Edit distance and LCS

Friday, November 3, 2023 3:44 PM

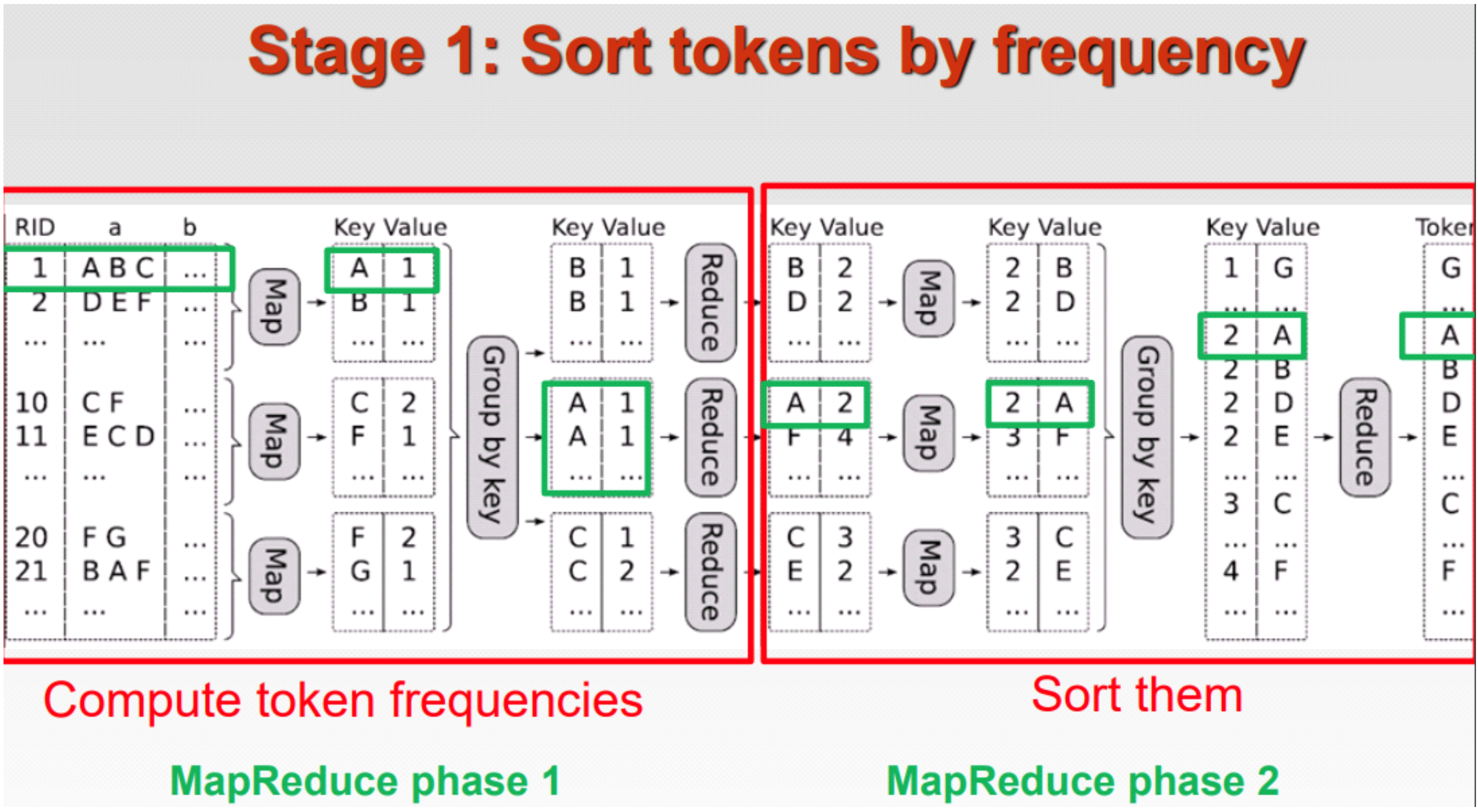
https://leetcode.com/problems/edit-distance/
Edit Distance - Dynamic Programming - Leetcode 72 - Python

https://leetcode.com/problems/longest-common-subsequence/
Longest Common Subsequence - Dynamic Programming - Leetcode 1143

Project 3

Friday, November 3, 2023 3:43 PM

Prefix Filtering



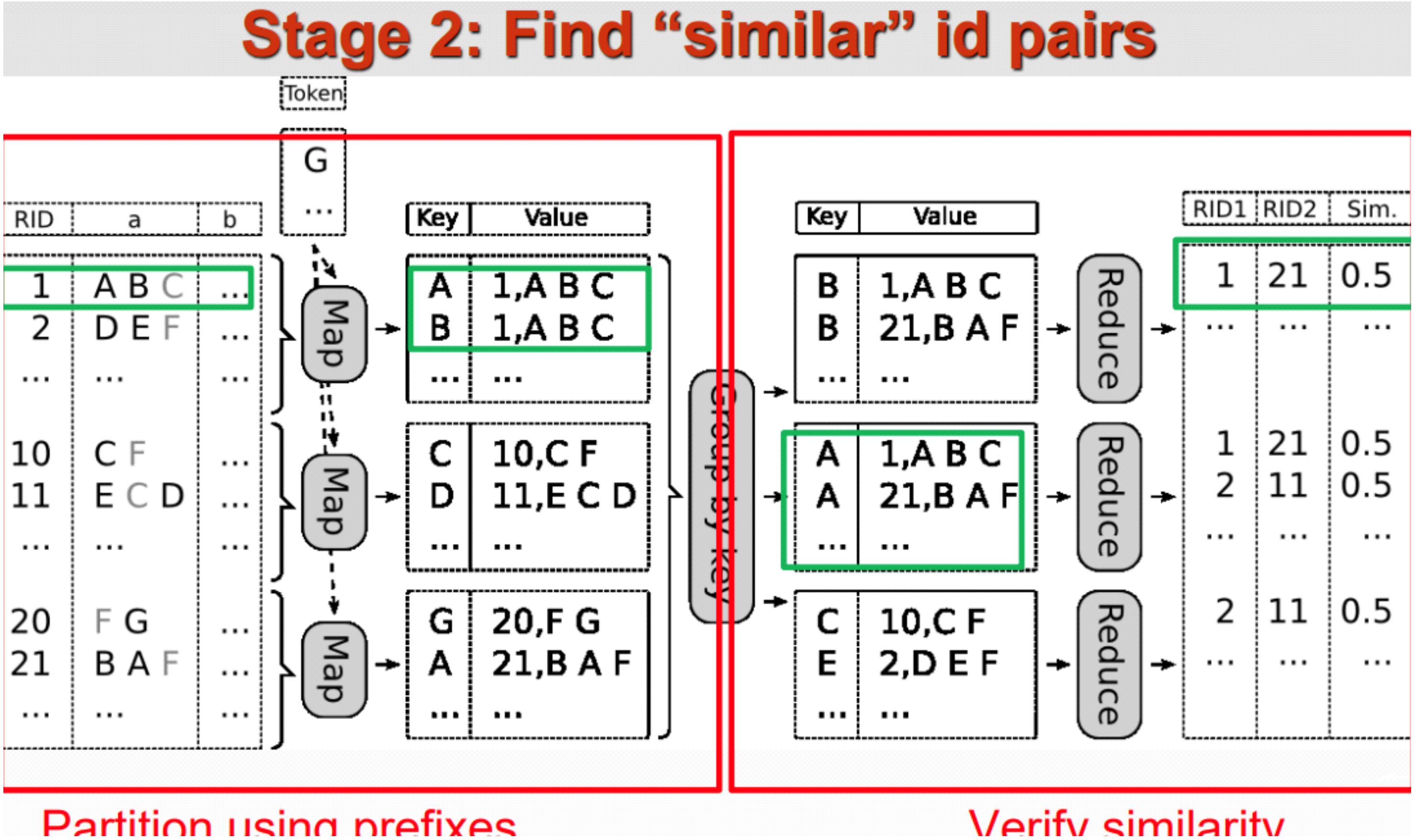
Stage 1: Sort tokens by frequency

Compute token frequencies

Sort them

MapReduce phase 1

MapReduce phase 2



Stage 2: Find “similar” id pairs

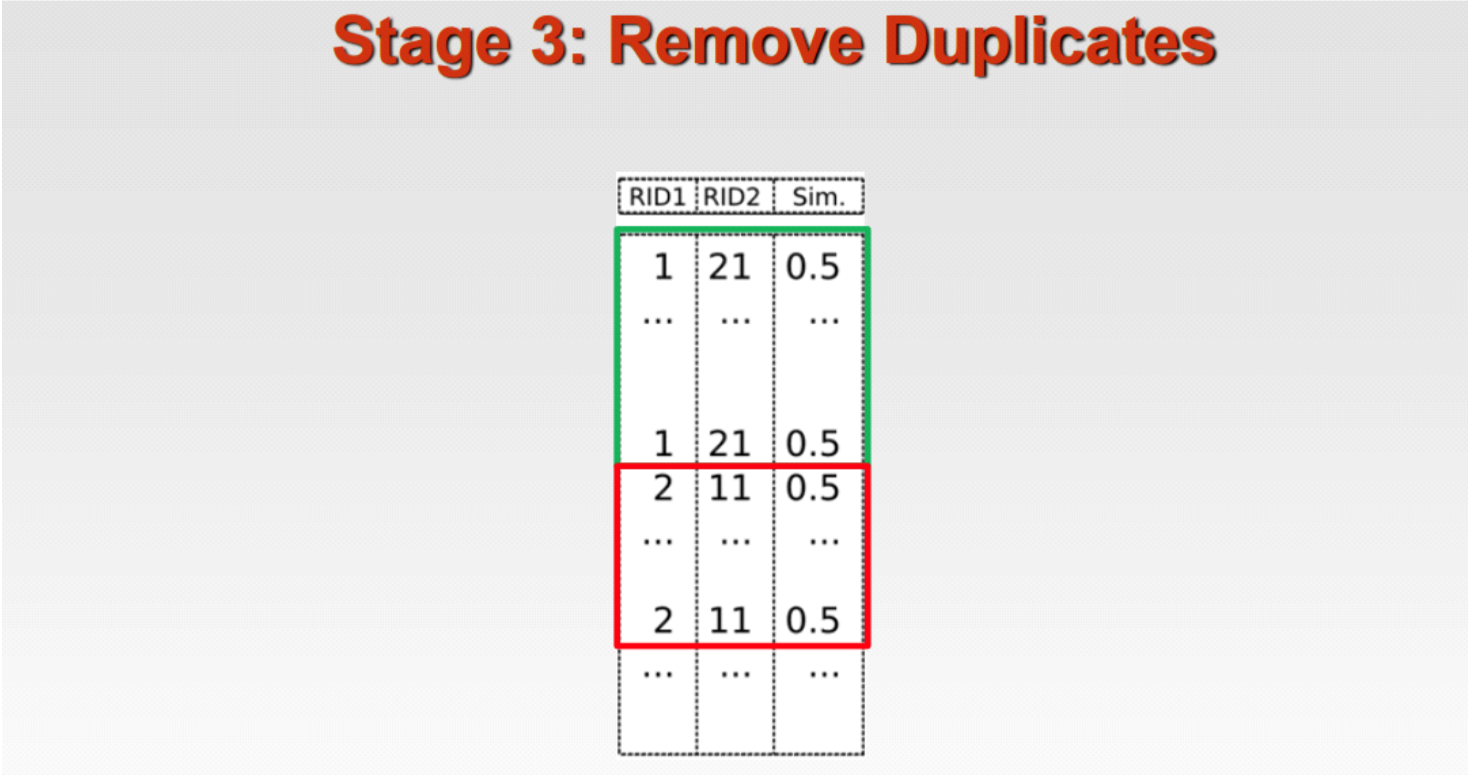
Partition using prefixes

Verify similarity

Partition using prefixes

Verify similarity

The key step here is to determine the prefix length in black



Stage 3: Remove Duplicates

RID1	RID2	Sim.
1	21	0.5
…	…	…
1	21	0.5
2	11	0.5
…	…	…
2	11	0.5
…	…	…

6.1.1 Sampling Data Streams

Thursday, November 23, 2023 12:10 PM

Goal: sampling should preserve the properties of our data

Problem 1: How often did a user run the same query in a single days?

Naïve Solution: generate a random integer in [0…9] for each query, store the query if the integer is 0, otherwise discard. This will store 1/10 of the query stream

Problem 2: What fraction of queries by an average search engine user are duplicates?

Suppose each user issues x queries once and d queries twice (total of x + 2d queries),

the correct answer: d/(x + d)

if we use sampling algorithm: sample will contain x/10 of the singleton queries and 2d/10 of the duplicate queries at least once. But only d/10 pairs of duplicates, so the sample based answer is

$$\frac{\frac{d}{100}}{\frac{x}{10}+\frac{d}{100}+\frac{18d}{100}}$$

So it is better to randomly sample user in this case, pick 1/10 of users and take all their searches in the sample

Reservoir Sampling

1. store all the first s elements of the stream to S
2. Suppose we have seen n - 1 elements, and now the nth element arrives (n > s)
   - With probability s/n, keep the nth element, else discard it
   - if we picked the nth element, then it replaces one of the s elements in the sample S, picked uniformly in random.

6.1.2 Querying Data Stream

Thursday, November 23, 2023 12:33 PM

Sliding Window Counting bits (0s and 1s)

Problem 1: how many 1s are there in the last k bits?

DGIM Algorithm: it doesn't assume data is distributed uniformly. It stores O($\log^2$N) bits per stream. It gives approximate answer, never off by more than 50%.

1. Each bit in the stream has a timestamp, starting 0,1,2…Record timestamps modulo N, so we can represent any relevant timestamp in O($\log^2$N) bits
2. We partition our timestamps into bukets. A bucket is a segment of the window. It is represented by timestamp of its end O($\log^2$N) bits, the number of 1s between its beginning and end O(loglogN) bits.

- Updating buckets: when a new bit comes in, drop the last(oldest bucket if its end time is outside of N before the current time. If current bit is 0, continue.

Explain the space complexity of maintaining one bucket in the DGIM algorithm in terms of the window size N. How many buckets you need to store at most?

Answer:

a). The timestamp of its end [O(logN) bits]

b). The number of 1s between its beginning and end [(loglogN) bits]

Reason of (loglogN): N bits, each bucket contains at most (logN) 1s, the number of 1s are orders of 2, and thus can only store the orders rather than the original number, and hence (loglogN).

c). At most logN buckets

Suppose we are maintaining a count of 1s using the DGIM method. We represent a bucket by (i, t), where i is the number of 1s in the bucket and t is the bucket timestamp (time of the most recent 1). Consider that the current time is 200, window size is 60, and the current list of buckets is: (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (1, 197) (1, 200). At the next ten clocks, 201 through 210, the stream has 0101010101. What will the sequence of buckets be at the end of these ten inputs?

Answer: when updating buckets, if the current bit is 1 we do two things:

1. create a new bucket of size 1, for just this bit, end timestamp = current time
2. if there are now there buckets of size n, combine the oldest two into a bucekt of size 2n and so on
3. if the current bit is 0, do nothing

(1) (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (1, 197) (1, 200)(1, 202) => (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (2, 200)(1, 202)

(2) (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (1, 197) (1, 200)(1, 202) => (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (2, 200)(1, 202)(1, 204)

(3) (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (1, 197) (1, 200)(1, 202) => (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (2, 200)(1, 202)(1, 204)(1, 206) => (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (1, 197) (1, 200)(1, 202) => (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (2, 200)(2, 204)(1, 206) => (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (1, 197) (1, 200)(1, 202) => (16, 148) (8, 162) (8, 177) (4, 183) (4, 200)(2, 204)(1, 206)

(4) (16, 148) (8, 162) (8, 177) (4, 183) (4, 200)(2, 204)(1, 206)(1, 208) =>(8, 162) (8, 177) (4, 183) (4, 200)(2, 204)(1, 206) (1, 208)

(5) (8, 162) (8, 177) (4, 183) (4, 200)(2, 204)(1, 206)(1, 208)(1, 210)=>(8, 162) (8, 177) (4, 183) (4, 200)(2, 204)(2, 208)(1, 210)

6.2.1 Bloom Filter & Counting Bloom Filter

Saturday, October 28, 2023 1:19 AM

filtering a data stream: select elements with property x from the stream

Application 1: email spam filtering

Application 2: publish-subscribe system

Problem: false positive due to hash collision, probability of false positive.

Throwing darts: an analysis for the number of false positives

if we have 10^9 darts, 8 * 10^9 targets

fraction of 1s in the set is 1-e^(-m/n) = 1 - e^(-1/8) = 0.1175

Bloom filter: an array of bits, together with a number of hash functions

Example: N = 11, using two hash functions:

h1(x) = take odd numbered bits from the right in the binary representation of x

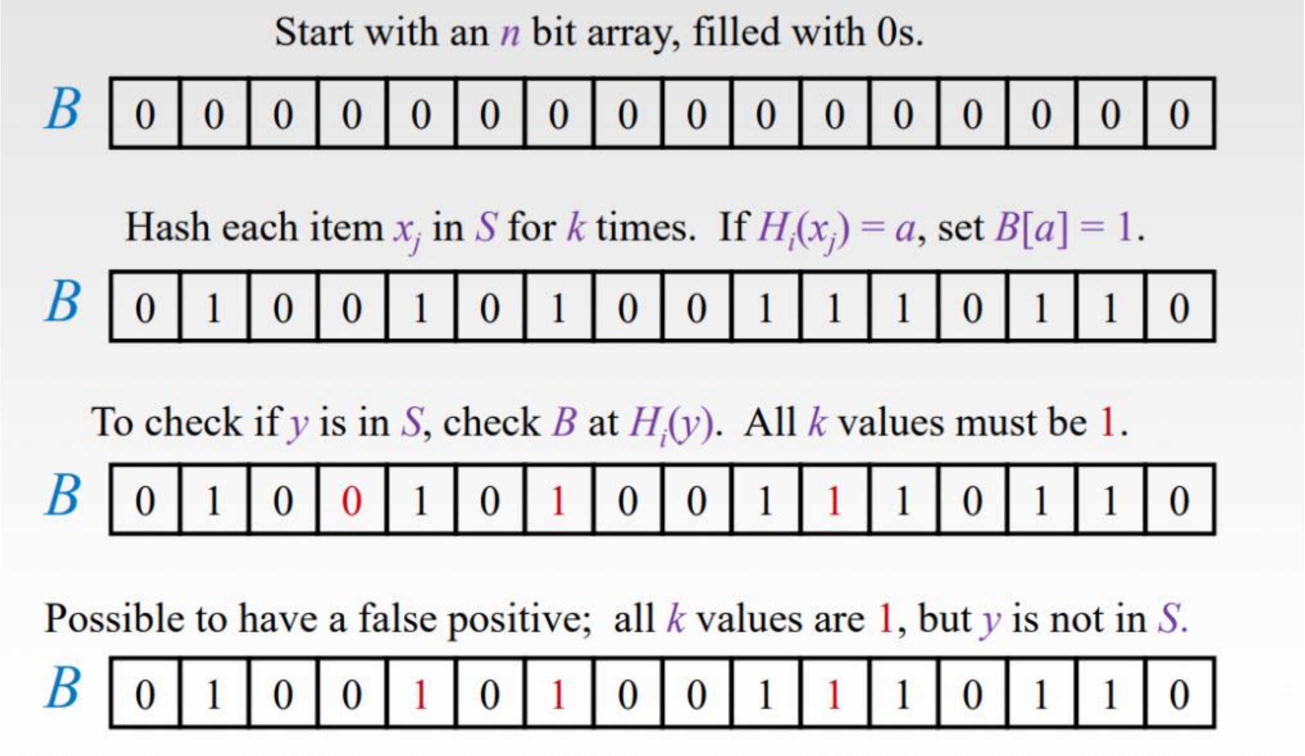
treat it as an integer i

result is i modulo 11

h1(x) = take even numbered bits from the right in the binary representation of x

number	h1	h2	Filter contents
25=11001	5	2	00100100000
159=10011111	7	0	10100101000
585=1001001001	9	7	10100101010

Bloom Filter Calculation



Start with an n bit array, filled with 0s.

B																
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Hash each item x_j in S for k times. If $H_i(x_j) = a$, set $B[a] = 1$.

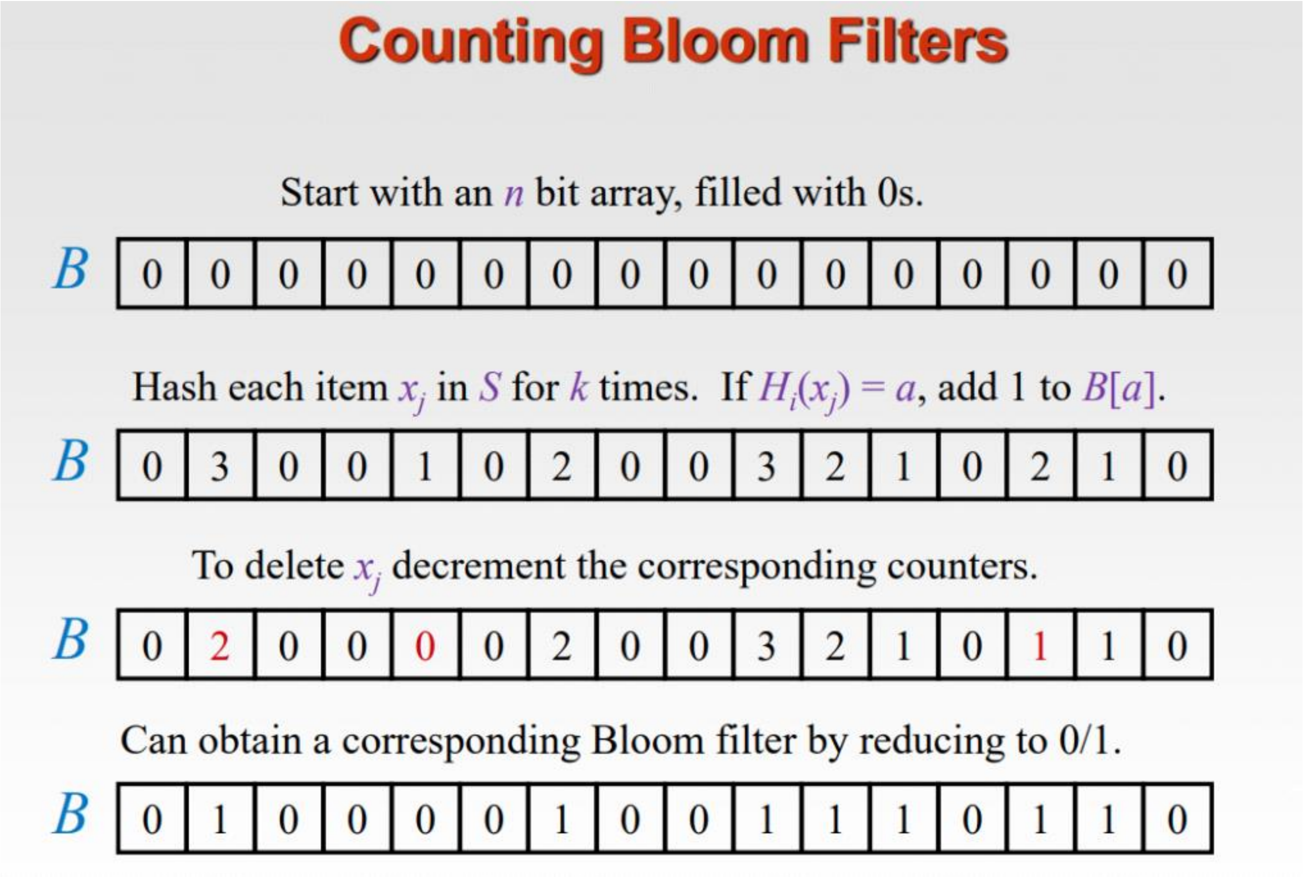
B																
	0	1	0	0	1	0	1	0	0	1	1	1	0	1	1	0

To check if y is in S, check B at $H_i(y)$. All k values must be 1.

B																
	0	1	0	0	1	0	1	0	0	1	1	1	0	1	1	0

Possible to have a false positive; all k values are 1, but y is not in S.

B																
	0	1	0	0	1	0	1	0	0	1	1	1	0	1	1	0



Counting Bloom Filters

Start with an n bit array, filled with 0s.

B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Hash each item x_j in S for k times. If $H_i(x_j) = a$, add 1 to $B[a]$.

B	0	3	0	0	1	0	2	0	0	3	2	1	0	2	1	0

To delete x_j decrement the corresponding counters.

B	0	2	0	0	0	0	2	0	0	3	2	1	0	1	1	0

Can obtain a corresponding Bloom filter by reducing to 0/1.

B	0	1	0	0	0	0	1	0	0	1	1	1	0	1	1	0

Consider a Counting Bloom filter of size m = 9 and 2 hash functions that both take a string (lowercase) as input:

h1(str) = $\sum_{c\ in\ str}(c - 'a')$ mod 9

h2(str) = (str.length * 2) mod 9

Here, c - ‘a’ is used to compute the position of the letter c in the 26 alphabetical letters, e.g., h1(“bd”) = (1 + 3) mod 9 = 4.

(i) (2 marks) Given a set of string S = {“hi”, “big”, “data”, “spark”}, show the update of the Bloom filter

(ii) (1 mark) Delete “big” from S, and use the bloom filter to check if “nosql” is contained in S.

1)

S = {“hi”, “big”, “data”, “spark”}

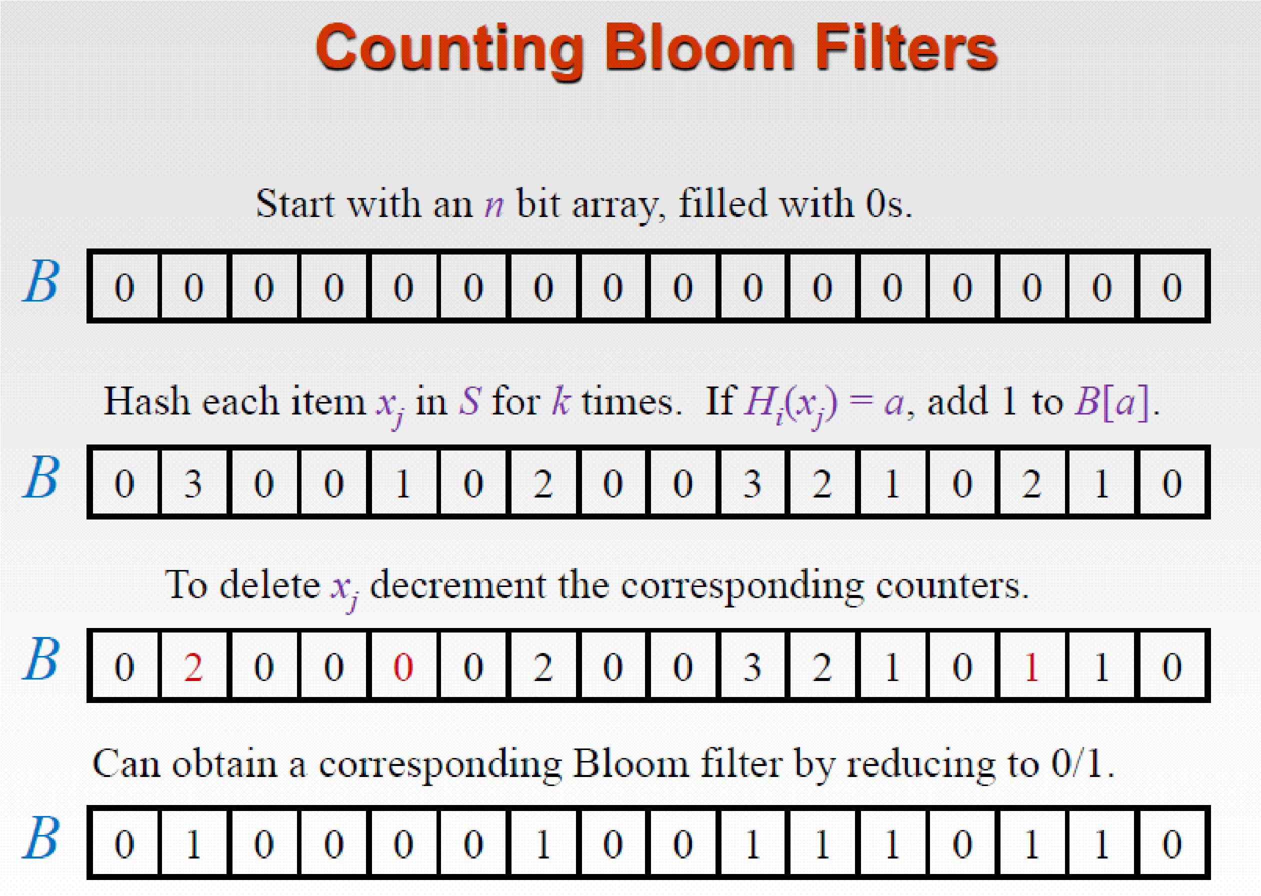
h1("hi") = (7 + 8) mod 9 = 6
h2("hi") = 2 * 2 mod 9 = 4

Counting Bloom filter contents: 000010100
h1("big") = (1 + 8 + 6) mod 9 = 6
h2("big") = (3 * 2) mod 9 = 6
Counting Bloom filter contents: 000010300
h1("data") = (3 + 19) mod 9 = 4
h2("data") = (4 * 2) mod 9 = 8
Counting Bloom filter contents: 000020301
h1("spark") = (18 + 15 + 17 + 10) mod 9 = 6
h2("spark") = (5 * 2) mod 9 = 1
Counting Bloom filter contents: 010020401

Final Corresponding Bloom filter contents: 010010101

2)
Delete "big" from S, the final corresponding bloom filter stays the same 010010101
h1("nosql") = (13 + 14 + 18 + 16 + 11) mod 9 = 0
h2("nosql") = (5 * 2) mod 9 = 1

It is not in the bloom filter



Counting Bloom Filters

Start with an n bit array, filled with 0s.

