BinarySearchTree

Generated by Doxygen 1.8.11

Contents

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

BST_t **2**

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

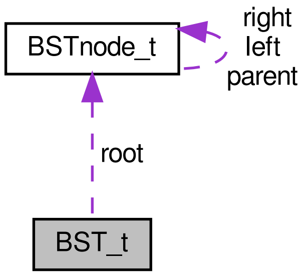
bst.c	5
bst.h **Binary search tree definition and basic operations**	11

3 Data Structure Documentation

3.1 BST_t Struct Reference

```
#include <bst.h>
```

Collaboration diagram for BST_t:



Data Fields

- size_t width
- BSTnode_t ∗ root
- int count
- int height
- int(∗ compare)(const void ∗, const void ∗)

3.1.1 Detailed Description

Definition at line 22 of file bst.h.

3.1.2 Field Documentation

3.1.2.1 int(∗ compare) (const void ∗, const void ∗)

Definition at line 27 of file bst.h.

3.1.2.2 int count

count element amount

Definition at line 25 of file bst.h.

3.1.2.3 int height

tree height

Definition at line 26 of file bst.h.

3.1.2.4 BSTnode_t∗ root

root node

Definition at line 24 of file bst.h.

3.1.2.5 size_t width

element size (in bytes)

Definition at line 23 of file bst.h.

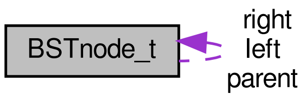
The documentation for this struct was generated from the following file:

- bst.h

3.2 BSTnode_t Struct Reference

```
#include <bst.h>
```

Collaboration diagram for BSTnode_t:



Data Fields

- struct BSTnode_t ∗ parent
- struct BSTnode_t ∗ left
- struct BSTnode_t ∗ right
- void ∗ data

3.2.1 Detailed Description

Definition at line 15 of file bst.h.

3.2.2 Field Documentation

3.2.2.1 void∗ data

data pointer

Definition at line 19 of file bst.h.

3.2.2.2 struct BSTnode_t∗ left

left child

Definition at line 17 of file bst.h.

3.2.2.3 struct BSTnode_t∗ parent

parent node

Definition at line 16 of file bst.h.

3.2.2.4 struct BSTnode_t∗ right

right child

Definition at line 18 of file bst.h.

The documentation for this struct was generated from the following file:

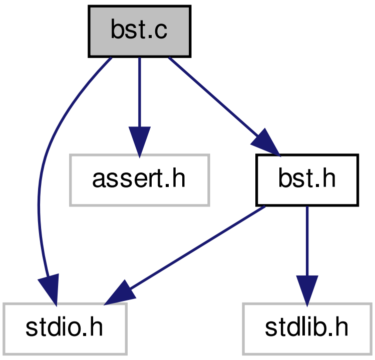
- bst.h

4 File Documentation

4.1 bst.c File Reference

```
#include <stdio.h>
#include <assert.h>
#include "bst.h"
```

Include dependency graph for bst.c:



Functions

- BST_t ∗ bst_create (size_t width, int(∗compare)(const void ∗, const void ∗))
- void bst_destruct (BST_t ∗bst)
- BSTnode_t ∗ bst_search (BST_t ∗bst, void ∗data)
- BSTnode_t ∗ bst_iterative_search (BST_t ∗bst, void ∗data)
- BSTnode_t ∗ bst_min (BSTnode_t ∗node)
- BSTnode_t ∗ bst_max (BSTnode_t ∗node)
- BSTnode_t ∗ bst_succ (BST_t ∗bst, BSTnode_t ∗node)
- BSTnode_t ∗ bst_pred (BST_t ∗bst, BSTnode_t ∗node)
- void bst_insert (BST_t ∗bst, void ∗data)
- void bst_delete (BST_t ∗bst, BSTnode_t ∗node)
- void bst_inorder_walk_int (BSTnode_t ∗node)

4.1.1 Function Documentation

4.1.1.1 BST_t∗ bst_create (size_t *width,* int(∗)(const void ∗, const void ∗) *compare*)

Definition at line 6 of file bst.c.

