## Module 3: Implementation of Data Structures

### Module Overview

Computer programs usually operate on tables of data (information). To realize efficient algorithms and programs, it is very important to supply pre-knowledge of data types to the compiler in order to avoid storage allocation problems (i.e. problem of dynamic storage allocation).

The amount of storage allocated, say to a variable, will have to be chosen according to the size of the range of values, and their relationships, that the variables may assume. Therefore, to use a computer properly, we need to understand the structural relationships present within data as well as the basic techniques for representing and manipulating such structure within a computer.

The data elements to be manipulated by a program are required to be associated with a type, which in turn is required to be made explicit by a declaration from the very beginning. From initial programming experience, one can recall that the standard primitive data types are; integer, real, Boolean, character, and set. The data type characterizes the set of values to which elements, such as constants, variables, expressions, or functions, belong to.

In general, the design of computer representations depends on the desired functions of the data as well as on its intrinsic properties (relationships). This module discusses the basic sets by which data stored in the computer memory can be manipulated. The sets are held in data structures depending on their intrinsic operability. They can change over time by growing or shrinking. Therefore, this module shows the several types of operations that can be performed on sets, such as insertion, deletion, or test of membership, and illustrates them as they apply to the basic data structures of stacks, queues, linked lists, and trees.

In this module, we shall be discussing the following study units:

Study Unit 1: Stacks and Queues

Study Unit 2: Linked Lists and Trees

**Study Unit 1: Stack and Queue Data Structure**

Expected Duration: # week

**Introduction**

The elementary data structures, which form many complex data structures, are identified with respect to the principal operations to be performed, just as it is notable that computer memories are distinguished by their intended applications.

Here, the elementary data structures of interest are those of stacks, queues, linked lists, and rooted trees. We, therefore, consider some of the basic operations involving these data structures.

**Learning Outcomes for Study Unit 1**

At the end of this study unit, you should be able to;

1.1 Define and use correctly the words in bold (SAQ 1.1).

1.2 List and explain the operations on stack (SAQ 1.2).

1.3 List and explain the operations on queue (SAQ1.3).

**1.1 What does Stack Data Structure mean?**

- The most recent item (youngest) currently in the list is normally the first to be removed. This mode of operation is popularly tagged (LIFO).
- The implementation of stacks is useful in diverse areas of computing: in the process of entering and leaving subroutines; for the processing of languages with a nested structure, like programming languages, arithmetic expressions, etc. In general, stacks occur most frequently in connection with explicitly or implicitly recursive algorithms.
- A Stack is typically identified with a top (accessible item) and a bottom (least accessible item). The INSERT operation on a stack is usually referred to as PUSH, while the delete operation, which does not take an element argument, is regarded as POP. These terminologies (PUSH and POP) come from an analogy with the stacks of plates often found in cafeterias. They don't portend a physical motion in computer memory, as in principle, items are only added onto the top.

- Consider a stack with at most n – elements, i.e. an array S[1…n]. The array has an attribute S.top that indexes the most recently inserted element. Thus, the stack consist of elements S [1,., S.top], where S[1] is the bottom element and S[S.top ] is the top element.

- When S.top = 0 – The stack is empty (comment: imagine a spring-based stack , where the top, a pointer, is always occupied as long as there is an item in the stack: Thus, empty stack implies nothing on top). An empty stack is tested with a query operation - STACK-EMPTY.

- An attempt to pop an empty stack, underflows, is an error, as well as attempt to exceed the top of the stack, overflows.

- We can, therefore, proceed to implement the stated operations on stack with the following pseudocode:

### 1.2 Operations on Stack Data Structure

Considering an array S [1…n ].

