コンピュータ・サイエンス1

第12回
アナログ情報とデジタル情報(2),
実習(標本化・量子化)

人間科学科コミュニケーション専攻
白銀 純子

第12回の内容

- アナログ情報とデジタル情報(2)
- 実習(標本化・量子化)

設問1

- 下記の中でASCII文字の番号を選択しなさい。
  1. 10001010
  2. 11010100
  3. 01111001
  4. 11111101
  5. 00011101
  6. 11001100

解答:3, 5

設問2

- やってみよう! の演習用文字コードを使った問題の1.の問題の解答を報告すること
  - 問題1: 62B7 2654 4C25 1D7F 720B

解答:さいえんす

設問3

- 下記の中で正しい説明を全て選びなさい
  1. Unicodeを使ったとしても、機種依存文字はありえる
     - Unicodeでも扱わない文字はあるので、機種依存文字はありえる
- Unicodeでは、日本語の文字とギリシア語の文字で、同じ番号を使っていることがある
  - Unicodeは様々な言語に違う番号を割り振るので、同じ番号は使っていない
- UTFで番号を割り振られている文字の中には、UCSの規定に含まれていない文字も存在する
  - UCSで扱う文字を決め、UTFでUCSの文字に番号を割り振るので、存在しない
- Unicodeを使うと、1つの文書の中で日本語やフランス語、アラビア語など様々な言語を使うことができる
  - Unicodeは様々な言語を扱うために作られた規格なので、使うことができる

解答:1, 4

前回の質問の回答

文字コードの見分け方[1]

- ISO-2022-JP
  - エスケープシーケンス(ASCIIで「ESC」の文字を使う)あり
  - 前半8ビットが$(20)_{16}$〜$(7E)_{16}$、$(21)_{16}$〜$(7E)_{16}$の範囲
  - 後半8ビットが$(21)_{16}$〜$(7E)_{16}$の範囲
- Shift JIS
  - エスケープシーケンスなし
  - 前半8ビットが$(81)_{16}$〜$(9F)_{16}$、$(E0)_{16}$〜$(FC)_{16}$の範囲
  - 後半8ビットが$(40)_{16}$〜$(7E)_{16}$、$(80)_{16}$〜$(FC)_{16}$の範囲

文字コードの見分け方[2]

- EUC-JP
  - エスケープシーケンスなし
  - 前半8ビットが$(20)_{16}$〜$(7E)_{16}$、$(8E)_{16}$、$(A1)_{16}$〜$(FE)_{16}$の範囲
  - 後半8ビットが$(A1)_{16}$〜$(FE)_{16}$の範囲

- 前半・後半の番号のパターンを分析して、文字コードを判定
- 判定に失敗することもあり、そうすると文字化け
  - ✓ コンピュータは文書内の言葉は理解していないので、番号の特徴だけで判断

Question!

前回の復習

モード切り替えの問題(p. 17)

- 文書を先頭から順番に見ていく場合には問題ない
- 文書を途中から見ていくときに問題が生じる
  - 見始めた途中の文字が、ASCII文字か日本語文字か、エスケープシーケンスかが判別できない

Ex. 見始めた途中の文字が「70」番だった場合
- ASCII文字の「F」?
- 日本語文字の一部?
- 韓国語の一部?

検索や置換などの文書処理に時間がかかる

EUCとSJIS[1](p. 17)

- ASCII文字: 8個の0と1で、1文字分を表現
  - 実際には、7個の0と1で1文字分を表現
  - 8ビット目は必ず0

必ず0 XXXXXXXX

0000000から1111111まで、フルに使ってASCII文字を1文字ずつ表現

= ASCII文字は、0XXXXXXX という形
Ex.「a」を0と1で表現すると: 01100001

8ビット目が1になる番号(1XXXXXXXという形の番号)はASCII文字ではない

EUCとSJIS[2](p. 17)

- ASCIIで使われていない番号を2バイト文字の番号にあてる方法
  - EUC(日本語のものをEUC-JP)
    - 第1バイト(前半の8ビット)と第2バイト(後半の8ビット)両方でASCII領域が避けられている
  - SJIS(Shift JIS)
    - 第2バイト(後半の8ビット)ではASCII領域も使われている
  - 文章のバイト数は、単純に、日本語文字で2バイト、ASCII文字で1バイトで数えれば良い

例えばある文章が...

