11. パス、木、発見的探索

（人工知能）

URL: https://www.kkaneko.jp/ai/mi/index.html

金子邦彦

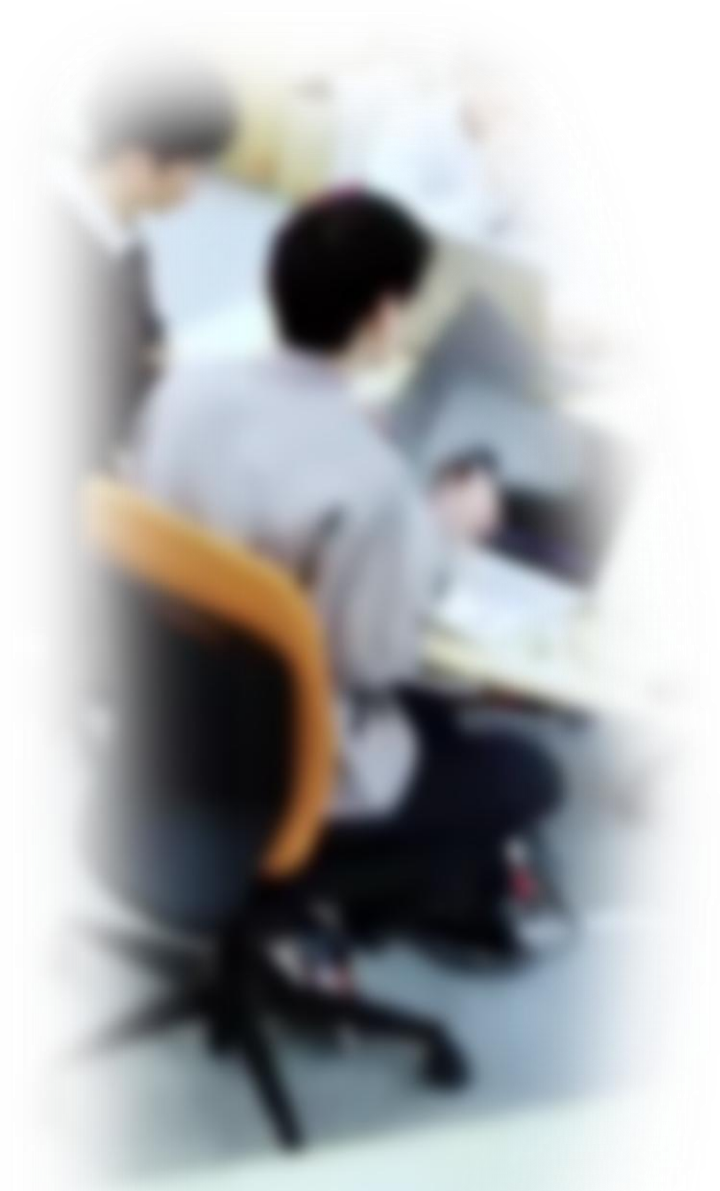
CC BY NC SA

①**パス**と**木構造**の概念を**アニメーションで視覚的に学び**、AIの基礎知識を習得

② **総当たり，単純探索**という**複数の探索手法**の特徴と違いを理解

③ **A*法**のような**発見的探索アルゴリズム**を学び、効率的な問題解決方法を身につける

④ trinketやVisuAlgoを使った演習で、AI理論を実装するスキルを向上



アウトライン

1. パスと木
2. グラフの中の木
3. グラフと全域木
4. 探索
5. A* 法（エイ・スター法）

trinket

- **Trinket** は**オンライン**の **Python、HTML** 等の**学習サイト**
- 有料の機能と無料の機能がある
- **自分が作成した Python プログラムを公開し、他の人に実行してもらうことが可能**（そのとき、書き替えて実行も可能）

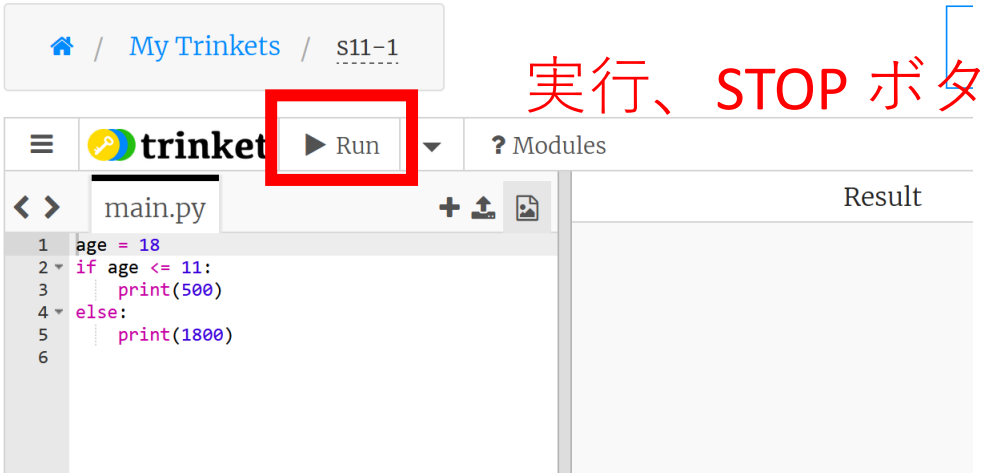
- **Python の標準機能**を登載、その他、次のパッケージがインストール済み

math, matplotlib.pyplot, numpy, operator, processing, pygal, random, re, string, time, turtle, urllib.request

trinket でのプログラム実行

- **trinket** は **Python, HTML などのプログラムを書き実行できる**サイト
- https://trinket.io/python/0fd59392c8

のように、違うプログラムには違う URL が割り当てられる

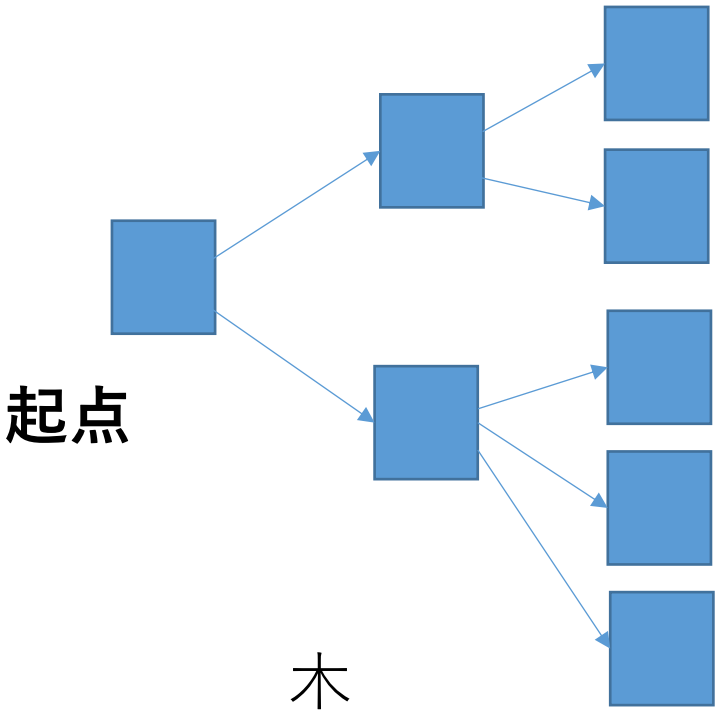


- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

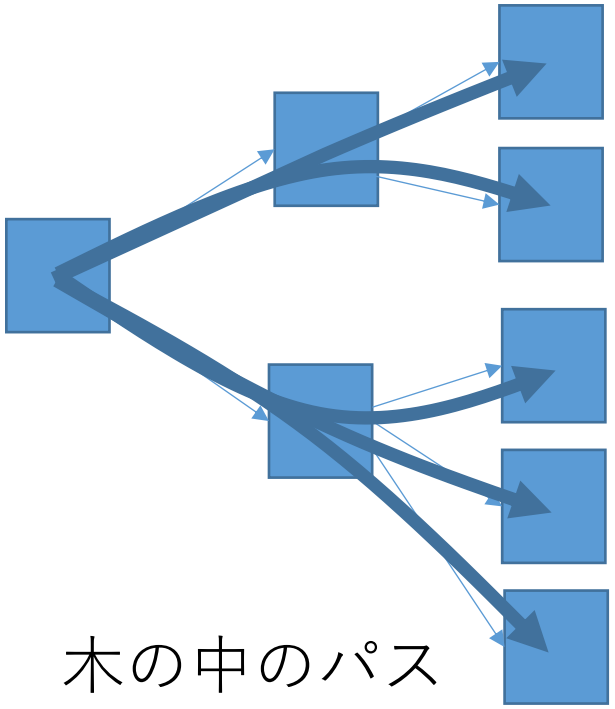
11-1 パスと木

パスと木

- **木**は、**共通の起点から**始まる**複数のパス（経路）**が**集まった**もの
- それぞれの**パス（経路）**は、**起点**から**各末端**へと向かって進むのが特徴。そして、**合流することはない**



木



木の中のパス

パスと木の用途

- **階層構造をもったデータを扱う**

  例えば、組織図、フォルダとファイルなど

- **最短経路問題**

  最も効率的な経路を求める問題

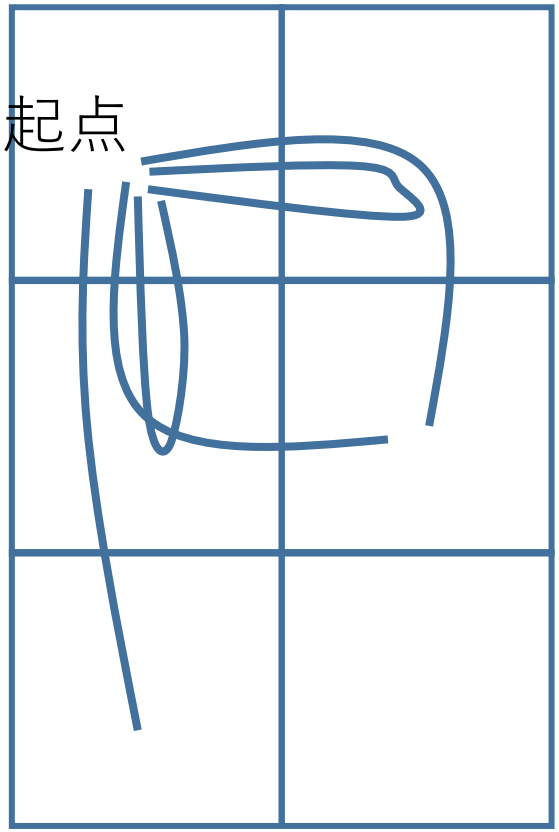
- **探索**

  人工知能の主な技術の１つで、**問題解決や意思決定に利用**。

  与えられた問題に対して、可能な解決策を調べ上げ、解を見つけ出す過程。

「**総当たり**」は**探索法の１つ**で、すべての可能性を網羅的に調べる方法。

総当たりの例　パス長２



総当たりの対象

- 一連の行動：**1, 2（右、上）**
- 一連の行動：**1, 3（右、下）**
- 一連の行動：**3, 1（下、右）**
- 一連の行動：**3, 3（下、下）**
- 一連の行動：**3, 4（下、上）**

それぞれが
経路（パス）

演習１

５つのパスをアニメーション表示。
ページ１０～１２

【トピックス】

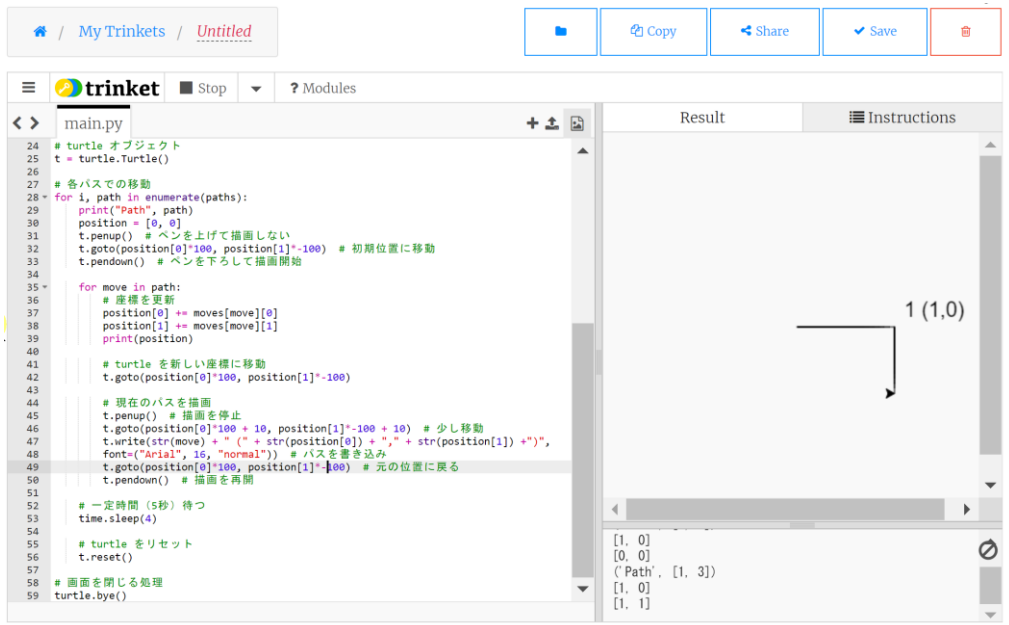
- trinketでのプログラム実行
- パス

① trinket の次のページを開く

https://trinket.io/python/912b1bb2e3

② 実行結果が，アニメーション表示される

- **5つのパスをアニメーション表示**で見ていく。
- **各パスが順番に表示**され、**起点からどのように進んでいくかを観察**してください。
- これにより、パスの概念と木構造の関係をより深く理解することができます。



