NANYANG TECHNOLOGICAL UNIVERSITY

SEMESTER 1 EXAMINATION 2020-2021

EE2008 / IM1001 – DATA STRUCTURES AND ALGORITHMS

November / December 2020 Time Allowed: 2 hours

INSTRUCTIONS

1. This paper contains 4 questions and comprises 3 pages.
2. Answer all 4 questions.
3. All questions carry equal marks.
4. This is a closed book examination.
5. Unless specifically stated, all symbols have their usual meanings.

---

1. (a) The array $A[0..(n-1)]$ contains n elements. Write an algorithm that returns the index and the value of the smallest element in the array A, where the index refers to the first occurrence of the minimum in the array.

(7 Marks)

(b) Consider the following recursive algorithm, where b and n are nonnegative integers.

```
ALGORITHM mighty(b, n) {
  if (n == 0)
    return b
  else
    return b*b*mighty(b, n-1)
}
```

(i) What does this algorithm compute?

(ii) Based on the key operation of the algorithm, set up a recursive relation for the *time complexity analysis* of the algorithm. Use the approach of backward substitutions to solve for the suggested recursive relation.

(iii) Identify two drawbacks of the algorithm `mighty(b, n)` and suggest methods to improve its performance.

(18 Marks)

2. (a) Using the approach of hashing, the location of a record is determined by performing an arithmetic computation on its key. The result of the hashing process yields the location of the record in a hash table. Each slot in the table is called a bucket. Suppose the keys are 16, 34, 83, 78, 26, 90, 51, 69, 60, 56 and 35.

   (i) Draw the 13-item hash table using the hash function $h(x) = x \bmod 13$. The collisions are handled by *chaining*.

   (ii) Draw the 13-item hash table using the hash function $h(x) = x \bmod 13$. The collisions are handled by *double hashing* using a second hash function $d(x) = 7 - (x \bmod 7)$.

(18 Marks)

(b) As shown in Figure 1, the binary tree has three nodes and four null branches. Use mathematical induction to prove that a binary tree with $n \geq 1$ nodes has exactly $n + 1$ null branches. Use tree diagrams to illustrate your ideas.

(7 Marks)

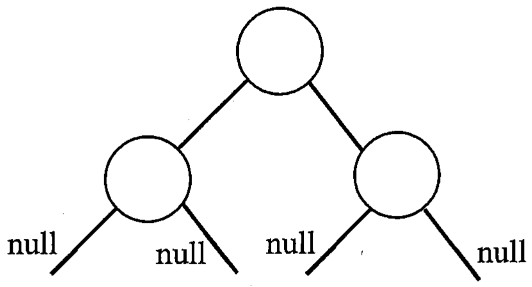


Figure 1

3. (a) Explain and show each step in using counting sort to sort the following array. Assume that the values in the array are integers in the range 1 to 5.

5	3	4	2	3	1	4	5	1	3

(10 Marks)

Note: Question No. 3 continues on page 3.

(b) (i) Given a pointer *last* to the last node in a singly linked list, write an algorithm *Attach* to attach a given node *p* to the end of the singly linked list. Return a pointer to the new last node. Note that the singly linked list may be empty.

(ii) Given two non-empty singly linked lists, suppose that each list has elements sorted in non-decreasing order. Write an algorithm to merge the two lists into a sorted singly linked list and return a pointer to its first node. Make use of the algorithm *Attach* you have written in part (i).

(15 Marks)

4. (a) Consider the weighted graph in Figure 2. Find a minimum spanning tree using Prim's algorithm starting at vertex H. Show each step clearly.

