



CTU Open 2026

Presentation of solutions

A year in the life of Míša, a young and promising Ph.D. student.

Vampires

Miša checks whether the big bat is a vampire number with the two small bats as fangs.

● ● ● ● ● implementation · strings

Vampires

Task

Given $t \leq 1000$ triples n, a, b with $1 \leq n, a, b < 10^8$, decide for each whether n is a vampire number with fangs a and b .

Vampires

Task

Given $t \leq 1000$ triples n, a, b with $1 \leq n, a, b < 10^8$, decide for each whether n is a vampire number with fangs a and b .

n is a vampire number with fangs a and b if and only if:

- ▶ $n = a \cdot b$
- ▶ $\text{len}(a) = \text{len}(b)$ and $\text{len}(n) = 2 \cdot \text{len}(a)$
- ▶ not both $a \equiv 0$ and $b \equiv 0 \pmod{10}$
- ▶ digits of a followed by b are a *permutation* of the digits of n

Vampires

Task

Given $t \leq 1000$ triples n, a, b with $1 \leq n, a, b < 10^8$, decide for each whether n is a vampire number with fangs a and b .

n is a vampire number with fangs a and b if and only if:

- ▶ $n = a \cdot b$
- ▶ $\text{len}(a) = \text{len}(b)$ and $\text{len}(n) = 2 \cdot \text{len}(a)$
- ▶ not both $a \equiv 0$ and $b \equiv 0 \pmod{10}$
- ▶ digits of a followed by b are a *permutation* of the digits of n
- ▶ Just check the four conditions literally, one by one.

Vampires – the permutation check, $102510 = 510 \cdot 201$

n

1	0	2	5	1	0
---	---	---	---	---	---

1. concatenate $a = 510$ and $b = 201$

$a \parallel b$

5	1	0	2	0	1
---	---	---	---	---	---

Vampires – the permutation check, $102510 = 510 \cdot 201$

n

1

0

2

5

1

0

$a \parallel b$

5

1

0

2

0

1

1. concatenate $a = 510$ and $b = 201$
2. sort both strings

Vampires – the permutation check, $102510 = 510 \cdot 201$

n

0

0

1

1

2

5

$a \parallel b$

0

0

1

1

2

5

1. concatenate $a = 510$ and $b = 201$
2. sort both strings

Vampires – the permutation check, $102510 = 510 \cdot 201$

n

0	0	1	1	2	5
---	---	---	---	---	---

 = = = = = =

$a \parallel b$

0	0	1	1	2	5
---	---	---	---	---	---

1. concatenate $a = 510$ and $b = 201$

2. sort both strings

3. compare

Vampires – the permutation check, $102510 = 510 \cdot 201$

n	0	0	1	1	2	5
	=	=	=	=	=	=
$a \parallel b$	0	0	1	1	2	5

1. concatenate $a = 510$ and $b = 201$
2. sort both strings
3. compare $\Rightarrow$ **YES**

Vampires – the permutation check, $102510 = 510 \cdot 201$

n

0	0	1	1	2	5
---	---	---	---	---	---

= = = = = =

$a \parallel b$

0	0	1	1	2	5
---	---	---	---	---	---

1. concatenate $a = 510$ and $b = 201$

2. sort both strings

3. compare $\Rightarrow$ **YES**

2278

2	2	7	8
---	---	---	---

34 $\parallel$ 67

3	4	6	7
---	---	---	---

$34 \cdot 67 = 2278$ holds,
but the digits differ $\Rightarrow$ **NO**

Vampires – the permutation check by counting

Count each digit 0..9 on both sides, then compare the ten counters.

digit	0	1	2	3	4	5	6	7	8	9
102510	0	0	0	0	0	0	0	0	0	0

Vampires – the permutation check by counting

Count each digit 0..9 on both sides, then compare the ten counters.

	digit	0	1	2	3	4	5	6	7	8	9
102510		0	1	0	0	0	0	0	0	0	0

↑

Vampires – the permutation check by counting

Count each digit 0..9 on both sides, then compare the ten counters.

digit	0	1	2	3	4	5	6	7	8	9
102510	1	1	0	0	0	0	0	0	0	0

↑

Vampires – the permutation check by counting

Count each digit 0..9 on both sides, then compare the ten counters.

	digit	0	1	2	3	4	5	6	7	8	9
102510		1	1	1	0	0	0	0	0	0	0

↑

Vampires – the permutation check by counting

Count each digit 0..9 on both sides, then compare the ten counters.

digit	0	1	2	3	4	5	6	7	8	9
102510	1	1	1	0	0	1	0	0	0	0

↑

Vampires – the permutation check by counting

Count each digit 0..9 on both sides, then compare the ten counters.

digit	0	1	2	3	4	5	6	7	8	9
102510	1	2	1	0	0	1	0	0	0	0

↑

Vampires – the permutation check by counting

Count each digit 0..9 on both sides, then compare the ten counters.

digit	0	1	2	3	4	5	6	7	8	9
102510	2	2	1	0	0	1	0	0	0	0

↑

Vampires – the permutation check by counting

Count each digit 0..9 on both sides, then compare the ten counters.

digit	0	1	2	3	4	5	6	7	8	9
102510	2	2	1	0	0	1	0	0	0	0
510 201	2	2	1	0	0	1	0	0	0	0

Vampires – the permutation check by counting

Count each digit 0..9 on both sides, then compare the ten counters.

digit	0	1	2	3	4	5	6	7	8	9	
102510	2	2	1	0	0	1	0	0	0	0	YES
510 201	2	2	1	0	0	1	0	0	0	0	

Vampires – the permutation check by counting

Count each digit 0..9 on both sides, then compare the ten counters.

digit	0	1	2	3	4	5	6	7	8	9	
102510	2	2	1	0	0	1	0	0	0	0	YES
510 201	2	2	1	0	0	1	0	0	0	0	
2278	0	0	2	0	0	0	0	1	1	0	NO
34 67	0	0	0	1	1	0	1	1	0	0	

Vampires – the traps

Pitfalls

Make sure your checks are correct:

Vampires – the traps

Pitfalls

Make sure your checks are correct:

- ▶ Trailing zeros: reject only if *both* fangs end in 0.
 - $a \bmod 10 = 0 \vee b \bmod 10 = 0$ rejects $1260 = 21 \cdot 60$
 - $n \bmod 100 = 0$ rejects $120600 = 201 \cdot 600$

Vampires – the traps

Pitfalls

Make sure your checks are correct:

- ▶ Trailing zeros: reject only if *both* fangs end in 0.
 - $a \bmod 10 = 0 \vee b \bmod 10 = 0$ rejects $1260 = 21 \cdot 60$
 - $n \bmod 100 = 0$ rejects $120600 = 201 \cdot 600$
- ▶ Permutations must check counts:
 - `set(n) == set(a||b)` does not reject $5775 = 75 \cdot 77$

Vampires – the traps

Pitfalls

Make sure your checks are correct:

- ▶ Trailing zeros: reject only if *both* fangs end in 0.
 - $a \bmod 10 = 0 \vee b \bmod 10 = 0$ rejects $1260 = 21 \cdot 60$
 - $n \bmod 100 = 0$ rejects $120600 = 201 \cdot 600$
- ▶ Permutations must check counts:
 - `set(n) == set(a||b)` does not reject $5775 = 75 \cdot 77$

Complexity: $\mathcal{O}(t)$

at most 8 digits per number $\Rightarrow$ counting and sorting are both effectively $\mathcal{O}(1)$

Trains

Miša watches m trains through a window of n cells.

● ○ ○ ○ ○ simulation · implementation

Trains

Task

For each track, print what is visible in cells $0..n-1$ at time t : engine !, car #, empty =.

Trains

Task

For each track, print what is visible in cells $0..n-1$ at time t : engine !, car #, empty =.

- ▶ Tracks are independent

Trains

Task

For each track, print what is visible in cells $0..n-1$ at time t : engine !, car #, empty =.

- ▶ Tracks are independent – a train is a rigid block, it only *shifts* by $v \cdot t$, where $v = +s$ for R and $v = -s$ for L.

Trains

Task

For each track, print what is visible in cells $0..n-1$ at time t : engine !, car #, empty =.

- ▶ Tracks are independent – a train is a rigid block, it only *shifts* by $v \cdot t$, where $v = +s$ for R and $v = -s$ for L.
- ▶ At time t it covers $[p + vt, p + vt + \ell - 1]$, intersect it with the window $[0, n - 1]$.

Trains

Task

For each track, print what is visible in cells $0..n-1$ at time t : engine !, car #, empty =.

- ▶ Tracks are independent – a train is a rigid block, it only *shifts* by $v \cdot t$, where $v = +s$ for R and $v = -s$ for L.
- ▶ At time t it covers $[p + vt, p + vt + \ell - 1]$, intersect it with the window $[0, n - 1]$.
- ▶ Engine = the end pointing in the direction of travel.

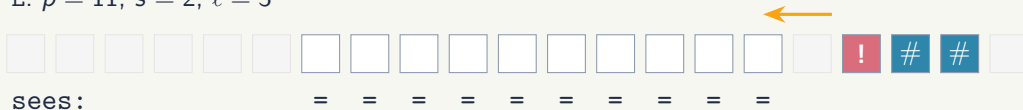
Trains – two tracks, $n = 10$

$t = 0$

R: $p = -3, s = 2, \ell = 4$



L: $p = 11, s = 2, \ell = 3$

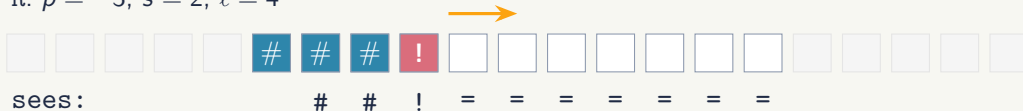


The engine is always the end facing the direction of travel.

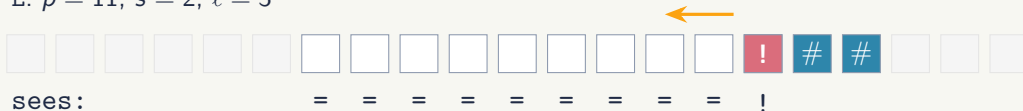
Trains – two tracks, $n = 10$

$t = 1$

R: $p = -3, s = 2, \ell = 4$



L: $p = 11, s = 2, \ell = 3$



The engine is always the end facing the direction of travel.

Trains – two tracks, $n = 10$

$t = 2$

R: $p = -3, s = 2, \ell = 4$



L: $p = 11, s = 2, \ell = 3$



The engine is always the end facing the direction of travel.

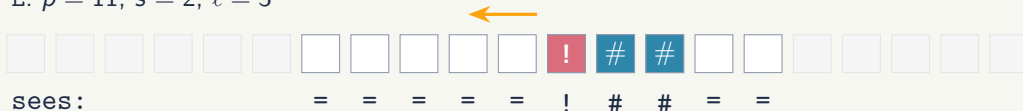
Trains – two tracks, $n = 10$

$t = 3$

R: $p = -3, s = 2, \ell = 4$



L: $p = 11, s = 2, \ell = 3$



The engine is always the end facing the direction of travel.

Trains – two tracks, $n = 10$

$t = 4$

R: $p = -3, s = 2, \ell = 4$



L: $p = 11, s = 2, \ell = 3$



The engine is always the end facing the direction of travel.

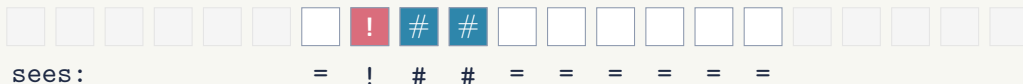
Trains – two tracks, $n = 10$

$t = 5$

R: $p = -3, s = 2, \ell = 4$



L: $p = 11, s = 2, \ell = 3$



The R engine left the window – only its cars are visible.

Trains – the traps

Pitfalls

- ▶ $|v \cdot t|$ up to 10^{10} – use 64-bit integers.

Trains – the traps

Pitfalls

- ▶ $|v \cdot t|$ up to 10^{10} – use 64-bit integers.

Trains – the traps

Pitfalls

- ▶ $|v \cdot t|$ up to 10^{10} – use 64-bit integers.

Trains – the traps

Pitfalls

- ▶ $|v \cdot t|$ up to 10^{10} – use 64-bit integers.

Trains – the traps

Pitfalls

- ▶ $|v \cdot t|$ up to 10^{10} – use 64-bit integers.
- ▶ Don't iterate over the whole train – only over the intersection with the window.

Complexity: $\mathcal{O}(n \cdot m)$

output size is $n \cdot m \leq 4 \cdot 10^6$

Milky way

Miša listens to cosmic tones and likes only non-decreasing stretches of them.

● ● ● ● ● prefix sums · precomputation

Milkyway

Task

Given $a_1, \dots, a_n$, answer q queries: is $a_\ell \leq a_{\ell+1} \leq \dots \leq a_r$?

Milkyway

Task

Given $a_1, \dots, a_n$, answer q queries: is $a_\ell \leq a_{\ell+1} \leq \dots \leq a_r$?

▶ $n \leq 10^6$, $q \leq 10^5$.

Milkyway

Task

Given $a_1, \dots, a_n$, answer q queries: is $a_\ell \leq a_{\ell+1} \leq \dots \leq a_r$?

- ▶ $n \leq 10^6$, $q \leq 10^5$.
- ▶ Scanning each query: $O(n \cdot q) \approx 10^{11}$ steps – too slow.

Task

Given $a_1, \dots, a_n$, answer q queries: is $a_\ell \leq a_{\ell+1} \leq \dots \leq a_r$?

- ▶ $n \leq 10^6$, $q \leq 10^5$.
- ▶ Scanning each query: $O(n \cdot q) \approx 10^{11}$ steps – too slow.
- ▶ Observation: An interval is not satisfactory iff it contains a *drop*: an i where $a_{i-1} > a_i$, $\ell < i \leq r$.

Task

Given $a_1, \dots, a_n$, answer q queries: is $a_\ell \leq a_{\ell+1} \leq \dots \leq a_r$?

- ▶ $n \leq 10^6$, $q \leq 10^5$.
- ▶ Scanning each query: $O(n \cdot q) \approx 10^{11}$ steps – too slow.
- ▶ Observation: An interval is not satisfactory iff it contains a *drop*: an i where $a_{i-1} > a_i$, $\ell < i \leq r$.
- ▶ Drops break up the array into maximal non-decreasing *segments*.

Milkyway

Task

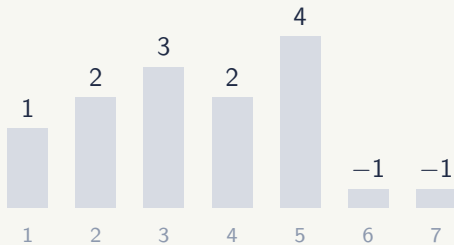
Given $a_1, \dots, a_n$, answer q queries: is $a_\ell \leq a_{\ell+1} \leq \dots \leq a_r$?

- ▶ $n \leq 10^6$, $q \leq 10^5$.
- ▶ Scanning each query: $O(n \cdot q) \approx 10^{11}$ steps – too slow.
- ▶ Observation: An interval is not satisfactory iff it contains a *drop*: an i where $a_{i-1} > a_i$, $\ell < i \leq r$.
- ▶ Drops break up the array into maximal non-decreasing *segments*.

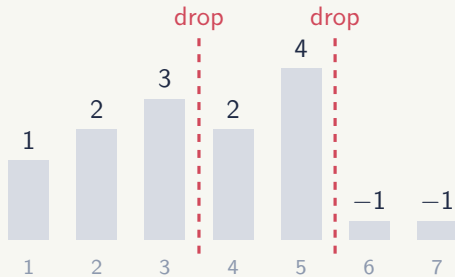
Key idea

Split a into non-decreasing segments using drops. Number the segments.
Query (ℓ, r) is **YES** iff $\text{segment}[\ell] = \text{segment}[r]$.

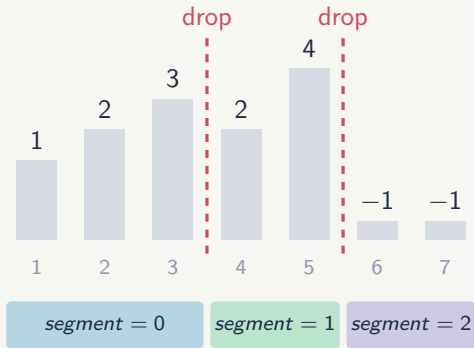
Milkyway – sample input



Milkyway – sample input

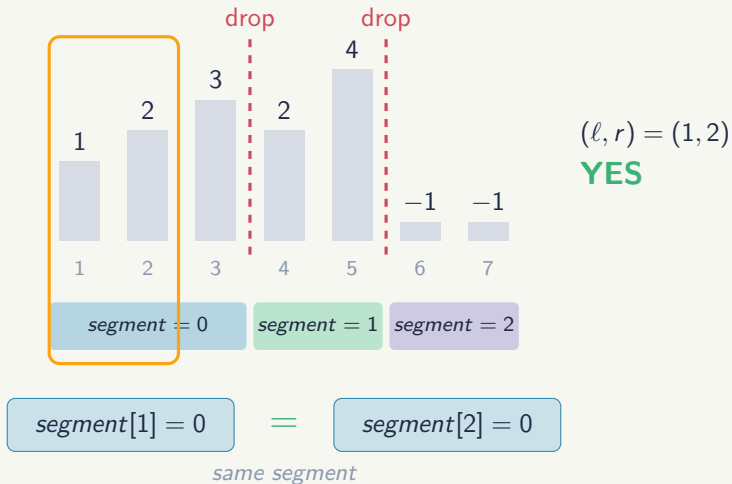


Milkyway – sample input

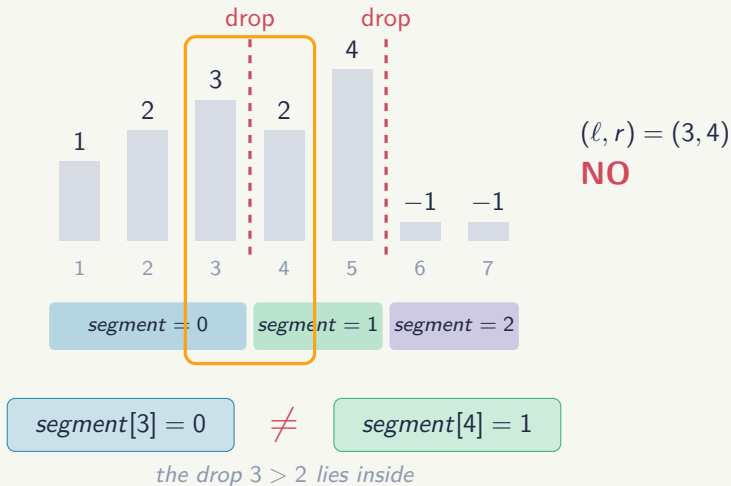


$$\text{segment}[i] = \text{segment}[i - 1] + [a_{i-1} > a_i]$$

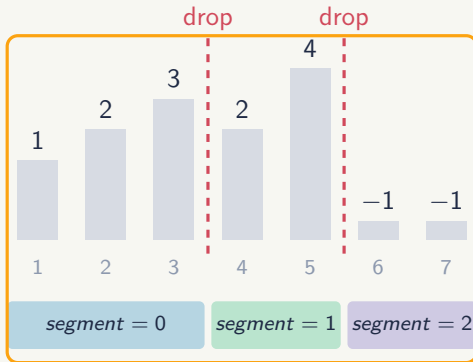
Milkyway – sample input



Milkyway – sample input



Milkyway – sample input



$(\ell, r) = (1, 7)$

NO

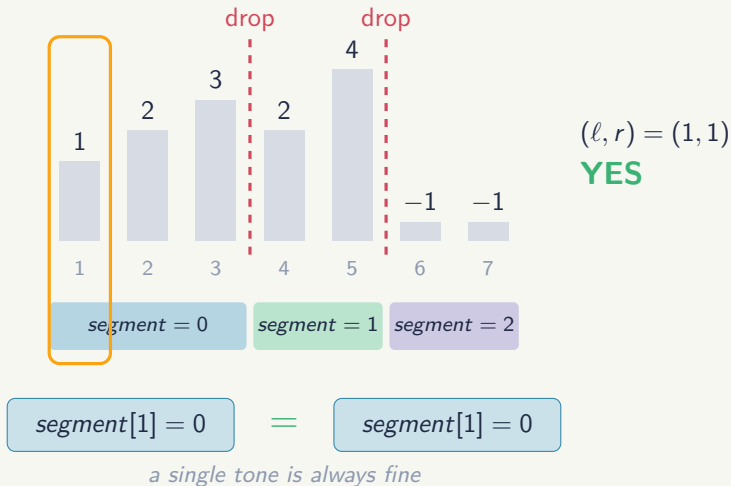
$segment[1] = 0$

$\neq$

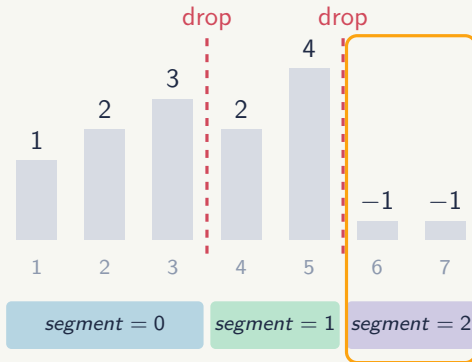
$segment[7] = 2$

different segments

Milkyway – sample input



Milkyway – sample input



$(\ell, r) = (6, 7)$
YES

$segment[6] = 2$

$=$

$segment[7] = 2$

equal neighbours are fine ($-1 \leq -1$)

Milkyway – summary and possible pitfalls

- ▶ One pass to compute *segment*[], then $\mathcal{O}(1)$ per query.
- ▶ Equivalent: prefix count of drops D , answer YES iff $D[r] = D[\ell]$.

Complexity: $\mathcal{O}(n + q)$

linear precomputation, constant-time queries

Pitfalls

- ▶ 10^6 numbers to read – use fast input (`scanf`, `sync_with_stdio(false)`). Shouldn't be needed, but it's good practice.

Milkyway – summary and possible pitfalls

- ▶ One pass to compute $segment[\cdot]$, then $\mathcal{O}(1)$ per query.
- ▶ Equivalent: prefix count of drops D , answer YES iff $D[r] = D[\ell]$.

Complexity: $\mathcal{O}(n + q)$

linear precomputation, constant-time queries

Pitfalls

- ▶ 10^6 numbers to read – use fast input (`scanf`, `sync_with_stdio(false)`). Shouldn't be needed, but it's good practice.
- ▶ Some teams stored and looked up segments as simple $[start, end]$ pairs using bin search, which resulted in $\mathcal{O}(q \cdot \log(n))$ solution. This was quick enough to pass, but significantly more error prone.

Accident

Miša dropped his right-angled triangle; one vertex must move back.

Accident

Task

Three non-collinear integer points. Move *one* of them by the smallest distance so that they form a right-angled triangle.

- ▶ Choose which point P moves; the other two U, V stay.

Accident

Task

Three non-collinear integer points. Move *one* of them by the smallest distance so that they form a right-angled triangle.

- ▶ Choose which point P moves; the other two U, V stay.
- ▶ Right angle at $P \iff P$ lies on the **Thales circle** with diameter UV .

Accident

Task

Three non-collinear integer points. Move *one* of them by the smallest distance so that they form a right-angled triangle.

- ▶ Choose which point P moves; the other two U, V stay.
- ▶ Right angle at $P \iff P$ lies on the **Thales circle** with diameter UV .
- ▶ Cheapest move onto a circle: along the ray from its centre M , cost $||PM| - r|$.

Accident

Task

Three non-collinear integer points. Move *one* of them by the smallest distance so that they form a right-angled triangle.

- ▶ Choose which point P moves; the other two U, V stay.
- ▶ Right angle at $P \iff P$ lies on the **Thales circle** with diameter UV .
- ▶ Cheapest move onto a circle: along the ray from its centre M , cost $||PM| - r|$.
- ▶ Right angle at U or V instead? Then P goes to a line – but we will see that *this never wins*.

Accident

Task

Three non-collinear integer points. Move *one* of them by the smallest distance so that they form a right-angled triangle.

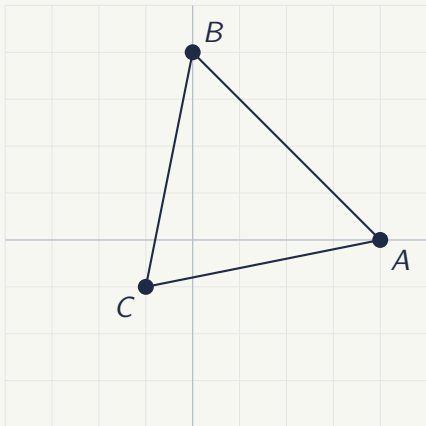
- ▶ Choose which point P moves; the other two U, V stay.
- ▶ Right angle at $P \iff P$ lies on the **Thales circle** with diameter UV .
- ▶ Cheapest move onto a circle: along the ray from its centre M , cost $||PM| - r|$.
- ▶ Right angle at U or V instead? Then P goes to a line – but we will see that *this never wins*.

Key idea

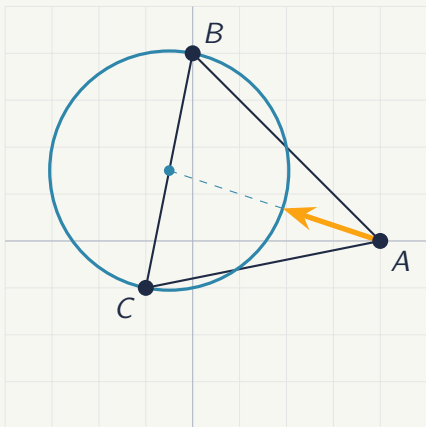
Answer = minimum over the 3 choices of P of the distance from P to the Thales circle over the other two.

Accident – sample 1: all three circles

$$A = (4, 0), B = (0, 4), C = (-1, -1)$$



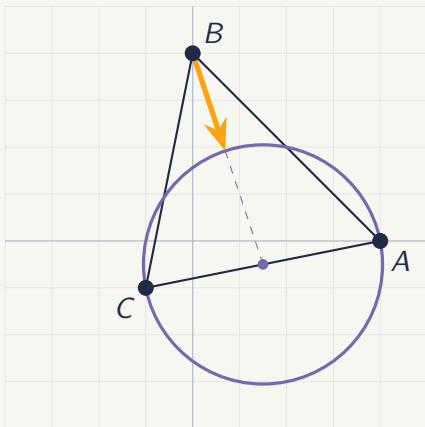
Accident – sample 1: all three circles



$$A = (4, 0), B = (0, 4), C = (-1, -1)$$

► move A, circle over BC:
 $\sqrt{22.5} - \frac{\sqrt{26}}{2} \approx 2.19$

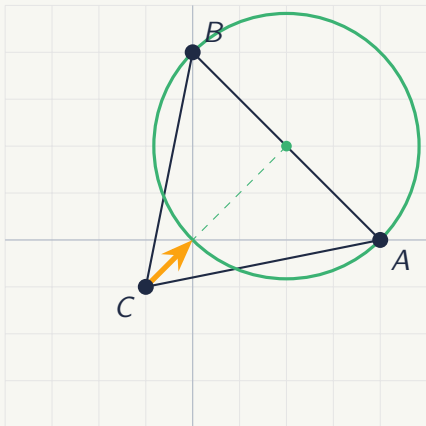
Accident – sample 1: all three circles



$$A = (4, 0), B = (0, 4), C = (-1, -1)$$

- ▶ move A, circle over BC:
 $\sqrt{22.5} - \frac{\sqrt{26}}{2} \approx 2.19$
- ▶ move B, circle over AC:
symmetric, also ≈ 2.19

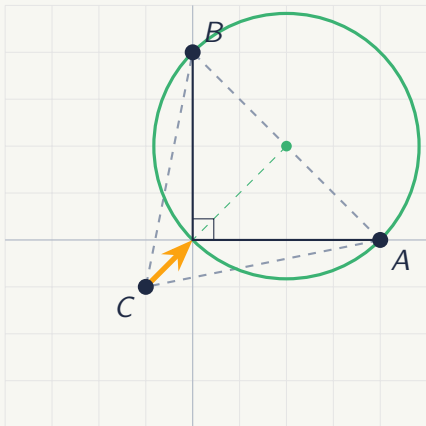
Accident – sample 1: all three circles



$$A = (4, 0), B = (0, 4), C = (-1, -1)$$

- ▶ move A, circle over BC :
 $\sqrt{22.5} - \frac{\sqrt{26}}{2} \approx 2.19$
- ▶ move B, circle over AC :
symmetric, also ≈ 2.19
- ▶ move C, circle over AB , $M = (2, 2)$:
 $|3\sqrt{2} - 2\sqrt{2}| = \sqrt{2} \approx 1.41$

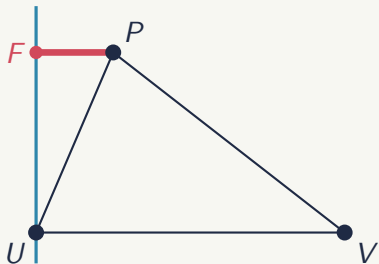
Accident – sample 1: all three circles



$$A = (4, 0), B = (0, 4), C = (-1, -1)$$

- ▶ move A, circle over BC :
 $\sqrt{22.5} - \frac{\sqrt{26}}{2} \approx 2.19$
- ▶ move B, circle over AC :
symmetric, also ≈ 2.19
- ▶ move C, circle over AB , $M = (2, 2)$:
 $|3\sqrt{2} - 2\sqrt{2}| = \sqrt{2} \approx 1.41$
- ▶ answer $\sqrt{2} \approx 1.41421$:
 $C \rightarrow (0, 0)$, right angle at the origin

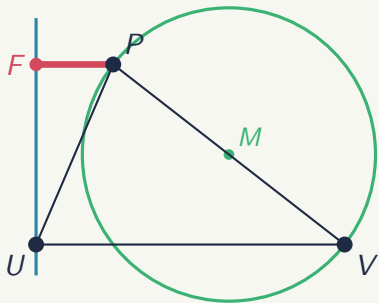
Accident – why the lines never win



Want the right angle at U . Two ways:

► move P to the line: cost $|PF| = 1.5$

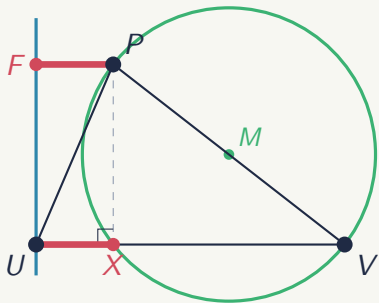
Accident – why the lines never win



Want the right angle at U . Two ways:

- ▶ move P to the line: cost $|PF| = 1.5$
- ▶ move U to the Thales circle over PV

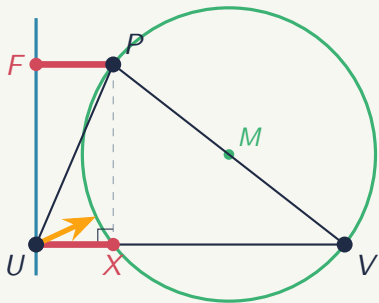
Accident – why the lines never win



Want the right angle at U . Two ways:

- ▶ move P to the line: cost $|PF| = 1.5$
- ▶ move U to the Thales circle over PV
- ▶ the circle meets UV again at X with $\angle PXV = 90^\circ$,
so $UFPX$ is a rectangle: $|UX| = |PF|$

Accident – why the lines never win



Want the right angle at U . Two ways:

- ▶ move P to the line: cost $|PF| = 1.5$
- ▶ move U to the Thales circle over PV
- ▶ the circle meets UV again at X with $\angle PXV = 90^\circ$,
so $UFPX$ is a rectangle: $|UX| = |PF|$
- ▶ X is only *one* point of the circle; going straight to the circle is never longer:
 $\approx 1.29 \leq 1.5 \Rightarrow$ the circle never loses.

