DataStructureHW5

2013011372박병욱

1. Assignment

Binary Search Tree를 구현한다.

1. Result

$./hw5



$./hw5 input2.txt



1. 소스코드
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <memory.h>
5. #include <string.h>
6. #include <stdio.h>
7. typedef struct TreeNode {
8. int key;
9. struct TreeNode \*left, \*right;
10. } TreeNode;
11. void display(TreeNode \*p)
12. {
13. if (p != NULL) {
14. printf("(");
15. display(p->left);
16. printf("%d", p->key);
17. display(p->right);
18. printf(")");
19. }
20. }
21. //루트부터 내려가 재귀적으로key를 가진 트리 노드를 찾아 반환한다.
22. TreeNode \*search(TreeNode \*node, int key)
23. {
24. while(node != NULL){
25. if( key == node->key ) return node;
26. else if( key < node->key )
27. return search(node->left, key);
28. else
29. return search(node->right, key);
30. }
31. return NULL; // 탐색에 실패했을 경우 NULL 반환
32. }
33. //루트부터 내려가 재귀적으로key를 가진 트리 노드의 부모를 찾아 반환한다.
34. //key값이 루트에 있거나, 값이 없으면 NULL을 반환한다.
35. TreeNode \*searchParent(TreeNode \*node, int key)
36. {
37. while(node != NULL){
38. if(node->left != NULL){
39. if(key == node->left->key) return node;
40. }
41. if(node->right != NULL){
42. if(key == node->right->key) return node;
43. }
44. if( key < node->key )
45. return searchParent(node->left, key);
46. else
47. return searchParent(node->right, key);
48. }
49. return NULL; // 탐색에 실패했을 경우 NULL 반환
50. }
51. void insert\_node(TreeNode \*\*root, int key)
52. {
53. TreeNode \*p, \*q;
54. TreeNode \*n;
55. p = \*root;
56. q = NULL;
57. // 인자로 받은 key 탐색 수행
58. if( search(p,key) != NULL){
59. printf("Alredy exist\n");
60. return;
61. }
62. //p가 가 들어갈 자리를 반복해서 찾고 내려가기 전에 q에 p를 넣어 부모를 저장한다
63. while(p != NULL ){
64. if( key < p->key){
65. q = p;
66. p = p->left;
67. }else if(key > p->key){
68. q = p;
69. p = p->right;
70. }
71. }
72. if(p == NULL)
73. n = (TreeNode\*)malloc(sizeof(TreeNode));
74. n->key = key;
75. n->left = NULL;
76. n->right = NULL;
77. //처음 root가 들어왔을경우 p가 NULL이었다가 메모리를 받으므로, q는 NULL이다
78. //이때 바로 루트에 n을 넣어준다.
79. if(q == NULL){
80. \*root = n;
81. return;
82. }
83. if(n->key < q->key){
84. q->left = n;
85. }else {
86. q->right = n;
87. }
88. }
89. //루트로 부터 key를 가진 tmp를 찾고,
90. //tmp의 부모를 pTmp에 넣는다.
91. //1. tmp가 left,right 둘다 없을 경우 바로 free하고, 부모에서 tmp쪽 방향에 NULL을 넣는다.
92. //2. tmp에 right 가 있을경우(right만 있거나 left, right둘다 있을경우)
93. // right에서 제일 왼쪽에 있는 노드를 찾아 tmp의 key값과 교환하고,
94. // free하고, 부모에서 left쪽 방향에 NULL을 넣는다.
95. //3. tmp에 right가 없고, left만 있을경우 tmp의 key값과 교환하고,
96. // left에서 제일 오른쪽에 있는 노드를 찾아 free하고, 부모에서 right쪽 방향에 NULL을 넣는다.
97. // root는 free를 해도 NULL이 되지 않으므로, 삭제할 수 없다.
98. void delete\_node(TreeNode \*node, int key)
99. {
100. TreeNode \*tmp, \*pTmp;
101. TreeNode \*iter, \*pIter = NULL;
102. int isleft;
103. tmp = search(node,key);
104. if(tmp == NULL){
105. printf("The node that has key : %d isn't in the tree\n",key);
106. return ;
107. }
108. pTmp = searchParent(node,key);
109. if(pTmp == NULL){
110. printf("Root can't be eliminated\n");
111. return;
112. }
113. if(pTmp->left == tmp) isleft=1;
114. if(pTmp->right == tmp) isleft=0;
115. if(tmp->left == NULL && tmp->right == NULL){
116. free(tmp);
117. tmp = NULL;
118. if(isleft) pTmp->left = NULL;
119. if(!isleft) pTmp->right = NULL;
120. return;
121. }else{
122. if(tmp->right != NULL){
123. iter = tmp->right;
124. while(iter->left != NULL){
125. iter = iter->left;
126. }
127. tmp->key = iter->key;
128. pIter = searchParent(node,iter->key);
129. free(iter);
130. if(pIter != NULL) pIter->left=NULL;
131. return;
132. }else {
133. iter = tmp->left;
134. while(iter->right != NULL){
135. iter = iter->right;
136. }
137. tmp->key = iter->key;
138. pIter = searchParent(node,iter->key);
139. free(iter);
140. if(pIter != NULL) pIter->right=NULL;
141. return;
142. }
143. }
144. }
145. void main(int argc, char \*argv[])
146. {
147. char command;
148. int key;
149. FILE \*input;
150. TreeNode \*root = NULL;
151. if (argc == 1) input = fopen("input.txt", "r");
152. else input = fopen(argv[1], "r");
153. while (1) {
154. command = fgetc(input);
155. if (feof(input)) break;
156. switch (command) {
157. case 'i':
158. fscanf(input, "%d", &key);
159. insert\_node(&root, key);
160. break;
161. case 'd':
162. fscanf(input, "%d", &key);
163. delete\_node(root, key);
164. break;
165. case 's':
166. fscanf(input, "%d", &key);
167. if (search(root, key) != NULL)printf("key is in the tree: %d\n", key);
168. else printf("key is not in the tree: %d\n", key);
169. break;
170. case 'p':
171. display(root);
172. printf("\n");
173. break;
174. default:
175. break;
176. }
177. }
178. fclose(input);
179. }