**STACK – EMPTY (S)**

(i) if S.top = = 0
(ii) return TRUE
(iii) else return FALSE

**PUSH (S,x)**

1. S.top = S.top + 1
2. If S.top > S[n], then OVERFLOW
3. else S [S.top] = x

**POP (S)**

1. if S.top = = 0, then UNDERFLOW;
2. else Y = S [S.top]
3. S.top = S.top - 1
4. return Y

// Alternative Pseudocode

**POP (S)**

1. IF STACK – EMPTY(S)
2. error " underflow''
3. else S.top = S.top - 1
4. return S [S.top + 1]

**In-text Question:** What is another name for pop operation in stack?

**Solution:** Delete

## 1.3 Queue data Structure and associated Operations

### 1.3.1 Meaning of Queue Data Structure

- A queue is another dynamic set, a linear data structure, for which all insertions are made at one end of the list, and all deletions (and usually all accesses) are made at the other end. In a queue, the element deleted is always the one that has been in the set for the longest time (FIFO).
- Note that there is another variant of the linear data structure called Deque (double–ended–queue), pronounced as deck and has some properties in common with a deck of cards. A deque is a linear list (data structure) for which all insertions and deletions (and usually all accesses) are made at the ends of the list. We shall ignore the details and operations on deque in this course, in order to keep things simple.
- Thus, continuing with queues; the Insert operations on a queue is usually regarded as ENQUEUE, while the delete operation is DEQUEUE (don't confuse this with deque)
- The FIFO property of a queue causes it to operate like a line of customers waiting to pay a cashier. The queue has a head and a tail attribute associated with it. A newly arriving element is at the tail, while the element dequeued is always the head.

### 1.3.2 Operations on Queues

The following pseudocodes can be used to implement the basic operations of a queue.
Considering a queue Q, an array Q [1,…, n] where n is the length of queue.

(1) ENQUEUE (Q, x)

1. If Q. tail = = Q. length //checking of Q is (empty/full?)
2. Q. tail = 1 //better to start at position 1
3. else Q.tail = Q.tail + 1
4. If Q.tail > Q.length, the OVERFLOW
5. Q [Q.tail ] = x

Compare this with the alternative pseudocode.

**In-text Question:** What does the program statement "Q.tail + 1" do in the code?

**Solution:** The statement takes the processor to the next item on the queue

ENQUEUE = (Q, x)

1. Q [Q.tail ] = x
2. If Q.tail = = Q. length
3. Q.tail = 1
4. else Q.tail = Q.tail + 1

(2) DEQUEUE ( Q )

1. If Q.head = Q.tail, then UNDERFLOW;
2. If Q.head = = Q.length
3. Q.head = 1
4. else Q.head = Q.head + 1
5. x = Q [Q.head ]

**NB**

a. The elements in the queue reside in locations Q.head, Q.head + 1 ,…, Q.tail – 1, where we "wrap around" in the sense that location I immediately follows location n in circular order.

b. When Q.head = Q.tail, the queue is empty.

c. Initially, we have Q.head = Q.tail = 1

d. An attempt to dequeue an empty queue causes underflows.

e. When Q.head = Q.tail + 1, the queue is full; an attempt to enqueue a full queue causes overflows.

## 1.4 Summary of Study Unit 1

In this unit, you have learned the following;

1. Stacks and queues are linear data structures.
2. Insertions and deletions on a stack are made at one end of the array or list.
3. The most recent item currently in the list is normally the first to be removed from a stack (LIFO) while in queues, the first element is usually the first to come out (FIFO)
4. The INSERT operation on a stack is usually referred to as PUSH, while the delete operation is regarded as POP.
5. Some operations on stack include POP, PUSH, and ISEMPTY
6. In queues, the Insert operations is usually regarded as ENQUEUE, while the delete operation is DEQUEUE.
7. The operations on a queue include ENQUEUE and DEQUEUE

## 1.5 Self-Assessment Questions (SAQs) for Study Unit 1

Now that you have completed this study session, you can check to see how well you have achieved its Learning Outcomes by answering the following questions.

**SAQ 1.1 (Tests learning outcome 1.1)**

1. A linear data structure which can be implemented as array in which all insertions and deletions are made at one end of the array is called?

