National University of Singapore

School of Computing

IT5003 - Data Structures and Algorithms
Final Assessment

(Saturday, 05 December 2020, AM)

Time Allowed: 2 hours

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this final assessment paper until you are told to do so.
2. This assessment paper contains THREE (3) sections.
   It comprises SEVEN (7) printed pages, including this page.
3. This is an **Open Book/Open Laptop/Open PC Assessment**.
   But you are NOT allowed to use the Internet (web browser, messaging/cloud services, etc).
   You are free to use your Laptop/PC as you see fit *without* accessing the Internet other than for Zoom e-proctoring and to upload the (scanned) soft copy answer at the end of the paper.
4. There are FIVE (5) pages of answer sheets (**Answer Sheets.docx**, not password protected).
   If you have printer, you can print the blank answer sheets earlier.
   If you don't have printer, just mimic the format on FIVE (5) blank pages as best as you can.
   Answer **ALL** questions within the **given (boxed) space** to make grading easier.
   When you write your answers, you can do so using either pen or pencil, just write **legibly**!
   Scan the printed Answer Sheets and upload the scan to LumiNUS files at the end of the paper.
   Note that if you prefer to write your answers digitally, we can also type your answers at **Answer Sheets.docx** and simply upload this softcopy to LumiNUS files at the end of the paper.
5. Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
   Read all the questions first! Some (subtask) questions might be easier than they appear.
6. You can use **pseudo-code** in your answer but beware of penalty marks for **ambiguous answer**.
   You can use **standard, non-modified** classic algorithm in your answer by just mentioning its name, e.g. run DFS on graph G, Dijkstra's on graph G', etc.
7. All the best :)

A Basic Python and Time Complexity Analysis (5x4 = 20 marks)

For each question below, write a short *one liner* Python code at the commented part (that starts with ###) to complete the program (3 marks). (You will -1 mark if your code is correct but you need more than one line). Afterwards, write the big-O time complexity of the short program and with a short explanation (1 mark). An example A-Q0 is shown below.

```
# A-Q0, Sample Input = "7", Sample Output = "12"
n = int(input())
### print the sum of even integers in [0/1/../n-1]
# answer for A-Q0 should be: print(sum(range(0, n, 2)))
print(sum(range(0, n, 2)))
# the time complexity is O(n), range and sum goes through n (over 2) integers
```

```
# A-Q1, Sample Input = "1 4 2 6 9 5 3 8 7 10", Sample Output = "8"
L = list(map(int, input().split())) # there are n (7 <= n <= 100000) integers in L
### print the first (0-based) index in L with value 7  print(L.index(7))
```

```
# A-Q2, Sample Input = "1 4 2 6 9 5 3 8 7 10", Sample Output = "[9, 8, 7, 10]"
L = list(map(int, input().split())) # there are n (7 <= n <= 100000) integers in L
### update list L by removing integers in L that is < 7, but preserve the order
print(L)    L = [n for n in L if n >= 7]
```

```
# A-Q3, Sample Input = "1 4 2 6 9 5 3 8 7 10", Sample Output = "4,5,6,7,8,9,10"
L = list(map(int, input().split())) # there are n (7 <= n <= 100000) integers in L
L.sort()
### print seven largest integers in L (comma separated, no [], non-decreasing)
```

print(L[-7:])

```
# A-Q4, Sample Input = "1 4 2 6 9 5 3 8 7 10", Sample Output = "4,5,6,7,8,9,10"
from heapq import heapify, heappop
# there are n (7 <= n <= 100000) integers in PQ
PQ = list(map(lambda x: -int(x), input().split())) # store negative x instead
heapify(PQ) # __doc__ 'Transform list into a heap, in-place, in O(len(heap)) time.'
### print seven largest integers in PQ (comma separated, no [], non-decreasing)
```

print(",".join(map(str, sorted([-heappop(PQ) for _ in range(7)]))))

```
# A-Q5, Sample Input = "2 3 2 1 1 3 2 8 7 3 6", Sample Output = "4,5,9"
# there are n (7 <= n <= 100000) integers in input, but each integer is in [1..9]
s = set(map(int, input().split()))
### print 1-digit integer(s) not in s (comma separated, no [], non-decreasing)
```

print(",".join(map(str, [n for n in range(1,10) if n not in s])))

B Theory Questions (45 marks)

B.1 List/Stack/Queue ADT (7x2 = 14 marks)

For each question below, fill in the blank with the best short answer (2 marks each).

1. The easiest index in a Python `list` of n integers that can be removed in $O(1)$ is index n-1.

2. The easiest index in a Python `list` of n integers for inserting a new integer in $O(1)$ is index n.

3. The fastest way (i.e., in $O(n)$) to reverse a Python `list` of n integers is reverse().

4. One of the best way to implement efficient Stack ADT is to use Python `list` with its right side as the top of the Stack.

