ICPC 2020 - Amritapuri Regionals

Presentation of solutions

Credits!

- **Chief Judge**: Balajiganapathi **Balajiganapathi** S
- **Contest Admin**: Vichitr **Vichitr** Gandas
- **Problem Coordinators**: Ashish **Ashishgup** Gupta, Kevin **kevinsogo** Atienza, Vaibhav **xennygrimmato** Tulsyan
- **Setters**: Kevin **kevinsogo** Atienza, Ajinkya **ajinkya1p3** Parab, Vichitr **Vichitr** Gandas, Vaibhav **xennygrimmato** Tulsyan
- **Testers**: Kshitij **kshitij_sodani** Sodani, Ashish **Ashishgup** Gupta, Kevin **kevinsogo** Atienza, Ajinkya **ajinkya1p3** Parab, Udit **T1duS** Sanghi, Divyansh **failed_coder** Verma, Naveen **mnaveenkumar2009** Kumar, Soumyaditya **socho** Choudhuri, Shashwat **SleepyShashwat** Chandra, Suneet **ScarletS** Mahajan, Vichitr **Vichitr** Gandas, Ritul **ritul_kr_singh** Singh, Het **hetshah1998** Shah, Varad **playing_it_my_way** Kulkarni, Aswin **aswinashok44** Ashok, Mayank **katana_handler** Pugalia, Saarang **saarang123** Srinivasan and Vaibhav **xennygrimmato** Tulsyan
- **Platform Coordinator**: Deepa **deepa_panwar** Panwar
- **RCD**: Maheshwara Chaitanya

Cakewalk	1. Thanos The Teacher
Simple	2. The Invasion of **Balaji**
Easy	3. Hackerland
Easy-Medium	4. Peace, Freedom, Justice, and Security
Easy-Medium	5. Lord of the Group
Medium	6. DFS Sequences
Medium	7. Blizzard Blitz
Medium-Hard	8. The Lag
Medium-Hard	9. A Cubical Romance
Hard	10. Dice Climbers
Hard	11. The Cat

Problem 1

Thanos The Teacher

No. of accepted solutions: 721

Author: Kevin Charles Atienza

Thanos The Teacher

- Let A be the common average.
- Each pair has an average of A, so they must be symmetric about A.
- Thus, the *whole* array must also be symmetric about A.
- Therefore,
  - the smallest gets paired with the largest,
  - the second smallest with the second largest,
  - etc.
- **Solution:** Sort the array, and pair them up end to end.
- $\mathbf{O(n \log n)}$ (even $O(n^2)$ is acceptable)

Problem 2

The Invasion of **Balaji**

No. of accepted solutions: 539

Author: Vichitr


invasion of
balaji

The Invasion of Balaji

- *Taking stuff from ends* is equivalent to *leaving a subarray*.
- Thus, the BtS Army is the largest subarray sum after q trainings.
- If $T[i] \leq 0$, then it makes no sense to train i.
  - Equivalently, set $T[i] := \max(T[i], 0)$.
- The subarray may be empty, so we initialize $ans := 0$.

The Invasion of Balaji

- **Insight:** We should only train one person.
- *Proof:*
  - If we train $i \neq j$, then presumably, they're both part of the subarray.
    - There's no point in training removed soldiers.
  - If $T[i] > T[j]$, replace training j with training i to yield a higher total.
  - Similarly, if $T[i] \leq T[j]$, replace training i with training j.
  - Therefore, there's an optimal solution where we train at most one soldier.

The Invasion of Balaji

- If we train i, then the subarray will contain i.
- So it has three parts:
  1. a subarray ending at $i - 1$ (possibly empty)
  2. i itself
  3. a subarray starting at $i + 1$ (possibly empty)
- Let
  - $last[i]$ = largest subarray sum ending at i
  - $first[i]$ = largest subarray sum starting at i
  - both possibly empty.

The Invasion of Balaji

- Then $\mathrm{start}[i]$ and $\mathrm{end}[i]$ can be computed with DP.
  - $\mathrm{first}[i] = \max(0,\ U[i] + \mathrm{first}[i + 1])$
  - $\mathrm{last}[i] = \max(0,\ U[i] + \mathrm{last}[i - 1])$
- The best result if we want to train i is then:
  - $\mathrm{last}[i-1] + U[i] + T[i] \cdot q + \mathrm{first}[i+1]$
- The answer is the largest of these among $i = 1, 2, \ldots, n$.
- Time complexity: $\mathbf{O(n)}$

Problem 3

Hackerland

No. of accepted solutions: 391

Author: Ajinkya Parab

Hackerland

- Two numbers have $\gcd > 1$ iff they share a prime factor.
- So, create a node for every prime appearing as a factor of some $R[i]$.
- Add an edge $(\text{prime } p, \text{city } i)$ if $p \mid R[i]$.
- Then we want this graph to be connected.
- To connect two components S and T, we'll multiply a member of either S or T by some $k > 1$.
  - It doesn't matter which member; they'll get connected regardless.

Hackerland

- **Insight 1:** We may multiply only by primes.
- *Proof:*
  - If we multiply by a non prime k, we can separately multiply by the prime factors of k individually.
  - This is cheaper because $x + y \leq xy$ if $x, y > 1$.

Hackerland

- **Insight 2:** We won't multiply by a prime ≥ 5.
  - Intuitively, it's cheaper to use smaller primes.
- *Proof:*
  - If we multiply by $p \geq 5$ a total of $t \geq 1$ times, then presumably that's to connect t components to the component already containing p.
    - Otherwise, just multiply them all by 2 or something.
  - But, just multiply all $t + 1$ components by 2 to connect them!
  - The cost is $2(t + 1)$ which is lower than pt, because
    - $pt - 2(t + 1) \geq 5t - 2t - 2 \geq 3 - 2 > 0$.

Hackerland

- The only remaining primes are 2 and 3.
- **Insight 3:** We won't multiply by 2 and 3 at the same time.
- *Proof:*
  - If we multiply by a 2, then after that, surely some component will have a 2.
  - But then, we wouldn't want to multiply by anything else anymore; the cheapest way to connect everything else to it is to multiply by 2.

Hackerland

- Another insight that can be used in some solutions:
- **Insight 4:** We won't multiply by 3 twice.
- *Proof:*
  - If we multiply by 3 a total of $t \geq 2$ times, then presumably that's to connect t components to the component already containing 3.
  - But then, just multiply all $t + 1$ components by 2!
  - The cost is $2(t + 1)$, which is not worse than $3t$ because
    - $3t - 2(t + 1) = t - 2 \geq 0$.

