Komprese obrazových dat s využitím Huffmanova kódování

Tato dokumentace popisuje principy využívány kompresním programem `huff_codec` implementovaném v jazyce C++ do předmětu kódování a komprese dat. Program slouží pro kódování a dekódování obrázků v odstínech šedi s využitím Huffmanova kanonického kódu. Pro zvýšení kompresního výkonu je uplatněno adaptivní skenování po blocích o velikosti 64x64 bitů a dále jsou využity čtyři modely pracující s rozdílem sousedících pixelů. Tento projekt byl vypracován výhradně na základě informací uvedených na přednáškách z předmětu kódování a komprese dat.

1 Model

V programu jsou využity čtyři modely pracující s rozdílem hodnot sousedních pixelů. Byly implementovány modely procházející obraz po řádcích z leva do prave, po sloupcích shora dolů, paralelně s hlavní diagonálou shora dolů a paralelně s vedlejší diagonálou shora dolů. Kompresní výkon modelu je aproximován velikosti abecedy modelu (počtem rozdílných hodnot). Pro kódování je využit pouze model s nejmenší abecedou. Pro použití modelu během kódování programem je nutné zadat přepínač `-m`.

1.1 Po řádcích

Při průchodu matice bodů X po řádcích zleva do prava je za vztažný bod zvolen levý horní bod $x_{0,0}$. Rozdíly sousedících bodů jsou vypočítány pomocí následujícího předpisu.

$$y_{i,j} = \begin{cases} x_{i,j} - x_{i,j-1} & j > 0 \\ x_{i,j} - x_{i-1,j} & j = 0 \wedge i > 0 \\ 0 & jinak \end{cases}$$

1.2 Po sloupcích

Při průchodu matice bodů X po sloupcích shora dolů je ze vztažný bod zvolen levý horní bod $x_{0,0}$. Rozdíly sousedících bodů jsou vypočítány pomocí následujícího předpisu.

$$y_{i,j} = \begin{cases} x_{i,j} - x_{i-1,j} & i > 0 \\ x_{i,j} - x_{i,j-1} & i = 0 \wedge j > 0 \\ 0 & jinak \end{cases}$$

1.3 Paralelně s hlavní diagonálou

Při průchodu matice bodů X paralelně s hlavní diagonálou z levého horního bodu ke spodnímu pravému je za vztažný bod zvolen levý horní bod $x_{0,0}$. Rozdíly sousedících bodů jsou vypočítány pomocí následujícího předpisu.

$$y_{i,j} = \begin{cases} x_{i,j} - x_{i-1,j-1} & i > 0 \wedge j > 0 \\ x_{i,j} - x_{i-1,j} & i > 0 \wedge j = 0 \\ x_{i,j} - x_{i,j-1} & i = 0 \wedge j > 0 \\ 0 & jinak \end{cases}$$

1.4 Paralelně s vedlejší diagonálou

Při průchodu matice bodů X paralelně s vedlejší diagonálou z pravého horního bodu ke spodnímu levému je za vztažný bod zvolen pravý horní bod $x_{n,n}$. Rozdíly sousedících bodů jsou vypočítány pomocí následujícího předpisu.

$$y_{i,j} = \begin{cases} x_{i,j} - x_{i-1;j+1} & i > 0 \wedge j < n \\ x_{i,j} - x_{i-1;j} & i > 0 \wedge j = n \\ x_{i,j} - x_{i;j+1} & i = 0 \wedge j < n \\ 0 & jinak \end{cases}$$

2 Adaptivní skenování

Velikost vstupní abecedy lze snížit s využitím adaptivního skenování, které rozdělí matici dat na menší bloky o velikosti 64x64 bitů. Během testů se ukázalo, že menší velikost bloku nezvyšuje kompresní výkon, naopak zvyšuje velikost hlaviček v komprimovaných datech. Pro využití adaptivního skenování v programu je nutné zadat přepínač `-a`.

3 Kanonický Huffmanův kód

Táto sekce popisuje využití kanonického Huffmanova kódu pro kódování a dekódování obrazových dat.

3.1 Kódování

Pro kódování dat v matici X s abecedou $\Sigma_{EOF} = \Sigma \cup \{EOF\}$, kde Σ je abeceda matice X a EOF je speciální znak, který se v Σ nevyskytuje, byl využit kanonický Huffmanův kód, který byl získán transformací stromu T reprezentujícího Huffmanův kód pro Σ_{EOF}. Listový uzel $n \in leaves(T)$ stromu T obsahuje znak $a \in \Sigma_{EOF}$, informaci o jeho četnosti výskytu v matici X ($cnt(a) \in \mathbb{N}$), kde $cnt(EOF) = 1$, a o hloubce uzlu ve stromu ($depth(n) \in \mathbb{N}$).

Algorithm 1: Konstrukce stromu Huffmanova kódu

Input: matice dat X, abeceda matice Σ_{EOF}
Output: strom T Huffmanova kódu

```
1  pQueue ← reversePriorityQueue()          // uzly s nejmenším cnt jsou na čele
2  foreach a ∈ Σ_EOF do
3      node ← Node()                                          // vytvoř nový uzel
4      node.symbol ← a
5      node.cnt ← cnt(a)
6      pQueue.put(node)
7  end
8  while |pQueue| > 1 do
9      n ← pQueue.pop()
10     m ← pQueue.pop()
11     node ← Node()                                          // vytvoř nový uzel
12     node.cnt ← n.cnt + m.cnt
13     pQueue.put(node)
14 end
15 return pQueue.pop()
```

Algorithm 2: Konstrukce kanonického Huffmanova kódu

Input: matice dat X, abeceda matice Σ
Output: kódovací funkce $huff : \Sigma_{EOF} \to \mathbb{N}_0$

$\Sigma_{EOF} \leftarrow \Sigma \cup \{EOF\}$
$tree \leftarrow getHuffmanTree(X, \Sigma_{EOF})$
$sortedLeaves \leftarrow ascendSort(leaves(tree))$ `// podle` $depth(node)$ `a` $(node.symbol)$
$n_0 \leftarrow sortedLeaves[0]$
$huff(n_0.symbol) \leftarrow 0$
$i \leftarrow 1$
for $0 < i < |sortedLeaves|$ **do**
    $n_{i-1} \leftarrow sortedLeaves[i-1]$
    $n_i \leftarrow sortedLeaves[i]$
    $huff(n_i.symbol) \leftarrow huff(n_{i-1}.symbol + 1) << n_i.depth - n_{i-1}.depth$
end

Dekódování

ahoj