Algoritmos genéticos para resolução de problemas de criptoaritmética.

Disciplina de Inteligência Computacional - GBC073

Gustavo Rezende Silva - 11311EMT018
Professora Dra. Gina Maira Barbosa de Oliveira

Universidade Federal de Uberlândia
17 de maio de 2017

Sumário

1 Objetivo

Este trabalho tem como objetivo a implementação de um algoritimo genético (AG) para resolução de problemas de criptoaritimética. E ainda, possibilitar que o aluno entenda a através da prática o efeito dos diversos parâmetros e diferentes avaliações nos problemas envolvendo AGs.

2 Desenvolvimento

2.1 Etapa 1

Nesta fase o AG foi configurado com algumas carecterísticas fixas, e outras que foram alteradas, totalizando 8 diferentes configurações, e seus resultados comparados. Estas se encontram a seguir:

Fixas:

- População de 100 indivíduos;
- Cross Over de 80% e cíclico;
- 200 gerações;
- Mutação deterministica.

Variáveis:

- Sorteio: Tour 3 e Roleta;
- Método de Substituição: Melhores entre pais e filhos (MPF) e Elitismo;
- Taxa de mutação: 10% e 20%.

Para cada configuração o programa foi executado 1000 vezes e sua taxa de convergência e tempo de execução foram anotados. Os resultados encontrados se encontram na Tabela 1.

Observando os resultados das configurações propostas a que se mostrou mais eficiente foi a numero oito (Tabela 1). Uma vez que a taxa de convergência é a ma mais alta e seu tempo de execução é relativamente baixo comparado com os demais.

N	Sorteio	Sub	Mutação	Convergência	Tempo
1	Tour 3	MPF	10%	43,6%	45,58s
2	Tour 3	ELIT	10%	46,2%	5,13s
3	Tour 3	MPF	20%	43,5%	50,39s
4	Tour 3	ELIT	20%	54,1%	5,8s
5	Roleta	MPF	10%	40,2%	46,47s
6	Roleta	ELIT	10%	69,8%	9,12s
7	Roleta	MPF	20%	42,4%	47,15s
8	Roleta	ELIT	20%	82,6%	10,33s

Tabela 1: Resultados da primeira etapa

2.2 Etapa 2

A segunda etapa teve como objetivo alterar os parâmetros, fixos ou variáveis, da primeira com o intuito de encontrar um resultado melhor. Neste trabalho a mutação deterministica foi trocada pela de probabilidade real.

O programa foi executado 1000 vezes para cada nova configuração e seus resultados se encontram na Tabela 2.

N	População	Gerações	Sorteio	Sub	Mutação	Convergência	Tempo
1	100	200	Roleta	ELIT	20%	90,1%	10,57s
2	100	200	Roleta	ELIT	25%	94,5%	10,86s
3	100	200	Roleta	ELIT	50%	98,3%	12,29s
4	200	200	Roleta	ELIT	50%	100,0%	51,92s
5	100	500	Roleta	ELIT	20%	97,0%	26,09s
6	150	250	Roleta	ELIT	20%	98,2%	28,19s
7	100	250	Roleta	ELIT	25%	95,1%	13,7s
8	100	200	TOUR 3	ELIT	50%	86,5%	8,23s
9	200	200	TOUR 3	ELIT	50%	94,6%	32.16s
10	200	200	TOUR 5	MPF	20%	85,5%	172,87s

Tabela 2: Resultados da segunda etapa

Ao analisar os resultados apresentados na Tabela 2 percebemos que a configuração de número 4 obteve 100% de taxa de convergência, porém, seu tempo de execução é 6,3 vezes o tempo do arranjo 8 que obteve 86,5%, ou seja, pouco aumento de convergência para muito tempo de execução.

Comparando o resultado 8 com os demais percebe-se que a disposição 2 é apenas 1,3 vezes mais demorada e sua taxa de convergência é 8% maior, apresentando assim uma boa relação entre tempo de execução e convergência, por isto foi a configuração escolhida.

2.3 Etapa 3

Nesta fase a melhor configuração da Etapa 2, número 2 da Tabela 2, foi utilizada para resolver novos problemas, sendo eles:

1. SEND + MORE = MONEY;
2. EAT + THAT = APPLE;
3. CROSS + ROADS = DANGER;
4. COCA + COLA = OASIS;
5. DONALD + GERALD = ROBERT.

