10. 探索、総当たり、発見的探索

（人工知能）

金子邦彦

「人工知能」第１０回の内容

AIは『状態・行動・遷移関数』で将来を読み、探索して選択する

状態（今の状況） → 行動（取りうる操作） → 遷移関数（行動で状態がどう変わる）

適用条件＝その行動が可能になる条件

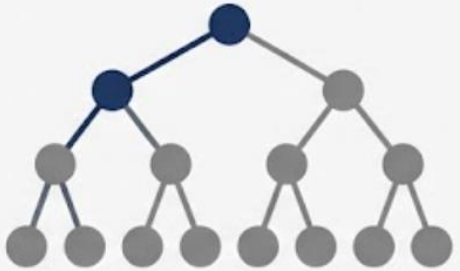
探索

遷移関数で
可能な行動と将来
のお状態を
広げて調べる

選択

状態を評価し
目標に最も近づく行動を
1つに絞る

木



起点から枝分かれする経路（パス）の集まり
調べ済みの経路を二度たどらず
未探索を一つずつ調べる

総当たり	A*法
全経路を試して 最適解を得る	『来た道の長さ＋ゴールまでの見積もり』が 小さい行動を優先して調べる

10-1. 状態、行動、遷移関数

状態・行動・遷移関数

行動を選ぶと、状態が次の状態へ変わる ― これを扱うのが状態・行動・遷移関数の枠組み

状態と行動：エレベーターの例

状態は『今どこにいるか』、行動は『次に何をするか』を表し、行動の結果として次の状態が決まる

状態

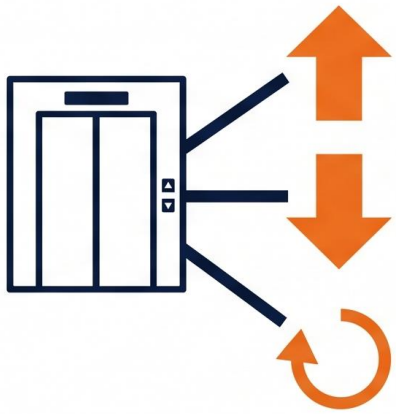
今おかれている具体的な状況

現在の階数
＝状態

状態は変数で表現する

行動

状態に対して取りうる操作
（結果として次の状態が決まる）



行動1：上に移動
（次の状態へ遷移）

行動2：下に移動
（次の状態へ遷移）

行動3：その場に留まる
（同じ状態のまま）

各行動には行動番号が割り当てられる。行動の結果、
状態は別の状態に遷移するか、同じ状態に留まる

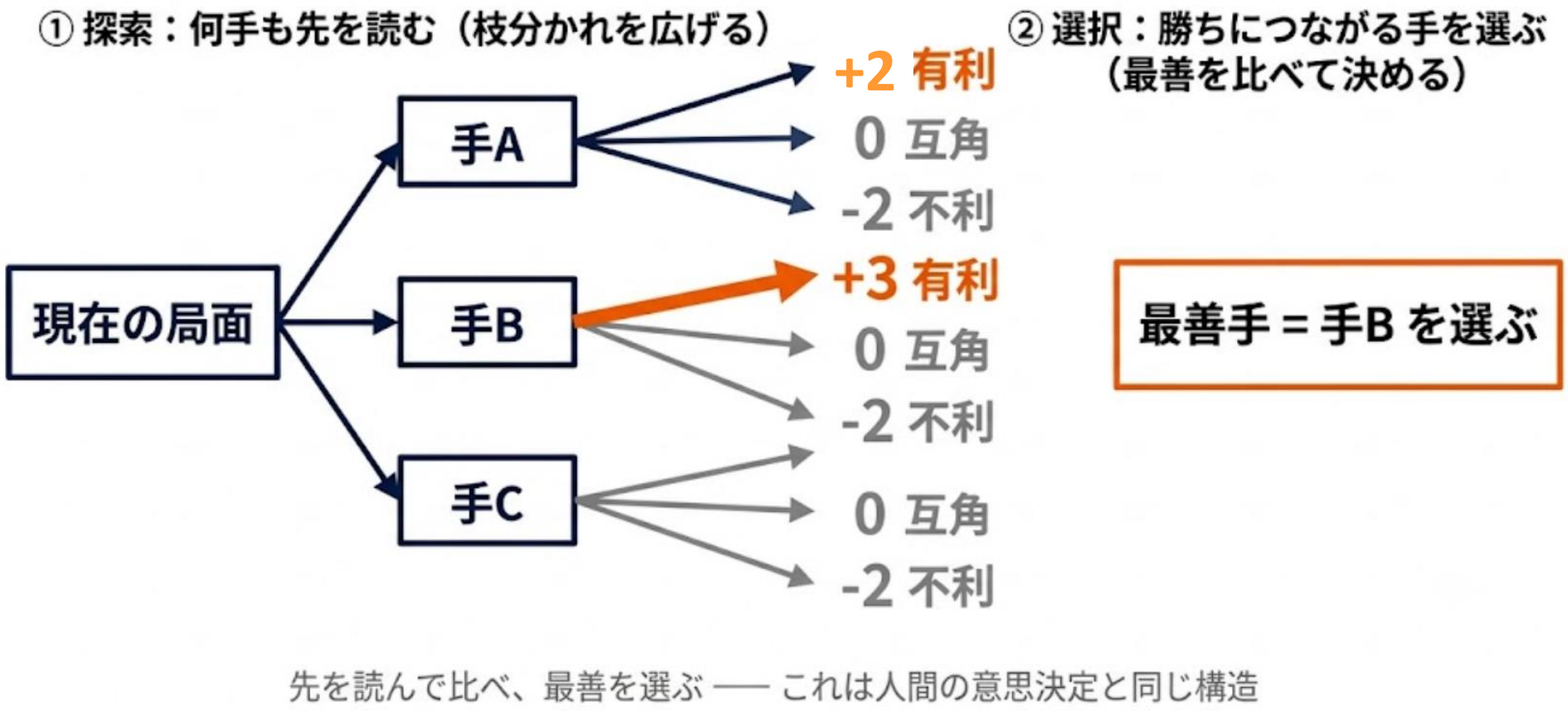
遷移関数と適用条件：エレベーターの例

遷移関数：行動を取ると状態 x（階数）がどう変わるかを定める規則。適用条件を満たすときだけ変化する

行動	遷移関数	適用条件
上へ移動	$x \rightarrow x+1$	x＜最上階のときのみ可能。 満たさなければ変化しない
下へ移動	$x \rightarrow x-1$	x＞1階のときのみ可能。 満たさなければ変化しない
留まる	$x \rightarrow x$	常に可能（状態は変わらない）

