알고리즘 수행시간 분석 과제 보고서

202202976 컴퓨터공학과 임준혁

1. 목적

ICPC는 문제를 풀면서 가장 효율적으로 해결할 수 있는 형태의 알고리즘을 떠올리는 것이 핵심이다. 여기서 효율적이라고 하는 것은 항상 의도된 답을 도출해 내면서도 실행되는 시간을 최소한으로 하는 것을 말한다. ICPC뿐만 아니라 어디서든, 프로그램이 결과값이 도출되기까지 적은 비용으로 빠르게 만들수록 좋다. 극단적인 예로 특정 유저를 찾는 프로그램이 걸리는 시간이 O(n^3)의 시간 복잡도를 가진다고 가정해본다. 1초에 10억회를 연산한다고 가정하면, 유저수가 5000명만 되어도 대략 2분(125×10^9회의 연산이므로)이 걸린다. 물론 조 아무도 유저의 수를 찾는 알고리즘을 n^3으로 알고리즘을 구상하지는 않겠지만 알 수 있는 점은 이제는 단순히 답을 도출하는 것 뿐만이 아니라 주어진 건(시간과 메모리공간 등)을 만족하는 알고리즘을 구상해야 한다는 것이다.

2. 구현

하나의 문제를 보고 각각의 시간 복잡도가 다른 세가지 알고리즘을 구현하고 이 알고리즘들의 실행 시간을 측정해 알고리즘의 시간 복잡도의 중요성을 확인한다.

해결할 문제는 -n에서 +n까지의 n개의 정수의 리스트에서 중복이 있는지 검출하는 문제며 n은 2에서 100,000으로 제한된다. 하지만 **3. 분석**에서는 시간 측정을 위해 n의 최대 크기를 늘려보았다.

세 알고리즘을 각각 O(n^2), O($n \log n$), O(n)이고 편의상 순서대로 A, B, C 알고리즘이라고 한다.

A 알고리즘 O(n^2)

```
def unique_n2(A):
    s = time.process_time()
    ans = False
    for i in range(n):
        for j in range(n):
            if i == j: continue
            if(A[i] == A[j]): ans = True
    e = time.process_time()
    print("수행시간 =", e-s)
    return ans
```

중복이 있으면 True, 없으면 False를 반환한다. A 알고리즘은 이중반복문을 적용하고 있고 리스트 A[i]와 A[j]를 비교하면서 같은 경우 ans(반환값)을 True로 바꾼다. 만약 i와 j가 같은 경우는 비교하지 않는다.

비교연산은 상수 시간에 처리가 가능하기 때문에 Big-O에 반영하지 않고 이중 반복문을 사용했기 때문에 시간 복잡도는 O(n^2)이다.

B 알고리즘 O($n \log n$)

```
def unique_nlogn(A):
    s = time.process_time()
    ans = False
    B = sorted(A)
    for i in range(n-1):
        if(B[i] == B[i+1]): ans = True
    e = time.process_time()
    print("수행시간 =", e-s)
    return ans
```

B에서는 정렬을 이용한다. 리스트가 오름차순으로 정렬된 경우에는 현재 A[i]과 A[i+1]값을 비교하면 되기 때문에 이중 반복문을 할 필요가 없어진다. 정렬된 리스트에서 단순히 n번의 반복으로 중복을 알아낼 수 있다.

하지만 주의해야할 점은 파이썬의 내장함수 sorted()인데 이 함수는 $n \log n$의 시간 복잡도를 가진다. 따라서 $O(n \log n + n) = O(n \log n)$ 이다.

C 알고리즘 O(n)

```
def unique_n(A):
    ans = False
    C = [0 for _ in range(2*n+1)]
    s = time.process_time()
    for i in range(n):
        C[A[i] + n] += 1
    for i in range(2*n+1):
        if (C[i] > 1): ans = True
    e = time.process_time()
    print("수행시간 =", e-s)
    return ans
```

n번에 해결하기 위한 방법으로 우선 -n부터 +n까지 정수의 개수를 크기로 하는 리스트 C를 0으로 초기화한다. 리스트 C의 (x+n)번째 인덱스의 값은 x값이 몇개 존재하는지를 의미한다.
첫번째 n번의 반복을 통해 A[i]가 몇 번 등장하는지 C[A[i]+n]에 기록한다. 두 번째 반복문은 리스트 C의 크기만큼 반복하며 C[i]의 값이 2 이상인지 비교하고 2 이상이라면 중복이라는 의미가 되므로 ans를 True로 바꾼다. 시간 복잡도는 $O(n + (2 * n + 1)) = O(n)$이다.

3. 분석

구현한 알고리즘이 실제로 시간 복잡도로 예상한 것과 동일하게 차이가 나는지 비교해 분석한다.

