# Astronomer Solution Sketch

David R. Lolck

29. April 2023

# Problem

## Problem

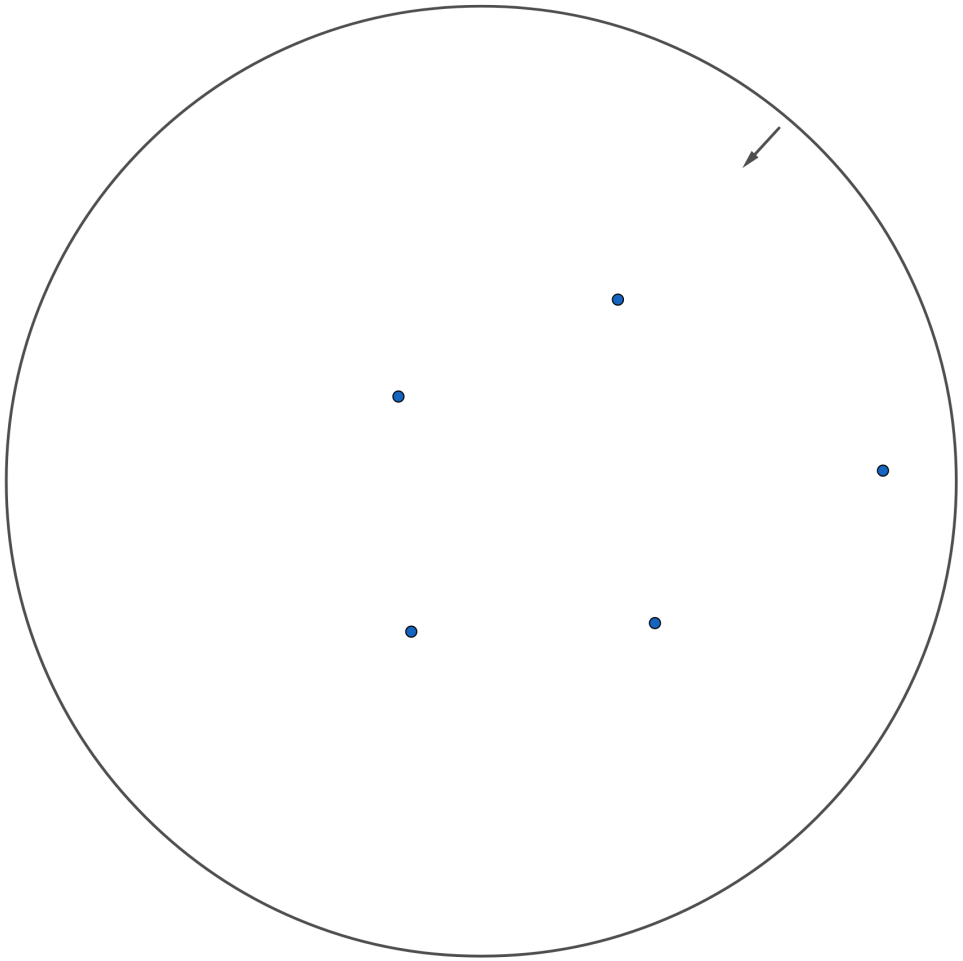
Given $n$ points and integers $k, s, t$, determine the minimum cost circle with center $C$ and radius $r$ that contain $k$ points where the cost is calculated as:

$$\text{cost}(C, r) = s \cdot d((0,0), C) + t \cdot r$$

# Sketch

## Observation 1

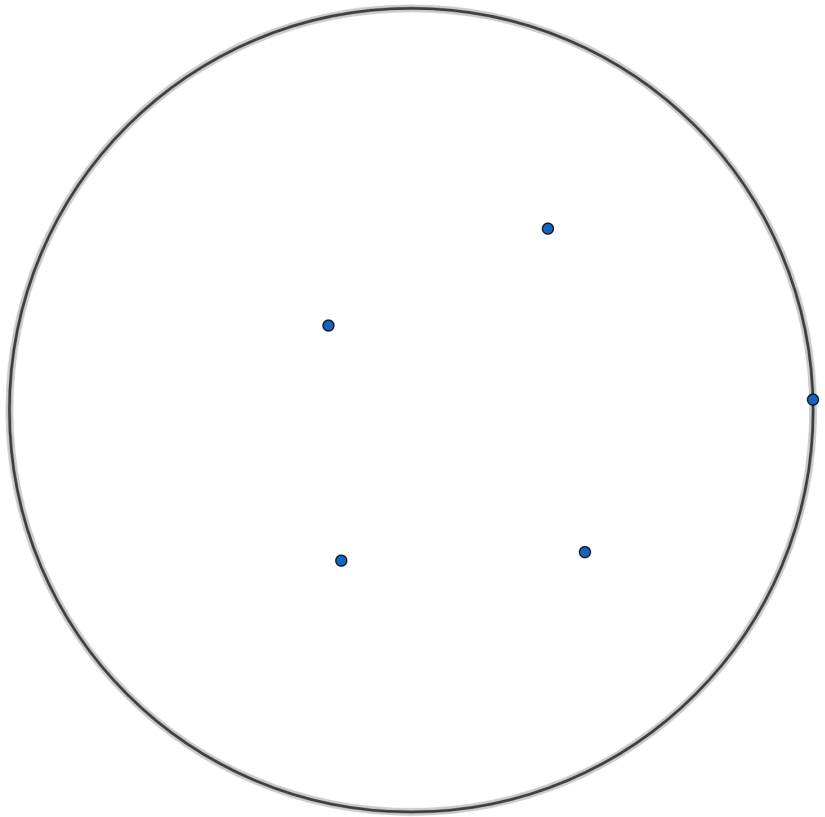
The optimal circle has at least 1 point on the perimeter.



# Sketch

## Observation 1

The optimal circle has at least 1 point on the perimeter.



