Assignment 3

Alan Ou
301320642

Q2:

The operations in the sub-matrices are as follows:

Replace	Insert
Delete	Minimum of three operations

Matrix when insertion, deletion, and replacement operations all carry the same weight

		f	a	t	b	r	o	o	k
	0	1 1	2 2	3 3	4 4	5 5	6 6	7 7	8 8
f	1	0 2	2 3	3 4	4 5	5 6	6 7	7 8	8 9
	1	2 0	1 1	2 2	3 3	4 4	5 5	6 6	7 7
a	2	2 1	0 2	2 3	3 4	4 5	5 6	6 7	7 8
	2	3 1	2 0	1 1	2 2	3 3	4 4	5 5	6 6
c	3	3 2	2 1	1 2	2 3	3 4	4 5	5 6	6 7
	3	4 2	3 1	2 1	2 2	3 3	4 4	5 5	6 6
e	4	4 3	3 2	2 2	2 3	3 4	4 5	5 6	6 7
	4	5 3	4 2	3 2	3 2	3 3	4 4	5 5	6 6
b	5	5 4	4 3	3 3	2 3	3 4	4 5	5 6	6 7
	5	6 4	5 3	4 3	4 2	3 3	4 4	5 5	6 6
o	6	6 5	5 4	4 4	4 3	3 4	3 5	4 6	6 7
	6	7 5	6 4	5 4	5 3	4 3	4 3	4 4	5 5
o	7	7 6	6 5	5 5	5 4	4 4	3 4	3 5	5 6
	7	8 6	7 5	6 5	6 4	5 4	5 3	4 3	4 4
k	8	8 7	7 6	6 6	6 5	5 5	5 4	4 4	3 5
	8	9 7	8 6	7 6	7 5	6 5	6 4	5 4	5 3

Operations:

To transform fatbrook into facebook

1. Replace one character (the character t with c) in fatbrook

f	a	c	e	b	o	o	k
f	a	c	b	r	o	o	k

2. Replace one additional character (the character b with e) in fatbrook

f	a	c	e	b	o	o	k
f	a	c	e	r	o	o	k

Assignment 3

Alan Ou
301320642

3. Replace one additional character (the character r with b) in fatbrook

f	a	c	e	b	o	o	k
f	a	c	e	b	o	o	k

Matrix when insertion and deletion operations carry weight 1 each, and replacement operation carries weight 2

		f	a	t	b	r	o	o	k
	0	1 1	2 2	3 3	4 4	5 5	6 6	7 7	8 8
f	1	0 2	3 3	4 4	5 5	6 6	7 7	8 8	9 9
	1	2 0	1 1	2 2	3 3	4 4	5 5	6 6	7 7
a	2	3 1	0 2	3 3	4 4	5 5	6 6	7 7	8 8
	2	3 1	2 0	1 1	2 2	3 3	4 4	5 5	6 6
c	3	4 2	3 1	2 2	3 3	4 4	5 5	6 6	7 7
	3	4 2	3 1	2 2	3 3	4 4	5 5	6 6	7 7
e	4	5 3	4 2	3 3	4 4	5 5	6 6	7 7	8 8
	4	5 3	4 2	3 3	4 4	5 5	6 6	7 7	8 8
b	5	6 4	5 3	4 4	3 5	6 6	7 7	8 8	9 9
	5	6 4	5 3	4 4	5 3	4 4	5 5	6 6	7 7
o	6	7 5	6 4	5 5	6 4	5 5	4 6	5 7	8 8
	6	7 5	6 4	5 5	6 4	5 5	6 4	5 5	6 6
o	7	8 6	7 5	6 6	7 5	6 6	5 5	4 6	7 7
	7	8 6	7 5	6 6	7 5	6 6	7 5	6 4	5 5
k	8	9 7	8 6	7 7	8 6	7 7	8 6	7 5	4 6
	8	9 7	8 6	7 7	8 6	7 7	8 6	7 5	6 4

Operations (Note: extra boxes at end of the word do NOT represent insertions):

Transform fatbrook into facebook

1. Insert 1 character (character c) into fatbrook

f	a	c	e	b	o	o	k		
f	a	c	t	b	r	o	o	k	

2. Insert 1 character (character t) into facebook

f	a	c	t	e	b	o	o	k	
f	a	c	t	b	r	o	o	k	

3. Insert 1 character (character e) into fatbrook

f	a	c	t	e	b	o	o	k	
f	a	c	t	e	b	r	o	o	k

4. Delete the r in fatbrook

f	a	c	t	e	b	o	o	k
f	a	c	t	e	b	o	o	k

Q3:

Threshold Adjustment Algorithm. Used some ideas from slide 10 of http://homepages.inf.ed.ac.uk/libkin/teach/dataintegr09/topk.pdf

Initialize n lists and n steps

Allocate and initialize to null, an array for the scores of each document

Initialize Upper Bound to 0

Do parallel access to each sorted list L_i

Retrieve all elements in lists

For i = 1 to n steps do *// assuming lists start at 1*

Calculate Upper Bound = Score(A_i) + 2 x Score(B_i) + Score (C_i)

Obtain the i^{th} document and score pair in each list, add score to existing document score, and store in score array

During calculation of new document scores, if any document score$_i$ > Upper Bound

Add to sorted list of top K answers

Return when there are K documents in top K *// K = 2 in our case*

End for

End procedure

Illustration of Algorithm using the example. Algorithm is in italics.

