人工知能入門

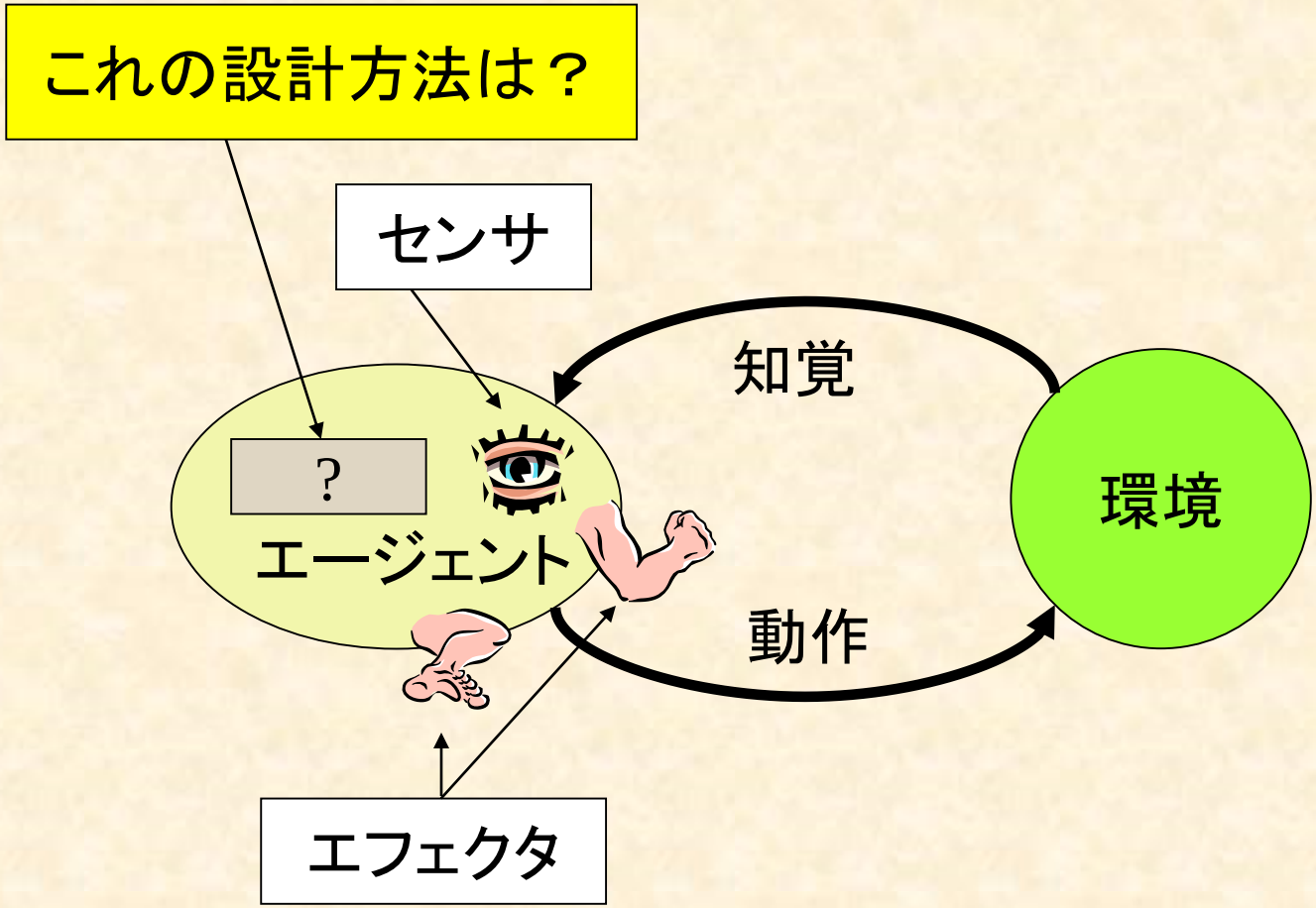
-探索による人工知能-

Lecture 6

盲目探索:幅優先探索

http://www2.teu.ac.jp/gamelab/LECTURES/ArtificialIntelligence/index.html

合理的エージェント



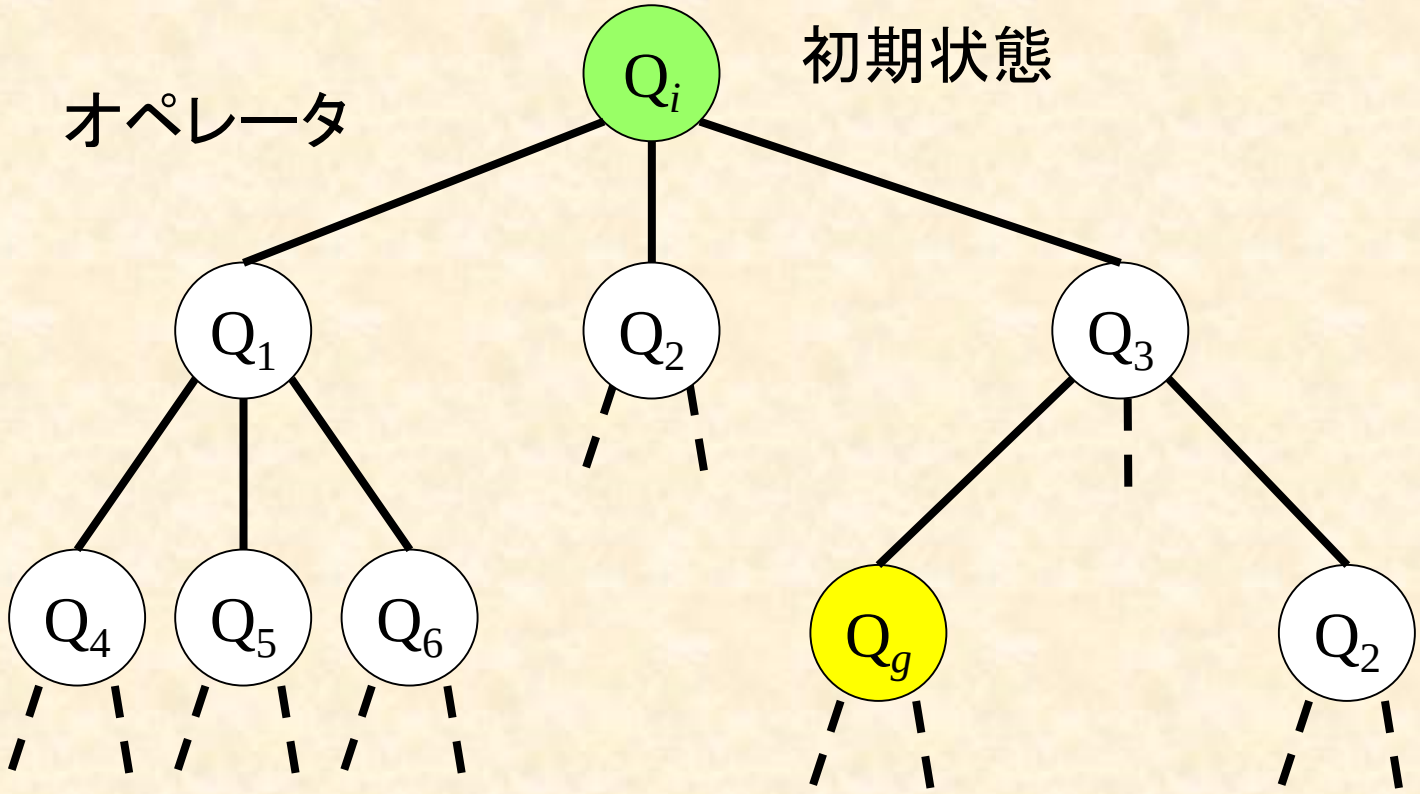
一般的探索アルゴリズム（1）

function General-Search(*problem*, *strategy*) **returns** a solution, or failure
    initialize the search using the initial state of *problem*
    loop do
        if there are no candidates for expansion **then return** failure
        choose a leaf node for expansion according to *strategy*
        if the node contains a goal state
        then return the corresponding solution
        else expand the node and add resulting nodes to the search tree
    end

探索木

- ノードは五つの構造要素
  - 状態空間においてノードが対応している状態
  - 親ノード：探索木においてこのノードを生成したノード
  - ノードを生成するために適用されたオペレータ
  - 深さ：ルートからこのノードへの経路の上のノードの数
  - 初期状態からノードまでの経路の経路コスト

探索木



探索木のためのデータ構造
待ち行列

- 縁あるいは先端
  - 展開されるのを待っているノードの集まり
  - 探索戦略は、展開される次のノードをこの集合から選ぶ
- 先端ノードの操作を実現するのは待ち行列
  - Make-Queue(*Elements*) : 与えられた要素を持つ待ち行列をつくる
  - Empty?(*Queue*) : 待ち行列に要素がない場合に限り真を返す
  - Remove-Front(*Queue*) : 待ち行列の先頭の要素を取り除いて、そしてそれを返す
  - Queuing-Fn(*Elements*, *Queue*) : 待ち行列に要素の集合を挿入する

Queuing-Fnを変化させることにより、
様々な種類の探索アルゴリズムが生まれる

一般的探索アルゴリズム（2）

```
function General-Search(problem, Queuing-Fn)
    returns a solution, or failure

    nodes ← Make-Queue(Make-Node(Initial-State[problem]))
    loop do
        if Empty?( nodes) then return failure
        node ← Remove-Front(nodes)
        if Goal-Test[problem] applied to State(node) succeeds
        then return node
         nodes ← Queuing-Fn(nodes, Expand(node, Operators[problem]))
    end
```

