現代的な屋敷 (Modern Mansion)

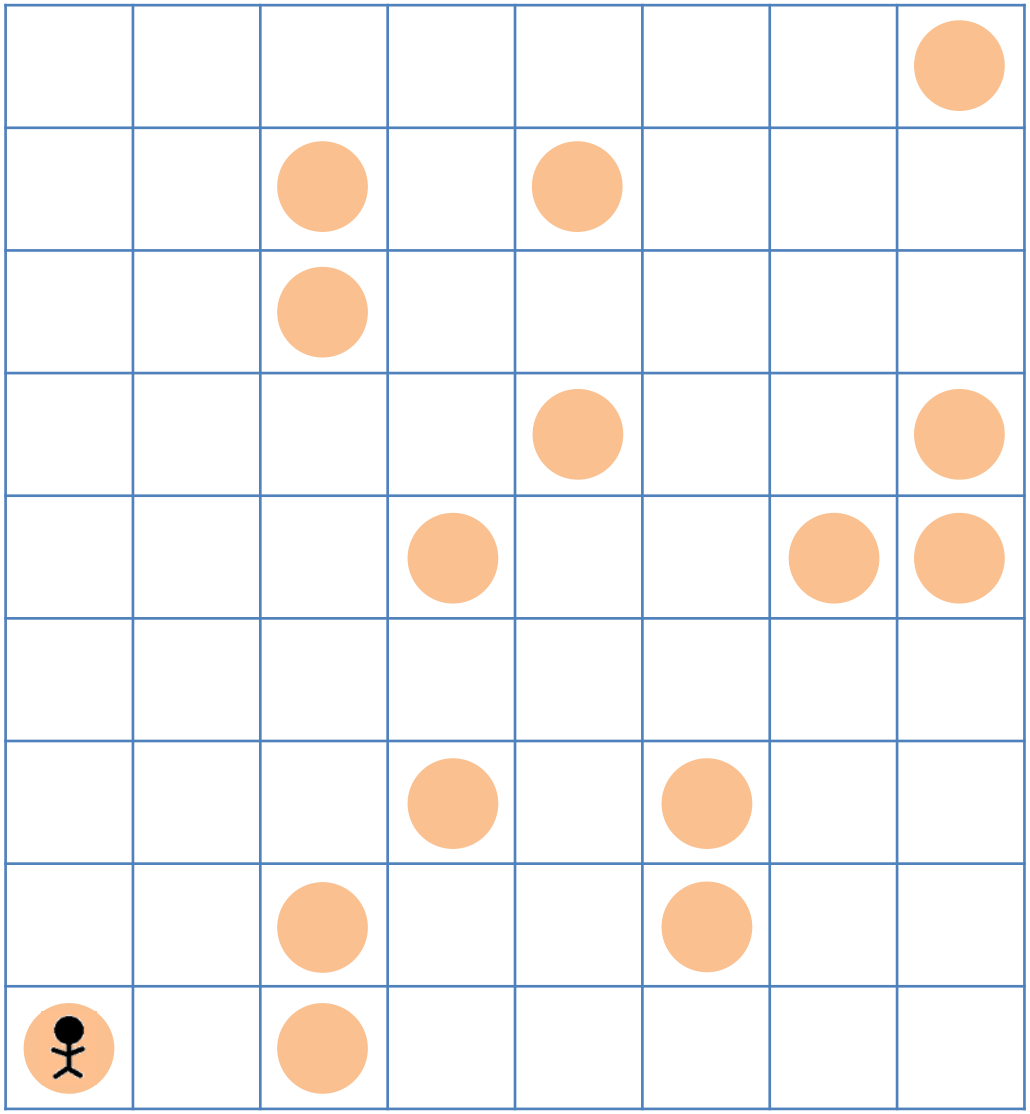
JOI 2013 本選 問題 3

解説： 保坂 和宏

問題概要

- 横 M × 縦 N のマス目の屋敷
  - 縦横 1 マスの移動に 1 分
  - 最初は縦移動のみ可能
- スイッチ：縦移動と横移動が切り替わる
  - K 個のマスにある
  - 押すのに 1 分
- 左下から右上まで行くための最短時間
- $M, N \leqq 100{,}000$, $K \leqq 200{,}000$