# Sketch

## Subtask 1: $t \leq s$

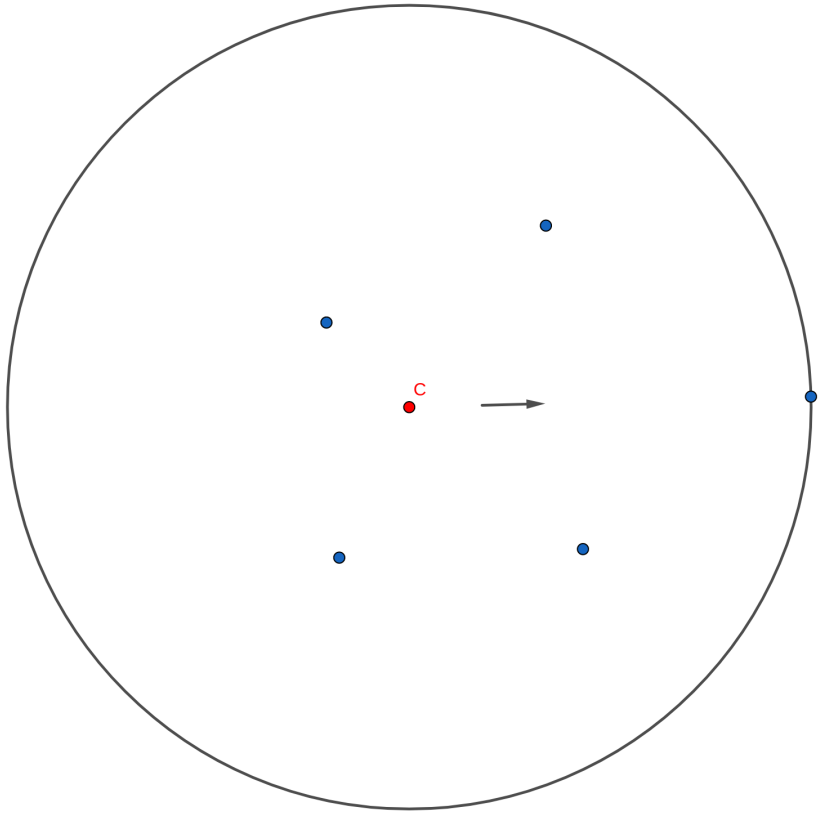
If $t \leq s$, then the solution is the distance to the $k$th closets point times $t$. Running time $O(n \lg n)$ for sorting.

We can therefore assume $t > s$ from this point onward. So increasing the circle is more expensive than moving the center.

# Sketch

## Observation 2

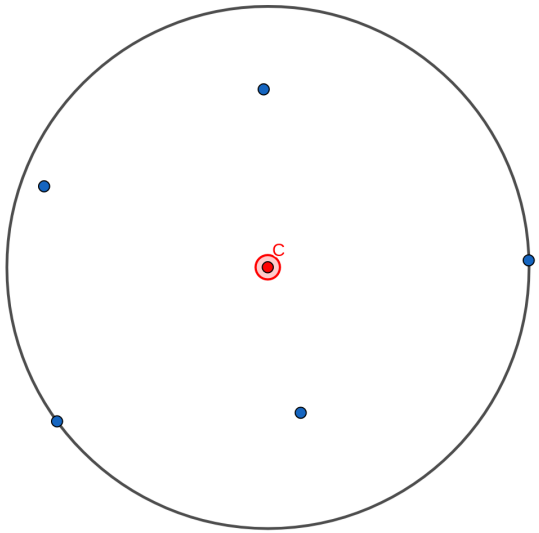
The optimal circle has at least 2 points on the perimeter.



# Sketch

## Observation 2

The optimal circle has at least 2 points on the perimeter.

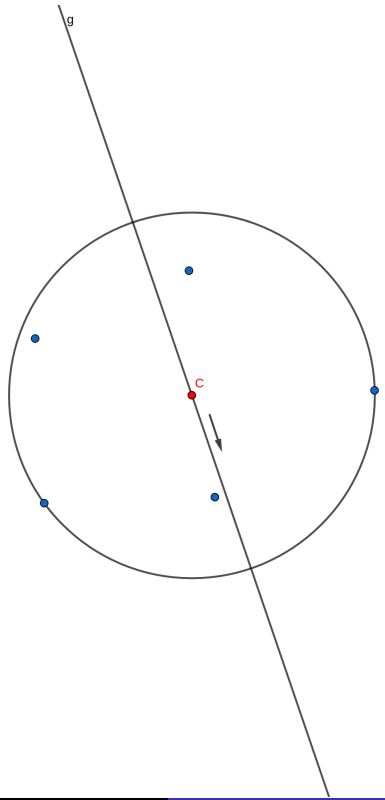


# Sketch

## Observation 3

The optimal circle either:

- Has at least 3 points on the perimeter, or
- is the minimal cost center on some bisector between two points $p$ and $q$, such that $p$ and $q$ lies on the perimeter.

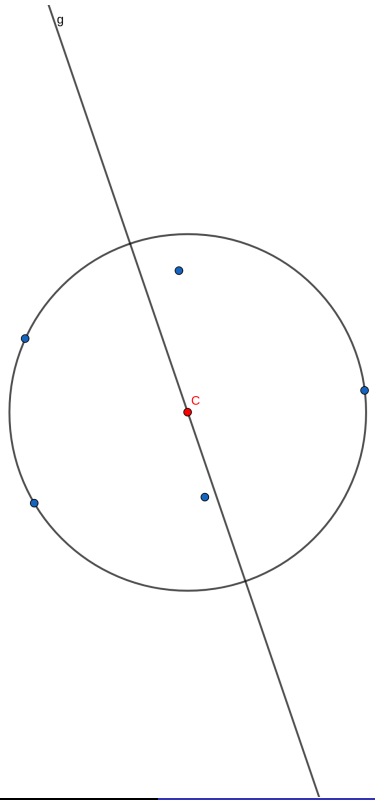


# Sketch

## Observation 3

The optimal circle either:

- Has at least 3 points on the perimeter, or
- is the minimal cost center on some bisector between two points $p$ and $q$, such that $p$ and $q$ lies on the perimeter.



# Sketch

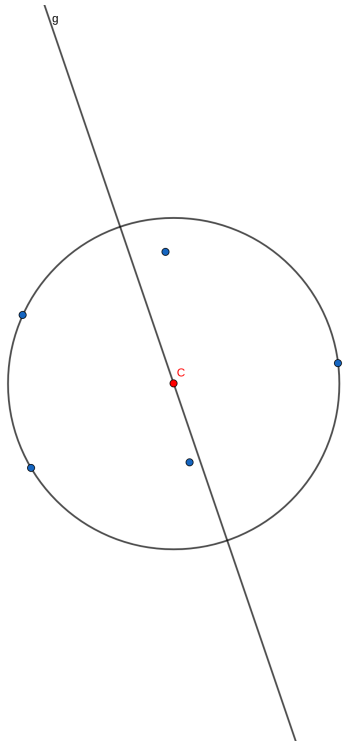
## Subtask 2: $n \leq 50, s = 0$

Check all $O(n^3)$ candidates for centers with 3 points, each in $O(n)$ time. Optimal cost between two points lie directly between them. Check all $O(n^2)$ candidates in $O(n)$ time, for total time $O(n^4)$.

## Subtask 4: $n \leq 50$

As before, but optimal cost between two points is found through ternary search on bisector. Check all $O(n^2)$ candidates in $O(n + \lg \epsilon^{-1})$ time, for total time $O(n^4)$.

# Sketch




## Subtask 5: $n \leq 350$

For every bisector between points $p$ and $q$, for each $r$ of all other points, determine the interval of the bisector where $r$ is contained in a circle with center on the bisector and $p$ and $q$ on the perimeter. Determine all points overlapped by $k$ intervals. Can be done in $O(n^3 \lg n)$ time.

