F74109016_hw25

程式碼

```
1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include <stdbool.h>
     #define DEGREE 3
    #define ORDER (DEGREE * 2)
    typedef struct node* node_t;
 8 = struct node{
         int keys[ORDER - 1];
 9
10
         node_t childs[ORDER];
11
         int n;
         bool leaf:
12
13 L };
14
15
     node_t node_new(bool);
16
     void tree_delete(node_t);
17
     node_t search(node_t, int);
18
     int search_index(node_t, int, int*);
19
20
     void node_insert(int);
     void insert_nonfull(node_t, int);
21
22
     void split_child(node_t, int, node_t);
23
24
     bool node_remove(node_t, int);
25
     void remove_nonleaf(node_t, int);
26
     void merge(node_t, int);
27
     void borrow_prev(node_t, int);
28
     void borrow_next(node_t, int);
29
30
    node_t root = NULL;
32 - node_t node_new(bool leaf){
33
         node_t node = malloc(sizeof(struct node));
34
         int i;
35
         for(i = 0; i < ORDER; i++)
           node->childs[i] = NULL;
37
        node->n = 0;
38
        node->leaf = leaf;
39
40
        return node;
41 - }
42
43 poid tree_delete(node_t node){
44
        int i;
45
         if (node == NULL)
46
            return;
48
         if(!node->leaf)
            for (i = 0; i <= node->n; i++)
49
               tree_delete(node->childs[i]);
50
51
         free(node);
52
53 L }
54
55 ☐ node_t search(node_t node, int key){
56
         int i:
57
         for (i = 0; i < node->n && node->keys[i] < key; i++);</pre>
59
         if (node->keys[i] == key)
60
            return node;
61
62
         if (node->leaf)
63
            return NULL;
64
         return search(node->childs[i], key);
65
66 L }
```

```
68 ☐ int search_index(node_t node, int index, int* count){
69 ☐ if (!node->leaf){
70
              int temp = search_index(node->childs[0], index, count);
71
              if (*count == -1)
72
                  return temp;
              int i;
73
74日
75日
              for (i = 0; i < node\rightarrow n; i++){
                  if (index == *count){
                       *count = -1;
76
77
                      return node->keys[i];
78
79
                  (*count)++;
80
                  int temp = search_index(node->childs[i + 1], index, count);
81
82
                  if (*count == -1)
83
                      return temp;
84
85
86日
          else{
              if (node->n > index - *count){
88
                  index -= *count;
89
                  *count = -1;
90
                  return node->keys[index];
91
92
93
                  *count += node->n;
94
95
          return 0;
98 void node_insert(int key){
99 🖨
          if (root == NULL){
               root = node_new(true);
100
101
               root->keys[0] = key;
102
              root->n = 1;
103
              return;
104
105 🖨
          if (root->n == ORDER - 1){
              node_t node = node_new(false);
106
107
              node->childs[0] = root;
108
109
              split_child(node, 0, root);
110
111
              int i = 0:
              if (node->keys[0] < key)</pre>
112
113
                   i++;
114
               insert_nonfull(node->childs[i], key);
115
116
              root = node:
117
118
          else
119
              insert_nonfull(root, key);
122 ☐ void insert_nonfull(node_t node, int key){
123
          int i;
124 🖨
          if (node->leaf) {
125
               for (i = node->n - 1; i >= 0 && node->keys[i] > key; i--)
                  node->keys[i + 1] = node->keys[i];
126
127
128
              node->keys[i + 1] = key;
129
              node->n++;
130
131
          else{
132
              for (i = node->n - 1; i >= 0 && node->keys[i] > key; i--);
133
134 🖨
              if (node\rightarrow childs[i + 1]\rightarrow n == ORDER - 1){
135
                  split_child(node, i + 1, node->childs[i + 1]);
                  if (node->keys[i + 1] < key)
136
137
                      i++:
