Iteration vs Recursion 1: Binary Search – Documentation

Csc 600-01

Keawa Rozet

Code available at: <https://github.com/krozet/Iteration-vs-Recursion-1.git>

The Problem

Make a sorted integer array a[i]=i, i=0,…,n-1. Let bs(a,n,x) be a binary search program that returns the index i of array a[0..n-1] where a[i]=x. Obviously, the result is bs(a,n,x)=x, and the binary search function can be tested using the loop

for(j=0; j<K; j++) for(i=0; i<n; i++) if(bs(a,n,i) != i) cout << “\nERROR”;

Select the largest n your software can support and then K so that this loop with an iterative version of bs runs 3 seconds or more. Then measure and compare this run time and the run time of the loop that uses a recursive version of bs. Compare these run times using maximum compiler optimization (release version) and the slowest version (minimum optimization or the debug version). If you use a laptop, make measurements using AC power, and then same measurements using only the battery. What conclusions can you derive from these experiments? Who is faster? Why?

Implementation

I used two very standard methods for my binary search and felt no need to deviate from these methods. In my source code I first call the method initArray() to give me an array or sorted integers to use for my tests. Then I call bsTimeMeasurement(). This method handles finding how long each method will run for under the required conditions.

In all my tests with A/C power the Iteration method proved to be substantially faster than the Recursion method by roughly 20%. While using battery power, however, the performance of the Iteration decreased enough to be outside the margin of error, while the Recursion’s performance stayed relatively the same.

While both methods are within O(log(n)), it’s clear that they do vary enough to warrant using one over the other. A performance gap of roughly 20% is no small feat and I believe this largely has to do with the call stack. Since the recursive methods revolve entirely around the use of the call stack, and iterative methods not nearly as much, the recursive methods will naturally have more overhead dealing with this stack. In this case, the recursive method and iterative method are so similar that the little addition of the overhead from the call stack makes the iterative more efficient.

Source Code

/\*

Keawa Rozet

CSc 600-01 - Programming Languages

Homework #2

Iteration vs Recursion 1

\*/

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <limits>

#define SIZE 89999

#define K 300

typedef std::numeric\_limits< double > dbl;

void initArray(int\* array);

void printArray(int\* array);

int binarySearchIteration(int\* array, int n, int i);

int binarySearchRecursion(int\* array, int left, int n, int i);

void bsTimeMeasurement(int\* array, int n);

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

int array[SIZE];

initArray(array);

bsTimeMeasurement(array, SIZE);

return 0;

}

int binarySearchIteration(int\* array, int n, int i) {

int mid, left = 0;

while (n >= left) {

mid = left + (n-left)/2;

//value is in the middle

if (array[mid] == i)

return mid;

//value is less than the middle

if (array[mid] < i)

left = mid + 1;

//value is greater than the middle

else

n = mid - 1;

}

//element not found

return -1;

}

int binarySearchRecursion(int\* array, int left, int n, int i) {

int mid = left + (n - left)/2;

if (n >= left) {

//value in the middle

if (array[mid] == i)

return mid;

//value greater than middle

if (array[mid] > i)

return binarySearchRecursion(array, left, mid-1, i);

//value less than middle

return binarySearchRecursion(array, mid+1, n, i);

}

//element not found

return -1;

}

void initArray(int\* array) {

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

//ascending sorted array

array[i] = i;

//random sorted array

// array[i] = array[i-1] + (rand() % 2);

}

}

void printArray(int\* array) {

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

printf("%d\n", array[i]);

}

}

void bsTimeMeasurement(int\* array, int n) {

int j, m;

//first test binarySearchIteration

clock\_t begin = clock();

for(j = 0; j < K; j++)

for(m = 0; m < n; m++)

binarySearchIteration(array, n, m);

clock\_t end = clock();

double bsI\_elapsed\_secs = double(end - begin)/CLOCKS\_PER\_SEC;

std::cout.precision(dbl::max\_digits10);

std::cout << "Binary Search Iteration at K= " << K <<": " << std::fixed << bsI\_elapsed\_secs << " seconds." << std::endl;

//second test binarySearchRecursion

clock\_t begin2 = clock();

for(j = 0; j < K; j++)

for(m = 0; m < n; m++)

binarySearchRecursion(array, 0, n, m);

clock\_t end2 = clock();

double bsR\_elapsed\_secs = double(end2 - begin2)/CLOCKS\_PER\_SEC;

std::cout.precision(dbl::max\_digits10);

std::cout << "Binary Search Recursion at K= " << K <<": " << std::fixed << bsR\_elapsed\_secs << " seconds." << std::endl;

}

Example Build and Execution through Terminal

**Using A/C Power:**

> g++ -o bs binarysearch.cpp

> ./bs

Binary Search Iteration at K= 300: 3.07653299999999996 seconds.

Binary Search Recursion at K= 300: 3.88366699999999998 seconds.

**Using Battery Power:**

> g++ -o bs binarysearch.cpp

> ./bs

Binary Search Iteration at K= 300: 3.13250800000000007 seconds.

Binary Search Recursion at K= 300: 3.88151199999999985 seconds.