探索と選択

複数の行動を比較し最善を選ぶ

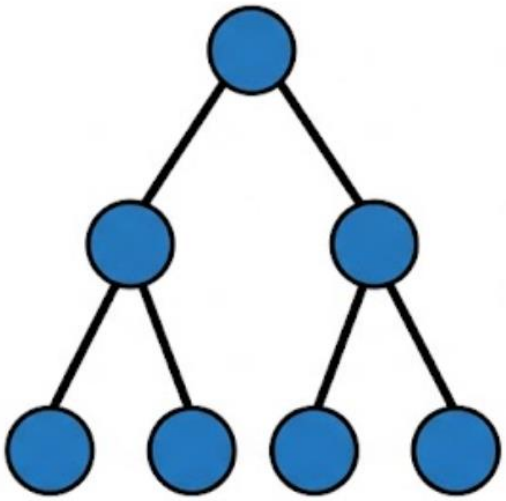


先を読んで比べ、最善を選ぶ ── これは人間の意思決定と同じ構造

木と探索

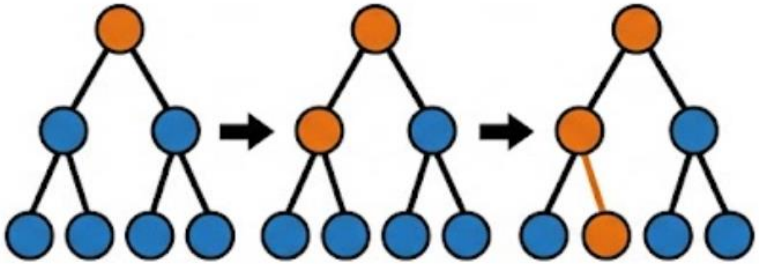
木構造を使うと、探索を重複なく1つずつ効率的に進めることができる

(1) 木構造とは



根から枝分かれする
階層のかたち

(2) 効率的な探索



まず根を調べる　未探索の枝へ進む　通った道は再び試さない

一度通ったパスを繰り返さず、
未探索へ1つずつ進む

(3) 主な用途

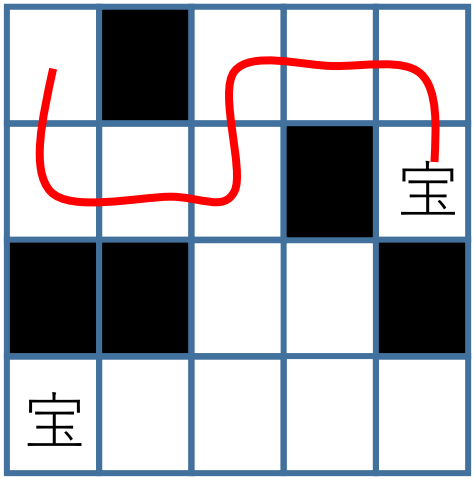
階層構造データ
組織図、フォルダとファイル

最短経路問題
最も効率的な経路を求める

探索問題
解決策を調べ、解を見つける

単純な探索

可能な経路を試して、正解の経路（パス）を見つけた時点で探索を終了する



メリット

不要なパスの探索を省けるので、効率がよい

デメリット

