Applied Algorithms:

# Finding a correlated pair

Martin Aumüller (maau@itu.dk)

Nov 2, 2020

# Course Overview

This week:

- Largeish project that incorporates most of what you’ve learned
- Correct and reasonably fast will probably be a challenge this week

# Finding Correlated Elements

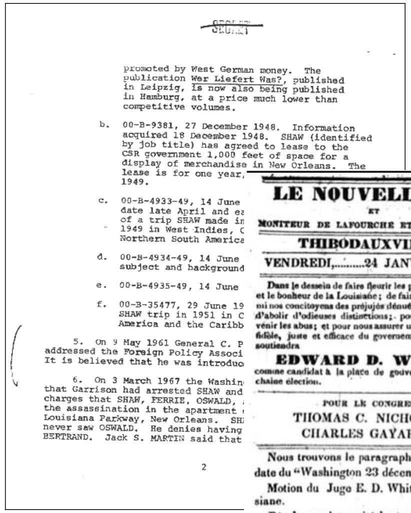
Fairly fundamental and widely used in modern computer science

**Plan for today**

- Motivate the problem and talk about what it means in general
- Restrict to the specific case we’ll be dealing with
- Solutions using algorithms we know
- New solution

# Features

Map large images, documents, etc. to digestible vectors



promoted by West German money. The publication Wer Liefert Was?, published in Leipzig, is now also being published in Hamburg, at a price much lower than competitive volumes.

b. 00-B-9381, 27 December 1948. Information acquired 18 December 1948. SHAW (identified by job title) has agreed to lease to the CSR government 1,000 feet of space for a display of merchandise in New Orleans. The lease is for one year,
1949.

c. 00-B-4933-49, 14 June
date late April and ea
of a trip SHAW made in
1949 in West Indies, C
Northern South America

d. 00-B-4934-49, 14 June
subject and background

e. 00-B-4935-49, 14 June

f. 00-B-35477, 29 June 19
SHAW trip in 1951 in C
America and the Caribb

5. On 9 May 1961 General C. P
addressed the Foreign Policy Associ
It is believed that he was introduc

6. On 3 March 1967 the Washin
that Garrison had arrested SHAW and
charges that SHAW, FERRIE, OSWALD,
the assassination in the apartment
Louisiana Parkway, New Orleans. SH
never saw OSWALD. He denies having
BERTRAND. Jack S. MARTIN said that

2

LE NOUVELLISTE,

ET

MONITEUR DE LAFOURCHE ET TERREBONNE.

THIBODAUXVILLE:

VENDREDI,..........24 JANVIER, 1834.

Dans le dessein de faire fleurir les plus chers intérêts et le bonheur de la Louisiane; de faire disparaitre parmi nos concitoyens des préjujés dénués de fondment et d'abolir d'odieuses distinctions; pour guérir et prévenir les abus; et pour nous assurer une administration fidèle, juste et efficace du goverement; cette feuille soutiendra

EDWARD D. WHITE,

comme candidat à la place de gouverneur, à la prochaine élection.

POUR LE CONGRES.

THOMAS C. NICHOLLS,
CHARLES GAYARRE.

Nous trouvons le paragraphe suivant en date du "Washington 23 décembre."

Motion du Juge E. D. White, de la Louisiane.

*Résolu*, que le comité des terres publiques, soit réquis de s'informer des avantages et de la justice qu'il y aurait à accorder, à l'Etat de la Louisiane, pour aider à ses améliorations internes, la même quantité de terre qui a été donné pour le congrès aux Etats de l'Ouest, et particulièrement à l'Etat d'Alabama.

Un navire arrivé le 18 courant à la Havane apporte des journaux de Cadix du 2 Décembre, et de Madrid du 27 Novembre, qui annoncent qu'il y a eut plusieurs engagemens sur différens points de l'Espagne, entre les troupes de la reines et les carlistes qui ont été partout défaits et mis en fuite. Le parti carliste paraissait entièrement anéanti. Nous n'avons pas pu nous procurer d'autres détails.—*Courrier*.

