universidade de aveiro

deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

Sets with more sums than differences

Pedro Escaleira, N.Mec. 88821

1. Índice

2. Introdução

Este problema foi um dos dois propostos pelo docente da disciplina Algoritmos e Estruturas de Dados para defesa de nota.

Ele consiste em encontrar, para um conjunto de números inteiros, A, o valor mínimo de n para o qual $|A+A|>|A-A|$, sendo que $A \cap \{n\}=\{n\}$. Para além desta questão, foram também solicitados os valores máximos e mínimos de $|A+A|$, $|A-A|$ e $|A+A|-|A-A|$ para $n=32$.

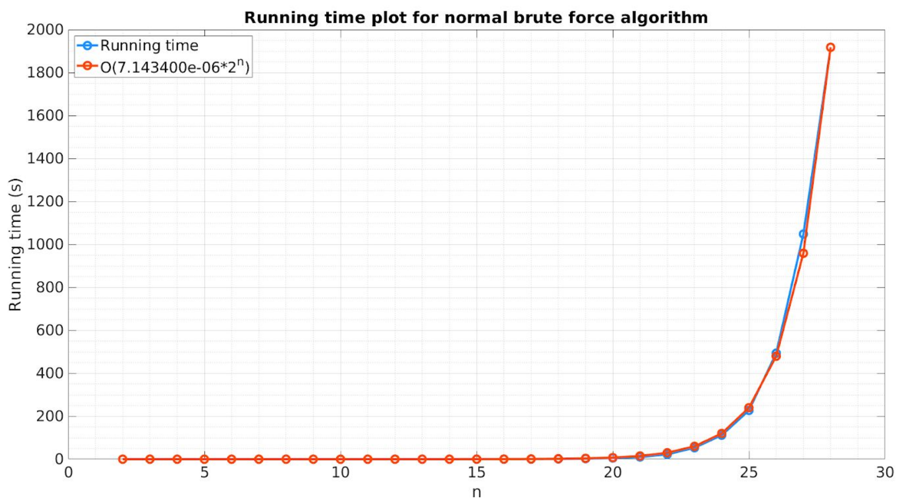
Desta forma, este relatório tem o intuito apresentar os algoritmos usados para obter os resultados requisitados e de os explicar.

3. Valor mínimo de *n* para o qual |*A*+*A*|>|*A*-*A*|

Nesta parte do trabalho, foi relativamente fácil obter uma solução para o que era pedido, usando para isso um algoritmo de força bruta algo elementar. Este baseia-se em gerar todos os conjuntos possíveis para um dado *n* (complexidade $O(2^{n-1})$) e, para cada conjunto formado, obter o conjunto das somas e das diferenças recorrendo, para isso, ao uso de dois ciclos *for* (complexidade $O(n^2)$).

Desta forma, foi possível perceber que o menor valor de *n* para o qual |*A*+*A*|>|*A*-*A*| é *n=15*. Para além deste resultado, verificou-se também que, para este valor de *n*, em 16384 conjuntos *A* diferentes que se podem formar, há 4 que respeitam esta regra.

Apesar de que, com este algoritmo rudimentar, tenha sido possível obter a resposta para este enigma, torna-se impraticável usá-lo para estudar o comportamento de *A* para valores de *n* muito elevados, como podemos observar nos seguintes gráficos:



Gráficos do tempo de execução do algoritmo e da função que para cada valor de n atribui uma aproximação do valor do tempo de execução, para $2 \leq n \leq 28$

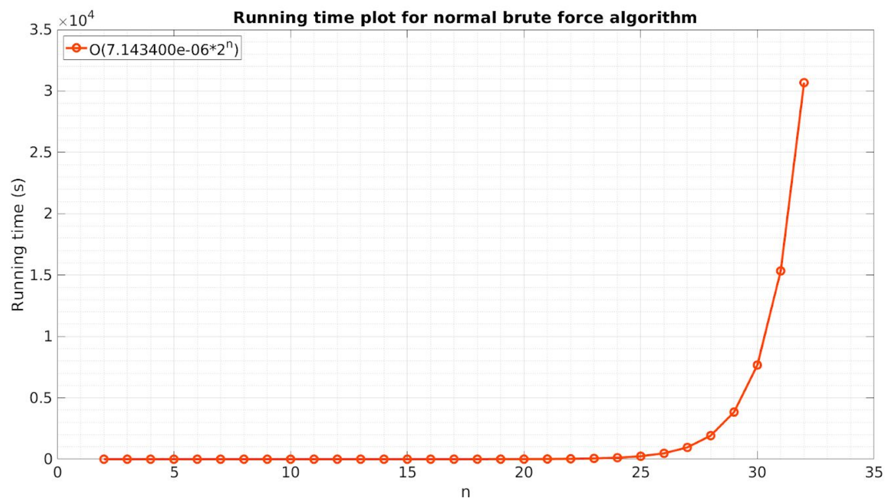


Gráfico da função que para cada valor de n atribui uma aproximação do valor do tempo de execução para $2 \leq n \leq 32$

Destes gráficos, pode concluir-se que é fulcral usar um algoritmo mais otimizado para o estudo deste problema para valores de *n* elevados.

