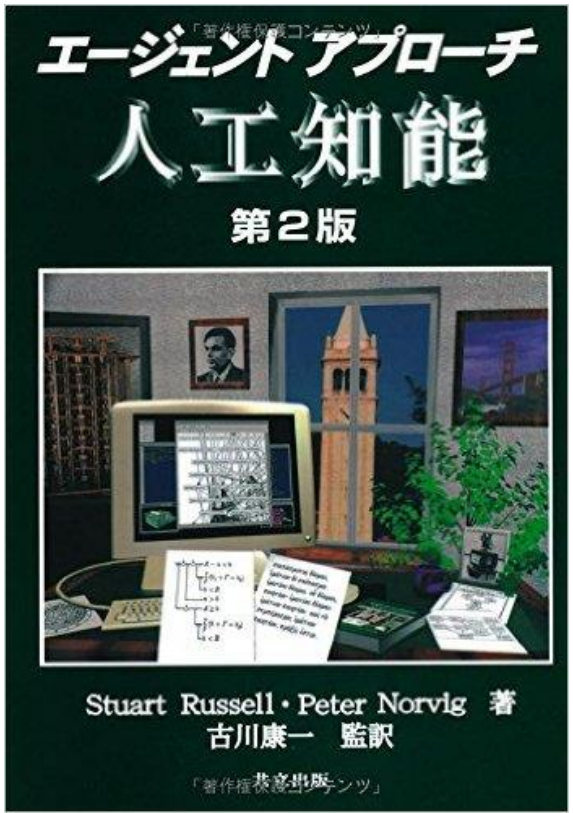
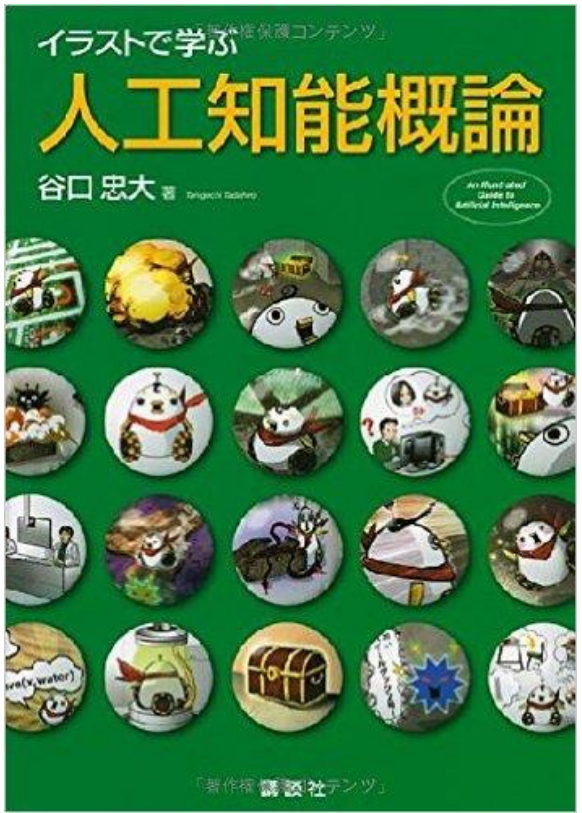
探索

本多 淳也
jhonda@k.u-tokyo.ac.jp
http://www.ms.k.u-tokyo.ac.jp

参考書

- ラッセル，ノーヴィグ：エージェントアプローチ 人工知能 第2版，共立出版，2008年
- 谷口：イラストで学ぶ人工知能概論，講談社，2014年

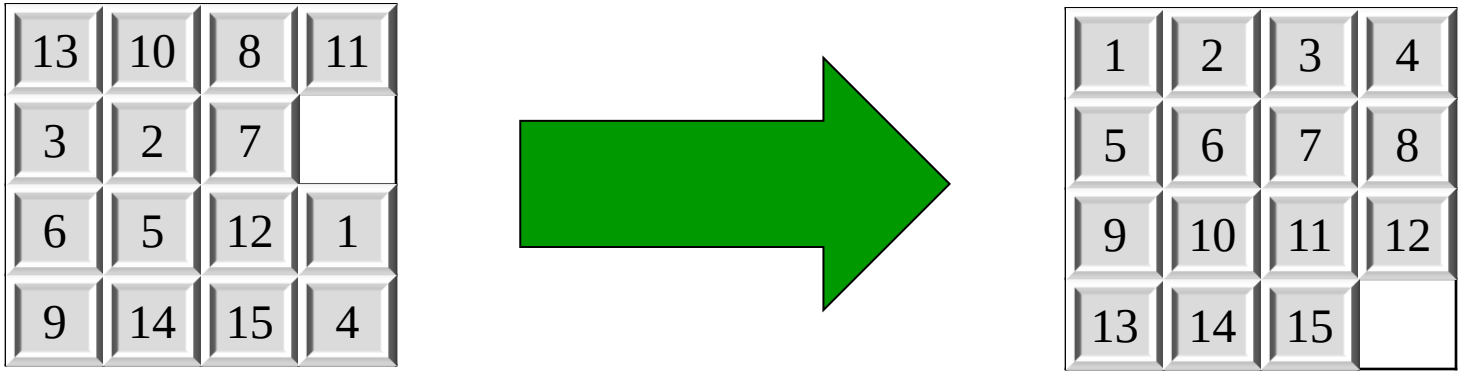




講義の流れ

1. 状態空間
   A) 15パズル
   B) 迷路
2. コスト無しグラフの探索
3. コスト付きグラフの探索
4. ゲーム木の探索

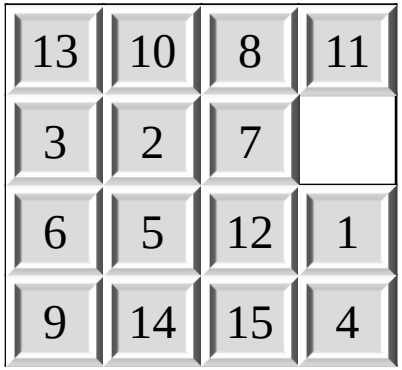
15パズル



- 目的：与えられた配置から元の配置に戻す
  - できればなるべく少ない手数で完了したい（可能か？）
- ルール：
  - 一度に動かせるのは1パネルのみ
  - 空きマスに隣のパネルをずらす操作のみ可能