Accident – summary

For each $P \in \{A, B, C\}$ with the other two U, V and $M = \frac{U+V}{2}$:

$$d_P = \left| |PM| - \frac{|UV|}{2} \right|, \quad \text{answer} = \min(d_A, d_B, d_C).$$

Complexity: $\mathcal{O}(1)$

three point-to-circle distances

Shopping

Miša buys n numbers and ORs together all their subset sums.

● ● ● ● ● bits · carries · prefix sums

Shopping

Task

Given $a_1, \dots, a_n$, compute $\bigoplus_{S \subseteq A} \sum_{a \in S} a$ (bitwise OR over all 2^n subset sums). $n \leq 10^5$, $\sum a_i \leq 10^{18}$.

Sample: $A = \{3, 5\}$

$\{\}$	0000
$\{3\}$	0011
$\{5\}$	0101
$\{3, 5\}$	1000
<hr/>	
OR	1111 = 15

Shopping

Task

Given $a_1, \dots, a_n$, compute $\bigoplus_{S \subseteq A} \sum_{a \in S} a$ (bitwise OR over all 2^n subset sums). $n \leq 10^5$, $\sum a_i \leq 10^{18}$.

Sample: $A = \{3, 5\}$

► 2^n subsets – no enumeration.

$\{\}$	0000
$\{3\}$	0011
$\{5\}$	0101
$\{3, 5\}$	1000
<hr/>	
OR	1111 = 15

Shopping

Task

Given $a_1, \dots, a_n$, compute $\bigoplus_{S \subseteq A} \sum_{a \in S} a$ (bitwise OR over all 2^n subset sums). $n \leq 10^5$, $\sum a_i \leq 10^{18}$.

Sample: $A = \{3, 5\}$

$\{\}$	0000
$\{3\}$	0011
$\{5\}$	0101
$\{3, 5\}$	1000
<hr/>	
OR	1111 = 15

- ▶ 2^n subsets – no enumeration.
- ▶ Sums up to 10^{18} – no subset-sum DP either.

Shopping

Task

Given $a_1, \dots, a_n$, compute $\bigoplus_{S \subseteq A} \sum_{a \in S} a$ (bitwise OR over all 2^n subset sums). $n \leq 10^5$, $\sum a_i \leq 10^{18}$.

Sample: $A = \{3, 5\}$

$\{\}$	0000
$\{3\}$	0011
$\{5\}$	0101
$\{3, 5\}$	1000
<hr/>	
OR	1111 = 15

- ▶ 2^n subsets – no enumeration.
- ▶ Sums up to 10^{18} – no subset-sum DP either.

Key idea

Decide every bit k *separately*:
can some subset sum have bit k set?

Shopping – one bit at a time

- ▶ Bit k of a *sum* depends only on the **lowest** $k+1$ **bits** of the summed elements – carries only go up, never down.

Shopping – one bit at a time

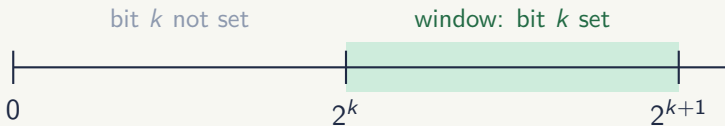
- ▶ Bit k of a *sum* depends only on the **lowest** $k+1$ **bits** of the summed elements – carries only go up, never down.
- ▶ So for bit k replace every number by $b_{i,k} = a_i \bmod 2^{k+1}$.

Shopping – one bit at a time

- ▶ Bit k of a *sum* depends only on the **lowest** $k+1$ **bits** of the summed elements – carries only go up, never down.
- ▶ So for bit k replace every number by $b_{i,k} = a_i \bmod 2^{k+1}$.
- ▶ Bit k of a sum s is set $\iff s \bmod 2^{k+1}$ lands in the **window** $[2^k, 2^{k+1})$.

Shopping – one bit at a time

- ▶ Bit k of a *sum* depends only on the **lowest** $k+1$ **bits** of the summed elements – carries only go up, never down.
- ▶ So for bit k replace every number by $b_{i,k} = a_i \bmod 2^{k+1}$.
- ▶ Bit k of a sum s is set $\iff s \bmod 2^{k+1}$ lands in the **window** $[2^k, 2^{k+1})$.



Shopping – when is bit k reachable?

Claim

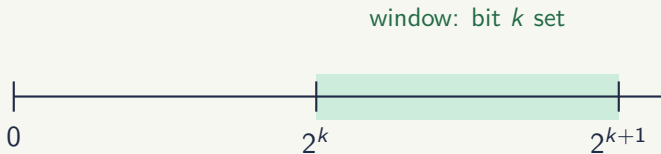
Some subset sum has bit k set $\iff \sum_i b_{i,k} \geq 2^k$, where $b_{i,k} = a_i \bmod 2^{k+1}$.

Shopping – when is bit k reachable?

Claim

Some subset sum has bit k set $\iff \sum_i b_{i,k} \geq 2^k$, where $b_{i,k} = a_i \bmod 2^{k+1}$.

$\Rightarrow$ If even *all* the $b_{i,k}$ together stay below 2^k , so does every subset – it never reaches the window.



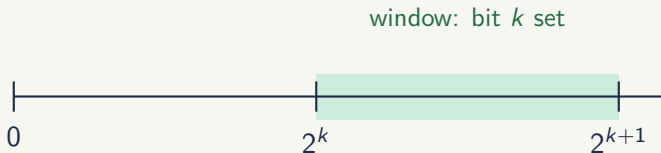
Shopping – when is bit k reachable?

Claim

Some subset sum has bit k set $\iff \sum_i b_{i,k} \geq 2^k$, where $b_{i,k} = a_i \bmod 2^{k+1}$.

$\Rightarrow$ If even *all* the $b_{i,k}$ together stay below 2^k , so does every subset – it never reaches the window.

$\Leftarrow$ Some $b_{i,k} \geq 2^k$? Then a_i *alone* has bit k .

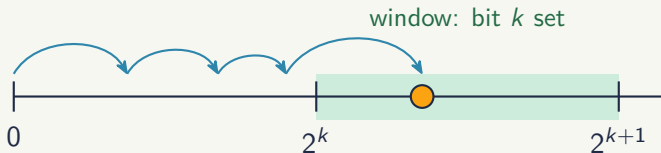


Shopping – when is bit k reachable?

Claim

Some subset sum has bit k set $\iff \sum_i b_{i,k} \geq 2^k$, where $b_{i,k} = a_i \bmod 2^{k+1}$.

- $\Rightarrow$ If even *all* the $b_{i,k}$ together stay below 2^k , so does every subset – it never reaches the window.
- $\Leftarrow$ Some $b_{i,k} \geq 2^k$? Then a_i *alone* has bit k . Otherwise every $b_{i,k} < 2^k$: add them one by one. The running total starts at 0, ends $\geq 2^k$, and every step is **shorter than the window** – it cannot jump over it.



Shopping – algorithm

1. For $k = 0, \dots, 59$: // $2^{60} > 10^{18}$
2. $b_{i,k} \leftarrow a_i \bmod 2^{k+1}$ for every i
3. $s_k \leftarrow \sum_i b_{i,k}$
4. if $s_k \geq 2^k$: set bit k .

No overflow: $s_k \leq \sum_i a_i \leq 10^{18}$ fits in 64 bits
– that is what the sum limit is for.

Shopping – algorithm

1. For $k = 0, \dots, 59$: // $2^{60} > 10^{18}$
2. $b_{i,k} \leftarrow a_i \bmod 2^{k+1}$ for every i
3. $s_k \leftarrow \sum_i b_{i,k}$
4. if $s_k \geq 2^k$: set bit k .

No overflow: $s_k \leq \sum_i a_i \leq 10^{18}$ fits in 64 bits
– that is what the sum limit is for.

Sample 3: $A = \{2, 9, 32\}$

k	$b_{i,k} = a_i \bmod 2^{k+1}$	s_k	vs. 2^k	bit
0	0, 1, 0	1	≥ 1	1
1	2, 1, 0	3	≥ 2	1
2	2, 1, 0	3	< 4	0
3	2, 9, 0	11	≥ 8	1
4	2, 9, 0	11	< 16	0
5	2, 9, 32	43	≥ 32	1
answer $101011_2 = 43 \checkmark$				

Shopping – algorithm

1. For $k = 0, \dots, 59$: // $2^{60} > 10^{18}$
2. $b_{i,k} \leftarrow a_i \bmod 2^{k+1}$ for every i
3. $s_k \leftarrow \sum_i b_{i,k}$
4. if $s_k \geq 2^k$: set bit k .

No overflow: $s_k \leq \sum_i a_i \leq 10^{18}$ fits in 64 bits
– that is what the sum limit is for.

Sample 3: $A = \{2, 9, 32\}$

k	$b_{i,k} = a_i \bmod 2^{k+1}$	s_k	vs. 2^k	bit
0	0, 1, 0	1	≥ 1	1
1	2, 1, 0	3	≥ 2	1
2	2, 1, 0	3	< 4	0
3	2, 9, 0	11	≥ 8	1
4	2, 9, 0	11	< 16	0
5	2, 9, 32	43	≥ 32	1

answer $101011_2 = 43 \checkmark$

Complexity: $\mathcal{O}(60 \cdot n)$

Shopping – even shorter: prefix sums

Answer

OR of **all elements** and **all prefix sums** $a_1 + \dots + a_j$ (in any order).

Each is a subset sum, so no extra bits. Missing none? Take a reachable bit k :

Shopping – even shorter: prefix sums

Answer

OR of **all elements** and **all prefix sums** $a_1 + \dots + a_j$ (in any order).

Each is a subset sum, so no extra bits. Missing none? Take a reachable bit k :

- ▶ **Some element has bit k :** that element alone sets it.

Shopping – even shorter: prefix sums

Answer

OR of **all elements** and **all prefix sums** $a_1 + \dots + a_j$ (in any order).

Each is a subset sum, so no extra bits. Missing none? Take a reachable bit k :

- ▶ **Some element has bit k :** that element alone sets it.
- ▶ **No element has bit k :** add the elements one by one and watch bit k .
 - **It must come.** Some subset sum has bit k , but none of its elements does – so it comes from a carry: the elements' bits below k add up to $\geq 2^k$. All elements together add up to even more – more than fits below bit k , so a carry into bit k has to happen.

Shopping – even shorter: prefix sums

Answer

OR of **all elements** and **all prefix sums** $a_1 + \dots + a_j$ (in any order).

Each is a subset sum, so no extra bits. Missing none? Take a reachable bit k :

- ▶ **Some element has bit k :** that element alone sets it.
- ▶ **No element has bit k :** add the elements one by one and watch bit k .
 - **It must come.** Some subset sum has bit k , but none of its elements does – so it comes from a carry: the elements' bits below k add up to $\geq 2^k$. All elements together add up to even more – more than fits below bit k , so a carry into bit k has to happen.
 - **It can't be skipped.** While bit k of the total is 0, column k adds $0 + 0$, so a carry into it just sets bit k – it can't pass through.

Shopping – prefix sums: example and code

$A = \{3, 5\}$: bit 3 is in no element

$$\begin{array}{r} 0011 \quad (3) \\ + 0101 \quad (5) \\ \hline 1000 \quad (8) \end{array}$$

column 3: $0 + 0 + \text{carry} = 1$

$$3 \mid 5 \mid 3 \mid 8 = 1111_2 = 15 \checkmark$$

Shopping – prefix sums: example and code

$A = \{3, 5\}$: bit 3 is in no element

$$\begin{array}{r} 0011 \quad (3) \\ + 0101 \quad (5) \\ \hline 1000 \quad (8) \end{array}$$

column 3: $0 + 0 + \text{carry} = 1$

$$3 \mid 5 \mid 3 \mid 8 = 1111_2 = 15 \checkmark$$

Whole solution

```
ll ans = 0, pre = 0;
for (ll x : a) {
    pre += x;
    ans |= x | pre;
}
```

$\text{pre} \leq \sum a_i \leq 10^{18}$ fits in a long long.

Shopping – prefix sums: example and code

$A = \{3, 5\}$: bit 3 is in no element

$$\begin{array}{r} 0011 \quad (3) \\ + 0101 \quad (5) \\ \hline 1000 \quad (8) \end{array}$$

column 3: $0 + 0 + \text{carry} = 1$

$$3 \mid 5 \mid 3 \mid 8 = 1111_2 = 15 \checkmark$$

Whole solution

```
ll ans = 0, pre = 0;
for (ll x : a) {
    pre += x;
    ans |= x | pre;
}
```

$\text{pre} \leq \sum a_i \leq 10^{18}$ fits in a long long.

Complexity: $\mathcal{O}(n)$

Shopping – the traps

Pitfalls

- ▶ OR of the elements, or of pairs, misses the carries: $\{1, 1, 1, 1\}$ gives 1 resp. 3, answer 7.

Shopping – the traps

Pitfalls

- ▶ OR of the elements, or of pairs, misses the carries: $\{1, 1, 1, 1\}$ gives 1 resp. 3, answer 7.
- ▶ Prefix sums *without* the elements: $\{1, 3\}$ gives $1 \mid 4 = 5$, answer 7.

Shopping – the traps

Pitfalls

- ▶ OR of the elements, or of pairs, misses the carries: $\{1, 1, 1, 1\}$ gives 1 resp. 3, answer 7.
- ▶ Prefix sums *without* the elements: $\{1, 3\}$ gives $1 \mid 4 = 5$, answer 7.
- ▶ “All bits from the lowest set bit up to the top of the sum”: right on random data, wrong for different magnitudes: $\{1, 4\}$ gives 7, answer 5.

Shopping – the traps

Pitfalls

- ▶ OR of the elements, or of pairs, misses the carries: $\{1, 1, 1, 1\}$ gives 1 resp. 3, answer 7.
- ▶ Prefix sums *without* the elements: $\{1, 3\}$ gives $1 \mid 4 = 5$, answer 7.
- ▶ “All bits from the lowest set bit up to the top of the sum”: right on random data, wrong for different magnitudes: $\{1, 4\}$ gives 7, answer 5.
- ▶ Write $1LL \ll k$; $1 \ll k$ overflows from bit 31 on.

Shopping – the traps

Pitfalls

- ▶ OR of the elements, or of pairs, misses the carries: $\{1, 1, 1, 1\}$ gives 1 resp. 3, answer 7.
- ▶ Prefix sums *without* the elements: $\{1, 3\}$ gives $1 \mid 4 = 5$, answer 7.
- ▶ “All bits from the lowest set bit up to the top of the sum”: right on random data, wrong for different magnitudes: $\{1, 4\}$ gives 7, answer 5.
- ▶ Write $1LL \ll k$; $1 \ll k$ overflows from bit 31 on.
- ▶ Go up to bit 59: $2^{59} \approx 5.8 \cdot 10^{17} \leq 10^{18}$.

Nightmare

Miša dreams of a red-blue graph: no cycle may have a single odd-coloured edge.

Nightmare

Task

Graph with red and blue edges. Print NO iff some simple cycle has *exactly one* red or *exactly one* blue edge; otherwise print YES.

Nightmare

Task

Graph with red and blue edges. Print NO iff some simple cycle has *exactly one* red or *exactly one* blue edge; otherwise print YES.

- ▶ $n, m \leq 3 \cdot 10^5$, simple graph.
- ▶ Exponentially many cycles – we cannot look at them.
- ▶ Turn the question around: fix the *lonely* edge.

Nightmare – the lonely edge

- ▶ A bad cycle = one red edge uv + all other edges blue.
- ▶ The rest of the cycle is a **blue path** from u to v .
- ▶ Conversely, any blue path $u \rightsquigarrow v$ (taken simple) + red edge uv is a simple cycle.
(no multi-edges, so the path has ≥ 2 edges)

Nightmare – the lonely edge

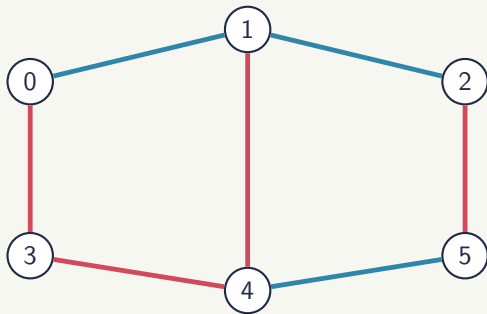
- ▶ A bad cycle = one red edge uv + all other edges blue.
- ▶ The rest of the cycle is a **blue path** from u to v .
- ▶ Conversely, any blue path $u \rightsquigarrow v$ (taken simple) + red edge uv is a simple cycle.
(no multi-edges, so the path has ≥ 2 edges)

Key idea

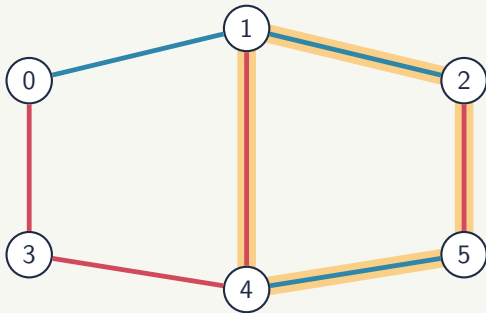
A red edge uv is bad $\iff u, v$ lie in the same **blue component**.
Symmetrically for blue edges and **red components**.

Nightmare – example

Three blue and four red edges.

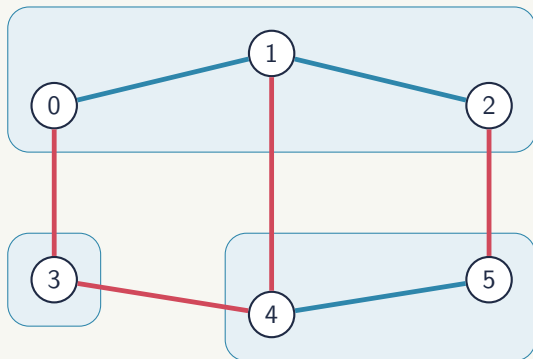


Nightmare – example



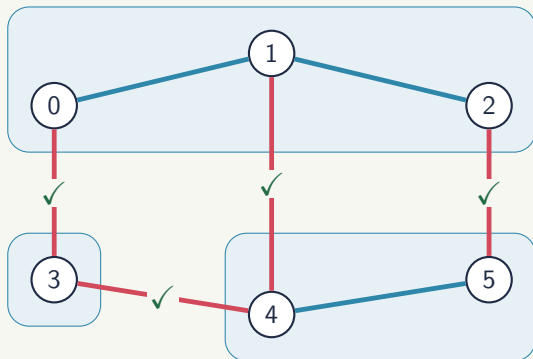
Cycle 1 2 5 4: 2 blue + 2 red
– mixed, but allowed.

Nightmare – example



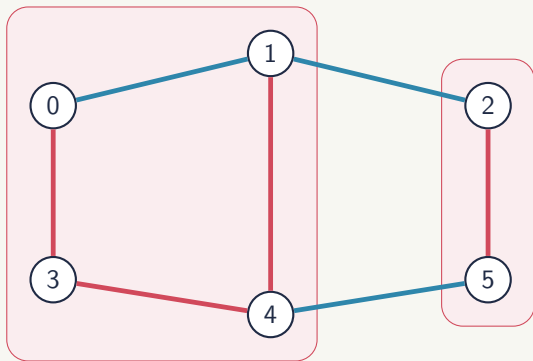
Blue components:
 $\{0, 1, 2\}$, $\{4, 5\}$, $\{3\}$.

Nightmare – example



Every red edge joins two *different* blue components
no lonely red edge.

Nightmare – example

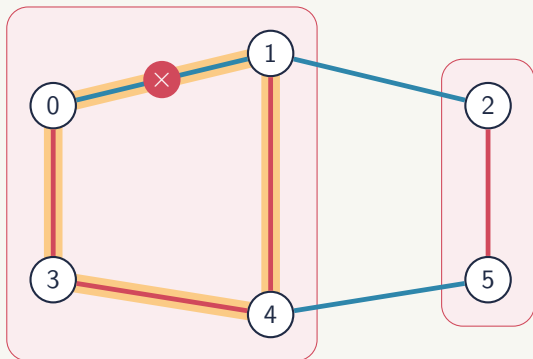


Red components:

$\{0, 1, 3, 4\}, \{2, 5\}.$

Now check the blue edges ...

Nightmare – example



Blue edge 0 1 lies *inside* the red component $\{0, 1, 3, 4\}$.

Cycle 0 1 4 3: 1 blue + 3 red.

Answer: NO

Nightmare – algorithm

1. Union-find (or BFS) over the red edges only $\rightarrow$ red components.
2. Union-find over the blue edges only $\rightarrow$ blue components.
3. NO iff some red edge has both ends in one blue component, or some blue edge has both ends in one red component.

Nightmare – algorithm

1. Union-find (or BFS) over the red edges only $\rightarrow$ red components.
2. Union-find over the blue edges only $\rightarrow$ blue components.
3. NO iff some red edge has both ends in one blue component, or some blue edge has both ends in one red component.

Complexity: $\mathcal{O}((n + m) \alpha(n))$

two DSU passes over the edges

Nightmare – algorithm

1. Union-find (or BFS) over the red edges only $\rightarrow$ red components.
2. Union-find over the blue edges only $\rightarrow$ blue components.
3. NO iff some red edge has both ends in one blue component, or some blue edge has both ends in one red component.

Complexity: $\mathcal{O}((n + m) \alpha(n))$

two DSU passes over the edges

Pitfalls

- ▶ Check *both* colours – the lonely edge can be red or blue.
- ▶ Searching cycles (DFS over all cycles, random walks) is exponential or misses the bad one.

Birthday

Míša's broken division machine only accepts palindromes.

Birthday

Task

Count palindromes with at most k digits divisible by the prime p , modulo $10^9 + 7$.

Birthday

Task

Count palindromes with at most k digits divisible by the prime p , modulo $10^9 + 7$.

- ▶ $k, p \leq 2000$ – numbers have up to 2000 digits.

Birthday

Task

Count palindromes with at most k digits divisible by the prime p , modulo $10^9 + 7$.

- ▶ $k, p \leq 2000$ – numbers have up to 2000 digits.
- ▶ There are $\approx 10^{k/2}$ palindromes, $10^k/p$ multiples of p – no enumeration.

Birthday – grow from the middle

- ▶ A palindrome is a core x of length L wrapped by the same digit d on both sides:

$$\underbrace{d x d}_{L+2} = d \cdot 10^{L+1} + 10x + d.$$

Birthday – grow from the middle

- ▶ A palindrome is a core x of length L wrapped by the same digit d on both sides:

$$\underbrace{d \times d}_{L+2} = d \cdot 10^{L+1} + 10x + d.$$

- ▶ Divisibility needs only $x \bmod p \Rightarrow \text{state} = (L, r = x \bmod p)$.

Birthday – grow from the middle

- ▶ A palindrome is a core x of length L wrapped by the same digit d on both sides:

$$\underbrace{d x d}_{L+2} = d \cdot 10^{L+1} + 10x + d.$$

- ▶ Divisibility needs only $x \bmod p \Rightarrow \text{state} = (L, r = x \bmod p)$.
- ▶ $\text{cnt}_L[r]$ = number of cores of length L with residue r ; each has 10 extensions:

$$r' = (d \cdot 10^{L+1} + 10r + d) \bmod p, \quad d = 0..9.$$

Birthday – grow from the middle

- ▶ A palindrome is a core x of length L wrapped by the same digit d on both sides:

$$\underbrace{d x d}_{L+2} = d \cdot 10^{L+1} + 10x + d.$$

- ▶ Divisibility needs only $x \bmod p \Rightarrow \text{state} = (L, r = x \bmod p)$.
- ▶ $\text{cnt}_L[r]$ = number of cores of length L with residue r ; each has 10 extensions:

$$r' = (d \cdot 10^{L+1} + 10r + d) \bmod p, \quad d = 0..9.$$

- ▶ Reuse the leftmost digit from last iteration

$$d \cdot 10^{L+1} = d \cdot 10^{L-1} \cdot 100$$

Birthday – algorithm

1. DP with indexes current length L and remainder class r

Birthday – algorithm

1. DP with indexes current length L and remainder class r
2. Initialization: **even** lengths start from the empty core ($cnt_0[0] = 1$), **odd** lengths from the ten 1-digit cores 0..9.

Birthday – algorithm

1. DP with indexes current length L and remainder class r
2. Initialization: **even** lengths start from the empty core ($cnt_0[0] = 1$), **odd** lengths from the ten 1-digit cores 0..9.
3. Extend each state while $L + 2 \leq k$; after every step add the counts of $r' = 0$ reached with $d \neq 0$.

Birthday – algorithm

1. DP with indexes current length L and remainder class r
2. Initialization: **even** lengths start from the empty core ($cnt_0[0] = 1$), **odd** lengths from the ten 1-digit cores 0..9.
3. Extend each state while $L + 2 \leq k$; after every step add the counts of $r' = 0$ reached with $d \neq 0$.

Complexity: $\mathcal{O}(10 \cdot k \cdot p)$

$$\approx k \text{ steps} \times p \text{ residues} \times 10 \text{ digits} \approx 4 \cdot 10^7$$

Lanterns

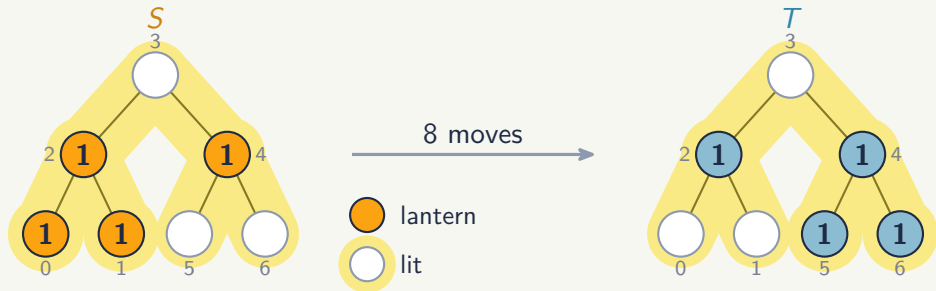
Miša rearranges the lanterns on his tree without ever letting a vertex fall into darkness.

● ● ● ● ● trees · greedy · meet in the middle

Lanterns

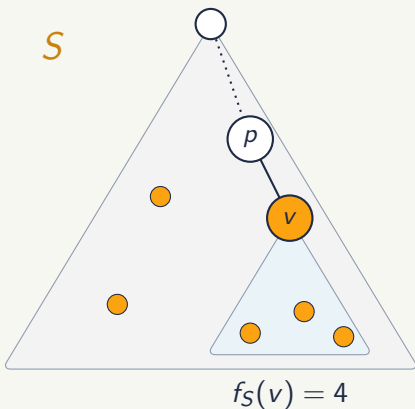
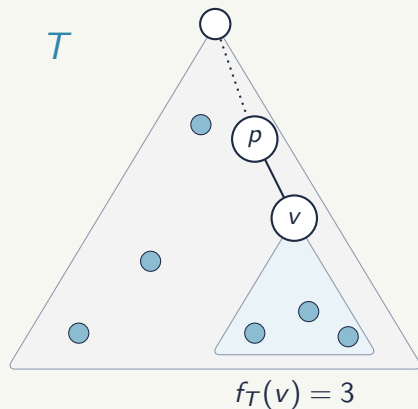
Task

Move k lanterns on a tree from dominating set S to dominating set T , $\ell \geq 1$ lanterns along one edge per command, always dominating. Minimise $\sum \ell$.



