

## 02-C-3 插入 & 02-C-4 区间删除

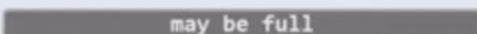
#数据结构邓神


### 插入

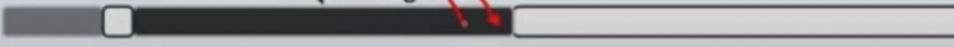
```
template <typename T>
Rank Vector<T>::insert(Rank r, T const& e){ // O(n-r)
    expand(); // 有需要时候拓展
    for (int i = _size; i > r; ++i) {
        _elem[i] = _elem[i-1]; // 后移元素
    }
    _elem[r] = e; // 赋值
    _size++; // 更新容量
    return r; // 返回秩
}
```

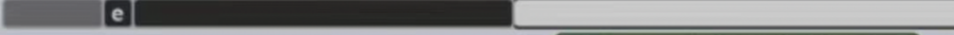
**插入**

```
❖ template <typename T> //e作为秩为r元素插入, 0 ≤ r ≤ size
Rank Vector<T>::insert(Rank r, T const & e) { //O(n-r)
    expand(); //若有必要, 扩容
    → for (int i = _size; i > r; i-- //自后向前
        _elem[i] = _elem[i-1]; //后继元素顺次后移一个单元
    _elem[r] = e; _size++; //置入新元素, 更新容量
    return r; //返回秩
}
```

(a) 

(b) 

(c) 

(d) 

### 区间删除

删除 [lo,hi) 内部的所有元素

```
template <typename T>
int Vector<T>::remove(Rank lo, Rank hi){
    if(lo == hi){
```

```

        return 0;
    }
    while(hi < _size){
        _elem[lo++] = _elem[hi++];
    }
    _size = lo;
    shrink(); // 如果有必要就减少空间
    return hi-lo; // 返回被删除元素的个数
}

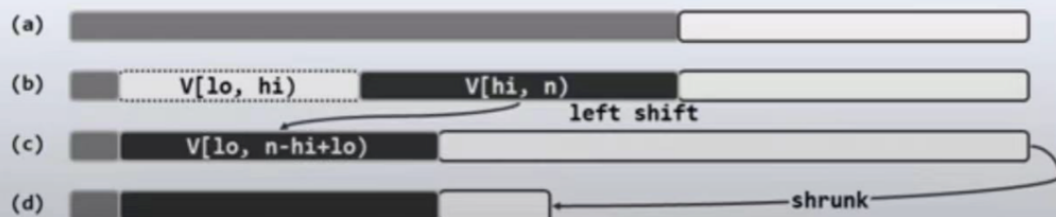
```

### 区间删除

```

❖ template <typename T> //删除区间[lo, hi), 0 <= lo <= hi <= size
int Vector<T>::remove(Rank lo, Rank hi) { //O(n - hi)
    if (lo == hi) return 0; //出于效率考虑, 单独处理退化情况
    → while (hi < _size) _elem[lo++] = _elem[hi++]; //[hi, _size)顺次前移hi-lo位
    _size = lo; shrink(); //更新规模, 若有必要则缩容
    return hi - lo; //返回被删除元素的数目
}

```



注意一定是从前面往后，否则如果有重叠的部分会造成问题（可能会在无意中被覆盖）