VG101 — Introduction to Computer and Programming

Lab 2

Instructor: Manuel

TA: Zekai — UM-JI (Summer 2019)

Goals of the lab

- Arrays and Matrices
- Function
- Recursion

1 Foreword

This time Haruka, Kana, and Chiaki went to an amusement park. It happened that they encountered many problems that can be solved by recursion.

2 Maze (Mandatory)

2.1 Introduction

They found that there was a large maze in that park. At the entrance of the maze, it wrote that there are in total k "treasures" hidden in the maze, and the one who collects all of them can win a prize!

However, Haruka found that up to now no one had won the prize. He doubted whether it is possible to find all the treasures. Maybe some treasures were put in places that cannot be reached. In order to find out the truth, they took a picture of the maze from a high sightseeing tower. However, the maze is too too too large, so they failed to figure out the number of treasures that can be found if they start searching from the entrance. Then they phoned Jane to help them solve this problem.

2.2 Workflow

2.2.1 Input and Output

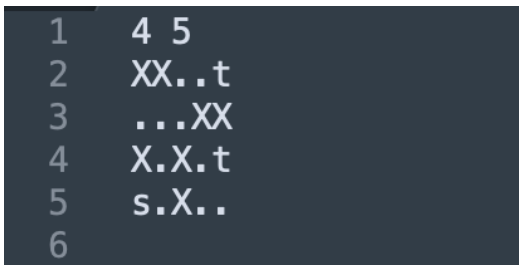
You should write a MatLab function to input the maze from a file `mazeinput.txt`.

i) The first line of the input is two positive integer n and m indicating that the maze has n rows and m columns.

ii) For the following n lines of the input, each line is composed of m characters representing the information of the corresponding cell.

- "." means a road (can pass).
- "X" means a wall (can't pass).
- "s" means the starting point.
- "t" means a treasure.

For example, the following input represents the maze on the right. The grey blocks indicate walls that Haruka cannot pass, and it's clear that Haruka cannot get out of the maze when searching. In this problem, Haruka can only walk in four directions: up, down, left, and right. In the example we give, there are in total 2 treasures, but Haruka can only find the one located on the upper right corner since the other one is surrounded by walls.



(a) Input file.

(b) Maze map.

Figure 1: The input of the maze

For the output, you should give the number of total treasures in the maze in the first line. In the second line, you should give the number of treasures that can be reached from the starting point.

2.3 Using DFS to solve the problem

The Depth-first search (DFS) is an algorithm frequently used to solve maze-related problem, realized by recursion. Recursion can be used here for the following 3 reasons:

i) The target is clearly defined, that we finished searching all the cells that can be reached.

ii) We are uncertain about how many steps are necessary to reach the target, which makes the problem hard to settle by simple loops.

iii) Each step of the process is generally the same. For this problem, we are just repeating the following steps.

- Check the four directions of the current cell. If it is neither unreachable nor reached before, take a step to the new cell.
- Check whether there's treasure in the new cell.
- Mark the new cell as reached.

3 Password (Optional)

3.1 Introduction

Chiaki is crazy of the room escape game, and he will join the game wherever there is one. Since Haruka, Kana and Chiaki are all very clever, they entered the last room of the game without any problem. However, the entrance to the last room closed automatically the moment they entered the room, and the only way they could get out was to correctly input a n digit password. Luckily, according to the clues they found in the previous rooms, Chiaki determined that the m characters that could possibly make up the password.

But it was still a tough work, since Chiaki had to try every single possible permutation. In despair about the endless trials, he again asked for Jane's help.

3.2 Workflow

3.2.1 Input and Output

You should write a MatLab function to input from a file `pwinput.txt`.

i) The first line of the input is two positive integer n and m indicating the length of the password and the number of characters that will appear in the permutation.

ii) The second line of the input is m different characters. Note that there is **no** spaces between each two characters.

The output should be written into a file `pwoutput.txt`, with each line a string indicating the possible password. Since all the characters are different, there should be in total $\mathcal{P}_m^n = \frac{m!}{(m-n)!}$ possible passwords. For example, if we look for all the permutation of length 3, composed of 4 characters, "u", "m", "j", and "i", then we have in total 24 of them.

Table 1: All the possible password

jmu	mju	muj	imu	miu	mui
jum	ujm	umj	ium	uim	umi
iju	jiu	jui	iuj	uij	uji
ijm	jim	jmi	imj	mij	mji

3.2.2 Using Steinhaus—Johnson—Trotter algorithm to solve the simplified problem

There are really a lot of algorithms in generating permutation, even if we only consider those with recursion. The Steinhaus—Johnson—Trotter algorithm is one of the most prominent permutation enumeration algorithms, and you can find the detailed introduction to it on Wikipedia. Using this algorithm, we can easily find all the permutation of length n with exactly n distinct elements.

