HPACK 05 の実装と解析

2014.1.28

山本和彦
@kazu_yamamoto

自己紹介

ネットワーク

メール、IPv6、HTTP
以前はよく IETF に参加

Haskell

Glasgow Haskell Compiler の開発に参加
Mighty という Web サーバを実装

2013.10.12 http2.0 ミニハッカソンでの会話

Jack

HTTP/2.0 は HPACK を実装しないと
何も始まりません。

そうなんだ。。。

お品書き

圧縮率

プロトコルの話
HPACK符号化側

高速化

実装の話
HPACK全般

圧縮率

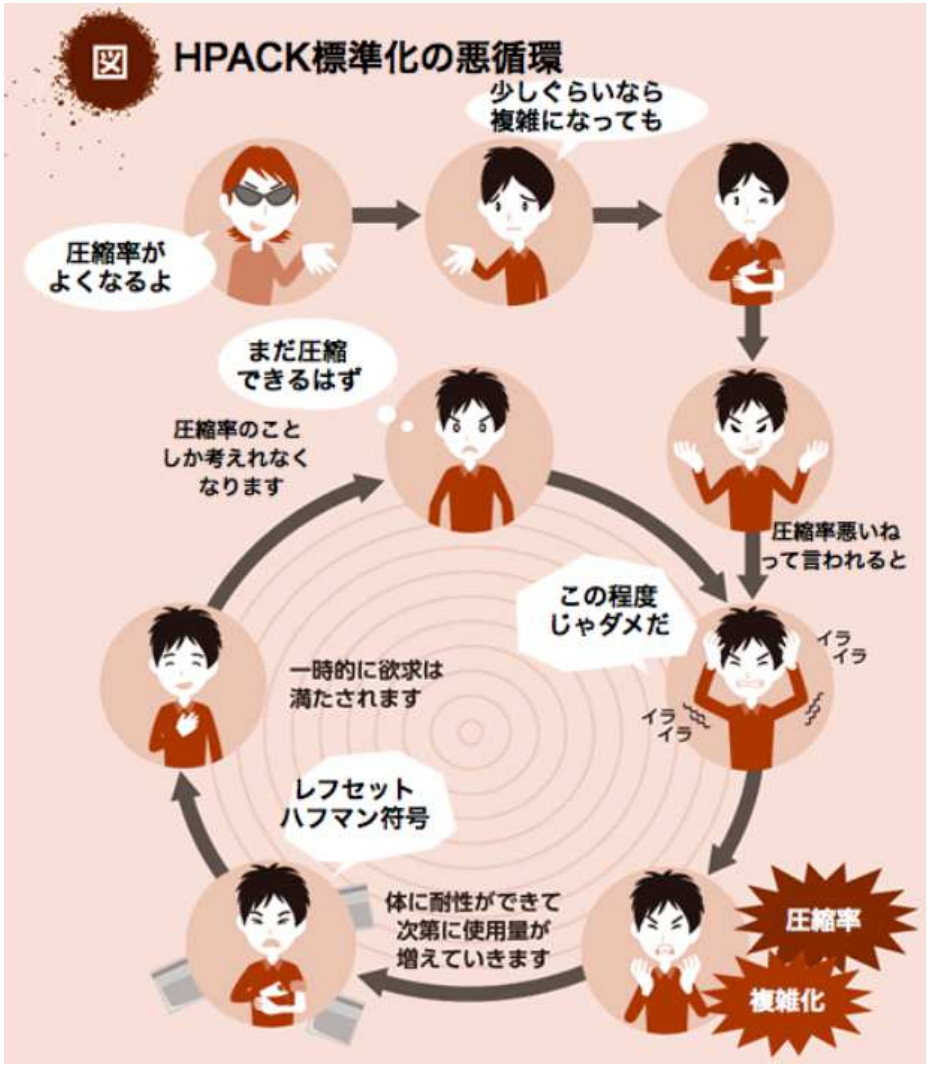


