Example sheet 6

Advanced data structures

Algorithms—DJW—2020/2021

Questions labelled $\circ$ are warmup questions. Questions labelled $*$ involve more thinking and you are not expected to tackle them all.

Question 1°. Consider a binary heap. Find a sequence of N items such that inserting them all takes $\Omega(N \log N)$ elementary operations.

Question 2. 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.
(i) Give pseudocode for `inc()`.
(ii) Explain why the worst-case cost of `inc()` is $O(k)$.
(iii) 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?]*

Question 3°. Consider a stack that, in addition to `push()` and `pop()`, supports `flush()` which repeatedly pops items until the stack is empty. Using the potential function

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

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

Question 4. (i) Consider the N-bit binary counter described in question 2. Let $\Phi(A)$ be the number of 1s in A. Use this potential function to calculate the amortized cost of `inc()`.
(ii) In a binomial heap with N items, show that the amortized cost of `push()` is $O(1)$.

Question 5. Consider the dynamic array from section 7.3 in lecture notes, but suppose that when the array needs to be expanded we multiply the capacity by a factor $k > 1$ rather than doubling. What is wrong with the following argument?

"Define the potential to be $\Phi = 2n - \ell$, where n is the number of items and ℓ is the capacity, with the special case $\Phi = 0$ when $n = 0$. In the case where we don't need to expand the array, true cost is $O(1)$ and $\Delta\Phi = 2$ so amortized cost is $O(1)$. In the case where we do need to expand the array, say from $\ell = n$ to $\ell = kn$, true cost is $O(n)$ and $\Delta\Phi = 2\Delta n - \Delta\ell = 2 - (k-1)n$, so the amortized cost is $O\big(n + (1-k)n\big) = O\big((2-k)n\big)$. If $k \geq 2$ the amortized cost is $O(1)$, and if $k < 2$ the amortized cost is $O(n)$."

Question 6. Consider the dynamic array from section 7.3 in lecture notes. What is wrong with the following argument?

"Suppose the array starts with $n = 2^m$ items, and is at capacity. A single `append` *will require copying these n items. The fundamental rule of amortized analysis is that the true cost of any sequence of operations is $\leq$ the sum of their amortized costs. Since a sequence consisting of a single* `append` *can cost $\Omega(n)$, it follows that the amortized cost of* `append` *is $\Omega(n)$."*

Question 7. Consider the dynamic array from section 7.3 in lecture notes, but suppose that when the array needs to be expanded we add a constant $k \geq 1$ to capacity rather than multiplying.
(i) Show that the aggregate cost of appending n items is $\Omega(n^2)$.
(ii) An engineer friend tells you excitedly that they have found a cunning potential function that proves that the amortized cost of appending an item is $O(1)$. Explain why your friend must be mistaken.
(iii) Your friend gives the following proof. Where is the flaw?
"We will add a term to the potential function, measuring how far each item is from the tail end of the array. Specifically, for a dynamic array with n items indexed by $0 \leq i < n$, and capacity $\ell \geq n$, let the potential be

$$\Phi = 2n - \frac{1}{k}\sum_{i=0}^{n-1}(\ell - i).$$

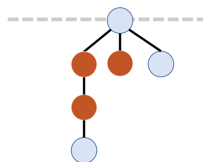
For example, with $k = 2$, a dynamic array holding $n = 2$ items with capacity $\ell = 4$ has potential $\Phi = 2\times2-(4+3)/2 = 1/2$. When we add an item without expanding capacity, $\Delta\Phi \leq 2$ so appending is $O(1)$. When we expand capacity, the cost of copying n items is $O(n)$, and Φ decreases by $nk/k = n$ because ℓ has increased by k; hence expanding capacity is $O(1)$."

Question 8*. 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 $O(n)$ 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 $O(1)$ amortized cost. What would go wrong if we halved capacity as soon as the array became less than 50% full, rather than 25% full?

Question 9*. 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, $O(1)$, 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-O sense.

Question 10°. Give a sequence of operations that would result in a Fibonacci heap of this shape. (The three darker nodes are losers.) What is the shortest sequence of operations you can find?



Question 11°. Prove the result from Section 7.8 of the handout, namely, that in a Fibonacci heap with n items the maximum degree of any node is $O(\log n)$. Must it be a root node that has maximum degree?

Question 12°. Sketch out how you might implement the lazy forest DisjointSet, so as to efficiently support "Given an item, print out all the other items in the same set". Explain carefully how `get_set_with` and `merge` are implemented.

Question 13. In the flat forest implementation of DisjointSet, using the weighted union heuristic, prove the following:

(i) After an item has had its 'parent' pointer updated k times, it belongs to a set of size $\geq 2^k$.

(ii) If the DisjointSet has N items, each item has had its 'parent' pointer updated $O(\log N)$ times.

(iii) Starting with an empty DisjointSet, and assuming the number of items added is $\leq N$, show that any sequence of m operations takes $O(m + N \log N)$ time in aggregate.

Question 14*. 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.]

This question is from Alstrup and Rauhe, via Inge Li Gortz.