## 一、加法的伪代码

|  |
| --- |
| **算法1** *AbacusAddition* (*Aug.d*, *Add.d*) |
| **输入**：被加数*Aug.d*，加数*Add.d*  **输出**：加法运算的结果*r*   1. 应用公式（2）初始化*Aug.Upper、Aug.Lower、Add.Upper、Add.Lower* 2. **for** *i* = 15 *to* 14-*Add.len* **do** //循环一位加法 3. **if** *Add.d[i]* > 0 **then** //当前档位的加数不为零 4. *AbacusBitAddition*(*Aug.d*, *Add.d*, *i*) 5. **end if** 6. **end for** 7. 应用公式（1）根据*Aug.Upper、Aug.Lower*计算*r* |

|  |
| --- |
| **算法2** *AbacusBitAddition* (*Aug.d*, *Add.d, i*) |
| **输入**：被加数*Aug.d*，加数*Add.d*，当前计算挡位*i*   1. **if** *Add.d[i]* + *Aug.d[i]* <= 9 **then** 2. **if** *Add.d[i]*<5&&*Aug.d[i]*<5&&*Add.d[i]+Aug.d[i]>=*5**then** //凑五加 3. *Aug.Upper[i]* += 1 4. *Aug.Lower[i]* -= 5 - *Add.d[i]* 5. **else** //直接加 6. *Aug.Upper[i]* += *Add.d[i]*/5 7. *Aug.Lower[i]* += *Add.d[i]*%5 8. **end if** 9. **else** 10. **if** *Aug.d[i]>=*5 *&& Add.d[i]*>5 *&& Aug.Lower[i]*<10-*Add.d[i]* **then** //破五进十加 11. *Aug.Upper[i]* -= 1 12. *Aug.Lower[i]* += 5 - (10-*Add.d[i]*) 13. **else** //进十加 14. *Aug.Upper[i]* -= ⌊(10-*Add.d[i]*)/5⌋ 15. *Aug.Lower[i]* -= (10-*Add.d[i]*)%5 16. **end if** 17. 应用公式（3）生成*Ca* 18. 调用*AbacusBitAddition*(*Aug.d*, *Ca.d, i*+1) //计算进位 19. **end if** 20. 应用公式（1）更新*Aug.d、Add.d* |

## 二、减法的伪代码

|  |
| --- |
| **算法3** *AbacusSubtraction* (*Mi.d*, *Mei.d*) |
| **输入**：被减数*Mi.d*，减数*Mei.d*  **输出**：减法运算的结果*r*   1. 应用公式（2）初始化*Mi.Upper、Mi.Lower、Mei.Upper、Mei.Lower* 2. **for** *i* = 15 *to* 13-*len* **do** //循环一位减法 3. **if** *Mei.d[i]* > 0 **then** //当前档位的减数不为零 4. *AbacusBitSubtraction*(*Mi.d*, *Mei.d*, *i*) 5. **end if** 6. **end for** 7. 应用公式（1）根据*Mi.Upper、Mi.Lower*计算*r* |

|  |
| --- |
| **算法4** *AbacusBitSubtraction* (*Mi.d*, *Mei.d*, *i*) |
| **输入**：被减数*Mi.d*，减数*Mei.d，*当前计算挡位*i*  **输出**：减法运算的结果*r*   1. **if** *Mi.d[i]* >= *Mei.d[i]* **then** 2. **if** *Mi.d[i]>=*5 && *Mei.d[i]<*5 && *Mei.d[i] > Mi.d[i]*-5 **then** //破五减 3. *Mi.Upper[i]* -= 1 4. *Mi.Lower[i]* += *5-Mei.d[i]* 5. **else** //直接减 6. *Mi.Upper[i]* -= *Mei.d[i]*/5 7. *Mi.Lower[i]* -= *Mei.d[i]*%5 8. **end if** 9. **else** 10. **if** *Mei.d[i]*>5 && *Mi.d[i]*<5 && (*Mei.d[i]*-*Mi.d[i]*<=5) **then** //退十还五减 11. *Mi.Upper[i]* += 1 12. *Mi.Lower[i]* -= *Mei.d[i]* - 5 13. **else** //退十减 14. *Mi.Upper[i]* += (10-*Mei.d[i]*) /5 15. *Mi.Lower[i]* += ⌊ (10-*Mei.d[i]*)%5⌋ 16. **end if** 17. 应用公式（3）生成*Bo* 18. 调用*AbacusBitSubtraction*(*Mi.d*, *Bo.d*, *i*) //计算借位 19. **end if** 20. 应用公式（1）更新*Mi.d、Mei.d* |

## 三、乘法的伪代码

|  |
| --- |
| **算法5** *AbacusMultiplication* (*Fac.d*, *Fac.len*, *Mul.d*, *Mul.len*) |
| **输入**：被乘数*Fac.d*，被乘数整数部分的位数*Fac.len*，乘数*Mul.d*，被乘数整数部分的位数*Mul.len*  **输出**：乘法运算的结果*r*   1. 应用公式（2）初始化*Fac.Upper、Fac.Lower、Mul.Upper、Mul.Lower* 2. *R.len* = *Fac.len*+*Mul.len + 2* //定积的位数（+2即加上小数的位数） 3. *OuterPtr* = 15 – *R.len +* 1//1\**n*位乘法的定位指针（+1为位置偏移量） 4. **for** *i* = 1 *to* *Mul.len*+2 **do** 5. **if** *Mul.d[15-(Mul.len+2)+i] ==* 0**then** //乘数当前位为0，跳过当前计算 6. **continue** 7. **end if** 8. *InnerPtr* = *OuterPtr* 9. **for** *j* = 1 *to* *Fac.len*+2 **do** 10. **if** *Fac.d[15-(Fac.len+2)+j] ==* 0 **then** //被乘数当前位为0，跳过当前计算 11. **continue** 12. **end if** 13. *GenerateProduct*(*Prod*, *Fac.d[15-(Fac.len+2)+j]*, *Mul.d[15-(Mul.len+2)+i]*, *InnerPointer*) 14. *AbacusAddition*(*R.d*, *Prod.d*) //将1位乘法的积累加到总积 15. 清空*Prod* 16. *InnerPtr* += 1 17. **end for** 18. *OuterPtr* += 1 19. **end for**   应用公式（1）根据*R.Upper*, *R.Lower*计算*r* |

|  |
| --- |
| **算法6** *GenerateProduct* (*Prod*, *a*, *b*, *InnerPointer*) |
| **输入**：未赋值的1位乘法乘积的算盘形式*prod*，1位被乘数*a*，1位乘数*b*，累加定位指针*InnerPointer*  **输出**：赋值的*Prod*   1. *i* = *InnerPointer* 2. //设置乘积十位的上珠 3. //设置乘积十位的下珠 4. *i* += 1 5. //设置乘积个位的上珠 6. //设置乘积个位的下珠 |

## 四、除法的伪代码

|  |
| --- |
| **算法7** *AbacusDivision* (*Di.d*, *Di.len*, *Div.d*, *Div.len*) |
| **输入**：被除数*Di.d*，被除数整数部分的位数*Di.len*，除数*Div.d*，除数整数部分的位数*Div.len*  **输出**：商*q*，余数*r*   1. 应用公式（2）初始化*Di.Upper、Di.Lower、Div.Upper、Div.Lower* 2. *Quo*.*len* = *Di*.*len* – *Div*.*len* + 1 3. *i* = *DiPtr*, *j* = *DivPtr* 4. **while** *i*<=15&&*j*<=15 *do* //定位 5. **if** *Di.d[i]*<*Div.d[i]* **then** 6. *Quo.len* = *Di.len* – *Div.len* 7. **break** 8. **else** 9. **if** *Di.d[i]*==*Div.d[i]* **then** 10. *i*+=1, *j*+=1 11. **else** 12. **break** 13. **end if** 14. **end if** 15. **end while** 16. *QuoPtr* = -1 //初始化为-1 17. **do** 18. *type* = 1 19. **while** *i*<=15&&*j*<=15 *do* //够除隔位商，不够除挨位商 20. **if** *Di.d[i]*<*Div.d[i]* **then** 21. *type* = 0 22. **break** 23. **else** 24. **if** *Di.d[i]*==*Div.d[i]* **then** 25. *i*+=1*, j*+=1 26. **else** 27. **break** 28. **end if** 29. **end if** 30. **end while** 31. **if** *Div.len* == 1 **then** //一位除估商 32. **if** *Di.d[DiPtr]* < *Div.d[DivPtr]* **then** 33. *qc* = *Di.d[DiPtr, DiPtr+1]* / *Div.d[DivPtr]* 34. *type* = 0 35. **else** 36. *qc* = *Di.d[DiPtr]* / *Div.d[DivPtr]* 37. **end if** 38. **else** //多位除估商 39. **if** *Di.d[DiPtr]* < *Div.d[DivPtr]* **then** //被首小于除首 40. *type* = 0 41. **if** *Div.d[DivPtr+1]* <= 4 **then** 42. *qc* = *Di.d[DiPtr, DiPtr+1]* / *Div.d[DivPtr]* 43. **else** 44. *qc* = *Di.d[DiPtr, DiPtr+1]* / *(Div.d[DivPtr]+1)* 45. **end** **if** 46. **else if** *Di.d[DiPtr] > Div.d[DivPtr]* //被首大于除首 47. **if** *Div.d[DivPtr+1]* <= 4 **then** 48. *qc* = *Di.d[DiPtr]* / *Div.d[DivPtr]* 49. **else** 50. *qc* = *Di.d[DiPtr]* / *(Div.d[DivPtr]+1)* 51. **end** **if** 52. **else**  //被首等于除首 53. *i* = 0 54. **while** (*DiPtr*<=*DivPtr*)?(*DivPtr*+1+*i*<15):(*DiPtr*+1+*i*<15) && *Di.d[DiPtr+1]*==*Div.d[DivPtr+1]* **do** 55. *i* += 1 56. **end while** 57. **if** *Di.d[DiPtr+1+i]* >= *Div.d[DivPtr+1+i]* **then** 58. *qc* = 1 59. **else** 60. *qc* = 9 61. **end if** 62. **end** **if** 63. **end** **if** 64. *loc* = *DiPtr* – 1 – *type* 65. **if** *QuoPtr* == -1 **then** 66. *QuoPtr* = *loc* 67. **end** **if** 68. *qco* = *loc* – *QuoPtr* + 1 69. *Quo.Upper[loc]* = *qc* / 5 //置商 70. *Quo.Lower[loc]* = *qc* % 5 71. 应用公式（1）更新*Di.d* 72. *rqc* = //计算*rqc* 73. **while** **do** //退商 74. *qc* -= 1 75. *rqc* = //更新*rqc* 76. **end** **while** 77. *Quo.Upper[loc]* = *qc* / 5 //重新置商 78. *Quo.Lower[loc]* = *qc* % 5 79. **for** *i* = *DivPtr* *to* 15 **do** //错位叠减商与除数的乘积 80. *product* = //计算商与除数某一位的乘积 81. **if** *product* == 1 **then** //乘积为零，跳过本次减积 82. *continue* 83. **end** **if** 84. 应用公式（2）根据*product*初始化*Mei.Lower*、*Mei.Upper、Mei.d* 85. **for** *j* = *DiPtr* *to* 15 **do** //减积 86. **if** *Mei.d[j]* != 0 **then** 87. *AbacusBitSubtraction*(*Di.d*, *Mei.d*, *j*) 88. **end** **if** 89. **end** **for** 90. 应用公式（1）更新*Di.d* 91. **end** **for** 92. 清空*Mei //将Mei.Upper、Mei.Lower中所有元素置0* 93. **while** **do** //补商 94. *qc*+=1 95. *Quo.Upper[loc]* = *qc* / 5 //重新置商 96. *Quo.Lower[loc]* = *qc* % 5 97. *product* = //除数乘以商所在位数 98. 应用公式（2）根据*product*初始化*Mei.Lower*、*Mei.Upper、Mei.d* 99. **for** *i* = *DiPtr* *to* 15 **do** //减除数 100. **if** *Mei.d[i]* != 0 **then** 101. *AbacusBitSubtraction*(*Di*, *Mei*, *i*) 102. **end** **if** 103. **end** **for** 104. 应用公式（1）更新*Di.d* 105. **end** **while** 106. **while** *Di.d[DiPtr]* == 0 **do** //跳过被除数为0的位，同时指针往后移动1 107. *DiPtr* += 1 108. **end** **while** 109. **while**(*Di.d* > 0 && (*qco*-*Quo.len*) < 2) //当余数不为0或商未达到两位小数时，继续计算下一位商 110. *r* = *Di.d* 111. *q* = 0 112. **for** *i* = *QuoPtr* *to* *QuoPtr* + *Quo.len + 2*, *j* = *Quo.len* - 1 *to* - 2 **do** //从*Quo*中计算商 113. *q* += 114. **end for** |

## 五、开平方算法的伪代码

|  |
| --- |
| **算法8** *AbacusRadication* (*Rad.d*, *Rad.len*) |
| **输入**：被开方数*Rad.d*，被开方数整数部分的位数*Rad.len*  **输出：**根*r*   1. 应用公式（2）初始化*Rad.Upper*，*Rad.Lower* 2. *sum* = 0 //初始化前根之和为0 3. **if** *Rad.len*%2 == 0 **then** //被开方数的位数为偶 4. *Root.len* = *Rad.len*/2 5. *sec* = *Rad.d*[14-*Rad.len*, 14-*Rad.len*+1] //取被开方数最后一节的数 6. **else** //被开方数的位数为奇 7. *Root.len* = *Rad.len*/2 + 1 8. *sec* = *Rad.d*[14-*Rad.len*] //取被开方数最后一节的数 9. **end** **if** 10. *RadPtr* = 14 - *Rad.len* //初始化被开方数的非零位指针 11. *RootPtr* = 14 - *Root.len* //初始化当前所求根的挡位指针 12. *cr* = //估首根 13. *Root.Upper[RootPtr]* = *cr*/5 14. *Root.Lower[RootPtr]* = *cr*%5 15. *rcr* = //计算带计数单位的当前根 16. *sum* += *rcr* //累加前根之和 17. *Tmp1.d* = //计算首根平方 18. 调用*AbacusSubtraction*(*Rad.d*,*Tmp1.d*) //减首根平方 19. *RootPtr* += 1 //已计算完毕首根，将根指针移至下一挡位 20. 清空*Tmp1* 21. *Tmp1.d* = 0.5 22. *Tmp1*.*len* = 0 23. 应用公式（2）初始化*Tmp1.Upper*、*Tmp1.Lower* 24. **while** *Rad.d[RadPtr]* == 0 **do** //跳过被开方数为0的位，同时指针往后移动1 25. *RadPtr* += 1 26. **if** *Rad.len* != 0**then** 27. *Rad.len* -= 1 28. **end if** 29. **end** **while** 30. 调用*AbacusMultiplication*(*Rad.d*, *Rad.len*, *Tmp1.d*, *Tmp1*.*len*) //余数减半（使用乘法实现） 31. 将乘法结果*R*赋值给*Rad* 32. 应用公式（1）更新*Rad.d* 33. **while**(*Rad.d* != 0 || *RootPtr* < 16) **do** //循环计算剩余根直到余数为0或开方达到预设精度 34. 清空*Tmp1* 35. *tmp = Tmp1.d* = *sum* 36. *Tmp1.len = 1* 37. **while** (*tmp*=*tmp*/10)>1 **do** //计算*sum*的位数 38. *Tmp1.len* += 1 39. **end** **while** 40. 应用公式（2）初始化*Tmp1.Upper*、*Tmp1.Lower* 41. *Tmp2* = *Rad* 42. **while** *Rad.d[RadPtr]* == 0 **do** ////更新*Rad.len* 43. *RadPtr* += 1 44. **if** *Rad.len* !=0**then** 45. *Rad.len* -= 1 46. **end if** 47. **end** **while** 48. 应用改进的算法7以*Tmp2.d*除以*Tmp1.d*，并将算法7的输出*qc*赋值给*cr* //估当前根 49. *rcr* = 50. *Root.Upper*[*RootPtr*] = *cr*/5 51. *Root.Lower*[*RootPtr*] = *cr*%5 52. *product* = 53. 清空*Tmp1* 54. *Tmp1.d* = *product* 55. 应用公式（2）初始化*Tmp1.Upper*、*Tmp1.Lower* 56. 调用*AbacusSubtraction*(*Rad.d*, *Tmp1.d*) //余数减去法数 57. **while** *Rad.d[RadPtr]* == 0 **do** ////更新*Rad.len* 58. *RadPtr* += 1 59. **if** *Rad.len* !=0**then** 60. *Rad.len* -= 1 61. **end if** 62. **end** **while** 63. 清空*Tmp1* 64. *Tmp1.d* = //计算当前根平方的一半 65. 调用*AbacusSubtraction*(*Rad.d*,*Tmp1.d*) //减当前根平方的一半 66. *sum* += *rcr* 67. *RootPtr* += 1 //已计算完毕首根，将根指针移至下一挡位 68. **while** *Rad.d[RadPtr]* == 0 **do** //更新*Rad.len* 69. *RadPtr* += 1 70. **if** *Rad.len* !=0**then** 71. *Rad.len* -= 1 72. **end if** 73. **end** **while** 74. 应用公式（1）更新*Rad.d* 75. **end** **while** 76. 应用公式（1）更新*Root.d* 77. *r* = *Root.d* |