GoDoc

Package dsa

```
import "goInAction2/assignment/packages/dsa"
```


Overview

ticket.go: Ticket is a custom struct containing all the fields to be tracked, pivoted, edited and displayed. It underlies both the priority queue and AVL tree data structures implemented in the application. For storage and handling of the data across function calls, Tickets are contained within TicketNodes, which contain other fields required to store Tickets within the ticket log AVL tree. A Ticket can only be created by a non-admin user. When a non-admin user creates a ticket, it is not directly added into the ticket log AVL tree; rather, it is first added to a priority queue called submissions, which is implemented via an array-based heap. From submissions, the ticket must first be approved by an admin user before the ticket is wrapped in a TicketNode and added to the ticket log AVL tree.

avltree.go: Implements an adapted AVL tree and associated functions to initialize, traverse, insert nodes, delete nodes and others. Each AVLtree contains a pointer to the root node (of type TicketNode), as well as a specification of the sorting criteria applied by the AVLtree: - Ticket ID - Product - Status - Category - Priority - Estimated Hours to Complete - Start Date - Due Date - Ticket Creator - Ticket Title - Ticket Description - Ticket Assignee AVLtrees can be pivoted to apply a different sorting criteria, changing the order in which tickets are displayed.

heap.go: Array-based implementation of a heap, which is used as a priority queue used to track user submissions. Note that in this heap, the topmost tickets are the highest-priority ones (lowest priority score). Thus, an admin appoving user ticket submissions will first start from the highest-priority ticket, making use of the min-heap property.

userhash.go: Implements a hash table, used in the application to record and manipulate information of user accounts in-memory. Hash table is implemented as an array of SLLs made up of UserNodes. UserNodes contain a User, and a pointer to another UserNode (thus forming the SLL). User info includes a username(string), bcrypt-hashed password([]byte), and admin status(bool)

Index ▾

Package files

avltree.go doc.go heap.go ticket.go userhash.go

Constants

```
const (
    Hashbuckets = 50 // Length of index array in hash table
)
```

Variables

EmptyUser is a placeholder variable for functions to return a nil result.

```
var EmptyUser = User{
    Name:  "",
    Pw:    []byte{},
    Admin: false,
}
```

func **AddUser**

```
func AddUser(hashtable *[]*UserNode, newuser User)
```

AddUser adds UserNode to SLL (hash bucket). Appends to front of the SLL for O(1) addition.

func **Addsubmission**

```
func Addsubmission(submissions *[]Ticket, newticket Ticket)
```

Addsubmission inserts a new node to the priority queue.

func **ByAssignee**

```
func ByAssignee(newticket *TicketNode, junction *TicketNode) bool
```

ByAssignee -- uses ByTicket ID as tiebreaker

func **ByCategory**

```
func ByCategory(newticket *TicketNode, junction *TicketNode) bool
```

ByCategory -- uses ByTicket ID as tiebreaker

func **ByCreator**

```
func ByCreator(newticket *TicketNode, junction *TicketNode) bool
```

ByCreator -- uses ByTicket ID as tiebreaker

func **ByDescription**

```
func ByDescription(newticket *TicketNode, junction *TicketNode) bool
```

ByDescription -- uses ByTicket ID as tiebreaker

func **ByDueDate**

```
func ByDueDate(newticket *TicketNode, junction *TicketNode) bool
```

ByDueDate -- uses ByTicket ID as tiebreaker

---

func **ByEstHours**

```
func ByEstHours(newticket *TicketNode, junction *TicketNode) bool
```

ByEstHours -- uses ByTicket ID as tiebreaker

func **ByPriority**

```
func ByPriority(newticket *TicketNode, junction *TicketNode) bool
```

ByPriority -- uses ByTicket ID as tiebreaker

func **ByProduct**

```
func ByProduct(newticket *TicketNode, junction *TicketNode) bool
```

ByProduct -- uses ByTicket ID as tiebreaker

func **ByStartDate**

```
func ByStartDate(newticket *TicketNode, junction *TicketNode) bool
```

ByStartDate -- uses ByTicket ID as tiebreaker

func **ByStatus**

```
func ByStatus(newticket *TicketNode, junction *TicketNode) bool
```

ByStatus -- uses ByTicket ID as tiebreaker

func **ByTicketID**

```
func ByTicketID(newticket *TicketNode, junction *TicketNode) bool
```

ByTicketID

func **ByTitle**

```
func ByTitle(newticket *TicketNode, junction *TicketNode) bool
```

ByTitle -- uses ByTicket ID as tiebreaker

func **DeleteUser**

```
func DeleteUser(hashtable *[]*UserNode, username string)
```

DeleteUser deletes UserNode with corresponding username from SLL(hash bucket). Assumes username is already verified to exist.

func **EditUser**

```
func EditUser(hashtable *[]*UserNode, retrieved, edited User)
```

EditUser modifies an existing user.

func **IOtraversal**

```
func IOtraversal(avlroot *TicketNode, priorities, products, statuses,
categories *[]string, result [][]string) [][]string
```

IOtraversal implements an in-order traversal of an already-existing AVL tree, for which the root pointer is passed as the first argument.

func **LOtraversal**

```
func LOtraversal(submissions *[]Ticket, priorities, products, statuses,
categories *[]string) [][]string
```

LOtraversal implements in-order traversal of the heap.

