Mikhael Abie Saputra – 5025221113

Tugas 6 Struktur Data

1. What is TREE data structure and how can it be used in a computer system?
   Jawab :
   TREE adalah data struktur non-linear yang dimana elemen - elemennya memiliki hubungan seperti *parent & child*. TREE juga merupakan struktur hierarkial yang digunakan untuk memudahkan navigasi dan pencarian data.

2. What are the different types of tree traversal?
   Jawab :
   - INORDER Traversal
   - PREORDER Traversal
   - POSTORDER Traversal
   - Level order traversal

3. Provide nonrecursive algorithms for TREE traversal.
   Jawab :
   Iterative Preorder Traversal
   Algoritma ini menelusuri *tree* dalam urutan *preorder (Root, Left, Right)* tanpa menggunakan rekursi. Kita akan menggunakan stack untuk tetap mengawasi *nodes*. Kita memulai dengan mendorong *root node* kedalam *stack*. Kemudian, selama *stack* tidak kosong, Kita akan melakukan *pop* kepada stack, memprosesnya, dan mendorong kanannya dan kiri *children* kedalam *stack*.

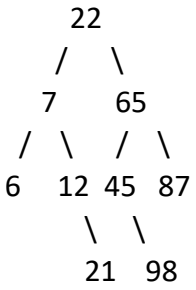
   Iterative Inorder Traversal:
   Algoritma ini menelusuri *tree* dalam urutan *inorder (Left, Root, Right)* tanpa menggunakan rekursi. Kita akan menggunakan *stack* untuk tetap mengawasi *nodes*. Kita akan memulai dengan mendorong *root node* kedalam *stack* untuk tetap mengawasi *nodes*. Kemudian, selama *stack* tidak kosong, kita pergi kekiri sampai kita mencapai *leaf node* dan dorong setiap *node* yang kita jumpai kedalam *stack*. Ketika kita menjumpai *leaf node*, kita *pop node* kedalam *stack*, memproses, dan mendorong *child* kanan kedalam *stack*. Kita akan mengulang *stack* sampai kosong.

   Iterative Postorder Traversal:
   Algoritma ini menelusuri *tree* dalam urutan *postorder (Left, Right, Root)* tanpa menggunakan rekursi. Kita menggunakan 2 *stack* untuk mengawasi *nodes*. Kita memulai dengan mendorong *root node* kedalam *stack* pertama. Kemudian, Selama *stack* pertama tidak kosong, kita *pop* sebuah *node* dari *stack* pertama dan mendorongnya kepada *stack* kedua. Kemudian, kita mendorong kiri dan kanan *children* kedalam *stack* pertama. Kita melakukan ini sampai *stack* pertama kosong.

Setelah *stack* pertama kosong, kita akan *pop* setiap *nodes* dari *stack* kedua dan memprosesnya dalam urutan mereka di*push* kedalam *stack*.

4. What is an expression and how can it be formed? Explain with a suitable example.
   Jawab :
   Dalam pemrograman komputer, sebuah ekspresi adalah sebuah kmobinasi dari nilai, variabel, operator, and fungsi-fungsi yang bisa di evaluasi untuk mendapatkan sebuah hasil. Expressions can be used to represent mathematical or logical computations and are often used in conditional statements, loops, and function calls.

5. What is Height balanced tree? Construct by using 12, 45, 65, 7, 87, 98, 6, 54, 22, 21.
   Jawab :



6. How to find the Lowest Common Ancestor of two nodes on a Binary Tree?
   Jawab :
   Untuk mencari *lowest common ancestor* dari *node* pada *binary tree*, kita dapat menggunakan algoritma seperti berikut:

   1. Mulai dari *root* dari *binary tree*.
   2. Jika ada dari antara 2 *node* adalah rootnya, return *root* sebagai *lowest common ancestor*.
   3. Secara rekursif akan mencari 2 *node* disebelah kiri dan kanan *subtrees*.
   4. Kalau kedua *node* ditemukan pada sebelah kiri *subtree*, secara rekursif mencari untuk *lowest common ancestor* pada sebelah kiri *subtree*.
   5. Jika kedua *node* ditemukan dalam kanan *subtree*, secar rekursif mencari *lowest common ancestor* dalam *subtree* kanan.
   6. Jika salah satu *node* ditemukan pada *subtree* kiri dan yan lainnya ditemukan pada *subtree* kanan, kemudian *node* yang sekarang di return sebagai *lowest common ancestor*.

7. What is the difference between B Tree and B+ Tree?
   Jawab :
   B-Tree dikenal dengan *self-balancing tree* dan *nodes*-nya disorting dengan *inorder traversal*. Dalam *B-tree*, sebuah *node* dapat memiliki lebih dari 2 *children*. *B-Tree* memiliki tinggi logM N (Dimana 'M' adalah urutan *tree* dan N adalah jumlah *nodes*). Dan tingginya disesuaikan secafra otomatis setiap *update*. Dalam *B-tree* data di

sorting dalam urutan spesifik, dengan *value* paling kecil di kiri dan *value* tertinggi di kanan.

B+ tree mengeliminasi kelemahan dari *B-tree* dengan menggunakan *indexing* dengan menyetorkan *data pointer* hanya pada *leaf nodes* dari *tree*. Dengan demikian, struktur dari *leaf nodes* dari *B+ tree* cukup berbeda dari struktur *internal nodes* pada *B tree*.

8. What is a Heap Tree? What is its use?

   Jawab :

   Adalah sebuah *complete binary tree* dan diimplementasi dalam sebuah *sequential array.*

   Sebuah *heap* dapat dibuat dengan 2 cara yang berbeda seperti *MAX – HEAP* atau *MIN – HEAP.*

9. What is a 2-3 TREE?

   Jawab :

   TREE yang memiliki properti 2-3 TREE :

   1. *Nodes* dengan 2 *children* disebut *2-nodes*. Setiap *2-nodes* memiliki satu *data value* dan dua *children*
   2. *Nodes* dengan 3 *children* disebut *3-nodes*. Setiap *3-nodes* memiliki dua *data values* dan tiga *children*.
   3. Data disetorkan dalam aturan yang sudah di *sort*.
   4. Adalah sebuah *balanced tree*.
   5. Semua *leaf nodes* ada pada level yang sama.
   6. Setiap *node* bisa menjadi antara *leaf*, *2 node*, or *3 node*.
   7. Pemasukan data selalu dimasukan pada *leaf*.

10. What is RED-BLACK TREE?

    Jawab :

    RED-BLACK TREE adalah *binary search tree* yang memiliki properti *red-black*:

    1. Semua *node* memiliki antara *red* atau *black.*
    2. Semua *leaf (NULL)* adalah black.
    3. Kalau sebuah *node* adalah *red*, maka kedua *children* adalah *black*.
    4. Semua *path* simpel dari *node* ke *descendant leaf* mengandung jumlah *black nodes* yang sama.