Hash Table Size

How large should it be?

World Records of Table Sizes

Quick Review

- Hash Table with Chaining
  - Each array slots stores a linked list.
  - All items mapped to the same slot are stored in the linked list.

- Open addressing:
  - Each array slot stores one element.
  - On collision, continue probing.
  - Probe sequence specifies order in which cells are examined.

How large should the table size be?

- #items = n and table size = m
- Assume: Simple Uniform Hashing
  - Expected search time: $O(1 + n/m)$
  - Optimal size: $m = \theta(n)$

- if $(m < 2n)$: too many collisions.
- if $(m > 10n)$: too much wasted space.

- Problem: we don’t know n in advance.

Idea?

- Start with small (constant) table size.
- Grow (and shrink) table as necessary.

Idea?

- Start with small (constant) table size.
- Grow (and shrink) table as necessary.

- Example :
  - Initially, $m = 10$.
  - After inserting 6 items, table too small! Grow…
  - After deleting $n - 1$ items, table too big! Shrink…

Time complexity of growing the table:

- Assume:
  - Let m_1 be the size of the old hash table.
  - Let m_2 be the size of the new hash table.
  - Let n be the number of elements already in the hash table.
- Costs:
  - Scanning old hash table: $O(m_1)$
  - Creating new hash table: $O(m_2)$
  - Inserting *each* element in new hash table: $O(1)$
  - Total: $O(m_1 + m_2 + n)$

How fast should we grow?

- Idea 1: Increment table size by 1

$$\text{if } (n == m_1)\text{: } m_2 := m_1 + 1$$

- Cost of resize:
  - For each insertion after table is full: $\mathrm{O}(m_1 + m_2 + n)$
  - **Each** new insertion needs $\mathrm{O}(n)$

How fast should we grow?

- Idea 2: Double the size of the table

$$\text{if } (n == m_1)\text{: } m_2 := 2m_1$$

- Assuming n is very large
  - resizing occurs when n was
    - n /2, n /4, n /8, …
  - Total time complexity =

$$O(1 + \ldots + n/16 + n/8 + n/4 + n/2 + n) = O(n)$$

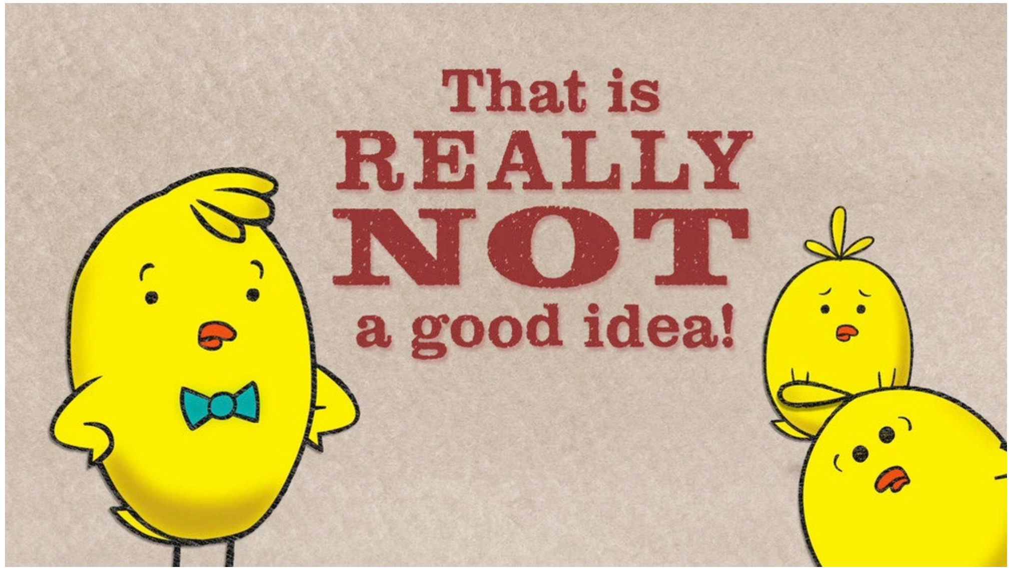
  - In average, every addition of an item cost $O(1)$

How fast should we grow?

- Idea 3: More the merrier!!! Let's square the size!

$$\text{if } (n == m_1)\text{: } m_2 := {m_1}^2$$

- Why is it not a good idea?
- When the point of time $n > m_1$, already $O(n^2)$



How fast should we grow?

