Министерство цифрового развития, связи и массовых коммуникаций

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»



Отчет по лабораторной работе №3

по дисциплине «Структура и алгоритмы обработки данных»

по теме «Методы поиска подстроки в строке»

Выполнила: студент группы

БВТ1902

Соколова А.Ю.

Проверил:

Москва

2021 г.

**Оглавление**

[Цель работы 3](#__RefHeading___Toc8473_3187879512)

[Задание 1 3](#__RefHeading___Toc8475_3187879512)

[Код программы 3](#__RefHeading___Toc8477_3187879512)

[Задание 2 «Пятнашки» 5](#__RefHeading___Toc8479_3187879512)

[Код программы 5](#__RefHeading___Toc8481_3187879512)

[Снимки экрана работы программ 13](#__RefHeading___Toc8483_3187879512)

[Вывод 14](#__RefHeading___Toc8485_3187879512)

## Цель работы

Реализовать методы поиска подстроки в строке. Добавить возможность ввода строки и подстроки с клавиатуры. Предусмотреть возможность существования пробела. Реализовать возможность выбора опции чувствительности или нечувствительности к регистру. Оценить время работы каждого алгоритма поиска.

## Задание 1

Алгоритмы:

1. Кнута-Морриса-Пратта
2. Упрощенный Бойера-Мура

## Код программы

const *MIN\_STR\_LEN*: usize = 0;  
const *MAX\_STR\_LEN*: usize = 10000;  
const *ALPHABET*: &*'static* [char] = &[  
 '1', '2', '3', '4', '5', '6', '7', '8', '9', 'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p',  
 'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', 'z', 'x', 'c', 'v', 'b', 'n', 'm', 'Q', 'W', 'E',  
 'R', 'T', 'Y', 'U', 'I', 'O', 'P', 'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'Z', 'X', 'C',  
 'V', 'B', 'N', 'M', ' ', '-', '.', ',', '?', '!', '/',  
];  
  
pub fn gen\_rand\_str\_with\_substr<R: Rng>(  
 count: usize,  
 dest: &mut Vec<(String, String)>,  
 mut rng: R,  
) {  
 for \_i in 0..count {  
 let mut buf = String::*new*();  
 for \_i in 0..(rng.gen\_range(*MIN\_STR\_LEN*..=*MAX\_STR\_LEN*)) {  
 let idx = rng.gen\_range(0..*ALPHABET*.len());  
 let ch = *ALPHABET*[idx];  
 buf.push(ch);  
 }  
  
 let sub\_str = if buf.len() > 0 {  
 let source = buf.chars().collect::<Vec<\_>>();  
 let start\_idx = rng.gen\_range(0..source.len());  
 let end\_idx = rng.gen\_range(start\_idx..=source.len());  
 source[start\_idx..end\_idx].iter().collect()  
 } else {  
 "".to\_string()  
 };  
  
 dest.push((buf, sub\_str));  
 }  
}  
  
pub fn kmp\_prefix\_function(s: &str) -> Vec<usize> {  
 let chars = s.as\_bytes();  
 let mut prefix\_func = Vec::*with\_capacity*(chars.len());  
 prefix\_func.push(0);  
  
 for current in 1..chars.len() {  
 let mut matched\_prefix = current - 1;  
 let mut candidate = prefix\_func[matched\_prefix];  
 while candidate != 0 && chars[current] != chars[candidate] {  
 matched\_prefix = prefix\_func[matched\_prefix] - 1;  
 candidate = prefix\_func[matched\_prefix];  
 }  
 if candidate == 0 {  
 let to\_push = if chars[current] == chars[0] { 1 } else { 0 };  
 prefix\_func.push(to\_push);  
 } else {  
 prefix\_func.push(candidate + 1);  
 }  
 }  
  
 prefix\_func  
}  
// Кнута-Морриса-Пратта  
pub fn kmp\_find(source: &str, pattern: &str) -> Option<usize> {  
 let pattern\_len = pattern.len();  
 if pattern.len() > source.len() {  
 *None*  
} else {  
 let union = pattern.to\_string().add("$").add(source);  
 let prefixes = kmp\_prefix\_function(&union);  
 prefixes.iter().enumerate().find\_map(|(idx, c)| {  
 *//println!("> {}", \*c);*  
if \*c == pattern\_len {  
 *Some*(idx - 2 \* pattern\_len)  
 } else {  
 *None*  
}  
 })  
 }  
}  
// Бойера Мура  
pub fn bm\_find(source: &str, template: &str) -> Option<usize> {  
 if template.len() == 0 {  
 return *Some*(0);  
 }  
 if template.len() > source.len() {  
 return *None*;  
 }  
  
 let source = source.as\_bytes();  
 let template = template.as\_bytes();  
 let source\_len = source.len();  
 let template\_len = template.len();  
  
 let mut offset\_table = Box::*new*([template\_len; 256]);  
  
 for (i, char) in template.iter().take(template\_len - 1).enumerate() {  
 offset\_table[\*char as usize] = template\_len - i - 1;  
 }  
  
 let mut i = template\_len - 1;  
 while i < source\_len {  
 let start = i + 1 - template\_len;  
 let matched = (0..template\_len).all(|j| source[start + j] == template[j]);  
 if matched {  
 return *Some*(i + 1 - template\_len);  
 }  
 *//println!("i = {}", i);*  
i += offset\_table[source[i] as usize];  
 }  
 return *None*;  
}