# Sketch

## Idea for $s = 0$

Fix $r$. Determine whether there exists a circle with radius $r$ that contain $k$ points. Use this to binary search $r$.

# Sketch

## Subtask 3: $s = 0$

Fix $r$ and some point $p$. Sweep a circle of radius $r$ around $p$, containing $p$ on the perimeter. For each other point, determine the angle interval where the circle contains $p$. Determine if there exist $k$ overlapping intervals. Binary search $r$ and iterate $p$. Running time $O(n^2 \lg \epsilon^{-1} \lg n)$

# Sketch

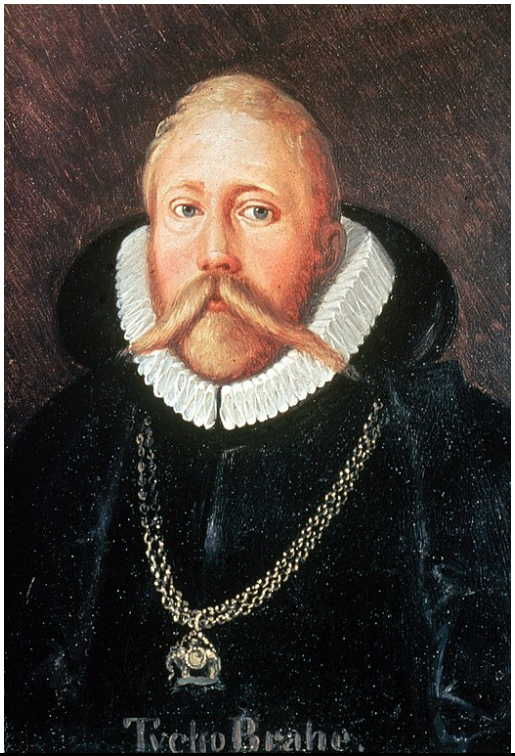
## Subtask 6: $\epsilon = 1/10$

Fix $c$ and some point $p$. Sweep a circle of cost $c$ around $p$, containing $p$ on the perimeter. For each other point, determine the angle interval where the circle contains $p$. Determine if there exist $k$ overlapping intervals. Binary search $c$ and iterate $p$. Gives a running time of $O(n^2 \lg \epsilon^{-2})$.

# Sketch

## Subtask 6: $\epsilon = 1/10$

Fix $c$ and some point $p$. Sweep a circle of cost $c$ around $p$, containing $p$ on the perimeter. For each other point, determine the angle interval where the circle contains $p$. Determine if there exist $k$ overlapping intervals. Binary search $c$ and iterate $p$. Gives a running time of $O(n^2 \lg \epsilon^{-2})$.



# Sketch

## Subtask 6: $\epsilon = 1/10$

Fix $c$ and some point $p$. Sweep a circle of cost $c$ around $p$, containing $p$ on the perimeter. For each other point, determine the angle interval where the circle contains $p$. Determine if there exist $k$ overlapping intervals. Binary search $c$ and iterate $p$. Gives a running time of $O(n^2 \lg \epsilon^{-2})$.

## Subtask 7: *No further constraints*

Same as subtask 6, but observe that we for each point $p$ can determine the best cost that has $p$ on the perimeter and contains $k$ points. They have some ordering. Like starring contest, if we shuffle the points, we only expect to $O(\lg n)$ times observe a lower cost. Gives a running time of $O(n^2 \lg \epsilon^{-1} + n \lg \epsilon^{-2} \lg n)$

# Staring Contest

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $a$ | 32 | 17 | 51 | 52 | 21 |

***Task***: Determine $a = (a_1, \ldots, a_n)$ (distinct integers), from queries of the form $q(i, j) = \min(a_i, a_j)$ with $i \neq j$. Minimize #queries.

***Note***: max $a$ can never be determined, so one underestimate is allowed.

***Note***: Problem is non-adaptive ($a$ fixed from beginning).

***First solution***: Perform all $\binom{n}{2}$ many comparisons:

**for** $i \in \{1, \ldots, n-1\}$
  **for** $j \in \{i+1, \ldots, n\}$
    $Q_{ij} = Q_{ji} = q(i, j)$

| $Q$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | | 17 | 32 | 32 | 21 |
| 2 | 17 | | 17 | 17 | 17 |
| 3 | 32 | 17 | | 51 | 21 |
| 4 | 32 | 17 | 51 | | 21 |
| 5 | 21 | 17 | 21 | 21 | |

# Staring Contest

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $a$ | 32 | 17 | 51 | 52 | 21 |

***Task***: Determine $a = (a_1, \ldots, a_n)$ (distinct integers), from queries of the form $q(i, j) = \min(a_i, a_j)$ with $i \neq j$. Minimize #queries.

***Note***: max $a$ can never be determined, so one underestimate is allowed.

***Note***: Problem is non-adaptive ($a$ fixed from beginning).

***#queries ≤ 3N***:

*Invariant:* Maintain $j$, $k$ such that $a_j$, $a_k$ maximal among $(a_1, \ldots, a_i)$.

Increment $i$, update $j$, $k$ according to $q(i,j)$, $q(i,k)$, $q(j,k)$

| | | | **i** | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| $a$ | 32 | 17 | 51 | 52 | 21 |
| | | **j** | **k** | | |

***#queries ≤ 2N***:

*Observation:* One of the three queries is redundant

# Staring Contest

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $a$ | 32 | 17 | 51 | 52 | 21 |

**Randomised *algorithm*:**

*Idea:* Choose "next index" $i$ randomly, not left to right.

As before, maintain $m_{jk} = \min(a_j, a_k)$

Query $m_{ik} = \min(a_k, a_i)$

🙂: If $\min(a_k, a_i) = m_{jk}$ we've learned $a_k$ and can proceed to the next $i$

☹️: If $\min(a_k, a_i) > m_{jk}$ we must also query $m_{jk} = \min(a_j, a_k)$

**How often does ☹️ happen?**

Exactly if $a_i$ is largest or next-largest among $(a_1, \ldots, a_i)$.