4. Valores máximos e mínimos de |*A*+*A*|, |*A*-*A*| e |*A*+*A*|−|*A*−*A*| para *n=32*

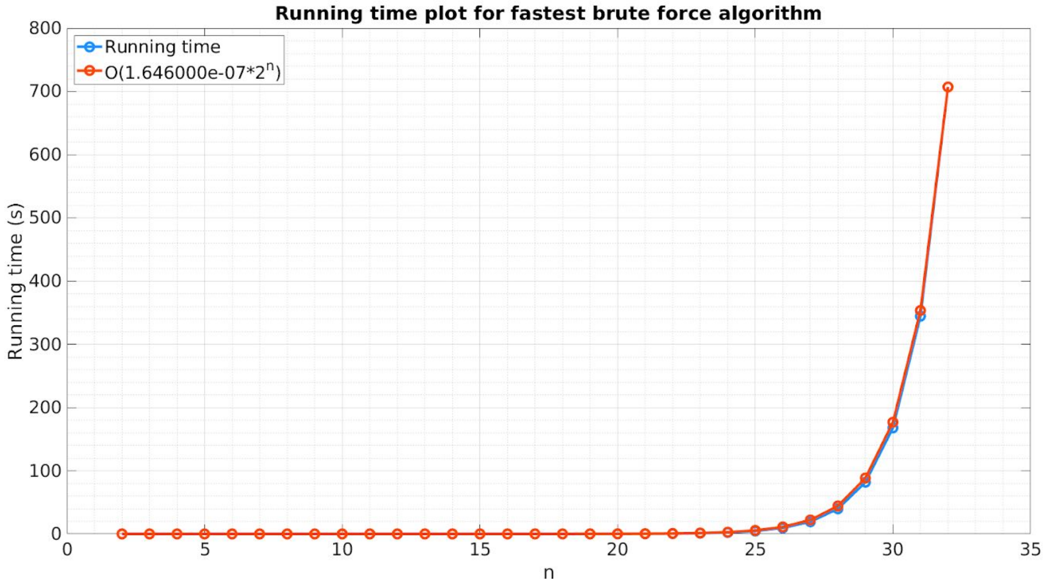
4.1. Explicação do algoritmo utilizado

Como podemos concluir nos resultados obtidos na questão atrás apresentada, para valores de *n* muito elevados, seria desagradável usar esse algoritmo, sendo que o tempo estimado para o cálculo dos valores pretendidos para *n=32* seria, sensivelmente, 11 horas (no meu computador pessoal).

Sendo assim, foi necessária uma análise exaustiva do algoritmo anteriormente utilizado para, desse modo, se encontrarem possíveis “espaços para melhorias”. A melhor delas foi reduzir a complexidade algorítmica do cálculo das somas e diferenças, sendo que se conseguiu diminuir esta de $O(n^2)$ para apenas $O(n)$. O novo algoritmo usado pode ser traduzido nos seguintes conceitos:

- As somas, diferenças e os valores do conjunto em estudo, foram armazenados em três variáveis de 64 *bits*, onde esses valores correspondem aos índices dos *bits* de valor 1.
- Para o cálculo de cada soma e diferença, são obtidos os valores presentes no conjunto *A* que nesse momento é analisado e, para cada dois valores, são realizados os cálculos:
  - ➢ $[\textit{novo conjunto das somas}] = [\textit{conjunto das somas atual}] \vee 2^{[\textit{valor } 1] + [\textit{valor } 2] - 1}$
  - ➢ $[\textit{novo conjunto das diferenças}] = [\textit{conjunto das diferenças atual}] \vee 2^{[\textit{valor } 1] - [\textit{valor } 2]}, [\textit{valor } 1] > [\textit{valor } 2]$
- Contudo, estes cálculos podem ser também obtidos da seguinte forma:
  - ➢ $[\textit{novo conjunto das somas}] = [\textit{conjunto das somas atual}] \vee ((1 << [\textit{valor } 1]) << ([\textit{valor } 2] - 1))$
  - ➢ $[\textit{novo conjunto das diferenças}] = [\textit{conjunto das diferenças atual}] \vee ((1 << [\textit{valor } 1]) >> [\textit{valor } 2]), [\textit{valor } 1] > [\textit{valor } 2]$
- Ora, sendo que na variável de 64 *bits* correspondente ao conjunto em estudo já se encontram os todos os valores $(1 << [\textit{valor } 1])$, resta apenas “resolver a segunda parte das equações” apresentadas no ponto anterior, sendo que desta forma se alcança o pretendido: um algoritmo para as somas e diferenças de complexidade $O(n)$.

Consequentemente, foi possível obter resultados com muita maior rapidez, como se pode observar no seguinte gráfico:



Gráficos do tempo de execução do algoritmo e da função que para cada valor de n atribui uma aproximação do valor do tempo de execução, para $2 \leq n \leq 32$

Ora, deste gráfico constata-se facilmente, por comparação com os da questão anterior, que o tempo necessário para obter todos os valores em estudo para *n=32* é muito menor que o tempo necessário para o primeiro algoritmo apresentado.

