※本資料は，課題の理解を助けるための資料であり，厳密な説明ではない部分がある．

第1回
人工知能実験 AI-1

基本的な迷路探索処理の実装
Keyword: 状態空間 State space，グラフ Graph，
幅優先探索 Breadth First Search (BFS)，
深さ優先探索 Depth First Search (DFS)，
スタック Stack，キュー Queue

1

1

AI-1 目次

2

2

1-1
迷路問題と問題の表現

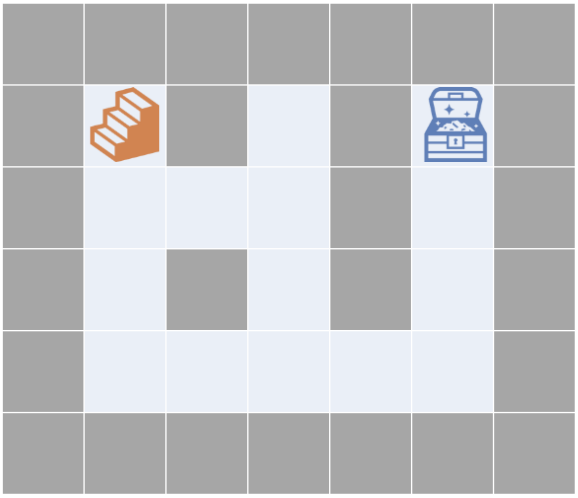
参考:
人工知能の理論 2.1節～2.1.1項 (pp. 6 – 9)

3

3

迷路問題 … 1/2

■ 階段(スタート)から出発して, 宝箱(ゴール)に辿り着きたい. ただし, 壁は乗り越えられない.



解法の例
・左手法

＊グレー部分は全て「壁」

4

4

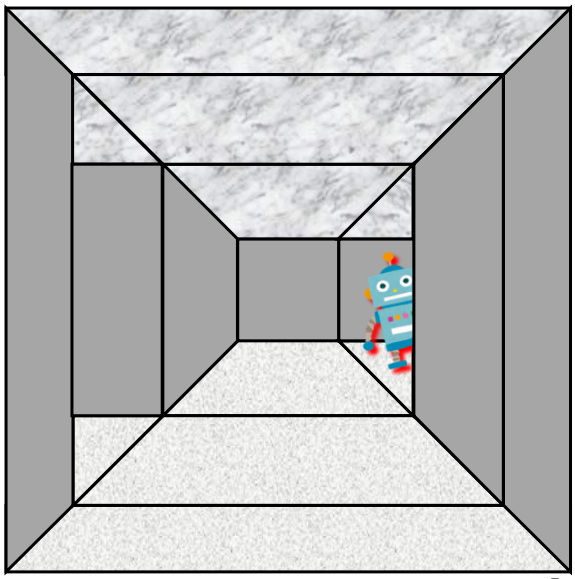
迷路問題 … 2/2

■ ただし，簡単のため，制約を与える

<u>制約1</u>
地図を所有している

<u>制約2</u>
正確な自己位置を，
常に把握可能である

<u>制約3</u>
自己の向きは，
無視できる

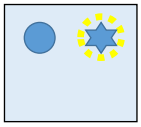


5

5

“問題を解く”ためには？

目標を定めよう！！

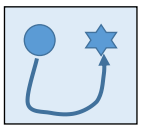


◆スタートからゴールにたどり着ける

➔ 第1回(AI-1; 本日)の目標

◆スタートからゴールへの経路を示せる

➔ 第2回(AI-2)の目標 その1



◆最短(最良)の経路を示す

➔ 第2回(AI-2)の目標 その2

6

6

問題の表現

■ 全ての「状態」を，「状態空間」上の1点とする
- 例：xy空間上の座標点（x, y）で自己位置を表現

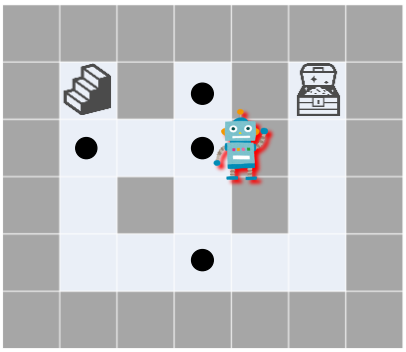
■「初期状態」から「目標状態」になるまで，「オペレータ」によって状態遷移をさせる