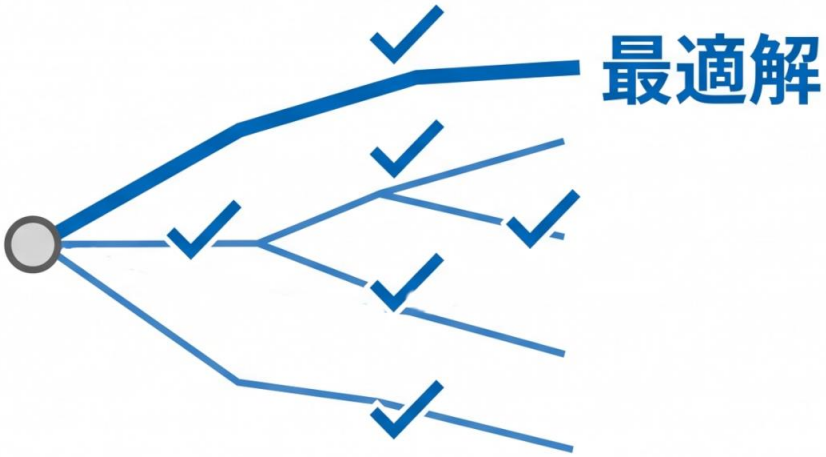
最初に見つけた解で止まるため、最適解とは限らない

正解の経路（パス）を見つけた時点で探索を終了

総当たりと単純な探索の比較

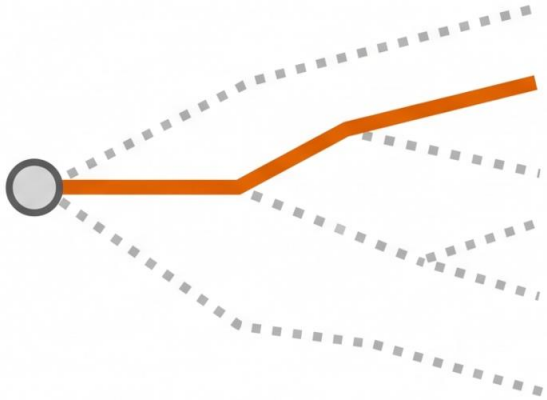
総当たりは『最適解』を、単純な探索は『速さ』を優先する

総当たり（全部試す）



- 必ず最適解が見つかる
- 時間とコストが大きい

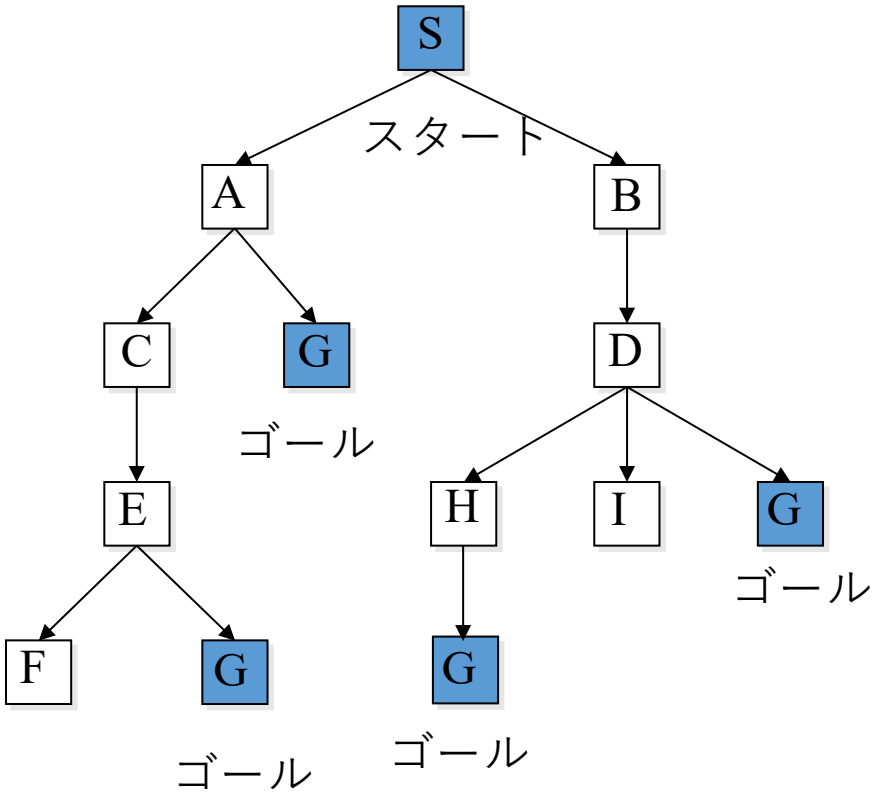
単純な探索（最初の解で止める）



- すぐに解が得られる
- 最適解とは限らない

問題の複雑さ・求める解の質・使える時間とコストに応じて手法を選ぶ

木

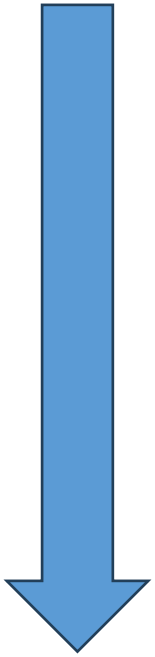


６つの経路（パス）

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：総当たりと単純な探索の比較

６つの経路（パス）



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