B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Hash each item x_j in S for k times. If $H_i(x_j) = a$, add 1 to $B[a]$.

B	0	3	0	0	1	0	2	0	0	3	2	1	0	2	1	0

To delete x_j decrement the corresponding counters.

B	0	2	0	0	0	0	2	0	0	3	2	1	0	1	1	0

Can obtain a corresponding Bloom filter by reducing to 0/1.

B	0	1	0	0	0	0	1	0	0	1	1	1	0	1	1	0

6.2.2 Finding Frequent Elements - Majority and Heavy Hitters

Thursday, November 2, 2023 10:55 AM

Finding frequent elements (majority and heavy hitters)

Given a stream of elements, find the majority if there is one: A majority element in the data stream is an element that appears more than n/2 times

A A B C D A A B B A A A A A A C C C D A B A A A

The answer is: A in this case.

Boyer-Moore Voting Algorithm

If we cancel out each occurrence of an element e with all the other elements that are different from e, then e will exist till the end.

Then, we can check if it is indeed the majority element. **(This algorithm takes O(n) time and O(1) space.)**

Phase 1:

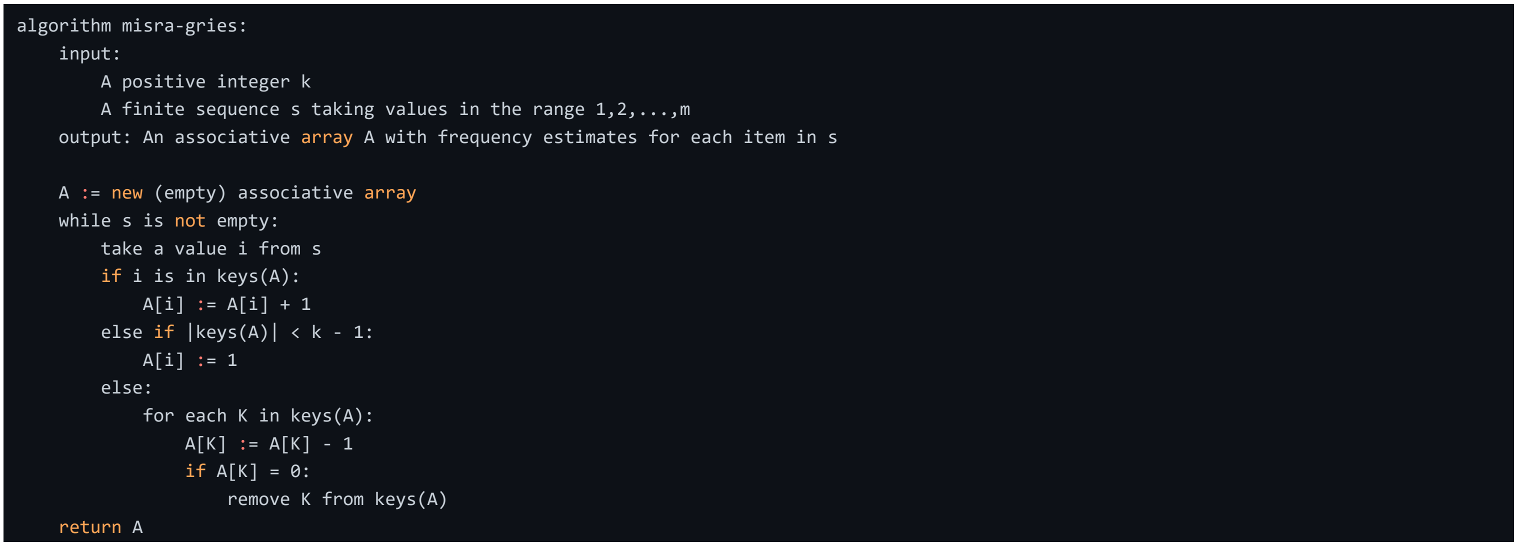
```
maj_index = 0
count = 1
for i in range(len(A)):
    if A[maj_index] == A[i]:
        count += 1
    else:
        count -= 1
    if count == 0:
        maj_index = i
        count = 1
return A[maj_index]
```

Phase 2:

```
count = 0
for i in range(len(A)):
    if A[i] == cand:
        count += 1
if count > len(A)/2:
    return True
else:
    return False
```

Heavy Hitter(A more general problem: find all elements with counts > n/k (k >=2)

Misra-Gries Algorithm



```
algorithm misra-gries:
    input:
        A positive integer k
        A finite sequence s taking values in the range 1,2,...,m
    output: An associative array A with frequency estimates for each item in s

    A := new (empty) associative array
    while s is not empty:
        take a value i from s
        if i is in keys(A):
            A[i] := A[i] + 1
        else if |keys(A)| < k - 1:
            A[i] := 1
        else:
            for each K in keys(A):
                A[K] := A[K] - 1
                if A[K] = 0:
                    remove K from keys(A)
    return A
```

Keep k-1 different candidates in hand (thus with space O(k))

For each element in stream:

- ➢ If item is monitored, increase its counter
- ➢ Else, if < k-1 items monitored, add new element with count 1
- ➢ Else, decrease all counts by 1, and delete element with count 0

Each decrease can be charged against k arrivals of different items, so no item with frequency N/k is missed

But false positive (elements with count smaller than n/k) may appear in the result

Question [1,1,2,3,4,5,1,1,1,5,3,3,1,1,2] with k=3, we want to find element that occurred more than 15/3 = 5 times.
k = 3, we keep k - 1 counters = 2, *[2,3,4,5,1,1,1,5,3,3,1,1,2]*

Counter	Count
1	2

[3,4,5,1,1,1,5,3,3,1,1,2]

Counter	Count
1	2
2	1

Then 3 is not in D, there are already two keys in D, which is not less than k-1, decrement all counters in D by 1 and remove 0
[4,5,1,1,1,5,3,3,1,1,2]

Counter	Count
1	1
~~2~~	~~0~~

[5,1,1,1,5,3,3,1,1,2]

Counter	Count
1	1
4	1

[1,1,1,5,3,3,1,1,2]

Counter	Count
~~1~~	~~0~~
4	~~0~~

[5,3,3,1,1,2]

Counter	Count
1	3

[3,3,1,1,2]

Counter	Count
1	3
5	1

[1,1,2]

Counter	Count
1	2
3	1

[2]

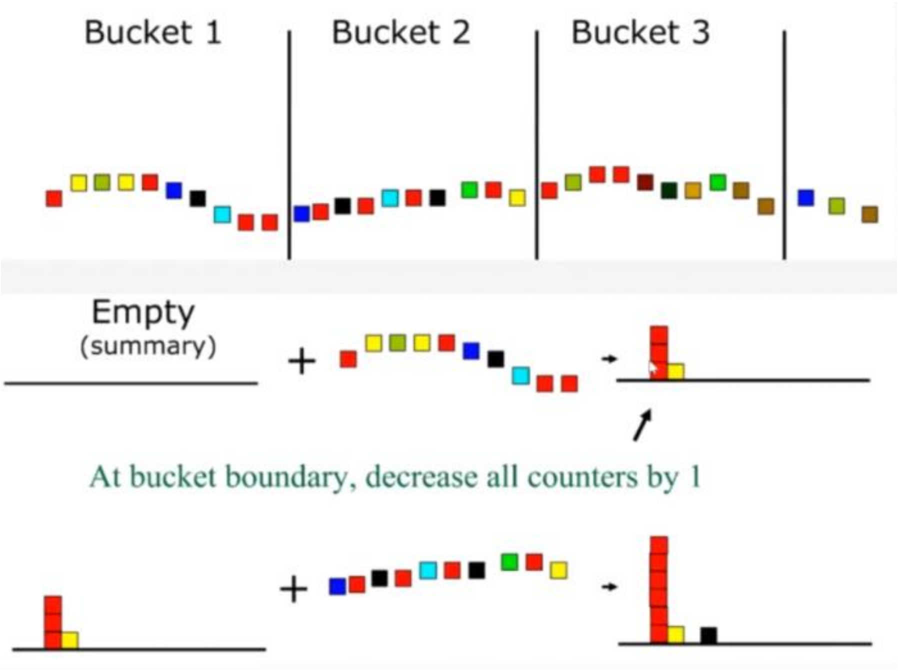
Counter	Count
1	4
3	1

Final Output

Counter	Count
1	4
~~3~~	~~0~~

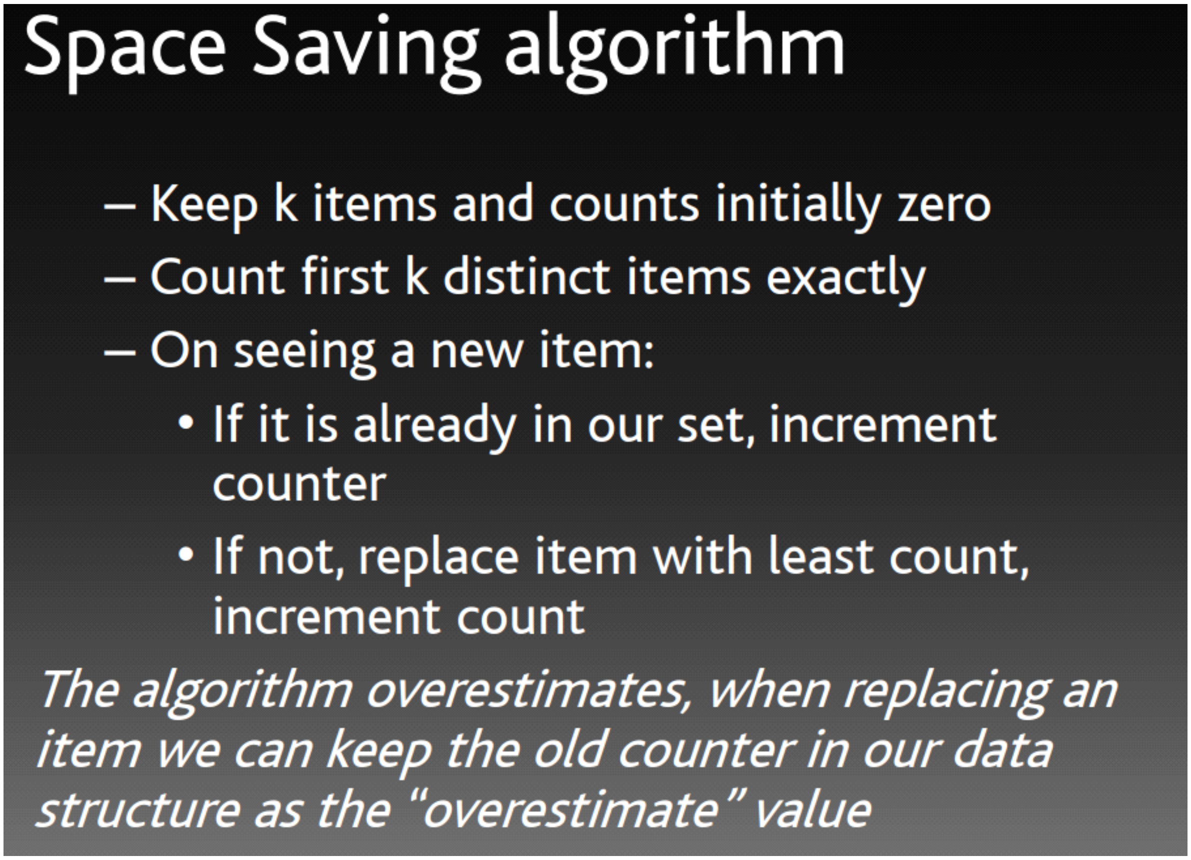
Lossy Counting

1. divide the incoming date stream into windows, and each window contains 1/e elements.
2. increment the frequency count of each item according to the new window values, after each window, decrement all counters by 1, drop elements with counter 0
3. repeat -update counters and after each window, decrement all counters by 1.



Space saving algorithm

https://romania.amazon.com/techon/presentations/DataStreamsAlgorithms_FlorinManolache.pdf



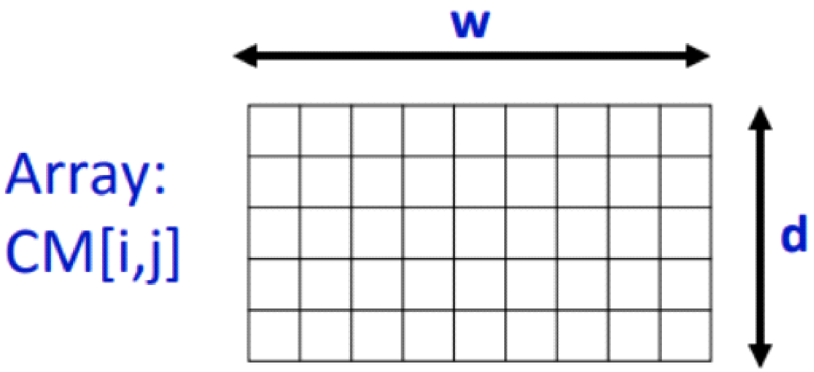
Space Saving algorithm

- Keep k items and counts initially zero
- Count first k distinct items exactly
- On seeing a new item:
  - If it is already in our set, increment counter
  - If not, replace item with least count, increment count

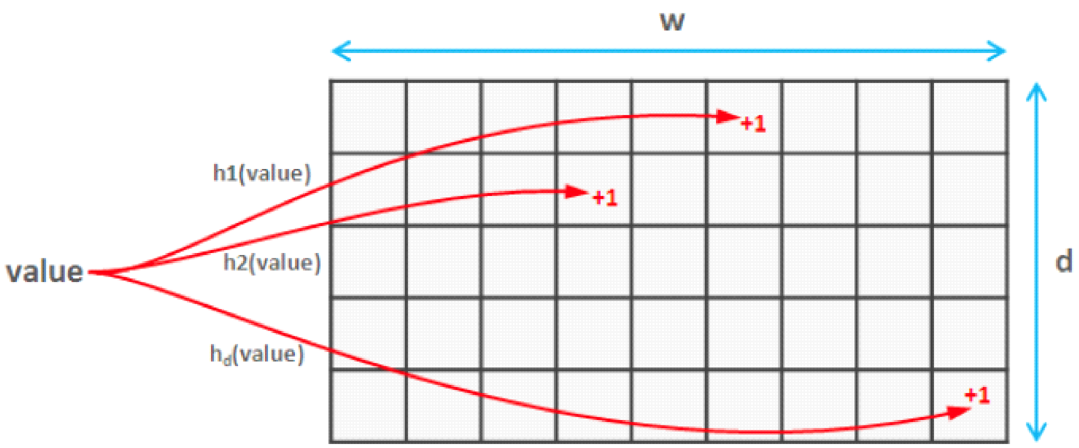
The algorithm overestimates, when replacing an item we can keep the old counter in our data structure as the "overestimate" value

6.2.3 CM (Count Min) Sketch

Thursday, November 2, 2023 11:07 AM



We maintain a 2D array with d rows and w columns. For each row, we have a hash function, for each new element that comes in, it will be hashed d times.

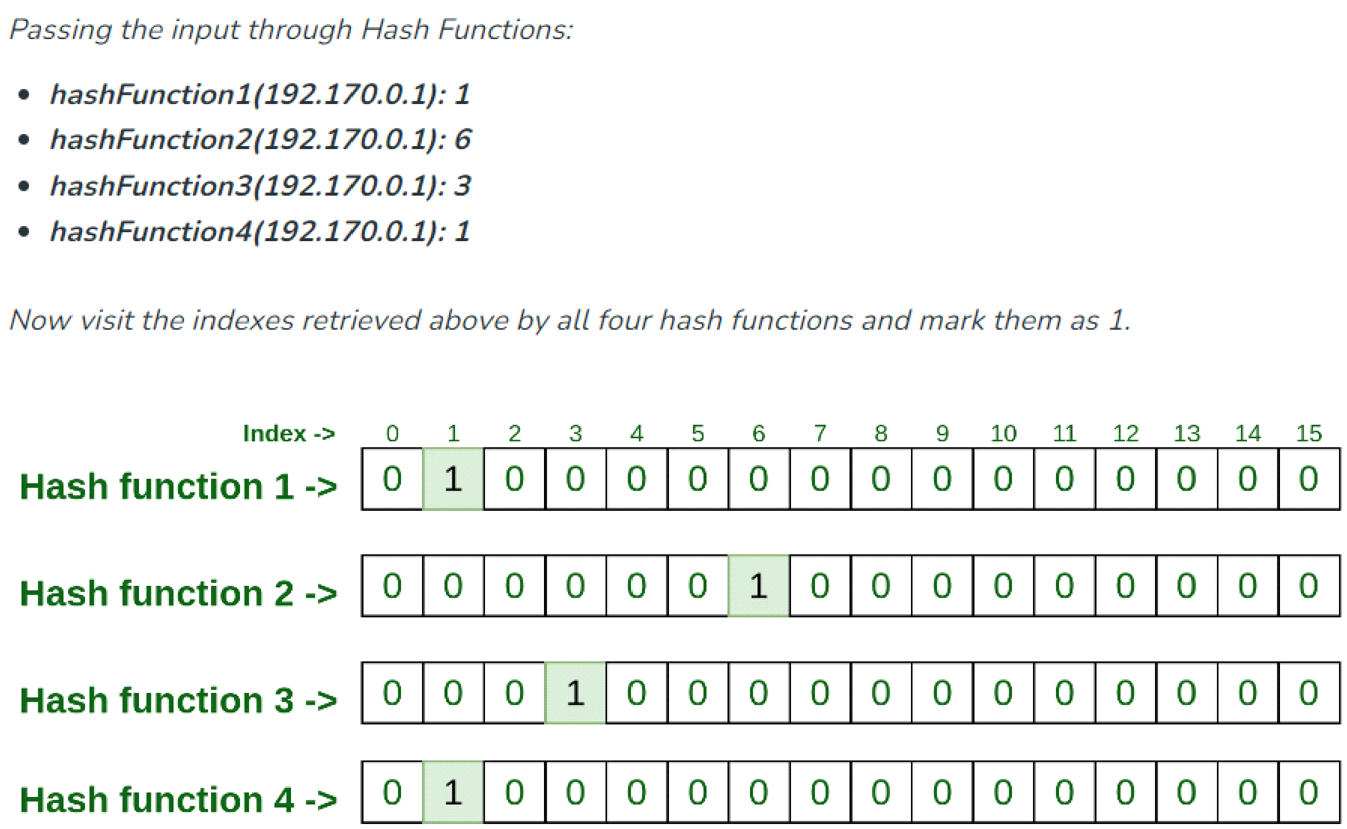


Since Count-Min Sketch is used to find the frequency of an element
The code for Count(x) is simply:

- return $\text{min}_{i=1}$=CMS[i][hi(x)]

https://www.geeksforgeeks.org/count-min-sketch-in-java-with-examples/

Input 1: 192.170.0.1



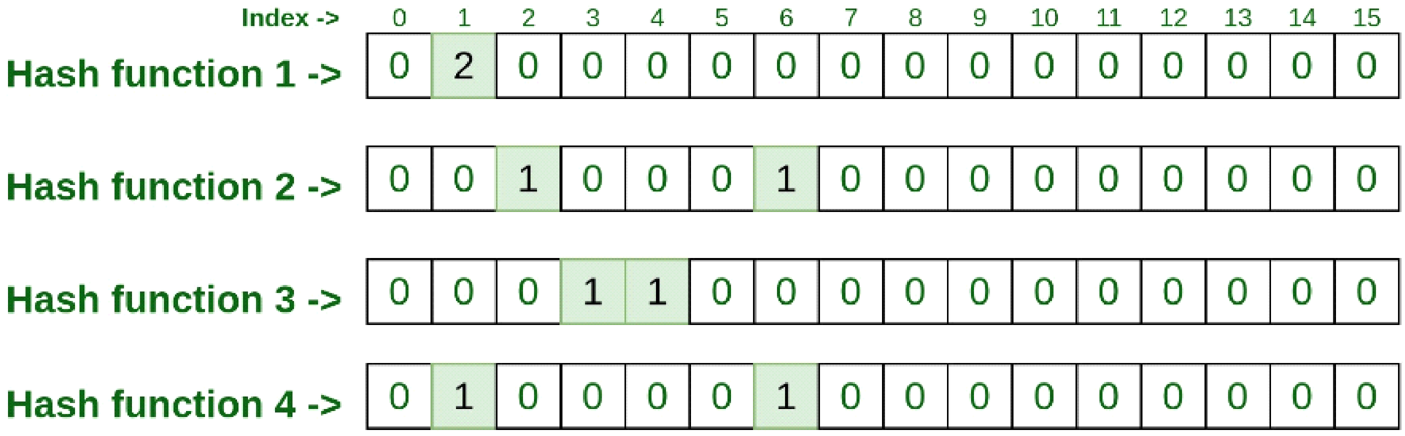
Passing the input through Hash Functions:

- ***hashFunction1(192.170.0.1): 1***
- ***hashFunction2(192.170.0.1): 6***
- ***hashFunction3(192.170.0.1): 3***
- ***hashFunction4(192.170.0.1): 1***

Now visit the indexes retrieved above by all four hash functions and mark them as 1.

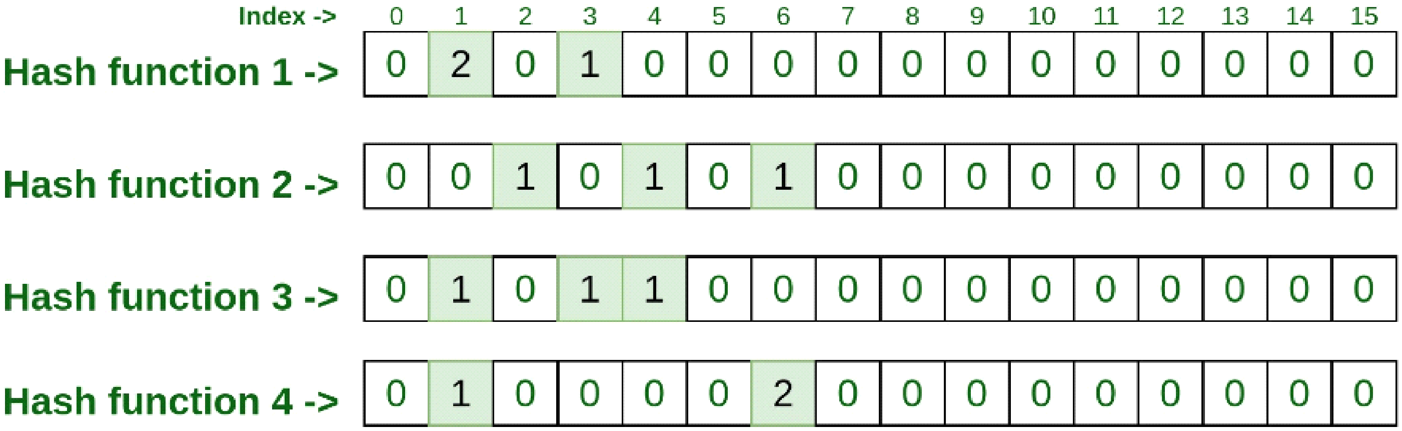
Index ->	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hash function 1 ->	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hash function 2 ->	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Hash function 3 ->	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Hash function 4 ->	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Input 2: 75.245.10.1



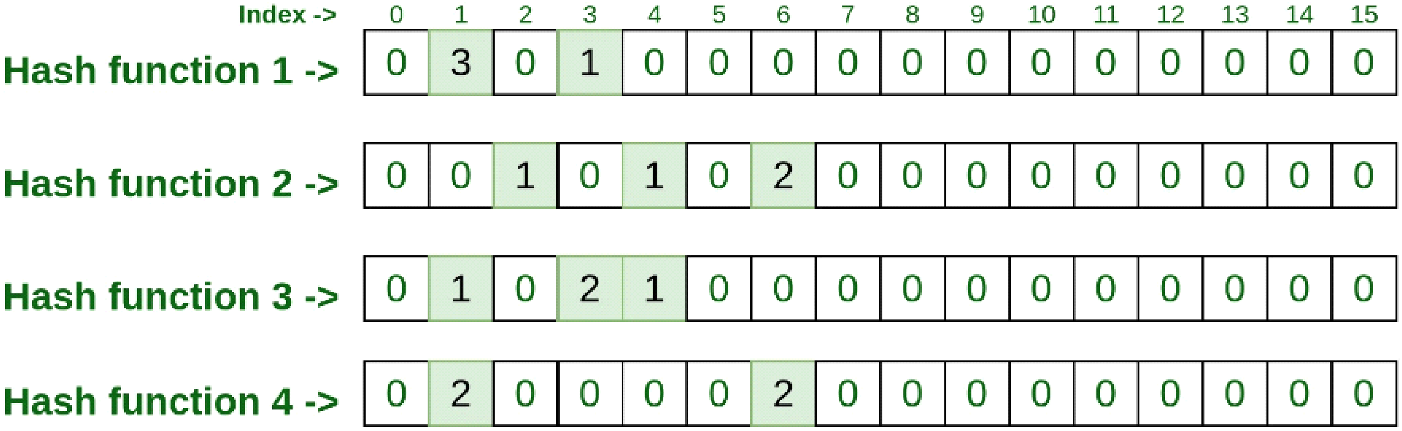
Index ->	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hash function 1 ->	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hash function 2 ->	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
Hash function 3 ->	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
Hash function 4 ->	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Input 3: 10.125.22.20



Index ->	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hash function 1 ->	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Hash function 2 ->	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
Hash function 3 ->	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
Hash function 4 ->	0	1	0	0	0	0	2	0	0	0	0	0	0	0	0	0

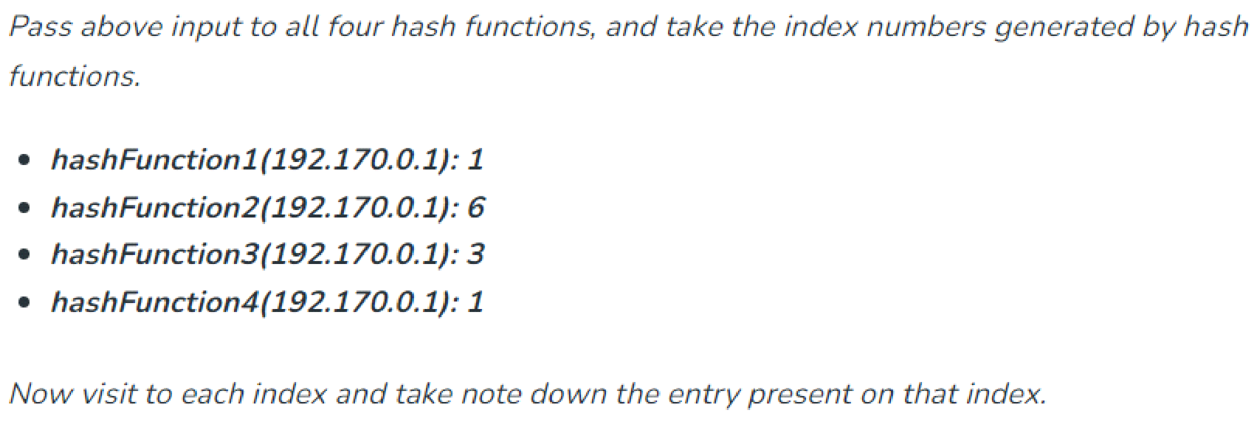
Input 4: 192.170.0.1



Index ->	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hash function 1 ->	0	3	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Hash function 2 ->	0	0	1	0	1	0	2	0	0	0	0	0	0	0	0	0
Hash function 3 ->	0	1	0	2	1	0	0	0	0	0	0	0	0	0	0	0
Hash function 4 ->	0	2	0	0	0	0	2	0	0	0	0	0	0	0	0	0

Testing Count-Min Sketch data structure against Test cases:

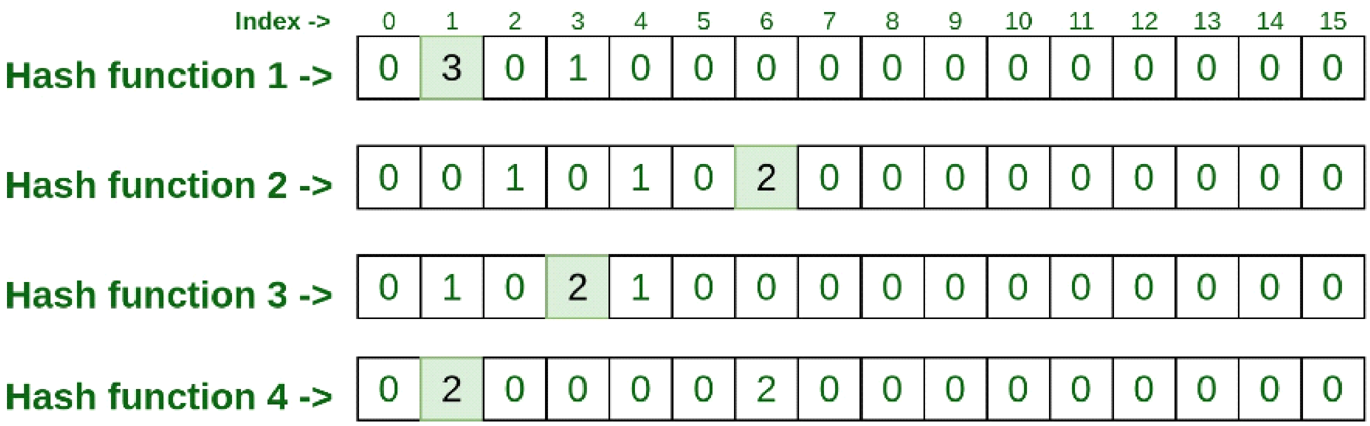
If we now test 192.170.0.1



Pass above input to all four hash functions, and take the index numbers generated by hash functions.

