ASSIGNMENT II

COEN 279 (36219)
Oct 2, 2021
Gowtham Rajeshshekaran (W1607812)
grajeshshekaran@scu.edu

---

1. Exercise 4.1 – 1

It will return only one element which will be the maximum value in the array.

---

2. Exercise 4.1 – 2

```
"""
Maximum Subarray - Brute Force Approach

Array A passed will hold the changes in the prices between 2 consecutive days
as values, with the first value being 0 for day 1 (since no change)"""

def max_subarray(A):
    N = len(A)
    profit = float('-inf')
    buy = 0 # Day to buy
    sell = 0 # Day to sell

    for i in range(1, N):
        summ = 0
        for j in range(i, N):
            summ += A[j]
            if summ > profit:
                buy = i-1
                sell = j
                profit = summ

    return (buy, sell, profit)
```

---

3. **Exercise 4.1 – 3**

Following are the inputs taken from Algorithms written in **python code.** I ran it on the sublime editor V3 in my local machine. My local machine is running the **M1 processor** from **MacBook Air M1** model.

Please find the code **here**

Input Size	Brute Force	Recursive
5	0.88e-05	1.79e-05
10	1.407e-05	4.506e-05
20	2.503e-05	5.698e-05
50	13.375e-05	16.832e-05
60	14.424e-05	15.521e-05
65	**16.737e-05**	**16.689e-05**
70	19.217e-05	17.905e-05
100	42.796e-05	28.801e-05

The crossover point occurs when the problem size lies between **(60 <= n0 <= 65)** by when the recursive algorithm starts beating the brute force algorithm.

When I modified the base case of the recursive algorithm to call the brute force algorithm and ran it when the base case reaches <= 65, **there isn't significant change** in the crossover point.

---

4. Exercise 4.2 – 1

<u>10 Sum/Diff Matrices</u>

S1 = B12 – B22 = 8 – 2 = 6
S2 = A11 + A12 = 1 + 3 = 4
S3 = A21 + A22 = 7 + 5 = 12
S4 = B21 – B11 = 4 – 6 = -2
S5 = A11 + A22 = 1 + 5 = 6
S6 = B11 + B22 = 6 + 2 = 8
S7 = A12 – A22 = 3 – 5 = -2
S8 = B21 + B22 = 4 + 2 = 6
S9 = A11 – A21 = 1 – 7 = -6
S10 = B11 + B12 = 6 + 8 = 13

<u>Doing the Products using the sums</u>

P1 = A11 * S1 = 6
P2 = S2 * B22 = 8
P3 = S3 * B11 = 72
P4 = A22 * S4 = -10
P5 = S5 * S6 = 48
P6 = S7 * S8 = -12
P7 = S9 * S10 = -84

<u>The result sub-matrix values</u>

C11 = P5 + P4 – P2 + P6 = 18
C12 = P1 + P2 = 14
C21 = P3 + P4 = 62
C22 = P5 + P1 – P3 – P7 = 66

From the above, we can clearly see that the answer is the matrix

$$\mathbf{C} = \begin{pmatrix} \mathbf{18} & \mathbf{14} \\ \mathbf{62} & \mathbf{66} \end{pmatrix}$$

5. Exercise 4.2 – 2

STRASSEN MATRIX MULTIPLICATION PSEUDOCODE

```
strassen_matrix_multiply(A, B):
    n = length of A
    C = nxn matrix

    if n == 1:
        c11 = a11 * b11
    else:
        partition A, B, C

        Initiate 10 new matrices, S1, ... S10 of size (n/2)x(n/2)
        Initiate 7 new matrices, P1, ... P7 of size (n/2)x(n/2)

        # Calculating the 10 sum/diff matrices
        S1 = B12 - B22
        S2 = A11 + A12
        S3 = A21 + A22
        S4 = B21 - B11
        S5 = A11 + A22
        S6 = B11 + B22
        S7 = A12 - A22
        S8 = B21 + B22
        S9 = A11 - A21
        S10 = B11 + B12

        # 7 Recursive calls for calculating the products
        P1 = strassen_matrix_multiply(A11, S1)
        P2 = strassen_matrix_multiply(S2, B22)
        P3 = strassen_matrix_multiply(S3, B11)
        P4 = strassen_matrix_multiply(A22, S4)
        P5 = strassen_matrix_multiply(S5, S6)
        P6 = strassen_matrix_multiply(S7, S8)
        P7 = strassen_matrix_multiply(S9, S10)

        # Result sub-matrices values
        C11 = P4 + P5 + P6 - P2
        C12 = P1 + P2
        C21 = P3 + P4
        C22 = P1 + P5 - P3 - P7

    return C
```

6. Exercise 4.2 – 3

When n is not an exact power of 2, we can increase the matrix size to the nearest 2^n value and pad the extra element spaces with the value 0.

