ИНФОРМАТИЧНИ ЗАДАЧИ ВЪРХУ МАТЕМАТИЧЕСКИ РЕБУСИ... ИЛИ ЕМПИРИЧЕН ПОДХОД ЗА ОЦЕНКА НА ВРЕМЕТО ЗА ИЗПЪЛНЕНИЕ НА ПРОГРАМИ

Павел Азълов

Резюме. Основен акцент в статията е пресмятане на времето за изпълнение на програми. С примери от математически ребуси се илюстрира как времето за изпълнение на една програма зависи най-вече от конкретния алгоритъм. Описани са три алгоритъма, които решават един и същ математически ребус. Приведени са резултатите от пресметнатите времена за изпълнение на програмите, написани по тези алгоритми в табличен и графичен формат. В края на статията са представени и няколко идеи за развитие на описания проект.

Keywords: program execution time, solving math puzzles, emperical approach

Ребусите са развлекателни задачи, които много хора приемат като интелектуално предизвикателство. Срещат се както в печатни издания, вестници, списания и книги, така и в конкурси и състезания за ученици. Разнообразието им е твърде голямо и се проявява не само в структурата на данните и на съответните отговори, но също така и в областта на знанията, необходими за решаването им. Една голяма част от ребусите са в областта на математиката и е естествено да ги наричаме *математически ребуси*. Само като един пример ще спомена за масово решаваните математически ребуси, познати с името Sudoku puzzles.

За някои ребуси съществуват алгоритми за решаването им. За ребусите, за които не се познават алгоритми и дори и най-общи идеи за решаването им, като основна възможност остава интуицията, която би могла да подскаже идея за решаване, базирана на конкретните данни на ребуса или на пълното изчерпване на всички варианти до намиране на решение.

В тази стаития е представен един конкретен вид математически ребуси, за които не са известни алгоритми, практически приложими за решаването им „на ръка“. Този избор не е случаен, защото целта на статията е да се разгледа един подход за решаване на клас от забавни математически задачи чрез съставяне на алгоритми, които се изпълняват от компютри, а също така и да се посочат методи за анализ на програмните решения на тези задачи. И не на последно място ще покажем, че за някои ребуси може да съществуват толкова много решения, които наистина да не са по възможностите да бъдат намерени само с помощта на лист хартия и молив.

1. Един специален клас от математически ребуси

В тази статия представяме един специален клас от математически ребуси. За разлика от математическия анализ, който традиционно присъства при решаването на математически задачи и в частност при математическите ребуси, тук подходът е различен. На математическия ребус ще гледаме като на информационен обект, който може да бъде „построен“ и „решен“ като математическа задача, но това да става автоматично с помощта на компютър. Това означава разработване на съответни алгоритми и компютърни програми.

Общият вид на ребусите, които предстои да разгледаме, се вижда от примерния ребус на фиг. 1. На фиг. 2 е дадено едно негово решение. Да поясним някои от особеностите в структурата на ребуса.

Ребус от ред 3 е квадратна таблица с 5 × 5 клетки и още 6 числа, които ще наричаме *коефициенти на ребуса*. В някои от клетките на таблицата са записани аритметичните операции {+, –, ×, /, %}, а другите клетки са празни. Има и трети вид клетки, които са празни, но те са маркирани (запълнени с цвят).

Решаването на ребуса се свежда до намирането на елементите на квадратна матрица от 9 ненулеви десетични цифри, които, записани в маркираните клетки на ребуса, определят изрази, стойностите на които са равни на коефициентите, дадени в края на съответните редове и стълбове с нечетни номера.

	-		x		=	18
x		+		x		
	x		/		=	5
-		-		+		
	+		%		=	3
=		=		=		
26		0		19		

Фиг. 1. Математически ребус

8	-	2	x	3	=	18
x		+		x		
4	x	7	/	5	=	5
-		-		+		
6	+	9	%	4	=	3
=		=		=		
26		0		19		

Фиг. 2. Решение на ребуса от фиг. 1

Клетката в най-горния ляв ъгъл на таблицата е маркирана. Всеки ред с нечетен номер представлява последователност от редуващи се една след друга празна маркирана клетка, следвана от клетка-операция. Тази последователност завършва накрая с маркирана клетка. На същия ред, но извън таблицата, е записано число, което е един от коефициентите на ребуса. Колонките с нечетни номера имат подобна структура и също завършват с коефициент на ребуса.

