

Introductory Session

Fine-Grained Algorithm Design and Engineering

October 30, 2026

Geri Gokaj, Sebastian Angrick

Why Algorithmic Challenges?

- theoretical research focusses on worst case analysis or special cases
- in practice, problems are solvable efficiently
  - even in the general case

at least more efficiently than suspected

How well do theoretic approaches work?
Can we find new ones?

- better understand engineers
- baseline: beat general purpose solvers!





  - important for industry!

You will ...

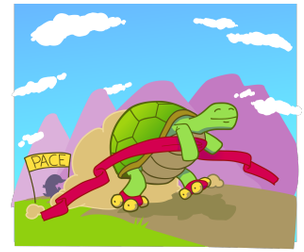
- ... learn to tackle real algorithmic problems
- ... differentiate between theoretic results and practical possibilites
- ... engineer a solver from scratch
- ... apply standard algorithmic techniques
  - ... and learn their limitations
- ... use heuristics and general purpose solvers
- ... learn to evaluate algorithmic approaches

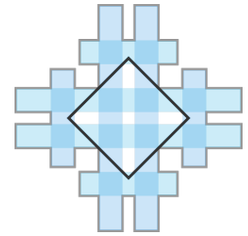
The Challenge

Learn how to design & implement fast algorithmic solutions!

- algorithmic challange
  - compuationally "hard" problems
  - write solver against set of (secret) test instances, public training set provided
  - evaluated by runtime and/or quality of solutions
- specifics determined by challenge
  - usage of libraries and other solvers allowed if open source

PACE challenge





CG:SHOP challenge

Your Contribution

- read into theory
- develop practically fast solvers
- evaluate on own and given training sets

we do **not expect** you to actually compete but **encourage** you to do so

Course Details

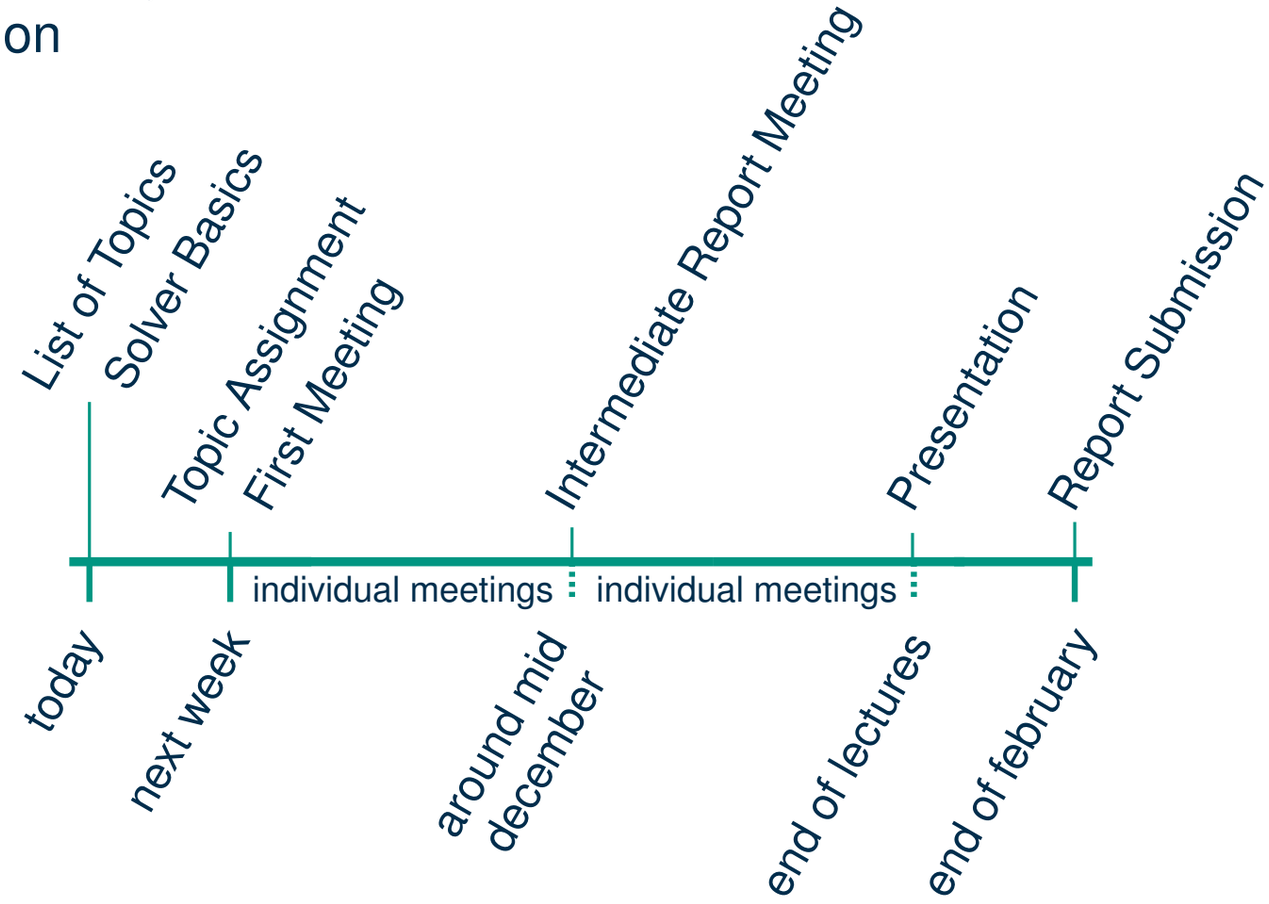
- prerequisites: Solid foundation in algorithms, familarity with either C++, Rust, or Python
- 6 ETCS
- regular meetings on individual basis



