Aufgabe 2: Twist

Team: Jan Niklas Groeneveld
Team-ID: 00828

24. November 2018

Inhaltsverzeichnis

1 Lösungsidee

Da die Aufgabe aus zwei Teilen besteht, werden diese hier auch in der Reihenfolge erläutert. Für den ersten Teil der Aufgabe gilt, dass bei Wörtern der mit drei oder weniger Buchstaben kein Twistvorgang stattfinden darf, da der erste und der letzte Buchstabe immer an ihren Positionen bleiben müssen. Bei genau vier Buchstaben werden nur der zweite und dritte Buchstabe getauscht, während bei jeder anderen Buchstabenzahl die tauschfähigen Buchstaben mithilfe eines Zufallsgenerators durchgewürfelt werden. Twistet man einen ganzen Text, muss sichergestellt werden, dass wirklich nur die Wörter getwistet werden, und nicht Satzzeichen und Leerzeichen. Nach dem Twisten muss sichergestellt werden, dass alle Wörter wieder an ihren richtigen Platz finden. Anschließend wird der getwistete Text in eine neue Datei geschrieben.

Zum enttwisten wird zuerst die Datei des Wörterbuches ausgelesen, und für jedes Wort ein Schlüssel erstellt, der sich ergibt, wenn die Buchstaben des Wortes außer dem ersten und dem letzten nach dem Alphabet sortiert werden. Der Schlüssel und das richtige Wort werden dann so gespeichert, dass man, nachdem man einen Schlüssel übergeben hat, das ursprüngliche Wort erhält. Anschließend werden die getwisteten Wörter aus dem Text herausgesucht, nachdem dieser aus der Datei ausgelesen wurde, und ihre Buchstaben mit dem gleichen Verfahren wie bei der Erstellung des Wörterbuches sortiert. Das Ergebnis dieser Sortierung ist ein Schlüssel, mit dem das Programm das ursprüngliche Wort findet und an die richtige Stelle einsetzt. Auf diese Art setzt sich der enttwistete Text zusammen, der nach Abschluss dieser Routine in eine neue Datei geschrieben wird.

Dieses Verfahren findet seine Grenzen im Rahmen der teilweise nicht vorhandenen Eindeutigkeit der Schlüssel. Ohne aufwendige Sachkontextanalyse kann so nicht unterschieden werden, wenn Wörter der gleichen Buchstabenanzahl, gleichen Anfangs- und Endbuchstaben und gleichem Vorhandensein der mittleren Buchstaben, auftreten. Beispiele hierfür sind die Wörterpaare »Frucht« und »Furcht« oder »dienen« und »deinen«. Des Weiteren ist es unmöglich, ein Wort zu enttwisten, wenn es in der Wörterliste nicht vorkommt. Ein Beispiel dafür ist das Wort »Twist« selber. Aus diesem Grund sieht das Programm die Möglichkeit vor, ein neues Wörterbuch zu erstellen oder ein bestehendes zu erweitern.

2 Umsetzung

Das Programm ist in der Programmiersprache Rust in einem funktionalen Stil programmiert. Daraus ergibt sich auch ein hohes Maß an Bibliothekenverwendung. Die Funktion `main()`, die als erstes aufgerufen

wird, ist relativ übersichtlich. Zuerst lässt sie die Attribute von der Kommandozeile über die Funktion `get_command()` auslesen. Diese wirft einen Fehler, wenn die Eingaben nicht passend sind. Anschließend unterscheidet sie folgende Fälle:

1. Wird gefordert, eine Datei zu twisten, wird die Funktion `twist_file()` aufgerufen. Sie nimmt als Übergabeparameter den Namen der Input- und Outputdatei.

2. Wird gefordert, eine Datei zu enttwisten, wird die Funktion `detwist_file()` aufgerufen, die zusätzlich noch den Namen der Wörterbuchdatei übergeben bekommt.