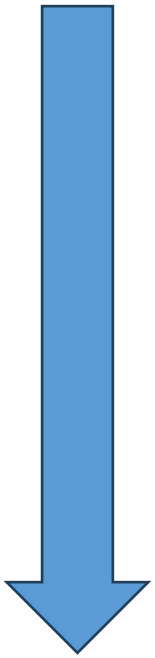
探索：総当たりと単純な探索の比較

「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**



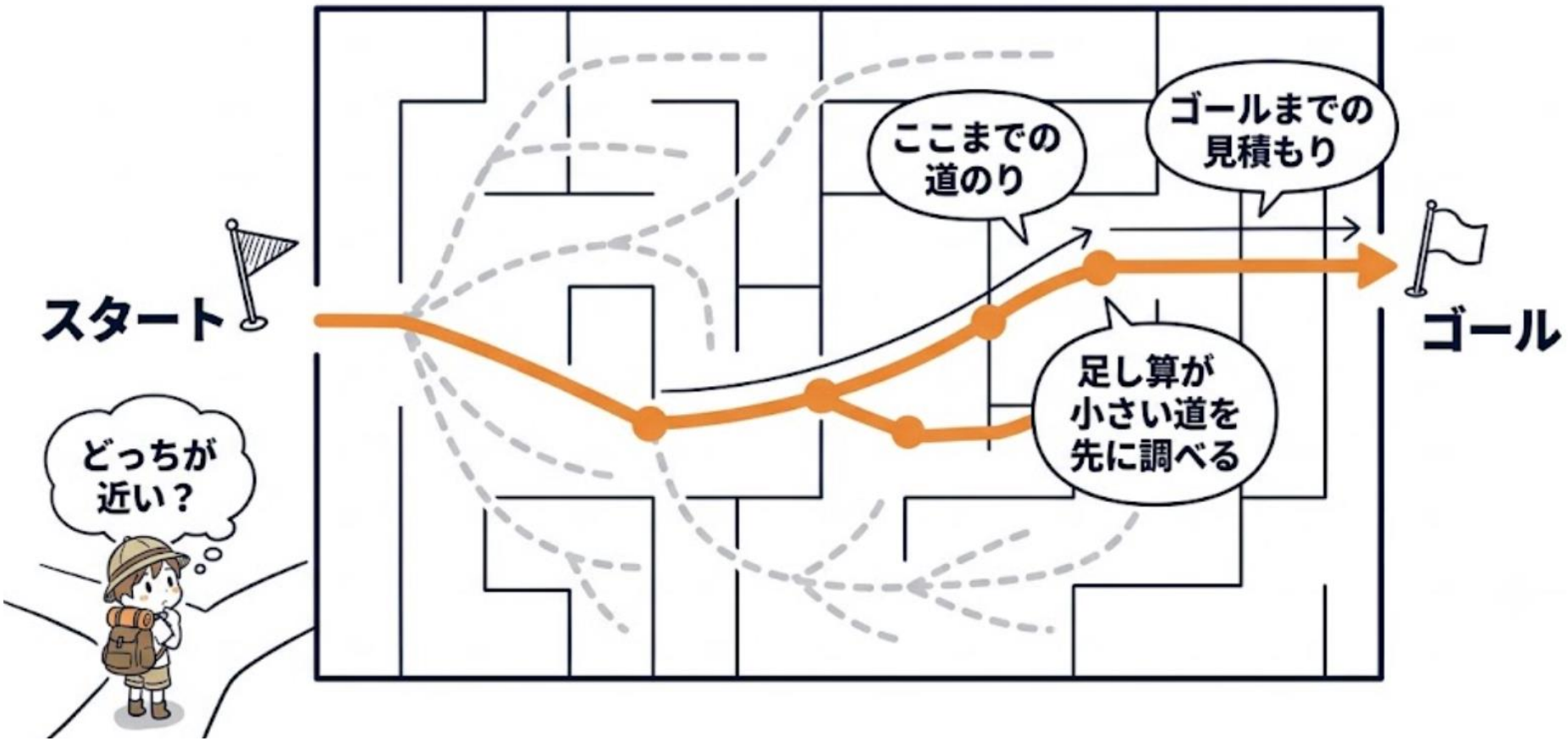
終了

総当たり

単純な探索

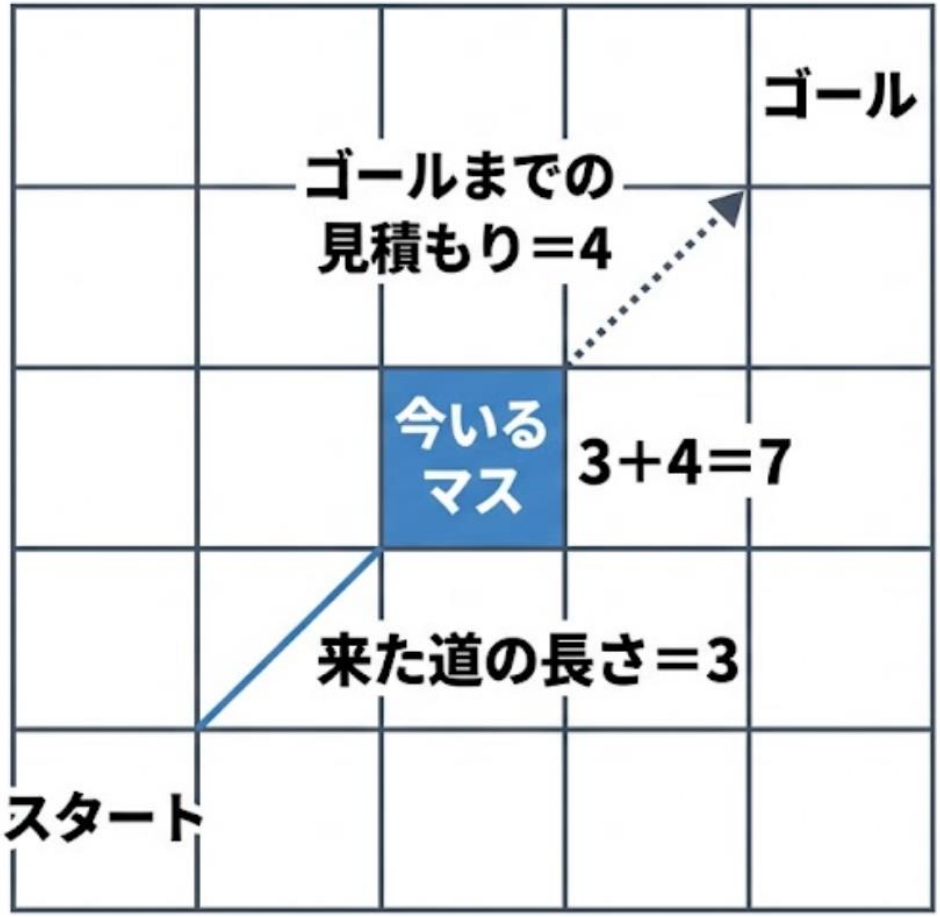
10-5. 発見的探索 A*法

A*法は『今まで歩いた道のり』と『ゴールまでの近道の見積もり』を足して、小さいものから優先して調べる



A* 法（エイ・スター法）

A*法：『来た道の長さ』＋『ゴールまでの見積もり』が小さいマスから先に調べる



次にどのマスへ進む？

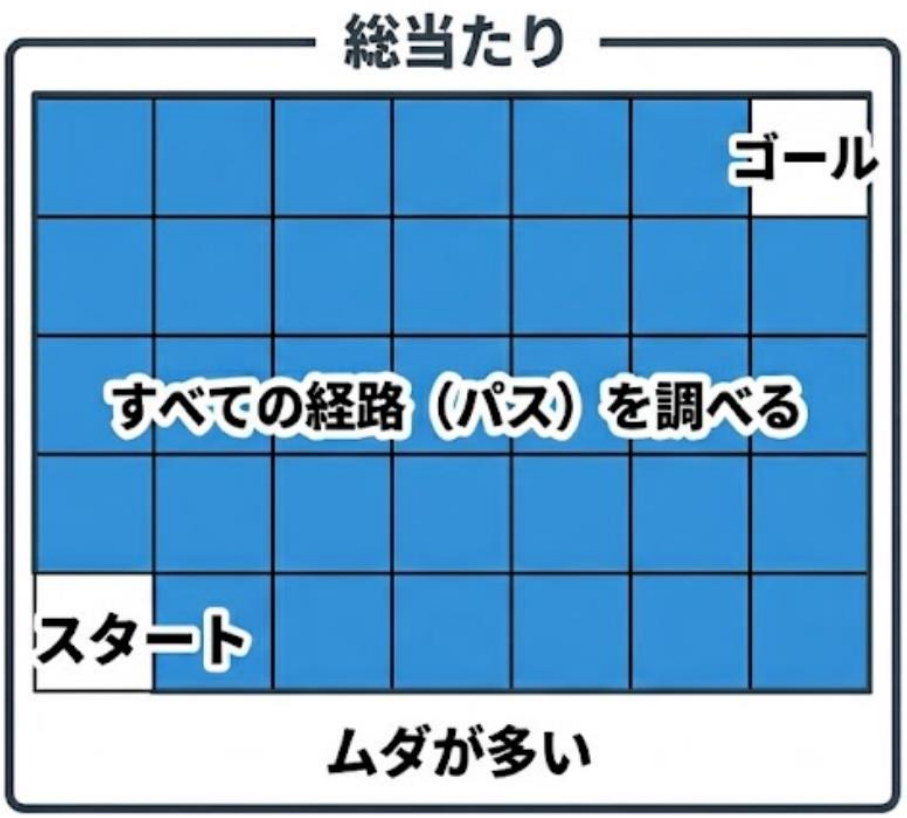
次に調べるマス

候補A：来た道3 ＋
ゴールまでの見積もり4 ＝ 7

候補B：来た道3 ＋
ゴールまでの見積もり6 ＝ 9

合計が小さい候補A（ゴールにゴールに近そう）を先に調べる

