# A Prize No One Can Win

AC before freeze: 55 / 100

First solve after 6 minutes

# A Prize No One Can Win

- From a list of numbers, how many can you select without any two summing to $> X$?
- If you can still select another item, you can always select the *cheapest* item.
- So easiest solution is: sort the list and greedily take the cheapest.

# A Prize No One Can Win

- From a list of numbers, how many can you select without any two summing to $> X$?
- If you can still select another item, you can always select the *cheapest* item.
- So easiest solution is: sort the list and greedily take the cheapest.
- A more elegant solution:
  - You know you want at least all items $\leq X/2$.
  - Whether or not you additionally want a single item $> X/2$ depends on the cheapest item $> X/2$ and the most expensive $\leq X/2$.
- You can find these three values in a single pass over the data.
- Run time: $\mathcal{O}(n \log(n))$ or $\mathcal{O}(n)$.

# A Prize No One Can Win

- From a list of numbers, how many can you select without any two summing to $> X$?
- If you can still select another item, you can always select the *cheapest* item.
- So easiest solution is: sort the list and greedily take the cheapest.
- A more elegant solution:
  - You know you want at least all items $\leq X/2$.
  - Whether or not you additionally want a single item $> X/2$ depends on the cheapest item $> X/2$ and the most expensive $\leq X/2$.
- You can find these three values in a single pass over the data.
- Run time: $\mathcal{O}(n \log(n))$ or $\mathcal{O}(n)$.
- Pitfall: you always want to select at least one item.

# Birthday Boy

AC before freeze: 41 / 120

First solve after 39 minutes

# Birthday Boy (1)

- Find the date which is the longest after any birthday on the calendar.
- Easiest algorithm: convert all dates to numbers, and look for the largest gap, then convert the number back into a date.
- Pitfall: make sure you do the tie breaks right.
- Pitfall: the longest gap might include new year ("wrap around").
- Pitfall: 01−00 is not a date.

# Birthday Boy (2)

- Pitfall: dates are terrible.

# Birthday Boy (2)

- Pitfall: dates are terrible.

```
public static int date2day(int m, int d) {
        if (m == 1)
            return d;
        if (m == 2)
            return 31+d;
        if (m == 3)
            return 31+28+d;
        if (m == 4)
            return 31+28+31+d;
        if (m == 5)
            return 31+28+31+30+d;
        if (m == 6)
            return 31+28+31+30+31+d;
        if (m == 7)
            return 31+28+31+30+31+30+d;
        if (m == 8)
            return 31+28+31+30+31+30+31+d;
        if (m == 9)
            return 31+28+31+30+31+30+31+31+d;
        if (m == 10)
            return 31+28+31+30+31+30+31+31+30+d;
        if (m == 11)
            return 31+28+31+30+31+30+31+31+30+31+d;
        if (m == 12)
            return 31+28+31+30+31+30+31+31+30+31+30+d;

        return 0;
}
```

# Financial Planning

AC before freeze: 38 / 146

First solved after 32 minutes

# Financial Planning (1)

- Which investments should you buy to be able to retire as soon as possible?
- Observe:
  - If you buy an investment, you want to buy it on day 0.
  - If you take $d$ days, you might as well buy every single investment which pays off by day $d$.
- Solution 1:
  - Binary search on the number of days.
  - For a candidate day $d$, buy every investment that pays off before $d$.
- Solution 2:
  - For each investment, compute at which day it starts paying off.
  - Sort investments by the day they start paying off.
  - Greedily add investments which pay off the soonest as long as they do not start paying off after your current retirement day.
- Run time: $\mathcal{O}(n \log(n))$.

# Financial Planning (2)

- Biggest pitfall: OVERFLOW!
- It can take up to $2 \times 10^9$ days.
- On the other hand, on day $2 \times 10^9$ you might have made $\approx 2 \times 10^{23}$ euros!

# Financial Planning (2)

- Biggest pitfall: OVERFLOW!
- It can take up to $2 \times 10^9$ days.
- On the other hand, on day $2 \times 10^9$ you might have made $\approx 2 \times 10^{23}$ euros!
- The test case which tests for this was added this morning.
- There were **13** submissions which only failed on this one test case.

# Cardboard Container

AC before freeze: 38 / 84

First solved after 14 minutes

# Cardboard Container

- Given a volume $V \leq 10^6$, find

$$\min\{2(ab + bc + ca) | a, b, c \in \mathbb{N},\ abc = V\}.$$

- Naive implementation: iterate over all triples $(a, b, c)$ with $1 \leq a, b, c \leq V$. Runtime $\mathcal{O}(V^3)$.

# Cardboard Container

- Given a volume $V \leq 10^6$, find

$$\min\{2(ab + bc + ca) | a, b, c \in \mathbb{N}, \; abc = V\}.$$

- Naive implementation: iterate over all triples $(a, b, c)$ with $1 \leq a, b, c \leq V$. Runtime $\mathcal{O}(V^3)$.
- Faster: iterate over all pairs $(a, b)$ with $1 \leq a, b \leq V$, and check that $c = V/ab$ is an integer. Runtime $\mathcal{O}(V^2)$.

