Aritmetické kódovanie (AK) vlastnosti

- Najdôležitejšia vlastnosť: flexibilita
  - Môže sa používať v spojení s ľubovoľným modelom, ktorý produkuje pravdepodobnosť udalostí – dôležité, lebo veľké kompresné pomery sa dajú dosiahnuť len pomocou sofistikovaného modelu vstupných dát
  - Modely môžu byť adaptívne
  - Môže byť použitých viacero rôznych modelov …
- Druhá najdôležitejšia vlastnosť: optimalita spomenutá na predchádzajúcom slide
  - Najmä ak pravdepodobnosť nejakého symbolu je blízka 1, potom aritmetické kódovanie dáva podstatne lepšie výsledky ako iné metódy.
- Nevýhody
  - Najdôležitejšia = Pomalosť / výpočtová náročnosť
  - Nevhodnosť použitia v aplikáciách v reálnom čase
  - Nutnosť indikovať koniec súboru (buď špeciálnym symbolom, alebo prenášaním počtu bitov výsledného čísla)
  - Slabá odolnosť voči chybám (najmä ak sa používajú adaptívne modely)

Základný algoritmus výpočtu AK

Cieľ:

vyslať na výstup jedno desatinné číslo, ktoré presne určuje našu množinu symbolov.

Postupne budeme spresňovať interval, v ktorom sa výsledné číslo nachádza

Vstup:

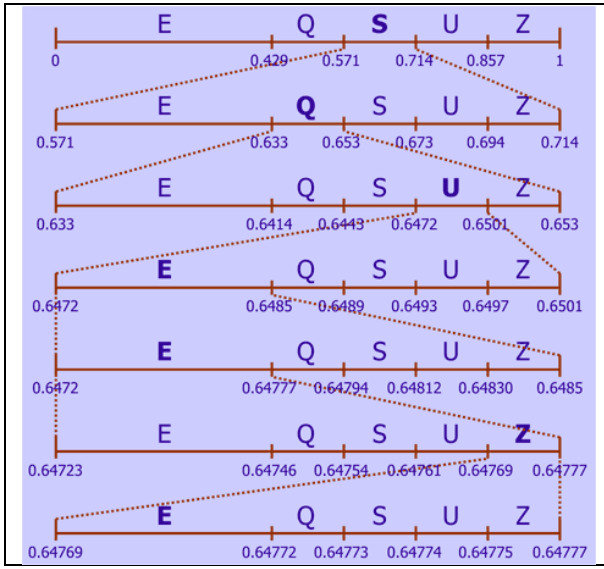
Majme usporiadanú množinu symbolov a_i ktoré tvoria súbor A o veľkosti N symbolov, ktorý máme zakódovať

Množinu rôznych symbolov a_i označme ako U a jej prvky u_j.

Veľkosť množiny U označme ako M. Pravdepodobnosť každého $u_j$označme p_j a odpovedajúcu množinu P

Algoritmus:

1) Nastavíme aktuálny interval $[L, H)$, na $[0, 1)$.
2) Opakujeme pre (i=0;i<N;i++)
   a. Aktuálny interval rozdelíme na M podintervalov proporcionálne k p_j
   b. Zvolíme podinterval j , pre ktorý $u_j = a_i$
   c. Na výstup pošleme dostatočný počet bitov, ktoré rozlíšia zvolený interval od všetkých ostatných podintervalov



Napr. kódujme slovo „SQUEEZE“

- Máme 5 rôznych symbolov s pravdepodobnosťami:
  - p(S)=p(Q)=p(U)= p(Z)=1/7=0.143
  - p(E)=3/7=0.429
- výsledok je v intervale 0.64769-0.64772
- t.j. stačí na výstup poslať ľubovoľné čislo z toho intervalu (z čo najmenším počom bitov)
- takéto číslo je $0.101001011101_2 = 0.647705_{10}$
- výsledok je 12 bitov, to je dokonca pod teoretickou hranicou, ktoré je 14.9 bitu (pre vysvetlenie pozri o nasledujúci slide)
- viac informácií nájdete napr. tu: http://www.bilsen.com/aic/cabac.shtml

