Final Examination

Date: 16/06/2021; Duration: 120 minutes

**Online, Open-book**

|  |  |
| --- | --- |
| **SUBJECT: Algorithms & Data Structures (IT013IU)** | |
| Approval by The SCSE  Signature  A picture containing line chart  Description automatically generated  Full name: Dr. Nguyen Van Sinh | Lecturer:  Signature  Full name: Trần Thanh Tùng |
| Proctor 1  Signature  Full name: | Proctor 2  Signature  Full name: |
| **STUDENT INFO** | |
| **Student name: Võ Anh Việt**  **Student ID: ITITIU19243** | |

# INSTRUCTIONS: the total point is 100 (equivalent to 40% of the course)

1. *Purpose:*

* Test your knowledge on data structures and algorithms in the following topics: Binary Tree, Hash Table, Graphs, Advanced graph algorithms
* Examine your skill in analysis and design algorithms

1. *Requirement:*

* Read carefully each question and answer it following the requirements
* Write the answers and draw models CLEAN and TIDY directly in a **WORD** file
* You can draw your trees, graphs by hand or by any tool (like draw.io)
* Include the **SETTING** session below in your answer file.

Note: For all calculations in this subject, the following **rounding convention** is used: **7/2 = 4**

1. **Setting – TO INSERT TO YOUR ANSWER FILE**
   1. Write the last 2 digits of your student ID (called is x):\_43\_\_\_\_\_\_\_\_\_\_\_\_ (TO FILL IN)
   2. Compute your **OFFSET** = x % 5 = \_3\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (TO FILL IN)
   3. Using the table below to compute your **Starting node**: \_D\_\_\_\_\_\_\_\_\_\_\_\_ (TO FILL IN)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **OFFSET** | **0** | **1** | **2** | **3** | **4** |
| **STARTING NODE** | **A** | **B** | **C** | **D** | **E** |

**Your list iteration procedure:**

For all lists of items in the following sessions, take items from left to right starting from your **OFFSET.** If the end of the list is reached before all items are taken, continue from index 0 (wrap around).

1. **Binary search tree (25pts)**

Given a list of items, take items one by one using **your list iteration procedure**.

Table 1 - Items

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 45 | 30 | 80 | 20 | 35 | 70 | 90 | 25 | 21 | 39 | 37 | 38 | 75 | 78 | 76 | 90 |

1.a. Insert all items into a binary search tree and draw the tree (15pts)

A picture containing sketch, diagram, drawing, white

Description automatically generated  
1.b. Delete the root node and redraw the tree (10pts)

A picture containing diagram, line, white

Description automatically generated

1. **Hash table (20pts)**

Given a list of items in table 1, take items using **your list iteration procedure**.

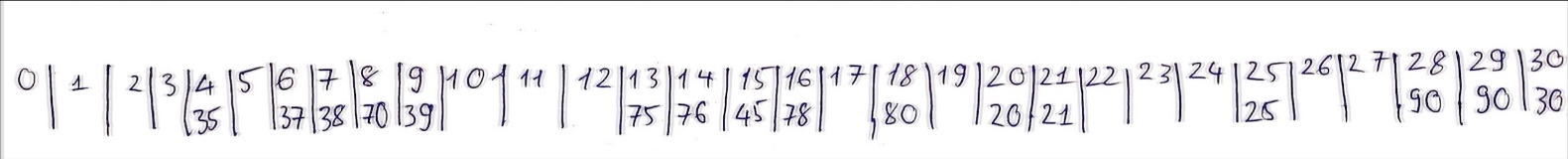
2.a. Insert all items into the hash table of size **27** by using the linear probing algorithm to solve collisions (10pts).

We use hash function: x % 27:

A picture containing text, handwriting, font, line

Description automatically generated  
2.b. Change the hash table’s size to **31**, redraw it (10pts)

We must rehashing with hash function: x % 31



1. **Graph - Elementary Algorithms (30pts)**

3.a Given the graph GS in Fig. 1, run the DFS algorithm from your **STARTING NODE** andredraw the graph with the discovery time and the finishing time for each node. (15pts)

Diagram

Description automatically generated

3.b. Find all strongly connected components in Gs and draw the Gs-1 with the finishing time for each node(15pts)

|  |
| --- |
| weighted digraph |
| *Fig. 1: Graph GS* |

**Diagram

Description automatically generated**

**Strongly Connected Component: A, DCE, I, G, F, B, H**

1. **Graph – Shortest path algorithm (15pts)**

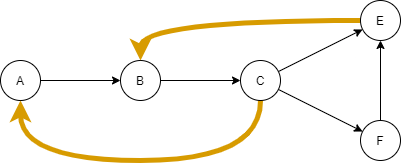
Run the Dijkstra’s algorithm on the graph Gs in Fig.1 **from your starting node**, and fill the following table with corresponding values after each step of the algorithm

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Selected nodes | A | B | C | D | E | F | G | H | I |
|  | ∞ | ∞ | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| D | ∞ | ∞ | ∞ | 0 | 4 | ∞ | ∞ | ∞ | 99 |
| D,E | ∞ | ∞ | 7 | 0 | 4 | 5 | 13 | ∞ | 18 |
| D,E,F | ∞ | 13 | 7 | 0 | 4 | 5 | 13 | 22 | 18 |
| D,E,F,C | ∞ | 9 | 7 | 0 | 4 | 5 | 13 | 22 | 18 |
| D,E,F,C,B | ∞ | 9 | 7 | 0 | 4 | 5 | 13 | 21 | 18 |
| D,E,F,C,B,G | ∞ | 9 | 7 | 0 | 4 | 5 | 13 | 20 | 18 |
| D,E,F,C,B,G,I | ∞ | 9 | 7 | 0 | 4 | 5 | 13 | 20 | 18 |
| D,E,F,C,B,G,I,H | ∞ | 9 | 7 | 0 | 4 | 5 | 13 | 20 | 18 |

1. **Algorithm to find backward edges (10pts)**

While traversing a graph using the DFS algorithm, backward edges are edges that link a node to another node in the path from the source node to the node.

For example, in the graph below, colored edges are backward edges.



* (10pts) Propose an algorithm (write a pseudo-code) based on the DFS algorithm to print out all the backward edges of a given graph starting from a source node.

DFS visit(u):

If visit[u] = true then

Return;

Visit[u[ <- true;

For (a in G.adj[u])

If visit[u] = true

Print(u+ “ ” + v);

Else

DFS visit(a)

--- The end ---