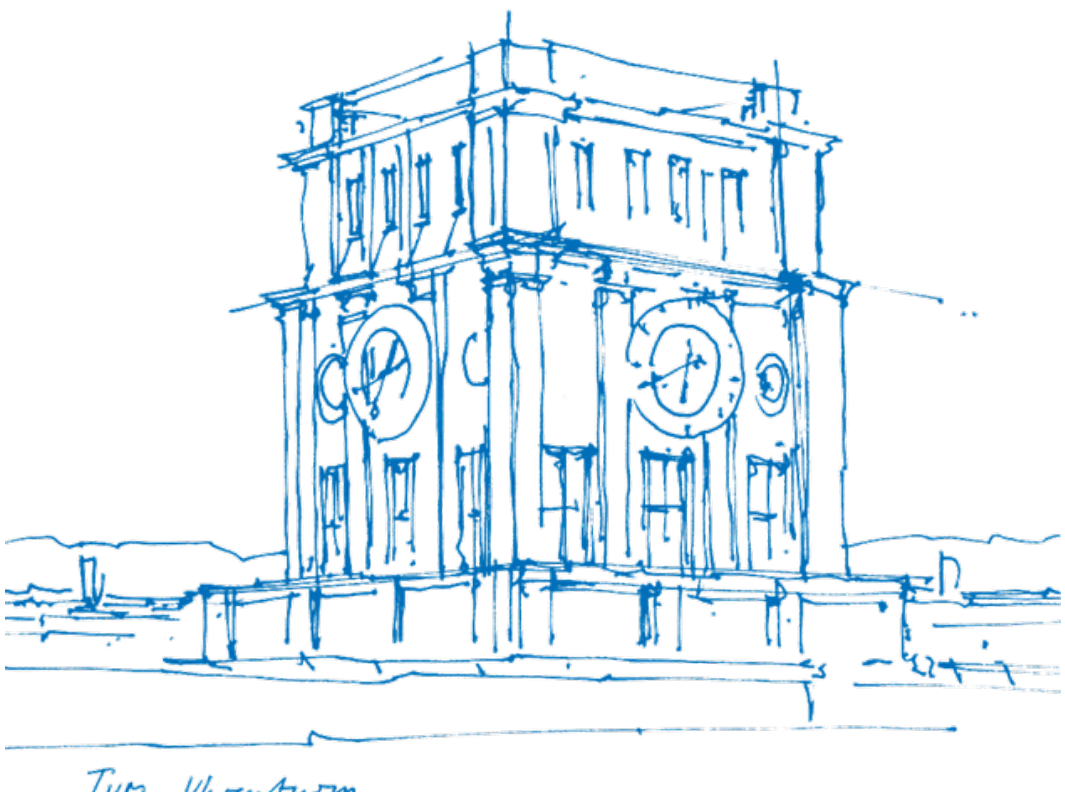
School of Computation, Information and Technology
Technical University of Munich

AACPP WiSe 2026/27

Preliminary meeting



Mateusz Gienieczko, Julia Spindler

School of Computation, Information and Technology
Technical University of Munich

2026.07.09

Structure of the course

Meetings on campus, we discuss the solution to the previous problem and an algorithmics topic.

Every meeting you get a new task to solve for the next one.

Each task worth $10 + 10$ points.

Grading

Before solution is given points count double.

You can submit until end of semester for the normal point amount.

Expect $50\% + \varepsilon$ points to pass.

Grading

Before solution is given points count double.

You can submit until end of semester for the normal point amount.

Expect $50\% + \varepsilon$ points to pass.

Example – you solve the simpler case in a problem worth $3/10$ points before the solution is given, and get 6 points. You later submit the model solution that gives the remaining 7 points. In total you have 13 points for this task.

Course plan

This is an example of topics we could cover, subject to change and reordering.

- Dynamic Programming, combinatorics
-
-
-
-
-
-
  -
  -

Course plan

This is an example of topics we could cover, subject to change and reordering.