Hackerland

- **Solution:** $\min(f(2), f(3))$
  - where $f(p)$ is the cost assuming we only multiply by p.
  - $f(p) = p \cdot (c - a(p))$ where
    - c = number of components
    - $a(p) = 1$ if p is already present in some component, else 0.
- Everything can be done in $\mathbf{O(N + pf_{total})}$
  - where pf_{total} is the total number of prime factors of all $R[i]$.
- Assumes a sieve has been done at the start to factorize all 1 to V.
  - $V \sim 10^6$
  - It takes $\mathbf{O(V \log \log V)}$.

Problem 4

Peace, Freedom, Justice and Security

No. of accepted solutions: 137

Author: Kevin Charles Atienza

Peace, Freedom, Justice and Security

- If $d(P, B) + d(B, T) < d(T, P)$ (or permutations), impossible
  - because $d(T, P)$ is meant to be the shortest path, so a detour through B should not yield a better cost.
- If $d(P, B) + d(B, T) + d(T, P)$ is odd, then also impossible
  - because walking along the grid in cardinal directions and returning to the same cell always takes even steps.
- Otherwise, possible.

Peace, Freedom, Justice and Security

- Translation doesn’t change area.
  - So, assume one of the points is $(0, 0)$.
  - Technically invalid coordinates, but we can just translate again to Quadrant I at the end.
  - In fact, for convenience, let’s also allow negative coordinates.

Peace, Freedom, Justice and Security

- *Insight:* Two of the points must be aligned: horizontally or vertically.
- Since rotation doesn't affect area, we can assume it's horizontal.
- We can also assume one of the aligned points is $(0, 0)$, so there's a point $(x, 0)$.
- We can even assume $x > 0$ by flipping horizontally if needed.
- x must be one of $d(P, B)$, $d(B, T)$ or $d(T, P)$.

Peace, Freedom, Justice and Security

- *Proof:*
  - Suppose not.
  - Rotate the three points so that there is a leftmost-bottommost point.
    - Always possible because there are only 3 points.
  - Let (x, y) be the leftmost-bottommost point.
  - The (signed) area will be $\alpha x + \beta y + \gamma$ where α, β and γ are dependent on the two other points.
    - via the shoelace formula

Peace, Freedom, Justice and Security

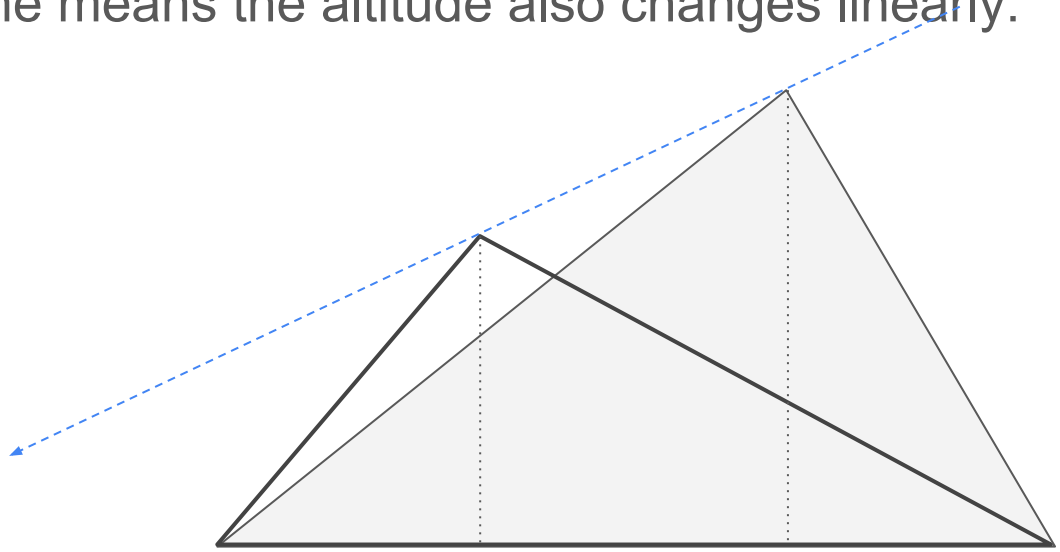
- *Proof (continued):*
  - The signed area is $\alpha x + \beta y + \gamma$.
  - However, $(x + 1, y - 1)$ and $(x - 1, y + 1)$ can also pair up with the two other points
    - because (x, y) is not aligned vertically or horizontally.
  - Using those points instead of (x, y), the area increases or decreases by $\alpha - \beta$.
  - Thus, one of the directions $\langle +1,-1 \rangle$ or $\langle -1,+1 \rangle$ must increase the area! (or make it stay the same)

Peace, Freedom, Justice and Security

- *Proof (continued):*
  - One of the directions must increase the area or make it stay the same.
  - Choose such a direction and walk until (x, y) aligns with one of the other two points.
  - The area will be at least as good as the original.
  - Therefore, there is a solution where two points are aligned.

Peace, Freedom, Justice and Security

- Graphically, the previous proof says that if you fix two points, one of the sides of the triangle stays fixed, but moving the third point along some line means the altitude also changes linearly:



Peace, Freedom, Justice and Security

- The third point is then easy to compute given $(0, 0)$ and $(x, 0)$ (and the given distances)
- In most cases, there are only two possibilities for the third point.
  - If there are more, choose the one with the highest y-coordinate.
- It can now be solved in $\mathbf{O(1)}$.
- *Important:* Print the points in the correct order!

Problem 5

Lord of the Group

No. of accepted solutions: 81

Author: Kevin Charles Atienza

Lord of the Group

- For simplicity, let's include subsequences of size 1 and 0.
  - Just subtract $n + 1$ at the end.
- First, we can't have two numbers $> d$.
  - Their sum will be too large:
    - If $x \geq y > d$ are the two largest, then $(x + y) - d > x$,
    - so $x + y$ is too far away from any element of the array.
- Similarly, we can't have two numbers $< -d$.
- Also, if $[x_1, \ldots, x_k]$ is d-admissible, then so is $[c, x_1, \ldots, x_k]$ if $|c| \leq d$
  - because $x_i + c$ is close enough to x_i itself.

Lord of the Group

- Therefore, the array consists of
  - maybe one value > d
  - maybe one value < $-d$
  - A bunch of numbers in $[-d, d]$.
- So there are four cases.
  - **Case 1:** We take no numbers > d and < $-d$.
  - **Case 2:** We take one number > d and no number < $-d$.
  - **Case 3:** We take no number > d and one number < $-d$.
  - **Case 4:** We take one number > d and < $-d$.
- The first three cases should be easy to count.

Lord of the Group

- Let
  - G = number of numbers > d
  - L = number of numbers < -d
  - M = number of numbers in $[-d, d]$.
- Then
  - For Case 1, there are 2^M subsequences.
  - For Case 2, there are $2^M \cdot G$.
  - For Case 3, there are $2^M \cdot L$.
- The only tricky case is Case 4.

