浅谈Rust在算法题和竞赛中的应用

吴翱翔

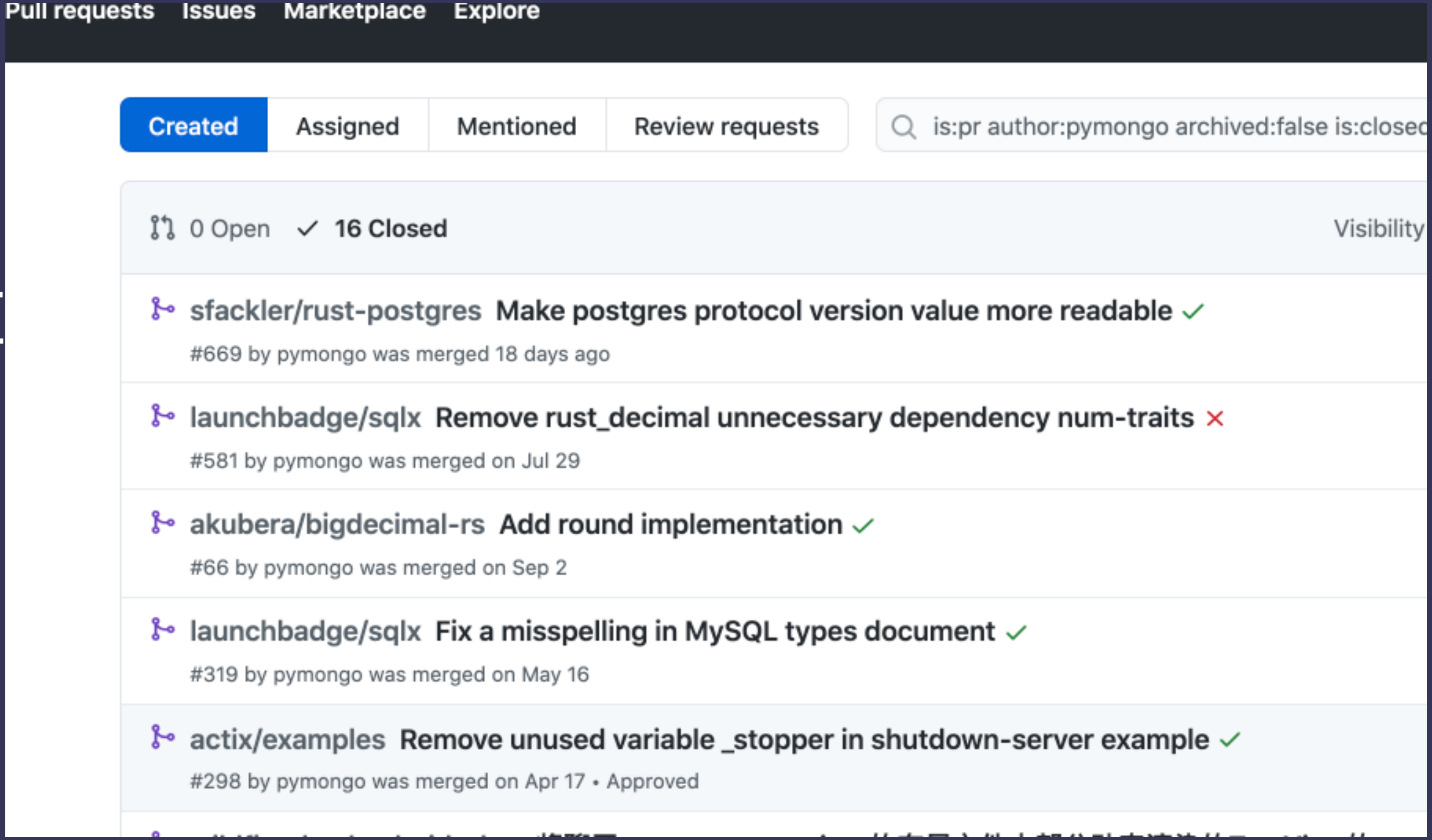
自我介绍

曾给rust-postgres、sqlx

bigdecimal-rs、actix等库
贡献过代码

leetcode刷题量400+

leetcode/codeforces周赛



Pull requests Issues Marketplace Explore

Created | Assigned | Mentioned | Review requests

is:pr author:pymongo archived:false is:closed

0 Open ✓ 16 Closed Visibility

sfackler/rust-postgres Make postgres protocol version value more readable ✓
#669 by pymongo was merged 18 days ago