プログラム実行結果

１．全てのパスは同じ起点から始まりましたか？

正解：はい

プログラム内で各パスの開始前に position = [0, 0] と設定されており、全てのパスが (0,0) から始まります。

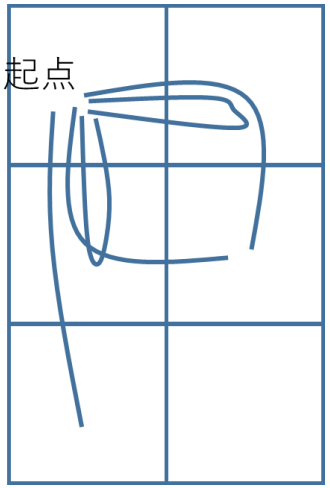
２．最も長いパスと最も短いパスはどれでしたか？

正解：全てのパスが同じ長さ（2ステップ）

paths リストを見ると、全てのパスが2つの移動で構成されています。そのため、全てのパスが同じ長さとなります。

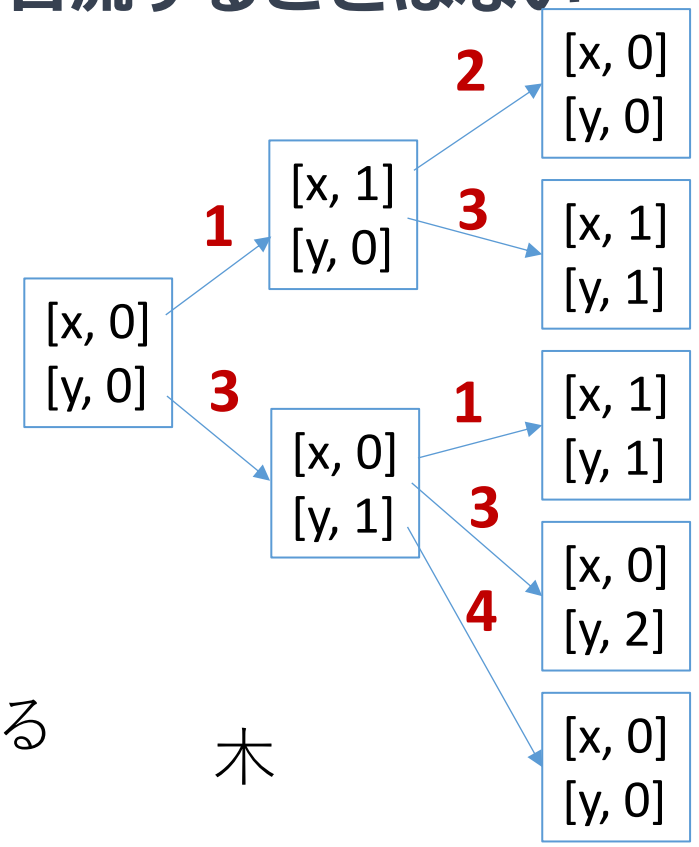
探索におけるパスや木の活用

- **共通の起点から**始まる**複数のパス**が存在するとき、これらの**パスの集まり**は「**木**」を形成
- **木**では、それぞれの**パス（経路）**は、**起点**から**各末端**へと向かって進むのが特徴。そして、**合流することはない**



・1, 2（右、上）
・1, 3（右、下）
・3, 1（下、右）
・3, 3（下、下）
・3, 4（下、上）

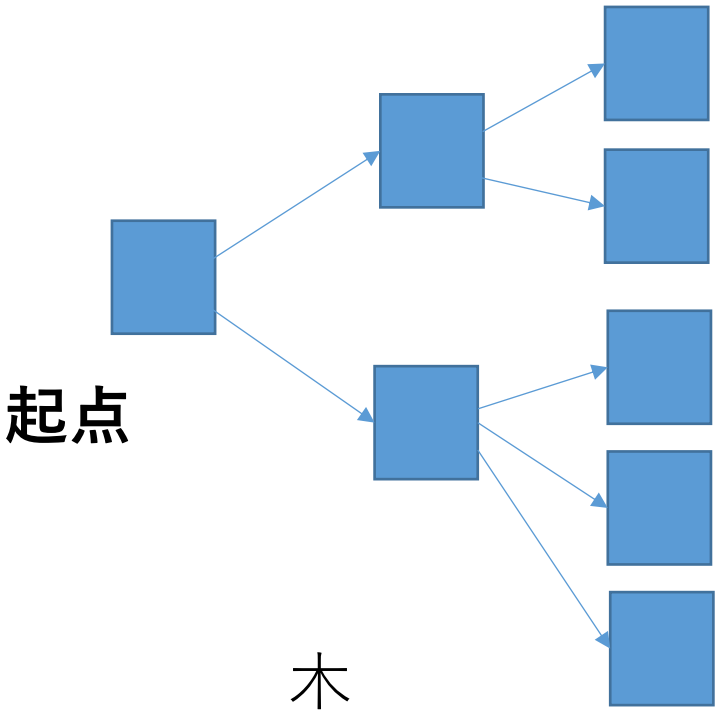
同じ起点から始まる
5つの経路



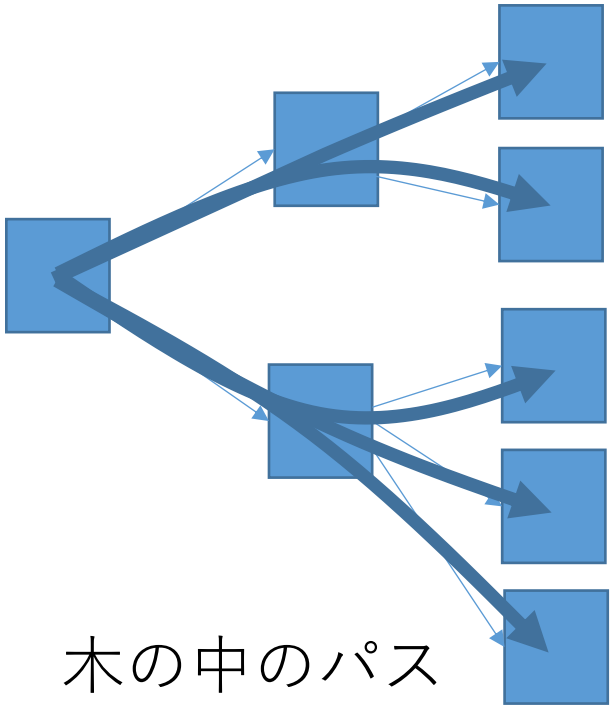
木

コンピュータによる探索で木を使うことのメリット

- コンピュータは、**木**の利用して探索を行う
- **一度試したパスを再び試すことなく、未探索のパスへと進む**こと（**パスを、効率的に１つずつ探索**すること）がようになる。



木



木の中のパス

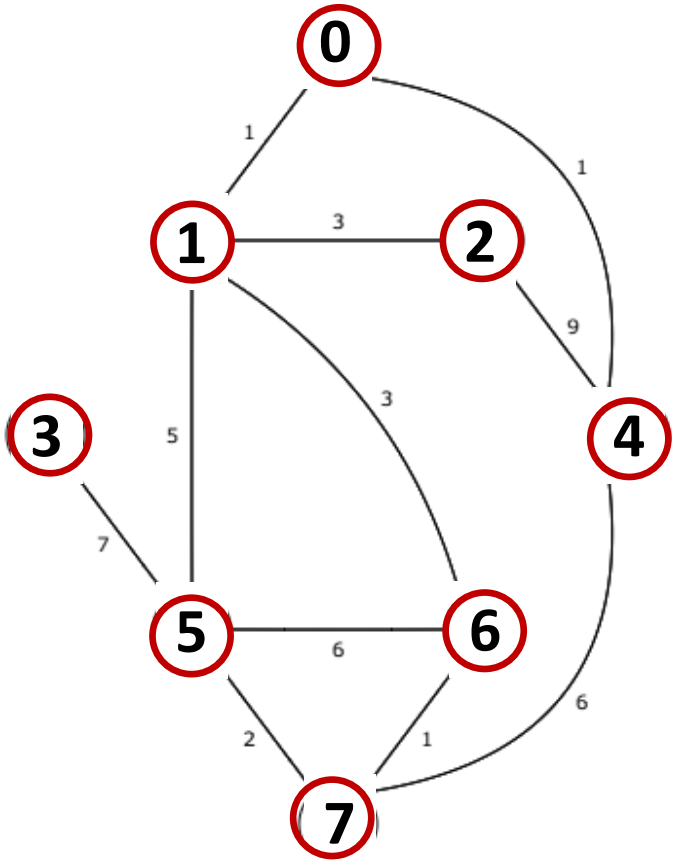
ここまでのまとめ

- **共通の起点から**始まる**複数のパス**が存在するとき、これらの**パスの集まり**は「**木**」を形成
- **木**では、それぞれの**パス（経路）**は、起点から各末端へと向かって進むのが特徴。そして、**合流することはない**
- **木**の利用により、**一度試したパスを再び試すことなく、未探索のパスへと進む**ことが容易になる

