Final Examination—Session II

Algorithms & Data Structures
NTNU IDATA 2302

May 2024

1 Basic Knowledge

Question 1.1 (1 pt.)**.** *In the worst-case, Algorithm X can search through a collection of n items in $O(\log_2 n)$ seconds. How big is a collection it can search through in 10 seconds? What well-known search algorithm has a similar runtime complexity?*

Solution. Given that Algorithm X runs in $O(\log_2 n)$, as does the *binary search*. To know how many item can be searched in 10 seconds, we can write the following equality:

$$\begin{aligned} \log_2 n &= 10 \\ n &= 2^{10} \\ n &= 1024 \end{aligned}$$

□

Grading.

- 0.5 pt. for the correct number of items, and appropriate explanation
- 0.5 pt. for naming the binary search

Question 1.2 (1 pt)**.** *Consider the binary search tree (BST) shown below. What is the runtime complexity of searching in such binary search tree? Why? Describe a insertion scenario that would lead to such a configuration. Explain your reasoning.*

Solution. The binary tree shown above is not well balanced: It has degenerated into a linked list, and searching a for a value will run in average in $O(n)$ (linear time). This happens when insertions come in ascending (or descending). For instance, the tree shown above could result from the insertion 25, 16, 13, and 7 (descending order). □

Grading.

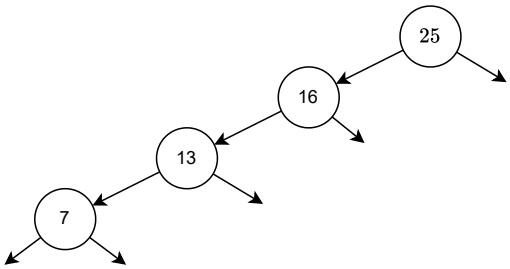

Figure 1: A sample binary search tree (BST)

- 0.5 pt for the linear runtime and the justification (e.g., linked list, degenerated, not balanced).
- 0.5 pt for the insertion scenario.

Question 1.3 (1 pt.)**.** *Consider a dynamic array, implemented using a static array with a capacity of 5. The dynamic array double in size whenever it gets full. Provided the length of the dynamic array is currently 5, what happens when we insert another item? Justify your answer.*

Solution. The underlying array is full because the length of the dynamic array matches the capacity of the underlying static array. In this situation, we need to proceed as follows:

1. Allocate another static array, twice longer. This new array has a capacity of 10.
2. Copy all the five existing items from the old array to the new array
3. We can now insert the new item

□

Grading.

- 0.5 pt for the doubling of the underlying array
- 0.5 pt for copying all the items from the old array to the new one.

Question 1.4 (1 pt.)**.** *Why is recursion considered less efficient than iteration (i.e. loops)? Justify your answer.*

Solution. Recursion requires the use of the call stack and therefore consumes more memory. Each recursive call pushes a new activation record on top of this stack. This record contains the given argument, as well as other information to ensure jumping back and forth between the caller and the callee procedures. Allocating these records also consumes extra runtime. □

Grading.

- 0.5 pt for mentioning the additional memory consumption ;
- 0.5 pt for mentioning the call stack

Question 1.5. *Consider the directed graph shown below. How many paths connect Node D to Node G? Explain your reasoning?*

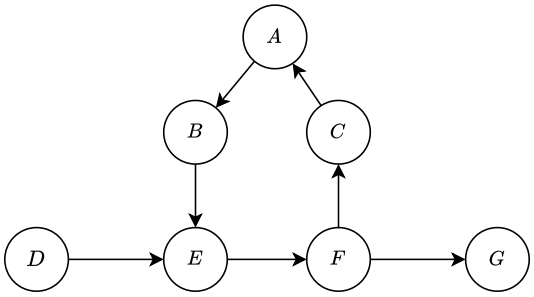

Figure 2: A directed graph

Solution. There is infinitely many paths that connect Node D to node G due to the cycle E-F-C-A-B-E. The shortest path from D to G is $D \to E \to F \to G$. However, the cycle can be traversed any number of times before reaching G. Below are some of the shorter paths:

- $D \to E \to F \to G$ (direct path)
- $D \to E \to F \to C \to A \to B \to E \to F \to G$ (1 cycle)
- $D \to E \to F \to C \to A \to B \to E \to F \to C \to A \to B \to E \to F \to$ G (2 cycles)
- ...

□

Grading.

- 1 pt. for the infinite number of cycles.

2 Nested-loop Join

Let's look at the *nested-loop join* algorithm: A simple yet naive solution to join two tables in databases. Consider the two tables shown below: The Customers table contains the identifier, the name and the country of the customers, whereas the *Order* table contains the date and amount (price) of their purchases.