launchbadge/sqlx Remove rust_decimal unnecessary dependency num-traits ✗
#581 by pymongo was merged on Jul 29

akubera/bigdecimal-rs Add round implementation ✓
#66 by pymongo was merged on Sep 2

launchbadge/sqlx Fix a misspelling in MySQL types document ✓
#319 by pymongo was merged on May 16

actix/examples Remove unused variable _stopper in shutdown-server example ✓
#298 by pymongo was merged on Apr 17 • Approved

github.com/pymongo/leetcode-rust

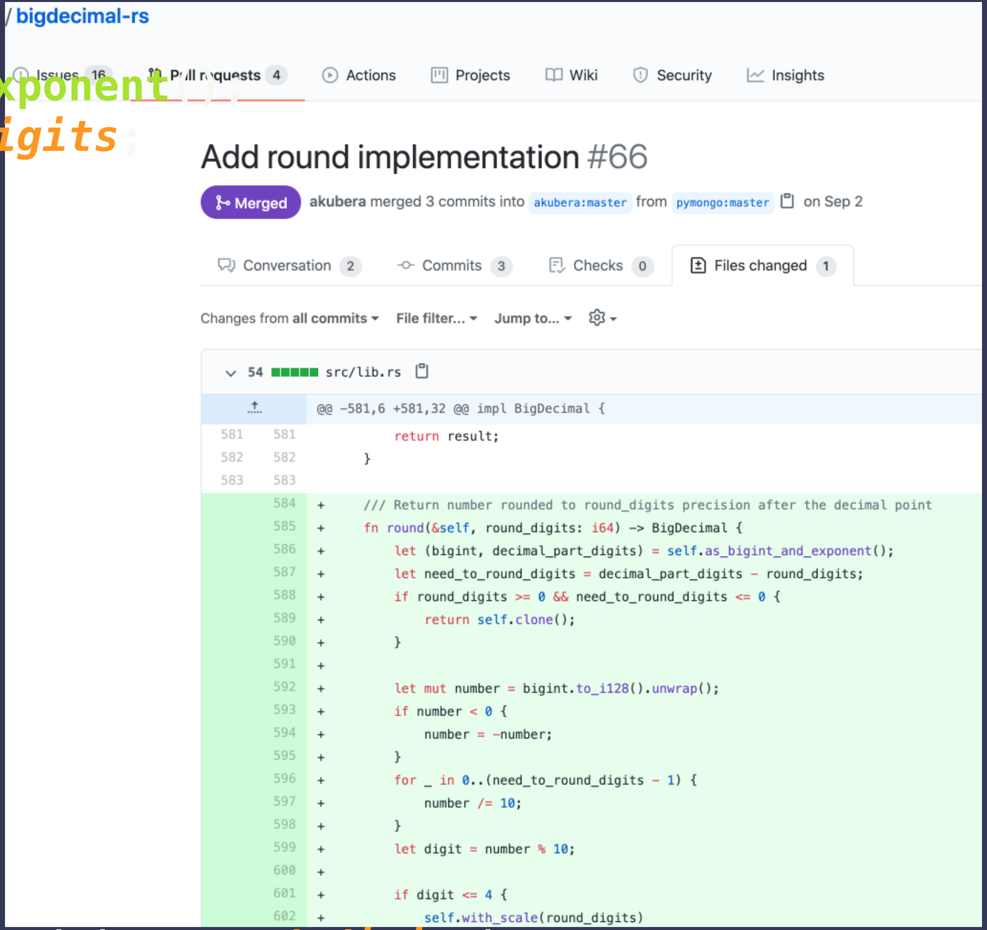
为什么要用Rust刷题?