5. The easiest indices in a Python `deque` of n integers that can be removed in $O(1)$ are indices n-1 and 0.
   For deque, remove and insert from the both boundary side of the queue costs O(1)

6. The easiest indices in a Python `deque` of n integers for inserting a new integer in $O(1)$ are indices n-1 and 0.

7. One of the best way to implement efficient Queue ADT is to use deque with its front/back as the front/back of the Queue.

B.2 Binary Heap Questions (3x2 = 6 marks)

For each question below, fill in the blank with the best short answer (2 marks each).

1. The height of a *complete binary tree* of n integers is log_2^N.

2. Binary Heap data structure is one of the best way to implement priority queue ADT because it can be used to `enqueue(new-value)` and `extract-max-value()` in efficient $O(\log n)$.

3. Binary Heap data structure can be implemented by using its one-to-one mapping with a compact Python `list` that ignores index 0: index 1 is the root, index 2/3 are the left/right child of the root, respectively, and so on.

   This way, we can navigate from index i to its left child ('L') / right child ('R') / parent ('P') by simply computing $(i * 2)$ / $(i * 2 + 1)$ / $(i//2)$, respectively.

   If starting from the root, we arrive at index 4 via an 'LRPL' path.

   Now, where will be be if we start from the root and use path 'RLLPLRPPRL' instead?
   Answer: At index 26.

B.3 Hash Table Questions (3x3 = 9 marks)

We implement a Hash Table of size m to store N Integers.
We use hash function $h(v) = v\%m$.
We resolve collision using *Separate Chaining*.

1. The Hash Table is initially empty and $m = 17$.
   Which sequence of $k = 7$ integers if inserted one by one will result in at least one of the chain to have length ≥ 3? Give a short justification!

   a). {1, 2, 3, 4, 5, 6, 7}

   b). {17, 34, 18, 35, 19, 36, 20}

   c). {32, 47, 88, 93, 23, 47, 22}

   d). {77, 25, 10, 34, 27, 44, 70}

   e). None of the above

   Calculate each sequence's hash value,
   Only option D has three same hash value

2. The Hash Table is initially empty and $m = 17$.
   Propose a sequence of $k = 7$ integers if inserted one by one will result in at least one of the chain to have length 7? Give a short justification! {17,34,51,68,85,102,119} will have a length 7 chain on hash value 0

3. We have at most $N \leq 10\,000$ integers to be inserted into the Hash Table.
   There can be some future deletions but not guaranteed.
   Which is the best setting for m? Give a short justification!

   a). $m = 17$

   b). $m = 97$

   c). $m = 3331$

   d). $m = 5000$

   e). None of the above

   Select M:
   1. Select a big PRIME number
   2. Not near a power of 2
   3. Load factor alpha = N/M < 0.5
   按照优先级排列

   Since 5000 is not a prime number, so the largest prime number 3331 is chosen.

B.4 Binary Search Tree Questions (3x2 = 6 marks)

For each question below, fill in the blank with the best short answer (2 marks each).

1. Without the self-balancing ability in the advanced version of Binary Search Tree (BST), the height of a possibly unbalanced BST of n distinct integers can be as tall as N-1 (in terms of number of edges from root to the deepest leaf). Without self-balancing, the tallest height can be the number of leaves, which is n-1, root is not included

2. The smallest integer in a BST of $n \geq 2$ distinct integers can be found by starting from root and keep going to the left subtree until we reach a vertex without a left child. Now the question is: how to find the second smallest integer in a BST? Move to the smallest integer's right child, if no right child, then the smallest's parent

3. How to find the median integer inside a BST of n distinct integers (assume $n \geq 3$ and n is odd so the median is clearly defined: larger than $(n-1)//2$ integers and smaller than the other $(n-1)//2$ integers)? Starting from the biggest integer, find biggest (n-1) integers, then the next biggest integer node is the median. Same way for starting from the smallest integer.

ChatGPT: From root node, visit each left child and push it to a stack. When there is no left child, pop a node then visit its right child. When the stack counter achieved (n+1)/2, the current visiting node is the meidan

B.5 Basic SSSP Questions (3+3+2+2 = 10 marks)

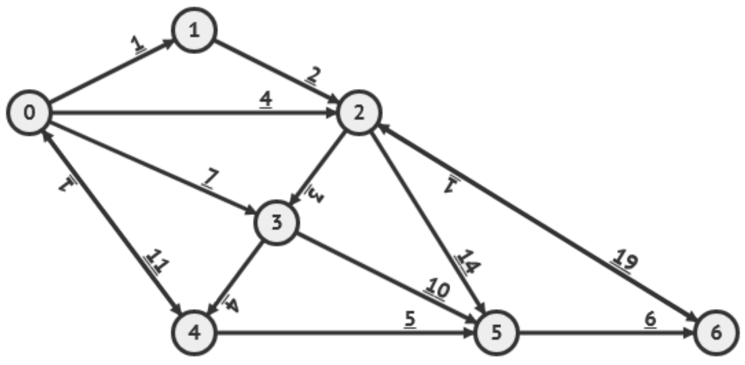

Figure 1: Directed Weighted Graph

For all questions below, please refer to the directed weighted graph shown in Figure 1.