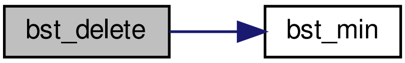
```
00006                                                                               {
00007     BST_t *bst = malloc(sizeof(BST_t));
00008     assert(bst);
00009     bst->root = NULL;
00010     bst->count = 0;
00011     bst->height = 0;
00012     bst->width = width;
00013     bst->compare = compare;
00014     return bst;
00015 }
```

4.1.1.2 void bst_delete (**BST_t** ∗ ***bst,*** **BSTnode_t** ∗ ***node***)

Definition at line 164 of file bst.c.

```
00164                                             {
00165     if (!node->left){
00166         bst_transplant(bst, node, node->right);
00167     } else if (!node->right){
00168         bst_transplant(bst, node, node->left);
00169     } else {
00170         BSTnode_t* y = bst_min(node->right);
00171         if (y->parent != node){
00172             bst_transplant(bst, y, y->right);
00173             y->right = node->right;
00174             y->right->parent = y;
00175         }
00176         bst_transplant(bst, node, y);
00177         y->left = node->left;
00178         y->left->parent = y;
00179     }
00180 }
```

Here is the call graph for this function:



4.1.1.3 void bst_destruct (**BST_t** ∗ ***bst***)

Definition at line 28 of file bst.c.

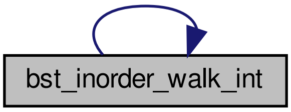
```
00028                                {
00029     bst_destruct_node(bst->root);
00030     free(bst);
00031 }
```

4.1.1.4 void bst_inorder_walk_int (**BSTnode_t** ∗ ***node***)

Definition at line 184 of file bst.c.

```
00184                                           {
00185     if(!node) return;
00186     bst_inorder_walk_int(node->left);
00187     printf("%d\n", *((int*)(node->data)));
00188     bst_inorder_walk_int(node->right);
00189 }
```

Here is the call graph for this function:



4.1.1.5 void bst_insert (BST_t ∗ *bst,* void ∗ *data*)

Definition at line 140 of file bst.c.

```
00140                                        {
00141     BSTnode_t* node = malloc(sizeof(BSTnode_t));
00142     node->data = data;
00143     node->parent = NULL;
00144     node->left = NULL;
00145     node->right = NULL;
00146     bst_insert_node(bst, node);
00147 }
```

4.1.1.6 BSTnode_t∗ bst_iterative_search (BST_t ∗ *bst,* void ∗ *data*)

Definition at line 50 of file bst.c.

```
00050                                                         {
00051     BSTnode_t *node = bst->root;
00052 
00053     while (node && !bst->compare(data, node->data)) {
00054         if (bst->compare(data, node->data) < 0) {
00055             node = node->left;
00056 
00057         } else {
00058             node = node->right;
00059         }
00060     }
00061 
00062     return node;
00063 }
```

4.1.1.7 BSTnode_t∗ bst_max (BSTnode_t ∗ *node*)

Definition at line 74 of file bst.c.

```
00074                                     {
00075     while (node->right) {
00076         node = node->right;
00077     }
00078 
00079     return node;
00080 }
```

4.1.1.8 BSTnode_t∗ bst_min (BSTnode_t ∗ *node*)

Definition at line 65 of file bst.c.

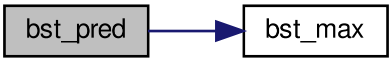
```
00065                                     {
00066     while (node->left) {
00067         node = node->left;
00068     }
00069 
00070     return node;
00071 }
```

4.1.1.9 BSTnode_t* bst_pred (BST_t * *bst,* BSTnode_t * *node*)

Definition at line 97 of file bst.c.

```
00097                                                 {
00098     if (node->left) {
00099         return bst_max(node->left);
00100     }
00101 
00102     BSTnode_t *x = node->parent;
00103 
00104     while (x && x->left && bst->compare(node->data, x->left->
      data) == 0) {
00105         node = x;
00106         x = x->parent;
00107     }
00108 
00109     return x;
00110 }
```

Here is the call graph for this function:



4.1.1.10 BSTnode_t* bst_search (BST_t * *bst,* void * *data*)

Definition at line 46 of file bst.c.