- 刷题是学习和练习Rust的一个途径
- 掌握标准库API: peekable, saturating_sub等
- 通过链表/二叉树题理解Box, Rc, RefCell等智能指针
- Rust性能优秀，leetcode运行时间容易跑进0ms
- 内存利用率高，能像C/C++一样原地修改字符串
- 可以调用C语言函数或汇编指令解题
- Rust工具链好用，单元测试强大(例如assert_eq两个链表)

我为Bigdecimal实现的round API

```rust
pub fn round(&self, round_digits: i64) -> Self {
    let (bigint, decimal_part_digits) = self.as_bigint_and_exponent();
    let need_to_round_digits = decimal_part_digits - round_digits;
    if round_digits >= 0 && need_to_round_digits <= 0 {
        return self.clone();
    }
    let mut number = bigint.to_i128().unwrap();
    if number < 0 {
        number = -number;
    }
    for _ in 0..(need_to_round_digits - 1) {
        number /= 10;
    }
    let digit = number % 10;

    if digit <= 4 {
        self.with_scale(round_digits)
    } else if bigint.is_negative() {
        self.with_scale(round_digits) - Self::new(BigInt::from(1), round_digits)
    } else {
        self.with_scale(round_digits) + Self::new(BigInt::from(1), round_digits)
    }
}
```



/ bigdecimal-rs

Issues 16 · Pull requests 4 · Actions · Projects · Wiki · Security · Insights

Add round implementation #66

Merged akubera merged 3 commits into `akubera:master` from `pymongo:master` on Sep 2

Conversation 2 · Commits 3 · Checks 0 · Files changed 1

Changes from all commits ▾ File filter... ▾ Jump to... ▾

54 🟩🟩🟩🟩🟩 src/lib.rs

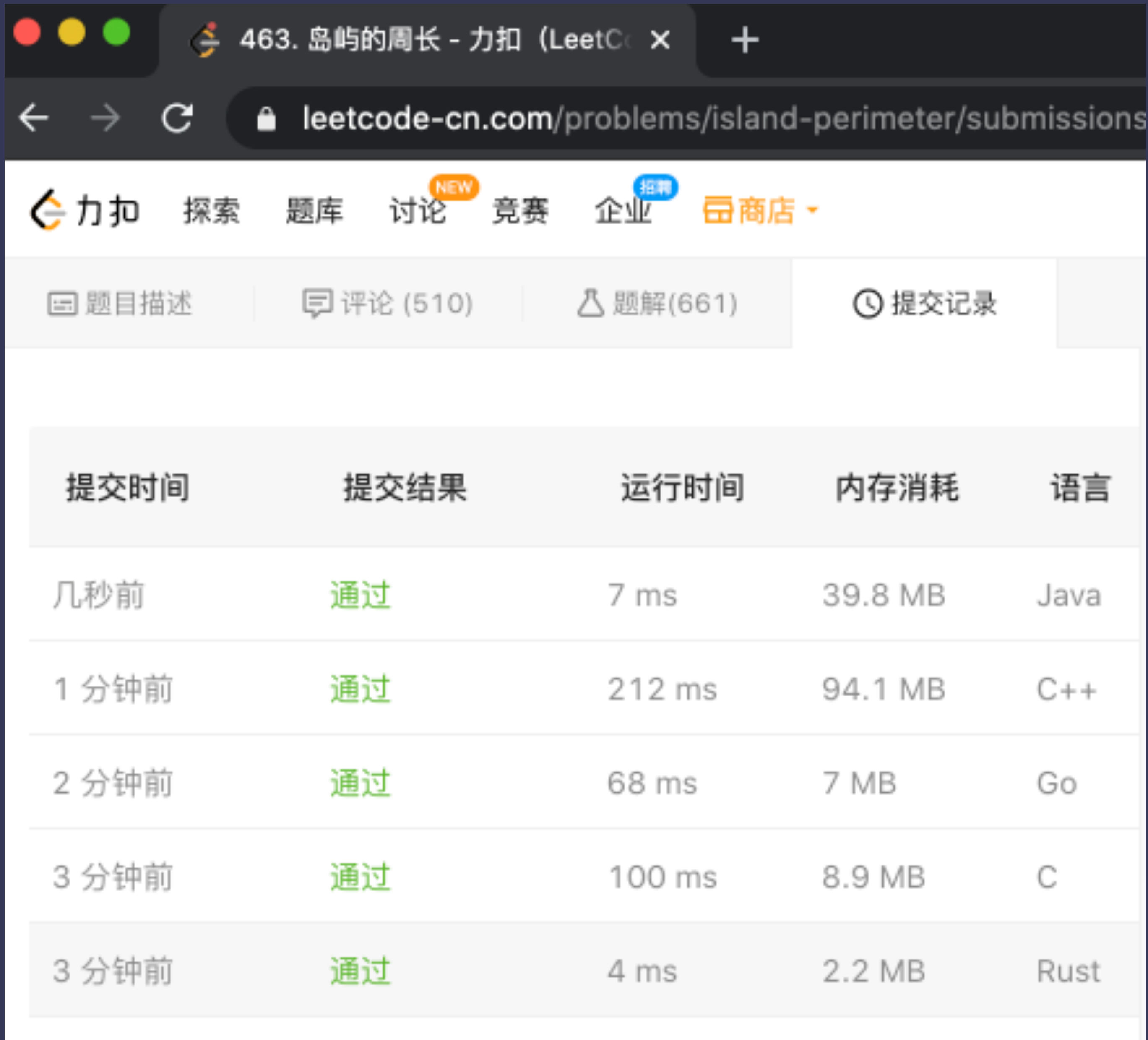
```diff
@@ -581,6 +581,32 @@ impl BigDecimal {
             return result;
         }
 
+    /// Return number rounded to round_digits precision after the decimal point
+    fn round(&self, round_digits: i64) -> BigDecimal {
+        let (bigint, decimal_part_digits) = self.as_bigint_and_exponent();
+        let need_to_round_digits = decimal_part_digits - round_digits;
+        if round_digits >= 0 && need_to_round_digits <= 0 {
+            return self.clone();
+        }
+
+        let mut number = bigint.to_i128().unwrap();
+        if number < 0 {
+            number = -number;
+        }
+        for _ in 0..(need_to_round_digits - 1) {
+            number /= 10;
+        }
+        let digit = number % 10;
+
+        if digit <= 4 {
+            self.with_scale(round_digits)
```

