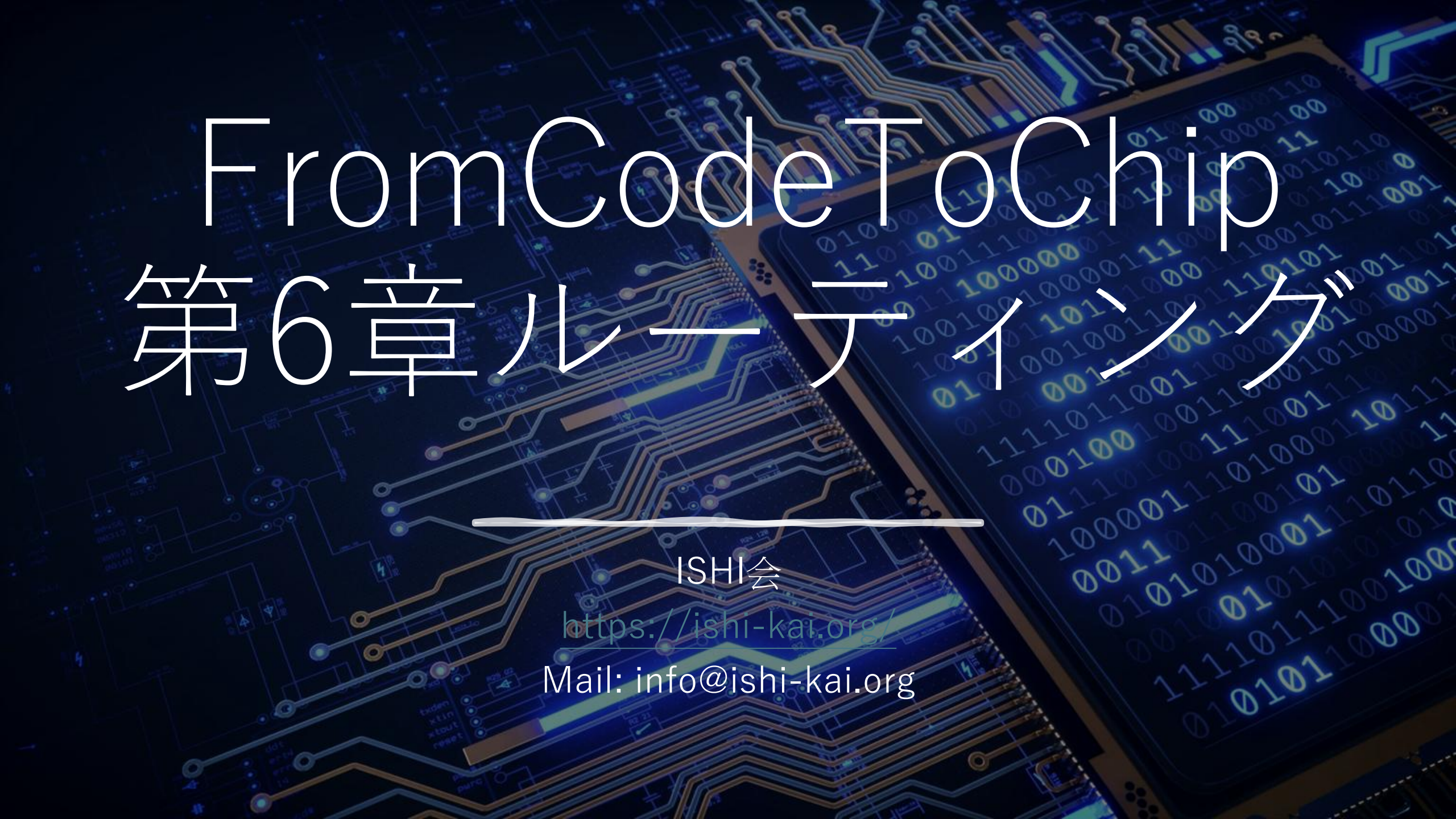
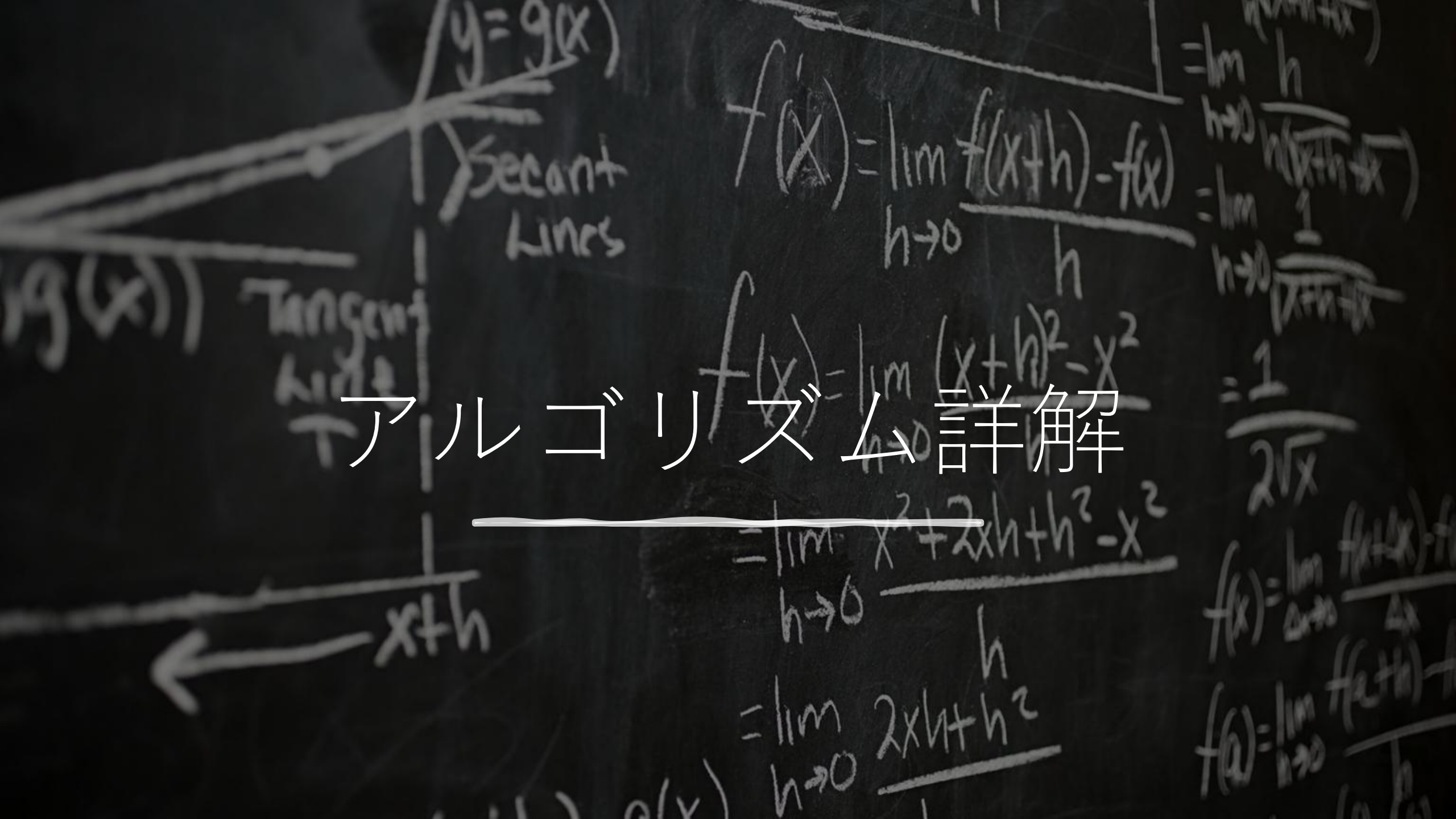


FromCodeToChip
第6章ルーティング

ISHI会
https://ishi-kai.org/
Mail: info@ishi-kai.org

6章の目次と本解説の目次

- 書籍の目次
  - 6.1 Detailed Routing Grid
  - 6.2 Resources
  - 6.3 Obstacles
    - 6.3.1 Cache Data Structure
    - 6.3.2 Feasibility Check
  - 6.4 Wire Planning
    - 6.4.1 Planning Graph and Modeling Resources
    - 6.4.2 Net Planning
  - 6.5 Detailed Routing
    - 6.5.1 Path
    - 6.5.2 Path to Path Routing
    - 6.5.3 Multi-Path Routing
    - 6.5.4 Route
  - 6.6 Magic View
- 本プレゼンの目次
  - アルゴリズム詳解
  - A*アルゴリズム詳解・2点間の配線
  - A*アルゴリズム詳解・障害物がある場合
  - 本章での配線の解説


y=g(x)
Secant Lines
Tangent
x+h
f(x)=lim h→0 f(x+h)-f(x)/h
f(x)=lim h→0 (x+h)²-x²/h
=lim h→0 x²+2xh+h²-x²/h
=lim h→0 2xh+h²

アルゴリズム詳解

本章の配線（ルーティング）とは何か？

- 配置済みのデバイス間の物理的な接続を見つける配線タスク
  - 取り上げる配線エンジン
    - Jason Congらが開発したDUNE配線システム

- 本章での手順
  - 1st:混雑駆動型配線プランナー
    - 粗いグリッド上で各ネットに対して大まかなガイドを提供するシナリオを決定
  - 2nd:グリッドレスな詳細配線ルーティング
    - 正確なルートを発見する
    - 配線空間に存在する障害物の取り扱いによって補完

基本アルゴリズム：A*アルゴリズム

- 最短経路問題のアルゴリズム
  - 与えられたグラフにおいて、ある始点から終点[illegible]路を構成する辺の重みの合計が最小となる経路を見つ[illegible]

- A*アルゴリズムの評価関数 f(n) = g(n) + h(n)
  - g(n): スタートノードから現在のノードnまでの実[illegible]コスト
    - これは、すでに通ってきた道のりの総コスト[illegible]す。「どれだけの距離をすでに走ってきたか」に相当します。
  - h(n): 現在のノードnからゴールノードまでの推定コスト
    - これがヒューリスティック関数によって計算される予測値です。「目的地まであとどれくらいか」という見込みです。