00110110100101101010010101101010

1から始まっているから日本語文字

0から始まっているからASCII文字

言語圏ごとの文字コード(p. 18)

- これまでの多バイト文字の扱い: 異なる言語圏ごとに文字集合を作成
  - 様々な文字集合ができてしまって不便
    - コンピュータネットワークの国際化が進んだ
    - コンピュータの資源が豊富になった

国際文字集合規格として各文字集合を統一化

- ASCII, ラテン文字, 日本語, 韓国語, 中国語, ベトナム語, ギリシャ文字, 記号, etc.

Unicode: どの文字を扱うかと文字の符号化の方法を決めた規格

- UCS(Universal multi-octet coded Character Set)でどの文字を扱うかを規定
- UTF(UCS Transformation Format)で文字の符号化の方法を規定

UTF-8(p. 18)

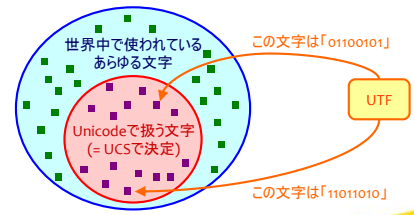
- Unicodeでの代表的な符号化方式(符号化方式はいくつか存在)
- 1文字を1～6バイトの可変長(文字によってバイト数が異なる)で符号化する方式
  - ASCIIやISO-2022-JP、Shift JIS、EUC-JPは1文字を同じバイト数で表現
- OS(WindowsやMacなどのオペレーティングシステム)でファイル名などの内部処理に利用
  - 半角英数を符号化した結果が、ASCII文字と全く同じになるため、従来のシステムと相性が良い

現在、UTF-8への移行が急速に進んでいる

- ただし、以前からのファイルを移行するのは大変なので、完全移行には時間がかかる
- 完全移行できたら、文字化けが起こらなくなる

UCSとUTF

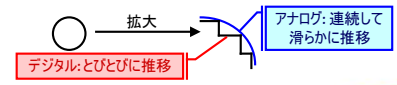
- UCS: 世界中のあらゆる文字のうち、どの文字をUnicodeで扱うかを決めた規格
- UTF: UCSで決めた文字に対し、どのように0と1を割り当てるかを決めた規格



アナログ情報とデジタル情報

アナログ情報とデジタル情報(p. 19)

- アナログ情報: 連続的な数値で表現できる情報
  - 物事を表現する数値の桁数が無限
  - Ex. アナログ時計: 1秒から2秒になるまで秒針が止まらず動き続ける(1秒と2秒の間も1.xxxx....秒が存在する)
- デジタル情報: 離散的な数値(とびとびの数値)で表現される情報
  - 物事を表現する数値の桁数が有限
  - Ex. デジタル時計: 1秒の次は2秒(1秒と2秒の間がない)



アナログ情報のデジタル化(p. 20)

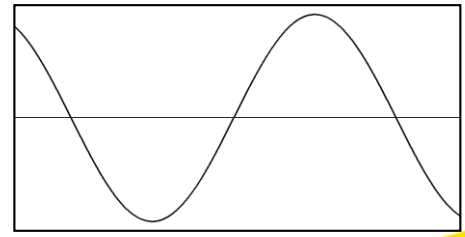
- アナログ情報: 数値化すると連続的
  - 音の強弱の変化
  - 画像の色の濃さ光の強弱の変化

グラフで表すと、なめらかな曲線(正弦波) = アナログ信号

デジタル化するには…
- 標本化
- 量子化

正弦波

- 波の形になっているグラフ
  - いくつかの特徴あり



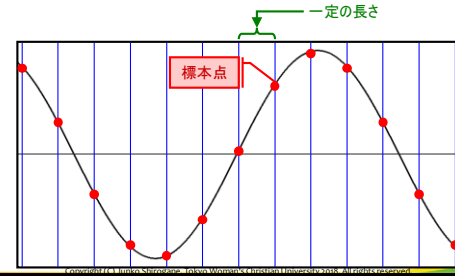
標本化[1](p. 20)

- アナログ信号: 様々な幅や高さの波(正弦波)の組み合わせ
  - 幅の狭い波・広い波、高い波・低い波
- 標本化: アナログ信号の横軸の一定の長さごとに、縦軸の値を調べること
  - 調べた点を「標本点」と呼ぶ
- 調べた以外のところの値は使用しない
  - 標本点の間隔を十分に細かくとれば、もとのアナログ信号の情報を完全に保持できる

どのくらいの間隔で標本化をすれば良いか???