7

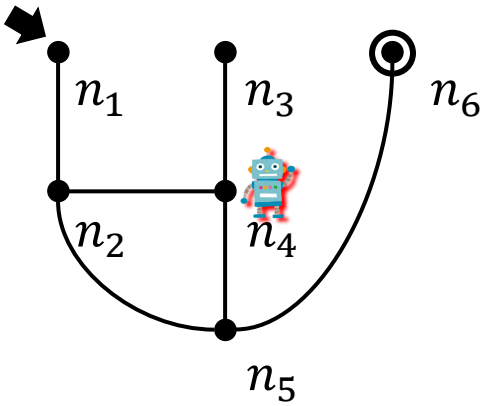
7

迷路問題から等価なグラフ探索問題への変換

■階段（スタート）から出発して，宝箱（ゴール）に辿り着きたい



■グラフ上で初期ノードから目標ノードへの経路を探す



※ グラフ理論では，節点（Node）ではなく，頂点（Vertex）と呼ぶことも多い

8

8

1-2 グラフ探索としての問題定義

参考:
人工知能の理論 2.1.2項 (pp. 9 – 11)
人工知能の理論 2.3節(pp. 12–16)

1-1
1-2
1-3
1-4

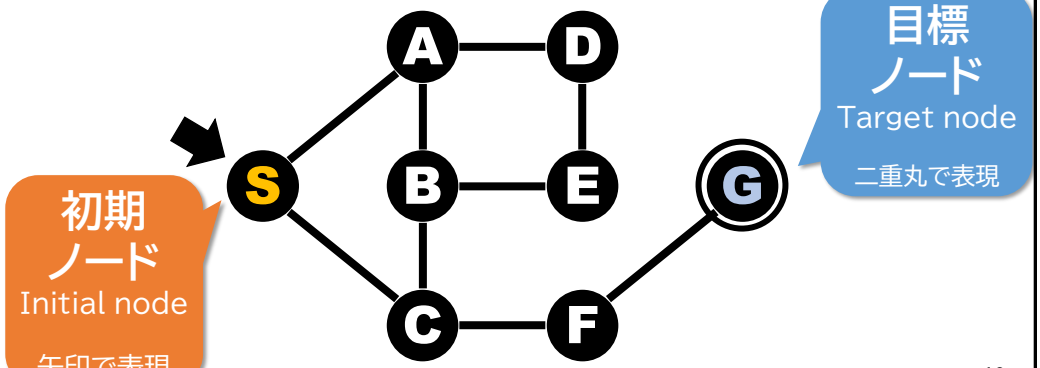
9

9

グラフ構造

注1: 一般的なグラフ理論と同様, 節点ではなく頂点(Vertex)として, 表現しても良い
注2: 開始(Start)と終了(End)として, 表現しても良い

■グラフ $G = (V, E)$
- V: 節点(ノード; node)の集合
- E: 辺(エッジ; edge)の集合



10

10

グラフ探索としての問題定義

■ *全ての「状態」を，「状態空間」上の1点とする*
- 状態空間 ➔ グラフ構造として表現
- 状態 ➔ 各ノード ※迷路問題では，現在位置(座標)に対応

■ *「初期状態」から「目標状態」になるまで，「オペレータ」によって状態遷移をさせる*
- 初期状態 ➔ 現在位置が初期ノード
- 目標状態 ➔ 現在位置が目標ノード
- オペレータ ➔ 現在位置から別の位置に移動

