Data Structure and Algorithm, Spring 2017
Final Examination

Date: Tuesday, June 20, 2017

Time: 13:20-16:20pm (180 minutes)

138 points

Non-Programming problems

Problem 1. In each of the following question, please specify if the statement is **true** or **false**. If the statement is true, explain why it is true. If it is false, explain what the correct answer is and why. (18 points. 1 point for true/false and 2 points for the explanation for each question)

1. In a red-black tree, the number of red nodes in the path from the root to ANY leaf node is the same.
2. In a red-black tree, the number of nodes in the path from the root to ANY leaf node is the same.
3. In a B-tree, the number of nodes in the path from the root to ANY leaf node is the same.
4. The only person who can close a bug report is the person who fix the bug.
5. When using evidence-based scheduling to estimate the time to complete the tasks in a software project, it is essential that the engineers' estimates of time to complete the tasks are very accurate.
6. Counting sort is a stable sorting algorithm.

Problem 2. Fill the blanks / short answer questions. (26 points)

1. What are the three essential things that should be in every bug report? (3 points)
2. Complete the COUNTING-SORT procedure by filling the blank on line 9. Note that you are allowed to write multiple lines of code in the blank. (4 points)

3. Assuming you have completed the blank in the previous problem and now COUNTING-SORT works correctly. Subsequently, if line 8 of COUNTING-SORT is changed to **for** $j = 1$ to $A.length$, will the COUNTING-SORT still work currently? Briefly explain why. (4 points)

4. Give 3 major functions that a version control system do. (3 points)

5. Draw the result of inserting R,Y,F,X,A,M,C,D,E,T,H,V,L,W,G, in order, to an initially empty B-tree. The minimum degree of the B-tree is 3. (6 points)

6. Draw the result of deleting C, P, and V, in order, from the B-tree shown in Figure 2. The minimum degree of the B-tree is 3. (6 points)

```
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  for $i = 1$ to $k$
7      $C[i] = C[i] + C[i-1]$
8  for $j = A.length$ downto 1
9      //Fill the blank here
```

Problem 3. Graph and Depth-First Search (32 points)

Figure 1 shows a directed graph $G = (V, E)$ with 10 vertices, labeled by letters from q to z. Answer the following related questions.

1. Perform depth-first search (DFS) on G using the pseudo code procedures DFS and DFS-VISIT shown below, and show the final results. Assume the for loop of lines 5-7 of DFS considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Mark the discovery and finishing times, and the parent next to each vertex on Figure 1. (10 points)

2. We can define four edge types in terms of the depth-first forest $G_\pi = (V, E_\pi)$ produced by a depth-first search on a graph, where E_π is the set of edges traversed by the search when a vertex is first visited:

(a) **Tree edges** are edges in the depth-first forest, i.e., E_π.

(b) **Back edges** are the edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree.

(c) **Forward edges** are the nontree edges (u, v) connecting to a vertex u to a descendant v in a depth-first tree.

(d) **Cross edges** are all other edges, which can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees.

Based on the solution to the last problem, mark the type of each edge in G. (7 points)

3. Explain why edge (u, v) is a back edge if and only if $v.d \leq u.d < u.f \leq v.f$. (note that you have to explain both directions) (4 points)

4. Explain why there is no back edge, if DFS is performed on an undirected graph. (3 points)

5. Give the running times of DFS in terms of $|V|$ and $|E|$, when using adjacency lists and adjacency matrix to represent the graph, respectively. (4 points)

6. Rewrite the procedure DFS-VISIT to use a stack and avoid the recursive function call on line 7. (4 points)

```
DFS($G$)
1  for each vertex $u \in G.V$
2      $u.color$ = WHITE
3      $u.\pi$ = NIL
4  $time = 0$
5  for each vertex $u \in G.V$
6      if $u.color$ == WHITE
7          DFS-VISIT($G, u$)
```

```
DFS-VISIT($G, u$)
 1  $time = time + 1$
 2  $u.d = time$
 3  $u.color =$ GRAY
 4  for each $v \in G.Adj[u]$
 5      if $v.color ==$WHITE
 6          $v.\pi = u$
 7          DFS-VISIT($G$,$v$)
 8  $u.color =$ BLACK
 9  $time = time + 1$
10  $u.f = time$
```

Problem 4. Heap (20 points)

The procedure BUILD-MAX-HEAP makes use of MAX-HEAPIFY in a bottom-up manner to convert the array $A[1\,..\,n]$, where $n = A.length$, into a max-heap.

