CSCI 5454 F24 Problem set 1

Jacob Krol

Sep 6 2024

Question 1a.
Conditions:

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

The continued fraction formula rewrites integers a, b in terms of a quotient (q), the integer multiple of a which divides b, and the integer remainder of the division (r).

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

Rearranging for q, it's clear that q represents the integer portion of $\frac{b}{a}$ as $\frac{r}{a}$ (the remainder/modulus) is subtracted. This can also be written using the floor function.

$$q = \frac{b}{a} - \frac{r}{a} = \lfloor \frac{b}{a} \rfloor$$

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

Solve for r

$$\frac{qa + r}{b} = 1$$
$$qa + r = b$$
$$r = b - qa$$
$$r = b - a(\lfloor \frac{b}{a} \rfloor)$$

Q1b

```
FUNCTION CONT_FRACT(a, b)
    arr = []
    r = INFINITY

    WHILE r > 0 DO
        q = FLOOR(b / a)
        APPEND q TO arr
        r = b - (a * q)
        a,b = r,a
    END WHILE

    RETURN arr
```

2-3 line explanation substituting both $q = \lfloor \frac{b}{a} \rfloor$ and $r = b - a\lfloor \frac{b}{a} \rfloor$ into $\frac{1}{q+\frac{r}{a}}$ recursively yeilds continued fraction forms of the latest $\frac{a}{b}$. Recursion ends once $r \leq 0$ since $\frac{r}{a}$ cannot be divided further. Quotients are stored to represent integers $a_1 ... a_n$ of the continued fraction.

Q1c

```
FUNCTION GCD(m, n)
    WHILE n > 0
        m, n = n, m MOD n
    END WHILE
    RETURN m

FUNCTION LCM(m, n)
    RETURN (m * n) // GCD(m, n)

FUNCTION ADD_FRACS(n1, d1, n2, d2)
    lcd = LCM(d1, d2)
    n1 = n1 * (lcd // d1)
    n2 = n2 * (lcd // d2)
    RETURN n1 + n2, lcd

FUNCTION CONT_FRAC2INTS(arr)
    i = 0
    j = 1
    WHILE j < LENGTH(arr) + 1
        a = SLICE(arr, i, j)
        REVERSE(a)
        n1 = 1
        d1 = a[0]
        FOR EACH k IN a[1:] DO
            n1, d1 = ADD_FRACS(n1, d1, k, 1)
```

```
            n1, d1 = d1, n1
        END FOR
        j = j + 1
    END WHILE
    RETURN n1, d1
```

Q1d

```
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
```

Question 2

Q2a

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 $([s_0, ..., s_{s-1}])$ 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 $QUERY(l, u)_{A,B}$, a 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 $3s-4 = 3(\lceil\sqrt{n}\rceil)-4 \approx O(\sqrt{n})$ the number of elements to sum given $QUERY(l, u)_{A,B}$ should be asymptotic to $O(\sqrt{n})$.

Q2b

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 $O(1)$.

Q3biii

```
# arrays are zero-indexed
# array indices are inclusive
BEGIN max_sub_arr_tbl(A)
n = LENGTH A
C = ARRAY[n,n]
FOR i = 0 UPTO n
    FOR j = 0 UPTO n
        IF i + (2 POWER j) <= n
            C[i,j] = MAX(A[i TO i+j])
```

```
        else:
            C[i,j] = NULL
RETURN C
END max_sub_arr_tbl(A)
```