The figure below shows an example of data stored in these two tables.

Say we need to find all the order placed by customers from Norway. We could use the *nested-loop join* as follows:

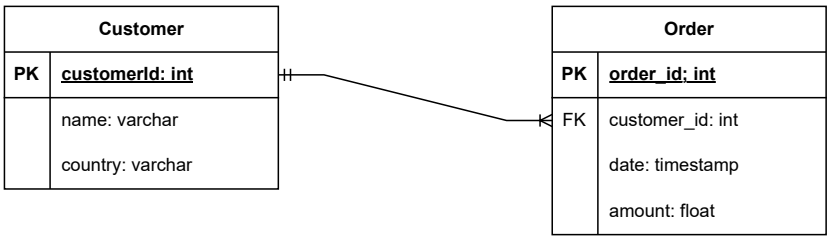

Figure 3: An Entity-Relationship diagram to illustrate the customer-order relationship

customer_id	name	country
1	Taylor	Norway
2	Olivia	Sweden
3	Jordan	Denmark
4	Emma	Norway

order_id	customer_id	date	amount
101	1	2023-11-02	399.99
102	2	2023-12-10	2540.00
103	1	2023-12-11	765.50
104	2	2023-12-11	1234.00
105	2	2023-12-23	9128.00
106	4	2024-01-05	499.99

Figure 4: Example of data stored in the Customer and Order tables

```
List<Order> selectNorwegianOrders() {
    var selectedOrders = new ArrayList<Order>();
    for(Customer customer : customers) {
        if(customer.getCountry().equals("Norway")) {
            for(Order order : orders) {
                if(order.getCustomerId() == customer.getCustomerId()) {
                    selectedOrders.add(order)
                }
            }
        }
    }
    return selectedOrders
}
```

Question 2.1 (1 pt)**.** *Consider the sample tables shown above. With these very tables, how many comparisons will be performed by the* nested-loop join*? Note that comparisons only include the* `.equals()` *and the* `==` *operators. Explain you calculation.*

Solution. The nested-loop algorithm has two comparisons: One on line 4 (`.equals()`) and one on line 6 (the `==` operator). With the given tables, the outer loop executes for all customers, the comparison on Line 4 will be evaluated 4 times. The comparison is true only for two customers (i.e., Emma and Taylor). For these two customer, the inner loop will executes and evaluates the second comparison for all 6 orders. That gives us a total of $4 + (2 \times 6) = 16$ comparisons. □

Grading.

- 0.5 pt for a correct answer, 16.
- 0.5 pt for a convincing explanation.

Question 2.2 (1 pt)**.** *Consider the nested-loop join algorithm below. Describe the best and worst-case scenarios for its runtime performance. Under which circumstances do they occur? Explain your reasoning.*

Solution. For this algorithm, the best case scenario occurs when there are no customer from Norway, because the inner loop, which iterates over the orders, never gets executed. By contrast, the worst-case scenario occurs when all customers are from Norway, because then the inner loop gets executed for every single customer. □

Grading.

- 0.5 pt for a correct and justified worst case scenario
- 0.5 pt for a correct and justified best case scenario.

Question 2.3 (2 pt)**.** *In a nested-loop join operation, given n customers and m orders, calculate the number of comparisons needed in the worst case to find all orders from a specific country c. Express your answer as a function of n and m, and provide a detailed step-by-step explanation of your calculations.*

Solution. In the worst-case, all customers are from the country c, and the comparison on line 4 executes for each customers. Now, for each customer, the inner loop iterates through all the orders and evaluates the second comparison (==) for each. That gives us:

$$\begin{aligned} t(n, m) &= n * (1 + (m * 1)) \\ &= n \times m \end{aligned}$$

□

Grading.

- 1 pt for a correct $t(m, n) = m \times n$
- 1 pt for a correct justification relating to the worst-case.

Question 2.4 (1 pt.)**.** *Is it correct to say that, in the worst case, the nested-loop join runs in $\Theta(n \times m)$ (Big-Theta)? Justify your reasoning.*

Solution. We saw in the previous question that, in the worst case, the number of comparisons is $t(n, m) = n \times n$, is both in the $O(n \times m)$ and in $\Omega(n \times m)$. As it is both big-O and big-Omega, it is also in $\Theta(n \times m)$ by the definition of big-Theta operator. □

Grading.

- 0.5 pt for a correct answer, Yes.
- 0.5 pt for a convincing explanation.

3 Book Cipher

A book cipher is a type of encryption where a book, so called the key, is used to encode messages. In this method, each letter of the message is replaced by a number representing the position or 'rank' of a word in the book that starts with that letter. To decode the message, the recipient refers to the same book, using the numbers to identify the starting letters of the corresponding words. Both sender and recipient must use the same text as the key to ensure that the message can be accurately decoded.