# Cardboard Container



- Given a volume $V \leq 10^6$, find

$$\min\{2(ab+bc+ca) | a,b,c \in \mathbb{N},\ abc = V\}.$$

- Naive implementation: iterate over all triples $(a, b, c)$ with $1 \leq a, b, c \leq V$. Runtime $\mathcal{O}(V^3)$.
- Faster: iterate over all pairs $(a, b)$ with $1 \leq a, b \leq V$, and check that $c = V/ab$ is an integer. Runtime $\mathcal{O}(V^2)$.
- Two of $a$, $b$, $c$ must be $\leq \sqrt{V}$: iterate over all pairs $(a, b)$ with $1 \leq a, b \leq \sqrt{V}$. Runtime $\mathcal{O}(V)$.

# Cardboard Container

- Given a volume $V \leq 10^6$, find

$$\min\{2(ab + bc + ca) | a, b, c \in \mathbb{N}, abc = V\}.$$

- Naive implementation: iterate over all triples $(a, b, c)$ with $1 \leq a, b, c \leq V$. Runtime $\mathcal{O}(V^3)$.
- Faster: iterate over all pairs $(a, b)$ with $1 \leq a, b \leq V$, and check that $c = V/ab$ is an integer. Runtime $\mathcal{O}(V^2)$.
- Two of $a$, $b$, $c$ must be $\leq \sqrt{V}$: iterate over all pairs $(a, b)$ with $1 \leq a, b \leq \sqrt{V}$. Runtime $\mathcal{O}(V)$.
- Make a list of at most 240 divisors of $V$, and try all pairs or triples.

# Game Night

AC before freeze: 30 / 47

First solve after 64 minutes

# Game Night

## Problem

Given a circular string of letters `ABC`, find the minimum number of people that must switch seats such that the teams are lined up correctly, i.e. `...AAABBCC...` or `...AAACCBB...`.

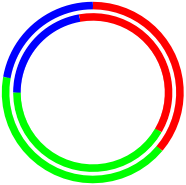
## Solution

The number of A's, B's, C's is fixed. Try all team orderings of $ABC$ or $ACB$ and every index for the first $A$.
However $\mathcal{O}(N^2)$ is too slow.

# Game Night

Use a sliding window.

1. Iterate over the index of $A$. Keep count of wrongly placed people.
2. Shifting index by one changes 3 people.



Calculating prefix sums and finding number of misplaced people for all A's, B's, C's in $\mathcal{O}(1)$ also works.
Runtime: $\mathcal{O}(N)$.

# Janitor Troubles

AC before freeze: 15 / 30

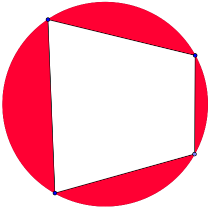
First solve after 32 minutes

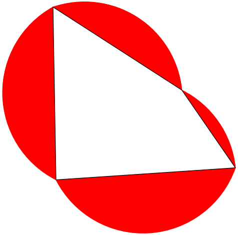
# Janitor Troubles (1)

- Given four integers, what is the maximal area of a quadrilateral with these side length?

# Janitor Troubles (1)

- Given four integers, what is the maximal area of a quadrilateral with these side length?
- Observation 1: The order of the edges doesn't matter.
- Observation 2: The area is maximal for a cyclic quadrilateral.



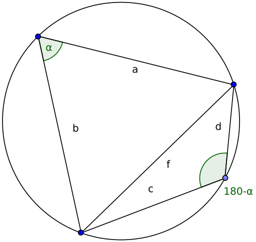


- Proof: A circle maximizes the area for a given circumference if we're allowed arbitrary shapes.
- Opposite corners sum to 180 degrees.

# Janitor Troubles (2)

- For the diagonal $f$, the cosine law gives us:

$$a^2 + b^2 - 2ab\cos(\alpha) = f = c^2 + d^2 - 2cd\cos(180 - \alpha).$$




- Solution 1: solve for $\alpha$. The total area is $(ab + cd)\sin(\alpha)/2$.
- Solution 2: solve for $f$. Use Heron's formula using sides $(a, b, f)$ and $(c, d, f)$:

$$\text{Area}(x, y, z) = \sqrt{s(s-x)(s-y)(s-z)}$$

for $s = (x + y + z)/2$.

# Janitor Troubles (3)

- Alternative 1: binary search the radius of the circle.
- Alternative 2: ternary search the length of a diagonal.

# Janitor Troubles (3)



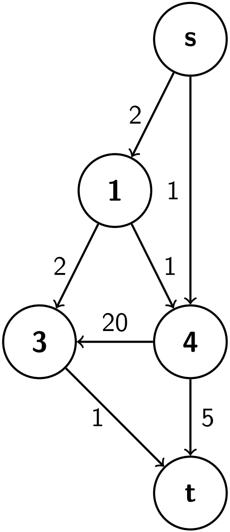
- Alternative 1: binary search the radius of the circle.
- Alternative 2: ternary search the length of a diagonal.
- Alternative 3: use Brahmagupta's formula directly:
  $s = (a + b + c + d)/2$

$$\text{Area}(a, b, c, d) = \sqrt{(s-a)(s-b)(s-c)(s-d)}.$$

# Harry the Hamster

AC before freeze: 5 / 22

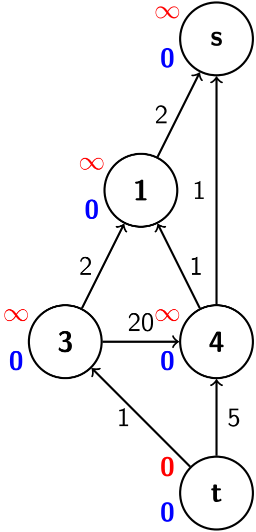
First solve after 161 minutes




Given a graph with start $s$ and end $t$, and a hamster with two brain halves: what is the shortest route from start to end (if it exists)?

