情報理論（No.5）　　2016/10/15

今後の講義予定

10／22　:休講(平塚祭)

10／29　:通常(第6回)

11／5　　:通常(第7回)

11／12　:通常(第8回)

講義目次

- 6. データ圧縮理論
  - 6.1 データ圧縮理論
  - 6.2 データ圧縮の種別
  - 6.3 ユニバーサル符号化
  - 6.4 LZ符号
  - 6.5 有歪み圧縮
- 情報源符号化のまとめ

6. データ圧縮理論

6.1 データ圧縮理論
6.2 データ圧縮の種別
6.3 ユニバーサル符号化
6.4 LZ符号
6.5 有歪み圧縮

6.1 データ圧縮理論

- データ圧縮は情報源系列をできるだけ短いビット系列に変換すること．「情報源の符号化」と同等と見ることもできる．
  - これまで説明してきた情報源符号化（例えばハフマン符号やシャノン・ファノ符号など）は，一定の仮定の下でのデータ圧縮であった．
  - 本節ではこれらを含めた一般的なデータ圧縮を扱う．

6.2 データ圧縮の種別(１)

- 元の情報量を保存するか否か
  - 無歪み圧縮(可逆圧縮)
    - データを復元したときに、完全に元にもどる圧縮方法．文章やプログラム，数値データなど，復号時に1ビットの誤りも許されないようなデータに用いる．
  - 有歪み圧縮(非可逆圧縮)
    - データを復元したときに、完全には元にもどらない圧縮方法．音声や動画のようにある程度の歪みを許しても高能率な圧縮を実現したい場合に用いる．

6.2 データ圧縮の種別(２)

- 情報源系列の確率構造(各事象の発生する確率分布)が分かっているかどうかによる分類
- エントロピー符号化とユニバーサル符号化
  - エントロピー符号化
    - 確率構造が既知の場合に適用される符号化．符号構造の違いから次の分類がある
      - FV符号とVF符号

        FV符号は固定長入力→可変長出力，VF符号はその逆

        FV符号の代表がハフマン符号やシャノン・ファノ符号
      - 平均最適符号と競合最適符号

        平均符号長を最小にするのが平均最適符号

        すべての符号長が他の符号の長さより小さいのが競合最適符号
      - アルファベット順符号

        入力情報源の辞書順と出力符号化の辞書順が一致する符号

        符号語のままソートできる特徴がある．

6.2 データ圧縮の種別(２)

- **ユニバーサル符号化**
  - 確率構造は未知であるが，出力データ長が十分長くなれば確率構造が既知とほぼ同等にエントロピーレートまで圧縮が可能となるような符号
  - 実現法としては，下記の2種類がある
    - 確率分布のユニバーサル推定＋算術符号
    - 確率分布を陽に用いない符号化アルゴリズム

      LZ77符号，LZ78符号等Lempel-Ziv符号はこの仲間

6.2 データ圧縮の種別(３)

■ 静的符号化と動的符号化

- **静的符号化**
  - 情報源の確率分布が不変の場合，符号化の木を固定して使用する符号化
    - **ハフマン符号**や**算術符号**はこれに属する.
- **動的符号化**
  - 情報源の確率分布が不明か可変の場合，系列の頻度分布を調べながら動的に符号木を変化させる符号化
    - **動的ハフマン符号**や**ユニバーサル符号**はこれに属する.

6.2 データ圧縮の種別：分類表

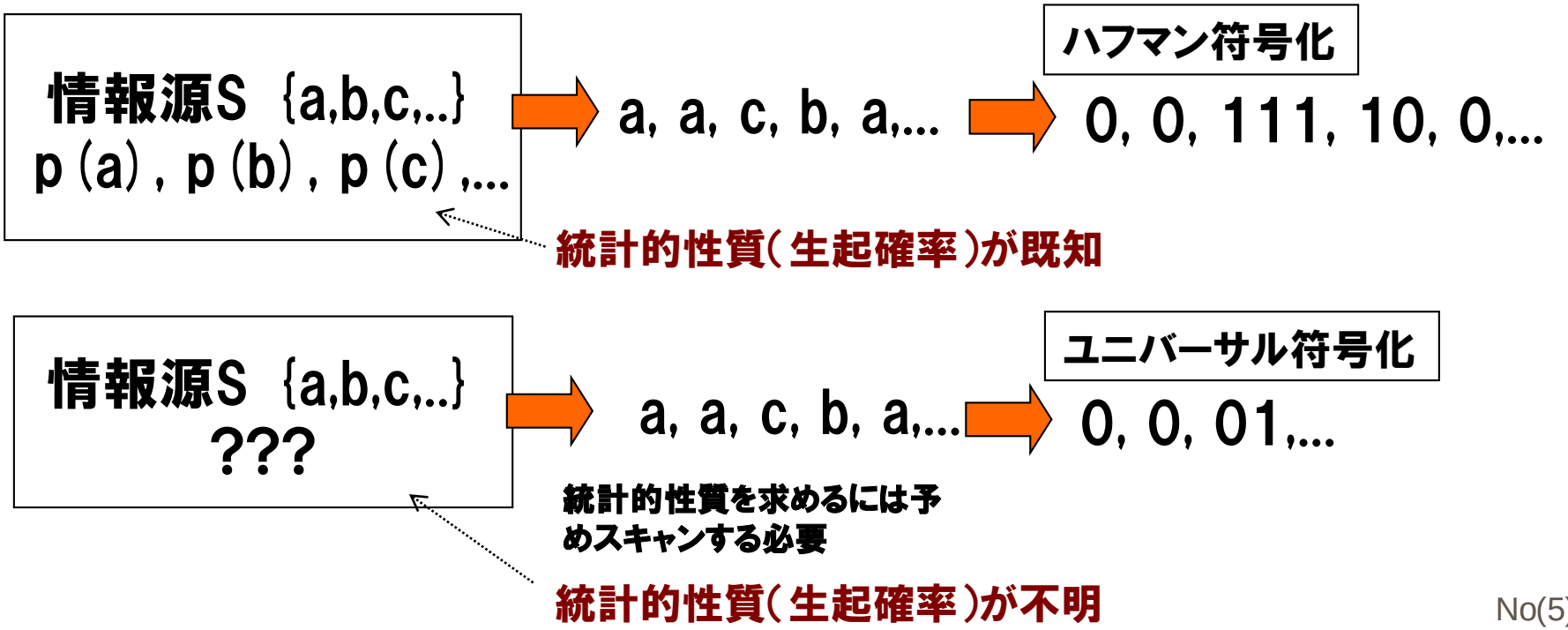
第1階層	第2階層	具体的例	
可逆圧縮	静的符号化	シャノン・ファノ符号	
		ハフマン符号	
		算術符号	
	動的符号化	ランレングス符号	
		動的ハフマン符号	
		ユニバーサル符号	LZ符号
			BSTW符号
			インターバル符号
非可逆圧縮	画像符号化	JPEG、MPEG	
	音声符号化	MP3、ATRAC	