(8 Marks)

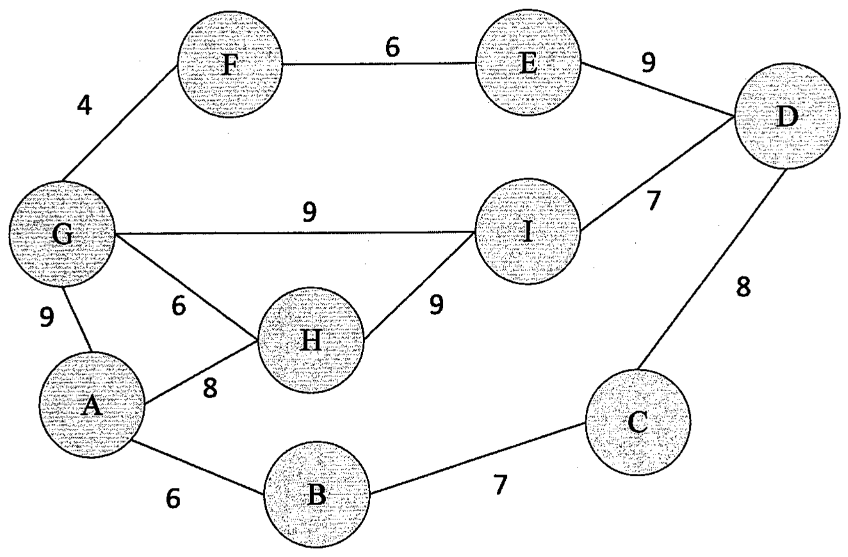


Figure 2

(b) Use Dijkstra's algorithm to find the shortest path tree for the source vertex H in the graph shown in Figure 2. Show each step clearly.

(10 Marks)

(c) Given the adjacency list of an undirected graph, write an algorithm to compute the average degree of all vertices that have even degree.

(7 Marks)

END OF PAPER

EE2008/IM1001 Data Structures and Algorithms | PYP Solution AY2020-21 Semester 1

Question 1

a) Input: array `A`
Output: smallest element `smallest`, index of the smallest element `smallest_index`

```
ALGORITHM smallest_element(A) {
  smallest = A[0]
  smallest_index = 0
  for (i = 1 to A.last) {
    if (A[i] < smallest)      // When the array element is smaller than the
      smallest = A[i]         // comparison value, store it as temporary variables
      smallest_index = i
  }
  return smallest, smallest_index
}
```

b) i) We observe that the algorithm computes the following table of results, and we extend the table for the general case $n = k$.

n	0	1	2	…	k
`mighty(b, n)`	$b = b^{2(0)} \cdot b$	$b^2 \cdot b = b^{2(1)} \cdot b$	$b^2 \cdot b^2 \cdot b = b^{2(2)} \cdot b$	…	$b^{2(k)} \cdot b = b^{2k+1}$

Hence, we can conclude that the algorithm computes and returns the value of b^{2n+1}.

ii) Let the basic operation be `*`, then $T(n) = \begin{cases} 2 + T(n-1) & \text{otherwise} \\ 0 & n = 0 \end{cases}$

$\therefore T(n) = 2 + T(n-1) = 2 + [2 + T(n-2)] = 2 \cdot 2 + T(n-2) = \cdots = 2 \cdot n + T(0) = O(n)$

iii) Drawback #1: The recursive algorithm, although easy to understand, introduces overhead costs, so in practical will be slower than a for-loop implementation. The space requirement for the recursive implementation is also larger due to the scope storage between each recursive call of the algorithm.

b^2 is calculated and stored into `b2_temp` to further reduce the time complexity to $T(n) = n + 1$.

Input: integer `b`, integer `n`
Output: integer `b`$^{2n+1}$

```
ALGORITHM mighty(b, n) {
  if (n == 0)
    return b
  b2_temp = b * b
  temp = b                    // Store b in temp as base case for n ≥ 1
  for (i = 1 to n)
    temp = temp * b2_temp
  return temp
}
```

Drawback #2: The time complexity $T(n) = 2n$ is in linear time and can be reduced to $T(n) = O(\log n)$, by reducing n by division instead of subtraction.

Note: This solution is adapted from https://cp-algorithms.com/algebra/binary-exp.html

$$b^{2n+1} = b \cdot \left[\begin{cases} 1 & \text{if } n = 0 \\ \left(b^{\frac{n}{2}}\right)^2 & \text{if } n > 0 \text{ and } n \text{ even} \\ \left(b^{\frac{n-1}{2}}\right)^2 b & \text{if } n > 0 \text{ and } n \text{ odd} \end{cases} \right]^2$$

b^{2n+1} can rewritten as shown in the left.