Geri Gokaj
geri.gokaj@kit.edu



Sebastian Angrick
sebastian.angrick@kit.edu



Today:
- list of topics
- expectation and remarks on grading
- basic techniques for solvers

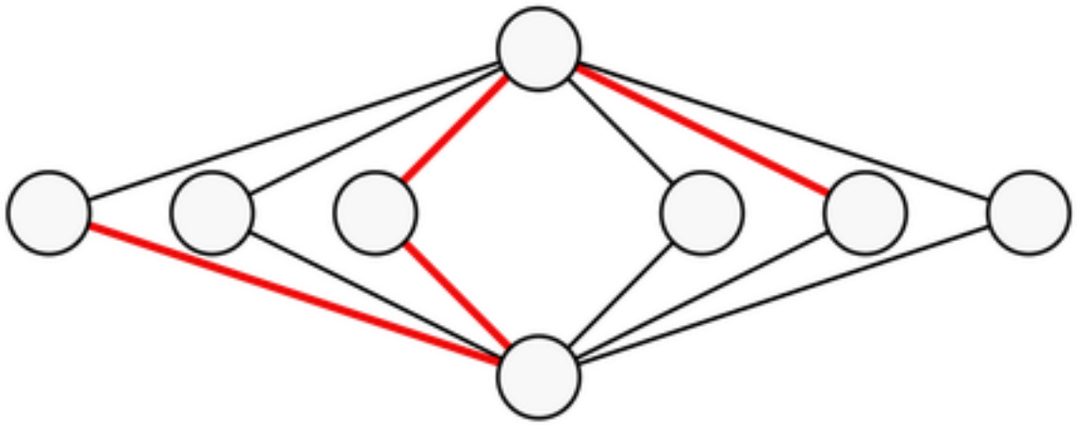
List of Topics

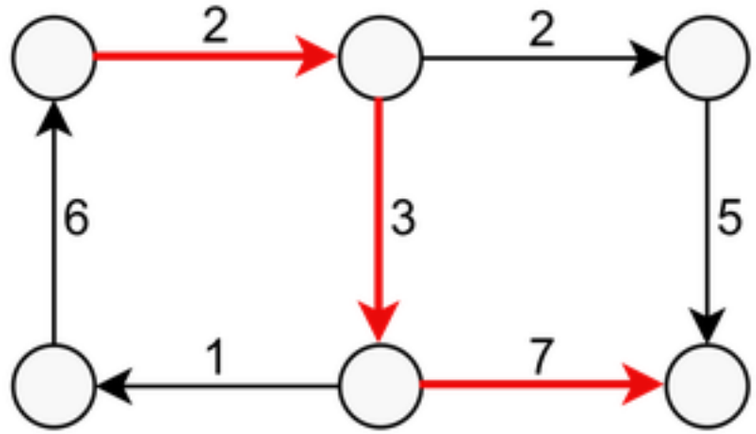
- PACE Challenge 2027: k-Path
  - unweighted track
  - weighted track
  - heuristic track
- CG Challenge 2027: Lawn Mowing