- ***hashFunction1(192.170.0.1): 1***
- ***hashFunction2(192.170.0.1): 6***
- ***hashFunction3(192.170.0.1): 3***
- ***hashFunction4(192.170.0.1): 1***

Now visit to each index and take note down the entry present on that index.



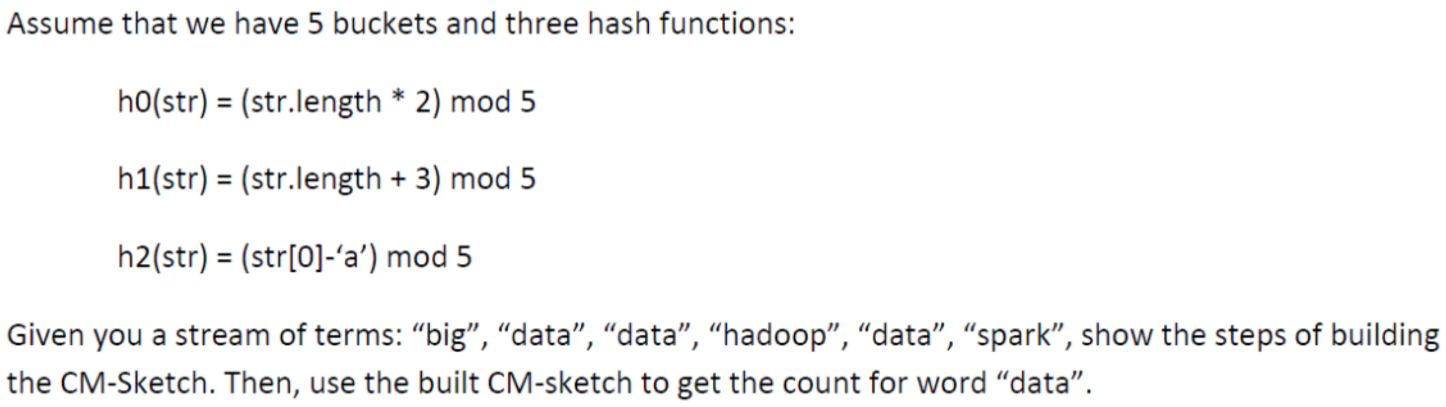
Index ->	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hash function 1 ->	0	3	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Hash function 2 ->	0	0	1	0	1	0	2	0	0	0	0	0	0	0	0	0
Hash function 3 ->	0	1	0	2	1	0	0	0	0	0	0	0	0	0	0	0
Hash function 4 ->	0	2	0	0	0	0	2	0	0	0	0	0	0	0	0	0

So the final entry on each index was 3, 2, 2, 2.

Now take the minimum count among these entries and that is the result. So min(3, 2, 2, 2) is 2, that means the above test input is processed 2 times in the above list.

Hence Output (Frequency of 192.170.0.1) = 2.

Exam question



Assume that we have 5 buckets and three hash functions:

h0(str) = (str.length * 2) mod 5

h1(str) = (str.length + 3) mod 5

h2(str) = (str[0]-'a') mod 5

Given you a stream of terms: "big", "data", "data", "hadoop", "data", "spark", show the steps of building the CM-Sketch. Then, use the built CM-sketch to get the count for word "data".

"big": h0 = 1, h1 = 1, h2 = 1

bucket	0	1	2	3	4
h0	0	1	0	0	0

h1	0	1	0	0	0
h2	0	1	0	0	0

"data": h0 = 3, h1 = 2, h2 = 3

bucket	**0**	**1**	**2**	**3**	**4**
h0	0	1	0	1	0
h1	0	1	1	0	0
h2	0	1	0	1	0

"data"

bucket	**0**	**1**	**2**	**3**	**4**
h0	0	1	0	2	0
h1	0	1	2	0	0
h2	0	1	0	2	0

"hadoop": h0 = 2, h1 = 4, h2 = 2

bucket	**0**	**1**	**2**	**3**	**4**
h0	0	1	1	2	0
h1	0	1	2	0	1
h2	0	1	1	2	0

"data"

bucket	**0**	**1**	**2**	**3**	**4**
h0	0	1	1	3	0
h1	0	1	3	0	1
h2	0	1	1	3	0

"spark": h0 = 0, h1 = 3, h2 = 3

bucket	**0**	**1**	**2**	**3**	**4**
h0	1	1	1	3	0
h1	0	1	3	1	1
h2	0	1	1	4	0

Min(CMS[0][3], CMS[1][2], CMS[2][3]) = 3

6.2.4 Counting Distinct Elements (FM Sketch)

Thursday, November 2, 2023 11:18 AM

sketch is a linear transform of the input, Model stream as defining a vector, sketch is result of multiplying stream vector by an (implicit) matrix

Problem definition: a data stream consists of elements chosen from a set of sized n, **maintain a count of the number of distinct elements seen so far**. However, we do not have space to store the complete set? We need a way to estimate the count in an unbiased way.

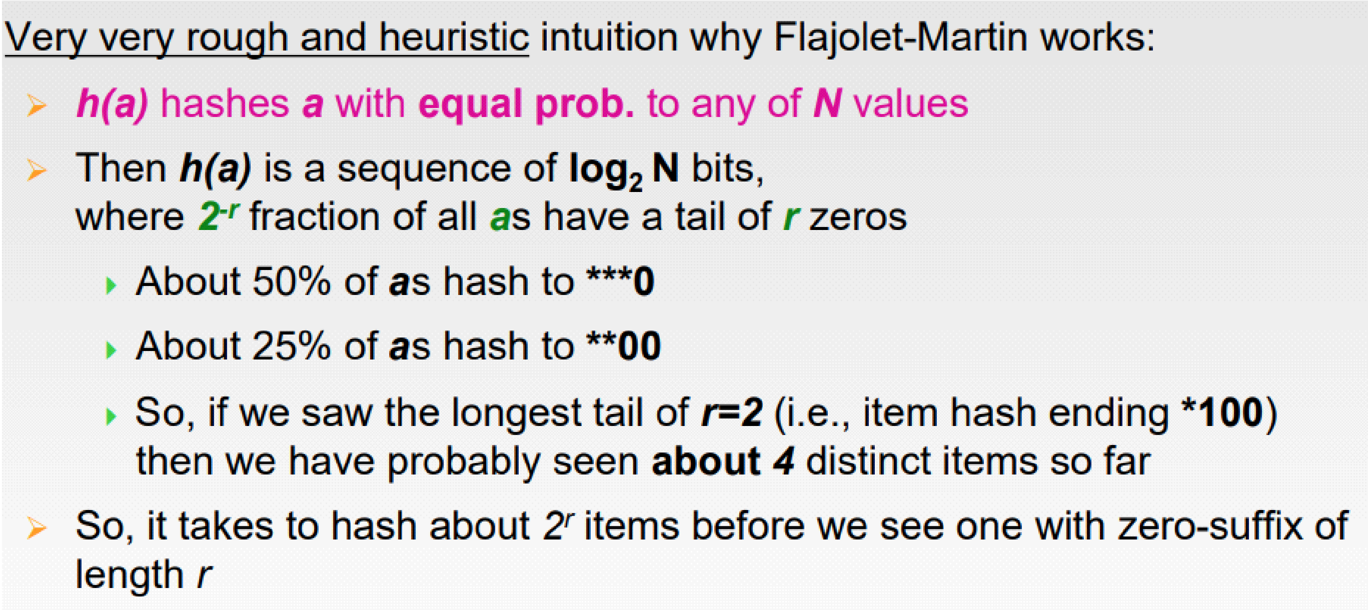
Application:

1. How many different words are found among the web pages being crawled at a site? For example spam detection
2. How many unique users visited Facebook this month?
3. How many different pages link to each of the pages we have crawled?

Flajolet Martin Sketch

- Pick a hash function h that maps each of the n elements to at least $\log_2 n$ bits.
- For each stream element a, let r(a) be the number of trailing 0's in h(a)
  - For example say h(a) = 12, then 12 is 1100 in binary, so r(a) = 2
- Record R = the maximum r(a) seen
  - R = $\max_a r(a)$ over all the items a seen so far
- Estimate = 2^R

Intuition



Very very rough and heuristic intuition why Flajolet-Martin works:

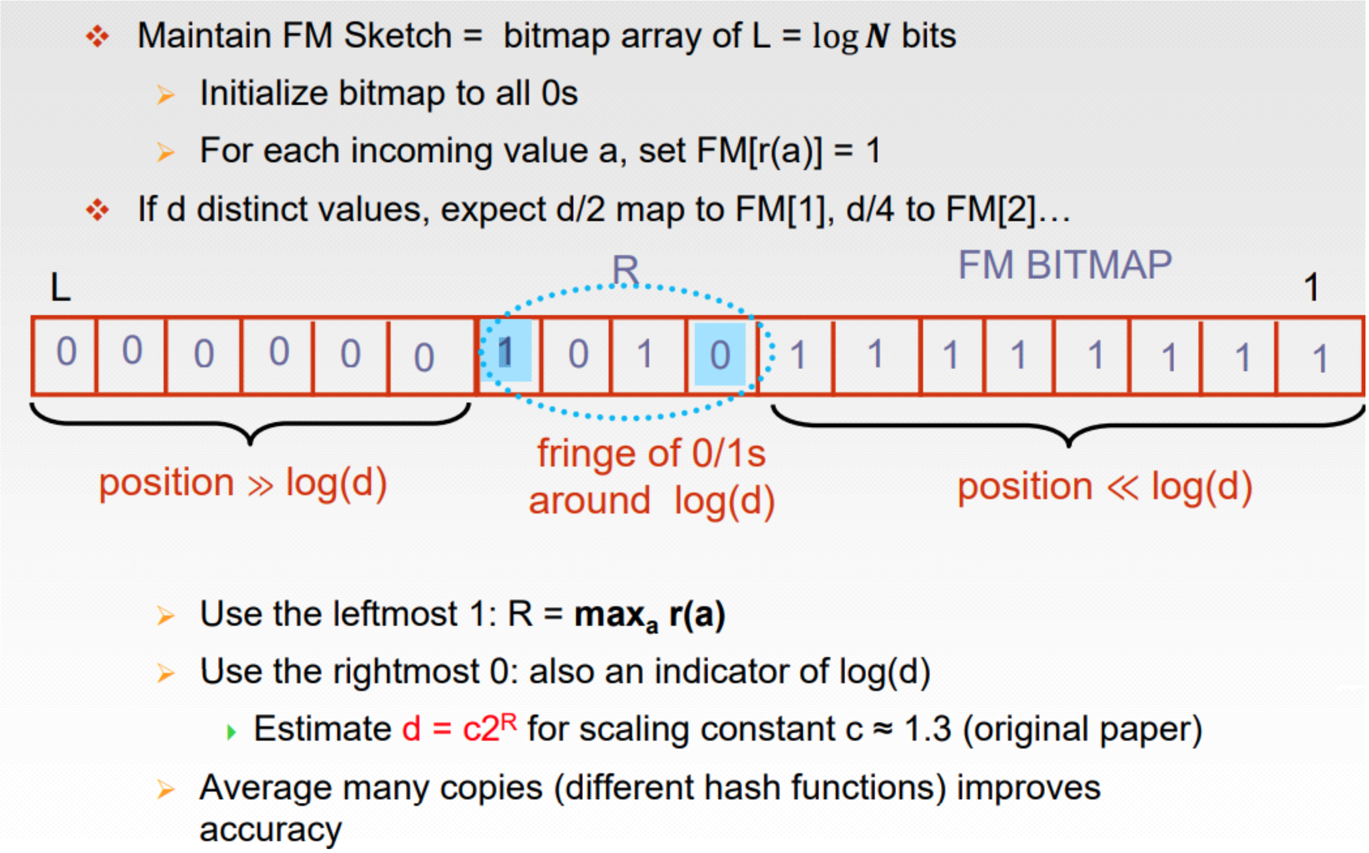
- ***h(a)*** hashes ***a*** with **equal prob.** to any of ***N*** values
- Then ***h(a)*** is a sequence of **$\log_2$ N** bits,
  where 2^{-r} fraction of all ***a***s have a tail of ***r*** zeros
  - About 50% of ***a***s hash to *****0**
  - About 25% of ***a***s hash to ****00**
  - So, if we saw the longest tail of ***r*=2** (i.e., item hash ending ***100**) then we have probably seen **about *4*** distinct items so far
- So, it takes to hash about 2^r items before we see one with zero-suffix of length *r*

```
Set Counter = 0
Set Max_R = 0
While counter !=  last element index:
     Val = binary of hash output of the current element
     Count no. of trailing zeroes in val:
     If Count > Max_R:
        Max_R = count
    Increment the counter
Return approximate Count of distinct elements: 2**(Max_R)
```

Consider stream, x = [1,5,10,5,15,1] and hash function h(x) = x mod 11, find the count of distinct/unique elements using the FM algorithm (answer is 4)

X	h(x)	Binary	Count of trailing 0's
1	1 mod (11) = 1	1	0
5	5	101	0
10	10	1010	1
15	4	100	2

In here, R = 2, the distinct element is 2^2 = 4



- Maintain FM Sketch = bitmap array of L = $\log N$ bits
    - Initialize bitmap to all 0s
    - For each incoming value a, set FM[r(a)] = 1
- If d distinct values, expect d/2 map to FM[1], d/4 to FM[2]…

- Use the leftmost 1: R = $\mathbf{max_a\ r(a)}$
- Use the rightmost 0: also an indicator of log(d)
    - Estimate d = $c2^R$ for scaling constant c ≈ 1.3 (original paper)
- Average many copies (different hash functions) improves accuracy

In this case, R = 12 (12th from the right), d = 1.3 * 2^12 = 5,324.8

7.1 Shingling

Thursday, November 2, 2023 12:36 PM

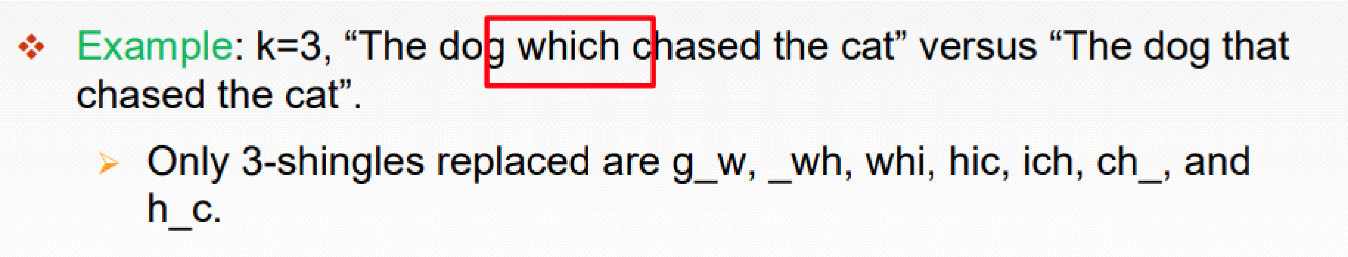
We want to find similar pairs in O(N) in high-dimension space, we can't afford the $O(N^2)$

Solution 1: Approximate Method

k-Shingling:

A sequence of k tokens that appears in the document, tokens can be characters or words

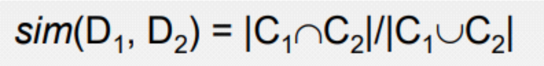
Convert documents to sets: For Example, k=2; document D1 = abcab, Set of 2-shingles: S(D1) = {ab, bc, ca}. This is 2-shingles



- Example: k=3, “The dog which chased the cat” versus “The dog that chased the cat”.
  - Only 3-shingles replaced are g_w, _wh, whi, hic, ich, ch_, and h_c.

Similarity Metric for Shingles

Jaccard Similarity



$$sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$

Jaccard Distance: 1 - Jaccard Similarity

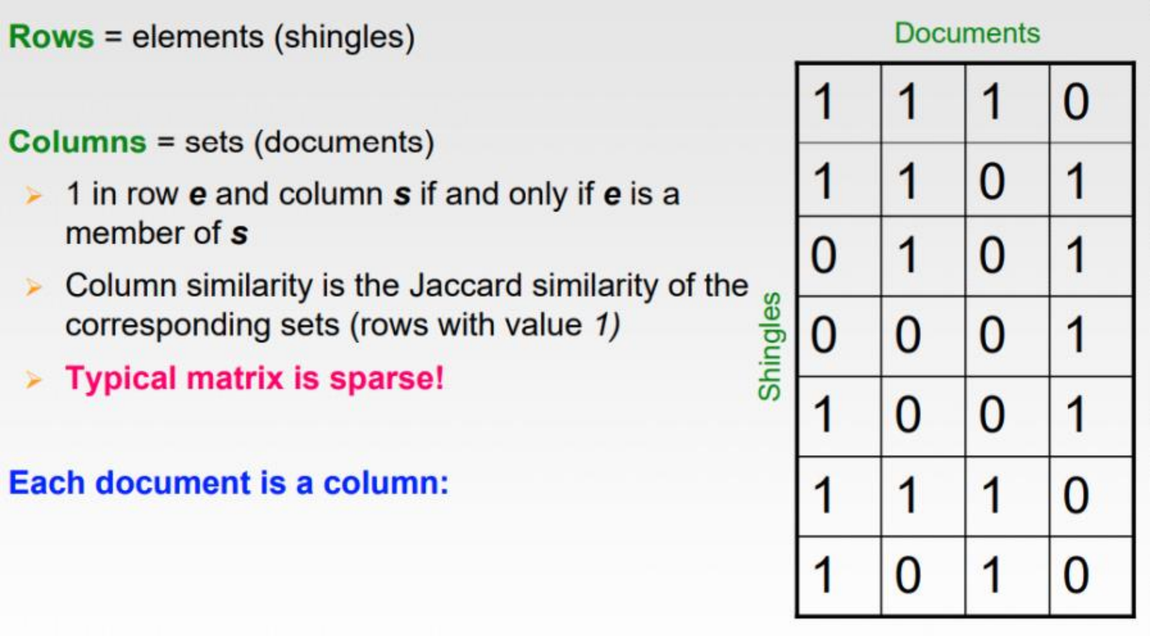
Example: C1 = 10111; C2 = 10011

Size of intersection = 3, size of union = 4

C1	1	0	1	1	1
C2	1	0	0	1	1

Jaccard similarity = 3 / 4 = 0.75

Jaccard distance = 1 - 3 / 4 = 0.25



Rows = elements (shingles)

Columns = sets (documents)

- 1 in row ***e*** and column ***s*** if and only if ***e*** is a member of ***s***
- Column similarity is the Jaccard similarity of the corresponding sets (rows with value *1)*
- **Typical matrix is sparse!**

Each document is a column:

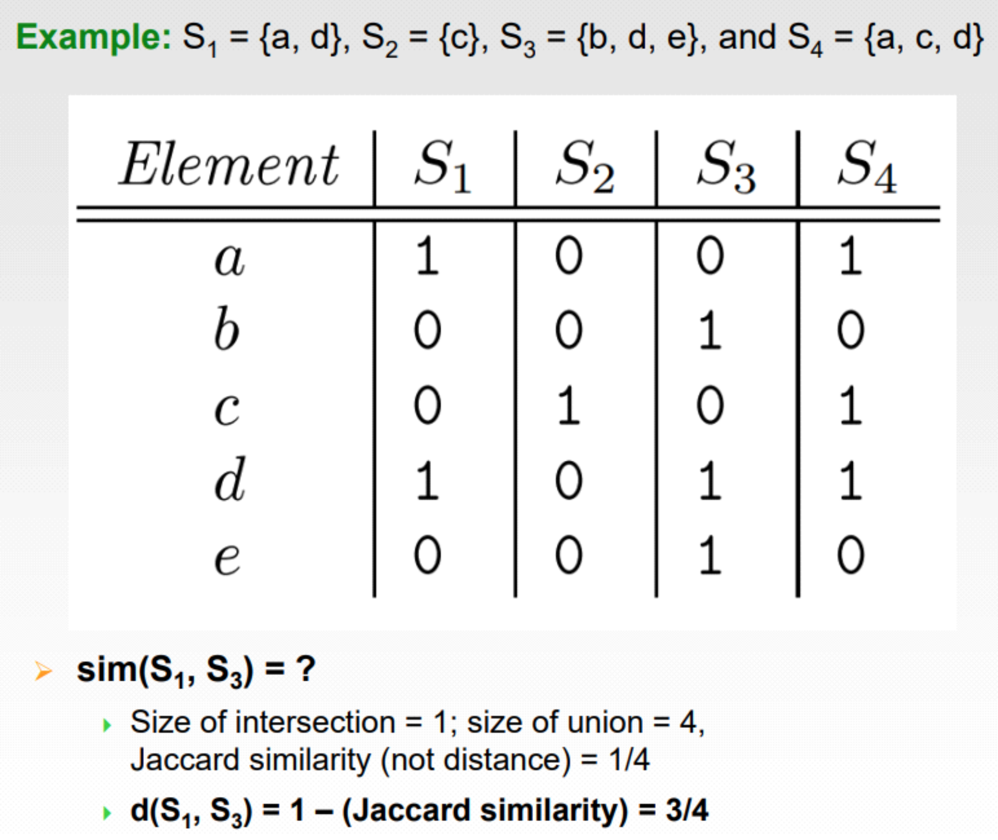
Documents

Shingles

1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

Each row is a shingle and each column is a document.

Working example



Example: S_1 = {a, d}, S_2 = {c}, S_3 = {b, d, e}, and S_4 = {a, c, d}

$Element$	S_1	S_2	S_3	S_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

- **sim(S_1, S_3) = ?**
  - Size of intersection = 1; size of union = 4, Jaccard similarity (not distance) = 1/4
  - **d(S_1, S_3) = 1 – (Jaccard similarity) = 3/4**

Exam Question:
2-Shingles S(A): {"the data", "data is", "is big", "big the", "the value", "value is", "is high"}
2-Shingles S(B): {"the value", "value in", "in the", "the data", "data is", "is high"}

Intersection(A, B) = {"the data", "the value", "data is", "is high"}
Union(A, B) = {"the data", "data is", "is big", "big the", "the value", "value is", "is high", "value in", "in

the"}

Jaccard Similarity (A, B) = 4/9

7.2 Min Hash

Friday, November 3, 2023 11:19 AM

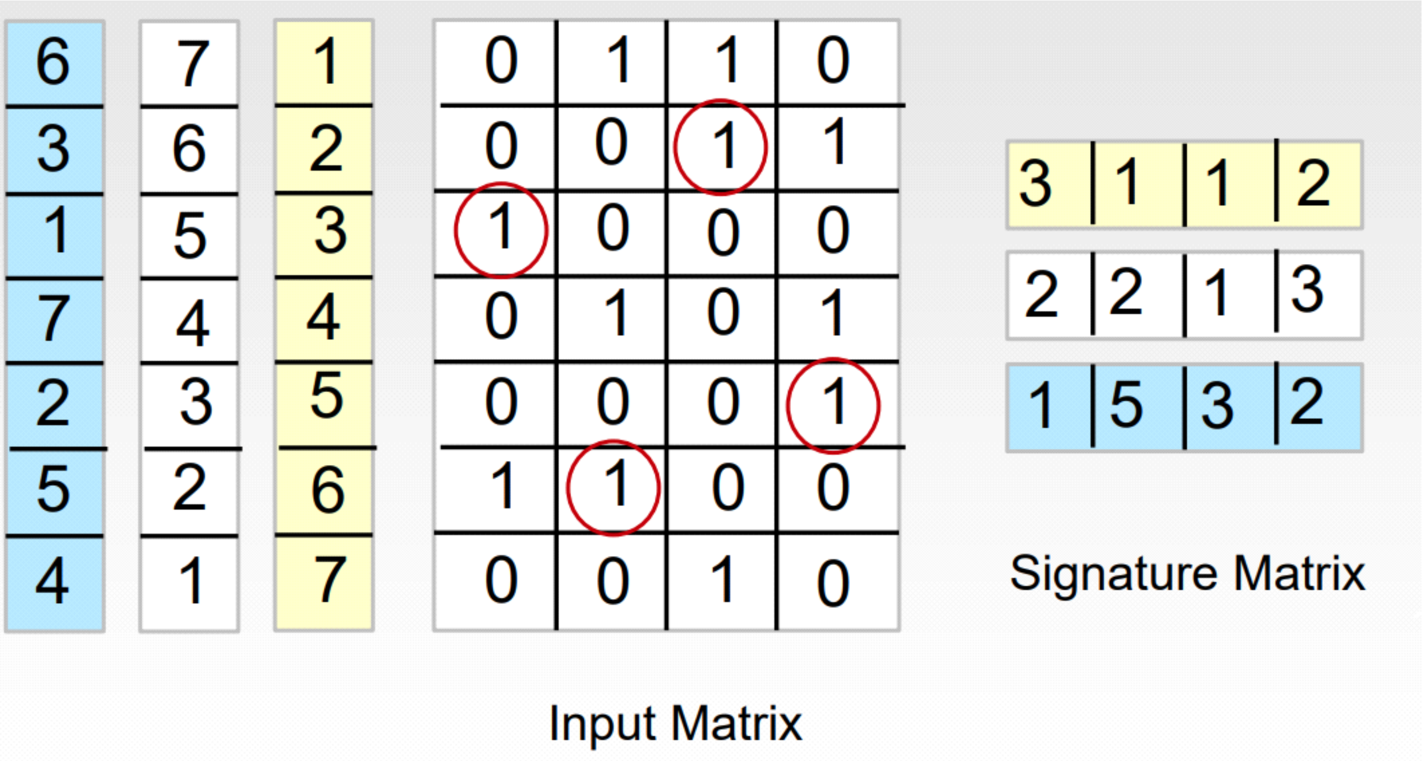
Min-Hashing

Covert large sets to short signatures, while preserving similarity

Our next goal is to compress the high dimensional boolean vectors into low dimensional signatures.

Right now the calculation takes too much space. We want to hash signatures (columns). The key idea is that, for hash function h, if sim(C1, C2) is high, then h(C1) = h(C2).

Working example:



6	7	1	0	1	1	0
3	6	2	0	0	1	1
1	5	3	1	0	0	0
7	4	4	0	1	0	1
2	3	5	0	0	0	1
5	2	6	1	1	0	0
4	1	7	0	0	1	0

Input Matrix

3	1	1	2
2	2	1	3
1	5	3	2

Signature Matrix

Question 4. Finding Similar Items (6 marks)

Given three documents A = (“the data is big the value is high”) and B = (“the value in the data is high”), using the ***WORDS*** as tokens, answer the following questions.