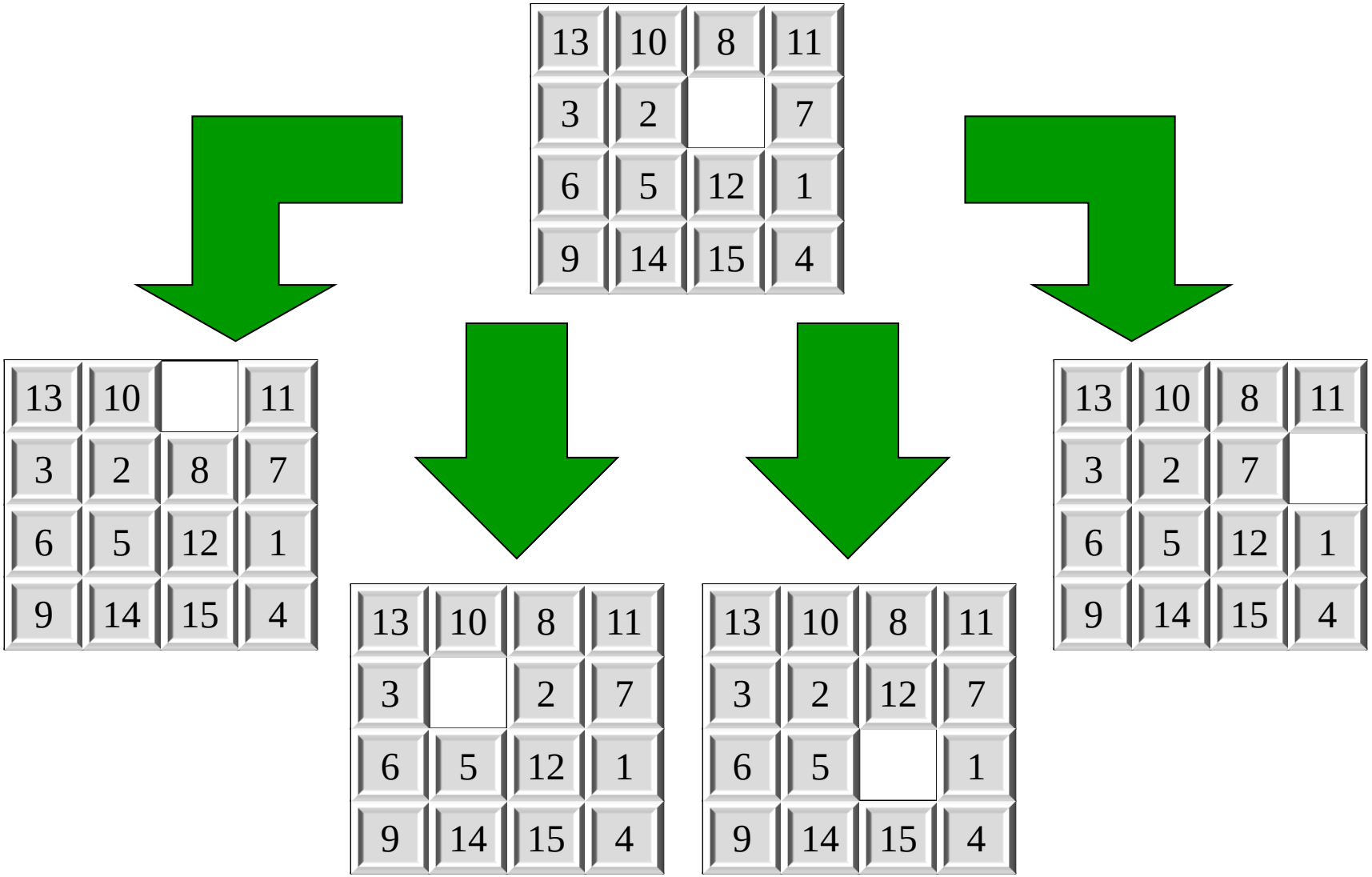
状態空間

- ありうる局面の一つ一つを状態とよび，
  それら全体からなる集合を状態空間という
- 状態の数は 16! 通り
  （実際にはその半数は到達不可能）



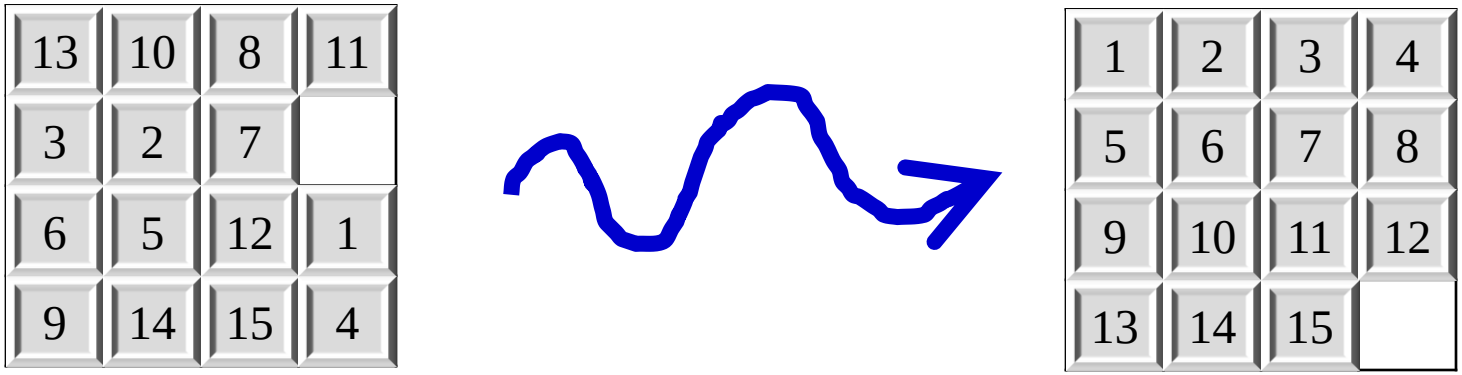
状態遷移

- 以下の状態から遷移できるのは4通り



状態空間の探索

- 目的：初期状態から出発して，許される状態遷移を繰り返し，最終状態へたどり着く



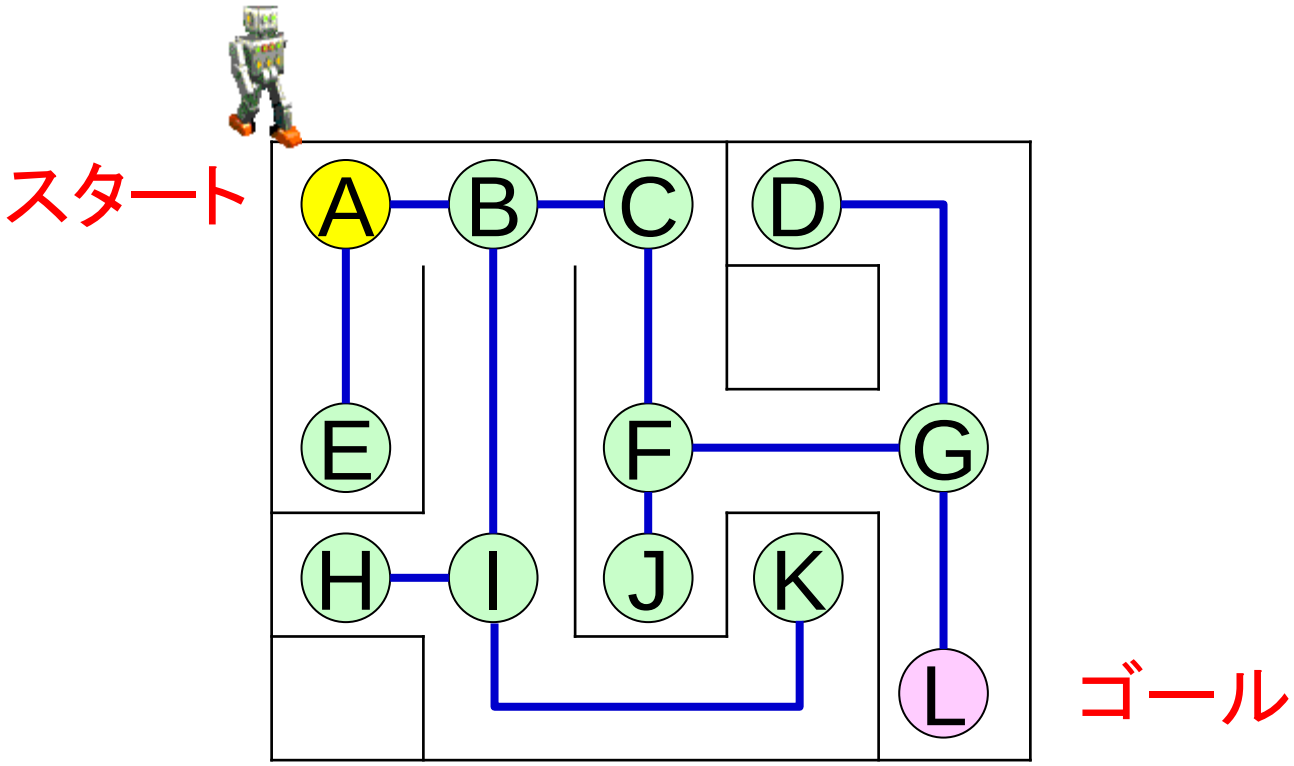
- これは，グラフ(graph)の探索(search)問題と等価（ノード：状態，エッジ：状態遷移）