Looks like a minimax, but this is probably too slow

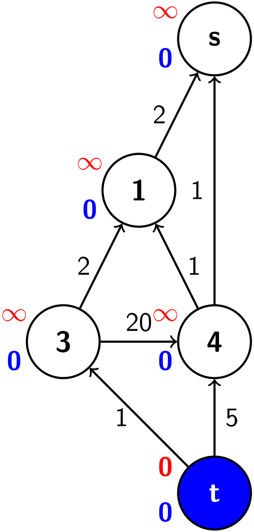
Idea: reverse the edges and do a modified Dijkstra (no game theory needed!)




Give every node two states: a red state for the path to $t$ given that the fast half chooses at this node, and a blue state for the path to $t$ given that the slow half chooses.

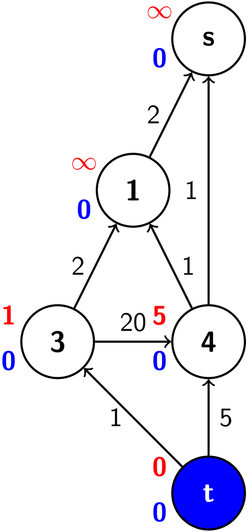
We can relax a blue node if we've relaxed all of its red parents (formerly children). If no blue nodes can be relaxed, then relax the node with the smallest value (just like Dijkstra). Relaxing means pushing your distance to all children of the other colour.

We are going to look at this process for an acyclic graph, but this also works for cyclic graphs.



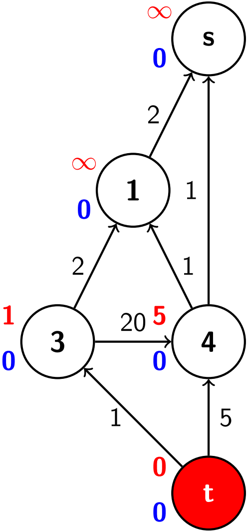

We can relax the blue node $t$ as it has no parents.

Red nodes 3 and 4 have updated distances: if the fast half gets to choose in node 4, then it is guaranteed that within 5 time units Harry will reach $t$.



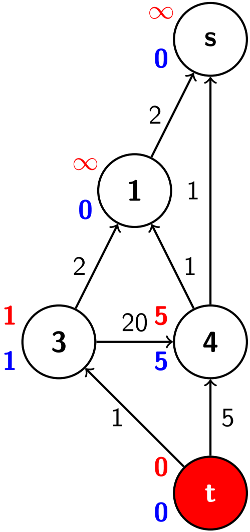

We can relax the blue node $t$ as it has no parents.

Red nodes 3 and 4 have updated distances: if the fast half gets to choose in node 4, then it is guaranteed that within 5 time units Harry will reach $t$.



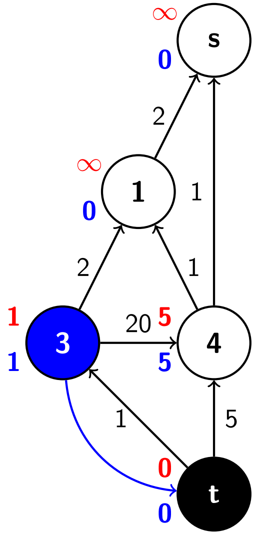

There are no more blue nodes to be relaxed, so we relax the red node $t$.

This updates the blue values of nodes 3 and 4: if the slow side gets to choose in node 4, it is guaranteed that getting to $t$ will take at least 5 time units

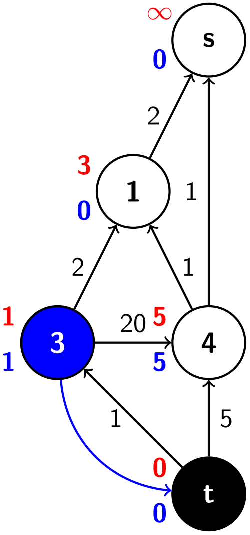



There are no more blue nodes to be relaxed, so we relax the red node $t$.

This updates the blue values of nodes 3 and 4: if the slow side gets to choose in node 4, it is guaranteed that getting to $t$ will take at least 5 time units

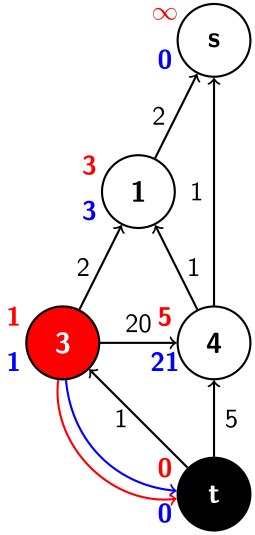



Node $t$ is now finished. Now the blue node 3 can be relaxed, and we know the choice of the slow brain half.