Rust性能适合刷题

leetcode_463. 岛屿周长性能对比

在leetcode运行环境中，
Rust的性能是最好的
(性能对比仅供参考)

在codeforces中
Rust会比C语言稍微慢点


463. 岛屿的周长 - 力扣（LeetC
leetcode-cn.com/problems/island-perimeter/submissions
力扣
探索
题库
讨论
NEW
竞赛
企业
商店
题目描述
评论 (510)
题解(661)
提交记录

提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	7 ms	39.8 MB	Java
1 分钟前	通过	212 ms	94.1 MB	C++
2 分钟前	通过	68 ms	7 MB	Go
3 分钟前	通过	100 ms	8.9 MB	C
3 分钟前	通过	4 ms	2.2 MB	Rust

Rust代码简洁和可读性强

官方解法

§ leetcode_3

这里的7和-8有点魔法数，

```
int reverse(int x) {
    int rev = 0;
    while (x != 0) {
        int pop = x % 10;
        x /= 10;
        if (rev > INT_MAX/10 || (rev == INT_MAX/10 && pop > 7)) return 0;
        if (rev < INT_MIN/10 || (rev == INT_MIN/10 && pop < -8)) return 0;
        rev = rev * 10 + pop;
    }
    return rev;
}
```

Rust解法

Option表达了None时整数x
不能被反转(反转后会溢出)的语义

```
fn reverse_integer(x: i32) -> i32 {
    fn helper(mut n: i32) -> Option<i32> {
        let mut ret = 0i32;
        while n.abs() != 0 {
            ret = ret.checked_mul(10)?.checked_add(n % 10)?;
            n /= 10;
        }
        Some(ret)
    }
    helper(x).unwrap_or_default()
}
```

Rust代码简洁和可读性强

§ leetcode_189

```
fn rotate_array(nums: &mut Vec<i32>, k: i32) {
    let len = nums.len();
    nums.rotate_left(k as usize % len);
}
```

§ leetcode_65
(这题的正统解法是DFA有限状态机)

```
fn is_number(s: String) -> bool {
    s.trim().parse::<f32>().is_ok()
}
```

