Final Examination Problem Sheet

TIME: 05/31/2022, 13:20–16:20

- This is an open-book online exam. You can use any materials that you are able to find, but **you cannot communicate with anyone other than the TAs and the instructors** during the exam. Any form of cheating, lying or plagiarism will not be tolerated. Students can get zero scores and/or get negative scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconduct.
- Both English and Chinese (if suited) are allowed for answering the questions. We do not accept any other languages.
- Unless explicitly asked, you do **NOT** need to justify why your algorithm runs in the desired time/space complexity, but you are welcomed to briefly explain the key ideas to help the TAs grade your solution.
- There are 16 problems. 5 of them are marked with * and are supposedly simple(r); 7 of them are marked with ** and are supposedly regular; 4 of them are marked with *** and are supposedly difficult. The simpler ones are worth 10 points, and others are worth 15 points. The full credit is 215 points.
- We will allow you to use at most one A4 page in your PDF for each problem. For most problems, you do not need a full A4 page. So you can put multiple problems in one A4 page as well (to save your time in scanning). But all solution lines to one problem must be on the same page. In addition, you must associate your solution to the right problem by tagging on Gradescope. As long as there is a clear indication of which problem you are solving and the right tagging, you do not need to sequentially answer by the order of problems in the PDF.
- You should keep your answers as *concise* as possible to facilitate grading. In particular, please use proper *pseudo code* and/or sufficiently-understandable words to describe your algorithm to the TAs clearly!

String Matching

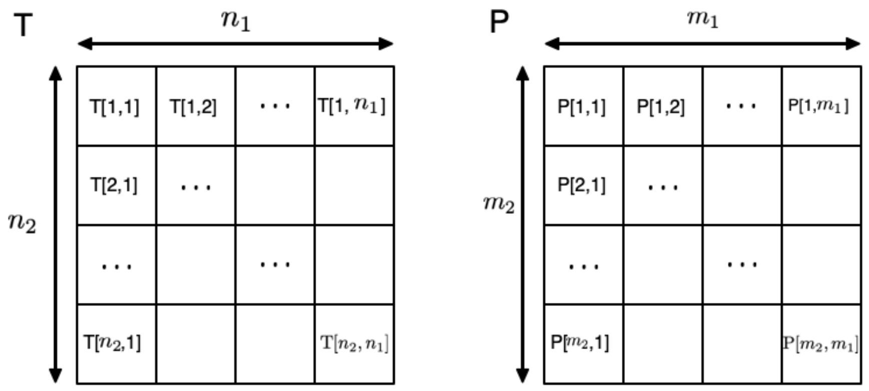

Figure 1: 2-dimensional pattern matching

In the following few questions, we consider a 2-dimensional (2D) pattern matching problem, which is quite similar to the 1-dimensional (1D) string matching problem covered in the lecture. You are given a 2D array T of size (n_2, n_1) and array P of size (m_2, m_1), as shown in Figure 1. The size of P is not larger than T, i.e., $m_1 \leq n_1$, $m_2 \leq n_2$. The elements of P and T are drawn from the character set

$\Sigma = \{0, 1, 2, \ldots, 255\}$, and the size of Σ is $d = |\Sigma| = 256$. Similar to the 1D string matching problem, the 2D pattern matching problem is to find all valid shifts (s_2, s_1) such that $T[s_2 + 1..s_2 + m_2, s_1 + 1..s_1 + m_1] == P[1..m_2, 1..m_1]$. We will now leverage the techniques from the Rabin-Karp algorithm to develop a $O(n_1 n_2)$-time algorithm to solve the problem.