Os resultados encontrados não foram satisfatórios e podem ser encontrados na Tabela 3.

Problema	Convergência	Tempo
2-EAT	26,2%	10,53s
3-CROSS	1,0%	10,56s
4-COCA	9,5%	10,34s
5-DONALD	5,1%	10,31s

Tabela 3: Resultados da terceira etapa

Com o intuito de melhorar os resultados testou-se as funções de avaliação a seguir:

$$Av = (\prod_{i=1}^{N} 1 + \mid R_i - E_i \mid) - 1 \tag{1}$$

$$Av = \sum_{i=1}^{N} \mid R_i - E_i \mid \tag{2}$$

Onde:

Av = Avaliação

R_i = i-esima letra do resultado da soma

E_i = i-esima letra do resultado esperado

Os resultados encontrados utilizando estas avaliações com combinações diferentes dos parâmetros estão listados nas Tabelas 4 até 8.

N	Av	Pop	Gerações	Sorteio	Sub	Mut.	Conver.	Tempo
1	1	100	200	Roleta	ELIT	50%	72,8%	12,42s
2	1	100	200	Roleta	ELIT	90%	87,2%	12,74s
3	1	100	200	Tour 1	ELIT	50%	71,0%	9,76s
4	1	100	200	Tour 1	ELIT	90%	86,1%	10,44s
5	1	150	150	Tour 1	ELIT	100%	91,8%	13,8s
6	1	300	30	Tour 2	ELIT	90%	96,0%	10,38s
7	2	100	200	Roleta	ELIT	50%	73,0%	9,83s
8	2	200	100	Roleta	ELIT	50%	80,1%	15,36s
9	2	200	100	Tour 1	ELIT	50%	82,6%	11,62s
10	2	200	90	Tour 1	ELIT	90%	91,6%	10,6s

Tabela 4: Novas avaliações para SEND + MORE = MONEY

Para o problema SEND + MORE = MONEY o melhor resultado foi o número 6 de acordo com a Tabela 4. Isto se deve ao fato de possuir a melhor taxa de convergência e tempo de execução inferior aos que possuem convergência semelhante.

N	Av	Pop	Gerações	Sorteio	Sub	Mut.	Conver.	Tempo
1	1	100	200	Roleta	ELIT	25%	19,2%	11,04s
2	1	100	200	Roleta	ELIT	50%	21,3%	13,84s
3	1	100	200	Roleta	ELIT	90%	28,6%	12,65s
4	1	100	200	Tour 1	ELIT	50%	23,01%	9,62s
5	1	100	200	Tour 1	ELIT	90%	30,1%	10,11s
6	1	150	130	Tour 1	ELIT	90%	43,6%	14,16s
7	1	300	30	Tour 2	ELIT	90%	33,1%	10,21s
8	2	200	90	Roleta	ELIT	50%	24,8%	14,55s
9	2	200	90	Tour 1	ELIT	90%	30,8%	10,75s
10	2	150	150	Tour 1	ELIT	90%	29,9%	11,79s

Tabela 5: Novas avaliações para EAT + THAT = APPLE

No problema EAT + THAT = APPLE o resultado com maior taxa de convergência foi o sexto de acordo com a Tabela 5, porém, seu tempo de execução foi elevado em comparação aos demais. Entretanto, existe uma difrença de 10% de convergência para o segundo maior o que justifica a escolha do número 6.

N	Av	Pop	Gerações	Sorteio	Sub	Mut.	Conver.	Tempo
1	1	100	200	Roleta	ELIT	50%	4,0%	14,10s
2	1	100	200	Tour 2	ELIT	50%	5,2%	10,31s
3	1	240	55	Tour 2	ELIT	90%	11,7%	13,95s
4	1	240	55	Tour 3	ELIT	90%	10,3%	12,86s
5	1	300	30	Tour 2	ELIT	90%	12,1%	11,49s
6	2	100	200	Roleta	ELIT	25%	2,4%	9,57s
7	2	100	200	Roleta	ELIT	50%	5,4%	9,88s
8	2	150	150	Roleta	ELIT	90%	6,8%	15,15s
9	2	100	200	Tour 1	ELIT	25%	3,5%	7,66s
10	2	100	200	Tour 1	ELIT	25%	5,4%	7,91s

Tabela 6: Novas avaliações para CROSS + ROADS = DANGER

Na criptoaritmética de CROSS + ROADS = DANGER o resultado melhor avaliado foi o número 5, Tabela 6, uma vez que possui a melhor taxa de convergência e tempo de execução próximo do original.

