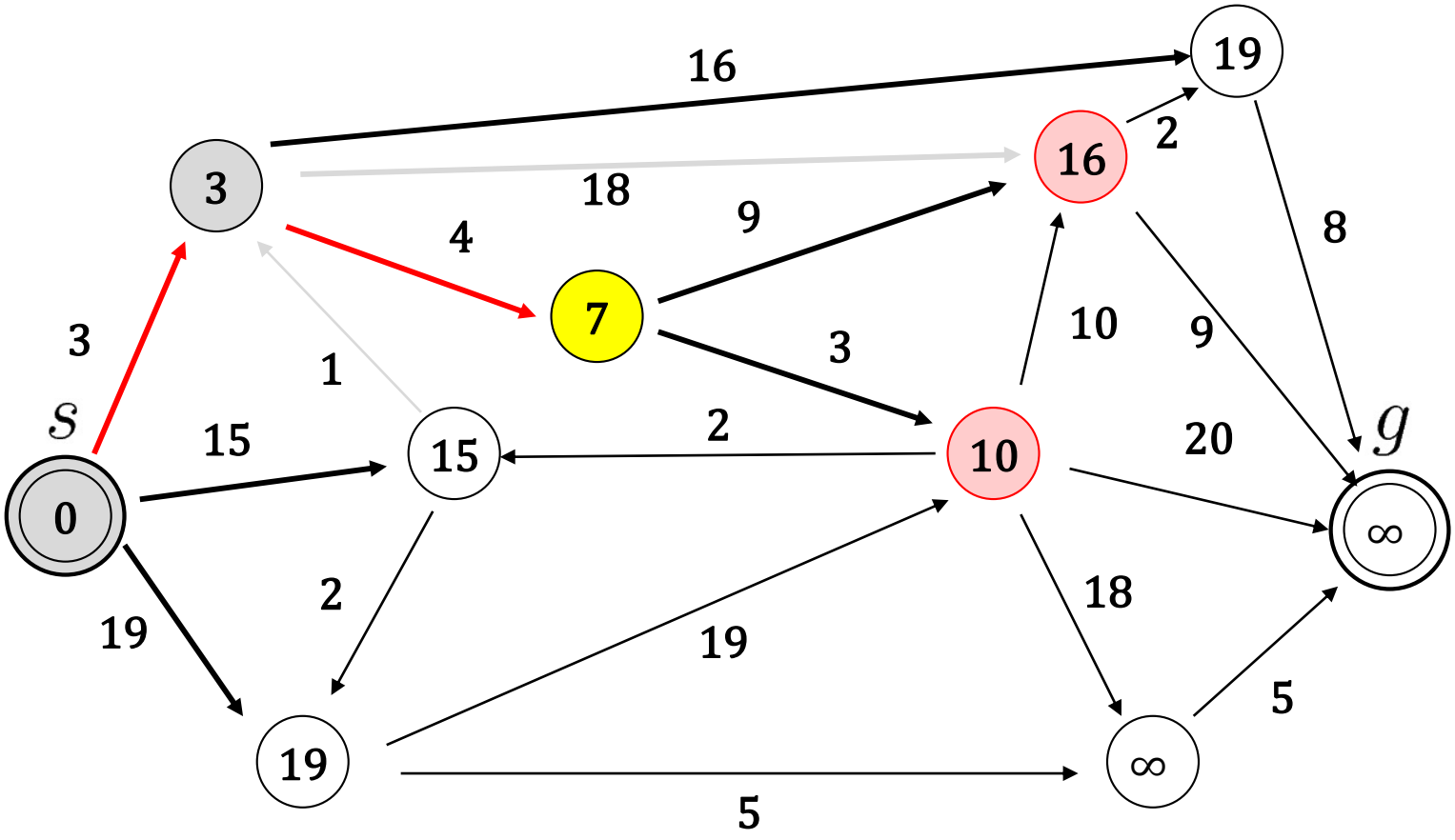
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ．
（辺の重みは頂点間の距離を表す．）