Given sorted lists

A: (D1, 1.00), (D2, 0.80), (D3, 0.50), (D4, 0.30), (D5, 0.10)

B: (D2, 0.40), (D3, 0.35), (D1, 0.15), (D4, 0.10), (D5, 0.05)

C: (D4, 0.80), (D3, 0.60), (D1, 0.20), (D5, 0.10), (D2, 0.00)

Initialize n lists and n steps

Lists = 3; steps = 5

Assignment 3

Alan Ou
301320642

Allocate and initialize to null, an array for the scores of each document

Document Scores [steps] = [null, null, null, null, null]

Initialize Upper Bound to 0

Upper Bound = 0

Do parallel access to each sort list L = 1 to n lists

Retrieve all elements in lists

A: (D1, 1.00), (D2, 0.80), (D3, 0.50), (D4, 0.30), (D5, 0.10) // Lists A, B, C and all its elements are

B: (D2, 0.40), (D3, 0.35), (D1, 0.15), (D4, 0.10), (D5, 0.05) // retrieved

C: (D4, 0.80), (D3, 0.60), (D1, 0.20), (D5, 0.10), (D2, 0.00)

For i = 1 to n steps do **// For i = 1**

Calculate Upper Bound = Score(A_i) + 2 x Score(B_i) + Score (C_i)

Upper Bound = *Score(A_1) + 2 x Score(B_1) + Score (C_1) = 1.00 + 2 x 0.40 + 0.80 =* 2.6

Obtain the i^{th} document and score pair in each list, add score to existing document score, and store in score array

1^{st} document and score pair in each list = (D1, 1.00), (D2, 0.40), and (D4, 0.80)

Document scores vector before addition = [null, null, null, null, null]

Now, add the document score of D1 to value stored in score array then store = 1 + null = 1.00

Add store the document score of D2 to value stored in score array then store = 0.40 + null = 0.40

Add the document score of D4 to value stored in score array then store = 0.80 + null = 0.80

Document scores array after the above computations = [1.00, 0.40, null, 0.80, null]

During calculation of new document scores, if any document score > Upper Bound

Add to sorted list of top K answers

No document scores are greater than the Upper Bound therefore pass

Return when there are K documents in top K

Assignment 3

Alan Ou
301320642

Step 2

For I = 1 to n steps do **// For i = 2**

Calculate Upper Bound = Score(A_i) + 2 x Score(B_i) + Score (C_i)

Upper Bound = *Score(A_2) + 2 x Score(B_2) + Score (C_2) = 0.80 + 2 x 0.35 + 0.60 =* 2.1

Obtain the i^{th} score in each list, add to existing document score, and store in score array

2^{nd} document and score pair in each list = (D2, 0.80), (D3, 0.35), and (D3, 0.60)

Document scores vector before addition = [1.00, 0.40, null, 0.80, null]

Now, add the document score of D2 to value stored in score array then store = 0.80 + 0.40 = 1.20

Add the document score of D3 to value stored in score array then store = 0.35 + null = 0.35

Add the document score of D3 to value stored in score array then store = 0.60 + 0.35 = 0.95

Document scores array after the above computations = [1.00, 1.20, 0.95, 0.80, null]

During calculation of new document scores, if any document score > Upper Bound

Add to sorted list of top K answers

No document scores are greater than the Upper Bound therefore pass

Return when there are K documents in top K

Step 3

For I = 1 to n steps do **// For i = 3**

Calculate Upper Bound = Score(A_i) + 2 x Score(B_i) + Score (C_i)

Upper Bound = *Score(A_3) + 2 x Score(B_3) + Score (C_3) = 0.50 + 2 x 0.15 + 0.20 =* 1.00

Obtain the i^{th} score in each list, add to existing document score, and store in score array

3^{rd} document and score pair in each list = (D3, 0.50), (D1, 0.15), and (D1, 0.20)

Document scores vector before addition = [1.00, 1.20, 0.95, 0.80, null]

Now, add the document score of D3 to value stored in score array then store = 0.50 + 0.95 = 1.45

Add the document score of D1 to value stored in score array then store = 0.15 + 1.00 = 1.15

