**Maximum Sum Sub Rectangle In An 2D Array**

20161566 권형준

1. 실험 환경

|  |  |
| --- | --- |
| CPU | Intel Core i5-8265U |
| CPU 속도 | 1.80 GHz |
| 메모리 용량 | 8.00 GB |
| 시스템 종류 | 64비트 운영체제, x64기반 프로세서 |
| OS 종류 | Windows 10 |

1. 실험 구조

우선 구조를 크게 나누면 3개의 부분으로 나눌 수 있다. Algorithm index가 1일 때, 2일 때, 3일 때. 각각의 경우는 시간 복잡도가 O(n^6), O(n^4), O(n^3)이 되게 하였다.

1. 변수 설명

N, m : 행과 열의 개수

Left, right : O(n^3) 알고리즘에서 좌측,우측 column을 선정하기위한 변수

i\_s, j\_s, i\_e, j\_e: 시작점과 끝점의 x,y좌표

table : 파일에서 읽어온 배열을 저장하는 배열

kadane : O(n^3)에서 연속하는 최대 숫자합을 구하기 위한 1차원 배열

algorithm : algorithm index를 저장하는 값

sum : 최종 합

temp\_sum: 임시 합

initial : sum이 비어있는지 확인해주는 값

temp\_initial : temp\_sum이 비어있는지 확인해주는 값

filename : 입력 파일명

result\_filename : 출력 파일명

1. 공통된 부분

읽어온 수는 table배열에 저장하였고 크기는 동적할당을 사용하여 nXm이 되게 하였다. Algorithm 3에서 사용하는 kadane배열 또한 n개의 1차원 배열로 동적할당 하였다. 또한 파일의 input명은 최대 200글자까지만 받기로 하였다. 만약 input에 파일명과 algorithm index 외의 input이 들어온 경우 “wrong input”을 출력하고 종료하였고, algorithm index의 값에 1, 2, 3을 제외한 값이 들어온 경우에도 “wrong input”을 출력하게 하였다. fopen으로 input, output 파일을 열고 만들었다. 또한 clock\_t 를 사용하여 총 실행시간을 계산하였다.

1. Algorithm index가 1인경우 🡪 O(n^6)

* 처음 for문을 두 번 사용하여 table에 값을 그대로 저장하였다. 🡪 n^2
* for문을 6번 사용하여 시작점의 x, y 좌표, 끝점의 x, y 좌표, 그리고 그 사이의 모든 값들의 합을 구한다. 이 값들을 다 비교해서 가장 최대의 값을 구한다. 🡪 n^6

★T(n)=n^2+n^6=O(n^6)

1. Algorithm index가 2인경우 🡪 O(n^4)

* 처음 for문을 두 번 사용하여 table에 값을 그대로 저장하지 않고, (0,0)와 (i,j)를 꼭지점으로 갖는 사각형 내부의 모든 숫자의 합을 table[i][j]에 저장한다. 이는 Dynamic programming에서와 같이 이전의 table값에 새 값을 더하는 식으로 저장한다. 🡪 n^2
* for문을 4번 사용하여 시작점과 끝점의 x, y좌표를 구한다. 시작점과 끝점을 알게 되면 다음과 같이 그 사이의 모든 숫자의 합을 알 수 있다. 만약 A가 시작점에 저장된 값( (0,0) ~ (시작점) 사이의 모든 숫자의 합)이고, (A+B)가 시작점의 row와 끝점의 column의 table에 저장된 값이고, (A+C)가 시작점의 column과 끝점의 row의 table에 저장된 값이라고 하면, (A+B+C+D)는 끝점의 table값에 저장된 값이라고 할 수 있다. 이때, 우리가 비교하고자 하는 값은 D임으로

D = (A+B+C+D) – (A+C) – (A+B) + A

=끝점의 table) – (시작점의 column, 끝점의 row table) – (시작점의 row, 끝점의 column table) + (시작점의 table)

이렇게 구한 D값들을 비교하여 최대값을 구한다 🡪 n^4

|  |  |
| --- | --- |
| A | B |
| C | D |

★T(n) = n^2+n^4 = O(n^4)