- Dynamic Programming, combinatorics
- Advanced BST structures (Splays, treaps...)
-
-
-
-
-
  -
  -

Course plan

This is an example of topics we could cover, subject to change and reordering.

- Dynamic Programming, combinatorics
- Advanced BST structures (Splays, treaps...)
- Advanced graph algorithms (flows, cuts, matchings...)
-
-
-
-
  -
  -

Course plan

This is an example of topics we could cover, subject to change and reordering.

- Dynamic Programming, combinatorics
- Advanced BST structures (Splays, treaps...)
- Advanced graph algorithms (flows, cuts, matchings...)
- Geometry (convex hulls, sweep algorithms...)
-
-
-
  -
  -

Course plan

This is an example of topics we could cover, subject to change and reordering.

- Dynamic Programming, combinatorics
- Advanced BST structures (Splays, treaps...)
- Advanced graph algorithms (flows, cuts, matchings...)
- Geometry (convex hulls, sweep algorithms...)
- String algorithms (suffix trees, KMR, LCP...)
-
-
  - ▸
  - ▸

Course plan

This is an example of topics we could cover, subject to change and reordering.

- Dynamic Programming, combinatorics
- Advanced BST structures (Splays, treaps...)
- Advanced graph algorithms (flows, cuts, matchings...)
- Geometry (convex hulls, sweep algorithms...)
- String algorithms (suffix trees, KMR, LCP...)
- Number theory (modular arithmetic, FFT...)
-
  - ▸
  - ▸

Course plan

This is an example of topics we could cover, subject to change and reordering.

- Dynamic Programming, combinatorics
- Advanced BST structures (Splays, treaps...)
- Advanced graph algorithms (flows, cuts, matchings...)
- Geometry (convex hulls, sweep algorithms...)
- String algorithms (suffix trees, KMR, LCP...)
- Number theory (modular arithmetic, FFT...)
- NP-complete problems
  - Approximation
  - Kernelisation

Evaluation

judge.db.cit.tum.de

You upload a single file in `.rs`, `.cpp`, or `.c`.

It gets compiled and ran on **multiple test cases**.

Your program must read from **standard input** and write to **standard output**.

Your output is checked for correctness.

The environment is constraint wrt. capabilities, **time**, and **memory**.

Real Life

Coding solutions for competitive programming is different than actual coding.

Real Life

Coding solutions for competitive programming is different than actual coding.

BUT

it **is** very educational!

Testing

Random tests are generally bad, but **fuzzing** is an important technique.

Widely used in production software.

Debugging

Duh.

Debugging

Duh.

Print debugging as well, we just call it “structured logging” to sound professional.

Complexity

No one will ask you to prove complexity at a Real Job™...

Complexity

No one will ask you to prove complexity at a Real Job™...

BUT

they will ask “will this work if we have a million entries”.

Problem solving

We have a giant toolbox at our disposal and a problem to solve.

Usually not as well defined, but translating requirements to something sensible is similar.

The main difference is there is no judging system.

Problem solving

We have a giant toolbox at our disposal and a problem to solve.

Usually not as well defined, but translating requirements to something sensible is similar.

The main difference is there is no judging system.

Try not to solve tasks by a thousand resubmissions, for your own sake.

How to solve a problem?

1. Read and understand the statement.
2. Translate from author's imagination into a more formal version.
3. Figure out a correct but perhaps slow solution, or solve a simplified version of the problem.
4. Try to speed it up with observations, generalise.

Example: XAP

Task: XAP
Experimental Assault Pigeons

AACPP SuSe 2024 Round 2 Memory: 32MiB (Java: 128MiB) 2024.05.28 – 2024.06.11

After helping with the Universal Signalling System, Byteman has been promoted higher in the ranks of Byteland's IT Task Force and is now overseeing the computerised mission control system for the Bytelandian Air Force. The hottest innovation in aerial operations? *Pigeons*.

