Amortized Analysis

Alexandra Nilles

Example one: mortgage

- Draw example on board
- Could pay lots of interest up front, less over time
- Or, could have constant payments for lifetime of mortgage

Example two: stack as array

- Define array A with variable `top` points to first free element (top of stack)
- Define `push(x)` as `A[top] = x; top++`
- Define `pop(x)` as `top--; x = A[top]`
- However, if array is full, need to allocate new array and move data
- Very expensive single operation for `pop`, but only sometimes. How to characterize performance of data structure?

Cost model: assign cost of push as 1, pop as 1, and resizing array is number of elements moved.

Algorithm 1: Resize to increase array size by 1

- Start with array of size 1
- perform n pushes
- total cost $= 1 + 2 + 3 + 4 + \ldots + n = n(n+1)/2$
- cost per operation (amortized!) is $(n+1)/2$

Algorithm 2: Double array when resized

- Again, start with array of size 1
- n pushes
- total cost for resizing $= 1 + 2 + 4 + 8 + \ldots + 2^i$
- sum is at most $2n - 1$, plus n for the push operations.
- Total cost $3n - 1$, amortized cost per operation is < 3.

Example 3: Binary counter

```
# B is array of bits
def Increment(B):
  i = 0
  while B[i] = 1:
    B[i] = 0
    i = i+1
  B[i] = 1
```

How long to terminate?

- If first k bits are all ones, it will take $\Theta(k)$ time.
- Binary representation of integer n is $lgn + 1$ bits long.
- So to call Increment n times, starting at zero, we estimate $O(nlgn)$ for total running time to count up to n.

We can actually do better!

Method one: Summation

- Observe that we don't flip all log(n) bits every time
- Least significant bit B[0] does flip every time
- B[1] flips every other time
- B[2] flips every 4th iteration
- B[i] flips every 2^ith iteration

Pattern yields the sum:

$$\sum_{i=0}^{lgn} \frac{n}{2^i} < \sum_{i=0}^{\infty} \frac{n}{2^i} = 2n$$

Thus, on average, each call to Increment runs in constant time.

Note this is a different sense of "on average" than we will consider with randomized algorithms.

Method 2: Accountant's method

Imagine if instead of paying for each bit flip, the "Increment Revenue Service" charges us two dollars whenever we set a bit from zero to 1. When we flip the same bit back to zero, the IRS pays us back a dollar.

So amortized cost of increment is just 2, since only one bit is flipped each time.

Can also think of this method as charging the cost of later steps in the algorithm to earlier steps.

Method 2: Physicist's method

Prepaid work is *potential* that can be used to power later operations.

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

Binary counter: define potential ϕ_i to be the number of bits with value 1

At step i, c_i is number of bits changed from 0 to 1 + bits from 1 to zero. Change in potential is number of bits changed from 0 to 1 - bits changed from 1 to zero.

Amortized cost:

$$a_i = c_i + \phi_i - \phi_{i-1} = 2(bitsfrom0to1)$$