University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
M. Fuchs, G. Schmid

Algorithms and Datastructures
Winter Term 2024
Sample Solution Exercise Sheet 4

Due: Wednesday, May 15th, 2pm

Exercise 1: Hashing with Open Addressing *(5 Points)*

Let $\mathcal{H}$ be a hash table of size $m = 13$ and let $h_1, h_2, h_3 : \mathbb{N}_0 \mapsto \{0, ..., m-1\}$ be hash functions defined as follows[1]:

- $h_1(k) := \overline{k} \mod m$
- $h_2(k) := 3 \cdot k \mod m$
- $h_3(k) := k + 1 \mod m$

Add the keys $23, 12, 75, 945, 30, 99, 345$ (in that order) into the initaly empty hash table $\mathcal{H}$. Solve conflicts as follows:

a) Linear Probing using hash function h_1. *(2 Points)*

b) Use Double Hashing using hash functions h_2 and h_3. *(3 Points)*

Write down every intermediate step!

Sample Solution

a)

$$h(x, i) := h_1(x) + i \mod m$$

0	1	2	3	4	5	6	7	8	9	10	11	12
-	-	-	-	-	23	-	-	-	-	-	-	-

0	1	2	3	4	5	6	7	8	9	10	11	12
-	-	-	12	-	23	-	-	-	-	-	-	-

0	1	2	3	4	5	6	7	8	9	10	11	12
-	-	-	12	-	23	-	-	-	-	-	-	75

0	1	2	3	4	5	6	7	8	9	10	11	12
-	-	-	12	-	23	945	-	-	-	-	-	75

0	1	2	3	4	5	6	7	8	9	10	11	12
-	-	-	12	30	23	945	-	-	-	-	-	75

0	1	2	3	4	5	6	7	8	9	10	11	12
-	-	-	12	30	23	945	99	-	-	-	-	75

0	1	2	3	4	5	6	7	8	9	10	11	12
345	-	-	12	30	23	945	99	-	-	-	-	75

[1]We define the digit sum of k by $\overline{k}$.

b)

$$h(x,i) := h_2(x) + i \cdot h_3(x) \mod m$$

0	1	2	3	4	5	6	7	8	9	10	11	12
-	-	-	-	23	-	-	-	-	-	-	-	-

0	1	2	3	4	5	6	7	8	9	10	11	12
-	-	-	-	23	-	-	-	-	-	12	-	-

0	1	2	3	4	5	6	7	8	9	10	11	12
-	-	75	-	23	-	-	-	-	-	12	-	-

0	1	2	3	4	5	6	7	8	9	10	11	12
-	945	75	-	23	-	-	-	-	-	12	-	-

0	1	2	3	4	5	6	7	8	9	10	11	12
-	945	75	-	23	-	-	-	-	-	12	-	30

0	1	2	3	4	5	6	7	8	9	10	11	12
-	945	75	-	23	-	-	-	-	-	12	99	30

0	1	2	3	4	5	6	7	8	9	10	11	12
-	945	75	-	23	-	-	-	345	-	12	99	30

Exercise 2: Hashing with Chaining *(5 Points)*

Given a Hash Table of size m and an arbitrary hash function $h : S \mapsto \{0, ..., m-1\}$. Let S be a set of at least $y \cdot m$ elements, so $|S| \geq y \cdot m$.

a) Show that S has a subset Y of at least y elements (hence $|Y| \geq y$) such that $h(x_1) = h(x_2)$ for all $x_1, x_2 \in Y$. *(4 Points)*

b) What does the result of *a)* tells us about the Worst-Case runtime of *"find"* in a hash table with Chaining (if the table is filled with all the elements of S before we call *"find"*)? *(1 Point)*

Sample Solution

a) We proof the statement by contradiction. We hence assume that there is no set Y of size $|Y| \geq y$ with the property that for all $x_1, x_2 \in Y$, the given hashfunction h maps $h(x_1) = h(x_2)$. Further, we define the sets $S_0, S_1, ..., S_{m-1}$ as follows:

$$S_i := \{x \in S \mid h(x) = i\}$$

Since the hash function h maps every element to exactly one value between 0 and $m-1$, we also have that every element $x \in S$ is contained in exactly one of the S_i sets. Thus,

$$|S_0| + |S_1| + \ldots + |S_{m-1}| = |S|. \tag{1}$$

However, by our assumption we have that for all these sets $|S_i| < y$. What implies that

$$|S_0| + |S_1| + \ldots + |S_{m-1}| < y + y + \ldots + y = m \cdot y. \tag{2}$$

Connecting (1) and (2) we get $|S| < m \cdot y$. Contradiction.

b) To search an element in that data structure, we need - independent of the hash function - time $\Omega(y)$.

Exercise 3: Application of Hashtables *(10 Points)*

Consider the following algorithm:

Algorithm 1 `algorithm` ▷ Input: Array A of length n with integer entries

```
1: for $i = 1$ to $n - 1$ do
2:     for $j = 0$ to $i - 1$ do
3:         for $k = 0$ to $n - 1$ do
4:             if $|A[i] - A[j]| = A[k]$ then
5:                 return true
6: return false
```

(a) Describe what `algorithm` computes and analyse its asymptotical runtime. *(3 Points)*
Hint: The difference $|A[i] - A[j]|$ may become arbitrarily large.

(b) Describe a different algorithm $\mathcal{B}$ for this problem (i.e., $\mathcal{B}(A) =$ `algorithm`(A) for each input A) which uses hashing and takes time $\mathcal{O}(n^2)$ (with proof). *(3 Points)*

Hint: You may assume that inserting and finding keys in a hash table needs $\mathcal{O}(1)$ if $\alpha = \mathcal{O}(1)$ (α is the load of the table).

(c) Describe another algorithm for this problem without using hashing which takes time $\mathcal{O}(n^2 \log n)$ (with proof). *(4 Points)*

Sample Solution

(a) The algorithm checks if there are two entries in the array whose distance (absolute value of the difference) equals some entry in the array. If so, it returns "true", otherwise "false". In case it returns "false", the algorithm runs completely through all three loops. It considers

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \mathcal{O}(n^2)$$

many pairs (i, j) and for each of this pair it checks n times the if-condition in line 4. Therefore, the runtime is $\mathcal{O}(n^3)$.

(b) We compute an array B of size $\mathcal{O}(n^2)$ which contains an entry $|A[i] - A[j]|$ for each pair (i, j) with $0 \le j < i < n$. This takes time $\mathcal{O}(n^2)$. Afterwards we allocate a hash table of size $\mathcal{O}(n^2)$ (therefore $\alpha = O(1)$), choose a suitable hash function h and hash the values from B into the table H (this takes $\mathcal{O}(n^2)$ under ther assumption that one insert operation takes $\mathcal{O}(1)$). Finally, we test for each entry in A if it is contained in H, taking n times $\mathcal{O}(1)$. Hence the overall runtime is $\mathcal{O}(n^2)$.

(c) We sort A, taking $\mathcal{O}(n \log n)$. Afterwards we compute array B as in part (b), taking $\mathcal{O}(n^2)$. Now we test for each entry in B if it is in A using binary search. This takes n^2 times $\mathcal{O}(\log n)$. The overall runtime is dominated by the last step and equals $\mathcal{O}(n^2 \log n)$.