※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2回
人工知能実験 AI-2

効率的な迷路探索処理の実装

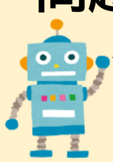
Keyword: 最良優先探索 Best First Search (BFS),
ダイクストラ法 Dijkstra’s algorithm,
A*アルゴリズム A* algorithm

27

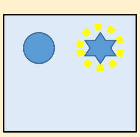
27

復習

“問題を解く”ためには？



目標を定めよう！！

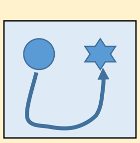


◆スタートからゴールにたどり着ける

➔ 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

28

28

AI-2 目次

2-1 経路探索

参考:
人工知能の理論 pp.12-13 あたりの記述

31

2-2 グラフの拡張と最適経路探索

参考:
人工知能の理論 図2.5 (p.11), 2.4節 (pp. 16-19)
アルゴリズム論 6.2.3項 (pp.108-109)

37

2-3 Best First Search (最良優先探索)

参考:
人工知能の理論 2.4節 (pp. 16-19)
※2.5節も関係する内容だが, 発展課題である
アルゴリズム論 6.5節 (pp. 118-122)

47

2-4 データ構造の表現2 (Priority Queue, and Weighted Adjacency Matrix)

参考:
人工知能の理論 図2.5 (p.11)
アルゴリズム論 6.2.3項 (pp.108-109)

55

29

29

※これは, ページ数合わせの白紙ページです.

30

30

2-1
経路探索

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

参考：
人工知能の理論 pp.12-13 あたりの記述

31

31

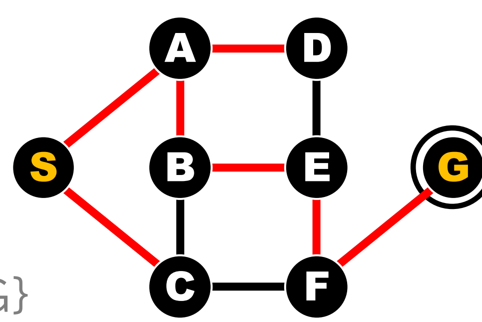
経路（Path）

■スタートからゴールに至ることができる道
- 探索終了後の Back-tracking で得られる

■Path の表現例

（1）ノードの順序集合
- {S, A, B, E, F, G}

（2）エッジの集合
- {SA, AB, BE, EF, FG}

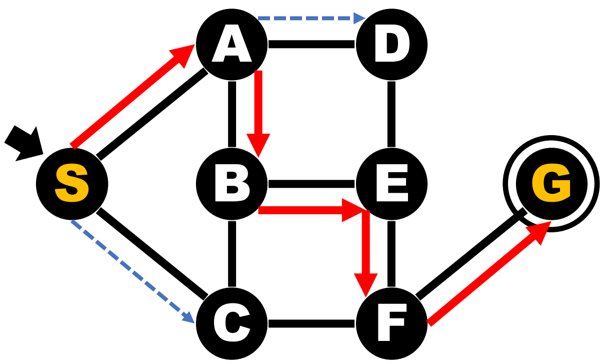


※ノード C や D は Path には含まれない

32

32

親ノードの保持



探索途中で，各ノードに辿り着いた際に，**親ノードを記録**しておく

探索過程の親ノードリスト

Node	Parent
S	NULL
A	S
B	A
C	S
D	A
E	B
F	E
G	F

33

33

経路探索の実装方針

■ 親ノードリストは，クローズドリストの機能を包含

*展開済みノードの管理

■ **実装方針：クローズド／オープンリストの機能拡張**

- 「ノード」の情報として，「ノード番号」と「親ノード番号」を，同時に保持できるようにする
- オープンリストにノードを追加する際，親ノード（展開元のノード）の情報も追加する

ノード
親ノード
コスト

34

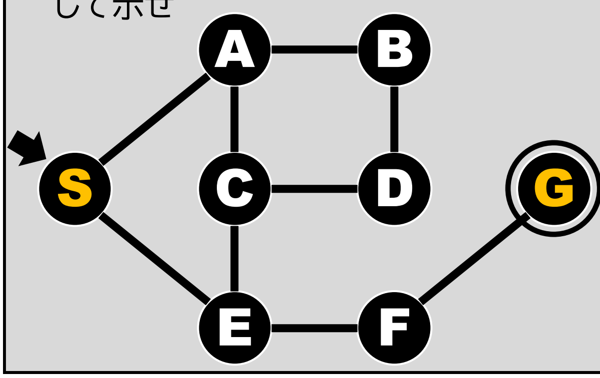
34

参考

例題2-1：探索結果のPathの抽出

解答例は
次ページ

1. 深さ優先探索(Dep.)と
幅優先探索(Bre.)を用いて探索し，
それぞれの親ノードリストを構築せよ
   - 未定義の場合はNULLとする
   - 同時なら辞書順に操作(例: CよりBを優先)
2. 探索結果のPathをノードの順序集合として示せ



