KAPITOLA 3

ZÁKLADNÉ BEZSTRATOVÉ KÓDY

3.1 HUFFMANOV A SHANNONOV - FANOV KÓD

Huffmanov kód patrí medzi kódy s meniteľnou (variabilnou) dĺžkou slova (VLC). Pretože jeho konštrukcia vychádza z entropie, často býva (spolu s jeho modifikáciami) zaraďovaný medzi tzv. entropické kódy. Využíva podobný princíp ako využil pre svoju abecedu už Morse, keď najviac sa vyskytujúcim hláskam v angličtine priradil kratšie znaky, než zriedkavejším. To znamená, že Huffmanov kód je kódom, ktorý sa snaží optimálne rozdeliť postupnosti binárnych symbolov podľa početnosti výskytu kódovaných prvkov so snahou priblížiť sa ich entropii. Necháme matematické zdôvodnenia matematikom a princíp konštrukcie Huffmanovho kódu si opäť ukážeme na jednoduchom príklade [19].

◆ ***Definícia 3.1*** Pre náhodnú premennú $f_1, f_2, ..., f_k$ a pravdepodobnosti $p(f_i) = p_i$ definujeme entropiu H vzťahom

$$H = -\sum_{i=1}^{k} p_i \cdot \log_2 p_i \,. \tag{3.1}$$

Majme pole hodnôt (napr. kvantizačných úrovní) $q_1, q_2, ..., q_8$ s nasledujúcimi pravdepodobnosťami alebo relatívnymi početnosťami výskytu:

$$p_1 = 0,4 \; p_2 = 0,08 \; p_3 = 0,08 \; p_4 = 0,2 \;\; p_5 = 0,12 \; p_6 = 0,08 \; p_7 = 0,04 \; p_8 = 0$$

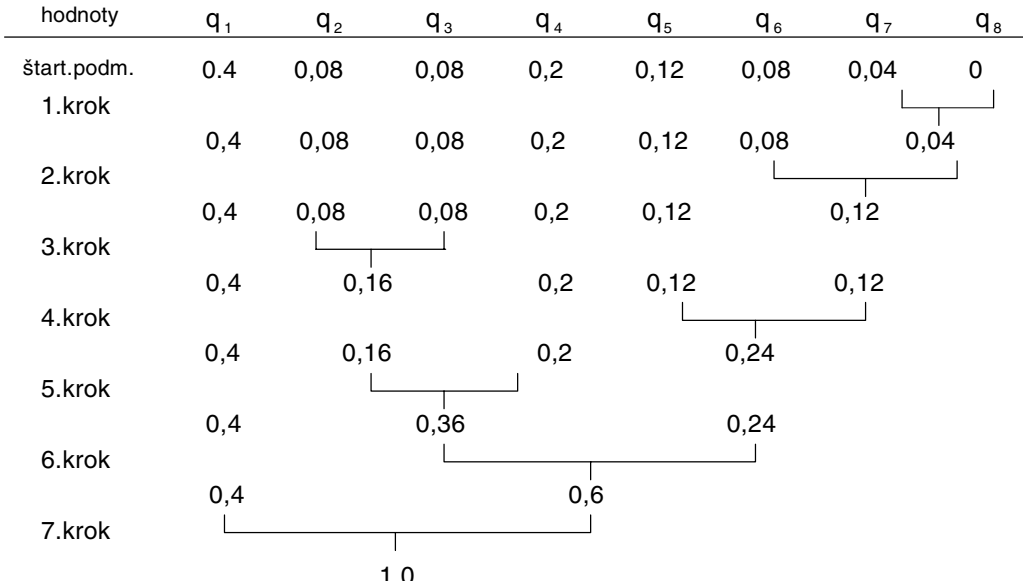
Pre binárne kódovanie ôsmich hodnôt by sme potrebovali 3 bity na hodnotu (prvok). Entropia je

