CS533 Homework - 6[1]

Solutions

1) For signature files, every document is represented with a term signature. Moreover, for each document, there are n bits. For our case each document 256bits and number of documents is 40000.

a) Sequential signature size = *Number of documents x sig of term*

= 256 x 40000 = 10240000bits

If we convert to bytes = 10240000 x 1/8 = 1280000bytes

b) For bit-sliced signatures, same situation; size = *Num of docs x sig of term*
= 256 x 40000 = 10240000 bits = 1280000bytes

2) We have 40000 documents
Signature size of object = 256 bits
Page size 0.5K = 2^{12} bits

a) Sequential signature -> place signatures one after the other

Pages = 40000 x 256 / 2^{12} = 4096 pages

b) Bit-sliced -> place bit slices one after the other

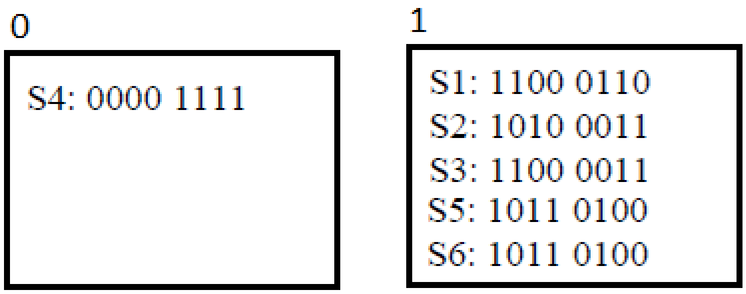
Pages = 4000 / 2^{12} = 9.765625 ≈ 10 pages for each bit
Total = 10 x 5 = 50 pages (For 1, 2, 50, 51, 60)

3) Signatures
S1: 1100 0110
S2: 1010 0011
S3: 1100 0011
S4: 0000 1111
S5: 1011 0100
S6: 1011 0100

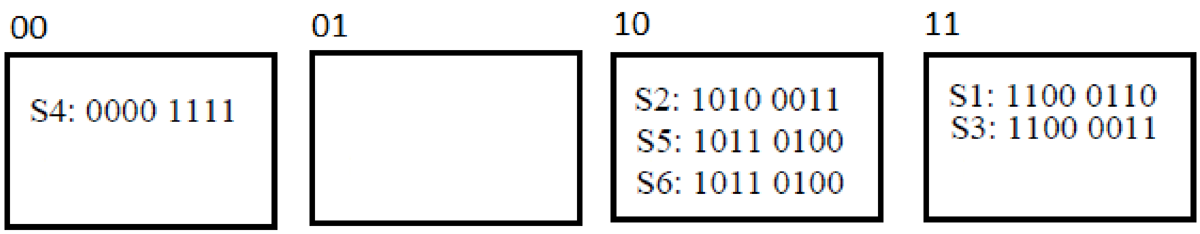
a) Fixed prefix method

[1] Solutions are due to Emir Gülümser.