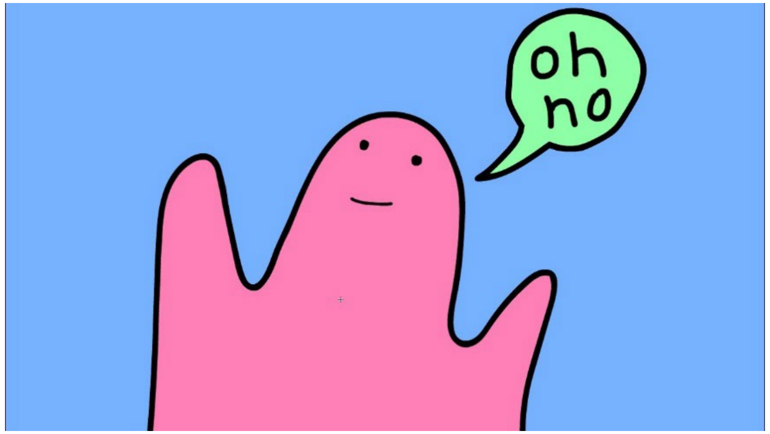
- ~~Idea 1: Increment table size by 1~~
- Idea 2: Double the size of the table
- ~~Idea 3: Square the size!~~

How about shrinking the table?

- Table is too big! Shrink the table…
- Try 1:

$$\text{if } (n == m_1/2)\text{: } m_2 := m_1/2$$

- However…
  - Start: $n = 100$, $m = 200$
  - Delete: $n = 99$, $m = 200$ → shrink to $m = 100$
  - Insert: $n = 100$, $m = 100$ → grow to $m = 200$
  - Repeat…
- What is the time complexity for EACH insertion?
- What should we do?



Deleting Elements

- Try 2:
  - if ($n ==\ m_1$): $m_2 := 2m_1$
  - if ($n < m_1/4$): $m_2 := m_1/2$

- Claim:
  - Every time you double a table of size m, at least m/2 new items were added.
  - Every time you shrink a table of size m, at least m/4 items were deleted.

Applications of Hashing

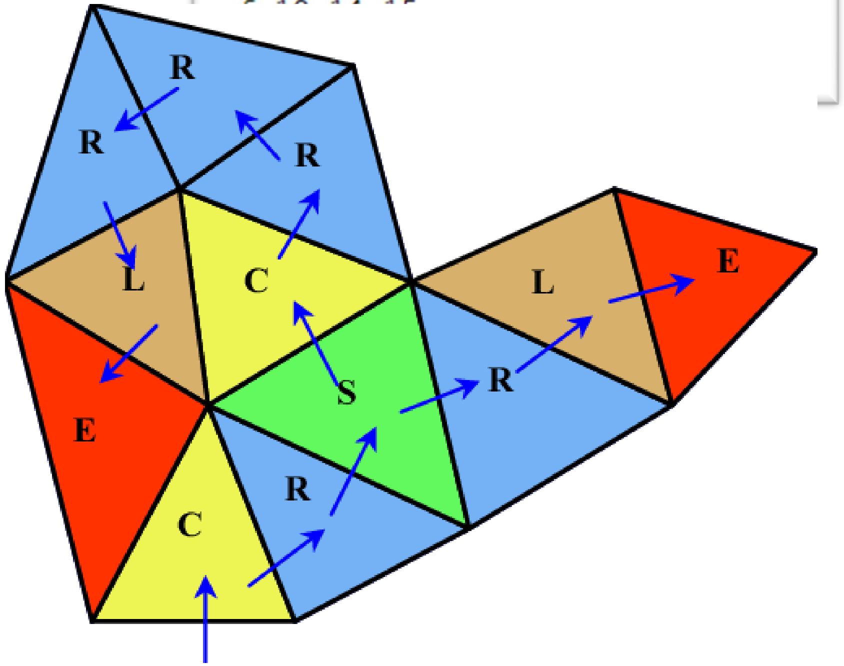
Symbol Table Applications

- 3D Objects
- E.g. OBJ Wavefront files
  - Each triangle has three vertices
  - But how do I connect them as a mesh?