これがキモ

手順：
A*アルゴリズム

1. スタートノード（S）を作成する。また計算中のノードを格納しておくための優先度付きキュー OPENリストと、計算済みのノードを格納しておくCLOSEリストを用意する。
2. ゴールは必ずしも 1 つである必要はないので、ゴール条件を満たすノード集合を G と呼ぶことにする。
3. スタートノードを Openリストに追加する、このとき g(S) = 0 であり f(S) = h(S) となる。また Closeリストは空にする。
4. Openリストが空なら探索は失敗とする（スタートからゴールにたどり着くような経路が存在しなかったことになる）。
5. Openリストに格納されているノードの内、最小の f(n) を持つノード n を1つ取り出す。同じ f(n) を持つノードが複数ある場合、タイブレーク手法によりどれかのノードを選択する。
6. n∈G であるなら探索を終了する。それ以外の場合は n を Close リストに移す。
7. n に隣接している全てのノード（ここでは隣接ノードを m とおく）に対して以下の操作を行う
    1. f′（m)=g(n)+COST(n,m)+h(m) を計算する、ここで COST(n,m) はノード n から m へ移動するときのコストである。また g(n)} は g(n)=f(n)-h(n)} で求めることができる。
    2. m の状態に応じて以下の操作を行う
        1. m が Openリストにも Closeリストにも含まれていない場合 f(m)←f′（m) とし m を Openリストに追加する。このとき m の親を n として記録する。
        2. m が Openリストにある場合、f′（m)<f(m) であるなら、m を Open から削除し、f(m)←f′（m) に置き換え、再び Open に挿入する（Open が正しくソートされていることを保証するため）。また、記録してある m の親を n に置き換える。
        3. m が Closeリストにある場合、f′（m)<f(m) であるなら f(m)←f′（m) として m を Openリストに移動する。また記録してある m の親を n に置き換える。
8. 3 以降を繰り返す。
9. 探索終了後、発見されたゴール nG から親を順次たどっていくと S から ゴール nG までの最短経路が得られる。

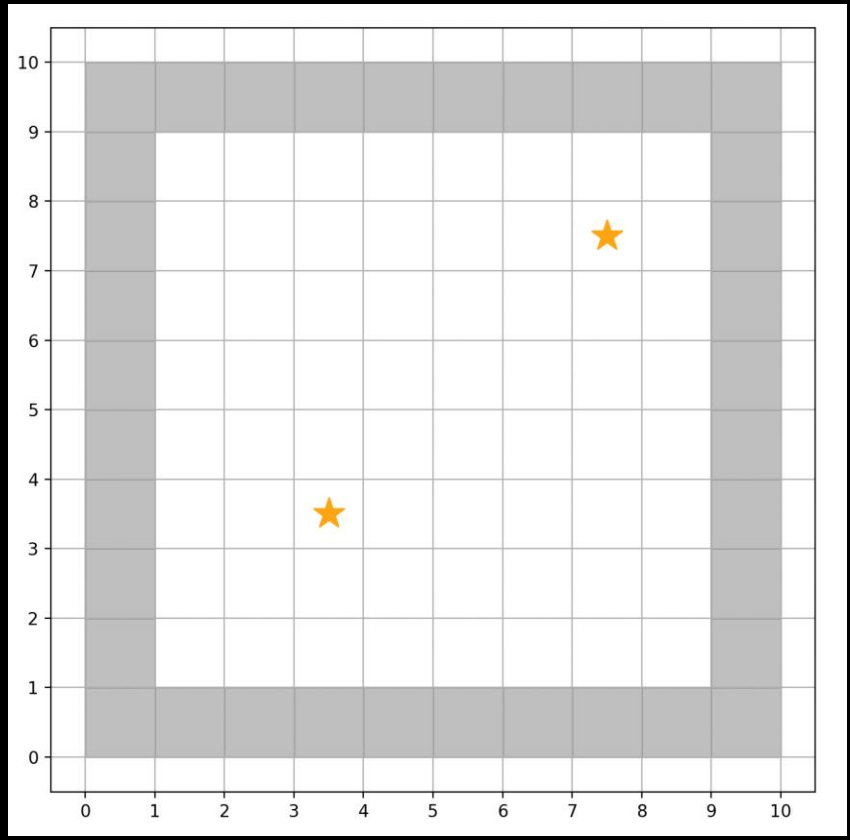
出典：https://ja.wikipedia.org/wiki/A*

A*アルゴリズム詳解

2点間の配線

A*アルゴリズム詳解 1/13

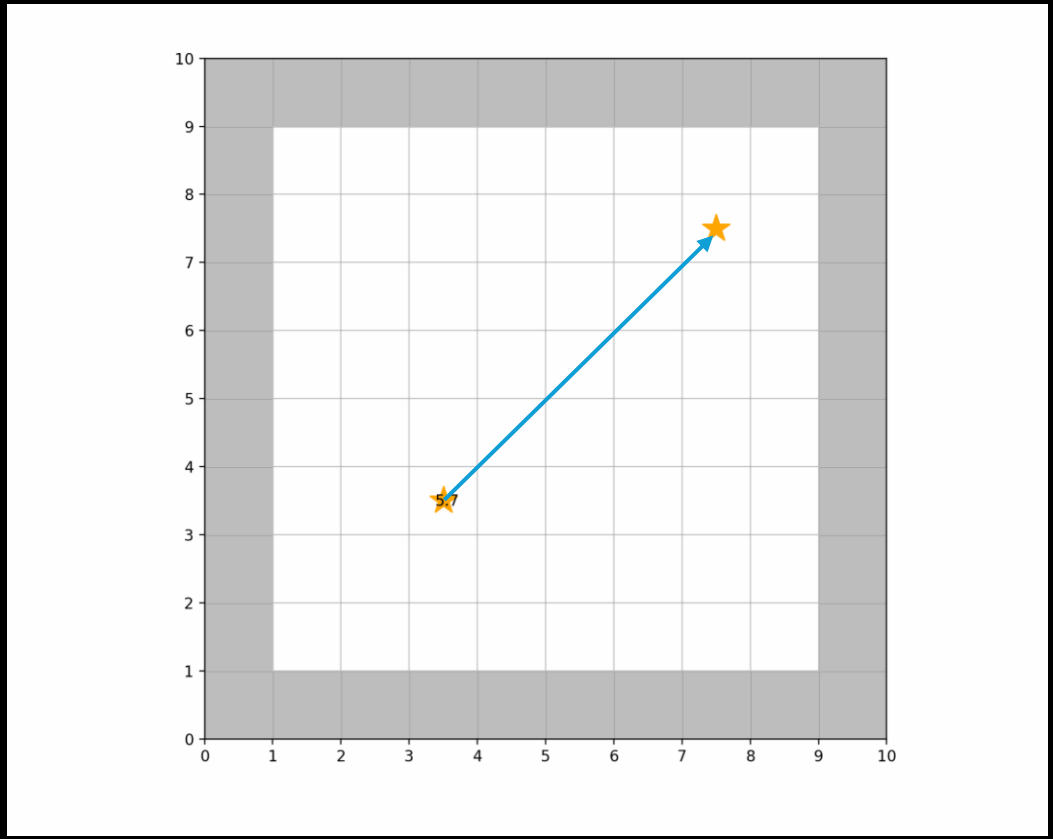
- 仕様
  - 10x10のグリッド
  - 左下の星が始点
  - 右上の星が終点



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 2/13

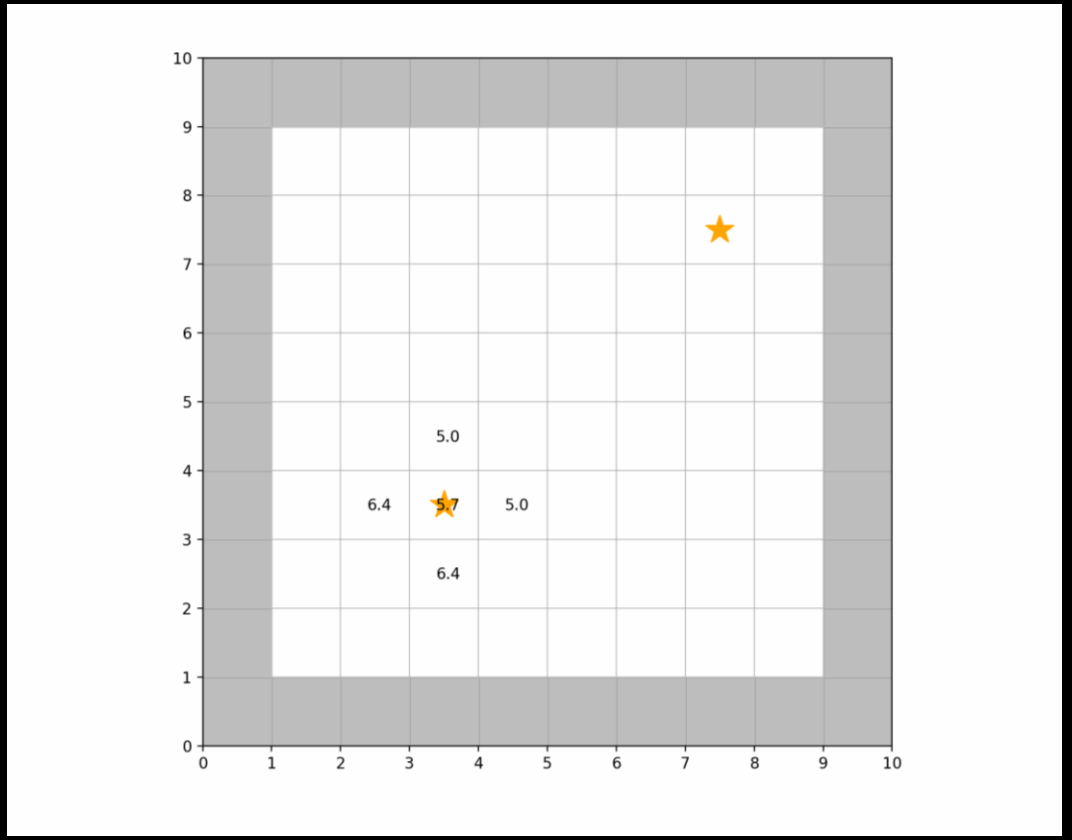
- 始点から終点までの距離を計算する
  - 5.7
    - ユークリッド距離