§ leetcode_468

```
fn valid_ip_address(ip: String) -> String {
    match ip.parse::<std::net::IpAddr>() {
        Ok(std::net::IpAddr::V4(_)) => "IPv4",
        Ok(std::net::IpAddr::V6(_)) => "IPv6",
        Err(_) => "Neither"
    }.to_string()
}
```

§ leetcode_1486

```
fn xor_operation(n: i32, start: i32) -> i32 {
    (start..)
        .step_by(2)
        .take(n as usize)
        .fold(0, |a, b| a ^ b)
}
```

Rust调用C语言函数解题

```rust
extern "C" {
    fn rand() -> i32;
}

struct RandomPickIndex {
    nums: Vec<i32>,
}
```

§ leetcode_398
等概率随机返回nums[i]=target的下标i

codeforces中Rust不能用任何第三方库
但是Rust可以调用C语言
标准库函数实现随机数或正则表达式等功能

```rust
impl RandomPickIndex {
    fn pick(&mut self, target: i32) -> i32 {
        let mut count = 0i32;
        let mut ret = 0usize;
        for (i, num) in self.nums.iter().enumerate()
            if target.ne(num) {
                continue;
            }
            count += 1;
            if unsafe { rand() } % count == 0 {
                ret = i;
            }
        }
        ret as i32
    }
}
```

Rust还能调用汇编语言解题

```
n.count_ones() as i32

fn hanming_weight(n: u32) -> i32 {
    let mut count = 0;
    let mut mask = 0b1;
    for _ in 0..32 {
        if n & mask == 1 {
            count += 1;
        }
        mask <<= 1;
    }
    count
}

fn hanming_weight(mut n: u32) -> i32 {
    let mut count = 0;
    while n != 0 {
        n &= n - 1;
        count += 1;
    }
    count
}
```

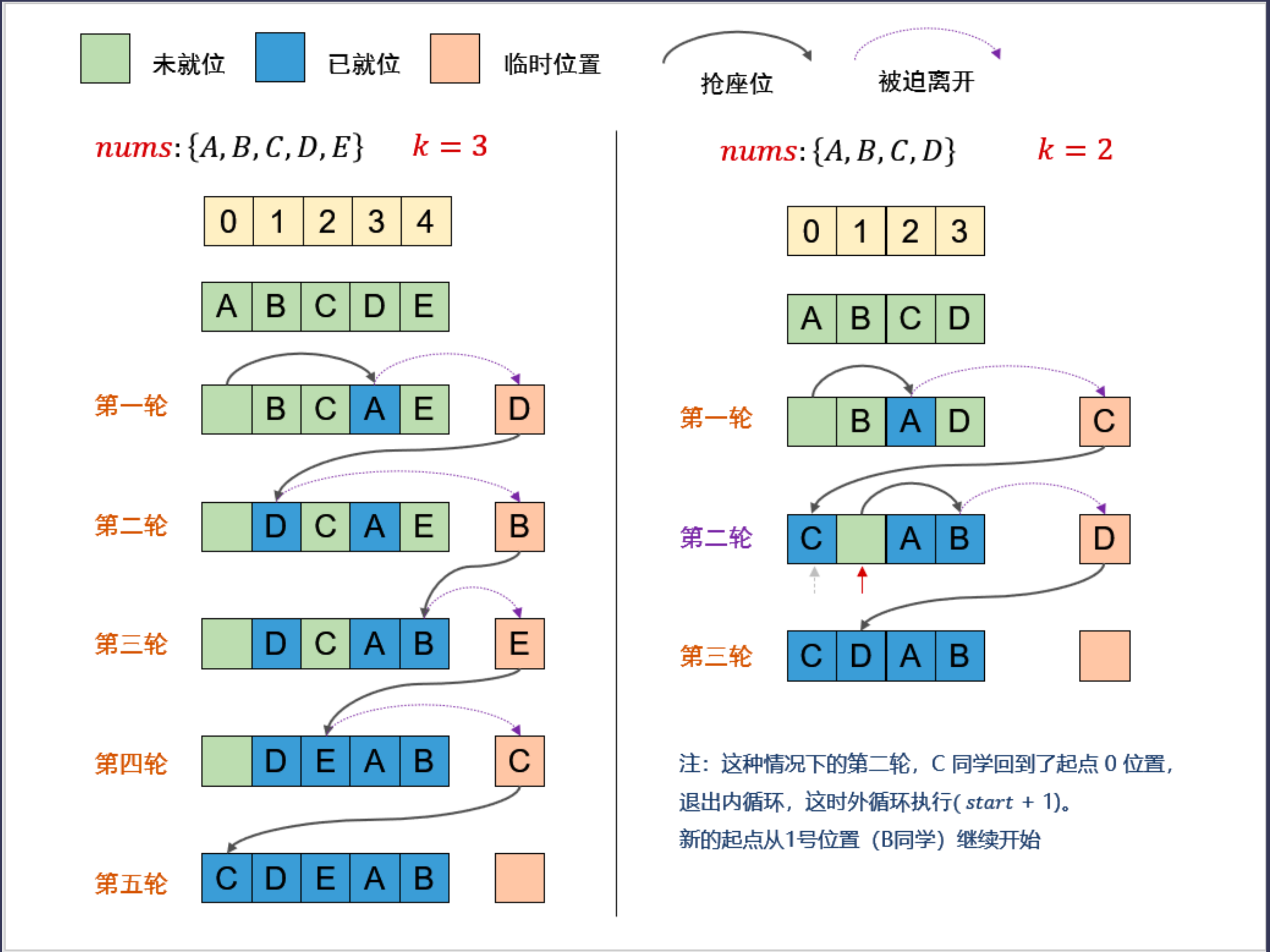
§ leetcode_191:
求出一个正整数的二进制表示中有几个1，也叫汉明权重
除左边三种常规解法，Rust还可以用内联汇编解题

```
fn hamming_weight(n: u32) -> i32 {
    let popcnt_input: usize = n as usize;
    let popcnt_output: usize;
    unsafe {
        asm!(
            "popcnt {popcnt_output}, {popcnt_input}",
            popcnt_input = in(reg) popcnt_input,
            popcnt_output = out(reg) popcnt_output,
        );
    }
    popcnt_output as i32
}
```