NOUVELLES IMPORTANTES DE ST. DOMINGUE.

Le capitaine Eldrige du navire Merrimack arrivé, hier au soir, de St. Domingue, dit que l'on attendait tous les jours, dans l'ile à être attaqué par les français et que l'on fésait de grands préparatifs pour se mettre en état de défense. On réparait les fortifications au Port au Prince, et on en établissait de nouvelle. A l'exception de la crainte d'être attaqué par les Français, rien ne troublait la tranquillité de l'ile.—*Idem*.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 15 | 0 | .2 | -3 | 4 | 22 | 1 | 1 |

# Example: Similarity Search on Words

- GloVe: learning algorithm to find vector representations for words
- *GloVe.twitter* dataset: **1.2M words**, vectors trained from **2B tweets**, **100 dimension real-valued vector associated with every word**
- Semantically similar words: nearest neighbor search on these vectors


https://nlp.stanford.edu/projects/glove/

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

# GloVe Examples (5-NN search for a query)

| “denmark” | “copenhagen” | “københavn” |
| --- | --- | --- |
| • “sweden” | • “brussels” | • “haugesund” |
| • “germany” | • “helsinki” | • “århus” |
| • “netherlands” | • “belgium” | • “sandefjord” |
| • “italy” | • “vienna" | • “kommune” |
| • “norway” | • “turin" | • “aalborg” |

# Features

Goals:

- Easier to compare and classify (can usually search features faster than we can search images)
- Want similar things to stay similar

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |

# Features

Goals:

- Easier to compare and classify (can usually search features faster than we can search images)
- Want different things to stay different

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

# Using Features

- We want to perform operations on these vectors

- Similarity queries (reverse image search), find close pair (plagiarism detection), much more complicated operations

# Bit Vectors

- Again: inputs are bit vectors
- Similar bit vectors = high *inner product*
  - A.k.a. the number of 1's in common

Recall: Inner Product = number of ones in x & y

Cheap!

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |

# Bit Vectors

- Dissimilar bit vectors = low *inner product*
  - A.k.a. the number of 1's in common

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

# Our Problem Today

- Given a list of bit vectors, and a given similarity (inner product), find the pair with at least that similarity.
  - Similarity is given to you (will always be the same in all tests)
  - Guaranteed to have one unique pair

Problem kind of orthogonal to “Orthogonal Vectors”

# Our Problem Today

- Given a list of bit vectors, and a given similarity, find the pair with at least that similarity.

Find pair with similarity $\geq 3$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **1** | **0** | **1** | **0** | **1** | **0** | **1** | **1** |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **1** | **0** | **1** | **0** |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **1** | **0** | **1** | **1** | **0** | **0** | **1** | **1** |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0** | **0** | **1** | **1** | **0** | **1** | **0** | **0** |

# Our Problem Today

- Given a list of bit vectors, and a given similarity, find the pair with at least that similarity.

Find pair with similarity >= 3

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |

Similarity 4

# Our Problem Today

- Given a list of bit vectors, and a given similarity, find the pair with at least that similarity.

Find pair with similarity >= 3

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |

Similarity 1

# Simplifying The Problem

- Similarity threshold is given
- Only one close pair
- All vectors have same length $\ell$

Expected similarity between correlated/non-correlated pair?

- All bit vectors are generated randomly:
  - n-1 of them are generated at random; each bit is independently set with probability 1/3 (expect $\ell/3$ ones)
  - In particular: each bit is set with the same probability
  - Then we generate one similar to a previous vector. With probability 7/8 it has the same bit; otherwise it's random.

# Simplifying The Problem

- Similarity threshold is given ( $\geq 70$)
- Only one close pair
- All vectors have same length (256 bits)

- All bit vectors are generated randomly

These assumptions should hopefully make the code easier to write

# How do we solve this?

Simple brute force algorithm?

