Final Examination Problem Sheet

Time: 13:20 - 16:20, Tuesday, June 15, 2021

PLEASE READ BEFORE STARTING TO WORK ON THE PROBLEMS

- **[Important] Complete the online sign-in process on NTU COOL (see instructions there). Only those who completed the sign-in process will receive score for the final exam.**
- This is an **open-book online exam**. You can use any printed or online materials as your reference during the exam.
- During the exam, **no communication with others, online or physically,** is allowed.
- 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 misconducts.
- Both English and Chinese (if suited) are allowed for answering the questions. We do not accept any other languages.
- There are **16** problems. Each problem is worth **10** points. The full credit is **160** points.
- Please submit your answer sheets on Gradescope to the assignment "Final Exam (submitted by students)". **Submission deadline is 16:20. Late submission is allowed until 17:00, but a penalty of 4 points per minute late will be applied to your exam score.** Only the last submission will be graded.
- Clearly **mark the solution of each problem with its problem ID**, similar to how it was done for the three homework assignments. Use separate pages for different problems. **Provide reference(s) for each and every problem.** Failure to comply to those instructions would result in **a penalty of up to five points per problem**, at the discretion of the grading TA. **Double check before you submit.**
- Problems marked with (*) are supposedly simpler, problems marked with (**) are supposedly regular, and problems marked with (***) are supposedly slightly difficult.
- You should keep your answers as *concise* as possible to facilitate grading. In particular, when applicable, please use proper *pseudo code* to describe your algorithm!

1 B-Tree

1. (***) Figure 1 shows a B-Tree with minimum degree $t = 2$. The keys are sorted in alphabetical order. Using the algorithm outlined in chapter 18.2 and 18.3 in the textbook, please show how the B-Tree is changed after *each* of the following operations are performed in the given order: (1) insert K; (2) insert L; (3) delete K ;(4) delete N; (5) delete I.

Below you can find the pseudo code of the three functions related to B-Tree insertion: B-Tree-Insert, B-Tree-Insert-Nonfull, B-Tree-Split-Child. These are from chapter 18.2 in the textbook.

2. (**) A student *Yui* argues that line 11 in B-Tree-Insert-Nonfull, i.e., $i = i + 1$ is not required and the code would work correctly without it. Do you agree with her? Please explain.
3. (**) Another student *Gen* wonders why line 2 - 9 in B-Tree-Insert looks redundant to line 13 - 17 in B-Tree-Insert-Nonfull, and argues that the former can be removed without affecting the correctness of the algorithm. Do you agree with him? Please explain.

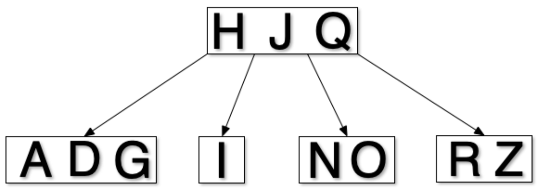

Figure 1: A B-Tree with minimum degree $t = 2$

```
B-TREE-INSERT($T, k$)
$r = T.root$
if $r.n == 2t - 1$
    $s =$ ALLOCATE-NODE()
    $T.root = s$
    $s.leaf =$ FALSE
    $s.n = 0$
    $s.c_1 = r$
    B-TREE-SPLIT-CHILD($s, 1$)
    B-TREE-INSERT-NONFULL($s, k$)
else B-TREE-INSERT-NONFULL($r, k$)
```

```
B-TREE-INSERT-NONFULL($x, k$)
$i = x.n$
if $x.leaf$
    while $i \geq 1$ and $k < x.key_i$
        $x.key_{i+1} = x.key_i$
        $i = i - 1$
    $x.key_{i+1} = k$
    $x.n = x.n + 1$
    DISK-WRITE($x$)
else while $i \geq 1$ and $k < x.key_i$
        $i = i - 1$
    $i = i + 1$
    DISK-READ($x.c_i$)
    if $x.c_i.n == 2t - 1$
        B-TREE-SPLIT-CHILD($x, i$)
        if $k > x.key_i$
            $i = i + 1$
    B-TREE-INSERT-NONFULL($x.c_i, k$)
```

```
B-TREE-SPLIT-CHILD($x, i$)
$z$ = ALLOCATE-NODE()
$y = x.c_i$
$z.leaf = y.leaf$
$z.n = t - 1$
for $j = 1$ to $t - 1$
    $z.key_j = y.key_{j+t}$
if not $y.leaf$
    for $j = 1$ to $t$
        $z.c_j = y.c_{j+t}$
$y.n = t - 1$
for $j = x.n + 1$ downto $i + 1$
    $x.c_{j+1} = x.c_j$
$x.c_{i+1} = z$
for $j = x.n$ downto $i$
    $x.key_{j+1} = x.key_j$
$x.key_i = y.key_t$
$x.n = x.n + 1$
DISK-WRITE($y$)
DISK-WRITE($z$)
DISK-WRITE($x$)
```

