BAPC 2023

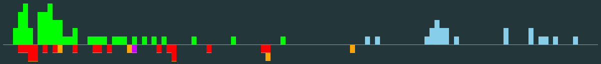
Solutions presentation

The BAPC 2023 jury

October 28, 2023

A: APT Upgrade

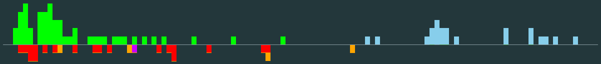
Problem Author: Ragnar Groot Koerkamp



Problem: Calculate the maximum overall completion percentage of downloading n packages, with m packages having finished downloading and k packages underway.

A: APT Upgrade

Problem Author: Ragnar Groot Koerkamp

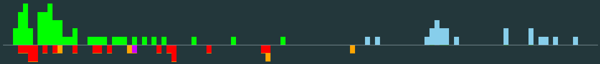


Problem: Calculate the maximum overall completion percentage of downloading n packages, with m packages having finished downloading and k packages underway.

Observation 1: The largest packages need to have finished downloading.

A: APT Upgrade

Problem Author: Ragnar Groot Koerkamp



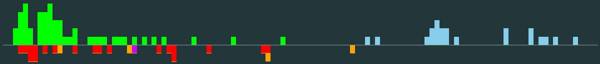
Problem: Calculate the maximum overall completion percentage of downloading n packages, with m packages having finished downloading and k packages underway.

Observation 1: The largest packages need to have finished downloading.

Observation 2: The packages underway need to be the next largest, and at $99.\bar{9}\%$.

A: APT Upgrade

Problem Author: Ragnar Groot Koerkamp



Problem: Calculate the maximum overall completion percentage of downloading n packages, with m packages having finished downloading and k packages underway.

Observation 1: The largest packages need to have finished downloading.

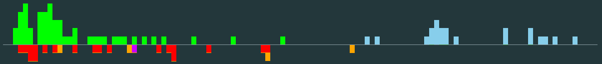
Observation 2: The packages underway need to be the next largest, and at $99.\bar{9}\%$.

Solution: Sort the list, sum the largest $m+k$ packages, divide by the total sum, multiply by 100:

$$\frac{\sum_{i=1}^{m+k} s_i}{\sum_{i=1}^{n} s_i} \cdot 100 \qquad \text{(assuming } s_i \text{ are sorted from large to small)}$$

A: APT Upgrade

Problem Author: Ragnar Groot Koerkamp



Problem: Calculate the maximum overall completion percentage of downloading n packages, with m packages having finished downloading and k packages underway.

Observation 1: The largest packages need to have finished downloading.

Observation 2: The packages underway need to be the next largest, and at $99.\bar{9}\%$.

Solution: Sort the list, sum the largest $m+k$ packages, divide by the total sum, multiply by 100:

$$\frac{\sum_{i=1}^{m+k} s_i}{\sum_{i=1}^{n} s_i} \cdot 100 \qquad \text{(assuming } s_i \text{ are sorted from large to small)}$$

Statistics: 95 submissions, 50 accepted, 21 unknown

B: Battle Bots

Problem Author: Mees de Vries

Problem: Given a size n robot, how many attacks do you need to reduce its size to 0? Two attacks available:

B: Battle Bots

Problem Author: Mees de Vries

Problem: Given a size n robot, how many attacks do you need to reduce its size to 0? Two attacks available:

- Sword: `size = size / 2`

B: Battle Bots

Problem Author: Mees de Vries

Problem: Given a size *n* robot, how many attacks do you need to reduce its size to 0? Two attacks available:

- Sword: `size = size / 2`
- Claw: `size = size - 1`

B: Battle Bots

Problem Author: Mees de Vries

Problem: Given a size n robot, how many attacks do you need to reduce its size to 0? Two attacks available:

- Sword: `size = size / 2`
- Claw: `size = size - 1`

Naive solution: Just try all possible combinations: S, C, SS, SC, CS, CC, SSS, SSC, $\ldots$, until you find one that works.

B: Battle Bots

Problem Author: Mees de Vries

Problem: Given a size n robot, how many attacks do you need to reduce its size to 0? Two attacks available:

- Sword: `size = size / 2`
- Claw: `size = size - 1`

Naive solution: Just try all possible combinations: S, C, SS, SC, CS, CC, SSS, SSC, $\ldots$, until you find one that works.

If m is the answer, this runs in $\mathcal{O}(m2^m)$. Since $m \approx \log_2(n)$, this is $\mathcal{O}(n\log(n))$. Too slow!

B: Battle Bots

Problem Author: Mees de Vries

Problem: Given a size n robot, how many attacks do you need to reduce its size to 0? Two attacks available:

- Sword: `size = size / 2`
- Claw: `size = size - 1`

Observation: An optimal strategy is to use a series of S attacks followed by a single C.

B: Battle Bots

Problem Author: Mees de Vries

Problem: Given a size n robot, how many attacks do you need to reduce its size to 0? Two attacks available:

- Sword: `size = size / 2`
- Claw: `size = size - 1`

Observation: An optimal strategy is to use a series of S attacks followed by a single C.

Solution: Use claw attacks until the remaining size is < 1, then a single claw. Run time: $\mathcal{O}(\log(n))$

B: Battle Bots

Problem Author: Mees de Vries

Problem: Given a size n robot, how many attacks do you need to reduce its size to 0? Two attacks available:

- Sword: `size = size / 2`
- Claw: `size = size - 1`

Observation: An optimal strategy is to use a series of S attacks followed by a single C.

Solution: Use claw attacks until the remaining size is < 1, then a single claw. Run time: $\mathcal{O}(\log(n))$

Solution: You can also compute the answer directly as $\lceil \log_2(n) \rceil + 1$, but only if you either

1. Use `long double` in C++, which has 18 digits of precision
2. Calculate the bit length (in Python: `(x - 1).bit_length() == ceil(log2(x))`)

B: Battle Bots

Problem Author: Mees de Vries

Problem: Given a size n robot, how many attacks do you need to reduce its size to 0? Two attacks available:

- Sword: `size = size / 2`
- Claw: `size = size - 1`

Observation: An optimal strategy is to use a series of S attacks followed by a single C.

Solution: Use claw attacks until the remaining size is < 1, then a single claw. Run time: $\mathcal{O}(\log(n))$

Solution: You can also compute the answer directly as $\lceil \log_2(n) \rceil + 1$, but only if you either

1. Use `long double` in C++, which has 18 digits of precision
2. Calculate the bit length (in Python: `(x - 1).bit_length() == ceil(log2(x))`)

Float note: 64-bit floating-point numbers (`double`) have too low precision (only 15 digits).

B: Battle Bots

Problem Author: Mees de Vries

Problem: Given a size n robot, how many attacks do you need to reduce its size to 0? Two attacks available:

- Sword: `size = size / 2`
- Claw: `size = size - 1`

Observation: An optimal strategy is to use a series of S attacks followed by a single C.

Solution: Use claw attacks until the remaining size is < 1, then a single claw. Run time: $\mathcal{O}(\log(n))$

Solution: You can also compute the answer directly as $\lceil \log_2(n) \rceil + 1$, but only if you either

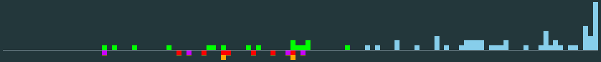
1. Use `long double` in C++, which has 18 digits of precision
2. Calculate the bit length (in Python: `(x - 1).bit_length() == ceil(log2(x))`)

Float note: 64-bit floating-point numbers (`double`) have too low precision (only 15 digits).

Statistics: 127 submissions, 50 accepted, 12 unknown

C: Compressing Commands

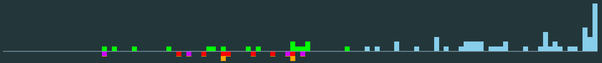
Problem Author: Ragnar Groot Koerkamp



Problem: Which working directory should you use to specify n file paths (with ../), with the minimal number of relative path components?

C: Compressing Commands

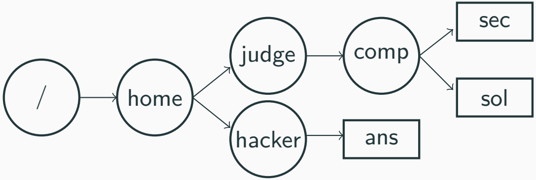
Problem Author: Ragnar Groot Koerkamp



Problem: Which working directory should you use to specify n file paths (with ../), with the minimal number of relative path components?