Lord of the Group

- Let's say the numbers we take are $x < -d$ and $y > d$.
- We can only do so if $s := x + y$ is close to some number in $[-d, d]$.
- Thus, it can only be between $-2d$ and $2d$.
- Also, we need to take some number in $[s - d, s + d]$
  - so $x + y$ can be close to it.
- So with sum s, we want to find how many collections of numbers in $[-d, d]$ there are such that at least one is also in $[s - d, s + d]$.
- Precompute this for every s from $-2d$ to $2d$. Let's call this $c[s]$.

Lord of the Group

- So for Case 4, the answer is $\Sigma_{-2d \le s \le 2d} c[s] \cdot p[s]$ where $p[s]$ is the number of pairs $x < -d$, $y > d$, and $x + y = s$.
- We can also compute $p[s]$ easily.
  - For a given x, the number y can only be $s - x$.
  - Therefore, $p[s]$ is just $\Sigma_{x < -d,\ s - x > d}\ \mathrm{count}(s - x)$.
- The whole answer can be computed in $\mathbf{O(d^2 + nd)}$.
  - This can definitely be improved!

Problem 6

DFS Sequences

No. of accepted solutions: 16

Author: Ajinkya Parab

DFS Sequences

- The DFS is the root followed by the DFSes of the subtrees.
- Thus, the subtrees appear in the DFS as consecutive subarrays.
- Naive solution:
  - $\mathrm{tree}(i, j, k)$ = number of trees with DFSes $A[i..i+k)$ and $B[j..j+k)$.
    - $X[l..r)$ includes $X[l]$ but excludes $X[r]$.
- The answer is then $\mathrm{tree}(1, 1, n)$.
- However, the recurrence may be a bit tricky.
  - If we select some subarray of $A[i..i+k)$ as a “subtree”, then the remaining elements may be disconnected in $B[j..j+k)$!

DFS Sequences

- *Insight:* That's fine!
  - The recurrence will take care of it.
  - We simply have to ensure we only "call" a subproblem if it is valid.
- Let
  - $\mathrm{tree}[\mathrm{i}, \mathrm{j})$ = number of trees with DFS $\mathrm{A}[\mathrm{i..j})$ and having a DFS that is a *subsequence* of B.
- The answer is still $\mathrm{tree}[1, \mathrm{n})$.

DFS Sequences

- For a more convenient DP, use the following alternative:
  - $\text{tree}[i, j)$ = number of trees with DFS $A[i..j)$ and having a DFS that is a *contiguous subarray* of B.
  - $\text{forest}[i, j)$ = number of *forests* with DFS $A[i..j)$ and such that all its subtrees have DFSes that are *contiguous subarrays* in B.
- **Note:** $\text{tree}[i, j)$ now only considers contiguous subarrays.
  - This is fine because this must be true for all subtrees of the original tree.

DFS Sequences

- The recurrences are now straightforward.
- For $\mathrm{tree}[i, j)$:
  - $\mathrm{tree}[i, j) = 0$ if $A[i..j)$ is not contiguous in B.
  - $\mathrm{tree}[i, j) = 0$ if $A[i]$ appears after some $A[k]$ $(i < k < j)$ in B.
  - $\mathrm{tree}[i, j) = \mathrm{forest}[i + 1, j)$ otherwise.
- For $\mathrm{forest}[i, j)$:
  - $\mathrm{forest}[i, j) = \Sigma_{i<k\leq j}\ \mathrm{tree}[i, k) \cdot \mathrm{forest}[k, j)$
  - $\mathrm{forest}[i, i) = 1$.
- The ‘k’ in the sum selects the nodes of the first tree.
- It is correct to just try all k; $\mathrm{tree}[i, k)$ just becomes 0 if it is invalid.

DFS Sequences

- Each state can be computed in $O(n)$ time.
  - Checking whether $A[i..j)$ is contiguous in B can be done by storing the location of every number in B, and just checking in linear time.
  - Checking whether $A[i]$ appears before some $A[k]$ can also be done in linear time.
  - The sum for $\text{forest}[i, j)$ can be computed in linear time.
- There are also $O(n^2)$ states.
- Therefore, the running time is $\mathbf{O(n^3)}$.

Problem 7

Blizzard Blitz

No. of accepted solutions: 12

Author: Kevin Charles Atienza

Blizzard Blitz

- *Insight:* from every location, there are *always* 4 available directions.
- After the first step, only 3 are allowed, because one of them is the opposite of the previous.
- Thus, there are exactly $4 \cdot 3^{n-1}$ possible paths from any location.

Blizzard Blitz

- We just need to force the path to go through (i, j) at step k.
- For a given direction sequence (among the $4 \cdot 3^{n-1}$), it's usually possible to reverse-engineer the starting location so it may visit (i, j) at step k.
  - *Idea:* start at (i, j) and "unwalk the first k steps".
- Thus, there's usually only one starting point for every direction sequence.
- There are just some edge cases we need to handle.

Blizzard Blitz

- If cell (i, j) is **ice**, then you can only slide through it, either vertically or horizontally. However:
  - vertical is only possible if there is a snow cell in column j,
  - horizontal is only possible if there is a snow cell in row i.
- Other than that, there are no more restrictions.
- There are 2 ways to go vertical, and 2 ways to go horizontal.
- And 3^{n-1} ways to choose the remaining directions.
- So the answer is $3^{n-1} \cdot ((\text{vertical} \ ? \ 2 : 0) + (\text{horizontal} \ ? \ 2 : 0))$.

Blizzard Blitz

- If cell (i, j) is **snow**, then for every direction sequence, there is exactly one path that lands at (i, j) at the *beginning* of step k.
  - This gives $4 \cdot 3^{n-1}$.
- However, there may be paths where (i, j) lands at the *end* of step k **but not** at the beginning.
  - It can happen vertically if there is *another* snow cell at column j.
  - It can happen horizontally if there is *another* snow cell at row i.
- If it happens, then there are again $2 \cdot 3^{n-1}$ possibilities.
- So the answer is $3^{n-1} \cdot (4 + (\text{vertical} \ ? \ 2 : 0) + (\text{horizontal} \ ? \ 2 : 0))$.

Problem 8

The Lag

No. of accepted solutions: 3

Author: Kevin Charles Atienza

The Lag

- Given two edges, removing them creates three connected components.
- We want to find the number of pairs (u, v) that belong to different components.
- Thus, for each node, we want to identify which of these three trees it belongs to.

The Lag

- Root the tree (arbitrarily).
- Let the edges be (x_1, y_1) and (x_2, y_2) where x_i is the parent of y_i.
- Then the tree containing node i is uniquely determined by two things:
  - whether i is a descendant of y_1, and
  - whether i is a descendant of y_2.
- Thus, for each node, we associate with it these *two bits* above, and call them the node's **profile**, and we want to find the number of (u, v) such that u and v have different profiles.
- You can also handle things case by case, but the above handles them in a more uniform way.

