中国人民大學

RENMIN UNIVERSITY OF CHINA

XXXX

XXXXX

学院 _____

专业 _____

学号 _____

姓名 _____

2025 年 8 月 25 日

摘　要

XXXXX

关键词：XXX

目录

1 Interface(API and ADT) vs data structure

Interface can be called as specification, specify what data can be stored, whereas the data structure will give an actual representation and tell how to store it. In the interface, you specify what the operations do, what operations are supported, and in some sense, what they mean. And the data structure actually gives you algorithms– this is where the algorithms come in for how to support those operations.

2 Two main interfaces:set and sequence

2.1 static sequence interface

The number of items doesn't change, though the items might.

- size(): return the number of items in the sequence
- get(i): return the item at position i
- set(i,x): set the item at position i to x

2.2 dynamic sequence interface

The number of items can change, and the items can be moved around.

- insert(i,x): insert x at position i
- remove(i): remove the item at position i

2.3 link list

We store items in a bunch of nodes, each node has an item in it and a next field. These can be considered as class objects with two class variables, the item and the next pointer.

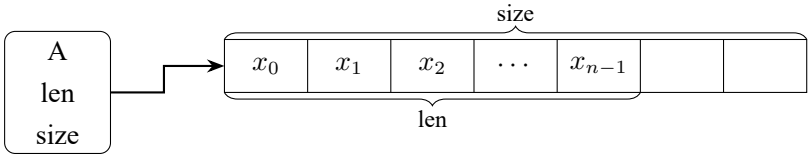
Insert and delete_first cost $\theta(1)$ time, but get cost $\theta(n)$ time.

2.4 static array

Insert and delete cost $\theta(n)$ time. For 2 reasons, reason 1 is that, if we are near the front, the we have to do shifting. In a static array, this is not a problem, but in a dynamic array, it can be. reason 2 is that you are not allowed to change the number of items, so you have to allocate a new array. you need to copy all the items over, and that takes $\theta(n)$ time.

2.5 dynamic array(Python lists)

The idea is to relax the constraint(约束) or the invariant(不变式), whatever that the size of the array we use equals n, which is the number of items in the sequence. We enforce that the size of the array is $\theta(n)$ probably also greater than or equal to n. We are still going to maintain that the ith item of the array represents x_i.



If want to do an insert_last, just go to a of length and set it to x, and increment length.

If len = size, we are going to allocate a new array, typically twice as big, and the resize operation takes $\theta(n)$ time ($\theta\left(\sum_{i=1}^{\log_2 n} 2^i\right)$).

3 amortization(摊还：把高昂操作的成本均摊到多次操作，分析平均复杂度)

An operation takes $T(n)$ amortized time, if any k operations takes $\leq k \cdot T(n)$ time. (averaging over operation sequence)

Amortized analysis means spreading the cost of expensive operations over many cheap ones. For a dynamic array, most inserts take constant time $O(1)$, but sometimes the array is full and needs to resize, which costs $O(n)$. However, resizing does not happen every time—only occasionally (like doubling the size). If you insert n items, the total cost of all resizes is about $2n$, so the average cost per insert is still $O(1)$. This is why we say appending to a dynamic array has amortized $O(1)$ time.

Data Structure	Container	Static		Dynamic			
	build(X)	get_at(i)	set_at(i, x)	insert_first(x)	delete_first()	insert_last(x)	delete_last(
Array	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
Linked List	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$
Dynamic Array	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$ (amortized)	$O(1)$ (amortiz

表 1 Worst-case and amortized time complexity of basic operations.

Set 操作	**含义**	**如何用 Sequence 实现**
insert(x)	插入元素 x（若已有相同 key 就替换）	遍历 $seq[0..n-1]$，若有相同 key 则 set_at(i, x)，否则 insert_last(x)
delete(k)	删除 $key = k$ 的元素	遍历 $seq[0..n-1]$，找到后执行 delete_at(i)
find(k)	查找 $key = k$ 的元素	遍历 $seq[0..n-1]$，若 get_at(i).key == k 则返回该元素
find_min()	找到最小 key	遍历所有元素，取最小 key
find_max()	找到最大 key	遍历所有元素，取最大 key
find_next(k)	找到大于 k 的最小 key	遍历所有元素，取所有 $x.key > k$ 的候选，返回最小的一个
find_prev(k)	找到小于 k 的最大 key	遍历所有元素，取所有 $x.key < k$ 的候选，返回最大的一个
iter_ord()	按 key 顺序遍历	先调用 find_min()，再不断调用 find_next()

表 2 Sequence 操作模拟 Set 操作的对照表