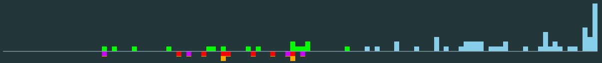
Solution: Convert to tree:

/home/judge/comp/sec
/home/judge/comp/sol
/home/hacker/ans



C: Compressing Commands

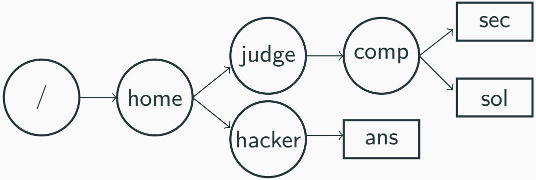
Problem Author: Ragnar Groot Koerkamp



Problem: Which working directory should you use to specify n file paths (with ../), with the minimal number of relative path components?

Solution: Convert to tree:

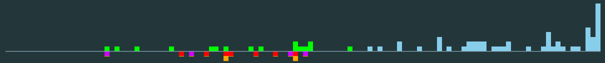
/home/judge/comp/sec
/home/judge/comp/sol
/home/hacker/ans



Compute #path components for all nodes in linear time.

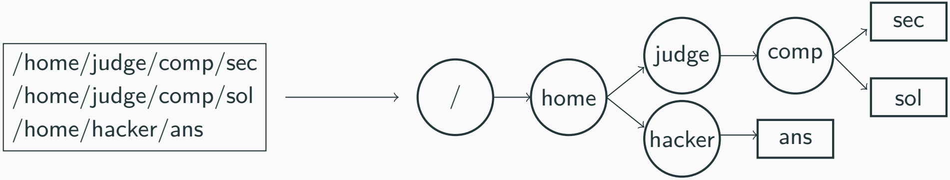
C: Compressing Commands

Problem Author: Ragnar Groot Koerkamp



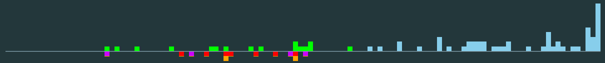
Problem: Which working directory should you use to specify n file paths (with ../), with the minimal number of relative path components?

Solution: Convert to tree:



Compute #path components for all nodes in linear time.

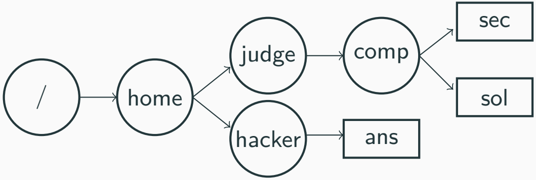
1. $cost("/") = \#\text{total\_path\_components}$.
2. For edge $u \to v$: $\quad cost(v) = cost(u) + n - 2 \cdot \#\text{fileswithprefix}(v)$.
3. Output $\min_u cost(u)$.



Problem: Which working directory should you use to specify n file paths (with ../), with the minimal number of relative path components?

Solution: Convert to tree:

/home/judge/comp/sec
/home/judge/comp/sol
/home/hacker/ans



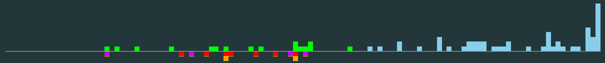
Compute #path components for all nodes in linear time.

1. $cost("/") = \#total\_path\_components$.
2. For edge $u \to v$: $cost(v) = cost(u) + n - 2 \cdot \#fileswithprefix(v)$.
3. Output $\min_u cost(u)$.

Insight: For edge $u \to v$, $cost(v) < cost(u)$ iff $\#fileswithprefix(v) > \frac{n}{2}$.

C: Compressing Commands

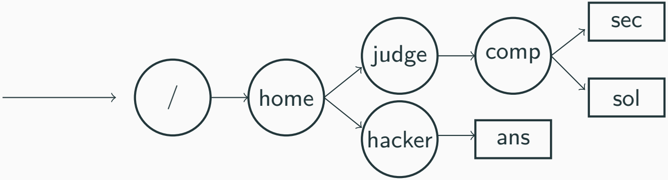
Problem Author: Ragnar Groot Koerkamp



Problem: Which working directory should you use to specify n file paths (with ../), with the minimal number of relative path components?

Solution: Convert to tree:

/home/judge/comp/sec
/home/judge/comp/sol
/home/hacker/ans



Compute #path components for all nodes in linear time.

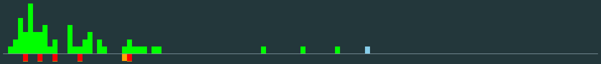
1. $cost("/") = \#\text{total\_path\_components}$.
2. For edge $u \to v$: $\quad cost(v) = cost(u) + n - 2 \cdot \#\text{fileswithprefix}(v)$.
3. Output $\min_u cost(u)$.

Insight: For edge $u \to v$, $cost(v) < cost(u)$ iff $\#\text{fileswithprefix}(v) > \frac{n}{2}$.

Statistics: 82 submissions, 16 accepted, 53 unknown

D: Democratic Naming

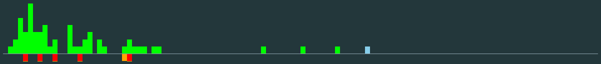
Problem Author: Ivan Fever



Problem: Given the list of city names, determine the new county's name based on the existing city names.

D: Democratic Naming

Problem Author: Ivan Fever

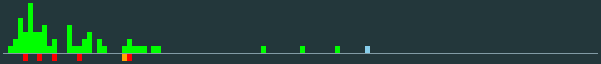


Problem: Given the list of city names, determine the new county's name based on the existing city names.

Observation: Every letter can be handled individually.

D: Democratic Naming

Problem Author: Ivan Fever



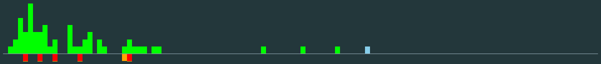
Problem: Given the list of city names, determine the new county's name based on the existing city names.

Observation: Every letter can be handled individually.

Solution: For every letter position, count which letter occurs the most often.

D: Democratic Naming

Problem Author: Ivan Fever



Problem: Given the list of city names, determine the new county's name based on the existing city names.

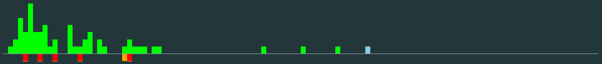
Observation: Every letter can be handled individually.

Solution: For every letter position, count which letter occurs the most often.

Statistics: 63 submissions, 56 accepted, 1 unknown

D: Democratic Naming

Problem Author: Ivan Fever



Problem: Given the list of city names, determine the new county's name based on the existing city names.

Observation: Every letter can be handled individually.

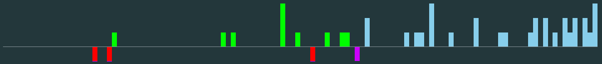
Solution: For every letter position, count which letter occurs the most often.

Statistics: 63 submissions, 56 accepted, 1 unknown

(spoiler: they solved it! 🎈)

E: Exam Study Planning

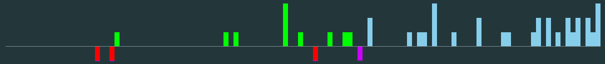
Problem Author: Jorke de Vlas and Reinier Schmiermann



Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

E: Exam Study Planning

Problem Author: Jorke de Vlas and Reinier Schmiermann

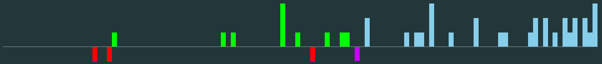


Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

Greedy approach: Study for the first exam you can pass. Doesn't work: maybe you can study for more shorter exams. (Sample 2!)

E: Exam Study Planning

Problem Author: Jorke de Vlas and Reinier Schmiermann



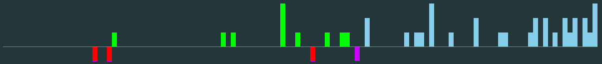
Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

Greedy approach: Study for the first exam you can pass. Doesn't work: maybe you can study for more shorter exams. (Sample 2!)

Greedy approach: Study for the shortest exams first. Doesn't work: maybe you can pass all exams if you study in order, but the first one takes a long time.

E: Exam Study Planning

Problem Author: Jorke de Vlas and Reinier Schmiermann



Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

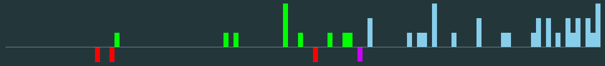
Greedy approach: Study for the first exam you can pass. Doesn't work: maybe you can study for more shorter exams. (Sample 2!)