標本化[2](p. 20)

- 標本化: 横軸を一定の長さで区切って縦軸の値を調べること



標本化[3](p. 20)

- 標本化の感覚をどうやって決めるか?
  - 1つの波の幅の半分より小さい間隔で標本点をとれば、元の波形を正確に表現可能(=方程式を算出可能)
  - 1つの波の幅の半分より大きい間隔で標本点をとれば、元の波形とは異なった波形が出現

エイリアシング

音や画像の情報: 波形の周期の長いもの・短いものを重ね合わせることで表現

波の中で最も狭い幅のものの半分よりも狭い間隔で標本化することが重要

量子化[1](p. 22)

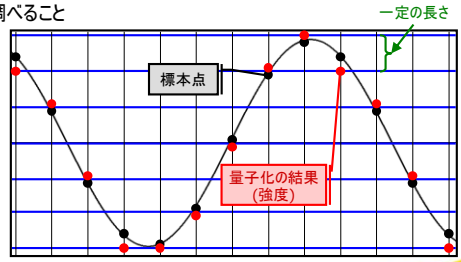
- 量子化: 縦軸の一定の長さごとに、横軸の値を調べること
  1. 横軸の値を調べるための縦軸の量(量子化レベル)をどの程度にするかを決める
  2. 標本化で取り出された標本点のy軸の値(強度)を最も近い量子レベルに置き換える

アナログ情報のデジタル化が完了

- 最終的に、量子化の結果(強度)の値をコンピュータに取り込む
  - もとのアナログの値とは異なる値が取り込まれる

量子化[2](p. 22)

- 量子化: 縦軸を一定の長さで区切り、各標本点と最も近い縦軸の値を調べること



画像の符号化[1](p. 23)

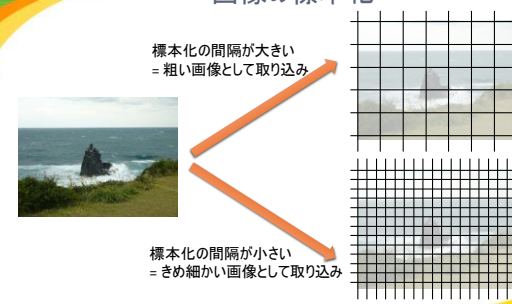
- 標本化した画像: 点の集まりと考えられる
  - 点: 細かい正方形のマス目
    - 画素(ピクセル, pixel)
  - 画像の長方形のキャンバスは点の集まり
  - 1つ1つの点の大きさによって画像の質が決定
    - 点が大きければ粗い画像
    - 点が小さければきめの細かい画像
    - ⇒ 標本化の間隔により、点の大きさが決定
  - 1つ1つの点が何色かを記録しておくことで画像を表現
    - ⇒ 量子化の間隔により、画像中で利用可能な色の種類が決定 (どの程度、微妙な色合いを表現するか)

画像の符号化[2](p. 23)

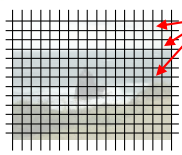
- カラー画像
  - コンピュータのディスプレイ: 赤(Red), 緑(Green), 青(Blue)の3つの光を利用
    - 3つの光にそれぞれ256段階の濃淡をつけ、3つの光を混ぜ合わせて色を作成
    - 256段階 = 8ビットで表現可能
    - 1つの色: 8ビット×3つの光 = 24ビットで表現
      - 画像中の1つの点を24ビットで表現

⇒ カラー画像: 1つの画素を0～16,777,215までの数値で表現可能

画像の標本化



画像の量子化



- 標本化で区切った1つ1つの格子を点として取り込み
- 量子化により、各点が何色かを決定
  - ✓ 1つの点につき1色
  - ✓ 量子化の間隔により、画像全体で何種類の色が使えるかが決定

実習(準備)

スキャナって?

