6.4

1. k^i
2. $\lfloor \frac{p+k-2}{k} \rfloor$
3. $kp-k+i+1$
4. $(p-1)\ mod\ k \neq 0$时 右兄弟编号为 $p+1$

6.6

$n-\frac{n-1}{k}$

6.8

可以这样考虑，每一个非叶子节点都是用一个节点换了K个节点出来，那么也就是说对于每一个非叶子节点，他们都能对叶子节点造成$k-1$的贡献，最开始有一个节点，换了n_1个，所以最终就是
$n_0=(k-1)n_1+1$

6.13

	前序遍历时，n在m前?	中序遍历时，n在m前?	后序遍历时，n在m前?
n在m左方	1	1	1
n在m右方	0	0	0
n是m祖先	1	#	0
n是m子孙	0	#	1

6.26

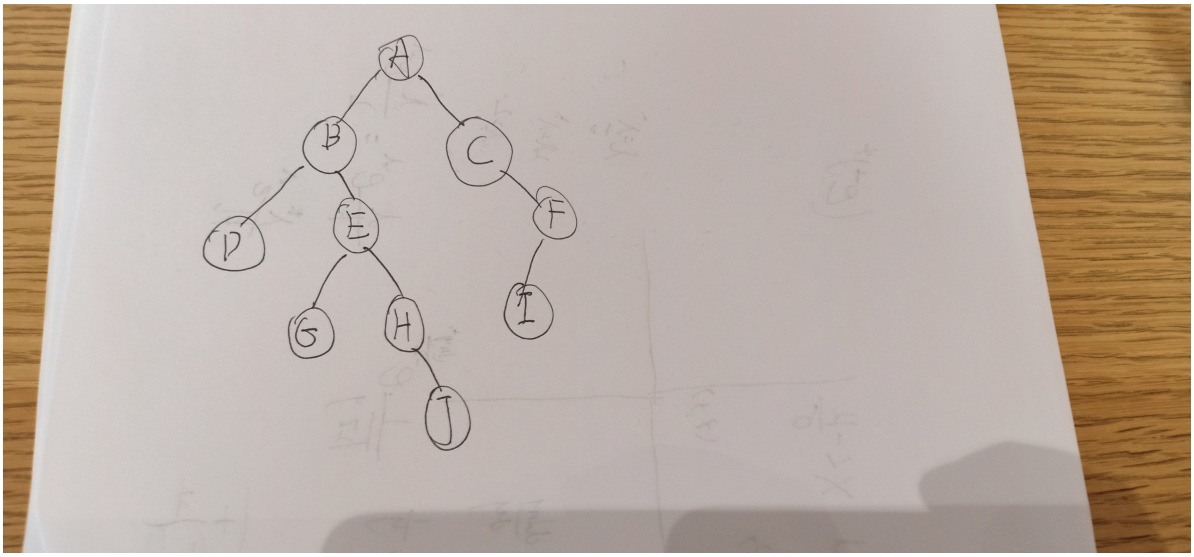
设为ABCDEFGH， 对应的权值为7 19 2 6 32 3 21 10

采用哈夫曼编码

根据左右子树的不同可以出现若干种情况，其中一种为

A:1101 B:01 C:11111 D:1110 E:10 F:11110 G:00 H:1100

6.29



6.33

```cpp
bool check(int v, int u) {
    if (v == 0) return false;
    if (v == u) return true;
    return check(L[v], u) || check(R[v], u);
}
```

6.37

```cpp
    vector<int> preOrder(TreeNode *root) {
    stack<TreeNode *> my_stack;
    my_stack.push(root);
  vector<int> ans;
    while (my_stack.size()) {
        TreeNode *cur = my_stack.top();
     my_stack.pop();
     ans.emplace_back(cur->val);
     if (cur->left != nullptr) my_stack.push(cur->left);
     if (cur->right != nullptr) my_stack.push(cur->right);
  }
  return ans;
}
```

6.42

```cpp
int count(TreeNode *root) {
    if (root == nullptr) return 0;
    if (root->left == nullptr && root->right == nullptr) return 1;
    return count(root->left) + count(root->right);
}
```