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 2/13

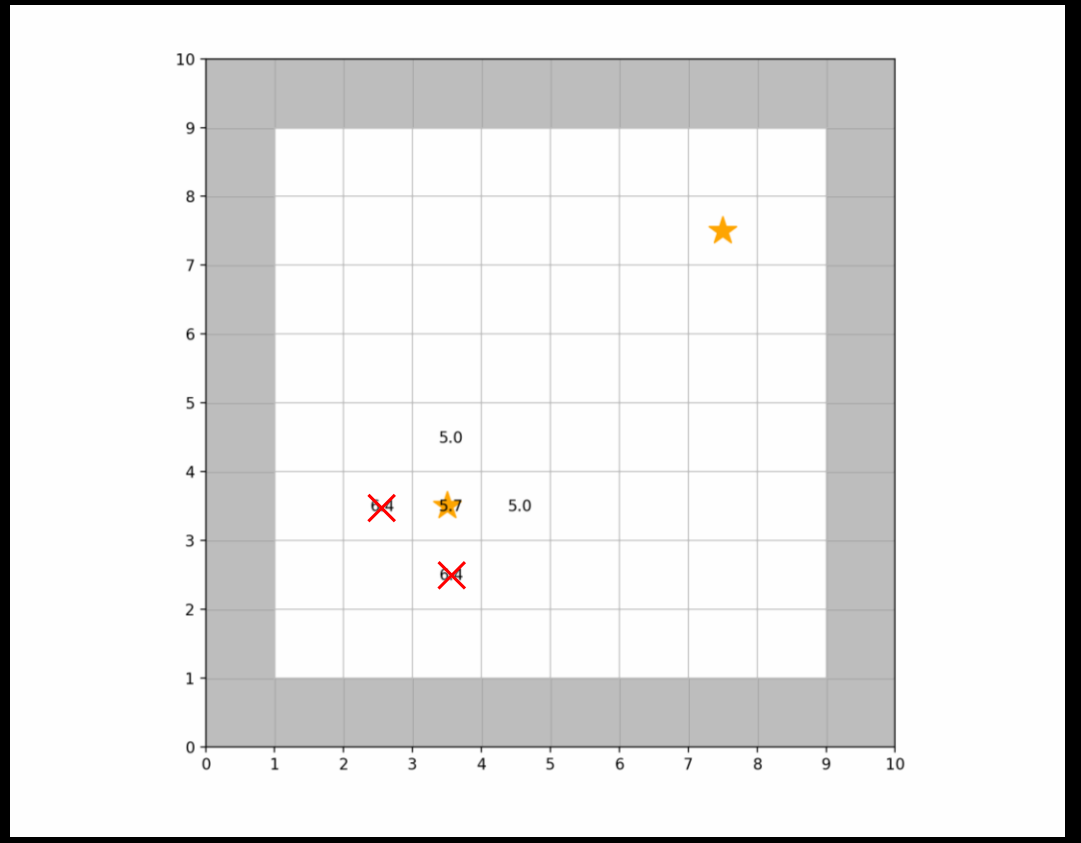
- 始点の隣接するノードの距離を計算する



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 4/13

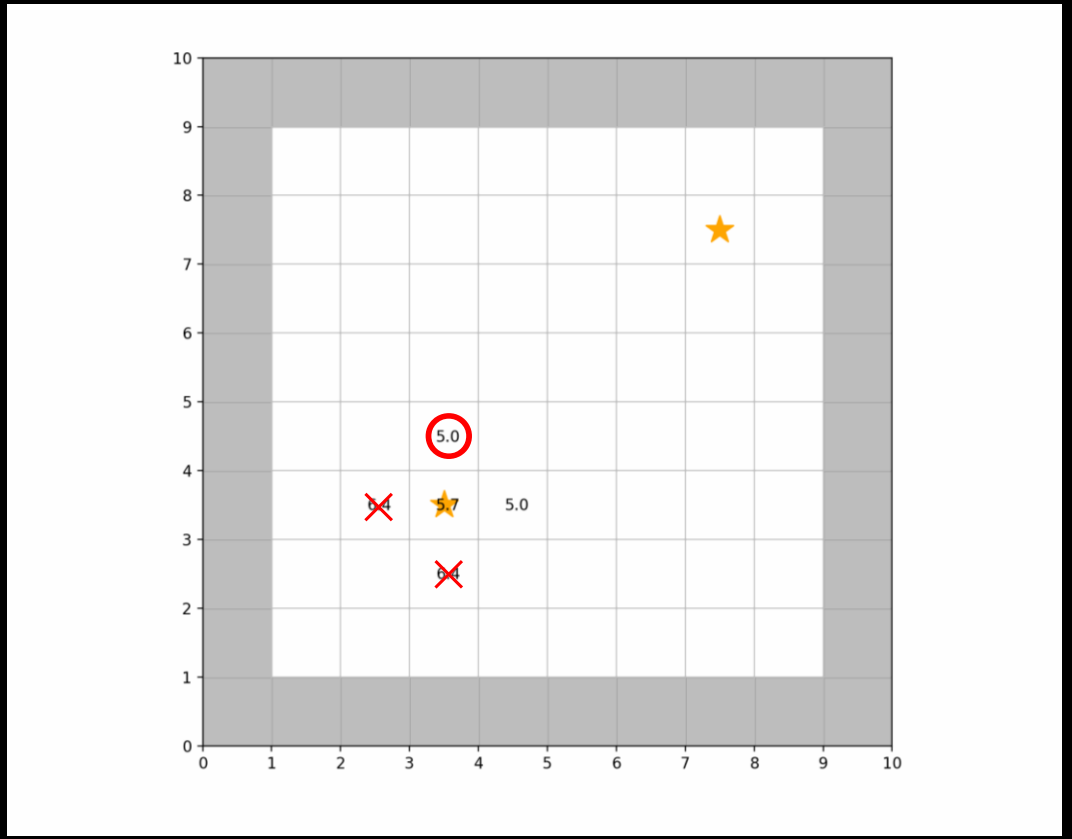
- 始点の隣接するノードで始点の距離より遠い（大きい）ノードは除外する



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 5/13

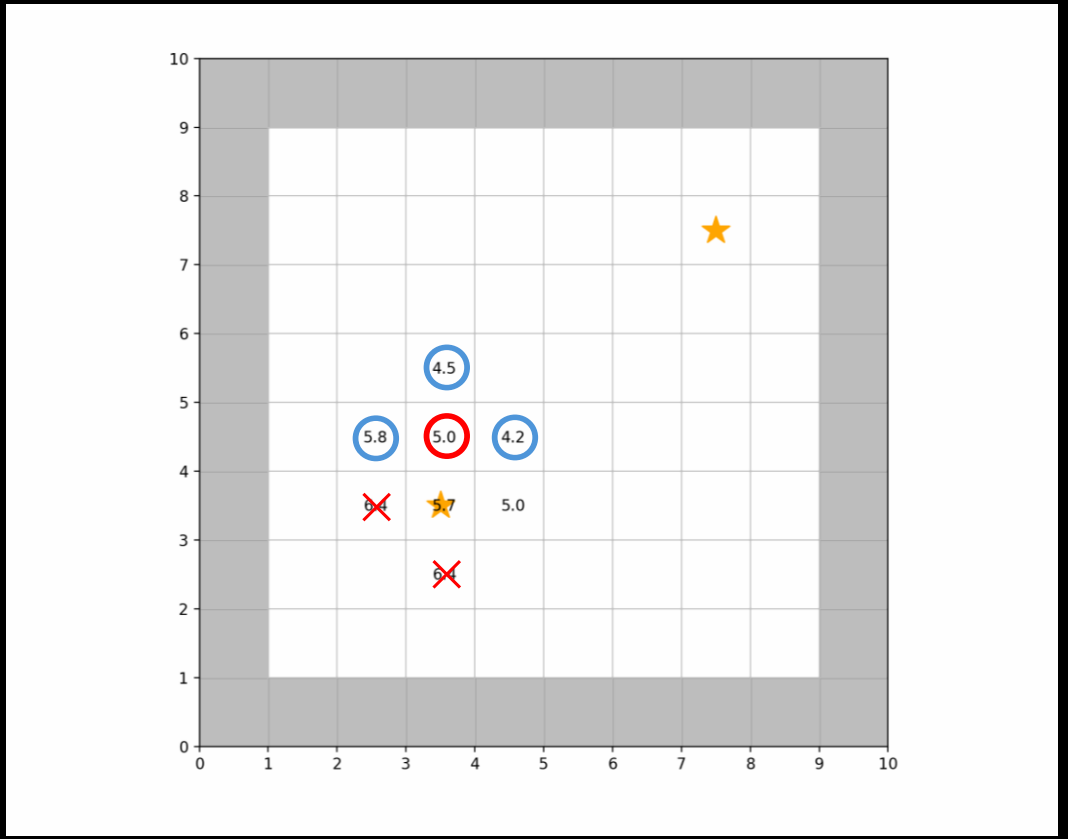
- 残ったノードに移動し、また、隣接するノードの距離を計算する