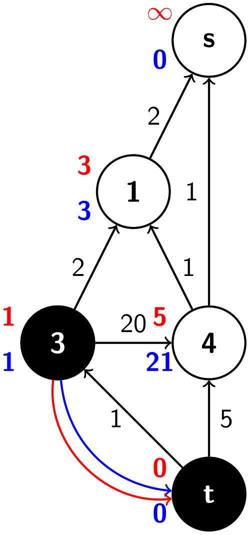

Node $t$ is now finished. Now the blue node 3 can be relaxed, and we know the choice of the slow brain half.

This updates the red value of node 1.

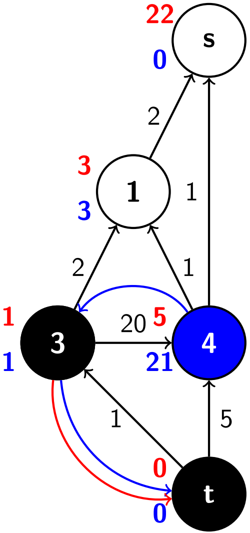



Now the red node 3 can be relaxed (it has a smaller value than the red node 4

This updates the blue node 4.

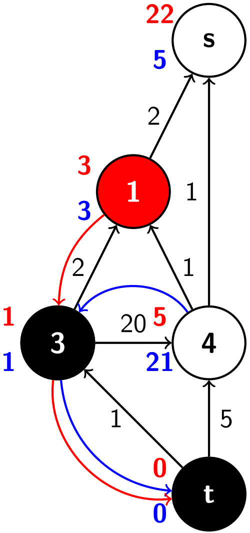



This finishes node 3 completely.

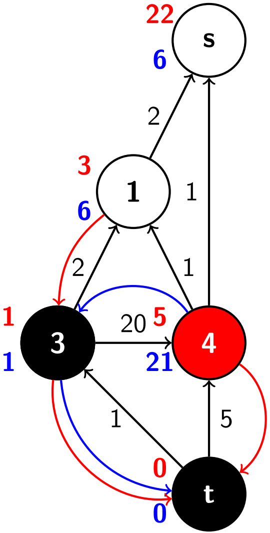



Now the blue node 4 is updated.

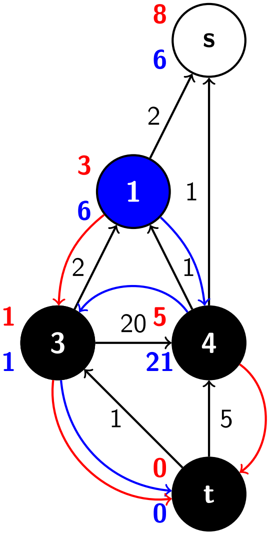
This updates the red node $s$, giving a first path of distance 22.



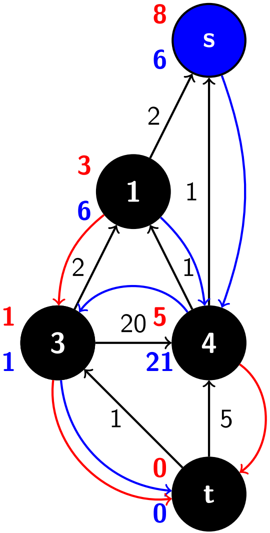

The red node 1 has distance 3, the red node 4 has distance 5, so we relax 1 first.



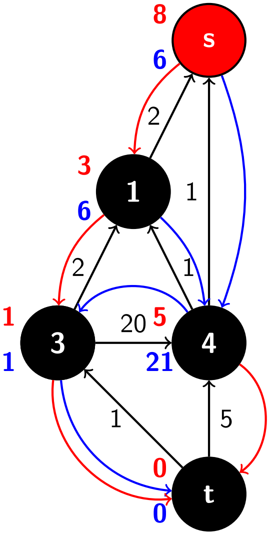

The red node 4 is now relaxed. Node 4 is completely finished.



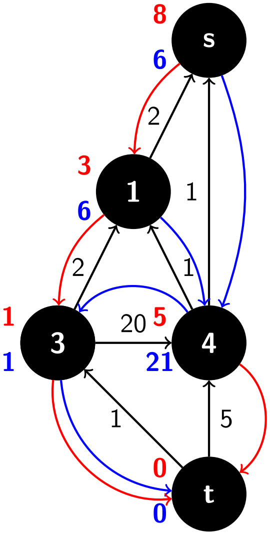

The blue node 1 is now relaxed. Node 1 is completely finished.




We finish by relaxing the nodes of $s$, finding a distance of 8 and completing the full decision graph.




We finish by relaxing the nodes of $s$, finding a distance of 8 and completing the full decision graph.




We finish by relaxing the nodes of $s$, finding a distance of 8 and completing the full decision graph.