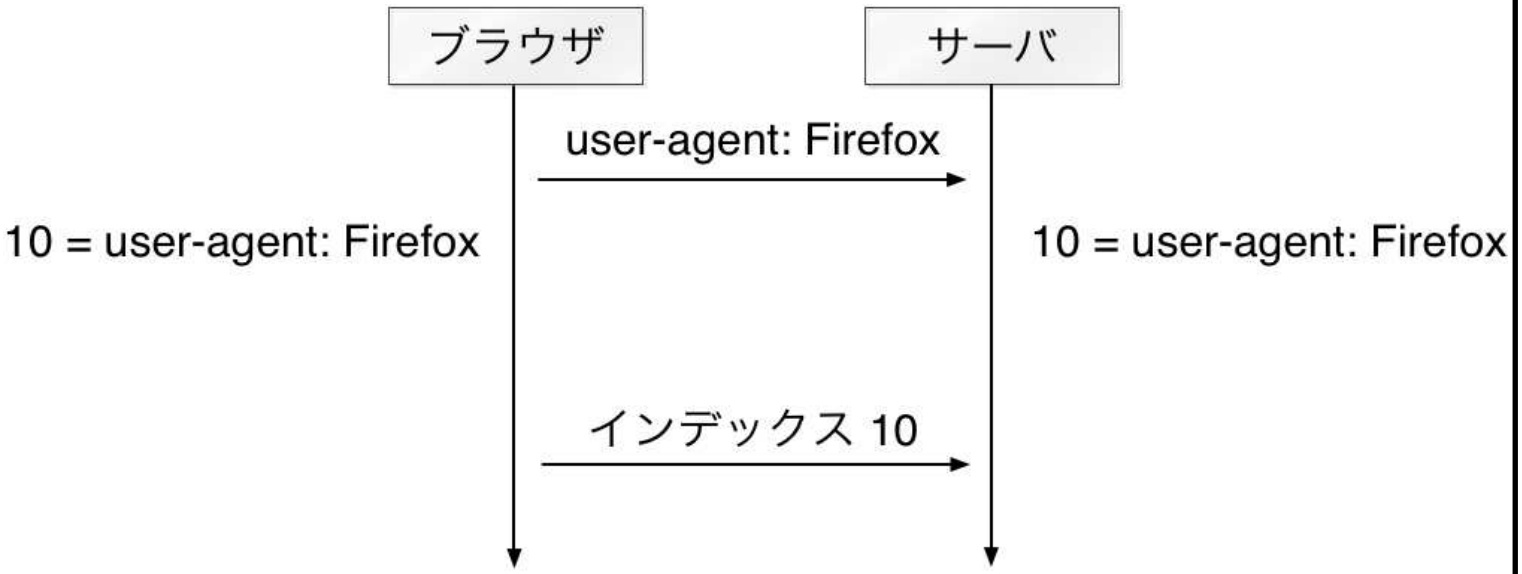
図
HPACK標準化の悪循環
圧縮率が
よくなるよ
少しぐらいなら
複雑になっても
圧縮率悪いね
って言われると
この程度
じゃダメだ
イラ
イラ
イラ
イラ
まだ圧縮
できるはず
圧縮率のこと
しか考えれなく
なります
一時的に欲求は
満たされます
レフセット
ハフマン符号
体に耐性ができて
次第に使用量が
増えていきます
圧縮率
複雑化

HPACK の圧縮技術

インデックス	ヘッダ名/ヘッダ値を番号で表現
リファレンス セット	インデックスの集合 新旧を比較することで差分を計算
ハフマン符号	ヘッダ名/ヘッダ値自体を圧縮