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 6/13

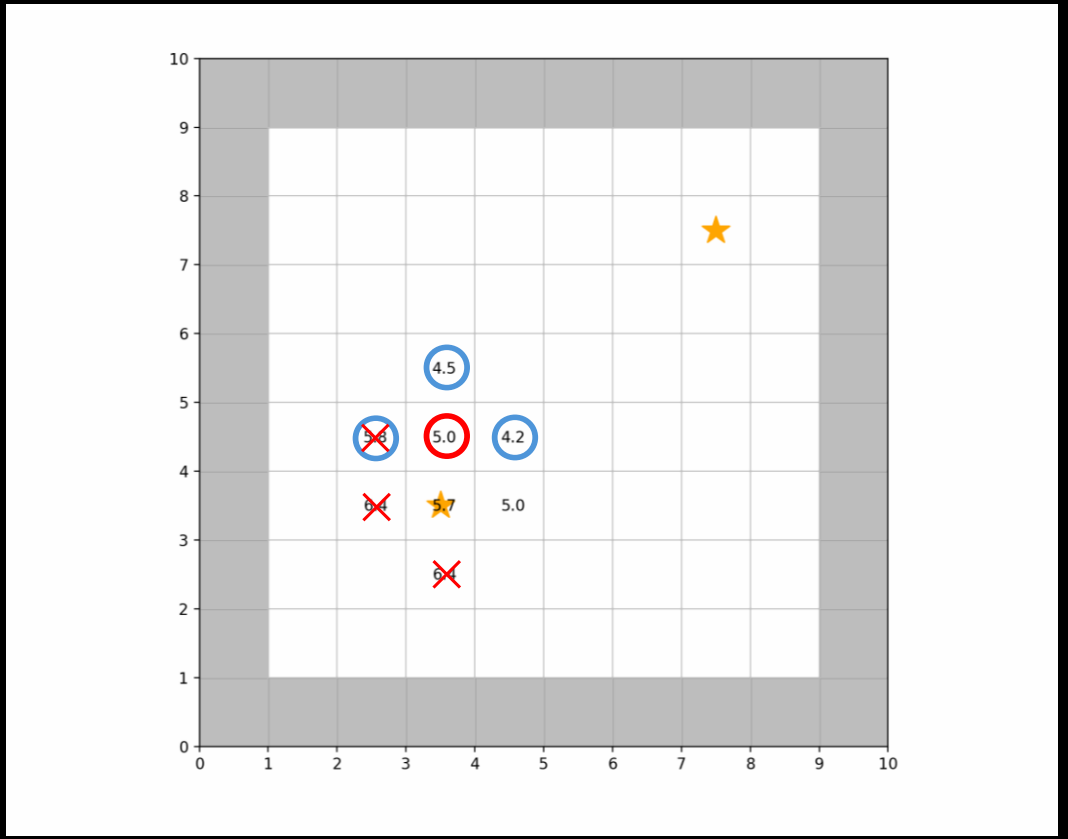
- 青丸が新しく距離を計算されたノード



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 7/13

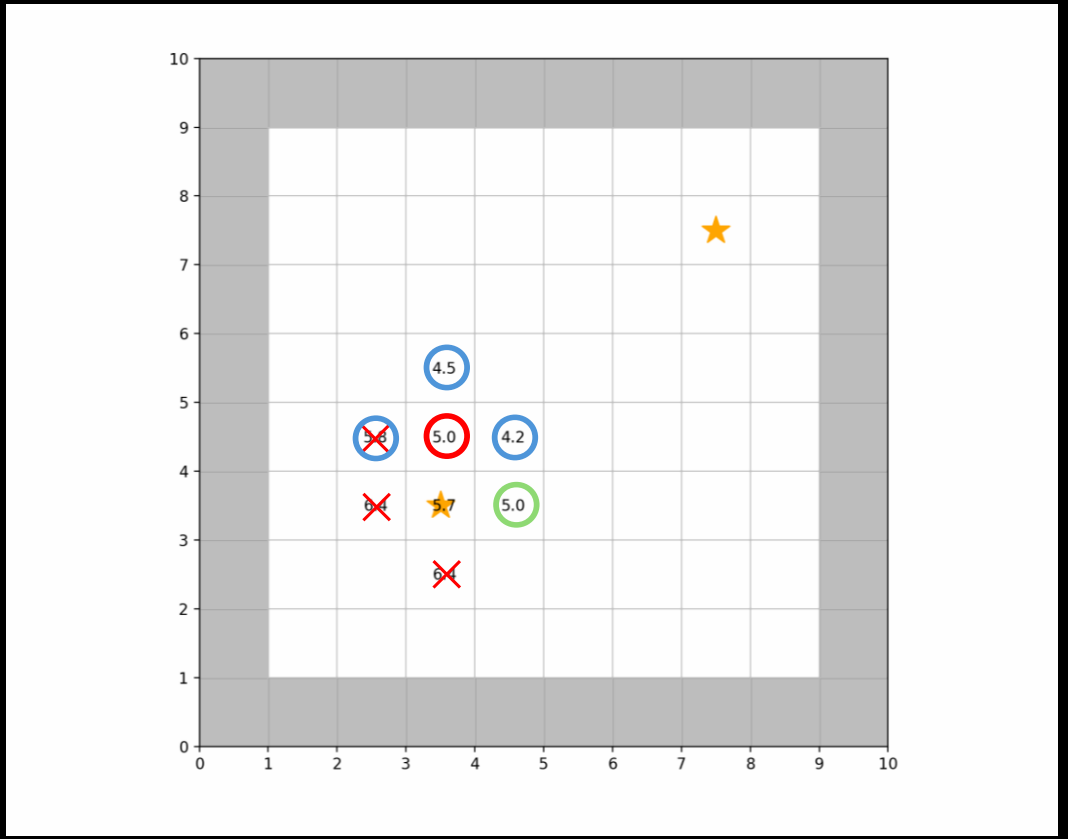
- 青丸のノードの中で赤丸より大きいノードを除外する



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 8/13

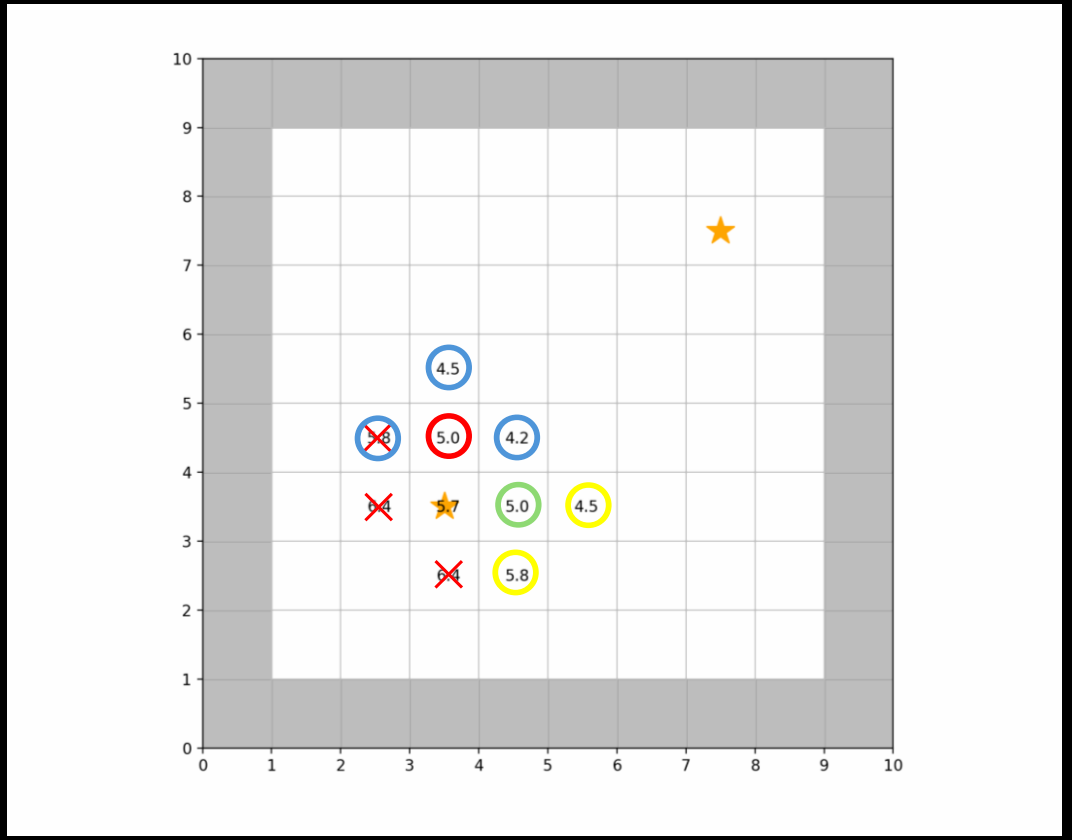
- まだ、計算されていないノードの隣接するノードの距離を計算する
  - この場合は、緑丸のノード