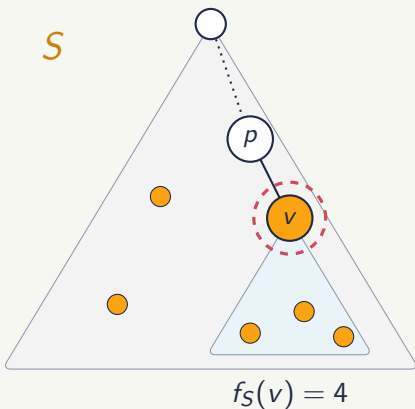
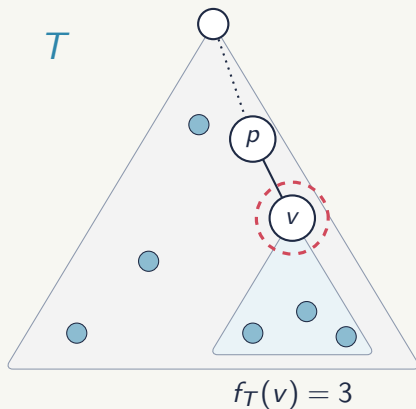
Dominating: every vertex has a lantern *or a neighbour with one*.

Lanterns – lower bound



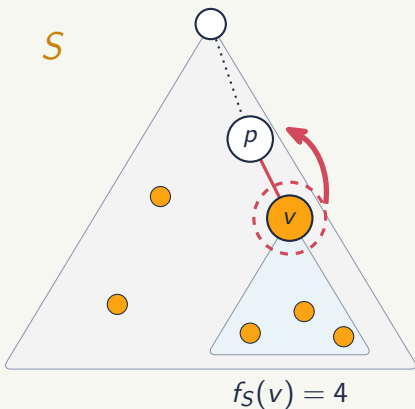
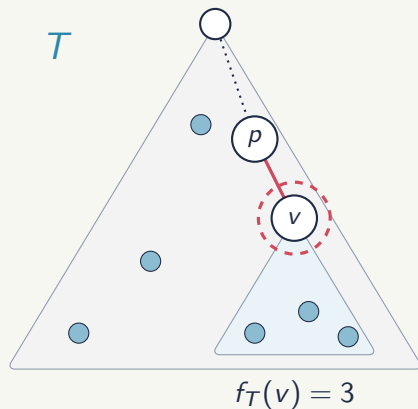
$f_S(v), f_T(v) =$ lanterns of S, T in the subtree of v .

Lanterns – lower bound



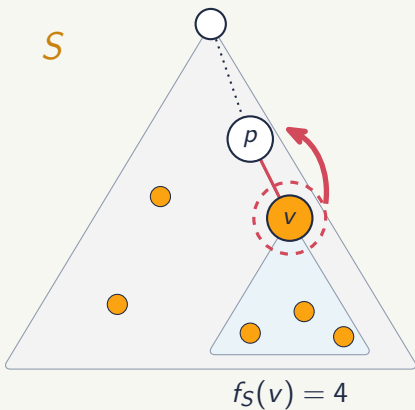
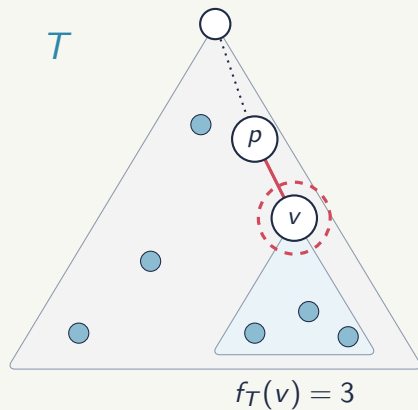
$\Delta(v) = f_S(v) - f_T(v) = 1$: S has a lantern there that T does not.

Lanterns – lower bound



At least $|\Delta(v)|$ lanterns must cross the red edge.

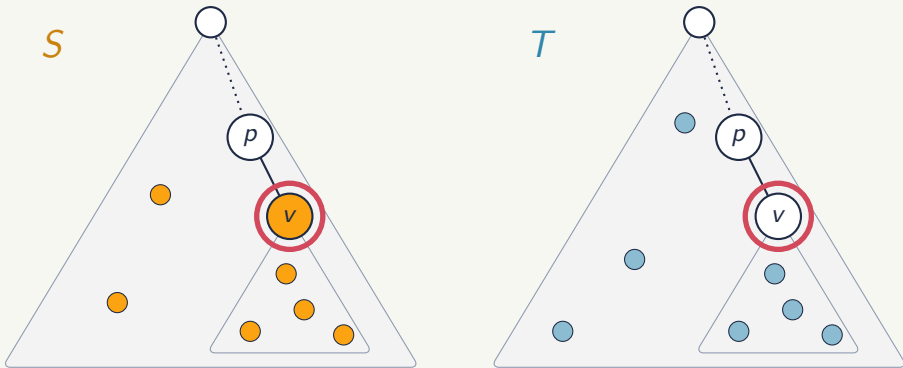
Lanterns – lower bound



Key idea

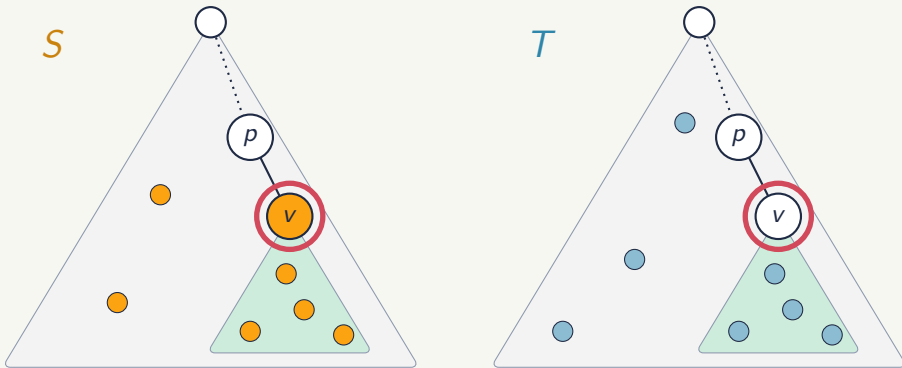
$$\text{answer} \geq \sum_v |\Delta(v)|$$

Lanterns – key observation



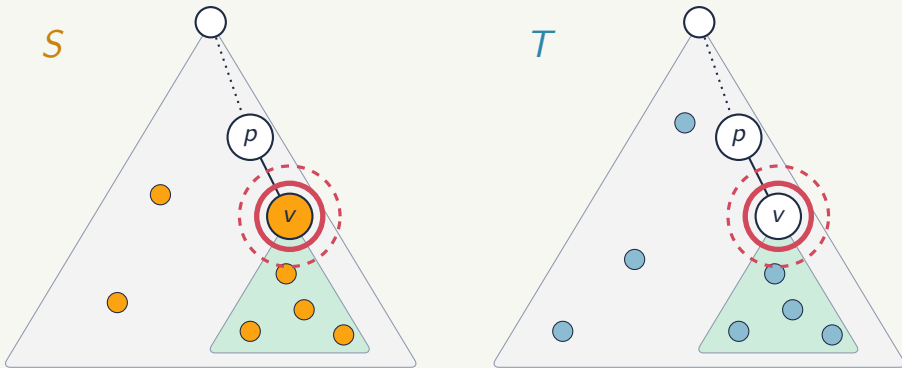
Take the **lowest** v with $\Delta(v) \neq 0$.

Lanterns – key observation



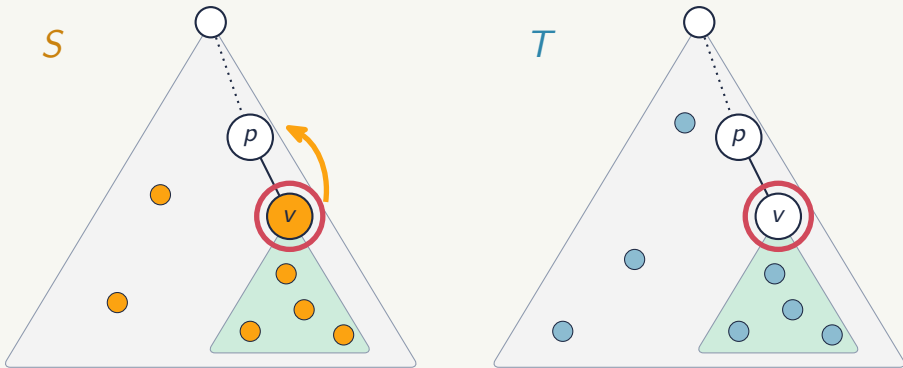
Below v all $\Delta = 0 \Rightarrow$ the same lanterns in S and T .

Lanterns – key observation



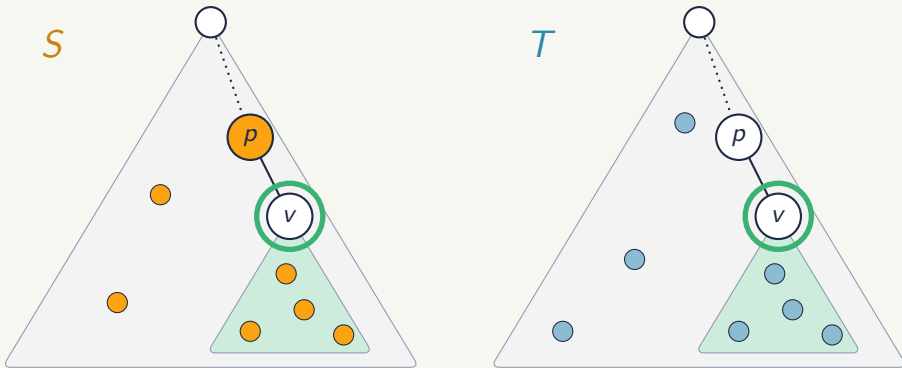
$\Rightarrow v$ itself differs: S has a lantern on v , T does not.

Lanterns – key observation



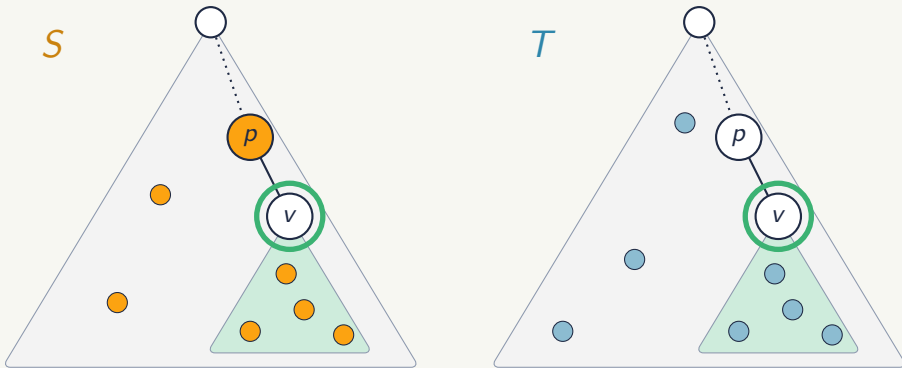
S must send it up anyway. Do it **now**.

Lanterns – key observation



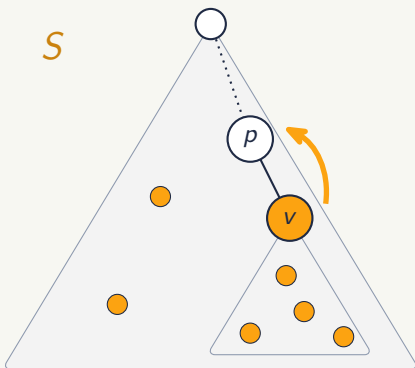
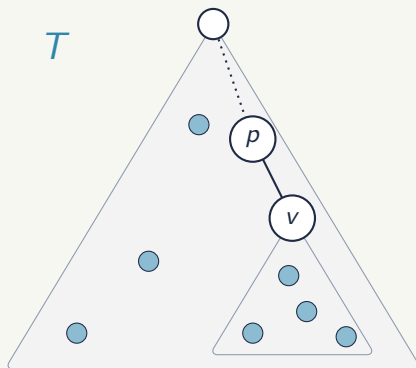
Now $\Delta(v) = 0$, every other Δ unchanged.

Lanterns – key observation



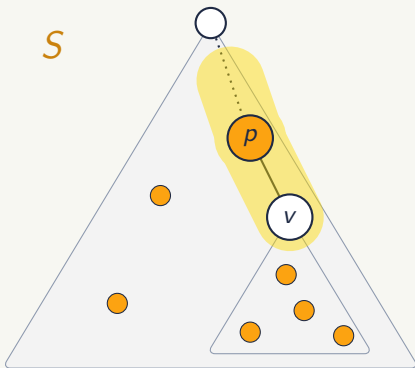
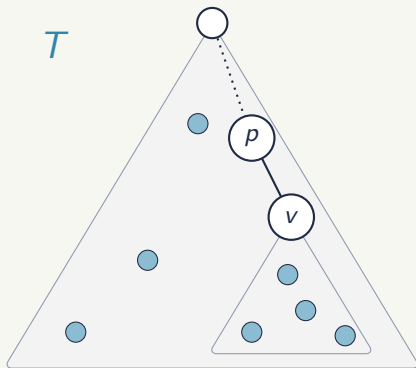
$\Delta(v) < 0$: the same move, on T 's side.

Lanterns – still dominating?



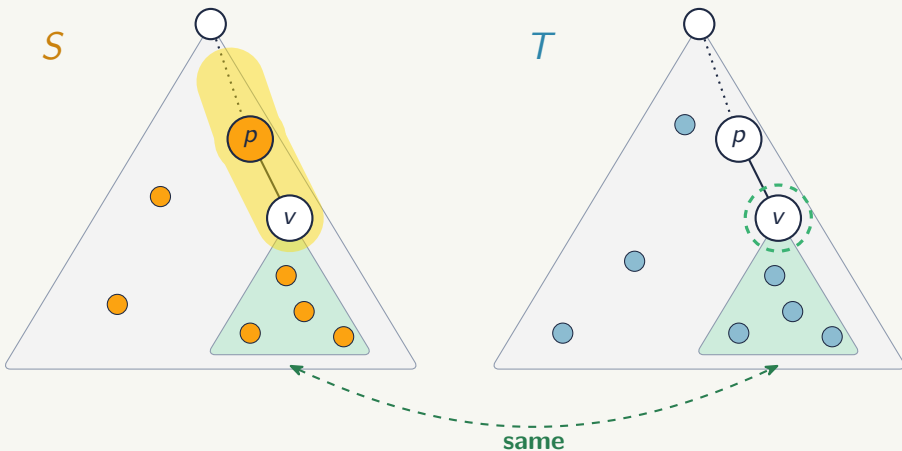
The move $v \rightarrow p$: is *S* still dominating?

Lanterns – still dominating?



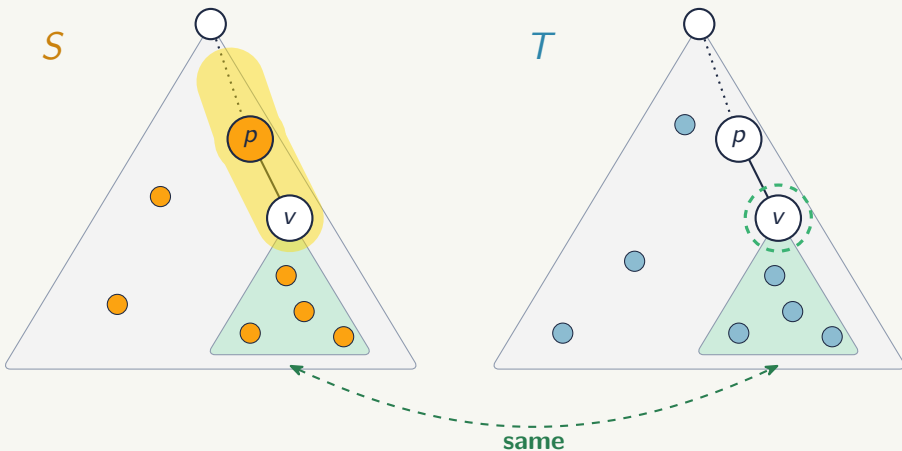
v is lit by p .

Lanterns – still dominating?



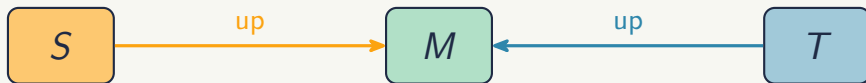
Below v it is exactly T , where v is empty too $\Rightarrow$ all lit.

Lanterns – still dominating?



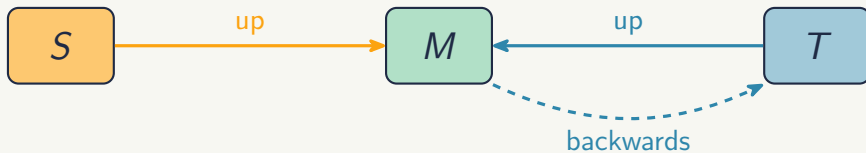
Above v nothing lost a lantern. Same for T 's moves.

Lanterns – meet in the middle



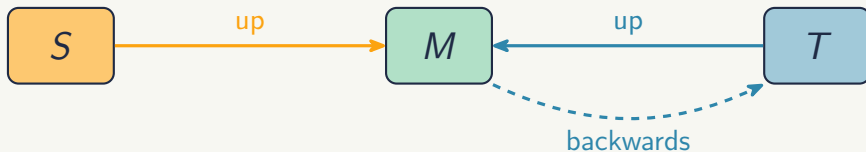
Repeat until all $\Delta = 0$: both runs meet in M .

Lanterns – meet in the middle



Play T 's run backwards: $M \rightarrow T$.

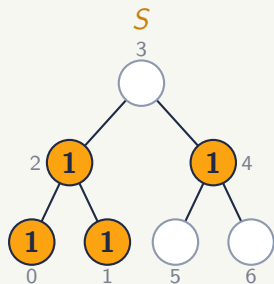
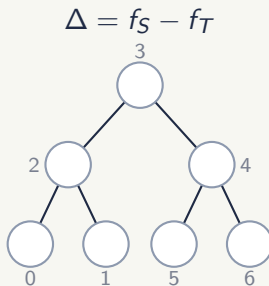
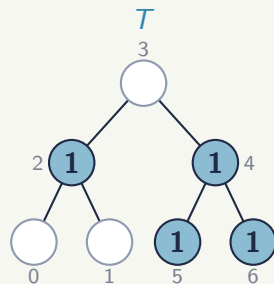
Lanterns – meet in the middle



Key idea

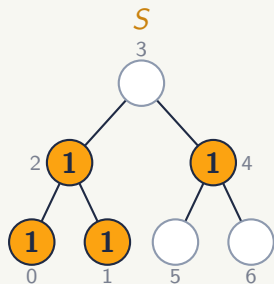
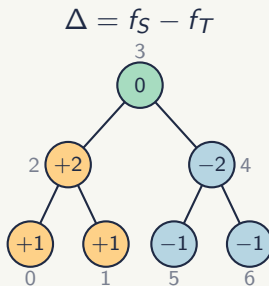
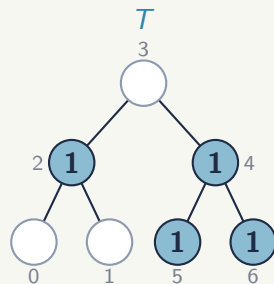
$\sum |\Delta|$ moves = lower bound $\Rightarrow$ optimal. Never -1 .

Lanterns – sample 2, rooted at 3



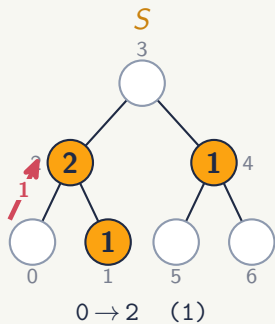
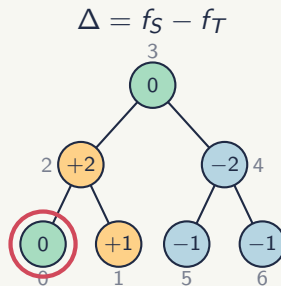
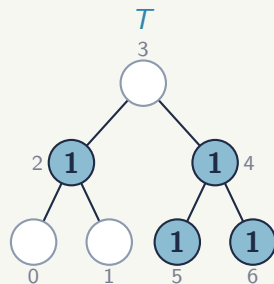
Start: $S = \{0, 1, 2, 4\}$, $T = \{2, 4, 5, 6\}$.

Lanterns – sample 2, rooted at 3



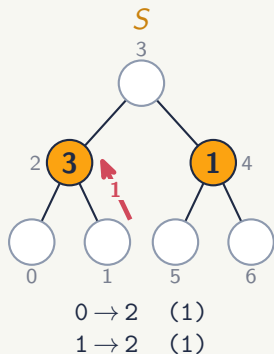
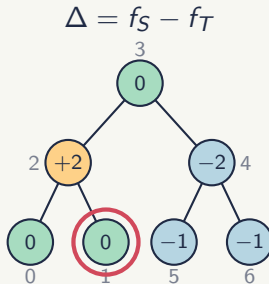
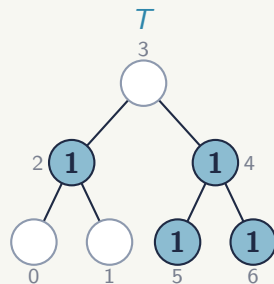
Lower bound $\sum |\Delta| = 1 + 1 + 2 + 1 + 1 + 2 = \mathbf{8}$.

Lanterns – sample 2, rooted at 3



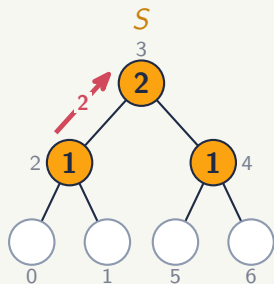
Vertex 0, $\Delta = +1 \Rightarrow S$ sends 1 up.

Lanterns – sample 2, rooted at 3

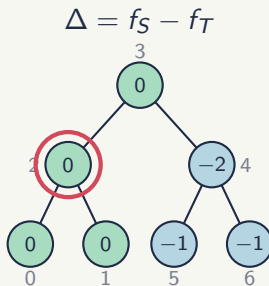
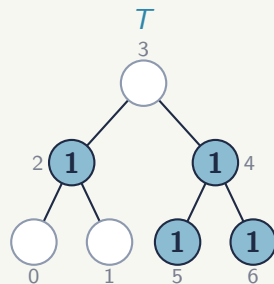


Vertex 1, $\Delta = +1 \Rightarrow S$ sends 1 up.

Lanterns – sample 2, rooted at 3

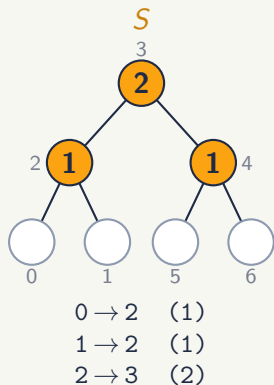
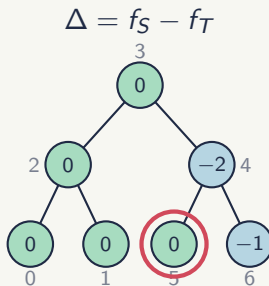
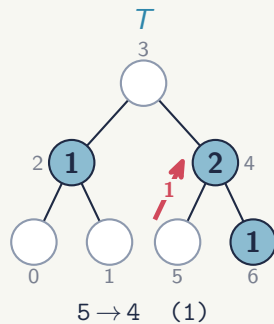


$0 \rightarrow 2$ (1)
 $1 \rightarrow 2$ (1)
 $2 \rightarrow 3$ (2)



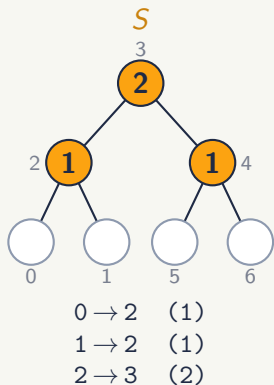
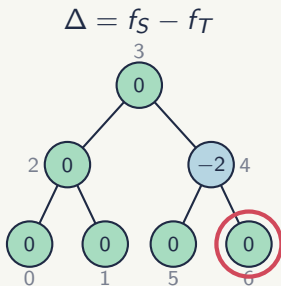
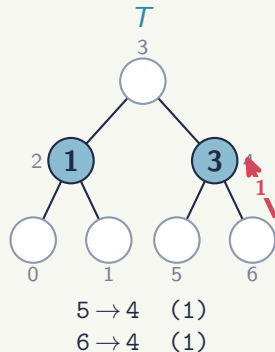
Vertex 2, $\Delta = +2 \Rightarrow S$ sends 2 up.

Lanterns – sample 2, rooted at 3



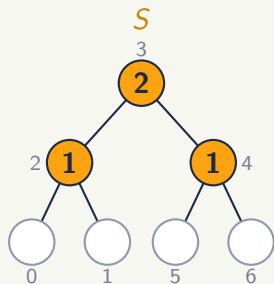
Vertex 5, $\Delta = -1 \Rightarrow T$ sends 1 up.

Lanterns – sample 2, rooted at 3

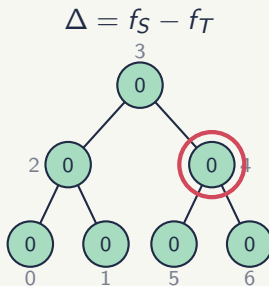


Vertex 6, $\Delta = -1 \Rightarrow T$ sends 1 up.

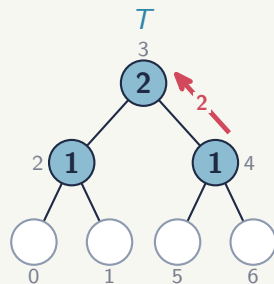
Lanterns – sample 2, rooted at 3



$0 \rightarrow 2$ (1)
 $1 \rightarrow 2$ (1)
 $2 \rightarrow 3$ (2)



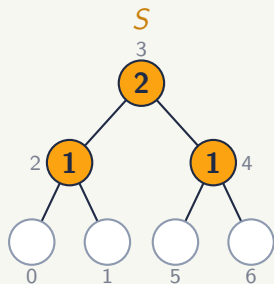
$S = T = M$



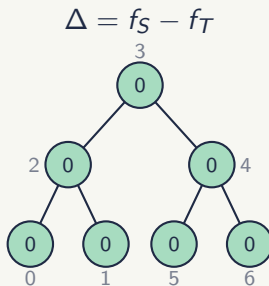
$5 \rightarrow 4$ (1)
 $6 \rightarrow 4$ (1)
 $4 \rightarrow 3$ (2)

Vertex 4, $\Delta = -2 \Rightarrow T$ sends 2 up. They meet.

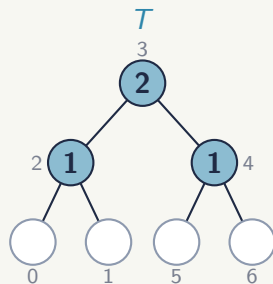
Lanterns – sample 2, rooted at 3



$0 \rightarrow 2$ (1)
 $1 \rightarrow 2$ (1)
 $2 \rightarrow 3$ (2)



$$S = T = M$$



$5 \rightarrow 4$ (1)
 $6 \rightarrow 4$ (1)
 $4 \rightarrow 3$ (2)

Output: S column, then T column reversed & flipped. 8 moves.

Lanterns – one DFS

Key idea

Post-order DFS: a finished v is always a lowest unresolved vertex.

Lanterns – one DFS

Key idea

Post-order DFS: a finished v is always a lowest unresolved vertex.

$$d = s[v] - t[v]$$

$d > 0$: output $v \rightarrow p$ (d)

$d < 0$: record $p \rightarrow v$ ($-d$)

Lanterns – one DFS

Key idea

Post-order DFS: a finished v is always a lowest unresolved vertex.

$$d = s[v] - t[v]$$

$d > 0$: output $v \rightarrow p$ (d)

$d < 0$: record $p \rightarrow v$ ($-d$)

Print the records reversed.

Lanterns – one DFS

Key idea

Post-order DFS: a finished v is always a lowest unresolved vertex.

$$d = s[v] - t[v]$$

$d > 0$: output $v \rightarrow p$ (d)

$d < 0$: record $p \rightarrow v$ ($-d$)

Print the records reversed.

Complexity: $\mathcal{O}(n)$

Lanterns – the traps

Pitfalls

- ▶ There is no -1 case.

Lanterns – the traps

Pitfalls

- ▶ There is no -1 case.
- ▶ Move all $|\Delta|$ lanterns at once (up to $\approx 5 \cdot 10^9$ moves).

Lanterns – the traps

Pitfalls

- ▶ There is no -1 case.
- ▶ Move all $|\Delta|$ lanterns at once (up to $\approx 5 \cdot 10^9$ moves).
- ▶ Paths of $2 \cdot 10^5$: recursive DFS overflows the stack.

Housing

Miša wants a house whose price is exactly the median of what is left of the street.

● ● ● ● ● medians · ± 1 trick · segment tree

Housing – when is m a median?

Task

For every i : is a_i the median of some window $[\ell, r] \ni i$ of odd length ≥ 3 ?



Housing – when is m a median?

Task

For every i : is a_i the median of some window $[\ell, r] \ni i$ of odd length ≥ 3 ?



Housing – when is m a median?

Task

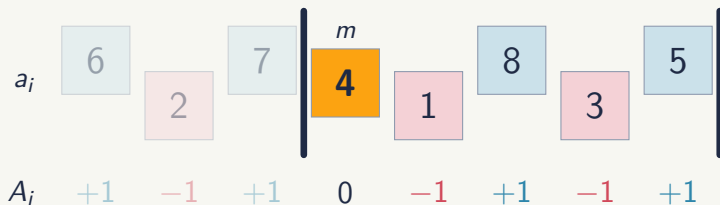
For every i : is a_i the median of some window $[\ell, r] \ni i$ of odd length ≥ 3 ?



Housing – when is m a median?