講義の流れ

1. 状態空間
   A) 15パズル
   B) 迷路
2. コスト無しグラフの探索
3. コスト付きグラフの探索
4. ゲーム木の探索

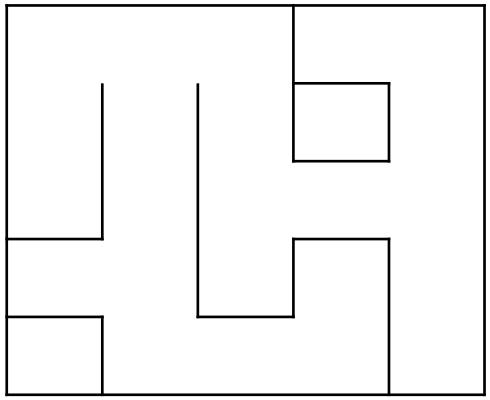
迷路問題

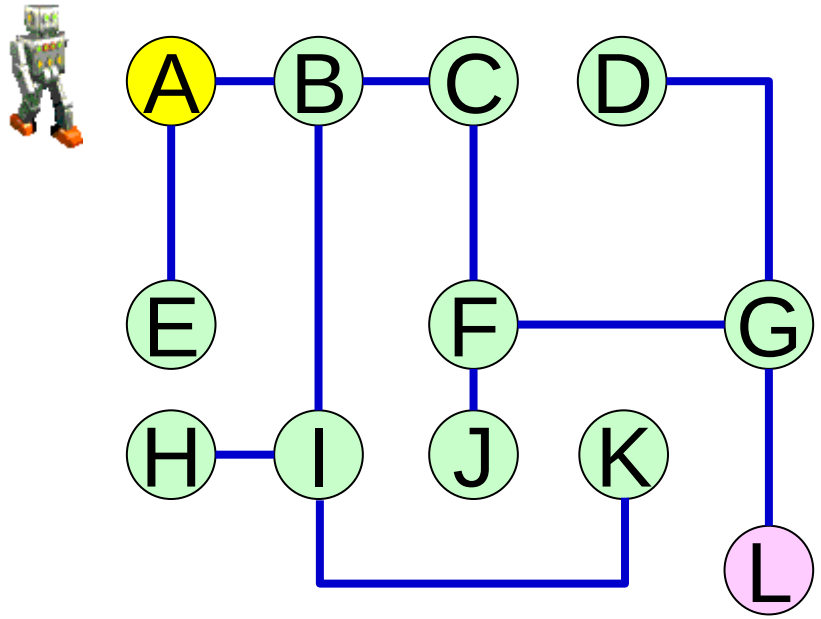
- ロボットをスタートからゴールまで誘導する



状態と行動

- 状態(state) s :ロボットが移動できる場所
- 行動(action) a :ロボットが進む方向

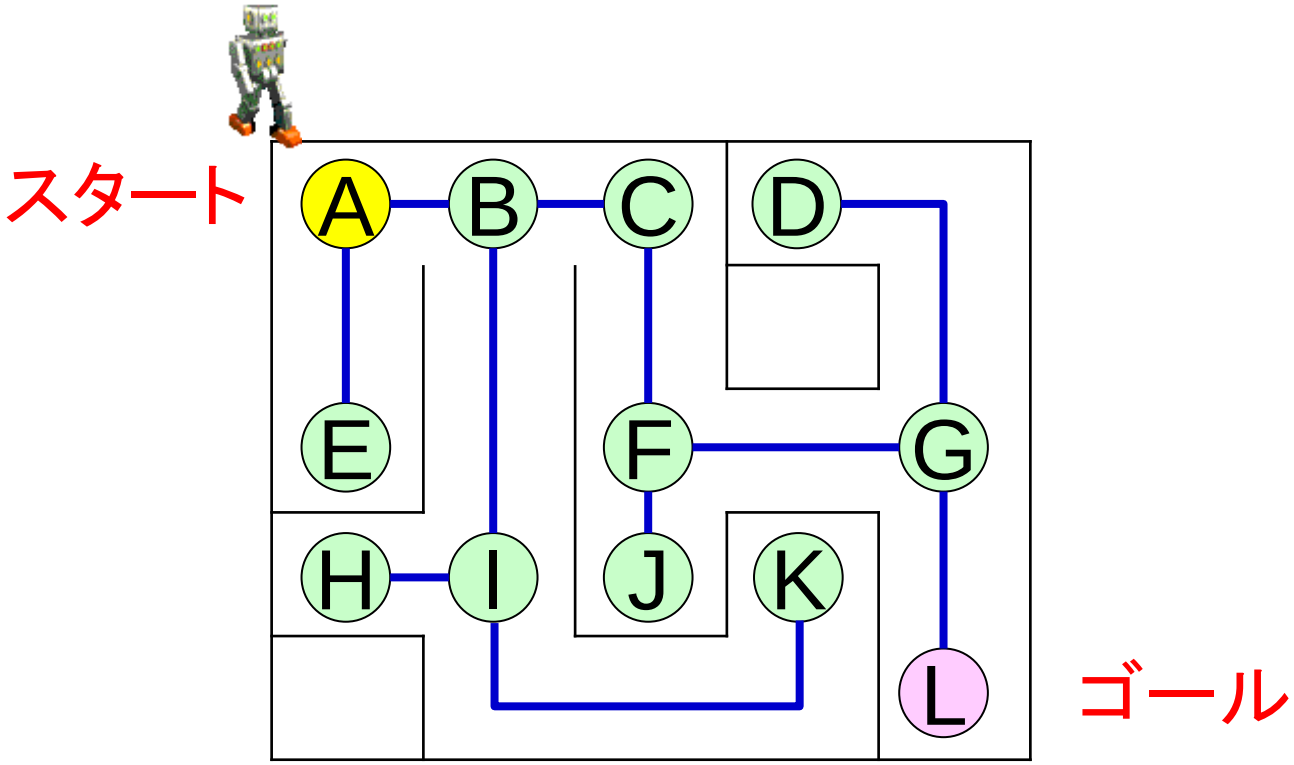




- 状態や行動が連続値を取ることもある
  - 例:ロボットのx-y座標，進む角度

状態空間の探索

- 離散状態，離散行動，確定的状態遷移のとき，迷路問題はグラフの探索問題と等価



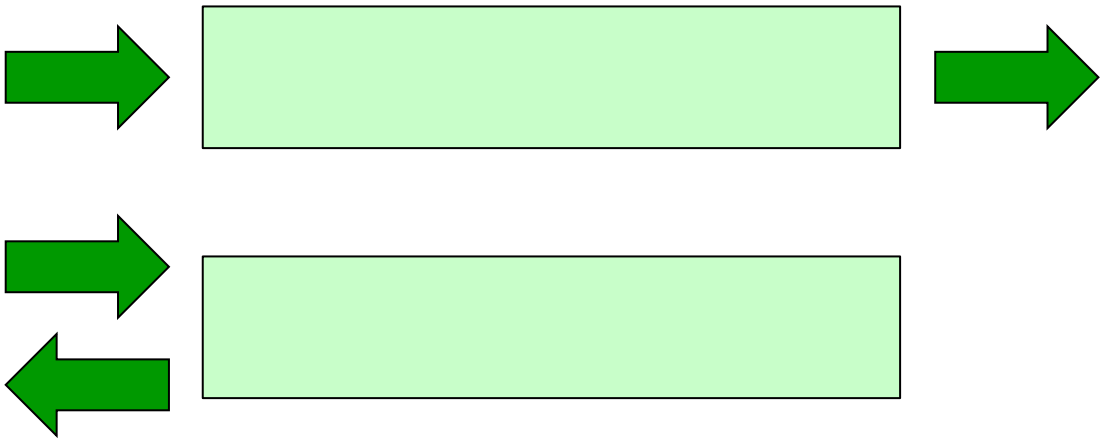
講義の流れ

1. 状態空間
2. コスト無しグラフの探索
3. コスト付きグラフの探索
4. ゲーム木の探索

用語

- オープンリスト(open list):
  - これから探索するノードの候補リスト
- クローズドリスト(closed list):
  - 探索が終わったノードのリスト

- キュー(queue):
  - 先入れ先出し
- スタック(stack):
  - 後入れ先出し

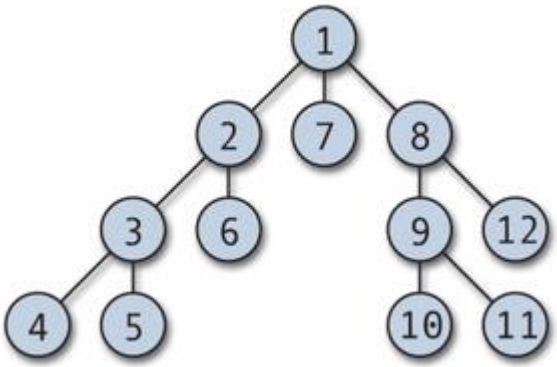


探索の基本アルゴリズム

- 初期化:
  - オープンリストは初期状態のみ
  - クローズドリストは空
- オープンリストが空になるまで以下を繰り返す
  - オープンリストから(何らかの規準で)状態sを取り出す
  - sをクローズドリストに追加する
  - sが最終状態ならば探索終了
  - sから遷移可能でまだクローズドリストに入っていない状態をオープンリストに追加する
- オープンリストからどの要素を選ぶか？

深さ優先探索 (depth-first search)

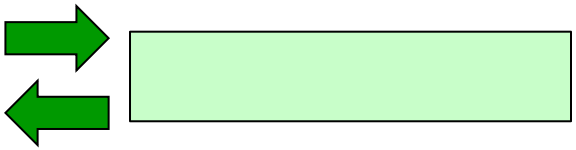
- 行き止まりに当たるまで進み，ゴールが見つからなかったら直近の分岐に戻って別の道を探す
- ☺ メモリ使用量が少ない
- ☹ ゴールが近くにあっても，他の深い別れ道に迷い込むと時間がかかる
- ☹ ゴールが複数ある時，一番近くのものが見つかるとは限らない



図はWikipediaより

深さ優先探索アルゴリズム

- オープンリストはスタックにする（後入れ先出し）
- 初期化：
  - オープンリストは初期状態のみ
  - クローズドリストは空



- オープンリストが空になるまで以下を繰り返す
  - オープンリストの先頭の状態sを取り出す
  - sをクローズリストに追加する
  - sが最終状態ならば探索終了
  - sから遷移可能でまだクローズリストに入っていない状態をオープンリストの先頭に追加する

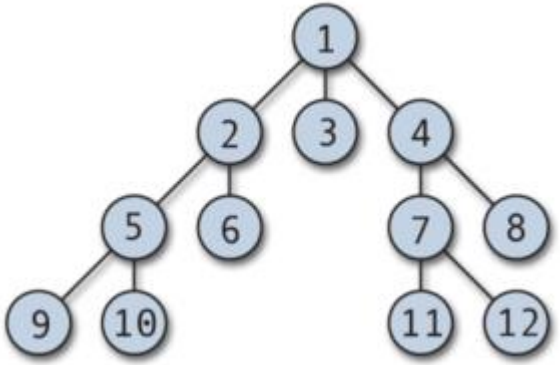
幅優先探索 (breadth-first search)

- 分かれ道に来たらそれぞれの道を一歩ずつ進み，ゴールが見つからなかったらそれぞれの道をもう一歩ずつ進む

☺ ゴールが近くにある時，早く見つかる．

☺ ゴールが複数ある時，一番近くのものが見つかる

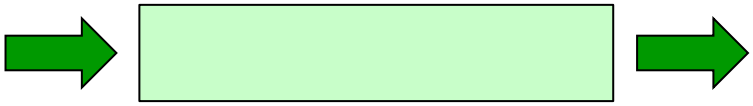
☹ 分かれ道での分岐数が多いとメモリ使用量が多い



図はWikipediaより

幅優先探索アルゴリズム

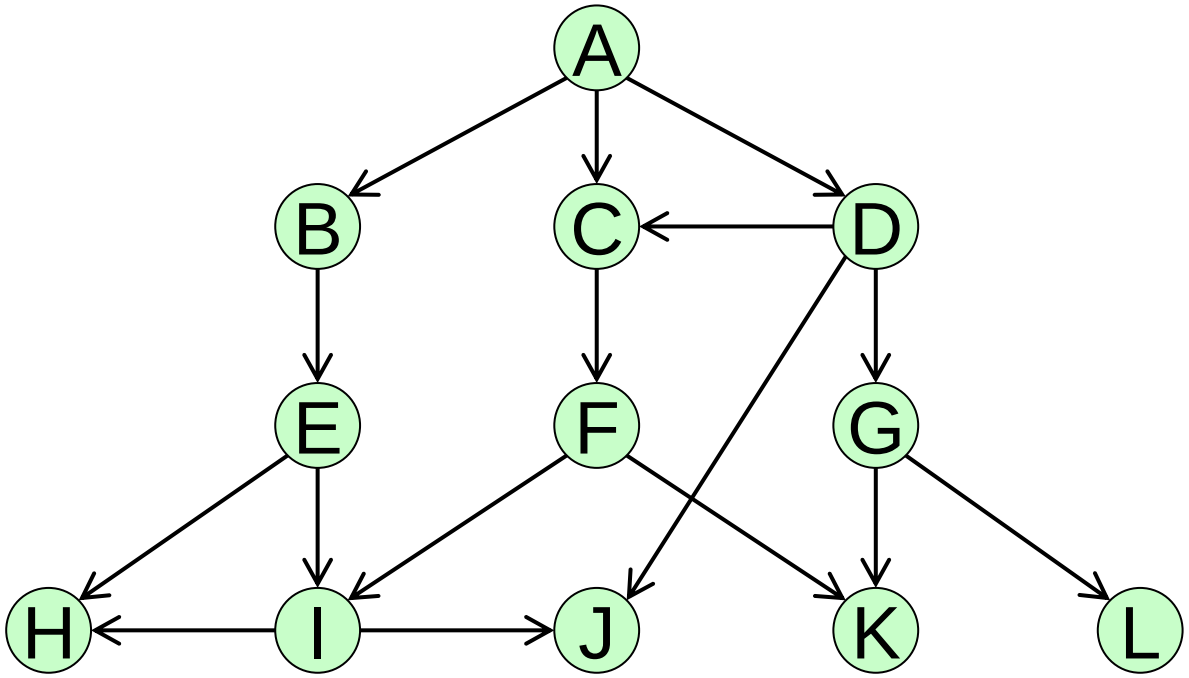
- オープンリストはキューにする（先入れ先出し）
- 初期化：



  - オープンリストは初期状態のみ
  - クローズドリストは空
- オープンリストが空になるまで以下を繰り返す
  - オープンリストの末尾の状態sを取り出す
  - sをクローズリストに追加する
  - sが最終状態ならば探索終了
  - sから遷移可能でまだクローズリストに入っていない状態をオープンリストの先頭に追加する

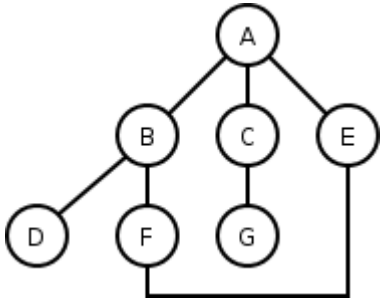
演習

- 有向グラフ(directed graph)に対しても同様に探索を行える
- 以下の有向グラフに対して，ノードAから深さ優先探索・幅優先探索で訪れるノードの順をそれぞれ求めよ．ただし，左側のノードを優先する



クローズドリストの有無

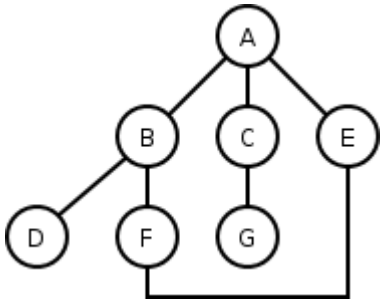
- クローズドリストと照合するのは時間・メモリがかかる
- クローズドリストなしで探索を行うと:
  - 幅優先探索:ABCEDFG...
  - 深さ優先探索:ABDFEABDFEABDFE…
- 深さ優先探索は無限ループに陥る



