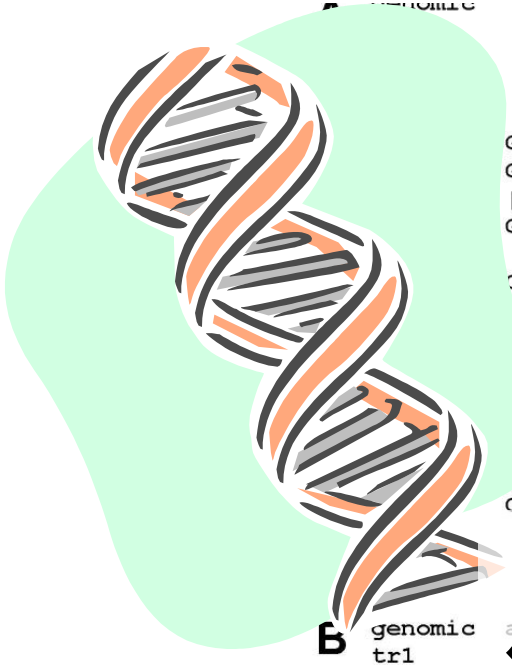
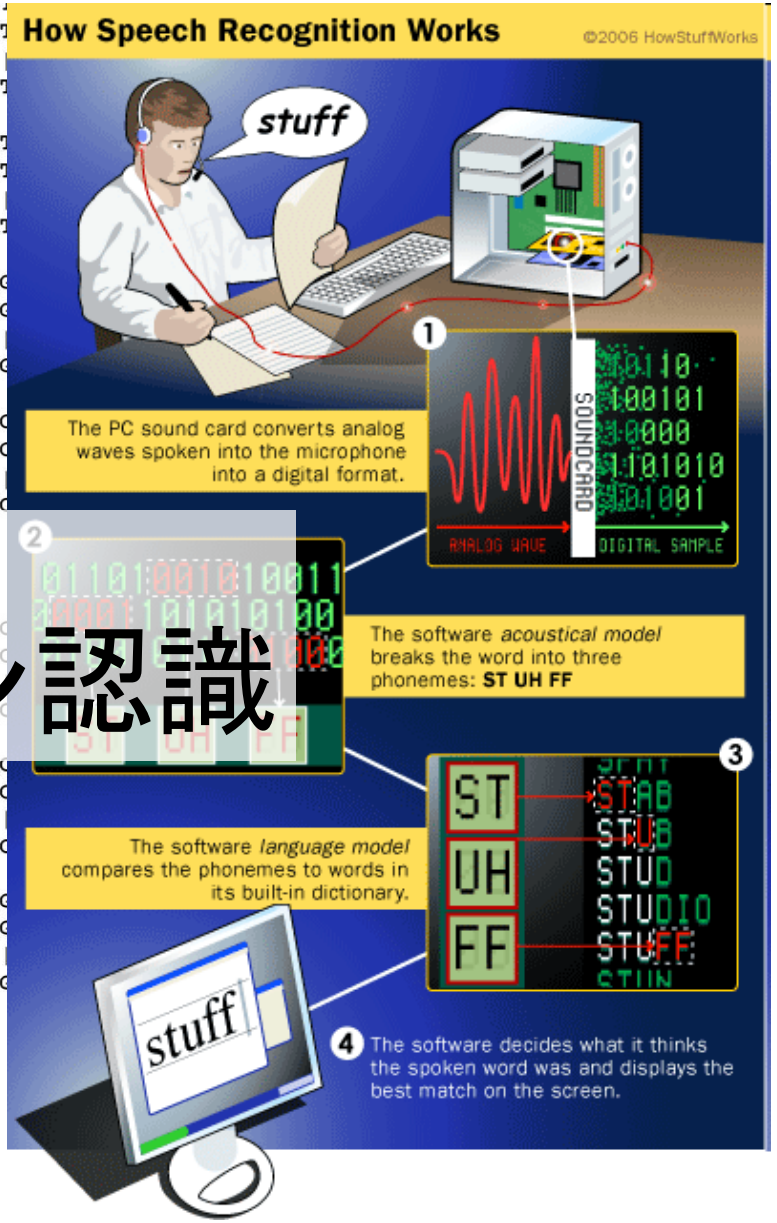