Greedy approach: Study for the shortest exams first. Doesn't work: maybe you can pass all exams if you study in order, but the first one takes a long time.

Brute force: Try all pass/fail combinations: runs in $\mathcal{O}(2^n)$, way too slow.

E: Exam Study Planning

Problem Author: Jorke de Vlas and Reinier Schmiermann



Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

Greedy approach: Study for the first exam you can pass. Doesn't work: maybe you can study for more shorter exams. (Sample 2!)

Greedy approach: Study for the shortest exams first. Doesn't work: maybe you can pass all exams if you study in order, but the first one takes a long time.

Brute force: Try all pass/fail combinations: runs in $\mathcal{O}(2^n)$, way too slow.

Observation: If at time e_i, end time of exam i, you have passed j exams, and have x minutes of study time unused, it doesn't matter which j exams you passed!

E: Exam Study Planning

Problem Author: Jorke de Vlas and Reinier Schmiermann

Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

Greedy approach: Study for the first exam you can pass. Doesn't work: maybe you can study for more shorter exams. (Sample 2!)

Greedy approach: Study for the shortest exams first. Doesn't work: maybe you can pass all exams if you study in order, but the first one takes a long time.

Brute force: Try all pass/fail combinations: runs in $\mathcal{O}(2^n)$, way too slow.

Observation: If at time e_i, end time of exam i, you have passed j exams, and have x minutes of study time unused, it doesn't matter which j exams you passed!

Use dynamic programming:

$$\mathrm{DP}(i,j) = \begin{cases} x, & \text{max extra study time at } e_i \text{ with } j \text{ exams passed,} \\ -\infty & \text{if it's impossible to pass } j \text{ exams at } e_i. \end{cases}$$

Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

DP

$$\mathrm{DP}(i,j) = \begin{cases} x, & \text{max extra study time at } e_i \text{ with } j \text{ exams passed,} \\ -\infty & \text{if it's impossible to pass } j \text{ exams at } e_i. \end{cases}$$

To determine $\mathrm{DP}(i,j)$ there are two options:

Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

DP

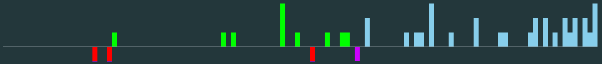
$$\mathrm{DP}(i,j) = \begin{cases} x, & \text{max extra study time at } e_i \text{ with } j \text{ exams passed,} \\ -\infty & \text{if it's impossible to pass } j \text{ exams at } e_i. \end{cases}$$

To determine $\mathrm{DP}(i,j)$ there are two options:

Fail exam i: $\mathrm{DP}(i,j) = \mathrm{DP}(i-1,j) + \underbrace{s_i - e_{i-1}}_{\text{Time between exams}}$

E: Exam Study Planning

Problem Author: Jorke de Vlas and Reinier Schmiermann



Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

DP

$$\mathrm{DP}(i,j) = \begin{cases} x, & \text{max extra study time at } e_i \text{ with } j \text{ exams passed,} \\ -\infty & \text{if it's impossible to pass } j \text{ exams at } e_i. \end{cases}$$

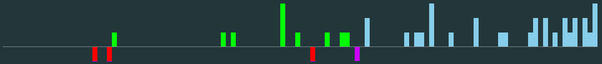
To determine $\mathrm{DP}(i,j)$ there are two options:

Fail exam i: $\mathrm{DP}(i,j) = \mathrm{DP}(i-1,j) + \underbrace{s_i - e_{i-1}}_{\text{Time between exams}}$

Pass exam i: $\mathrm{DP}(i,j) = \mathrm{DP}(i-1,j-1) + \underbrace{s_i - e_{i-1}}_{\text{Time between exams}} - \underbrace{a_i}_{\text{Prep time}} + \underbrace{e_i - p_i}_{\text{Time saved on exam}}$

E: Exam Study Planning

Problem Author: Jorke de Vlas and Reinier Schmiermann



Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

DP

$$\mathrm{DP}(i,j) = \begin{cases} x, & \text{max extra study time at } e_i \text{ with } j \text{ exams passed,} \\ -\infty & \text{if it's impossible to pass } j \text{ exams at } e_i. \end{cases}$$

To determine $\mathrm{DP}(i,j)$ there are two options:

Fail exam i: $\mathrm{DP}(i,j) = \mathrm{DP}(i-1,j) + \underbrace{s_i - e_{i-1}}_{\text{Time between exams}}$

Pass exam i: $\mathrm{DP}(i,j) = \mathrm{DP}(i-1,j-1) + \underbrace{s_i - e_{i-1}}_{\text{Time between exams}} - \underbrace{a_i}_{\text{Prep time}} + \underbrace{e_i - p_i}_{\text{Time saved on exam}}$

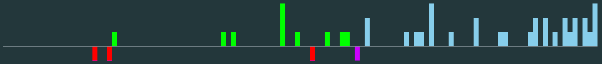
Take the maximum of these options!

Note: you can only pass exam i if you have time to prep:

$$DP(i-1,j-1) + s_i - e_{i-1} \geq a_i.$$

E: Exam Study Planning

Problem Author: Jorke de Vlas and Reinier Schmiermann



Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

DP

$$\mathrm{DP}(i,j) = \begin{cases} x, & \text{max extra study time at } e_i \text{ with } j \text{ exams passed,} \\ -\infty & \text{if it's impossible to pass } j \text{ exams at } e_i. \end{cases}$$

To determine $\mathrm{DP}(i,j)$ there are two options:

Fail exam i: $\mathrm{DP}(i,j) = \mathrm{DP}(i-1,j) + \underbrace{s_i - e_{i-1}}_{\text{Time between exams}}$

Pass exam i: $\mathrm{DP}(i,j) = \mathrm{DP}(i-1,j-1) + \underbrace{s_i - e_{i-1}}_{\text{Time between exams}} - \underbrace{a_i}_{\text{Prep time}} + \underbrace{e_i - p_i}_{\text{Time saved on exam}}$

Take the maximum of these options!

Note: you can only pass exam i if you have time to prep:

$$DP(i-1,j-1) + s_i - e_{i-1} \geq a_i.$$

The solution is $\max\{j : \mathrm{DP}(n,j) \geq 0\}$. Run time: $\mathcal{O}(n^2)$.

E: Exam Study Planning

Problem Author: Jorke de Vlas and Reinier Schmiermann

Problem: Given an exam schedule, determine how many exams you can pass with optimal scheduling.

DP

$$\mathrm{DP}(i,j) = \begin{cases} x, & \text{max extra study time at } e_i \text{ with } j \text{ exams passed,} \\ -\infty & \text{if it's impossible to pass } j \text{ exams at } e_i. \end{cases}$$

To determine $\mathrm{DP}(i,j)$ there are two options:

Fail exam i: $\mathrm{DP}(i,j) = \mathrm{DP}(i-1,j) + \underbrace{s_i - e_{i-1}}_{\text{Time between exams}}$

Pass exam i: $\mathrm{DP}(i,j) = \mathrm{DP}(i-1,j-1) + \underbrace{s_i - e_{i-1}}_{\text{Time between exams}} - \underbrace{a_i}_{\text{Prep time}} + \underbrace{e_i - p_i}_{\text{Time saved on exam}}$

Take the maximum of these options!

Note: you can only pass exam i if you have time to prep:

$$DP(i-1,j-1) + s_i - e_{i-1} \geq a_i.$$

The solution is $\max\{j : \mathrm{DP}(n,j) \geq 0\}$. Run time: $\mathcal{O}(n^2)$.

Statistics: 44 submissions, 10 accepted, 30 unknown

F: Funicular Frenzy

Problem Author: Ragnar Groot Koerkamp

Problem: Determine at which minute you should enter the queue, such that the waiting time is minimized.

F: Funicular Frenzy

Problem Author: Ragnar Groot Koerkamp

Problem: Determine at which minute you should enter the queue, such that the waiting time is minimized.

Solution: Simulation.

- Start the queue with 0 passengers.

F: Funicular Frenzy

Problem Author: Ragnar Groot Koerkamp

Problem: Determine at which minute you should enter the queue, such that the waiting time is minimized.

Solution: Simulation.

- Start the queue with 0 passengers.
- For every minute i, add a_i, save the current queue length, and subtract c.