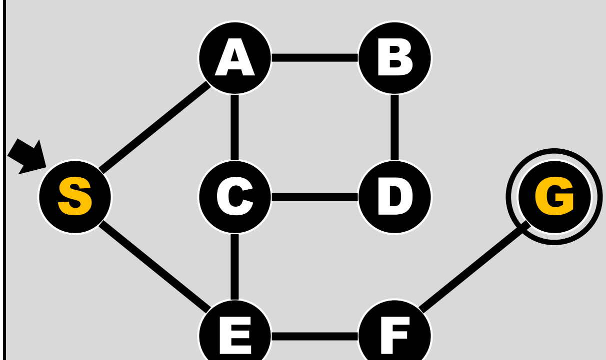
Node	Parent	
	Dep.	Bre.
S	NULL	NULL
A	S	S
B		
C		
D		
E		S
F		
G		

参考

例題2-1：探索結果のPathの抽出

解答例

- Depth First Search
S → A → B → D → C → E → F → G
- Breadth First Search
S → E → F → G



Node	Parent	
	Dep.	Bre.
S	NULL	NULL
A	S	S
B	A	A
C	D	A
D	B	B
E	C	S
F	E	E
G	F	F

2-2
グラフの拡張と最適経路探索

参考:
人工知能の理論 図2.5 (p.11), 2.4節 (pp. 16–19)
アルゴリズム論 6.2.3項 (pp.108–109)