情報科学第12週

第7章 パターン認識



パターン認識

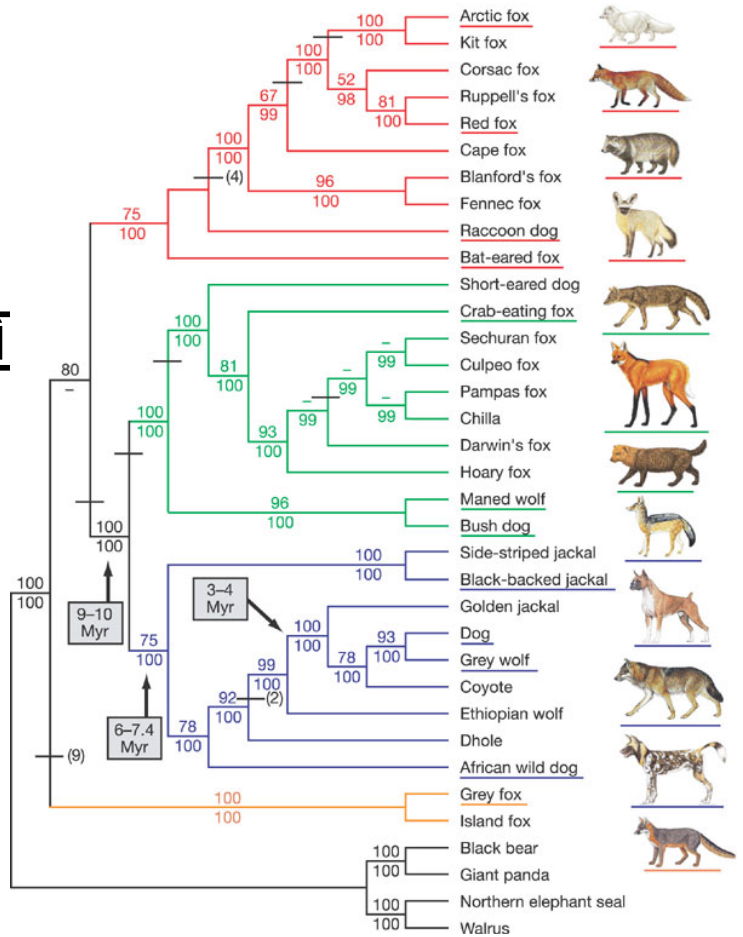
- 音や画像に中に隠れた
  パターンを認識する。
  - 音素・音節・単語・文・・・
  - 基本図形・文字・指紋・物体・人物・顔・・・
- 「パターン」は唯一のデータではなく、
  似通ったデータの集まりを表している。
  - 多様性
  - ノイズ
- 「等しい」から「似ている」へ
- 「～だ」から「～らしい」へ

「等しい」から「似ている」へ

- 完全に等しいかどうかではなく、
  「似ているか」どうかを判定する。
  - パターンを代表する模範的データと
    どのくらい似ているか。
- 例：二つの文字列の比較
  - 多少の文字の食い違いや、
    文字の欠落を許して、似ているか。
  - アラインメントという。
- 動的計画法を活用。

応用例: 系統樹の作成

- 旧来の系統樹: 見た目や行動様式から近さを推定
- DNAを用いた系統樹:
  塩基配列の似てる度を計算
  → 分化した年代を推定
  - 似てる度: 塩基の欠落や置き換えを考慮した一致数



Kerstin Lindblad-Toh, et al., Genome sequence, comparative analysis and haplotype structure of the domestic dog, *Nature* 438, 803-819 (8 December 2005)

クイズ: ヒトのDNAに含まれる塩基配列の長さは?

(1つの塩基対を長さ1とする)

