**알고리즘 설계와 분석**

**CSE3081-MP1**

2017xxxx 000

* Experiment environment

Window 10 Home

프로세서 : Intel® Core™ i5-7200U CPU @ 2.50GHz 2.71GHz

RAM : 8GB

시스템 종류 : 64비트 운영체제, x64 기반 프로세서

* Experiment setup

Argorithm1의 경우, time complexity를 계산하였을 때, 대략적인 식은 (C : 상수, N : 행의 수, M : 열의 수)이 나왔다.

Argorithm2의 경우, time complexity를 계산하였을 때, 대략적인 식은 (C : 상수, N : 행의 수, M : 열의 수)이 나왔다.

Argorithm3의 경우, time complexity를 계산하였을 때, 대략적인 식은 (C1, C2 : 상수, N : 행의 수, M : 열의 수)이 나왔다.

모든 알고리즘에서 시간에 영향을 미치는 변수들은 행과 열의 수였다. 이라고 가정하면, 세 알고리즘의 time complexity는 각각 , , 가 된다. 이를 확인하기 위해 테스트 케이스를 N=M인 경우를 주었으며, 이는 10\*10, 20\*20, 30\*30, , 100\*100 이다.

이 때, N\*M의 배열을 입력 받기 위해 정적 배열을 사용하였다. A[10000]이 N\*M 배열의 셀들의 값을 1차원적으로 입력 받으므로 N\*M의 크기가 10000이 넘어가는 input 값은 받을 수 없다. 또한, Algorithm3을 구현하는 데에 있어서 열 단위로 sum을 입력 받는 1차원 배열 B[1000]을 이용하였으므로 M은 1000을 넘어선 안 된다. 마지막으로, input filename과 output filename은 각각 크기가 200인 문자열을 사용하였으므로 파일명이 200자를 넘어선 안 된다.

* Observation

다음은 각 알고리즘 별로 test case에 따른 time(ms)값을 표와 그래프로 나타낸 것이다.

, , 에서 input size n의 증가에 따른 time complexity를 확인하기 위해 앞서 말한 대로 M=N인 경우를 test case로 주었다. 즉, input은 n\*n (n=10, 20, , 100)행렬이다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 1 | 0.507 | 215.87 | 156.601 | 683.775 | 2103.851 | 5740.200 | 13826.157 | 29908.631 | 58369.064 | 101029.849 |

(ms)

Algorithm1의 경우, test case를 통해 나온 time값의 추세선을 그려보면 거듭제곱 함수의 형태로 증가한다는 것을 알 수 있다. 이 추세선의 식을 구해보면 대략 n의 5제곱으로 bigO를 통해 구한 시간 복잡도 와 차이는 좀 있지만 비슷하다는 것을 볼 수 있다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2 | 0.049 | 0.264 | 1.473 | 4.547 | 10.412 | 20.723 | 35.966 | 57.909 | 86.683 | 119.877 |

(ms)

Algorithm2의 경우 역시 test case를 통해 나온 time값의 추세선을 그려보면 거듭제곱 함수의 형태로 증가한다. 이 추세선의 식을 구해보면 대략 n의 4제곱으로 bigO를 통해 구한 시간 복잡도 와 비슷하다는 것을 볼 수 있다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 3 | 0.018 | 0.088 | 0.321 | 0.804 | 1.360 | 2.665 | 3.984 | 5.968 | 9.254 | 11.837 |

(ms)

Algorithm3의 경우에도 test case를 통해 나온 time값의 추세선을 그려보면 거듭제곱 함수의 형태로 증가한다. 이 추세선의 식을 구해보면 대략 n의 3제곱으로 bigO를 통해 구한 시간 복잡도 와 매우 비슷하다.

세 알고리즘 모두 거듭제곱의 형태로, 초반에 input size가 작을 때는 time값이 큰 차이가 없었다. 그러나 input size가 증가할수록 그 차이는 엄청나게 커졌고, 특히 Algorithm1의 경우에는 뒤쪽으로 갈수록 알고리즘을 돌릴 때 결과 값이 나오기까지 눈에 띌 정도로 오랜 시간이 걸렸다. 시간 복잡도를 나타내는 식을 비교할 때 거듭제곱 꼴이 시간 측면에서 안 좋은 알고리즘이라는 말을 몸소 체감할 수 있었다.