University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn

Algorithms and Data Structures
Winter Term 2020/2021
Sample Solution Exercise Sheet 4

Exercise 1: Hashing - Collision Resolution with Open Addressing

(a) Let $h(s,j) := h_1(s) - 2j \bmod m$ and let $h_1(x) = x + 2 \bmod m$. Insert the keys 51, 13, 21, 30, 23, 72 into the hash table of size $m = 7$ using linear probing for collision resolution (the table should show the final state).

0	1	2	3	4	5	6

(b) Let $h(s,j) := h_1(s) + j \cdot h_2(s) \bmod m$ and let $h_1(x) = x \bmod m$ and $h_2(x) = 1 + \big(x \bmod (m-1)\big)$. Insert the keys 28, 59, 47, 13, 39, 69, 12 into the hash table of size $m = 11$ using the double hashing probing technique for collision resolution. The hash table below should show the final state.

0	1	2	3	4	5	6	7	8	9	10

Sample Solution

(a)

30	13	21	72	51	23	
0	1	2	3	4	5	6

(b)

	69	13	47	59	39	28	12			
0	1	2	3	4	5	6	7	8	9	10

Exercise 2: Application of Hashtables

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.

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

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 $\mathcal{O}(n^2 \log n)$.

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 $0 \leq j < i < n$. This takes time $\mathcal{O}(n^2)$. Afterwards we allocate a hash table of size $\mathcal{O}(n^2)$, 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 $\mathcal{O}(n^2 \log n)$.