1. 約2万2千
2. 約464万
3. 約1億
4. 約31億

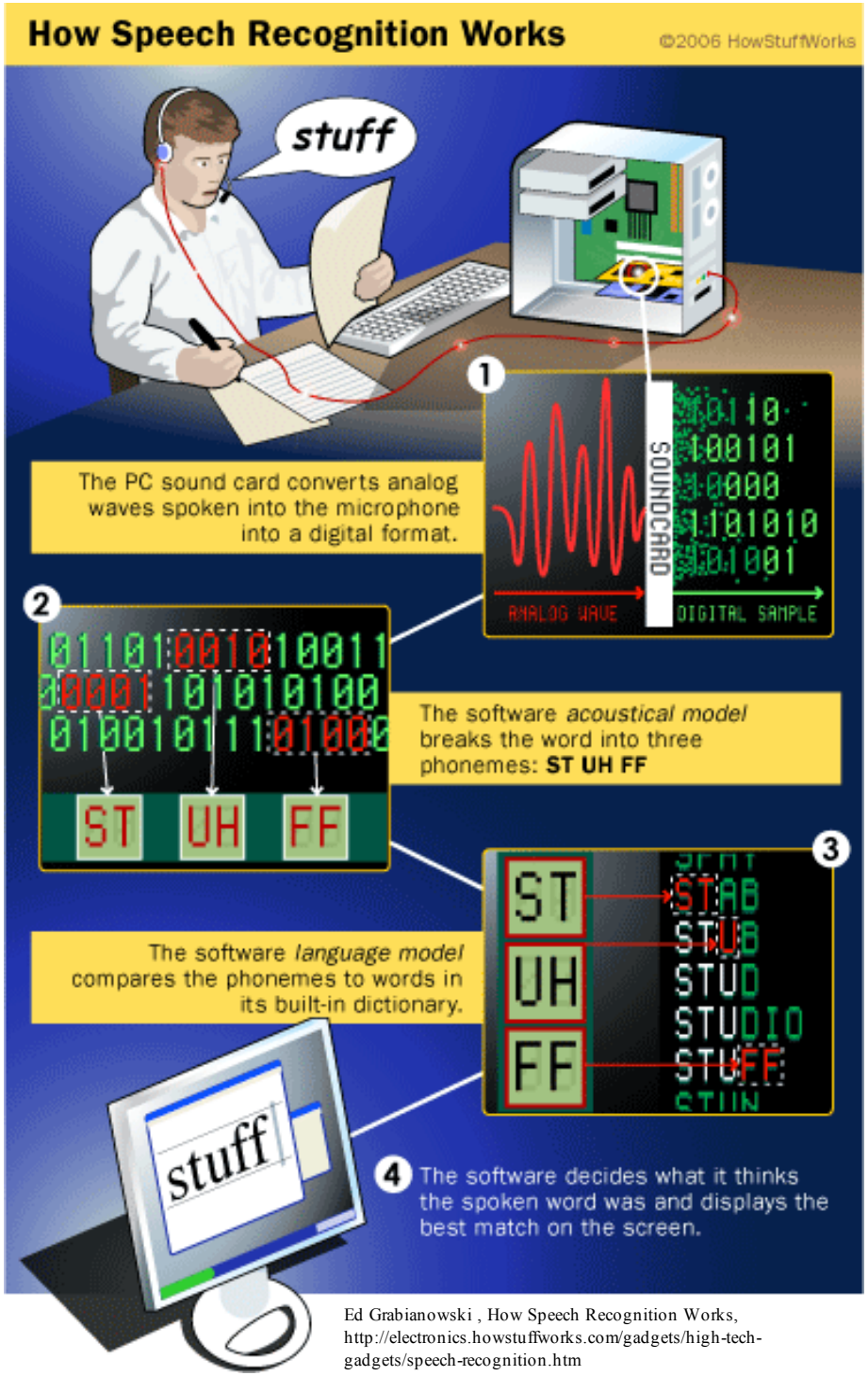
ターミナルで

```
$ cd Downloads
$ ruby c.rb 1から4
```

応用例: 音声認識

- 音素ごとの波形を用意
- 単語ごとに音素列を用意

- 入力した波形と照合、「似ている」音素を(複数)選び並べる
- 音素の並びに最も近い発音を持つ語を探し並べる



Ed Grabianowski , How Speech Recognition Works, http://electronics.howstuffworks.com/gadgets/high-tech-gadgets/speech-recognition.htm

DNAやタンパクの比較

- DNA --- 塩基（四種類）の配列
  - セントラル・ドグマ

    DNA → RNA → タンパク
  - 三塩基（コドン）が一つのアミノ酸に
  - 似ている配列は似ているタンパクに
  - 似ている配列は同じ祖先から進化
- タンパク --- アミノ酸（20種類）の配列
  - 似ている配列は似た構造に
  - 似ている配列は似た機能を

どれとどれが似ている?