→　人工知能の重要な技術の一つである**探索**を**効率的に行うことが可能に**

11-2 グラフの中の木

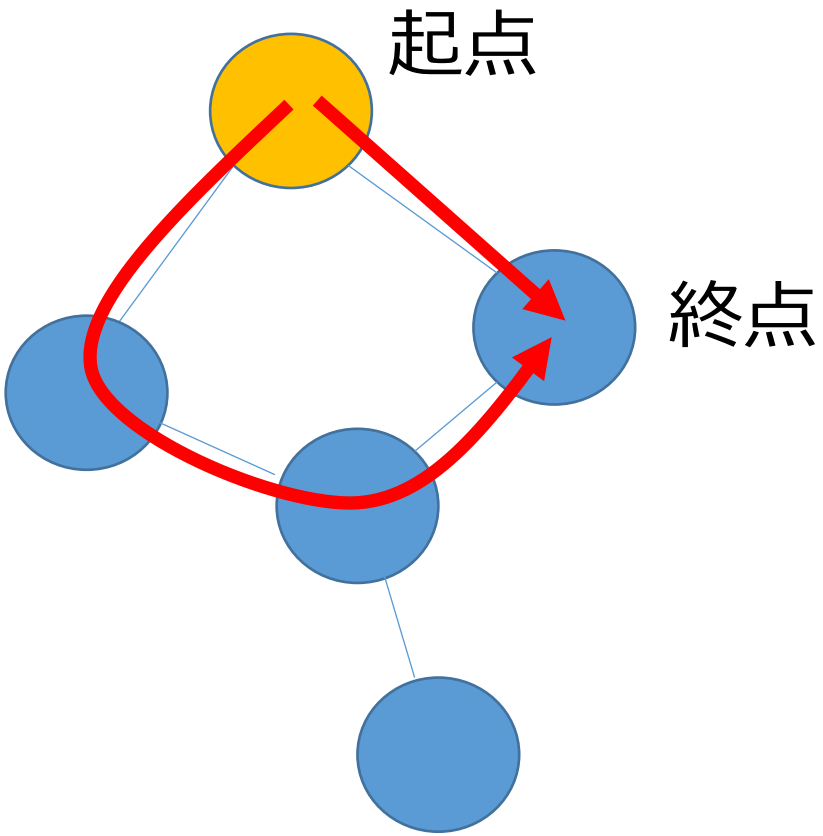
グラフ



○　ノード
線　エッジ

用途
・道路のつながり具合
・バス路線
など

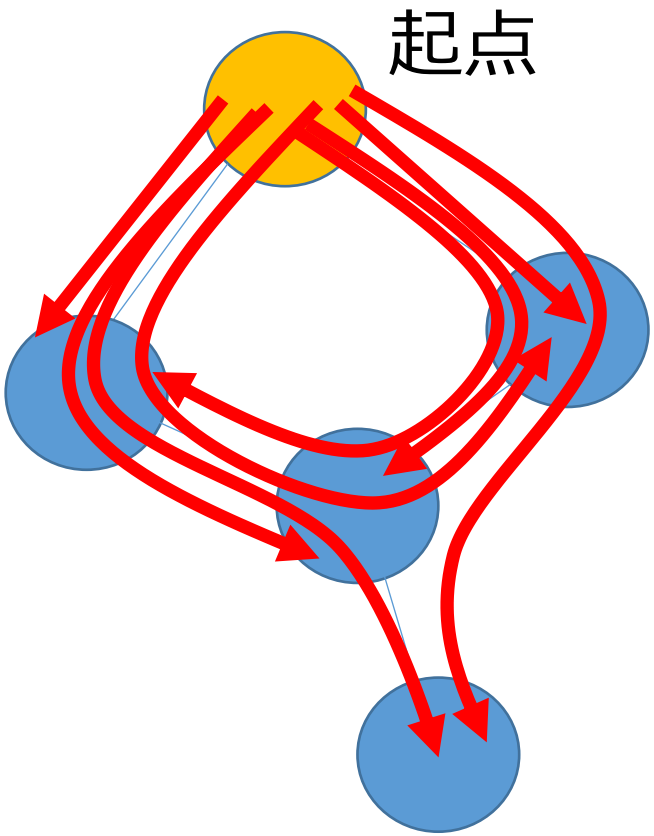
グラフとパス



○　ノード
線　エッジ

グラフの中で，
起点と**終点**を指定
→　**パスは複数あり得る**
（分岐と合流）

グラフとパス

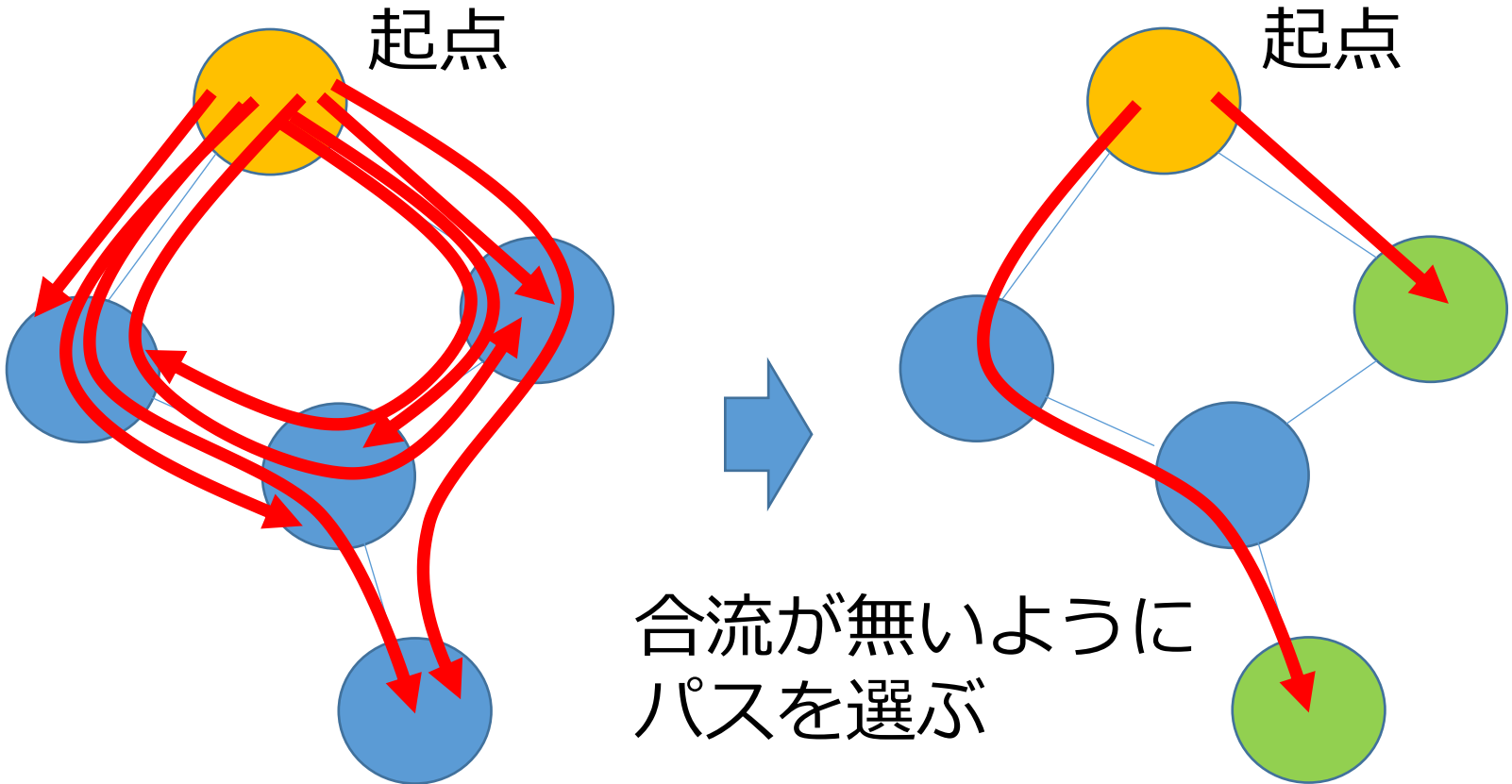


○　ノード
線　エッジ

グラフの中で，
起点を指定
→　**パスは複数あり得る**
（分岐と合流）

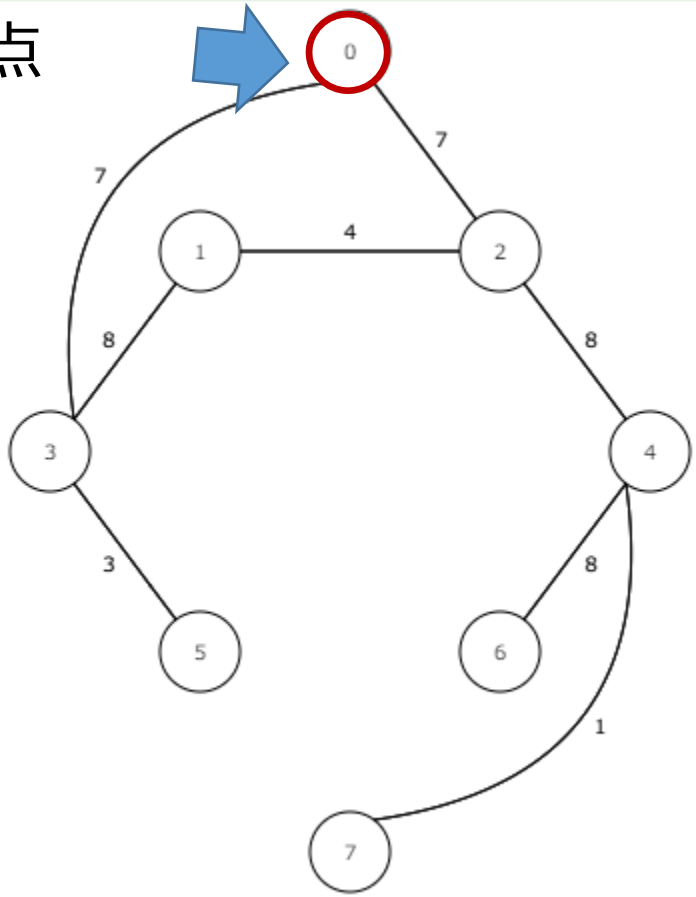
グラフの中の木

グラフの中で，**起点**を指定し，**木**を構成する



確認クイズ

起点



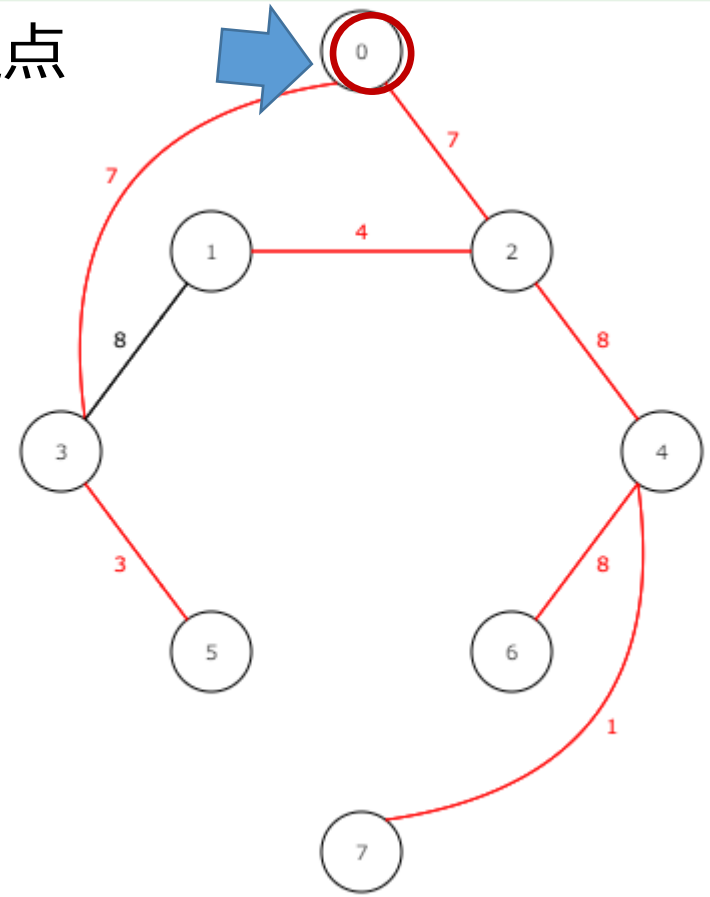
このグラフに

木を書き込んでみよう

※　正解は１つではない

確認クイズ　正解の例

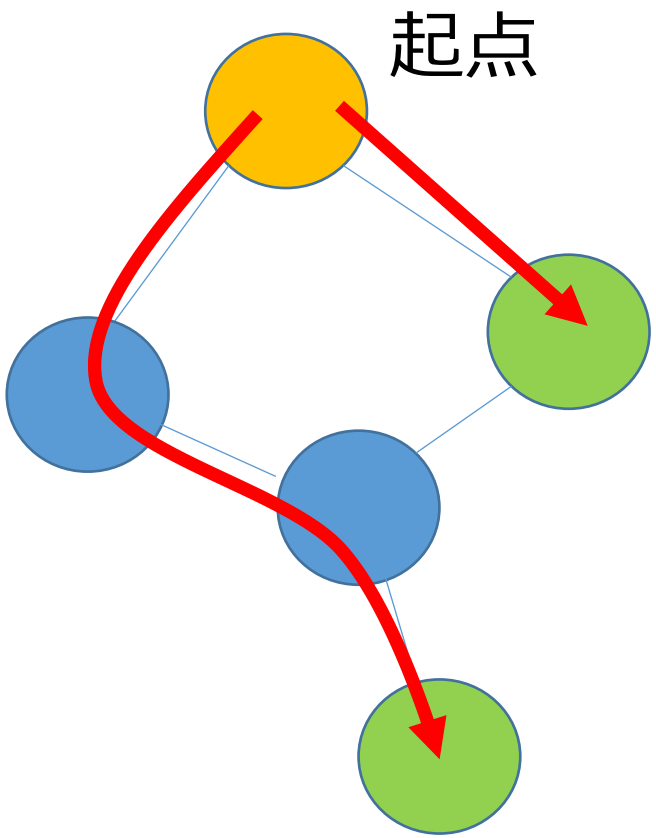
起点



グラフの中の**木**
（赤線で**木**の**パス**を示す）

このグラフに
木を書き込んでみよう
※　正解は１つではない

グラフの中の木を考えることのメリット



- 最短経路問題

路線図などで、始点から終点までの**最短経路**や**最小コスト**を見つける

- 探索問題