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 9/13

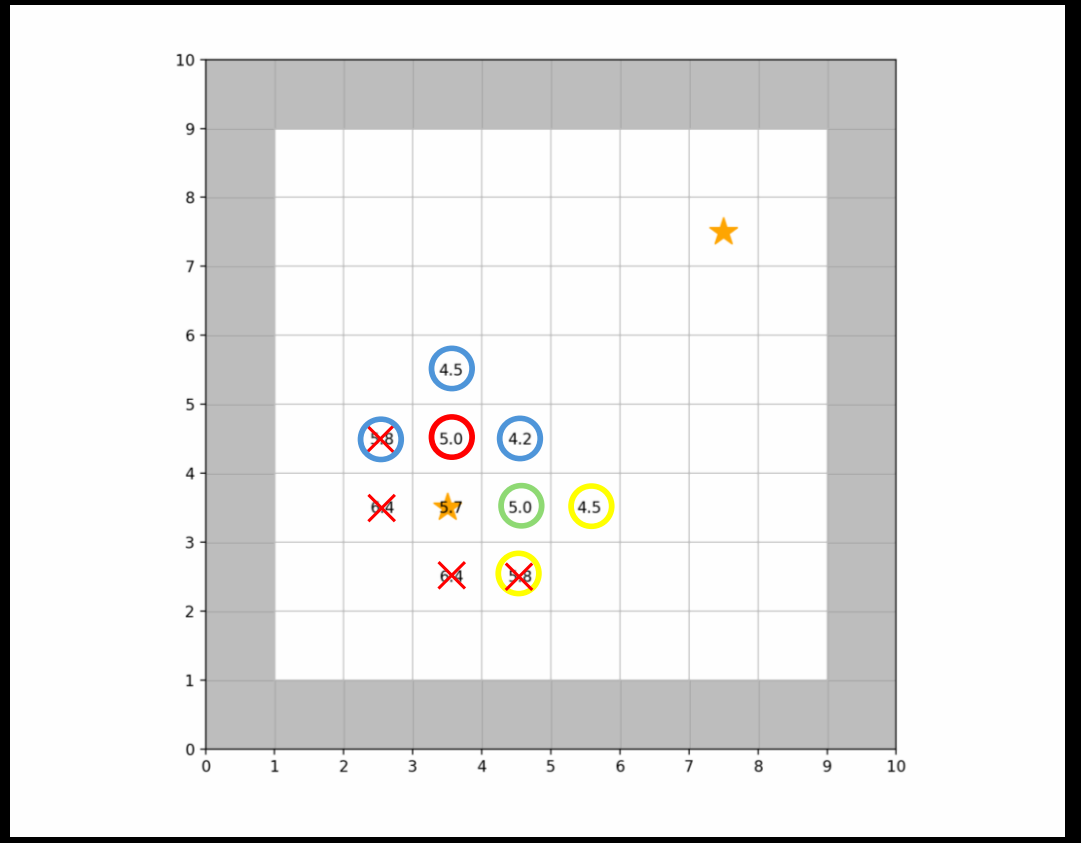
- 黄色丸が新しく距離を計算されたノード



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 10/13

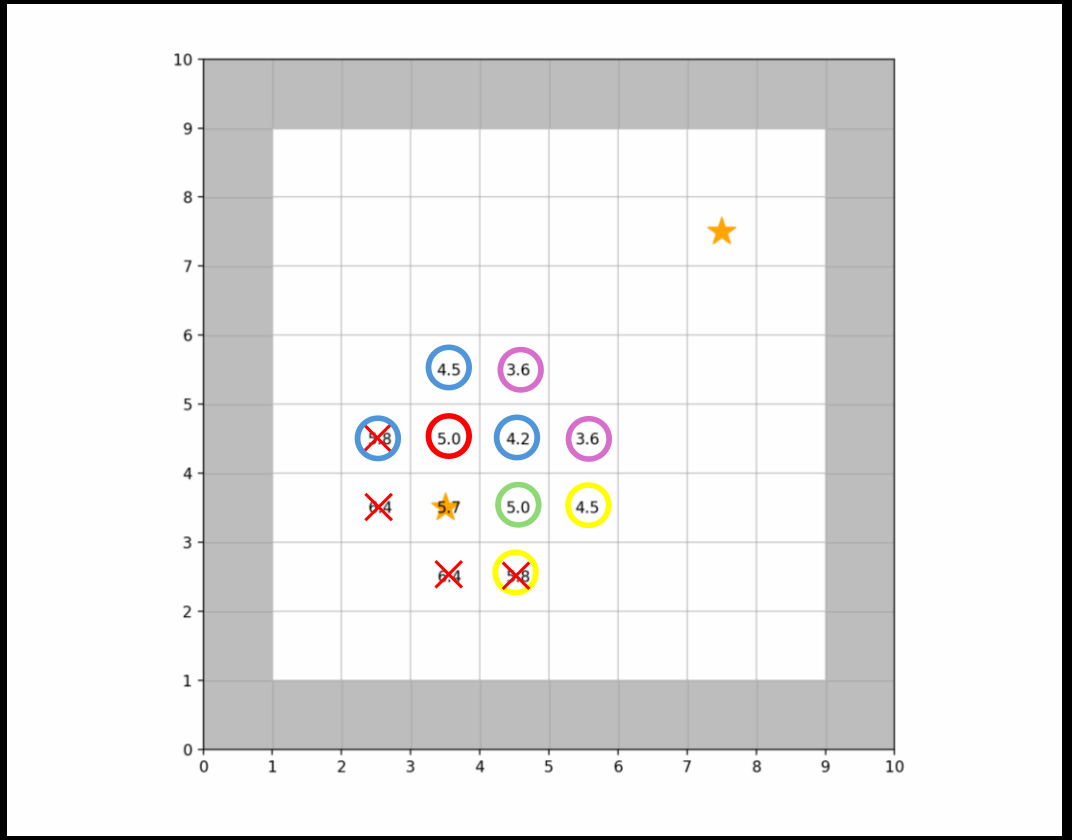
- 黄色丸のノードの中で緑丸より大きいノードを除外する



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 11/13

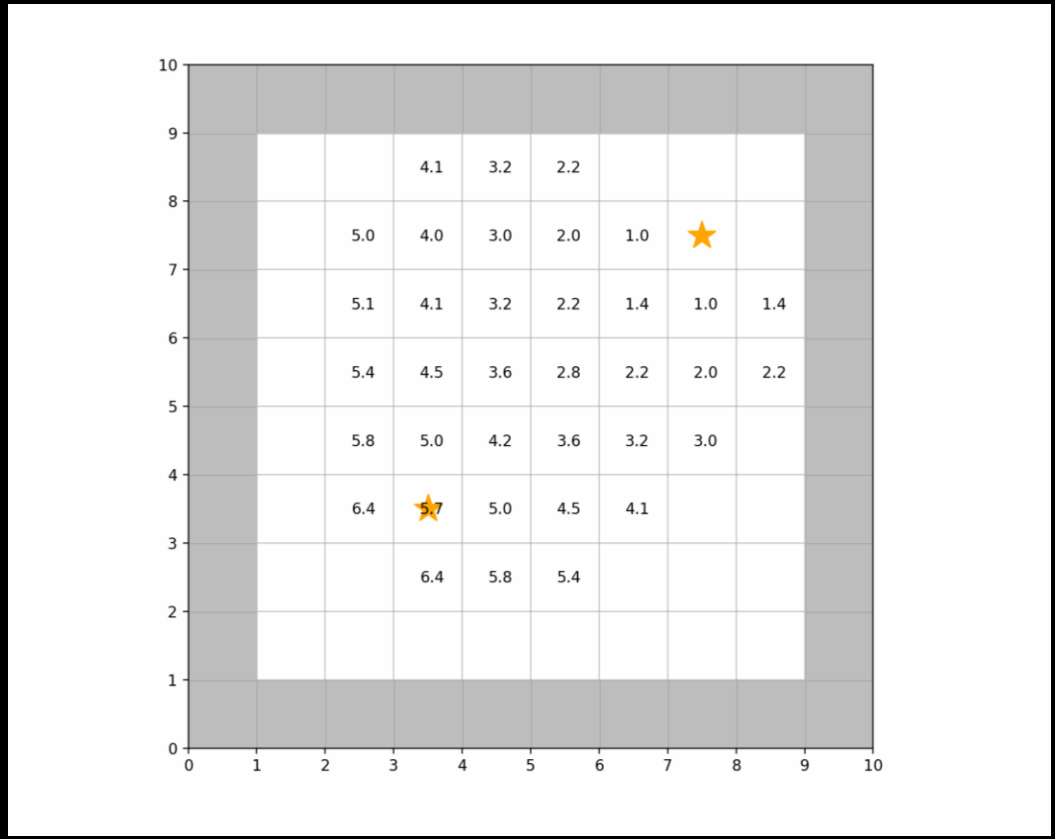
- まだ、計算されていない4.2の青丸のノードの隣接するノードの距離を計算する
  - この場合は、紫丸のノード



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 12/13

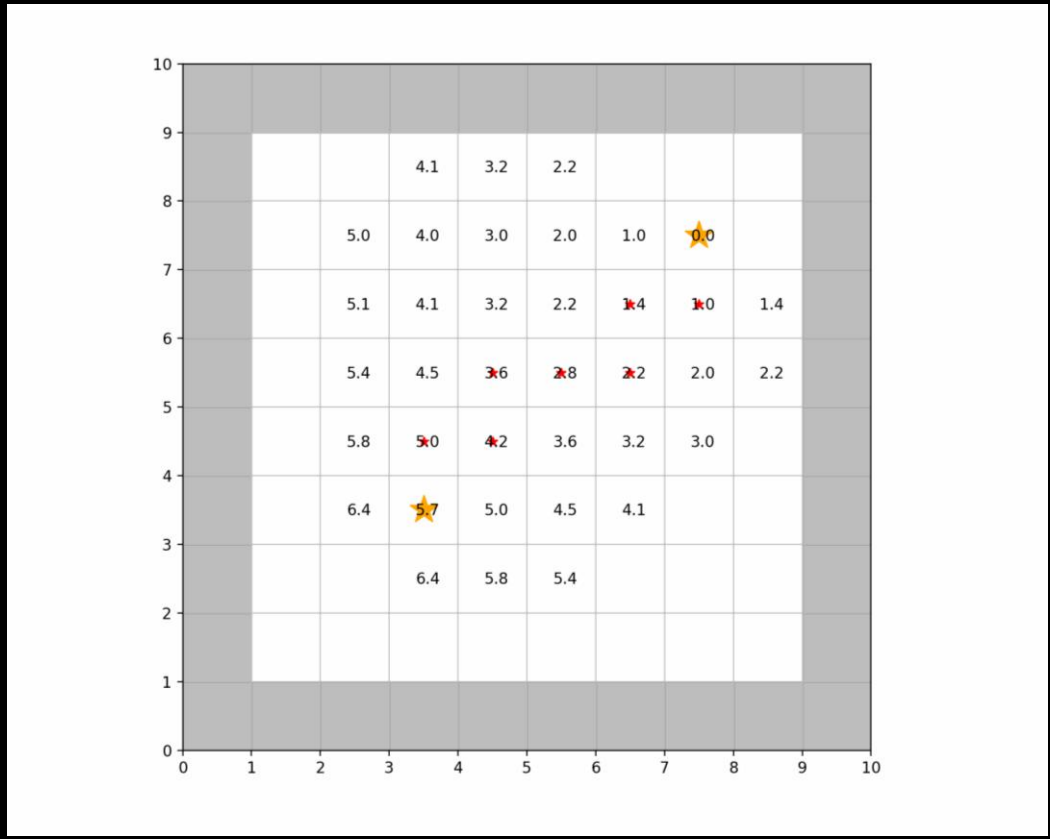
- これまでの手順を終点まで繰り返す



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 13/13

- 最小距離のノードを選択してくと配線が完了



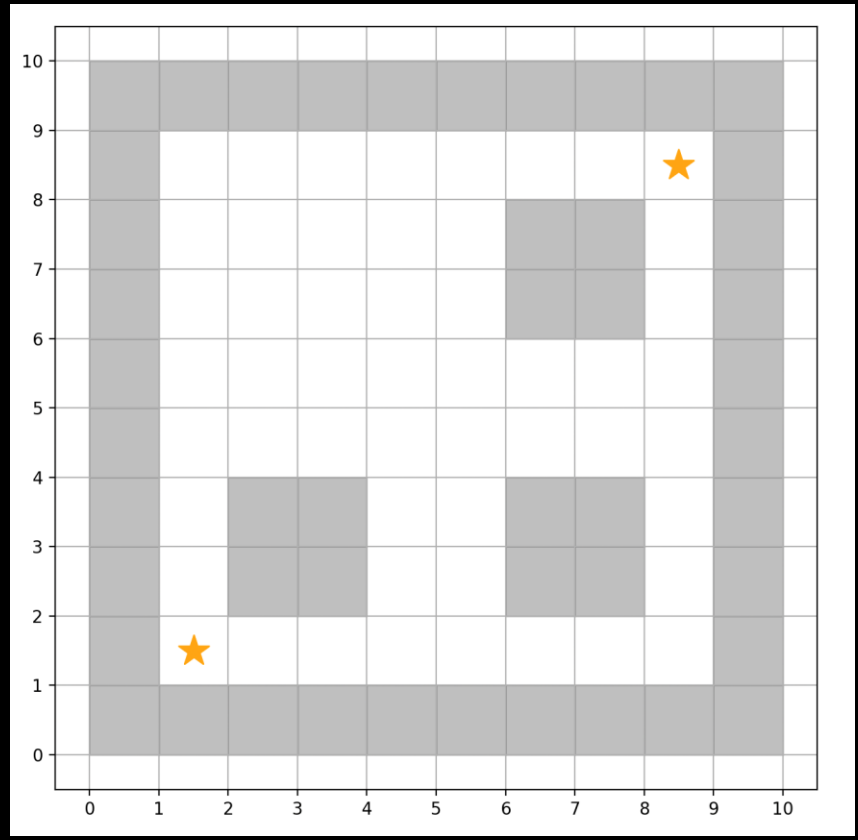
出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解