BAF wants to use specially trained pigeons for its experimental Rapid Aerial Payload Delivery programme. In theory they could be used to deliver messages, electronic interference devices, small explosives, or smaller military animals[1], while remaining undetected in enemy territory. This "undetected" part is Byteman's assignment right now – he needs to help XAP units evade radar.

Byteman has access to a training area simulating the territory to inflitrate. From a bird's eye view, the area is a rectangular 2D grid divided into three areas along the X axis. The top and bottom area spanning y values $[0, 100]$ and $[w + 100, w + 200]$ are radar arrays – there are

Example: XAP

n radars in total, and each has a circular detection radius. The edge of the detection range is still dangerous and can detect a pigeon. The flight path is restricted to the middle area of width w. The pigeons start at $x = 0$ and any $y \in (100, w + 100)$, while its goal is to reach l on the X axis. The third dimension does not matter – the radars' operating range is much higher than the maximum altitude of any pigeon.

Fortunately, unlike regular pigeons, the XAP units are smart and agile fliers. They can turn sharp angles in an instant to evade radar detection – their trajectory forms an arbitrary polygonal chain. The catch is that a well-designed radar network might make navigating through the entire territory *impossible*. XAP units might require preliminary strikes – a number of radars to be taken down before the operation begins. To make operations cost-effective, though, the BAF wants to *minimise* the number of destroyed radars.

Example: XAP

Input

In the first line of standard input there are three integers n, w, l, in order: the number of radars, the width of the middle area, and the target x coordinate.

The next n lines contain the description of the radar system. In the i-th line there are three integers, x_i, y_i, r_i, describing the coordinates of the i-th radar and its detection radius, respectively.

There is always at least two radars, one in each array. The x coordinates are always between 0 and l, whereas the y coordinates are always in $[0, 100] \cup [w + 100, w + 200]$.

Output

Your program should output two lines. The first line should contain one integer $0 \leq k \leq n$ – the minimal number of radars that have to be removed to clear a valid flight path.

In the next line there should be k unique integers between 1 and n, denoting which of the radars need to be destroyed, in ascending order. If $k = 0$ then the line must be left blank.

While the minimal k is well defined, there might be more than one correct set of radars of size k that can be destroyed. Your program may output any of them.

Example: XAP

Example

For the input:

```
7 200 700
175 88 100
362 44 200
548 88 125
100 312 62
294 312 106
456 308 127
553 326 88
```

one of the correct outputs is:

```
2
2 6
```

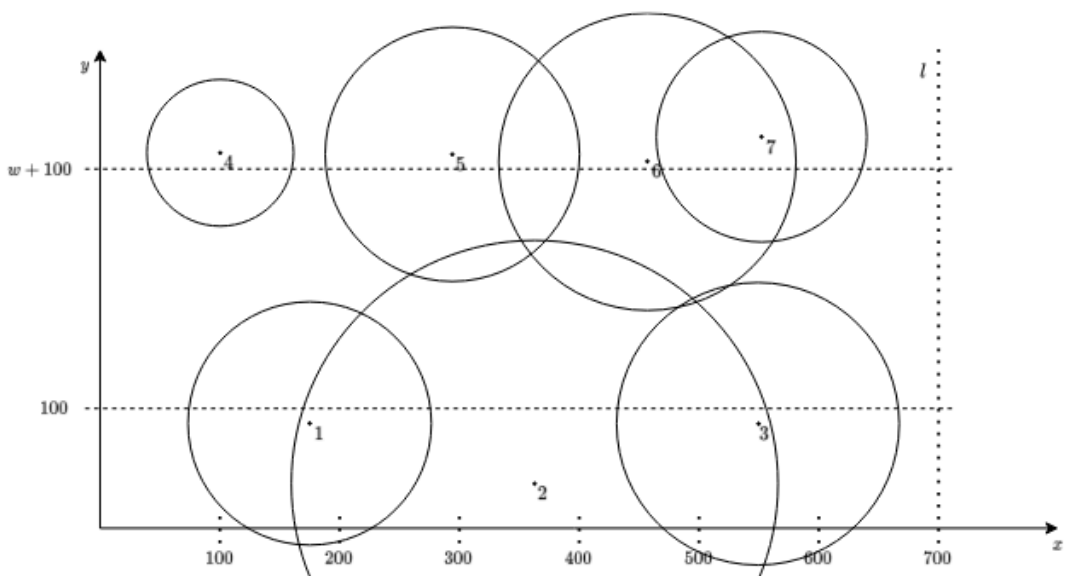


Figure 1: Test setup with 3 radars in the lower and 4 in the upper area. There is no clear path through the radars.