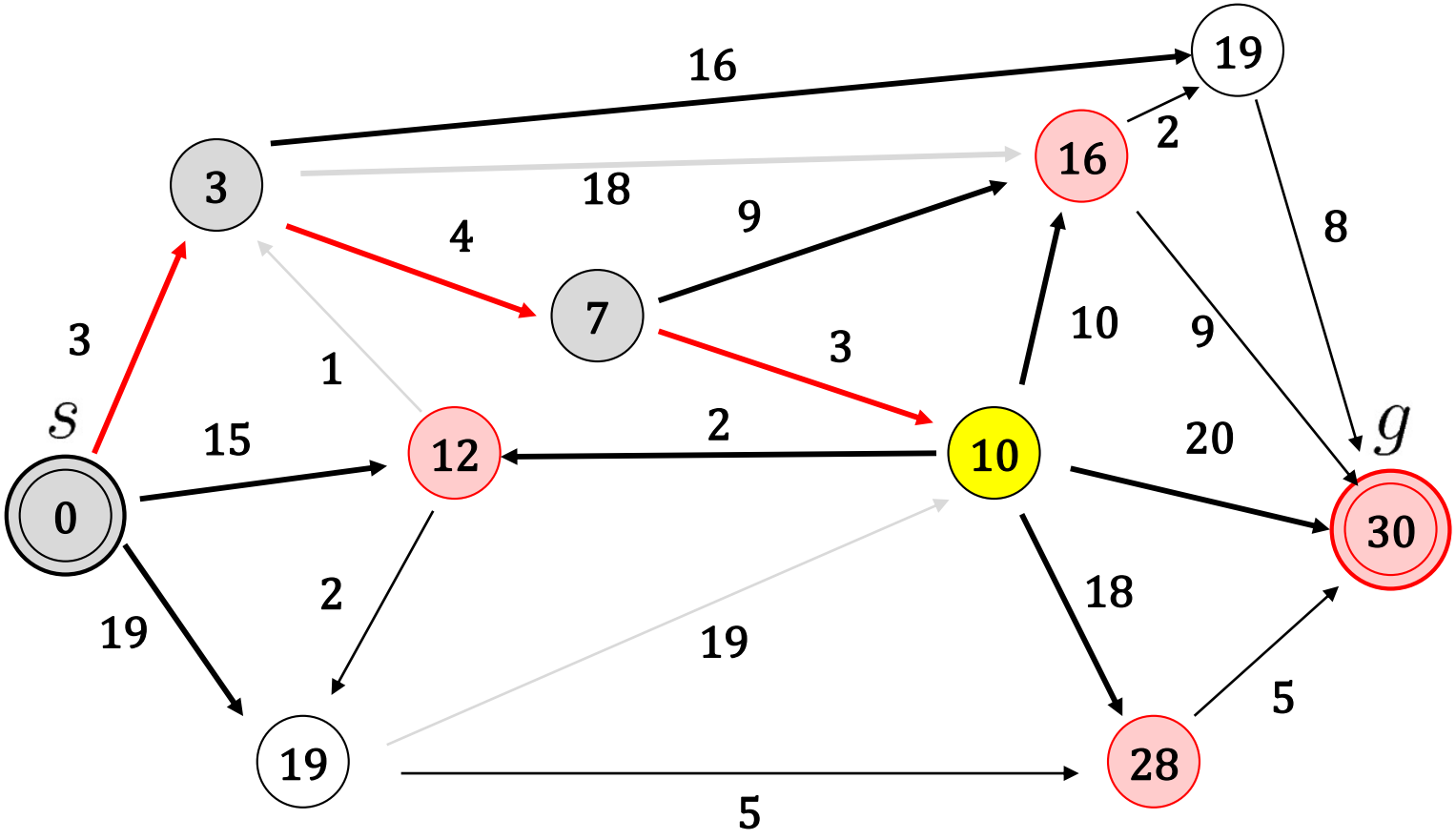
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



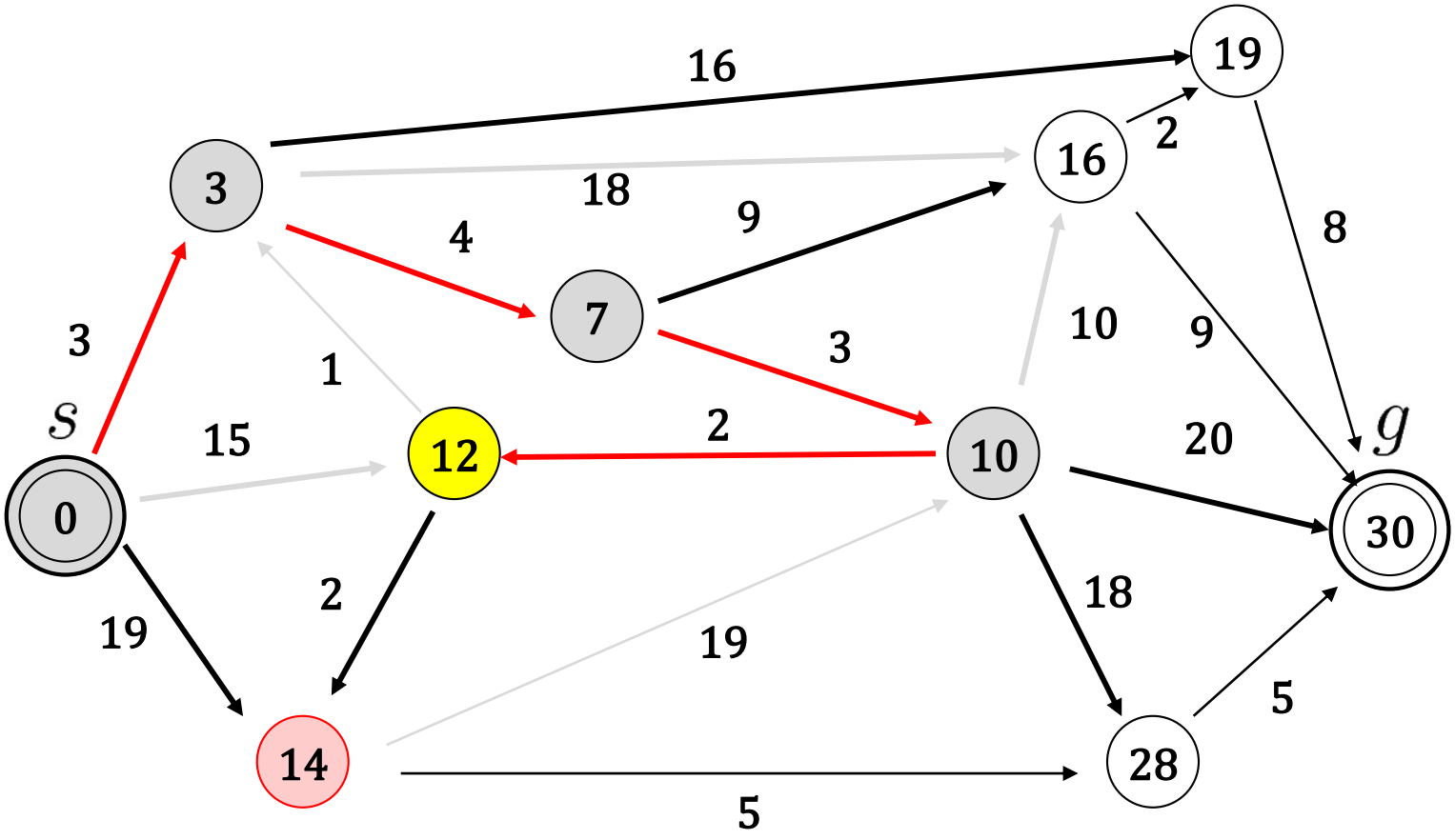
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ．
（辺の重みは頂点間の距離を表す．）



