

1 Dictionaries

A set is a collection of distinguishable objects/elements/members. A set is dynamic if it can grow, shrink or otherwise change over time. A dynamic set is a dictionary if it supports at the least algorithmic operations of inserting elements into it, deleting elements from it and testing membership of it. (See Textbook Section 4.4)

The dictionary ADT includes:

Objects: Sets S where each element x has field $key[x]$ – some totally ordered value.

Operations: The following is three of the most typical operations on a given dictionary S .

$SEARCH(S, k)$: return x in S such that $key[x] = k$, or NIL if no such x ;

$INSERT(S, x)$: insert x in S ;

$DELETE(S, x)$: remove x from S (given the element x in S , not just the key of x).¹

Many variations on the ADT dictionary are possible: - require unique keys / allow duplicate keys; add operations to change information associated with a specific key, etc. (e.g., when we indeed have the assumption that keys are distinct – no two keys are the same, for the operation $INSERT(S, x)$ then, it is required then if some y in S has $key[y] = key[x]$, we replace y by x);

Lets look at two very simple ones first: unsorted array and sorted array. The worst-case time complexity for each operation are given here:

Data Structure	SEARCH	INSERT	DELETE
unsorted array	$\Theta(n)$	$\Theta(1)^*$	$\Theta(1)^{**}$
sorted array	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n)$

★) If replacement is required (to ensure that keys are distinct), then it takes $\Theta(n)$ instead.

★★) If deletion is defined in the form “ $DELETE(S, k)$ ”, then it takes $\Theta(n)$ instead, as “ $SEARCH(S, k)$ ” need to be invoked first to locate the element x in the array.

2 Linked Lists

A linked list is a data structure in which the objects are arranged in a linear order. Each element in a doubly linked list L is an object with a key field and two other pointer fields: next and prev. Given an element x in the list, $next[x]$ points to its successor in the linked list, and $prev[x]$ points to its predecessor. If $prev[x] = NIL$, the element x has no predecessor and is therefore the first element or head of the list. If $next[x] = NIL$, the element x has no successor and is therefore the last element, or tail of the list. An attribute $head[L]$ points to the first element of the list. If $head[L] = NIL$, the list is empty.

¹But why not $DELETE(S, k)$? Answer: This can be achieved with $DELETE(S, SEARCH(S, k))$; However we want to separate “searching” phase from “deletion” phase, making it possible to analyse each one separately.

Consider a doubly linked list L . The procedure $ListSearch(L, k)$ finds the first element with key k in list L , returning a pointer to this element. If no object with key k appears in the list, then NIL is returned. The procedure takes $\Theta(n)$ time in the worst-case, since it may have to search the entire list.

```
ListSearch(L, k)
1: x ← head[L]
2: while (x ≠ NIL and key[x] ≠ k)
3:   x ← next[x]
4: return x
```

Given an element x whose key field has already been set, the $ListInsert$ procedure inserts x onto the front of the linked list. The running time is $\Theta(1)$.

```
ListInsert(L, x)
1: next[x] ← head[L]
2: if head[L] ≠ NIL
3:   then prev[head[L]] ← x
4: head[L] ← x
5: prev[x] ← NIL
```

The procedure $ListDelete$ removes an element x from L . A pointer to x is given, and it then delete x out of L by updating pointers. The procedure runs in $\Theta(1)$ time. Again (see the footnote of Page one), if we wish to delete an element with a given key, $ListSearch$ is called to retrieve a pointer to the element. In that case, $\Theta(n)$ time is required in the worst case.

```
ListDelete(L, x)
1: if prev[x] ≠ NIL
2:   then next[prev[x]] ← next[x]
3:   else head[L] ← next[x]
4: if next[x] ≠ NIL
5:   then prev[next[x]] ← prev[x]
```

What if L is singly linked list?

Data Structure	SEARCH	INSERT	DELETE
singly linked list	$\Theta(n)$	$\Theta(1)^*$	$\Theta(n)^{**}$
doubly linked list	$\Theta(n)$	$\Theta(1)^*$	$\Theta(1)$

★) If replacement is required (to ensure that keys are distinct), then it takes $\Theta(n)$ instead.

★★) An element in a singly linked list does not store information of its predecessor, which is needed for deletion to ensure that elements are still linked after the deletion. This takes $\Theta(n)$ time.