Connecting Triangles

- For each triangle, I want to know its adjacent triangle neighbors
- A lot of 3D file format only give you the three vertex indices of each triangle
- E.g.
  - Triangle A with vertices 3, 4, 2
  - Triangle B with vertices 1, 2, 4
  - Triangle C with vertices 5, 6, 3
- Triangles A and B are sharing one edge
  - Because they both have vertices 2 and 4

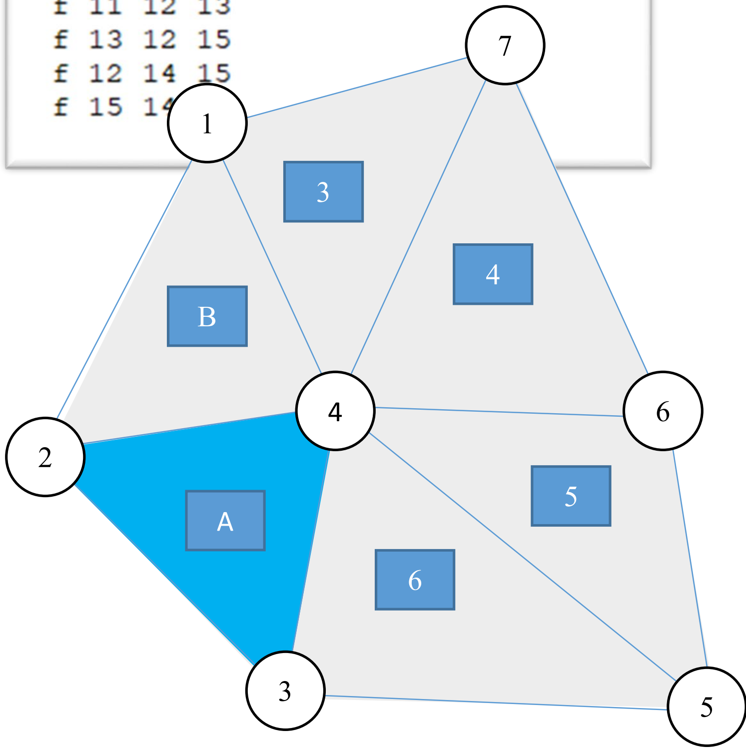
```
v 3.065544 -0.788768 0.000000
v 3.107533 -0.691019 -0.264602
v 3.118124 -0.703925 0.000000
v 3.053144 -0.609681 -0.326931
f 3 4 2
f 1 2 4
f 5 6 3
f 4 3 6
f 6 5 7
f 7 8 6
f 8 9 6
f 10 4 9
f 6 9 4
f 11 1 10
f 4 10 1
f 10 12 11
f 11 12 13
f 13 12 15
```



Connecting Triangles

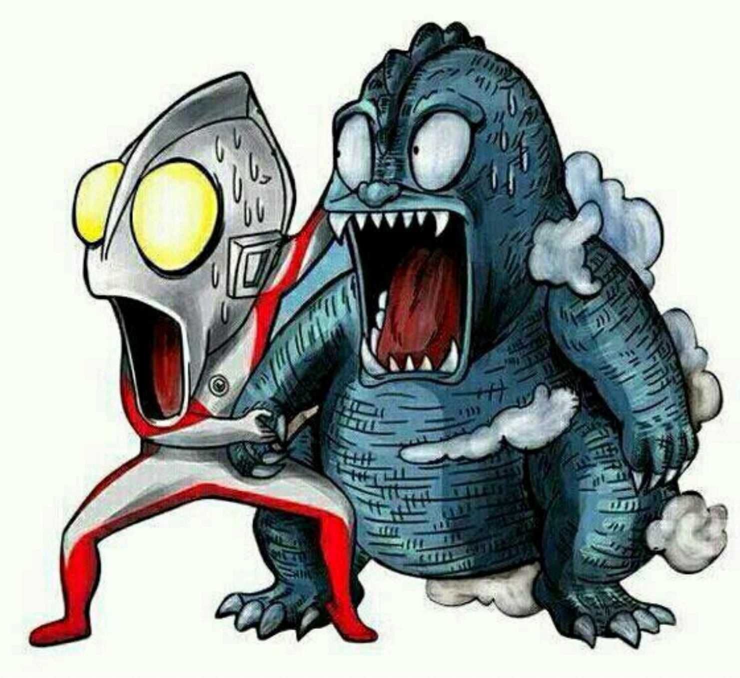
- For each triangle, I want to know its adjacent ones
- Triangles A and B are sharing one edge
  - Because they both have vertices 2 and 4
  - How do I know A and B are sharing one edge?
- Solution:
  - Hash `(key, value) = (edge, triangle)`
  - e.g. For triangle A, hash `((2,4), A), ((3,4),A))` and `((2,3),A)`
  - Before we put another edge into the table, we check if the edge `(2,4)` exists first
    - e.g. `((2,4), B)`

```
v 3.065544 -0.788768 0.000000
v 3.107533 -0.691019 -0.264602
v 3.118124 -0.703925 0.000000
v 3.053144 -0.609681 -0.326931
f 3 4 2
f 1 2 4
f 5 6 3
f 4 3 6
f 6 5 7
f 7 8 6
f 8 9 6
f 10 4 9
f 6 9 4
f 11 1 10
f 4 10 1
f 10 12 11
f 11 12 13
f 13 12 15
f 12 14 15
f 15 14
```