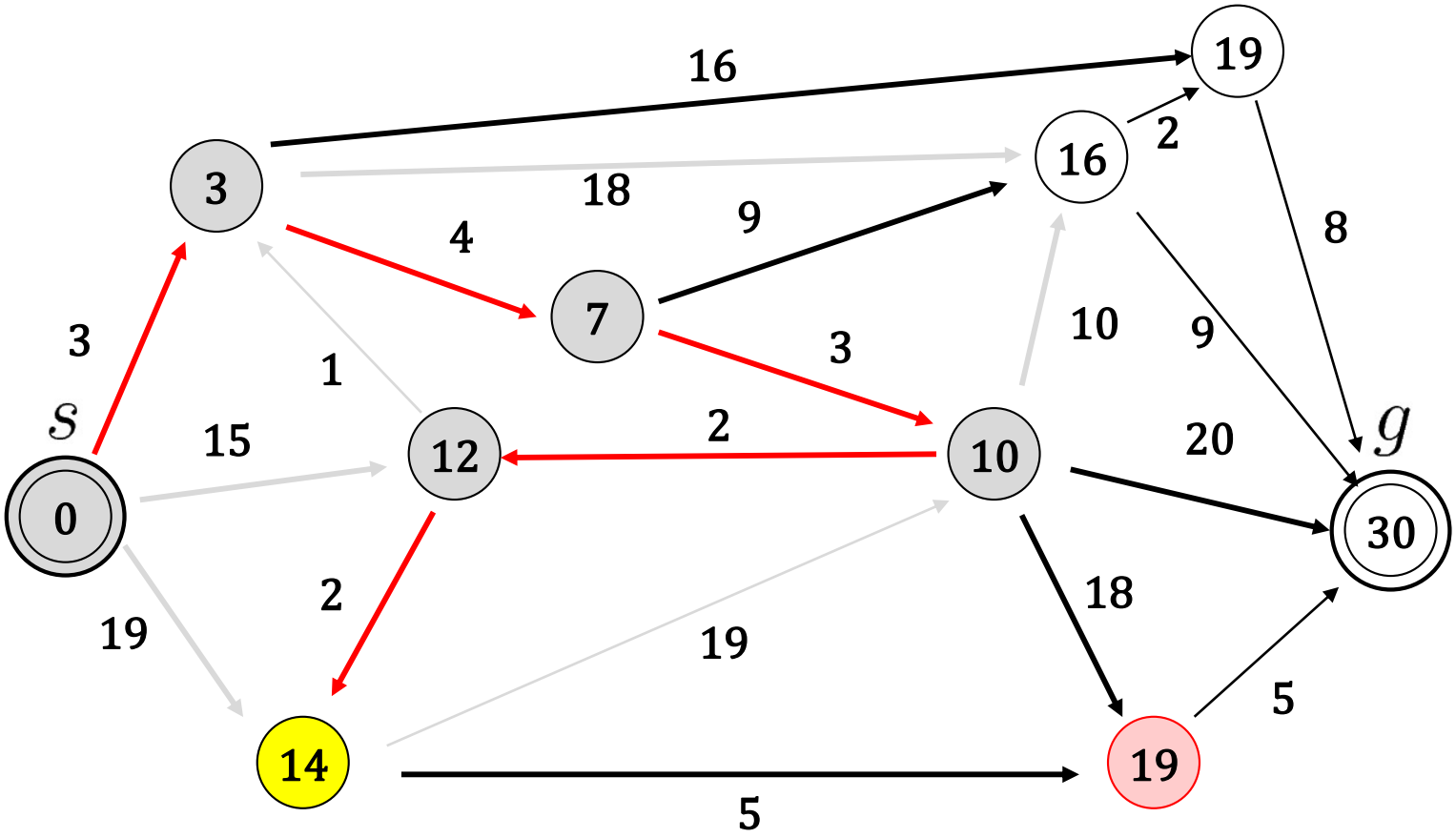
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



