Assignment 3 Solutions

1.

a. Sequential signature method assigns 512 bits to all objects. Therefore, the file size becomes:

$$filesize = NxF = 60{,}000x512\ bits = 30{,}720{,}000\ bits = 3750\ kbyte$$

In this equation, *N* represents the number of objects and *F* represents signature size.

b. In bit-sliced signature method, each memory element holds the bits of the associated columns of object signatures. Therefore, there are F number of elements (as it is the total number of columns) where each element is N bits long (as there are N objects). The file size therefore becomes:

$$filesize = FxN = 512\ bitsx60{,}000 = 30{,}720{,}000\ bits = 3750\ kbyte$$

which is the same result as previous method.

2.

We have:

- A query with 1s in 5 bit positions (1, 2, 50, 51, 60) and 0s otherwise
- 512 bit signatures
- 60,000 objects
- Page size of 0.5 kbyte

In SS, as each object signature is 512 bits, there are $\frac{0.5\ kbyte}{512\ bit} = \frac{0.5x8x1024\ bit}{512\ bit} = 8$ signatures per page, which results in $\frac{60{,}000\ objects}{8} = 7500$ pages.

As long as the page signatures comply with the query signatures, the page must be accessed. As the query is mostly 0, a high portion of the pages will be accessed. In the worst case, all 7500 will be accessed.

In BS, there are $\frac{512\ memory\ objects}{8} = 64$ pages. As there are 5 nonzero bit positions, 5 of these pages will be accessed for further operation.

3.

a.

00	**01**	**10**	**11**
S4		S2	S1
		S5	S3
		S6	S7

b.

$turnaround\ time = time\ of\ completion - time\ of\ arrival$

where the processing time of each query is assumed to be 1 tu (time unit).

Sequential: (all queries are assumed to arrive at time 0)

Q1: time of arrival = 0tu, time of completion = 1tu => tat = 1tu

Q2: time of arrival = 0tu, time of completion = 3tu => tat = 3tu

Q3: time of arrival = 0tu, time of completion = 4tu => tat = 4tu

Q4: time of arrival = 0tu, time of completion = 8tu => tat = 8tu

$$average\ turnaround\ time = \frac{1+3+4+8}{4} = 4.00tu$$

Parallel: (each prefix is handled in a different processor)

Q1: time of arrival = 0tu, time of completion = 1tu => tat = 1tu

Q2: time of arrival = 0tu, time of completion = 2tu => tat = 2tu

Q3: time of arrival = 0tu, time of completion = 3tu => tat = 3tu

Q4: time of arrival = 0tu, time of completion = 4tu => tat =4tu

$$average\ turnaround\ time = \frac{1+2+3+4}{4} = 2.50tu$$

$$speed\ up\ ratio = \frac{4.00}{2.50} = 1.60$$

4.

a. In EPP (extended prefix partitioning) with z = 2, all signatures are interchanged with keys that include 2 zeros, from the beginning. As a result:

S1: 1100

S2: 1010

S3: 1100

S4: 00

S5: 1010

S6: 10110

S7: 1100

The resulting partition is as follows:

00	**1010**	**1100**	**10110**
S4	S2	S1	S6
	S5	S3	
		S7	

b. In FKP (floating key partitioning) with k = 2, each signature is reduced to length 2. This reduced signature is selected to start anywhere that includes minimum number of 1s. Therefore the signatures become (numbers include the starting position of the key & resulting key) (in ties, leftmost possible keys are selected):

S1: 3 - 00

S2: 5 - 00

S3: 3 - 00

S4: 1 - 00

S5: 1 – 10

S6: 7 – 00

S7: 3 – 00

And the resulting partition:

1 - 00	3 - 00	5 - 00	7 - 00	1 - 10
S4	S1	S2	S6	S5
	S3			
	S7			

c. Q1 becomes 11100 in EPP and 5 – 00 in FKP (notice the separation). No page signatures comply with 11100 in EPP, so no pages are accessed. In FKP, three pages are accessed (5 – 00, 7 – 00, 1 - 10). As these pages do not contain Q1, they can be labelled as false matches. Therefore, for this operation, it can be stated that EPP is more efficient.

Note that the pages that are accessed are selected by applying the logic 'AND' operation between the query signature and the page signature. If the result equals to the query signature, the page is accessed.

