Assignment 5 – Selected Model Answers

EXERCISE 1. Augment a Fibonacci heap H to support two new operations without changing the amortised running time of any other Fibonacci-heap operations:

(i) The operation `change_Key`(H, x, k) changes the key of node x to the value k. Give an efficient implementation of this operation and analyse the amortized time complexity of your implementation for the cases in which k is greater than, less than, or equal to $x.key$.

(ii) Give an efficient implementation of `prune`(H, r), which deletes $r \leq H.n$ nodes from H. You may choose any r nodes to delete. Analyse the amortised time complexity of your implementation.

SOLUTION.

(i) The operation `change_Key`(H, x, k) can be simply realised by `decrease`(H, x, k), if $x.key > k$ holds. In case $x.key = k$ the operation becomes `skip` doing nothing. In these two cases the amortised complexity is in $O(1)$.

In the remaining case $x.key < k$ we have to execute first `delete`(H, x) followed by `insert`(H, k) using the handle x for the newly inserted node. The amortised complexity of `delete`(H, x) is in $O(\log n)$ and the amortised complexity of `insert`(H, k) is in $O(1)$. In combination we obtain an amortised complexity of `change_Key`(H, x, k) in $O(\log n + 1) = O(\log n)$.

(ii) The implementation of `prune`(H, r) simply removes all the selected subtrees, and adds all subtrees of the corresponding roots to the root list. This is followed by a `consolidate` operation.

The root list before the `consolidate` operation has the length $t(H) + rD(n)$. As the number of iterations in the *consolidate* operation is bounded by the length of the root list, we obtain a complexity bound for `prune`(H, r) in $O(t(H) + rD(n))$.

Furthermore, degrees are bound by $D(n)$, so after consolidation there will be at most $D(n) + 1$ trees left. Thus, the potential before `prune`(H, r) is $\Phi(H) = t(H) + 2m(H)$ and afterwards it will be $\Phi(H') \leq D(n) + 1 + 2m(H)$, as the number of marked nodes can only decrease. Hence the amortised complexity for `prune`(H, r) is bounded by

$$c \cdot (t(H) + rD(n)) + c(\Phi(H') - \Phi(H)) \leq c(r + 1)D(n) + c\,.$$

As we have $r \leq t(H) \leq D(n)$, we get $c(r+1)D(n) + c \in O(D(n)^2) = O((\log n)^2)$.

EXERCISE 2. Instead of finite sets with elements in a set T consider finite multisets with elements in T. You can use associative arrays to represent finite multisets with explicit storage of multiplicities or you can extend the idea of hashing with chaining to multisets.

(i) Implement the data structure for finite multisets with explicit storage of multiplicities including the basic operations for insertion, deletion and retrieval as well as multiset union, intersection and difference.

(ii) Implement the alternative data structure for finite multisets with hashing with chaining and the same operations.

EXERCISE 3. The waste of space in hashing with chaining is due to empty lists in some entries of the representing array A. For a random hash function determine the expected number of empty entries in the hash table as a function of the number m of possible hash values and the size n of the represented set.

Hint. As in the analysis of the average case complexity of hashing with chaining use Bernoulli-distributed random variables $X_0, \ldots, X_{m-1}$ with $X_i = 1$ iff $A[i]$ is empty.

SOLUTION. Let S be the set represented by A with the hash function $h : S \to \{0, \ldots, m-1\}$ and $|S| = n$.

For $0 \leq i \leq m-1$ we use a Bernoulli-distributed random variable X_i with

$$X_i = \begin{cases} 1 & \text{if } A[i] = [] \\ 0 & \text{else} \end{cases} \qquad \text{and let} \quad X = \sum_{i=0}^{m-1} X_i$$

be the random variable for the number of empty entries in the array A. We have $A[i] = []$ iff $h(e) \neq i$ holds for all $e \in S$. We have $P(h(e) = i) = 1/m$, as h is a random hash function. Then $P(h(e) \neq i) = 1 - \frac{1}{m} = \frac{m-1}{m}$. Furthermore, the values of $h(e)$ are independent of each other, so the probability that $h(e) \neq i$ holds for all $e \in S$ is $\left(\frac{m-1}{m}\right)^n$ independent from the value i.

With this we compute the expectation

$$E(X) = \sum_{i=0}^{m-1} E(X_i) = \sum_{i=0}^{m-1} P(A[i] = []) = m \left(\frac{m-1}{m}\right)^n .$$

EXERCISE 4. There are many approaches that aim to improve the performance of hashing. One possibility is to reorganise the hash table during insertions. Implement the following two methods and compare the performance with the performance of the methods on the HASHSET class defined in the lectures:

Last-come-first-served hashing. Here an element to be inserted is always placed in the position given by its hash value. If this position is occupied, the current "resident" is moved to another position using the search procedure defined for HASHSET.

Robin Hood hashing. Here it is checked for two elements competing for the same position, how far away they are stored from the ideal position given by their hash value. Then the chaining search is applied to the element closer to this ideal position.