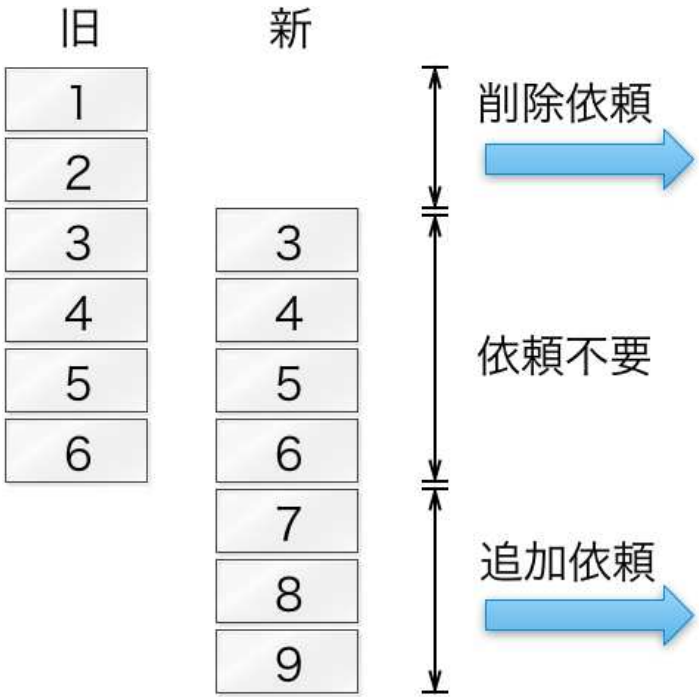
インデックス

- 学習辞書を共有する



リファレンスセット

- インデックスの集合から差分を計算



- 実装は難しい
  - @tatsuhiro-t さんのアルゴリズム
    - http://lists.w3.org/Archives/Public/ietf-http-wg/2013JulSep/1135.html

ハフマン符号

- 出現率の高い文字を短く符号化する

```
'F'      'i'   'r'   'e'
0x46     0x69  0x72  0x65
```

ハフマン符号化

```
1110000 01010 01111 0001
```

ビット列の連結

```
11100000 10100111 10001..
0xe0     0xa7     ...
```

HPACK符号化の種類

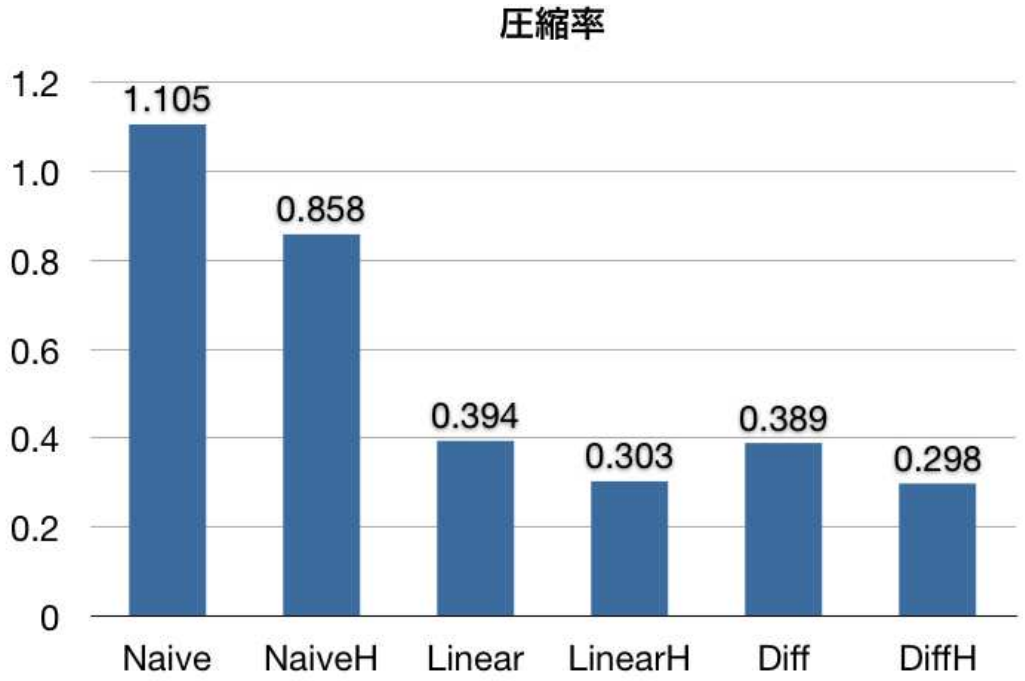
- HPACK の復号化は一意
  - 仕様通りに作ればよい
- HPACK の符号化はだいたい6種類

