CPSC 331: Assignment 3

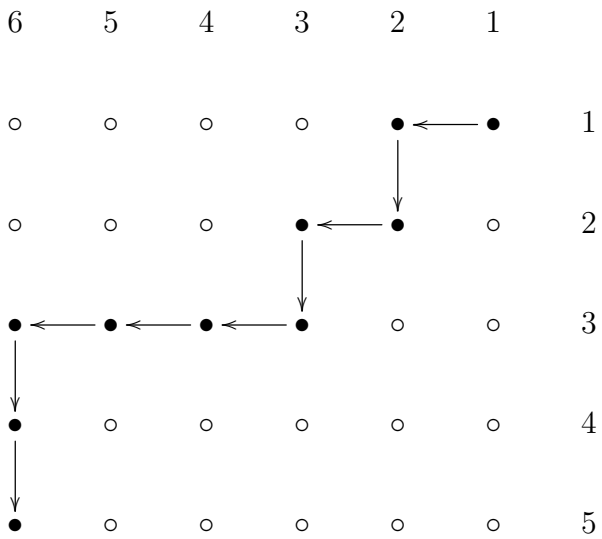
Fall 2023

See the D2L site for due date/time.

All references to exercises, page numbers, and sections refer to the textbook CLRS.

1. [20%] **Inversions.** Exercise 5.2-6 on page 134.

2. [20%] **Generating Random Paths.** We would like to automate the planning of movement for a robot situated in an $m \times n$ rectangular grid. Initially, the robot is located in the cell at the North-East cornor of the grid (the so-called initial position), and the goal is to find a path to the cell located at the South-West cornor of the grid (the so-called goal position). The robot can only make two types of movement: (a) move to the neighbouring cell South of its current location, or (b) move to the neighbouring cell West of its current location. The following is a path in a 5×6 grid:



   (a) Design an algorithm that randomly generates a path for the robot, so that all paths from the initial position to the goal position have an equal probability of being generated. Analyze the running time of your algorithm, and express it in terms of m and n.

   (b) Design an algorithm that takes an extra input (i, j), and randomly generates a path that starts at the initial position, then passes through cell (i, j), and finally reaches the goal position. (You may assume that $1 < i < m$ and $1 < j < n$.) All paths fulfilling the above requirement is generated with equal probability.

3. [20%] **Sorting Almost-Sorted Arrays.** Exercise 7.2-4 on page 191. To make your argument formal, consider ***k-sorted arrays***: an array A is k-sorted if and only if every element $A[i]$ will be sorted into one of the positions in $A[i-k..i+k]$. In other words, every element is located at no more than k positions away from its sorted position. For example, the following array is 2-sorted:

3	4	1	2	5	7	9	6	8

Observe that no element is more than 2 positions away from its sorted position.

1	2	3	4	5	6	7	8	9

Ask yourself, what is the performance of insertion sort and of quicksort when the input array is k-sorted? What value of k (in relation to n) will give insertion sort a performance advantage over quicksort (even when quicksort is randomized)?

4. [20%] **Anonymization via Scrapbooking.** In old movies from the last century, criminals blackmail their victims by writing anonymous letters. To do that, they would cut out characters from some magazine or newspapers, and paste the characters onto the letter so their handwriting will not be revealed. We use a string to represent a letter. We also represent a magazine by a string. Design an algorithm that takes an anonymous letter (a string) and a magazine (a string) as inputs, and returns a boolean value indicating if it is possible to construct the anonymous letter using characters from the magazine. Analyze the running time of your algorithm. **Hint:** Do remember that we now live in the era of globalization: You cannot assume that there are only 26 characters. For example, Java characters are encoded in the Unicode standard, which is capable of representing CJK characters (Chinese/Japanese/Korean). In short, the universe of characters is vast.

5. [20%] **Hash Table.** Implement a hash table class in Java.

   (a) The keys are of type **`String`**.

   (b) Your class shall be named **`HashTable`**. It shall be declared in the package **`ca.ucalgary.cpsc331.a3`**. Submit the Java source file **`HashTable.java`**. This is the only source file you need to submit.

   (c) Your class shall implement the following Java interface:

```
package ca.ucalgary.cpsc331.a3;

public interface Dictionary {
    // full(): returns true iff Dictionary is full.
    boolean full();

    // member(k): returns true iff key k is a member
    // of the Dictionary.
    boolean member(String k);

    // insert(k): If key k is not a member of the
    // Dictionary and the latter is not yet full,
```

```
    // then the method inserts k and returns true.
    // If key k is not a member of the Dictionary,
    // but the latter is already full, then the
    // method raises a RuntimeException. Otherwise,
    // the method returns false.
    boolean insert(String k);

    // delete(k): If key k is a member of the
    // Dictionary, then the method deletes k and
    // returns true. Otherwise, the method returns
    // false.
    boolean delete(String k);
}
```

(d) Your implementation shall employ open addressing, and more specifically, linear probing (page 297).

(e) Use **`java.lang.String.hashCode()`** to compute the hash code of a key.

(f) Your hash table has a maximum capacity of 17 slots (indices run from 0 to 16).

(g) When a key is deleted, leave a marker DELETED at the slot it previously occupied (page 295). When inserting a key that does not already exist in the hash table, if there are DELETED markers in the probe sequence before a NIL is encountered (or before the probe sequence is exhausted), then the first slot in the probe sequence that carries a DELETED marker shall be used for storing the new key.

(h) Your implementation shall raise a **`RuntimeException`** when the user attempts to insert a new key into a full hash table.

(i) The class shall provide a constructor that does not take any arguments. This constructor initializes the hash table to an empty one.

(j) The class shall override the **`toString()`** method. Your implementation of **`toString()`** shall return a **`String`** in the following format:

- The string consists of as many "lines" as there are non-empty slots. (A slot that is marked as DELETED is considered non-empty as well.) For example, if four of the slots are occupied by keys, and two are marked DELETED, then there shall be six lines. An empty hash table with no DELETED slot is represented by an empty string.
- Each "line" is terminated by a newline character. No line shall contain leading or trailing whitespaces.
- Each line represents a non-empty slot. Lines are ordered according to increasing indices of the slots.
- A line has the following format:

  `index:content`

  Note that there is no spaces around the colon character (`:`).

  - **`index`** is the index of the slot

- For a slot marked as DELETED, **`content`** is the string **`"deleted"`**. In other words, if the slot at index 3 is marked DELETED, then the corresponding line is the following:

  ```
  3:deleted
  ```

- For a slot occupied by a key, **`content`** is the key itself, delimited by double quotation marks. For example, if the slot at index 3 stores the key **`"deleted"`**, then the corresponding line is the following:

  ```
  3:"deleted"
  ```

  Note the difference in representation between a slot marked as DELETED and one storing the key **`"deleted"`**.