

向量

无序向量：基本操作

02-C1

邓俊辉

deng@tsinghua.edu.cn

A scientist discovers that which exists,  
an engineer creates that which never was.

# 元素访问

❖ `V.get(r)`和`V.put(r,e)`不够便捷、直观，可否沿用数组的访问方式`V[r]`?

可以！比如，通过重载下标操作符`[]`

❖ `template <typename T> //可作为左值:  $V[r] = (T) (2*x + 3)$`

```
T & Vector<T>::operator[] ( Rank r ) { return _elem[ r ]; }
```

❖ `template <typename T> //仅限于右值:  $T\ x = V[r] + U[s] * W[t]$`

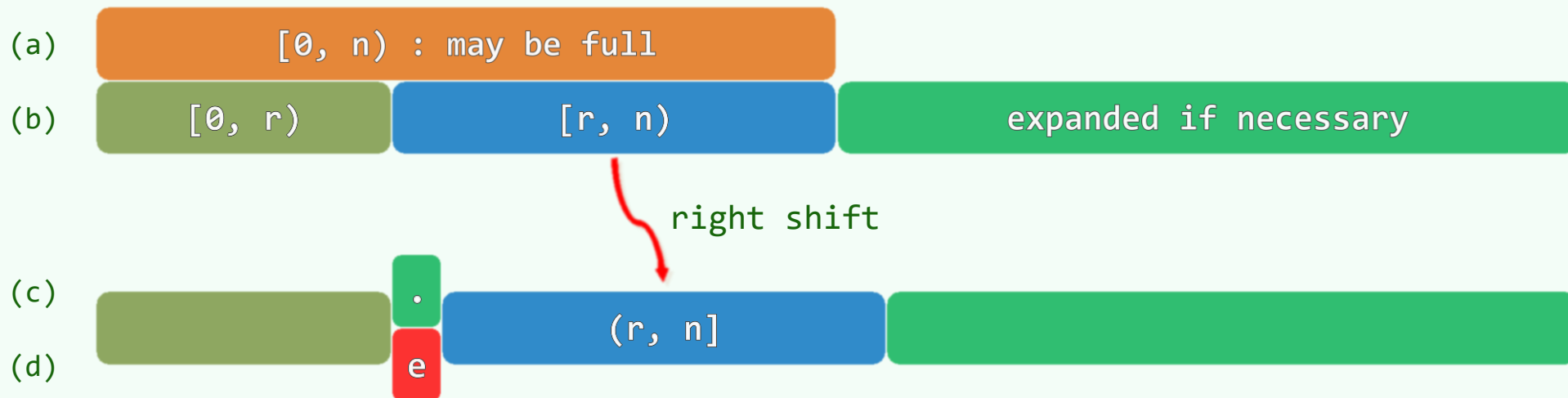
```
const T & Vector<T>::operator[] ( Rank r ) const { return _elem[ r ]; }
```

❖ 这里采用了简易的方式处理意外和错误（比如，入口参数约定： $0 \leq r < \_size$ ）

实际应用中，应采用更为严格的方式

# 插入

```
template <typename T> Rank Vector<T>::insert( Rank r, T const & e ) { //0<=r<=size  
    expand(); //如必要, 先扩容  
    for ( Rank i = _size; r < i; i-- ) //0(n-r): 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++; return r; //置入新元素, 更新容量, 返回秩  
}
```



## 区间删除

```
template <typename T> Rank Vector<T>::remove( Rank lo, Rank hi ) { //0<=lo<=hi<=n

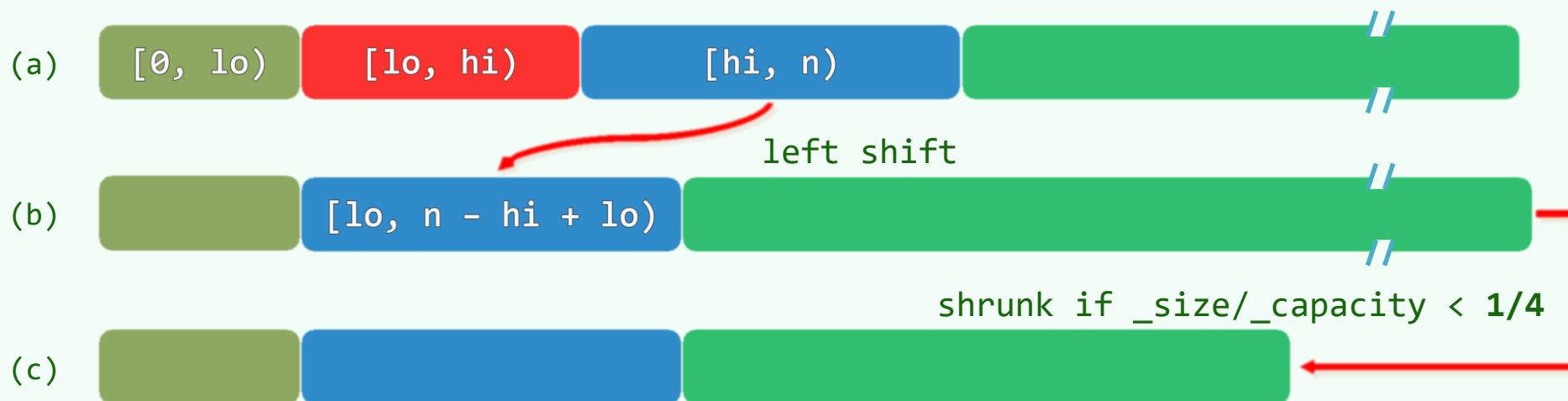
    if ( lo == hi ) return 0; //出于效率考虑, 单独处理退化情况

    while ( hi < _size ) _elem[ lo++ ] = _elem[ hi++ ]; //后缀[hi,n)前移

    _size = lo; shrink(); //更新规模, lo = _size之后的内容无需清零; 如必要, 则缩容

    return hi - lo; //返回被删除元素的数目

}
```



# 单元素删除

```
template <typename T>

T Vector<T>::remove( Rank r ) {

    T e = _elem[r]; //备份

    remove( r, r+1 ); // “区间” 删除

    return e; //返回被删除元素

} //O(n-r)
```

❖ 也就是将**单元素**视作**区间**的特例:

$$[r] = [r, r + 1)$$

❖ 反过来, 通过反复调用remove(r)接口实现  
remove(lo, hi)呢?

❖ 每次循环耗时, 正比于删除区间的后缀长度

$$n - hi = O(n)$$

而循环次数等于区间宽度

$$hi - lo = O(n)$$

如此, 将导致总体 $O(n^2)$ 的复杂度