2 Graph

4. (*) Modify the following BFS pseudo code (from chapter 22.2 in the textbook) to determine the number of connected components in undirected graph G. Your modified function should output a single number which is the number of connected components in $G = (V, E)$. Analyze the running time of your modified BFS and give its asymptotic notation in terms of $|V|$ and $|E|$.

```
BFS($G, s$)
for each vertex $u \in G.V - \{s\}$
    $u.color$ = WHITE
    $u.d = \infty$
    $u.\pi$ = NIL
$s.color$ = GRAY
$s.d = 0$
$s.\pi$ = NIL
$Q = \emptyset$
ENQUEUE($Q, s$)
while $Q \neq \emptyset$
    $u$ = DEQUEUE($Q$)
    for each $v \in G.Adj[u]$
        if $v.color$ == WHITE
            $v.color$ = GRAY
            $v.d = u.d + 1$
            $v.\pi = u$
            ENQUEUE($Q, v$)
    $u.color$ = BLACK
```

5. (*) Consider a different way to find the number of connected components in an undirected graph using the *disjoint set* data structure, as shown in chapter 21.1, using COUNT-CONNECTED-COMPONENTS

shown below. You can assume the disjoint set data structure utilizes weighted-union and path compression. Analyze the running time of COUNT-CONNECTED-COMPONENTS in terms of $|V|$ and $|E|$ using asymptotic notation.

```
COUNT-CONNECTED-COMPONENTS($G$)
$n = 0$
for each vertex $v \in G.V$
    MAKE-SET($v$)
    $n = n + 1$
for each edge $(u, v) \in G.E$
    if FIND-SET($u$) $\neq$ FIND-SET($v$)
        UNION($u, v$)
        $n = n - 1$
print $n$
```

6. (***) In the class, we have learned that we can use BFS to find the shortest path from a starting vertex to all other vertices in a graph. However, this is limited to the case where the edges have equal weight, e.g., 1. For the general case where edges have different weights, Dijkstra's algorithm (which you will learn in Algorithm Design and Analysis class soon, if you are a CSIE student) can find the shortest path from a single starting vertex to all other vertices in $\Theta(|E| + |V| \log |V|)$-time.

   However, here we consider a special case, where there are only two different edge weights $w(e)$, 3 and 6. That is, $\forall e \in E, w(e) \in \{3, 6\}$. Denote $|E_3|$ as the number of edges with weight value 3 and $|E_6|$ as the number of edges with weight value 6. $|E| = |E_3| + |E_6|$. Show that you can beat Dijkstra's algorithm in term of running time, using BFS to solve the special case shortest path problem. Analyze the running time of your algorithm in terms of $|V|, |E_3|$ and $|E_6|$.

7. (*) Prove that when performing DFS on an undirected graph, cross edge cannot exist.

8. (**) Modify the following DFS pseudo code, as given in chapter 22.3 in the textbook, to determine whether there is a cycle in the given directed graph G. Your pseudo code should output a single boolean, TRUE or FALSE, to indicate whether a cycle exists in G. (Hint: out of the four types of edges, which, if exists in the input graph of DFS, would result in a cycle?)

```
DFS($G$)
for each vertex $u \in G.V$
    $u.color$ = WHITE
    $u.\pi$ = NIL
$time = 0$
for each vertex $u \in G.V$
    if $u.color$ == WHITE
        DFS-VISIT($G, u$)
```

```
DFS-VISIT($G, u$)
$time = time + 1$
$u.d = time$
$u.color$ = GRAY
for each $v \in G.Adj[u]$
    if $v.color$ == WHITE
        $v.\pi = u$
        DFS-VISIT($G, v$)
$u.color$ = BLACK
$time = time + 1$
$u.f = time$
```