We can use the integer division "`/`" operator to compute $n/2$ or $(n-1)/2$.

We will implement $(b^n)^2 \cdot b$ and b^n in two separate algorithms `mighty` and `mighty_recur` respectively.

Input: integer `b`, integer `n`
Output: integer `b`$^{2n+1}$

```
ALGORITHM mighty(b, n) {
  if (n == 0)
    return b
  else
    b_n = mighty_recur(b, n)
    return b * b_n * b_n
}
```

```
ALGORITHM mighty_recur(b, n) {
  if (n == 0)
    return 1
  base = mighty_recur(b, n/2)          // Integer division is used here
  if (n%2 == 1)                        // For n odd
    return base * base * b
  else                                 // For n even
    return base * base
}
```

A time complexity analysis of `mighty` yields $T(n) = O(1)$ while `mighty_recur` runs in $T(n) = O(\log n)$ time. Hence, the two algorithms will run in $T(n) = O(\log n)$ time. The for-loop implementation of this algorithm is left as a exercise to the readers and can be referenced from the link provided earlier.

Question 2

a) i)

0	1	2	3	4	5	6	7	8	9	10	11	12
78			16	69	83			34	35			90
26				56				60				51

ii) $N = 13,\ h(x) = x \bmod 13,\ d(x) = 7 - (x \bmod 7)$ × marks a collision

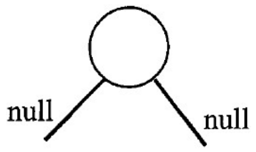
x	$h(x)$	$d(x)$	Probes: $[h(k) + jd(k)] \bmod 13$ for $j = 1, \dots, 12$
16	3	5	3
34	8	1	8
83	5	1	5
78	0	6	0
26	0	2	0× 2
90	12	1	12
51	12	5	12× 4
69	4	1	4× 5× 6
60	8	3	8× 11
56	4	7	4× 11× 5× 12× 6× 0× 7
35	9	7	9

Hence, after *double hashing* collision handling, the hash table is as below:

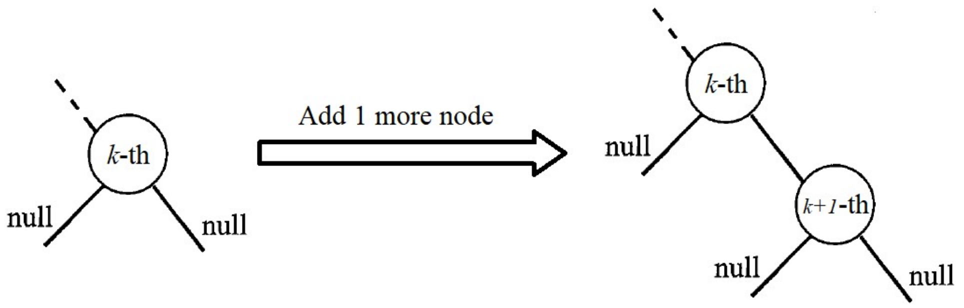
0	1	2	3	4	5	6	7	8	9	10	11	12
78		26	16	51	83	69	56	34	35		60	90

b) *Hypothesis: A binary tree with $n \geq 1$ nodes has exactly $n + 1$ null branches.*

Base case: For $n = 1$, there are 2 null branches (as seen in the diagram) which equals $1 + 1 = 2$ null branches from the hypothesis.



Assume that the $n = k$ case is true ie. k nodes has exactly $k + 1$ null branches, we will now try to prove that it is also true for the $n = k + 1$ case ie. ie. $k + 1$ nodes has exactly $k + 2$ null branches.



A new node is only allowed to be added at the location of an existing null branch. As seen in the diagram above, adding one node to the null branch removes that null branch but introduces two more new null branches from the new node, resulting in a total of $(k + 1) - 1 + 2 = k + 2$. This is consistent with the hypothesis.

Therefore, it is proven by mathematical induction that a binary tree with $n \geq 1$ nodes has exactly $n + 1$ null branches.

Question 3

a) First, initialise an array `count` which stores the frequency of each integer. Then, calculate the cumulative sum in `count`.