N	Av	Pop	Gerações	Sorteio	Sub	Mut.	Conver.	Tempo
1	1	100	200	Tour 1	ELIT	50%	45,7%	11,19s
2	1	255	60	Tour 2	ELIT	90%	77,5%	14,32s
3	1	300	30	Tour 2	ELIT	90%	71,0%	9,88s
4	2	100	200	Roleta	ELIT	50%	44,3%	10,06s
5	2	100	200	Roleta	ELIT	90%	57,4%	10,32s
6	2	120	120	Roleta	ELIT	90%	60,4%	8,15s
7	2	100	200	Tour 1	ELIT	50%	45,1%	7,79s
8	2	100	200	Tour 1	ELIT	90%	58,4%	8,42s
9	2	200	100	Tour 1	ELIT	90%	67,2%	12,06s
10	2	200	100	Tour 2	ELIT	90%	73,0%	11,44s

Tabela 7: Novas avaliações para COCA + COLA = OASIS

No problema COCA + COLA = OASIS obteve-se como melhor resultado o número 3, Tabela 7. Esta resposta teve a segunda maior taxa de convergência, mas como seu tempo de execução foi consideravelmente menor para uma diferença de apenas 6,5% na taxa ela foi escolhida.

N	Av	Pop	Gerações	Sorteio	Sub	Mut.	Conver.	Tempo
1	1	100	200	Roleta	ELIT	70%	45,7%	14,42s
2	1	100	200	Roleta	ELIT	90%	53,5%	14,95s
3	1	100	200	Tour 1	ELIT	50%	38,0%	11,61s
4	1	240	55	Tour 2	ELIT	90%	76,2%	13,37s
5	1	300	30	Tour 2	ELIT	90%	71,5%	12,03s
6	2	100	200	Roleta	ELIT	90%	53,9%	10,22s
7	2	100	200	Tour 1	ELIT	50%	36,1%	8,04s
8	2	100	200	Tour 1	ELIT	90%	49,0%	8,57s
9	2	200	100	Tour 1	ELIT	90%	65,0%	12,42s
10	2	200	100	Tour 3	ELIT	90%	65,5%	12,36s

Tabela 8: Novas avaliações para DONALD + GERALD = ROBERT

Na pergunta DONALD + GERALD = ROBERT a resposta selecionada foi a número 4, Tabela 8, uma vez que seu indice de convergência foi o maior e o tempo de execução foi próximo dos restantes.

Pode-se perceber que para os diferentes problemas os melhores resultados foram dados pela avaliação 1 utilizando o método de sorteio Tour, com diferentes tamanhos, substituição Elitista e mutação de 90%, apresentando variações apenas nos outros parâmetros.

E ainda, ao observar a Tabela 9 fica evidente que os resultados foram melhores do que a avaliação original.

Problema	Convergência	Tempo
1-SEND	96,0%	10,38s
2-EAT	43,6%	14,16s
3-CROSS	12,1%	11,49s
4-COCA	71,0%	9,88s
5-DONALD	76,2%	13,37s

Tabela 9: Resultados da terceira etapa novas avaliações

3 Referências Bibliográficas

- Roteiro do trabalho 1 de criptoaritimética. Profa Dra. Gina Maira Barbosa de Oliveira, 2017-1;

- Apresentações de computação evolutiva. Profa Dra. Gina Maira Barbosa de Oliveira, 2017-1;

- Solving Cryptarithmetic Problems Using Parallel Genetic Algorithm. Reza Abbasian and Masoud Mazloom, 2009.