Pr (☹️ in round $i$) $= 2/i$

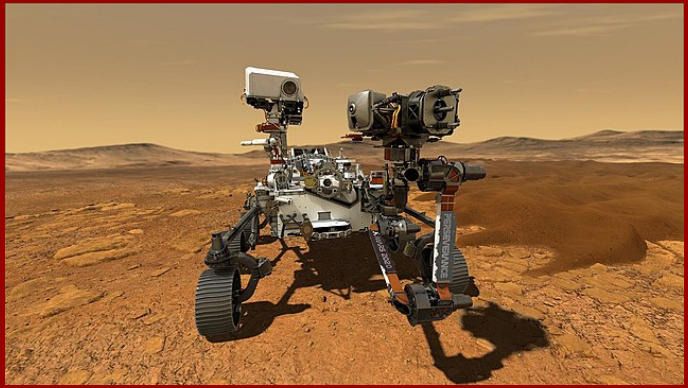
E[# ☹️ in round $i$] $= 2/i$

$$\text{E[\# ☹️ in any round]} = \frac{2}{3} + \frac{2}{4} + \frac{2}{5} + \cdots + \frac{2}{n} \sim 2 \ln n$$

**Variance** (thanks to team Latvia):

$$\Pr[\,|X - EX|\,] \leq 14 \ln N] \leq \frac{22}{196 \ln N}$$

DTU



Daniel Rutschmann AKA dacin21

# Tycho

$b$ distance to base $n$ hiding spots $p$ pulse period $d$ pulse damage

**Subtask 1** Try waiting $0, 1, \ldots, p-1$ seconds. The $i$-th hiding spot helps only when waiting $a[i] \bmod p$ seconds. $\mathcal{O}(p+n)$

$b$ distance to base  $n$ hiding spots  $p$ pulse period  $d$ pulse damage

**Subtask 1** Try waiting $0, 1, \ldots, p-1$ seconds. The $i$-th hiding spot helps only when waiting $a[i] \bmod p$ seconds. $\mathcal{O}(p+n)$
**Subtask 3** Dijkstra/DP over states (position, time until next pulse). $\mathcal{O}(b\,p)$

$b$ distance to base  $n$ hiding spots  $p$ pulse period  $d$ pulse damage

**Subtask 1** Try waiting $0, 1, \ldots, p-1$ seconds. The $i$-th hiding spot helps only when waiting $a[i] \bmod p$ seconds. $\mathcal{O}(p+n)$
**Subtask 3** Dijkstra/DP over states (position, time until next pulse). $\mathcal{O}(b\,p)$

## Observations

Only wait at hiding spots. Wait until right after the next pulse.

**Subtask 2** Try all $2^n$ subsets of hiding spots. $\mathcal{O}(2^n \cdot n)$

*b* distance to base  *n* hiding spots  *p* pulse period  *d* pulse damage

## Dynamic Programming

$\mathrm{DP}[i]$: minimum damage taken after waiting at $i$-th hiding spot.

$$\mathrm{DP}[i] = \min_{j<i} \left( \mathrm{DP}[j] + \underbrace{\left\lceil \frac{a[i]-a[j]}{p} \right\rceil \cdot p}_{\text{environment}} + \underbrace{\left\lceil \frac{a[i]-a[j]}{p} \right\rceil \cdot d - d}_{\text{radiation pulses}} \right)$$

$$= \min_{j<i} \left( \mathrm{DP}[j] + \left\lceil \frac{a[i]-a[j]}{p} \right\rceil \cdot (p+d) - d \right)$$

*b* distance to base  *n* hiding spots  *p* pulse period  *d* pulse damage

## Dynamic Programming

$\mathrm{DP}[i]$: minimum damage taken after waiting at $i$-th hiding spot.

$$\mathrm{DP}[i] = \min_{j<i} \left( \mathrm{DP}[j] + \underbrace{\left\lceil \frac{a[i]-a[j]}{p} \right\rceil \cdot p}_{\text{environment}} + \underbrace{\left\lceil \frac{a[i]-a[j]}{p} \right\rceil \cdot d - d}_{\text{radiation pulses}} \right)$$

$$= \min_{j<i} \left( \mathrm{DP}[j] + \left\lceil \frac{a[i]-a[j]}{p} \right\rceil \cdot (p+d) - d \right)$$

**Subtask 4** DP in $\mathcal{O}(n^2)$.
**Subtask 5** For each $a[j] \bmod p$, check only the latest $j$. $\mathcal{O}(n\,p)$

$b$ distance to base $n$ hiding spots $p$ pulse period $d$ pulse damage

$$\mathrm{DP}[i] = \min_{j<i} \left( \mathrm{DP}[j] + \left\lceil \frac{a[i] - a[j]}{p} \right\rceil \cdot (p + d) - d \right)$$

$$\left\lceil \frac{a[i] - a[j]}{p} \right\rceil = \left\lfloor \frac{a[i]}{p} \right\rfloor - \left\lfloor \frac{a[j]}{p} \right\rfloor + \begin{cases} 1 & a[i] \bmod p > a[j] \bmod p \\ 0 & \text{otherwise} \end{cases}$$

$b$ distance to base $n$ hiding spots $p$ pulse period $d$ pulse damage

$$\mathrm{DP}[i] = \min_{j<i} \left( \mathrm{DP}[j] + \left\lceil \frac{a[i] - a[j]}{p} \right\rceil \cdot (p + d) - d \right)$$

$$\left\lceil \frac{a[i] - a[j]}{p} \right\rceil = \left\lfloor \frac{a[i]}{p} \right\rfloor - \left\lfloor \frac{a[j]}{p} \right\rfloor + \begin{cases} 1 & a[i] \bmod p > a[j] \bmod p \\ 0 & \text{otherwise} \end{cases}$$

$$\underbrace{\mathrm{DP}[i] - \left\lfloor \frac{a[i]}{p} \right\rfloor \cdot (p + d)}_{\text{depends on } i} = \min_{j<i} \Bigg( \underbrace{\mathrm{DP}[j] - \left\lfloor \frac{a[j]}{p} \right\rfloor \cdot (p + d)}_{\text{depends on } j} + \underbrace{\begin{Bmatrix} p + d & \dots \\ 0 & \dots \end{Bmatrix}}_{\text{range query}} \Bigg) - d$$

$\rightarrow$ Range min-query on $[0, a[i] \bmod p)$ and $[a[i] \bmod p, p)$.

$b$ distance to base $n$ hiding spots $p$ pulse period $d$ pulse damage

$$\mathrm{DP}[i] = \min_{j<i} \left( \mathrm{DP}[j] + \left\lceil \frac{a[i] - a[j]}{p} \right\rceil \cdot (p + d) - d \right)$$

$$\left\lceil \frac{a[i] - a[j]}{p} \right\rceil = \left\lfloor \frac{a[i]}{p} \right\rfloor - \left\lfloor \frac{a[j]}{p} \right\rfloor + \begin{cases} 1 & a[i] \bmod p > a[j] \bmod p \\ 0 & \text{otherwise} \end{cases}$$

$$\underbrace{\mathrm{DP}[i] - \left\lfloor \frac{a[i]}{p} \right\rfloor \cdot (p + d)}_{\text{depends on } i} = \min_{j<i} \Bigg( \underbrace{\mathrm{DP}[j] - \left\lfloor \frac{a[j]}{p} \right\rfloor \cdot (p + d)}_{\text{depends on } j} + \underbrace{\begin{cases} p + d & \dots \\ 0 & \dots \end{cases}}_{\text{range query}} \Bigg) - d$$

$\rightarrow$ Range min-query on $[0, a[i] \bmod p)$ and $[a[i] \bmod p, p)$.

