1. Encode the numbers 100, 200, and 400 as:

   (a) Elias-γ codes;

   (b) VByte codes.

a) Elias-gamma codes represent an integer n>0 as a pair of selector and offset.

- *Offset: the integer in binary, with the first digit cut off*
- *Selector: length of the offset (nr of digits in the offset) written in Unary = that same amount in zeros, followed by a 1.*
- *Write a new number (elias-gamma code) which is the selector followed by the offset*

100:

- Offset Bin(100) = 1100100 ; cutting off 1st digit = 100100
- Selector nr of digits in offset = 6 ; Unary(6) = 0000001
- Elias-gamma(100) = 0000001100100

200:

- Offset Bin(200) = 11000100 ; cutting off 1st digit = 1000100
- Selector nr of digits in offset = 7 ; Unary(7) = 00000001
- Elias-gamma(200) = 000000011000100

400:

- Offset Bin(400) = 110001000 ; cutting off 1st digit = 10001000
- Selector nr of digits in offset = 8 ; Unary(8) = 000000001
- Elias-gamma(400) = 00000000110001000

b) For computing VB code:

- *Convert given number to binary*
- *From the least significant bit (from the right) take 7 bits; it will be your last Byte and a "1" will be added to the left of that Byte*
- *Repeat from the remaining of the number: take 7 bits (if available) and add a "1" to the left. If less than 7 digits remaining, then take them and put zeros to complete the Byte (total 8 digits).*
- *Write the numbers in the order that you processed them.*

100:

- Bin(100) = 1100100
- VB code (100) = **1**1100100

200:

- Bin(200) = 11000100
- VB code (299 = **1**1000100**0**0000011

400:

- Bin(400) = 110001000
- VB code (400) = **1**0001000**0**0000011

2. Are the following statements true or false?

   (a) In a Boolean retrieval system, stemming never lowers precision.

   (b) In a Boolean retrieval system, stemming never lowers recall.

   (c) Stemming increases the size of the vocabulary.

   (d) Stemming should be invoked at indexing time, but not while processing a query.

a) **FALSE**. Precision is the fraction of query results that are relevant (= Number of pages that were retrieved and relevant / Total number of retrieved pages). Two words that are completely unrelated can be stemmed to the same base and, so, querying one of those words will also give us all the non-relevant results where the same-stem unrelated word is found. This then increases the amount of non-relevant results and decreases the precision.

b) **TRUE**. Recall = Number of pages that were retrieved and relevant / Total number of relevant pages. No relevant pages are "lost" with stemming, so the Recall should remain the same.

c) **TRUE**. A lot more terms can be found with one stem. (hopefully most of them related to what we are actually querying)

d) **FALSE**. Stemming at indexing time should be done because it reduces the number of tokens. When processing a query the query needs to be stemmed as well since otherwise there will be no matching token (since they already got stemmed).

3. The following pairs of words are stemmed to the same form by the Porter stemmer. Which pairs would you argue should not be made the same. Give your reasoning.

   (a) abandon/abandonment

   (b) absorbency/absorbent

   (c) marketing/markets

   (d) university/universe

   (e) volume/volumes

Should not be made the same:

d) University/universe – these are two very different concepts. The only point of intersection is that you might study the Universe at a University, but for that line of studies you would use a different query. The query results from "Universe" would not be relevant for someone wanting to search for "University" and vice-versa.

4. In the context of a web crawler, what is the URL-seen test? Describe an efficient data structure for performing the URL-seen test and describe how it is used.

"URL seen test" is a data structure that maintains all the URLs seen so far.

This is how a "URL seen test" is used:

Web crawlers perform discovery incrementally and continuously and, in each iteration:

- A new target URL is selected from the URLs available in the Frontier
- URL goes to high-level FS for download
- Returned HTML content goes to Repository Manager, which decides whether the pages enters the Web repository
- Content then goes to the parser where the URLS in the page are extracted and normalized (lowercase, www, etc)
- Normalized URLs are looked up the in a datastructure that maintains all the URLs seen so far: seen-test
- Previously unseen URLs are added to that data structure, expanding the crawler's Frontier

5. Why is it better to partition hosts (rather than individual URLs) between the nodes of a distributed crawl system?

Often URLs of a host link to other URLs at the same host. We would risk showering the host with fetch requests if we partitioned by URL instead of host.

