【第三十一周】哈弗曼编码（Halfman-Coding）与二叉字典树

1、1167. 连接棒材的最低费用

1. 每次取堆中的最小的两个值，当数组中只有一个值时，退出循环。
2. 注意需将每次连接的结果值放入数组中。

```
/**
 * @param {number[]} sticks
 * @return {number}
 */
//  dequeue是按照进入队列的先后顺序来取出元素。而在堆中，我们不是按照元素进入队列的先后顺序取出元素的，而是按照元素的优先级取出元素
var connectSticks = function(sticks) {
    let ans = 0;
    let p = new MinPriorityQueue();
    for(const x of sticks){
        p.enqueue(x,x);
    }
    while(p.size() > 1){
        let a = p.dequeue().element;
        a+= p.dequeue().element;
        ans+=a;
        p.enqueue(a,a);

    }
    return ans;
};
```

2、255. 验证前序遍历序列二叉搜索树

1. 因为二叉搜索树是左子树结点的值小于根结点值，右子树 结点的值大于根结点 值 ，所以我们可以从这个特点入手，如果可以找到左子树大于根节点或者右子树小于根节点的值，则说明该数组队列不满足二叉搜索树特点
2. 所以维护一个单调递减的栈，以及最新pop出来的值。待push进栈的节点值必须大于已经pop出来的所有元素的值，才能是合法的BST。

```
 var verifyPreorder = function(preorder) {
    const stack = [];
    let currMax = -Infinity;
    for (let n of preorder) {
        while (stack.length && n > stack[stack.length - 1]) currMax =
stack.pop();
        if (n < currMax) return false;
        stack.push(n);
    }
    return true;
};
```

3、676. 实现一个魔法字典

1. 我们替换searchWord的每一个位置的字符（与原来位置的字符不一样），然后在字典树中查询即可。

```
  function Trie() {
  this.map = {};
  this.isEnd = false;
}
/**
 * Initialize your data structure here.
 */
var MagicDictionary = function() {
  this.root = [];
};

/**
 * @param {string[]} dictionary
 * @return {void}
 */
MagicDictionary.prototype.buildDict = function(dictionary) {
  dictionary.forEach((item) => {
    let tmp = new Trie();
    this.insert(tmp, item);
    this.root.push(tmp);
  });
};

MagicDictionary.prototype.insert = function(node, word) {
  for (let i = 0; i < word.length; i++) {
    node.map[word[i]] = node.map[word[i]] || new Trie();
    node = node.map[word[i]];
  }
```

```
  node.isEnd = true;
};

/**
 * @param {string} searchWord
 * @return {boolean}
 */
MagicDictionary.prototype.search = function(searchWord) {
  return this.root.reduce((total, item) => {
    return total || this.searchDfs(item, searchWord);
  }, false);
};

MagicDictionary.prototype.searchDfs = function(node, searchWord) {
  let flag = false;
  for (let i = 0; i < searchWord.length; i++) {
    if (node.map[searchWord[i]]) {
      node = node.map[searchWord[i]];
    } else if (!flag) {
      flag = true;
      let keys = Object.keys(node.map);
      return keys.reduce((total, item) => {
        return total || this.dfs(node.map[item], searchWord.slice(i + 1));
      }, false);
    } else {
      return false;
    }
  }
  return false;
};


MagicDictionary.prototype.dfs = function(node, word) {
  for (let i = 0; i < word.length; i++) {
    if (node.map[word[i]]) {
      node = node.map[word[i]];
    } else {
      return false;
    }
  }
  return node.isEnd;
};
```

4、76. 最小覆盖子串

1. 我们在字符串 S 中使用双指针中的左右指针技巧，初始化 left = right = 0，把索引闭区间 [left, right] 称为一个「窗口」。
2. 我们先不断地增加 right 指针扩大窗口 [left, right]，直到窗口中的字符串符合要求（包含了 t 中的所有字符）
3. 我们停止增加 right，转而不断增加 left 指针缩小窗口 [left, right]，直到窗口中的字符串不再符合要求（不包含 T 中的所有字符了）。同时，每次增加 left，我们都要更新一轮结果
4. 重复第 2 和第 3 步，直到 right 到达字符串 S 的尽头

```
/**
 * @param {string} s
 * @param {string} t
 * @return {string}
 */
var minWindow = function(s, t) {
  // 需要的
  let need = {};
  // 窗口中的字符
  let window = {};
  for (let a of t) {
    // 统计需要的字符
    need[a] = (need[a] || 0) + 1;
  }
  // 左右指针
  let left = 0,
    right = 0;
  let valid = 0;
  // 最小覆盖子串的起始索引及长度
  let start = 0,
    len = Number.MAX_VALUE;
  while (right < s.length) {
    // 即将移入窗口的字符
    let c = s[right];
    // 右移窗口
    right++;
    if (need[c]) {
      // 当前字符在需要的字符中，则更新当前窗口统计
      window[c] = (window[c] || 0) + 1;
      if (window[c] == need[c]) {
        // 当前窗口和需要的字符匹配时，验证数量增加1
        valid++;
      }
    }
    // 当验证数量与需要的字符个数一致时，就应该收缩窗口了
    while (valid == Object.keys(need).length) {
      // 更新最小覆盖子串
      if (right - left < len) {
        start = left;
        len = right - left;
```

```
      }
      //即将移出窗口的字符
      let d = s[left];
      // 左移窗口
      left++;
      if (need[d]) {
        if (window[d] == need[d]) {
          valid--;
        }
        window[d]--;
      }
    }
  }
  return len == Number.MAX_VALUE ? "" : s.substr(start, len);
};
```