4 Apêndice

Avaliação origina:

```
#include "stdio.h"
#include "stdlib.h"
#include "time.h"
#include <string.h>

enum selection_t { MPF, ELIT };
enum cases_t { SEND, EAT, CROSS, COCA, DONALD };

void generatePop(int pop[][11], int pop_size);
void getMinMaxAv(int pop[][11], int pop_size);
void sortRandom(int array[]);
void sortOrderAv(int array[][11], unsigned int array_size);
void getAv(int array[]);
void tour(int parents[][11],int p_size, int p_index[], int
    tour_size, int n_sons);
void crossOverAll(int parents[][11], int p_index[], int sons
    [][11], int n_sons,
                  int mutate_percent);
void crossOver(int parent1[], int parent2[], int son1[], int
    son2[],
               int mutate_percent);
void mutation(int array[]);
void updatePop(int parents[][11], int p_size, int sons[][11],
    int n_sons,
               int type);
void setRoulette(int parents[][11], int p_size, double roulette
    []);
void spinRoulette(double roulette[], int p_size, int p_index[],
    int n_sons);

int av_type = CROSS;
int av_min = 1000000000, av_max = 0;
long int roulette_inv = 100000;

int main() {
  // sendmory
  srand((unsigned)666);

  // PARAMETROS DO AG
  static int pop_size = 100;
  double cros_over = 0.8;
```

```
  int n_sons = pop_size*cros_over;
37  static int n_ger = 200;
  static int mutate_percent = 25;
39
  // PAIS E FILHOS
41  int p[pop_size][11];
  int p_index[n_sons];
43  int sons[n_sons][11];

45  // Numero de execucoes
  int n_execucao = 1000;
47  int n_zero = 0;

49  double roulette[pop_size];

51  for (int ag_n = 0; ag_n < n_execucao; ag_n++) {
    av_min = 10000000, av_max = 0;
53
    generatePop(p, pop_size);
55    getMinMaxAv(p, pop_size);

57    for (size_t i = 0; i < n_ger; i++) {
      // tour(p, p_index, 10, n_sons);
59      setRoulette(p, pop_size, roulette);
      spinRoulette(roulette, pop_size, p_index, n_sons);
61      crossOverAll(p, p_index, sons, n_sons, mutate_percent);
      updatePop(p, pop_size, sons, n_sons, ELIT);
63      getMinMaxAv(p, pop_size);
    }
65    printf("\n Melhor avalicao na execucao %i foi: %i", ag_n,
    av_min);

67    if (av_min == 0) {
      n_zero++;
69    }
  }
71
  printf("\nPorcentagem de sucesso %lf\n", (double)n_zero /
    n_execucao);
73}

75void generatePop(int pop[][11], int pop_size) {
  int n[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
77  for (int i = 0; i < pop_size; i++) {
    sortRandom(n);
79    memcpy(pop[i], n, sizeof(int) * 10);
    getAv(pop[i]);
81  }
}
```

```
83
void getMinMaxAv(int pop[][11], int pop_size) {
85  for (int i = 0; i < pop_size; i++) {
      if (pop[i][10] < av_min) {
87      av_min = pop[i][10];
      }
89
      if (pop[i][10] > av_max) {
91      av_max = pop[i][10];
      }
93  }
}
95
void sortRandom(int array[]) {
97  for (int i = 9; i > 0; --i) {
      // index
99    int w = rand() % i;
      // swap
101   int t = array[i];
      array[i] = array[w];
103   array[w] = t;
    }
105 }

107 void sortOrderAv(int array[][11], unsigned int array_size) {
    int a[11];
109 for (int i = 0; i < array_size; ++i) {
      for (int j = i + 1; j < array_size; ++j) {
111     if (array[i][10] > array[j][10]) {
          memcpy(a, array[i], sizeof(int) * 11);
113       memcpy(array[i], array[j], sizeof(int) * 11);
          memcpy(array[j], a, sizeof(int) * 11);
115     }
      }
117 }
}
119
void getAv(int array[]) {
121 int aux;
    switch (av_type) {
123 case SEND:
      aux = (array[4] * 10000 + array[5] * 1000 + array[2] * 100 +
       array[1] * 10 +
125           array[7]) -
          (array[0] * 1000 + array[1] * 100 + array[2] * 10 +
     array[3]) -
127       (array[4] * 1000 + array[5] * 100 + array[6] * 10 +
     array[1]);
```

```
129       roulette_inv = 100000;
          break;
131     case EAT:
          aux = (array[1] * 10000 + array[4] * 1000 + array[4] * 100 +
          array[5] * 10 +
133               array[0] * 1) -
                (array[0] * 100 + array[1] * 10 + array[2] * 1) -
135             (array[2] * 1000 + array[3] * 100 + array[1] * 10 +
        array[2] * 1);
          roulette_inv = 100000;
137       break;
      case CROSS:
139       aux = (array[5] * 100000 + array[4] * 10000 + array[6] *
        1000 +
                array[7] * 100 + array[8] * 10 + array[1] * 1) -
141             (array[0] * 10000 + array[1] * 1000 + array[2] * 100 +
          array[3] * 10 +
                array[3] * 1) -
143             (array[1] * 10000 + array[2] * 1000 + array[4] * 100 +
