Greedy Lab Tutorial

Kousshik Raj

September 24, 2021

Problem

- There are $N (2 \leq N \leq 8)$ customers coming to pick up their orders from a shop on a particular day. In that day, which consists of M minutes, each customer has a specific time frame which they are available in, denoted by the minutes range $[l_i, r_i], \forall\ 1 \leq i \leq N$, $0 \leq l_i \leq r_i < M (M \leq 10^5)$.

Problem

- There are $N (2 \leq N \leq 8)$ customers coming to pick up their orders from a shop on a particular day. In that day, which consists of M minutes, each customer has a specific time frame which they are available in, denoted by the minutes range $[l_i, r_i], \forall\ 1 \leq i \leq N$, $0 \leq l_i \leq r_i < M (M \leq 10^5)$.

- Each customer comes and gets their order at minute x_i $(l_i \leq x_i \leq r_i)$. Assume x_i is an integer. But due to Covid, we need to make sure that the minimum difference in the time two consecutive customers (in terms of time) come and get their order is as large as possible.

Problem

- There are $N (2 \leq N \leq 8)$ customers coming to pick up their orders from a shop on a particular day. In that day, which consists of M minutes, each customer has a specific time frame which they are available in, denoted by the minutes range $[l_i, r_i], \forall\ 1 \leq i \leq N$, $0 \leq l_i \leq r_i < M (M \leq 10^5)$.
- Each customer comes and gets their order at minute x_i $(l_i \leq x_i \leq r_i)$. Assume x_i is an integer. But due to Covid, we need to make sure that the minimum difference in the time two consecutive customers (in terms of time) come and get their order is as large as possible.
- Design an efficient algorithm to determine the ideal time each customer should come and get their order so that the above condition is satisfied. Also output the minimum time difference between any two customers.

Problem

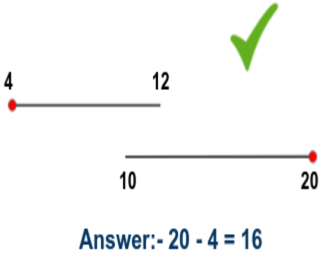
Formally...

You are given N ranges of the form $[l_i, r_i]$, $0 \le l_i \le r_i < M$. Design an efficient algorithm to output N integers x_i, $l_i \le x_i \le r_i$, $\forall 1 \le i \le N$, such that $\min_{1 \le i < j \le N} |x_i - x_j|$ is the maximum possible. Also print $\min_{1 \le i < j \le N} |x_i - x_j|$ for your timings.

Problem

Formally...

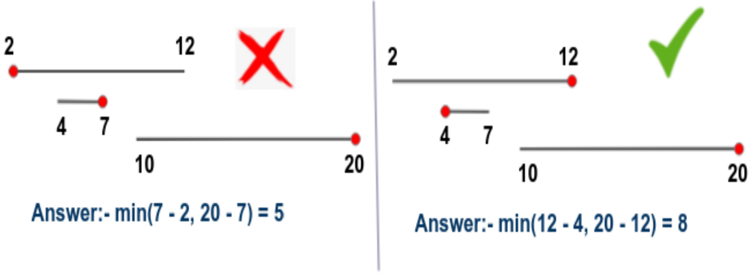
You are given N ranges of the form $[l_i, r_i]$, $0 \le l_i \le r_i < M$. Design an efficient algorithm to output N integers x_i, $l_i \le x_i \le r_i$, $\forall 1 \le i \le N$, such that $\min_{1\le i<j\le N} |x_i - x_j|$ is the maximum possible. Also print $\min_{1\le i<j\le N} |x_i - x_j|$ for your timings.



Problem

Formally...

You are given N ranges of the form $[l_i, r_i]$, $0 \le l_i \le r_i < M$. Design an efficient algorithm to output N integers x_i, $l_i \le x_i \le r_i$, $\forall 1 \le i \le N$, such that $\min\limits_{1 \le i < j \le N} |x_i - x_j|$ is the maximum possible. Also print $\min\limits_{1 \le i < j \le N} |x_i - x_j|$ for your timings.



Approach (Hints)

- What is one of the primary things involved with a greedy algorithm? The ordering of the elements so that a greedy choice is part of the optimal solution.

Approach (Hints)

- What is one of the primary things involved with a greedy algorithm? The ordering of the elements so that a greedy choice is part of the optimal solution.

- If you are given that the minimum difference is L, can you find the timings when the customers should come and get their orders, or determine if it is not possible?