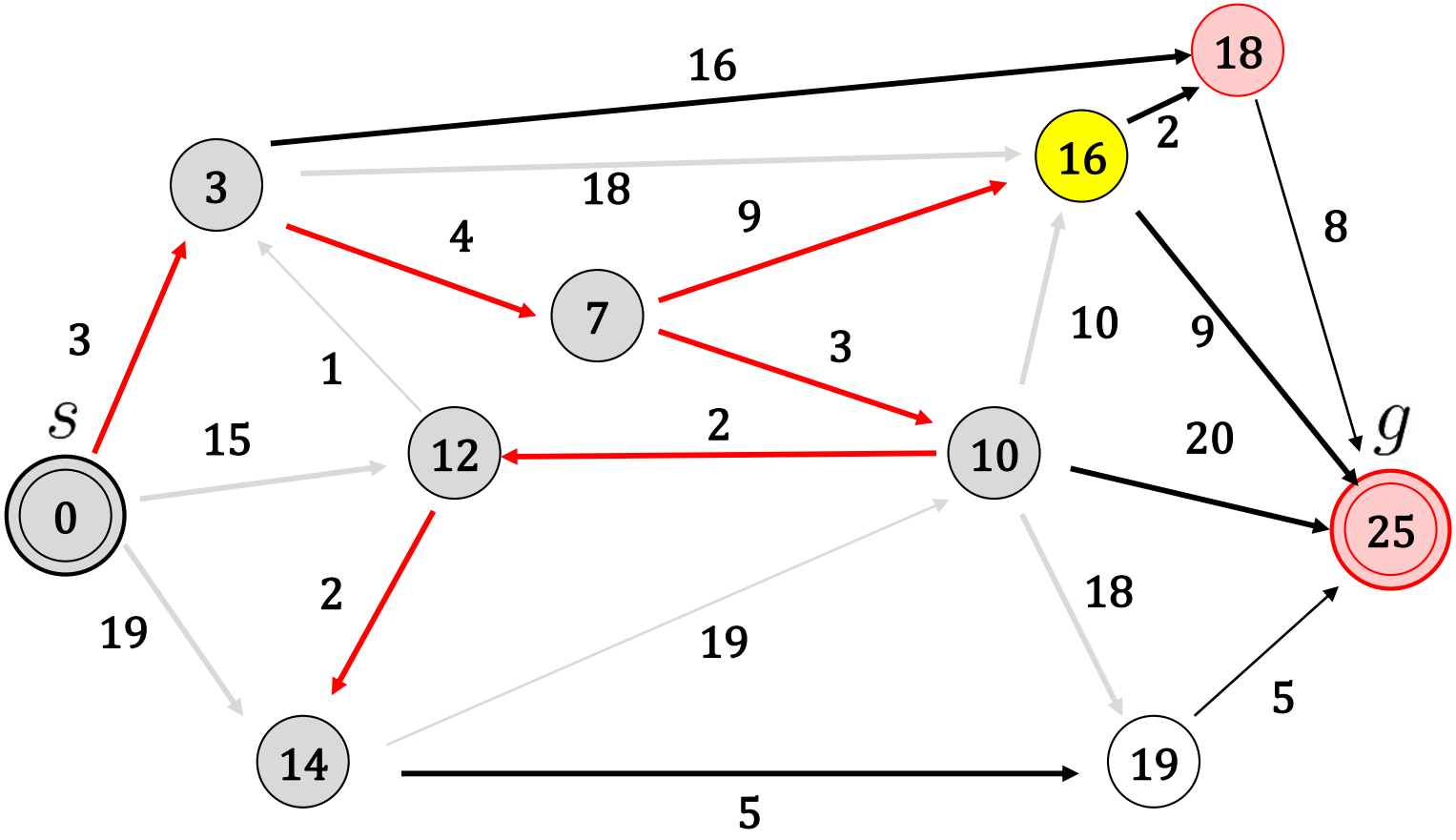
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