Wikipediaより

反復深化探索 (iterative deepening search)

- 深さに制限をつけて深さ優先探索を行い，徐々に深さを深くしていく
- 右の例では
  - 深さ制限1: ABCE
  - 深さ制限2: ABDFCGEF
  - 深さ制限3: ABDFECGEFB



Wikipediaより

☺ ゴールが近くにある時，早く見つかる

☺ ゴールが複数ある時，一番近くのものが見つかる

☹ 同じノードを何度も訪れる（分岐が多いと影響小）

コスト無し探索：まとめ

- 深さ優先探索，幅優先探索，反復深化探索は，グラフに関する特別な知識を使わずにオープンリストから状態を選択することから，ブラインド探索(blind search)とよばれる

	深さ優先探索	幅優先探索	反復深化探索
完全性（必ず解が見つかるか）(completeness)	mが有限ならYes	Yes	Yes
時間計算量 (time complexity)	$O(b^m)$	$O(b^d)$	$O(b^d)$
空間計算量 (space complexity)	$O(bm)$	$O(b^d)$	$O(bd)$
最適性（一番近くの解が見つかるか）(optimality)	No	Yes	Yes

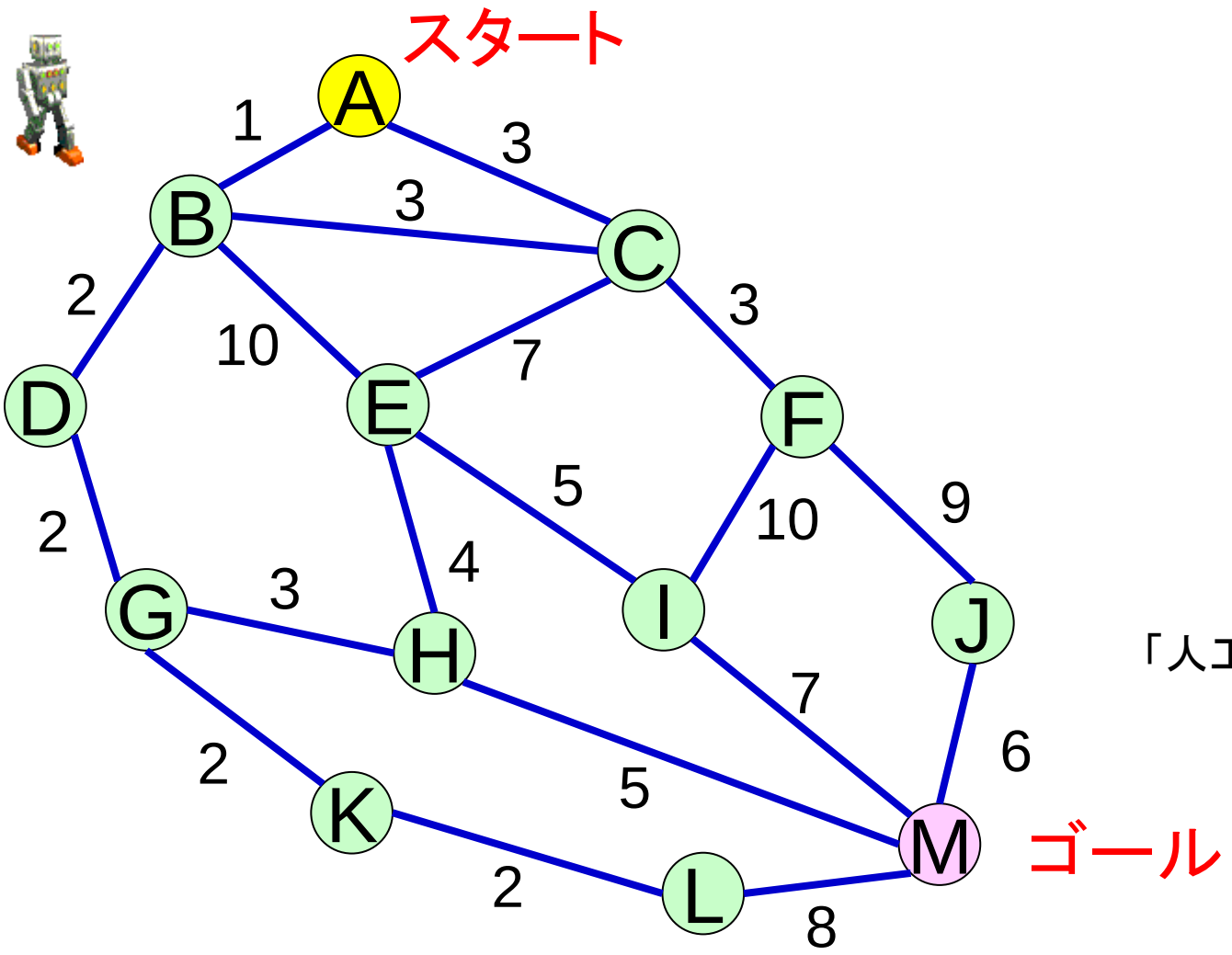
b：最大分岐数，d：一番浅い解の深さ，m：最大の深さ

講義の流れ

1. 状態空間
2. コスト無しグラフの探索
3. コスト付きグラフの探索
4. ゲーム木の探索

コスト付きグラフ

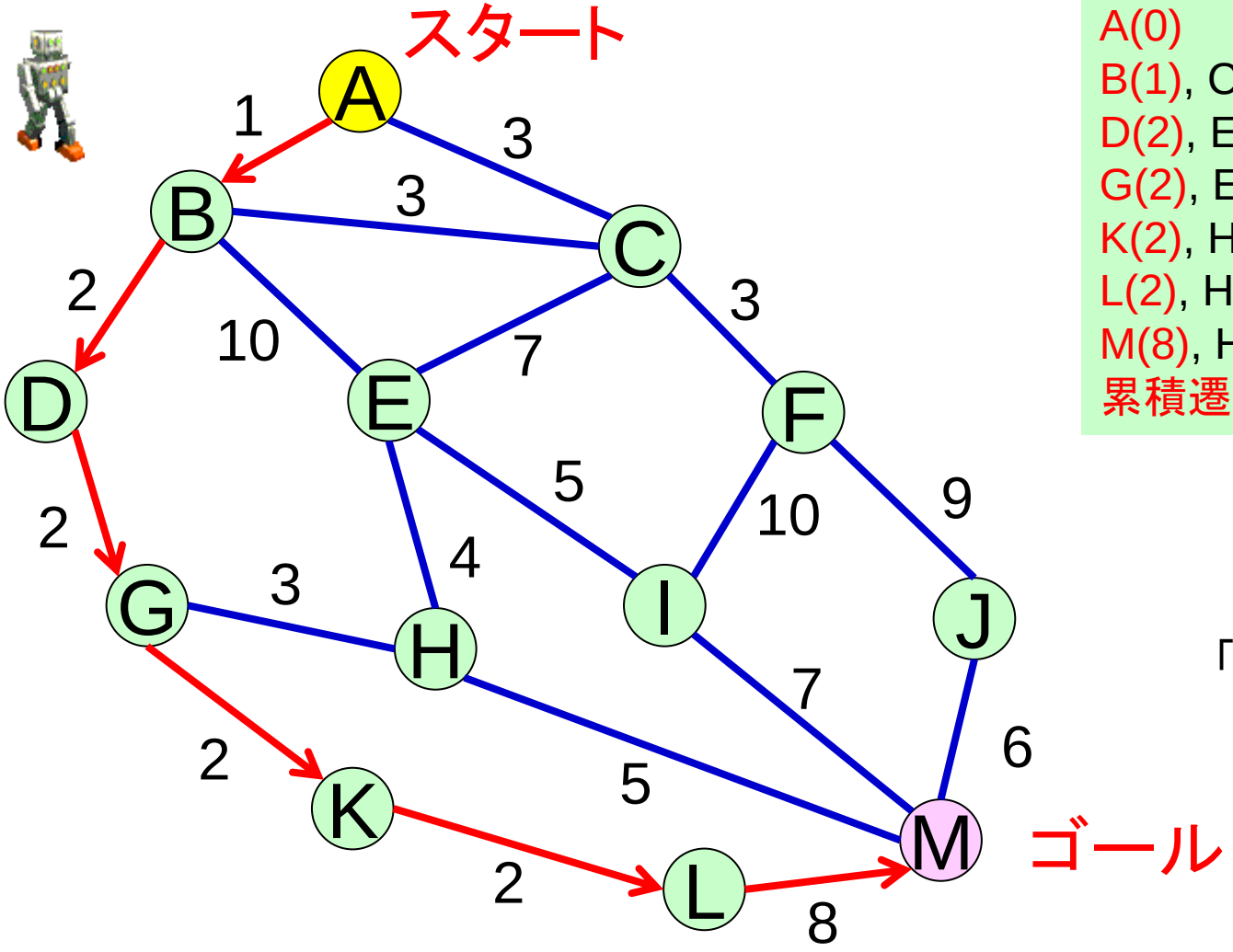
- 各エッジに遷移コストが割り当てられている



木下哲男
「人工知能と知識処理」
昭晃堂より

貪欲探索(greedy search)

