|  |
| --- |
|  |
| 알고리즘설계와분석 <HW1> |
|  |
|  |
| 컴퓨터공학과 20161620 이수연 |
| **CSE3081-02 임인성 교수님** |
| **2017. 10. 16. (월)** |

|  |
| --- |
|  |

**A. Maximum Subsequence Sum Problem (최대 부분 수열 합 문제)**

주어진 수열에서 특정 구간을 정하여 그 안의 원소들을 모두 더한 합을 구하였을 때, 그 값이 최대가 되도록 하는 구간과 그때의 값을 구하는 문제이다. 알고리즘은 크게 4가지가 있으며 시간 복잡도가 각각 n^3, n^2, nlogn, n 으로 나뉜다. 이번 과제에서는 시간 복잡도가 n^2, nlogn, n이 되게끔 세가지 알고리즘을 구현해보고 실제 시간을 측정하여 시간 복잡도를 확인한다.

**3. 입력 데이터 크기에 따른 수행 시간과 시간 복잡도의 관계를 충실히 분석할 수 있도록, 다양한 입력 크기 n에 대해 상기 세 가지 알고리즘에 대한 실험을 수행하라. 실험 결과의 신빙성을 높이기 위하여 어떤 노력을 기했는지 서술하라.**

이번 과제에서 입력 파일은 이진(binary)파일이었다. 우선 코드의 결과가 맞는지부터 확인하기 위해서 텍스트 파일로 내가 직접 답을 알고 있는 여러 가지 예시를 넣어 실험하였다. 그 후에 제공받은 파일로 크기가 큰 파일을 입력하여 시간을 측정하였다.

1) 수행 결과가 맞는지 여부

최대한 예외 사항을 고려하기 위하여 다음과 같은 예시 파일들을 만들었고 수행 결과는 아래와 같았다.

* 합의 최대값이 음수가 나오는 경우 (입력 값이 모두 음수)
* 입력 값이 모두 음수이면 최대값으로 0을 출력하고 구간은 (-1, -1)을 출력한다.
* 동일한 최대값을 가지는 구간이 두 개 이상 존재하는 경우
* 앞의 구간이 출력되도록 한다. \*\*
* 최대값에 해당하는 구간의 왼쪽과 오른쪽이 같은 경우 (예를 들어 한 수만 양수이고 나머지는 모두 음수인 경우)
* 양수인 수에 해당하는 인덱스가 구간의 양쪽이 된다. 예를 들어 양수인 수가 배열의 4번 째 인덱스라면 구간은 (4, 4)를 출력한다.
* 강의 자료에 나와있던 예시 (일반적 예시)
* 강의 자료에 제시된 답이 잘 나왔고 그 외에 인터넷에서도 일반적인 예시들을 찾아서 넣어보아서 잘 나오는 것을 확인하였다.
* 크기가 100 이상으로 꽤 큰 경우 (제공받은 코드로 큰 크기의 파일을 만든 후 그 파일을 읽어내는 코드를 돌려 생성된 숫자를 복사하여 입력파일에 넣음)
* 이것은 답을 알지는 못하기에 맞았는지 여부는 확인하지 못했으나 세가지 알고리즘에 대하여 모두 같은 값이 출력되는 것을 확인하였다.

\*\*두 번째, 동일한 최대값을 가지는 구간이 두 개 이상 존재하는 경우에 대해서는 알고리즘 1번과 3번에 대해서는 문제가 없었으나, 2번 divide and conquer 기법을 사용하는 알고리즘에서는 첫 번째 구간을 출력하지 않고 가운데에 걸쳐있는 구간을 출력하는 듯 하였다. 가령 입력파일의 수열이 <-1, 2, 3, -5, 2, 3>인 경우엔 최대값이 5이고 그 구간은 (1, 2)와 (1, 5) 두 개이다. 원칙상으로는 앞 구간인 (1, 2)를 출력해야 하지만 중간을 걸쳐있는 (1, 5)가 출력되었다. 재귀함수를 호출하는 과정에서 인덱스를 구하는 위치가 잘못된 것 같았는데, 결국 고치지 못하였다.