The Lag

- The tree containing node i is uniquely determined by two things:
  - whether i is a descendant of y_1, and
  - whether i is a descendant of y_2.
- Thus, we want a way to write "j is a descendant of i" in a *data-structure-friendly* way.

The Lag

- *Insight:* Perform a DFS, and store $\text{first}[i]$ and $\text{last}[i]$ visitation times for each node i. Then j is a descendant of i iff $\text{first}[i] \leq \text{first}[j] \leq \text{last}[i]$.
  - This is *data-structure-friendly*!
- Conversely, j is not a descendant of i iff either
  - $\text{first}[j] < \text{first}[i]$ or
  - $\text{first}[j] > \text{last}[i]$.
- Therefore, we're now able to express all our conditions in terms of inequalities!

The Lag

- For example, the number of pairs (u, v) such that
  - u is a descendant of y_1,
  - u is a descendant of y_2,
  - v is a descendant of y_1,
  - v is not a descendant of y_2,
- is equal to the number of pairs satisfying
  - $first[y_1] \leq first[u] \leq last[y_1]$,
  - $first[y_2] \leq first[u] \leq last[y_2]$,
  - $first[y_1] \leq first[v] \leq last[y_1]$,
  - $first[v] < first[y_2]$ or $first[v] > last[y_2]$.

The Lag

- Thus, the number of pairs with different *profiles* can be expressed as the number of pairs satisfying some sets of inequalities.
- This can be expressed geometrically as well:
  - Identify the pair (u, v) with the point $(\text{first}[u], \text{first}[v])$.
  - Then the queries reduce to 2D *rectangular* queries, i.e., counting the number of points in some axis-aligned rectangle.
- And it's well-known that such queries can be efficiently handled by data structures, e.g., *segment trees*!

The Lag

- Our 2D range queries are *static*, meaning the points do not change.
- If we interpret one of the dimensions as the *time* dimension, that dimension turns into *updates*, and our problem turns into 1D *dynamic* range queries.
- These can be solved with Fenwick trees.
- It runs in $O(n + (p + 25m) \log n)$.
  - Because $25m \gg p$, improvements are possible by using a structure with faster queries and slower update time,
    - e.g., a “fat” segment tree.

Problem 9

A Cubical Romance

No. of accepted solutions: 1

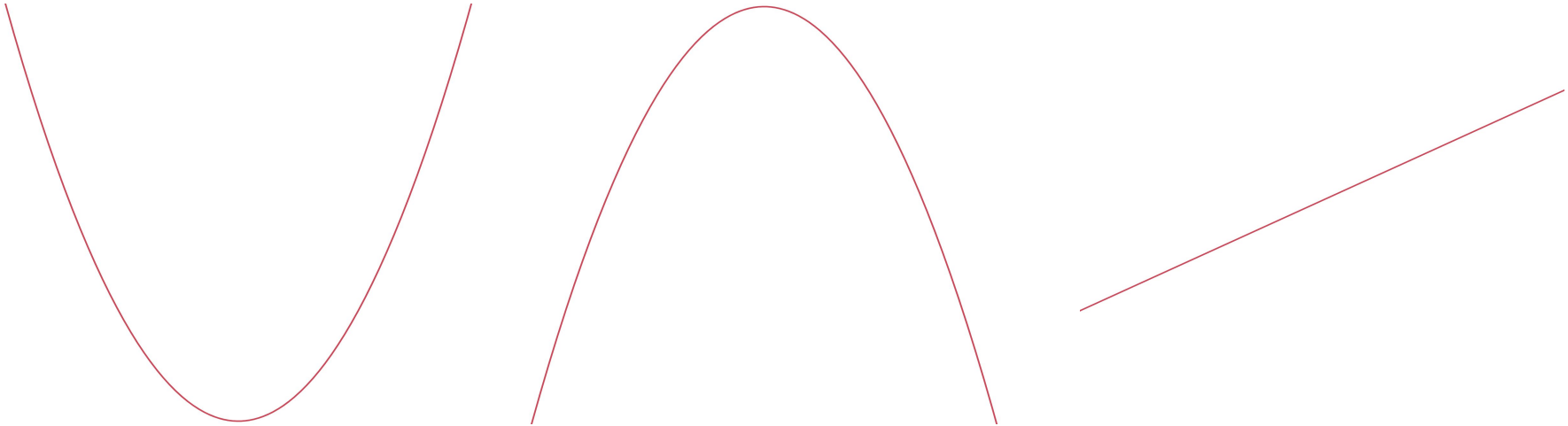
Author: Kevin Charles Atienza

A Cubical Romance

- k consecutive numbers uniquely determine a degree < k polynomial.
  - because, e.g., for $k = 4$,
    $L[i] = 4 \cdot L[i-1] - 6 \cdot L[i-2] + 4 \cdot L[i-3] - L[i-4]$.
- The above allows you to generate the whole array (in both directions) using just k consecutive numbers.
- So we want to find 4 consecutive numbers.

A Cubical Romance

- Try quadratics first, before cubics. We want 3 consecutive numbers.
- Classify degree ≤ 2 polynomials:

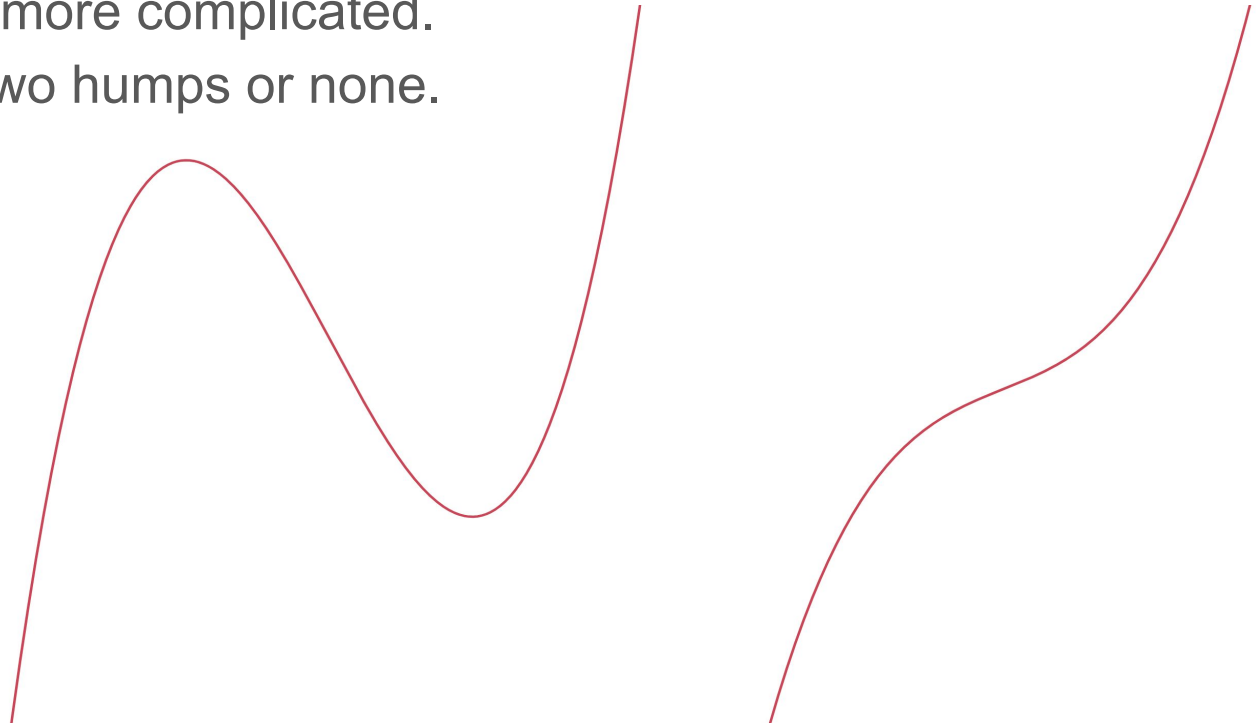


A Cubical Romance

- A degree ≤ 2 polynomial is either
  - a parabola opening up,
  - a parabola opening down,
  - a line.
- For a parabola opening up, assuming the sequence contains the vertex, the smallest three numbers must be around the vertex.
- We have found three consecutive numbers!
- Can handle other cases similarly.

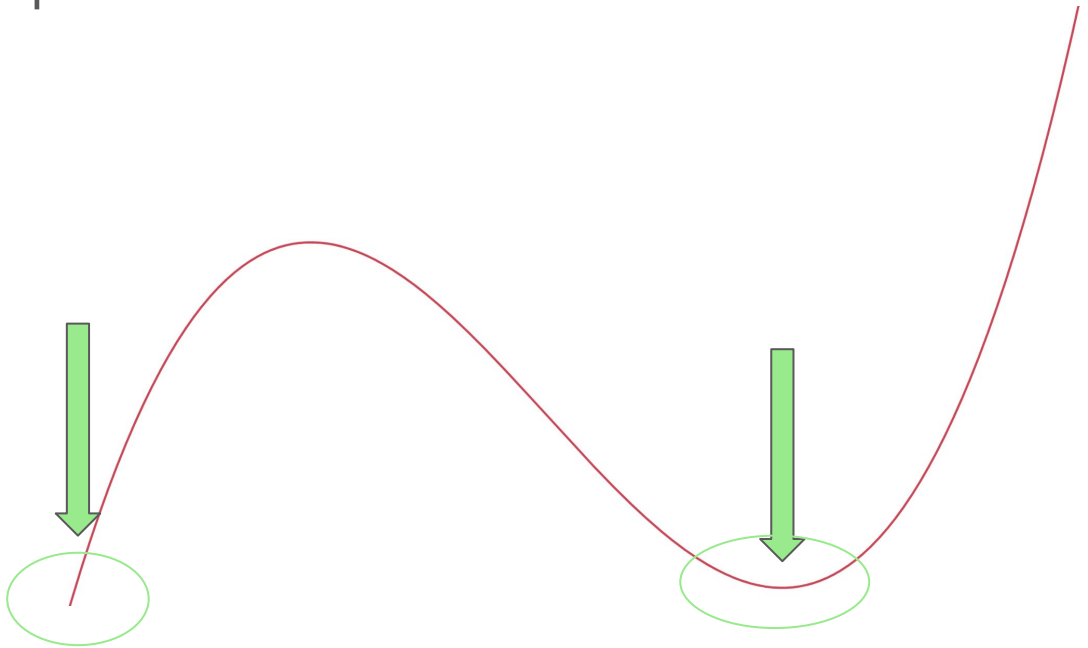
A Cubical Romance

- Cubics are more complicated.
- Can have two humps or none.



A Cubical Romance

- *Insight:* The smallest numbers in a range only pool up either at the end or around some hump.



A Cubical Romance

- There are only two locations where they may pool up.
  - So, 4 of the 7 smallest numbers must be consecutive!
- We still need to determine their order.
- Try all $4!$ permutations.
  - $4!/2$ is enough since reversals are equivalent.
- Running time is $\mathbf{O(n)}$ with a constant around $(7 \text{ choose } 4) \cdot 4!/2 = 420$.

A Cubical Romance

- Running time is $O(n)$ with a constant around 420.
- In practice, the constant is much smaller since most permutations will fail early.
- **Gotcha:** $[1\ 1\ 1\ 1\ \ldots\ 1\ 2]$ will trigger the “worst” case.
  - Fix: do it only once for every *distinct* four-tuple of numbers.

Problem 10

Dice Climbers

No. of accepted solutions: 0

Author: Vaibhav Tulsyan

Dice Climbers

- Let $E(x, y)$ be the expected number of moves if the locations are x and y, and it is x's turn to play.
- If $\max(x, y) = m$, then $E(x, y) = 0$.
- Otherwise, for each (x, y) there are 6 states it can transition to,
  - say, (x_1, y_1), …, (x_6, y_6),
  - not necessarily distinct, or even different from (x, y).
- The probability of each is $\frac{1}{6}$.
  - $E(x, y) = 1 + \frac{1}{6} \cdot \Sigma_i E(x_i, y_i)$
  - the "1" counts the first move

Dice Climbers

- There are m^2 possible states, and we have m^2 linear equations:
  - $E(x, y) = 0$ if $\max(x, y) = m$.
  - $E(x, y) = 1 + \frac{1}{6} \cdot \sum_i E(x_i, y_i)$ otherwise.
- So with Gaussian elimination or something, it can be solved in $O((m^2)^3) = O(m^6)$ time.
  - Clearly too slow.

