به نام خدا

تمرین سری سوم توزیع شده

حسام مومیوند فرد

۸۱۰۸۰۳۰۶۳

Table of Contents

پاسخ سؤال اول

چون ما دو نود خطا دار داریم پس الگوریتم سه راند اجرا خواهد شد.

مطابق تصویر به علت اینکه مجموعه نود های غیر خطا دار ما بعد از سه راند مقدارهای ۰ و ۱ را ذخیره کرده‌اند پس تصمیم بر ۰ خواهند گرفت.

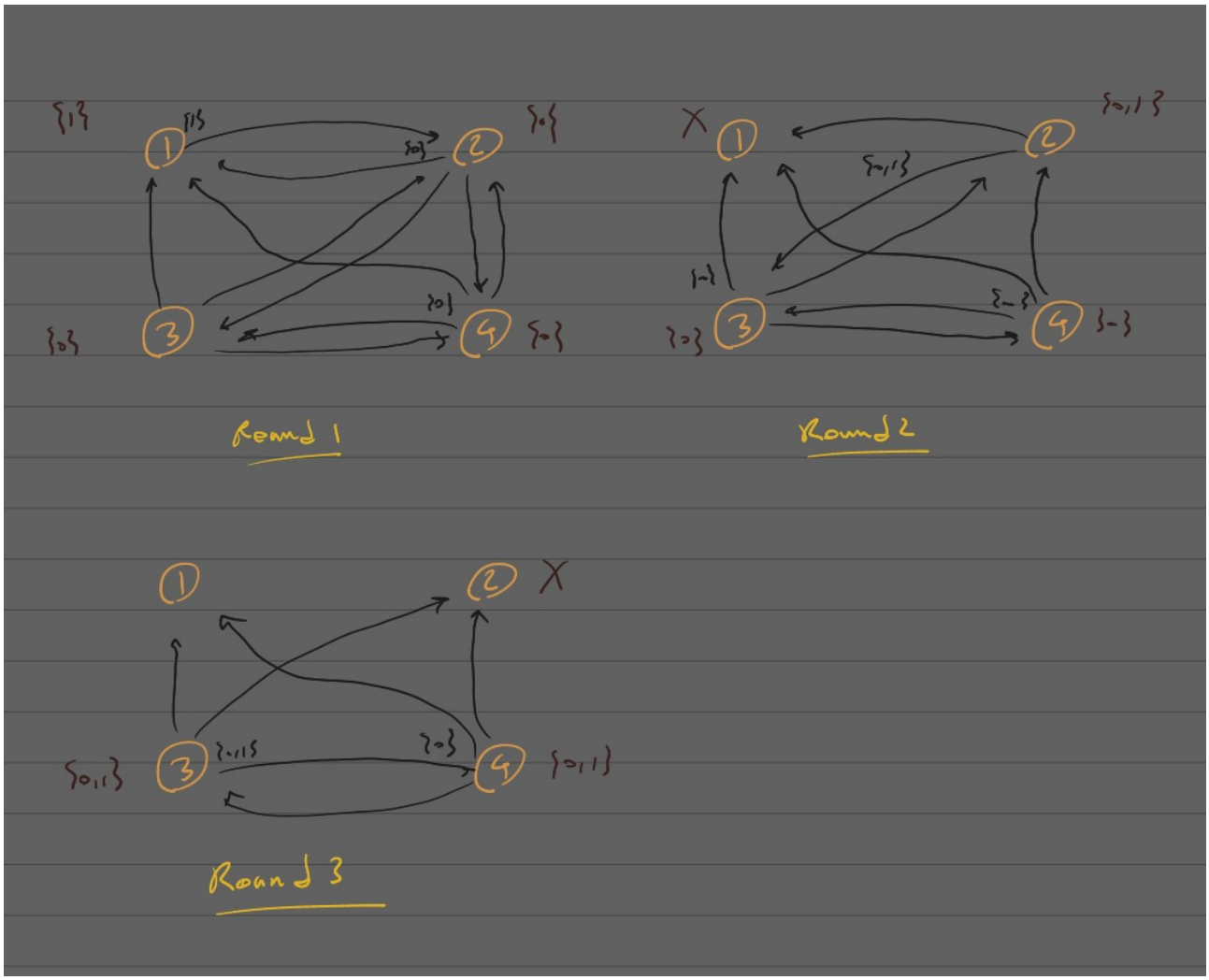

Figure ۱: مسئله‌ی floodset

پاسخ سؤال سوم

رویکرد کلی بدون امضای دیجیتال

رویکرد کلی ما پیاده‌سازی کلاس‌های زیر است:

- node
- process
- tree

هم چنین از یک فایل کمکی برای هندل کردن پیام‌ها استفاده میکنیم

- message_handler

و در نهایت شبیه سازی خود را در فایل زیر انجام میدهیم

- main

در زیر به اختصار هرکدام را توضیح میدهیم:

node

در این کلاس ما سه مولفه را نگه داری میکنیم:

- value
- leaf_flag
- path

مولفه value مقدار نود را نگه میدارد. Path برای نگهداری مسیر نود است (مسیر به معنای همان دنباله‌ای از id پراسس هاست مثل ۱۲۳ و یا ۲۳۱ و ...) و در نهایت leaf_flag به عنوان یک متغیر بولی بیانگر این است که نود ما یک برگ است یا خیر.

```
class Node:
    def __init__(self, value, path, leaf_flag=True):
        self.value = value
        self.path = path
        self.leaf_flag = leaf_flag

    def __repr__(self):
        message = "value:"+ str(self.value)+ " path:"+ str(self.path)+ "
leaf_flag:"+str(self.leaf_flag)
        return message
```

Tree

در این کلاس ما با استفاده از تعدادی نود یک درخت میسازیم

مقادیری که در این کلاس نگه داری میشوند:

- root
- list_of_nodes

در ابتدای ساختن این کلاس ما یک ریشه به درخت پاس میدهیم و پس از ساختن لیست نود ها درخت ریشه را به لیست اضافه میکند

همچنین توابع زیر نیز در درخت پیاده‌سازی میشوند:

- get_childes
- get_leafs

در تابع گرفتن فرزندان ما یک نود میگیریم و تمام فرزندان آن نود را بازمیگردانیم.

در تابع گرفتن برگ‌ها هم ما تمام برگ‌های درخت را به صورت یک لیست باز میگردانیم.

```
class Tree:
    def __init__(self, root):
        self.root = root
        self.list_of_nodes = []
        self.list_of_nodes.append(self.root)

    def get_childes(self, parent):
        childes = []
        for node in self.list_of_nodes:
            # check that the length of child be grater than the parent
            # and the path of parent exactly repeated at the first of child's path
            if (len(node.path) == len(parent.path)+1) and (parent.path ==
node.path[:-1]):
                childes.append(node)
        return childes

    def get_leafs(self):
        leafs = []
        for node in self.list_of_nodes:
            if node.leaf_flag:
                leafs.append(node)
        return leafs
```

Process

این کلاس چند ویژگی و چندین تابع را در خود نگه میدارد:

- process_id
- faulty
- initial_decision
- number_of_processes
- root
- tree
- level

در زمان ساختن یک شی از این کلاس ما تمام این مقادیر اولیه را به کلاس پاس میدهیم و مقدار اولیه فالتی بودن به صورت پیش‌فرض برابر false است و در صورتی که بخواهیم نود ما فالتی باشد آن را با مقدار true به کلاس پاس میدهیم.

```
    def __init__(self, process_id, initial_decision, number_of_processes,
am_i_faulty=False):
        self.process_id = process_id
        self.faulty = am_i_faulty
        self.initial_decision = initial_decision
        self.number_of_processes = number_of_processes
        self.root = Node(value=self.initial_decision, path="", leaf_flag=True)
        self.tree = Tree(self.root)
        self.level = len(self.root.path)
```

همچنین توابع زیر نیز در کلاس ما پیاده‌سازی شده‌اند:

- send_messages برای تولید تمام پیام‌ها و ارسال یک لیست از پیام‌های پراسس در هر راند
- generate_next_level_nodes تولید نودهای سطح بعدی
- update_leafs آپدیت کردن برگ‌ها (بعد از هر راند تمام نود های سطوح قبلی به حالت غیر برگ درخواهند آمد.
- update_tree آپدیت کردن کل درخت (بعد از هر راند و دریافت پیام‌ها ما مقادیر نودهای جنریت شده در هر سطح را از پیام‌ها دریافت کرده و در صورت ولید بودن مقدار نود مربوطه را آپدیت میکنیم.)
- decision_making تصمیم گیری

- print_final_info چاپ کردن مقادیر به صورت فرمت خواسته شده در صورت تمرین

همچنین یک تابع majority هم در فایل این کلاس و نه در خود کلاس پیاده‌سازی شده است.

```
def majority(nodes):
    counter0 = 0
    counter1 = 0
    for node in nodes:
        if int(node.value) == 1:
            counter1 += 1
        else:
            counter0 += 1
    if counter1 > counter0:
        return 1
    else:
        return 0
```

این تابع برای تصمیم گیری نودهای میانی استفاده میشود.