Rust原地修改字符串

§ leetcode_1528:
字符串各字母按新索引表重排

Rust可以像C/C++一样
原地(In-Place)修改字符串
所以能实现空间复杂度O(1)的解法



注：这种情况下的第二轮，C 同学回到了起点 0 位置，
退出内循环，这时外循环执行($start + 1$)。
新的起点从1号位置（B同学）继续开始

Rust原地修改字符串示例

```rust
fn shuffle_string(s: String, indices: Vec<i32>) -> String {
    let mut s = s.into_bytes();
    let mut indices: Vec<usize> = indices.into_iter().map(|x| x as usize).collect();
    let len = s.len();
    for i in 0..len {
        if indices[i] != i {
            let mut ch = s[i];
            let mut ch_correct_idx = indices[i];
            while ch_correct_idx != i {
                std::mem::swap(&mut s[ch_correct_idx], &mut ch);
                std::mem::swap(&mut indices[ch_correct_idx], &mut ch_correct_idx);
            }
            s[i] = ch;
            indices[i] = i;
        }
    }
    unsafe { String::from_utf8_unchecked(s) }
}
```

案例分析: number_of_good_pairs

§ leetcode_1512: Number Of Good Pairs
题目描述: 统计输入的数组内总共有几对下标(i,j)，满足: i<j and nums[i]==nums[j]
思路: 统计数组内每个值的出现次数，例如1出现了3次，那么新增排列组合3*2/2对

```
return sum(map(lambda val: math.comb(val, 2), collections.Counter(nums).values()))
```

这是Rust类似Python Counter的代码实现

```
fn number_of_good_pairs_v1(nums: Vec<i32>) -> i32 {
    let mut counter = std::collections::HashMap::new();
    for &num in nums.iter() {
        *counter.entry(num).or_insert(0) += 1;
    }
    counter.iter().map(|(_k, &v)| (v - 1) * v / 2).sum()
}
```

