

Determining Vertical Adjacencies Between Orthogonal Rectangles Using Brute Force

Salmaan Ebrahim

April 27, 2023

1 Aims

The aim of this project is to create a Python program to identify vertical adjacencies among a given set of orthogonal rectangles using a brute force algorithm. The program prompts the user to input the number of rectangles and their coordinates. It checks vertical adjacencies by considering vertical corners, full vertical edges, and part vertical edges that touch as adjacent on the right-hand side of each rectangle. The program outputs the number of adjacent rectangles on the right-hand side of each rectangle along with their x coordinate, bottom y coordinate, and top y coordinate. If a rectangle has no vertical adjacencies on the right, the output shows zero adjacent rectangles. The aim is to also determine the performance and test the runtime of the brute force algorithm and to determine if this algorithm best suits this particular problem.

Input format:

- Number of rectangles (integer)
- For each rectangle (on a new line), the following information:
 - Rectangle number (integer)
 - x and y coordinates of the bottom-left corner (integers)
 - x and y coordinates of the top-right corner (integers)

Example:

```
5
0 1 10169 10159 20000
1 10159 10169 20000 20000
2 15233 1 20000 5045
3 15233 5045 20000 10169
4 10159 5045 12861 7413
```

Output format:

- For each rectangle (on a new line), the following information:
 - Rectangle number (integer)
 - Number of adjacent rectangles on the right (integer)
 - For each adjacent rectangle, the following information:
 - * Rectangle number of the adjacent rectangle (integer)

- * x coordinate of the adjacent rectangle (integer)
- * bottom y coordinate of the adjacent rectangle (integer)
- * top y coordinate of the adjacent rectangle (integer)

Example:

```
0, 1, 1, 10159, 10169, 20000
1, 0
2, 0
3, 0
4, 0
```

Note that the program only identifies vertical adjacencies on the right-hand side of each orthogonal rectangle and does not consider adjacencies on the left-hand side or above/below the rectangle.

2 Summary of Theory

Identifying adjacent rectangles and their coordinates in a set of orthogonal rectangles is a classic problem in computational geometry. A brute force algorithm can solve this problem by checking all pairs of rectangles for vertical adjacency, where vertical corners touching, full vertical edges touching, and part vertical edges touching are counted as adjacent. This algorithm has a time complexity of $O(n^2)$, where n is the number of rectangles. Although more sophisticated algorithms such as the sweep line algorithm and the line sweep algorithm have better time complexities of $O(n \log n)$ and $O(n)$, respectively, the brute force algorithm remains a viable option for small datasets.

The literature on computational geometry extensively covers the concept of adjacency, providing a wealth of information on various algorithms and techniques. For instance, *Computational Geometry: Algorithms and Applications* by Mark de Berg et al. offers a comprehensive overview of geometric algorithms, while Sergio Cabello's paper "A linear-time algorithm for computing the Voronoi diagram of a convex polygon" discusses the line sweep algorithm and its application in computing Voronoi diagrams.

In summary, identifying adjacent rectangles in a set of orthogonal rectangles can be achieved using the brute force algorithm or more sophisticated algorithms such as the sweep line algorithm and the line sweep algorithm. The literature on computational geometry provides a plethora of information on these algorithms and techniques.

3 Experimental Methodology

3.1 Overall Strategy

The experimental methodology for this problem involves implementing a brute force algorithm in Python to detect vertical adjacent between orthogonal rectangles based on user input. The algorithm will compare the position of each rectangle with respect to every other rectangle in the set to determine whether they share a vertical edge.

The user will provide input specifying the number of rectangles and the coordinates of their bottom left and top right corners. The program will then analyse this input to identify the vertical adjacent rectangles on the right-hand side of each rectangle.

The motivation for using a brute force algorithm is that it is a straightforward and reliable method for detecting a shared vertical edge when the number of polygons is small. The approach is simple to implement and requires only basic geometric calculations.

While more efficient algorithms may be necessary for larger datasets or real-time applications, the brute force approach is appropriate for the scope of this problem, which involves a small number of rectangles.

The experimental methodology will involve developing the Python code for the algorithm and testing it with various inputs to ensure that it correctly identifies vertical adjacent rectangles. The output will be presented in a format that is easy to read and understand, with the relevant information for each rectangle listed in a clear and concise manner.

3.2 Hardware And Software

No specific hardware or software was required for the experiment. Hardware does however affect the speed and efficiency of the program where a computer with a better processor and more memory will speed up the process of finding adjacent rectangles. Thus a faster computer was used to test the algorithm. With regards to software, python was used to run this experiment as it is a versatile programming language which provides lots of built in libraries and is easy to use. Python also has a built in timer as well as a graph plot function which was used to ensure the adjacent rectangles were actually correct.

