```
(ns n-gram.words.good-turing (:require [n-gram.misc.misc-functions :refer :all]
                                       [n-gram.words.file-reader :refer :all]))

(defn generate-counts-of-counts "Generates a map of counts of counts" [input-counts]
  (frequencies (map val input-counts)))

(def generate-counts-of-counts-memo "Memoized generate-counts-of-counts" (memoize
generate-counts-of-counts))

(def counts-of-counts-1 "1-gram counts-of-counts map" (generate-counts-of-counts-memo
counts-1))

(def counts-of-counts-2 "2-gram counts-of-counts map" (generate-counts-of-counts-memo
counts-2))

(def counts-of-counts-3 "3-gram counts-of-counts map" (generate-counts-of-counts-memo
counts-3))

(def counts-of-counts-4 "4-gram counts-of-counts map" (generate-counts-of-counts-memo
counts-4))

(defn r "Returns count of given n-gram" [n-gram n]
        (get-count-memo (resolve (symbol (str "counts-" n))) n-gram))

(def r-memo "Memoized r" (memoize r))

(defn n-r "Returns count of given count" [freq n]
        (if (> freq 0) ((resolve (symbol (str "counts-of-counts-" n))) freq)
        (count (var-get (resolve (symbol (str "counts-" n)))))))

(def n-r-memo "Memoized n-r" (memoize n-r))

 (defn n-r-plus-one "Returns count of frequency plus one" [freq n]
  (if (=  (val (apply max-key val (var-get (resolve (symbol (str "counts-" n)))))) freq)
    ((resolve (symbol (str "counts-of-counts-" n))) freq)
  (get-count-memo (resolve (symbol (str "counts-of-counts-" n))) (inc freq))))

(def n-r-plus-one-memo "Memoized n-r-plus-one" (memoize n-r-plus-one))

(defn g-t "Returns Good-Turing count of given word" [n-gram n]
  (let [freq (if (= (type n-gram) (type "")) (r-memo [n-gram] n) (r-memo n-gram n))]
    (* (+ freq 1) (/ (n-r-plus-one-memo freq n) (n-r-memo freq n)))))

(def g-t-memo "Memoized g-t" (memoize g-t))

(defn g-t-prob "Returns probability of a given n-gram using Good-Turing smoothing" [n-
gram]
  (let [n (if (= (type n-gram) (type "")) (count [n-gram]) (count n-gram))]
```

```
    (if (> n 1)
      (let [g-t-n-minus-1 (g-t-memo (butlast n-gram) (dec n))]
        (if (zero? g-t-n-minus-1)
          "Inf"
          (float (/ (g-t-memo n-gram n) g-t-n-minus-1)) ))
      (float (/ (g-t-memo n-gram n) (count (var-get (resolve (symbol (str "counts-"
n)))))))))))

(def g-t-prob-memo "Memoized g-t-prob" (memoize g-t-prob))
```