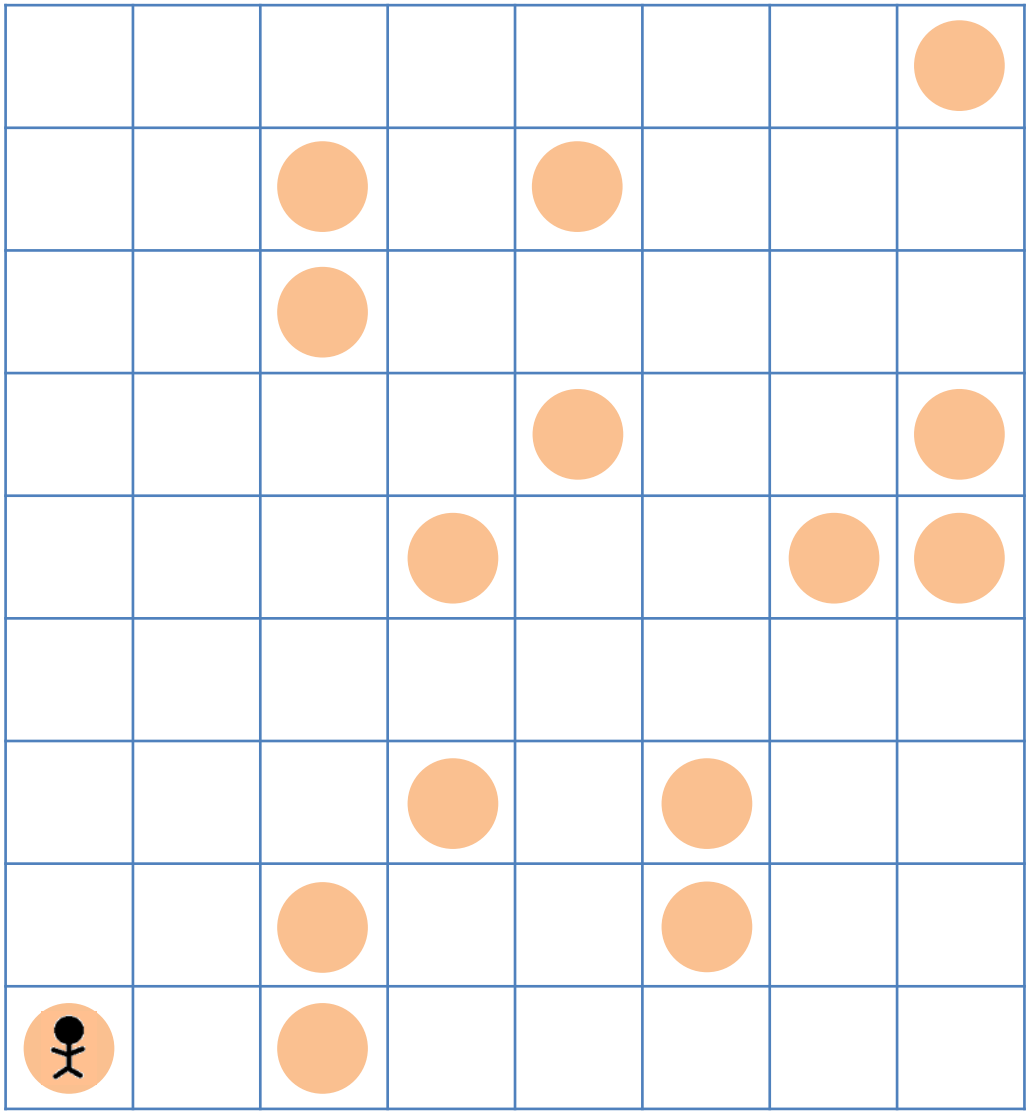
問題概要



移動：縦

時間： 0

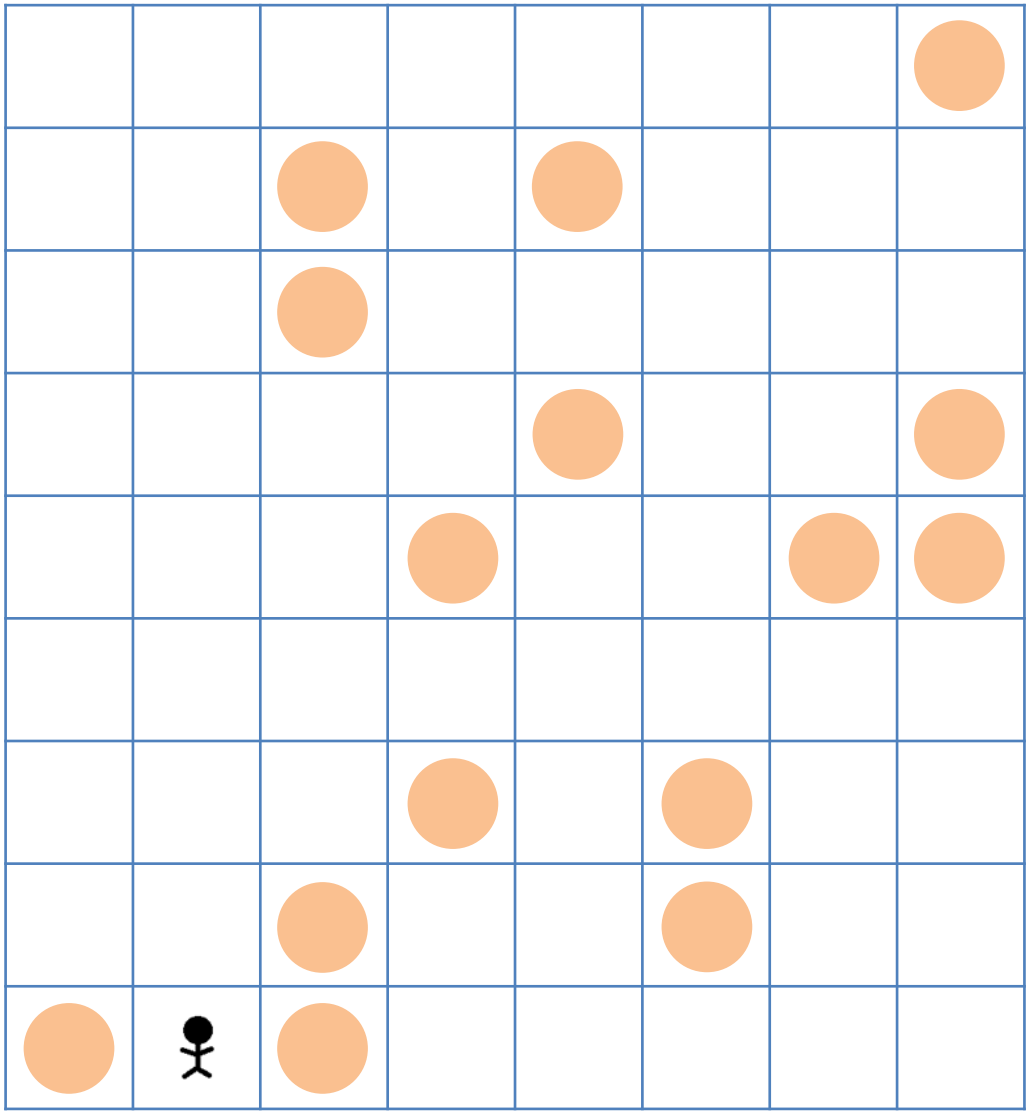
問題概要



移動：横

時間： 1

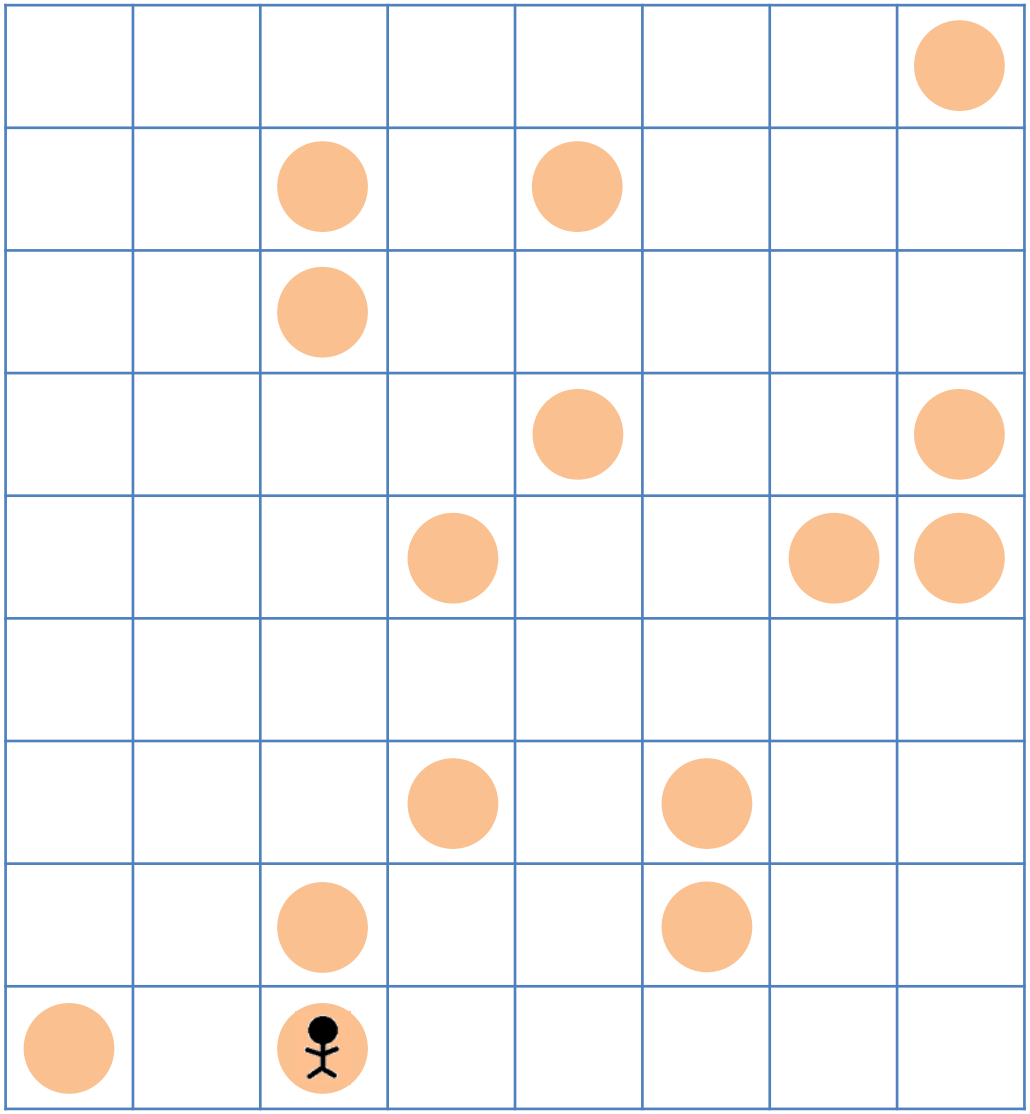
問題概要



移動：横

時間： 2

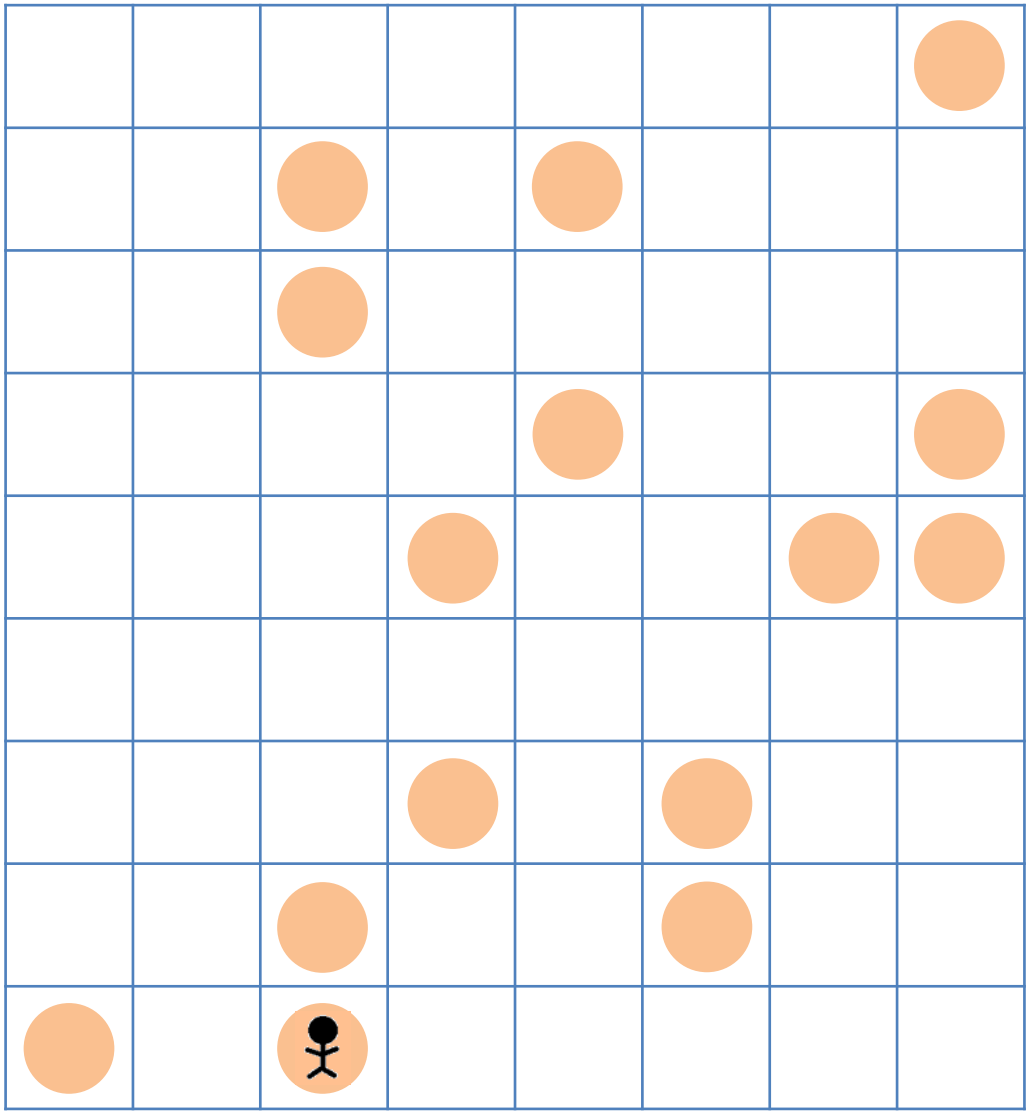
問題概要



移動：横

時間： 3

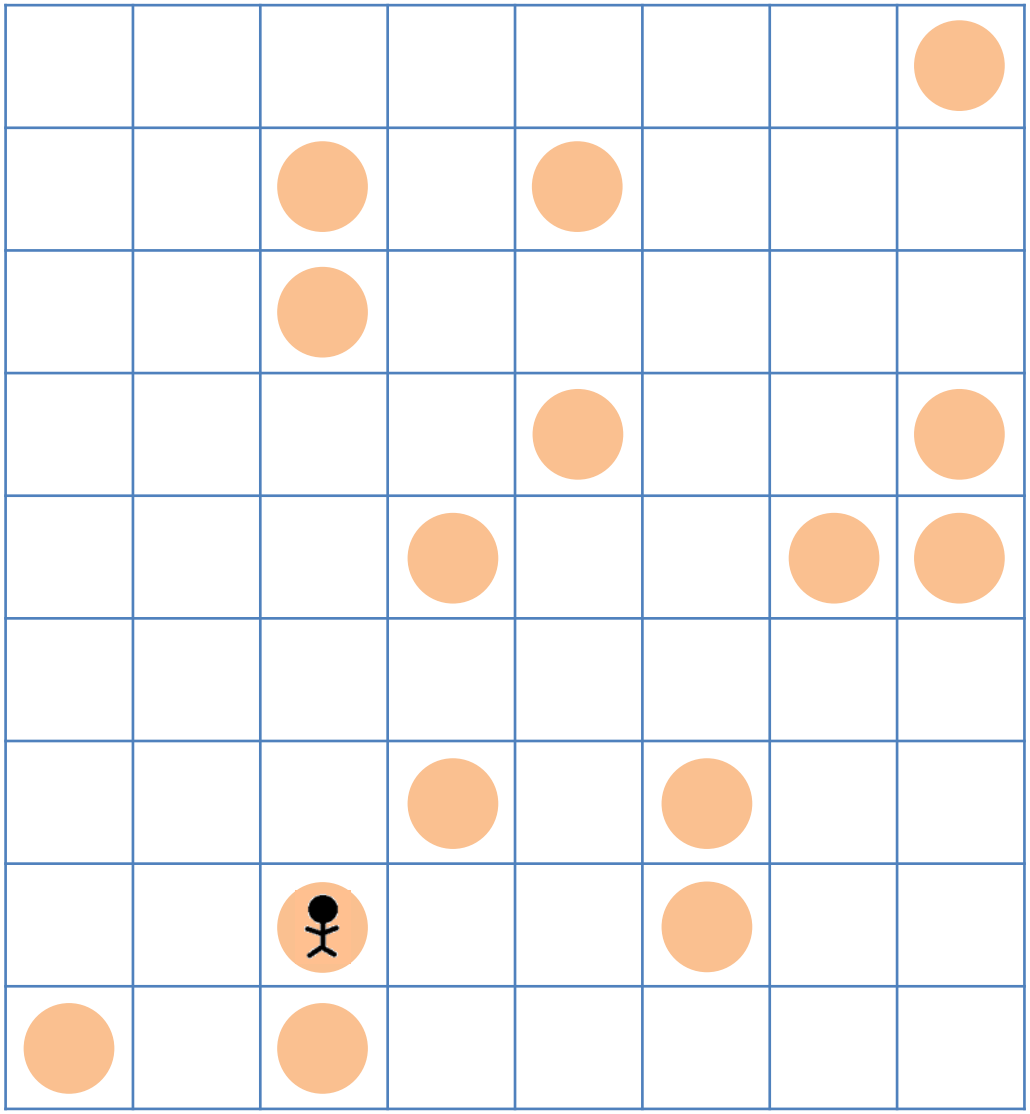
問題概要



移動：縦

時間： ４

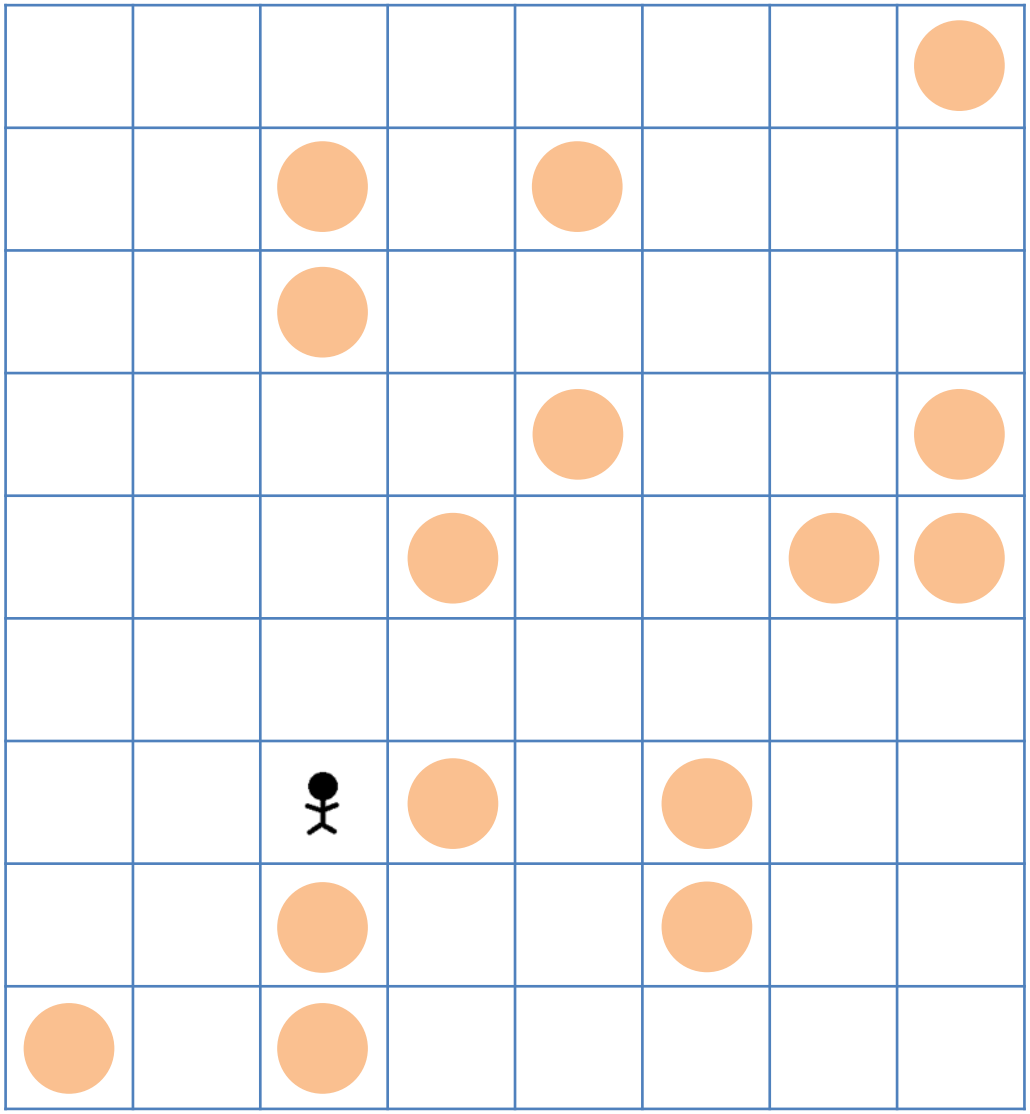
問題概要



移動：縦

時間： 5

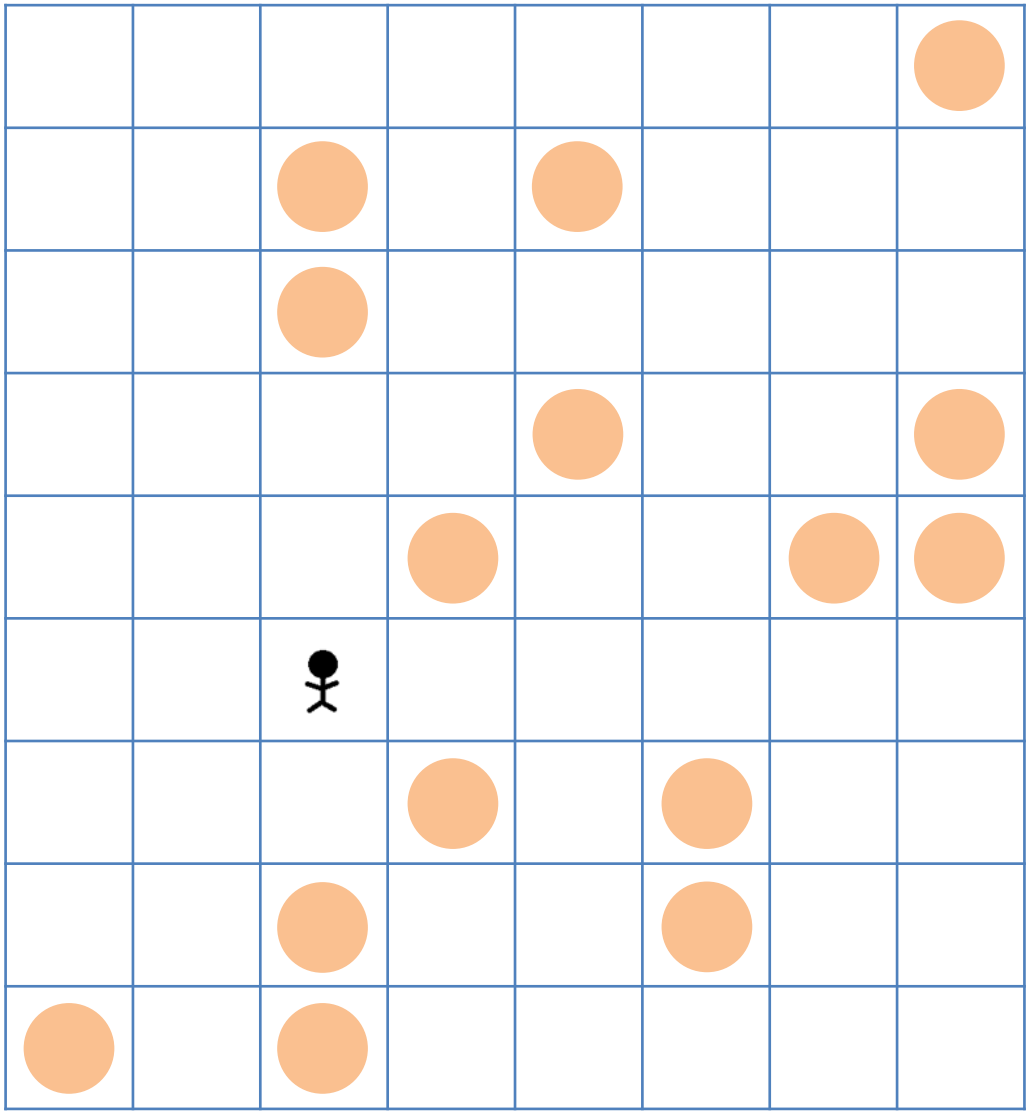
問題概要



移動：縦

時間： 6

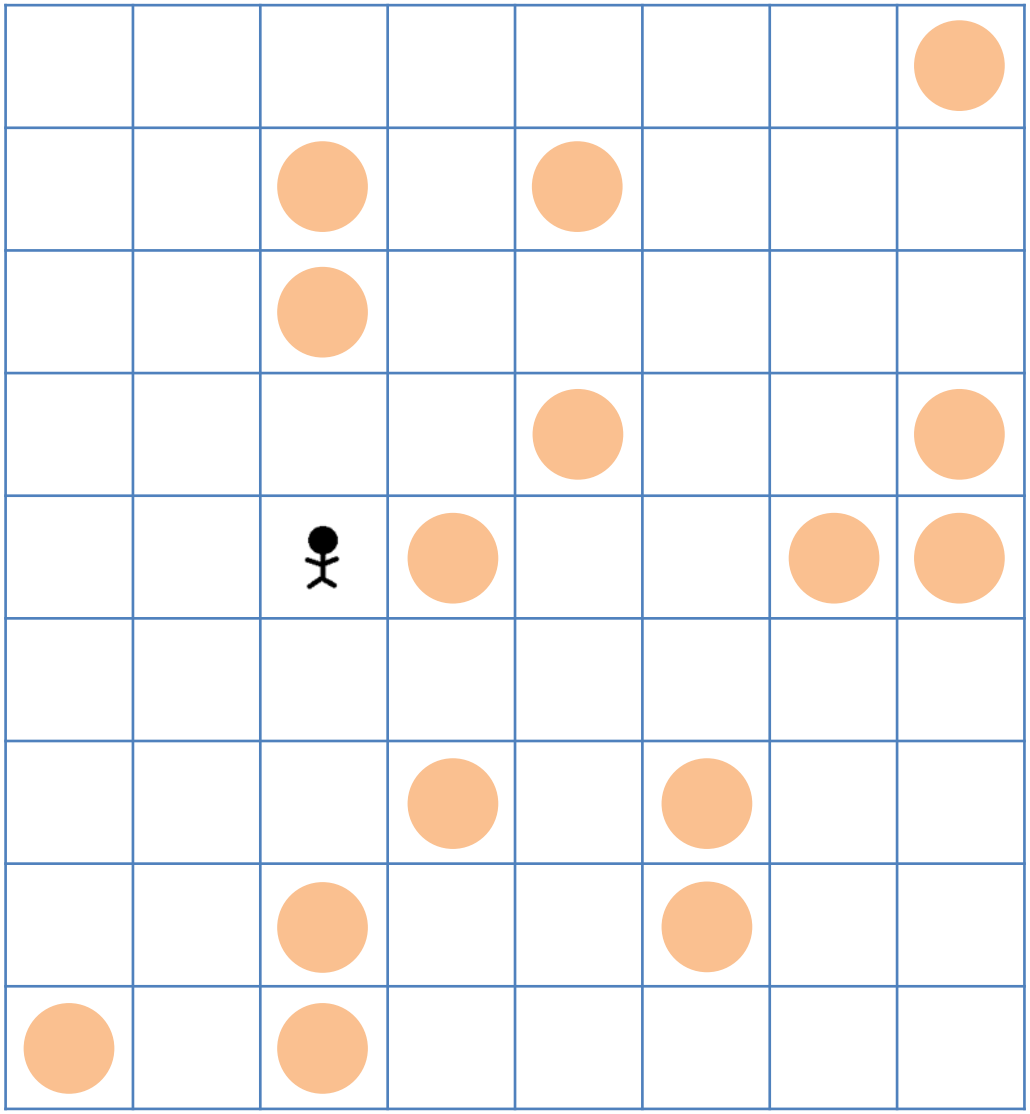
問題概要



移動：縦

時間： 7

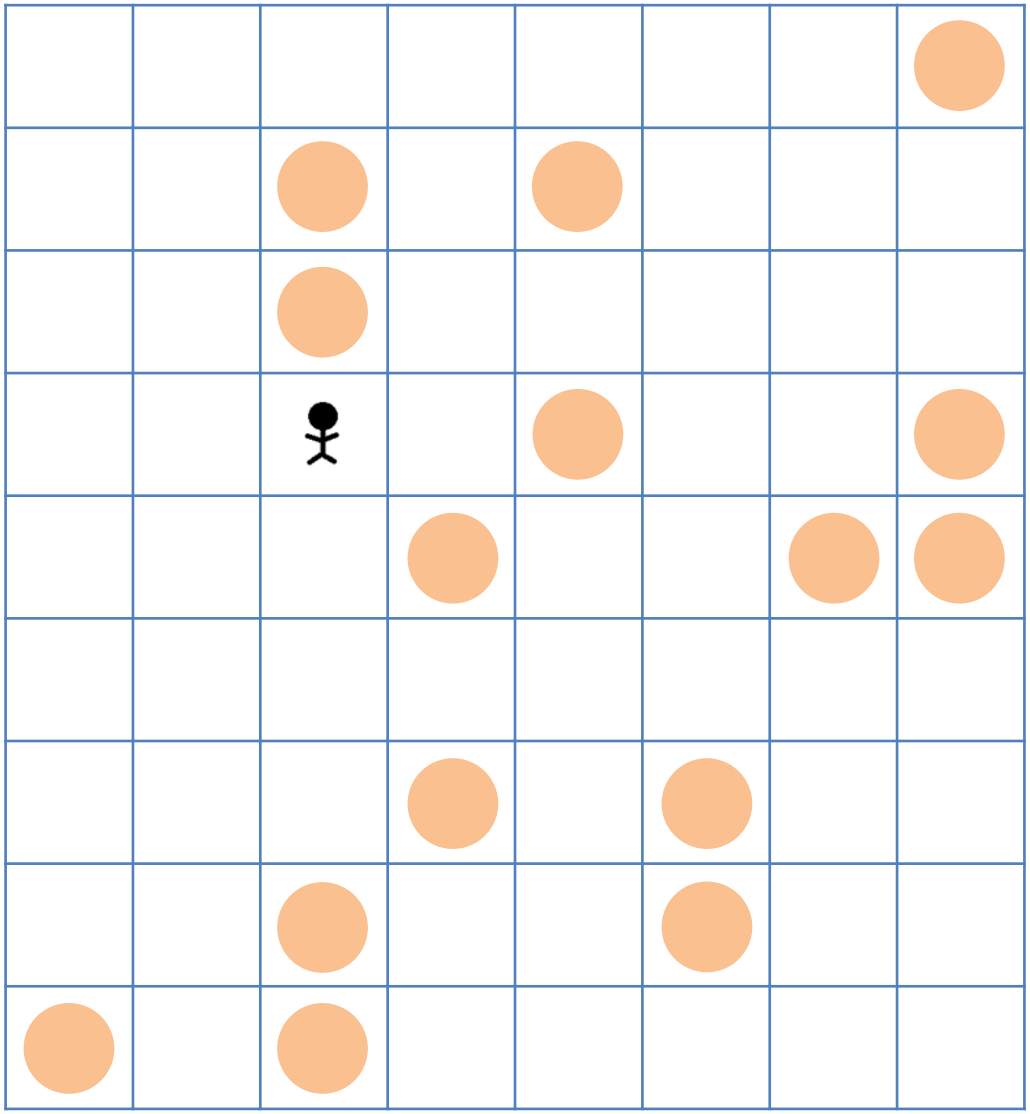
問題概要



移動：縦

時間： 8

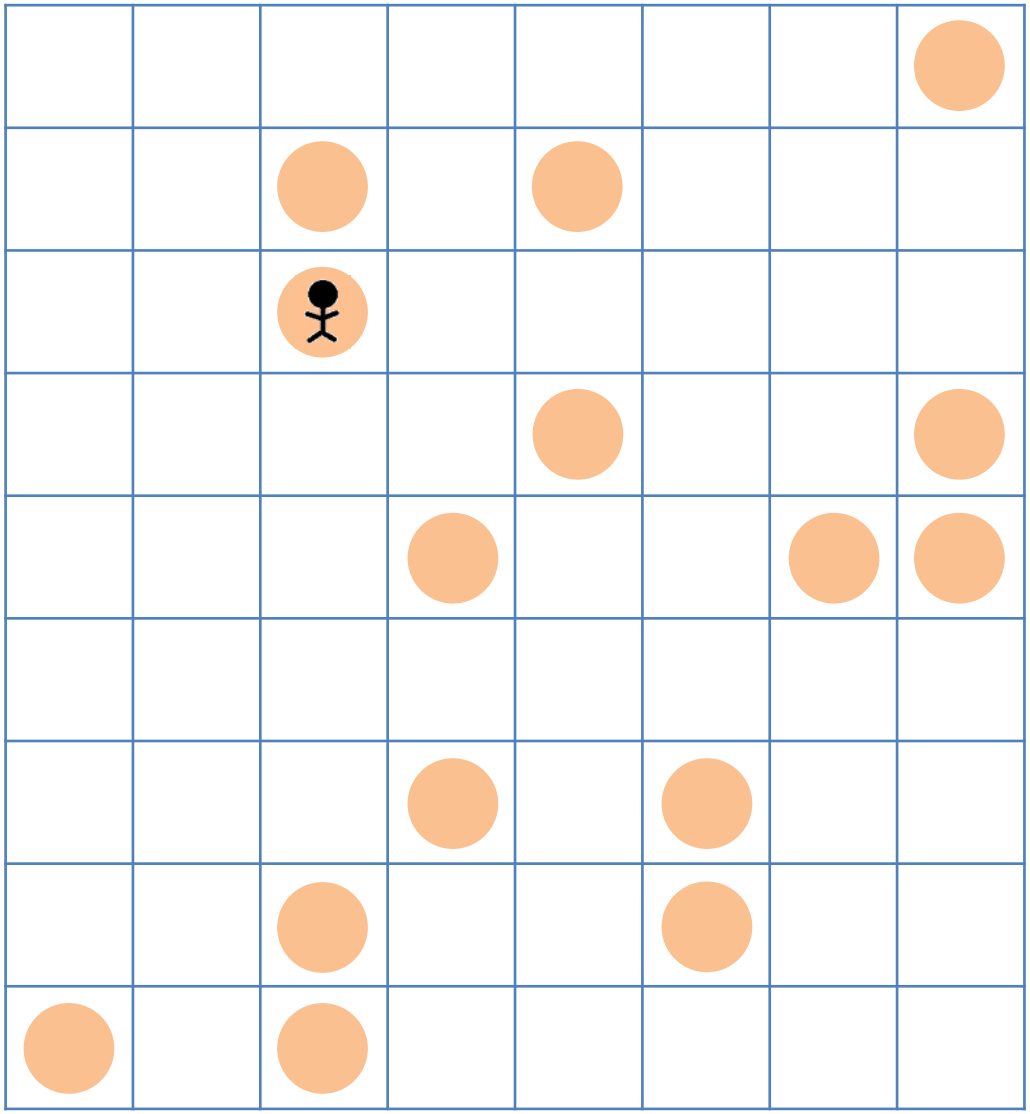
問題概要



移動：縦

時間： 9

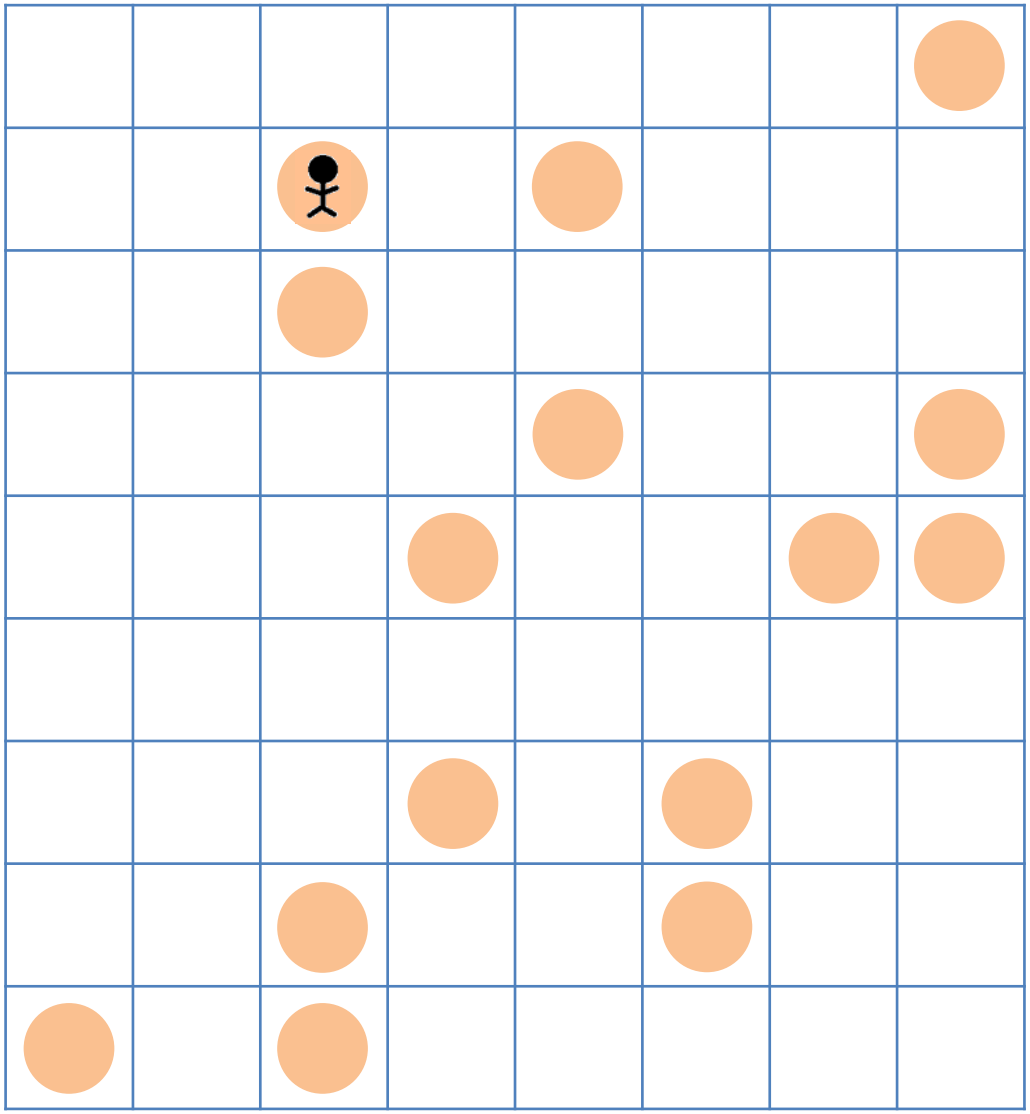
問題概要



移動：縦

時間：10

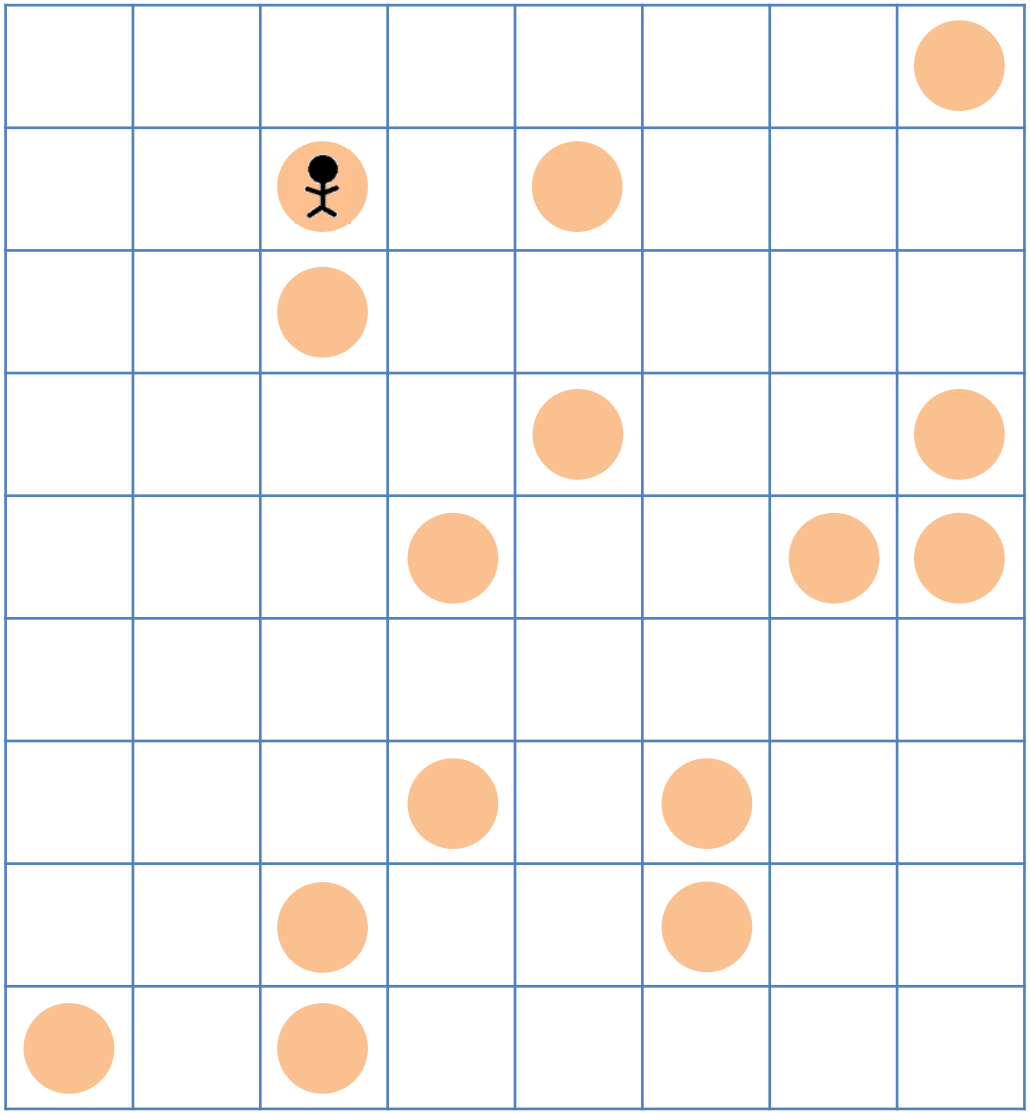
問題概要



移動：縦

時間：11

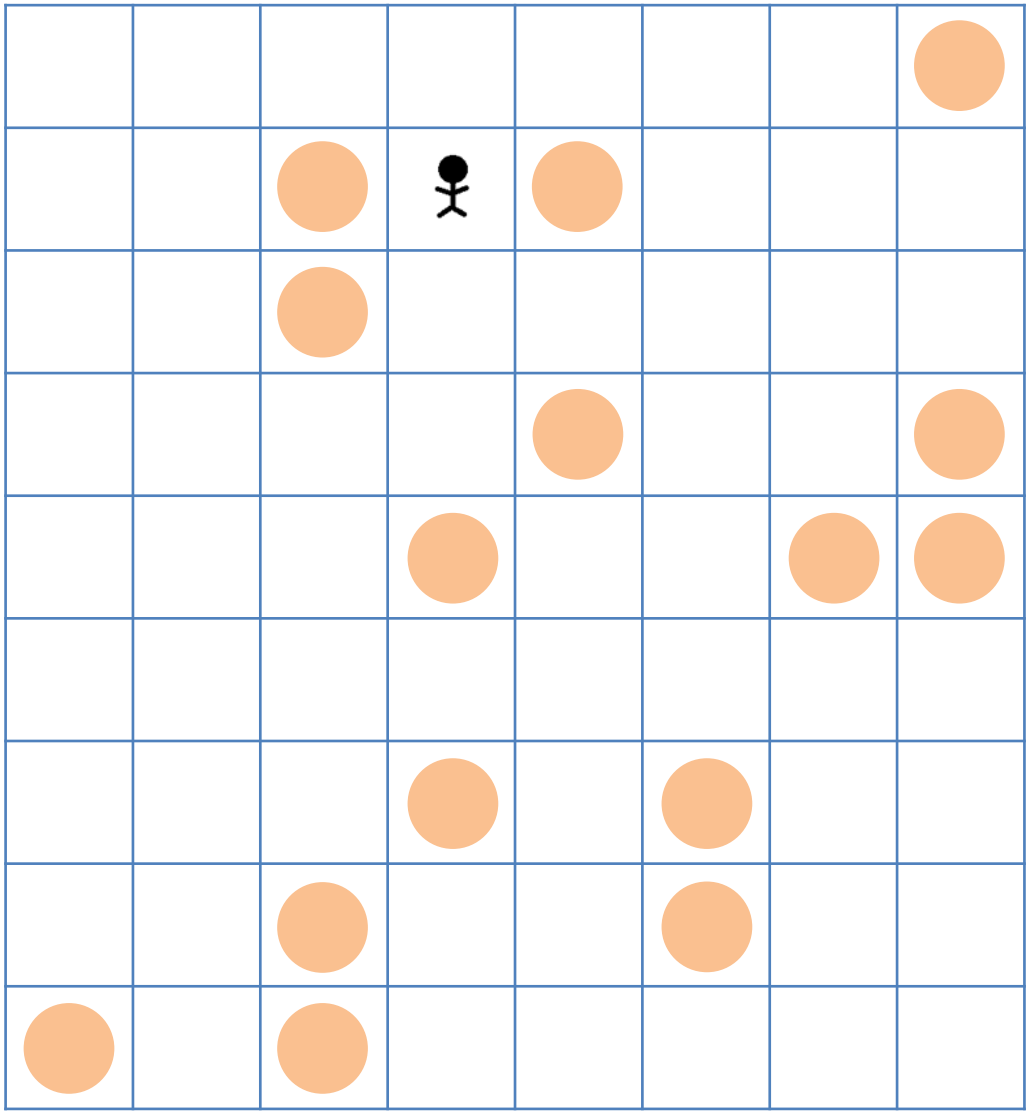
問題概要



移動：横

時間：12

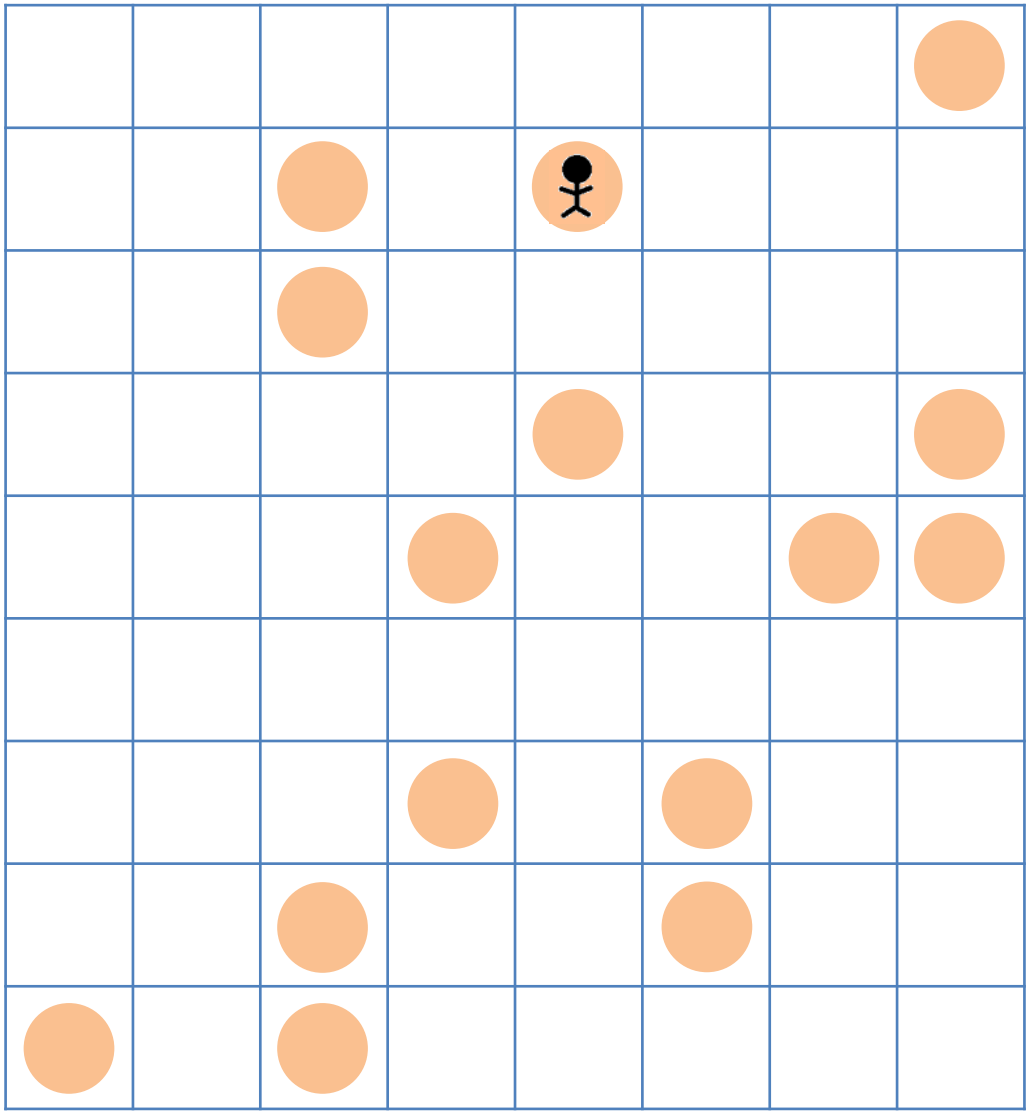
