

# アルゴリズムとデータ構造III

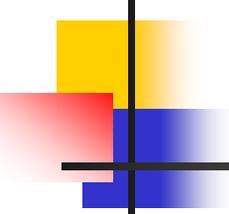
## 7回目: 11月26日

---

### グラフ

(動的計画法, ダイクストラ法, DPマッチング,  
A\*アルゴリズム)

授業資料 <http://ir.cs.yamanashi.ac.jp/~ysuzuki/algorithm3/index.html>

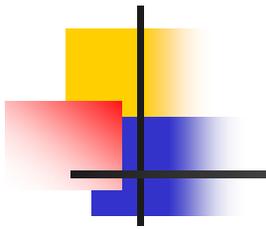


# 12月03日 午後 工場見学

---

- 工場見学(ハードウェア実験IIの授業)
- 見学場所:東京エレクトロンAT(葦崎)
- 希望者10名くらいは参加可能
- 当日の予定
  - 12:30 守衛所前集合, 大学発
  - 13:20 東京エレクトロンAT(葦崎)着
  - 13:30-15:30 見学
  - 15:30 東京エレクトロンAT発
  - 16:20 大学着
- 見学希望者はこの授業の後, 教卓の周りに集まってください

# 授業評価アンケート(中間期評価)



---

- CNSの授業のコミュニティに以下の項目について記入してください(匿名での記入が可能).
1. この授業の良いところはどこですか？
  2. この授業の改善してほしいところはどこですか？

# 授業の予定(中間試験まで)

1	10/01	スタック(後置記法で書かれた式の計算)
2	10/15	文脈自由文法, 構文解析, CYK法
3	10/22	構文解析 CYK法
4	10/29	構文解析 CYK法
5	11/12	構文解析 CYK法, 動的計画法
6	11/19	構文解析(チャート法), グラフ(ダイクストラ法)
7	11/26	グラフ(ダイクストラ法, DPマッチング, A*アルゴリズム)
8	12/03	グラフ(A*アルゴリズム), 前半のまとめ
9	12/04 4時限	<b>教室:A1-41</b> 全文検索アルゴリズム (simple search, KMP)

# 授業の予定(中間試験以降)

10	12/10	中間試験(8回目までの範囲)
11	12/11 4時限	教室:A1-41 全文検索アルゴリズム(BM, Aho-Corasick)
12	12/17	全文検索アルゴリズム(Aho-Corasick), データ圧縮
13	01/07	暗号(黄金虫, 踊る人形) 符号化(モールス信号, Zipfの法則, ハフマン符号)テキスト圧縮
14	01/14	テキスト圧縮(zip), 音声圧縮(ADPCM, MP3, CELP), 画像圧縮(JPEG)
15	01/21	期末試験

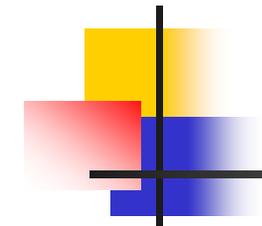


# 本日のメニュー

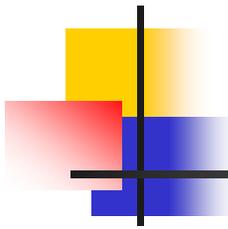
---

- 動的計画法
- ダイクストラ法(復習)
  - 動作例
  - アルゴリズム
- DPマッチング
  - 動作例
  - アルゴリズム
- A\*アルゴリズム

# 動的計画法(Dynamic Programming)



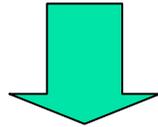
- 部分問題の解をより大きな問題を解くために利用
- 同じ問題を2度解かなくても済むように解を格納
  
- アルゴリズムの例
  - CYK法(構文解析)
  - ダイクストラ法(最短経路問題)
  - DPマッチング(パターンマッチング DNAの解析にも利用)
  - DPを使った解法(ナップサック問題)
  - ビタビアルゴリズム(音声認識など)



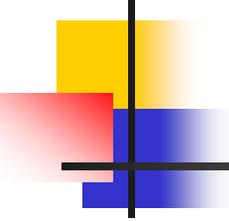
# ダイクストラ法

---

- 動的計画法を最短経路問題に適用



- 最適経路中の部分経路もまた最適経路になっている



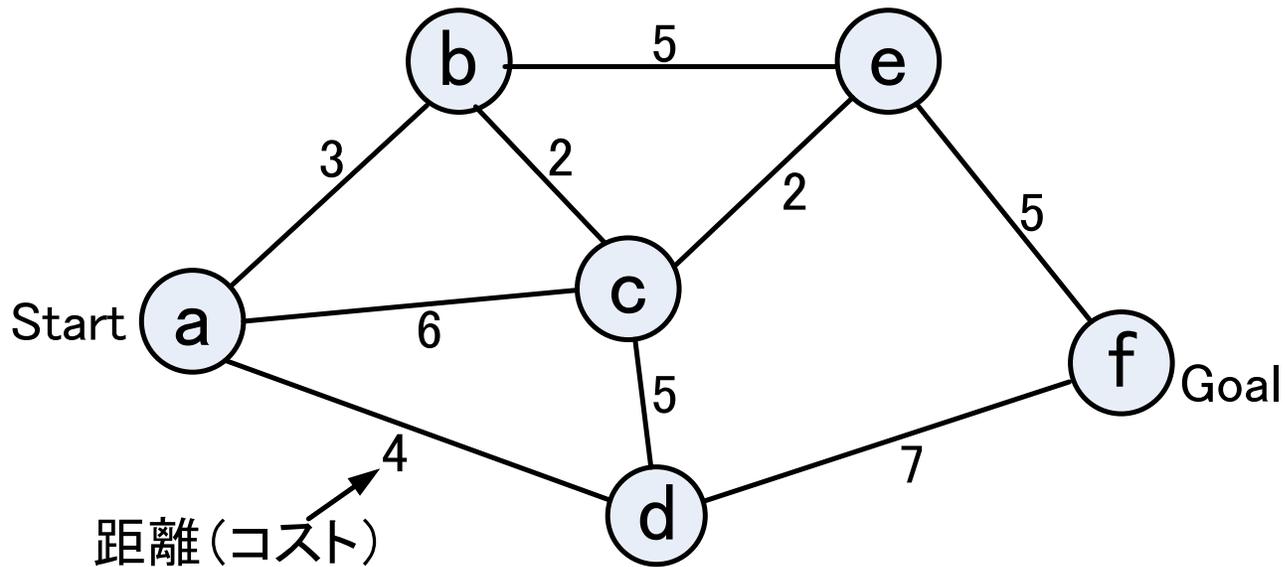
# 身近な最短経路問題

---

- 道路の経路探索(カーナビなど)

# ダイクストラ法 (最短経路問題用 アルゴリズム)

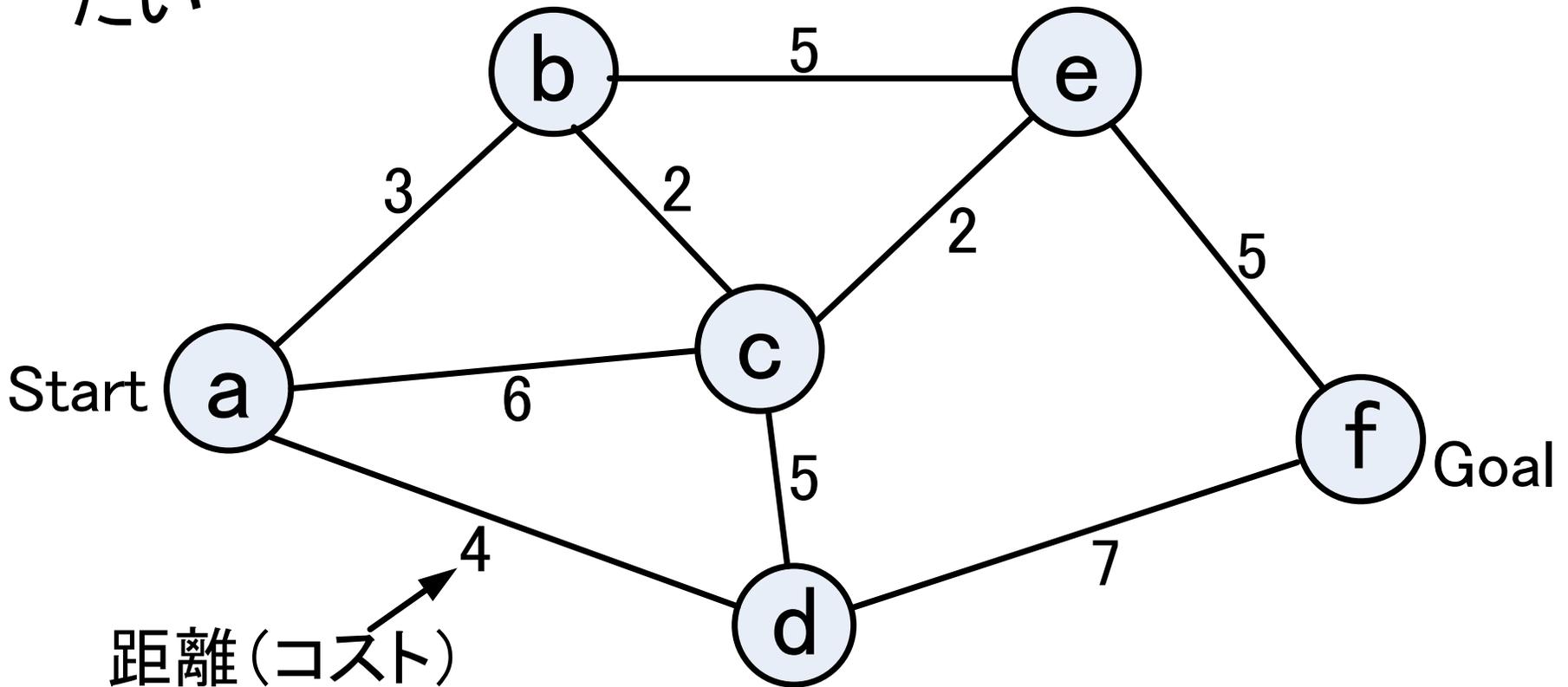
- StartノードからGoalノードへ最小コストで移動したい



- a-e, a-dなどの最短距離をa-fの最短距離を見つけるために利用
- 部分問題の解をより大きな問題を解くために利用
- 同じ問題を2度解かなくても済むように解を格納

