Lab06-Heaps and BST

VE281 - Data Structures and Algorithms, Xiaofeng Gao, TA: Li Ma, Autumn 2019

∗ Please upload your assignment to website. Contact webmaster for any questions.

∗ Name:Wu Jiayao   Student ID:517370910257   Email: jiayaowu1999@sjtu.edu.cn

1. **D-ary Heap.** D-ary heap is similar to binary heapbut (with one possible exception) each non-leaf node of d-ary heap has d children, not just 2 children.

   (a) How to represent a d-ary heap in an array?

   (b) What is the height of the d-ary heap with n elements? Please use n and d to show.

   (c) Please give the implementation of insertion on the min heap of d-ary heap, and show the time complexity with n and d.

```
// Input: an integer k
// Output: null
void enqueue(int k)
{
    // TODO;
}
```

Solution.

(a) The root is the first element in the array. The i^{th} child of node x in the tree is the $(x \cdot d + i)^{th}$ element in the array. The parent of node x is $\lfloor (x-1)/d \rfloor^{th}$ element in the array.

(b) Suppose the height of the heap is h. The maximum number of nodes at level $k (0 \le k \le h)$ is d^k . Hence, the number of nodes n at height h meets that

$$\sum_{k=0}^{h-1} d^k < n \le \sum_{k=0}^{h} d^k$$

$$\frac{d^h - 1}{d-1} < n \le \frac{d^{h+1}-1}{d-1}$$

The height of the d-ary heap with n elements is

$$h = \lceil \log_d (nd - n + 1) - 1 \rceil$$

(c) Since the worst case happens i reaches 1 or 0, the total loop times is $\mathcal{O}(\log_d n)$,which is the time complexity.

```
// Input: an integer k
// Output: null
void enqueue(int k)
{
    // TODO;
    array[n++]=k;
    int i = n;
    while(i > 0 && array[i] < array[(i-1)/d] )
    {
```

```
        int tmp = array[i];
        array[i] = array[(i-1)/d];
        array[(i-1)/d] = tmp;
        i = (i-1)/d;
    }
}
```

□

2. **Median Maintenance.** Input a sequence of numbers $x_1, x_2 ..., x_n$, one-by-one. At each time step i, output the median of $x_1, x_2 ..., x_i$. How to do this with $O(\log i)$ time at each step i? Show the implementation.

Solution.

```
void get_median()
{

    priority_queue<double, std::vector<double>, std::less<double>> max_heap;
    priority_queue<double, std::vector<double>, std::greater<double>> min_heap;
    int counter = 0;
    while(true)
    {
        string str;
        cin >> str;
        if(str == "exit")
        {
            break;
        }
        stringstream ss;
        ss << str;
        double p = 0;
        ss >> p;
        double median = 0;
        //Suppose max heap >= min heap in numbers
        if (counter == 0)
        {
            max_heap.push(p);
            median = p;
        }
        else if (counter % 2 == 0)
        {
            median =  (max_heap.top() + min_heap.top()) / 2;
            max_heap.pop();
            min_heap.pop();
        }
        else
        {
            if (p >= max_heap.top())
```

```
            {
                min_heap.push(p);
            }
            else
            {
                double tmp = max_heap.top();
                max_heap.pop();
                min_heap.push(tmp);
                max_heap.push(p);
            }
            median = max_heap.top();
        }
        cout << median << "\n";
        counter++;
    }
}
```

□

3. **BST**. Two elements of a binary search tree are swapped by mistake. Recover the tree without changing its structure. Implement with a constant space.

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
void recoverTree(TreeNode *root)
{
    // TODO;
}
```

Solution.

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
TreeNode* first = NULL;
TreeNode* second = NULL;
TreeNode* prev = NULL;
void traverse(TreeNode* root)
{
```

```
    if(root == NULL)
    {
        return;
    }
    traverse(root->left);
    if(prev != NULL && root->val < prev->val && !first)
    {
        first = prev;
    }
    if(prev != NULL && root->val < prev->val && first)
    {
        second = root;
    }
    prev = root;
    traverse(root->right);

}

void recoverTree(TreeNode *root)
{
    traverse(root);
    if (first && second)
    {
        int temp = first->val;
        first->val = second->val;
        second->val = temp;
    }

}
```

□

4. **BST**. Input an integer array, then determine whether the array is the result of the post-order traversal of a binary search tree. If yes, return Yes; otherwise, return No. Suppose that any two numbers of the input array are different from each other. Show the implementation.

```
// Input: an integer array
// Output: yes or no
bool verifySquenceOfBST(vector<int> sequence)
{
    // TODO;
}
```

Solution.
Input of the function **root** is added only to specified the tree to traverse.

```
// Input: an integer array
// Output: yes or no
bool verifySquenceOfBST(vector<int> squence)
{
```

```
    stack<int> s;
    int root = INT32_MAX;
    for (auto it = squence.rbegin(); it != squence.rend();it++)
    {
        if(*it > root)
        {
            return false;
        }
        while(!s.empty() && *it < s.top())
        {
            root = s.top();
            s.pop()
        }
        s.push(*it);
    }
    return true;
}
```

□