CSCI 5454 F24 Problem set 1

Jacob Krol

Sep 6 2024

1 Question 1

1.1 Question 1a

Conditions:

- a, b, q, r are positive integers
- $0 < a < b$
- $r < a$

It is provided that division of two integers $a, b : (a < b)$ is

$$\frac{a}{b} = \frac{1}{q + \frac{r}{a}}. \tag{1}$$

which can be rearranged for q

$$q = \frac{b}{a} - \frac{r}{a} \tag{2}$$

Noticing that $0 \leq \frac{r}{a} < 1$, it can be shown $\frac{b}{a} - \frac{r}{a}$ from Eq.2 is equivalent to $\lfloor \frac{b}{a} \rfloor$. Let $n = \frac{b}{a}$ and $\alpha = \frac{r}{a} : 0 \leq \alpha < 1$. Substituting n and α into Eq.2 yields $q = n - \alpha$ which supports the inequality

$$n - 1 < n - \alpha \leq n \tag{3}$$

and satisfies the floor equivalence

$$\lfloor n \rfloor = n - \alpha \tag{4}$$

Recall that $n = \frac{b}{a}$ and $\alpha = \frac{r}{a}$, therefore $\frac{b}{a} - \frac{r}{a}$ from Eq.2 can be re-written using the floor equivalence from Eq.4. And, q can be written solely in terms of a, b as the integer part (quotient) of division $\frac{b}{a}$.

$$q = \lfloor \frac{b}{a} \rfloor \tag{5}$$

r is also solved in terms of a, b

$$r = b - aq \tag{6}$$

$$r = b - a(\lfloor \frac{b}{a} \rfloor) \tag{7}$$

1.2 Question 1b

Algorithm 1 CONTINUED-FRACTION(a,b)

Require: $a < b$
  $arr \leftarrow [\,]$
  $r \leftarrow \infty$
  while $r > 0$ **do**
    $q \leftarrow \lfloor \frac{b}{a} \rfloor$
    $arr \leftarrow APPEND(arr, q)$
    $r \leftarrow b - (a * q)$
    $a \leftarrow r$
    $b \leftarrow a$
  end while
  return arr

Eq.5 and Eq.7 from **Q1a** are applied to $\frac{a}{b}$ retrieving the quotient, q which is stored in arr, and remainder r. $\frac{r}{a}$ of iteration i serves as $\frac{a}{b}$ for iteration $i+1$. Continue until convergence at $r = 0$ when fraction is no longer expandable.

1.3 Question 1c

Algorithm 2 GCD(m,n)

Require: $m > n$
  while $n > 0$ **do**
    $m \leftarrow n$
    $n \leftarrow m\ MOD\ n$
  end while
  return m

Algorithm 3 LCM(m,n)

  return $\lfloor \frac{m*n}{GCD(m,n)} \rfloor$

Algorithm 4 ADD-FRACTIONS(n1,d1,n2,d2)

  $c \leftarrow LCM(d1, d2)$
  $n1 \leftarrow n1 + \lfloor \frac{c}{d1} \rfloor$
  $n2 \leftarrow n2 + \lfloor \frac{c}{d2} \rfloor$
  return $n1 + n2, c$ ▷ return value is numerator, denominator

Final function for **Question 1c** on next page.

Algorithm 5 CONTINUED-FRACTIONS-TO-INTEGERS(arr)

$i \leftarrow 0$
$j \leftarrow 1$
$l \leftarrow LENGTH(arr)$
while $j < l + 1$ **do**
  $a \leftarrow arr[i : j]$ ▷ Stop index is exclusive
  $a \leftarrow REVERSE(A)$
  $n1 \leftarrow 1$
  $d1 \leftarrow a[0]$
  for each k in $a[1 :]$ **do** ▷ From start index onward
    $n1, d1 \leftarrow$ ADD-FRACTIONS$(n1, d1, k, 1)$
  end for
  $j \leftarrow j + 1$
end while
return n1,d1