Example: XAP

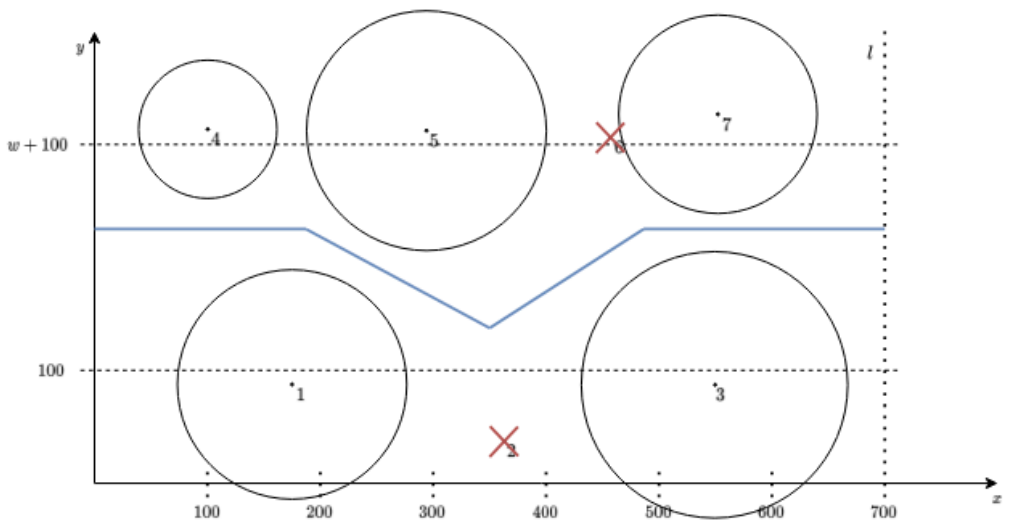


Figure 2: Two destroyed radars (2 and 6) clear the flight path (in blue).

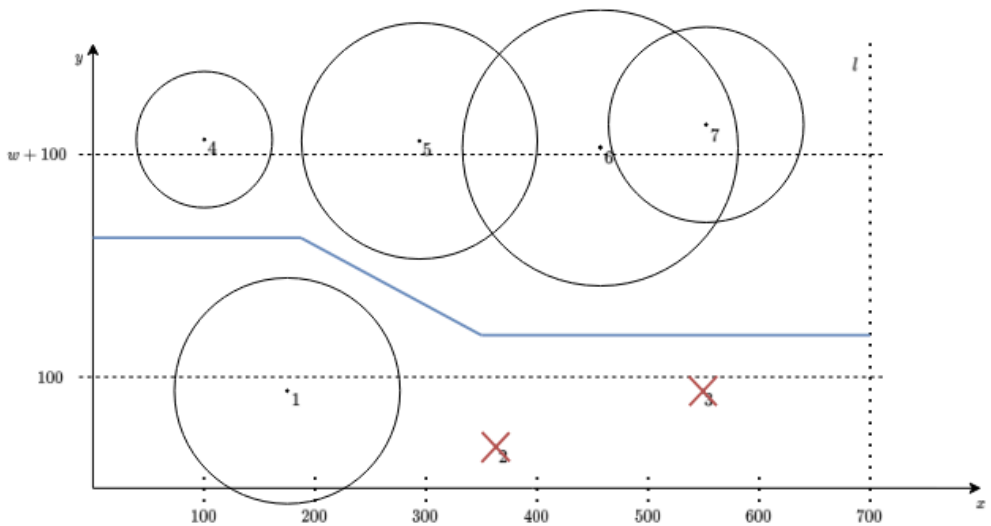


Figure 3: Alternative correct solution where radars 2 and 3 are destroyed.

