Introduction to Marathons

BECAUSE THE ONE TO ALGORITHMS IS NOT SUFFICIENT

What do you get in Marathons

- One problem
- Usually 1+ week long
- Can't be solved perfectly in reasonable time limit

Examples of problems

- Travelling salesman problem (TSP)
- Elevator navigation system in hotel
- Playing (simplified) poker vs AI

Common misconceptions

- You need very good algorithmic background
- You need to invest a lot of time
- You need to try a lot of different approaches to each problem

Required skills in SRMs

- Short-term concetration (being fast, avoiding bugs)
- Long-term memory (remembering old problems)
- Lots of experience in old contests

- Short-term concetration (being fast, avoiding bugs)
- Long-term memory (remembering all of the code that you wrote)
- Time managament
- Being open-minded

How do you solve a tough problem

- 10. Understand the problem
- 20. Solve the problem
- 30. Analyze the results
- 40. If (not dead) goto 2

How do you solve a tough problem

- 10. Understand the problem
- 20. Solve the problem
- 30. Analyze the results
- 40. If (not dead) goto ~~2~~ 1

How do you solve a tough problem

- ~~10.~~ Understand the problem
- ~~20.~~ Solve the problem
- ~~30.~~ Analyze the results
- ~~40. If (not dead) goto 2~~ ~~1~~

Using Hill Climbing

- Defining state
- Defining transposition function
- Defining evaluation function

```
current_state = init_state();
while (time_available > 0) {
        new_state = transposition(current_state);
        if (evaluate(new_state) > evaluate(current_state))
                current_state = new_state;
}
```

Using Simulated Annealing

- Defining state
- Defining transposition function
- Defining evaluation function

```
current_state = init_state();
while (time_available > 0) {
        new_state = transposition(current_state);
        if (P(evaluate(new_state)-evaluate(current_state)) < rand())
                current_state = new_state;
}
```

When you should use SA?

- Always when it works!

Random hints

- Write something that works ASAP and improve it by small steps, or at least treat it as a benchmark
- Work only when you feel you'll be productive
- If you can, work on paper
- Visualize what you're doing when getting lost
- Try to dissect complex problem into smaller ones
- Don't do tedious work, when you're feeling creative
- There's no such thing as solving the problem
- Test often

Beyond Marathons

+ Data mining / Data analysis
+ Machine vision
+ Game AI
+ Low-level optimization
+ …
= Problem solving