- work in groups of 2-3 people (preferred 2)
  - in case of a preferred team partner, please both mention it
- send an email with your ranked preferences until 02.11 (12pm) to
  `geri.gokaj@kit.edu` and `sebastian.angrick@kit.edu`
- we notify you via email about your topic & coordinate meeting times

PACE 2026: k-Path

`pacechallenge.org/2027`





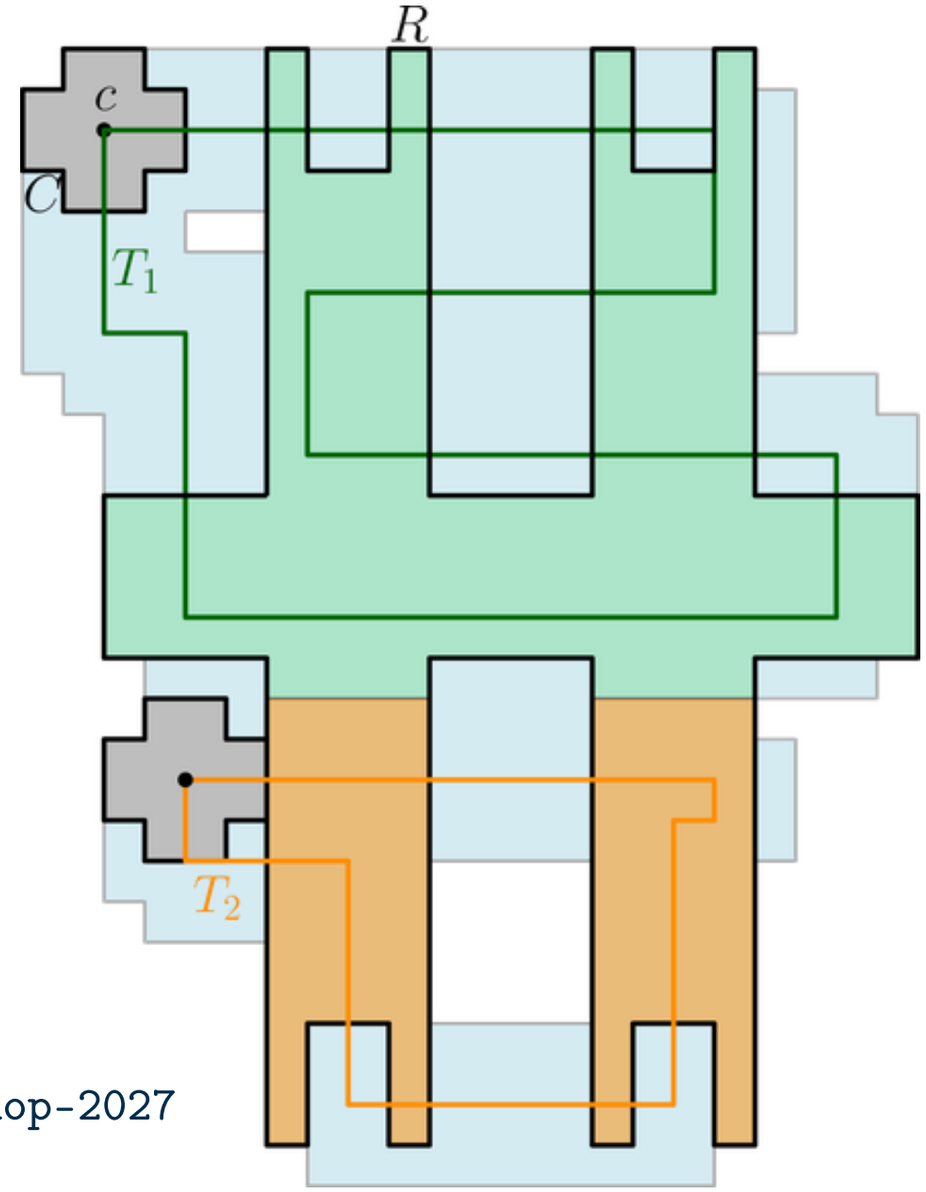
- undirected: determine the maximum length of any simple path
- weighted, directed: find a path of length k with maximum weight
- heuristic: give a best-possible lower bound for the maximum length path