## Задание 2 «Пятнашки»

Написать программу, определяющую, является ли данное расположение «решаемым», то есть можно ли из него за конечное число шагов перейти к правильному. Если это возможно, то необходимо найти хотя бы одно решение - последовательность движений, после которой числа будут расположены в правильном порядке.

## Код программы

use ndarray::Array2;  
use rand::Rng;  
use std::borrow::Borrow;  
use std::cmp::Ordering;  
use std::collections::{BinaryHeap, HashMap, HashSet, VecDeque};  
use std::fmt::{Display, Formatter};  
use std::rc::Rc;  
  
pub const *WIDTH*: usize = 4;  
pub const *HEIGHT*: usize = 4;  
  
#[derive(Debug, Clone, Copy, Hash, Eq, PartialEq)]  
pub struct Board([[u8; *WIDTH*]; *HEIGHT*]);  
  
#[derive(Debug, Copy, Clone, Default, Eq, PartialEq)]  
pub struct BoardCreateError;  
  
impl Board {  
 pub fn *new*() -> Self {  
 let mut arr = [[0u8; *WIDTH*]; *HEIGHT*];  
 for y in 0..*WIDTH* {  
 for x in 0..*HEIGHT* {  
 arr[y][x] = ((y \* *WIDTH* + x + 1) % (*WIDTH* \* *HEIGHT*)) as u8  
 }  
 }  
 Board(arr)  
 }  
  
 pub fn *from\_array*(arr: [[u8; *WIDTH*]; *HEIGHT*]) -> Result<Self, BoardCreateError> {  
 let w = *WIDTH*;  
 let h = *HEIGHT*;  
 let mut tile\_count = vec![0; w \* h];  
 for y in 0..*HEIGHT* {  
 for x in 0..*WIDTH* {  
 tile\_count.get\_mut(arr[y][x] as usize).map(|x| \*x += 1);  
 }  
 }  
 let has\_one\_of\_all = tile\_count.iter().all(|x| \*x == 1);  
 if has\_one\_of\_all {  
 *Ok*(Board(arr))  
 } else {  
 *Err*(BoardCreateError)  
 }  
 }  
  
 pub fn size(&self) -> (usize, usize) {  
 (*WIDTH*, *HEIGHT*)  
 }  
  
 pub fn empty\_at(&self) -> (usize, usize) {  
 for y in 0..*HEIGHT* {  
 for x in 0..*WIDTH* {  
 if self.0[y][x] == 0 {  
 return (x, y);  
 }  
 }  
 }  
 panic!()  
 }  
  
 #[inline(always)]  
 pub fn swap(&mut self, p1: (usize, usize), p2: (usize, usize)) {  
 let arr = &mut self.0;  
 let t1 = arr[p1.1][p1.0];  
 let t2 = arr[p2.1][p2.0];  
 arr[p1.1][p1.0] = t2;  
 arr[p2.1][p2.0] = t1;  
 }  
  
 pub fn apply(&mut self, dir: Dir) -> Result<(), ()> {  
 let (zx, zy) = self.empty\_at();  
 let (w, h) = self.size();  
 match dir {  
 Dir::*Right* if zx < w - 1 => {  
 self.swap((zx, zy), (zx + 1, zy));  
 *Ok*(())  
 }  
 Dir::*Down* if zy < h - 1 => {  
 self.swap((zx, zy), (zx, zy + 1));  
 *Ok*(())  
 }  
 Dir::*Left* if zx > 0 => {  
 self.swap((zx, zy), (zx - 1, zy));  
 *Ok*(())  
 }  
 Dir::*Up* if zy > 0 => {  
 self.swap((zx, zy), (zx, zy - 1));  
 *Ok*(())  
 }  
 \_ => *Err*(()),  
 }  
 }  
  
 pub fn is\_solved(&self) -> bool {  
 let (w, h) = self.size();  
 for y in 0..h {  
 for x in 0..w {  
 if ((y \* w + x + 1) % (w \* h)) as u8 != self.0[y][x] {  
 return false;  
 }  
 }  
 }  
 true  
 }  
  