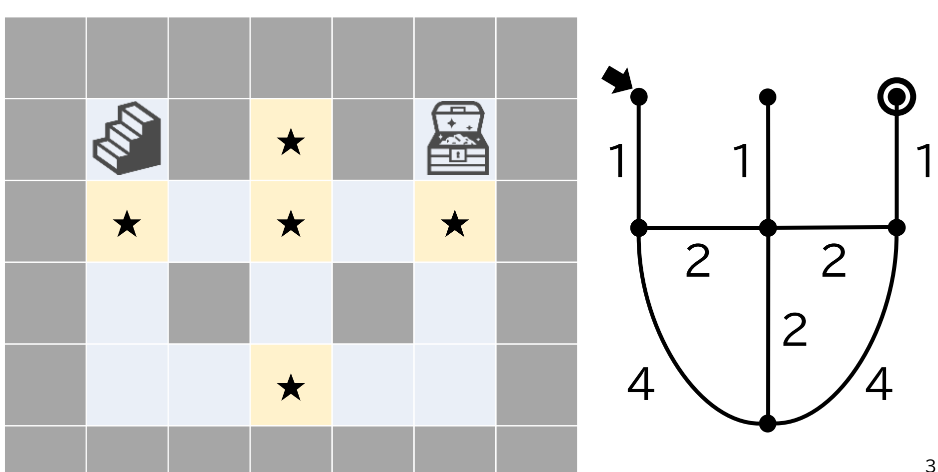
2-1
2-2
2-3
2-4

37

37

グラフの拡張 … 1/3

■ コストの概念を加えた迷路を書いてみる
- 例:交点★基準で, マス目の数がコスト



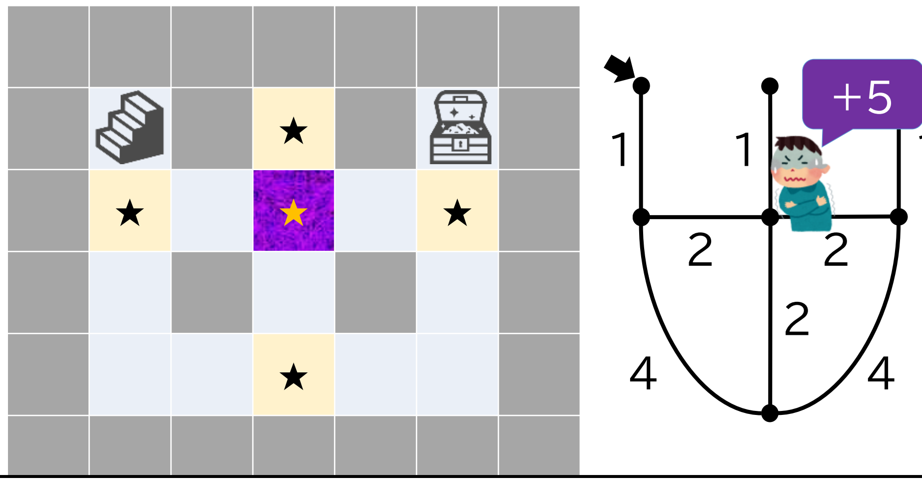
38

38

グラフの拡張 … 2/3

■ ノードにもコストがありえるのでは？

- 例：罠が仕掛けられた道

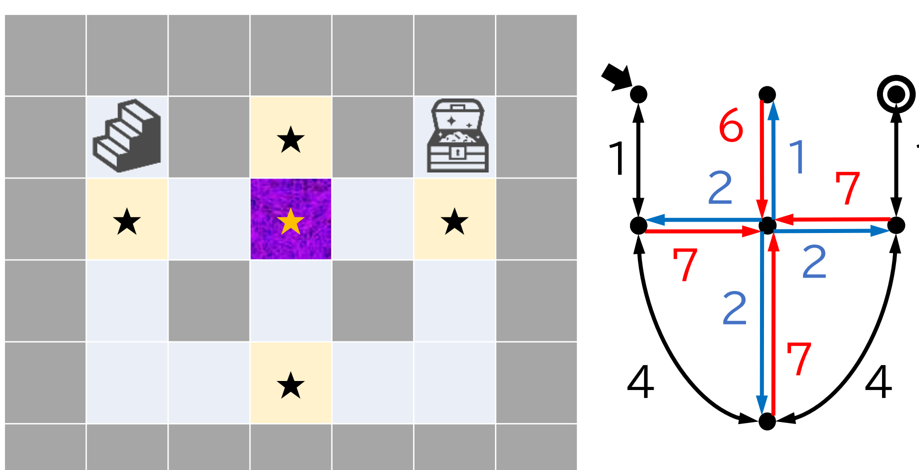


39

39

グラフの拡張 … 3/3

■ 有向グラフと考えて，**進入時にコストが大きく**なるようにしてみる



40

40

参考資料

コストの考え方について

Costs on Edges/Nodes

■State machine の考え方が参考になるかも？
- Moore machine
  - ステートに基づいた出力
- Mealy machine
  - 入力と現在ステートに基づいた出力

■両者は，変換することで等価な機械を作ることが可能
- 詳しくは「パタヘネ」を読みましょう
  - ハードウェアやシステムプログラミングの定番教科書

“Cost” and “Score”

■ コスト cost
- 大きいほどネガティブな意味の数値
- 例）

コスト最大＝悪い
コスト最小＝良い

■ スコア score
- 大きいほどポジティブな意味の数値
- 例）

スコア最大＝良い
スコア最小＝悪い

41

41

最適解の探索（知識無しから知識有りへ）

■最短経路で，効率よく，迷路（グラフ）を攻略

Q1. 経路の距離はどのように測る？
- 隣接節点は，その節点間（のエッジ）のコストを，そのまま距離とする
- 非隣接節点は，最短経路の累積コストを距離とする

➔ 累積コスト最小の経路（最短経路）を**探索**する処理によって，自然と「経路の距離」も求められるのでは？

Q2. 効率良く探索をするには？
- 全探索は避けるべき
- 幅優先探索や深さ優先探索 ⇒ オープンリストの“先頭”ノードから展開
  - “先頭”の意味は何か？ ヒント：LIFO, FIFO

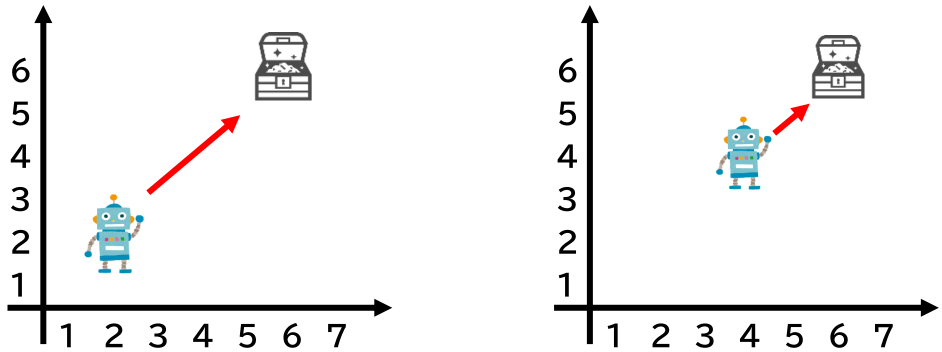
➔ オープンリスト内で**“最も良い”ノード**から展開していけば，効率良く，かつ，所望の探索を実現できるのでは？