---

func **Makeheap**

```
func Makeheap(submissions *[]Ticket, root int)
```

Makeheap preserves the heap property of the priority queue.

func **NewHT**

```
func NewHT() *[]*UserNode
```

NewHT initializes the hash table.

func **PrintHT**

```
func PrintHT(hashtable *[]*UserNode, printfunc func(SLL *UserNode)
[]string) [][]string
```

PrintHT returns a slice of slices of strings to be passed into the relevant HTML template for printing in the client. Reflects all users currently recorded in hash table.

func **PrintSLLnoadmin**

```
func PrintSLLnoadmin(SLL *UserNode) []string
```

PrintSLLnoadmin function taken as argument for PrintHT(); prints username of non-admin users only (for non-admin users to set ticket assignee)

func **PrintSLLusername**

```
func PrintSLLusername(SLL *UserNode) []string
```

PrintSLLusername function taken as argument for PrintHT(); prints usernames of admin and non-admin users (login screen ref)

func **Searchsubmissions**

```
func Searchsubmissions(submissions *[]Ticket, ticketID int64) (bool,
int64)
```

Searchsubmissions for a given node in priority queue. Returns true if the ticket exists, alongside the target ticket ID.

type **AVLtree**

AVLtree is a struct describing AVL trees as used in this application. It comprises a pointer to the AVL tree root, as well as the sorting criteria (out of TicketNode fields) used to build the tree.

```
type AVLtree struct {
    Root     *TicketNode
    Sortfunc func(newticket *TicketNode, junction *TicketNode) bool
}
```

func NewAVLT

```
func NewAVLT(sortfunc func(newticket *TicketNode, junction *TicketNode)
bool) *AVLtree
```

NewAVLT creates a new AVLTree specified with some sortfunc, and returns its pointer.

type **Ticket**

```
type Ticket struct {
    TicketID                                    int64
    Product, Status, Category, Priority, EstHours int
    StartDate, DueDate                          time.Time
    Creator, Title, Description, Assignee       string
}
```

func Newticket

```
func Newticket(
    ticketID int64,
    product, status, category, priority, estHours int,
    startDate, dueDate time.Time,
    creator, title, description, assignee string,
    priorities, products, statuses, categories *[]string,
) Ticket
```

Newticket creates a new Ticket (taking user input) and encapsulates it in a TicketNode for processing in priority queue / AVL tree, returning its pointer.

func Popsubmission

```
func Popsubmission(submissions *[]Ticket) Ticket
```

Popsubmission removes the root node of the priority queue, returning the removed node.

type TicketNode

TicketNode struct specifies fields for an Ticket struct, two pointers to other Nodes within storage AVL Tree, and height for maintaining balance within AVL tree.

```
type TicketNode struct {
    Ticket Ticket
    Height int
    Left   *TicketNode
    Right  *TicketNode
}
```

func AVLdelete

```
func AVLdelete(subtree *TicketNode, targetID int64) *TicketNode
```

AVLdelete recursively deletes a target node (with a certain TicketID). Returns root of the modified subtree. Important: Assumes tree is sorted by ticketID.

func AVLinsert

```
func AVLinsert(newticket *TicketNode,
    sortfunc func(newticket *TicketNode, junction *TicketNode) bool,
    subtree *TicketNode) *TicketNode
```

AVLinsert recursively inserts a node root of a specified subtree, does required rotations. For valid sortfuncs to use as arguments, see section below on AVLtree sortfuncs. Returns new root of the subtree.

func AVLpivot

```
func AVLpivot(source, destination *TicketNode,
    sortfunc func(newticket *TicketNode, junction *TicketNode) bool)
*TicketNode
```

AVLpivot takes an existing AVL tree and re-sorts it using a different sorting function. Returns a pointer to the re-sorted AVL tree. For valid sortfuncs to use as arguments, see section below on AVLtree sortfuncs.

func AVLsearch

```
func AVLsearch(subtree *TicketNode, targetID int64) *TicketNode
```

AVLsearch searches the AVLtree (sorted by ticketID only) and returns to pointer to that node, if it exists (otherwise, returns nil).

func MinIDnode

```
func MinIDnode(subtree *TicketNode) *TicketNode
```

MinIDnode returns the node within a subtree with the minimum key value in that tree

func Myassigns

```
func Myassigns(avlroot, result *TicketNode,
    sortfunc func(newticket *TicketNode, junction *TicketNode) bool,
    user string) *TicketNode
```

Myassigns traverses an AVL tree (at a given root node), and returns pointer to a subsetted AVL tree containing only nodes assigned to a particular username.

---

func Mytickets

```
func Mytickets(avlroot, result *TicketNode,
    sortfunc func(newticket *TicketNode, junction *TicketNode) bool,
    user string) *TicketNode
```

Mytickets traverses an AVL tree (at a given root node), and returns pointer to a subsetted AVL tree containing only nodes with a particular username as creator.

type User

User struct logs fields for managing login user/admin accounts.

```
type User struct {
    Name  string
    Pw    []byte
    Admin bool
}
```

type UserNode

UserNode specifies a SLL node in the Userlog hash table.

```
type UserNode struct {
    User User
    // contains filtered or unexported fields
}
```

func SearchUser

```
func SearchUser(hashtable *[]*UserNode, username string) (bool,
*UserNode)
```

SearchUser looks up particular value from hash table: checks for presence of a particular username.