**Subtask 6** Min-segment tree over $a[j] \bmod p$. $\mathcal{O}(p + n \log p)$.

**Subtask 7** Coordinate Compression / Implicit segment tree. $\mathcal{O}(n \log p)$

Daniel Rutschmann AKA dacin21

# Minequake

**Subtask 1** The tree is a path. Start at one end. Answer: $n \cdot (n-1)/2$.
**Subtask 2** The tree is a subdivision of a star. Start at the leaf on the longest “arm”.

**Subtask 1** The tree is a path. Start at one end. Answer: $n \cdot (n-1)/2$.
**Subtask 2** The tree is a subdivision of a star. Start at the leaf on the longest “arm”.

**Subtask 3** Brute force: try all orders. $\tilde{\mathcal{O}}(n!)$

**Subtask 1** The tree is a path. Start at one end. Answer: $n \cdot (n-1)/2$.
**Subtask 2** The tree is a subdivision of a star. Start at the leaf on the longest “arm”.

**Subtask 3** Brute force: try all orders. $\tilde{\mathcal{O}}(n!)$

## Observations

An optimal solution traverses every tunnel at most twice. Then, only the choice of starting hall matters. (proof later)

**Subtask 4** DFS from every hall. $\mathcal{O}(n^2)$

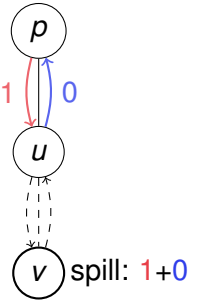
## Claim: Only starting hall matters

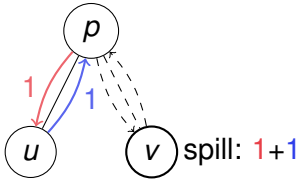
Root the tree at the starting hall. The total spill is *at least* $\sum_{u<v} s(u,v)$ where

$$s(u,v) = \begin{cases} 0 & u \text{ or } v \text{ is the root} \\ 1 & u, v \text{ are ancestors/descendants} \\ 2 & u, v \text{ are incomparable} \end{cases}$$

If every tunnel is visited at most twice, the total spill is *exactly* $\sum_{u<v} s(u,v)$.

Let $p = \text{parent}(u)$.
Consider spill at $v$ when traversing $e = \{p, u\}$.



## Claim: Only starting hall matters

Root the tree at the starting hall. The total spill is *at least* $\sum_{u<v} s(u,v)$ where

$$s(u,v) = \begin{cases} 0 & u \text{ or } v \text{ is the root} \\ 1 & u, v \text{ are ancestors/descendants} \\ 2 & u, v \text{ are incomparable} \end{cases}$$

If every tunnel is visited at most twice, the total spill is *exactly* $\sum_{u<v} s(u,v)$.

**Subtask 5** Root the tree arbitrarily. Compute $\mathsf{ans}[\mathit{root}]$ and subtree sizes with DFS. Then

$$\mathsf{ans}[u] = \mathsf{ans}[\mathsf{parent}(u)] + 2 \cdot \mathsf{size}[u] - n.$$

# Mineral Deposits Solution Sketch

David R. Lolck

April 30, 2023

# Problem

## Problem

There are $k$ hidden points $(x_1, y_1), \ldots, (x_k, y_k)$.
Access to following query for:

$$Ask((s_1, t_1), \ldots, (s_d, t_d)) = \{|s_i - x_j| + |t_i - y_j| \text{ for } (i,j) \in \{1, \ldots, d\} \times \{1, \ldots, k\}\}$$

Determine the hidden points, minimising the number of queries.

# Sketch

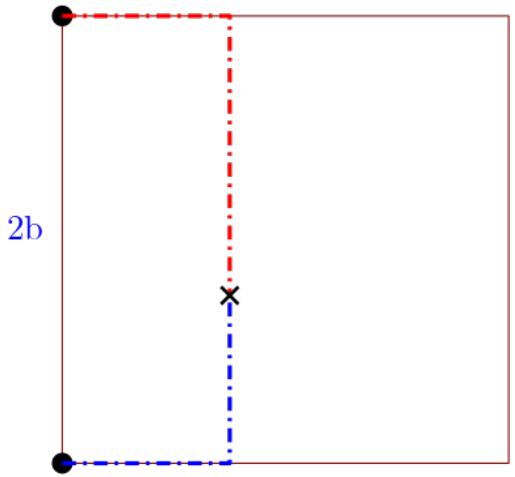
## Subtask 1: $k = 1, w = 10^4$

We can with the two queries

$$Ask((-b, -b)) = a$$

$$Ask((-b, b)) = c$$

uniquely determine the single point



# Sketch

## Observation 1

If we were to ask the queries

$$Ask((-b, -b)) = a_1, \ldots, a_k$$

$$Ask((-b, b)) = c_1, \ldots, c_k$$

Then for each hidden point $(x_i, y_i)$ there exists a pair of distances $c_s, d_t$, such that those would be returned if $(x_i, y_i)$ was the only hidden point.
Trying all pairs of distances gives $k^2$ candidate points. This is a superset of the hidden points.

# Sketch

## Observation 1

If we were to ask the queries

$$Ask((-b, -b)) = a_1, \ldots, a_k$$

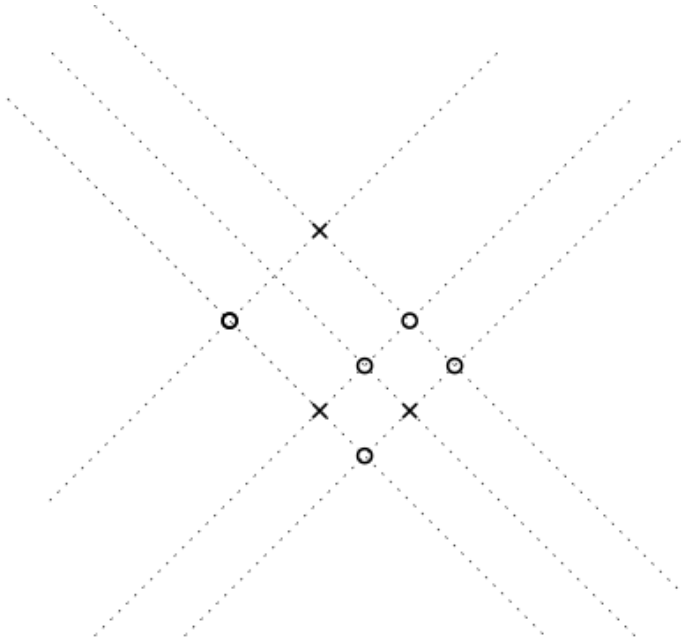
$$Ask((-b, b)) = c_1, \ldots, c_k$$

Then for each hidden point $(x_i, y_i)$ there exists a pair of distances $c_s, d_t$, such that those would be returned if $(x_i, y_i)$ was the only hidden point.
Trying all pairs of distances gives $k^2$ candidate points. This is a superset of the hidden points.