Example: XAP

Limits

Your solution will be evaluated on a number of hidden test cases divided into groups. Points for a group are awarded if and only if the submission returns the correct answer for each of the tests in the group within the allotted time limit. These groups are organised into subtasks with the following limits and points awarded.

In all tests each radar radius is limited by $10{,}000$.

Partial points

If your solution outputs the correct number of radars (first line of output), and the second line is left blank or not correct, it will receive 50% of the points for a given test group.

Subtask	Limits	Points
1.	$2 \leq n \leq 20$, $1 \leq w \leq 800$, $1 \leq l \leq 1{,}000$	2
2.	$2 \leq n \leq 40$, $1 \leq w \leq 800$, $1 \leq l \leq 4{,}000$	2
3.	$2 \leq n \leq 1{,}000$, $1 \leq w \leq 800$, $1 \leq l \leq 25{,}000$	4
4.	$2 \leq n \leq 5{,}000$, $1 \leq w \leq 2{,}500$, $1 \leq l \leq 40{,}000$	2

Simplify

What if radars are only on one side?

Simplify

What if radars are only on one side?

When are they on both sides but answer is 0?

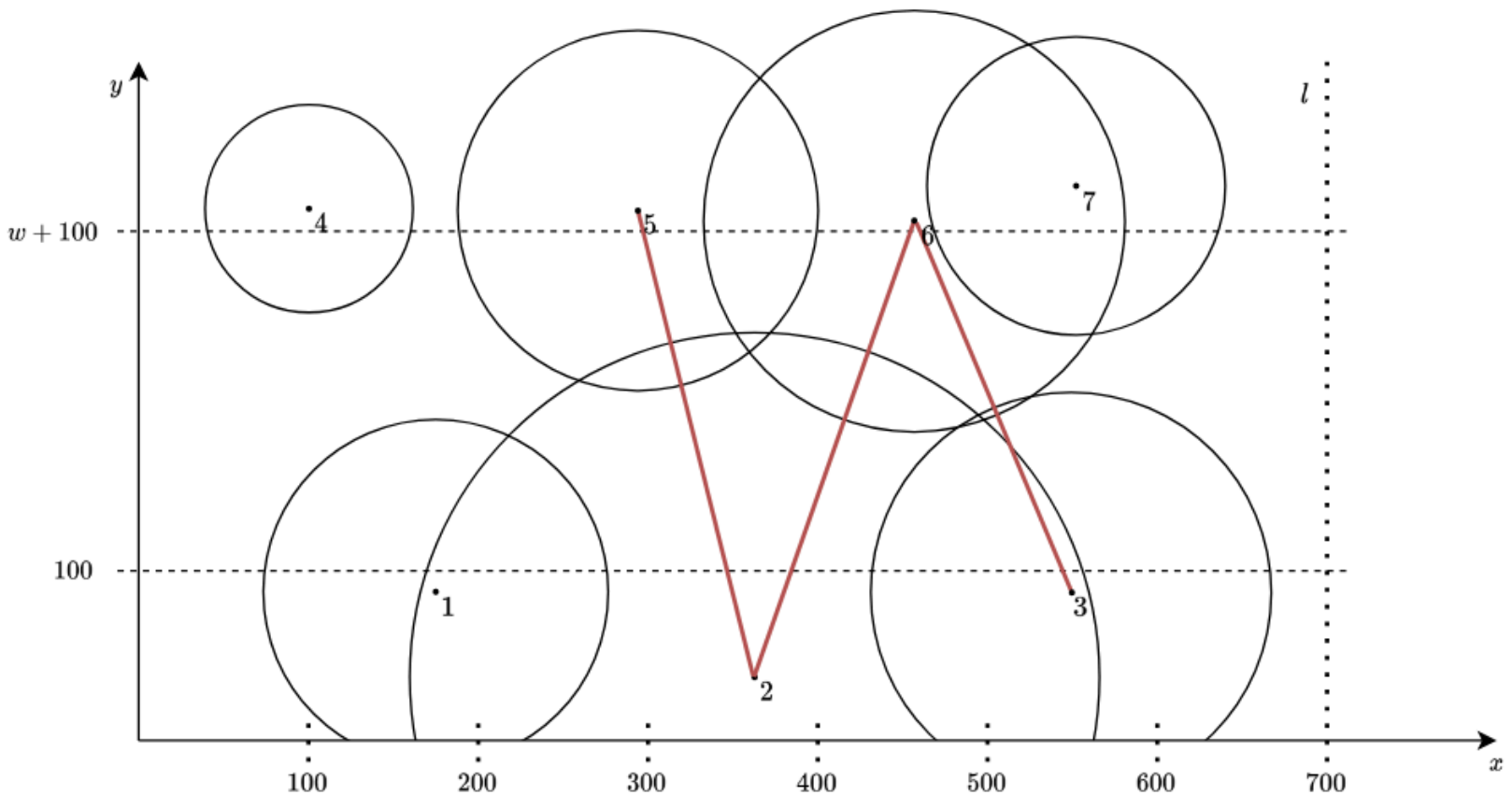
Simplify

What if radars are only on one side?

When are they on both sides but answer is 0?

Simplest case where answer is 1?

Formal characterisation



Formal characterisation

Remove minimal number of vertices to remove all edges.

Minimal Vertex Cover in a bipartite graph.

Greedy idea?

Remove the radar that causes the most interferences?

Oh no it doesn’t work

If it doesn’t work on example or your manual tests – good!

What if we don’t have a counterexample?

Proving correctness

Sometimes solution doesn’t work fundamentally.

Sometimes it’s a coding bug.

Hard to find a bug if you’re not sure that your solution is supposed to work.

At least try to prove it to yourself

Often you'll talk yourself into a counterexample.