CG:SHOP 2027: Lawn Mowing

given an axis aligned "lawn" *R* ...

and *k* polyomino cutters, ...

find a round-tour for each cutter so that the full lawn is mowed



`https://cgshop.ibr.cs.tu-bs.de/competition/cg-shop-2027`

Expectations and Grading

- regular attendance in meetings
- regular, incremental commits for coding work (in git)
- independent work, both theoretically and practically
- open communication about achievements **and problems**
- we do not expect you to participate in the challenge
  - ... but to mock it

Grading based on your contribution in ...

- meetings / project progress
- final solver
- presentation(s)
- report

Questions?

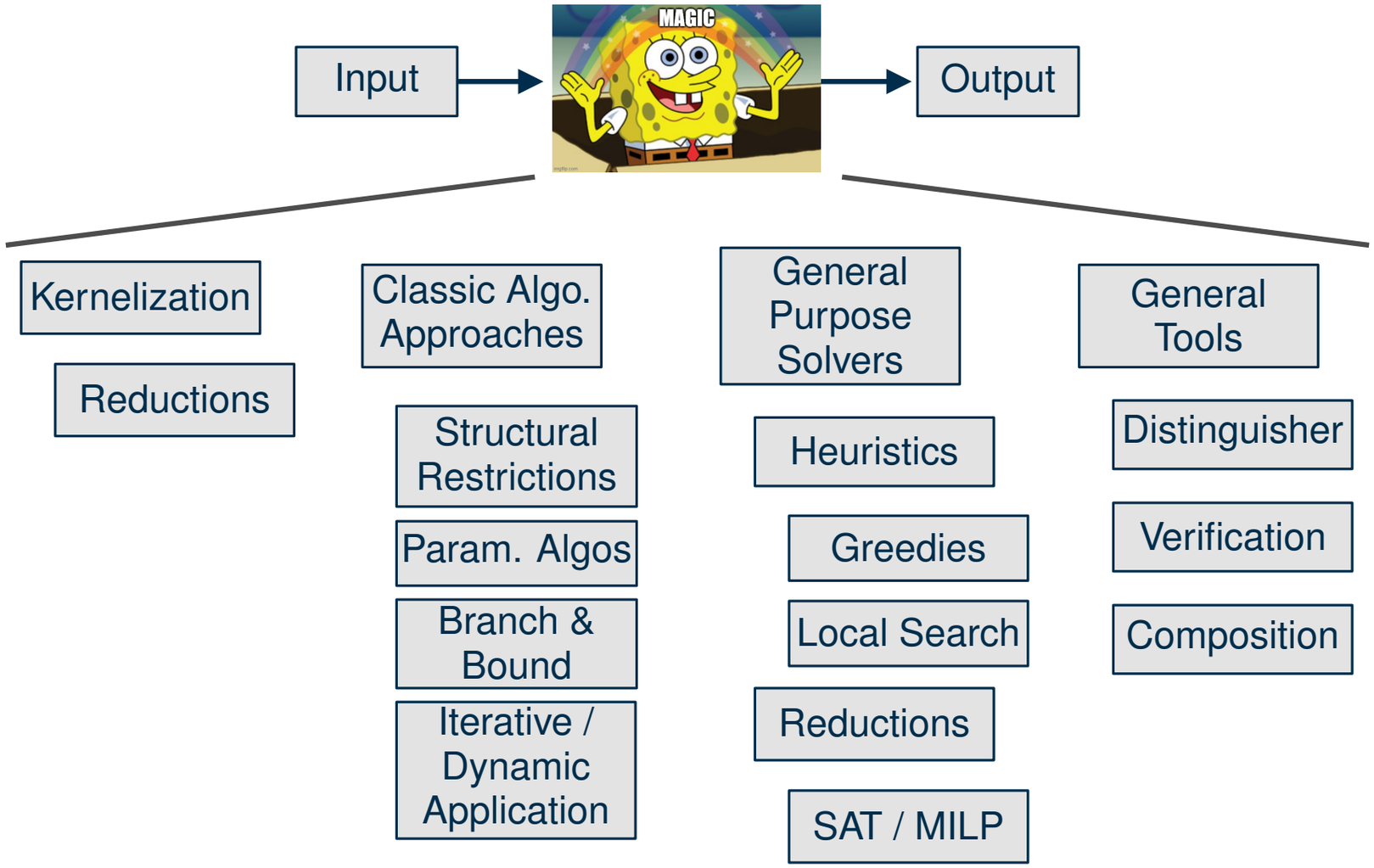
For a very good grade ...

