Project Boxes: Project Documentation

Group Member: Lu Zhicong, Zhang Boxuan

1. Project Abstraction

To solve the matryoshka boxes problem, we use dynamic programming method in this project. According to our analysis, the algorithm has O(N log N) time complexity in total. We implement this algorithm in C# language.

In this project, we use 4 steps to implement this algorithm.

- The first step is to sort the box set into sequences by length. In this step we implement a merge sort function for Box class, so the time complexity of this step is O(N log N).
- The second step is to initialize the data structure for dynamic programming, we use array type in C# to store and the time complexity of initializing is O(N).
- The third step is the main part of this project. By iterating through the sorted boxes sequence, we calculate the max matryoshka sequence length ended in the i-th box for every box. Using another array to store the preceding smallest box width, here we can use binary search algorithm to reduce the time complexity of one single iteration to O(log N). Thus, the total time complexity for this step is O(N log N).
- The last step is to restore a possible matryoshka sequence with max length. We iterate through the max sequence length array in reversed order and then get a complete box sequence. The total time complexity for this step is O(N).

In conclusion, the total time complexity cost for this algorithm is

O(N log N) + O(N) + O(N log N) + O(N) = O(N log N)

2. Change Log

- **Change the sorting algorithm form Heap sort to Merge sort in the first step to sort the whole box set.**
- **Add a data generator for this project. User can call a different command to generate random data.**

2.1. Data Generator Description

Using `./bin/Boxes-Alpha **generate**` command to call this function, this function will generate given number of boxes with random width and random length. The width and length are double type number between 1 and 50.

- **Change the output and input format in functional document. Now the input file and output file are in the same format, thus we can use the same io function to generate input file, read input and write output.**

2.2. Input Format

First Line: **N** (integer that shows the number of boxes)

Next **N** lines: **index x y** (index is the index of box (from 1 to N), x and y are double values larger than 0; split char: ‘ ’)

2.3. Input Example with comment

```
3                  //number of boxes
1 2.4 2.0          //B1(x1 = 2.4, y1 = 2.0)
2 5.0 6.0          //B2(x2 = 5.0, y2 = 6.0)
3 1.0 10.0         //B3(x3 = 1.0, y3 = 10.0)
```

2.4. Output Format

First Line: L_{max} (integer that shows the length of the longest sequence)

Next L_{max} lines: **index x y** (index is the index of box getting from input file, x and y are width and length of box).

2.5. Output Example with comment

```
2                  //length of longest sequences
1 2.4 2.0          //B1(x1 = 2.4, y1 = 2.0)
2 5.0 6.0          //B2(x2 = 5.0, y2 = 6.0)
```

3. User Manual

The whole project is under “Boxes-Alpha” directory. Source codes are listed under the root directory of this project. Executable file is under “./bin/” directory.

In general, the program has 2 main functions: to generate boxes input file, and to implement the algorithm. Additional function includes run algorithm in batch (for time complexity experiments).

To run this program, use “./bin/Boxes-Alpha”. Otherwise use “dotnet run” to compile and run.

3.1. Data Generator

Function:

Generate boxes data for algorithm input. Number of boxes are user-defined. Width and length are generated randomly.

Command :

./bin/Boxes-Alpha **generate** -boxes_num=[boxes_number] -boxes_file=[boxes_filepath]

[boxes_number]: the number of boxes you want to generate.

[boxes_filepath]: the file to store these randomly generated boxes data. Also used for algorithm input.

3.2. Run Algorithm to Sort Boxes in Matryoshka

Function:

Implement the algorithm. This command will show the result in console and also store result in user defined path.

Command :

```
./bin/Boxes-Alpha sort -boxes_file=[boxes_filepath] -result_file=[result_filepath]
```

[boxes_filepath]: the file to store these randomly generated boxes data. Also used for algorithm input.

[result_filepath]: the file to store algorithm result.

3.3. Other Functions

3.3.1. runbatch

Function:

To run the algorithm in batch. This command will use a set of boxes number from 50 to 100,000 and for each boxes number, program will generate random boxes for 10 times, and calculate the max, min, and average running time of each iteration.

Command :

```
./bin/Boxes-Alpha runbatch > time_analysis.txt
```

3.3.2. help

Function:

Show program command line usage.

Command :

```
./bin/Boxes-Alpha help
```

4. Time Complexity Test and Analysis

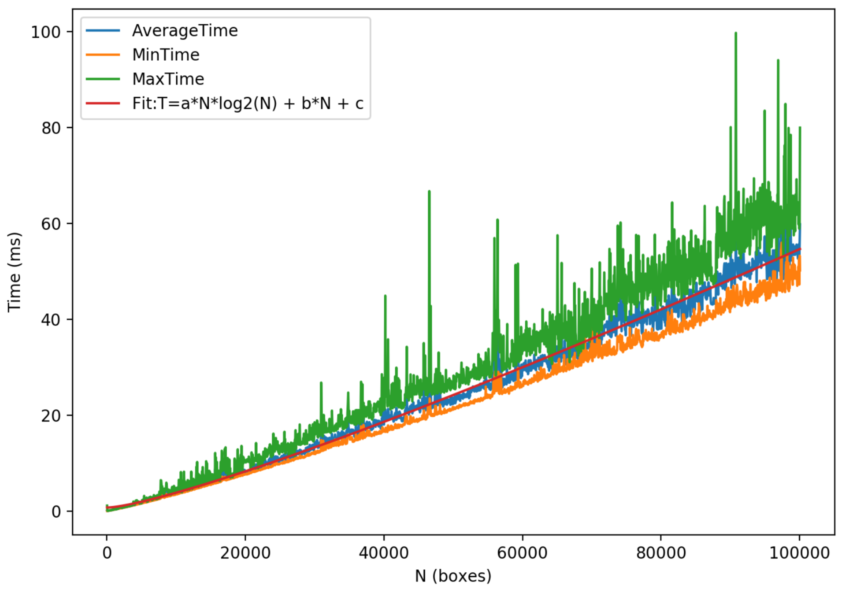
4.1. Description

In time complexity test (using ./bin/Boxes-Alpha runbatch command), we test the running time of boxes number from 50 to 100,000. And from each boxes number N, we also run 10 different tests to calculate the maximal time cost, minimal time cost and average time cost. For analysis, we use a theoretical Time Cost function O(N log N) :

$$t = a * N * logN + b * N + c$$

to fit the average time cost data.

4.2. Result



In the figure, green line shows the maximal time cost, orange line shows the minimum and the blue line shows the average time cost in 10 tests. There is also a red line that shows the fit result of function $t = a * N * logN + b * N + c$.

From this plot we can see that the average time cost is generally fitted to O (N log N) time complexity, which prove that our time complexity analysis is true.

5. Partition

5.1. Functional Document

Solution: **together**.

Problem description, Pseudocode: **Lu Zhicong**.

Correctness, Time Complexity Proof, File Format: **Zhang Boxuan**.

5.2. Program

Implement: **Zhang Boxuan**.

Data Generator: **Lu Zhicong**.

5.3. Final Document

Time Complexity Experiments and Analysis: **Lu Zhicong**.

Other parts of Documents: **Zhang Boxuan**.