問題概要



移動：横

時間：13

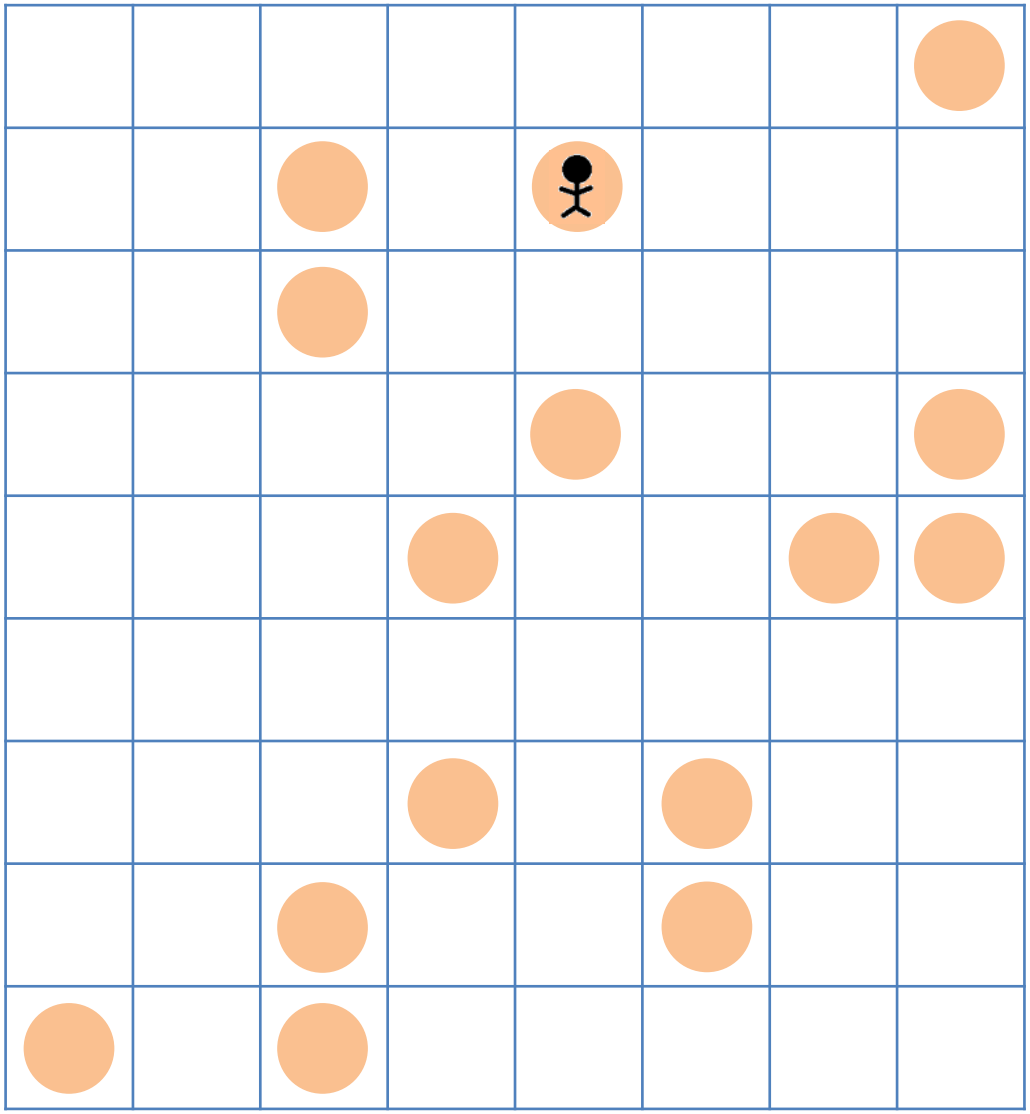
問題概要



移動：横

時間：14

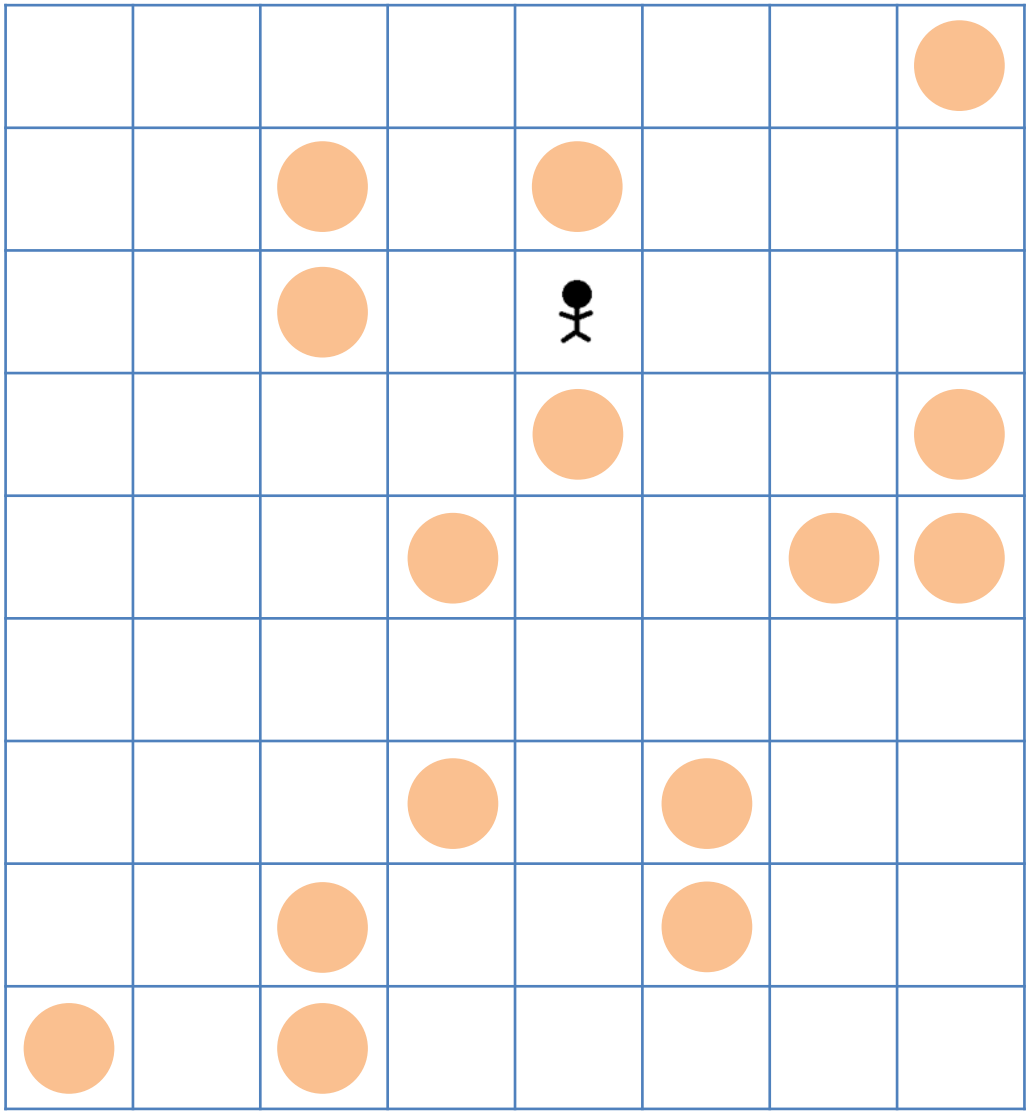
問題概要



移動：縦

時間：15

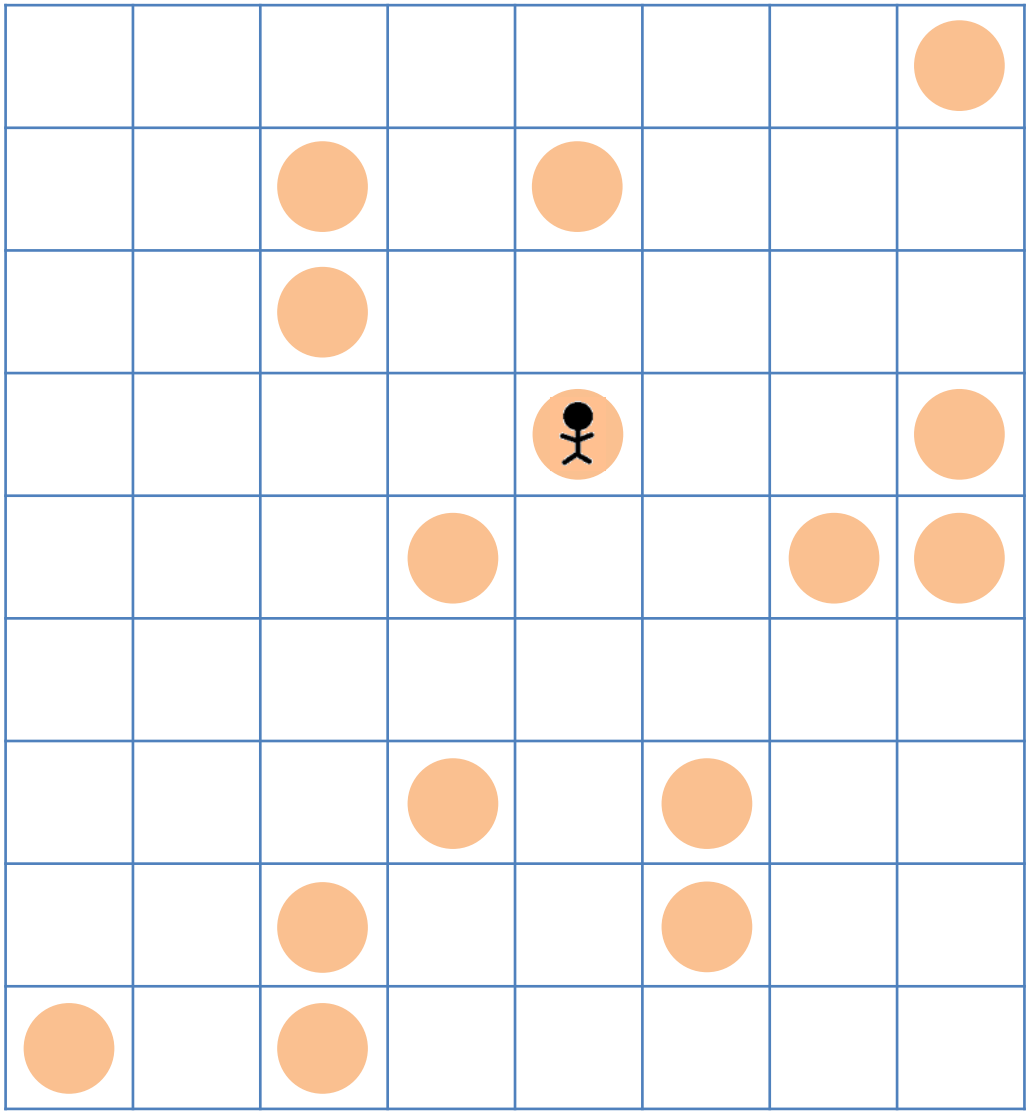
問題概要



移動：縦

時間：16

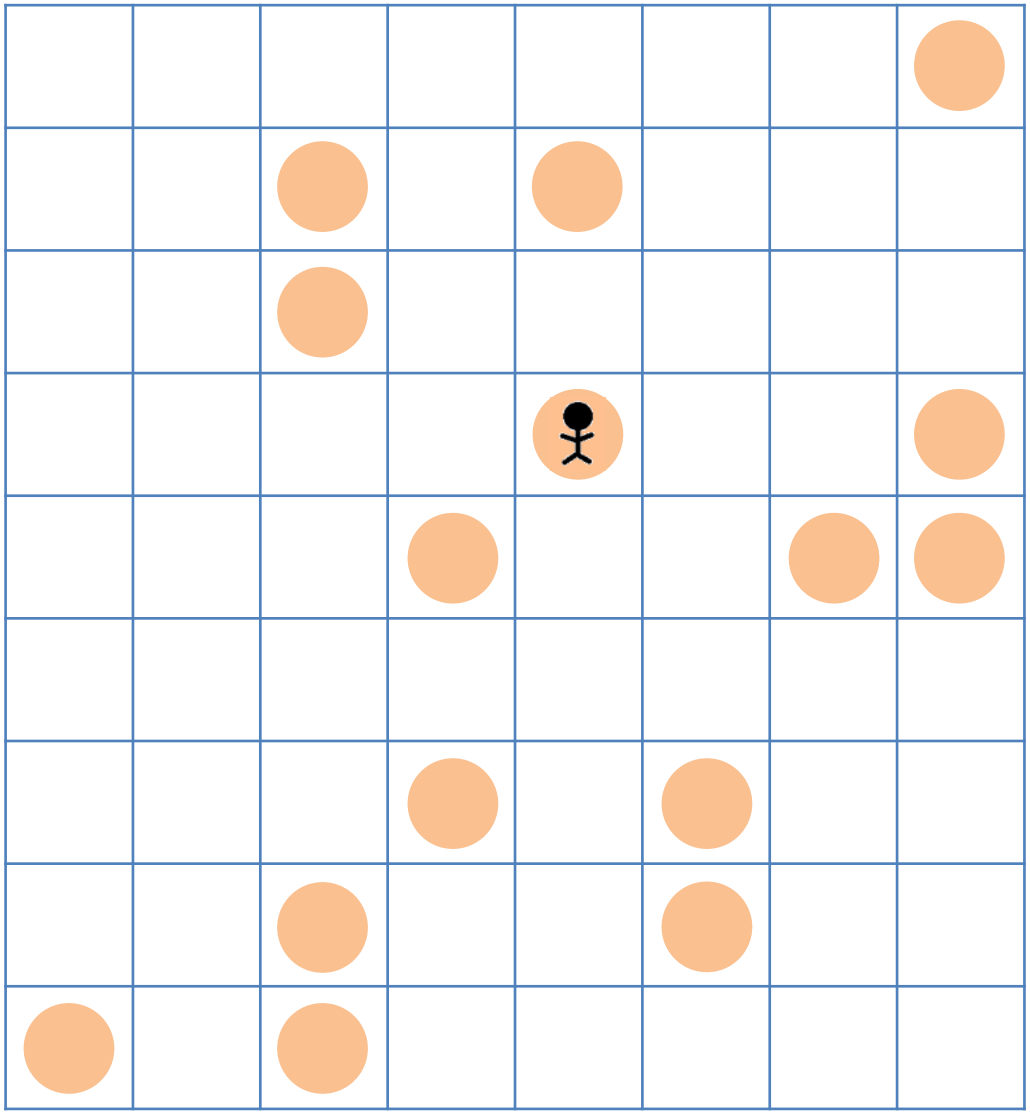
問題概要



移動：縦

時間：17

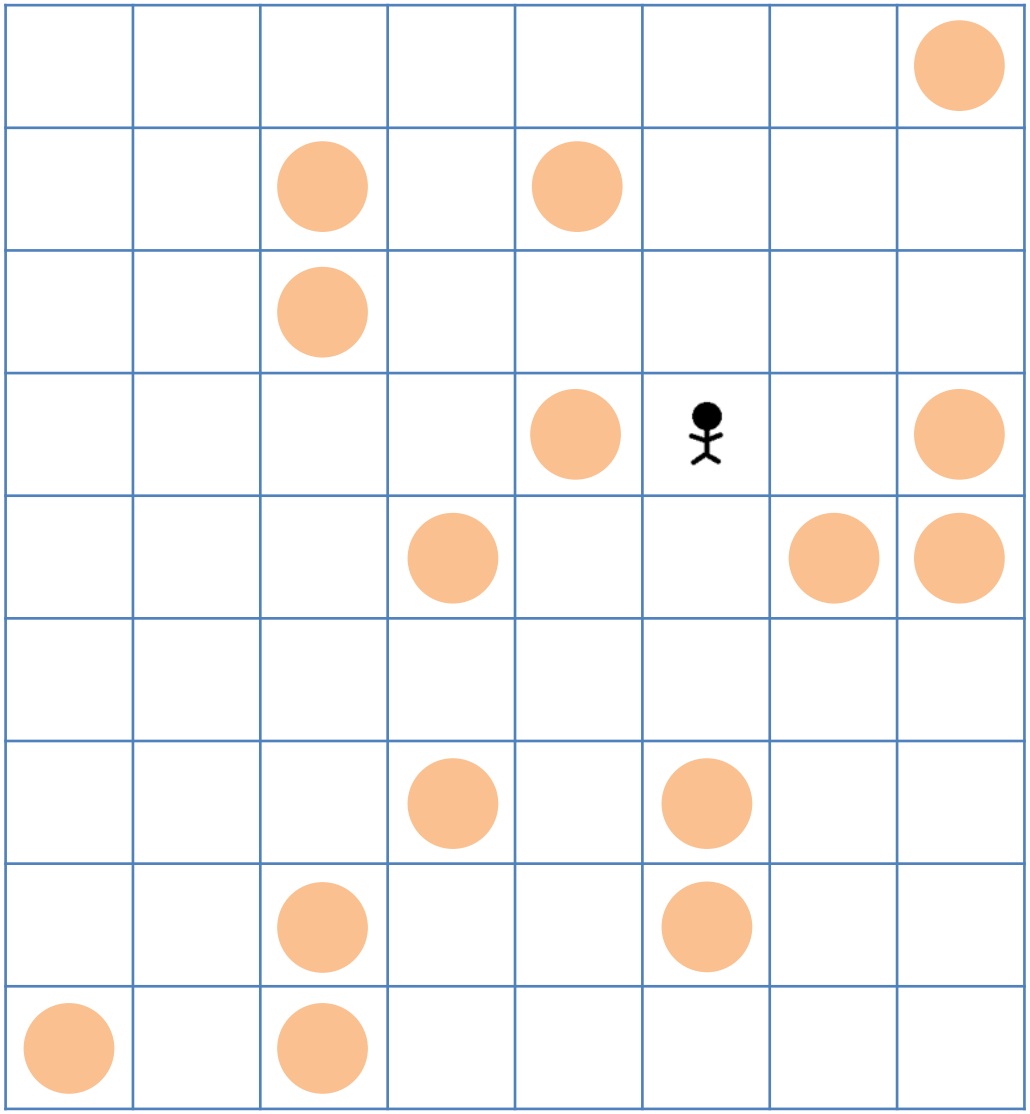
問題概要



移動：横

時間：18

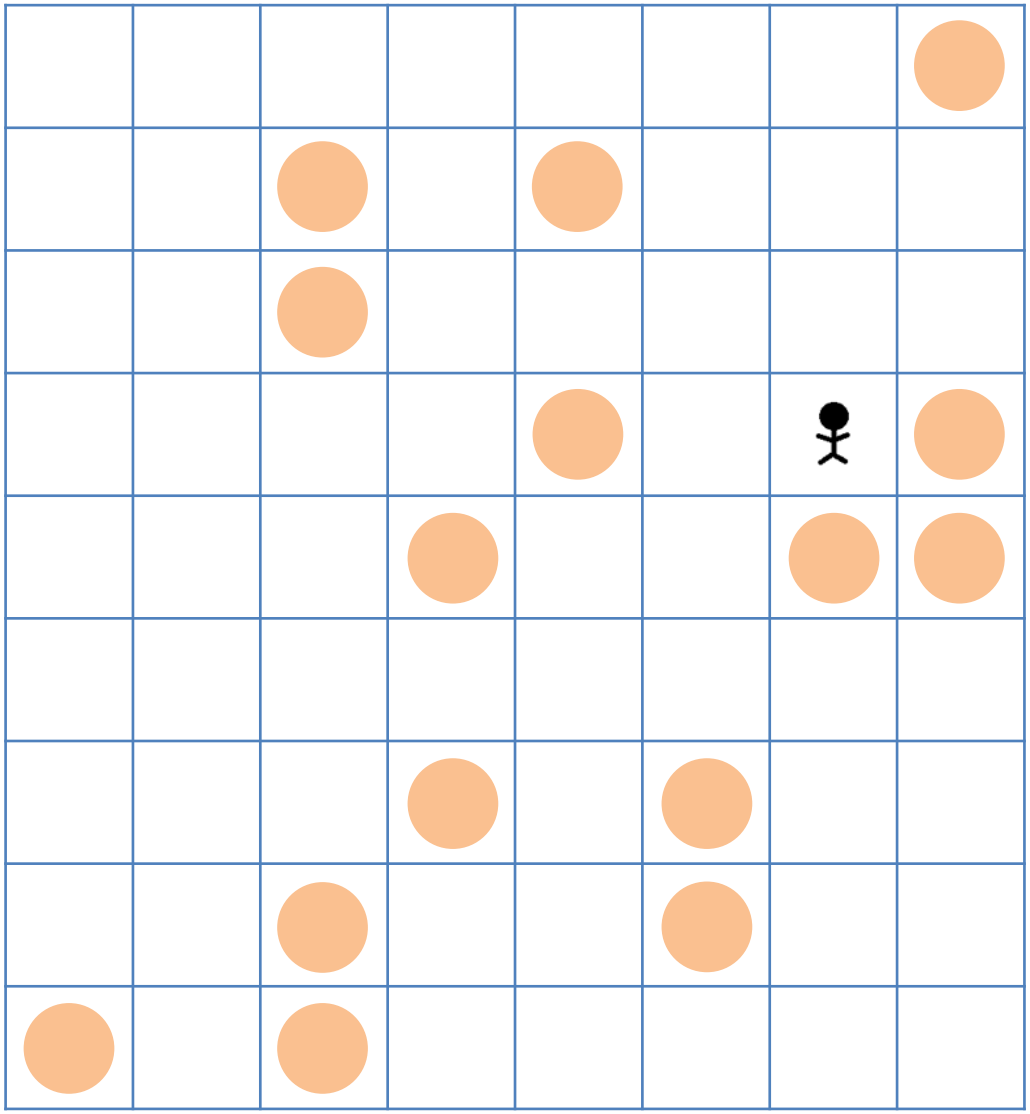
問題概要



移動：横

時間：19

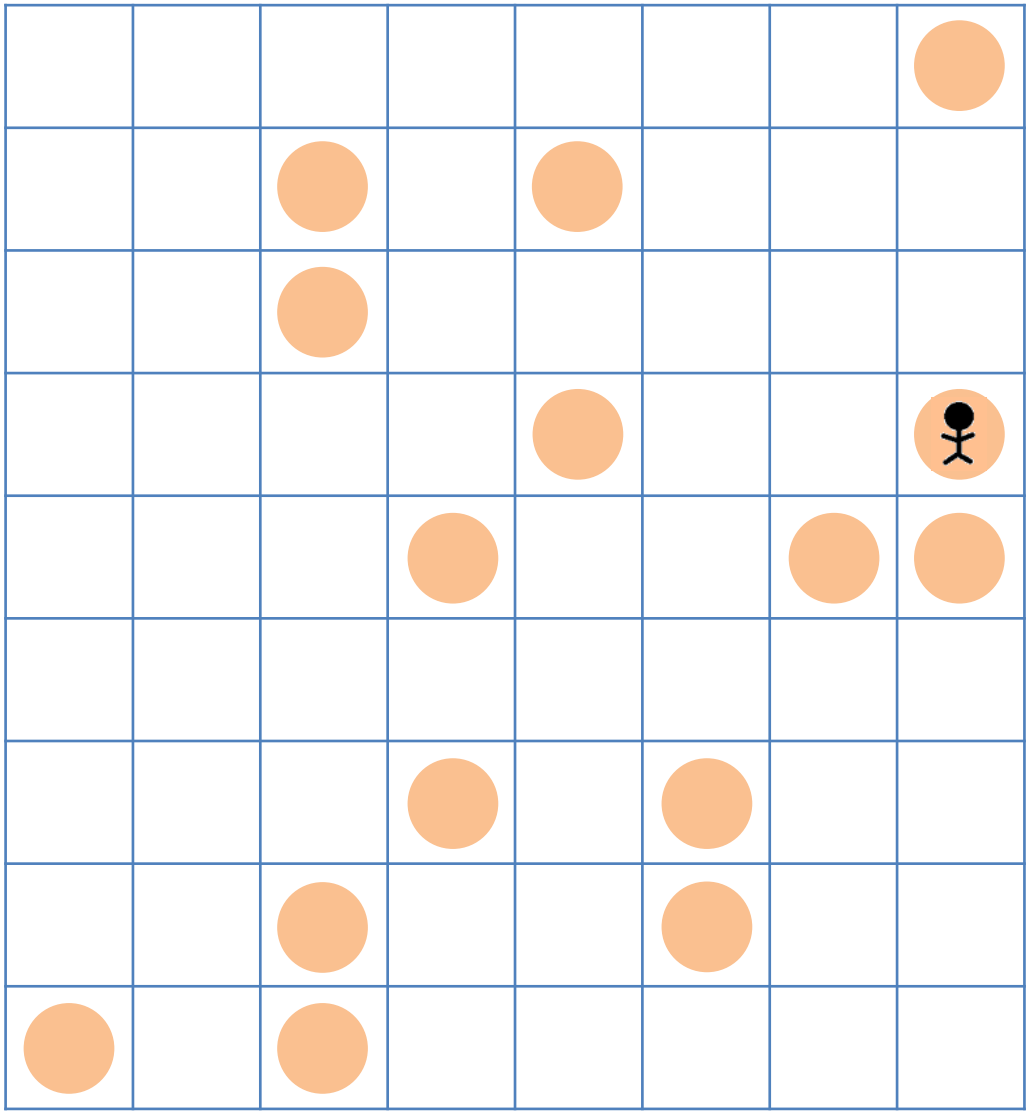
問題概要



移動：横

時間：20

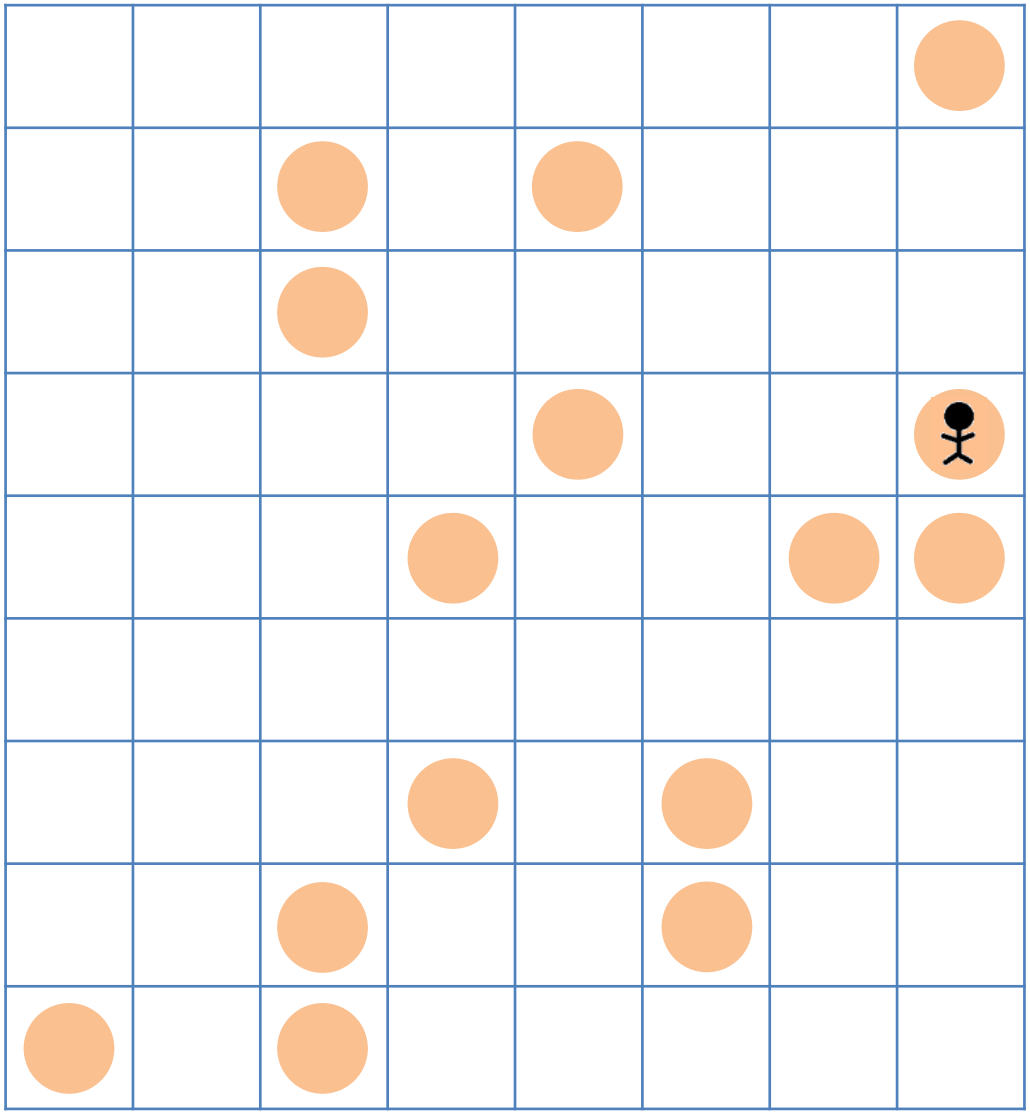
問題概要



移動：横

時間：21

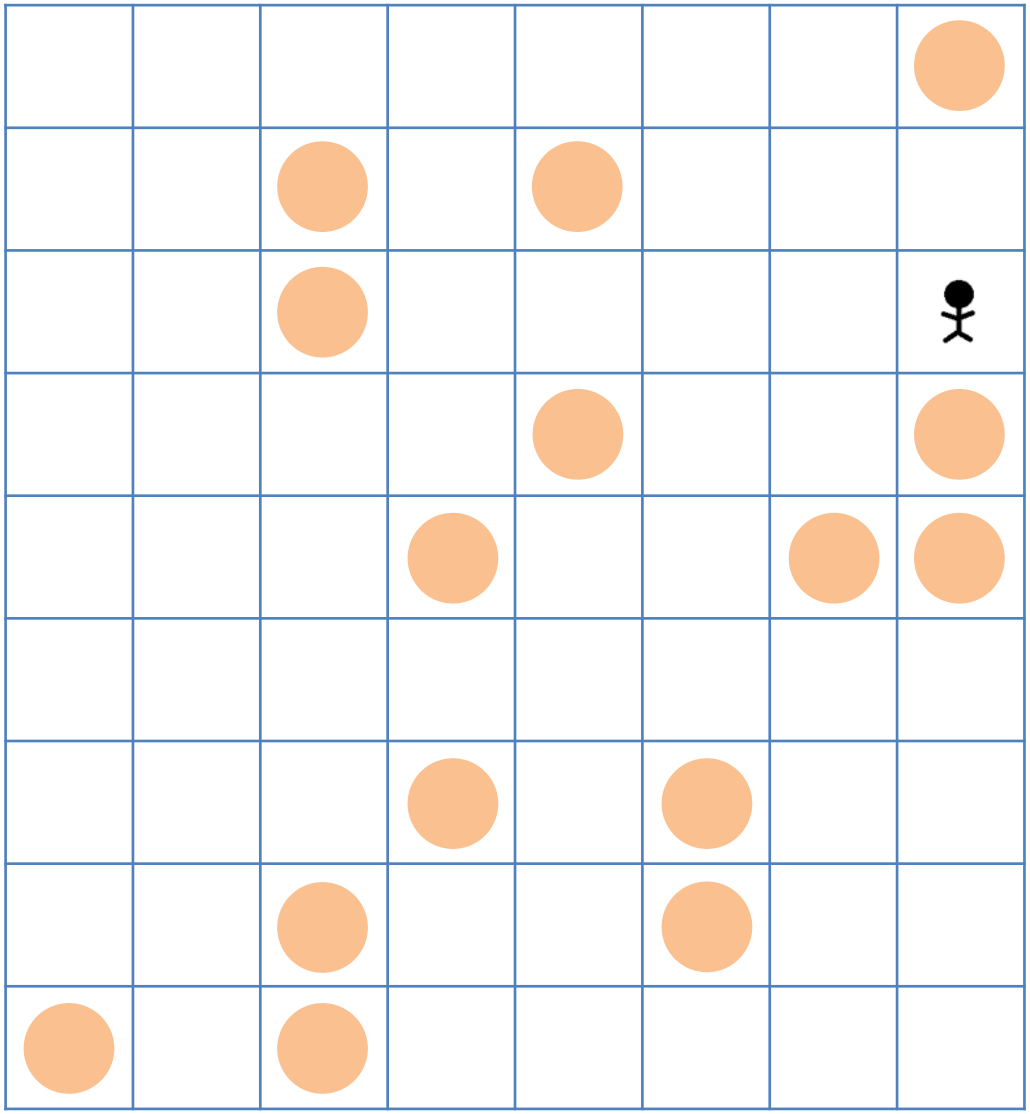
問題概要



移動：縦

時間：22

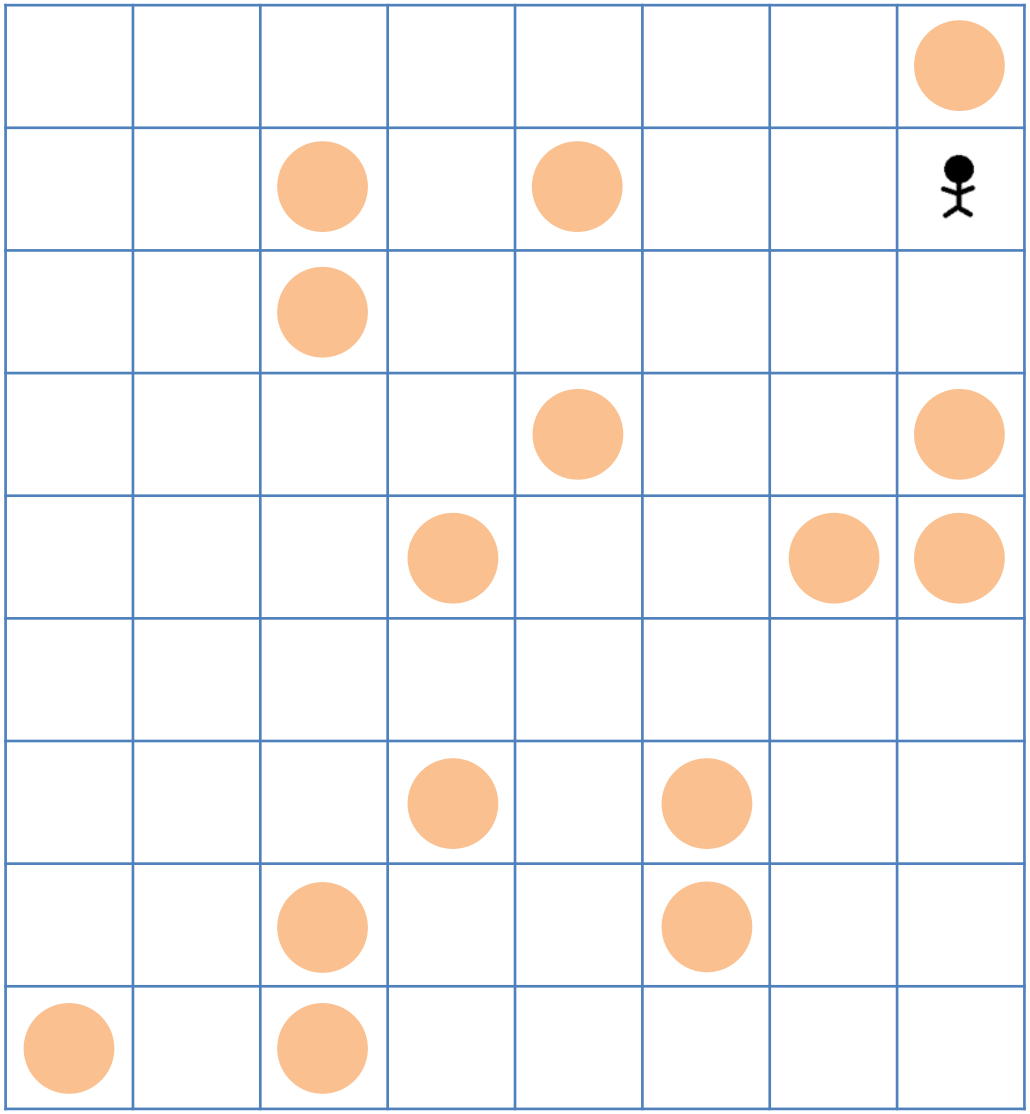
問題概要



移動：縦

時間：23

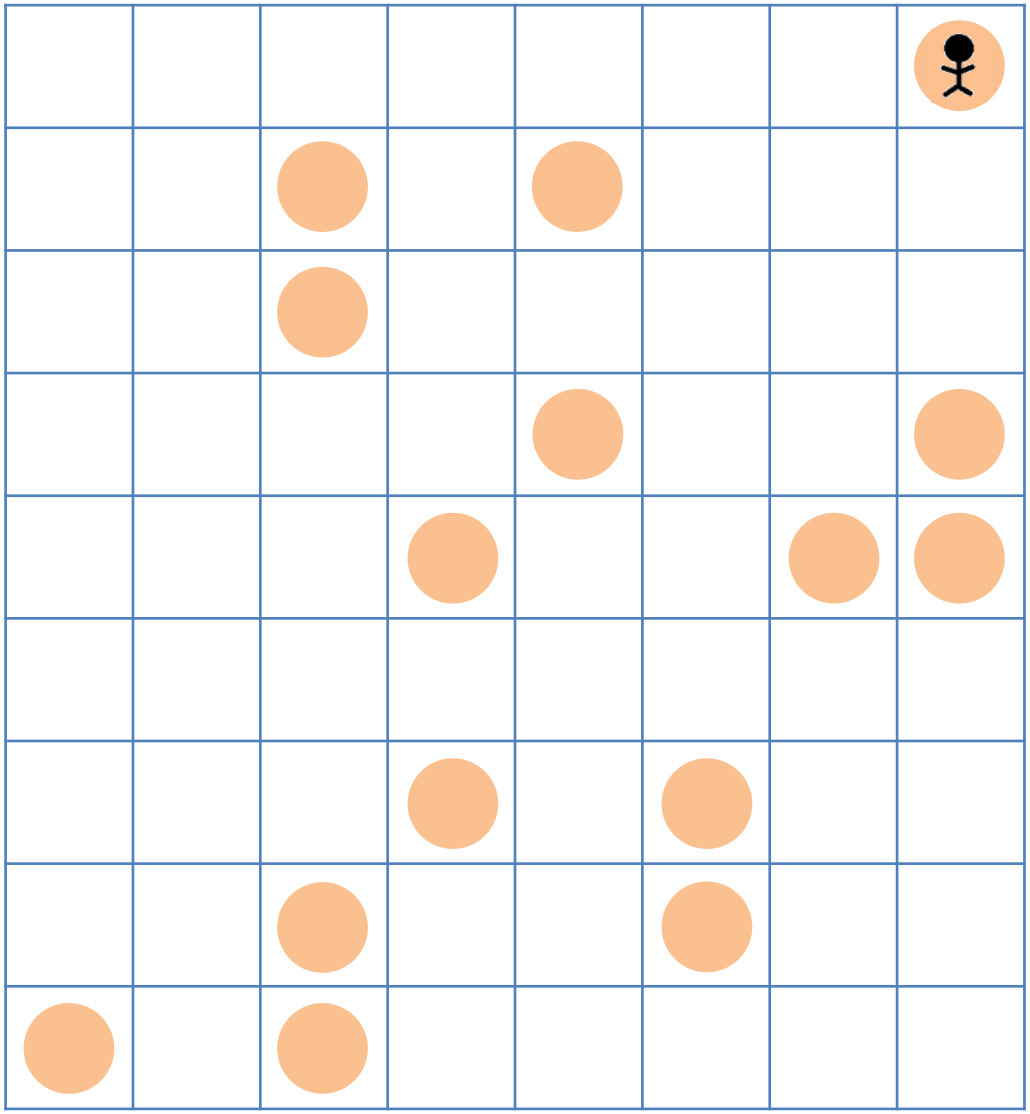
問題概要



移動：縦

時間：24

問題概要



移動：縦

時間：25

部分点解法 (1)

- 20 点： $M, N \leqq 1{,}000$

- (今いるマス, 移動できる方向) の組を「状態」として **幅優先探索 (BFS)**
- 状態の遷移
