과제

장유선

2023.08.21

1. 문제 정의

1. 이전 과제인 부분집합을 구하고 집합 연산을 수행하는 알고리즘에 대해서 빅오 표기법으로 알고리즘의 복잡도를 구하세요
2. 여러 정렬 알고리즘에 대해 조사하고, 각 알고리즘의 복잡도를 확인하세요

1번 과제

```
import time
import random

# 전체집합
full_set = list(range(1, 1000001))

# 부분집합 생성 함수 정의
def generate_random_subset(count, source_set):
    subset = set()  # 빈 집합
    while len(subset) < count:  # 원하는 개수만큼 원소가 추가될 때까지 반복
        index = random.randint(0, len(source_set) - 1)  # 무작위 인덱스를 선택
        element = source_set.pop(index)  # 선택한 인덱스의 원소를 전체집합에서 뽑는다
        subset.add(element)  # 부분집합에 추가
    return subset

# 부분집합 생성
subset1 = generate_random_subset(700000, full_set.copy())
subset2 = generate_random_subset(700000, full_set.copy())

start_time = time.time()  # 시작 시간 기록

# 합집합, 교집합, 차집합 계산
union_set = subset1.union(subset2)  # 두 부분집합의 합집합
intersection_set = subset1.intersection(subset2)  # 두 부분집합의 교집합
difference_set = subset1.difference(subset2)  # 첫 번째 부분집합에서 두 번째 부분집합을 뺀 차집합

end_time = time.time()  # 종료 시간 기록
```

- 부분집합을 구하는 알고리즘

  부분집합을 구하는 알고리즘은 count 개수만큼 원소를 무작위로 선택하여 부분집합을 생성한다. While문을 사용하여 count 개수만큼 원소를 선택한다. while문은 count만큼 반복되고, source_set에서 해당 원소를 찾기 위해 O(count)의 시간이 걸린다. 따라서 이 루프의 시간 복잡도는 O(count^2)이다.

- 집합 연산을 수행하는 알고리즘

  합집합, 교집합, 차집합 연산은 set 객체의 내장 메서드를 사용하므로 O(n) 시간 복잡도를 갖는다. 여기서 n은 두 부분집합의 크기 중 작은 값이다.

전체 알고리즘에서 가장 시간이 많이 걸리는 알고리즘은 generate_random_subset 함수(부분집합을 구하는 알고리즘)이므로, 전체 알고리즘의 시간 복잡도는 O(count^2)이다.

2번 과제

1. 버블 정렬 (Bubble Sort):

- 인접한 두 원소를 비교하면서 큰 값을 뒤로 이동시키는 정렬 알고리즘
- 시간 복잡도: 최선, 평균, 최악 모두 O(n^2)이다.
- 장점: 구현이 쉽다, 코드 자체가 직관적이다.
- 단점: 비효율적이다.

2. 선택 정렬 (Selection Sort):

- 주어진 배열에서 가장 작은 값을 선택해 맨 앞으로 이동시키는 정렬 알고리즘
- 시간 복잡도: 최선, 평균, 최악 모두 O(n^2)이다.
- 장점: 구현이 쉽다, 정렬을 위한 비교 횟수는 많지만 실제로 교환하는 횟수는 적기 때문에 많은 교환이 일어나야하는 자료 상태에서 효율적으로 사용될 수 있다.
- 단점: 비효율적이다.

3. 퀵 정렬 (Quick Sort):

- 피벗(pivot)을 선택하고, 피벗을 기준으로 작은 값은 왼쪽, 큰 값은 오른쪽으로 분할하며

정렬하는 알고리즘

- 시간 복잡도: 평균 O(n log n), 최악 O(n^2)이다.
- 장점: 실행시간이 길지 않다.
- 단점: 기준값에 따라서 시간복잡도가 크게 달라진다.

4. 힙 정렬 (Heap Sort):

- 힙 자료구조를 이용하여 정렬하는 알고리즘으로, 힙을 구성하고 최대 힙에서 원소를 꺼내면서 정렬
- 시간 복잡도: 항상 O(n log n)이다.
- 장점: 효율적이다
- 단점: 안정성을 보장받지 못한다.

5. 삽입 정렬 (Insertion Sort):

- 배열을 정렬된 부분과 정렬되지 않은 부분으로 나누고, 정렬되지 않은 원소를 정렬된 부분에 삽입
- 시간 복잡도: 최선의 경우 O(n), 평균 및 최악의 경우 O(n^2)이다.
- 장점: 빠르다.
- 단점: 데이터의 상태 및 데이터의 크기에 따라 성능의 편차가 심하다.

6. 병합 정렬 (Merge Sort):

- 분할 정복 전략을 사용하여 배열을 작은 조각으로 나눈 뒤 병합하면서 정렬
- 시간 복잡도: 항상 O(n log n)이다.
- 장점: PIVOT에 따라서 성능이 안좋아지는 경우가 없다.
- 단점: 추가적인 메모리가 필요하다.

7. 셸 정렬 (Shell Sort):

- 간격을 줄여가며 부분적으로 삽입 정렬을 수행하여 정렬하는 알고리즘입
- 시간 복잡도: 평균적으로 O(n log n), 최악의 경우 O(n^2)이다.
- 장점: 삽입정렬의 단점을 보완해서 만든 정렬법으로 삽입정렬도 성능이 뛰어난 편이지만 더 뛰어난 성능을 갖는 정렬법이다.
- 단점: 일정한 간격에 따라서 배열을 바라봐야 한다. 즉, 이 '간격'을 잘못 설정한다면 성능이 굉장히 안 좋아질수 있다.

8. 기수 정렬 (Radix Sort):

- 각 자리수를 기준으로 정렬하는 알고리즘으로, 비교 연산을 하지 않음
- 시간 복잡도: O(nk), 여기서 k는 숫자의 자리수이다.
- 장점: 빠르다.
- 단점: 버킷이라는 추가적인 메모리가 할당되어야 한다.

9. 카운팅 정렬 (Counting Sort):

- 주어진 입력을 세서 각 원소의 개수를 세고, 이를 기반으로 정렬하는 알고리즘
- 시간 복잡도: O(n + k), 여기서 k는 입력 범위의 크기이다.
- 장점: 빠르다.
- 단점: 추가적인 메모리가 필요하다.