Approach (Hints)

- What is one of the primary things involved with a greedy algorithm? The ordering of the elements so that a greedy choice is part of the optimal solution.
- If you are given that the minimum difference is L, can you find the timings when the customers should come and get their orders, or determine if it is not possible?
- What is the most efficient way to determine L, given that you are able to solve the above two points? Think of something sublinear.

The Order

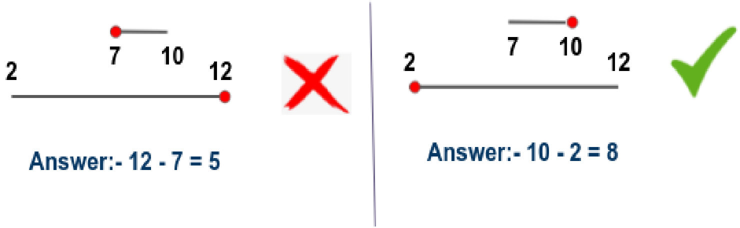
- What is the order of customers in the optimal strategy? Try some common approaches.

The Order

- What is the order of customers in the optimal strategy? Try some common approaches.
    - Sort by the size of the interval?

The Order

- What is the order of customers in the optimal strategy? Try some common approaches.
  - Sort by the size of the interval?

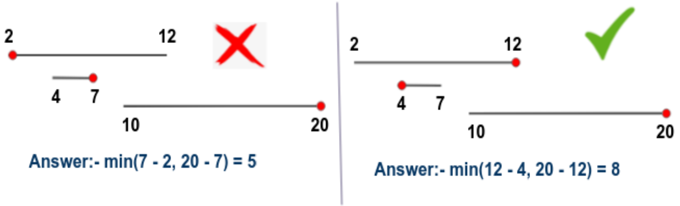


The Order

- What is the order of customers in the optimal strategy? Try some common approaches.
  - Sort by the size of the interval?
  - Sort by the starting point?

The Order

- What is the order of customers in the optimal strategy? Try some common approaches.
  - Sort by the size of the interval?
  - Sort by the starting point?

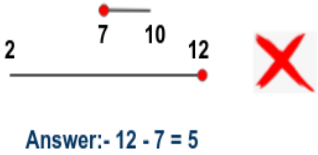


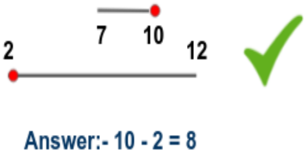
The Order

- What is the order of customers in the optimal strategy? Try some common approaches.
  - Sort by the size of the interval?
  - Sort by the starting point?
  - Sort by the ending point?

The Order

- What is the order of customers in the optimal strategy? Try some common approaces.
  - Sort by the size of the interval?
  - Sort by the starting point?
  - Sort by the ending point?





The Order

- What is the order of customers in the optimal strategy? Try some common approaches.
  - Sort by the size of the interval?
  - Sort by the starting point?
  - Sort by the ending point?

- There is no particular order that works for the optimal strategy. We have to enumerate all possible order the customers can come in.

The Order

- What is the order of customers in the optimal strategy? Try some common approaches.
  - Sort by the size of the interval?
  - Sort by the starting point?
  - Sort by the ending point?
- There is no particular order that works for the optimal strategy. We have to enumerate all possible order the customers can come in.
- This is feasible, as N is small enough to accomodate $O(N.N!)$ algorithm.

The Greedy Part

- You have an order P (P_i denotes the original index of the i^{th} customer in terms of timing), and minimum difference between any customer L. Check whether this is possible.

The Greedy Part

- You have an order P (P_i denotes the original index of the i^{th} customer in terms of timing), and minimum difference between any customer L. Check whether this is possible.

- Choose $x_{P_i} = \max(l_{P_i}, x_{P_{i-1}} + L)$. If $x_{P_i} > r_{P_i}$ we can say that this combination is not possible. Complexity is $O(N)$.

The Greedy Part

- You have an order P (P_i denotes the original index of the ith customer in terms of timing), and minimum difference between any customer L. Check whether this is possible.
- Choose $x_{P_i} = \max(l_{P_i}, x_{P_{i-1}} + L)$. If $x_{P_i} > r_{P_i}$ we can say that this combination is not possible. Complexity is $O(N)$.
- **Intuitively:-** We are trying to fit in the next customer as early as possible. This will ensure best compatibility for later customers.