  - 移動できる方向に 1 マス移動
  - スイッチがあるならば押す
- 計算量： $O(MN)$

部分点解法 (1)

- 幅優先探索
  - (スタート地点, 縦移動) には時間 0 で到達
  - 時間 1 で到達できる状態，時間 2 で到達できる状態，……を順に求める
  - キューを利用
  - 問題例
    - JOI 2010-2011 予選 問題 5 「チーズ」

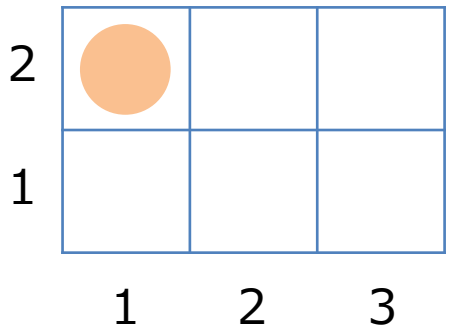
部分点解法 (2)

- 30 点： $K \leqq 2{,}000$

- スイッチのあるマス (とスタート・ゴール) のみに着目したい
  - 他のマスは縦か横に素通りするだけ

部分点解法 (2)

- スイッチ・スタート・ゴールのみに着目



(1, 1)
縦移動

(1, 1)
横移動

(1, 2)
縦移動

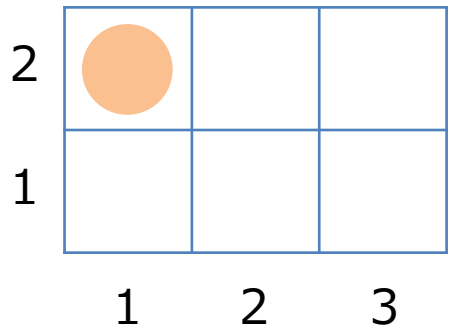
(1, 2)
横移動

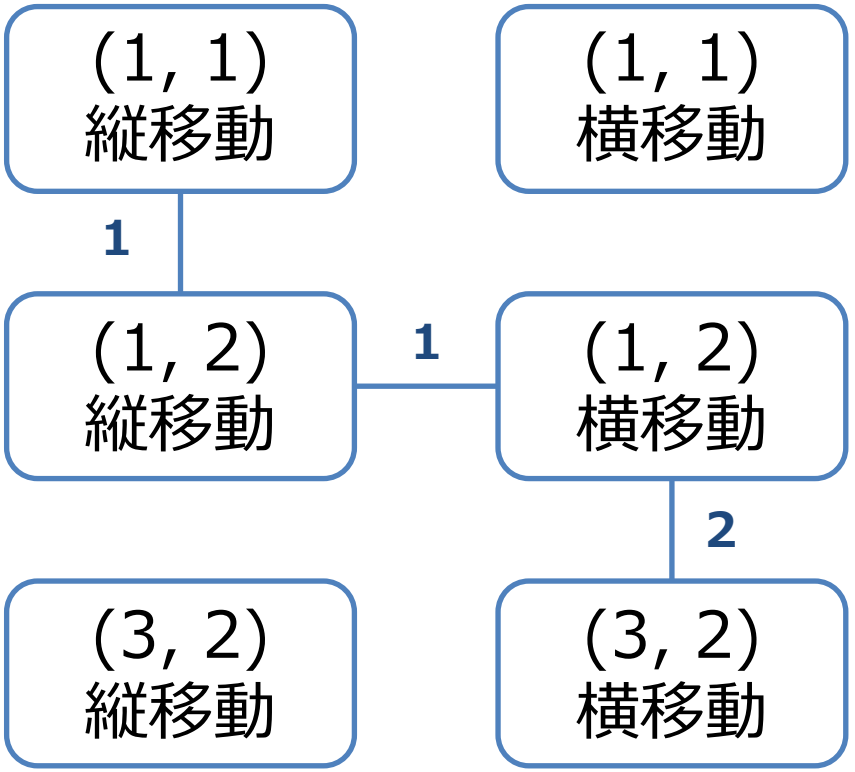
(3, 2)
縦移動

(3, 2)
横移動

部分点解法 (2)

- スイッチ・スタート・ゴールのみに着目





部分点解法 (2)

- グラフの最短路を求める問題
  - 頂点： (スイッチ or スタート or ゴール のマス, 移動できる方向) の組
    - 始点： ((1, 1), 縦移動)
    - 終点： ((*M*, *N*), 縦移動), ((*M*, *N*), 横移動)
  - 辺：
    - 縦 or 横 に並んだ 2 マスの 縦移動 or 横移動 に対応する頂点を結ぶ辺 (コストは距離)
    - スイッチのあるマスに対して，縦移動に対応する頂点と横移動に対応する頂点を結ぶ辺 (コスト 1)

部分点解法 (2)