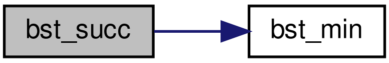
```
00046                                              {
00047     return bst_search_node(bst->root, data, bst->compare);
00048 }
```

4.1.1.11 BSTnode_t* bst_succ (BST_t * *bst,* BSTnode_t * *node*)

Definition at line 82 of file bst.c.

```
00082                                                 {
00083     if (node->right) {
00084         return bst_min(node->right);
00085     }
00086 
00087     BSTnode_t *x = node->parent;
00088 
00089     while (x && x->right && bst->compare(node->data, x->right->
      data) == 0) {
00090         node = x;
00091         x = x->parent;
00092     }
00093 
00094     return x;
00095 }
```

Here is the call graph for this function:



4.2 bst.c

```c
#include <stdio.h>
#include <assert.h>

#include "bst.h"

BST_t *bst_create(size_t width, int (*compare)(const void *, const void *)) {
    BST_t *bst = malloc(sizeof(BST_t));
    assert(bst);
    bst->root = NULL;
    bst->count = 0;
    bst->height = 0;
    bst->width = width;
    bst->compare = compare;
    return bst;
}

static void bst_destruct_node(BSTnode_t *node) {
    if (!node) {
        return;
    }

    free(node->data);
    bst_destruct_node(node->left);
    bst_destruct_node(node->right);
    free(node);
}

void bst_destruct(BST_t *bst) {
    bst_destruct_node(bst->root);
    free(bst);
}

static BSTnode_t *bst_search_node(BSTnode_t *node, void *data, int (*compare)(const void *, const void *)) {
    if (node || !compare(data, node->data)) {
        return node;
    }

    if (compare(data, node->data) < 0) {
        return bst_search_node(node->left, data, compare);

    } else {
        return bst_search_node(node->right, data, compare);
    }
}

BSTnode_t *bst_search(BST_t *bst, void *data) {
    return bst_search_node(bst->root, data, bst->compare);
}

BSTnode_t *bst_iterative_search(BST_t *bst, void *data) {
    BSTnode_t *node = bst->root;

    while (node && !bst->compare(data, node->data)) {
        if (bst->compare(data, node->data) < 0) {
            node = node->left;

        } else {
            node = node->right;
        }
    }

    return node;
}

BSTnode_t *bst_min(BSTnode_t *node) {
    while (node->left) {
        node = node->left;
    }

    return node;
}


BSTnode_t *bst_max(BSTnode_t *node) {
    while (node->right) {
        node = node->right;
    }

    return node;
}

BSTnode_t *bst_succ(BST_t *bst, BSTnode_t *node) {
    if (node->right) {
        return bst_min(node->right);
    }

    BSTnode_t *x = node->parent;

    while (x && x->right && bst->compare(node->data, x->right->data) == 0) {
        node = x;
        x = x->parent;
    }

    return x;
}

BSTnode_t *bst_pred(BST_t *bst, BSTnode_t *node) {
    if (node->left) {
        return bst_max(node->left);
    }

    BSTnode_t *x = node->parent;

    while (x && x->left && bst->compare(node->data, x->left->data) == 0) {
        node = x;
        x = x->parent;
    }

    return x;
}

//TODO:update height and count
void static bst_insert_node(BST_t *bst, BSTnode_t *node) {
    BSTnode_t *x = bst->root, *y = NULL;

    while (x) {
        y = x;

        if (bst->compare(node->data, x->data) < 0) {
            x = x->left;

        } else {
            x = x->right;
        }
    }

    node->parent = y;

    if (!y) {
        bst->root = node;

    } else if (bst->compare(node->data, y->data) < 0) {
        y->left = node;

    } else {
        y->right = node;
    }
}

void bst_insert(BST_t* bst, void* data){
    BSTnode_t* node = malloc(sizeof(BSTnode_t));
    node->data = data;
    node->parent = NULL;
    node->left = NULL;
    node->right = NULL;
    bst_insert_node(bst, node);
}
```

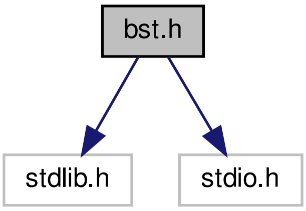
```

//TODO:update height and count
static void bst_transplant(BST_t* bst, BSTnode_t* u, BSTnode_t* v){
    if (!u->parent){
        bst->root = v;
    } else if (u == u->parent->left){
       u->parent->left = v;
    } else {
       u->parent->right = v;
    }
    if (v){
        v->parent = u->parent;
    }
}

//TODO:update height and count
void bst_delete(BST_t* bst, BSTnode_t* node){
    if (!node->left){
        bst_transplant(bst, node, node->right);
    } else if (!node->right){
        bst_transplant(bst, node, node->left);
    } else {
        BSTnode_t* y = bst_min(node->right);
        if (y->parent != node){
            bst_transplant(bst, y, y->right);
            y->right = node->right;
            y->right->parent = y;
        }
        bst_transplant(bst, node, y);
        y->left = node->left;
        y->left->parent = y;
    }
}


//TODO:generalize this function
void bst_inorder_walk_int(BSTnode_t* node){
    if(!node) return;
    bst_inorder_walk_int(node->left);
    printf("%d\n", *((int*)(node->data)));
    bst_inorder_walk_int(node->right);
}