The Greedy Part

- You have an order P (P_i denotes the original index of the i$^{\text{th}}$ customer in terms of timing), and minimum difference between any customer L. Check whether this is possible.
- Choose $x_{P_i} = \max(l_{P_i}, x_{P_{i-1}} + L)$. If $x_{P_i} > r_{P_i}$ we can say that this combination is not possible. Complexity is $O(N)$.
- **Intuitively:-** We are trying to fit in the next customer as early as possible. This will ensure best compatibility for later customers.
- **Formal Proof:-** Assume you have an optimal solution $\{y_i\}_{i=1}^{N}$, not necessarily matching our algo. We can say that $y_i \geq x_i, \ \forall 1 \leq i \leq N$ (induction on the order of customers). Find the first i such $y_{P_i} > x_{P_i}$. Changing y_{P_i} to x_{P_i} will not make a feasible solution infeasible. Repeat until $y_i = x_i, \ \forall 1 \leq i \leq N$.

Examples

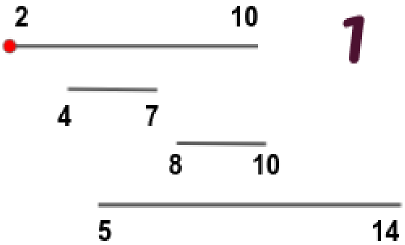
Example 1

There are 4 customers. Their available timings are $[8, 10], [4, 7], [2, 10]$, $[5, 14]$, respectively. You want to check whether a minimum difference of $L = 4$ is possible for the order $P = [3, 2, 1, 4]$.

Examples

Example 1

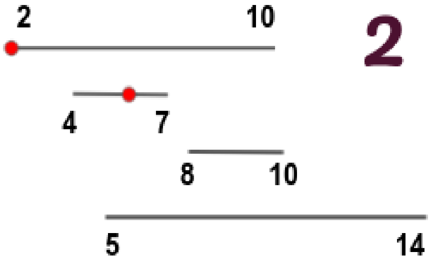
There are 4 customers. Their available timings are $[8, 10], [4, 7], [2, 10]$, $[5, 14]$, respectively. You want to check whether a minimum difference of $L = 4$ is possible for the order $P = [3, 2, 1, 4]$.



Examples

Example 1

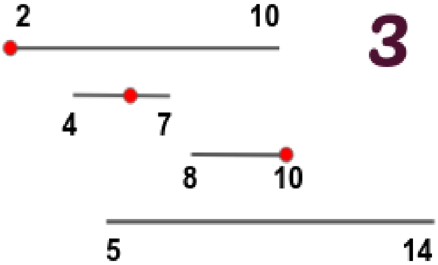
There are 4 customers. Their available timings are $[8, 10]$, $[4, 7]$, $[2, 10]$, $[5, 14]$, respectively. You want to check whether a minimum difference of $L = 4$ is possible for the order $P = [3, 2, 1, 4]$.



Examples

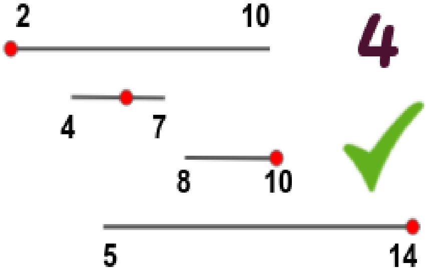
Example 1

There are 4 customers. Their available timings are $[8, 10], [4, 7], [2, 10]$, $[5, 14]$, respectively. You want to check whether a minimum difference of $L = 4$ is possible for the order $P = [3, 2, 1, 4]$.



Example 1

There are 4 customers. Their available timings are $[8, 10]$, $[4, 7]$, $[2, 10]$, $[5, 14]$, respectively. You want to check whether a minimum difference of $L = 4$ is possible for the order $P = [3, 2, 1, 4]$.



Examples

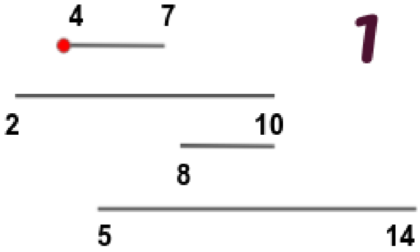
Example 2

There are 4 customers. Their available timings are $[8, 10], [4, 7], [2, 10]$, $[5, 14]$, respectively. You want to check whether a minimum difference of $L = 4$ is possible for the order $P = [2, 3, 1, 4]$.