A* 法（エイ・スター法）と総当たりの比較



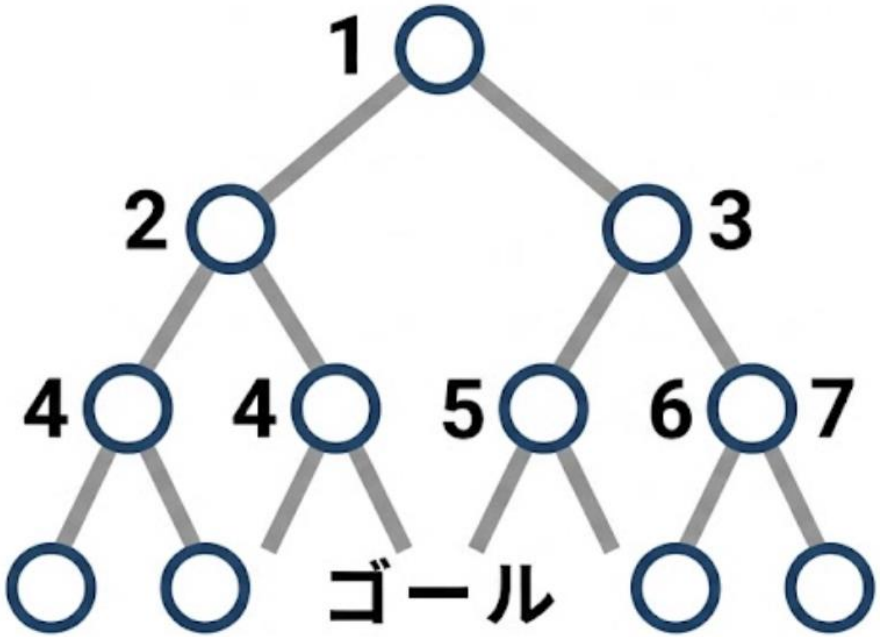
ゴールまでの見積もりを加えるだけで、調べる範囲が大きく減る

A* 法（エイ・スター法）と総当たりの比較

探索とは木をたどる作業であり、A*は『たどる順番』を賢くしたもの

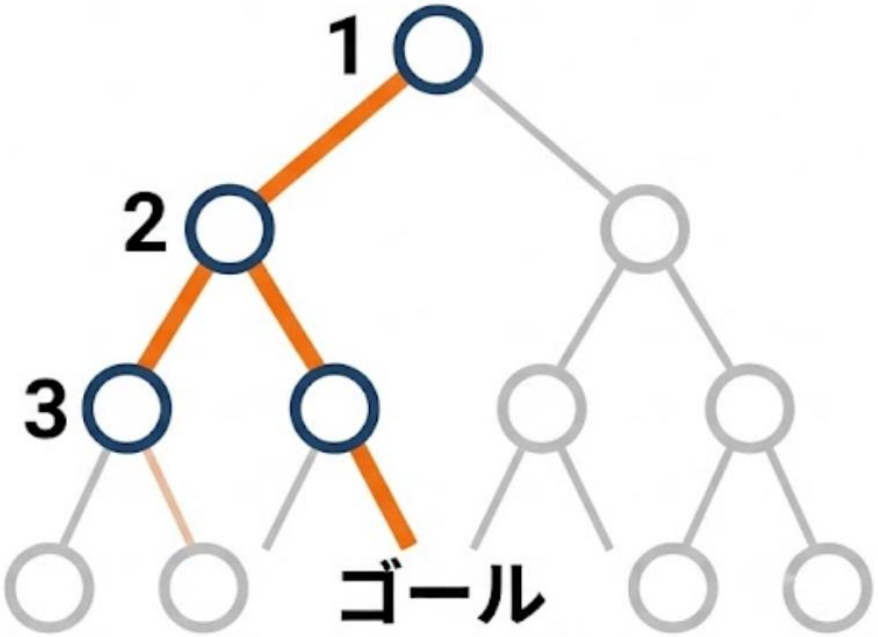
総当たり

手当たり次第にたどる



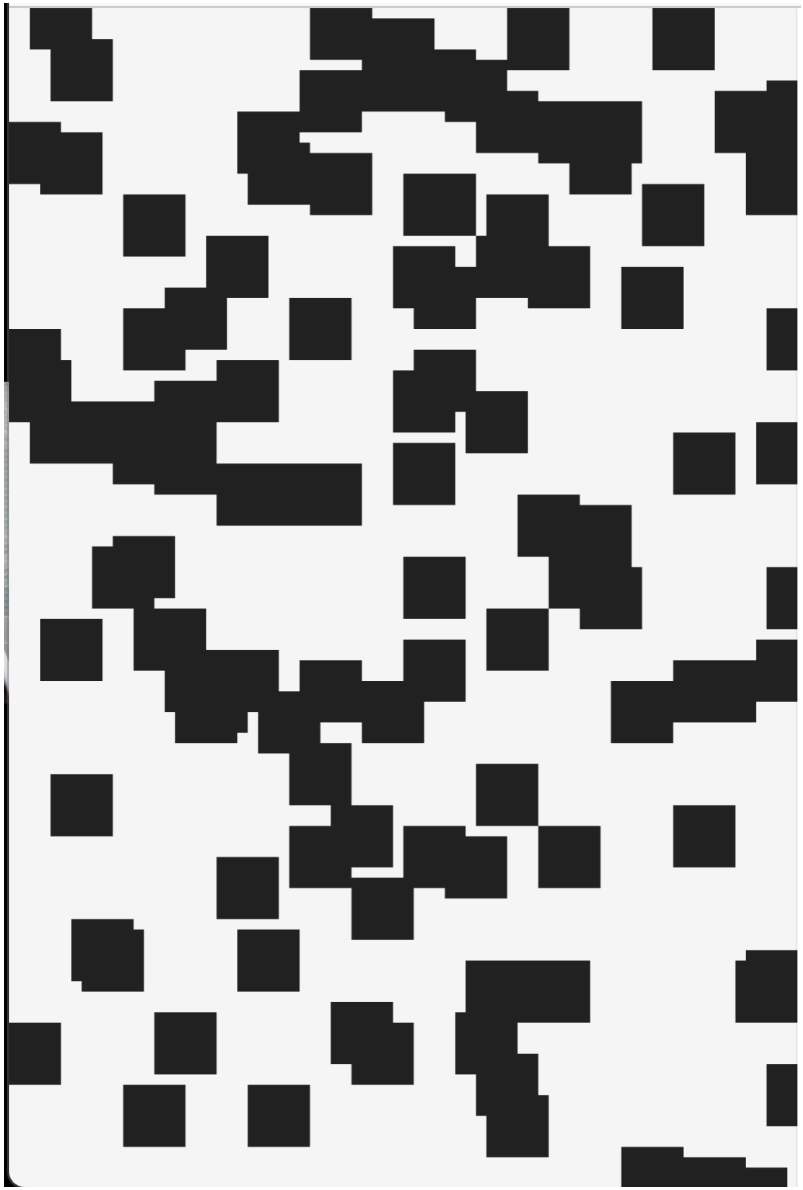
A*

解に近い枝を優先してたどる



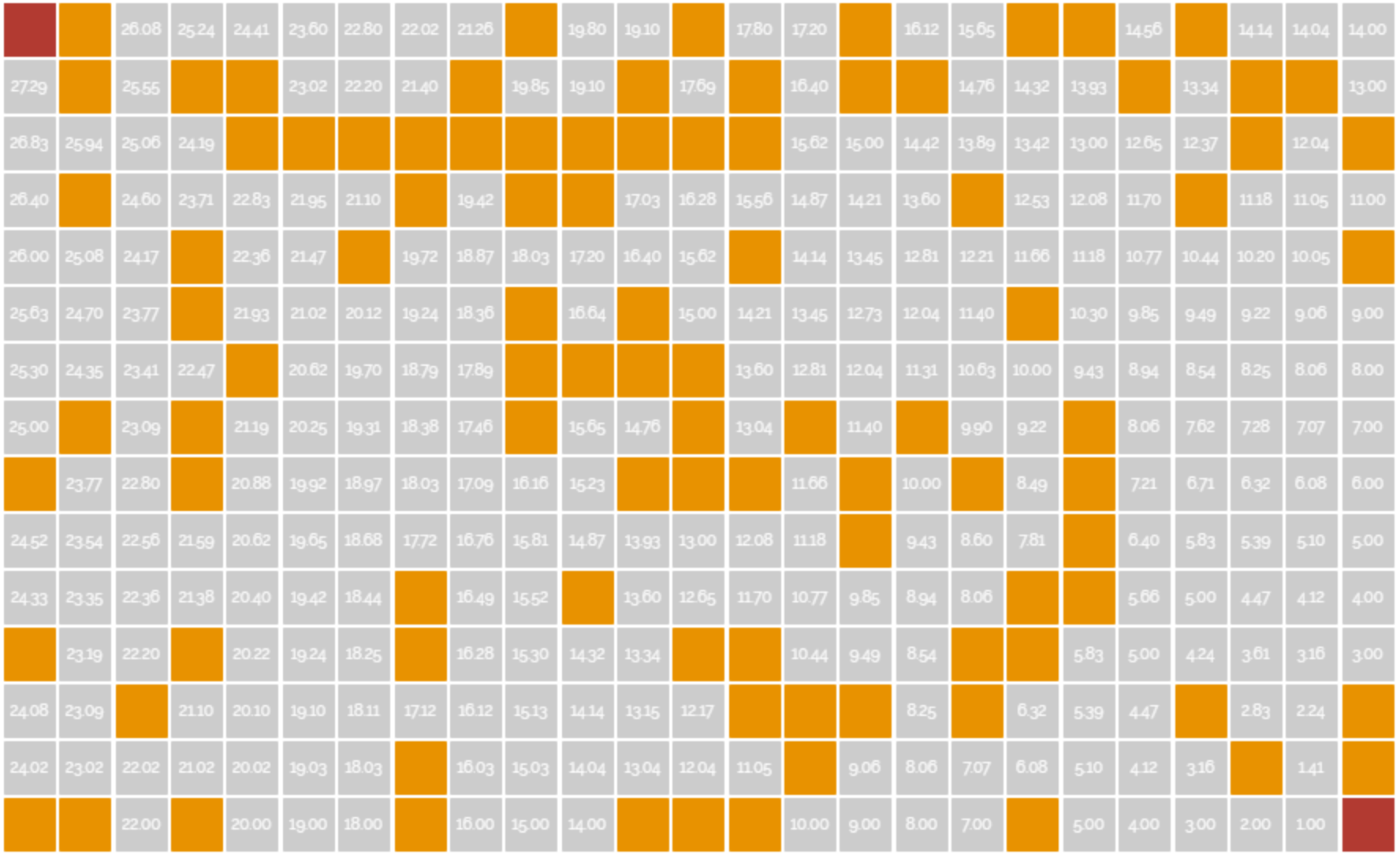
木の構造は同じでも、どの枝から先に調べるかを変えることで無駄なたどりを減らす

A* 法による探索の例



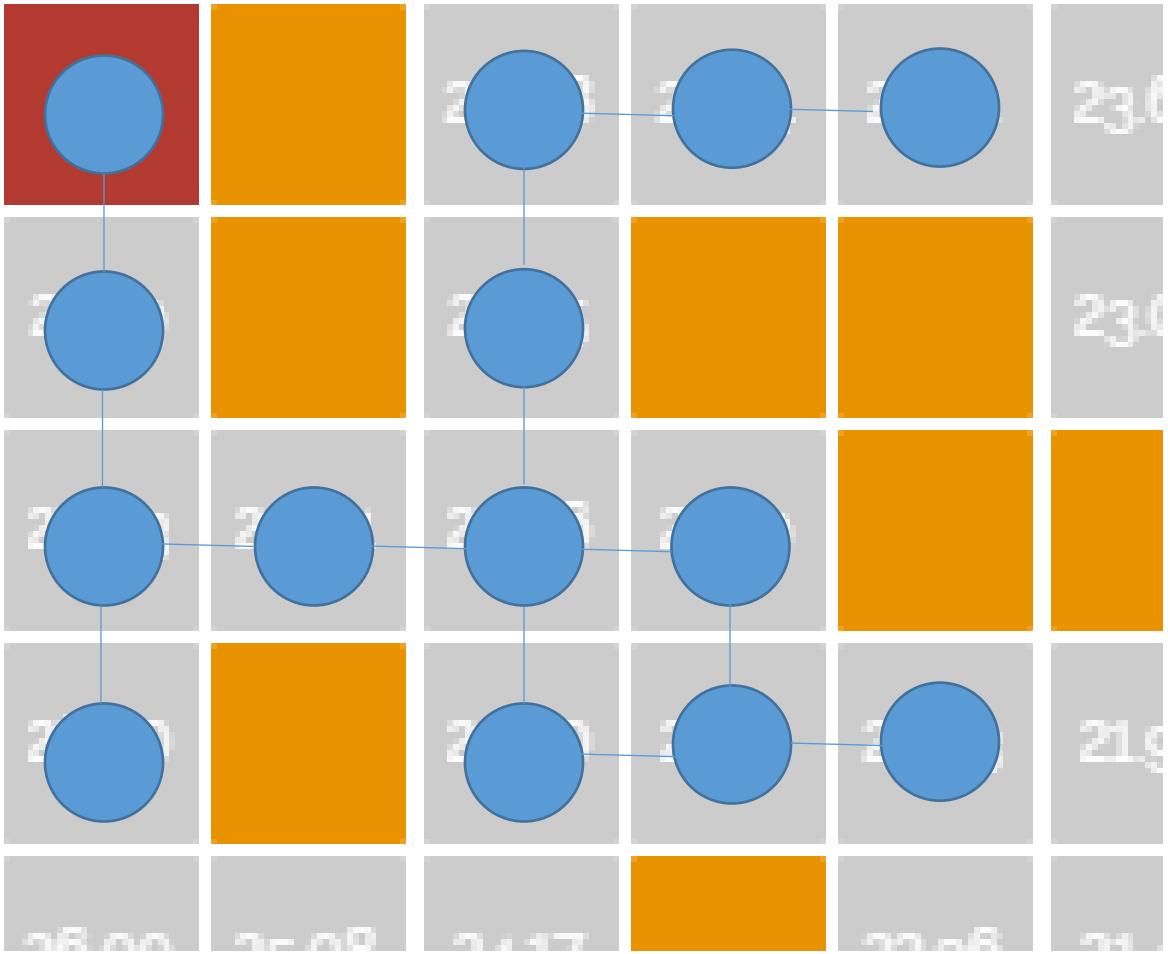
https://www.growingwiththeweb.com/projects/pathfinding-visualiser/