Task

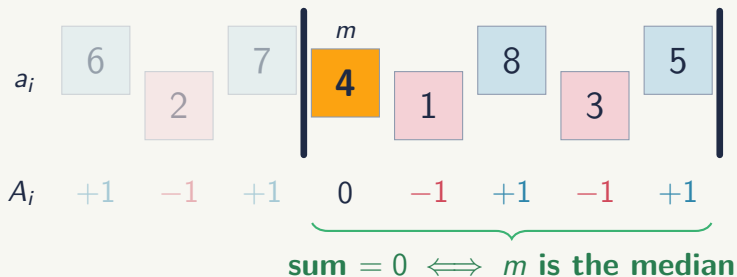
For every i : is a_i the median of some window $[\ell, r] \ni i$ of odd length ≥ 3 ?



Housing – when is m a median?

Task

For every i : is a_i the median of some window $[\ell, r] \ni i$ of odd length ≥ 3 ?

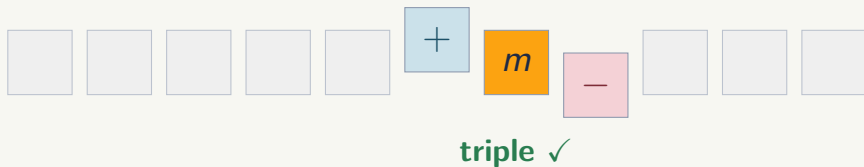


Housing – key observation



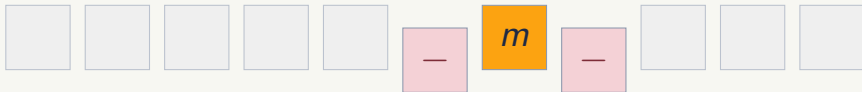
Candidate m .

Housing – key observation



$+ m -$ or $- m + \Rightarrow$ done.

Housing – key observation

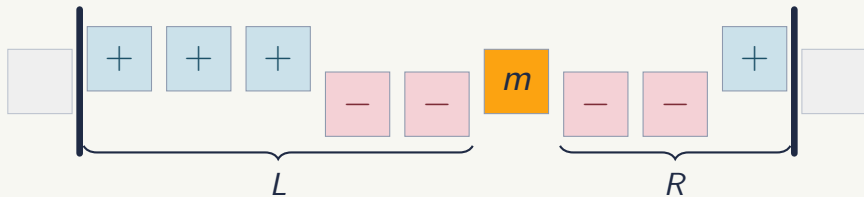


WLOG

neighbours – m –

Otherwise WLOG – m –.

Housing – key observation

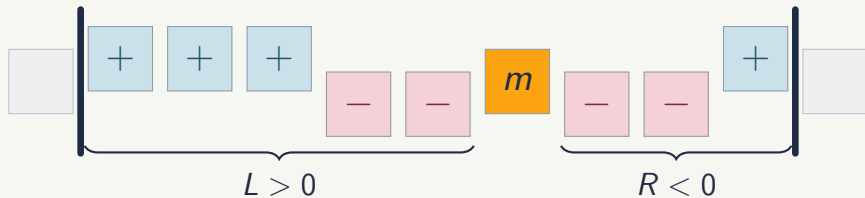


WLOG

neighbours – m –

A median window needs $L + R = 0$.

Housing – key observation

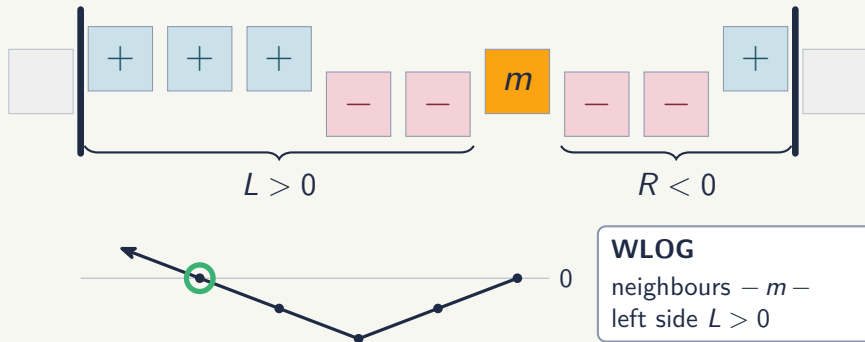


WLOG

neighbours $- m -$
left side $L > 0$

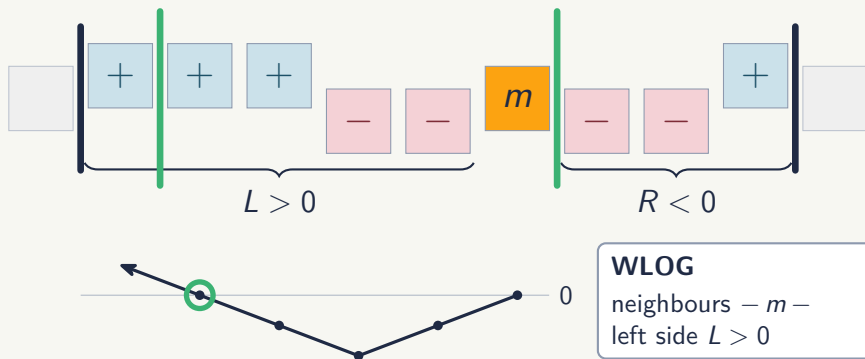
So one side has sum > 0 .

Housing – key observation



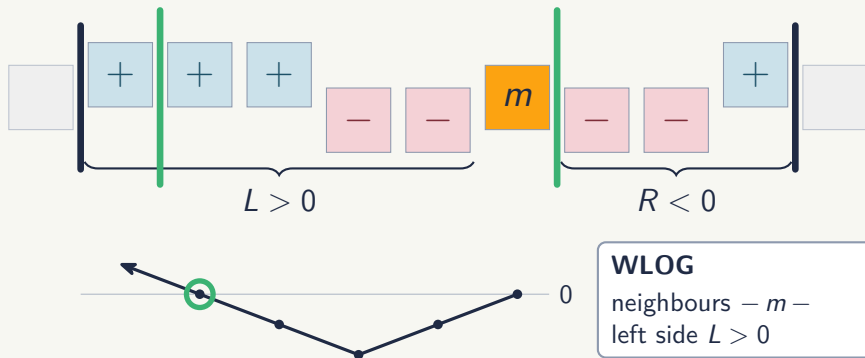
It starts with $-$ and ends $> 0 \Rightarrow$ it crosses 0.

Housing – key observation



$\Rightarrow$ a window with m at its **edge** works.

Housing – key observation

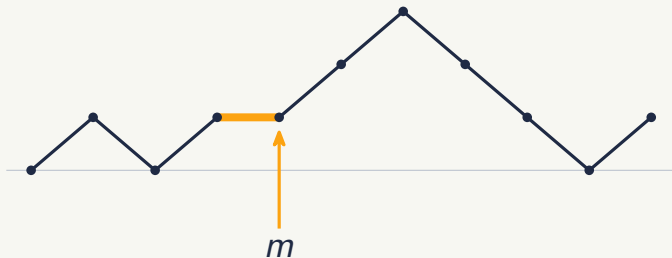


Key idea

Test only the **triple** and windows with m at an **edge**.

Housing – m at an end: prefix sums

$$P_j = A_1 + \cdots + A_j, \quad P_0 = 0$$



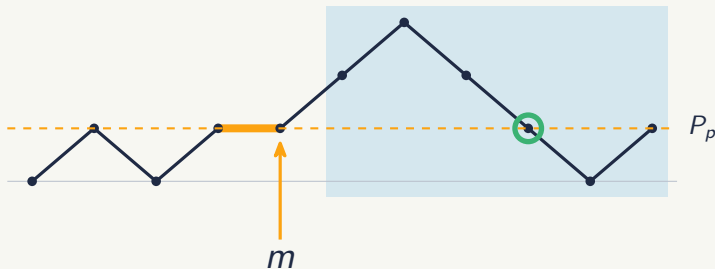
notation

- A_i ± 1 sign
- P_j prefix sum
- p position of m
- ℓ, r window ends
- n array length

Housing – m at an end: prefix sums

$$P_j = A_1 + \cdots + A_j, \quad P_0 = 0$$

$$j \in [p+1, n]$$



$$[p, r] \text{ works} \iff P_r = P_p$$

notation

A_i ± 1 sign

P_j prefix sum

p position of m

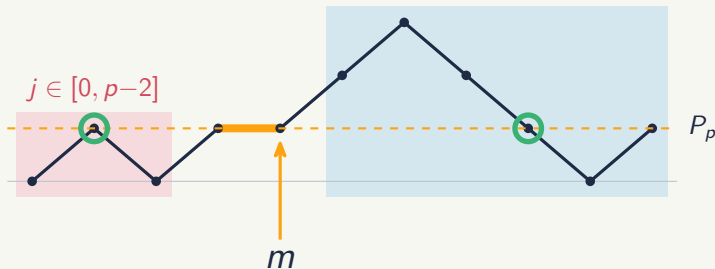
ℓ, r window ends

n array length

Housing – m at an end: prefix sums

$$P_j = A_1 + \cdots + A_j, \quad P_0 = 0$$

$$j \in [p+1, n]$$



$$[p, r] \text{ works} \iff P_r = P_p$$

$$[\ell, p] \text{ works} \iff P_{\ell-1} = P_p$$

notation

A_i ± 1 sign

P_j prefix sum

p position of m

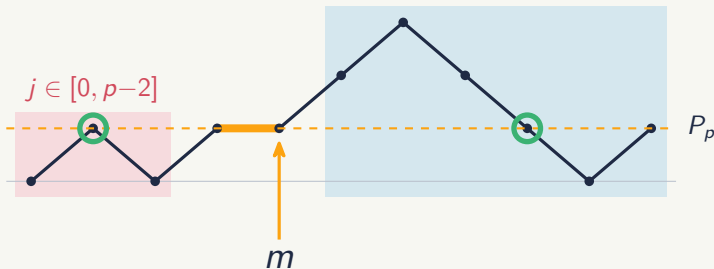
ℓ, r window ends

n array length

Housing – m at an end: prefix sums

$$P_j = A_1 + \cdots + A_j, \quad P_0 = 0$$

$$j \in [p+1, n]$$



notation

A_i ± 1 sign

P_j prefix sum

p position of m

ℓ, r window ends

n array length

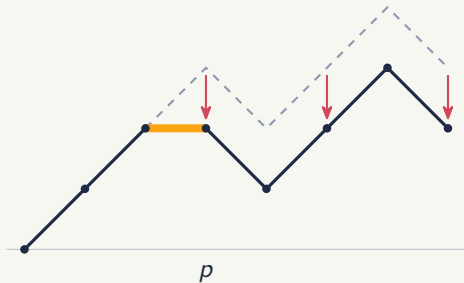
$$[p, r] \text{ works} \iff P_r = P_p$$

$$[\ell, p] \text{ works} \iff P_{\ell-1} = P_p$$

steps are ± 1 : **hit** $\iff \min P \leq P_p \leq \max P$ on the range

Housing – sweep $m = 1, 2, \dots, n$

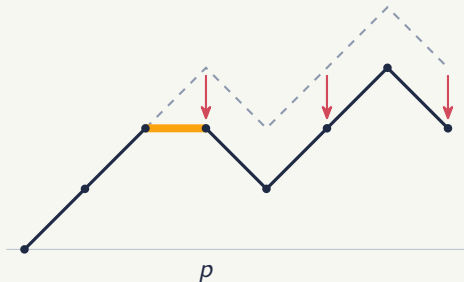
$$A_p: \quad +1 \xrightarrow{\text{test } m} 0 \xrightarrow{\text{done}} -1$$



$$P_j \leftarrow P_j - 1 \quad \text{for all } j \geq p$$

Housing – sweep $m = 1, 2, \dots, n$

$$A_p: +1 \xrightarrow{\text{test } m} 0 \xrightarrow{\text{done}} -1$$



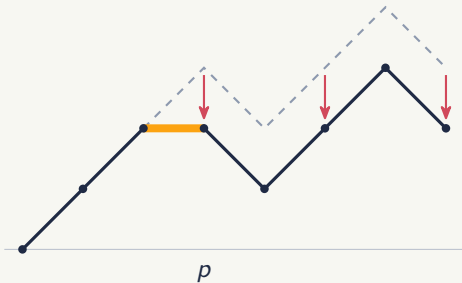
$$P_j \leftarrow P_j - 1 \text{ for all } j \geq p$$

Needed operations

- ▶ flip +1 to -1 in A
- ▶ min and max of P on a range
- ▶ read a single P_p

Housing – sweep $m = 1, 2, \dots, n$

$A_p: +1 \xrightarrow{\text{test } m} 0 \xrightarrow{\text{done}} -1$



$P_j \leftarrow P_j - 1$ for all $j \geq p$

Needed operations

- ▶ flip $+1$ to -1 in A
- ▶ min and max of P on a range
- ▶ read a single P_p

Data Structure over P

- ▶ Point update on $A \Leftrightarrow$ range update on P
- ▶ Lazy segment tree with range updates and min/max range queries
- ▶ Total running time: $\mathcal{O}(n \log n)$.

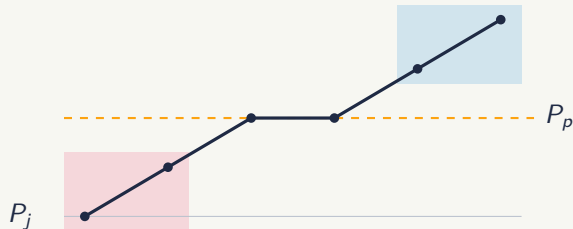
Housing – example

a_i	2	5	1	4	3
-------	---	---	---	---	---

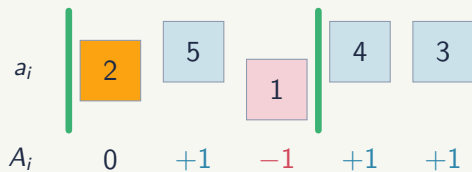
A_i	+1	+1	0	+1	+1
-------	----	----	---	----	----

$$m = 1$$

NO

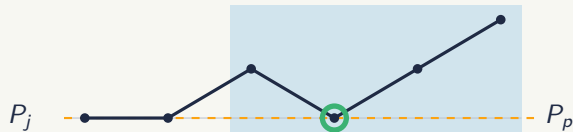


Housing – example

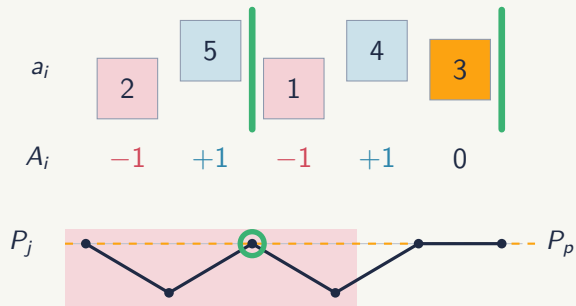


$$m = 2$$

YES



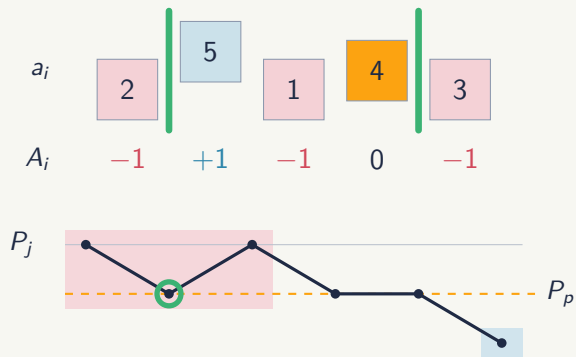
Housing – example



$$m = 3$$

YES

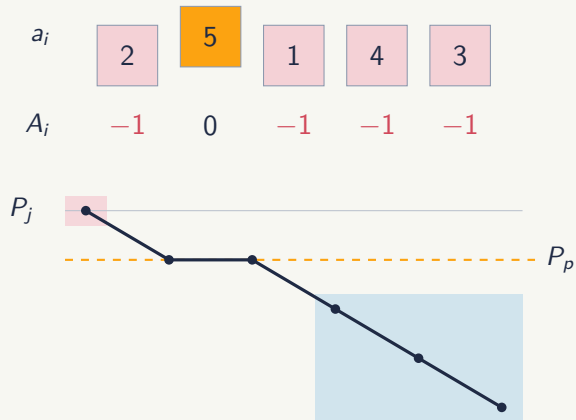
Housing – example



$$m = 4$$

YES

Housing – example

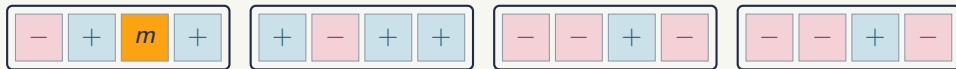


$$m = 5$$

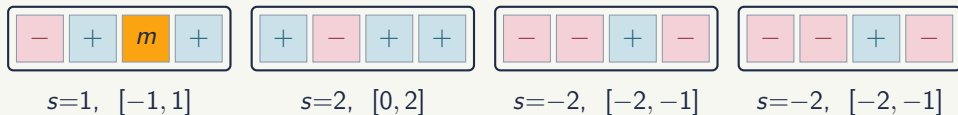
NO

YES NO NO YES YES

Housing – alternative solution: square-root decomposition



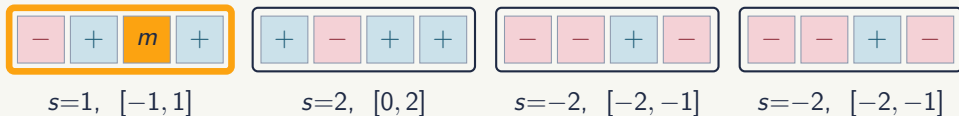
Housing – alternative solution: square-root decomposition



block = sum s + range $[lo, hi]$ of its prefix sums

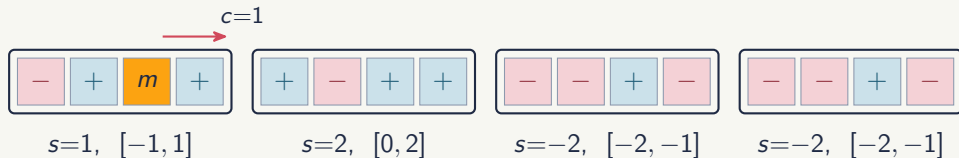
Housing – alternative solution: square-root decomposition

rebuild: $\mathcal{O}(\sqrt{n})$



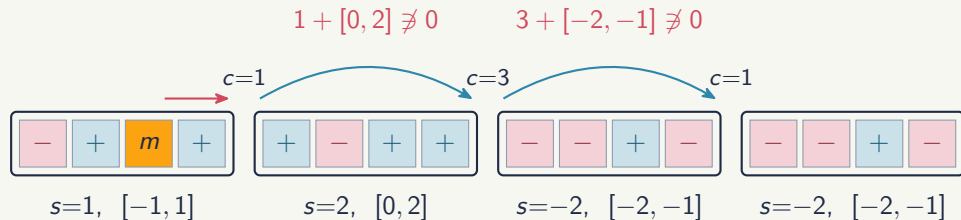
block = sum s + range $[lo, hi]$ of its prefix sums

Housing – alternative solution: square-root decomposition



block = sum s + range $[lo, hi]$ of its prefix sums

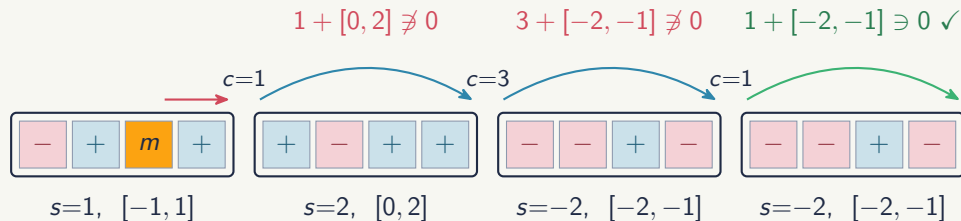
Housing – alternative solution: square-root decomposition



block = sum s + range $[lo, hi]$ of its prefix sums

hit inside the block $\iff 0 \in c + [lo, hi]$, else $c \leftarrow c + s$

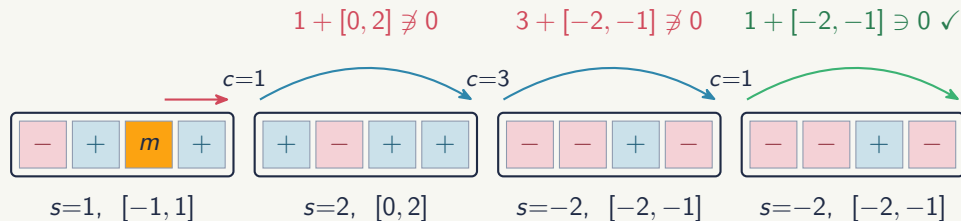
Housing – alternative solution: square-root decomposition



block = sum s + range $[lo, hi]$ of its prefix sums

hit inside the block $\iff 0 \in c + [lo, hi]$, else $c \leftarrow c + s$

Housing – alternative solution: square-root decomposition



block = sum s + range $[lo, hi]$ of its prefix sums

hit inside the block $\iff 0 \in c + [lo, hi]$, else $c \leftarrow c + s$

$\mathcal{O}(\sqrt{n})$ per step $\Rightarrow \mathcal{O}(n\sqrt{n})$ (left side: mirror)

Park

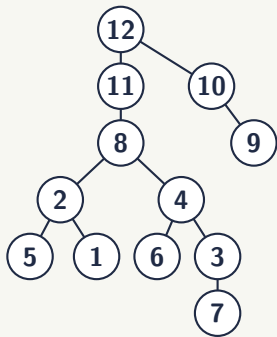
Squirrels run along every path of Míša's tree. How popular is the whole tree?



DSU · small-to-large · HLD · Fenwick

Task

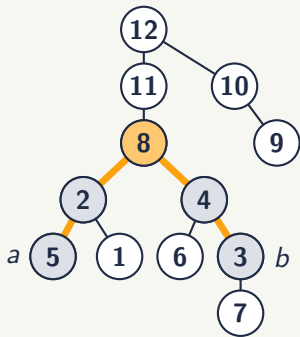
Tree on $N \leq 10^5$ vertices, beauties b form a permutation of $1..N$. Compute $\sum_{\{u,v\}} \text{dist}(u,v) \cdot \max_{x \in \text{path}(u,v)} b_x \pmod{10^9 + 7}$.



Numbers inside the vertices are the beauties.

Task

Tree on $N \leq 10^5$ vertices, beauties b form a permutation of $1..N$. Compute $\sum_{\{u,v\}} \text{dist}(u,v) \cdot \max_{x \in \text{path}(u,v)} b_x \pmod{10^9 + 7}$.

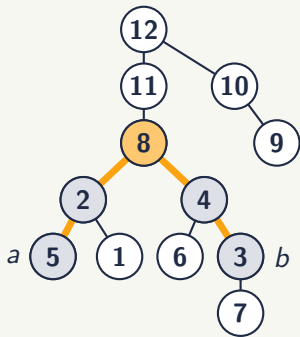


One pair $\{a, b\}$: length 4 edges, largest beauty 8:

$$4 \cdot \max(5, 2, 8, 4, 3) = 4 \cdot 8 = 32$$

Task

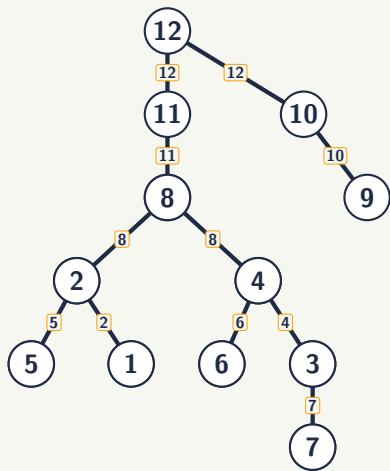
Tree on $N \leq 10^5$ vertices, beauties b form a permutation of $1..N$. Compute $\sum_{\{u,v\}} \text{dist}(u,v) \cdot \max_{x \in \text{path}(u,v)} b_x \pmod{10^9 + 7}$.



$\Theta(N^2)$ pairs: too many.

Group the pairs by their maximum.

Park – grow components, cheapest first



w :

2

4

5

6

7

8

8

10

11

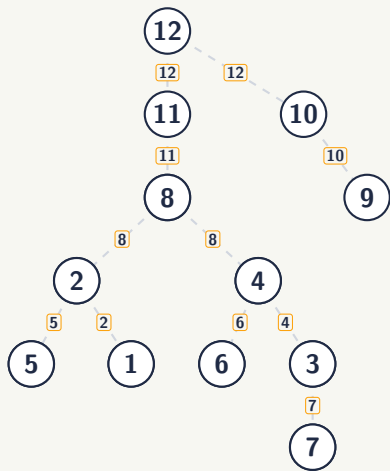
12

12

Edge weight $w(uv) = \max(b_u, b_v)$.

Max over the path's vertices = max over its edges.

Park – grow components, cheapest first



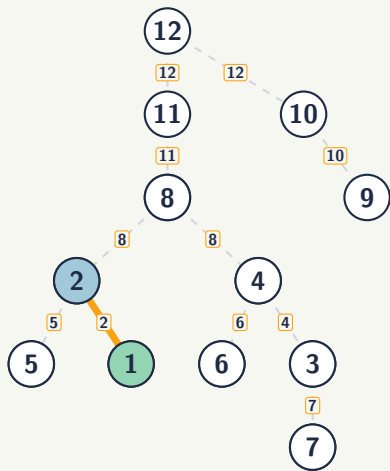
w :

2	4	5	6	7	8	8	10	11	12	12
---	---	---	---	---	---	---	----	----	----	----

Remove all edges.

Put them back in *increasing* order of w ,
joining components (DSU).

Park – grow components, cheapest first



w :

2

4

5

6

7

8

8

10

11

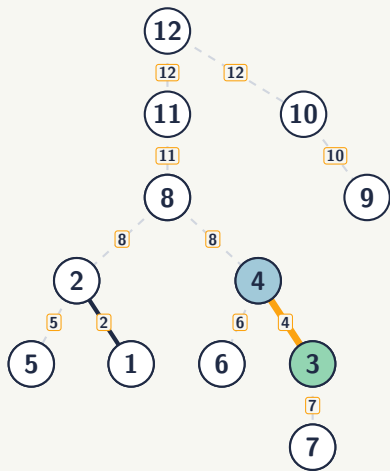
12

12

The orange edge joins A and B :

all $|A| \cdot |B|$ new pairs have maximum w .

Park – grow components, cheapest first



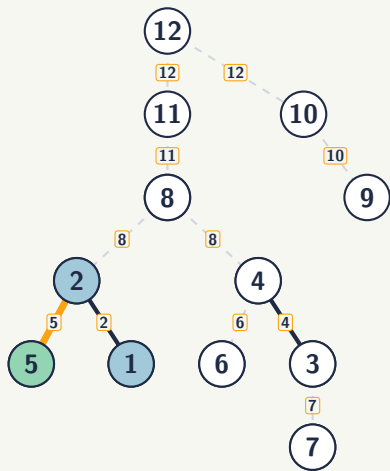
w :

2	4	5	6	7	8	8	10	11	12	12
---	---	---	---	---	---	---	----	----	----	----

The orange edge joins A and B :

all $|A| \cdot |B|$ new pairs have maximum w .

Park – grow components, cheapest first



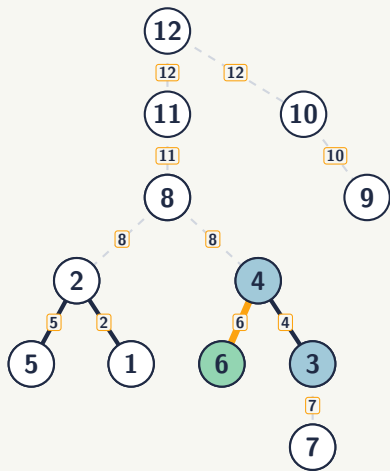
w :

2	4	5	6	7	8	8	10	11	12	12
---	---	---	---	---	---	---	----	----	----	----

The orange edge joins A and B :

all $|A| \cdot |B|$ new pairs have maximum w .

Park – grow components, cheapest first



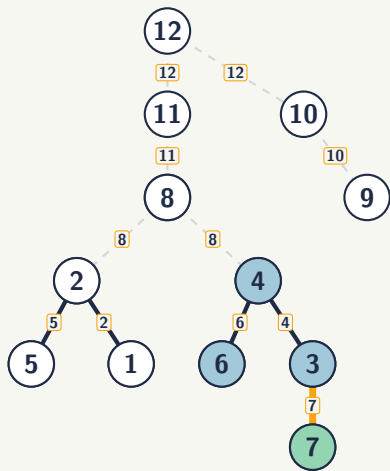
w :

2	4	5	6	7	8	8	10	11	12	12
---	---	---	---	---	---	---	----	----	----	----

The orange edge joins A and B :

all $|A| \cdot |B|$ new pairs have maximum w .

Park – grow components, cheapest first



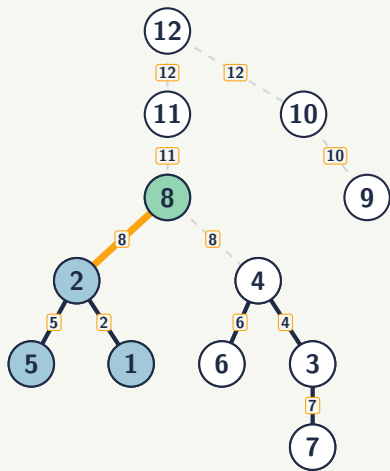
w :

2	4	5	6	7	8	8	10	11	12	12
---	---	---	---	---	---	---	----	----	----	----

The orange edge joins A and B :

all $|A| \cdot |B|$ new pairs have maximum w .

