Practice Final Exam: Part I and II (Solutions)

Date: December 6, 2023

DIRECTIONS:

- Write your answers on the exam paper.
- If you need extra space, please use the back of a page.
- You are allowed one cheat sheet.
- You have 80 minutes to complete the exam.
- Please do not turn the exam over until you are instructed to do so.
- Good Luck!

1. MAXIMUM(A) procedure iterates over an array to find the largest integer in the array.

   Input: A is a input array with indices $A[6, 4, 1, 2, 10, 3, 5]$
   Output: 10

   MAXIMUM(A)

   ```
   1  maxval = A[1]
   2  for i = 2 to A.length
   3      if A[i] ≥ maxval
   4          maxval = A[i]
   5  return maxval
   ```

   Here is a loop invariant for MAXIMUM procedure:

   > Before j-th iteration, $maxval$ is greater than or equal to all elements in the subarray $A[1, . . , j - 1]$.

   Use the loop invariant to prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

   Solution:

   - Initialization/Base case: Before the $j = 2$-th iteration, the subarray $A[1, 2, ...j - 1] = A[1]$, which is an array with single element, therefore, trivially, $maxval = A[1]$, establishing that the loop invariant holds for the base case.
   - Maintenance/Base case: We assume that before j-th iteration, $maxval$ is greater than or equal to all elements in the subarray $A[1, . . j - 1]$. In j-th iteration, with line 3 and 4, we update the value of $maxval$ if $A[j]$ is greater than the current $maxval$, otherwise, we stay with the same value. By the end of the j-th iteration, we have $maxval$ greater than or equal to $A[1 . . j]$, making it ready for $(j + 1)$-th iteration. **This step shows that the loop invariant is preserved while moving from j-th iteration to $(j + 1)$-th iteration.**
   - **Here, the loop terminates when** $j = n + 1$, where n is the size of the array. When the loop terminates, $A[1 . . n + 1 - 1] = A[1 . . n]$. As per the maintenance step, before $(n + 1)$-th iteration, $maxval$ will be greater than or equal to all elements in $A[1 . . n]$, which is the desired result.

   Notes:

   - In order to prove the correctness of an algorithm using loop invariant, we must show that the loop invariant is preserved throughout the loop, before the loop starts till the loop terminates.
   - In the base case/initialization, we show that the loop invariant is preserved by finding the subarray before the loop starts. In this particular algorithm, the loop starts at $j = 2$, therefore, the subarray is $A[1, . . 2 - 1] = A[1]$. Note that the subarray is $A[1 . . j - 1]$ in the loop invariant above.

- In the maintenance step/inductive step, we show that the loop invariant is preserved while we move from j-th iteration to the start of $(j + 1)$-th iteration. While doing so, we essentially show that with each iteration, the loop invariant is preserved. **Once we show that the base case holds for the loop invariant condition, we can assume that the loop invariant is true for before some j-th iteration.** Then, we use the assumption before the j-th iteration to establish the loop invariant is also true/preserved before $(j + 1)$-th iteration using the lines within the body of loop.
- Finally, we need to show the loop invariant is preserved when the loop terminates.

(Please use this page to write your answer for question (2) if the previous page is not enough.)

2. Use either substitution method or a recursion tree to show that $T(n) = 2T(n/2) + n^2$, and $T(1) = 1$ and $T(2) = 6$ is $T(n) = O(n^2)$.

   Substitution method

   Base case: Given $T(1) = 1$, we can calculate $T(2) = 2 * 1 + 2^2 = 6 < c \cdot 2^2 = 4c$, which holds the base case for $c \geq 2$.

   Inductive step:

   Hypothesis: We assume that for $m < n$, $T(m) \leq cm^2$ is true. As per the recurrence relation we have, we choose $m = n/2 < n$, therefore, $T(n/2) \leq c(n/2)^2$.