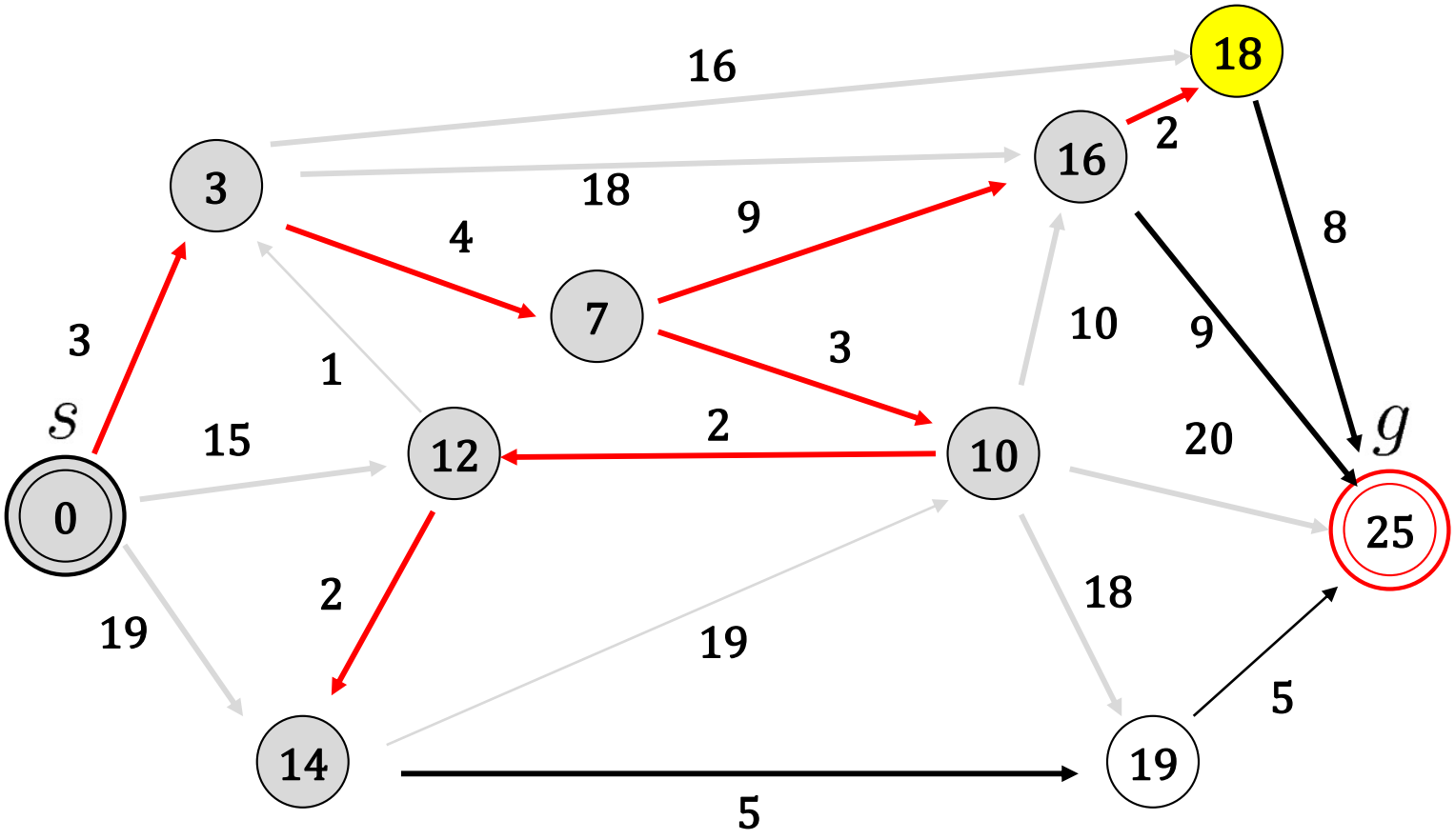
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ.
（辺の重みは頂点間の距離を表す．）



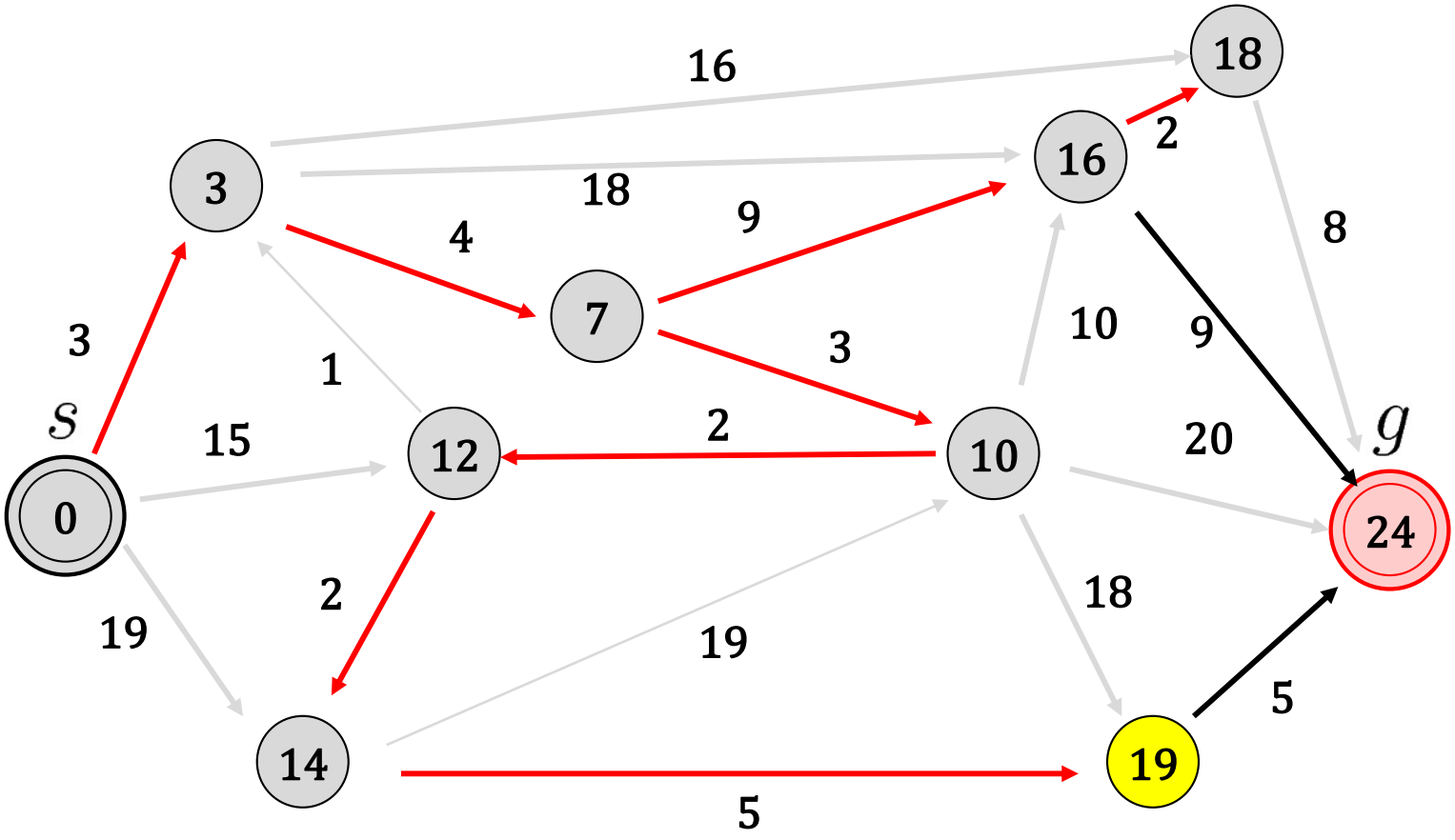
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ．
（辺の重みは頂点間の距離を表す．）