* 소스코드 설명

주어진 given..c를 이용하여 구현하였다.

Argv[1]을 인풋파일로 인식하였고, argv가 없을 경우 input.txt를 열었다.

While문 안에서 fgetc로 input파일을 읽었다.

인풋 파일로부터 각각의 명령을 읽었고 그에 알맞은 함수를 구현하였다.

Search()는 루트부터 내려가 재귀적으로key를 가진 트리 노드를 찾아 반환한다.

searchParent()는 루트부터 내려가 재귀적으로key를 가진 트리 노드의 부모를 찾아 반환한다.

key값이 루트에 있거나, 값이 없으면 NULL을 반환한다.

Insert\_node()는 반복문을 이용하여 key값을 가진 노드가 들어갈 자리를 찾고, 새로 노드를 만들고 그 자리에 노드를 넣어주었다. Root에 key를 넣어야 하는 경우도 따로 처리하였다.

Delete\_node()는루트로 부터 key를 가진 tmp를 찾고,

tmp의 부모를 pTmp에 넣는다.

1. tmp가 left,right 둘다 없을 경우 바로 free하고, 부모에서 tmp쪽 방향에 NULL을 넣는다.

2. tmp에 right 가 있을경우(right만 있거나 left, right둘다 있을경우)

right에서 제일 왼쪽에 있는 노드를 찾아 tmp의 key값과 교환하고,

free하고, 부모에서 left쪽 방향에 NULL을 넣는다.

3. tmp에 right가 없고, left만 있을경우 tmp의 key값과 교환하고,

left에서 제일 오른쪽에 있는 노드를 찾아 free하고, 부모에서 right쪽 방향에 NULL을 넣는다.

root는 free를 해도 NULL이 되지 않으므로, 삭제할 수 없다.