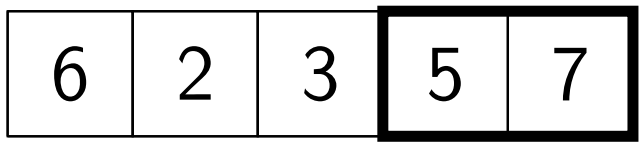
Presentation of Solutions

CTU Open 2017

Amusement Anticipation

Amusement Anticipation

- Naive solution sufficed – for each index i of the array, check whether it is a start of an arithmetic sequence $a_i, a_{i+1} \ldots, a_n$ by iterating through all of its elements
- The smallest such index is the result
- This approach runs in $\mathcal{O}(n^2)$ time



- However, last two numbers of the array are always contained in the optimal arithmetic sequence ending at index $n \Rightarrow$ the difference d of the optimal sequence is $a_n - a_{n-1}$ (for $n > 1$)
- It then suffices to check, starting from the end of the array, how many numbers correspond to the difference
- The first index i, s.t. $a_i + d \neq a_{i+1}$, is the result (or if no such index exists, the whole array is an arithmetic sequence)
- This approach yields an $\mathcal{O}(n)$ solution

Pond Cascade

Pond Cascade

- One way of solving the problem is simply to simulate it
- Another way is to binary search over time

Pond Cascade – Simulation

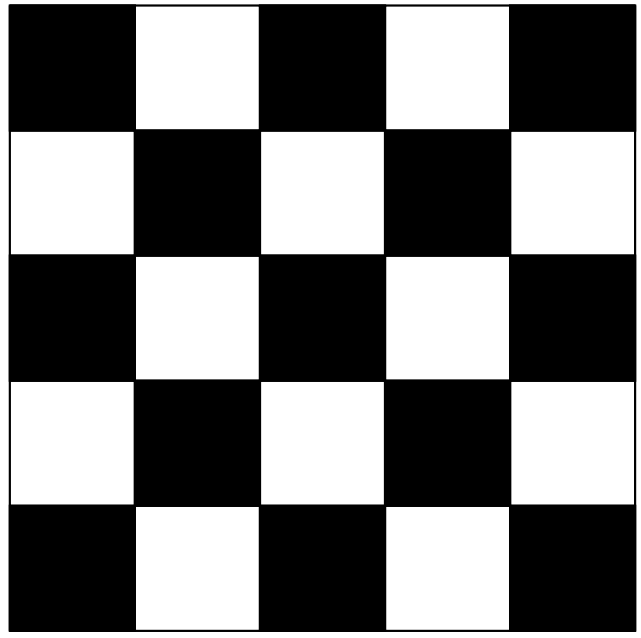
- Use priority queue to track order of events when some pond fills up to the top
- We will keep an array which will contain, for each pond, subsequent pond where the water will overflow
- While processing an event set remaining volume of current pond to zero and decrease remaining empty volume of subsequent pond and create an event based on new flow to the subsequent pond
- All that is left is to memorize the time for the last and for all ponds
- Total running time: $\mathcal{O}(n \cdot log\ n)$

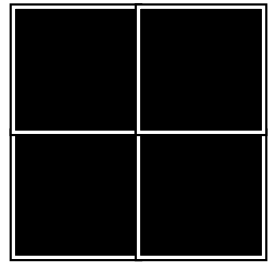
Pond Cascade – Binary search

- Use binary search to guess time when the last and all of the ponds fill up separately
- Based on the guessed time we can loop through the ponds and find out how many litres will be poured in and consequently how much liquid is poured to the subsequent in order
- Based on the result of binary search (either we got too much liquid in pond or not enough to fill up) we adjust our guess
- It can be easily proven that our guess has to be between 1 and 10^9 seconds
- Then we can simply iterate and keep trying till we get our desired precision or for fixed number of iterations (50 is more than sufficient)