**SAQ 1.2 (Tests learning outcome 1.2)**

1. What is the difference between PUSH and POP operations in stack?
2. Which of the following is associated with stack? a. LIFO b. FIFO

**SAQ 1.3 (Tests learning outcome 1.3)**

1. Can Queues be regarded as linear data structures? (YES/NO)
2. The two basic operations on queues are…… and …….

**References for further reading**

Eiselt, Horst A et al. Integer Programming and Network Models. Springer, 2011.

*Introduction to Algorithms* (Cormen, Leiserson, Rivest, and Stein) 2001, Chapter 16 "Greedy Algorithms".

Eiselt, Horst A et al. Integer Programming and Network Models. Springer, 2011.

J.H. Holland (1975) *Adaptation in Natural and Artificial Systems,* University of Michigan Press, Ann Arbor, Michigan; re-issued by MIT Press (1992).

Optimal design of heat exchanger networks, Editor(s): Wilfried Roetzel, Xing Luo, Dezhen Chen, Design and Operation of Heat Exchangers and their Networks, Academic Press, 2020, Pages 231-317, ISBN 9780128178942, https://doi.org/10.1016/B978-0-12-817894-2.00006-6.

Wang FS., Chen LH. (2013) Genetic Algorithms. In: Dubitzky W., Wolkenhauer O., Cho KH., Yokota H. (eds) Encyclopedia of Systems Biology. Springer, New York, NY. https://doi.org/10.1007/978-1-4419-9863-7_412

Fred Glover (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence". Computers and Operations Research. **13** (5): 533–549,https://doi.org/10.1016/0305-0548(86)90048-1

Optimization of Preventive Maintenance Program for Imaging Equipment in Hospitals, Editor(s): Zdravko Kravanja, Miloš Bogataj, Computer-Aided Chemical Engineering, Elsevier, Volume 38, 2016, Pages 1833-1838, ISSN 1570-7946, ISBN 9780444634283, https://doi.org/10.1016/B978-0-444-63428-3.50310-6.

Glover, Fred, and Gary A Kochenberger. Handbook Of Metaheuristics. Kluwer Academic Publishers, 2003.

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671-680. Retrieved November 25, 2020, from http://www.jstor.org/stable/1690046

Brief review of static optimization methods, Editor(s): Stanisław Sieniutycz, Jacek Jeżowski, Energy Optimization in Process Systems and Fuel Cells (Third Edition), Elsevier, 2018, Pages 1-41, ISBN 9780081025574, https://doi.org/10.1016/B978-0-08-102557-4.00001-3.

NIU, M., WAN, C. & Xu, Z. A review on applications of heuristic optimization algorithms for optimal power flow in modern power systems. J. Mod. Power Syst. Clean Energy 2, 289–297 (2014), https://doi.org/10.1007/s40565-014-0089-4

J. L. Wang, Y. H. Lin and M. D. Lin, "Application of heuristic algorithms on groundwater pumping source identification problems," 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Singapore, 2015, pp. 858-862, https://doi.org/10.1109/IEEM.2015.7385770.

Matott, L. Shawn, et al. "Application of Heuristic Optimization Techniques and Algorithm Tuning to Multilayered Sorptive Barrier Design." Environmental Science & Technology, vol. 40, no. 20, 2006, pp. 6354–6360., https://doi.org/10.1021/es052560+.

Larranaga P, Calvo B, Santana R, Bielza C, Galdiano J, Inza I, Lozano JA, Armananzas R, Santafe G, Perez A, Robles V (2006) Machine learning in bioinformatics. Brief Bioinform 7(1):86–112

## Study Unit 2: Linked List and Tree Data Structures

Expected Duration: # week

### Introduction

In this module, we shall be discussing another interesting linear data structure called linked lists. The implementation of linked list is different from stacks and queues. We shall also be looking at a more advanced data structure known as Tree data structure.

