Fundamental Data Structures Project 1: Performance Measurement

Group Members | *Li Yutze(Yuze)*, *Zhu Yibo*
Date | *Oct 18 2014*

Introduction

This is the first chapter project in Zhejiang University **Fundamental Data Structures** course. In this project we will practise measuring algorithm run times and run time analyses. The first task is to implement two algorithms that print out a list of numbers - the *iteractive algorithm* print out the list one by one and the *recursive algorithms* which divides the list at middle and print the left half, middle element and the right half in sequence. The second task is to evaluate the time and space complexity, as well as in *silico* measuring / comparing the running time of the two algorithms.

Algorithm Specification

We utilized the array model to implement list data type. The iterative print function can be described using folowing pseudocode: (pseudocodes are based on R language)

```
iterative.print <- function(list, length) {
    for (i in length) print(list[i])
}
```

The function will require two parameters, the first is the pointer to the list head, and another is the length of the list.

The recursive function could be as following:

```
recursive.print <- function(list, left.index, right.index) {
    if (not left.index = right.index) {
        mid.index <- (left.index + right.index) integer devide by 2
        mid <- middle of list[mid.index]
        recursive.print(list, left.index, mid.index - 1);
        print(mid);
        recursive.print(list, mid.index + 1, left.index);
    }
}
```

This function will require additional parameters, indicating the left and right index number of current list ranges.

Testing Results

The `project1_interactive.c` done by Yibo is an interactive testing program. I made a non-interactive one `project1_test.c` based on this code which generate a `project1_runtime.csv` data table reporting the runing time.

Following I will use **R** and **R mark down** to visualize the data.

Note: all testing a are done under OS X which lacks the `CLK_TCK` macro defination. I fixed the code by conditionaly define the value as 1, when the macro is absent in `<time.h>`, because in these systems `clock` function returns values in microseconds.

I used the required list lengths: **100, 500, 1000, 2000, 4000, 6000, 8000, 10000**. Generated list contents are incresing integers which equals to `index + 1`. Two algorithms are both tested 500 times and taken the average under each condition. Here we first import the data to **R** and show you the values:

```
run.time <- read.csv('project1_runtime.csv', header = T);
run.time[c(1, 2:5)]; run.time[c(1, 6:9)];
```

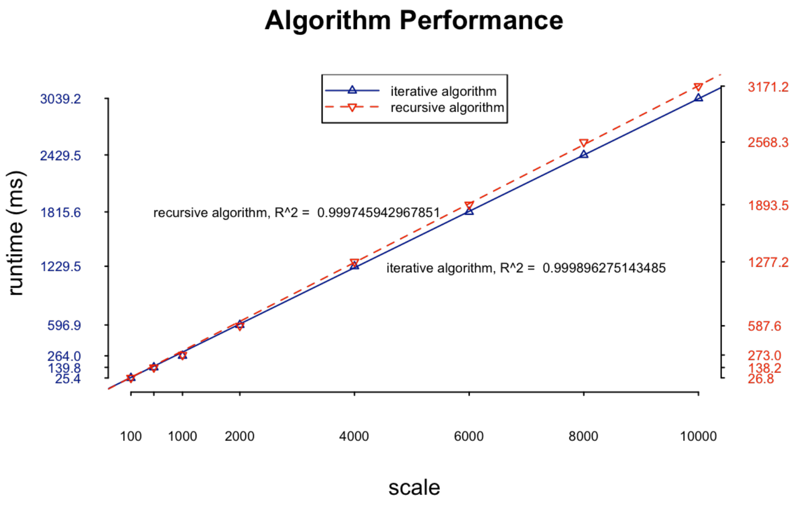
```
##   scale iter.rep iter.tck iter.ttime iter.time
## 1   100      500    12691      12691     25.38
## 2   500      500    69883      69883    139.77
## 3  1000      500   132017     132017    264.03
## 4  2000      500   298453     298453    596.91
## 5  4000      500   614747     614747   1229.49
## 6  6000      500   907816     907816   1815.63
## 7  8000      500  1214752    1214752   2429.50
## 8 10000      500  1519616    1519616   3039.23
```

```
##   scale recr.rep recr.tck recr.ttime recr.time
## 1   100      500    13381      13381     26.76
## 2   500      500    69108      69108    138.22
## 3  1000      500   136524     136524    273.05
## 4  2000      500   293812     293812    587.62
## 5  4000      500   638605     638605   1277.21
## 6  6000      500   946750     946750   1893.50
## 7  8000      500  1284157    1284157   2568.31
## 8 10000      500  1585576    1585576   3171.15
```

The first table show the data for the iterative algorithm and the second table is for the recursive one. Header names `.rep`, `.tck`, `.ttime` and `.time` stand for repeating test times, measured ticks (for Mac OS, one tick is 1 ms), measured total running time and average running time.