5.

Extendible hashing starts with 1 prefix bits. As overflows occur, either table's depth is increased by one or data blocks are split under following conditions; if the overflowing data block's height is equal to table's height, the table is doubled (height is increased by one). If it is smaller compared to the table, the data block is split in higher height. The resulting steps therefore are as follows:

Initially:

1

0				1
1	S1	S2	S3	1

Overflow in “1”, table is doubled.

00	S4		
01			
10	S2	S5	S6
11	S1	S3	

Overflow in “10”, table is doubled.

000	S4		
001			
010			
011			
100			
101	S2	S5	S6
110	S1	S3	
111			

Overflow in “101”, table is doubled.

0000	S4		
0001			
0010			
0011			
0100			
0101			
0110			
0111			
1000			
1001			
1010	S2	S5	
1011	S6		
1100	S1	S3	S7
1101			
1110			
1111			

Overflow in “11”, block is split to a higher height.

0000	S4		
0001			
0010			
0011			
0100			
0101			
0110			
0111			
1000			
1001			
1010	S2	S5	
1011	S6		
1100	S1	S3	S7
1101			
1110			
1111			

Overflow in “110”, block is split to a higher height.

0000	S4		
0001			
0010			
0011			
0100			
0101			
0110			
0111			
1000			
1001			
1010	S2	S5	
1011	S6		
1100	S1	S3	S7
1101			
1110			
1111			

Overflow in “1100”, table is doubled.

00000	S4		
00001			
00010			
00011			
00100			
00101			

00110			
00111			
01000			
01001			
01010			
01011			
01100			
01101			
01110			
01111			
10000			
10001			
10010			
10011			
10100	S2	S5	
10101			
10110	S6		
10111			
11000	S1	S3	
11001	S7		
11010			
11011			
11100			
11101			
11110			

11111			

No overflows – operation complete.

To determine which pages will be accessed, we must determine page signatures that produce query signatures when logic “AND” operation is applied, with also considering block lengths. For Q1, “111” page is accessed. For Q2, again “111” page will be accessed.

6.

We have Bkfr = 2 and LF = 0.5. In representation, first column indicates the suffix, second and third column indicate prime data areas and the last column indicates the overflow locations. Current boundary value (bv) and current load (Lf) will be indicated at each step.

Initially (bv starts from 0):

0	S1		
1	S2	S3	

Now, $Lf = \frac{3}{4} = 0.75$ and LF is surpassed. As bv = 0, “0” registry or row will be split.

00			
1	S2	S3	
10	S1		

bv became 1. $Lf = \frac{3}{6} = 0.5$ so there is no need to split. S4 is added.

00			

1	S2	S3	S4
10	S1		

and S4 became overflow. Now, $Lf = \frac{4}{6} = 0.67$ and a split is needed. As bv = 1, "1" registry will be split.

00			
01			
10	S1		
11	S2	S3	S4

S4 became overflow again. As h = 4 (height of the directory), bv becomes 0 again. , $Lf = \frac{4}{8} = 0.50$ so no split is needed, another document is added.

00			
01			
10	S1	S5	
11	S2	S3	S4

Now, $Lf = \frac{5}{8} = 0.63$ so "00" registry will be split.

000			
01			
10	S1	S5	
11	S2	S3	S4

100			

and bv becomes 1. Now, another document is added as $Lf = \frac{5}{10} = 0.50$.

000			
01			
10	S1	S5	
11	S2	S3	S4
100	S6		

Now, $Lf = \frac{6}{10} = 0.60$ so "01" registry will be split & another document will be added.

000			
001			
10	S1	S5	S7
11	S2	S3	S4
100	S6		
101			

Now, bv = 2, S7 became overflow and $Lf = \frac{7}{12} = 0.59$. "01" registry will be split.

000			
001			
010	S7		
11	S2	S3	S4

100	S6		
101			
110	S1	S5	

and this finalizes the operation. Finally, bv = 3, $Lf = \frac{7}{14} = 0.50$ and h = 7. Also, S4 is in an overflow location.

For queries Q1 & Q2, we must look at page signatures that comply with their query signatures. Therefore, for Q1, "11" is accessed and for Q2 "010", "11" and "110" pages will be accessed.