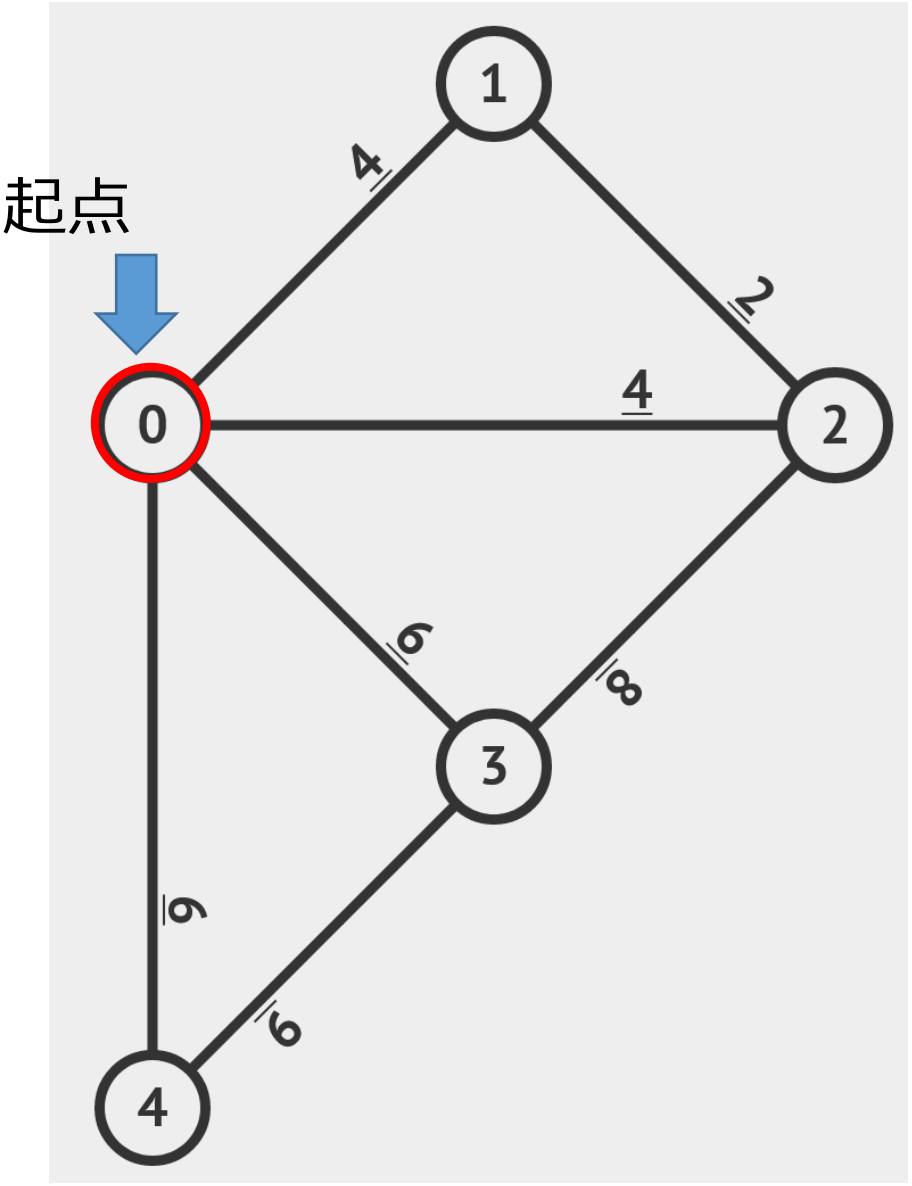
目標のノードを見つけるという探索

- 全域木（次の節で説明）

電話線や水道管などのネットワーク設計で、**すべてのノードを最小のコストで接続**

11-3 グラフと全域木

グラフ内のノードをすべて含む木（全域木）



最短パス探索（カーナビなど）

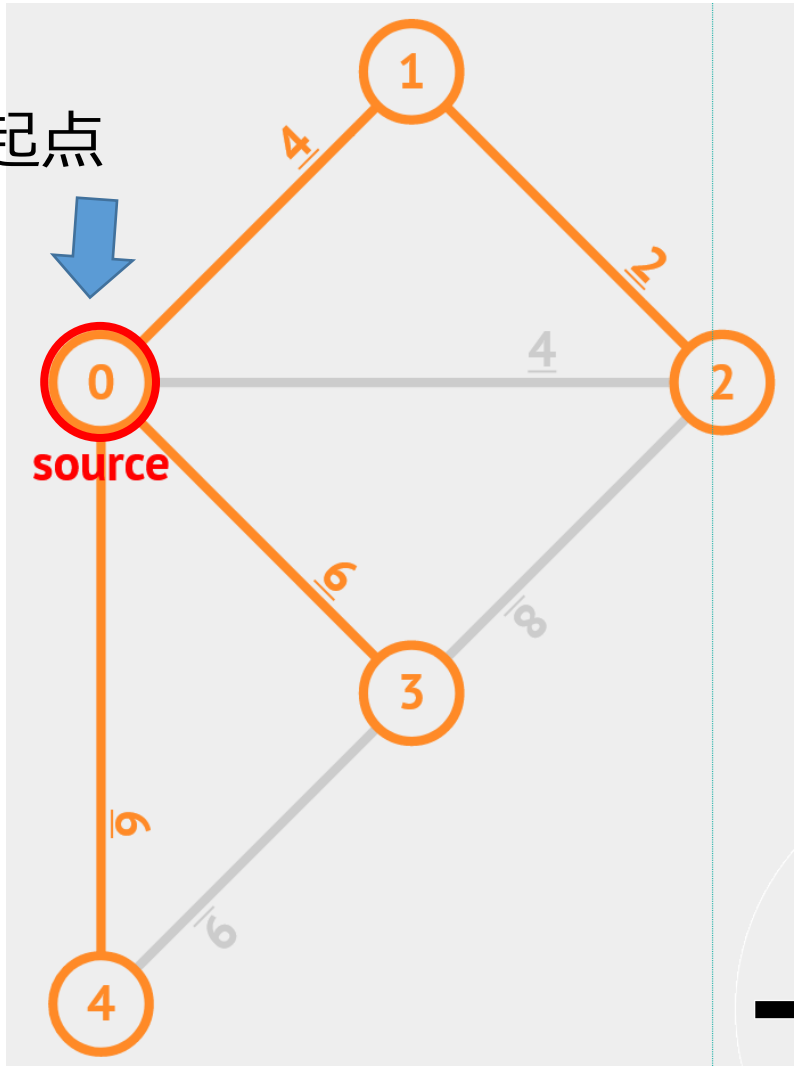
○　ノード
線　エッジ

このグラフでは，**エッジ**ごとの
移動コストが分かっている

すべてのノードをつなぐ．
コストが最小になるようにする．
例）水道管，電気配線

グラフ　https://visualgo.net/en よりコピー

グラフ内のノードをすべて含む木（全域木）



グラフの中の**木**
（赤線で**木**の**パス**を示す）

最短パス探索（カーナビなど）

○　ノード
線　エッジ

このグラフでは，**エッジ**ごとの
移動コストが分かっている

すべてのノードをつなぐ．
コストが最小になるようにする．
例）水道管，電気配線

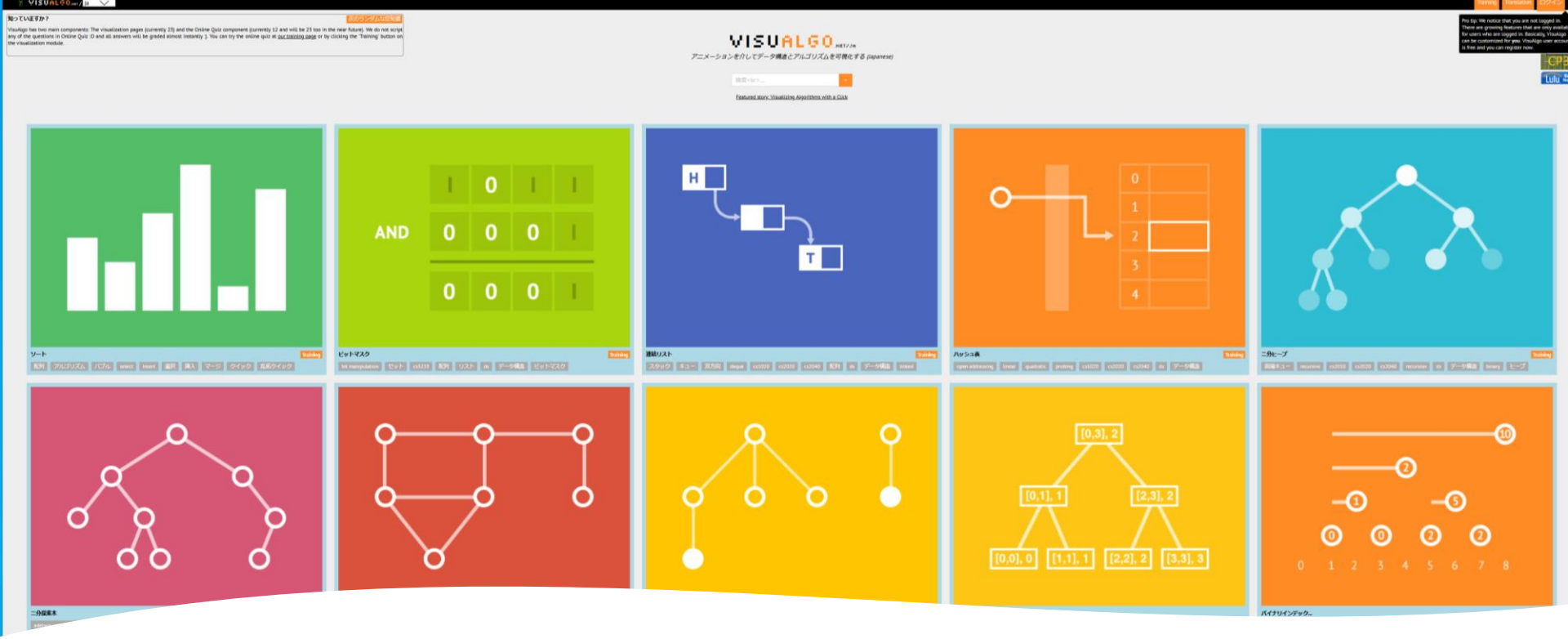
https://visualgo.net/en よりコピー

演習２

全域木をアニメーションで観察。
ページ２７～３１

【トピックス】

- **VisuAlgo**
- **グラフ**
- **全域木**



VisuAlgo

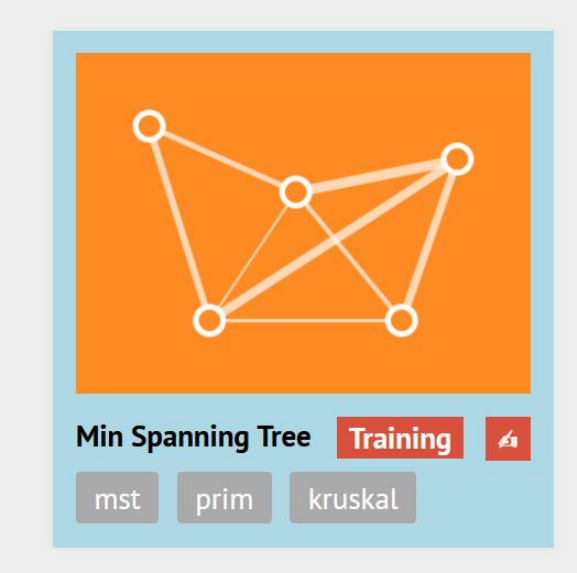
ビジュアルに，**グラフ**や**木**などの情報技術を無料で学習できる**オンラインサービス**