3. Wird gefordert, ein Wörterbuch zu erstellen oder zu erweitern, folgt ein Aufruf der Funktion `extend_dictionary()`.

Die Funktion `twist_file()` arbeitet, indem sie die Funktion `read_file_to_string()` aufruft, die die Datei einliest. Das Einlesen präferiert UTF-8-Codierung, konvertiert jedoch auch Windows-1252, weil Rust intern mit UTF-8 arbeitet. Der zurückgegebenen String wird der Variable `text` gespeichert. In die Variable `twisted_text` gespeichert und in die neu erstellte Textdatei geschrieben wird der String, der von der Funktion `twist_text()` zurückgegeben wird. Sie wendet einen regulären Ausdruck an, um die Wörter aus dem Text herauszusuchen, und ersetzt jedes einzelne Wort, durch den Rückgabestring, den man erhält, wenn mit diesem die Funktion `twist_word()` aufruft. Diese Funktion überträgt den String zuerst in einen Vektor mit Zeichen. Sie gibt die Eingabe in dem Fall direkt zurück, wenn die Länge des Wortes bei drei oder weniger liegt. Über eine Stringtauschoperation wechseln im Fall der vier Buchstaben der zweite und der dritte ihre Positionen. In jedem anderen Fall wird das Durchwürfeln dadurch realisiert, dass aus dem Vektor der Bereich nach dem ersten und vor dem letzten Buchstaben herausgenommen wird, und über eine `shuffle()`-Funktion durchgewürfelt wird. Zur Kontrolle, dass das Ergebnis zufriedenstellend ist, wird geprüft, ob der erste oder zweite Buchstabe geändert wurde. Wenn das nicht der Fall ist, wird das Durchwürfeln bis zu 20 mal wiederholt.

Die Funktion `detwist_file()` ruft ebenfalls zuerst die Einleseroutine auf, um einmal die Datei mit dem getwisteten Text einzulesen, und einmal die Datei mit dem Wörterbuch. Die Funktion `create_btree_map_2()` erstellt dann mithilfe des Inhaltes des Wörterbuches einen binären Suchbaum, der als Bibliothek verwendet wird, indem sie ebenfalls über einen regulären Ausdruck die Wörter heraussucht. Anschließend wird der Schlüssel erstellt, der bei weniger als vier Buchstaben dem ursprünglichen Wort gleicht. Andernfalls wird der String in einen char-Vektor übertragen, dessen Inhalt im Bereich des zweiten und vorletzten Buchstaben sortiert wird. Dieser Vektor wird wieder in einen String übertragen und zurückgegeben. Der Suchbaum ist nach diesem Schlüssel sortiert, und enthält in jedem Knoten neben dem Schlüssel auch das ursprüngliche Wort. Im Anschluss wird die Funktion `detwist_text()` aufgerufen. Sie erhält als Übergabeparameter den Text als String und den binären Suchbaum. Aus dem String werden über den regulären Ausdruck die einzelnen Wörter herausgesucht, und eine Routine versucht, jedes Wort durch das Wort zu ersetzen, das vom Binärbaum zurückgegeben wird, wenn das getwistete Wort, über die Schlüsselroutine sortiert, als Schlüssel übergeben wird. Sollte ein Wort mit diesem Schlüssel nicht im Suchbaum vorhanden sein, bedeutet das, dass das Wort nicht im Wörterbuch stand. In diesem Fall wird das ursprüngliche Wort beibehalten.

Die Funktion `extend_dictionary()` prüft zuerst, ob ein Wörterbuch mit gleichem Namen bereits vorhanden ist. Falls das nicht der Fall ist, wird ein neues erstellt. Dazu werden aus der Datei alle Wörter über den regulären Ausdruck herausgesucht, und in die Wörterbuchdatei geschrieben, bzw. angehängt.