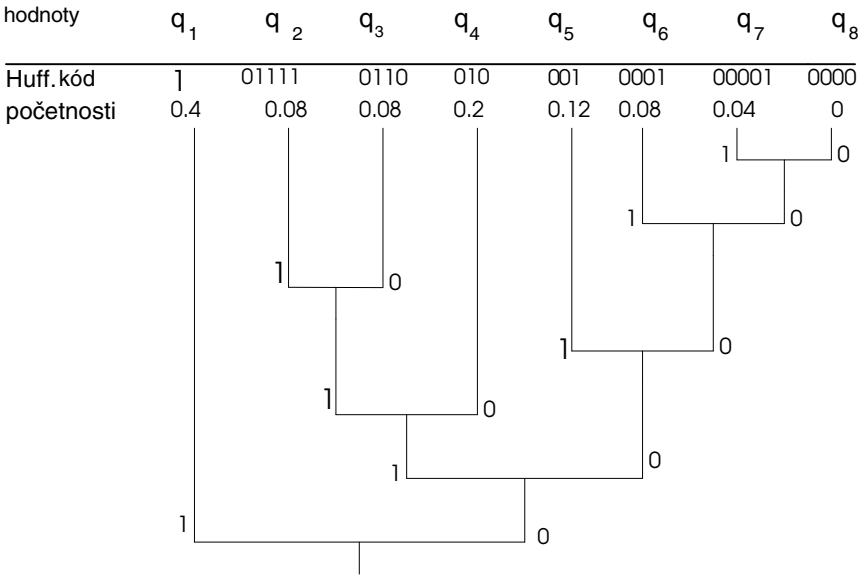
$$H = -[0,4.\log_2(0,4)+3.0,08.\log_2(0,08)+0,2.\log_2(0,2)+0,12.\log_2(0,12)+0,04.\log_2(0,04)]=2,42 \text{ bitu}$$

Postup kódovania je nasledujúci:

1. nájdeme najmenšie dve relatívne početnosti a spočítame ich,
2. vo výsledku z kroku 1 nájdeme dve najmenšie početnosti a spočítame ich,
3. pokračujeme v predchádzajúcom až po relatívnu početnosť 1,
4. kódové slová budú zložené z 0 a 1 podľa stromu, ktorý vznikne sčítavaním, a to tak, že postupujeme spätne od hodnoty početnosti jedna, kde vľavo bude 1 a vpravo 0, čo sú najvyššie bity a rovnakým postupom vypočítame nižšie bity.

Konkrétne na našom príklade to vidno z nasledujúcich dvoch stromových štruktúr [19]:





Výsledná bitová náročnosť na jeden prvok potom je

$$b = 1.0{,}4+3.4.0{,}08+3.0{,}2+3.0{,}12+5.0{,}04 = 2{,}52 \text{ bitu.}$$

Ukážeme si ešte jednu možnú konštrukciu Huffmanovho kódu v jeho štandardnom tvare. Majme dve hodnoty s početnosťami $p_1 = 1/8$ a $p_2 = 7/8$. Pri binárnom kóde by sme potrebovali 1 bit na jednu hodnotu. Ak by sme konštruovali Huffmanov kód predchádzajúcim postupom, dosiahli by sme rovnaký výsledok. Entropia nám však hovorí o možnosti komprimovať tieto údaje, pretože jej hodnota $H = 0{,}544$ bitu. Existuje možnosť kódovať napríklad dibity. Huffmanov kód pre dibity potom je:

dibity	q1q1	q1q2	q2q1	q2q2
relatívne početnosti	1/64	7/64	7/64	49/64
Huffmanov kód	000	001	01	1

Výsledná bitová náročnosť potom je $b = 0{,}680$ bitu.

Huffmanov kód sa používa na priame bezstratové kódovanie samostatne, prípadne ako doplnok predikčného kódovania, "run - lenght" kódovania, či iných bezstratových kódov [2], [29], [39]. Pri kompresii s degradáciou je spravidla poslednou fázou kódovania v transformačnom kódovaní [29], [39], rôznych vektorových kvantizátorov [69], BTC (pri kódovaní rekonštrukčných úrovní) [71] a iných hybridných kodérov. Je to jeden z najdôležitejších a najpoužívanejších kódov, preto mu treba venovať veľkú pozornosť. Pre kódovanie obrazu sa pre nevhodnosť tvaru histogramu používa ako predspracovanie pred Huffmanovym kódovaním ešte predikcia [19], a to buď len jednoriadková, alebo aj v závislosti od iných riadkov. Je samozrejmé, že sa s výhodou použijú väčšinou celistvé násobky, pretože nám ide o bezstratovú kompresiu.