- 現在の状態からの遷移コストが最小の状態を選ぶ
  - 深さ優先探索に対応

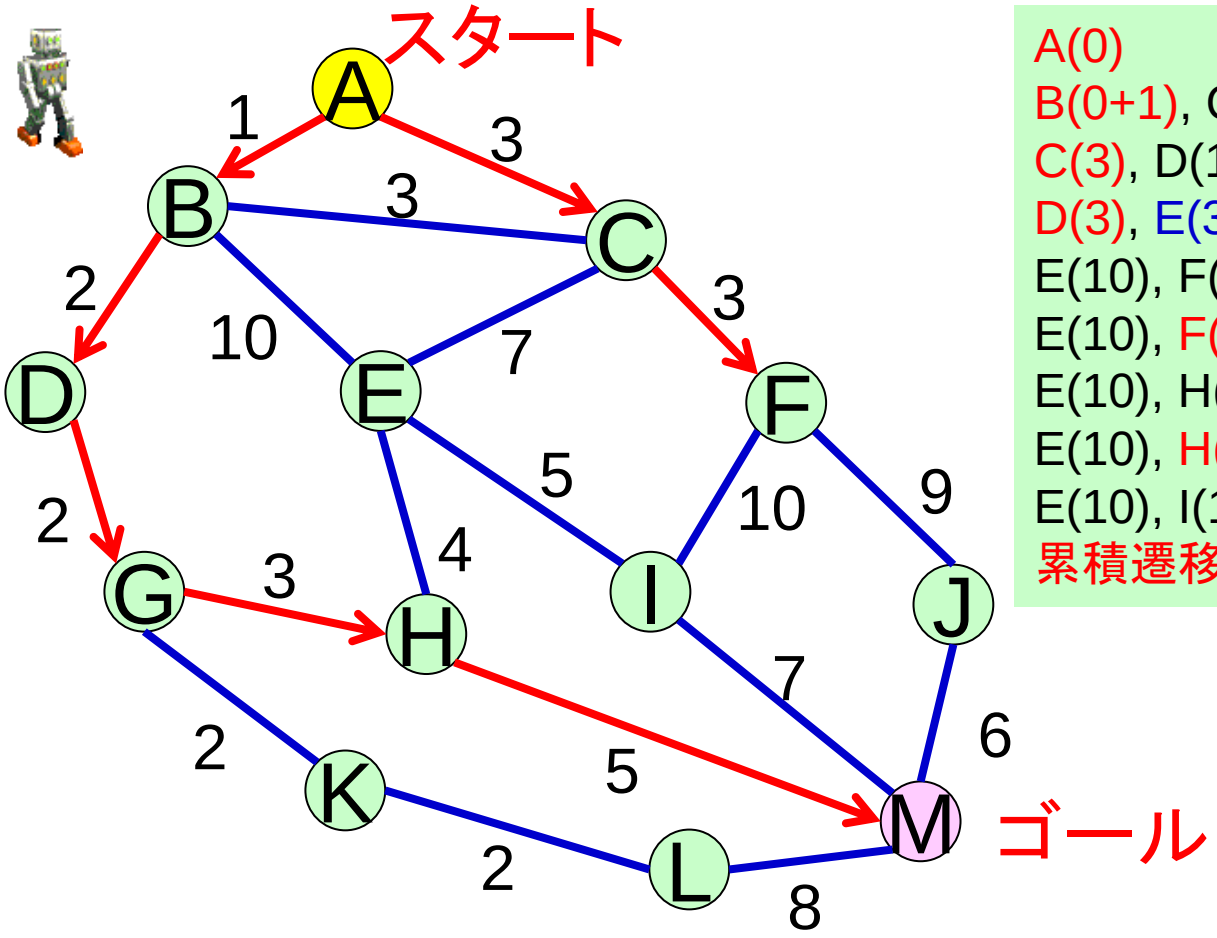


A(0)
B(1), C(3)
D(2), E(10), C(3)
G(2), E(10), C(3)
K(2), H(3), E(10), C(3)
L(2), H(3), E(10), C(3)
M(8), H(3), E(10), C(3)
累積遷移コスト17

木下哲男
「人工知能と知識処理」
昭晃堂より

最適探索(optimal search)

- 初期状態からの遷移コスト和が最小の状態を選ぶ
  - 各コストが1だと幅優先探索になる
  - ダイクストラ法(Dijkstra's algorithm)ともよばれる



A(0)
B(0+1), C(0+3)
C(3), D(1+2), E(1+10)
D(3), E(3+7), F(3+3)
E(10), F(6), G(3+2)
E(10), F(6) , H(5+3), K(5+2)
E(10), H(8), I(6+10), J(6+9), K(7)
E(10), H(8) , I(16), J(15), L(7+2)
E(10), I(16), J(15), L(9), M(8+5)
累積遷移コスト13

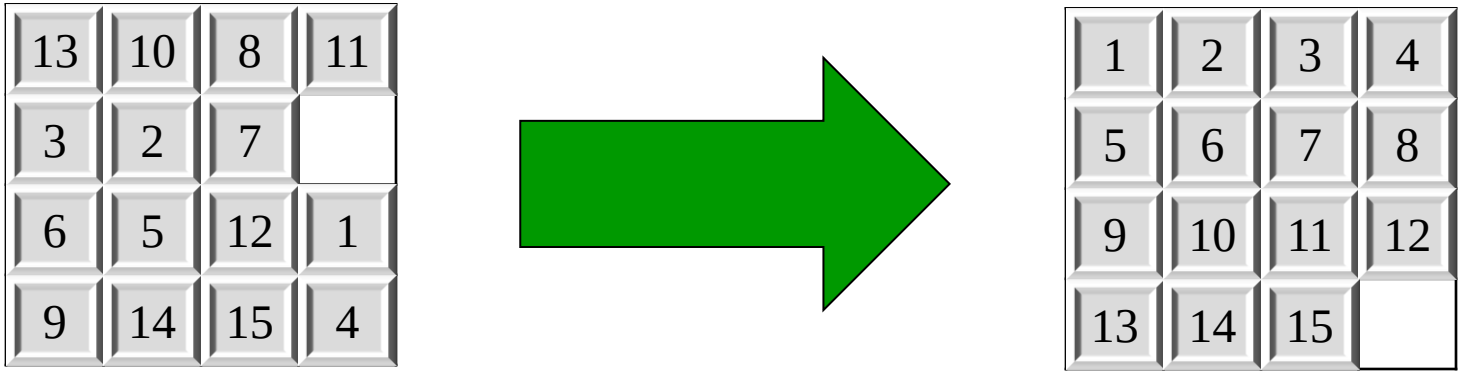
木下哲男
「人工知能と知識処理」
昭晃堂より

様々な探索法

- ブラインド探索：グラフに関する特別な知識を使わずにオープンリストから状態を選択する
  - 貪欲探索（次のコスト最小）
  - 最適探索（累積コスト最小）
- ヒューリスティック探索
  （最良優先探索, best-first search）：
  グラフに関する何らかの知識を使ってオープンリストから適切と思われる状態を選択
  - 貪欲最良優先探索
  - A*探索

ヒューリスティクス(heuristics)：
問題に関する事前知識を使って，最適とは限らないが，十分に精度の良い解を簡便に得る方法

15パズル



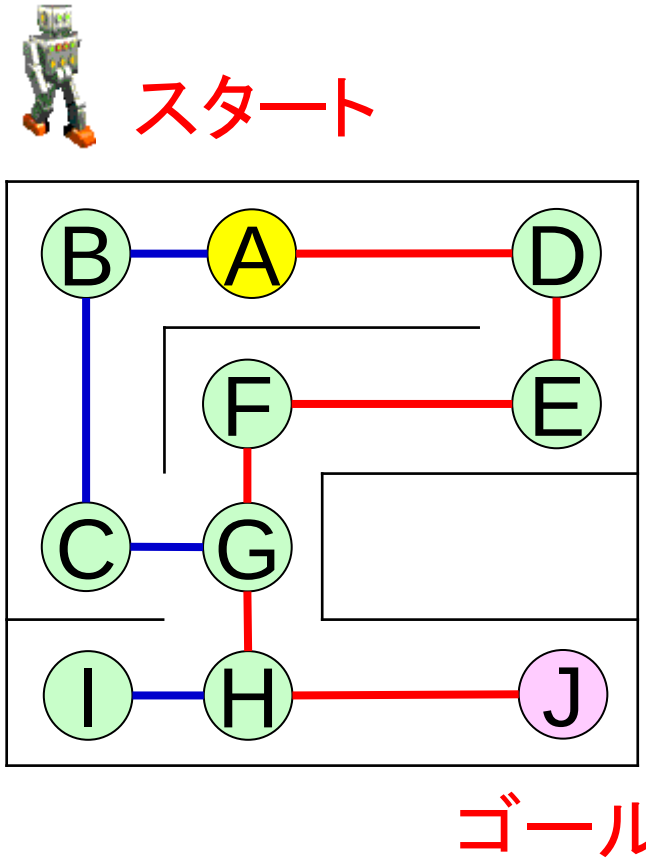
- ダイクストラ法での探索は非常に遅い
- 一般に24パズル，35パズル，48パズル,...の最短手順の探索は計算量理論的にはほぼ不可能（NP困難）
- 最短手順を諦める / 最悪ケースの場合は諦めることによって良い手順を高速に求めたい

貪欲最良優先探索

- $\widehat{h}(s)$ を最小にする状態を選ぶ
  - $h(s)$: s から最終状態までの遷移コスト和の最小値
  - $\widehat{h}(s)$: $h(s)$の推定（ヒューリスティック関数）
    - 一度オープンリストに入った s の評価値更新は不要
- ヒューリスティック関数 $\widehat{h}(s)$ の選び方：
  - ユーザが事前知識により構築（例：直線距離）
  - データから機械学習により自動構築
- 一般には完全性も最適性もないが，実用上は（そこそこ）うまくいくことが（それなりに）多い

貪欲最良優先探索

- 例: $\widehat{h}(s)$としてゴールまでのマンハッタン距離（XY各方向への距離の和）を使用



A(5)
B(6), D(3)
B(6), E(2)
B(6), F(4)
B(6), G(3)
B(6), C(4), H(2)
B(6), C(4), I(3), J(0)
累積遷移コスト9

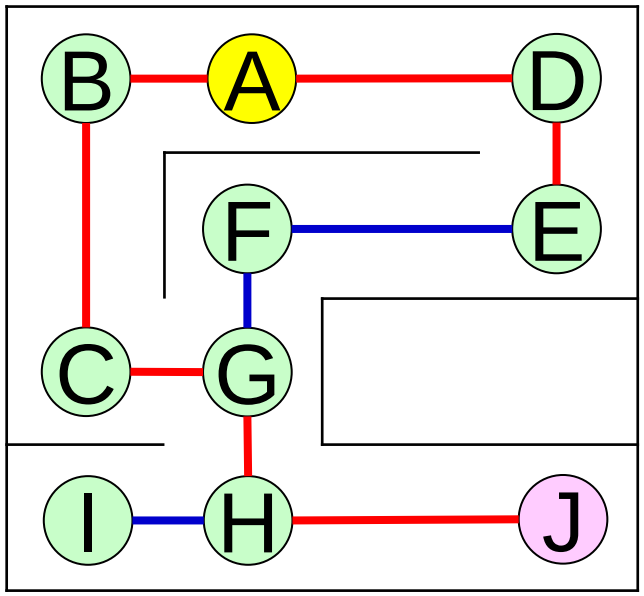
A*探索（A-star search)

- sを経由する場合の遷移コスト和の推定値 $\widehat{g}(s)+\widehat{h}(s)$ を最小にする状態を選ぶ
  - $g(s)$: 初期状態からのsまでの遷移コスト和の最小値
  - $\widehat{g}(s)$: 探索済みノードから遷移する場合の最小値（最適探索と同じ）
    - ノードsを訪れると隣接ノードの$\widehat{g}(s')$が更新される
  - $h(s)$: sから最終状態までの遷移コスト和の最小値
  - $\widehat{h}(s)$: $h(s)$のヒューリスティック推定値

