Data structue 234218 – wet HW 1 dry part:

General data structure:

AVL:

A balanced AVL tree as learned in the lectures, capable of: (where n is the tree size)

-initialization time of O(1)

-insertion, deletion and access(or notification of non existence) time of O(log(n))

-destruction time of O(n)

- ordered travel of all nodes according to a given total order

MAVL:

A wrapper class of the AVL tree the supports the following additional functionalities:

-access to the highest ordered element of the AVL O(1)

-keeping track of the size of the AVL O(1)

- returning an array of elements according to a given total order

Assignment specific data structures:

The main DS class will hold the following fields & methods:

* An AVL tree for trainers sorted by trainer ID (t\_AVL)
* An AVL tree for all pokemons sorted by pokemon ID(p\_AVL)
* An MAVL tree for pokemon sorted by level and then by pokemon id (p\_MAVL).
* All methods outlined in the assignment description

Each trainer class will hold the following fields & methods:

* The trainers ID
* An MAVL tree for the trainer’s pokemon sorted by level and then by pokemon. (tp\_MAVL)
* An initializer
* Id getter
* Wrappers for all methods in the MAVL class
* destroyer

Each pokemon class will hold the following fields & methods:

* Its ID
* Its Level
* Its trainer’s ID
* An initializer
* Id getter
* Level getter and setter
* destroyer

Complexity correctness:

We will explain the idea behind each public DS method and prove it works for the given upper complexity bounds.

***void\* Init():***

We will initialize DS which requires initializations of 2 AVLs and an MAVL, all 3 are initialized O(1), and so the entire initialization is O(1).

***StatusType AddTrainer(void \*DS, int trainerID):***

We check for the trainer’s existence in the trainer AVL (O(log(k)),

If ID is new we create a new trainer.

Trainers creation is O(1) because it contains only 1 int field and an MAVL, both have initialization time of O(1).

Then, we add a new trainer reference to the trainers AVL (O(log(k)).

Total time complexity is C1\*(O(log(k)) + C2\*O(1) = (O(log(k))

(C1,C2 are constants)

***StatusType CatchPokemon(void \*DS, int pokemonID, int trainerID, int level):***

We check for existence of pokemon and trainer by the ID ordered AVL of both classes (O(log(n) and O(log(k)) respectively).

If pokemon does not exist and trainer does, we continue (having accessed the trainer while checking for existence)

We create pokemon (O(1) since pokemon contains only 2 basic fields).

We then add a the pokemon reference to the pokemon AVL and MAVL in DS (O(log(n)) and to the correct trainer’s MAVL (O(log(n))

Total time complexity is C3\*(O(log(n)) + C1\*(O(log(k)) + C2\*O(1) = O(log(n)+O(log(k)) <= O log(n) + O(k)

(Ci are constants)

***StatusType FreePokemon(void \*DS, int pokemonID):***

Check for existence of pokemon in p\_AVL (O(log(n)).

If it exist get its level and trainer ID.

Then using the pokemon ID and level we remove it from the p\_AVL and p\_MAVL (O(log(n))

We access its trainer through the t\_AVL (O(log(k)) and remove it from the trainers tp\_MAVL (O(log(n)).

Total time complexity is C1\*(O(log(n)) + C2\*(O(log(k)) = O(log(n)+O(log(k)) <= O log(n) + O(k)

(Ci are constants)

StatusType LevelUp(void \*DS, int pokemonID, int levelIncrease)