Homework Assignment 6

Due Date: April 2, 2022, 23:59

EXERCISE 1. 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.

total points: 10

EXERCISE 2.

(i) Assume you have a large file consisting of triples (transaction, price, customer ID). Explain how to compute the total payment due for each customer. Your algorithm should run in linear time.

(ii) Show how to scan a hash table for hashing with chaining and for hashing with linear probing, i.e. return all values stored in the table. What is the running time of your solutions?

total points: 8

EXERCISE 3. A simple variant of *linear hashing* exploits a fixed maximum length l of the lists stored in a table entry. For this fix a number m and a hash function h with values in $\{0, \ldots, m-1\}$. The hash table will have a size $M \geq m$. Then define inductively hash functions h_i with $h_0 = h$ and $h_{j+1}(e) = h_j(e) + m \cdot 2^j$, so h_j takes $2^j \cdot m$ different hash values. The hash value of an element e is $h_j(e)$, where j is maximal with $h_j(e) \leq M - 1$.

(i) Show that whenever an element e is given, it suffices to compute at most two hash values.

(ii) Define a *rehash* operation that comes into action, when the maximum list length l will be exceeded by insertion of a new element e. In this case use h_{j+1} to distribute the elements of the list into two sublists and increase M accordingly. If necessary, split other lists as well according to the new value M.

(iii) Show that if it is permitted to use also h_{j-1} instead of h_j to determine the hash value of an element e, then rehashing can be done in a lazy way by splitting only one list.

total points: 12

EXERCISE 4.

(i) An alternative possibility is to reorganise the hash table during insertions, known as *Robin Hood hashing* is to check 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. Implement this method and compare the performance with the performance of the methods on the HASHSET class.

(ii) A *map* M is a partial function that is defined on a set K, i.e. we can write M as a set of *key-value pairs* (k, v) with unique keys $k \in K$. *Memoisation* is a programming technique useful for cases, where the same call of a function is used multiple times. In order to avoid costly recomputation it uses a map to store results from previous calls.

(a) Define a class HASHMAP in analogy to HASHSET to represent maps. As keys uniquely determine key-value pairs let the hash values only depend on the keys of such pairs.

(b) Use your class HASHMAP for memoisation on an example of your choice—you could use a naive algorithm for computing Fibonacci numbers, the towers of Hanoi problem, or just the location of elements on a given position in a doubly-linked list.

total points: 20