Homework5

2017141493004 常家奇

- 5.5

  1. Modify the preorder traversal of Section 5.2 to perform an inorder traversal of a binary tree.

```
template <typename E>
void inorder(BinNode<E>* root){
  if (root == NULL) return;
  inorder(root->left());
  visit(root);
  inorder(root->right());
}
```

  2. Modify the preorder traversal of Section 5.2 to perform a postorder traversal of a binary tree.

```
template <typename E>
void postorder(BinNode<E>* root){
  if (root == NULL) return;
  postorder(root->left());
  postorder(root->right());
  visit(root);
}
```

- 5.7

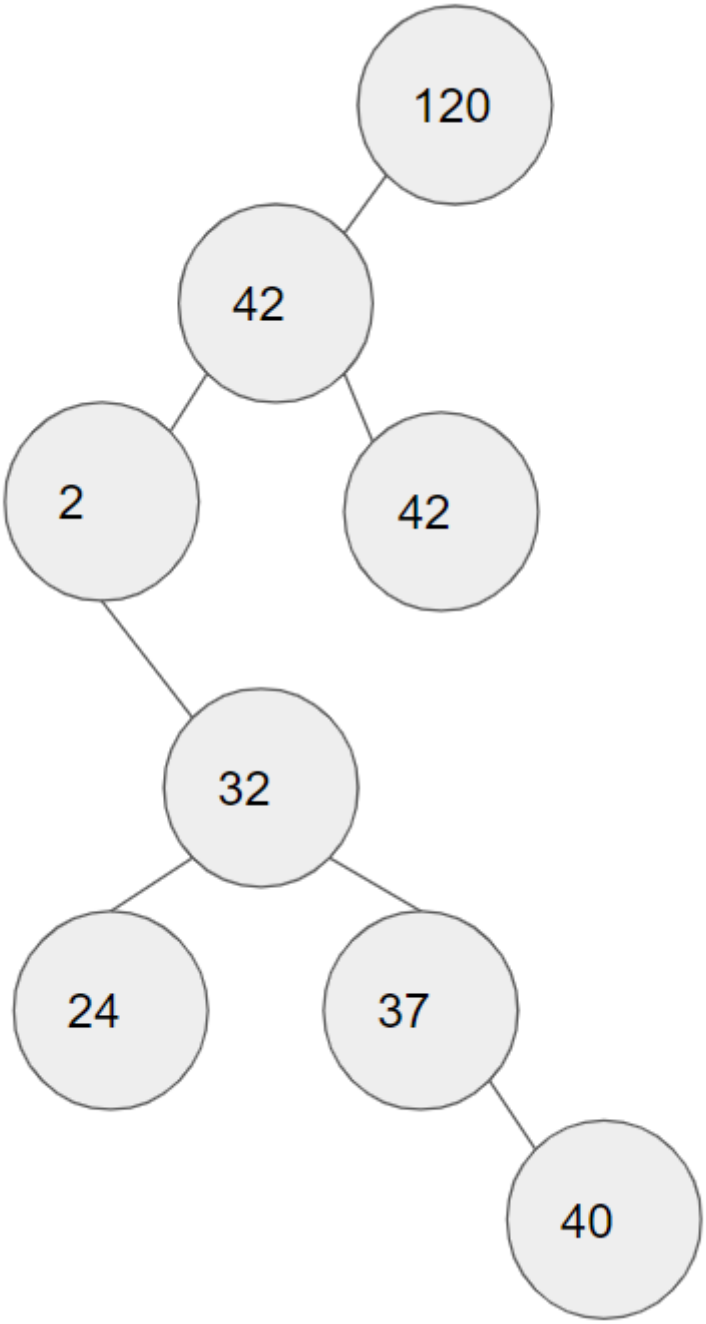
> 使用 `stl` 库中的 `queue` 来实现宽度优先遍历.

```
void bfs(BinNode<E>* root){
  typedef pair<BinNode<E>*,int> P;
  queue<P> q;
  q.push(P(root,0));
  while (!q.empty()){
    P pNode = q.front(); q.pop();
    cout << "Level" << pNode.second << ":" << pNode.first->value << endl;
    if (pNode.first->left()!=NULL) q.push(P(pNode.first->left(),pNode.second+1));
    if (pNode.first->right()!=NULL) q.push(P(pNode.first->right(),pNode.second+1));
  }
}
```

- 5.14

一般计算机科学中,区间表示有个不成文的规定:左闭右开.

- 5.17



- 5.21

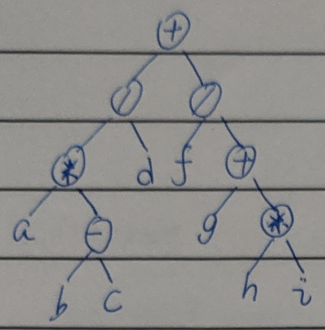
```
bool checkBST(BinNode<E>* root){
  if (root == NULL) return true;
  E v = root->value;
  bool left = checkBST(root->left());
  if (root->left() != NULL) left &= v>root->left()->value;
  bool right = checkBST(root->right());
  if (root->right() != NULL) right &= v<=root->right()->value;
  return left && right;
}
```

补充作业题

1. 1.b 2.a

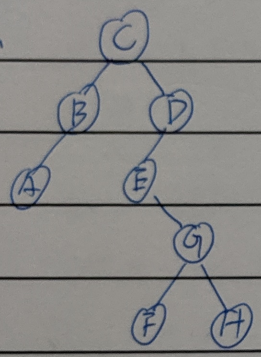
2.

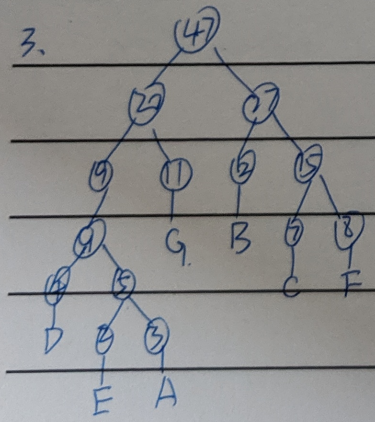
二、综合题

1. ① 

② abc-xd/fghi*+/+

③ a*(b-c)/d+f/(g+h*i)

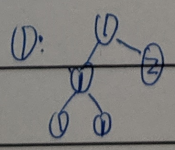
2. 

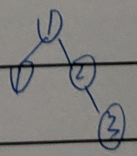
3. 

4.
```
void print(Node* root, int space=0){
    if(root==NULL) return;
    if(root->right != NULL) print(root->right, space+1);
    for(int i=0; i<space; i++) putchar(' ');
    printf("%c\n", root->value);
    if(root->left != NULL) print(root->left, space+1);
}
```

5. (1) ① 根左子树所有节点值与根相同

② 左子树只有一个节点，且值与根相同.

① 

② 

(2) ① 根右子树所有节点值与根相同

② 右子树只有一个节点，且值与根相同

同上

(3) 所有节点值相同.

6. (1)

```
int count(Node* root){
    if(root==Null return 0;
    return count(root->left)+1+count(root->right);
}
```

P.S: Naive Solution $O(2^n)$, 更高效的算法：在建树时就为每个节点做标记，[$O(1)$] 标记以其为根时叶结点个数，插入/删除时更新一个路径 $O(\log n)$

(2)

```
void change(Node* root){
    if(root==Null || root->left==Null && root->right==Null) return;
    Node* tmp=root->left;
    root->left= root->right;
    root->right=tmp;
    if(root->left!=NULL) change(root->left);
    if(root->right!=Null) change(root->right);
}
```