          array[5] * 10 +
                array[3] * 1);
145       roulette_inv = 1000000;
          break;
147     case COCA:
          aux = (array[1] * 10000 + array[2] * 1000 + array[4] * 100 +
          array[5] * 10 +
149               array[4] * 1) -
                (array[0] * 1000 + array[1] * 100 + array[0] * 10 +
        array[2] * 1) -
151             (array[0] * 1000 + array[1] * 100 + array[3] * 10 +
        array[2] * 1);
          roulette_inv = 100000;
153       break;
      case DONALD:
155       aux = (array[7] * 100000 + array[1] * 10000 + array[8] *
        1000 +
                array[6] * 100 + array[7] * 10 + array[9] * 1) -
157             (array[0] * 100000 + array[1] * 10000 + array[2] *
        1000 +
                array[3] * 100 + array[4] * 10 + array[0] * 1) -
159             (array[5] * 100000 + array[6] * 10000 + array[7] *
        1000 +
                array[3] * 100 + array[4] * 10 + array[0] * 1);
161       roulette_inv = 1000000;
          break;
163   }
    array[10] = abs(aux);
165 }
```

```
167 void tour(int parents[][11], int p_size, int p_index[], int
      tour_size, int n_sons) {
169   for (int i = 0; i < n_sons; i++) {
        int best_av = 1000000000;
171     int best_index = -1;
173     for (int n = 0; n < tour_size; n++) {
          int aux_i = rand() % p_size;
175       int aux_av = parents[aux_i][10];
177       if (aux_av < best_av) {
            best_av = aux_av;
179         best_index = aux_i;
          }
181
          p_index[i] = best_index;
183     }
    }
185 }
187 void crossOverAll(int parents[][11], int p_index[], int sons
      [][11], int n_sons,
                      int mutate_percent) {
189   for (size_t i = 0; i < n_sons; i += 2) {
        crossOver(parents[p_index[i]], parents[p_index[i + 1]], sons
      [i],
191               sons[i + 1], mutate_percent);
    }
193
    // int mutation_size = n_sons*((double)mutate_percent/100);
195 // for (int m = 0; m < mutation_size; m++) {
    //   mutation(sons[rand()%80]);
197 // }
}
199
void crossOver(int parent1[], int parent2[], int son1[], int
      son2[],
201                int mutate_percent) {
    unsigned int index = rand() % 10;
203 unsigned int cycle[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    while (!cycle[index]) {
205   cycle[index] = 1;
      for (size_t i = 0; i < 10; i++) {
207     if (parent2[index] == parent1[i]) {
          index = i;
209       break;
        }
211   }
```

```
  }

  for (size_t j = 0; j < 10; j++) {
    if (cycle[j] == 1) {
      son1[j] = parent1[j];
      son2[j] = parent2[j];
    } else {
      son1[j] = parent2[j];
      son2[j] = parent1[j];
    }
  }
  if ((rand() % 101) < mutate_percent) {
    mutation(son1);
  }
  if ((rand() % 101) < mutate_percent) {
    mutation(son2);
  }

  getAv(son1);
  getAv(son2);
}

void mutation(int array[]) {
  int index = rand() % 10;
  int index2 = rand() % 10;
  int temp = array[index];
  array[index] = array[index2];
  array[index2] = temp;

}

void updatePop(int parents[][11], int p_size, int sons[][11],
    int n_sons,
                int type) {
  int parents_sons[p_size + n_sons][11];
  switch (type) {
  case MPF:
    memcpy(parents_sons[0], parents, sizeof(int) * p_size * 11);
    memcpy(parents_sons[p_size], sons, sizeof(int) * n_sons *
    11);

    sortOrderAv(parents_sons, p_size + n_sons);

    memcpy(parents, parents_sons, p_size);
    break;
  case ELIT:
    sortOrderAv(parents, p_size);
    memcpy(parents[p_size - n_sons], sons, sizeof(int) * n_sons
    * 11);
```

```
    break;
  default:
    break;
  }
}

void setRoulette(int parents[][11], int p_size, double roulette
    []) {
  long int av_total = 0;
  for (int i = 0; i < p_size; i++) {
    av_total += (roulette_inv - parents[i][10]);
  }
  long int mult_factor = 100000;
  roulette[0] =
      mult_factor * ((double)(roulette_inv - parents[0][10]) /
   av_total);
  // printf("\n%lf",roulette[0]);
  for (int j = 1; j < p_size; j++) {
    roulette[j] =
        (roulette[j - 1] +
         mult_factor * (roulette_inv - parents[j][10]) / (double
   )av_total);
    // printf("\n%lf",roulette[j]);
  }
}

void spinRoulette(double roulette[], int p_size, int p_index[],
    int n_sons) {
  for (int i = 0; i < n_sons; i++) {
    double chance = (rand() % 100001);
    for (int r_index = 0; r_index < p_size; r_index++) {
      if (chance <= roulette[r_index]) {
        p_index[i] = r_index;
        break;
      }
    }
    // printf("\n%i",p_index[i]);
  }
}
```

