What's going to be different about our last paper from what we've seen so far

input

output

runtime : worst - case

Problem	Input	Output	Runtime
Max flow	$G = (V,E), s,t \in V, C: E \to \mathbb{R}$	Maximum flow $F: E \to \mathbb{R}$	$O(E^{1+o(1)})$ $O(VE^2)$
Flow decomposition	$G = (V,E), s,t \in V, F: E \to \mathbb{R}$	P, w decomposing flow	$O(E^2)$
Minimum flow decomposition	$G = (V,E), s,t \in V, F: E \to \mathbb{R}$	P, w decomposing flow $\|P\|$ minimized	NP-Hard
Linear programming	$\max c^t x$ $Ax \leq b$ $x \geq 0$ $x \in \mathbb{R}^d$	Feasible x^* maximizing objective (or infeasible/unbounded)	Matrix multiplication time
Integer linear programming	$\max c^t x$ $Ax \leq b$ $x \geq 0$ $x \in \mathbb{Z}^d$	Feasible x^* maximizing objective (or infeasible/unbounded)	NP-Hard

Single output

Data structures vs. algorithms

Data structures	algorithms
well-defined, specific input	well-defined, specific input
thing that can give many outputs (queries)	one output
space	runtime
queries must be very fast e.g. Google	okay (??) w/ slower runtimes in some case e.g. modeling physical systems

Goals for today

Query that it has to enable?

data structures: hash tables

bloom filter

↳ set membership query

randomness

estimate vs. exact answer

What do you already know about hash functions and hash tables?

