Admin

- Today's topics
  - Finish Editor Buffer case study
  - Start Map implementation, trees
- Reading
  - Ch 9
  - Ch 13
- Café today after class

Buffer: Vector vs Stack

<table>
<thead>
<tr>
<th></th>
<th>Vector</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>~Buffer()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>moveCursorForward()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>moveCursorBackward()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>moveCursorToStart()</td>
<td>O(1)</td>
<td>O(N)</td>
</tr>
<tr>
<td>moveCursorToEnd()</td>
<td>O(1)</td>
<td>O(N)</td>
</tr>
<tr>
<td>insertCharacter()</td>
<td>O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>deleteCharacter()</td>
<td>O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Space used</td>
<td>1N</td>
<td>2N</td>
</tr>
</tbody>
</table>

Buffer as linked list

- Inspiration: contiguous memory is constraining
  - Connect chars without locality
  - Linked list to the rescue!

- Buffer contains: AB|CDE

```cpp
// for Buffer class
private:
  struct cellT {
    char ch;
    cellT *next;
  };
  cellT *head, *cursor;
```

Cursor design

- Where does cursor point?
  - To cell before or after?
  - Where is cursor if between B & C?
  - After E?
  - Before A?
  - 5 letters, 6 cursor positions…
Use of dummy cell for linked list

- Add "dummy cell" to front of list
  - Simplifies logic
  - Every cell holding actual data has a predecessor
  - Cursor can point to cell before insertion point

Linked list insert/delete

```cpp
void Buffer::insertCharacter(char ch) {
    cellT *cp = new cellT;
    cp->ch = ch;
    cp->next = cursor->next;
    cursor->next = cp;
    cursor = cp;
}
void Buffer::deleteCharacter() {
    if (cursor->next != NULL) {
        cellT *old = cursor->next;
        cursor->next = old->next;
        delete old;
    }
}
```

Linked list cursor movement

```cpp
void Buffer::moveCursorToBegin() {
    cursor = head;
}
void Buffer::moveCursorForward() {
    if (cursor->next != NULL)
        cursor = cursor->next;
}
void Buffer::moveCursorToEnd() {
    while (cursor->next != NULL)
        moveCursorForward();
}
void Buffer::moveCursorBackward() {
    if (cursor != head) {
        cellT *cp = head;
        while (cp->next != cursor)
            cp = cp->next;
        cursor = cp;
    }
}
```

Compare implementations

<table>
<thead>
<tr>
<th></th>
<th>Vector</th>
<th>Stack</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer()</td>
<td>0(1)</td>
<td>0(1)</td>
<td>0(1)</td>
</tr>
<tr>
<td>~Buffer()</td>
<td>0(1)</td>
<td>0(1)</td>
<td>0(N)</td>
</tr>
<tr>
<td>moveCursorForward()</td>
<td>0(1)</td>
<td>0(1)</td>
<td>0(1)</td>
</tr>
<tr>
<td>moveCursorBackward()</td>
<td>0(1)</td>
<td>0(1)</td>
<td>0(N)</td>
</tr>
<tr>
<td>moveCursorToStart()</td>
<td>0(1)</td>
<td>0(N)</td>
<td>0(1)</td>
</tr>
<tr>
<td>moveCursorToEnd()</td>
<td>0(1)</td>
<td>0(N)</td>
<td>0(N)</td>
</tr>
<tr>
<td>insertCharacter()</td>
<td>0(N)</td>
<td>0(1)</td>
<td>0(1)</td>
</tr>
<tr>
<td>deleteCharacter()</td>
<td>0(N)</td>
<td>0(1)</td>
<td>0(1)</td>
</tr>
<tr>
<td>Space used</td>
<td>1N</td>
<td>2N</td>
<td>5N</td>
</tr>
</tbody>
</table>
Fancier linked list

- Add tail pointer to get direct access to last cell
- How to speed up moving backwards?
  - Add prev link, symmetric with next link

Space-time tradeoff

- Doubly-linked list is $O(1)$ on all six operations
  - But, each char uses 1 byte + 8 bytes of pointers => 89% overhead!

- Compromise: chunklist
  - Array and linked list hybrid
  - Shares overhead cost among several chars
  - Chunksize can be tuned as appropriate

- Cost shows up in code complexity
  - (as you will discover on pqueue assignment)
  - Cursor must traverse both within and across chunks
  - Splitting/merging chunks on insert/deletes

Compare implementations

<table>
<thead>
<tr>
<th></th>
<th>Vector</th>
<th>Stack</th>
<th>Single</th>
<th>Double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer()</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>~Buffer()</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>moveCursorForward()</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>moveCursorBackward()</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>moveCursorToStart()</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>moveCursorToEnd()</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>insertCharacter()</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>deleteCharacter()</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Space used

- Vector: $1N$
- Stack: $2N$
- Single: $5N$
- Double: $9N$

Implementing Map

- Map is super-useful
  - Any kind of dictionary, lookup table, index, database, etc.
- Stores key-value pairs
  - Fast access via key
  - Operations to optimize: add, getValue
- How to make work efficiently?
**Simple Map implementation**

- Layer on Vector
  - Provides convenience with low overhead
- Define pair struct
  - Holds key and value together
  - Store Vector<pair>
- Vector sorted or unsorted?
  - If sorted, sorted by what?
- How to implement getValue?
- How to implement add?

**Map as Vector**

<table>
<thead>
<tr>
<th></th>
<th>Unsorted</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map()</td>
<td>0(1)</td>
<td>0(1)</td>
</tr>
<tr>
<td>~Map()</td>
<td>0(1)</td>
<td>0(1)</td>
</tr>
<tr>
<td>add()</td>
<td>0(N)</td>
<td>0(N)</td>
</tr>
<tr>
<td>getValue()</td>
<td>0(N)</td>
<td>0(\log N)</td>
</tr>
<tr>
<td>Overhead per entry</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

**A different strategy**

- Sorting the Vector
  - Provides fast lookup, but still slow to insert (because of shuffling)
- Does a linked list help?
  - Easy to insert, once at a position
  - But hard to find position to insert...
  - Will rearranging pointers help?

Bashful → Doc → Dopey → Grumpy → Happy → Sleepy → Sneezy