

# 会泽百家，安定天下！

主要发表和转载信息安全相关，算法相关文章。

目录视图

摘要视图

RSS 订阅

## 个人资料



mcim

访问: 4967次  
积分: 130  
等级: 2  
排名: 千里之外

原创: 8篇  
转载: 1篇  
译文: 0篇  
评论: 3条

## 文章搜索

## 文章分类

程序设计 (7)  
信息安全 (2)  
密码学 (1)  
软件测试 (2)

## 文章存档

2016年09月 (1)  
2016年08月 (1)  
2015年12月 (1)  
2013年11月 (1)  
2013年10月 (1)

展开

## 阅读排行

大整数乘法的Karatsuba (1595)  
大数素性检测与随机大素 (1074)  
学生信息系统简单实现C (733)  
在时间复杂度为O(n)内将 (433)  
在时间复杂度O(n)内, 实 (425)  
KMP字符串匹配 (189)  
基于odb的dll在xp系统下 (158)  
各种语言单元测试框架汇 (124)

【活动】2017 CSDN博客专栏评选 【5月书讯】流畅的Python，终于等到你！ CSDN日报20170519 ——《思维的局限》  
Kotlin 专场

## 大整数乘法的Karatsuba算法实现

标签: 大整数乘法 快速乘法 Karatsuba 算法

2013-10-14 17:03 1595人阅读 评论(0) 收藏 举报

分类:

信息安全 (1)

版权声明：本文为博主原创文章，未经博主允许不得转载。

两个整数相乘，使用基本算法，时间复杂度为 $O(n^2)$ ，这对于日趋庞大的数据来说是很慢的，目前比较常见的一种大整数的快速算法是 Karatsuba 算法，当然他不是最快的，但是比基本算法要好的多，时间复杂度为 $O(n^{1.59})$ ，在密码运算中相差是很大的。

现在考虑分治算法。取 $m = (n+1)/2$ ，把 $x$ 写成 $10^m \cdot a + b$ 的形式， $y$ 写成 $10^m \cdot c + d$ 的形式，则 $a, b, c, d$ 都是 $m$ 位整数(如果不足 $m$ 位，前面可以补0)。

$$xy = (10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc + ad) + bd$$

递归方程为 $T(n) = 4T(n/2) + O(n)$ ，其中系数4为四次乘法 $ac, bd, bc, ad$ ，附加代价 $n$ 为最后一个return语句的两次高精度加法。方程的解为 $T(n) = O(n^2)$ ，和暴力乘法没有区别。

Anatolii Karatsuba在1962年提出一个改进方法（并由Knuth改进）：用 $ac$ 和 $bd$ 计算 $bc + ad$ ，即：

$$bc + ad = ac + bd - (a - b) * (c - d)$$

这样一来，只需要进行三次递归乘法，即递归方程变为了 $T(n) = 3T(n/2) + O(n)$ ，解为 $T(n) = O(n \log 3) = O(n^{1.585})$ ，比暴力乘法快。

计算整数乘法的最快算法是基于FFT的，它的时间复杂度为 $O(n \log n)$ 。【参考：

<http://blog.csdn.net/jiyanfeng1/article/details/8543846>】

下面是Karatsuba's multiplication algorithm 大整数的快速乘法实现，使用递归。

理论上应该计算速度很快，但是在测试中，内存消耗巨大，速度也较慢，还希望大家帮帮忙，优化一下。和FFT\_MULT相比，相差近100倍！

```
[cpp]
01. /*
02. * 时间: 2013年10月11日16:16:19
03. * 作者: xdc
04. * 功能: Karatsuba's multiplication algorithm 大整数的快速乘法实现
05. * 输入: 两个大整数f&g, 位数不大于2048
06. * 输出: f*g
07. */
08.
09. # include "miracl.h" //调用大整数库miracl.h
10. # include <time.h>
11. # include "math.h"
12. # define TIMES 1
13.
14. big mult_k(big a, big b, long len); //快速乘法
15. long length(big a); //计算数字的长度
16. long max(long a, long b); //求较大值
17. long max_len(big a, big b); //求长度n, n满足是2的次幂
18.
19. /*****主函数*****/
20. int main(void) {
21.     int i=0, j=0, k=0;
```

