AVL_tree.a

AVL樹使用說明書

611121212 李奕承

摘要

本library提供的結構如下

```
typedef struct avl_node {
    struct avl_node *left;
    struct avl_node *right;
    int height;
} avl_node_t;
```

本 library 提供以下幾種函數

- 節點初始化函數
- 印出樹函數
- 插入節點函數
- 尋找節點函數
- 尋找最小節點函數
- 尋找最大節點函數
- 刪除節點函數

使用步驟

1.帶入標頭檔

```
#include"AVLSpec.h"
```

2.創造AVL tree

宣告一個 avl_node_t 的空指針用來指向樹的root，此處以 avl_root_a 為例

```
avl_node_t * avl_root_a = NULL;
```

3.編譯

- AVLSpec.h 要記得放在引用標頭檔的路徑下
- 編譯時記得加入 AVL_tree.a

```
gcc -I include -Wall -o test test.c AVL_tree.a
```

函數介紹與範例

範例使用的結構

下面是範例用到的結構

```
typedef struct test_avl{
    char name;
    int value;
    avl_node_t node;
}test_avl_t;
```

下面的回調函數用來比較兩個 test_avl_t 節點中的 value 值

```
int compare_value(void * elementA, void * elementB)
{
    if(return_to_user_struct_pointer(test_avl_t, node, elementA)->value <
       return_to_user_struct_pointer(test_avl_t, node, elementB)->value)
    {
        return -1;
    }
    else if(return_to_user_struct_pointer(test_avl_t, node, elementA)->value ==
            return_to_user_struct_pointer(test_avl_t, node, elementB)->value)
    {
        return 0;
    }
    else if(return_to_user_struct_pointer(test_avl_t, node, elementA)->value >
            return_to_user_struct_pointer(test_avl_t, node, elementB)->value)
    {
        return 1;
    }
    else
    {
        return 2;
    }
}
```

下面的回調函數用給定的值來比較 test_avl_t 節點中的value值

```
int compare_key(int key, void * in_tree_element)
{
    if(key <
       return_to_user_struct_pointer(test_avl_t, node, in_tree_element)->value)
    {
        return -1;
    }
    else if(key ==
            return_to_user_struct_pointer(test_avl_t, node, in_tree_element)->value)
    {
        return 0;
    }
    else
    {
        return 1;
    }
}
```

下面的回調函數用 avl_node_t 指標取得 test_avl_t 中的value值

```
int get_value(avl_node_t * element)
{
    return return_to_user_struct_pointer(test_avl_t, node, element)->value;
}
```

下面的回調函數用 avl_node_t 指標取得 test_avl_t 中的name值

```
int get_name(avl_node_t * element)
{
    return return_to_user_struct_pointer(test_avl_t, node, element)->name;
}
```

下面的回調函數用 avl_node_t 指標取得該節點的樹高

```
int get_height(avl_node_t * element)
{
    return element->height;
}
```

節點初始化函數

```
void AVL_init(avl_node_t * node);
```

當你宣告一個包含 avl_node_t 的結構之後，可以使用這個函數來初始化 avl_node_t 中的值

以下是範例中創建的root指標及結構

```
avl_node_t * avl_root_a = NULL;


test_avl_t a;
a.name = 'A';
a.value = 20;
AVL_init(&a.node);

test_avl_t b;
b.name = 'B';
b.value = 4;
AVL_init(&b.node);

test_avl_t c;
c.name = 'C';
c.value = 26;
AVL_init(&c.node);

test_avl_t d;
d.name = 'D';
d.value = 3;
AVL_init(&d.node);

test_avl_t e;
e.name = 'E';
e.value = 9;
AVL_init(&e.node);

test_avl_t f;
f.name = 'F';
f.value = 21;
AVL_init(&f.node);

test_avl_t g;
g.name = 'G';
g.value = 30;
AVL_init(&g.node);

test_avl_t h;
h.name = 'H';
h.value = 2;
AVL_init(&h.node);

test_avl_t i;
i.name = 'I';
i.value = 7;
AVL_init(&i.node);

test_avl_t j;
j.name = 'J';
```

```
    j.value = 11;
    AVL_init(&j.node);

    test_avl_t k;
    k.name = 'K';
    k.value = 15;
    AVL_init(&k.node);

    test_avl_t l;
    l.name = 'L';
    l.value = 8;
    AVL_init(&l.node);
```

印出樹函數

```
void printf_AVL_tree(avl_node_t * root, int(*get_data)(avl_node_t * queue_member),
                     int ascii);
```

