CS50 Week 6

Sam Green ’17

samuelgreen@college.harvard.edu

646-457-2340

Agenda: Data Structures

Linked Lists

Hashtables

Stacks

Queues

Linked Lists (1)

List data structure

Supports dynamic changes in length

Gives up random access

Linked List (2)

Example Struct

```
typedef struct node
{
    int i;
    struct node* next;
}
```

Linked List (3)

As an exercise, let’s draw:

(1) Create
(2) Insert
(3) Lookup/Iterate
(4) Delete

for a linked list.

Linked List (4)

Let’s look at some code.

Linked Lists (4)

Key Takeaway: Pointer Hygiene.

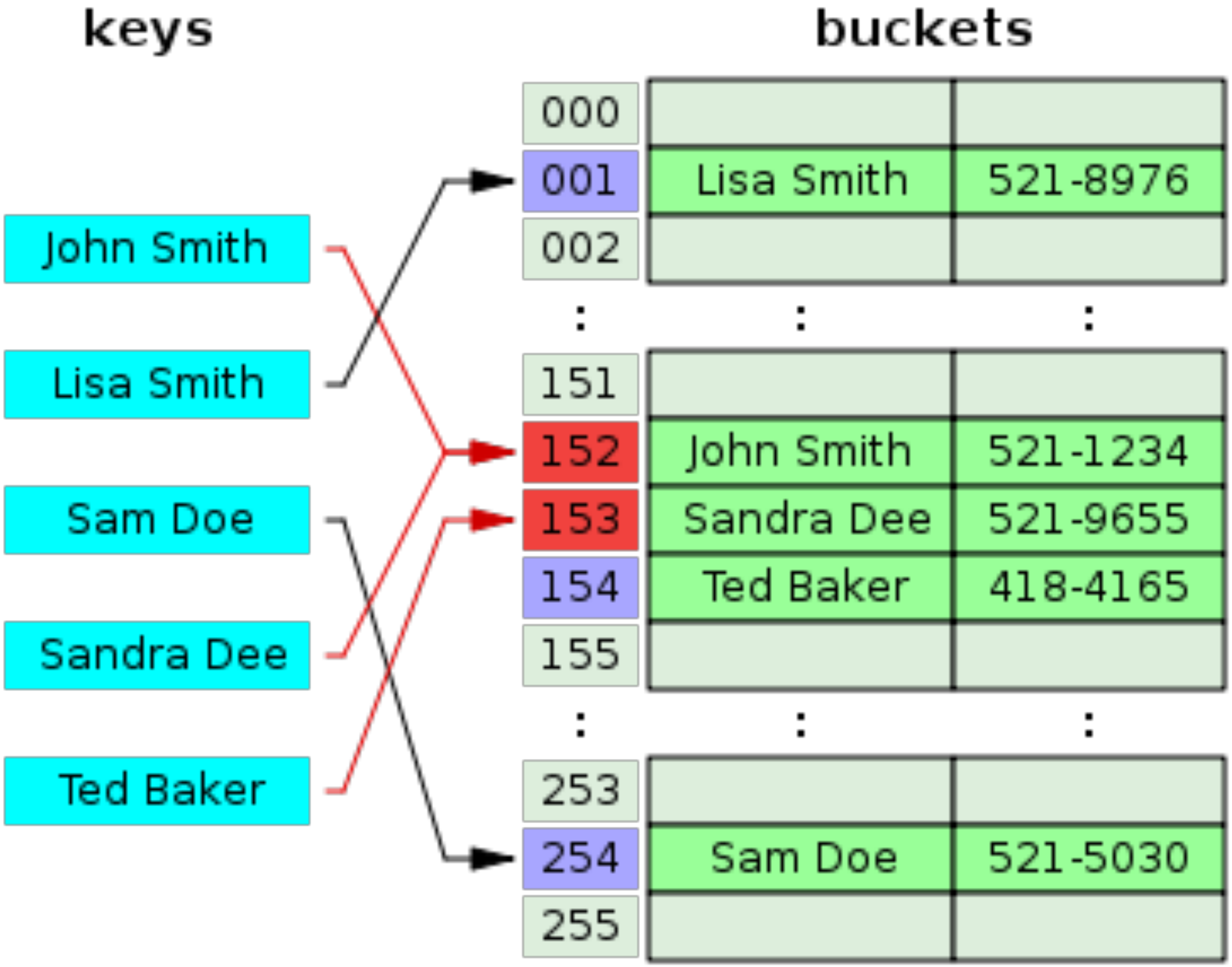
If you can draw a clear picture of the linked list you need to work with, coding it will be easy!

Hashtables (1)

Main Idea: More Pointer Hygiene

Chaining vs. Linear Probing

Hash Functions



keys

John Smith

Lisa Smith

Sam Doe

Sandra Dee

Ted Baker

buckets

000		
001	Lisa Smith	521-8976
002		
:	:	:
151		
152	John Smith	521-1234
153	Sandra Dee	521-9655
154	Ted Baker	418-4165
155		
:	:	:
253		
254	Sam Doe	521-5030
255		

Hash Function

Think of a hash function as a sorter.

(1) Give it a value.

(2) It gives you back where that value should go.

Let’s think back to the address book analogy.

What makes a hash function good?

Hash Tables

What happens after hashing?

Let’s think back to our address book example.

2 options: separate chaining, or linear probing.

Hash Tables: Takeaway

Separate chaining is a good strategy for this week's problem set. Once you understand linked lists, you're already 75% of the way done!

One last thing: what are the run times involved?

Tries

Tries are a second approach, based on trees, to solving this pset. Based on the struct below:

```
typedef struct t_node
{
    bool is_word;
    struct t_node* children[27];
} t_node;
```

What are the tradeoffs?

Stacks & Queues

Questions about them?