「移動」？

＊ここでは，ノードとして定義された位置のみ考える．

11

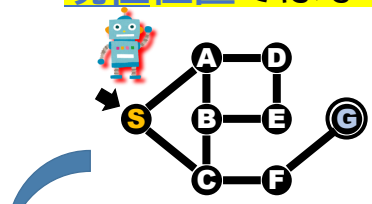
11

※注：このページでは少しだけアヤシイ説明をしています．詳細は後の参考資料を参照してください．

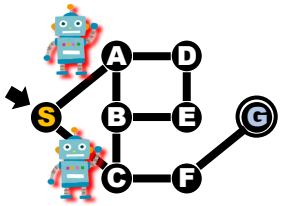
「探索」の基本的な考え方

current *hypothesis*

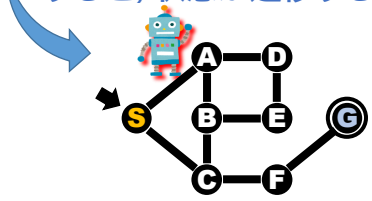
■ 現在位置ではなく，位置の仮説(候補)とし，探索



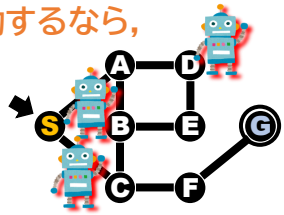
Sから，AとCに移動可能



SからAに「現在位置を移動」すると，状態が遷移する



もし，Aに移動するなら，次はBとDに移動可能（Cもまだ候補）



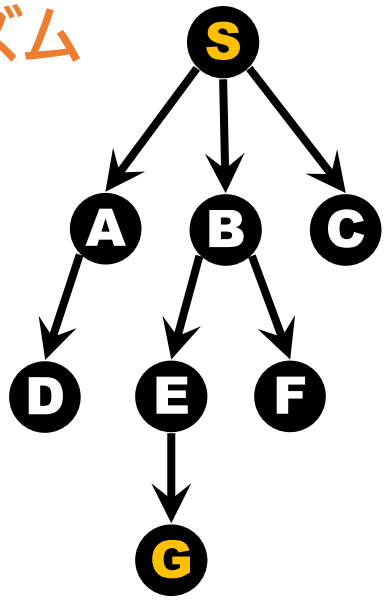
疑問：行き止まりはどうする？

もし，XXに・・・(以下繰り返し)

12

12

基本的な探索アルゴリズム



- **Depth First Search**
  深さ優先探索（DFS）※縦型探索
  - S→A→D→B→E→G
- **Breadth First Search**
  幅優先探索（BFS）※横型探索
  - S→A→B→C→D→ … →G

- 探す“順番”が重要
  ➔順番を規定するためには
  適切なデータ構造が必要

「アルゴリズムとデータ構造」の出番です！！

13

13

※これは，ページ数合わせの白紙ページです．

14

14

1-3
探索アルゴリズムの実装

1-1
1-2
1-3
1-4

参考:
人工知能の理論 2.3節(pp. 12–16)
アルゴリズム論 6.4節(pp. 111–118)

15

15

探索の基本的な方法

■ 展開待ちの探索候補をリストに保持する
- このリストをオープンリストと呼ぶ

1) 初期状態
オープンリストに初期ノードのみ保持
2) 目標状態
オープンリストに目標ノードが存在
3) オペレータ
1. オープンリストから, ノード (cur_node)を1つ取得
2. オープンリストから, cur_node を削除
3. ノード cur_node に**隣接した全ノード**を,
次の探索候補を示す仮説ノード(hyp_nodes)とする
4. 仮説ノードを, オープンリストに追加

