Tarea 1

Práctica de cadenas y colecciones

```rust
use std::collections::HashMap;
use std::env;
use std::fs;

fn main() {
    let args: Vec<String> = env::args().collect();
    let file_path = parse_args(&args);
    let text = open_file(file_path);
    let frequencies = word_count(&text);
    for ( &word,freq) in &frequencies {
        println!("{word}: {freq}");
    }
}

fn parse_args(args: &Vec<String>) -> &str {
    if args.len() < 2 {
        panic!("Not enough arguments");
    }
    &args[1]
}

fn open_file(path: &str) -> String {
    fs::read_to_string(path).expect("Should have been able to read the
file")
}

fn word_count(text: &str) -> HashMap<&str, u32> {
    let mut map: HashMap<&str, u32> = HashMap::new();
    for word in text.split_whitespace() {
        let count = map.entry(word).or_insert(0);
        *count += 1;
    }
    map
}
```

Práctica de algoritmos y generics

```rust
fn mysort<T: Ord + Clone>(arr: &[T]) -> Vec<T> {
    let mut vec = arr.to_vec();
    vec.sort_by(|a, b| a.cmp(b));
    vec
}
```

Aplicación de tu función

```
use std::collections::HashMap;
use std::env;
use std::fs;

fn main() {
    let args: Vec<String> = env::args().collect();
    let (file_path, order) = parse_args(&args);
    let text = open_file(file_path);
    let frequencies = word_count(&text);
    let frequencies_words: Vec<&str> = frequencies.iter().map(|(key, _)|
*key).collect();
    let frequencies_freq: Vec<&u32> = frequencies.iter().map(|(_, value)|
value).collect();
    let sorted_vec: Vec<usize> = match order {
        "w" => my_sort(&frequencies_words),
        &_ => my_sort(&frequencies_freq),
    };
    for idx in sorted_vec {
        println!("{}: {}", frequencies_words[idx], frequencies_freq[idx]);
    }
}

fn parse_args(args: &Vec<String>) -> (&str, &str) {
    if args.len() < 2 {
        panic!("Not enough arguments");
    }
    let path = &args[1];
    let order = if args.len() > 2 { &args[2] } else { "c" };
    (path, order)
}

fn open_file(path: &str) -> String {
    fs::read_to_string(path).expect("Should have been able to read the
file")
}

fn word_count(text: &str) -> HashMap<&str, u32> {
    let mut map: HashMap<&str, u32> = HashMap::new();
    for word in text.split_whitespace() {
        let count = map.entry(word).or_insert(0);
        *count += 1;
    }
    map
}

fn my_sort<T: Ord>(arr: &[T]) -> Vec<usize> {
    let mut idxs: Vec<usize> = (0..arr.len()).collect();
    idxs.sort_by(|&i, &j| arr[i].cmp(&arr[j]));
    idxs
```

```
}
```