CS 242: Data Structures and Algorithms

A Comparison of Four Algorithms for the Maximum Consecutive Subvector Problem

Timing table

The following timings are based on runs done on `onyx`, an HP 735 workstation rated at around 170 MIPS. The programs were compiled with version 2.7.2.1 of the `gcc` compiler with the optimizer flag `-O` enabled. The run-time equations were generated using curve-fitting using the `maple` package on `onyx`.

Algorithm		1	2	3	4
Lines of C code		10	9	17	7
Run time in microsecs		$0.046n^3$	$0.21n^2$	$1.83n\log_{10} n$	$0.67n$
Time to solve a problem of size	100	0.05 s	0.021 s	0.37 ms	0.067 ms
	1000	46 s	0.21 s	5.5 ms	0.67 ms
	10000	12.8^* hrs	21 s	73.2 ms	6.7 ms
	100000	1.45^* yrs	35^* min	920 ms	67 ms
	1000000	1454^* yrs!	58^* hrs	11 s	0.67 s

The values with an asterisk besides them are projected values.

The Test Program

The following test program as well as other related files can be found in the directory `~amit/cs242/lab/examples/maxsubvector`.

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

#define MAXSIZE 1000000
#define LOW     -5.0
#define HIGH    +5.0
#define MAX(x,y) ((x) > (y) ? (x) : (y))

void GenerateRandomVector(int N, float X[])
{
    int I;
    double drand48();

    for (I=1; I<=N; I++) {
        X[I] = drand48()*(HIGH-LOW) + LOW;
    }
}

float MaxSum1(int N, float X[])
{
    float MaxSoFar, Sum;
    int I, L, U;

    MaxSoFar = 0.0;
    for (L=1; L<=N; L++) {
        for (U=L; U<=N; U++) {
            Sum = 0.0;
            for (I=L; I <= U; I++)
                Sum = Sum + X[I];
            MaxSoFar = MAX(MaxSoFar, Sum);
        }
    }
    return (MaxSoFar);
}

float MaxSum2(int N, float X[])
{
    float MaxSoFar, Sum;
    int I, L, U;

    MaxSoFar = 0.0;
    for (L=1; L<=N; L++) {
        Sum = 0.0;
        for (U=L; U<=N; U++) {
            Sum = Sum + X[U];
            MaxSoFar = MAX(MaxSoFar, Sum);
        }
    }
    return (MaxSoFar);
}


float MaxSum3(int L, int U, float X[])
{
    float MaxToLeft, MaxToRight, MaxCrossing;
    float MaxInA, MaxInB, Sum;
    int I, M;

    if (L > U) return 0.0;
    if (L == U) return MAX(0.0, X[L]);

    M = (L+U)/2;
    /* Find max crossing to left */
    Sum = 0.0; MaxToLeft = 0.0;
    for (I=M; I>=L; I--) {
        Sum = Sum + X[I];
        MaxToLeft = MAX(MaxToLeft, Sum);
    }

    /* Find max crossing to right */
    Sum = 0.0; MaxToRight = 0.0;
    for (I=M+1; I<=U; I++) {
        Sum = Sum + X[I];
        MaxToRight = MAX(MaxToRight, Sum);
    }
    MaxCrossing = MaxToLeft + MaxToRight;

    MaxInA = MaxSum3(L,M,X);
    MaxInB = MaxSum3(M+1,U,X);
    return MAX(MAX(MaxCrossing, MaxInA), MaxInB);
}

float MaxSum4(int n, float X[])
{
    float MaxSoFar, MaxEndingHere;
    int I;

    MaxSoFar = 0.0;
    MaxEndingHere = 0.0;
    for (I=1; I<=n; I++) {
        MaxEndingHere = MAX(MaxEndingHere+X[I], 0.0);
        MaxSoFar = MAX(MaxSoFar, MaxEndingHere);
    }
    return (MaxSoFar);
}
```

```
int main(int argc, char *argv[])
{
    int N, choice;
    float X[MAXSIZE];
    float sum;
    float time_before, total_time;
    float report_cpu_time();


    printf("Enter size of vector X: ");
    scanf("%d",&N);
    printf("\n");

    printf("Enter which algorithm to test: ");
    scanf("%d",&choice);
    printf("\n");

    GenerateRandomVector(N,X);

    switch (choice) {
    case 1: time_before = report_cpu_time();
            sum=MaxSum1(N,X);
            total_time = report_cpu_time() - time_before;
            printf(" time used = %f   Maximum Sum = %f \n",total_time, sum);
            break;

    case 2: time_before = report_cpu_time();
            sum=MaxSum2(N,X);
            total_time = report_cpu_time() - time_before;
            printf(" time used = %f   Maximum Sum = %f \n",total_time, sum);
            break;

    case 3: time_before = report_cpu_time();
            sum=MaxSum3(1,N,X);
            total_time = report_cpu_time() - time_before;
            printf(" time used = %f   Maximum Sum = %f \n",total_time, sum);
            break;

    case 4: time_before = report_cpu_time();
            sum=MaxSum4(N,X);
            total_time = report_cpu_time() - time_before;
            printf(" time used = %f   Maximum Sum = %f \n",total_time, sum);
            break;
    default: printf("Unknown option \n");
    }
}
```

The timing function is shown below:

```
#include <stdio.h>
#include <sys/times.h>
#include <unistd.h>
```

```
/*-------------------------------------------------------------------------
clock_t times(struct tms *buffer);

times() fills the structure pointed to by buffer with
time-accounting information.  The structure defined in
<sys/times.h> is as follows:

struct tms {
    clock_t tms_utime;       user time
    clock_t tms_stime;       system time
    clock_t tms_cutime;      user time, children
    clock_t tms_cstime;      system time, children

The time is given in units of 1/CLK_TCK seconds where the
value of CLK_TCK can be determined using the sysconf() function
with the agrgument _SC_CLK_TCK.
-------------------------------------------------------------------------*/

float report_cpu_time()
{
    struct tms buffer;
    clock_t times();
    float cputime;

    times(&buffer);
    cputime = (buffer.tms_utime)/ (float) sysconf(_SC_CLK_TCK);
    return (cputime);
}
```

Maple commands for curve-fitting

```
fit[leastsquare[[x,y], y=a*x^3,{a}]]\
            ([[500,600,700,800],[5.88,10.13,15.69,23.78]]);


fit[leastsquare[[x,y], y=a*x^2, {a}]]\
([[1000,2000,3000,4000],[0.21,0.83,1.89,3.39]]);


fit[leastsquare[[x,y], y=a*x*log[2](x),{a}]]\
([[100000,200000,300000,400000,500000],[0.92,1.93,2.96,4.07,5.17]]);


fit[leastsquare[[x,y], y=a*x, {a}]]\
([[100000,200000,300000,400000,500000],[0.07,0.13,0.20,0.27,0.34]]);
```