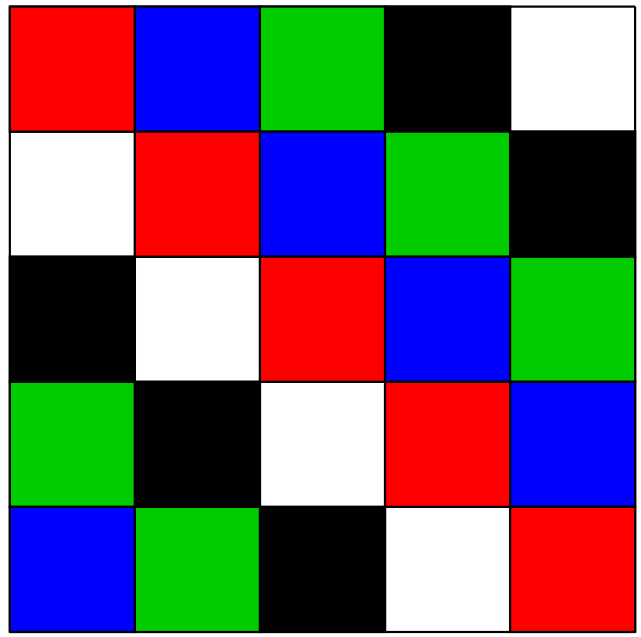
Chessboard Dancing

Chess - Knight

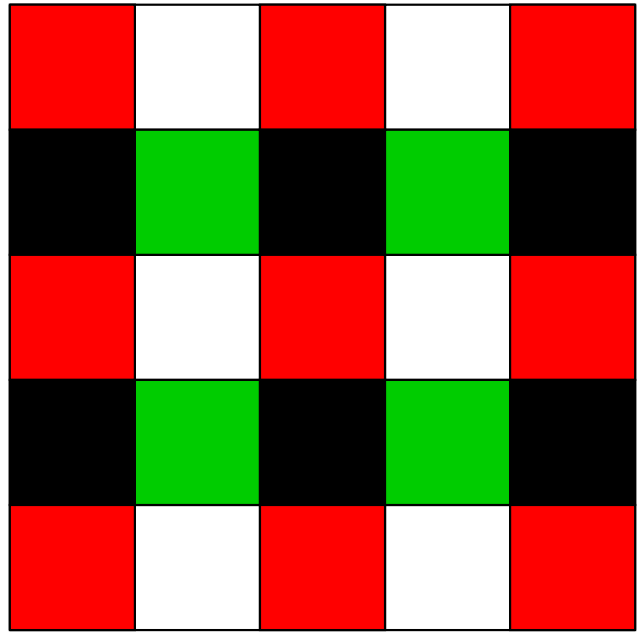


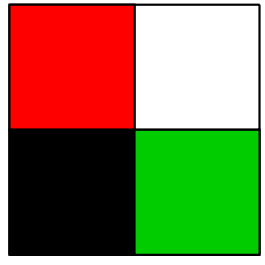


Chess - Rook

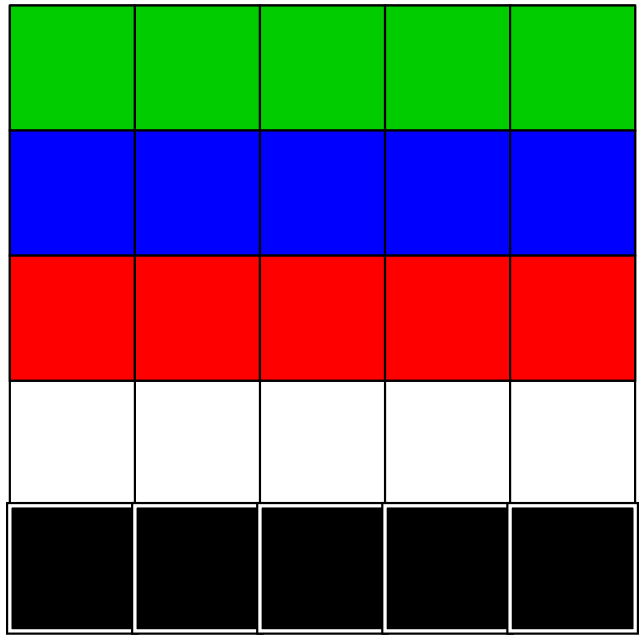


Chess - King