   Substituting our assumption, which is true, in our recurrence relation, in order to calculate $T(n)$, we get:

$$\begin{aligned} T(n) &\leq 2 \cdot c(n/2)^2 + n^2 \\ &= c(n^2)/2 + n^2 \\ &= (c/2 + 1)n^2 \\ &\leq cn^2 \quad \text{for } c \geq 2 \end{aligned}$$

   Since we have shown that our hypothesis $T(n/2) \leq c(n/2)^2$ led to the **same exact form**, $T(n) \leq cn^2$ for $c \geq 2$, it proves that $T(n) \leq cn^2$ for $c \geq 2$, eventually, implying that $T(n) = O(n^2)$.

   ** *Correction*: There was an error in the answer to this question in practice exam 01, where in the substitution step, $T(n) \leq c(n/2)^+n^2$, which should be $T(n) \leq 2c(n/2)^+n^2$ instead.

   Recursion tree method

   The answers to recursion tree method has already been answered in practice exam 01 (Question 3).

3. Priority Queues

   (a) For a priority queue data structure, write pseudocode for HEAP-EXTRACT-MAX(S), where S is a set of elements, each with an associated value called a ***key***.

   Solution:

   The pseudocode to this problem is in Chapter 6 Heapsort, pg. 163 CLRS, 3rd Edition.

   (b) State an upper bound for the runtime of HEAP-EXTRACT-MAX(S) procedure. Justify your answer.

   Solution:

   In the pseducode/procedure HEAP-EXTRACT-MAX(S), from line 1 to 6 takes constant time. MAX-HEAPIFY($A, 1$) takes $O(\lg n)$, where n is the size of the heap, therefore, the upper bound for the runtime of HEAP-EXTRACT-MAX(S) is $O(\lg n)$.

4. What is the running time of QUICKSORT when all elements of array A have the same value?

   Solution:

   If all the elements in array A are the same, the partitioning will have $T(n) = T(n-1) + T(0) + \Theta(n)$ recurrence runtime relation. Because for each pivot, $A[r]$, $(n-1)$ elements will be less than or equal to $A[r]$ for a subarray of length n, leaving the other parition to be empty. Solving $T(n) = T(n-1) + \Theta(n)$ is $\Theta(n^2)$, therefore, the runtime of QUICKSORT is $\Theta(n^2)$, when all elements of array A have the same value.

5. Write a procedure called LIST-DELETE removes an element x from a linked list L. A pointer to x is given.

   Solution:

   The pseudocode/procedure is in Chapter 10 pg. 238, CLRS, 3rd Edition.

6. Write a procedure to find the predecessor of a key, x in a binary-search tree. State the upper bound of the runtime to this procedure.

```
TREE-PREDECESSOR(x)
1  if x.right ≠ NIL
2      return TREE-MAXIMUM(x.left)
3  y = x.p
4  while y ≠ NIL and x == y.left
5      x = y
6      y = y.p
7  return y
```

7. A Catalan number is recursively defined as follows:
   $C_0 = 1$, $C_1 = 1$, and $C_n = \sum_{i=0}^{n-1} C_i * C_{n-i-1}$.
   Provide a $O(n^2)$ dynamic-programming procedure to calculate the n-th Catalan number.

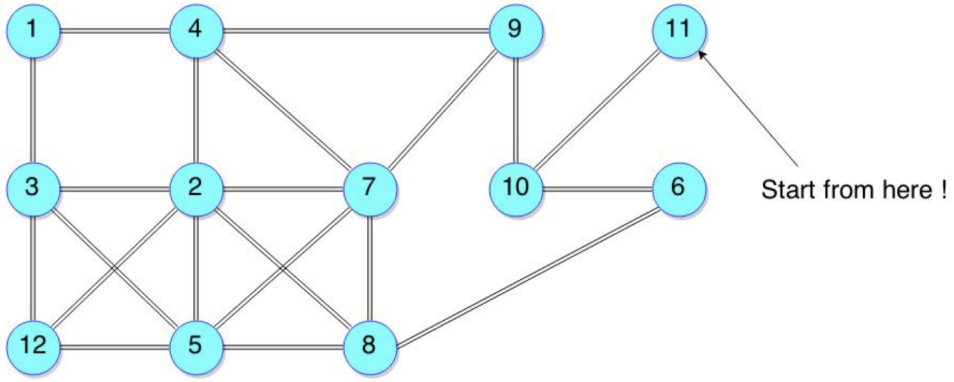
   Top-Down Approach

```
CATALAN-NUMBER(memo, n)
 1  if n ≤ 1
 2      return n
 3  if n ∈ memo
 4      return memo[n] // if n has already been calculated and stored.
 5  else
 6      res = 0
 7      for i = 0 to n − 1
 8          res = res + CATALAN-NUMBER(i) * CATALAN-NUMBER(n − i − 1)
 9      memo[n] = res
10      return res
```