138
139
              insert_nonfull(node->childs[i + 1], key);
140
141 }
142
```

```
143 ☐ void split_child(node_t parent, int i, node_t child){
144
          node_t node = node_new(child->leaf);
145
          node->n = DEGREE - 1;
146
          int j;
147
          for (j = 0; j < DEGREE - 1; j++)
148
              node->keys[j] = child->keys[j + DEGREE];
149
150
          if(!child->leaf)
               for (j = 0; j < DEGREE; j++)</pre>
151
152
                  node->childs[j] = child->childs[j + DEGREE];
153
          child->n = DEGREE - 1:
154
155
156
          for (j = parent->n; j >= i + 1; j--)
157
              parent->childs[j + 1] = parent->childs[j];
158
159
          parent->childs[i + 1] = node;
160
          for (j = parent->n - 1; j >= i; j--)
    parent->keys[j + 1] = parent->keys[j];
161
162
163
164
          parent->keys[i] = child->keys[DEGREE - 1];
          parent->n++;
166 L }
168 bool node_remove(node_t node, int key){
169
          int i:
          if (node == NULL)
170
              return false;
171
172
173
          int index;
174
          for (index = 0; index < node->n && node->keys[index] < key; index++);</pre>
175
176 🖨
          if (index < node->n && node->keys[index] == key){
177 白
               if (node->leaf) {
                   for (i = index + 1; i < node->n; i++)
178
179
                      node->keys[i - 1] = node->keys[i];
180
                   node->n--;
181
182
               else
                   remove_nonleaf(node, index);
183
184
185 🖨
           else if (!node->leaf){
186
               bool flag = (index == node->n) ? true : false;
187
188
               if (node->childs[index]->n < DEGREE){
189
                   if (index != 0 && node->childs[index - 1]->n >= DEGREE)
190
                       borrow_prev(node, index);
191
                   else if (index != node->n && node->childs[index + 1]->n >= DEGREE)
192
                       borrow_next(node, index);
193 🖨
                   else{
194
                       if (index != node->n)
195
                           merge(node, index);
                       else
196
197
                           merge(node, index - 1);
198
199
200
200 -
201
202
              if (flag && index > node->n)
203
                  return node_remove(node->childs[index - 1], key);
204
205
                  return node_remove(node->childs[index], key);
206
207
          else
208
              return false;
209
210
          return true:
211 L }
```

```
213 poid remove_nonleaf(node_t node, int index){
214
          int k = node->keys[index];
215
216
          if (node->childs[index]->n >= DEGREE){
              node_t temp = node->childs[index];
217
218
219
              while (!temp->leaf)
220
              temp = temp->childs[temp->n];
221
222
              node->keys[index] = temp->keys[temp->n - 1];
              node_remove(node->childs[index], temp->keys[temp->n - 1]);
223
224
225
226 🖨
          else if (node->childs[index + 1]->n >= DEGREE){
227
              node_t temp = node->childs[index + 1];
228
229
              while (!temp->leaf)
                 temp = temp->childs[0];
230
231
              node->keys[index] = temp->keys[0];
232
              node_remove(node->childs[index + 1], temp->keys[0]);
234
235
236 🖨
          else{
              merge(node, index);
237
              node_remove(node->childs[index], k);
238
239
240 L }
node_t sibling = node->childs[index + 1];
244
245
           int i:
           child->keys[DEGREE - 1] = node->keys[index];
246
247
248
          for (i = 0; i < sibling->n; i++)
249
              child->keys[i + DEGREE] = sibling->keys[i];
250
251
           if (!child->leaf)
252
               for (i = 0; i <= sibling->n; i++)
253
                   child->childs[i + DEGREE] = sibling->childs[i];
254
255
           for (i = index + 1; i < node->n; i++)
256