1.4 Question 1d

```
a=11,b=39
1/3 = 0.3333333333333333
1/4 = 0.25
2/7 = 0.2857142857142857
11/39 = 0.28205128205128205

a=113,b=312
1/2 = 0.5
1/3 = 0.3333333333333333
4/11 = 0.36363636363636365
21/58 = 0.3620689655172414
46/127 = 0.36220472440944884
113/312 = 0.36217948717948717

a=14159265359,b=100000000000
1/7 = 0.14285714285714285
15/106 = 0.14150943396226415
16/113 = 0.1415929203539823
4687/33102 = 0.1415926530119026
4703/33215 = 0.14159265392142104
9390/66317 = 0.1415926534674367
14093/99532 = 0.14159265361893664
37576/265381 = 0.14159265358107778
51669/364913 = 0.14159265359140397
244252/1725033 = 0.14159265358981538
295921/2089946 = 0.14159265359009277
```

```
540173/3814979 = 0.14159265358996734
836094/5904925 = 0.14159265359001172
1376267/9719904 = 0.14159265358999432
2212361/15624829 = 0.1415926535900009
8013350/56594391 = 0.14159265358999976
10225711/72219220 = 0.14159265359
703361698/4967501351 = 0.14159265359
1416949107/10007221922 = 0.14159265359
6371158126/44996389039 = 0.14159265359
14159265359/100000000000 = 0.14159265359
```

2 Question 2

2.1 Question 2a

Storing $s = \lceil\sqrt{n}\rceil$ number of partial sums of A into B will support subarray sum queries in $O(\sqrt{n})$ time. First, A is split into $\lceil\sqrt{n}\rceil$ subarrays $(s_0, \ldots, s_{\lfloor\sqrt{n}\rfloor-1})$ where $\lceil\sqrt{n}\rceil$ defines both the size and number of subarrays (s). An arbitrary $B[k]$ is filled during preprocessing by $\sum_{i=k*s}^{min(n-1,(sk)+s-1)} A[i]$. For example if $A = [1, 2, 3, 4], s = \lceil\sqrt{n}\rceil = 2$:

$$B[0] = \sum_{0*2}^{min(4-1,(2*0)+2-1)} A[i] = \sum_{0}^{1} A[i] = 1 + 2 = 3$$

For $QUERY(l, u)_{A,B}$, the best case scenario is that l and u define the lower and upper bounds of a pre-processed block, $B[k]$; return $B[k]$. A worst case scenario is that l and u are at the $s[1]$ and $s[n-1]$ (zero-indexed) indices of separate blocks, respectively. p and q are subarrays: $|p| = |q| = s - 1$. Continuing the worst case, l and u also happen to index the first ($B[0]$) and last ($B[s-1]$) arrays with s - 2 intermediate pre-computed sums for a total of $2(s-1)+s-2 = 3s-4$ elements. Therefore, by reducing the number of elements to $3s - 4 = 3(\lceil\sqrt{n}\rceil) - 4$ the number of elements to sum given $QUERY(l, u)_{A,B}$ should be asymptotic to $O(\sqrt{n})$.

2.2 Question 2b

To support $O(1)$ lookup of subarray sum queries of an array, A, with size n using a second array, B, also with size n, $B[i]$ can be filled with a partial sums from $A[0]$ to $A[i]$. For example, $A = [1, 2, 3, 4]$:

$$\begin{aligned} B[0] &= 1 \\ B[1] &= 1 + 2 = 3 \\ B[2] &= 1 + 2 + 3 = 6 \\ B[3] &= 1 + 2 + 3 + 6 = 10 \end{aligned}$$

Now, $QUERY(l, u)_{A,B}$ is computed as $B[u] - B[l-1]$ which includes the sum over the range l to u while subtracting the sum of values which prefix the left-hand side of the query (i.e., $-B[l-1]$). Treating the subtraction and lookup as a single operation, the complexity is $O(1)$.