For k = 1

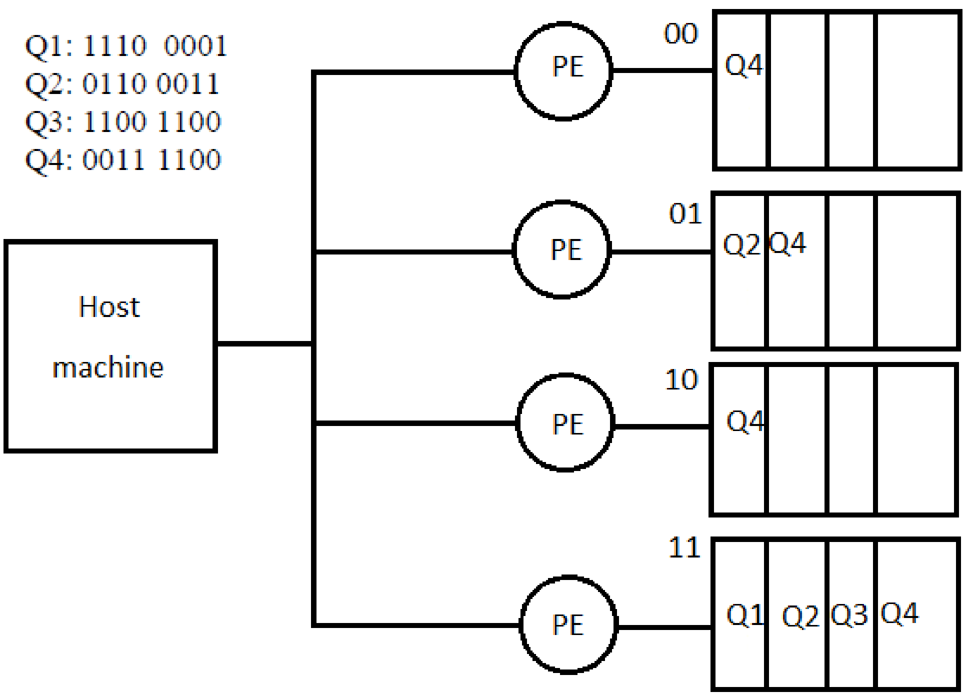


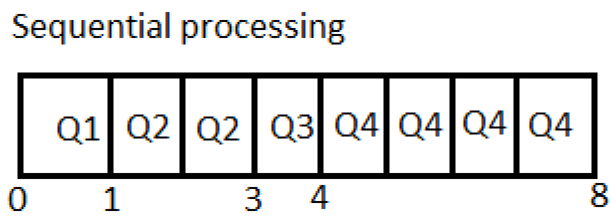
For k = 2



b) Calculation time (assumption processing of one page = 1 time unit)

Parallel processing





<u>Execution times:</u>

Q1 = 1 TU, Q2 = 2TU, Q3 = 3TU, Q4 = 4TU

Total turnaround (sequential) = 1 + 3 + 4 + 8 = 16

ATT (sequential) = 16 /4 = 4

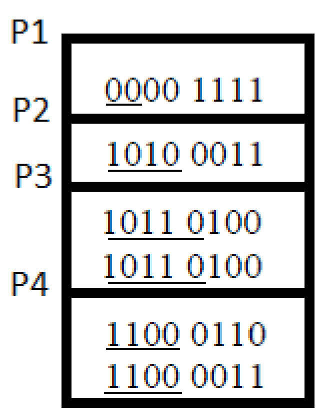
Total turnaround (parallel) = 1 + 2 + 3 + 4 = 10

ATT (parallel) = 10 / 4 = 2.5

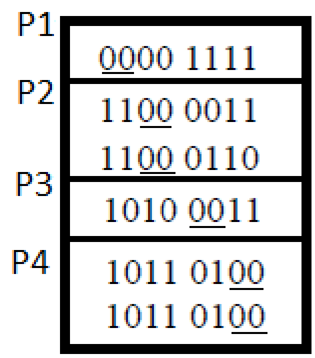
Speed-up Ratio = S-ATT / P-ATT = 4 / 2.5 = 1.6

4)

a) For Extended prefix partitioning z = 2, that means a signature must contain at least 2 zeros.



b) For Floating key partitioning k = 2, we select the 2-substring which contains the least number of ones. Specifically, each of the consecutive nonoverlapping 2-substrings o the signature is examined from left to right, and the leftmost substring with the smallest weight is chosen as the key [1].