Да отбележим, че със знаците „/“ и „%“ са означени съответно целочислено деление и остатък от целочислено деление и че всички операции са без приоритет и се извършват последователно отляво надясно. Поради наличието на операцията деление, изискването за десетичните цифри, елементи на матрицата, да са само ненулеви числа, е съществено и винаги ще предполагаме, че то е изпълнено без изрично да се споменава за него.

Да разгледаме примера от фиг.2. Вижда се, че:

4 x 7 / 5 = 5 (*Втори ред*)

6 + 9 % 4 = 3 (*Трети ред*)

8 x 4 - 6 = 26 (*Първи стълб*)

Има още един въпрос, отнасящ се до множеството от операции в този вид ребуси. Тъй като в хода на пресмятане на един израз като междинен резултат може да се получи отрицателно число, то добре е да се определи смисълът на операциите „/“ и „%“ в случаите, когато един или и двата аргумента са отрицателни числа. Това се налага, понеже в различните езици за програмиране тези операции са определени по различен начин. Операцията „%“ (остатък от целочислено деление) в езиците C++ и Java се дефинира чрез израза:

$$a \ \% \ b \ = a - (a/b)*b$$

Примерите от табл. 1 дават пълна представа за използването на операциите целочислено деление и остатък от целочислено деление.

И така, всеки ребус от ред 3 е определен от 6 коефициента, 12 операции и квадратна матрица с 9 ненулви десетични цифри. По-нататък за по-кратко с *MP* ще означаваме произволен математически ребус от ред 3 (*MP – **M**атематически **P**ебус* или *MP – **M**ath **P**uzzle*).

Табл.1. Примери на операциите целочислено делене и остатък от целочислено деление в C++ и Java

a	b	a/b	a%b	a-(a/b)*b
17	5	3	2	2
5	15	0	5	5
0	4	0	0	0
9	-4	-2	1	1
-9	3	-3	0	0
9	-3	-3	0	0
5	7	0	5	5
-7	5	-1	-2	-2
-8	-3	2	-2	-2
-3	-8	0	-3	-3

Да отбележим, че математическа задача *MP* представлява нелинейна система от 6 уравнения и 9 неизвестни от вида:

$$a_1 \oplus a_2 \oplus a_3 = k_1 \quad (1)$$
$$b_1 \oplus b_2 \oplus b_3 = k_2 \quad (2)$$
$$c_1 \oplus c_2 \oplus c_3 = k_3 \quad (3)$$
$$a_1 \oplus b_1 \oplus c_1 = k_4 \quad (4)$$
$$a_2 \oplus b_2 \oplus c_2 = k_5 \quad (5)$$
$$a_3 \oplus b_3 \oplus c_3 = k_6 \quad (6),$$

където $\oplus$ е произволна аритметична операция от множеството $\{+, -, x, /, \%\}$, а неизвестните $\{a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3\}$ са ненулеви десетични цифри.

За решаването на *MP* ще конструираме три алгоритъма и ще напишем съответни програми, с които да се пресмятат всичките решения на ребуса. Като основен език за програмиране ще бъде използван езикът за програмиране C++.

За удобство при формулирането на задачите, разглеждани по-долу, ще въведем още няколко означения. Означаваме с *K* списъка от коефициентите на ребуса, с *O* – списъка от операциите му и с *M* – матрицата от 9-те десетични цифри. Тогава, ако матрицата *M* е решение на ребуса *MP*, ще казваме, че *M* е съвместима с операциите *O* и с коефициентите *K* на ребуса. Символично това ще записваме с уравнението *MP* (*M*, *O*) = *K*.

С така въведения клас от *MP* можем да дефинираме следните три основни информатични задачи:

P1. [*Права задача*] Да се формулира алгоритъм и да се напише съответна програма, която по дадени операции *O* и коефициенти *K* на *MP* от ред 3 намира всички матрици *X* (с 3 реда и 3 стълба от десетични ненулеви цифри), които са решения на *MP*, т.е. такива, за които *MP* (*X*, *O*) = *K*.

P2. [*Обратна задача*] Да се формулира алгоритъм и да се напише съответна програма, която по дадени коефициенти *K* и матрица *M* от 9 ненулеви десетични цифри на *MP* от ред 3 намира всички операции *Y*, съвместими с *M* и *K*, т.е. онези операции, за които е в сила *MP* (*M*, *Y*) = *K*.

P3. [*Анализ на времето за решаване на MP*] Да се пресметне времето за изпълнение на програмите, решаващи задачи P1 и P2.

