Submit answers to isak@laiout.co by providing a direct link to your github repo, and make sure to invite https://github.com/isakbm/ to the repo.

Rust Challenges

FizzBuzz

Implement the following function

```
/// fizzbuzz
/// ---
/// returns “Fizz” if n is divisible by 3
/// returns “Buzz” if n is divisible by 5
/// returns “FizzBuzz” if n is divisible by both 3 and 5
pub fn fizzbuzz(n: usize) -> String {
    todo!()
}
```

Test the function by adding a mod tests module.

```
mod tests {
    #[test]
    fn it_works() {
        todo!()
    }
}
```

NOTE: In this challenge, the algorithm is easy to implement. What we are looking for is not your ability to construct the algorithm, we are expecting you to show your ability to structure the code cleverly, and to make use of nice Rust language features, such as match statements or closures. There is no right answer in terms of style, but readability, comments (when they are necessary), and robustness will be considered.

Fibonacci

Implement the following function

```rust
/// fib
/// ---
/// fib(0) = 0
/// fib(1) = 1
/// fib(n) = fib(n-1) + fib(n-2) for all n >= 0
pub fn fib(n: usize) -> usize {
    todo!()
}
```

Implement it using recursion.

Speed it up using memoization.

Bonus points if you package this fibonacci function in a crate, that is, make it importable and easy to use by other rust developers - it should be possible to do

```rust
use fibonacci::fib;

fn main() {
    let n = 40;
    println!(“fib({}) = {}”, n, fib(n))
}
```

Rot13

In this challenge we want you to figure out on your own what to do based on the partially completed code. Complete the implementations and submit the code together with the answer. The answer is a decoded message.

Please do not spoil the fun by using some website that does rot13 for you. :)

```rust
use std::io::BufWriter;
use std::io::Write;

struct Rot13Writer<T>
where
    T: Write,
{
    todo!()
}

impl<T> Rot13Writer<T>
where
    T: Write,
{
    pub fn new(inner: T) -> Self {
        todo!()
    }
}

impl<T> Write for Rot13Writer<T>
where
    T: Write,
{
    fn write(&mut self, buf: &[u8]) -> std::io::Result<usize> {
        todo!()
    }

    fn flush(&mut self) -> std::io::Result<()> {
        todo!()
    }
}

fn main() {
    let mut content = Vec::<u8>::default();

    let mut buff = Rot13Writer::new(&mut content);
    buff.write(b"Lbh penpxrq zl fhcre qvssvphyg pbqvat punyyratr... pbqr vf ddommNst")
        .unwrap();

    println!(
        "result: {:?}",
        content.iter().map(|x| *x as char).collect::<String>()
    );
}

mod tests {
    #[test]
    fn test_rot13() {
        todo!()
    }
}
```

Traveling Salesman Problem (TSP)

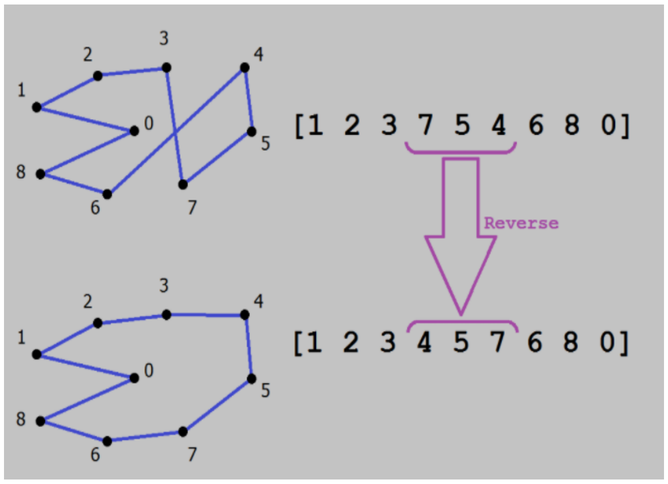
Your task is to solve the traveling salesman problem by implementing one or both of the following heuristics

- Hill climbing
- Simulated Annealing

Problem formulation

This is the classical traveling salesman problem, an np-hard problem, where your task is to find the shortest path that visits all nodes once and only once. Below you'll find some code that indicates how you should approach this, preferably using Rust.

```
struct Node {
    id: usize,
    x: f64,
    y: f64,
}

fn random_nodes(N: usize) -> Vec<Node> {
    todo!();
}

fn tsp_hill_climb(nodes: &Vec<Node>) -> Vec<f64> {
    todo!();
}

fn tsp_simulated_annealing(nodes: &Vec<Node>) -> Vec<f64> {
    todo!();
}

fn main() {
    let N = 10;
    let nodes = random_nodes(N);

    let hc_history = tsp_hill_climb(&nodes);
    let sa_history = tsp_simulated_annealing(&nodes);
}
```



Here is a useful graphic that illustrates how an ordered list of nodes corresponds to a specific candidate solution.

The reversal indicated in this graphic maps one candidate to another, but you are urged to do simpler manipulations than reversals. Get creative, and don't forget to plot the histories of the optimization, for instance using python. The histories are simply a list of lengths of the path at each iteration during the optimization and will give you insight. Please do not hesitate to ask isak@laiout.co questions regarding this exercise.

$O(n!)$ —> exact answer

$O(m)$ where $m <<<<<<< n!$ —> approximate answer (Heuristics!!)