Algorithms Example Sheet 6: Problems

Amortised analysis

Recommended reading:

- Jeff Errickson's notes
- CLRS Chapter 17
- Okasaki's Chapter 5 on amortised analysis

Exercise 6.P.1 [k-bit counter] Consider a k-bit binary counter. This supports a single operation, `inc()`, which adds 1. When it overflows i.e. when the bits are all 1 and `inc()` is called, the bits all reset to 0. The bits are stored in an array, $A[0]$ for the least significant bit, $A[k-1]$ for the most significant.

(a) Give pseudocode for `inc()`.

(b) Explain why the worst-case cost of `inc()` is $\mathcal{O}(k)$.

(c) Starting with the counter at 0, what is the aggregate cost of m calls to `inc()`? [**Hint:** *How many times can each bit get flipped?*]

(d) Let $\Phi(A)$ be the number of 1s in A. Use this potential function to calculate the amortized cost of `inc()`.

(e) In a binomial heap with N items, show that the amortized cost of `push()` is $\mathcal{O}(1)$.

Note: *We have seen this before, but it might be good practice to attempt this again, now that you have seen amortised analysis in more detail.*

[Exercise 2 in Lecturer's handout]

Exercise 6.P.2 Consider a stack that, in addition to `push()` and `pop()`, supports `flush()` which repeatedly pops items until the stack is empty.

(a) Using the potential function

$$\Phi = \text{ number of items in the stack,}$$

show that the amortized cost of each of these operations is $\mathcal{O}(1)$.

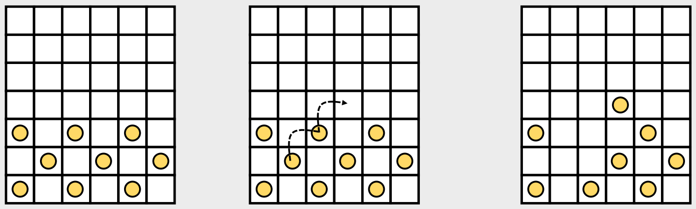
[Exercise 3 in Lecturer's handout]

(b) What is the worst-case time complexity for an operation? Could this matter in practice?

(c) How would you implement the operations so that the worst-case time complexity for each operation is $\mathcal{O}(1)$?

Exercise 6.P.3 [Functional Deque] Attempt **[2018P1Q2 (b)]**.

Exercise 6.P.4 In the game of solitaire chequers, the goal is to move one piece to the top row, via moves of the type shown. A move consists of a diagonal jump by one piece over another; the latter piece is then removed.



Define a potential function

$$\Phi(\text{board state}) = \sum_{\text{pieces } p} \phi(y\text{-coordinate of } p),$$

where you should define ϕ in such a way that any valid move leaves Φ unchanged. Use this potential function to prove that it is impossible to win the game, starting from the board configuration on the left.

[Source: Dr. Sauerwald]

Dynamic Array

Exercise 6.P.5 [Dynamic array with removals] Consider a stack, implemented using a dynamically-sized array. The rightmost item in the array represents the top of the stack, where we pop from and where we push new items. Suppose that the array's capacity is doubled when it becomes full, and halved when it becomes less than 25% full; and that the cost of these resizing operations is $\mathcal{O}(n)$ where n is the number of items to copy across into the new array. Using the potential method, show that the `push` and `pop` operations both have $\mathcal{O}(1)$ amortized cost. What would go wrong if we halved capacity as soon as the array became less than 50% full, rather than 25% full?

[Exercise 8 in Lecturer's handout]

Amortised analysis in search trees

Exercise 6.P.6 I have an algorithm that uses a 2-3-4 tree. I've noticed that in a typical run there are several intervals in the key space into which items get inserted, but in which the algorithm doesn't end up searching. I don't want to waste time balancing a tree with these items, but I don't want to discard them because I don't know in advance what the searches will be. I'd like to adapt the 2-3-4 tree so that insertions are lazy, $\mathcal{O}(1)$, and so that the relevant parts of the tree are tidied up on search. Suggest an appropriate design. Using the potential method, explain why your design is no worse than the original 2-3-4 tree, in the big-$\mathcal{O}$ sense.

[Exercise 9 in Lecturer's handout]

Exercise 6.P.7 [Restructuring RBTs] (optional) Attempt Problem 7.2 from this problem sheet.

Disjoint sets

Exercise 6.P.8 [Extensions to Disjoint Sets] Sketch out how you might implement the lazy forest disjoint set, so as to efficiently support:
(a) "Given an item, print out all the other items in the same set".

[Exercise 12 in Lecturer's handout]

(b) "Given an item, print the smallest index in the same set".
Explain carefully how `get_set_with` and `merge` are implemented.

Exercise 6.P.9 Consider an undirected graph with n vertices, where the edges can be coloured blue or white, and which starts with no edges. The graph can be modified using these operations:

- `insert_white_edge(u,v)` inserts a white edge between vertices u and v
- `colour_edges_of(v)` colours blue all the white edges that touch v
- `colour_edge(u,v)` colours the edge $u \leftrightarrow v$ blue
- `is_blue(u,v)` returns True if and only if the edge $u \leftrightarrow v$ is blue

Give an efficient algorithm that supports these operations, and analyse its amortized cost. Extend your algorithm to also support the following operation, and analyse its amortized cost:

- `are_blue_connected(u,v)` returns `True` if and only if u and v are connected by a blue path Extend your algorithm to also support the following operation, and analyse its amortized cost.
- `remove_blue_from_component(v)` deletes all blue edges between pairs of nodes in the blue-connected component containing v

*[**Note:** It's easy to gloss over difficulties, so be sure to be explicit about all operations. If you change your data structure to answer a later part, make sure your earlier answers are still complete. This is the sort of question you might be asked in a Google interview; the interviewer will be looking for you to take ideas that you have been taught and to apply them to novel situations.]*

[Source: This question is from Alstrup and Rauhe, via Inge Li Gortz]

Exercise 6.P.10 [Disjoint set problems] (optional) Think how you would solve some of the following problems:

(a) **[CSES Road Construction]**

(b) **[Usaco Closing the farm]**

(c) **[Usaco MooTube]**