ADS 2021: Week 11 Exercises

Exercises for week 11 of Algorithms and Data Structures at ITU. The exercises are from *Algorithms, 4th Edition* by Robert Sedgewick and Kevin Wayne unless otherwise specified. This week also includes exercises from *Algorithms and Data Structures* by Kurt Mehlhorn and Peter Sanders, and from Thore Husfeldt's notes on the course website. Color-coding of difficulty level and alterations to the exercises (if any) are made by the teachers of the ADS course at ITU.

From Thore's notes - Green

a) Bob is a runner. He has been running 4 km every day of the week, except in the week-ends. What is the worst case number of kilometers he runs per day? He started this new workout schedule on a Saturday. What is the amortised number of kilometers he runs per day?

b) Ran is also a runner. He rolls a die every morning and runs as many kilometers. (i) What is the expected number of kilometers Ran runs per day, assuming he has a perfect die? (Or what about enchanted dice that always come up 1? Or cursed dice that always come up 6?) (ii) What is the worst-case number of kilometers Ran runs per day? (iii) For any sequence of days starting on a Saturday, what is the worst-case amortised number of kilometers Ran runs per day? (Note that this answer must hold *even if his dice are cursed.*)

c) My phone company charges 10 DKK for 1 minute of voice call. This is exorbitant, but I never use my phone for making a phone call anyway. According to the contract, I have to pay at least 100 DKK per month for voice calls whether I use them or not, but the contract automatically 'rolls over' the unused minutes to the next month. I have to call my mum for Christmas for a 2-hour call. Describe the expense in terms of 'money I spend on voice calls each month' in the worst case and in the amortised sense.

d) A multiride ticket in the Danish amusement part Tivoli costs 200 DKK and is valid for 10 rides. What is the worst case cost for a single ride? What is the amortised cost for a ride? (*Careful!*)

Old exam set 110530: 2 - Green See question 2 in the exam set bads-110530.pdf on learnit.

Mehlhorn-Sanders book: 3.9 - Green Your manager asks you to change the initialization of α to $\alpha = 2$. He argues that it is wasteful to shrink an array only when three-fourths of it are unused. He proposes to shrink it when $n \leq w/2$, where w is the length of the array. Convince him that this is a bad idea by giving a sequence of m pushBack and popBack operations that would need time $\Theta(m^2)$ if his proposal was implemented.

Mehlhorn-Sanders book: 3.13 - Yellow Implement an operation popBack(k) that removes the last k elements in amortized constant time independent of k.

MultiPush - Yellow Implement an operation multiPush(k, e) that extends the stack with k copies of element e. Analyse the running time.

Old exam set 120531: 2 - Yellow See question 2 in the exam set bads-120531.pdf on learnit. Note: g) and h) are also covered in the quiz.

Set Union - Red In the Set Union problem we have n elements, that each are initially in n singleton sets, and we want to support the following operations:

- Union(A,B): Merge the two sets A and B into one new set $C = A \cup B$, destroying the old sets.
- SameSet(x, y): Return true, if x and y are in the same set, and false otherwise.

We can implement it the following way. Initially, give each set a color. When merging two sets, recolor the smallest one with the color of the larger one (break ties arbitrarily). To answer SameSet queries, check if the two elements have the same color.

a) Analyze the worst case cost of the two operations.

b) Show that the amortized cost is O(log n) for Union and O(1) for SameSet. That is, show that any sequence of m unions and s SameSet operations takes time O(m log n + s). Hint: Give a bound on the number of times an element can be recolored.

Mehlhorn-Sanders book: 3.14 - Red Suppose, for a real-time application, you need an unbounded array data structure with a worst-case constant execution time for push and pop operations. Design such a data structure. Hint: store the elements in up to two arrays. Start moving elements to a larger array well before a small array is completely exhausted. Assume array allocation is constant time (which it is not in Python and Java).