# Kingpin Escape

AC before freeze: 2 / 19

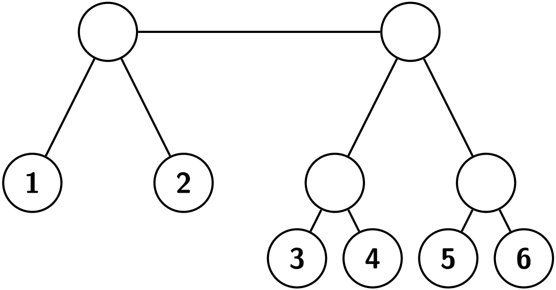
First solve after 179 minutes

# Kingpin Escape

- Given a tree $T$, add a minimal number of edges to make it biconnected.

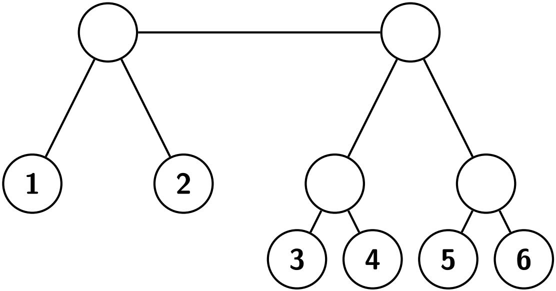
# Kingpin Escape

- Given a tree $T$, add a minimal number of edges to make it biconnected.
- We need to add an edge to every leaf, so $ans \geq \lceil \ell/2 \rceil$.
- WA: Matching $(3, 5)$, $(4, 6)$ won't work:



# Kingpin Escape

- Given a tree $T$, add a minimal number of edges to make it biconnected.
- We need to add an edge to every leaf, so $ans \geq \lceil \ell/2 \rceil$.
- WA: Matching $(3, 5)$, $(4, 6)$ won't work:




- Intuition: match far away leaves.

# Kingpin Escape

## Solution

- Do a DFS and number the leaves in order $v_1, v_2, \ldots, v_\ell$.
- Match $v_i$ with $v_{i+\lfloor \ell/2 \rfloor}$ and the optional leftover $v_\ell$ with any other vertex.
- Hence $ans = \lceil \ell/2 \rceil$.

# Kingpin Escape

## Solution

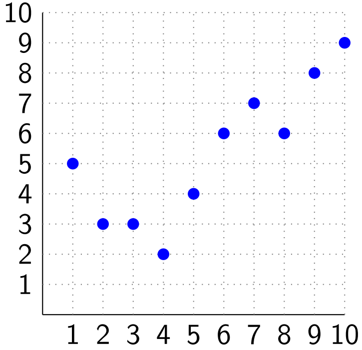
- Do a DFS and number the leaves in order $v_1, v_2, \ldots, v_\ell$.
- Match $v_i$ with $v_{i+\lfloor \ell/2 \rfloor}$ and the optional leftover $v_\ell$ with any other vertex.
- Hence $ans = \lceil \ell/2 \rceil$.
- Proof: for every edge $e \in T$, the numbers on each side form intervals $L$ and $R$ (modulo $\ell$).
  Observation: $(L + \lfloor \ell/2 \rfloor) \cap R \neq \emptyset$ □

# Entirely Unsorted Sequences

AC before freeze: 0 / 0

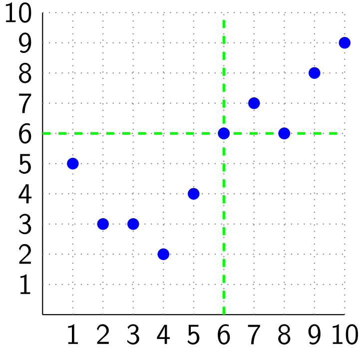
# Entirely Unsorted Sequences (1)

In a sequence of numbers, an element is *sorted* if there is no lower number after and no higher number before.



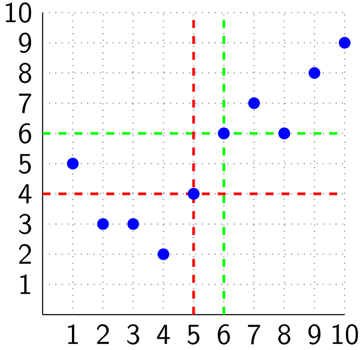
# Entirely Unsorted Sequences (1)

In a sequence of numbers, an element is *sorted* if there is no lower number after and no higher number before.



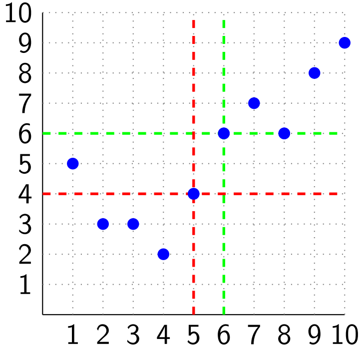
# Entirely Unsorted Sequences (1)

In a sequence of numbers, an element is *sorted* if there is no lower number after and no higher number before.



# Entirely Unsorted Sequences (1)

In a sequence of numbers, an element is *sorted* if there is no lower number after and no higher number before.




Given some integers, how many sequences without sorted elements can you make?

# Entirely Unsorted Sequences (2)

We solve the problem in two steps:

1. Solve the problem for all distinct elements.
2. Check what needs to change for possibly repeated elements.

(From now on, assume the sequence is sorted.)

