Jeff Erickson: Algorithms

Chapter 4' Amortized analysis

Example 1. Incrementing a binary counter

```
INCREMENT(B[0..∞]):
    i ← 0
    while B[i] = 1
        B[i] ← 0
        i ← i+1
    B[i] ← 1.
```

Lemma 1. INCREMENT must terminate. (Each number write as $\Sigma 2$

The base case $n=0$ is trivial, For any $n>0$, the inductive hypothesis implies there ~~are~~ is a set of distinct powers of 2 whose sum is $n-1$.

If we add 2^0 to the list, obtain a multiset of powers of 2 whose sum is n, which might not contain two copies of 2^0. Then as long as there are 2 copies of 2^0, remove them both and add a 2^1.

Generally, if we have 2 copies of 2^i, we remove that two and give a 2^{i+1}, and the sum is unchanged, for $2^{i+1} = 2^i + 2^i$. Each iteration decreases the multiset by 1, so the process eventually terminates.

But how quickly does it terminates?

Attempt #1. Increment: $O(1)$

Carry: $O(\log n)$.

$\downarrow$ Executing n times

$O(n\lg n)$.

How often do it carries? 。

In fact, it's only ($\Theta(n)$).

- INCREMENT don't flip $\Theta(\lg n)$ bits every time it was called.

B[0]	each time
B[1]	every other iter
B[2]	every 4th iter
⋮	
B[i]	every 2^i iter

↳ Total number of bit flips of entire seq is

$$\sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor < \sum_{i=0}^{\infty} \frac{n}{2^i} = 2n.$$

Here we look at some methods.

1、The summation method. Let $T(n)$ be the worst-time running time for a seq of n operations. The amortized time for each op. is $T(n)/n$.

2、The taxation ~~problem~~ method.

Look back to the example.

- Suppose it cost a dollar to toggle a bit.

  But at each time flip:
  - receive 2$ each time it adds one
  - pays $1 each time it toggles a bit.

> Certain steps in the algorithm charge you taxes, so that the total cost incurred by the algorithm is never more than the total tax you pay. The amourtized cost of an operation is overall tax charge during that operation

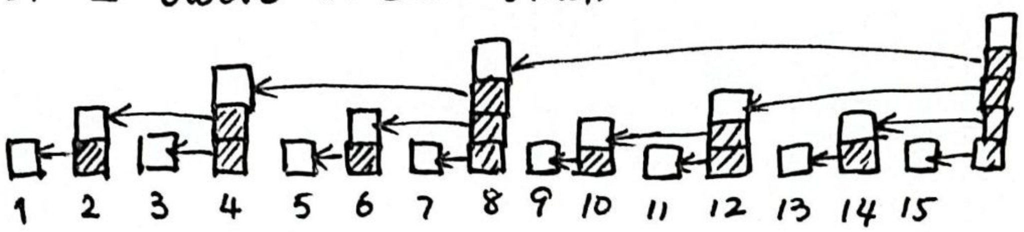
- for every bit → charge us a tax at every op. B[i] charge a tax of $\frac{1}{2^i}$ dollars for each increment. Every time B[i] needs to be flipped, collect $1.

> Certain portion of the D.S. charge you taxes at each op., so that total cost of maintaining the d.s. is never more than total taxes you pay. The amortized cost of an operation is overall tax you pay during that operation

▷ Different 'tax schedule' result in different bounds.

3. Charging. Charge the cost of some steps of algo. to earlier steps, or to steps in some earlier operation. The amortized cost of algorithm is its actual running time, minus Σcharges to past operations, + total charge from future op.

- If we moved each shaded block onto the white block directly to its left, there would at most 2 blocks in each stack



□ = toggle from 0 to 1

▨ = toggle from 1 to 0.

} h = running time of ith increment.

4. Potential Method.

- repr prepaid work as <u>potential</u> s.t. can be used later operations.
- is own by the data structure.

Let D_i denote our DS after i ops have been performed, and let ϕ_i denote its potential.

c_i denote actual cost of ith operation ($D_{i-1} \to D_i$)

The amortized cost of ith op. denoted as a_i, is

$$a_i = c_i + \phi_i - \phi_{i-1}.$$

So the total amourtized cost of n ops. is the actual total cost plus total increase in potential:

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} (c_i + \phi_i - \phi_{i-1}) = \sum_{i=1}^{n} c_i + \phi_n - \phi_0$$

A potential func. is <u>valid</u> if $\phi_i - \phi_0 \geq 0.\ \forall i$.

that is actual cost of any seq is less than total amourtized cost.

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} a_i - \phi_n \leq \sum_{i=1}^{n} a_i.$$