              node->keys[i - 1] = node->keys[i];
257
258
           for (i = index + 2; i <= node->n; i++)
259
               node->childs[i - 1] = node->childs[i];
260
261
           child->n += sibling->n + 1;
262
263
264
           free(sibling);
265 L }
267 

void borrow_prev(node_t node, int index){
268
          node_t child = node->childs[index];
269
          node_t sibling = node->childs[index - 1];
270
          int i;
          for (i = child->n - 1; i >= 0; i--)
    child->keys[i + 1] = child->keys[i];
271
272
273
274
          if (!child->leaf)
275
              for (i = child->n; i >= 0; i--)
276
                 child->childs[i + 1] = child->childs[i];
277
278
         child->keys[0] = node->keys[index - 1];
279
280
          if (!child->leaf)
             child->childs[0] = sibling->childs[sibling->n];
281
282
283
          node->keys[index - 1] = sibling->keys[sibling->n - 1];
284
285
          child->n++;
286
          sibling->n--;
287 L
```

```
289 ☐ void borrow_next(node_t node, int index){
290
           node_t child = node->childs[index];
291
           node_t sibling = node->childs[index + 1];
292
           int i:
293
           child->keys[child->n] = node->keys[index];
294
295
296
               child->childs[child->n + 1] = sibling->childs[0];
297
298
           node->keys[index] = sibling->keys[0];
299
           for (i = 1; i < sibling->n; i++)
300
301
              sibling->keys[i - 1] = sibling->keys[i];
302
           if (!sibling->leaf)
303
304
                for (i = 1; i <= sibling->n; i++)
305
                 sibling->childs[i - 1] = sibling->childs[i];
306
           child->n++;
307
308
           sibling->n--;
309 L }
310 ☐ int main(){
311
           char s[8];
           char stos,
int n, x,i;
scanf("%d", &n);
for (i = 0; i < n; i++){
    scanf("%s", s);
    scanf("%d", &x);
    if (stromn(s, "add"))</pre>
312
313
314 🖨
315
316
317 🗀
                if (strcmp(s, "add") == 0){
318
                     node_insert(x);
319
                     printf("add(%d) = ok\n", x);
320
321 🖨
                else if (strcmp(s, "get") == 0){
                     node_t temp = search(root, x);
322
                     if (temp == NULL)
323
324
                         printf("get(%d) = not found\n", x);
325
                     else
                         printf("get(%d) = %d\n", x, x);
326
327
328 🖨
                else if (strcmp(s, "getk") == 0){
329
                     if (x <= 0)
330
                         printf("getk(%d) = not found\n", x);
331 🖨
                     else{
332
                         int count = 1;
333
                          int* count_ptr = &count;
334
                          int key = search_index(root, x, count_ptr);
335
                          if(*count ptr == -1)
336
                             printf("getk(%d) = %d\n", x, key);
                          else
337
338
                              printf("getk(%d) = not found\n", x);
339
340
341
342 -
                else if (strcmp(s, "remove") == 0){
343
                    if(node_remove(root, x))
                       printf("remove(%d) = %d\n", x, x);
344
345
                        printf("remove(%d) = not found\n", x);
346
347
348
               else if (strcmp(s, "removek") == 0){
349
                    if (x <= 0)
350
                        printf("removek(%d) = not found\n", x);
351 🖨
352
                       int count = 1;
353
                        int* count_ptr = &count;
                        int key = search_index(root, x, count_ptr);
if (*count_ptr == -1 && node_remove(root, key))
    printf("removek(%d) = %d\n", x, key);
354
355
356
357
                        else
358
                           printf("removek(%d) = not found\n", x);
359
360
361 白
                else{
                   printf("Wrong Instruction\n");
362
363
364
365
           tree_delete(root);
366
           return 0:
```

