Example sheet 7/8

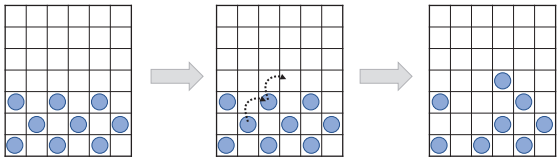
Amortized analysis. Heaps. Disjoint sets.
Algorithms—DJW*–2017/2018

Questions labelled ∘ are warmup questions. Questions labelled ∗ involve more thinking and you are not expected to attempt all of them. The questions are arranged in the order that the course is taught; consult your supervisor to find out which questions to answer for which supervision.

Question 1∘. Consider a stack that, in addition to `push()` and `pop()`, supports `flush()` which repeatedly pops items until the stack is empty. Explain why the amortized cost of each of these operations is $O(1)$.

Question 2. Consider a dynamically-sized array that supports appending an element on the right, and deleting the rightmost element. Suppose that the array's capacity is expanded when the array becomes full, and that the array's capacity is reduced when the array becomes less than 25% full. Using the potential method, show that the append and delete operations both have $O(1)$ amortized cost.

Question 3 (TS)*. 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

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

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.

Question 4. 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 n calls to `inc()`? *[Hint. Your answer may be asymptotic in k, but it must not be asymptotic in n.]*
(iv) Let $\Phi(A)$ be the number of 1s in A. Use this potential function to calculate the amortized cost of `inc()`.

Question 5. In a binomial heap with n items, show that the amortized cost of `push()` is $O(1)$. *[Hint: use your answer to Question 4(iv)].*

*Questions labelled FS are from Dr Stajano. Questions labelled TS are from Dr Sauerwald. Question 14 is from Alstrup and Rauhe, via Inge Li Gortz.

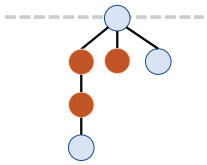
Question 6*. Consider a binomial heap on which we perform a sequence of operations comprising n_1 `push()`, n_2 `decreasekey()`, and n_3 `popmin()`. Show that the worst-case total cost is $\Omega(n_1 + n_2 \log n_1 + n_3 \log n_1)$.

Question 7. Consider a Fibonacci heap on which we have only performed `push()` and `popmin()`. Show that, after the cleanup part of of `popmin`, the heap has the form of a binomial heap.

Question 8°. Prove the result from Section 7.4 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 9 (FS49)°. In the Fibonacci heap, suppose that `decreasekey()` only cuts out (if necessary) the node whose key has been decreased, i.e. it doesn't use the `loser` flag and it doesn't recursively inspect the node's parents. Show that it would be possible for a node with n descendants to have $\Omega(n)$ children.

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. In a Fibonacci heap, can a node x aquire a child node y, then lose it, then gain it again? *[In the complexity analysis of the Fibonacci heap, we carefully wrote "children ... in the order of when they last became children of x." This question explains why we needed the word 'last'.]*

Question 12. 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, any sequence of m operations of which n are `add_singleton()` takes $O(m + n \log n)$ time in aggregate.

Question 13. 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".

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–v blue
- `is_blue(u,v)` returns True if and only if the edge u–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