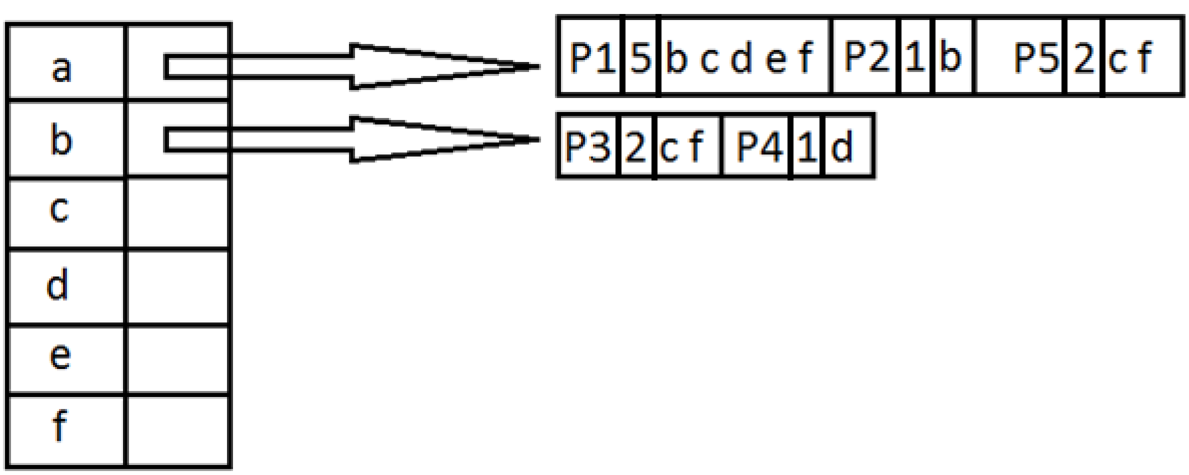
c) EPP:

Q1: 1110 0001 -> No page, because there is no match for key of signatures.
Q2: 0110 0011 -> No page, because there is no match for key of signatures.
Q3: 1100 1100 -> P4, match with key 1100
Q4: 0011 1100 -> P1, match with key 00

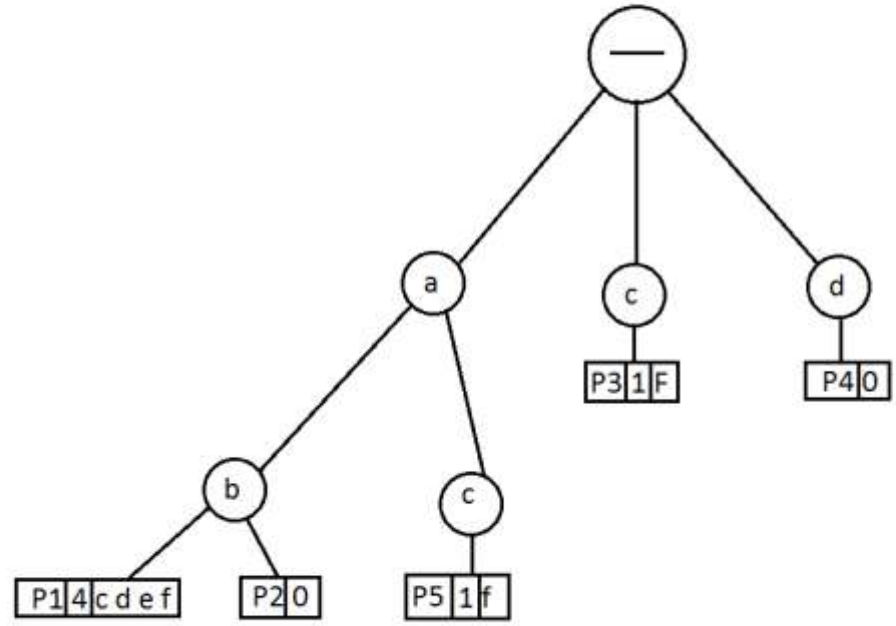
FKP

Q1: 1110 0001 -> P3, match with key XXXX00XX
Q2: 0110 0011 -> P3, match with key XXXX00XX
Q3: 1100 1100 -> P2 and P4, match with keys: XX00XXXX and XXXXXX00
Q4: 0011 1100 -> P1 and P4, match with keys: 00XXXXXX and XXXXXX00

5) For ranked key method, we store the profile in the list of the word with the lowest rank. Below showing the directory and the posting lists.