1．GATTAGGCAG

2．GACGGATTAG

3．GCTTCGTTGATTAG

4．GATCGGAATAG

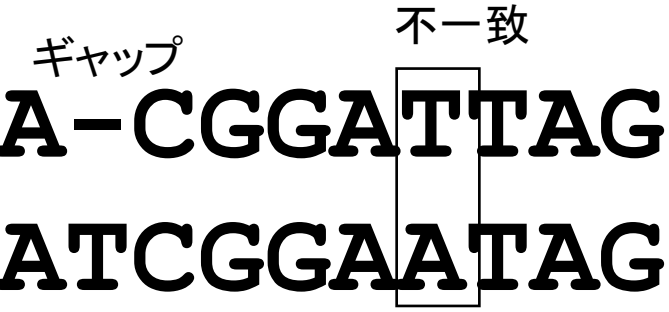
	1	**2**	**3**	**4**
1		2	3	6
2	2		9	15
3	3	9		10
4	6	15	10	

- どうして?
- 機械的に判定するには?

アラインメント

- アラインメント
  二つ(複数)の文字列の比較
  - 音声認識・文字認識
  - DNAやタンパクの比較

GACGGATTAG と GATCGGAATAG



アラインメント

- ぴったり同じでなくとも、
  似ているかどうかを判定。
- スコア
  - 一致
  - 不一致
  - ギャップ

  足し合わせる → アラインメントの類似度
- 類似度が最大の
  アラインメント（ギャップの入れ方）を求める。

最適化

スコア:　一致　+2
不一致　−1
ギャップ　−2

アラインメントの例

ATAG と AACの場合

A	T	A	G
A	A	C	

A	T	A	G
A	A		C

A	T	A	G
A		A	C

A	T	A	G
	A	A	C

	A	T	A	G
A	A	C		

A		T	A	G
A	A	C		

A	T		A	G
A	A	C		

A	T	A		G
A	A		C	

A		T	A	G
A	A		C	

最小アラインメントの求め方

スコア: 一致 +2
不一致 −1
ギャップ −2

ATAG と **AAC** をアラインするには

ATA と **AA** のアラインメントに **G** と **C** を付加

2 **ATA** → **ATAG** 1 これを採用！
A-A **A-AC**

ATA と **AAC** のアラインメントに **G** と **-** を付加

0 **ATA** → **ATAG** −2
AAC **AAC-**

ATAG と **AA** のアラインメントに **-** と **G** を付加

0 **ATAG** → **ATAG-** −2
A-A- **A-A-C**

再帰版のアラインメント

```
def align_sub(s,t,i,j)
  if i==0 || j==0
    i*g() + j*g()
  else
    v0 = align_sub(s,t,i,  j-1) + g()
    v1 = align_sub(s,t,i-1,j-1) + q(s[i-1],t[j-1])
    v2 = align_sub(s,t,i-1,j)   + g()
    max(v0,max(v1,v2))
  end
end
def align_rec(s,t)
  align_sub(s,t,s.length(),t.length())
end
```

g() : ギャップのペナルティ

q(c, d) : c と d の類似度

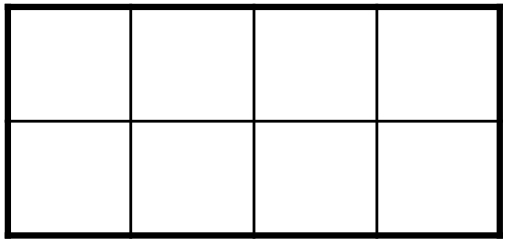
試してみよう:

```
align_rec("ATAG","AAC")
align_rec("AAAAA","AAAAA")
align_rec("AAAAAA","AAAAAA")
align_rec("AAAAAAA","AAAAAAA")
```

align_rec.rb

アラインメントの可能性

- 長さ2の配列2つにギャップを入れる組み合わせは何通り?
  - 1配列に3個以上ギャップを入れても無意味
  - ギャップ0個: 1通り
  - ギャップ1個: 3×2=6通り
  - ギャップ2個: ${}_4C_2 \times {}_2C_2 = 6$通り
  - 計: 1+6+6=13通り



クイズ: アライメントの可能性

30文字の2つの配列に、
ギャップを入れる組み合わせは何通り?