Let the new size be **m.** This m will be **atmost double** the value of n because each 2-power increase doubles the existing size.

The run-time can be calculated as,

$$m^{lg7} <= (2n)^{lg7} = 7n^{lg7} \rightarrow O(n^{lg7})$$

$$m^{lg7} >= n^{lg7} \rightarrow \Omega(n^{lg7})$$

Therefore, we get the run-time as $\Theta(n^{lg7})$

---

7. Exercise 4.2 – 6

Let A be a matrix of size ***kn x n***
Let B be a matrix of size ***n x kn***
Let C be the resultant matrix when we multiply both A and B

Resultant Matrix C will be of size ***kn x kn*** which will produce k^2 multiplications of ***n x n*** matrices. Since we can calculate each product in $\Theta(n^{lg7})$ time using Strassen's Algorithm, we can compute the given matrix multiplication in $\Theta(k^2n^{lg7})$ time using Strassen's Algorithm.

REVERSING THE ORDER

Reversing the order of input matrices is the same as calculating *B x A.* In this case the resultant matrix C will be of size ***n x n***.

Here we will be performing only **k** multiplications of ***n x n*** matrices. Hence the resultant matrix C can be calculated in $\Theta(k\,n^{lg7})$ time using Strassen's Algorithm.

---

8. Exercise 4.2 – 7

(a + bi) * (c + di) = ac + ad(i) + bc(i) – bd
= ac – bd + i(ad + bc)

Real Component: ac – bd
Imaginary Component: ad + bc

To calculate the above components, we need to do four multiplications (ac, -bd, ad, bc). But we can do only three multiplications (which we can consider as P1, P2, P3).

Technique: We can reduce addition of 2 multiplications to 1 multiplication and 1 addition, if there exists a common factor between those two multiplications.

We can use the above technique for ac and bc and we can consider it as P1

P1 = (ac) + (bc) = c * (a+b)

To find the real component R,

R = ac – bd
= ac + bc – bd – bc ------ (adding and sub bc)
= P1 – bd – bc
= P1 – b(c+ d) ------ (applying the technique for -bd and -bc)
R = P1 – P2 ------ (considering it as P2)

Likewise, To find the imaginary component Z,

Z = ad + bc
= ad + bc + ac – ac ------ (adding and sub ac)
= ad – ac + P1
= P1 – a(c – d) ------ (applying the technique for ad and -ac)
Z = P1 – P3 ------ (considering it as P3)

Thus, we can calculate the real and imaginary components of two complex numbers just by doing 3 multiplications (P1, P2 and P3). But one thing to note here is that we can end up with different solutions since the solutions can't be unique.

ALGORITHM

```
complex_no_multi(a, b, c, d):
    P1 = c * (a + b)
    P2 = b * (c + d)
    P3 = a * (c – d)

    real = P1 – P2
    imag = P1 – P3

    return real, imag
```

9. Problem 4 – 2

a) 1. Pass by Reference:

Recurrence: $T(n) = T(n/2) + \Theta(1)$

Upper Bound: $T(N) = \Theta(\lg N)$

2. Pass by Copy:

Recurrence: $T(n) = T(n/2) + \Theta(N)$

Upper Bound: $T(N) = \Theta(N \lg N)$

3. Pass by Copy (Subrange):

Recurrence: $T(n) = T(n/2) + \Theta(n/2)$

Upper Bound: $T(N) = \Theta(N)$

---

b) 1. Pass by Reference:

Recurrence: $T(n) = 2T(n/2) + cn$

Upper Bound: $T(N) = \Theta(N \lg N)$

2. Pass by Copy:

Recurrence: $T(n) = 2T(n/2) + cn + 2\Theta(N)$

Upper Bound: $T(N) = \Theta(N \lg N) + \Theta(N^2) = \Theta(N^2)$

3. Pass by Copy (Subrange):

Recurrence: $T(n) = 2T(n/2) + cn + 2c` n/2$

Upper Bound: $T(N) = \Theta(N \lg N)$

---

10. Problem 4 – 6

a) $A[i, j] + A[i+1, j+1] <= A[i, j+1] + A[i+1, j]$ ----- 1

ROWS

Base Case

Let i = 1. Since i < k, k = 2 (k = i + 1)

Inductive Step

We assume that the inductive hypothesis holds true. If equation 1 holds true for k = i + 1, then it should hold true for k + 1 = i + 2.