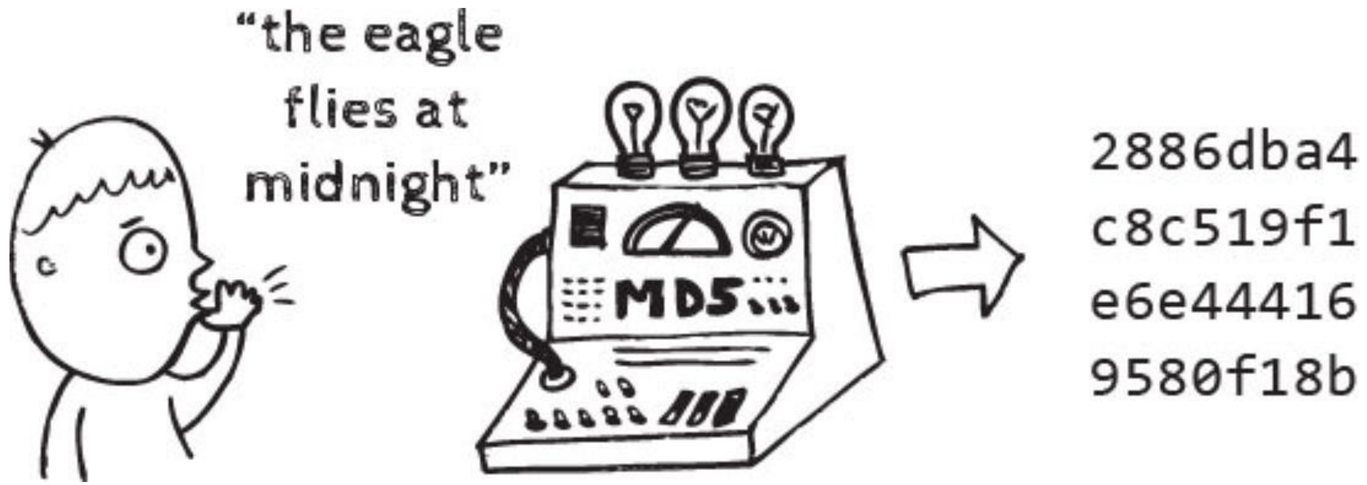
Other Applications

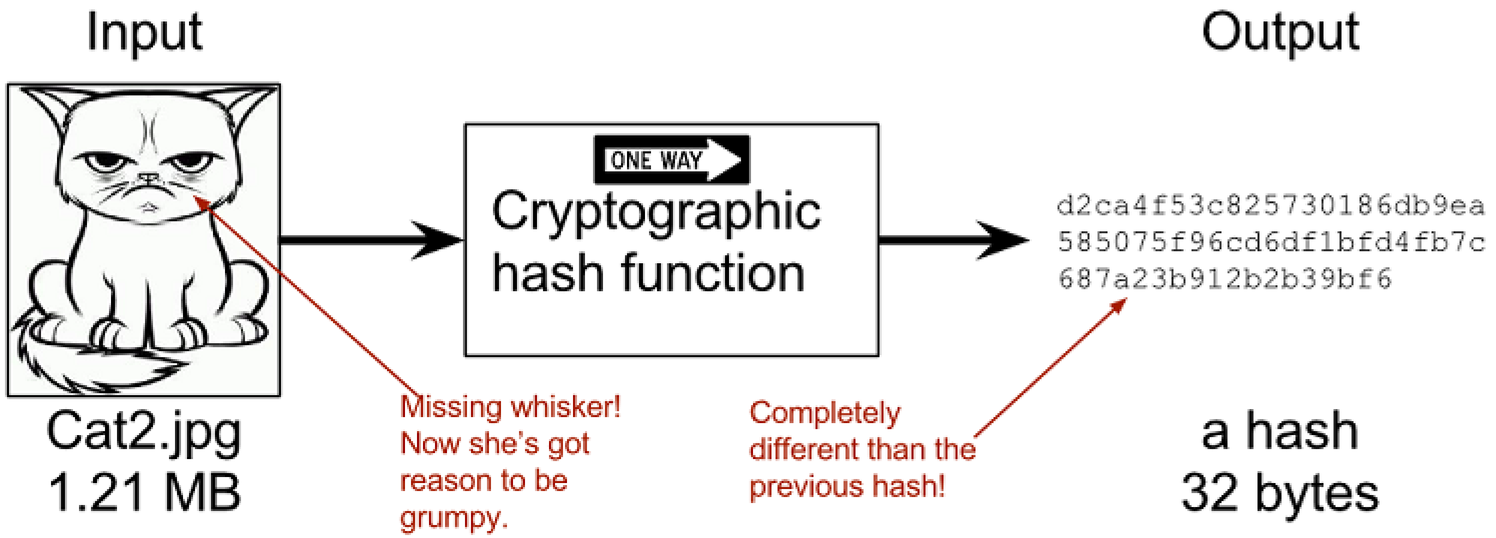
- File system
- Password verifications
  - Assuming hard to have collision
  - There exists another ‘password’ that can unlock your account
  - Hashing ≈ one way Encryption
- Online Storage
  - Hashing as a digital signature