① ウェブブラウザで次の URL を開く

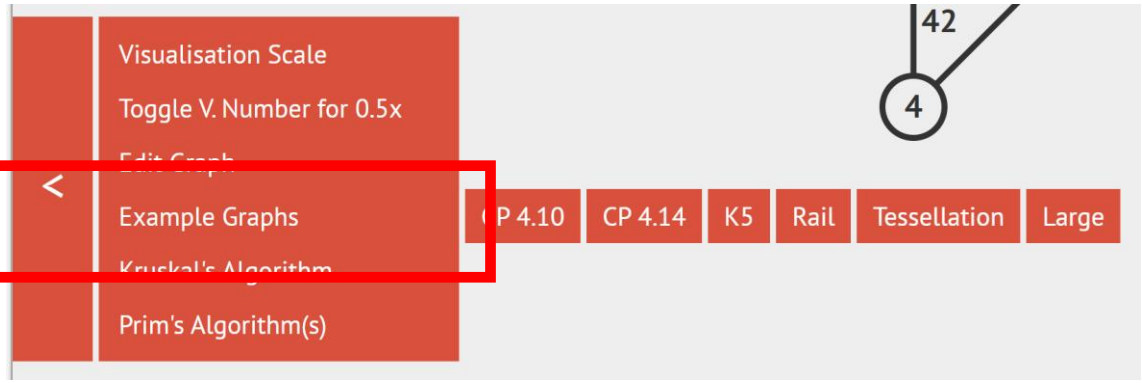
https://visualgo.net/en

② **メニュー**が**表示される**ので確認

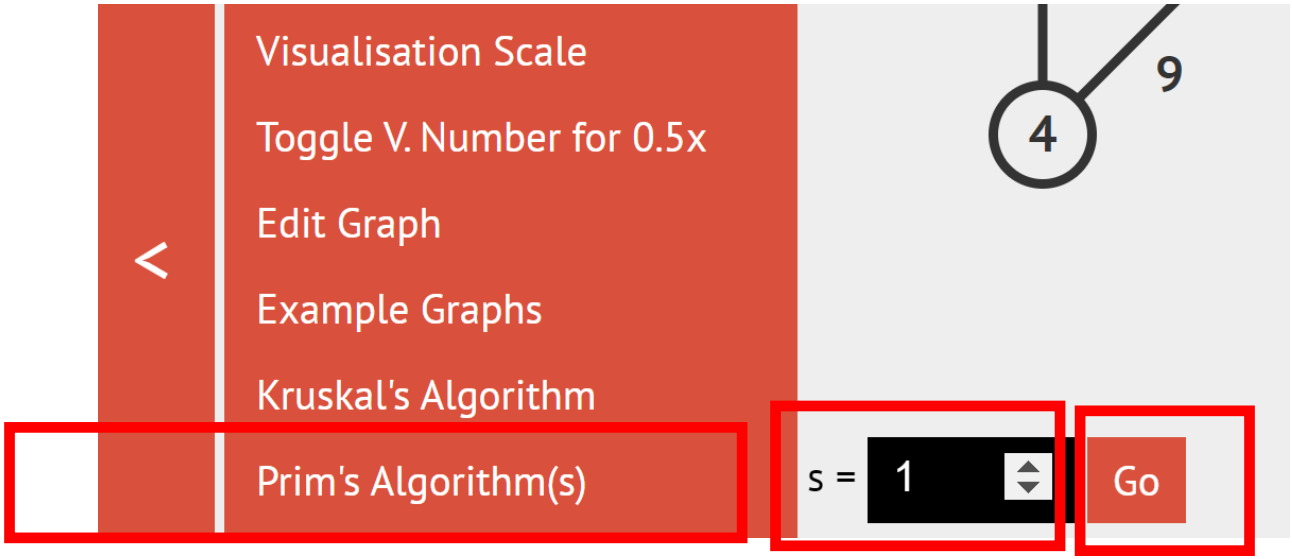
①「Min Spanning Tree (最小全域木）」をクリック



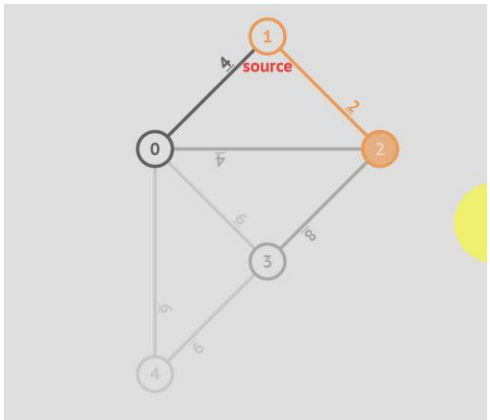
②「Example Graphs」をクリックし、グラフを選ぶ

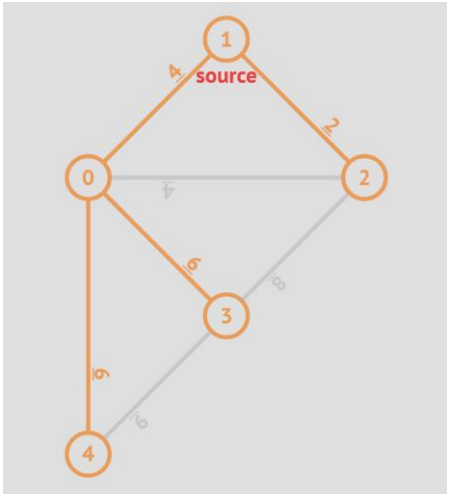


③「**Prim's Algorithm(s)**」を選び，起点とするノードの番号を設定．「**行く**」をクリック．

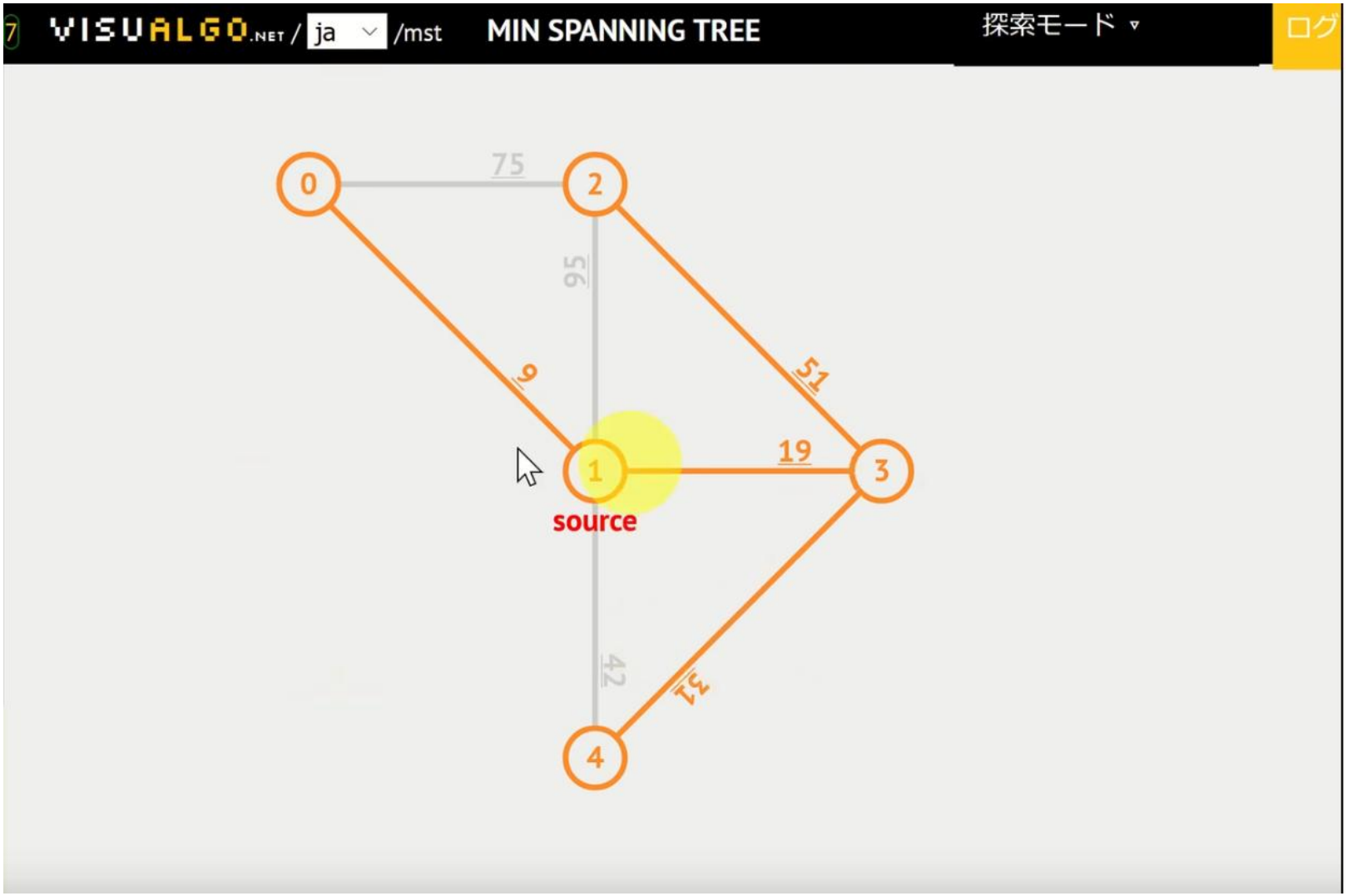


④ アニメーションで，算出過程が表示されたのち，結果が表示される





結果として表示される全域木



グラフと全域木

- **全域木**は、ある**グラフ**の中で、**全てのノードを結ぶ木**のことである。

- **全域木**は、配線、配送路の最適化など、**効率的な経路を求める場面**で活用される。

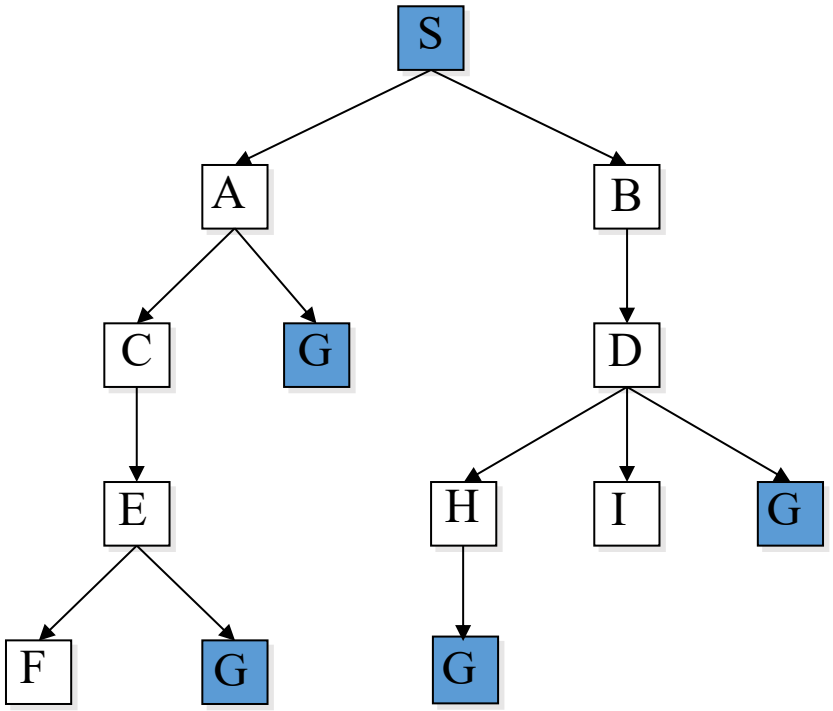
- 元の**グラフ**のままでは複雑で探索が難しい場合でも、**全域木**を考えることで、**さまざまなメリット**がある。
  - 「全てのノードを効率的に調べる」ような**探索**が簡単になる。
  - **グラフの全体的な構造を把握**しやすくなる
  - **最短経路問題**を解決しやすくなる。

11-4 探索

- **パス、木、グラフ**は情報を構造化し、**問題解決を見通し良く行う**ための考え方。
- これらは、特に**探索問題を解く**際に重要。
- **総当たり**は全ての可能性を試す探索手法。必ず**正解へと至るパスを見つけることができる。**

- **単純な探索**は、**正解へ至るパス**を**１つでも見つけた時点**で**探索**を**終了**。これにより、不必要なパスの探索を省くことができ、**効率化**を図る。ただし、**最適解を見つけられない可能性がある**点に注意が必要。

探索



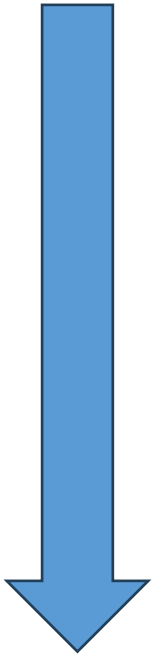
パスは6つ

1. S A C E F
2. S A C E G
3. S A G
4. S B D H G
5. S B D I
6. S B D G

探索

パスは６つ

1. S A C E F
2. S A C E G
3. S A G
4. S B D H G
5. S B D I
6. S B D G



終了

総当たりでは,
すべてのパスを試す

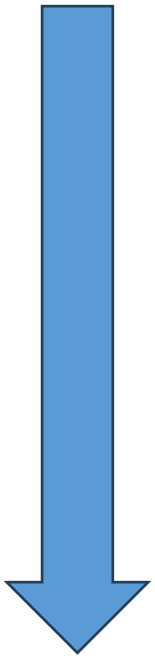
単純な探索では、
「正解 G に至るパス」を１つ
でも見つけた時点で探索を終了

木を用いた単純な探索

「単純な探索」では，**終点状態**を指定して，**パス**を探索している

終点状態：G

パスは６つ



1. S A C E F
2. S A C E **G**
3. S A **G**
4. S B D H **G**
5. S B D I
6. S B D **G**

終了

総当たり

単純な探索

演習３

総当たりの結果と、単純な探索の結果を表示。ページ３８～３９

【トピックス】

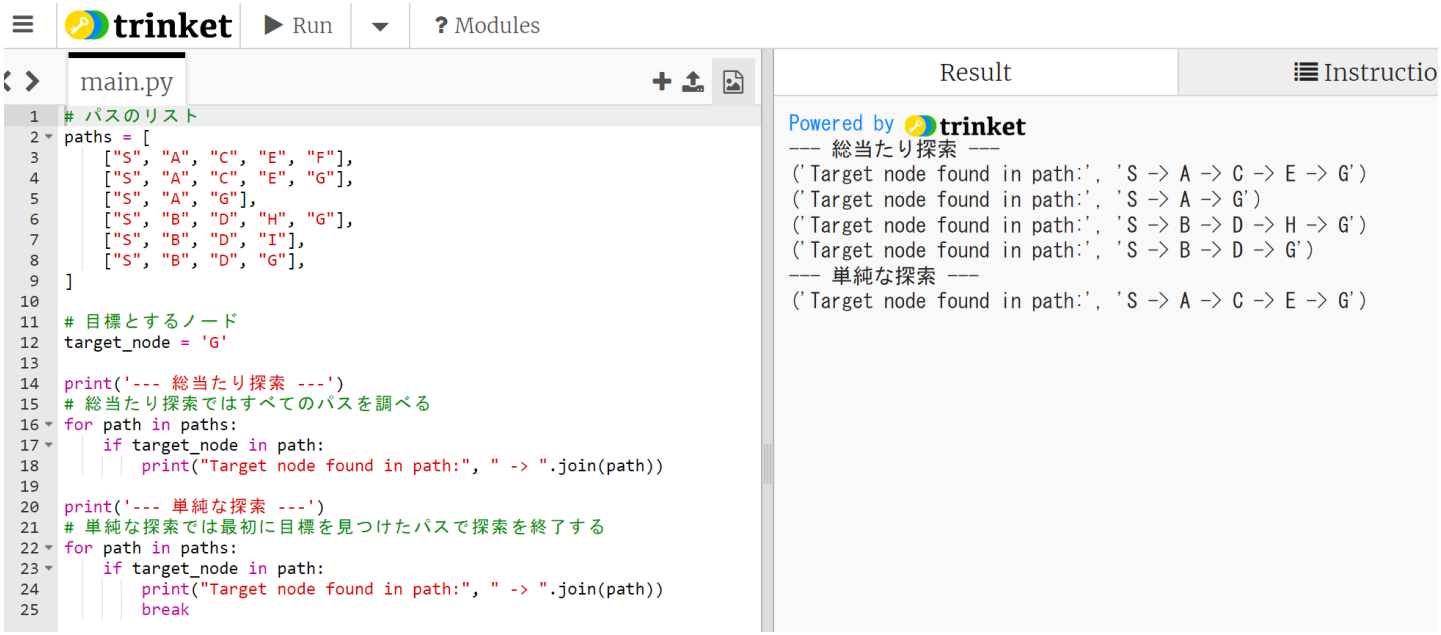
- **trinketでのプログラム実行**
- **総当たり**
- **単純な探索**

① trinket の次のページを開く

https://trinket.io/python/030e758441

② 実行結果を確認

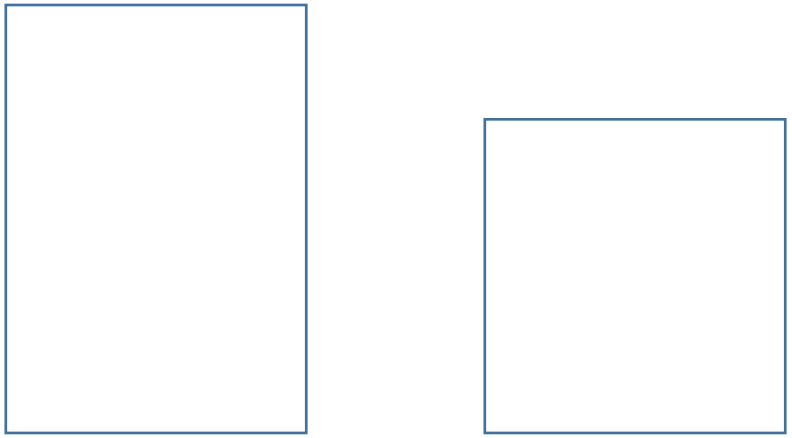
- **総当たり**では、**Gに至るパスすべて**（４つ）が求まる。
- **単純な探索**では、**Gに至るパスが１つだけ**求まる。




```python
# パスのリスト
paths = [
    ["S", "A", "C", "E", "F"],
    ["S", "A", "C", "E", "G"],
    ["S", "A", "G"],
    ["S", "B", "D", "H", "G"],
    ["S", "B", "D", "I"],
    ["S", "B", "D", "G"],
]

# 目標とするノード
target_node = 'G'

print('--- 総当たり探索 ---')
# 総当たり探索ではすべてのパスを調べる
for path in paths:
    if target_node in path:
        print("Target node found in path:", " -> ".join(path))

print('--- 単純な探索 ---')
# 単純な探索では最初に目標を見つけたパスで探索を終了する
for path in paths:
    if target_node in path:
        print("Target node found in path:", " -> ".join(path))
        break
```

２つの水差し

- 水差し①　大きさ**4**
- 水差し②　大きさ**3**



水差し①　　水差し②

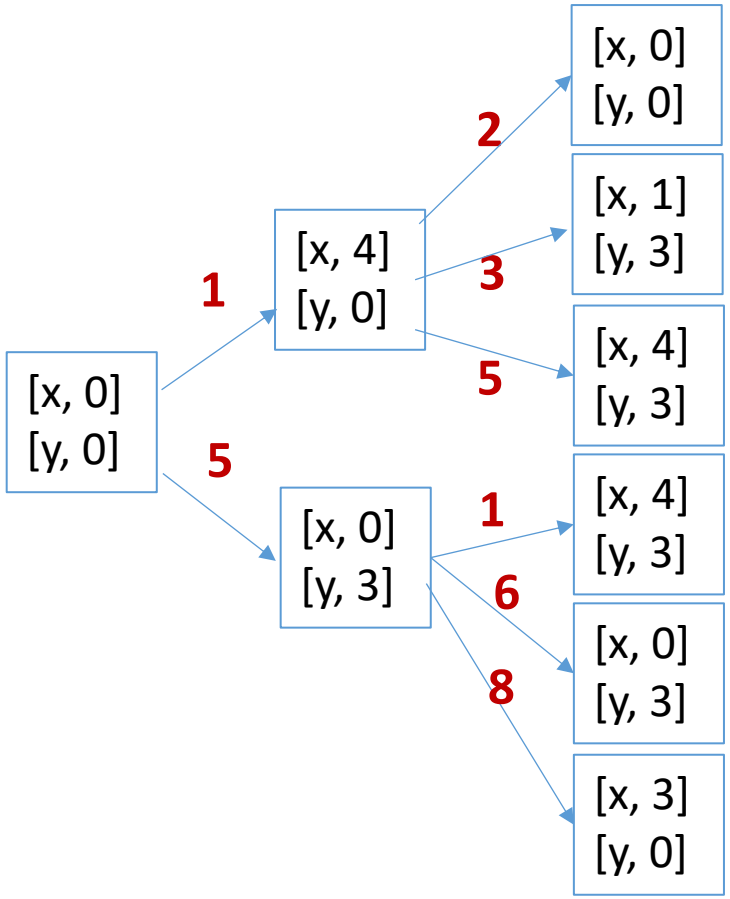
２つの水差しで、行動回数が２回

```
(1, 2) 0 0
(1, 3) 1 3
(1, 5) 4 3
(5, 1) 4 3
(5, 6) 0 3
(5, 8) 3 0
```