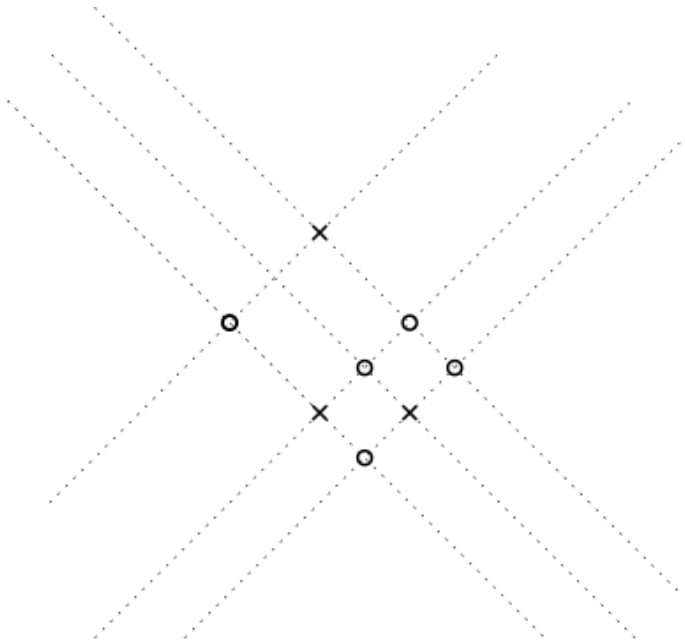
## Observation 2

If you ask a query $Ask((s_1, t_1), \ldots, (s_d, t_d))$, then the number of times 0 occurs in the answer corresponds to the number of mineral deposits in $(s_1, t_1), \ldots, (s_d, t_d)$.

# Sketch



# Sketch



## Subtask 2: $w \geq 500$

Reduce to $k^2$ candidate points. Ask a query for each of them. If the result contains a 0, queried point is a mineral deposit.

# Sketch

## Subtask 3:$w \geq 210$

Reduce to $k^2$. We can locate 1 point in $\lceil \lg(k^2) \rceil$ queries. Split $k^2$ candidates in two sets and query one of them. Then we can determine if there is a mineral deposit in this set or the other one. Once we have located a deposit, never ask about it again.

# Sketch

## Subtask 3: $w \geq 210$

Reduce to $k^2$. We can locate 1 point in $\lceil \lg(k^2) \rceil$ queries. Split $k^2$ candidates in two sets and query one of them. Then we can determine if there is a mineral deposit in this set or the other one. Once we have located a deposit, never ask about it again.

## Subtask 4: $w \geq 130$

Same as before, but count the number of mineral deposits in each set, and find all of them at once.

# Sketch

## Subtask 3: $w \geq 210$

Reduce to $k^2$. We can locate 1 point in $\lceil \lg(k^2) \rceil$ queries. Split $k^2$ candidates in two sets and query one of them. Then we can determine if there is a mineral deposit in this set or the other one. Once we have located a deposit, never ask about it again.

## Subtask 4: $w \geq 130$

Same as before, but count the number of mineral deposits in each set, and find all of them at once.

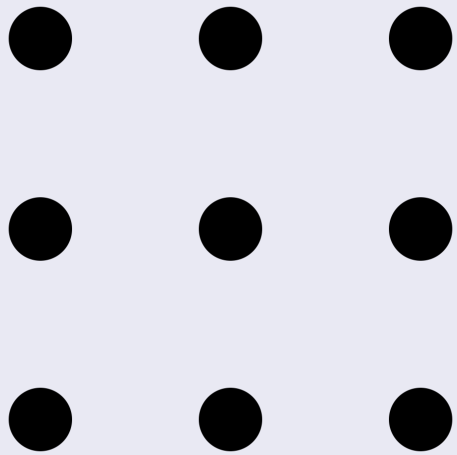
## Honorable mention: Random selection

If you simply select a random candidate point, query it, and then remove inconsistent candidate points, this will perform better than the binary search based solutions.

# Sketch

## Aside

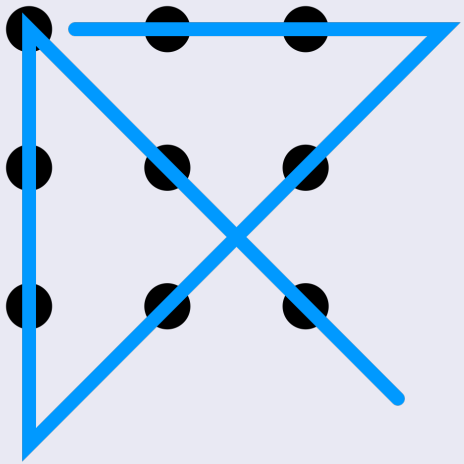
Some of you have seen the following puzzle: Cover all 9 dots with only 4 lines.



# Sketch

## Aside

Some of you have seen the following puzzle: Cover all 9 dots with only 4 lines.



You have to think outside the box, and we will do the same.

# Sketch

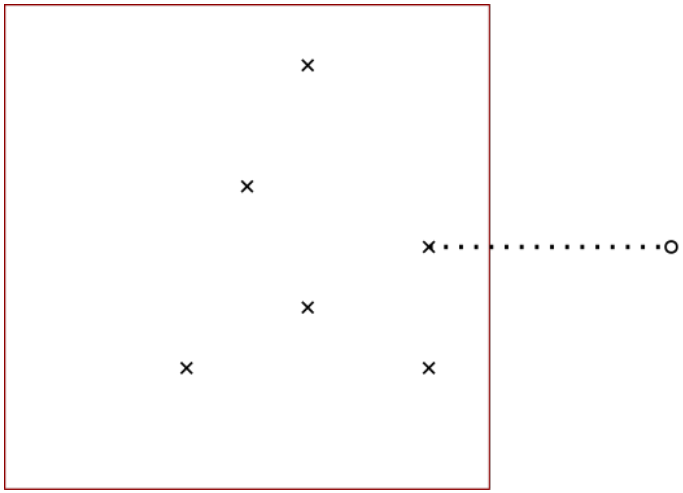
## Observation

Let $(u, v)$ be the right-most candidate point. Then placing a point at coordinate $(b + x, v)$ for some $x$, tells us whether $(u, v)$ is a deposit.