1. 60通り
2. 900通り
3. 3991万6800通り
4. 約45.5億通り
5. 約9.6×10^{21}通り

ターミナルで

```
$ cd Downloads
$ ruby c.rb 1から5
```

アライメントの可能性

30文字どうしの場合で考える
ギャップを

(入れない)　　1通り
+ (1箇所に入れる)　　31×30通り
+ (2箇所に入れる)　　${}_{32}C_2 \times {}_{30}C_2 = 215760$通り
+ (3箇所に入れる)　　${}_{33}C_3 \times {}_{30}C_3 = 22151360$通り
…
+ (30箇所に入れる)　　${}_{60}C_{30} \times {}_{60}C_{30}$通り

$$= \sum_{k=0}^{30} {}_{30+k}C_k \cdot {}_{30}C_k$$   通り

ターミナルで

```
$ cd Downloads
$ ruby c.rb ??/??
```

動的計画法

- プロ野球日本シリーズにおける優勝の確率

P(i, j): Aがシリーズに勝つまでにあとi勝、
Bはあとj勝という状況で、
最終的にAがシリーズに勝つ確率

$$P(i, j) = 1 \quad (i=0 \text{ かつ } j>0)$$
$$= 0 \quad (i>0 \text{ かつ } j=0)$$
$$= (P(i-1, j) + P(i, j-1))/2 \quad (i>0 \text{ かつ } j>0)$$

i →
j ↓

	0	0	0	0
1	1/2	1/4	1/8	
1	3/4	1/2		
1	7/8			

ここは？

AとBは同じ強さと仮定

動的計画法

- テーブルを用意して、
  再帰的な計算の重複を避ける。
- テーブルの中身を順に埋めることにより、
  求める値を計算する。
- 各種の最適化問題に適用。
  - アラインメント
  - DNA・RNAのエネルギー最小化
  - 構文解析

アラインメントの動的計画法

- 二つの配列s0とs1の間の類似度
- a[i][j]：s0の部分配列s0[0],...,s0[i-1]と
  s1の部分配列s1[0],...,s1[j-1]の間の類似度
- a[i][j] = max{a[i][j-1] + g,
  a[i-1][j-1] + q(s0[i-1], s1[j-1]),
  a[i-1][j] + g }

  g() : ギャップのペナルティ（負の数）

  q(c, d) : c と d の類似度

  - c と d が等しければ適当なスコア（正）
  - 似ていればそれなりのスコア
  - 似ていなければ不一致のペナルティ（負）

境界条件

- a[0][0] = 0
- a[0][j] = a[0][j-1] + g() (j>0)
- a[i][0] = a[i-1][0] + g() (i>0)

例えば g = -2

- 結局

  a[0][j] = j*g()

  a[i][0] = i*g()

- となる。要するに、ギャップばかり。

スコア:　一致　+2
不一致　−1
ギャップ　−2

A-AC
ATAG

	s1	**A**	**T**	**A**	**G**
s0	0	−2	−4	−6	−8
A	−2				
A	−4				
C	−6				

スコア:　一致　+2
不一致　−1
ギャップ　−2

A-AC
ATAG

	s1	**A**	**T**	**A**	**G**
s0	0	−2	−4	−6	−8
A	−2	2	0	−2	−4
A	−4	0	1	2	0
C	−6	−2	−1	0	

A-A
ATA

A-A-
ATAG

AAC
ATA

スコア:　一致　+2
不一致　−1
ギャップ　−2

A-AC
ATAG

	s1	**A**	**T**	**A**	**G**
s0	0	−2	−4	−6	−8
A	−2	2	0	−2	−4
A	−4	0	1	2	0
C	−6	−2	−1	0	1

A-A
ATA

A-A-
ATAG

AAC
ATA

A-AC
ATAG

align.rb

ギャップ
ペナルティ

```
def g()
 -2
end
```

文字 c と文字 d の
類似度を返す

```
def q(c,d)
    if c == d
        2
    else
        -1
    end
end
```

一致

不一致

```
def align(s0,s1)
    m = s0.length()
    n = s1.length()
    a = make2d(m+1,n+1)
    a[0][0] = 0
    for j in 1..n
        a[0][j] = a[0][j-1] + g()
    end
    for i in 1..m
        a[i][0] = a[i-1][0] + g()
    end
    for i in 1..m
        for j in 1..n
            # ここを埋めてみよう
        end
    end
    a
end
```

