2143030 황진석 과제1 리포트

1. 개요

> **파이썬 3.9.13** 버전을 사용했습니다!
> 대체선택으로 RUN리스트를 만드는 프로그램입니다.

> **Github**
> https://github.com/dudasdaily/ComputerScience/tree/main/2학년_2학기/파일처리론/과제/replacement

> 2119943 조예영 학생과 같이 버퍼 내 동결 상태를 구현하기 위해
> Freeze 클래스를 만들고 boolean값을 가지는 필드변수를 만드는 것을 논의 했습니다. 또한 상대경로로 파일입출력을 하기위해 pathlib를 사용하는것을
> 논의했습니다.

2. 코드 설명

2.1. 필요한 라이브러리 및 설정

```
from pathlib import Path

buffer_size = 5

input_path = Path(__file__).parent / 'replacement_input.txt'
output_path = Path(__file__).parent / 'replacement_output.txt'

max_value = 100
```

> pathlib를 사용하여 상대경로로 파일입출력을 하도록 했습니다.
> 또 buffer크기와 원소의 최대 크기 max_value를 변수로 따로 빼두었습니다.

2.2 Freeze 클래스

```
class Freeze:
    def __init__(self, value: int, freeze=False):
        self.value = value
        self.is_freeze = freeze
```

> 정수값과 동결 상태를 가지는 Freeze 클래스를 통하여 버퍼 관리를 했습니다.

2.3 최소값 찾기 함수

```
def find_min(buffer: list):
    min_value = max_value + 1
    min_index = -1

    for i in range(len(buffer)):
        if not buffer[i].is_freeze and buffer[i].value < min_value:
            min_value = buffer[i].value
            min_index = i

    return min_index
```

> 인자로 버퍼 리스트를 줍니다.
> 버퍼 리스트는 Freeze 클래스로 구성된 원소들만 있습니다.
> 버퍼 리스트 원소들 중 동결되지 않은 원소의 최소값 인덱스를 리턴합니다.

2.4 파일 읽기 및 처리

```
write_file = open(output_path, "w")

with open(input_path, "r") as read_file:
    cases_num = int(read_file.readline())
```

> ***read_file***로 replacement_input.txt를 읽고,
> ***write_file***로 replacement_output.txt를 작성합니다.
> ***cases_num***은 입력 파일 첫번째 라인의 테스트 케이스 수를 저장할 변수입니다.

2.5 메인

```
for n in range(cases_num):
        buffer = []
        runs = [[]]
        runs_idx = 0
        freeze_cnt = 0

        element_num = int(read_file.readline())  # 원소 개수
        line = read_file.readline()  # str 한줄 읽기

        pharse_num = ""
        current_num = None
```

> - **buffer**
>   runs리스트를 만들기 전 임시공간인 버퍼입니다.

> - **runs**
>   버퍼에서 동결되지 않은 최솟값이 들어가는 리스트입니다.
>   버퍼가 다 동결되었을 때, 빈 리스트를 리스트 끝에 추가합니다.

- **runs_idx**
  runs 리스트의 인덱스입니다.
  버퍼가 다 동결되었을 때, 새로 만들어진 runs 리스트에 레코드를 추가하기 위한 인덱스 변수입니다.

- **freeze_cnt**
  버퍼에서 동결된 원소의 수를 저장하기 위한 변수입니다.

- **line**
  입력 파일에서 레코드가 입력되어있는 한 줄을 읽어 str으로 저장합니다.

- **pharse_num**
  입력 파일에서 정수들이 공백단위로 구분되는데, 정수 하나가 담길 str변수입니다.

- **current_num**
  가장 최근에 버퍼에서 runs로 내려간 값을 저장하는 int변수입니다.

2.5.1 레코드 읽기

```
for i in range(len(line)):
    # 버퍼가 꽉 찬 경우 생략
    '''
    '''

    # 버퍼 채우기
            if line[i] == " " or i == len(line) - 1:
                # 버퍼에 들어오는 값이 동결해야되는 값일 경우
                if pharse_num and (current_num != None) and
(int(pharse_num) < current_num):
                    buffer.append(Freeze(int(pharse_num), True))
                    freeze_cnt += 1
                    pharse_num = ""

                elif pharse_num:
                    buffer.append(Freeze(int(pharse_num)))
                    pharse_num = ""

            else:
                pharse_num += line[i]
```

반복문을 사용하여 line을 하나씩 읽어 pharse_num에 저장합니다.