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

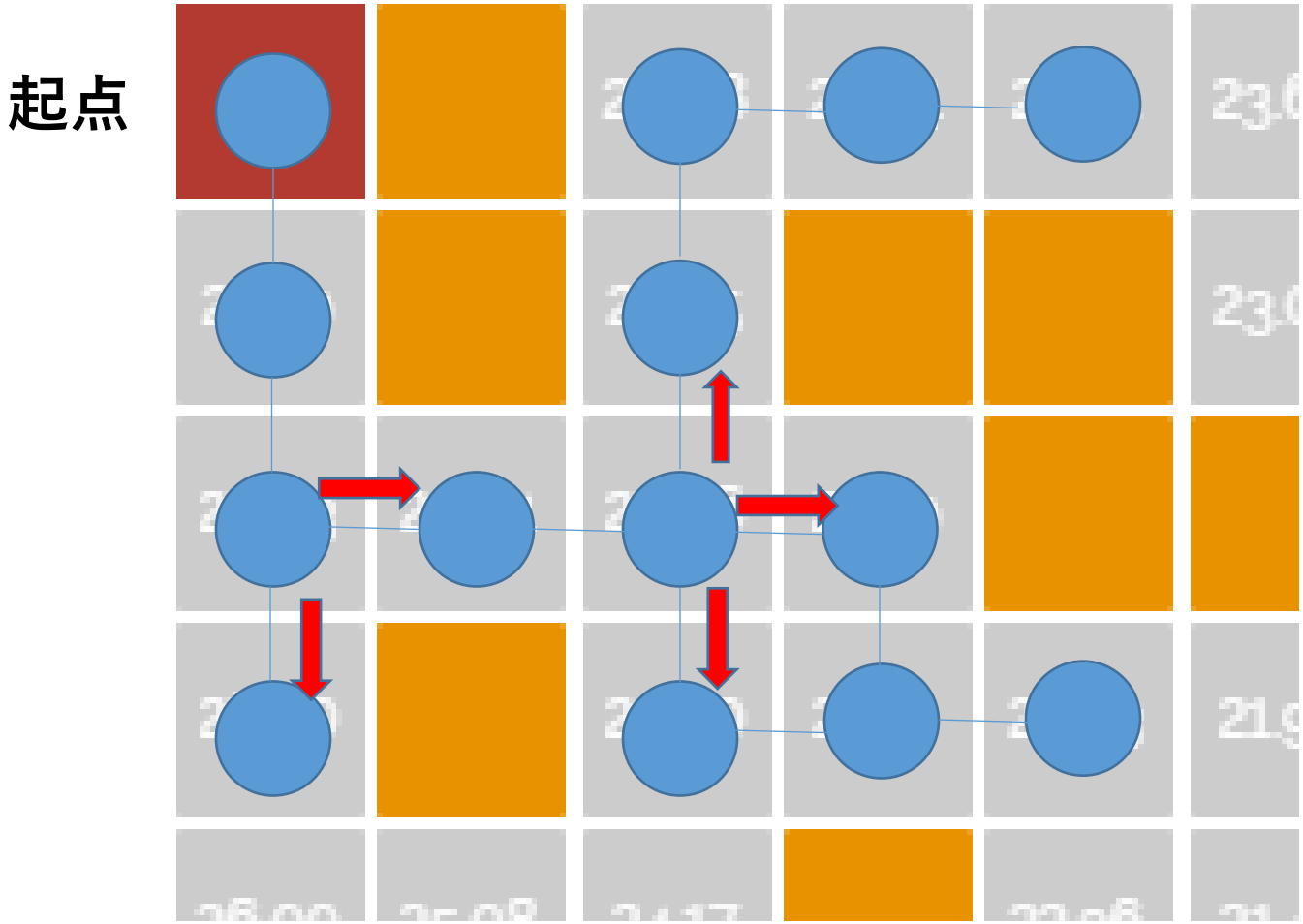
迷路の探索

起点



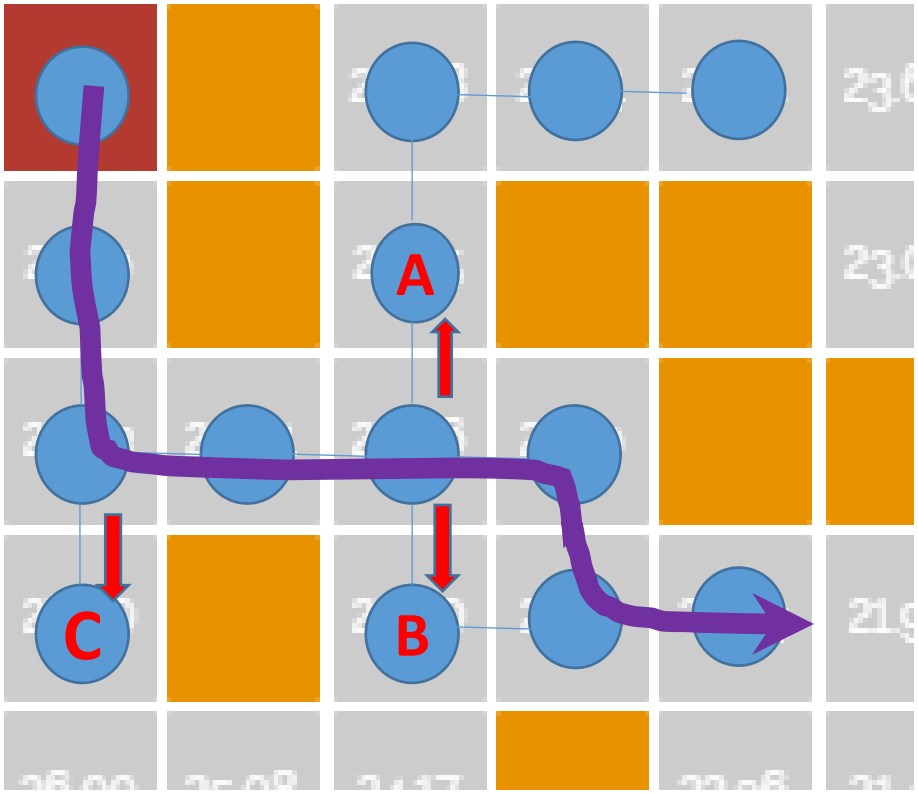
A* 法のルール

１．分岐では，「**解に近いと推定されるパス**」を**優先的に選ぶ**



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

演習

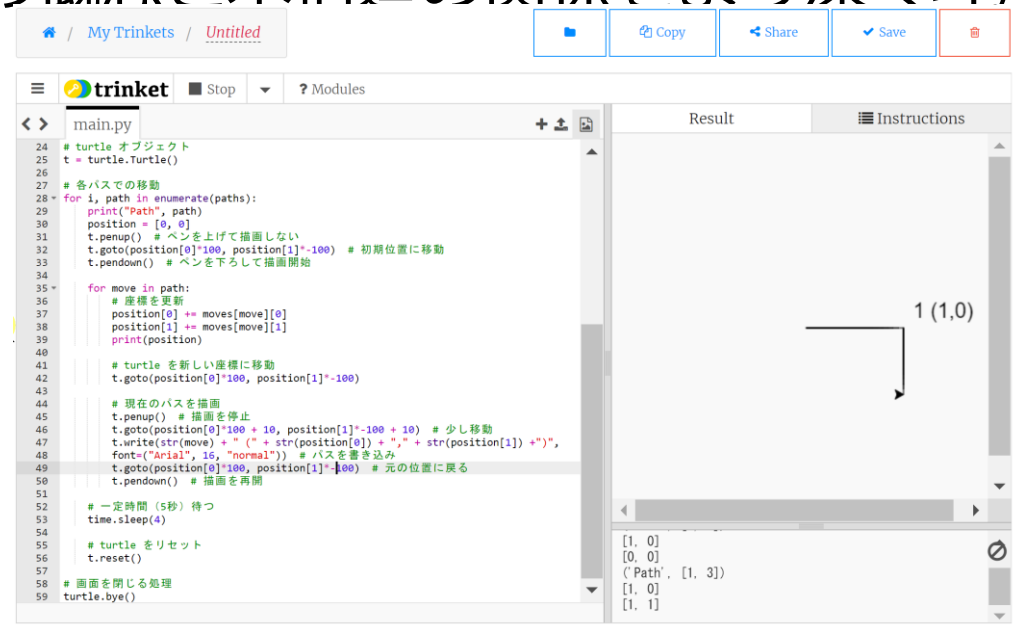
演習１．総当たり

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。



演習１

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。