- 紙やフィルムなどに印刷された(書かれた)画像をデジタルデータに変換するための装置
  - コンピュータに取り込むことができる
- コピー機のようなもの
- 画像をデジタルデータに変換することを「スキャンする」と呼ぶ

dpi(解像度)(1)

- Dot Per Inchの略
- スキャナやプリンタでの解像度の単位
  - 画像のきめ細かさ
- 標本化の間隔
- 1 inch(約2.54 cm)を何個の点で表すかという単位
  - ➡ 300 dpi: 1 inchを300個の点として取り込み
- 画像をスキャンするときに、解像度を選択

dpi(解像度)を大きくすると...

- 画像の中の1つ1つの点が小さい
  - 標本化の間隔が小さい

利点

元の画像により近い状態でコンピュータに取り込むことができる(画質が良い)

欠点

ファイルサイズが大きくなる

⬅ コンピュータでは、点の大きさの大小に関わらず、点が持つ情報量は同じため、点の数が多いほどファイルサイズが大きい

dpi(解像度)を小さくすると...

- 画像の中の1つ1つの点が大きい

利点

ファイルサイズが小さくなる

欠点

画質が悪くなる

適度な解像度は?

- Webページに掲載するとき: 72〜96 dpi
  - ディスプレイの解像度がこのくらい
  - ➡ 解像度を大きくしても、ディスプレイ上で表示される大きさが大きくなるだけ
- 印刷するとき: 240〜350 dpi程度

今回授業前に取り込んだ写真

- 4種類の設定で取り込み
  - 画像1: 解像度 50, イメージタイプ 24bitカラー
  - 画像2: 解像度 300, イメージタイプ 24ビットカラー
  - 画像3: 解像度 300, イメージタイプ 8bitグレー
  - 画像4: 解像度 300, イメージタイプ モノクロ

今後、「画像1」から「画像4」という呼び方をするので、取り込んだ写真のファイルのどれがどの画像に対応するかをわかるようにしておくこと

比較(標本化)(1)

- 画像1と画像2をFinderでダブルクリックして開く
  - 「プレビュー」というアプリケーションで開かれる

どちらが画像が小さく表示されるか?

Question!

比較(標本化)(2)

- 画像1と画像2をFinderでダブルクリックして開く
  - 「プレビュー」というアプリケーションで開かれる
- プレビューの「+」(拡大)ボタンを何度か押して、2つの画像が同じくらいの大きさに見えるように調整

どちらが画質が悪いと思うか???

Question!

比較(標本化)(3)

- 画像1と画像2をFinderで右クリック→「情報を見る」
- ファイルサイズ(「サイズ」の欄)を比較

どちらのファイルサイズが小さいか???

比較(量子化)(1)

- 画像3と画像4をFinderでダブルクリックして開く
  - 「プレビュー」というアプリケーションで開かれる

どちらが多くの種類の色を使っているか???

比較(量子化)(2)

- 画像3と画像4をFinderで右クリック→「情報を見る」
- ファイルサイズ(「サイズ」の欄)を比較

どちらのファイルサイズが小さいか???

解像度の小さい画像を大きくすると...?(1)

- 解像度の小さい画像を、画像編集ソフトで大きくすると、大きい画像と同じようになるか?
  - 画像1は画像2よりも6倍小さいので、画像編集ソフトで大きくしてみよう!

解像度の小さい画像を大きくすると...?(2)

- Gimpで画像1のファイルを開く
  - Finder→「アプリケーション」→「gimp」をダブルクリック
  - 「ファイル」→「開く/インポート」をクリック
  - 表示されたウィンドウで、画像1のファイルを選択し、「開く」をクリック
- Gimpで画像1の解像度を変更する
  - 「画像」→「画像の拡大縮小」を選択
  - 表示されたウィンドウの「幅」の欄を、元の数から6倍に変更
    - 横か縦のどちらかを入力すると、もう一方は、もとの写真の大きさに応じて自動的に変わる
  - 「拡大縮小」ボタンをクリック

解像度の小さい画像を大きくすると...?(3)

- Gimpで解像度変更後のファイルを保存する
  - 「ファイル」→「エクスポート」を選択
    - 「ファイル」→「保存」ではないので注意!
  - 画像1とは違うファイル名で保存
    - 拡張子は「.tiff」にすること(画像1のファイル名が「photo.tif」なら、「photo-6times.tif」など)
    - 表示されたウィンドウはそのまま「エクスポート」ボタンをクリックでOK
- 画像2と保存したファイルを比較する
  - 画像2と保存したファイルをダブルクリックして開く

➢ 画像の品質は同じか???
➢ 違うのであれば、なぜ違うか???