F: Funicular Frenzy

Problem Author: Ragnar Groot Koerkamp

Problem: Determine at which minute you should enter the queue, such that the waiting time is minimized.

Solution: Simulation.

- Start the queue with 0 passengers.
- For every minute i, add a_i, save the current queue length, and subtract c.
- The queue length can not go negative.

F: Funicular Frenzy

Problem Author: Ragnar Groot Koerkamp

Problem: Determine at which minute you should enter the queue, such that the waiting time is minimized.

Solution: Simulation.

- Start the queue with 0 passengers.
- For every minute i, add a_i, save the current queue length, and subtract c.
- The queue length can not go negative.
- Find the minute for which the queue length was the shortest.

F: Funicular Frenzy

Problem Author: Ragnar Groot Koerkamp

Problem: Determine at which minute you should enter the queue, such that the waiting time is minimized.

Solution: Simulation.

- Start the queue with 0 passengers.
- For every minute i, add a_i, save the current queue length, and subtract c.
- The queue length can not go negative.
- Find the minute for which the queue length was the shortest.

Run time: $\mathcal{O}(n)$.

F: Funicular Frenzy

Problem Author: Ragnar Groot Koerkamp

Problem: Determine at which minute you should enter the queue, such that the waiting time is minimized.

Solution: Simulation.

- Start the queue with 0 passengers.
- For every minute i, add a_i, save the current queue length, and subtract c.
- The queue length can not go negative.
- Find the minute for which the queue length was the shortest.

Run time: $\mathcal{O}(n)$.

Edge case: The answer is "`impossible`" when the current queue length in minute i is never smaller than $c \cdot (n - i)$.

F: Funicular Frenzy

Problem Author: Ragnar Groot Koerkamp

Problem: Determine at which minute you should enter the queue, such that the waiting time is minimized.

Solution: Simulation.

- Start the queue with 0 passengers.
- For every minute i, add a_i, save the current queue length, and subtract c.
- The queue length can not go negative.
- Find the minute for which the queue length was the shortest.

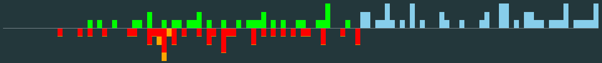
Run time: $\mathcal{O}(n)$.

Edge case: The answer is “`impossible`” when the current queue length in minute i is never smaller than $c \cdot (n - i)$.

Statistics: 111 submissions, 45 accepted, 18 unknown

G: Geometry Game

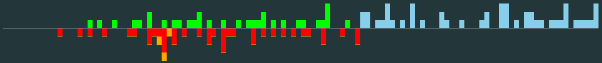
Problem Author: Jorke de Vlas



Problem: Determine the *most restrictive* type of quadrilateral from four points.

G: Geometry Game

Problem Author: Jorke de Vlas

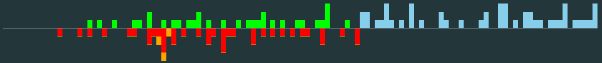


Problem: Determine the *most restrictive* type of quadrilateral from four points.

Possible solution: There are multiple ways of determining the shapes, this is one of them:

- If all four sides have equal length, output “`square`” if the two diagonals have equal length, else “`rhombus`”.
- If two pairs of opposite sides each have equal length, output “`rectangle`” if the two diagonals have equal length, else “`parallelogram`”.
- If two pairs of adjacent sides each have equal length, output “`kite`”.
- If two pairs of opposite sides are parallel, output “`trapezium`”, else “`none`”.

G: Geometry Game



Problem Author: Jorke de Vlas

Problem: Determine the *most restrictive* type of quadrilateral from four points.

Possible solution: There are multiple ways of determining the shapes, this is one of them:

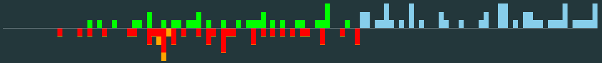
- If all four sides have equal length, output "`square`" if the two diagonals have equal length, else "`rhombus`".
- If two pairs of opposite sides each have equal length, output "`rectangle`" if the two diagonals have equal length, else "`parallelogram`".
- If two pairs of adjacent sides each have equal length, output "`kite`".
- If two pairs of opposite sides are parallel, output "`trapezium`", else "`none`".

Parallel test: Check if out-product of two vectors equals zero:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = x_1 \cdot y_2 - x_2 \cdot y_1 = 0$$

G: Geometry Game

Problem Author: Jorke de Vlas



Problem: Determine the *most restrictive* type of quadrilateral from four points.

Possible solution: There are multiple ways of determining the shapes, this is one of them:

- If all four sides have equal length, output "`square`" if the two diagonals have equal length, else "`rhombus`".
- If two pairs of opposite sides each have equal length, output "`rectangle`" if the two diagonals have equal length, else "`parallelogram`".
- If two pairs of adjacent sides each have equal length, output "`kite`".
- If two pairs of opposite sides are parallel, output "`trapezium`", else "`none`".

Parallel test: Check if out-product of two vectors equals zero:

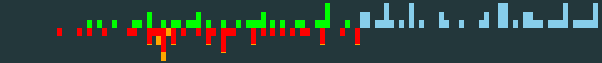
$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = x_1 \cdot y_2 - x_2 \cdot y_1 = 0$$

Float note: Calculating the length of an edge ($\sqrt{x^2 + y^2}$) requires 18 digits (59 bits) of precision. `double` only has 53!

I.e. 64-bit integers (without $\sqrt{}$) or C++ `long double` with an epsilon of 10^{-19} works.

G: Geometry Game

Problem Author: Jorke de Vlas



Problem: Determine the *most restrictive* type of quadrilateral from four points.

Possible solution: There are multiple ways of determining the shapes, this is one of them:

- If all four sides have equal length, output "`square`" if the two diagonals have equal length, else "`rhombus`".
- If two pairs of opposite sides each have equal length, output "`rectangle`" if the two diagonals have equal length, else "`parallelogram`".
- If two pairs of adjacent sides each have equal length, output "`kite`".
- If two pairs of opposite sides are parallel, output "`trapezium`", else "`none`".

Parallel test: Check if out-product of two vectors equals zero:

$$\binom{x_1}{y_1} \times \binom{x_2}{y_2} = x_1 \cdot y_2 - x_2 \cdot y_1 = 0$$

Float note: Calculating the length of an edge ($\sqrt{x^2 + y^2}$) requires 18 digits (59 bits) of precision. `double` only has 53!
I.e. 64-bit integers (without $\sqrt{}$) or C++ `long double` with an epsilon of 10^{-19} works.

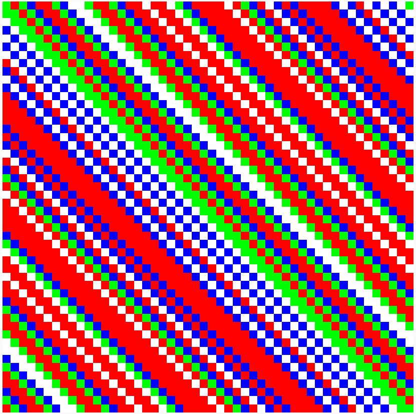
Statistics: 122 submissions, 32 accepted, 48 unknown

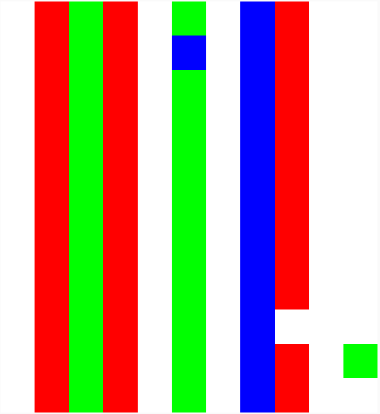
H: Hidden Art

Problem Author: Reinier Schmiermann

Problem: Given an infinitely repeating four-colour pattern, can you find a square whose corners have four different colors?

Problem: Given an infinitely repeating four-colour pattern, can you find a square whose corners have four different colors?

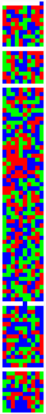


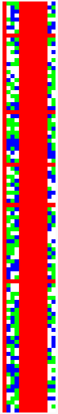


H: Hidden Art

Problem Author: Reinier Schmiermann

Problem: Given an infinitely repeating four-colour pattern, can you find a square whose corners have four different colors?