障害物がある場合

A*アルゴリズム詳解 1/9

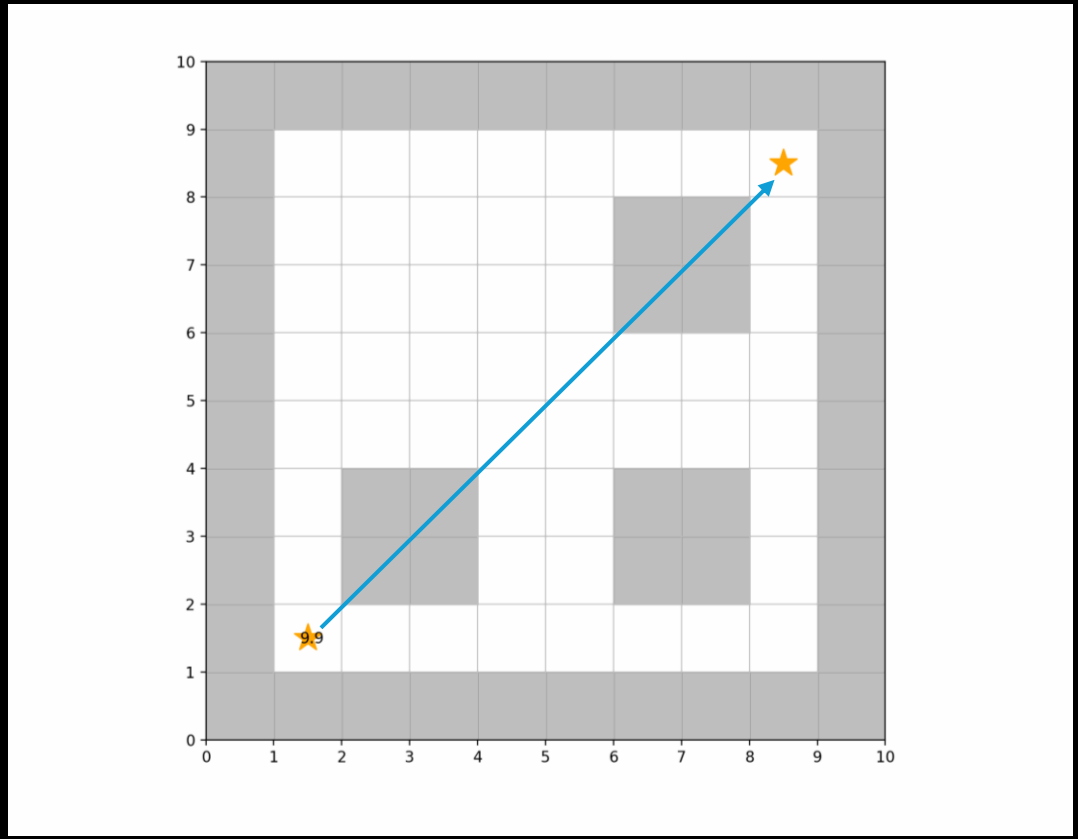
- 仕様
  - 3つの障害物があるパターン



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 2/9

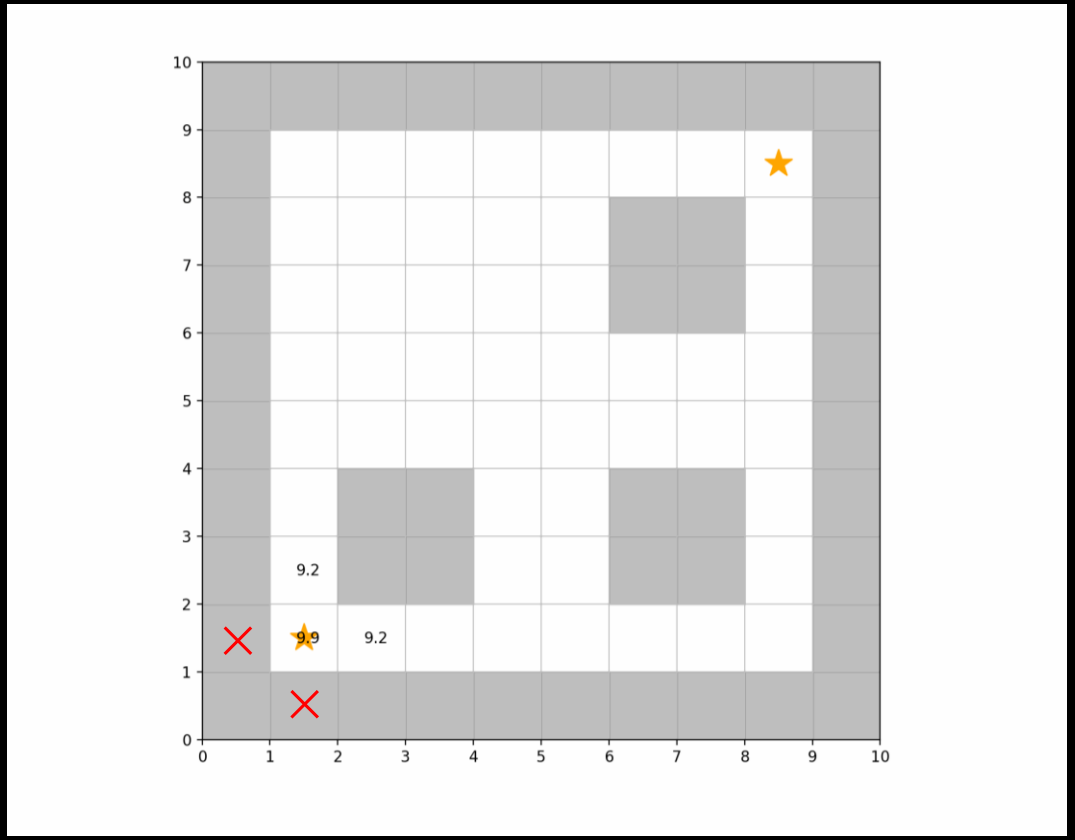
- 始点から終点までの距離を計算する
  - 9.9
    - ユークリッド距離



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 3/9

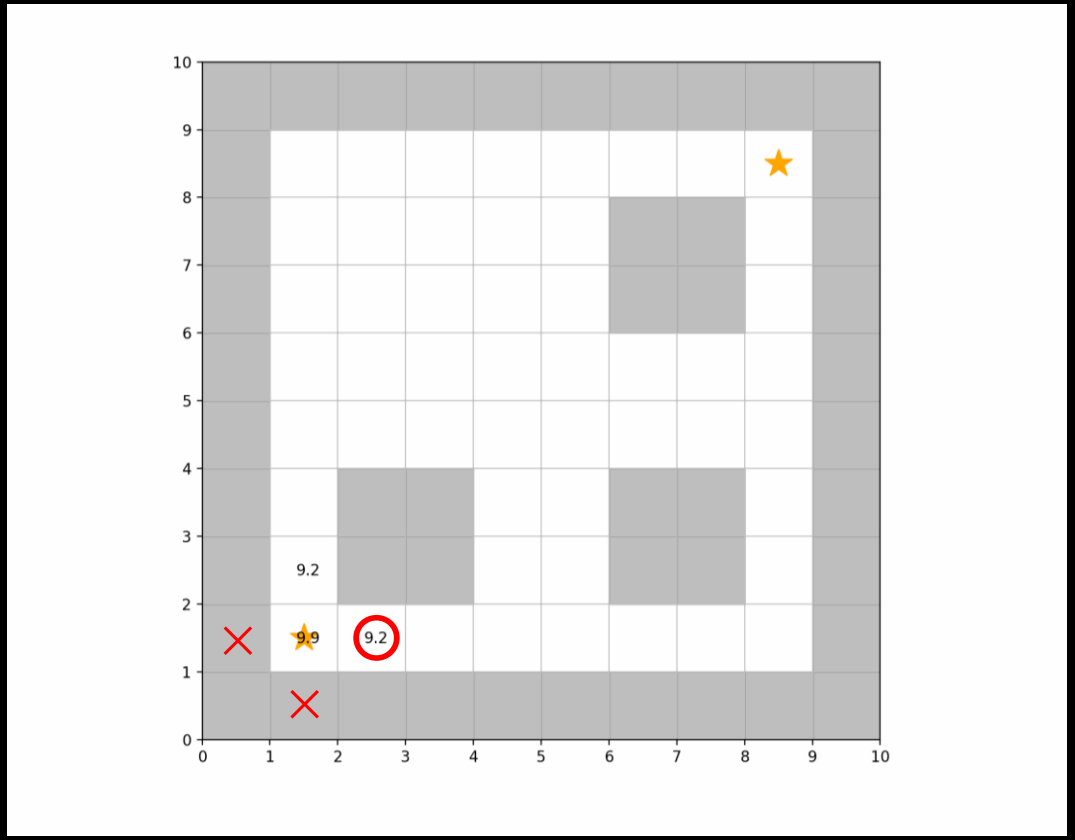
- 始点の隣接するノードの距離を計算する
  - 壁（障害物）のノードは除外する