探索戦略

探索戦略の評価

- 探索戦略の評価
  - 完全性: 解が一つ存在するときに、それを見つけることが保証されているか？
  - 時間計算量: 解を見つけるためにどれくらい時間がかかるか？
  - 空間計算量: 探索を行うためにどのぐらいメモリを必要とするか？
  - 最適性: いくつかの異なる解があるとき、戦略は最高高品質の解を見つけるか？

探索戦略

探索戦略の種類

- 情報のない探索（盲目探索）
  - 現在の状態からのゴールに至るステップの数や経路コストに関する情報を持たない
  - ゴール状態と非ゴール状態との区別することだけ
- 情報のある探索（ヒューリスティック探索）
  - ゴールに近そうな状態を先に展開する等の情報を利用する探索戦略

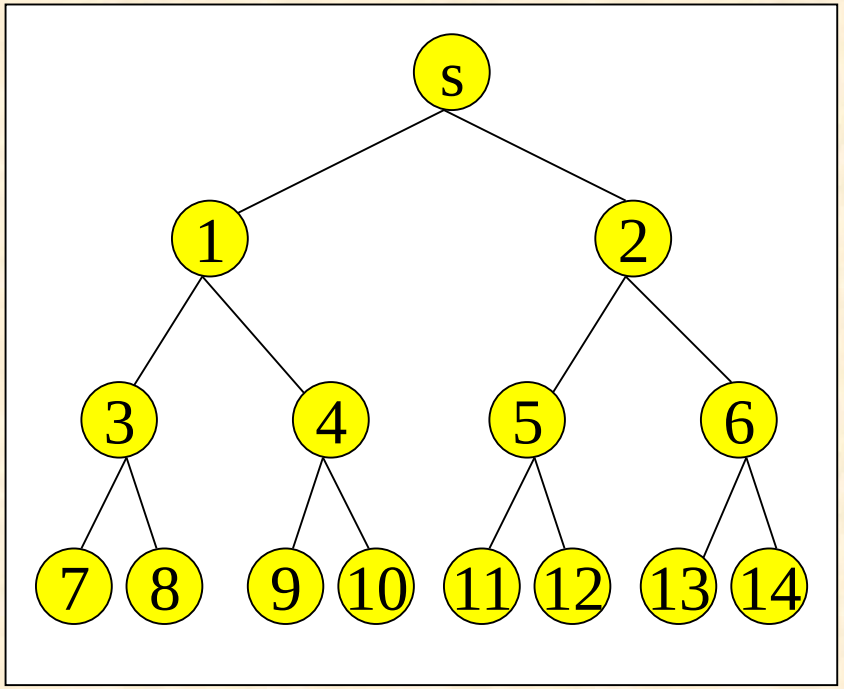
盲目探索

- 幅優先探索
- 均一コスト探索
- 深さ優先探索
- 深さ制限探索
- 反復進化探索

盲目探索

幅優先探索

- まずルートノードを展開
- 探索木における深さdのすべてのノードが、深さd+1のノードの前に展開される
- 新たに生成された状態を待ち行列の最後に置くことによって実現する