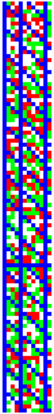


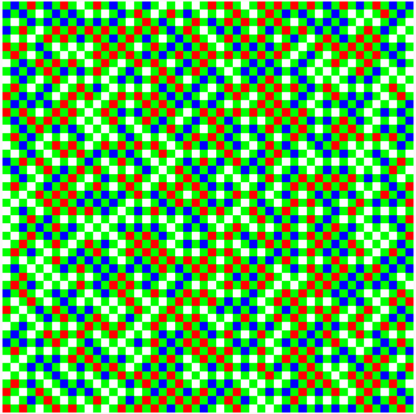


H: Hidden Art

Problem Author: Reinier Schmiermann

Problem: Given an infinitely repeating four-colour pattern, can you find a square whose corners have four different colors?

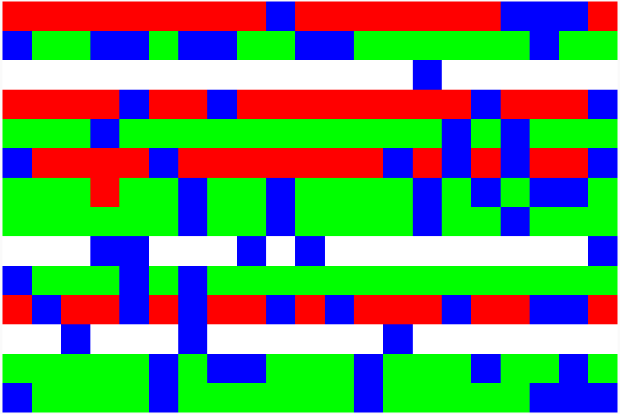


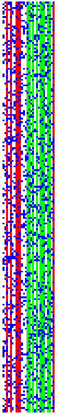


H: Hidden Art

Problem Author: Reinier Schmiermann

Problem: Given an infinitely repeating four-colour pattern, can you find a square whose corners have four different colors?

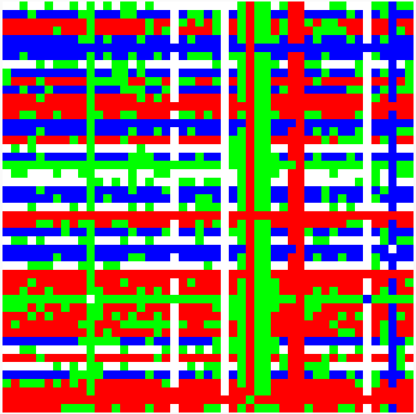


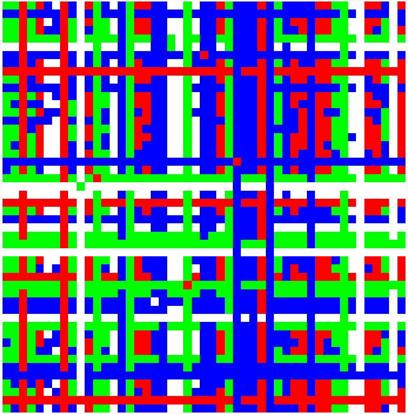


H: Hidden Art

Problem Author: Reinier Schmiermann

Problem: Given an infinitely repeating four-colour pattern, can you find a square whose corners have four different colors?





H: Hidden Art

Problem Author: Reinier Schmiermann

Problem: Given an infinitely repeating four-colour pattern, can you find a square whose corners have four different colors?

Observation: If you have a solution in the infinite grid, then it forms a rectangle in the original grid.

w	r	w	r	w	r
w	g	w	g	w	g
b	g	b	g	b	g
w	r	w	r	w	r
w	g	w	g	w	g
b	g	b	g	b	g

H: Hidden Art

Problem Author: Reinier Schmiermann

Problem: Given an infinitely repeating four-colour pattern, can you find a square whose corners have four different colors?

Observation: If you have a solution in the infinite grid, then it forms a rectangle in the original grid.

w	r	w	r	w	r
w	g	w	g	w	g
b	g	b	g	b	g
w	r	w	r	w	r
w	g	w	g	w	g
b	g	b	g	b	g

Observation: However, not all rectangles in the original grid make squares.

g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w
g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w

Observation: However, not all rectangles in the original grid make squares.

g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w
g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w

Observation: However, not all rectangles in the original grid make squares.

g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w
g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w

Which rectangles correspond to squares?

H: Hidden Art

Problem Author: Reinier Schmiermann

Observation: However, not all rectangles in the original grid make squares.

g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w
g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w

Which rectangles correspond to squares?

Observation: For an $x \times y$ rectangle in a $w \times h$ grid, we can obtain all rectangles $(x + kh) \times (y + \ell w)$.

Observation: However, not all rectangles in the original grid make squares.

g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w
g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w

Which rectangles correspond to squares?

Observation: For an $x \times y$ rectangle in a $w \times h$ grid, we can obtain all rectangles $(x + kh) \times (y + \ell w)$.

Question: For which x, y can we pick k, ℓ such that

$$x + kh = y + \ell w \iff x - y = \ell w - kh?$$

H: Hidden Art

Problem Author: Reinier Schmiermann

Observation: However, not all rectangles in the original grid make squares.

g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w
g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w

Which rectangles correspond to squares?

Observation: For an $x \times y$ rectangle in a $w \times h$ grid, we can obtain all rectangles $(x + kh) \times (y + \ell w)$.

Question: For which x, y can we pick k, ℓ such that

$$x + kh = y + \ell w \iff x - y = \ell w - kh?$$

Answer: Bézout's theorem: if and only if $\gcd(h, w) \mid x - y$.

Observation: However, not all rectangles in the original grid make squares.

g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w
g	w	b	w	g	w	b	w
w	w	r	w	w	w	r	w

Which rectangles correspond to squares?

Observation: For an $x \times y$ rectangle in a $w \times h$ grid, we can obtain all rectangles $(x + kh) \times (y + \ell w)$.

Question: For which x, y can we pick k, ℓ such that

$$x + kh = y + \ell w \iff x - y = \ell w - kh?$$

Answer: Bézout's theorem: if and only if $\gcd(h, w) \mid x - y$.

In the example above: it doesn't work, because $x - y = 2 - 1 = 1$ while $\gcd(w, h) = \gcd(4, 2) = 2$.

H: Hidden Art

Problem Author: Reinier Schmiermann

Naive solution: For every rectangle in the grid, check if its corners have all four colors, and if the difference between height and width is divisible by $g = \gcd(w, h)$.

H: Hidden Art

Problem Author: Reinier Schmiermann

Naive solution: For every rectangle in the grid, check if its corners have all four colors, and if the difference between height and width is divisible by $g = \gcd(w, h)$.

Run time: $\mathcal{O}((hw)^2)$ – too slow for $h \cdot w = 200{,}000$.

H: Hidden Art

Problem Author: Reinier Schmiermann

Naive solution: For every rectangle in the grid, check if its corners have all four colors, and if the difference between height and width is divisible by $g = \gcd(w, h)$.

Run time: $\mathcal{O}((hw)^2)$ – too slow for $h \cdot w = 200{,}000$.

Observation: Once the width of the rectangle is fixed, all possible rectangle heights are known, and they all differ by multiples of g.

H: Hidden Art

Problem Author: Reinier Schmiermann

Naive solution: For every rectangle in the grid, check if its corners have all four colors, and if the difference between height and width is divisible by $g = \gcd(w, h)$.

Run time: $\mathcal{O}((hw)^2)$ – too slow for $h \cdot w = 200{,}000$.

Observation: Once the width of the rectangle is fixed, all possible rectangle heights are known, and they all differ by multiples of g.

Observation: There are not that many combinations of colors possible.

r	b	w	g	r	r
r	w	b	b	g	r
g	r	b	b	w	w
b	b	b	w	g	r
g	r	g	g	r	b
w	w	w	b	w	g
r	w	b	w	g	w
w	r	g	w	b	w
w	r	b	g	w	b

Row 0 (mod 3): { w r b g b g }

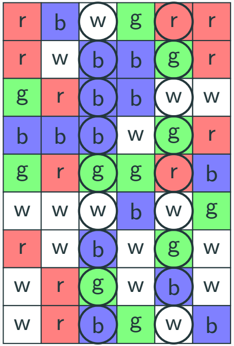
Row 1 (mod 3): { b g g r g b }

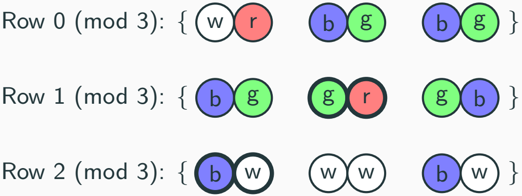
Row 2 (mod 3): { b w w w b w }

Solution: Fix two columns. Then check all colour combinations in those two columns, and store them by their row $(\mathrm{mod}\ g)$.