2) 시간 측정

제공받은 파일로 크기가 큰 파일을 생성 후 입력하여 시간을 측정하였다. 우선은 같은 크기에 대하여 5번씩 실험 후 각 알고리즘이 소비하는 시간의 평균을 구하였다. 이는 정확성을 위한 작업이었다. 그 후에 다른 크기의 데이터를 5개 정도 이용하여 이 과정을 반복하였다. 이론상으로 시간 복잡도가 각각 n^2, nlogn, n 이었기 때문에 결과를 대비하기 쉽게 하기 위하여 n을 주로 2의 지수 승으로 설정하였다. 그 외에 100과 같이 일반적인 큰 값도 실험하였다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **(단위 = millsec)** | **N = 8 (2^3)** | **N = 100** | **N = 512 (2^9)** | **N = 16384 (2^14)** | **N = 1048576 (2^20)** |
| **A1** | 0.0188 | 0.0589 | 1.8254 | 1502.8622 | 대략 1분  (측정 제대로 되지 x) |
| 0.0022 | 0.0588 | 1.3764 | 1557.5913 |
| 0.0094 | 0.0594 | 1.4253 | 1554.1813 |
| 0.0102 | 0.0602 | 1.9909 | 1509.0020 |
| 0.0074 | 0.0545 | 1.4958 | 1594.0010 |
| **평균** | **0.0058** | **0.0584** | **1.6223** | **1543.5276** | **X** |
| **A2** | 0.0027 | 0.0292 | 0.1257 | 6.8048 | 306.6066 |
| 0.0038 | 0.0508 | 0.1820 | 6.1591 | 304.1777 |
| 0.0054 | 0.0508 | 0.2231 | 11.8767 | 619.5005 |
| 0.0054 | 0.0513 | 0.2630 | 15.0245 | 564.4802 |
| 0.0054 | 0.0508 | 0.2220 | 8.2051 | 412.1838 |
| **평균** | **0.0045** | **0.0464** | **0.2031** | **9.6140** | **441.3898** |
| **A3** | 0.0021 | 0.0044 | 0.0092 | 0.1934 | 11.8102 |
| 0.0022 | 0.0043 | 0.0097 | 0.1923 | 11.6055 |
| 0.0021 | 0.0052 | 0.0102 | 0.1930 | 11.4292 |
| 0.0023 | 0.0039 | 0.0093 | 0.1928 | 11.7794 |
| 0.0019 | 0.0042 | 0.0092 | 0.1933 | 11.8409 |
| **평균** | **0.0021** | **0.0044** | **0.0096** | **0.1930** | **11.6930** |

대체적으로 N값이 클수록 걸리는 시간도 커지며, 알고리즘의 이론적 시간 복잡도에 따라 같은 크기여도 측정 시간이 달라짐이 보인다.

**4. 상기 실험을 통하여 산출한 실험 결과로부터 각 알고리즘의 이론적인 시간 복잡도와 수행 시간 간의 관계를 분석하고 그로부터 자신이 발견한 사실과 그에 대한 의견을 정확히 기술하라.**

위의 표에서 평균의 결과 값을 가져오면 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **(단위 = millsec)** | **N = 8 (2^3)** | **N = 100** | **N = 512 (2^9)** | **N = 16384 (2^14)** | **N = 1048576 (2^20)** |
| **A1 (o(n^2))** | **0.0058** | **0.0584** | **1.6223** | **1543.5276** | **X** |
| **A2 (o(nlogn))** | **0.0045** | **0.0464** | **0.2031** | **9.6140** | **441.3898** |
| **A3 (o(n))** | **0.0021** | **0.0044** | **0.0096** | **0.1930** | **11.6930** |

표와 차트를 보았을 때 N이 커질수록 측정 시간이 커지는 것이 현저히 보이며, 3번째 알고리즘으로 갈수록 측정 시간이 작아지는 것이 확인된다. 특히 N이 커질수록 이 차이가 크게 나타난다. 또 정확히 시간 복잡도를 비교하기 위해 2의 지수 승의 값을 택하였다. 예를 들어 N = 8 일 때와 N=512일 때를 비교해보자. 단순히 이론적 시간 복잡도에 N을 넣으면 N=8일 때 알고리즘 별로 소요되는 시간이 대략 64, 24, 8이므로 즉 비율이 8 : 3 : 1이 되어야 하는데, 아주 맞지는 않으나 비슷한 비율을 보인다. 또 N=8 일 때와 N=512일 때를 비교해보면 첫 번째 알고리즘의 경우 (N=8일 때 걸리는 시간 : N=512일 때 걸리는 시간) = 1 : 256 정도 인데 실제로는 1 : 300 정도의 비율을 보인다. 두 번째 알고리즘의 경우 이론적 비율은 1 : 45 정도 인데 실제로도 그러하여 아주 잘 맞았다. 세 번째 알고리즘의 경우 이론적 비율은 1 : 8 정도이며 실제 비율은 1 : 5 로, 어느 정도 맞는 것을 확인할 수 있었다. 이론적 값과 아주 정확히 들어맞을 수 없는 가장 큰 이유는 시간 복잡도를 나타낼 때 빅오 표기법을 나타내면서 각종 상수들이 생략됨에 있다. 또한 사용자의 실행 환경에 따라 수시로 달라지기 때문이기도 하다. 그러나 대략적인 맥락 (N이 커질수록 측정 시간이 커지며, 3번째 알고리즘으로 갈수록 측정 시간이 작아지는 것) 은 실험에서 육안으로 확인될 정도였다.