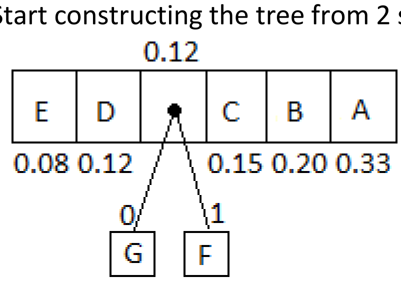
Below shows the directory and posting lists for tree method



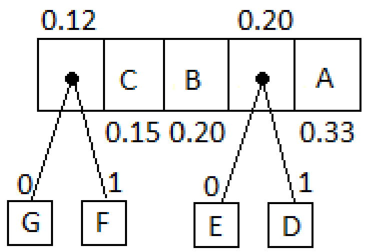
6) First sort the probabilities according to frequency

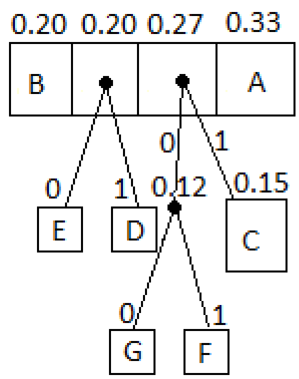
G	F	E	D	C	B	A
0.04	0.08	0.08	0.12	0.15	0.20	0.33

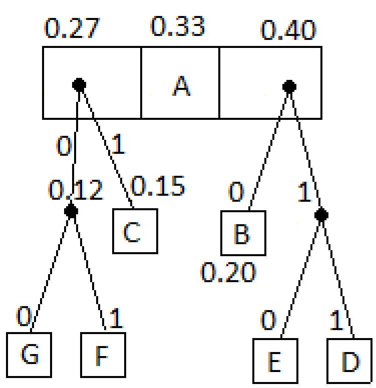
Start constructing the tree from 2 smallest probability elements

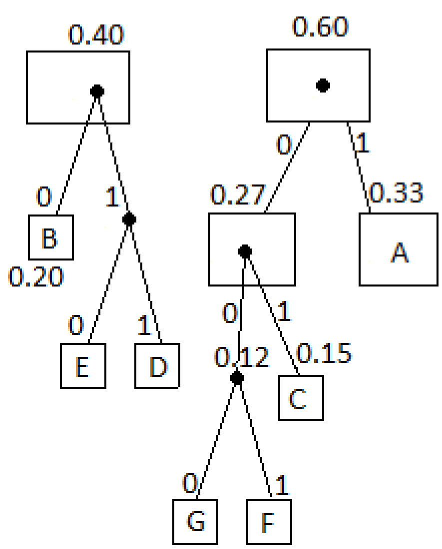


You repeat until there is only one element left in the list

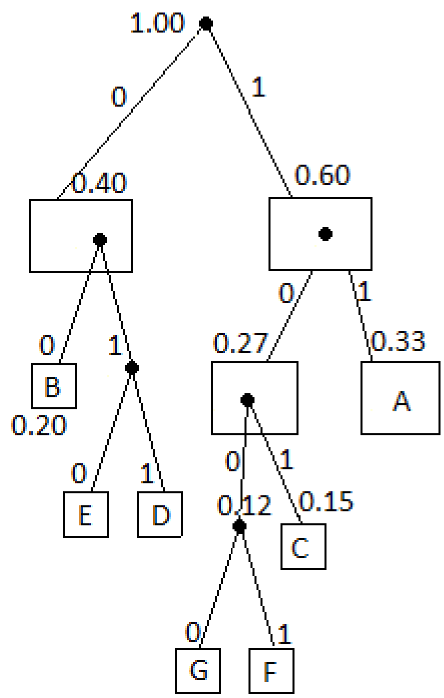

0.12
0.20
C
B
A
0.15 0.20
0.33
0
1
G
F
E
D


0.20 0.20 0.27 0.33
B
A
0
1
0.12
0.15
E
D
C
G
F


0.27
0.33
0.40
A
0
1
0.12
0.15
C
B
0.20
G
F
E
D


0.40
0.60
0
1
0.27
0.33
B
0.20
A
E
D
0.12
0.15
C
G
F

And finally tree is shown below



Code results from Huffman Tree:

B: 00
E: 010
D: 011
G: 1000
F: 1001
C: 101
A: 11

7) Gamma Code: To use gamma decoding, we must store the number in two parts [2]:

- First part: Encode integer ***1 + floor (log n)*** as 1s followed by a 0.
- Second part: $n - 2^{floor(log\ n)}$ in binary (using ***floor (log n)*** bits)

7 ->

- floor ($\log_2 7$) = 2 + 1 = 3 -> 110 first part
- 7 - 4 = 3 using 2 bits -> 11
- In gamma code -> 11011

17 ->

- floor ($\log_2 17$) = 4 + 1 = 5 -> 11110 first part
- 17 - 16 = 1 using 4 bits -> 0001
- In gamma code -> 111100001

23->

- floor ($\log_2 23$) = 4 + 1 = 5 -> 11110 first part

- 23 - 16 = 7 using 4 bits -> 0111
- In gamma code -> 111100111

50->

- floor ($\log_2$ 50) = 5 + 1 = 6 -> 111110 first part
- 50 - 32 = 18 using 5 bits -> 10010
- In gamma code -> 11111010010

Delta Code: To use delta coding, we must store the number in two parts [2]:

- First part: Encode integer as ***1 + floor (log n)*** in Gamma Code
- Second part: ***$n - 2^{floor(\log n)}$*** in binary (using ***floor (log n)*** bits)

7->

- floor ($\log_2$ 7) = 2 + 1 = 3 in Gamma code
  - floor ($\log_2$ 3) = 1 + 1 = 2 -> 10 first part
  - 3 - 2 = 1 using 1 bits -> 1
  - In gamma code -> 101
- 7 - 4 = 3 using 2 bits -> 11
- In Delta Code -> 10111

17->

- floor ($\log_2$ 17) = 4 + 1 = 5 in Gamma code
  - floor ($\log_2$ 5) = 2 + 1 = 3 -> 110 first part
  - 5 - 4 = 1 using 2 bits -> 01
  - In gamma code -> 11001
- 17 - 16 = 1 using 4 bits -> 0001
- In Delta Code -> 110010001

23->

- floor ($\log_2$ 23) = 4 + 1 = 5 in Gamma code
  - floor ($\log_2$ 5) = 2 + 1 = 3 -> 110 first part
  - 5 - 4 = 1 using 2 bits -> 01
  - In gamma code -> 11001
- 23 - 16 = 7 using 4 bits -> 0111
- In Delta Code -> 110010111

50->

- floor ($\log_2$ 50) = 5 + 1 = 6 in Gamma code
  - floor ($\log_2$ 6) = 2 + 1 = 3 -> 110 first part
  - 6 - 4 = 2 using 2 bits -> 10
  - In gamma code -> 11010
- 50 - 32 = 18 using 5 bits -> 10010
- In Delta Code -> 1101010010

Huffman code provides optimal compression, and usually gives better compression than universal coding (delta and gamma codes) paradigms. However, people need universal codes when Huffman coding cannot be used or creates an overhead. For example, universal codes are useful (when we don't know the exact probabilities Huffman code cannot be used) when exact probabilities are not known, and we only know the ranking of their probabilities. Moreover, receiver and transmitter example; only one side knows the probabilities other side tries to transmit the probabilities to the not known side, which creates an overhead. That overhead is not available and required for universal coding [4].

Result of the generated delta code is longer than the gamma code for the values where n < 15, however for the rest delta generated values are never worse than gamma code [3]. For example for the value 8, gamma code produces 1110000 (7 bits), and delta code produces 11000000 (8 bits).

References

[1] Dik Lun Lee and Chun-Wu Leng. 1989. Partitioned signature files: design issues and performance evaluation. *ACM Trans. Inf. Syst.* 7, 2 (April 1989), 158-180. DOI=10.1145/65935.65937 http://doi.acm.org/10.1145/65935.65937

[2] Online, Classnotes. http://www.csee.umbc.edu/~ian/irF02/lectures/05Compression-for-IR.pdf

[3] Justin Zobel, Alistair Moffat, Ron Sacks-Davis: Searching Large Lexicons for Partially Specified Terms using Compressed Inverted Files. VLDB 1993: 290-301

[4] Online, http://en.wikipedia.org/wiki/Universal_code_%28data_compression%29