Data Structures and Amortization

Minqueue Abstract Data Type (ADT)

- enqueue(x)
- dequeue()
- findMin()

>>> enqueue(42)
>>> enqueue(17)
>>> enqueue(19)
>>> findMin()
17
>>> dequeue()
42

What kind of data structure could implement this ADT efficiently?

Amortization

- Aggregation Method
- Accounting Method
- Potential Method

Sometimes we don’t care about the cost of one operation but instead we care about the TOTAL cost of a sequence of n operations.

Multipop Stacks

- push(x)
- pop()
- multiPop(k) # do k successive pops

multiPop(k) looks “O(k)”!

Incrementing a Binary Counter

Counter starts off with value 0...

... and we plan to perform n successive increments

I can upper bound the running time of n increments by O(n log n)

Prof. I. Lai, Pasadena Institute of Technology
### Multipop Stacks

Accounting Method

- `push(x)`  # stack starts empty
- `pop()`
- `multiPop(k)`  # do k successive pops

### Extendible Arrays

- `insert(x)` is the only operation (for now)

Before the increment...

```
1 0 0 1 1 1
```

And right after...

```
1 1 0 0 0 0
```

### Union-Find ADT

- `makeSet(x)`
- `union(x, y)`
- `findSet(x)`

```
>>> makeSet(Eann)
>>> makeSet(Stacey)
>>> makeSet(Feras)
>>> makeSet(Rodrigo)
>>> findSet(Eann)
Blorg
>>> findSet(Stacey)
Shmork
>>> union(Eann, Stacey)
>>> findSet(Stacey)
Plump
>>> findSet(Eann)
Plump
>>> union(Feras, Rodrigo)
>>> union(Rodrigo, Eann)
>>> findSet(Feras)
Zing!
>>> findSet(Stacey)
Zing!
```
We assume that we have our own pointers to each of these objects.

What happened to those good set names like Flump, Blorg, Shmork, and Zing?

We need to do one more thing...
### The Potential Method

The slide seems a bit empty...

**Amortized Analysis of “Fancy” Linked Lists for Union-Find**

**Disjoint Set Forests**

**The Potential Method**

Recall...

- Let $c_1, c_2, c_3, \ldots, c_n$ denote a sequence of $n$ of these operations. We’d like to show that the total cost $\leq n$.
- Define a “potential function” $\Phi$ where $\Phi_i$ denotes the value of that function immediately after operation $i$.

Define: $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$

The BIG Theorem: If $\Phi_0 = 0$ and $\Phi_n \geq 0$ then

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

And this is the actual cost of our sequence of $n$ operations!

**Proving the “Big” Theorem**

If $\Phi_0 = 0$ and $\Phi_n \geq 0$ then

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

And this is the actual cost of our sequence of $n$ operations!

**Proof:**

**Binary Counters Revisited...**

Let’s try the number of zeros in the counter!
Incrementing and Decrementing!

$$0000000 +$$
$$0000001 +$$
$$0000010 +$$
$$0000011 +$$
$$...$$
$$0010000 -$$
$$0011111$$

What goes wrong with our amortized analysis now?

The Secret: Use Base Fibonacci!

$$1, 1, 2, 3, 5, 8, 13, ...$$
$$13 8 5 3 2 1 1$$
$$0 0 0 1 1 0 1$$

You must be joking!

Now on Pset 2!

Extendible Arrays Revisited

Length $k$

| Insert 6 |
| 42 | 23 | 47 | 5 |
| 42 | 23 | 47 | 5 | 6 |

1. What’s wrong with each of the following potential functions?
   (But one of them almost works!)

   1. $\Phi_i = \text{The length of the current array}$
   2. $\Phi_i = \# \text{filled cells in the current array}$
   3. $\Phi_i = \# \text{unfilled cells in the current array}$
   4. $\Phi_i = 2^\ast \text{filled cells} - \text{length of current array}$

2. One of these can be fixed. Which one? How does the fix work?

Minqueues Revisited...

enqueue(x)
enqueue()
dequeue()
findMin()

>>> enqueue(42)
>>> enqueue(17)
>>> enqueue(19)
>>> findMin()
17
>>> dequeue()
42

Weight Balanced Trees Binary Search Trees

A 0.8-weight balanced tree

For each node $x$, $\text{size}(x) = 1 + \text{size}(\text{left}(x)) + \text{size}(\text{right}(x))$, $\text{size}(\text{left}(x)) \leq \alpha \text{ size}(x)$, $\text{size}(\text{right}(x)) \leq \alpha \text{ size}(x)$

Properties of $\alpha$-balanced trees...

Observation 1: For any $\frac{1}{2} < \alpha < 1$, an $\alpha$-balanced tree has height $O(\log n)$

So each find(x) operation takes $O(\log n)$ time!
Properties of $\alpha$-balanced trees...

Observation 2: If the $\alpha$-balance breaks on an insertion, we can fix it!

I'm worried that it might take a long time to do that fix!

Analysis Goes Here...

$$i = \frac{\text{size(left(v))}}{\text{size(right(v))}}$$

<table>
<thead>
<tr>
<th>Size</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>100</td>
</tr>
<tr>
<td>23</td>
<td>20</td>
</tr>
<tr>
<td>75</td>
<td>79</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>35</td>
<td>15</td>
</tr>
</tbody>
</table>