Отчет По Лабораторной работе 1

Капанин Дмитрий

PPMA

Алгоритм PPMA строится на арифметическом кодере и декодере.
Для кодирования нового символа нужно.
1) Найти для него максимальный контекст.
2) Уменьшать контекст, пока символ не встречался ранее с этим контекстом и кодировать esc для контекста (contexts[ctxHash], ALPHABET_SIZE)
3) Если контекст после уменьшения не пустой, то кодировать его в соответствии с вероятностью в этом контексте (contexts[ctxHash], in[i])
4) Если контекст пустой, то проверить встречался ли ранее символ
5) Если встречался закодировать в соответствии с вероятностью появления в последовательности (context[DEFAULT_CONTEXT_HASH], in[i])
6) Иначе передать esc и закодировать в соответствии с равномерным распределением неиспользованных символов в последовательности (context[DEFAULT_CONTEXT_HASH], ALPHABET_SIZE и context[UNIQUE_CONTEXT_HASH], in[i])

Для декодирования нужно
1) Поддерживать контексты в идентичном состоянии
2) Совершать операции в том же порядке
Я использовал ваш арифметический кодер, видоизмененный под свои нужды. Починил свой кодер и декодер путем удаления логики исключения символов из контекстов больше, теперь все декодится бит в бит но с большими затратами

Формулы

$$\hat{p}_t(a|\#) = \frac{\tau_t(\boldsymbol{a})}{t+1}; \qquad \hat{p}_t(esc|\#) = \frac{1}{t+1}, \tau_t(\boldsymbol{a}) > 0$$

$$\hat{p}_t(a|\boldsymbol{s}) = \frac{\tau_t(\boldsymbol{s}, a)}{\tau_t(\boldsymbol{s})+1}; \quad \hat{p}_t(esc|\boldsymbol{s}) = \frac{1}{\tau_t(\boldsymbol{s})+1}, \tau_t(\boldsymbol{s}, \boldsymbol{a}) > 0$$

Результаты

name	H(X)	H(X|X)	H(X|XX)	Avg	init byte	enc bit	enc byte
bib	5.200676	3.364127	2.307505	2.46961	111 261	290280	36285
book1	4.527149	3.584518	2.814074	2.90718	768 771	2234952	279369
book2	4.792633	3.745216	2.735674	2.58073	610 856	1576456	197057
geo	5.646376	4.264226	3.457736	7.74922	102 400	793520	99190
news	5.189632	4.091893	2.922759	3.29286	377 109	1241768	155221
obj1	5.948171	3.463658	1.400440	6,1224	21 504	131656	16457
obj2	6.260381	3.870373	2.265427	3,59192	246 814	886536	110817
paper1	4.982983	3.646085	2.331768	3,29821	53 161	175336	21917
paper2	4.601435	3.522351	2.513645	3,18155	82 199	261520	32690
paper3	4.665104	3.554845	2.559874	3,59696	46 526	167352	20919
paper4	4.699726	3.477308	2.205155	4,03854	13 286	53656	6707
paper5	4.936154	3.526005	2.041391	4,15125	11 954	49624	6203
paper6	5.009503	3.611169	2.250973	3,4154	38 105	130144	16268
pic	1.210176	0.823654	0.705194	1,04722	513 216	537448	67181
progc	5.199016	3.603391	2.134001	3,41663	39 611	135336	16917
progl	4.770085	3.211607	2.043555	2,37992	71 646	170512	21314
progp	4.868772	3.187547	1.755124	2,40685	49 379	118848	14856
trans	5.532781	3.354794	1.930493	2,07021	93 695	193968	24246
TOTAL				**3,285928**	**3 251 493**	**9148912**	**1143614**

encoder args пример
```
encoder
--input
../test_data/test.txt
--output
../test_data/test-enc-out.txt
```

decoder args пример
```
decoder
--input
../test_data/test-enc-out.txt
--output
../test_data/test-dec-out.txt
```