6.3 ユニバーサル符号化の原理

- 情報源から出るシンボル列に含まれる繰り返しパターンを求め、最適な符号を割り当てる

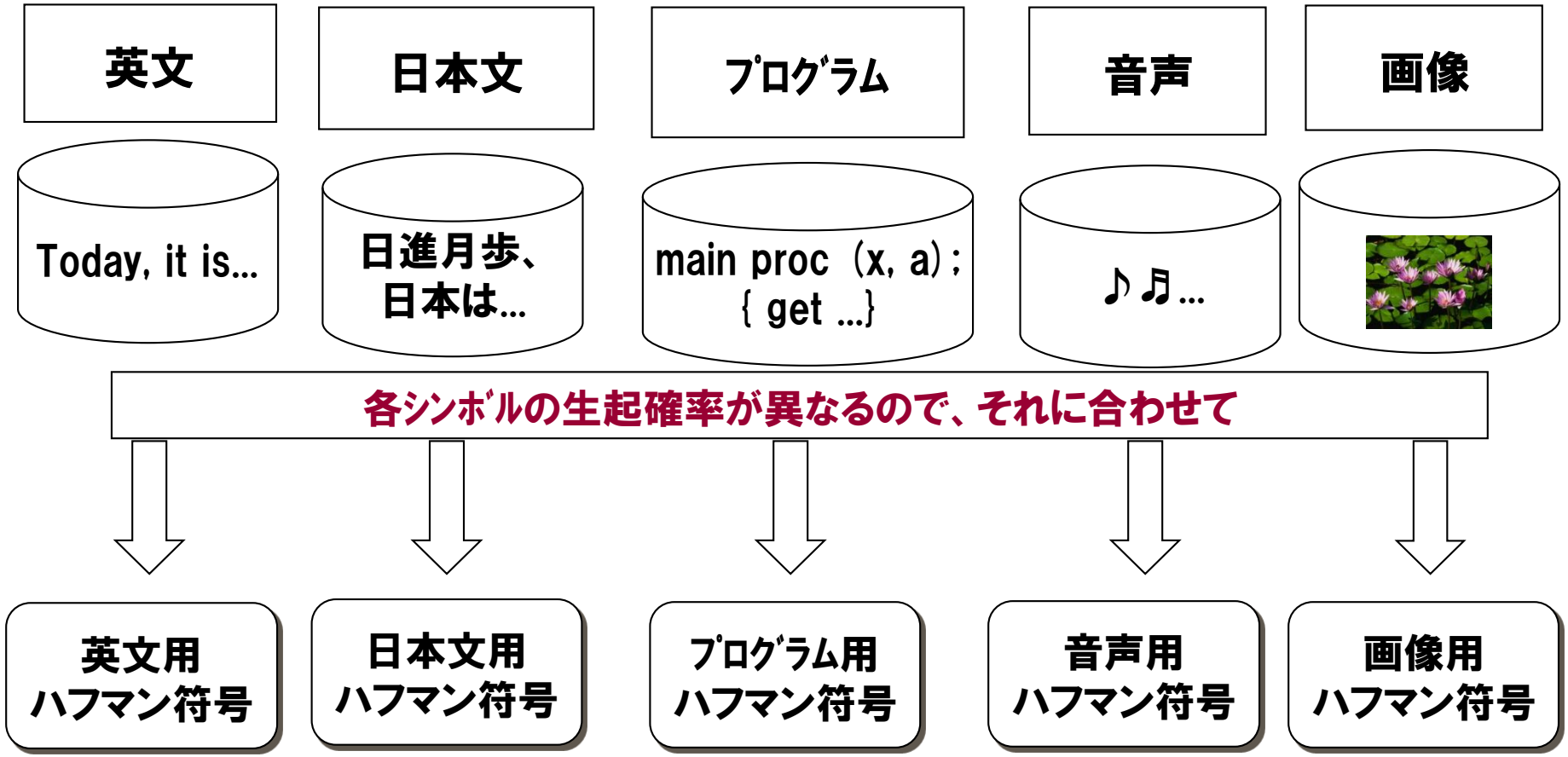
  なぜならば、

  - 情報源のシンボルの統計的性質（生起確率）を予め予想できない
  - 統計的性質を予想するために、まず系列全体をスキャンし、その後符号を割り当てる方法（２パスアルゴリズムという）をとると、符号化スピードが遅くなりリアルタイム処理には適さない



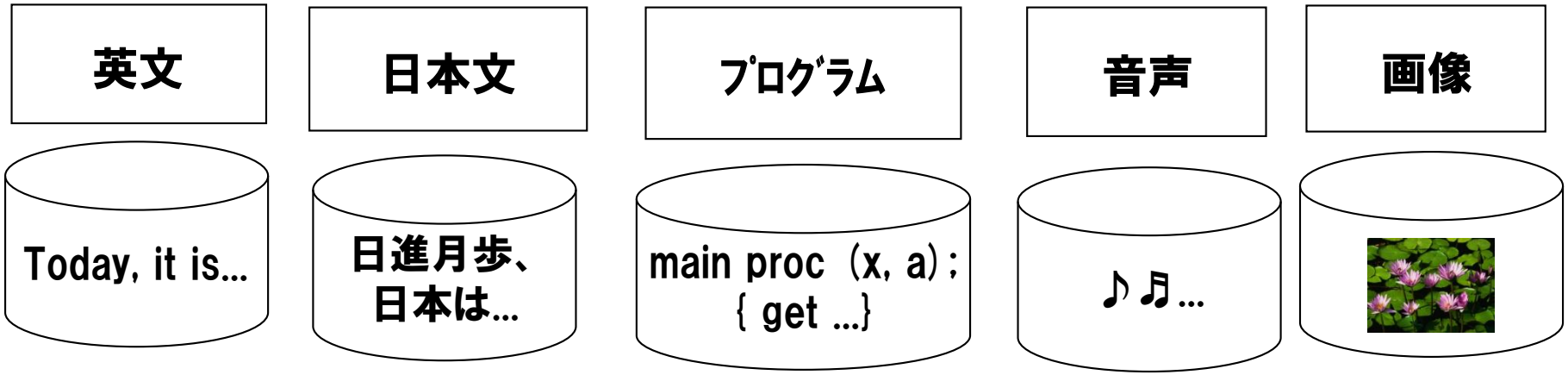
6.3 ユニバーサル符号の特徴：従来のハフマン符号

- 情報源のシンボル生起確率に合わせて符号を割り当てる
  - 情報源データの種類分だけ符号化アルゴリズムが必要となる



6.3 ユニバーサル符号の特徴：ユニバーサル符号

- 情報源のシンボルの繰り返しパターンを調べ符号化する
  - 情報源の種類によらず全て同じ符号化アルゴリズムが適用できる



各ｼﾝﾎﾞﾙの生起確率を求めるのでなく、繰り返しパターンを求める

共通の圧縮符号化：
すなわち、ユニバーサル符号

6.3 ユニバーサル符号化の考え方

- 繰り返しパターンによる圧縮の例（１）
  - 「かえるぴょこぴょこみぴょこぴょこあわせてぴょこぴょこむぴょこぴょこ」（３３文字）