As both Q1 and Q2 are not present in document list, recall rate is automatically 0 for both of them. As a direct result, false drop resolution does not change recall rates.

Lastly, the false drop resolution is the operation after obtaining a block match, to check whether the query signature is included in the contents of the block. If not checked, one can assume that the recall rate is higher by assuming that the block contains related documents. However, false drop resolution can find that there are no related documents, so the recall rate is actually smaller.

7.

The formula for calculating false drop probability of a Bloom Filter:

$$p = (1 - (1 - \frac{1}{m})^{kn})^k = (1 - (1 - \frac{1}{10})^{2x3})^2 = 0.220$$

Finally, (theoretical) total cost of making 10,000 lookups is:

$$0.220x10{,}000x1ms + (1 - 0.220)x10{,}000x0.5ms = 6100ms = 6.1sec$$

8.

Reciprocal rank:

1. Rank of each document is calculated by the formula

$$r(d_i) = \frac{1}{\sum_{systems}(1/position(d_{ij}))}$$

r(a) = 0.353

r(b) = 0.600

r(c) = 0.333

r(d) = 4.000

r(e) = 4.000

r(f) = 4.000

2. When these values are sorted in ascending order, we obtain the rank in descending order:

Rank(c) > Rank(a) > Rank(b) > Rank(d) = Rank(e) = Rank(f)

Borda count:

1. Borda value of each document is calculated by assigning votes for each retrieval system, depending on their position.

$BC(a) = BC_A(a) + BC_B(a) + BC_C(a) + BC_D(a) = 5 + 4 + 6 + 6 = 21$

$BC(b) = BC_A(b) + BC_B(b) + BC_C(b) + BC_D(b) = 4 + 5 + 4 + 5 = 18$

$BC(c) = BC_A(c) + BC_B(c) + BC_C(c) + BC_D(c) = 6 + 6 + 5 + 5 = 22$

$BC(d) = BC_A(d) = 3$

$BC(e) = BC_B(e) = 3$

$BC(f) = BC_C(f) = 3$

2. Sorting these values on descending order gives the rank order of the documents.

Rank(c) > Rank(a) > Rank(b) > Rank(d) = Rank(e) = Rank(f)

Condorcet:

1. In this method, we must first construct the pairwise competition matrix (win, lose, tie).

	a	b	c	d	e	f
a	-	3, 1, 0	2, 2, 0	1, 0, 0	1, 0, 0	1, 0, 0
b	1, 3, 0	-	0, 3, 1	1, 0, 0	1, 0, 0	1, 0, 0
c	2, 2, 0	3, 0, 1	-	1, 0, 0	1, 0, 0	1, 0, 0
d	0, 1, 0	0, 1, 0	0, 1, 0	-	0, 0, 0	0, 0, 0
e	0, 1, 0	0, 1, 0	0, 1, 0	0, 0, 0	-	0, 0, 0
f	0, 1, 0	0, 1, 0	0, 1, 0	0, 0, 0	0, 0, 0	-

2. Next, we must determine the total number of wins and losses for each document.

a: 4 wins, 0 losses

b: 3 wins, 2 losses

c: 4 wins, 0 losses

d: 0 wins, 3 losses

e: 0 wins, 3 losses

f: 0 wins, 3 losses

3. Finally, the ranking is based on win & loss counts.

Rank(a) = Rank(c) > Rank(b) > Rank(d) = Rank(e) = Rank(f)

