Assignment1(Group 18)

Wang Dingkun/Qian Shuaicun/Hua Bingsheng/Xu Ke

Ex.1

(i)Consider using unbounded array to realise the list. We only consider $append(insert)$ and $delete$ operations since other operations rely on them.
Each $append/insert$ opertion is $O(1)$ and we assume the cost is 1 and we consider the $delete$ operation as the last operation.
Suppose there are n_0 elements in the list now, we consider the sequence of $append/insert$ operations after the previous $delete$ operation.
There is at least n_0 $append/insert$ operations and the total cost is n_0.
The $delete$ operation has a cost k and $k <= n_0$, so we let the cost be n_0. And if the $delete$ operation makes a deallocation (before or after), which will copy at most n_0 to a new list, so the cost is also n_0. We can $deallocate$ at most $log_2(n_0)$ times each time the cost will be $n_0/2, n_0/4...$ The total cose of $deallocate$ is no more than $2n_0$. Thus for the operation sequence before last $delete$ (or from an empty list), the total cost is no more than $4n_0$, the amortised complexity is $O(\frac{4n_0}{n_0}) = O(1)$ for each operation.

For double-linked list, the $insert$ cost is the same n_0 from one $delete$ to the next, and the cost of $delete$ is n_0 because no deallocation we need to do. The total cost of at most n_0 operations are $2n_0$, thus the amortised complexity is $O(\frac{2n_0}{n_0}) = O(1)$.
In both cases, the complexity is $O(1)$, independent of k.

Ex.2

(i)Since if the list l and parameters $[e, f, g]$ are determined, the result of $src[e, f, g]$ is also uniquely defined, the well-defined conditions are those that ensure the uniqueness of the result with given parameters.
For any split of a list l, $l = l_1 + l_2$, $g(src[e, f, g](l_1), src[e, f, g](l_2)) = src[e, f, g](l)$ must be the same.
Or we can see g is associative with respect to the concatenation of the same (sub)list l.
$l = l_1 + l_2 + l_3$ then $g(src[e, f, g](l_1), g(src[e, f, g](l_2), src[e, f, g](l_3))) = src[e, f, g](l_1 + l_2 + l_3) = g(g(src[e, f, g](l_1), src[e, f, g](l_2)), src[e, f, g](l_3))$
Also we have a netural element e of g such that $g(src[e, f, g](l), e) = g(e, src[e, f, g](l)) = g(src[e, f, g](l), src[e, f, g]([])) = src[e, f, g](l)$.

(ii)(1) Let the length function $len(l) = src[0, 1, +](l)$, we let

$$len([]) = e = 0$$
$$len([x]) = f(x) = 1$$
$$len(l_1 + l_2) = g(len(l_1), len(l_2)) = len(l_1) + len(l_2)$$
$$T' = N_{\geq 1}$$

(2) Let the function implemented is $F : T \to P$, let $func(l) = src[[], [F(x)], +](l)$, which $+$operation concatenates two lists. Then we have

$$func([]) = e = []$$
$$func([x]) = f(x) = [F(x)]$$
$$func(l_1 + l_2) = g(func(l_1), func(l_2)) = func(l_1) + func(l_2)$$
$$T' = \{l'\}$$

where l' are lists with elements $\in P$.
(3) Let the sub function $sub(l) = src[[], f(x), +](l)$, we have

$$sub([]) = e = []$$
$$sub([x]) = f(x) = \begin{cases} [x], & \varphi(x) \\ [], & else \end{cases}$$
$$sub(l_1 + l_2) = g(sub(l_1), sub(l_2)) = sub(l_1) + sub(l_2)$$
$$T' = \{l'' | l'' \subset l\}$$

(iii) Let the length of the whole list be n, let $x, x \in T$ be a single element of the list, let $y_1, y_2 \in T'$.
We apply $f(x)$ exactly n times and $g(y_1, y_2)$ exactly $n - 1$ times, suppose each time cost of $f(x)$ is T_f, each time cost of $g(y_1, y_2)$ is T_g, the total complexity is $O(n * T_f + (n - 1) * T_g) = O(n * max(T_f, T_g))$
Since the complexity of g may depend on how the list is realised and how to do the structural recursion (e.g. When we want to combine sublists, the complexity of g varies with your strategies), the real complexity will improve if choosing more effective ways to implement f and g.
Furthermore, the complexity of $f(x)$ and $g(y_1, y_2)$ also changes with different parameters, so we use the average time, the total complexity is $O(n * max(\overline{T_f}, \overline{T_g}))$.

Ex.3

(i)$pushback(l, x)$ can be expressed by $append(l, x)$.
$pushfront(l, x)$ can be expressed by $insert(l, 1, x)$.
$popback(l)$ can be expressed by $n = l.getlength()$ and $l.getitem(n)$ and $remove(l, n)$.
$popfront(l)$ can be expressed by $l.getitem(1)$ and $remove(l, 1)$.