In the example below, we have encoded the message "HELLO PEOPLE" with the using the first paragraph of the book "Harry Potter and the Philosopher's Stone" as a key. We have first numbered each word of the paragraph, such that

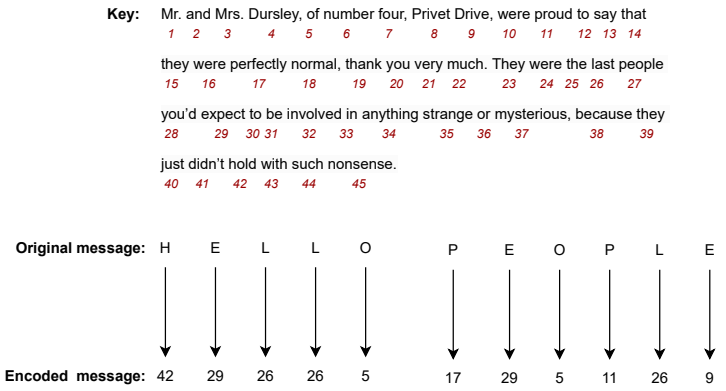

Figure 5: Encoding "HELLO PEOPLE" using "Harry Potter"

- "H" becomes 42 because the 42th word is "hold", which starts with an "H".
- "E" becomes 29 because the 29th word is "expect", which starts with an "E"
- "L" becomes 26 because the 26th word is "last", which starts with a "L"
- "O" becomes 5 because the 5th word is "of", which starts with a "O"
- etc.

Note that several numbers can represent the same letter. In our Harry Potter example, "P" can be encoded by either of 8 (Privet), 11 (proud), 17 (perfectly), or 27 (people)

In this exercise, we assume we are given the following `Key` interface, with operation to encode and decode individual letters.

```
class Key {
    char decodeNumber(int number);

    int encodeLetter(char letter);
}
```

Question 3.1 (1 pt.)**.** *Propose an algorithm to decode a given encoded text using a given key text. The text comes as a array of numbers, and the key as an object instance of the Key class. Assume the following:*

- *The Key class is already implemented: You do not need to implement its operations, but you should use them in your algorithm.*
- *The given text does not contains punctuation, space, or any other symbols besides the 26 letters of the English alphabet.*

```
char[] decode(int[] encodedText, Key key) {
    // Provide some pseudo-code here.
}
```

Solution. To decode the given text, one has to go through each letter, in order, and decode them using the given key. Here is an example of how that could look like in pseudo-Java:

```
char[] decode(int[] encodedText, Key key) {
    var clearText = new char[encodedText.length]
    for(int i=0 ; i<encodedText.length ; i++) {
        clearText[i] = key.decodeNumber(encodedText[i])
    }
    return clearText
 }
```

□

Grading.

- 0.5 pt for adhering to the listed assumption
- 0.5 pt for a correct algorithm. Anything works, pseudo-java, bullets points, text, etc.

Question 3.2 (1 pt)**.** *Assume the* `decodeNumber` *method of the Key interface runs in $O(n)$ time (worst-case), where n is the number of words in the "key" text. What is the worst-case runtime complexity of decoding a text that contains m characters? Justify your answer based on the number of calls to* `decodeNumber`*.*

Solution. The algorithm we proposed in the previous question iterates through the given encoded message and decodes each number using the `decodeNumber` method. Provided the given encoded text contains m characters, the `decodeNumber` method gets called m times, and, if this methods runs in $O(n)$, the whole decoding process runs in $m \times O(n)$, that is $O(m \times n)$. □

Grading.

- 0.5 pt for a correct answer, that is $O(m \times n)$
- 0.5 pt for a convincing explanation.

Question 3.3 (3 pts.)**.** *Given that the* `decodeNumber` *method runs in O(n) time, how would you improve its runtime efficiency. What pre-processing steps and what data structure would you use to achieve faster encoding of a letter? Explain how your proposed data structure and processing steps enhance performance.*

Solution. One way to reduce the time spent decoding is to preprocess the text and build an hashtable that would map each number to the letter it replaces. Because a hashtable offers retrieval by key in $O(1)$, the runtime of the