### Learning Outcomes for Study Unit 2

At the end of this study unit, you should be able to;

2.1 Define and use correctly the words in bold (SAQ 2.1)

2.2 List and explain the operations on Linked lists (SAQ 2.2).

2.3 List and explain the types and operations on Tree data structure (SAQ 2.3).

2.4 List the operations on Tree data structure (SAQ 2.4)

2.5 List the application areas of Tree data Structure (SAQ 2.5).

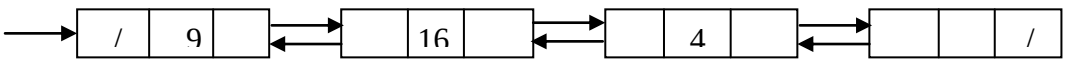
.

### 2.1 The Meaning of Linked Lists

- A linked list is a data structure in which the objects are arranged in a linear order. Unlike an array, however, in which the linear order is determined by the array indices, the order in a linked list is determined by a pointer in each object.

- Linked lists provide simple and flexible representation for dynamic data sets.

### Illustration

(a) L. head 

- As shown in the above figure, each element in a doubly linked list (as the illustration portrays) is an object with an attribute key and two pointer attributes, next and previous.

- Given an element x in the list, where x. previous = NIL implies no predecessor and thus the first element or the head of the list.

- IF x . next = NIL – no successor or tail of the list.

- A list may be singly linked or doubly linked, it may be sorted or not and circular or not. In singly linked list, previous pointers are absent. In sorted linked list, the linear order of the lists corresponds to the linear order of the keys. In circular linked list, the previous pointer of the head points to the tail while the next pointer of the tail points to the head.

## 2.2 Operations on Linked Lists

The following pseudocodes may be used to implement search, insertion, and deletion operation on a linked list.

(1) LIST – SEARCH (L,k)

```
1    x = L.head
2.   while x ≠ NIL and x.key ≠ k
3        x  =  x . next
4.   return x.
```

(2) LIST – INSERT (L, x)

```
1.   x.next  =  L.head
2.   If L.head ≠ NIL
3.   L.head. prev  = x
4.   L.head =  x
5.   x . prev = NIL
```

**Comment:**

Given an element x whose key attribute has already been set, the LIST-INSERT procedure joins (splices) x unto the front of the linked list. The attribute notation can cascade, so that

L.head previous denotes the previous attribute of the object that L.head points to. The running time of LIST-INSERT on a list of n-elements is **O**(1) (big Oh of 1).

(3) LIST-DELETE (L, x)

1. If x . prev ≠ NIL
2. x.prev.next = x.next
3. else L.head = x.next
4. If x.next ≠ NIL
5. x.next.prev = x.prev.

- The above code reassigns the next pointer of x to the previous adjacent element and the previous pointer to the next adjacent (i.e. succeeding) element. Thereby removes all pointers to x that was intended to be deleted.
  LIST – DELETE runs in **O**(1), but to delete an element with a given key, the worst case requires Θ(n) time.

**In-text Question:** What determines the order in a linked list Pointer or array indices?
**Solution:** Pointer

## 2.3 Tree Data Structure

A **tree data structure** stores and organises data in a hierarchical structure. The **tree data structure** starts with a root node and has descendant branches. It has nodes connected by edges, where each node has a value, and some nodes have child nodes while others don't. This type of data structure resembles a tree, hence its name.
A **tree data structure** helps organise a data hierarchy and is a non-linear data structure. Moreover, it helps in accessing data faster because it is non-linear.

## 2.4 Types of Tree Data Structure

The various types of the tree data structure are:

**a. Binary Tree**

The **binary tree in data structure** is one where every element or parent node has up to two children. Hence, each node could have 1, 2, or 0 children. The two children are called the left child and the right child. The properties of a binary tree are:

- It is either an empty tree or consists of nodes - root node, right subtree, and left subtree.
- The topmost node in **the binary tree data structure** is called the root.
- The nodes that don't have children are called terminal or leaf nodes.
- The tree's height is measured as the longest distance between the root node and a leaf node.
- The depth of the node is equal to the length of the path to its root.

