COT 5405 – Analysis of Algorithms

Programming Project 1 – Greedy Algorithms

27th Feb 2023

Shreya Shah

Siju Sakaria

# **Team members**

1. Shreya Shah (UFID: )

Worked on design and analysis of Greedy strategies 1 and 3, Bonus, generated Makefile, conducted experimental study and worked on report

1. Siju Sakaria (UFID: 1015-4248)

Worked on design and analysis of Greedy strategies 2 and 4, plotted graphs, added test cases, tested on Thunder server and worked on report

# **Design and Analysis of Algorithms**

**Greedy Strategy 1**

**Design**

*Initialize an empty list to keep track of the painted houses.*

*Keep track of house index*

*For day = 1 to n:*

*Initialize a list of available houses on the current day.*

*while i = house\_index to m and current day > end day:*

*Increment house\_index*

*If day lies between start and end day.*

*Add the painted house index to painted\_houses.*

*Return painted\_houses*

**Time complexity**

Since the input is sorted according to start date and we run the code for n days, the time complexity is O(n)

**Space complexity**

The space complexity of the algorithm is O(m), where m is the number of houses painted.

**Example**

1. N = 10, m = 5, houses = [(0, 1), (2, 4), (4, 6), (5, 7), (8, 10)]

Here, the output would be 0 1 2 3 4.

We will be able to paint all the houses available since there is no overlap of house availability.

1. N = 18, m = 18, houses = [ (1, 2), (1, 5), (1, 5), (1, 6), (2, 3), (2, 6), (2, 7), (3, 4), (4, 5), (5, 9), (5, 11), (6, 8), (6, 12), (8, 12), (9, 14), (12, 15), (16, 17)]

Here, Strategy 1 will be able to paint only 14 houses with index 0 1 2 3 4 6 7 10 11 13 14 15 16 17. Strategy 4 will paint more houses than this.

**Greedy Strategy 2**

**Design**

*Initialize an empty list to keep track of the painted houses.*

*Keep track of house index*

*For day = 1 to n:*

*Initialize a list of available houses on the current day.*

*while i = house\_index to m and current day lies between start and end day:*

*Maintain max heap(available\_houses) of unpainted house by start date*

*Increment heap index*

*If available\_houses*

*Pop the house from heap*

*If day lies between start and end day.*

*Add the painted house index to painted\_houses.*

*Return painted\_houses*

**Time complexity**

Since we run a for loop for n days and we use a priority queue to maintain a max heap, the time complexity is O(n + m log(m))

**Space complexity**

The space complexity of the algorithm is O(m), where m is the number of houses painted.

**Example**

1. N = 10, m = 5, houses = [(0, 1), (2, 4), (4, 6), (5, 7), (8, 10)]

Here, the output would be 0 1 2 3 4.

We will be able to paint all the houses available since there is no overlap of house availability.

1. N = 18, m = 18, houses = [ (1, 2), (1, 5), (1, 5), (1, 6), (2, 3), (2, 6), (2, 7), (3, 4), (4, 5), (5, 9), (5, 11), (6, 8), (6, 12), (8, 12), (9, 14), (12, 15), (16, 17)]

Here, Strategy 2 will be able to paint only 12 houses with index 0 5 8 9 10 12 13 14 15 11 16 17. Strategy 4 will paint more houses than this.

**Greedy Strategy 3**

**Design**

*Initialize an empty list to keep track of the painted houses.*

*Keep track of house index*

*For day = 1 to n:*

*Initialize a list of available houses on the current day.*

*while i = house\_index to m and current day lies between start and end day:*

*Maintain min heap(available\_houses) of unpainted house by shortest duration*

*Increment heap index*

*If available\_houses*

*Pop the house from heap*

*If day lies between start and end day.*

*Add the painted house index to painted\_houses.*

*Return painted\_houses*

**Time complexity**

Since we run a for loop for n days and we use a priority queue to maintain a min heap, the time complexity is O(n + m log(m)).

**Space complexity**

The space complexity of the algorithm is O(m), where m is the number of houses painted.