Park – grow components, cheapest first



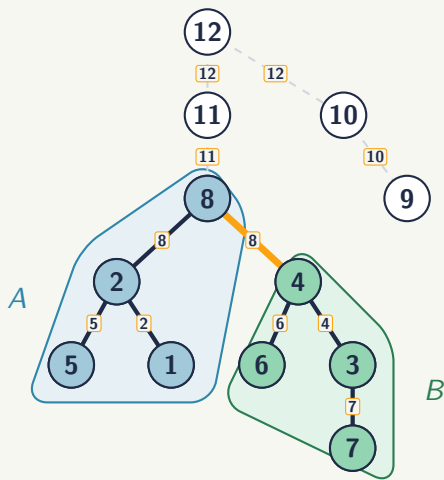
w :

2	4	5	6	7	8	8	10	11	12	12
---	---	---	---	---	---	---	----	----	----	----

The orange edge joins A and B :

all $|A| \cdot |B|$ new pairs have maximum w .

Park – grow components, cheapest first



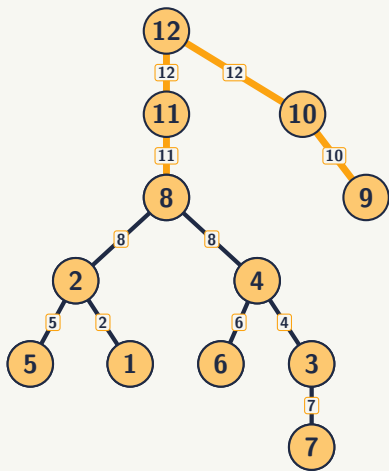
w :

2	4	5	6	7	8	8	10	11	12	12
---	---	---	---	---	---	---	----	----	----	----

$w = 8$ joins A and B :

the merge we look at next.

Park – grow components, cheapest first



w :

2

4

5

6

7

8

8

10

11

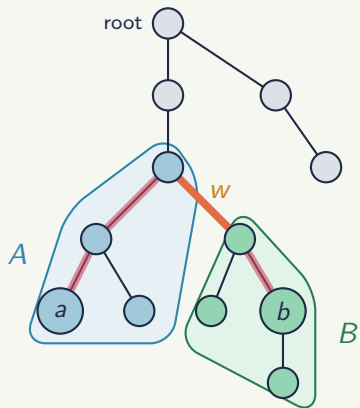
12

12

Every pair is counted exactly once.

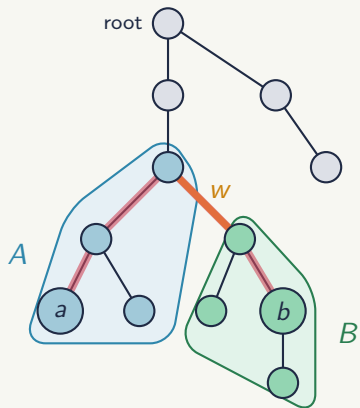
Ties ($w = 8$, $w = 12$) can go in any order.

Park – one merge



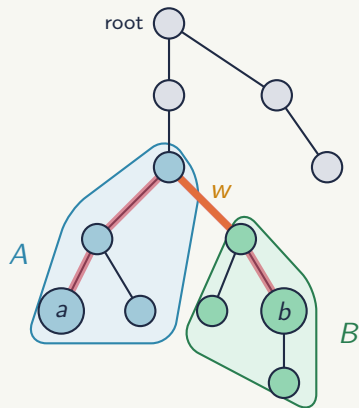
The edge of weight w joins components A and B .

Park – one merge



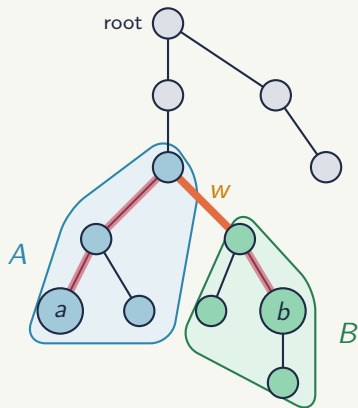
Every pair $a \in A$, $b \in B$ gets connected *now*.

Park – one merge



Edges inside A , B came earlier
 $\Rightarrow w$ is the maximum on every a - b path.

Park – one merge



Edges inside A , B came earlier
 $\Rightarrow w$ is the maximum on every a - b path.

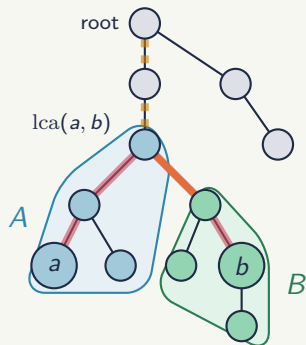
Key idea

$$\text{res} += w \cdot \sum_{a \in A, b \in B} \text{dist}(a, b)$$

Park – sum of distances

Root the tree; $\text{dep}(v)$ = vertices on root $\rightarrow v$.

$$\text{dist}(a, b) = \text{dep}(a) + \text{dep}(b) - 2 \text{dep}(\text{lca}(a, b))$$

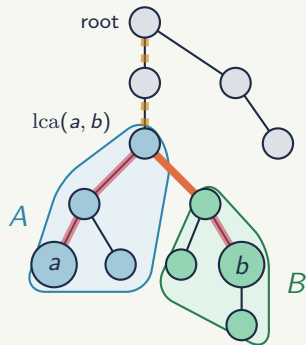


Park – sum of distances

Root the tree; $\text{dep}(v)$ = vertices on root – v .

$$\text{dist}(a, b) = \text{dep}(a) + \text{dep}(b) - 2 \text{dep}(\text{lca}(a, b))$$

$$\sum_{a,b} \text{dist}(a, b) = \sum_{a,b} \text{dep}(a) + \sum_{a,b} \text{dep}(b) - 2 \sum_{a,b} \text{dep}(\text{lca})$$



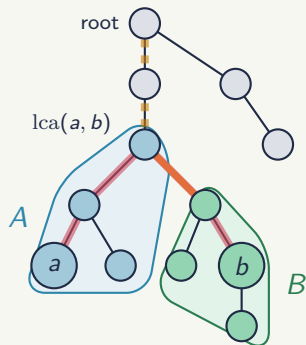
Park – sum of distances

Root the tree; $\text{dep}(v)$ = vertices on root – v .

$$\text{dist}(a, b) = \text{dep}(a) + \text{dep}(b) - 2 \text{dep}(\text{lca}(a, b))$$

$$\begin{aligned} \sum_{a,b} \text{dist}(a, b) &= \sum_{a,b} \text{dep}(a) + \sum_{a,b} \text{dep}(b) - 2 \sum_{a,b} \text{dep}(\text{lca}) \\ &= |B| \text{SD}(A) + |A| \text{SD}(B) - 2 \sum_{a,b} \text{dep}(\text{lca}(a, b)) \end{aligned}$$

where $\text{SD}(X) = \sum_{x \in X} \text{dep}(x)$.



Park – sum of distances

Root the tree; $\text{dep}(v)$ = vertices on root – v .

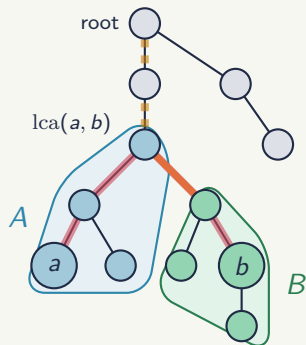
$$\text{dist}(a, b) = \text{dep}(a) + \text{dep}(b) - 2 \text{dep}(\text{lca}(a, b))$$

$$\begin{aligned} \sum_{a,b} \text{dist}(a, b) &= \sum_{a,b} \text{dep}(a) + \sum_{a,b} \text{dep}(b) - 2 \sum_{a,b} \text{dep}(\text{lca}) \\ &= |B| \text{SD}(A) + |A| \text{SD}(B) - 2 \sum_{a,b} \text{dep}(\text{lca}(a, b)) \end{aligned}$$

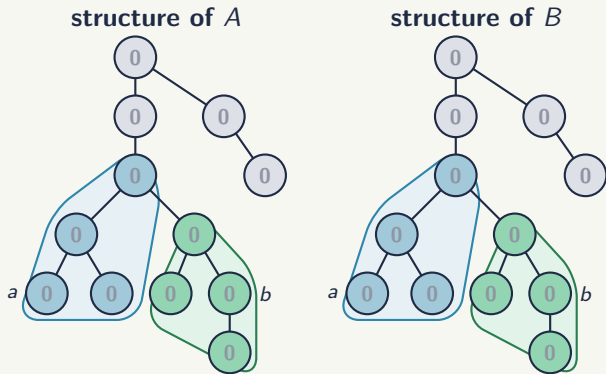
where $\text{SD}(X) = \sum_{x \in X} \text{dep}(x)$.

Every component X keeps

size $|X|$, $\text{SD}(X)$, and a data structure for the last sum.



Park – summing LCA depths

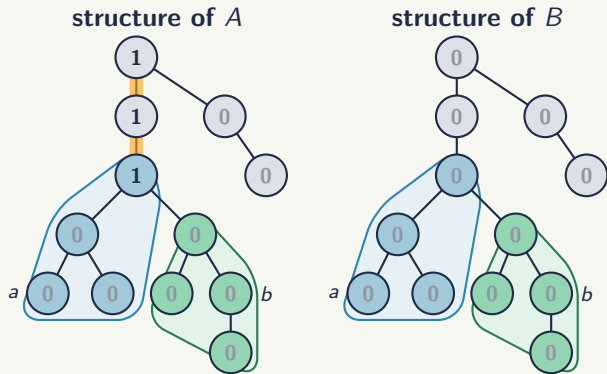


update(v): +1 on every u on root- v

query(v): sum over u on root- v

Every component has its own structure.

Park – summing LCA depths

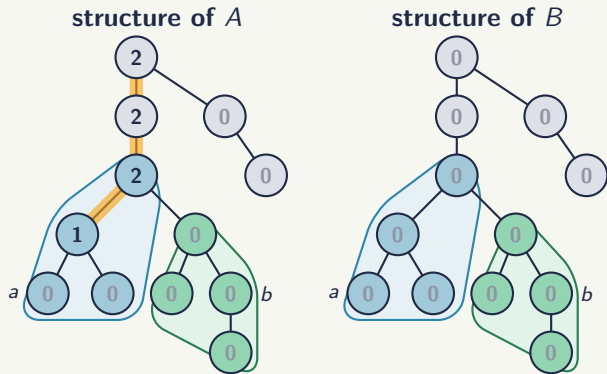


$\text{update}(v)$: +1 on every u on root- v

$\text{query}(v)$: sum over u on root- v

$\text{update}_A(v)$ for every $v \in A$, one at a time: +1 along its root path (orange).

Park – summing LCA depths

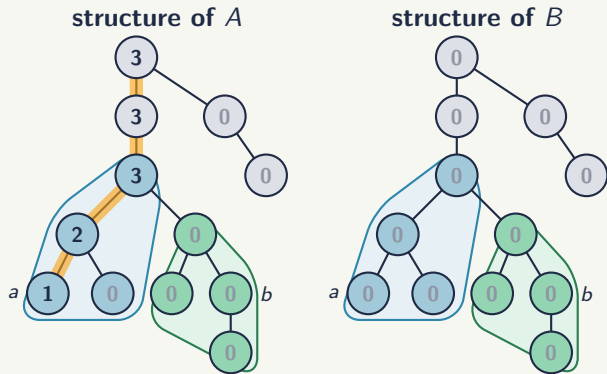


$\text{update}(v)$: +1 on every u on root- v

$\text{query}(v)$: sum over u on root- v

$\text{update}_A(v)$ for every $v \in A$, one at a time: +1 along its root path (orange).

Park – summing LCA depths

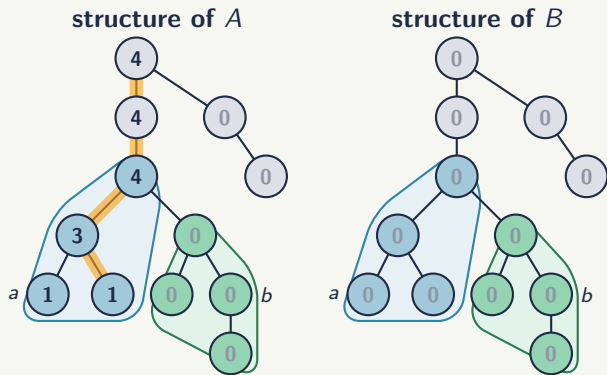


update(v): +1 on every u on root- v

query(v): sum over u on root $\rightarrow v$

update_A(v) for every $v \in A$, one at a time: +1 along its root path (orange).

Park – summing LCA depths



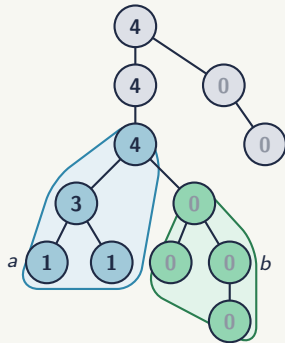
$\text{update}(v)$: +1 on every u on root- v

$\text{query}(v)$: sum over u on root- v

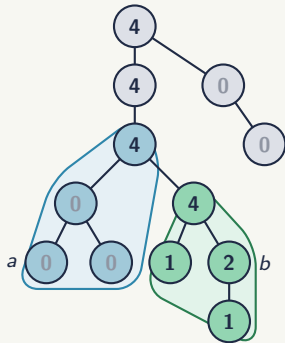
$\text{update}_A(v)$ for every $v \in A$, one at a time: +1 along its root path (orange).

Park – summing LCA depths

structure of A



structure of B

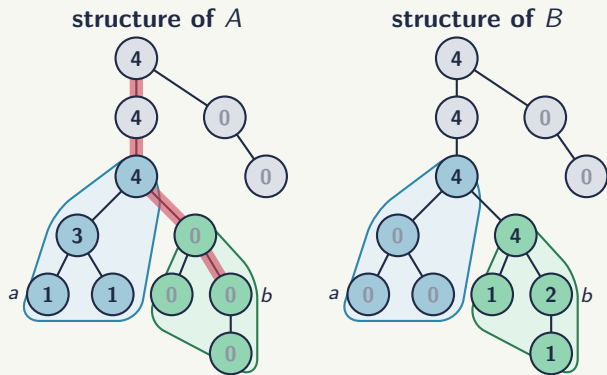


$\text{update}(v)$: +1 on every u on root- v

$\text{query}(v)$: sum over u on root- v

Same for B , with its own vertices.

Park – summing LCA depths



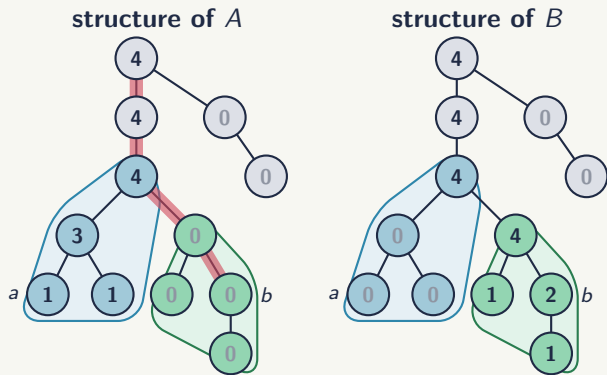
update(v): +1 on every u on root- v

query(v): sum over u on root- v

$$\text{query}_A(b) = \\ 0 + 0 + 4 + 4 + 4 = 12.$$

It collects +1 from every a for each *common* ancestor of a and b , i.e. $\text{dep}(\text{lca}(a, b))$ times.

Park – summing LCA depths



update(v): +1 on every u on root- v

query(v): sum over u on root- v

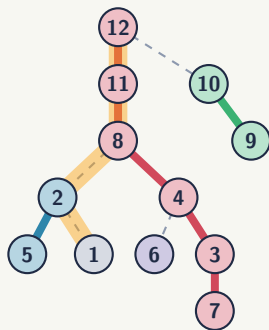
$$\sum_{a,b} \text{dep}(\text{lca}(a, b)) = \sum_{b \in B} \text{query}_A(b)$$

Every $b \in B$ gives 12: 48.

With $\text{SD}(A) = 17$, $\text{SD}(B) = 20$:

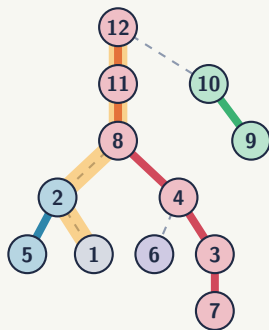
$$4 \cdot 17 + 4 \cdot 20 - 2 \cdot 48 = \mathbf{52} \quad \checkmark$$

Park – root paths fast: HLD + Fenwick



Heavy-light decomposition

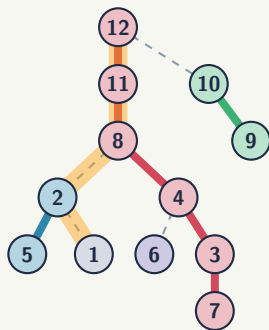
Park – root paths fast: HLD + Fenwick



Heavy-light decomposition

12	11	8	4	3	7	6	2	5	1	10	9
----	----	---	---	---	---	---	---	---	---	----	---

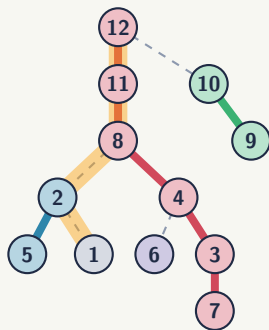
Park – root paths fast: HLD + Fenwick



Heavy-light decomposition

12	11	8	4	3	7	6	2	5	1	10	9
----	----	---	---	---	---	---	---	---	---	----	---

Park – root paths fast: HLD + Fenwick



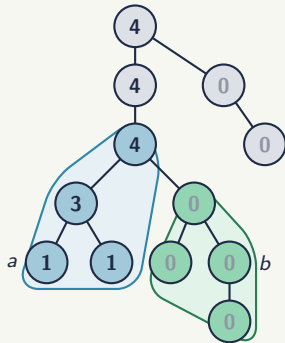
Heavy-light decomposition

12	11	8	4	3	7	6	2	5	1	10	9
----	----	---	---	---	---	---	---	---	---	----	---

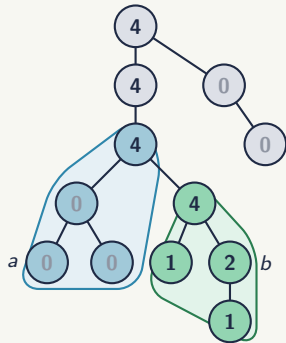
$\Rightarrow$ update, query in $\mathcal{O}(\log^2 N)$

Park – small to large

structure of A

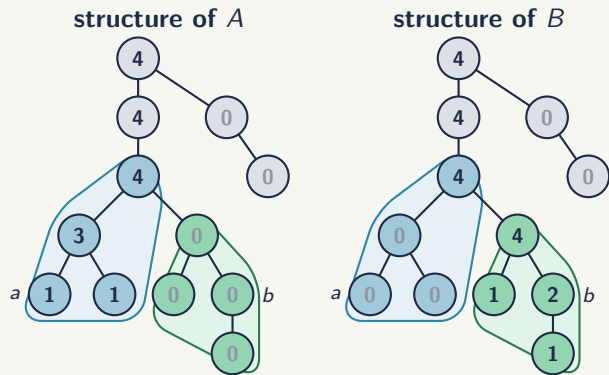


structure of B



One structure per component,
stored sparsely (hash map of
touched cells).

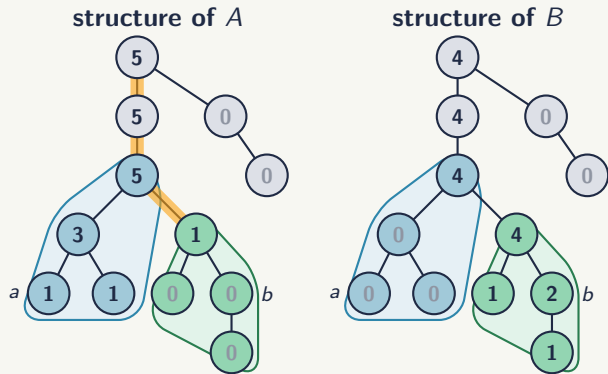
Park – small to large



One structure per component, stored sparsely (hash map of touched cells).

Merge: first query $_A(b)$ for every b in the *smaller* side.

Park – small to large

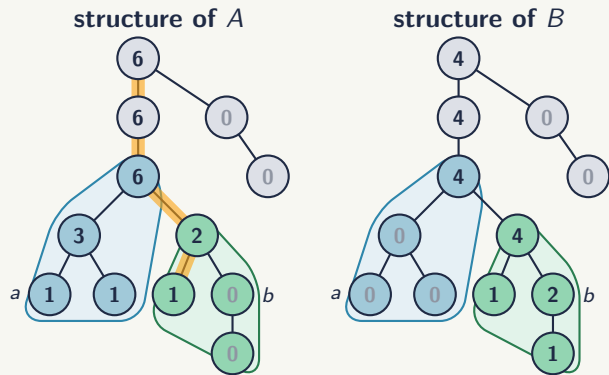


One structure per component, stored sparsely (hash map of touched cells).

Merge: first $\text{query}_A(b)$ for every b in the *smaller* side.

Then $\text{update}_A(b)$, one b at a time.

Park – small to large

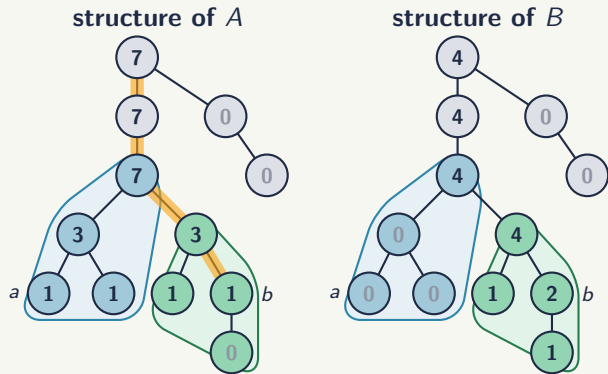


One structure per component, stored sparsely (hash map of touched cells).

Merge: first $\text{query}_A(b)$ for every b in the *smaller* side.

Then $\text{update}_A(b)$, one b at a time.

Park – small to large

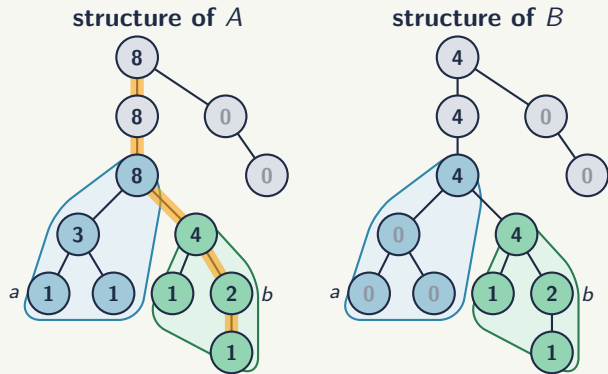


One structure per component, stored sparsely (hash map of touched cells).

Merge: first $\text{query}_A(b)$ for every b in the *smaller* side.

Then $\text{update}_A(b)$, one b at a time.

Park – small to large



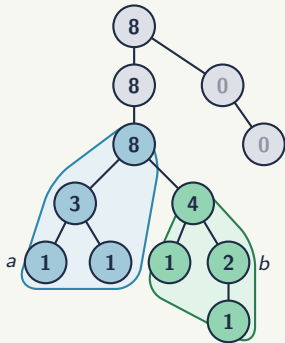
One structure per component, stored sparsely (hash map of touched cells).

Merge: first $\text{query}_A(b)$ for every b in the *smaller* side.

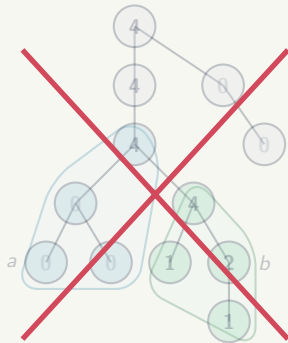
Then $\text{update}_A(b)$, one b at a time.

Park – small to large

structure of $A \cup B$



structure of B



One structure per component, stored sparsely (hash map of touched cells).

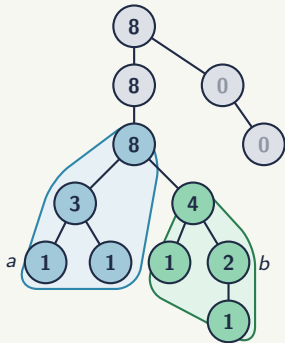
Merge: first $\text{query}_A(b)$ for every b in the *smaller* side.

Then $\text{update}_A(b)$, one b at a time.

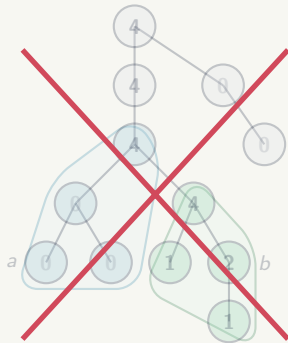
B 's structure is thrown away; A 's now serves $A \cup B$. Add up sizes and SD.

Park – small to large

structure of $A \cup B$



structure of B



One structure per component, stored sparsely (hash map of touched cells).

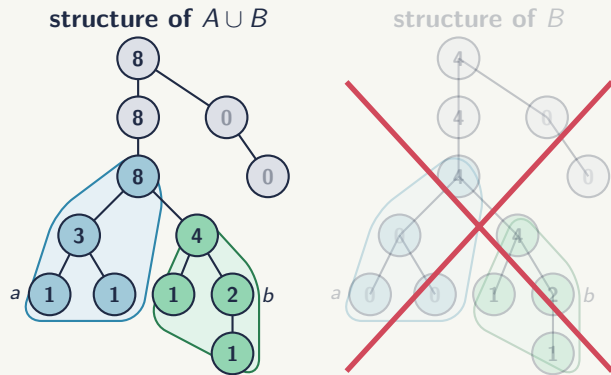
Merge: first $\text{query}_A(b)$ for every b in the *smaller* side.

Then $\text{update}_A(b)$, one b at a time.

B 's structure is thrown away; A 's now serves $A \cup B$. Add up sizes and SD.

A vertex moves only into a component at least twice as big $\Rightarrow \mathcal{O}(\log N)$ moves.

Park – small to large



One structure per component, stored sparsely (hash map of touched cells).

Merge: first $\text{query}_A(b)$ for every b in the *smaller* side.

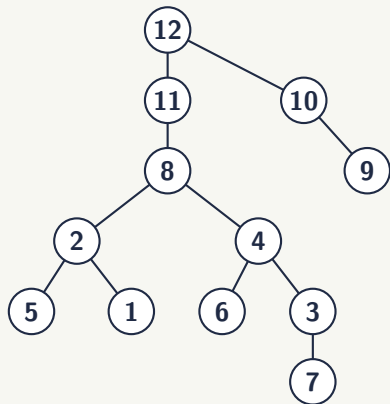
Then $\text{update}_A(b)$, one b at a time.

B 's structure is thrown away; A 's now serves $A \cup B$. Add up sizes and SD.

A vertex moves only into a component at least twice as big $\Rightarrow \mathcal{O}(\log N)$ moves.

Complexity: $\mathcal{O}(N \log N)$ moves $\times \mathcal{O}(\log^2 N) = \mathcal{O}(N \log^3 N)$

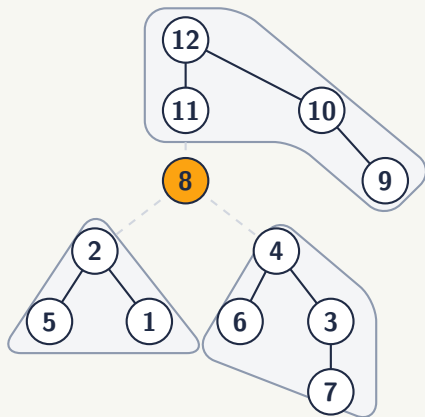
Park – alternative: centroid decomposition



Centroid: a vertex whose removal leaves pieces of size $\leq N/2$.

Always exists. Find it: start anywhere, step into a subtree bigger than $N/2$ while there is one.

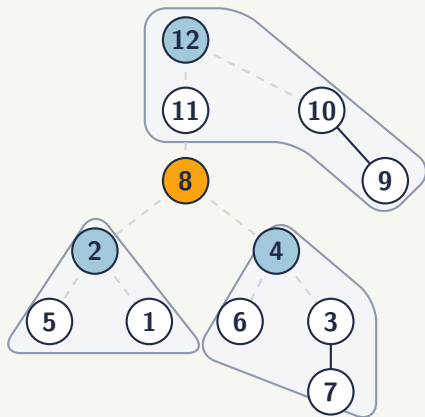
Park – alternative: centroid decomposition



8 is the centroid: pieces of sizes 4, 3, 4.

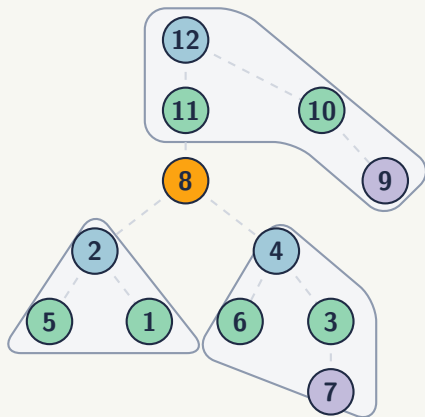
Every path either *goes through* 8, or stays inside one piece.

Park – alternative: centroid decomposition



Count the paths through 8 (next slide),
delete 8, recurse into every piece.

