我自己做的,不一定对

一

1-4. CBBB
5. BCD 42
6. 14 {12,24,33,65,33,56,48,92,86,70}

> 这里两个33很奇怪

7 C

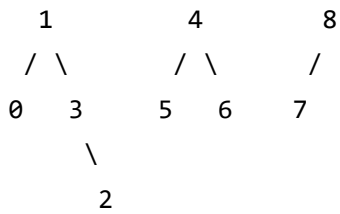
二

1.

顺序栈与链式栈在时间效率上相同，均可在常数时间内实现插入和删除。在空间效率上，顺序栈长度固定，可能造成空间浪费;链式栈长度可变，节约了空间，但构建它需要指针等结构性开销，与导致额外开销。

> 空间效率和顺序数组链式数组一样, 但是时间效率栈只需要再顶部插入删除

1. BGI BYW UVFZ
2. f 0; c 100; d 101; e 111;a 1100; b 1101, 节省了25.33%的空间
3. (1)



(2)
[1,-6,1,1,-3,4,4,8,1]

我认为2union前应该在find的时候路径压缩到1了，7则还是指向8

三

1.

```
!st.empty()
st.pop()
topElem.rd==1
getChildrenNum() - i - 1
stackElement(child,0)
```

> 这里第4个不应该是 i, zl回放似乎写错了

四

1.

```
class StackUsingQueues {
private:
    queue<int> A, B; // 使用两个队列 A 和 B
    // 单独写四个函数即可
public:
    // 入栈操作
    void push(int x) {
        // 把 A 中的所有元素先放到 B 中
        while (!A.empty()) {
            B.push(A.front());
            A.pop();
        }

        // 把新元素放到 A 中
        A.push(x);

        // 把 B 中的元素再放回 A 中
        while (!B.empty()) {
            A.push(B.front());
            B.pop();
        }
    }

    // 出栈操作
    void pop() {
        if (A.empty()) {
            cout << "Stack is empty!" << endl;
            return;
        }

        // 直接从队列 A 中移除元素（队首即为栈顶元素）
        A.pop();
    }

    // 获取栈顶元素
    int top() {
        if (A.empty()) {
            cout << "Stack is empty!" << endl;
            return -1; // 或者返回适当的错误值
        }

        // 队首即为栈顶元素
        return A.front();
```

```
    }

    // 判断栈是否为空
    bool empty() {
        return A.empty();
    }
};
```

2.

计算二维数组 next

- next[x][y]表示字符串 S 中以 x 开头以 y-1 为结尾的子串的前缀的最大相同后缀的长度, 然后返回以下值:

$$\min_{\exists y, \text{next}[x][y]=\frac{y-x}{2}} x$$

下证明此处的 x 一定是第一个AA出现的位置.

- 假如不是, 那么第一个AA的位置有 $\text{next}[x][y] \neq \frac{y-x}{2}$. 由于AA就是相等的前后缀, 因此 $\text{next}[x][y] > \frac{y-x}{2} = c$
- 则A可以被拆分成 XY, 使得 $XYX = YXY$,从而X和Y等长, 从而$X = Y$, 这与XYXY是第一个AA矛盾, 证毕

3.

两种方法,一种差分暴力,另一种动态规划(f(X,sum)表示X开头的路径和为sum的数目),复杂度都是n^2

五

(1) $(k-1)m+1$
(2) $\frac{k^h-1}{k-1}, 1+(h-1)k$