1	2	3	4	5
2	1	3	2	2

→

1	2	3	4	5
2	3	6	8	10

Initialise an empty array with length same as the original array. Starting from the right, reference the integer from the `count` array to find the location to insert the integer. Then, decrement the count value that you have just referenced.

					3				

1	2	3	4	5
2	3	6 → 5	8	10

Continue filling the new array with the same steps, while shifting left in the original array until the start is reached.

	1				3				

1	2	3	4	5
2 → 1	3	5	8	10

	1				3				5

1	2	3	4	5
1	3	5	8	10 → 9

	1				3		4		5

1	2	3	4	5
1	3	5	8 → 7	9

1	1				3		4		5

1	2	3	4	5
1 → 0	3	5	7	9

1	1			3	3		4		5

1	2	3	4	5
0	3	5 → 4	8	9

1	1	2		3	3		4		5

1	2	3	4	5
0	3 → 2	4	8	9

1	1	2		3	3	4	4		5

1	2	3	4	5
0	2	4	7 → 6	9

1	1	2	3	3	3	4	4		5

1	2	3	4	5
0	2	4 → 3	6	9

1	1	2	3	3	3	4	4	5	5

1	2	3	4	5
0	2	3	6	9 → 8

Hence, the last array on the left is the sorted array, sorted by counting sort.

b) i) Input: pointer to last node `last`, node to be inserted `p`
Output: pointer to new last node `last`

```
ALGORITHM Attach(last, p) {
  if (last == null)              // For empty singly linked list case
    return p
  last.next = p                  // Attach p to the last node, then return pointer to p
  return last.next
}
```

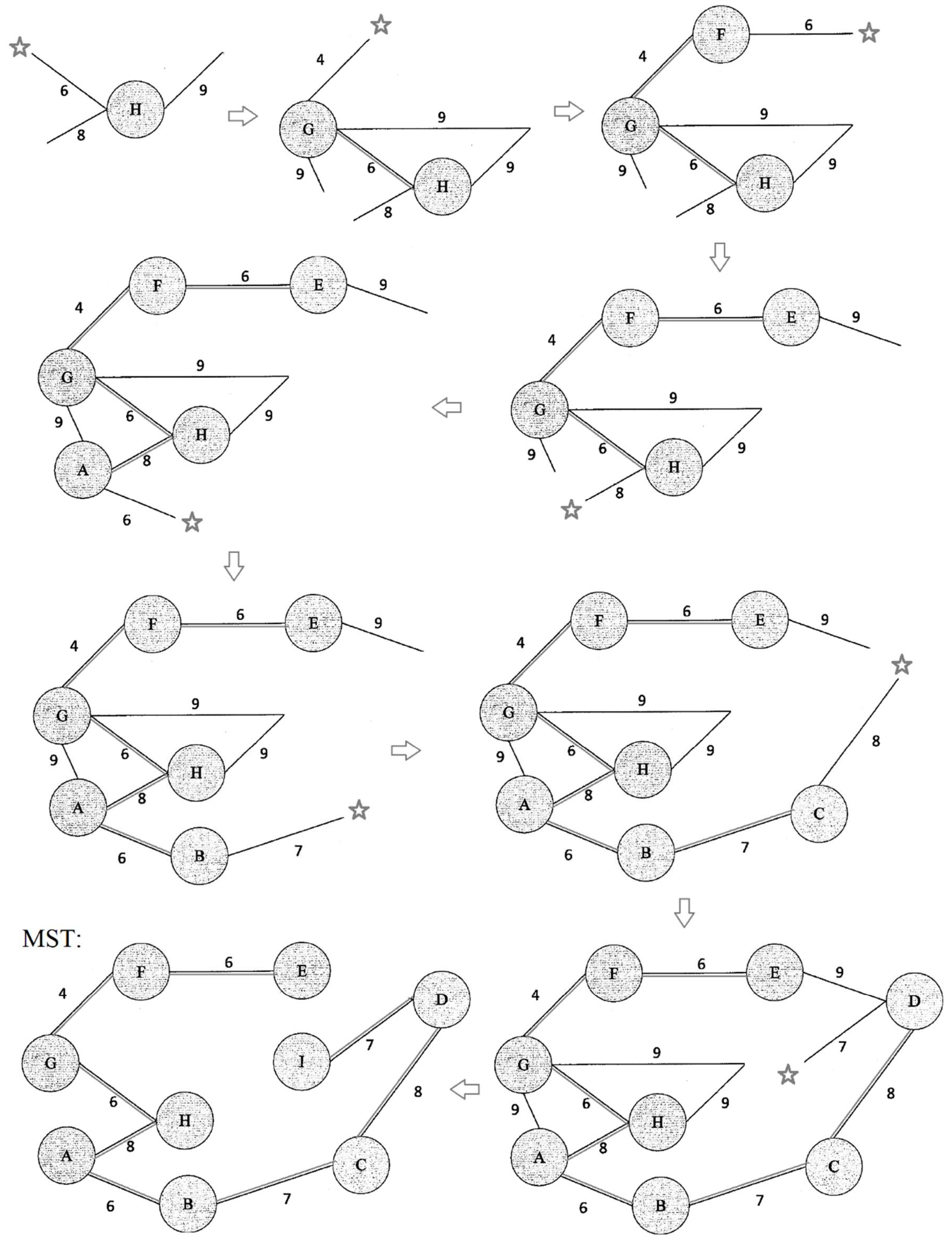
ii) This algorithm is similar to the merge algorithm in the mergesort topic. Elements of `list2` are compared individual and inserted one-by-one into `list1` if allowed. At the end, the remaining nodes of `list2` are attached to the end of `list1` (`null` is attached if nothing remaining).