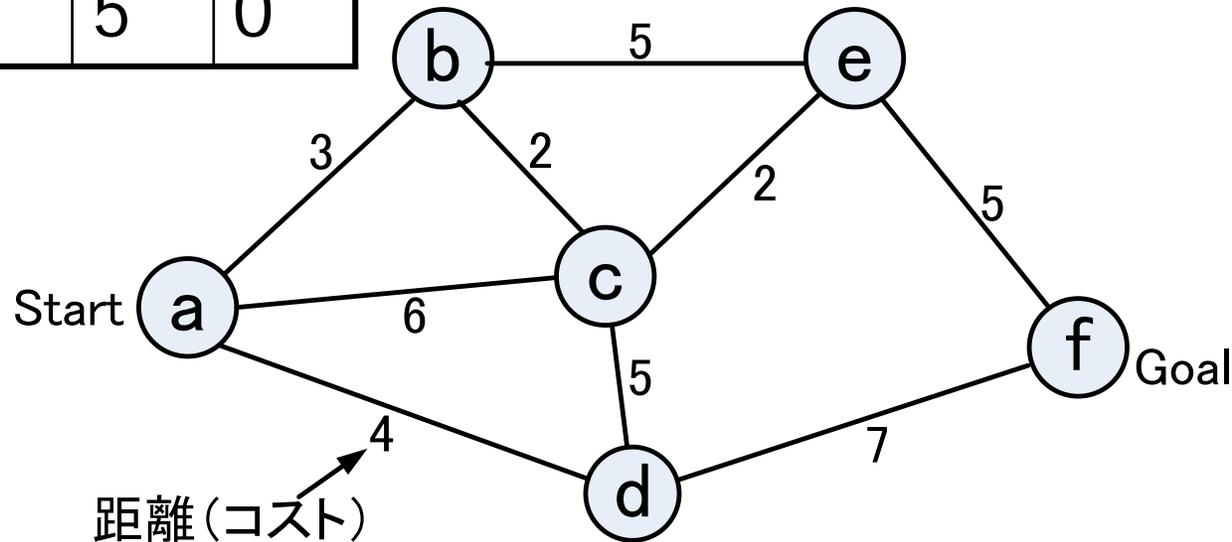
# ダイクストラ法 (最短経路問題用 アルゴリズム)

- StartノードからGoalノードへ最小コストで移動したい

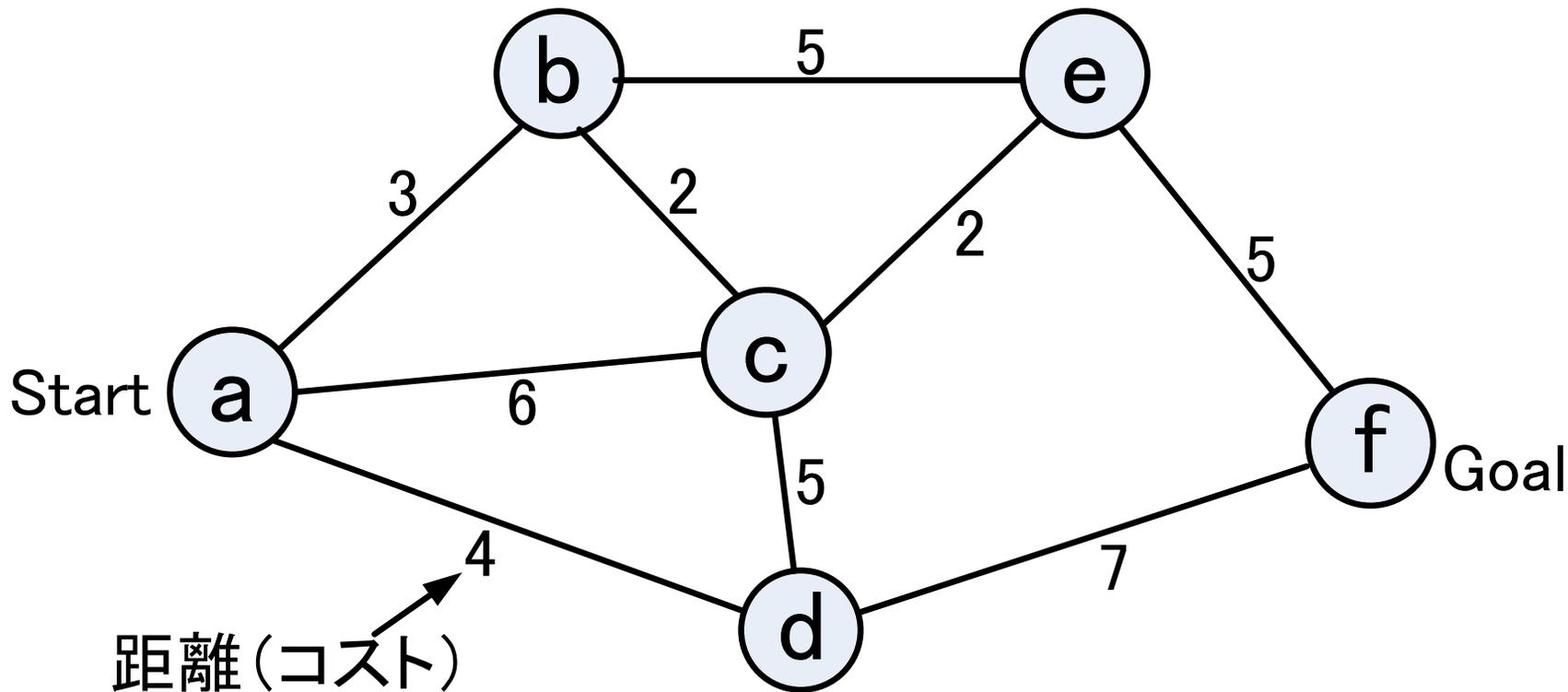


# 隣接行列 (コスト付き)

	a	b	c	d	e	f
a	0	3	6	4	-	-
b	3	0	2	-	5	-
c	6	2	0	5	2	-
d	4	-	5	0	-	7
e	-	5	2	-	0	5
f	-	-	-	7	5	0



# ダイクストラ法 動作例 1/13



Startからの最短経路が確定していないノード



Startからの最短経路を確定中のノード



Startからの最短経路が確定したノード

7

Startからの最短距離候補(未確定)

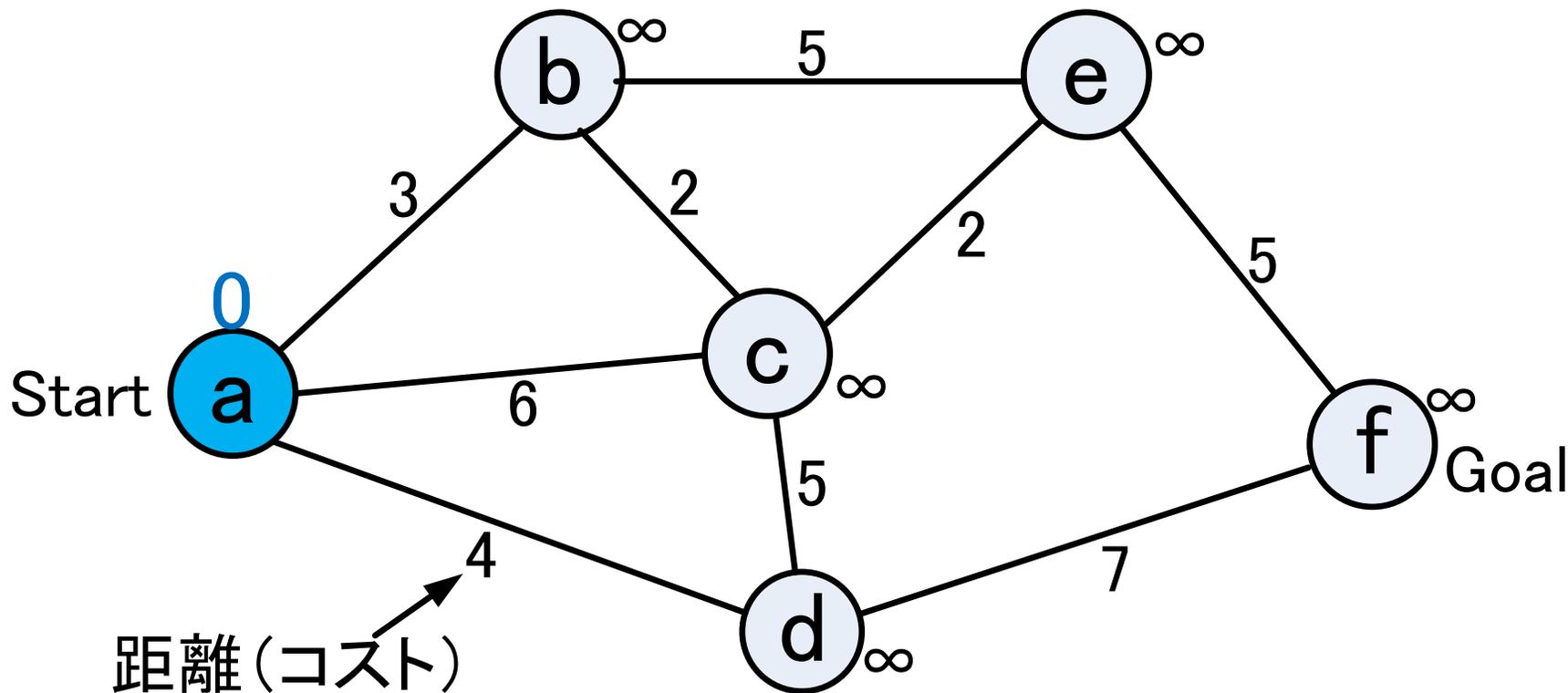
7

Startからの最短距離(確定済)