4.2. Análise dos resultados obtidos

Tal como solicitado pelo docente da disciplina, foram obtidos os valores máximos e mínimos de $|A+A|$, $|A-A|$ e $|A+A|-|A-A|$ para $n=32$:

- Valor máximo de $|A+A|=63$
- Valor mínimo de $|A+A|=1$
- Valor máximo de $|A-A|=63$
- Valor mínimo de $|A-A|=1$
- Valor máximo de $|A+A|-|A-A| = 4$
- Valor mínimo de $|A+A|-|A-A| = -22$

Os primeiros 4 resultados apresentam explicações matemáticas um tanto simples:

- No caso em que o conjunto *A* contém todos os números inteiros entre 1 e 32, é trivial que, das somas entre todos os valores se obtêm todos os números (inteiros) entre 2 e 64 (inclusivé). Uma demonstração simplificada pode ser traduzida da seguinte forma:
  - ➢ Qualquer número inteiro somado a si mesmo e aos números a si adjacentes resulta numa progressão aritmética de razão 1 (ex.: 1+1=2; 1+2=3; 1+3=4; 1+4=5 …). Ora, como neste caso concreto, o valor máximo que uma das parcelas da adição é 32, obtêm-se todos os números inteiros entre 2 e 33, se a primeira parcela for 1, todos os números inteiros entre 4 e 34, se a primeira parcela for 2 e, seguindo esta ordem de ideias, no seu total, obtêm-se todos os números inteiros entre 2 e 64.
  - ➢ Sendo assim, o número de somas possíveis do conjunto *A* é 63. Pensando "um pouco mais além", percebe-se que $|A+A|=2*n-1$, para qualquer $n \geq 1$.
- Para o valor de máximo de $|A-A|$ deparamo-nos também com uma justificação similar à de $|A+A|$:
  - ➢ Se o conjunto *A* possuir todos os valores entre 1 e 32, então a diferença entre 32 e todos os números de *A* resulta num novo conjunto possuindo todos os números entre 0 e 31 (trivial). Ora, sendo que $a-b=-(b-a)$, para quaisquer

números inteiros *a* e *b*, conclui-se que para esse conjunto *A*, *A-A* possuirá todos os números inteiros entre -31 e 31. Desta forma, |*A-A*|=*63* e, “seguindo a mesma ordem de ideias”, temos que |*A-A*|=*2*n-1*, para qualquer $n \geq 1$.

- Já os valores mínimos de |*A+A*| *e* |*A-A*| são obtidos quando *A={32}*, sendo que *A+A={64}* e *A-A={0}, pelo que* |*A+A*|=*1 e* |*A-A*|=*1.* Com a mesma linha de pensamentos, percebe-se também que, para qualquer $n \geq 1$, os mínimos valores de |*A+A*| *e* |*A-A*| serão sempre 1, resultado obtido quando *A={n}*.

Contudo, os valores de |*A+A*|−|*A−A*| possuem explicações matemáticas de nível mais complexo pelo que, neste relatório, não irão ser apresentadas. Contudo, podem ser analisados e serem a raiz de algumas conclusões curiosas (e um tanto triviais):

- Para *n=32* há pelo menos um conjunto *A* onde o conjunto soma possui mais elementos que o conjunto diferença, já que |*A+A*|−|*A−A*| = 4.
- Há pelo menos um conjunto *A* para o qual o conjunto soma apresenta mais 4 elementos que o conjunto diferença, sendo que não há qualquer conjunto *A* cujo conjunto soma possua mais de 4 elementos que o conjunto diferença.
- Há pelo menos um um conjunto *A* para o qual o conjunto diferença contém mais 22 elementos que o conjunto soma, sendo que não há qualquer conjunto *A* cujo conjunto diferença possua mais de 22 elementos que o conjunto soma.

4.3. Análise de outros resultados obtidos

Para além dos resultados acima descritos, foram também "capturados" outros, que podem ter alguma relevância. Estes são o número de conjuntos *A* que, para cada valor de *n*, respeitam a condição |*A*+*A*|>|*A*-*A*| e as proporções destes valores em relação ao número total de conjuntos *A* diferentes que podem ser formados para cada *n*:

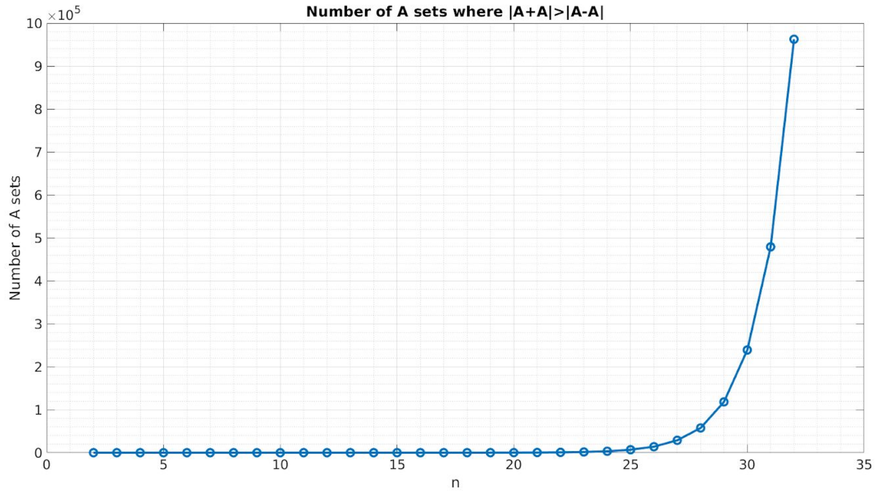


Gráfico correspondente ao número de conjuntos *A* onde |*A*+*A*|>|*A*-*A*|

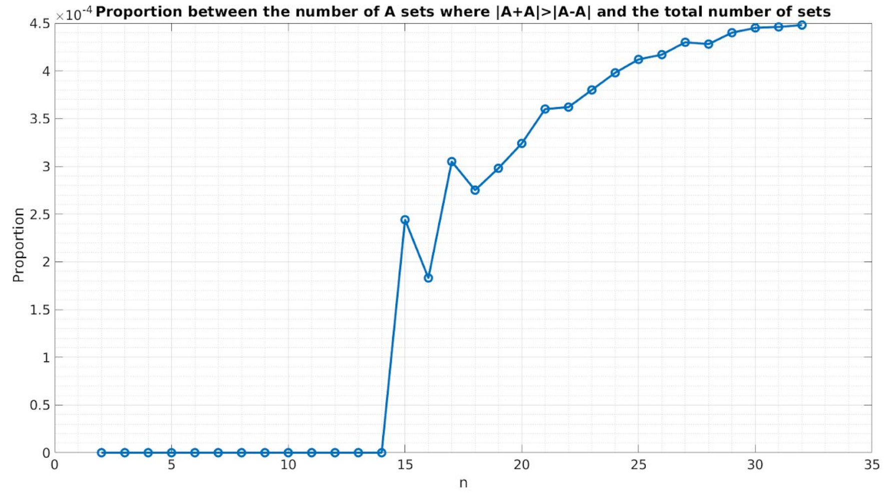
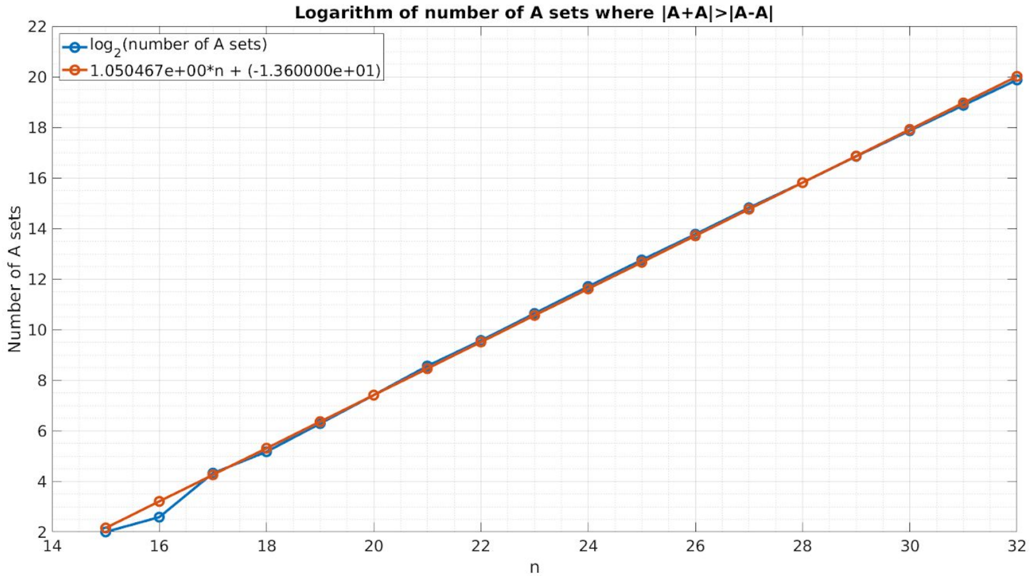


Gráfico correspondente à proporção entre o número conjuntos *A* onde |*A*+*A*|>|*A*-*A*| e o número de conjuntos *A* diferentes

Como se pode observar do primeiro gráfico, o número de conjuntos *A* onde |*A*+*A*|>|*A*-*A*| tem um crescimento exponencial em relação ao valor de *n*. Isto torna-se mais evidente quando observado o gráfico do logaritmo de base dois do número de conjuntos *A* com essa característica:



Pela linearidade dos pontos obtidos, conclui-se que o número de conjuntos *A* onde |*A*+*A*|>|*A*-*A*| para *n+1* é sensivelmente o dobro do número destes conjuntos para *n*, com $n \geq 15$.

Do segundo gráfico, pode-se constatar que, de uma forma geral, a percentagem do número de conjuntos *A* onde |*A*+*A*|>|*A*-*A*| aumenta com o aumento do valor de *n*, para $15 \leq n \leq 32$.