使用這個函數可以印出AVL tree ，在之後的範例會用到

插入節點函數

```
void AVL_insert(avl_node_t * element, avl_node_t ** root,
                int(*compare)(void * element, void * in_tree_element));
```

使用這個函數可以在 AVL tree 中插入節點

```
    printf("\n--------------------------------------");
    printf("\ncase a");
    printf("\n--------------------------------------");
    AVL_insert(&a.node, &avl_root_a, compare_value);
    AVL_insert(&b.node, &avl_root_a, compare_value);
    printf_AVL_tree(avl_root_a,  get_value);

    AVL_init(&a.node);
    AVL_init(&b.node);
    avl_root_a = NULL;

    printf("\n----- case a insert 15 -----\n");
    AVL_insert(&a.node, &avl_root_a, compare_value);
    AVL_insert(&b.node, &avl_root_a, compare_value);
    AVL_insert(&k.node, &avl_root_a, compare_value);
    printf_AVL_tree(avl_root_a,  get_value);

    AVL_init(&a.node);
```

```
    AVL_init(&b.node);
    AVL_init(&k.node);
    avl_root_a = NULL;
```

終端機的輸出

```
--------------------------------------
case a
--------------------------------------
  20
  /
 /
4

----- case a insert 15 -----

  15
  / \
 /   \
4    20

----- case a insert 8 -----

   8
  / \
 /   \
4    20
```

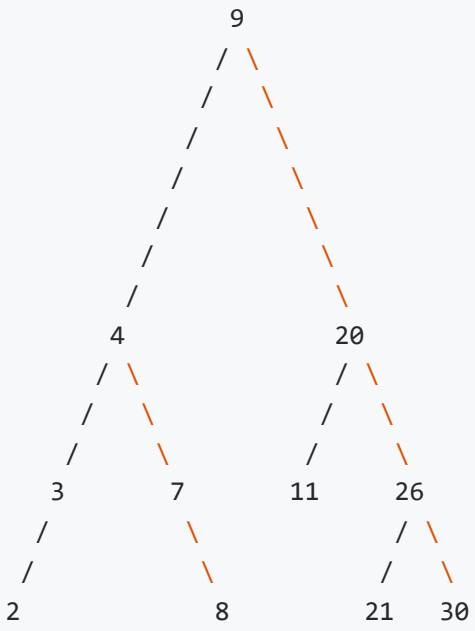
尋找節點函數

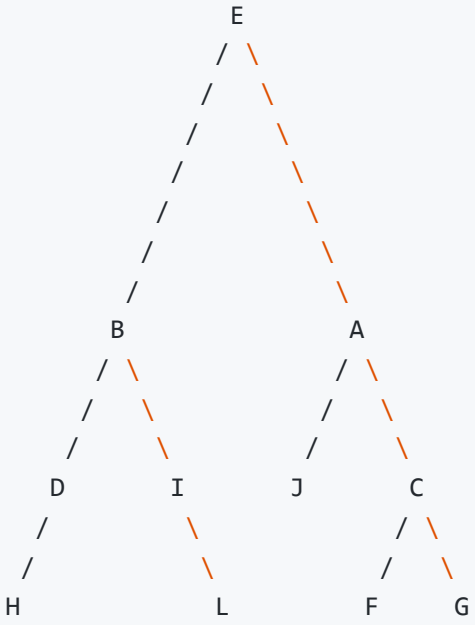
```
avl_node_t * AVL_find(int key, avl_node_t ** root,
                      int(*compare)(int key, void * in_tree_element));
```

使用這個函數可以在 AVL tree 中尋找符合key值的節點，以下是範例

```
printf_AVL_tree(avl_root_a,  get_value, 0);
printf("\n");
printf_AVL_tree(avl_root_a,  get_name, 1);
printf("\n");

avl_node_t * found = AVL_find(8, &avl_root_a, compare_key);
printf("%d\n", return_to_user_struct_pointer(test_avl_t, node, found)->value);
```

終端機的輸出






```
who's value is 8 ? : L
```

尋找最小節點函數

```
avl_node_t * AVL_find_minimum(avl_node_t * root);
```

使用這個函數可以在 AVL tree 中尋找key值最小的節點，以下是範例

```
found = AVL_find_minimum(avl_root_a);
printf("who's value is minimum ? : %c\n",
       return_to_user_struct_pointer(test_avl_t, node, found)->name);
```