1. Show the Single-Source Shortest Paths (SSSP) distance values `D` of Figure 1 if the source vertex is vertex $s_1 = 0$. What algorithm that you will use to compute these values and what is its time complexity in terms of V and E? Dijkstra Algorithm, with Time Complexity O((V+E)logV), can be optimised to O(logV+E) using priority queue
2. **If we assume that all edges in Figure 1 have weight 1**, show the unweighted SSSP distance values `D` if the source vertex is vertex $s_1 = 0$. What algorithm that you will use to compute these values and what is its time complexity in terms of V and E? BFS, O(V+E)
   Note that DFS is not suitable for SSSP, since it will search the far node first
3. Show the SSSP distance values `D` of Figure 1 if the source vertex is vertex $s_2 = 5$.
   Node: Shortest distance; 6:6, 2:7, 3:10, 4:14, 0:15, 1:16
4. **If we assume that all edges in Figure 1 have weight 1**, show the unweighted SSSP distance values `D` if the source vertex is vertex $s_2 = 5$.
   6:1, 2:2, 3:3, 4:4, 0:5, 1:6

C Applications (35 marks)

C.1 Efficient Additions (15 marks)

You need to add a (multi)set of integers (that may contain duplicates). However, each addition operation has a cost now. The cost is the addition of those two to be added. So, to add 2 integers $\{1, 10\}$, you need a cost of 11 (and the final sum is also 11). Now, if you want to add 3 integers: $\{1, 2, 3\}$, there are several ways:

1. Add $2 + 3 = 5$ first with cost 5, then add $5 + 1 = 6$ with another cost 6, we get the final sum 6, but with total addition cost of $5 + 6 = 11$.

2. Add $1 + 3 = 4$ first with cost 4, then add $4 + 2 = 6$ with another cost 6, we also get the final sum 6, but with smaller total addition cost of $4 + 6 = 10$.

3. Add $1 + 2 = 3$ first with cost 3, then add $3 + 3 = 6$ with cost 6, we also get the final sum 6, but with the smallest total addition cost of $3 + 6 = 9$.

The first line of input contains an integer N ($2 \leq N \leq 100\,000$).
The second line is a list L that contains N positive integers not more than 100 000.

The final sum is definitely `sum(L)` so we are not interested with that. Your job is to write a `full` Python code to print the *minimum total cost of addition* instead. Write a short comment about the overall time complexity of your code.

Your code will be graded first by correctness and if correct, by its time complexity. You will get up to 10 marks if you can only describe your solution in pseudo-code.

Here are four sample inputs with the associated sample outputs.

```
|Input 1|Output 1| |Input 2|Output 2| |Input 3  |Output 3| |Input 4  |Output 4 |
|-------|--------| |-------|--------| |---------|--------| |---------|---------|
|2      |11      | |3      |9       | |4        |19      | |5        |32       |
|1 10   |        | |1 2 3  |        | |1 2 3 4  |        | |7 1 3 1 4|         |
```

C.2 Lots of Tasks (20 marks)

Steven has n tasks to do ($1 \leq n \leq 100\,000$, yes, that's a lot, sequentially numbered from 1 to n). Unfortunately, the tasks are not independent and the execution of one task is only possible if all its dependent tasks have already been executed.

He has an initial plan to do these n tasks one after another (as he cannot split himself). Your job is to write a Python code help him check on whether his initial plan is valid (or not). If it is valid, print "`Go Ahead`" (without the quotes), otherwise print **any** valid ordering of n tasks that Steven has to follow instead.

The first line of input containing two integers n and m ($0 \leq m \leq min(100\,000, n \times (n-1)/2)$). m is the number of dependency relations between the n tasks. There is no ~~a~~cyclic dependency.

The next m lines of input contains two integers u and v ($1 \leq u, v \leq n$) that means task u must be executed before task v.

Finally, in the last line of input, Steven describes his initial plan as n integers that is a permutation of $\{1, 2, \ldots, n\}$.