    - １文字毎：か（1/33）、え（1/33）、る（1/33）、ぴ（8/33）、ょ（8/33）、こ（8/33）、み（1/33）、あ（1/33）、わ（1/33）、せ（1/33）、て（1/33）、む（1/33）
    - ハフマン符号で符号化すると122ビット。
    - 語句の辞書：かえる（0）、ぴょこぴょこ（1）、み（２）、あわせて（３）、む（4）
      - 語句ごとに符号化：０ １ ２ １ ３ １ ４ １
    - 符号長は8文字＝64ビット（1文字8ビットとして）

6.3 ユニバーサル符号化の考え方

- 繰り返しパターンによる圧縮の例（２）
  - a big black bat bit a big black bear（スペース込み36文字）
  - a big black が繰り返し現れる
    - これをPと置けば、P bat bit P bear、と短縮できる
    - 11文字のPを１つのパターンとすると、36文字から16文字に短縮できる

- 上記２つの例から、繰り返しパターンを適当に選べば大幅に符号長が短くできることが分かる。
  - 繰り返しパターンは文字列に限らない。同じ繰り返しであれば意味のないパターンでもOK。

6.4 代表的なユニバーサル符号:LZ符号

- Lempel-Ziv符号(または　Ziv-Lempel符号)
  - 繰り返しパターンを用いた代表的圧縮符号化法。1977年、Lempel(レンペル)とZiv(チフ)によって発表されたアルゴリズムでLZ符号と略
  - その後の拡張・改良によりいくつかの変形がある