- independent work
- adequate own contribution, novel improvements (in some cases)
- thorough empirical and theoretical analysis of approach (if applicable)
- well-comprehensible presentation
- we do not expect a winning solver!

gen-AI usage

- project phase: AI usage allowed in principal, but ...
  - full declaration required *at all times*
    - in-code indicated for AI-generated functions, orally for AI-generated ideas
    - afterwards: explicit declaration including list of usages and used models
  - ability to defend all your contributions *at all times*
  - expectations are adjusted accordingly
- you may opt-out for AI usage
- presentation / report: usage only allowed for "positive use cases"
  - e.g., language editing, visualizations (again with declaration)
- we may check your submitted material for non-declared AI-usage
  - **non-declared AI usage** is considered **cheating**
- more information: `https://act.iti.kit.edu/resources/ai-usage`

Where the Magic Happens



Techniques

Reductions

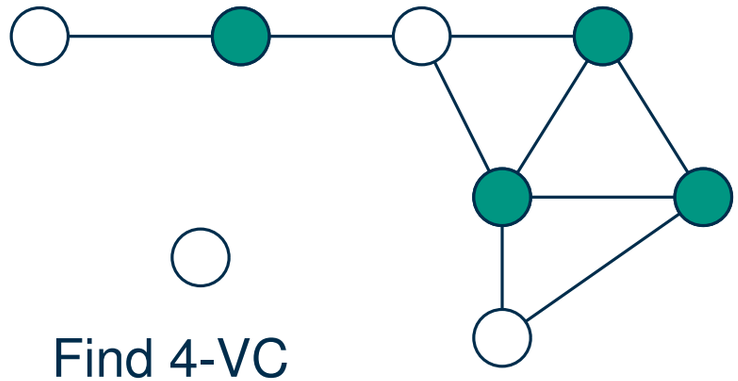
Kernelization

Classic Algo. Approaches

General Purpose Solvers

General Tools

- simplify your problem!
  - less vertices, edges, solution size, etc.
  - (try to) stay away from introducing new problems (weights, etc.)



Find 4-VC

- low vertex degree (0, 1, 2, ...)
- Buss-type arguments
- neighborhood-type arguments
- see any parameterized algorithms lecture

not necessary for challenge
but useful for understanding previous solvers

- very important for many solvers
  - may only reduce special cases, still relevant
  - be aware of runtime, may be run excessively

Techniques

Structural Restrictions | Param. Algos | Branch & Bound | Iterative / Dynamic Application

see any parameterized algorithms lecture

Kernelization

- keep track of your parameters
  - structural parameter (e.g. treewidth, ILP variables, embedding), output size

Classic Algo. Approaches

- runtime exponential in parameter, polynomial in input size
  - parameter small $\Longrightarrow$ fast runtime
- reduce optimization to decision problem

General Purpose Solvers

- selected techniques (with limitations): may still work well!
  - divide & conquer → where to cut?
  - branch & bound → generally too slow
  - treewidth-DP → often too large parameter

General Tools

  - iterative improvement → not really less complicated

Techniques

Heuristics | Greedies | Local Search | Reductions | SAT / MILP

Kernelization

Classic Algo. Approaches

General Purpose Solvers

General Tools

- Undecided which branch to follow? Make an educated guess with heuristics!
  - greedies — super fast!
  - (dual) LP-value — LP-solving in P!
  - Local Search — often non-optimal

→ too bad if used exclusively

→ where to start?

