Acknowledgement: The course slides are adapted from the slides prepared by R. Sedgewick and K. Wayne of Princeton University.
Today

- Tries
- R-way tries
Tries

- R-way tries
**Tries.** [from retrieval, but pronounced "try"]

- Store characters in nodes (not keys).
- Each node has $R$ children, one for each possible character.
- Store values in nodes corresponding to last characters in keys.

For now, we do not draw null links.

### Key-Value Table

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>by</td>
<td>4</td>
</tr>
<tr>
<td>sea</td>
<td>6</td>
</tr>
<tr>
<td>sells</td>
<td>1</td>
</tr>
<tr>
<td>she</td>
<td>0</td>
</tr>
<tr>
<td>shells</td>
<td>3</td>
</tr>
<tr>
<td>shore</td>
<td>7</td>
</tr>
<tr>
<td>the</td>
<td>5</td>
</tr>
</tbody>
</table>
Search in a trie

Follow links corresponding to each character in the key.
- **Search hit:** node where search ends has a non-null value.
- **Search miss:** reach a null link or node where search ends has null value.

get("shells")

```
return value associated with last key character (return 3)
```
Search in a trie

Follow links corresponding to each character in the key.

- **Search hit:** node where search ends has a non-null value.
- **Search miss:** reach a null link or node where search ends has null value.

get("she")

search may terminated at an intermediate node (return 0)
Search in a trie

Follow links corresponding to each character in the key.

- **Search hit:** node where search ends has a non-null value.
- **Search miss:** reach a null link or node where search ends has null value.

get("shell")

```
    b
     y 4
    /
   e
    /
   h
    /
   s
     \
    a
     6
    /
   e
    /
   l
    \
   i
    \
   e
    \
   l
     \
   i
    \
   s
     1
    /
   l
    \
   e
     7
    /
   s
     3
```

No value associated with last key character (return null)
Search in a trie

Follow links corresponding to each character in the key.

• Search hit: node where search ends has a non-null value.
• Search miss: reach a null link or node where search ends has null value.

get("shelter")

no link to 't'
(return null)
Follow links corresponding to each character in the key.
- Encounter a null link: create new node.
- Encounter the last character of the key: set value in that node.

\text{put("shore", 7)}
Trie construction demo
Trie construction demo

```
put("she", 0)
```

- **Key:** Sequence of characters from root to value
- **Value:** In node corresponding to last character
Trie construction demo

she
trie

 trie
Trie construction demo

she
trie

trie

0
Trie construction demo

she

put("sells", 1)
Trie construction demo

she
sells
trie
Trie construction demo

she
sells
trie
she
sells

put("sea", 2)
Trie construction demo

she
sells
sea
trie
Trie construction demo

she
sells
sea

put("shells", 3)
Trie construction demo

she
sells
sea
trie
Trie construction demo

she
sells
sea
put("by", 4)
Trie construction demo

she
sells
sea
by
trie
Trie construction demo

she
sells
sea
by
put("the", 5)
she
sells
sea
by
the
trie
Trie construction demo

```plaintext
put("sea", 6)
```

![Trie Diagram]

- **overwrite old value with new value**
Trie construction demo

trie

```
trie
```

```
+---+---+---+
| b | s | t |
+ --+---+---+
    |   |   |
    |   |   |
    +---+---+
        |   |
        |   |
        +---+
            |   |
            |   |
            +---+
                |   |
                |   |
                +---+
                    |   |
                    |   |
                    +---+
                        |   |
                        |   |
                        +---+
                            |   |
                            |   |
                            +---+
                                |   |
                                |   |
                                +---+
                                    |   |
                                    |   |
                                    +---+
                                        |   |
                                        |   |
                                        +---+
                                            |   |
                                            |   |
                                            +---+
                                                |   |
                                                |   |
                                                +---+
                                                    |   |
                                                    |   |
                                                    +---+
                                                        |   |
                                                        |   |
                                                        +---+
```

```
trie construction demo
Trie construction demo

she
sells
sea
by
the
put("shore", 7)
Trie construction demo

she
sells
sea
by
the
shore
trie
Trie representation: implementation

Node. A value, plus references to $R$ nodes.

```
struct Node
{
    int value;
    Node * next[R];
}
```

A child node for each character in Alphabet. No need to search for character, but a pointer reserved for each character in memory.
#define R 256

Node * root;

put(&root, key, val, 0);

void put(Node ** x, char *key, int val, int d) {
    if (*x == null) *x = getNode();
    if (d == strlen(key)) { *x->value = val; return; }
    char c = key[d];
    put((&(x->next[c]), key, val, d+1);
}

:
Node * getNode()
{
    Node * pNode = NULL;
    pNode = (Node *)malloc(sizeof(Node));
    if (pNode){
        for (int i = 0; i < R; i++)
            pNode->next[i] = NULL;
    }
    return pNode;
}
int get(Node * x, char * key, int d)
{
    if (x == null) return -1;  // -1 refers no match
    if (d == strlen(key)) return x->value;
    char c = key[d];
    return get(x->next[c], key, d+1);
}

Trie performance

**Search hit.** Need to examine all $L$ characters for equality.

**Search miss.**
- Could have mismatch on first character.
- Typical case: examine only a few characters (sublinear).

**Space.** $R$ null links at each leaf.
(but sublinear space possible if many short strings share common prefixes)

**Bottom line.** Fast search hit and even faster search miss, but wastes space.
### String symbol table implementations cost summary

<table>
<thead>
<tr>
<th>implementation</th>
<th>search hit</th>
<th>Search miss</th>
<th>insert</th>
<th>space (references)</th>
</tr>
</thead>
<tbody>
<tr>
<td>hashing (separate chaining)</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>R-way trie</td>
<td>L</td>
<td>log (R) (N)</td>
<td>L</td>
<td>RNw</td>
</tr>
</tbody>
</table>

\(N = \) number of entries, \(L = \) key length, 
\(R = \) alphabet size, \(w = \) average key length

### R-way trie.
- Method of choice for small \(R\).
- Too much memory for large \(R\).

### Challenge.
Use less memory, e.g., 65,536-way trie for Unicode!