در زیر کد تمام توابع پیاده‌سازی شده را قرار داده ایم:

```
def send_messages(self):
    leaf_nodes = self.tree.get_leafs()
    messages = []
    for node in leaf_nodes:
        if not self.faulty:
            message = "value" + str(node.value) + "path" + str(node.path) + "sender" +
str(self.process_id)
            messages.append(message)
        else:
            message = "this message is garbage"
            messages.append(message)
    return messages

# generate the next level leafs of the tree
def generate_next_level_nodes(self):
    self.level += 1
    digits = ''.join(str(i) for i in range(1, self.number_of_processes+1))
    next_level_paths = [''.join(p) for p in permutations(digits, self.level)]
    nodes = []
    for path in next_level_paths:
        nodes.append(Node(value=None, path=path, leaf_flag=True))
    return nodes

# before generating new level of nodes all the old nodes should be non leaf
def update_leafs(self):
    for node in self.tree.get_leafs():
        node.leaf_flag = False


def update_tree(self, messages):
    valid_messages = message_handler.process_nested_message_lists(messages) # validation
all messages as string
    messages_as_dict = [] # extract all messages to dict
    for message in valid_messages:
        messages_as_dict.append(message_handler.extract_message_parts(message))
    self.update_leafs()
```

```
    # now all the messages are in dict type
    new_level_nodes = self.generate_next_level_nodes()
    for node in new_level_nodes:
        value = None
        for message in messages_as_dict:
            if message["sender"] == node.path[-1] and message["path"] == node.path[:-1]:
                value = message["value"]
                break
        node.value = value
        self.tree.list_of_nodes.append(node)


def decision_making(self):
    for node in self.tree.list_of_nodes:
        if node.leaf_flag:
            if node.value is None or node.value == "None":
                node.value = 0
    for node in reversed(self.tree.list_of_nodes):
        if not node.leaf_flag:
            childes = self.tree.get_childes(node)
            node.value = majority(childes)
    return self.root.value

def print_final_info(self):
    values = []
    for node in self.tree.list_of_nodes:
        values.append(node.value)
    return values
```

فایل message_handler

در این فایل ما با استفاده از re و تعریف کردن یک فرمت کلی برای پیام‌ها یک سری توابع را پیاده‌سازی میکنیم که کارکردن با پیام‌های متنی را برای ما آسان کنند:

- is_valid_string_message برای چک کردن اینکه فرمت کلی پیام متنی درست هست یا نه
- extract_message_parts تبدیل کردن پیام متنی به یک دیکشنری و بازگرداندن آن
- get_valid_messages_as_strgin این تابع یک لیست از پیام‌ها را گرفته و به ازای تک تک آن‌ها تابع ولیدیشن یک پیام را صدا میزند
- process_nested_messages_list به علت اینکه ما تمام پیام‌ها را در یک آرایه ذخیره میکنیم و سپس صحت تمام پیام‌ها را (برای هر پردازش یک ایندکس در آرایه و در هر ایندکس تمام پیام‌های آن پردازش) به صورت یک جا بررسی کنیم این تابع را پیاده‌سازی کرده‌ایم .
- extract_part_from_nasted_messages این تابع هم با رویکردی مشابه تابع بالا پیاده‌سازی شده و در آن تابع extract_message_parts را به ازای هر پیام صدا میزنیم

یک متغیر message_pattern هم داریم که در آن الگوی پیام درست را نگه داری میکنیم.

در زیر کدهای پیاده‌سازی شده در این کلاس را قرار داده ایم:

```
message_pattern = r"value(.+?)path(.*?)sender(.+?)$"


def is_valid_string_message(message):
    return bool(re.match(message_pattern, message))

def extract_message_parts(message):
    match = re.match(message_pattern, message)
    if match:
        return {
            "value": match.group(1),
            "path": match.group(2),
            "sender": match.group(3)
        }
    return None  # If not valid

def get_valid_messages_as_strings(messages):
    return [message for message in messages if is_valid_string_message(message)]

def process_nested_message_lists(nested_messages):
    valid_messages = []
    for messages in nested_messages:
        valid_messages.extend(get_valid_messages_as_strings(messages))
    return valid_messages

def extract_parts_from_nested_messages(nested_messages):
    extracted_parts = []
    for messages in nested_messages:
        for message in messages:
            if is_valid_string_message(message):
                extracted_parts.append(extract_message_parts(message))
    return extracted_parts
```

فایل main

در این فایل ما یک شبیه سازی از الگوریتم را خواهیم داشت و از توابع زیر استفاده میکنیم:

- generate_processes ساختن پراسس ها و مقدار دهی اولیه به آن‌ها
- simulation_round1 شبیه سازی راند اول
- simulation_other_rounds شبیه سازی راندهای دیگر تا تعداد نود فالتی + ۱
- print_result نمایش نتایج

کد تابع مین در زیر آمده است:

```
def generate_process(total, faulty):
    processes = []
    for i in range(1, total-faulty+1):
        process = Process(process_id=i, initial_decision=1, number_of_processes=total)
        processes.append(process)
    for i in range(total-faulty+1, total+1):
```

```
        process = Process(process_id=i, initial_decision=1, number_of_processes=total,
am_i_faulty=True)
        processes.append(process)
    return processes

def simulation_round1(processes):
    messages = []
    for process in processes:
        messages.append(process.send_messages())
    for process in processes:
        process.update_tree(messages)
    return processes

def simulation_other_rounds(processes, number_of_faults):
    for round in range(1, number_of_faults+1):
        messages = []
        for process in processes:
            messages.append(process.send_messages())
        for process in processes:
            process.update_tree(messages)

    processes_decisions = []
    for process in processes:
        processes_decisions.append(process.decision_making())

    return processes, processes_decisions

def print_result(processes, processes_decisions):
    for index, process in enumerate(processes):
        print("level f+1 info: ", process.print_final_info())
        print("final decision:", processes_decisions[index])

if __name__ == '__main__':
    number_of_processes = int(input("enter the number of processes: "))
    number_of_faulty_processes = int(input("enter the number of faulty processes: "))
    list_of_processes = generate_process(number_of_processes, number_of_faulty_processes)
    list_of_processes = simulation_round1(list_of_processes)
    list_of_processes, decisions = simulation_other_rounds(list_of_processes,
number_of_faulty_processes)
    print_result(list_of_processes, decisions)
```

در ادامه نتایج شبیه سازی رو با ورودی های خواسته شده در صورت تمرین قرار میدهیم:

```
enter the number of processes: 5
enter the number of faulty processes: 2
level f+1 info:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1', 0, 0, '1', 0, 0, 0, 0, 0, 0, 0, 0, '1', 0, 0, '1', 0
final decision: 0
level f+1 info:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1', 0, 0, '1', 0, 0, 0, 0, 0, 0, 0, 0, '1', 0, 0, '1', 0
final decision: 0
level f+1 info:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1', 0, 0, '1', 0, 0, 0, 0, 0, 0, 0, 0, '1', 0, 0, '1', 0
final decision: 0
level f+1 info:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1', 0, 0, '1', 0, 0, 0, 0, 0, 0, 0, 0, '1', 0, 0, '1', 0
final decision: 0
level f+1 info:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '1', 0, 0, '1', 0, 0, 0, 0, 0, 0, 0, 0, '1', 0, 0, '1', 0
final decision: 0
```

Figure ۲: نتایج شبیه سازی با ۵ نود و ۲ فالتی بدون امضای دیجیتال

به علت اینکه مقدار ۵ از سه برابر تعداد نود های فالتی بیشتر نیست پس قاعدتاً تصمیم تمام نودها باید ۰ باشد که هست.

در صورتی که ما قاعده‌ی $n < 3f$ را رعایت کنیم و مقادیر اولیه تمام نود ها ۱ باشد خروجی قابل قبول برای ما این است که تصمیم تمام نود ها نیز ۱ باشد.

```
enter the number of processes: 7
enter the number of faulty processes: 2
level f+1 info:  [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
final decision: 1
level f+1 info:  [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
final decision: 1
level f+1 info:  [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
final decision: 1
level f+1 info:  [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
final decision: 1
level f+1 info:  [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
final decision: 1
level f+1 info:  [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
final decision: 1
level f+1 info:  [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
final decision: 1
```

Figure ۳: خروجی اجرا با ۷ نود و ۲ فالتی بدون امضای دیجیتال

رویکرد کلی با امضای دیجیتال

مشابه با حالت بدون امضای دیجیتال پیش می‌رویم با این تفاوت که پیام‌ها را با استفاده از کلید های خصوصی رمز کرده و در هر لول برای اینکه مقدار value یک پیام را بدست بیاوریم با استفاده از اعمال پیاپی کلید های عمومی مورد انتظار به مقدار مد نظر خواهیم رسید.

خب اومدم پیاده‌سازی کنم به دیوار خوردم :)
نمیشه پیاده سازیش کرد با مدلی که من ساختم چون من همه پیام‌ها رو جمع میکنم و گیرنده پیام‌ها مشخص نیست و باید کل برنامه رو از بیس تغییر بدم سووووو فقط میتونم امیدوار باشم نمره این بخش تمرین زیاد نباشه :)