Ďalší príklad

Kódujme $A = \{bbb\}$, pričom $U = \{a, b, EOF\}$, pričom ich pravdepodobnosti sú $P = \{0.4, 0.5, 0.1\}$

Aktuálny interval (AI)	Akcia	Subintervaly			Vstup
		a	b	EOF	
[0.000, 1.000)	Delenie	[0.000, 0.400)	**[0.400, 0.900)**	[0.900, 1.000)	b
[0.400, 0.900)	Delenie	[0.400, 0,600)	**[0.600, 0,850)**	[0.850, 0,900)	b
[0.600, 0,850)	Delenie	[0.600, 0,700)	**[0.700, 0,825)**	[0.825, 0,850)	b
[0.700, 0,825)	Delenie	[0.700, 0,750)	[0.750, 0,8125)	**[0.8125, 0,825)**	EOF
[0.8125, 0,825)					

Pomocné výpočty:

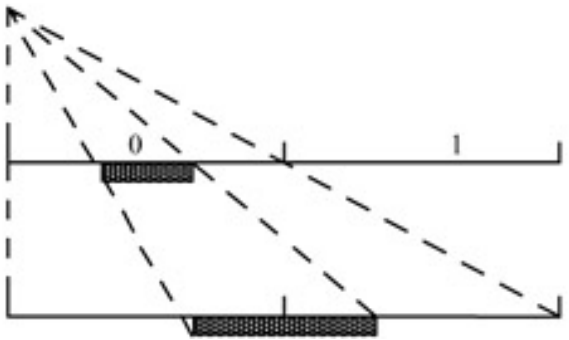
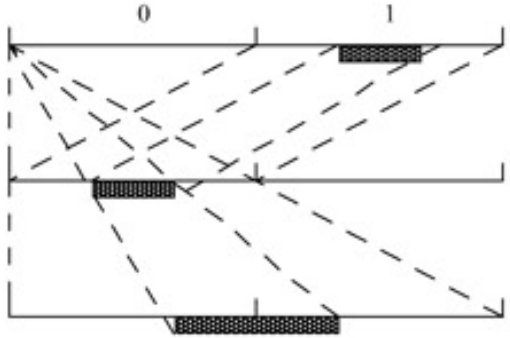
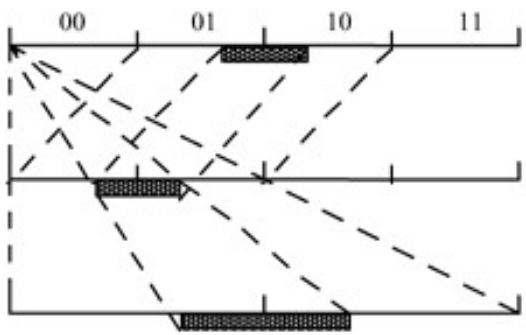
proporcionálne delenie intervalu [0.400, 0.900)	0.4*0.5=0.200→ [0.400, 0,600) 0.5*0.5=0.250→ **[0.600, 0,850)** 0.1*0.5=0.050→ [0.850, 0,900)
proporcionálne delenie intervalu [0.600, 0,850)	0.4*0.25=0.100→ [0.600, 0,700) 0.5*0.25=0.125→ **[0.700, 0,825)** 0.1*0.25=0.025→ [0.825, 0,850)
proporcionálne delenie intervalu [0.700, 0,825)	0.4*0.125=0.0500→ [0.700, 0,750) 0.5*0.125=0.0625→ [0.750, 0,8125) 0.1*0.125=0.0125→ **[0.8125, 0,825)**

Výsledný interval je

- $[0.8125, 0{,}825)_{10} = [0.1101, 0{,}11010011)$
- výstupom a teda môže byť napr. číslo 1101, ale pozor ! ak prenášame počet bitov výsledného čísla, treba preniesť **1101000**, lebo až to nám identifikuje interval jednoznačne
    - keby sme vyslali iba **110100**, možný vysledok je **0.110100**1111 > 0,11010011 (t.j. vačší ako horná hranica nášho intervalu!)