Park – alternative: centroid decomposition

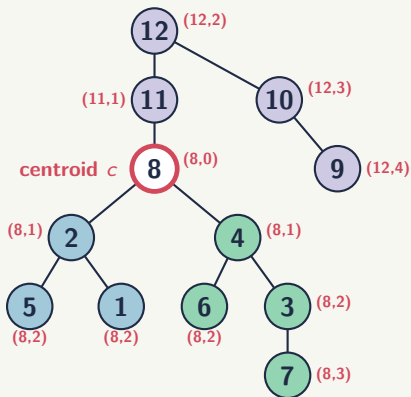


Pieces at least halve $\Rightarrow$ at most $\log_2 N + 1$ levels.

Every vertex is in one piece per level $\Rightarrow$ $\mathcal{O}(N)$ vertices per level.

● level 0 ● level 1 ● level 2 ● level 3

Park – paths through the centroid

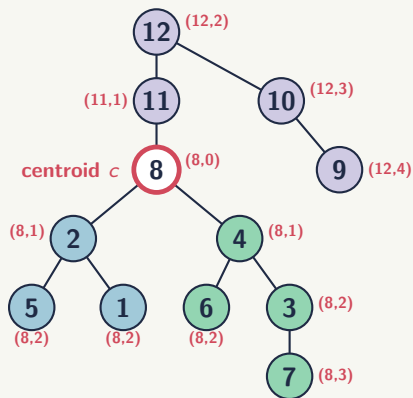


For every x , on the path $c \rightarrow x$: $d_x = \text{edges}$, $m_x = \text{largest beauty (with } c)$. Red: (m_x, d_x) .

Path $x-y$ through c : length $d_x + d_y$, $\max = \max(m_x, m_y)$.

c itself is $(b_c, 0)$: pairs (c, x) come out right with no special case.

Park – paths through the centroid



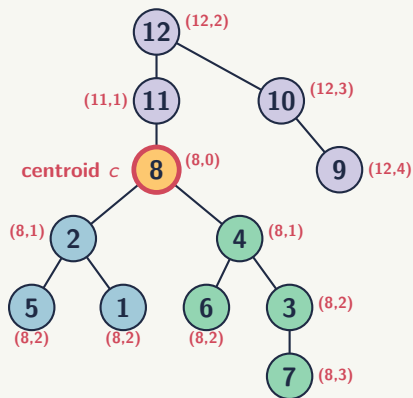
m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (cnt \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

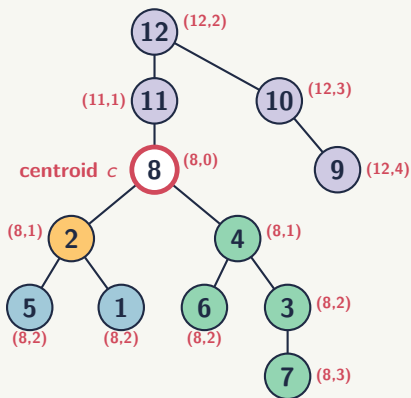
$\text{cnt} = 0, \Sigma D = 0: + 8 \cdot (0 \cdot 0 + 0) = 0 \quad \text{sum} = 0$

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

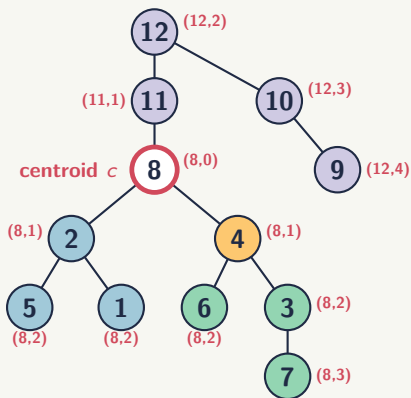
$\text{cnt} = 1, \Sigma D = 0: + 8 \cdot (1 \cdot 1 + 0) = 8 \quad \text{sum} = 8$

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

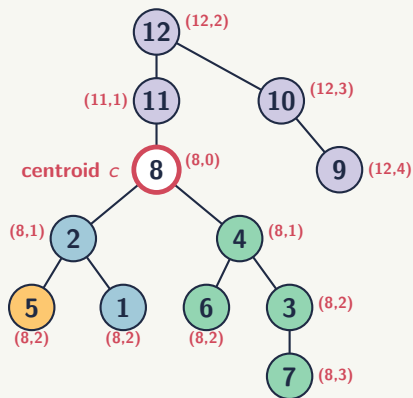
$\text{cnt} = 2, \Sigma D = 1: + 8 \cdot (2 \cdot 1 + 1) = 24 \quad \text{sum} = 32$

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

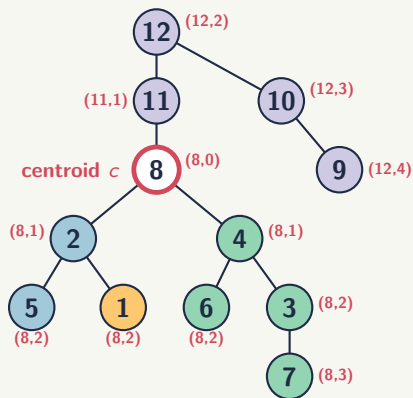
$\text{cnt} = 3, \Sigma D = 2: + 8 \cdot (3 \cdot 2 + 2) = 64 \quad \text{sum} = \mathbf{96}$

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

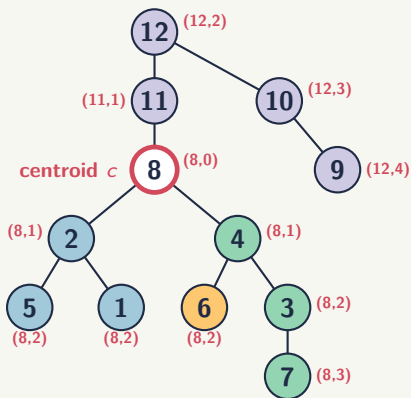
$\text{cnt} = 4$, $\Sigma D = 4$: $+ 8 \cdot (4 \cdot 2 + 4) = 96$ $\text{sum} = 192$

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

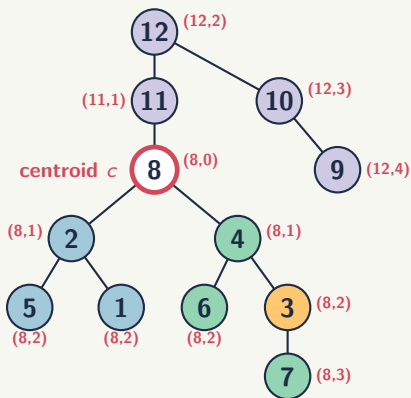
$\text{cnt} = 5$, $\Sigma D = 6$: $+ 8 \cdot (5 \cdot 2 + 6) = 128$ $\text{sum} = 320$

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

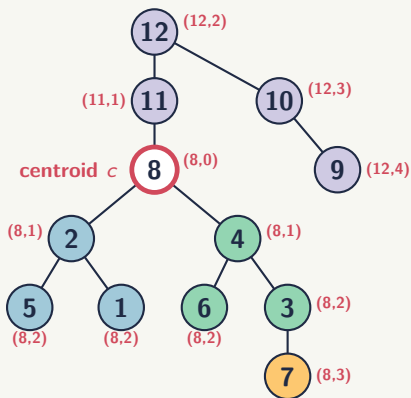
$\text{cnt} = 6, \Sigma D = 8: + 8 \cdot (6 \cdot 2 + 8) = 160 \quad \text{sum} = \mathbf{480}$

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

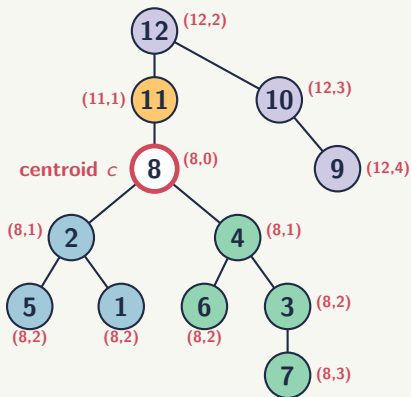
$\text{cnt} = 7$, $\Sigma D = 10 + 8 \cdot (7 \cdot 3 + 10) = 248$ sum = **728**

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

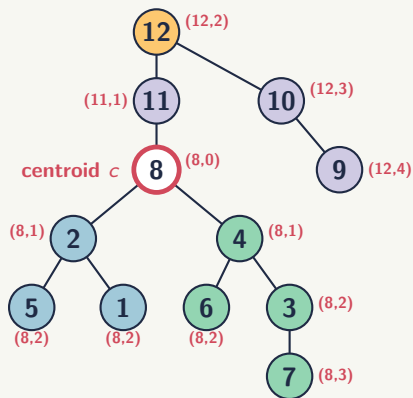
$\text{cnt} = 8$, $\Sigma D = 13$; $+ 11 \cdot (8 \cdot 1 + 13) = 231$ sum = **959**

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

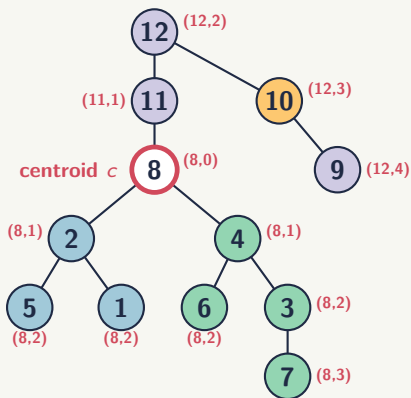
$\text{cnt} = 9$, $\Sigma D = 14$; $+ 12 \cdot (9 \cdot 2 + 14) = 384$ sum = **1343**

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

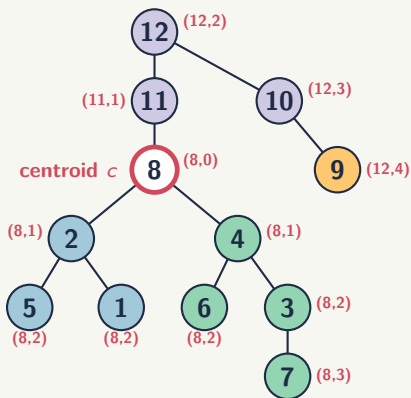
$\text{cnt} = 10$, $\Sigma D = 16 + 12 \cdot (10 \cdot 3 + 16) = 552$, $\text{sum} = \mathbf{1895}$

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

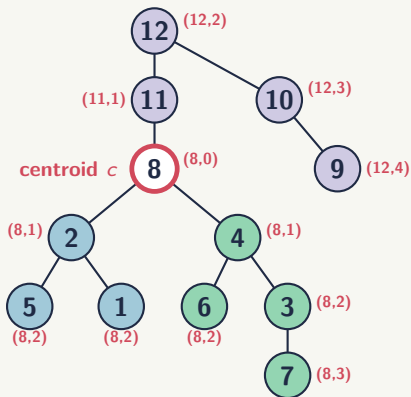
$\text{cnt} = 11$, $\Sigma D = 19 + 12 \cdot (11 \cdot 4 + 19) = 756$, $\text{sum} = \mathbf{2651}$

Sort by m . Go left to right, keeping cnt and ΣD of the entries already seen.

x is the max of every pair with an earlier entry:

$$\text{add } m_x \cdot (\text{cnt} \cdot d_x + \Sigma D)$$

Park – paths through the centroid



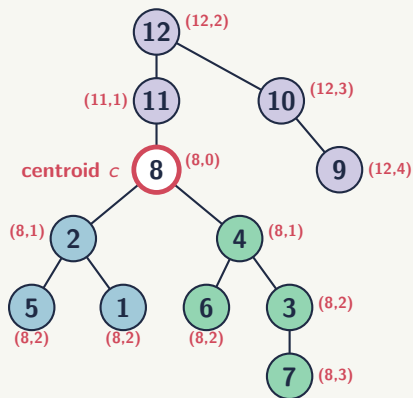
m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

sum = 2651

But this also counted pairs *inside one branch*, e.g. 12 and 10:

their path does not go through c .

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

11	12	12	12
1	2	3	4

→ **360**

8	8	8
1	2	2

→ **80**

8	8	8	8
1	2	2	3

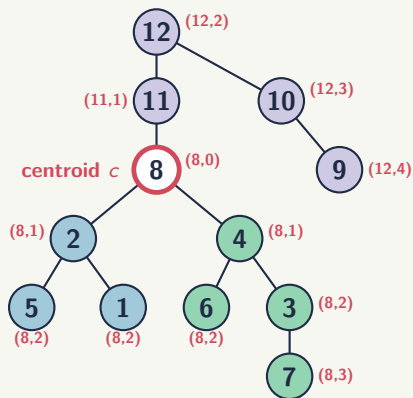
→ **192**

Run the same thing on each branch's own list and subtract.

A same-branch pair gets the same value in both runs, so it cancels exactly:

$$2651 - 360 - 80 - 192 = \mathbf{2019} \checkmark$$

Park – paths through the centroid



m	8	8	8	8	8	8	8	8	11	12	12	12
d	0	1	1	2	2	2	2	3	1	2	3	4

11	12	12	12
1	2	3	4

 → 360

8	8	8
1	2	2

 → 80

8	8	8	8
1	2	2	3

 → 192

Then delete c and recurse into the pieces.

Per level: one sort of $\mathcal{O}(N)$ entries.

Complexity: $\mathcal{O}(N \log N)$ per level $\times \log N$ levels $= \mathcal{O}(N \log^2 N)$

Cables

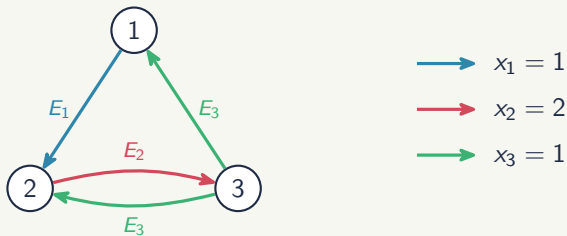
Find the shortest run of consecutive edge groups that can carry a balanced flow.

● ● ● ● ● linear algebra · Gaussian elimination · arithmetic mod p

Cables

Task

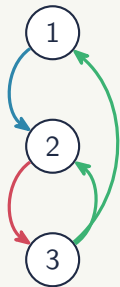
Edge groups $E_1, \dots, E_k$ on n nodes. Give every group a weight x_q (negative = reversed), not all zero, so that every node has in = out. Find the shortest block of *consecutive* groups where this works, or -1 .



$$n \leq 100, \quad k \leq 2 \cdot 10^5$$

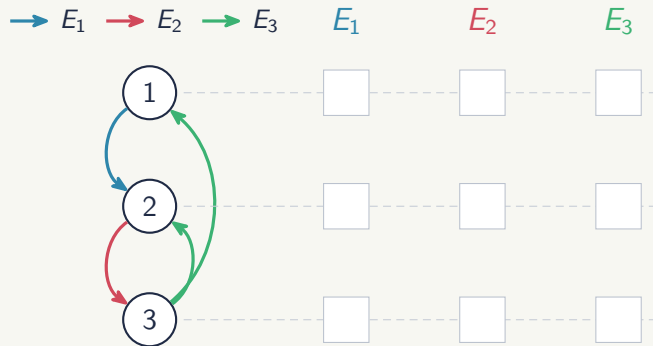
Cables – edge groups are vectors

→ E_1 → E_2 → E_3



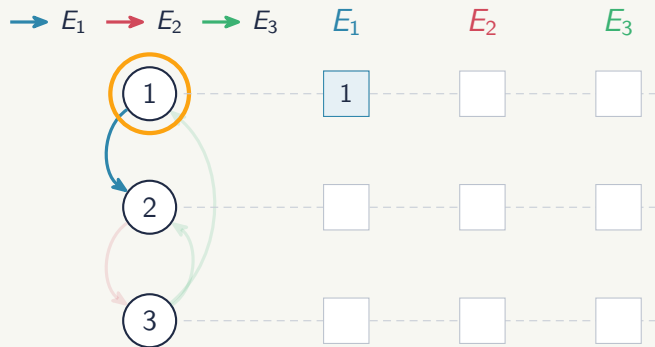
Sample 2, the nodes in one column.

Cables – edge groups are vectors



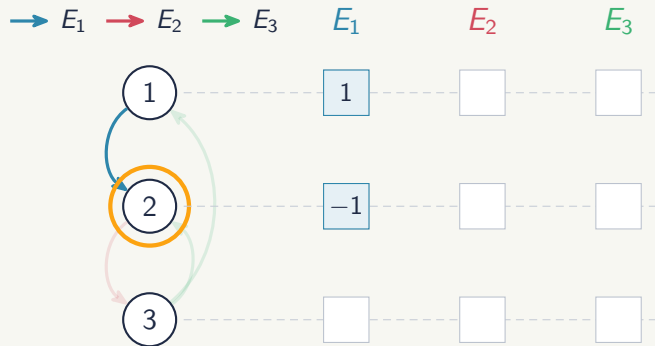
One vector per group, **one entry per node.**

Cables – edge groups are vectors



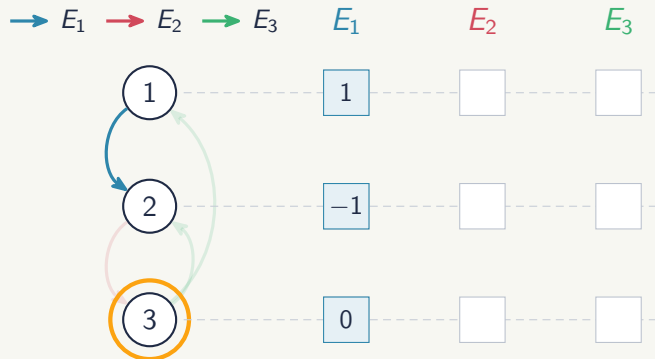
E_1 , node 1: one edge **out** $\Rightarrow +1$.

Cables – edge groups are vectors



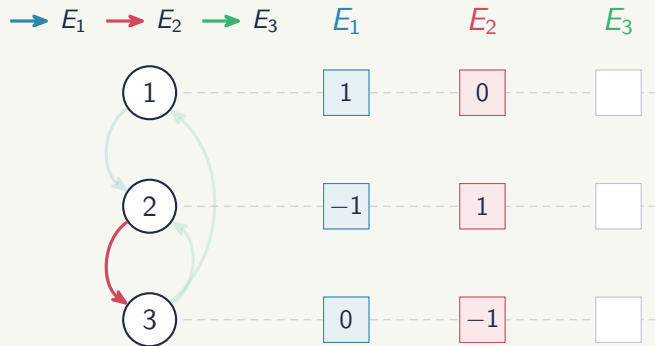
Node 2: one edge **in** $\Rightarrow -1$.

Cables – edge groups are vectors



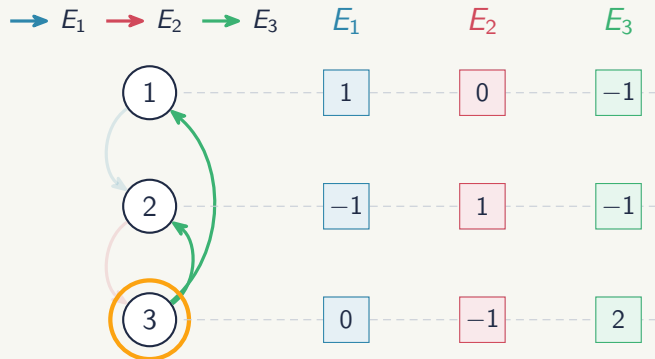
Node 3: no edge of $E_1 \Rightarrow 0$.

Cables – edge groups are vectors



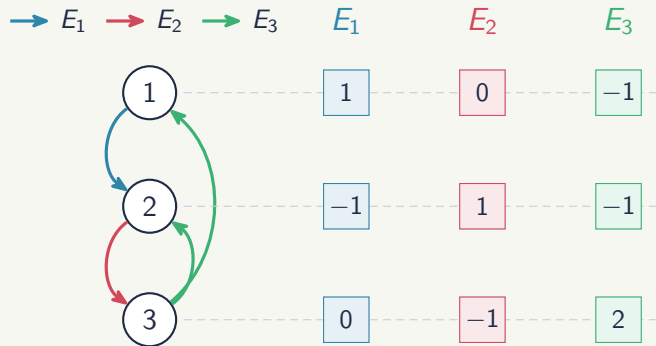
E_2 : the edge $2 \rightarrow 3$.

Cables – edge groups are vectors



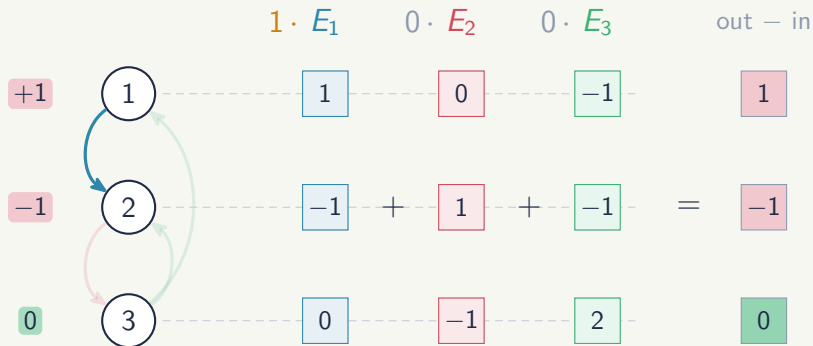
E_3 : node 3 sends **two** edges $\Rightarrow +2$.

Cables – edge groups are vectors



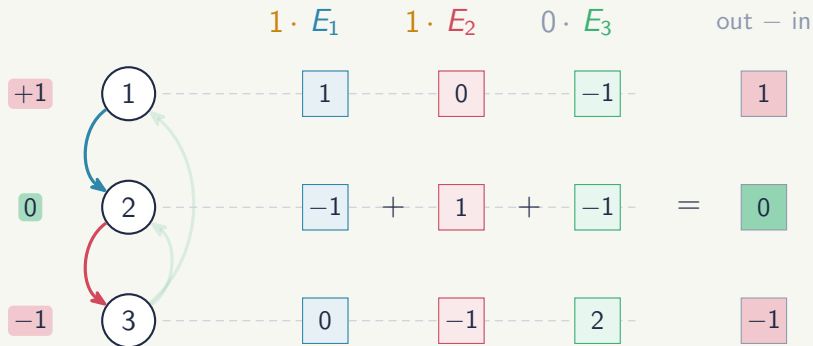
$$E_i[w] = \text{out} - \text{in} \text{ at node } w.$$

Cables – edge groups are vectors



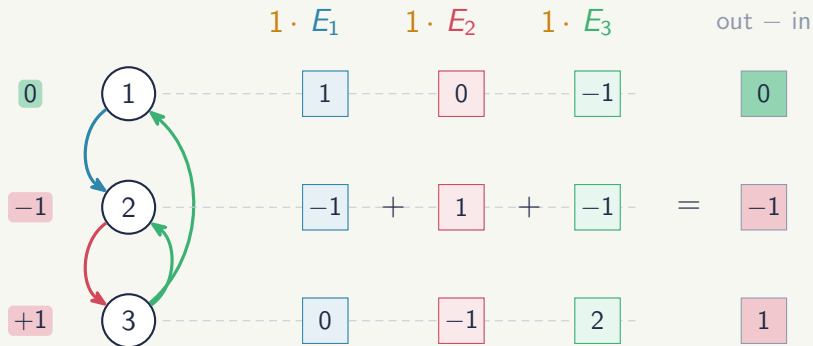
Weight 1 for E_1 only: the sum is $E_1 = \text{out} - \text{in}$ at every node.

Cables – edge groups are vectors



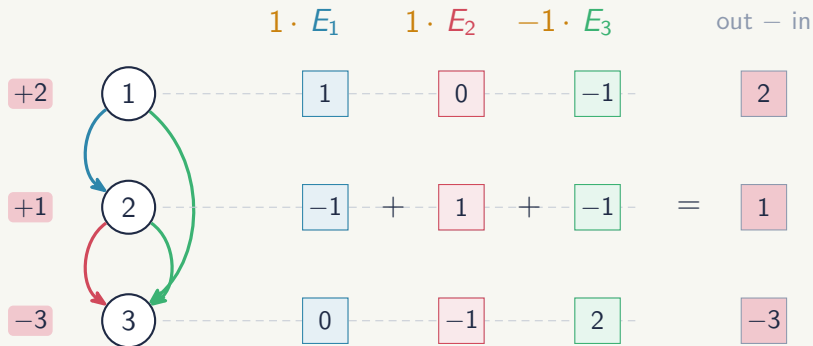
Add E_2 : node 2 is balanced, 1 and 3 are not.

Cables – edge groups are vectors



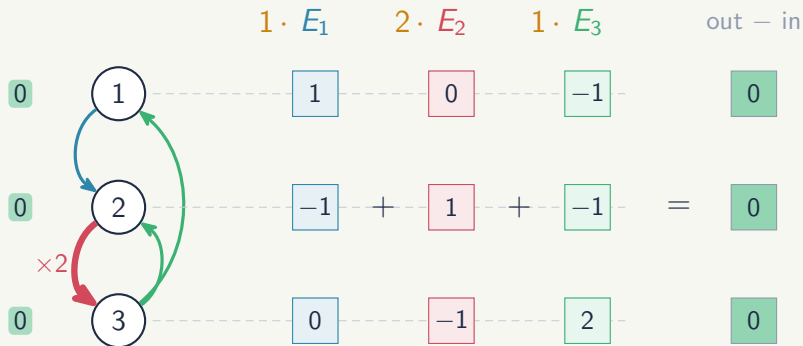
Add E_3 : node 1 is balanced, node 2 now gets too much.

Cables – edge groups are vectors



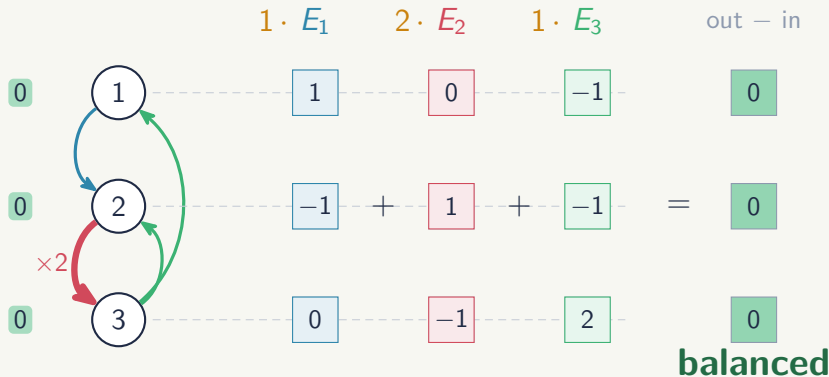
Weight $-1 = E_3$ **reversed**: even worse.

Cables – edge groups are vectors



Weights 1, 2, 1: every entry is 0.

Cables – edge groups are vectors



Balanced $\iff \sum x_q E_q = 0 \iff$ the vectors are **linearly dependent**.

Cables – add groups one by one

	E_1	E_2	E_3
	1→2	2→3	3→1 3→2
1	1	0	-1
2	-1	1	-1
3	0	-1	2

1	2	3	built from
1	-1	0	E_1
0	1	-1	E_2

in hand

Gaussian elimination: E_1, E_2 are new rows.

Cables – add groups one by one

	E_1	E_2	E_3
	1→2	2→3	3→1 3→2
1	1	0	-1
2	-1	1	-1
3	0	-1	2



in hand

	1	2	3	built from
E_1	1	-1	0	
E_2	0	1	-1	
E_3	-1	-1	2	

Add E_3 and eliminate it with the rows.

Cables – add groups one by one

	E_1	E_2	E_3
	1→2	2→3	3→1 3→2
1	1	0	-1
2	-1	1	-1
3	0	-1	2



	1	2	3	built from
E_1	1	-1	0	
E_2	0	1	-1	

in hand

0	-2	2	$E_3 + E_1$
---	----	---	-------------

+ E_1 clears column 1.

Cables – add groups one by one

	E_1	E_2	E_3
	1→2	2→3	3→1 3→2
1	1	0	-1
2	-1	1	-1
3	0	-1	2



	1	2	3	built from
E_1	1	-1	0	
E_2	0	1	-1	

in hand

0	0	0
---	---	---

$$E_3 + E_1 + 2 E_2 = 0$$

+ 2 E_2 : **zero** $\Rightarrow$ dependent, weights 1, 2, 1.

Cables – add groups one by one

	E_1	E_2	E_3
	1→2	2→3	3→1 3→2
1	1	0	-1
2	-1	1	-1
3	0	-1	2



in hand

0	0	0
---	---	---

built from

1	2	3
1	-1	0
0	1	-1

E_1

E_2

$$E_3 + E_1 + 2 E_2 = 0$$

Redo for every start: too slow.

Cables – one pass: keep the newest

	E_1	E_2	E_3	E_4	E_5						
	1→2	3→4	1→2 3→2	2→3	1→3	start	1	2	3	4	built from
1	1	0	1	0	1						
2	-1	0	-2	1	0						
3	0	1	1	-1	-1						
4	0	-1	0	0	0						

in hand

Add the groups one by one. Every row has a **start**: the oldest group in it.

Cables – one pass: keep the newest

	E_1	E_2	E_3	E_4	E_5							
	1→2	3→4	1→2 3→2	2→3	1→3	start	1	2	3	4	built from	
1	1	0	1	0	1							
2	-1	0	-2	1	0							
3	0	1	1	-1	-1							
4	0	-1	0	0	0							
	▲											
						in hand	1	1	-1	0	0	E_1

Take E_1 , start 1. Its first non-zero entry is in column 1.

Cables – one pass: keep the newest

	E_1	E_2	E_3	E_4	E_5
	1→2	3→4	1→2 3→2	2→3	1→3
1	1	0	1	0	1
2	-1	0	-2	1	0
3	0	1	1	-1	-1
4	0	-1	0	0	0



in hand

start	1	2	3	4	built from
1	1	-1	0	0	E_1

Pivot 1 is free $\Rightarrow$ new row.