Dice Climbers

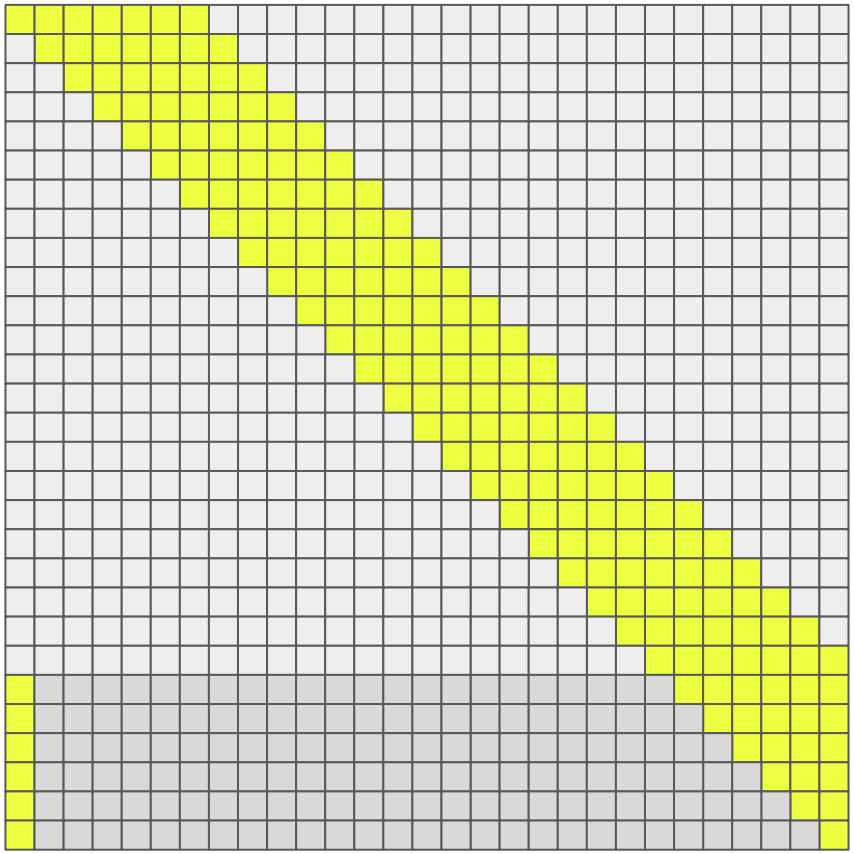
- If there are no cycles in the states, then we can simply use DP with $E(x, y) = 1 + \frac{1}{6} \cdot \Sigma_i E(x_i, y_i)$.
- However, there are cycles.
- BUT not a lot, because **$\max(x, y)$ doesn't decrease**.
- Therefore, we can compute all $E(x, y)$ by batches:
  - For each M from m to 1, compute all $E(x, y)$ for all $\max(x, y) = M$.
  - Assuming it's already computed for $> M$, some $E(x_i, y_i)$ will already be known in the equation above if $\max(x_i, y_i) > M$.

Dice Climbers

- For each M, there are $O(M)$ states, so it takes $O(m \cdot m^3) = O(m^4)$ time.
  - Still too slow.
- *Simplification:* Compute it only for $x < y$.
- If $x_i \geq y_i$ in $E(x, y) = 1 + \frac{1}{6} \cdot \Sigma_i E(x_i, y_i)$, then just expand $E(x_i, y_i)$ using its own equation, say $E(x_i, y_i) = 1 + \frac{1}{6} \cdot \Sigma_j E(x_{i,j}, y_{i,j})$.
  - We only need to expand at most one layer. (Why?)
- After that, compute $E(x, y)$ for $x \geq y$ via the equation for it with DP.
- Still $O(m^4)$ but has a smaller constant factor.

Dice Climbers

- Now, for states $x < y = M$, the corresponding matrix is actually quite *sparse*.
- Also, it has quite a regular structure: the nonzero coefficients are only in 7 diagonals and the last 6 rows.



Dice Climbers

- We exploit this structure to solve the linear system in much faster than $O(m^3)$:
  - only do elementary matrix operations on the last 6 rows. (and each row is quite “short” as well; only 7 nonzero elements)
- This solves the system in $O(md^2)$!
  - $d = 6$ is the number of sides of the die
- Therefore, the overall solution becomes $O(m^2d^2)$, which is fast enough!

Dice Climbers

- Can also be done in $\sim O(m^2 d)$ by noticing that if $d < M \leq m - d$, then the states are acyclic, so we can just do DP then.
  - i.e., we only have to solve a linear system for $M \leq d$ and $M > m - d$.
- But there's no need to do it in contest.
  - You probably shouldn't do it as well; it just wastes your precious time for no reason.

Dice Climbers

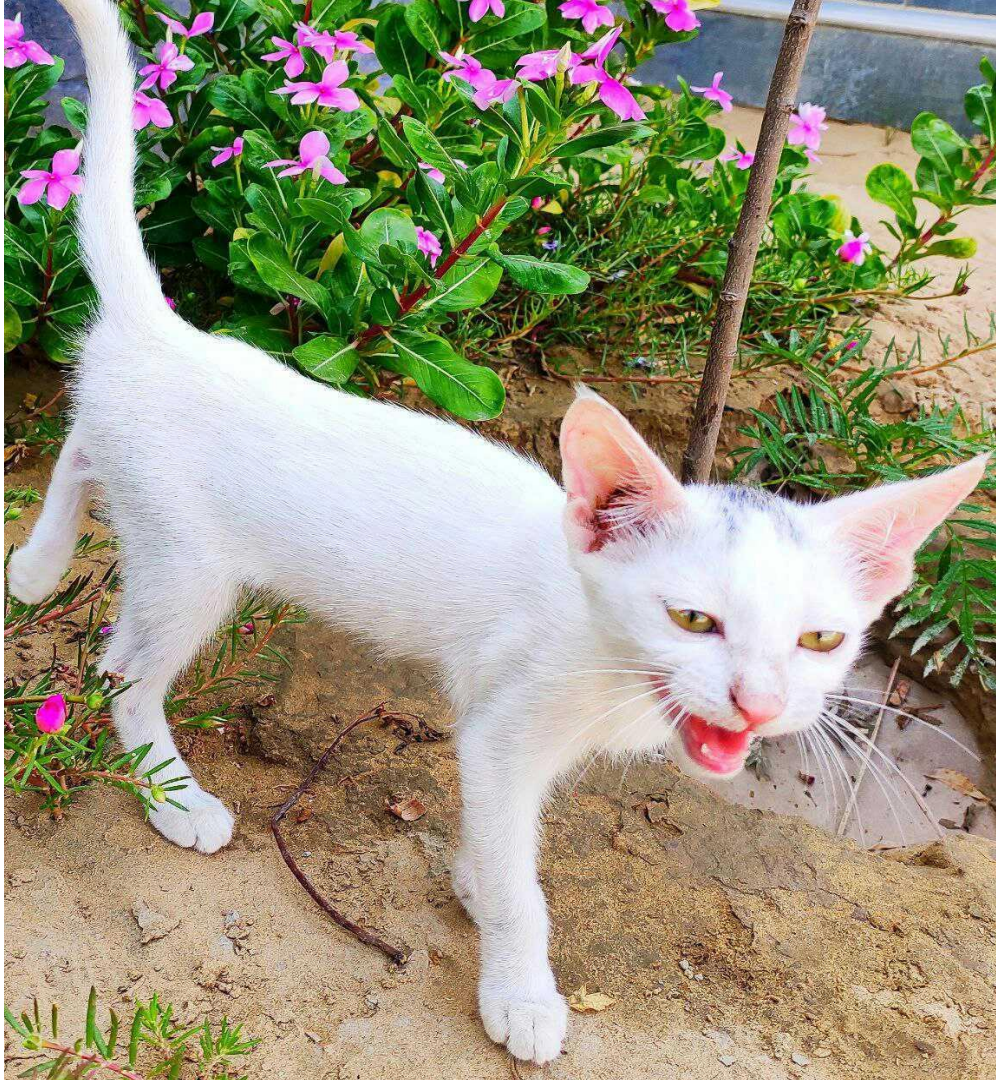
- *General lesson:*
  - In a Markov chain, you usually solve a linear system to get some properties of it (such as expected times).
  - But if it is not strongly connected, then you can analyze it separately per SCC, in some reverse topological sort order.
    - This is generally faster because SCCs may be much smaller.
  - In the most extreme case, the Markov chain is acyclic, and it simply becomes DP.
  - Also, this doesn’t just apply to Markov chains, but to all problems where there is a digraph of transitions among some states.