   Bottom-up Approach

```
CATALAN-NUMBER(n)
1  let C be a new array of size n
2  C[0] = 1
3  C[1] = 1
4  for i = 2 to n
5      res = 0
6      for j = 0 to i − 1
7          res = res + C[j] * C[i − j − 1]
8      C[i] = res
9  return C[n]
```

8. Run BFS (Breadth-first search) procedure on the following undirected graph and write out the $v.d$ and $v.\pi$ for each vertex in the graph. Here $v.d$ is the **distance** from the source node, and $v.\pi$ is the predecessor of a node. The source node is 11 as stated in graph below.



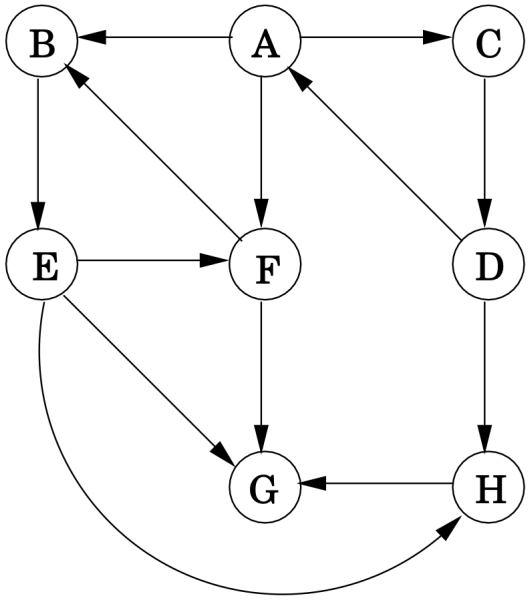
vertex	1	2	3	4	5	6	7	8	9	10	11	12
$v.d$	4	4	5	3	4	2	3	3	2	1	0	5
$v.\pi$	4	8	2	9	8	10	9	6	10	11	NIL	2

Table 1: Caption

Note: Although the question has not mentioned about a queue, BFS procedure uses a queue while exploring the neighbors for each dequeued node. You should use a queue as a used in the illustration used in the book to keep track of the nodes.

9. DFS

(a) Run DFS (Depth-first search) procedure on the following directed graph and write out the $v.d$ and $v.f$ for each vertex in the graph. Here, $v.d$ is the discovery time, and $v.f$ is the finishing time. The source node here is node A.

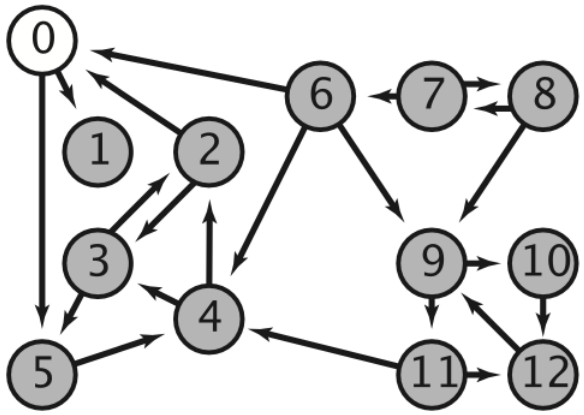


(b) Write out the topological ordering of the nodes based on the DFS run in the previous question.