3 Beispiele

Bei dieser Aufgabe erscheint es besonders zweckmäßig, das Programm selber auszuprobieren, um sich von seiner Funktionstüchtigkeit zu überzeugen, da das Abdrucken eines enttwisteten Textes wenig überzeugend wirken könnte, weil die Texte bereits ungetwistet auf dem Server liegen. Um das Programm zu testen, wird über die Kommandozeile `Twist_Jan_Groeneveld.exe` eingegeben. Nach einem Leerzeichen werden nach diesem Programmnamen die Befehle eingegeben. Steht hier `twist`, so wird als nächstes Attribut der Name der Datei angegeben, und als letztes der gewünschte Name der Zieldatei. Wird dem Programm `detwist` übergeben, so erwartet das Programm als nächste Übergabeparameter zuerst den Namen der getwisteten Datei, anschließend den Namen der Datei, in die enttwistet werden soll, und der Name der Wörterbuchdatei. Eine Übergabe des Befehls `create` veranlasst das Programm dazu, aus den Wörtern der Datei, dessen Name als nächster Übergabeparameter übergeben wird, ein neues Wörterbuch zu erstellen, bzw. Wörter an ein bestehendes anzuhängen. Es sollte bei der Verwendung des Programmes beachtet werden, dass sich alle Dateien im gleichen Ordner wie das Programm befinden muss.

Als Beispiele werden jetzt die getwisteten Dateien angehängt. Hier ist die erste getwistete Datei:

```
Der Tiwst
(Ecnlsgih tiwst = Dehrung, Vdruerhneg)
war ein Mneadotz im 4/4-Tkat,
der in den ferhün 1096er Jaehrn ppläuor
wdrue und zu
Rcok'n'Rlol, Rhhytm and Belus oedr sepleiezlr
Tswit-Msuik genatzt wrid.
```

Hier ist das zweite Beispiel:

```
Hat der atle Hnemeiexster scih dcoh eamnil wbgebeegen! Und nun sleoln senie Geteisr acuh
ncah mineem Wlieln lbeen. Siene Wrot und Wrkee mrekt ich und den Baurch, und mit Gtässtsiereke
tu ich Weudnr acuh.
```

Jetzt folgt das dritte Beispiel:

```
Ein Rnseaurtat, whleecs a la ctare atieerbt, biteet sien Agboent onhe enie verhor feglsegttee
Mogeifnnhlerüee an. Drucdah heabn die Gtäse zawr mher Suaperlim bei der Whal iehrr Sepesin,
für das Rnautsreat eenhtsetn jcedoh zuehälzitcsr Awufand, da weinegr Phglriusahcsenniet
vrdonahen ist.
```

Das erhält man beispielsweise, wenn man das vierte Beispiel twistet:

```
Astuuga Ada Boryn Knig, Cnuestos of Laolcvee, war enie biistrhce Aleigde und Mtmaeiaierhtkn,
die als die estre Peoimerrgrriamn üehraubpt glit. Breites 100 Jrhae vor dem Auoefmkmn der
etrsen Pperamerrismraochgn ensarn sie enie Rheecn-Mhenaick, der eginie Ktzpoene moednerr
Pmerohiargepcarmrsn vgorenawhm.
```

Weil das letzte Beispiel sehr viel Platz einnähme, ist es hier nicht abgedruckt, es befindet sich jedoch im Ordner oder kann ohne großen Aufwand getwistet werden.

4 Quellcode

Die Funktion `main()`:

```
// Die main()-Funktion liest die Übergabeparameter aus der Kommandozeile, und twistet, enttwistet oder erweitert das Wörterbuch
fn main() -> std::io::Result<()> {
    match get_command() {
        Command::Twist(input, output) => {
            twist_file(&input, &output)?;
        }
        Command::Detwist(input, output, dictionary) => {
            detwist_file(&input, &output, &dictionary)?;
        }
        Command::CreateDictionary(input, dictionary) => {
            extend_dictionary(&input, &dictionary)?;
        }
        Command::None => {}
    }

    Ok(())
}
```