The time **vs.** input scale plot is shown followed (generated by corresponding R code): Regression lines (with intercept = 0) are appended and r squared values are specified on the plot.

```
par(mar = c(4, 4, 4, 3) + 0.1);
with(run.time, {
    plot(iter.time ~ scale, type = 'n', ann = F, axes = F,
         ylim = c(0, max(c(iter.time, recr.time)))); # set ploting frame
    lines(iter.time ~ scale, type = 'p', pch = 2, cex = 0.6, col = 'blue4');
    i.lm.fit <- lm(iter.time ~ scale - 1);
    abline(i.lm.fit, col = 'blue4', lty = 1);
        # plot data for iterative algorithm
    lines(recr.time ~ scale, type = 'p', pch = 6, cex = 0.6, col = 'red2');
    r.lm.fit <- lm(recr.time ~ scale - 1);
    abline(r.lm.fit, col = 'red2', lty = 2);
        # plot data for recursive algorithm
    axis(side = 1, at = run.time$scale,
         las = 1, cex.axis = 0.6, tck = -0.01);
    axis(side = 2, at = round(run.time$iter.time,1),
         las = 2, col.axis = 'blue4', cex.axis = 0.6, tck = -0.01);
    axis(side = 4, at = round(run.time$recr.time,1),
         las = 2, col.axis = 'red2', cex.axis = 0.6, tck = -0.01);
    legend('top', legend = c('iterative algorithm', 'recursive algorithm'),
           pch = c(2, 6), lty = c(1, 2), col = c('blue4', 'red2'),cex = 0.6, seg.len = 4
.4);
    text(c(7000, 3000), c(1200, 1800),
         c(paste('iterative algorithm, R^2 = ', summary(i.lm.fit)$r.squared),
           paste('recursive algorithm, R^2 = ', summary(r.lm.fit)$r.squared)),
         cex = 0.6);
    title(main = 'Algorithm Performance', xlab = 'scale', ylab = 'runtime (ms)');
})
```



Analysis and Comments

Both algorithms visited each list elements limited constant times, so the time comlexities for both iterative and recursive algorithms are linear (**T(N) = O(N)**). From the simulation plot, it is not hard to find both algorithm runtime restrict to the regression line perfectly, while the iterative algoritm performed a bit better than the recursive one.

For space complexities, the recursive algorithm would be worse. For each recursive calling of function 3 to 4 integer values are pushed into the stack in the recursive implementation. But for the iterative one, overall only 1 index is required.

Appendix: C code

The implementation for the algorithms was encoded in the `project1_functions.h` file. This piece code was done by **Yibo**.

```
#include <stdlib.h>
#include <stdio.h>

//a list of N integers are delivered to subfunctions by array a[];

void IteractiveMethod(int a[], int N){
    //define the iteractive method for performance measurement
    for(int i=0; i<N; i++) {
        //print the integers one by one through a for-loop
        printf("%d ",a[i]);
    }
};

void RecursiveMethod(int a[], int left, int right) {
    //define the recursive method for performance measurement

    int center = (right + left)/2;

    if (1 == right - left) {
        //The first base case: There are two elements in the subarray.
        printf("%d ", a[left]);     //Sequentially output the two elements to console.
        printf("%d ", a[right]);
    }
    else if (1 == (right - left)%2) {
        //When there are even number of elements in the subarray,
        //there is no middle element.
        RecursiveMethod(a,left,center);
        //Recursively print the first part, and then the second part.
        RecursiveMethod(a,center+1,right);
    }
    else if (0 == right - left) {
        //The second base case: There are one elements in the subarray.
        printf("%d ", a[left]);
    }
    else if (0 == (right - left)%2) {
        //When there are even number of elements in the subarray,
        //there is no middle element.
        RecursiveMethod(a, left, center - 1);
        //Recursively print the first part,
        //followed by printing the integer in the middle, and finally the second part.
        printf("%d ", a[center]);
        RecursiveMethod(a, center+1, right);
    }
};
```