Note: There is no topological ordering for this graph, because it's not a directed acyclic graph (DAG). If there was no edge going from D to A, then, a topological ordering is possible.

vertex	$v.d$	$v.f$
A	1	16
B	2	11
C	12	15
D	13	14
E	3	10
F	4	7
G	5	6
H	8	9

10. Find the strongly connected components in the following graph using the STRONGLY-CONNECTED-COMPONENTS procedure.



Solution:

The strongly connected components are:

- {7, 8}
- {6}
- {9, 12, 10, 11}
- {0, 2, 3, 4, 5}
- {1}

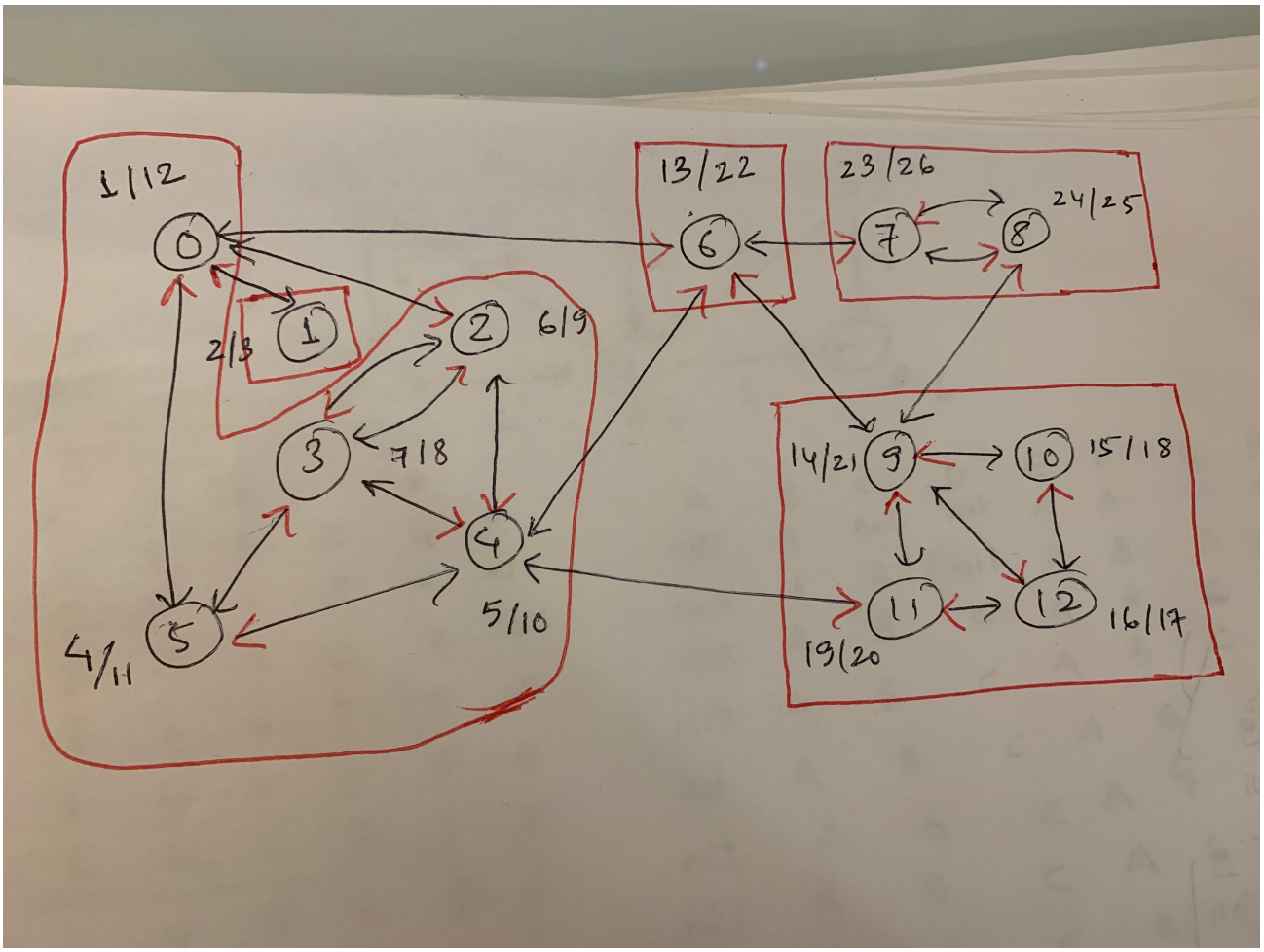
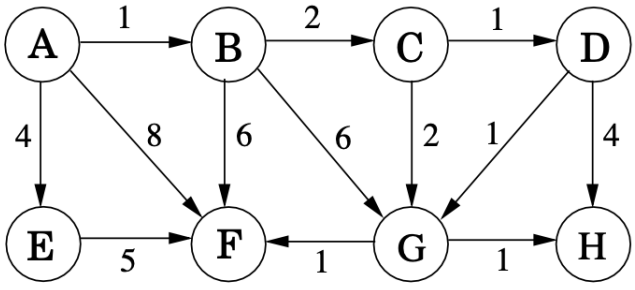


