Recitation 12

Overhead:
- Quiz
  - mean 90.77
  - median 94
  - mode 94
  - max 112
- check your grades match!

Radix sort

Problem: counting sort performs badly for large input range
Solution: perform a series of counting sorts on small input ranges
go from least significant to most significant.

running time = \((\text{# of rounds})(\text{# inputs + input range})\)

ex: degenerate (counting sort) = 1(n+k)
decimal w/digits = d(n+10)
words w/8 letters = 8(n+26)
b-bit binary #, radix r = \(\frac{b}{r}(n+2^r)\)

# bits to consider each time.

- best choice of r? \(r \leq \lfloor \log(n) \rfloor \rightarrow n+2^r = n+2^{\lfloor \log(n) \rfloor} = 2n\).

Bucket Sort

idea: split input into "buckets" with few elements each. Then sort.
  - like counting sort!
  - but for more than integers.
  - E(# els) = 1
  - need to know insertion distribution!

it's like generalized counting sort.
**GRAPH SEARCHING**

→ configuration graph

vertex : a possible configuration
edges : transition between configurations.

ex: games - vertex = game state
edge = move.
prerequisites graph - vertex = class
edge = prereq.
(hopefully no loops!)

→ graphs vs. trees

tree = a graph with no directed cycles.
- can choose any node as "root" then define an unambiguous parent/child hierarchy.

graph = anything goes.

→ path: Sequence of nodes $x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \ldots \rightarrow x_k$ s.t. $(x_i, x_{i+1}) \in E$

→ reachable
1. $x$ reachable from $x$ (identity)
2. $uRx$ and $uRv$ then $vRx$ (transitive)
3. directed: not reflexive
   undirected: reflexive $xRx$ then $yRx$.

**Thm** $u$ reachable from $x$ then $\exists$ path from $x$ to $u$.

[Who cares? Well if someone swaps stickers on your Rubik's cube, the SDN may not be reachable.
Or if you make a wrong move in a game a winning config may not be reachable.]
**Goal:** traverse all nodes reachable from X.

**Breadth First Search**

*idea:* if it were a tree we would traverse it breadth first.
- i.e. all nodes in level i done before level i+1
- finds shortest paths!
- use a FIFO queue

**Depth First Search**

*idea:* if it were a tree we would traverse it depth first.
- uses LIFO queue (a stack)
- less memory requirements

**Goal 2:** classify nodes into trees and edges into types.
- perform DFS on all nodes.

**Bidirectional path search.**

Given start + end node: use BFS on both simultaneously to find shortest path from start to end.

better than DFS: finds shortest
BFS: quicker.
a. Give the running time to sort in phone numbers using (i) **RADIX** (ii) **Counting**

(i) radix
digits = 10
base = 10
\[ d(n+b) = 10(n+10) \]

b. RADIX sort the following.

```
<table>
<thead>
<tr>
<th>Input</th>
<th>TAP</th>
<th>D13E</th>
<th>T5AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT</td>
<td>D04G</td>
<td>CAT</td>
<td></td>
</tr>
<tr>
<td>DOG</td>
<td>LO5L</td>
<td>D1IE</td>
<td></td>
</tr>
<tr>
<td>LOL</td>
<td>TA5P</td>
<td>D3OG</td>
<td></td>
</tr>
<tr>
<td>DIE</td>
<td>L0L</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

c. What other sorts could you use in RADIX other than Counting sort?

- Any stable sort
  - Insertion
  - Merge
  ...

d. Draw the configuration graph for 2x2 tic-tac-toe i.e. X/O ...

See how even a simple game has a "complex" graph.
e. Traverse the graph using Breadth First Search, starting at B

f. Same DFS

g. Find a path from start to end using BFS on both simultaneously (one step from start side, then one step from end side ...)

Try doing it purely from start to see how many more steps it takes!