 pub fn can\_solve(&self) -> bool {  
 let mut arr = [0u8; *WIDTH* \* *HEIGHT*];  
 let s = *WIDTH* \* *HEIGHT*;  
 for y in 0..*HEIGHT* {  
 for x in 0..*WIDTH* {  
 arr[y \* *WIDTH* + x] = self.0[y][x];  
 }  
 }  
  
 let mut inv = 0;  
 for i in 0..s {  
 if arr[i] > 0 {  
 for j in 0..i {  
 if arr[j] > arr[i] {  
 inv += 1;  
 }  
 }  
 }  
 }  
 for i in 0..s {  
 if arr[i] == 0 {  
 inv += 1 + i / *WIDTH*;  
 }  
 }  
  
 inv % 2 == 0  
 }  
  
 pub fn wrong\_tiles(&self) -> usize {  
 let (w, h) = self.size();  
 let mut c = 0;  
 for y in 0..h {  
 for x in 0..w {  
 if ((y \* w + x + 1) % (w \* h)) as u8 != self.0[y][x] {  
 c += 1;  
 }  
 }  
 }  
 c  
 }  
  
 pub fn solve(&self) -> Result<Rc<Path>, ()> {  
 if !self.can\_solve() {  
 return *Err*(());  
 }  
  
 let mut checked\_position\_length = HashMap::*new*();  
 let mut heap = BinaryHeap::*with\_capacity*(1000);  
  
 heap.push(QPath(Rc::*new*(Path::*new*(self.clone()))));  
  
 let mut i = 0;  
 loop {  
 i += 1;  
 let current = {  
 if heap.len() < 1\_000\_000 {  
 heap.pop().unwrap()  
 } else {  
 let x = heap.pop().unwrap();  
 heap.clear();  
 x  
 }  
 };  
 let last = checked\_position\_length.get\_mut(&current.0.current\_board);  
  
 if i % 1\_00\_000 == 0 {  
 println!(  
 "iter = {}M, path len = {}, euristic = {}, in heap {} el",  
 i / 1\_000\_000,  
 current.0.len(),  
 current.cost(),  
 heap.len()  
 );  
 println!("{}", current.0.current\_board());  
 }  
  
 match last {  
 *Some*(last) if \*last <= current.0.len() => continue,  
 *Some*(last) => {  
 \*last = current.0.len();  
 *//remove\_longer(&mut heap, current.0.current\_board);*  
}  
 \_ => {  
 checked\_position\_length.insert(current.0.current\_board, current.0.len());  
 *//remove\_longer(&mut heap, current.0.current\_board);*  
}  
 }  
  
 *// println!("Current board with {}", current.cost());*  
if current.0.current\_board().is\_solved() {  
 return *Ok*(current.0);  
 }  
 let mut push\_or\_ignore = |dir| {  
 *// Oh... Remove?*  
 *// if heap.len() > 1\_000\_000 {*  
 *// let mut replacement = BinaryHeap::with\_capacity(1\_000\_005);*  
 *// for \_i in 0..10\_000 {*  
 *// replacement.push(heap.pop().unwrap());*  
 *// }*  
 *// heap = replacement;*  
 *// }*  
 *// ^^^^^^^*  
let mut c = &current;  
 let path = c.0.add\_step(dir);  
 if let *Ok*(path) = path {  
 if !checked\_position\_length.contains\_key(path.current\_board()) {  
 let path = Rc::*new*(path);  
 heap.push(QPath::*new*(path));  
 }  
 }  
 }; *// 15 2 1 12 8 5 6 11 4 9 10 7 3 14 13 0*  
 *// 4 2 1 12 8 3 15 7 9 6 5 11 14 10 13 0*  
push\_or\_ignore(Dir::*Up*);  
 push\_or\_ignore(Dir::*Right*);  
 push\_or\_ignore(Dir::*Down*);  
 push\_or\_ignore(Dir::*Left*);  
 }  
 }  
  
 pub fn inner(&self) -> &[[u8; *WIDTH*]; *HEIGHT*] {  
 &self.0  
 }  
}  
  
impl Display for Board {  
 fn fmt(&self, f: &mut Formatter<*'\_*>) -> std::fmt::Result {  
 let (w, h) = self.size();  
 for y in 0..h {  
 for x in 0..w {  
 match w \* h {  
 0..=9 => write!(f, "{:1} ", self.0[y][x])?,  
 10..=99 => write!(f, "{:2} ", self.0[y][x])?,  
 100..=999 => write!(f, "{:3} ", self.0[y][x])?,  
 \_ => panic!(""),  
 };  
 }  
 writeln!(f)?;  
 }  
 *Ok*(())  
 }  
}  
  
