寻找自我的博客

编程珠玑第五章 编程小事

2012-08-29 14:36 45人阅读 评论(0) 收藏 举报

1. 原理

脚手架	可以很方便的访问函数。
编码	对于比较难写的函数，可以写出伪代码来构建程序框架，然后翻译为实现的语言
测试	在脚手架中对组件进行测试要在大系统中更容易，更彻底
调试	对个隔离在其脚手架的程序很难调试，但是比起真实环境要好。
计时	进行实验，主要是为了达到我们预期的性能

习题5： 常见的错误就是把二分搜索运用于未排序的数组，而在每次搜索前

检测整个数组是否有序需要进行n-1次额外的比较，能否为该函数添加部分

检测程序，以显著降低的开销

首先 可以利用断言来测试 是否有序：

```
for (i = 0; i < n - 1; ++i)
   assert(a[i] < a[i+1]);
```

然后 我们怎么样修改才能降低开销：

```
int bs(int *a, int b, int e, int v)
{
    int *begin = a + b, *end = a + e, *mid;
    if (!a || b >= e) return -1;
    while (begin < end)
    {
        mid = begin + ((end - begin) >> 1);
        assert(*begin <= *mid && *mid <= *end);
        if (*mid > v) end = mid;
        else if (*mid < v) begin = mid + 1;
        else return mid - a;
    }
    return -1;
}
```

我们assert(*begin <= *mid && *mid <= *end)，确保比较的时候 前后数的顺序。但是遇到多个数就会报错。

另外一种：

```
int bs(int *a, int b, int e, int v)
{
    int *begin = a + b, *end = a + e, *mid, i = b;
    static int *record = 0;
    if (!a || b >= e) return -1;
    if (!record || record != a)
    {
        while (i < e && a[i] < a[i+1])        ++i;
        assert(i == e);
    }

    while (begin < end)
    {
        mid = begin + ((end - begin) >> 1);
        assert(*begin <= *mid && *mid <= *end);
        if (*mid > v) end = mid;
        else if (*mid < v) begin = mid + 1;
        else return mid - a;
    }
    return -1;
}
```