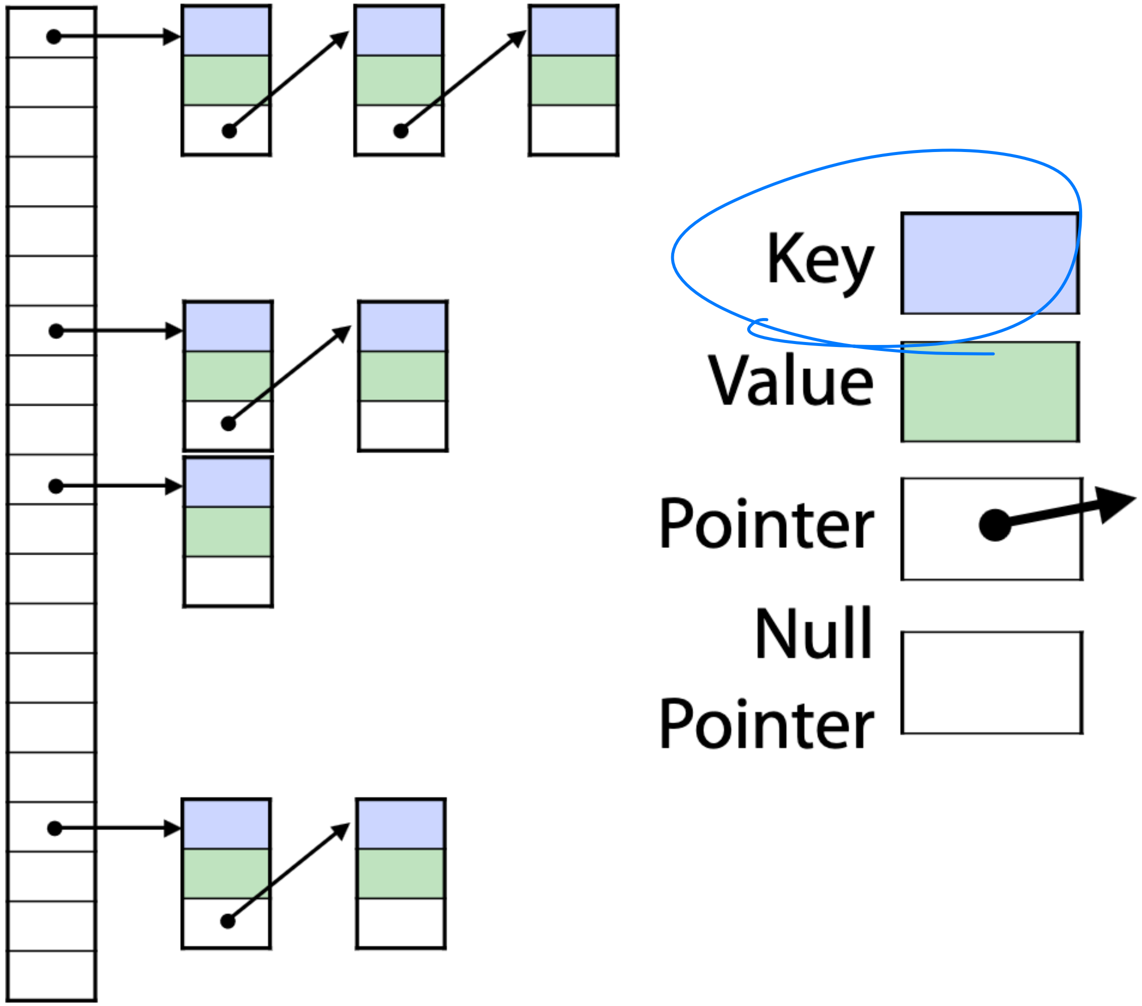
list at least 5 things

Hash Table

"Hashing with chaining" or *"chain hashing"*

Lucy, Levy
Daniel, Mango
⋮

n



~~dictionary:~~
key: value

Lucy : Levy

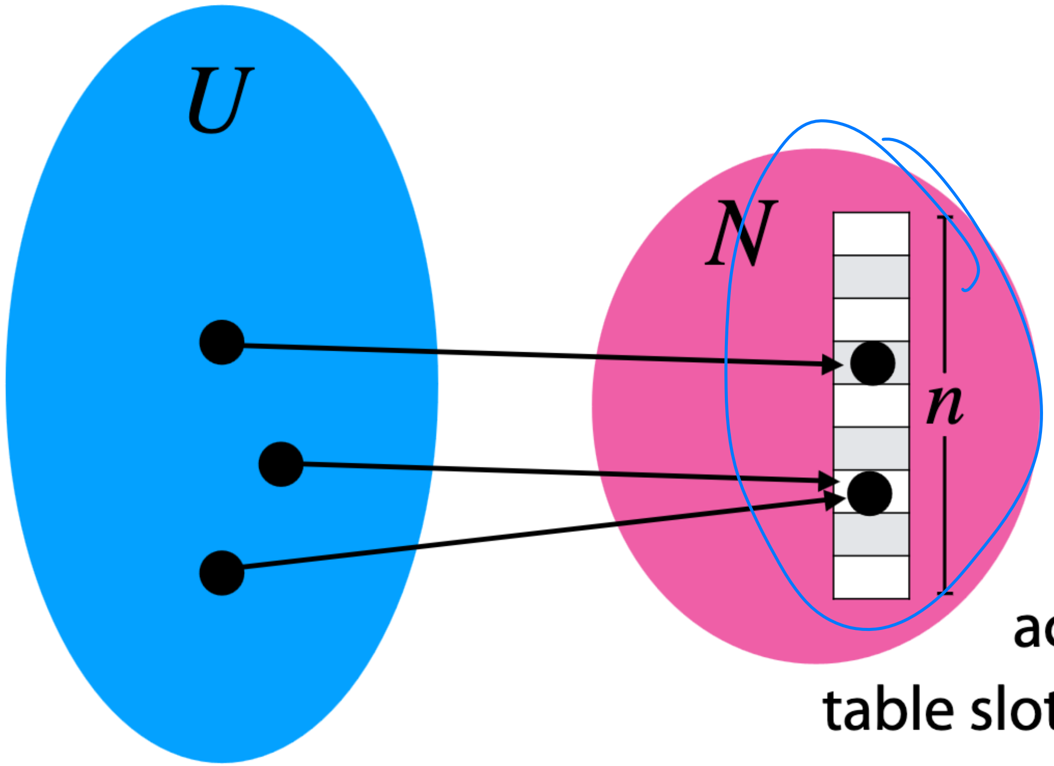
insert(Lucy, Levy)

query(Lucy)

insert : O(1)

query : O(1)