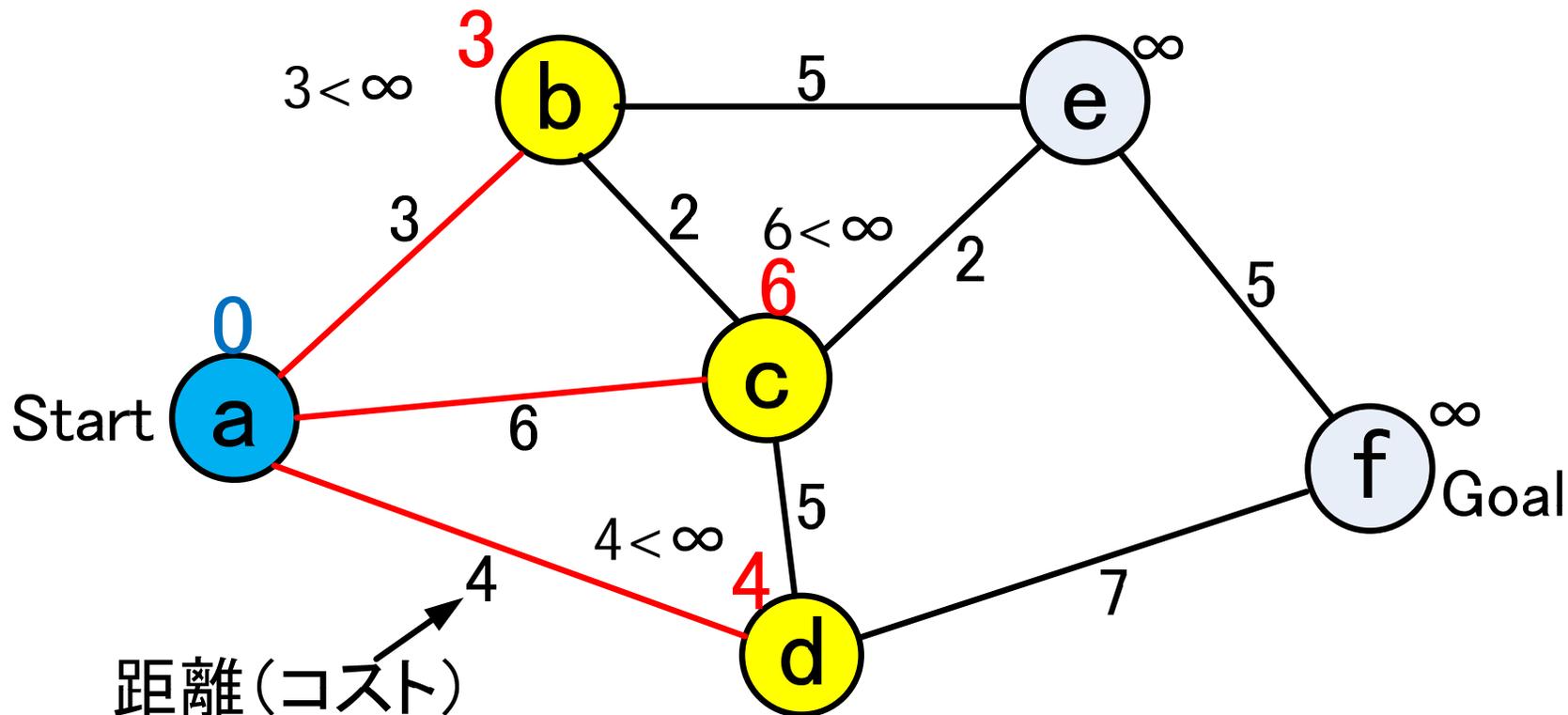
確定済ノードからのアーク  
次期確定ノード決定に使用

# ダイクストラ法 動作例 2/13



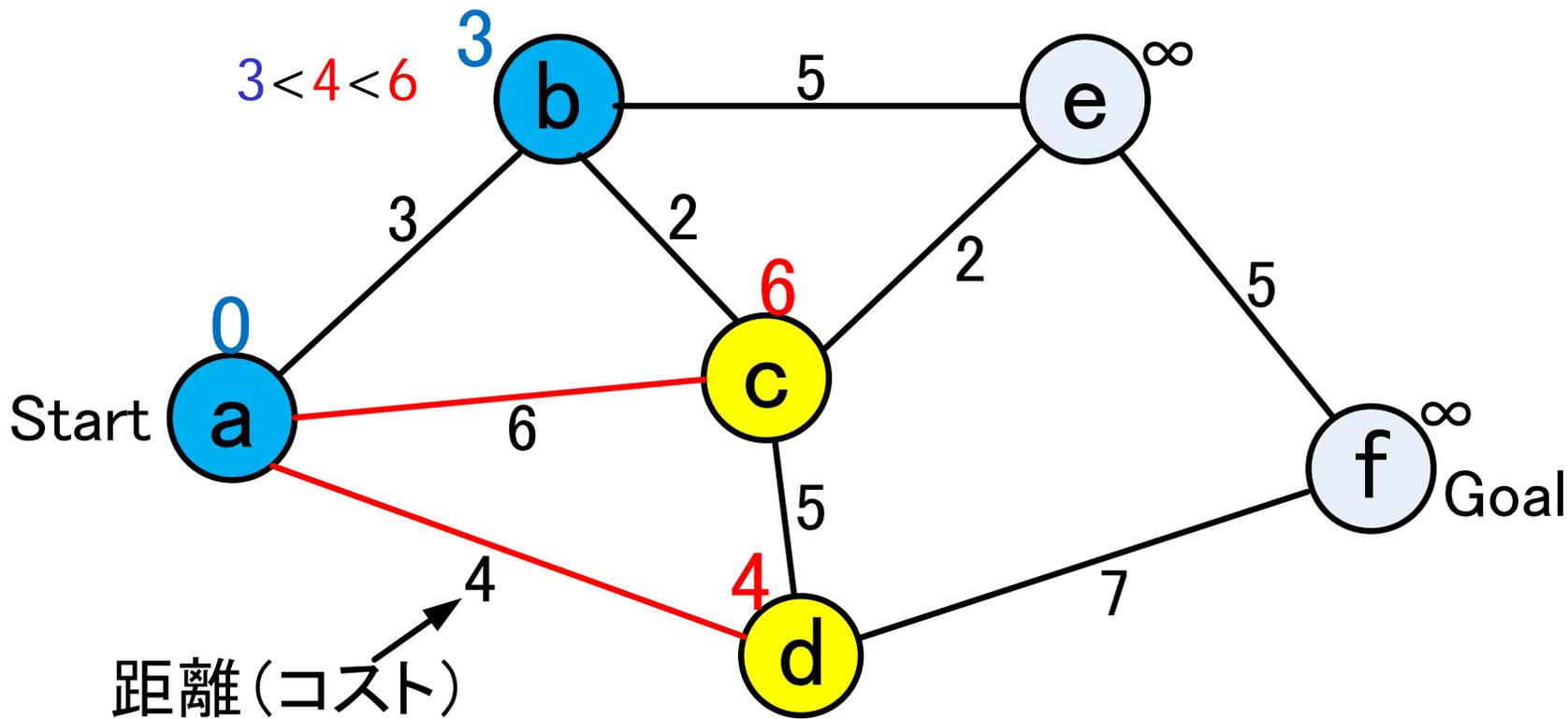
- Startからの最短経路が確定していないノード
- Startからの最短経路を確定中のノード
- Startからの最短経路が確定したノード
- 7 Startからの最短距離候補(未確定)
- 7 Startからの最短距離(確定済)
- 確定済ノードからのアーク
- 次期確定ノード決定に使用