42

42

例題2-2: ”良い”とは？ … 1/2

（どちらの候補を優先的に探索すべきか？）



方針1

ゴールまでの距離が短い候補の方が”良さそう”

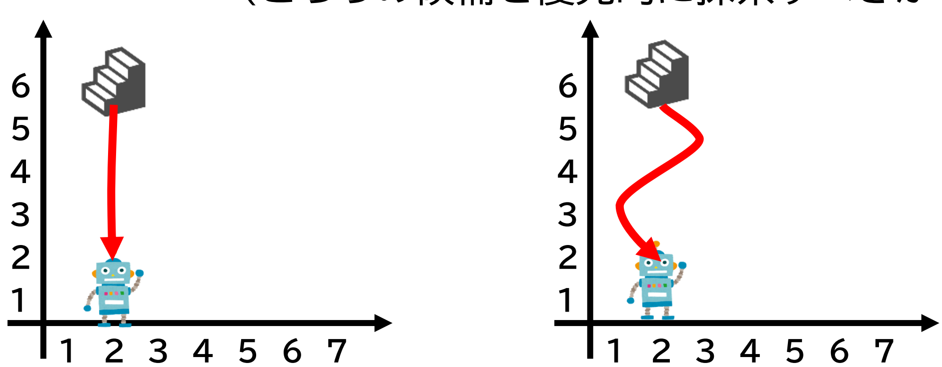
➔ 後述の方式A（貪欲な最良優先探索）の考え方

43

43

例題2-2: ”良い”とは？ … 2/2

（どちらの候補を優先的に探索すべきか？）



方針2

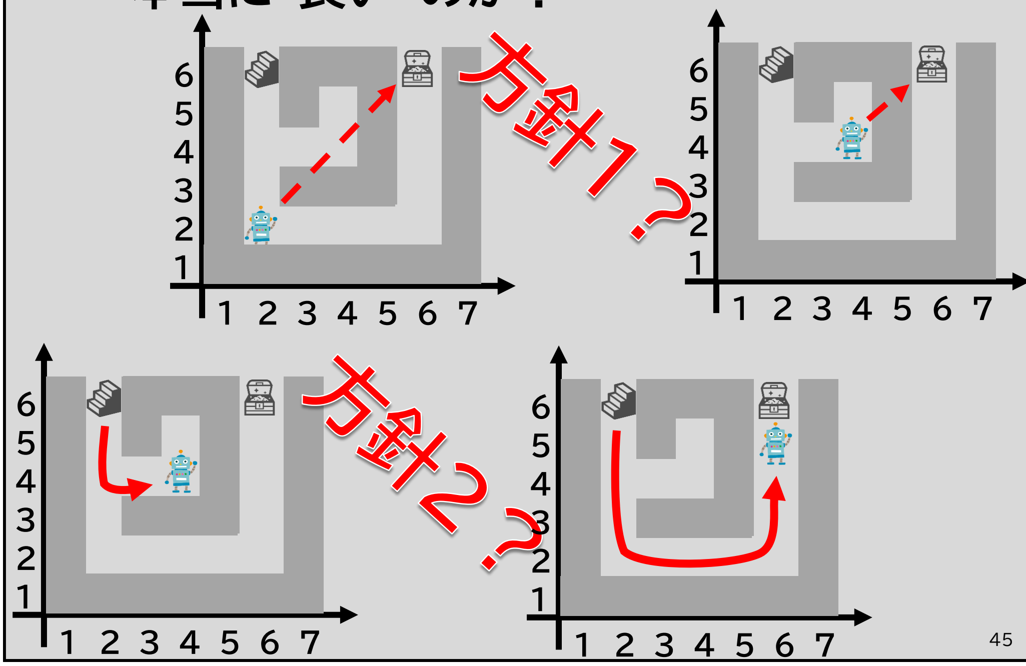
累積の移動距離が短い候補の方が”良さそう”