1. Algorithm index가 3인경우 🡪 O(n^3)

* for문을 두 번 사용하여 table에 값을 그대로 저장한다. 🡪 n^2
* 여기서 사용할 아이디어는 우선 column을 두개 정한다. 그 column 사이의 모든 row의 값을 더한다. 그렇게 나온 1차원 배열을 dynamic program을 사용하여 배열내에서 연속하는 수의 최대합을 구한다. 모든 경우의 수를 따진 후의 최댓값이 결과이다.

for문을 2개 사용하여 좌측 column을 먼저 정하고 우측 column을 정하면서 하나씩 row의 값들을 더해서 1차원 배열에 저장한다. for문을 하나 사용하여 그 1차원 배열에서의 연속하는 수의 최대합을 구한다.

table이 다음과 같다면

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 좌측 column |  |  | 우측 column |  |
| A\_0 | A\_1 | A\_2 | A\_3 | A\_4 | A\_5 |
| B\_0 | B\_1 | B\_2 | B\_3 | B\_4 | B\_5 |
| C\_0 | C\_1 | C\_2 | C\_3 | C\_4 | C\_5 |
| D\_0 | D\_1 | D\_2 | D\_3 | D\_4 | D\_5 |
| E\_0 | E\_1 | E\_2 | E\_3 | E\_4 | E\_5 |

1차원 배열은 다음과 같다

|  |
| --- |
| A\_1 + A\_2 + A\_3 + A\_4 |
| B\_1 + B\_2 + B\_3 + B\_4 |
| C\_1 + C\_2 + C\_3 + C\_4 |
| D\_1 + D\_2 + D\_3 + D\_4 |
| E\_1 + E\_2 + E\_3 + E\_4 |

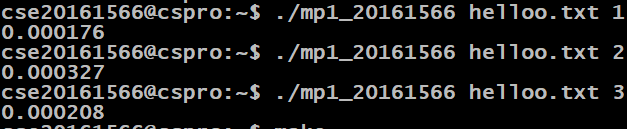
이때 이 1차원 배열에서 연속하는 최대 숫자합을 구하는 방법은 다음과 같다. 우선 배열의 첫번째 값을 최댓값으로 저장한다. 그리고 첫 숫자를 임시 변수 temp에 저장한다. 다음 두번째 변수를 temp에 더하고 그 값이 최댓값에 저장된 값보다 크다면 최댓값에 temp를 저장해준다. 만약 temp가 0보다 작다면 temp를 0으로 초기화한다. 이 과정을 마지막 배열까지 반복한다. 🡪 n^3

★T(n) = n^2 + n^3 = O(n^3)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Time(sec) | O(n^6) | O(n^4) | O(n^3) |
| N |  |
| 10 | | 0.000666 | 0.000225 | 0.000204 |
| 20 | | 0.018262 | 0.001004 | 0.000407 |
| 30 | | 0.134650 | 0.004261 | 0.000809 |
| 40 | | 0.547646 | 0.012961 | 0.001401 |
| 50 | | 1.643036 | 0.029418 | 0.002485 |
| 60 | | 4.404192 | 0.056860 | 0.003605 |
| 70 | | 10.675297 | 0.091738 | 0.005901 |
| 80 | | 23.246763 | 0.142397 | 0.007965 |
| 90 | | 46.483224 | 0.197756 | 0.011286 |
| 100 | | 86.184328 | 0.272373 | 0.014819 |

1. 결과 분석

Input의 크기가 작을 때는 모든 알고리즘의 실행 시간에는 큰 차이가 보이지 않는다. 하지만 input이 (N=50)이상만 돼도 O(n^6)의 실행시간은 기아급수적으로 증가하게 된다. Input이 매우 작을 시에는 O(n^6)이 O(n^3)보다 적은 시간이 걸리는 결과가 나오기도 한다. 그 이유는 사실상 O(n^3)의 T(n)이 n^3이 아닌 n^2 + n(n+n(n+n))= 2(n^3+n^2)이기 때문이다. 따라서 n의 값이 작을 때에는 O(n^6)이 O(n^3)보다 더 적은 시간이 걸릴 수 있다. 다음 예시를 살펴보면 더 확실하게 알 수 있다.



다음은 3X3의 행렬에서 algorithm index가 각각 1,2,3일때의 경우이다. 확인해보면 O(n^3)이 O(n^6)보다 더 오래 걸린다는 것을 확인할 수 있다. 하지만 결국에 input이 커지면 차이는 엄청나게 된다.