H: Hidden Art

Problem Author: Reinier Schmiermann





Solution: Fix two columns. Then check all colour combinations in those two columns, and store them by their row $(\mathrm{mod}\ g)$.

Then go through compatible rows, and see if they have compatible color combinations.

H: Hidden Art

Problem Author: Reinier Schmiermann

r	b	w	g	r	r
r	w	b	b	g	r
g	r	b	b	w	w
b	b	b	w	g	r
g	r	g	g	r	b
w	w	w	b	w	g
r	w	b	w	g	w
w	r	g	w	b	w
w	r	b	g	w	b

Row 0 (mod 3): { w r b g b g }

Row 1 (mod 3): { b g g r g b }

Row 2 (mod 3): { b w w w b w }

Solution: Fix two columns. Then check all colour combinations in those two columns, and store them by their row $(\mathrm{mod}\ g)$.

Then go through compatible rows, and see if they have compatible color combinations.

Run time: $\mathcal{O}(hw^2)$ – fast enough, but program efficiently, especially in Python!

H: Hidden Art

Problem Author: Reinier Schmiermann

r	b	w	g	r	r
r	w	b	b	g	r
g	r	b	b	w	w
b	b	b	w	g	r
g	r	g	g	r	b
w	w	w	b	w	g
r	w	b	w	g	w
w	r	g	w	b	w
w	r	b	g	w	b

Row 0 (mod 3): { w r   b g   b g }

Row 1 (mod 3): { b g   g r   g b }

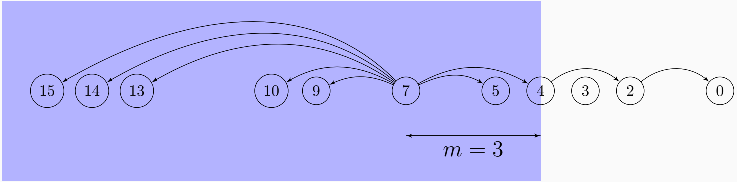
Row 2 (mod 3): { b w   w w   b w }

Solution: Fix two columns. Then check all colour combinations in those two columns, and store them by their row $(\mathrm{mod}\ g)$.
Then go through compatible rows, and see if they have compatible color combinations.

Run time: $\mathcal{O}(hw^2)$ – fast enough, but program efficiently, especially in Python!

Statistics: 63 submissions, 1 accepted, 41 unknown

Problem: Given are $n \leq 10^5$ countries with ascending infection rates r_i, and quarantine times t_i.



- *Hop*: if $r_j \geq r_i - m$, go without quarantine (1 day).
- *Jump*: go with quarantine ($1 + t_j$ days).

Answer 10^5 queries: What is the fastest route from x to y.

Solution If $r_x < r_y$: We can hop directly, so print 1.

I: International Irregularities

Problem Author: Ragnar Groot Koerkamp

Solution If $r_x < r_y$: We can hop directly, so print 1.

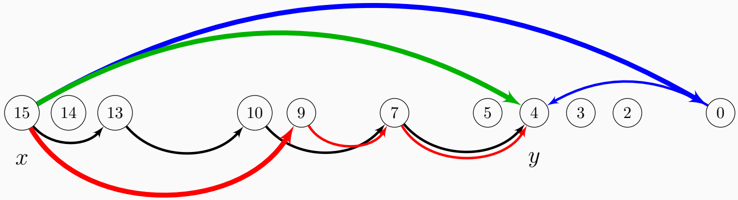
Observation Jump at most once, and only in the very beginning.

Solution If $r_x < r_y$: We can hop directly, so print 1.

Observation Jump at most once, and only in the very beginning.

If $r_x > r_y$, four options:

1. *Hop* to the right up to m at a time.
2. *Jump* directly to y.
3. *Jump* right of y, then hop left once.
4. *Jump* left of y, then hop right some times.



Case 1: Hop to the right up to m at a time.

Define $H_k(i)$ as the rightmost country reachable within 2^k hops.

I: International Irregularities

Problem Author: Ragnar Groot Koerkamp

Case 1: Hop to the right up to m at a time.

Define $H_k(i)$ as the rightmost country reachable within 2^k hops.

Compute H_0 with two-pointers / sliding window.

Case 1: Hop to the right up to m at a time.

Define $H_k(i)$ as the rightmost country reachable within 2^k hops.

Compute H_0 with two-pointers / sliding window.

Compute $H_{k+1}(i)$ as $H_k(H_k(i))$.

Case 1: Hop to the right up to m at a time.

Define $H_k(i)$ as the rightmost country reachable within 2^k hops.

Compute H_0 with two-pointers / sliding window.

Compute $H_{k+1}(i)$ as $H_k(H_k(i))$.

To compute hops from x to y:
Try to go right 2^k steps without overshooting y, for decreasing k.

Case 1: Hop to the right up to m at a time.

Define $H_k(i)$ as the rightmost country reachable within 2^k hops.

Compute H_0 with two-pointers / sliding window.

Compute $H_{k+1}(i)$ as $H_k(H_k(i))$.

To compute hops from x to y:
Try to go right 2^k steps without overshooting y, for decreasing k.

$O(n \log_2(n))$ space and $O(\log_2(n))$ time per query.

Case 1: Hop to the right up to m at a time.

Define $H_k(i)$ as the rightmost country reachable within 2^k hops.

Compute H_0 with two-pointers / sliding window.

Compute $H_{k+1}(i)$ as $H_k(H_k(i))$.

To compute hops from x to y:
Try to go right 2^k steps without overshooting y, for decreasing k.

$O(n \log_2(n))$ space and $O(\log_2(n))$ time per query.

Case 2: Jump directly to y: trivial.

Case 1: Hop to the right up to m at a time.

Define $H_k(i)$ as the rightmost country reachable within 2^k hops.

Compute H_0 with two-pointers / sliding window.

Compute $H_{k+1}(i)$ as $H_k(H_k(i))$.

To compute hops from x to y:
Try to go right 2^k steps without overshooting y, for decreasing k.

$O(n \log_2(n))$ space and $O(\log_2(n))$ time per query.

Case 2: Jump directly to y: trivial.

Case 3: Hop to the right of y, then hop left once.

Case 1: Hop to the right up to m at a time.

Define $H_k(i)$ as the rightmost country reachable within 2^k hops.

Compute H_0 with two-pointers / sliding window.

Compute $H_{k+1}(i)$ as $H_k(H_k(i))$.

To compute hops from x to y:
Try to go right 2^k steps without overshooting y, for decreasing k.

$O(n \log_2(n))$ space and $O(\log_2(n))$ time per query.

Case 2: Jump directly to y: trivial.

Case 3: Hop to the right of y, then hop left once.

Keep suffix-minimum $\min_{j<i} t_j$.

Case 1: Hop to the right up to m at a time.

Define $H_k(i)$ as the rightmost country reachable within 2^k hops.

Compute H_0 with two-pointers / sliding window.

Compute $H_{k+1}(i)$ as $H_k(H_k(i))$.

To compute hops from x to y:
Try to go right 2^k steps without overshooting y, for decreasing k.

$O(n \log_2(n))$ space and $O(\log_2(n))$ time per query.

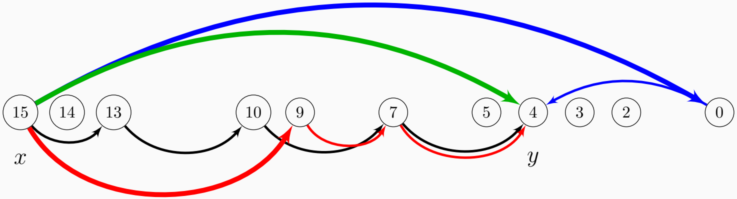
Case 2: Jump directly to y: trivial.

Case 3: Hop to the right of y, then hop left once.

Keep suffix-minimum $\min_{j<i} t_j$.