3 Question 3

3.1 Question 3a

Algorithm 6 MAX-SUM-TABLE-B(arr)

$n \leftarrow LENGTH(arr)$
$B \leftarrow TABLE(n, n)$ ▷ Initialize nxn table with arbitrary starting values
for $i = 0$ to $n - 1$ **do**
  for $j = 0$ to $n - 1$ **do**
    $B[i, j] \leftarrow MAX(arr[i : i + j + 1]$ ▷ Stop index is exclusive
  end for
end for
return B

3.2 Question 3bi

The first column of C corresponds to the array unsorted array, A; meaning, there are always n rows. However, the column size is upper bounded by $\log_2(n+1)$:

$$i + 2^j - 1 \leq n \tag{8}$$
$$2^j \leq n - i + 1 \tag{9}$$
$$\log_2 2^j \leq \log_2(n - i + 1) \tag{10}$$
$$j \leq \log_2(n - i + 1) \tag{11}$$

The right-hand side of the inequality is maximized when $i = 0$

$$j \leq \log_2(n - 0 + 1) \tag{12}$$
$$j \leq \log_2(n + 1) \tag{13}$$

In summary, the first column contains all elements $A[i] \in A$; this defines the row size of C as the length (n) of A. The column size is at most $\log_2 n + 1$ (+1 is negligible for large sizes of n. Therefore, the table size is asymptotic to $\Theta(n \log n)$.

3.3 Question 3bii

The recurrence of $C[i, j+1]$ in terms of $C[i, j]$ $C[i + 2^j, j]$ is

$$C[i, j+1] = max(C[i, j], C[i + 2^j, j]) \tag{14}$$

I observed this empirically for various tables, but the logical basis is not obvious to me. Below is an example output from my program.

```
array: [5, -9, 13, -7, 3, 1, 5, -20]
C table:
[[  5   5  13  13]
 [ -9  13  13   0]
 [ 13  13  13   0]
 [ -7   3   5   0]
 [  3   3   5   0]
 [  1   5   0   0]
 [  5   5   0   0]
 [-20   0   0   0]]
recurrence
n 8
C[i, j+1] 5  C[i,j] 5  C[i + 2^j, j] -9  i 0  j 0
C[i, j+1] 13  C[i,j] 5  C[i + 2^j, j] 13  i 0  j 1
C[i, j+1] 13  C[i,j] 13  C[i + 2^j, j] 5  i 0  j 2
C[i, j+1] 13  C[i,j] -9  C[i + 2^j, j] 13  i 1  j 0
C[i, j+1] 13  C[i,j] 13  C[i + 2^j, j] 3  i 1  j 1
C[i, j+1] 13  C[i,j] 13  C[i + 2^j, j] -7  i 2  j 0
C[i, j+1] 13  C[i,j] 13  C[i + 2^j, j] 3  i 2  j 1
C[i, j+1] 3  C[i,j] -7  C[i + 2^j, j] 3  i 3  j 0
C[i, j+1] 5  C[i,j] 3  C[i + 2^j, j] 5  i 3  j 1
C[i, j+1] 3  C[i,j] 3  C[i + 2^j, j] 1  i 4  j 0
C[i, j+1] 5  C[i,j] 3  C[i + 2^j, j] 5  i 4  j 1
C[i, j+1] 5  C[i,j] 1  C[i + 2^j, j] 5  i 5  j 0
C[i, j+1] 5  C[i,j] 5  C[i + 2^j, j] -20  i 6  j 0
```

3.4 Question 3biii

Algorithm 7 MAX-SUM-TABLE-C(arr)

$n \leftarrow LENGTH(arr)$
$C \leftarrow TABLE(n, \lceil \log_2(n+1) \rceil)$
for $i = 0$ to $n-1$ **do** $j \leftarrow 0$
  while $i + 2^j \leq n$ **do**
    $stop \leftarrow i + 2^j$
    $C[i,j] \leftarrow MAX(arr[i : stop]$
    $j \leftarrow j + 1$
  end while
end for
return C

3.5 Question 3c

Algorithm 8 LOOKUP-C(arr)

$j \leftarrow \lceil \log_2(u - l + 1) \rceil$
return $MAX(C[l, j], C[l + j - 1, j]$