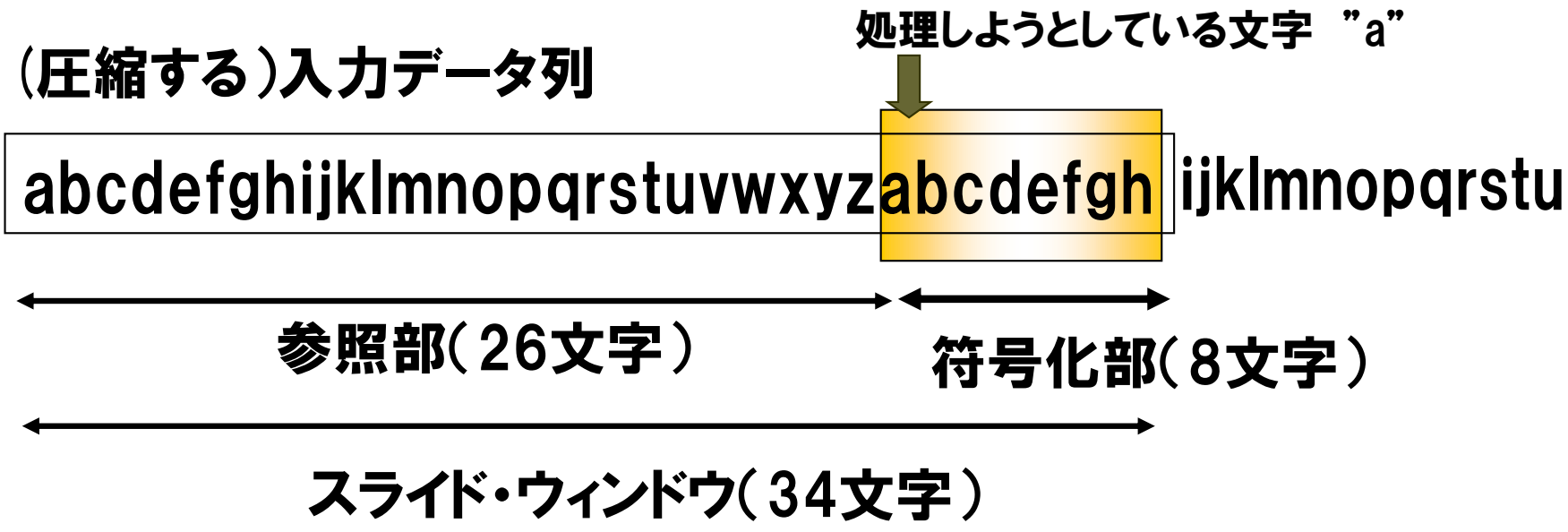
名称	説明
LZ77	1977年に発表された最初のアルゴリズム
LZ78	1978年に発表された上記の改良版。辞書に改良を加えた。
LZSS	1982年にStorer(ストーラー)、Szymanski(シマンスキー)により発表されたLZ77の改良版。冗長性を排除して圧縮効率を上げた。
LZW	1984年にWelch(ウェルチ)により発表されたLZ78の改良版。辞書の扱いをさらに改善した。
LZH	LZSSにハフマン符号を組み合わせたアルゴリズム。1988年に吉崎栄泰(よしざきはるやす)氏が発表した。LZHUFとも呼ばれ、"H"はharuyasuの頭文字。
Deflate	LZHと同様にハフマン符号と組み合わせる方式。圧縮しながら逐次ハフマン木を再構築する動的ハフマン符号で符号化する。LHZは先にデータをすべて調査したうえでハフマン木を構築し圧縮する静的ハフマン符号。RFC1951 (1996年)。

6.4 Lempel-Ziv（LZ）符号のアルゴリズム

- データに含まれるパターンをコード化して、データを圧縮する。パターンを見つけ出す方法は、スライド辞書法と動的辞書法の2種類
  - スライド辞書法を用いる「LZ77符号」
    - 特定のバッファーにデータを読み込みながら、繰り返しパターン（文字列）を見つけ出す。そのパターンを相対位置と文字数でコード化し、圧縮していく。
  - 動的辞書法を用いる「LZ78符号」
    - データを先頭から走査し、初出の文字列を辞書に登録していく。登録した文字列が見つかったら、それを辞書に参照させることにする。この作業の繰り返しにより、参照を増やして圧縮を行う。