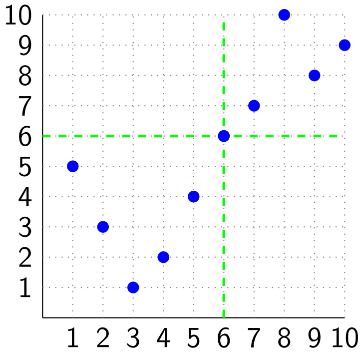
# Entirely Unsorted Sequences (3)

- If all elements are distinct, WLOG they are $1, \ldots, N$.
- Let $P[n]$ be the number of EUSs you can make with $1, \ldots, n$.
- Assume you know $P[0], \ldots, P[k-1]$, and let's compute $P[k]$.

$$\begin{aligned} P[k] &= k! - (\#\text{ of sequences with} \geq 1 \text{ sorted element}) \\ &= k! - \sum_{i=1}^{k} (\#\text{ of sequences with first sorted element at } i) \end{aligned}$$

What does a sequence with first sorted element at index $i$ look like?

# Entirely Unsorted Sequences (4)




- It has value $i$ at index $i$.
- The first $i - 1$ elements are an EUS on $1, \ldots, i - 1$.
- The last $k - i$ elements are any permutation of $i + 1, \ldots, k$.

# Entirely Unsorted Sequences (5)

$$P[k] = k! - (\#\text{ of sequences with } \geq 1 \text{ sorted element})$$
$$= k! - \sum_{i=1}^{k}(\#\text{ of sequences with first sorted element at } i)$$
$$= k! - \sum_{i=1}^{k} P[i-1] \times (k-i)!$$

You can do this computation in $O(k)$, for a total run-time of $O(N^2)$ (as long as you reduce intermediate numbers mod $10^9 + 9$).

# Entirely Unsorted Sequences (6)



- What about repeated elements? The whole analysis works, only one thing changes:

$$P[k] = k! - \sum_{i=1}^{k} P[i-1] \times (k-i)!$$

- We need to replace this with the number of permutations of $a_1, \ldots, a_k$ respectively $a_{i+1}, \ldots, a_k$.
- The number of permutations is given by a multinomial:

$$\mathrm{perm}(1,1,2,2,2,3,4,4) = \frac{8!}{2! \times 3! \times 1! \times 2!}.$$

- If we add just one number, we can compute the new multinomial in $O(1)$, which is fast enough for $O(N^2)$:

$$\mathrm{perm}(1,1,2,2,2,3,4,4,4) = \frac{8! \times 9}{2! \times 3! \times 1! \times 2! \times 3}.$$

# In case of an Invasion, please...

AC before freeze: 0 / 6

First solve after 6 minutes

# In case of an Invasion, please... (1)

### Problem

Given a road network with $n \leq 10^5$ locations with people, and $s \leq 10$ shelters, find the fastest way to move everyone to a shelter.

# In case of an Invasion, please... (2)

## Solution

- If we can move everyone to a shelter in $t$ time, we can also move everyone to a shelter in $t + 1$ time (just let everyone stand still for one time unit).

- We are asked to find the smallest feasible $t$.

This is exactly the structure of a binary search. So let's try solving the decision problem for a fixed $t = T$ and then do a binary search.

# In case of an Invasion, please... (3)

BAPC 2018

## Solution

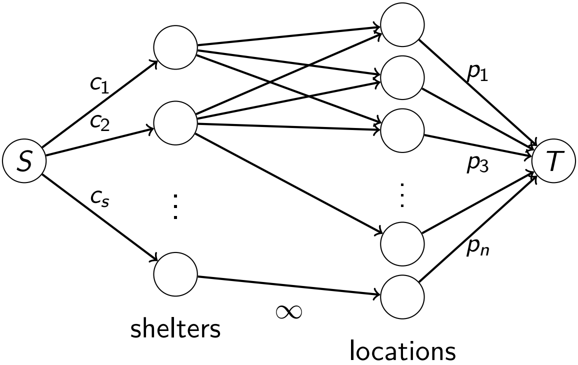
The structure of the graph is not particularly relevant, since edges have no maximal capacity anyway. We can preprocess by running Dijkstra from each shelter, in $O(s(n+m)\log(m))$.

Now for shelter $i$ and location $j$ we can send everyone in $j$ to $i$ in time $T$ if $d(i,j) \leq T$.

The result is a classical flow problem.

# In case of an Invasion, please... (4)

Let's build the corresponding flow graph:




Possible if and only if the flow is $\sum_i p_i$, i.e. every location is saturated.

# In case of an Invasion, please... (5)

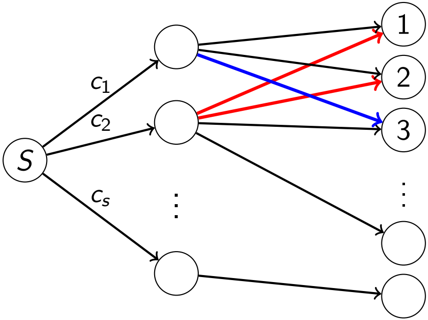
Verdict: `Time Limit Exceeded`. This graph is really large, it has $s + n + 2$ vertices, which can be a bit more than $10^5$, and up to $sn$ edges, which can be up to $10^6$.