**b. Binary Search Tree**

Every node in the binary search tree has a key and associated value. This provides quick addition, lookup, and removal. It also has a maximum of two nodes, like binary trees. However, every binary search tree is a binary tree, but not vice versa.

The difference between the two is that in a binary search tree, the left child node's value is less than the parent's, while the right side node's value is higher. The properties of a binary search tree are:

- The value of the root node must be higher than the value in all the nodes of the left subtree.
- The value of the root node must be higher than all the values in nodes of a right subtree.

**c. AVL Tree**

In a **tree data structure,** the AVL (Adelson, Velskii, and Landis who are the inventors) tree is a binary tree that is self-balanced after checking the balance factor of every mode. Hence, the right and left subtree heights must be balanced. The balance factor could be either 0.1 or -1. The biggest difference between the left subtree and the right subtree should be 1. If the difference between the subtrees is more than 1, the trees must be re-balanced using the rotation techniques. The properties of an AVL tree are:

i. The difference between the height of any two child subtrees should be at most one.
ii. The balance factor is the difference between the left subtree's height and the right subtree's height.
iii. The -1 balance factor states that the right subtree is one level higher than the left subtree.
iv. The 0 balance factor states that the left and right subtree heights are equal.

v. 1 balance factor means the left subtree is one level higher than the right subtree.

**d. B-Tree**

B-tree in **data structure** is a self-balancing tree where each node has more than one key and more than two children. It can store multiple keys in a single node and have many child nodes. This leads to a reduction in the height of the tree and enables fast desk access. The properties of a B-tree include:

i. There are at most m children in every node.

ii. Every node, except the root node and leaf node, has at least m/2 children.

iii. All leaf nodes are at the same level.

iv. All root nodes have a minimum of 2 nodes

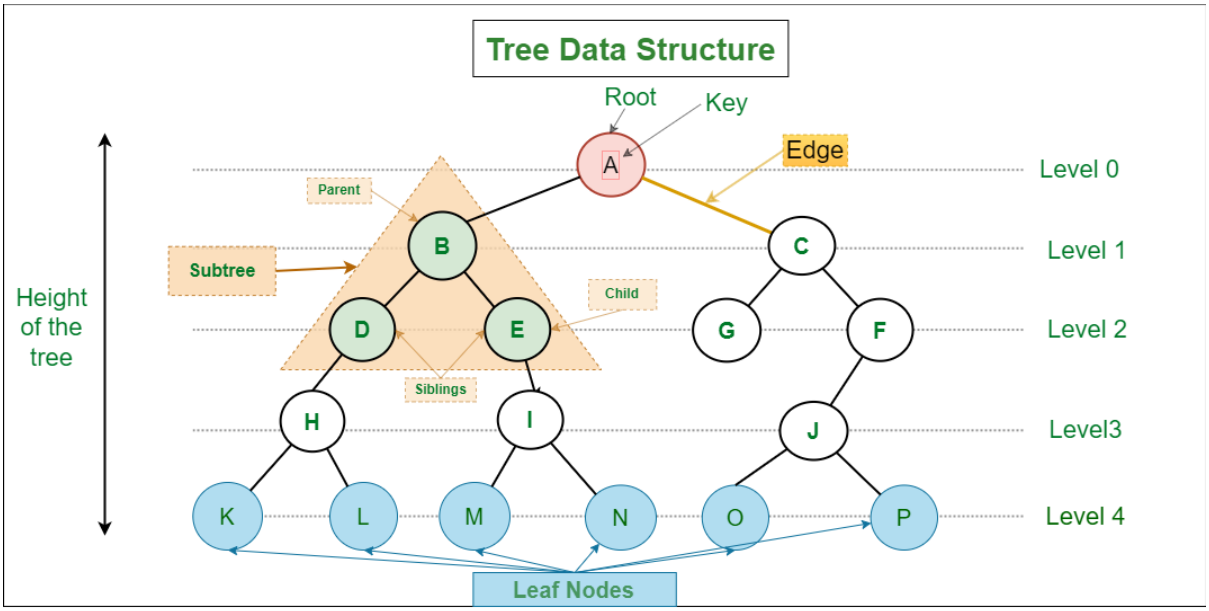



**Illustration of Tree Data Structure**

### 2.4.1 Operations on Tree Data Structure

The basic operations that can be performed in a **tree data structure** include:

- **Insertion:** There are multiple ways to insert an element depending on the location, like inserting the element in the rightmost or the leftmost vacant position or inserting the element in the first empty position available.