6.4.1 スライド辞書、スライドウィンドウの概念（1/2）

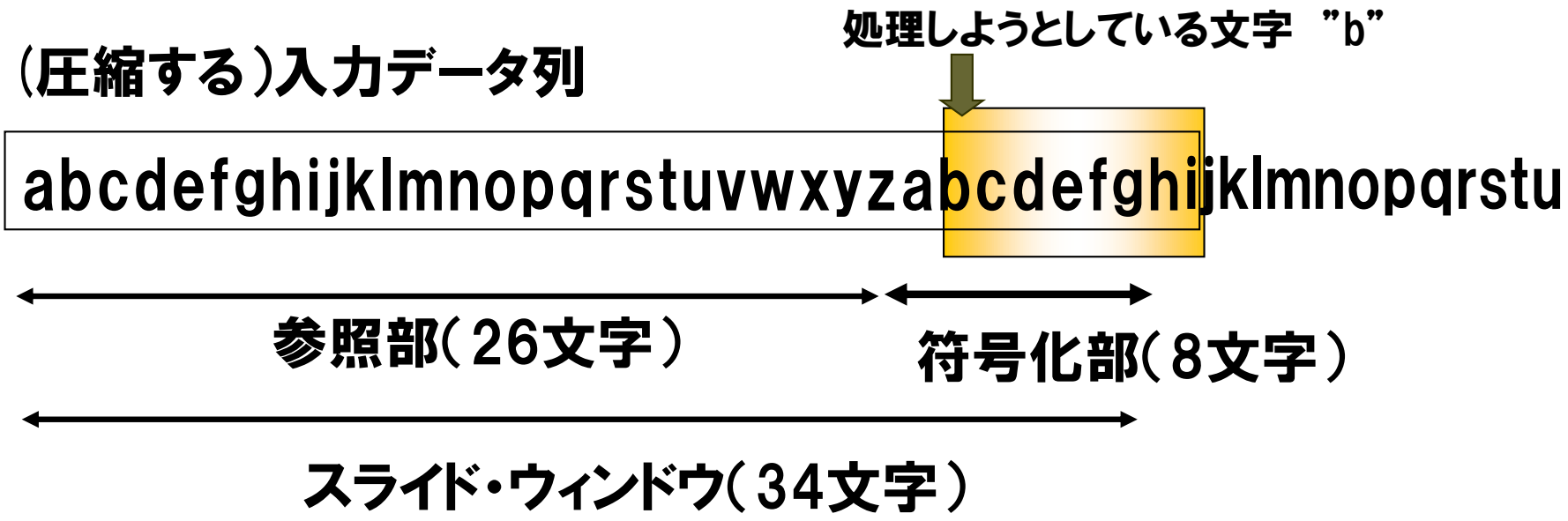
- 入力データ列を少しずつ読み込み処理する。この読み込まれるデータを「ずらし窓（スライドウィンドウ）」という。



- 参照部はすでに圧縮処理が終わった部分
- 符号化部はこれから処理をする部分
- 文字”a”の処理が終わると、ウィンドウを右に1つずらす

6.4.1 スライド辞書、スライドウィンドウの概念 (2/2)

- 文字”b”の処理に移る



- 参照部はすでに圧縮処理が終わった部分
- 符号化部はこれから処理をする部分

6.4.2 LZ77アルゴリズムの説明（1）

- 「LZ77 法」では、前に出現したデータ列から最も一致した部分列を検索して、その場所を指すことによって圧縮を行う。例えば、
- 01234012301234012301234 0123
- のデータ列の場合、
- 6 文字め以降の "0123" は先頭のデータ列 "0123" と一致するため、一致するデータ列の「何文字前から」で表わして［5,4］と置換する。
- さらに続く "01234012301234 0123" は、［9,18］と置換する。
- 展開によって新たに追加されたデータ列が一致するならば、展開を開始した位置を超えてさらに展開する。

6.4.2 LZ77アルゴリズムの説明 (2)

- 展開によって新たに追加されたデータ列が一致するならば、展開を開始した位置を超えて更に展開。
- 012340123012340123012340123
- +--------------------+
- +--------------------+
- 9文字前から18文字分が一致
- 前にあるデータを全てチェックしていたのではデータ量が多くなると処理時間も劇的に長くなるので、LZ77 法では**前にあるデータの中から最も近い固定長の部分データ**を使って比較処理を行う。
- この固定長の部分データは「Sliding Window」と呼ばれ、処理が進むごとにシフトしていく。具体的な処理の流れは以下のようになる。

6.4.2 LZ77アルゴリズムの説明(3)

1. Sliding Window の末尾の位置（swEnd）を、データ列先頭（dataStart）の一つ前にする（ つまり Sliding Window は空の状態 ）。
2. dataStart から始まるデータ列と一致する部分データ列を Sliding Window から探し、その中で最も長いデータ列を抽出する。
3. もしデータ列が見つからなければ、データそのものを出力して、swEnd を一つ後ろにシフトする。
4. データ列が見つかった場合、その位置と長さを出力してから、一致したデータ列の長さ分だけ swEnd をシフトする。

6.4.2 LZ77アルゴリズムの説明 (4)

	Sliding Window	入力データ列	出力データ列
1	**********	0123401230123401230123401230	
2	**********0	1234012301234012301234012301	0
3	**********01	2340123012340123012340123012	01
4	**********012	3401230123401230123401230123	012
5	**********0123	4012301234012301234012301234	0123
6	**********01234	**0123**012340123012340123012	01234
7	*******012340123	**01234012**3012340123012340	01234[5,4]
8	2340123012340123012		0123[5,4][9,18]

6.4.2 LZ77アルゴリズムの説明 (5)

圧縮したデータを展開する場合、以下のように処理を行う。

1. データそのものならば、そのまま出力する。
2. 圧縮されたデータならば、1 番目のパラメータで前に出力されたデータ列内の出力する位置を決めて、2 番目のパラメータの数だけ順次コピーしていく。

6.4.2 LZ77アルゴリズムの説明 (6)

	入力データ列	出力データ列
1	01234[5,4][9,18]	
2	[5,4][9,18]	01234
3	[9,18]	012340123*********
4	[9,18]	01234012301234012
5		012340123012340123012340123