データ圧縮と情報量

画像データの伝送[1](p. 25)

- 地上デジタル放送: 1440×1080の解像度でフレームレートが約30fps
  - fps(frame per second): 1秒あたりに切り替える画像の数

動画の世界では「フレーム」と呼ぶ

Ex. 1つの点を24ビット(3byte)で表現すると...
(1440×1080画素)×(3byte)×(30フレーム/秒)
=139968000byte/秒
≒140Mbyte/秒
(ただし、音声なし)

画像データの伝送[2](p. 25)

- デジタル放送: 140Mバイト/秒の伝送が必要
  - テレビ局から家庭に映像を送るときに、1秒間に140Mバイト送れないと、なめらかな映像が見られない
  - Ex. 家庭のコンピュータのネットワーク環境
    - ADSL: 50Mビット/秒(bps)= 6.25Mバイト/秒
    - 光: 100Mビット/秒(bps)= 12.5Mバイト/秒

理論上の速さで、実際はもっと遅い

↓

140Mバイト/秒は莫大な伝送量!!

画像のデジタル表現[4](p. 25)

- 140Mバイト/秒は現実的に伝送不可能
  →データを圧縮して伝送
  - 圧縮: 決まった手順に従ってデータのサイズを小さくすること
- 静止画
  - 画像の中の特定の領域の色の変化は少ない
- 動画
  - 動画の中で実際に動く部分は大きくない
  - 動く部分を過去の様子から大体予測できる

デジタル放送では、この性質を使って圧縮
(圧縮技術: MPEG-2)

圧縮と復号(p. 26)

- 数値・文字・画像・音声: 符号化(2進数に変換)して処理
- 特に画像・音声はデータ量が多い
  - 伝送に時間が多く必要
  - 保存場所の容量が多く必要

↓

- 伝送・保存時にはビット列(2進数の桁数)を短くする
  = データ量を少なくする
- 表示などの処理の際には元の形に戻す

復号 ← 符号圧縮(圧縮)

可逆圧縮(p. 26)

- 可逆圧縮
  - 復号時に1ビットの違いもなく元のビット列が復元される圧縮法
    - 圧縮したファイルから、圧縮前のファイルを取り出すことができる
  - 非可逆圧縮に比べて、ビット列をあまり短くすることはできない
    - 非可逆圧縮よりもデータ量の減量分は少ない

非可逆圧縮(p. 26)

- 非可逆圧縮
  - 復号時に元のビット列とは多少の違いが生じてしまう圧縮法
    - 圧縮したファイルから、圧縮前のファイルを取り出すことができない
  - 可逆圧縮に比べて、ビット列がかなり短くなる
    - 可逆圧縮よりもデータ量の減量分が多い

数字・文字情報(p. 26)

- 復号したときに内容が変わってはならない
  - 文書中の1文字が抜け落ちたら…?
  - 数値の小数点の桁数が少なくなったら…?
- 一般的に(画像・音声に比べて)データ量は少ない
  - 圧縮によるデータの減量分はそれほど(画像・音声ほど)多くなくて良い

↓

可逆圧縮が使われる

画像・音声情報(p. 26)

- 復号したときに内容が多少変わっても良い
  - 画像: 表示したときに見た目が変わらなければ良い
  - 音声: 聞いたときに元と同じように聞こえれば良い
- 一般的に(数値・文字に比べて)データ量が多い
  - 圧縮によるデータの減量分が(数値・文字に比べて)多いことが必要

↓

非可逆圧縮が使われる

- 動画・音声のデータ量は膨大なので、ほとんどの場合非可逆圧縮が使われる
- 静止画のデータ量はそれほど多くないので可逆圧縮・非可逆圧縮のどちらも使われる

非可逆圧縮の圧縮法

- 余分な情報を取り除くことで圧縮
  - 画像の場合、同じ色が集まっているところの色の情報など
  - 音声の場合、人間の耳には聞こえないような音

人間の感覚ではわからないものを削るので、見た目・聞いた感じでは品質は変わらない

- 画像は拡大すると、画質が落ちているのがわかる
- 音声は、良いスピーカーを使うと音質が落ちているのがわかる
- 圧縮率を上げる(ファイルサイズをより小さくする代わりに、取り除くものを多くする)と、質が落ちているのがわかる

可逆圧縮の圧縮法(p. 28)

- ハフマン符号化
- ランレングス符号化
- etc.