ag.c

Avaliação 1:

```
void getAv(int array[]) {
  unsigned int aux, sum;
  switch (av_type) {
  case SEND:

    sum = (array[0] * 1000 + array[1] * 100 + array[2] * 10 +
```

```
    array[3]) +
           (array[4] * 1000 + array[5] * 100 + array[6] * 10 +
    array[1]);

    aux = (1 + abs(array[7] - sum % 10)) *
           (1 + abs(array[1] - (sum / 10) % 10)) *
           (1 + abs(array[2] - (sum / 100) % 10)) *
           (1 + abs(array[5] - (sum / 1000) % 10)) *
           (1 + abs(array[4] - (sum / 10000) % 10));

    aux = aux - 1;
    roulette_inv = pow(10, 5) + 1;

    break;
  case EAT:
    sum = (array[0] * 100 + array[1] * 10 + array[2] * 1) +
           (array[2] * 1000 + array[3] * 100 + array[1] * 10 +
    array[2] * 1);

    aux = (1 + abs(array[0] - sum % 10)) *
           (1 + abs(array[5] - (sum / 10) % 10)) *
           (1 + abs(array[4] - (sum / 100) % 10)) *
           (1 + abs(array[4] - (sum / 1000) % 10)) *
           (1 + abs(array[1] - (sum / 10000) % 10));
    aux = aux - 1;
    roulette_inv = pow(10, 5) + 1;
    break;
  case CROSS:
    sum = (array[0] * 10000 + array[1] * 1000 + array[2] * 100 +
     array[3] * 10 +
            array[3] * 1) +
           (array[1] * 10000 + array[2] * 1000 + array[4] * 100 +
     array[5] * 10 +
            array[3] * 1);

    aux = (1 + abs(array[1] - sum % 10)) *
           (1 + abs(array[8] - (sum / 10) % 10)) *
           (1 + abs(array[7] - (sum / 100) % 10)) *
           (1 + abs(array[6] - (sum / 1000) % 10)) *
           (1 + abs(array[4] - (sum / 10000) % 10)) *
           (1 + abs(array[5] - (sum / 100000) % 10));
    aux = aux - 1;
    roulette_inv = pow(10, 6) + 1;
    break;
  case COCA:
    sum = (array[0] * 1000 + array[1] * 100 + array[0] * 10 +
    array[2] * 1) +
           (array[0] * 1000 + array[1] * 100 + array[3] * 10 +
    array[2] * 1);
```

```
50      aux = (1 + abs(array[4] - sum % 10)) *
              (1 + abs(array[5] - (sum / 10) % 10)) *
52            (1 + abs(array[4] - (sum / 100) % 10)) *
              (1 + abs(array[2] - (sum / 1000) % 10)) *
54            (1 + abs(array[1] - (sum / 10000) % 10));
      aux = aux - 1;
56    roulette_inv = pow(10, 5) + 1;
      break;
58  case DONALD:
      sum = (array[0] * 100000 + array[1] * 10000 + array[2] *
     1000 +
60           array[3] * 100 + array[4] * 10 + array[0] * 1) +
            (array[5] * 100000 + array[6] * 10000 + array[7] *
     1000 +
62           array[3] * 100 + array[4] * 10 + array[0] * 1);

64    aux = (1 + abs(array[9] - sum % 10)) *
            (1 + abs(array[7] - (sum / 10) % 10)) *
66          (1 + abs(array[6] - (sum / 100) % 10)) *
            (1 + abs(array[8] - (sum / 1000) % 10)) *
68          (1 + abs(array[1] - (sum / 10000) % 10)) *
            (1 + abs(array[7] - (sum / 100000) % 10));
70    aux = aux - 1;
      roulette_inv = pow(10, 6) + 1;
72    break;
  }
74  array[10] = aux;
}
```

