HY-240

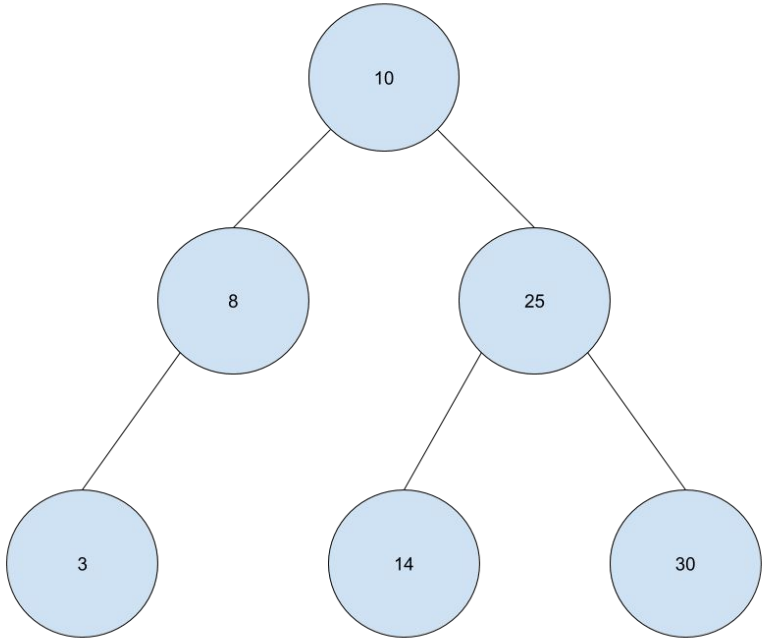
2nd Programming Project

What will you learn

- Binary Search Trees
- Priority Queues (as heaps)
- Complete Trees
- Hash Tables

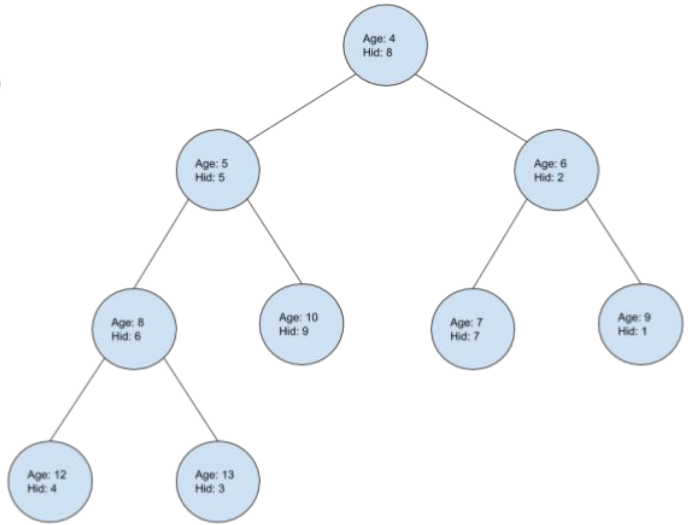
Alexander the Great army

- Implemented as a **Binary Search Tree**
- Sorted on soldier's ID
- struct Army
  - int sid; // soldier's id
  - struct Army *lc;
  - struct Army *rc;



Alexander the Great army

- A priority queue implemented as a **heap**
- Priority is based on the horses' ***age***
- The number of horses will be given
- struct Horse
  - int id;
  - Int age;
- struct HorsePQ
  - int size; // the queue's size at the moment
  - struct Horse *horsePQ;