읽은 글자가 " " 이거나 line의 마지막이고, pharse_num이 빈 문자열이 아닌경우
pharse_num과 current_num을 비교하여 동결할지 말지를 정하고,
버퍼에 Freeze 클래스 형태로 pharse_num을 저장합니다.

2.5.2 버퍼에서 runs로

```
for i in range(len(line)):
            # 버퍼가 꽉 찬 경우
```

```
            if len(buffer) == buffer_size:
                # 버퍼가 다 동결 된 경우
                if freeze_cnt == buffer_size:
                    runs.append([])
                    runs_idx += 1

                    for j in range(len(buffer)):
                        buffer[j].is_freeze = False
                    freeze_cnt = 0

                    min_idx = find_min(buffer)
                    current_num = buffer.pop(min_idx).value
                    runs[runs_idx].append(current_num)

                else:
                    min_idx = find_min(buffer)
                    current_num = buffer.pop(min_idx).value
                    runs[runs_idx].append(current_num)

            # 버퍼 채우기 생략
            '''
            '''
```

> 버퍼크기만큼 버퍼가 꽉 차면, 버퍼에서 동결되지 않은 최소값을 runs에 넣어줍니다.
> 만약 버퍼의 원소들이 전부 동결상태이면 새로운 runs에 빈 리스트를 만들어서 새로운 run을 생성합니다.

2.5.3 버퍼에 값이 남은 경우

```
        for i in range(len(line)):
            '''
            생략
            '''
        # 입력파일을 다 읽고 버퍼가 남은 경우
        while buffer:
            # 버퍼에 남은 값이 다 동결되어있는 경우
            if freeze_cnt == len(buffer):
                runs.append([])
                runs_idx += 1

                for k in range(len(buffer)):
                    buffer[k].is_freeze = False
                freeze_cnt = 0

                min_idx = find_min(buffer)
                current_num = buffer.pop(min_idx).value
                runs[runs_idx].append(current_num)

            else:
                min_idx = find_min(buffer)
                current_num = buffer.pop(min_idx).value
                runs[runs_idx].append(current_num)
```

> 앞서 반복문을 통해 line의 레코드들을 처리하여 버퍼에 넣고, runs에 추가했습니다
> 반복문 안에서 버퍼의 값을 runs에 추가했기 때문에,
> 반복문이 끝난 뒤 버퍼에 값이 있을경우 따로 처리해줘야 합니다.

