Agenda

- Questions on anything
- Quick Huffman Review
- PA2: LZ78/LZW
- Specific questions
Questions on anything?

- Questions?
Quick Huffman Review

Review of what we need to do (remember this is ONE way):

1.) Create minheap class: can have <key,object> pair
2.) Create trie class
3.) Read in input data (M&Ms)
4.) Put input data into minheap “keyed” by frequency
   - This means lowest key at root
5.) Construct trie (keep pulling off the root of minheap to get pairs)
   - Remember: have to put sub-trie back onto minheap sometimes!
6.) When trie is built, we can create the codes for each symbol
7.) Create compressed code table
8.) For encoded data, just use the code table to determine what goes where
**PA2: LZ78/LZW Encoding**

- **Encoding process**
  - Quick review of exam question

```
CAT SCATS CAT
```

<table>
<thead>
<tr>
<th>STEP</th>
<th>PHRASE (input)</th>
<th>C (next char)</th>
<th>OUTPUT</th>
<th>ADD TO DICT (&gt;255)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>256=CA</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>T</td>
<td>A</td>
<td>257=AT</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td></td>
<td>T</td>
<td>258=T</td>
</tr>
<tr>
<td>4</td>
<td>_</td>
<td>S</td>
<td>_</td>
<td>259=S</td>
</tr>
<tr>
<td>5</td>
<td>S</td>
<td>C</td>
<td>S</td>
<td>260=SC</td>
</tr>
<tr>
<td>6</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CA</td>
<td>T</td>
<td>256</td>
<td>261=CAT</td>
</tr>
<tr>
<td>8</td>
<td>T</td>
<td>S</td>
<td>T</td>
<td>262=TS</td>
</tr>
<tr>
<td>9</td>
<td>S</td>
<td></td>
<td>S</td>
<td>263=S</td>
</tr>
<tr>
<td>10</td>
<td>_</td>
<td>C</td>
<td></td>
<td>264=C</td>
</tr>
<tr>
<td>11</td>
<td>C</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>CA</td>
<td>T</td>
<td>261</td>
<td></td>
</tr>
</tbody>
</table>

**Final encoded string:**  CAT_S256TS_261 (Underscore for spaces)
Okay, so we know how to create codes. How do we store them?

Trie

- Putting into trie (Initial size: $256 = 0-255$)
- Start with 9-bit symbols (256-511)
- Increase size if necessary
- When you move from 9-bit to 10-bit symbols, don’t use 511 for any code (char* is NULL). This implies that you are now moving to 10-bit symbols
- Perhaps, an n-ary trie would be useful here.
### PA2: LZ78/LZW Encoding

<table>
<thead>
<tr>
<th>magic cookie</th>
<th>LZ</th>
<th>orig sym size</th>
<th>initial sym size</th>
<th>encoded data</th>
<th>$2^k-1$</th>
<th>encoded data</th>
<th>last bit pos</th>
</tr>
</thead>
</table>

- **Magic Cookie**: 16-bits
- **LZW**: 0010
- **Original Sym Size**: 8-bits (If ASCII, then 8-bits)
- **Initial Sym Size*: It is the symbol size you start with (If your first code string=256 , then initial sym size = 9)
  - Size will increase dynamically
  - Use trie to do the encoding
  - Reserve $2^k-1$ symbol for transition to next symbol size
PA2: LZ78/LZW Decoding

**RUN 256258260**

(NOTE: There is a space between RUN and 2. Also 256 and 260 are underlined to indicate that digits 256, 258 and 260 are individual numbers)

<table>
<thead>
<tr>
<th>STEP</th>
<th>PHRASE (input)</th>
<th>C (next char)</th>
<th>OUTPUT</th>
<th>ADD TO DICT (&gt;255)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R</td>
<td>U</td>
<td>U</td>
<td>256=RU</td>
</tr>
<tr>
<td>3</td>
<td>U</td>
<td>N</td>
<td>N</td>
<td>257=UN</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>_</td>
<td>_</td>
<td>258=N_</td>
</tr>
<tr>
<td>5</td>
<td>256</td>
<td>RU</td>
<td></td>
<td>259= R</td>
</tr>
<tr>
<td>6</td>
<td>RU</td>
<td>258</td>
<td>N_</td>
<td>260=RUN</td>
</tr>
<tr>
<td>7</td>
<td>N_</td>
<td>260</td>
<td>RUN</td>
<td>261=N_R</td>
</tr>
</tbody>
</table>

**Final Output:** RUN_RUN_RUN (Underscore for spaces)
PA2: LZ78/LZW

- Do we need the table from encoding phase?
- RESTRICTION: Must use strings to represent codes
- Nelson’s code is helpful here

\[
\text{CODE:} \\
\text{Read OLD_CODE} \\
\text{output OLD_CODE} \\
\text{CHARACTER = OLD_CODE} \\
\text{WHILE there are still input characters DO} \\
\quad \text{Read NEW_CODE} \\
\quad \text{IF NEW_CODE is not in the translation table THEN} \\
\quad \quad \text{STRING = get translation of OLD_CODE} \\
\quad \quad \text{STRING = STRING+CHARACTER} \\
\quad \text{ELSE} \\
\quad \text{STRING = get translation of NEW_CODE} \\
\text{END of IF} \\
\text{output STRING} \\
\text{CHARACTER = first character in STRING} \\
\text{add OLD_CODE + CHARACTER to the translation table} \\
\text{OLD_CODE = NEW_CODE} \\
\text{END of WHILE}
\]
PA2: LZ78/LZW Encoding

- Visit this website for issues regarding use of stack for getting the decoded strings

  [Link](http://marknelson.us/1989/10/01/lzw-data-compression/)

  - Strings are stored in a reverse order in our string buffer
  - Retrieve characters and store them in our stack buffer and pop them to get the string
Any questions?