6.4.3 LZ77符号のアルゴリズム：例示（1/5）

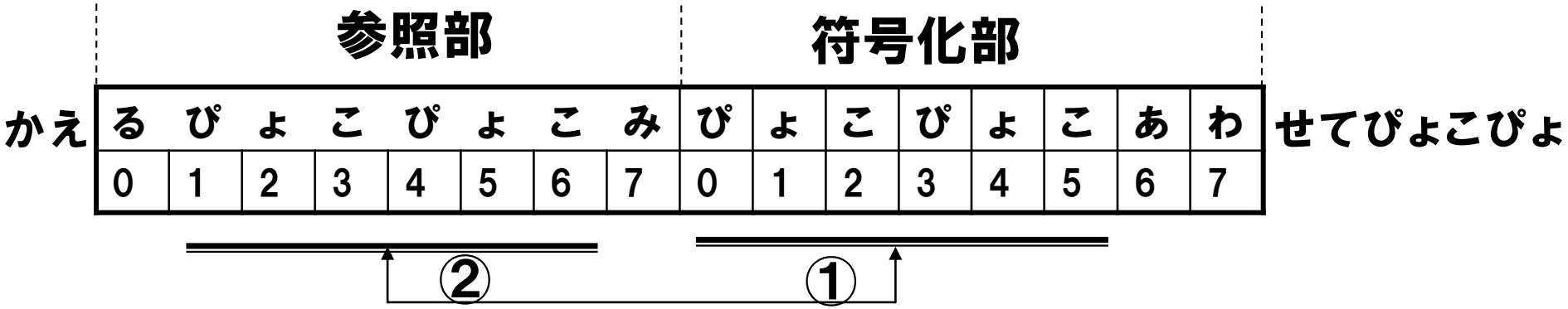
例題：下記の早口言葉文を圧縮する

- **対象データ＝「かえるぴょこぴょこみぴょこぴょこあわせてぴょこぴょこむぴょこぴょこ」**
- **スライドウィンドウ長=16文字、参照部＝8文字、符号化部＝8文字**
- **文字位置を示す番号を振る：0~7が位置（オフセット値）**

（注）以降のページの例示は、杉本氏のWebページから一部転用しています。
http://www.snap-tck.com/room03/c02/comp/comp041.html

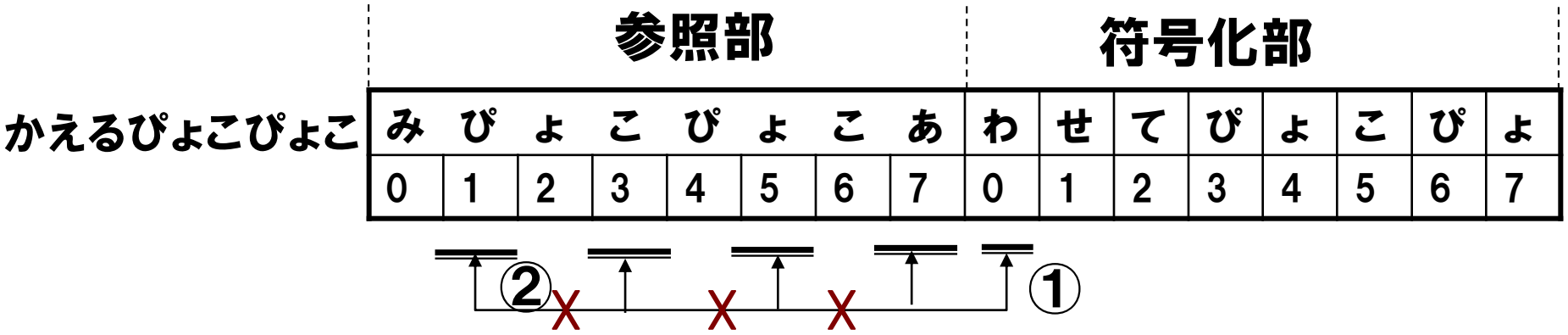
6.4.3 LZ77符号のアルゴリズム：例示（2/5）

- 対象データの2番目の「ぴょこぴょこ」を符号化する場合、符号化する部分が符号化部の先頭になるように、バッファにコピーする
- 符号化部の①の部分（6文字）が参照部の「2番目から6文字分」と一致する（＝最長一致系列）
  - 最長一致系列：符号化部の先頭から始まる文字列と一致する参照部の文字列の中で最も長いもの
- 符号化した系列は、次の3つの組として表す
  - 最長一致系列の参照部先頭からのオフセット値
  - 最長一致系列の記号数（長さ）
  - 一致しない最初の記号
- 下記の例では、オフセット値＝1，一致系列長＝6，不一致記号＝あ，となるので、（1，6，あ）と符号化する



6.4.3 LZ77符号のアルゴリズム：例示（3/5）

- バッファの文字を符号化した文字数分（本例では7文字分）左に移動する
  - 16文字分の窓が対象データを7文字分左にスライドする
- 符号化部の先頭「わ」から始まる系列は、参照部のどれとも一致しない
  - オフセット値＝0，一致系列長＝0，不一致記号＝わ，となるので、（0，0，わ）と符号化する
- 1文字だけ左に移動する