Figure 1: After running the first DFS, we have their discovery and finishing times logged here. Then the arrows are reversed, and we run the DFS on the transposed graph based on the decreasing finishing times to get the DFS trees, which are the strongly connected components.

11. Run DIJKSTRA procedure in the following graph, with the source node A:



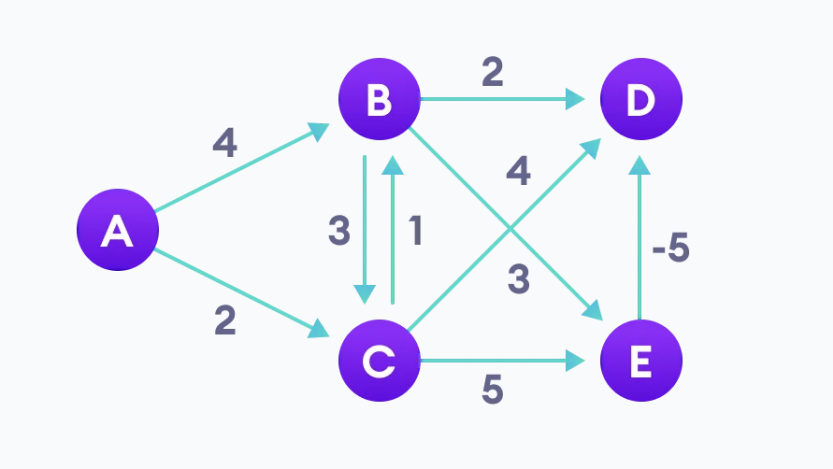
A	B	C	D	E	F	G	H
0	∞	∞	∞	∞	∞	∞	∞
0	1	∞	∞	4	8	∞	∞
0	1	3	∞	4	7	7	∞
0	1	3	4	4	7	5	∞
0	1	3	4	4	7	5	8
0	1	3	4	4	7	5	8
0	1	3	4	4	6	5	6
0	1	3	4	4	6	5	6
0	1	3	4	4	6	5	6

Table 2: d values

A	B	C	D	E	F	G	H
NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
NIL	A	NIL	NIL	A	A	NIL	NIL
NIL	A	B	NIL	A	B	B	NIL
NIL	A	B	C	A	B	C	NIL
NIL	A	B	C	A	B	C	D
NIL	A	B	C	A	B	C	D
NIL	A	B	C	A	G	C	G
NIL	A	B	C	A	G	C	G
NIL	A	B	C	A	G	C	G

Table 3: π values

12. Run BELLMAN-FORD procedure in the following graph, with the source node A.



Here is the order of the edges in $G.E$ for the graph G above:

1. E - D

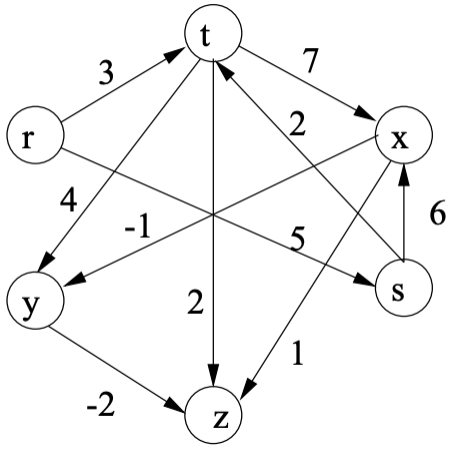
2. C - E
3. C - D
4. B - E
5. B - D
6. B - C
7. C - B
8. A - B
9. A - C