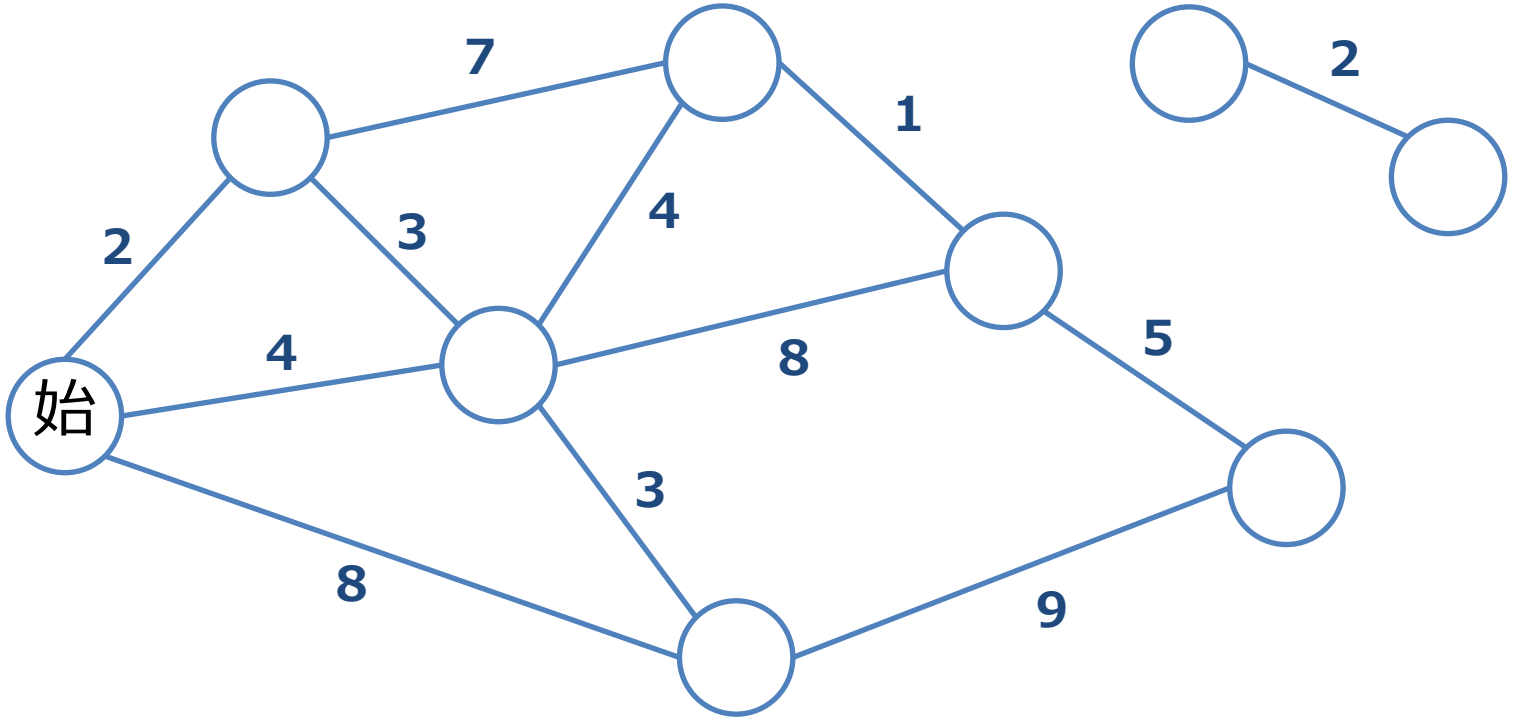
- グラフの最短路を求める問題

- **Dijkstra (ダイクストラ) 法**
  - コストが非負のグラフについて，1 つの始点から他の頂点への最短距離を求める

  - 問題例
    - JOI 2010-2011 本選 3 「JOI 国の買い物事情」

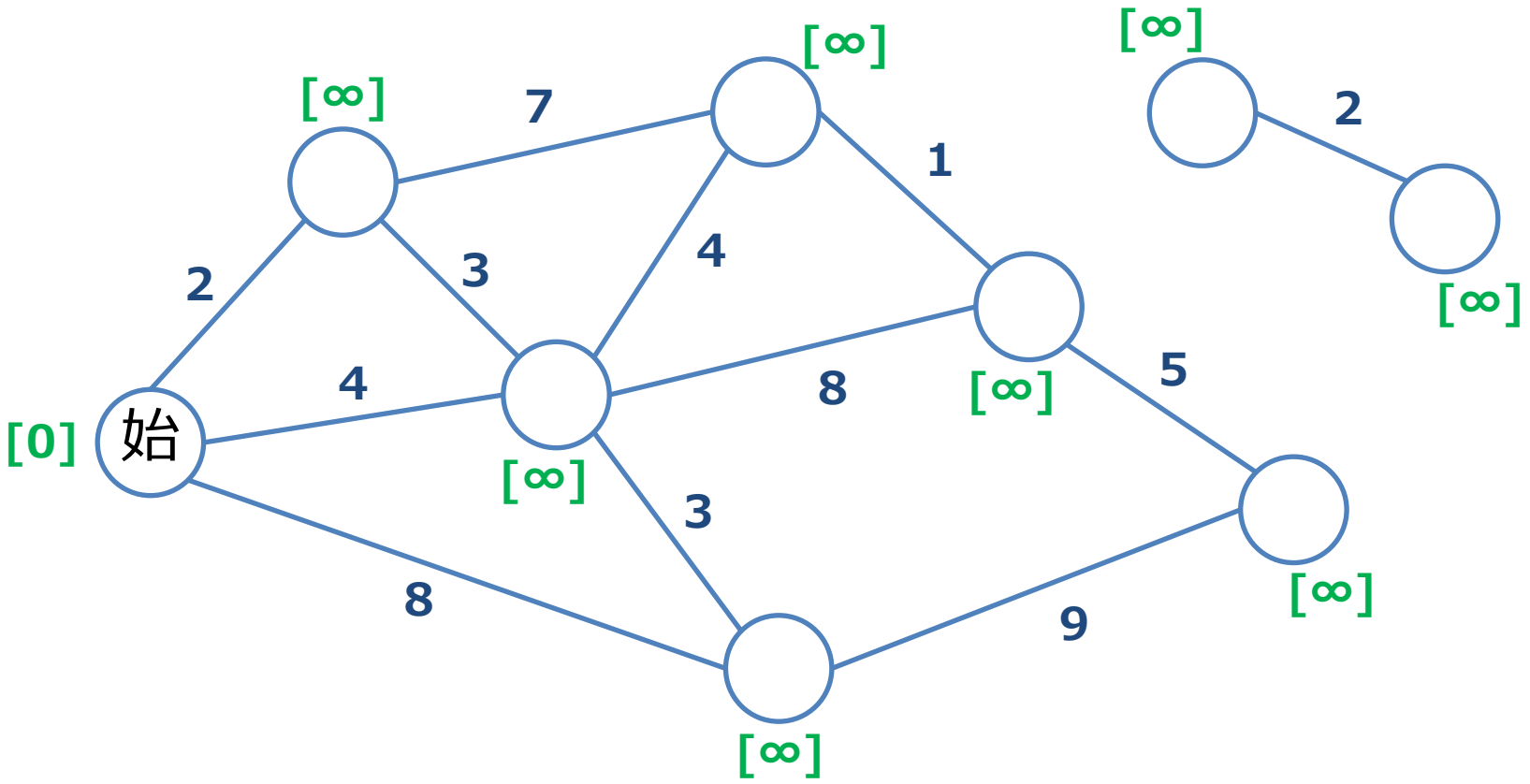
Dijkstra 法

- 方針：始点から近い順に確定させていく



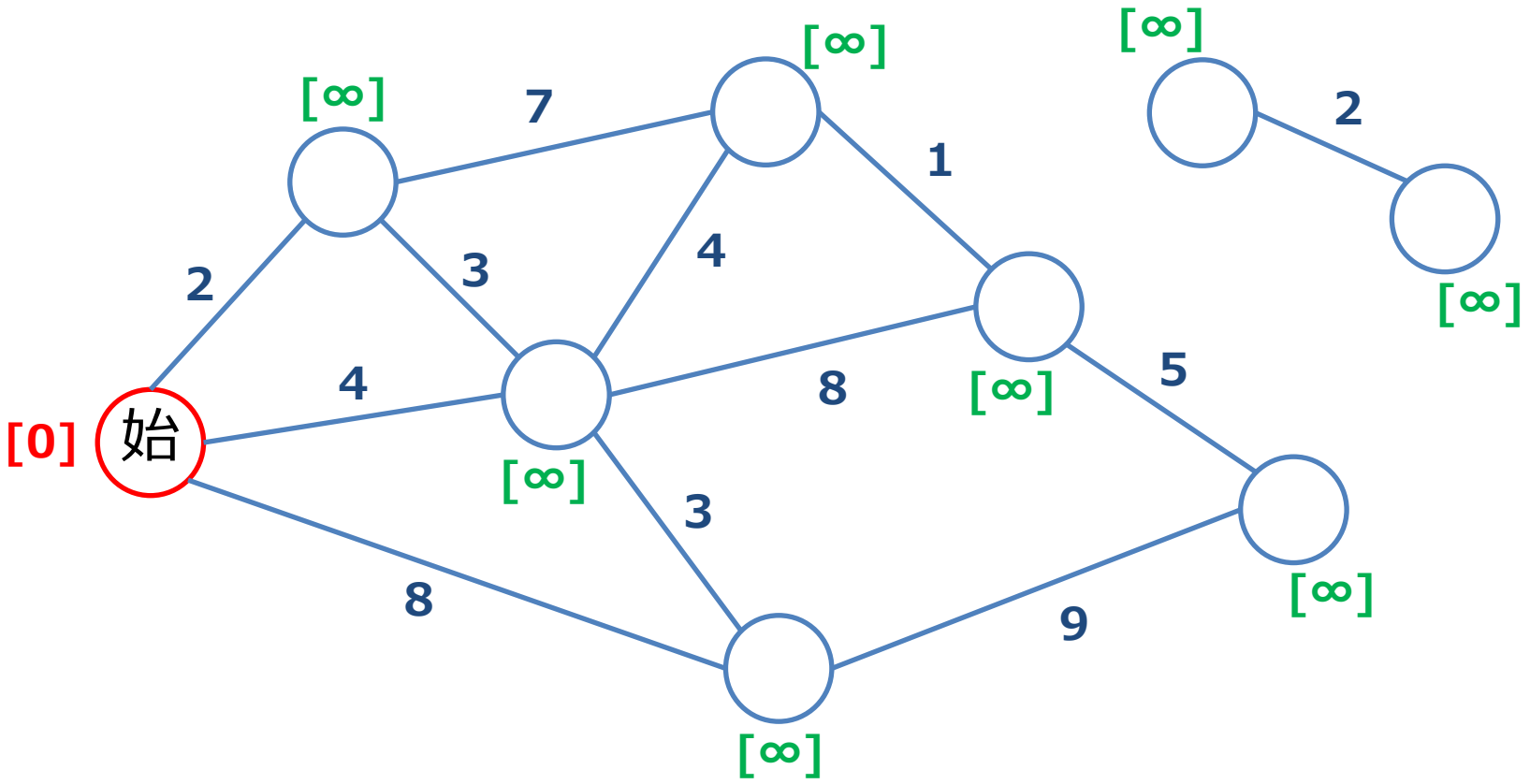
Dijkstra 法

- 方針：始点から近い順に確定させていく



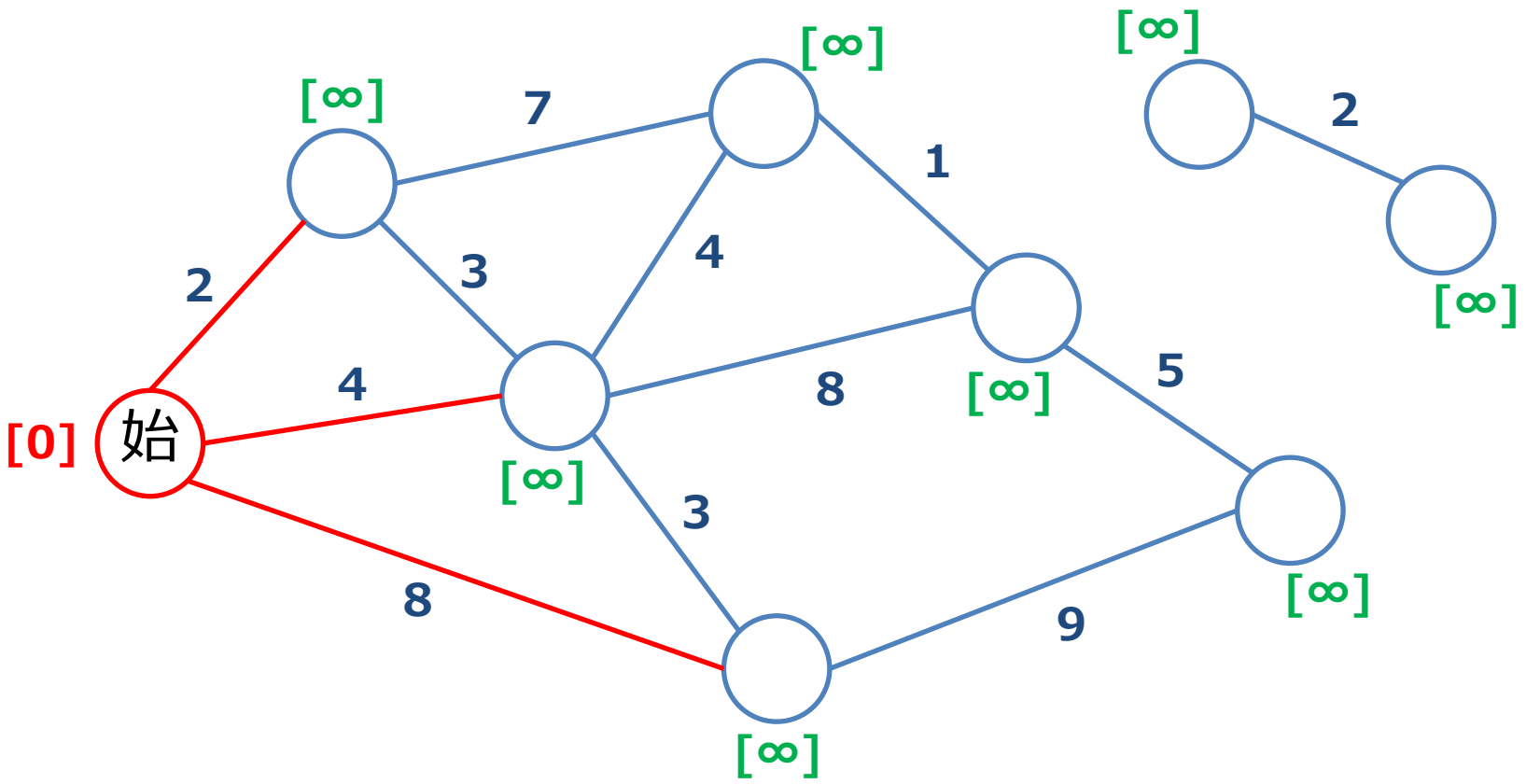
Dijkstra 法

- 方針：始点から近い順に確定させていく



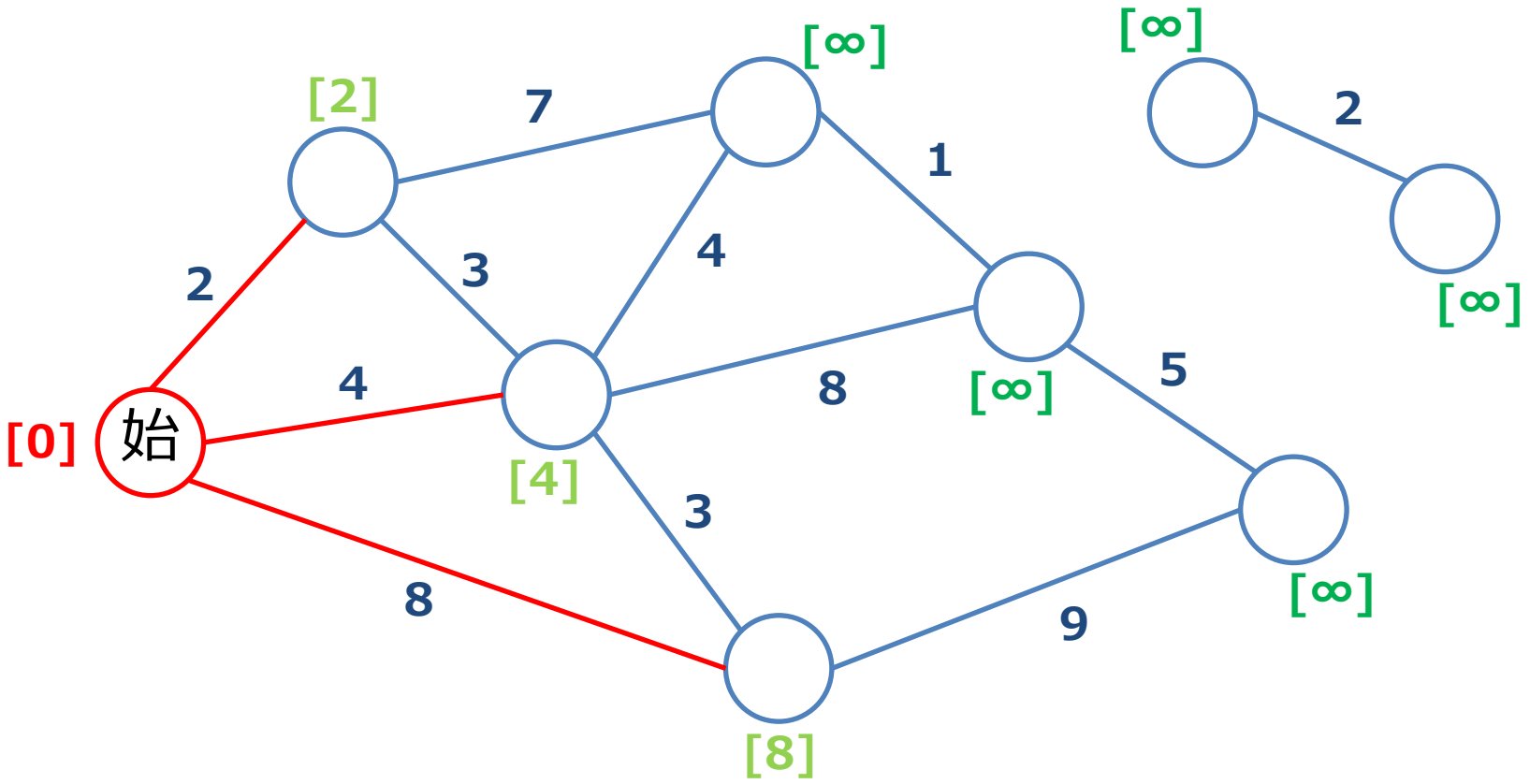
Dijkstra 法

- 方針：始点から近い順に確定させていく



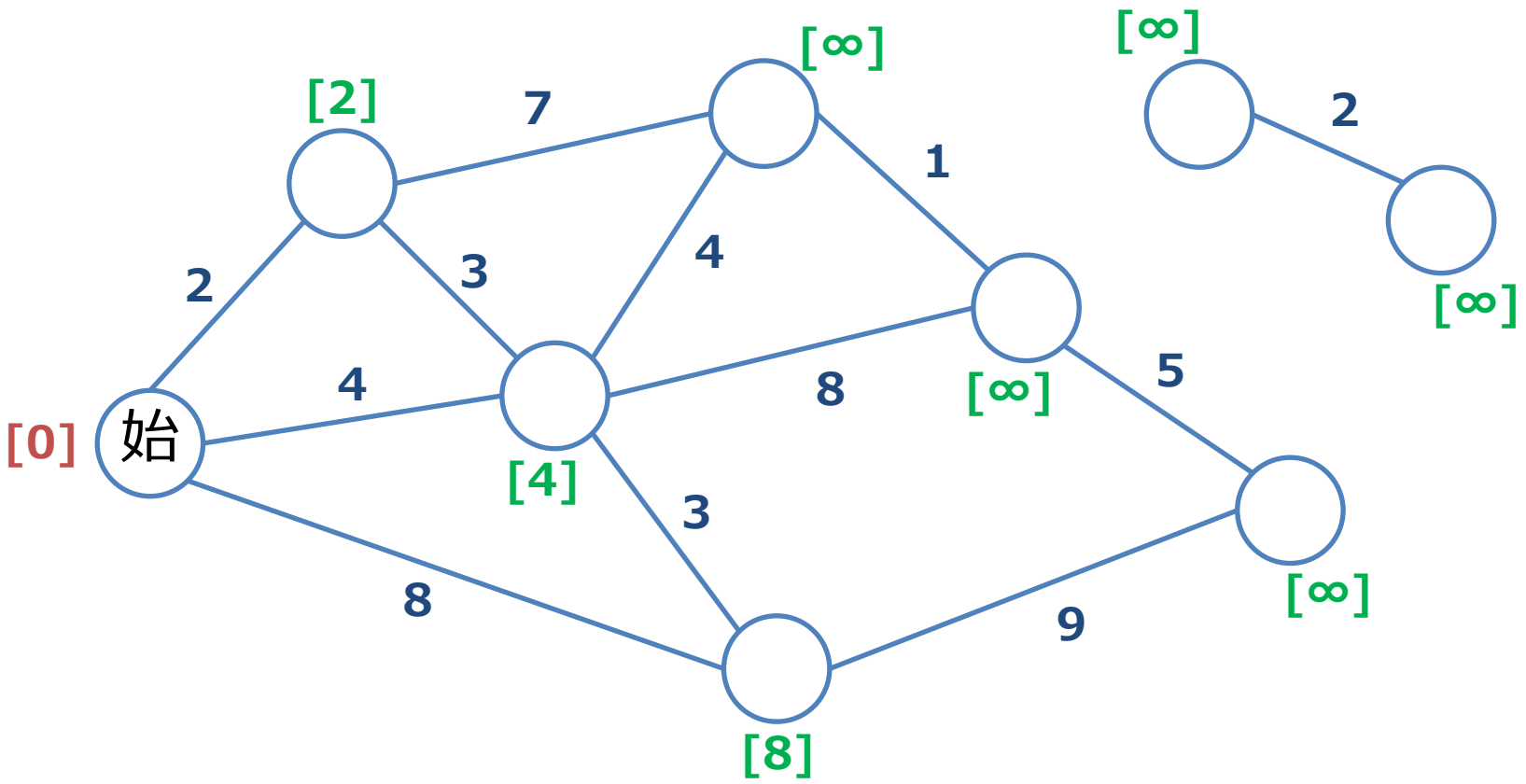
Dijkstra 法

- 方針：始点から近い順に確定させていく



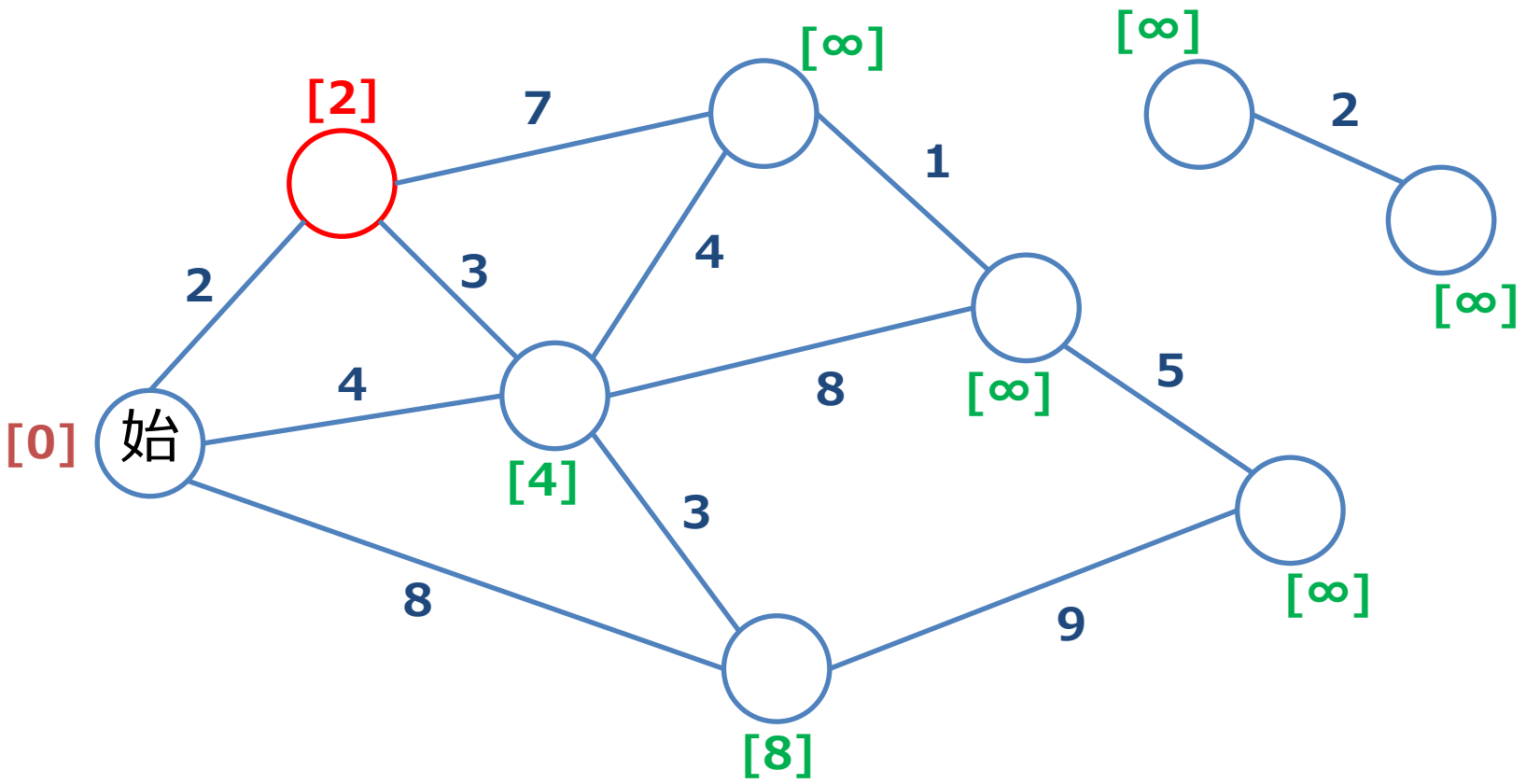
Dijkstra 法

- 方針：始点から近い順に確定させていく



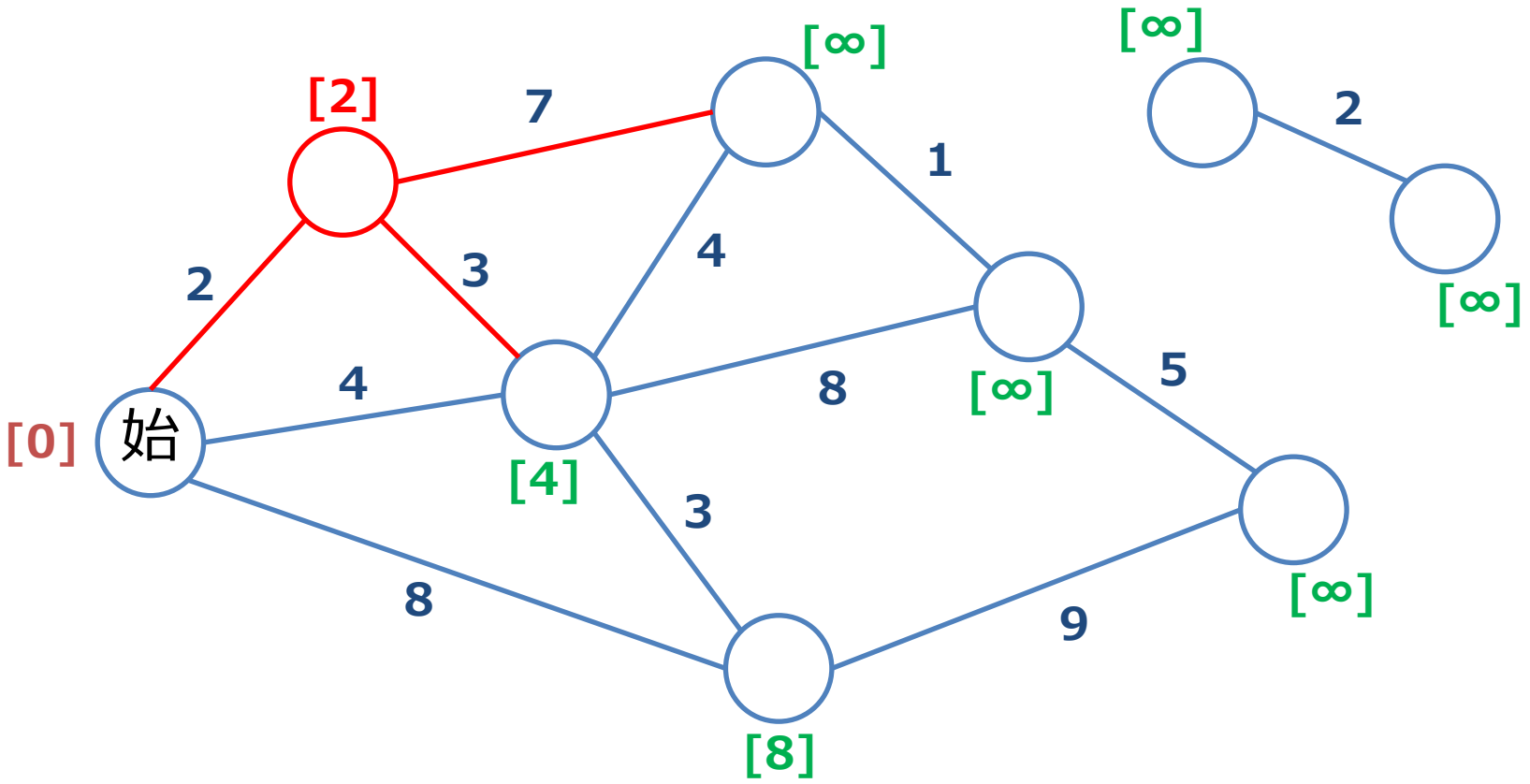
Dijkstra 法

- 方針：始点から近い順に確定させていく



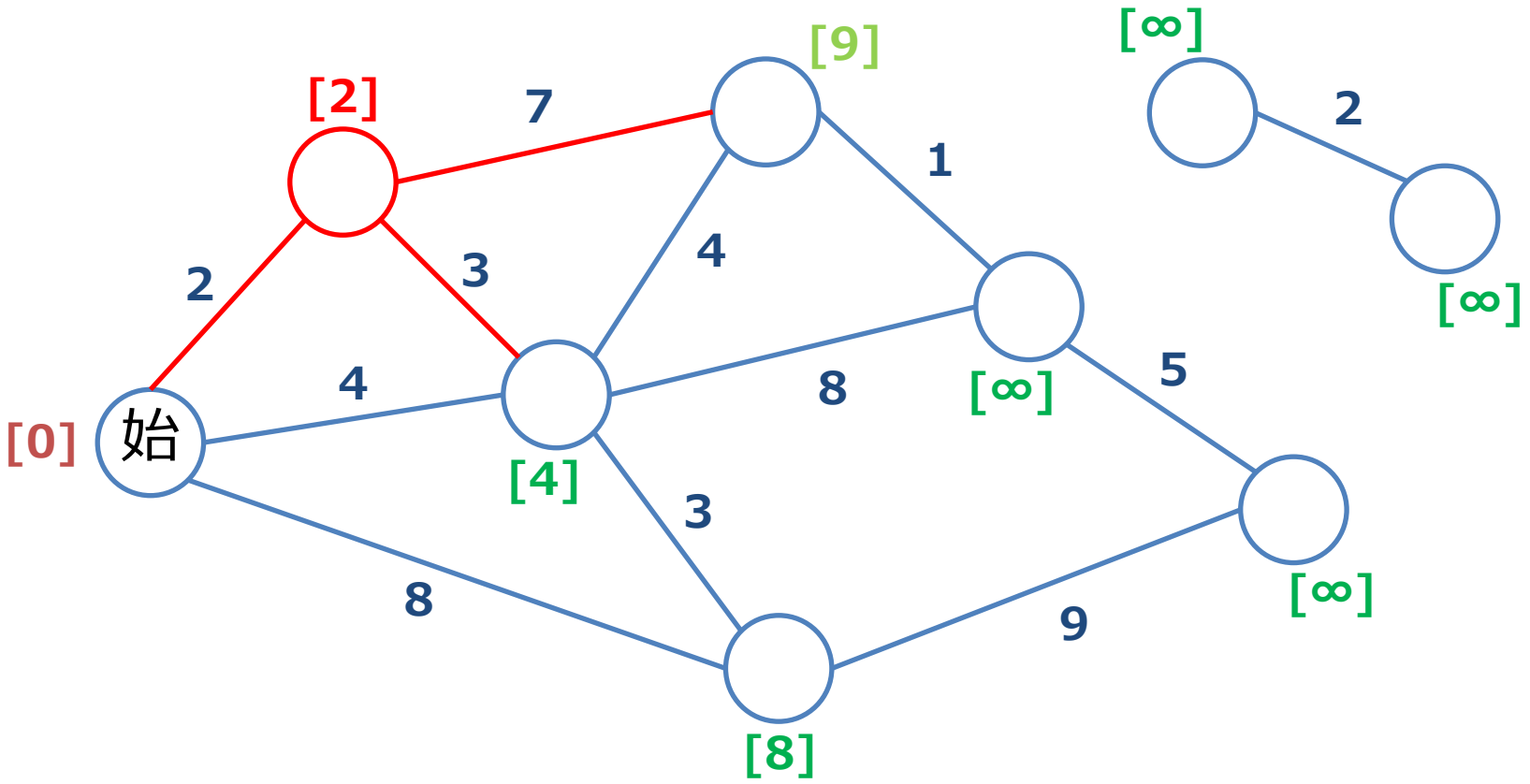
Dijkstra 法

- 方針：始点から近い順に確定させていく



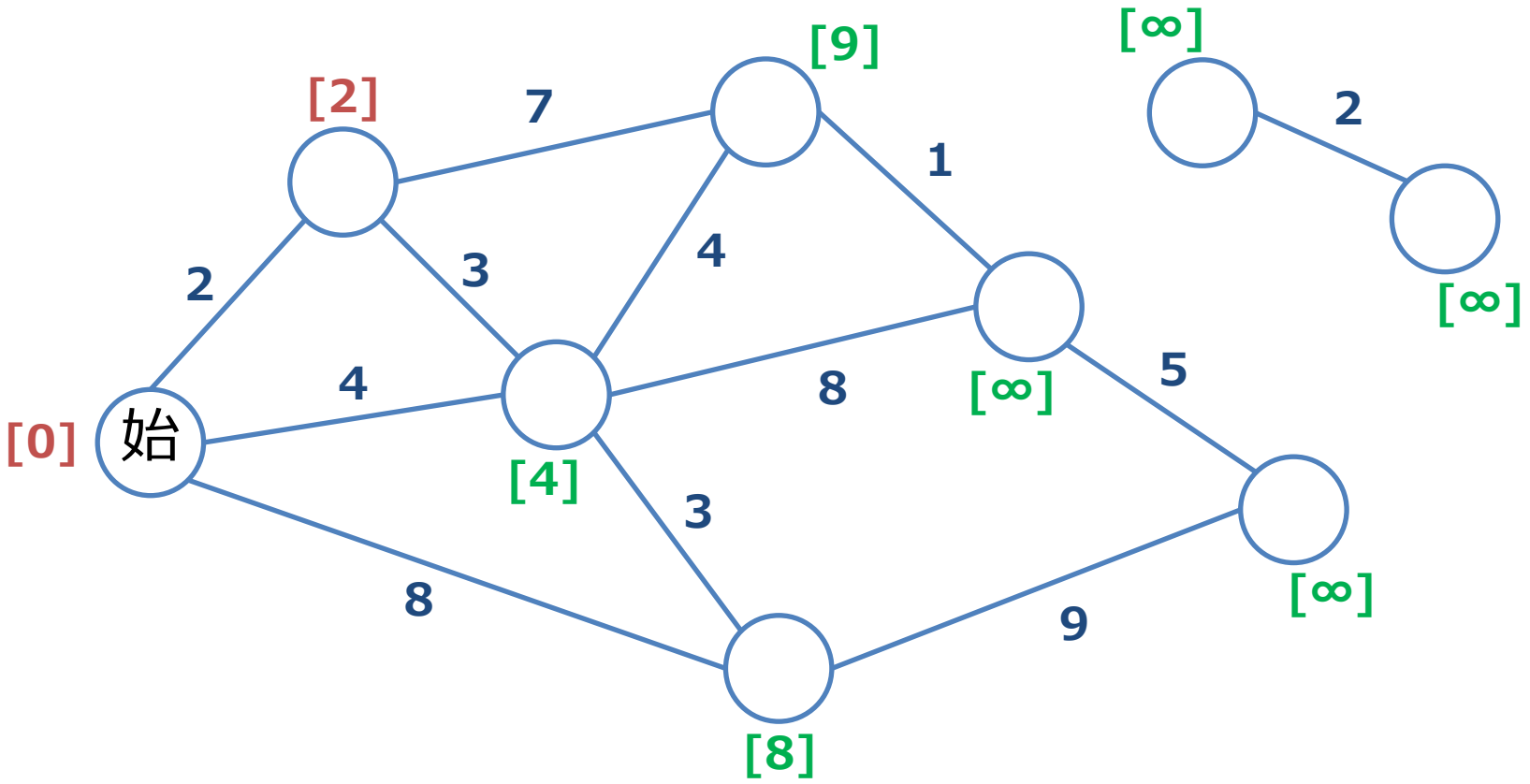
Dijkstra 法

- 方針：始点から近い順に確定させていく



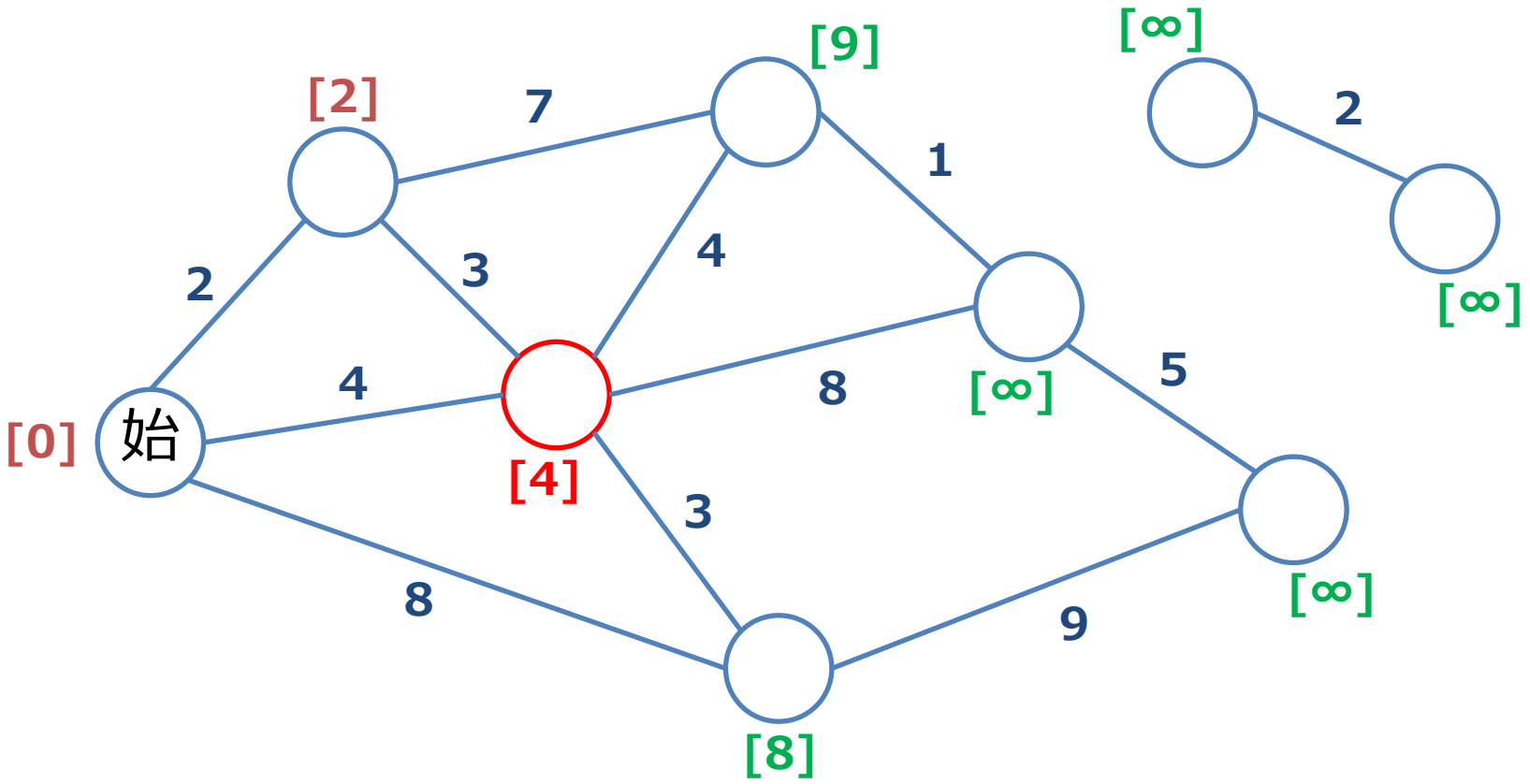
Dijkstra 法

- 方針：始点から近い順に確定させていく



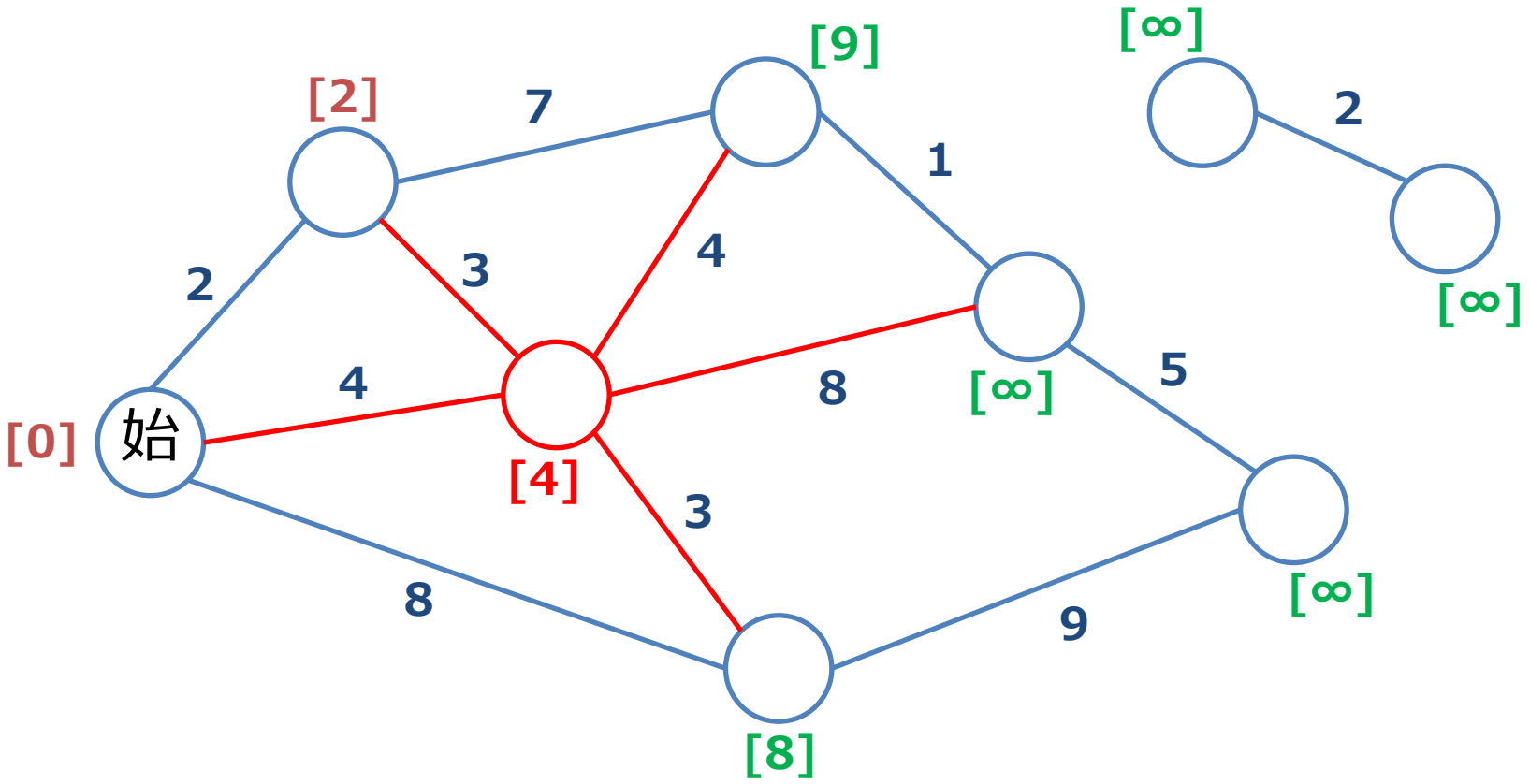