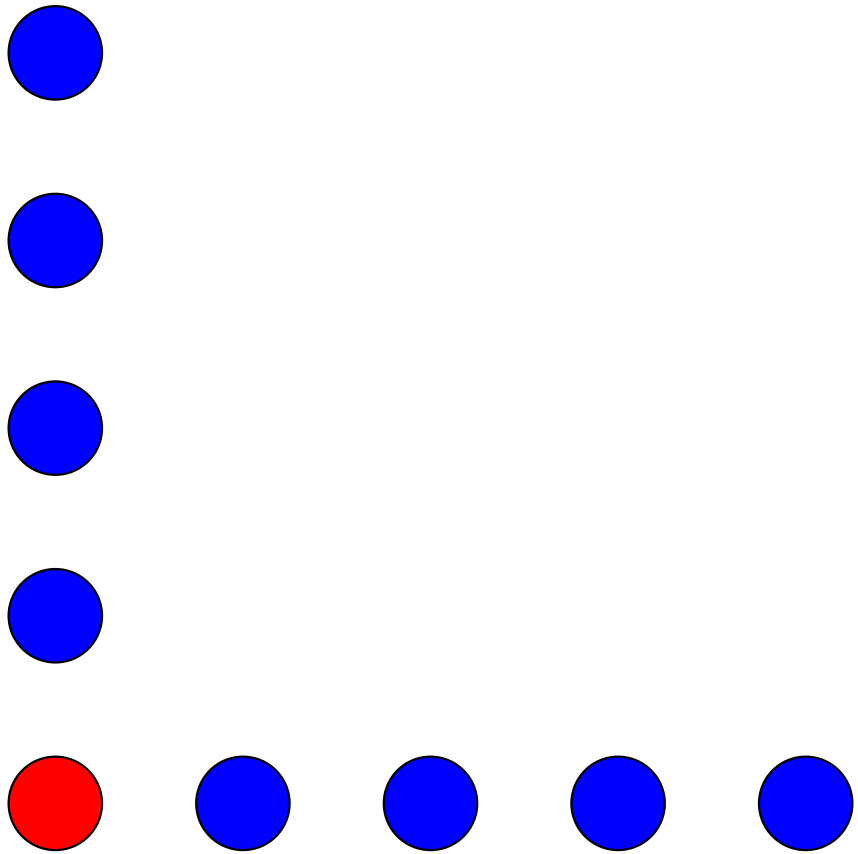


Chess - Bishop

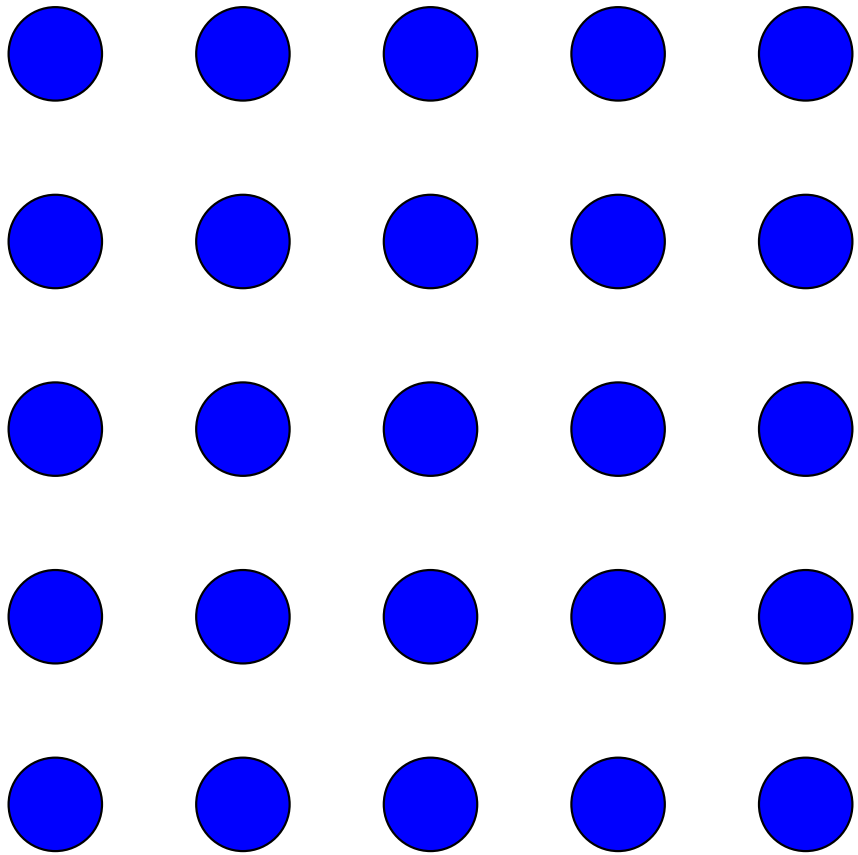


Equinox Roller Coaster

Equiroaster



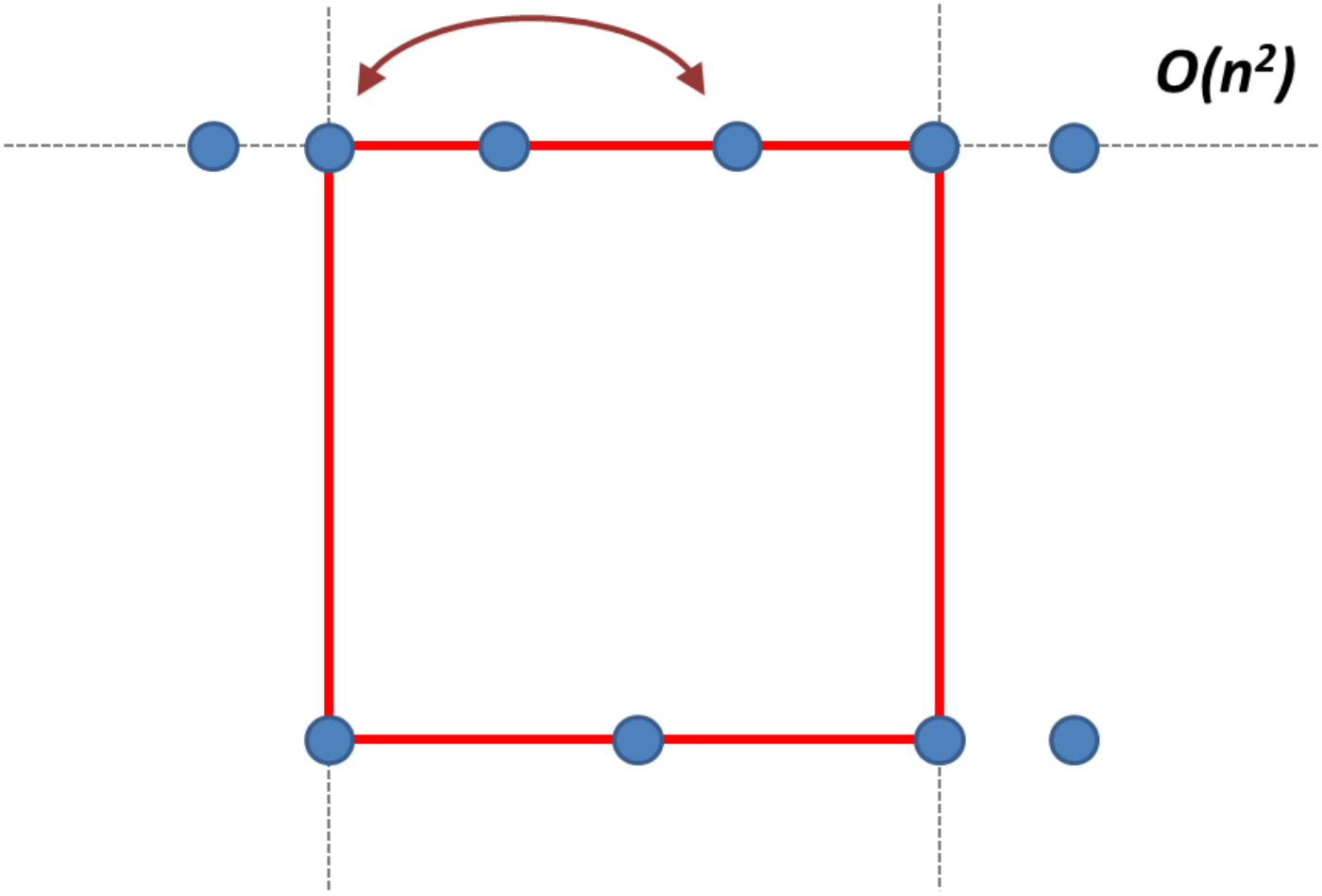
Equiroaster



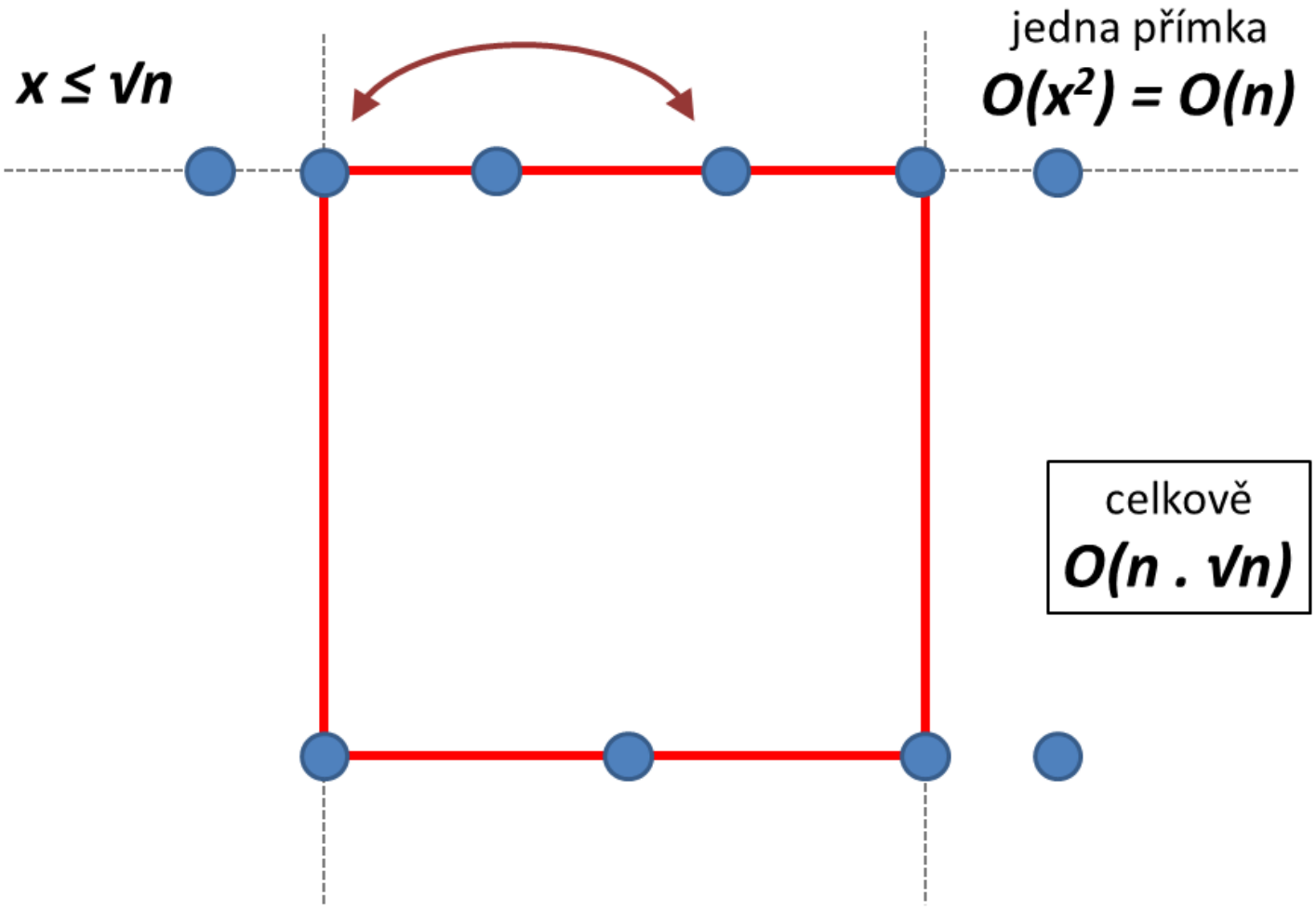
Equiroaster

- Create a list for each X/Y coordinate.
- For each point, choose the shorter list
- Iterate over all points from the list
- For each pair calculate the two remaining points
- Check whther the points exists (map)
- $O(N\sqrt{N}log(N))$ (logarithm can be futher optimized)