1. (*) We first calculate a hash number p to represent the 2D pattern P. p is given by

$$\begin{aligned} p =& \sum_{i=1}^{m_2} \left[d^{(i-1)m_1} \left(\sum_{j=1}^{m_1} d^{m_1-j}\ P[i,j] \right) \right] \qquad \pmod q \\ =& d^0(d^{m_1-1}P[1,1] + d^{m_1-2}P[1,2] + \cdots + d^0P[1,m_1]) + \\ & d^{m_1}(d^{m_1-1}P[2,1] + d^{m_1-2}P[2,2] + \cdots + d^0P[2,m_1]) + \\ & d^{2m_1}(d^{m_1-1}P[3,1] + d^{m_1-2}P[3,2] + \cdots + d^0P[3,m_1]) + \\ & \cdots + \\ & d^{(m_2-1)m_1}(d^{m_1-1}P[m_2,1] + d^{m_1-2}P[m_2,2] + \cdots + d^0P[m_2,m_1]) \qquad \pmod q \end{aligned} \tag{1}$$

Similar to the 1D Rabin-Karp, we will perform all calculation with modulo q. Please write down pseudo code to perform this calculation in $O(m_1 m_2)$ time and using only $O(1)$ additional space.

2. (**) We can define a hash number t_{s_2,s_1} to present shift (s_2, s_1), i.e., corresponding to pattern $T[s_2 + 1..s_2 + m_2, s_1 + 1..s_1 + m_1]$. t_{s_2,s_1} is given by

$$\begin{aligned} t_{s_2,s_1} =& \sum_{i=1}^{m_2} \left[d^{(i-1)m_1} \left(\sum_{j=1}^{m_1} d^{m_1-j}\ T[s_2+i, s_1+j] \right) \right] \qquad \pmod q \\ =& d^0(d^{m_1-1}T[s_2+1,s_1+1] + d^{m_1-2}T[s_2+1,s_1+2] + \cdots + d^0T[s_2+1,s_1+m_1]) + \\ & d^{m_1}(d^{m_1-1}T[s_2+2,s_1+1] + d^{m_1-2}T[s_2+2,s_1+2] + \cdots + d^0T[s_2+2,s_1+m_1]) + \\ & d^{2m_1}(d^{m_1-1}T[s_2+3,s_1+1] + d^{m_1-2}T[s_2+3,s_1+2] + \cdots + d^0T[s_2+3,s_1+m_1]) + \\ & \cdots + \\ & d^{(m_2-1)m_1}(d^{m_1-1}T[s_2+m_2,s_1+1] + d^{m_1-2}T[s_2+m_2,s_1+2] + \cdots + d^0T[s_2+m_2,s_1+m_1]) \\ & \qquad \pmod q \end{aligned} \tag{2}$$

Obviously we cannot start the calculation of this hash number from scratch with every possible shift value, as we know from the last problem that it would take $O(m_1 m_2)$ time. To be able to calculate the hash number efficiently, we define $t_{\text{col}}(i,j)$ that is a sum of partial hash values contributed from a column of size $(m_2, 1)$ in the pattern, with the top of the column at element $T[i,j]$, as follows:

$$t_{\text{col}}(i,j) = T[i,j] + d^{m_1}T[i+1,j] + d^{2m_1}T[i+2,j] + \cdots + d^{(m_2-1)m_1}T[i+m_2-1,j], \tag{3}$$

for $1 \le i \le n_2 - m_2 + 1$ and $1 \le j \le n_1$.

Write down an equation to show how to calculate t_{s_2,s_1} from the values of $t_{\text{col}}(i,j)$, assuming $t_{\text{col}}(i,j)$ is known for any valid values of i and j.

3. (**) Using the result from the last problem, if t_{s_2,s_1} is known, show how we can derive t_{s_2,s_1+1} in constant time, assuming $t_{\text{col}}(s_2+1, j)$ is known for any $1 \le j \le n_1$.

4. (**) Assuming $t_{\text{col}}(i,j)$ is known. Show how we can calculate $t_{\text{col}}(i+1,j)$ in constant time.