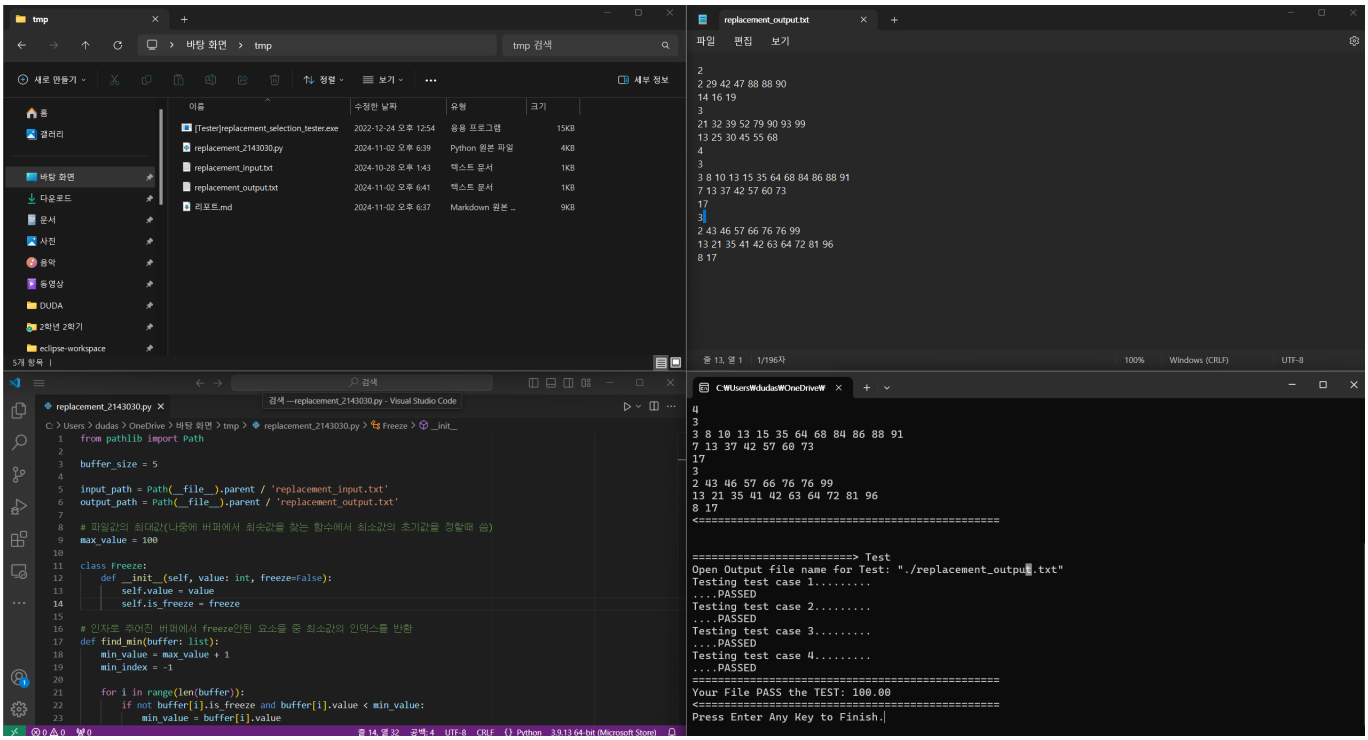
> 그래서 버퍼에 최소값을 runs로 옮기기 위해,
> 버퍼에 동결되지 않은 값을 runs에 추가하고
> 버퍼에 남은 값들이 다 동결된 경우, 동결상태를 풀고, 새 runs을 만들어 최소값을 넣도록 했습니다.

2.6 출력파일 생성

```
        for n in range(cases_num):
            # runs 생성 과정 생략
            '''
            '''

            # 생성된 Runs 숫자 쓰기
            write_file.write(str(len(runs)) + "\n")

            # runs[0], runs[1], ... 작성
            for i in range(len(runs)):
                for j in range(len(runs[i])):
                    if j != (len(runs[i]) - 1):
                        write_file.write(str(runs[i][j]) + " ")

                    elif j == (len(runs[i]) - 1) and i != len(runs):
                        write_file.write(str(runs[i][j]) + "\n")

                    else:
                        write_file.write(str(runs[i][j]))
```

> 각 케이스마다 첫번째 줄에 생성된 runs의 수를 쓰고 나머지 줄에 runs를 각각 한줄씩 입력했습니다.
> 각 runs는 '\n'으로 구분됩니다.

3. 프로그램 실행 결과



4. 어려웠던 점

> 레코드를 읽는 과정에서 레코드 전체 크기만큼의 리스트를 만들지 않는 것이 힘들었습니다.

```
line = read_file.readline()
line.split()
```

> 처음에는 위와 같이 split()함수를 사용해서 공백으로 구분 된 입력 레코드를 처리하려고 했지만,
> 그러면 레코드 크기 전체만큼의 리스트가 만들어지는 단점이 있어서,
> 빈 문자열(pharse_int)을 만들고 line 문자열을 하나씩 읽어와서 공백 단위로 정수 버퍼에 넣는 방식으로
> 메모리 공간을 아꼈습니다.

> 또 버퍼에서 runs에 가장 마지막에 들어간 숫자보다 레코드에서 버퍼로 읽은 값이 작은경우
> 그 값이 버퍼 최소값을 찾을 때 제외되도록 freeze 상태를 구현해야했는데,
> 이 과정에서 Freeze 클래스를 따로 만들었습니다.

> 원래는 버퍼에서 최솟값을 찾을 때 min()함수를 사용하면 편했겠지만, 버퍼가 Freeze클래스로 이루어져 있기때문에
> Freeze클래스 중 최소값을 찾는 find_min()함수를 따로 만드는 것도 힘들었습니다.
> 특히 find_min()함수에서 반환값의 초기값을 설정할 때 힘들었는데,
> 과제에서 원소의 범위가 0 <= x <= 100 인 정수인 것을 보고 max_value 전역변수를 만들고,
> 초기값 = max_value + 1 로 만들어 해결했습니다.