2. Алгоритми и програми за решаване на *MP*

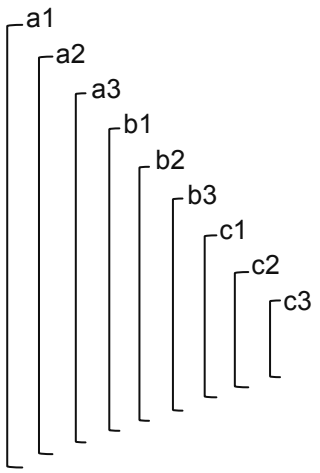
2.1. Алгоритми за решаване на задача P1

Алгоритъм A [*Девет вложени цикъла*]

Системата с уравнения от по-горе подсказва идеята, че стойността на всяко неизвестно може да се получи като стойност на управляващата променлива *x* на цикъл от вида:

```
for(int x = 1; x < 10; x++)
```

Ако се конструират девет вложени цикъла с управляващи променливи a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3 (фиг. 3) , тогава в тялото на най-вътрешния цикъл ще се генерират всички възможни 9-торки от цифрите 1, 2, ..., 9. Техният брой е $9^9 = 387420489$. Може да се провери кои от тях са решения на уравненията (1) – (6). Тази първа идея може съществено да се подобри. Още в тялото на третия цикъл с управляваща променлива a_3 може да се тества дали тройката $< a_1, a_2, a_3 >$ е решение на уравнение (1). Само в случай, че тази тройка е решение, се продължава с изпълнението на следващия цикъл с управляваща променлива b_1. Този подход ще се приложи и за следващите две тройки от цикли $< b_1, b_2, b_3 >$ и $< c_1, c_2, c_3 >$, с които ще се тества дали те са решения съответно на уравненията (2) и (3). Това означава, че тялото на най-вътрешния цикъл ще се изпълнява, когато стойностите на деветте управляващи променливи са решения на първите три уравнения. Сега идва ред да се провери дали тези девет цифри са решения и на следващите три уравнеия (4), (5) и (6).



Фиг. 3. Девет вложени цикъла

Алгоритъм B [*Декомпозиция на 9-значно число на отделни цифри*]

Генерирането на всички възможни 9-орки от цифрите 1, 2, …, 9 може да се извърши от декомпозирането на отделните цифри на целите числа в интервала [111111111, 999999999]. Понеже тези числа са част от числата на типа данни int (INT_MAX = 2147483647), те могат да се генерират само с един цикъл от вида:

```
for(int x = 111111111; x < 1000000000; x++)
```

Всяко от числата *x* ще се декомпозира до съставните си цифри, съхранени в едномерен масив. Например това може да се извърши със следната функция:

```
void int2arr(int arr[], int x)
{
   for (int i = 1; i < 10; i++)
   {
     arr[9 - i] = x%10;
     x = x/10;
   }
}
```

В случай, че някоя от цифрите е нула, текущото изпълнение на тялото на цикъла се прекъсва и се продължава със следващото число. В тялото на цикъла се извършва проверка дали първите три цифри удовлетворяват уравнението (1). В случай, че те не са решение на уравнението, тялото на цикъла също се прекъсва и се продължава със следващата стойност на управляващата променлива на цикъла. Ако тройката цифри е решение, тогава се проверява дали следващите три цифри са решение на уравнение (2). Следвайки тази процедура, ще се търсят онези 9-значни числа, чиито цифри са решения на всичките шест уравнения.

Алгоритъм C [*Подобрен вариант на алгоритъм A*]

Генерирането на девет цифри, измежду които да се търси решение на ребуса, може да се сведе до генерирането само на три цифри. След това тези три цифри се тестват дали удовлетворяват всяко от уравненията (1), (2) и (3) поотделно (виж по-долу функцията `rowSol`). Да означим с A, B и C три едномерни масива, в които са съхранени трицифрени числа, чиито цифри са решения съответно на първото, второто и третото уравнение. Например за ребуса от фиг. 1 числата в тези три множества са следните:

A = { 319, 416, 429, 526, 539, 636, 649, 713, 746, 759, **823**, 856, 869, 933, 966, 979 }
B = { 151, 252, 283, 353, 374, 395, 443, 454, **475**, 486, 497, 511, 522, 533, 544, 555, 566, 576, 577, 587, 588, 598, 599, 656, 667, 678, 689, 734, 745, 756, 757, 768, 779, 823, 846, 857, 858, 869, 935, 947, 958, 959}
C = { 124, 125, 126, 127, 128, 129, 164, 175, 186, 197, 214, 215, 216, 217, 218, 219, 254, 265, 276, 287, 294, 298, 344, 355, 366, 377, 384, 388, 399, 434, 445, 456, 467, 474, 478, 489, 495, 524, 535, 546, 557, 564, 568, 579, 585, 614, 625, 636, 647, 654, 658, 669, 675, **694**, 696, 715, 726, 737, 744, 748, 759, 765, 784, 786, 816, 827, 834, 838, 849, 855, 874, 876, 897, 917, 924, 928, 939, 945, 964, 966, 987, 995 }

Понеже решенията на следващите три уравнения (4), (5) и (6) са цифри, които се получават съответно от цифрите на числата от множествата A, B и C, то решението на ребуса се свежда до намиране на всички тройки числа < *a*, *b*, *c* > A x B x C, на които първите цифри са решения на уравнението (4), вторите им цифри са решения на уравнението (5), а последните им цифри са решения на уравнението (6). Връщайки се отново към примерния ребус, може да се забележи, че цифрите на числата 823 A, 475 B и 694 C са решения на ребуса.

2.2. Програмна реализация на алгоритъм *C*

По-долу следват описанията на константите, структурите от данни и дефинициите на функциите, които са основни за програмата. Кодът е пояснен със съответни коментари.

Константи и прототипи на функциите

```
const int N = 3;                    // Ред на ребуса
const int NROP = 2*N*(N-1);         // Брой операции в ребуса
const int NR = 9*9*9 + 1;           // Макс. брой решения в у-ние
                                       9x9x9=729

                                    // Прототипи на функции
int solution (int[]);               // Пресмята всички решения на МР
void rowSol (int [][NR]);           // Пресмята решенията на у-ния (1-
3)
const char op[NROP] = {             // Операции в ребуса
                        '-', '*',   // Първи ред
                        '*', '/',   // Втори ред
                        '+', '%',   // Трети ред
                        '*', '-',   // Първи стълб
                        '+', '-',   // Втори стълб
                         '*', '+'   // Трети стълб
                      };

const int coef[NRCOEF] = {18, 5, 3, 26, 0, 19}; // Константи на ребуса
```

Дефиниция на две от основните функции на програмата

```
int solution(int final_sol[])       // Пресмята всички решения на МР
{
  int rSol[N][NR] = {0};            // Решения на отделни уравнения
  rowSol(rSol);                     // Първият елемент на всеки ред от
                                    // rSol съдържа броя на решенията
  int counter = 0;                  // Брой на решенията на ребуса

  for (int i = 1; i <= rSol[0][0]; i++)
   for (int j = 1; j <= rSol[1][0]; j++)
    for (int k = 1; k <= rSol[2][0]; k++)
    {                               // Проверка дали решенията на
                                    // уравненията (1),(2) и (3) са
      int ijk;                      // и решения на (4), (5) и (6)
```

```
      if(!calcExpr (rSol[0][i]/100, op[6],
                    rSol[1][j]/100, op[7],
                    rSol[2][k]/100, ijk)) continue;
      if (coef[3] != ijk) continue; // coef[3] - коефициент в 1-ви стълб

      if (!calcExpr (rSol[0][i]/10%10, op[8],
                     rSol[1][j]/10%10, op[9],
                     rSol[2][k]/10%10, ijk)) continue;
      if (coef[4] != ijk) continue; // coef[4] - коефициент във 2-ри стълб

      if (!calcExpr (rSol[0][i]%10, op[10],
                     rSol[1][j]%10, op[11],
                     rSol[2][k]%10, ijk)) continue;
      if (coef[5] != ijk) continue; // coef[5] - коефициент в 3-ти стълб

      counter++;                    // Намерено е ново решение!!!
      final_sol[0] = rSol[0][i];    // Решение на у-нието (1)
      final_sol[1] = rSol[1][j];    // Решение на у-нието (2)
      final_sol[2] = rSol[2][k];    // Решение на у-нието (3)
      printFinalSol (counter, final_sol); // Печат на решението като матрица
     }
  return counter;
}

void rowSol(int rSol[][NR])         // Пресмята решенията на у-нията
{                                   // (1),(2) и (3) и ги записва в
  for (int r = 0; r < N; r++)       // отделните редове на rSol
  {                                 //  r = 0,1,2 --> (у-ния (1),(2) и (3))
    int counter = 0;                // Брой на решенията на у-нието r
    for (int i = 1; i < 10; i++)
     for (int j = 1; j < 10; j++)
      for (int k = 1; k < 10; k++)
      {                             // Тройката цифри <i,j,k>
        int ijk;                    // е кандидат за решение
        if (!calcExpr (i, op[2*r], j, op[2*r+1], k, ijk)) continue;
        if(coef[r] == ijk)          // Проверка дали <i,j,k> е
        {                           // решение на уравнение r
          counter++;
         rSol[r][counter] = 100*i + 10*j + k;// Пакетиране на <i,j,k> в
        }                           // едно число трицифрено число
      }
```

```
      rSol[r][0] = counter;         // Брой решения на уравнение r
  }
}
```

Резултат от изпълнението на програмата

```
Solution #1
  7  1  3
  5  5  5
  9  6  4
Solution #2
  8  2  3
  4  7  5
  6  9  4
Total number of solutions = 2
Time elapsed = 0.015 seconds
```

Три бележки:

– По-добрата читаемост на програмния код е предпочетена пред по-добрата скорост на изпълнението на програмата.

– Използването на динамични масиви (обекти от класа `vector`) могат да намалят обема на използваната памет от масива `rSol` .

– Програмите са изпълнявани в среда на `Windows 7`, `CPU` 2.50 `GHz` и `Microsoft Visual C++ 2010 Express Edition`.

2.3. Алгоритъм за решаване на задача P2

Да се генерира *MP* означава програмно по случаен начин да се генерират операциите и коефициентите на ребуса, т.е. да се определят елементите на множествата *O* и *K*. Това не е трудна задача. Но направените експерименти подсказват, че много голям процент от така генерираните ребуси нямат решения. Затова е по-добре по случаен начин да се генерират елементите на множествата *M* и *O* и след това да се пресметнат коефициентите на ребуса. По този начин всеки ребус ще има поне едно решение. По-долу следва примерен код, който може да се използва за генериране на множествата *M* и *O*.

```
const int N = 3;                    // Ред на ребуса
const int NROP = 2*N*(N-1);         // Общ брой на операциите в ребуса
const int NRCOEF = 2*N;             // Брой на коефициентите в ребуса
const char OPERATORS[] = „+-*%/”;  // Списък на операциите в ребуса
char op[NROP];
```

```
. . .
for (int i = 0; i < NROP; i++)
  op[i] = OPERATORS[rand()%5];     // Случаен избор на операциия

for (int i = 0; i < N; i++)
{
   m[i] = rand() % 9 + 1;          // Случаен избор на число
                                      от интервала [1,9]
   . . .
}
```

3. Пресмятане на времето за изпълнение на програми

Времето за изпълнение на всяка програма е нейна важна характеристика. И ако този въпрос не винаги се дискутира явно, това е защото най-вероятно изчислителният процес на съответната програма е с ограничен обем от пресмятания и не създава безпокойство от продължителността на изпълнение на програмата или защото не винаги е лесно да се даде ясен и точен отговор на този въпрос. Не рядко проблемът за скоростта на изпълнение на една програма е интересен и при съпоставянето ѝ със скоростта на други програми, които решават същата задача. Такъв е случаят с трите програми, написани по алгоритмите A, B и C. Те дават идентичен резултат, но времето за изпълнение на всяка от тях е различно. Можем ли отнапред да кажем коя от тях изисква най-малко време за изпълнение и защо? Можем ли да подобрим времето им за изпълнение и как да извършим това? По същността на тези въпроси има теория, която в много случаи помага добре да се опише поведението на функцията, представляваща времето за изпълнение на програмата, когато размерът на данните нараства неограничено. В тази статия ще „заобиколим“ теорията и ще приложим директни методи за пресмятане на времето за изпълнение на произволна програма.

3.1. Как да пресметнем времето за изпълнение на една програма?

Пресмятането на точното време за изпълнение на дадена програма или на част от нея се извършва като се засече времето t_0 на нейното начало и времето t_1 на завършването ѝ и след това се пресметне разликата $t_1 - t_0$. Определянето на тези времена изисква използването на съответни библиотеки в конкретните среди за програмиране. Накратко това ще преставим при пресмятане на времето за изпълние на програмен код на езиците C++ и Java.

Пресмятане на времето за изпълнение на програми на езика C++

За целта е необходима библиотеката `ctime`[1]. От нея се използват типът данни `clock_t`, функцията `clock()` и константата `CLOCKS_PER_SEC`, с която измереното време се трансформира в секунди.

```
#include <ctime>                    // Необходима библиотека
. . .
clock_t start, end;
. . .
start = clock();                    //start time
  // Програмен код, чието време за изпълнение се пресмята
end = clock();                      //end time
double elapsed = ((double) (end - start)) / CLOCKS_PER_SEC;
cout << „Time elapsed = „ << elapsed << „ seconds“ << endl;
```

Пресмятане на времето за изпълнение на програми на езика Java

Съществуват няколко начина за пресмятане на времето за изпълннеие на една програма на езика Java. Ето един от тях, в който времето се пресмята в милисекунди[2] , е:

```
long start = System.currentTimeMillis();
  // Програмен код, чието време за изпълнение се пресмята
long end = System.currentTimeMillis();
long elapsed = end - start;
System.out.println((“Time elapsed ... „ + elapsed);
```

3.2. Пресмятане на времето за изпълнение на програмите, използващи алгоритмите A, B и C

В табл. 2 са посочени данните на 10 „синтетични“ ребуса, генерирани с програмата, решаваща задача P2. В четвъртата колонка е записан броят на възможните решения на всеки ребус, а в петата – съотношението на мултипликативните към

Табл. 2. Десет „синтетични“ ребуса, определени с операции и коефициенти

MP#	*Операции* (*O*)	*Коефициенти* (*K*)	*Брой решения*	*Операции* (**,/,%*) : (*+,-*)
1	“++%%-**+%*%-”	{17, 1,16, 9, 9, -4}	12	7 : 5
2	“*+%/+/+/%+--”	{10, 0, 0, 8, 2,-14}	246	6 : 7
3	“-%*%*%**++-/”	{2, 0, 3,147, 9, 1}	57	8 : 4
4	“-//+%%-+*-/%”	{-1, 2, 1, 5, 56, 1}	242	7 : 5
5	“-%-//*-%+*++”	{3, 1, 0, 0, 24, 12}	1962	6 : 8
6	“%/%/+/*+**%%”	{0, 0, 1, 63, 6, 4}	828	10: 2
7	“-%+%+--***-/”	{2, 3,10,-28, 72, 0}	33	6 : 6
8	“/*+++-%/*%%+”	{0, 13, 6, 0, 0, 7}	47729	7 : 5
9	“/*+**++//*%*”	{0, 72,61, 1, 0, 5}	327	9 : 3
10	“+---%/-*%+-*”	{2,-10, 0, 27, 6, 0}	187	4 : 8

адитивните операции. Времето за решаване на всеки от 10-те ребуса, съответно на всяка от трите програми, е приведено в табл. 3.

Табл. 3. Времена за изпълнение на трите програми, решаващи ребусите от табл. 1

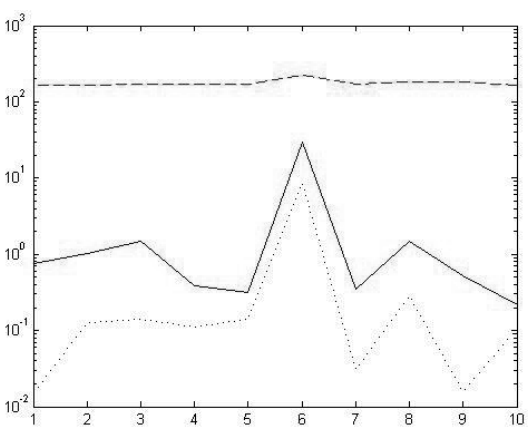
MP#	*Време за изпълнение на програмата, реализираща*		
	алгоритъм A	*алгоритъм* B	*алгоритъм* C
1	0.748	166.038	0.015
2	1.014	164.565	0.125
3	1.498	168.605	0.14
4	0.39	168.246	0.11
5	0.312	167.95	0.14
6	29.858	222.422	8.533
7	0.344	168.271	0.031
8	1.482	181.459	0.281
9	0.514	180.493	0.016
10	0.218	166.374	0.109
Общо	36.378	1754.423	9.5

Анализът на данните от табл. 1 и табл. 2 дава възможност да се направят няколко коментара:

– Определено алгоритъм C изисква най-малко време за намиране на решенията. Но тук следва да се напомни, че при този алгоритъм се използва и допълнителна памет (три масива).

– Най-продължително време за изпълнението е използвано от програмата, реализирана чрез алгоритъм B. Очевидно декомпозицията на 9-значното число на съставните му цифри е основен фактор, който силно влияе на времето за изпълнение.

– И при трите алгоритъма се вижда, че ребус 6 изисква най-много време за решаването му. Защо? Дали конкретните операции влияят толкова силно на времето (вижте последната колонка на табл. 1)? Вижда се, че в ребус 6 мултиплкативните операции



Фиг. 4. Графики на времената за изпълнение на трите програми

са общо 10, а адитивните са само 2. Времето за изпълнение на ребус 5 и при трите алгоритъма е почти най-малкото, а отношението на мултипликативните към адитивните операции е 6 : 8.

Като графики, времената за изпълнение на трите програми с 10-те ребуса са представени на фиг. 4. Те са построени със системата MATLAB. Забележете, че поради големите разлики в стойностите на трите графики за оста Oy е използвана логаритмична скала.

4. Проект в развитие

4.1. Идея за още един алгоритъм за решаване на *MP*

Алгоритъм D [*Вариант на алгоритъм* C]

С алгоритъм C се генерират трите множества A, B и C, всяко от които съдържа 3-цифрени числа, цифрите на които са решения съответно на уравнения (1), (2) и (3). Във функцията `rowSol` те са представени с един двумерен масив.

Нека $< a, b, c > \in A \times B \times C$ е произволна тройка, която е решение на уравнения (1), (2) и (3), като $a = \overline{a_2 a_1 a_0}$, $b = \overline{b_2 b_1 b_0}$ и $c = \overline{c_2 c_1 c_0}$. Да означим с X, Y и Z трите множества, всяко от които съдържа 3-цифрени числа, цифрите на които са решения съответно на уравнения (4), (5) и (6). Тогава задача P1 се свежда до намиране на всички тройки 3-цифрени числа, за които съществуват тройки от 3-цифрени числа $< x, y, z > \in X \times Y \times Z$ със свойството $x = \overline{a_2 b_2 c_2}$, $y = \overline{a_1 b_1 c_1}$ и $z = \overline{a_0 b_0 c_0}$.

Пример: Ако цифрите на числата $a = 246$, $b = 531$ и $c = 973$ са решения на уравненията (1), (2) и (3), а цифрите на числата $x = 259$, $y = 437$ и $z = 613$ са решения на уравненията (4), (5) и (6), то цифрите на a, b и c са решение на *MP*.

Тук ще направим една бележка, отнасяща се до реализацията на този алгоритъм.

Вместо да се съхраняват тройките числа $< x, y, z > \in X \times Y \times Z$, могат да се съхраняват тройките, за които цифрите на числото x` са последователно първите цифри на $x, y z$, цифрите на числото $y$` са съответно вторите им цифри, цифрите на числото z` са съответно третите им цифри. По този начин след сортиране на множествата X, Y и Z при търсене на тройките числа $< a, b, c >$ от $A \times B \times C$ съответно в множествата X, Y и Z може да се приложи двоично търсене.

Въпрос. Ако $T(\alpha)$ е времето за изпълнение на програмата, чиито входни данни са 10-те ребуса (табл. 2), реализирана по алгоритм α, кое от следните неравенства е в сила?

a) $T(D) < T(C) < T(A) < T(B)$
b) $T(C) < T(D) < T(A) < T(B)$
c) $T(C) < T(A) < T(D) < T(B)$
d) $T(C) < T(A) < T(B) < T(D)$

Проверете твърдението си чрез експеримент.

Въпрос. Пресметнете времената за изпълнение на програмите, написани по алгоритмите A, B, C и D, които имат за входни данни ребусите от табл.2 и намират само по едно решение.

Задача. Конструирайте нов алгоритъм и напишете съответна програма. Изпълнете я с данните от табл. 2 и сравнете времето й за изпълнение с това на разгледаните програми.

4.2. Един опит за обобщение

Разгледаните алгоритми трудно могат да бъдат директно обобщени и използвани за решаването на ребуси от ред $n > 3$. Числото 3 е силно „вградено“ в структурата и на трите алгоритъма. Но ако по някакъв начин можем да генерираме всичките n-мерни вектори, чиито компоненти са ненулеви десетични цифри, тогава е възможно да се продължи с идеята от алгоритъм B.