5. Código usado

5.1. problem1.c

Código criado com o intuito de resolver as duas perguntas apresentadas pelo docente da disciplina. Quando compilado e executado, permite não só calcular os valores apresentados neste relatório até um valor de *n* introduzido pelo utilizador, mas também usar um dos dois algoritmos acima exibidos e, caso seja do interesse do utilizador, armazenar os resultados num ficheiro *.txt*, de nome introduzido por ele.

```c
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <math.h>
#include <strings.h>
#include <string.h>
#include "elipsed_time.h"

const unsigned short MIN_N = 1,                                                          //minimum value of n
                     MAX_N = 32;                                                        //maximum value that n can have in this implementation
FILE *data_file;                                                                         //file where is stored all the data collected while this program runs, if user wants to save it


//function given by AED professor and returns the number of 1 bits of given mask (argument)
int population_count(unsigned long mask){
  int count;

  for(count = 0;mask != 0ul;count++)
     mask &= mask - 1ul;
  return count;
}

/* Function to calc the values stored in a given set (by index)
*
* Arguments:
*    -> unsigned long set: set where are the values to extract
*    -> unsgined short n: actual n (that is the max value in set)
*    -> long *set_values: address of the array where is to store the values
* Return:
*    -> number of values in set
* Example:
*    -> the set 1101 (n = 4) corresponds to values {4, 3, 1}. So, this set has 3 values
*/
unsigned int setValues(unsigned long set, unsigned short n, long *set_values){
  unsigned int index = 0;

  for(int value = 1; set != 0; value++){
     if((set & 1ul) == 1ul)
        set_values[index++] = value;

     set = set >> 1ul;
  }
```

```
  return index;
}

/* Function to calc the sum and diff sets of A (slowest version)
*
* Arguments:
*    -> unsigned long set: set where are the values to sum and subtract
*    -> unsigned short n: actual n (that is the max value in set)
*    -> unsigned long *result: where are stored the results of sum and diff sets of A (result[0] = result_sum; result[1] = result_dif)
*
* Attention:
*    -> the unsigned long *result argument must be allocated before calling this function
*/
void normalBruteForce(unsigned long set, int n, unsigned long *result){
  unsigned long result_sum = 0,
                result_dif = 0;

  long set_values[n];
  unsigned int size = setValues(set, n, set_values);

  for(int index1 = 0; index1 < size; index1++)
     for(int index2 = index1; index2 < size; index2++){
        result_sum |= (unsigned long)pow(2, set_values[index1] + set_values[index2] - 1);
        result_dif |= (unsigned long)pow(2, set_values[index2] - set_values[index1]);
     }

  result[0] = result_sum;
  result[1] = result_dif;

}

/* Function to calc the sum and diff sets of A (fastest version)
*
* Arguments:
*    -> unsigned long set: set where are the values to sum and subtract
*    -> unsigned short n: actual n (that is the max value in set)
*    -> unsigned long *result: where are stored the results of sum and diff sets of A (result[0] = result_sum; result[1] = result_dif)
*
* Attention:
*    -> the unsigned long *result argument must be allocated before calling this function
*    -> the explanation for this method is in the report attached with this document. However, due to my elementary english knowledge,
*       it is in portuguese. Nevertheless, if you feel comfortable with binary logic, you will find no problems to understand this piece of code, I think
*/
void fastestBruteForce(unsigned long set, unsigned short n, unsigned long *result){

  unsigned long result_sum = 0,
                result_dif = 0;

  unsigned long set_copy = set;
  for(int value = 1; set_copy != 0; value++){
     if((set_copy & 1ul) == 1ul){
        result_sum |= set << value;
        result_dif |= set >> (value - 1);
     }
     set_copy >>= 1ul;
  }

  result[0] = result_sum;
  result[1] = result_dif;

}

/* Function to calc, essentially, the number of MSTDs for a given n
*
```

```
* Arguments:
*    -> unsigned short n: actual n (that is the max value in set)
*    -> short save_data: boolean value (value >0 if the user wants to save the data to a file; <=0 if user doesn't wants to save the data)
*    -> unsigned short version: variable with the version that user wants to run (0 if normal brute force; 1 if fastest brute force)
* Return:
*    -> number of MSTDs for a giver n
*/
unsigned long mstd(unsigned short n, short save_data, unsigned short version){

  unsigned long A_set,                                                                          //where is stored the values of A in the next while loop (the values of A corresponds to the indexes of 1 bits of A_set plus 1)
                max_value = pow(2, n - 1);                                                      //this is the max value that A_set can reach, because it can just have n bits with the value of 1

  unsigned int number_of_mstd_sets = 0;                                                         //number of MSTD sets for given n
  unsigned short biggest_value_sum = 0,                                                         //biggest value of |A + A| for given n
                 biggest_value_dif = 0,                                                         //smallest value of |A + A| for given n
                 smallest_value_sum = __SHRT_MAX__*2 + 1,                                       //biggest value of |A - A| for given n
                 smallest_value_dif = __SHRT_MAX__*2 + 1;                                       //smallest value of |A - A| for given n
  short biggest_value_sum_and_dif = 0,                                                          //biggest value of |A + A| - |A - A| for given n
        smallest_value_sum_and_dif = __SHRT_MAX__;                                              //smallest value of |A + A| - |A - A| for given n

  unsigned short sum,                                                                           //current value of |A + A|
                 dif;                                                                           //current value of |A - A|
  short sum_and_dif;                                                                            //current value of |A + A| - |A - A|

  unsigned long result_sum,                                                                     //variable where is stored all the A + A values (each index of 1 bit corresponds to one of the sum's value)
                result_dif,                                                                     //variable where is stored all the positive (and 0) A - A values (each index of 1 bit corresponds to one of the sum's value)
                *result = malloc(2*sizeof(result_sum));                                         //variable where are stored the current values of result_sum (index 0) and result_dif (index 1) (this array is obtained through the normalBruteForce() or
fastestBruteForce() functions)


  for(int n1 = 0; n1 < max_value; n1++){
     A_set = max_value + n1;                                                                    //to cover all the possible A set values

     if(version == 0)
        normalBruteForce(A_set, n, result);
     else
        if(version == 1)
           fastestBruteForce(A_set, n, result);
     result_sum = result[0];                                                                    //update the data struct where is saved all the A set sums
     result_dif = result[1];                                                                    //update the data struct where is saved all the positive (and zero) A set differences

     sum = population_count(result_sum);
     dif = 2*population_count(result_dif) - 1;                                                  //the reason for such value is that in result_dif just are placed the positive (and zero) results of A - A. As for any two different numbers there are two
possible results for the difference, beig one mymmetrical to the other (-15 and 15 for exemple), the number of differences in the A - A set is equal to 2*(number of positive results of all differences) - 1 (-1 because "there are not two 0")
     sum_and_dif = sum - dif;

     //update the number of MSTD sets and the biggest and the smallest values of |A + A|, |A - A| and |A + A| - |A - A|
     if(sum > dif)
        number_of_mstd_sets++;
     if(sum > biggest_value_sum)
        biggest_value_sum = sum;
     if(sum < smallest_value_sum)
        smallest_value_sum = sum;
     if(dif > biggest_value_dif)
        biggest_value_dif = dif;
     if(dif < smallest_value_dif)
        smallest_value_dif = dif;
     if(sum_and_dif > biggest_value_sum_and_dif)