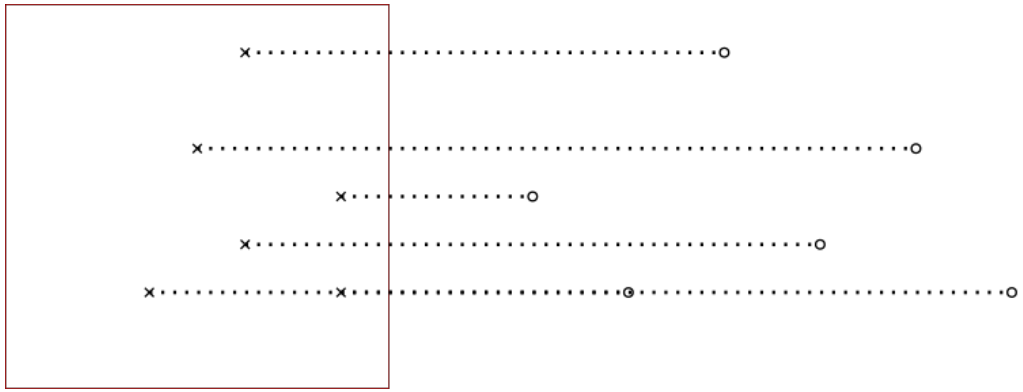
# Sketch

## Observation

Let $(u, v)$ be the right-most candidate point. Then placing a point at coordinate $(b + x, v)$ for some $x$, tells us whether $(u, v)$ is a deposit.



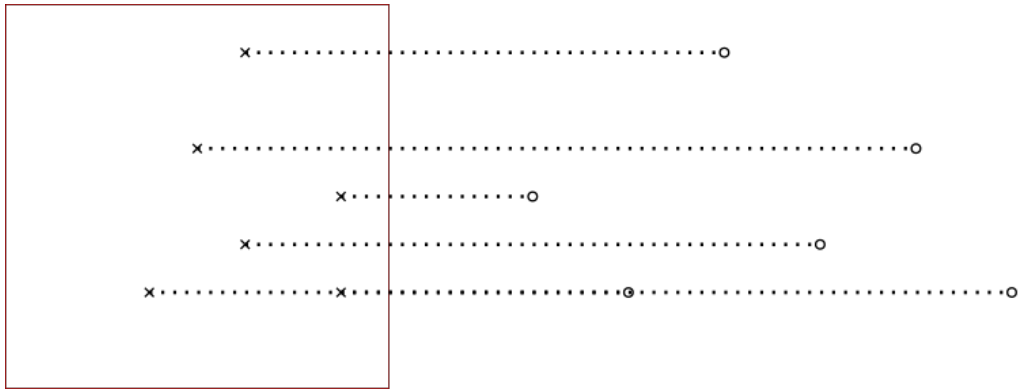
# Sketch



## Subtask 5: $w \geq 3, b \leq 10^4$

Process candidates from right to left. On the same $y$ coordinate as the candidate, place outside the query point. If the distance between the query points are at least $5b$, then the distance in the response can each be associated with a query point. E.g. the query points become $(x, 5b), (y, 10b), (z, 15b), \ldots$. Query the points

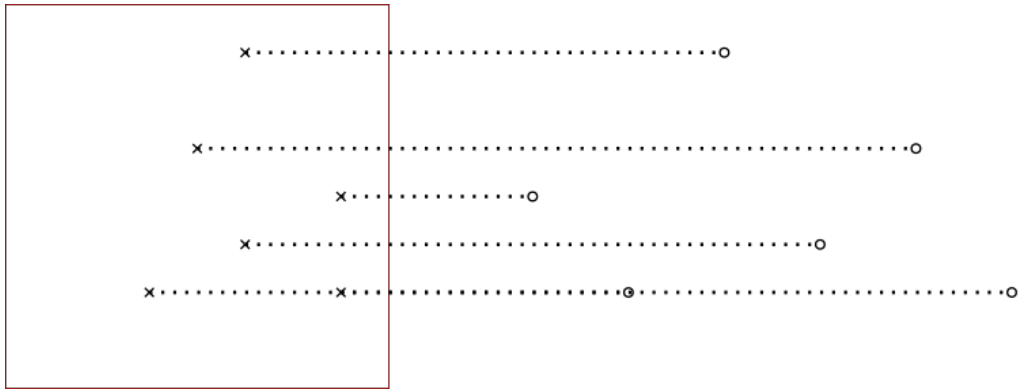
# Sketch



## Subtask 5: $w \geq 3, b \leq 10^4$

Process candidates from right to left. On the same $y$ coordinate as the candidate, place outside the query point. If the distance between the query points are at least $5b$, then the distance in the response can each be associated with a query point. E.g. the query points become $(x, 5b), (y, 10b), (z, 15b), \ldots$. Query the points For each candidate, if the distance between the associated query point and the candidate is in the result of the query, it is a deposit. Then undo all distances that are a result of this deposit. Uses distances of $O(k^2 b)$ outside the $b$-box

# Sketch



## Subtask 6: $w \geq 3, b \leq 10^7$

Same idea as before. Observe that we can instead compute all the $k^2$ distances that could arise from each query point. Determine the smallest distance that hasn't been generated further away than the previous point and place it there. It can be shown that this uses query points of at most $k^4$ distance outside the $b$-box.

# Sketch

## Observation

We can reduce to $4k^2$ query points using only a single query, by combining the two queries into one and taking every possible combination of distances.

# Sketch

## Observation

We can reduce to $4k^2$ query points using only a single query, by combining the two queries into one and taking every possible combination of distances.

## Subtask 7: *No further restrictions*

Observe that when $b$ is very large, there is going to be a lot of empty space. Do essentially the same as subtask 6, but in all 4 directions at the same time. You have to make sure that the points remain uniquely decodeable while placing new query points. You also have to make sure that new query points are not too close to candidate points, and that you don't duplicate some important distances.
Running time: Depending on implementation either $O(k^4)$ or $O(k^6)$.

# Sequence

**Given:** Positive weights $w = (w_1, \ldots, w_n)$

**Define:** *weight* of index sequence $x = (x_1, \ldots, x_m)$ as

$$W(x) = w_{x_1} + \ldots + w_{x_m}$$

**Define:** index sequence $x = (x_1, \ldots, x_m)$ is *good* if $x = (1)$ or

$$x_j = \begin{cases} x_{j-1} + 1 & \textbf{or} \\ x_k \cdot x_l & \textbf{for some } k \leq l < j \end{cases}$$

increment (indigo)

multiply (maroon)

**Task:** given $w$ and index v, compute $\min_{x \in G:\ v \in x} W(x)$

**Good:**

1
1 1
1 2
1 2 2
1 2 3
1 2 4
1 2 4 5
1 2 4 8
1 2 4 16

**Not good:**

1 3

**Small n:** Systematically generate all *short good* sequences by following the two rules

**Too slow:** Systematically generate all $10^{10}$ index sequences of length at most 10 and check each for goodness.

# Sequence

**Given:** Positive weights $w = (w_1, \ldots, w_n)$

**Define:** *weight* of index sequence $x = (x_1, \ldots, x_m)$ as

$$W(x) = w_{x_1} + \ldots + w_{x_m}$$

**Define:** index sequence $x = (x_1, \ldots, x_m)$ is *good* if $x = (1)$ or

$$x_j = \begin{cases} x_{j-1} + 1 & \textbf{or} \\ x_k \cdot x_l & \textbf{for some } k \le l < j \end{cases}$$

increment (indigo)

multiply (maroon)