ハフマン符号化[1](p. 28)

- ハフマン符号化: データ内に出現する情報を統計的に処理し、ビット列の長さを変えて情報を表現

これまで: どの情報も同じ長さのビット列で表現

- 出現率の高い情報を短いビット列で表現
- 出現率の低い情報を長いビット列で表現

ハフマン符号化[2](p. 28)

Ex. ある地方の320日の天気

天気	天気である日数	ビット列での表現
雨が降っていない	160日(2日に1日)	0
小雨が降っている	40日(8日に1日)	100
適度な雨が降っている	20日(16日に1日)	1010
やや強い雨が降っている	20日(16日に1日)	1011
非常に強い雨が降っている	20日(16日に1日)	1100
強い雨が降っている	20日(16日に1日)	1101
弱い雪が降っている	20日(16日に1日)	1110
大雪が降っている	20日(16日に1日)	1111

↓

「10100100…」は、何日分のどんな天気?

ハフマン符号化[3](p. 28)

- 「10100100…」
  - 最初のビットが「1」なので、「雨が降っていない」という天気ではない
  - 2ビット目が「0」なので、「非常に強い雨」、「強い雨」、「弱い雪」、「大雪」という天気ではない
  - 3ビット目が「1」なので、「小雨」という天気ではない
  - 4ビット目が「0」なので、「やや強い雨」という天気ではない

1日目の天気は「適度な雨が降っている」

↓

それぞれの情報を同じ長さのビット列にしなくても情報の表現は可能

※情報の種類ごとに割り当てるビット数を変える方式: 可変長符号

ハフマン符号化[4](p. 29)

- 8種類の天気を普通のやり方で現すと…?
  - 1種類を3ビットで表現: 320日では、320×3=960ビット

ハフマン符号化では…

雨が降っていない	160日	0	1ビット×160日
小雨が降っている	40日	100	3ビット×40日
適度な雨が降っている	20日	1010	4ビット×20日
やや強い雨が降っている	20日	1011	4ビット×20日
非常に強い雨が降っている	20日	1100	4ビット×20日
強い雨が降っている	20日	1101	4ビット×20日
弱い雪が降っている	20日	1110	4ビット×20日
大雪が降っている	20日	1111	4ビット×20日

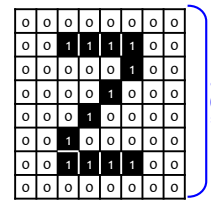
合計: 760ビット

200ビット少なくなっている

※どれだけ少なくなるかは扱う情報によって違う

ランレングス符号化[1](p. 29)

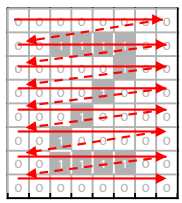
- ランレングス符号化: 白黒2値画像(灰色のない、白と黒のみの画像)の圧縮方法の1つ
  - 主にFAXで使われている



8×8=64ビット
(2値画像は1つ1つの点を1ビットで表現するため)

ランレングス符号化[2](p. 29)

- 2値画像: 白画素と黒画素はある程度固まっている

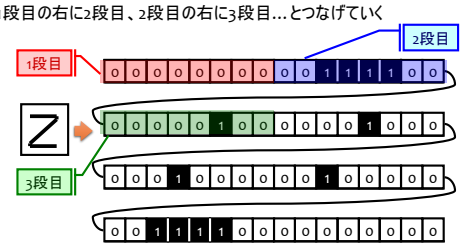


白画素が10個
黒画素が4個
白画素が7個
……
黒画素が4個
白画素が10個

矢印の方向に画素を見ていったとき

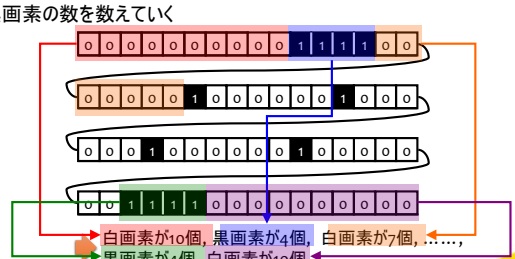
ランレングス符号化[圧縮][1]