Dijkstra 法

- 方針：始点から近い順に確定させていく



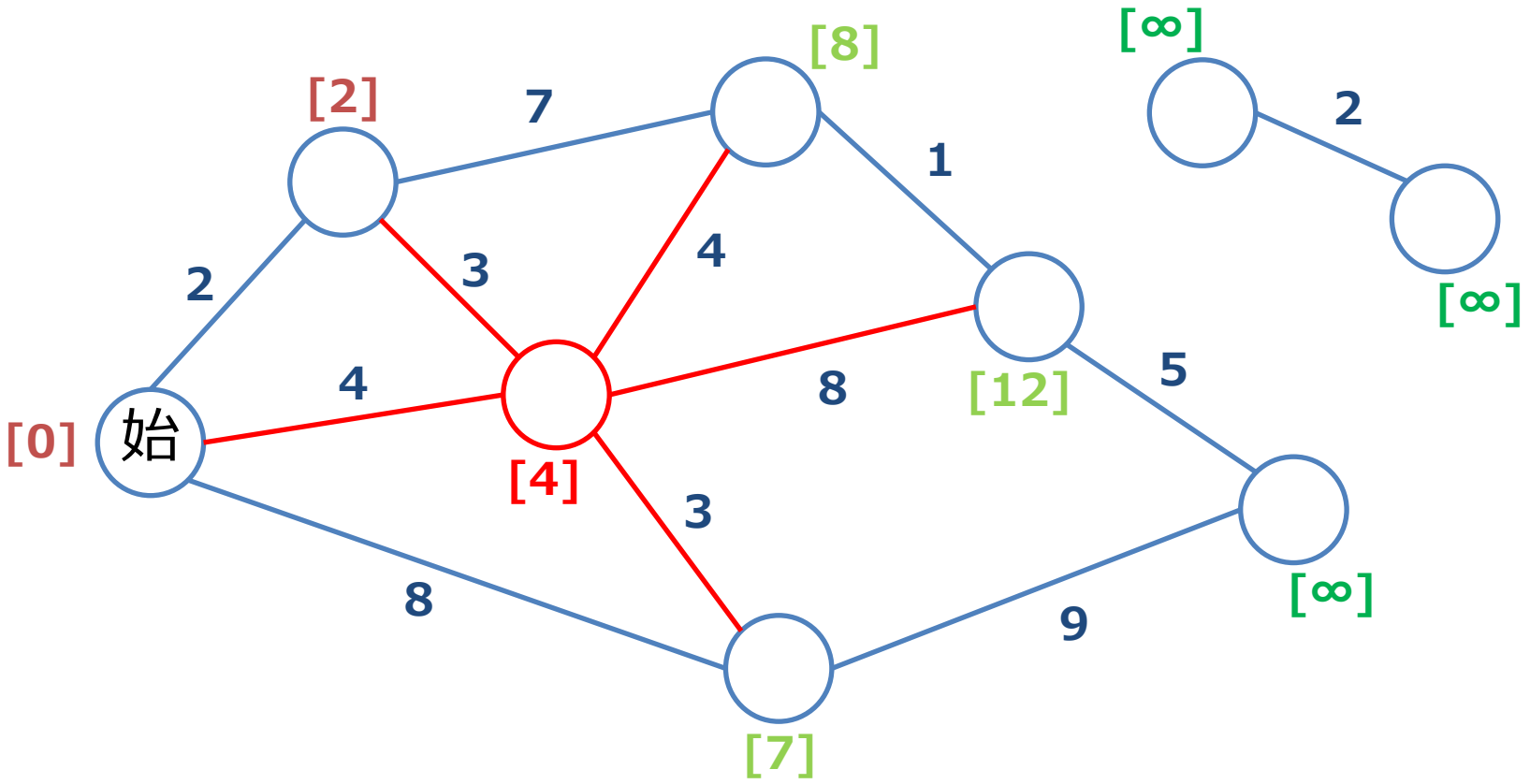
Dijkstra 法

- 方針：始点から近い順に確定させていく



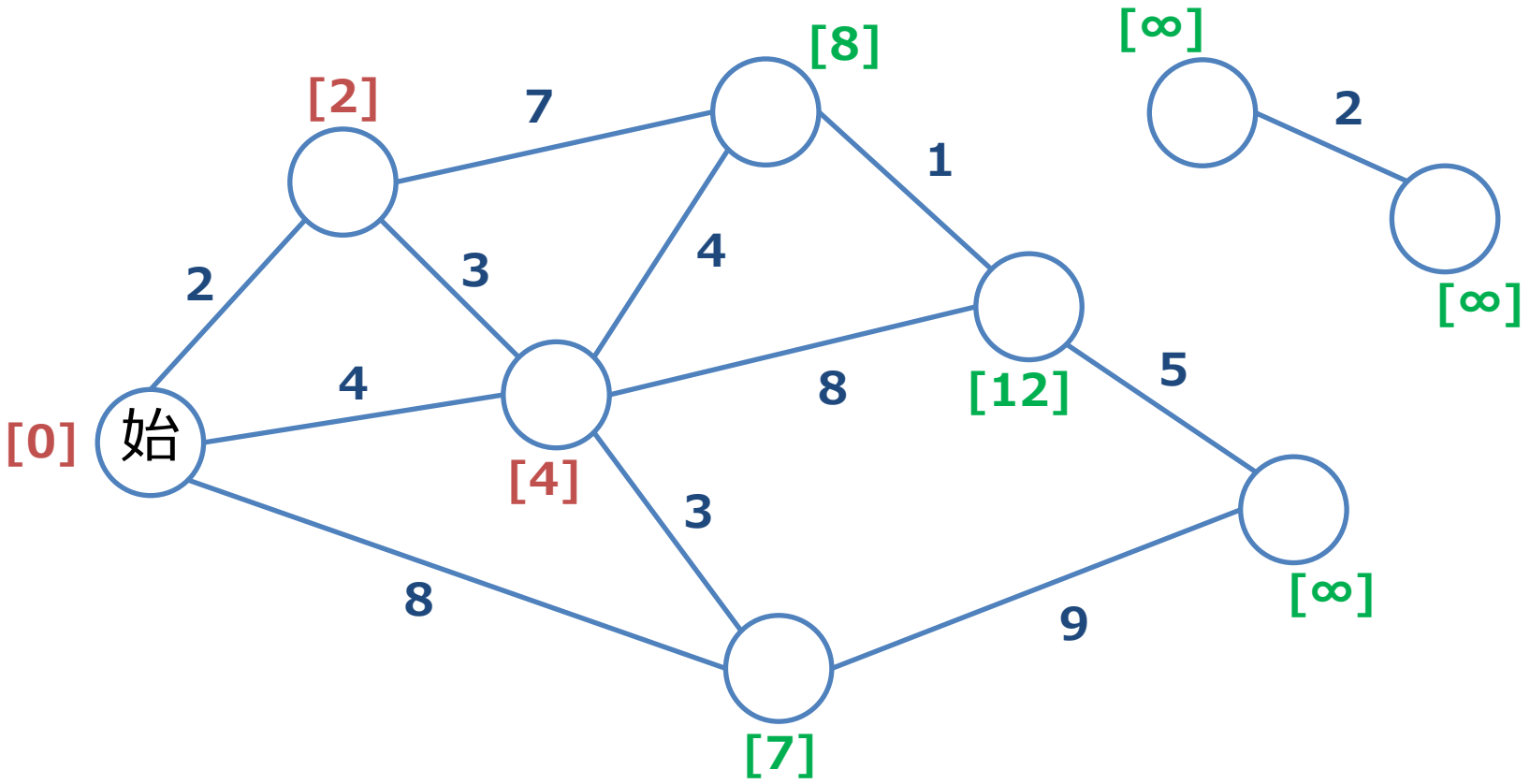
Dijkstra 法

- 方針：始点から近い順に確定させていく



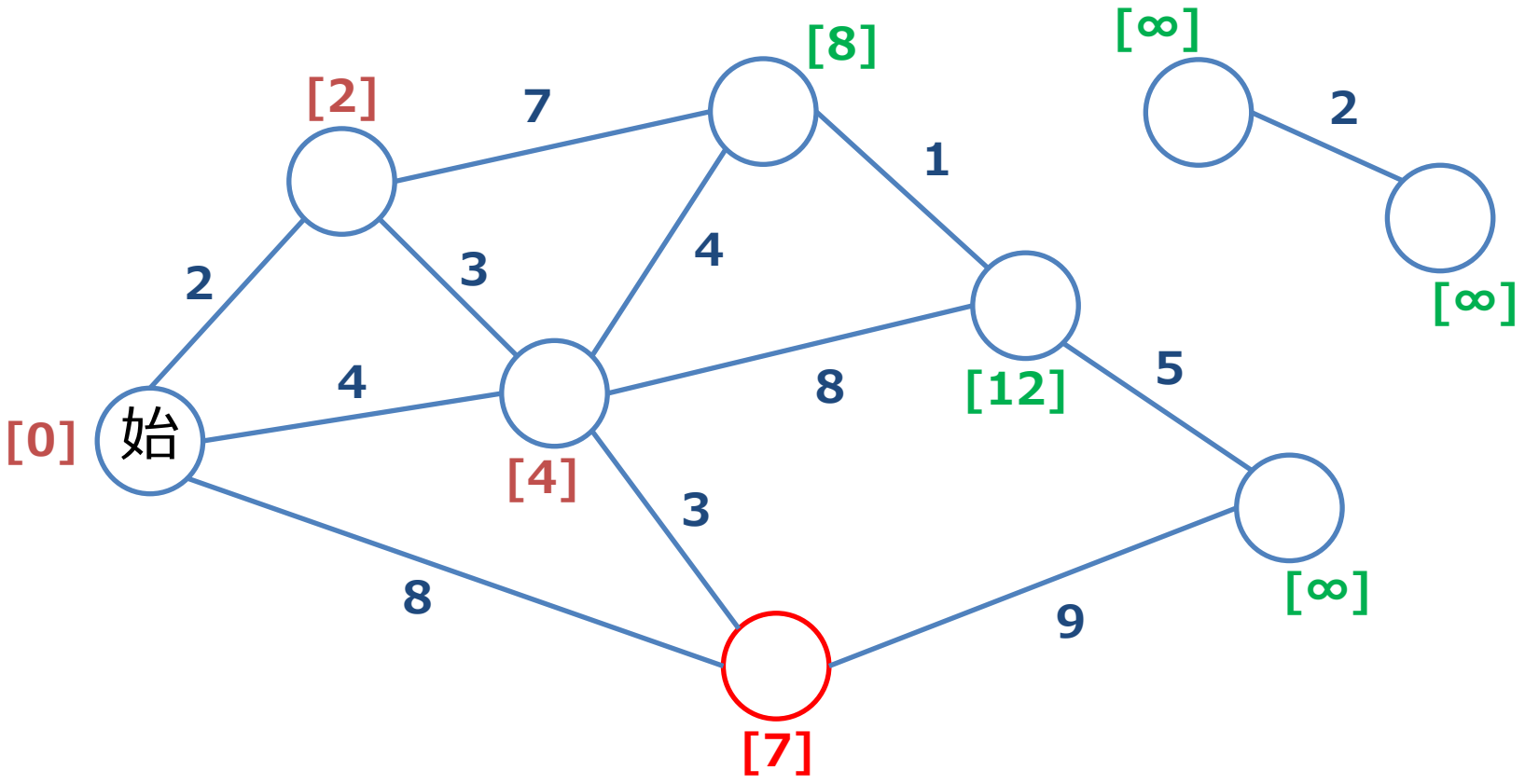
Dijkstra 法

- 方針：始点から近い順に確定させていく



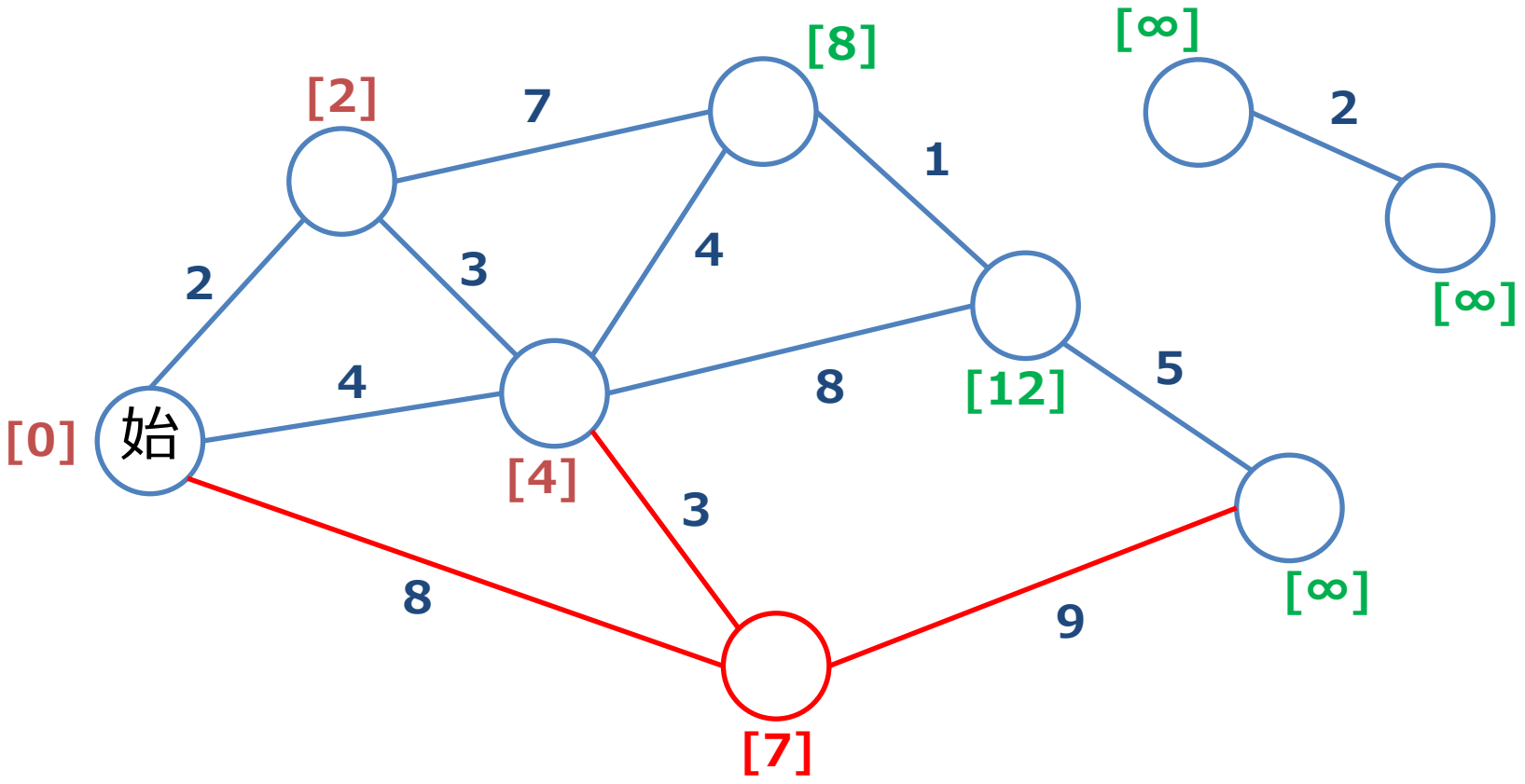
Dijkstra 法

- 方針：始点から近い順に確定させていく



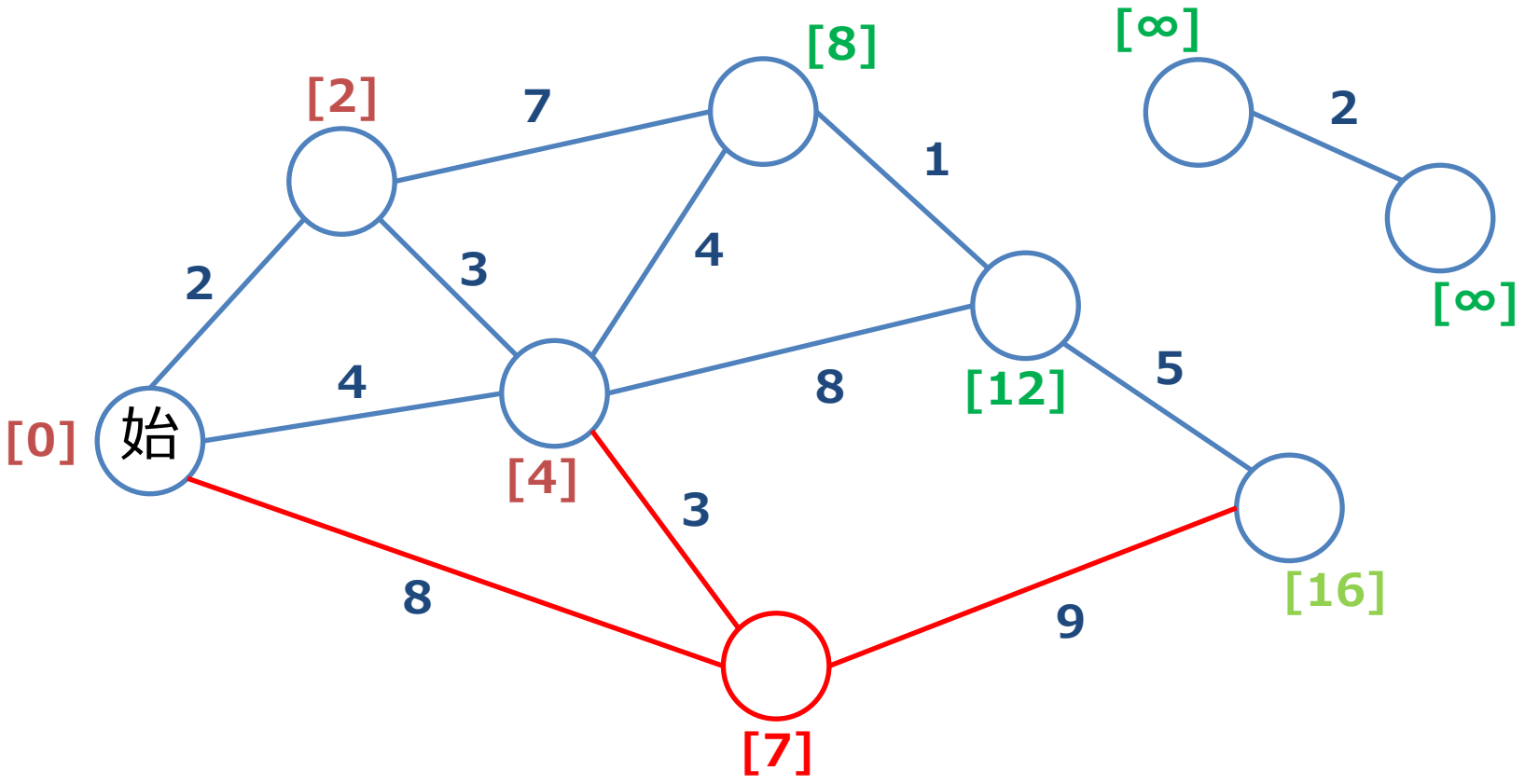
Dijkstra 法

- 方針：始点から近い順に確定させていく



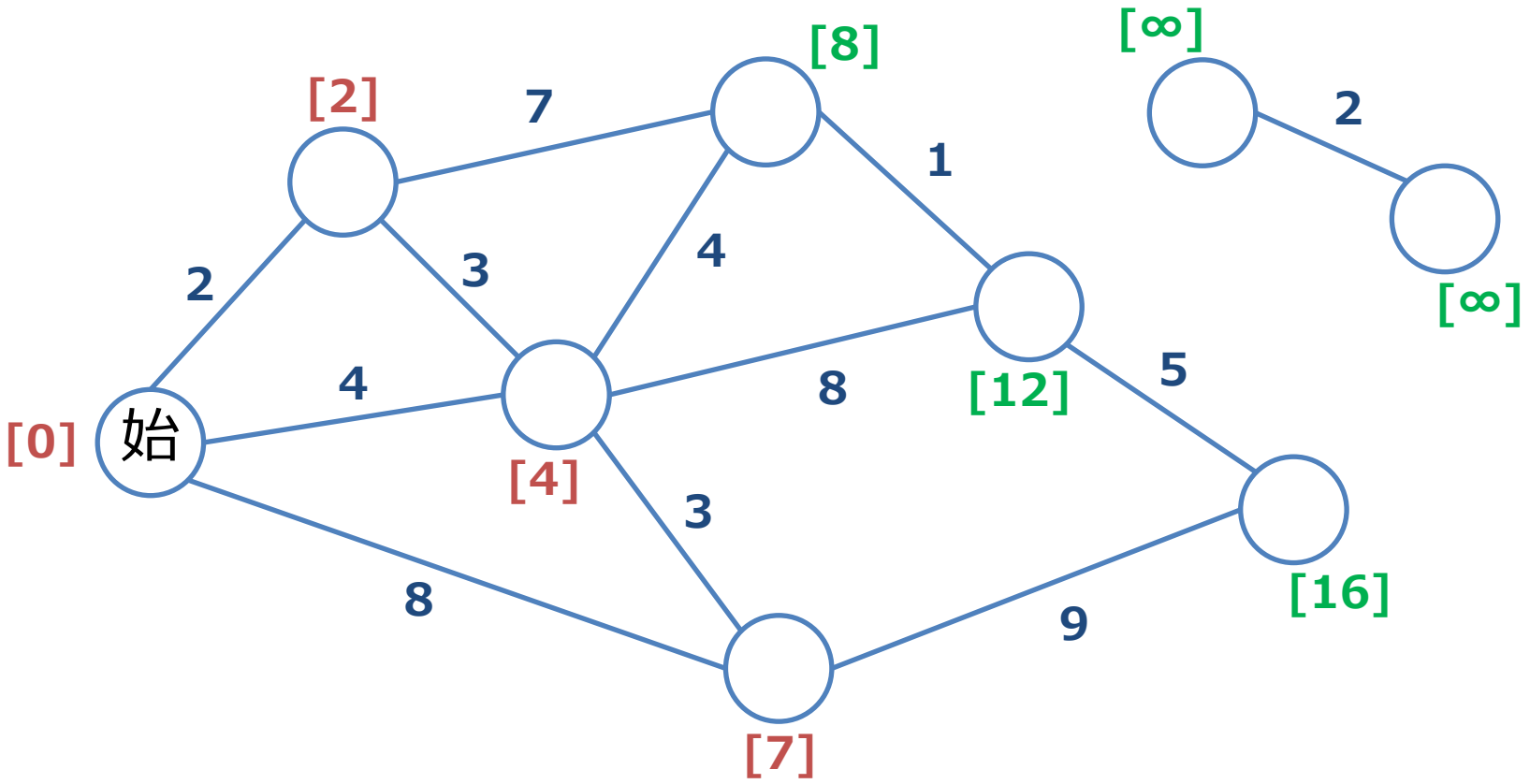
Dijkstra 法

- 方針：始点から近い順に確定させていく



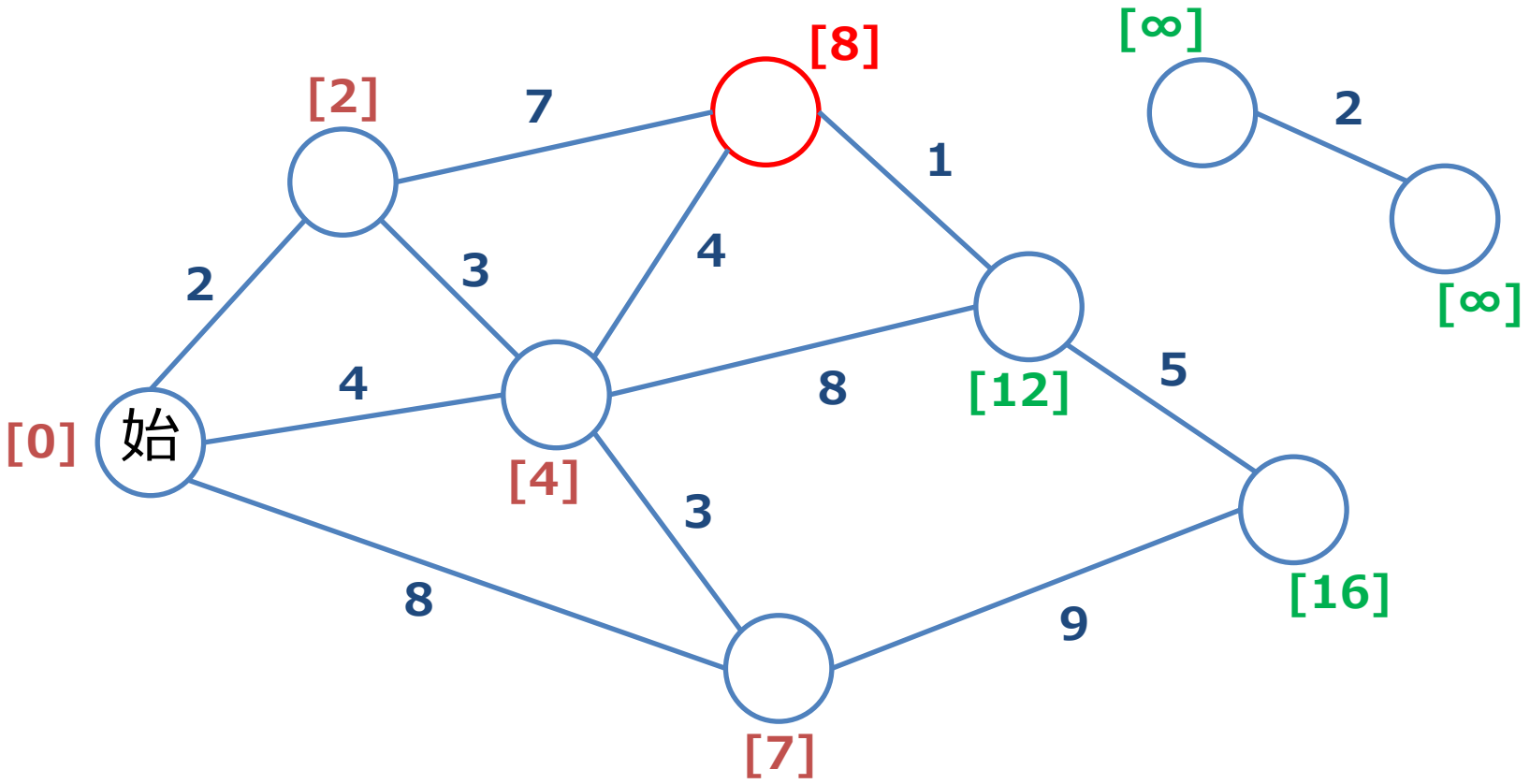
Dijkstra 法

- 方針：始点から近い順に確定させていく



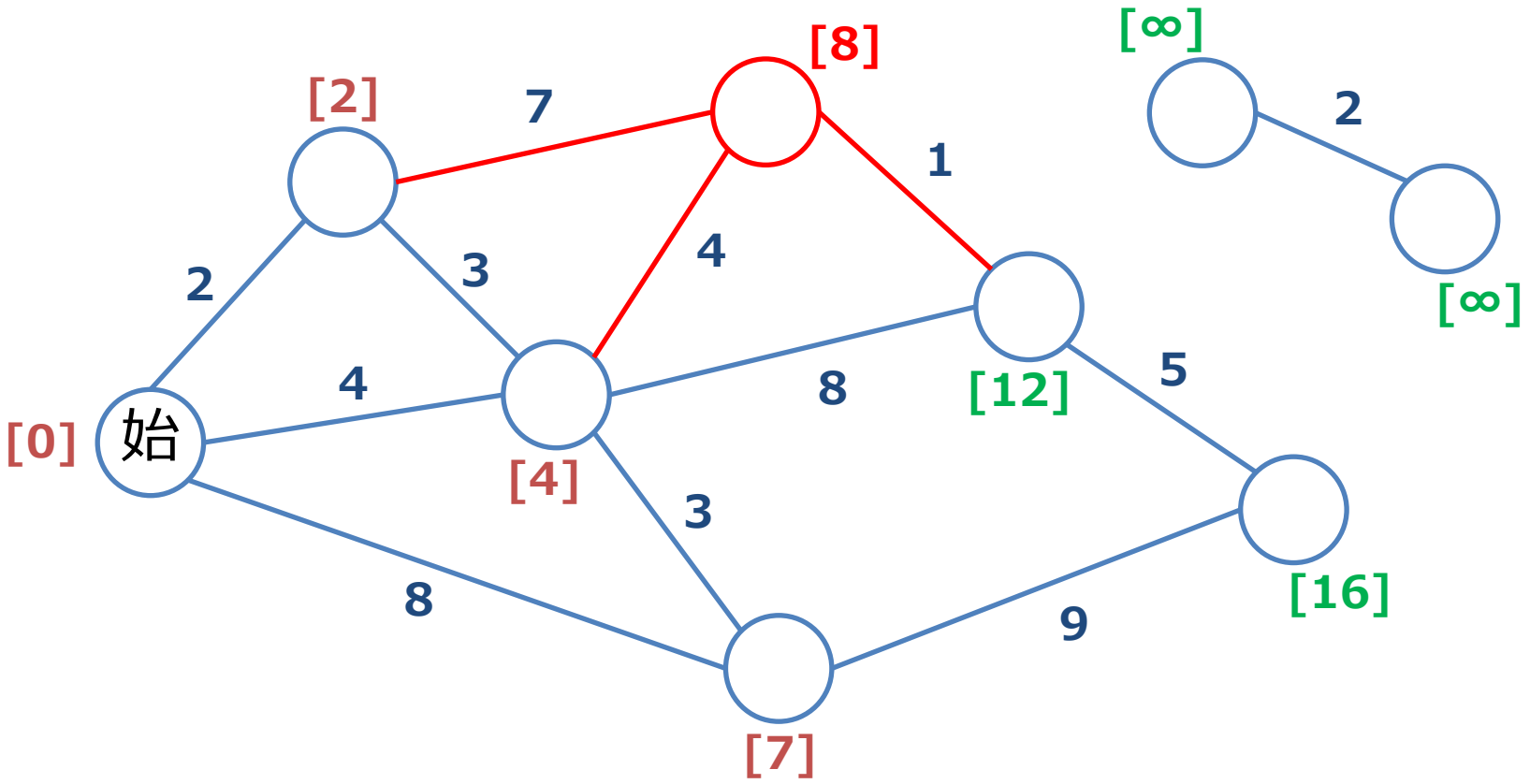
Dijkstra 法

- 方針：始点から近い順に確定させていく



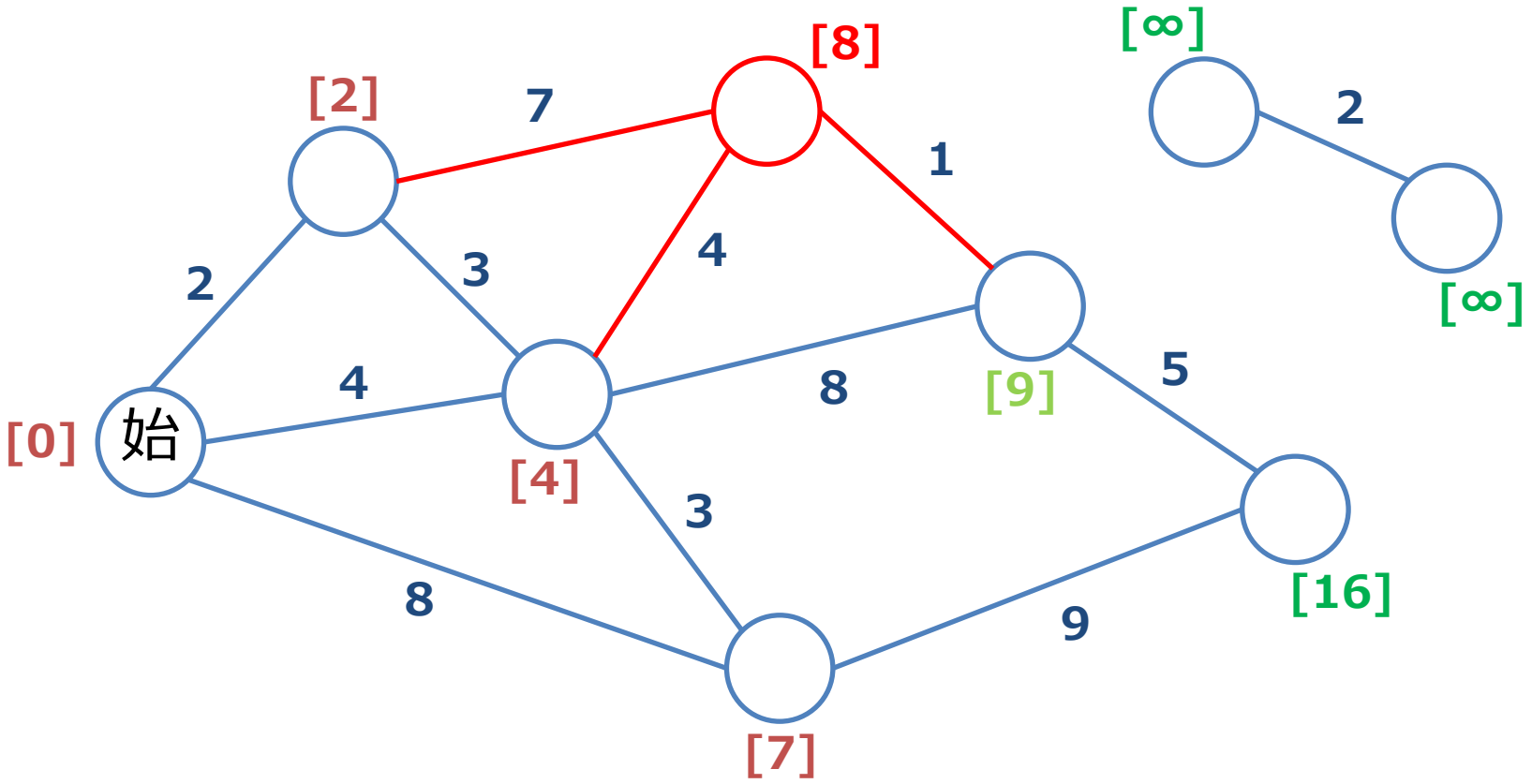
Dijkstra 法

- 方針：始点から近い順に確定させていく



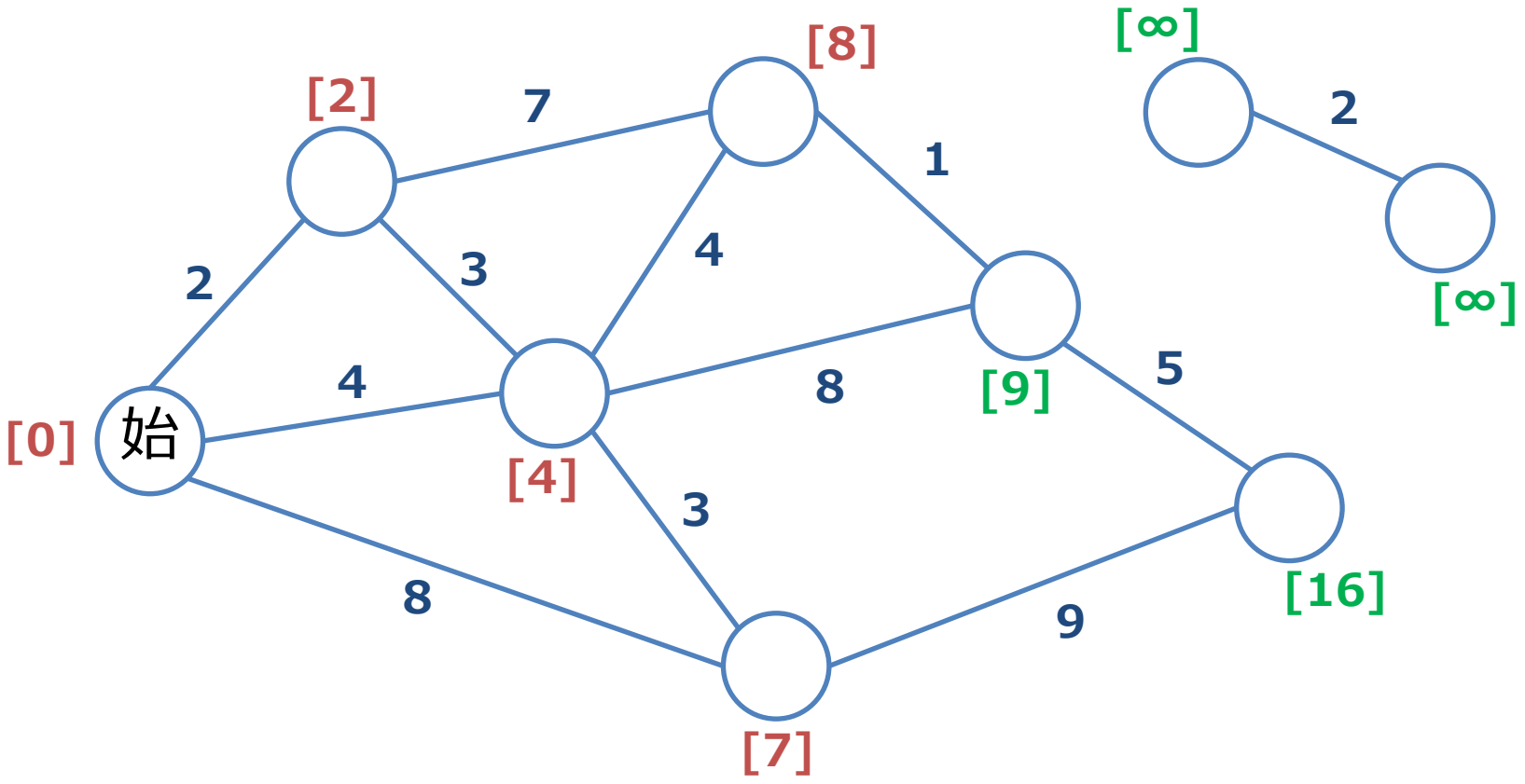
Dijkstra 法

- 方針：始点から近い順に確定させていく



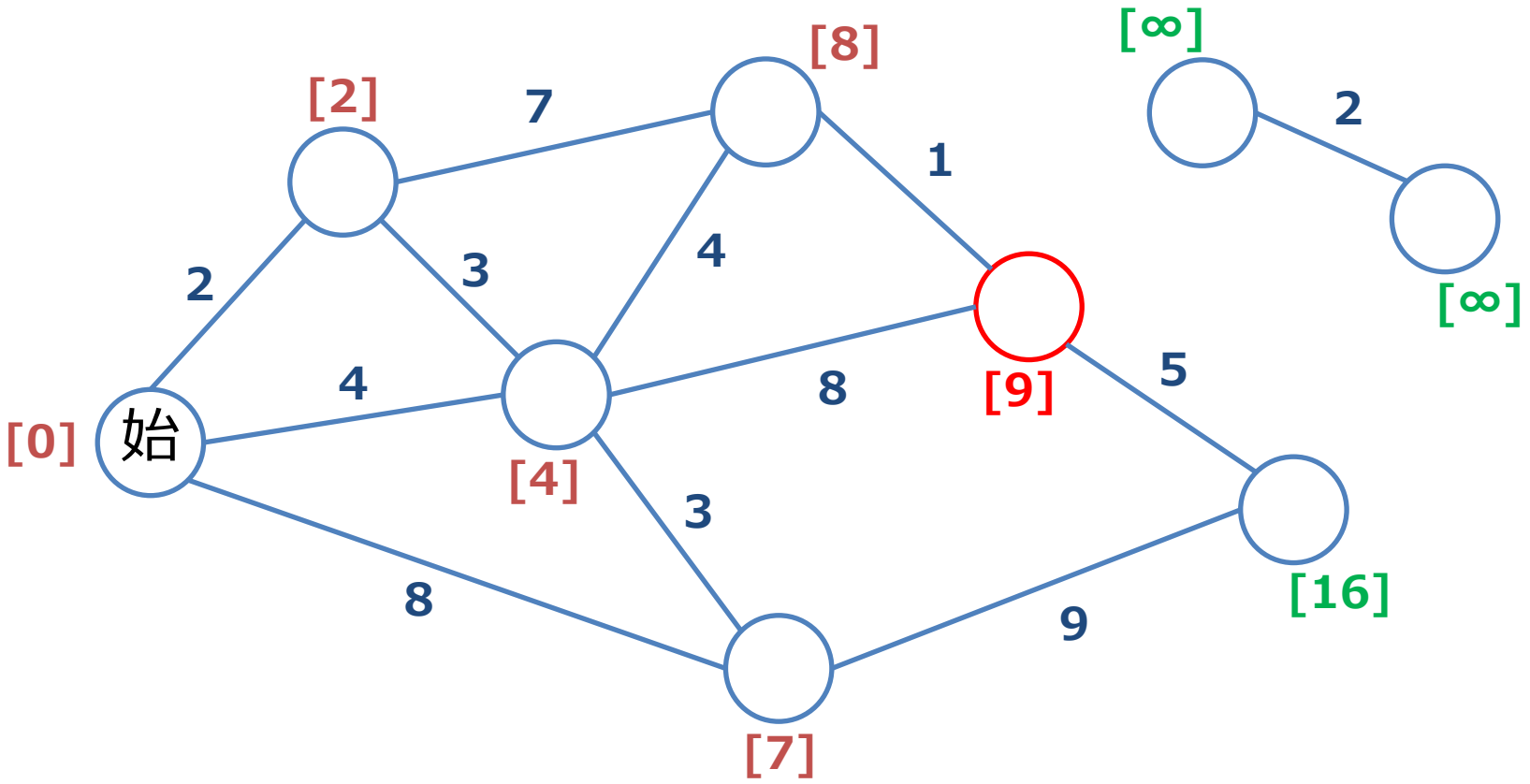
Dijkstra 法

- 方針：始点から近い順に確定させていく



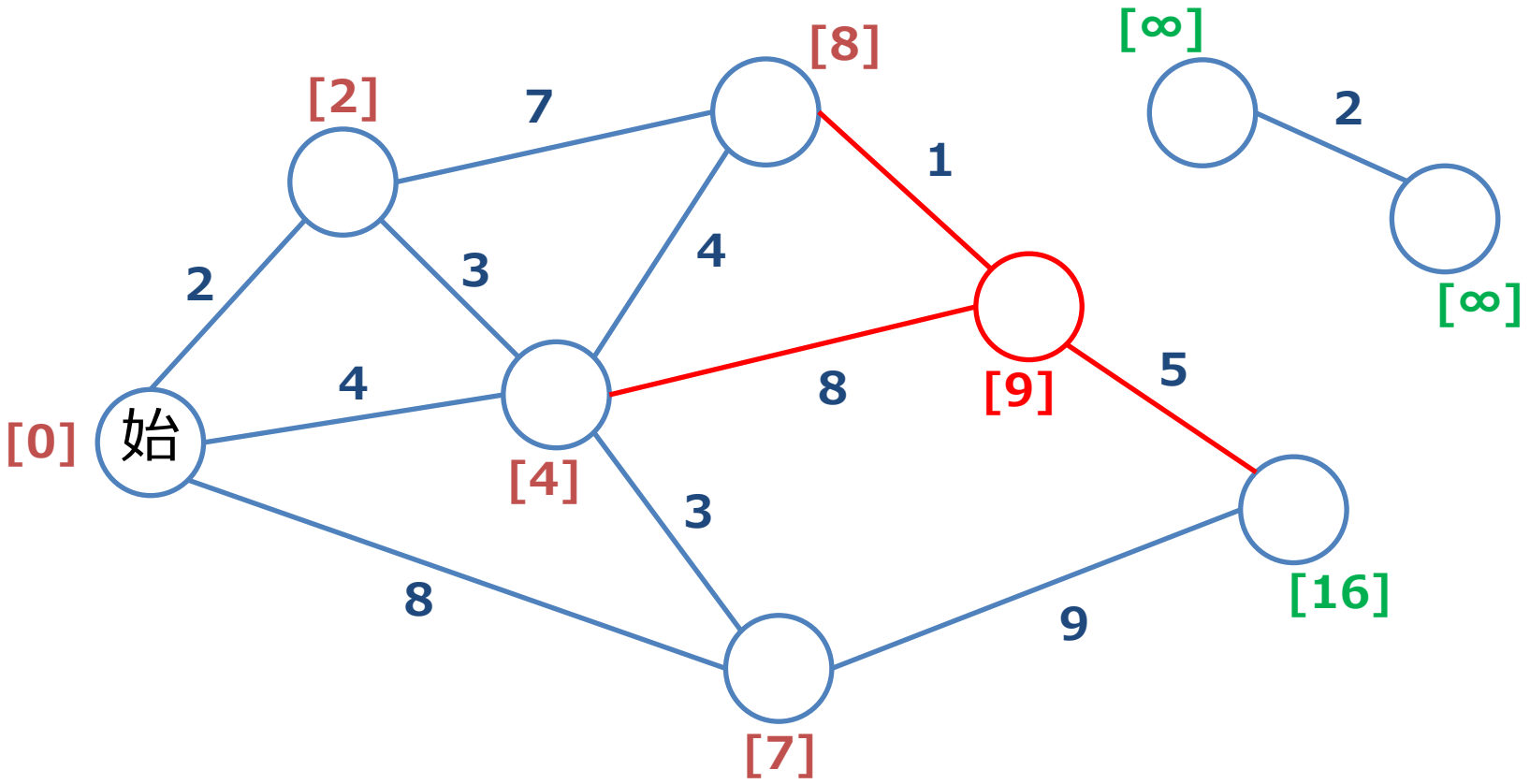
Dijkstra 法

- 方針：始点から近い順に確定させていく



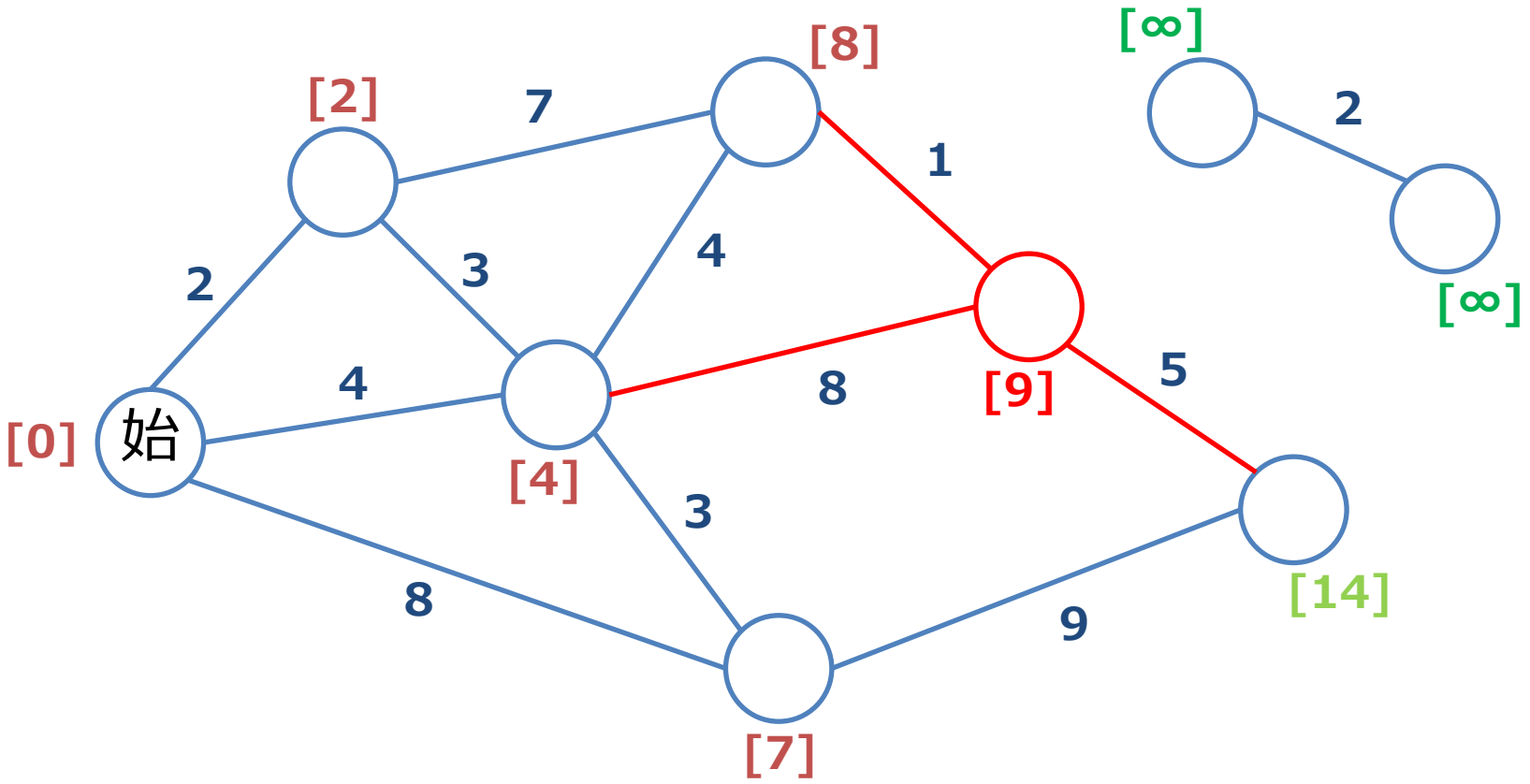