#[derive(Copy, Clone, Eq, PartialEq, Debug, Hash)]  
pub enum Dir {  
 *Up*,  
 *Right*,  
 *Down*,  
 *Left*,  
}  
  
#[derive(Clone, Debug, Hash, Eq, PartialEq)]  
pub struct Path {  
 pub current\_board: Board,  
 pub len: usize,  
 pub step: Option<(Rc<Path>, Dir)>,  
}  
  
impl Path {  
 pub fn current\_board(&self) -> &Board {  
 &self.current\_board  
 }  
 pub fn step(&self) -> &Option<(Rc<Path>, Dir)> {  
 &self.step  
 }  
 pub fn len(&self) -> usize {  
 self.len  
 }  
}  
  
impl Path {  
 pub fn *new*(start\_board: Board) -> Self {  
 Self {  
 current\_board: start\_board,  
 len: 0,  
 step: *None*,  
 }  
 }  
  
 pub fn add\_step(self: &Rc<Self>, dir: Dir) -> Result<Path, ()> {  
 let mut new\_board = self.current\_board.clone();  
 new\_board.apply(dir)?;  
 *Ok*(Self {  
 current\_board: new\_board,  
 len: self.len + 1,  
 step: *Some*((Rc::clone(self), dir)),  
 })  
 }  
}  
  
#[derive(Clone)]  
struct QPath(Rc<Path>);  
impl QPath {  
 fn *new*(p: Rc<Path>) -> Self {  
 Self(p)  
 }  
 pub fn cost(&self) -> usize {  
 let g = self.0.len();  
 let m = |x: u8| if x == 0 { 15 } else { (x-1) as usize };  
 let f = {  
 let mut t\_cost = 0;  
 for y in 0..4 {  
 for x in 0..3 {  
 let current = m(self.0.current\_board.0[y][x]);  
 if current != 15 && current >= 4\*y && current <= 4\*(y + 1) {  
 for x1 in (x + 1)..4 {  
 if m(self.0.current\_board.0[y][x1]) < current {  
 t\_cost += 2;  
 }  
 }  
 }  
 }  
 }  
 for x in 0..4 {  
 for y in 0..3 {  
 let current = m(self.0.current\_board.0[y][x]);  
 if current != 15 && (current % 4) == x {  
 for y1 in (y + 1)..4 {  
 if m(self.0.current\_board.0[y1][x]) < current {  
 t\_cost += 2;  
 }  
 }  
 }  
 }  
 }  
 *// Манхеттанское расстояние*  
for x in 0..4 {  
 for y in 0..4 {  
 let current = m(self.0.current\_board.0[y][x]);  
 if current != 15 {  
 let ox = current % 4;  
 let oy = current / 4;  
 t\_cost += x.max(ox) - x.min(ox);  
 t\_cost += y.max(oy) - x.min(oy);  
 }  
 }  
 }  
 t\_cost*// + self.0.current\_board.wrong\_tiles()*  
};  
 g + f  
 }  
}  
impl Ord for QPath {  
 fn cmp(&self, other: &Self) -> Ordering {  
 (other.cost()).cmp(&self.cost())  
 }  
}  
impl PartialOrd for QPath {  
 fn partial\_cmp(&self, other: &Self) -> Option<Ordering> {  
 *Some*(self.cmp(other))  
 }  
}  
impl PartialEq for QPath {  
 fn eq(&self, other: &Self) -> bool {  
 self.cmp(other) == Ordering::*Equal*  
}  
}  
impl Eq for QPath {}

## Снимки экрана работы программ

Задача 1

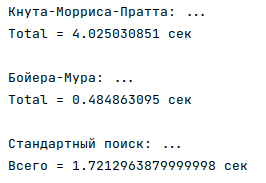


Рисунок 1 – Замер скорости алгоритмов поиска подстроки

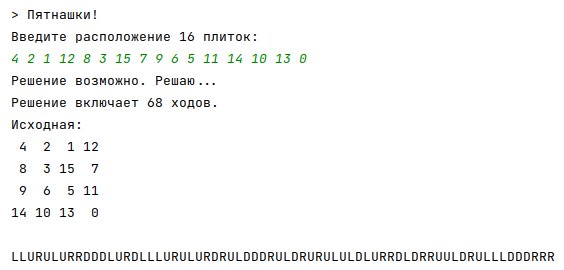


Рисунок 2 – Результат решения пятнашек без вывод поля каждый ход

## Вывод

Я реализовал методы поиска подстроки в строке, предусмотрел возможность существования пробела, и оценил время работы каждого алгоритма поиска. А так же написал алгоритм решения «пятнашек» с использование эвристики Линейный конфликт + Манхэттенское расстояние, способный выдать решение с малым числом ходов.