Existuje množstvo modifikácií Huffmanovho kódu, z ktorých najvýznamnejšia modifikácia je pevná tabuľková stromová forma v JPEG (bude spomenuté neskôr) [39], [71] a MPEG. Nejde už v podstate o Huffmanov kód, ale je to podobná štruktúra. Tieto kódy všeobecne nazývame *kódmi s variabilnou dĺžkou slova.* V súčasných odporúčaniach je možne tvoriť tabuľku na základe požiadaviek užívateľa. Ďalšími významnými modifikáciami sú Huffmanove kódy pre vopred neznáme postupnosti, kedy sa Huffmanova tabuľka mení adaptívne podľa kódovaných dát [74]. Tieto kódy sú najdôležitejšie pri prenose dát v reálnom čase bez ich degradácie. Za dáta tu samozrejme môžeme považovať napr. i pohyblivý obraz a pod. Podobný princíp tvorby efektívneho kódu ako je Huffmanov kód, používa *Shannonov - Fanov algoritmus.* Tento algoritmus má o niečo menšiu účinnosť ako Huffmanov kód, ale naproti tomu je rýchlejší a jednoduchší.

Postup kódovania pomocou Shannonovho - Fanovho algoritmu je nasledojúci [75]:

1. usporiadame zdrojové znaky podľa frekvencie výskytu,
2. rozdelíme ich na dve podskupiny s približne rovnakou relatívnou početnosťou, hornej podskupine priradíme 0 a dolnej 1,
3. znaky z každej podskupiny rozdelíme analogicky ako v bode 2,
4. postup opakujeme.

V jednej z nasledujúcich kapitol si ukážeme zaujímavú formu Huffmanovho kódu, ktorý je určený pre dáta z veľkým počtom hodnôt, keď by Huffmanova tabuľka, či strom, boli neúmerne veľké a ich výpočet by tiež mohol zaberať drahocenný čas, ktorý sa dá tak krásne prežiť aj ináč.

3.2 KÓDOVANIE "RUN - LENGTH"

Princíp jednorozmerného kódovania "run-length" je dostatočne známy [29], [39]. Ide o kódovanie binárnych postupností, kde sa kódujú vzdialenosti prechodov postupností núl a jednotiek. V podstate sa predpokladá, že začiatok postupnosti sú nuly (alebo jednotky - záleží na dohode) a ďalej sa číselne vyjadrujú len vzdialenosti od predchádzajúceho prechodu z núl na jednotky alebo z jednotiek na nuly. Napr.:
kódovaná postupnosť 30 bitov:

0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 ,

výsledok kódovania je

3 5 1 2 6 5 1 7

Záleží na počte bitov, ktorými chceme kódovať vzdialenosti. V tomto prípade, ak sme ochotní venovať na kódovanie počtu núl, či jednotiek 3 bity, výsledkom kódovania bude binárna postupnosť

0 1 1 1 0 1 0 0 1 0 1 0 1 1 0 1 0 1 0 0 1 1 1 1,

čo je 24 bitov oproti pôvodným 30 bitom.

Ak však na vzdialenosti predpokladáme 4 bity, čo je do počtu 15, výsledkom bude postupnosť:

0 0 1 1 0 1 0 1 0 0 0 1 0 0 1 0 0 1 1 0 0 1 0 1 0 0 0 1 0 1 1 1,

čo je 32 bitov oproti pôvodným 30 bitom.

Ak by sme počítali len do 3, čo sú 2 bity, zakódovaná postupnosť by bola nasledujúca:
kódovaná postupnosť 30 bitov :

0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 ,

výsledok kódovania:

3 3 0 2 1 2 3 0 3 3 0 2 1 3 0 3 0 1,

binárne: 1 1 1 1 0 0 1 0 0 1 1 0 1 1 0 0 1 1 1 1 0 0 1 0 0 1 1 1 0 0 1 1 0 0 0 1,

čo je až 36 bitov oproti pôvodným tridsiatim.

Záleží teda veľmi významne na tom, akým spôsobom zabezpečíme optimálnu kódovanú dĺžku. Existuje niekoľko spôsobov:

1. štatisticky prehľadáme kódované dáta a zistíme, ktoré počty sú najčastejšie a optimalizujeme pomocou štatistiky maximálny kódovaný počet,
2. metódou pokusov a omylov kódujeme ako v príklade - pre rôzne počty a kódovaným dátam predradíme počet,
3. výsledok kódovania nerobíme binárne, ale využijeme niektorý z kódov s meniteľnou dĺžkou slova - napr. Huffmanov kód alebo niektorú jeho kombináciu s binárnym kódom, prípadne sa neobmedzujeme na maximálny počet, ale priamo kód určujeme podľa vzdialenosti napr.:

vzdialenosť d	kód h(d)
1 - 4	0 x x
5 - 20	1 0 x x x x
21 - 84	1 1 0 x x x x x x
85 - 340	1 1 1 0 x x x x x x x x
341 - 1361	1 1 1 1 0 x x x x x x x x x x
1361 - 3408	1 1 1 1 1 0 x x x x x x x x x x x x

x x x x x x x - napríklad modifikovaný binárny kód čísel v intervale zoradených tak, že prvok, kódovaný ako nula, je prvý prvok v intervale.