ag_mult.c

Avaliação 2:

```
1 void getAv(int array[]) {
    unsigned int aux, sum;
3   unsigned int mult = 1;
    switch (av_type) {
5   case SEND:

7     sum = (array[0] * 1000 + array[1] * 100 + array[2] * 10 +
     array[3]) +
            (array[4] * 1000 + array[5] * 100 + array[6] * 10 +
     array[1]);
9
      aux = (abs((array[7] - sum % 10) * mult)) +
11          (abs((array[1] - (sum / 10) % 10) * mult)) +
            (abs((array[2] - (sum / 100) % 10) * mult)) +
13          (abs((array[5] - (sum / 1000) % 10) * mult)) +
            (abs((array[4] - (sum / 10000) % 10) * mult));
```

```
15
      roulette_inv = 9*5*mult +1;
17
      break;
19  case EAT:
      sum = (array[0] * 100 + array[1] * 10 + array[2] * 1) +
21          (array[2] * 1000 + array[3] * 100 + array[1] * 10 +
     array[2] * 1);
23    aux = (abs((array[0] - sum % 10) * mult)) +
          (abs((array[5] - (sum / 10) % 10) * mult)) +
25        (abs((array[4] - (sum / 100) % 10) * mult)) +
          (abs((array[4] - (sum / 1000) % 10) * mult)) +
27        (abs((array[1] - (sum / 10000) % 10) * mult));
      roulette_inv = 9*5 +1;
29  break;
    case CROSS:
31    sum = (array[0] * 10000 + array[1] * 1000 + array[2] * 100 +
      array[3] * 10 +
            array[3] * 1) +
33        (array[1] * 10000 + array[2] * 1000 + array[4] * 100 +
      array[5] * 10 +
            array[3] * 1);
35
      aux = (abs((array[1] - sum % 10) * mult)) +
37        (abs((array[8] - (sum / 10) % 10) * mult)) +
          (abs((array[7] - (sum / 100) % 10) * mult)) +
39        (abs((array[6] - (sum / 1000) % 10) * mult)) +
          (abs((array[4] - (sum / 10000) % 10) * mult)) +
41        (abs((array[5] - (sum / 100000) % 10) * mult));
      roulette_inv = 9*6 + 1;
43    break;
    case COCA:
45    sum = (array[0] * 1000 + array[1] * 100 + array[0] * 10 +
     array[2] * 1) +
          (array[0] * 1000 + array[1] * 100 + array[3] * 10 +
     array[2] * 1);
47
      aux = (abs((array[4] - sum % 10) * mult)) +
49        (abs((array[5] - (sum / 10) % 10) * mult)) +
          (abs((array[4] - (sum / 100) % 10) * mult)) +
51        (abs((array[2] - (sum / 1000) % 10) * mult)) +
          (abs((array[1] - (sum / 10000) % 10) * mult));
53    roulette_inv = 9*5 + 1;
      break;
55  case DONALD:
      sum = (array[0] * 100000 + array[1] * 10000 + array[2] *
     1000 +
57          array[3] * 100 + array[4] * 10 + array[0] * 1) +
```

```
            (array[5] * 100000 + array[6] * 10000 + array[7] *
    1000 +
59           array[3] * 100 + array[4] * 10 + array[0] * 1);

61     aux = (abs((array[9] - sum % 10) * mult)) +
            (abs((array[7] - (sum / 10) % 10) * mult)) +
63          (abs((array[6] - (sum / 100) % 10) * mult)) +
            (abs((array[8] - (sum / 1000) % 10) * mult)) +
65          (abs((array[1] - (sum / 10000) % 10) * mult)) +
            (abs((array[7] - (sum / 100000) % 10) * mult));
67     roulette_inv = 9*6 + 1;
       break;
69   }
   array[10] = aux;
71 }
```

ag_difind.c