## 评论排行

基于odb的dll在xp系统下:	(2)
大数素性检测与随机大素	(1)
在时间复杂度O(n)内, 实	(0)
在时间复杂度为O(n)内将	(0)
KMP字符串匹配	(0)
学生信息系统简单实现C	(0)
大整数乘法的Karatsuba	(0)
各种语言单元测试框架汇	(0)
简单的C/C++内存泄漏及	(0)

## 推荐文章

- \* 程序员4月书讯: Angular来了!
- \* CSDN日报20170516 ——《驱动小白和硬件老司机关于硬件那点事儿的一次密谈》
- \* 简单粗暴地入门机器学习
- \* AntShares区块链的节点部署与搭建私有链
- \* 分布式机器学习的集群方案介绍之HPC实现
- \* Android 音频系统: 从AudioTrack 到 AudioFlinger

## 最新评论

- 大数素性检测与随机大素数生成zmd0132: 写的很好, 支持楼主!
- 基于odb的dll在xp系统下动态加载mcim: @zhao4zhong1:一开始是使用动态链接的, 老板觉得dll太多, 然后就这样子了。
- 基于odb的dll在xp系统下动态加载赵4老师: 不鼓励静态链接

```

22. big f, g, res;
23. long len = 0;
24. FILE *fp;
25. clock_t tBegin1, tEnd1;
26. clock_t tBegin2, tEnd2;
27. clock_t tBegin3, tEnd3;
28. miracl *mip = mirsys(4096, 16); //最大4096位,输入输出使用16进制
29.
30. //初始化
31. f = mirvar(0);
32. g = mirvar(0);
33. res = mirvar(0);
34. fp = fopen("input.txt", "r+");
35.
36. mip->IOBASE = 16;
37.
38. cinnum(f, fp);
39. cinnum(g, fp);
40. fclose(fp);
41.
42. printf("f:\n");
43. cotnum(f, stdout);
44. printf("g:\n");
45. cotnum(g, stdout);
46.
47. /*
48. //TIMES次普通大整数乘法
49. tBegin1 = clock();
50. for (i=0; i<TIMES; i++)
51.     multiply(f, g, res);
52. cotnum(res, stdout);
53. tEnd1 = clock();
54. */
55.
56. //TIMES次miracl FFT大整数乘法
57. tBegin2 = clock();
58. for (j=0; j<TIMES; j++)
59.     fft_mult(f, g, res);
60. printf("FFT_MULT:\n");
61. cotnum(res, stdout);
62. tEnd2 = clock();
63.
64.
65. //TIMES次自写大整数乘法
66. tBegin3 = clock();
67.
68. len = max_len(f, g);
69. printf("length: %ld\n", len);
70.
71. for (j=0; j<TIMES; j++)
72.     copy(mult_k(f, g, len), res);
73. tEnd3 = clock();
74.
75. //输出
76. printf("XDC_MULT:\n");
77. cotnum(res, stdout);
78.
79. //释放内存
80. mirkill(f);
81. mirkill(g);
82. mirkill(res);
83. mirexit();
84.
85. //printf("\n\n进行%d次%ld比特的普通大整数乘法运算所消耗的时间
为:%ld ms\n\n", TIMES, len, tEnd1 - tBegin1);
86. printf("\n\n进行%d次%ld比特的miracl FFT大整数乘法运算所消耗的时间
为:%ld ms\n\n", TIMES, len, tEnd2 - tBegin2);
87. printf("\n\n进行%d次%ld比特的自写大整数乘法运算所消耗的时间
为:%ld ms\n\n", TIMES, len, tEnd3 - tBegin3);
88.
89. return (0);
90. }
91.
92. //计算最大长度
93. long max_len(big a, big b) {
94.     long len = 0, n = 1; //保存数字的长度,二进制
95.     len = max(length(a), length(b)); //计算数字长度,取较大值
96.
97. //len变为2的方幂