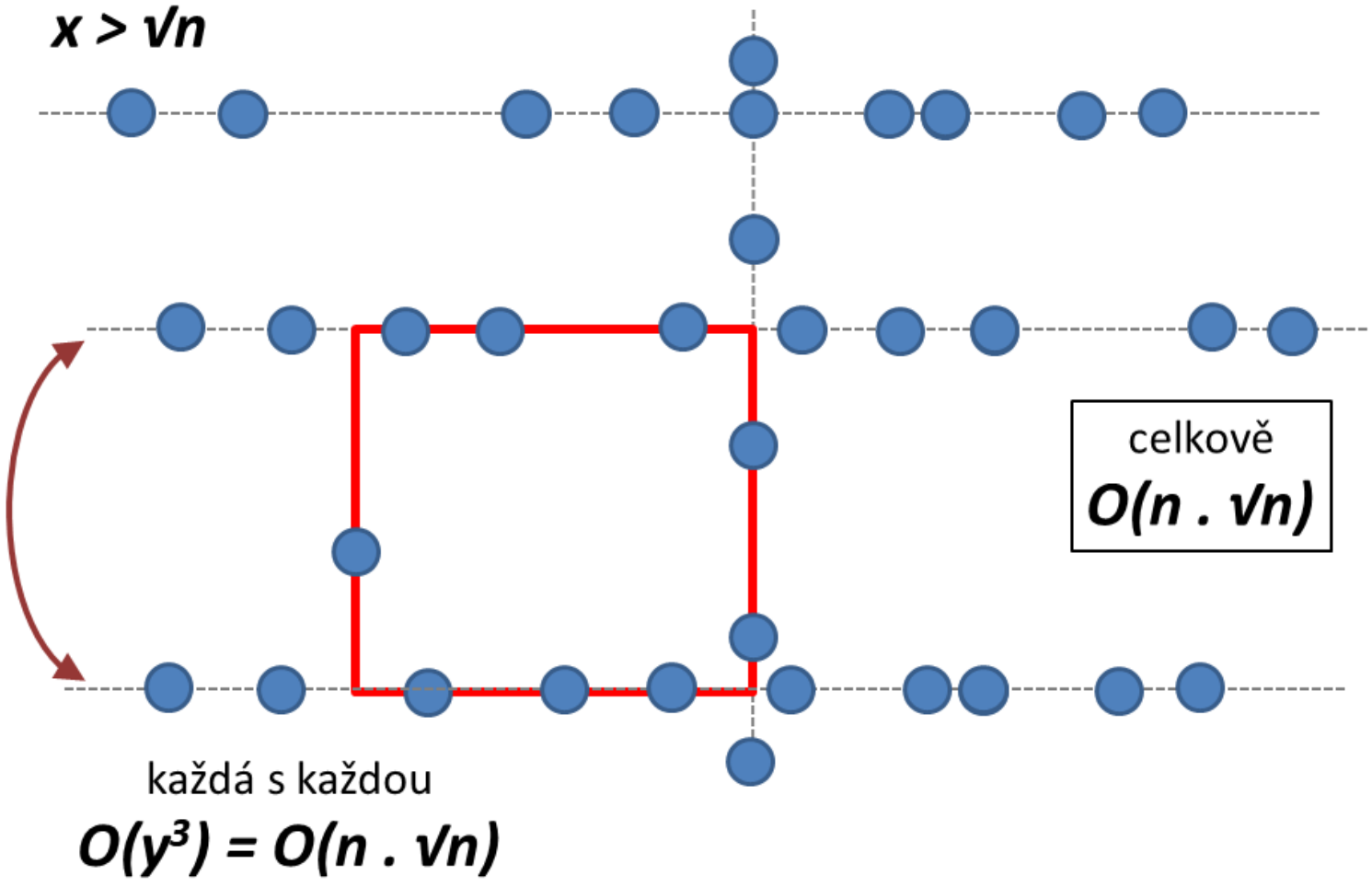
Equiroaster — Alternative Solution



Equiroaster — Alternative Solution



Equiroaster — Alternative Solution



Forest Picture

Forest Picture

- Naive solution sufficed – prepare 2D array full of dots and for each tree in the input loop through each character it consists of and if the character is in visible part of picture then place it in the array
- There were no overlaps allowed not even of their bounding boxes
- This approach runs in $\mathcal{O}(m^2 + n)$ time

```
*********
*_o_^_o_*
*../|\..*
*..._|_..*
*********
```

Shooting Gallery

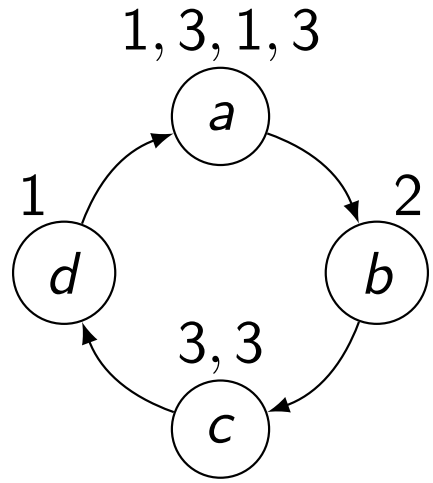
Galery

- Dynamic Programming
- Keep begin/end: Take maximum of [begin+1,end]/[begin,end-1] (if not equal)
- $O(N^2)$

