2.向量

无序向量

基本操作

它污浊，它美丽，它衰老，它活泼，它杂乱，
它安闲，它可爱，它是伟大的夏初的北平

邓俊辉

deng@tsinghua.edu.cn

元素访问

- 似乎不是问题：通过`V.get(r)`和`V.put(r, e)`接口，已然可以读、写向量元素
- 但就**便捷性**而言，远不如数组元素的访问方式：`A[r]`　　//可否沿用借助下标的访问方式？
- 可以！为此，需**重载**下标操作符“[]”

```
template <typename T> //0 <= r < _size
T & Vector<T>::operator[]( Rank r ) const { return _elem[ r ]; }
```

- 此后，对外的`V[ r ]`即对应于内部的`V._elem[ r ]`

右值：`T x = V[r] + U[s] * W[t];`

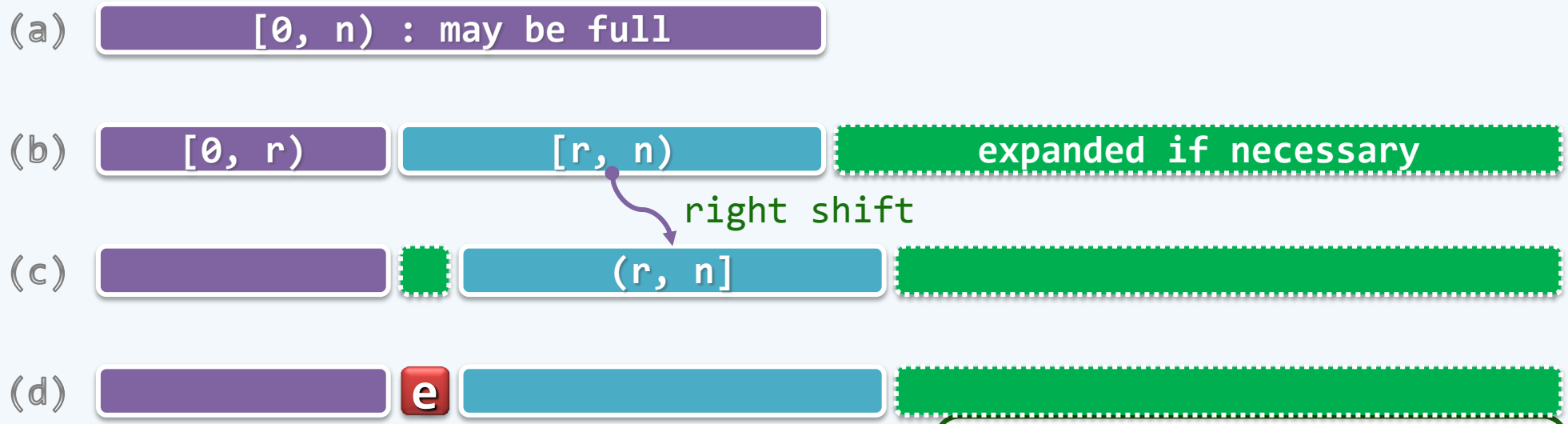
左值：`V[r] = (T) (2*x + 3);`

- 为便于讲解，这里采用了简易的方式处理意外和错误（比如，入口参数越界等）

实际应用中，应采用更为严格的方式

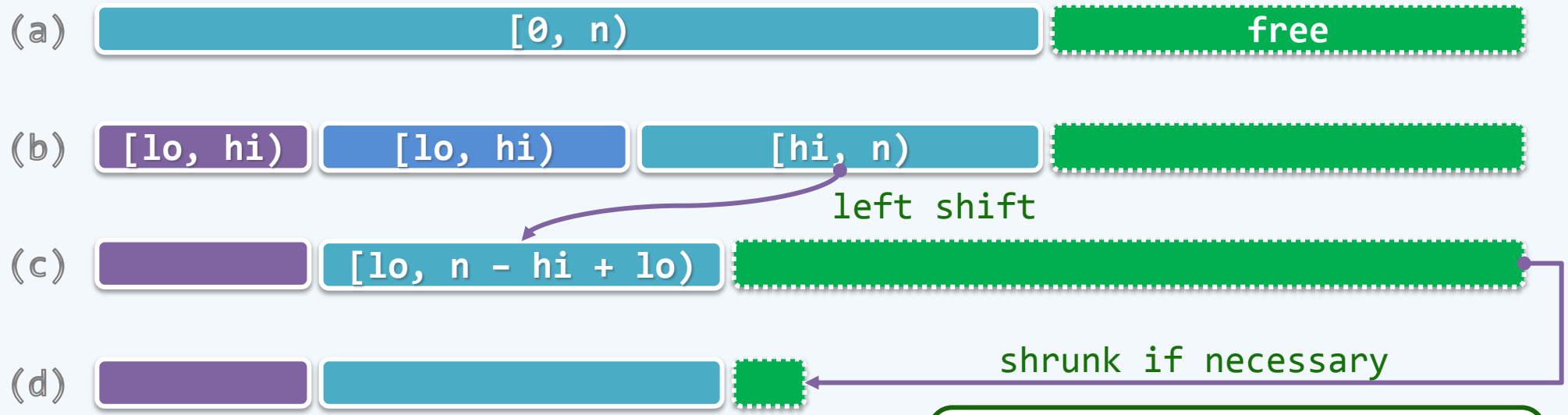
插入

```
❖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++; return r;  //置入新元素，更新容量，返回秩
  }
```



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
     _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
  }
```



单元素删除

- 可以视作区间删除操作的特例：[r] = [r, r + 1)
- template <typename T> //删除向量中秩为r的元素，0 <= r < size

```
T Vector<T>::remove( Rank r ) { //O(n - r)
   T e = _elem[r]; //备份被删除元素
   remove( r, r + 1 ); //调用区间删除算法
   return e; //返回被删除元素
}
```

- 反过来，基于remove(r)接口，通过反复的调用，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