Problem 11

The Cat

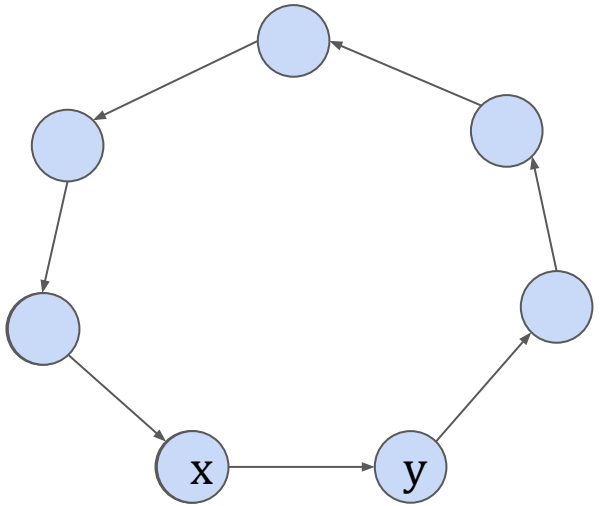
No. of accepted solutions: 0

Author: Kevin Charles Atienza



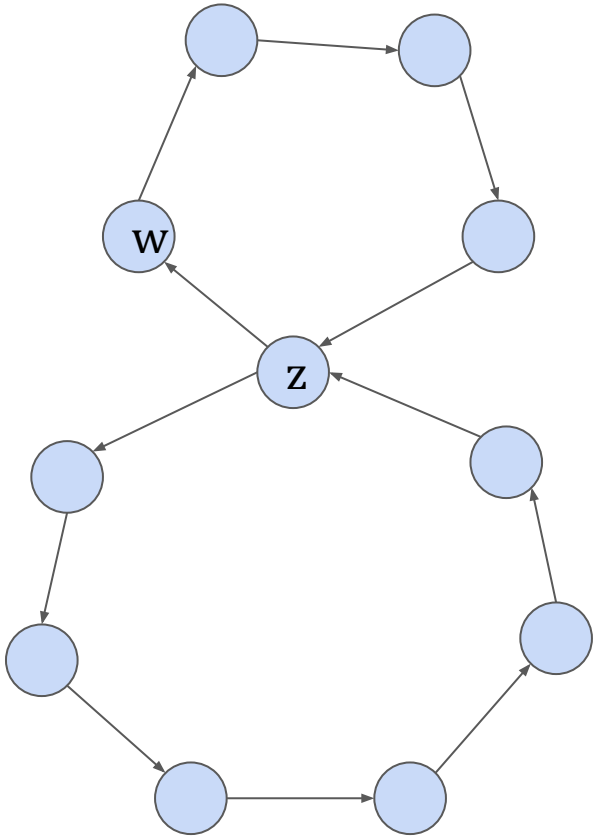
The Cat

- Notation:
  - “$x \to y$” means edge.
  - “$x \rightsquigarrow y$” means path.
- Choose an edge $x \to y$.
  - There will be a path back $y \rightsquigarrow x$, so we’ll form a (directed) cycle.



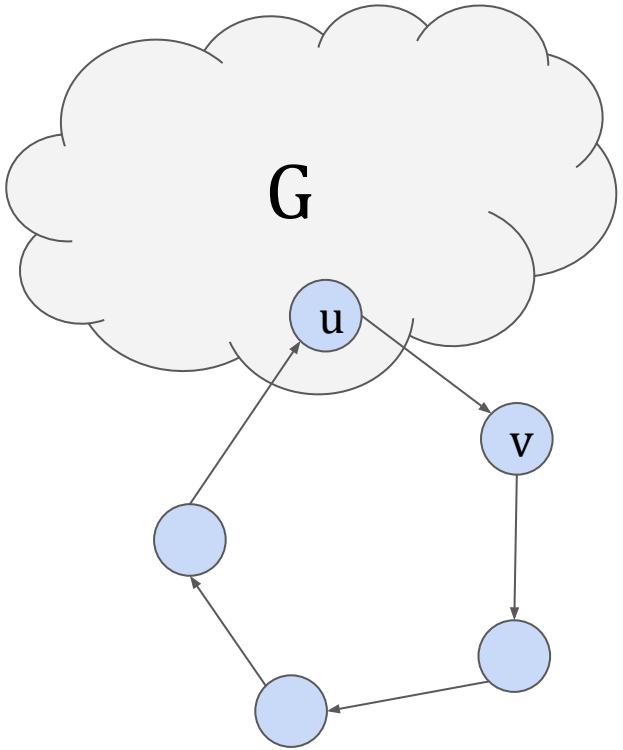
The Cat

- Choose another edge going out of the cycle, say $z \to w$.
- There will be a path back:
  - $w \rightsquigarrow z' \rightsquigarrow z$, where z' is in the original cycle.
- But if $z' \neq z$, then there will be two paths from z' to z.
- Hence, $z' = z$, and we now have *two directed cycles conjoined at a node* (z).