```

```

98.     while ((pow(2, n) - len) < 0) {
99.         n++;
100.    }
101.
102.    len = (long)pow(2, n);
103.
104.    return (len);
105. }
106.
107. /*****K氏乘法
******/
108. big mult_k(big a, big b, long len) {
109.     big res, tmp_0;           //保存结果
110.     big a1, b1, a0, b0, power_2; //保存分解的因子
111.     big a0_b0, a1_b1;
112.     long m = 0;               //保存数字的长度
113.
114.     //初始化
115.     res = mirvar(0);
116.     a1 = mirvar(0);
117.     b1 = mirvar(0);
118.     a0 = mirvar(0);
119.     b0 = mirvar(0);
120.     power_2 = mirvar(0);
121.     tmp_0 = mirvar(0);
122.     a0_b0 = mirvar(0);
123.     a1_b1 = mirvar(0);
124.
125.     if (len == 1) {
126.         multiply(a, b, res);
127.
128.         //释放内存, 内存泄露
129.         mirkill(b1);
130.         mirkill(b0);
131.         mirkill(a1);
132.         mirkill(a0);
133.         mirkill(tmp_0);
134.         mirkill(power_2);
135.         mirkill(a0_b0);
136.         mirkill(a1_b1);
137.
138.         return (res);
139.     }
140.     else
141.     {
142.         m = len / 2;
143.         //优化,减少中间变量可以减少内存消耗
144.         /*分解 a & b*/
145.         sftbit(a, (-1)*m, a1);           //移位, 相当于除2的m次方
146.         sftbit(a1, m, power_2);
147.         negify(power_2, power_2);
148.         add(a, power_2, a0);
149.
150.         sftbit(b, (-1)*m, b1);           //移位, 相当于除2的m次方
151.         sftbit(b1, m, power_2);
152.         negify(power_2, power_2);
153.         add(b, power_2, b0);
154.
155.         copy(mult_k(a1, b1, m), a1_b1);
156.         copy(mult_k(a0, b0, m), a0_b0);
157.
158.         add(a0, a1, a0);                 //计算a0+a1
159.         add(b0, b1, b0);                 //计算b0+b1
160.         copy(mult_k(a0, b0, m), a0);
161.
162.         //计算返回值
163.         sftbit(a1_b1, len, tmp_0);       //移位, 相当于乘2的len次方
164.         add(tmp_0, a0_b0, tmp_0);
165.         //求负值
166.         negify(a0_b0, a0_b0);
167.         negify(a1_b1, a1_b1);           //取负值
168.         add(a0_b0, a1_b1, a0_b0);       //负值相加
169.         add(a0_b0, a0, a0_b0);
170.         sftbit(a0_b0, m, a0_b0);
171.         add(a0_b0, tmp_0, res);
172.     }
173.
174.     //释放内存
175.     mirkill(b1);