# ダイクストラ法 動作例 3/13



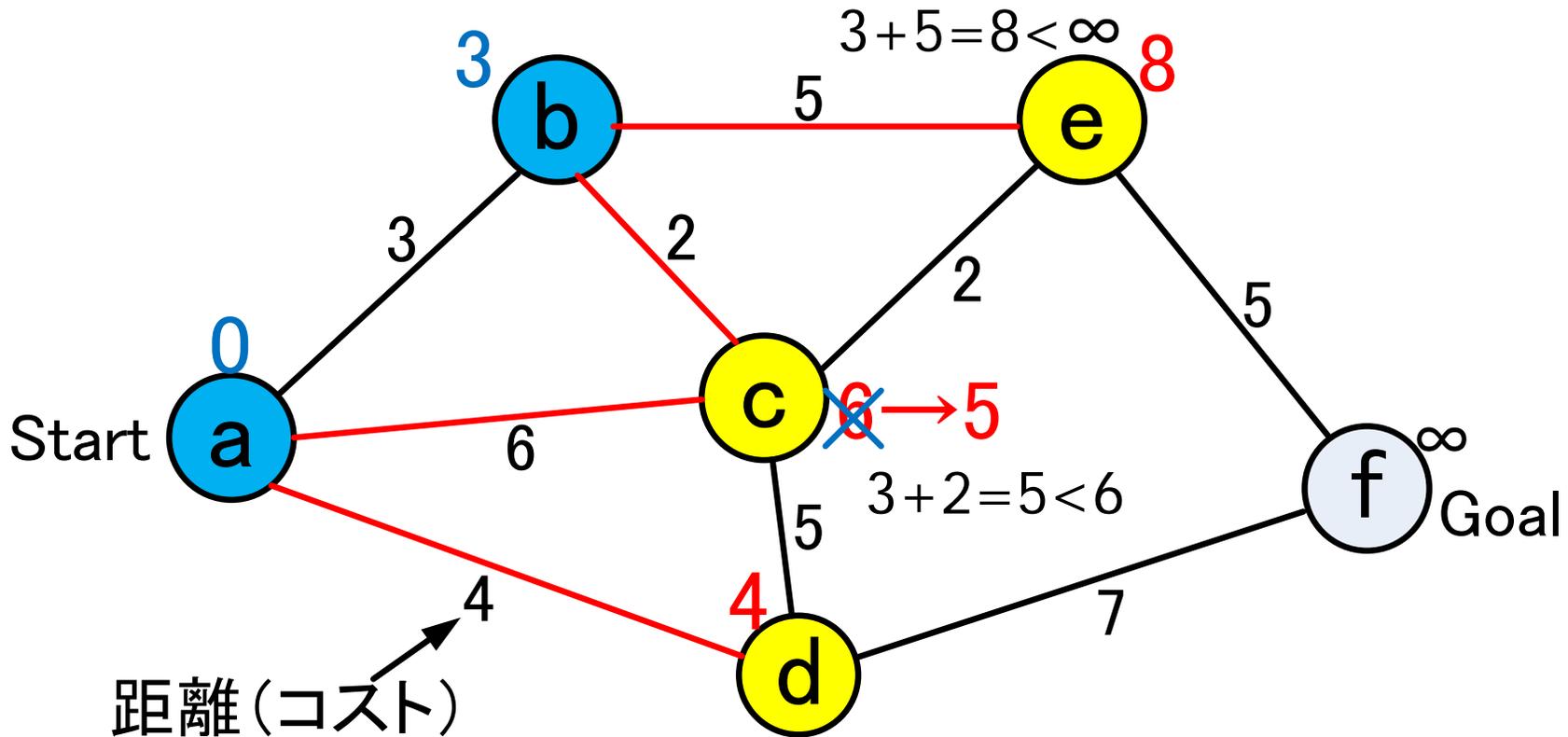
- Startからの最短経路が確定していないノード
- Startからの最短経路を確定中のノード
- Startからの最短経路が確定したノード
- 7 Startからの最短距離候補(未確定)
- 7 Startからの最短距離(確定済)
- 確定済ノードからのアーク
- 次期確定ノード決定に使用

# ダイクストラ法 動作例 4/13



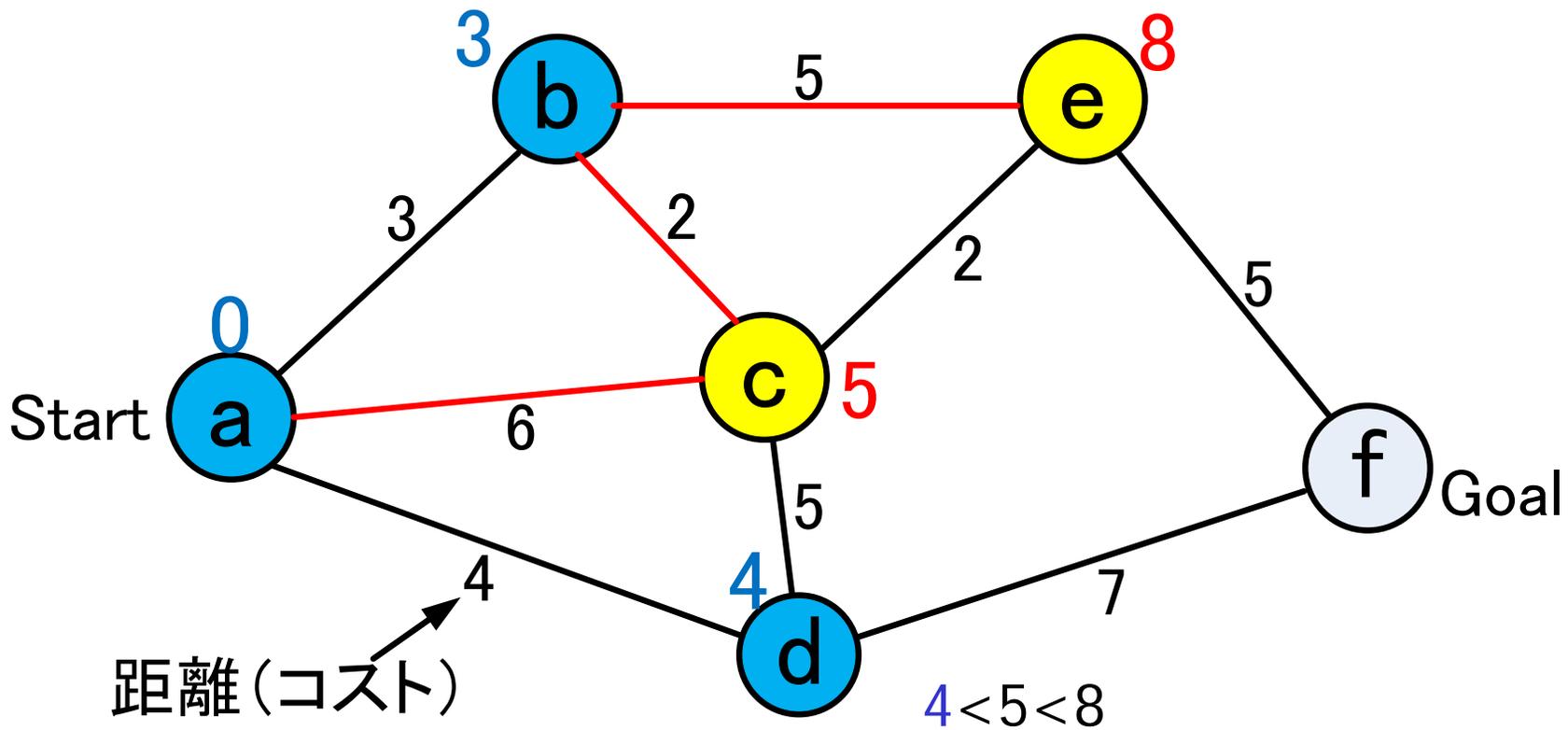
-  Startからの最短経路が確定していないノード
-  Startからの最短経路を確定中のノード
-  Startからの最短経路が確定したノード
-  Startからの最短距離候補(未確定)
-  Startからの最短距離(確定済)
-  確定済ノードからのアーク
-  次期確定ノード決定に使用

# ダイクストラ法 動作例 5/13



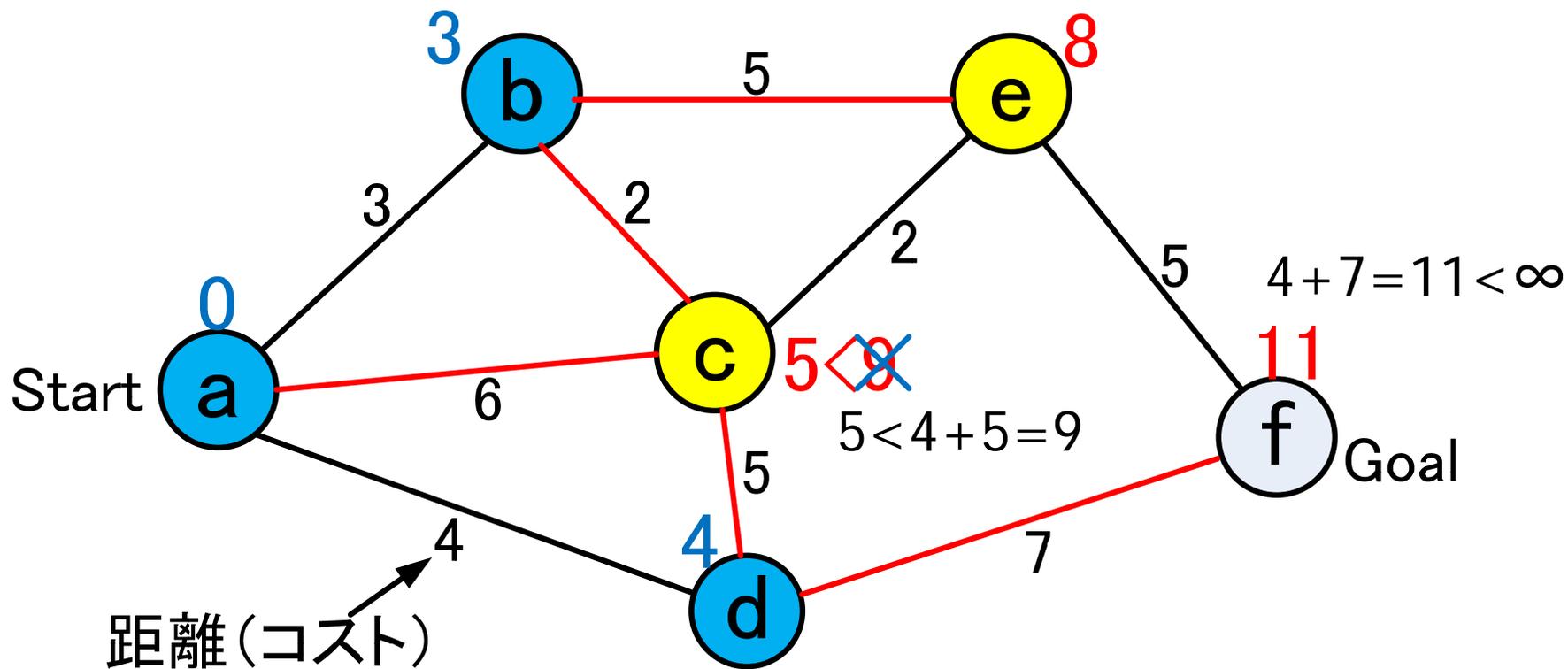
- Startからの最短経路が確定していないノード
- Startからの最短経路を確定中のノード
- Startからの最短経路が確定したノード
- 7 Startからの最短距離候補 (未確定)
- 7 Startからの最短距離 (確定済)
- 確定済ノードからのアーク
- 次期確定ノード決定に使用

# ダイクストラ法 動作例 6/13



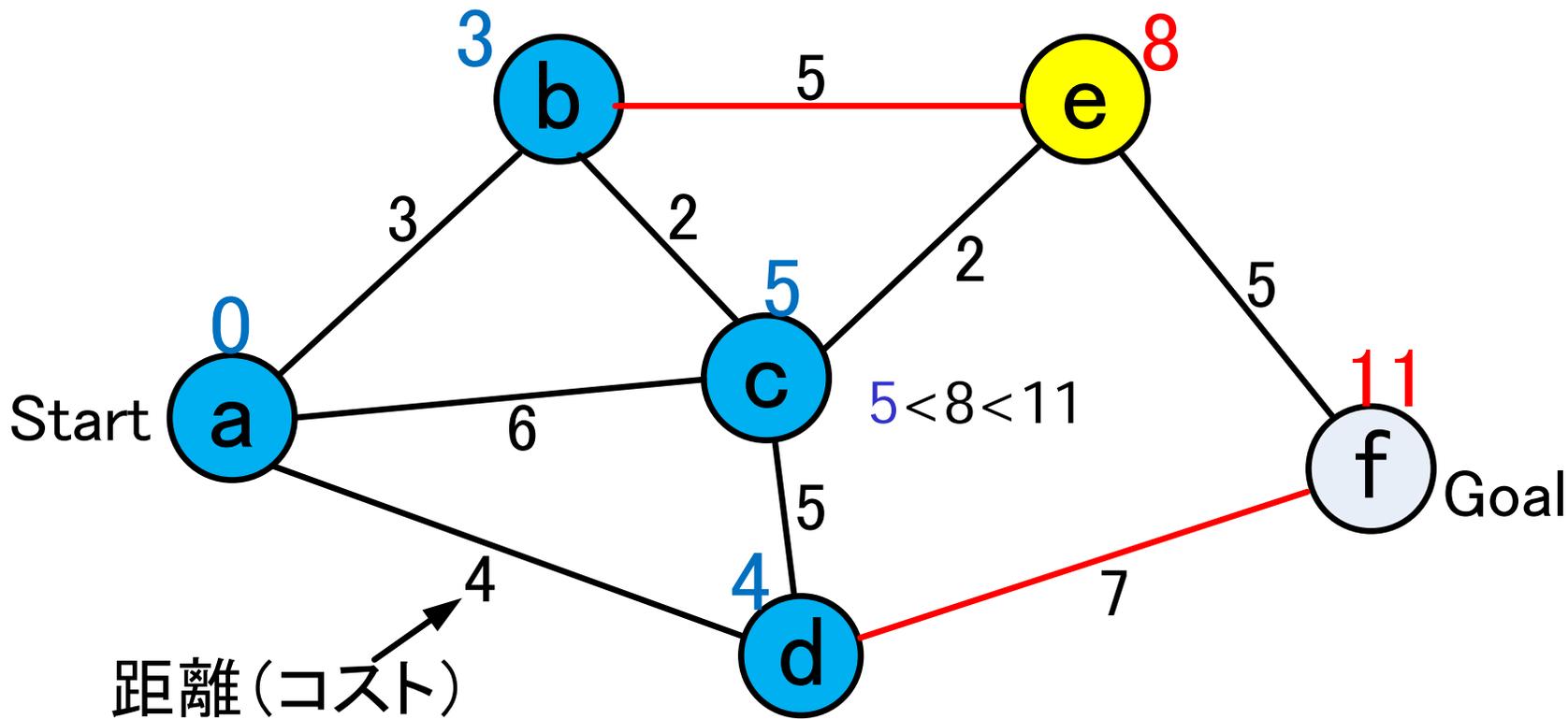
- Startからの最短経路が確定していないノード
- Startからの最短経路を確定中のノード
- Startからの最短経路が確定したノード
- 7 Startからの最短距離候補(未確定)
- 7 Startからの最短距離(確定済)
- 確定済ノードからのアーク
- 次期確定ノード決定に使用

# ダイクストラ法 動作例 7/13



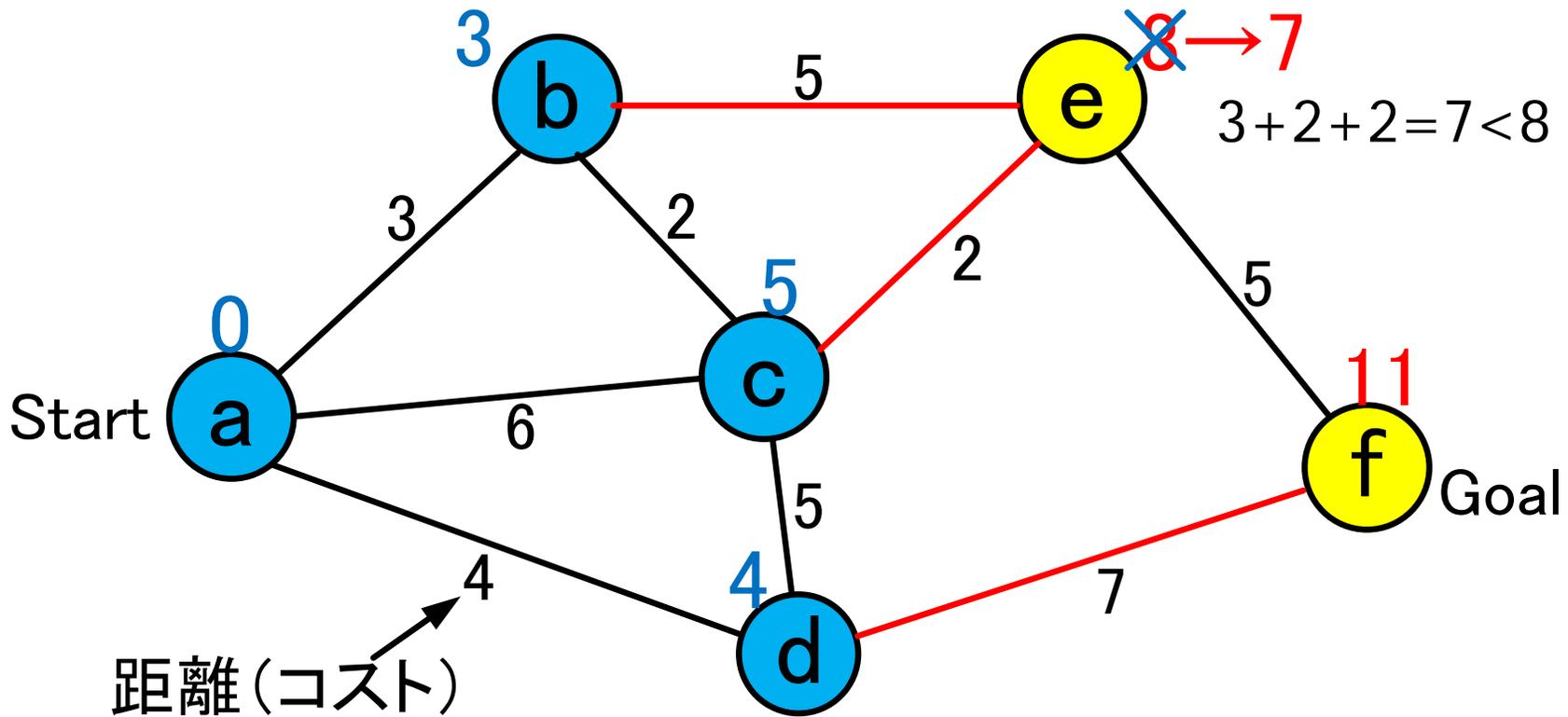
- Startからの最短経路が確定していないノード
- Startからの最短経路を確定中のノード
- Startからの最短経路が確定したノード
- 7 Startからの最短距離候補(未確定)
- 7 Startからの最短距離(確定済)
- 確定済ノードからのアーク  
次期確定ノード決定に使用

# ダイクストラ法 動作例 8/13



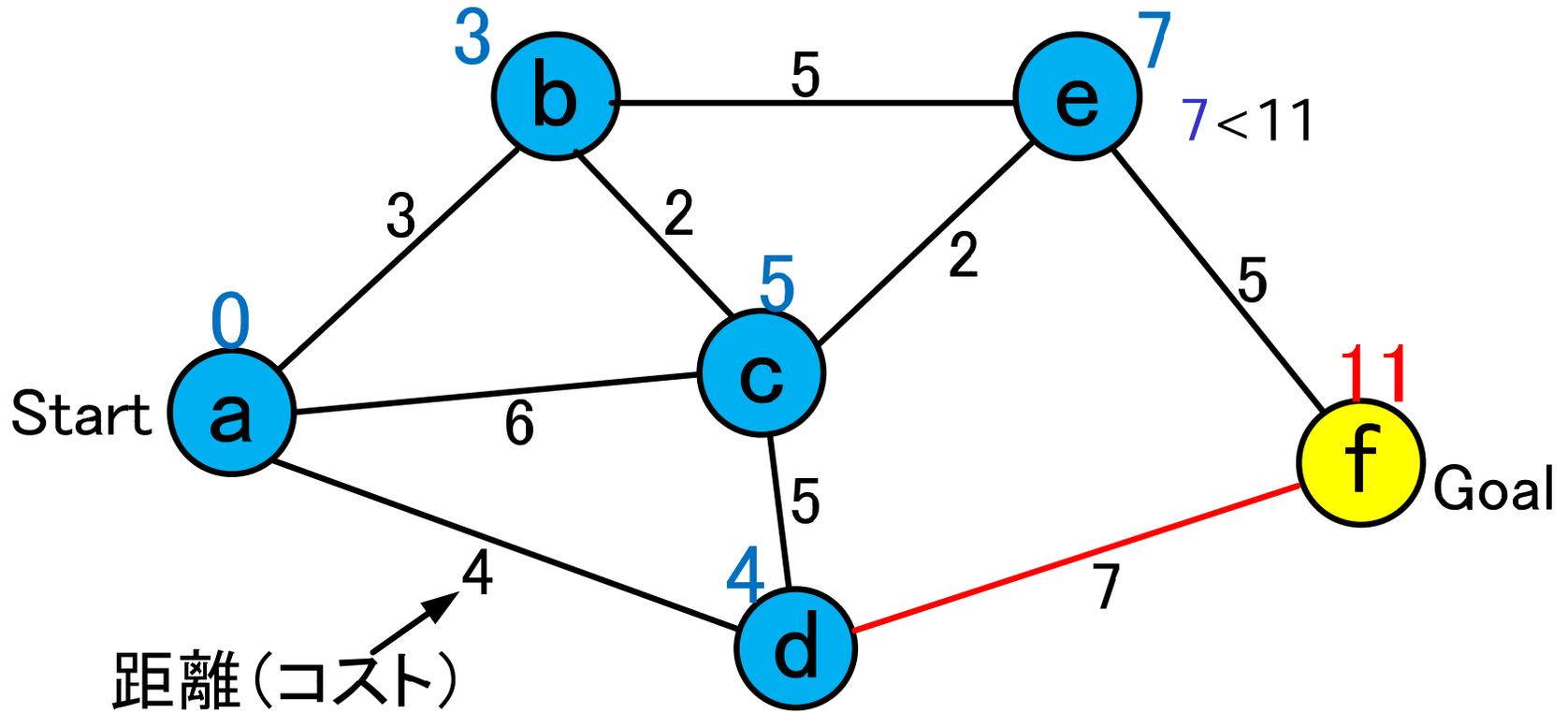
-  Startからの最短経路が確定していないノード
-  Startからの最短経路を確定中のノード
-  Startからの最短経路が確定したノード
-  Startからの最短距離候補(未確定)
-  Startからの最短距離(確定済)
-  確定済ノードからのアーク
- 次期確定ノード決定に使用

# ダイクストラ法 動作例 9/13



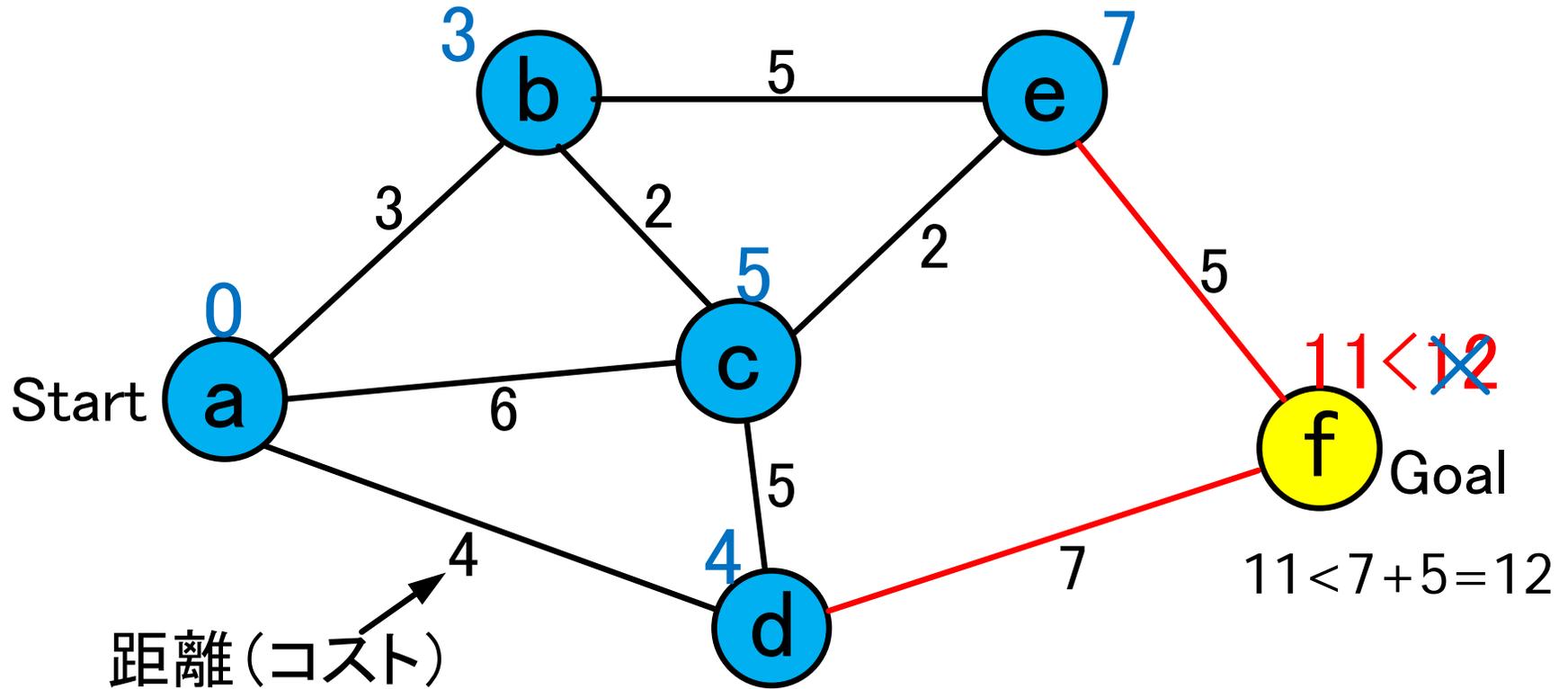
- Startからの最短経路が確定していないノード
- Startからの最短経路を確定中のノード
- Startからの最短経路が確定したノード
- 7 Startからの最短距離候補 (未確定)
- 7 Startからの最短距離 (確定済)
- 確定済ノードからのアーク  
次期確定ノード決定に使用

