Олимпиада «Ломоносов» по информатике. 2022-2023 учебный год
Работа участника с логином inf23f_281

Прайморадичная система счисления

Посылка по задаче 1 «Прайморадичная система счисления»

```
def cnt(s):
    num = s.split(':')
    if num.count('0') == len(num):
        return 10**12
    for i in range(len(num)-1,-1,-1):
        if num[i] != '0':
            return len(num) - i - 1

n = int(input())
arr = [cnt(input()) for x in range(n)]
mx = max(arr)
for x in range(len(arr)):
    if arr[x] == max(arr):
        print(x + 1)
```

Протокол проверяющей системы по задаче 1 «Прайморадичная система счисления»

см. файл report1.txt

Посылка по задаче 2 «Сундуки»

```
def cnt(s):
    ans = 0
    d = {'a': 0.01, 'b': 0.05, 'c': 0.1, 'd': 0.5, 'e': 1, 'f': 2, 'g': 5, 'h': 10, 'i': 25,
         'A': 5, 'B': 10, 'C': 50, 'D': 100, 'E': 200, 'F': 500, 'G': 1000, 'H': 2000, 'I': 5000}
    for el in s:
        ans += d[el]
    return ans


def solve():
    n = int(input())
    arr = [cnt(input()) for x in range(n)]
    mn, mx = min(arr), max(arr)
    first_min = arr.index(mn)
    first_mx = arr.index(mx)
    last_mn = -1
    last_mx = -1
    for x in range(n-1,-1,-1):
        if arr[x] == mn:
            last_mn = x
```

```
            break
    for x in range(n-1,-1,-1):
        if arr[x] == mx:
            last_mx = x
            break
    if abs(last_mx - first_min) >= abs(first_mx - last_mn):
        print(min(last_mx, first_min) + 1, max(last_mx, first_min) + 1, sep = '\n')
    else:
        print(min(first_mx, last_mn) + 1, max(first_mx, last_mn) + 1, sep = '\n')

solve()
```

Протокол проверяющей системы по задаче 2 «Сундуки»

```
OK
28 total tests runs, 28 passed, 0 failed.
Score gained: 100 (out of 100).
```

Посылка по задаче 3 «Кубик»

```
track_list = {}
good = tuple([i for i in "1111223344556666"])
def L(cube):
    new_cube = cube.copy()
    new_cube[0] = cube[12]
    new_cube[2] = cube[14]
    new_cube[4] = cube[6]
    new_cube[6] = cube[4]
    new_cube[14] = cube[2]
    new_cube[12] = cube[0]
    new_cube[8], new_cube[9] = cube[9], cube[8]
    return tuple(new_cube)


def R(cube):
    new_cube = cube.copy()
    new_cube[1], new_cube[3] = cube[13], cube[15]
    new_cube[13], new_cube[15] = cube[1], cube[3]
    new_cube[7], new_cube[5] = cube[5], cube[7]
    new_cube[10], new_cube[11] = cube[11], cube[10]
    return tuple(new_cube)


def U(cube):
    new_cube = cube.copy()
```

```
    new_cube[0], new_cube[15] = cube[15], cube[0]
    new_cube[1], new_cube[14] = cube[14], cube[1]
    new_cube[8], new_cube[10] = cube[10], cube[8]
    new_cube[6], new_cube[7] = cube[7], cube[6]
    return tuple(new_cube)


def D(cube):
    new_cube = cube.copy()
    new_cube[2], new_cube[13] = cube[13], cube[2]
    new_cube[3], new_cube[12] = cube[12], cube[3]
    new_cube[9], new_cube[11] = cube[11], cube[9]
    new_cube[4], new_cube[5] = cube[5], cube[4]
    return tuple(new_cube)


def F(cube):
    new_cube =cube.copy()
    new_cube[0], new_cube[1], new_cube[2], new_cube[3] = cube[2], cube[3], cube[1], cube[0]
    new_cube[6], new_cube[7], new_cube[10], new_cube[11], new_cube[5], new_cube[4], new_cube[9],
new_cube[8] = cube[9], cube[8], cube[6], cube[7], cube[10], cube[11], cube[5], cube[4]
    new_cube[14], new_cube[15], new_cube[13], new_cube[12] = cube[12], cube[14], cube[13], cube[15]
    return tuple(new_cube)


def backtrack(cube):
    ans = ""
    while track_list[cube][1] != "":
        ans += track_list[cube][1]
        cube = track_list[cube][0]
    return ans[::-1]


def solve():
    cube = tuple([i for i in input()])
    track_list[cube] = (-1, "")
    stack = [cube]
    while True:
        new_stack = []
        for el in stack:
            new_cube = L(list(el))
            if new_cube not in track_list:
                new_stack.append(new_cube)
                track_list[new_cube] = (el, 'L')
```

```
                if new_cube == good:
                    print(backtrack(new_cube))
                    return
            new_cube = R(list(el))
            if new_cube not in track_list:
                new_stack.append(new_cube)
                track_list[new_cube] = (el, 'R')
                if new_cube == good:
                    print(backtrack(new_cube))
                    return
            new_cube = U(list(el))
            if new_cube not in track_list:
                new_stack.append(new_cube)
                track_list[new_cube] = (el, 'U')
                if new_cube == good:
                    print(backtrack(new_cube))
                    return
            new_cube = D(list(el))
            if new_cube not in track_list:
                new_stack.append(new_cube)
                track_list[new_cube] = (el, 'D')
                if new_cube == good:
                    print(backtrack(new_cube))
                    return
            new_cube = F(list(el))
            if new_cube not in track_list:
                new_stack.append(new_cube)
                track_list[new_cube] = (el, 'F')
                if new_cube == good:
                    print(backtrack(new_cube))
                    return
        stack = new_stack.copy()

solve()
```

