University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
S. Faour, M. Fuchs, A. Malyusz

Algorithm Theory
Exercise Sheet 5

Due: Monday, 22th of November 2024, 10:00 am

Assumption: You may assume that calculations with real numbers can be performed with arbitrary precision in constant time.

Exercise 1: DP *(10 Points)*

Given a tree $G = (V, E)$, an integer $d \in \mathbb{N}$, and a weight function $w : V \to \mathbb{R}_+$, your task is to select a subset $S \subset V$ of the nodes that maximizes the sum $\sum_{u \in S} w(u)$ while ensuring that the subgraph induced by S, denoted $G[S]$, has a maximum degree of at most d.

(a) Solve the problem if $d = 1$ and $w \equiv 1$ in $O(n \log n)$. *(2 Points)*

(b) Now solve the general problem in $O(d \cdot n \log n)$ *(5 Points)*

(c) Now solve the general problem in $O(n \log n)$ *(3 Points)*

Exercise 2: Making binary search dynamic *(10 Points)*

Binary search of a sorted array takes logarithmic search time, but the time to insert a new element is linear in the size of the array. You can improve the time for insertion by keeping several sorted arrays. Specifically, suppose that you wish to support Search and Insert on a set of n elements. Let $k = \lceil \lg(n + 1) \rceil$, and let the binary representation of n be $\langle n_{k-1}, n_{k-2}, \dots, n_0 \rangle$. Maintain k sorted arrays $A_0, A_1, \dots, A_{k-1}$, where for $i = 0, 1, \dots, k - 1$, the length of array A_i is 2^i. Each array is either full or empty, depending on whether $n_i = 1$ or $n_i = 0$, respectively. The total number of elements held in all k arrays is therefore $\sum_{i=0}^{k-1} n_i \cdot 2^i = n$. Although each individual array is sorted, elements in different arrays bear no particular relationship to each other.

(a) Describe how to perform the Search operation for this data structure. Analyze its worst-case running time.

(b) Describe how to perform the Insert operation. Analyze its worst-case and amortized running times, assuming that the only operations are Insert and Search.

(c) Describe how to implement Delete. Analyze its worst-case and amortized running times, assuming that there can be Delete, Insert, and Search operations.

Exercise 3: Problem for the exercise session *(0 Points)*

Your plan to implement a `Stack` with the classical operations `push`, `pop` and `peek`. As underlying data structure you use a dynamic array that will grow its size whenever 'many' elements are stored and on the other hand also shrinks its size when only a view elements remain in the array. In the following let n_i be the number of elements stored in the array and let s_i be the size of the array after the i-th operation.

- Before you **push** a new element x to the array, you check if $n_{i-1} + 1 < 80\% \cdot s_{i-1}$. If this is the case then you simply add x. We say for simplicity, that this can be done in 1 time unit. If on the other hand $n_{i-1} + 1 \geq 80\% \cdot s_{i-1}$, you set up a new (empty) array of size $s_i := 2s_{i-1}$ and copy all elements (and x) into the new one. We assume this can be done in s_{i-1} time units[1].
- To **pop** an element from the array, you first check if $n_{i-1} - 1 > 20\% \cdot s_{i-1}$. If this is the case then pop x within 1 time unit. If the table size is small, say $s_{i-1} \leq 8$, you also just pop x. But, if $n_{i-1} - 1 \leq 20\% \cdot s_{i-1}$ and $s_{i-1} > 8$, create a new (empty) array of size $s_i := s_{i-1}/2$ and copy all values except x into this new array. By assumption, this step takes s_i time units.
- The **peek** operation returns the last inserted element in 1 time unit. Note that state of the array does not change, i.e., $n_{i-1} = n_i$ and $s_{i-1} = s_i$.

Initially, the array is of size $s_0 = 8$. Assume that this initial step can also be done in 1 time unit. Note that by this initial size and the definition of the pop method we have $s_i \geq 8$ for all $i \geq 0$. Also note that after every operation that resized the array at least one element can be pushed or popped until a further resize is required.

a) Let i be a push operation that resized the array. Show that the following holds.

$$0.4 \cdot s_i \leq n_i < 0.55 \cdot s_i$$

Further, show that if i is a pop operation that resized the array, the following holds.

$$0.25 \cdot s_i < n_i \leq 0.4 \cdot s_i$$

b) Use the `Accounting Method` from the lecture to show that the **amortized running times** of push, pop and peek are $O(1)$, i.e., state by how much you additionally charge these three operation and show that the costs you spare on 'the bank' are enough to pay for the costly operations.
Hint: Use the previous subtask, even if you didn't manage to show them.

c) Show the same statement as in the previous task, but use the `Potential Function Method` this time, i.e., find a potential function $\phi(n_i, s_i)$ and show that this function is sufficient to achieve constant amortized time for the supported operations.
Hint: There is not just one but infinitely many potential functions that work here. However, you may want to use a function of the form $c_0 \cdot |n_i - c_1 \cdot s_i|$ for some properly chosen constants $c_0 > 0$ and $c_1 > 0$.

[1] For a simpler calculation we use normalized time units, such that all the operations that would take $O(1)$ time will take at most 1 time unit and operations that would take $O(s_{i-1})$ time will take at most s_i time units.