# ダイクストラ法 動作例 10/13



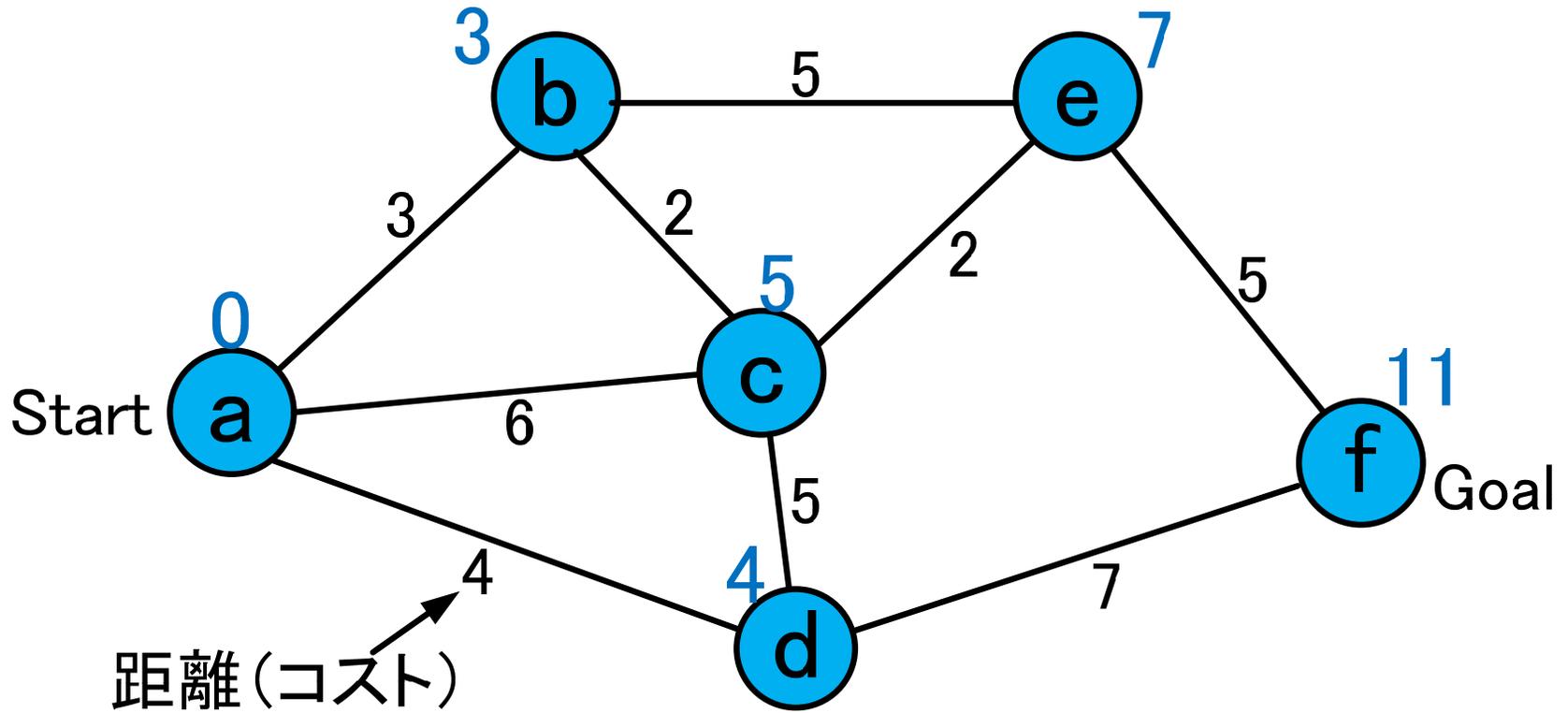
- Startからの最短経路が確定していないノード
- Startからの最短経路を確定中のノード
- Startからの最短経路が確定したノード
- 7 Startからの最短距離候補(未確定)
- 7 Startからの最短距離(確定済)
- 確定済ノードからのアーク  
次期確定ノード決定に使用

# ダイクストラ法 動作例 11/13



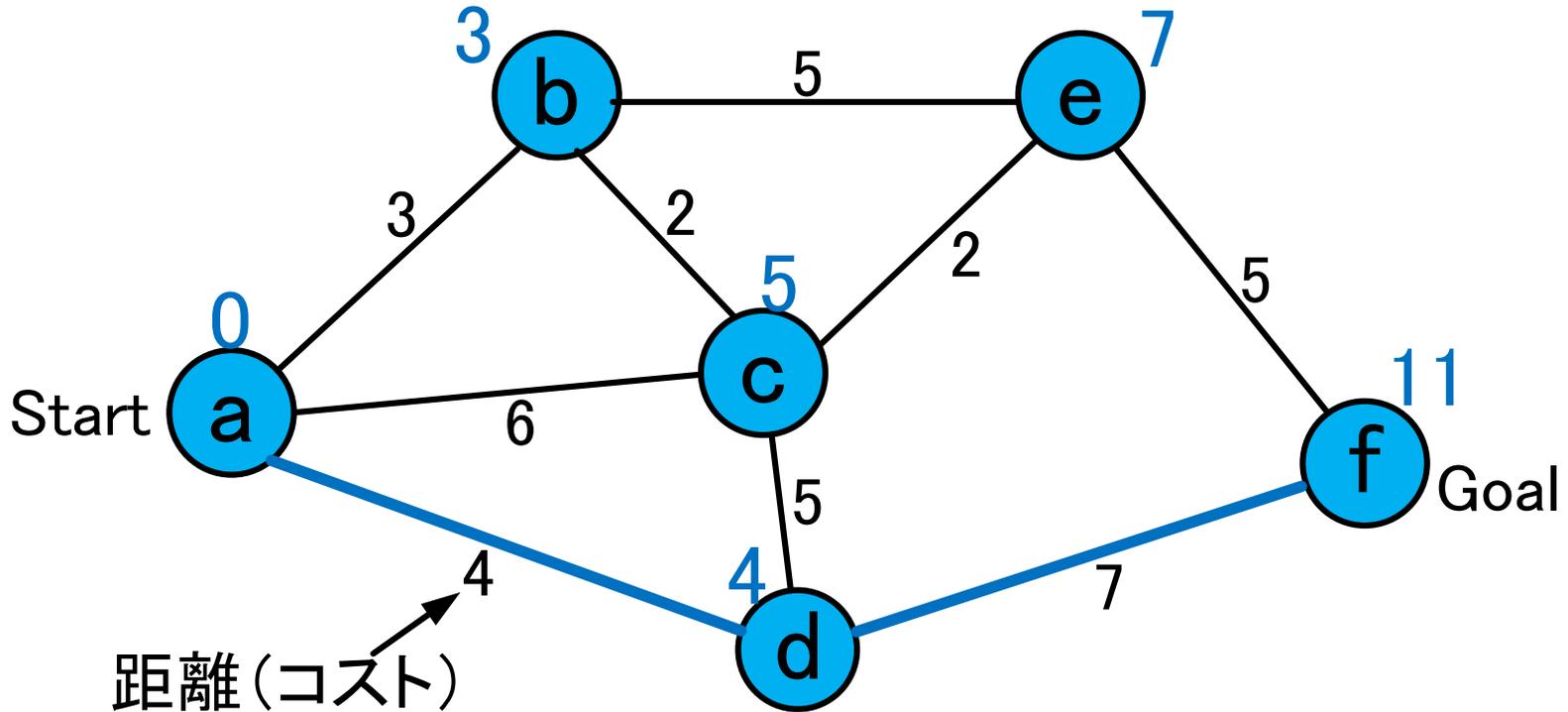
- Startからの最短経路が確定していないノード
- Startからの最短経路を確定中のノード
- Startからの最短経路が確定したノード
- 7 Startからの最短距離候補(未確定)
- 7 Startからの最短距離(確定済)
- 確定済ノードからのアーク
- 次期確定ノード決定に使用

# ダイクストラ法 動作例 12/13



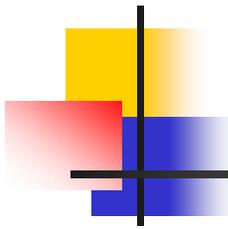
- Startからの最短経路が確定していないノード
- Startからの最短経路を確定中のノード
- Startからの最短経路が確定したノード
- 7 Startからの最短距離候補(未確定)
- 7 Startからの最短距離(確定済)
- 確定済ノードからのアーク
- 次期確定ノード決定に使用

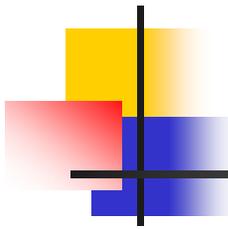
# ダイクストラ法 動作例 13/13



- Startからの最短経路が確定していないノード
- Startからの最短経路を確定中のノード
- Startからの最短経路が確定したノード
- 7 Startからの最短距離候補(未確定)
- 7 Startからの最短距離(確定済)
- 確定済ノードからのアーク  
次期確定ノード決定に使用
- StartからGoalまでの最短経路

# ダイクストラ法 アルゴリズム

- 
1. 初期化: スタートノードの値(最小コスト候補)を0, 他のノードの値を無限大に設定
  2. 未確定ノードが無くなるまで以下のループを繰り返す.
    1. 確定中ノードのうち, 最小の値を持つノードを見つけ, 確定ノードとする.
    2. 確定ノードからのエッジに対して「確定ノードまでのコスト + エッジのコスト」を計算し, そのノードの現在値よりも小さければ更新.



# ダイクストラ法の特徴

---

- 最短経路の見つけ方
  - ゴールノードから「どこから来たのか」調べ, さかのぼる(距離更新時に直前のノードを記述しておく).
- マイナスのコストを持つエッジは扱えない.
- 特定のノードからの最短距離およびその経路が全てのノードに対して求まる.
  