Hier folgt die Funktionen zum Twisten eines Textes:

```
// Diese Funktion lässt einen Text einlesen, ruft die Funktion zum Twisten des Textes auf, und schreibt das Ergebnis in einen neue Textdatei mit entsprechendem Namen
fn twist_file(input: &str, output: &str) -> std::io::Result<()> {
    let text = read_file_to_string(input)?;
    let twisted_text = twist_text(&text);
    let mut file = File::create(output)?;
    file.write_all(twisted_text.as_bytes())?;
    file.sync_all()?;
    Ok(())
}

// Aus dem übergebenen String werden die Wörter mithilfe eines regulären Ausdruckes herausgesucht. Diese werden dann einzeln durch das Wort ersetzt, das als Rückgabeparameter der "shuffle_word()"-Funktion zurückgegeben wird.
fn twist_text(text: &str) -> String {
    lazy_static! {
        static ref re: Regex = Regex::new(r"(\w+)").unwrap();
    }
    let output = re.replace_all(&text, |captures: &Captures| {
        let text1 = captures.get(1).map_or("", |m| m.as_str());
        //println!("{:?}", text1);
        twist_word(text1.to_string())
    });
    output.to_string()
}
```

Das sind die Funktionen, die zum Enttwisten verwendet werden:

```
// Diese Funktion lässt einen getwisteten Text einlesen, erstellt einen Binärbaum mit den Wörtern des Wörterbuches, und ruft die Funktion zum Enttwisten auf. Das Ergebnis wird in eine neue Textdatei mit entsprechendem Namen geschrieben.
fn detwist_file(input: &str, output: &str, dictionary: &str) -> std::io::Result<()> {
    let text = read_file_to_string(input)?;
    let dictionary = read_file_to_string(dictionary)?;
    let btm = create_btree_map_2(&dictionary);
    //print_btree_map_2(&btm);
    let text2 = detwist_text(&text, &btm);
    let mut file = File::create(output)?;
    file.write_all(text2.as_bytes())?;
    file.sync_all()?;
    Ok(())
}

// Hier wird ein String mit dem Inhalt des Wörterbuches übergeben, dann wird er mithilfe eines regulären Ausdruckes in die einzelnen Wörter zerlegt, die dann in den Binärbaum eingefügt werden.
```

```rust
fn create_btree_map_2(input_path: &str) -> BTreeMap<String, String> {
    lazy_static! {
        static ref re: Regex = Regex::new(r"(\w+)").unwrap();
    }

    re.find_iter(&input_path)
        .map(|mat| mat.as_str())
        .fold(BTreeMap::new(), |mut acc, word| {
            acc.entry(sort_word(word.to_string()))
                .or_insert(word.to_string());
            acc
        })
}

// Aus dem übergebenen String werden die Wörter mithilfe eines regulären Ausdruckes herausgesucht. Dann wird eine Routine aufgerufen, die jedes Wort einzeln durch das Wort ersetzt, das im Binärbaum mit dem entsprechenden Schlüssel abgespeichert ist.
fn detwist_text(text: &str, btm: &BTreeMap<String, String>) -> String {
    lazy_static! {
        static ref re: Regex = Regex::new(r"(\w+)").unwrap();
    }
    let output = re.replace_all(&text, |captures: &Captures| {
        let text1 = captures.get(1).map_or("", |m| m.as_str());
        //println!("{:?}", text1);
        let key = sort_word(text1.to_string());
        if let Some(value) = btm.get(&key) {
            /*println!(
                "FOUND: twist : {}, sorted: {}, correct: {}",
                text1, key, value
            ); */
            value
        } else {
            // Wort wurde nicht gefunden. Sollte nicht vorkommen. Das urspruengliche Wort wird beibehalten
            println!(
                "NOT FOUND: twist : {}, sorted: {}, correct: {}",
                text1, key, text1
            );
            text1
        }
    });
    output.to_string()
}

// Diese Funktion sortiert alle Buchstaben eines Wortes, außer dem Ersten und dem Letzten, nach dem Alphabet. Dadurch erhält man für jedes Wort einen Schlüssel, dem im Binärbaum das ursprüngliche Wort zugeordnet ist.
fn sort_word(word: String) -> String {
    if word.len() < 4 {
        return word;
    }
    let mut chars: Vec<char> = word.chars().collect();
    let len = chars.len();
    chars[1..len - 1].sort_by(|a, b| a.cmp(b));
    chars.into_iter().collect::<String>()
}
```