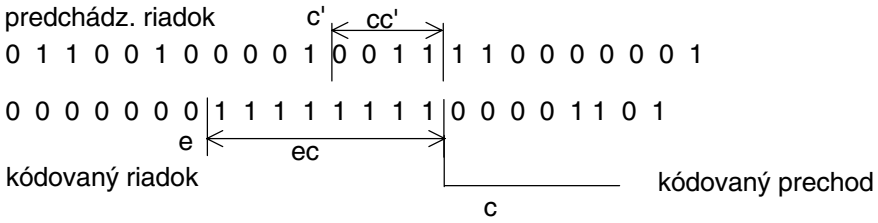
Pri dvojrozmernom kódovaní sa s výhodou využíva dekompozícia poľa na bitové roviny, a potom sa na tieto aplikuje "run-length" kódovanie. Je to preto, lebo pri takomto rozdelení sú dlhšie postupnosti núl a jednotiek. Pretože dokonca nejde len o jednorozmerné postupnosti, ale o celé dvojrozmerné zhluky (čierne a biele plochy), bol tento spôsob modifikovaný pre dvojrozmerné bitové polia [39]. Pri kódovaní vzdialenosti máme na výber (kódujeme po riadkoch):

1. vzdialenosť od posledného prechodu v tom istom riadku,
2. vzdialenosť od rovnakého prechodu v predchádzajúcom riadku smerom vľavo,
3. vzdialenosť od rovnakého prechodu v predchádzajúcom riadku smerom vpravo.

Potom potrebujeme:

a) vybrať tú vzdialenosť, ktorá je najkratšia, a preto kódovateľná najmenším počtom bitov,
b) určiť prefix tejto vzdialenosti d, aby bolo rozlíšiteľné, o ktorú z troch vzdialeností ide.

Takýto spôsob kódovania sa v literatúre nazýva RAC - relative address coding - kódovanie s relatívnymi adresami [39]. Je veľa možností kódovania prefixov a vzdialeností. Jedna možnosť je napríklad [39]:



možná vzdialenosť	vzdialenosť	kód
cc'	0	0
ec alebo cc' (vľavo)	1	100
cc' (vpravo)	1	101
ec	d (d>1)	111h(d)
cc' (c' vľavo)	d (d>1)	1100h(d)
cc' (c' vpravo)	d (d>1)	1101h(d)

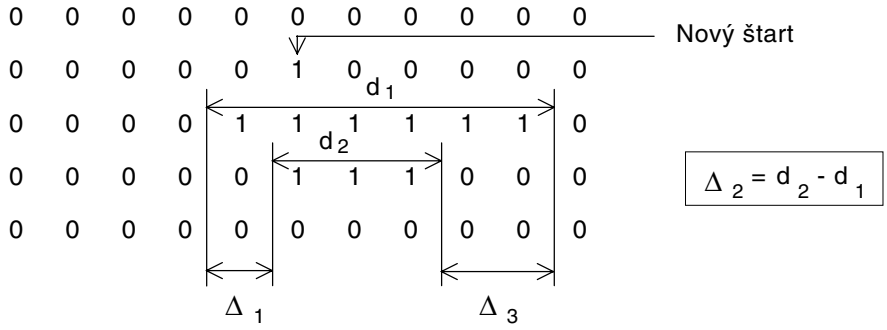
a h(d) môžeme napríklad určiť tak, ako pri jednorozmernom kódovaní podľa tabuľky intervalov [39].

"Run-length" kódovanie je samozrejme kódovanie bez degradácie, preto je vhodné na kódovanie binárnych postupností bez straty informácie. Vo svojej jednorozmernej podobe sa používa napríklad vo faksimilnej telegrafii, dvojrozmerné kódovanie sa používa pri kódovaní obrazu. Tento spôsob kódovania je možné použiť pri rôznych hybridných kódoch, z ktorých ten najvýznamnejší - CTC [2], si ukážeme neskôr.