**Example**

1. N = 10, m = 5, houses = [(0, 1), (2, 4), (4, 6), (5, 7), (8, 10)]

Here, the output would be 0 1 2 3 4.

We will be able to paint all the houses available since there is no overlap of house availability.

1. N = 18, m = 18, houses = [ (1, 2), (1, 5), (1, 5), (1, 6), (2, 3), (2, 6), (2, 7), (3, 4), (4, 5), (5, 9), (5, 11), (6, 8), (6, 12), (8, 12), (9, 14), (12, 15), (16, 17)]

Here, Strategy 3 will be able to paint only 9 houses with index 0 1 5 8 9 12 14 16 17. Strategy 4 will paint more houses than this.

**Greedy Strategy 4**

**Design**

*Initialize an empty list to keep track of the painted houses.*

*Keep track of house index*

*For day = 1 to n:*

*Initialize a list of available houses on the current day.*

*while i = house\_index to m and current day lies between start and end day:*

*Maintain min heap(available\_houses) of unpainted house by end date*

*Increment heap index*

*If available\_houses*

*Pop the house from heap*

*If day lies between start and end day.*

*Add the painted house index to painted\_houses.*

*Return painted\_houses*

**Time complexity**

Since we run a for loop for n days and we use a priority queue to maintain a min heap, the time complexity is O(n + m log(m))

**Space complexity**

The space complexity of the algorithm is O(m), where m is the number of houses painted.

**Example**

1. N = 10, m = 5, houses = [(0, 1), (2, 4), (4, 6), (5, 7), (8, 10)]

Here, the output would be 0 1 2 3 4.

We will be able to paint all the houses available since there is no overlap of house availability.

1. N = 18, m = 18, houses = [ (1, 2), (1, 5), (1, 5), (1, 6), (2, 3), (2, 6), (2, 7), (3, 4), (4, 5), (5, 9), (5, 11), (6, 8), (6, 12), (8, 12), (9, 14), (12, 15), (16, 17)]

Here, Strategy 4 will be able to paint 15 houses with index 0 1 5 8 2 4 7 12 10 11 13 14 15 16 17. This is the maximum houses painted by any Strategy and hence, it is the most optimal solution.

**Proof of correctness**

**Experimental Comparative Study**

**Bonus**

**Design**

*Initialize an empty list to keep track of the painted houses.*

*Initialize an empty list to keep track of the available houses*

*Keep track of house index*

*While house\_index < num of houses or available\_houses:*

*If no available houses:*

*If start day < 1: set day to 1*

*Else set day to start day*

*while i = house\_index to m and start day <= current day:*

*Maintain min heap(available\_houses) of unpainted house by end date*

*Increment heap index*

*If available\_houses*

*Pop the house from heap*

*If day lies between start and end day.*

*Add the painted house index to painted\_houses.*

*Return painted\_houses*

**Time complexity**

Since we no longer run a loop for n days and just maintain a priority queue to maintain a min heap, the time complexity is O(m log(m))

**Space complexity**

The space complexity of the algorithm is O(m), where m is the number of houses painted.

**Example**

1. N = 10, m = 5, houses = [(0, 1), (2, 4), (4, 6), (5, 7), (8, 10)]

Here, the output would be 0 1 2 3 4.

We will be able to paint all the houses available since there is no overlap of house availability.

1. N = 18, m = 18, houses = [ (1, 2), (1, 5), (1, 5), (1, 6), (2, 3), (2, 6), (2, 7), (3, 4), (4, 5), (5, 9), (5, 11), (6, 8), (6, 12), (8, 12), (9, 14), (12, 15), (16, 17)]

**Proof of correctness**

**Conclusion**

To summarize on the learning experience, we both learned a lot while working on this project. One of us is proficient in Java while other is proficient in C++, so we chose a middle ground and decided to choose Python as our main language to allow both of us to learn and benefit from the process. While implementing the greedy strategies, Strategy 1 was straight forward to implement while other strategies required us to do a bit of research on how to use priority queue in Python. Also, it took a bit of research to figure out implementing a max heap for Strategy 2 since max heap is implemented in a different way in Python compared to other programming languages. Also, it took us a considerable amount of time to come up with test cases which would help us decide on an optimal solution.