#중복을 허용하는 랜덤 데이터

dataset = random.sample(range(-n, n+1), n)

	N = 1,000	10,000	100,000	1,000,000	10,000,000	30,000,000
A	0.07724	4.762629	시간초과	시간초과	시간초과	시간초과
B	0.000167	0.001292	0.016450	0.174898	2.225916	11.100354
C	0.000137	0.001233	0.013262	0.141267	2.181560	7.130406

(단위: 초)

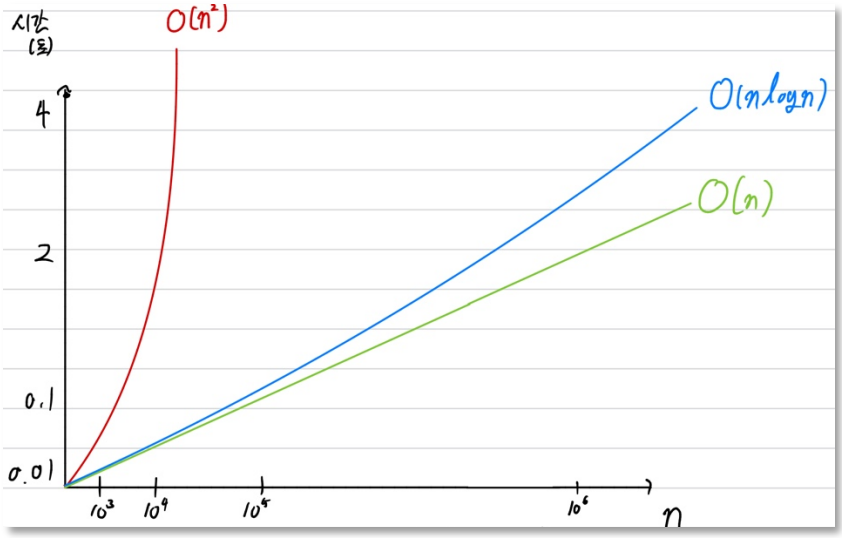
#중복을 허용하지 않는 랜덤 데이터

dataset = [random.randint(-n, n) for _ in range(n)]

	N = 1,000	10,000	100,000	1,000,000	10,000,000	20,000,000
A	0.076317	4.946069	시간초과	시간초과	시간초과	시간초과
B	0.000122	0.001244	0.016662	0.224022	2.813554	6.356054
C	0.000131	0.0012031	0.013698	0.178003	2.398926	5.184749

(단위: 초)

위 두 표를 기반으로 그린 n에 따른 시간의 그래프이다.



중복 탐색 알고리즘 A, B, C 시간 복잡도 그래프

4. 결론

A 알고리즘은 O(n^2)이기 때문에 n이 100,000일 때부터는 오래 걸려 측정하지 못 했지만 10,000,000회의 연산을 1초안에 할 수 있다고 가정할 때 10,000,000,000회 즉, 최소 1,000초(=약16분)이 걸릴 것으로 예상된다. C는 B와 유사한 수치를 보이고 있지만 더 커질 n이 들어오면 B는 언제나 C보다 크다고 할 수 있다.

흥미로웠던 점은 그래프를 AI로 그리는 것을 시도하려고 아무런 다른 추가정보 없이 오직 표를 대화형 AI 'ChatGPT'에 입력해봤을 때, 오히려 정확하게 알고리즘을 역으로 분석했다.

관찰

1. **A 알고리즘**
   - $N = 10,000$까진 실행되지만 이후부턴 **시간 초과**.
   - 따라서 $O(N^2)$ 이상의 복잡도를 가진 것으로 보임 (아마도 버블 정렬, 삽입 정렬 같은 단순 정렬).
2. **B 알고리즘**
   - 큰 N에서도 실행은 가능.
   - 실행 시간이 대략 $N \log N$ 패턴처럼 증가.
   - 전형적인 효율적인 정렬 알고리즘(퀵 정렬, 병합 정렬, 힙 정렬 등)일 가능성이 큼.
3. **C 알고리즘**
   - B와 비슷하지만 더 빠름.
   - 실행 시간이 안정적으로 낮고, 큰 N에서도 B보다 약 30~40% 정도 성능이 좋음.
   - B보다 최적화된 정렬 알고리즘이거나, 입력 분포에 최적화된 알고리즘일 수 있음.
     (예: Timsort → 파이썬 기본 정렬, 혹은 Radix Sort처럼 데이터 특성을 활용하는 선형 정렬)

👉 정리하면:

- **A = 단순 정렬 알고리즘 (O(N^2))**
- **B = 효율적인 정렬 알고리즘 (O(N log N))**
- **C = 더 빠른 정렬 알고리즘 (최적화된 N log N 또는 O(N))**

(A, B, C가 알고리즘이라는 정보도 일체 입력하지 않았다.)

효율적인 알고리즘 구상은 프로그램 개발자에게 있어서 개발 자체라고도 할 수 있을 만큼 중요하다. 'ChatGPT'와 같은 AI 모델을 이용해 개발을 할 수 있겠지만 오류나 보안의 문제에서는 직접 분석하는 능력을 길러야 한다. 따라서 알고리즘 분석이 AI 활용의 근간이 된다고 생각한다. AI를 이용하는 시대가 다가온 만큼 알고리즘 공부와 분석에 열중해야 한다고 느낀다.