Dijkstra 法

- 方針：始点から近い順に確定させていく



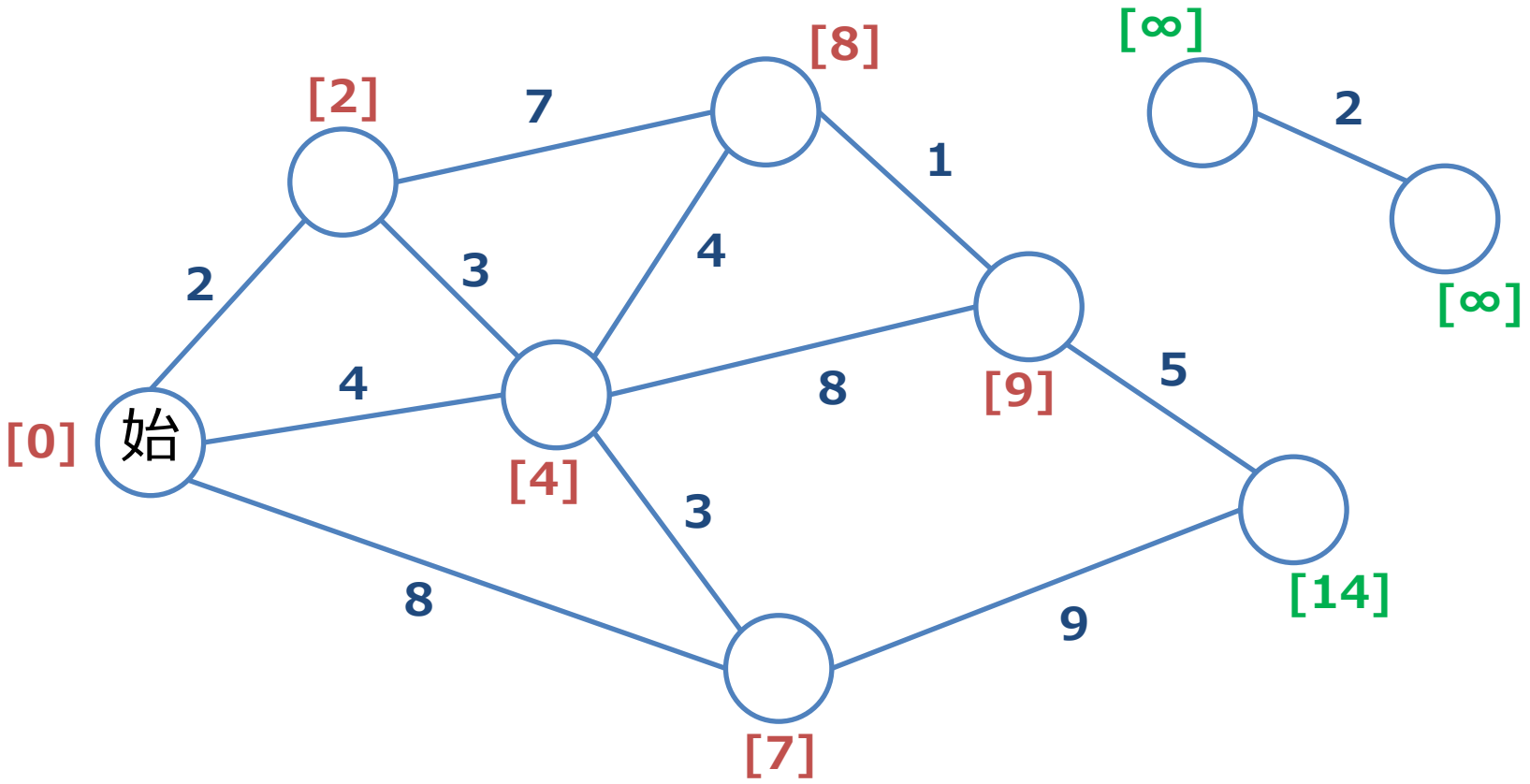
Dijkstra 法

- 方針：始点から近い順に確定させていく



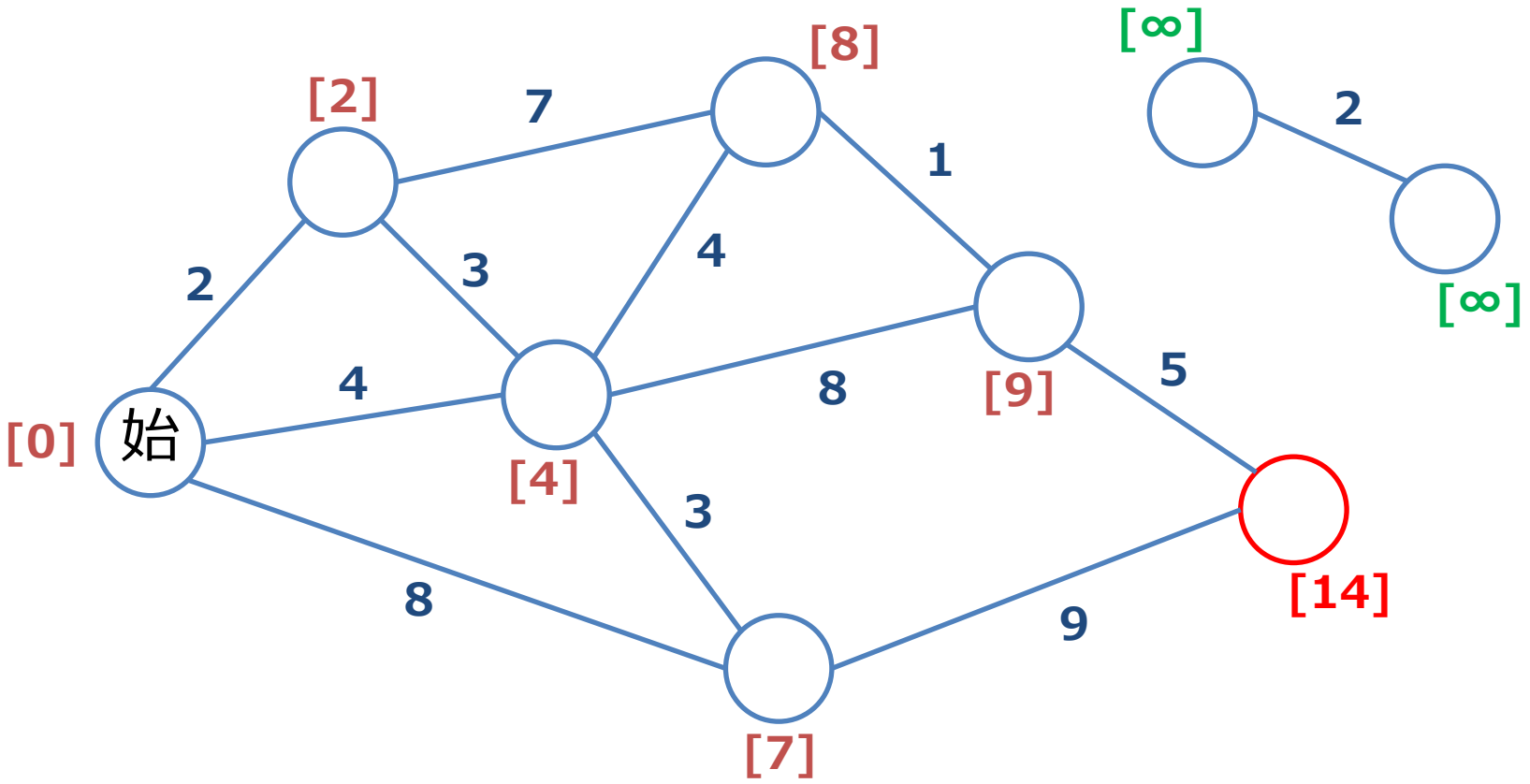
Dijkstra 法

- 方針：始点から近い順に確定させていく



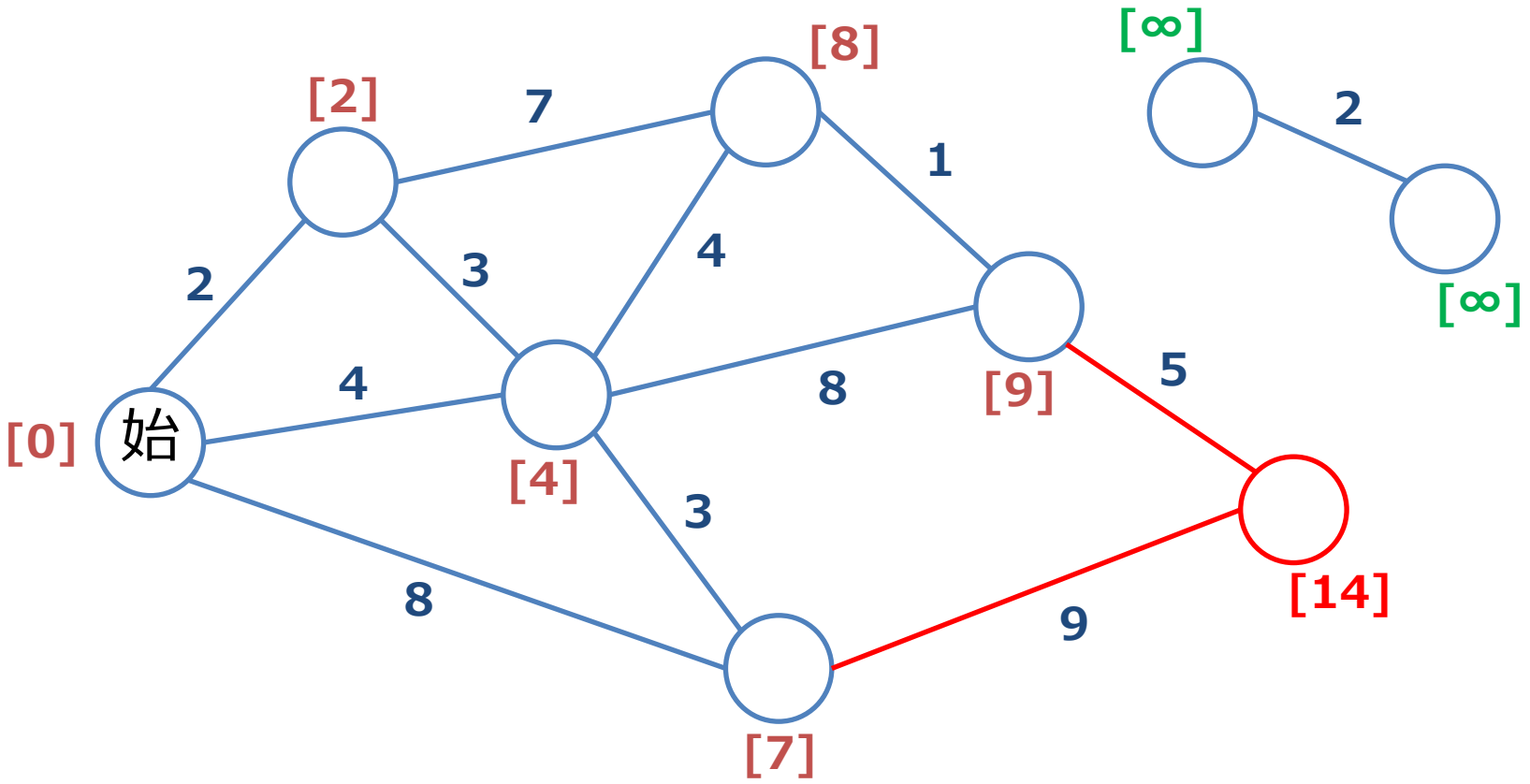
Dijkstra 法

- 方針：始点から近い順に確定させていく



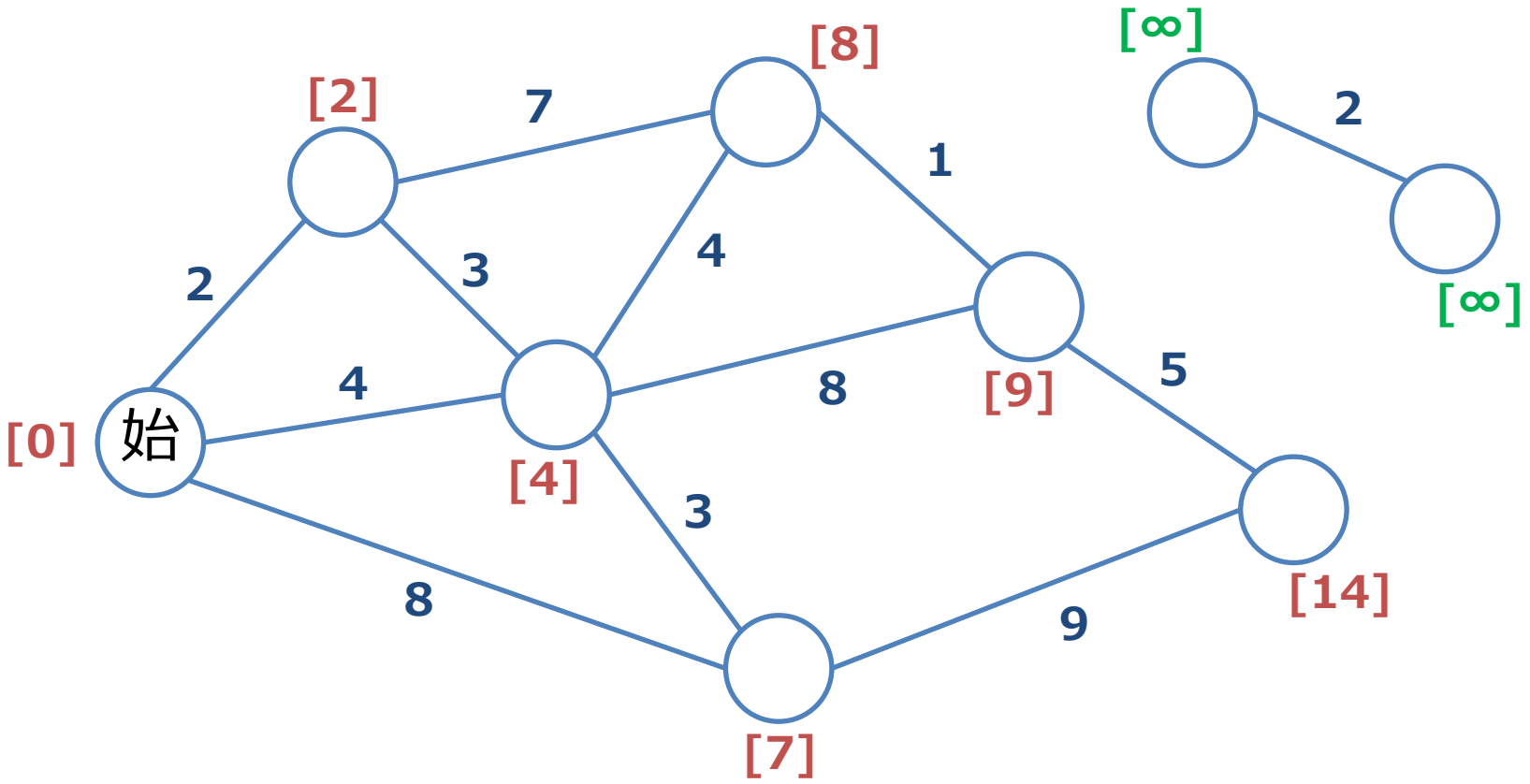
Dijkstra 法

- 方針：始点から近い順に確定させていく



Dijkstra 法

- 方針：始点から近い順に確定させていく



Dijkstra 法

各頂点 *u* に対し *dist*[*u*] := ∞, *flag*[*u*] := *false*.
dist[*start*] := 0.
以下を繰り返す:
    u := *flag* が *false* の中で *dist* が最小なもの.
    flag[*u*] := *true*.
    u に隣接している頂点 *v* それぞれに対し
        if *dist*[*v*] > *dist*[*u*] + (*u* から *v* への距離):
            dist[*v*] = *dist*[*u*] + (*u* から *v* への距離).

Dijkstra 法

- 計算量 (頂点数 V，辺数 E のとき)
  - 時間 $O(V^2)$
    - 繰り返しが高々 V 回
    - 「最も近い頂点を選ぶ」のに $O(V)$
  - メモリ $O(V)$

- 本問では，頂点数が $O(K)$ なので $O(K^2)$

Dijkstra 法

各頂点 *u* に対し *dist*[*u*] := ∞, *flag*[*u*] := *false*.
dist[*start*] := 0.
以下を繰り返す:

u := flag が false の中で dist が最小なもの.