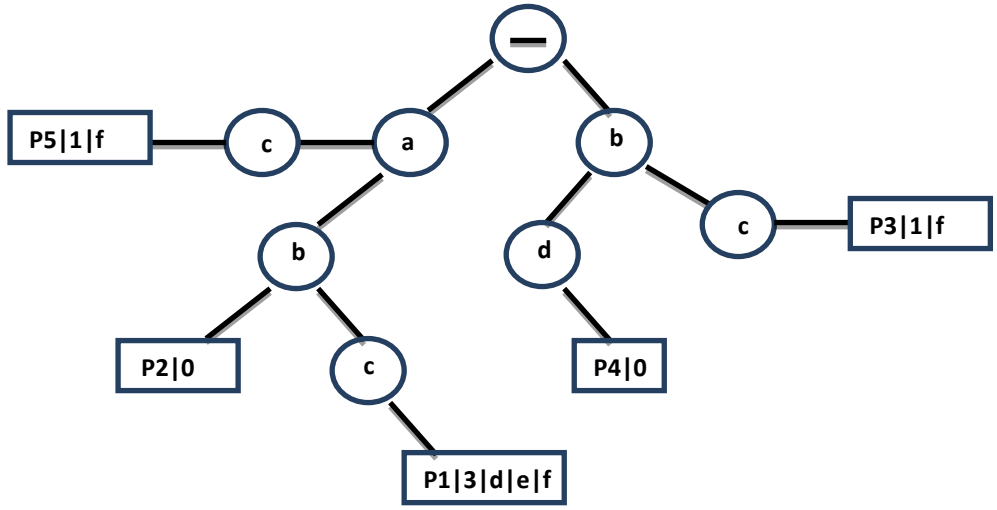
9.

Ranked key method:

a	P1|5|b|c|d|e|f||P2|1|b||P5|2|c|f
b	P3|2|c|f||P4|1|d

c	
d	
e	
f	

Tree method:



10.

The formula to be used here is (taken from the source paper):

$$\pi_j = \sum_i \pi_i h_{i,j}$$

Here, π denotes the page rank score of a page and h denote the connectivity matrix, which is formed by taking the inverse of number outgoing links from that page, if a link exists from i to j. Else, it is 0. Therefore, we need to form the h matrix:

	A	B	C	D	E	F	G	H	I	L	M
A	0	0	0	0	0	0	0	0	0	0	0
B	0	0	1.000	0	0	0	0	0	0	0	0
C	0	1.000	0	0	0	0	0	0	0	0	0
D	0.500	0.500	0	0	0	0	0	0	0	0	0
E	0	0.333	0	0.333	0	0.333	0	0	0	0	0
F	0	0.500	0	0	0.500	0	0	0	0	0	0
G	0	0.500	0	0	0.500	0	0	0	0	0	0
H	0	0.500	0	0	0.500	0	0	0	0	0	0
I	0	0.500	0	0	0.500	0	0	0	0	0	0
L	0	0	0	0	1.000	0	0	0	0	0	0
M	0	0	0	0	1.000	0	0	0	0	0	0

Normally, the definition is recursive as no page rank values are known. However, as other pages have their values calculated, page rank values of pages A and F can be calculated easily:

$\pi_A = 0.500x3.9 = 1.95$

$\pi_F = 0.333x8.1 = 2.697$

(Of course, the reason for these numbers being different than indicated numbers under nodes is that they were normalized.)

11.

Before passing to individual cases, we must form the similarity matrix between documents (using Dice coefficient):

1.00	0.67	0.50	0.00
-	1.00	0.80	0.00
-	-	1.00	0.00
-	-	-	1.00

Also, we must calculate the similarity between Q (query) and documents.

0.00	0.33	0.25	0.00

a.

As λ = 1.00, we choose the documents that are most similar with the query and those are: d2 and d3. We can use the similarity matrix (& similarity coefficients) to state the similarity or diversity between documents. In this case, as we only considered maximum similarity between the query and documents, retrieved documents are inevitably pretty similar (with similarity value of 0.80 out of 1.00). The usage in this case therefore limits MMR and the results are not diversified & very close to each other.

b.

In λ = 0.00, we select the documents that are most distant from each other. As all similarity values are 0.00 with d4, we can return any document (say d1) with d4. In this case, returned results are diverse, but possibly unrelated with the query so MMR algorithm did not work well again.

c.

By selecting λ as 0.50, we are now able to use all features of MMR. We first select the most similar document to the query (d2). Then, we calculate the MMR values for other documents, by also considering d2:

$$MMR(d1) = 0.5x0.00 - (1 - 0.5)x0.67 = -0.335$$

$$MMR(d3) = 0.5x0.25 - (1 - 0.5)x0.80 = -0.275$$

$$MMR(d4) = 0.5x0.00 - (1 - 0.5)x0.00 = 0.000$$

Here, we select the maximum MMR value, which is d4. Now, MMR algorithm returned both related and diverse documents, so it can be stated that the algorithm is beneficial.