Ice cream samples

Ice cream samples

- Naive solution – fix a starting stand and consecutively include following stands until all brands of ice cream are collected
- This solution runs in at least $\mathcal{O}(n^2)$ time $\Rightarrow$ TLE



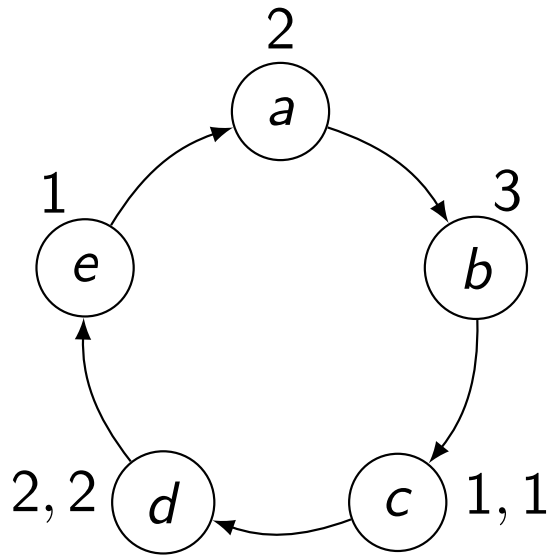
- The faster solution is to use two-pointer technique to dynamically enlarge and shorten currently observed sequence of stands
- In the beginning the sequence is empty
- We must enlarge the sequence if there are still some brands yet to be collected
- Conversely, if all brands have been collected, we can shorten the sequence

Ice cream samples

We still have two issues to solve:

- First, we need to effectively calculate the amount of different brands that have been collected
- Solution is simple – keep a frequency array of size k, i.e. for each brand dynamically store the number of collected samples
- Then, while extending the sequence, we increase the count of different brands on the first occurence of a brand's sample
- While shortening the sequence, we decrease the count of different brands on the last occurence of a brand's sample

- Secondly, as the stands are positioned circularly, the solution might span over the last stand to the first ones
- It suffices to go through the input twice (imagine as if the input was concatenated with itself once)

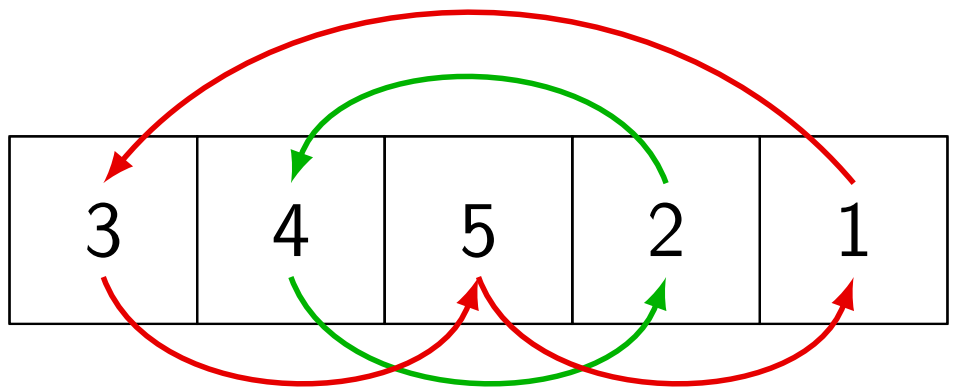


- $\Rightarrow$ By combining mentioned techniques, we obtain an approach that runs in time linear with the amount of samples on the input

Dark Ride with Monsters

Dark Ride with Monsters

- Finding permutation cycles
- $O(N)$

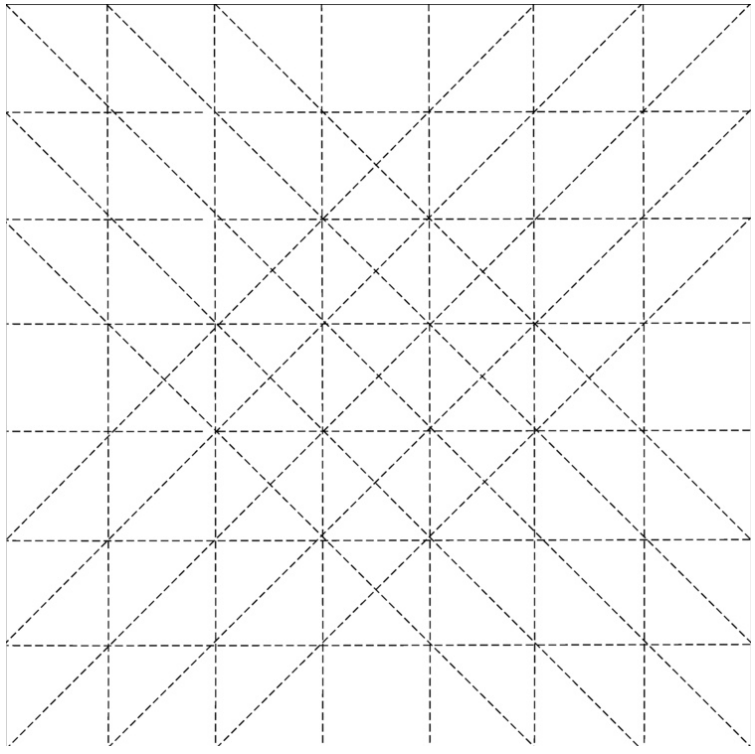


Go Northwest!