주목해야 할 것은 제일 큰 N=1048576의 경우 첫 번째 알고리즘을 사용하면 거의 시간 측정이 안될 정도로 오래 걸린다는 것이다. 다른 것들의 경우 단 몇 초 만에 결과가 나오는데 비해 이 경우엔 결과를 꽤 기다려야 했으며, N의 값이 더 커지면 커질수록 더욱 오래 걸릴 것이다. 그러나 똑 같은 데이터를 세 번째 알고리즘으로 수행하면 바로 답이 출력되는 것을 볼 수 있다. 이를 통해 더 큰 프로그램이나 실생활에서 이용할 공학적 프로그램을 디자인 할 때 실제로 시간복잡도를 고려하는 것이 얼마나 중요한 일인지를 알게 되었다. 공학에서 가장 중요한 점은 ‘효율성’ 이므로 반드시 시간 복잡도를 고려하면서 조금이라도 시간을 줄이기 위해 노력해야 한다고 생각한다.

<수행 환경>

* OS : Windows 8.1 Enterprise K
* RAM : 4.00GB
* CPU : Intel® Pentium® 3805U @ 1.90GHz 1.90GHz
* 시스템 종류 : 32비트 운영 체제, x64 기반 프로세서
* Compiler : visual studio 2015 Release Mode

**5. 2^20보다 더 큰 입력 값일 때?**

-> N = 2^100으로 시도해 보니 알고리즘에 따라 측정되는 시간의 차이가 더욱 두드러졌으며, 특히 첫 번째 알고리즘으로 할 때는 결과가 계속 나오지 않아 도중에 멈춰야 했다.

**B. Inversion Counting Problem**

주어진 수열에서 특정 구간을 골랐을 때 첫 번째 숫자와 마지막 숫자의 크기 순서가 서로 뒤바뀌어있는 경우, 즉 첫 번째 숫자의 크기 < 마지막 숫자의 크기 인 경우를 카운트 해주는 문제이다.

**O(nlogn)의 시간 복잡도를 가지는 알고리즘 설계 방법과 강의자료 57, 58쪽의 merge sort 코드에 어떠한 변경을 가했는지 기술한다.**

-> 만약 두 개의 정렬된 구간이 있다면, 두 구간 사이의 Inversion Pair의 개수는 O(n)에 구할 수 있다는 점을 이용한다. 이미 정렬되어 있으므로 n번만 비교해서 inversion pair가 나온다면 그 뒤의 것들도 마찬가지가 되므로 n번만 비교하면 되기 때문 이다. 두 개의 정렬된 구간을 가지려면 merge sort를 이용하면 되므로 이를 통해 nlogn의 시간 복잡도를 가지는 알고리즘의 설계할 수 있다. 이번 과제의 경우 강의자료의 merge sort코드에서 최소한의 변경만 할 것을 요구하였으므로, merge sort를 진행하는 merge 함수에서 inversion pair를 세는 한 줄만 추가하였다. (빨간색 코드)

void merge(int \*arr, int left, int middle, int right)

{

int i, i\_left, i\_right;

memcpy(buffer + left, arr + left, sizeof(int\*)\*(right - left + 1));

i\_left = left;

i\_right = middle + 1;

i = left;

while ((i\_left <= middle) && (i\_right <= right))

{

if (buffer[i\_left] < buffer[i\_right])

arr[i++] = buffer[i\_left++];

else

{

arr[i++] = buffer[i\_right++];

count = count + (middle - i\_left + 1); ///추가한 부분

}

}

while (i\_left <= middle)

arr[i++] = buffer[i\_left++];

while (i\_right <= right)

arr[i++] = buffer[i\_right++];

}

(끝)