Cables – one pass: keep the newest

	E_1	E_2	E_3	E_4	E_5	
	1→2	3→4	1→2 3→2	2→3	1→3	start 1 2 3 4 built from
1	1	0	1	0	1	① 1 -1 0 0 E_1
2	-1	0	-2	1	0	
3	0	1	1	-1	-1	
4	0	-1	0	0	0	
		▲				
						in hand ② 0 0 1 -1 E_2

Take E_2 , start 2. First non-zero entry: column 3.

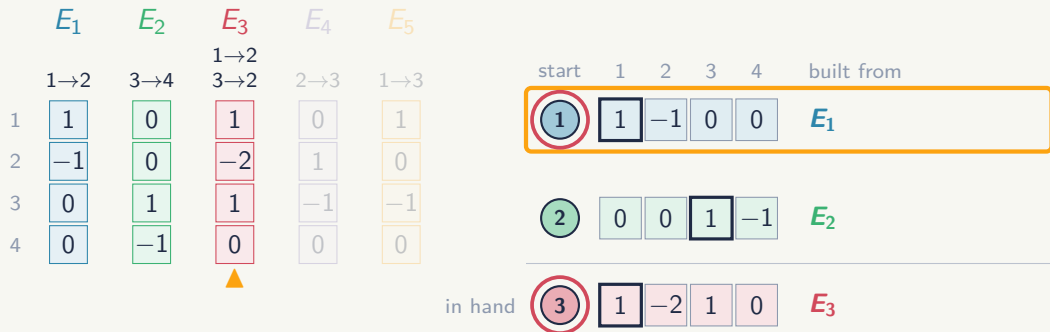
Cables – one pass: keep the newest

	E_1	E_2	E_3	E_4	E_5	
	1→2	3→4	1→2 3→2	2→3	1→3	
1	1	0	1	0	1	start
2	-1	0	-2	1	0	1
3	0	1	1	-1	-1	2
4	0	-1	0	0	0	3
		▲				4
						built from
						E_1
						E_2

in hand

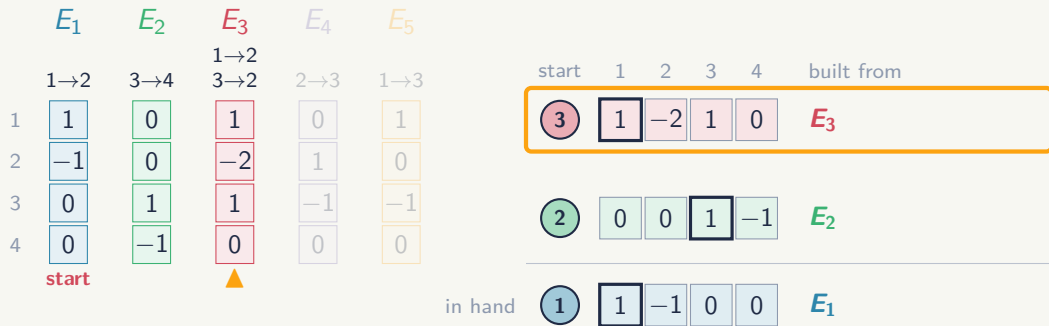
Pivot 3 is free $\Rightarrow$ new row.

Cables – one pass: keep the newest



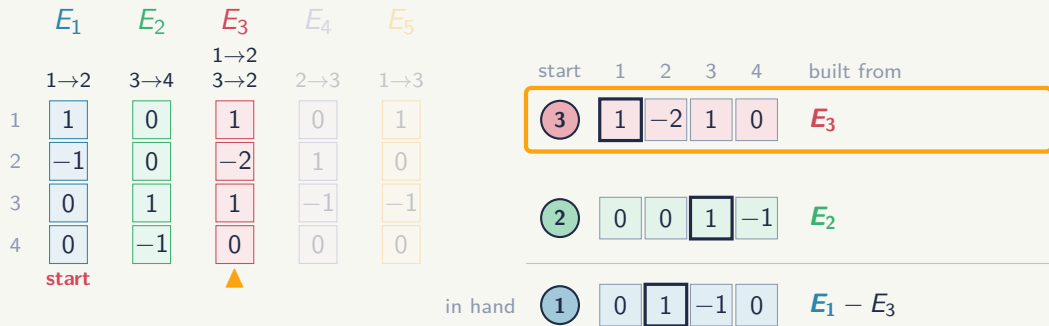
Take E_3 , start 3. Pivot 1 is held by E_1 , start 1.

Cables – one pass: keep the newest



$3 > 1$: E_3 is newer $\Rightarrow$ **swap**: E_3 keeps the row, we go on with E_1 .

Cables – one pass: keep the newest



Eliminate: $E_1 - E_3$ clears column 1. E_3 is newer, so the start stays 1.

Cables – one pass: keep the newest

	E_1	E_2	E_3	E_4	E_5
	1→2	3→4	1→2 3→2	2→3	1→3
1	1	0	1	0	1
2	-1	0	-2	1	0
3	0	1	1	-1	-1
4	0	-1	0	0	0

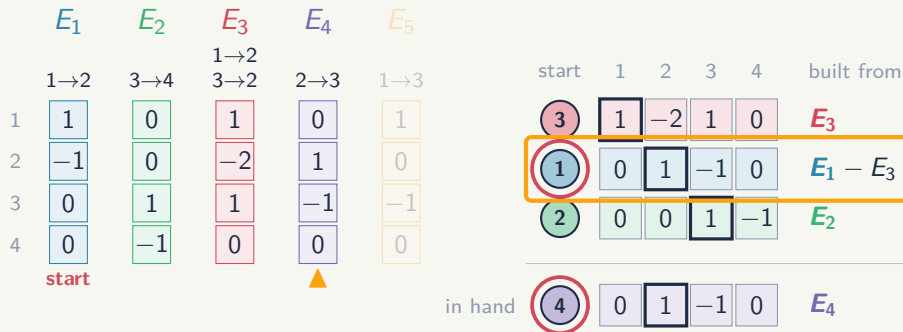
start ▲

in hand

start	1	2	3	4	built from
3	1	-2	1	0	E_3
1	0	1	-1	0	$E_1 - E_3$
2	0	0	1	-1	E_2

Next non-zero entry: column 2. Pivot 2 is free $\Rightarrow$ new row, start 1.

Cables – one pass: keep the newest



Take E_4 , start 4. Pivot 2 is held by $E_1 - E_3$, start 1.

Cables – one pass: keep the newest

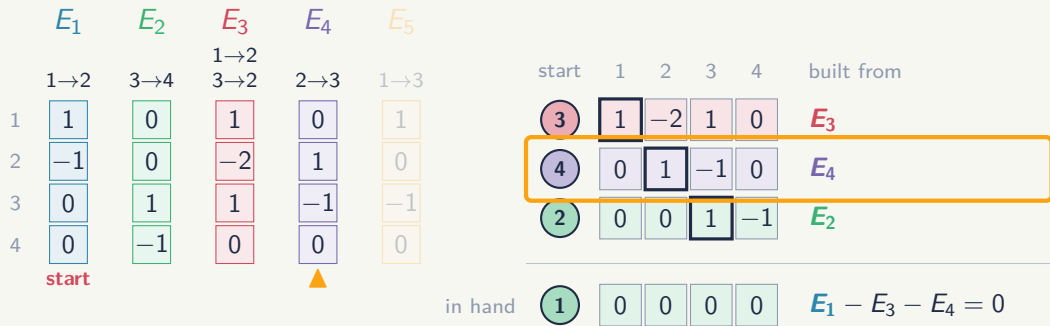
	E_1	E_2	E_3	E_4	E_5	
	1→2	3→4	1→2 3→2	2→3	1→3	
1	1	0	1	0	1	
2	-1	0	-2	1	0	
3	0	1	1	-1	-1	
4	0	-1	0	0	0	
	start			▲		

	start	1	2	3	4	built from
3	1	-2	1	0		E_3
4	0	1	-1	0		E_4
2	0	0	1	-1		E_2

in hand	1	0	1	-1	0	$E_1 - E_3$
---------	---	---	---	----	---	-------------

$4 > 1 \Rightarrow$ **swap**: E_4 keeps the row, we go on with $E_1 - E_3$.

Cables – one pass: keep the newest



Eliminate: $(E_1 - E_3) - E_4 = 0$.

Cables – one pass: keep the newest

	E_1	E_2	E_3	E_4	E_5
	1→2	3→4	1→2 3→2	2→3	1→3
1	1	0	1	0	1
2	-1	0	-2	1	0
3	0	1	1	-1	-1
4	0	-1	0	0	0

start ▲

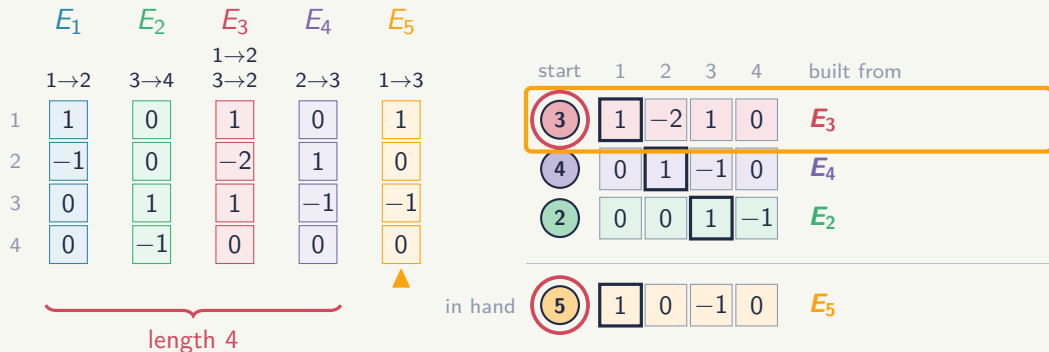
length 4

start	1	2	3	4	built from
3	1	-2	1	0	E_3
4	0	1	-1	0	E_4
2	0	0	1	-1	E_2

in hand	1	0	0	0	$E_1 - E_3 - E_4 = 0$
---------	---	---	---	---	-----------------------

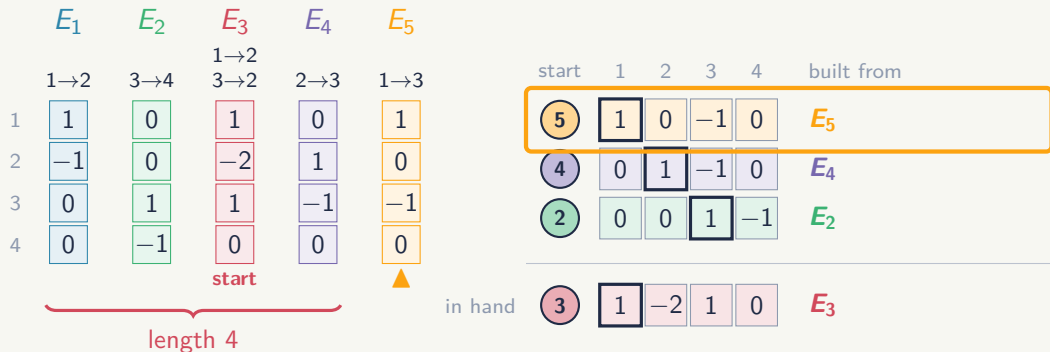
Zero with start 1 $\Rightarrow$ groups 1..4 are dependent, length 4.

Cables – one pass: keep the newest



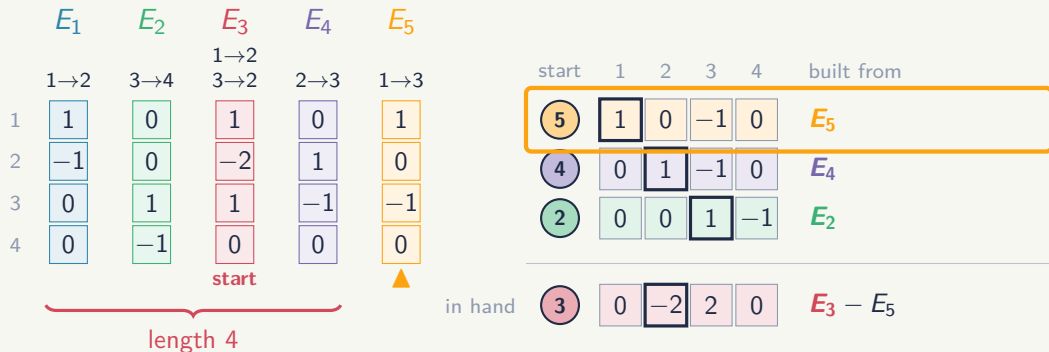
Take E_5 , start 5. Pivot 1 is held by E_3 , start 3.

Cables – one pass: keep the newest



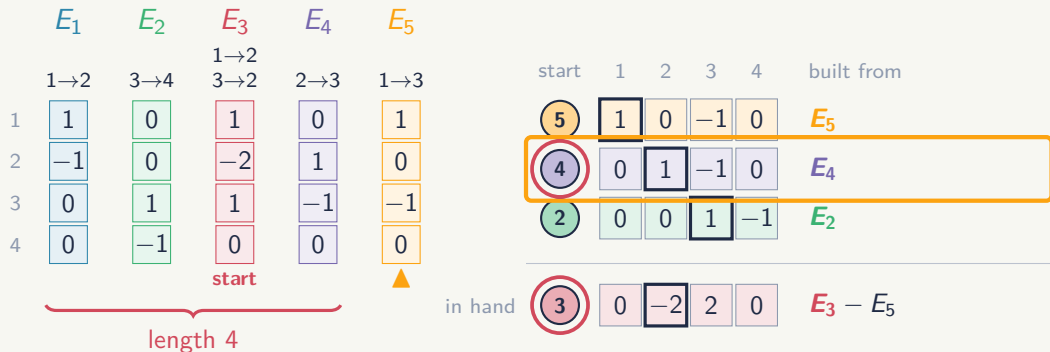
$5 > 3 \Rightarrow$ **swap:** E_5 keeps the row, we go on with E_3 .

Cables – one pass: keep the newest



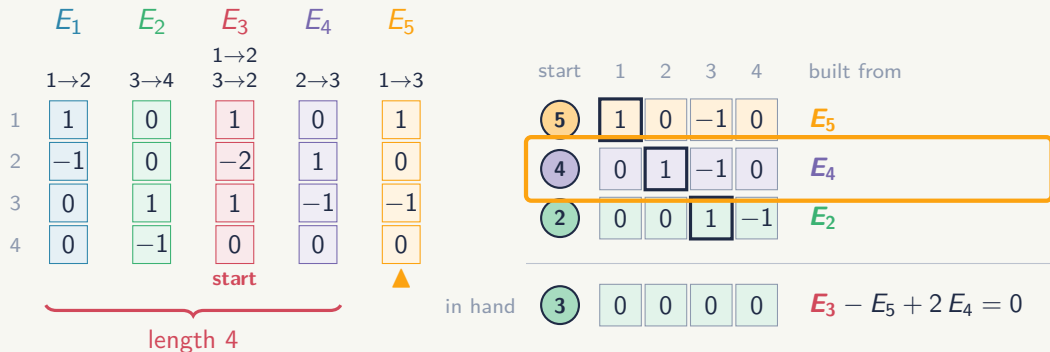
Eliminate: $E_3 - E_5$ clears column 1. The start stays 3.

Cables – one pass: keep the newest



Pivot 2 is held by E_4 , start $4 > 3$: the row is newer $\Rightarrow$ **no swap**.

Cables – one pass: keep the newest



Eliminate: $+ 2 E_4 \Rightarrow$ zero.

Cables – one pass: keep the newest

	E_1	E_2	E_3	E_4	E_5
	1→2	3→4	1→2 3→2	2→3	1→3
1	1	0	1	0	1
2	-1	0	-2	1	0
3	0	1	1	-1	-1
4	0	-1	0	0	0

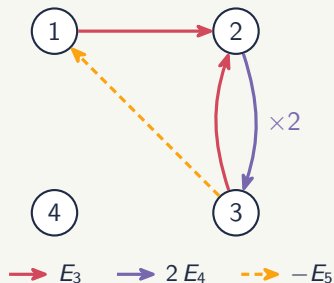
start ▲

length 4 length 3

	start	1	2	3	4	built from
5	1	0	-1	0		E_5
4	0	1	-1	0		E_4
2	0	0	1	-1		E_2
3	0	0	0	0		$E_3 - E_5 + 2E_4 = 0$

Zero with start 3 $\Rightarrow$ groups 3..5, length 3 = answer.

Cables – why it works



Zero with start $t \Rightarrow$ groups $t..R$ are dependent.

Keeping the newest makes t as **late** as possible.

Key idea

$$\text{answer} = \min_R (R - t + 1)$$

Cables – numbers

Use a large prime modulus.

Complexity: $\mathcal{O}\left(\sum |E_i| + k \cdot n^2\right)$

Rope

*Miša lays a loop of rope of length r on a pasture:
how many cows can he catch?*

● ● ● ● ● geometry · convex hull · DP over triangle fans

Rope

Task

Given n points (cows) and a length r , find a closed curve of length $\leq r$ that contains (inside or on it) as many points as possible.

Rope

Task

Given n points (cows) and a length r , find a closed curve of length $\leq r$ that contains (inside or on it) as many points as possible.

▶ $n \leq 100$, $|x|, |y| \leq 100$, $r \leq 10^4$.

Rope

Task

Given n points (cows) and a length r , find a closed curve of length $\leq r$ that contains (inside or on it) as many points as possible.

- ▶ $n \leq 100$, $|x|, |y| \leq 100$, $r \leq 10^4$.
- ▶ General position: no two points equal, **no three collinear**.

Rope

Task

Given n points (cows) and a length r , find a closed curve of length $\leq r$ that contains (inside or on it) as many points as possible.

- ▶ $n \leq 100$, $|x|, |y| \leq 100$, $r \leq 10^4$.
- ▶ General position: no two points equal, **no three collinear**.
- ▶ The curve may touch itself: two cows at distance d cost $2d$ (there and back).

Rope

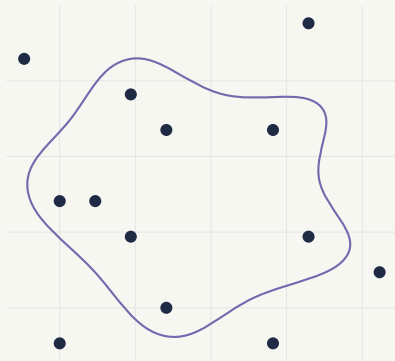
Task

Given n points (cows) and a length r , find a closed curve of length $\leq r$ that contains (inside or on it) as many points as possible.

- ▶ $n \leq 100$, $|x|, |y| \leq 100$, $r \leq 10^4$.
- ▶ General position: no two points equal, **no three collinear**.
- ▶ The curve may touch itself: two cows at distance d cost $2d$ (there and back).
- ▶ Precision: any $r' \in [r - 10^{-5}, r + 10^{-5}]$ gives the same answer, so double comparisons are safe.

Rope – pull the rope tight

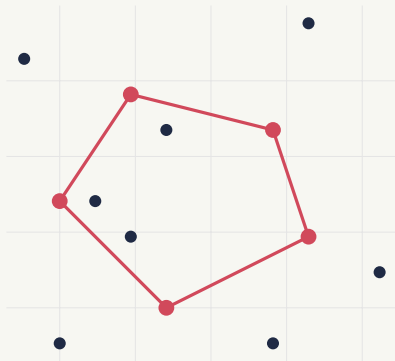
- ▶ Fix the cows S we want. The shortest loop around S is the **perimeter of its convex hull**.



Some loop around the cows ...

Rope – pull the rope tight

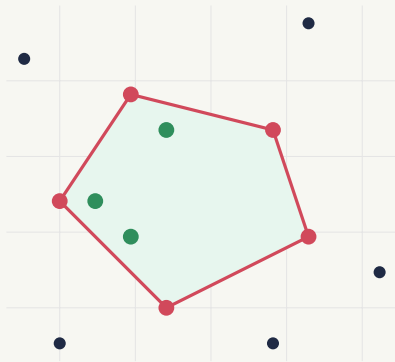
- Fix the cows S we want. The shortest loop around S is the **perimeter of its convex hull**.



... pulled tight: the convex hull.

Rope – pull the rope tight

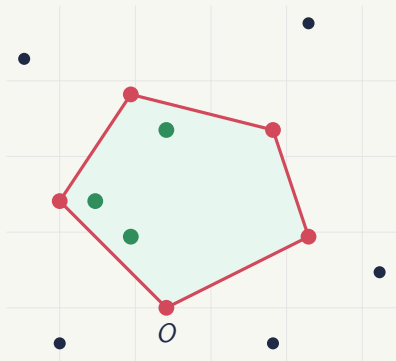
- ▶ Fix the cows S we want. The shortest loop around S is the **perimeter of its convex hull**.
- ▶ We look for a **convex polygon with cows as vertices** and perimeter $\leq r$. Cows inside are caught for free.



Inside cows come for free.

Rope – pull the rope tight

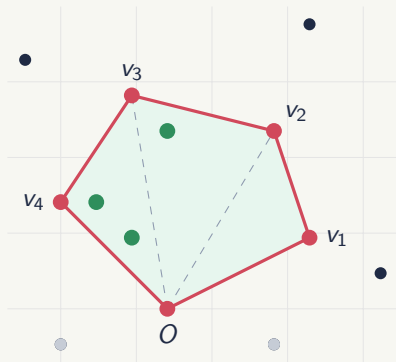
- ▶ Fix the cows S we want. The shortest loop around S is the **perimeter of its convex hull**.
- ▶ We look for a **convex polygon with cows as vertices** and perimeter $\leq r$. Cows inside are caught for free.
- ▶ Root O = vertex with smallest (y, x) . Cows below O are out. Others sorted by angle $([0, \pi))$ give a **fan** of triangles $O v_i v_{i+1}$.



Root O : the lowest cow of the polygon.

Rope – pull the rope tight

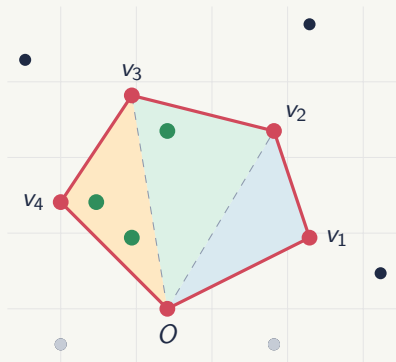
- ▶ Fix the cows S we want. The shortest loop around S is the **perimeter of its convex hull**.
- ▶ We look for a **convex polygon with cows as vertices** and perimeter $\leq r$. Cows inside are caught for free.
- ▶ Root O = vertex with smallest (y, x) . Cows below O are out. Others sorted by angle $([0, \pi))$ give a **fan** of triangles $O v_i v_{i+1}$.



Cows below O are out. Fan from O .

Rope – pull the rope tight

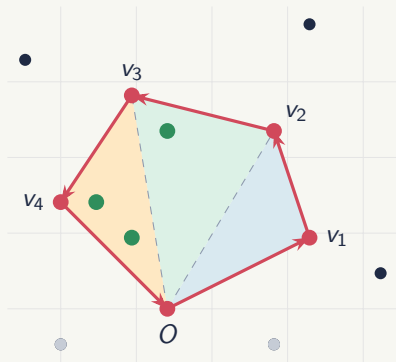
- ▶ Fix the cows S we want. The shortest loop around S is the **perimeter of its convex hull**.
- ▶ We look for a **convex polygon with cows as vertices** and perimeter $\leq r$. Cows inside are caught for free.
- ▶ Root $O =$ vertex with smallest (y, x) . Cows below O are out. Others sorted by angle $([0, \pi))$ give a **fan** of triangles $O v_i v_{i+1}$.
- ▶ No three cows are collinear, so no cow lies on an edge. Thus
$$\#caught = 2 + \sum_i (1 + \text{in}(O, v_i, v_{i+1})).$$



$$2 + (1+0) + (1+1) + (1+2) = 8.$$

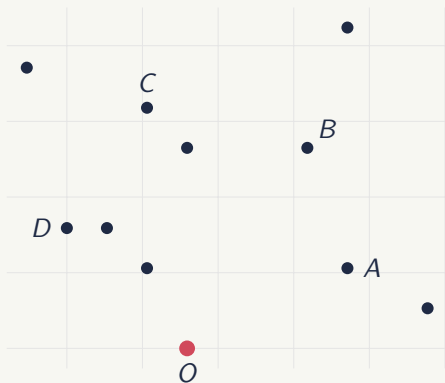
Rope – pull the rope tight

- ▶ Fix the cows S we want. The shortest loop around S is the **perimeter of its convex hull**.
- ▶ We look for a **convex polygon with cows as vertices** and perimeter $\leq r$. Cows inside are caught for free.
- ▶ Root $O =$ vertex with smallest (y, x) . Cows below O are out. Others sorted by angle $([0, \pi))$ give a **fan** of triangles $O v_i v_{i+1}$.
- ▶ No three cows are collinear, so no cow lies on an edge. Thus
$$\#caught = 2 + \sum_i (1 + \text{in}(O, v_i, v_{i+1})).$$
- ▶ Build the fan edge by edge $\Rightarrow$ DP over (last vertex, cows so far).



$O \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow O$.

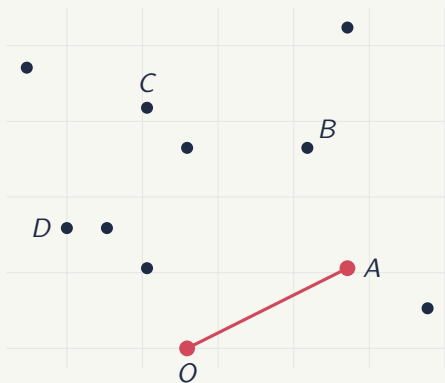
Rope – the DP on an example, $r = 20$



Root O . We build the loop $O A B C D$ one edge at a time.

step	new cows	length	cows
------	----------	--------	------

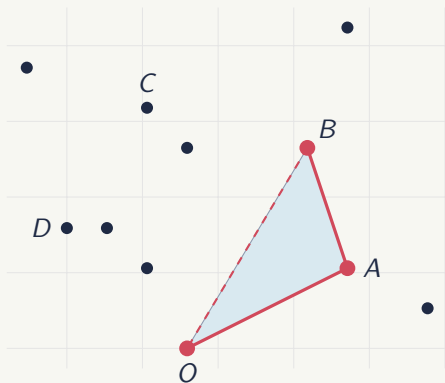
Rope – the DP on an example, $r = 20$



Start: $O \rightarrow A$. Cows O and A . Dashed: the closing edge back to O .

step	new cows	length	cows
$O \rightarrow A$	O, A	4.47	2

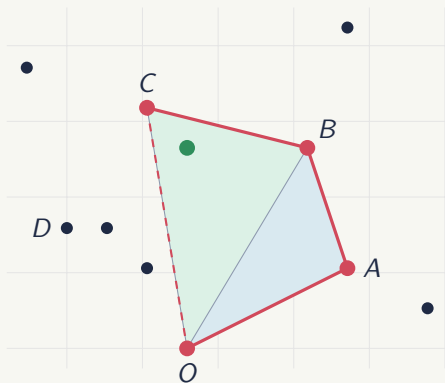
Rope – the DP on an example, $r = 20$



$A \rightarrow B$: add $|AB|$ and cow B . No cow inside OAB .

step	new cows	length	cows
$O \rightarrow A$	O, A	4.47	2
$A \rightarrow B$	$B + 0$	7.63	3

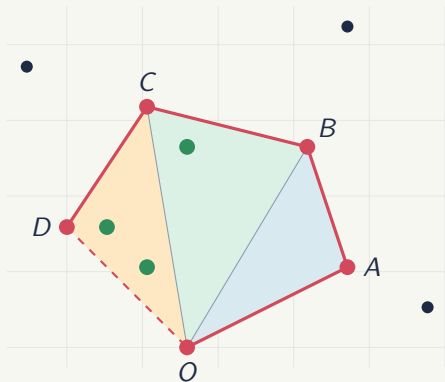
Rope – the DP on an example, $r = 20$



Step 3, $B \rightarrow C$: cow C and the 1 cow inside OBC .

step	new cows	length	cows
$O \rightarrow A$	O, A	4.47	2
$A \rightarrow B$	$B + 0$	7.63	3
$B \rightarrow C$	$C + 1$	11.76	5

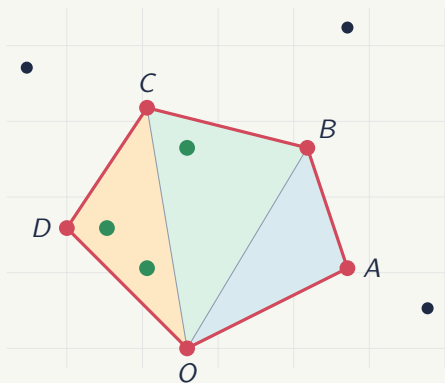
Rope – the DP on an example, $r = 20$



$C \rightarrow D$: cow D and 2 cows inside OCD .

step	new cows	length	cows
$O \rightarrow A$	O, A	4.47	2
$A \rightarrow B$	$B + 0$	7.63	3
$B \rightarrow C$	$C + 1$	11.76	5
$C \rightarrow D$	$D + 2$	15.36	8

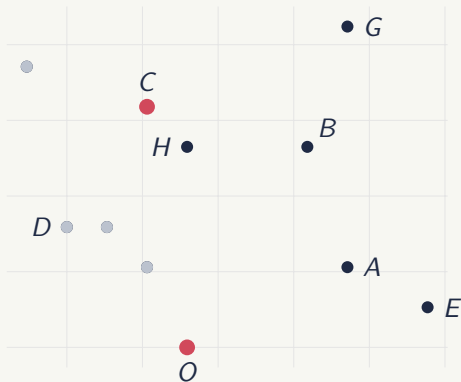
Rope – the DP on an example, $r = 20$



Close $D \rightarrow O$: $19.61 \leq 20 \Rightarrow$ **8 cows**.

step	new cows	length	cows
$O \rightarrow A$	O, A	4.47	2
$A \rightarrow B$	$B + 0$	7.63	3
$B \rightarrow C$	$C + 1$	11.76	5
$C \rightarrow D$	$D + 2$	15.36	8
close $D \rightarrow O$		19.61	8

Rope – the DP on an example, $r = 20$

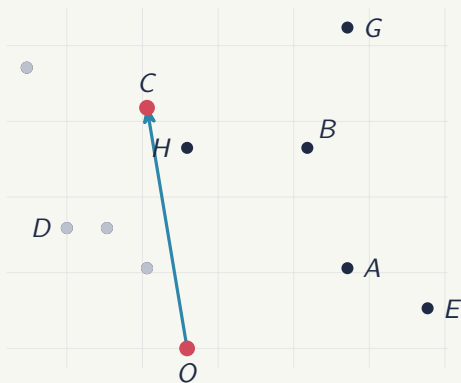


Zoom into C. States with **one** vertex, chain $O \rightarrow v$:
C is the 2nd vertex.

$$f[v][c] + |vC| \rightarrow f[C][c + 1 + \text{in}(O, v, C)]$$

v	state	in	new candidate
-----	-------	----	---------------

Rope – the DP on an example, $r = 20$

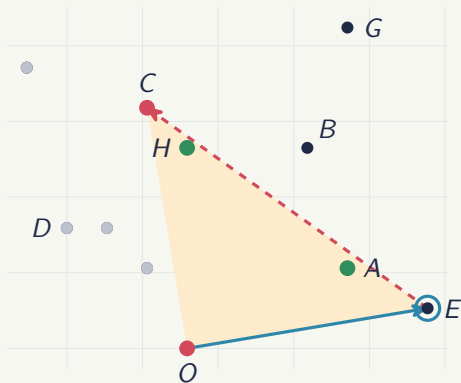


First the shortest chain: just $O \rightarrow C$, so $f[C][2] = |OC| = 6.08$.

$$\underbrace{f[v][c]}_{\text{state}} + |vC| \rightarrow \underbrace{f[C][c+1 + \text{in}(O, v, C)]}_{\text{new candidate}}$$

Blue: the chain behind the state $f[v][c]$.

