Final Documentation

Seyda Koclar - Athmar Nabil Shamhan

January 2022

1 Introduction

In this document, the algorithm constructed in the project will be analysed not theoretically as it was done in the functional document but practically with the implementation of the algorithm and results of the generated test inputs. As a reminder let us briefly explain the algorithm:

- Creating the necessary rotations with taking width as maximum of the dimensions and as first dimension; length as minimum of the dimensions and as second one.
- Applying merge sort only according to first coordinate, i.e., width, and sort the array in decreasing order.
- Applying the solution of Longest Decreasing Subsequence algorithm with binary search having the complexity of $\Theta(NlogN)$

Overall time complexity of the algorithm was found as $\Theta(NlogN)$ and in this document, the actual behavior of execution time of the algorithm itself and the behavior of function NlogN will be compared to check the correctness of our solution.

2 Modifications

The implementation of the algorithm was done according to the pseudo-code given in the first documentation and tested. After the testing phase, it was observed that the algorithm does not work for some specific cases. LDS with binary search works for one dimensional array perfectly but when two dimensional array is taken into account the first dimension, i.e. width, changes the sequence and unsolved errors were occurred. For instance, for a sequence stated below:
(900,2)
(558,315)
(555,312)
(547,313)

(547,268)
(539,263)
(530,259)
(524,287)
(522,246)
(521,264)
(509,251)
(501,241)

The algorithm finds the sub sequence as:
(501,241)
(509,251)
(521,264)
(524,287)
(547,313)
(558,315)

However, we can have:
(501,241)
(522,246)
(530,259)
(539,263)
(547,268)
(555,312)
(558,315)

In the binary search, always the best option for the position is chosen and in that case in the fourth iteration when i=3, box (547,313) overrides the box (555,312) and because of this the correct sub sequence stated above is never chosen by the algorithm.

As a result of the reason stated above the algorithm was changed and the solution with time complexity O(N^2) was implemented. In this solution the idea is very similar to the one with binary search hence with couple modifications the algorithm was corrected.

O(N^2) solution of LDS finds the sub sequence having maximum length among all possible sub sequences ending with the i-th box considering the elements up until the i-th box in each iteration of i from 1 to n.

Z[i] stores the maximum sub sequence length among all the sub sequences ending with the i-th box. Hence, in each iteration, instead of finding the sub sequences again, the algorithm checks each box j positioned before i-th box and if box i has smaller length than box j and their width are not the same, which means it is decreasing since we already sorted the boxes according to their widths, and Z[i] is less than Z[j] + 1(appending the sub sequence found before since its length will be greater than Z[i]), it will make Z[i] equals to Z[j]+1 and store j as parent of i in Parents array.

For restoring the sub sequence itself, at first the maximum length is found

by looking at the elements of array Z and the index of having maximum value is taken as 'parentIndex', which shows the innermost box at first. Then in the while loop the boxes of the longest decreasing sub sequence is written to the file.

All of these modifications can be seen below:

```
public static void LDS(Box[] Boxes, string fileNumber)
{
    string outputfile = Directory.GetCurrentDirectory() + "\\output" + fileNumber + ".txt";

    using (StreamWriter outputFile = new StreamWriter(outputfile))
    {
        int n = Boxes.Length;
        int[] Parents = new int[n];
        int[] Z = new int[n];

        //initialize Parent and Index arrays referred as P and I in the document
        //Parent is used to keep track of the subsequence
        //Z used to keep the length of the subsequences
        for (int i = 0; i < n; i++)
        {
            Z[i] = 1;
            Parents[i] = -1;
        }

        for(int i=1; i<n; i++)
        {
            for(int j=0; j<i; j++)
            {
                if (Boxes[i].length < Boxes[j].length && Boxes[i].width != Boxes[j].width && Z[i] < Z[j] + 1)
                {
                    Z[i] = Z[j] + 1;
                    Parents[i] = j;
                }

            }
        }

        int length = Z[0], parentIndex = 0;
        for (int i = 0; i < n; i++)
        {
            if (Z[i] > length)
            {
                length = Z[i];
                parentIndex = i;
            }
        }

        //write sequence itself to the output file
        outputFile.WriteLine(length.ToString());

        while (parentIndex != -1)
        {
            outputFile.WriteLine(Boxes[parentIndex].ToString());
            parentIndex = Parents[parentIndex];
        }

        outputFile.Close();
    }
}
```