6.43

```
void swap(TreeNode *root) {
    if (root == nullptr) return;
    swap(root->left, root->right);
    swap(root->left);
    swap(root->right);
}
```

6.45

```
void del(TreeNode *root) {
    if (root == nullptr) return;
    del(root->left);
    del(root->right);
    root->left = root->right = nullptr;
    delete(root);
}
void del_x(TreeNode *root, int x) {
    if (root == nullptr) return;
    if (root->val == x) del(root);
    else {
        del_x(root->left, x);
        del_x(root->right, x);
    }
}
```

6.47

```
vector<int> bfs(TreeNode *root) {
    vector<int> ans;
    queue<TreeNode *>my_queue;
    my_queue.push(root);
    while (my_queue.size()) {
        TreeNode *now = my_queue.front();
        my_queue.pop();
        ans.emplace_back(now->val);
        if (now->left != nullptr) my_queue.push(now->left);
        if (now->right != nullptr) my_queue.push(now->right);
    }
    return ans;
}
```

6.56

```
TreeNode* preorderThreading(TreeNode *root) {
    TreeNode *pre = nullptr;
    TreeNode *head = new TreeNode();
    head->LTag = Link;
    head->RTag = Thread;
    head->right = head;
    if (root == nullptr) head->left = head;
    else {
        head->left = root;
```

```
        pre = head;
        preThreading(root, pre);
        pre->right = head;
        pre->RTag = Thread;
        head->right = pre;
    }
  return head;
}
void preThreading(TreeNode *root, TreeNode* &pre) {
   if (root->left != nullptr) {
       root->LTag = Thread;
       root->left = pre;
   }
   if (pre != nullptr && pre->right == nullptr) {
          pre->RTag = Thread;
          pre->right = root;
   }
   pre = root;
   if (root->LTag == Link) preThreading(root->left, pre);
   if (root->RTag == Link) preThreading(root->right, pre);
}
TreeNode *getNxtNode(TreeNode *root, Elem *p) {
    if (root == nullptr) return nullptr;
    if (root->val == *p) {
        if (root->LTag == Link) return root->left;
        return root->right;
    }
    if (now->LTag == Link) now = now->left;
    else now = now->right;
    while (now != root && now->val != *p) {
        if (now->LTag == Link) now = now->left;
        else now = now->right;
    }
    if (root == now) return nullptr;
    if (now->LTag == Link) return now->left;
    return now->right;
}
```

6.57

```
TreeNode* postorderThreading(TreeNode *root) {
    TreeNode *pre = nullptr;
    TreeNode *head = new TreeNode();
    head->LTag = Link;
    head->RTag = Thread;
    head->right = head;
    if (root == nullptr) head->left = head;
    else {
        head->left = root;
        pre = head;
        postThreading(root, pre);
        pre->right = head;
        pre->RTag = Thread;
        head->right = pre;
    }
  return head;
}
```

```
void postThreading(TreeNode *root, TreeNode* &pre) {
    if (root == nullptr) return;
    if (root->LTag == Link) postThreading(root->left, pre);
    if (root->RTag == Link) postThreading(root->right, pre);
    if (root->left == nullptr) {
        root->LTag = Thread;
        root->left = pre;
    }
    if (pre && pre->right == nullptr) {
        pre->RTag = Thread;
        pre->right = root;
    }
    pre = root;
}
TreeNode *getNxtNode(TreeNode *root, Elem *p) {
    if (root == nullptr) return nullptr;
    TreeNode *pre = root;
    TreeNode *now = root;
    if (now->RTag == Link) now = now->right;
    else now = now->left;
    while (now != root && now->val != *p) {
        pre = now;
        if (now->RTag == Link) now = now->right;
        else now = now->left;
    }
    if (now->val != *p) return nullptr;
    return pre;
}
```

6.62

```
int getDep(TreeNode *root) {
    if (root == nullptr) return 0;
    return max(getDep(root->child) + 1, getDep(root->nextsibling));
}
```