➔ 後述の方式B（最適探索）の考え方

44

44

参考 本当に"良い"のか？



45

45

※これは，ページ数合わせの白紙ページです．

46

46

2-3
Best First Search （最良優先探索）

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

参考：
人工知能の理論 2.4節 （pp. 16–19）
※2.5節も関係する内容だが，発展課題である
アルゴリズム論 6.5節 （pp. 118–122）

47

47

Best First Search （最良優先探索）

■ **最良のノード**から展開する探索方式

➔ オープンリスト内で，評価値 $f(n)$ が最小のノード n

方式A：$f(n)$ をゴールまでのコスト $h(n)$ とする

- 例：Greedy Best First Search （Greedy BFS）

方式B：$f(n)$ を累積コスト $g(n)$ とする

- 例：Optimal search （Dijkstra’s algorithm）

方式C：$f(n)$ を $g(n)$ と $h(n)$ の和とする

- 例：A* algorithm

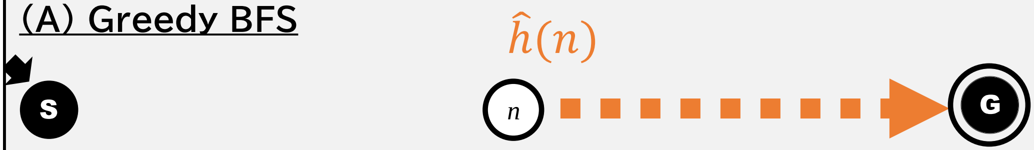
※ 重要：”コスト”はすべて正とする．

48

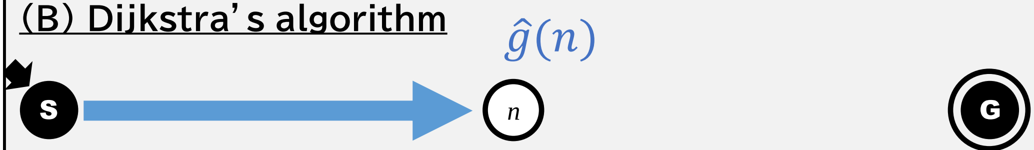
48

評価値計算の基本的な考え方

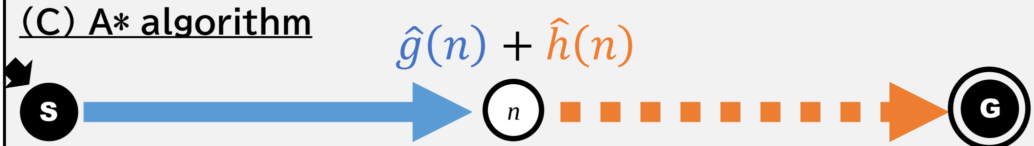
(A) Greedy BFS



(B) Dijkstra’s algorithm



(C) A* algorithm



注：推定値にはˆ（ハット）付きの記号を使う

49

49

参考

真のコストと推定値

■真のコストは未知 ➔ 推定値を利用

- 推定値にはˆ（ハット）付きの記号を使う

$$g(n) \to \hat{g}(n) \qquad h(n) \to \hat{h}(n)$$

- 探索途中では, $\hat{g}(n) = g(n)$ とは限らない
  - 特別な条件下では, 等号成立が保証される
- $h(n)$ の推定値 $\hat{h}(n)$ には<u>事前知識</u>を使う
  → heuristicな知識
  - 例：ゴールまでの直線距離

50

50

Dijkstra’s algorithm

*参考書「人工知能」
pp. 16–19
参考書「アルゴリズム論」
pp. 118–122

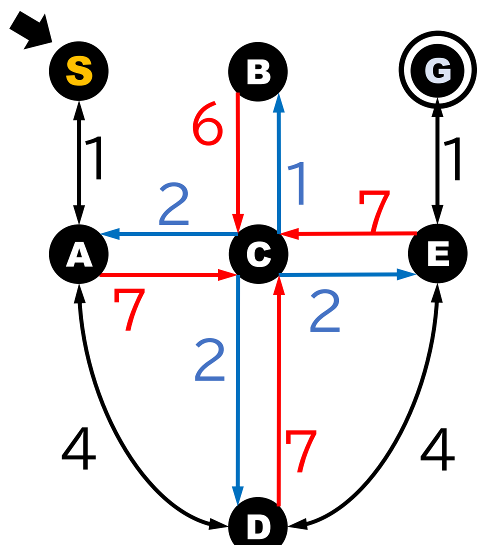
- オープンリストやクローズドリストに格納するデータ
  - 3つ組：(1)ノード，(2)親ノード，(3)累積コスト$g(n)$
- 探索の基本方針：オープンリスト内で評価関数 $f(n_i) = g(n_i)$が**最も小さいノードn_i**を展開
  - 展開元のノード n_i は，オープンリストから削除
  - 展開元のノード n_i を，クローズドリストに追加
  - 展開先のすべてのノード n_j は，
    経路コスト$c(n_i, n_j)$を加えてからオープンリストに追加
    - ただし，オープンリストにノードが存在している場合，
      既存ノードのコストと追加候補ノードのコストを比較して，
      追加候補ノードのコストの方が小さければ更新する

