Problem (3 / 6 points): It was a dark and mysterious recursion…

Consider the recursive procedure `gather` that takes a sentence of at least two single-character words (i.e., letters such as 'a', 'b', etc.):

```
;; sent-of-ltrs is a sentence of at least 2 words that are single
;;   letters
(define (gather sent-of-ltrs)
   (cond ((empty? sent-of-ltrs) '())
         ((empty? (bf sent-of-ltrs))
          (se (first sent-of-ltrs)))
         ((equal? (first (first sent-of-ltrs))
                  (first (bf sent-of-ltrs)))
          (gather (se (word (first sent-of-ltrs)
                            (first (bf sent-of-ltrs)))
                      (bf (bf sent-of-ltrs)))))
         (else
          (se (first sent-of-ltrs)
              (gather (bf sent-of-ltrs))))))
```

Part A (3 points). What will `(gather '(a b b b c d d))` return?

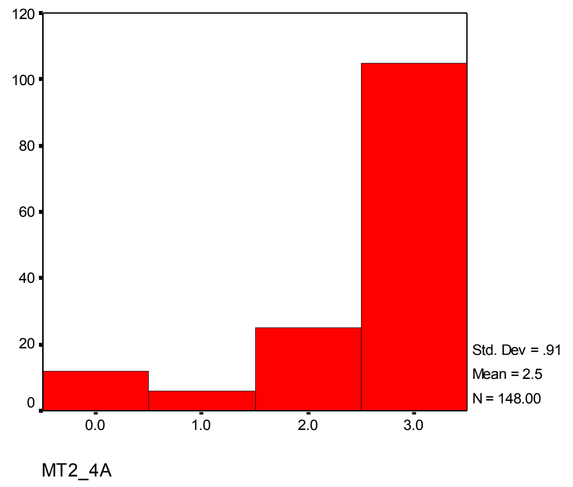
This expression returns `(a bbb c dd).`

-1: A common mistake was to return `(a bb b c dd).` Note that in the third case of `gather`, if the first two letters are equal, they are formed into a word and passed into `gather` again. The letters are not extracted until the next letter is different.

-2: Another mistake was to return the argument as is: `(a b b b c d d).`

-3: No points were given for the answer: `(a b c d).` No letters from the argument should have been removed.

Some of you put down `'(a bbb c dd)` with a quote in front. No points were taken off for this. When we ask for a return value, make sure you put down whatever `STK` would return, which is *without* the quote.



Part B (6 points). Write `gather-hof`, which behaves the same as `gather` but uses no explicit recursion.

Solution:

```
(define (gather-hof sent)
   (accumulate
       (lambda (new so-far)
           (cond ((not (sentence? so-far))
                  (if (equal? new so-far)
                      (se (word new so-far))
                      (se new so-far)) )
                 ((equal? new (first (first so-far)))
                  (se (word new (first so-far)) (bf so-far)) )
                 (else (se new so-far) )) )
       sent) )
```

-2: If you did not take into consideration that `so-far` could be either a word or a sentence. This is similar to the trick we used in the `diagonal` homework with the `position` procedure.

A few of you did a cool trick where the `so-far` variable is always `sentence`d. This guarantees that `so-far` would consistently be a sentence, eliminating the need to check for it being a word or sentence, or the case of taking the `bf` of a word. In this case, an `if` with two cases would be sufficient:

```
…(lambda (new so-far)
     (if (equal? new (first (first (se so-far))) )
         (se (word new (first so-far)) (bf (se so-far)) )
         (se new so-far) ) )
```

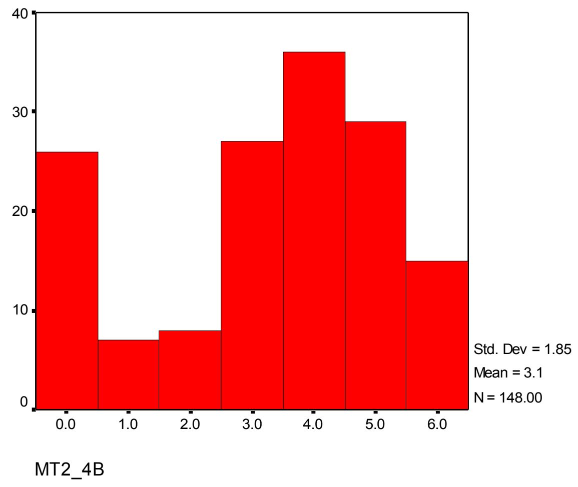
-2: If you did not use `accumulate` correctly.

-1: If the equality check was on only `(first so-far)`. We must use `(first (first so-far))` here. Consider the case of `so-far` being `(bb c d)`: it is just the letter `b` that we want to compare `new` to, not the entire word `bb`. Also, consider the case of `so-far` being `(b c d)`: the letter `b` would still be returned from `(first (first so-far))` because `(first 'b)` is still `b`.

-.5 to -2: If the sentence/return value of `lambda` is not correct. The number of points taken off depended on how far your answer deviated from the correct solution. One point was taken off if the return value was only `(se so-far)`, discarding new if they were not equal. One point was also taken off if `(se (word new (first so-far)))` was returned, forgetting about `(bf so-far)`.

Some of you kept the `cond` (the first two cases) from the given recursion code, and called `accumulate` in the `else` case. This was unnecessary! The problem guarantees that the argument given to `gather` is a sentence of at least two single-character words. No points were taken off for this in this particular problem. However, if you used a `cond`, but did not correctly include the call to `accumulate`, say, put it into `else`, one point was taken off for this.

REMEMBER: `wd` is not the same as `word` A good number of you still make the procedure call `(`**`wd`**` new so-far)`. Make sure you know that `wd` is **not** a procedure. The procedure you want to use is `word`. SPELL IT OUT! If you were to type it into `STK`, you would get an unbound variable error because `wd` does not exist! No points were taken off for this, but we might in the future!


Std. Dev = 1.85
Mean = 3.1
N = 148.00
MT2_4B