3 String Matching

9. Answer the following questions:

   (a) (*) Working modulo $q = 17$, how many *spurious hits* does Rabin-Karp-Matcher encounter in the text $T = 3141592653589793$ when looking for the pattern $P = 41592$? Here, assume the set of alphabets $\Sigma = \{0, 1, 2, \ldots, 9\}$ is used for T and P. Since $d = |\Sigma| = 10$, here you can use a length-k decimal number (as the hash value) to represent a string of k consecutive characters. (The pseudo code is on page 993 of the textbook; you can also look for it in the course slides.)

   (b) (*) Assume the prefix function $\pi[q]$ is defined as described in the class for the KMP string matching algorithm. Give a string whose prefix function $\pi[] = \{0, 0, 1, 2, 3, 1, 2, 0, 1, 2, 3, 4\}$.

10. (**) In the KMP-Matcher below (as given in chapter 32.4 in the textbook), why can't we change line 12 to $q = 0$ instead? Please explain with an example - a pair of input T and P which would give incorrect result if $q = 0$ is used.

```
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 $P[q+1] \neq T[i]$
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         $q = \pi[q]$
```

4 Hashing

11. (*) Answer each of the following questions with a few sentences.

    (a) What is uniform hashing?

    (b) Give an example to show that quadratic probing prevents primary clustering better than linear probing.

12. (*) Answer each of the following questions with a few sentences.

    (a) When is chaining a better collision resolution scheme than open addressing in terms of load factor? Please explain.

    (b) What is the main advantage of dynamic hashing schemes compared to standard hashing tables?

13. (**) Figure 2 gives a directoryless hash table with parameters $r = 2, q = 0$. Each table entry can hold two elements. Assume a hash function $h(k, i) = k \ \& \ ((1 \ll i) - 1)$, i.e., the hash function outputs the i least significant bits. Show how the content of the hash table and the values of r and q are changed after *each* of the following insertion is performed: (1) insert value `0x2E`; (2) insert value `0xA2`; (3) insert value `0x08`; (4) insert value `0xB0`. Note that the keys are given as hexadecimal number.

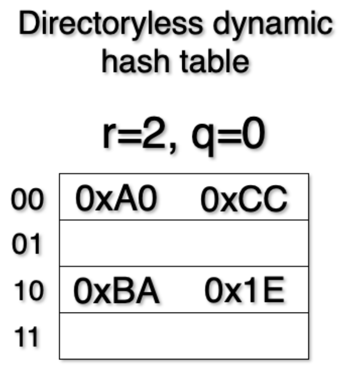

Figure 2: A directoryless dynamic hash table

5 Sorting

14. (**) Complete the ALT-COUNTING-SORT procedure by filling the three blanks on line 6, 8 and 10, without changing other parts of the code. Out of the input parameters, $A[]$ is the input array with index starting from 1 and ending in $A.length$. $B[]$ is the output array, with index starting from 1 and ending in $A.length$, and should contain sorted array in ascending order at the end of the function. The values in array $A[]$ range between 0 and input parameter k. (These are the same as the original COUNTING-SORT)

    Note that you are allowed to write multiple lines of code in the blanks. You can leave the blank of line 6 empty, but not the others. In addition, similar to the original COUNTING-SORT in the textbook, your ALT-COUNTING-SORT algorithm should be *stable*.

```
ALT-COUNTING-SORT($A, B, k$)
1   let $C[0..k]$ be a new array
2   for $i = 0$ to $k$
3       $C[i] = 0$
4   for $j = 1$ to $A.length$
5       $C[A[j]] = C[A[j]] + 1$
6   //Fill the blank here
7   for $i = k - 1$ downto 0
8       //Fill the blank here
9   for $j = 1$ to $A.length$
10      //Fill the blank here
```

6 Red-Black Tree

15. (*) A node with key x is inserted into an existing red-black tree, and then *without* performing any other deletion or insertion operations, the very same node with key x is deleted. Would the resulting red-black tree always be the same as before x is inserted? Justify your answer.

7 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, as the second half of the 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!