	インデックス	リファレンスセット	ハフマン符号
Naive	×	×	×
NaiveH	×	×	○
Linear	○	×	×
LinearH	○	×	○
Diff	○	○	×
DiffH	○	○	○

実験

- Haskell の http2 ライブラリ
  - 6種類の HPACK 符号化を実装
  - 相互接続性テストはすべて通過
  - リポジトリ
    - https://github.com/kazu-yamamoto/http2
- 実験
  - 6種類の HPACK 符号化に対する圧縮率を計算
  - 圧縮率の妥当性も確認
  - 実験データ
    - https://github.com/Jxck/hpack-test-case

結果


圧縮率
1.2
1.0
0.8
0.6
0.4
0.2
0
1.105
0.858
0.394
0.303
0.389
0.298
Naive
NaiveH
Linear
LinearH
Diff
DiffH

悲報

Diff は Linear に対して
HEADERS フレーム 1 つあたり
平均 1.57 バイトしか短くならない

(ハフマンは 31.36 バイト)

提案

リファレンスセットを削除する

～ Naive/NaiveH/Linear/LinearH のみ ～

提案の利点

実装が容易になる

相互接続性が向上する

圧縮率は同等

ヘッダの順番が守られる

高速化

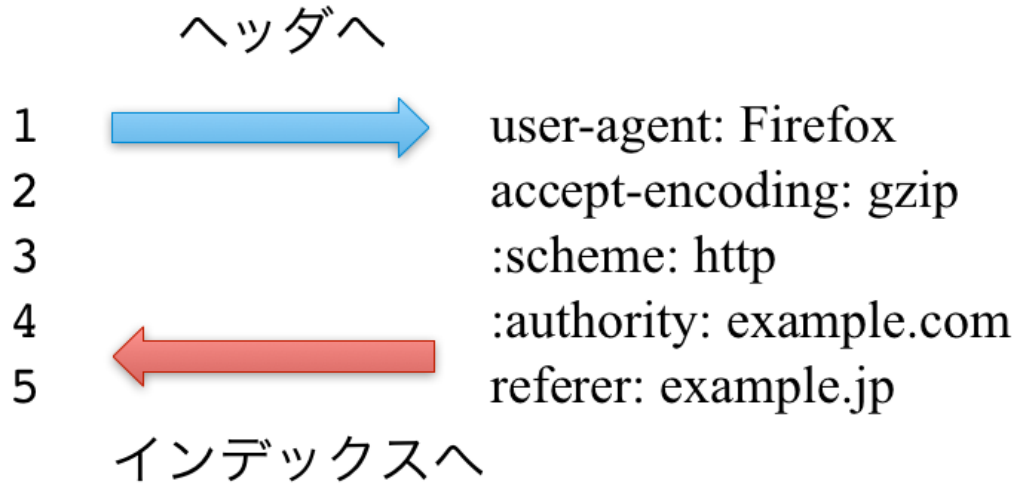
ボトルネック

- GHC プロファイリング調べ

インデックスの探索	ヘッダに対応するインデックスを探索する
ハフマン復号化	ビット列を文字に変換する
ハフマン符号化	文字から変換したビット列を連結する

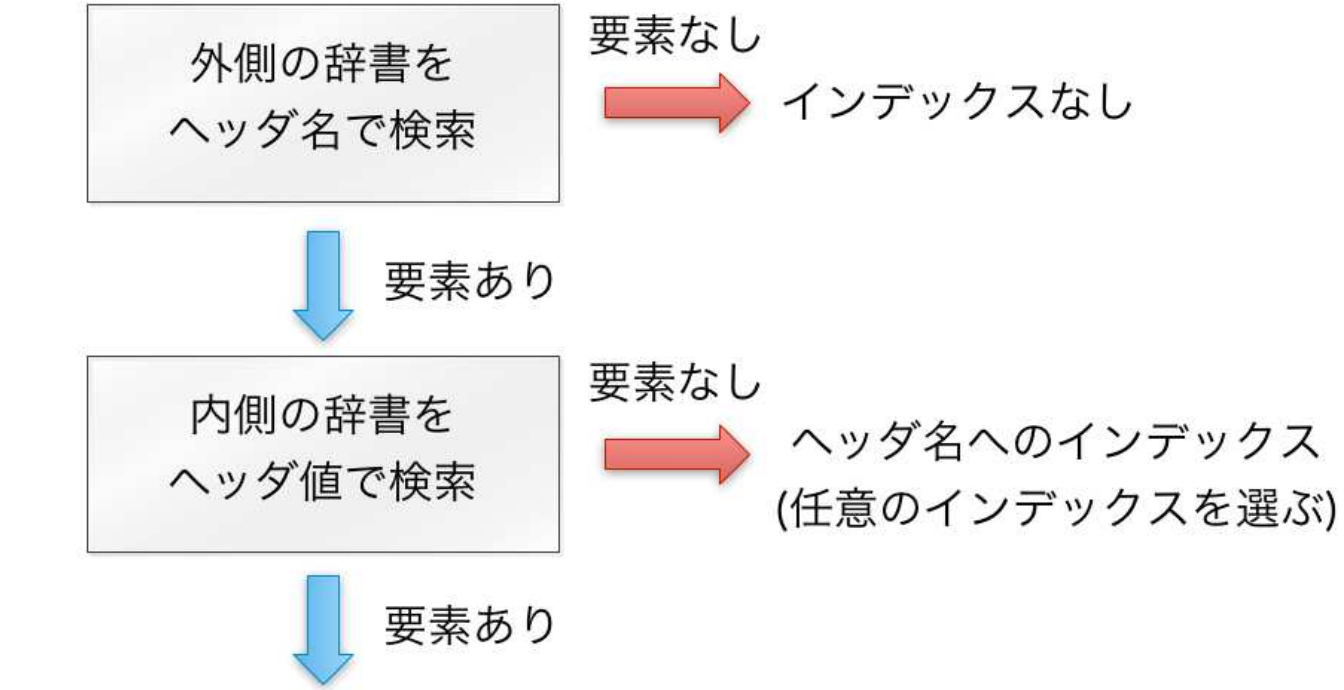
インデックスの探索

- インデックスからヘッダへの変換は高速
  - 配列を引くだけ O(1) in worst
- ヘッダからインデックスへの変換は低速
  - 単純な実装では線形探索 O(N) in worst