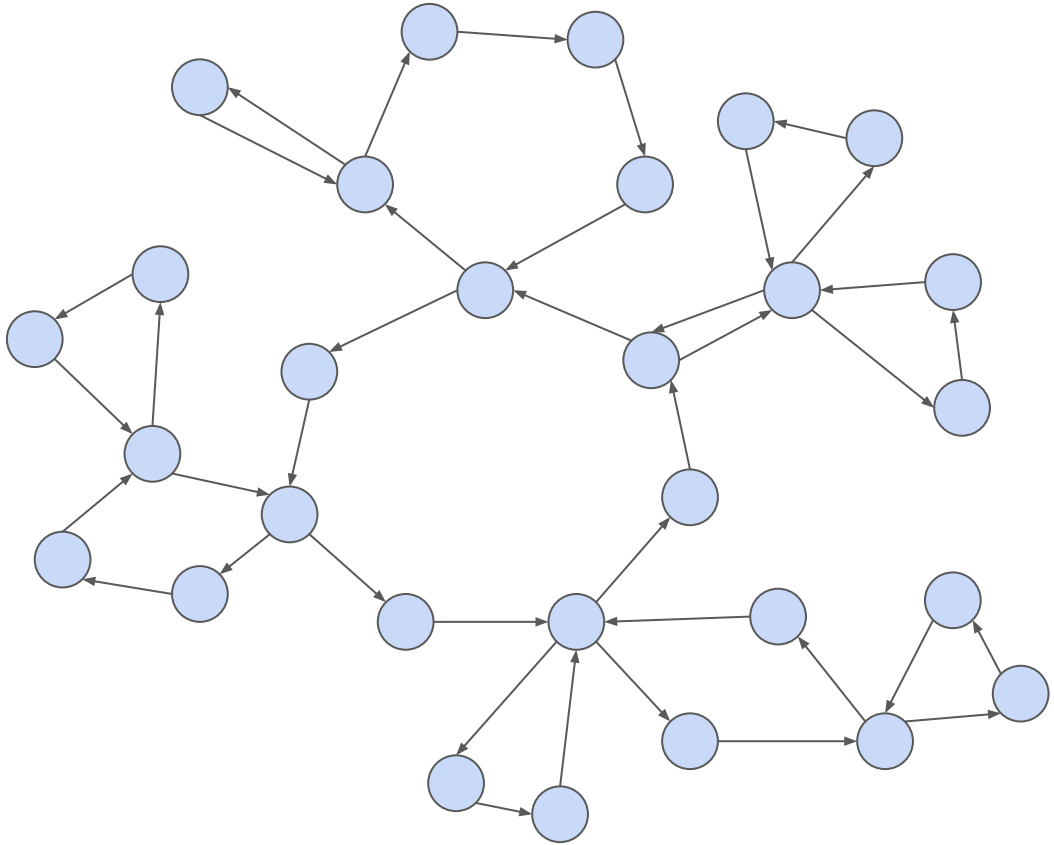
The Cat

- Repeat the argument:
  - Our current subgraph is G.
  - Choose another outgoing edge $u \to v$.
  - Then there will be another path back:
    - $v \twoheadrightarrow u' \twoheadrightarrow u$.
  - With the same argument, $u' = u$.
  - Thus, we *conjoin* another cycle (at u).
- Therefore, *a decisive graph consists of directed cycles conjoined at points, forming a tree-like structure.*



The Cat

- A decisive graph consists of directed cycles conjoined at points, forming a tree-like structure.
- It resembles a *cactus*.

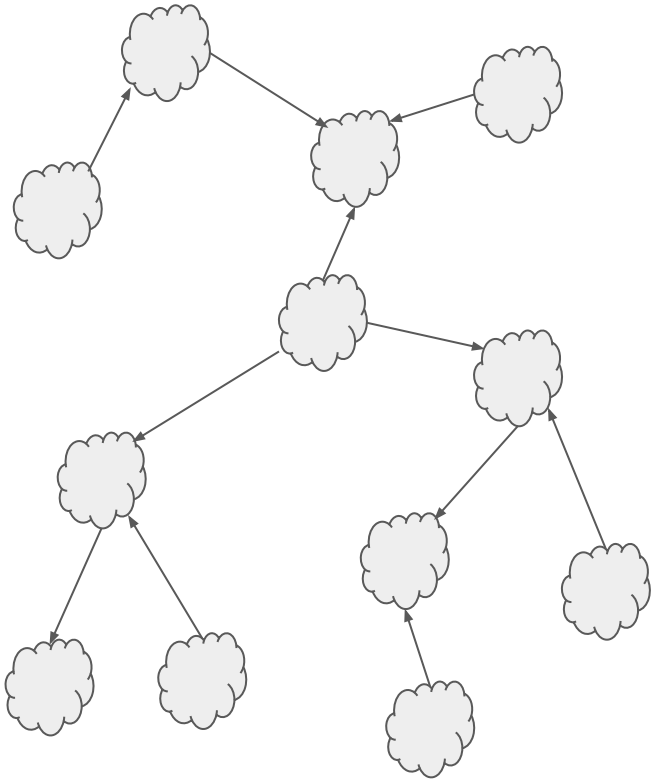


The Cat

- The SCCs of the input graph must be decisive, by the above criterion.
  - If an SCC fails it, then answer is simply 0.
- Decisiveness can be checked in linear time.
- What about the edges connecting the SCCs?
- Construct the *SCC graph* by shrinking all SCCs to a single point.

The Cat

- *Claim:*
  - If there are c SCCs, then there must be exactly $c - 1$ non-SCC edges.
  - In other words, the SCC graph must look like a “tree” (except edges are directed).



The Cat

- *Claim:* The “SCC graph” must look like a tree (with directed edges).
- *Proof:*
  - Suppose not. Then there is a “simple cycle” among nodes across multiple SCCs, say $a_1 \leftrightarrow a_2 \leftrightarrow \ldots \leftrightarrow a_k \leftrightarrow a_1$
    - where each “$\leftrightarrow$” is either $\rightarrow$ or $\leftarrow$.
  - They can’t all be $\rightarrow$ or $\leftarrow$, otherwise they’ll all belong to one SCC.
  - However, this is impossible in a decisive graph: All “simple cycles” must actually be directed cycles!

The Cat

- These $c - 1$ non-SCC edges do not necessarily form a tree, because they don’t necessarily connect the same nodes.
  - They make a tree among SCCs, but not with the original nodes.
- They form a “forest” among the original nodes though (with directed edges).
- We now need to add edges to make the whole thing decisive.

The Cat

- *Claim:* The additional edges must connect nodes in the same “tree” of the forest.
- *Proof:*
  - Left to the reader.
  - The proof uses a similar property of decisive graphs (“simple cycles” must actually be directed cycles).
- Therefore, the answer is the product of the answers across all “trees” of the forest.
- We have reduced the problem to a “**tree**”! (with directed edges)

The Cat

- The final part is finding a solution for a “tree” (with directed edges).
- I’ll leave it to you to do this final step!
- *Hints:*
  - Choices can be made per node, independently of other nodes.
  - The choices for a node is kinda like matching.
- The overall complexity is $\mathbf{O(n + e)}$.