Implementácia AK pri konečnej aritmetickej presnosti

- každý bit poslať na výstupu okamžite, ako je jasné že patrí do výstupného intervalu
- pravidlá sú nasledovné
  - okamžite po kroku 2b („Zvolíme podinterval j , pre ktorý $u_j = a_i$„) v algoritme opakujeme nasledovné kroky dokola, pokiaľ nenastane KONIEC (K) - toto sa volá testovacia slučka:
    - ak aktuálny podinterval (AP) neleží celý ani v jednom z podintervalov [0, ½), [1/4, ¾) [½, 1) nedáme na výstup žiadny bit a je KONIEC (**K**) – **AK znižuje bitovú náročnosť hlavne pri výskyte veľkých AP**
    - ak aktuálny podinterval je v intervale [0, ½) dáme na výstup 0 a všetky F bity ako JEDNOTKY
      - Potom zdvojíme (ZDV) veľkosť intervalu smerom **doprava (ZDVP)**
      - **Vysvetlenie:** Výsledný interval sa jednoznačne nachádza v prvej polovici, druhú polovicu intervalu môžeme zahodiť a prvú polovicu rozšíriť na celý interval (viď obr. Vľavo)
    - ak aktuálny podinterval je v intervale [½,1) dáme na výstup 1 a všetky F bity ako NULY
      - Potom zdvojíme (ZDV) veľkosť intervalu smerom **doľava (ZDVL)**
    - ak aktuálny podinterval je v intervale [1/4, ¾), pričom jedna hodnota je menšia ako 0.5 a druhá väčšia, alebo rovná) zapamätáme si, že treba neskôr vyslať 1x F (follow up) bit
      - Potom zdvojíme (ZDV) veľkosť intervalu smerom **na obe strany od stredu (ZDVS)**

ZDVP	ZDVL	ZDVS
		

Aktuálny interval (AI)	Akcia	Subintervaly AI			Vstup
		a	b	EOF	
[0.000, 1.000)	AI nespĺňa podmienky(K) → delenie AI	[0.000, 0.400)	**[0.400, 0.900)**	[0.900, 1.000)	b
[0.400, 0.900)	AI nespĺňa podmienky(K) → delenie AI	[0.400, 0,600)	**[0.600, 0,850)**	[0.850, 0,900)	b
[0.600, 0,850)	AI ∈[½,1) → DAJ **1**, nový AI (ZDVL1)				
[0.200, 0,700)	AI nespĺňa podmienky(K) → delenie AI	[0.200, 0,400)	**[0.400, 0,650)**	[0.650, 0,700)	b
[0.400, 0,650)	AI ∈[1/4,3/4) → F, nový AI (ZDVS2)				
[0.300, 0,800)	AI nespĺňa podmienky(K) → delenie AI	[0.300, 0,500)	[0.500, 0,750)	**[0.750, 0,800)**	EOF
[0.750, 0,800)	AI ∈[½,1) → DAJ **1+F(0)**, nový AI (ZDVL3)				
[0.500, 0,600)	AI ∈[½,1) → DAJ **1**, nový AI (ZDVL4)				
[0.000, 0,200)	AI ∈[0, ½) → DAJ **0**, nový AI (ZDVP5)				
[0.000, 0,400)	AI ∈[0, ½) → DAJ **0**, nový AI (ZDVP6)				
[0.000, 0,800)	AI nespĺňa podmienky(K) → delenie AI				neni

ZDVL1: Zdvojenie intervalu [0.600, 0,850) smerom doľava= [0.2, 0.7)

- Zdvojenie smerom doľava, t.j. 0.6 je vzdialené od 1 o 0.4, teraz má byť o 0.8 → hodnota 0.2
- Zdvojenie smerom doľava, t.j. 0.85 je vzdialené od 1 o 0.15, teraz má byť o 0.3 → hodnota 0.7

proporcionálne delenie intervalu [0.200, 0.700)	0.4*0.5=0.200→ [0.200, 0,400) 0.5*0.5=0.250→ **[0.400, 0,650)** 0.1*0.5=0.050→ [0.650, 0,700)