A	B	C	D	E
0	∞	∞	∞	∞
0	4	2	∞	∞
0	3	2	6	7
0	3	2	2	6
0	3	2	1	6

Table 4: d values

A	B	C	D	E
NIL	NIL	NIL	NIL	NIL
NIL	A	A	NIL	NIL
NIL	C	A	C	C
NIL	C	A	E	B
NIL	C	A	E	B

13. Run DAG-SHORTEST-PATHS procedure in the following graph, with the source node r



After running DFS on the graph:

vertex	$v.d$	$v.f$
r	1	12
s	2	11
t	3	10
x	4	9
y	5	8
z	6	7

Based on the finishing times, we order the nodes in topological order.

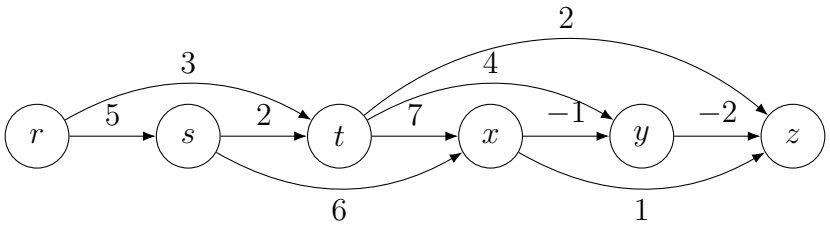


Figure 2: Topologically sorted graph

r	s	t	x	y	z
0	∞	∞	∞	∞	∞
0	5	3	∞	∞	∞
0	5	3	11	∞	∞
0	5	3	10	∞	∞
0	5	3	10	7	5
0	5	3	10	7	5
0	5	3	10	7	5

Table 5: d values

r	s	t	x	y	z
NIL	NIL	NIL	NIL	NIL	NIL
NIL	r	r	NIL	NIL	NIL
NIL	r	r	s	NIL	NIL
NIL	r	r	t	t	t
NIL	r	r	t	t	t
NIL	r	r	t	t	t
NIL	r	r	t	t	t

Table 6: π values

14. Run a bottom-up LCS-LENGTH procedure on the following two sequences to find the longest common subsequence using a table.

    $Y = CGATAATTGAGA$
    $X = GTCCTAATA$

		0	1	2	3	4	5	6	7	8	9	10	11	12
		y_j	C	G	A	T	A	A	T	T	G	A	G	A
0	x_i	0	0	0	0	0	0	0	0	0	0	0	0	0
1	G	0	0↑	↖1	←1	←1	←1	←1	←1	←1	↖1	←1	↖1	←1
2	T	0	0↑	1↑	↑1	↖2	←2	←2	↖2	↖2	←2	←2	←2	←2
3	C	0	↖1	1↑	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2	↑2	↑2
4	C	0	↖1	1↑	↑1	↑2	↑2	↑2	↑2	↑2	↑2	↑2	↑2	↑2
5	T	0	1↑	1↑	↑1	↖2	↑2	↑2	↖3	↖3	←3	←3	←3	←3
6	A	0	1↑	1↑	↖2	↑2	↖3	↖3	↑3	↑3	↑3	↖4	←4	↖4
7	A	0	1↑	1↑	↖2	↑2	↖3	↖4	←4	←4	←4	↖4	↑4	↖5
8	T	0	1↑	1↑	↑2	↖3	↑3	↑4	↖5	↖5	←5	←5	←5	↑5
9	A	0	1↑	1↑	↖2	↑3	↖4	↖4	↑5	↑5	↑5	↖6	←6	↖6

We simply follow the arrows from $[9, 12]$ in order to print out the longest common subsequence. Following the arrows, we get $GTAATA$. The length of LCS is 6.