パス長２のパスと最終状態

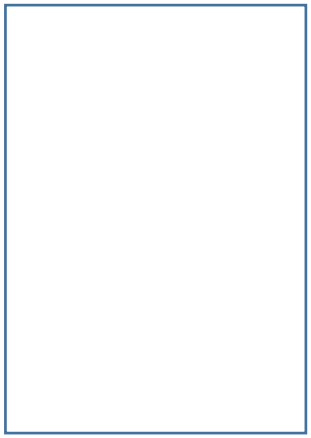
パス：（１，２）など

最終状態：　０　０などは x, y の値

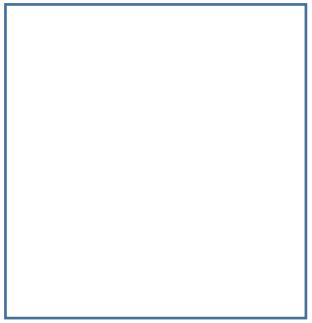


木

どちらの水差しでもよいから，**量「１」の水**が欲しい

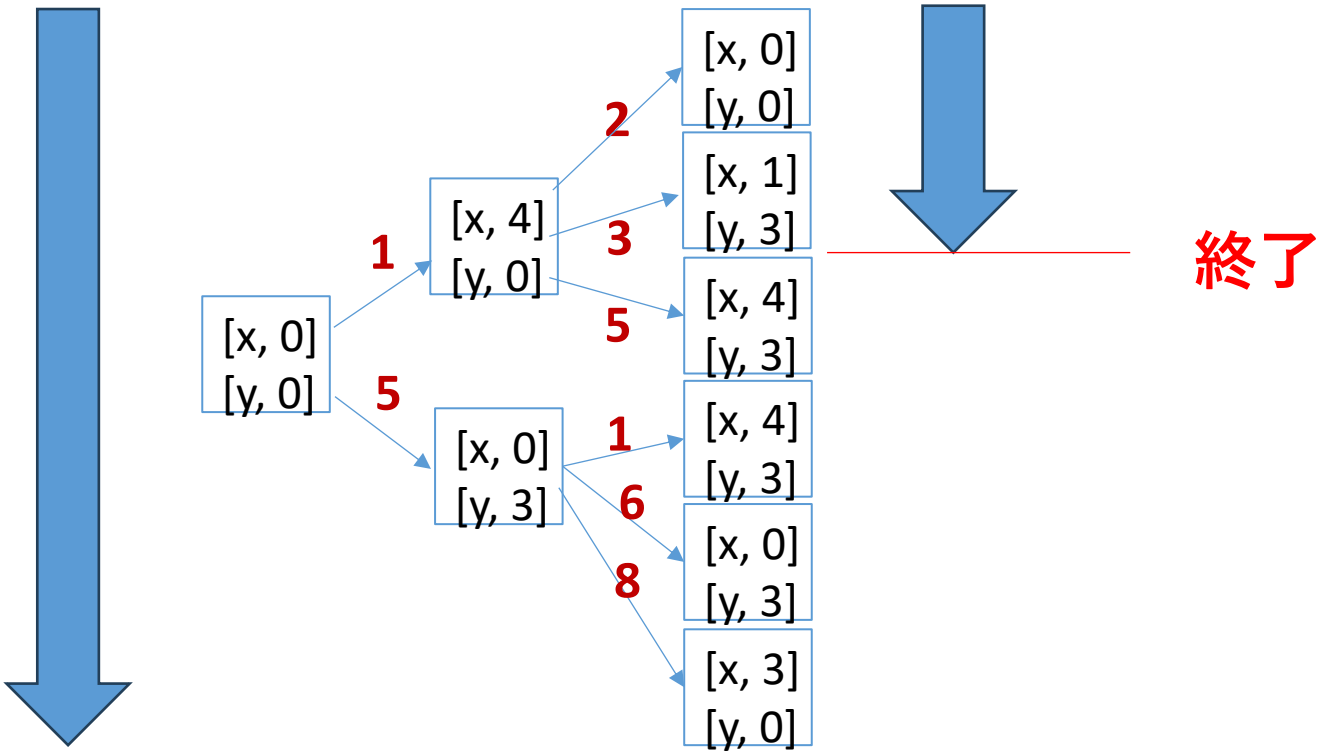


水差し①



水差し②

水の量を「１」にできるかを探索



総当たりでは,

すべてのパスを試す

単純な探索では、

「**水の量１に至るパス**」を１つでも見つけた時点で探索を終了

探索

- **探索手法の選択：探索の手法**には「**総当たり**」や「**単純な探索**」など、問題の性質や求める解の種類によって**適したものが異なる。**
- **総当たり：全てのパスを試す**ので、必ず最適解を見つけることができるが、時間とコストを多く消費する。
- **単純な探索：最初に見つかった解を採用**するため、迅速に解を得られるが、**必ずしも最適解ではない可能性**がある。

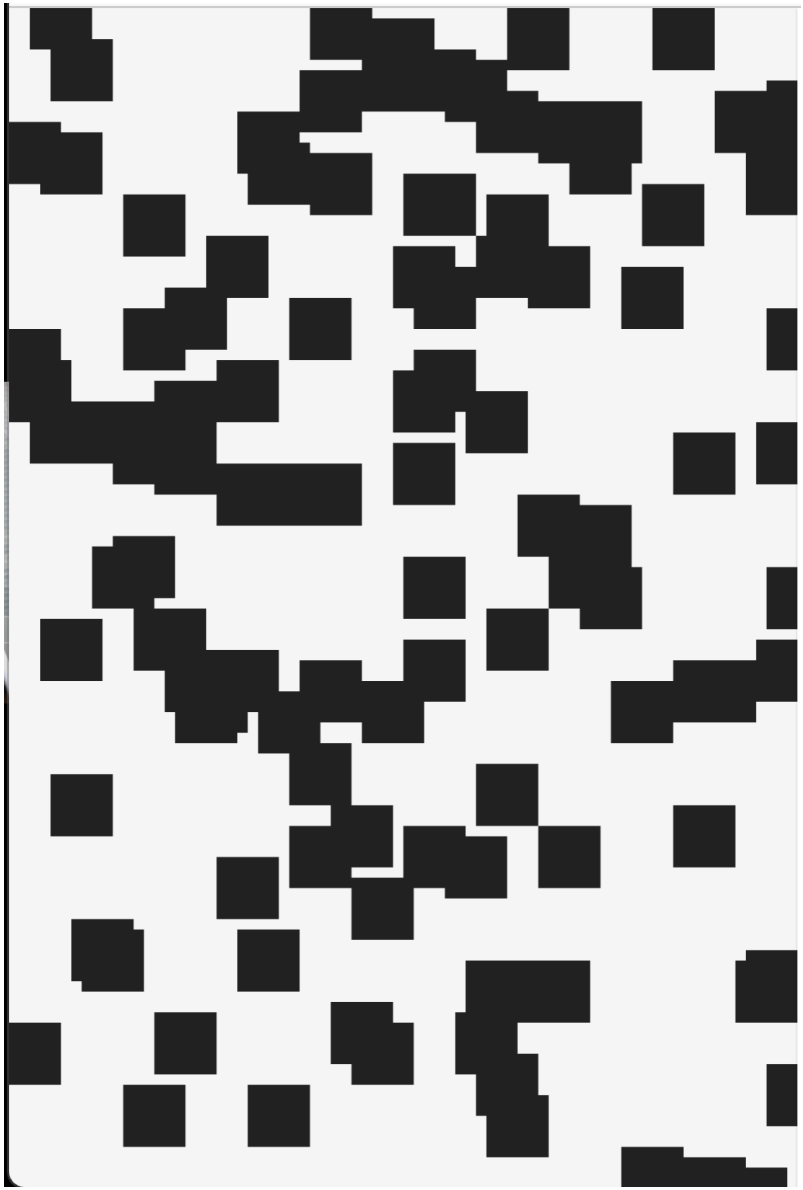
問題の複雑さ、求める解の質、利用可能な時間やコストなどを考慮して適切な手法を選択することが重要。

11-5 A* 法（エイ・スター法）

A* 法（エイ・スター法）

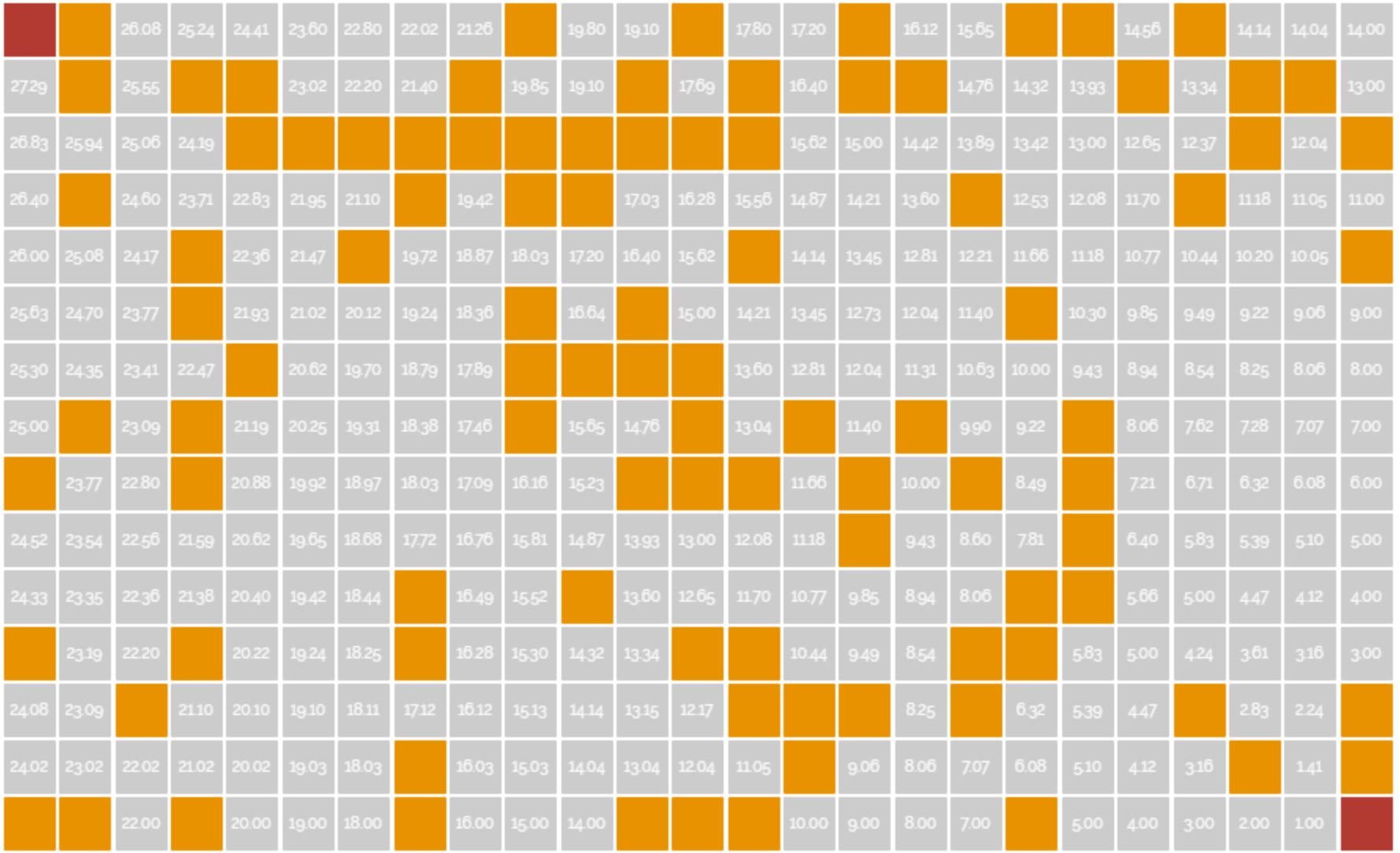
- **A* 法（エイ・スター法）**は、発見的な探索手法の一つ。
- **解に近い**と**推定**される**パス**を**優先的に探索**。
- 推定の判断基準は、**始点から現在のノードまでの実際のコスト**と、**現在のノードから目標までの推定コスト**の**合計**である。
- **不必要なパスの探索**を**減らし**、**効率的**に**最適な解を見つける可能性**が高まる。