ZDVS2: Zdvojenie intervalu [0.400, 0,650) smerom na obe strany= [0.3, 0.8)

- Zdvojenie smerom doľava, t.j. 0.4 je vzdialené od 0.5 o 0.1, teraz má byť o 0.2 → hodnota 0.3
- Zdvojenie smerom doprava, t.j. 0.65 je vzdialené od 0.5 o 0.15, teraz má byť o 0.3 → hodnota 0.8

proporcionálne delenie intervalu [0.300, 0.800)	0.4*0.5=0.200→ [0.300, 0,500) 0.5*0.5=0.250→ [0.500, 0,750) **0.1*0.5=0.050→ [0.750, 0,800)**

ZDVL3: Zdvojenie intervalu [0.750, 0,800) smerom smerom doľava = [0.5, 0.6)

- Zdvojenie smerom doľava, t.j. 0.75 je vzdialené od 1 o 0.25, teraz má byť o 0.5 → hodnota 0.5
- Zdvojenie smerom doľava, t.j. 0.80 je vzdialené od 1 o 0.20, teraz má byť o 0.4 → hodnota 0.6

ZDVL4: Zdvojenie intervalu [0.500, 0,600) smerom smerom doľava = [0.0, 0.2)

- Zdvojenie smerom doľava, t.j. 0.50 je vzdialené od 1 o 0.5, teraz má byť o 1.0 → hodnota 0.0
- Zdvojenie smerom doľava, t.j. 0.60 je vzdialené od 1 o 0.4, teraz má byť o 0.8 → hodnota 0.2

ZDVP5 Zdvojenie intervalu [0.000, 0,200) smerom smerom doprava = [0.0, 0.4)

- Zdvojenie smerom doprava, t.j. 0.00 je vzdialené od 0 o 0.0, teraz má byť o 0.0 → hodnota 0.0
- Zdvojenie smerom doprava, t.j. 0.20 je vzdialené od 0 o 0.2, teraz má byť o 0.4 → hodnota 0.4

ZDVP6 Zdvojenie intervalu [0.000, 0,400) smerom smerom doprava = [0.0, 0.8)

- Zdvojenie smerom doprava, t.j. 0.00 je vzdialené od 0 o 0.0, teraz má byť o 0.0 → hodnota 0.0
- Zdvojenie smerom doprava, t.j. 0.40 je vzdialené od 0 o 0.4, teraz má byť o 0.8 → hodnota 0.8

Nemáme žiadne ďalšie vstupy na konci, nedá sa deliť, t.j. sme na konci algoritmu, náš konečný interval je [0.000, 0.800) a doteraz vyslané bity sú 110100. Platí:

- Náš konečný interval nepokrýva celý interval [0, 1). Existuje teda možnosť, že z neho „vylezieme“
- Chceme vytvoriť číslo, ktoré jednoznačne patrí tomuto intervalu
- stačí pridať **0** – jednoznačne identifikuje [0, 0.5). Alebo pridáme **00** - jednoznačne identifikujeme [0, 0.25)), alebo **01** - jednoznačne identifikujeme [0.25, 0.5), alebo **10** –jednoznačne identifikuje [0.5, 0.75). Pozor, **11** pridať nesmieme, lebo [0.75, 1) siaha mimo nášho intervalu. Najkratšia jednoznačná voľba je teda **0**.
- Celý výsledok je teda: 110100**0**., toto číslo jednoznačne patrí výslednému intervalu bez ohľadu na to aké bity mu doplníme napravo.

Poznámka k adaptivite AC:

- Po každej testovacej slučke sa môžu aktualizovať pravdepodobnosti symbolov ak používame adaptívny aritmetický kód.

Ako toto implementovať pomocou celočíselnej aritmetiky?

Jednoducho:

- Namiesto Intervalu [0,1) a reálnych čísiel → použijeme interval [0, N) a celé čísla
  - kde N je počet symbolov, ktoré máme kódovať
  - Pravdepodobnosť každého symbolu u_j je potom jeho početnosť c_j delená N
  - a platí, že $1 = \sum_j \frac{c_j}{N}$, t.j. sumárna pravdepodobnosť je jednotková
- následne pri upravovaní intervalu rátame s početnosťami namiesto pravdepodobnosťami robíme trunc (pracujeme s celočíselnou časťou)
- Tento postup pri 32 bitoch umožňuje ísť po 4GB vstupnej veľkosti (t.j. 2^{32}) ak symbol je jeden BYTE ...