5. (***) Complete the following RABIN-KARP-MATCHER-2D pseudo code and show that indeed it has $O(n_1 n_2)$ running time. For simpler proof, you may assume there is no spurious hit and one valid shift for the given T and P.

```
RABIN-KARP-MATCHER-2D($T, P, d, q$)
 1  $(n_1, n_2) = T.size$
 2  $(m_1, m_2) = P.size$
 3  // blank 1, calculate several powers of $d$ for later use
 4  $p = 0$
 5  $t = 0$
 6  // use the results from problem 1 here to calculate $p$
 7
 8  // next we will initialize an array of size $n_1$, $tcol[]$, to store $T_{\text{col}}(1, j), 1 \le j \le n_1$.
 9  for $j = 1$ to $n_1$
10      $tcol[j] = 0$
11      for $i = 1$ to $m_2$
12          // blank 2, fill in to calculate $tcol[j]$
13  for $s_2 = 0$ to $n_2 - m_2$
14      // use the results from problem 2 to initialize $t = t_{s_2,0}$
15      for $s_1 = 0$ to $n_1 - m_1$
16          if $p == t$
17              if $T[s_2 + 1..s_2 + m_2, s_1 + 1..s_1 + m_1] == P[1..m_2, 1..m_1]$
18                  print "Pattern occurs with shift" $(s2, s1)$
19              if $s_1 < n_1 - m_1$
20                  // use the results from problem 3 to update $t = t_{s_2,s_1+1}$
21      if $s_2 < n_2 - m_2$
22          // use the results from problem 4 to update $tcol[]$
23          // from $T_{\text{col}}(s_2 + 1, j)$ to $T_{\text{col}}(s_2 + 2, j)$.
```

6. (*) Compute the prefix function π for the pattern `ababbabbabbababbabb`.

7. (**) Some argue that in line 12 of KMP-MATCHER (see the pseudo code below, identical to the one on the textbook) is incorrect and should be $q = 0$ instead. Please give a pair of T and P which would produce incorrect result with this modification. Please explain how they would generate the incorrect result.

```
KMP-MATCHER(T,P)
 1  $n = T.length$
 2  $m = P.length$
 3  $\pi$ = COMPUTE-PREFIX-FUNCTION(P)
 4  $q = 0$
 5  for $i = 1$ to $n$
 6      while $q > 0$ and $P[q + 1] \ne T[i]$
 7          $q = \pi[q]$
 8      if $P[q + 1] == T[i]$
 9          $q = q + 1$
10      if $q == m$
11          print "Pattern occurs with shift" $i - m$
12          $q = \pi[q]$
```

```
COMPUTE-PREFIX-FUNCTION(P)
$m = P.length$
let $\pi[1..m]$ be a new array
$\pi[1] = 0$
$k = 0$
for $q = 2$ to $m$
    while $k > 0$ and $P[k+1] \neq P[q]$
        $k = \pi[k]$
    if $P[k+1] == P[q]$
        $k = k + 1$
    $\pi[q] = k$
return $\pi$
```

Linear-Time Sorting

8. (**) Complete the ALT-COUNTING-SORT procedure by filling the blanks on line 8 and line 11, without changing the existing parts of the code. At the end of the function, the output array $B[]$ should contain the sorted numbers originally in the input array $A[]$ in ascending order. See additional specifications at the top of the pseudo code below. In addition, your ALT-COUNTING-SORT algorithm should be *stable*.

```
ALT-COUNTING-SORT$(A, B, k, n)$
 1  // Input array is given as $A[0..n-1]$
 2  // Output array should be given as $B[0..n-1]$
 3  // the values in $A$ and $B$ are nonnegative integers no larger than $k$
 4  let $C[0..k+1]$ be a new array
 5  for $i = 0$ to $k+1$
 6      $C[i] = 0$
 7  for $j = 0$ to $n-1$
 8      // blank
 9  $C[0] = 0$
10  for $i = 1$ to $k$
11      C[i]= // blank
12  for $j = 0$ to $n-1$
13      $B[C[A[j]]] = A[j]$
14      $C[A[j]] = C[A[j]] + 1$
```