A decent flow solution will run in $O(V^2 E)$. Runtime of flow algorithms can be misleading (usually much faster) but this is really pushing it.

Idea: exploit the structure of the graph, one side is really small $s \leq 10$.

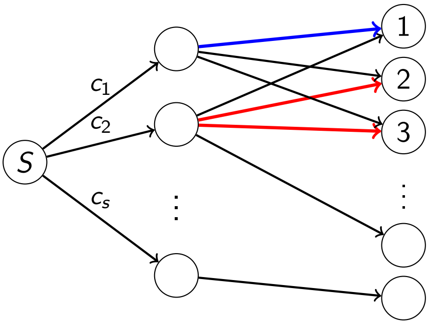
# In case of an Invasion, please... (6)

Lots of equivalent solutions. Focus on locations 1, 2, 3.



# In case of an Invasion, please... (6)

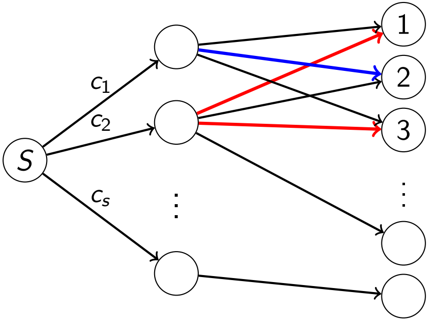
Lots of equivalent solutions. Focus on locations $1, 2, 3$.



# In case of an Invasion, please... (6)

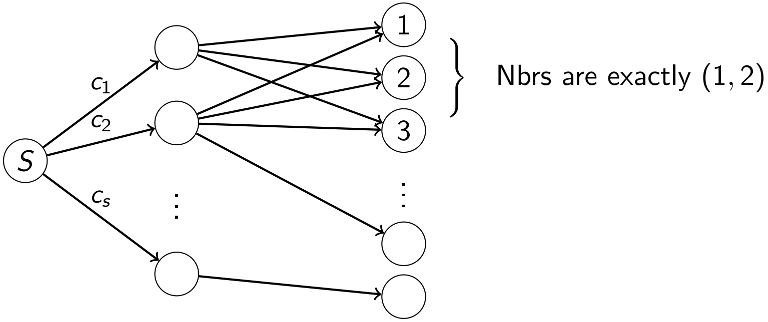
BAPC 2018

Lots of equivalent solutions. Focus on locations $1, 2, 3$.



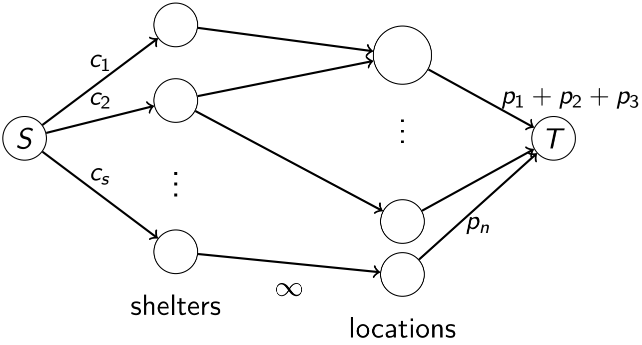
# In case of an Invasion, please... (6)

If some locations have the same neighbour set (i.e., set of reachable shelters), we can safely merge them into a single location with the sum of their populations. (note: only merge for this binary search $T$!)



# In case of an Invasion, please... (7)

Merge:




The resulting right hand side has at most $2^s$ vertices, a factor of 1000 less. Dinic will run in time.

**Homework exercise**: Solve without flow, but use Hall's marriage theorem.

# Driver Disagreement

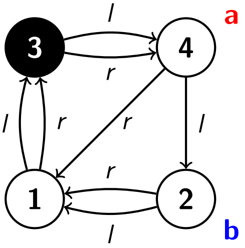
AC before freeze: 0 / 8

# Driver Disagreement (1)

## Problem

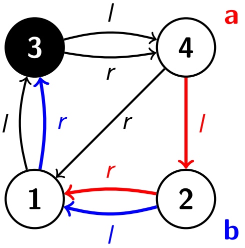
Given a finite state machine over the alphabet $\Sigma = \{l, r\}$, with each state colored white or black, and two initial states $a$ and $b$. Find the shortest word (i.e. $lrlrrlr\ldots$) that distinguishes them.

For example:




Following "$r$" moves both $a$ and $b$ to 1.