終端機的輸出

```
                9
               / \
              /   \
             /     \
            /       \
           /         \
          /           \
         /             \
        4               20
       / \             / \
      /   \           /   \
     /     \         /     \
    3       7       11     26
   /         \             / \
  /           \           /   \
 2             8         21   30
```

```
                E
               / \
              /   \
             /     \
            /       \
           /         \
          /           \
         /             \
        B               A
       / \             / \
      /   \           /   \
     /     \         /     \
    D       I       J       C
   /         \             / \
  /           \           /   \
 H             L         F     G
```

```
who's value is minimum ? : H
```

尋找最大節點函數

```
avl_node_t * AVL_find_maximum(avl_node_t * root);
```

使用這個函數可以在 AVL tree 中尋找key值最大的節點，以下是範例

```
found = AVL_find_maximum(avl_root_a);
printf("who's value is maximum ? : %c\n",
```

```
        return_to_user_struct_pointer(test_avl_t, node, found)->name);
```

終端機的輸出

```
                9
               / \
              /   \
             /     \
            /       \
           /         \
          /           \
         /             \
        4               20
       / \             / \
      /   \           /   \
     /     \         /     \
    3       7       11     26
   /         \             / \
  /           \           /   \
 2             8         21   30


                E
               / \
              /   \
             /     \
            /       \
           /         \
          /           \
         /             \
        B               A
       / \             / \
      /   \           /   \
     /     \         /     \
    D       I       J       C
   /         \             / \
  /           \           /   \
 H             L         F     G

who's value is maximum ? : G
```

刪除節點函數

```
void AVL_delete(avl_node_t * delete_element, avl_node_t ** root,
                int (*compare)(void * delete_element, void * in_tree_element));
```

使用這個函數可以在 AVL tree 中刪除指定key值的節點，以下是範例

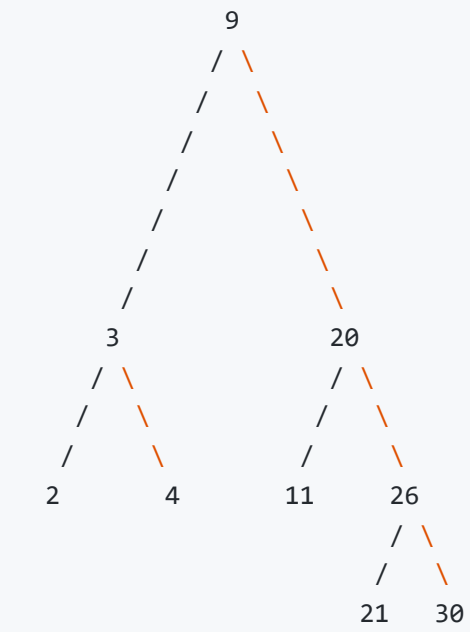
```
found = AVL_find(8, &avl_root_a, compare_key);
printf("delete %d\n", return_to_user_struct_pointer(test_avl_t, node, found)->value);
AVL_delete(found, &avl_root_a, compare_value);

printf_AVL_tree(avl_root_a, get_value, 0);
printf("\n");
printf("--------------------------------\n");

found = AVL_find(7, &avl_root_a, compare_key);
printf("delete %d\n", return_to_user_struct_pointer(test_avl_t, node, found)->value);
AVL_delete(found, &avl_root_a, compare_value);

printf_AVL_tree(avl_root_a, get_value, 0);
printf("\n");
printf("--------------------------------\n");

found = AVL_find(11, &avl_root_a, compare_key);
printf("delete %d\n", return_to_user_struct_pointer(test_avl_t, node, found)->value);
AVL_delete(found, &avl_root_a, compare_value);

printf_AVL_tree(avl_root_a, get_value, 0);
printf("\n");
printf("--------------------------------\n");

found = AVL_find(26, &avl_root_a, compare_key);
printf("delete %d\n", return_to_user_struct_pointer(test_avl_t, node, found)->value);
AVL_delete(found, &avl_root_a, compare_value);

printf_AVL_tree(avl_root_a, get_value, 0);
printf("\n");
printf("--------------------------------\n");

found = AVL_find(21, &avl_root_a, compare_key);
printf("delete %d\n", return_to_user_struct_pointer(test_avl_t, node, found)->value);
AVL_delete(found, &avl_root_a, compare_value);

printf_AVL_tree(avl_root_a, get_value, 0);
printf("\n");
printf("--------------------------------\n");

found = AVL_find(30, &avl_root_a, compare_key);
printf("delete %d\n", return_to_user_struct_pointer(test_avl_t, node, found)->value);
AVL_delete(found, &avl_root_a, compare_value);

printf_AVL_tree(avl_root_a, get_value, 0);
printf("\n");
printf("--------------------------------\n");

found = AVL_find(2, &avl_root_a, compare_key);
printf("delete %d\n", return_to_user_struct_pointer(test_avl_t, node, found)->value);
AVL_delete(found, &avl_root_a, compare_value);
```

```
printf_AVL_tree(avl_root_a, get_value, 0);
printf("\n");
printf("--------------------------------\n");

found = AVL_find(20, &avl_root_a, compare_key);
printf("delete %d\n", return_to_user_struct_pointer(test_avl_t, node, found)->value);
AVL_delete(found, &avl_root_a, compare_value);

printf_AVL_tree(avl_root_a, get_value, 0);
printf("\n");
```