「探索の基本的な考え方」スライドへの疑問
AからBとDへ移動？Sも「隣接」してるのでは？

16

16

参考

考察の例：現状の考え方に抜けは無いか？

疑問1

オープンリストからノードを取り出し，そのノードを基準として，“隣接するすべてのノード”をオープンリストに入れている

- 例えば，ノードAからノードBにたどり着いたとする．すると，ノードBの隣接ノードにはノードAが必ず含まれている．

➔ 2節点を行ったり来たりして，探索が進まないのでは？

疑問2

例のように，グラフには閉路（cycle; closed path）があり得る

- 例えば，ノードCはノードSの隣接ノードとして，オープンリストに追加済みである．ノードAの後，ノードBから隣接ノードを見つけると，またノードCがある．

➔ Cycleをぐるぐる回って，探索が終わらないのでは？ 17

17

クローズドリストの導入と探索処理 … 1/2

■ 検討済みの探索候補を保持するリストを用意する
- このリストを，クローズドリストと呼ぶ

■ 検討済み/展開待ちの候補は，再探索しない

1）初期状態
- オープンリストには初期ノードのみ存在
- クローズドリストは空

2）目標状態
- オープンリストは任意
- クローズドリストには，目標ノードが存在

（次ページに続く） 18

18

クローズドリストの導入と探索処理 … 2/2

（前ページからの続き）

3）オペレータ

1. オープンリストから，ノード（cur_node）を1つ取得
2. オープンリストから，cur_node を削除
3. クローズドリストに，cur_node を追加
4. cur_node に隣接した全ノードを，仮の候補ノード hyp_nodes とし，以下の処理を加える．
   1. 仮説ノードhyp_nodesから，クローズドリストに含まれるノードを削除（疑問1の解消）
   2. 仮説ノードhyp_nodesから，オープンリストに含まれるノードを削除（疑問2の解消）
5. 処理された hyp_nodes を，オープンリストに追加

19

19

データ構造の検討

■探索問題ではオープンリストの扱いが重要

- オープンリストから1つ取り出すという操作は，何らかのデータ構造から1データを取得（参照）して，そのデータを消去（削除）する，ということである．
- 「操作」の多くは「データ構造」と密接に関連している

■今回の問題に適したデータ構造の例

Stack: Last-In First-Out　　Queue: First-In First-Out

20

20

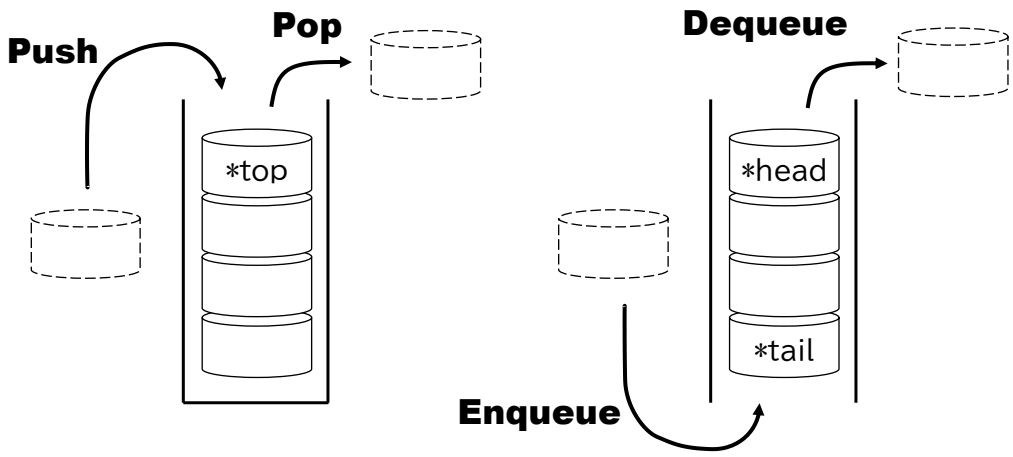
1-4
データ構造の表現
(Stack, Queue, and Graph)

1-1
1-2
1-3
1-4

参考:
アルゴリズム論 6.2節(pp. 106–109)
※重み付きグラフ/重み付き隣接行列は次回のAI-2で利用

21

スタック(Stack) と キュー(Queue)



リスト(配列)に対する, データ構造特有の操作を意識しよう!
(MATLAB実装では, *topなどのポインタは不要)