But following "$lr$":



# Driver Disagreement (2)

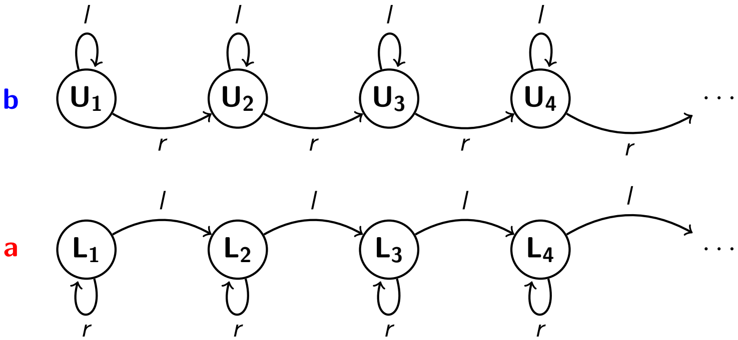
## Solution

Let's forget the shortest path for a second and try to decide whether an answer exists. The simplest solution: BFS.

```
Q ← (a, b)                              ▷ Enqueue (a, b)
S ← ∅                                   ▷ Visited states
while Q ≠ ∅ do
    (a′, b′) ← Q                        ▷ Dequeue a state
    if col(a′) ≠ col(b′) then           ▷ Found a solution?
        return True
    if (a′, b′) ∈ S then                ▷ Seen before?
        continue
    S ← S ∪ {(a′, b′)}                  ▷ Mark as visited
    Q ← (l(a′), l(b′)), (r(a′), r(b′))  ▷ Enqueue l/r turns.
return False                            ▷ Found nothing
```

# Driver Disagreement (3)

Verdict: `Time Limit Exceeded`. Consider the following automaton:




BFS will visit every state $(L_i, U_j)$ for $i, j \geq 1$, so $O(n^2)$ states asymptotically.

# Driver Disagreement (4)

## Solution

Do we have to check all states? Suppose that somehow we knew that $a$ and $b$ are indistinguishable, and so are $b$ and $c$. Should we bother checking $a$ and $c$?

# Driver Disagreement (4)

## Solution

Do we have to check all states? Suppose that somehow we knew that *a* and *b* are indistinguishable, and so are *b* and *c*. Should we bother checking *a* and *c*?

Suppose following some path *llrlrrlr*... from *a* ends in a white vertex, while following it from *c* ends in a black vertex. What happens when we follow this path from *b*?

# Driver Disagreement (4)

## Solution

Do we have to check all states? Suppose that somehow we knew that *a* and *b* are indistinguishable, and so are *b* and *c*. Should we bother checking *a* and *c*?

Suppose following some path *llrlrrlr*... from *a* ends in a white vertex, while following it from *c* ends in a black vertex. What happens when we follow this path from *b*?

A contradiction, so *a* and *c* must also be indistinguishable. In other words, indistinguishability is an equivalence relation[1].

[1] Reflexivity and symmetry are trivial.

# Driver Disagreement (5)

## Solution

This would reduce the number of states we have to care about, but there is a problem: even knowing that $(a, b)$ and $(b, c)$ are indistinguishable might require traversing the whole graph.

# Driver Disagreement (5)

## Solution

This would reduce the number of states we have to care about, but there is a problem: even knowing that $(a, b)$ and $(b, c)$ are indistinguishable might require traversing the whole graph.

**Key observation**: if $(a, b)$ and $(b, c)$ have passed through our BFS queue, we can just pretend they are indistinguishable and discard $(a, c)$.

Why does this work?

# Driver Disagreement (5)

## Solution

This would reduce the number of states we have to care about, but there is a problem: even knowing that $(a, b)$ and $(b, c)$ are indistinguishable might require traversing the whole graph.

**Key observation**: if $(a, b)$ and $(b, c)$ have passed through our BFS queue, we can just pretend they are indistinguishable and discard $(a, c)$.

Why does this work? Either:

- We were *right* and all of $a$, $b$ and $c$ are indistinguishable, and we correctly discarded $(a, c)$.
- We were *wrong* and $(a, c)$ are not indistinguishable. But then so are one of $(a, b)$ and $(b, c)$, and they were already expanded by the BFS, so we will still get the correct answer.

# Driver Disagreement (6)

## Solution

We can easily maintain the equivalence relation of indistinguishability using a disjoint-set datastructure.

```
Q ← (a, b)                                   ▷ Enqueue (a, b)
U ← { {1}, {2}, ..., {n} }                   ▷ Initialize UnionFind.
while Q ≠ ∅ do
    (a′, b′) ← Q                             ▷ Dequeue a state
    if col(a′) ≠ col(b′) then                ▷ Found a solution?
        return True
    if U.same(a′, b′) then                   ▷ Indistinguishable?
        continue
    U.merge(a′, b′)                          ▷ Mark a′ and b′ indist.
    Q ← (l(a′), l(b′)), (r(a′), r(b′))       ▷ Enqueue l/r turns.
return False                                 ▷ Found nothing
```

# Driver Disagreement (7)

## Solution

Let's analyze this algorithm:

- If we discard some $(a', b')$, then there must be some $(a', u_1), (u_1, u_2), \ldots (u_k, b')$ having already passed through the queue. If $a'$ and $b'$ are distinguishable, so is one of these pairs. So there is no need to consider $(a', b')$.
- Each time we expand a state $(a', b')$ we also merge two sets. This can happen at most $n - 1$ times, so the runtime is $O(n\,\alpha(n))$.
- We get the shortest path for free by construction (BFS).

Corollary: if the answer exists, it is at most $n - 1$.

# Some statistics

- Number of commits to the repository: `1015`.
- Total number of test cases: `467`.
- Percentage of AC jury solutions: `45.07%`.