Amortized Analysis and Hashing

Ted Pyne

Table of Contents

Amortized Analysis

- Normal: Require our data structure to be fast on every operation
- Amortized: Require our data structure to be fast "in total"

Track runtime using **potential function**

Cost of operation = real work + change in potential function

If potential function has positive delta, "saving" for future operations

NB: Potential function is **not real** - not tracked by algorithm, only used for analysis

Faster Heaps via Amortized Analysis

1. Before: bin-heap with O(log n) insert, O(n) makeheap, O(log n) deletemin.
2. Goal: heap with O(1) insert and O(log n) *amortized* deletemin.

Idea: Build up “credit” from prior lazy inserts to pay for our extra deletemin work.

Data Structure

Maintain integer array A, heap-of-heaps H.

Elements of H are of form [k,h] where:

1. h is a heap of integers
2. k is smallest element of h

Invariant: Potential function Φ is always equal to $|A|$.

Insert Operation

insert(k):

A.append(k)

Increment p

Deletemin Operation

deletemin():

h = makeheap(A)

insert(H, [peek(h), h])

h’ = deletemin(H)

elt = deletemin(h’)

insert(H, [peek(h), h])

Set **p** to 0

return elt

Runtime Analysis

insert(): appends to list, increments **p**

Total cost $1 + \Delta\mathbf{p} = 1 + 1 = O(1)$

deletemin(): turns A into heap, does 6 other heap operations, sets **p** to 0

Total cost $|A| + 6 \log n + \Delta\mathbf{p} = |A| - |A| + 6 \log n = O(\log n)$

Hashing

Hash function: map from [u] to [m] where u >> m.

We have a **family H** of hash functions and choose h from H at random.

Requirement: for all x,y in [u]:

$$\Pr_h[h(x) = h(y)] = 1 / m.$$

Families of Hash Functions

H is **all functions** on [u] to [m], so we pick a truly random function **h.**

Is this a valid family of hash functions?

Question

Does there exist a hash family with **one** element?

Second Attempt

Truly random functions are expensive to generate.

A cheaper family: for all i in [u], define the function:

$$h_i(x) = (x + i) \bmod m$$

Is this a valid hash family?

Exercise 1

Consider the sequence of operations:

insert(2), insert(3), insert(1), deletemin(), insert(5), insert(0), deletemin(), insert(6)

What does the data structure look like after each operation?

Exercise 2

Suppose we want to implement a dynamically sized array. Let the current number of elements be k, and let m be the size of the current array.

As additional appendLast() and removeLast() operations occur:

1. If k = m, allocate a new array of size 2m and move all elements over.
2. If k = m / 4, allocate a new array of size m / 2 and move all elements over.

Assuming moving a single element takes O(1) time, show this data structure has O(1) *amortized* runtime for appendLast() and removeLast() operations.

Exercise 3

Our “linear shift” hash family failed. Let’s try something else.

Assume $u = \{0,1\}^s$ and $m = \{0,1\}^k$ with + and * defined bitwise. Then let **M** be the set of all matrices with entries in {0,1} with k rows and s columns. For all A in **M**, define:

$$h_A(x) = Ax$$

Show **$\{h_A$: A in M$\}$** is a hash family.