        biggest_value_sum_and_dif = sum_and_dif;
     if(sum_and_dif < smallest_value_sum_and_dif)
        smallest_value_sum_and_dif = sum_and_dif;

  }
```

```
  //print to terminal
  printf("Biggest value of |A + A|: %u\n"                                        //biggest_value_sum
         "Smallest value of |A + A|: %u\n"                                       //smallest_value_sum
         "Biggest value of |A - A|: %u\n"                                        //biggest_value_dif
         "Smallest value of |A - A|: %u\n"                                       //smallest_value_dif
         "Biggest value of |A + A| - |A - A|: %u\n"                              //biggest_value_sum_and_dif
         "Smallest value of |A + A| - |A - A|: %d\n",                            //smallest_value_sum_and_dif
         biggest_value_sum, smallest_value_sum, biggest_value_dif,
         smallest_value_dif, biggest_value_sum_and_dif, smallest_value_sum_and_dif);

  //print to data file
  if(save_data)
     fprintf(data_file, "%u\t"                                                   //biggest_value_sum
                        "%u\t"                                                   //smallest_value_sum
                        "%u\t"                                                   //biggest_value_dif
                        "%u\t"                                                   //smallest_value_dif
                        "%u\t"                                                   //biggest_value_sum_and_dif
                        "%d\t",                                                  //smallest_value_sum_and_dif
             biggest_value_sum, smallest_value_sum,
             biggest_value_dif, smallest_value_dif,
             biggest_value_sum_and_dif, smallest_value_sum_and_dif);

  //deallocate memory
  free(result);

  return number_of_mstd_sets;

}

int main(int argc, char **argv){

  short chosen_opt = -1,                                                         //option chosen by user -> corresponds to the version of MSTD to run
        save_data = 0;                                                           //boolean variable -> true ( > 0 ) if user wants to save data to a file; false ( <= 0 ) if user wants the opposite
  char *file_name;                                                               //file name where is to to save data
  unsigned short final_n;                                                        //max value of n to calc

  /**************** options ****************/
  int opt,                                                                       //integer where is stored the value returned by getopt_long() function
      user_option_index = 0,                                                     //integer used to know wich option of user_options[] was chosen by the user (used in getopt_long() function)
      number_user_option = 3;                                                    //number of options stored in user_options[]

  struct option user_options[] = {
         {"nb", required_argument, 0, 0},                                        //normal brute force solution
         {"fb", required_argument, 0, 0},                                        //fast brute force solution
         {"save", required_argument, 0, 0},                                      //last one must be always the save option
         {0, 0, 0, 0}
  };

  while((opt = getopt_long(argc, argv, "", user_options, &user_option_index)) != -1){
     switch(opt){
        case 0:
           if(!strcmp(user_options[user_option_index].name,                      //option --save
                      user_options[number_user_option - 1].name)){

              int name_size;                                                     //number of characters in document's name where is to be stored the data

              if((name_size = strlen(optarg)))
                 file_name = malloc(name_size + 4);
              else
                 goto error_message;                                             //this block of code is placed after this while loop, and is where is printed the message explaining how to use this program to the user

              strcpy(file_name, optarg);
              strcat(file_name, ".txt");
              save_data = 1;                                                     //save_data is now true, because the user selected the option --save
           }else
```

```
            for(int num = 0; num < number_user_option - 1; num++)                       //other options (--save not included)
               if(!strcmp(user_options[user_option_index].name,
                          user_options[num].name)){

                  if(chosen_opt < 0){                                                   //verify if user dont choose more than one option (--save option not included)

                     chosen_opt = num;

                     if((final_n = atoi(optarg)) < MIN_N ||                             //verify if the n's value given by the user does not exceed the predefined range
                         final_n > MAX_N)
                        goto error_message;                                             //this block of code is placed after this while loop, and is where is printed the message explaining how to use this program to the user

                  }else
                     goto error_message;                                                //this block of code is placed after this while loop, and is where is printed the message explaining how to use this program to the user

               }
         break;                                                                         //break of case 0
      default:
         goto error_message;                                                            //this block of code is placed after this while loop, and is where is printed the message explaining how to use this program to the user
   }
}

if(chosen_opt < 0){                                                                     //verify if user chose at least one option (--save option not included)
   error_message:
      fprintf(stderr, "ERROR on input arguments!\n"
                      "Usage: %s [--nb n <or> --fb n] (--save file_name)\n"
                      "  > %-10s normal brute force solution\n"
                      "  > %-10s fastest brute force solution\n"
                      "  > %-10s save data to a txt document\n"
                      "Argements between [] are required (obviously, just one of them),"
                      "and arguments between () are optional\n",
              argv[0], "--nb", "--fb", "--save");
      return EXIT_FAILURE;                                                              //exit with failure code
}

 /**************** calc MSTD for 1 <= n <= final_n ****************/
unsigned short n = 0;                                                                   //initialize n's value
unsigned int number_of_mstd_sets;                                                       //number of MSTD sets in "current value of n" (this value is updated in the next while loop)
double ratio_of_mstd_sets,                                                              //proportion between the number of MSTD sets in all calculated sets
       time_to_complete;                                                                //time spent on each mstd() call for the respective value of n


if(save_data)                                                                           //open file to save data and print table's first line
   data_file = fopen(file_name, "w");

//increase n's value until it reaches final_n, and with given n, compute MSTD with the algorithm's version chosen by the user
do{
   n++;

   //print actual n to terminal
   printf("N -> %d\n", n);

   //print actual n to data file
   if(save_data)
      fprintf(data_file, "%d\t", n);

   //compute MSTD
   elapsed_time();
   number_of_mstd_sets = mstd(n, save_data, chosen_opt);
   time_to_complete = elapsed_time();
   ratio_of_mstd_sets = number_of_mstd_sets/pow(2, n - 1);

   //print to terminal
   printf("Number of MSTD sets: %u\n"
          "Ratio of MSTD sets: %f\n"
```

```
              "Time: %f\n\n\n",
              number_of_mstd_sets, ratio_of_mstd_sets, time_to_complete);

       //print to data file
       if(save_data)
          fprintf(data_file, "%u\t"                                                    //number_of_mstd_sets
                             "%f\t"                                                    //ratio_of_mstd_sets
                             "%f\n",                                                   //time_to_complete
                   number_of_mstd_sets, ratio_of_mstd_sets, time_to_complete);

    }while(n < final_n);

    //close data file
    if(save_data)
       fclose(data_file);

    return EXIT_SUCCESS;

}
```

