Lab06-Heaps and BST

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

* Name: Jin Zhejian Student ID: 517370910167 Email: jinzhejian@outlook.com

1. **D-ary Heap.** D-ary heap is similar to binary heap but (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.

   Solution.

   (a) A d-ary heap can be represented in an array similarly to the binary heap. Suppose the array starts with index 1. The indexes of its parent and children can be calculated as:

   1) Parent of node i: $\lceil \frac{i-1}{d} \rceil$.
   2) The j th children of node i: $d(i-1)+j+1$ where j = 1,2,...,d.

   If the array starts with the index 0, the indexes can be calculated as:

   1) Parent of node i: $\lfloor \frac{i-1}{d} \rfloor$.
   2) The j th children of node i: $d \cdot i + j + 1$ where j = 0,1,...,d-1.

   (b) We have:

$$1 + d + d^2 + ... + d^{h-1} < n \leq 1 + d + d^2 + ... + d^h$$
$$\Rightarrow d^h < (d-1)n + 1 \leq d^{h+1}$$

   So, $h = \lceil \log_d((d-1)n+1) - 1 \rceil$

   (c)

```
1         // Input: an integer k
2         // Output: null
3         void enqueue(int k)
4         {
5         n = n + 1;
6         A[n] = k;
7         //percolateUp
8         int id = n;
9         while(id > 0 && A[id] < A[(i-1)/d]){
10            swap(A[id], A[(i-1)/d]);
11            id = (i-1)/d;
12            }
13        }
```

   For the percolateUp part, in the worst case, the **while loop** loops h times, where h is the height of the heap. So, it runs in $O(\lceil \log_d((d-1)n+1) - 1 \rceil) = O(\log_d(dn)) = O(1 + \log_d(n)) = O(\log_d(n))$ time. Since inserting the integer and adding the size only need constant time, the time complexity for this enqueue part is$O(\log_d(n))$.

□

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. The algorithm for finding the median with $O(\log i)$ is shown as follows:

```
1     if the maxHeap is empty {
2         insert a[i] into maxHeap
3         median = the max value in the maxHeap
4     }
5     Else If  maxHeap is not empty & n (before insertion) is even{
6        If new item <= min(minHeap){ insert it into maxHeap}
7         Else (new item > min(minHeap)){
8             first extract min value from minHeap, then insert that
                  value in maxHeap, and finally insert newitem into
                 minHeap
9             }
10        median = the max value in the maxHeap
11     }
12    Else  If n (before insertion) is odd{
13        If new item >= max(maxHeap) {insert it into minHeap}
14        Else (new item < max(maxHeap)){
15            first extract max value from maxHeap, then insert that
                  value in minHeap, and finally insert new item into
                  maxHeap
16            }
17        median = the average value of the two root of in two heaps
18    }
```

I use the .h file done in lab05, and I choose the binary heap implemented by me in lab05 to play the role of min heap and max heap.

```
1     #include <iostream>
2     #include "priority_queue.h"
3     #include "binary_heap.h"
4     #include "unsorted_heap.h"
5     #include "fib_heap.h"
6
7     using namespace std;
8     struct compare_t
9     {
10        bool operator()(int a, int b) const
11        {
12            return a > b;
13        }
14    };
15
16    int main(int argc, char* argv[])
17    {
18        int a[] = {3, 6, 7, 9, 4, 12, 2, 5, 8,1};
19        // int a[] = {9,8,6,1};
```

```
20          int i;
21          int size = sizeof(a)/sizeof(int);
22          cout << "Input: ";
23          for(i = 0; i < size; i++)
24          cout << a[i] << " " << flush;
25          cout << endl;
26
27          priority_queue<int, compare_t> *pql = new binary_heap<int,
                compare_t>;
28          priority_queue<int> *pqs = new binary_heap<int>;
29          double median = 0;
30          for(i = 0; i < size; i++){
31              if (pql->empty()){
32                  pql->enqueue(a[i]);
33                  median = pql->get_min();
34              }
35              else if (!pql->empty() && i%2 == 0){
36                  if (a[i] <= pqs->get_min()){
37                      pql->enqueue(a[i]);
38                  }
39                  else{
40                      int tmp = pqs->dequeue_min();
41                      pql->enqueue(tmp);
42                      pqs->enqueue(a[i]);
43                  }
44                  median = pql->get_min();
45              }
46              else if(i%2 == 1) {
47                  if  (a[i] >= pql->get_min()){
48                      pqs->enqueue(a[i]);
49                  }
50              else{
51                  int tmp = pql->dequeue_min();
52                  pqs->enqueue(tmp);
53                  pql->enqueue(a[i]);
54              }
55              median =(double)(pql->get_min() + pqs->get_min()) /2;
56          }
57          cout << "The average value at step " << i+1 << " is " <<
               median << endl;
58      }
59
60      delete pql;
61      delete pqs;
62      return 0;
63      }
```

The output for the code is shown as follows:

```
    Input: 3 6 7 9 4 12 2 5 8 1
    The average value at step 1 is 3
    The average value at step 2 is 4.5
    The average value at step 3 is 6
    The average value at step 4 is 6.5
    The average value at step 5 is 6
    The average value at step 6 is 6.5
    The average value at step 7 is 6
    The average value at step 8 is 5.5
    The average value at step 9 is 6
    The average value at step 10 is 5.5

    Process finished with exit code 0
```

From the output, the previous C++ implementation code is correct.

□

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.

Solution.
My .cpp file is shown as follows:

```cpp
 struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

void recoverTree(TreeNode *root)
{
    TreeNode *first = nullptr;
    TreeNode *second = nullptr;
    TreeNode *now = root;
    TreeNode *pre = nullptr;

    //Morris Traval
    while(now){
        if (now->left){
            TreeNode *p = now->left;
            while (p->right && p->right != now){
                p = p->right;
            }
            if (!p->right){
                p->right = now;
                now = now->left;
            }
        else{
```

```
27              p->right = nullptr;
28              }
29          }
30          // find the two wrong nodes
31          if(pre && now->val < pre->val){
32              if (!first){
33                  first = pre;
34              }
35              second = now;
36          }
37          pre = now;
38          now = now->right;
39      }
40
41      //swap
42      if (first && second) {
43          int tmp = first->val;
44          first->val = second->val;
45          second->val = tmp;
46      }
47  }
```

□

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.

Solution.
My .cpp file is shown as follows:

```
// Input: an integer array
// Output: yes or no
using namespace std;

bool verifyBSThelper(vector<int> sequence, int start, int end){
    if (end <= start) return true;
    int left;
    int root = sequence[end];
    int i = end -1;
    while(i >=start && sequence[i] > root){
        i--;
    }
    left = i;
    while (i >=start && sequence[i] < root){
        i--;
    }
    if (i != start-1) return false;
    return verifyBSThelper(sequence, start, left) &&
    verifyBSThelper(sequence, left+1, end-1);
}


bool verifySequenceOfBST(vector<int> sequence){
    int length = sequence.size();
    int left;
    return verifyBSThelper(sequence, 0, length -1);
}
```

□