String Data Structures
Why do we need string data structures?
Why do we need string data structures?
Why do we need string data structures?
Why do we need string data structures?
String Data Structures

Before Computers
Can binary search for a word in time $O(\log n)$. Can interpolation search in average time $O(\log \log n)$. 
Array accesses take time $O(1)$.

Jump to the drawer, then do an $O(d)$ lookup, where $d$ is the number of elements in the drawer.
Miriam-Webster’s physical “Backward Index:”
All English words, written in reverse, in sorted order.

*Why would you want this?*
Find all words ending in "iatriecs."  
Time required: $O(\log n + k)$. 
String Data Structures

*With Computers*
a
about
ad
adage
adagio

b
bar
bard
barn
bed
bet
beta

c
can
cane
cat
d
dikdik
diktat
The double circle means "this is a word."
Because we've remembered that “a” is a word, we'll remove it from our list.
about
ad
adage
adagio

bar
bard
barn
bed
bet
beta

can
cane
cat
dikdik
diktat
about
adage
adagio
bard
barn

can
cane
cat
dikdik
diktat

dad
dad
be
ta
This data structure is called a trie. It’s pronounced “try.” It comes from the word “retrieval.” This is a terrible pun and we’re stuck with it forever.
Using a Trie
Operation 1:
Check if a string is contained in the trie.
**Operation 1:**
Check if a string is contained in the trie.

```
c a n e
```
**Operation 1:**
Check if a string is contained in the trie.
Operation 1:
Check if a string is contained in the trie.
Operation 1:
Check if a string is contained in the trie.

```
c a n e
```
**Operation 1:**
Check if a string is contained in the trie.
**Operation 1:**
Check if a string is contained in the trie.

c a n e
**Operation 1:**
Check if a string is contained in the trie.

```
c a n e
```
**Operation 1:**
Check if a string is contained in the trie.

```
c a n e
```
**Operation 1:**
Check if a string is contained in the trie.

```plaintext
c a n e
```
Operation 1:
Check if a string is contained in the trie.

\[\text{c a n e}\]
Operation 1:
Check if a string is contained in the trie.

c an e
Operation 1:
Check if a string is contained in the trie.
**Operation 1:**
Check if a string is contained in the trie.

Input: "adaa"
**Operation 1:**
Check if a string is contained in the trie.

```
ada
```
Operation 1:
Check if a string is contained in the trie.

`ada`
Operation 1:
Check if a string is contained in the trie.
Operation 1:
Check if a string is contained in the trie.
Operation 1:
Check if a string is contained in the trie.

```
ada
```
Operation 1:
Check if a string is contained in the trie.

```
ada
```
Operation 1: Check if a string is contained in the trie.

adaa
Operation 1:
Check if a string is contained in the trie.

a d a
**Operation 1:**
Check if a string is contained in the trie.
**Operation 1:**
Check if a string is contained in the trie.

```
b a r e
```
Operation 1:
Check if a string is contained in the trie.

bare
**Operation 1:** Check if a string is contained in the trie.

```
bare
```
Operation 1:
Check if a string is contained in the trie.

bare
**Operation 1:**
Check if a string
is contained in
the trie.

```
bar
```
Operation 1:
Check if a string is contained in the trie.

bare
Operation 1:
Check if a string is contained in the trie.

bare
Operation 2:
Add a string into the collection of words.
Operation 2:
Add a string into the collection of words.

[Diagram showing a tree with nodes labeled with letters, leading to a box labeled 'cats']
Operation 2: Add a string into the collection of words.
Operation 2:
Add a string into the collection of words.

```
cats
```
**Operation 2:**
Add a string into the collection of words.
Operation 2: Add a string into the collection of words.
**Operation 2:**
Add a string into the collection of words.

```
cats
```
Operation 2:
Add a string into the collection of words.
**Operation 2:**
Add a string into the collection of words.

```
cats
```
Operation 2:
Add a string into the collection of words.

cats

---

a
b
c
d
e
f
g
h
i
j
k
l
m
n

---

cats
**Operation 2:**
Add a string into the collection of words.

```
cats
```
Operation 2:
Add a string into the collection of words.

cats
**Operation 2:**
Add a string into the collection of words.

```
cats
```
Operation 2:
Add a string into the collection of words.
Operation 2:
Add a string into the collection of words.
Operation 2:
Add a string into the collection of words.

b    e
**Operation 2:**
Add a string into the collection of words.

```
be
```
Operation 2:
Add a string into the collection of words.

b e
Operation 2:
Add a string into the collection of words.
**Operation 2:**
Add a string into the collection of words.

```
be
```
Operation 2:
Add a string into the collection of words.
Operation 2:
Add a string into the collection of words.
Operation 3:
See if a string is a prefix of a string in the trie.
Operation 3:
See if a string is a prefix of a string in the trie.

a b o u
**Operation 3:**
See if a string is a prefix of a string in the trie.

```
abouu
```
Operation 3:
See if a string is a prefix of a string in the trie.

abou
Operation 3:
See if a string is a prefix of a string in the trie.
Operation 3:
See if a string is a prefix of a string in the trie.
Operation 3:
See if a string is a prefix of a string in the trie.

abou
Operation 3:
See if a string is a prefix of a string in the trie.

a b o u
**Operation 3:**
See if a string is a prefix of a string in the trie.

```plaintext
abou
```
Operation 3: See if a string is a prefix of a string in the trie.
Operation 3:
See if a string is a prefix of a string in the trie.
Operation 3:
See if a string is a prefix of a string in the trie.
Operation 3: See if a string is a prefix of a string in the trie.
Operation 3:
See if a string is a prefix of a string in the trie.
**Operation 3:**
See if a string is a prefix of a string in the trie.
Other Operations on Tries

• Find all strings in the trie that start with a given prefix.
  