A*探索

- 例: $\widehat{h}(s)$としてゴールまでのマンハッタン距離（XY各方向への距離の和）を使用

スタート



ゴール

A(0,5)
B(1,6), D(2,3)
B(1,6), E(3,2)
B(1,6), F(5,4)
C(3,4), F(5,4)
G(4,3), F(5,4)
H(5,2), F(5,4)
I(6,3), J(7,0), F(5,4)
累積遷移コスト7

A*探索（A-star search)

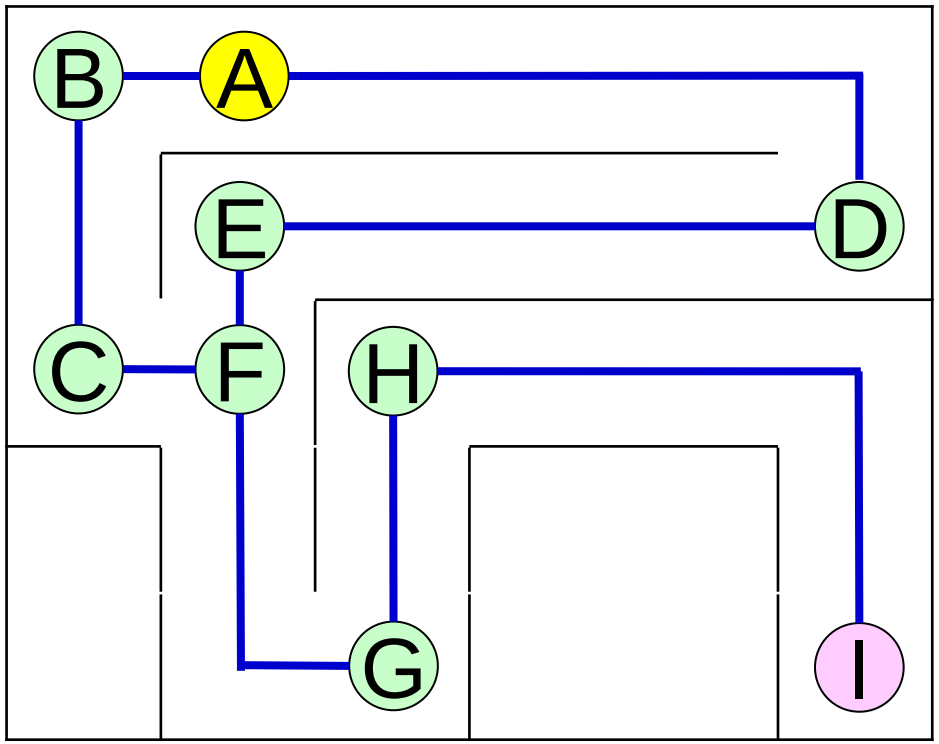
- sを経由する場合の遷移コスト和の推定値 $\widehat{g}(s)+\widehat{h}(s)$ を最小にする状態を選ぶ

- $\forall s,\ 0 \leq \widehat{h}(s) \leq h(s)$ が成り立つとき$\widehat{h}(s)$は許容的(admissible)であるといい，この場合にはA*探索は最適性をもつ
  - 最適探索（ダイクストラ法）は$\widehat{h}(s)=0$の場合に対応