Go Northwest!

- We need to count the number of ways to select two (not necessarily distinct!) points such that when we connect them by line the angle between the line and horizontal axis is 45 degrees
- How?

Go Northwest!



- The coordinates of points can help us to index diagonals to find out how many points lie on the same diagonal
- The northwest diagonals can be indexed by $x + y$
- The northeast diagonals can be indexed by $x - y$

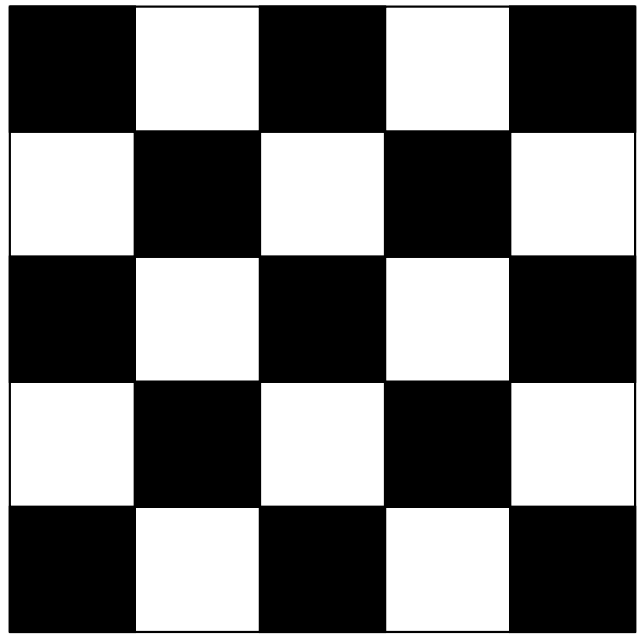
Go Northwest!

- For each diagonal where lie x points we can find out the number of ways to select two points to form a line as $x \cdot (x - 1)$
- That leaves us to sum result over all diagonals and divide it by number of all options n^2
- Total running time: $\mathcal{O}(n)$ or $\mathcal{O}(n \cdot log\ n)$ depending on the indexing of diagonals

Punching Power

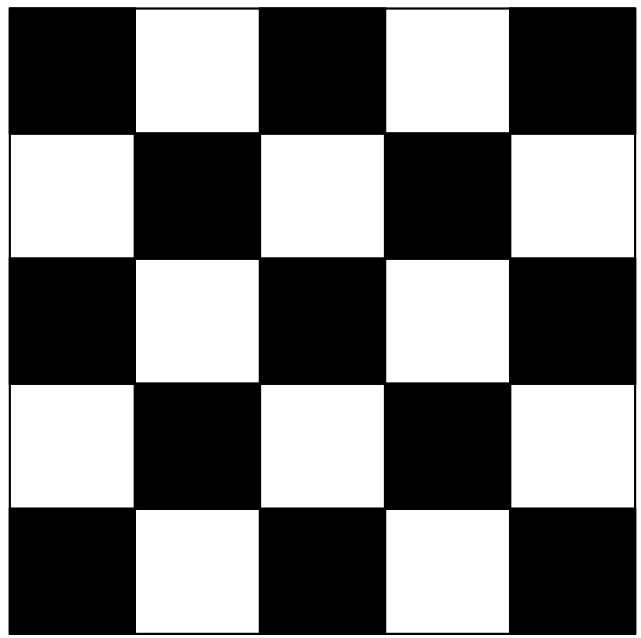
Problem of Pissoir

Problem of Pissoir



Problem of Pissoir

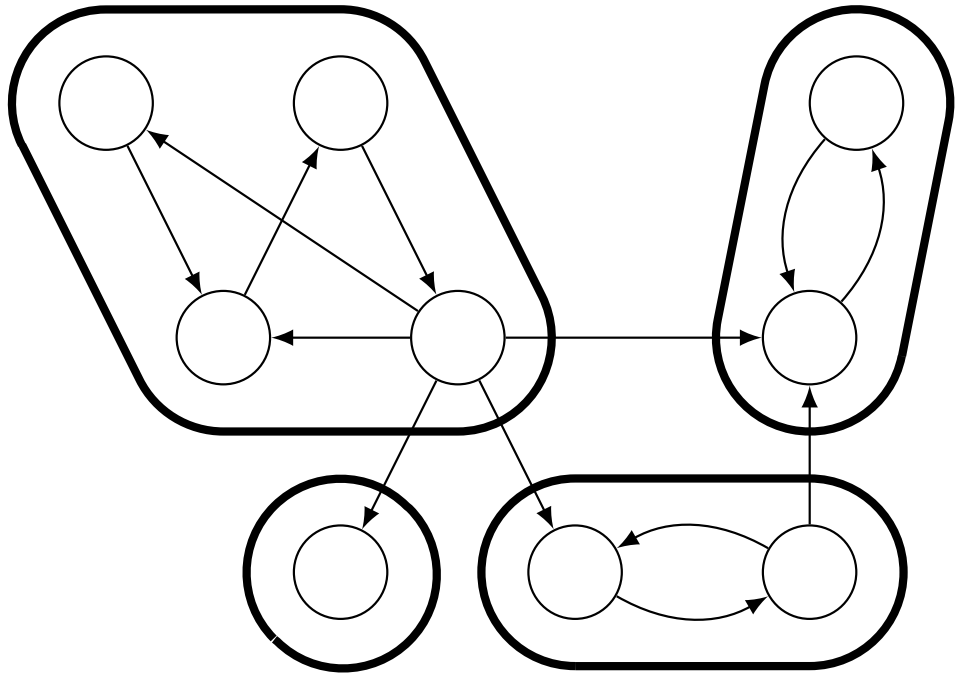
- Number of minimal erases == minimal match
- Minimal match can be solved for example by minimum flows
- Optimal $O(N\sqrt{N})$ (but worse matching was alowed too)



