Genome 540 discussion

January 21st, 2025
Joe Min

Agenda

Homework 2 overview

Dictionaries and hash functions

Homework 2 overview

Homework 2: build regional model(s)

Get nucleotide frequencies

- Read in the fasta for the mouse 10mb region
- Nucleotide counts and frequencies
- Dinucleotide counts and frequencies
    - Additionally, dinucleotide *conditional* frequencies

Homework 2: use the models generatively

For each of the three mouse models (equal frequency, 0-th order Markov, 1st order Markov), generate a 10mb sequence

- Generate the sequence base-by-base (iteratively) by appropriately sampling from model frequencies

Homework 2: calculating frequencies

The equal frequency model:

- A: 0.25
- C: 0.25
- G: 0.25
- T: 0.25

Homework 2: calculating frequencies

0-th order Markov model

- Sequence: ACTGA
  - Nucleotide counts:
  - A: 2
  - C: 1
  - G: 1
  - T: 1

Total: 5

$\xrightarrow[\text{(5)}]{\text{Divide by total}}$

- Sequence: ACTGA
  - Nucleotide frequencies:
  - A: 0.4
  - C: 0.2
  - G: 0.2
  - T: 0.2

Order 1 Markov Model

Sequence: ACTGATGATGGTACA
Length = 15; number of dinucleotides = 14

first nucleotide

	A	T	G	C
A	0	2	0	2
T	1	0	3	0
G	2	1	1	0
C	1	1	0	0

Dinucleotide Frequencies
e.g. # AT = 2

	A	T	G	C
A	0	.143	0	.143
T	.071	0	.214	0
G	.143	.071	.071	0
C	.071	.071	0	0

Dinucleotide Probabilities
e.g. P(AT) = 0.143

	A	T	G	C
A	0	.5	0	.5
T	.25	0	.75	0
G	.5	.25	.25	0
C	.5	.5	0	0

Nucleotide
Conditional Probabilities
e.g. P(T|A) = 0.5

Homework 2: use the models generatively

For each of the three mouse models (continued)

- Output simulated sequence to a fasta file
- Run your HW1 program between the simulated sequence and the human 10mb region
  - Simulating sequences should be a relatively quick process, so if your HW1 takes a long time, might be best to start early!

Homework 2: final thoughts

Make sure to submit short answers to questions in part 4

Please match the template

- Can use the command line utility `diff` or even just an online text comparison tool
- Will begin to dock points

Dictionaries and hash functions

Dictionaries

Dictionaries (hashmaps in C++) are data structures that are a collection of data values that can be accessed by their corresponding keys

- { ‘key_1’: ‘value_1’,
  ‘key_2’: ‘value_2’,
  …
  ‘key_n’: value_n’ }

Dictionaries

Values can be any data type

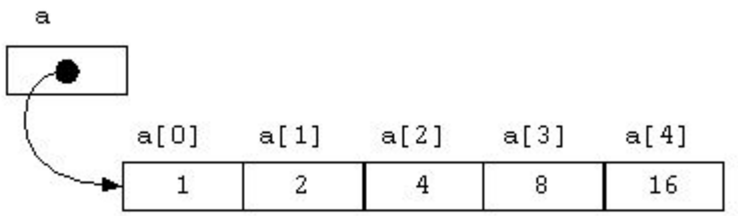
Keys have different requirements per language, but in general the keys can’t change

- Why? Let’s take a look at its implementation

…but first, back to lists

Remember: lists and arrays are sequential chunks physical memory

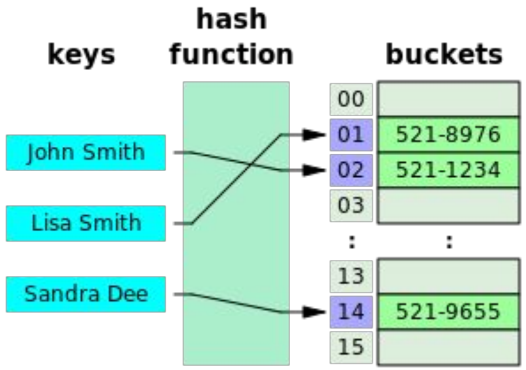
Here we have an array “a” holding 5 integer values



https://www2.hawaii.edu/~walbritt/ics212/materials/arrays_h1_381x122.gif

Dictionaries hash keys

Dictionaries instead use a "hash function" to transform the keys into memory addresses