3.3 KÓDOVANIE OBRYSOV

3.3.1 Kódovanie v bitových rovinách

Podobný princíp kódovania bitových rovín ako je RAC, je kódovanie obrysov v bitových rovinách priamym kódovaním kontúr alebo diferenčným kódovaním vzdialeností prechodov [29], [39]. Predstavme si bielu plochu na čiernom podklade



Sú dve možnosti kódovania:

1. kódujeme nový štart (súradnice), dĺžku prvého riadku objektu a Δ_1, Δ_2 - túto metódu nazývame PDQ - predikčná diferenčná kvantizácia,

2. kódujeme nový štart a dĺžku prvého riadku objektu ako u PDQ a Δ_1, Δ_3 - túto metódu nazývame DDC - dvojnásobné delta kódovanie.

Samozrejme, že na binárne zakódovanie všetkých týchto vzdialeností použijeme, podobne ako u RAC, niektorý z kódov s variabilnou dĺžkou slova. Gonzalez [29] odporúča: C=0 pre postupnosť núl, C=1 pre postupnosť jednotiek, C=0 pre Δ_1, C=1 pre Δ_2 alebo Δ_3.

dĺžka postupnosti 0 alebo 1	$\Delta_1, \Delta_2, \Delta_3$	kódové slovo
1	0	C0
2	+1	C1
3	-1	C0C0
4	+2	C0C1
5	-2	C1C0
6	+3	C1C1
7	-3	C0C0C0
8	+4	C0C0C1
9	-4	C0C1C0
:		
14	+7	C1C1C1
:		

Kód je výhodný pre veľké uzavreté zhluky jednotiek (alebo núl) na pozadí núl (alebo jednotiek). Preto, tak ako aj pri RAC, je potrebné zvoliť takú dvojkovú reprezentáciu bitových rovín, pre ktorú tieto kódy budú najúčinnejšie. Niekedy je to obyčajný dvojkový kód, ale veľmi často je výhodnejšie použiť iný, napríklad Grayov kód [39]. Vtedy vznikajú v bitových rovinách logickejšie čierne plochy, čo je spôsobené jeho logickejšou konštrukciou, ktorá viac pripomína dekadickú sústavu.

3.3.2 Viacúrovňové kódovanie

Podobný spôsob ako je PDQ a DDC kódovanie, je možný realizovať pre kódovanie obrazu, ktorý obsahuje viac kvantizačných úrovní [29]. Principiálne ide o opis uzavretých geometrických útvarov s rovnakou kvantizačnou úrovňou.

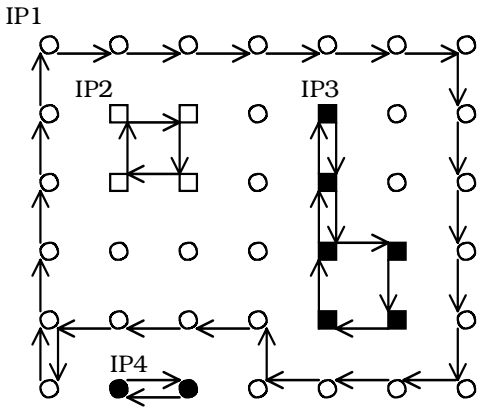
Pre realizáciu takéhoto kódovania je potrebné kódovať:

1. úroveň jasu (prípadne poradie farby),

2. štartovací bod príslušného objektu,

3. tvar objektu - uzavretú cestu od štartovacieho bodu opisujúcu obrys objektu a vracajúcu sa späť k štartu.

Postup je už len logickým riešením daného problému. Je potrebné obrysy označovať tak, aby v každom uzavretom reťazci existovala buď len príslušná kvantizačná úroveň alebo ďalší uzavretý objekt, s tou istou podmienkou. Ukážeme si to na príklade so štyrmi úrovňami:



Obr. 3.1 Príklad viacúrovňového kódovania obrysov

Gonzalez [29] odporúča použiť nasledujúcu tabuľku kódu s variabilnou dĺžkou slova:

poradie obrysu	kódové slovo
1	00
2	01
3	10
4	11

úroveň	kódové slovo
○	00
□	01
■	10
●	11

riadky a stĺpce	kódové slovo
1	000
2	001
:	
8	111

smer popisu	kódové slovo
↑	00
→	01
↓	10
←	11

Na záver potrebujeme zakódovať podľa tabuľky znaky napr. v poradí :
poradie obrysu (IPx), úroveň, riadková súradnica IPx, stĺpcová súradnica IPx, smer prvého posunu, smer druhého posunu, ...

Kód je samozrejme najúčinnejší pri malom počte kvantizačných úrovní a veľkých jednoúrovňových objektoch.