5.2. elipsed_time.h

Código cedido pelo docente da disciplina, que permite obter intervalos de tempo de execução em segundos.

```c
//
// Tomás Oliveira e Silva, AED, September 2018
//
// code to measure the elapsed time used by a program
//
// use as follows:
//
//   (void)elapsed_time();
//   // put your code to be time measured here
//   dt = elapsed_time();
//

#ifdef __linux__

//
// GNU/Linux code to measure elapsed time
//

#include <time.h>

double elapsed_time(void)
{
 static struct timespec last_time,current_time;

 last_time = current_time;
 if(clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&current_time) != 0)  // the first argument could also be CLOCK_REALTIME
   return -1.0; // clock_gettime() failed!!!
 return ((double)current_time.tv_sec - (double)last_time.tv_sec) +
        1.0e-9 * ((double)current_time.tv_nsec - (double)last_time.tv_nsec);
}

#endif


#if defined(_MSC_VER) || defined(_WIN32) || defined(_WIN64)

//
// Microsoft Windows code to measure elapsed time
//

#include <windows.h>

double elapsed_time(void)
{
 static LARGE_INTEGER frequency,last_time,current_time;
 static int first_time = 1;

 if(first_time != 0)
 {
   QueryPerformanceFrequency(&frequency);
   first_time = 0;
 }
 last_time = current_time;
```

```
 QueryPerformanceCounter(&current_time);
 return (double)(current_time.QuadPart - last_time.QuadPart) / (double)frequency.QuadPart;
}

#endif
```

5.3. make_plots.m

Código de *matlab* usado para criar os gráficos das execuções dos algoritmos apresentados.

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% How to use:
%%    > file_name -> file's name where is stored the data
%%    > version -> 1 if the data was obtained using the normal brute force algorithm;
%%                 2 if the data was obtained using the fastest brute force algorithm;
%%    > make_approximation_until_max_n -> boolean value: true if user wants just to obtain approximation function's plot
%%                                                       false if user wants to obtain the plot of data collected alongside approximation function's plot
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function make_plots(file_name, version, make_approximation_until_max_n)
  %%%%%%%%%%%%%%%%%% constants %%%%%%%%%%%%%%%%%%
  LINE_WIDTH = 3;
  DOT_WIDTH = 10;
  FIGURE_DIMENSIONS = [0 0 1920 1080];
  PLOT_TITLE_SIZE = 22;
  PLOT_LABELS_SIZE = 20;
  PLOT_VALUES_SIZE = 20;
  PLOT_LEGEND_SIZE = 20;
  PLOT_LEGEND_LOCATION = 'northwest';
  PLOT_TITLES = "Running time plot for";
  PLOT_TITLES_VERSION = ["normal brute force algorithm"
                         "fastest brute force algorithm"];
  PLOT_XLABELS = "n";
  PLOT_YLABELS = "Running time (s)";
  PLOT_COLORS = [[30, 144, 255],
                 [255, 69, 0]]/255;

  PNG_NAMES_VERSION = ["nb", "fb"];
  PNG_NAMES_APPROXIMATION = ["approximation", "real"];

  MAX_N = 32;
  ALL_M = [[10^-6: 10^-10: 10^-5],
           [10^-7: 10^-11: 10^-6]];


  %%%%%%%%%%%%%%%%%% work with data %%%%%%%%%%%%%%%%%%
  data_file = fopen(file_name, 'r');
  data = fscanf(data_file, '%f', [10, Inf]);
  fclose(data_file);

  png_file_name = sprintf('%s_%s', ...
                          PNG_NAMES_VERSION(version));

  n = data(1, :);
  n_approximation_function = n;
  time_seconds = data(10, :);
  time_seconds_estimation = -time_seconds;
  m = 0;   %approximation function 'slope'

  %approximation function for brute force
  for i = ALL_M(version, :)
     new_estimation = (2.^n)*i;
     if sum(abs(new_estimation - time_seconds)) < sum(abs(time_seconds_estimation - time_seconds))
        time_seconds_estimation = new_estimation;
```