(i) (2 marks) compute the 2-shingles for A and B, and then compute their Jaccard similarity based on their 2-shingles.

(ii) (4 marks) We want to compute min-hash signature for the two documents A and B given in Question (i), using three pseudo-random permutations of columns defined by the following functions:

h1(n) = (2n + 1) mod M

h2(n) = (7n + 2) mod M

h3(n) = (5n + 3) mod M

Here, n is the row number in original ordering of the 2-shingles (**according to their occurrences in A then B**), and M is the number of all 2-shingles you have computed from A and B. Instead of explicitly reordering the columns for each hash function, we use the implementation discussed in class. Complete the steps of the algorithm and give the resulting signatures for A and B.

i)
2-Shingles S(A): {"the data", "data is", "is big", "big the", "the value", "value is", "is high"}
2-Shingles S(B): {"the value", "value in", "in the", "the data", "data is", "is high"}

Intersection(A, B) = {"the data", "the value", "data is", "is high"}

Union(A,B) = {"the data", "data is", "is big", "big the", "the value", "value is", "is high", "value in", "in the"}

Jacarrd Similarity is 4/9

ii)

h(1)	h(2)	h(3)	row	shingle	A	B
(2*0+1) mod 9 = 1	(7 * 0 + 2) mod 9 = 2	(5 * 0 + 3) mod 9 = 3	0	the data	1	1
(2*1+1) mod 9 = 3	(7 * 1 + 2) mod 9 = 0	(5 * 1 + 3) mod 9 = 8	1	data is	1	1

(2*2+1) mod 9 = 5	(7 * 2 + 2) mod 9 = 7	(5 * 2 + 3) mod 9 = 4	2	is big	1	0
(2*3+1) mod 9 = 7	(7 * 3 + 2) mod 9 = 5	(5 * 3 + 3) mod 9 = 0	3	big the	1	0
(2*4+1) mod 9 = 0	(7 * 4 + 2) mod 9 = 3	(5 * 4 + 3) mod 9 = 5	4	the value	1	1
(2*5+1) mod 9 = 2	(7 * 5 + 2) mod 9 = 1	(5 * 5 + 3) mod 9 = 1	5	value is	1	0
(2*6+1) mod 9 = 4	(7 * 6 + 2) mod 9 = 8	(5 * 6 + 3) mod 9 = 6	6	is high	1	1
(2*7+1) mod 9 = 6	(7 * 7 + 2) mod 9 = 6	(5 * 7 + 3) mod 9 = 2	7	value in	0	1
(2*8+1) mod 9 = 8	(7 * 8 + 2) mod 9 = 4	(5 * 8 + 3) mod 9 = 7	8	in the	0	1

Signatures:

	A	B
h1(7)	0	0
h2(7)	0	0
h3(7)	0	2

h(1)	h(2)	h(3)	row	shingle	A	B
(2*1+1) mod 9 = 3	(7 * 1 + 2) mod 9 = 0	(5 * 1 + 3) mod 9 = 8	1	the data	1	1
(2*2+1) mod 9 = 5	(7 * 2 + 2) mod 9 = 7	(5 * 2 + 3) mod 9 = 4	2	data is	1	1
(2*3+1) mod 9 = 7	(7 * 3 + 2) mod 9 = 5	(5 * 3 + 3) mod 9 = 0	3	is big	1	0
(2*4+1) mod 9 = 0	(7 * 4 + 2) mod 9 = 3	(5 * 4 + 3) mod 9 = 5	4	big the	1	0
(2*5+1) mod 9 = 2	(7 * 5 + 2) mod 9 = 1	(5 * 5 + 3) mod 9 = 1	5	the value	1	1
(2*6+1) mod 9 = 4	(7 * 6 + 2) mod 9 = 8	(5 * 6 + 3) mod 9 = 6	6	value is	1	0
(2*7+1) mod 9 = 6	(7 * 7 + 2) mod 9 = 6	(5 * 7 + 3) mod 9 = 2	7	is high	1	1
(2*8+1) mod 9 = 8	(7 * 8 + 2) mod 9 = 4	(5 * 8 + 3) mod 9 = 7	8	value in	0	1
(2*9+1) mod 9 = 1	(7 * 9 + 2) mod 9 = 2	(5 * 9 + 3) mod 9 = 3	9	in the	0	1

Signatures:

	A	B
h1(5)	0	1
h2(5)	0	0
h3(5)	0	1

Class example:

row	h(1)	h(2)	A	B	C	D
0	(0 + 1) mod 5 = 1	(3 * 0 + 1) mod 5 = 1	1	0	0	1
1	(1 + 1) mod 5 = 2	(3 * 1 + 1) mod 5 = 4	0	0	1	0
2	(2 + 1) mod 5 = 3	(3 * 2 + 1) mod 5 = 2	0	1	0	1
3	(3 + 1) mod 5 = 4	(3 * 3 + 1) mod 5 = 0	1	0	1	1
4	(4 + 1) mod 5 = 0	(3 * 4 + 1) mod 5 = 3	0	0	1	0

Signatures:

	A	B	C	D
h1	∞	∞	∞	∞
h2	∞	∞	∞	∞

Row = 0

h1	1	∞	∞	1
h2	1	∞	∞	1

Row = 1

h1	1	∞	2	1
h2	1	∞	4	1

Row = 2

h1	1	3	2	1
h2	1	2	4	1

Row = 3

h1	1	3	2	1
h2	0	2	0	0

Row = 3

h1	1	3	0	1
h2	0	2	0	0

You can also observe the table to get this matrix

row	h(1)	h(2)	A	B	C	D
1	(1 + 1) mod 5 = 2	(3 * 1 + 1) mod 5 = 4	1	0	0	1
2	(2 + 1) mod 5 = 3	(3 * 2 + 1) mod 5 = 2	0	0	1	0
3	(3 + 1) mod 5 = 4	(3 * 3 + 1) mod 5 = 0	0	1	0	1
4	(4 + 1) mod 5 = 0	(3 * 4 + 1) mod 5 = 3	1	0	1	1

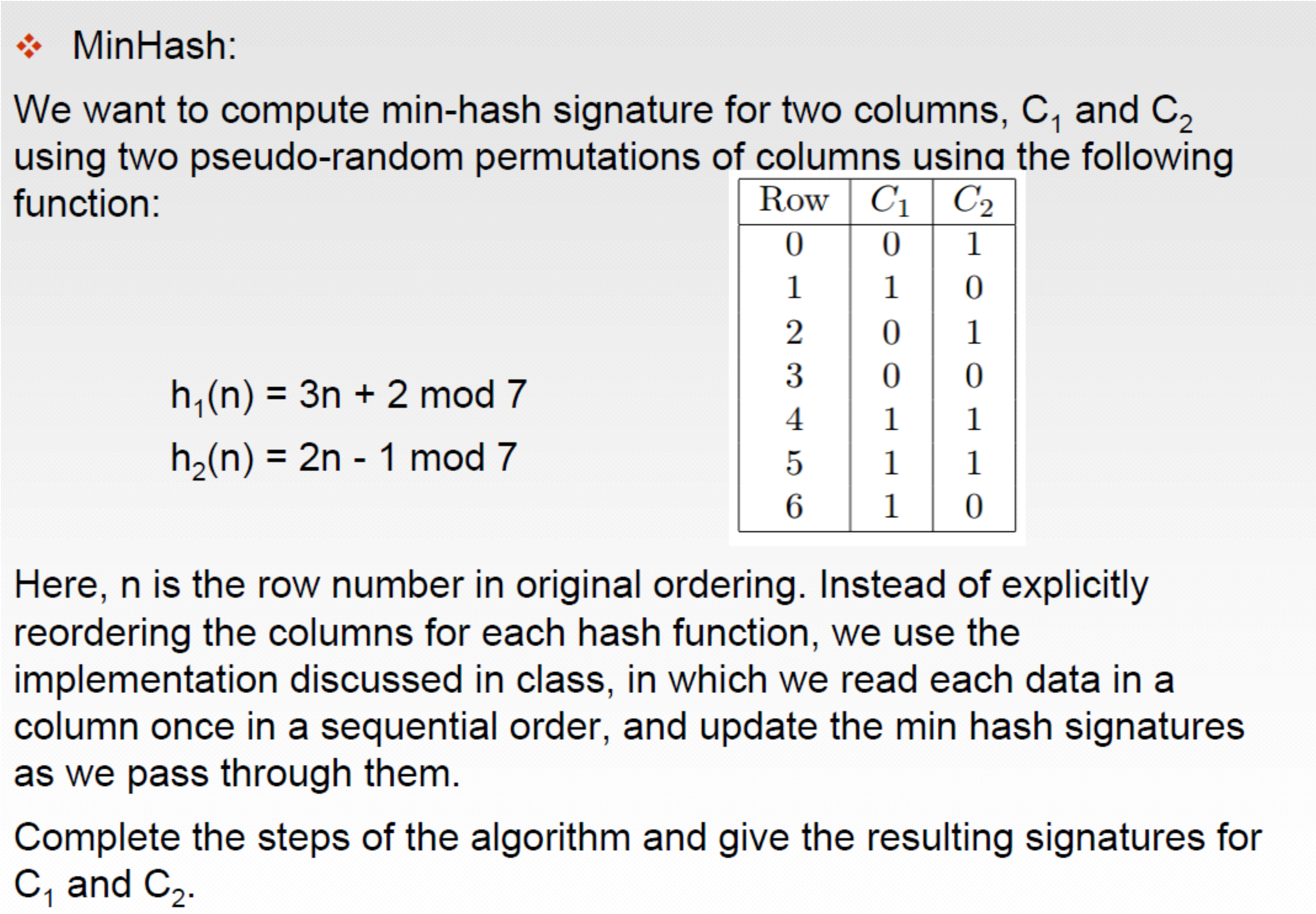
5	(5 + 1) mod 5 = 1	(3 * 5 + 1) mod 5 = 1	0	0	1	0

	A	B	C	D
h1	0	4	0	0
h2	3	0	1	0

Claim: the probability of two signatures are the same is equal to the similarity of two columns

Focus on pairs of signatures likely to be from similar documents
Imagine the rows of the boolean matrix permuted under random permutation π

- ❖ Define a "hash" function hπ(C) = the index of the first (in the permuted order π) row in which column C has value 1:
hπ(C) = minπ π(C)
- ❖ Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column



❖ MinHash:

We want to compute min-hash signature for two columns, C_1 and C_2 using two pseudo-random permutations of columns using the following function:

$h_1(n) = 3n + 2 \bmod 7$

$h_2(n) = 2n - 1 \bmod 7$

Row	C_1	C_2
0	0	1
1	1	0
2	0	1
3	0	0
4	1	1
5	1	1
6	1	0

Here, n is the row number in original ordering. Instead of explicitly reordering the columns for each hash function, we use the implementation discussed in class, in which we read each data in a column once in a sequential order, and update the min hash signatures as we pass through them.

Complete the steps of the algorithm and give the resulting signatures for C_1 and C_2.

Signatures:

h1	h2	Row	C1	C2
2	6	0	0	1
5	1	1	1	0
1	3	2	0	1
4	5	3	0	0
0	0	4	1	1
3	2	5	1	1
6	4	6	1	0

	C1	C2
h1(0)	∞	2
h2(0)	∞	6

	C1	C2
h1(1)	5	2
h2(1)	1	6

	C1	C2
h1(2)	5	1
h2(2)	1	3

	C1	C2
h1(3)	5	1
h2(3)	1	3

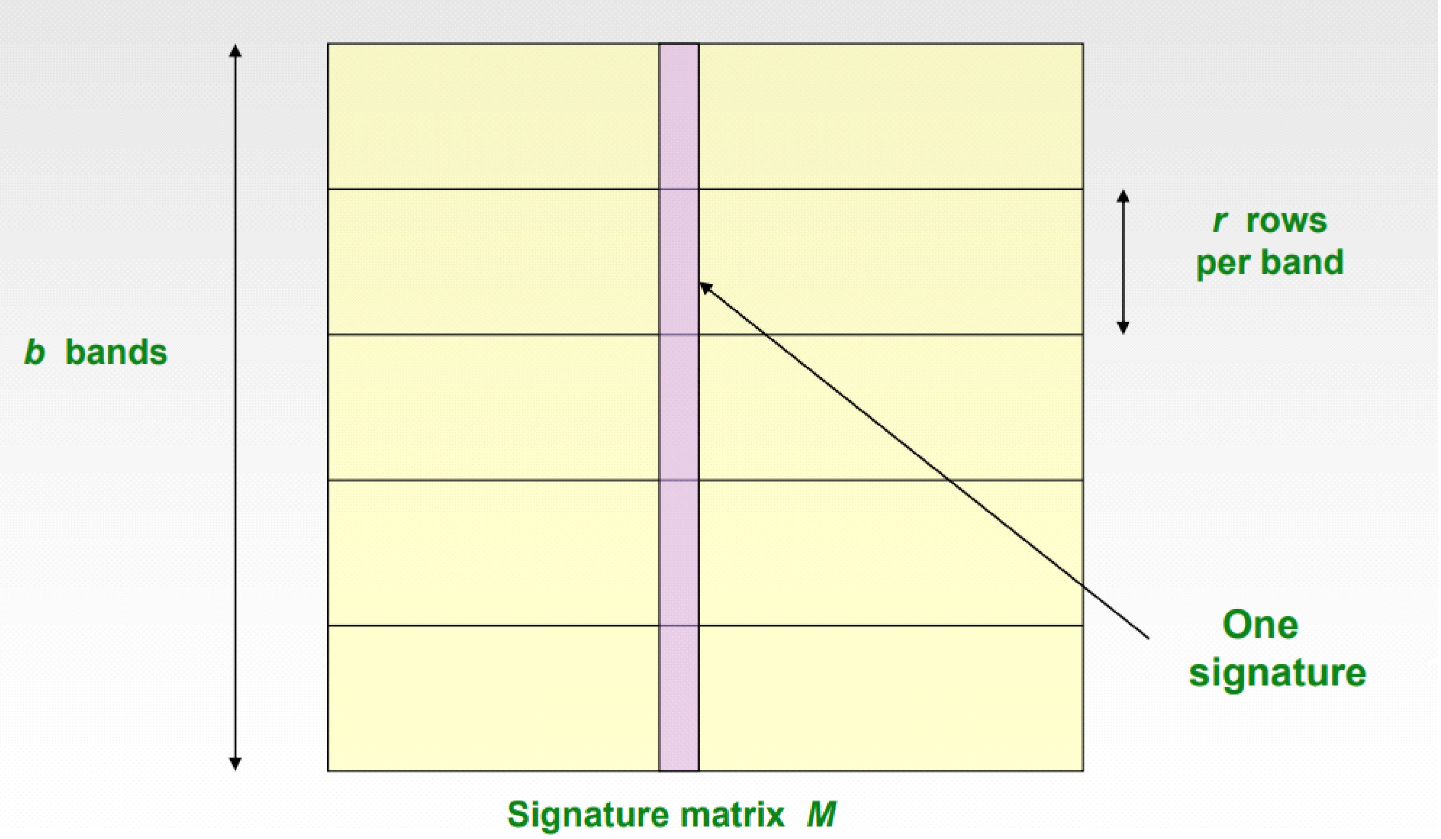
	C1	C2
h1(4)	0	0
h2(4)	0	0

7.3 Locality-sensitive Hashing

Friday, November 3, 2023 11:19 AM

Goal: Find documents with Jaccard similarity at least s (for some similarity threshold, e.g. s = 0.8 (Similar to project question)

LSH: Use a function f(x,y) that tells whether x and y is a candidate pair. LSH is also a hash function. If two signatures fall under the same bucket, then they are candidate pairs.



Signature matrix *M*

Algorithm:

1. Divide matrix M into b bands of r rows
2. For each band, hash its portion of each column to a hash table with k buckets
3. Candidate column pairs are those that hash to the same bucket for >= 1 band
4. Tune b and r to catch most similar pairs

Why does LSH work?

Example:

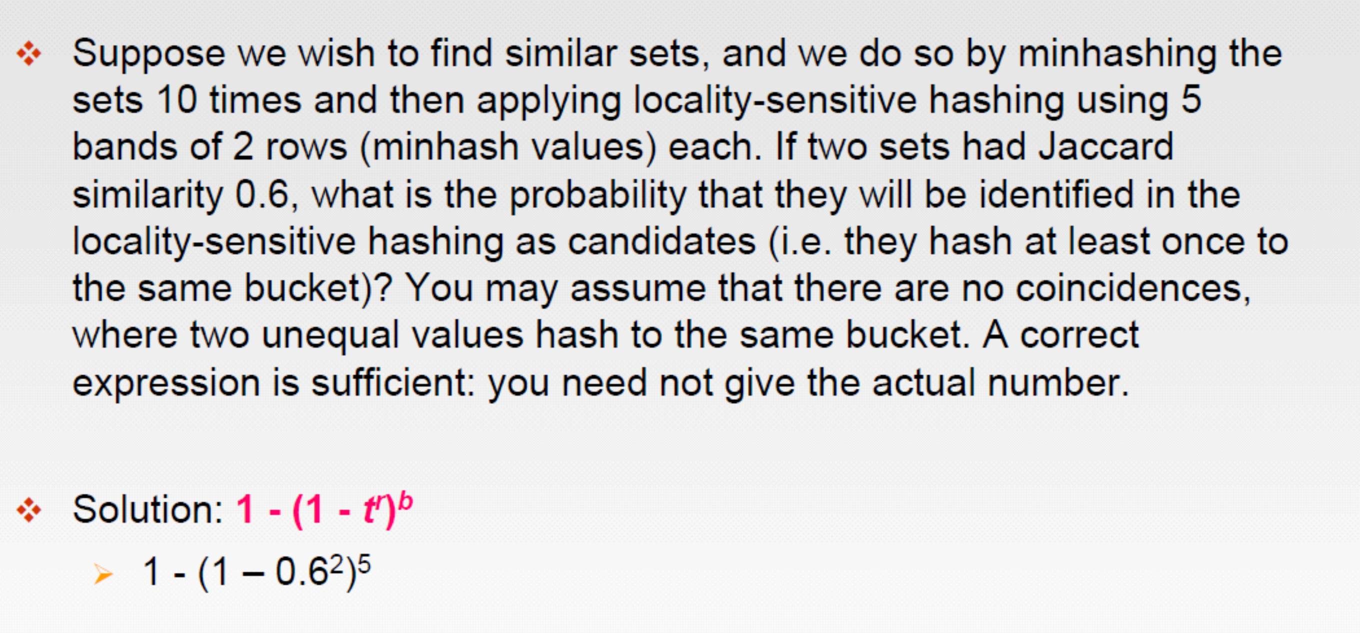
- Suppose 100,000 columns of M (100k docs)
- Signatures of 100 integers (rows)
- Signatures take 40 Mb, choose b = 20 bands or r = 5 rows

Goal is to find pairs of documents that are at least S(C1,C2) = 0.8 similar

Since b = 20 and r = 5, probability that C1 and C2 identical in one particular band is 0.8^5 = 0.3277
The probability that this pair is returned = 1 - (1 - 0.3277) ^ 20 = 0.9996

When S(C1,C2) = 0.3, (0.3)^5 = 0.0024 is the probability that they are identical in particular band
Probability that C1, C2 identical in at least 1 of 20 bands: 1 - (1- 0.0024)^20 = 0.0469.
Therefore, LSH is more likely to return candidate pairs when C1 and C2 are similar

Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures. Check in main memory that candidate pairs really do have similar signatures



- Suppose we wish to find similar sets, and we do so by minhashing the sets 10 times and then applying locality-sensitive hashing using 5 bands of 2 rows (minhash values) each. If two sets had Jaccard similarity 0.6, what is the probability that they will be identified in the locality-sensitive hashing as candidates (i.e. they hash at least once to the same bucket)? You may assume that there are no coincidences, where two unequal values hash to the same bucket. A correct expression is sufficient: you need not give the actual number.

- Solution: **$1 - (1 - t^r)^b$**
  - $1 - (1 – 0.6^2)^5$

8.1.1 Graph in Map Reduce

Friday, November 10, 2023 9:49 AM

Definition

G = (V, E)

- V represents the set of vertices,
- E represents the set of edges,
- Both vertices and edges may contain additional information.

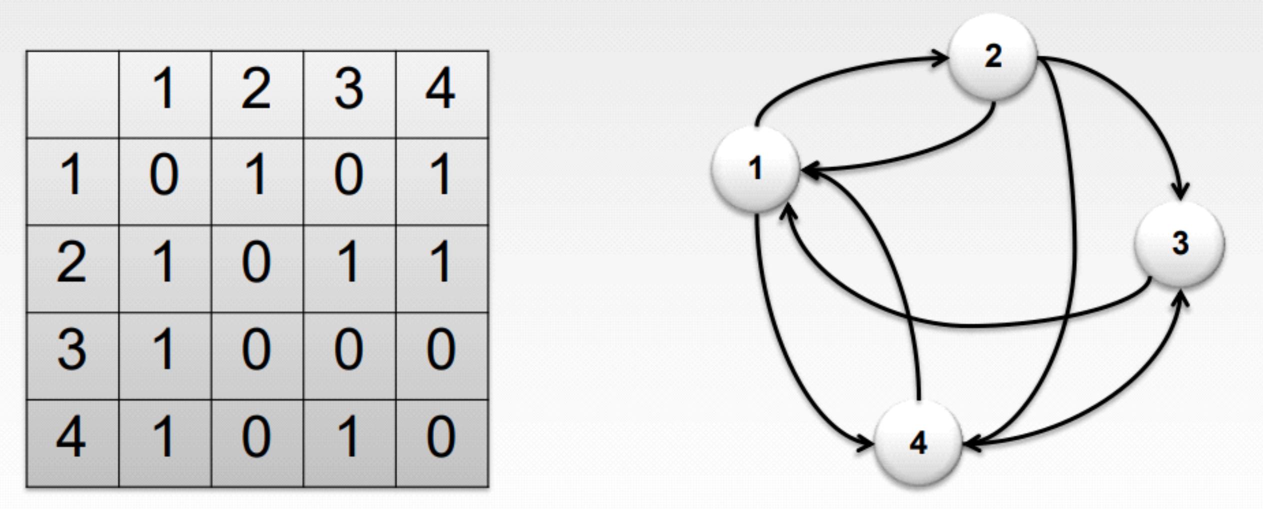
Difference Types of Graphs:

Directed vs undirected edges

How do you represent graph data in MapReduce?

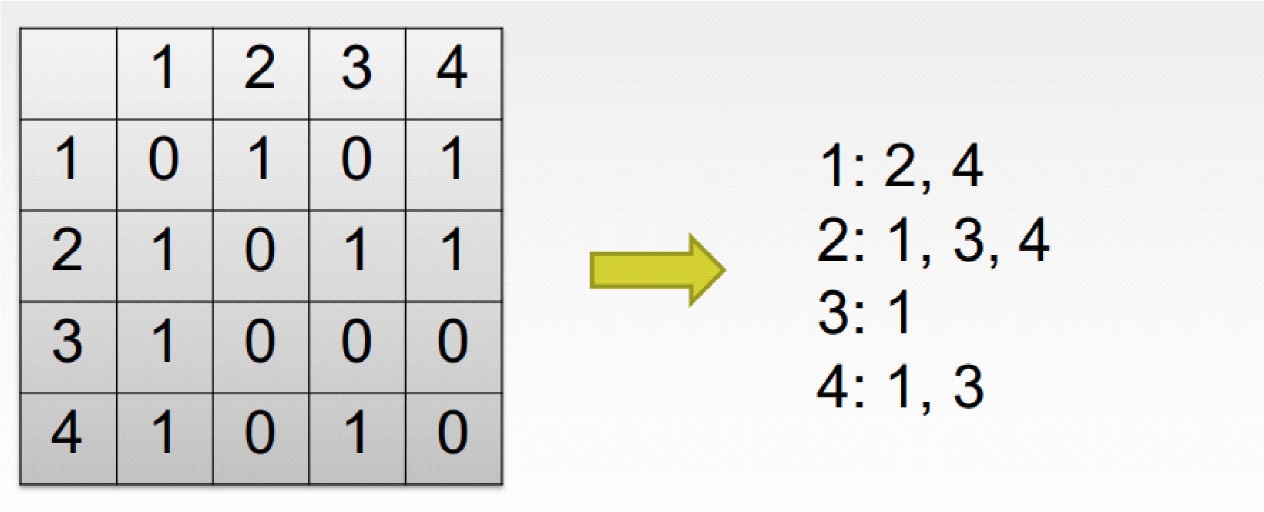
Adjacency Matrices: represent a graph as an n by n square matrix M where

- n = |V|
- M_{ij} = 1 means a link from node i to j



	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

Adjacency Lists: Take adjacency matrices, and throw away all the zeros

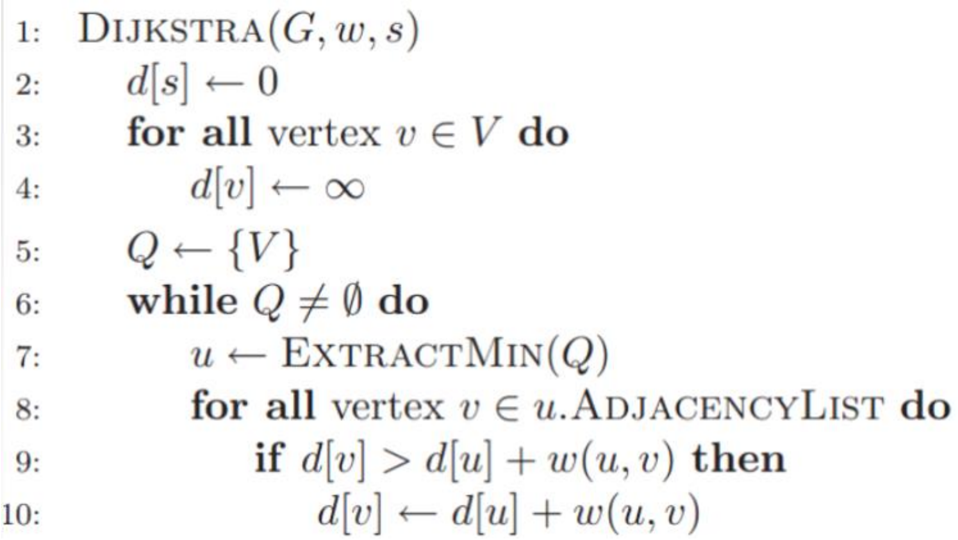


	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

How do you traverse a graph in MapReduce?

Single Source Shortest Path



```
DIJKSTRA($G, w, s$)
    $d[s] \leftarrow 0$
    for all vertex $v \in V$ do
        $d[v] \leftarrow \infty$
    $Q \leftarrow \{V\}$
    while $Q \neq \emptyset$ do
        $u \leftarrow$ EXTRACTMIN($Q$)
        for all vertex $v \in u$.ADJACENCYLIST do
            if $d[v] > d[u] + w(u, v)$ then
                $d[v] \leftarrow d[u] + w(u, v)$
```

Dijkstra's Algorithm cannot be applied on graphs having negative weight function because calculation of cost to reach a destination node from the source node becomes complex.