function Breadth-First-Search(*problem*) **returns** a solution or failure

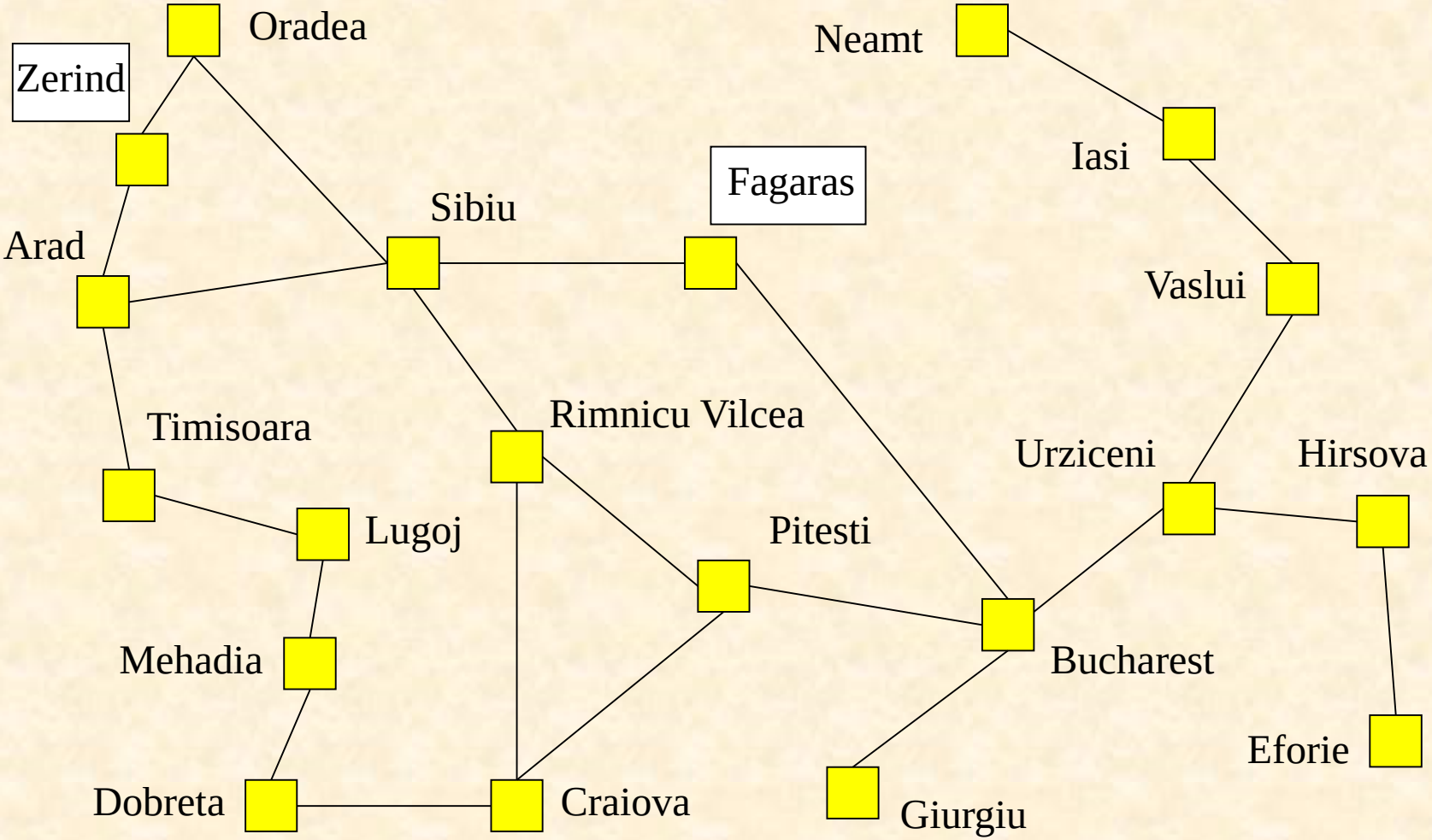
  return General-Search(*problem*, Enqueue-At-End)

幅優先探索の展開

展開回数	待ち行列	展開されたノード
0	[s]	∅
1	[1 2]	[s]
2	[2 3 4]	[1 s]
3	[3 4 5 6]	[2 1 s]
4	[4 5 6 7 8]	[3 2 1 s]
5	[5 6 7 8 9 10]	[4 3 2 1 s]
6	[6 7 8 9 10 11 12]	[5 4 3 2 1 s]
:	:	:

幅優先探索

演習問題6：FagarasからZerindへ



盲目探索

幅優先探索

- 幅優先探索のメリット
  - 完全性: もし解があれば、見つけることが保証されている
  - 最適性: 複数の解があるならば、最も浅いゴール状態（行為回数が最も少ない）を最初に見つける
- 幅優先探索の欠点：計算量
  - 深さdの解を分枝度bの探索木に見つけるために最大$1+b+b^2+b^3+\ldots+b^d$ のノードを展開しなければならない

盲目探索

幅優先探索

深さ	ノード数	時間	メモリ
0	1	1 msec	100 bytes
2	111	.1 sec	11 Kb
4	11,111	11 sec	1 Mb
6	10^6	18 min	111 Mb
8	10^8	31 hours	11 Gb
10	10^{10}	128 days	1 Tb
12	10^{12}	35 years	111 Tb
14	10^{14}	3500 years	11,111 Tb

仮定

1. 分枝度: $b = 10$;
2. 探索速度: 1000 ノード/秒;
3. メモリ: 100 バイト/ノード

- 実行時間よりメモリ必要量の方が大きな問題である
- 時間の必要量は主要な要因である

まとめ

- 一般的探索アルゴリズム
  - 待ち行列によって実施
- 盲目探索
  - 幅優先探索、均一コスト探索、深さ優先探索、深さ制限探索、反復進化探索
- 幅優先探索
  - メリット：完全性、最適性
  - 問題：計算量

ミニテストについて

- 明日の朝は1回目のミニテストを行う
  - 持ち込み可
  - 遅刻しないように
  - 内容：問題定式化（第1回～第4回）