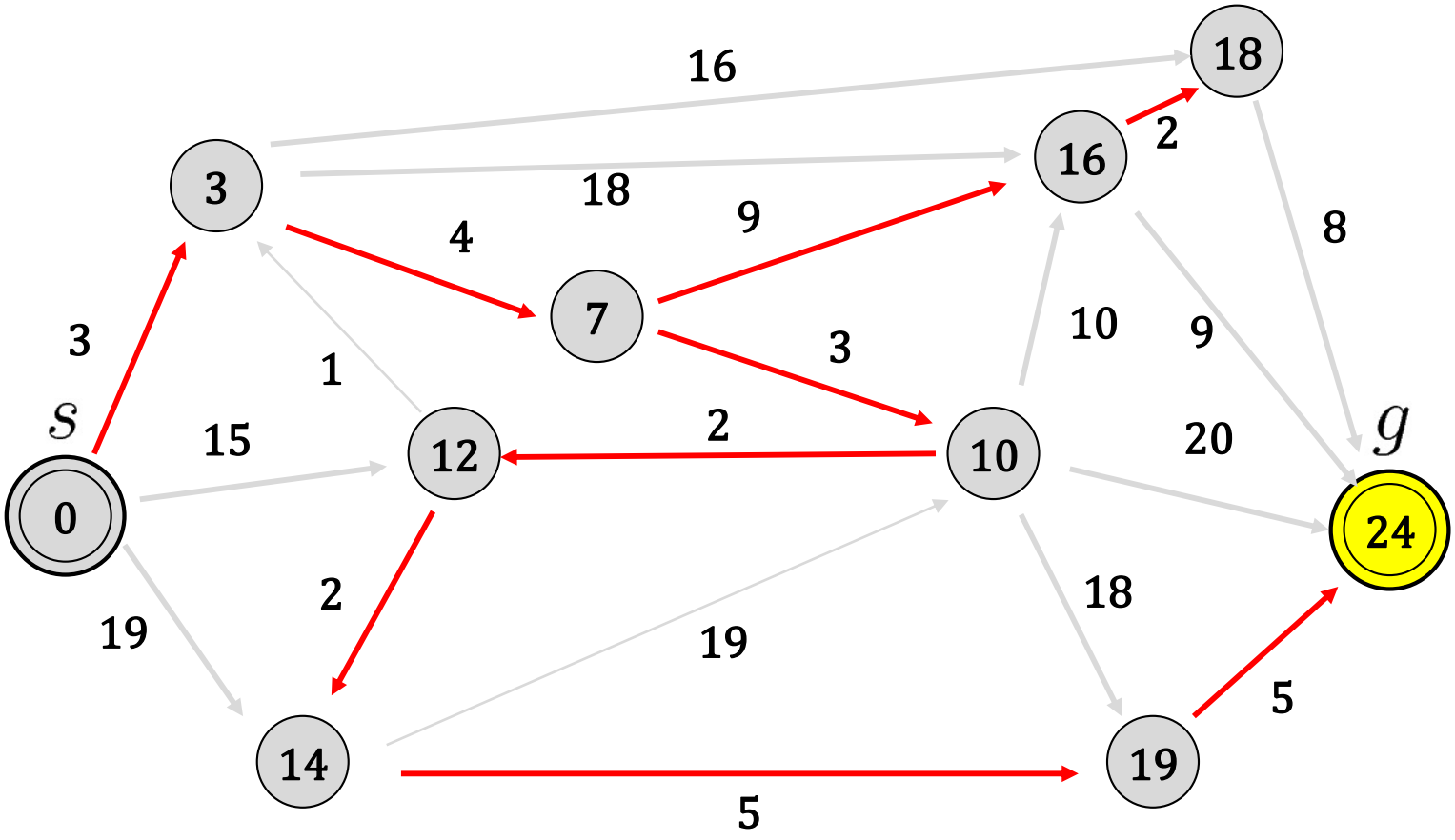
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ．
（辺の重みは頂点間の距離を表す．）