Replace i by i+1 in Eq 1,

$A[i+1, j] + A[i+2, j+1]$ <= $A[i+1, j+1] + A[i+2, j]$ ----- 2

From 1

$A[i, j] + A[i+1, j+1] – A[i, j+1]$ <= $A[i+1, j]$ ----- 3

Sub 3 in 2

$A[i, j] + \sout{A[i+1, j+1]} – A[i, j+1] + A[i+2, j+1]$ <= $\sout{A[i+1, j+1]} + A[i+2, j]$

$A[i, j] – A[i, j+1] + A[i+2, j+1]$ <= $A[i+2, j]$

$A[i, j] + A[i+2, j+1]$ <= $A[i+2, j] + A[i, j+1]$

This is the same as equation 1 where k = i + 1 is replaced by k + 1= i + 2

Hence by mathematical induction this holds true.

COLUMNS

Base Case

Let j = 1. Since i < l, l = 2 (l = j + 1)

Inductive Step

We assume that the inductive hypothesis holds true. If equation 1 holds true for l = j + 1, then it should hold true for l + 1 = j + 2.

Replace j by j+1 in Eq 1,

$A[i, j+1] + A[i+1, j+2]$ <= $A[i, j+2] + A[i+1, j + 1]$ ----- 2

From 1

$A[i, j] + A[i+1, j+1] – A[i+1, j]$ <= $A[i, j+1]$ ----- 3

Sub 3 in 2

$A[i, j] + \sout{A[i+1, j+1]} – A[i+1, j] + A[i+1, j+2]$ <= $\sout{A[i+1, j+1]} + A[i, j+2]$

$A[i, j] – A[i+1, j] + A[i+1, j+2]$ <= $A[i, j+2]$

$A[i, j] + A[i+1, j+2]$ <= $A[i+1, j] + A[i, j+2]$

This is the same as equation 1 where l = j + 1 is replaced by l + 2 = j + 2

Hence by mathematical induction this holds true.

b) 37 23 22 32

21 06 **05** 10

53 34 30 31

32 13 09 06

43 21 15 08

Change the value of 07 to 05 to make it a Monge array.

---

c) Condition:

$$A[i, j] + A[k, l] <= A[i, l] + A[k, j] \quad \text{-----} \quad 1$$

$$1 <= i < k <= m \quad \text{-----} \quad 2$$

$$1 <= j < l <= n \quad \text{-----} \quad 3$$

f(i) = x (index of the column which contains the minimum element for the given row i) ---- 4

Similarly, f(k) = y ---- 5

Sub j = x and l = y in 1

since j < l → x < y ---- 6

A[i, x] + A[k, y] <= A[i, y] + A[k, x]

Let's assume A[i, x] > A[i, y], then A[k, x] > A[k, y]

If this is true, then f(i) != x

Hence, By Proof of Contradiction,

A[i, x] should be lesser than A[i, y]

Let's assume A[k, x] < A[k, y]

If this is true then f(k) != y

Hence, By Proof of Contradiction,

A[k, x] has to be more than A[k, y]

If f(x) = y and f(i) = x, then f(i) <= f(k) [Since x < y from Eq 6]

Hence, **f(1) <= f(2) <= …… <= f(m)**

---

d) Scan row 1 from index 1 through f(2) and take the minimum element as f(1).
Similarly, scan indices f(2) through f(4) from row 3 for the minimum element.

In general, we scan indices **f(2k)** through **f(2k + 2) of row (2k + 1)** to find the leftmost minimum element of that row.

If m is odd, we'll search indices f(m – 1) through n to find leftmost minimum in row m.
From (c), we know that indices of leftmost minimum elements are increasing and hence we can find the minimum we wanted among the indices scanned.

The total number of comparisions is m + n. Therefore **T(n) = O(m + n)**

---

e) Let T(m, n) denotes the running time of algorithm applied to ***m x n*** matrix.
T(m, n) = T(m/2, n) + c(m + n) for some constant c.
Now we need to show, *T(m, n) <= c(m + nlgm) – 2cm*

T(m, n)	=	T(m/2, n) + c(m + n)
	<=	c(m/2 + nlg(m/2)) – 2cm + c(m + n)
	=	c(m/2 + nlgm) – cn + cn – cm
T(m, n)	<=	c(m + nlgm) – cm

Therefore, By Induction we can say that,

T(m, n) <= O(m + nlgm)