Hash Function

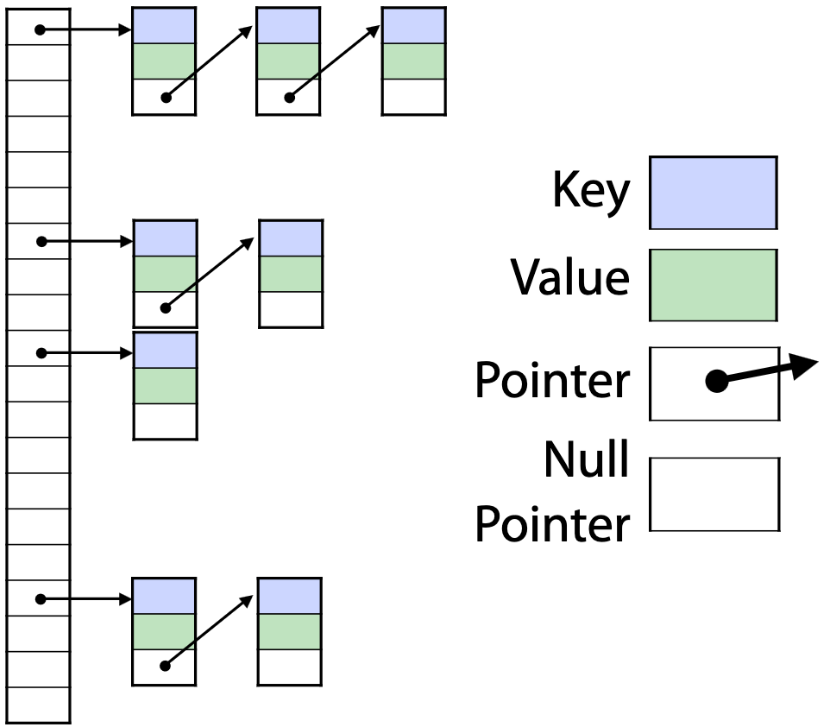


Assume accessing table slot is $O(1)$

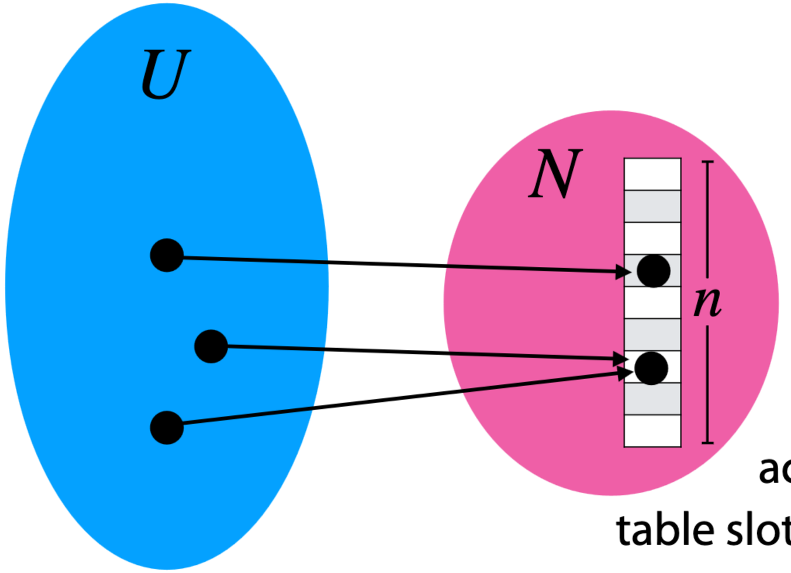
Assume hash function operates on any item from U (integers, strings, etc) and is $O(1)$ time

Hash Table

"Hashing with chaining" or *"chain hashing"*



Hash Function



Assume accessing table slot is $O(1)$

Assume hash function operates on any item from U (integers, strings, etc) and is $O(1)$ time

Hash Function

```
int hash(int x) {
    int a = 349534879; // randomly chosen
    int b = 23479238;  // randomly chosen

    ...

    // return some function of x, a and b
}
```

E.g. The family $h_{a,b}(x) = (ax + b) \mod p$ where p is prime & a, b are uniform, independent draws from $\{0, 1, \ldots, p-1\}$

When did we choose a and b?

Spread input over hashtable:

- truly random hashfunction:

```
hash(x):
    return random_num mod n
```

- completely deterministic

```
hash(x):
    x mod n
```

Algorithm phases

Phase 1

Choose algorithm

Determines *where* randomness is needed & *how much*

Phase 2

Random interlude

Make random draws.

Choose hash functions.

Phase 3

Data arrives; Execute!

Use hash functions chosen in Phase 2.

Algorithm phases

Random variables used in analysis are random over the ***choice of hash functions***

Not over the input data

We make ***no distributional assumptions*** about the input.

Phase 1

Choose algorithm

Phase 2

Random interlude

Phase 3

Data arrives; Execute!