Greedy Algorithms: Celebration Party

Michael Levin

Department of Computer Science and Engineering
University of California, San Diego

Outline

1. Celebration Party Problem
2. Greedy Algorithm
3. Implementation and Analysis

Many children came to a celebration. Organize them into the minimum possible number of groups such that the age of any two children in the same group differs by at most two years.

Naive Algorithm

- Try all possible distributions of children into one or more groups

Naive Algorithm

- Try all possible distributions of children into one or more groups
- For each distribution, check whether any two children in any group differ by at most 2 years of age

Naive Algorithm

- Try all possible distributions of children into one or more groups
- For each distribution, check whether any two children in any group differ by at most 2 years of age
- Return the minimum number of groups among valid distributions

Running time

Lemma

The running time of the naive algorithm is at least 2^n, where n is the number of children.

Proof

This algorithm will consider all possible distributions of children into two groups (and many other distributions of children into groups). First of these two groups corresponds to any subset of children, and there are 2^n different subsets.

Asymptotics

- Naive algorithm works in time $\Omega(2^n)$

Asymptotics

- Naive algorithm works in time $\Omega(2^n)$
- For $n = 50$ it is at least

$$2^{50} = 1125899906842624$$

operations!

Asymptotics

- Naive algorithm works in time $\Omega(2^n)$
- For $n = 50$ it is at least

$$2^{50} = 1125899906842624$$

  operations!
- We will improve this significantly

Outline

1. Celebration Party Problem
2. Greedy Algorithm
3. Implementation and Analysis

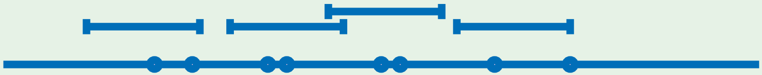
Covering points by segments

Input: A set of n points $x_1, \ldots, x_n \in \mathbb{R}$.

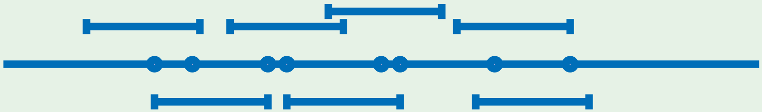
Output: The minimum number of segments of length at most 2 needed to cover all the points.

Example

Example



Example



Connection with Grouping Children

- Points $x_1, \ldots, x_n$ correspond to children' ages

Connection with Grouping Children

- Points $x_1, \ldots, x_n$ correspond to children' ages
- Segments correspond to groups

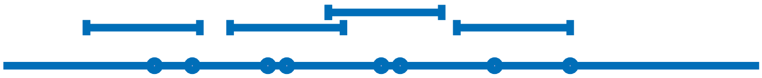
Connection with Grouping Children

- Points $x_1, \ldots, x_n$ correspond to children' ages
- Segments correspond to groups
- Any two children within the same segment of length 2 differ by at most 2 years of age

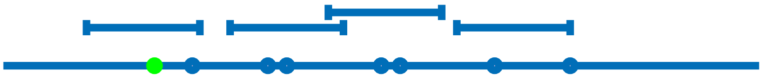
Connection with Grouping Children

- Points $x_1, \ldots, x_n$ correspond to children' ages
- Segments correspond to groups
- Any two children within the same segment of length 2 differ by at most 2 years of age
- Any valid group of children can be put into a segment of length 2

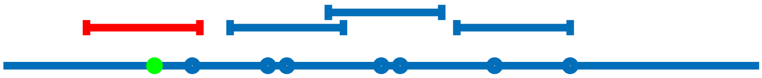
Safe choice: cover the leftmost point with a segment of length 2 which starts in this point.



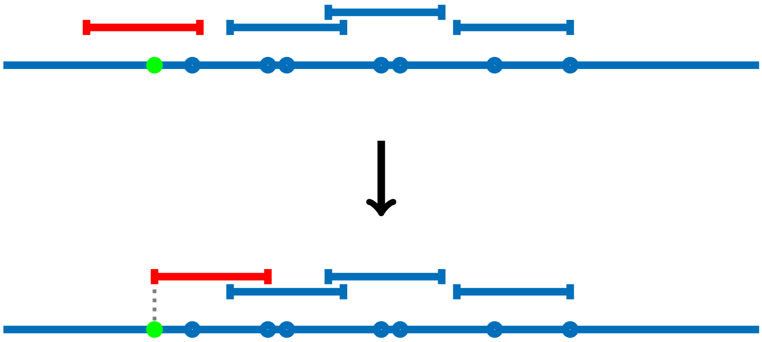
Safe choice: cover the leftmost point with a segment of length 2 which starts in this point.



Safe choice: cover the leftmost point with a segment of length 2 which starts in this point.



Safe choice: cover the leftmost point with a segment of length 2 which starts in this point.