- Compare each pair of vectors
- $O(n^2)$ comparisons
- How do we compare two 256-dim bit vectors quickly?
  - 4 bitwise ANDs
- Can evaluate n = 100 000 vectors in about 35 seconds on my laptop
- Let’s do this together!

How do we improve over this?

# Heuristics

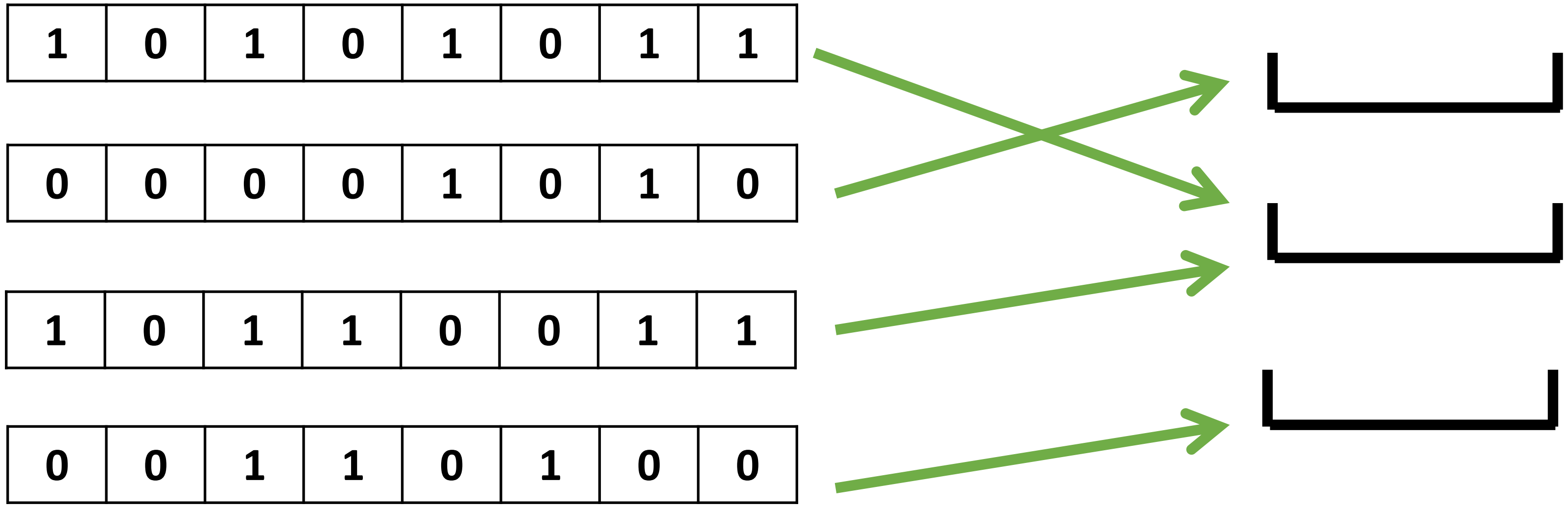
- Be creative
- I tried stopping each comparison short if the first 64 bits (the first AND) had low correlation.  Gave ~50% speedup
  - Only OK because the data is generated randomly.  Could make us miss the close pair on certain datasets

# MinHash

- Way to hash bit vectors while retaining information about their similarity
  - “Locality-sensitive hashing”
- Used fairly widely in practice
- We will modify it a little bit today for our purposes