Протокол проверяющей системы по задаче 3 «Кубик»

```
см. файл report3.txt
```

Посылка по задаче 4 «Codemirror»

```
def solve():
    text = ""
    buffer = ""
    shift_start = -1
```

```
shift_end = -1
cursor = 0

stack = [""]
shift_on = False

n = int(input())
s = input()
for el in s:
    if 'a' <= el <= 'z':
        text = text[:cursor] + el + text[cursor:]
        cursor += 1
    elif el == '<':
        cursor = max(0, cursor - 1)
    elif el == '>':
        cursor = min(cursor + 1, len(text))
    elif el == '{':
        shift_start = cursor
        shift_end = shift_start
        shift_on = True
    elif el == '}':
        shift_on = False
    elif el == 'C':
        shift_start, shift_end = min(shift_start, shift_end), max(shift_start, shift_end)
        if shift_start == -1 or shift_start == shift_end:
            buffer = ""
        else:
            buffer = text[min(shift_start, shift_end): max(shift_start, shift_end)]
    elif el == 'X':
        shift_start, shift_end = min(shift_start, shift_end), max(shift_start, shift_end)
        buffer = text[min(shift_start, shift_end): max(shift_start, shift_end)]
        cursor = shift_start
        text = text[:shift_start] + text[shift_end:]
        shift_start, shift_end = -1, -1
    elif el == 'V':
        shift_start,shift_end = min(shift_start,shift_end), max(shift_start,shift_end)
        if shift_start == -1:
            text = text[:cursor] + buffer + text[cursor:]
            cursor += len(buffer)
        else:
            text = text[:shift_start] + buffer + text[shift_end:]
            cursor = shift_start + len(buffer)
        shift_start, shift_end = -1, -1
    elif el == 'D':
```

```
            shift_start, shift_end = min(shift_start, shift_end), max(shift_start, shift_end)
            if shift_start == -1:
                text = text[:cursor-1] + text[cursor:]
                cursor = min(0, cursor-1)
            else:
                text = text[:shift_start] + text[shift_end:]
                cursor = shift_start
            shift_start, shift_end = -1, -1
        if shift_on:
            shift_end = cursor
        stack.append(text)
    k = int(input())
    for i in range(k):
        print(stack[int(input())])
solve()
```

Протокол проверяющей системы по задаче 4 «Codemirror»

см. файл report4.txt

Посылка по задаче 5 «Библиотека»

```
books = {}
people = {}
books_ptr = {}
people_ptr = {}
owner = {}
next = {}
waiting = 0
max_waiting = 0
cycle = False


def process_book(book):
    global waiting
    if books_ptr[book] >= len(books[book]):
        return
    waiting -= 1
    person = books[book][books_ptr[book]]
    process(person)


def process(person):
    global waiting
    global max_waiting
    global cycle
```

```
    if people_ptr[person] >= len(people[person]):
        return
    cmd = people[person][people_ptr[person]]
    if cmd[0] == 'B':
        book = cmd[1]
        if book not in owner or owner[book] == -1:
            owner[book] = person
            next[person] = -1
            people_ptr[person] += 1
            process(person)
        elif person not in books[book]:
            books[book].append(person)
            waiting += 1
            max_waiting = max(waiting, max_waiting)
            if next[owner[book]] == -1:
                next[person] = owner[book]
            else:
                next[person] = next[owner[book]]
            if next[person] == person:
                cycle = True
                return
    else:
        book = cmd[1]
        books_ptr[book] += 1
        owner[book] = -1
        process_book(book)
        people_ptr[person] += 1
        process(person)


def check_empty():
    if len(owner) == 0:
        return False
    for el in owner:
        if owner[el] != -1:
            return False
    return True


def main():
    cnt = 1
    while 1:
        s = input().split()
        type_, person, book = s[0], s[1], " ".join(s[2:])
```

```
        if person not in people:
            people[person] = []
            people_ptr[person] = 0
            next[person] = -1
        if book not in books:
            books[book] = []
            books_ptr[book] = -1
            owner[book] = -1
        people[person].append((type_, book))
        process(person)
        if cycle:
            print(-cnt)
            return
        if check_empty():
            break
        cnt += 1
    print(max_waiting)

main()
```

Протокол проверяющей системы по задаче 5 «Библиотека»

см. файл report5.txt