Map Reduce (what is key/value, what is map/reducer)

➢ Define: b is reachable from a if b is on adjacency list of a

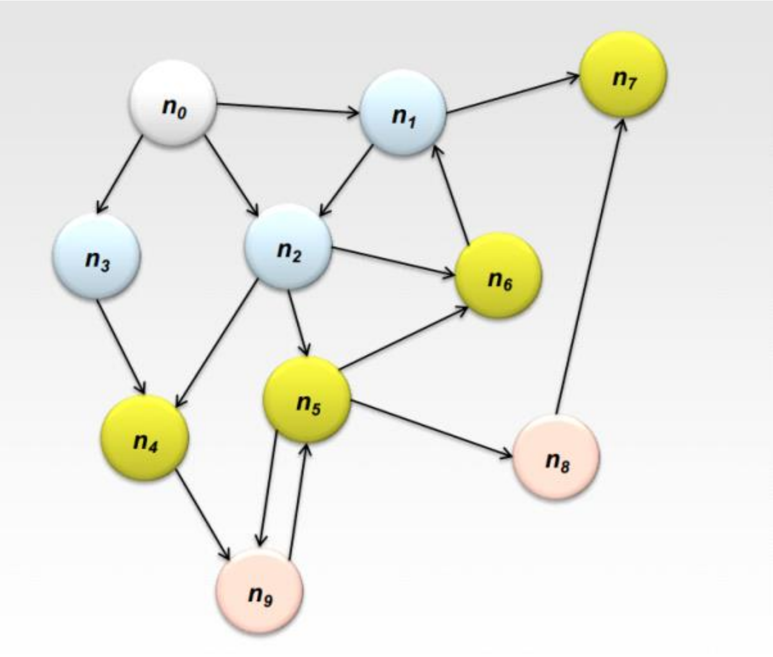
- DISTANCETO(s) = 0
- For all nodes p reachable from s, DISTANCETO(p) = 1
- For all nodes n reachable from some other set of nodes M, DISTANCETO(n) = 1 + min(DISTANCETO(m)), $x \in A$

```
data representation
  1. key: node n
  2. value d (distance from start), adjacency list (list of nodes reachable from n)
  3. initialization: for all nodes except for start node d = inf+
Mapper:
    While there exists m in the adjacency list: emit(m, d + 1)
```

```
Sort/Shuffle
     Groups distance by reachable nodes
Reducer:
     Select minimum distance path for each reachable node
```

For example in the mapper phase

For the below graph,



if we are looking at n2 node

n2, d2

For all the nodes connected to n2, we will have

(n4, d2 + 1)

(n5, d2 + 1)

(n6, d2 + 1)

If we are looking at n3 node

n3, d3

(n4, d3 + 1)

Then in the reducer phase, we can compare d3 + 1 and d2 + 1 and return the minimum distance

How to obtain paths? (we need additional storage space to keep track of the shortest path to every node)

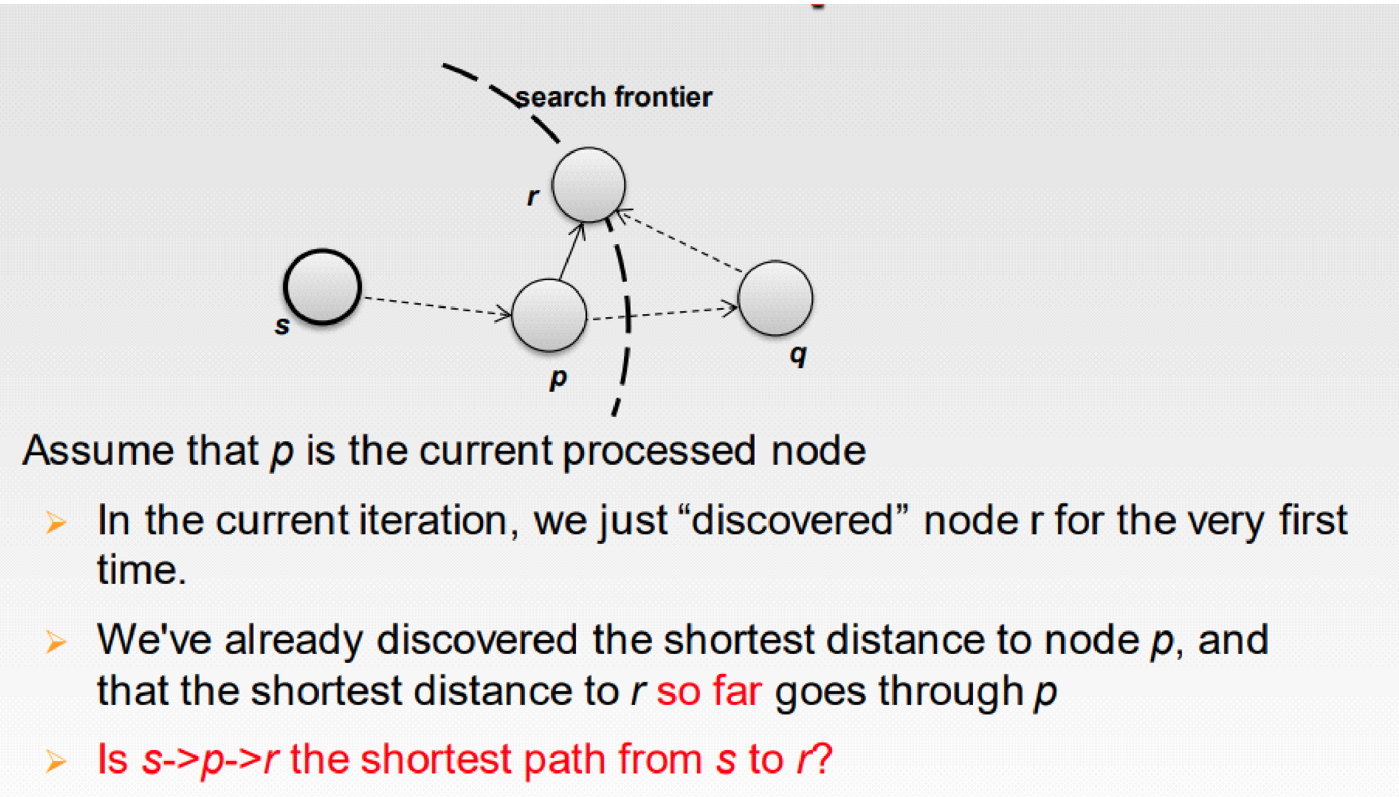
How many iterations do we need? When do we stop? (six degree of seperation, or use counter to keep track of the number of infinity)

```
class Mapper:
     method Map(nid n, node N)
     d = N.distance
     Emit(nid n, N.adjacencyList)
     for all node id m in N.adjacencyList do
          Emit(nid m, d + weight) // How to deal with weighted edges
```

```
class Reducer:
     method Reduce(nid, [d1, d2, …])
     $d_{min}$ = +inf
     M = ∅
     for all d in counts [d1, d2, …] do:
          if isNode(d) then
               M.adjacency list = d
          else if d < $d_{min}$ then
               $d_{min}$ = d
```

```
    M.distance = d_min
    Emit(nid m, node M) // store this node in some kind of data structure
```

For general cases, additional complexity is that, your shortest path can be outside of your search frontier whilst for equal weight BFS, when you stop, the path is guaranteed to be the shortest



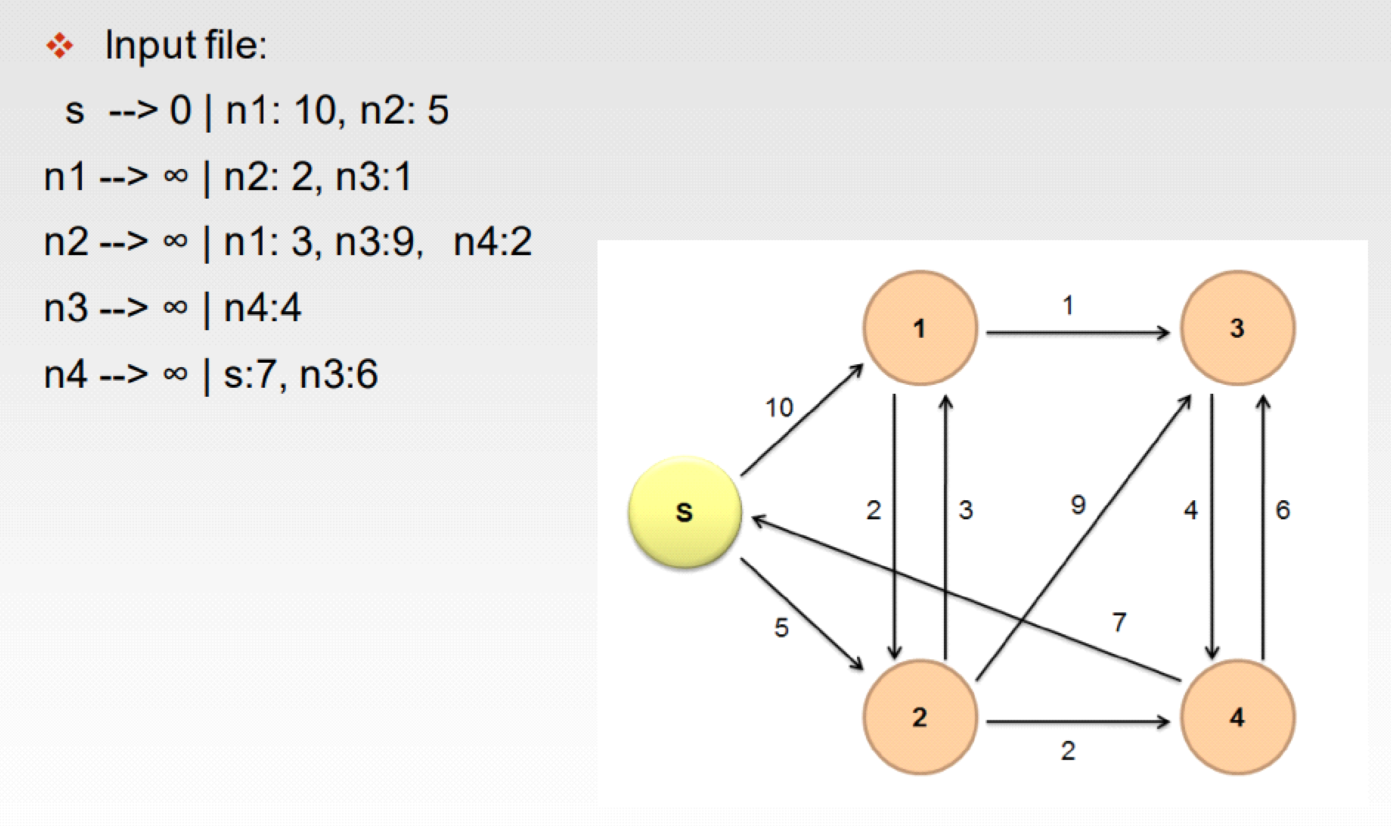
Assume that *p* is the current processed node

- In the current iteration, we just "discovered" node r for the very first time.
- We've already discovered the shortest distance to node *p*, and that the shortest distance to *r* so far goes through *p*
- Is *s*->*p*->*r* the shortest path from *s* to *r*?

8.1.2 Single Source Shortest Path (SSSP)

Thursday, November 30, 2023 12:32 PM

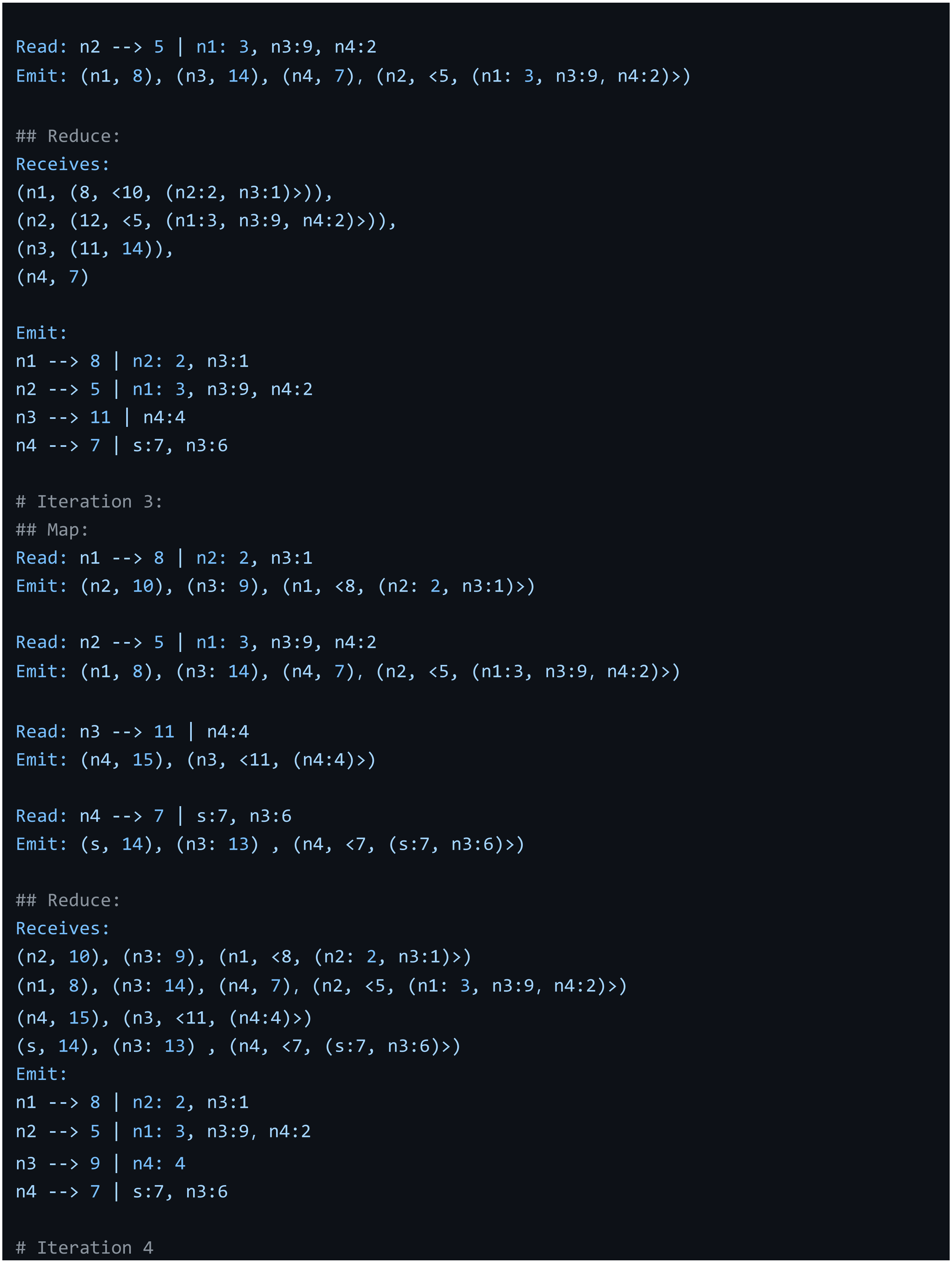
Example



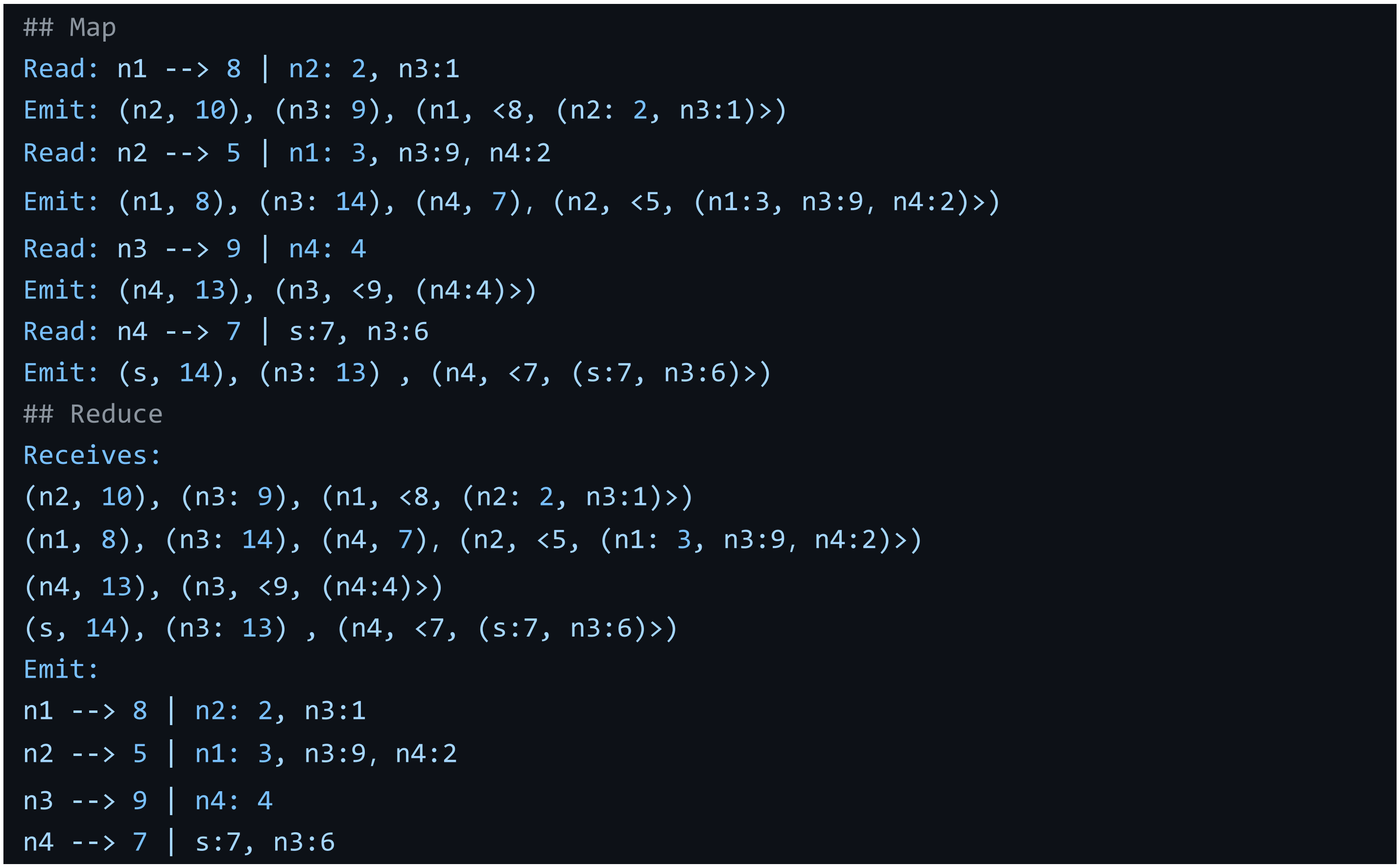
- Input file:

s --> 0 | n1: 10, n2: 5
n1 --> ∞ | n2: 2, n3:1
n2 --> ∞ | n1: 3, n3:9, n4:2
n3 --> ∞ | n4:4
n4 --> ∞ | s:7, n3:6

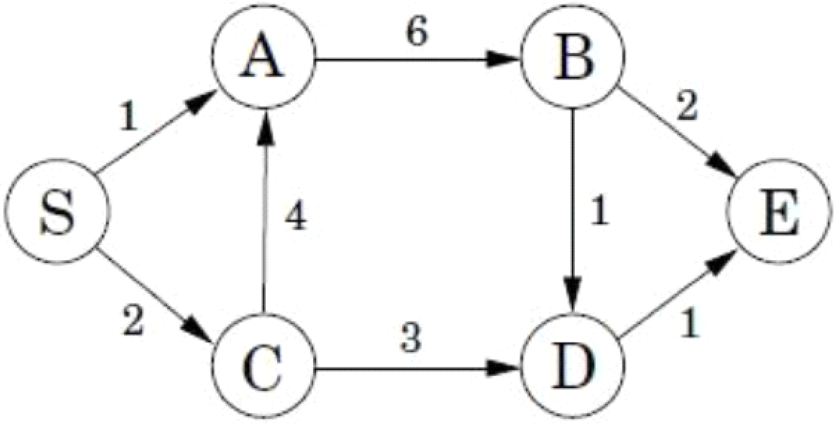
```
Example

# Iteration 1:
## Map
Read s -> 0 | n1: 10, n2: 5
Emit: (n1, 10), (n2, 5), and the adjacency list is (s, n1: 10, n2: 5)
The other lists will also be read and emit, but they do not contribute, and
thus ignored
## Reduce
Receives: (n1, 10), (n2, 5), (s, <n1: 10, n2: 5>)
Emit:
s --> 0 | n1: 10, n2: 5
n1 --> 10 | n2: 2, n3:1
n2 --> 5 | n1: 3, n3:9, n4:2

# Iteration 2:
## Map
Read: n1 --> 10 | n2: 2, n3:1
Emit: (n2, 12), (n3, 11), (n1, <10, (n2: 2, n3:1)>)
```



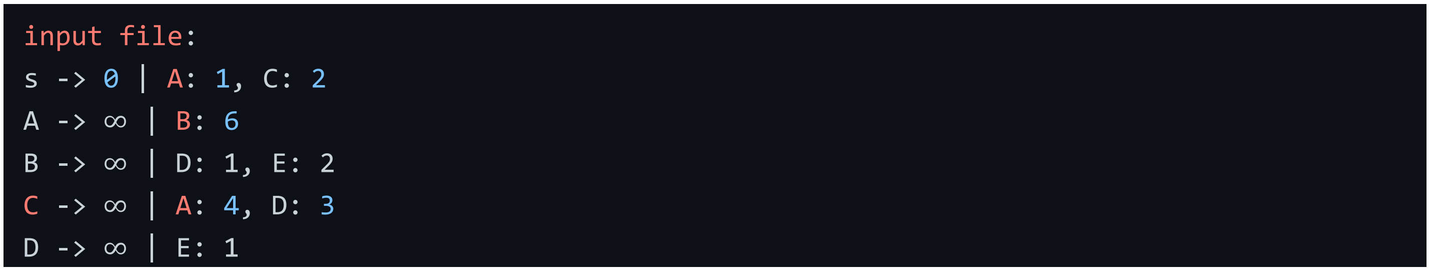
```
Read: n2 --> 5 | n1: 3, n3:9, n4:2
Emit: (n1, 8), (n3, 14), (n4, 7), (n2, <5, (n1: 3, n3:9, n4:2)>)

## Reduce:
Receives:
(n1, (8, <10, (n2:2, n3:1)>)),
(n2, (12, <5, (n1:3, n3:9, n4:2)>)),
(n3, (11, 14)),
(n4, 7)

Emit:
n1 --> 8 | n2: 2, n3:1
n2 --> 5 | n1: 3, n3:9, n4:2
n3 --> 11 | n4:4
n4 --> 7 | s:7, n3:6

# Iteration 3:
## Map:
Read: n1 --> 8 | n2: 2, n3:1
Emit: (n2, 10), (n3: 9), (n1, <8, (n2: 2, n3:1)>)

Read: n2 --> 5 | n1: 3, n3:9, n4:2
Emit: (n1, 8), (n3: 14), (n4, 7), (n2, <5, (n1:3, n3:9, n4:2)>)

Read: n3 --> 11 | n4:4
Emit: (n4, 15), (n3, <11, (n4:4)>)

Read: n4 --> 7 | s:7, n3:6
Emit: (s, 14), (n3: 13) , (n4, <7, (s:7, n3:6)>)

## Reduce:
Receives:
(n2, 10), (n3: 9), (n1, <8, (n2: 2, n3:1)>)
(n1, 8), (n3: 14), (n4, 7), (n2, <5, (n1: 3, n3:9, n4:2)>)
(n4, 15), (n3, <11, (n4:4)>)
(s, 14), (n3: 13) , (n4, <7, (s:7, n3:6)>)
Emit:
n1 --> 8 | n2: 2, n3:1
n2 --> 5 | n1: 3, n3:9, n4:2
n3 --> 9 | n4: 4
n4 --> 7 | s:7, n3:6

# Iteration 4
```



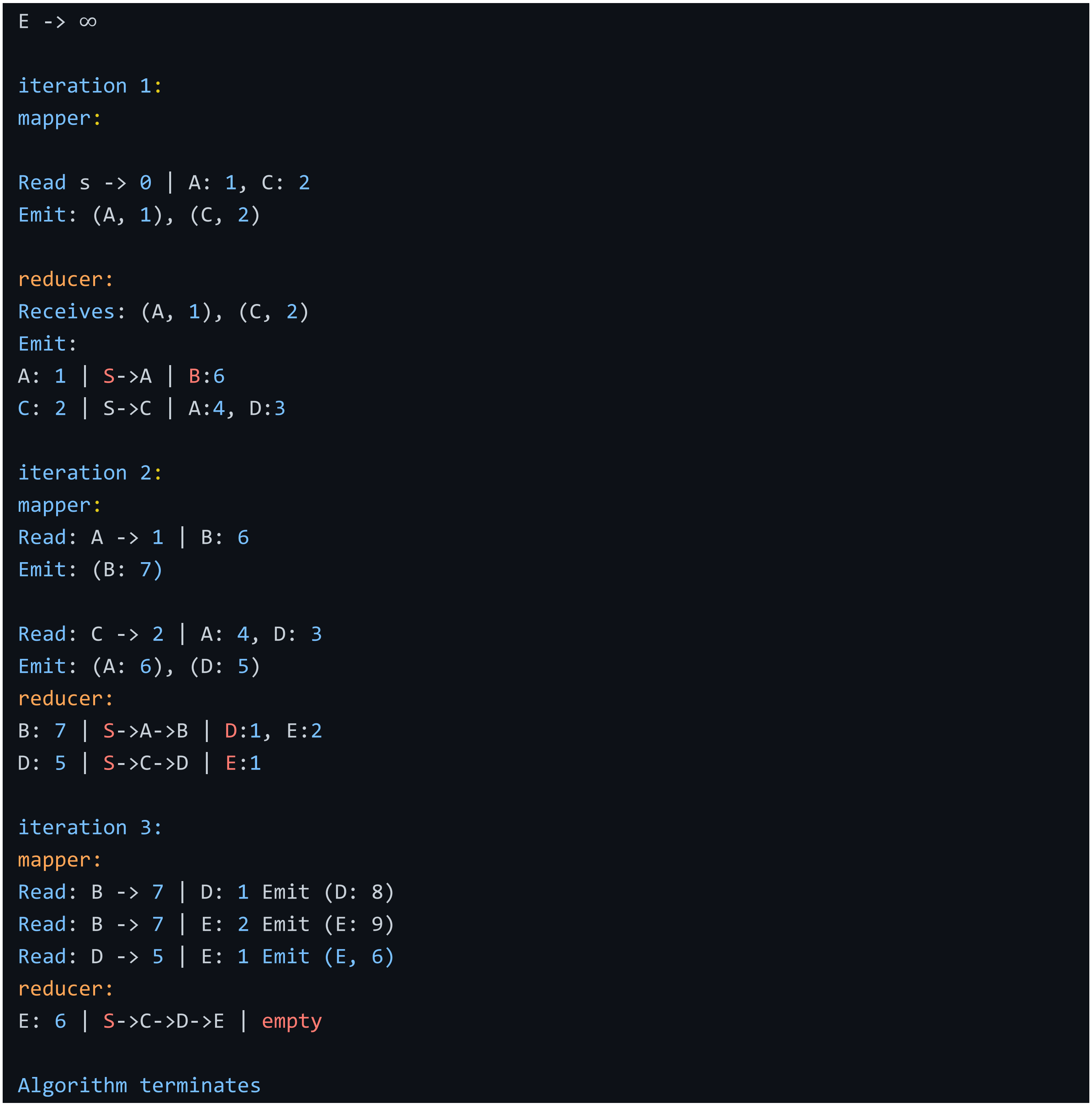
```
## Map
Read: n1 --> 8 | n2: 2, n3:1
Emit: (n2, 10), (n3: 9), (n1, <8, (n2: 2, n3:1)>)
Read: n2 --> 5 | n1: 3, n3:9, n4:2
Emit: (n1, 8), (n3: 14), (n4, 7), (n2, <5, (n1:3, n3:9, n4:2)>)
Read: n3 --> 9 | n4: 4
Emit: (n4, 13), (n3, <9, (n4:4)>)
Read: n4 --> 7 | s:7, n3:6
Emit: (s, 14), (n3: 13) , (n4, <7, (s:7, n3:6)>)
## Reduce
Receives:
(n2, 10), (n3: 9), (n1, <8, (n2: 2, n3:1)>)
(n1, 8), (n3: 14), (n4, 7), (n2, <5, (n1: 3, n3:9, n4:2)>)
(n4, 13), (n3, <9, (n4:4)>)
(s, 14), (n3: 13) , (n4, <7, (s:7, n3:6)>)
Emit:
n1 --> 8 | n2: 2, n3:1
n2 --> 5 | n1: 3, n3:9, n4:2
n3 --> 9 | n4: 4
n4 --> 7 | s:7, n3:6
```



Given the following graph, assume that you are using the single shortest path algorithm to compute the shortest path from node S to node E. Show the output of the mapper (sorted results of all mappers) and the reducer (only one reducer used) in each iteration (including both the distances and the paths).



```
input file:
s -> 0 | A: 1, C: 2
A -> ∞ | B: 6
B -> ∞ | D: 1, E: 2
C -> ∞ | A: 4, D: 3
D -> ∞ | E: 1
```



```
E -> ∞

iteration 1:
mapper:

Read s -> 0 | A: 1, C: 2
Emit: (A, 1), (C, 2)

reducer:
Receives: (A, 1), (C, 2)
Emit:
A: 1 | S->A | B:6
C: 2 | S->C | A:4, D:3

iteration 2:
mapper:
Read: A -> 1 | B: 6
Emit: (B: 7)

Read: C -> 2 | A: 4, D: 3
Emit: (A: 6), (D: 5)
reducer:
B: 7 | S->A->B | D:1, E:2
D: 5 | S->C->D | E:1

iteration 3:
mapper:
Read: B -> 7 | D: 1 Emit (D: 8)
Read: B -> 7 | E: 2 Emit (E: 9)
Read: D -> 5 | E: 1 Emit (E, 6)
reducer:
E: 6 | S->C->D->E | empty

Algorithm terminates
```

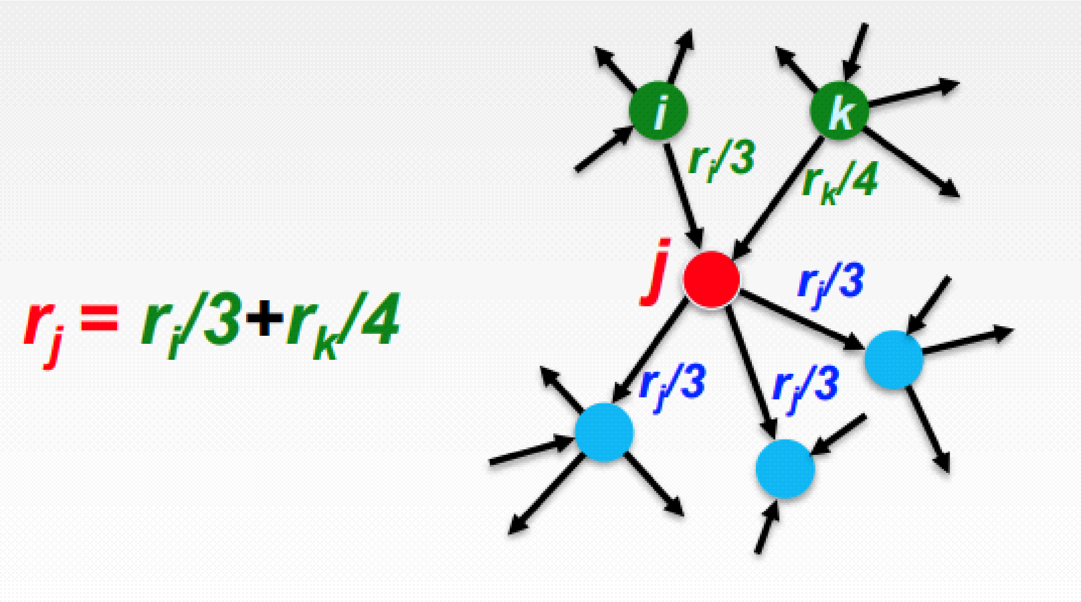
8.2 Page Rank

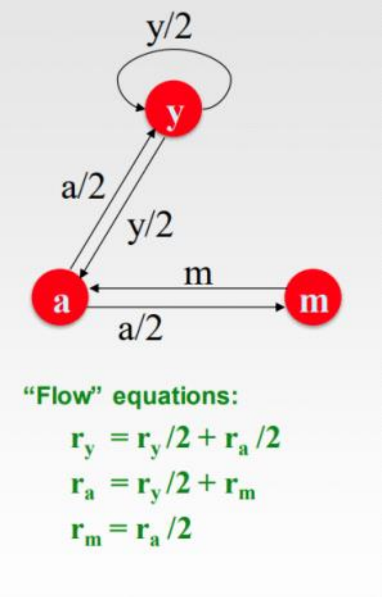
Friday, November 10, 2023 10:13 PM

In links: the links that come into the vertex

Simple Recursive Formulation

each link vote is proportional to the importance of its source page. In the example below, j has two in-links from two green vertices.