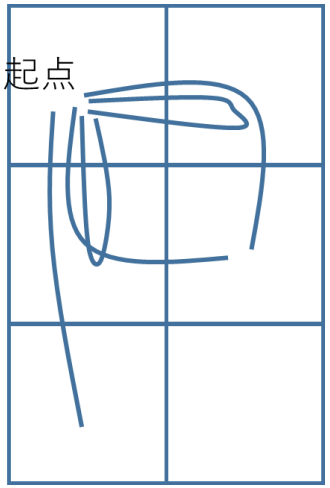
２．最も長いパスと最も短いパスはどれでしたか？

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。

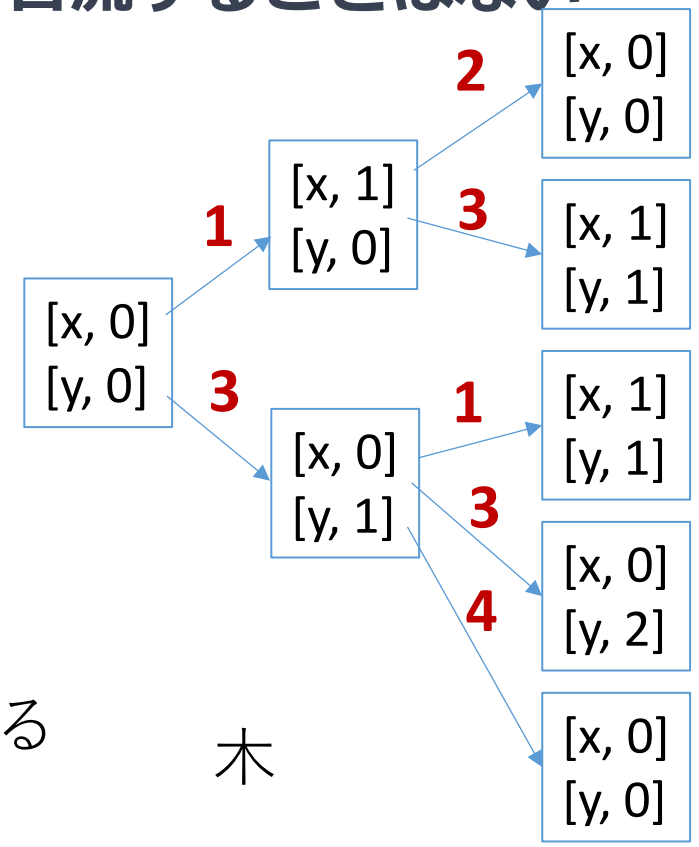
探索におけるパスや木の活用

- **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

演習 2

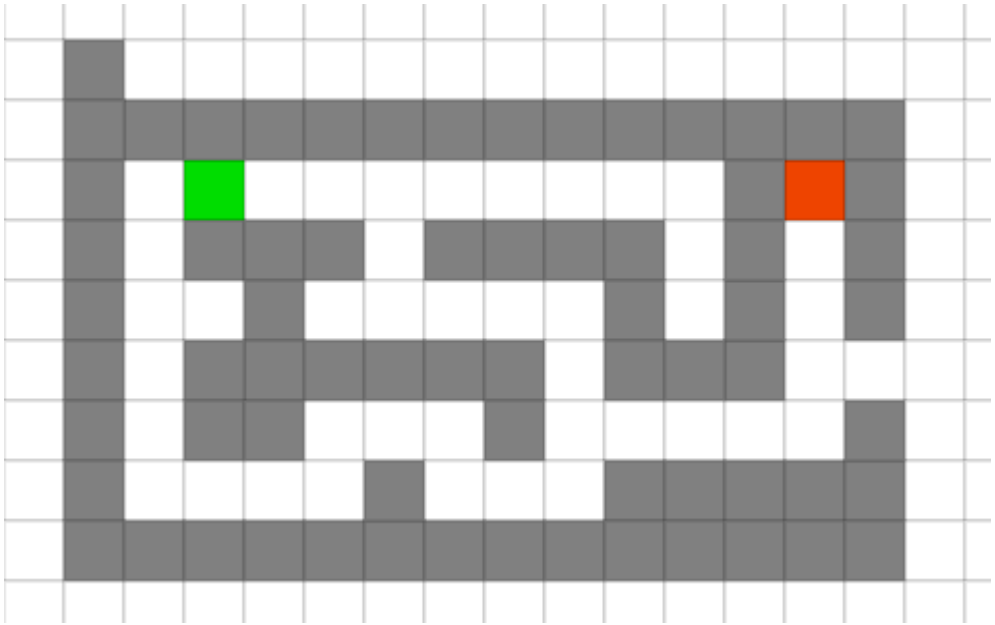
① A*法による迷路探索をシミュレーションできるウェブツールを使用します

Webブラウザで開く

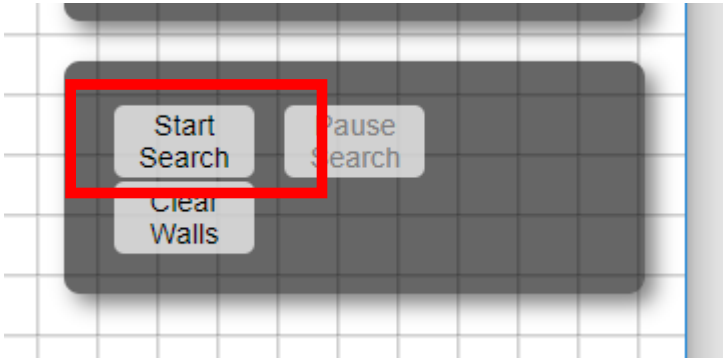
https://qiao.github.io/PathFinding.js/visual/