境界条件

完成したら
align("AAC","ATAG")
align("ATC","ATAC")
など

スコア:　一致　+2
不一致　−1
ギャップ　−2

ここは？

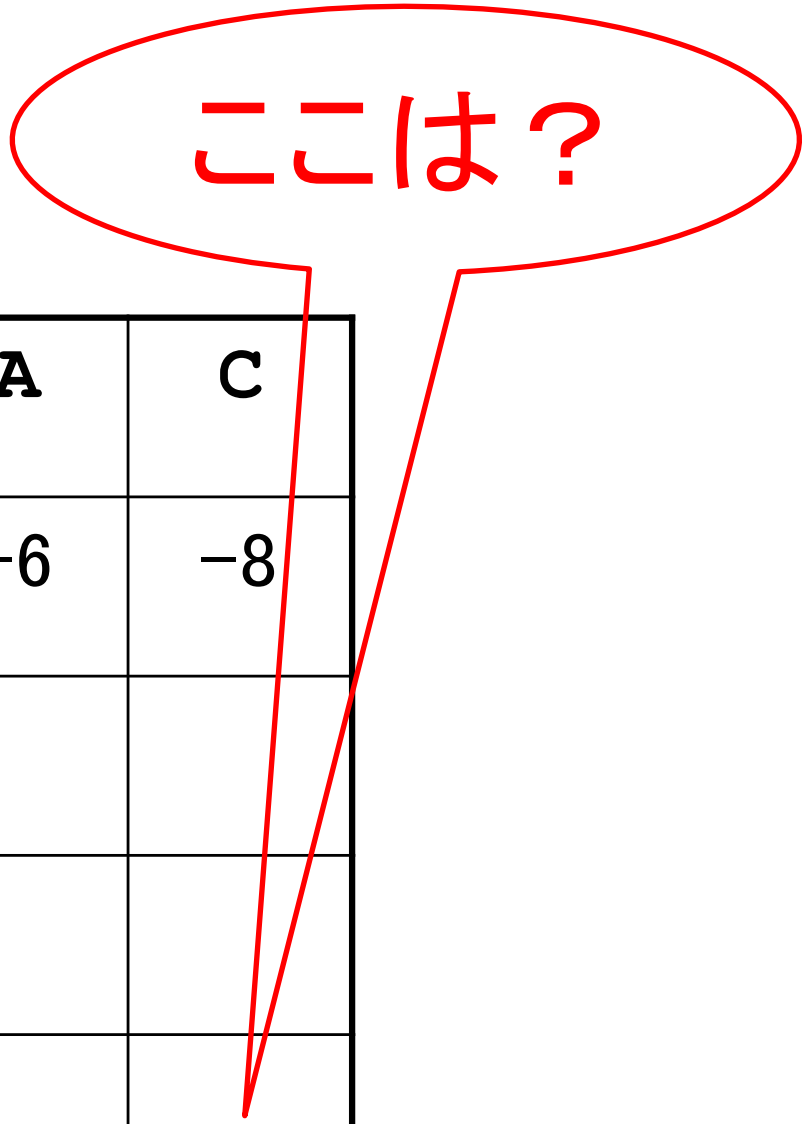
	s1	**A**	**T**	**A**	**C**
s0	0	−2	−4	−6	−8
A	−2				
T	−4				
C	−6				

スコア:　一致　+2
不一致　−1
ギャップ　−2

ここは？

	s1	**A**	**T**	**A**	**C**
s0	0	−2	−4	−6	−8
A	−2				
T	−4				
C	−6				

スコア:　一致　+2
不一致　−1
ギャップ　−2



	s1	**A**	**T**	**A**	**C**
s0	0	−2	−4	−6	−8
A	−2				
T	−4				
C	−6				

トレースバック

- 最大の類似度を与えるアラインメントを提示するために、配列の最後から、それまでの選択を振り返る(トレースバック)。

スコア:　一致　+2
不一致　−1
ギャップ　−2

A−AC
ATAG

→j

↓i

	s1	**A**	**T**	**A**	**G**
s0	0	−2	−4	−6	−8
A	−2	2	0	−2	−4
A	−4	0	1	2	0
C	−6	−2	−1	0	1

u =

v =

トレースバック

```
def traceback(a,s,t)
  u = ""
  v = ""
  i = s.length()
  j = t.length()
  while i>0 || j>0
    if j>0 && a[i][j] == a[i][j-1] + g()
      u = "-"           + u
      v = t[j-1 .. j-1] + v
      j = j - 1 # go left
    else
↗
```

```
↘
      if i>0 && j>0 &&
         a[i][j] == a[i-1][j-1] + q(s[i-1], t[j-1])
        左上から求められた場合
      else
        if i>0 && a[i][j] == a[i-1][j] + g()
          上から求められた場合
        end
      end
    end
  end
  [u,v]
end
```

