REPORT #1

|  |
| --- |
| **제 출 일** : 2019년 9월 18일 |
| **과 목 명** : 알고리즘설계와해석 |
| **담당교수 : 김희철 교수님** |
| **학 과** : 컴퓨터전자시스템공학부 |
| **학 번** : 201803818 |
| **성 명 : 한아영** |

**HANKUK UNIVERSITY OF FOREIGN STUDIES**

****

**Problem Definition**

n개의 실수에 대해 합치 최대가 되는 연속하는 실수들의 개수와 그 합을 구하는 4가지 알고리즘에 대하여 수행시간을 비교하고 결과를 분석한다.

**Solution**

아톰 에디터로 파이썬 언어를 수행하여 여러 case들에 대해 수행시간을 측정한다.

수행시간이 한 시간을 넘을 경우 강제 종료하며 각 알고리즘에 대한 testcase는 파일 입출력 형식을 이용하여 수행한다.

먼저 공통적으로 파일을 읽어와 각 testcase에서 알고리즘을 수행할 개수를 num에, 그 숫자들을 리스트 num\_list에 저장한다.

알고리즘 1)

합이 나올 수 있는 모든 경우에 대해서 계산한다. for문이 3중으로 중첩되어 있으며 첫 번째 for문은 각 경우에 대한 첫번째 원소를, 두번째 for 문은 마지막 원소를, 마지막 for문은 그 첫번째 원소와 마지막 원소 사이의 값들을 더하기 위함이다.

알고리즘 2)

알고리즘 1에 있던 마지막 for문은 같은 연산을 여러 번 하게 함에 있어 비효율적이다. 첫번째 for문에서 첫번째 원소가 지정되면 두번째 for문에서 마지막 인덱스가 지정되고 그 합을 변수에 저장해놓은 후 loop variable이 가리키는 인덱스가 1씩 늘어날 때마다 저장해놓은 합의 값에 마지막 인덱스를 더한다.

알고리즘 3)

먼저, 길이를 저장할 sum리스트과 합을 저장할 length 리스트를 선언한다. 두 리스트는 num\_list의 해당 인덱스를 i라고 할 때 i번째까지의 합 중 가장 큰 값과 그 때의 길이를 저장한다. sum[i]는 sum[i-1]이 양수이면 sum[i-1]의 값에 num\_list[i]값을 더한다.

알고리즘 4)

알고리즘 3과 비슷한 맥락에서 배열을 사용하지 않고 currentSum이라는 변수를 지정한다.

currentSum이 0보다 크면 currentSum값에 num\_list의 값을 더하고 그렇지않으면 currentSum값을 num\_list의 값으로 초기화한다.

maxSum값은 currentSum값이 커질때마다 그 때의 currentSum값으로 초기화된다.

**Result**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ALGORITHM 1 | ALGORITHM 2 | ALGORITHM 3 | ALGORITHM 4 |
| TESTCASE 1 | 0.152 | 0.179 | 0.199 | 0.167 |
| TESTCASE 2 | 0.156 | 0.182 | 0.2 | 0.186 |
| TESTCASE 3 | 0.193 | 0.158 | 0.156 | 0.193 |
| TESTCASE 4 | 0.19 | 0.19 | 0.179 | 0.182 |
| TESTCASE 5 | 0.211 | 0.181 | 0.194 | 0.189 |
| TESTCASE 6 | 0.384 | 0.205 | 0.182 | 0.179 |
| TESTCASE 7 | 3.064 | 0.214 | 0.186 | 0.191 |
| TESTCASE 8 | 23.278 | 0.307 | 0.184 | 0.198 |
| TESTCASE 9 | 179.838 | 0.602 | 0.185 | 0.195 |
| TESTCASE 10 | 2920.277 | 3.12 | 0.2 | 0.194 |
| TESTCASE 11 | 측정불가 | 13.505 | 0.207 | 0.198 |
| TESTCASE 12 | 측정불가 | 43.404 | 0.215 | 0.205 |
| TESTCASE 13 | 측정불가 | 265.277 | 0.251 | 0.224 |
| TESTCASE 14 | 측정불가 | 1098.966 | 0.304 | 0.249 |

각 알고리즘에 대한 수행시간 비교 차트이다. 알고리즘3과 알고리즘 4의 수행시간이 비슷해 육안으로 확인이 힘들어 두 개의 알고리즘만 따로 차트를 만들어 비교하였다.