```

```

176.     mirkill(b0);
177.     mirkill(a1);
178.     mirkill(a0);
179.     mirkill(tmp_0);
180.     mirkill(power_2);
181.     mirkill(a0_b0);
182.     mirkill(a1_b1);
183.
184.     return (res);
185. }
186.
187. /*****计算整数的长度*****/
188. long length(big a) {
189.     long i = 0;
190.     big tmp;
191.     tmp = mirvar(0);
192.
193.     expb2(i, tmp);
194.     while (compare(tmp, a) == -1) {
195.         i++;
196.         expb2(i, tmp);
197.     }
198.
199.     mirkill(tmp);
200.     return (i);
201. }
202.
203. //max
204. long max(long a, long b){
205.     return (a >= b ? a : b);
206. }
207.
208. /*
209. 测试环境:
210. -----
211. 操作系统: windows7, x86
212. 硬件: 2G内存, 主频2.67GHz
213. 编译平台: VC6.0企业版
214. -----
215.
216. 在VC6.0中输出测试结果为:
217. -----
218. f:
219. F05085869EF4BA2514D08635E180E138DCD2AAAF1B04C69A4C3A9B612A6FAF9784393B5B49026FEA
220. 2F0E244D84506A7A1D44B8745CE489B0C83668FD83BADEFCA2AECC3D80BA5A3CEB1CB538C25199B0
221. 5E3E3535F3276020F53C8E9D2B518465BD2F6322C1751A00C6EF5186614D9EC955841B2CCFD59882
222. 853E4131233BC2E3D98E5FC36267464CE6947FEE0EC8BC7AA611AD15D68F234BAC62C18C9DEF38B
223. A13550D54EBCD179EA40F377A01066B13E61FF8C9639B2D3A19EC7B8CC58877F7266FDDDC776C56
224. 3D277DB0204C9CE7213D87E76750478531E3B09685629B1B9FEB06E118A5F3E978F8AED1D0C202A5
225. 728021831A5012D43DE53C9CAFF4E1D
226. g:
227. D98E5FC36267464CE6947FEE0EC8BC7AA611AD15D68F234BAC62C18C9DEF38BA13550D54EBCD17
228. 9EA40F377A01066B13E61FF8C9639B2D3A19EC7B8CC58877F7266FDDDC776C563D277DB0204C9CE7
229. 213D87E76750478531E3B09685629B1B9FEB06E118A5F3E978F8AED1D0C202A5728021831A5012D4
230. 3DE53C9CAFF4E1DD98E5FC36267464CE6947FEE0EC8BC7AA611AD15D68F234BAC62C18C9DEF38B
231. A13550D54EBCD179EA40F377A01066B13E61FF8C9639B2D3A19EC7B8CC58877F7266FDDDC776C56
232. 3D277DB0204C9CE7213D87E76750478531E3B09685629B1B9FEB06E118A5F3E978F8AED1D0C202A5
233. 728021831A5012D43DE53C9CAFF4E1D
234. FFT_MULT:
235. CC39E7BE78AC0D920B95B029E2005B1DB44E62400D8C455AE03E02FAD84143091F245DDBAD74DCDF
236. 32B5C19991E06B04E8A06F716943966FB2C53F62129DADC841A24094C453D407FB65C6C7B2140AC
237. 4948CE1DE2C574274D6BE136D8D3E0B6AA7F0E625B50C0663EA1EB84D68D16C9F7C695AE1BF5C545
238. 52B4D8446E0D212DAFB66B4EE69F5BE5E1208D2B655D9CB54F68B239A88C9EBB851C292C2D967BEB
239. 936FB96D29FFC64CE50A0BEDF66020C2D3B6982D1DE645C6603EA7C8E2477C8CE76B1E3B8669D54
240. C9A29B2F50A98D598F0CF44166E0C538817B37177FE46171AF5094D4424DE3EEE0E99FC3BE237320
241. 91623AA160D58B56F7B641549845911438F2963CBDD7FAFC854E3DC24F88E4EC04D93E59150476F2
242. 0CA4B384A4E2042A1FA261A4EF3C7D10A51976C090B3624259A9F42DBCCC1975C9278C66FAC6857E
243. 01BA4D7FA09F91FB795ECD02EF2FBA3B9B086A51F490FE4282898F426CBFAFF11C86F84FD5F4A8D
244. E2514778C62518950E0647B8B61C67BDBA73FD085D41F30557612AC4FE4ACA8AFC360C0CC2BA354
245. 69BEE5F7A041D9137C68D534F8CCB47AB57061372B193A2F2C52C6C2C33AC846E93CB7208224889
246. 15E8E14C7F3FB884B75DBFA5347E31E72F728E4A596AAEF673EF60819B2DBED2F41943137FB87444
247. 0CF6E9131662270540099339DE8EBC3E6744BF884681D06A36A5D6C05B9BAF49
248. length: 2048
249. xdc_mult:
250. CC39E7BE78AC0D920B95B029E2005B1DB44E62400D8C455AE03E02FAD84143091F245DDBAD74DCDF
251. 32B5C19991E06B04E8A06F716943966FB2C53F62129DADC841A24094C453D407FB65C6C7B2140AC
252. 4948CE1DE2C574274D6BE136D8D3E0B6AA7F0E625B50C0663EA1EB84D68D16C9F7C695AE1BF5C545
253. 52B4D8446E0D212DAFB66B4EE69F5BE5E1208D2B655D9CB54F68B239A88C9EBB851C292C2D967BEB
254. 936FB96D29FFC64CE50A0BEDF66020C2D3B6982D1DE645C6603EA7C8E2477C8CE76B1E3B8669D54