- NP-membership gives reduction to SAT, (M)ILP
  - ridiculously strongly optimized solvers
  - baseline for all practical use cases
- ask for partial solutions / allow slack
- dynamically add constraints to reflect branching decisions
- reduce to other (better solvable) problems (e.g. VC)

EvalMaxSat (github: FlorentAvellaneda)

FindMinHS (github: Felerius, i.a. KIT dev-ed)

GLPK (GNU Linear Programming Kit)



not in solver,
ok for evaluation

Techniques

Verification | Distinguisher | Composition

Kernelization

- fast verification allows better use of heuristics
  - how does a certificate look like? how a negative certificate?
  - heuristic may yield (partial) certificates

Classic Algo. Approaches

- many instances are special enough to be easily solvable
  - what kind of special?
  - distinguishing itself complex, keep track of parameters
    - degree distribution, number of triangles, bound (guess) your parameters

General Purpose Solvers

- compose multiple ideas
  - keep track of bottlenecks

General Tools

- circumvent worst cases: add randomness

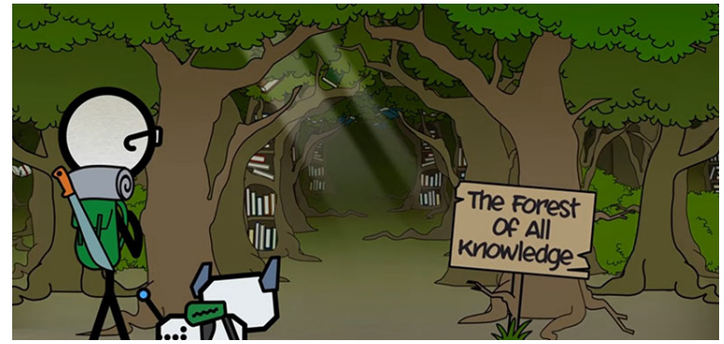
Test Instances

- look at previous write-ups
  - public test instances
  - real-world instances — SNAP project (Stanford Large Network Dataset Collection)
  - randomly generated instances (often hard to solve)
  - other instances harder to predict (and generate)
- course-specific: instances encoding other problems — → (fine-grained) reductions
- persistent output $\Longrightarrow$ file output
  - evaluate your solutions $\Longrightarrow$ feasible?
- visualize your instances!
  - plotting overall results, have individual figures for hard instances
- scripts to run/evaluate large batches

A Good Start ...

- start small: have your boilerplate code running quickly
  - input/output, validation, testing (metrics or visuals)
- progress from easy to hard instances
  - allow unsolved outputs, look at unsolved parts
  - collect hard instances, investigate them
  - find approaches to next portion of instances
- have a naive / general purpose solver running for own evalutation
- switch to general solver should not happen too early
  - often tradeoff between simplicity and generality
  - interesting insights often found for specific instances, generalized later
- sit back and think about the problem
- look at existing algorithmic ideas (last year's winning solvers)

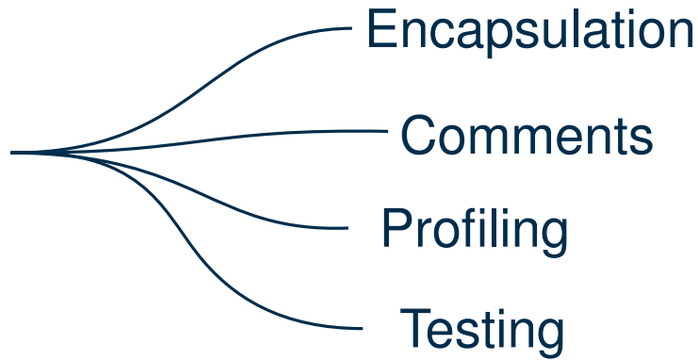
Be Curious!



source: CGP Grey

Common Mistakes

- the "general solution" may be too slow
  - while unneccessary and cumbersome to implement
- try to reuse existing solvers instead of reimplementing
- employ good engineering practises
  - you will change large portions of your code
  - we also need to understand your code
  - keep it extendable, you might need additional operations later on
- give yourselves time to ponder about the problem
  - good solvers find a new angle to look at the problem



Organizational

- repository, please invite us
  - all code, test instances and output, visualizations, etc.
  - scripts for building, evaluation, ...
  - continuous updates (don’t forget to push!)
- readme
  - explain usage: how to build and run solver, how to evaluate
  - contribution (bullet points)
    - what was taken from which source, who worked on what part

Questions?