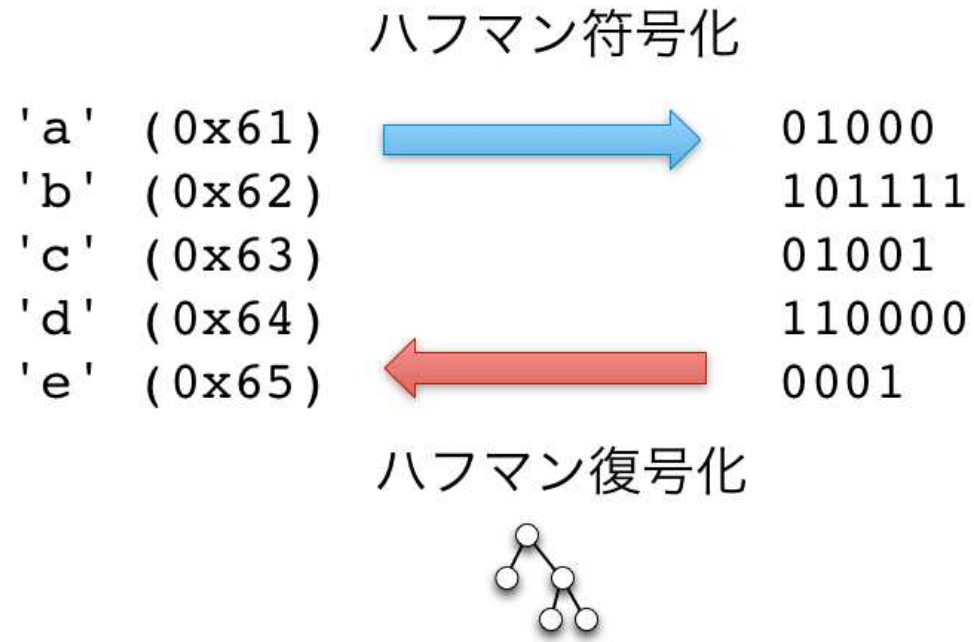
逆インデックス

- 要求
  - ヘッダ名とヘッダ値の組からインデックスを見つける
  - ヘッダ名からインデックスを見つける



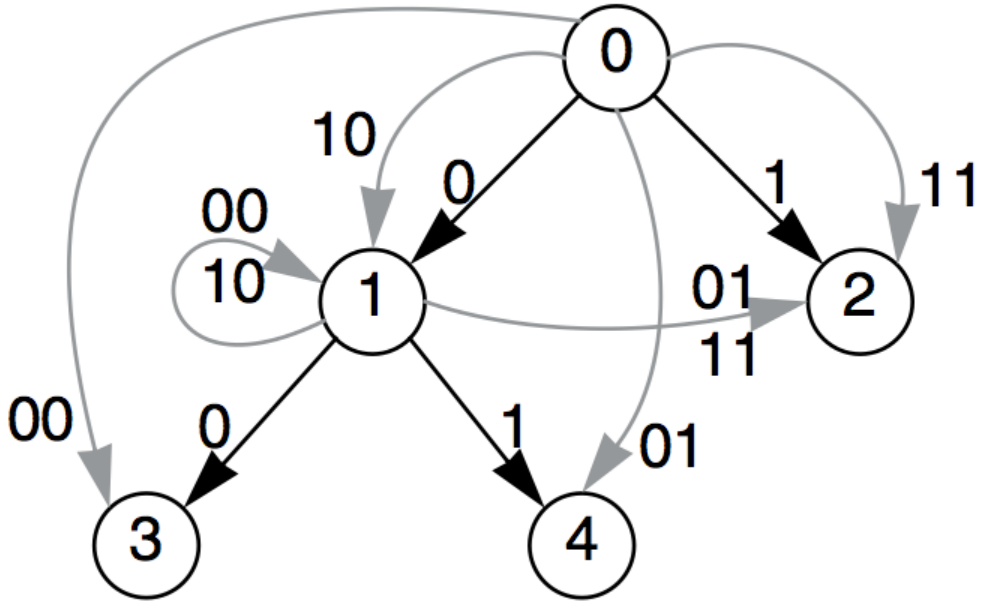
ハフマン符号

- ハフマン符号化は高速
  - 配列を引くだけ O(1) in worst
- ハフマン復号化は低速
  - 単純な実装では二分木を辿る O(log N) in average?