練習

- align.rb を完成させて、**ATAG** と **AAC** のアラインメントを求めよう。
- RNase_P.rb をロードすると、文字列を返す（引数なしの）関数 seq0() と seq1() が定義される。これらのアラインメントを求めよう。

進捗状況の確認

1. align の結果に対して traceback が正しく動いた時点で投票してください。
2. align は動いたが、traceback の結果が正しくない。
3. align はできたが、うまく動かない。
4. align がまだできていない。

ターミナルで

```
$ cd Downloads
$ ruby c.rb 1から4
```

今日の内容

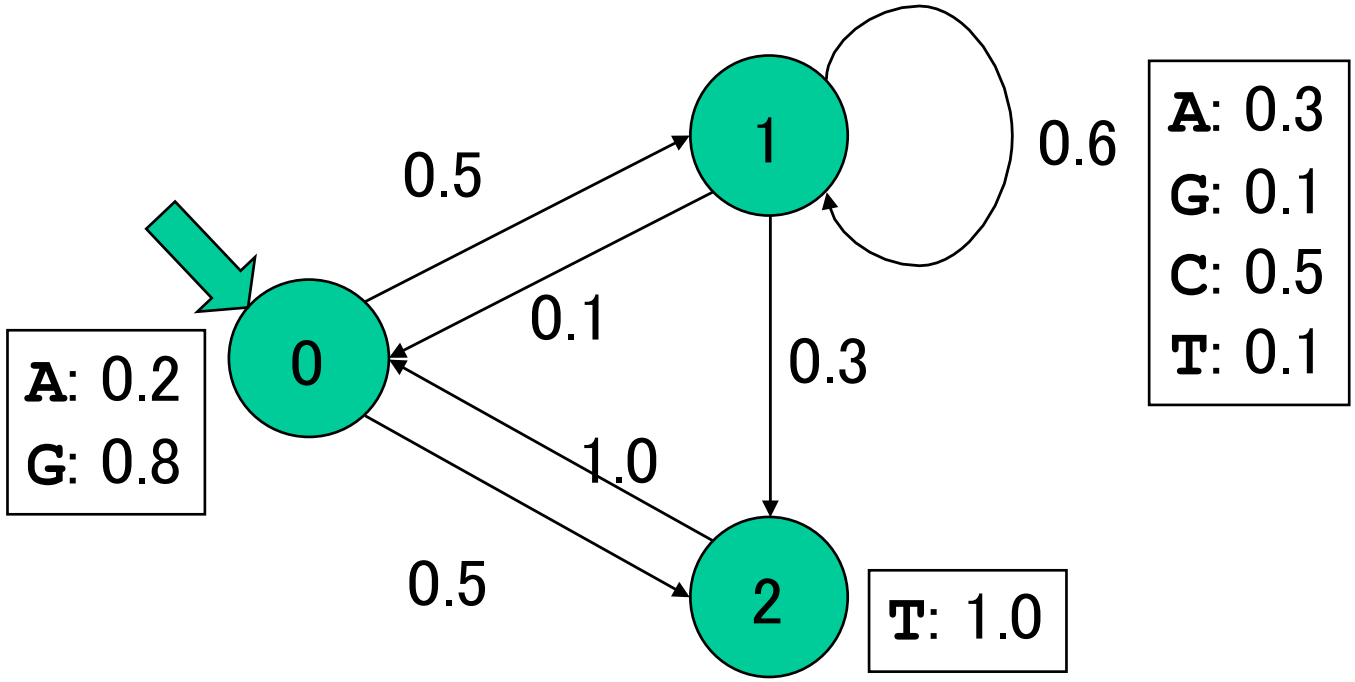
- パターン認識問題の1つ: アラインメント
- アルゴリズムの1つ: 動的計画法