1. Please show that BUILD-MAX-HEAP takes $O(n)$ time to complete. (6 points)
2. TA Hsun came up with a different method to build a max heap - inserting the elements one by one into an initially empty max heap. The method is implemented as BIULD-MAX-HEAP-I. Please show that this method can in fact take up to $\Omega(n \log n)$ to complete in the worst case. (6 points)
3. Give an example to show that BUILD-MAX-HEAP and BUILD-MAX-HEAP-I can generate different heaps. (4 points)
4. Is heapsort a stable algorithm? If yes, please prove that it is stable. If not, please give a counter-example. (4 points)

```
PARENT($i$)
1  return $\lfloor i/2 \rfloor$

LEFT($i$)
1  return $2i$

RIGHT($i$)
1  return $2i+1$
```

```
MAX-HEAPIFY($A, i$)
 1  $l$ = LEFT($i$)
 2  $r$ = RIGHT($i$)
 3  if $l \le A.\mathit{heap\text{-}size}$ and $A[l] > A[i]$
 4      $\mathit{largest} = l$
 5  else
 6      $\mathit{largest} = i$
 7  if $r \le A.\mathit{heap\text{-}size}$ and $A[r] > A[\mathit{largest}]$
 8      $\mathit{largest} = r$
 9  if $\mathit{largest} \neq i$
10      exchange $A[i]$ with $A[\mathit{largest}]$
11      MAX-HEAPIFY($A, \mathit{largest}$)
```

```
BUILD-MAX-HEAP($A$)
1  $A.\mathit{heap\text{-}size} = A.\mathit{length}$
2  for $i = \lfloor A.\mathit{length}/2 \rfloor$ downto 1
3      MAX-HEAPIFY(A,$i$)
```

```
MAX-HEAP-INSERT(A,$key$)
1  $A.heap\text{-}size = A.heap\text{-}size + 1$
2  $i = A.heap\text{-}size$
3  $A[i] = key$
4  while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$
5      exchanage $A[i]$ with $A[\text{PARENT}(i)]$
6      $i = \text{PARENT}(i)$
```

```
BUILD-MAX-HEAP-I($A$)
1  $A.heap\text{-}size = 1$
2  for $i = 2$ to $A.length$
3      MAX-HEAP-INSERT($A, A[i]$)
```

Problem 5. Hashing (12 points)

Consider inserting the keys $10, 22, 31, 4, 15, 28, 17, 88, 59$ into a hash table of length $m = 11$ using the following methods. Show the results after the insertion of each number (i.e., for each method, there will be 9 drawings).

1. Linear hashing (3 points)
2. Quadratic probing with $c_1 = 1$ and $c_2 = 3$ (3 points)
3. Double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \mod (m - 1))$ (3 points)
4. Chaining (3 points)

Programming problem

Problem 6. N-way Merge (30%)

In a typical merge sort, the merging phase is to combine two sorted sub-arrays into one where all elements are sorted properly. That's what we call a "2-way merge". In this problem, you are asked to merge N sorted arrays of integers, i.e., an "N-way merge". You will be given N sorted arrays, each with M integers. Please output all $N \times M$ integers in ascending order.

I/O Format

Input

The first line contains an integer `T`, the number of test cases. For each test case, the first line contains two integers, N_i and M_i, and N_i lines follow, each with M_i integers.

Output

Please output all $N_i \times M_i$ integers in ascending order in a single line for each test case.

Input Constraints

It is guaranteed that:

- $1 \leq N_i, M_i \leq 10^4$, for all test cases.
- All $N_i \times M_i$ integers are between -2^{31} to $2^{31} - 1$
- $1 \leq \sum_i N_i \times M_i \leq 10^6$

Sample input

```
2
3 3
1 3 5
2 4 6
10 11 12
5 2
1 2
3 4
5 6
7 8
9 10
```

Sample output

```
1 2 3 4 5 6 10 11 12
1 2 3 4 5 6 7 8 9 10
```

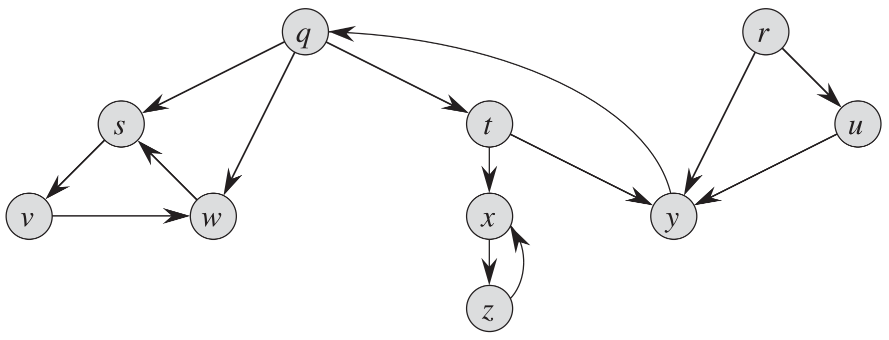

Figure 1: A directed graph

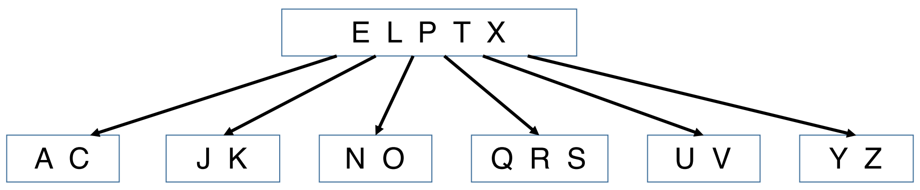

Figure 2: A B-Tree