我们加一个static 的 record来记录是否检测过。

9. search. c

```
/* Copyright (C) 1999 Lucent Technologies */

/* From 'Programming Pearls' by Jon Bentley */



/* search.c -- test and time binary and sequential search
   Select one of three modes by editing main() below.
   1.) Probe one function
   2.) Test one function extensively
   3.) Time all functions
                       Input lines:  algnum n numtests
                       Output lines: algnum n numtests clicks nanosecs_per_elem

                       See timedriver for algnum codes
 */

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

#define MAXN 1000000

typedef int DataType;

DataType x[MAXN];
int n;

/* Scaffolding */

int i = -999999;
#define assert(v) { if ((v) == 0) printf("  binarysearch bug %d %d\n", i, n); }

/* Alg 1: From Programming Pearls, Column 4: raw transliteration */

int binarysearch1(DataType t)
{           int l, u, m;
            l = 0;
            u = n-1;
            for (;;) {
                       if (l > u)
                                  return -1;
                       m = (l + u) / 2;
                       if (x[m] < t)
                                  l = m+1;
```

```
                        else if (x[m] == t)
                                return m;
                        else /* x[m] > t */
                                u = m-1;
                }
        }

        /* Alg 2: Make binarysearch1 more c-ish */

        int binarysearch2(DataType t)
        {       int l, u, m;
                l = 0;
                u = n-1;
                while (l <= u) {
                        m = (l + u) / 2;
                        if (x[m] < t)
                                l = m+1;
                        else if (x[m] == t)
                                return m;
                        else /* x[m] > t */
                                u = m-1;
                }
                return -1;
        }

        /* Alg 3: From PP, Col 8 */

        int binarysearch3(DataType t)
        {       int l, u, m;
                l = -1;
                u = n;
                while (l+1 != u) {
                        m = (l + u) / 2;
                        if (x[m] < t)
                                l = m;
                        else
                                u = m;
                }
                if (u >= n || x[u] != t)
                        return -1;
                return u;
        }

        /* Alg 4: From PP, Col 9 */

        int binarysearch4(DataType t)
        {       int l, p;
                if (n != 1000)
                        return binarysearch3(t);
                l = -1;
                if (x[511]   < t) l = 1000 - 512;
                if (x[l+256] < t) l += 256;
                if (x[l+128] < t) l += 128;
                if (x[l+64 ] < t) l += 64;
                if (x[l+32 ] < t) l += 32;
                if (x[l+16 ] < t) l += 16;
                if (x[l+8  ] < t) l += 8;
                if (x[l+4  ] < t) l += 4;
                if (x[l+2  ] < t) l += 2;
                if (x[l+1  ] < t) l += 1;
                p = l+1;
                if (p >= n || x[p] != t)
```

```
                        return -1;
            return p;
}

/* Alg 9: Buggy, from Programming Pearls, Column 5 */

int sorted()
{   int i;
    for (i = 0; i < n-1; i++)
        if (x[i] > x[i+1])
            return 0;
    return 1;
}

int binarysearch9(DataType t)
{           int l, u, m;
/* int oldsize, size = n+1; */
            l = 0;
            u = n-1;
            while (l <= u) {
/* oldsize = size;
size = u - l +1;
assert(size < oldsize); */
                        m = (l + u) / 2;
/* printf("  %d %d %d\n", l, m, u); */
                        if (x[m] < t)
                                    l = m;
                        else if (x[m] > t)
                                    u = m;
                        else {
                                    /* assert(x[m] == t); */
                                    return m;
                        }
            }
            /* assert(x[l] > t && x[u] < t); */
            return -1;
}

/* Alg 21: Simple sequential search */

int seqsearch1(DataType t)
{           int i;
            for (i = 0; i < n; i++)
                        if (x[i] == t)
                                    return i;
            return -1;
}

/* Alg 22: Faster sequential search: Sentinel */

int seqsearch2(DataType t)
{           int i;
            DataType hold = x[n];
            x[n] = t;
            for (i = 0; ; i++)
                        if (x[i] == t)
                                    break;
            x[n] = hold;
            if (i == n)
                        return -1;
            else
                        return i;
```

```
}

/* Alg 23: Faster sequential search: loop unrolling */

int seqsearch3(DataType t)
{           int i;
            DataType hold = x[n];
            x[n] = t;
            for (i = 0; ; i+=8) {
                        if (x[i] == t)   {          break; }
                        if (x[i+1] == t) { i += 1; break; }
                        if (x[i+2] == t) { i += 2; break; }
                        if (x[i+3] == t) { i += 3; break; }
                        if (x[i+4] == t) { i += 4; break; }
                        if (x[i+5] == t) { i += 5; break; }
                        if (x[i+6] == t) { i += 6; break; }
                        if (x[i+7] == t) { i += 7; break; }
            }
            x[n] = hold;
            if (i == n)
                        return -1;
            else
                        return i;
}



/* Scaffolding to probe one algorithm */

void probe1()
{           int i;
            DataType t;
            while (scanf("%d %d", &n, &t) != EOF) {
                        for (i = 0; i < n; i++)
                                    x[i] = 10*i;
                        printf(" %d\n", binarysearch9(t));
            }
}


/* Torture test one algorithm */

#define s seqsearch3
void test(int maxn)
{           int i;
            for (n = 0; n <= maxn; n++) {
                        printf("n=%d\n", n);
                        /* distinct elements (plus one at top) */
                        for (i = 0; i <= n; i++)
                                    x[i] = 10*i;
                        for (i = 0; i < n; i++) {
                                    assert(s(10*i)     ==  i);
                                    assert(s(10*i - 5) == -1);
                        }
                        assert(s(10*n - 5) == -1);
                        assert(s(10*n)     == -1);
                        /* equal elements */
                        for (i = 0; i < n; i++)
                                    x[i] = 10;
                        if (n == 0) {
                                    assert(s(10) == -1);
                        } else {
```

```
                        assert(0 <= s(10) && s(10) < n);
                }
                assert(s(5) == -1);
                assert(s(15) == -1);
        }
}

/* Timing */

int p[MAXN];

void scramble(int n)
{       int i, j;
        DataType t;
        for (i = n-1; i > 0; i--) {
                j = (RAND_MAX*rand() + rand()) % (i + 1);
                t = p[i]; p[i] = p[j]; p[j] = t;
        }
}

void timedriver()
{       int i, algnum, numtests, test, start, clicks;
        while (scanf("%d %d %d", &algnum, &n, &numtests) != EOF) {
                for (i = 0; i < n; i++)
                        x[i] = i;
                for (i = 0; i < n; i++)
                        p[i] = i;
                scramble(n);
                start = clock();
                for (test = 0; test < numtests; test++) {
                        for (i = 0; i < n; i++) {
                                switch (algnum) {
                                case 1:  assert(binarysearch1(p[i]) == p[i
                                case 2:  assert(binarysearch2(p[i]) == p[i
                                case 3:  assert(binarysearch3(p[i]) == p[i
                                case 4:  assert(binarysearch4(p[i]) == p[i
                                case 9:  assert(binarysearch9(p[i]) == p[i
                                case 21: assert(seqsearch1(p[i]) == p[i]);
                                case 22: assert(seqsearch2(p[i]) == p[i]);
                                case 23: assert(seqsearch3(p[i]) == p[i]);

                                }
                        }
                }
                clicks = clock() - start;
                printf("%d\t%d\t%d\t%d\t%g\n",
                        algnum, n, numtests, clicks,
                        1e9*clicks/((float) CLOCKS_PER_SEC*n*numtests));
        }
}

/* Main */

int main()
{       /* probe1(); */
        /* test(25); */
        timedriver();

        return 0;
}
```