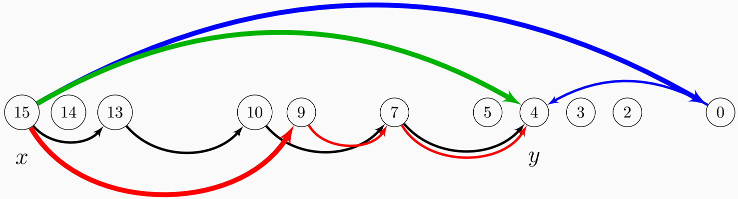
Add one for the hop.

Case 4: Hop to the left of y, then hop right some times.



Iterate through the countries from left to right, keeping track of the best country to jump to first.

Case 4: Hop to the left of y, then hop right some times.

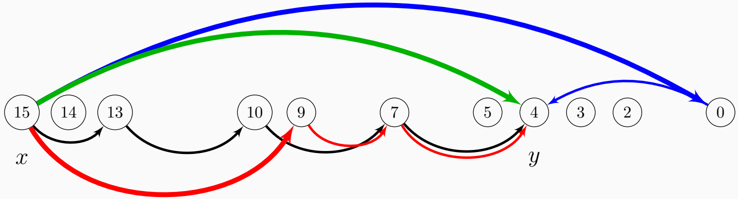


Iterate through the countries from left to right, keeping track of the best country to jump to first.

For each country, either:

- jump to the stored best and hop from there, or
- jump directly and update the stored best.

Case 4: Hop to the left of y, then hop right some times.



Iterate through the countries from left to right, keeping track of the best country to jump to first.

For each country, either:

- jump to the stored best and hop from there, or
- jump directly and update the stored best.

Statistics: 14 submissions, 0 accepted, 12 unknown

J: Jungle Job

Problem Author: Jorke de Vlas and Mike de Vries

Problem: Given a tree, can you find the number of connected subtrees of each size? (modulo $10^9 + 7$ because the answer is huge).

J: Jungle Job

Problem Author: Jorke de Vlas and Mike de Vries

Problem: Given a tree, can you find the number of connected subtrees of each size? (modulo $10^9 + 7$ because the answer is huge).

Observation: Let's define $F(v, c)$ - the number of connected subtrees, that have node v as the root and have exactly c nodes.

Problem: Given a tree, can you find the number of connected subtrees of each size? (modulo $10^9 + 7$ because the answer is huge).

Observation: Let's define $F(v, c)$ - the number of connected subtrees, that have node v as the root and have exactly c nodes.

If we can compute F, we can get the answer to the problem by calculating $\sum_{i \in V} F(i, c)$.

Problem: Given a tree, can you find the number of connected subtrees of each size? (modulo $10^9 + 7$ because the answer is huge).

Observation: Let's define $F(v, c)$ - the number of connected subtrees, that have node v as the root and have exactly c nodes.

If we can compute F, we can get the answer to the problem by calculating $\sum_{i \in V} F(i, c)$.

Solution: Use dynamic programming

Base case: If v is a leaf:

$F(v, c) = 1$ if c is 0 or 1.

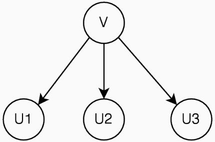
$F(v, c) = 0$ if $c \geq 2$.

Base case: If v is a leaf:

$F(v,c) = 1$ if c is 0 or 1.

$F(v,c) = 0$ if $c \geq 2$.

DP idea: Consider the following subtree:

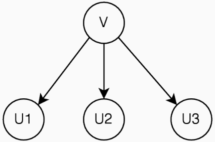


Base case: If v is a leaf:

$F(v, c) = 1$ if c is 0 or 1.

$F(v, c) = 0$ if $c \geq 2$.

DP idea: Consider the following subtree:



To calculate $F(v, c)$ we need to consider every way to distribute $c - 1$ remaining nodes among three child subtrees of v:

$F(v, c) = \sum_{c_1=0}^{c-1} \sum_{c_2=0}^{c-1-c_1} F(u_1, c_1) F(u_2, c_2) F(u_3, c - c_1 - c_2)$

J: Jungle Job

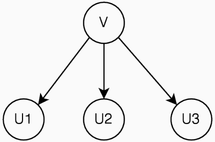
Problem Author: Jorke de Vlas and Mike de Vries

Base case: If v is a leaf:

$F(v, c) = 1$ if c is 0 or 1.

$F(v, c) = 0$ if $c \geq 2$.

DP idea: Consider the following subtree:



To calculate $F(v, c)$ we need to consider every way to distribute $c - 1$ remaining nodes among three child subtrees of v:

$F(v, c) = \sum_{c_1=0}^{c-1} \sum_{c_2=0}^{c-1-c_1} F(u_1, c_1) F(u_2, c_2) F(u_3, c - c_1 - c_2)$

Problem: For a node with many children m, this will hit the time limit:

$$F(v, c) = \sum_{c_1=0}^{c-1} \sum_{c_2=0}^{c-1-c_1} \dots \sum_{c_{n-1}=0}^{c-1-\dots} \prod_{i=1}^{m} F(u_i, c_i)$$

Fix: Introduce $F'(v, i, c)$ - the number of connected subtrees, that have node v as the root, have exactly c nodes and only include first i children of node v.

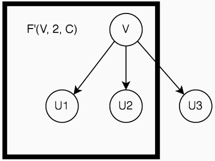
Fix: Introduce $F'(v, i, c)$ - the number of connected subtrees, that have node v as the root, have exactly c nodes and only include first i children of node v.

Base cases for node v that has m children:

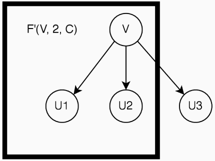
$F(v, c) = F'(v, m, c)$,

$F'(v, 1, c) = F(u_1, c - 1)$,

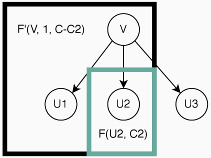
DP Let's calculate $F'(v, 2, c)$ for this graph:



DP Let's calculate $F'(v, 2, c)$ for this graph:



For that we just need to decide how many nodes will be in the subtree of the second child and then we can recurse: $F'(v, 2, c) = \sum_{c2=0}^{c-1} F'(v, 1, c - c_2)F(u_2, c_2)$



Runtime: Computing $F'(v, i, c)$ for all c takes $O(|u_i| \cdot \sum_j |u_j|)$ time, where $|u_i|$ denotes the size of the subtree at u_i.

Runtime: Computing $F'(v, i, c)$ for all c takes $O(|u_i| \cdot \sum_j |u_j|)$ time, where $|u_i|$ denotes the size of the subtree at u_i.

Total time spent at $|v|$ is $O(\sum_i \sum_j |u_i| \cdot |u_j|)$.

Runtime: Computing $F'(v, i, c)$ for all c takes $O(|u_i| \cdot \sum_j |u_j|)$ time, where $|u_i|$ denotes the size of the subtree at u_i.

Total time spent at $|v|$ is $O(\sum_i \sum_j |u_i| \cdot |u_j|)$.

Observation: $\sum_i \sum_j |u_i| \cdot |u_j|$ is the number of pairs of nodes with lowest common ancestor v.

Runtime: Computing $F'(v, i, c)$ for all c takes $O(|u_i| \cdot \sum_j |u_j|)$ time, where $|u_i|$ denotes the size of the subtree at u_i.

Total time spent at $|v|$ is $O(\sum_i \sum_j |u_i| \cdot |u_j|)$.

Observation: $\sum_i \sum_j |u_i| \cdot |u_j|$ is the number of pairs of nodes with lowest common ancestor v.

Since every pair of nodes has one LCA, the total runtime is $O(n^2)$.

J: Jungle Job

Problem Author: Jorke de Vlas and Mike de Vries

Runtime: Computing $F'(v, i, c)$ for all c takes $O(|u_i| \cdot \sum_j |u_j|)$ time, where $|u_i|$ denotes the size of the subtree at u_i.

Total time spent at $|v|$ is $O(\sum_i \sum_j |u_i| \cdot |u_j|)$.

Observation: $\sum_i \sum_j |u_i| \cdot |u_j|$ is the number of pairs of nodes with lowest common ancestor v.

Since every pair of nodes has one LCA, the total runtime is $O(n^2)$.

Statistics: 14 submissions, 6 accepted, 5 unknown

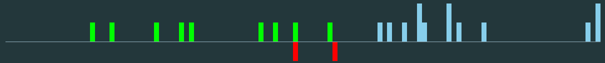
D – Kindergarten Excursion

- If a 1 is to the left of a 0, these two have to be swapped at some point. The same is true for 2/0 and 2/1.
- Process the sequence from left to right. Keep track of the number of 1's and 2's to the left of current number and calculate the result.
- Watch out for overflow.
- Linear time solution.