"Flow" equations:

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

The pagerank algo follows the flow model, the flow equation can be used to solve the importance of verticies

Gaussian elimination method works for small examples, we need to add another equation, assume the sum of all importance is 1. This only works on small examples

$$r_y = \frac{2}{5}, r_a = \frac{2}{5}, r_m = \frac{1}{5}$$

Matrix Formulation

stochastic adjacency matrix M

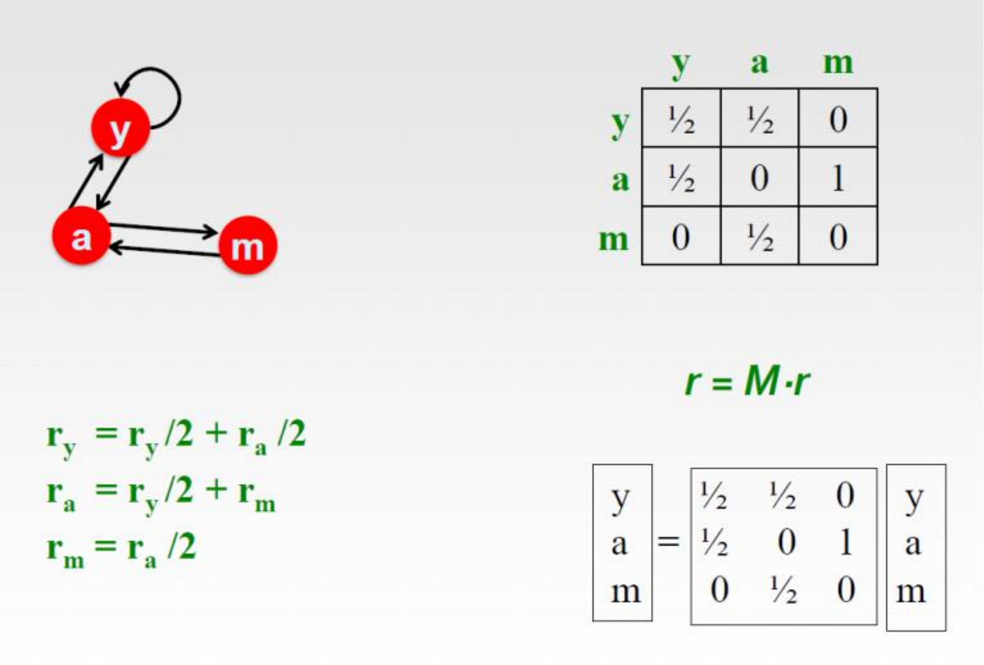
- let page i has d_i out-links
- if i -> j then $M_{ji} = 1/d_i$ else $M_{ji} = 0$. This means a column has a sum of 1

Rank vector r: vector with an entry per page

- r_i is the importance score of page i
- the sum of $r_i = 1$ (similar to the flow equation)

Eigenvector Formulation

The rank vector r is an eigenvector of the stochastic web matrix M.



$$\begin{array}{c|ccc} & y & a & m \\ \hline y & \frac{1}{2} & \frac{1}{2} & 0 \\ a & \frac{1}{2} & 0 & 1 \\ m & 0 & \frac{1}{2} & 0 \end{array}$$

$$r = M \cdot r$$

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 1 \\ 0 & \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

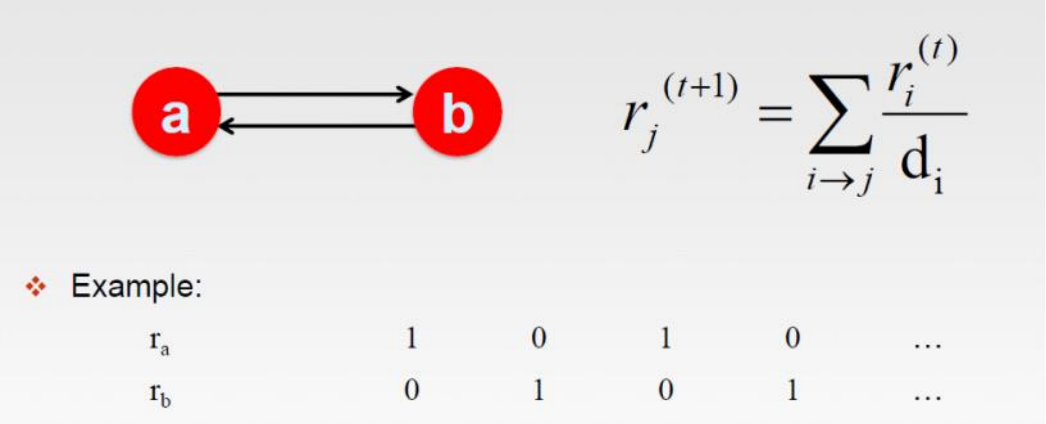
Power Iteration

See spreadsheet

Converge?

spider trap problem (cyclic graph, all out-links are within the group, random walker gets stuck) and

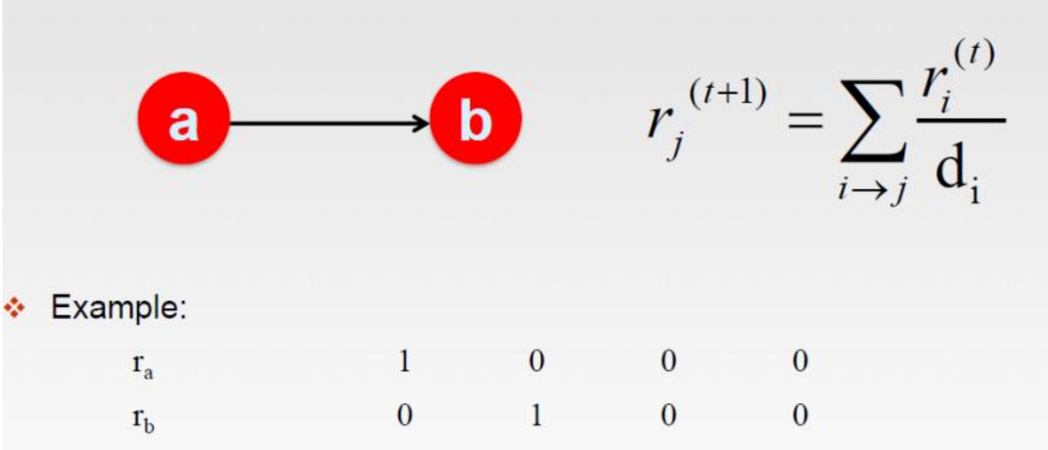
dead end problem (a page doesn't have outlinks)



$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d}_\mathrm{i}}$$

- Example:

r_a	1	0	1	0	…
r_b	0	1	0	1	…

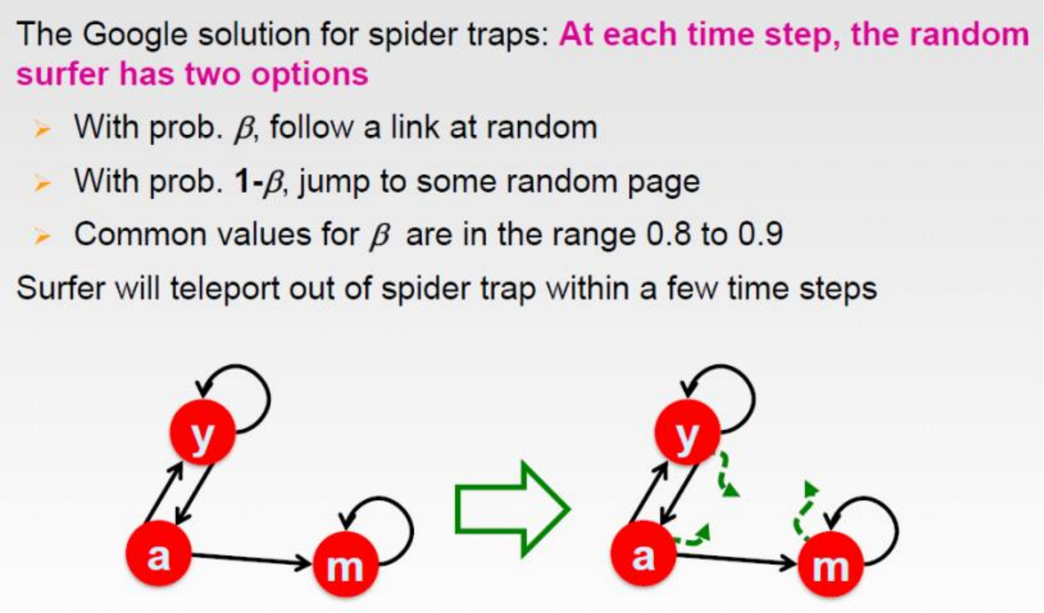


$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d}_\mathrm{i}}$$

- Example:

r_a	1	0	0	0
r_b	0	1	0	0

Spider Trap Solution: Teleport

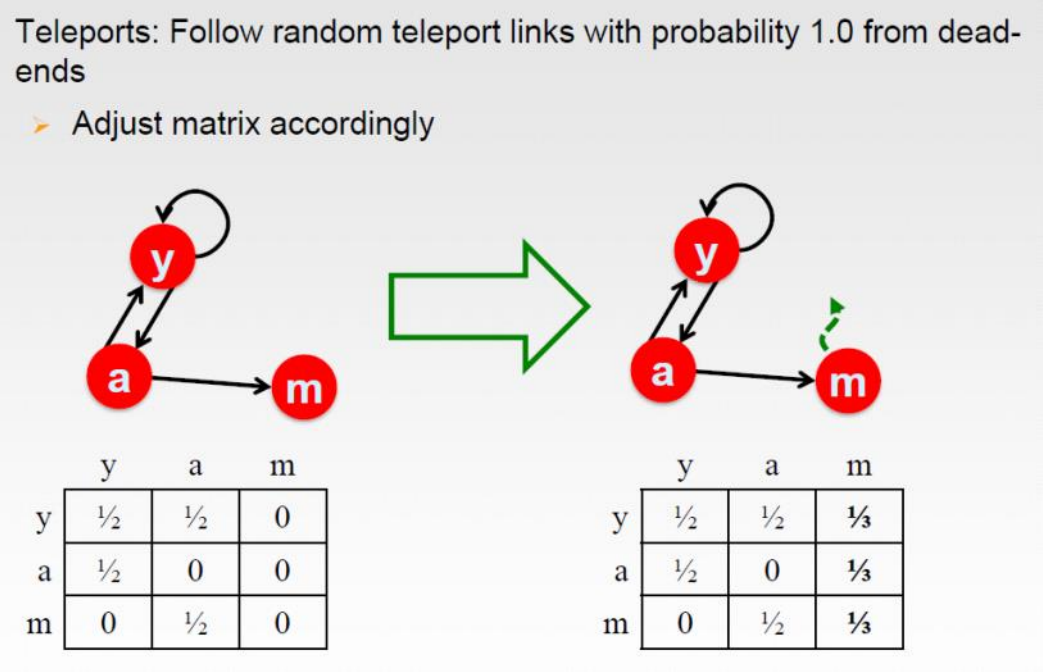


The Google solution for spider traps: **At each time step, the random surfer has two options**

- With prob. β, follow a link at random
- With prob. **1-**β, jump to some random page
- Common values for β are in the range 0.8 to 0.9

Surfer will teleport out of spider trap within a few time steps

Dead End Solution: Always Teleport



Teleports: Follow random teleport links with probability 1.0 from dead-ends

- Adjust matrix accordingly

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

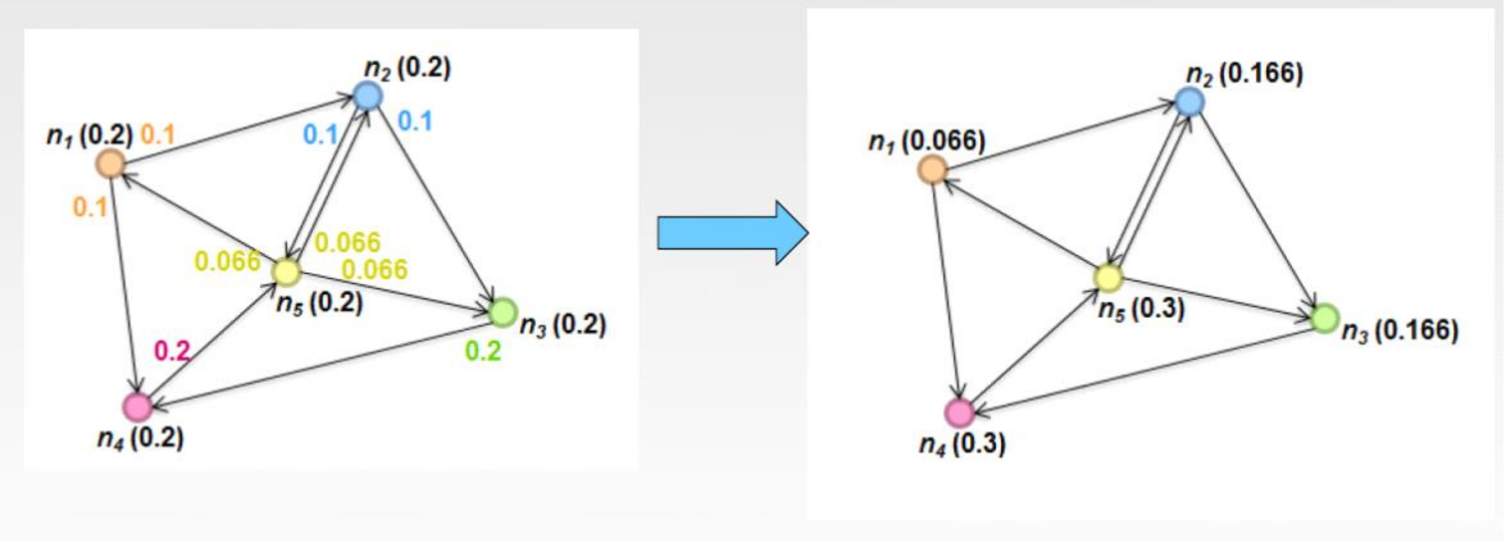
	y	a	m
y	½	½	**⅓**
a	½	0	**⅓**
m	0	½	**⅓**

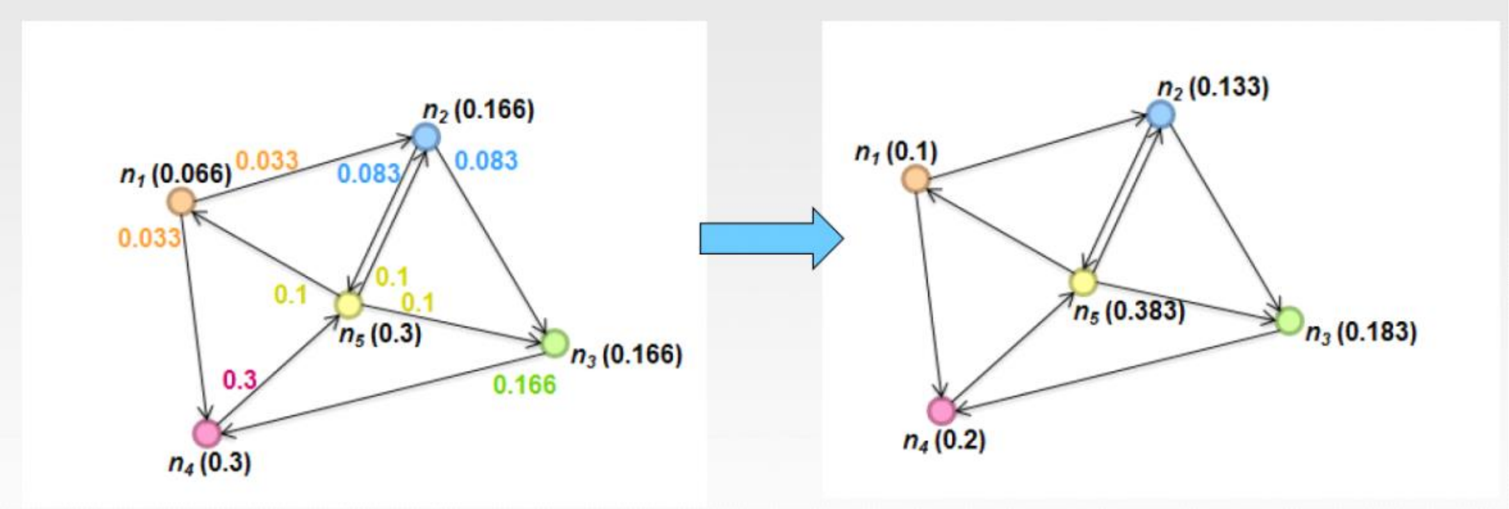
see excel for calculation

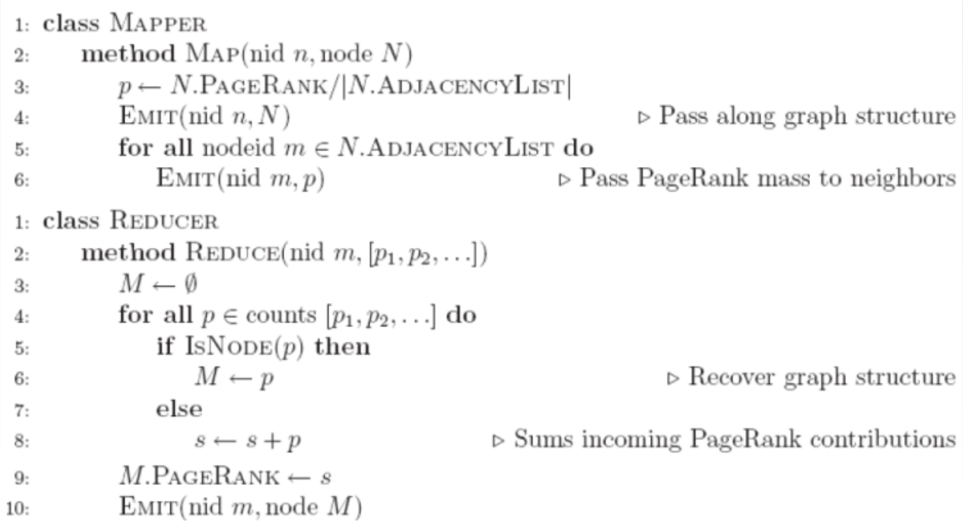
8.3 Pagerank in Mapreduce

Friday, December 1, 2023 12:25 PM

Sample Pagerank







```
class Mapper
    method Map(nid $n$, node $N$)
        $p \leftarrow N.\text{PageRank}/|N.\text{AdjacencyList}|$
        Emit(nid $n, N$)                              ▷ Pass along graph structure
        for all nodeid $m \in N.\text{AdjacencyList}$ do
            Emit(nid $m, p$)                          ▷ Pass PageRank mass to neighbors

class Reducer
    method Reduce(nid $m, [p_1, p_2, \ldots]$)
        $M \leftarrow \emptyset$
        for all $p \in$ counts $[p_1, p_2, \ldots]$ do
            if IsNode($p$) then
                $M \leftarrow p$                      ▷ Recover graph structure
            else
                $s \leftarrow s + p$                  ▷ Sums incoming PageRank contributions
        $M.\text{PageRank} \leftarrow s$
        Emit(nid $m$, node $M$)
```

```
mapper1:
    emit (node_id, N) # similar to shortest path, we need this list to do the iterations
    p = Np / len(adjacency_list)
    for each node in N.adjacency_list:
        emit(node_id, p)
reducer1:
    M = ∅
    for all p in counts [p1,p2…] do
    if is_node(p) then:
        Matrix = p # recover the graph structure
    else:
        s = s+p # sums incoming page rank contributions
    M.pagerank = s
    emit(node_id, Matrix)
```

Q1: HDFS, MapReduce, and Spark concepts

Monday, November 20, 2023 1:12 PM

In what kind of problems a combiner class and a reducer class can be used interchangeably? Please use an example to explain your answer.

Combiners can be used interchangeably with Reducers when the operation being performed is both commutative and associative. A combiner operates on each map output key, it must have the same output key-value types as the Mapper class. In mathematical terms, if the operation satisfies the commutative and associative properties, it can be used interchangeably between the Combiner and Reducer.

One example is the word count example, our task is to count the frequency of each word. The Combiner is used to perform a local aggregation on the output of the mappers before the data is transferred over the network to the reducers. The purpose of the Combiner is to reduce the volume of data that needs to be transferred, leading to a more efficient MapReduce job. The combiners take advantages of all opportunities for local aggregation.

One counter example is to count the average length of words started with 1 letter, the average calculation is not communitive or assotiative, therefore, we can't use combiner and reducer interchangeably in this case.

Why do we want to define our own partitioner?

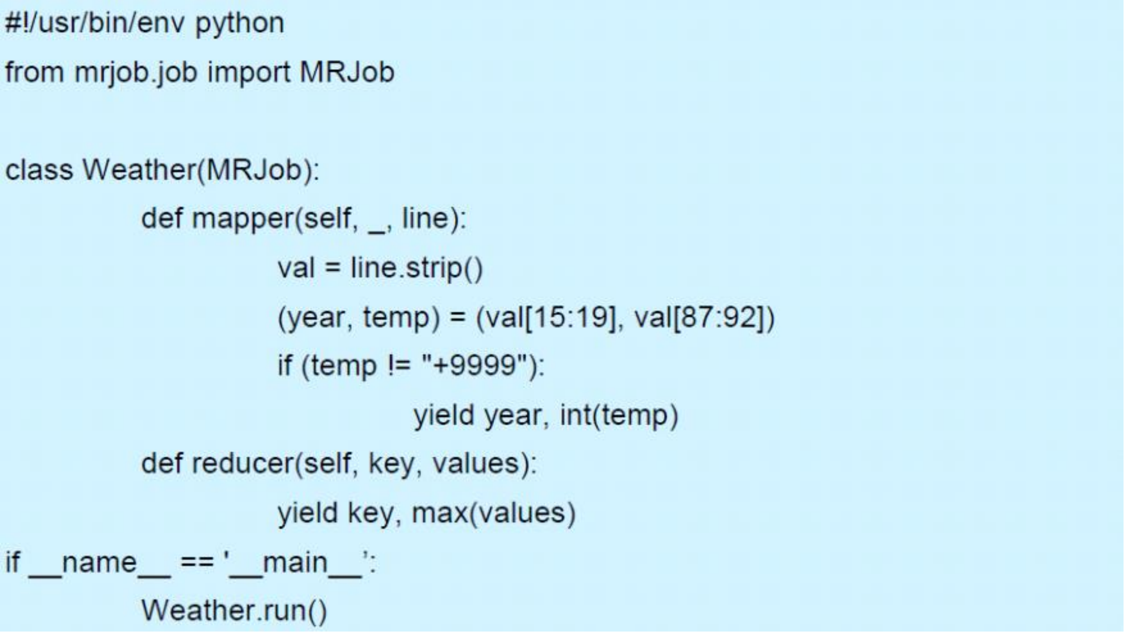
The partitioner in mapreduce controls the partitioning of the keys of the intermediate mapper output, often we have multiple machines and a partitioner is responsible for determining how the data is distributed across different partitions in a distributed computing environment.

Find the maximum temperature in the dataset

Mapper: extract (year, tempature), handle bad readings, missing values etc

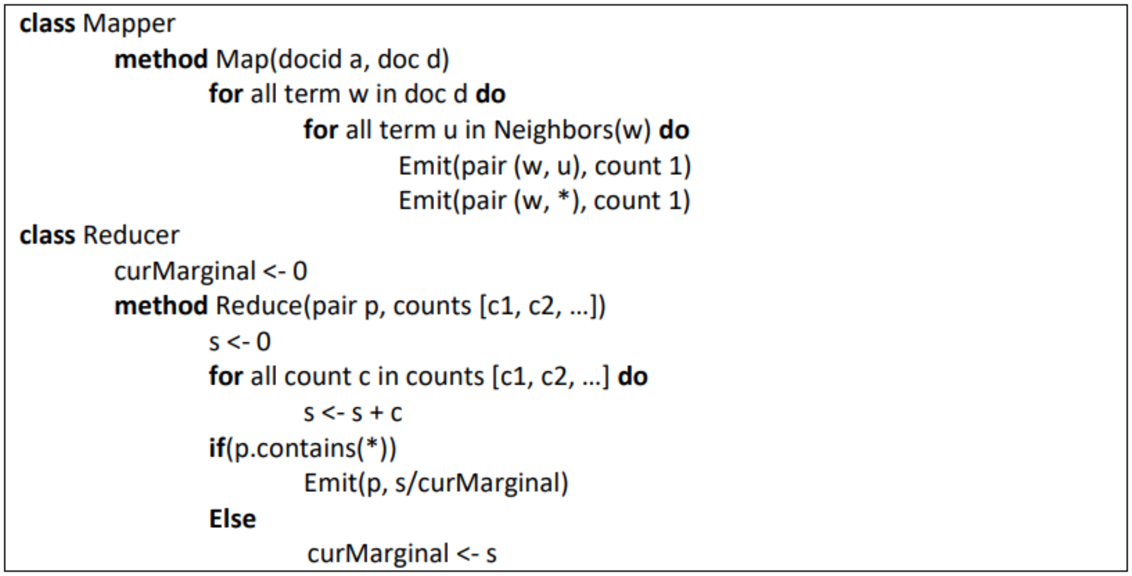
Combiner: compute local maximum temperature, it is necessary in this question

Reducer: performs aggregation of temperature on the same key and find the maximum value



```
#!/usr/bin/env python
from mrjob.job import MRJob

class Weather(MRJob):
        def mapper(self, _, line):
                val = line.strip()
                (year, temp) = (val[15:19], val[87:92])
                if (temp != "+9999"):
                        yield year, int(temp)
        def reducer(self, key, values):
                yield key, max(values)
if __name__ == '__main__’:
        Weather.run()
```

Count relevant frequency



```
class Mapper
    method Map(docid a, doc d)
        for all term w in doc d do
            for all term u in Neighbors(w) do
                Emit(pair (w, u), count 1)
                Emit(pair (w, *), count 1)
class Reducer
    curMarginal <- 0
    method Reduce(pair p, counts [c1, c2, …])
        s <- 0
        for all count c in counts [c1, c2, …] do
            s <- s + c
        if(p.contains(*))
            Emit(p, s/curMarginal)
        Else
            curMarginal <- s
```

In line Emit(pair(w,u), 1) and Emit(pair(w,), 1), it is not guaranteed that they will be sent to the same partition, therefore, a partitioner is required to guarantee that the key-value pairs relevant to the same term w are sent to the same reducer. Another issue is that p.contain() should be not p.contains(*)

In one project, a student complained that her approach took a lot of time at the step when using the reduce() function, but all the previous operations including reading the data by textFile(), filtering the data by filter(), and transform the data by map() and flatmap(). Could you please explain the reason to her?

