DATA STRUCTURES.2

2022-2023

PROJECT

Ali Mert YILMAZ

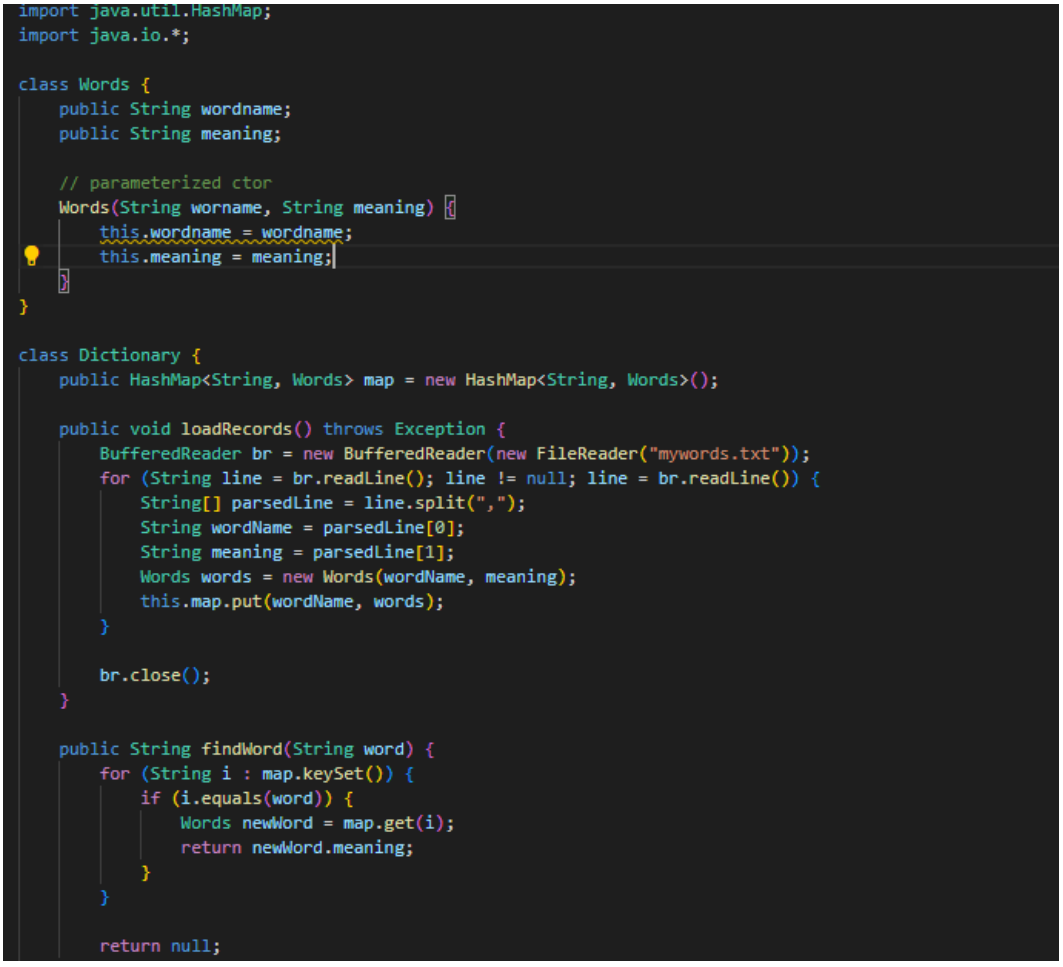
In first program, B-Tree is a self-balancing search tree. In most of the other self-balancing search trees, it is assumed that everything is in the main memory. Like other balanced Binary Search Trees, the time complexity to search, insert and delete is O(log n). All leaves are at the same level and insertion of a Node in B-Tree happens only at leaf node. Search is similar to the search in Binary Search Tree. Let the key to be searched is k. Start from the root and recursively traverse down. For every visited non-leaf node, If the node has the key, we simply return the node. Otherwise, we recur down to the appropriate child (The child which is just before the first greater key) of the node. If we reach a leaf node and don't find k in the leaf node, then return null.

```java
/* As per program requirements, We have implemented B-tree to sort array elements: */

import java.util.*;

class Sort {
    class Mini {
        int value;
        Mini left, right;

        public Mini(int items) {
            value = items;
            left = right = null;
        }
    }

    Mini root;

    Sort() {
        root = null;
    }

    void add(int value) {
        root = insertVal(root, value);
    }

    Mini insertVal(Mini root, int value) {
        if (root == null) {
            root = new Mini(value);
            return root;
        }
        if (value < root.value)
            root.left = insertVal(root.left, value);
        else if (value > root.value)
            root.right = insertVal(root.right, value);
        return root;
    }
```

```java
    void sortedValue(Mini root) {
        if (root != null) {
            sortedValue(root.left);
            System.out.print(root.value + " ");
            sortedValue(root.right);
        }
    }

    void trees(int array[]) {
        for (int i = 0; i < array.length; i++) {
            add(array[i]);
        }
    }

    Run | Debug
    public static void main(String[] args) {
        Sort tree = new Sort();
        int array[] = { 5, 6, 4, 8, 10, 22, 35, 46, 15, 52 };
        tree.trees(array);
        tree.sortedValue(tree.root);
    }
}

/* Explanation:
In this code we used B-tree to sort array elements in ascending order,
after implemented this program we got desired output . */
```

```
PS C:\Users\AliMert\Desktop\dataodevi>  c:; cd 'c:\Users\AliMert\Desktop\dataodevi'; & 'C:\Program Files\Java\jre1.8.0_231\bin\java.exe' '-cp' 'C:\Users\AliMert\AppData\Roaming\Code\User\workspaceStorage\3c67e28248394b64d08f8659abc60548\redhat.java\jdt_ws\dataodevi_10c80330\bin' 'Sort'
4 5 6 8 10 15 22 35 46 52
PS C:\Users\AliMert\Desktop\dataodevi>
```