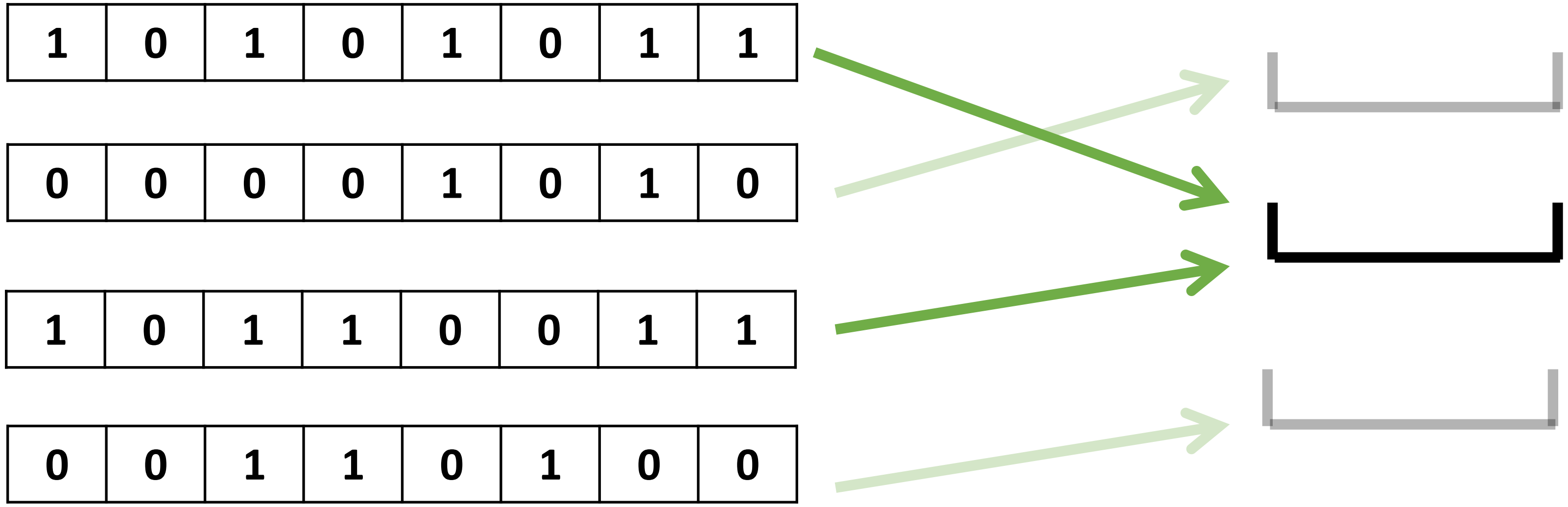
# Goal of MinHash

- Hash vectors to buckets
- Only compare two vectors if they are in the same bucket
- Notion: Two vectors "*collide*" if they are in the same bucket



# Goal of MinHash

- Hash vectors to buckets
- Only compare two vectors if they are in the same bucket
- Notion: Two vectors "*collide*" if they are in the same bucket




Only need to compare these two

# Goal of MinHash

- Hash vectors to buckets
- Only compare two vectors if they are in the same bucket
- Different to HyperLogLog hashing:
  - Want that similar items collide!
  - Not spread them out randomly

What we need:

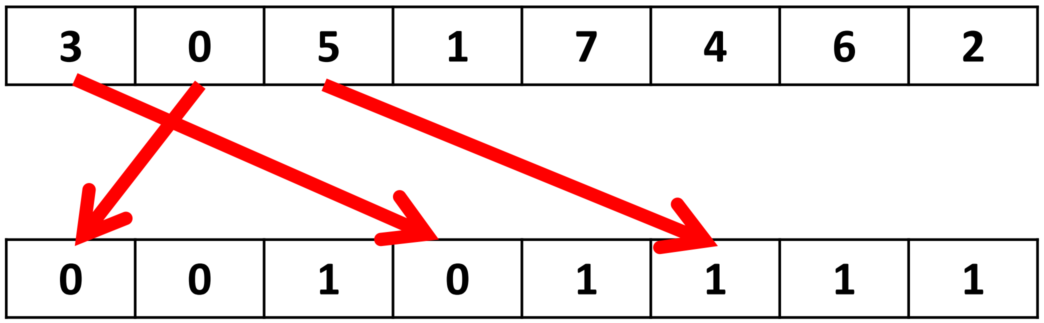
- Buckets shouldn’t be too big
- Similar pair should be likely to be in the same bucket.

