Lab 07

I. El-Shaarawy & M. Touny

December 3, 2023

Contents

Exercises

7.1.2

Homework.

7.1.5

We tell students sort by a single loop rather than a single-line.

```
def sortBySingleLoop(A[0..n-1])
    # initialize a zeros list of size n
    S = [0] * n

    # loop on A values, Convert to corresponding index, Set that index
    for i in 0..n-1
        S[ A[i]-1 ] = A[i]

    # S is A but sorted
    return S
```

7.1.10

Homework.

7.2.2

Homework.

7.2.3

Homework.

7.2.11

a

Hints

- The question asks for memory. So any timely inefficient solution is acceptable.
- Use naive brute-force.

Solution

```
# input: strings S[0..n-1] and T[0..n-1]
# output: True if and only if T is right cyclic shift
def bruteForceRightCyclicShift(S[0..n-1], T[0..n-1])

    # try all ith positions
    for i in 0..n-1
```

```
        # counter of matched characters
        k = 0

        # check from the ith position to last nth character, cycling if needed
        while k < n and S[(i+k) mod n] = T[k]
            k = k + 1

        # all n characters are matching, i.e strings are matching
        if k == n return True

    # if no position matches
    return False
```

If the `mod` operation is troublesome to students, we show

```
# input: position x
# output:
    # x if x did not pass string length n
    # if x passed n, return only the additional length beyond n
def myPosition(x, n)
    if x < n
        return x
    return x - n

# input: strings S[0..n-1] and T[0..n-1]
# output: True if and only if T is a right cyclic shift of S
def bruteForceRightCyclicShift(S[0..n-1], T[0..n-1])

    # try all ith positions
    for i in 0..n-1

        # counter of matched characters
        k = 0

        # check from the ith position to last nth character, cycling if needed
        while k < n and S[ myPosition(i+k,n) ] = T[k]
            k = k + 1

        # all n characters are matching, i.e strings are matching
        if k == n return True

    # if no position matches
    return False
```

Observe greatest value of `x` is $n-1+n-1 = 2n-2 < 2n$. So our function `myPosition` is equivalent to `mod` operation in this case.

Complexity. Extra space is $\mathcal{O}(1)$. Time is $\mathcal{O}(n^2)$.

b

Hints

- Use *Boyer-Moore* algorithm as a subroutine.
- What is the input enhancement so that a linear scan, of all possible positions, of first character, is feasible?
- Repeat the input so the check is equivalent to cycling.

Solution

```
# input: string S
# output: S but with n-1 prefix appended
def appendPrefix(S[0..n-1])

    # copy S
    X = S

    # for each character of n-1 prefix
    for i in 0..n-2

        # append to the end
        X.append( X[i] )

    # return appended string
    return X

# input: string S[0..n-1] and T[0..n-1]
# output: True if and only if T is a cyclic right shift of S
def BoyerMooreRightCyclicShift(S[0..n-1], T[0..n-1])

    # enhance the input by appending n-1 prefix
    S = appendPrefix(S)

    # right cyclic shift is equivalent to matching T in enhanced input S
    return BoyerMoore(S, T)
```

For enhanced X of given input S, Observe $S[i \mod n] = X[i]$. In other words, our condition on the enhanced input is equivalent to the brute-force algorithm. Since we know the brute-force is correct by definition, so is `BoyerMooreRightCyclicShift`.

Complexity.

- Time. $\Theta(n)$ for appending prefix. $\mathcal{O}(n)$ for *Boyer-Moore algorithm* (given from the levitin).

- Space. Extra space is $\Theta(n)$ for appended prefix. $\Theta(|\sum|)$ for the *good-suffix table*. $\Theta(n)$ for the *bad-symbol table*.

7.3.1

Homework.

7.3.2

Homework.

7.3.4

Given the even distribution of hash function, We have a uniform distribution. Fixing cell c_j the probability of hashing to it is $Pr[R_i = c_j] = \frac{1}{m}$ for the ith element out of the n elements. Since the hash events are pairwise independent, $Pr[C = c_j] = Pr[R_1 = c_j \wedge \cdots \wedge R_n = c_j] = Pr[R_1 = c_j] \cdot .. \cdot Pr[R_n = c_j] = \left(\frac{1}{m}\right)^n$. Since the events of hashing all elements to a particular cell are disjoint, $Pr[C = c_0 \vee \cdots \vee C_{m-1}] = Pr[C = c_0] + \cdots + Pr[C = c_{m-1}] = m\left(\frac{1}{m}\right)^n = \frac{1}{m^{n-1}}$.