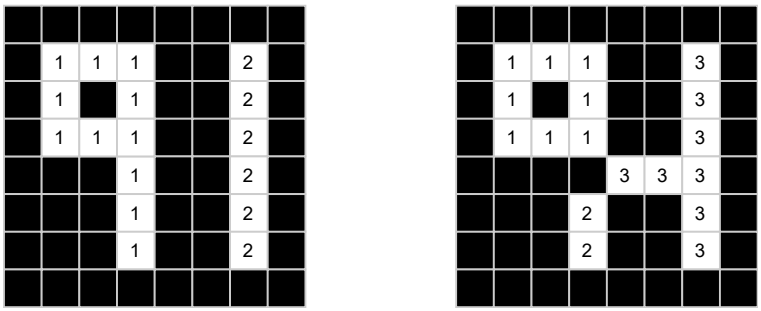

Figure 6: Two binary images and their connected components. On the left, there are only two connected components, whereas there are three on the right because pixel do not connect in diagonal

`decodeMessage` would becomes $m \times O(1)$, that is $O(m)$. In essence, a hashtable avoids re-scanning the whole key text for every single number. Building this hashtable requires however traversing the key text once, and for each word, mapping its position in text to its first letter. Because inserting a new entry in a hashtable runs in $O(1)$, building this table would therefore take $n \times O(1)$, that $O(n)$ time. □

Grading.

- 1 pt. for the presentation of the data structure, hash table, binary tree, etc.
- 1 pt. for explaining why this data structure reduces the overall runtime of the decoding process
- 1 pt for explaining how to build this data structure and how long that would take.

4 Connected Components Analysis

We now turn to computer vision and the detection of *connected components* in a binary image. A binary image is a grid of pixel that are either black or white.

The figures below illustrates the idea of binary images and connected components. A connected component is a group of contiguous white pixels. For instance on the left image, there are two connected components, labelled 1 and 2, respectively. By contrast, on the right image, there are three connected components, because their pixel are not contiguous (they are connected by a corner and not by an edge).

In this exercise, we are looking for an algorithm to detect such connected components in binary images, using graph algorithms.

Question 4.1 (1 pt.)**.** *How can a depth-first search (DFS) help us find connected components in a graph?*

Solution. Given a "start" node, a depth-first search will explore all the node that are reachable from that start node (in any number of edges). This set of nodes forms a single connected component. So by applying the DFS from all possible starting node, one can identify all the connected components. □

Grading.

- 0.5 pt for explaining that the DFSs reveal a single connected component
- 0.5 pt for explaining that multiple DFS runs would be needed

Question 4.2 (1 pt.)**.** *How would you model a binary images such as those shown above using a graph so that a DFS can be used? What would be a node, what would be an edge?*

Solution. To model a binary image as a graph, pixels would becomes nodes and edges would connect adjacent nodes of the same color. □

Grading.

- 0.5 pt for modeling pixels as node
- 0.5 pt for modeling edges as contiguous pixels of the same color

Question 4.3 (2 pt.)**.** *Propose an algorithm to count the connected components in a given image.*

Solution. To count the connected component in a graph, one could proceed as follows:

1. Initialize a counter $c = 0$
2. Initialize a hash-table. This table will eventually maps every positions in the image to the label of the connected component it belongs to.
3. For each pixel of the image,
   - If p is already in the hash-table, then continue to the next pixel
   - If p is black, add it to the table with "0" (i.e., black) as a label
   - Otherwise, if p is white, increment a the counter c by one, and use a depth-first-search (DFS) to obtain all the white pixels that are reachable from p. Add p and all these "reachable" pixels to the table with the label c
4. return c

□

Grading.

- 1 pt for an idea that is workable (even if the pseudo code, explanation is not 100 % correct). There are many approaches, e.g., two-passes labelling, sequential labelling and merging, etc.
- 1 pt for the quality of the solution/algorithm and the overall explanation.

Question 4.4 (2 pt.)**.** *In the worst case, what is the runtime complexity of your algorithm? Explain your reasoning?*

Solution. This algorithm gradually labels every pixels so the runtime complexity is $O(n)$ where n is the number of pixel in the image. There is in fact no worst case: Whatever is the arrangement of black and white pixels, the algorithm will have to look at each pixels at least once. The use of a hash table avoid recomputing pixels that are already in a connected components. Take for examples the following images configurations:

- The chess board where black and white pixels alternate so that there is no two white pixels contiguous. The algorithm will trigger a DFS for every white pixel, but this DFS will complete in $O(1)$ because there is no "connected" white pixels. So in total is will be roughly $\frac{n}{2} \times O(1)$, which is $O(n)$
- The white image, where all pixels are white. In that case, the algorithm will trigger a single DFS that will mark every pixels. This DFS will explore all the n nodes and the $4n$ edges (roughly). That gives us as well a runtime in $O(n)$.

□

Grading.

- 0.5 pt for the right answer, that is $O(n)$
- 0.5 pt for a convincing justification
- 1 pt (bonus) for understanding that there is no difference between the worst and the best cases.