既然数组长度已知，用with_capacity
创建HashMap能提高内存分配效率

只用了一次，应该换成into_iter consume/move掉nums和counter

优化思路: number_of_good_pairs

用into_iter()和with_capacity()优化后:

```rust
fn number_of_good_pairs_v2(nums: Vec<i32>) -> i32 {
    let mut counter = std::collections::HashMap::with_capacity(nums.len());
    for num in nums.into_iter() {
        *counter.entry(num).or_insert(0) += 1;
    }
    counter.into_iter().map(|(_k, v)| (v - 1) * v / 2).sum()
}
```

那么还能不能继续优化下去呢? 再看看题目提示:

1 <= nums.length <= 100

1 <= nums[i] <= 100

优化思路一: nums长度范围是1..=100意味着可以用u8类型进行计数

优化思路二: 既然数组中值的范围是固定的, 可以用固定长数组作为counter

例如ASCII编码的字符串可以用长度256的数组作为counter

继续优化: number_of_good_pairs

counter的次数从编译器默认的i32改成u8，u8占据内存空间比i32更小，减少了空间复杂度

但是HashMap的value是u8的话，(v-1)*v/2会有两个溢出问题:

1. v-1当v=0时会向下溢出成负数，v-1要改成v.saturating_sub(1)
2. (V-1)*v两个u8相乘可能会超过255

```rust
fn number_of_good_pairs_v3(nums: Vec<i32>) -> i32 {
    let mut counter = std::collections::HashMap::with_capacity(nums.len());
    for num in nums.into_iter() {
        *counter.entry(num).or_insert(0u8) += 1;
    }
    counter
        .into_iter()
        .map(|(_k, v)| (v as i32 - 1) * v as i32 / 2)
        .sum::<i32>()
}
```

性能测试: number_of_good_pairs

counter的数据类型HashMap可以继续优化为固定长array

Rust的工具链提供了bench性能测试工具进行测试

```rust
#[bench]
fn bench_fixed_len_array_counter(bencher: &mut test::Bencher) {
    bencher.iter(|| {
        let mut counter = [0u8; 101];
        for &num in NUMS.iter() {
            counter[num as usize] += 1;
        }
        let _ret = counter
            .iter()
            .map(|&v| (v as i32 - 1) * v as i32 / 2)
            .sum::<i32>();
    });
}
#[bench]
fn bench_hashmap_counter(bencher: &mut test::Bencher) {
    // 和上页PPT的代码相同，省略
}
```

benchmark测试结果

```
test bench_fixed_len_array_counter
... bench: 51 ns/iter (+/- 10)
test bench_hashmap_counter
... bench: 1,772 ns/iter (+/- 158)
```

使用固定长的数组作为Counter性能更优

Rust stdin/stdout的单元测试与codeforces

§ codeforce介绍

leetcode是给定了函数签名(入参和返回值)，再完成函数的实现代码

codeforces默认通过上传代码文件判题，类似ACM竞赛从stdin中读输入数据，stdout输出

§ 为什么需要stdin/stdout级别的单元测试

- 不同操作系统的换行符不同，手工往stdin输入数据会与codeforces环境的换行符不同
- 方便重构优化题解，可以基于函数去管理题解不需要一个题解一个binary/executable file
- 单元测试方便本地调试代码

stdin/stdout的单元测试示例

§ codeforce_71a:
英语中有一种将很长的单词缩写方法是: 首字母+中间有几个字母+尾字母
例如 kubernetes 开头k结尾s，k和s中间有8个字母，所以缩写成k8s
例如 internationalization 开头i结尾n中间18个字母，缩写成i18n
题目规定字母数>=10的单词为长单词，需要进行缩写
题目要求将输入的单词数组进行缩写处理，返回缩写后的结果

