Advent of Code

A short introduction

Benedikt Otto

KIM — Forschungsnahe Dienste
University of Hohenheim

November 4, 2026



Advent of Code

Advent of Code is an Advent Calendar with daily programming puzzles in december, created by **Eric Wastl**: `https://adventofcode.com`

- **Free to join**, just requires linking a GitHub, Google, Twitter or Reddit account.
- Problems are solvable with **any programming language** on any system.
- One puzzle per day. Every puzzle consists of two parts.
- The second part is revealed only when the first part is completed.
- 25 Puzzles per year (until 2024), 12 Puzzles per year (since 2025).

AoC has been around **since 2015**, and is known amongst many programmers.

- Users with at least one correct solution: 330k (in 2023), 290k (in 2024), 210k (in 2025).
- Some well-known companies are sponsors of Advent of Code.
- Until 2024, there was a **Global Leaderboard**. (now only **Private Leaderboards**).



Advent of Code

Every day is part of wacky story that is about you helping Elfs prepare for christmas. The input is in text form and the solution is usually a single number.

- A Puzzle comes with a small **example input** and its solution.
- The real input is **substantially** larger.
- Every player gets their unique input, so **everyone has a different solution**.



Advent of Code

Advent of Code generally has problems with **medium difficulty**, so it is recommended that you know the basics of your programming language.

The difficulty **ramps up** over the days.

- You need to **parse your input** into a form that you can work with.
- The first days are usually simple programming puzzles.
- The later puzzles require you to know and implement **typical algorithms**.
- Some puzzles require **performance optimizations**.

Sometimes puzzles try to throw you for a loop, or require you to realise that they are better solved using mathematics.



Examples

Path finding

Some of the most popular puzzles are **path finding algorithms**, where you need to find the **shortest distance** between two points in your input.

Example: **Reindeer Maze** (2024 Day 16: `https://adventofcode.com/2024/day/16`).

To make it more interesting, these puzzles usually come with an **additional twist**, e.g. Clumsy Crucible (2023 Day 17: `https://adventofcode.com/2023/day/17`).



Examples

Path finding

The common way to solve these kinds of problems is by turning the input into a Graph and using a shortest-path algorithm like **Dijkstra's Algorithm** or **A*-Algorithm**.



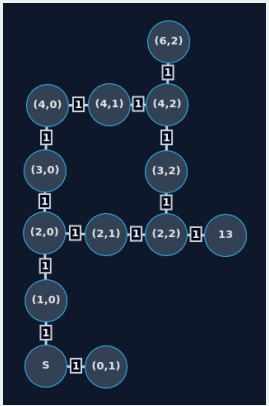


Examples

Path finding

The common way to solve these kinds of problems is by turning the input into a Graph and using a shortest-path algorithm like **Dijkstra's Algorithm** or **A*-Algorithm**.



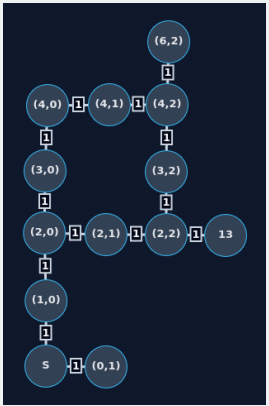


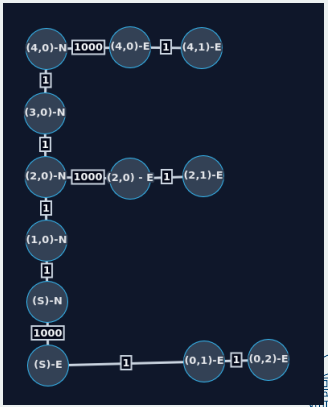
Examples

Path finding

The common way to solve these kinds of problems is by turning the input into a Graph and using a shortest-path algorithm like **Dijkstra's Algorithm** or **A*-Algorithm**.







Examples

Memoization

Some puzzles turn out to be too big for straight brute-forcing and require a bit more insight into the problem. For example:

Hot Springs (2023 Day 12 Part2: `https://adventofcode.com/2023/day/12`)

While the first part is done using a normal recursive approach, the second part involves doing the same thing with an input that is **five times as long**. This means that traditional approach will end up in millions of deep recursions.



Examples

Memoization

Some puzzles turn out to be too big for straight brute-forcing and require a bit more insight into the problem. For example:

Hot Springs (2023 Day 12 Part2: `https://adventofcode.com/2023/day/12`)

While the first part is done using a normal recursive approach, the second part involves doing the same thing with an input that is **five times as long**. This means that traditional approach will end up in millions of deep recursions.

However, once we recognize that we keep re-evaluating the same patterns over and over, we can use a technique called **Memoization** (or **Dynamic Programming**) to cache the results, significantly speeding up the solution.



Examples

Cyclic Solutions

Some solutions seem to be infeasible to simulate fully because the amount of steps needed are way too big. Some Examples are:

- **Parabolic Reflector Dish** (2023 Day 14 Part2: `https://adventofcode.com/2023/day/14`)
- **Pyroclastic Flow** (2022 Day 17 Part2: `https://adventofcode.com/2022/day/17`)

While the first part is a straightforward simulation, the second part asks us to do this simulation for a billion steps, which is impossible to simulate.



Examples

Cyclic Solutions

Some solutions seem to be infeasible to simulate fully because the amount of steps needed are way too big. Some Examples are:

- **Parabolic Reflector Dish** (2023 Day 14 Part2: `https://adventofcode.com/2023/day/14`)
- **Pyroclastic Flow** (2022 Day 17 Part2: `https://adventofcode.com/2022/day/17`)

While the first part is a straightforward simulation, the second part asks us to do this simulation for a billion steps, which is impossible to simulate.

However, there are only a finite amount of possible combinations, so at some point the simulation state will repeat. So the problem reduces to figuring out when this occurs, and then we can calculate the final state by figuring out how many cycles we run through.



Examples

Notable Favourites

Some of my favourite AoC puzzles:

- **Guard Gallivant** (2024 Day 6: `https://adventofcode.com/2024/day/6`)
- **Warehouse Woes** (2024 Day 15: `https://adventofcode.com/2024/day/15`)
- **Keypad Conundrum** (2024 Day 21: `https://adventofcode.com/2024/day/21`)



Examples

My Solutions

You are allowed to publish your AoC solutions, as long as you don't put the puzzle text or your input into your solution. I usually push my solutions to my AoC GitHub page:

`https://github.com/mafuba8/adventofcode`.

I have created a private leaderboard for the Tinkering Club / Hohenheim. If you are interested, you can join it with the code `3363064-fc4da8ab`.



bwHPC Team at the University of Hohenheim

Dr. Konstantin Kuck
Garbenstraße 11, 70599 Stuttgart
2nd floor, Room no. 243
Phone: +49 (0)711 459–22085
Mail: konstantin.kuck@uni-hohenheim.de

M.Sc. Benedikt Otto
Garbenstraße 11, 70599 Stuttgart
2nd floor, Room no. 243
Phone: +49 (0)711 459–23886
Mail: benedikt.otto@uni-hohenheim.de

Thank you for your attention.