Treetop Walkway

Treetop Walkway

- The problem is to make graph strongly connected with minimal number of edge additions
- First, we find the condensation of the input graph
- This is possible to do in $\mathcal{O}(n+m)$ time



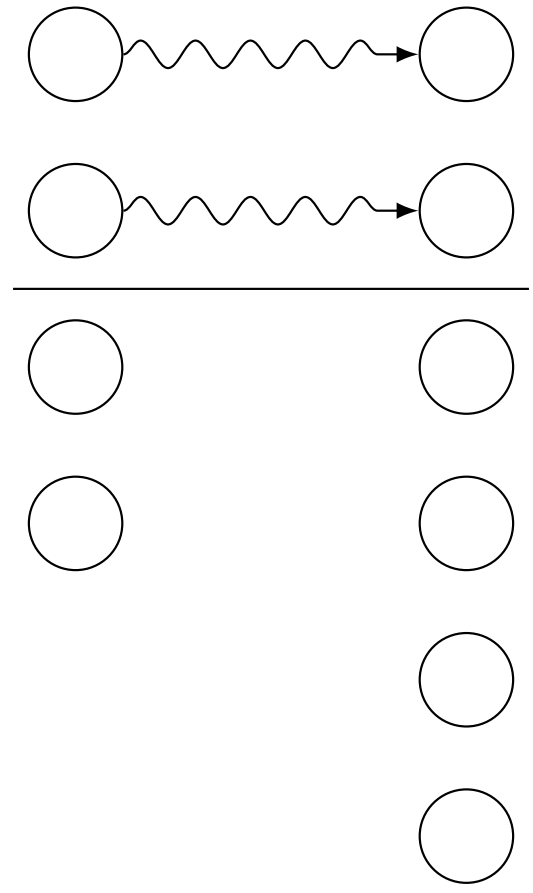
- The resulting graph is a DAG (directed acyclic graph)

Treetop Walkway

- We have to add at least $\max\{\#\text{sources}, \#\text{sinks}\}$ edges
- But is this number of edges always sufficient?

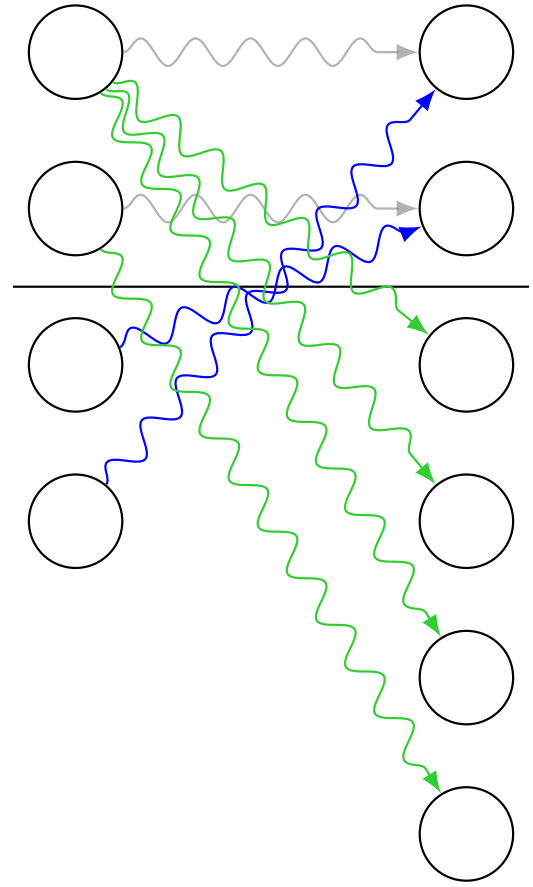
The idea:

- WLOG assume $\#\text{sources} \leq \#\text{sinks}$
- Repeatedly run DFS from all sources; after the search finds a sink, terminate the search, and go to the next source (keep the information about visited sources)
- This approach divides sources/sinks into those that found a match and those who didn't



Treetop Walkway

- Observe that for every unmatched source, there is a path from it to a matched sink
- Similarly, for every unmatched sink, there is a path from a matched source to it



Treetop Walkway

- Now it suffices to do the following:
- Create an oriented cycle by connecting matched sinks/sources and unmatched sinks that do not have a source counterpart
- Connect unmatched sinks with unmatched sources that haven't been connected yet
- Observe that graph is now strongly connected

- Last question – how to map edges from condensation nodes to vertices of the original graph? Solution – choose any vertex from the strongly connected component
- Total time: $\mathcal{O}(n+m)$