Input: pointer to first node of 1st linked list `list1`, pointer to first node of 2nd linked list `list2`
Output: pointer to first node of merged sorted linked list `start`

```
ALGORITHM Merge(list1, list2) {
  if (list1.data <= list2.data)
    start = list1
  else {                                    // Attach 1st node of list2 to front of list1
    start = list2
    list2_remaining = list2.next
    list2.next = list1
    list2 = list2_remaining                 // Attach back the remaining terms
  }
  while(list1.next != null) {
    if (list1.data <= list2.data)
      list1 = list1.next
    else {
      chain_end = Attach(list1, list2)      // Insert node from list2 into list1
      chain_end.next = list1.next           // The broken linked list chain is fixed
      list1 = list1.next                    // Shift to next nodes of list 1 and 2
      list2 = list2.next
    }
  }
  list1.next = list2                        // Attach the remaining nodes of list2
  return start
}
```

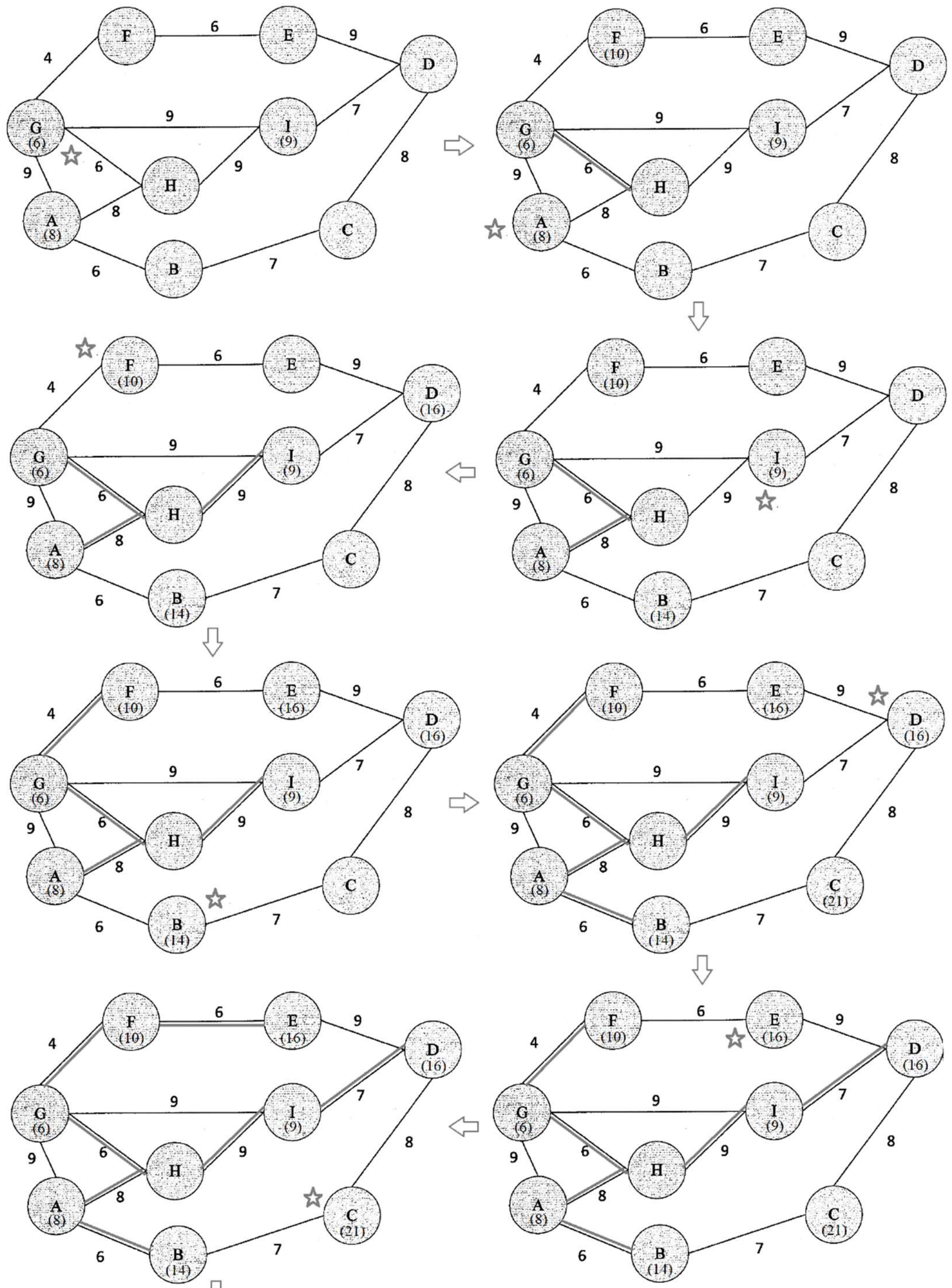
Question 4

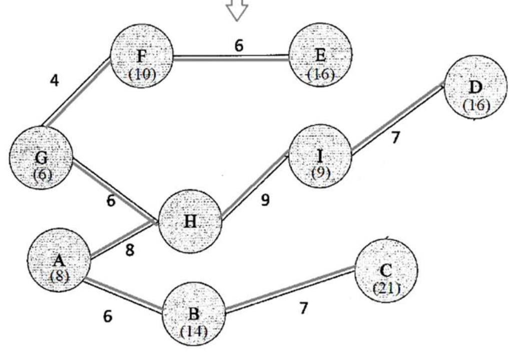
a) For Prim's algorithm, start from one vertex, then keep adding the smallest edge (from any visited vertex), until there are a total of $(n-1)$ edges.



Note: New vertices are added at ☆, of which the edge extending to ☆ is minimum.

b) For Dijkstra's Algorithm, add a minimum edge starting from the source vertex. If the number of edges of the shortest path tree (SPT) $< n - 1$, keep adding edges that have the shortest path length.





Note: Path length is written in parentheses;
New vertices are added at ☆.

c) Traverse through each linked list in the adjacency list array and count the edges for that vertex. Then check whether it is even and sum it, while counting the number of even vertices. After that, find the average by dividing this sum with the number of even vertices.

Input: adjacency list `adj`
Output: average degree of all vertices that have even degree `avg`

```
ALGORITHM avg_even_deg(adj) {
  n = adj.last
  even_deg_count = 0
  total_edges = 0

  for (i=1 to n) {
    ref = adj[i]
    edges_count = 0
    while (ref != null) {
      edges_count = edges_count + 1
      ref = ref.next
    }
    if (edges_count % 2 && edges_count != 0) {
      even_deg_count = even_deg_count + edges_count
      total_edges = total_edges + 1
    }
  }

  if (even_deg_count == 0)
    return 0
  return total_edges / even_deg_count
}
```

All the best for exams! - PL