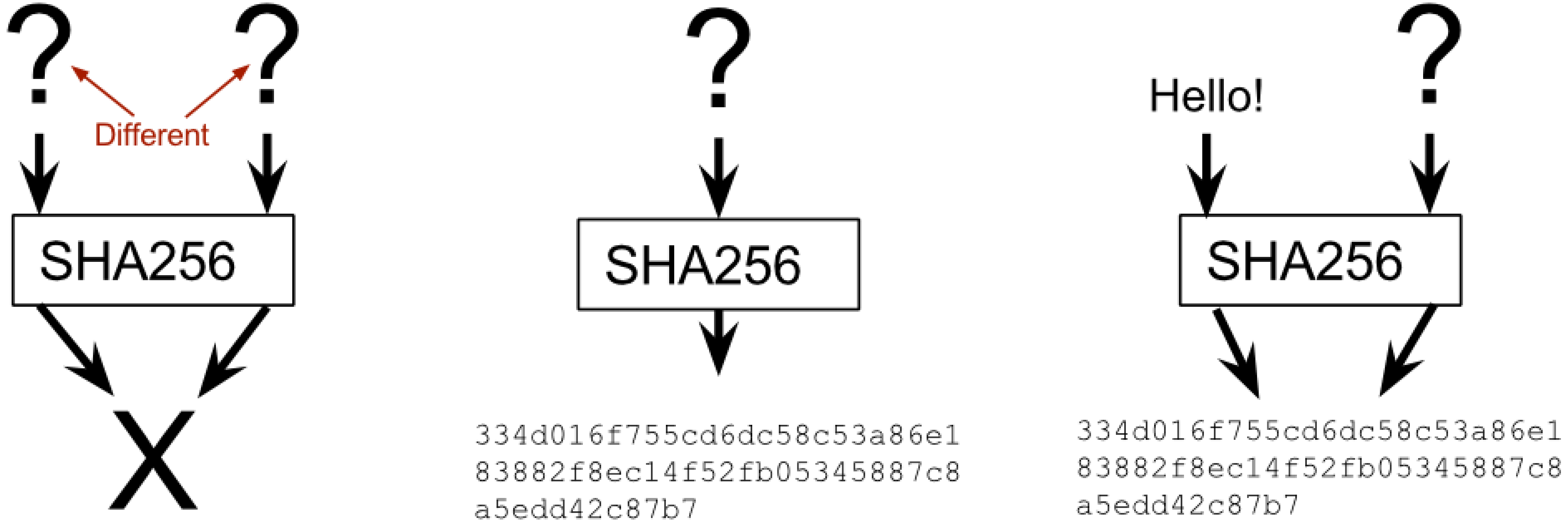
Fingerprint

- Cryptographic hash function
  - Cryptocurrencies





- Collision resistance
  - It’s hard to find two inputs that give the same hash.
- Preimage resistance
  - It’s hard to find an input that gives a certain hash.
- Second-preimage resistance
  - It’s hard to find an input that gives the same hash as a certain other input.



Collision resistance | Preimage resistance | Second-preimage resistance