- 情報処理技術者試験で頻出
  - 平成15年秋期 基本情報技術者試験 午後の問題
  - 応用情報技術者試験にも出題されている

# DPマッチング

## (例: 文字列の照合)

- 2つの文字列がどのくらい似ているかを調べる.
  - Yamanashi と kamonohashi, takahashi との比較
- 音声認識にも使える
  - 音声を文字列に変換した後, 登録単語と比較
  - (現在主流の)HMM(Hidden Markov Model)に拡張可能
- DNAの比較にも使える
  - A(アデニン), G(グアニン), C(シトシン), T(チミン)の並び方の比較
  - ACTGAGCATTとCTGGACTACGの比較

# DPマッチング

## (例: 文字列の照合)

- 簡単に比較できる例
  - abcdef
  - abzdef
- A: abcdef に対して脱落, 挿入, 置換
  - A: abcdef
  - B: abdef (脱落)
  - C: abcdef (挿入)
  - D: abzdef (置換)

DPマッチング: 脱落, 挿入, 置換誤りを考慮して文字列照合可能

# DPマッチング(例:文字列の照合) 1/8

takeda と nakadai の照合

不一致コスト表

文字が一致 → 0  
文字が不一致 → 3

	n	a	k	a	d	a	i
t	3	3	3	3	3	3	3
a	3	0	3	0	3	0	3
k	3	3	0	3	3	3	3
e	3	3	3	3	3	3	3
d	3	3	3	3	0	3	3
a	3	0	3	0	3	0	3

# DPマッチング (例: 文字列の照合) 2/8

■ takeda と nakadai



の値を求める

不一致コスト表

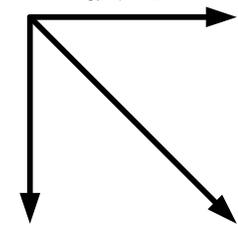
	n	a	k	a	d	a	i
t	3	3	3	3	3	3	3
a	3	0	3	0	3	0	3
k	3	3	0	3	3	3	3
e	3	3	3	3	3	3	3
d	3	3	3	3	0	3	3
a	3	0	3	0	3	0	3

1文字ずらしたけれど文字が不一致:  $1+3=4$ を加算

移動のペナルティ

横だけ1字ずらす

1



縦だけ1字  
ずらす

同時に1文字  
移動

不一致のペナルティ

文字が一致 → 0

文字が不一致 → 3

	n	a	k	a	d	a	i
t	3	7	11	15	19	23	27
a	7						
k	11						
e	15						
d	19						
a	23						

# DPマッチング (例: 文字列の照合) 3/8

takeda と nakadai

 の値を求める

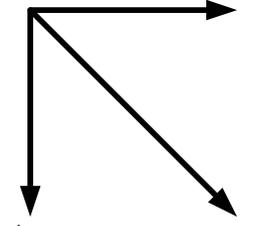
不一致コスト表

	n	a	k	a	d	a	i
t	3	3	3	3	3	3	3
a	3	0	3	0	3	0	3
k	3	3	0	3	3	3	3
e	3	3	3	3	3	3	3
d	3	3	3	3	0	3	3
a	3	0	3	0	3	0	3

移動のペナルティ

横だけ1字ずらす

1



1

縦だけ1字  
ずらす

0

同時に1文字  
移動

不一致のペナルティ

文字が一致 → 0

文字が不一致 → 3

	n	a	k	a	d	a	i
t	3	7	11	15	19	23	27
a	7	3	7	8	12	13	17
k	11						
e	15						
d	19						
a	23						

# DPマッチング (例: 文字列の照合) 4/8

■ takeda と nakadai

の値を求める

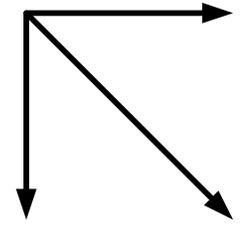
不一致コスト表

	n	a	k	a	d	a	i
t	3	3	3	3	3	3	3
a	3	0	3	0	3	0	3
k	3	3	0	3	3	3	3
e	3	3	3	3	3	3	3
d	3	3	3	3	0	3	3
a	3	0	3	0	3	0	3

移動のペナルティ

横だけ1字ずらす

1



0

同時に1文字移動

不一致のペナルティ

文字が一致 → 0

文字が不一致 → 3

	n	a	k	a	d	a	i
t	3	7	11	15	19	23	27
a	7	3	7	8	12	13	17
k	11	7	3	7	11	15	16
e	15						
d	19						
a	23						

# DPマッチング (例: 文字列の照合) 5/8

■ takeda と nakadai

 の値を求める

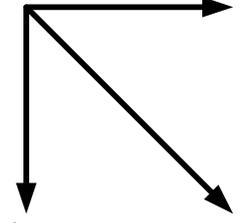
不一致コスト表

	n	a	k	a	d	a	i
t	3	3	3	3	3	3	3
a	3	0	3	0	3	0	3
k	3	3	0	3	3	3	3
e	3	3	3	3	3	3	3
d	3	3	3	3	0	3	3
a	3	0	3	0	3	0	3

移動のペナルティ

横だけ1字ずらす

1



1

縦だけ1字  
ずらす

0

同時に1文字  
移動

不一致のペナルティ

文字が一致 → 0

文字が不一致 → 3

	n	a	k	a	d	a	i
t	3	7	11	15	19	23	27
a	7	3	7	8	12	13	17
k	11	7	3	7	11	15	16
e	15	11	7	6	10	14	18
d	19						
a	23						

# DPマッチング (例: 文字列の照合) 6/8

■ takeda と nakadai

■ の値を求める

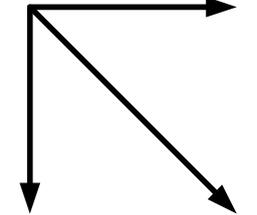
不一致コスト表

	n	a	k	a	d	a	i
t	3	3	3	3	3	3	3
a	3	0	3	0	3	0	3
k	3	3	0	3	3	3	3
e	3	3	3	3	3	3	3
d	3	3	3	3	0	3	3
a	3	0	3	0	3	0	3

移動のペナルティ

横だけ1字ずらす

1



0

同時に1文字

移動

不一致のペナルティ

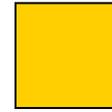
文字が一致 → 0

文字が不一致 → 3

	n	a	k	a	d	a	i
t	3	7	11	15	19	23	27
a	7	3	7	8	12	13	17
k	11	7	3	7	11	15	16
e	15	11	7	6	10	14	18
d	19	15	11	10	6	10	14
a	23						

# DPマッチング (例: 文字列の照合) 7/8

takeda と nakadai



の値を求める

不一致コスト表

	n	a	k	a	d	a	i
t	3	3	3	3	3	3	3
a	3	0	3	0	3	0	3
k	3	3	0	3	3	3	3
e	3	3	3	3	3	3	3
d	3	3	3	3	0	3	3
a	3	0	3	0	3	0	3

移動のペナルティ

横だけ1字ずらす

1

1

0

同時に1文字

移動

不一致のペナルティ

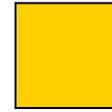
文字が一致 → 0

文字が不一致 → 3

	n	a	k	a	d	a	i
t	3	7	11	15	19	23	27
a	7	3	7	8	12	13	17
k	11	7	3	7	11	15	16
e	15	11	7	6	10	14	18
d	19	15	11	10	6	10	14
a	23	16	15	11	10	6	10

# DPマッチング (例: 文字列の照合) 8/8

takeda と nakadai



の値を求める

不一致コスト表

	n	a	k	a	d	a	i
t	3	3	3	3	3	3	3
a	3	0	3	0	3	0	3
k	3	3	0	3	3	3	3
e	3	3	3	3	3	3	3
d	3	3	3	3	0	3	3
a	3	0	3	0	3	0	3

移動のペナルティ

横だけ1字ずらす

1

1

0

同時に1文字

移動

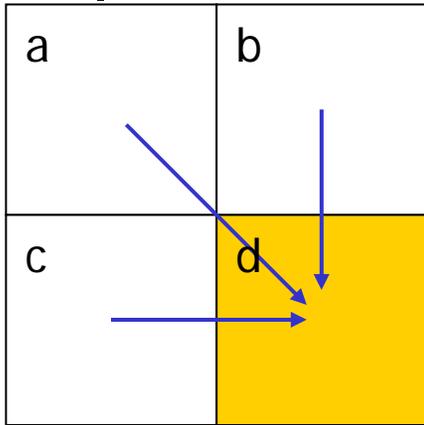
不一致のペナルティ

文字が一致 → 0

文字が不一致 → 3

	n	a	k	a	d	a	i
t	3	7	11	15	19	23	27
a	7	3	7	8	12	13	17
k	11	7	3	7	11	15	16
e	15	11	7	6	10	14	18
d	19	15	11	10	6	10	14
a	23	16	15	11	10	6	10

# DPマッチング(例:文字列の照合) アルゴリズム



**d** へのルートは3種類

aまでの距離+斜め移動のペナルティ+不一致ペナルティ

bまでの距離+下移動のペナルティ+不一致ペナルティ

cまでの距離+右移動のペナルティ+不一致ペナルティ

の内の最短距離をdに書き込む

## 移動のペナルティ

横だけ1字ずらす

1

1

縦だけ1字  
ずらす

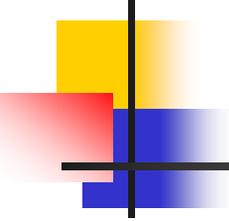
同時に1文字  
移動

0

## 不一致のペナルティ

文字が一致 → 0

文字が不一致 → 3



# DPマッチングの応用

---

- DPマッチングの探索空間を制限し、探索時間を削減する方法
  - ビームサーチ
  - A\*アルゴリズム
- HMM(隠れマルコフモデル)とビタビアルゴリズム
  - 音声認識手法の主流

# A\*アルゴリズム

## 最短経路探索問題

- ダイクストラ法にすこし工夫を加えた方法
- 各ノードからゴールまでの推定距離を利用
  - $0 \leq \text{推定距離} \leq \text{最短距離}$  でなければならない
  - 推定距離=0なら推定していないと同じ→ダイクストラ法