Statistics: 60 submissions, 12 correct, first at 1:13:57.

L: Locking Doors

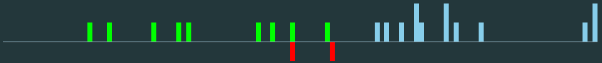
Problem Author: Jorke de Vlas and Mike de Vries



Problem: Given the layout of a building, with doors that lock from only one side, how many exits on the outside do we need to close all doors?

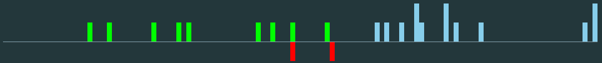
L: Locking Doors

Problem Author: Jorke de Vlas and Mike de Vries



Problem: Given the layout of a building, with doors that lock from only one side, how many exits on the outside do we need to close all doors?

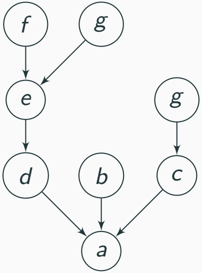
Observation: If a room *a* has an exit, then which doors can we close using that exit?



Problem: Given the layout of a building, with doors that lock from only one side, how many exits on the outside do we need to close all doors?

Observation: If a room a has an exit, then which doors can we close using that exit?

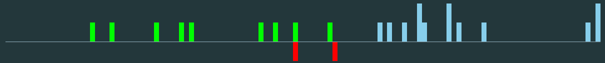
Write $b \rightarrow a$ to mean there is a door you can close from side a. Then consider:



If there is an exit at a, you can close all these doors: just start at any leaf, close that door, and repeat.

L: Locking Doors

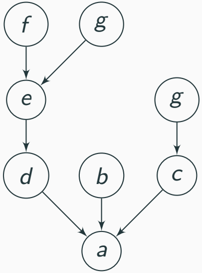
Problem Author: Jorke de Vlas and Mike de Vries



Problem: Given the layout of a building, with doors that lock from only one side, how many exits on the outside do we need to close all doors?

Observation: If a room a has an exit, then which doors can we close using that exit?

Write $b \to a$ to mean there is a door you can close from side a. Then consider:



If there is an exit at a, you can close all these doors: just start at any leaf, close that door, and repeat.

Maybe you can close *more* doors, but definitely these ones.

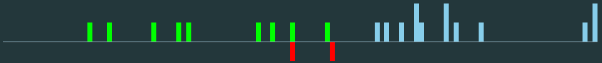
Strongly connected: For a, b nodes, if you can walk from a to b via arrows, and also b to a, we call a and b *strongly connected*.

Strongly connected: For a, b nodes, if you can walk from a to b via arrows, and also b to a, we call a and b *strongly connected*.

SCC: We can collect strongly connected nodes into groups, called *strongly connected components*.

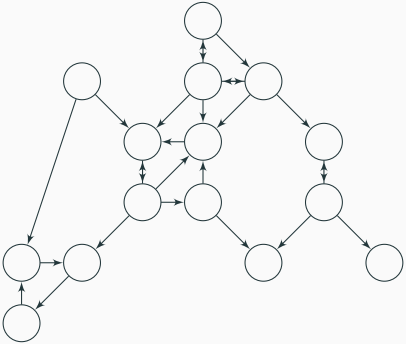
Strongly connected: For a, b nodes, if you can walk from a to b via arrows, and also b to a, we call a and b *strongly connected*.

SCC: We can collect strongly connected nodes into groups, called *strongly connected components*.



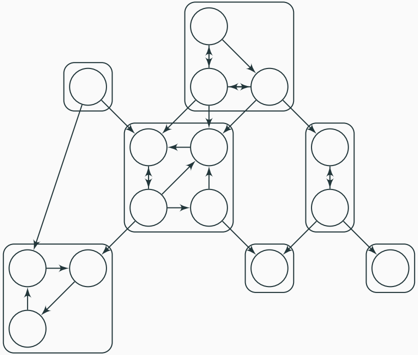
Strongly connected: For a, b nodes, if you can walk from a to b via arrows, and also b to a, we call a and b *strongly connected*.

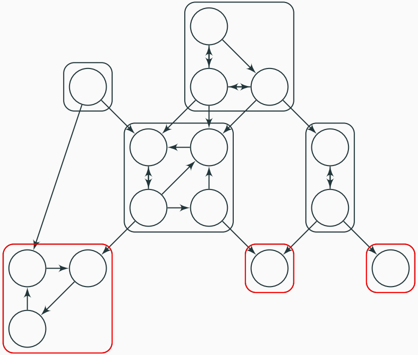
SCC: We can collect strongly connected nodes into groups, called *strongly connected components*. Those components themselves form an acyclic graph.



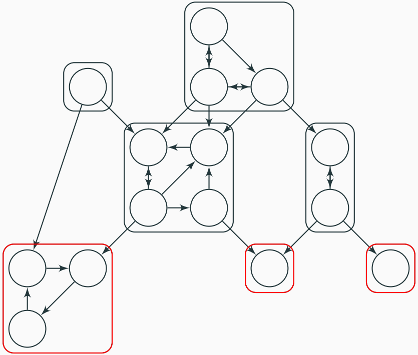
Strongly connected: For a, b nodes, if you can walk from a to b via arrows, and also b to a, we call a and b *strongly connected*.

SCC: We can collect strongly connected nodes into groups, called *strongly connected components*. Those components themselves form an acyclic graph.





Necessary How many exits does this graph need? We need *at least one* in the (red) components without outgoing edges. Otherwise you can never leave it once you close the last incoming door.

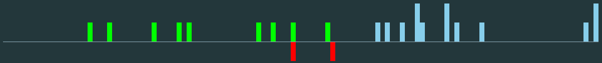


Necessary How many exits does this graph need? We need *at least one* in the (red) components without outgoing edges. Otherwise you can never leave it once you close the last incoming door.

Sufficient That is also *enough exits*: from any node you can follow the arrows to one of those components, which we saw is enough to close all doors.

L: Locking Doors

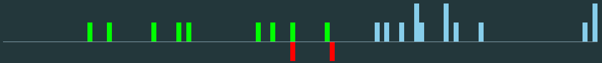
Problem Author: Jorke de Vlas and Mike de Vries



Solution Find the strongly connected components, e.g. with Tarjan's algorithm. Output the number of SCCs without outgoing edges.

L: Locking Doors

Problem Author: Jorke de Vlas and Mike de Vries

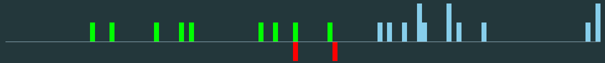


Solution Find the strongly connected components, e.g. with Tarjan's algorithm. Output the number of SCCs without outgoing edges.

Since we only have to count root-SCCs, simpler algorithms are also possible.

L: Locking Doors

Problem Author: Jorke de Vlas and Mike de Vries



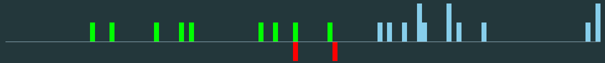
Solution Find the strongly connected components, e.g. with Tarjan's algorithm. Output the number of SCCs without outgoing edges.

Since we only have to count root-SCCs, simpler algorithms are also possible.

Note Be careful with recursion on python. Use a stack instead.

L: Locking Doors

Problem Author: Jorke de Vlas and Mike de Vries



Solution Find the strongly connected components, e.g. with Tarjan's algorithm. Output the number of SCCs without outgoing edges.

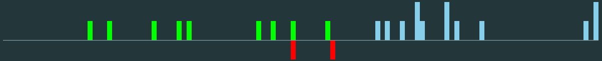
Since we only have to count root-SCCs, simpler algorithms are also possible.

Note Be careful with recursion on python. Use a stack instead.

Runtime Runs in $\mathcal{O}(m)$.

L: Locking Doors

Problem Author: Jorke de Vlas and Mike de Vries



Solution Find the strongly connected components, e.g. with Tarjan's algorithm. Output the number of SCCs without outgoing edges.

Since we only have to count root-SCCs, simpler algorithms are also possible.

Note Be careful with recursion on python. Use a stack instead.

Runtime Runs in $\mathcal{O}(m)$.

Statistics: 24 submissions, 9 accepted, 13 unknown