**Good:**

1
1 1
1 2
1 2 2
1 2 3
1 2 4
1 2 4 5
1 2 4 8
1 2 4 16

**Task:** given $w$ and index v, compute $\min_{x \in G:\ v \in x} W(x)$

**Observation:** $w_i > 0 \Rightarrow$

- Ignore $x$ if it is "supersequence" of other $y \in G$
- Can assume $x$ is *strictly* increasing (speeds up generation)
- Can assume $v$ is *last* index $x_m$

1 2 3
1 2 4
1 2 2 4
1 2 4 5
1 2 3 4 5

# Uniform weights

**Given:** Positive weights $w = (w_1, \ldots, w_n)$ $(w_1 = \cdots = w_n)$

**Define:** *weight* of index sequence $x = (x_1, \ldots, x_m)$ as

$$W(x) = w_{x_1} + \ldots + w_{x_m} \quad = w_1 \cdot |x|$$

**Define:** index sequence $x = (x_1, \ldots, x_m)$ is *good* if $x = (1)$ or

$$x_j = \begin{cases} x_{j-1} + 1 & \textbf{or} \\ x_k \cdot x_l & \textbf{for some } k \le l < j \end{cases}$$

increment (indigo)

multiply (maroon)

**Task:** given $w$ and index $v$, compute $\min_{x \in G:\ v \in x} W(x)$ $= w_i \cdot \min_{(x_1, \ldots, v) \in G} |x|$

**Good:**

1
1 1
1 2
1 2 2
1 2 3
1 2 4
1 2 4 5
1 2 4 8
1 2 4 16

**Simpler task:** Find shortest good sequence ending in *v.*

v < 300**:** Just do it.

v < 1439**:** Sequence lengths are $|x| \le 14$. Precompute *all* of them on your machine in an hour.

1 2 3
1 2 4
1 2 2 4
1 2 4 5
1 2 3 4 5

# Intended solution

**Given:** Positive weights $w = (w_1, \ldots, w_n)$

**Define:** *weight* of index sequence $x = (x_1, \ldots, x_m)$ as

$$W(x) = w_{x_1} + \ldots + w_{x_m}$$

**Define:** index sequence $x = (x_1, \ldots, x_m)$ is *good* if $x = (1)$ or

$$x_j = \begin{cases} x_{j-1} + 1 & \textbf{or} \\ x_k \cdot x_l & \textbf{for some } k \le l < j \end{cases}$$

increment (indigo)

multiply (maroon)

**Good:**

1
1 1
1 2
1 2 2
1 2 3
1 2 4
1 2 4 5
1 2 4 8
1 2 4 16

**Task:** given $w$ and index v, compute $\min_{x \in G:\ v \in x} W(x)$

```
generator extend(x_1, ..., x_j):
    yield(x_1, ..., x_j, x_j + 1)
    for k ∈ {2,...,j}
        for j ∈ {k, ..., j}
            yield(x_1, ..., x_j, x_k · x_l)
```

Generalise problem: for index subset $A \subseteq \{1,\ldots,n\}$ define

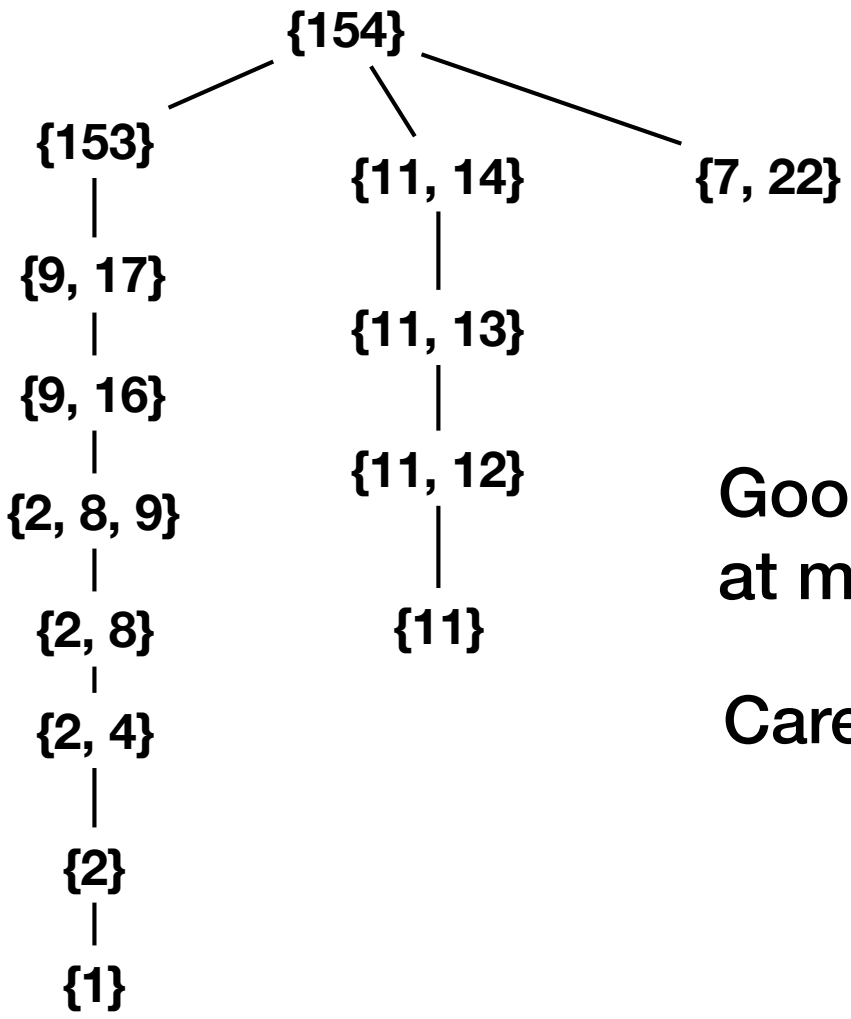
$$F(A) = \min_{x \in G:\ A \subseteq x} W(x)$$

Original problem: compute $F(\{v\})$.

$$F(A \cup \{a_k\}) = w_{a_k} + \min \begin{cases} F(A \cup \{a_k - 1\}) \\ \min_{1 < i \le j \le a_k:\ ij = a_k} F(A \cup \{i, j\}) \end{cases}$$

# Why is this fast?

$$F(A \cup \{a_k\}) = w_{a_k} + \min \begin{cases} F(A \cup \{a_k - 1\}) \\ \min_{1<i\le j\le a_k:\ ij=a_k} F(A \cup \{i, j\}) \end{cases}$$




Invariant

$$a_1 \cdot \cdots \cdot a_k \le v \le n$$

Good exercise (induction, calculus): at most poly($n$) such sets

Careful analysis (Team Poland):

$$ne^{2\cdot\sqrt{\log n}}$$