演習

- 以下のグラフに対して，マンハッタン距離を用いてノードAからのA*探索を行え．

スタート



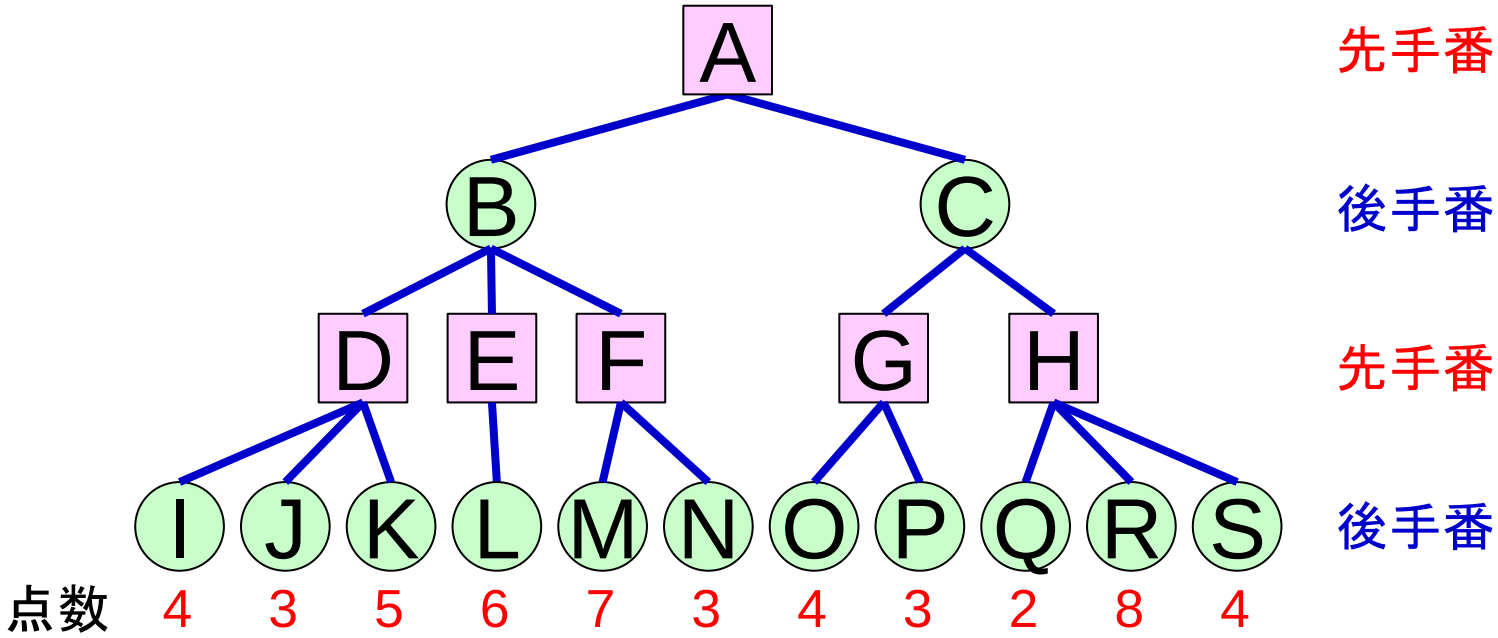
ゴール

講義の流れ

1. 状態空間
2. コスト無しグラフの探索
3. コスト付きグラフの探索
4. ゲーム木の探索

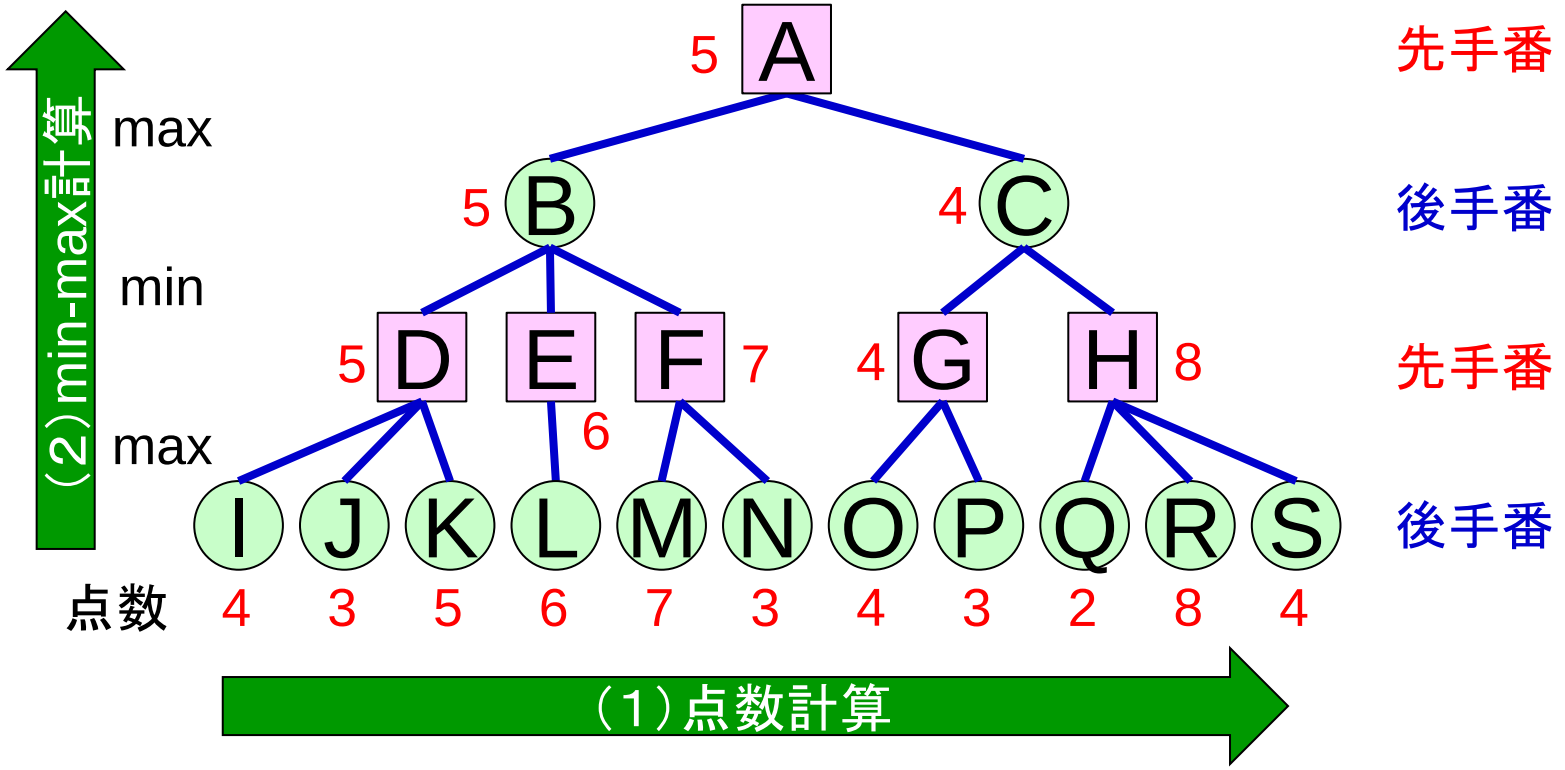
ゲーム木

- 二人のプレイヤーが交互に遷移先を決める
  - 将棋，囲碁，リバーシ（オセロ®），〇×など



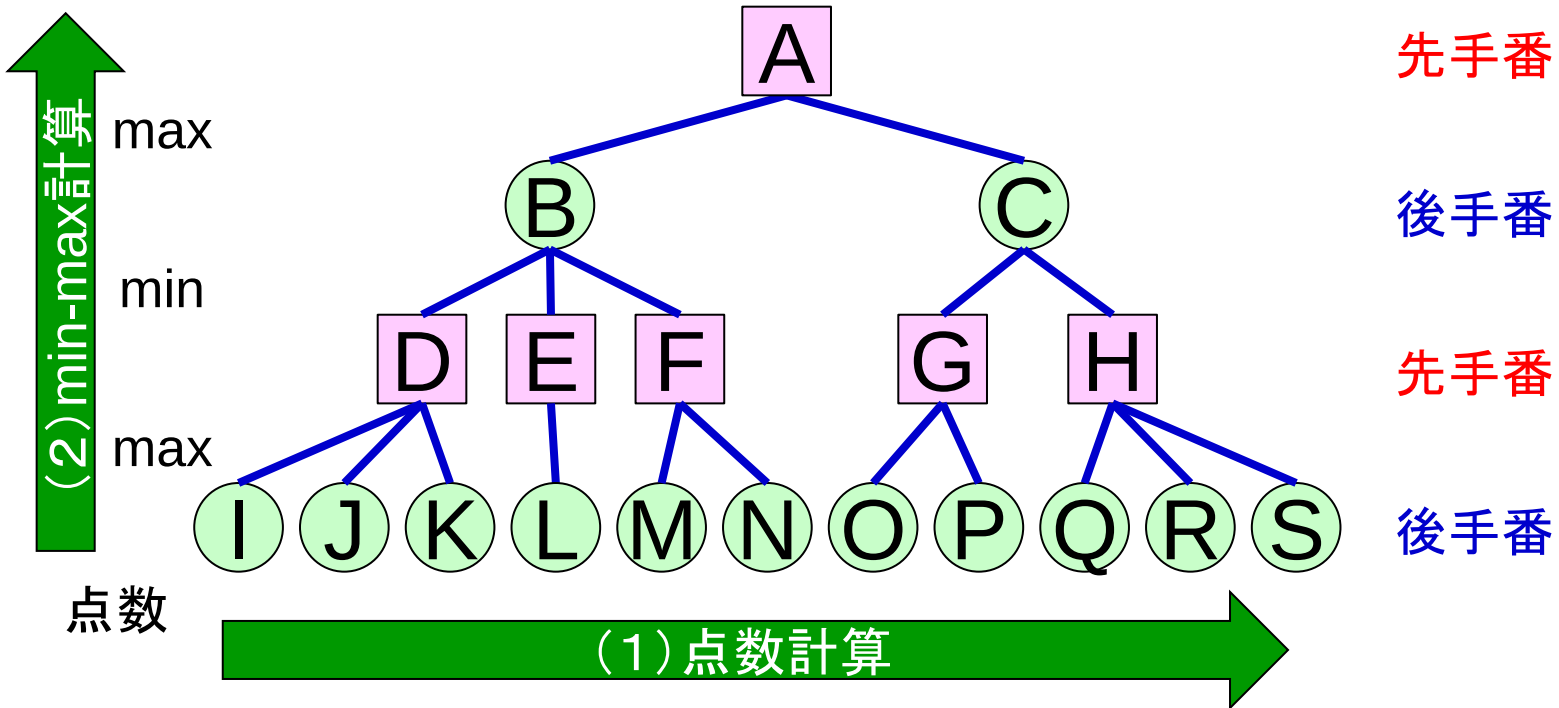
ミニ・マックス探索

- 自分は一番良い（点数を最大化する）手を選ぶ
- 相手は一番悪い（点数を最小化する）手を選ぶ



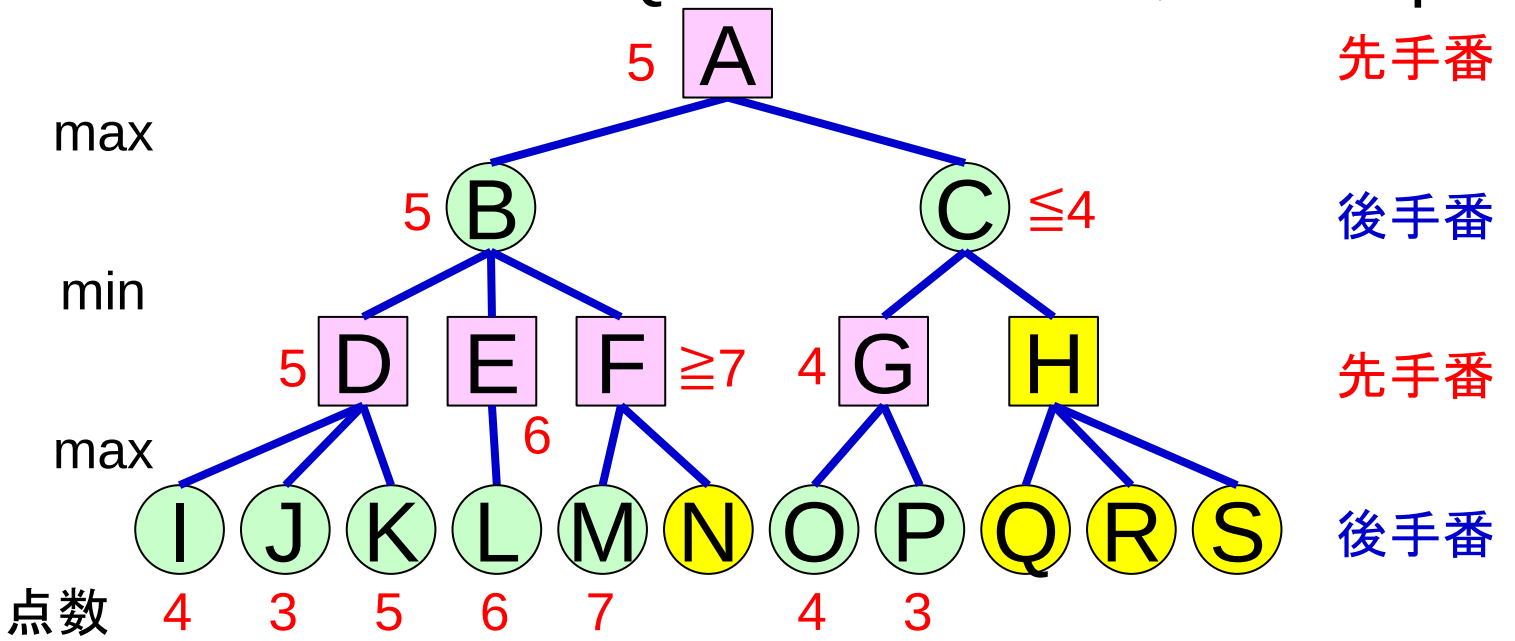
アルファ・ベータ探索

- ミニ・マックス探索では，全ての局面に対する点数を求める必要があり，時間がかかる
- 不要な点数計算を省略する
  - α：max計算の際の下限値
  - β：min計算の際の上限値



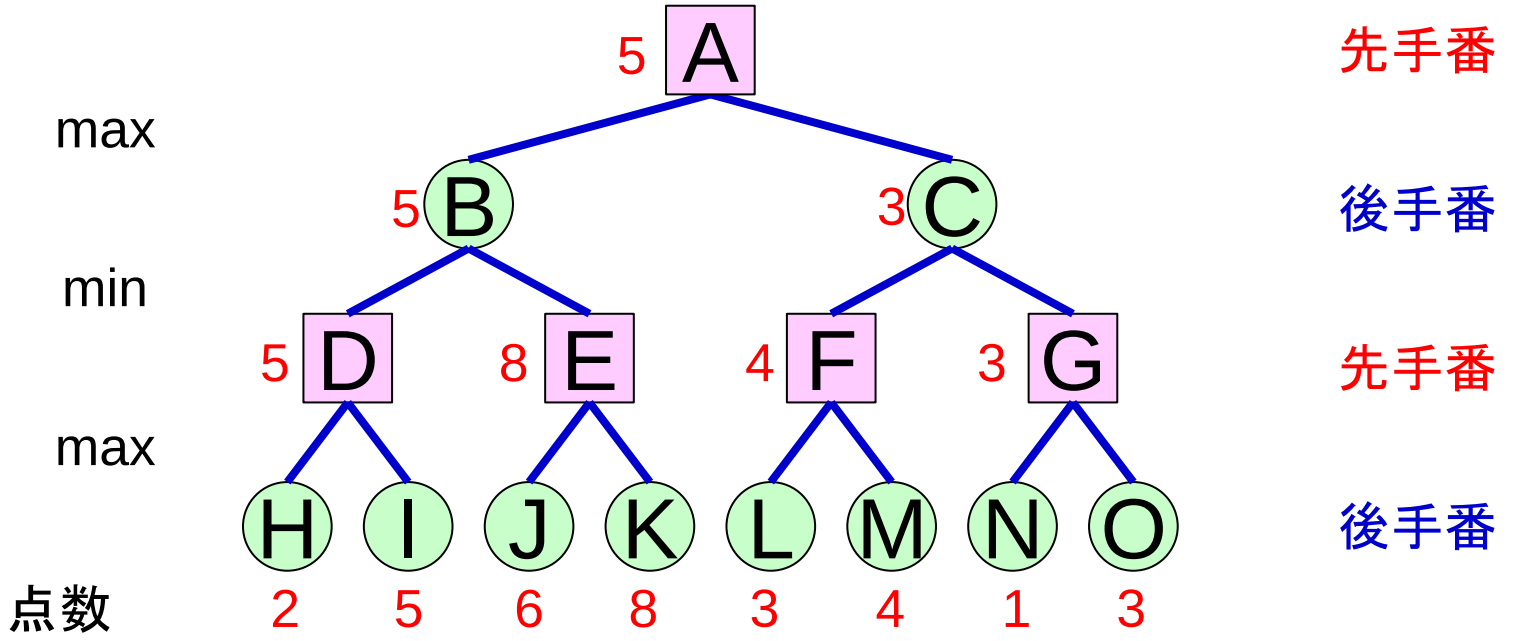
アルファ・ベータ探索

- IJKDLEMの時点で，
  - Fは7以上が確定，Bは5以下が確定
  - Fは選ばれないので，Nの点数は計算不要（αカット）
- OPGの時点で，
  - Cは4以下が確定
  - Hは選ばれないので，QRSの点数は計算不要（βカット）