② マウスを使って迷路を描く

緑：起点　赤：終点



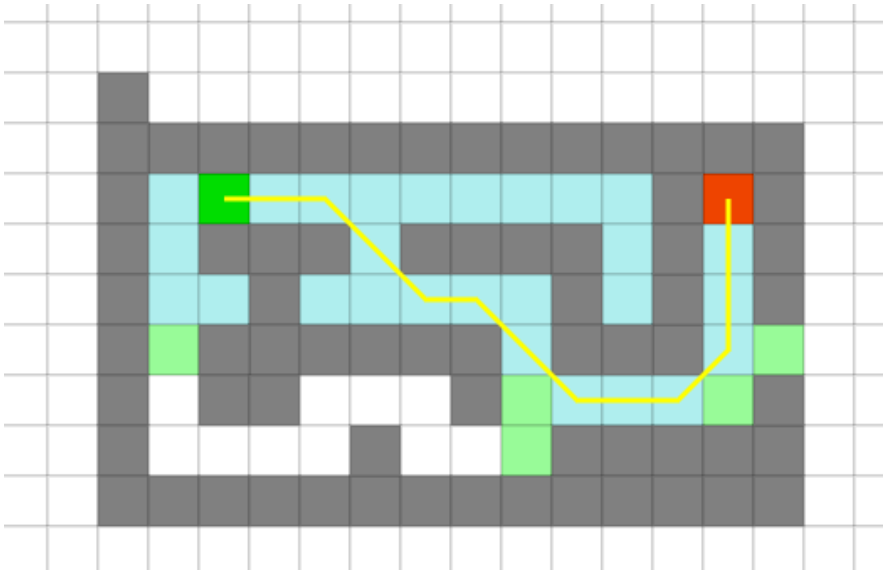
③「Start Search」をクリック



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

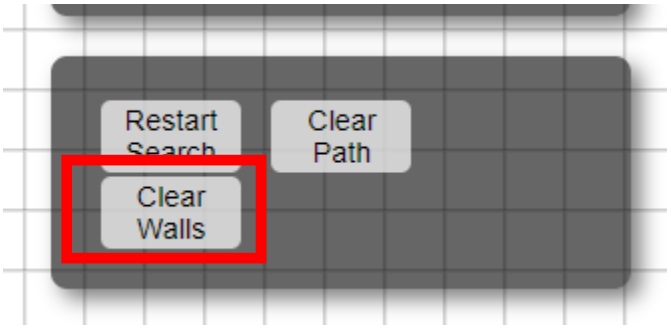


最終的に表示される黄色の線が**最短経路**。
A法がどのように効率的に目標に向かって探索を進めるかを観察できる。

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

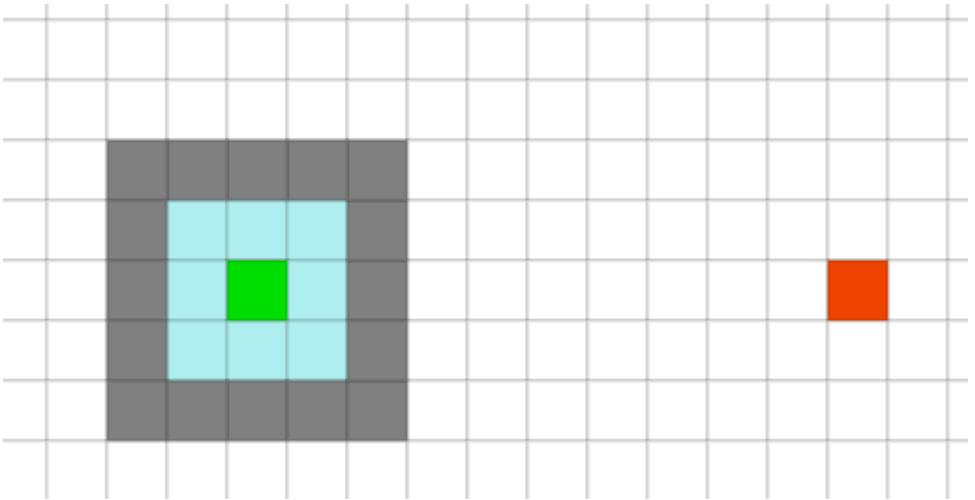
１．　障害物の配置によって探索範囲はどのように変化しますか？

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。

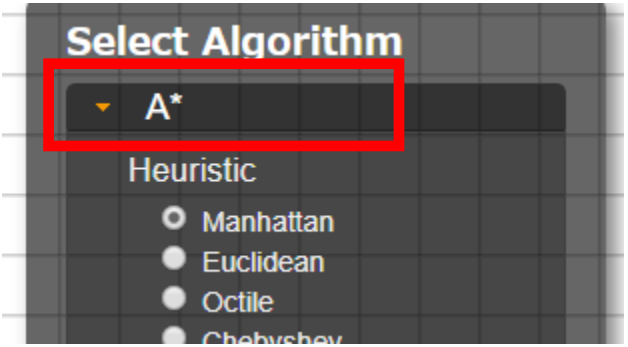
- A*法は最適経路を見つけようとする
- 障害物により直線的経路が遮られると、迂回路を探索
- 障害物が少ない → 狭い範囲で目標到達
- 障害物が多い → 様々な経路を探索 → 探索範囲拡大

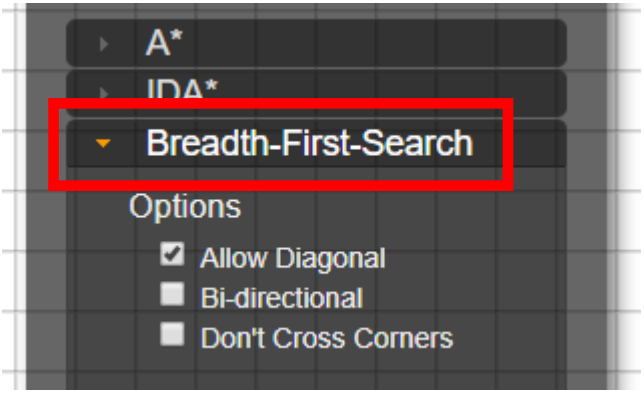
始点と終点の位置関係によって、探索効率はどのように変わりますか？

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。

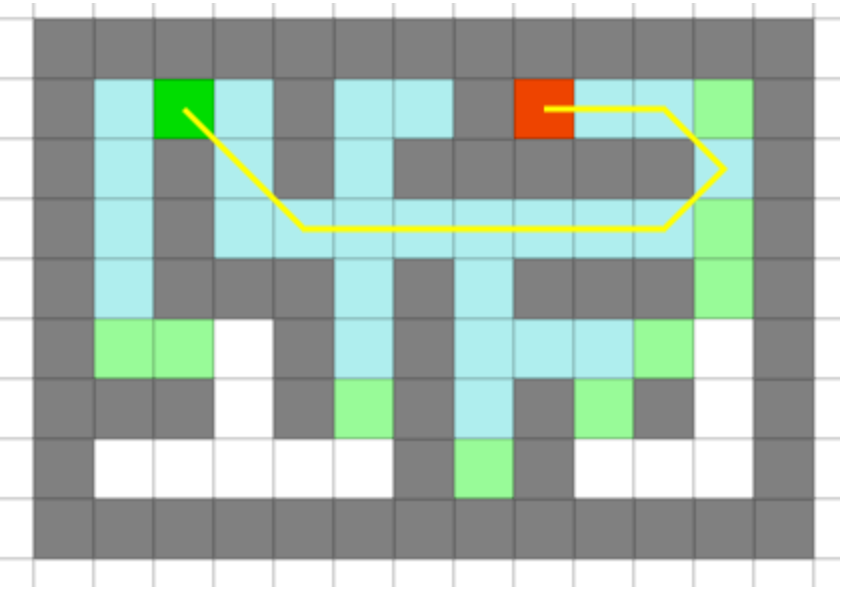
- 直線的 → 推定精度が高い → 効率的探索
- 遠く複雑 → 推定精度が低下 → より多くの経路探索が必要
- 迂回が必要な場合 → 探索範囲拡大 → 効率低下

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