PS4

**1- Uninformed state-space search (6pts)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | 6 | 5 | 8 |  |  |  |  |  |  |
|  |  |  |  |  |  | 7 | \_ | 1 |  |  |  |  |  |  |
|  |  |  |  |  |  | 3 | 4 | 2 |  |  |  |  |  |  |
| 2 |  |  |  | 3 |  |  |  | 4 |  |  |  | 5 |  |  |
| 6 | 5 | 8 |  | 6 | 5 | 8 |  | 6 | \_ | 8 |  | 6 | 5 | 8 |
| \_ | 7 | 1 |  | 7 | 1 | \_ |  | 7 | 5 | 1 |  | 7 | 4 | 1 |
| 3 | 4 | 2 |  | 3 | 4 | 2 |  | 3 | 4 | 2 |  | 3 | \_ | 4 |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 | 5 | 8 |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 | \_ | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 4 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |

**a. Breadth first search**

**b. Depth first search**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | 6 | 5 | 8 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | 7 | \_ | 1 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | 3 | 4 | 2 |  |  |  |  |  |  |
|  |  |  |  | 2 |  |  |  | 6 |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | 6 | 5 | 8 |  | 6 | 5 | 8 |  |  |  |  |  |  |  |  |
|  |  |  |  | \_ | 7 | 1 |  | 7 | 1 | \_ |  |  |  |  |  |  |  |  |
|  |  |  |  | 3 | 4 | 2 |  | 3 | 4 | 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  | 4 |  |  |  | 5 |  |  |  |  |  |  |  |  |  |  |
| 6 | 5 | 8 |  | \_ | 5 | 8 |  | 6 | 5 | 8 |  |  |  |  |  |  |  |  |
| 7 | \_ | 1 |  | 6 | 7 | 1 |  | 3 | 7 | 1 |  |  |  |  |  |  |  |  |
| 3 | 4 | 2 |  | 3 | 4 | 2 |  | \_ | 4 | 2 |  |  |  |  |  |  |  |  |

**c. Iterative deepening search**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | 6 | 5 | 8 |  |  |  |  |  |  |
|  |  |  |  |  |  | 7 | \_ | 1 |  |  |  |  |  |  |
|  |  |  |  |  |  | 3 | 4 | 2 |  |  |  |  |  |  |
| 2 |  |  |  | 3 |  |  |  | 4 |  |  |  | 5 |  |  |
| 6 | 5 | 8 |  | 6 | 5 | 8 |  | 6 | \_ | 8 |  | 6 | 5 | 8 |
| \_ | 7 | 1 |  | 7 | 1 | \_ |  | 7 | 5 | 1 |  | 7 | 4 | 1 |
| 3 | 4 | 2 |  | 3 | 4 | 2 |  | 3 | 4 | 2 |  | 3 | \_ | 4 |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 | 5 | 8 |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 | \_ | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 4 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |

**2- Determining the depth of a node (12pts)**

1. **Time complexity**

Binary tree with n nodes:

**Best case:**

it is the root

key == root.key at first recursive call.

Time complexity: O(1)

**Worst case:**

node is not in the tree.

Problem is divided by 2 at each iteration

balanced tree:

unbalanced tree

equivalent to a linked list

Time complexity: O(n)

1. **Revised depthInTree()**

**private static int depthInTree(int** key**,** **Node** root**) {**

if (key == root.key)

return 0;

// true is we should go left

// false if we should go right

Boolean goLeft = False;

if(key < root.key)

goLeft = True;

if (root.left != null && goLeft){

int depthInLeft = depthInTree(key, root.left);

if (depthInLeft != -1)

return depthInLeft + 1;

}

if (root.right != null && !goLeft) {

int depthInRight = depthInTree(key, root.right);

if (depthInRight != -1)

return depthInRight + 1;

}

return -1;

**}**

1. **Time complexity of revised**

Binary tree with n nodes:

**Best case:**

it is the root

key == root.key at first recursive call.

Time complexity: O(1)

**Worst case:**

node is not in the tree.

go through all nodes: n recursive calls

balanced tree:

unbalanced tree

where: m + p = n-1

**3- Tree traversal puzzle (10pts)**

pre-order: node, left, right

post-order: left, right, n

in-order: left, node, right

1. **inorder: SKBPCJRDME, preorder: PSBKRJCMDE**
2. **postorder: IBGOZKHNSL, preorder: LOGIBSHKZN**

**4- Huffman encoding (7pts)**

|  |  |
| --- | --- |
| **Ch** | **Fq** |
| F | 8 |
| C | 14 |
| O | 17 |
| I | 20 |
| E | 40 |

1. **Huffman tree**
2. **Encoding of a string**

From the tree, we get the following encoding:

**O = 00**

**F = 110**

**I = 01**

**C = 111**

**E = 10**

Therefore: **office = 00 110 110 01 111 10**

**5- Binary search trees (10pts)**

1. **preorder, at right**

28, 53, 62, 80

1. **post order**

28, 23, 35, 48, 57, 80, 62, 53, 44

1. **insert 25, insert 51**
2. **delete 53, delete 35**
3. **balanced tree**

The original tree **is not** balanced because:

The right subtree of 35 has a 0 height.

The left subtree of 35 has a 2 height.

They differ by more than 1. Therefore, the tree is not balanced.

**6- 2-3 trees and B-Trees (10pts)**

keys: A, D, G, B, F, C, H, I, E, J

1. **empty 2-3 tree**

**1- insert A**

**2- insert D**

**3- insert G**

**split**

**4- insert B**

**5- insert F**

**6- insert C**

**split**

**7- insert H**

**split**

**split**

**8- insert I**

**9- insert E**

**10- insert J**

**split**

1. **empty B-tree of order 2**

**1- insert A**

**2- insert D**

**3- insert G**

**4- insert B**

**5- insert F**

**split**

**6- insert C**

**7- insert H**

**8- insert I**

**9- insert E**

**split**

**10- insert J**