Greedy Algorithm

- Cover the leftmost point with a segment of length 2

Greedy Algorithm

- Cover the leftmost point with a segment of length 2
- Remove all the points within this segment

Greedy Algorithm

- Cover the leftmost point with a segment of length 2
- Remove all the points within this segment
- Solve the same problem with the remaining points

Outline

1. Celebration Party Problem
2. Greedy Algorithm
3. Implementation and Analysis

Assume $x_1 \leq x_2 \leq \ldots \leq x_n$

PointsCoverSorted$(x_1, \ldots, x_n)$

```
segments ← empty list
left ← 1
while left ≤ n:
    (ℓ, r) ← (x_left, x_left + 2)
    segments.append((ℓ, r))
    left ← left + 1
    while left ≤ n and x_left ≤ r:
        left ← left + 1
return segments
```

Assume $x_1 \leq x_2 \leq \ldots \leq x_n$

```
PointsCoverSorted(x_1, ..., x_n)

segments ← empty list
left ← 1
while left ≤ n:
   (ℓ, r) ← (x_left, x_left + 2)
   segments.append((ℓ, r))
   left ← left + 1
   while left ≤ n and x_left ≤ r:
      left ← left + 1
return segments
```

Assume $x_1 \le x_2 \le \ldots \le x_n$

PointsCoverSorted$(x_1, \ldots, x_n)$

```
segments ← empty list
left ← 1
while left ≤ n:
   (ℓ, r) ← (x_left, x_left + 2)
   segments.append((ℓ, r))
   left ← left + 1
   while left ≤ n and x_left ≤ r:
      left ← left + 1
return segments
```

Assume $x_1 \leq x_2 \leq \ldots \leq x_n$

PointsCoverSorted$(x_1, \ldots, x_n)$

```
segments ← empty list
left ← 1
while left ≤ n:
    (ℓ, r) ← (x_left, x_left + 2)
    segments.append((ℓ, r))
    left ← left + 1
    while left ≤ n and x_left ≤ r:
        left ← left + 1
return segments
```

Assume $x_1 \le x_2 \le \ldots \le x_n$

```
PointsCoverSorted(x_1, ..., x_n)

segments ← empty list
left ← 1
while left ≤ n:
   (ℓ, r) ← (x_left, x_left + 2)
   segments.append((ℓ, r))
   left ← left + 1
   while left ≤ n and x_left ≤ r:
      left ← left + 1
return segments
```

Assume $x_1 \leq x_2 \leq \ldots \leq x_n$

PointsCoverSorted($x_1, \ldots, x_n$)

```
segments ← empty list
left ← 1
while left ≤ n:
   (ℓ, r) ← (x_left, x_left + 2)
   segments.append((ℓ, r))
   left ← left + 1
   while left ≤ n and x_left ≤ r:
      left ← left + 1
return segments
```

Assume $x_1 \leq x_2 \leq \ldots \leq x_n$

PointsCoverSorted$(x_1, \ldots, x_n)$

```
segments ← empty list
left ← 1
while left ≤ n:
   (ℓ, r) ← (x_left, x_left + 2)
   segments.append((ℓ, r))
   left ← left + 1
   while left ≤ n and x_left ≤ r:
      left ← left + 1
return segments
```

Assume $x_1 \leq x_2 \leq \ldots \leq x_n$

PointsCoverSorted($x_1, \ldots, x_n$)

```
segments ← empty list
left ← 1
while left ≤ n:
    (ℓ, r) ← (x_left, x_left + 2)
    segments.append((ℓ, r))
    left ← left + 1
    while left ≤ n and x_left ≤ r:
        left ← left + 1
return segments
```

Assume $x_1 \leq x_2 \leq \ldots \leq x_n$

PointsCoverSorted$(x_1, \ldots, x_n)$

```
segments ← empty list
left ← 1
while left ≤ n:
    (ℓ, r) ← (x_left, x_left + 2)
    segments.append((ℓ, r))
    left ← left + 1
    while left ≤ n and x_left ≤ r:
        left ← left + 1
return segments
```

Lemma

The running time of `PointsCoverSorted` is $O(n)$.

Lemma

The running time of `PointsCoverSorted` is $O(n)$.

Proof

- *left* changes from 1 to n

Lemma

The running time of `PointsCoverSorted` is $O(n)$.

Proof

- *left* changes from 1 to n
- For each *left*, append at most 1 new segment to solution

Lemma

The running time of `PointsCoverSorted` is $O(n)$.

Proof

- *left* changes from 1 to n
- For each *left*, append at most 1 new segment to solution
- Overall, running time is $O(n)$ □

Total Running Time

- `PointsCoverSorted` works in $O(n)$ time

Total Running Time

- `PointsCoverSorted` works in $O(n)$ time
- Sort $\{x_1, x_2, \ldots, x_n\}$, then call `PointsCoverSorted`

Total Running Time

- `PointsCoverSorted` works in $O(n)$ time
- Sort $\{x_1, x_2, \ldots, x_n\}$, then call `PointsCoverSorted`
- Soon you'll learn to sort in $O(n \log n)$

Total Running Time

- `PointsCoverSorted` works in $O(n)$ time
- Sort $\{x_1, x_2, \ldots, x_n\}$, then call `PointsCoverSorted`
- Soon you'll learn to sort in $O(n \log n)$
- Sort + `PointsCoverSorted` is $O(n \log n)$

Asymptotics

- Straightforward solution is $\Omega(2^n)$

Asymptotics

- Straightforward solution is $\Omega(2^n)$
- Very long for $n = 50$

Asymptotics

- Straightforward solution is $\Omega(2^n)$
- Very long for $n = 50$
- Sort + greedy is $O(n \log n)$

Asymptotics

- Straightforward solution is $\Omega(2^n)$
- Very long for $n = 50$
- Sort + greedy is $O(n \log n)$
- Fast for $n = 10\,000\,000$

Asymptotics

- Straightforward solution is $\Omega(2^n)$
- Very long for $n = 50$
- Sort + greedy is $O(n \log n)$
- Fast for $n = 10\,000\,000$
- Huge improvement!

Conclusion

- Straightforward solution is exponential
- Important to reformulate the problem in mathematical terms
- Safe choice is to cover leftmost point
- Sort in $O(n \log n)$ + greedy in $O(n)$