- 画像の中の全ての画素を横一列に並べる
  - 横1段目の右に2段目、2段目の右に3段目…とつなげていく



ランレングス符号化[圧縮][2]

- 横一列に並べた画素について、左から順に、連続している白画素と黒画素の数を数えていく



白画素が10個, 黒画素が4個, 白画素が7個, ……,
黒画素が4個, 白画素が10個

ランレングス符号化[圧縮][3]

白画素が10個 黒画素が4個 白画素が7個 ……
黒画素が4個 白画素が10個

画素の個数を2進数で表すと…

1010 0100 0111 0001 0110 0001 0110
0001 0110 0001 0111 0100 1010

52ビット

※個数の中で「1010」(2進数)が最も大きな個数なので、他の個数も、2進数で表現したときの桁数を「1010」(4桁)にあわせる

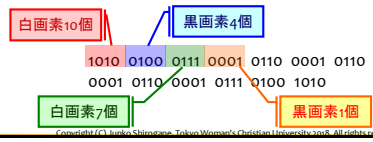
通常の方法で表す(1画素を1ビットで表す)と…
8×8=64ビット

12ビット少なくなっている

※どれだけ少なくなるかは扱う画像の内容によって違う

ランレングス符号化[復号][1]

- 圧縮したものを復号化するには…？
  - 画像の画素の数だけを表したもの(色の情報はない)
  - 白画素の並びと黒画素の並びは必ず交互になる
  - 画像の最も左上隅の画素は白であることが多い
    - 先頭の個数は、白画素の個数として扱う
    - 最も左隅の画素が黒の場合、圧縮時に、最初に出現する白画素の個数を0個としておく

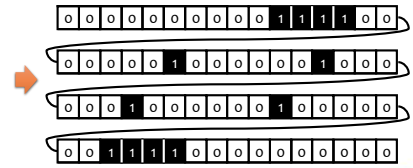


ランレングス符号化[復号][2]

- 画素の個数から、横一列の画素の並びが復元

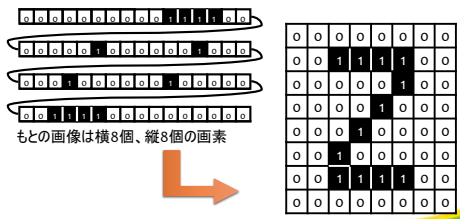
1010 0100 0111 0001 0110 0001 0110
0001 0110 0001 0111 0100 1010

白画素が10個，黒画素が4個，白画素が7個，……，
黒画素が4個，白画素が10個



ランレングス符号化[復号][3]

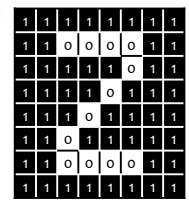
- もとの画像の縦横の画素の数は記録されてある
  - 横一列の画素の並びが復元できると、もとの画像も復元できる



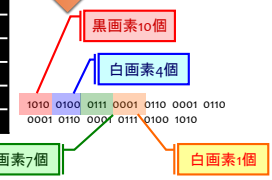
もとの画像は横8個、縦8個の画素

先頭が黒画素のとき[1]

- 前のスライドの画像の、白と黒を逆にした画像を考えると…

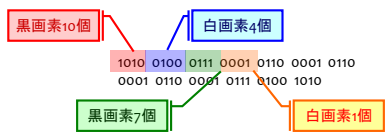


白画素・黒画素の個数を数えると…(前のスライドの画像の白と黒を逆にした画像なので…)

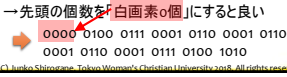


先頭が黒画素のとき[2]

- ランレングス符号化の考え方
  - 白画素・黒画素の個数だけを並べたとき、先頭の個数は白画素の個数とみなす



先頭の個数は黒画素の個数!…でもこれでは困る
→先頭の個数を「白画素0個」にすると良い



次回

- 実習をするので24102教室で授業
  - 圧縮についての説明と実習

期末試験

- 7月31日(火) 1限 24101教室
- 範囲: 前期の内容全て
- 持ち込みはすべて不可
- 試験時間: 60分
- 期末試験の重点勉強ポイント:
  http://www.cis.twcu.ac.jp/~junko/Science/CS1/ExamCheckList.html