搜尋 getk

找到某個 index 的值為多少。Worst case 發生在那個 index 的是整個數最後的 index,需要走訪每個節點,計算所有節點的數量後才能找到。時間複雜度是 O(n)。

搜尋函數

```
int search_index(node_t node, int index, int* count){
   if (!node->leaf){
                                                       如果有小孩的話,就找小
      int temp = search index(node->childs[0], index, count);
                                                       孩,從第一個小孩開始找
      if (*count == -1)
          return temp;
      int i;
      for (i = 0; i < node->n; i++){
          if (index == *count){
             *count = -1;
                                           找到了,就回傳
             return node->keys[i];
          (*count)++:
                                                               從找下個小孩
          int temp = search_index(node->childs[i + 1], index, count);
          if (*count == -1)
                                                 從如果在下個小孩有找到,就回傳
             return temp;
   else{
      if (node->n > index - *count){
                                     如果是葉節點,就直接算該節點的數量加上目前的
          index -= *count;
                                     數量是否大於 index,如果大於,就找到了回傳那
          *count = -1;
                                     個 index 的值
          return node->keys[index];
      else
          *count += node->n;
   return 0:
                                     如果該節點的數量加上目前的數量是否大於
}
```

index,如果小於,將目前的數量加上該節點的是量

删除 removek

要先找到某個 index 的值為多少,其實時間複雜度為上面函數的時間複雜度,最壞的情況為要刪除的 index 為整棵樹的最後一個 index。時間複雜度為 O(n)。刪除函數:

```
168 ☐ bool node_remove(node_t node, int key){
                                                                                    如果找到節點, node
             169
                      int i:
             170
                      if (node == NULL)
                                                                                    是葉節點,就直接將
             171
                          return false;
                                                                                    key 的值往前移
             172
             173
                      int index:
             174
                      for (index = 0; index < node->n && node->keys[index] < key; index++);</pre>
             175
             176 E
                      if (index < node->n && node->keys[index] == key){
             177 🗀
                          if (node->leaf) {
             178
                             for (i = index + 1; i < node->n; i++)
             179
                                 node->keys[i - 1] = node->keys[i];
             180
                                                                           Index 等於 node 值的數量或
             181
             182
                          else
                                                                           是在這個 node 沒有找到值
             183
                             remove_nonleaf(node, index);
             184
             185 🗀
                      else if (!node->leaf){
小孩不滿
                                                                                   如果前面可以合併就
             186
                          bool flag = (index == node->n) ? true : false;
             187
足 degree
                                                                                   跟前面的節點合併
             188 🖨
                          if (node->childs[index]->n < DEGREE){</pre>
的規則
             189
                              if (index != 0 && node->childs[index - 1]->n >= DEGREE)
             190
                                 borrow_prev(node, index);
             191
                             else if (index != node->n && node->childs[index + 1]->n
                                                                                   前面不行後面可以合
             192
                                 borrow_next(node, index);
             193 🗀
                             else{
                                                                                   併就跟後面的節點合
             194
                                 if (index != node->n)
                                                                                   併
             195
                                     merge(node, index);
             196
                                 else
             197
                                     merge(node, index - 1);
                                                              Sibling 都不行合併就
             198
                                                              跟 parent 合併。
             199
              200
              201
                           if (flag && index > node->n)
              202
                               return node_remove(node->childs[index - 1], key);
                                                                                找要 remove 的節
              203
                           else
              204
                               return node_remove(node->childs[index], key);
                                                                                點是在哪裡。
              205
              206
                        else
              207
                           return false;
              208
              209
                        return true;
              210 L }
```

```
212 poid remove_nonleaf(node_t node, int index){
          int k = node->keys[index];
214
215 🖨
          if (node->childs[index]->n >= DEGREE){
              node_t temp = node->childs[index];
216
217
218
              while (!temp->leaf)
219
                  temp = temp->childs[temp->n];
220
221
              node->keys[index] = temp->keys[temp->n - 1];
222
              node_remove(node->childs[index], temp->keys[temp->n - 1]);
223
224
           else if (node->childs[index + 1]->n >= DEGREE){
225 🗀
226
              node_t temp = node->childs[index + 1];
227
228
              while (!temp->leaf)
229
                  temp = temp->childs[0];
230
231
              node->keys[index] = temp->keys[0];
              node_remove(node->childs[index + 1], temp->keys[0]);
232
233
234
235 🖨
236
              merge(node, index);
237
              node remove(node->childs[index], k);
238
239 L }
```

非葉節點刪除某個 key 值,就要移動 child

將 node 最後一個值(最大值)刪掉,因為他已經到parent 節點去了

將 node 最後第一個值(最小值)刪掉,因為他已經到parent 節點去了

如果都不大於 degree 就直接合併了。

刪除動作的 Algorithm:

- 1.首先查詢 Btree 中需要刪除的元素。上面的搜尋函數。找到就到第二步驟。
- 2.刪除後,判斷該元素是否有左右小孩節點,如果有就將左小孩最大的值或右 小孩小的值拉到現在被刪除的元素中。

如果某節點中的元素小於 degree/2-1,則看看 sibling 是否滿。

如果不滿就跟 sibling 合併。

如果滿就跟父節點借一個元素來放入被刪除的元素李。

Remove 的最壞情況就是一直不符合要求,需要跟上面的不斷合併。如例子中刪 5 的情況。其時間複雜度為樹的高度。

Btree 樹的高度:

根結點至少兩個 child,第二層有一兩個節點。

第三層至少有 2*((節點最大數量/2)取天花板)

第四層至少有2*((節點最大數量/2)取天花板)2

第 k 層至少有 2*((節點最大數量/2)取天花板)^{k-2}

如果 Btree 有 N 個 key 值,則數的 leaf 節點可以為 N+1 個,leaf 都在 k 層 N+1>=2*((節點最大數量/2)取天花板) $^{k-2}$

→k<=log(((節點最大數量/2)取天花板)*(N+1)/2)+1

此為 remove 動作 worst case 的時間複雜度。

log(((節點最大數量/2)取天花板)*(N+1)/2)+1 < O(N) (找 index 的值)

所以 remove 的 worst case 時間複雜度也為 O(N)

刪除某個 key 值要一直合併的例子(最後刪 5)