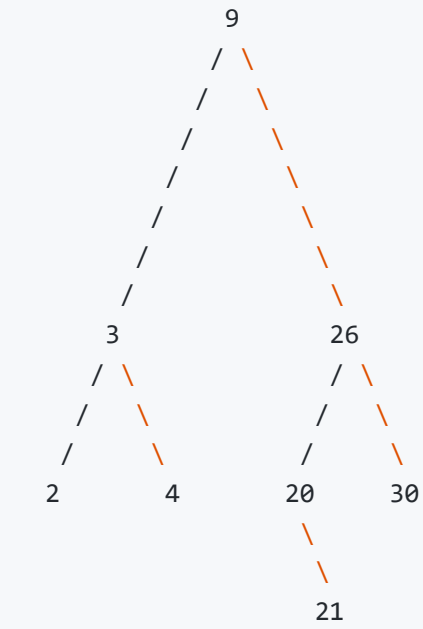
終端機的輸出

```
                9
              / \
             /   \
            /     \
           /       \
          /         \
         /           \
        /             \
       4               20
      / \             / \
     /   \           /   \
    /     \         /     \
   3       7       11     26
  /         \             / \
 /           \           /   \
2             8         21   30

--------------------------------
delete 8

                9
              / \
             /   \
            /     \
           /       \
          /         \
         /           \
        /             \
       4               20
      / \             / \
     /   \           /   \
    /     \         /     \
   3       7       11     26
  /                       / \
 /                       /   \
2                       21   30

--------------------------------
```

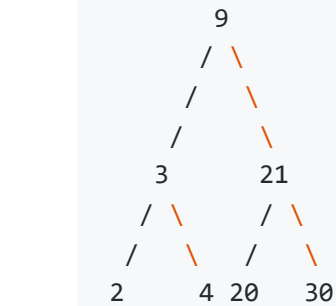
```
delete 7
```



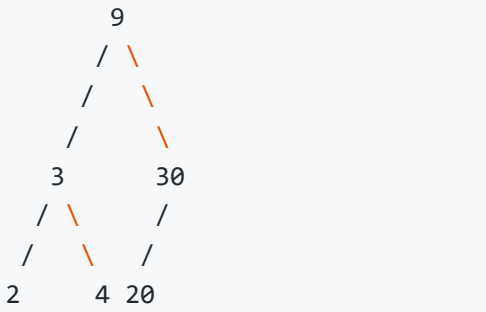

```
--------------------------------
delete 11
```




```
--------------------------------
delete 26
```

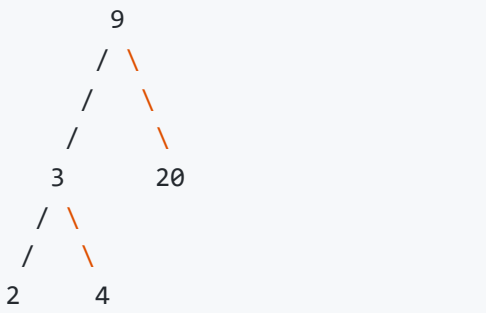



```
--------------------------------
delete 21
```



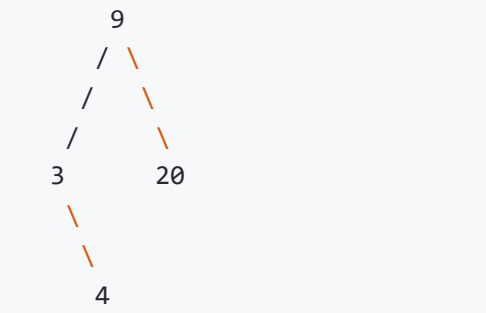
---

delete 30



---

delete 2



---

delete 20