输入示例:

```
2
kubernetes
internationalization
```

输出示例:

```
2
k8s
i18n
```

stdin/stdout的单元测试示例

```
fn solution(
    reader: impl std::io::BufRead,
    mut writer: impl std::io::Write,
) -> Result<(), Box<dyn std::error::Error>> {
    let mut input: Vec<String> = Vec::new();
    for line in reader.lines() {
        if let Ok(str) = line {
            input.push(str);
        }
    }
    for string in input.into_iter().skip(1) {
        let len = string.len();
        if len <= 10 {
            writeln!(&mut writer, "{}", string)?;
        } else {
            let bytes = string.into_bytes();
            writeln!(
                &mut writer,
                "{}{}{}",
                bytes[0] as char,
                len - 2, // len - 2(first and last)
                *bytes.last().unwrap() as char
            )?;
        }
```

```
fn main() {
    solution(
        std::io::stdin().lock(),
        std::io::stdout().lock()
    ).unwrap();
}

#[test]
fn test_solution() {
    const TEST_CASES: [(&[u8], &[u8]); 1] = [(
        b"2\kubernetes\internationalization\n",
        b”k8s\ni18\n",
    )];
    for &(input, expected) in &TEST_CASES {
        let mut output = Vec::new();
        solution(input, &mut output).unwrap();
        assert_eq!(output, expected);
    }
}
```

clippy工具改善Rust代码

$ leetcode_231

```
fn is_power_of_two(n: i32) -> bool {
    if n == 0 {
        return false;
    }
    n & (n - 1) == 0
}
```

这个判断浮点数是不是整数的函数
编译通过，但是大家想想会有什么问题?

```
#[test]
fn test_is_power_of_two() {
    fn is_integer(float: f32) -> bool {
        float == float.round()
    }
    extern "C" {
        fn time(time: *mut isize) -> isize;
        fn rand() -> i32;
        fn srand(seed: u32);
    }
    unsafe {
        srand(time(&mut std::mem::zeroed()) as u32);
    }
    for _ in 0..100_000 {
        let input = unsafe { rand() };
        let expected = is_integer((input as f32).log2())
        assert_eq!(is_power_of_two(input), expected);
    }
}
```

clippy工具改善Rust代码

```
fn is_integer(float: f32) -> bool {
    float == float.round()
}
```

```
fn is_integer(float: f32) -> bool {
    (float - float.round()).abs() < f32::EPSILON
}
```

```
error: strict comparison of `f32` or `f64`
   = note: `#[deny(clippy::float_cmp)]` on by default
   = note: `f32::EPSILON` and `f64::EPSILON` are available for the `error_margin`
```

有兴趣的同学可以思考另一个浮点数相关的问题:

为什么所有语言BigDecimal的都不建议用浮点数进行构造?

例如以下调用了Decimal的float构造方法的Python代码为什么会计算出错?

```
>>> from decimal import Decimal
>>> Decimal(1.07) * Decimal(2.103)
```

当前Rust在算法题或竞赛上的不足

- 大部分竞赛不支持Rust
- leetcode对Rust的支持有限(例如不支持N叉树题型)
- 难以删除或挪动链表中间几个节点(例如leetcode_237不支持Rust)
- 必须要调用clone或take才能非递归地遍历二叉树
- leetcode/codeforce社区上Rust的讨论和题解较少

Rust的所有权机制导致链表的操作很难

```rust
fn sort_list(mut head: Option<Box<ListNode>>) -> Option<Box<ListNode>> {
    if head.as_ref()?.next == None {
        return head;
    }
    let mut slow = &mut head as *mut Option<Box<ListNode>>;
    let mut fast = head.as_ref()?.next.as_ref();
    let mid = unsafe {
        while fast.is_some() && fast.as_ref()?.next.is_some() {
            slow = &mut (*slow).as_mut()?.next as *mut _;
            fast = fast?.next.as_ref()?.next.as_ref();
        }
        let mid = (*slow).as_mut()?.next.take();
        // cut left_part_list and right_part_list
        (*slow).as_mut()?.next = None;
        mid
    };
    let left_part = sort_list(head);
    let right_part = sort_list(mid);
    merge_two_sorted_lists(left_part, right_part)
}
```

Rust链表归并排序示例
慢指针需要可变借用，快指针不可变借用
可变借用类似RwLock或RWMutex中WLock
当存在一个WLock时，不能存在其它RLock/WLock
所以需要使用unsafe裸指针去绕开借用检查

Rust做题经验分享

- leetcode字符串入参用into_bytes()处理后可以下标访问
- 注意整数溢出，leetcode运行Rust代码溢出时不会panic
- 尽量用into_iter()代替iter()
- 排序用sort_unstable(快速排序)比sort(归并排序)要快
- 了解一些In-Place操作API(例如swap, replace, take)

Thank you

Rust China Conf 2020

Shenzhen, China

2020conf.rustcc.cn