Druhá možnosť je zvoliť celočíselnú aritmetiku, ktorú máme k dispozícii (napr. iba 16 bitov) a aproximovať AC v rámci možností.

- Hoci aj trochu nepresne, ak bitovo identicky bude pracovať enkóder aj dekóder, postup bude fungovať (hoci aj suboptimálne)

Najjednoduchší aritmetický kóder (AK)?

- Binárny (pozná 2 vstupné symboly) aritmetický kóder
- Používaný napr. pre čiernobiele obrazy
- Vysoko efektívny je, keď pravdepodobnosť jedného symbolu je blízka 1
- Experiment
  - Predpokladajme 2 symboly s pravdepodobnosťami 0.01 a 0.99.
  - Vyšlime 1000 symbolov (990 + 10)
  - Aký je teoretický limit? $pocet_{bitov} = 990(-\log_2 0.99) + 10(-\log_2 0.01) = 14.35 + 66{,}44 = 80{,}79$
  - Ako si s tým poradí Huffmanov kód?
    - Symbol 1 = 0, Symbol 2 = 1
    - Správa=990+10=1000 bitov
  - Ako si s tým poradí aritmetický kód?
    - Ako postupujeme ako minule
      - Kódujeme číslo v dvojkovej sústave, napr.: $0.000 \ldots 0001111111111_2$
      - T.j. napr. 990 symbolov1, potom 10 symbolov2
      - Výsledok nemusíme prevádzať, už ho mame v dvojkovej sústave ...
    - Nič sme neušetrili ... kde je pes zakopaný ?????
      - No predsa v tom, že sme skĺzli do neefektivity, prešli sme na rovnomerné delenie pravdepodobnosti 0.5 a 0.5 (nie 0.01 a 0.99)
  - v čom teda šetrí aritmetický kód? Ako vie zakódovať niečo pod 1/bit symbol ???
- Aritmetický kód spotrebuje toľko bitov, koľko potrebujeme na presnú identifikáciu výsledného intervalu.
  - ak pri každom delení zachováme 99% šírky intervalu a iba veľmi zriedka skočíme na 1% intervalu, stačí nám na identifikáciu výsledného intervalu málo bitov (približne toľko, koľko hovorí teoretický limit)
  - t.j. každý symbol s p=0.99 nás stojí cca 0.15 bitu, a symbol s p=0.01 nás stojí cca 6,64 bitu.
- <u>Spomínate, čo bolo cieľom EBCOTu? Vysoká pravdepodobnosť symbolu 0 a čím menej 1 ...(ak fungujú predikcie)</u>

Ako je implementovaný bezstratový mód v JPEG2000?

- Požíva sa celočíselný lifting s birtogonálnym waveletom CDF (5,3) (CDF=Cohen-Daubechies-Feauveau)
  - Filtre majú veľkosť 5 a 3
  - Počet nulových momentov DP filtrov je 2 (K-regularita je 2)
- Kvantizačný krok ma veľkosť 1
- Používa sa taktiež EBCOT s artimetickým MQ kóderom
  - kódujú sa všetky bitové roviny

Porovnanie efektivity (približné) – od najhorších k najlepším:

Metóda	**Algoritmus**		**Poznámka**
	Predikcia	**Entropický kóder**	
GIF		LZW	Do 256 farieb
TIFF		LZW	
PNG	Lineárna	LZ77+Huffman	
JPEG	Lineárna	Huffman	Bezstratový mód
FELICS	Max(P1,P2)-min(P1,P2) známe pixely	Golomb-Rice	
JPEG2000	DWT pomocou CDF (5,3)	EBCOT	
JPEG-LS	Mediánová predikcia	Kontextový Golomb-Rice kóder	Metóda LOCO
CALIC	Kontextová lineárna + nelineárna korekcia	Huffman / Aritmetický	m-árny AK s názvom CACM++
FLIF?	MANIAC (Meta-Adaptive Near-zero Arithmetic Coding)		Variant CABACu ...