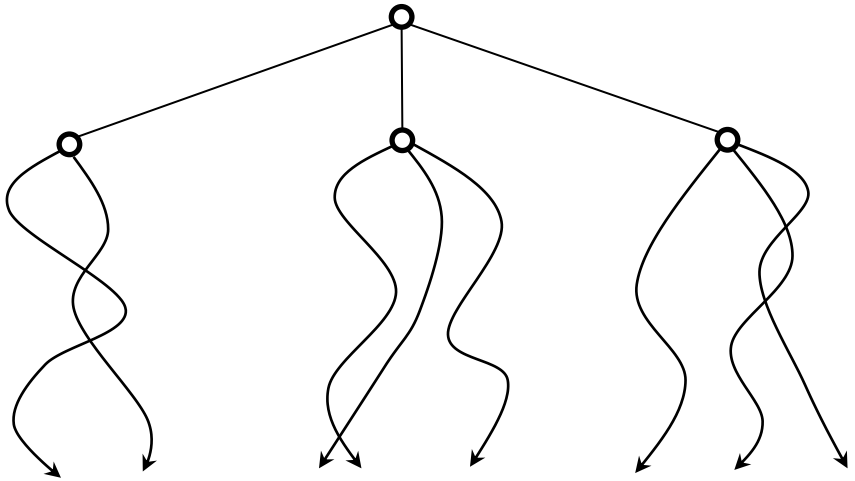
演習

- 以下のゲーム木に対してアルファ・ベータ探索を行え（点数は左側のノードから計算）



モンテカルロ木探索

- アルファ・ベータ法を用いても，ゲーム木を深く探索するのは困難（b^d が $b^{d/2}$ になる程度）
- 全探索せず，ランダムに手を打つ
- 囲碁やスケジューリングなどで活用されている

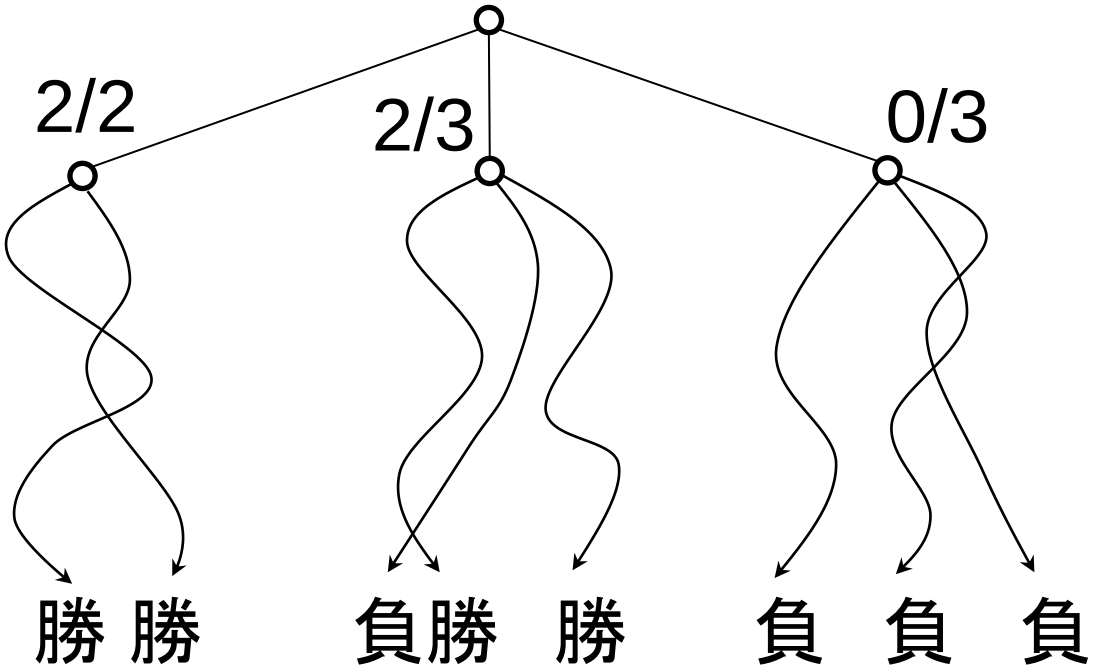


Wikipediaより

乱数を用いてシミュレーションを行うことを
モンテカルロ法(Monte Carlo method)という.
モンテカルロはモナコにあるカジノの名前

評価値の計算

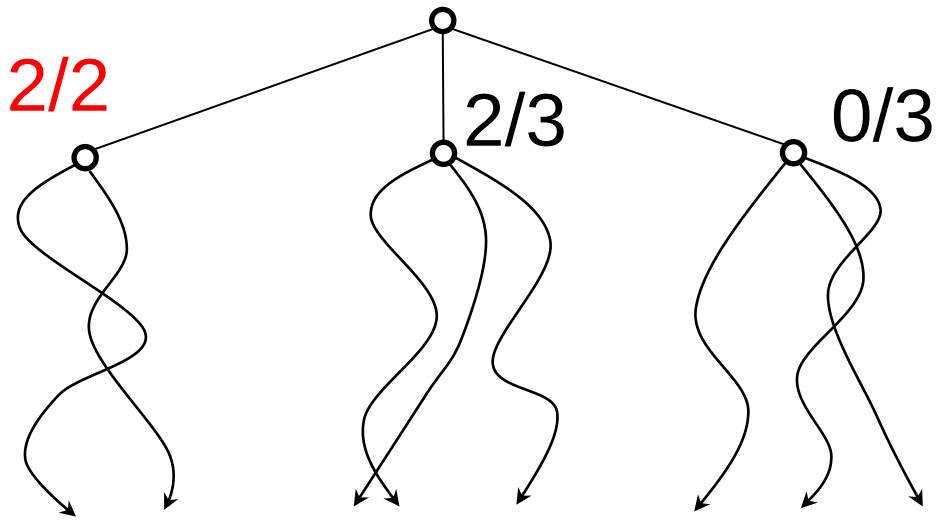
- 初手以外は（何らかのルールに従って）ランダムに打つ
  - 適当に定めた深さまで進めてから評価値を計算
  - 終局まで進めて勝敗を直接判定
    （プレイアウトまたはロールアウトという）



試行する手の選択

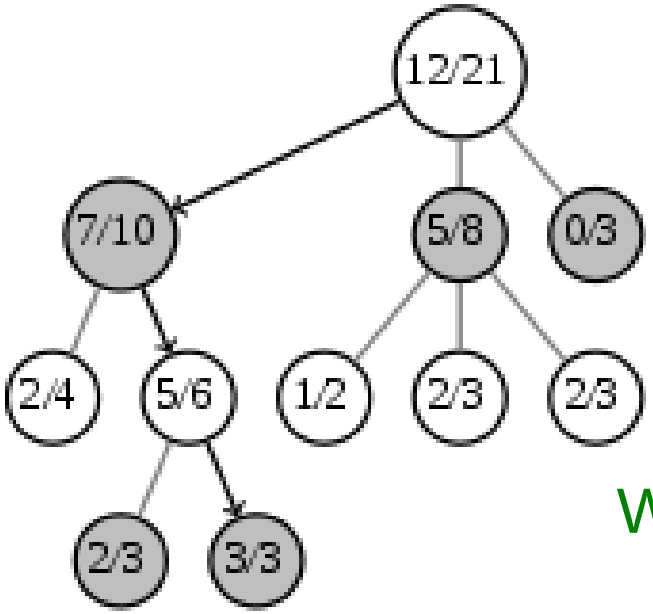
- 各手の勝率・試行回数に基づいて各試行 t で試行すべき手を選択
  - 勝率 $\hat{p}_i$ が高いか試行回数 n_i が少ない手を優遇
  - 例: UCBスコア: $\hat{p}_i + \sqrt{\frac{2 \log t}{n_i}}$

(UCB: Upper Confidence Bound, 信頼上界)



木の拡張

- 試行回数が一定数に達したノードは拡張する



Wikipediaより

  - 最終的な着手は試行数最大の手を選ぶのが標準的

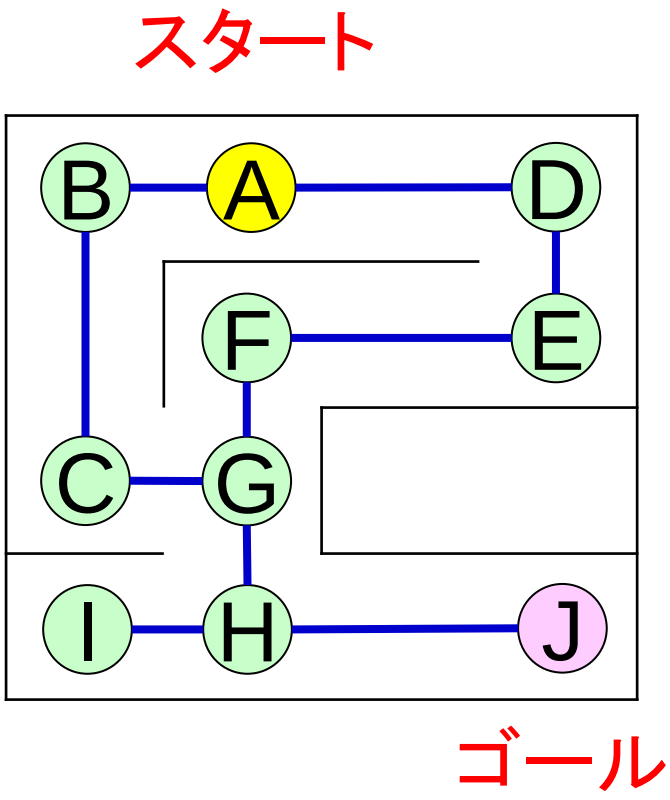
- 理論面では「無限回試行した場合には最適手順に収束」程度のことしか分かっていない

まとめ

- 広大な状態空間を効率よく探索したい
  - コスト無しグラフの探索
  - コスト付きグラフの探索
  - ゲーム木の探索
- 膨大な状態空間の探索には，近似的な手法が有用：
  - 事前知識や機械学習によるヒューリスティック関数の構築
  - モンテカルロ木探索

宿題1

- 貪欲探索，最適探索（ダイクストラ法）によって以下の迷路を探索せよ



宿題2

- 以下の5パズルの最短手順をA*探索によって求めよ
- ヒント：以下の $\widehat{h}(s)$ はいずれも許容的
  - 位置が間違っているパネルの個数（左図では5）
  - 各パネルの正しい位置へのマンハッタン距離の和（左図では2+2+1+1+2=8）