「～だ」から「～らしい」へ

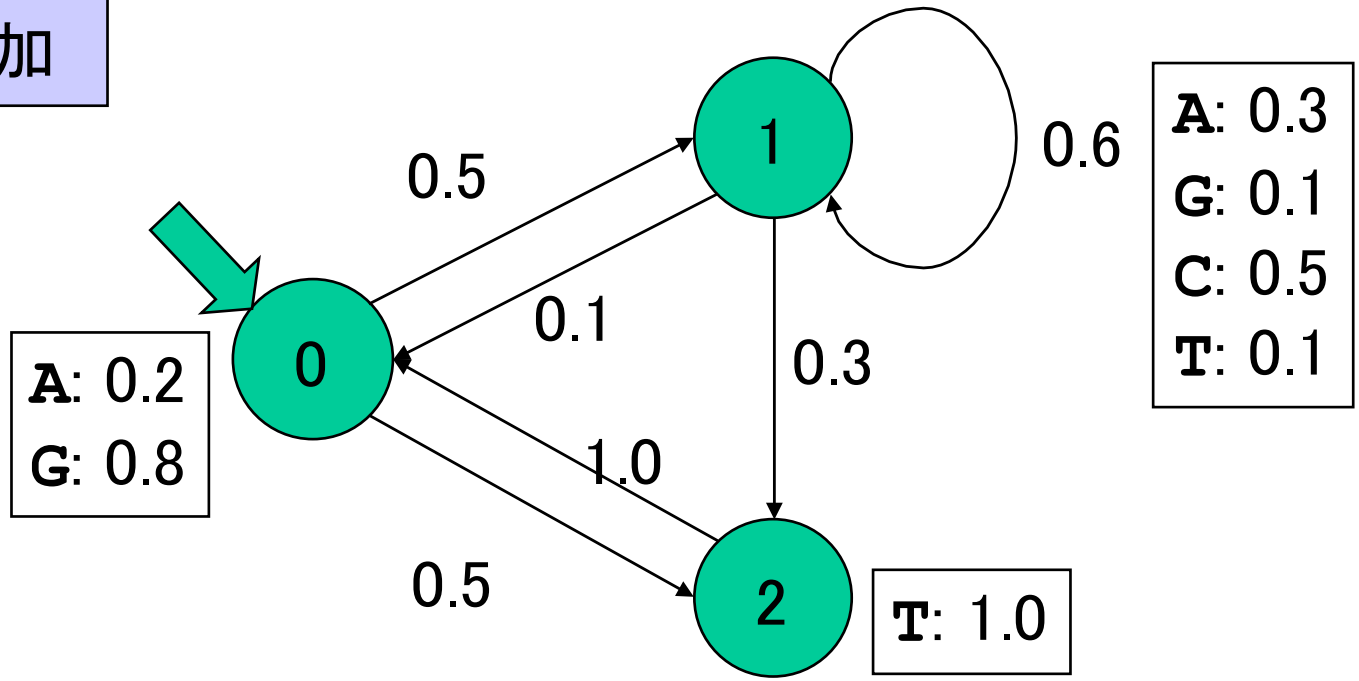
- あるデータがパターンに従っているか、
  否かを、断定するのではなく、
  その確からしさを求める。
- 例：文字列の判別
  - 文字列がパターンに従っている
    確からしさを求める。
  - 隠れマルコフモデル
- ここでも動的計画法を活用。

隠れマルコフモデル

- 有限オートマトンに確率を付加したようなもの。
- 遷移ではなく、状態に文字が対応。
  出力文字と考える。（本質的ではない。）
- 各遷移と各出力文字に確率が与えられる。



遷移と出力に
確率が付加



状態遷移

0 1 0　　0.5*0.1 = 0.05

0 1 1　　0.5*0.6 = 0.30

0 1 2　　0.5*0.3 = 0.15

…

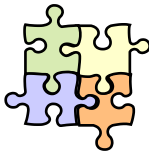
出力文字列

GCG　　0.8*0.5*0.8=0.32 → 0.016

GCG　　0.8*0.5*0.1=0.04 → 0.012

GCT　　0.8*0.5*0.1=0.04 → 0.012

GCT　　0.8*0.5*1.0=0.40 → 0.060



隠れマルコフ過程

- マルコフ過程
  - 次の状態は、現在の状態のみに依存し、過去の履歴には依存しない。
- 隠れ？
  - 各状態の出力も確率的に定まる。
  - 従って、状態遷移は直接観測できない。
- 観測可能な現象の背後にある確率的なモデル

実際のパターン認識

- 音声認識
  - 波形データから音素を抽出
  - 音素列に対する隠れマルコフモデル
- 手書き文字認識
  - ストロークをセグメントに分割
  - セグメントの列に対するアラインメント
- 画像認識
  - 様々な前処理
    - エッジ検出・背景分別・・・
  - アラインメント・隠れマルコフモデル・・・

実際は
はるかに複雑

実際は
もっと複雑

画像の種類に
応じた様々な
手法