1. #include <stdio.h> -
2. #include <stdlib.h> -
3. #include "arvore.h" -
4. arvoreBI \* create(){ -
5. arvoreBI \* arv = (arvoreBI \*) malloc(sizeof(arvoreBI)); 1
6. if(arv != NULL){ 1
7. arv->raiz = NULL; 1
8. arv->tam = 0; 1
9. arv->nivel = 0; 1
10. } -
11. return arv; 1
12. } -
13. No \* createRoot(arvoreBI \*arv,char valor){ -
14. arv->raiz = (No\*) malloc(sizeof(No)); 1
15. if(arv->raiz != NULL){ 1
16. arv->raiz->filhoEsquerda = NULL; 1
17. arv->raiz->filhoDireita = NULL; 1
18. arv->raiz->valor = valor; 1
19. arv->tam ++; 1
20. } -
21. return arv->raiz; 1
22. } -
23. No \* add(arvoreBI \* arv,int lado,No \*raiz,char valor){ -
24. No \* aux = (No\*) malloc(sizeof(No)); 1
25. if(aux != NULL){ 1
26. aux->filhoEsquerda = NULL; 1
27. aux->filhoDireita = NULL; 1
28. aux->valor = valor; 1
29. if(lado == L\_ESQUERDO){ 1
30. raiz->filhoEsquerda = aux; 1
31. arv->tam ++; 1
32. } -
33. else{ -
34. raiz->filhoDireita = aux; 1
35. arv->tam ++; 1
36. } -
37. } -
38. } -
39. void clear(arvoreBI \* arv){ -
40. arv->raiz->filhoDireita = NULL; 1
41. arv->raiz->filhoDireita = NULL; 1
42. arv->raiz = NULL; 1
43. arv->tam = 0; 1
44. printf("Limpo com sucesso!\n\n"); 1
45. } -
46. int isEmpty(arvoreBI \* arv){ -
47. if(arv->raiz == NULL){ 1
48. return 0; 1
49. } -
50. else{ 1
51. return -1; 1
52. } -
53. } -
54. int size(arvoreBI \* arv){ -
55. return arv->tam; 1
56. } -
57. int find\_aux(arvoreBI \* arv,No \*raiz,char valor){ -
58. No \* aux= (No\*) malloc(sizeof(No)); n
59. aux->valor = valor; n
60. arv->nivel += 1; n O(n)
61. int aux1 = 0; n
62. if (raiz == NULL) { n
63. return -1; n
64. } -
65. else{ n
66. if(raiz->valor == 'e'){ n
67. arv->nivel = arv->nivel/2; n
68. } -
69. if(raiz->valor == 'c'){ n
70. arv->nivel = arv->nivel/2; n
71. arv->nivel --; n
72. } -
73. if(raiz->valor == 'f'){ n
74. } -
75. if(raiz->valor == 'g'){ n
76. arv->nivel = arv->nivel/2; n
77. } -
78. if(raiz->valor == aux->valor){ n
79. printf("find: %d\n\n",arv->nivel); n
80. } -
81. find\_aux(arv,raiz->filhoEsquerda,valor); n
82. find\_aux(arv,raiz->filhoDireita,valor); n
83. } -
84. } -
85. void find(arvoreBI \*arv,char valor) { -
86. find\_aux(arv,arv->raiz,valor); 1
87. } -
88. void printAllPre\_aux(No \*raiz) { -
89. if (raiz != NULL) { n
90. printf("%c\n", raiz->valor); n
91. printAllPre\_aux(raiz->filhoEsquerda); n O(n)
92. printAllPre\_aux(raiz->filhoDireita); n
93. } -
94. } -
95. void printAllPre(arvoreBI \*arv) { -
96. printAllPre\_aux(arv->raiz); 1
97. } -
98. void printAllIn\_aux(No \*raiz) { -
99. if (raiz != NULL) { n
100. printAllIn\_aux(raiz->filhoEsquerda); n
101. printf("%c\n", raiz->valor); n O(n)
102. printAllIn\_aux(raiz->filhoDireita); n
103. } -
104. } -
105. void printAllIn(arvoreBI \*arv) { -
106. printAllIn\_aux(arv->raiz); 1
107. } -
108. void printAllPost\_aux(No \*raiz) { -
109. if (raiz != NULL) { n
110. printAllPost\_aux(raiz->filhoEsquerda); n
111. printAllPost\_aux(raiz->filhoDireita); n O(n)
112. printf("%c\n", raiz->valor); n
113. } -
114. } -
115. void printAllPost(arvoreBI \*arv) { -
116. printAllPost\_aux(arv->raiz); 1
117. } -

Essa implementação tem ordem n.