Examples

Example 2

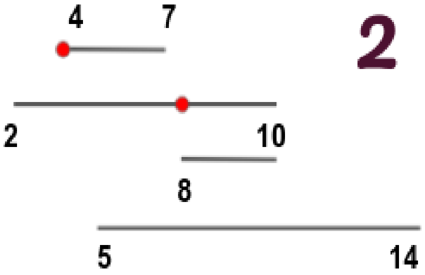
There are 4 customers. Their available timings are $[8, 10], [4, 7], [2, 10]$, $[5, 14]$, respectively. You want to check whether a minimum difference of $L = 4$ is possible for the order $P = [2, 3, 1, 4]$.



Examples

Example 2

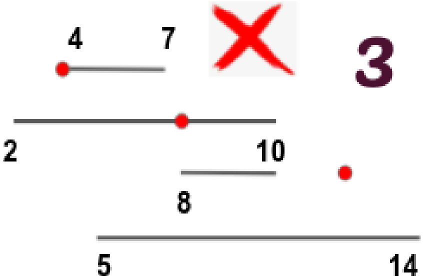
There are 4 customers. Their available timings are $[8, 10], [4, 7], [2, 10], [5, 14]$, respectively. You want to check whether a minimum difference of $L = 4$ is possible for the order $P = [2, 3, 1, 4]$.



Examples

Example 2

There are 4 customers. Their available timings are $[8, 10]$, $[4, 7]$, $[2, 10]$, $[5, 14]$, respectively. You want to check whether a minimum difference of $L = 4$ is possible for the order $P = [2, 3, 1, 4]$.



Efficiently Finding L

- Now you are given only the order P. How do we find L?

Efficiently Finding L

- Now you are given only the order P. How do we find L?
- One way is to try out $L, \forall\ 0 \leq L < M$. Then for each L we can use the above discussed strategy to check feasibility. Then take the maximum. Complexity will be $O^*(L)$.

Efficiently Finding L

- Now you are given only the order P. How do we find L?
- One way is to try out $L, \forall\ 0 \leq L < M$. Then for each L we can use the above discussed strategy to check feasibility. Then take the maximum. Complexity will be $O^*(L)$.
- Can we do better than checking all L in $[0, M)$? Notice that, if it is not feasible for $L = x$, then it is also not feasible for any $L > x$.

Efficiently Finding L

- Now you are given only the order P. How do we find L?
- One way is to try out $L, \forall\ 0 \leq L < M$. Then for each L we can use the above discussed strategy to check feasibility. Then take the maximum. Complexity will be $O^*(L)$.
- Can we do better than checking all L in $[0, M)$? Notice that, if it is not feasible for $L = x$, then it is also not feasible for any $L > x$.
- Imagine an array A, where $A_i = isPossible(i), \forall\ 0 \leq i < M$. *isPossible(i)* returns 1 if there exists timing configuration where the minimum difference is i, else it returns 0. This array will be monotonous (sorted descendingly). Now you have to find the index of the last 1 (or first 0).

Efficiently Finding L

- Now you are given only the order P. How do we find L?
- One way is to try out $L, \forall\ 0 \leq L < M$. Then for each L we can use the above discussed strategy to check feasibility. Then take the maximum. Complexity will be $O^*(L)$.
- Can we do better than checking all L in $[0, M)$? Notice that, if it is not feasible for $L = x$, then it is also not feasible for any $L > x$.
- Imagine an array A, where $A_i = isPossible(i), \forall\ 0 \leq i < M$. *isPossible(i)* returns 1 if there exists timing configuration where the minimum difference is i, else it returns 0. This array will be monotonous (sorted descendingly). Now you have to find the index of the last 1 (or first 0).
- We can use binary search for this. Complexity will be $O^*(\log M)$

Order Enumeration

- How do we enumerate all possible permutations? Can use inbuilt function for this - *next_permutation*.

Order Enumeration

- Can also use a simple recursive strategy to generate all permutations at once.

Algorithm 1 *Permutation-Recursive (A, pos)*

if $pos = A$.size **then**
  Save permutation A
  return
for $j = pos$ to A.size **do**
  Swap(A_{pos}, A_j)
  Permutation-Recursive (A, $pos + 1$)
  Swap(A_{pos}, A_j)

Order Enumeration

- But iteratively generating them one by one is the most efficient (same as *next_permutation*)

Algorithm 2 *Permutation-Iterative (A)*

```
pos = −1
for i = 1 to A.size −1 do
    if A_i < A_{i+1} then
        pos = i
if pos = −1 then
    return false
swapPos = A.size
while A_pos > A_swapPos do
    swapPos − = 1
Swap (A_pos, A_swapPos)
Reverse (A[pos + 1:])
return true
```

Solution Summary

- Enumerate all possible permutation of the given customers (as no particular ordering strategy works).