Add the document score of D1 to value stored in score array then store = 0.20 + 1.15 = 1.35

Document scores array after the above computations = [1.35, 1.20, 1.45, 0.80, null]

During calculation of new document scores, if any document score > Upper Bound

Add to sorted list of top K answers

Document scores for documents 1 and 3 are greater than upper bound, therefore add both to top K list

Return when there are K documents in top K *// K = 2 in our case*

Return because there are 2 documents with scores > Upper Bound

End for

End procedure

Q4:

Number	2-bit number	Compressed number method #1	Compressed number method #2	Compressed number method #3
0	00	0	0	0
1	01	01	1	1
2	10	10	10	0
3	11	11	11	1

In the above table, we have a list of the possible numbers that can be encoded using 2-bits, its representation in bits, and three possible methods of compression.

Now, onto the proof. The company claims that SuperShrink is an “unambiguous lossless compression scheme” and also claims that it will “reduce the size of any sequence of 2-bit numbers by at least 1 bit”; however, this cannot both be true. Let’s prove this using an example, consider the sequences 3131 and 0101.

Sequence	Sequence in 2-bit notation	After applying compressed number method #1	After applying compressed number method #2	After applying compressed number method #3
3131	11 01 11 01	11 01 11 01	11 1 11 1	1 1 1 1
0310	00 11 01 00	0 11 01 0	0 11 1 1	0 1 1 0
3333	11 11 11 11	11 11 11 11	11 11 11 11	1 1 1 1

First, let’s discuss the resulting compressed number using compression method #1. The sequence 3131 does not change, while the sequence 0310 shrinks by two bits. It appears that SuperShrink’s claim is true at least in terms of the bits saved, but the problem here is decoding. Recall that SuperShrink claims to be an “unambiguous lossless compression scheme”. Now, looking at the compressed bit sequence. 0310 is compressed into 0 11 01 0. With spaces, it may be clear as to which bit(s) decodes to which number, but in the computer, there are no spaces, then the sequence will become 011010. When trying to decode, one possible sequence can potentially become 0 11 0 10 which decodes to 0302, which is not what we originally started with.

Looking at the resulting compressed bit sequence for 3131 using compressing method #2. Again, if we removed the spaces, we would not be able to identity without doubt, that the sequence was 3131 and not a sequence of all 1's (111111). Again, this compression method causes ambiguity when trying to decode.

Same case with method #3, looking at the compressed sequence 1 1 1 1, we would not be 100% sure which "1" represents 3 and which represents 1.

Regardless of what method, even those not mentioned in this example, creating compressed encodings through the removal of 1 or more bits causes ambiguity to arise. The reason is that when we originally had 2-bits for each number, each 2-bit combination was the same length, and each represented a unique number. If we removed just one bit from 1 of the 4 possible number bit encodings, then ambiguity would arise as it would no longer be clear which bit goes with which bit in the case of numbers that are represented using two-bits, and which bit was only a single bit representing one number.

Now, SuperShrink claims that it is an "unambiguous lossless compression scheme". But as proved above, removing 1 or more bits from a bit encoding causes ambiguity, therefore if SuperShrink wants to be an "unambiguous lossless compression scheme", it must maintain all the bits; this means that SuperShrink never uses less space than an uncompressed encoding.

Q5:

Map and reduce functions adapted from Fig.5.11 – Fig.5.14 in the textbook

Map1 works similarly to Fig.5.13 as it parses every document in the input, and for each word in the parsed document, it emits a key/value pair for which the key is the word and the value is 1.

```
procedure Map1 (input)
    while not input.done() do
        document ← input.next()
        tokens ← Parse(document)
        for each word w in tokens do
            Emit(w, "1")
        end for
    end while
end procedure
```

The reducer is similar to Fig.5.12. It takes the shuffled words and adds all the words that are the same. In other words, the reducer adds together the number of times the same word occurs in the set of documents. At the end, the key (word) and value (total) are emitted.

```
procedure Reducer1 (key, values)
        word ← key
        int total = 0;
        while not values.done() do
                total ← total + 1
        end while
        Emit(word, total)
end procedure
```

Map2 words similar to Fig.5.11. It takes in the results produced by the first mapreduce and swaps the key and value pair so that the shuffler can shuffle and "group together" the words that have the same number of occurrences in the set of documents.

```
procedure Map2 (key, value)
        word ← key
        total ← value
        Emit(value, key)
end procedure
```

Reducer2 takes the shuffled results, which are now sorted in ascending order based on the number of occurrences a word has shown up in the set of documents and adds together the number of words that show up the same number of times. At the end, the total number of words that show up the same number of times and the number of times the words show up in the set of documents are emitted.

```
procedure Reducer2 (key, values)
        int total = 0;
        while not values.done() do
                total ← total + 1
        end while
        Emit(total, key)
end procedure
```