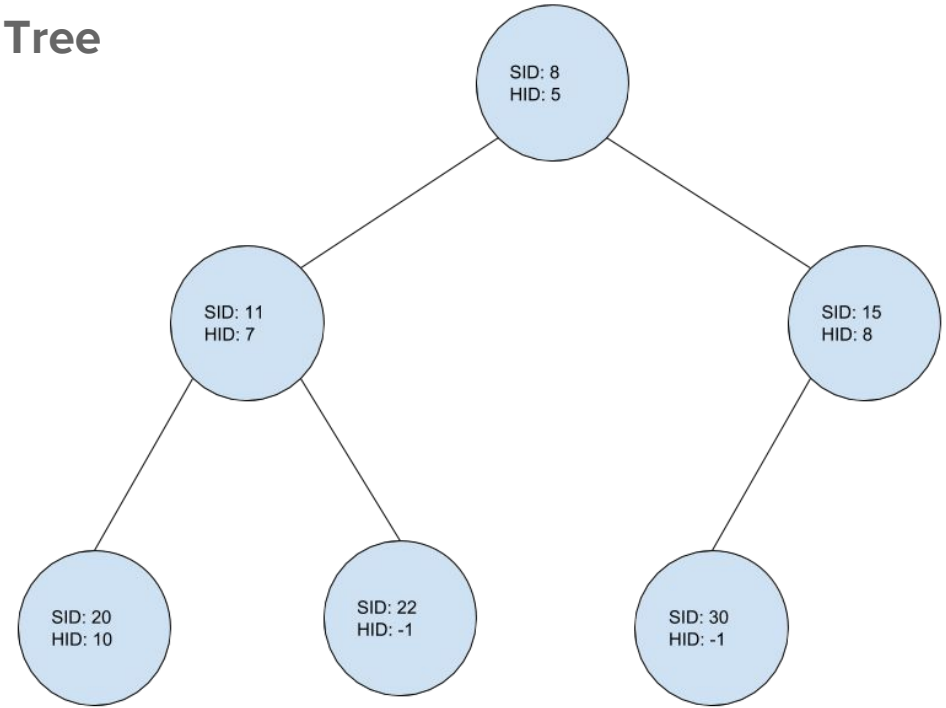


Age: 4 Hid: 8	Age: 5 Hid: 5	Age: 6 Hid: 2	Age: 8 Hid: 6	Age: 10 Hid: 9	Age: 7 Hid: 7	Age: 9 Hid: 1	Age: 12 Hid: 4	Age: 13 Hid: 3		

Σχήμα 2: Η ουρά προτεραιότητας αλόγων του Μεγάλου Αλεξάνδρου. Η προτεραιότητα των κόμβων καθορίζεται από το πεδίο age. Στο πάνω σχήμα βλέπετε βοηθητικά το δένδρο του σωρού.

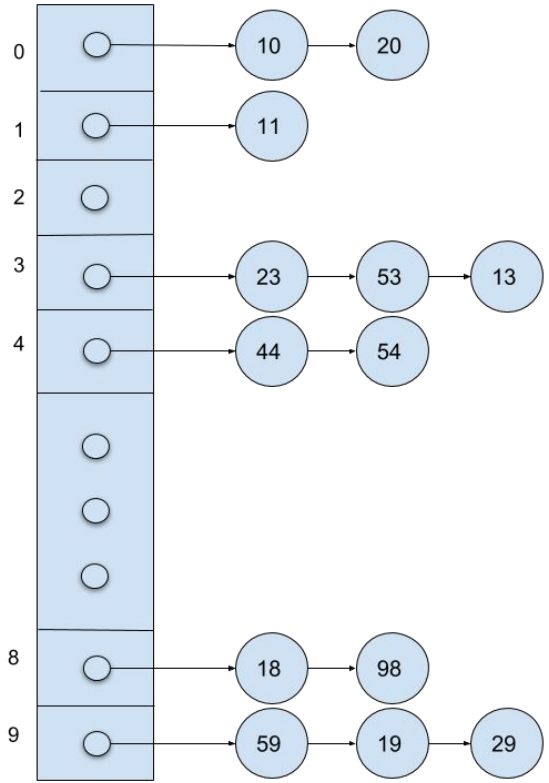
Alexander the Great battle tree

- Implemented as a **Complete Tree**
- struct GA_Battle
  - int soldier_id;
  - int horse_id
  - struct GA_Battle *lc;
  - struct GA_Battle *rc;
- struct GA_Tree
  - int size;
  - struct GA_Battle *Ga_tree;



Enemy's Army

- Implemented as a hash table
- Collisions are handled using the **separate chaining** method
- Use of **universal hashing**
- The max # of soldiers and their max ID will be given
- struct AR_Battle
  - int id;
  - struct AR_Battle *next;



Testfile format & events

- Number of horses // 1st line
- Max # of enemy soldiers // 2nd line
- Max ID of enemy soldiers //3rd line
- Events // Rest of testfile
  - R <sid>
  - H <hid> <age>
  - A <aid>
  - P
  - T <X>
  - K
  - D
  - W,X,Y,Z

Events

- R <sid>
  - Insert a new node in Alexander's army BST
- H <hid> <age>
  - Insert a new node in Alexander's horses priority queue
- A <aid>
  - Insert a new node in enemies' army hash table
- P
  - Traverse Alexander's army BST **in-order**
  - Each soldier reserves a horse (*delete_min*) from the priority queue
    - If there are no available horses, the soldier is on foot
  - Insert a new node with soldier's ID and horse's ID (-1 if on foot) in Alexander's battle complete tree (see function ***CalculatePath***)

Events (continued)

- T <X>
  - Traverse Alexander's battle complete tree
    - You choose in what order
  - Delete *1-every-X* nodes (see function ***CalculatePath***)
  - The tree should remain complete **at all times**
- K
  - Traverse Alexander's battle complete tree (any order)
  - Use each soldier's ID as the input to the hash function
  - Delete the first three enemy soldiers in that chain

Events (continued)

- D
  - Split Alexander's army BST into 5 differents BSTs (one for each of his generals)
  - Soldiers with ID in the range [0, 500) should go in the 1st general's BST, [500, 1000) in the 2nd etc.
  - Complexity should be **O(h)**
    - Where ***h*** is the height of Alexander's BST
- W, X, Y, Z
  - Print your data structures (see project)

Bonus

- Implement Alexander's battle tree as an AVL tree (instead of a complete tree)
- The AVL tree should support **only insertions**, nothing more

- The bonus should be in a **different directory**
- All other events and data structures are **mandatory**

General Info

- Your project should:
  - Contain everything it needs to compile
  - Compile and run in CSD UNIX machines
- For any questions contact the TAs using the course's e-mail or list (hy240a@csd.uoc.gr)