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 4/9

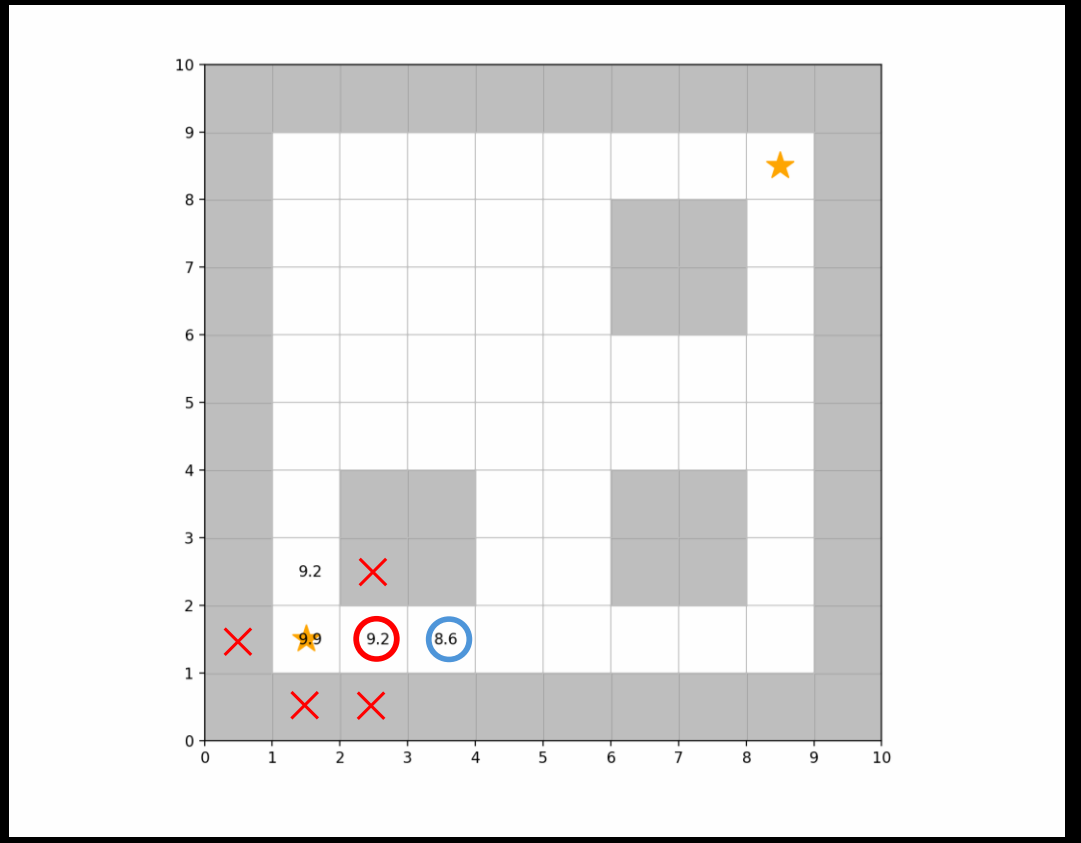
- 残ったノードに移動し、また、隣接するノードの距離を計算する



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 5/9

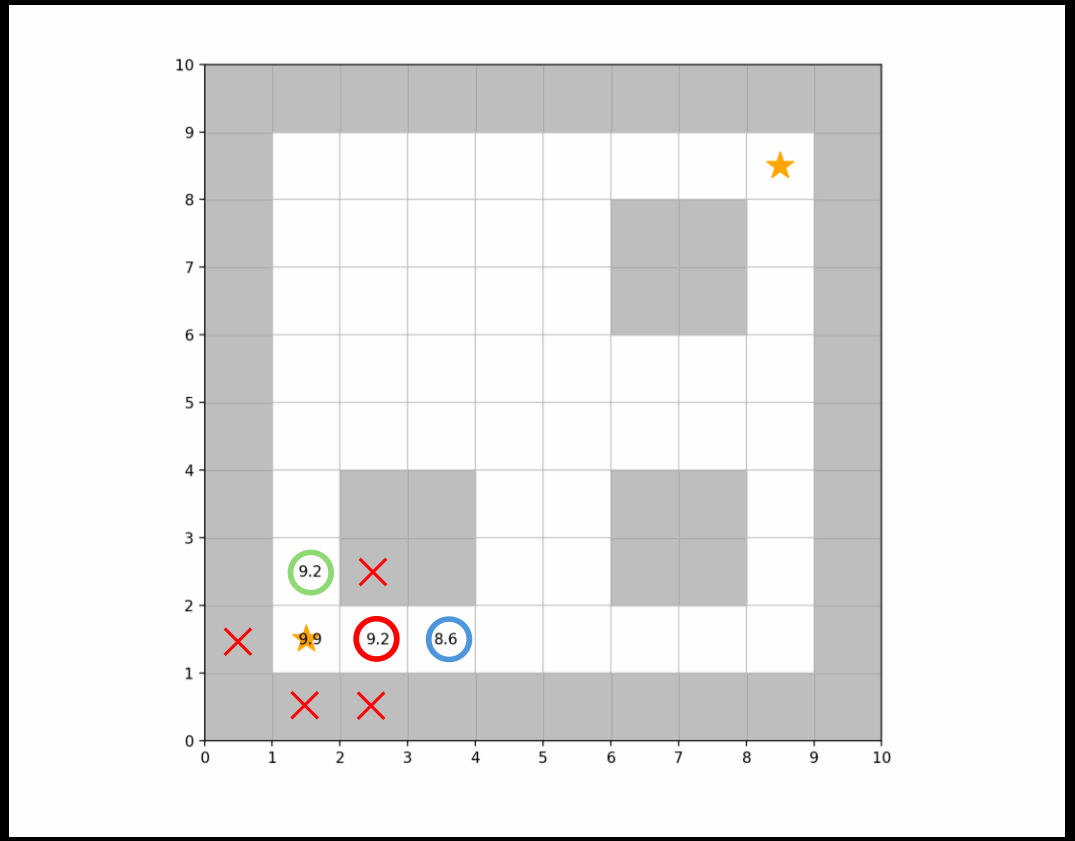
- 青丸が新しく距離を計算されたノード
  - 青丸のノードの中で赤丸より大きいノードを除外する
  - 壁（障害物）のノードは除外する



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 6/9

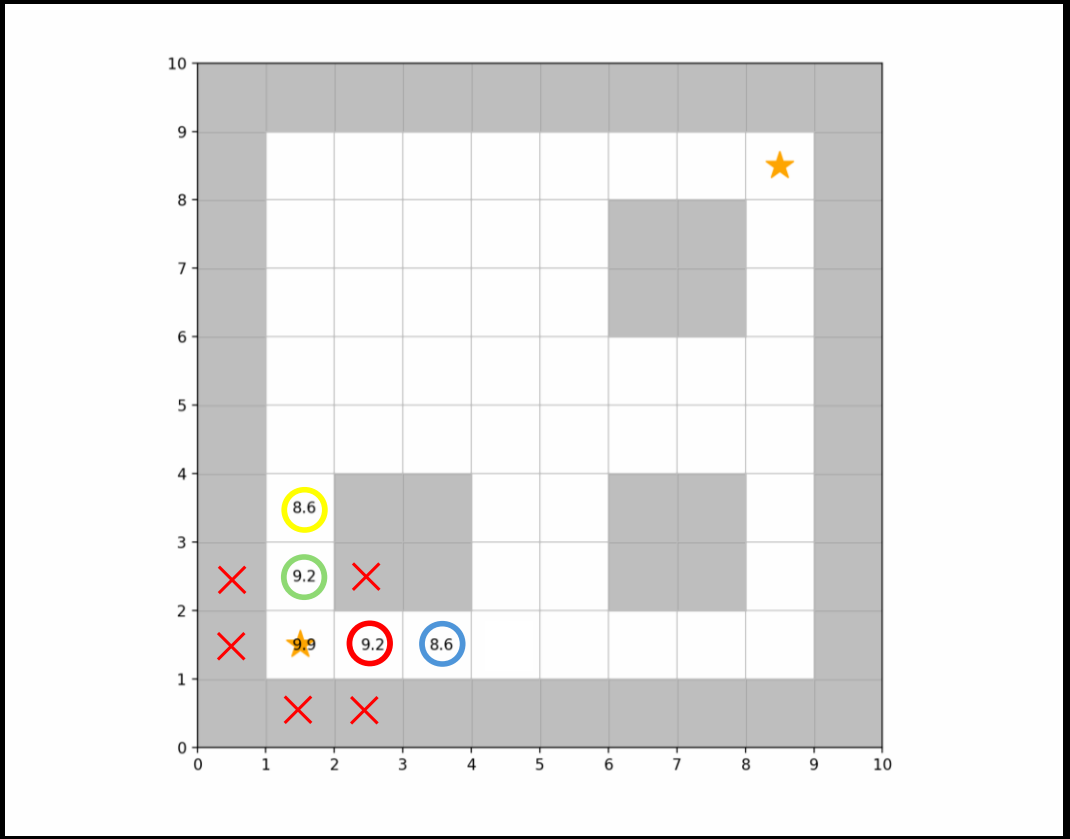
- まだ、計算されていないノードの隣接するノードの距離を計算する
  - この場合は、緑丸のノード