If we don’t find the similar pair, try again with new hash functions

# MinHash

- Choose your MinHash function: A random permutation of {1,…,256}
- To hash: Take the position of the first 1 in the vector encountered by traversing the permutation
- That position is the MinHash of the vector

Random permutation:




MinHash of this vector is 5, because the first 1 we found is in position 5

# Very brief analysis

- Probability that x and y hash together is proportional to similarity(x,y)
- The precise value is the Jaccard Similarity:

(# 1s in common)/(total positions that have a 1 in either)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |

If one of these is the min
they collide

# Very brief analysis

- Probability that x and y hash together is proportional to similarity(x,y)
- The precise value is the Jaccard Similarity:

(# 1s in common)/(total positions that have a 1 in either)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **1** | **0** | **1** | **1** | **0** | **0** | **1** | **1** |
| **1** | **0** | **1** | **0** | **1** | **0** | **1** | **1** |

If one of these is the min
they don't collide

# Very brief analysis

- Probability that x and y hash together is proportional to similarity(x,y)
- The precise value is the Jaccard Similarity:

(# 1s in common)/(total positions that have a 1 in either)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |

These can never be the min
so we ignore them

# Very brief analysis

- Probability that x and y hash together is proportional to similarity(x,y)
- The precise value is the Jaccard Similarity:

(# 1s in common)/(total positions that have a 1 in either)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |

6 things to choose from, 4 collisions

Prob(collision) = 4/6 = 2/3

# How much does this help us?

- How likely are we to see two random vectors collide?
- Order of magnitude when hashing 1 million elements:
  - Bucket size ~100k
  - Bucket size ~1k
  - Bucket size ~1

- Pretty likely to collide—something like probability 1/3 (so bucket size ~100k)

- This isn't worth it, so what do we do?

# Hash several times

- Use new hash function to split our buckets further (in other words, a new random value for each bit *position*)
  - Have to be careful—have to choose new hash functions, not reuse old ones!
- Two vectors only collide if they collide in each of our several hashes

# Hash several times

Hash = 5

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **3** | **0** | **5** | **1** | **7** | **4** | **6** | **2** |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0** | **0** | **1** | **0** | **1** | **1** | **1** | **1** |

Hash = 6

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **6** | **2** | **0** | **7** | **1** | **4** | **5** | **3** |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0** | **0** | **1** | **0** | **1** | **1** | **1** | **1** |

Hash = 7

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0** | **7** | **1** | **3** | **2** | **4** | **5** | **6** |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0** | **0** | **1** | **0** | **1** | **1** | **1** | **1** |

So our MinHash is (5,6,7). We will only compare to vectors that also hash to (5,6,7)

# Hash several times

- Our collision probability is getting low. What happens when we compare all to all and we don’t find the pair of vectors?
- Repeat again with a new hash
- If we know the close vectors have probability $p$ of collision, how many expected repetitions do we need?
  - $1/p$

- Don’t need to parameterize by this; can just keep trying until you get the vector

# How big do we want our buckets?

- Definitely small enough to fit in memory
- Run experiments to see what is fastest