... or maybe it actually works conceptually and then you need to debug.

Output?

Sometimes it’s possible to generate a test for a given output, but not always.

Brute-force solutions to compare against.

XAP brute force

Just select a subset to circles to remove and check if it works.

Go over all subsets.

$\mathcal{O}\left(2^n n^2\right)$, but it works for subtask 1.

Generate tests with $n \leq 20$.

XAP improvements

... we actually only need to select a subset on one side and the other side is determined.

$\mathcal{O}\left(2^{\frac{n}{2}}\right)$ (a.k.a $\mathcal{O}\left(\sqrt{2}^{n}\right)$)

XAP actual solution

Use Kőnig's theorem.

Find a maximal matching and retrieve the cover using the proof.

$\mathcal{O}\left(m\sqrt{n}\right)$

Other formats

Actual competitive programming contests are usually more restricted.

E.g. a round on Codeforces is limited to 120-180 minutes.

Usually no subtasks, penalties for wrong submits (“bombs”).

Team competitive contests where multiple (usually 3) people solve a problemset with **one** computer.

Telegram: https://t.me/joinchat/U-UErY30fWmcgMWB

Other contests

Codeforces

LeetCode

TopCoder

SPOJ

CodeChef

CodinGame

CodeWars

CodeCombat

External resources

Introduction to Algorithms (4th ed.), Cormen et al. (The Algorithms Bible)

Algorithms and Data Structures book by the CS department of Virginia Tech

Competitive Programming Resources by Kunal Kushwaha

Programming Challenges: The Programming Contest Training Manual, Skiena et al.

Competitive Programming book(s) by S. Halim, F. Halim and S. Effendy

Competitive Programming Handbook and others by Antti Laaksonen

Course site

https://db.in.tum.de/teaching/ws2627/aacpp/

For priority in matching send emails with motivation (e.g. CodeForces profile) to giem@in.tum.de **with subject starting with [AACPP]!**