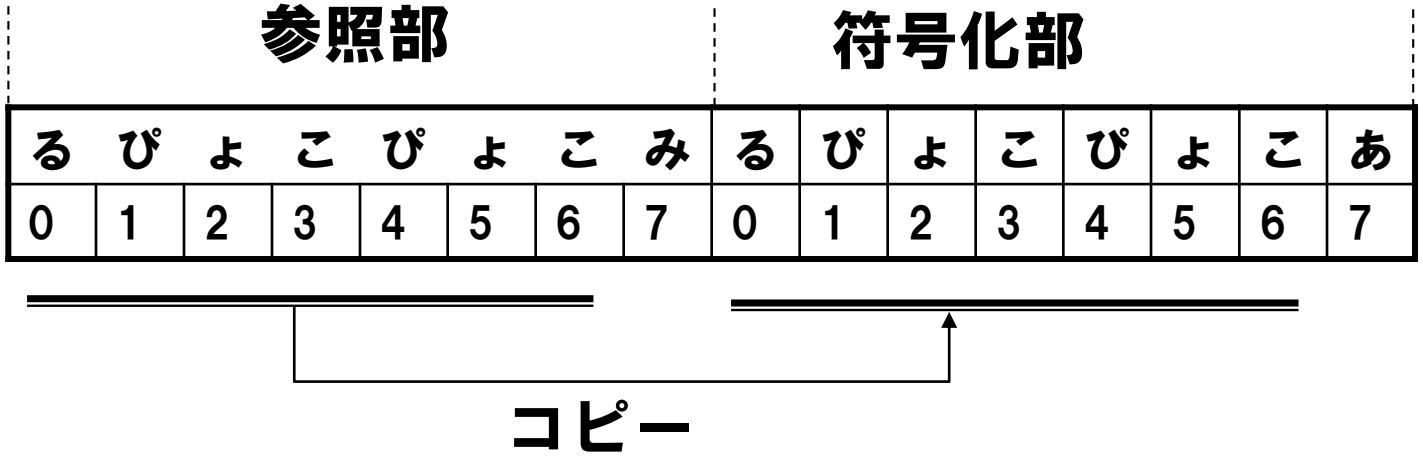
6.4.3 LZ77符号のアルゴリズム：例示（4/5）

- 同様な手順で符号化を繰り返す

step	参照部（8文字）	符号化部（8文字）	出力符号（圧縮）
1		かえるぴょこぴょ	（0,0,か）=（か）
2	か	えるぴょこぴょこ	（0,0,え）
3	かえ	るぴょこぴょこみ	（0,0,る）
4	かえる	ぴょこぴょこみぴ	（0,0,ぴ）
5	かえるぴ	ょこぴょこみぴょ	（0,0,ょ）
6	かえるぴょ	こぴょこみぴょこ	（0,0,こ）
7	かえるぴょこ	ぴょこみぴょこぴ	（5,3,み）
8	るぴょこぴょこみ	ぴょこぴょこあわ	（1,6,あ）
9	みぴょこぴょこあ	わせてぴょこぴょ	（0,0,わ）
10	ぴょこぴょこあわ	せてぴょこぴょこ	（0,0,せ）
11	ょこぴょこあわせ	てぴょこぴょこむ	（0,0,て）

6.4.3 LZ77符号のアルゴリズム：例示（5/5）

- 復号化
  - 符号化と同じスライド窓を使う
- 復号化した文字列の最後の8文字分を参照部にコピーし、符号化部を空にする
- 復号化するデータ＝（1, 6,あ）
  - 参照部オフセット＝1、一致文字列＝6文字、その後に続く文字＝あ
  - 参照部の「ぴょこぴょこ」を符号化部の先頭にコピーし、そのあとに「あ」をコピーする



6.5 有歪み圧縮(非可逆圧縮)

- 音声や画像、動画の圧縮法について、代表的な方式を概説する

(注)本稿の部分は、池田・小暮・安田による、標準化教育プログラム「第17章　画像・映像圧縮(JPEG/MPEG)」を一部参照・利用しています

6.5.1 有歪み圧縮が必要な理由

- 映像の情報量は膨大
  - 映画は24コマ/秒、TVは30コマ/秒の静止画を連続的に送り再生している。
  - 画素のピクセル数を、横720、縦480とすると、1画素を16ビットで表現するとして、
  - 16 x 720 x 480 x 30=160Mb/secの情報量(必要なデータ転送速度)となる
  - 一方、CDに収まるデータ量は、音声2ch、1サンプル16ビット、を44.1KHzで標本化、74分収録とすると、2 x 16 x 44,100 x 60 x 74=6,250Mbitとなる。

  ⇒CD1枚に、6250/160=37秒しか入らない
- 従って、大幅なデータ圧縮が必要(エントロピー符号化だけでは無理)

6.5.2 代表的な音声の圧縮方式

- MP3 (MPEG-1 Audio Layer 3)
  - ブロック符号化
  - 32サブバンドへ周波数分割
  - サブバンド毎MDCT(Modified Discrete Cosine Transform)
- MPEG-2 AAC (Advanced Audio Coding)
  - 適応ブロック長MDCT
  - 4次元ハフマン符号化