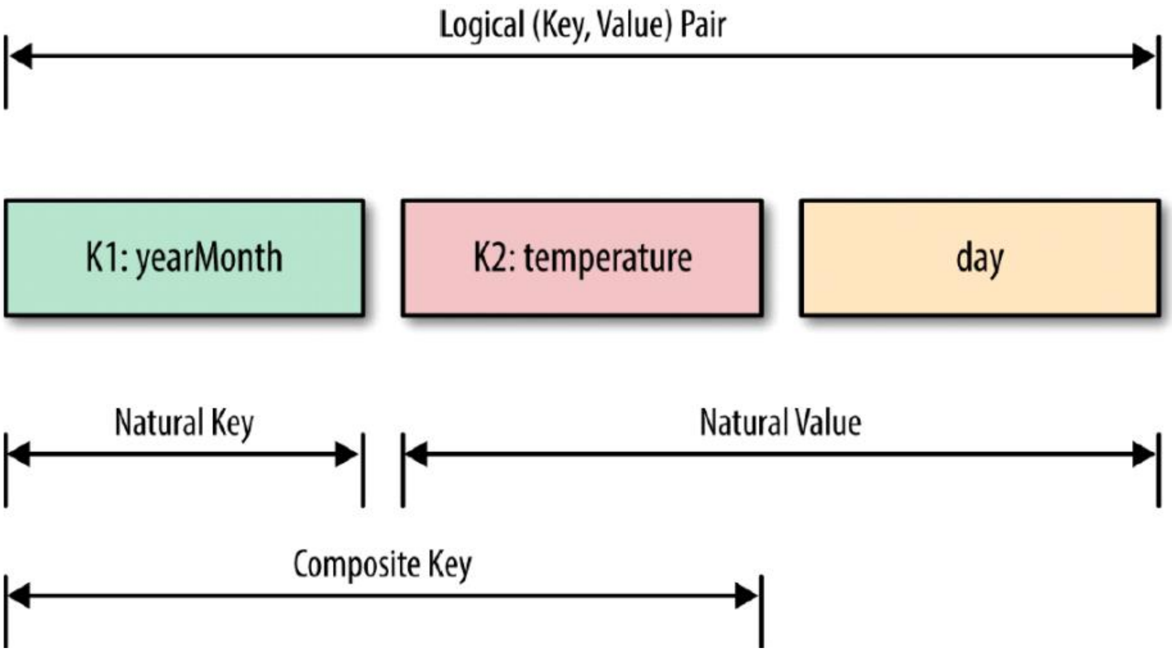
This is because o f the lazy evaluation used by spark. All transformations in Spark are lazy, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset (e.g. a file). The transformations are only computed when an action requires a result to be returned to the driver program. In this case, creating the df by textFile(), the actual reading of the file occurs when an action is performed on the RDD. Filtering the dtaa by filter(), and transform the data by map() and flatmap() are all lazy transformations. They are not executed right away until some actions are called, for example reduce().

Please briefly explain why the order inversion and value-to-key design patterns applied in Project 1 can improve the performance on MapReduce.

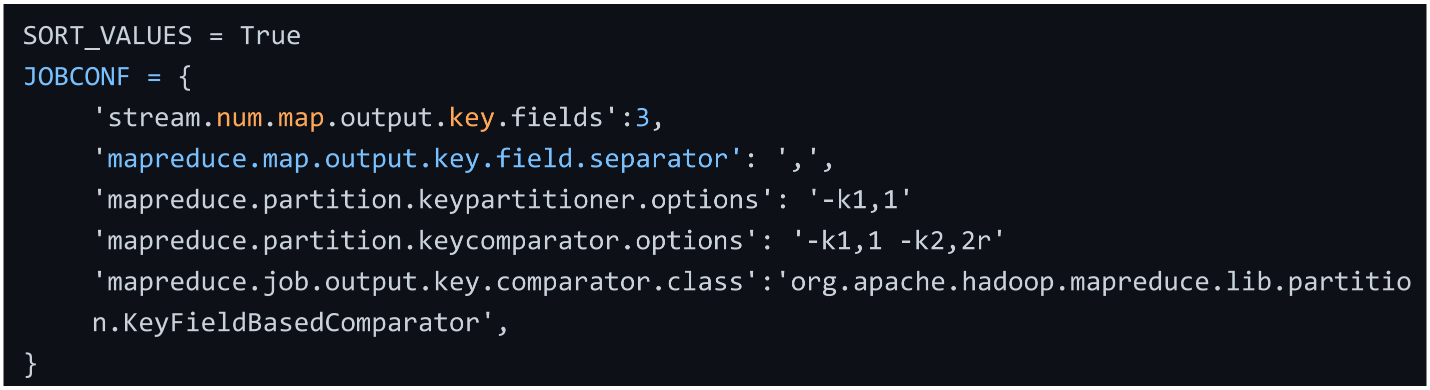
“Order inversion”, where the main idea is to convert the sequencing of computations into a sorting problem. For example, we can additionally emits a special key of the form (wi, *) in the mapper. In addition, you need to guarantee that all key-value pairs relevant to the same term are sent to the same reducer. Sorting key-value pairs allows the system to efficiently group related data together. Grouping related data together minimizes the amount of data that needs to be transferred over the network during the shuffle and sort phase.

“Value-to-key conversion”, which provides a scalable solution for secondary sorting. You can customize the sorting logic based on the values associated with each key. This flexibility is beneficial when the default sorting order may not be suitable for the specific requirements of your computation. Reducers can efficiently process data when the values associated with each key are sorted in the desired order. This can lead to more straightforward and optimized reducer logic.

"Secondary Sort", The "Value-to-Key Conversion" design pattern, often referred to as secondary sort, involves forming a composite key (K, V), where K is the natural key, and V is the secondary key. This pattern is particularly useful for scenarios where you need to sort values associated with a key during the reduce phase of a MapReduce job. For example, we can have something like t, (m, r) as our line, we can convert it to (t,m), r and leverage mapreduce framework to do the sorting for us.

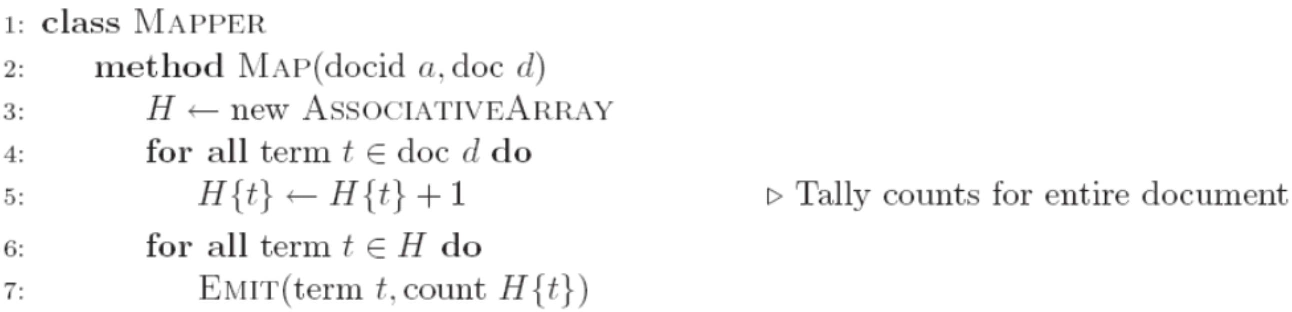


Partitioner, sorting and JOBCONF



```
SORT_VALUES = True
JOBCONF = {
    'stream.num.map.output.key.fields':3,
    'mapreduce.map.output.key.field.separator': ',',
    'mapreduce.partition.keypartitioner.options': '-k1,1'
    'mapreduce.partition.keycomparator.options': '-k1,1 -k2,2r'
    'mapreduce.job.output.key.comparator.class':'org.apache.hadoop.mapreduce.lib.partitio
    n.KeyFieldBasedComparator',
}
```

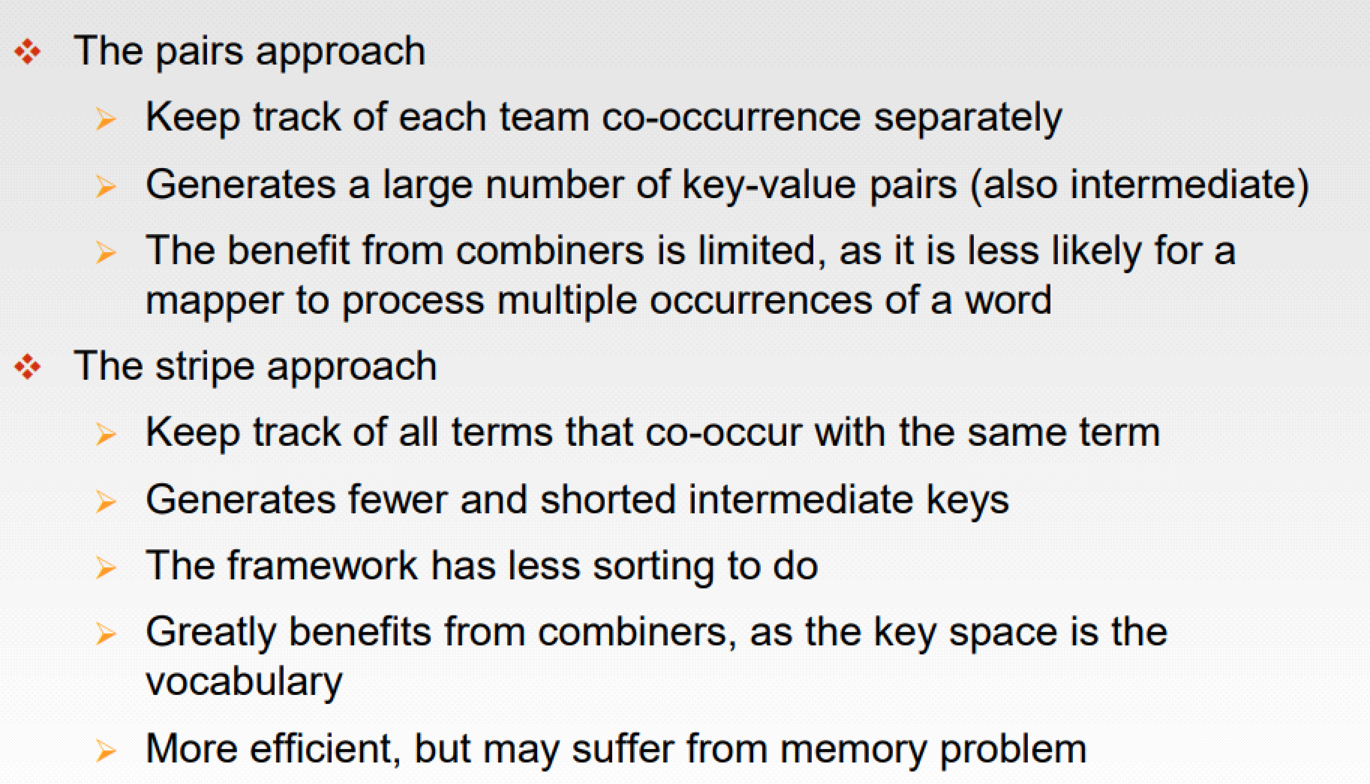
Is combiner needed? What's wrong with this code?



```
class MAPPER
    method MAP(docid $a$, doc $d$)
        $H \leftarrow$ new ASSOCIATIVEARRAY
        for all term $t \in$ doc $d$ do
            $H\{t\} \leftarrow H\{t\} + 1$                ▷ Tally counts for entire document
        for all term $t \in H$ do
            EMIT(term $t$, count $H\{t\}$)
```

The combiner is still needed here. Because one map function is only dealing with one line, instead of the entire data block. Here, we will still have the same number of records emitted from the mapper. We can improve this code by creating an associative array for the entire data block, for example, using mapper_init() function, and use the mapper_final() function to do the local aggregation. The issue of this code is that scalability issue (not suitable for huge data). More memory is required for a mapper to store intermediate results.

Pairs approach vs Stripes approach

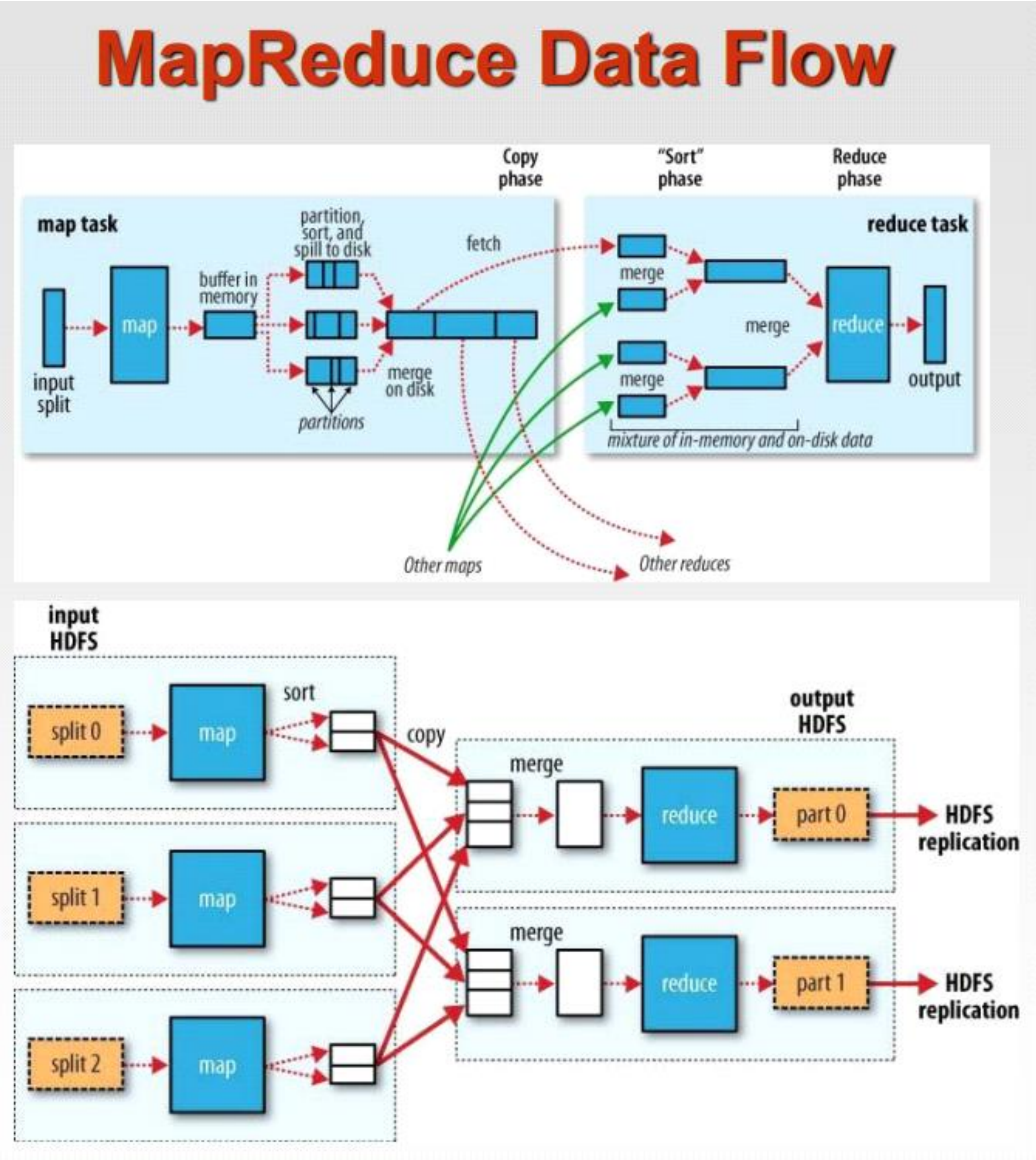


- The pairs approach
  - Keep track of each team co-occurrence separately
  - Generates a large number of key-value pairs (also intermediate)
  - The benefit from combiners is limited, as it is less likely for a mapper to process multiple occurrences of a word
- The stripe approach
  - Keep track of all terms that co-occur with the same term
  - Generates fewer and shorted intermediate keys
  - The framework has less sorting to do
  - Greatly benefits from combiners, as the key space is the vocabulary
  - More efficient, but may suffer from memory problem

Given a large text dataset, find the top-k frequent terms (considering that you can utilize multiple reducers, and the efficiency of your method is evaluated).

Two rounds: first round compute term frequency in multiple reducers, and each reducer only stores local top-k. Second round get the local top-k, and compute the final top-k using a single reducer.

Explain the data flow in MapReduce using the word count problem as an example



MapReduce Data Flow

HDFS File System: The input data is stored in the Hadoop Distributed File System (HDFS). Let's assume we have a set of text documents. When we have multiple reducers, the map tasks partition their output so that there is one partition for each reduce task, the records for every key are all in a single partition, and partitioning can be controlled by a user-defined partitioning function. The input data in HDFS is divided into manageable chunks called input splits. Each split is processed by a separate mapper.

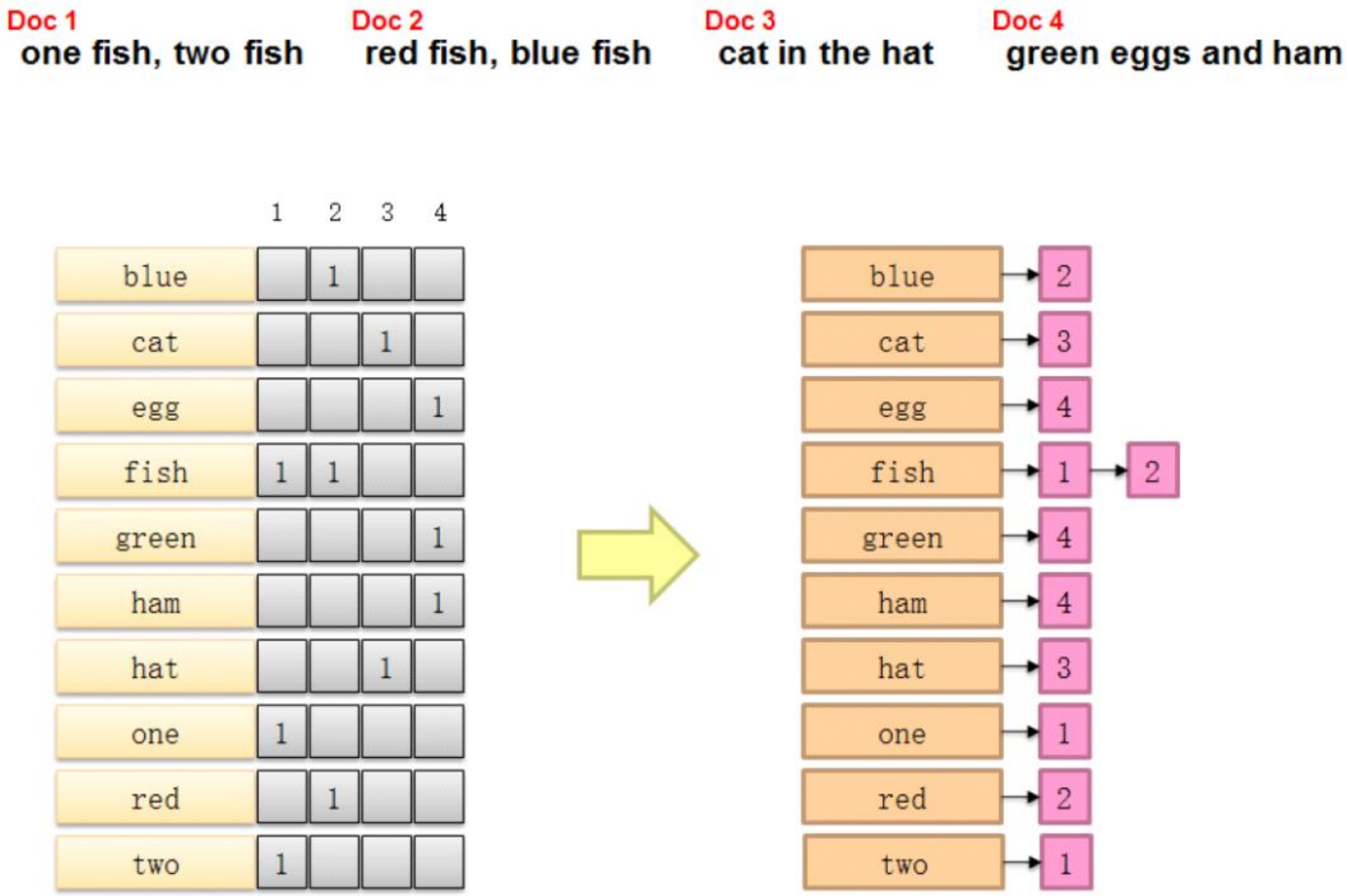
Mapper: The Mapper function is applied to each input split independently. In the Word Count example, the Mapper function processes each document and emits key-value pairs where the key is a word, and the value is 1. The output of the Mapper is a set of intermediate key-value pairs where the key is a word, and the value is 1 for each occurrence of that word in the document. The mapper buffers all the data in memory unless there is a spill. When spill happends, data will be saved in HDFS file system temporary. Each partitioner sorts the data in the partitioner and merge the data in disk. Before the data is sent to the reducer, we have shuffle and sort phase where the intermediate key-value pairs from all the mappers are shuffled and sorted based on the key. This ensures that all occurrences of a particular word are grouped together.

Combiner: The Combiner is an optional local aggregation step that runs on each mapper's output before data is sent to the reducers. In the Word Count example, the Combiner can locally aggregate word counts.

Reducer: reducers will wait for all the mappers to finish because they need to communicate with the mappers. The data are sent from mappers to reducers via some protocols, then the reducers perform merge sort and use reduce function to aggregate data based on keys, producing the final output

Explain the data flow in Spark using the word count problem as an example

Inverted Index



Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

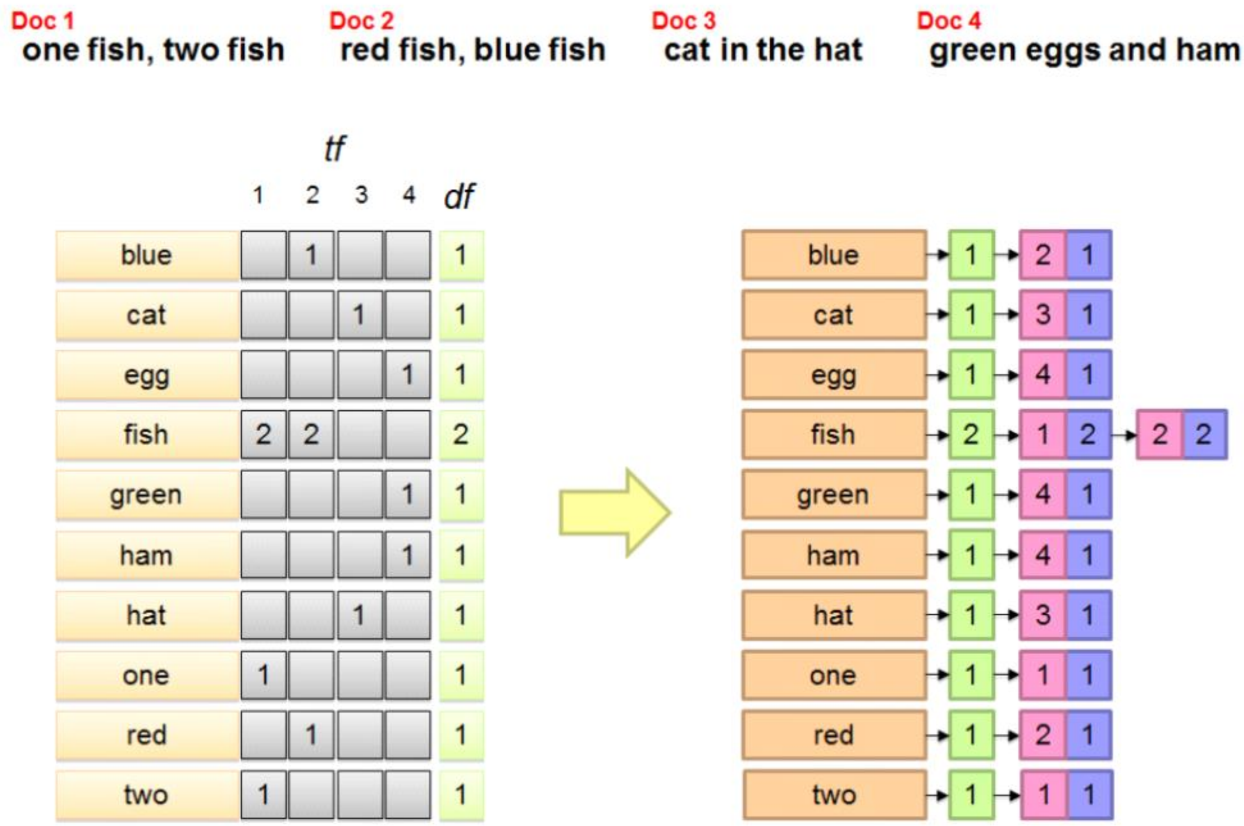
Doc 4
green eggs and ham

	1	2	3	4
blue		1		
cat			1	
egg				1
fish	1	1		
green				1
ham				1
hat			1	
one	1			
red		1		
two	1			

➢ Step 1 (vocabulary search): find each term/word in q in the inverted index.

➢ Step 2 (results merging): Merge results to find documents that contain all or some of the words/terms in q.