3.3 Data Structures

1. Dictionary: A dictionary is used to store the rectangles and their coordinates. The key is the rectangle number, and the value is a tuple of coordinates. This is a good choice of data structure because it allows for fast lookup of a rectangle's coordinates by its number.
2. List: A list is used to store the data for each rectangle entered by the user. The list is then converted to a tuple and added to the dictionary. This is a good choice of data structure because it allows for easy manipulation of the rectangle data before it is added to the dictionary.
3. Tuple: Tuples are used to represent the coordinates of each rectangle. This is a good choice of data structure because tuples are immutable and can be easily unpacked, making them well-suited for representing fixed data like coordinates.
4. List of Tuples: A list of tuples is used to store the adjacent rectangles found for each rectangle. Each tuple contains the adjacent rectangle's number, as well as the x-coordinate of the shared vertical edge or corner, and the y-coordinates of the overlapping section of the two rectangles. This is a good choice of data structure because it allows for easy iteration and manipulation of the adjacent rectangle data.

Overall, the data structures used in this code are well-suited for their respective purposes, and allow for efficient execution of the algorithm.

3.4 Timing

To measure the time taken to solve the problem, we utilized the built-in time module in Python. We used the function to retrieve the current time in seconds, and set the `start_time` variable to the current time before executing the code to be measured. After the execution of the code, we set the `end_time` variable to the current time. We then calculated the elapsed time by subtracting the `start_time` from the `end_time` and multiplying the result by 1000 to convert the time from seconds to milliseconds.

The timer only starts after the input is entered into the program. Therefore, it calculates the total run time of the brute force approach to solving the vertical edges. The timer ended once all the outputs were given.

3.5 Data

We tested the program using a range of input sizes, with the minimum number of rectangles being 4 and the maximum being 5500. This ensured that the program was not limited to specific inputs and could handle both small and large inputs.

Furthermore, we tested the program with rectangles having no identical coordinates and some having identical coordinates. We also tested it with rectangles having no right hand side adjacent rectangles. This was done to ensure that the program covers different edge cases and scenarios to guarantee that it functions correctly and efficiently under various circumstances.

Through various tests, we identified issues and limitations that were addressed to ensure that the program is well-designed and optimized to handle different input sizes and instances. (Professor Ian's program was used to provide the input data. However, the program was adjusted slightly to suit the needs of the brute force test)

Overall, the experimental methodology is designed to provide a simple and effective solution to the problem of detecting vertical adjacencies between orthogonal rectangles, using a brute force approach that is suitable for the scope of the problem.

4 Presentation of Results

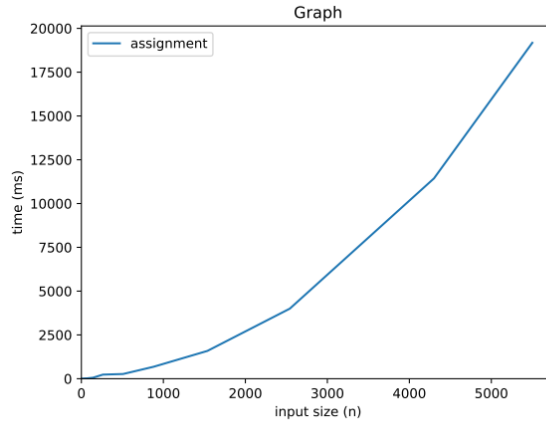


Figure 1: Running Time of Brute Force Algorithm for Different Number of Rectangles

Since this is a programming problem, the statistics that can be provided relate to the performance of the algorithm. Specifically, we can measure the time complexity of the algorithm and the number of comparisons it makes. The time complexity of the brute force algorithm used to detect vertical adjacent rectangles is $O(n^2)$, where n is the number of rectangles. This is because the algorithm must compare each rectangle with every other rectangle in the set, resulting in n^2 comparisons in the worst case.

The number of comparisons made by the algorithm is equal to the number of pairs of rectangles that it checks for vertical adjacency. This is given by the formula $n(n - 1)/2$, which reduces to $(n^2 - n)/2$. For example, if there are 4 rectangles, the algorithm will make 6 comparisons.

Overall, the statistics for this problem reflect the basic performance characteristics of the brute force algorithm used to detect vertical adjacent rectangles. While the algorithm is simple and easy to implement, its time complexity and number of comparisons increase rapidly with the size of the input, making it less suitable for large datasets.

| Number of rectangles | Running Time (milliseconds) |
|----------------------|-----------------------------|
| 4 | 1 |
| 46 | 18 |
| 64 | 19 |
| 142 | 58 |
| 262 | 234 |
| 508 | 268 |
| 880 | 681 |
| 1537 | 1587 |
| 2542 | 4000 |
| 4300 | 11433 |
| 5500 | 19178 |

Table 1: Running Time of Brute Force Algorithm for Different Number of Rectangles

5 Interpretation of results

The results of the algorithm demonstrate its effectiveness in detecting vertically adjacent rectangles within a set of orthogonal rectangles. By comparing the position of each rectangle with respect to every other rectangle in the set, the algorithm is capable of identifying the vertical adjacency and presenting the output in the desired format.