알고리즘 1의 경우, for문이 세번 중첩 되어있고 최악의 경우 첫번째 원소부터 마지막 원소까지 살펴보는 경우인데 이때 수행시간은 O(n^3)이라 할 수 있다. 그렇기 때문에 숫자들이 적을 때에는 딱히 문제가 되지 않았는데 수가 늘어날수록 수행시간은 기하급수적으로 증가했다. 2000개일때는 수행시간이 약 3분밖에 되지 않았는데 5000개일때는 48분으로 급격하게 늘나더니 숫자가 10000개가 되자 한시간이 넘어 측정할 수 없었다.

알고리즘 2의 경우 for문이 하나 빠지면서 O(n^2)의 수행시간을 갖는다. 다행히도(?) 14개의 테스트케이스가 한 시간 안에 결과가 나왔지만 100000개의 경우 수행시간이 20분이 되면서 여전히 오래 기다려야 했다

알고리즘 3과 4는 10만개라는 단어가 작게 느껴질 만큼 순식간에 결과값이 나왔으며 시간이 증가하기는 했지만 아주 극소한 차이로 증가했음을 확인할 수 있다.

**Conclusion**

이번 과제로 인해 알고리즘을 어떻게 짜는 지에 따라 결과물의 차이가 급격하게 달라진다는 것을 확인하였다. 앞으로 프로그램을 효율적으로 구현하는 방법에 대해서 많이 고민해봐야 겠다는 생각이 들었다. 같은 입력에 같은 결과를 출력하는 데도, 또 코드의 길이가 그렇게 많이 차이가 나지 않는데도 수행시간은 0.n초에서 한시간 넘게까지 차이 날 수 있다는 것이 놀라웠다. 알고리즘 1과 2, 3에서 시간 차이를 비교했다면 알고리즘 3과 4에서는 사용하는 메모리 차이를 비교할 수 있었다. 알고리즘 3에서 길이와 합에 대한 리스트를 만들었었는데 이에 대한 정보를 currentSum으로 업데이트하면서 리스트를 사용하지 않아 메모리를 아낄 수 있었다.

**Source Code**

**Algorithm1)**

savepath = "c:/Workspace/assignment\_files/testcase10.txt"

with open (savepath,"r") as f:

num = int(f.readline())

num\_list = f.readline().split()

num\_list = [int (i) for i in num\_list]

max\_sum = num\_list[0]

min\_length = 1

for i in range(0,num):

for j in range(i,num):

sum = 0

for k in range (i,j+1):

sum = sum + num\_list[k]

if sum > max\_sum :

max\_sum = sum

min\_length = j-i+1

elif sum == max\_sum :

if min\_length > j-i+1:

min\_length = j-i+1;

print (min\_length, max\_sum)

**algorithm2**

savepath = "c:/Workspace/assignment\_files/testcase14.txt"

with open (savepath,"r") as f:

num = int(f.readline())

num\_list = f.readline().split()

num\_list = [int (i) for i in num\_list]

max\_sum = num\_list[0]

min\_length = 1

for i in range(0,num):

sum = 0

for j in range(i,num):

sum = sum + num\_list[j]

if sum > max\_sum :

max\_sum = sum

min\_length = j-i+1

elif sum == max\_sum :

if min\_length > j-i+1:

min\_length = j-i+1;

print (min\_length, max\_sum)

**algorithm3**

savepath = "c:/Workspace/assignment\_files/testcase14.txt"

with open (savepath,"r") as f:

num = int(f.readline())

num\_list = f.readline().split()

num\_list = [int (i) for i in num\_list]

sum = [0]\*num

length = [0]\*num

sum[0] = num\_list[0]

length[0] = 1

for i in range (1,num):

if sum[i-1] > 0 :

sum[i] = sum[i-1] + num\_list[i]

length[i] = length[i-1] + 1

else:

sum[i] = num\_list[i]

length[i] = 1

maxSum = sum[0]

minLength = length[0]

for i in range (1,num):

if maxSum < sum[i]:

maxSum = sum[i]

minLength = length[i]

elif maxSum == sum[i] :

if length[i] < minLength :

minLength = length[i]

print (minLength, maxSum)

**algorithm4**

savepath = "c:/Workspace/assignment\_files/testcase2.txt"

with open (savepath,"r") as f:

num = int(f.readline())

num\_list = f.readline().split()

num\_list = [int (i) for i in num\_list]

maxSum = currentSum = num\_list[0]

minLength = num+1

currentLength = 1

for i in range (1,num) :

if currentSum > 0 :

currentSum += num\_list[i]

currentLength += 1

else :

currentSum = num\_list[i]

currentLength = 1

if maxSum < currentSum :

maxSum = currentSum

minLength = currentLength

else:

if maxSum == currentSum:

if currentLength < minLength:

minLength = currentLength