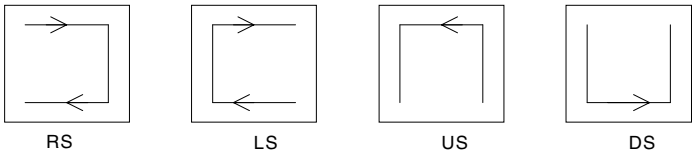
3.4 APLIKÁCIA KRIVIEK VYPĹŇAJÚCICH PRIESTOR NA ZIVOVE - LEMPELOVE KÓDY

Zivove - Lempelove kódy patria medzi slovníkové metódy kódovania (kompresie) údajov. Existujú viaceré jeho modifikácie pre jednorozmerné i dvojrozmerné dáta. Najvýznamnejšie sú asi Welchove modifikácie pre jednorozmerné kódovanie a kódovanie farebného obrazu a metóda samotných autorov pre dvojrozmerné polia.

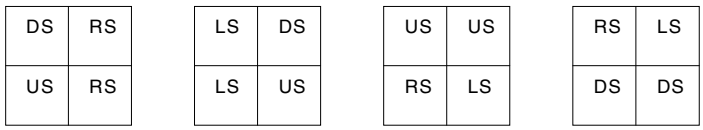
Pri jednorozmernom kódovaní [16] rozdeľuje kodér vstupujúcu postupnosť údajov na menšie časti rôznej dĺžky. V kodéri existuje vyrovnávacia pamäť, obsahujúca údaje, ktoré už boli skôr zakódované. Vo vyrovnávacej pamäti sa tieto údaje prehľadávajú a porovnávajú sa so vstupujúcimi postupnosťami. Týmto spôsobom sa vyhľadáva v údajoch časť, ktorá je totožná s niektorou časťou vstupujúcej postupnosti. Takáto časť sa nazýva prefix. Najdlhší prefix, pre ktorý je nájdený ekvivalent, je potom zakódovaný kódovým slovom, ktoré pozostáva z dvoch častí. Prvú časť tvorí ukazovateľ, ktorý udáva, kde v bufri začína časť predstavujúca ekvivalent kódovaného prefixu. Druhá časť udáva dĺžku prefixu.

V jednorozmernej modifikácii existuje v UNIXe ako príkaz compres, prípadne gzip, ktorého verzia je aj pod operačným systémom DOS.

Dvojrozmerné kódovanie sa od jednorozmerného v zásade líši postupom snímania dát v dvojrozmernom poli. Jeden rekurzívny spôsob, ktorý je priamo od Ziva a Lempela [17], naznačuje obr.3.2. Rekurzívne načítanie môžeme urobiť pomocou štyroch typov blokov:

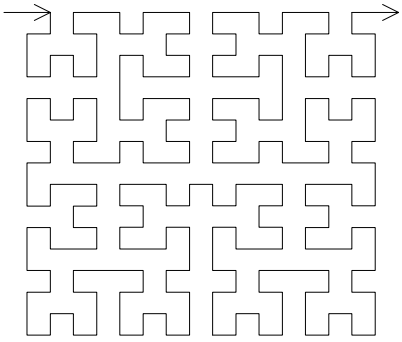


Ďalšia rekurzia potom je:



Obr. 3.2 Rekurzívna definícia snímania

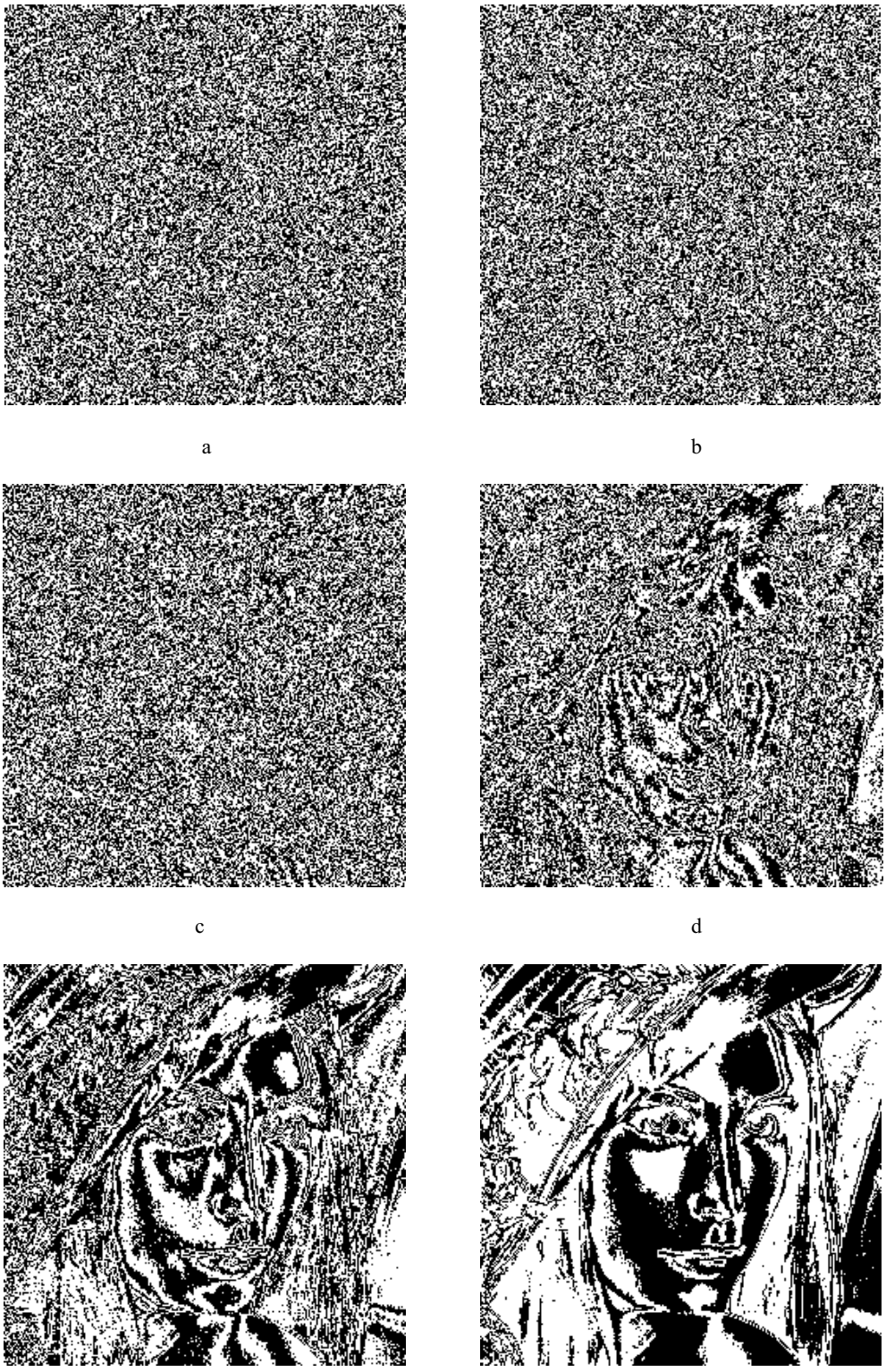
Potom snímanie väčších blokov vyzerá nasledovne:



Obr. 3.3 Štandardné snímanie pre 2^r, $r = 4$

Ďalší postup kódovania je totožný s jednorozmerným kódovaním. Zivove-Lempelove kódy sa používajú pre kompresiu dát bez strát samostatne, prípadne ako doplnok niektorých iných kódov, a to

všade tam, kde sa odporúča použitie niektorého z bezstratových kódov. Veľmi často sú tieto kódy účinnejšie ako Huffmanov kód. Majú však odlišnú podstatu konštrukcie, preto ich použitie namiesto Huffmanovho kódu nie vždy prináša uspokojivý výsledok. Ide hlavne o prípady kódovania reálnych a komplexných čísel napr. v spektrálnej oblasti.



a b c d e f

Obr. 3.4 a, b, c, d, e, f

g h

Obr. 3.4 Výrez obrazu Lena rozložený na bitové roviny: nultá (a), prvá (b), druhá (c), tretia (d), štvrtá (e), piata (f), šiesta (g), siedma (h)

3.5 ARITMETICKÉ KÓDOVANIE

Toto kódovanie prekonáva myšlienku kódovať symbol postupnosťou núl a jednotiek, ale nahrádza postupnosť symbolov jedným desatinným číslom [39]. Len nedávno sa podarilo vyriešiť problémy s implementáciou aritmetického kodovania použitím celých čísel. Výsledkom celého kódovacieho procesu je desatinné číslo menšie ako 1 a väčšie alebo rovnajúce sa nule.