Figure 1: Modification of LDS

There is a slight modification in merge sort which was not necessary but done for differentiate the sorting of the boxes having the same widths. In the below picture the modification can be seen:

```
else if(LeftArray[leftIndex].width < RightArray[rightIndex].width)
{
    Boxes[k] = RightArray[rightIndex];
    rightIndex++;
}
else
{
    if (LeftArray[leftIndex].length > RightArray[rightIndex].length)
    {
        Boxes[k] = LeftArray[leftIndex];
        leftIndex++;
    }
    else
    {
        Boxes[k] = RightArray[rightIndex];
        rightIndex++;
    }
}
```

Figure 2: Modification of Merge

The second 'else' part is added for sorting the boxes according to second dimension, i.e, length, in the case when the widths of the boxes are equal.

Since the algorithm was changed, the amount of test cases were changed as well. For O(N^2) complexity the algorithm was very slow with 50000 boxes hence further test cases such as 100000, 1000000, 200000 were not used. That is why python script is changed according to this.

The last modification that was made is about reading multiple input files. There are extra code lines added to the algorithm so that program can read multiple input files and operate on them, then writes output files for each one of them in a single run. In addition to this, a watch is added to the code so that the execution time can be measured and all of the measurements are collected in one file called "measurements.txt". Execution time is measured as EllapsedTicks which shows the number of timer ticks. This measurement is chosen so that the value can be compared with the value of N^2. If a time measurement was chosen such as milliseconds, it would give different unit of measurement, hence, would be wrong to compare it with the result of N^2.

For generating multiple test cases another C# code was implemented and for plotting results and the function itself a python script was written and used.

3 User's Manual

To be able to run the program, three steps are needed:

- Arranging names of all input files in a format like: "input*.txt" where integer numbers should be placed instead of *. These numbers represent the

file number and program will generate files in a format like: "output*.txt" where values of * will match with the values of input files. For example, the program will write the output of "input1.txt" to "output1.txt"

- All of the input files must be in the same folder with the .exe file
- Double click on .exe file is enough to run the program if the above conditions are satisfied. After clicking, all output files and "measurement.txt" file should be generated.

Note that if an input file is not provided the program will create a log file and inside this file the error log can be seen. Moreover, because of the way the program works, please do not forget to remove measurements.txt file if you want to test the program again. Since it appends the result into measurements.txt, it will not override and the file will keep the existing results found before.

In the project folder, the code for the test generator is given as well. One can test the algorithm differently than ours and by running the given python script, the plot based on measurements can be generated. Note that "measurements.txt" file must be in the same folder with the "main.py" file so that program can plot figures.

This script creates one .png file named "graph.png" showing both real execution time measured by our algorithm and the result of N^2 based on the designated values of N in the test generator. To execute the script, typing ".\main.py" command is enough. As final remarks:

- Since the folder size for inputs and outputs are over 500MB, IO folder containing all the inputs and outputs is sent as another link for simplicity.
- Inside the folder called TestGenerator, .exe file of the generator and the codes of it can be found. If you run .exe, the program will ask to enter number of boxes and inputs, after that new randomly created inputs should appear in the same folder.
- In folder named PythonScript, main.py, graph.png and measurements.txt files can be found.
- The file called MainAlgorithm.cs is the code for our algorithm and the .exe file, which is named as ProjectBoxes, is the executable of this main program.
- To make the .zip file smaller, we prefer to give only the codes for test generator, the main algorithm and script for plotting instead of giving all project files. In addition to thos

4 Test Description

In the test generator, 320 input files are created for 16 different values of N in total. The values 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 100, 500, 1000, 5000,

10000, 50000 are chosen as number of boxes, which is N, and for each of them 20 input files are created. For each N, the dimensions of the boxes in the first 10 file is randomly created in range (1,100) and the other 10 is created in range (100,1000). These files included in the project folder.

The results of each input are written into the corresponding output files and also included in the project folder with "measurements.txt" as well.

5 Conclusion

When the results are checked it can be seen that algorithm can find the longest sequence and print it correctly. To plot a graph, the average execution times are calculated for each N in the python script. For instance, when N = 100 we have 20 different input files and measurements for each of them. The average is calculated by summing all of the execution times and dividing it to the number of instances we have.