6.5.3 代表的な画像、映像圧縮方式

- JPEG（Joint Photographic Expert Group）
  - 静止画の圧縮技術
    - 動画対応では静止画をつなぎ合わせたMotion JPEGがある
  - DCT（Discrete Cosine Transform, 離散コサイン変換）
  - DCT係数の可変長ハフマン符号化
- JPEG2000
  - DCTを使わず、ウェーブレット(Wavelet)変換を用いる

6.5.4 DCT（離散コサイン変換）1/4

（IT用語辞典 e-Wordsより）

- 離散関数を各周波数成分の大きさへと変換する関数変換のひとつ。時系列に並んだ信号成分の列を周波数成分の列に変換する信号変換の一種。
- 画像や音声などをサンプリングして離散的な信号に変換し、DCTを行なった後に符号化を行うことで、元の信号の大部分を損ねずにデータの容量を減らすことができる。
- JPEGなどの画像圧縮技術や、AAC、MP3などの音声圧縮技術において利用されている。

6.5.4 DCT（離散コサイン変換） 2/4

- DCTは離散フーリエ変換に工夫を加えたもので、変換後の信号の周波数成分が低周波数領域に集中するのが特徴。
- データ圧縮にDCTを利用する場合は、変換後の信号を量子化し符号化するが、この際に情報の集中していない領域に対して少ない符号化ビットを割り当てるか、または0で近似し切り捨てることで、データの容量を減らすことができる。
- DCTには主な手法が8つあり、そのうちの4つが一般的に使われる。音声圧縮の場合は対象領域の境界でノイズが生じないよう、領域を重複させながら半分ずつずらして計算していく修正離散コサイン変換（MDCT：Modified DCT）がよく用いられる。DCTの逆変換を逆離散コサイン変換（iDCT：inverse DCT）という。

6.5.4 DCT（離散コサイン変換） 3/4

水平方向 (i)

149	120	79	49
120	94	58	32
78	58	29	9
49	33	9	0

垂直方向 (j)

DCT

IDCT

水平方向空間周波数 (h)

120	59	1	-1
59	15	-2	1
2	-2	1	-1
-1	0	-1	0

垂直方向空間周波数 (k)

オリジナルデータf(i,j)　　**変換後データF(h,k)**

- MPEGにおけるDCT変換は、8x8のピクセル単位で行う
- 変換後データF (k,k) は、オリジナルデータf (i,j) のcos変換で与えられる
  ⇒次ページ式
- 相関性の高い画像の場合、変換後の周波数成分の係数は大きく偏る。
  高周波成分は0に近い値をとる。

6.5.4 DCT（離散コサイン変換） 4/4

$$F(h,k)=\frac{4C_hC_k}{N^2}\sum_{i,j}f(i,j)\cos\frac{(2i+1)h\pi}{2N}\cos\frac{(2j+1)k\pi}{2N}$$

$$C_h,C_k=\frac{1}{\sqrt{2}},(h,k=0)$$

$$=1,(\text{その他})$$

- 変換のみによる符号量は変わらないが、空間周波数が低周波に偏りがあることを利用して量子化することで、符号量を削減する

情報源符号化のまとめ

これまでの結果とこれからの内容

- **これまでのまとめ**
  - **情報源に着目**し、情報源の性質と符号化方法を考えた
  - **情報源符号化定理（シャノン）**で、情報源のもつエントロピーに等しい長さの符号で符号化できることが示された
  - この符号化は、**冗長度をできるだけ小さくした**もの（＝理想的符号化）
- **これから**
  - 情報源から**通信路に着目点を移す**
  - 誤りの起きる通信路では、**冗長性をもたせた符号化**が必要であり、この符号化法を調べる

情報源符号化で扱わなかった問題(１)

- 歪みを許さない情報源符号化(**可逆符号化**)
  - これまでの情報源符号化は、符号が復号化すると元の情報源シンボルに誤りなく戻せる、というものであった。これを**可逆符号化(=Lossless符号化)**という。
    - 可逆符号化は、平均符号長がエントロピーより小さくできない制限がある.
- 歪みを許す情報源符号化(**非可逆符号化**)
  - 場合によっては、ある程度の誤り(ひずみ:歪み)があっても、平均符号長を短くするほうが望まれる。
  - 歪みを許す情報源符号化を、**情報圧縮(データ圧縮)**あるいは**非可逆符号化(=Lossy符号化)**、と呼ぶ。
    - 歪みを許す情報源符号化は、歪みを「雑音(誤り)」ととらえて、雑音を発生する通信路における符号化、と同様な議論ができる。

情報源符号化で扱わなかった問題(２)

- 確率分布(モデル)が不明な情報源の符号化(**ユニバーサル符号化**)
  - これまでの符号化は、情報源の確率モデルが既知と仮定していた。

    すなわち、情報源シンボルが($s_1, s_2, .. s_n$) 分かっており(有限で)、かつ各シンボル生起確率が与えられている、と仮定した。しかし、このような確率構造が分かっている情報源は限られており、さらに時間的に変動することもあるのが現実である。
  - 従って、情報源の確率パラメータを仮定しないで符号化・復号化する方法を考える理論が考えられる。
  - この方向に沿って研究され、データ長が十分大きい状況では、確率パラメータが既知である場合と同等な最適な符号化が可能であることがわかった。これを、**ユニバーサル(Universal)符号**、という。