Algoritmus aritmetického kódovania nie je zložitý. Pred kódovaním sa zistí relatívna početnosť jednotlivých znakov a každému sa podľa toho priradí časť intervalu medzi jednotkou a nulou. Zakódovanie symbolu je potom nájdenie intervalu, ktorý mu bol priradený a nasleduje rozdelenie tohto intervalu na podintervaly, úmerné relatívnym početnostiam jednotlivých symbolov znakov. Týmto postupom budeme konvergovať k istému intervalu, z ktorého číslo s najkratšou dekadickou reprezentáciou je výsledkom kódovania. Dekódovanie prebieha obráteným postupom. Podľa prvého čísla nájdeme znak prislúchajúci danému intervalu, potom tento znak z nášho desatinného čísla odstránime a opäť hľadáme do akého intervalu nám číslo patrí. Tu sa vyskytuje jeden problém, a to ako rozhodnúť, kedy sme dekódovali celý prúd znakov. Jednou možnosťou je koniec označiť špeciálnym EOB znakom alebo uchovávať dĺžku kódovanej správy.

Ďalším problémom sa zdá byť obmedzená presnosť desatinných čísel v rôznych implementáciach. V prípade straty presnosti by došlo k nesprávnemu dekódovaniu symbolov a tým k poškodeniu dát. Preto je rozumné nepoužívať desatinné čísla, ale prejsť na celé čísla. Na prvý pohľad sa to nejaví ako vhodná cesta, ale stačí, ak si uvedomíme, že z matematického hľadiska je 1=0.9999999.... (periodicky) a binárna jednotka sa rovná 0.1111111... (binárne). Táto úprava nám umožní prejsť z desatinných čísel na celé.

Na jednoduchom príklade uvádzame spôsob kódovania správy [39], napr. "abccd", ak sme predtým zistili relatívne početnosti.

symbol	početnosť	inicializačný interval
a	0,2	<0,0; 0.2)
b	0,2	<0,2; 0.4)
c	0,4	<0,4; 0.8)
d	0,2	<0,8; 1.0)

Aplikovaním opísaného algoritmu postupne dostávame tieto rozsahy:

kódovaný symbol	dolná hranica	horná hranica
	0.0	1,0
a	0,0	0,2
b	0,04	0,08
c	0,056	0,072
c	0,0624	0,0688
d	0,06752	0,0688

Výslednou hodnotou pre text abccd je napríklad číslo 0,068. Ak si pozorne prezrieme jednotlivé čísla v kódovanom texte vidíme, že akonáhle má horná a dolná hranica na príslušných miestach totožné číslice, dané číslo sa už nemení a môže sa uložiť, pričom v ďalších výpočtoch ho už netreba brať do úvahy. To znamená, že sme zakódovali postupnosť 5 znakov troma desatinnými miestami.

Proces dekódovania sa vykonáva opať podľa opísaného algoritmu. Tučné písmo vyjadruje interval, do ktorého kódové slovo padne v príslušnom kroku dekódovania postupnosti:

kódové slovo	výstupný znak	dolná hranica	horná hranica	rozsah
0,068	**a**	**0,0**	**0,2**	**0,2**
	b	0,2	0,4	0,2
	c	0,4	0,8	0,4
	d	0,8	1,0	0,2
	aa	0,0	0,24	0,04
0,068	**ab**	**0,04**	**0,08**	**0,04**
	ac	0,08	0,16	0,08
	ad	0,16	0,2	0,04
	aba	0,04	0,048	0,008
	abb	0,048	0,056	0,008
0,068	**abc**	**0,056**	**0,072**	**0,016**
	abd	0,072	0,08	0,008
	abca	0,056	0,0592	0,0032
	abcb	0,0592	0,0624	0,0032
0,068	**abcc**	**0,0624**	**0,0688**	**0,0064**
	abcd	0,0688	0,072	0,0032
	abcca	0,0624	0,0638	0,00128
	abccb	0,0638	0,06496	0,00128
	abccc	0,06496	0,06752	0,00256
0,068	**abccd**	**0,06752**	**0,0688**	**0,00128**

Opísaním tučných znakov získavame dekódovanú postupnosť abccd. (Napríklad d predstavuje v celej množine, ktorú takto budeme kódovať, znak EOB.)

Napriek niektorým tažkostiam pri realizácii sa oplatí aritmetickému kódovaniu venovať zvýšenú pozornosť, pretože v prevažnej väčšine aplikácií dosahuje lepšie výsledky ako Huffmanove kódovanie. Preto sa už v mnohých aplikáciách vyskytuje aj pri návrhu nových štandardov.