Solution Summary

- Enumerate all possible permutation of the given customers (as no particular ordering strategy works).
- For each permutation, binary search the minimum time difference between any consecutive customers.

Solution Summary

- Enumerate all possible permutation of the given customers (as no particular ordering strategy works).
- For each permutation, binary search the minimum time difference between any consecutive customers.
- For each permutation and a minimum time difference, use a greedy strategy to assign the timings for customers that follow the constraints or determine if it is not possible.

Solution Summary

- Enumerate all possible permutation of the given customers (as no particular ordering strategy works).
- For each permutation, binary search the minimum time difference between any consecutive customers.
- For each permutation and a minimum time difference, use a greedy strategy to assign the timings for customers that follow the constraints or determine if it is not possible.
- Then take the maximum minimum difference obtained across all the permutations.

Solution Summary

- Enumerate all possible permutation of the given customers (as no particular ordering strategy works).
- For each permutation, binary search the minimum time difference between any consecutive customers.
- For each permutation and a minimum time difference, use a greedy strategy to assign the timings for customers that follow the constraints or determine if it is not possible.
- Then take the maximum minimum difference obtained across all the permutations.
- Enumerating all permutation takes $O(N.N!)$, binary searching for the optimal minimum difference takes $O(\log M)$, and checking feasibility takes $O(N)$. Overall Complexity:- $O(N.N! \log M)$

Some Other Problems

Problem 1

You have to give candies to people. The people can be modelled as an array, which shows their liking to candies. Everyone receives at least 1 candy. Moreover, if two people are next to each other, then the one with the higher liking must get more candies. What is the minimum sum of candies given to the people?

Some Other Problems

Problem 1

You have to give candies to people. The people can be modelled as an array, which shows their liking to candies. Everyone receives at least 1 candy. Moreover, if two people are next to each other, then the one with the higher liking must get more candies. What is the minimum sum of candies given to the people?

- **Hint:-** Classify the array as segments from every minima to maxima (and vice versa) and then try to assign the candies.

Some Other Problems

Problem 1

You have to give candies to people. The people can be modelled as an array, which shows their liking to candies. Everyone receives at least 1 candy. Moreover, if two people are next to each other, then the one with the higher liking must get more candies. What is the minimum sum of candies given to the people?

- **Hint:-** Classify the array as segments from every minima to maxima (and vice versa) and then try to assign the candies.
- **Source:-** HackerRank — Candies

Some Other Problems

Problem 2

The country of Berland can be modelled by a number line with each integer points from 1 to N representing a city. You have to build power plants in cities so that all cities get a power supply. If you build a power plant at a city, every city within a distance of k will have power. You are also given in which cities plants can be built. Find the minimum number of power plants that has to be built (or determine if its not possible).

Some Other Problems

Problem 2

The country of Berland can be modelled by a number line with each integer points from 1 to N representing a city. You have to build power plants in cities so that all cities get a power supply. If you build a power plant at a city, every city within a distance of k will have power. You are also given in which cities plants can be built. Find the minimum number of power plants that has to be built (or determine if its not possible).

- **Hint:-** Moving from left to right, for every city without power plant build a power plant at the farthest city that covers this city.

Some Other Problems

Problem 2

The country of Berland can be modelled by a number line with each integer points from 1 to N representing a city. You have to build power plants in cities so that all cities get a power supply. If you build a power plant at a city, every city within a distance of k will have power. You are also given in which cities plants can be built. Find the minimum number of power plants that has to be built (or determine if its not possible).

- **Hint:-** Moving from left to right, for every city without power plant build a power plant at the farthest city that covers this city.
- **Source:-** HackerRank — Goodland Electricity

Problem 2

The country of Berland can be modelled by a number line with each integer points from 1 to N representing a city. You have to build power plants in cities so that all cities get a power supply. If you build a power plant at a city, every city within a distance of k will have power. You are also given in which cities plants can be built. Find the minimum number of power plants that has to be built (or determine if its not possible).

- **Hint:-** Moving from left to right, for every city without power plant build a power plant at the farthest city that covers this city.
- **Source:-** HackerRank — Goodland Electricity
- **Bonus:-** Try to solve when the k is variable for each power plant.

THANK YOU!