National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2013/2014

Recitation 6
Lists and List Processing II

List Library

Recall the following library:

```
function map(f, xs) {
    return (is_empty_list(xs))
        ? []
        : pair(f(head(xs)), map(f, tail(xs)));
}

function filter(pred, xs){
    if (is_empty_list(xs)) {
        return xs;
    } else {
        if (pred(head(xs))) {
            return pair(head(xs), filter(pred, tail(xs)));
        } else {
            return filter(pred, tail(xs));
        }
    }
}

function accumulate(op, initial, sequence) {
    if(is_empty_list(sequence)) {
        return initial;
    } else {
        return op(head(sequence), accumulate(op, initial, tail(sequence)));
    }
}
var fold_right = accumulate;
```

Problems:

1. A *queue* is a data structure that stores elements in order. Elements are `enqueue`d onto the tail of the queue. Elements are `dequeue`d from the head of the queue. Thus, the first element enqueued is also the first element dequeued (FIFO, first-in-first-out). The `qhead` operation is used to get the element at the head of the queue.

```
qhead(enqueue(5, empty_queue()))
// Value: 5

var q = enqueue(4, enqueue(5, enqueue(6, empty_queue())));

qhead(q);
// Value: 6

qhead(dequeue(q));
// Value: 5
```

(a) Decide on an implementation for queues, then draw a box-and-pointer represention of the value of `q` as defined above.

(b) Write `empty_queue`.

```
function empty_queue() {

}
```

(c) Write `enqueue`; a procedure that returns a new queue with the element added to the tail.

```
function enqueue(x, q) {

}
```

Order of growth in time? Space?
Can you do this with `fold_right`?

(d) Write `dequeue`; a procedure that returns a new queue with the head element removed.

```
function dequeue(q) {

}
```

Order of growth in time? Space?

(e) Write `qhead`; a procedure that returns the value of the head element.

```
function qhead(q) {

}
```

Order of growth in time? Space?

Note: an alternative design is to have head of the list as rear of the queue. For practise, implement the corresponding procedures above. In the process, it will be clear why the choice of a representation matters *a lot*.

2. Suppose `x` is bound to `list(1, 2, 3, 4, 5, 6, 7)`. Using `map`, `filter`, and/or `fold_right`, write an expression involving `x` that returns:

(a) `list(1, 4, 9, 16, 25, 36, 49)`

(b) `list(1, 3, 5, 7)`

(c) `list(list(1, 1), list(2, 2), list(3, 3), list(4, 4), list(5, 5), list(6, 6), lis`

(d) `list(list(2), list(list(4), list(list(6), false)))`

(e) The maximum element of `x`: 7

(f) The last pair of `x`: `pair(7,list())`

6. **Homework:** Implement the procedure `union_set` for the different set representations discussed in Lecture 11. What is the order of growth (time and space) for your procedures?