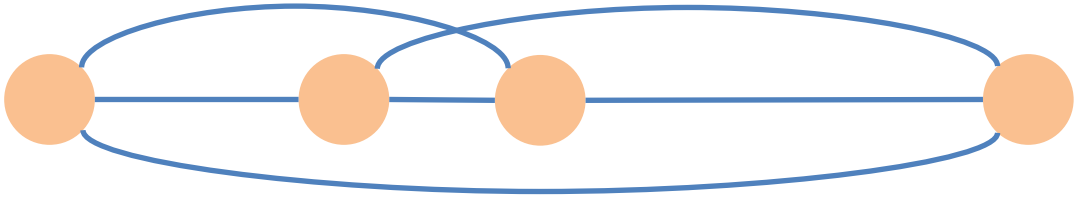
flag[*u*] := *true*.
u に隣接している頂点 *v* それぞれに対し
if *dist*[*v*] > *dist*[*u*] + (*u* から *v* への距離):
dist[*v*] = *dist*[*u*] + (*u* から *v* への距離).

Dijkstra 法

- Dijkstra 法の高速化
  - 優先度付きキュー (priority queue)
    - 「要素の追加」「最小値の取得・削除」の操作を $O(\log$ (要素数)) 時間で行うデータ構造
    - 二分ヒープで実装される
    - C++ の標準ライブラリ std::priority_queue
  - 時間 $O(E \log V)$
    - 注： $E = O(V^2)$ のときは $O(V^2)$ より遅い

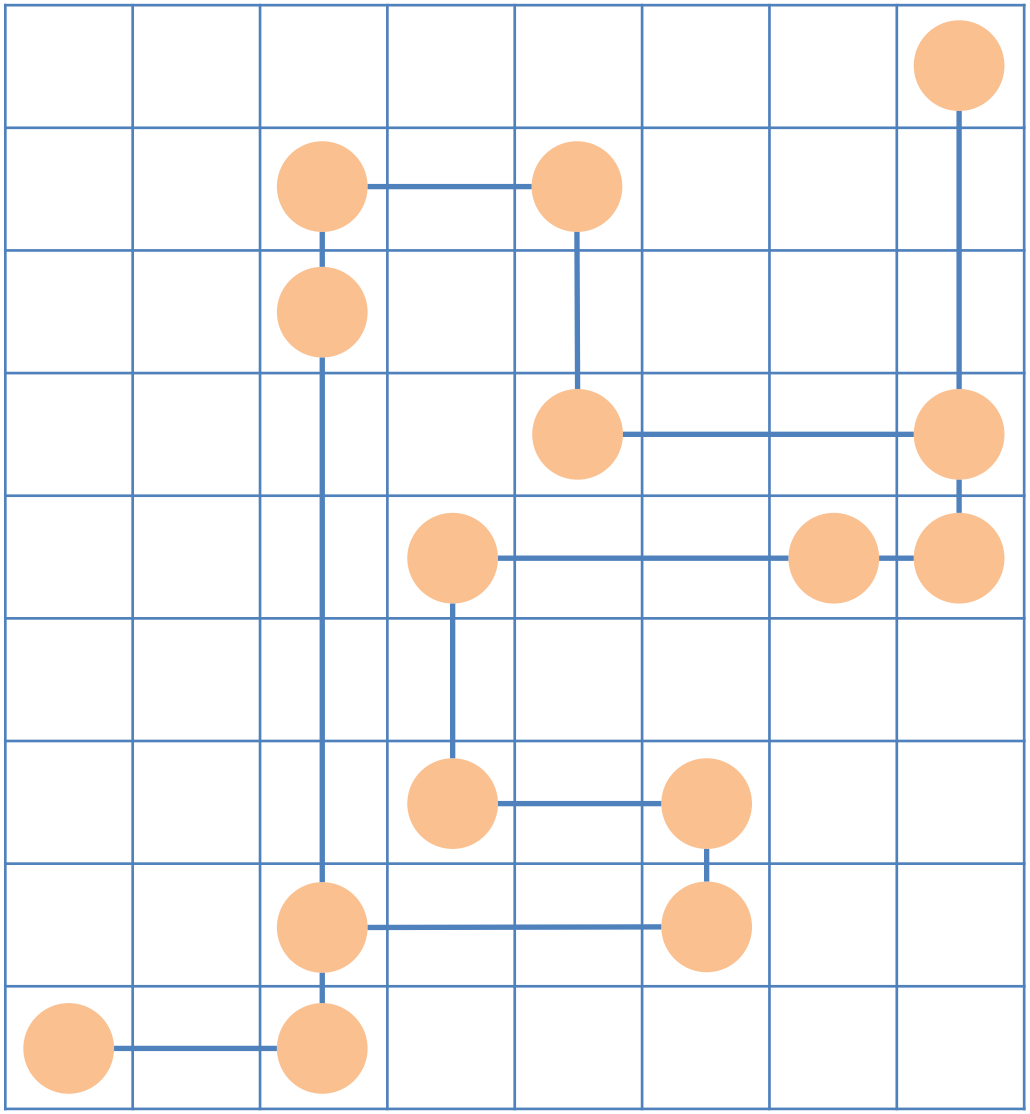
満点解法

- 辺数を $O(K^2)$ から $O(K)$ にしたい
  - 辺の張りすぎ



  - 辺数を減らす

満点解法



満点解法

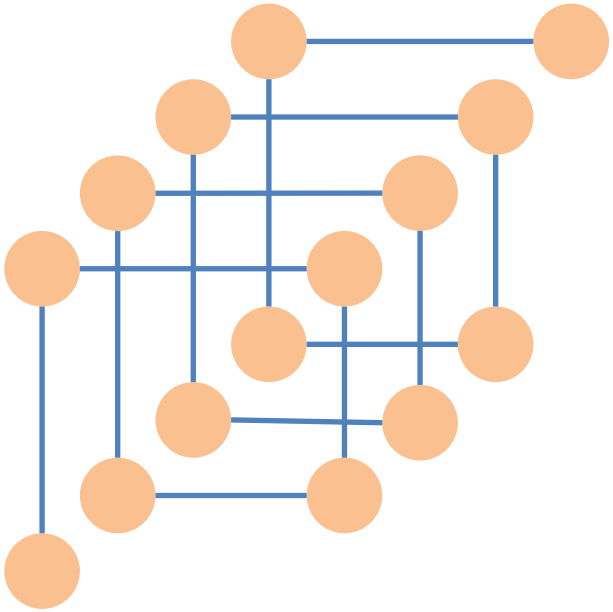
- 不要な辺を張らない
  - 必要なマスを x 座標の小さい順→ y 座標の小さい順でソートして，隣り合ったもの以外は縦の辺で結ばない
  - 必要なマスを y 座標の小さい順→ x 座標の小さい順でソートして，隣り合ったもの以外は横の辺で結ばない
- これで辺数が $O(K)$，全体で $O(K \log K)$

注意点

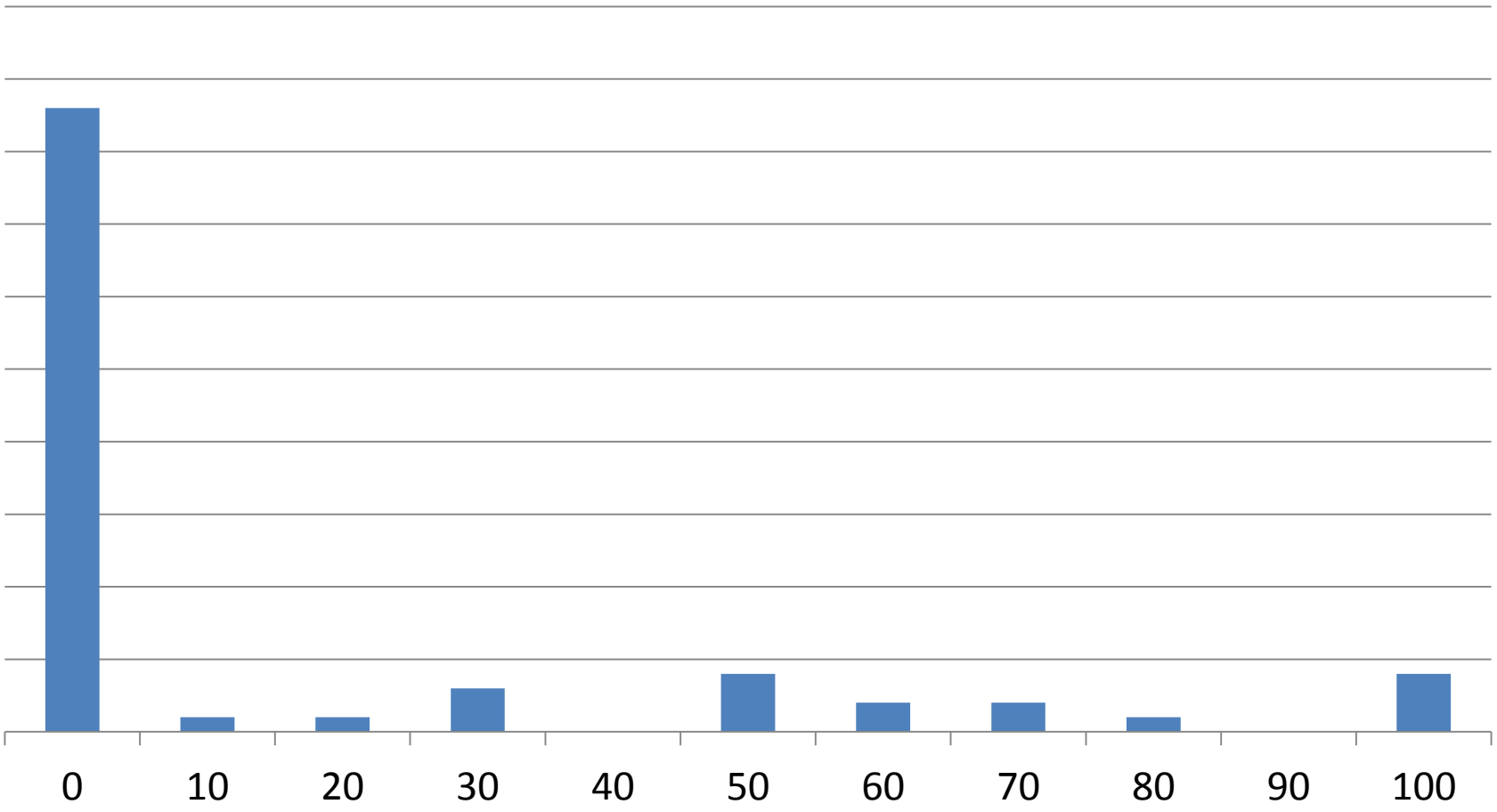
- (1, 1) や (M, N) にはスイッチがあったりなかったりする
  - 頂点数いつもが $2K$ ではないので実装に注意
  - サンプルにもあり

注意点

- 答えが非常に大きくなる可能性
  - 32-bit 整数型 (int) のオーバーフロー
    - 答えが 10^{10} くらいになる例：



得点分布



おまけ

- priority queue についての参考資料
  - http://hos.ac/blog/