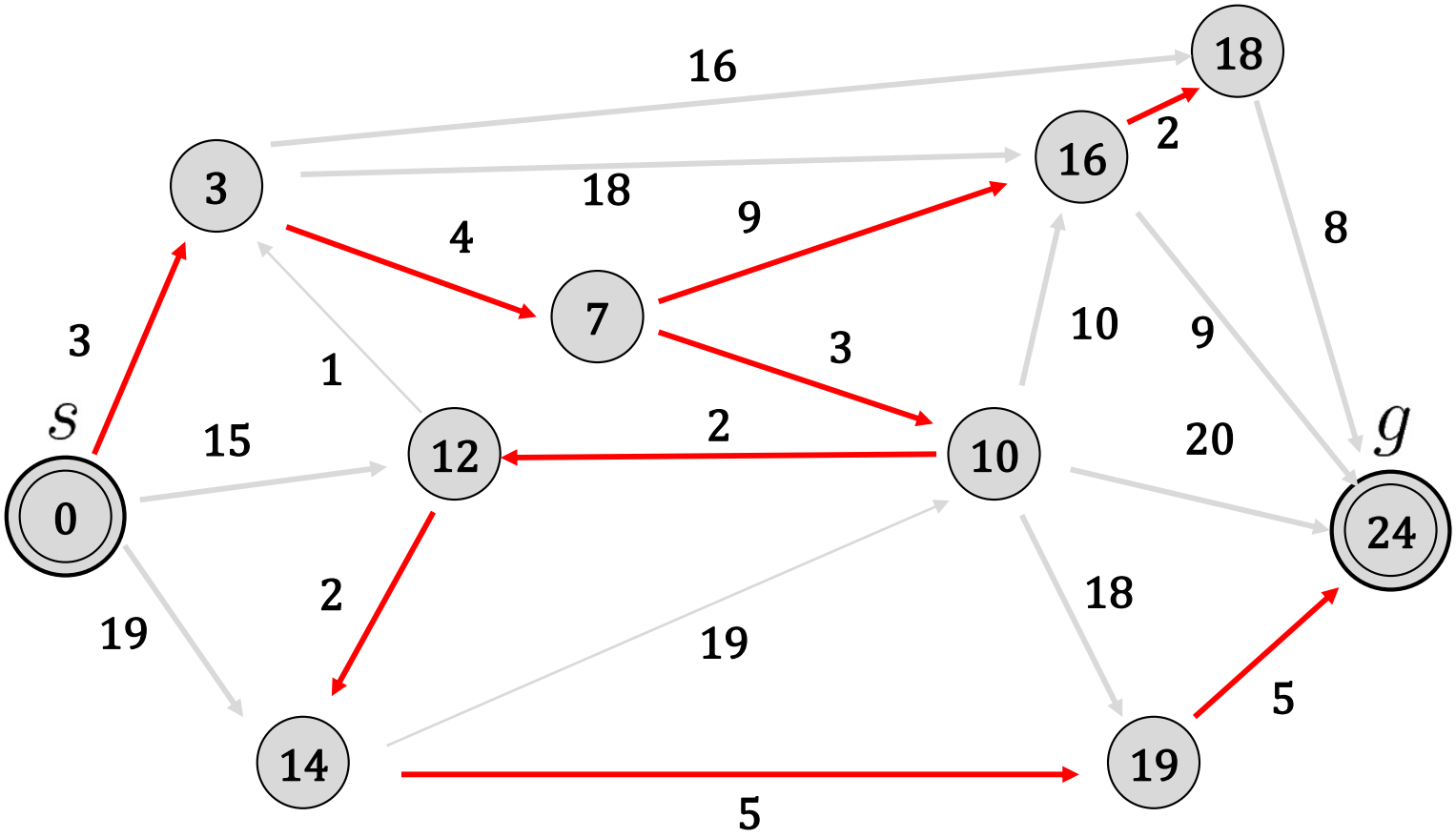
ダイクストラ法（Dijkstra's algorithm）

出発地点 s から目的地点 g まで最短距離で行くルートを求めよ．
（辺の重みは頂点間の距離を表す．）



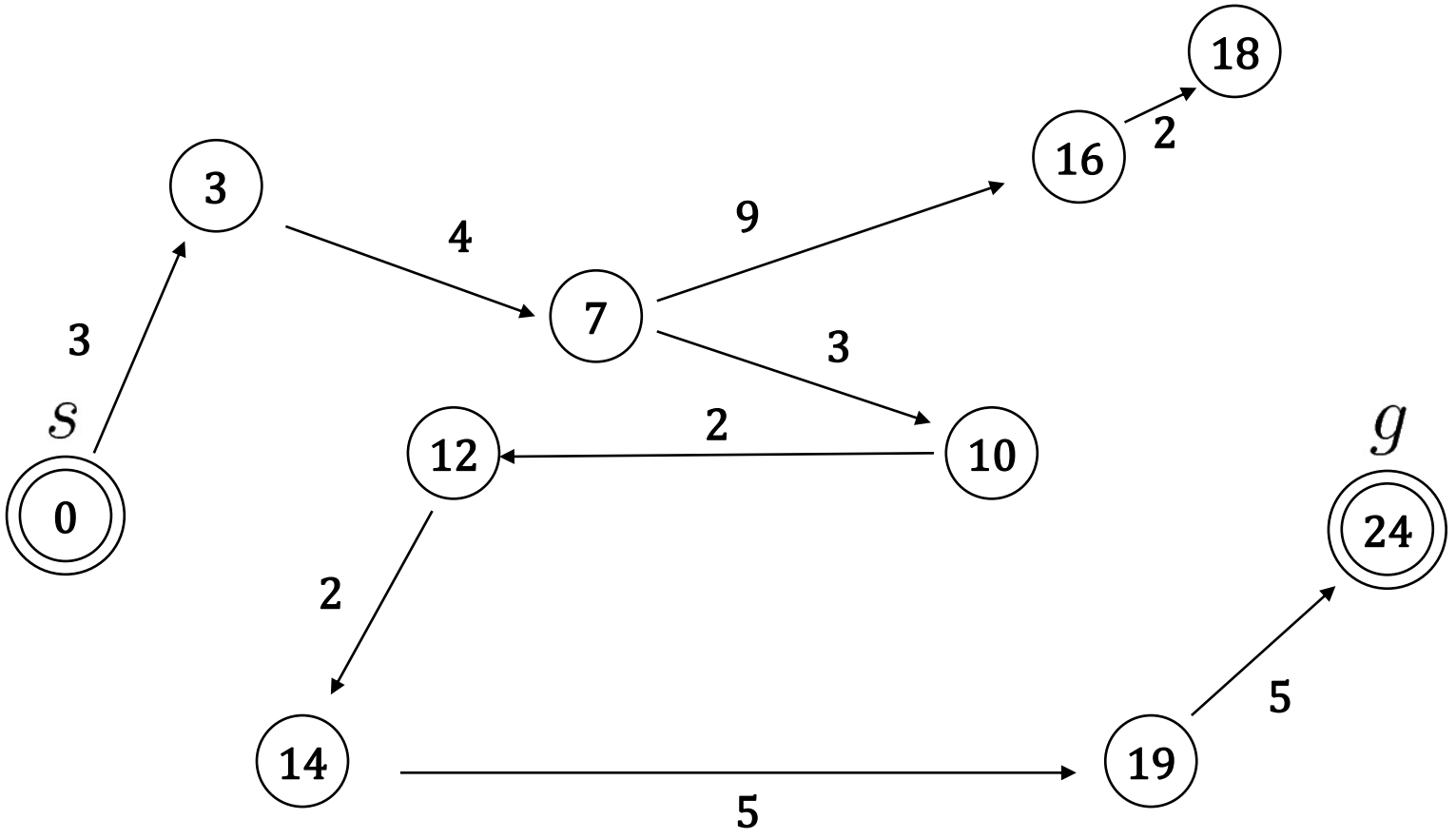
ダイクストラ法（Dijkstra's algorithm）

終了



ダイクストラ法（Dijkstra's algorithm）

解



ダイクストラ法の特徴

- 出発点から近い順に点を見つけ出してゆく
- 出発点からすべての点への経路と最小距離が求まる

ベルマン・フォード法との比較

- 必ず一周目で解が求まる
- 全部の辺を調べる必要がない

→ **ダイクストラ法のほうが速い！**

ダイクストラ法の欠点

- 負の重みがあるときにはうまくいかない
  →ベルマン・フォード法は負の重みがあってもよい

一方、すべて正であれば必ずうまくいく

→ このため、実際にカーナビなどで現在広く使われている。

- 必要のない他の点までの最短ルートも求まってしまう
  →目的地だけに絞ればもっと速くなるかも

- 近い順に求まるため、並列処理に向かない
  →並列処理ができればもっと速くなるかも

「引っ張り法」で目的点を固定しなかったら？
「最短経路じゃなさそうな経路」を先に排除できたら？
etc.

まとめ

- グラフの基礎知識

- 最短経路探索問題を解くアルゴリズムとして、
  ベルマン・フォード法と**ダイクストラ法**を紹介した。

- 辺の重さが**すべて正**のときは**ダイクストラ法**がよい。
- 一方、**負の重さ**があるときは**ベルマン・フォード法**がよい。

- どちらのアルゴリズムも、**すべての点**への経路が求まる。

- アルゴリズムの改善をすることで、よりよいアルゴリズムが作れそう。

- http://karly.ran-maru.net/
  に当ファイルおよびサンプルファイル、
  今回の内容の詳細などをアップしています。