6. Consider the following collection of documents:
   Page 1: The smallest planet is Mercury.
   Page 2: Jupiter, the largest planet!
   Page 3: Neptune is smaller than Uranus, but is heavier.
   Page 4: Uranus is smaller than Saturn.
   Page 5: Neptune is heavier than Uranus.
   Page 6: Jupiter > Saturn > Uranus > Neptune

   (a) Draw the inverted index for the collection after normalizing by applying case folding and removing punctuation (normalization you like, e.g., stemming). Include document frequencies (the number of times each normalized term occurs in the collection); term frequencies (the number of times each document contains the term); and also term positions. See Slide 5 of Lecture 4 for a guide (you can ignore the "Weights" part shown on that slide).

(see next page)

terms		postings				doc freq.	doc ID				within doc frequency (position counts)				term position				
but	->	3				1	3				1				6				
heav	->	3	5			2	3	5			1	1			8	3			
is	->	1	3	4	5	4	1	3	4	5	1	2	1	1	4	2	7	2	2
jupiter	->	2	6			2	2	6			1	1			1	1			
largest	->	2				1	2				1				3				
mercury	->	1				1	1				1				5				
neptune	->	3	5	6		3	3	5	6		1	1	1		1	1	4		
planet	->	1	2			2	1	2			1	1			3	4			
saturn	->	4	6			2	4	6			1	1			5	2			
small	->	1	3	4		3	1	3	4		1	1	1		2	3	3		
than	->	3	4	5		3	3	4	5		1	1	1		4	4	4		
the	->	1	2			2	1	2			1	1			1	2			
uranus	->	3	4	5	6	4	3	4	5	6	1	1	1	1	4	1	5	3	

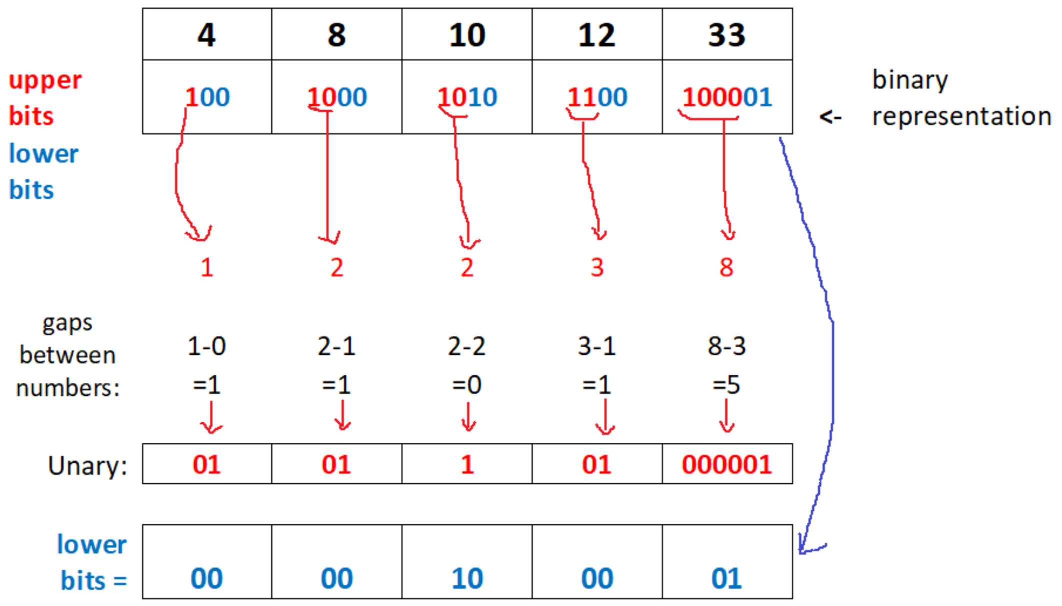
7. Give an overview of the Elias-Fano encoding scheme and illustrate it for the example set $\{4, 8, 10, 12, 33\}$. Describe how the ith integer can be extracted from the encoding.

- Start point: ordered list of positive integers (postings list, etc). In our example {4, 8, 10, 12, 33}
- each integer is first converted to binary representation.
- There are n elements in the list and the maximum is Z. Here we have Z=33 and n=5.
- Each binary representation of the elements is split in two:
  - the higher part consisting of the first (left to right) [log n] bits and
  - the lower part with the remaining [log Z − log n] = [log Z / n] .

The concatenation of the lower part of each element of the list is the actual stored representation and takes n log Z bits.

The higher part is a unary representation (sequence of uncary-coded gaps), specifically a bit-vector of size $n + m / 2^{[\log m / n]} = 2n$ bits.

- the Elias-Fano representation is the bitvector resulting from the concatenation of the higher and the lower part



4
8
10
12
33
upper bits
lower bits
100
1000
1010
1100
100001
<-
binary representation
1
2
2
3
8
gaps between numbers:
1-0 =1
2-1 =1
2-2 =0
3-1 =1
8-3 =5
Unary:
01
01
1
01
000001
lower bits =
00
00
10
00
01