  • How might you implement this?

• Print all strings in sorted order.
  
  • How might you implement this?

• Find the first string that’s alphabetically before or after another.
  
  • How might you implement this?
A Useful Perspective
A Linked List is Either...

...an empty list, represented by `nullptr`, or...

A single linked list cell that points...

...at another linked list.
A Binary Search Tree Is Either...

an empty tree, represented by `nullptr`, or...

... a single node, whose left subtree is a BST of smaller values ...

... and whose right subtree is a BST of larger values.
A Trie is Either…

an empty trie, represented by

nullptr
A Trie is Either...

an empty trie, represented by `nullptr`, or...

a single node, which might be marked as a word...

... with some number of child tries labeled by letters.
Assignment 6: Think through these design decisions!

**Singly-Linked List**

```c
struct Cell {
    Type value;
    Cell* next;
};
```

**Binary Search Tree**

```c
struct Node {
    Type value;
    Node* left;
    Node* right;
};
```

**Trie**

```c
struct Name? {
    /* ? */
};
```
Time-Out for Announcements!
oSTEM
CAREER OFFICE HOURS

---
Friday, 3/1 4pm-5pm
QSpot (2nd floor Fire Truck House)
Snacks, Headshots, & Advice!
Assignment 6

• Assignment 6 (*MiniBrowser*) goes out today. It’s due one week from Friday at the start of class.
  • Play around with linked lists and tree data structures!
  • Build integral pieces of a larger system!
  • See why all this stuff matters.
• YEAH hours will be held today at 5:00PM in 380-380Y. Slides will be posted.
Back to CS106B!
Twists on Tries
(a sneak peek of beautiful CS concepts!)
Twist: *Finite Automata*
General rule to look up a string:

1. Start at some node.
2. Follow links based on the letters you read.
3. The string is there if you don’t get stuck and land at a double circle.
Breaking the Rules

start → g → r → a → n → d → m → a → a → a

grandma
grandpa
Breaking the Rules

This isn’t a tree, but we can still follow the same rules!

grandma
grandpa
great-grandma
great-great-grandpa
great-great-great-grandpa...

start 

1. t a
2. g r a n d
3. m p a
What sorts of strings does this weird thingy contain?

/*aaaa*/
/***aaaa***/
/*aaa*aaa*/
/*********/
...

start

/*aaaa*/
/***aaaa***/
/*aaa*aaa*/
/*********/
...
Finite Automata

• A finite automaton is a generalization of a trie.

• It’s not necessarily a tree; there can be circular paths, places where branches come together, etc.

• Finite automata power many compilers and pattern-matching tools.

• Want to learn more? Take CS103!
Twist: *Suffix Trees*
Cancer cells often have multiple repeated copies of the same gene.

Given a cancer genome (length ~3,000,000,000 letters) and a gene, count the occurrences of that gene.
A Fundamental Theorem

• The *fundamental theorem of stringology* says that, given two strings $w$ and $x$, that

  $w$ is a substring of $x$

  if and only if

  $w$ is a prefix of a suffix of $x$
A Fundamental Theorem

- The *fundamental theorem of stringology* says that, given two strings \( w \) and \( x \), that

  \( w \) is a substring of \( x \) if and only if \( w \) is a prefix of a suffix of \( x \)
A Fundamental Theorem

- The **fundamental theorem of stringology** says that, given two strings \( w \) and \( x \), that

\[
\text{\( w \) is a substring of \( x \)}
\]

**if and only if**

\[
\text{\( w \) is a prefix of a suffix of \( x \)}
\]
A Fundamental Theorem

• The *fundamental theorem of stringology* says that, given two strings $w$ and $x$, that

  $w$ is a substring of $x$

  if and only if

  $w$ is a prefix of a suffix of $x$
A Fundamental Theorem

• The *fundamental theorem of stringology* says that, given two strings $w$ and $x$, that

$$w \text{ is a substring of } x \text{ if and only if } w \text{ is a prefix of a suffix of } x$$
A Fundamental Theorem

• The *fundamental theorem of stringology* says that, given two strings $w$ and $x$, that

\[ w \text{ is a substring of } x \quad \text{if and only if} \quad w \text{ is a prefix of a suffix of } x \]
A Fundamental Theorem

• The *fundamental theorem of stringology* says that, given two strings $w$ and $x$, that

  \[ w \text{ is a substring of } x \text{ if and only if} \]
  \[ w \text{ is a prefix of a suffix of } x \]

• **Recall:** Tries make it really easy to check if something is a prefix of any number of strings.

• **Idea:** Store all the suffixes of a string in a trie!
Suffix Trees

• With a lot of creativity, it’s possible to compress the trie shown earlier to have only $O(n)$ nodes.
• This is called a suffix tree and is a workhorse of a data structure.
• Want to learn more? Take CS166!
Next Time

• *The Magic of Hash Functions*
  • A beautiful mathematical idea with incredible power.

• *Hash Tables*
  • Surpassing BST performance!