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 7/9

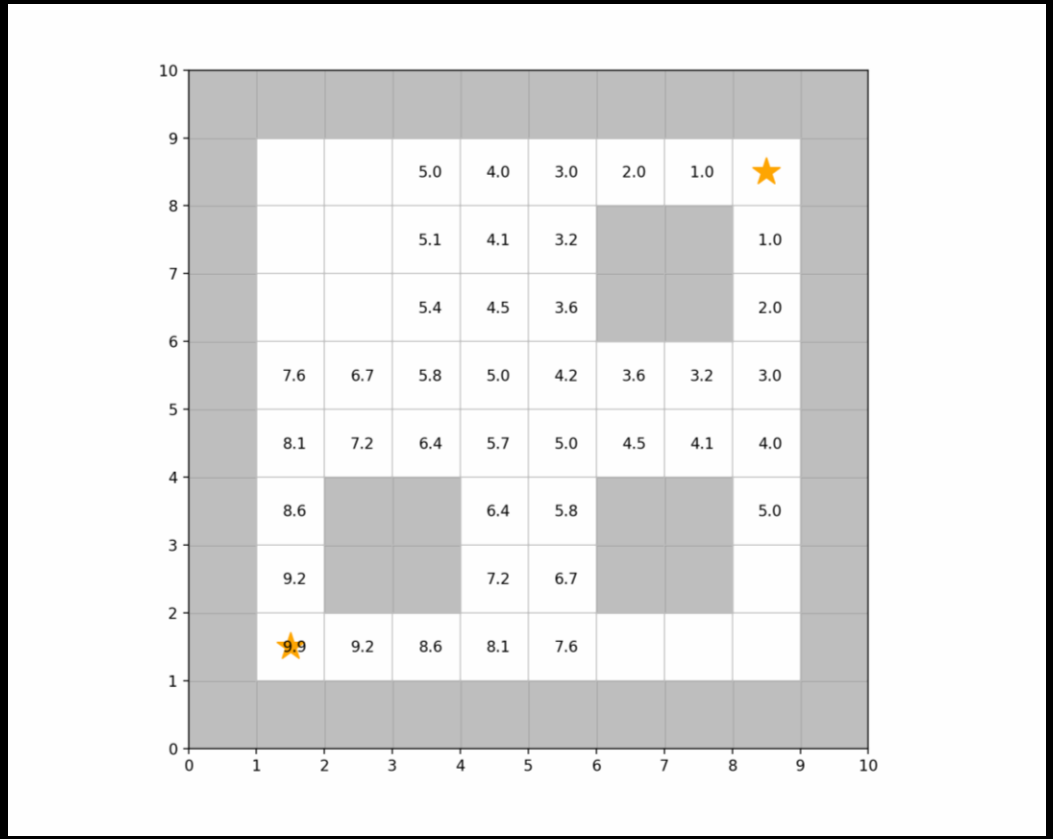
- 黄色丸が新しく距離を計算されたノード
  - 黄色丸のノードの中で緑丸より大きいノードを除外する
  - 壁（障害物）のノードは除外する



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 8/9

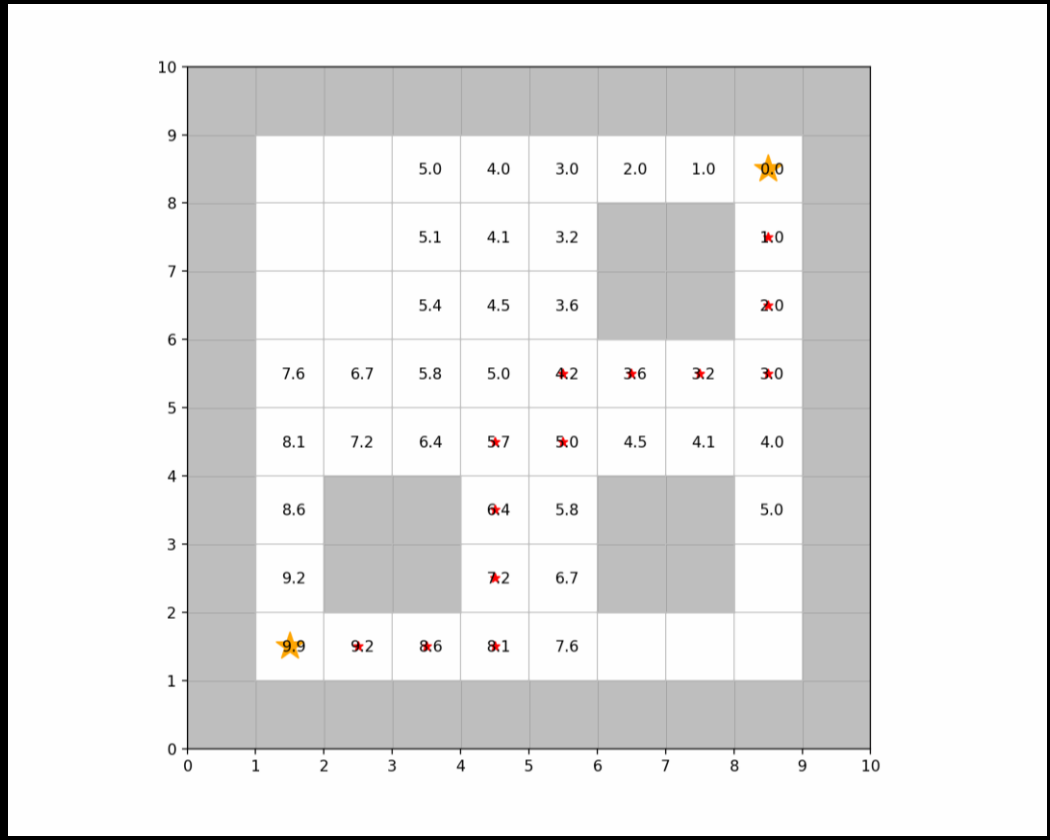
- これまでの手順を終点まで繰り返す



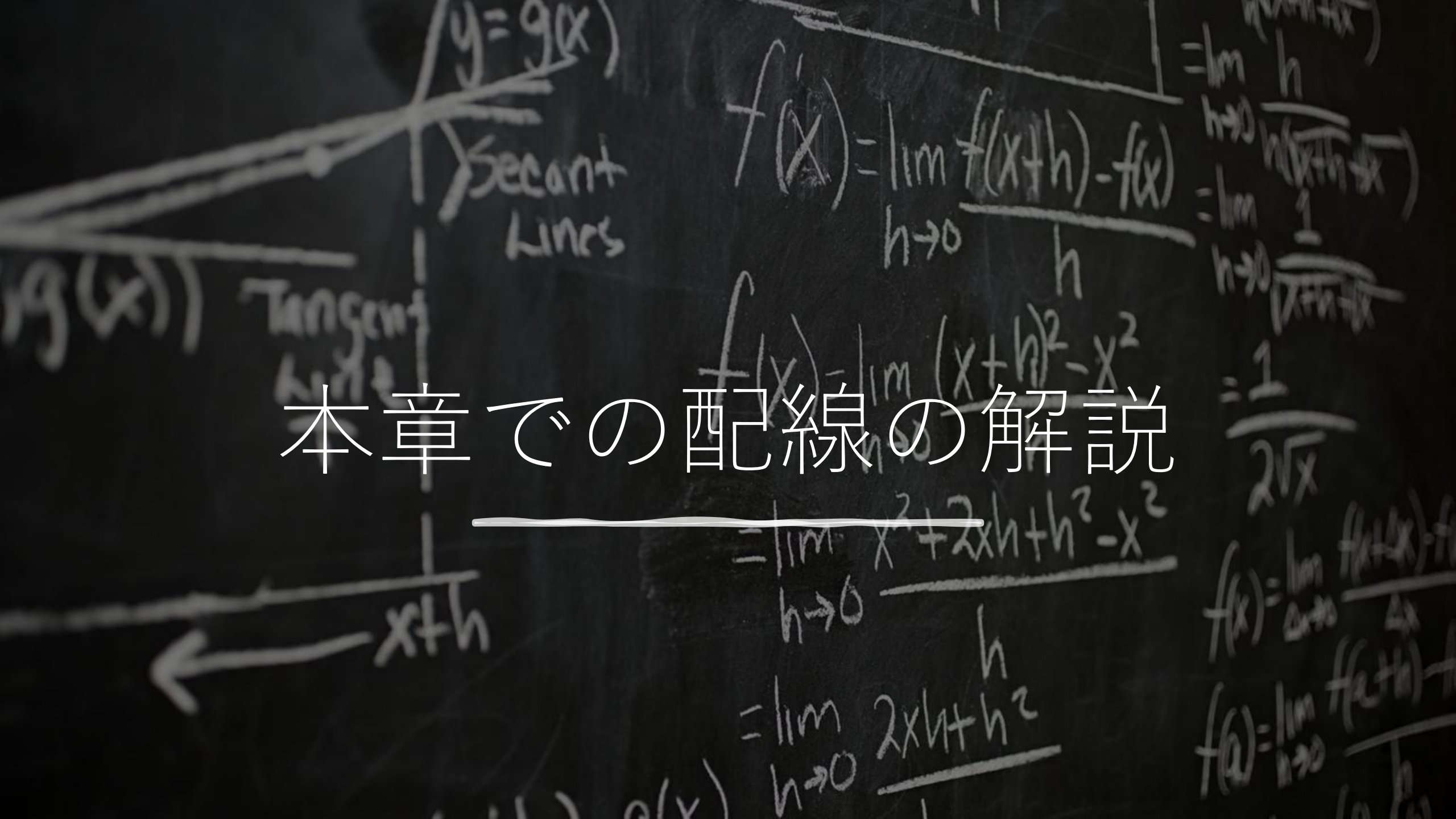
出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758

A*アルゴリズム詳解 9/9

- 最小距離のノードを選択してくと配線が完了



出典：https://qiita.com/Karin-Sugi/items/4e364a7263ee84575758


y=g(x)
Secant
Lines
Tangent
x+h
f(x)=lim h→0 f(x+h)-f(x)/h
=lim h→0 (x+h)²-x²/h
=lim h→0 x²+2xh+h²-x²/h
=lim h→0 2xh+h²/h
=1/2√x

本章での配線の解説

DUNE配線アルゴリズム

- FPGAの3次元集積化における配線問題を効率的に解決するために提案された研究
  - ダイオード制御による単方向配線ネットワーク:
    - 各配線セグメントにダイオードのような単方向のスイッチを導入します。これにより、信号の流れが一方向に制限されます。
      - 一見すると制約のように見えますが、この単方向性によって、配線リソースの競合を減らし、より効率的な配線が可能になります。
  - 柔軟な3D配線リソースの活用:
    - 単方向性を持つ配線セグメントとVIAを組み合わせることで、複雑な3次元的な配線経路を柔軟に構築できます。
  - 効率的な配線アルゴリズム:
    - これらのアルゴリズムは、単方向性の制約下で、グローバルな配線経路とローカルな詳細配線を効率的に決定することを目指します。

本章での
アルゴリズム

- 1st:混雑駆動型配線プランナー
  - 配線を集中させると「性能が落ちる」ことを考慮する
    - 配線不能に陥る
    - 寄生容量になる
      - など
- 2nd:グリッドレスな詳細配線ルーティング
  - 配線幅が可変なことを考慮する
    - 電源用の太い線
    - ダブルVIA
      - など

- これらの半導体要素をパラメータ（重み）として考慮したアルゴリズム

ソースコード概要

- ソースコードのリポジトリ
  - https://github.com/iic-jku/IIC-RALF/tree/main/Routing_v2
- 実行コマンド
  - python3 main_routing.py