- **Searching:** It is a simple process to search in a binary tree to check if the current node value matches the required value. The process can be repeated to the right and left subtrees with a recursive algorithm until a match is found.

- **Deletion:** Deleting is a tricky process in a **tree data structure.** When we delete a node, complications may occur to the right and left children. A deleted node is a leaf node that occurs. Thus, the deletion process of a node includes:

- Checking if the root is NULL.

- Searching for an item in the left and right subtree and recursing it.

- Deleting the root node of a tree.

**In-text Question:** What is the difference between a Binary Tree and a Binary Search Tree?
**Solution:** The difference between the two is that in a binary search tree, the left child node's value is less than the parent's, while the right side node's value is higher.

## 2.5 Applications of a Tree Data Structure

The **tree data structure** stores the data in a hierarchical manner. The nodes are arranged at multiple levels. Thus, the application of a tree data structure is:

i. Information in the computer is stored hierarchically. There are multiple folders in a drive, and each folder has many sub-folders. The files include images, documents etc.
ii. It is easy to search for a node in **a tree data structure.** Operations like insertion and deletion are easy to perform.
iii. The **tree data structure** is used to store data in the routing tables.
iv. Programmers also use syntax trees to verify the syntax of the programs they write.

Data structures are the building blocks of a programming language. They are used for simple and complex computations in various areas of computer science. Having strong knowledge of **tree data structures** is important as they are the foundation of writing code. Hence, this knowledge can help one secure well-paying jobs.

## 2.6 Summary of Study Unit 2

In this unit, you have learned the following;

1. Linked list is a data structure in which the objects are arranged in a linear order
2. A list may be singly linked or doubly linked, it may be sorted or not and circular or not.
3. The operations on a linked list include; search, insertion, and deletion operations.
4. A tree data structure stores and organises data in a hierarchical structure
5. A tree data structure helps organise a data hierarchy and is a non-linear data structure
6. The types of Tree data structure include; Binary Trees, Binary Search Trees, AVL and B-Tress.
7. Some of the operations on Tree data structure include; Serach, Delete and Insert operations.
8. Programmers also use syntax trees to verify the syntax of the programs they write.

## 2.7 Self-Assessment Questions (SAQs) for Study Unit 2

Now that you have completed this study session, you can check to see how well you have achieved its Learning Outcomes by answering the following questions.

**SAQ 2.1 (Tests learning outcome 2.1)**

1. What is a linked list data structure?

**SAQ 2.2 (Tests learning outcome 2.2)**

1. Given the following program statement, IF x . next = NIL, explain the implication of x being equal to NIL.
2. The operations on a linked list exclude the following? a. Insertion b. Deletion c. Concatenation d. Search

**SAQ 2.3 (Tests learning outcome 2.3)**

1. Can Binary Tree search be regarded as a type of Tree data structure?

**SAQ 2.4 (Tests learning outcome 2.4)**

1. Can the expression IFISNULL be said to be part of operations on trees?

**SAQ 2.5 (Tests learning outcome 2.5)**

1. One known application of Trees to programming is ?

**References for further reading**

Eiselt, Horst A et al. Integer Programming and Network Models. Springer, 2011.

*Introduction to Algorithms* (Cormen, Leiserson, Rivest, and Stein) 2001, Chapter 16 "Greedy Algorithms".

Eiselt, Horst A et al. Integer Programming and Network Models. Springer, 2011.

J.H. Holland (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan; re-issued by MIT Press (1992).