A* 法による探索の例



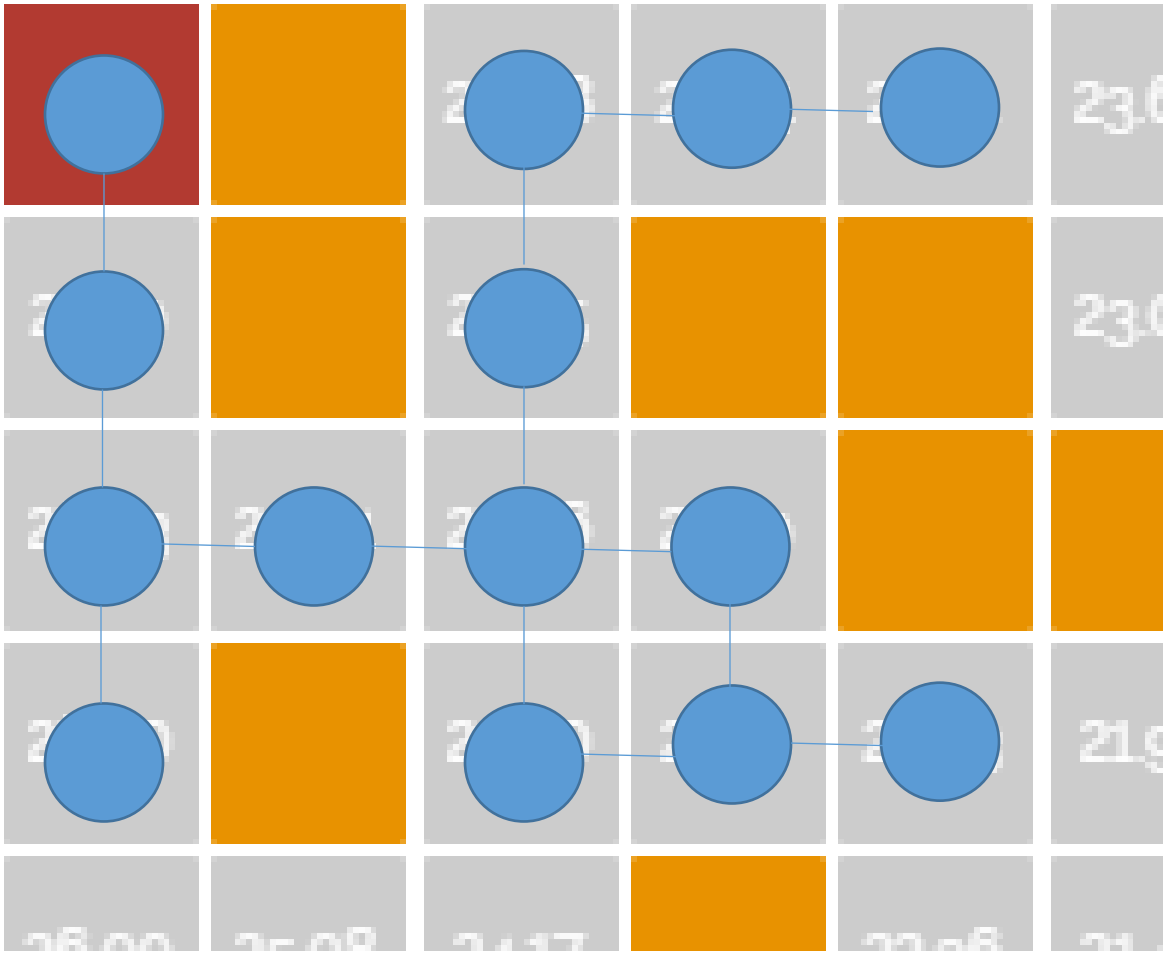
https://www.growingwiththeweb.com/projects/pathfinding-visualiser/

迷路の例


26.08 25.24 24.41 23.60 22.80 22.02 21.26 19.80 19.10 17.80 17.20 16.12 15.65 14.56 14.14 14.04 14.00
27.29 25.55 23.02 22.20 21.40 19.85 19.10 17.69 16.40 14.76 14.32 13.93 13.34 13.00
26.83 25.94 25.06 24.19 15.62 15.00 14.42 13.89 13.42 13.00 12.65 12.37 12.04
26.40 24.60 23.71 22.83 21.95 21.10 19.42 17.03 16.28 15.56 14.87 14.21 13.60 12.53 12.08 11.70 11.18 11.05 11.00
26.00 25.08 24.17 22.36 21.47 19.72 18.87 18.03 17.20 16.40 15.62 14.14 13.45 12.81 12.21 11.66 11.18 10.77 10.44 10.20 10.05
25.63 24.70 23.77 21.93 21.02 20.12 19.24 18.36 16.64 15.00 14.21 13.45 12.73 12.04 11.40 10.30 9.85 9.49 9.22 9.06 9.00
25.30 24.35 23.41 22.47 20.62 19.70 18.79 17.89 13.60 12.81 12.04 11.31 10.63 10.00 9.43 8.94 8.54 8.25 8.06 8.00
25.00 23.09 21.19 20.25 19.31 18.38 17.46 15.65 14.76 13.04 11.40 9.90 9.22 8.06 7.62 7.28 7.07 7.00
23.77 22.80 20.88 19.92 18.97 18.03 17.09 16.16 15.23 11.66 10.00 8.49 7.21 6.71 6.32 6.08 6.00
24.52 23.54 22.56 21.59 20.62 19.65 18.68 17.72 16.76 15.81 14.87 13.93 13.00 12.08 11.18 9.43 8.60 7.81 6.40 5.83 5.39 5.10 5.00
24.33 23.35 22.36 21.38 20.40 19.42 18.44 16.49 15.52 13.60 12.65 11.70 10.77 9.85 8.94 8.06 5.66 5.00 4.47 4.12 4.00
23.19 22.20 20.22 19.24 18.25 16.28 15.30 14.32 13.34 10.44 9.49 8.54 5.83 5.00 4.24 3.61 3.16 3.00
24.08 23.09 21.10 20.10 19.10 18.11 17.12 16.12 15.13 14.14 13.15 12.17 8.25 6.32 5.39 4.47 2.83 2.24
24.02 23.02 22.02 21.02 20.02 19.03 18.03 16.03 15.03 14.04 13.04 12.04 11.05 9.06 8.06 7.07 6.08 5.10 4.12 3.16 1.41
22.00 20.00 19.00 18.00 16.00 15.00 14.00 10.00 9.00 8.00 7.00 5.00 4.00 3.00 2.00 1.00

迷路の探索

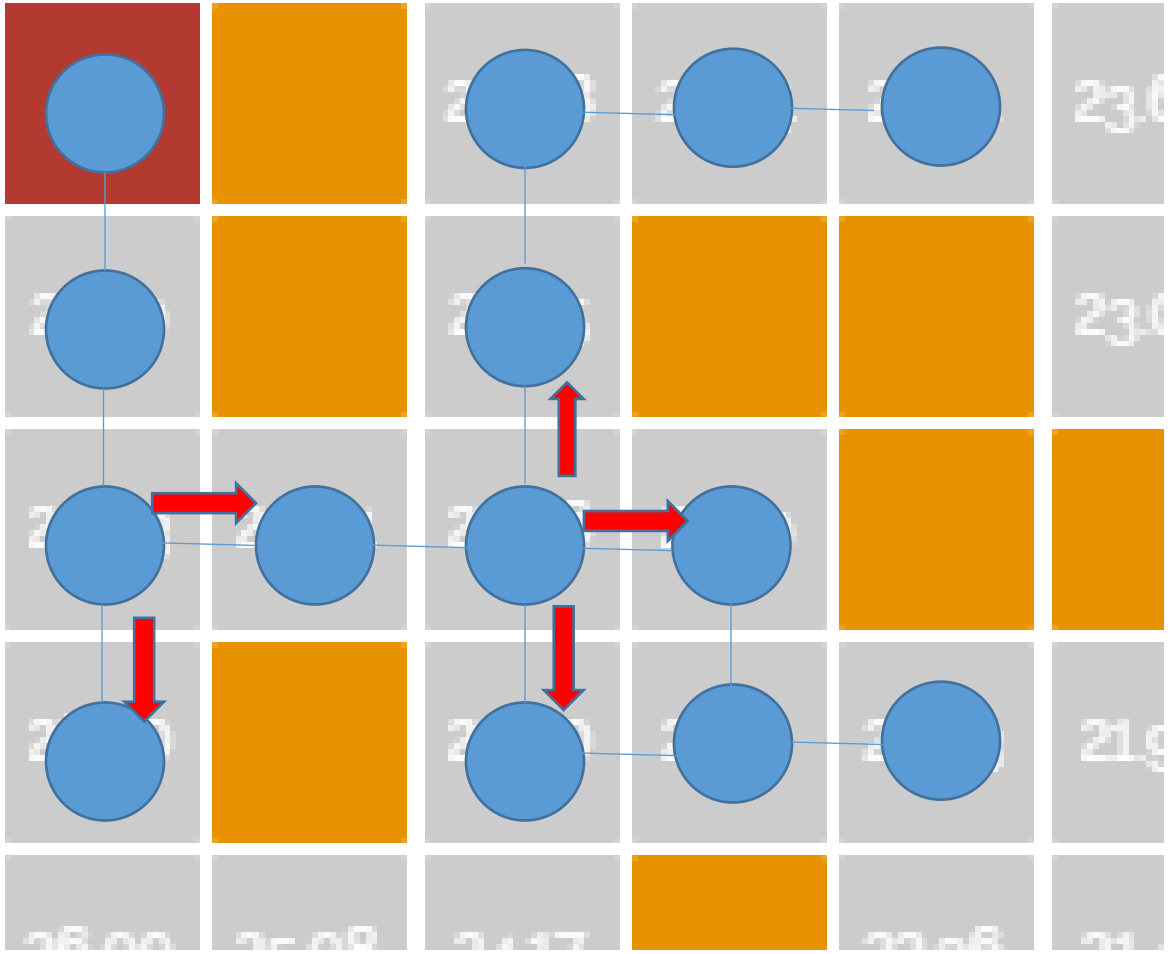
起点



A* 法のルール

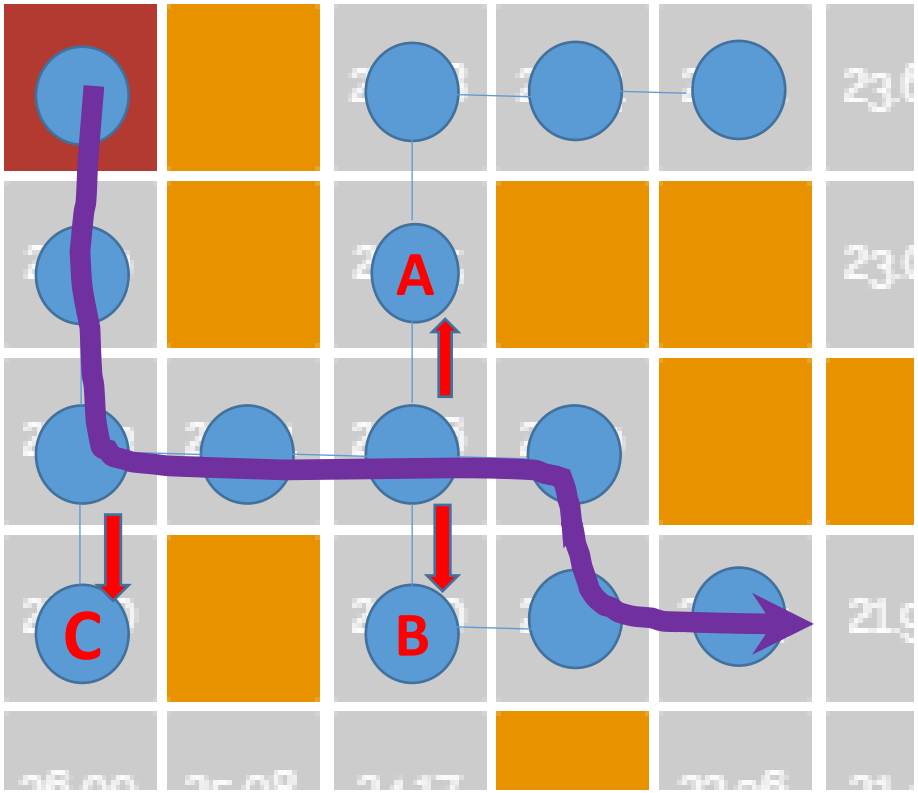
１．分岐では，「**解に近いと推定されるパス**」を**優先的に選ぶ**

起点



A* 法のルール

２．行き止まりに来たら，元の分岐に戻る．
今まで通ったすべての分岐の**中で**，
「**起点から分岐までにかかったコスト**」と
「**その分岐からもう1歩動くコスト**」の合計が**最小の分岐まで戻る**



A, B, C の中で
解に近いと推定される
パスを優先を変える

今のパスに行き止まりや
難所があったとする

ここまでのまとめ

- **A* 法（エイ・スター法）**は、**解に近い**と**推定**される**パス**を**優先的に探索**する手法。
- **始点から現在のノードまでの実際のコスト**と、**現在のノードから目標までの推定コスト**を合わせて**最短経路を推定**し、発見的に最適解を探す。
- 行き止まりに来た場合は、コストが最小の分岐に戻る。