The idea of this algorithm is pretty simple, that if we already have a permutation P of length $n-1$, we can insert the n^{th} element between any two elements of P or to the very beginning or the very end of P. For example, the permutation of length 3 can be generated like this:

$$\mathbf{1} \Longrightarrow \begin{matrix} 1\mathbf{2} \\ \mathbf{2}1 \end{matrix} \Longrightarrow \begin{matrix} 12\mathbf{3} & 21\mathbf{3} \\ 1\mathbf{3}2 & 2\mathbf{3}1 \\ \mathbf{3}12 & \mathbf{3}21 \end{matrix}$$

Here, the idea we employ to solve this problem is called `Divide and Conquer`. We are faced with a problem with large size but we have no idea how to solve it directly. However, we can divide the problem into several subproblems that look the same as the original problem, but with a smaller size. In this way, we can keep this kind of division until we get a problem easy to solve. With all the subproblems solved, it becomes possible for us to solve the original one.

Note that for this algorithm, you can easily write it in a iterative way without using recursion. This is very different from the other two problems included in this lab, and the fundamental reason is that we know exactly how many steps we need to reach the target, and for each step, we only need information from the previous step.

3.2.3 Necessary modification to solve the original problem

Note that in the simplified problem there are only n characters for us to choose. If there are more than n characters, or some of the given characters are the same, maybe you will have to use another algorithm to solve the problem, otherwise you must modify the Steinhaus—Johnson—Trotter algorithm, because it cannot be directly applied under these circumstances.

In this problem, there are m available characters. That means when we are constructing the permutation of length n we are not necessarily using the n^{th} character, which is the essential difference from the simplified one. An intuitive idea is that, since we don't have to select the next character, we can skip some characters and choose the one after them. Because the length of the permutation is shorter than the number of the available characters, we know that in the recursion we can skip at most $m-n$ characters so that we can still construct a complete permutation.

4 Toy (Optional)

4.1 Introduction

Kana may be the best one in the world at grabbing toys from a crane machine. After the day in the park, Kana got in total n toys, and he decided to give all of them to his friends (because he has already owned a lot).

However, Kana wanted to sort these toys by their tagged price (though they're not bought) from high to low so that he could give the most expensive toy to his best friend, and give the second most expensive toy to his second best friend...

Instead of asking Haruka to do this job manually, Kana paid Jane (a toy) to write a program for him.

4.2 Workflow

4.2.1 Input and Output

You should write a MatLab function to input from a file `toyinput.txt`.

i) The first line of the input is a positive integer n indicating the number of toys Kana had grabbed.

ii) The second line of the input is n integers separated by a space, indicating the price of each toy.

The output should be written into a file `toyoutput.txt`, with one line of n integers indicating the sorted price from high to low.

4.2.2 Using Quick Sort algorithm to solve the problem

There are quite a lot of way to sort a sequence of numbers. Maybe many of you have heard about the Insertion Sort or the Bubble Sort. Quick Sort is generally a more efficient sorting algorithm than the mentioned two, and again the idea of `Divide and Conquer` is used in it.

The Quick Sort algorithm can be described as follows.

i) We pick an element from the unsorted array, and it is called the `pivot`. We can actually pick any element in the array as the `pivot`.

ii) We rearrange the array, so that all the elements with value smaller than the `pivot` is put after it, and all the elements with value greater than the `pivot` is put before. In this way, we can see that the `pivot` is actually in its final position.

iii) Now that we have partitioned the array into two unsorted subarray, we can do the recursion, repeating the above two steps to make the whole array in order.

There are several ways to implement this algorithm, see Wikipedia. Here, let's take a look at an example from Wikipedia to see how Quick Sort works. Note that in this example, we do the sort in the order from low to high.

In this simulation, the last element is always picked as the `pivot`. We can see that first 4 is selected as the `pivot`, then all the elements greater than 4 are switched after 4. At the same time, 4 is switched forward and at last goes to its final position.

Then the `pivot` partitioned the original array into two subarraies, and the recursion goes on until all the elements are in their final positions.

We can see that in this process the relative position of the numbers other than 4 changed. The numbers smaller than 4 are 1,2, and 3, and initially they're in the order of 3, 2, 1. But after the switch is over, they're in the order of 3, 1, 2. However, since we only need to separate the elements in terms of their relation to the `pivot`, we don't care their relative position.

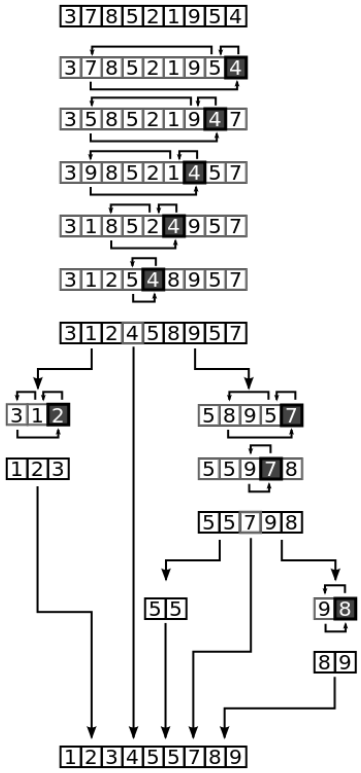


Figure 2: Quick Sort Process

5 End

Haruka, Kana, and Chiaki finally had a happy day in the amusement park. However, Jane only hoped that she wouldn't ever need to do so many recursion problems in one day.