Optimal design of heat exchanger networks, Editor(s): Wilfried Roetzel, Xing Luo, Dezhen Chen, Design and Operation of Heat Exchangers and their Networks, Academic Press, 2020, Pages 231-317, ISBN 9780128178942, https://doi.org/10.1016/B978-0-12-817894-2.00006-6.

Wang FS., Chen LH. (2013) Genetic Algorithms. In: Dubitzky W., Wolkenhauer O., Cho KH., Yokota H. (eds) Encyclopedia of Systems Biology. Springer, New York, NY. https://doi.org/10.1007/978-1-4419-9863-7_412

Fred Glover (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence". Computers and Operations Research. **13** (5): 533–549,https://doi.org/10.1016/0305-0548(86)90048-1

Optimization of Preventive Maintenance Program for Imaging Equipment in Hospitals, Editor(s): Zdravko Kravanja, Miloš Bogataj, Computer-Aided Chemical Engineering, Elsevier, Volume 38, 2016, Pages 1833-1838, ISSN 1570-7946, ISBN 9780444634283, https://doi.org/10.1016/B978-0-444-63428-3.50310-6.

Glover, Fred, and Gary A Kochenberger. Handbook Of Metaheuristics. Kluwer Academic Publishers, 2003.

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by Simulated Annealing. *Science, 220*(4598), 671-680. Retrieved November 25, 2020, from http://www.jstor.org/stable/1690046

Brief review of static optimization methods, Editor(s): Stanisław Sieniutycz, Jacek Jeżowski, Energy Optimization in Process Systems and Fuel Cells (Third Edition), Elsevier, 2018, Pages 1-41, ISBN 9780081025574, https://doi.org/10.1016/B978-0-08-102557-4.00001-3.

NIU, M., WAN, C. & Xu, Z. A review on applications of heuristic optimization algorithms for optimal power flow in modern power systems. J. Mod. Power Syst. Clean Energy 2, 289–297 (2014), https://doi.org/10.1007/s40565-014-0089-4

J. L. Wang, Y. H. Lin and M. D. Lin, "Application of heuristic algorithms on groundwater pumping source identification problems," 2015 IEEE International Conference on Industrial

Engineering and Engineering Management (IEEM), Singapore, 2015, pp. 858-862, https://doi.org/10.1109/IEEM.2015.7385770.

Matott, L. Shawn, et al. "Application of Heuristic Optimization Techniques and Algorithm Tuning to Multilayered Sorptive Barrier Design." Environmental Science & Technology, vol. 40, no. 20, 2006, pp. 6354–6360., https://doi.org/10.1021/es052560+.

Larranaga P, Calvo B, Santana R, Bielza C, Galdiano J, Inza I, Lozano JA, Armananzas R, Santafe G, Perez A, Robles V (2006) Machine learning in bioinformatics. Brief Bioinform 7(1):86–112

**Notes to the Self- Assessment Questions (SAQs) for Study Unit 1**

**SAQ 1.1 (Tests learning outcome 1.1)**

1. A Stack

**SAQ 1.2 (Tests learning outcome 1.2)**

1. PUSH operations represent insertion while POP operations represents deletion
2. LIFO

**SAQ 1.3 (Tests learning outcome 1.3)**

1. YES
2. The two basic operations on queues are ENQUEUE and DEQUEUE

**Notes to the Self- Assessment Questions (SAQs) for Study Unit 2**

**SAQ 2.1 (Tests learning outcome 2.1)**

1. Linked list is a data structure in which the objects are arranged in a linear order

**SAQ 2.2 (Tests learning outcome 2.2)**

1. It means that the list does not have a successor or tail.
2. C - Concatenation

**SAQ 2.3 (Tests learning outcome 2.3)**

1. No but Binary Search Tree is a type of Tree data structure

**SAQ 2.4 (Tests learning outcome 2.4)**

1. Yes IFISNULL is a type of operation on Trees.

**SAQ 2.5 (Tests learning outcome 2.5)**

1. Trees can be used by to verify the syntax of programs before execution