In second program, Advantages of using binary search tree.It is a fast data structure with a time complexity of O(log n).It is easy to insert and delete elements. It is able to store large amounts of data efficiently.It is easy to traverse the tree in order to find elements. Disadvantages of using binary search tree. It is not suitable for real-time applications as it may take a long time to search for elements.It is not very efficient when dealing with large datasets.It can be difficult to maintain if elements are not inserted in the correct order.Advantages of using Hash Table.It is fast and efficient for searching, inserting and deleting elements.It is able to store large amounts of data efficiently.It is easy to implement and maintain.Disadvantages of using Hash Table.It can be difficult to maintain if elements are not inserted in the correct order.It is not suitable for real-time applications as it may take a long time to search for elements.It can be difficult to implement if the data structure is not implemented properly.



```
import java.util.HashMap;
import java.io.*;

class Words {
    public String wordname;
    public String meaning;

    // parameterized ctor
    Words(String worname, String meaning) {
        this.wordname = wordname;
        this.meaning = meaning;
    }
}

class Dictionary {
    public HashMap<String, Words> map = new HashMap<String, Words>();

    public void loadRecords() throws Exception {
        BufferedReader br = new BufferedReader(new FileReader("mywords.txt"));
        for (String line = br.readLine(); line != null; line = br.readLine()) {
            String[] parsedLine = line.split(",");
            String wordName = parsedLine[0];
            String meaning = parsedLine[1];
            Words words = new Words(wordName, meaning);
            this.map.put(wordName, words);
        }

        br.close();
    }

    public String findWord(String word) {
        for (String i : map.keySet()) {
            if (i.equals(word)) {
                Words newWord = map.get(i);
                return newWord.meaning;
            }
        }

        return null;
```

```
    }

    public void saveRecords() throws Exception {
        BufferedWriter bw = new BufferedWriter(new FileWriter("newmywords.txt"));
        for (String i : map.keySet()) {
            Words w = map.get(i);
            bw.write(w.wordname);
            bw.write(",");
            bw.write(w.meaning);
            bw.write("\n");
        }
        bw.close();
    }

}

public class MyDictionary {
    Run | Debug
    public static void main(String[] args) throws Exception {
        Dictionary obj = new Dictionary();
        obj.loadRecords();
        System.out.println(obj.findWord(word: "snooze"));
        obj.saveRecords();
    }
}
```

```
Data\Roaming\Code\User\workspaceStorage\3c67e28248394b64d08f8659abc60548\redhat.java\jdt_ws\dataodevi_10c80330\bin' 'MyDictionary'
to put to sleep temporarily
Exception in thread "main" java.lang.NullPointerException
        at java.io.Writer.write(Unknown Source)
        at Dictionary.saveRecords(MyDictionary.java:46)
        at MyDictionary.main(MyDictionary.java:61)
PS C:\Users\AliMert\Desktop\dataodevi>
```

In third program, Here We have implemented a class called Stacks for stack creation, push and pop function for inserting and deletion of data from stack. For an efficient implementation of N-stacks, instead of dividing the array equally among the N stacks, we use two extra arrays. The extra arrays used are top and next. The top array is used to keep track of the indices of the top elements of every stack. The next array is an array of size N which is used to keep track of the next (lower) element of the stack and also the next free space available in the array. For any index 'i' in the array, if arr[i] stores an element of a stack, then next[i] stores the index of the next (lower) element in the stack. Otherwise, if arr[i] is a free slot (i.e. doesn't store any element), then next[i] stores the index of the next free slot available in the array.