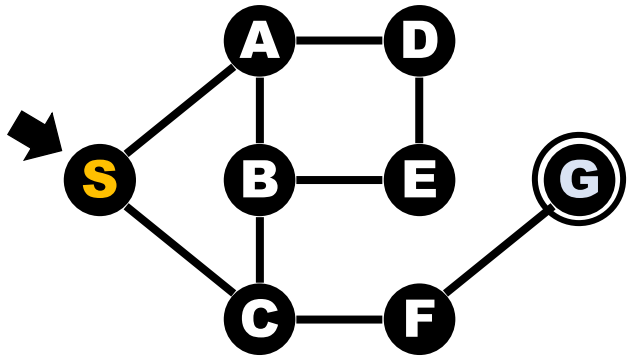
22

グラフ（Graph）

■ ノードの集合とエッジの集合で定義

■ グラフ $G = (V, E)$

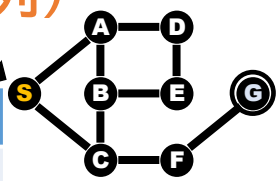
- $V = \{S, A, B, C, D, E, F, G\}$
- $E = \{SA, SC, AB, AD, BC, BE, CF, DE, FG\}$



23

23

Adjacency Matrix（隣接行列）



Source \ Destination	S	A	B	C	D	E	F	G
S	0	1	0	1	0	0	0	0
A	1	0	1	0	1	0	0	0
B	0	1	0	1	0	1	0	0
C	1	0	1	0	0	0	1	0
D	0	1	0	0	0	1	0	0
E	0	0	1	0	1	0	0	0
F	0	0	0	1	0	0	0	1
G	0	0	0	0	0	0	1	0

移動可能ノードを1とした行列

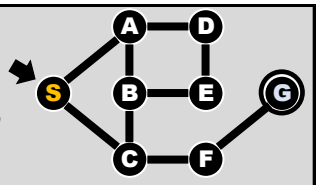
＊無向グラフの場合，対称行列となる

24

24

参考

異なる表現方法

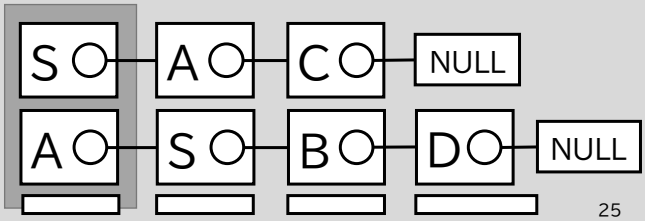


参考:隣接行列では多重辺(Multiple edges)は表現できないが,
隣接リストでは, 多重辺の表現も可能である.
(下記でいえば, Destination (next) に重複を許せばよい.)

■ Adjacency List(隣接リスト)

Source	Destination
S	{A, C}
A	{S, B, D}
B	{A, C, E}
C	{S, B, F}
D	{A, E}
E	{B, D}
F	{C, G}
G	{F}

```
/* Linked List */
struct List {
  char          node;
  struct List* next;
};
```



25

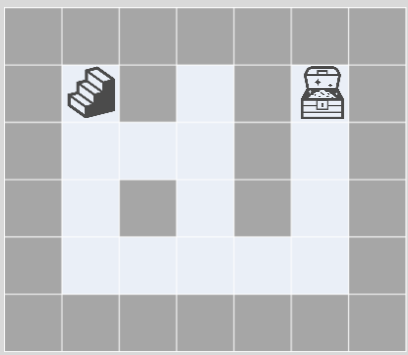
25

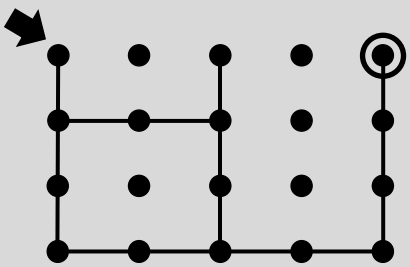
参考

グリッドベースのグラフ化

■ 説明を簡単にするため, 迷路の交点のみに注目したグラフを作成するという説明をしました

■ 実装上は, グリッドを結ぶグラフを作成した方が簡単です
- ただし, 探索における新たな課題が見つかるかもしれません
- 参考: コンパイラ(言語処理)における DFA の最小化





26

26