Updates

- Lab exam on 18th, time will be announced later
- You will be using the lab systems
  - Vim will be present get used to it for basic stuff
  - We will try to provide VScode but we cannot guarantee it
- You will not be required to code in ADTs anymore (including A3), but you should understand the concepts and should be able to implement ADTs if asked.

Hashing

Key:Value pair

A key is something which represents some data, which is called a value

- {1, Himalaya}, {Dog, Brown}, {7, [Seven, seven]}, {5, 5} are all key value pairs where the first element is the key and the second value

All keys have to be unique, we cannot have the same key pointing to different values
Same values might have different keys, there is no such restriction.

- {A, [Age = 18]}, {B, [Age = 18]}

Hash function

A hash function takes a key and returns a hash value
A good hash function has two properties:

- It is fast to compute
- Should minimize the number of keys with the same hash value

Hash table

A hash table uses the hash value to efficiently store and retrieve data.
A hash table converts the hash value to a hash code which is used to identify the bucket in which a key should go.

Collisions

If two keys have the same hash value, then storing those keys in a hash table causes a collision.

Collision resolution

How do we resolve collisions?
You can

- Ignore the new key
- Replace the old key
- Or follow one of the methods mentioned below

Separate Chaining

- Every bucket is a linked list, and if two keys have the same hash, we just add those two keys to the linked list.

Open Addressing

Search for alternative locations in the hash table. Various ways to search for these alternative locations

Linear probing

https://en.wikipedia.org/wiki/Linear_probing

Quadratic probing

https://en.wikipedia.org/wiki/Quadratic_probing

h(k, i) with $h(k) = 5, m = 100, c_1 = 1, c_2 = 2$
First try : h(k, 0) = 5
Second try : h(k, 1) = 5 + 1 + 2 = 8
Third try; h(k, 2) = 5 + 2 + 8 = 15
Fourth try: h(k, 3) = 5 + 3 + 18 = 26

Assignment 2 solutions

Question 1

- Replacing two numbers with their LCM can affects the previous and the next numbers of the two numbers.
- If we iterate from the start of the array to the end, we only have to deal with the LCM affecting the previous number since the next number will be dealt with later
- So all we need is a way to store the numbers we have already iterated over with the ability to access the last number quickly
- FILO/LIFO structure ? So we use a stack for storing all the numbers we have iterated over.

Start with a stack with 1. Iterate over the array. If the number in the array and the top of the stack have GCD 1, then push the number inside the stack and continue to the next number. Otherwise, pop the number out of the stack and replace the number in the array with their LCM, then repeat the check again.
When this terminates, the final array will be stored in reverse order in the stack. Keep on popping the numbers till you reach the size 1 (since we had started with a dummy 1 in the stack), and reverse all the popped numbers.

Question 2

- When we move a window 1 step forward, the answer only changes if either one of these conditions is met
  - The new number is smaller than our current smallest number
  - The smallest number is leaving the window
- We somehow have to store the number of the window in increasing order
- But do we need to store all of them?
- Consider you have a sorted non-decreasing window and you are adding a number. You now know that all numbers greater than this new number cannot be the answer for this and future windows till the new number is present. So we can safely remove them.
- Now consider the other case when the smallest number is leaving the window. If we have maintained the window in non-decreasing order, then the smallest number is the first number of the window. What will be the second smallest number, aka, our next smallest number for the window?
  - Second number in the sorted window
- Assume we are storing this non-decreasing window in some X. So all we have to do is while adding a number to the window, remove

all the numbers greater than the number we are adding to the X, and if the smallest number is leaving the window, then remove the first number from X.

- Here we are adding numbers to the back and removing numbers from the back and front. Which data structure supports these 3 operations? A Deque

When adding the number to the back of the deque (push_back()), remove all the numbers greater than the number we are adding from the deque (pop_back()).
When moving the window to the next window, if the leaving number is our current smallest number, then remove it from the deque (pop_front()).
Answer of the current window is the number at front of the deque (front())
No need to explicitly maintain the window while iterating over the array, just use the new number and the leaving number to make these decisions.

Question 3

- For a pair of buildings i, j, If it is possible to go from i to j, then the following has to be true
  - All buildings between i and j have their heights smaller than min(h(i), h(j))
- What does this mean?
  - If h(i) < h(j), then j is the first building with height greater than h(i) after i, and we cannot go from i to any k > j
  - If h(j) < h(i), then i is the first building with height greater than h(j) before j, and we cannot go from any k < i to j
- Technical terms
  - For case 1: h(j) is the next greatest element of h(i)
  - For case 2: h(i) is the previous greatest element of h(j)
- What if we have a decreasing order number list and we add a new element to it. Can we calculate the next greatest element for all the numbers if it exists?
  - The element we are adding will be the next greatest element for all numbers less than the element

    - What if we remove these elements whose NGE we found? We will end up with a list in decreasing order again.
    - What if we add another element to this new list? Can we say the same things we said earlier?
- Now what if we iterate over the array and keep on adding and using this number list? Will we get the NGE of all elements (if it exists)?
- What are the operations we are doing?
    - Adding to the back and removing from the back.
    - Stack

To implement it, just find the NGE and PGE for all elements. Store the indices of the buildings and use the heights for NGE and PGE. Implementation will be similar to what you did in Q2

- You won't be removing anything from the front.
- When popping, store the NGE/PGE index of the popped index as the index of the element we have just added

Question 4

- Read online about `Maximum size square sub-matrix with all 1s`, if you have doubts, ask me later
