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:______________ Student ID:______________ Email: ____________________

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?

   Solution. Store the elements in an array in the order produced by a level order traversal. If the first element is stored at index 1. Then the children of the first element are stored in the array from A[2] to A[d+1], and the children of element in A[2] are stored in the array from A[d+2] to A[2d+1]...

   Then, we can get that:

   - A node at index $i(i \neq 1)$ has its parent at index $\lfloor \frac{i-2+d}{d} \rfloor$
   - A node at index $i(d(i-1)+1+j \leq n)$ has its j_{th} child at index $d(i-1)+1+j$.

   If the first element is stored at index 0. Then the children of the first element are stored in the array from A[1] to A[d], and the children of element in A[1] are stored in the array from A[d+1] to A[2d]...

   Then, we can get that:

   - A node at index $i(i \neq 0)$, it has its parent at index $\lfloor \frac{i-1}{d} \rfloor$
   - A at index $i(di+j \leq n)$, it has its j_{th} child at index $di+j$.

   □

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

   Solution. The height of the d-ary heap is $\Theta(log_d(n))$. For a d-ary heap with n nodes, we have:

   - The minimum height appears when every non-leaf node has d children, then we can get:
   $$1 + d + d^2 + ... + d^h = n$$
   $$h = log_d \lceil (d-1)n + 1 \rceil - 1$$
   - The maximum height appears when there is only one node at bottom level, then we can get:
   $$1 + d + d^2 + ... + d^{(h-1)} = n - 1$$
   $$h = log_d \lfloor (d-1)(n-1) + 1 \rfloor$$

   Therefore, we can get $log_d[(d-1)n+1] - 1 \leq h \leq log_d[(d-1)(n-1)+1]$, which means $h = \Theta(log_d(n))$. □

   (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)
{
    data[++size]=k;
    percolateUp(size);
}

void percolateUp(int id)
{
    while(id>0 && (data[id]<data[(id-1)/d]))
    {
        swap(data[(id-1)/d],data[id]);
        id = (id-1)/d;
    }
}
```

The time complexity of insertion 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.

```
private:
 vector<int> min;
 vector<int> max;
public:
void insert(int num)
{
 int size=min.size()+max.size();
 if((size&1)==0)
 {
  if(max.size()>0 && num<max[0])
  {
   max.push_back(num);
   push_heap(max.begin(),max.end(),less<int>());
   num=max[0];
   pop_heap(max.begin(),max.end(),less<int>());
   max.pop_back();
 }
 min.push_back(num);
 push_heap(min.begin(),min.end(),greater<int>());
}
else
{
  if(min.size()>0 && num>min[0])
  {
    min.push_back(num);
    push_heap(min.begin(),min.end(),greater<int>());
    num=min[0];
```

```
      pop_heap(min.begin(),min.end(),greater<int>());
      min.pop_back();
    }
    max.push_back(num);
    push_heap(max.begin(),max.end(),less<int>());
  }
}

double GetMedian()
{
  int size=min.size()+max.size();
  if(size<=0)
    return 0;
  if((size&1)==0)
    return (max[0]+min[0])/2.0;
  else
    return min[0];
}
```

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.

   Implement with n space and $\log n$ space.

   Ref: https://www.bilibili.com/video/av74697184?from=search&seid=2968730106680647932

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
 // n space
 private:
  int x=-1, y=-1;
  void inorder(TreeNode* node, vector<int>& nums)
  {
    if(node==nullptr) return;
    inorder(node->left, nums);
    nums.push_back(node->val);
    inorder(node->right, nums);
  }
  void findTwoSwappedNums(vector<int>& nums)
  {
    for(int i=0;i<nums.size()-1;++i)
    {
      if(nums[i]>nums[i+1])
      {
        y=nums[i+1];
```

```
            if(x==-1) x=nums[i];
            else break;
        }
    }
  }
  void recover(TreeNode* node)
  {
    if(node==nullptr) return;
    if(node->val==x)
    {
      node->val=y;
    }else if(node->val==y)
    {
      node->val=x;
    }
    recover(node->left);
    recover(node->right);
  }
public:
    void recoverTree(TreeNode *root) {
        vector<int> nums{};
        inorder(root, nums);
        findTwoSwappedNums(nums);
        recover(root);
    }
```

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
 // logn space
 private:
    TreeNode *x=nullptr, *y=nullptr, *pred=nullptr;
  void inorder(TreeNode* node)
  {
    if(node==nullptr) return;
    inorder(node->left);
    if(pred!=nullptr && node->val < pred->val)
    {
      y=node;
      if(x==nullptr) x=pred;
      else return;
    }
    pred=node;
    inorder(node->right);
```

```
  }
 public:
     void recoverTree(TreeNode *root) {
          inorder(root);
          int tmp=x->val;
          x->val=y->val;
          y->val=tmp;
     }
```

Implement with a constant space.

Ref: https://blog.csdn.net/shoulinjun/article/details/19051503

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)
{
     int len=sequence.size();
          if(len<=0) return false;

          //find the root
          int root=sequence[len-1];

          //left-subtree
          int i=0;
          vector<int> sequenceleft;
          for(; i<len-1; ++i) {
               if(sequence[i] > root)
                    break;
               sequenceleft.push_back(sequence[i]);
          }
          //right-subtree
          int j=i;
          vector<int> sequenceright;
          for(;j<len-1; ++j) {
               if(sequence[j]<root) return false;
               sequenceright.push_back(sequence[j]);
          }
          //judge
          bool left=true;
          if(i>0)
               left=VerifySquenceOfBST(sequenceleft);
          bool right=true;
          if(i<len-1)
               right=VerifySquenceOfBST(sequenceright);
          if(left && right){
               cout<<"Yes"<<endl;
               return true;
```

```
        }
        else {
            cout<<"No"<<endl;
            return false;
        }
}
```