B (Азълов, 1985) и (Азълов и др., 2003) е описан алгоритъм, реализиран с програми на езиците FORTRAN и Pascal, с който се генерират n-мерни вектори, чиито компоненти принадлежат на дадено множество. Алгоритъмът се базира на идеята за конструиране на *нова управляваща структура*, реализираща n вложени цикъла, да я наречем n-цикъл, в тялото на който при всяка поредна итерация се получава следващият n-мерен вектор. N-цикълът може да се онагледи чрез обикновен *for* цикъл, в който управляващата променлива x, началната стойност a, последната стойност b и стъпката на нарастване c са едномерни масиви:

```
const int N = ...;
int x[N], a[N], b[N], c[N];
for (int i = 0; i < N; i++)
{
  a[i] = 1;  // Начална стойност на i-та управляваща променлива
  b[i] = 9;  // Крайна стойност на i-та управляваща променлива
  c[i] = 1;  // Стъпка на нарастване на i-та управляваща променлива
}
. . .
for (x = a; x <= b; x  = x + c)
{
  // В тялото на този цикъл чрез променливата x ще се генерират
  // всички n–мерни вектори с компоненти от интервала [1,9]
}
```

Естествено, реализацията на n-мерния цикъл ще се извърши чрез функция, чийто прототип би могъл да бъде следният:

```
bool nLoops(int n, int x[], int a[], int b[], int c[]);
```

Обръщението към функцията `nLoops` може да се извърши в цикъл:

```
while (nLoops(N, x, a, b, c))
{
  // В тялото на този цикъл се тества дали x = <x1, x2, ..., xN>
  // удовлеторява ребуса, т.е. дали MP(X,O) = K
}
```

Така при всяко следващо обръщение към `nLoops` в тялото на цикъла ще е достъпен следващият по ред n-мерен вектор, съдържащ се в масива x.

4.3. Една изледователска задача

Нека с P1[n] да означим задачата P1 за ребус от ред $n \geq 2$. Напишете съответни програми по идеята на алгоритъм *C* за решаване на задачите за $n = 2$, 3 и 4 и ги изпълнете с входни данни от табл. 2. Нека T(2), T(3), и T(4) са времената за изпълнение на съответните програми.

След като направите съответните експерименти за $n = 2$, 3, и 4, формулирайте *Вашата хипотеза*, отнасяща се до порядъка на времето T(5), необходимо за изпълнение на програмата, решаваща задача P1[5].

4.4. Две развлекателни задачи

Започнахме статията с ребус от ред 3 и едно негово решение. Да я завършим с предложение за намиране на едно решение на ребусите от ред 3 и 4.

Бележка: С програма, написана по алгоритъм *C*, всички решения на ребуса от фиг. 5 са пресметнати за 536.1 секунди.

	-		x		=	42
-		x		+		
	%		x		=	4
x		-		%		
	x		/		=	5
=		=		=		
9		3		1		

Фиг. 5. Ребус от ред 3

	+		x		/		=	4
+		-		x		+		
	-		+		/		=	1
+		+		x		+		
	x		+		/		=	3
+		x		/		-		
	/		+		x		=	26
=		=		=		=		
29		18		5		22		

Фиг. 6. Ребус от ред 4

ЛИТЕРАТУРА

1. Азълов, П. (1985). *ФОРТРАН в примери и задачи*. (*Математика – извънкласна работа*). София: Народна просвета.
2. Азълов, П., Златарова, Ф. & Тодорова, М. (2003). *ИНФОРМАТИКА*. *Профилирана подготовка*. София: Просвета.

БЕЛЕЖКИ

1. Standard C++ Library Overview: http://msdn.microsoft.com/en-us/library/4e2ess30.aspx (сайтът е посетен за последен път на 27 ноември 2012)
2. Java™ Platform, Standard Edition 7: http://docs.oracle.com/javase/7/docs/api/ (сайтът е посетен за последен път на 27 ноември 2012)

COMPUTING PROBLEMS ON MATHEMATICAL PUZZLES OR… AN EMPIRICAL APPROACH FOR EVALUATING THE PROGRAM EXECUTION TIME

Abstract. The main focus of this paper is the calculation of the program execution time. By using math puzzles as examples, it is shown how the program execution time depends primarily on the algorithm. We present three algorithms for solving the same math puzzle. For the programs using these algorithms, the execution time is displayed in both tabular and graphical format. A set of open questions and problems is included in the last section.

Pavel Azalov

✉ Associate Prof. of Computer Science, Ph.D.
Pennsylvania State University, USA
E-mail: pka10@psu.edu