Plateau Program – Documentation

Csc 600-01

Keawa Rozet

Code available at: <https://github.com/krozet/Integer-Plot-Function.git>

Implementation

My method for returning the length of the longest sequence of identical numbers involves using a for loop that iterates over an array of numers, counts the frequency of each number, and stores the largest frequency in a variable which is returned by maxlen().

What seperates my method from others, is that it takes into account for the average frequency of numbers and attempts to skip over large portions when able. The average frequency of numbers is calculated by dividing the total number of elements in the array already examined, by how many unique numbers have been presented. When the next unique number is indexed, this snippit of code will attempt to see if it is at the current index plus the average frequency:

temp == array[i + avgLength]

If the new number is located there, then the array will skip to that point and continue it’s evaluation. If the new number is not located there, then it will try again but at half of the average frequency:

temp == array[i + avgLength/2]

If this returns true, it will behave like mentioned above. If false, then the code will proceed through this number index by index until the next unique number is reached.

I decided to use this method so that my average time complexity case would not always be equal to my worst time complexity case. This algorithm works best with number frequencies that are of roughly similar length. This code does not receive a boost of efficiency with low frequency numbers.

Source Code

/\*

Keawa Rozet

CSc 600-01 - Programming Languages

Homework #2

Plateau Program (max sequence length)

\*/

#include <stdio.h>

int maxlen(int \*array, int elements);

int main(int argc, char const \*argv[]) {

int array[] = {1,1,1,1,2,2,2,3,3,4,4,4,4,4,5,6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,8,8,9,9,9,9,10,10,10,10,10,10};

printf("Max length: %d\n", maxlen(array, sizeof(array)/sizeof(int)));

return 0;

}

int maxlen(int \*array, int elements) {

int count = 0; //keeps track how many times each number occurs

int maxCount = 0;

int times = 0;

int numUnique= 1;

int avgLength = 1;

int max = array[0];

int temp = array[0];

for (int i = 0; i < elements; i++) {

if (temp == array[i])

count++;

//the next unique number is found

else {

//set new max number

if (count > maxCount) {

maxCount = count;

max = array[i-1];

}

count = 1;

temp = array[i];

numUnique++;

//increased performance for large arrays with plenty reoccuring numbers

//takes the average length a number occurs

//check if the current number's occurance is equal to the average length or half

//if so, move to that location in the array to save time

avgLength = i/numUnique; //total num of elements so far divided by unique numbers

if ((i + avgLength) <= elements) {

if (temp == array[i + avgLength]) {

i += avgLength;

count += avgLength;

}

else if (temp == array[i + avgLength/2]) {

i += avgLength/2;

count += avgLength/2;

}

}

else if ((i + avgLength/2) <= elements) {

if (temp == array[i + avgLength/2]) {

i += avgLength/2;

count += avgLength/2;

}

}

}

times++;

}

//performance evaluation

printf("Max number: %d\tO(n) = %d\tActual complexity = %d\n", max, elements, times);

return maxCount;

}

Example Build and Execution through Terminal

> g++ -o pl plateau.cpp

> ./pl

Max number: 6 O(n) = 44 Actual complexity = 20

Max length: 11