```

```
255. C9A29B2F50A98D598F0CF44166E0C538817B37177FE46171AF5094D4424DE3EEE0E99FC3BE237320
256. 91623AA160D5B856F7B641549845911438F2963CBDD7FAFC854E3DC24F88E4EC04D93E59150476F2
257. 0CA4B384A4E2042A1FA261A4EF3C7D10A51976C090B3624259A9F42DBCCC1975C9278C66FAC6857E
258. 01BA4D7FA09F91FB795ECD0D2EF2FBA3B9B086A51F490FE4282898F426CBF4FF11C86F84FD5F4A8D
259. E2514778C625189500E0647B8B61C67BDBA73FD085D41F30557612AC4FE4ACA8AFC360C0CC2BA354
260. 69BEE5F7A041D9137C68D534F8CCB47AB57061372B193A2F2C52C6C2C33AC846E93CB7208224889
261. 15E8E14C7F3FBB84B75DBFA5347E31E72F728E4A596AAEF673EF60819B2DBED2F41943137FBB7444
262. 0CF6E9131662270540099339DE8EBC3E6744BF884681D06A36A5D6C05B9BAF49
263.
264.
265. 进行1次2048比特的miracl FFT大整数乘法运算所消耗的时间为:177 ms
266.
267.
268.
269. 进行1次2048比特的自写大整数乘法运算所消耗的时间为:16001 ms
270.
271. Press any key to continue
272. -----
273. */
```

这是自己第一次使用大数库写代码，请大家多指教。

顶

0

踩

0

上一篇

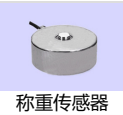
学生信息系统简单实现C语言

下一篇

大数素性检测与随机大素数生成

相关文章推荐

- 算法设计与分析第三章 分治 33 二进制大整数的乘法
  - Karatsuba 乘法
  - karatsuba乘法
  - python实现大整数相乘---格子乘法
  - 大整数乘法-C语言实现
- 大整数乘法C语言实现
  - 大整数乘法C++实现
  - 算法之计算 整数乘法
  - 乘法Karatsuba乘法



参考知识库



.NET知识库  
3896 关注 | 839 收录



操作系统知识库  
6022 关注 | 2210 收录



软件测试知识库  
4714 关注 | 318 收录



算法与数据结构知识库  
16294 关注 | 2320 收录

猜你在找

使用决策树算法对测试数据进行分类实战  
使用决策树算法对测试数据进行分类实战  
C语言系列之 递归算法示例与 Windows 趣味小项目  
C++ 单元测试（GoogleTest）  
Windows Server 2012 DHCP Server 管理

Windows Server 2012 DHCP Server 管理  
《C语言/C++学习指南》加密解密篇（安全相关算法）  
Windows Server 2012 Active Directory 管理  
C语言系列之 数组与算法实战  
C语言系列之 字符串相关算法



[查看评论](#)

暂无评论

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

\* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

#### 核心技术类目

全部主题   Hadoop   AWS   移动游戏   Java   Android   iOS   Swift   智能硬件   Docker   OpenStack  
VPN   Spark   ERP   IE10   Eclipse   CRM   JavaScript   数据库   Ubuntu   NFC   WAP   jQuery  
BI   HTML5   Spring   Apache   .NET   API   HTML   SDK   IIS   Fedora   XML   LBS   Unity  
Splashtop   UML   components   Windows Mobile   Rails   QEMU   KDE   Cassandra   CloudStack  
FTC   coremail   OPhone   CouchBase   云计算   iOS6   Rackspace   Web App   SpringSide   Maemo  
Compuware   大数据   aptech   Perl   Tornado   Ruby   Hibernate   ThinkPHP   HBase   Pure   Solr  
Angular   Cloud Foundry   Redis   Scala   Django   Bootstrap

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#) | [合作伙伴](#) | [论坛反馈](#)

[网站客服](#)   [杂志客服](#)   [微博客服](#)   [webmaster@csdn.net](mailto:webmaster@csdn.net)   400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved 