ASSIGNMENT 4

DFS LAB

Deadline: December 08, 2017

Each question carries 25 marks.

Q1. *Range search.* Write a program that takes distinct integers $x_1, x_2, \ldots, x_n$ as command line arguments and does the following.

(a) Sorts $x_1, x_2, \ldots, x_n$. Let $S = \{a_1, a_2, \ldots, a_n\}$ represent the sorted set (i.e., $a_1 < a_2 < \ldots < a_n$).

(b) Constructs a binary tree T from S as follows.

(i) Computes the median a_{mid} of S and inserts it into the root of T.

(ii) Partitions $S \setminus \{a_{mid}\}$ into 2 parts S_L and S_H where $S_L = \{a_i \mid a_i \in S \text{ and } a_i < a_{mid}\}$ and $S_H = \{a_i \mid a_i \in S \text{ and } a_i > a_{mid}\}$.

(iii) Recursively constructs 2 binary trees T_L and T_H from S_L and S_H respectively, and attaches them to the left-child and right-child links of the root node created in Step 1.

For example, if the command-line arguments to your program are 5, 10, 7, 20, 25, -10, and 15, the following tree should be constructed.

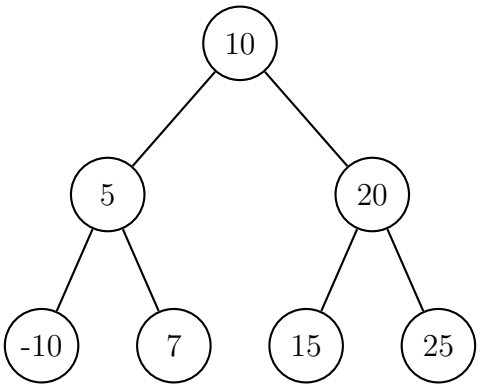

Figure 1: Example of binary tree constructed from input data

(c) Reads a pair of integers a and b from stdin and efficiently finds and prints the elements in T that lie within the interval $[a, b]$. For the example given in Figure 1, if the specified interval is $[4, 21]$, then your answer should be $\{5, 7, 10, 15, 20\}$.

For full credit, your program should take no more than $O(\log n + l)$ time, where l is the number of elements of T lying within the interval.

Q2. *Snakes and Ladders.* For this problem, you have to write a program to play the Snakes and Ladders game. Recall that the game is played using a 10×10 board, with the cells numbered 1–100. Each cell is of one of the following three types:

- *blank:* Nothing special happens when the player lands on such a cell.
- *ladder:* The cell contains the lower end of a ladder. If a player's marker lands on such a cell, it moves at once to the upper end of the ladder (i.e., a cell of a higher number).
- *snake:* The cell contains the mouth of a snake. If a player's marker lands on such a cell, it moves at once to the tail end of the snake (i.e., a cell of a lower number).

Any board may be described using 2 lines. The first line contains a list of cell numbers corresponding to the 2 end points of snakes; similarly, the second line contains a list of cell numbers corresponding to the end points of the ladders. For example, if a board is described using the following 2 lines:

21 15 99 79 35 18
41 59 70 94 77 84 12 28

it contains 3 snakes (mouths at 21, 99 and 35; tails at 15, 79 and 18 respectively), and 4 ladders (from 41 to 59, 70 to 94, etc.).

The game is played as follows. Each player in turn rolls a regular die (six faces) and moves her marker an appropriate number of cells. If the marker lands on a cell containing the bottom of a ladder or the mouth of a snake, the marker is moved to the top of the ladder / tail end of the snake. The game ends when one of the player's markers reaches cell number 100.

Special conditions:

- Each player has to roll a 1 on the die to get started (by moving to cell 1); otherwise, she remains stuck at zero, i.e., outside the board.
- A player's marker will not move at all if its target position is outside the board (e.g., if the marker is on cell 97, and the player rolls a 4).

Your program should take two command line arguments: a file name containing the board description, and an integer between 2 and 6 specifying the number of players.

It should first **check that the board is valid**, i.e., (i) all cell numbers are between 1 and 100; (ii) snakes go from higher numbered cells to lower numbered cells, while ladders go from lower numbered cells to higher numbered cells; (iii) no cell contains both the lower end of a ladder and the mouth of a snake; and (iv) there are no loops formed by the snakes and ladders (e.g., a snake from 21 to 19, a ladder from 19 to 34, and a snake from 34 to 21). Note that other combinations are permitted. A cell may contain both the upper end of a ladder, and the tail end of a snake. Snakes and / or ladders may also form chains (but no loops).

If the board is invalid, your program should print an error message and terminate. Othwerwise, it should describe how the game progresses, by printing 1 line corresponding to each roll of the die. You should use an appropriate random number generator to simulate the die. An example description (based on the example board containing 3 snakes and 4 ladders given above) is shown below.

```
Player 1 (cell 0): rolls 3, no movement.
Player 2 (cell 0): rolls 1, moves to cell 1.
...
Player 4 (cell 37): rolls 4, moves to cell 41, climbs ladder to cell 59.
...
Player 1 (cell 97): rolls 5, no movement.
Player 2 (cell 98): rolls 2, wins.
Game over.
```

Q3. *Term-document matrix.* Consider the following collection of text fragments, which we call book01, book02 and book03.

book01 = `John and Bob are brothers.`
book02 = `John went to the store.  The store was closed.`
book03 = `Bob went to the store too.`

If the words contained in these three books are converted to all-lowercase strings and sorted, we get the following list of distinct words.

`and`, `are`, `bob`, `brothers`, `closed`, `john`, `store`, `the`, `too`, `to`, `was`, `went`

This list is called the *lexicon* corresponding to the above collection of "books". We can now represent the collection of books as an $N \times M$ matrix A (called the *term-document matrix*), where

- N is the number of books in the collection;
- M is the number of distinct words in the lexicon; and
- $A[i, j]$ represents the number of times word j occurs in book i.

The term-document matrix for the above collection is given below.

	`and`	`are`	`bob`	`brothers`	`closed`	`john`	`store`	`the`	`too`	`to`	`was`	`went`
book01	1	1	1	1	0	1	0	0	0	0	0	0
book02	0	0	0	0	1	1	2	2	0	1	1	1
book03	0	0	1	0	0	0	1	1	1	1	0	1

Write a program that takes a list of text files as command-line arguments and prints the corresponding term-document matrix for the given collection. You may assume that the given text files will contain only upper and lower case letters and white space.