51

51

例題2-3：
Dijkstra’s algorithmによる探索

- 右のグラフのSからGに至る経路の最適解を，Dijkstra’s algorithmによって解け．
  - オープンリストとクローズドリストの変化がわかるように書くこと
  - Gがクローズドリストに追加された状態で終了する[*1]こと

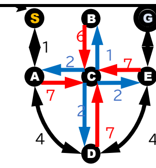


[*1] オープンリストにGが入っても，まだ探索は終了していない．

52

52

例題2-3：解答例（途中まで）



ノード
親ノード
コスト

Open List

初期化

S
0

S展開後

A
S
1

A展開後

D	C
A	A
5	8

※コストでソート済み

D展開後

C	E
A	D
8	9

C展開後

E	B
D	C
9	9

※同コストなので，実装次第で，
逆順になる場合もある

Closed List

初期化

S展開後

S
0

A展開後

S	A
	S
0	1

D展開後

S	A	D
	S	A
0	1	5

C展開後

S	A	D	C
	S	A	A
0	1	5	8

Dijkstra’s algorithm では，クローズドリスト内のノードに
スタートからそのノードに至る最小コストの値が保持される

53

53

参考

Dijkstra’s algorithm では，クローズドリスト内のノードに
スタートからそのノードに至る最小コストの値が保持される

の補足説明

定理2.1

手続き optimal-search が節点 n を展開した時点で，すでに n までの最適な道が見つけられている．

すなわち，$\hat{g}(n) = g(n)$ が成立している．

※「人工知能の理論」p.19 から引用

■ 解説

- 「手続きoptimal-search」とは，本資料で説明している Dijkstra’s algorithm のことです
- この定理の証明には，本節冒頭にさりげなく書いている，”コストに関する仮定”が，重要な意味を持ちます

54

54

2-4
データ構造の表現2
(Priority Queue, and Weighted Adjacency Matrix)

参考:
人工知能の理論 図2.5 (p.11)
アルゴリズム論 6.2.3項 (pp.108–109)

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

55

55

Priority Queue

- 最良優先探索では, オープンリストの中から最良の(最小コストの)ノードを選ぶ必要がある
- Priority Queue (優先度付きキュー)

仕様

1. キューに格納するデータにはコストが付属している
2. Dequeue時に, **コスト最小のデータ**が得られる

典型的な実装

- 格納(Enqueue)の際に, コストでソートしておく
- 同コストの場合は, First-In First-Out (FIFO)とする

56

56

重み付き隣接行列 … 1/2

■ 隣接行列 Adjacency Matrix

- ノードとノードの間を<u>移動できるか否か</u>を判定する行列
- 遷移不可を0, 遷移可能を1とする

■ **重み付き隣接行列** Weighted Adjacency matrix

- ノードからノードに<u>移動するために必要なコスト</u>を表した行列
- 数字が大きいほど大きいコスト(代償)
  - ただし, 0の場合は特別. 遷移不可とする.
  - cf. 無限大のコスト≒遷移不可

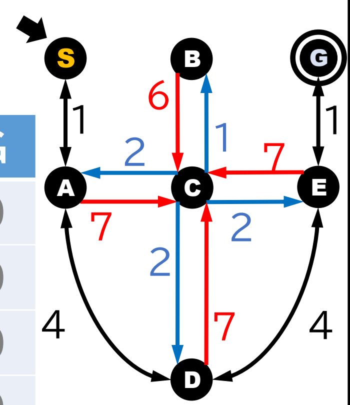
57

57

重み付き隣接行列 … 2/2

↓SRC →DST

	S	A	B	C	D	E	G
S	0	1	0	0	0	0	0
A	1	0	0	7	4	0	0
B	0	0	0	6	0	0	0
C	0	2	1	0	2	2	0
D	0	4	0	7	0	4	0
E	0	0	0	7	4	0	1
G	0	0	0	0	0	1	0



58

58