The interactive testing program was encoded in the `project1_interactive.c` file. This piece code was also done by **Yibo**. (in the following code, the indentation is slightly changed)

```
#include <time.h>
#include "project1.h"

#ifndef CLK_TCK
#define CLK_TCK 1
#endif
// tester added: if compiling syst do not present clock tick in time.h, define it as 1

clock_t start, stop; /* clock_t is a built-in type for processor time (ticks) */
double duration;  /* records the run time (seconds) of a function */

int main() {

    int N;      // The number of integers
    int K;      // The number of iterations
    int a[12000];

    printf("Please input the number of integers in your list.\n");
    scanf("%d", &N);

    for(int i=0; i<N; i++){    // initiate the array a[], which holds N elements.
        a[i]=i;
    }

    printf("Please input the number of iterations(K) for iteractive method.\n");
    scanf("%d", &K);

    printf("Output of iteractive method:\n");
    start = clock()  /* records the ticks at the beginning of the function call */

    for(int i=0; i<K; i++){
        //repeat the iteractive function calls for K times to obtain a total run time
        IteractiveMethod(a, N);
    }

    stop = clock();    /* records the ticks at the end of the function call */
    duration = ((double)(stop - start))/CLK_TCK/K;
    printf("\nFor iteractive method, the ticks are %lu ;", stop-start);
    printf("the total time is %lfs; ", duration*K);
    printf("the duration time is %lf.\n", duration);

    printf("Please input the number of iterations(K) for iteractive method.\n");
    scanf("%d", &K);

    printf("Output of recursive method:\n");
    start = clock();     /* records the ticks at the beginning of the function call */

    for(int i=0; i<K; i++){
        ////repeat the recursive function calls for K times to obtain a total run time
        RecursiveMethod(a, 0, N-1);;
    }

    stop = clock();    /* records the ticks at the end of the function call */
    duration = ((double)(stop - start))/CLK_TCK/K;
    printf("\nFor recursive method, the ticks are %lu ;", stop-start);
    printf("the total time is %lfs; ", duration*K);
    printf("the duration time is %lf.\n", duration);

    return 0;
}
```

The program which generate the comma separeted file `project1_runtime.csv` was encoded in the `project1_test.c` file. This piece code was done by **Yutze**. And this .c file is **compiled** to UNIX executable file `project1` by **gcc** under Mac OS X 10.9.

```
#include <time.h>
#include "project1_functions.h"

#ifndef CLK_TCK
#define CLK_TCK 1
#endif
/* For systems lack the define of CLK_TCK, let it = 1, for they count ticks in ms */

int main() {
    FILE *file_p;
    file_p = fopen("./project1_runtime.csv", "wt");
        /* load the outputs into a .csv file */
    fprintf(file_p,
    "scale, iter.rep, iter.tck, iter.ttime, iter.time, recr.rep, recr.tck, recr.ttime, r
ecr.time\n");
        /* write the header of the .csv file */

    /* ================================================================ */

    int list_lengths[] = {100, 500, 1000, 2000, 4000, 6000, 8000, 10000};
    int list_len_index = 0;
        /* lengths of lists */
    int rep_time = 500;
        /* let procedures run 500 times */

    int current_list[10000 + 100];
        /* current list for manupulating */
    int through = 0;
        /* filling current list index */

    clock_t start, stop;

    while (list_len_index - 8) {

        while (through - list_lengths[list_len_index]) {
            current_list[through] = through + 1;
            through++;
        }
            /* filling the current list */

        fprintf(file_p, "%d, ", list_lengths[list_len_index]);

        start = clock();
        for(int i = 0; i < rep_time; i++)
            IteractiveMethod(current_list, list_lengths[list_len_index]);
        stop = clock();
            /* iteractive method  time counted */
        fprintf(file_p, "%d, %lu, %f, %f, ", rep_time, stop - start,
            (float)(stop - start) / CLK_TCK, (float)(stop - start) / CLK_TCK / rep_time)
;
            /* iteractive method data filled */

        start = clock();
        for(int i = 0; i < rep_time; i++)
            RecursiveMethod(current_list, 0, list_lengths[list_len_index] - 1);
        stop = clock();
            /* recursive method time counted */
        fprintf(file_p, "%d, %lu, %f, %f\n",rep_time, stop - start,
            (float)(stop - start) / CLK_TCK, (float)(stop - start) / CLK_TCK / rep_time)
;
            /* recursive method data filled */

        list_len_index++;
    }   /* go through all lengths */

    /* ================================================================ */

    fclose(file_p);
    return 0;
}
```

Declaration

> *We hereby declare that all the work done in this project titled “Title.of.Project” is of our independent effort as a group.*

Duty Assignment

- Programmer: **Zhu Yibo**
- Tester: **Li Yutze**
- Report Writer: **Li Yutze**