In the python script we have an array of showing the summation of execution times and the occurrences of each N as below for our test cases:

[[5, 879820, 20], [10, 756916, 20], [15, 698811, 20], [20, 777971, 20], [25, 846090, 20], [30, 986598, 20], [35, 757568, 20], [40, 757135, 20], [45, 749733, 20], [50, 857428, 20], [100, 980915, 20], [500, 1291056, 20], [1000, 1849300, 20], [5000, 17532650, 20], [10000, 56528201, 20], [50000, 1206328633, 20]]

These values is shown in the below table to make them more readable:

N	ET	O
5	879820	20
10	756916	20
15	698811	20
20	777971	20
25	846090	20
30	986598	20
35	757568	20
40	757135	20
45	749733	20
50	857428	20
100	980915	20
500	1291056	20
1000	1849300	20
5000	17532650	20
10000	56528201	20
50000	1206328633	20

The first column shows N, second shows total execution time for this N and third is the occurrence of N. Then we created an array for showing all values of

N as below:

[5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 100, 500, 1000, 5000, 10000, 50000]

For the average execution time for each of these N, we created another array and filled with the calculated values as below:

[43991.0, 37845.8, 34940.55, 38898.55, 42304.5, 49329.9, 37878.4, 37856.75, 37486.65, 42871.4, 49045.75, 64552.8, 92465.0, 876632.5, 2826410.05, 60316431.65]

These values are shown in the below table to make them more readable:

N	AET
5	43991.0
10	37845.8
15	34940.55
20	38898.55
25	42304.5
30	49329.9
35	37878.4
40	37856.75
45	37486.65
50	42871.4
100	49045.75
500	64552.8
1000	92465.0
5000	876632.5
10000	2826410.05
50000	60316431.65

We can see average execution time, which is denoted as AET, for each N. Then we plotted the graph of these alongside with the result of calculation of N^2 for designated values of N. The graph in Figure 1 shows both the values derived from our algorithm and calculation of N^2.

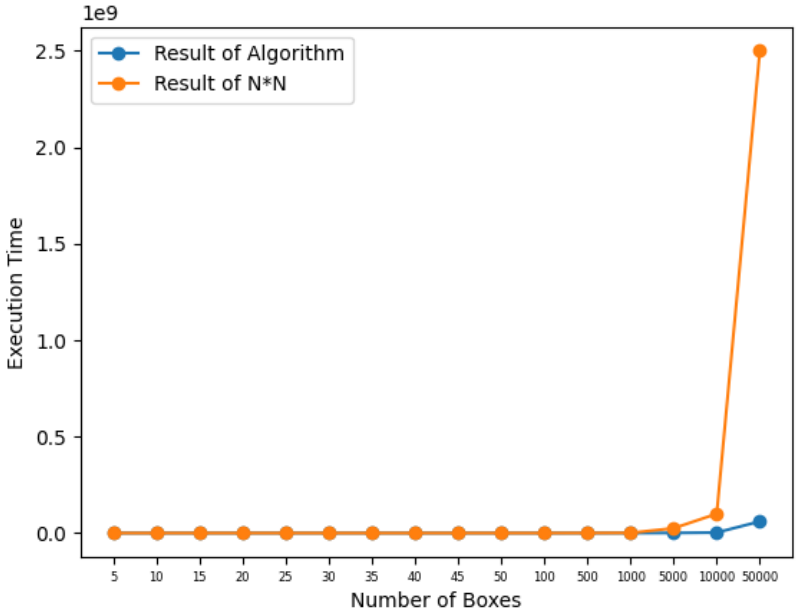

Figure 3: Graph showing execution time depending on the number of boxes

In the graph, the x-axis shows the number of boxes while y-axis is showing the execution time in ticks and as it can be seen it is denoted with scientific notation, i.e., 2.5 means it is 2.5e9 and it is around 2500000000 ticks. These values shows approximation of the actual results given above.

It is seen that the result derived from our changed algorithm and the theoretical execution time has a similar behavior and $O(N^2)$ is actually an upper bound for our actual execution time as O notation states.

To conclude from the observations explained above, our modified algorithm works correctly.

6 Job Partition

In this project all workloads have been done together by the project members. There is no sharing of the tasks but instead the tasks are done with collaboration of the members.