- How to control bucket size?
  - Different # concatenated hashes

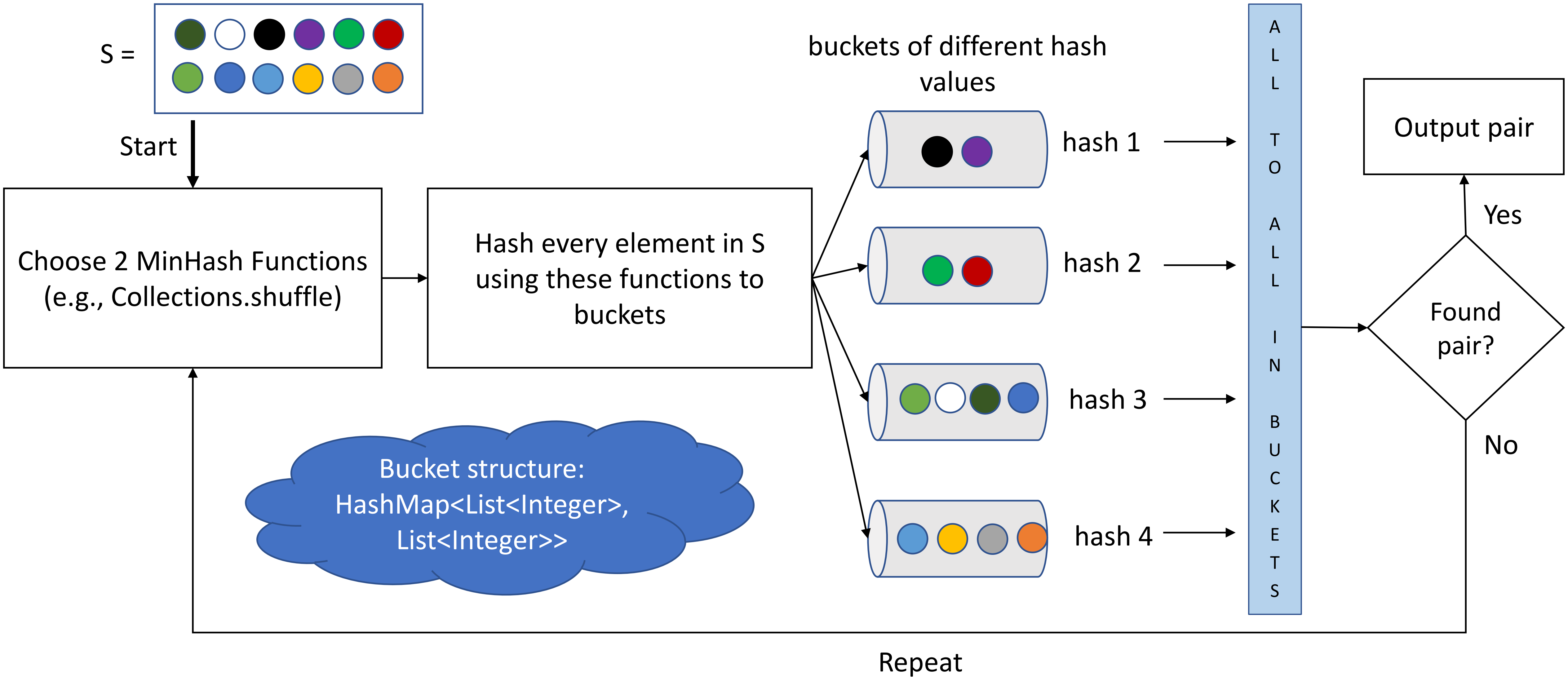

# Implementation

- All-compare-all within a bucket uses the naïve method which we have already implemented
  - Just need to get the vectors into an array (or vector, etc)

- How do you allocate buckets?
  - Dynamically (?)
  - Can count bucket sizes first

- How do you *label* buckets?
  - Don’t want to create a bucket for all possible MinHashes (most will be empty)
  - Use hash table?
  - Only 256 positions, i.e., MinHash has 8 bits. Store many of them as one 32-bit or 64-bit integer?

# Overview w/ 2 MinHash functions on input S



# Speeding Up

- MinHash is much much faster can do 50 000 vectors in 0.3 seconds; naïve takes 184 seconds

- This speedup is probably language-independent, but need to make sure you don’t lose time

# Assignment

- Input: List of 256-dimensional vectors, each listed as 4 signed 64-bit integers, from stdin, randomized as described before
- Output: index of the correlated pair, smaller index first

- CodeJudge
  - No race
  - Largest instance you can solve in 10 seconds
  - 1k, 10k, 100k, etc
  - Should be fairly tamper-proof; let me know if it’s not