자료구조
수행시간 비교 체험 과제 보고서

컴퓨터공학과 202202976 임준혁

1. 목적

수학에서 어느 문제를 풀 때 풀이과정이 여럿 존재하는 것처럼 프로그래밍에 있어 문제를 해결하는 방법도 제각각이다. 하지만 알고리즘의 구상이 비슷하다면 수행시간 또한 비슷하고 BigO 표기법도 동일할 수 있다. 본 보고서에서는 어떠한 한 문제를 해결하는데 있어서 알고리즘이 수행시간에 얼마나 영향을 끼치는지 분석한다.

2. 분석

예제 문제를 요약하자면, 자연수 n을 입력 받고 [-n, n] 범위의 정수의 값 n개를 랜덤하게 생성해 리스트에 저장할 때 리스트에 서로 다른 값의 개수를 계산해 출력하는 문제다. 이 문제를 해결하기 위한 알고리즘 4개를 정리하고 실행시간을 분석한다.

a. 알고리즘 정리

i) 알고리즘 1 #O(n^2)

브루트포스 알고리즘

가장 단순한 방법으로 전체를 탐색하는 방법이다.
이중 반복문을 사용하여 탐색하여 같은 값이 존재하지 않으면 sum 변수를 1씩 올려 서로 다른 수를 센다.
같은 값이 존재한다면 다음 탐색에서 중복되게 수를 세지 않도록 랜덤 값으로 존재할 수 없는 변수를 삽입한다.

가장 쉽게 떠올릴 수 있는 알고리즘이지만 시간 복잡도가 O(n^2)인 치명적인 단점이 존재한다.

```python
def sol_1():  #O(n^2)
    arr = randArr.copy()
    sum = 0
    for i in range(n):
        for j in range(n):
            if (i==j):  continue
            if(arr[i] == arr[j]):
                arr[i] = None
        if(arr[i] != None):
            sum += 1
    print(sum)
    return
```

ii) 알고리즘 2 #O(nlogn)

정렬 알고리즘

우선 리스트를 오름차순으로 정렬한다. 반복을 통해 i번째 값과 i-1번째의 값을 비교해 다른 경우 sum 변수를 1씩 올려 서로 다른 수를 세는 방법이다.

이 방법은 정렬시간에 따라 시간 복잡도가 바뀔 수 있다. Python의 정렬 함수는 O(nlogn)의 시간 복잡도로 설계되어 있으며 이 후에 n번 반복되는 1차 반복문은 BigO 표기법에서 큰 영향을 미치지 않는다. 따라서 이 알고리즘은 O(nlogn)의 시간 복잡도를 갖는다.

```python
def sol_2(): #O(nlogn)
    arr = randArr.copy()
    arr.sort()
    sum = 1
    for i in range(1,n):
        if(arr[i] != arr[i-1]):
            sum += 1
    print(sum)
    return
```

iii) 알고리즘 3 #O(n)

딕셔너리 자료구조

Python의 딕셔너리 자료형을 이용하여 해결한다. 딕셔너리는 Key : Value 형태를 이루며 저장되며 특정 Key값의 존재여부도 알 수 있다. 따라서 randArr의 i번째 값이 딕셔너리의 Key값으로 존재한다면 그 수를 더 이상 세지 않으면 된다.
만약 존재한다면 딕셔너리에 randArr[i]를 Key값으로 추가한다. 결국 딕셔너리의 크기가 서로 다른 수의 개수가 된다.

```
def sol_3():  #O(n)
    arr = randArr.copy()
    D = {}
    sum = 0
    for i in range(n):
        if not arr[i] in D:
            sum += 1
            D[arr[i]] = True
    print(sum)
    return
```

딕셔너리 자료형을 이용하면 n번의 반복만으로도 문제를 해결할 수 있으면서 메모리도 많이 사용하지 않아 가장 효율적인 문제해결법이라고 할 수 있다. 시간 복잡도는 O(n).

iv) 알고리즘 4 #O(n)

DP 알고리즘

마찬가지로 n번의 반복만으로 이 문제를 해결할 수 있는 방법이 있다. 임의의 리스트의 [랜덤한 수]번 째 값을 하나 씩 올린다. 하지만 랜덤한 수의 범위는 -n부터 시작한다. 즉, 음의 정수의 값도 포함이기 때문에 임의의 리스트 tempArr[음의 정수]는 원하지 않는 값을 초래할 수 있다.
따라서 임의의 리스트의 [랜덤한 수 + n]번 째 값을 하나 씩 올리는 방법을 사용한다. 그렇다면 임의의 리스트 tempArr의 크기는 $2 \times n + 1$ 이며 모든 값은 0으로 선언한다. 이제 $2 \times n + 1$번의 반복을 하며 0이 아닌 값이 존재하는 tempArr[i]의 개수는 서로 다른 수의 개수가 된다.

```
def sol_4():  #O(n)
    arr = randArr.copy()
    tempArr = [0] * (n*2+1)
    sum = 0
    for i in range(n):
        tempArr[arr[i]+n] += 1
    for i in range(n*2+1):
        if(tempArr[i] != 0):
            sum += 1
    print(sum)
    return
```

이 방법도 O(n)의 시간 복잡도를 가지게 되지만 메모리를 (랜덤한 수를 가진 리스트) + (두 배 큰 임의의 리스트)를 사용하여 총 세 배 사용하므로 메모리 제한에 주의해야 한다.

b. 실행시간 분석

각 알고리즘을 실행한 후 끝나는 시간에서 시작했던 시간을 빼면 실행시간을 알 수 있다.
아래는 알고리즘 별로 n의 크기가 달라질 때 실행시간을 정리한 내용이다.

	N=100	1,000	10,000	20,000	1,000,000	10,000,000
Sol_1	0.00161	0.07808	4.93746	44.97651	시간초과	시간초과
Sol_2	0.00004	0.00013	0.00141	0.004433	0.188958	2.328044
Sol_3	0.00005	0.00011	0.00085	0.002750	0.139882	2.306446
Sol_4	0.00006	0.00015	0.00144	0.004393	0.162292	2.329689

3. 후기

과제를 하면서 알고리즘과 자료구조의 사용에 따라 실행시간에 엄청난 영향을 끼치는 것을 체감할 수 있었고 입력 조건에 맞는 최적의 알고리즘을 구현하는 연습을 해야 한다고 생각한다. 최악의 경우의 시간 복잡도를 염려해두면서 알고리즘을 계획하는 것이 중요한 문제가 되겠다.