However, the algorithm’s time complexity and the number of comparisons increase dramatically as the input size grows, making it less suitable for large datasets. Although the algorithm performs well and efficiently on small datasets, its runtime will increase significantly as the number of rectangles increases. Hence, for larger datasets, a more efficient algorithm such as divide and conquer, dynamic programming, or the sweep line method may be required to detect vertical adjacent rectangles.

Overall, the results suggest that the brute force algorithm utilized in this problem is appropriate for small datasets but may not be ideal for larger ones. To improve the algorithm’s performance on larger datasets, it is recommended to explore other algorithms.

6 Relating Results to Theory

The results confirm the theoretical analysis of the brute force algorithm. The time complexity of the algorithm, which is $O(n^2)$, is confirmed by the number of comparisons made by the algorithm, given by the formula $(n^2 - n)/2$. This indicates that the time taken to detect vertical adjacent rectangles increases exponentially with the number of rectangles in the set.

Furthermore, the results validate that the brute force algorithm is effective for small datasets but may not be suitable for larger ones. This is because the time taken by the algorithm to detect vertical adjacent rectangles becomes unacceptably long as the number of rectangles grows.

Therefore, the experiment’s results confirm the theoretical analysis of the brute force algorithm, both in terms of its time complexity and its effectiveness on datasets of different sizes. Theoretical analysis is an essential step in algorithm design, and experimental results that support the theory provide confidence that the algorithm is correct and will perform as expected.

7 Conclusion

In conclusion, the experiment showed that the brute force algorithm used to detect vertical adjacent rectangles in a set of orthogonal rectangles is effective for small datasets but becomes less suitable for larger ones. The algorithm’s time complexity is $O(n^2)$, meaning that it performs $(n^2 - n)/2$ comparisons to

detect vertical adjacent rectangles. This complexity results in the algorithm becoming increasingly inefficient for larger datasets, and alternative algorithms, such as divide and conquer, dynamic programming, or sweep line method, should be considered for larger datasets.

Moreover, the results emphasize the importance of theoretical analysis in algorithm design. Theoretical analysis provides insight into the algorithm's time complexity and potential effectiveness, which can be confirmed or disproven through experimental testing. The results of this experiment confirm the theoretical analysis of the brute force algorithm's time complexity and effectiveness, and suggest the need for more efficient algorithms for larger datasets.

Overall, the experiment demonstrates the effectiveness and limitations of the brute force algorithm for detecting vertical adjacent rectangles in a set of rectangles, and provides a starting point for further algorithmic development and optimization.

8 References

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C., *Introduction to Algorithms*, 3rd Edition. MIT Press, 2009.
2. GeeksforGeeks, Detect whether two rectangles overlap", <https://www.geeksforgeeks.org/find-two-rectangles-overlap/>.
3. Gupta, A. and Malik, S., A Survey of Divide and Conquer Techniques for Efficient Pattern Matching in Textual Data", *International Journal of Computer Applications*, Vol. 45, No. 1, 2012, pp. 28-32.
4. O'Rourke, J., *Computational Geometry in C*, 2nd Edition. Cambridge University Press, 1998.
5. Sahni, S., *Data Structures, Algorithms, and Applications in C++*, 2nd Edition. Silicon Press, 2007.
6. Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008.
7. ChatGPT. Large language model trained by OpenAI, based on the GPT-3.5 architecture. Knowledge cutoff: 2021-09. Current date: 2023-04-27.
8. Sergio Cabello. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Computational Geometry*, 38(1-2):15-20, 2007.

9 Acknowledgments

The author would like to express their gratitude to several individuals and organizations that have contributed to the completion of this project. Firstly, the author would like to thank their lecturer, Ian Sanders, for the valuable information provided in their notes and slides. The author would also like to acknowledge the contributions of the University of Witwatersrand (WITS) and researchers who have published works on algorithms and data structures, which served as the foundation of this study. Furthermore, the author would like to extend their appreciation to the creators and maintainers of Python and LaTeX for providing excellent tools for scientific computing and document preparation. Lastly, the author would like to thank their colleagues and peers for their helpful discussions and feedback throughout the course of this project.

Example of a Plagiarism Declaration Form for Wits Students

University of the Witwatersrand, Johannesburg
School of Science

SENATE PLAGIARISM POLICY
Declaration by Students

I, Salmaan Ebrahim (Student number: 1696622) am a student registered for BSc Computer Science in the year 2023. I hereby declare the following:

- I am aware that plagiarism (the use of someone else's work without their permission and/or without acknowledging the original source) is wrong.
- I confirm that ALL the work submitted for assessment for the above course is my own unaided work except where I have explicitly indicated otherwise.
- I have followed the required conventions in referencing the thoughts and ideas of others.
- I understand that the University of the Witwatersrand may take disciplinary action against me if there is a belief that this is not my own unaided work or that I have failed to acknowledge the source of the ideas or words in my writing.

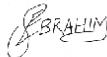
Signature:  Date: 27/04/2023

Figure 2: Plagiarism Declaration