Rope – the DP on an example, $r = 20$



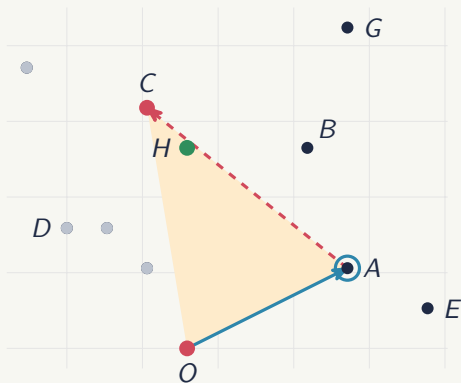
Zoom into C . States with **one** vertex, chain $O \rightarrow v$:
 C is the 2nd vertex.

$$f[v][c] + |vC| \rightarrow f[C][c + 1 + \text{in}(O, v, C)]$$

v	state	in	new candidate
E	$f[E][2]=6.08$	2	$f[C][5] \leftarrow 14.69$

Blue: the chain behind the state $f[v][c]$.

Rope – the DP on an example, $r = 20$



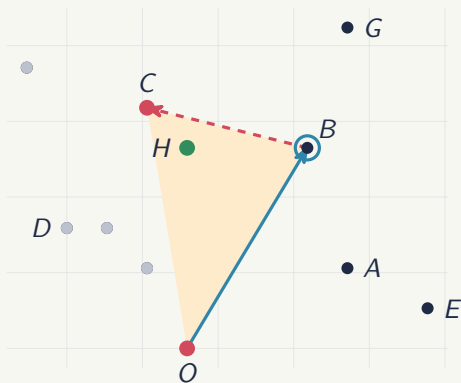
Zoom into C . States with **one** vertex, chain $O \rightarrow v$:
 C is the 2nd vertex.

$$f[v][c] + |vC| \rightarrow f[C][c + 1 + \text{in}(O, v, C)]$$

v	state	in	new candidate
E	$f[E][2]=6.08$	2	$f[C][5] \leftarrow 14.69$
A	$f[A][2]=4.47$	1	$f[C][4] \leftarrow 10.88$

Blue: the chain behind the state $f[v][c]$.

Rope – the DP on an example, $r = 20$



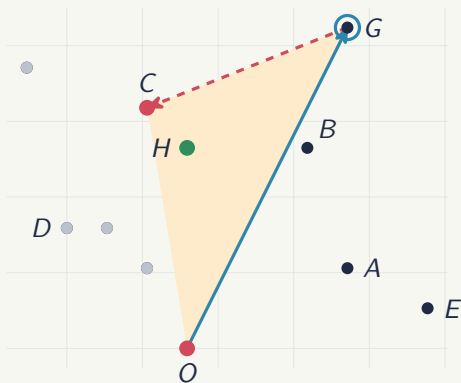
Zoom into C . States with **one** vertex, chain $O \rightarrow v$:
 C is the 2nd vertex.

$$f[v][c] + |vC| \rightarrow f[C][c + 1 + \text{in}(O, v, C)]$$

v	state	in	new candidate
E	$f[E][2]=6.08$	2	$f[C][5] \leftarrow 14.69$
A	$f[A][2]=4.47$	1	$f[C][4] \leftarrow 10.88$
B	$f[B][2]=5.83$	1	$f[C][4] \leftarrow 9.95$

Blue: the chain behind the state $f[v][c]$.

Rope – the DP on an example, $r = 20$



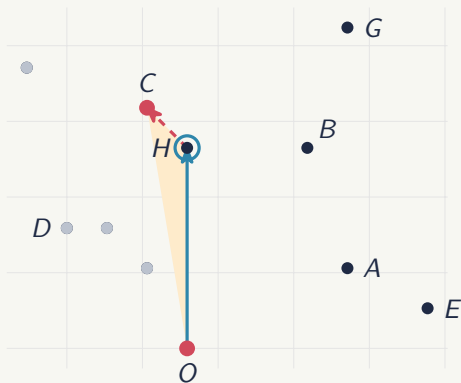
Zoom into C . States with **one** vertex, chain $O \rightarrow v$:
 C is the 2nd vertex.

$$f[v][c] + |vC| \rightarrow f[C][c + 1 + \text{in}(O, v, C)]$$

v	state	in	new candidate
E	$f[E][2]=6.08$	2	$f[C][5] \leftarrow 14.69$
A	$f[A][2]=4.47$	1	$f[C][4] \leftarrow 10.88$
B	$f[B][2]=5.83$	1	$f[C][4] \leftarrow 9.95$
G	$f[G][2]=8.94$	1	$f[C][4] \leftarrow 14.33$

Blue: the chain behind the state $f[v][c]$.

Rope – the DP on an example, $r = 20$



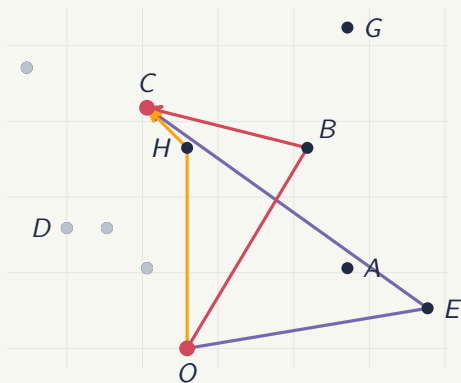
Zoom into C. States with **one** vertex, chain $O \rightarrow v$:
C is the 2nd vertex.

$$f[v][c] + |vC| \rightarrow f[C][c + 1 + \text{in}(O, v, C)]$$

v	state	in	new candidate
E	$f[E][2]=6.08$	2	$f[C][5] \leftarrow 14.69$
A	$f[A][2]=4.47$	1	$f[C][4] \leftarrow 10.88$
B	$f[B][2]=5.83$	1	$f[C][4] \leftarrow 9.95$
G	$f[G][2]=8.94$	1	$f[C][4] \leftarrow 14.33$
H	$f[H][2]=5.00$	0	$f[C][3] \leftarrow 6.41$

Blue: the chain behind the state $f[v][c]$.

Rope – the DP on an example, $r = 20$



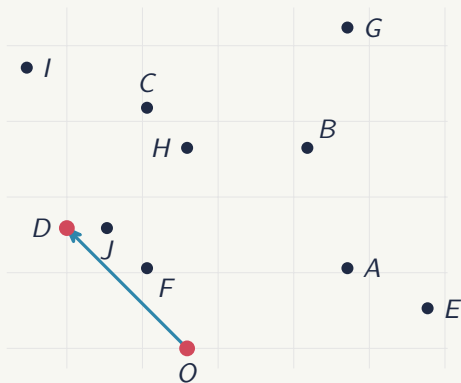
For each count, the DP keeps only the **best** chain. Longer chains improve $f[C][5]$ later to 11.76.

$$f[v][c] + |vC| \rightarrow f[C][c + 1 + \text{in}(O, v, C)]$$

v	state	in	new candidate
E	$f[E][2]=6.08$	2	$f[C][5] \leftarrow 14.69$
A	$f[A][2]=4.47$	1	$f[C][4] \leftarrow 10.88$
B	$f[B][2]=5.83$	1	$f[C][4] \leftarrow 9.95$
G	$f[G][2]=8.94$	1	$f[C][4] \leftarrow 14.33$
H	$f[H][2]=5.00$	0	$f[C][3] \leftarrow 6.41$

One color per count: 3, 4, 5 cows.

Rope – the DP on an example, $r = 20$

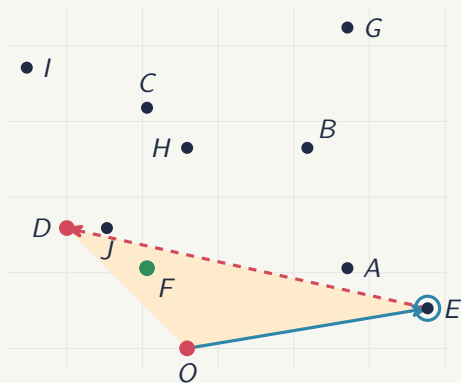


Fill D : list **all** candidates, from every state of every earlier v . Start: $|OD| = 4.24$.

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \setminus c$	2	3	4	5	6	7	8	9	10	11
O	4.2									

Rope – the DP on an example, $r = 20$

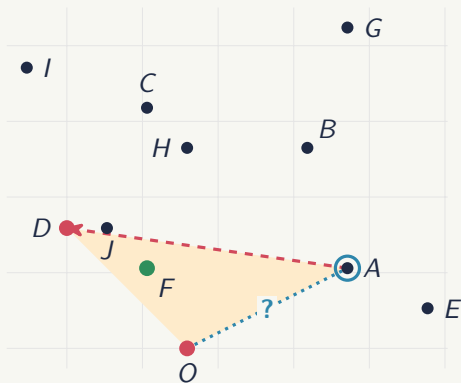


$v = E$: length $+|ED| = +9.22$, cows $+1+1$ (1 inside OED).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \setminus c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							

Rope – the DP on an example, $r = 20$



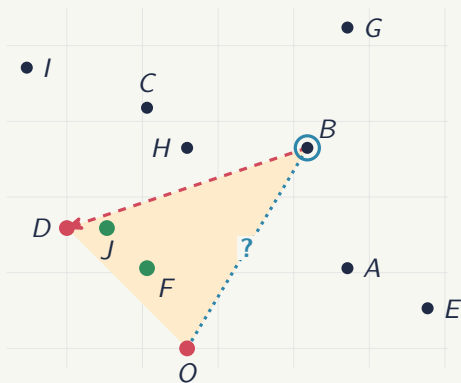
$v = A$: length $+|AD| = +7.07$, cows $+1+1$ (1 inside OAD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \backslash c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						

Blue $?$: the rest of the chain is inside $f[v][c]$.

Rope – the DP on an example, $r = 20$



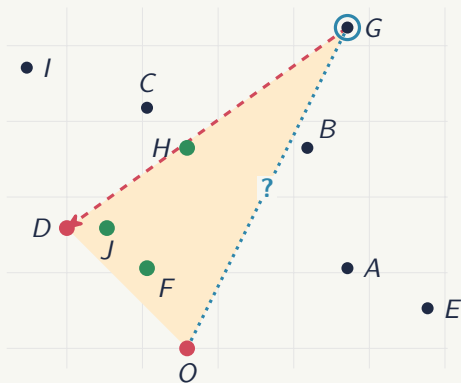
$v = B$: length $+|BD| = +6.32$, cows $+1+2$ (2 inside OBD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \backslash c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				

Blue $?$: the rest of the chain is inside $f[v][c]$.

Rope – the DP on an example, $r = 20$



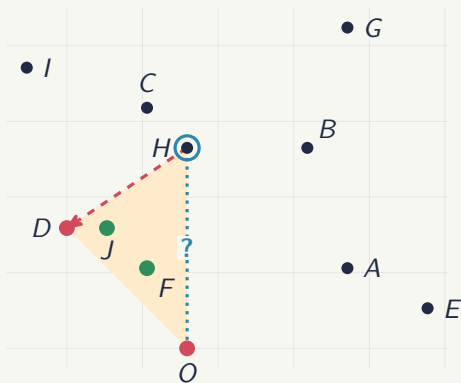
$v = G$: length $+|GD| = +8.60$, cows $+1+3$ (3 inside OGD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \backslash c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		

Blue $?$: the rest of the chain is inside $f[v][c]$.

Rope – the DP on an example, $r = 20$



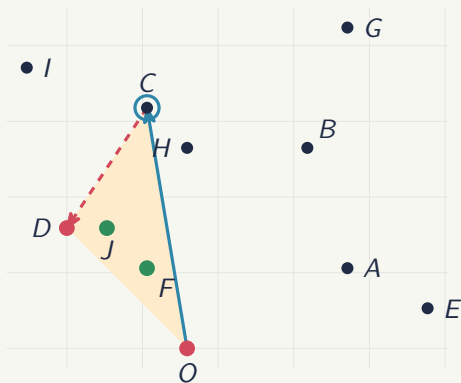
$v = H$: length $+|HD| = +3.61$, cows $+1+2$ (2 inside OHD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \backslash c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		
H				8.6	12.4	14.2	17.7	22.0		

Blue $?$: the rest of the chain is inside $f[v][c]$.

Rope – the DP on an example, $r = 20$

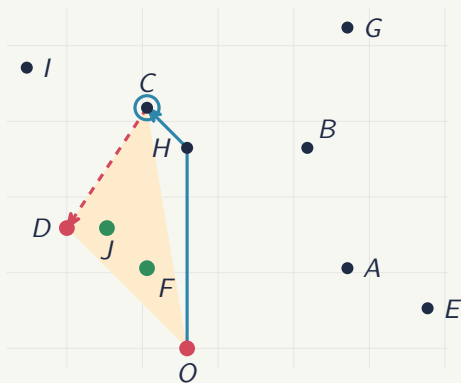


$v = C$: length $+|CD| = +3.61$, cows $+1+2$ (2 inside OCD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \backslash c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		
H				8.6	12.4	14.2	17.7	22.0		
C				9.7						

Rope – the DP on an example, $r = 20$

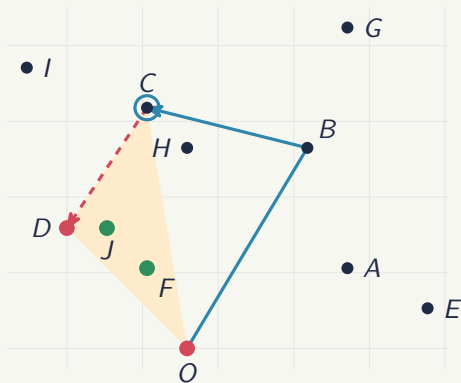


$v = C$: length $+|CD| = +3.61$, cows $+1+2$ (2 inside OCD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \backslash c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		
H				8.6	12.4	14.2	17.7	22.0		
C				9.7	10.0					

Rope – the DP on an example, $r = 20$

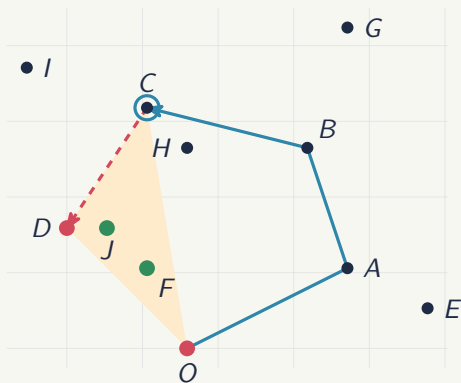


$v = C$: length $+|CD| = +3.61$, cows $+1+2$ (2 inside OCD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \backslash c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		
H				8.6	12.4	14.2	17.7	22.0		
C				9.7	10.0	13.6				

Rope – the DP on an example, $r = 20$

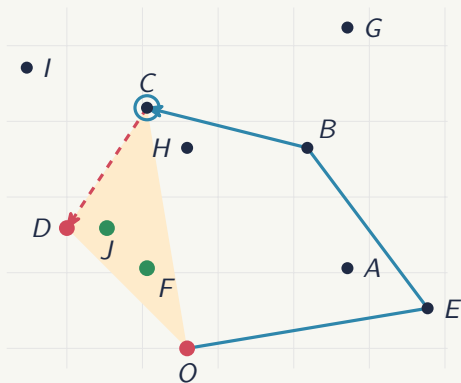


$v = C$: length $+|CD| = +3.61$, cows $+1+2$ (2 inside OCD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \backslash c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		
H				8.6	12.4	14.2	17.7	22.0		
C				9.7	10.0	13.6	15.4			

Rope – the DP on an example, $r = 20$

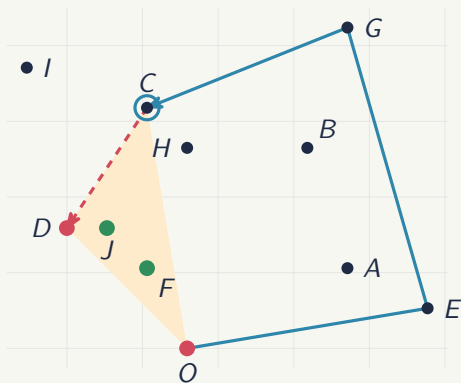


$v = C$: length $+|CD| = +3.61$, cows $+1+2$ (2 inside OCD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \backslash c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		
H				8.6	12.4	14.2	17.7	22.0		
C				9.7	10.0	13.6	15.4	18.8		

Rope – the DP on an example, $r = 20$

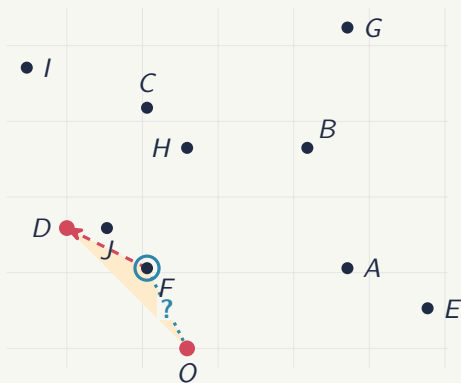


$v = C$: length $+|CD| = +3.61$, cows $+1+2$ (2 inside OCD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \backslash c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		
H				8.6	12.4	14.2	17.7	22.0		
C				9.7	10.0	13.6	15.4	18.8	22.4	

Rope – the DP on an example, $r = 20$



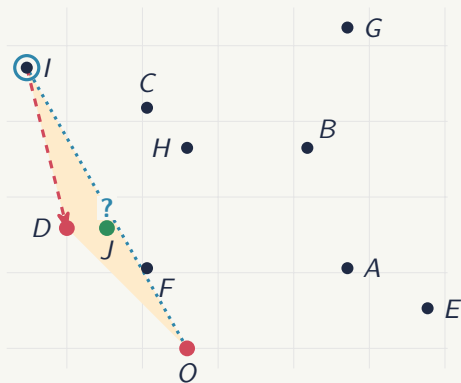
$v = F$: length $+|FD| = +2.24$, cows $+1+0$ (0 inside OFD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \setminus c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		
H				8.6	12.4	14.2	17.7	22.0		
C				9.7	10.0	13.6	15.4	18.8	22.4	
F		4.5	10.4	12.7	16.0	18.0	21.4	25.0		

Blue $?$: the rest of the chain is inside $f[v][c]$.

Rope – the DP on an example, $r = 20$



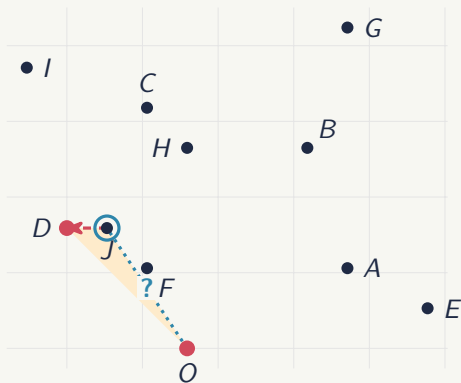
$v = I$: length $+|ID| = +4.12$, cows $+1+1$ (1 inside OID).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \setminus c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		
H				8.6	12.4	14.2	17.7	22.0		
C					9.7	10.0	13.6	15.4	18.8	22.4
F		4.5	10.4	12.7	16.0	18.0	21.4	25.0		
I			12.2	12.2	13.4	13.7	17.2	19.0	22.5	25.5

Blue $?$: the rest of the chain is inside $f[v][c]$.

Rope – the DP on an example, $r = 20$



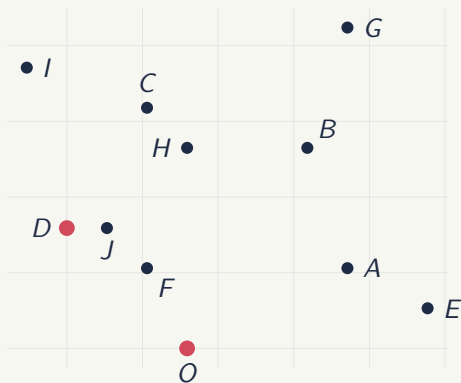
$v = J$: length $+|JD| = +1.00$, cows $+1+0$ (0 inside OJD).

$$f[v][c] + |vD| \rightarrow f[D][c + 1 + \text{in}(O, v, D)]$$

$v \backslash c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		
H				8.6	12.4	14.2	17.7	22.0		
C					9.7	10.0	13.6	15.4	18.8	22.4
F		4.5	10.4	12.7	16.0	18.0	21.4	25.0		
I			12.2	12.2	13.4	13.7	17.2	19.0	22.5	25.5
J		4.6	4.7	8.8	10.6	14.1	15.9	19.4	22.9	26.9

Blue $?$: the rest of the chain is inside $f[v][c]$.

Rope – the DP on an example, $r = 20$

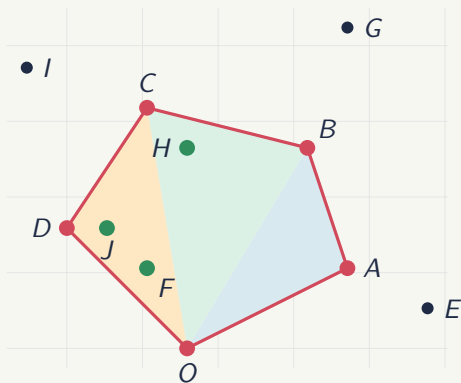


Drop every **dominated** candidate: another one has at least as many cows and is not longer.

What stays: more cows always cost more length.

$v \setminus c$	2	3	4	5	6	7	8	9	10	11
O	4.2									
E			15.3							
A			11.5	15.4						
B				12.2	14.0	17.4				
G					17.5	17.6	19.1	22.0		
H				8.6	12.4	14.2	17.7	22.0		
C				-9.7	10.0	13.6	15.4	18.8	22.4	
F		4.5	10.4	12.7	16.0	18.0	21.4	25.0		
I			12.2	12.2	13.4	13.7	17.2	19.0	22.5	25.5
J		-4.6	4.7	-8.8	10.6	14.1	15.9	19.4	22.9	26.9

Rope – the DP on an example, $r = 20$



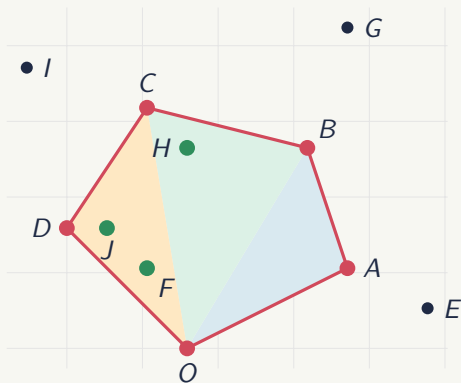
All states of D . Close the loop with $|DO| = 4.24$.

$$\text{answer} = \max \{ c : f[D][c] + |DO| \leq 20 \}$$

c	$f[D][c]$	$+ DO $	c	$f[D][c]$	$+ DO $
2	4.24	8.49	7	13.56	17.80
3	4.47	8.71	8	15.36	19.61
4	4.65	8.89	9	18.81	23.05
5	8.61	12.85	10	22.35	26.60
6	10.02	14.26	11	25.55	29.79

$19.61 \leq 20 \Rightarrow$ **8 cows.**

Rope – the DP on an example, $r = 20$



So the states of D hold the best loop for every count (root O , last vertex D):

cows	loop	length
2	OD	8.49
3	OFD	8.71
4	$OFJD$	8.89
5	OHD	12.85
6	$OHCD$	14.26
7	$OBCD$	17.80
8	$OABCD$	19.61
9	$OEBCD$	23.05
10	$OEGCD$	26.60
11	$OEGID$	29.79

Rope – the algorithm

State: $f[v][c] = \text{min length of a chain } O \rightarrow \dots \rightarrow v \text{ that catches } c \text{ cows.}$

Rope – the algorithm

State: $f[v][c] = \text{min length of a chain } O \rightarrow \dots \rightarrow v \text{ that catches } c \text{ cows.}$

1. For every root O : sort the cows after O in (y, x) order by angle around O .

Rope – the algorithm

State: $f[v][c] = \text{min length of a chain } O \rightarrow \dots \rightarrow v \text{ that catches } c \text{ cows.}$

1. For every root O : sort the cows after O in (y, x) order by angle around O .
2. Precompute $\text{in}(O, v, w)$ for all pairs v, w (not inside the count loop).

Rope – the algorithm

State: $f[v][c] = \text{min length of a chain } O \rightarrow \dots \rightarrow v \text{ that catches } c \text{ cows.}$

1. For every root O : sort the cows after O in (y, x) order by angle around O .
2. Precompute $\text{in}(O, v, w)$ for all pairs v, w (not inside the count loop).
3. Start: $f[v][2] = |Ov|$.

Rope – the algorithm

State: $f[v][c] = \text{min length of a chain } O \rightarrow \dots \rightarrow v \text{ that catches } c \text{ cows.}$

1. For every root O : sort the cows after O in (y, x) order by angle around O .
2. Precompute $\text{in}(O, v, w)$ for all pairs v, w (not inside the count loop).
3. Start: $f[v][2] = |Ov|$.
4. In angle order, for each v , each later w , each c :
$$f[w][c + 1 + \text{in}(O, v, w)] \stackrel{\text{min}}{\leftarrow} f[v][c] + |vw|.$$

Rope – the algorithm

State: $f[v][c] = \text{min length of a chain } O \rightarrow \dots \rightarrow v \text{ that catches } c \text{ cows.}$

1. For every root O : sort the cows after O in (y, x) order by angle around O .
2. Precompute $\text{in}(O, v, w)$ for all pairs v, w (not inside the count loop).
3. Start: $f[v][2] = |Ov|$.
4. In angle order, for each v , each later w , each c :
$$f[w][c + 1 + \text{in}(O, v, w)] \stackrel{\text{min}}{\leftarrow} f[v][c] + |vw|.$$
5. Close the loop: answer = largest c with $f[v][c] + |vO| \leq r$ (at least 1).

Rope – the algorithm

State: $f[v][c] = \text{min length of a chain } O \rightarrow \dots \rightarrow v \text{ that catches } c \text{ cows.}$

1. For every root O : sort the cows after O in (y, x) order by angle around O .
 2. Precompute $\text{in}(O, v, w)$ for all pairs v, w (not inside the count loop).
 3. Start: $f[v][2] = |Ov|$.
 4. In angle order, for each v , each later w , each c :
$$f[w][c + 1 + \text{in}(O, v, w)] \stackrel{\text{min}}{\leftarrow} f[v][c] + |vw|.$$
 5. Close the loop: answer = largest c with $f[v][c] + |vO| \leq r$ (at least 1).
-

- ▶ Count in the state, length as the value (counts are integers, lengths are reals).
- ▶ No convexity check: any fan is a valid loop, the shortest one is convex by minimality.

Rope – the algorithm

State: $f[v][c] = \text{min length of a chain } O \rightarrow \dots \rightarrow v \text{ that catches } c \text{ cows.}$

1. For every root O : sort the cows after O in (y, x) order by angle around O .
2. Precompute $\text{in}(O, v, w)$ for all pairs v, w (not inside the count loop).
3. Start: $f[v][2] = |Ov|$.
4. In angle order, for each v , each later w , each c :
$$f[w][c + 1 + \text{in}(O, v, w)] \stackrel{\text{min}}{\leftarrow} f[v][c] + |vw|.$$
5. Close the loop: answer = largest c with $f[v][c] + |vO| \leq r$ (at least 1).

-
- ▶ Count in the state, length as the value (counts are integers, lengths are reals).
 - ▶ No convexity check: any fan is a valid loop, the shortest one is convex by minimality.

Complexity: $\mathcal{O}(n^4)$

n roots $\times n^2$ pairs $(v, w) \times n$ counts

Thank you, and see you in 2027!