➢ Step 3 (Rank score computation):



Doc 1 one fish, two fish

Doc 2 red fish, blue fish

Doc 3 cat in the hat

Doc 4 green eggs and ham

	tf				
	1	2	3	4	df
blue		1			1
cat			1		1
egg				1	1
fish	2	2			2
green				1	1
ham				1	1
hat			1		1
one	1				1
red		1			1
two	1				1

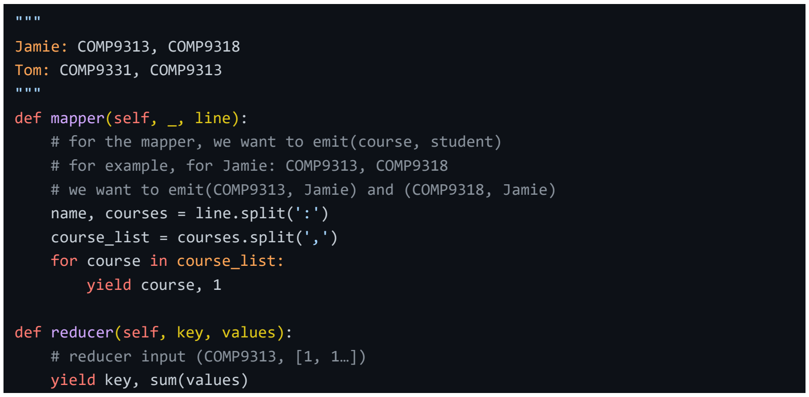
To computer term frequency and Inverse document frequency

Q2: MapReduce Algorithm Design (pseudo-code only)

Monday, November 20, 2023 1:12 PM

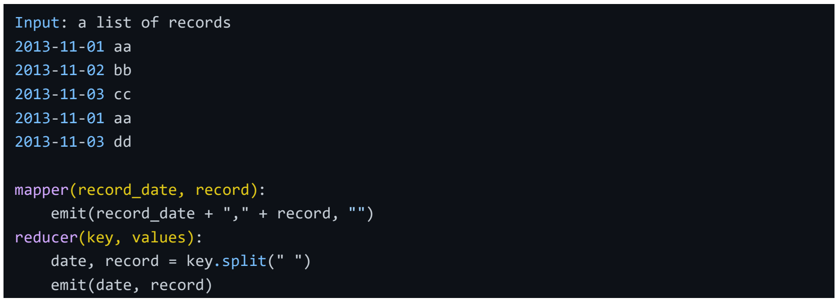
Counting total enrollments of two specified courses

Input Files: A list of students with their enrolled courses



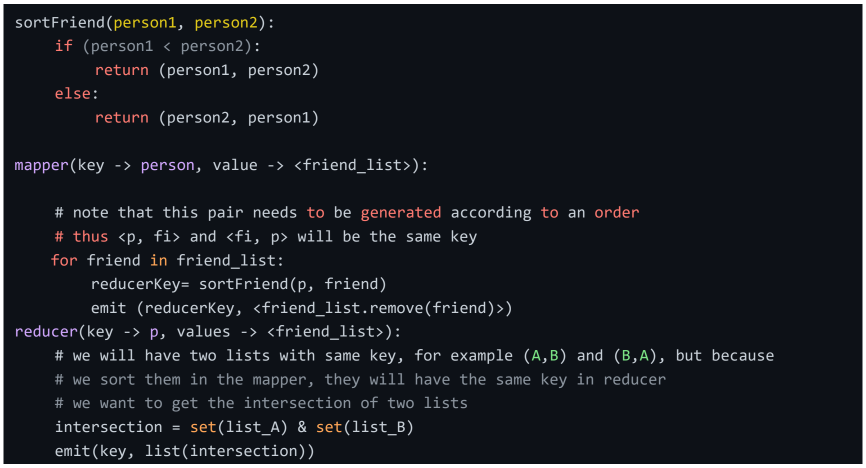
```
"""
Jamie: COMP9313, COMP9318
Tom: COMP9331, COMP9313
"""
def mapper(self, _, line):
    # for the mapper, we want to emit(course, student)
    # for example, for Jamie: COMP9313, COMP9318
    # we want to emit(COMP9313, Jamie) and (COMP9318, Jamie)
    name, courses = line.split(':')
    course_list = courses.split(',')
    for course in course_list:
        yield course, 1

def reducer(self, key, values):
    # reducer input (COMP9313, [1, 1…])
    yield key, sum(values)
```

Remove duplicate records



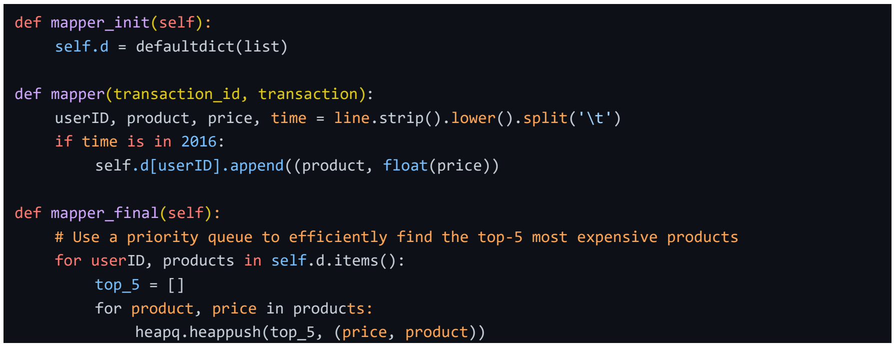
```
Input: a list of records
2013-11-01 aa
2013-11-02 bb
2013-11-03 cc
2013-11-01 aa
2013-11-03 dd

mapper(record_date, record):
    emit(record_date + "," + record, "")
reducer(key, values):
    date, record = key.split(" ")
    emit(date, record)
```

Calculate the common friends for each pair of users in Facebook. Assume the friends are stored in format of Person -> [List of friends]. e.g.: A -> [B C D], B -> [A C D E], C -> [A B D E], D -> [A B C E], E -> [B C D]. Note that the "friendship" is bi-directional, which means that if A is in B's list, B would be in A's list as well. Your result should be like:



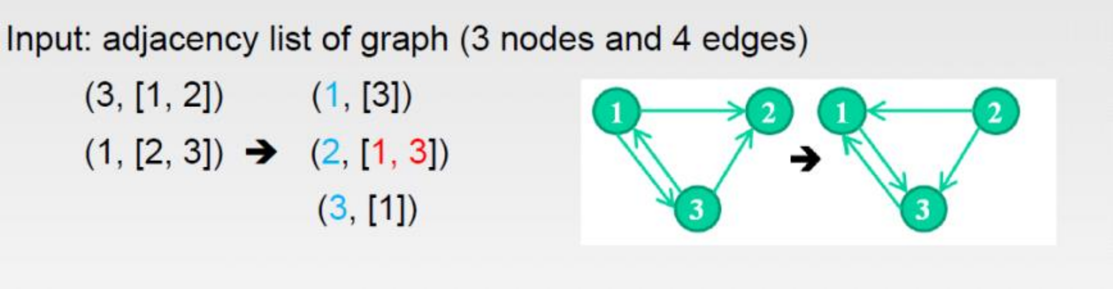
```
sortFriend(person1, person2):
    if (person1 < person2):
        return (person1, person2)
    else:
        return (person2, person1)

mapper(key -> person, value -> <friend_list>):

    # note that this pair needs to be generated according to an order
    # thus <p, fi> and <fi, p> will be the same key
    for friend in friend_list:
        reducerKey= sortFriend(p, friend)
        emit (reducerKey, <friend_list.remove(friend)>)
reducer(key -> p, values -> <friend_list>):
    # we will have two lists with same key, for example (A,B) and (B,A), but because
    # we sort them in the mapper, they will have the same key in reducer
    # we want to get the intersection of two lists
    intersection = set(list_A) & set(list_B)
    emit(key, list(intersection))
```

Assume that in an online shopping system, a huge log file stores the information of each transaction. Each line of the log is in format of "userID\tproduct\t price\t time". Your task is to use MapReduce to find out the top-5 most expensive products purchased by each user in 2016.



```
def mapper_init(self):
    self.d = defaultdict(list)

def mapper(transaction_id, transaction):
    userID, product, price, time = line.strip().lower().split('\t')
    if time is in 2016:
        self.d[userID].append((product, float(price))

def mapper_final(self):
    # Use a priority queue to efficiently find the top-5 most expensive products
    for userID, products in self.d.items():
        top_5 = []
        for product, price in products:
            heapq.heappush(top_5, (price, product))
            if len(top_5) > 5:
                heapq.heappop(top5)
        emit(userID, top_5)

def reducer(self, key->userID, values->list of queues[]):
    # The reducer can remain unchanged based on your original logic
    # It will receive the top-5 lists from different mappers
    # and can merge them to find the global top-5 for each user
    emit(key, merged_top_5)
```

Reverse graph edge directions & output in node order



Input: adjacency list of graph (3 nodes and 4 edges)

(3, [1, 2]) (1, [2, 3]) ➔ (1, [3]) (2, [1, 3]) (3, [1])

(1,3), (2,3), (2,1), (3,1)

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class ReverseEdgeDirection(MRJob):
    def mapper(self, _, value):
        fromNode, neighbor = value.split("\t", 1)
        nodes = neighbor.strip().split(" ")
        for node in nodes:
            yield "a," + node + "," + fromNode, ""

    def reducer_init(self):
        self.neighbor=[]
        self.curNode=""

    def reducer(self, key, values):
        a, node, fromNode = key.split(",")
        if(node == self.curNode):
            self.neighbor.append(fromNode)
        else:
            if self.curNode!="":
                yield self.curNode, " ".join(self.neighbor)
            self.neighbor=[]
            self.neighbor.append(fromNode)
            self.curNode=node

    def reducer_final(self):
        # Do not forget the last node
        yield self.curNode, " ".join(self.neighbor)

    SORT_VALUES = True

    def steps(self):

        JOBCONF = {
          'stream.num.map.output.key.fields':2,
          'mapreduce.map.output.key.field.separator': ',',
          'mapreduce.job.reduces':1,
          'mapreduce.partition.keypartitioner.options':'-k1,2',
          'mapreduce.job.output.key.comparator.class':'org.apache.hadoop.mapreduce.lib.partition.KeyFieldBasedComparator',
          'mapreduce.partition.keycomparator.options':'-k2,2n -k3,3n'
        }
        return [MRStep(jobconf=JOBCONF, mapper=self.mapper, reducer=self.reducer, reducer_init= self.reducer_init, reducer_final=self.reducer_final)]

if __name__ == '__main__':
    ReverseEdgeDirection.run()
```

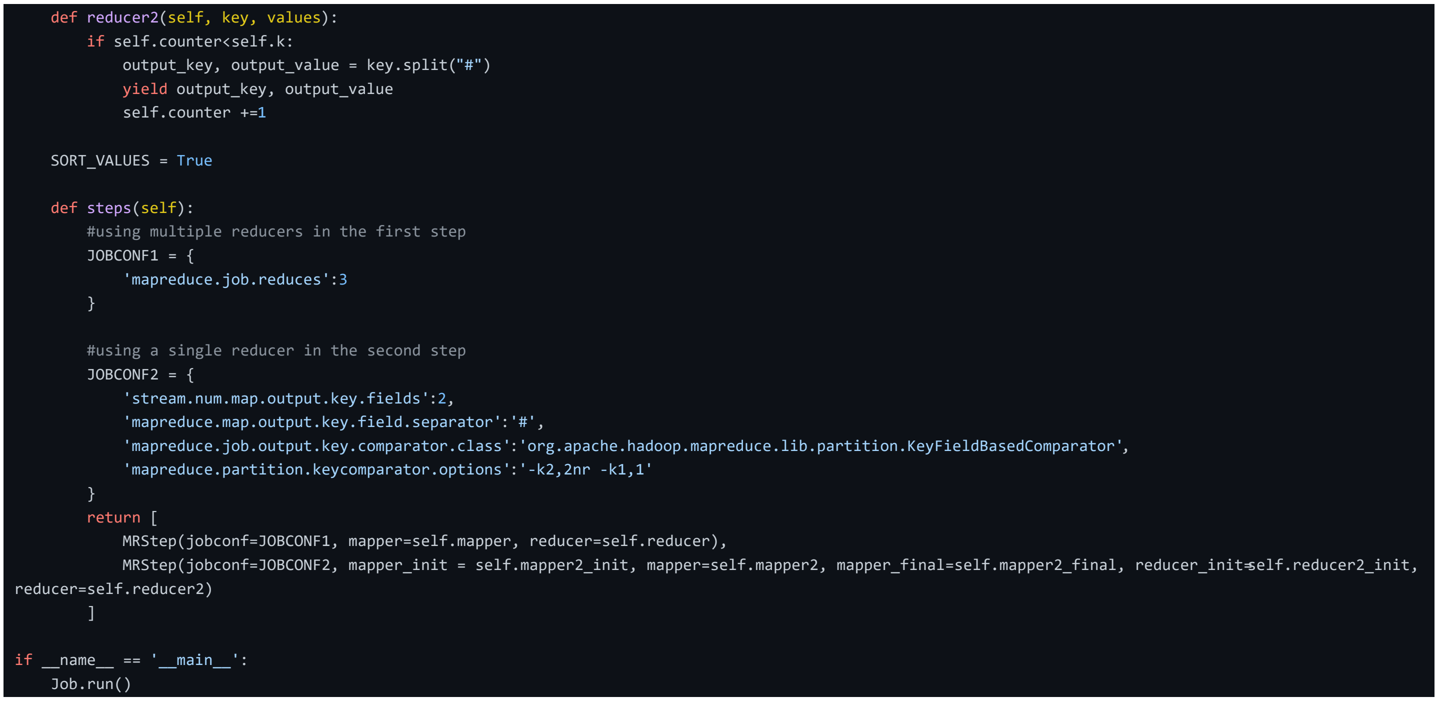
Assume that you are given a data set crawled from a location-based social network, in which each line of the data is in format of (userID, a list of locations the user has visited <loc1, loc2…>). Your task is to compute for each location the set of users who have visited it, and the users are sorted in ascending order according to their IDs.

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class LocationUser(MRJob):
    def mapper(self, _, line):
        userID, loc_string = line.split(':')
        locations = loc_string.split(',')
        for loc in locations:
            yield "a," + loc.strip() + "," + userID.strip(), ""

    def reducer_init(self):
        self.d = {}
    def reducer(self, key, value):
        k, v = key.split(",", 1)[1].split(",")
        self.d[k.strip()] = self.d.get(k.strip(), []) + list(v.strip())
    def reducer_final(self):
        for k, v in self.d.items():
            yield k, v
    SORT_VALUES = True

    JOBCONF = {
            'stream.num.map.output.key.fields': 2,
            'mapreduce.map.output.key.field.separator': ',',
            'mapreduce.partition.keypartitioner.options':'-k1,2',
            'mapreduce.job.output.key.comparator.class':'org.apache.hadoop.mapreduce.lib.partition.KeyFieldBasedComparator',
            'mapreduce.partition.keycomparator.options':'-k2,2n -k3,3n'
        }

if __name__ == '__main__':
    LocationUser.run()
```

Given a large text dataset, find the top-k frequent terms (considering that you can utilize multiple reducers, and the efficiency of your method is evaluated).

Two rounds: first round compute term frequency in multiple reducers, and each reducer only stores local top-k. Second round get the local top-k, and compute the final top-k using a single reducer.

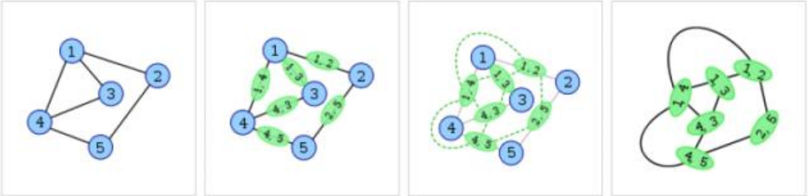
```
from mrjob.job import MRJob
from mrjob.step import MRStep
from mrjob.compat import jobconf_from_env
import heapq
import re

class Pair:
    def __init__(self, term, count):
        self.term = term
        self.count = count
    def __lt__(self, other):
        if self.count != other.count:
            return self.count < other.count
        else:
            return self.term < other.term

class Job(MRJob):

    def mapper(self, key, value):
        value = value.strip()
        words = re.split("[ *$&#/\t\n\f\"\'\\,.:;?!\[\](){}<>~\-_]", value.lower())
        for word in words:
            if len(word):
                yield word, 1

    def reducer(self, key, values):
        count = sum(values)
        yield key, count

    def mapper2_init(self):
        self.k = int(jobconf_from_env('myjob.settings.topk'))
        self.topk = []
        #use a heap to get the local top k from each mapper
        heapq.heapify(self.topk)

    def mapper2(self, key, value):
        p = Pair(key, value)

        heapq.heappush(self.topk,p)
        if len(self.topk)>self.k:
            heapq.heappop(self.topk)

    def mapper2_final(self):
        for i in range(0,self.k):
            yield self.topk[i].term + "#" + str(self.topk[i].count), None

    def reducer2_init(self):
        self.k = int(jobconf_from_env('myjob.settings.topk'))
        self.counter = 0
```



```
    def reducer2(self, key, values):
        if self.counter<self.k:
            output_key, output_value = key.split("#")
            yield output_key, output_value
            self.counter +=1

    SORT_VALUES = True

    def steps(self):
        #using multiple reducers in the first step
        JOBCONF1 = {
            'mapreduce.job.reduces':3
        }

        #using a single reducer in the second step
        JOBCONF2 = {
            'stream.num.map.output.key.fields':2,
            'mapreduce.map.output.key.field.separator':'#',
            'mapreduce.job.output.key.comparator.class':'org.apache.hadoop.mapreduce.lib.partition.KeyFieldBasedComparator',
            'mapreduce.partition.keycomparator.options':'-k2,2nr -k1,1'
        }
        return [
            MRStep(jobconf=JOBCONF1, mapper=self.mapper, reducer=self.reducer),
            MRStep(jobconf=JOBCONF2, mapper_init = self.mapper2_init, mapper=self.mapper2, mapper_final=self.mapper2_final, reducer_init=self.reducer2_init,
reducer=self.reducer2)
        ]

if __name__ == '__main__':
    Job.run()
```

Given a text file, compute the average length of words starting with each letter. This means that for every letter, you need to compute: the total length of all words that start with that letter divided by the total number of words that start with that letter

Exam question

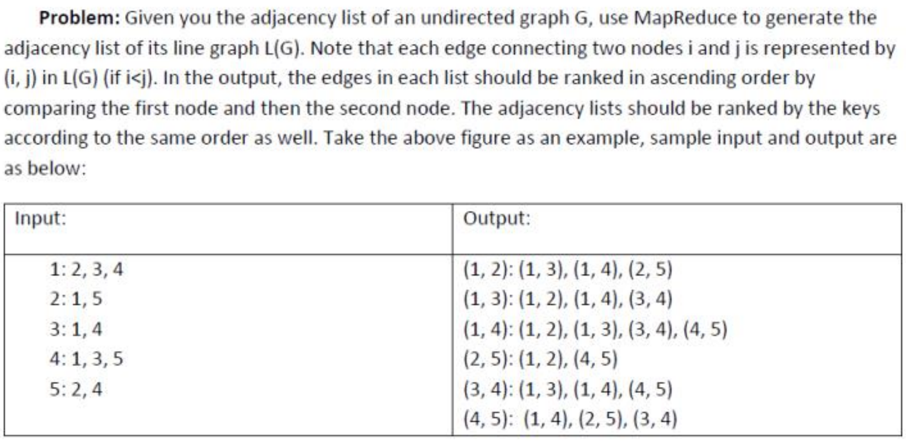


Graph G

Vertices in L(G) constructed from edges in G

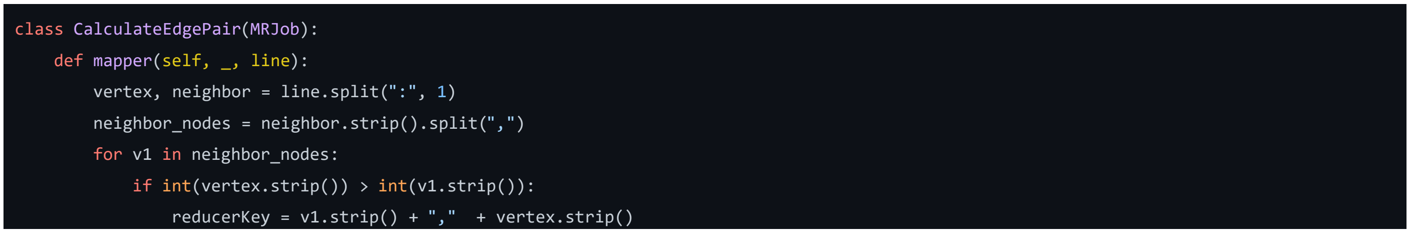
Added edges in L(G)

The line graph L(G)

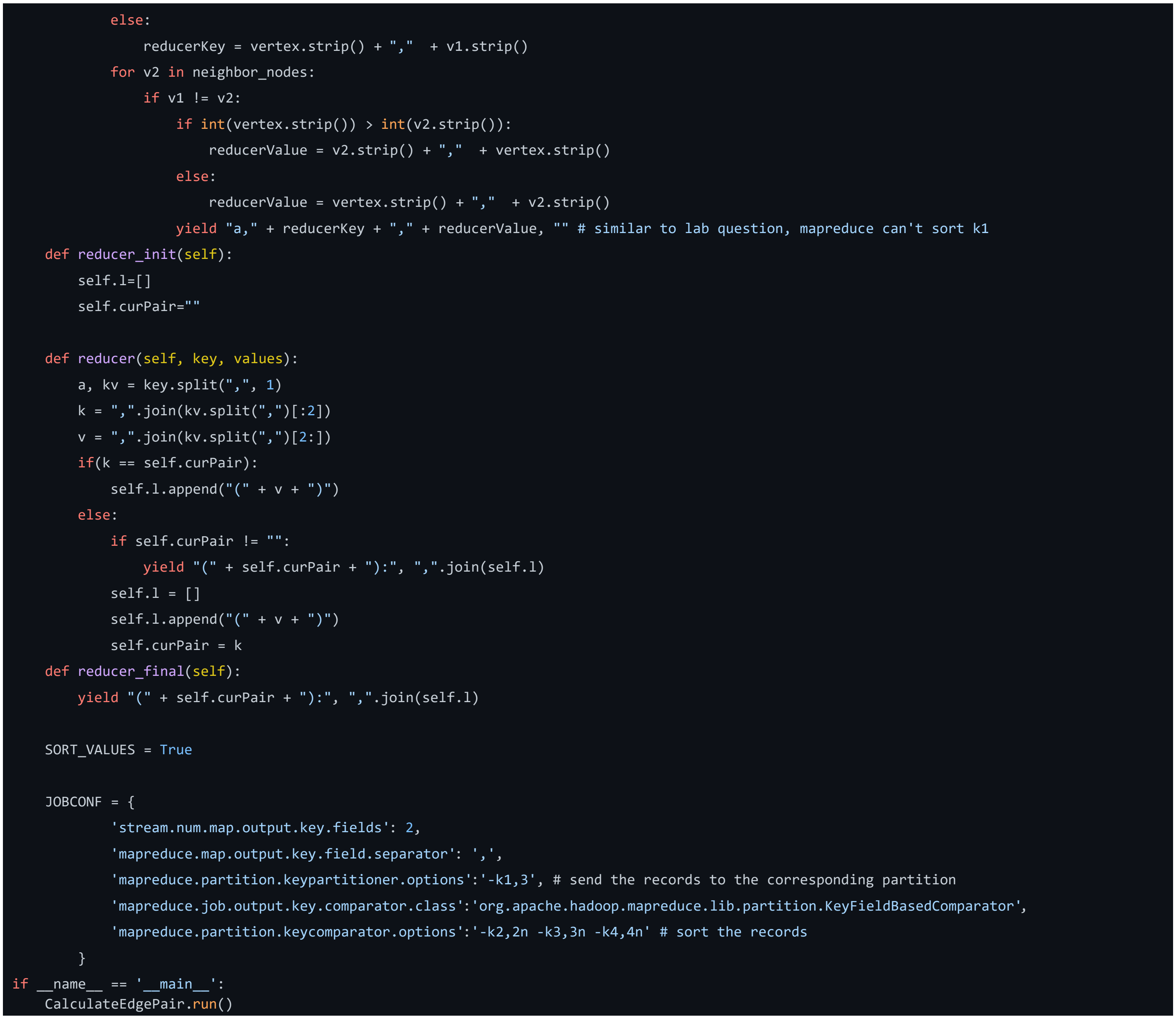


Problem: Given you the adjacency list of an undirected graph G, use MapReduce to generate the adjacency list of its line graph L(G). Note that each edge connecting two nodes i and j is represented by (i, j) in L(G) (if i<j). In the output, the edges in each list should be ranked in ascending order by comparing the first node and then the second node. The adjacency lists should be ranked by the keys according to the same order as well. Take the above figure as an example, sample input and output are as below:

Input:	Output:
1: 2, 3, 4	(1, 2): (1, 3), (1, 4), (2, 5)
2: 1, 5	(1, 3): (1, 2), (1, 4), (3, 4)
3: 1, 4	(1, 4): (1, 2), (1, 3), (3, 4), (4, 5)
4: 1, 3, 5	(2, 5): (1, 2), (4, 5)
5: 2, 4	(3, 4): (1, 3), (1, 4), (4, 5)
	(4, 5): (1, 4), (2, 5), (3, 4)

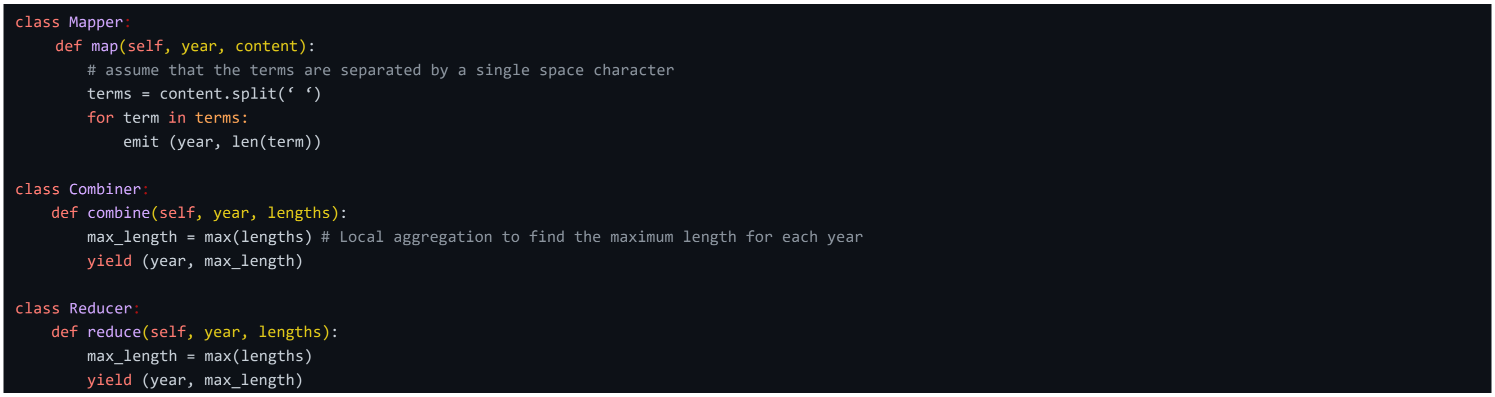


```
class CalculateEdgePair(MRJob):
    def mapper(self, _, line):
        vertex, neighbor = line.split(":", 1)
        neighbor_nodes = neighbor.strip().split(",")
        for v1 in neighbor_nodes:
            if int(vertex.strip()) > int(v1.strip()):
                reducerKey = v1.strip() + ","  + vertex.strip()
```



```
            else:
                reducerKey = vertex.strip() + ","  + v1.strip()
            for v2 in neighbor_nodes:
                if v1 != v2:
                    if int(vertex.strip()) > int(v2.strip()):
                        reducerValue = v2.strip() + ","  + vertex.strip()
                    else:
                        reducerValue = vertex.strip() + ","  + v2.strip()
                    yield "a," + reducerKey + "," + reducerValue, "" # similar to lab question, mapreduce can't sort k1
    def reducer_init(self):
        self.l=[]
        self.curPair=""

    def reducer(self, key, values):
        a, kv = key.split(",", 1)
        k = ",".join(kv.split(",")[:2])
        v = ",".join(kv.split(",")[2:])
        if(k == self.curPair):
            self.l.append("(" + v + ")")
        else:
            if self.curPair != "":
                yield "(" + self.curPair + "):", ",".join(self.l)
            self.l = []
            self.l.append("(" + v + ")")
            self.curPair = k
    def reducer_final(self):
        yield "(" + self.curPair + "):", ",".join(self.l)

    SORT_VALUES = True

    JOBCONF = {
            'stream.num.map.output.key.fields': 2,
            'mapreduce.map.output.key.field.separator': ',',
            'mapreduce.partition.keypartitioner.options':'-k1,3', # send the records to the corresponding partition
            'mapreduce.job.output.key.comparator.class':'org.apache.hadoop.mapreduce.lib.partition.KeyFieldBasedComparator',
            'mapreduce.partition.keycomparator.options':'-k2,2n -k3,3n -k4,4n' # sort the records
        }
if __name__ == '__main__':
    CalculateEdgePair.run()
```

Given a large news articles dataset from several years (year as the key and news content as the value), use MapReduce to compute the longest length of terms for each year. You can assume that the terms are separated by a single space character. Note that one term could be split into two lines (the end of the first line and the beginning of the second line).



```
class Mapper:
    def map(self, year, content):
        # assume that the terms are separated by a single space character
        terms = content.split(‘ ‘)
        for term in terms:
            emit (year, len(term))

class Combiner:
    def combine(self, year, lengths):
        max_length = max(lengths) # Local aggregation to find the maximum length for each year
        yield (year, max_length)

class Reducer:
    def reduce(self, year, lengths):
        max_length = max(lengths)
        yield (year, max_length)
```

Q3 Spark Programming (code template will be given)

Monday, November 20, 2023 1:12 PM

Q4 Finding Similar Items

Monday, November 20, 2023 1:13 PM

k-Shingles: convert documents to sets

MinHash: convert large sets to short signatures, while preserving similarity

LSH: focus on pairs of signatures likely to be from similar documents

Q5 Mining Data Streams

Monday, November 20, 2023 1:13 PM

Sampling Data Stream **Reservoir Sampling**

DGIM

Bloom Filter

Boyer-Moore voting algorithm and Misra-Gries algorithm, lossy counting, space saving

count-min sketch

Counting data stream – FM-Sketch

Q6 Graph Data Management

Sunday, November 19, 2023 2:28 PM