演習４

A*法を用いた迷路探索のシミュレーション。ページ５３～５９

【トピックス】

- **A*法**

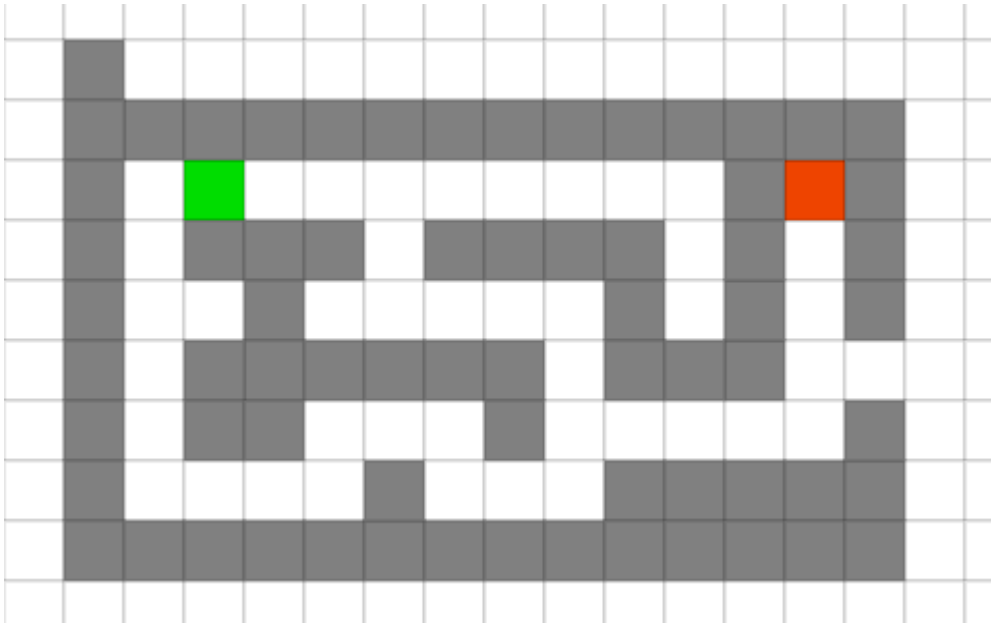
① A*法による迷路探索をシミュレーションできるウェブツールを使用します

Webブラウザで開く

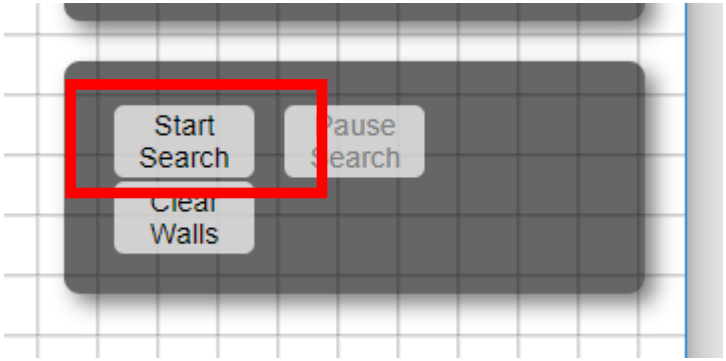
https://qiao.github.io/PathFinding.js/visual/

② マウスを使って迷路を描く

緑：起点　赤：終点



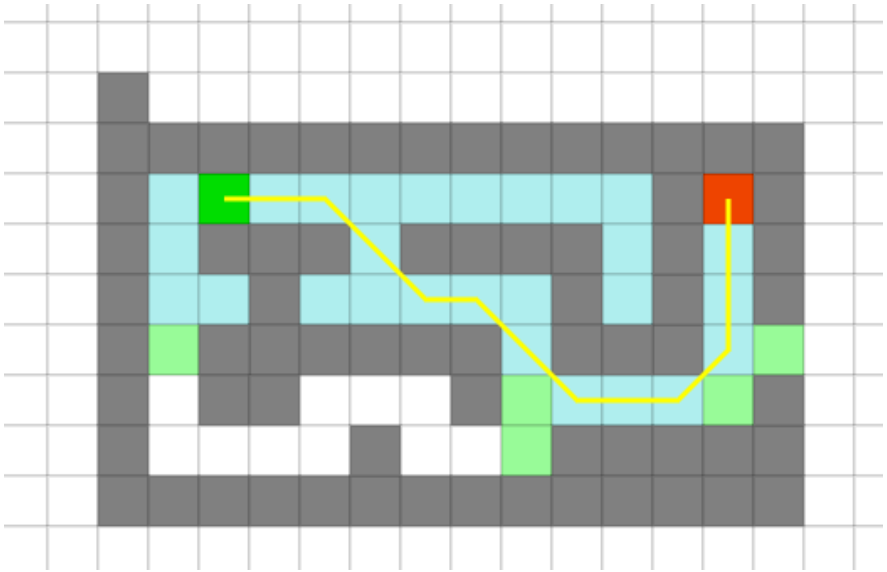
③「Start Search」をクリック



④　A* 法による探索結果を確認

水色は，探索された部分

※ **薄緑**は，**通らないことにした**方向

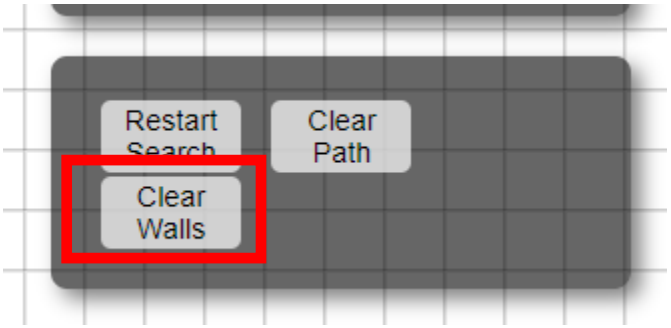


最終的に表示される黄色の線が**最短経路**。
A法がどのように効率的に目標に向かって探索を進めるかを観察できる。

⑤　演習：異なる迷路パターンでA*法の探索過程を観察し、以下の点について考察

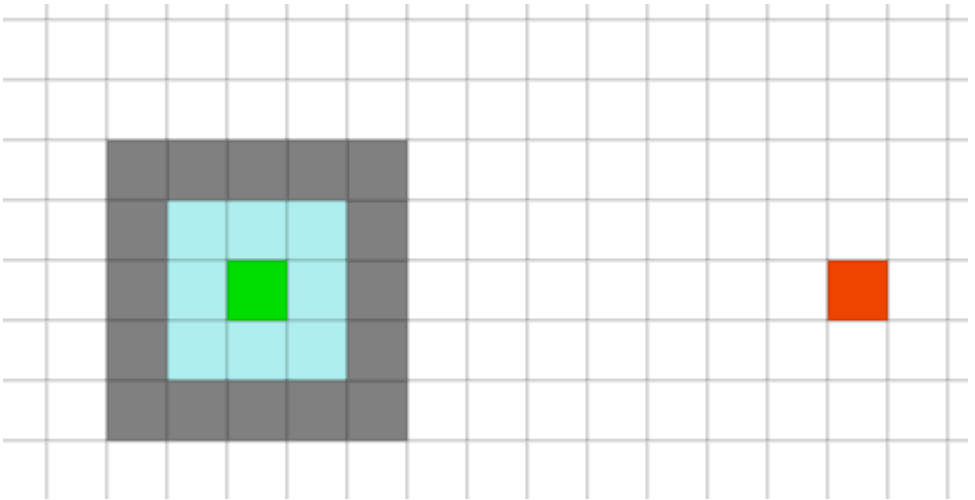
１．　障害物の配置によって探索範囲はどのように変化しますか？

２．始点と終点の位置関係によって、探索効率はどのように変わりますか？



新規作成は

「Clear Walls」



解けないことも

当然ある

障害物の配置によって探索範囲はどのように変化しますか？

考察例：障害物が多いほど、または複雑に配置されているほど、探索範囲が広がる傾向がある。

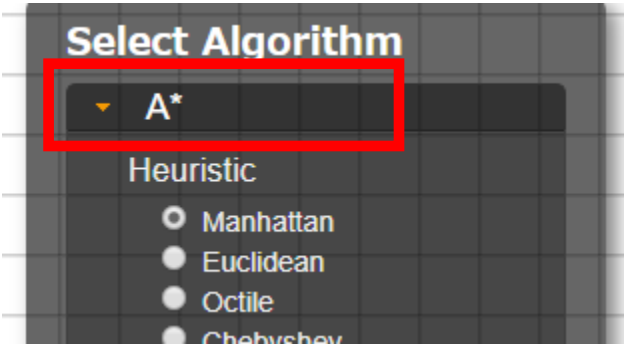
- A*法は最適経路を見つけようとする
- 障害物により直線的経路が遮られると、迂回路を探索
- 障害物が少ない → 狭い範囲で目標到達
- 障害物が多い → 様々な経路を探索 → 探索範囲拡大

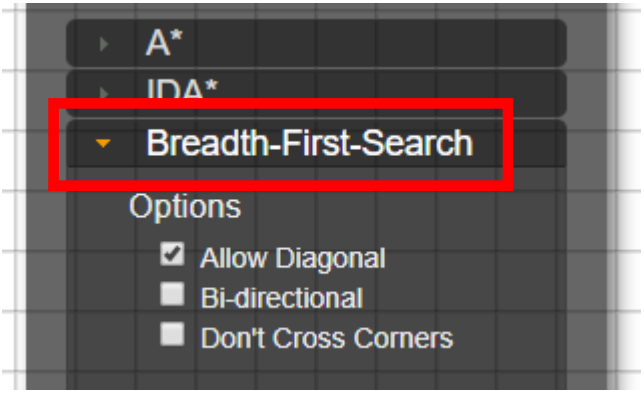
始点と終点の位置関係によって、探索効率はどのように変わりますか？

考察例：始点と終点が近く、直線的に結べる場合は探索効率が高く、遠く離れていたり、間に多くの障害物がある場合は効率が低下する。

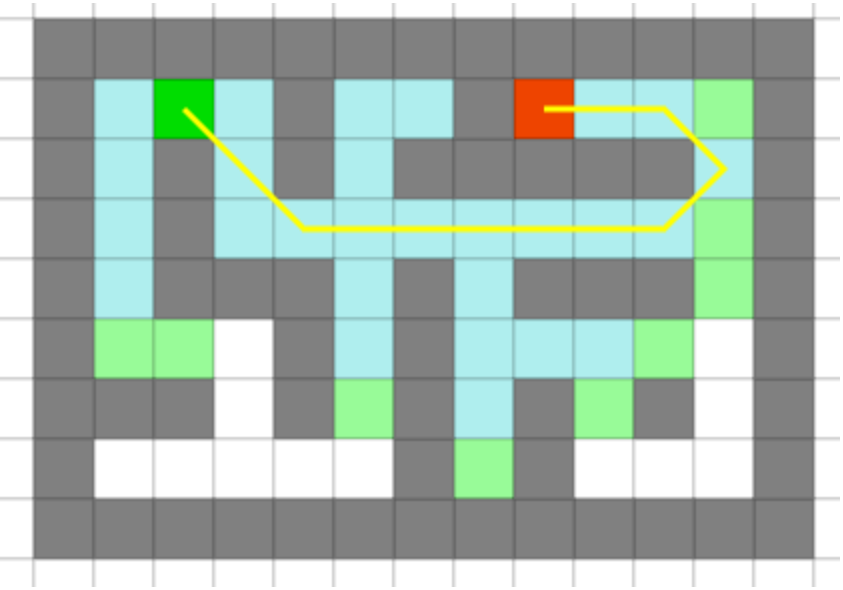
- 直線的 → 推定精度が高い → 効率的探索
- 遠く複雑 → 推定精度が低下 → より多くの経路探索が必要
- 迂回が必要な場合 → 探索範囲拡大 → 効率低下

A* 法と総当たりの比較



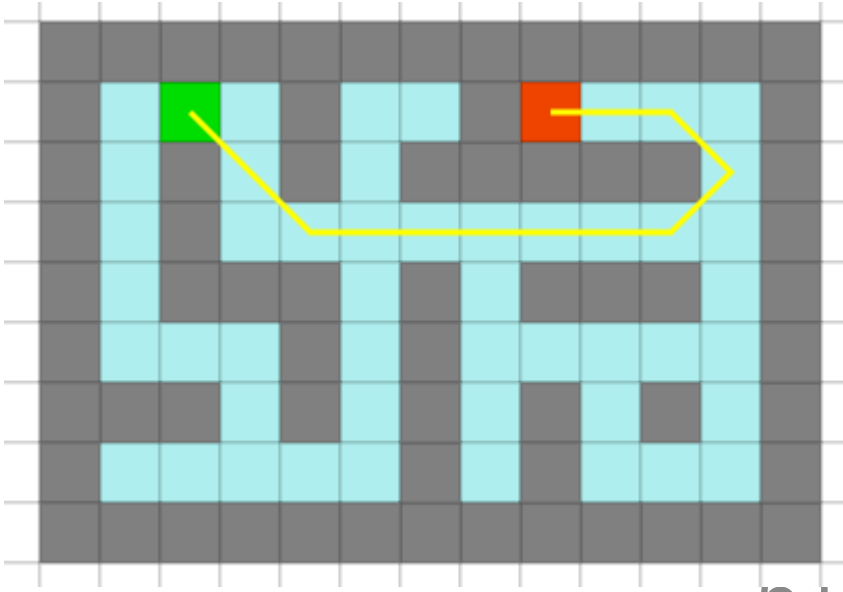


A* を選ぶ場合 **A***



総当たりを選ぶ場合

Breadth-First-Search



全体まとめ

- **木**：**共通の起点から始まる複数のパスの集まり**。階層構造を持つデータを扱ったり、最短経路問題や探索などに用いられる。
- **パス**：**木**においては、**起点から各末端へと進むパス**があり、それらは合流することはない。
- **探索**：**特定の目標に向けた手段や方法を見つけるためのプロセス**。一度試したパスを再び試さずに未探索のパスへと進むことが効率のために大切。
- **総当たり**：**全ての可能性を試す**手法で、必ず解を見つけることができるが、時間がかかることがある。
- **単純な探索**：**最初に見つかった解を採用**し、解が見つかった時点で探索を**打ち切る**。
- **グラフ**：**ノード**（頂点）と**エッジ**（辺）からなる構造で、道路のつながり具合やバス路線などを表現するのに使用される。
- **全域木**：**グラフ内のすべてのノードをつなぐ木**。電話線や水道管などのネットワーク設計に活用される。
- **A*法**：発見的な探索手法の一つで、**解に近いと推定されるパスを優先的に探索**する。

① **AI基礎概念の体系的理解。パス、木、グラフ、探索**などAIの基本概念。実世界の問題に応用する能力。

② 効率的な**問題解決手法の習得。総当たり法、単純探索、A*法など**様々な探索手法を比較学習

③ 視覚的ツールによる**実践的スキル向上。**trinketやVisuAlgoなどの対話的ツールを使用し、**動作を視覚的に理解**しながら、スキルを向上

④ **AIの応用と影響力の認識。探索の応用例**を学び、AI技術が与える影響を理解。