Zhejiang University

College of Computer Science and Technology

Strip Packing

Project6 Report by Group1

Author:
Guo Jiahao
Wu Yihang
Sun Xinjie

Supervisor:
Mao Yuchen

Abstract:

An Assignment submitted for the ZJU:

21120491 Advanced Data Structure and Algorithm Analysis

May 24, 2022

1 Introduction of the Project

In this project, we need to solve the strip packing problem. The strip packing problem is a 2-dimensional geometric minimization problem. Given a set of axis-aligned rectangles and a strip of bounded width and infinite height, determine an overlapping-free packing of the rectangles into the strip minimizing its height. We may assume that the width of any rectangle is no more than the width of the bin, and we are not allowed to rotate the rectangles, and that the rectangles should not overlap.

Note that we need to implement at least two polynomial-time approximation algorithms for this problem. We must generate test cases of different sizes with different distributions of widths and heights, compare and analysis the solution quality and the running time of the two algorithms on these test cases.

2 Introduction of the Algorithms

2.1 Next Fit(NF)

We first discuss the simplest method to solve this problem which called next fit algorithm. In this algorithm, We place the rectangles one by one. When the bottom edge of the next rectangle cannot be placed at the same height as the previous rectangle, we cover the "cover" of the previous layer, that is, place the bottom edge of the next rectangle on the height of the top of the highest rectangle on the current layer. Pseudo code is shown below:

Algorithm 1: Next Fit(NF)

```
  Input: width $w$ and height $h$ of input rectangles
1 let $h = 0$;
2 for Rectangle $R = (w, h)$in the sequence do
3     Try to fit the rectangle onto the current open shelf;
4     if It does not fit then
5         close the current shelf and open a new one;
6         $h = h +$ max height of the previous layer;
7     end
8 end
```

2.2 First Fit(FF)

Then we discuss the first fit algorithm, which is the first optimized algorithm we discuss today. This time we don't cover the "cover" immediately after we can't put the next rectangle at the previous level. This time if the next rectangle can be put into the interval of any previous layer, we put this rectangle into the first layer that satisfies. Pseudo code is shown below:

```
Algorithm 2: First Fit(FF)
Input: width w and height h of input rectangles
let $h = 0$;
for Rectangle $R = (w, h)$ in the sequence do
    Using bisection method to find the first layer that can fit the rectangle;
    if No previous layer fit then
        close the current shelf and open a new one;
        $h = h$ + height of the rectangle;
    end
end
```

2.3 Best Width Fit(BWF)

Now we need to discuss the second optimized algorithm which is similar to the first fit algorithm, but this time we don't put the next rectangle to the first layer that fit the rectangle but the best layer where best means the it has the smallest interval that fit the new rectangle. Pseudo code is shown below:

```
Algorithm 3: Best Width Fit(BWF or BF briefly)
Input: width w and height h of input rectangles
let $h = 0$;
for Rectangle $R = (w, h)$ in the sequence do
    Using balanced tree to find the best fit layer;
    if No previous layer fit then
        close the current shelf and open a new one;
        $h = h$ + height of the rectangle;
    end
end
```

Lets look at an example to understand thoroughly how the algorithms works. When the width required is 10 and the following rectangles are to be placed:

Rectangle (5, 6) - Rectangle (3, 5) - Rectangle (9, 4) - Rectangle (10, 3) - Rectangle (1, 2) - Rectangle (2, 1)

