Beihang University
School of General Engineering
Beijing, Dec. 6th, 2019
Sebastian Wandelt, Junyu Zhao, Pengfei Yi

Computer Science and Programming
Lab Class 12

Task 1. ***Birthday Paradox*** *(10 minutes)*

Implement a function that verifies the Birthday Paradox, which simply indicates that the probability of having two people with same birthday is higher than expected. Implement the function by following:

1. Implement some code that generates n values in the range [1,365] by using the randint function in random library.
2. Execute the code from Subtask 1 for different values of n for 100 times each and calculate the average probability of two people having the same birthday. Show at which n the probability surpasses 50%.

Task 2. ***Hashing – quadratic probing*** *(15 minutes)*

Implement the hashing class with linear probing in Python. There should be a constructor, insert function, search function and delete function. We only want to handle integer elements for now.

(1) __init__(self,size)
(2) insert(self,value)
(3) search(self,value)
(4) delete(self,value)

Task 3. ***Hashing – double hashing*** *(15 minutes)*

Implement the double hashing class in Python. There should be a constructor, insert function, search function and delete function. For double hashing, you need to provide two hash functions.

(1) __init__(self,size)
(2) insert(self,value)
(3) search(self,value)
(4) delete(self,value)

Hint: Modify the solution of the previous task, in order to avoid rewriting the whole code.

Task 4. ***Hashing – Incremental growth*** *(30 minutes)*

Implement the hashing class which works for a variable number of elements, according to what was presented in the lecture: You start with a small prime number $p = 7$ and then double the number of hash slots (=buckets), whenever the fill density reaches a value of α larger than 50%. You only need to implement the function *insert*, not *search*/*delete*. Once you are finished, convince yourself that the running time is linear in the number of elements. Try to recall the reason/proof. For testing your code, add up to 2000 elements to the hashing class, by calling function *insert*.

Task 5. ***Max heap*** *(10 minutes)*

Implement a function which checks whether or not a given tree instance is a max heap. The function should verify the following rules:

1. The maximum value of a max heap is always stored at the root of the tree.
2. Each subtree is still a max heap.

Task 6. ***Hashing – No numbers*** *(remaining time)*

Implement the hashing class which works for strings, instead of integers. Update your code for Task 3 by defining an appropriate hash-function for strings.