```

4.3 bst.h File Reference

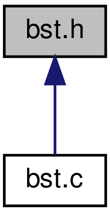
binary search tree definition and basic operations

```
#include <stdlib.h>
#include <stdio.h>
```

Include dependency graph for bst.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct BSTnode_t
- struct BST_t

Typedefs

- typedef struct BSTnode_t BSTnode_t
- typedef struct BST_t BST_t

Functions

- BST_t ∗ bst_create (size_t width, int(∗compare)(const void ∗, const void ∗))
- void bst_destruct (BST_t ∗bst)
- void bst_insert (BST_t ∗bst, void ∗data)
- void bst_delete (BST_t ∗bst, BSTnode_t ∗node)
- BSTnode_t ∗ bst_search (BST_t ∗bst, void ∗data)
- BSTnode_t ∗ bst_iterative_search (BST_t ∗bst, void ∗data)
- BSTnode_t ∗ bst_max (BSTnode_t ∗node)
- BSTnode_t ∗ bst_min (BSTnode_t ∗node)
- BSTnode_t ∗ bst_succ (BST_t ∗bst, BSTnode_t ∗node)
- BSTnode_t ∗ bst_pred (BST_t ∗bst, BSTnode_t ∗node)
- void bst_inorder_walk_int (BSTnode_t ∗node)

4.3.1 Detailed Description

binary search tree definition and basic operations

Author

Firmin MARTIN

Version

0.1

Date

16/03/2018

Definition in file bst.h.

4.3.2 Typedef Documentation

4.3.2.1 typedef struct BST_t BST_t

4.3.2.2 typedef struct BSTnode_t BSTnode_t

4.3.3 Function Documentation

4.3.3.1 BST_t∗ bst_create (size_t *width,* int(∗)(const void ∗, const void ∗) *compare*)

Definition at line 6 of file bst.c.

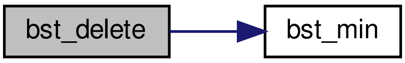
```
00006                                                                      {
00007     BST_t *bst = malloc(sizeof(BST_t));
00008     assert(bst);
00009     bst->root = NULL;
00010     bst->count = 0;
00011     bst->height = 0;
00012     bst->width = width;
00013     bst->compare = compare;
00014     return bst;
00015 }
```

4.3.3.2 void bst_delete (BST_t ∗ *bst,* BSTnode_t ∗ *node*)

Definition at line 164 of file bst.c.

```
00164                                              {
00165     if (!node->left){
00166         bst_transplant(bst, node, node->right);
00167     } else if (!node->right){
00168         bst_transplant(bst, node, node->left);
00169     } else {
00170         BSTnode_t* y = bst_min(node->right);
00171         if (y->parent != node){
00172             bst_transplant(bst, y, y->right);
00173             y->right = node->right;
00174             y->right->parent = y;
00175         }
00176         bst_transplant(bst, node, y);
00177         y->left = node->left;
00178         y->left->parent = y;
00179     }
00180 }
```

Here is the call graph for this function:



4.3.3.3 void bst_destruct (BST_t ∗ *bst*)

Definition at line 28 of file bst.c.

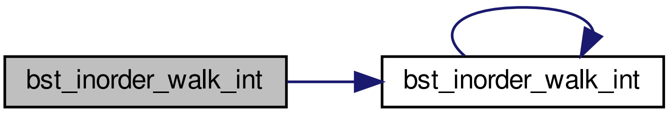
```
00028                             {
00029     bst_destruct_node(bst->root);
00030     free(bst);
00031 }
```

4.3.3.4 void bst_inorder_walk_int (BSTnode_t ∗ *node*)

Definition at line 184 of file bst.c.

```
00184                                        {
00185     if(!node) return;
00186     bst_inorder_walk_int(node->left);
00187     printf("%d\n", *((int*)(node->data)));
00188     bst_inorder_walk_int(node->right);
00189 }
```

Here is the call graph for this function:



4.3.3.5 void bst_insert (BST_t ∗ *bst,* void ∗ *data*)

Definition at line 140 of file bst.c.

```
00140                                     {
00141     BSTnode_t* node = malloc(sizeof(BSTnode_t));
00142     node->data = data;
00143     node->parent = NULL;
00144     node->left = NULL;
00145     node->right = NULL;
00146     bst_insert_node(bst, node);
00147 }
```

4.3.3.6 BSTnode_t∗ bst_iterative_search (BST_t ∗ *bst,* void ∗ *data*)

Definition at line 50 of file bst.c.

```
00050                                                   {
00051     BSTnode_t *node = bst->root;
00052 
00053     while (node && !bst->compare(data, node->data)) {
00054         if (bst->compare(data, node->data) < 0) {
00055             node = node->left;
00056 
00057         } else {
00058             node = node->right;
00059         }
00060     }
00061 
00062     return node;
00063 }
```

4.3.3.7 BSTnode_t* bst_max (BSTnode_t * *node*)

Definition at line 74 of file bst.c.

```
00074                                   {
00075     while (node->right) {
00076         node = node->right;
00077     }
00078 
00079     return node;
00080 }
```

4.3.3.8 BSTnode_t* bst_min (BSTnode_t * *node*)

Definition at line 65 of file bst.c.

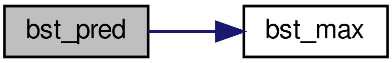
```
00065                                   {
00066     while (node->left) {
00067         node = node->left;
00068     }
00069 
00070     return node;
00071 }
```

4.3.3.9 BSTnode_t* bst_pred (BST_t * *bst,* BSTnode_t * *node*)

Definition at line 97 of file bst.c.

```
00097                                                {
00098     if (node->left) {
00099         return bst_max(node->left);
00100     }
00101 
00102     BSTnode_t *x = node->parent;
00103 
00104     while (x && x->left && bst->compare(node->data, x->left->
      data) == 0) {
00105         node = x;
00106         x = x->parent;
00107     }
00108 
00109     return x;
00110 }
```

Here is the call graph for this function:



4.3.3.10 BSTnode_t* bst_search (BST_t * *bst,* void * *data*)

Definition at line 46 of file bst.c.

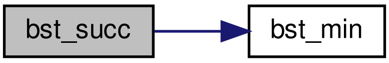
```
00046                                             {
00047     return bst_search_node(bst->root, data, bst->compare);
00048 }
```

4.3.3.11 BSTnode_t* bst_succ (BST_t * *bst,* BSTnode_t * *node*)

Definition at line 82 of file bst.c.

```
00082                                                   {
00083     if (node->right) {
00084         return bst_min(node->right);
00085     }
00086 
00087     BSTnode_t *x = node->parent;
00088 
00089     while (x && x->right && bst->compare(node->data, x->right->
      data) == 0) {
00090         node = x;
00091         x = x->parent;
00092     }
00093 
00094     return x;
00095 }
```

Here is the call graph for this function:



4.4 bst.h

```
#ifndef BST_H
#define BST_H

#include <stdlib.h>
#include <stdio.h>

typedef struct BSTnode_t {
    struct BSTnode_t* parent;
    struct BSTnode_t* left;
    struct BSTnode_t* right;
    void* data;
}BSTnode_t;

typedef struct BST_t {
    size_t width;
    BSTnode_t* root;
    int count;
    int height;
    int (*compare)(const void *, const void *);
}BST_t;

BST_t* bst_create(size_t width, int (*compare)(const void *, const void *));
void bst_destruct(BST_t *bst);
void bst_insert(BST_t *bst, void *data);
void bst_delete(BST_t* bst, BSTnode_t* node);
BSTnode_t* bst_search(BST_t *bst, void *data);
BSTnode_t* bst_iterative_search(BST_t *bst, void *
data);
BSTnode_t* bst_max(BSTnode_t *node);
BSTnode_t* bst_min(BSTnode_t *node);
BSTnode_t* bst_succ(BST_t *bst, BSTnode_t *node);
BSTnode_t* bst_pred(BST_t *bst, BSTnode_t *node) ;
void bst_inorder_walk_int(BSTnode_t* node);

#endif
```

Index