Poznámky:

- Na porovnanie FLIF verzus VALIC verzus JPEG-LS som zatiaľ nenašiel žiadnu štúdiu ... príležitosť pre študentskú DP?
- Čo že je to ten CABAC ??

Aritmetický kóder v H.264 – CABAC

CABAC=Context Adaptive Binary Arithmetic Coding

- Používa len 2 symboly 0 a 1 (ako binárny AK)
- Využíva, že v H.264 sú predikcie a že pravdepodobnosť symbolu 0 je typicky nad 95%
- Zložitejšie symboly sú „binarizované“
  - Napr. predikčný mód sa binarizuje (8 módov sa prenáša pomocou 3 bitov), každý z týchto 3 bitov sa prenáša ako samostatný symbol
- Používa adaptívne aritmetické kódovanie
  - napr. v predikčnom móde „Coefficient map“ (kódovanie signifikancií)
    - kontext je pozícia pixelu v matici 8x8, t.j. máme 64 kontextov
    - začína s s východzími pravdepodobnosťami, že koeficient má hodnotu 0
    - postupne sa tieto pravdepodobnosti adaptujú aktuálne kódovaným hodnotám
  - predikčný mód „väčší ako 1“ – je veľká pravdepodobnosť, že ak koeficient je nenulový, tak má hodnotu +-1.
  - predikčný mód „absolútna hodnota koeficientu“ – ak koeficient má väčšiu abs. hodnotu ako 1, potom je táto binarizovaná ako exponenciálny Golombov kód. Znamienko nie je kódované v tomto kontexte
  - ...

MANIAC

- použitý vo FLIF (Free Losless Image Format)
- na rozdiel od CABACu sú namiesto viacrozmerných polí kvantizovaných koeficientov (64 kontextov) ako kontexty použité uzly rozhodovacích stromov, ktoré sa vyvíjajú počas kódovania

Čo je to exponenciálny Golombov kód?

- Je to univerzálny kód
  - Efektívne kóduje malé čísla
  - vieme, aké bity ešte patria ešte akému číslu, priradenie je jednoznačné
- Pravidlá kódovania čísla X do Exp. Golombovho kódu:
  - M=počet bitov čísla (X+1)
  - zakódované číslo v binárnej forme = <M-1 krát bit 0><X+1>

Príklady kódovania čísiel

X_{10}	$(X+1)_2$	M-1	<M-1 krát bit 0><X+1>	H.264 –signed X_{10}
0	1	0	1	0
1	10	1	010	1
2	11	1	011	-1
3	100	2	00100	2
4	101	2	00101	-2
5	110	2	00110	3
6	111	2	00111	-3
7	1000	3	0001000	4
8	1001	3	0001001	-4
…	…	…	…	…

Pri H.264 a H.265 je na niektorých miestach použité aj „predmapovanie“, aby bolo možné kódovať aj záporné čísla

- ak $x \leq 0$ potom sa mapuje na $-2x$
- ak $x > 0$ potom sa mapuje na 2x-1

Príklad: Aké sú toto unsigned čísla ak boli zakódované exp. Golombovým kódom? 00100110110110101100010010010101

Odpoveď: 3, 0, 0, 2, 2, 1, 0, 0, 8, 4.

Exp. Golombov kód – pokračovanie

Algoritmus dekódovania (keď nemáme tabuľku):

- Spočítaj počet núl po prvú jednotku. K nasledovnej 1 pridaj toľko bitov, koľko bolo na začiatku núl. Odčítaj 1.
- T.j. ak prijatý prúd bitov 001100110010 1... aké čísla to znamená?
  - 00 -> k 1 pridám 2 bity -> $110_2=6_{10}$, odpočítam 1 -> prvé číslo je 5
  - 0-> k 1 pridám 1 bit -> 11 -> ..... ->druhé číslo je 2
  - 101 -→ tretie číslo je 4
- znamená to čísla (5,2,4)