The advantages and disadvantages of data structure are as follows.Here we have used stacks in array since In array we can't allocate memory dynamically so to remove this drawback we use stack .In stack we can allocate memory dynamically In array it take lots of effort to add new element or remove an element . In stack we can easily add or remove elements from stack. Because of dynamic memory allocation if we not use all memory space then there will be wastage of memory space.Hence it is Less flexible.

```java
import java.util.*;

/**
 * Implement k stacks in array
 * Time - O(1)
 * Space- O(n+k)
 */

public class Stacks {
    int[] arr;
    int[] top;
    int[] next;
    int free;
    int k, n;

    // k is the number of stacks and n is the length of the array
    Stacks(int k, int n) {
        this.k = k;
        this.n = n;
        arr = new int[n];
        top = new int[k];
        next = new int[n];
        for (int i = 0; i < k; i++) {
            top[i] = -1;
        }
        for (int i = 0; i < n - 1; i++) {
            next[i] = i + 1;
        }
        next[n - 1] = -1;
        free = 0;
    }
```

```java
    void push(int data, int kn) {
        if (isFull()) {
            System.out.print("\nStack Overflow\n");
            return;
        }
        int i = free;
        free = next[i];
        next[i] = top[kn];
        top[kn] = i;
        arr[i] = data;
        System.out.print("\nPush element in stack\t " + kn + "\tdata\t " + data);
    }

    int pop(int kn) {
        if (isEmpty(kn)) {
            System.out.print("\nStack overflow\n");
            return Integer.MAX_VALUE;
        }
        int i = top[kn];
        top[kn] = next[i];
        next[i] = free;
        free = i;
        return arr[i];
    }

    boolean isFull() {
        return free == -1;
    }

    boolean isEmpty(int kn) {
        return top[kn] == -1;
    }
```

In last program, We will represent the graph in adjacency matrix its a 2D Array. We are working on prim's algorithm so it's efficient to use adjacency matrix. Main method is for taking the input graph and calling printing function.(prim(graph)).minKey finds the minimum possible edge connected to the given vertex and adding that edge does not creates cycle.prim is the method which creates Array to store constructed ST, and add the minimum possible weighted edges in to our ST.

The advantages of the chart are as follows. By using graphs we can easily find the shortest path, neighbors of the nodes, and many more. Graphs are used to implement algorithms like DFS and BFS. It is used to find minimum spanning tree which has many practical applications. It helps in organizing data.Because of its non-linear structure, helps in understanding complex problems and their visualization.

The disadvantages of the chart are as follows. Graphs use lots of pointers which can be complex to handle. It can have large memory complexity. If the graph is represented with an adjacency matrix then it does not allow parallel edges and multiplication of the graph is also difficult.

```java
import java.io.*;
import java.lang.*;
import java.util.*;

class STgraph {
    private static final int B = 8;

    int minKey(int key[], Boolean Set[]) {
        int min = Integer.MAX_VALUE, min_index = -1;
        for (int b = 0; b < B; b++)
            if (Set[b] == false && key[b] < min) {
                min = key[b];
                min_index = b;
            }
        return min_index;
    }

    void print(int parent[], int graph[][]) {
        int sum = 0;
        for (int i = 1; i < B; i++) {
            System.out.println(parent[i] + " - " + i + "\t"
                    + graph[i][parent[i]]);
            sum = sum + graph[i][parent[i]];
        }
        System.out.println(sum);
    }
```

```java
void prim(int graph[][]) {
    int parent[] = new int[B];
    int key[] = new int[B];
    Boolean Set[] = new Boolean[B];
    for (int i = 0; i < B; i++) {
        key[i] = Integer.MAX_VALUE;
        Set[i] = false;
    }
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < B - 1; count++) {
        int a = minKey(key, Set);
        Set[a] = true;
        for (int b = 0; b < B; b++)
            if (graph[a][b] != 0 && Set[b] == false
                    && graph[a][b] < key[b]) {
                parent[b] = a;
                key[b] = graph[a][b];
            }
    }
    print(parent, graph);
}

Run | Debug
public static void main(String[] args) {
    STgraph t = new STgraph();
    int graph[][] = new int[][] { { 0, 12, 17, 20, 0, 0, 0, 0 },
            { 12, 0, 21, 0, 0, 0, 0, 19 },
            { 17, 21, 0, 4, 88, 0, 6, 0 },
            { 20, 0, 4, 0, 0, 15, 13, 0 },
            { 0, 0, 88, 0, 0, 30, 37, 19 },
            { 0, 0, 0, 15, 30, 0, 44, 0 },
            { 0, 0, 6, 13, 37, 44, 0, 0 },
            { 0, 19, 0, 0, 19, 0, 0, 0 } };
    t.prim(graph);
}
}
```

```
PS C:\Users\AliMert\Desktop\dataodevi>  c:; cd 'c:\Users\AliMert\Desktop\dataodevi'; & 'C:\Program Files\Java\jre1.8.0_231\bin\java.exe' '-cp' 'C:\Users\AliMert\AppData\Roaming\Code\User\workspaceStorage\3c67e28248394b64d08f8659abc60548\redhat.java\jdt_ws\dataodevi_10c80330\bin' 'STgraph'
0 - 1   12
0 - 2   17
2 - 3   4
7 - 4   19
3 - 5   15
2 - 6   6
1 - 7   19
92
PS C:\Users\AliMert\Desktop\dataodevi>
```

Explanation:

A is 0, B is 1, C is 2, D is 3, E is 4, F is 5, G is 6, H is 7

(92 is the total weight)