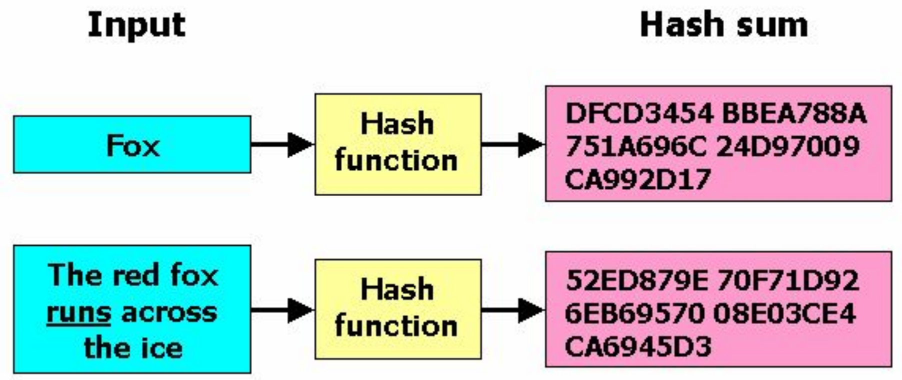
- For example, the key "John Smith" is passed to a function that returns "01" as the memory location
- The value at this entry is the string "521-8976"
- Values live in discontinuous locations of memory

https://cdn.sanity.io/images/kuana2sp/production-main/c015c55fca5c286607aff134760bb35e5d54db32-1011x704.webp

Hash functions

Broadly, hash functions take in a variably-sized input and map them to a fixed-size output

- A very trivial example could be the *modulo* function (%)



https://ms.codes/cdn/shop/articles/0_VfVhDHeHJW7LKpkr.png?v=1707823797

Hash functions

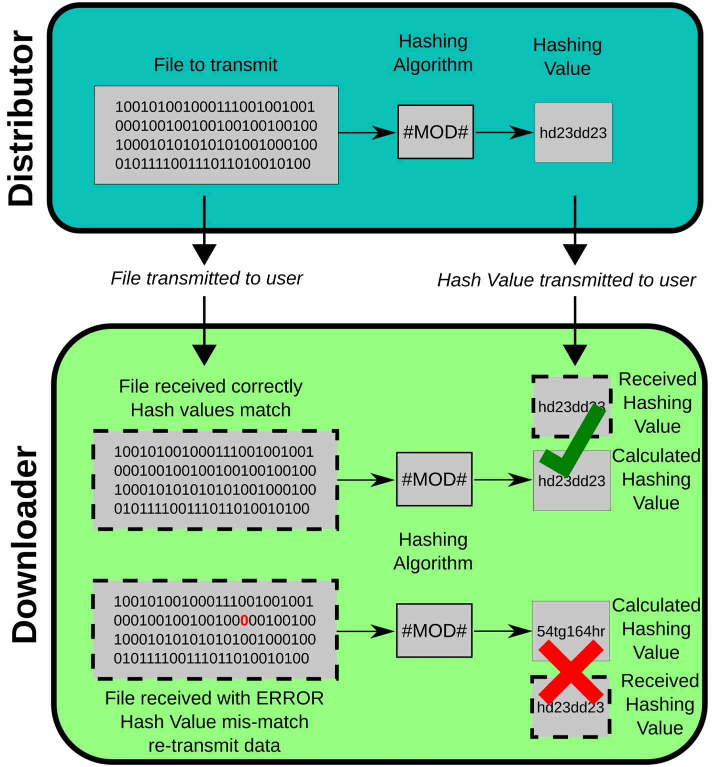
Other key properties of a hash function are it should:

- Be deterministic (otherwise we could never retrieve the correct value for the same key)
- Uniformly distribute output (to avoid memory location collisions as much as possible)
- Experience an “avalanche effect” (small changes in input should have drastic implications for the output)