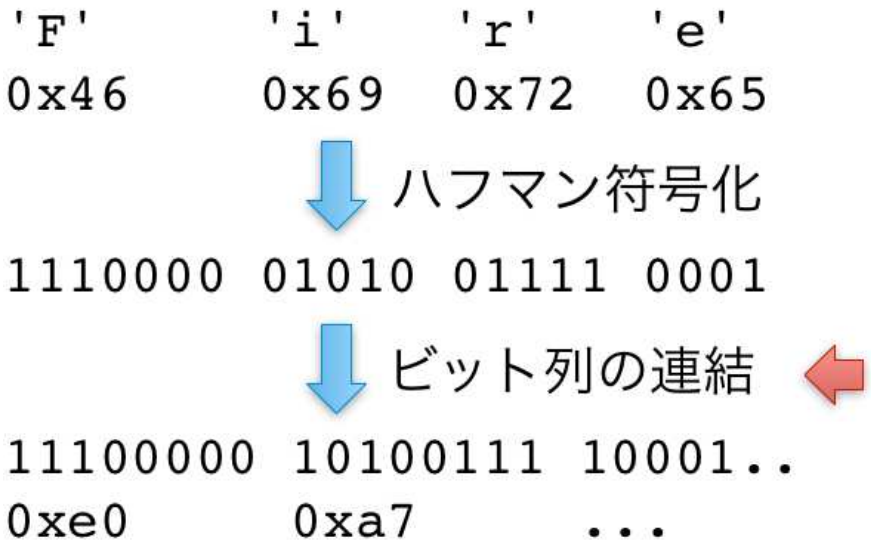
高速なハフマン復号化

- 論文 "Fast Prefix Code Processing"
  - n ビット単位での状態遷移を事前計算
    - n は 4 ビットか 8 ビットがよい
  - 葉にたどり着いたら文字を出力し
    根に戻って残りのビットを消費

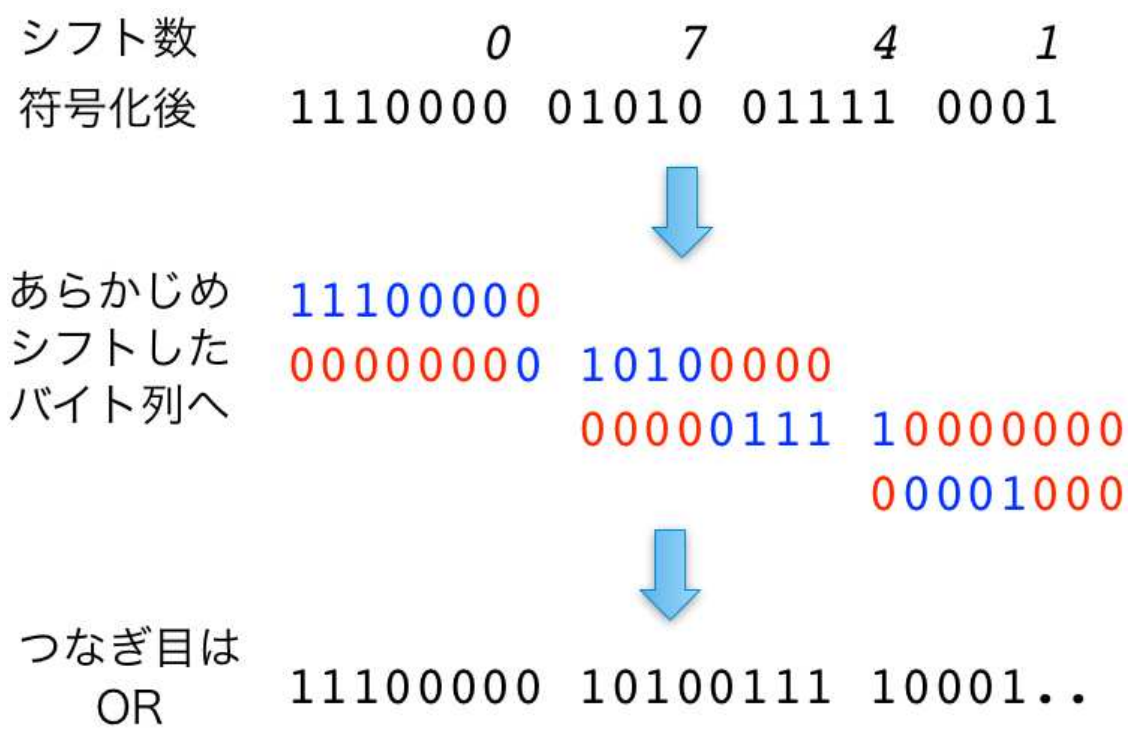


ハフマン符号化の問題

- ビット列の連結は遅い



ビット列の高速な連結


シフト数
0
7
4
1
符号化後
1110000 01010 01111 0001
あらかじめ
シフトした
バイト列へ
11100000
00000000 10100000
00000111 10000000
00001000
つなぎ目は
OR
11100000 10100111 10001..

まとめ

インデックス	高い圧縮率 高速化は可能
リファレンス セット	複雑で圧縮率に貢献しない 仕様から削るべき
ハフマン符号	圧縮率はそれなり 仕様から削ってもいいかも 高速化は可能