Here are four sample inputs with the associated sample outputs.

```
|Input 1|Output 1| |Input 2|Output 2| |Input 3  |Output 3| |Input 4  |Output 4 |
|-------|--------| |-------|--------| |---------|--------| |---------|---------|
|3 2    |Go Ahead| |3 2    |1 2 3   | |5 0      |Go Ahead| |5 1      |5 1 2 3 4|
|1 2    |        | |1 2    |        | |5 2 1 3 4|        | |1 2      |         |
|1 3    |        | |1 3    |        | |         |        | |5 2 1 3 4|         |
|1 3 2  |        | |2 1 3  |        | |         |        | |         |         |
```

Graph Data Structure (4 marks)

This is clearly a graph problem.
How are you going to store the input graph? Describe the details of your chosen graph DS.
Is the graph in this problem special? If yes, what is its name?

Adjacency List
It is a DAG (directed acyclic graph)

Subtask 1 (3 marks)

Let's assume that the last line of input (Steven's initial plan) is **always valid**.

What is/are the significance of this constraint? (1 mark) Since the plan is always valid, we can don't use topological sort to generate a task ordering.

What is your proposed algorithm to solve this Subtask 1 (`full` Python)? (1 mark)

What is the time complexity of your Subtask 1 algorithm in terms of n and m? (1 mark)

O(m+n), where m is the number of dependencies and n is the number of tasks

Subtask 2 (6 marks)

Let's assume that the last line of input (Steven's initial plan) is **always invalid**.

What is/are the significance of this constraint? (1 mark) Now we need to validate the current dependencies and generate a new task ordering by topological sort

What is your proposed algorithm to solve this Subtask 2 (`full` Python)? (4 marks)

What is the time complexity of your Subtask 2 algorithm in terms of n and m? (1 mark)

O(m+n), for both Khan Algorithm and create adjacency list

Final Subtask 3 (7 marks)

Let's assume that the last line of input (Steven's initial plan) can be **either** valid or invalid. Obviously if it is valid, we call Subtask 1 solution, otherwise we call Subtask 2 solution.

What is your proposed algorithm to solve this final Subtask 3 (`full` Python)? (6 marks)

What is the time complexity of your final Subtask 3 algorithm in terms of N and M? (1 mark)

– End of this Paper, All the Best –

Both algorithm in subtask 1 and 2 are O(n+m), the subtask 3 is calling these, so it is also O(n+m)

```python
# C1: Efficient Additions
## Since we need to calculate the minimum cost of addition, so we need to optimise in each step
## The optimisation is to add two smallest number in each step
## The heapq (minimum heap) can do this
import heapq
def solution():
    N = int(input())
    M = list(map(int, input().split()))
    total_cost = 0
    heapq.heapify(M) # heapify the list, let the smallest items on the top
    while len(M) > 1:
        # Pop out two smallest number in first loop
        # After first loop, pop out the cost and one smallest number
        num1 = heapq.heappop(M)
        num2 = heapq.heappop(M)
        cost = num1 + num2
        total_cost += cost
        # Push back the cost
        heapq.heappush(M, cost)
    return total_cost
print(solution())
```

```python
# C2 - Subtask 1
def isValidPlan():
    n = 5 # Number of tasks
    m = 3 # Number of dependencies
    dependencies = [(1,2), (1,3)]
    plan = [1,3,2]
    #Add dictionary for pre-requests for each task
    prereq = {i: set() for i in range(1, n+1)}

    for u, v in dependencies:
        prereq[v].add(u)

    completed = set()
    for task in plan:
        if not prereq[task].issubset(completed):
            print("Invalid")
            return
        completed.add(task)
    print("Go Ahead")
    return
isValidPlan()
```

```python
# C2 - Subtask 2
from collections import deque

def find_valid_order(n, dependencies):
    adj_list = {i: [] for i in range(1, n + 1)}
    in_degree = {i: 0 for i in range(1, n + 1)}

    # 填充邻接列表和入度列表
    for u, v in dependencies:
        adj_list[u].append(v)
        in_degree[v] += 1

    # 使用 Kahn 算法进行拓扑排序
    queue = deque([v for v in in_degree if in_degree[v] == 0])
    valid_order = []

    while queue:
        vertex = queue.popleft()
        valid_order.append(vertex)

        for neighbour in adj_list[vertex]:
            in_degree[neighbour] -= 1
            if in_degree[neighbour] == 0:
                queue.append(neighbour)

    if len(valid_order) != n:
        return "不存在有效排序"
    return valid_order

# 示例使用
n = 5
dependencies = [(1, 2), (2, 3), (1, 3), (3, 4), (4, 5)]
print(find_valid_order(n, dependencies))
```

```python
# C2 - Subtask 3
def subtask3():
    if isValidPlan():
        return "Go Ahead"
    else:
        return find_valid_order()
```