Disjoint Sets

9. (***) Assume the use of linked-list representation of disjoint sets (section 21.2 in the textbook, or method three in the lecture slides). n is the number of elements in the sets, i.e., the number of performed MAKE-SET operations. Now, in addition to MAKE-SET, FIND-SET, and UNION operations, we also need to support PRINT-SORTED which prints out all elements in a set in a sorted manner. PRINT-SORTED should only take $O(n)$ time. Show that a sequence of m operations consisting of MAKE-SET, FIND-SET, and UNION (but without PRINT-SORTED) would take $O(m + n^2)$. (Hint: weighted union does not improve the running time to $O(m + n \log n)$ in this case)

Hashing

Suppose you are given the following key numbers: (17, 6, 83, 57, 23, 15, 28, 45, 120), and a hash function $h_1(x) = x \mod 11$. We ask you to insert these numbers *in the given order* into an initially empty hash table with 11 entries, each having space to hold one key.

10. (*) Assume *chaining* is used to resolve overflow. Show the content of the hash table after all numbers are inserted.

11. (**) Assume *double hashing* is used to resolve overflow. Here the second hash function is given by $h_2(x) = 1+(x \mod 12)$. The hash function for double hashing is given by $h(x,i) = (h_1(x)+i \cdot h_2(x)) \mod 11$, where i indicates the index in the corresponding probe sequence, $0 \leq i < 11$.

Red-Black Tree

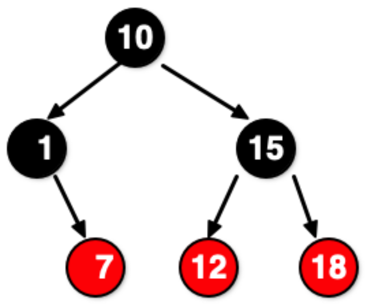

Figure 2: A red-black tree

For the following questions, you are NOT required to show how each operation is performed *in a step-by-step manner*; only *the outcome of each operation is required.* However, you are welcomed to show some intermediate steps to help the grading TA to understand how the operations are performed.

12. (***) Given the red-black tree shown in Figure 2, show the results after the following operations are performed in the given order: (1) insert 25; (2) insert 9; (3) insert 20.

13. (***) After performing the operations in the last question, continue with the following operations in the given order and show *all possible results* after each operation: (4) delete 7; (5) delete 10.

B-Tree

For the following B-tree related problems, you may want to look at the pseudo code of B-Tree-Search shown below, which is identical to the one in the textbook.

```
B-Tree-Search($x, k$)
1   $i = 1$
2   while $i \leq x.n$ and $k > x.key_i$
3       $i = i + 1$
4   if $i \leq x.n$ and $k == x.key_i$
5       return $(x, i)$
6   elseif $x.leaf$
7       return NIL
8   else
9       Disk-Read($x.c_i$)
10      return B-Tree-Search($x.c_i, k$)
```

14. (*) Show that if we use *binary search* instead of *linear search* within each node when searching a B-Tree (line 1-3 in B-Tree-Search), the CPU time requires $O(\log_2 n)$, independent of the minimum degree t. (Note: the original linear-search version has $O(t \log_t n)$ CPU time)

15. (**) Does the use of binary search change the number of disk accesses from $O(\log_t n)$? Please justify your answer. In reality, the use of binary search does not produce much improvement in actual running time, which accounts for both CPU time and disk read time. Please give an explanation of why this is the case.

Feedback

16. (*) Please give some constructive suggestions to the course this semester. In particular, what we did right this semester, and what we should change in the future? Your feedback is particularly important for us to improve the course, almost the entire semester was largely impacted by COVID-19. We hope most of you can kindly write a few sentences at least (plus it counts the same points as other problems). Thank you!