```
        m = i;
      end
    end

    %time plot
    figure('Position', FIGURE_DIMENSIONS);
    legend_text = cell(1, 2);
    legend_text{2} = sprintf('O(%d*2^n)', m);

    if ~make_approximation_until_max_n
      plot(n, time_seconds, '-o', ...
           'LineWidth', LINE_WIDTH, ...
           'MarkerSize', DOT_WIDTH, ...
           'color', PLOT_COLORS(1, :));
      hold on;
      legend_text{1} = 'Running time';

      png_file_name = sprintf('%s_%s', ...
                              png_file_name, ...
                              PNG_NAMES_APPROXIMATION(2));
    else
      n_approximation_function = [n(1):MAX_N];
      time_seconds_estimation = (2.^n_approximation_function)*m;

      legend_text{1} = legend_text{2};
      legend_text(end) = [];

      png_file_name = sprintf('%s_%s', ...
                              png_file_name, ...
                              PNG_NAMES_APPROXIMATION(1));
    end

    plot(n_approximation_function, time_seconds_estimation, '-o', ...
         'LineWidth', LINE_WIDTH, ...
         'MarkerSize', DOT_WIDTH, ...
         'color', PLOT_COLORS(2, :));
    legend(legend_text, ...
           'FontSize', PLOT_LEGEND_SIZE, ...
           'Location', PLOT_LEGEND_LOCATION);

    title(sprintf('%s %s', PLOT_TITLES, PLOT_TITLES_VERSION(version)), ...
          'FontSize', PLOT_TITLE_SIZE);
    xlabel(PLOT_XLABELS, ...
           'FontSize', PLOT_LABELS_SIZE);
    ylabel(PLOT_YLABELS, ...
           'FontSize', PLOT_LABELS_SIZE);
    set(gca, 'FontSize', PLOT_VALUES_SIZE);

    grid on;
    grid minor;

    print(png_file_name,'-dpng','-r0'); %Generates automaticaly a png image from displayed figure
  end
```

5.4. make_plots_number_mstd.m

Código de *matlab* usado para criar os 3 gráficos apresentados no ponto **4.3**.

```
clear all;

LINE_WIDTH = 3;
DOT_WIDTH = 10;
PLOT_TITLE_SIZE = 20;
PLOT_LABELS_SIZE = 18;
PLOT_VALUES_SIZE = 18;
PLOT_LEGEND_SIZE = 18;
PLOT_LEGEND_LOCATION = 'northwest';


file_name = 'data_fb.txt';

data_file = fopen(file_name, 'r');
data = fscanf(data_file, '%f', [10, Inf]);
fclose(data_file);

n = data(1, :);

%number of sets plot
figure('Position', [0 0 1920 1080]);
plot(n, data(8, :), '-o', ...
    'LineWidth', LINE_WIDTH, ...
    'MarkerSize', DOT_WIDTH);
title('Number of A sets where |A+A|>|A-A|', ...
     'FontSize', PLOT_TITLE_SIZE);
xlabel('n', ...
      'FontSize', PLOT_LABELS_SIZE);
ylabel('Number of A sets', ...
      'FontSize', PLOT_LABELS_SIZE);
set(gca, 'FontSize', PLOT_VALUES_SIZE);
grid on;
grid minor;

%proportions plot
figure('Position', [0 0 1920 1080]);
plot(n, data(9, :), '-o', ...
    'LineWidth', LINE_WIDTH, ...
    'MarkerSize', DOT_WIDTH);
title('Proportion between the number of A sets where |A+A|>|A-A| and the total number of sets', ...
     'FontSize', PLOT_TITLE_SIZE);
xlabel('n', ...
      'FontSize', PLOT_LABELS_SIZE);
ylabel('Proportion   ', ...
      'FontSize', PLOT_LABELS_SIZE);
set(gca, 'FontSize', PLOT_VALUES_SIZE);
grid on;
grid minor;

%Logarithm
log_funct = log2(data(8, :));
```

```
log_funct = log_funct(n >= 15);
n = n(n >= 15);

m = (log_funct(n == 32) - log_funct(n == 18))/(32 - 18);
b = 0;
approximation_funct = m*n + b;

for i = [-100: 0.01: 0]
      new_estimation = m*n + i;
      if sum(abs(new_estimation - log_funct)) < sum(abs(approximation_funct - log_funct))
             approximation_funct = new_estimation;
             b = i;
      end
end

figure('Position', [0 0 1920 1080]);
plot(n, log_funct, '-o', ...
    'LineWidth', LINE_WIDTH, ...
    'MarkerSize', DOT_WIDTH);
hold on;
plot(n, approximation_funct, '-o', ...
    'LineWidth', LINE_WIDTH, ...
    'MarkerSize', DOT_WIDTH);
title('Logarithm of number of A sets where |A+A|>|A-A|', ...
     'FontSize', PLOT_TITLE_SIZE);
xlabel('n', ...
      'FontSize', PLOT_LABELS_SIZE);
ylabel('Number of A sets', ...
      'FontSize', PLOT_LABELS_SIZE);
set(gca, 'FontSize', PLOT_VALUES_SIZE);
legend({'log_2(number of A sets)', sprintf('%d*n + (%d)', m, b)}, ...
      'FontSize', PLOT_LEGEND_SIZE, ...
      'Location', PLOT_LEGEND_LOCATION);
grid on;
grid minor;
```