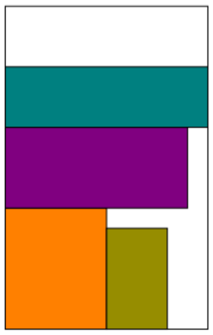

Figure 1: After inserting the first four rectangles

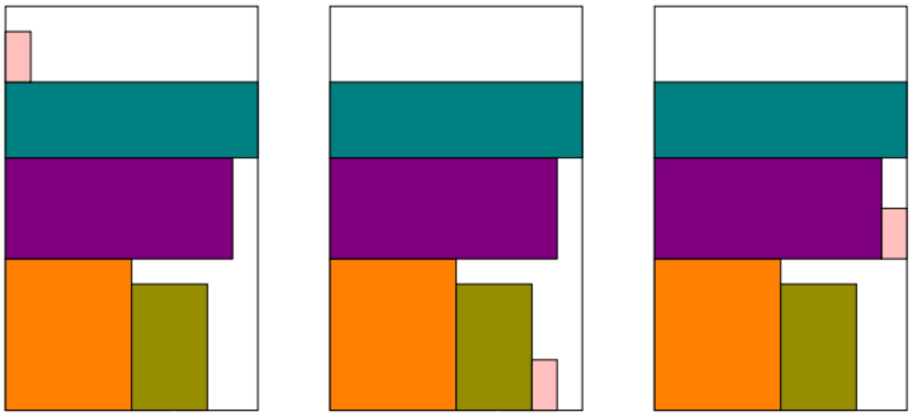
(a) NF   (b) FF   (c) BF

Figure 2: After inserting the fifth rectangle

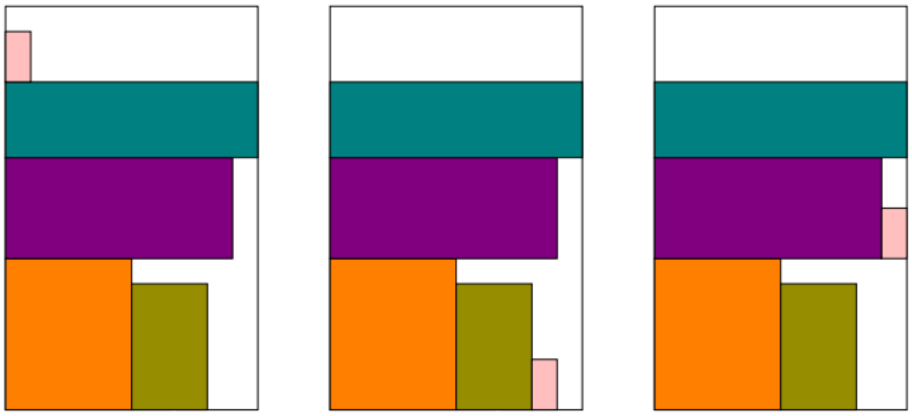
(a) NF   (b) FF   (c) BF

Figure 3: After inserting the sixth rectangle

As is shown in 1, the three algorithms yield the same result when the rectangles are inserted in specific way. While the empty space for the current level is not enough, first fit and best fit use different way to manage it rather than creating a new level, as is shown in 2b and 2c. The way for best fit is cleverer than first fit since the empty space is managed better, as is shown in 3c.

3 Algorithm Analysis

3.1 Running time

3.1.1 Next Fit

It's easy to see that we only need $O(n)$ time where n is the number of inout rectangles(the same in the following sections) because for every input rectangle, we only need to judge if it can be put in the current layer in constant time, so the total time is linear.

3.1.2 First Fit

Because for every input rectangle, we need to find the previous layers to determine where to put. If we use linear scan to find the first fit position, $O(n)$ time is needed. If we use bisection method, we only need $O(\log n)$ time to implement this algorithm.

3.1.3 Best Width Fit

Because for every input rectangle, we need to find the previous layers to determine where to put. If we use linear scan to find the best fit position, $O(n)$ time is needed. If we use balanced tree to find the proper position, we only need $O(\log n)$ time to implement this algorithm.

3.2 Approximation rate

Lets denote the optimal result as OPT, the sum of areas of all rectangles A, the result of NF, FF and BWF NF, FF and BWFrespectively. We have implied in the example that

$$OPT \leq BWF \leq FF \leq NF$$

For the convenience of our proof, assume all width are normalized to the required width, i.e set the required width to 1 and all rectangles has width smaller than 1 for convenience. First of all, we would estimate the approximation ratio of NF as an upperbound of all the algorithms[1].

Theorem 1. $NF \leq 2OPT + h_{max}$ where h_{max} is the maximun height of all the input rectangles.

Proof. Denote the height of level i H_i, then

$$NF = \sum_i H_i$$

Because the next level is generated as long as the first rectangle in the next level is not compactible with the current level, we have

$$A_i + A_{i+1} \geq H_{i+1}$$

where A_i denotes the total area of the rectangles in level i. Thus:

$$NF = H_1 + \sum_{i=2}^{n} H_i \leq H_1 + 2A \leq H_1 + 2OPT$$

Then we are to show a result about the upperbound of the FF algorithm, which is more more complicated.

Theorem 2. $FF \leq 1.7OPT + h_{max}$ where h_{max} is the maximun height of all the input rectangles.

Proof of the above theorem[1] is omitted here for clarity. BF approximation ratio is hard to estimate, with no related works found.

4 Experiment and Result

Operation	**Binary heap**	**Leftist heap**	**Binomial heap**	**Fibonacci heap**
Make heap	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Find min	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Insert	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$
Delete min	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
Merge	$O(n)$	$O(\log n)$	$O(\log n)$	$O(1)$
Delete	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
Decrese key	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$

Table 1: Running times of all operations of different heaps

5 Discussion and Conclusion

6 References

[1] *Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms*, E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan, SIAM Journal on Computing 1980 9:4, 808-826