Example usage: file sameness

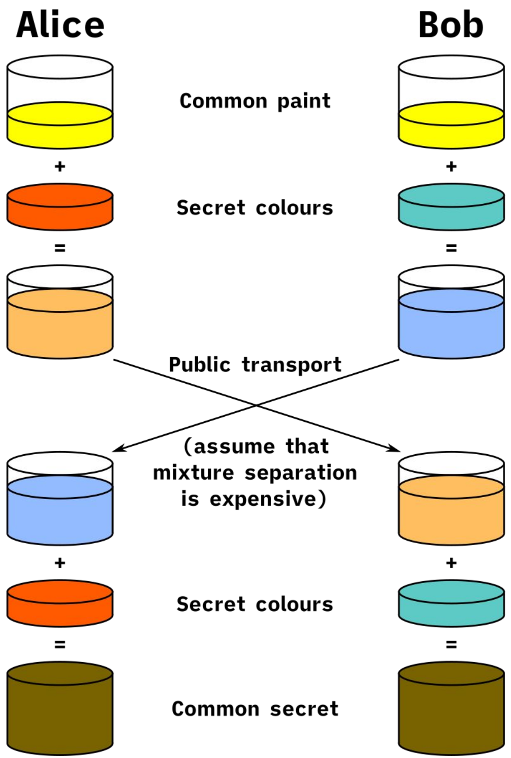
Often, hash function "checksums" are used to ensure a downloaded file is the same as the intended file

- Pass the whole file to a hashing algorithm



https://en.wikipedia.org/wiki/MD5#/media/File:CPT-Hashing-File-Transmission.svg

Example usage: public/secret key exchange



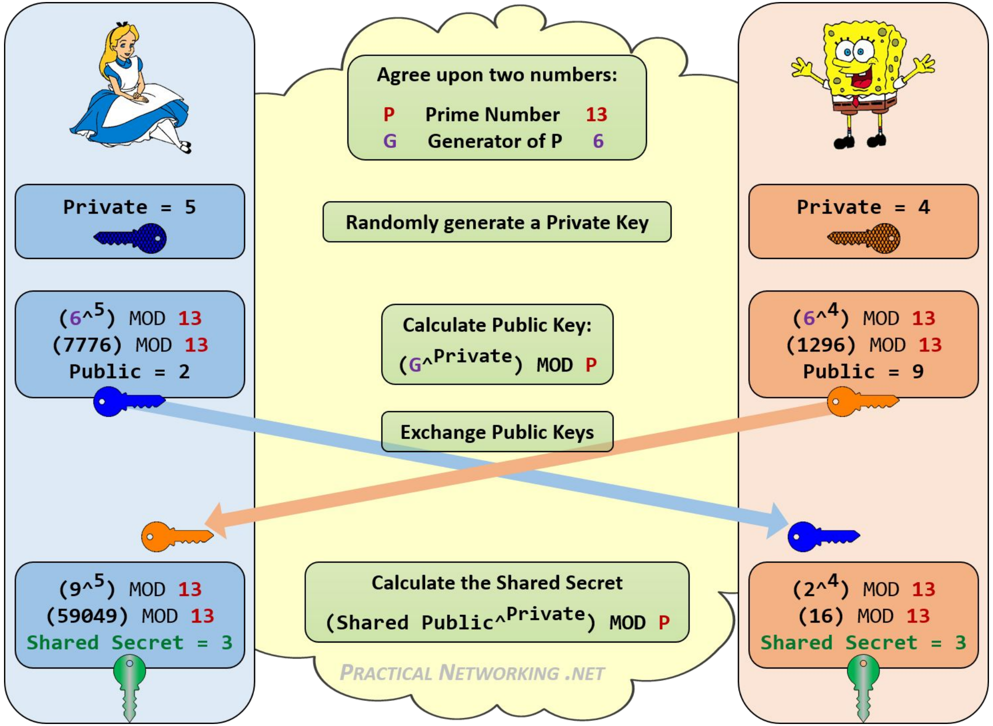
In Diffie-Hellman key exchange (e.g., when using an SSH key), hashing a known quantity (public key) with an unknown quantity added (secret key) creates a unique outcome that only the intended secret key can use or decode

https://traefik.io/static/e1495bb25e44a6cbd76273ca34ecff77/800px-Diffie-Hellman_Key_Exchange.svg.png

Example usage: public/secret key exchange

Stepping out of the abstract paint analogy:

- The shared secret can now be used for things like authentication



https://www.practicalnetworking.net/series/cryptography/diffie-hellman/

Office hours

Good work on homework 1!

Reminder:

Homework 2 is due this Sunday (Jan 26) at 11:59pm!